



LAG ERASURE FOR ZERO-LAG DISTRIBUTED SIMULATION

Title	LAG ERASURE FOR ZERO-LAG DISTRIBUTED SIMULATION
Item Type	Thesis
Authors	Peitso, Loren E.
URI	https://hdl.handle.net/10945/75027
Publisher	Monterey, CA; Naval Postgraduate School
Date Issued	2026-03
Rights	This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.
Download date	2026-10-04 12:17:56
Link to Item	https://hdl.handle.net/10945/75027

Downloaded from NPS Archive: Calhoun



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

**LAG ERASURE FOR ZERO-LAG
DISTRIBUTED SIMULATION**

by

Loren E. Peitso

March 2026

Dissertation Supervisor:

Don Brutzman

**Research for this dissertation was performed at the MOVES Institute.
Distribution Statement A. Approved for public release: Distribution is unlimited.**

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC, 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE March 2026	3. REPORT TYPE AND DATES COVERED Dissertation	
4. TITLE AND SUBTITLE LAG ERASURE FOR ZERO-LAG DISTRIBUTED SIMULATION			5. FUNDING NUMBERS	
6. AUTHOR(S) Loren E. Peitso				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Distribution Statement A. Approved for public release: Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) Communications latency generates lag within distributed simulations, creating mismatched expectations and causality violations that prevent repeatable, verifiable states and reduce confidence in simulation progression and final results. To train and build trust in both humans and complex artificial intelligence-enabled systems, it is often necessary to run simulations that exceed a single cluster site's capabilities or require specialized hardware from geographically distributed sites. Lagging simulations are incompatible with this training. Lag is documented in the literature as unavoidable, leading to weak mitigation and acceptance rather than elimination. This work shows that the assumption of unavoidable lag is implementation-dependent (not absolute), enabling the design of a simulation architecture that limits, then erases lag effects from a distributed simulation's state. Following a root-cause analysis of lag, techniques from both the analytical and virtual environments simulation communities are hybridized to define a software architecture for delivering lag-free distributed simulations. The affected scope is widespread—from command and control systems and Live Virtual Constructive training, to online multiplayer gaming and shared massively multi-user virtual/augmented reality metaverse applications. Lag-free distributed simulations are repeatable and verifiable, enabling a level of confidence in simulation results previously unreachable.				
14. SUBJECT TERMS network distributed simulation, asynchronous, discrete event, physically based modeling, live virtual constructive, command and control, augmented reality, virtual reality, virtual environment, MMOG, metaverse, latency hiding, lag			15. NUMBER OF PAGES 235	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Distribution Statement A. Approved for public release: Distribution is unlimited.

LAG ERASURE FOR ZERO-LAG DISTRIBUTED SIMULATION

Loren E. Peitso
Lieutenant Commander (Ret.), U.S. Navy
Civilian, Department of the Navy
BS, University of Minnesota, Twin Cities, 1986
MS, Modeling, Virtual Environments, and Simulation, Naval Postgraduate School, 2002

Submitted in partial fulfillment of the
requirements for the degree of

**DOCTOR OF PHILOSOPHY IN MODELING, VIRTUAL ENVIRONMENTS,
AND SIMULATION**

from the

**NAVAL POSTGRADUATE SCHOOL
March 2026**

Approved by: Don Brutzman
Department of
Information Sciences
Dissertation Supervisor

Rudolph P. Darken
Department of
Computer Science
Dissertation Chair

Arnold H. Buss
MOVES Institute

Peter J. Denning
Department of
Computer Science

James B. Michael
Department of
Computer Science

Gurminder Singh
Department of
Computer Science

Approved by: Gurminder Singh
Chair, Department of Computer Science

James B. Michael
Vice Provost of Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Communications latency generates lag within distributed simulations, creating mismatched expectations and causality violations that prevent repeatable, verifiable states and reduce confidence in simulation progression and final results. To train and build trust in both humans and complex artificial intelligence-enabled systems, it is often necessary to run simulations that exceed a single cluster site's capabilities or require specialized hardware from geographically distributed sites. Lagging simulations are incompatible with this training. Lag is documented in the literature as unavoidable, leading to weak mitigation and acceptance rather than elimination. This work shows that the assumption of unavoidable lag is implementation-dependent (not absolute), enabling the design of a simulation architecture that limits, then erases lag effects from a distributed simulation's state. Following a root-cause analysis of lag, techniques from both the analytical and virtual environments simulation communities are hybridized to define a software architecture for delivering lag-free distributed simulations. The affected scope is widespread—from command and control systems and Live Virtual Constructive training, to online multiplayer gaming and shared massively multi-user virtual/augmented reality metaverse applications. Lag-free distributed simulations are repeatable and verifiable, enabling a level of confidence in simulation results previously unreachable.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1	Introduction	1
1.1	The Problem	3
1.1.1	Lag Is Provably Unavoidable?	3
1.1.2	Why Should Anyone Care? It is Settled, and Proven!	4
1.1.3	Getting Technical: Isolating the Root Cause of the Lag Problem	8
1.2	Kobayashi Maru: Change a Rule to Allow a Solution	8
1.2.1	Identify which Rule (Assumption) to Change	9
1.2.2	Introducing <i>Lag Erasure</i>	12
1.3	Contribution and Scope	15
1.3.1	Contribution	15
1.3.2	Scope	16
1.4	Dissertation Organization	18
2	Background and Related Work	21
2.1	Lag	21
2.1.1	Definition	21
2.1.2	Previously Attempted Metrics for Measuring Lag	23
2.1.3	A Precise and Consistent Metric for Measuring Lag	26
2.1.4	The Basic Effects of Lag	28
2.1.5	Assessment: Why is Lag Still a Research Problem?	29
2.1.6	Approach: Minimizing Latency is Insufficient and Ineffective	30
2.1.7	Lag Compensation—Offers a Mitigation, Not a Solution	31
2.1.8	An Alternative to Lag Compensation—But Worse	33
2.1.9	Prediction Layers	34
2.2	Distributed Simulation and Distributed Computing	36
2.2.1	Similarities	36
2.2.2	Divergence	37
2.3	Time Considerations	45
2.3.1	Virtual Time	45
2.3.2	Hardware Clock Synchronization	47
2.3.3	Simulation Software Distributed Time Management	48
2.3.4	Time Advance	49

2.4	Parallel and Distributed Discrete Event Simulation (Analytical Simulations) .	63
2.4.1	The Independence of the Logical Process	64
2.4.2	Conservative vs. Optimistic Advancement and Rollback	70
2.5	Analytical vs. Virtual Environment Simulations	72
2.5.1	Hybridization, for the Win.	73
3	How <i>Lag Erasure</i> Works: Algorithms and Operational Walkthroughs	75
3.1	Speculative Evaluation Provides the Foundation	76
3.1.1	The <code>EventList</code>	77
3.1.2	Movement Representation	78
3.1.3	Collision Detection	84
3.1.4	Multi-threaded Simulation.	91
3.1.5	Distributing Asynchronous Discrete Event Simulation.	98
3.1.6	Speculative Evaluation in Simulation Example—Boids, No External Inputs	101
3.1.7	Speculative Evaluation and Latency Hiding Limits	111
3.2	Asynchronous Discrete Event Simulation with <i>Lag Erasure</i>	113
3.2.1	A <i>Lag Erasure</i> Example—Boids, With External Inputs	114
4	<i>Lag Erasure</i>: Zero-Lag System Design	127
4.1	Design Space and Goals	127
4.1.1	Equipment-in-the-Loop	127
4.1.2	Model-View-Controller-Participant Top-Level Design	129
4.2	Latency and Lag	133
4.2.1	The Mathematics of Latency and Perceived Lag	133
4.2.2	The Consistency-Throughput Tradeoff	135
4.2.3	Operationalizing the Just-in-Time Delivery Pattern	138
4.3	Causality	145
4.3.1	Virtual Time and Clock Conditions	146
4.3.2	The Jugglers, an Example	147
4.3.3	The Singular Virtual Timeline	149
4.4	Asynchronous Discrete Event Simulation.	150
4.4.1	Event Evaluation Loops.	151
4.4.2	Entity State	155
4.4.3	The <code>SpatialMotionApproximation</code> for Motion State	156
4.4.4	Progression Constraints and the Logical Process	158

4.5	Additional Design Considerations.	176
4.5.1	Regions for Area of Interest Management	176
4.5.2	Thresholding and Error Tolerance.	177
5	Use-Case Requirements	179
5.1	Perfect Playback.	179
5.2	The Digital Twin of Reality: Virtual-Space	180
5.2.1	Virtual-Space vs. The Metaverse	181
5.2.2	Digital Twins are the Glue Between Virtual and Physical Realities	182
5.3	Augmented Reality in the Cockpit.	183
5.3.1	Live Aircraft vs. Constructive Forces	185
5.4	Undersea Warfare Tactical Collaboration	185
5.5	Augmented Reality for Everyday Vehicle Safety	187
5.5.1	Motivating Example: The Difficult-to-See Bicycle	187
5.5.2	Shared Reality	188
5.6	Online Multiplayer Action Games.	190
5.6.1	Anti-Cheat Considerations	190
5.6.2	Security Through Ubiquity	191
5.7	Emergent System-of-Systems Analysis.	192
5.8	Realizing These Use Cases	193
6	Future and Enabled Work	195
6.1	Future <i>Lag Erasure</i> Development and Testing	195
6.1.1	Post-Processing of Archived Event Stream Data	195
6.1.2	<i>Lag Erasure</i> Capability Expansion	196
6.2	Enabled Research and Implementation Work	196
6.2.1	Sensor Visualization: Augmented Reality in the Cockpit	196
6.2.2	Sensor Visualization: The Undersea Warfare Sand Table.	197
6.2.3	Online Gaming Distributed Simulation Security Enhancements	197
6.2.4	Reliable <i>What-if?</i> Scenario Exploration	198
6.2.5	Virtual-Space Applications	199
	List of References	203
	Initial Distribution List	213

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

Figure 1.1	Artist’s depiction of a Live-Virtual-Constructive simulation. . . .	6
Figure 2.1	Interlocking states reflecting logical interdependence.	42
Figure 2.2	Simplified synchronous time advance simulation loop.	50
Figure 2.3	A traditional time step state-snapshot trajectory.	50
Figure 2.4	Moments before bat-ball impact.	51
Figure 2.5	Isolation on the bat-ball interaction over one frame.	52
Figure 2.6	The next-event algorithm.	53
Figure 2.7	A typical discrete event simulation state trajectory.	54
Figure 2.8	A fundamental Event-Graph.	56
Figure 2.9	A cancelling edge.	57
Figure 2.10	Iteration and passing arguments in an Event-Graph.	57
Figure 2.11	Sequence, selection and iteration within an Event-Graph.	58
Figure 3.1	An <code>EventList</code> at the start of a simulation.	77
Figure 3.2	The <code>EventList</code> after removing an event for evaluation.	78
Figure 3.3	The <code>startMove</code> event cycle pattern.	78
Figure 3.4	A line-swept sphere capsule wrapping a linear motion approximation.	80
Figure 3.5	The <code>MotionWitness</code> , represented by a buffered capsule.	81
Figure 3.6	A typical discrete event simulation state trajectory.	85
Figure 3.7	A typical time-stepped simulation state trajectory.	86
Figure 3.8	Collision detection visualization, <code>MotionWitnesses</code> in an <code>R*-Tree</code>	87
Figure 3.9	The blue box represents a simulation scene <code>Region</code>	88

Figure 3.10	An R*-Tree intermediate node containing two leaves.	88
Figure 3.11	The scenario's initial EventList.	89
Figure 3.12	EventList reflecting the initial movement recalculation.	89
Figure 3.13	EventList approaching first contact.	90
Figure 3.14	EventList showing StartMove cancellation.	91
Figure 3.15	EventList revealing the cycle continues.	91
Figure 3.16	Basic sender-receiver protocol.	92
Figure 3.17	The protected service process interaction loop.	93
Figure 3.18	Inter-Region communication protocol.	94
Figure 3.19	Entity movement between Regions.	96
Figure 3.20	The Boids World division of responsibility.	107
Figure 3.21	Succession of the lead eagle's MotionWitnesses.	123
Figure 4.1	The Model-View-Controller paradigm.	129
Figure 4.2	Model-View-Controller-Participant system design.	130
Figure 4.3	The always positive measurements on the number-line.	134
Figure 4.4	Relabeling the number-line.	134
Figure 4.5	Traditional distributed simulation event communication timeline.	139
Figure 4.6	Just-in-time communication timeline.	141
Figure 4.7	Visualizing the relationships between event transmission timelines.	142
Figure 4.8	Ping time between New York and Los Angeles.	144
Figure 4.9	Four juggler interaction scenarios.	148
Figure 4.10	A typical discrete event simulation state sequence.	150
Figure 4.11	The asynchronous discrete event simulation next-event algorithm.	151
Figure 4.12	Simplified synchronous time step simulation loop.	151

Figure 4.13	The hybridized simulation loop.	152
Figure 4.14	The core details of the Model simulation event loop.	153
Figure 4.15	A line-swept sphere capsule.	157
Figure 4.16	Jugglers and emcee scenario.	163
Figure 4.17	The MotionWitness containing non-linear entity motion.	166
Figure 4.18	The worst case for relative motion closure evaluations.	170
Figure 5.1	Notional target-tracking radar naïve volume.	184
Figure 5.2	Notional target-tracking radar with standard atmospheric refraction.	184
Figure 5.3	The crossing bicycle is hidden from A, and visible to B and C.	188

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

3D	three dimensional
4D	four dimensional
AABB	axis-aligned bounding box
ACK	acknowledgement
ADES	asynchronous discrete event simulation
AI	artificial intelligence
AoI	area of interest
AR	augmented reality
ART	application response time
AFCS	automated flight control system
AUV	autonomous underwater vehicle
BV	bounding volume
BVH	bounding volume hierarchy
BVR	beyond visual range
C2	command and control
CACM	Communications of the Association for Computing Machinery
CCD	continuous collision detection
CDC	combat direction center
CFD	computational fluid dynamics
CPU	central processing unit
CRUD	create, read, update, delete
DES	discrete event simulation
DEVS	Discrete Event System Specification

DIA	distributed interactive application
DIS	Distributed Interactive Simulation
DIVE	Distributed Interactive Virtual Environment
DR	dead reckoning
ESPDU	Entity State Protocol Data Unit
EODAS	Electro-Optical Distributed Aperture System
EO	electro-optical
FIFO	first-in first-out
FSA	finite state automata
GVT	global virtual time
GNSS	global navigation satellite systems
HLA	High Level Architecture
HUD	heads-up display
HMD	helmet mounted display
IEEE	Institute of Electrical and Electronics Engineers
I/O	input/output
IR	infrared
Hz	Hertz
LAN	local area network
LCW	LatencyCompensationWindow
lhs	left-hand-side
LP	logical process
LLM	large language model
LVC	live, virtual, and constructive
MCAS	Marine Corps Air Station
MMOG	massively multiplayer online game

MOBA	multiplayer online battle arena
MSMA	MultiSpatialMotionApproximation
ms	milliseconds
MVC	Model-View-Controller
NAS	Naval Air Station
NS	Naval Station
NIST	National Institute of Standards and Technology
NLT	no later than
NPC	non-player character
NTP	Network Time Protocol
NVE	networked virtual environment
ODE	ordinary differential equation
PADS	International Conference on Principles of Advanced Discrete Simulation
PaRADE	Predictive Real Time Architecture for Distributed Environments
PDES	parallel discrete event simulation
PDU	Protocol Data Unit
PP	physical process
PTP	Precision Time Protocol
PvP	player vs. player
rhs	right-hand-side
RF	radio frequency
RNG	random number generator
RTI	run-time infrastructure
SI	International System of Units
SMA	SpatialMotionApproximation
USN	U.S. Navy

UDP	User Datagram Protocol
USW	undersea warfare
VE	virtual environment
VR	virtual reality
VRML	Virtual Reality Markup Language
VVA	verification, validation and accreditation
XR	extended reality

Acknowledgments

I am deeply appreciative of my wife Mary, and family, for their love, patience, and support. Innumerable conversations and support sessions about this long term PhD side project have come and gone. Support in managing interactions with a world that often seems unhinged has made it possible to carve out this final result. Thank you.

I have deep thanks for the contributions of my committee. Each of them supported me and this work long before the committee relationship was formalized. Don and I have been debating this topic for nearly two decades, I learned the basics of what is needed to communicate the technical aspects of what follows from those conversations as he had me teach him how I saw the solution. Rudy is an imperturbable conduit for focusing my thoughts on why this work matters to anyone other than a simulation coder. Bret has been a daily colleague who's constant support and frequent mental pushes to continue have had an immeasurable impact on getting over the final hump and into the degree documenting stage. Gurminder and Peter have provided countless years of leadership, sounding-board services and listened when the day-to-day chaos has exceeded normal limits. Gurminder's focused desire to know why this works is reflected in this documents organization, because it is the same question the rest of the world will have. Arnie set the stage, providing the physical and conceptual infrastructure that made asking the questions of how to eliminate lag in a simulation even possible. And without my years of co-teaching with Peter and soaking in the concepts of unifying questions, by repeated osmosis, it is hard to imagine that I could have wrapped my head around the simplifying choices and themes that moved this work away from the Rube Goldberg idea machine it began as. I don't know that any other place on Earth has the correct confluence of people and ideas to enable and grow this research idea to fruition than the MoVES Institute and Computer Science Department at the Naval Postgraduate School in Monterey, CA.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:

Introduction

Simulations are immensely useful training tools, ranging from individual level skills training up through multi-participant large-scale tactical training. The multi-participant simulations make a large jump in technical difficulty because not only must each participant have a valid simulation, but the interdependent interactions for multiple participants must also be consistently valid. The technical difficulty of maintaining the validity of the interdependent interactions takes yet another jump when the simulation participants are not all located in close vicinity, but separated by large geographic distances. This geographic separation of simulation sites creates the *distributed simulation* cited in this dissertation's title.

Lag is the result of a perceptual timing mismatch between a globally expected series of events in a distributed simulation, and the series of events as evaluated locally. Varying physical communication latencies between different sets of simulation hosts further confound the perception of the timing mismatches because the variability destroys ground truth, and each simulation host may display a different evaluation of the event series than is presented at any other simulation host.

Lag is a symptom, not a cause. The user's perception of timing mismatch (lag) directly results from information transmission and display delays outside of the simulation itself, which can be isolated and measured. This idea results in Equation 1.1:

$$t_{lag} = t_{latency} + t_{algorithmic}, \quad (1.1)$$

the total time delay that results in a participant's perception of lag is quantified as t_{lag} . The time delay's components are any network transmission latencies ($t_{latency}$) and time delays resulting from software algorithm implementations and the compute capability of the system ($t_{algorithmic}$). While it is not possible to precisely measure the durations of a user's perception of a simulated event, this perception is directly correlated with the actual event display timeline. So analyzing a simulation event progression, and its relative

timing components may provide a more concrete understanding of how this lag may create unresolvable failures in the causality of event progression.

Misalignment of Timing Expectations. While all participants can see the same time on their local wall clock, and seemingly be provided perfect timing synchronization, it takes time after an action is applied on one host for the announcement of that action to reach the other participants. When a simulation scenario puts geographically separated participants into the same simulated scene, there is an artificial mismatch between the expected instantaneous event-happens-communications in-scene delivery timeline, and the far slower real world message delivery timeline.

In systems programming terms, this mismatch can create a race condition, but unlike logical and methodical systems programming there is no acceptable synchronization primitive or identifiable location for a code critical section to solve the race. Dynamic live simulation displays at every host and tens-of-milliseconds communication latencies present too much communication overhead for synchronization primitives to meet the required real-time interaction rate.

The worst-case example of timing mismatch is illustrated with a simple two participant simulation: **player-A** who is physically located on the west coast and **player-B** who is physically on the east coast of the United States. **Player-A** is holding a virtual pin, and **player-B** is holding a virtual helium-filled balloon.

- **Player-A** initiates an action sequence `moveHandTo(target)` and sends an action message to **Player-B**.
 - On **Player-A**'s display their hand moves and the balloon in **player-B**'s hand pops after a split-second.
- **Player-B** initiates an action sequence `releaseBalloon()` and sends an action message to **Player-A**.
 - This may occur before or after **Player-A** initiates the `moveHandTo(target)` action, either way neither player's message is received prior to taking their initial action.
 - On **Player-B**'s display they see the balloon fly away, **Player-A**'s target location is now invalidated, ensuring the balloon's safety.

The simulation is now in an inevitable fault situation, the action by player-B guarantees a causal chain-of-events failure compared to player-A's version of the simulation.

1.1 The Problem

Lag is what participants blame for weird results. Resolution of this communication timing fault is always problematic, either player-A sees the intact balloon magically reappear in player-B's hand, or player-B sees the flying balloon disappear and a magically popped one appear in their hand. **Both of the resolution possibilities are unsatisfying and break one or the other of the participant's immersion and trust that the simulation is fair or correct.** This example may feel exaggerated, but this is the logic of post-lag causality resolution, and is all too possible when a simulation includes a task involving closely-shared simulation objects (states), transmission delays between sites of more than a tenth of a second (within round-off error for trans-continental network traffic, see Figure 4.8), or exhibit high jitter (unstable and highly variable delivery timelines.)

The simulation style described in this example is “optimistic,” where the simulation hosts simulate through time optimistic that everything is working out fine. Optimistic simulation is good for raw speed, and where there are not any live participants that can be bothered by either of the causality fixes described above. The alternate simulation style is called “conservative” or “deterministic lockstep” where extra timing delays are inserted to allow all simulation hosts to agree that all messages are received and applied before the simulation is allowed to progress forwards in time. While this guarantees correctly ordered results, the overhead necessary to accomplish the ordering makes the simulation progress too slowly compared to participant expectations.

This leaves live-participant simulation designers with a conundrum: accept causality errors and try to find alternate means to make them as unnoticeable as possible, avoid distribution keeping simulations to a limited scope, or ignore the problem and attempt to post-event-debrief-away the effects of timing mismatches. None of these options are good.

1.1.1 Lag Is Provably Unavoidable?

The communications timing mismatch between in-scene expectations and real-world physical message delivery timelines is well documented in the simulation community and is

considered to be impossible to avoid, with informal proofs provided in textbooks and confirmed in later textbooks. In 1999, Singhal and Zyda's *Networked Virtual Environments* [1] formally identifies the problem and provides a proof named the *Consistency-Throughput Tradeoff*. A decade later Steed and Oliveira agree in their 2010 title, *Networked Graphics* [2] reiterating the same concepts.

The simulation community remains in consensus that the lag problem is impossible to eliminate, so simulation community research concentrates on the means to disguise the effects of lag instead. Disguising is not fixing. Literally, disguising lag means abandoning the goal of correctness. Unfortunately this abandonment of the correctness choice and the simulations that result from it limit the scope and credibility of what a distributed simulation can provide its users.

1.1.2 Why Should Anyone Care? It is Settled, and Proven!

The desired scale of interaction for simulations and the physical location of specialized hardware for testing or training are driving factors in the growing necessity for distributed simulations rather than remaining with small-scale local-only simulations. For multiple participants at different simulation sites to make the same appropriate time-critical decisions, there is a requirement for consistency in how the simulated world-states are presented across all the simulation sites. This means the timing-expectations mismatch is hampering the ability to create useful, correct, distributed simulations. The simple example presented at the beginning of this Chapter becomes ever more interwoven as each and every interaction between distributed participants creates increasingly different world-states and diverging event timelines across the entirety of the simulation. In military training contexts it is common for a desired decision rate to be split-second once the proper presentations are made. If all participants split-second decisions are based on different timelines the capability to train interdependent actions is compromised.

1.1.2.1 Why Should a Nugget¹, or an Admiral, Care?

While the problem description is getting into the simulation developer's technical territory, the effects of the problem for the users of the simulations remain front and center.

¹a nugget is a first-tour Naval Aviator on or preparing for their first deployment.

Nuggets and admirals both rely on the same simulations, and the complexities of simulating modern military operations are requiring ever-more realism and fine-grained fidelity to achieve the training goals for the simulation session. While the nugget “does” the simulated tasks to learn new skills, the admirals are the consumers of the results of all the trained participants including the nugget(s).

The most complex of these simulations are the live, virtual, and constructive (LVC) simulations, which include both live units on the training range, and trainees in systems simulators that are usually located hundreds to thousands of miles apart. The level of effort to support the logistics and technical requirements in putting an LVC simulation together is massive, but it is far less resource-intensive than duplicating the underway time and flight hours over multiple training sessions before successfully completing the fully-live version of the same training.

The existence of simulations has created cost-reduction pressure on operating forces field training for decades, and therefore simulation efficacy is a critical part of the overall training cost-reduction conversation. It is important for the Navy users, nuggets through admirals, to understand any profound limitations in the simulations they rely upon, not just leaving limitations to the simulation designers and coders.

1.1.2.1.1 A Distributed Live, Virtual and Constructive (LVC) Training Simulation

Battlegroup workups often use a consulate evacuation scenario from a fictitious country as an advanced training exercise. Simulation mission rehearsals prepare less experienced personnel on the proper communication and interoperability procedures before the full costs of underway fuel, dozens of aircraft’s flight time, and safety of operations in a complex environment are fully exercised. When a nugget participates in a large simulation that is part of the battlegroup training pipeline they may do so as the junior member of a flight of two to four fighters in flight simulators physically located at Naval Air Station (NAS) Lemoore in California’s Central Valley, hundreds of miles away from the combat direction center (CDC) on the battlegroup’s carrier which is pier-side at NAS North Island on the San Diego Bay.

The consulate-evacuation mission under training additionally requires the participation of two battlegroup helicopter simulators from NAS North Island, the CDC of a big-deck landing platform dock amphibious support ship across the bay at Naval Station (NS) San

Diego, multiple Marine helicopter simulators from Marine Corps Air Station (MCAS) Miramar, the CDC of a battlegroup cruiser pier-side at NS Pearl Harbor, multiple adversary aircraft providing live-tracked flight services at air training ranges near NAS Fallon in west-central Nevada, and the offshore range of the San Clemente Island Range Complex where a battlegroup E-2 aircraft is flying live on the range with some of the adversaries. In addition to providing the standard range services for the airborne aircraft in the simulation, the training ranges also provide the computational hubs for a large number of simulated friendly and adversary entities in the simulation scenario.

This LVC simulation mixes **live** participants (the E-2 and adversary aircraft as well as the ship CDC teams) with **virtual** participants (all the human-controlled flight simulators) and the purely **constructive** computer simulated entities added to the simulation. A conceptual illustration is provided as Figure 1.1 [3].



Figure 1.1. Artist's depiction of an LVC simulation. Source: [3].

Delivering a LVC simulation typically uses capabilities of the training ranges to integrate the information generated from live sensors and computer-generated information from the virtual and constructive participants, provide all of the participating simulators the appropriate networked simulation data, and pass along the fully-fused LVC simulation data using command and control (C2) link technologies to live platforms and trainees participating physically, both on the training range and also those platforms not onsite or underway, such as the ships.

1.1.2.1.2 Simulation Trustworthiness is On The Line The nugget wants to learn “what to do,” the admiral wants everyone to learn how to work together effectively and minimize future mission risk levels. If the realities of the networked simulation mean different site participants are presented world-state display data that paints a different picture than other participants are receiving, the decisions which participants at each site make are at risk of not supporting each other appropriately. When the simulation has failures due to the artificialities of this timing and world-state difference risk, the effectiveness of the training is reduced, and further it reduces trust that the training is meaningful. *How can anyone trust a simulation that does not provide consistent information to everyone and therefore leads to undesirable decisions?*

1.1.2.1.3 Why Should This Make a Nugget and an Admiral Uneasy? The age-old “good-enough” solution to “fixing” the network simulation related inconsistencies is analysis and debrief, guided by personnel with the experience to help the participants mentally resolve simulation inconsistencies and get the maximal positive training value possible. **Now for the unease:** how does an expert debrief the artificial intelligence (AI) tactical assistant-in-training to resolve simulation inconsistencies and receive maximal positive training value?

AI accomplishes stunning tasks when it is properly trained. AI is also amazingly brittle and performs poorly if the task falls at all outside the training data provided. The availability of human-level common-sense is still well into the unknown future of AI development, meaning the ability to provide an AI post-event analysis and debrief to resolve inconsistencies is currently and near-term-future infeasible. If simulations are not providing consistent enough data for the AI trainee the only option is live training. But live training everything necessary to produce enough training data for an AI tactical assistant is infeasible; there is

not enough time or resources to execute the number and breadth of repetitions necessary to adequately insure the AI's training against brittleness.

This unresolvable problem indicates it may be difficult to ever trust an AI tactical assistant because the training is too limited in scope, insufficient numbers of live training repetitions, or the preemptive avoidance of inconsistent distributed simulations. This collective barrier to training non-brittle future autonomous and AI systems is unacceptable. AI tactical platforms are coming, sooner rather than later. *Better networked simulation technology that removes the artificial training deficiencies is the only alternative.*

1.1.3 Getting Technical: Isolating the Root Cause of the Lag Problem

Section 1.1 refers to lag as “what participants blame for weird results,” which is not precise or actionable, but the definition: **Lag is the result of a perceptual timing mismatch between a globally expected series of events in a distributed simulation, and the series of events as evaluated locally**, is precise and actionable. This remains true despite the confounding variable effects of physical communication latencies in different host pairs. The right-hand-side terms of Equation 1.1 are measurable, and it is their total magnitude that constrains the maximal misalignment of timing expectations.

Currently, the simulation field treats both terms on the right-hand-side (rhs) of Equation 1.1 as positive values to be minimized. The resulting t_{lag} is therefore *always* positive. This always-positive left-hand-side (lhs) total value supports the textbook proofs [1], [2] mentioned in Section 1.1.1. The consensus follow-on result is that lag can only be visually mitigated with techniques to disguise the problem. None of this mitigation acts to correct the growing state divergence across the distributed sites in the larger simulation.

1.2 Kobayashi Maru: Change a Rule to Allow a Solution

Star Trek's Kobayashi Maru scenario is the fictitious Starfleet simulation designed to test cadet's command leadership qualities under a no-win scenario. Cadet James T. Kirk famously reprogrammed the simulation's rules, and is the only cadet to ever succeed and survive in the simulation. The existing no-win rule of simulation is that the $t_{latency}$ and $t_{algorithmic}$ terms from Equation 1.1 may only ever create a positive value, making significant values of t_{lag} inescapable.

The inevitability of $t_{latency}$ and $t_{algorithmic}$ to only ever create a positive value is dependent on specific implementation assumptions and choices used in the traditional synchronous fixed time-step simulation paradigm. Determining which combination of relevant assumptions to challenge is not a trivial task since the current simulation technology stack with all its supporting libraries is well established around the same common implementation choices and assumptions.

1.2.1 Identify which Rule (Assumption) to Change

Attempting a top-down design exploration is not likely to be successful, too much opportunity to lock in an incompatible design choice early and preclude reaching a true reduction or elimination of lag. The bottom-up approach begins by taking inspiration from Section 1.1 which describes why lag occurs, then turning the issue inside out by examining the necessary mathematical relationships, and evaluating to determine if a design concept emerges that supports an end-result of a lag-free simulation. Starting with the goal of no lag it follows that the user-perceived delay (t_{lag}) from Equation 1.1 must be zero, yielding

$$0 = t_{latency} + t_{algorithmic}, \quad (1.2)$$

and to attain this the strictly positive values of the absolute network transmission latencies ($t_{latency}$) must be offset by creating a negative value for $t_{algorithmic}$, resulting in

$$-t_{algorithmic} = t_{latency}, \quad (1.3)$$

which indicates latency hiding at the distributed application layer can result in lag delays of zero, and this may be implemented via speculative evaluation. This is similar in concept to other hardware and software layers of the technology stack, where latency hiding techniques are used to provide just-in-time delivery of information and eliminate avoidable information waiting delays.

The assumption that lag is inevitable, is one step closer to falling. The next assumption to be tackled is that time measured must always be represented with a positive value because time only flows in one direction. This assumption is central to the virtual environment (VE)

simulation community understanding that lag is inevitable because it results in a rhs of Equation 1.1 that can only be positive valued. The rearranging of terms in Equation 1.3 does not imply that the clock moves in reverse, the absolute magnitudes of both $-t_{algorithmic}$ and $t_{latency}$ are still positive. The rearranging of terms explores the relative relationships of before-after, and opens the question of whether or not there is a design which specifies how $t_{algorithmic}$ evaluates as negative, or “before,” and relative to what?

All simulations have multiple timelines in play, the real world wall-clock timeline that the physical progress of the central processing unit (CPU) is measured against, and the simulation’s event timeline which indirectly results from those CPU evaluations. Traditional virtual environment simulations physically evaluate an event on the CPU at or slightly after (on a scale of nano-to-microseconds) the event’s simulation timeline time, and due to synchronous time step scheduling this is “close-enough” to consider it “simultaneous.” Binding both timelines to the same zero-point is a design choice and if the CPU is fast enough it can be a good choice. It is also the choice that ensures lag is inevitable because the rhs of Equation 1.1 can only be positive valued.

1.2.1.1 Evaluate Speculatively, Not Late

The simulation timeline emerges as a virtual result based on the relative posting of the CPU event evaluations, rather than the ticks of a physical clock. This allows the design choice of considering an event on the simulation’s virtual timeline to be the zero point when considering the timing relationships of computations that document that event. Following that idea one step farther towards implementation leads to the deceptively simple answer to turning $t_{algorithmic}$ negative—use speculative evaluation to pre-compute events.

The pre-computation takes advantage of the difference between the real world wall-clock timeline and the simulation’s event timeline. Arranging so that the CPU’s wall-clock timeline evaluation of an event falls “before” or to the left (negative side) relative to where the event falls on the virtual simulation timeline. Translating the absolute time of an event on the simulation timeline into a zero point for this relative comparison is trivial and can be accomplished without any loss of generality or additional effects on the simulation display presentation.

When using speculative evaluation the event's evaluations are completed milliseconds before the event's simulation timeline time, simulation designers are able to statically choose this pre-computation offset or dynamically determine it at runtime based on network observations. Either way, the choice allows the transmission latency to conclude before the event's simulation timeline time, instead of after.

1.2.1.2 Result: A Lag-Resistant Foundation

Evaluating an event computation in negative relative-time on the simulation's timeline constitutes a significant change of the rules in simulation construction. The resulting just-in-time information delivery eliminates the lateness of event message delivery to every simulation participant in the distributed simulation, preventing them from seeing a different simulation world-state. By maintaining the just-in-time information delivery, every participant's simulation host can maintain a consistent state shared with every other participant in the simulation. There are a myriad of engineering tradeoffs and implementation details that follow the choice to use speculative evaluation, which are the subject of Chapter 3 and Chapter 4.

This lag-resistant foundation that speculative evaluation provides is not enough on its own. Speculative evaluation provides latency hiding for every scheduled simulation event, but the system cannot speculatively evaluate something that does not derive from an earlier processed event. When a simulation user makes an input/output (I/O) device input, a new event is generated, one that is not derived from a previous event, and it is obvious that the simulation system cannot speculatively evaluate an event that does not know about until after the real-world I/O occurs. This means the worst-case timing mismatch problem is not yet managed.

The speculative evaluation provides the foundation for lag-resistance, but is not the entire answer. Continuous speculative management of non-input device related simulation progression and use of physics models advantageously constrain the magnitude of any state divergences that develop in direct response to the input device events arriving late. The constraints limit any resultant state differences between the distributed sites to a level that is algorithmically "erasable" with a limited amount of local-to-each-distributed-site rework, and prior to the state differences creating a causality issue noticeable by a simulation partic-

ipant as a lag situation. Without this foundational principle lag erasure is not possible. With this foundational principle the rework process is able to repair the effects of communications latency quickly enough to be useful.

1.2.2 Introducing *Lag Erasure*

Lag Erasure is the solution to the problem distributed simulation actually has, rather than yet another technique for papering over (mitigating) the problem that the simulations field believes it has.

Lag Erasure is the simulation system capability that uses the lag-resistant foundation provided via speculative evaluation, and “erases” the effects of late-arriving events from the simulation state. Late-arriving events include all participant input-device activation events, but they also includes the occasional incoming message that is affected by bad network jitter conditions and fails to meet the expected delivery timeline.

Lag Erasure acknowledges latency and unavoidable transient moments of state divergence between distributed simulation sites due to the worst-case situation of late-arriving events such as input-device control inputs by trainees or players, since there is no way to transmit a real-world input device activation speculatively. *Lag Erasure* however leverages the state consistency that speculative evaluation foundation is able to provide to accomplish the minimal rework necessary to “erase” state divergences and heal the simulation back to a consistent shared state across the entire set of participating distributed hosts. With a consistent shared state, the root cause of user experienced lag that is identified in Section 1.1.3 is eliminated.

It is time to revisit the previously exercised worst-case example of timing mismatch to cover how the *Lag Erasure* system resolves the situation and removes lag from the shared distributed state of the simulation.

1.2.2.1 Revisiting the Balloon Example with *Lag Erasure*

Revisiting the two participant balloon simulation: player-A is physically located on the west coast and player-B is physically on the east coast. Player-A is holding a virtual pin, and player-B is holding a virtual helium-filled balloon. This is the same starting configuration as the Chapter-opening version. Both players initiate their actions as external

I/O device inputs, generating the worst-case setup for a misalignment of timing expectations because there is no way to speculatively resolve the initiated actions. Despite the foundational shared consistent simulation state prior to the player inputs, the full time-late arrival latency for each transmit event exists, creates a mis-ordering and potentially begins the breaking of expectations, a situation which must be “erased” before the mis-ordering is noticed and contributes to the players perceiving lag.

- **player-A** initiates an action sequence `moveHandTo(target)` and sends an action message to **player-B**.
 - On **player-A**’s display, their hand begins moving towards the target point in accordance with an anthropometric motion model.
- **player-B** takes the action `releaseBalloon()` and sends an action message to **player-A**.
 - This may occur before or after **player-A** initiates the `moveHandTo(target)` action, either way neither player’s message is received prior to taking their initial action.
 - On **player-B**’s display they see the balloon begin to rise in a physically-realistic accelerating manner.
- Some delay after **player-A** initiates the attempt to pop **player-B**’s balloon, the message from **player-B** arrives to **player-A** announcing that **player-B** has released the balloon.
 - **player-A**’s simulation host executes a rework procedure to resolve the late-arriving action from **player-B**.
 - * The rework locally evaluates equivalent physics code to that employed by **player-B** and determines that the balloon moved slightly upward from its original position.
 - * The balloon position is adjusted to the newly correct position and use of visual motion blur from the original balloon position is employed. The rising balloon is now in the same position and moving according to the same motion model on both players simulation displays. **Player-A**’s visualization of their hand on the way to popping the balloon is unchanged.
 - In **player-A**’s visual perception the ballon motion seems entirely natural despite the slight absolute teleportation. The limited magnitude of

the possible motion, on the order of millimeters across a couple frame delays, and a lifetime of interaction with moving visual media has reinforced the mental synchronization of slight visual variances when multi-cue timing mismatches are within one tenth of a second or less. The motion blur seals the illusion.

- Some delay after **player-A** initiates the action sequence `moveHandTo(target)` the message arrives at **player-B's** host.
 - **player-B's** simulation host invokes rework, using an equivalent anthropometric motion model to the one used by **player-A** and also applies the motion blur technique on the resulting motion of **player-A's** extending hand. The advancing hand is now in the same position and moving according to an equivalent motion model on both players simulation displays.
- The simulation progresses without further input and at the appropriate time, the needle intercepts the balloon which continued to accelerate upwards. The contact point of the needle is low and closer to the tie than **player-A's** target aim-point. Model-based inertial follow-through of the hand slightly beyond the original target point worked in **player-A's** favor.
- At the time-of-contact, the simulation hosts of both players make a pseudo-random pull from local synchronized pseudo-random number generators and both hosts determine the balloon does not violently pop, it is punctured and jets away with the extra thrust from the escaping helium. Both **player-A** and **player-B** see the same result from their own vantage points.

The previously inevitable fault is avoided, the causal chain-of-events is in agreement for both players despite network transit times and jitter. The example does appear more complicated on first glance, but the extra complexity in time management and causality resolution is managed by the ***Lag Erasure*** system itself. The simulation designer's focus is slightly shifted to considering event communications as start points rather than resultant-end points in the sequence, and still the same two events: **player-A's** popping, and **player-B's** release defense. The motion blur is not a component of the ***Lag Erasure*** system, the example mentions this common visual technique to illustrate that the positional differences rework causes are bounded, small, and hide-able by the visualization layer without use of

in-simulation-layer dead reckoning or smoothing, and with no breaking of user immersion due to unmet expectations of realism.

The ***Lag Erasure*** algorithms provide a key and previously unattainable distributed simulation property—consistent shared distributed state—which translates in participant experience to *lag-free interactions* across the entire distributed simulation.

- This lag-free property for a simulation removes the need for expert-led analysis and debrief to explain-away or “fix” the network simulation-related inconsistencies (identified in Section 1.1.2.1) that trainees experience.
- Distributed simulation world-state differences due to the random latency-related timing effects of lag are eliminated, making distributed simulations repeatable and verifiable in a way that increases trust that the simulation results are meaningful and indicate real world actionable insights.

This verifiability and trustability is necessary for large-scale training and evaluation of autonomous vehicles and robots, because meaningful test scaling and resource requirements require more than isolated single-site provisioning. If the pre-***Lag Erasure*** existing situation is that the required distributed test environment is demonstrably not trustable, and any tests dependent on that environment cannot be trustable, it is a significant upgrade to eliminate this problem. Lag-free and therefore trustable distributed simulations are a unique capability of the ***Lag Erasure*** distributed simulation architecture.

1.3 Contribution and Scope

1.3.1 Contribution

This dissertation’s primary original contribution is the *Lag Erasure* architecture and algorithms which provides users a lag-free distributed simulation. The *Lag Erasure* architecture includes novel handling of time offsets to decouple network latency (including jitter) effects from maintaining distributed shared state.

Lag-free distributed simulation exhibiting consistent distributed shared state is repeatedly claimed by the literature to be impossible. Removing the impossibility enables capabilities that are considered to be grand challenge level [4], [5].

This architecture provides consistent distributed shared state across all participants, and this shared state closely approximates the results expected of using a real-time deterministic lockstep protocol without the egregious progression penalties. Externally generated inputs from remote participants do create a transient state divergence between hosts on the network, but the **Lag Erasure** algorithms successively erase these divergences restoring the consistent state within a short, bounded time. Thus, the hard mathematical limits of the *Consistency-Throughput Tradeoff* [1] still hold in a small corner of the operating envelope that manages the external inputs, but the new **Lag Erasure** approach erases that unavoidable component of lag before it is noticeable, and opens the rest of the distributed simulation operating envelope to be lag free.

1.3.2 Scope

The *Lag Erasure* architecture describes a software architecture targeting simulation models and events at human time and distance scales intended to support real-time human/equipment/world-in-the-loop applications.

Equipment-in-the-loop users include both software agents executing outside of the code of the simulation as well as any form of hardware-based entity that evaluates and makes decisions based on the simulation state but is not computing those decisions inside the simulation. This includes hardware as simple as a ground-fault power outlet that consumes simulation data on electrical current states, through highly complex autonomous vehicles that have their sensors and output activation systems connected to the simulation.

Of related note, a resurgent interest in world-in-the-loop simulation is re-appearing in the literature, but without consistent usage. World-in-the-loop simulation is described at least as far back as [6] and as considered here conceptually flips the concept of equipment-in-the-loop inside out to where an autonomous vehicle (or C2 system) runs an internal VE simulation to assist in interpretation and prediction of real-world states to facilitate decision making.

Human-scale is generous in range, basic microscopic distances to whole Earth models and low Earth orbit, even extra-Earth object motions within the Solar System are appropriate, with extra care for maintaining appropriate numeric limits and fidelities. But attempting to model the high-fidelity relativistic effects on time with fast moving satellites and

slow-moving gravity-well resident surface entities is out of the intended scope, as the *Lag Erasure*'s concept of a singular definitive timeline is not reconcilable with multiple relativistic effects on time. The prime consideration is that the simulation includes an interacting physical participant and that the modeled situation meets the level of abstraction necessary to support the desired interactions with the participant.

Lag Erasure simulations can support faster than real-time analytical distributed simulations where the simulated models use physically based modeling of motion and collision physics, and there are no humans or equipment-in-the-loop participating, especially large-scale simulations taking advantage of simulation supporting assets at multiple sites. The explicit and intrinsic use of physics provides useful constraints that are leveraged in partitioning and parallelizing the simulation advantageously, both locally and across widely distributed geographic sites.

This research does not provide a general distributed simulation solution for arbitrary situations and models at arbitrary computational rates; it is primarily targeted at simulations with human and machine participants. Despite not being a general solution, *Lag Erasure* enables an extensive design space as evidenced in the previous paragraph. The design space limits also exclude evaluating models requiring high-frequency updates of densely packed entities. Examples of what to avoid are simulating the internals of an avalanche as it progresses down a mountainside or executing detailed computational fluid dynamics (CFD) simulations, which is a modality also noted by Mirtich as challenging [7].

Importance:

Consistent shared state across a distributed networked simulation is a profound new property in the simulation domain that appears to address the root cause(es) of multiple grand challenge level issues [4], [5] which have prevented shared simulated worlds from achieving widespread viability. Consistent shared state via *Lag Erasure* is not the solution to every problem and challenge in network distributed simulations; it does however provide the verifiable and repeatable state baseline that is necessary to engage on meaningful research and resulting implementations moving forward.

The lag problem cuts across all distributed shared simulations including: C2 systems, the LVC paradigm in military training, multi-platform weapons systems data sharing and displays, digital twins, multi-user virtual worlds for training systems, multi-user augmented

reality (AR) for both training and in-the-field, the entirety of the multiplayer online gaming community. It also applies to the shared massively multi-user AR and virtual reality (VR) applications that are developing into the long awaited metaverse and the ubiquitous everyday virtual-space that Brin [8] predicts the metaverse is diffusing into. Every listed application here is either measurably degraded by laggy communications or precluded from being viable due to how lag breaks proper sharing of the simulated world.

This research explicitly does not cover specific simulation networking protocol comparisons because this is not a protocol-related dissertation. **Lag Erasure** is an architecture and set of algorithms that provide the new capability for erasing lag from the users perception of the simulation. There are undeniable benefits to a future network distributed simulation protocol or updates to existing distributed simulation protocols, but that constitutes future work.

1.4 Dissertation Organization

This dissertation is organized in the typical chapter structure for academic writing. The presented chapter ordering is:

- This chapter (Chapter 1: *Introduction*) describes the lag problem and introduces the **Lag Erasure** solution.
- Chapter 2: *Background and Related Work* provides the theoretical background coverage laying out previously existing state of the art.
- Chapter 3: *How Lag Erasure Works: Algorithmic and Operational Walkthroughs* contains a progressive set of descriptions illustrating **how** the **Lag Erasure** architecture and algorithms work at a Software Architect or Project Technical Lead level to support creation of a lag-free simulation.
- Chapter 4: *Lag Erasure Zero Lag System Design* describes **why** the **Lag Erasure** architecture and algorithms work at a Simulation Designer and Software Engineer level, covering the pragmatic detailed low-level engineering design and systemic technique selections and tradeoffs used in constructing the **Lag Erasure** system.
- Chapter 5: *Use-Case Requirements* contains a set of descriptions useful to Project and Program Managers for where and how **Lag Erasure** system may make current simulation applications better or enable new ones.

- Finally, Chapter 6: *Future and Enabled Work* is coverage of follow-on work that may benefit from using **Lag Erasure** and more importantly a discussion of what types of future work is newly enabled by providing an effectively consistent shared state.

Not all readers of this document are accessing it from an academic perspective, for professional readers from the simulation and gaming industries the preferred reading order may be different to avoid tangling with the heavy background information before an indication of why that background is included.

- It is suggested that industry readers in management or product lead roles begin with Chapter 1 and scan 3 as deeply as necessary for gaining the basic understanding of how the **Lag Erasure** architecture operates and why it matters to simulation projects.
- For line and lead software developers, begin with Chapters 1 and 3 which provide high level coverage useful for understanding of how the **Lag Erasure** architecture operates. Then Chapter 4 contains the lower-level theoretical and engineering reasoning behind the design choices for the **Lag Erasure** architecture, and some of these design concepts rest upon understandings of previous industry and academic knowledge contained in Chapter 2.
- Non-technical defense industry readers are suggested to begin by reading Chapter 1, scan Chapter 3 as deeply or shallowly as comfortable, and then move into reading Chapter 5 to gain an understanding of where the new capabilities provided by the **Lag Erasure** architecture fits from a programmatic and future uses viewpoint.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:

Background and Related Work

This chapter provides the theoretical background coverage, beginning with the common issue of lag then laying out previously existing state-of-the-art in parallel and distributed analytical simulation, and virtual environment-based simulation.

2.1 Lag

Lag is the enemy of so much in the realm of geographic distribution of simulation systems. To some it is a boogey-man to be blamed for whatever currently ill the participant, to others it is a problem to hide behind a veil of techniques attempting to minimize the harm to the process being simulated. Others just believe it is inevitable and accept the artifacts and problems it creates in a simulation because information only travels through wires and airwaves at speeds less than that of light through a vacuum. They are all simultaneously right and wrong, afflicted by imprecision of concept as to exactly what lag is and how that limits the ability to tackle the problem of lag head on.

The upcoming section starts with the working definition, and how to measure lag. Then covers previous techniques to minimize lag's effects in the realm of distributed simulations.

2.1.1 Definition

Lag is the result of a perceptual timing mismatch between a globally expected series of events in a distributed simulation, and the series of events as evaluated locally.

Varying physical communication latencies between different sets of simulation hosts further confound the perception of the timing mismatches because the variability destroys ground truth, and each simulation host may display a different evaluation of the event series than is presented at any other simulation host. In systems programming terms, each individual timing mismatch can create a virtual race condition, but unlike controlled and methodical systems programming, there is no acceptable synchronization pattern to solve every race simultaneously and consistently.

The definition in the above paragraph differs significantly from the existing simulation domain definition of lag: *the perceived time delay between an action and displaying the result of that action back to the participant who input the action*. This is simple, and local, and useless once the network is considered with multiple dependent interactions of multiple users that are not all in the same geographic location, let alone the same machine.

Perception as part of the definition, both in this dissertation's definition and in the (deficient) traditional version is a double-edged sword. Lag is what the user perceives, but it is caused by other factors that collectively contribute to the problem, and it is those other factors indicated in the following

$$t_{lag} = t_{latency} + t_{algorithmic}, \quad (2.1)$$

that require addressing. The ultimate goal always comes back to providing a simulation that the user perceives as having no lag. When considering Equation 2.1 in terms of over-the-network effects, the lag is the wall-clock measure of time-late from when some event happens on the local host and eventually is delivered to a remote host. Both the network transmission and the simulation software contribute to this lateness, and when the total lateness (right-hand-side) contributes to displayed perceivable inconsistencies or causality failures the lag is finally realized. The component contributing delays are always there, but not always perceived by the user. Not considering the user's perception strongly enough, leads to systemically not addressing the causes of lag, while over-weighting the issue of perceptibility has different problems that make precise measurement and understanding impossible. The next section begins by deconstructing the specific problems in over-weighting perception.

2.1.1.1 Perception in the Traditional Definition is Problematic

Perception is non-specific and yields different results in different contexts. For a human user engaged with a single-machine simulation the absolute delay could be less than a single frame update. In this case the delay would be imperceptible. However, if the user is a hardware-in-the-loop entity with the ability to discriminate fine-grained differences between an expected state and the lagged state within the time duration of a single frame then the imperceptibility is gone. This means a lag definition that relies on human relative

and absolute perceptibility is incapable of identifying lag problems that may be problematic for user-participant hardware and software entities that are becoming more necessary and common in large-scale simulations. Complex future AI tactical assistants are obvious entity-in-the-loop candidates where the above lag definition is inadequate, but current day simple robots without advanced time-analysis capability are even more likely to be negatively affected by lagged display presentations. This dichotomy between one type of user (human) potentially being satisfied while hardware or software entities find the interactive world state to be frequently corrupted to the point it cannot make a verifiably correct decision indicates the need for a more precise and consistent metric of codifying lag.

The mushiness of perception gets even more imprecise because the user-perceived lag in a network distributed simulation is not isolated to displaying the result of locally input actions back to the entity who input the action. The lagged display incorporates the results of combined behavioral interactions resulting during the round-trip of transmitting a local input action-1 to the remote host(s), the remote host(s) incorporating the input for display to the associated remote user(s) who may incorporate action-2 and action-3 which further depends on state that should contain action-1, but has not yet. This intertwined creating and transmitting of new compound-changes to the scenario which upon receipt back to the original user's site might possibly obfuscate or modify the local display of action-1 [9]. In other words, the ripples from distributed errors have cascading effects. Is it even possible for a user to conceptually separate any expected simulation state delays sensed on their local display(s) into the local and round-trip compound-effect components deriving from processing local input device activations?

The inherent imprecision in simulation participant perception and commingling of interdependent sources of lag on the displayed simulation state means that relying on perception makes it challenging, or potentially impossible, to devise a means for discretely identifying and removing lag contributions, and therefore an alternative non-perceptual approach is necessary to characterize lag.

2.1.2 Previously Attempted Metrics for Measuring Lag

The measure of lag needs to be precise and identical for both engaged humans and hardware-in-the-loop users. Instrumenting users to be able to measure user perception indirectly via

reactions does not provide user-system independent metrics. Evaluating simulation event data in relation to a definitive simulation timeline isolates all user perceptual mechanism differences from the lag measurement problem by analyzing the source data from which all users are sampling via their individual displays. The lag measure presented as Equation 2.1 is a starting point, however network distributed simulation round-trip effects cause additional state-of-the-world changes dependent on the combined interactions of the whole distributed simulation, making it challenging to isolate exactly what changes are attributed to which inputs and corresponding lag time-delays. The integrated combined effects in the resulting round-trip simulation state make identifying specific sub-effects attributable to a specific event's t_{lag} a fool's errand precisely because the lag effects change how the local and remote users perceive the simulation state, and they make decisions based on states impossible for each other to know definitively. Attempting to unpack decision trees and supporting state data re-injects ambiguous perceptual effects into the data. The computational overhead and ambiguity create an intractable mess for isolating individual event t_{lag} display delays out of the combined interactions of the whole distributed simulation. This conundrum repeats similarly and uniquely for each participant. So, although t_{lag} is a useful metric as an expected maximum target value to use in a future design tradeoff, it does not directly provide a consistently measurable metric on individual event propagations within the distributed simulation.

For the purpose of developing an unambiguous and efficiently computable metric, consider Singhal and Zyda's [1] proof of the *Consistency-Throughput Tradeoff* which states:

To guarantee absolute consistency between hosts, the data source must wait until everybody has received the information before it may proceed with the computations (or, in particular, update the shared state again). [1]

Rephrasing with respect to the measurable lag metric: in a simulation without lag, the states of all the entities across all the participating distributed hosts are the same. This restatement works because “wait until everybody has received” implies a time-delay of zero if there is no lag in the distributed simulation. In addition, if a real-time simulation progresses without lag, then there is “consistency between hosts.” Unfortunately the mentioned “wait” is not feasible for a simulation progressing in real-time because for distributed participants there

is never a time-delay of zero and the clock on the wall moves ahead oblivious to whether a state update might require a delay or not.

Further stated in Singhal and Zyda's [1] proof of the *Consistency-Throughput Tradeoff*:

Conversely, when a state change is generated at a particular host, there is a necessary delay before the other hosts learn of the change; this delay consists of the network latency for transmitting the update over the network. Consequently, if the shared state is updated often enough then the shared state might be updated while the previous update messages are still in transit to remote hosts. As a result, some hosts may see a new state value while others still see older values being disseminated by the state consistency protocol. [1]

That last sentence, “some hosts may see a new state value while others still see older values being disseminated,” indicates that deltas between individual entities' states across the distributed simulation is a measurable proxy metric. Unfortunately, this measurement and strict discussion of this consistency is almost universally avoided, with the exception of a lone publication in 2004 titled *Time-Space Consistency in Large-Scale Distributed Virtual Environments* [10].

Zhou *et al.* [10] presented a mathematical definition for their consistency metric

$$\Omega = \begin{cases} 0, & \text{if } |\Delta(t)| < \epsilon, \\ \int_{t_0}^{t_0+\tau} |\Delta(t)| dt, & \text{if } |\Delta(t)| \geq \epsilon, \end{cases} \quad (2.2)$$

which contains human perception within the statement “minimum discernible distance” where the value ϵ is the human perceptible difference over the relative time duration τ . Equation 2.2 is then simplified to

$$\Omega = \begin{cases} 0, & \text{if } |\Delta| < \epsilon, \\ |\Delta| \cdot \tau, & \text{if } |\Delta| \geq \epsilon, \end{cases} \quad (2.3)$$

under the assumption that $|\Delta(t)|$ may remain constant over the interval τ [10]. The utility of computing a three dimensional (3D) volume encapsulating the “minimum discernible distance” of positional inconsistencies and multiplying that by a time interval is not obvious and the post-processing computational effort required to generate it is significant. Further, the inclusion of perception renders the metric inherently variable and impossible to generate reproducible deterministic results per the discussions of Section 2.1.1.1. In a system where entities undergo 2^{nd} order accelerations the assumption of velocity constancy is invalid because position is not linear over time and would thus require more expensive computational integration methods to implement Equation 2.2 directly. This makes the proposed metric less useful than it first appears. Another issue with the metric is that it resists any algorithmic mechanisms to dynamically update local entity motion data with received information applied at its timestamp time instead of the later receive time. This makes it unsuitable for comparing consistency performance of differing simulation algorithms in any situation where state lag effects are retroactively corrected to the timestamp time.

2.1.3 A Precise and Consistent Metric for Measuring Lag

It is possible to continue to simplify the concept of consistency down to **measuring the time intervals of inconsistency**. Removing Zhou’s [10] computation of perceptual volumes associated with the positional deltas does not negatively impact the intended goal of measuring consistency, because even when entities show inconsistencies the actual deltas are small compared to the relative magnitudes of the entity motion. It is true that faster movers may generate larger errors in absolute terms, but that error computation is always in direct proportion to the magnitude of the velocity vectors and those velocity vectors are always multiplied by the time intervals of the inconsistency to generate the positions used to measure the inaccuracy. In the unfortunate circumstance that a simulation’s coherency breaks due to the magnitude of the timing created errors, the problems are more likely temporal disconnects generating non-realistic computations further cyclically generating ever more unrealistic velocity vectors, rather than simple velocity vector disconnects between frames. This makes the metric based on volumes of positional deltas weak, especially since any simulation breaking is already akin to a consistency metric of *infinitely bad*, where feedback and improvement are no longer possible.

Because time duration of inconsistency also affects creating any incorrect velocity vectors, the velocity vectors themselves are not true independent variables to sample when computing

the consistency metric. Thus, determining the collective time durations for inconsistencies may effectively and more efficiently identify the overall consistency of a distributed simulation. When the goal is to eliminate lag effects altogether, measuring perceived severity of lag divergences is not germane—close but demonstrably wrong all the time is not good enough, even though it is the accepted practice in the simulation community. This dissertation challenges that practice.

Events are the creators of state changes and include all the necessary information for forensic evaluation. So, comparing the recorded events for each entity at each host of the distributed network identifies the durations where hosts disagree on the state progression. This metric indicates the aggregate time interval that it is at all possible for a user to perceive any lag effects, thus removing any concept of perception from the measurement. This value may then be divided by the total simulation time each entity is simulated across the entire network to yield a percentage of the total simulated time that the simulation is inconsistent. Moreover, this metric avoids the trap of only working for a single simulation algorithm, because any effects of mitigating and repairing state deltas is reflected by a reduced overall time of inconsistency. Redefining Ω from Equation 2.3 to now be the Ω_i in this derivation:

$$\Omega_i = \begin{cases} 0, & \text{if } \tau < \epsilon, \\ \tau, & \text{if } \tau \geq \epsilon, \end{cases} \quad (2.4)$$

captures the time duration of a single inconsistency, a filter ϵ is applied to reject cases in which the time duration is: 1) so small that it is both below any perceptible limit; and 2) just large enough to reject differences in the deep bits of the binary values that may be created by heterogeneous platform computation differences. For example, ϵ might be set to one nanosecond for this purpose. To capture the sum of all (u) discrepancy durations (Ω_i) for all (v) entities, across all (w) hosts in the distributed simulation requires

$$\Omega_{Total} = \sum_{k=0}^w \sum_{j=0}^v \sum_{i=0}^u \Omega_i, \quad (2.5)$$

and the total of all the runtime durations for each entity in the distributed simulation may be documented with

$$R_{Total} = \sum_{k=0}^w \sum_{j=0}^v (endTime - startTime), \quad (2.6)$$

which allows putting Equations 2.5 and 2.6 together as

$$Inconsistency\% = \frac{100 \cdot \Omega_{Total}}{R_{Total}}, \quad (2.7)$$

to present the overall percentage of the simulation runtime that any entity exhibited state discrepancies between the distributed hosts of the simulation.

2.1.4 The Basic Effects of Lag

In a real-time simulation these compounded effects ensure that hosts across the network never see the same exact total state displayed due to the transmission delays [2]. The user blames lag for the perception of display issues from unexpected temporal jumpiness to artificially delayed responses or seemingly impossible sequences of events. The underlying simulation manifests these cases at best as simple differences in state between hosts, or sometimes as odd simulation artifacts that do not have a logically repeatable or explainable cause and which may lead in the worst case to verifiable causality issues and inconsistent states within the simulations.

When the effect of lag is considered in context of what the simulation is designed to accomplish, whether training testing or gaining insight to a problem, the lag creates execution artifacts and spurious results that are not deterministic within the simulation's design or code. Distributed time delays allow the opportunity for users to make decisions and take actions which are applied on one host, after remote actions happen but before the receipt of the associated remote update. A received "late" update may invalidate or corrupt the locally determined action when considered against the state of the host that transmits the incoming update. The combined effects may spiral combinatorially and may cause participants to question the causality of the progressing simulation or their own decision-making processes—this is obviously bad and contributes to both negative training and incremental loss of trust in simulations in general. If the combined effects are not critical because the

simulation is not concerned with strict correctness, the time delays themselves are effectively randomized due to network jitter. This injected randomness makes it impossible to deterministically replicate a particular distributed simulation session. While it is not strictly “incorrect” for a simulation unconcerned with strict correctness, this is a regrettably accepted if unwanted problem that further adds to an incremental loss of trust in simulations by non-simulation community professionals who are unaware of the technical lag issues and only see a lack of repeatability where they expect repeatability to reasonably exist.

The problem, described as the *Consistency-Throughput Tradeoff* by Singhal and Zyda [1]:

It is impossible to allow dynamic shared state to change frequently and guarantee that all hosts simultaneously access identical versions of that state. [1]

Nine years later, Steed and Oliveira [2] informally referred to the same problem in designing a networked virtual environments (NVEs), referring to it this way:

This is a fundamental property of NVEs and other realtime systems if they run in what is called an asynchronous mode, meaning that they evolve their state without waiting for the network. [2]

The above-described lag phenomena is not confined to networked visual simulations. The sub-field of parallel and distributed analytical simulations is directly affected by lag as well but the term is seldom used in the literature. Instead there is coverage of causality errors due to communication between computational units (a direct result of lag) and how to manage these causality errors via scheduling paradigms such as optimistic vs. conservative scheduling covered in subsection 2.4.2.

2.1.5 Assessment: Why is Lag Still a Research Problem?

Previous literature in the distributed virtual environments community provided proof that a distributed shared state is mathematically unobtainable for a dynamic simulation, Singhal and Zyda [1] termed their proof of this the *Consistency-Throughput Tradeoff*, and the concept is referenced by Steed and Oliveira [2] in their introduction to latency compensation techniques. These solid sources, and the community-wide belief that the speed of

electrons through a wire means latency itself is unavoidable, leads to the result that nobody tried to eliminate lag. The problem with this latency-unavoidability assumption is that lag is not latency as evidenced in Equation 1.1 and further covered in Section 4.2.

One of the latest papers covering distributed shared state coincides with the closing timeline of one of the last large published research efforts into distributed VE, the Distributed Interactive Virtual Environment (DIVE) project. Park *et al.* [11] covered distributed state consistency in their 2004 paper *Dynamic Shared States Management For Distributed Interactive Virtual Environment*, noting the ability to maintain fine-grained shared state within the confines of a local area network (LAN) but unable to do so between non-local LANs, resulting in a course-grained result that they characterized as an illusion on the remote systems. Beyond this granularity-based description of communication limitations between systems, there is no specific discussion of latency and its role in forcing the loss of fidelity from fine-grained to course-grained solutions, but the indicators are there.

While there is no explicit callout of latency as creating an insurmountable problem that sunset the golden age of distributed VE research in the 1990s, a turn-of-the-century shift away from multiple research programs requiring tight latency control to provide acceptable real-time user interaction (but unable to achieve it) occurred. Large volumes of VE gaming-related research and software development replaced the distributed VE academic research programs, and this gaming research tacitly accepted communications latency and its physical effects.

What remained is a sustained research and practical engineering effort into best effort managing consistency of distributed state via lag mitigation approaches designed to minimize adding additional network communications latency and then masking the effects of the inevitable divergent states via post-processing the visual representations.

2.1.6 Approach: Minimizing Latency is Insufficient and Ineffective

The primary latency minimization goal is reducing the needed number of message packets, therefore reducing required bandwidth contention and probability of network layer conflicts requiring re-transmissions. Examples of techniques to achieve this goal include client-prediction contracts (e.g., dead reckoning), relevance filtering, packet bundling and data compression, as listed in a survey by Delaney [12]. These techniques reduce the overall

magnitude of latency but do not eliminate latency or lag. Instead they are used as part of best practices in the design of distributed network simulation systems.

Some simulation approaches minimize latency through physically avoiding it by limiting how far apart the distributed hosts are allowed to be within the hardware supporting the simulation. While this works in certain contexts, it prevents distribution beyond close physical co-location. For example, SPEEDES is touted as being the fastest distributed simulation architecture, but it indirectly advocates a physical avoidance technique, with Steinman [13] stating:

External modules view the parallel simulation as a central controller. The external modules are still event-driven, but they must not communicate too often with the central simulation. Otherwise, the simulation will be bogged down by the large communication latencies. [13]

2.1.7 Lag Compensation—Offers a Mitigation, Not a Solution

Yahn Bernier [9] of *Valve Software* published a 2001 web article, *Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization*, one of the earliest robust descriptions of the latency generating lag problem, and mitigations used within the game *Half-Life*. As per the title, Bernier presents the design challenges latency creates, shifting interchangeably between the terms lag and latency, and then defining *Valve's* solution named lag compensation:

[A] method of normalizing server-side the state of the world for each player as that player's user commands are executed. You may think of lag compensation as taking a step back in time, on the server, and looking at the state of the world at the exact instant that the user performed some action. [9]

The process involves:

The introduction of lag compensation allows for each player to run on his or her own clock with no apparent latency. In this respect, it is important to understand

that certain paradoxes or inconsistencies may occur. . . . For instance, if a highly lagged player shoots at a less lagged player and scores a hit, it may appear to the less lagged player that the lagged player has somehow “shot around a corner.” In this case, the lower lag player may have darted around a corner. But the lagged player is seeing everything in the past. [9]

Bernier’s design conclusion is:

Lag compensation is a tool to ameliorate the effects of latency on today’s action games. The decision of whether to implement such a system rests with the game designer since the decision directly changes the feel of the game. For *Half-Life*, *Team Fortress* and *Counter-Strike*, the benefits of lag compensation easily outweighed the inconsistencies noted above. [9]

Lag compensation techniques are refined and published in countless academic and industry papers; it is still the go-to set of techniques in distributed gaming simulations. A 2020 article on the *Riot Games* website by deWet [14] titled *Peeking Into Valorant’s Netcode* describes *Riot’s* approach to addressing the inconsistencies Bernier identifies and how certain gameplay situations are handled to eliminate advantages that lag may provide to a player experiencing a different magnitude of lag than an opponent. Here it is important to understand that the lag deWet is referring to is the lag from each player to a common central server, not the total lag between players themselves. The discussion is explicit that the entertainment value of perceived fairness is the most important metric to *Riot*:

As with many challenges in software development, building a competitive tactical shooter involves making tradeoffs across many (often competing) design goals. Let’s briefly cover some of our goals that relate to netcode and competitive integrity, and afterwards we’ll take a look at some of the technical challenges we had, and how we resolved them in ways that amplify these goals.

Games are fair.

Movement is smooth and highly responsive.

Gunplay feels rewarding.

Give players holding territory an advantage.
Support a wide range of setups. [14]

Both Bernier in 2001 [9] and deWet in 2020 [14] explicitly state that they are making design decisions based on entertainment value, and they both leave discussions of “reality” as out of the application design scope. They describe the difficulty of designing a simulation system to provide desirable levels of perceived consistency to their players. Both also describe their solutions with respect to frames and frame rate where the frame rate is a discretized measure of time advance. Discretizing via frame rates and time steps in visual simulations is simply the overwhelming implementation choice and not discussed in terms of a design option. These choices only contribute to an ever-more-complicated special-case handling system, that is not compatible with real-time training requirements.

2.1.8 An Alternative to Lag Compensation—But Worse

A 2005 book chapter by Kim et al., *A Global Timestamp-Based Approach to Enhanced Data Consistency and Fairness In Collaborative Virtual Environments* [15] describes and defines the same problem as Bernier from 2001, albeit using academic terminology rather than software engineering prose, and containing no reference(s) to publications on industry engineering approaches. This omission of not utilizing or referencing non-academic publications is not confined to Kim et al.’s article, but instead all of the academic approaches proposed since 2005 have failed to gain any traction, while Bernier’s [9] lag compensation is the de facto common practice in implementing real-time distributed network VEs.

Kim et al.’s proposed solution is a real-time distributed networked VE which replicates the intent of the client/server game protocol used by Bernier, but not attempting to minimize additional communications latency, rather by adding bounded amounts of **additional latency** before posting incoming updates. The effort to gain participant perceived “fairness” based on manipulating a global time stamp by injecting additional delays only makes the problem worse for a user interacting with a simulation in real-time. While correct event ordering may be guaranteed, the extra time delays only make lag worse despite the author’s expending great effort to demonstrate that their solution performed better than a worst-case situation.

The computed lag-compensating time values are presented as absolute time delays labeled application response time (ART). The network distribution is performed across a LAN, which does not adequately represent long-haul transmission delays and yet there are still system results that approached 100milliseconds (ms) of ART (larger than cross-continent magnitude lag). Despite capturing and regularizing the network transmission delays and jitter, the additional application-imposed latency does nothing except terminally frustrate users, even if global order-of-operation fairness is absolutely preserved.

2.1.9 Prediction Layers

The Predictive Real Time Architecture for Distributed Environments (PaRADE) system presented in Roberts' *Maximising Concurrency and Scalability in a Consistent, Causal, Distributed Virtual Reality System, Whilst Minimising the Effect of Network Delays* [16] is a run-time infrastructure (RTI) implementation. It has the design goal of meeting "sufficiently-causal delivery" via "time based reasoning may allow the effects of events to be predetermined, ensuring that they are enacted simultaneously at each federate" [16], PaRADE also includes multiple references to predicting user actions. As published in the article, the implementation of the approach does not work, reportedly due to intermittent delivery of best-effort unreliable multicast User Datagram Protocol (UDP) packets.

Problems were encountered with the update control mechanism. This was manifested in a user having to make several attempts at picking up or hitting the ball before anything happened. This was found to be caused by loss of the request message which is sent via multicast. The system is currently being updated to use the reliable resend facility used for other vital events.

A significant amount of unreliably sent events were lost. The delay estimation program was used to determine network behavior and message loss using unreliable multicast. Message loss was found to vary throughout the day but was found to have a mean of around five percent. One possible cause is that audio was being sent using tunneled multicast between the two sites. The problem is currently being investigated. [16]

Follow-up work is summarized in 2004 by Roberts in *Controlling Consistency within Collaborative Virtual Environments* [17] and summarizes work implemented in two additional

demonstration applications. The article's coverage of its three different consistency control implementations does not make any statements about the success of any of them and closes with a generic conclusion that can be summarized as—achieving consistency is hard:

Much is known about consistency management in analytical simulation but relatively little about the effect of various approaches on human-in-the-loop simulation such as collaborative virtual environments. This paper outlined the principles and described how we applied these within three CVEs. Consistency management is a complex issue and it is not surprising that many CVEs ignore the issues. This, however, holds back the development of more advanced applications. Immersive and especially walk-in and multi-modal displays are showing huge improvements in the realism of collaboration by supporting verbose non-verbal communication. Combining this with shared object interaction is probably today's toughest challenge for consistency control within CVE's. [17]

Progress on the consistency topic seems to have petered out with Otto's *A Review on Effective Closely-Coupled Collaboration Using Immersive CVE's* [18] (with Roberts), which provides yet another overview of consistency challenges and does not provide any working solutions. Previous predictive-consistency research, such as noted in the papers so far cited in this chapter is only referenced in regard to documenting difficulties and user dissatisfaction with the techniques. It appears that attempting to use predictions for VE consistency management has closed without success.

Note that the approaches used are not systemically considered but layered on top of the era's working VE paradigms, adding timestamps and other administrative overhead in an attempt to obtain consistency. There are no references to any analytical simulation research in which the concept of timestamps and causality (although not with respect to interactive real-time simulations) are already in widespread use and consideration. This approach is doomed to failure as the prediction layer is nothing more than an additional computational load that is not even guaranteed to be correct, and the underlying architecture is already crashing against the hard limitations presented in Singhal and Zyda's *Consistency-Throughput Tradeoff*.

2.2 Distributed Simulation and Distributed Computing

The question of whether a distributed simulation is simply yet another distributed computing system is not as simple as it appears. The word “distributed” does appear in both and similarly refers to a geographic separation of computers which are networked together and able to exchange messages. Distributed simulation and distributed computing also operate in a similar environment in that any system which is geographically distributed and connected via the Internet must deal with the same set of network-related challenges, even if the applications require differing implementation details.

Beyond that similarity are the differences in goal, execution, and replication strategies which contributes to a weakening of the premise that the word “distributed” in both domains is enough to consider distributed simulation and distributed computing solutions the same. Since simulation is the behavior of a model(s) over time, distributing the design brings specific requirements to inter-dependent timing considerations and their effects on the maintenance of the overall shared state that is in constant update. This section details the problem space similarities as well as the divergence in execution requirements.

2.2.1 Similarities

Considering the obvious similarity first: distribution implies a geographic separation of computers which are networked together and able to exchange messages. This is a generic paraphrase of multiple distributed systems and algorithms textbook opening chapters [19], [20], [21], [22]. Distributed systems are typically organized as transaction processing systems, and communications and handling of simulation events show a significant overlap with respect to the basic communications requirements of transaction processing systems. There are also distributed systems for using or manipulating persistent objects [23], [24] which from a terminology aspect appears similar to managing the topic of persistent simulation worlds and environments.

Fault tolerance is a common requirement as well, especially with respect to an inherently unreliable communications infrastructure. The requirement for applications to not fail based on transient unreliable delivery and jittery latency timelines is also common. Castro and Liskov’s *Practical Byzantine Fault Tolerance* [25] “presents a new, practical algorithm for state machine replication that tolerates Byzantine faults.” The concept of the distributed

system's Byzantine (i.e., arbitrary) fault as mapped across the need for a consistent shared distributed state encompasses the potential for distributed simulation systems to also produce incorrect results due to failures, bugs, or malicious actions. By simple inspection of the title, *Tolerating Latency in Replicated State Machines Through Client Speculation* by Wester *et al.* [26] (including Liskov) appears as if it might already solve the **Lag Erasure** problem via speculative execution. However, it does not once the details are examined.

2.2.2 Divergence

From the previous section it is clear that distributing a computational system generates a list of problems that is common in terminology and source code whether the application is a classical distributed system or the more specialized form of distributed simulation. The basic problem requirement that distributed simulation is addressing though is different than that covered in the broader base of literature on distributed systems. Once again, the timing delays of inter-dependent simulation events in the overall problem of state coherency is central to the divergence.

A clear theme across distributed system literature is that of a large unitary program being computed by or queried by multiple distributed nodes and that the causality and collective replicated state communications between those distributed nodes is managed towards a single common and correct result. The common and correct result is typically segmented and replicated for distributed storage across the set of systems to reduce systemic data access bottlenecks and ensure fault tolerance against a node in the system becoming unavailable. This concept is common across the applications of:

- distributed databases
- distributed filesystems
- transaction processing systems
- interactions with a central persistent object store

Design choices for how to represent and store the shared state all typically expect changes to produce a momentary update and then a period of stasis for the state data. The data may either be stored in a single location and accessed dynamically by multiple distributed nodes [23], [24] or the data is replicated across a network with a required consensus between each of the replications that hold copies of the replicated state chunks [19], [20], [21], [22],

[26]. These data organizational principles are different from how distributed simulations organize their distributed state. Direct cause-and-effect relationships can be preserved and maintained in a timely manner for non-simulation distributed systems because the cause-and-effect relationship is not a contributor to a dynamically lengthening chain of additional successive cause-and-effect relationships with cyclic dependencies on earlier changes.

2.2.2.1 State is Not the Same in Every Domain

This seems obvious but must be explicitly stated to avoid unintentionally falling into a trap of over-equating too much simply because the problem terminology is similar. A database is a static entity between updates, the distributed database consistency problem is about determining which updates happen in which global order, and algorithms to provide that canon resulting consistency. The mere existence of discrete simulation states does not make the simulation a collection of time-sorted database create, read, update, delete (CRUD) primitive operations. A distributed simulation has near constant updating of entity state data in both local and distributed hosts meeting real-time deadlines, inter-host communication of information that may cause an incorrect local or remote calculation due to incoming superseding information, yet the now-incorrect result may already be in transmit to yet another remote host, or maybe the host that superseded the calculation itself! Thus errors propagate and compound effects in other subsequent errors. All of this exchanged information is used to create a perceptually continuous simulation state to an observer, and that state whether correct or incorrect due to the intercommunication problems, is used to make decisions in the same continuous timeline of the simulation. Should an observer make a decision based on incorrect simulation state compounded by cascading errors there is the potential for the observer to make a catastrophically wrong decision that may affect the real world, especially if the simulation is the data display output of a C2 system used for warfighting.

2.2.2.2 State is Bigger Than a Simple Snapshot Series

By definition state in a simulation *is everything* necessary to evaluate the model through time, often simulation state is not quiescent. Entity state is not primarily its orientation and trajectory, rather the state contains all necessary initial values (variables) and can even refer to a particular system of differential equation(s) to use in evaluation. The simulation state also contains more mundane non-entity specific variable such as maintaining knowledge

of the next pull from a shared pseudo-random random number generator (RNG) and any existing parameters necessary for determining and computing non-player character (NPC) or AI-driven entity actions.

A traditional time stepped simulation computes state updates on a regular timeline, with results being instantaneous approximations of phenomena that are continuous. While the snapshot might infer a similarity to the database state update, a snapshot is not the sole definitive arbiter of simulation state or places where state may change. It is not uncommon to provide special-case handling for events that occur between the computations to more faithfully approximate the **continuous nature of the model being simulated**. Even though there are delays between time-step computations updating some of the state, other simulation computations may be in progress that interact with any particular state snapshot, between (or even during) the state computations and may occur in either a local or remote host.

State is all the data and information required to evaluate and repeat the simulation. If state is different on two participating hosts, two different answers are the expected result; any failure to capture every state component affecting an event during a session makes it impossible to repeat in the scientific sense of repeatability. Non-repeatability invokes Naval Aviation's "same day rules" which is a catch-all for something similar enough because it happens close enough in time but without any expectation for precision-of-result².

2.2.2.3 Continuous State

Management of continuous state within an asynchronous discrete event simulation (ADES) paradigm presents a substantial divergence from the state descriptions and uses in the distributed systems literature. State under this type of paradigm is continuous in time, not a snapshot, and this makes communicating computed state snapshots inadvisable because they are by-definition stale (or wrong.) Rather the ADES paradigm communicates events that adjust the state trajectory, and this means state is not a simple position and orientation. Continuous state management uses the appropriate differential equation(s) and initial conditions to describe how the state, including position and orientation, changes across time.

²"Same day rules" is also a derisive term for formation execution that is so poor that the participating aircraft look like they are independently flying to the same destination. Unfortunately, distributed simulation is trapped in the "same day rules" paradigm of total inability to deterministically repeat replications due to the intractable problem of maintaining and recording state under the fixed time step state snapshot paradigm.

Continuous state management techniques provide a state trajectory which is continuous and fully deterministic across time. This provides a differing set of details and assumptions than classical distributed systems employ. It also invalidates Mauve's [27] conclusions about state, events and determinacy being solved distributed database management applications, and which is cited by Delaney in *On Consistency and Network Latency in distributed interactive applications (DIAs): A Survey—Parts 1 and 2* [12], [28] with respect to the domains of simulation and virtual environments.

2.2.2.4 Distributed Simulation is Not Distributed Database Management

Separate from the overall consideration of whether distributed simulation is a form of solved distributed system is an in-community consideration of simulation equating to having an underlying distributed database. Because this comes from within the simulation community it deserves a deeper treatment. In the DIA survey, Delaney [12] states:

The input from users of a DIA generates events that then modify an underlying database that is shared in some way by all users across a communications network. [12]

This is a dangerously inaccurate statement about both what a distributed simulation is, and constraints on how it must be implemented. Being part of a comprehensive survey may lend outsize weight to an interpretation that is counterproductive to the title topic of the paper. Why? There is an entire field of research and development concerning how to effectively create and manage distributed databases to maintain or create a definitive time-ordered state, and with globally nanosecond-precise Precision Time Protocol (PTP) hardware timing techniques as presented by Meta [29] (more detail in Section 2.3.2.2) with the time-sorting database problem is well solved.

If the simulation as a DIA is a simple distributed database problem and consistency management is a solved issue, the *Consistency-Throughput Tradeoff* [1] is incorrect and does not actually exist; Steed and Oliveira's 2009 textbook *Networked Graphics* [2] would not have agreed with, and cited, Singhal and Zyda, but instead would have discussed the distributed database management techniques for full distributed state consistency management. None of this is the case and more than an additional decade of distributed simulation research and development (a representative cross-section is presented in Section 2.1) remains in agreement that the distributed state consistency problem is not solved for simulation.

Continuous state as discussed in the previous section is a much deeper interpretation of state than presented in the dissertation by Mauve *Distributed Interactive Media* [27] which claimed to solve the distributed state problem for all types of interactive media, including simulations (because Mauve considers them a form of media database.) Yet when implementing a networked Virtual Reality Markup Language (VRML) VE scene between hosts Mauve [27] states “Consistency support is therefore only best-effort: events are injected into the VRML object as soon as they are received.” This means no real-time consistency support or solution is provided for the VE domain. By extension it also indicates the successful consistency support Mauve presents for static multimedia content (which is similar to streaming database content) is a different problem than consistency support for distributed simulations, despite implied claims to the contrary. With Mauve’s paper providing substantial support to Delaney’s quote above, the position that DIAs, which include distributed simulations, are implemented using an underlying database is ungrounded and invalid. The need for simultaneous consistent operations on distributed shared data remains a primary distinguishing characteristic for simulations. Sequential updates are insufficient.

2.2.2.5 Distributed Simulation is Not a Distributed Filesystem

For essentially the same static vs. dynamic reasons as discussed above in Section 2.2.2.4, the distributed filesystem does not map effectively to distributed simulation. A distributed filesystem is less dynamic and more static than a database simply because the external storage systems they directly map are four-to-six orders of magnitude slower than main memory read/write transactions. This deliberate pace is incompatible with implementation of dynamic real-time or faster-than-real-time simulations.

2.2.2.6 Distributed Simulation’s are not Transaction Processing Systems or a Central Persistent Object Store Service

Simulation events have a superficial similarity to system transactions in a distributed system as both events and transactions trigger state updates, require proper time sorting, and each issued interaction seemingly appears as if it originated from independent entities. The transaction processing and object store systems are based on algorithms that allow some portion of the replicated stores backing them to independently fail yet that the overall system remains correctly operable [23], [24], [25]. This concept of independent replications in a distributed system that may fail yet leave the overall series of handled transactions correct,

is dramatically different from a distributed simulation where a host failure categorically removes some portion of the participants and/or environmental components from the larger simulation, dooming the simulation to significant degradation or even a failed session—there is no possible technique to recover to a fully functional and corrected simulation.

Figure 2.1 illustrates a close coupling in the state synchronization of the individual distributed hosts (P1-P6) throughout the network.

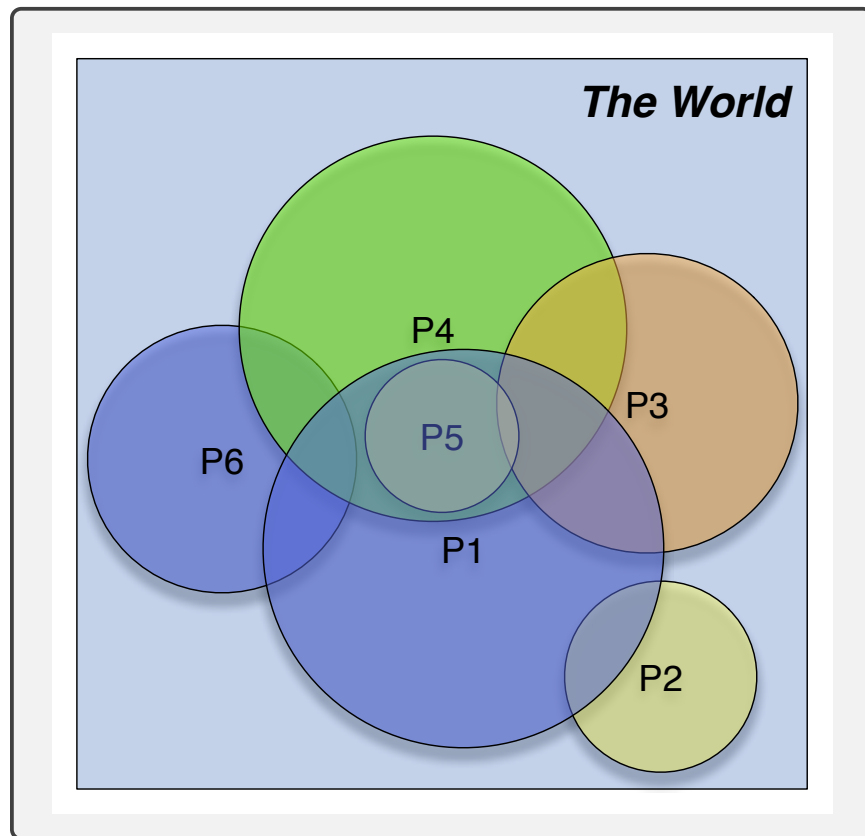


Figure 2.1. Interlocking state maintenance requirements reflecting the logical interdependence in distributed simulation.

When external participants are involved, they necessarily interact via the one host where they maintain their physical I/O connection, and the states they interact with may also be affected by multiple other external entities on other hosts. This interdependence means that failures at one of the distributed hosts inevitably cascade to affect states that other hosts

require, creating a large and unfixable differences (and inconsistent contradictions) in the overall simulation outcome.

While such a result is not acceptable in non-simulation distributed systems and are the reason why multiple replication-based fault tolerance algorithms and protocols exist, the concept of a single flight crew simultaneously manning two or more full-motion flight simulators participating in a LVC simulation as an effort to gain redundancy is physically silly and indicates that a generalized redundancy solution for distributed simulation is infeasible.

2.2.2.7 Speculative Execution in Distributed Systems, Similar Yet Different

Tolerating Latency in Replicated State Machines Through Client Speculation by Wester *et al.* [26] (including Liskov) implements speculative execution to reduce the latency penalties that are inherent in a distributed system. At first blush this title reads as if it solves the problem that **Lag Erasure** is presented to solve. The difference though is in distributed systems use of replicated state machines and the algorithmic latency added to the problem in the consensus management scheme [26]. Paragraph 2.1 of the paper states:

Speculative execution is a general latency-hiding technique. Rather than wait for the result of a slow operation, a computer system may instead predict the outcome of that operation, checkpoint its state, and speculatively execute further operations using the predicted result. If the speculation is correct, the checkpoint is committed and discarded. If the speculation is incorrect, it is aborted, and the system rolls back its state to the checkpoint and re-executes further operations using the correct result.

In general, speculative execution is beneficial only if the time to checkpoint state is less than the time to perform the operation that generates the result. Further, the outcome of that operation must be predictable. Incorrect speculations waste resources since all work that depends on a mis-predicted result is thrown away. This waste lowers throughput, especially when multiple entities are participating in a distributed system, since the system might have been able to service other entities in lieu of doing work for the incorrect speculation. Thus, the decision of whether or not to speculate on the result of an operation often boils down to determining which operations will be slow and which slow operations have predictable results. [26]

The result of the slow operation that Wester *et al.* avoid is the delay in generating the replicated state consensus inherent to a Byzantine Fault Tolerance protocol evidenced by:

[T]he maximum latency that may be hidden by speculative execution, in the absence of 0-reply speculation, is the time between when the client receives the first reply from any replica and when the client receives enough replies to determine the consensus response, a protocol should be designed to get the first reply to the client as quickly as possible. [26]

Distributed simulations are not typically concerned with creating this consensus-based algorithmic latency. Instead the timeline of concern is much tighter as the transmission latency of the initial message is the dominating factor. Use of checkpoints in the paper's solution is also something not applicable to distributed simulation. How maintaining explicitly stored checkpoints is not required and the necessary rollback accomplished for an optimistic distributed simulation is covered below in Section 2.4.2. The checkpointing and rollback differences revolve around operational implementation requirements due to each domain's different operational requirements, rather than differing generic network performance requirements.

Another significant difference is that the algorithm of Wester *et al.* [26] accounts for differing latencies and procedures between read-only and concurrent write operations on the distributed data. In a distributed simulation, *event updates are all concurrent write operations* which constitute a “modifying request in an agreement protocol” [26] in the paper's parlance. The solution management needed for these modifying requests that a “primary [server] should be located near the most active clients in a system to reduce their [replicated state consensus] latency” [26]. However, for the typical distributed simulation most if not all distributed hosts include one or more external participants and thus it is not possible to adjust geographic distributions and provide a geographically centralized server to reduce latency. Worse, a central server increases the latency a distributed simulation endures because at a minimum it adds an additional receive-transmit cycle of the server into the network route. This approach also constitutes a partial addition of lag compensation techniques as discussed in Section 2.1.7, and lag compensation does nothing to solve the correctness and/or causality issues presented by latency in communications.

The high sensitivity to lag affecting state correctness in distributed simulations differs from the overall throughput sensitivity affecting state correctness that the other forms of distributed systems are primarily concerned with. This differing sensitivity, lag vs. throughput, is what drives the differing solution for these domains despite using similar terminology and facing the same reliability problems of the Internet. The overlaps and differences between *Tolerating Latency in Replicated State Machines Through Client Speculation* by Wester *et al.* [26] (including Liskov) and speculative execution as used in this dissertation, illustrate such value explicitly.

This analysis makes it evident that despite the same environmental and systemic unreliability risks, distributed simulation and other distributed systems have different operating requirements and characteristics that necessitate different mitigations and handling of those risks. The divergences do not imply that the different application domains need to ignore each other's foundational and developmental knowledge bases, since understanding what considerations apply where and why is immensely useful. Understanding the tandem yet distinctly different solutions to the same problem but different sub-domain lends confidence that both the problem and solutions are coherent and provide a wider ready-illuminated base for future work to progress from in either domain.

2.3 Time Considerations

“A computer simulation is a computation that models the behavior of some real or imagined system over time”—Fujimoto [30]. This makes understanding and control of time-based interactions central to the entire concept of making a simulation. **The preeminent role of time control is in maintaining causality between simulated events.** Understanding that this preeminent role is **separate from and supported by** hardware-provided measures of time and/or choices of how to compute model advances across time is necessary to making advantageous choices for how to compute the models that users are interested in, over time.

2.3.1 Virtual Time

Here lies the baseline concept for time control in maintaining causality between simulated events, **a single timeline upon which the entire simulation is mapped.** Virtual time is described in Jefferson's [31] seminal paper *Virtual Time*:

Virtual time provides a flexible abstraction of real time in much the same way that virtual memory provides an abstraction of real memory.

The virtual time paradigm is a method of organizing distributed systems by imposing on them a temporal coordinate system more computationally meaningful than real time and defining all user-visible notions of synchronization and timing in terms of it. [31]

Jefferson [31] uses virtual time within the *Time Warp* mechanism to manage distributed simulations, and their virtual time event ordering is described as an inverse to Lamport's [32] 1978 distributed systems algorithm, which described partial and total orderings for concurrently executing events. Jefferson [31] states:

Virtual time systems are subject to two fundamental semantic rules:

- Rule 1. The virtual send time of each message must be less than its virtual receive time.
- Rule 2. The virtual time of each event in a process must be less than the virtual time of the next event at that process.

[i]n extremely simple terms: If an event A causes event B, then the execution of A and B must be scheduled in real time so that A is completed before B starts.

These rules are exactly Lamport's Clock Conditions ([32]), and embody our (Jefferson's) desire that the arrow of causality, or the direction of information transfer, always be pointed in the direction of increasing virtual time.

With virtual time we (Jefferson) do the reverse: we assume that every event is labeled with a clock value from a totally ordered virtual time scale in a manner obeying Lamport's Clock Conditions (Rules 1 and 2), and we show how to unfold a fast concurrent execution (i.e., a wide and shallow partial ordering that is consistent with the total ordering). [31]

The approach used in this dissertation uses virtual time similarly as described above, but defines an alternate mechanism to the *Time Warp* mechanism for partitioning and interacting with the virtual timeline. See Section 4.3 for details.

2.3.2 Hardware Clock Synchronization

Hardware clock synchronization is a challenge. If the physical clocks of the participating machines are sufficiently out of synch, no technique to manage a virtual clock timeline for a simulation is effective.

2.3.2.1 Network Time Protocol (NTP)

NTP [33] is one central standard which has multiple independent implementations. For the purposes of this dissertation, choice of implementation and specific options presented are not important, while the expected limits of synchronization are germane. Per RFC5905:

Typical secondary servers and clients on fast LANs are within a few hundred microseconds with poll intervals up to 1024 seconds, which was the maximum with NTPv3. With NTPv4, servers and clients are precise within a few tens of milliseconds with poll intervals up to 36 hours. [33]

This provides a practical expectation limit on differences between the hardware clocks of the distributed simulation participants at “tens of milliseconds” [33] on the upper-bound, which is on-the-order of expected regional Internet ping latencies, and where the distribution is geographically constrained, or within a LAN, reduces towards negligible (“within . . . microseconds” [33]). The most important takeaway is that all participants need to subscribe to a common time synchronization protocol. After years of deployment and development NTP is effective, efficient and currently ubiquitous.

2.3.2.2 Precision Time Protocol (PTP)

PTP [34] is a more-targeted standard, designed for distribution across a geographically limited scope, that is in local subnets.

The protocol will support system-wide synchronization accuracy in the sub-microsecond range with minimal network and local clock computing resources. [34]

The PTP standard appears to be more useful than its intended scope though. An online blog, *Engineering at Meta* [29] posted an entry on Meta’s worldwide implementation of PTP across their data centers, not only locally to a single datacenter. They used time signals

from global navigation satellite systems (GNSS) constellations as the means to ensure the distributed data centers received times and offsets that are deterministically computable, which could then be propagated through their local subnets by the PTP protocol itself. This provides Meta with sub-microsecond precision timing and computable windows of uncertainty (WOU) used as the upper-bound offsets in generating timestamps [29].

While the effort of entry is currently high for a PTP installation, the effort is a datacenter infrastructure level issue, and as more applications have demand for precision timing, datacenter infrastructure providers may begin to provide it. The benefit of this level of precision timing to distributed simulations is that the additional precision removes variation and ambiguity to any operation that references a hardware clock. While the architecture presented in this dissertation uses virtual time rather than hardware time, the hardware clock is necessary for determining relative offsets to generate new virtual event times whenever an externally generated human or hardware-in-the-loop input is injected into the simulation. Additional fidelity and precision in generating virtual timestamps reduces hardware variability and jitter in the hardware contributions to communications latency across the network-distributed simulation. This additional precision is not required for proper operation, but if available it helps reduce lag both absolutely on the hardware side of the Lag Equation (eq. 1.1) and in the algorithmic component as there is less magnitude in hardware variability to include in the compensation.

2.3.3 Simulation Software Distributed Time Management

Time management affects nearly every decision in constructing a distributed simulation architecture. Distributed communications standards address advancement of time in excruciating detail and with multiple special cases because the standards delineate how drastically different simulation methodologies must interact to create a correct heterogeneous simulation system of systems. Time management in these heterogeneous simulation systems of systems is complex. For example the High Level Architecture (HLA) incorporates multiple methods to handle mixes of differing time advancement models in potential federate members, per Fujimoto [35]:

- federates with different event ordering requirements, e.g., Distributed Interactive Simulation (DIS) and Aggregate Level Simulation Protocol (ALSP) federates.

- federates using different time flow mechanisms, e.g., time stepped and event scheduled mechanisms.
- real-time (or scaled real-time) and as-fast-as-possible simulations; it is assumed that individual federates executing in conjunction with the RTI perform at least as fast as scaled wall clock time in federations requiring (scaled) real-time execution.
- federates executing on parallel/distributed platforms using conservative or optimistic synchronization.
- federates using a mixture of event ordering and transportation services, e.g., a DIS-like federate can use time stamp ordering and reliable message delivery for certain types of events, and receive-ordered, best-effort delivery for others. This facilitates gradual, evolutionary exploitation of previously unused HLA time management services. [35]

These are all valid use-case requirements for building a large heterogeneous simulation system of systems, which is beyond the scope of this dissertation. Integrating this dissertation's described system architecture into a heterogeneous distributed system constitutes future work. The scope of this dissertation is limited to demonstrating latency hiding to support a consistent distributed state within a single homogeneous distributed simulation.

2.3.4 Time Advance

The technique for advancing time in a simulation is a central and profound choice in turning the static model into a live simulation. The choice of time advancement mechanisms begins the chain of assumptions and necessary implementation details, and affects the decision space for every other follow-on implementation decision.

2.3.4.1 Fixed Increment Time Advance (Traditional Time Steps)

Synchronous time step simulations are formally described as fixed-increment time advance discrete event simulations (DESSs) by Law and Kelton [36], see Figure 2.2, and they are a special case of next-event time advance DES described in subsection 2.3.4.2. Synchronous time advance weakly reflects that **simulations evaluate modeled events** and in fixed-increment time advance simulations these events are accumulated if they would have happened between steps and then be evaluated synchronously.

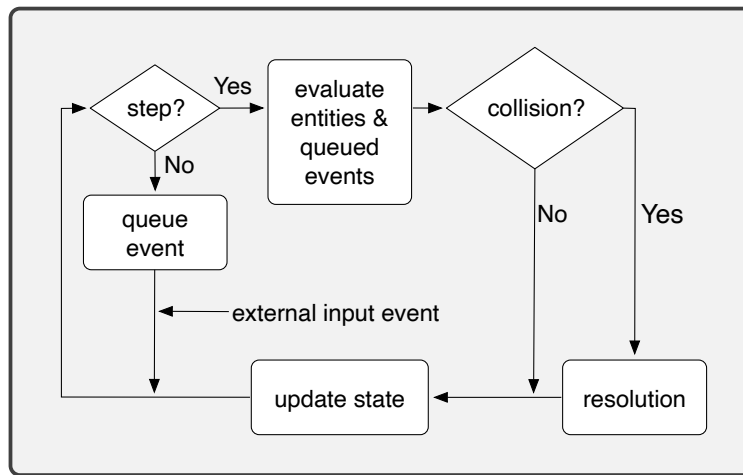


Figure 2.2. Simplified synchronous time advance simulation loop.

Accumulating events, managing order in the face of interdependencies between the accumulated events and resolution, as well as dealing with special case situations between snapshots where state does not explicitly exist (e.g., between the dots of Figure 2.3) obviously incurs overhead and injects potential artificialities which in turn must be managed.

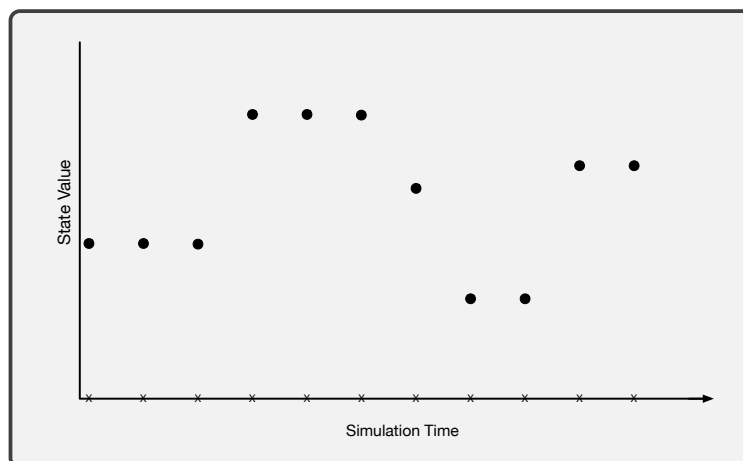


Figure 2.3. A traditional time step state-snapshot trajectory. Source: [37].

Time stepped simulation is the overwhelmingly implemented choice for simulations with a visualization component, or for evaluating fields of ordinary differential equations (ODEs). The literature does not delve into the practicalities and limitations of traditional time step implementations, and books on physically based simulation design such as those by Eberly

[38] and Millington [39] use synchronous time steps without further mention apparently because the concept of Δt is embedded in numerical methods for evaluating ODEs.

The uniformity of the synchronous time step supports code creation that is optimized for high performance evaluation of the simplest acceptable numerical evaluation methods. This further supports a goal of leveraging the *Locality Principle* (per Denning [40]) for the code as well as for the data the code manipulates. Locality supports acceleration through leveraging hardware caching algorithms minimizing the number and magnitude of time delays incurred by interactions with data all the way out to main memory. This performance-first goal drives a requirement to mitigate multiple follow-on effects.

Round-off and truncation error is always present in binary digital evaluations of equations derived for real numbers. The more chained calculations occur, the more the combined error grows, per Hamming [41]. Compensation requires renormalizing, requiring significant time that may create visual artifacts if the required normalizations are sufficiently large. Not normalizing runs the risk of a simulation “blowing up” from numerical instability.

Collision detection implementations are profoundly affected by time step length, and states that are snapshots. Artifacts such as interpenetrations or totally missed collisions (tunneling) may occur. Tunneling is illustrated with baseball batting scenario illustrated in Figure 2.4.

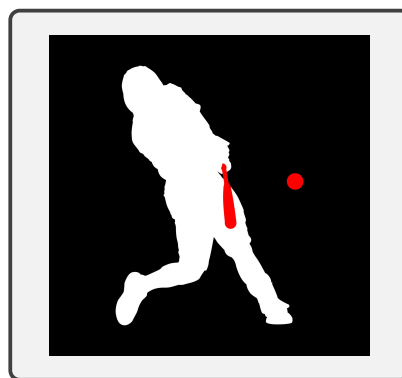


Figure 2.4. Moments before bat-ball impact. Source: [42].

The specifics of the bat-ball impact timeline and potential for tunneling are illustrated in Figure 2.5. According to MLB.com [43] the average bat speed is 72mph, and a top-tier reliever throws at 100mph; the ball maximum circumference is 9.25” and bats have a

maximum diameter of 2.61" [44]. Figure 2.5 shifts to metric for easier computations and comparison, it shows a 78% chance for tunneling (i.e., not detecting the collision) in the interaction.

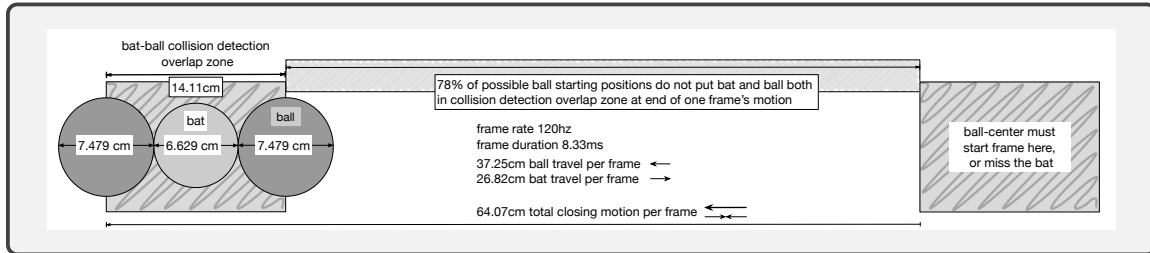


Figure 2.5. Isolation on the bat-ball interaction over one frame.

Interpenetrations are allowed to occur by design and are managed by detecting and retroactively determining when contact should have occurred, then applying approximated corrections. That works well enough when the corrections do not cause follow-on collision issues, or when the non-physical approximated post-adjustment corrections to an object's position and motion do not also contribute to discrepancies in the energy state of objects, (which is another reason renormalization may be necessary.) Algorithms for continuous collision detection (continuous collision detection (CCD)) do exist which are more expensive computationally, avoiding misses and interpenetrations on initial collisions, but depending on the specific algorithm implemented, may still generate follow-on misses as (or after) the first collision is resolved [45].

Interactions between physics engine time steps and visualization frames (ticks) require further coordination in the simulation engine to ensure consistent physics results are displayed in the visualization. There are techniques for managing true fixed-synchronous physics engine time steps independent of the frame rate, but these cyclically generate spurious and undesirable double computation steps for some visualization frames [14]. There are also techniques that use dynamic length synchronous physics engine time steps exactly matched to the visualization frame rate [14]. Notice that the visualization frame rate is referenced in both techniques. Either way, samples of the computation are generated at each step and then some number of the samples are used by the visualizer. To minimize artifacts in this process, the physics time step rate is often tuned to be an integer multiple of the visualization frame

rate. This indicates visualizations and physics are not entirely decoupled, and progression of the interdependent components are subject to common hardware scheduling effects.

As local computation and rendering capabilities continue to improve, such schemes are increasingly effective within a single simulation host. However the techniques do not adequately support the need of consistent physical state across a distributed set of simulation hosts because they do not provide lag reduction, and (perhaps even worse) the short synchronous time steps are the final contributing factor to why the *Consistency-Throughput Tradeoff* is believed to be unavoidable, locking network lag in from the beginning of the distributed simulation's design.

2.3.4.2 Asynchronous Time Advance (Discrete Event Simulation)

Asynchronous time step simulations are formally described as next-event time advance DESs by Law and Kelton [36] and the typical progression of events is depicted in Figure 2.6. They directly honor the stipulation that **simulations evaluate modeled events** and primarily differ from synchronous time step simulations in that the *events* are when the state of an entity change.

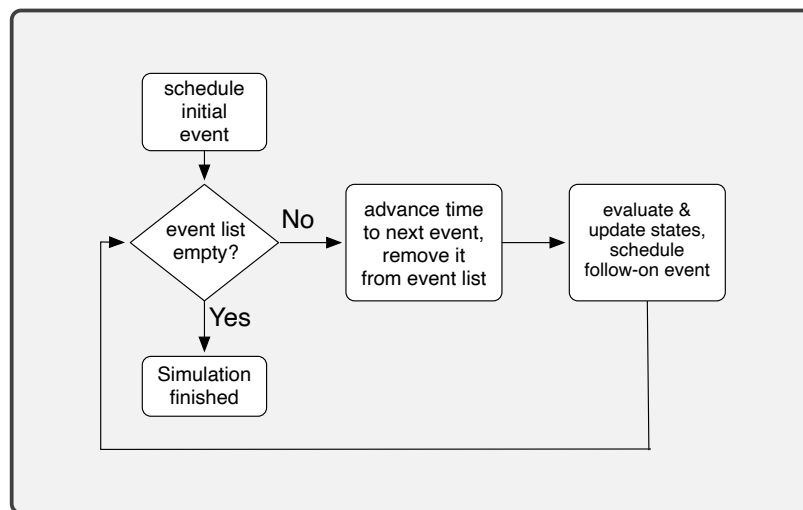


Figure 2.6. The next-event algorithm. Adapted from: [37].

Entity state exists continuously along the simulation timeline, although not continuously differentiable, in a step function as depicted in Figure 2.7.

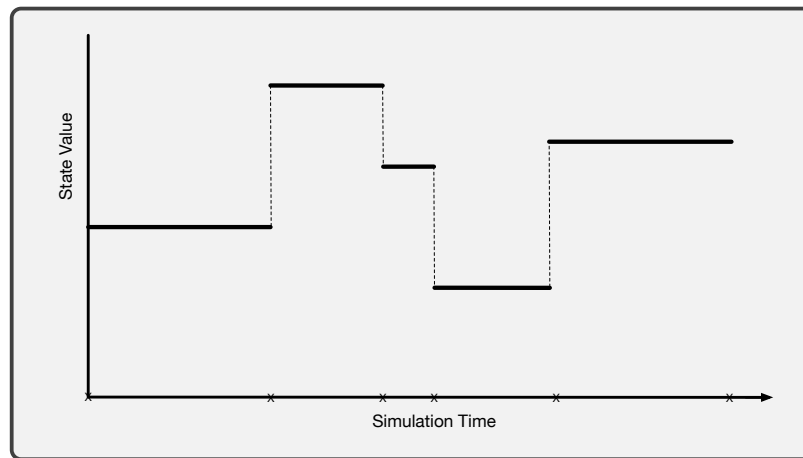


Figure 2.7. A Typical DES state trajectory. Source: [37].

This state is not simply position and orientation of an entity. All varying inputs to simulation evaluations are included, making Figure 2.7's state abstract and of high dimensionality. Because state exists continuously along the horizontal timeline, ADES does not suffer from the same engineering challenges surrounding state (snapshots and its lack of defined state between snapshots) as do synchronous time step simulations.

The use of an ADES methodology reduces the computational load in simulations of small electromagnetic systems and single-object interactions as described by Kofman [46]. This property is attractive when considering large-scale physically based simulations. Reduction of computational load may be used to scale up the number of entities in a simulation or to enhance the fidelity of a simulation while leaving the number of entities constant [46]. The magnitude of the computational load reduction per object depends on the time advance intervals achievable, and algorithms are used for managing the collision detection problem.

Computationally stable time advance is a design focus of numerical methods and families of adaptive time step algorithms in order to provide a mathematically sound method that can generate the maximum-length advances for a desired absolute and relative error profile, as described in Hamming [41], and with provided implementation examples in the seminal *Numerical Recipes* [47]. On a per-single-step comparison basis, adaptive time step algorithm are more computationally expensive than a single step of a fixed step ODE algorithm, but the tradeoff is elimination of large numbers of simple ODE computations, and a side-benefit of reduced error accumulation chains. Collision detection is a separate difficult challenge

as the published works center around techniques resting on the assumptions contained in synchronous time step simulation implementations. Of primary concern there may be a loss of collision detection data locality as collision detection is not synchronously evaluated across an entire array of entities in a simulation. Managing this tradeoff is one of the goals in this dissertation's presented architecture and is covered in Chapter 4.

The formal definition of what constitutes DES *state* is clear but often broader than the simulation community has treated it. Law and Kelton [36] write that state “quantities will generally be a random variable.” The definition seems straightforward, but there is unrealized potential in that definition. The name of a function is an identifier, during compilation or interpretation the identifier's name is assigned to a memory location, and that memory location's contents may be redefined or changed at runtime. Therefore it is a *variable* which contains the memory address (integer value) for the zeroth instruction of the function. This functional interpretation adds dramatic flexibility to the concept of state variables as it is now possible to go beyond single-value representations and refer to a specific function as part of the state. This allows taking the initial values from traditionally considered state, and then evaluating the state-referenced function, yielding a new functionally-continuous variable value that truly varies at runtime, but in a fully deterministic, predictable and repeatable manner. When used in the context of motion evaluation, such an interpretation of entity state provides a true continuous function trajectory for an entity through space-time.

This definition differs from a weakly-defined-state implementation that takes initial value state and then simply routes that initial value to some library function for evaluation. Any non-state memory variables accessed from outside the library function's scope constitute “invisible” and untracked state that makes a code base more special-case heavy and brittle. Reasoning about state also becomes simpler with the removal of invisible state.

The state trajectory of figure 2.7 fully describes the entity state which unambiguously provides a continuous state trajectory, which also explicitly contains the computed continuous four dimensional (4D) (space-time) entity motion trajectory. This resultant computed trajectory is continuous and differentiable within its defined interval, and this may be leveraged by the collision detection subsystem. Differentiability over the state interval is guaranteed with use of second order evaluation. In cases where a sudden non-differentiable change in acceleration (a third order motion derivative is also called a jerk) occurs, the motion interval

is terminated and a new deterministic interval begins at the jerk-point. The endpoints are differentiable within their own interval even when the transition from one interval to the next is not. As the computed trajectory is mathematically continuous, the collision detection tunneling and interpenetration difficulties described in subsection 2.3.4.1, as well any as extra overhead of CCD are trivially eliminated (see Sections 4.4.2 and 4.4.4.3.1.)

2.3.4.3 The Event-Graph Formalization

Event-Graphs are directed graphs with nodes and edges, but they differ from typical graph treatments as Event-Graphs do not represent connected communicating networks or data relationships, and Event-Graph edges are not weighted costs used for determining efficient routes to transit the network. Event-Graphs, are initially introduced by Schruben [48] in 1988 and refined by others [49], [50], then summarized in the 2000 textbook *Simulation Modeling and Analysis* by Law and Kelton [36] as an accepted generalized practice in discrete event simulation. Law and Kelton also reference a 1996 tutorial by Buss [51], which is the specific Event-Graph methodology used in this dissertation.

The specifics of the Event-Graph formalization begins with the fundamental Event-Graph depicted in Figure 2.8 which shows an Event-A conditionally scheduling Event-B t seconds following the start of Event-A if condition (i) is true. The edges of an Event-Graph are always directional, and the presence of a reverse-ess indicates the edge is conditional on condition (i). Scheduling edges indicate that the target event is added to an EventList with a `startTime` exactly t seconds following the `startTime` of the scheduling Event-A.

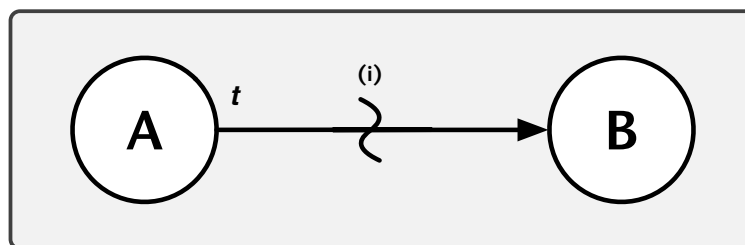


Figure 2.8. A fundamental Event-Graph. Source: [37].

Similarly, Event-A may cause some already scheduled Event-B to be conditionally superseded as depicted in figure 2.9. The dashed directed edge is the cancelling edge and indicates

that the target Event-B is to be removed from the EventList preventing it from evaluating. Conditionals work as they do with regular scheduling edges. There are no scheduling times associated with a cancelling edge, that would only serve to make possible special failure cases where Event-B executes before the cancellation code evaluates.

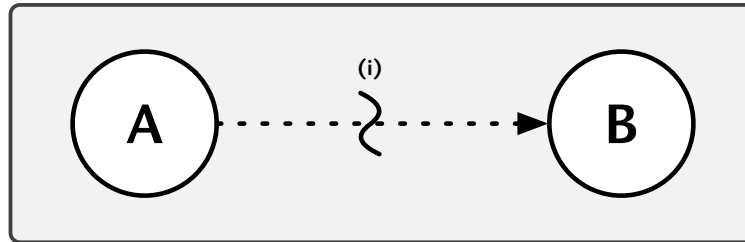


Figure 2.9. A conditional cancelling edge. Source: [37].

The fundamental Event-Graph may also be extended as illustrated in Figure 2.10.

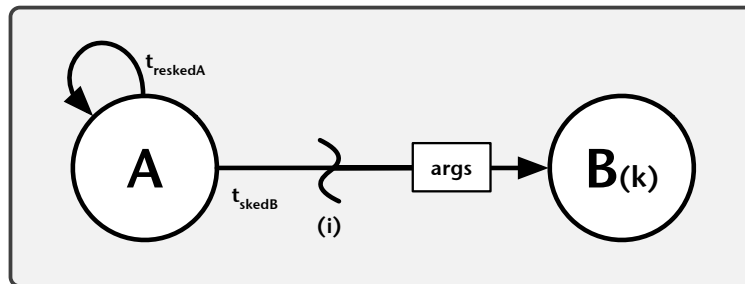


Figure 2.10. Iteration and passing arguments. Adapted from: [37].

1. Self-scheduling events where Event-A reschedules itself at $t_{reskedA}$ seconds after Event-A's startTime are a special case of the fundamental Event-Graph. The rescheduling interval $t_{reskedA}$ is not constrained to be fixed for all iterations of the self-scheduling loop, the evaluation of Event-A may also compute the value of $t_{reskedA}$ to be used for rescheduling. Using adaptive-step numeric algorithms to determine the appropriate maximum event length to assign $t_{reskedA}$ is a precise methodology that only depends on the entity state available to Event-A to generate deterministic yet variable rescheduling intervals.

2. The box attached to the scheduling edge indicates arguments to be passed to Event-B, and this modality is reinforced by the optional representation Event-B(k).

Other valid Event-Graph elements are covered in detail in Buss [37], the elements presented here represent a Turing-Complete computational representation as covered in Section 2.3.4.3.1 and thus can graphically describe any required simulation event sequence.

2.3.4.3.1 Event-Graphs are Turing Complete Of note is that the Event-Graph methodology represents the basic components to (graphically) describe the basic elements of modern structured computation: *sequence*, *selection* and *iteration*, as described in Böhm and Jacopini [52]. Böhm and Jacopini describe relationships in terms of computational flow diagrams, and this computational flow representation has functional overlaps with Event-Graph depictions. Of interest is that this interpretation may simplify the organization and understanding of Event-Graph formalities, compared to the elaborate and complex formalization description by Som *et al.* [50] and the Event-Graph's ability to represent any simulate-able systems. Turing completeness of the Event-Graph formalization is presented in *On the Generality of Event-Graph Models* by Savage, Schruben and Yücesan [53] and indicates there are no computational limits that make Event-Graphs unsuitable for describing simulation models. Figure 2.11 illustrates these structured computation elements.

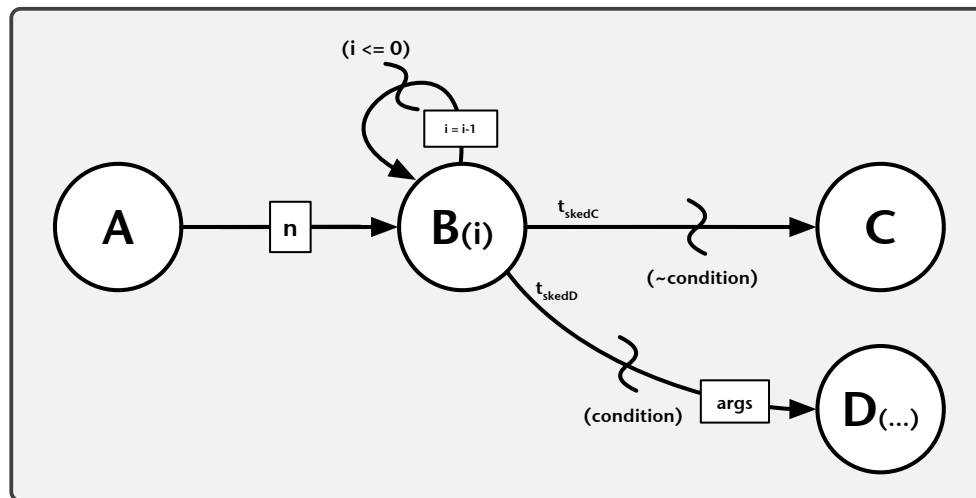


Figure 2.11. Sequence, selection and iteration within an Event-Graph.

In Figure 2.11 **sequence** is modeled between Event-A and Event-B, **selection** is modeled between Event-B and Event-C or Event-D, and **iteration** modeled over the self-scheduling edge of Event-B. The iteration structure may also be extended across a multi-event cycle using similar controls via argument passing (indicated in the boxes on scheduling edges) and conditional execution (indicated with the reverse-esses on scheduling edges.)

This basic set of scheduling capabilities, plus the ability to indicate cancellation of an event using cancelling edges (per Figure 2.9) is powerful enough to represent any computable event sequence model. Providing the opposite claim in their paper, Kozen and Tseng *The Böhm-Jacopini Theorem Is False, Propositionally* [54], the opposing claim is based on restating Böhm and Jacopini's proof using propositional calculus and thus eliminating auxiliary variables. By the end of the paper Kozen and Tseng provide an alternate propositional calculus compliant formulation of all programs using generic loops and structured trees, instead of Böhm and Jacopini's while-loop derivations. These differences derive from the particular expression favored by Kozen and Tseng avoiding the underlying reality that both papers validate a structured programming approach that contains defined forms of **sequence**, **selection** and **iteration**. From a simulation pragmatics standpoint, these differences are uninteresting and Kozen and Tseng's superficial and niche claim of falsity is no more than peripheral semantics.

2.3.4.3.2 What Event-Graphs Are Not Event-Graphs are not Petri Nets as they do not describe *places* or *locations* of transitions across a distributed system and there is no concept of input token at the transition locations. While a Petri Net is used to specify a distributed system design, the Event-Graph itself is silent with respect to distribution. The topic of this dissertation is not about partitioning and distributing the Event-Graph itself, so this research does not lend itself to the expression via a Petri Net.

Event-Graphs are conceptually similar in some respects to Harel State Charts but differ in that Event-Graphs are not intended to graphically enumerate the potential states the computation could enter into and the associated transitions. Harel State Charts treat the state as the thing being modeled whereas a simulation uses the Event-Graphs to depict a sequence of events as the thing that is modeled changes over time, and the state is updated as required. This event as the first-class object, rather than just the transition between

specified states, represents a different representational modality between Harel State Charts and Event-Graphs.

Other finite state automata (FSA) graphs hold superficial visual similarities to Event-Graphs, but in practice are accomplishing different things. The FSA class of graphs attempt to fully enumerate states and the transitions between them, and most FSA do not accept external inputs into the transition rules. A Mealey Machine FSA does accept inputs in its transition rules, but it still holds the goal of enumerating all the potential states explicitly, something which is not the goal of Event-Graphs.

2.3.4.4 Atemporal Models and Hybrid Time Advance

Atemporal simulation models are evaluated without explicit reference to a specific time in a live simulation. For example, a high-fidelity physics model of the sensor envelope created by a ground-based radar system may be generated independently of the running simulation that eventually incorporates the model, which the masters thesis by Tan [55] illustrates, or similarly CFD may be used to generate lookup tables of flight performance data. Offline computed (atemporal) data accessed by a running simulation provides significant runtime computational savings. Any combination of atemporal models and live runtimes defines a hybrid time advance simulation as described in Cheng's 2016 dissertation *Hybrid High Fidelity Modeling of Radar Scenarios Using Atemporal, Discrete Event, and Time Step Simulation* [56].

2.3.4.5 Asynchronous Time Advance in Physics-Based Simulations

In 2000, Mirtich's *Time Warp Rigid Body Simulation* [7] explored directly mapping physics engine calculations into a *Time Warp* simulation. This included close adherence to the logical process (LP) abstraction, which expanded the synchronization problem per the reasoning previously covered in Section 2.4. Mirtich also has challenges with event length determination as the physics implementation he used could not provide precise timing for discontinuities such as collisions, between the selected step sizes.

Lack of precise timing for discontinuities appears to stem from mapping adaptive length events on top of a collision detection implementation originally designed for fixed step computation. The hybridization works but presents extra special case handling requirements.

An aspect where special case handling need is most apparent is in the generation of contact groups between objects commencing in-contact with each other [7]. Fixed step methods require this form of tracking so that all member's contributions to the group may be computed and incorporated in how the group's members should progress over the next step interval. This is different from an event-first methodology where contact groups are not required, rather a train of events follows impulses such as described by velocity-based propagation per Erleben [57], between in-contact objects which may be properly sequenced as individual events.

Mirtich's [7] collision detection broad phase is managed using dynamically sized axis-aligned bounding boxes and data structures sorted for overlaps, a standard methodology for fixed-step collision detection. This also contributes collision discontinuities as mentioned in the above paragraph. Mirtich concludes the approach shows promise and that next steps include parallel rigid body simulation, but published follow-on work appears to be scant and where it does appear, follows both Mirtich and Jefferson closely, such as Koenig [58] in 2013, and does not appear to address any of the challenges Mirtich identified nor improve the overall algorithmic designs, adoption, or lead to significant additional research.

2.3.4.6 The True Price of Traditional Time Step vs. Asynchronous Advance

Time and space are continuous at the perceptual level and scale of a human. Motion of objects through time and space is continuous at these scales as well. Arbitrary discretization of a continuous problem may all too easily lead to special case anomalies where most of the special cases are self-inflicted and unnecessary. Hamming [41] dives deeply into the considerations of algorithm and sampling frequencies for determining step sizing when simulating an approximation of a continuous system. Mentions of formalisms or requirements with respect to visual or analytical simulation time advance step sizes are otherwise scant in the distributed simulation literature.

Nevertheless, this scant coverage does include two dissertations that explicitly tackle numerical analysis requirements of novel special-purpose differential equation implementations designed for parallel computation: a 2003 dissertation by Nutaro, *Parallel Discrete Event Simulation with Application to Continuous Systems* [59] and a 2003 dissertation by Kofman, *Discrete Event Based Simulation and Control of Continuous Systems* [46]. Nutaro and Kof-

man's works both center around a specific simulation framework, the Discrete Event System Specification (DEVS) and address local parallelization only, not network distributed parallelization, and are targeted for general purpose parallel discrete event simulation (PDES) systems.

The work of this dissertation is not attempting to design any new numerical method, but applies already settled methods, and as previously stated this work is targeted to human-scale interactive simulations. Nutaro and Kofman's DEVs specific differential equation design work is beyond the scope of what is required for further discussion, but numerical analysis itself remains on the table. It is possible that the component-wise derivations of Nutaro and Kofman are over-engineered and unnecessarily generate shorter than optimal advancement intervals. Rather it is possible to use vectors and settled adaptive-step numeric methods to compute over the continuous domain, and more simply avoid the problems of fixed sampling frequencies causing computational mismatches while simulating an approximation of a continuous system (per warnings provided in Hamming [41].)

Unfortunately, the lack of formal consideration on time step sampling frequency in simulation algorithms results in the simulation community ignoring the issue, and instead choosing time step lengths through the practical engineering tradeoff metric of available computational resources and usage per time step. Consideration of the dependencies between the simulation and visualization step-rates inform per step function-point computational budgets and reduced fidelity differential equation choices, not ensuring numerical accuracy. The fixed-rate simulation steps are the norm, with display rates often but not always related to an integer multiple of the simulation steps (e.g., $N * \text{simulationSteps} == \text{oneDisplayStep}$).

Peeking Into Valorant's Netcode [14] details the logical special-case deep-dive through a time-stepped simulation design at 128Hz, decoupled from client visualization frame rates. The resulting cascade of choices follows the classical pattern of the children's book *If You Give a Pig a Pancake* [60], and the pancake is lag. Attempting to address the player's lag problem required implementing an entire set of lag compensation techniques, then dealing with the resulting additional complications and, inconsistencies; this then requires further additional follow-on adjustments, yet the lag problem is never actually solved.

An interesting aspect mentioned is that *Valorant* uses fixed-length motion event descriptions that are of a different length than the display rendering steps, so that client and server may both refer to the same movement segment in all instances, even if display frame rates change over time. This communicated movement segment appears to be treated similarly to an event in the context of asynchronous time advances, including queuing, despite being evaluated within a synchronous time step simulation algorithm.

A common root cause of this lag problem and its cascaded response is the arbitrary choice of fixed time advance for a simulation. The *Consistency Throughput Tradeoff* discussed in subsection 2.1.5 deals with the update rate tradeoffs but does not examine the implementation implications of the rate tradeoff with respect to time advance choices. Steed and Oliveira [2] do mention the interaction of time advance and network packet latency and identify the significant practical problem of latencies that far exceed frame rates, together with the difficulties this presents in either converging states or dealing with causality issues that make convergence impossible.

The choice to use synchronous time step generates forced decision-points at each time step, even if the correct remote information is not available from the network until multiple time steps later. Where multiple steps are executed and a non-convergent causality-affecting situation arises, the computational and memory-space costs of rolling the state back across multiple time steps and recomputing are prohibitive, so an arbitrary adjustment results. These arbitrary adjustments are adequate for entertainment-related applications, but they are not satisfactory for applications that have hard causality-maintenance requirements.

2.4 Parallel and Distributed Discrete Event Simulation (Analytical Simulations)

In his 1990 Communications of the Association for Computing Machinery (CACM) article *Parallel Discrete Event Simulation*, Fujimoto [61] provides the definition for parallel discrete event simulations:

Parallel discrete event simulation (PDES), sometimes called distributed simulation, refers to the execution of a single discrete event simulation program on a parallel computer. [61]

2.4.1 The Independence of the Logical Process (LP)

Central to PDES analytic simulation literature is the concept of *logical process* or (*LP*). A LP is a useful abstraction to document the independence of one simulated system from another. This design pattern flows from the 1981 paper by Chandy and Misra, *Asynchronous Distributed Simulation via a Sequence of Parallel Computations* [62], which describes the relationship between a physical process (PP) and LP as:

There is a logical process (LP) corresponding to every PP. Each LP simulates the corresponding PP. . . . the sequence of messages sent by an LP must correspond exactly to the actual sequence of messages sent by the corresponding PP. [62]

Additionally, under Chandy and Misra's [62] LP model each LP executes at its own rate, independent of any other LP. This statement supports the body of literature on the use of the LP abstraction in simulation, but Chandy and Misra's [62] LP coverage is far deeper describing communication channels timing and deadlock-related handling, these specifics do not appear when other literature invokes the LP abstraction. As an example in a 2023 paper, Piccione [63] indirectly identifies a problem between the abstraction model and the implementations that provide the execution independence. Piccione [63] states:

This is because when fewer LPs are bound to a HW thread, the system becomes over-optimistic, and the high activity partition is subject to more rollbacks. [63]

For over-optimism to occur and allow some thread to race ahead of another, the waiting phases of Chandy and Misra's [62] LP model must be missing from the implementation(s) studied. The over-optimism is explored further in Section 2.4.1.2. The rollbacks mentioned are a inefficiency phenomena associated with the optimistic simulation paradigm. Conservative vs. optimistic simulation techniques, and rollback are covered in depth in Section 2.4.2.

Piccione is implicitly identifying that the over-optimism results from an unbalanced excess of some hardware thread's compute capability, and the proposed solution is to dynamically choose to change the scheduling paradigm to conservative in an attempt to prevent the over-optimism, and switch back to optimistic to not lose too much throughput efficiency.

The identified over-optimism is the result of some LP's code progressing through time on its hardware thread, but leaving out that the over-optimistic progression through time is only possible if the LP is implemented and executed without concern for the progress of any other LP it may interact with.

The community-accepted implementation pattern for the abstract LP requires Chandy and Misra's [62] assumption of full independence between the LPs. This independence implies that LPs are treated as first-class objects which retain the assumption of independence all the way down into the implementation of a simulation's code, and that assumption of independence implicitly precludes a hardware thread from using existing standard thread and process synchronization techniques. As Fujimoto stated in the Section's opening paragraph: "A sufficiently general solution to the PDES problem may lead to new insights in parallel computation as a whole." This quote from 1990 indicates the techniques available for thread synchronization (Dijkstra introduced semaphores in the early 1960's) remained unknown, misunderstood, or inexplicably considered not compatible with PDES.

2.4.1.1 The Logical Process (LP) and Virtual Time

Jefferson's [31] seminal paper *Virtual Time* implicitly deals with the difficulties of resolving the inter-LP synchronization issue via the implementation of virtual time and post-causality-violation rollback.

We envision systems of many (maybe thousands) of small processes all executing concurrently on a computer with many processors. It is useful to consider each process as occupying a point in virtual space, with its unique name acting as its spatial coordinate. Every primitive action (changing a variable, sending a message, etc.) thus has both a virtual time coordinate and a virtual space coordinate, and the set of all actions (side effects) that take place at the same virtual place x and virtual time t is referred to as the event at (x, t) . [31]

Jefferson however does not use virtual time as a LP synchronization shared memory value even though he has the *global virtual time (GVT)* shared memory value available. This appears to be a nod to the independence of LPs in his *Time Warp* implementation as confirmed in Jefferson's [31] discussion of virtual clocks:

each process has its own local virtual clock variable . . . At any moment some local virtual clocks will be ahead of others. [31]

With threads that are implementing LPs which are accessing multiple local virtual clocks it is apparent that each thread-bound LP is allowed to progress fully-independently from every other, which is the baseline intent of the fully-independent LP.

It could be possible to confuse that there are multiple virtual times to match those multiple virtual clocks, but this would be problematic because as Segal's Law states: A man with a watch knows what time it is, a man with two watches is never sure. Jefferson avoided Segal's problem by considering that there is only one virtual timeline upon which timestamps are registered, and a virtual clock is simply a most-recent measure (variable) of the most recent event times that a particular computational thread has evaluated. This interpretation delegates to the larger algorithm the task of managing the progression of those virtual clocks (thread variables) with respect to computational penalties for evaluating too far ahead, or too slowly.

When a LP simulates too far ahead and precipitates a causality issue, Jefferson [31] resolved it using his *GVT* variable that indicates all events before *GVT* are safe from causality issues. This bounds the time interval of the problem and Jefferson rolls back the necessary LPs to *GVT* and carefully re-simulates events strictly in relation to the singular timeline timestamps until the causality interactions are resolved [31]. Because there are no inter-LP constraints on progress in the optimistic scheduling paradigm, this may lead to a significant amount of work to redo.

Jefferson's choice to avoid synchronization until rollback is necessary masks a contradiction between the definition and concept of a fully-independent abstract LP and how an LP is implemented for interactions located on a virtual timeline. For causality repair a simulation-wide shared value, the *GVT*, is used as the post-causality-violation rearward window limit indicating the threshold before which all simulated events are considered safely correct in causality order. This globally shared *GVT* value is literally the shared memory value necessary for an inter-thread synchronization structure, but is not used as such until the optimistic scheduling is paused to effect rollback.

2.4.1.2 Are Logical Processes (LP) Actually Independent?

As a driving design pattern for distributed simulation, the faithful implementation of the abstract fully-independent LP into simulation code has not aged well despite still being in use today. As identified in the Piccione [63] paper, over-optimism is the same concern in 2023 as Jefferson's [31] concern in 1985. There is little-to-no progress on solving over-optimism, just incremental adjustments to techniques over the years, as evidenced by Piccione [63] stating their the 2023 paper is a derivative substitution of Steinman's [64] conservative advancement hybridization algorithm in the 1991 paper *SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation*.

The well settled concepts and terminology of thread and process synchronization from the operating systems sub-domain of computer science are not discussed and the absence remains a missed opportunity for the simulation community.

A potential reason for the 40-year old problem not using 50-plus year old techniques to solve it may be a analytical simulation community-wide bias to strictly implement LP in adherence to Chandy and Misra's [62] guidance that each LP executes at its own rate, independent of any other LP. This can explain the total absence of implementations that use existing standard thread and process synchronization models, because the thread and process synchronization models explicitly require the maximum forward progress of any particular hardware thread to be dependent on some shared memory variable with at least one other hardware thread.

Because the interpretation of the abstract fully-independent LP precludes execution time synchronization between LP threads and simulated events, synchronization is relegated to a scheduler implementation by-product towards maintaining causality. But if the scheduler is indirectly enforcing causality across the set of results from conceptually-implemented-as-fully-independent LP, this explicitly means *the results of those LP computations are not actually totally independent*.

Abstract LP cannot be fully independent at the same time their results are dependent on each other for proper causality in the simulation results. This logical contradiction is not resolvable under current paradigms and their widespread practice as the necessary supporting assumptions and implementation requirements are mutually exclusive.

2.4.1.2.1 Physical Processes are Not Fully-Independent This ubiquitous interdependence becomes obvious when stated in terms of the physical processes that are simulated. For example, the operations of each aircraft landing and departing at a commercial airport are not totally independent from all other operations at the airport. A runway is not used simultaneously, aircraft are time-ordered for access to the runway, in the air, on the taxiway and in combination. The airport tower personnel are responsible for communicating the safe ordering and monitoring that it is adhered to by all participants. This assignment and monitoring task can never make the airport fully-independent from the aircraft, there is communication between the entire set of them over the radio. The radio frequency is the shared medium that all these time-interacting entities use to coordinate their time and space use of other shared simulation assets such as the runway, taxiway, or boarding gate. Otherwise internally independent actions are effected without needing explicit interaction with the other airport simulation entities, but that internally-independent progression honors the time and space limits on accessing shared assets established via the synchronization radio calls.

Thus such physical processes are not fully-independent, more properly they indicate loose-coupling based on the need to share time and space assets within their scope of operation. If the LP is the abstract representation of the PP as per Chandy and Misra [62], and the PP are not fully-independent, then there is no forcing-function from the PP to require an LP to be fully-independent. It is not clear from reading [62] that Chandy and Misra intended the abstract concept of LP independence to be as dogmatic as it is seemingly treated by the implementing analytical simulation community, or if the intent is simply interpreting an LP as a black-box where internal computations are not sampling inputs other than those that arrive on the incoming communication channel of an LP, which indicates a loose-coupling strategy as the desired inter-thread interaction mode rather than the full-independence strategy that the analytical simulation community has settled on.

2.4.1.3 Logical Process (LP) Independence vs. Thread Synchronization

The success of the *Time Warp* algorithm both academically and practically cemented the continued use of the fully independent abstract LP interpretation throughout the literature and primed the environment into which Fujimoto [61] stated in his article *Parallel Discrete Event Simulation* (PDES) that:

From an academic point of view, parallel simulation is interesting because it represents a problem domain that often contains substantial amounts of parallelism, yet paradoxically, is surprisingly difficult to parallelize in practice. A sufficiently general solution to the PDES problem may lead to new insights in parallel computation as a whole. [61]

It may be that rather than parallel discrete event simulation leading the way into new compute paradigms, the parallel discrete event simulation community just needed to more closely follow the larger computing parallelization efforts including Dijkstra's semaphore synchronization and Osborne on speculative execution. Both topics are published and available at the time of the Fujimoto's article, semaphores are from the early 1960s, and Osborne's [65] speculative computation paper in 1990. The key difference between PDES's optimistic evaluation of independent LPs and the larger parallel computation concept of speculative evaluation is that speculative execution requires a means to constrain the speculative code progress [65], while optimistic evaluation (as covered in more detail in Section 2.4.2) explicitly does not have the necessary per-LP progress limitations. This difference is a key disadvantage for optimistic simulation, and it derives from the literal interpretation of the LP concept.

As identified above in Section 2.4.1.2, **the mutually exclusive requirements of the LP abstraction vs. LP implementation relationship makes it impossible to develop any synchronization solution for abstractly independent LPs.**

Proper synchronization of any parallel software program requires explicit use of some element of shared data between the threads or processes which are being synchronized. This required use of shared data breaks the strict independence assumptions used by the analytical simulation community even though they are not defined as such in [62] by Chandy and Misra.

This explicitly means that virtual timeline computational sequencing of entity behaviors for any particular LP thread is dependent on the progression of some other LP computational progression, but only dependent in terms of when those computations fall onto the virtual timeline creating a consistent causal event sequence.

Any solution to inter-thread synchronization requires acknowledging that some element of shared data is required for any LP thread that must be synchronized and thus the resulting set of synchronized LP threads are not fully-independent. This explicitly makes the virtual timeline computational sequencing of entity behaviors that any particular LP thread contains dependent on the time-space progression of some other LP progression, but only dependent in terms of creating a consistent causal event sequence. The last phrase may appear self-contradictory, that only the causal event sequence is dependent and not the execution of the events themselves being interdependent. Such a contradiction is illusory, since the conditions of sharing the time-space between multiple threads still allows each LP the independence to determine its own reaction to any inputs generated in the time-space synchronization process. These inputs can be communicated to an LP via a communication channel and thus allow the LP to evaluate itself and generate the appropriate responses that can then be further communicated outbound to other awaiting LP threads. This interpretation meets the basic requirements of Chandy and Misra's [62] LP concept, yet allows alternate implementations for managing input and output channels to the LP black-box.

2.4.2 Conservative vs. Optimistic Advancement and Rollback

Advancement of a distributed simulation is typically considered in terms of synchronizing the causal flow of events from logical processes by offering techniques for ensuring causality across the entire set of LPs. Per Fujimoto [61]:

A parallel discrete event simulation program may be viewed as a collection of sequential discrete event simulations that interact by exchanging timestamped messages. Each message represents an event that is scheduled (sent) by one simulator for another. Each sequential simulator is referred to as a logical process or LP. [61]

This interpretation has design utility, but as noted above does not directly lead to a functionally correct programming solution to enforcing causality. Instead it makes the task harder by generating an arbitrarily large number of potential computational thread requirements that must be synchronized—all without a clear metric to do so. The simulation community has settled on describing these synchronization algorithms as conservative vs. optimistic.

Conservative approaches strictly avoid the possibility of any causality error ever occurring. These approaches rely on some strategy to determine when it is safe to process an event (i.e., they must determine when all events that could affect the event in question have been processed). [61]

This approach is a pre-commitment technique (known as deterministic lockstep in VE networking) where an event is not committed to an LP until it can be verified as safe from the possibility of creating a causality artifact. These conservative and deterministic lockstep techniques have a well-deserved reputation as being slow (and unable to perform in real-time) due to the extra overhead involved in the verification and two-way communication delays imposed to guarantee time advance safety.

Optimistic approaches allow individual simulation LP threads, whether in the same machine or distributed to different machines, to progress without explicitly verifying that all potential causality contributors are equally time advanced. [61]

This approach is a post-synchronization technique and requires undoing or rolling-back the portion of the LPs progression which has proceeded too far ahead and is generating a causality exception.

From a mathematical perspective, an optimistic thread getting too far ahead “perceives” the same magnitude of delay-while-awaiting-other-threads-to-catch-up as might a time synchronized distributed thread subject to network latency creating the same magnitude of a round-trip response delay. This implies that the amount of work necessary to synchronize optimistic post-synchronized threads is likely in the worst-case similar to that necessary for synchronizing network-distributed threads.

Rollback is traumatic to the progression of an analytical simulation as it requires determining what simulation time an LP must revert to allow removing a causality error, discarding previously completed computational results after that time, and then computing forward again with no guarantee that the next unconstrained advances do not repeat the rollback process for a follow-on causality artifact. Jefferson [31] introduced the idea of simulation

rollback in the 1985 *Virtual Time* paper and describes management of global virtual time and antimessage use and transmission to support the ability to rollback a local LP and any other LPs that receive messages which must be reversed. Such a process requires both time and additional bandwidth without guarantee against cycles or deadlocks in the rollback execution [31]. These requirements affect every event on the LP from the rollback time forward, including cases where some discarded event has no direct interaction with the event sequence that triggered the rollback criteria.

Even with these potential pitfalls, the *Time Warp* algorithm is time-tested, considered efficient, and is still used as a reference implementation, measuring stick, or starting point for current-day parallel and distributed simulation development, as evidenced by Mirtich [7] in 2000, Koenig [58] in 2013, as well as Piccione [63] in 2023; there are a large number of other papers across the intervening decades. **This long running acceptance of rollback-vulnerable simulation algorithms in offline analytical simulations is not compatible with the real-time requirements of distributed interactive simulations with human or hardware-in-the-loop participants.** Arbitrary clock stoppages and restarts necessary for the resolving of rollback too readily create perceivable artificial discontinuities in the simulation progression, harming training engagement or generating non-physically possible situations that are not reasonably resolvable by the participant and create unfixable failure modes in the simulation.

2.5 Analytical vs. Virtual Environment Simulations

Fujimoto describes the differences between analytical simulation and virtual environments simulation in his book *Parallel and Distributed Simulation*. The primary differentiating factor presented is the presence of humans or hardware in the loop as part of the simulation, and “to give users the look and feel of being embedded in the system being modeled. As such, it is not always essential for these simulations to exactly emulate the actual system” [30]. This acceptance of lack-of-correctness actually results in a limiting assumption that analytical and VE simulations remain separate concerns:

In summary it is useful to distinguish between analytic and virtual environments simulations because they have different objectives, leading to different requirements and constraints. To some extent, this dichotomy is historic, as

much of the basic research in these applications has been conducted by different communities. We use this dichotomy here because these two areas have given rise to different problems and solutions. [30]

This is a false dichotomy that does not serve the shared scientific basis across the analytical and the VE simulations communities. The “different problems” are a difference in superficial description, not actually different problems.

There are sub-communities in the VE entertainment and gaming spaces where the dichotomy is at least not harmful, and those are generally not interested in large, networked implementations. The training and engineering evaluation uses of VE, especially multi-participant events, however, require correctness of simulations to avoid negative training and faulty design. Further, techniques providing correctness are directly useful to the entertainment and gaming space as well, as even gamers have a sense of justice that requires the presentation of a level and apparently consistent playing field. Real world processes with synchronous simulations, such as shared C2 or real-time collections of digital twins which are simultaneously forms of VE simulation, and that engage in sequences of analytically-based events, are simply not possible under this prevalent false dichotomy.

2.5.1 Hybridization, for the Win

This dissertation documents the hybridization of analytic and VE simulation techniques, and this work solves a simulation grand challenge level problem only through the hybridization of concepts. There are no new technical advances in the most recent 30 years that enable this dissertation topic compared to what existed prior. The average age of references cited in this dissertation is an indicator of the long-ago maturity of this research’s enabling concepts. More recent than the average references are primarily used to document the community’s lack of progress instead of a healthy continuously progressing state-of-the-art. By not addressing and cross-pollinating the areas of strong overlap between analytic and VE simulations the overall simulation community forfeit decades of otherwise potential technical advance.

The previous dichotomy quotation should not be interpreted narrowly as having created the dichotomy itself and causing the communities to progress independently. The quota-

tion’s observation draws from the consensus of both the VE and analytical simulations communities of that time, and the dichotomy is steadfastly maintained by the breadth of both communities—leading to technical stagnancy and intellectual disconnects. A recent indicator of this lack of technical progression is a peer-reviewed paper implementing a variation of the *Time Warp* algorithm from the analytics simulation community’s leading conference series, the 2023 International Conference on Principles of Advanced Discrete Simulation (PADS) Conference. The paper, *Hybrid Speculative Synchronisation for Parallel Discrete Event Simulation* [63] describes a hybridization of the *Time Warp* optimistic advancement and a conservative advancement algorithm.

The *Time Warp* algorithm is described in the seminal 1985 paper *Virtual Time* by Jefferson [31]. The specific technique of hybridization is a derivative substitution of Steinman’s [64] conservative advancement hybridization algorithm in the 1991 paper *SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation*. That a refereed journal paper in the preeminent conference of a discipline is selected and published while the paper presents barely incremental improvements in specific circumstances compared to 35+ year old published techniques either documents a solved domain, or stagnancy in the cutting edge—parallel and distributed simulation is *not* a solved domain. The VE community is also not unscathed in this area, the previous coverage of lag compensation in Section 2.1.5: *Analysis: Why is Lag Still a Research Problem?* documents the VE community’s continued simulation research and development stagnation.

The first step in rectifying a long-accepted problem is to recognize that it exists. The next steps require applying an improved theory into practice. The next chapters pursue such a path.

CHAPTER 3:

How *Lag Erasure* Works:

Algorithms and Operational Walkthroughs

This chapter is a series of explored examples that describe how *Lag Erasure* uses speculative evaluation techniques to create the baseline of a shared consistent state across the distributed simulation, and the specific *Lag Erasure* techniques which erase lag effects from the simulation state in situations that fall outside the envelope of the baseline techniques. Users activating input devices are an example of an event that the baseline techniques do not manage, because the simulation code can not know exactly when the user might make any specific input. These external input events always arrive to distributed hosts in a time-late manner, and in current simulation paradigms often generate anomalous situations in the local simulation progression that the users interpret as lag. *Lag Erasure* seamlessly manages any time-late event arrivals and automatically repairs the local simulation state to agree with the globally shared state. The state repair is the erasure of lag's effect on the simulation state.

The start to enabling the erasure of lag's effects is to have as consistent a distributed state as possible. The use of speculative evaluation within the simulation is foundational to this process and the design choice to employ it drives numerous additional follow-on design choices. The first portion of this chapter covers how these design choices operate together to create a consistent globally shared state within a simulation. The second part of the chapter covers the follow-on design choices that are able to compensate for latency in transmission that would otherwise create the perception of lag for the user, this is the process is where lag is erased from the state of the simulation.

There are intentional redundancies in the descriptions of this chapter with respect to content in Chapter 4: *Lag Erasure: Zero-lag System Design*. This chapter is presented as a standalone *how does it work* architectural design level description that contains enough detail to dispel the need to treat the *Lag Erasure* algorithms and techniques as pure notional theory. The additional low-level design and implementation tradeoffs necessary to understand how to

implement the *Lag Erasure* algorithms at a coding development level are provided in Chapter 4 which covers *why does it work* questions.

3.1 Speculative Evaluation Provides the Foundation

To enable the full set of *Lag Erasure* algorithms requires challenging the foundational yet unstated assumption behind the *Consistency Throughput Tradeoff* as discussed in Section 2.1.2 and Section 2.1.5. The unstated assumption in how the *Consistency Throughput Tradeoff* precludes dynamic state under real-time constraints is: dynamic state change messages arriving from a remote host after some number of local frame updates during the message's in-transmission time requires too long to roll back and recompute the correct state. The excess recompute time results in breaking the real-time constraint of a VE simulation in a manner that cannot be repaired.

Separating real-time simulation hardware frame display computation rate constraints from the message arrival timeline is accomplished by decoupling simulation entity updates from the frame computation times. This returns the Model-View-Controller (MVC) pattern to three independent components within the simulation architecture rather than forcing the Model implementation details to intermingle with and meet constraints that are internal to the View. Further MVC coverage is in Section 4.1. The decoupling is implemented with the use of ADES as it is the general case of simulation in a theoretical view, and fixed-step synchronous time advance simulation used in traditional VE and game applications is the special case. The relationship is simple: ADES allows every event to have an individual different time advance, and these events are time-sorted in a queue. The special case fixed-step simulation is simply that the time advances are set to be identical, and the entities are added to the queue in the order that the designer wants them evaluated.

The differences in implementation do create different tradeoffs for efficiency and optimization, but behaviorally a fixed-step synchronous time advance simulation is also properly labelled as a DES. For example, a fixed-step synchronous time advance evaluation is optimized by grouping necessary entity data into dedicated arrays that may be rapidly consumed in-order by a small chunk of code that remains resident in cache. This yields memory access efficiencies gained via the Locality Principle [40] and hardware cache fetching mechanisms. The design of each phase of a time step maximizes how it may be mass-evaluated this way.

Because an ADES does not iterate over all entities in a once-per-time-step manner, the ADES optimizes instead by requiring fewer total evaluations using event lengths that are greatly extended compared to fixed time steps. Inevitably, the differing implementation and optimization techniques have follow-on effects in the rest of the simulation evaluation chain.

Following is a description of how an ADES simulation works which is most useful to traditional VE simulation developers because to the analytical simulation community this portion is yet another ADES implementation. The next subsections, 3.1.2 and 3.1.3, are targeted to both ADES and traditional VE simulation communities and cover the ground-work for incorporating physics-based motion into an ADES in an efficient manner. The final subsections traverse through multithreading (subsection 3.1.4) as the lead in to network distributed ADES simulations, where it is established that the communications implementation for subdividing the collision detection problem are simultaneously tuned to support the communications requirements associated with arrival delays between distributed hosts.

3.1.1 The EventList

Every event in an Event-Graph (see Section 2.3.4.3) is eventually placed in an `EventList` where it is time-sorted and awaits being selected as the next event to evaluate, events are never selected from the middle, ever. Figure 3.1 is a snapshot of an `EventList` for one moving and one stationary entity (ping-pong balls), each with the beginning event titled `startMove`. Note the times appearing on the far left, this is when the event is pulled off the `EventList` and evaluated to create the next event in sequence, if applicable. Following the event time and title is a state summarization for the entity scheduled, generated in these examples via a `toString` function call. The specifics of the depicted spatial motion approximations are covered in Section 3.1.2.

```
** Event List 1 -- **  
0.010      StartMove      {PK_MovingEntity[ID:2, SMA[3, 0.000 to 0.010, @(0.000, 5.403, 0.000)]  
1.798e+308 StartMove      {PK_StationaryEntity[ID:1, SMA[2, 0.000 to 1.798e+308, @(0.000, -0.500, 0.000)] to (0.000, -0.500, 0.000)]  
** End of Event List -- **
```

Figure 3.1. An `EventList` at the start of a simulation.

If the event being evaluated generates any follow-on event(s), they are placed into the `EventList`, if not the `EventList` simply becomes one event shorter. Figure 3.2 illustrates the `EventList` shortly after removing the 0.01s scheduled `startMove` event for evaluation.

In this example the output is generated following the computation of follow-on motion for the event but before generating its follow-on event.

```
Time: 0.0100    CurrentEvent: StartMove
{PK_MovingEntity[ID:2, MSMA[0.0100 to 0.220, @(0.000, 5.403, 0.000) to (0.000, 5.166, 0.000)
    SMA[4, 0.010 to 0.020, @(0.000, 5.403, 0.000) to (0.000, 5.401, 0.000)
    SMA[5, 0.020 to 0.060, @(0.000, 5.401, 0.000) to (0.000, 5.386, 0.000)
    SMA[6, 0.060 to 0.220, @(0.000, 5.387, 0.000) to (0.000, 5.166, 0.000)

** Event List 1 -- **
1.798e+308    StartMove    {PK_StationaryEntity[ID:1, SMA[2, 0.000 to 1.798e+308, @(0.000, -0.500, 0.000) to (0.000, -0.500, 0.000)
** End of Event List -- **
```

Figure 3.2. The EventList after removing the 0.01s scheduled startMove event for evaluation.

3.1.2 Movement Representation

3.1.2.1 On the EventList

Every shared entity in the simulation that may move (or be interacted with) appears on the EventList with a startMove event, exactly once. Non-moving objects, whether permanently stationary, or movable but just not moving now, have event durations set to the largest representable non-infinite value. This improves computational efficiency as these events are never accessed during a simulation run, they are too far into the universe's future, thus avoiding a requirement to repeatedly toggle and de-toggle motion and/or collision handling.

While a motion event is under evaluation it is designated the CurrentEvent of the EventList. When a CurrentEvent motion-related event is evaluated it schedules the next motion event (startMove) to the EventList. This way, moving entities always have one and only one motion event on the EventList at-a-time. This fits within the startMove event cycle pattern depicted in Figure 3.3.

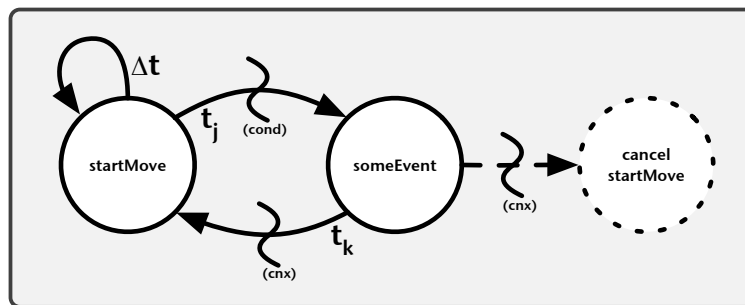


Figure 3.3. The startMove event cycle pattern.

The `startMove` computations fully specify a movement interval that extends into the future, it is not a stale snapshot of the recent past. The `startMove` event always reschedules itself for reevaluation at its adaptively computed step length (Δt), and the function implementing it only needs to compute the next motion event represented by the Δt rescheduling loop. If the `startMove` event conditionally schedules `someEvent`, this event may be a single event or represent an entire event chain. If an event or chain determines an event is superseded, the event is canceled and any necessary replacement event scheduled, unless of course the reason for supersession is removal and then the entity just does not reappear in the `EventList`. An example of supersession is when the collision detection system evaluates the contact resolution from a collision; at the moment of contact the involved entities `startMove` events currently on the `EventList` are obsolete and new `startMove` event computations are triggered by the collision resolution.

With respect to movement, following this cycle approximates the concept of the ADES LP construct (see Section 2.4) as the cycle is defined in a manner unconstrained by other entities, yet the ability to react to other entities movements and/or physical constraints such as collisions is explicitly maintained and scheduled via the `someEvent` side of the cycle. The same basic LP-approximation pattern may be adapted by simulation designers for non-`startMove` events as well.

3.1.2.2 Spatial Representation for Movement

Each movement event is plotted in the 4D space-time continuum via a static 3D representation. This dimensionality collapse simplifies the determination of time-dependent locality queries between entities. Entity trajectories are documented via a `SpatialMotion` object which tracks the entity center-of-mass, a `RotationalMotion` object separately tracks rotations about the center-of-mass.

`SpatialMotion` objects have two versions, the `SpatialMotionApproximation` (SMA) which is always present, and the `MultiSpatialMotionApproximation` (MSMA) which is a wrapper for multiple brief-duration SMAs. The MSMA provides efficiency when step sizes are temporarily brief, especially in the vicinity of acceleration sign changes, by holding multiple SMAs in a threshold designated minimum-duration event. This both preserves computational fidelity by not truncating or approximating component motions, while still limiting the event scheduling overhead of `EventList` operations by not requiring separate

events for every one of the brief-duration SMAs. From this point onward, the term SMA equivalently covers operation of the MSMA within the presented simulation architecture.

The end goal is to provide an efficient means of determining where the entity is at all times. An entity composed of complex geometry is computationally difficult to evaluate repeatedly so collision proxies are used to wrap and approximate the actual geometric shape, and these shapes are formed into a tree structure more commonly termed a **bounding volume hierarchy (BVH)**. The root geometry of the BVH is a sphere that provides a worst-case proxy bounding volume (BV) enclosing the full entity volume, co-located at the center-of-mass, as the sphere travels along the SMA for the specified event duration.

3.1.2.2.1 The Capsule is a line-swept sphere volume (see Figure 3.4) and the primary tool to document entity spatial motion. An efficiency benefit of capsules is that mathematically determining whether two capsules overlap is a trivial distance-between-line-segments computation and comparing that result to the sum of the two involved spherical radii.

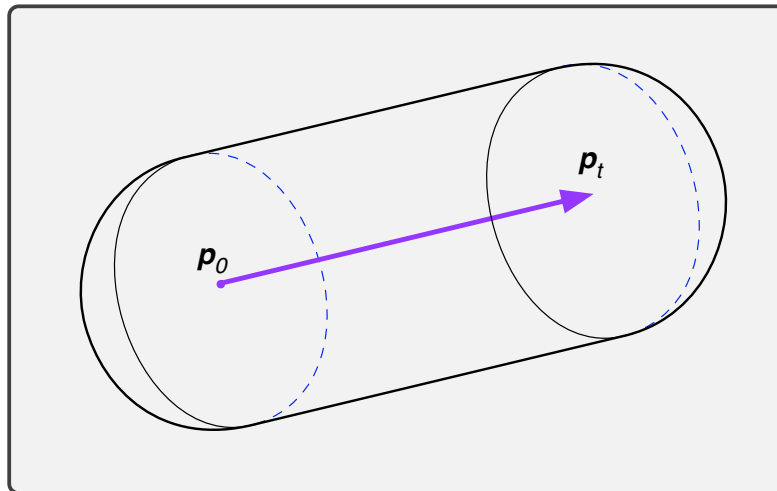


Figure 3.4. A line-swept sphere capsule wrapping a linear motion approximation.

3.1.2.2.2 The MotionWitness is an extension of the capsule which serves as the *witness* to future non-colliding motion of the entity. Due to the adaptively derived step lengths in motion computations and use of non-linear second-order accelerated motion instead of first order linear approximations, a single MotionWitness as appearing in Figure 3.5 may

replace dozens to hundreds of traditional time-stepped spatial relationship evaluations yet maintain increased accuracy and fidelity. This topic receives deep-dive coverage as part of the collision detection subsystem in Section 3.1.3.

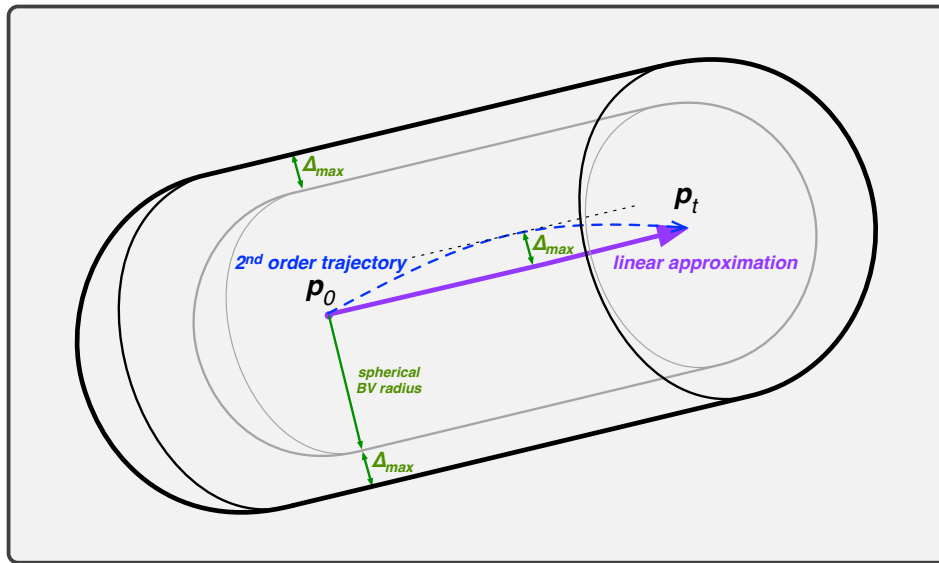


Figure 3.5. The MotionWitness containing non-linear entity motion, represented by a buffered capsule.

An SMA is a second-order approximation for an entity's motion over the defined interval, e.g., a section of the parabolic arc a ball follows as players toss it. Under a fixed-step synchronous time advance, a curved second-order approximation is inefficient to work with because the mathematics do not have the linear approximation simplifications necessary for fixed-step related collision detection algorithms. The repeated high evaluation-cost math collision detection problem is avoided in this ADES by trading multiple short-duration first order motion evaluations for fewer longer-duration motion evaluations and each of these longer durations is documented with a single MotionWitness. While a naïve linear approximation between SMA endpoints might be used, it could be possible for some portion of the entity to fall outside a capsule using the naïve linear approximation because the entity's actual path is curved. This problem is avoided with a one-time adjustment of the capsule volume, ensuring that the volume the bounding sphere sweeps within the curved second-order approximation is fully contained within a slightly increased radius capsule defined to handle second order acceleration effects over the naïve linear approximation between SMA endpoints.

This results in a *buffered capsule* as depicted in Figure 3.5, and this buffered capsule is the geometric volume of the MotionWitness. A strength of this second order MotionWitness is that once the buffered capsule is computed it is valid for the duration of the SMA and geometrically this MotionWitness simply acts like any another efficient-to-test-for-overlap capsule. The buffering to ensure containment is computed by parametrically determining the maximum distance ($max\Delta$) the curved path deviates from the naïve linear approximation between SMA endpoints. The extra volume created by enlarging the capsule has a small effect on increasing false positives within the collision detection broad-phase which is covered within Section 3.1.3.2.

3.1.2.3 The MotionWitness is Not Dead Reckoning

Some distributed simulation standards formally define the process of dead reckoning (DR) in an effort to reduce bandwidth requirements between participating hosts. For example, from the Institute of Electrical and Electronics Engineers (IEEE) DIS standard:

A method of position and/or orientation estimation called dead reckoning shall be employed to limit the rate at which Entity State Protocol Data Units are issued. By estimating the position and orientation of other entities, it is not necessary to receive a report about every change in position and orientation that occurs in the entity's trajectory. Only when a change in position and orientation differs by a pre-specified amount (threshold) from the dead reckoned position and orientation is a new update required. [66]

The Entity State Protocol Data Units (ESPDU)s that are being “issue rate-reduced” per the above quote are transmitted (and received) “time late” compared to the absolute time of the state they document. This is not an accident or a situation that may be changed, explicitly a note in the DIS standard Sec 4.4 states “A Protocol Data Unit (PDU) shall never be issued in advance of the occurrence of the event or state communicated by the PDU” [66]. This alone is enough to determine that entity motion computed in the speculative evaluation technique does not comply with the specification of a DR, but there are additional important differences remaining to cover.

The DR process begins upon receipt of a new ESPDU. The timestamp of this new packet is the start time for the new DR, and its position and/or orientation information is added

to the existing entity info in the DR algorithm to determine the new DR. As the DR is using rate-limited updates, the information it is based on is increasingly stale between those updates and this diverges the DR from the true track. Upon receiving a new update with a true position, a smoothing algorithm is used to re-converge the DR towards the true track.

By intent, an updated DR is more accurate than an older and stale DR, but it may be visually jarring and counter to maintaining training immersion to simply teleport an entity avatar to the updated DR position. Smoothing is used to visually drag the avatar from its inaccurate stale position to a more accurate (yet still somewhat inaccurate) updated DR position. The inherent inaccuracy of the smoothed position generally decreases as the new DR position is approached. The primary benefit is avoiding jarring or distracting visual artifacts resulting from positional updates.

It is readily apparent that using DR and smoothing leads to a tail-chase by each remote host to provide less stale and less inaccurate remote-host-controlled entity positions. Despite the tail-chase behavior it is a useful technique for simulations where a high degree of precision in time and space absolute accuracy is not critical to the participants. For example, inaccuracies on the order of centimeters in smoothed aircraft DR position is unlikely to cause decision difficulties for pilots in a one-on-one air-combat dogfighting scenario.

In contrast, at creation and transmission, the `MotionWitness` is never stale as is track information for an ESPDU and any resultant DR. The `MotionWitness`'s contained SMA is the true track and remains so until its completion, replacement due to new control inputs, or a collision requires its replacement. This is a contract by the originating host for the entity that the motion is precisely represented during the `MotionWitness` duration.

3.1.2.4 The `MotionWitness` and the `startMove` Event Cycle

As the simulation executes the `startMove` event cycle of Figure 3.3, the `CurrentEvent` is pulled from the `EventList` and exactly one future movement SMA is generated. This SMA is first used to generate the follow-on event to be inserted into the `EventList`, then the SMA is incorporated into a `MotionWitness` and the `MotionWitness` is passed to the collision detection subsystem for further evaluation. If the collision detection subsystem determines there is no potential contact over the `MotionWitness` interval, the `someEvent` event of Figure 3.3 is not selected, the cycle is satisfied and the `CurrentEvent` is now properly replaced leaving exactly one `startMove` event on the `EventList` for the entity.

Alternately, if the collision detection subsystem determines there is a contact during the `MotionWitness` interval the `startMove` event currently on the `EventList` is cancelled, removing it. The cancelled event is replaced by a new `startMove` event resulting from computing the collision's response, again leaving exactly one `startMove` event on the `EventList` for the entity. Note that this cancellation does not directly affect the `MotionWitness` itself, only the entity's next scheduled `startMove` event in the `EventList` is removed because the entity never reaches the corresponding endpoint of the current `MotionWitness` motion due to the collision response. Insertion of the newly computed SMA into the collision detection subsystem will trigger the replacement of the current `MotionWitness`, details are covered separately in the next subsection.

3.1.3 Collision Detection

Collision detection follows the typical two-phase process which includes a broad phase that has the goal to eliminate or cull as much potential unnecessary detailed work as possible, and a narrow phase which is the surviving necessary detailed work that is needed to precisely determine any potential contact.

3.1.3.1 Traditional Time-step Collision Detection Synopsis

Time stepped collision detection generally uses the broad phase goal of identifying what entity pairs are physically close enough together to warrant further investigation. Because time steps are small, the list of potential contact pairs does not change quickly enough to invalidate physical caching and other algorithmic optimizations. These identified-close-enough pairs may, depending on specific technique, be further tested for possible rejection and those pairs that are not rejected are passed along to the narrow-phase.

The narrow phase executes detailed static testing for geometric interpenetration with between objects. Results are used to inform collision response, and the cycle repeats in the next time step. When an interpenetration (collision) is detected during the current time step, the contact resolution first requires objects to be non-physically moved (or updated) to remove the interpenetrations caused by naïve movement before the response computation begins. If the repositioning is not accomplished first, then the normal vectors from the polygon face to the penetrating vertices are facing the wrong directions, and non-realistic collision responses ensue. This explicitly non-physically based position update to remove the interpenetrations mathematically distorts the balance of kinetic energy in the system

requiring periodic energy-state re-normalization for moving objects to avoid a simulation simply blowing up from the extra energy injected from fixing interpenetration.

Some systems remove the interpenetration problem through a process termed continuous collision detection (CCD), where the interpenetrating objects are backed off in time using the correct math until simple tangential contact. Response is calculated from this correct position and then the objects are forward simulated again to the end of the time step. The special-case handling and adjustment calculation in both directions is considered computationally expensive compared to the above traditional non-continuous methodology. This provides demotivation towards widespread choice to use CCD methods even though the mode is widely available, for example the *Unity* [45] game engine provides a continuous collision detection option, which uses simplified collision proxies such as spheres, capsules and boxes to offset some of the computational expense.

3.1.3.2 Collision Detection With Continuous Motion

The `startMove` event cycle of Figure 3.3 coupled with adaptive-step computation provides a continuous motion state trajectory across time rather than the series of state snapshots provided by traditional time step evaluation. Having a continuous state natively available to every distributed host as depicted in Figure 3.6 rather than the snapshot states of Figure 3.7 means some of the assumptions and performance tradeoffs in CCD no longer apply. Each section of this continuous trajectory is mathematically represented with a `MotionWitness` placing that trajectory into 3D-space.

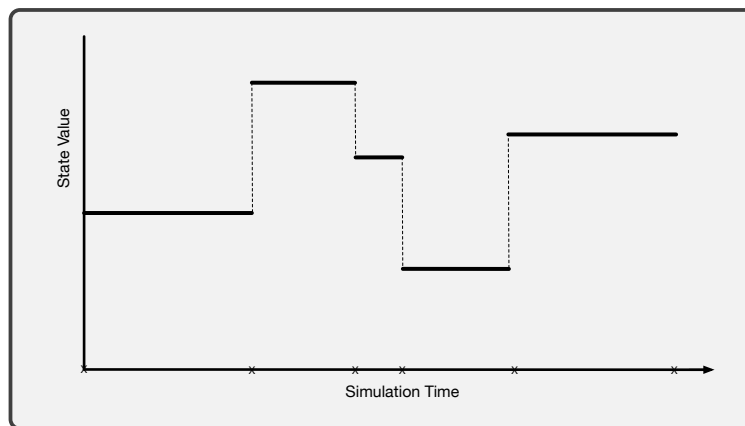


Figure 3.6. A typical DES state trajectory. Source: [37].

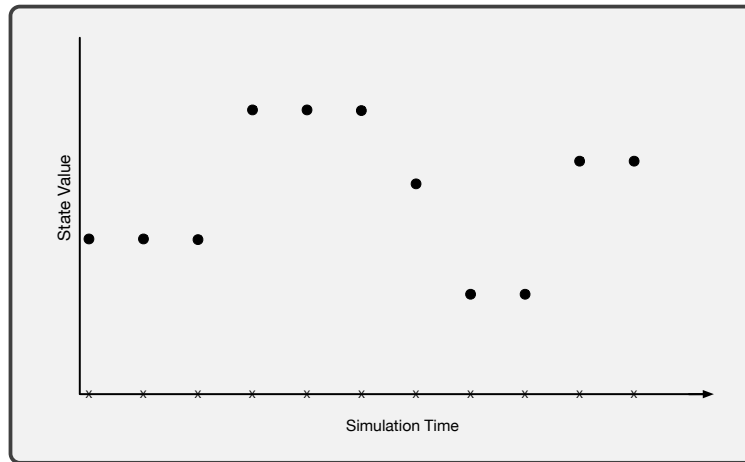


Figure 3.7. A typical time-stepped simulation state trajectory. Source: [37].

3.1.3.2.1 The R*-Tree The R*-Tree data structure developed by Beckmann [67] is selected based on a deep-dive through Samet’s immensely useful *Foundations of Multidimensional and Metric Data Structures* [68] for the R*-Tree dynamic maintainability and efficiency at answering location queries for multiple objects. The R*-Tree is used to group these MotionWitnesses to provide efficient broad-phase culling power. To further influence that efficiency, only verified non-overlapping (non-colliding) motion is committed to the R*-Tree.

The R*-Tree is a self-balancing axis-aligned bounding box (AABB)-based tree data structure where the grouping goal is to maintain nodes above the leaf level to hold children whose collective AABBs are enclosed in as cubic a volume as practicable. This minimizes the internal volume compared to the external surface, maximizing sorting power as this minimizes overlap between subtrees [67], this is a useful property for the broad-phase culling stage. The basic usage concept of the R*-Tree is that a data trace, in this case the MotionWitnesses SMA, is bounded by an AABB that encompasses the MotionWitness volume as tightly as possible. The resulting AABB is what is inserted into the tree as a leaf.

A historic progression of leaf nodes associated with the movement of a ball is visualized in Figure 3.8 as an example of the R*-Tree operation as used in this architecture. A translucent blue AABB, barely visible except at the far right just outside the green box, defines the fixed Region the tree operates within. The transparent green box is an intermediate node whose children may be other intermediate nodes, or leaves, but never both. Leaves are depicted in

red, sized based on the contained MotionWitness. The MotionWitnesses themselves are not part of the tree structure, so MotionWitness visualizations are not depicted.

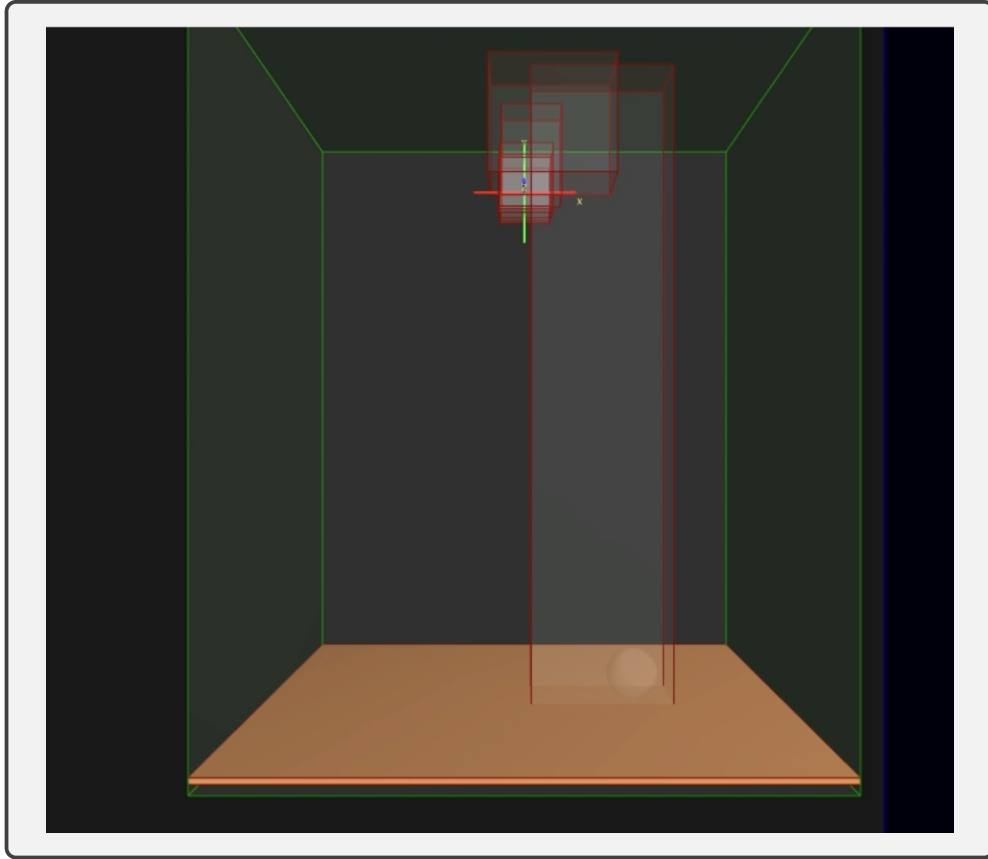


Figure 3.8. Collision detection data structure visualization Snapshot of a series of MotionWitnesses inserted into an R*-Tree.

3.1.3.2.2 The Lozenge appears as the orange-colored object at the bottom of Figure 3.8 is a rectangle-swept sphere volume, or Lozenge. A Lozenge looks like a box with radiused minor edges rather than square edges. Lozenges provide efficient capsule-style calculations for minor faces, and the major faces use simple planar standoff checks. Large single boxes that would create excessive false positives on the minor faces may be specified with a set of compressed Lozenges to limit the minor face radius effects to a negligible factor.

3.1.3.2.3 Putting It All Together into a walkthrough of the system as it generates the motion representations and collision support data structures from the adaptively determined motion. To start is a scene continuing two ping-pong balls, one fixed, one moving within in

a vacuum and subject to National Institute of Standards and Technology (NIST) standard Earth surface gravity [69]. The scene is constrained to a single spatial Region defined by the fixed extent of the simulation's R*-Tree, depicted by the blue box of Figure 3.9.

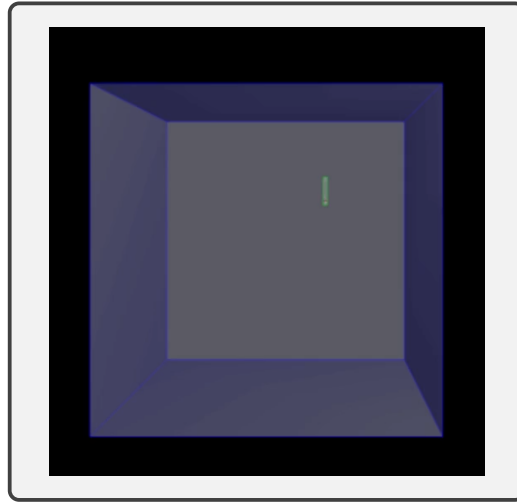


Figure 3.9. The blue box represents a simulation scene Region.

The two ping-pong balls and their leaves are further contained by the intermediate node of the tree, the small green rectangle in Figure 3.9 and zoomed-in upon in Figure 3.10.

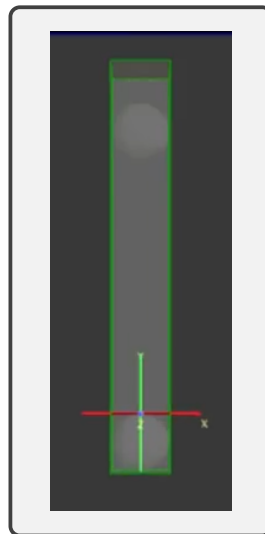


Figure 3.10. An R*-Tree intermediate node containing two leaves, one containing each ball.

The axes in the scene are for visual orientation only, they are not simulated objects. The red Leaf AABB boundaries are not visible due to browser-based aliasing favoring the green AABB over the red AABB. At simulation start the falling ball is initialized with a 0.01 second motion duration to allow the initial motion update to calculate the best motion duration to use for the next event, with the `EventList` appearing as in Figure 3.11.

```
** Event List 1 -- **
0.010      StartMove    {PK_MovingEntity[ID:2, SMA[3, 0.000 to 0.010, @(0.000, 5.403, 0.000)]
1.798e+308  StartMove    {PK_StationaryEntity[ID:1, SMA[2, 0.000 to 1.798e+308, @(0.000, -0.500, 0.000)] to (0.000, -0.500, 0.000)
** End of Event List -- **
```

Figure 3.11. The scenario's initial `EventList`.

The adaptive algorithm operates with a designer-choice parameter limiting maximum step growth to 4x from the previous step, resulting in a MSMA comprised of three SMA segments of length 0.01, 0.04, and 0.16 sec each, advancing the simulation to time 0.22 per the current event depicted in Figure 3.12.

```
Time: 0.0100      CurrentEvent: StartMove
{PK_MovingEntity[ID:2, MSMA[0.0100 to 0.220, @(0.000, 5.403, 0.000) to (0.000, 5.166, 0.000)
      SMA[4, 0.010 to 0.020, @(0.000, 5.403, 0.000) to (0.000, 5.401, 0.000)
      SMA[5, 0.020 to 0.060, @(0.000, 5.401, 0.000) to (0.000, 5.386, 0.000)
      SMA[6, 0.060 to 0.220, @(0.000, 5.387, 0.000) to (0.000, 5.166, 0.000)

** Event List 1 -- **
1.798e+308  StartMove    {PK_StationaryEntity[ID:1, SMA[2, 0.000 to 1.798e+308, @(0.000, -0.500, 0.000)] to (0.000, -0.500, 0.000)
** End of Event List -- **
```

Figure 3.12. `EventList` reflecting the initial movement recalculation.

This MSMA is one event movement segment and the basis for the `MotionWitness` encapsulated into a Leaf and inserted into the `R*-Tree`. This is an efficiency-increasing grouping for the broad-phase culling stage by reducing dynamic tree operations and the impact on event scheduling for multiple motion events that barely move. The threshold for creating a MSMA out of multiple SMA objects is a motion duration less than the `LatencyCompensationWindow` (LCW) duration (which is fixed to 100ms for this example.) Inspecting Figure 3.12 shows that the first two SMA of the MSMA are not yet totaling to more than 100ms of duration, and inclusion of the third SMA satisfies passing 100ms.

The MSMA containing event completes as expected and computes its successor's 0.64 sec duration event, posting it to the `EventList`, generating the associated `MotionWitness` and Leaf which updates the `R*-Tree` uneventfully. Following the next motion update, the broad phase detects the potential for a collision. Initially, the event is computed to be 2.56 sec in

duration, but on R*-Tree insertion the overlap of the new leaf AABB and the fixed ball's Leaf AABB triggers the generation of a NarrowPhase event computation, with a potential intersection time computed at time 1.0s.

This leaves the non-colliding motion occurring up through 1.0s simulation time, reflecting that the motion interval is truncated to time 1.0 (from 2.56s) both shortening the MotionWitness, and contracting the extent of the Leaf AABB, providing a guaranteed to be non-colliding-upon-insertion motion event. This algorithmic optimization provides major benefits on both computational efficiency and physical correctness. Note that non-colliding-upon-insertion means only that, other object motions may be updated in the simulation that contact the newly inserted object motion before reaching 1.0s sim time. This specific possibility is why the NarrowPhase event is scheduled at time 1.0, not immediately evaluated, and why NarrowPhase events are self-discarding if they begin evaluation, but the involved entities motions have changed resulting in mis-matched SMAs. This avoids potentially immense amounts of problematic rework that might be required if the NarrowPhase event is evaluated immediately, but ends up as superseded.

During evaluation of the NarrowPhase event, showing as the CurrentEvent in Figure 3.13, it is determined that a physical collision occurs between the two balls at 1.0s sim time.

```
Time: 0.9990    CurrentEvent: NarrowPhase    {PK_MovingEntity[ID:2, SMA[9, 0.860 to 1.000, @(0.000, 1.777, 0.000) to (0.000, 0.500, 0.000)]
** Event List 1 -- **
0.999    EvaluateContact {PK_MovingEntity[ID:2, SMA[9, 0.860 to 1.000, @(0.000, 1.777, 0.000) to (0.000, 0.500, 0.000)]
1.000    StartMove      {PK_MovingEntity[ID:2, SMA[9, 0.860 to 1.000, @(0.000, 1.777, 0.000) to (0.000, 0.500, 0.000)]
1.798e+308 StartMove      {PK_StationaryEntity[ID:1, SMA[2, 0.000 to 1.798e+308, @(0.000, -0.500, 0.000) to (0.000, -0.500, 0.000)]
** End of Event List -- **
```

Figure 3.13. EventList approaching first contact.

This double-checks because the initial conditions are set so that this happens. With the contact time determination complete the EvaluateContact event is scheduled for the predicted contact time of 1.0 for the same reasons of potential confounding events as described above, and is depicted in the EventList. Also notice that while the NarrowPhase and EvaluateContact events are in play, the moving ball's next projected StartMove event is also in the EventList showing how the interrelating portions of the Figure 3.3 StartMove event cycle are reflected at runtime in the EventList. The StartMove event

being the result of the tight self-reschedule loop, and the collision subsystem related events realizing the conditional rhs of the cycle.

Once the `EvaluateContact` is selected as the new `CurrentEvent` (see Figure 3.14) the previously scheduled `StartMove` event is superseded by the collision response calculations generating a new upward trajectory for the moving ball. `EvaluateContact` shows that it has computed the next SMA necessary for scheduling a replacement `StartMove` event to complete the rhs of the `StartMove` event cycle.

```
Time: 0.999 CurrentEvent: EvaluateContact {PK_MovingEntity[ID:2, SMA[10, 1.000 to 2.000, @(0.000, 0.500, 0.000) to (0.000, 5.403, 0.000)]
** Event List 1 -- **
1.798e+308 StartMove {PK_StationalEntity[ID:1, SMA[2, 0.000 to 1.798e+308, @(0.000, -0.500, 0.000) to (0.000, -0.500, 0.000)]
** End of Event List -- **
```

Figure 3.14. EventList showing `StartMove` cancellation.

The moving ball's endless bouncing cycle reaches its first apogee at 2.0 sec sim time and the downward trajectory begins again with the `CurrentEvent` displaying the descending `StartMove` SMA and the inevitably scheduled `NarrowPhase` event in Figure 3.15.

```
Time: 1.9990 CurrentEvent: StartMove {PK_MovingEntity[ID:2, SMA[12, 1.999 to 3.000, @(0.000, 5.403, 0.000) to (0.000, 0.5000, 0.000)]
** Event List 1 -- **
2.999 NarrowPhase {PK_MovingEntity[ID:2, SMA[12, 1.999 to 3.000, @(0.000, 5.403, 0.000) to (0.000, 0.5000, 0.000)]
1.798e+308 StartMove {PK_StationalEntity[ID:1, SMA[2, 0.000 to 1.798e+308, @(0.000, -0.500, 0.000) to (0.000, -0.500, 0.000)]
** End of Event List -- **
```

Figure 3.15. EventList revealing the cycle continues.

3.1.4 Multi-threaded Simulation

The above simulation operation descriptions occur within a single `Region` and a single thread of execution. To this point there is no requirement for inter-thread interaction or synchronization of any shared data. Extending into multi-threading simply adds additional `Regions` and the means to migrate information about entities from one `Region` to another as necessary, extension of this multi-threading into distribution is covered in Section 3.1.5. This architecture uses a simplified one-way version of the protected service process model from the basic operating systems online resource by Denning, *The Art of Operating Systems: An On-Line Book of Slides* [70]. While `Regions` may be abutting, this is not a requirement,

one Region may overlap another when this is advantageous for the modeled scenario. Consider a simulation with a house, a Region may be set up to simulate inside the house which is generally unconcerned with what is happening in the larger neighborhood Region that contains it and other house Regions. As entities move into or out of the house, the Region they are simulated within changes. This change of Region requires communication and includes the time period where the entity is physically in both Regions. Details of how this transpires is in the discussion of the Liminal entity representation in Section 3.1.4.3.

3.1.4.1 The Protected Service Process Model

The basic protected service process model of the operating system as covered in Module 5.2 of Denning [70] is a request-and-response sequence as illustrated in Figure 3.16. The sender is requesting some OS service to take an action on the sender's behalf. The sender builds a message that includes the request and the return address and places it into the service process' message buffer (mb) and the reference to the mb is retained so that the reply uses the same buffer and precludes any potential buffer-resource deadlock [70]. The receiver is the service process, it gets the message and unpacks the mb to find the specific request and note the return address. Upon completing the requested action the reply is packaged into mb and transmit back to the return address (original sender) [70].

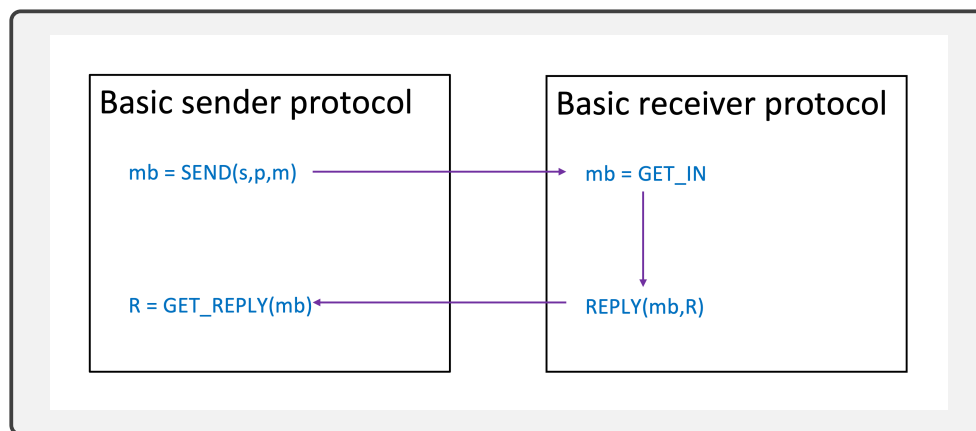


Figure 3.16. Basic sender-receiver protocol. Source: [70].

Figure 3.17 shows the filesystem service protocol interaction between processes. The user process makes its request by placing the mb into the inbox of the service process (1), this insertion takes place within the user-process' thread of execution. Then the user process puts itself to sleep awaiting the return reply, and the sleeping ensures that the shared mb is not simultaneously accesses by both participants.

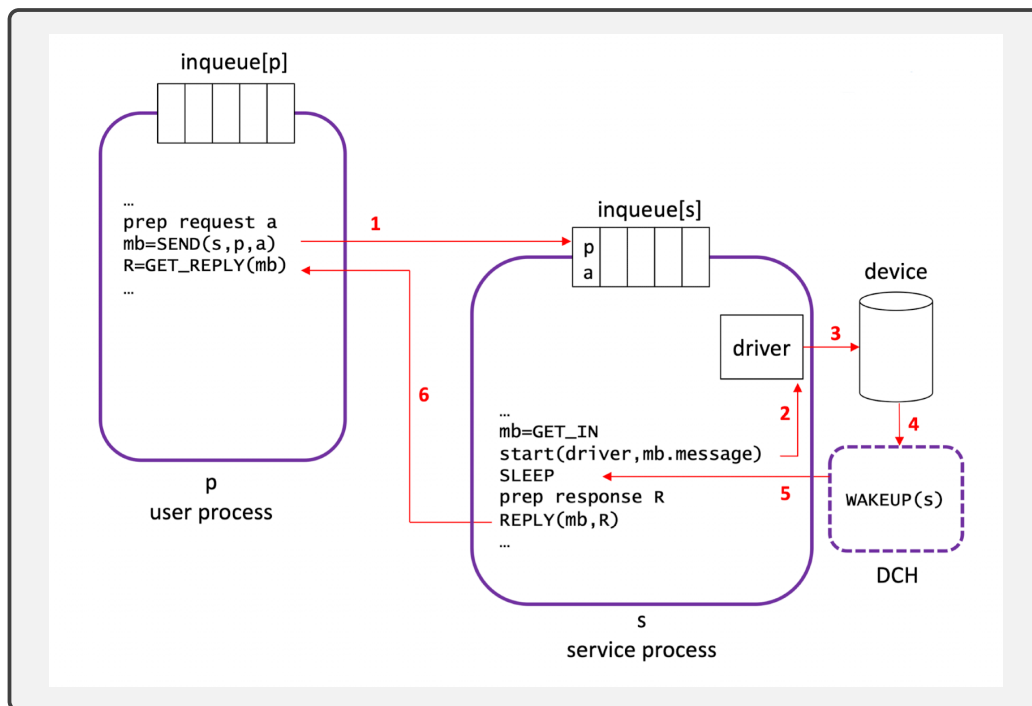


Figure 3.17. The protected service process interaction loop. Source: [70].

The service process then does its job, and in this case a further service request (2) is submit to the disk driver that controls (3) the physical drive itself. When the drive finishes the read/write operations it invokes (4) the disk completion handler which wakes the service process back up again (5). The service process then completes its job and packages the reply into the `mb`, then as the last step in executing its reply (6) wakes the user process so that the user process may access the `mb` and proceed on its way [70].

The scheme cleanly manages two-way communication of information between process and provides assurances against any potential deadlock by preventing any potentially circular requests on shared data. The protected service process model does not require two-way

communication for every instance. The fire-and-forget nature of sending a job to a print service that does not need to report completion is an example.

3.1.4.2 A Region as a Protected Service Process

In this dissertation's simulation architecture a **Region** has five components:

1. A collision detection R*-Tree with a fixed physical extent of responsibility
2. An **EventList** for managing the event progression within the **Region**
3. A thread of execution
4. A thread-safe inbox for incoming communications
5. An outgoing communications capability

Regions leverage the protected service process concept of Section 3.1.4.1, but simplify the interaction by limiting it to one-way, as illustrated in Figure 3.18.

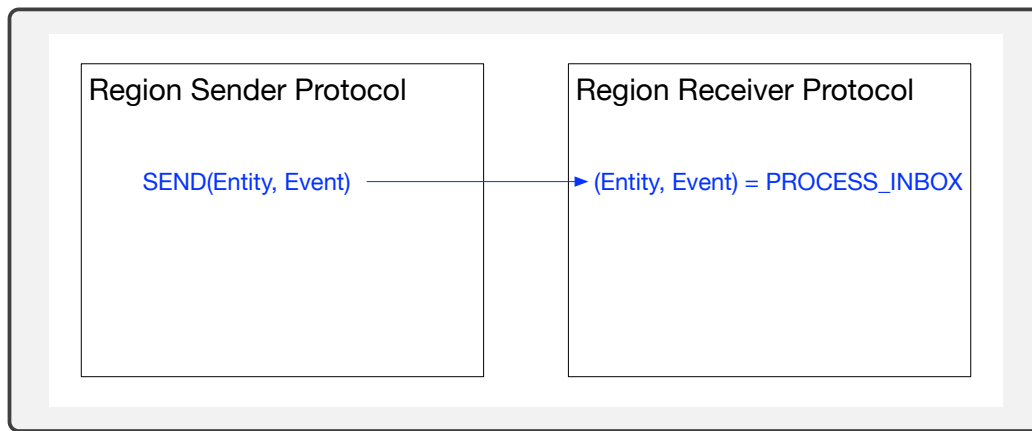


Figure 3.18. Inter-Region communication protocol.

One-way is adequate because each **Region** computes independently and does not share any data or entities with other **Regions**. This one-way no-shared-data communication preserves the deadlock avoidance property of the protected service model while eliminating the need for the sender to sleep to maintain sequential access on a shared object such as the **mb**.

The fixed **Region** extent is also the extent for the R*-Tree root node. Any entity actions that occur entirely inside the extent are free of requirement to use synchronization because

all code interactions occur only in the Region's thread of execution. The R*-Tree is a recursive data structure so each Region root-node is also an intermediate node in the simulation's WorldTree. The WorldTree insertion process relieves individual Regions from requiring knowledge of other individual Regions, further supporting independence of operations. Algorithm 1 defines the Region behavior loop and interaction with the inbox and EventList.

Algorithm 1: The RegionalSchedule, a wall-clock synchronized event scheduler.

```

1 begin
2   oneFrame_ms  $\leftarrow$  frame length in ms
3   while running do
4     do
5       while inbox.hasContents() do                                /* process inbox */
6         e  $\leftarrow$  inbox.remove()
7         EventList.add(e)
8       maxSleep  $\leftarrow$  delay duration from now for synch with wall-clock
9       if maxSleep < 0.0 then
10        rework()                                                /* see Algorithm 2 */
11        if maxSleep > oneFrame_ms then                            /* wall-clock sync */
12          SLEEP(maxSleep)                                       /* also wakes on inbox.add() */
13        while maxSleep > oneFrame_ms
14          CurrentEvent  $\leftarrow$  EventList.pop()
15          CurrentEvent.evaluate()

```

The Region's EventList is responsible for the local progression of the simulation as described above in Section 3.1.1. It contains the loop code structure of Algorithm 1, supporting the Region's analogous behavior to the protected service process. The inbox is a first-in first-out (FIFO) lockless linked-queue where an add may be safely initiated from any thread in the simulation, and remove calls are only executed by the owning Region to empty the inbox and schedule the incoming events to the Region's EventList.

During event evaluation, the collision detection R*-Tree is responsible for determining if an entity is extending beyond the boundary of a Region and crossing into another. This is illustrated in Figure 3.19, Case-A and Case-B each trigger the outgoing communication

to the WorldTree which recursively determines the destination Region(s) and places messages in the appropriate inbox(es). Transfer of responsibility to evaluate an entity occurs via the inbox once the entity is fully in the gaining Region, illustrated as Case-C in Figure 3.19.

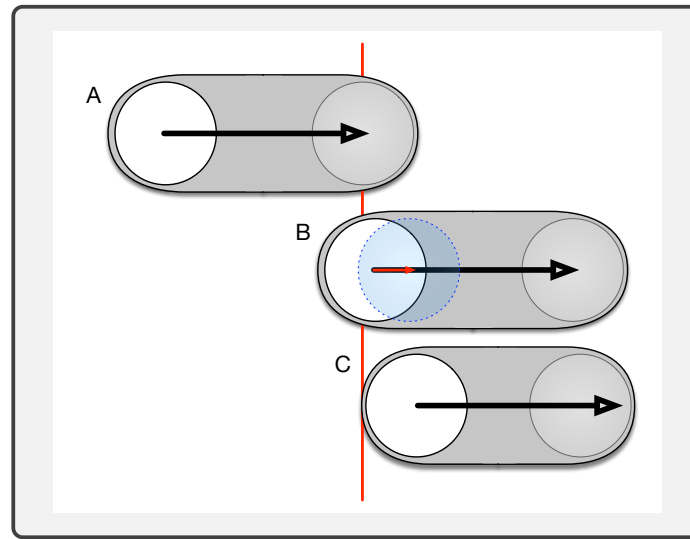


Figure 3.19. Entity movement between Regions.

An *in-between* situation arises though as an entity begins to extend into a neighboring Region, but not yet departed its originating Region. This *liminal* status requires special handling to maintain the one-way no-shared-data communication requirement.

There are no other interfaces between Region's and the threads executing the Region's business. Any entity actions that occur entirely inside a Region are free of requirement to use synchronization because all code interactions occur sequentially and only in the Region's thread of execution. These design constraints positively prevent code executing in one thread from accessing the execution state of another thread, thus eliminating a need for sharing data in threads and the otherwise required multiple mutual exclusion locks that are necessary with that requirement.

Because the fixed Region extent is the extent for the R*-Tree root node, and a Region root-node is an intermediate node in the simulation's WorldTree, it should also be possible to specify intermediate WorldTree's where more localization of geometrically close Regions

can enhance efficiency compared to always starting at a single WorldTree root node and navigating downwards when that extra navigation provides no possible value.

3.1.4.3 Liminal Entities, Entering a New Region and Thread

An incoming entity's first appearance in a new Region's inbox occurs during an event that initiated in the originating Region. The movement of this transitional event is represented in the standard manner using a MotionWitness wrapped around the SMA defining the motion interval. Because it is necessary to not manipulate the entity as shared-data in both Regions, the Liminal exists as a status or wrapper to allow the entity to appear in the R*-Tree using all the existing process, but prevent the incoming Region from modifying the entity or scheduling it into the EventList.

Liminal entity transfers fall into the two cases depicted in Figure 3.19. Case-A is whenever the crossing entity does not complete the crossing during the scheduled movement, and in Case-B the entity does fully cross the boundary.

Case-A requires the entity MotionWitness to be inserted into the gaining Region's R*-Tree and evaluation for potential collisions during the movement interval. If there is no collision potential detected, the Liminal insertion is complete. If there is a potential collision a NarrowPhase event is scheduled for potentially colliding pair with the gaining Region's entity as the aggressing collider entity. This means the NarrowPhase event is not prematurely taking control of the Liminal entity before it is formally passed along by the originating Region, the Liminal entity is simply treated as the collidee.

With the scheduled NarrowPhase event on the EventList it is possible that the incoming entity moves fully into the new Region before the NarrowPhase event scheduled time, this situation is represented by the blue circle on Case-B. The originating Region explicitly reports this time to formalize the inter-Region handoff for responsibility of computation; no changes to the MotionWitness are necessary and the entity is unwrapped from its Liminal status. Because the past has not affected the present, the net effect is the gaining Region virtually treats the new entity as if it is now Case-C, even though the MotionWitness of Case-B is still the geometry in use.

Alternately, if the NarrowPhase event evaluates before the incoming entity moves fully into the new Region and a collision is confirmed, the resulting EvaluateContact event for the incoming entity is provided directly to the inbox of the originating Region, and the already

resident entity evaluates its `EvaluateContact` event normally. The originating Region is responsible for computing the collision response and sending the resulting `StartMove` to the `WorldTree` as before. This ensures that both any additional interactions with other entities fully in the originating Region are tracked and the resulting motion event is passed to all affected Regions. This must occur even if there are only abutting AABB Regions, since in 3D space there are corners where an entity might potentially be present in *eight* Regions simultaneously (although rarely).

3.1.5 Distributing Asynchronous Discrete Event Simulation (ADES)

At its core, distributing the simulation over a network requires forwarding local events to the networked participants, and maintaining the previously described procedures. As stated in Section 1.3.2, this dissertation is not a network protocol description or comparison, rather it is an application-level architecture and set of algorithms. Distribution does require a network which implies use of a protocol, but as long as the packets get to their destination, then the architecture and algorithms do the described job(s). Whatever protocol is selected to transport the packets, the basics of data transmission and error handling are managed and those choices are orthogonal to the application-level requirements for *Lag Erasure*.

While the tens-of-millisecond latencies expected over a network dwarf the microsecond or less latencies of inter-thread communications within a single machine, the established LCW-related handling in this dissertation's architecture is already designed to accomplish the basic latency hiding necessary for running networked simulation that do not use external inputs. Latency hiding does not account for potential variances in hardware and/or code that are part of the larger simulation. The two following design requirements provide a guide towards efficiently distributing the existing multi-threaded simulation techniques and provisioning the capability to maintain a consistent shared state across the network.

3.1.5.1 First Requirement: All Hosts Share Functionally *Equivalent* Code

The equivalence requirement is that each set of code provides results that test as equivalent, within designer-set tolerances. Functionally equivalent code is not identical code, there is no attempt to rule out heterogeneous platforms, or drive single-source for simulation software. It is a key aspect in the ability to maintain the consistent shared state provided by the latency hiding layer of the *Lag Erasure* architecture. If the code at each distributed host is able to generate an output within the shared-defined error tolerance ϵ that code may be considered

to be *equivalent* with respect to generating the same simulated outcome. This tolerance is the same one used during the assessment of whether connected hosts are maintaining a consistent shared state for an entity.

The utility of an ϵ is not ubiquitous, it only applies to function outputs, not inputs. The equivalent code that generates equivalent results does so from the same binary input, which is efficiently managed by the second constraint for distribution.

3.1.5.2 Second Requirement: Entities Have an *Owning* Host

One host is designated as the authority for an entity's inputs, and the updates that it promulgates are authoritative and provide the values that other hosts use to compute the follow-on event. This ownership requirement matches expectations based on existing distributed networking protocols also dealing similarly with ownership.

Distributed hosts often compute the same simulation Region(s), and such similarity is expected in a simulation with multiple aircraft simulators flying a common mission while the simulators are at different sites, e.g., F-35s at NAS Lemoore, HH-60s at NAS North Island, and F-18Gs at NAS Whidbey Island.

Every host computes all events within the Region(s) it tracks, but only entities that are owned by the host have their events transmit over the network as the authoritative version. Upon reception of an incoming authoritative event, the host assigns it to the appropriate Region and assesses if the authoritative version matches the locally computed version. If it does, no computational changes are required, and the necessary authoritative binary values are committed to the local copy ensuring that the next computation uses identical inputs as the owning host's version. This satisfies the first constraint's requirement that the same binary input is used when invoking the equivalent code. The re-baselining of the same binary input requirement is a safeguard from "equivalent" within error tolerance ϵ but binary differing inputs eventually contributing to results that drift away from equivalent following one or more function evaluations.

In cases where the authoritative version of the event does not match the locally computed version the local version must be superseded and replaced in the event stream and collision detection subsystem, this is beginning of the *rework* process. The magnitude of effort to effect rework differs from rollback via one event design element and two primary constraints

providing strict limits on the potential amount of recomputation necessary to restore proper causality, and neither constraint is available in LP rollback implementations:

1. Events that may generate follow-on events include a rework functionality which cancels any existing follow-on event from the event stream.
2. The *Lag Erasure* architecture does not use traditional unconstrained progression ADES Logical Processes (LP), rather the wall clock and LCW constrain forward progression through an entity's events.
3. The implementation of the LCW supporting speculative computation and transmission strictly limits the time window where any possible causality violations could occur to less than 2x the LCW in milliseconds. In cases where network jitter magnitude jumps beyond the bounds of the LCW, the magnitude of the excess jitter is added to the 2x LCW rework window limitation (this should be uncommon.)

3.1.5.3 Rework

The rework procedure is covered in this section, the process of the unconstrained general rollback in optimistic ADES is covered in Section 2.4.2.

The R*-Tree update to effect rework is straightforward because the incoming authoritative Leaf which is triggering rework is created as part of the process to determine the correct Region to assign it to. The new Leaf's startTime_{RWE} (ReWorkEvent) on the virtual timeline is the baseline of the rework window. Because speculative evaluation is used on the sending end, the sending-end computation is accomplished at the wall clock time of $\text{startTime}_{RWE} - \text{LCW}$. Similarly on the reception-end timeline the receiving-host events that are pulled from the EventList and speculatively evaluated can not have startTimes beyond the wall clock time of $\text{startTime}_{RWE} + \text{LCW}$. This limits the magnitude (typically in ms) of the rework window to no more than 2x the LCW in milliseconds.

If the LCW is set to 100ms representing cross-continental latencies plus jitter the rework may only include events that computed in the most recent 200ms. As rework events are evaluated they either supersede a follow-on event or they don't, and only event chains that include superseded events need further evaluation in rework, providing a precise manner to determine when to terminate rework. Rework event chains may never proceed further than the LCW into the speculatively evaluated future because nothing farther ahead than that

may be evaluated yet, providing a hard upper-bound on event-chain rework requirements. It should be obvious at this point that only a limited number of potentially interacting events may execute in this $2 \cdot \text{LCW}$ time window, indicating that the speculative progression constraint associated with the LCW is providing a strong defense preventing unconstrained volumes of rework.

In the unfortunate situation where network transmission difficulties result in an event arriving later than the expected reception interval, the hard upper-bound on event chain rework requirements still applies, never further than the LCW into the speculatively evaluated future from the receive time. The start of the rework window is farther into the virtual past due to the external transmission problems, and as long as exceeding the LCW is rare this situation should not cause efficiency problems.

Lag Erasure is not a magic solution for a bad network, nothing is. The *Lag Erasure* procedure is able to erase lag associated with jitter creating greater than LCW network instabilities, but too much excess latency creates unavoidable negative effects on the proper and/or perceptual continuity of the simulation data stream. Simulation motion jerkiness is an expected result of jitter introducing too much excess latency, and at some magnitude way-too-much excess latency is catastrophic to all forms of real-time networked simulation.

3.1.6 Speculative Evaluation in Simulation Example—Boids, No External Inputs

This Section examines the simulation of the Boids flocking example introduced in 1987 by Reynolds [71], and used by Steed and Oliveira [2] in their textbook's analysis of how latency prevents consistent states between networked clients. As a useful simplification of the Boids flocking algorithm there are three rules: 1) avoid collisions with flock-mates; 2) progressively match velocity to flock-mates; 3) progressively maintain proximity to nearby flock-mates [71]. This provides background to the coverage in the following subsection which rests upon (but does not reiterate specifics of) the Boids algorithm.

3.1.6.1 Networked Boids, Before use of Speculative Evaluation

Steed and Oliveira [2] describe a basic networked implementation of Boids where two distinct flocks ($Flock_A$ and $Flock_B$) are hosted on two different clients using a data push model for communications where:

Client_A generates the state for $Flock_A$ at time t , and *Client_B* sees that state at time $t + t_{network}$, where $t_{network}$ is the time it takes to send information from *Client_A* to *Client_B*. This is known as *network latency*. In this case it is arguable that *Client_B* does see the same state as *Client_A*, but delayed in time. . . . In the data sharing model, we effectively have two data push models operating in different directions. . . . Thus each client see each other's state delayed in time. . . . *Client_A* never sees a rendering of both flocks that is exactly the same as a rendering of both flocks on *Client_B*. This is a fundamental property of NVEs and other real-time systems if they run in what is called asynchronous mode, meaning they evolve their state without waiting for the network. We can attempt to force them to present the same state, but only by inserting delays into the system to compensate for $t_{network}$. [2]

In the analysis of this situation and the state differences it creates, Steed and Oliveira [2] comment

Despite this discrepancy between the two clients, is the simulation now broken?
The answer is no: the simulation is not broken, although its behavior might be a bit more noisy and less holistically consistent. [2]

While initially true on the surface, this analysis requires a concealed assumption and expectation such as an explicit acceptance of uncontrolled variability in the simulation. At the minimum, the behavior of the multiple flocks is non-deterministic and not repeatable. For the concealed assumption to remain true there must be no expectation of ever achieving repeatability in the simulation, and this dissertation rejects that acceptance of uncontrolled variability and non-determinism in the simulation.

First, the expectation that the difference in boids positions are not problematic is a poor starting point for any generalization about simulations. For humans in a NVE where the

boids are window dressing (i.e., non-critical NPCs) this may be acceptable, but what if the boids are primary-player simulations of drone swarms and the simulation is exploring coordinated defensive actions against these swarms by electro-mechanical systems where each client is simulating based on its own separate sensor system? Precise repeatability may be a matter of life or death. In the latter simulation the randomly different and unrepeatable boid flock states are catastrophic to the usefulness of the simulation and understanding of real world potential behaviors of the defensive system because the collective picture of the flock is NEVER correct. Comprehensive testing and validation of such systems cannot be achieved.

Second, more subtle and specific to the Boids example is that Boids is a special algorithm that has the property of always providing a force towards regrouping the flock towards nearby flock-mates. Boids behavior never diverges so badly that it blows up and becomes unrecognizable, because it is constantly regrouping the overall dispersion of the flock as part of its intrinsic algorithm. With latencies of dozens of milliseconds the individual boids may eventually end up in dramatically different parts of the flock, yet the flock may average out to occupy a somewhat similar blob in space. Using the metric of similarly localizing blobs in space also does not generalize well across the domain of simulations, even if it is acceptable in some entertainment applications. Finally, it is algorithm specific that accepting the difference is because it seems to be not-that-different even makes sense. This means the second concealed, and silently application-specific assumption, is often altogether false when applied to any other scenario.

Third, the statement about evolving state in asynchronous mode hides an assumption that the evolution computation progresses faster than the communications may manage, which is confirmed by the stated need to insert delays into the system as the only means of compensation. In other words, not only are distributed instances of the common model always different and never identical, but the only way to compensate is to evaluate the models painfully slowly.

Finally, all of the graphs and figures supporting the Boids discussion as provided in Steed and Oliveira [2] text are constructed in a manner that visually depicts and implies time-continuous motion, *which is impossible if the states are transmitted as snapshots over the network.* This issue is independent of any computation method used by the implementation.

Continuous graphs provide information in the description and analysis that does not exist within the depicted simulations and therefore visually provides a platform for unintentionally minimizing estimation of the effects that network latency is having on the actual overall simulation state.

These four items collectively only make sense if the simulation is assumed to be implemented using a short step-length synchronous time advance, and the purpose for the Boids portion of the simulation is background visuals rather than a portion of the critical path for training or data acquisition. Because distributed models often require deterministic repeatability in order to be useful, the background visuals only level of utility is short of the mark, better simulation methods must avoid these pitfalls.

3.1.6.2 Networked Boids Using Speculative Evaluation for Latency Hiding

While the Boids discussion in Steed and Oliveira [2] is missing implementation detail, programmers inherently understand that each of Reynolds' three rules [71] presented in the lead paragraph of Section 3.1.6.1 may be implemented by sampling the set of individual boid states at each time advance, and computing the resultant motions. How these resultant motions may be successfully generated by asynchronous evaluation methods without needing a short step-length synchronous time advance is not obvious to traditional simulation programmers. The following sections describe how boid behaviors may be implemented in an asynchronous time advance scheme.

3.1.6.2.1 ADES Boids Setup To implement an ADES version of Boids with adaptive steps that on average are much longer than the typical synchronous time step only requires considering the rules as longer term goals rather than a cycle of immediate evaluations on each time advance. Rules 1 and 2 are complimentary as per [71], meaning the goals are supportive of each other. These goals may be achieved by adapting the `MotionWitness` slightly to provide the equivalent of a proximity warning or sensor, and the `MotionWitness`'s linear approximation already provides a velocity queue to other boids. Using the nearby flock-mates `MotionWitness` velocity cues and extrapolating if the boid needs to close distance or not provides the means to meet Rule 3's goal. These sensors require less computation than the existing synchronous-step versions require because they execute less often and

simultaneously retain more predictive power via the `MotionWitness` projecting motion information into the future.

The first consideration is whether it is fair to use the `MotionWitness` of one boid as information that another boid may use to make its own future motion decisions. In the case of rule 2 velocity matching, the `MotionWitness`'s information is no different than using a data structure entry for another boid's velocity, and the forward projection to the next time step's location is naturally part of the process because that is literally the definition of velocity, change in position over time. The `MotionWitness` simply captures the same intent of a boid to go "there" over an interval in time and simultaneously indicates the potential volume of space that transit may affect, except "there" when computed adaptively tends to be farther into the future. Because "there" is farther into the future, the equivalent of multiple synchronous time steps, the ability to determine when near-future inter-boid proximity passes the too-close threshold is necessary as a means of preventing naïve boid paths from blindly creating unavoidable collisions.

The proximity sensor for rule 1: do not collide, is implemented by simply making the root sphere of a boid's BVH larger than the physical extent of its geometry. This way any time a boid's motion approximation `MotionWitness` intersects another `MotionWitness`, the evaluation for whether or not to schedule a `NarrowPhase` event is based on the MWs enlarged "proximity warning" spherical volume. What would be an volumetric over-estimation leading to a false positive for potential contact in a non-Boids simulation becomes the threshold check indicating a pair of boids need to re-evaluate and update their converging motions by shortening the time until the next `startMove` event, and this always happens earlier than the requirement for a `NarrowPhase` thus avoiding unnecessary detailed potential collision computations. In this way existing responsibility for the `NarrowPhase` event to determine if a `EvaluateContact` event is unaffected. Because boids are highly adept at avoiding collisions with other boids, completing the process requires tuning the magnitude of the BVH root expansion to provide the boids their necessary reaction time and maneuver space, thus avoid the need for scheduling a future-collision resolution `EvaluateContact` event unless the circumstance is somehow intentionally forced.

The "distance sensor" for implementing rule 3 is implemented as similarly to the collision detection tree update process. A tree query may be executed for which boids have

a `MotionWitness` overlapping an `AABB` wrapped around a boids volume-of-regard. The query works the same way the narrow-phase checks work, but return the full list of overlaps, not just the earliest overlap. This provides the list of boids perceptually close enough to perform a weighted analysis for time-distance to develop any necessary regrouping movement or acceleration component. The analysis result is not an immediate change to the new motion in evaluation, it provides the time for when the threshold triggers and the magnitude of the recentering behavior. Because this trigger time never extends the existing movement `endTime`, the “distance sensor” functionality does not require any collision detection re-evaluations for any event already resolved in the `R*-Tree`.

3.1.6.2.2 ADES Boids Operational Example Consider a training simulation in preparation for a large flock of migratory boids departing the Queen Elizabeth Islands of northern Canada. The simulation training event includes multiple human trainees, but before that happens the overall simulation software must be validated and verified indicating it is meeting its intended purpose and exhibits the expected behaviors. Controlled, repeatable inputs for this verification and validation stage require the future participants to be simulated by software agents which guarantees all inputs to the simulation testing sessions are controllable and auditable. For this example the simulation software also meets the “all hosts share equivalent code” requirement cited above in Section 3.1.5.1.

As the boids head south they fragment into smaller flocks and disperse across the east and west migrating to seasonal feeding grounds across mid-North America. As the boids approach the mid-latitudes a flock of raptors is waiting for them near the Great Lakes and a flock of eagles awaiting over the western Missouri River valley. Simple boids treat learning of these hunter flocks as they would treat a tree in their immediate path and the flocks further bifurcate to flow around these hunter threat areas as they continue on their journey.

In this simulated world there are farmers that want to protect their unharvested fields from the boids by employing high speed falcons that get one chance to intercept a boid before it must climb back to altitude and reset for another try. The farmers cooperate and use radios to direct falcons from coordination centers in New York and Los Angeles, splitting the responsibility for Kansas City per the respective state lines of Kansas and Missouri.

Scenario Setup. To support scenario computation the U.S. and Canada is divided into 5 simulation Regions: Canada is assigned as the first Region as there is little interaction in the early stages of the boids flight. The U.S. is divided into 4 Regions, 2 to the west of Kansas City, and 2 to the east, for this discussion the only boundary that is important is the extended north/south boundary between Missouri and Kansas which bisects Kansas City.

The Los Angeles coordination center hosts the western Regions and is assigned ownership of boids beginning motion within the two western Regions, and the New York center is arranged similarly on the eastern Regions. A central training center is assigned the Canada Region and holds ownership of all the boids at scenario start until they are handed off to Los Angeles or New York centers. Figure 3.20 illustrates these divisions of responsibility.

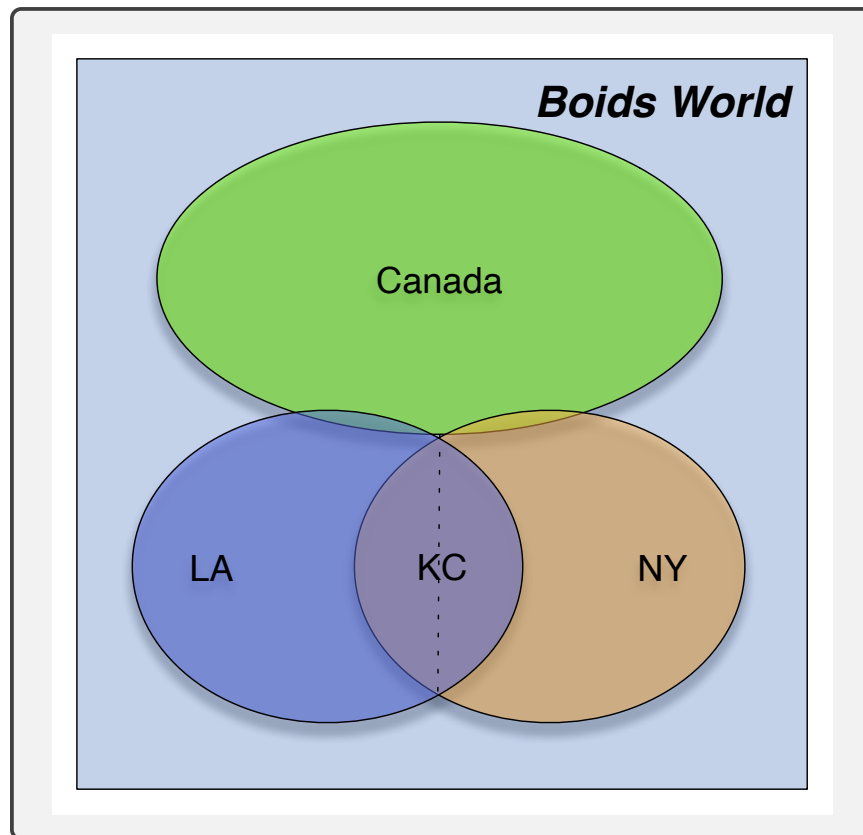


Figure 3.20. The Boids World division of responsibility.

The Los Angeles site actively evaluates the simulation of its two assigned Regions as well as the westernmost of New York's Regions, and similarly New York actively evaluates the

simulation of its two assigned Regions as well as the easternmost of Los Angeles' Regions. With reference to Figure 3.20, the left-half of the Los Angeles-New York overlap is owned by Los Angeles and duplicated by New York and the right-half of the Los Angeles-New York overlap is owned by New York and duplicated by Los Angeles. The overlaps between Canada and the U.S. Regions represent where events enter the U.S. Regions, but originate from outside the U.S. Regions.

The overlap in evaluation of the adjoining Los Angeles and New York Regions provides the common picture necessary to provide proper latency hiding across the simulation while reducing computational overhead at each site by not requiring computation of events which do not directly affect the owned Regions. The Canada Region also proceeds independently, only communicating events that initiate boundary crossings into one of the U.S. Regions. These north-south boundary crossings result in ownership handoff of each boid.

This interplay of independent and duplicated regional coverage with the associated computation assignments illustrates that the ADES latency hiding technique is not limited to a full duplication of the entire simulation everywhere in order to work. This supports control of information flow such that entities in some arbitrary Region are not able to react to entities they have no business to be aware of, and simultaneously increases scalability of the total simulation by inherently partitioning the simulation load in a way that not every host must compute every entity. As examined by Macedonia in [72], careful shifting of ownership can reduce long haul network effects for entities (e.g., boids) traversing long distances into other regions of interest.

Scenario Execution. At scenario start all of the boids are in Canada's Region and for much of the early portion of the simulation the boids flock travels in a static configuration. This results from the complementarity of rules 1 and 2 resulting in the boids settling into same direction and speed flight while maintaining their favored in-flock personal space. This is similar to how Pelicans or Canadian Geese maintain a flock V-formation with minimal changes while transiting hundreds of miles. The simulation maintains the motion efficiently with long duration MWs that the boids self-arrange in parallel directions during periods that the flock is not perturbed. These transiting motion events are only evaluated within the Canada simulation host. As the boids pass through a trigger object placed in central Canada,

the Canada simulation host transmits the triggered warning orders to the Los Angeles and New York coordination centers that those centers in turn use as prompts to scramble and deploy the raptors and eagles to their airborne staging areas.

The Canada simulation host is aware of the Region boundaries with the U.S. Regions and following the same procedures as described in Section 3.1.4.3 a boid with a BVH root volume touching another Regions boundary requires a Liminal entity representation to coordinate the boundary crossing. In the introduction of Liminal entities in Section 3.1.4.3, the Liminal status is managed locally via a simple R*-Tree entity update as both Regions are evaluated within the same host, the addition in the networked version of the simulation is that from within the owning Region the Liminal entity event status is transmit over the network to the incoming Region hosts. Because speculative evaluation is used to generate the event in the owning Region, and there is some amount of owning Region motion to accomplish before the entity first touches the Regions boundary, the event normally arrives well before it is even required to be added to the incoming Region's R*-Tree as a Liminal entity. Once this R*-Tree insertion is initiated by the receiving Region the entity handling procedures are exactly the same as presented in Section 3.1.4.3.

The only time a receiving Region should need to communicate back to the originating Region is if a collision with the Liminal entity occurs before the entity ownership is passed to the receiving Region. In this case the receiving Region issues and transmits a EvaluateContact event back to the owning Region. Locally, the receiving Region is able to compute and post an updated Liminal entity to its local R*-Tree, and once the owning Region completes the authoritative motion update and transmits the authoritative follow-on Liminal entity this locally computed version may be compared to the incoming authoritative version as previously described in Section 3.1.5.2.

Because the inter-Region communication process involves a Liminal entity on the receiving end, there is no ambiguity for whether or not an ownership transfer message should be cancelled or not in the rare case that an ownership transfer is initiated but a collision only detectable in the receiving Region occurs during the window after the transfer is transmit but before it is committed on the receiving end, effectively resulting in the messages to cross as they transmit the network. On the owning end, the order of events leading to cancelling a scheduled ownership transfer event is resolvable with singular timeline timestamps once the

Liminal-related EvaluateContact event is received. On the receiving Region end of the process, an incoming ownership transfer event contains the SMA assigned as of transmission time. If the receiving Region executes an EvaluateContact event in the interim, the Liminal entity receives a different SMA and this SMA difference allows the ownership transfer event to be discarded silently. This also prevents any ambiguity from the exceptionally rare circumstance where multiple ownership transfer events are subject to jittery network burps that rearrange the reception order. When the SMA matches, the SMA matches, execute the transfer; reject any mismatched SMA transfers as superseded.

Scenario results. From the above sketch of simulation operation, it should be apparent that when equivalent code is used on all geographically distributed sites the procedures yield equivalent results across the entire simulation. The importance of these equivalent results are that the results *are* the collective simulation state. Stated with more scenario-specific detail, entities in the two Regions containing Kansas City which are simultaneously and independently simulated by both the Los Angeles center and the New York center have equivalent states (per Section 3.1.5) in each center's representation. These entity states are comprised of the fixed entity attributes and the varying SMA. The SMA are directly comparable in 4D spacetime as they contain both continuous time and positional data. If the simulation designer wants to also consider some entity attribute as varying, such as fuel load, this still follows the same concepts and equivalence results.

Direct comparisons of entity states for entities simulated on multiple hosts may be evaluated using the metric and equations presented in Section 2.1.3, Equation 2.4 thru Equation 2.7. This process determines the percentage of time that the collective state of the simulation is not equivalent (or divergent) across all entities and all hosts. When this percentage is near zero it means nearly every state is consistent across the network nearly all the time. Any differences that have temporarily appeared in the simulation states due to adverse network conditions have *demonstrably not propagated in an uncontrolled manner, creating additional cascading divergence*. If divergences do propagate and cascade it is impossible to measure a near zero percentage of state difference across the entire simulation. So it is intuitively obvious that a near zero percentage of state divergence during a simulation session means that the speculative evaluation latency hiding algorithm must have maintained

a consistent shared state for all entities via normal network operation, and reconciled any temporary state differences that resulted from network instability and jitter.

Providing a baseline of equivalent entity state across the entire distributed network is the necessary capability that enables the addition of algorithms to achieve *Lag Erasure*. This baseline is also a useful constraint in preventing large amounts of rework when external inputs are added to the mix.

3.1.7 Speculative Evaluation and Latency Hiding Limits

The latency hiding technique component of the *Lag Erasure* system is not universally unlimited in scope of applicability. In Section 1.3.2 it is stated that “the *Lag Erasure* architecture describes a software architecture targeting simulation models and events at human scales intended to support real-time human/equipment/world-in-the-loop applications. . . . This research does not provide a general distributed simulation solution for arbitrary situations and models at arbitrary computational rates . . . Despite not being a general solution, *Lag Erasure* enables an extensive design space.”

The concept of *Lag Erasure* is new and not yet fully explored, so it should not be surprising that every nuance of the limitations may not be detailed here. The intuition though is that the limitations for usefulness are directly related to the relationship between expected latency and the necessary event generation rate of the simulated entities. If an entity requires multiple event generations or state updates in the duration of the expected LCW, then the ADES simulation is forced into a potentially perpetual state of catch-up and forced rework, and if there are multiple entities with this high-frequency update requirement it is possible that the effective consistent shared is not attainable.

The scoping concept of “human scales” is that the focus is on the support of “the real-time human/equipment/world-in-the-loop applications.” When the simulation models are considered there are potential simulation aspects that mismatch abstraction level of the goal of training. For example, consider a detailed silicon-level simulation module for braking activator circuitry to be used in a tank procedures simulator. When executing as a training simulation with a human or AI participant, that simulation of silicon-level braking activator circuitry is irrelevant, it is the black-box-brakes-behavior that is relevant. Simulation models

meeting the appropriate level of abstraction for live real-time simulations is not a new thing, this is the intent.

3.1.7.1 False Limits

If high frequency updates are not generally compatible with the presented ADES *Lag Erasure* techniques why bother at all? Aircraft simulators include internal simulations of an automated flight control system (AFCS) that have 1000Hertz (Hz) update rates, robotic vehicle sensors contain internal computational software running at similar rates, this must make them incompatible! No, it does not. This type of argument removes too much context and creates a straw-man that does not honor the advantages that time-continuous ADES state representations provide to support these situations.

3.1.7.1.1 Continuous State is Powerful Continuous state is powerful, and all of the states that a designer wishes to provide as continuous are available as continuous. Even table-lookup environmental values may be constructed as continuous, while step-function-like changes may not be differentiable at the step-boundaries, the continuous values still cover the entire axis they are continuous over. This means that all of the sensor inputs that the previously mentioned high frequency AFCS or robotic control system consume may be provided via continuous functions that provide high-fidelity data as fast as the unit wishes to consume it. This indicates that not every type of high update rate simulation is incompatible with ADES and by extension *Lag Erasure* simulation techniques.

3.1.7.1.2 High Frequency Updates in Black-Box Components Considering high update rates as automatically counterfeiting ADES latency hiding simulation techniques also requires an invalid deeper concealed assumption, an assumption that a high update rate component such as the AFCS is running as an independent simulation entity exposed to and independently communicating with the larger simulation. In reality the AFCS is just a black box component embedded inside the larger black box abstraction of the aircraft, which the pilots are flying. As a black box component of the simulated aircraft, the AFCS is not required to be issuing events across the simulation network. Additionally, a high frequency update AFCS system does not have full control authority for the aircraft at the full update rate, so the black box cannot even generate gross aircraft motion inputs at the black box's internal high update rate. To do so would cause a serious fatigue limit on a real

airframe, so it is not a valid consideration for a simulation either. An AFCS with an internal high frequency update rate normally has a low-frequency update rate as its aircraft control interface which closely matches the human's control rate capabilities for generating flight control inputs. It is this flight control input update rate that is the key rate for comparing to the LCW duration to estimate if an effective consistent shared is attainable.

3.1.7.2 No External Participant, Faster Than Realtime Simulation

For simulations that do not require human or equipment-in-the-loop participation, the simulations may be executed at faster than wall clock rates and the *Lag Erasure* techniques remain useful if the scope of the simulation includes evaluation across distributed sites. The latency hiding techniques are intended to enable simulation scale simply not attainable in individual cluster installations. Where exactly the most useful tradeoff breakpoints fall between locally efficient cluster-based simulations and latency-affected geographically distributed simulations are a topic for future work.

3.2 ADES with *Lag Erasure*

The coverage in Section 3.1 provides consistent distributed shared state for simulations that do not contain external actors outside the boundaries of the computations that comprise the simulation. This capability provides the baseline which helps constrain the scope of rework necessary when external actor inputs are added to a simulation.

Equipment-in-the-loop simulations have large variability in the potential agency possessed by the equipment within the simulation. At the low end there might be an underwater listening sensor that is programmed to transmit simple binary codes over a tether upon sensing certain programmed or hardwired inputs; this example does not have much direct influence on the larger simulation state but it could be tested for integration into larger system operations by feeding it simulated sensor inputs as it sits on the bench, and other simulated entities consuming the listening devices code transmissions. Extending the concept is useful for tank or bench testing an autonomous underwater vehicle (AUV) used for research, survey or surveillance. The AUV may be exercised with safety of navigation related inputs and presented sensor inputs that should trigger required investigation data gathering modes. Further extension into testing AI imbued autonomous vehicles for deployment in the real

world is a natural evolution and where the scope-of-regard and responsibility scales to the level where simple local cluster simulation may no longer be viable. It should be apparent that further increasing the scope-of-regard and responsibility for larger scale AI training and/or human interactions in complex operations further stresses the capabilities of delivering a useful simulation and eventually requires additional distributed sites to collaborate in generating the necessary scenario breadth and fidelity to accomplish the training task.

This large-scale type of training evolution is already a routine occurrence, but it is executed without the ability to remove the negative effects of lag (see Section 2.1.4) from the execution. Human common sense and ability to ignore lag effects on-the-fly and then debrief away any potential negative training effects are the current state-of-the-art in these large scale training exercises. If this is the most demanding simulation use then ***Lag Erasure*** would not be necessary, but there are simulation uses that are far less accommodating of imprecision and divergence.

AI is everywhere, and it is being integrated into new devices and vehicles with ever increasing rates of adoption. The complexity of emerging situations which entail multitudes of opposing autonomous platforms and human participants is driving efforts to press AI into the decision space engaging with these scenarios. When training an AI that is not already human equivalent in common sense the standard of “human common sense and ability to ignore lag effects on-the-fly, and then debrief away any potential negative training effects” is not viable. The negative training effects of lag-driven artifacts in the presented sensor data also corrupt any autonomy-related or AI training processes with non-physical and unrealistic events. Debriefing those away is not an option, if lag-driven artifacts create negative training it may be excruciatingly difficult to rectify. Taking control of the flow of training data presented and ensuring proper causality to the maximum extent possible is necessary and the ***Lag Erasure*** simulation techniques provide this capability.

3.2.1 A *Lag Erasure* Example—Boids, With External Inputs

Now that the simulation testing, validation and verification described in Section 3.1.6 is complete, the training scenario is ready to add human participants in Los Angeles and New York centers as well as multiple falconers associated with the raptors, eagles and falcons. Each actor has accurate tracking and periodic sharing of data. For each training evolution there are dozens of live participants scattered across the continental U.S. at their normal

posting locations. These remote participants are all operating a local flight simulator which simulates its own Region and adjacent Regions.

3.2.1.1 Lag Erasure Simulation Setup Supporting Use of External Inputs

While this example could further subdivide the Region(s) to reduce the computational load of entities, the geographically distributed flight simulators never interact directly, therefore would only add a layer of distraction because a Region is a Region whether labeled as so or called a sub-Region. All simulation Regions are defined as subtrees of the overall simulation R*-Tree and the Regions at scenario start are fixed, meaning all hosts hold an identical copy of the overall R*-Tree map of Regions and that tree is responsible for sorting and determining the routing of events between subtree's, both locally for multi-processing and via the network to distributed hosts.

The addition of inputs is an important distinction because real-world actions of an external actor at one site via some I/O device is plotted onto the virtual timeline, but that event is not able to be speculatively transmitted to the other distributed hosts in the simulation. These externally triggered events are considered to be a surprise to the rest of the simulation, and that surprise is always learned of time-late. As per the old saw, *bad news does not get better with time*, this surprise needs to be incorporated as fast as computationally able to minimize the overall effects on simulation causality and doing so constitutes the erasure of state divergence that may be perceived as lag by users.

To enable the quickest resolution of surprise, the simulation designer is presented with a standard tradeoff; increased bandwidth requirement vs. additional event arrival delay. The minimum delay occurs when the simulation events are designed to transmit an input-device state-change message immediately after a local host has determined that the input device has passed a threshold for creating a larger state change of the associated entity.

This early message is not much more than a “heads-up” to other hosts that a rework window begins with the virtual timestamp of the arriving event, temporarily pausing the forward simulation of the receiving host, there is no other additional state information contained so the packet bandwidth overhead could be considered to increase compared to waiting for a surprising event to be fully computed and transmit. This pause to allow rework to be as constrained in time-scope as possible is an additional limit of the independent progression

of an LP and provides an additional constraint on the maximum volume of rework that must be effected due to the arrival of the surprising input.

When the designer believes the input threshold crossing generates a significant computation, at least milliseconds worth, before a final event state update is generated then the heads-up event likely makes sense despite the lack of new state information contained in its network packet. When the expected time to compute the full results of the surprising event is measured in microseconds or less, it is most efficient to complete the computation and then transmit the full state event. Either way the event is tasked to be transmit over the network during the standard `EventList` scheduling and evaluation process.

Network communications are not fire-and-forget guaranteed, so a means of redundancy is required that is as supportive as possible of the real-time requirement of a training simulation with human or robotic external participants. Because each simulation host maintains a copy of the starting `Region` configuration, network communications may be transmit in a peer-to-peer mode to the necessary recipients, minimizing arrival delays that would be inherent with a mandatory central server maintaining an authoritative state, such as used for lag compensation techniques (see Section 2.1.7). Rather, a network-monitor server that maintains an accounting of all assigned host simulation state transmissions is implemented to provide systemic network delivery redundancy, not authoritative state control.

Using a variation of Fiedler's *Reliable Ordered Messages* [73], the network-monitor listens to each assigned host and proactively uses sequence acknowledgements (ACKs) to request re-transmission to itself for any messages that are missing based on the sequence number tracking. Other network simulation hosts operate and ACK similarly but they interact with the network-monitor rather than the original sender, this separates the redundancy-fulfilling communications traffic from the peer-to-peer network event traffic and insulates the simulation hosts from receiving multiple redundant redundancy-fulfilling requests while simultaneously continuing timely outbound event notification traffic. As with other simulation construction methodologies, simulation designers are still required to analyze the expected bandwidth requirements for the entities in the simulation and use that information to properly provision the network, in this case, determine the number and assignment of network-monitor hosts to simulation hosts.

3.2.1.2 Scenario Execution

As the full training simulation begins execution it follows the same flow as described in Section 3.1.6.2.2 with the boids flock traveling southerly in a generally static configuration. Eventually the flock splits with roughly half of the boids tending towards the east and half to the west, and both continue as before and pass through a trigger-volume placed in central Canada, per Section 3.1.6.2.2. This eventually leads the two coordination centers to scramble the eagles and raptors to deploy towards their respective airborne staging areas.

As each eagle or raptor flight simulator joins the larger simulation, they transmit their location to the other simulation hosts. The local flight simulators retain ownership of their aircraft and use the equivalent code of other simulation entities, such as formation members and eventually boids to display the relevant localized training event entity state to each trainee in their own cockpit. The majority of simulation events are managed and transmit exactly as described in the no external inputs procedures as delineated in Section 3.1.6.

An example of managing external inputs is during the formation flight of two eagles orbiting their assigned patrol point. As the flight lead begins yet another left-hand racetrack turn, the flight simulator processes the flight control inputs and computes the high fidelity in-simulator flight path. This high resolution CFD-derived simulator flight path is locally mirrored into the local host dedicated to connecting with the larger simulation, the local updates to this shared-code flight model are the external inputs to the overall simulation. As the local aircraft orientation and flight path registers a discernible change, a new flight control input event is transmit over the network.

3.2.1.2.1 External Events and Rework Upon reception by other interested remote hosts the incoming event is added to the appropriate `Region`'s `inbox` which also wakes the sleeping `EventList` scheduling loop at Algorithm 1's line 12. Upon waking the incoming event is added to the `EventList` and tested for whether or not the `EventList` scheduling loop should go back to sleep. Because the input-event arrives later than its virtual timeline time, the event immediately sleeping is not appropriate and therefore the rework procedure at line 10 of Algorithm 1 is triggered. Algorithm 2 depicts the steps used to resolve rework.

Line 3 of Algorithm 2 is technically not part of the rework process because the affected event is not being recomputed, but the condition in line 2 is the final determination for whether rework is indicated and the check is always necessary for late-arrival events.

Algorithm 2: The rework algorithm.

```
1 begin
2   if authoritativeEvent matches local version then
3     | assign authoritative binary starting values to local event
4   else
5     | reworkTime  $\leftarrow$  authoritativeEvent.startTime
6     | reworkList  $\leftarrow$  shallowCopyOf(archivedEvents  $\geq$  reworkTime)
7     for eachEvent in reworkList do
8       | entity.reworking  $\leftarrow$  True
9       | entity.evaluateEvent() {
10        |   if required then cancel existing follow-on event(s)
11        |   if required then schedule new follow-on event
12        |   }
13     | entity.reworking  $\leftarrow$  False
```

The *reworkList* of line 6 is the subset of archived (completed) events on the local host, in effect the *reworkList* is virtually pre-pended to the *EventList* allowing event analysis and cancellation decisions to be determined as if there is only one virtual *EventList* to manage. External to the rework process, but supporting it, the in-memory portion of the archive list is time-pruned by writing the oldest events to external storage keeping the archive list efficient for searching and generating the *reworkList*. This housekeeping is accomplished locally without any additional external communication necessary, the event sequence ACK tracking introduced in Section 3.2.1.1 provides the necessary data to determine the latest virtual time identifying when all incoming sequence numbers are successfully acknowledged and how much of the archive list should be committed to secondary storage. This is analogous to, but a different measure from Jefferson's *global virtual time* metric [31] used to track how far back to restore states in cases of rollback.

Jefferson's *antimessages* [31] are not necessary as standard event cancellations and knowledge of event sequence numbers provide the necessary information to locally implement functionality equivalents directly on the rework's virtual event list. In cases where follow-on events are self-scheduled by an event, there might be a computed chain of follow-on events in the *reworkList* that each need cancelling in-turn, this is the reason for the plural in line

10 of Algorithm 2. The chain of cancellations terminates when the first unevaluated future event is cancelled from the `EventList` itself.

3.2.1.2.2 Tradeoffs with Boundaries and Regions While the eagles and raptors continue their patrols the boids continue in their generally southward directions. As the western group of boids approach the eagle patrols in the vicinity of the western Missouri River valley, some boids are spooked hearing eagle calls and break in multiple directions. Los Angeles Center is simulating the boids behavior and the spooked boids each cause new `MotionWitnesses` to be placed onto the `R*-Tree` precipitating a wave of boid rule 1 collision avoidance actions successively generating `MotionWitnesses` that collectively begin to roughly align to the direction of the closest spooked boid and the rule 2 velocity matching comes into play. As boids that are further away from the maneuvers start begin reacting to boids beginning to open away from them the rule 3 grouping behavior triggers boids to follow along and re-close the flock and generate rule 2 velocity-matched `MotionWitnesses`. All of this action proceeds in the Los Angeles eastern `Region` server in waves as the distance related thresholds are triggered and the boids move as if in a simplified murmuration.

As the Los Angeles eastern `Region` server computes boids entering any of the remote eagle's `Regions`, the boid events are transmit to the appropriate eagle and the assigned network-monitor. This allows the eagle's simulation host to begin simulating the boid(s) locally with equivalent-code. In this case the boids are not under any external participant control, so the equivalent-code implementing the boids across the simulation generates identical states at every host until a particular boid begins reacting to the pilot controlled eagle actions (an external input). As boids enter the an assigned sub-`Region` adjacent to a sub-`Region` that is hosting an eagle, the Los Angeles eastern host transfers boid ownership to the eagle's simulation host. For non-external participant controlled entities or NPCs, this reduces the potential effects of intercommunications latency simply by co-locating the computation of entities that are close and likely to interact in the simulated world. This is an additional benefit of using the pre-existing spatial partitions to advantageously remove as much latency potential in the NPC interactions as possible, further limiting the potential scope of rework that could result from human-controlled eagle and NPC boid interactions.

The net effect is that eagle and boid interactions that are transmitted from the eagle's host back to the Los Angeles host do not require round trip interactions as the eagle host's

versions are authoritative. The likelihood that the eagle host computed boids trigger direct reactions in other Los Angeles host computed boids is small because any nearly all boids close enough to be reacting to the eagle are already in the same Region and under eagle host ownership, only boids close to the Region boundary on the Los Angeles eastern host could possibly be affected, and per Section 3.1.4.3 only interactions with those boids that cause eagle host boids to react before fully crossing the boundary into Los Angeles eastern host control requires the Los Angeles eastern host to create a feedback response to the eagle host.

This boundary-related potential interaction indicates a simulation designer tradeoff. Small Regions are good for partitioning space for multithreading, because each Region is assigned a single thread to execute the collision detection R*-Tree operations. If appropriate hardware resources are available this granular threading is advantageous toward scaling the total number of moving entities the simulation may manage. Conversely, smaller Regions create the need for more communication when entities move across regional boundaries. Within an individual host the boundary crossings are managed by message passing within the same memory space, but when small regions are associated with more hosts the inter-host message passing happens over the network and is roughly a million times slower to resolve. The generic relationship between boundary and volume is that boundary increases as a square function while volume increases as a cubic function, illustrating that larger regions have more room for entities to move about before approaching a boundary and necessitating cross-boundary communications. Where the tradeoff lies differs depending on the simulation, determining general design rules-of-thumb are beyond the scope of this proof-of-concept dissertation and fall into the future work category.

3.2.1.2.3 Shared Objects The eagle flights maneuver to engage the boids and the majority of interactions are managed as described above, with a lone eagle hot on the tail of a small flock of NPC boids maneuvering about each other and fleeing the single eagle. The wing-mate's intent to join on the lead-eagle's left-hand side and position for engaging the flock's left-most boids means that both eagles and the boids are soon to be collectively reacting to each other. This situation is devolving into an eagle, a wing-eagle, and a flock operating within the same defined Region with the eagle lead "owning" the boids computation. This closely coupled set of interactions is exactly the circumstance where lag effects are at the

worst, potentially creating divergent worldviews between the eagles, differences which may lead to incompatible decisions and unrealistic actions.

The beginning of the end of this mini-flock commences with the left-most boid closing the remainder of the flock, holding a flight path with a rightward drift compared to the others. The lead-eagle's host has computed this result, determined the associated `MotionWitness` for the motion and transmit it to the wing-eagle who is focusing on this particular boid as it attempts to both maneuver into attack position and take a safe formation position. As the wing-eagle stabilizes in position off the lead-eagle the boid computations trigger a decision threshold based on proximity of the predator lead-eagle, and the two center-most boids turn hard in opposite directions fleeing the advancing predator. Because both the lead and wing-eagle's simulation hosts are running equivalent-code, both simulations generate the same boid results at the same wall-clock time, meaning both human pilots receive displays with an identically-reacting flock-split of the boids at the same time.

The Lead Eagle as Viewed by the Wing-mate. The synchronization of the distributed state means the next maneuver by the lead occurs from a common starting state. The lead-eagle pilot chooses to follow the left-breaking sub-flock and maneuvers to follow the right-most boid of the new sub-flock in its left-and-descending turn. In a lagged simulation this may generate a catastrophic situation with the lead-eagle breaking directly into the wing-eagle's flight path while the wing-eagle is still presented a stable flock and lead to their right. A mere tenth of a second delay in the wing's ability to react may end with both eagles in-extremis, resulting in an inescapable mid-air that may end the training simulation for both of them. An invalid shared state serving as the primary factor in terminating some participants roles in a large-scale training simulation is a showstopper-negative event that is not acceptable.

The shared-state of the boids and the equivalent code that is maintaining that shared state ensures the wing sees the same breakup of the flock at the same time as the lead closes on its quarry. Because simulation states are the same on both hosts the wing-eagle pilot is able to understand exactly which boid their lead is closing on so that when the flock breakup begins the wing-eagle is already aware that their lead may break-left as well. The ability to make simultaneous separate decisions based on the shared state means the wing is able to

decide to also begin their leftward break on a timeline that ensures they have the ability to safely under-run the expected lead-eagle's turn into them.

This is exactly how the simulation plays out, the lead making a decision and a flight control input while simultaneously calling their break turn over the radio. The audio channel uses the same network communications trunk as the simulation events so there is no magical extra advantage to the audio stream's arrival at the wing-eagles headset. The lead's initial hard turn flight control inputs are integrated by the flight simulator hardware indicating to the lead's simulation host that a substantial flight path change is coming. This indication triggers an event transmission that does not yet contain a significant flight path change, but along with the small initial change as the eagle's large inertia catches up to the aerodynamic effects on the flight surfaces, the significant flight control deflections are also communicated.

The associated network packets arrive at the wing-eagle's host one simulator bay over in the same building within a handful of milliseconds, in this case well less time than the LCW which is set for compensating cross-continental latencies. The lead's control inputs occur after the boid's flock-split actions so despite the late arrival at the wing-eagle's host, the only event that needs changing is lead-eagle's previous motion event and that consists of truncating the `MotionWitness` to the time of the control input and laying in the associated new rates. This event application does not yet have a significant change in the flight vector or eagle attitude, the differences are a couple degrees of orientation yielding a visual difference in the lead eagle's nose position of a handful of centimeters over two or three display frames of relative time. In an absolute sense these differences are near un-detectable by the wing pilot who has no absolute reference points to compare against, and who has high mental task loading from the formation flying execution and beginning to formulate their mental plan for the earlier mentioned under-run maneuver.

As the wing pilot places more of their attention on the lead eagle they see the building large control deflections in progress and this mentally aligns with their real-world expectations of close formation flight and the developing scenario. From this point on the lead's host is sending motion updates on leads maneuver that reflect diminishing control inputs as lead's eagle orients in the way lead desires. These updates generate the `MotionWitnesses` which inform the wing-eagles display system. It is illustrated in Figure 3.21 that as the lead's flight control inputs stabilize, the successive `MotionWitnesses` (Figure 3.21's traces 1 thru 3)

are ever closer to the actual flight path and as lead relaxes the pull into the turn the input's changing generated acceleration yields projected motion on the inside of the turn (Figure 3.21's trace 4).

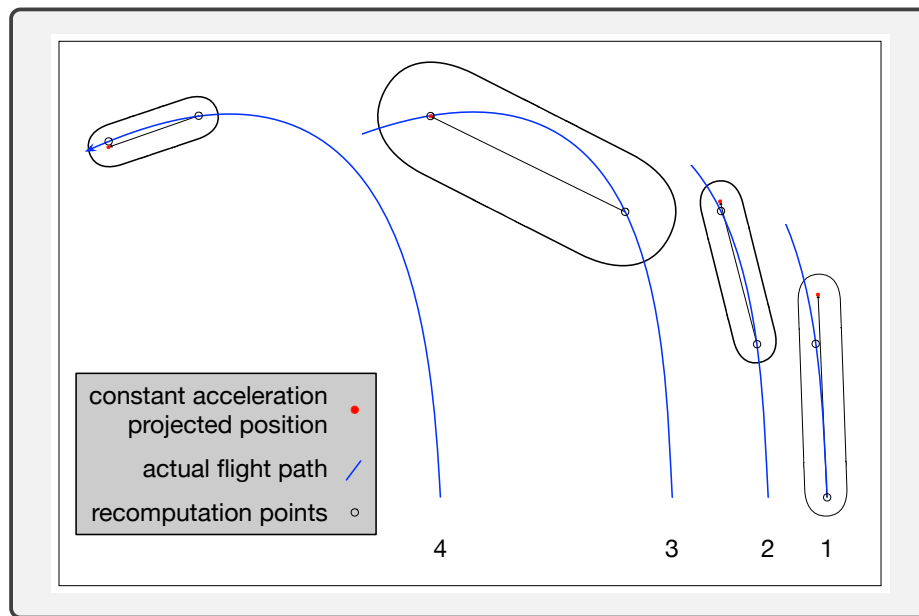


Figure 3.21. Succession of the lead eagle's MotionWitnesses.

These MotionWitness representations of the lead's flight path are not dead-reckons (per Section 3.1.2.3) each MotionWitness update contains the latest position and rate information necessary to approximate the eagle's continued motion should there not be additional flight control inputs. As lead rolls into and out of the hard turn the generated accelerations on the eagle are changing in response to the pilots changing flight control inputs. The flight model in the lead's simulation host uses the flight control inputs communicated from the flight simulator to determine when accelerations are different enough to require an update to the flight path. This may seem similar to how a DR determines it is time to send an ESPDU, but the mechanism is different. The trigger for an MotionWitness update is not determined by comparing the stale DR position and the current correct flight path then looking for a large enough difference in position or orientation to trigger a new ESPDU, rather in *Lag Erasure* the flight control input is processed to determine the resulting accelerations and when the

acceleration delta is large enough a new `MotionWitness` is generated. This causes the current `MotionWitness` to be truncated at the time the new `MotionWitness` is to begin. Because accelerations (second-order values) rather than positions (first order values) are used, the recompute position is not able to diverge much from the correct flight path before a new `MotionWitness` computation is indicated. Also, because the accelerations are directly dependent on the control inputs, and change of position is derivative of the acceleration acting on the entity, both in the simulation models and in reality, using changing accelerations for determining motion recomputation intervals yields more accurate tracks which generate less positional divergence in the event of rework, compared to DR techniques.

A degenerative operational possibility in the eagle simulator is that a pilot rapidly moves a control back and forth, rapidly, so fast that the adaptive algorithms are overwhelmed with rapidly changing acceleration inputs. In a real world flight simulator this would likely result in the simulation instructor immediately shutting down the simulator cab motion so that the hydraulic rams do not get jammed from being abused by the errant pilot, and an exceptionally poor debrief to follow. So while this degenerative situation exists, it falls far enough outside the normal operational envelope that it is not problematic in actual use³.

The degenerative version of rapid control motions is also different from the large and somewhat cyclic control inputs a pilot may use at low speeds such as landing in a turbulent environment. In this case the stick-stirring appearance of the inputs is because in this flight regime it requires more control deflection to get a smaller airframe response, and the pilot is reacting to more than just the smooth flight path of the simulated aircraft. The flight simulator itself resolves all those larger inputs in its CFD computations, and the resulting minor attitude changes which are not changing rapidly are passed along to the simulation host the flight simulator hardware is connected to.

The Boids Shared Between Lead and Wing-mate. With the lead eagle maneuvering to follow the boids making a defensive turn, the boid senses the continued proximity and executes a follow-on defense juke to the right. These maneuvers are computed at both the

³In a game if any player thinks they may gain an advantage via lag for doing something like this the developer may simply penalize the malicious motions by not allowing this type of control input to lead to any form of task success. The training feedback would be immediate and consistent, players may not successfully complete a task on any shared or potentially-shared entity via this form of illogical motion.

lead and wing-mate simulation hosts with the lead-host owning the boid's authoritative state. During intervals where neither of the eagles are close enough to a boid to precipitate a reaction, both the lead's and wing-mate's simulation hosts maintain consistent states for the NPC boids per the procedures discussed above in Section 3.1.6.2.2. In instances where the lead's motions cause a boid reaction but lead does not generate a new active control input for the duration of the LCW, the rework procedure has the opportunity and time to heal the states of the boids on the wing-mate's simulation host using the procedures introduced previously in Section 3.2.1.2.1. But what about the wing-mate now on the right side of the formation closing in on the far right boid? This boid is authoritatively maintained by the lead's simulation host, not the wing-mate's, so how is the delay in input to the authoritative state managed?

To begin, the lead's representation of the wing-mate's eagle is maintained to be close to actual, and have any temporary divergent state healed following the same procedures as discussed above in Section 3.2.1.2.3. Also, it is not realistic to think that when a pilot is completing final maneuvers to engage a shared NPC entity that the final maneuver suddenly becomes a violently oscillating set of control inputs that make it difficult for the *Lag Erasure* procedure to keep up with the rapidly changing acceleration and state updates. This natural tendency towards predictability and smoother motion adjustments when attempting to physically interact with an object is especially useful when the object of interest is shared between users on separate simulation hosts. The smoother motion adjustments lend greater predictability to the remote host's local computations of remote host expected motion. This is entirely compatible with the training mentality of "slow is smooth, smooth is fast," specifically because the smoothness provides an enhanced ability for remote simulation hosts to project motion forward for both remotely owned entities and the local entities and have those projections be reasonably accurate.

The accuracy of local and remote host projections does not need to be binary-exact for every motion projection to allow *Lag Erasure* to properly resolve any small differences within the threshold of the LCW on the overall simulation virtual timeline. It is important to always remember that all simulations are wrong, but some are useful⁴. When the binary value compatibility of two floating point numbers gets a dozen digits into the mantissa, the

⁴a paraphrase of the aphorism commonly attributed to George Box: "All models are wrong, but some are useful."

numeric scale is getting into the nano-realm of International System of Units (SI) values. As an example, if the simulation model is determining success or failure of a human to touch or grab an object based on nano-thresholds the model is both wrong and not useful, how could it be useful if it claims 100 percent chance of failure dependent on whether a real human might be slightly dehydrated making the finger imperceptibly thinner? Or alternately is somewhat bloated? Do either of these normal variances change the fingers micro-shape enough that a model should always determine non-success? No, not if the model is useful. And this directly maps onto the small variances that the *Lag Erasure* rework procedure is resolving for entity movement and orientation. The critical aspect for the simulation to manage are the motion and sensor thresholds that the model requires to be useful, and in human-scales these thresholds do not require full binary bit identically to the least significant bit of the mantissa⁵.

Continuing, once the positions and orientations are maintained closely and consistently enough that the model may make useful determinations across the shared state, the reactions of the NPC entities are generally consistent. The thresholding mentioned in the previous paragraph reinforces this as the thresholding reduces the opportunity for radical slope changes caused by low-significance bits rounding off correctly but inconsistently with the expected differential equation evaluation. In the rare cases that a NPC event threshold is validly tripped based on results of rework, per the discussion in Section 3.2.1.2.3, the NPC does not have the opportunity to make large obvious state changes that contribute towards making corrective movement appear herky-jerky.

The end result is that the lead-eagle's simulation host treats the shared object representation of the wing-mate in exactly the same manner as the wing-mate's simulation host treats the shared representation of the lead-eagle. The effects on the NPC entities managed on remote simulation hosts due to transmitting external control inputs for local entities are managed in the same way as well, but the resolution is simpler than needed for the externally controlled entity as the NPC does not require resolving its own non-existent external inputs during its updates during rework, and it is often possible to require NPC reactions to integrate sensor inputs for durations longer than the LCW, further insulating the rework procedure from creating large follow-on update requirements.

⁵This makes classic binary lockstep network protocol comparisons irrelevant, and opens the simulation envelope up to heterogeneous platform computation in a controlled manner.

CHAPTER 4:

Lag Erasure: Zero-Lag System Design

This chapter describes *why* the *Lag Erasure* architecture and algorithms work, covering the pragmatic detailed low-level engineering design and systemic technique selections and tradeoffs used in constructing the *Lag Erasure* system. *Lag Erasure* uses the speculative evaluation technique to compute next-events and allows the necessary time for network traffic documenting this event to reach the other geographically distributed hosts. This is done with the goal of providing a just-in-time delivery for this incoming information to be incorporated into the receiving hosts computations, and thus avoiding delays creating lag.

External inputs at some host such as a pilot manipulating a simulator flight control can not be delivered just-in-time, these late arriving to the remote host event transmissions are processed and reworked to resolve any causality-violations within a tight space-time bounds to heal the event progression back to causality correct while maintaining a real-time simulation progression rate. Maintaining causal correctness at real-time progression rates provides participants a perceptually lag-free simulation.

4.1 Design Space and Goals

The goal of this dissertation is to define a distributed simulation system which provides a novel property: elimination of the effects of lag across the entire geographically distributed set of participants. **The design scope** is evaluating simulation models and events at human scales while supporting real-time human or equipment-in-the-loop participants.

4.1.1 Equipment-in-the-Loop

Equipment-in-the-loop participants exhibit large variability in the potential agency of the equipment to control the simulation. A low-agency example is an underwater listening sensor that is programmed to transmit simple binary codes over a tether upon sensing certain programmed or hardwired inputs. This device may not have much direct influence on the simulation's progression, but bench-testing the physical device with a connected simulation allows verification of its designed utility towards integration into larger system operations. Extending the concept is useful for tank or bench testing an AUV used for research, survey

or surveillance, exercising safety-of-navigation related and presented sensor inputs that should trigger the required operating modes. This level of systems integration naturally encompasses the integration of digital twins, which are covered as a potential use case for the ***Lag Erasure*** architecture in Section 5.2.

Further extension into testing AI-imbued autonomous vehicles for deployment in the real world is a natural evolution and where the scope-of-regard and responsibility scales to the level where simple local cluster simulation may no longer be viable. It should be apparent that further increasing the scope-of-regard and responsibility for larger scale AI training and/or human interactions in complex operations further stresses the capabilities of delivering a useful simulation and eventually requires additional distributed sites to collaborate in generating the necessary scenario breadth and fidelity to accomplish the training task.

This large-scale type of training evolution is already a routine occurrence, but it is executed without the ability to remove the negative effects of lag (see Section 2.1.4) from the execution. Human common sense and ability to mentally accept or dismiss lag effects on-the-fly and then debrief-away potential negative training effects are the current state-of-the-art in these large scale training exercises. If this is the most demanding simulation use, and common sense debrief is always successful, then ***Lag Erasure*** techniques would not be necessary. But unresolvable disbelief is a regular occurrence, and there are additional simulation uses that are less accommodating of this level of imprecision and divergence.

4.1.1.1 Artificial Intelligence-in-the-Loop

AI is being integrated into new devices and vehicles with ever increasing rates of adoption. The complexity of emerging situations which entail multitudes of opposing autonomous platforms and human participants is driving efforts to press AI into the decision space engaging with these scenarios. When training an AI that is not already human equivalent in common sense, the standard of “human common sense and ability to ignore lag effects on-the-fly, and then debrief away any potential negative training effects” is not viable. The negative training effects of lag-driven artifacts in the presented sensor data also corrupt any autonomy-related or AI training processes with non-physical and unrealistic events. Debriefing those away is not an option, if lag-driven artifacts create negative training it may be excruciatingly difficult to rectify. Taking control of the flow of training data presented

and ensuring proper causality to the maximum extent possible is necessary and the *Lag Erasure* simulation techniques provide this capability.

4.1.2 Model-View-Controller-Participant (MVC-P) Top-Level Design

The simulation design space *Lag Erasure* is intended to serve is expansive, including applications within: C2 systems, the LVC paradigm in military training, multi-platform weapons systems data sharing and displays, digital twins, multi-user virtual worlds, multi-user AR, the multiplayer online video gaming community, and future development of shared massively multi-user AR and VR applications that are evolving into the metaverse and beyond to the ubiquitous everyday vir-space⁶ that Brin [8] predicts surpassing the metaverse. Despite the breadth of this application-space, they all are participant-centric in a manner that is profoundly-enveloping the participant as opposed to a participant simply filling a generic user-role outside of the applications design space. The classic software engineering design pattern for the wholly external user is the classic MVC pattern illustrated in Figure 4.1. The MVC design pattern solely describes a modular manner to organize functional modules for development, and has been an excellent design pattern for 38 years.

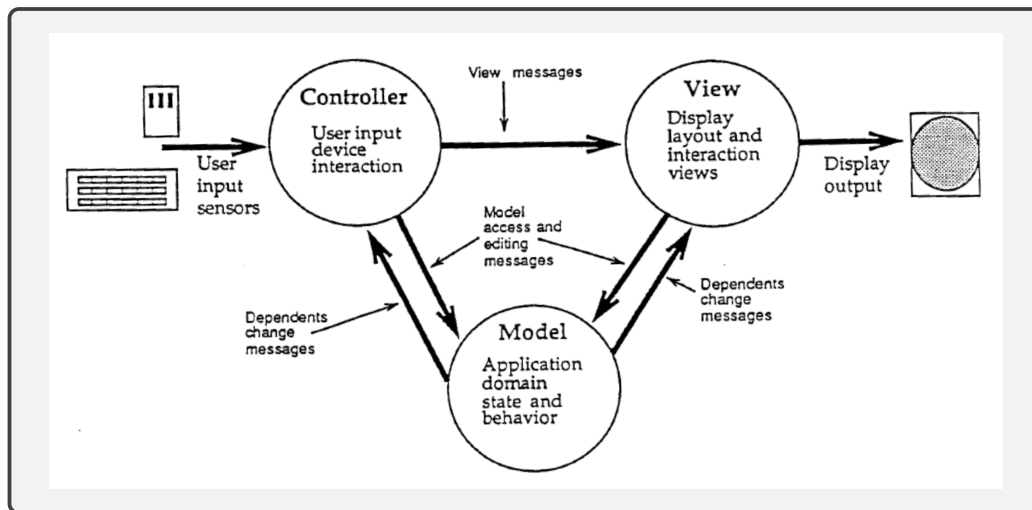


Figure 4.1. The Model-View-Controller design paradigm. Source: [74].

⁶virtual space, a near seamless blending and multi-layered overlay of virtual information into and properly geo-registered over every aspect of the real world space [8].

Unlike the external user that is omitted from the MVC design pattern description, simulations immerse the participant in a deep and rich set of display visualizations and in addition to other sensory input modalities. The necessary immersion of the simulation makes the participant an intrinsic part of the simulated system or scenario, not an external entity disassociated from concern.

This new system, depicted in Figure 4.2 positively recognizes the participant(s) potential interactions explicitly. This seemingly slight design difference matters, because without the participant as part of the design, the perceptual existence of lag is obfuscated. Lag is too easily ignored in simulations system design because the lower left quadrant in Figure 4.2 does not even exist in software patterns such as the original MVC. This leaves the consideration of lag to be outside the design purview and helps set the expectations that lag must be compensated for (covered up after-the-fact) rather than eliminated at the source.

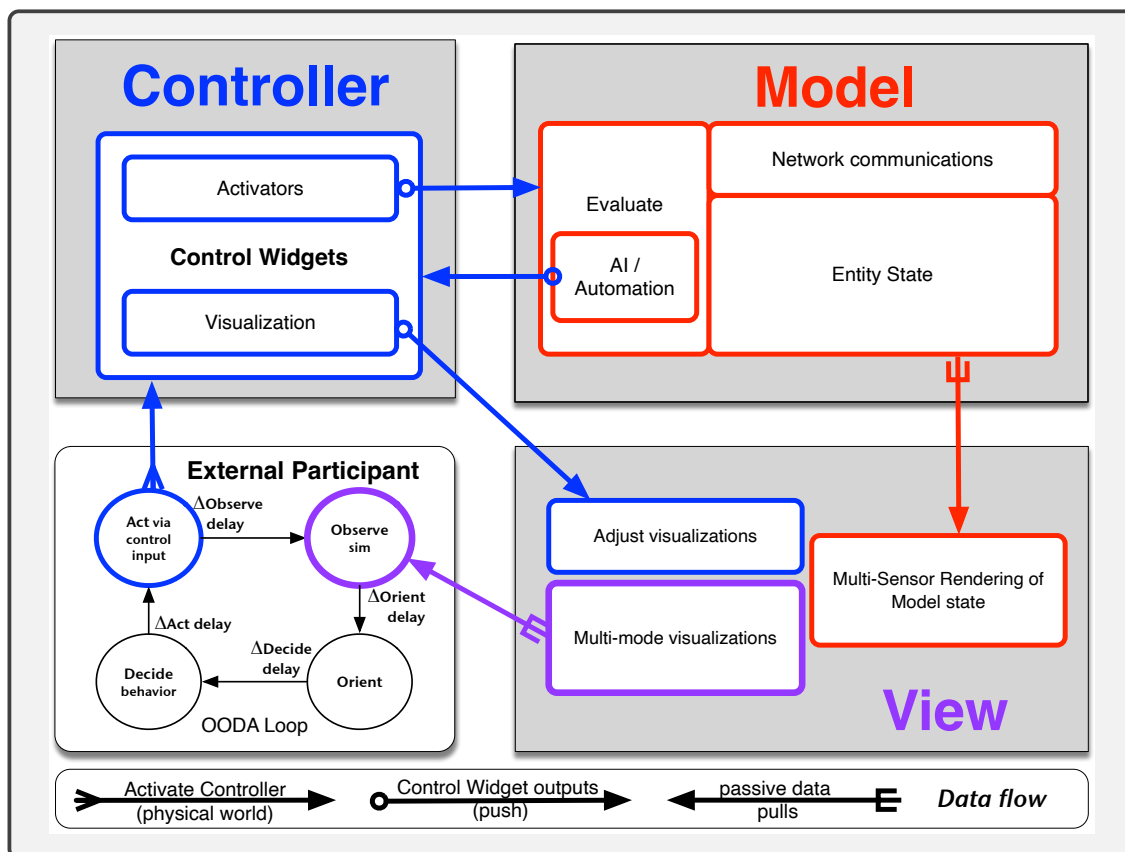


Figure 4.2. Model-View-Controller-Participant (MVC-P) system design.

The addition of the external participants to the full-system design as provided in Figure 4.2 not only recognizes the participants but also delineates the necessary data flows to maintain separation-of-concerns between modules and avoid the intermingling of **Model** and **Participant** code which can corrupt the integration of hardware-based participants by exposing or including internal **Model** concerns. The simulation **Model** software *must* explicitly remain separate from the **Participant**, since obviously humans are not inside the simulation **Model** software, but that same obviousness must also apply to equipment-in-the-loop so that the appropriate sensory channels are indicated to be provided in the **View** (display) component as necessary. Incorporating the **Participant** code inside the **Model**, or vice versa, subverts the ability to properly do verification, validation and accreditation (VVA) related to participant software or hardware-in-the-loop as the participant will not include the simulation **Model** when it is operating in the real world, nor will the real world provide any external computation for internal states of the **Participant**.

Simulation designers have a tough task deciding whether some aspects of autonomy are external participants, or if they are inherent parts of the simulated entity's model, e.g., autonomous flying software directing control surfaces based on access to an internal data bus connected to both the sensors and storage of assigned navigational requirements. Making choices on whether the internal bus connection indicates the autonomy is part of the **Model**, or if that data-jack represents just another display of the **View** has a measurable impact on the ability to effect a successful VVA process for the overall set of systems and situations the designers are attempting to gain more insight into.

4.1.2.1 Trust Your Artificial Intelligence? It All Starts With Verifiability

Section 4.1.1.1 introduced the expectation of AI entities as participating users of large scale distributed simulations. The capability for directly querying why an AI makes a decision or takes one action over another is still into the far-enough-future that separate means of verification are necessary. This requires an unambiguous understanding of what is presented to the AI during training before exposing it to unconstrained operations.

The intentional and disciplined *capability* to apply separation-of-concerns as described above in Section 4.1.2 is critical to large-scale AI projects such as tactical assistants which need to recursively manage an internal representation of the world. If the AI's internal

representation and the simulation representation are allowed to directly commingle it may lead the AI to build its internal representations expressly using parts of the simulation, rather than exclusively “sensing” the simulated world and building the internal representation from this data. Commingling simulation and internal representations may make it impossible to even understand if the AI tactical assistant is capable of generating a correct and useful internal representation, or worse, the simulation-trained AI goes rampant and displays catastrophically illogical behavior when disassociated from the simulation. **“Do you want [Skynet]? Because *that’s* how you get [Skynet].”** (adapted, Malory Archer S1E1 [75].)

4.1.2.2 Real-Time Interfaces Require Uncompromising Constraints

The Controller, View and Participant components of the MVC-P design presented in Figure 4.2 operate with reference to progression of the physical wall clock in the real world. The Controller receives command inputs directly from the Participant on-the-fly in real time; the View is sensed by the Participant in real time. The only component of the system operating with exclusive reference to the virtual time of the simulation is the Model.

This requires the relative passage of simulation time to successfully overlay wall clock time passage in the Model’s data that is sampled by the Display. This means the inherent time expectations mismatches caused by distributing the simulation and participants sharing a scene must be resolved within the Model, and because resolving the mismatches require network communications, the network communications capability is defined to be a sub-component of the Model. This avoids the artificial timing challenges in resolution of multiple component contributions before the Model receives a “final answer.” This indicates the most important part of the ***Lag Erasure*** implementation is realized within the Model component.

Because the Model is the focus for achieving the zero lag goal within the ***Lag Erasure*** architecture does not mean the other components are unimportant, they remain central to the creation of any successful simulation but their implementations are not direct contributors to creating the successful erasure of lag from the simulation. The Display component is more rigorously isolated from the Model’s internals than in other simulation engine implementations and this drives a difference in implementation for data flow in generating output display frames, however the data flow difference is an interface-level issue, not an internal operational issue responsible for providing the capability to erase lag.

4.2 Latency and Lag

The traditional definition of lag, presented in Section 2.1.1 is: lag is considered to be the perceived time delay between an action and displaying the result of that action back to the entity who input the action. Section 2.1.1.1 established the basis for not accepting “perception” as a lag measurement metric, Section 2.1.3 removes the troubling inclusion of perception by establishing that the actual latency delay is the time-interval being perceived and provides a metric that is precisely measurable within the simulation’s operation itself. No dependencies on external perceptions. This works because if there are no latency delays the views presented to the user can not be perceived as lagged, and the longer the latency delays, the worse the perceived lag is. The latency measure itself is the in-simulation proxy for expressing lag.

4.2.1 The Mathematics of Latency and Perceived Lag

With latency measure providing a measurable in-simulation proxy for lag, lag can be broken into two primary contributing bins, resulting in Equation 1.1, reproduced here for convenience as:

$$t_{lag} = t_{latency} + t_{algorithmic}, \quad (4.1)$$

where the timing delay that results in the perception of lag is quantified as t_{lag} . The delay’s components are any network transmission latencies ($t_{latency}$) and any delays generated by simulation software algorithm implementation choices ($t_{algorithmic}$). Solving for the goal where the measured latency delay is equal to zero yields:

$$0 = t_{latency} + t_{algorithmic}, \quad (4.2)$$

and further simplifying to

$$-t_{algorithmic} = t_{latency}, \quad (4.3)$$

indicates that when $t_{algorithmic}$ takes on a negative value with an absolute magnitude equal to that of the combined latency delays represented by $t_{latency}$ overall latency is offset.

This result is seemingly contradicted by the plain fact that running code requires time on a wall clock to complete, naïvely the value for $t_{algorithmic}$ must always be positive! But this (always-positive) interpretation is both incomplete and includes an artificial assumption about when the clock may be started. Figure 4.3 visually represents the always-positive outcome on the number-line. When an event happens at time t_0 all the times to the right are later and increasing, which is also true for the magnitudes of those time's relative offsets from the t_0 zero baseline.

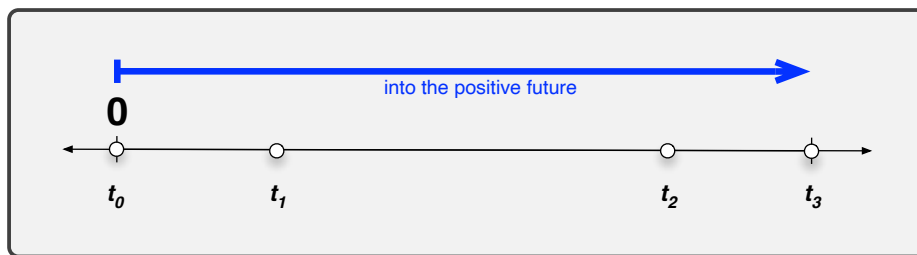


Figure 4.3. The always positive measurements on the number-line.

The artificial assumption responsible for limiting the value of $t_{algorithmic}$ to always be positive is eliminated by realizing that the measurement may be made with respect to the desired outcome as the zero-point rather than using the time of calculation as the zero point. This re-baselining is accomplished with a simple translation of subtracting the magnitude of t_3 from each point relocating zero on the number-line (the points may be re-labeled without loss of generality.) The same relative timing relationships in the same positions are depicted in Figure 4.4.

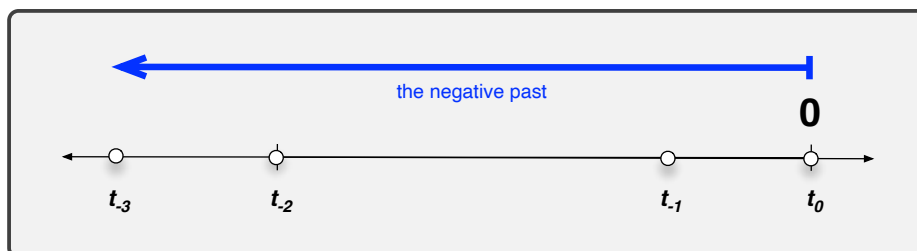


Figure 4.4. Relabeling the number-line, same absolute locations as Figure 4.3.

This makes obvious that while the magnitude of the offset between t_{-3} and t_0 is the same as the distance between Figure 4.3's t_0 and t_3 , the signed value of $t_{algorithmic}$ must be negative because it is to the left of the Figure 4.4 zero point. This relationship verifies that a timing-zero-point located at the simulation time an event is committed to the overall simulation state implies that to accomplish a just-in-time delivery, the computation must be algorithmically completed at least $t_{latency}$ before the necessary simulation time for the event.

4.2.2 The Consistency-Throughput Tradeoff [1]

It is impossible to allow dynamic shared state to change frequently and guarantee that all hosts simultaneously access identical versions of that state. [1]

This statement is labeled the *Consistency-Throughput Tradeoff* by Singhal and Zyda [1]. It is supported by the following reasoning:

To guarantee absolute consistency between hosts, the data source must wait until everybody has received the information before it may proceed with the computations (or, in particular, update the shared state again). [1]

and

Conversely, when a state change is generated at a particular host, there is a necessary delay before the other hosts learn of the change; this delay consists of the network latency for transmitting the update over the network. Consequently, if the shared state is updated often enough then the shared state might be updated while the previous update messages are still in transit to remote hosts. As a result, some hosts may see a new state value while others still see older values being disseminated by the state consistency protocol. [1]

There are assumptions hiding in this logic that are implicitly believed without statement by the VE simulation community. The first assumption is deep within the need to “wait.” The

cited waiting is absolutely necessary, it represents the $t_{latency}$ term of Equation 4.1, the covert assumption is that the waiting can only be started after, that it may not be accomplished before within a just-in-time delivery scheme as introduced in Section 4.2.1. In the time that Singhal and Zyda wrote their text *Networked Virtual Environments* [1] this is a natural result because VE simulations are and have always been, evaluated using synchronous time step advances and the time step that an event is computed is by default the zero-point reflected in Figure 4.3. In that historical timeframe there is no alternative option to after-waiting on transmission and acknowledgement.

The second covert assumption is that “waiting” always breaks real-time simulation progression rates because “if the shared state is updated often enough then the shared state might be updated while the previous update messages are still in transit to remote hosts.”[1]. This interpretation of the “waiting” covert assumption requires the waiting periods to be longer than the time step evaluation intervals, and further that “not waiting” before receiving an update message and locally updating shared state through one or more time step evaluations is a situation that is not repairable within the required dynamic timeline of the simulation. This viewpoint is easily understandable when considering a simulation which uses the near-entirety of the CPU capacity to maintain the simulation’s forward time progression; stopping, discarding one or more time steps, and recomputing them just does not work because there is little-to-no excess CPU capacity available for the repair.

4.2.2.1 Adding Nuance to the *Consistency-Throughput Tradeoff*

Identification of the two covert assumptions within the heart of the *Consistency-Throughput Tradeoff* enables the questions of *what if these assumptions are not absolute?* and *What does that look like?* Then, does use of new explicit assumptions violate the *Consistency-Throughput Tradeoff*, or just constrain it?

4.2.2.1.1 Eliminating the Short Synchronous Time Step Assumption The short answer is that removing short synchronous time steps addresses the waiting and repairability aspects of the *Consistency-Throughput Tradeoff*. Asynchronous time advances that are long compared to traditional step lengths do not invalidate the *Consistency-Throughput Tradeoff*, they simply extend the length of the window where waiting does not create a conflict with the dynamic shared state. Long time advances explicitly mean fewer total advancement

computations, creating time opportunity for the repair work in those cases where repair is required due to network latency longer than the associated event time advance.

4.2.2.1.2 Resolving the When to Wait Assumption When the “waiting” must transpire, *before* or *after* the event’s simulation time, is already exemplified by analytical simulation methods and use of optimistic advancement. Under optimistic advancement, computations routinely occur early or before the simulation time, because a simulated process is allowed to compute itself without constraint from other processes. Optimistic advancement simulations compute in relative event order per simulated process, rather than relative global order, and the overall simulation timeline is managed under the hope that there will be no conflicts in causality from non-constrained progression. Repair in cases where there is a causality conflict is accomplished using the technique of rollback, and the rollback technique explicitly manages when the *before* computation was too early compared to the global simulation timeline.

4.2.2.1.3 The Result Asynchronous time advance is directly supported by numeric adaptive time step algorithms to generate the longest stable time advance at each evaluation. This provides connection to formal mathematical methods which minimize absolute and relative computational errors per advance interval. Not only does this reduce overall error in the simulation, the fewer computations executed with the asynchronous time advance provides fewer opportunities for computational error to accumulate or feed round-off error propagation.

An asynchronous time advance meshes well with event-driven analytical simulation techniques. The combination presents longer duration events (compared to synchronous time advance) as the baseline intervals for the “waits” behind *Consistency-Throughput Tradeoff*. When these longer waits are considered in relation to Equation 4.2, and the waits are accomplished before the posting simulation time, they provide a negative value for the $t_{algorithmic}$ component of Equations 4.2 and 4.3. The magnitude of $t_{algorithmic}$ then represents the maximum latency that this arrangement may hide as illustrated within Figure 4.4.

These interpretations fully support the basic limitations expressed in the *Consistency-Throughput Tradeoff* but limit the impossibility limitations on dynamic shared state maintenance into an existing corner of a now enlarged simulation operating envelope.

- In conditions where the time advances are shorter than the transmission latency, the full effects of the *Consistency-Throughput Tradeoff* remain and *it is impossible to allow dynamic shared state to change frequently and guarantee that all hosts simultaneously access identical versions of that state* [1]. These temporary periods of state divergence require rework (repair) to resolve.
- When transmission latencies are shorter than the entity-update time advances, the just-in-time arrival pattern for network transmissions may be invoked and effectively hide the associated transmission latencies. In this case *it is [possible] to allow dynamic shared state to change frequently and guarantee that all hosts simultaneously access identical versions of that state* (adapted from [1].)

The specifics of achieving the just-in-time delivery pattern are not trivial, while a just-in-time delivery simulation must behave in exactly the same manner as the does traditional synchronous fixed time step simulation, the internal implementation choices are not directly compatible with each other. Speculative evaluation techniques will be covered in Section 4.2.3.3.1, which provide the necessary implementation requirements for the necessary just-in-time delivery pattern.

4.2.3 Operationalizing the Just-in-Time Delivery Pattern

The event communications before and after timelines of Figures 4.3 and 4.4 are relative and abstract without detail on procedural specifics. The first step in removing the abstraction is fully decorating the timelines with labels and segments that represent the various segments of the event network communications process.

4.2.3.1 Before Just-in-Time Delivery

Figure 4.5 illustrates the traditional communications process where the event posting to the simulation timeline is the zero point, and the beginning of the event communication timeline. The timeline contains not only the total latency, but also the breakdowns between the simulation software's algorithmic components and the transmission latency itself, becoming a visualization of the Lag Equation (Equation 4.1.) At the level-of-detail necessary to account for transmission timeline interactions, the hardware and software levels of the Internet Protocol networking stack and operating system buffer transfer are included in the transmission latency ($t_{latency}$) in addition to this obvious flow of packets outside the

local host machine the application layer of the simulation software is responsible for the algorithmic delay ($t_{algorithmic}$) components.

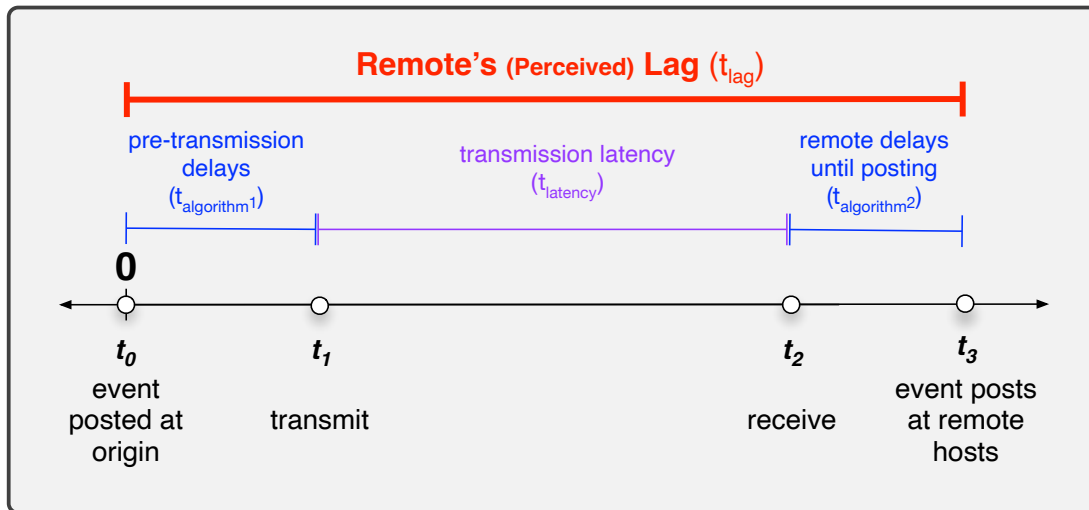


Figure 4.5. Traditional distributed simulation event communication timeline, sub-intervals not to scale.

The interval *Remote's Perceived Lag* identifies the time that contributes to the remote user's perception of lag as presented in Section 2.1.3: *A Precise Metric for Measuring Lag*. The nefarious connotation of this t_{lag} interval is that the distant distributed hosts may evaluate any number of time steps before receiving the transmission. With a visual frame rate of 60Hz, the computed steps progress at one per every 16.67ms. This seems like a minimal interval, but while it appears like only one frame if your network connection is fast, once a frame begins processing it will not be rolled back and recomputed because of incoming network transmissions. So this fast single-frame equivalent 16.67ms is actually a 33.33ms two-frame delay by the time the receiving end has displayed it. A longer transmission delay itself does not reduce the additional processing delay at the remote host, and it is unreasonable to expect communications between distributed simulation sites to push latencies down to compete with cluster transmission times of a millisecond or two.

With an expectation of two computed simulation steps for the 33.33ms of simulation time before the remote host integrates the incoming event transmission, the level of difficulty

increases dramatically to address the challenges of multiple moving objects, associated collisions and responses, and timelines for information contributing to user decisions. An initial reaction is to bump up the frame rate, 120Hz is not uncommon, but this does nothing to solve the latency problem. While the time penalty for the additional frame on the receiving end may be cut in half to 8.33ms, the doubled frame rate doubles the number of frames computed during the transmission delay. This translates to a best case example of three frames late over 25ms compared to the original 2 frames late over 33.33ms. While these relative timings are oversimplified, the inescapable result is that frame computation rates are not profitable latency reduction methods.

Whenever additional time advancement steps are taken during the transmission and processing latency window the risk exists that an incoming event will need to affect some entity or item that also took some action during that t_{lag} window and if that action should have changed due to the incoming event, a causality failure occurs. Causality failures are frequent enough that they must be managed in some fashion, they cannot be ignored. Recomputation is too expensive, this derives from the *Consistency-Throughput Tradeoff* [1]. To sidestep the problem the standard VE simulation community response is to visually hide the problem using a lag compensation technique (per Section 2.1.7.) This does not repair the causality error, it just makes the errant results less visually offensive, and hopefully more believable to the participants. The ADES community does directly fix the causality issue with the technique of rollback, but rollback is expensive to execute from a simulation progression metric and cannot guarantee the necessary real-time progression rates for engaging with participants.

4.2.3.1.1 Rollback and Lag Compensation Rollback requires freezing simulation progression, communicating as required to alert other hosts that received events which may now be superseded due to the rollback, and then finally rolling back the clock and recomputing the events that transpired in the rolled-back window. Rollback suffers inefficiencies from both reevaluating events and also requiring additional compute time execute the rollback algorithm, a delay that a simulation user will perceive as additional lag. Potentially worsening the effects of rollback are that the extra required recompute time may also create additional artificial delays within a host that then cause follow-on event transmissions to go out to other hosts even later, additively worsening the effect of the systemic lag. Rollback is incompatible with real-time simulation rates and must be avoided.

Rather than roll back to fix this incoherent and inconsistent situation, the VE simulation community developed the lag compensation techniques to avoid the need to recompute, visually manipulating the scene to mitigate effects of the lagged mess in the perception of the user. Section 2.1.7 established that lag compensation does not fix the problem, it papers over it with an inaccurate but visually-acceptable-for-the-designed-use presentation.

In either paradigm, ADES or VE, there is a risk that the practicalities of causality errors triggering the need to recompute already committed results may create a recomputation cascade until the situation becomes intractable. ADES as typically executed carries a lower risk of intractability as rollback may make a non-real-time simulation take longer but not break the ability to eventually generate a correct answer. The time-stepped VE simulation paradigm typically interacts with human or equipment participants and therefore requires the simulation to progress faithfully in real-time with respect to the wall clock to meet its mission objectives, and recomputation delays can break that requirement if their collective effect magnitude is more than one or two tenths-of-a-second.

4.2.3.2 With Just-in-Time Delivery

With the timeline zero-point defined as *the time an event needs to post at every host* to properly maintain causality, associated relabeling of the same absolute timeline from Figure 4.5 results in the depiction of Figure 4.6.

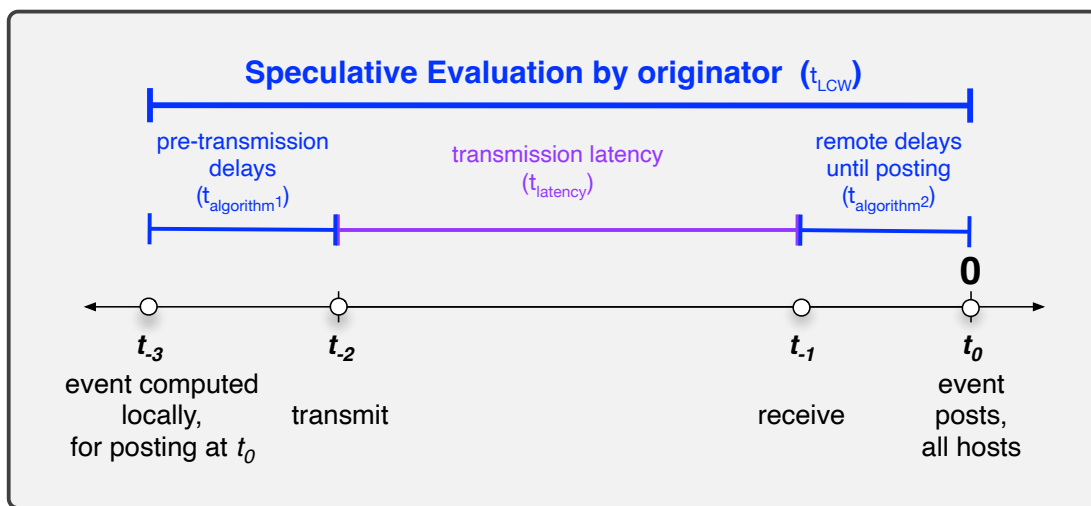


Figure 4.6. Just-in-time communication timeline, sub-intervals not to scale.

When the computation occurs as depicted at t_{-3} there is enough time for the information to arrive and satisfy a just-in-time delivery requirement. While there is no penalty if the information packet arrives early, there is the potential harsh causality violation penalty discussed in Section 4.2.3.1 if it arrives late. This just-in-time delivery modality and its early accomplishment of computational work is a form of speculative evaluation which is a well-known concept in the field of computer science.

4.2.3.3 Speculative Evaluation and Latency Hiding

Speculative evaluation accomplishes necessary work and pays unavoidable latency costs before the results are required, rather than after. When speculative evaluation is used specifically to deal with large latency penalties it is also commonly called *latency hiding*. When the code is executed to evaluate an event is an arbitrary algorithmic implementation choice, the time that the event is commit to the simulation timeline is the requirement. A combined comparison of Section 4.2.3.1's and Section 4.2.3.2's event communication timelines is presented as Figure 4.7.

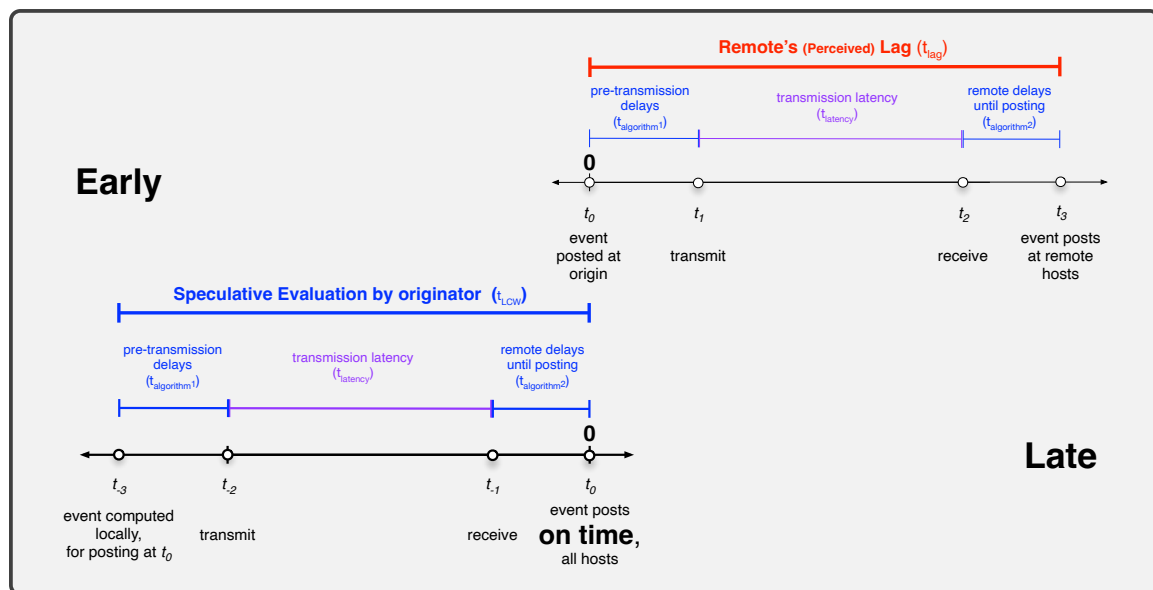


Figure 4.7. Visualizing the relationships between event transmission timelines.

The early computation choice represented in Figures 4.6 and 4.7 is causally consistent because nearly all physical causality at the human scale derives from Newton's Laws of Motion⁷ [76] (as paraphrased in English and hosted by NASA Glenn Research Center [77]):

1. An object at rest remains at rest, and an object in motion remains in motion at constant speed and in a straight line unless acted on by an unbalanced force.
2. The acceleration of an object depends on the mass of the object and the amount of force applied.
3. Whenever one object exerts a force on another object, the second object exerts an equal and opposite on the first. [77]

Newton's Laws *explicitly* do not state **when** the math must be evaluated within a simulation; they do in combination imply requirements for ordering causality of motion and interaction. Thus, as long as code is otherwise implemented to maintain correct causality sequences, speculatively evaluating events is consistent with classical Newtonian physics.

In this architecture the specific goal is to set zero no later than (NLT) the required t_0 arrival time at the remote distributed hosts which describes the just-in-time delivery requirement. The new time duration label t_{LCW} represents the estimated interval required to hide the required transmission latency plus jitter and any other additional delays imposed by algorithmic components, this is termed the **latency compensation window (LCW)**. The LCW may be updated at runtime based on network monitoring, or an expected worst-case fixed value may be designated. In development testing code, a fixed value of 100ms is implemented to verify the ability to cover transmission latencies ($t_{latency}$) on a continent-wide scale (averaging 65ms New York to Los Angeles, per Figure 4.8.)

4.2.3.3.1 Design Requirements for Implementing Speculative Evaluation To speculatively evaluate events ahead-of-time by the LCW interval, including physics-controlled events, there are two interlocking design issues that must be addressed: (1) The tyranny of the fixed short time step and (2) Unconstrained LP progression.

⁷Other mechanisms of causality are described by thermodynamics and electromagnetism, use the appropriate physics where needed. For the simulation approximations within the scope of this dissertation, Classical Newtonian motion suffices.

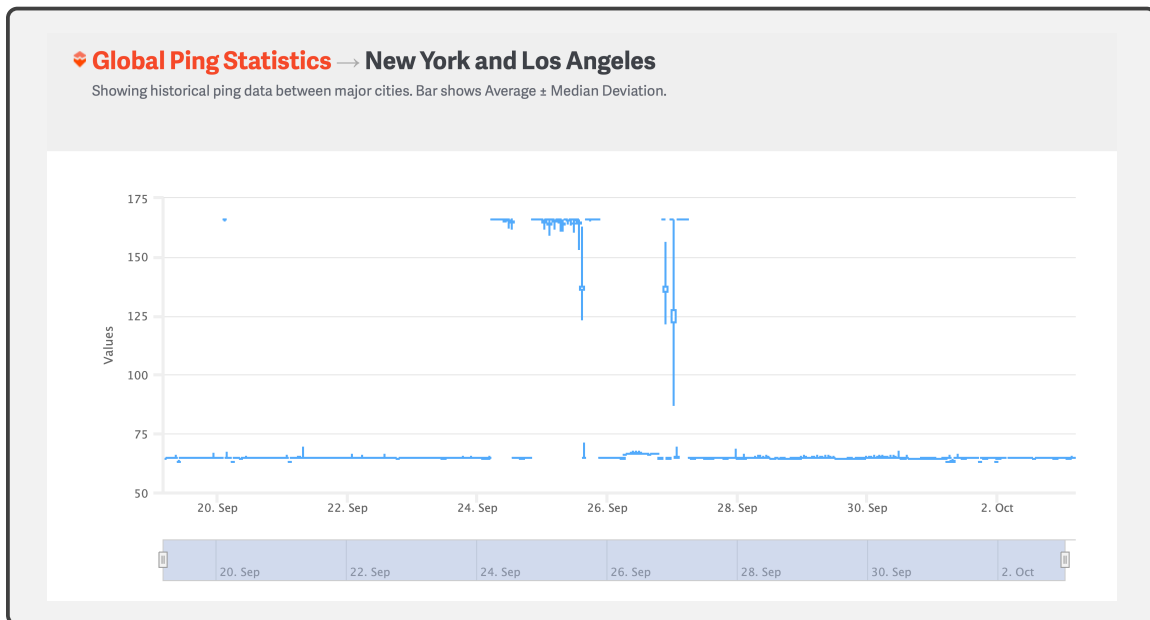


Figure 4.8. Ping time (ms) between New York and Los Angeles. Source: [78].

1. A synchronous physics time step at frame rate or some multiple of frame rate frequency (e.g., 16.67ms at 60Hz, 8.3ms at 120Hz) does not provide enough time between steps to use an LCW that can hide latencies on the order of the continent-wide averages noted in Figure 4.8.

To resolve this, the length between evaluations must be severed from the concept of fixed display rate time step, while not sacrificing required fidelity and accuracy in computations. The simulation needs to progress at a mathematically derived optimal rate and the display computations should sample the simulation state as necessary. Not only will this be in keeping with the MVC-P design goal (see Figure 4.2) it also simplifies implementation of the simulation model as there are no View dependencies forced into it via frame-rate matching requirements. The frame rate coupling is a weakness noticed in every open-source simulation physic engine investigated and further confirmed by cutting-edge industry self-reporting in [14].

Adaptive numeric algorithms provide the capability to generate mathematically sound maximum-length time advance intervals while preserving fidelity and reducing sources of cumulative error in the system. Using the adaptively generated time advance intervals as event durations meshes well with the ADES paradigm, and since conservative methods will prevent meeting real-time integration with humans and equipment-in-

the-loop, optimistic ADES simulation is required. Subsection 4.4 covers the particulars of the ADES design choice.

2. Existing optimistic ADES techniques do not include progress constraints that meet the engineering needs for speculative evaluation use, as presented in Section 2.4.2. To resolve this the community-standard interpretation of the optimistic simulation LP concept must be modified to include progression constraints. Section 4.4.4 covers the specifics of this requirement.

4.3 Causality

Lamport's 1978 paper *Time, Clocks, and the Ordering of Events in a Distributed System* [32] conveys the basic requirements for causality in a distributed simulation and is fully compatible with motion adhering to classical Newtonian Physics. Lamport's base Clock Condition is: "For any events a, b : if $a \rightarrow b$ then $C\langle a \rangle < C\langle b \rangle$," where $C\langle a \rangle$ indicates the time on the clock of event a [32]. Maintaining this strict distributed event relationship is mandatory to any simulation that maintains causality of one event causing another.

Lamport also defined the relationship between events that happen on the timeline, but do not have an interaction, he termed these events as concurrent. Concurrent events maintain the relationship $a \nrightarrow b$ and $b \nrightarrow a$ [32] as events that fall onto the virtual timeline but are not resultant of a causality chain with respect to each other. This definition of concurrent events does not imply that there is no ordering of the events with respect to the virtual timeline, only that the concurrent events are not causal to each other.

Lamport [32] further delineated two sub-conditions, C1 and C2 defining clock ticks between processes.

C1. If a and b are events in process P_i , and a comes before b , then $C_i\langle a \rangle < C_i\langle b \rangle$.

C2. If a is the sending of a message by process P_i and b is the receipt of that message by process P_j , then $C_i\langle a \rangle < C_j\langle b \rangle$. [32]

To ensure the clock conditions Lamport [32] provides simple implementation rules:

- IR1. Each process P_i increments C_i between any two successive events.
- IR2.
 - (a) If event a is the sending of a message m by process P_i , then the message m contains a timestamp $T_m = C_i\langle a \rangle$.
 - (b) Upon receiving a message m , process P_j sets C_j greater than or equal to its present value and greater than T_m . [32]

Lamport's conditions and implementation rules are defined with respect to a real-world operating distributed system without invoking use of any physical clocks, but maintaining the concept of time advancement indicated via the before-after relationships of events. Jefferson [31] defines event before-after causality in view of multiple clock variables that must eventually be synchronized to provide the equivalent of clock conditions in his *Time Warp* algorithm, for additional detail see Section 2.3.1.

This architecture simulates a model over time rather than provide real-world distributed services in real time, this enables the further simplification of the virtual time concept. The use of a single virtual clock and event list driven scheduling serves the same purpose as Lamport's before-after replacement for a physical clock, with scheduled events across the distributed network maintaining Lamport's Clock Conditions with respect to the single virtual timeline.

4.3.1 Virtual Time and Clock Conditions

Because events are generated following the Event-Graph methodology (see Section 2.3.4.3) every event- a comes before event- b because a schedules b . This relationship holds true even when simulation events a and b have the same virtual timestamp due to the construction of a particular Event-Graph model, because it is that evaluation of a that schedules b (following monotonicity of events on the virtual timeline). Thus, even when the virtual timeline delay between event- a and event- b are zero seconds, such as the evaluation of a `NarrowPhase` event scheduling an immediate `EvaluateContact` event, the ordering of events as removed and evaluated from the `EventList` provides the valid partial ordering even when the virtual timestamps are the same.

The use of a single virtual clock simplifies away the concept of each host i and j separately measuring times C_i and C_j . This redundancy arises as all host's times C_i and C_j are actually expressed on the single virtual timeline as a time C_v . All event description and sorting are accomplished on this one timeline so the C2 concept of $C_i\langle a \rangle < C_j\langle b \rangle$ [32] simplifies to a slight adjustment of C1: $C_v\langle a \rangle < C_v\langle b \rangle$. Even when individual distributed hosts introduce real-time I/O events into the simulation, evaluating the real-world I/O events result in generating an appropriate event for scheduling onto the singular virtual timeline with a corresponding C_v . This is also simpler than Jefferson's *Time Warp* because there is no need to consider time of send or time of arrival in the overall scheme, only the event `startTime` is germane, which is where (C_v) the event falls on the virtual timeline.

The partial ordering of the `EventList`, even for events that have the same execution time, means IR1 is always satisfied. And like C2, IR2 is redundant as all events are already following the full ordering of IR1 on the virtual timeline. This makes distributed event ordering trivially simple right up until one of the real-time I/O events attempts to violate the monotonicity of the virtual timeline and puts the existing status of causality into doubt. In Jefferson's *Time Warp* this generates criteria for a rollback (see Section 2.4.2), in the ***Lag Erasure*** architecture there are constraints not available to *Time Warp* and these constraints allow identification of the minimum required subset of events that must be reworked to restore the virtual timeline to its necessary monotonicity and compliance with Lamport's C1. The specifics of rework are covered separately in Section 3.1.5.3.

4.3.2 The Jugglers, an Example

A practical example illustrating Lamport's clock conditions and relationships to a virtual timeline follows: Start with a lone juggler, represented as in Panel A of Figure 4.9, juggling five clubs. Each club moves through spacetime independently, no club exerts any direct influence on any other. Within the scenario these clubs are operating in a mode that exhibits Lamport's concurrency ($a \nrightarrow b$ and $b \nrightarrow a$ [32]), and the trajectories are mappable to the virtual timeline because they are following a time ordering as they flow through their trajectories, a-b-c-d-e, always in that order. Even with this regular sequence on the virtual timeline, club-a does not cause club-b to do anything.

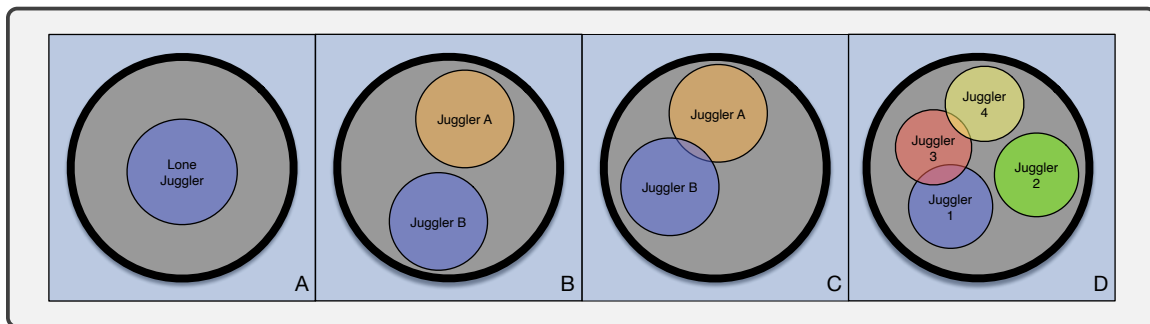


Figure 4.9. Four juggler interaction scenarios.

Looking a bit deeper, this time ordering is explicitly provided and enforced by the interaction the juggler takes with each individual club during the rhythmic cycle of the juggle, catch-throw, catch-throw, . . . The catch-throw event pairs themselves are causal, juggler catches club-a then throws club-a, this event pair causes the follow-on catch-throw event pair for club-a to be scheduled at the correct point in the future. This illustrates Lamport’s C1, and because the juggler is a lone actor it should be obvious that the time duration of the act of moving a hand from the throw to the ready position for the next catch enforces the incrementing of time that is required and described in Lamport’s IR1 [32].

As far as this Panel A is developed, everything is easily simulated in a single host machine interacting with the virtual timeline, therefore the message related conditions of C2 and IR2 are moot. Next we extend the scenario to two jugglers each simulated on a separate simulation host.

Both jugglers are independently juggling while facing each other, represented as Panel B of Figure 4.9, and therefore each maintaining local C1 and IR1 causality compliance, but global concurrence. As rehearsed, both jugglers adjust a catch-throw to send a club to the partner instead of themselves, their practiced skill sends it so that a thrown club replaces the partners thrown club in the timing sequence. The pair of jugglers are now represented by Panel C Figure 4.9. This illustrates that both jugglers interacted with respect to a shared global timeline, allowing them to transform a previously independent concurrent event sequence into a causal event entering their partners cycle. This “add” event required precisely delivering the club in both time and space so that their partners routine cycle remains valid, we acknowledge the positional delivery as a result jugglers skill and shift focus onto the still skillful timeline interaction.

The throwing juggler necessarily determined the shared timeline catching-moment that the catching juggler requires to maintain the rhythmic juggle, not with reference to some absolute external clock, but rather with reference to the shared relative timeline of interaction. This shared timeline interaction is where Lamport's C2 and IR2 are satisfied—one jugglers throw (message) is before the other jugglers catch with respect to the shared timeline. Advantageously each jugglers internal sense of the timeline and the sensed relative timing between the jugglers is directly mappable onto the globally shared timeline that is the wall clock. This overall situation illustrates the obvious idea that both jugglers actions can be mapped to the global timeline, even when they are operating independently, and that interactions can be interpreted both globally and locally, using the mapping of events to memorialize both the absolute timeline global ordering, while simultaneously maintaining the local partial orderings within each host.

4.3.3 The Singular Virtual Timeline

Stated simply and plainly, all events are order-able and map-able against one global timeline, and in this architecture it is the simulation's singular virtual timeline. The relative time mapping of a local event that is shared between two or more simulation hosts necessarily "synchronizes" the sequence of events globally into the correct causal position in the mapping of the union of all local events, including the shared events to the singular virtual timeline. Chains of directly interacting simulation hosts extend the scope of explicitly synchronized event orderings, which is represented by Panel D Figure 4.9. Even when simulation hosts are not directly interacting, the mapping of events onto the singular virtual timeline will maintain the necessary synchronization, ensuring that even Juggler 2 in Panel D is properly evaluating events both locally and with respect to the other participants in the simulation.

In this manner, multiple clocks are not needed nor desired. Times within each simulation host are the times on the global timeline, and any host-specific variables holding times are the equivalent of labels on the singular virtual timeline, not separate clocks as is the previous simulation practice.

4.4 Asynchronous Discrete Event Simulation (ADES)

Not all simulation communities use similar terminology in the same detailed manner, so the terminology playing field across the simulation communities must be leveled first. Traditional VE simulation is time-stepped thus the community has a different concept of *event* terminology than the analytical community who should be familiar with the terminology used below, but not specifically how it will be applied in the hybridization between paradigms.

In a useful oversimplification, the traditional fixed time-step simulation advance is the degenerate case of ADES simulation time advance, where every entity state change is called an event and evaluated with a time advance of identical length and all events in a fixed-time-step-window are scheduled to be evaluated at the same time. With both simulation paradigms having that strong overlap it is a far smaller mental leap to move from fixed time step thinking to working with adaptive time advances and asynchronous scheduling, in both paradigms the same concerns must be addressed.

The term *event* is more tightly defined in analytical simulation than it is in VE simulation, and this architecture uses the analytical definition: **an event occurs at the times an entity state must change, and the event is considered to persist until the next event occurs** [36]. This ensures the combined trajectory of an entity's events are continuous in time as depicted in Figure 4.10. Of importance is that the entity state trajectory *is not* the physical trajectory of an entity, specifics about entity state are discussed below in Section 4.4.2.

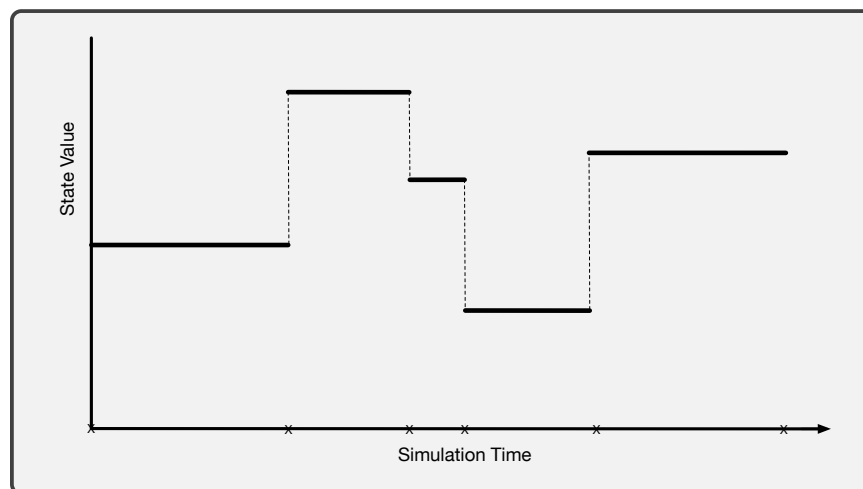


Figure 4.10. A Typical DES state sequence. Source: [37].

4.4.1 Event Evaluation Loops

The generation of the state sequence is managed via a loop structure in both ADES and VE implementations. This structural design similarity is identified and illustrated in Figure 4.11 showing the ADES next-event loop and Figure 4.12 presenting the synchronous step loop favored by the VE community.

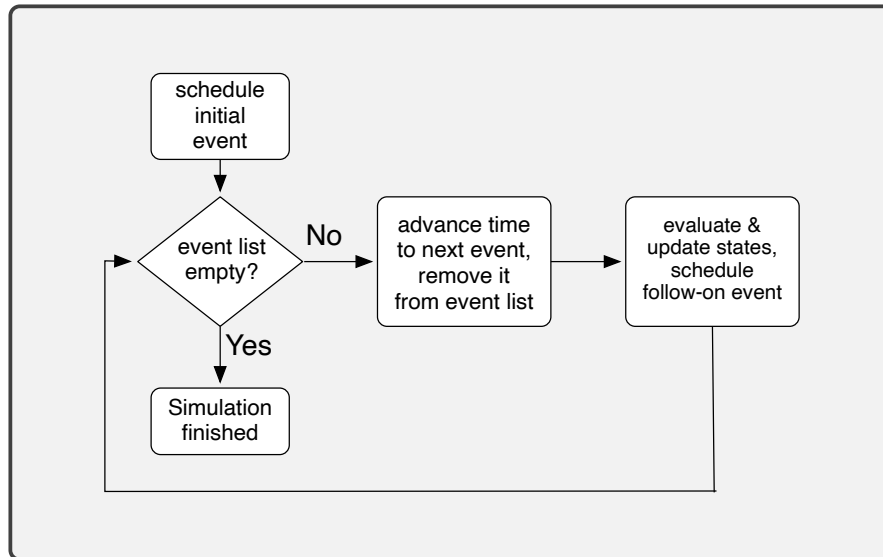


Figure 4.11. The ADES next-event algorithm. Adapted from: [37].

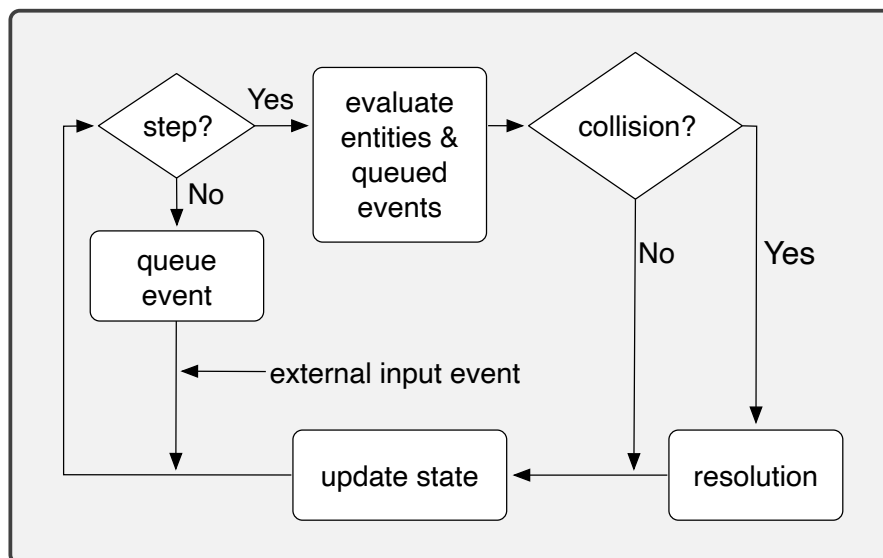


Figure 4.12. Simplified synchronous time step simulation loop.

The similarities are leveraged in creating the hybridized simulation loop shown in Figure 4.13, harmonizing the simulation paradigms.

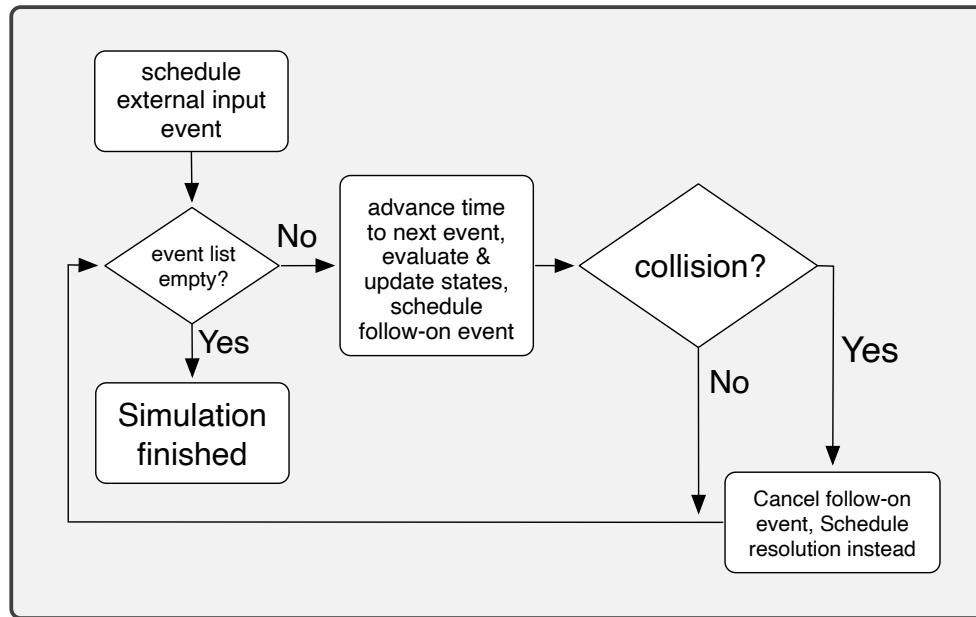


Figure 4.13. The ADES-VE hybridized simulation loop.

The basic ADES next-event algorithm in Figure 4.11 is typically considered as a single-thread implementations on a single machine thus avoiding multi-thread synchronization concerns in the simulation. Extending a single-threaded Figure 4.11 design which follows the current analytical community practice for LP abstraction, into a multi-threaded design based on Figure 4.13 requires each design level LP to be implemented as an independent hybridized simulation loop in its own dedicated thread, with each independent loop managing its own progression with respect to the clock. This strict independence in implementation is why optimistic ADES threads can progress into the future without reference to other LP, which can result in causality failures that require computationally expensive rollback techniques to rectify.

Of note in the classic simplified synchronous VE time step simulation loop depicted in Figure 4.12 is that the methodology not only explicitly includes events as it must, it queues events! The queue time-depth is restricted so that all events in the queue are evaluated at the next time advance. This stealthy implementation detail increases the complexity and overhead

in managing both the time steps, and the event queue dependencies within a particular time step to ensure overall correct causality. Other than the addition of collision handling in the VE synchronous simulation loop, the two paradigms loop design are similar, as they must be. The dichotomy covered in Section 2.5 is therefore in the implementation details and optimizations rather than different problems requiring different solutions. This supports the intuition that hybridizing the two simulation implementation patterns is possible rather than forever separate concerns.

The harmonization of ADES and traditional VE simulation loops presented in Figure 4.13 shows the ease of grafting inputs and collision detection, which results in the hybridized ADES loop. Incorporating the traditional VE simulation loop intra-step event queuing into the main `EventList` adds less overhead than imagined because the intra-step queue was already required to be time-sorted. This hybrid design is specified with one singular timeline and clock; multi-processing, multi-threading, and distribution specifics are implementation level choices which support the single timeline with a single clock concept.

4.4.1.1 Hybridized Simulation Loop Design Considerations

Moving the hybridized simulation loop a step closer towards implementation requires specifying operations of the MVC-P Model component, the result is Figure 4.14.

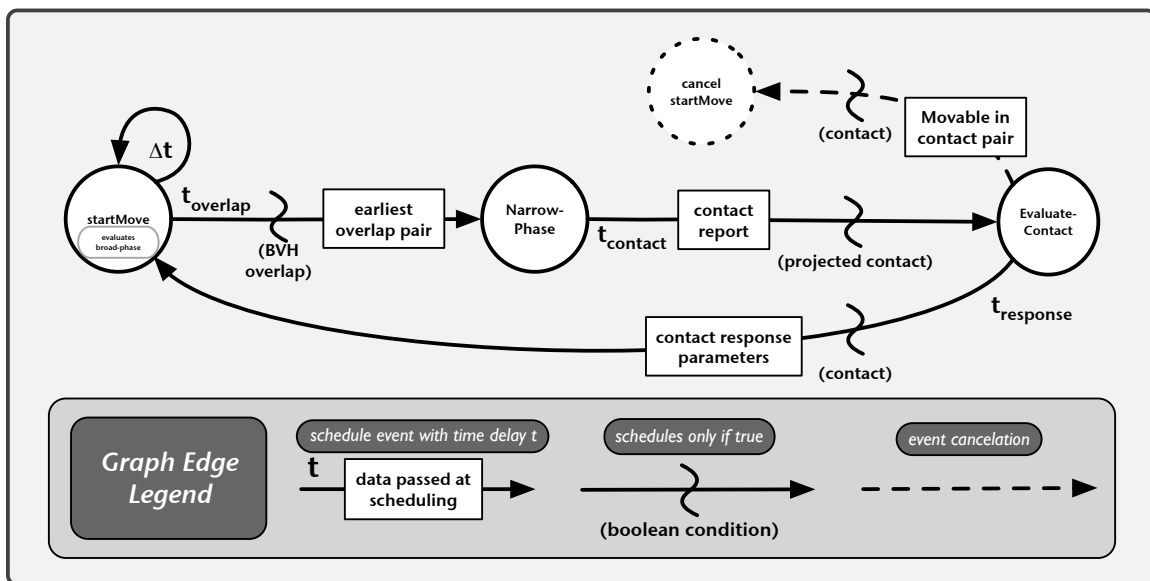


Figure 4.14. The core details of the Model simulation event loop.

Implementing this Event-Graph model is straightforward and only requires that the entity's implementation for the `startMove` event determines when the current action will next be required to change state again. The step-length determination (Δt) is accomplished for the longest event intervals allowable with acceptable error tolerances by using second-order adaptive step numerical methods.

The simulation collision detection event infrastructure is provided the SMA of the newly computed motion and determines if the `NarrowPhase` event is scheduled. If at any point the potential contact evaluation on the right side of the loop determines the entities will miss, the associated event silently exits and the `startMove` event currently on the `EventList` remains valid. If a contact is potentially indicated then the most recently computed event `endTime` is simply truncated to the projected contact time, and the `EventList` remains unchanged until an actual collision is determined. If no other event changes the motions of the involved colliding entities, the collision happens; both involved entities current `startMove` events on the `EventList` are cancelled and they both compute their next event based on the collision response parameters forwarded from the collision detection infrastructure.

This lazy-evaluation style of processing contact events minimizes the requirement for computationally expensive collision detection evaluations which may be subject to supersession by follow-on events. More detail on the steps and additional events used by the collision detection subsystem is covered in Section 4.4.4.3.1 and Section 4.4.4.3.2.

4.4.1.1.1 Integrating the Logical Process (LP) Figure 4.14's Model simulation event loop meets the expectation of an independent LP implemented with an individual `EventList` and thread. But rather than each LP having its own loop-thread-clock, in this architecture localized entities are scheduled together in an `EventList` bound to the same thread and evaluated against the singular timeline. At the implementation level this does not force a single thread and single `EventList`, rather all `EventLists` refer to the same clock/time-line, therefore acting in aggregate as if they are one `EventList`. The `EventLists` measure of progression is the `startTime` of the `CurrentEvent`, and the `startTime` is a variable that documents a time represented on the singular timeline, not a clock on its own. In this manner the singular timeline provides the time-synchronizing functionality to use multiple `EventLists`, each bound to their own compute thread.

This facilitates efficient interactions within the collision detection subsystem (`NarrowPhase` and `EvaluateContact` events) providing synchronization points where each entity independently generates events that may be identified as explicitly dependent on another entity's event for continued progression. Non-movement actions are also scheduled on the timeline and will generate the appropriate state changes, but do not directly cause physical movement of objects, for simplicity these are not depicted in Figure 4.14.

The concept of the LP is honored in spirit as each entity schedules its own movements and actions as if it is independent of all other entities, with the LP placing exactly one `startMove` event into the `EventList` and should the collision detection interaction events indicate that scheduled move is superseded, it is cancelled and replaced by the `EvaluateContact` event that in turn generates the follow-on `startMove` event, maintaining the property of the moving entity having exactly one `startMove` event on the `EventList`. No entity schedules the move of another, the `EvaluateContact` event signals to the entity that it must schedule its own next event, in this way the entity's own scheduling follows the LP pattern even though it is sharing an `EventList` and computation thread. While non-movement actions are not expected to require a loop similar to the `startMove`-loop, if the need arises the simulation designer can specify similar action loops that use the `EventList` similarly.

4.4.2 Entity State

Non-traditionally, this architecture's entity state is not primarily its orientation and trajectory, rather the state contains both those initial values, the current equation(s) of motion and any other variables necessary to fully specify the entity state. The biggest change from both traditional ADES and VE designs is not treating the currently assigned equation of motion as just some simulation code to be externally triggered at runtime for evaluation. This interpretation meets the requirements of state continuity and progression and mathematically matches the requirements of implementing adaptive step numeric methods in an intuitive way. State follows the concept of a continuous-across-time trajectory as depicted in Figure 4.10, and further defined in Section 2.3.4.2 which includes reference to the equations of motion as a component of the state.

When used in the context of motion evaluation, this interpretation of entity state provides a true continuous function trajectory for an entity through 4D space-time. The state trajectory

of Figure 4.10 now fully describes the entity state in a way that unambiguously provides not only a time-continuous set of basic entity attributes but explicitly contains the fully computed continuous in 4D space-time entity motion trajectory. Despite the entity state being continuous in the fashion of a non-differentiable step function, the resultant computed trajectory is continuous and differentiable within its defined interval, there is no possibility of computational cross-event ambiguity at event endpoints, and these properties are leveraged by the collision detection subsystem. As the computed trajectory is mathematically continuous, the collision detection tunneling and interpenetration difficulties described in subsection 2.3.4.1, as well any as extra overhead of CCD are trivially eliminated (see Sections 4.4.2 and 4.4.4.3.1).

While this interpretation falls outside of traditional implementation for both the ADES and VE simulation communities, it changes nothing in the motion evaluation results. The concept of computing the next motion event becomes more functional in style where everything necessary is specified via arguments to an initiating function, and the resulting event state remains static for its entire duration avoiding concerns about runtime side effects. This property is useful for implementing multi-threaded operations because an event state that remains static for its entire duration can be represented with immutable data, and sharing immutable data between threads and/or processes does not require efficiency sapping mutual exclusion locks.

The seemingly odd thing here is that an entity's trajectory through 4D spacetime is not traditionally considered as being immutable—because an entity moves. This issue disappears when considering that the purpose of the trajectory representation through 4D spacetime solely exists to answer one of three queries, 1) *where is it? (at some particular time)* typically used by the View module for construction the visualization; or the collision detection related queries of 2) *is it here?*; and 3) *Is it about to contact something else?* The collision detection aspect is covered in Section 4.4.4.3.1, the answer to *where is it?* is answered by querying the SMA.

4.4.3 The SpatialMotionApproximation (SMA) for Motion State

Having everything necessary to provide the continuous trajectory location answer for any arbitrary time in the SMA means there is no need to store snapshots and interpolations of

snapshots to resolve interaction queries, it is only necessary to ask and answer the queries exactly when required. Efficiently determining the exactly when required part is often a question of proximity, and the first step to understanding proximity is to approximate the volume that an entity may interact with by providing computationally simple geometry.

In this architecture the 4D space-time entity motion is represented with a dimensionality collapsing 3D fixed geometric representation called a capsule. A capsule is a line-swept sphere as depicted in 4.15 where the line is the linear motion approximation of the SMA and the radius is defined as the maximum distance from the entity center-of-mass that the entity volume can reach. Use of capsules was inspired by a collision detection tech report, *Fast Proximity Queries with Swept Sphere Volumes*, by Larsen *et al.* [79] of the University of North Carolina, however the methodology used here is different.

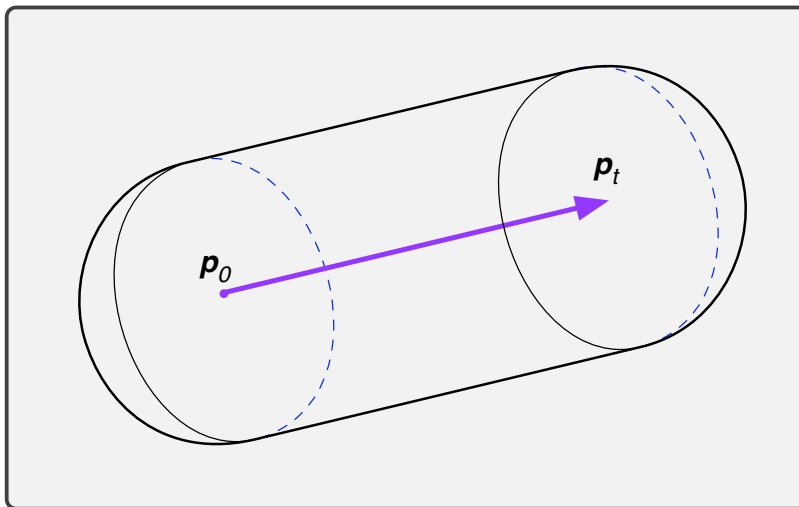


Figure 4.15. A line-swept sphere capsule.

Figure 4.15 visually depicts that the linear approximations from motion evaluations are expressed parametrically where the parametric- t of 1.0 matches the duration of the adaptively computed advance interval. Most entities do not move linearly for extended periods though, for example tossing a ball results in a parabolically curved trajectory under the influence of gravity. The MotionWitness (MW) is an augmented capsule that contains the necessary information to address non-linear motion approximations and is covered in Section 4.4.4.3.1 as part of the collision detection system.

4.4.4 Progression Constraints and the Logical Process (LP)

As mentioned in Section 2.4.2, ADES traditionally implements entities as individual independent simulators each referred to as a logical process. In the optimistic-scheduling ADES paradigm these LP are implemented as independent threads which do not constrain each other's event progression, and this results in a likelihood of causality violations if any LP simulates farther ahead than some interaction initiated by a different LP.

This dissertation's architecture changes the relationship by removing the independent thread per LP implementation convention. With respect to movement, following the event cycle of Figure 4.14 adheres to the concept of the ADES LP construct because the cycle is defined in a manner unconstrained by other entities, yet the ability to react to other entities movements and/or physical constraints such as collisions is explicitly maintained and scheduled via the collision detection subsystem managed functionality of the Figure 4.14 cycle. This event cycle does not require a dedicated thread for an entity to remain logically independent. Simply there is no place in the depicted loop for one entity to directly compute or constrain the operations of another, rather indirect interactions are identified and resulting parameters passed to each affected entity which will compute its appropriate response independently.

Collision detection management provides the most obvious of the indirect inter-LP effects when a collision is detected, and may seem to require interrelated follow-on computations, but this set of interaction-indicated computations are limited in scope to determining what is needed to resolve the interaction and provide the entities the necessary response parameters so that the follow-on computations for each entity are once again independent. To eliminate the possibility for inter-thread communication synchronization issues or inefficiencies, all of the entity computation including collision detection management for a Region is done within a single thread of execution.

4.4.4.1 Per-Region Event Queue Management, the EventList

This design uses a time-sorted event queue (the EventList) bound to one execution thread per physical Region, and each Region also has an dedicated incoming message queue (or inbox) for receiving event messages which cross into the Region's volume.

The execution thread for a Region follows the typical design pattern of a loop pulling and executing exactly one event per iteration. All of the thread's internal business is synchronized

via the sequential nature of progressing through the loop, and no external threads are allowed to access the internal working variables, eliminating the need for mutual-exclusion synchronization of event evaluation. This loop is implemented to serve as the `EventList` scheduler and thus called a `RegionalSchedule`, the algorithm outline is provided in Algorithm 3.

Algorithm 3: The `RegionalSchedule`, a wall-clock synchronized `EventList`.

```

1 begin
2   oneFrame_ms  $\leftarrow$  frame length in ms
3   while running do
4     do
5       while inbox.hasContents() do                                /* process inbox */
6         e  $\leftarrow$  inbox.remove()
7         EventList.add(e)
8       maxSleep  $\leftarrow$  delay duration from now for synch with wall-clock
9       if maxSleep < 0.0 then
10        rework()                                                /* see Algorithm 2 */
11        if maxSleep > oneFrame_ms then                            /* wall-clock sync */
12          SLEEP(maxSleep)                                       /* also wakes on inbox.add() */
13      while maxSleep > oneFrame_ms
14        CurrentEvent  $\leftarrow$  EventList.pop()
15        CurrentEvent.evaluate()

```

Only a single event is evaluated per `RegionalSchedule` loop iteration, positively providing sequential control over the `EventList` and entity states affected by the currently evaluated event. With respect to the event cycle of Figure 4.14 introduced in Section 4.4.4, the circular entries are the events, so only one of the events within the Figure 4.14 event cycle is evaluated per iteration through the `RegionalSchedule` loop.

The `RegionalSchedule` evaluates events out of the `EventList` one-at-a-time, and the event itself has a `startTime` for when on the singular virtual timeline it is to be commit to the state of the simulation. This allows each LP to map its events against the single global virtual timeline and submit these events to the correct `Region` and `EventList`. Normally this means the LP's next event is added back into the current `Region`'s `EventList`, but if

the event evaluation indicates a `Region` boundary intersection, a special copy of the event called a `Liminal` is also put into the incoming `Region`'s `inbox`.

If `NarrowPhase` or `EvaluateContact` events are indicated, they are added to the `EventList` with a simulation time of indicated overlap or contact relatively. If either `NarrowPhase` or `EvaluateContact` events are evaluated and determine the scheduling condition is false, the event silently terminates without scheduling the follow-on event, and the self-scheduled `startMove` event remains in the `EventList` ensuring the entity action progression is properly tracked through continuous time.

Algorithm 3 references a display frame-rate in the `oneFrame_ms` variable. Unlike frame-rate dependent computations as discussed in Section 2.3.4.1 where the frame-rate or a multiple is used directly in computations, the `oneFrame_ms` variable serves as a scheduling buffer only used to prevent poor-efficiency thread micro-sleeps with durations less than a display frame interval. Poor-efficiency thread micro-sleeps may result in an “awake” thread being returned to the operating systems scheduler but not executing on the CPU in time for the intended event, creating an artificially late event evaluation that may lead to stale state data being used by the rendering engine computing the next frame. Implementing a fixed value for the `oneFrame_ms` variable is adequate as there is no penalty per se for any slight overstatement of the buffering, the overstatement is simply a couple additional milliseconds added to the `LCW`.

The `maxSleep` variable of Algorithm 3 is computed with reference to the `LCW` although the specific computation details are not represented in Algorithm 3 for conciseness. The goal is to maximize the duration that a `RegionalSchedule` does not need the CPU thus increasing overall simulation efficiency, while minimizing the probability of late wake-ups with the `oneFrame_ms` variable.

The `inbox.add()` behavior triggered by an external `Region` sending a message to a target `Region`, explicitly issues a `thread.wake()` call to prevent a sleeping `RegionalSchedule` from missing incoming events until a scheduled wake-time. Thus the Algorithm 3 loop is constructed in the same manner as any thread that might need to handle spurious wake-ups, both to manage any of those, but to also go back to sleep after processing the `inbox` if that is the result of recomputing the `maxSleep` variable.

4.4.4.2 Thread and EventList Isolation

The `inbox` provides the safe separation preventing commingling of one thread of execution with another thread's internal variables. A thread of execution is an abstract virtual construct in computer science, providing a name for a sequence of instructions executing through one CPU core. This is not the same as looking at the source code for a program and seeing the sequence of coded instructions. It is possible that the same code instructions are in execution within multiple different threads concurrently (or even simultaneously.) Each thread has access to a different part of the program's stack, which is how each thread of execution keeps the business of its execution streams separate from any other threads' execution streams.

Undesirable race conditions can arise if one thread of execution begins executing and affecting variables that are supposed to be unique to a different thread of execution. This may arise if for example `RegionalSchedule-A` was to directly add an event to `RegionalSchedule-B` by naïvely calling the shared code responsible for adding an event to the `EventList`. The race condition(s) arise if `RegionalSchedule-B` accesses any of its own memory variables that are in also update by the execution within `RegionalSchedule-A`. Two solutions to this problem exist: provide mutual exclusion locking of shared data, or write code that eliminates the possibility of inter-thread sharing of data.

There are two potential performance drawbacks to the mutual exclusion locking option.

- A single thread does not require mutual exclusion locking for its own internal operation because it is already sequential. Mutual exclusion locking adds nothing except locking overhead to a properly sequentialized single thread. This indicates that if shared data between threads can be avoided, the locking overhead is not necessary for the single properly serialized thread accessing its own non-shared data.
- If data must be shared between threads, code synchronization locking is a performance bottleneck if the critical sections are required to be entered rapidly and repeatedly by multiple threads. Especially if unfortunate code design raises the probability that a thread may leave the CPU while it is holding a lock. This consideration favors very short critical sections with code that is not located within tight loops.

The interaction of these two performance considerations favor minimizing sharing, and avoidance of quickly-repeated sharing. The `RegionalSchedules` are coded and intended

to each run as an isolated loop interacting with the data object(s) uniquely assigned to that `RegionalSchedule`. This makes the internal operations of a `RegionalSchedule` non-shared, simplifies the coding requirements and avoids the need provide mutual exclusion to support basic scheduling loop interactions, enhancing overall efficiency.

When a entity must inform a `Region` other than its owning `Region` of a future event, a copy of the necessary information is placed in an `inbox` to the incoming `Region`. The information copy eliminates the need for mutual exclusion protecting it, and the `add()` behavior interaction with the `inbox` is short and not inside a rapidly executing tight loop. This pattern is a variation of the Protected Service Process Model as covered in Section 3.1.4.1 and Section 3.1.4.2.

4.4.4.3 Virtual Time, Multiple Threads and Localization

The single timeline Clock Condition as introduced in Section 4.3.1 faces a seeming paradoxical implementation situation when multiple execution threads are implemented, as the threads all individually compute local events, how can these multiple, physically separated in CPU and memory allocation entities be completely synchronized with events on the single timeline if the system is not sharing an irresponsible amount of scheduling data? The complexities of this issue is a likely motivation for the previously used concept of independent LPs each with their own clock, enforcing assumed simplicity and eliminating the need for locks and blocking behaviors that sap performance. This paradox does not exist, and the supposed simplicity of totally independent LPs actually contributes to increased complexity for synchronization requirements between computational threads while dogmatically withholding the data via independence, that is necessary to efficiently manage thread synchronization. The analytical simulation communities inability to find “a sufficiently general solution to the PDES problem” or “lead to new insights in parallel computation as a whole” [61] flows from conflating model independence and code construct independence all with respect to the concept and implementation of the LP.

Considering the LP independence as a model-level requirement rather than a code implementation requirement provides a simplification opportunity by allowing threading code to manage synchronization *outside* of the simulation models themselves. Illustrating this model-level independence we return to the jugglers, this time on opposite sides of a high

wall so that they cannot see each other as illustrated in Figure 4.16. There is also an emcee who will provide verbal commands to the jugglers to toss a club over the wall to their unseen partner.

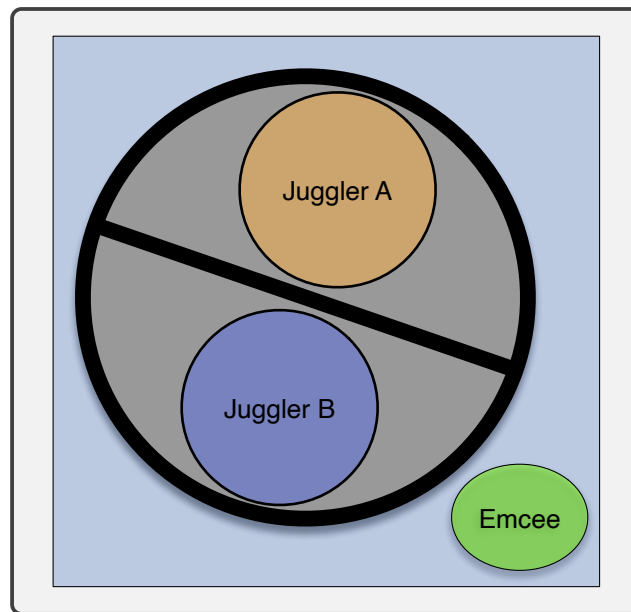


Figure 4.16. Jugglers and emcee scenario.

In this example each juggler is the equivalent of a totally independent LP, as is the Emcee. Each juggler operates within their own local relative timing loop as they juggle their clubs, operating independently. Occasionally the Emcee directs NOW and each juggler tosses a club over the wall, blindly to their partner and each skillfully catch and throw the club into their personal cycle. In the real world the jugglers are not appearing to refer to any shared clock to ensure their throws are at the correct moments for the partner to catch on-cycle-time, but there is music, and the jugglers both juggle in-time to the music. The music explicitly provides the knowledge of the shared timeline, the Emcee only needs to see one juggler to know when to say NOW that will work for both jugglers.

A simulated juggler on singular virtual timeline does not need to literally register the music the same way a real-world juggler does, any shared signal for a start time and consistent catch-throw cycle code will simulate the same shared-timing of the real world jugglers. A simulated juggler also does not need the real-world physiological muscle memory to

precisely toss the club to the exact point in space time for the partner to catch it, rather the infinitely repeatable simulated throw of a club using code that computes the correct trajectory to send the club precipitates transmitting that event over a communication channel to the **Region inbox** of the thread that is computing the partner's receiving of the clubs.

Invoking the term synchronization requires care, as synchronization has multiple valid meanings and context-dependent choices for techniques. The 'soft' version of synchronization is that exhibited through the event-related causality chains that the jugglers create, especially the causality of exchanging clubs. There are no low-level inter-thread or inter-process shared-memory synchronization techniques required to memorialize these before-after or concurrent event relationships; order of events and events-dependencies sufficiently cover the synchronization between the jugglers. The 'hard' version of synchronization is the proper management of shared data to avoid errant race conditions that corrupt state in a non-deterministic manner.

In the simulation's runtime context data is either computed in one thread, or shared between threads. Local computations within a single thread are serialized by-definition, because the data is not shared when it resides within a single thread. This simplifies computations within a single **Region** because no shared-data protection is required, and no shared-data protection means there is no loss of efficiency or bottlenecking from shared data lock contention. This provides motivation for single thread evaluation of simulation code for objects that are in proximity to maximize the efficiency of avoiding shared-data protection.

Obviously there is a threshold where too much computation load can be assigned to a single thread and it is incapable of meeting real-time rates, sharing the computation load between multiple threads helps as long as the shared-memory synchronization requirements between threads do not obliterate performance gains from parallelization. With the **Region** concept, no data shared when an event crosses a **Region** boundary requiring one or more **Regions** to need to interact with the transiting entity. Rather the "shared" data provided to the gaining **Region's inbox** is actually an immutable copy that does not require mutual exclusion protection.

The technique of providing a copy of the shared data rather than a shared reference limits the computational penalties to that of the data duplication. This duplicated data is then passed via the **Region's inbox** that prevents sending threads from interacting with receiving

threads data and vice versa. Receiving Regions may register some interaction with the shared copy of the incoming entity before the entity fully enters the incoming the region Region. Interactions are communicated back to the owning Region and the `startTime` of each potential new event is used to resolve which events are committed and which may need to be reworked due to an emergent change from another Region. Only a single Region at a time retains responsibility for an entity state on a host, and in extension one host in the distributed system maintains final authority for an entity state within the simulation.

While this reads like it could cause rework cascades and be very inefficient, the practicality is that most imagined situations that may trigger inefficiencies either require intentional setup near the corner(s) of multiple regions; or are dense regions of computation that should be rightfully computed by some other appropriate method. The out-of-scope (per Section 1.3.2, Scope) inside-of-an-avalanche type of degenerate situation is a bad fit for a ADES simulation, this is not surprising or news. Even then the capability for ADES to manage in rollback necessary scenarios is not in question, Jefferson's *Time Warp* algorithm and its potential for unconstrained rollback is faced with the same issues. For decades *Time Warp* is considered a gold standard for performance comparisons. *Time Warp* has fewer constraints on thread progression into the future than *Lag Erasure*, improved progression constraints will only limit the scope of potential rollback-related inefficiencies leaving *Lag Erasure* to compare favorably with the *Time Warp* gold standard.

4.4.4.3.1 Collision Detection within a Region Collision detection determines entity physical contact and causality chains defined with respect to entity contact. Collision detection in an ADES style simulation is typically avoided without comment, it simply is not a thing in the literature. How to do collision detection approximation efficiently is well developed for synchronous fixed time step simulations, and the techniques do not map well to an ADES paradigm and adaptive step lengths. Section 2.3.4.5 covered the research arc of Asynchronous Time Advance in Physics-Based Simulations.

ADES collision detection requires replacing traditional collision detection optimization techniques with new techniques, the most profound difference is in considering where something is not, as opposed to the traditional optimization question of what is sufficiently close? The *Lag Erasure* collision detection architecture maintains a collision detection tree

which documents non-contacting continuous motion of entities through 4D space-time. The Leaf nodes of this tree are constructed about the MotionWitness.

The MotionWitness (MW) Visualization of Motion State. The MotionWitness extends the basic capsule and its linear SMA to include the maximum swept-volume that a non-linearly moving entity may possibly interact with or transiently occupy in that event interval. Figure 4.17 visually depicts the maximal swept space the entity traverses over the event interval.

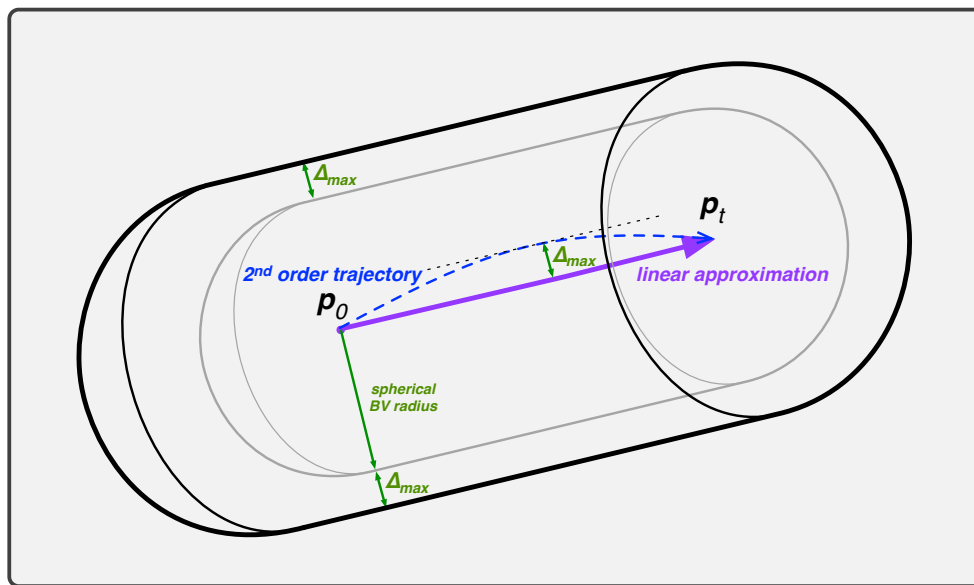


Figure 4.17. The MotionWitness containing non-linear entity motion.

The MotionWitness is still a capsule, it is an expanded version that includes the additional geometric volume necessary to envelop the non-linear motion. The additional volume necessary is determined by parametrically computing the maximum trajectory deviation from the SMA's linear approximation (Δ_{max} at the time of the zero derivative with respect to the linear approximation) and use that Δ_{max} as the magnitude to buffer the radius of the capsule. The result is the buffering depicted in Figure 4.17 and this buffered capsule represents the maximal swept space the entity traverses over the event interval. This geometric event trajectory depiction is the fundamental geometric construct bridging the abstract mathematics of ADES event evaluation and efficient physics-based methods necessary for efficient collision detection supporting VEs and AR. Usefully, if any set of MotionWitnesses does

not intersect, the entities whose motion are represented by them cannot possibly intersect in the represented time intervals. This non-intersection is the genesis of the name **MotionWitness**, which is a witness to non-colliding motion through 4D space-time. The set of **MotionWitness**'s is maintained for a **Region**, wrapped by the **AABB** components of the **R*-Tree** data structure as introduced in Section 3.1.3.2.1.

The R*-Tree and Collision Broad Phase. The collision detection sequence of action as described in Section 3.1.3.2.3 is effected by insertion or update of a **MotionWitness** into a 3D implementation of an **R*-Tree** per Beckmann [67]. The presented **MotionWitness** is wrapped in a **AABB**, and this is the candidate **Leaf** used to meet the rectangular bounding area requirements for the **R*-Tree** data structure. The collision detection broad phase, consists of determining the correct tree position for the **Leaf** and simultaneously determining if the **Leaf**s and their contained **MotionWitness** overlaps any other contained **MotionWitness** in the tree. If the overlap answer is no, the presented **Leaf** and **MotionWitness**s are representing non-colliding motion and the insertion or update is complete.

If any **Leaf** overlap answer is yes it is added to the set of **Leaf**s to assess against. The new **MotionWitness** and each identified existing-in-tree collision candidate are assessed to determine the time that the moving spherical bounding volumes contact. One possible outcome is that despite **AABB** and **MotionWitness** overlapping, there is no potential contact of the entities, and they simply pass as do ships in the night. Any noted bounding volume overlaps are compared for time-of-contact and the earliest overlap is selected for creation of a **NarrowPhase** event.

The **NarrowPhase** event is placed onto the **EventList**, and the inserted **MotionWitness** is truncated to end at the same time the **NarrowPhase** event is scheduled. The **MotionWitness** truncation also shortens the duration of the **SMA** and causes the **Leaf AABB** to shrink. The broad phase is now complete and every **MotionWitness** represented in the **R*-Tree** is guaranteed to represent non-colliding motion.

The low-level math of this broad phase process is accomplished parametrically and avoids expensive root-finding to solve the general form of closest-approach distance presented as:

$$\|\mathbf{w}_0\| = \|\mathbf{p}_0 - \mathbf{q}_0 + (\mathbf{v}_p - \mathbf{v}_q)t + \frac{1}{2}(\mathbf{a}_p - \mathbf{a}_q)t^2\|, \quad (4.4)$$

from Peitso and Brutzman [80], [81]. Avoiding the need to test three root solutions of Equation 4.4 begins with the base requirement that if the objects are to intersect they must close from a starting distance and direction to the distance defined by the sum of their MotionWitness bounding radii. Because this is one potential answer to Equation 4.4, the displacement vector carries the same designation of $\mathbf{w}_0$, and its length, $\|\mathbf{w}_0\|$, is the magnitude that must be closed between the two entities for the MotionWitness bounding radii intersection to occur.

As second-order differential equations, the non-linear motion approximations are constrained to constant acceleration over the interval, meaning the combined acceleration's effect on the combined velocity vector generates a displacement vector with the same magnitude as the target $\|\mathbf{w}_0\|$ a maximum of once, or zero times in cases of parallel/opening relative motion. When beginning to solve for the target $\|\mathbf{w}_0\|$, the time of the eventual solution is unknown. To find the time t that the displacement vector reaches the target value for intersection, start with

$$\mathbf{v} = \mathbf{v}_0 + \mathbf{a}t, \quad (4.5)$$

for motion under constant acceleration, rearranging to isolate t results in

$$(\mathbf{v} - \mathbf{v}_0)/\mathbf{a} = t, \quad (4.6)$$

and substituting the lhs into the basic accelerated parametric motion of

$$\mathbf{p} - \mathbf{p}_0 = \mathbf{v}_0t + \frac{1}{2}\mathbf{a}t^2, \quad (4.7)$$

derives:

$$\mathbf{a}(\mathbf{p} - \mathbf{p}_0) = \frac{1}{2}(\mathbf{v}^2 - \mathbf{v}_0^2), \quad (4.8)$$

which does not require solving for t . Rearrange and solve Equation 4.8 for the magnitude on both sides, thus isolating the $\|\mathbf{v}^2\|$ term where $\mathbf{v}$ is the joint velocity which will occur at the target intersection distance. By working with the magnitudes, there is no need to know the exact point $\mathbf{p}$ for computing the left-hand side, rather substitute the magnitude of $\mathbf{w}_0$, the

initial relative distance to contact. Finally, take the magnitude of both sides of Equation 4.6, substitute in the previously computed $\|\mathbf{v}^2\|$ solving for t without higher-order root-finding. If the distance between the objects at time t is less than the sum of the BVH radii, correct t back to correspond with the necessary sum of the BVH radii distance [81]. Together, the result is Algorithm 4:

Algorithm 4: Time of Bounding Sphere to Bounding Sphere Contact.

```

1 Function timeAtContact(  $\mathbf{p}_1, \mathbf{p}_2, \mathbf{v}_1, \mathbf{v}_2, \mathbf{a}_1, \mathbf{a}_2, r_1, r_2$ )
2  $\mathbf{w}_0 \leftarrow \mathbf{p}_1 - \mathbf{p}_2, \mathbf{v} \leftarrow \mathbf{v}_2 - \mathbf{v}_1, \mathbf{a} \leftarrow \mathbf{a}_2 - \mathbf{a}_1, v_w \leftarrow \mathbf{v} \cdot \mathbf{w}_0, a_w \leftarrow \mathbf{a} \cdot \mathbf{w}_0;$ 
3 if  $a_w$  and  $v_w \leq 0$  then                                     /* opening motion */
4   | return  $-\infty$ ;
5 else if  $\|\mathbf{a}\| = 0$  then                                       /* non-accelerated closure */
6   | if  $\|\mathbf{v}\| = 0$  then                                         /* parallel motion */
7   | | return  $-\infty$ ;
8   | else                                                       /* velocity-only closure */
9   |    $t \leftarrow$  compute standard linear time-of-CPA;
10  |   compute new posits for both objects at time  $t$ ;
11  |   distance  $\leftarrow$  compute distance between new posits;
12  |   if distance  $> (r_1 + r_2)$  then
13  |   | return  $-\infty$ ;
14  |   else if distance  $< (r_1 + r_2)$  then
15  |   | return  $t - (\sqrt{(r_1 + r_2)^2 - \text{distance}^2} / \|\mathbf{v}\|);$ 
16  |   else
17  |   | return  $t$ ;
18 else                                                         /* accelerated closure */
19 |    $\text{magToTravel} \leftarrow \|\mathbf{w}_0\| - (r_1 + r_2);$ 
20 |    $\text{velocityMag} \leftarrow \sqrt{2 \cdot \|\mathbf{a}\| \cdot \text{magToTravel} + v_w};$ 
21 |    $t \leftarrow (\text{velocityMag} - v_w) / \|\mathbf{a}\|;$ 
22 |   compute new posits for both objects at time  $t$ ;
23 |   distance  $\leftarrow$  compute distance between new posits;
24 |   if distance  $> (r_1 + r_2)$  then
25 |   | return  $-\infty$ ;
26 |   else
27 |   | return  $t$ ;

```

Resolving the solution in this manner is not a one-shot and done, the non-linear motion and accelerations may conspire to create a complex closing situation such as described in Figure 4.18 which illustrates the worst case for successive determinations of t and $\mathbf{w}_0$.

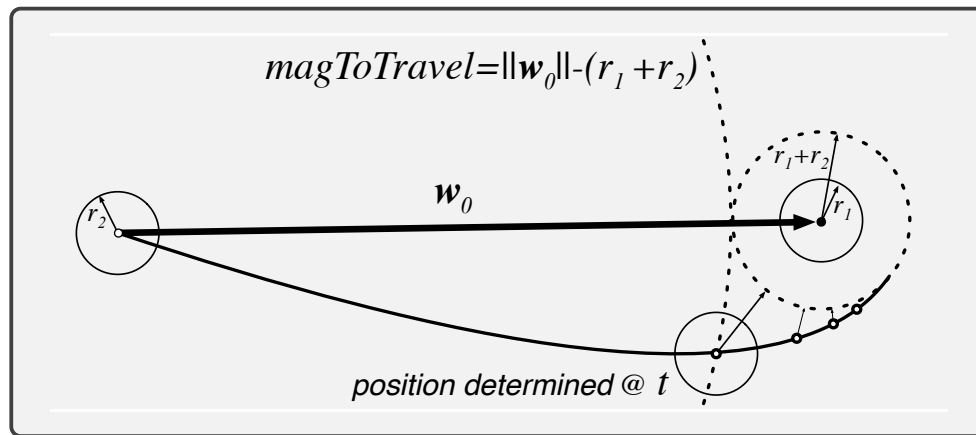


Figure 4.18. The worst case for relative motion closure evaluations, a path which approaches tangentially with a constant acceleration leading to a curved path with successively closer tangential passages. This is advantageously terminated with tolerance-based contact comparisons.

Without compensation techniques the iterations may run intolerably long, until absolute error gets down to the minuscule levels out in the far least-significant reaches of the mantissa's binary bits. Using thresholding on the computations, sets a meaningful cutoff to computation that in practical tests has ended the worst-case of Figure 4.18 in seven or fewer iterations dependent on the thresholds set. The practicalities are that the if Figure 4.18 represents an planetary fly-by, atmospheric boundary differences represented beyond the 15th significant binary bit is not likely relevant nor precisely known well enough that setting the $\mathbf{w}_0$ minimum tolerance to the thickness of automotive paint (approx. 2×10^{-4} meters) for terminating the computation iteration is going to cause a simulation failure. Where a simulation designer believes otherwise there is the flexibility to define the tolerance desired, provide other numeric algorithms, or define enclosing regions where custom techniques may be employed. As always, **Lag Erasure** is not the general solution to every simulation problem, simulation designers have the responsibility to guarantee they are using the right techniques and numeric methods.

The Collision Narrow Phase and Contact Response. The narrow phase may include detailed collision detection polygon-to-polygon computations and is therefore considered computationally expensive. Some simulations choose to employ BVH trees for successively better fitting proxy collision objects. Either way the goal is to exit the narrow phase at the earliest opportunity whenever possible. Collision proxy computations for point, line and rectangle-swept volumes (spheres, capsules, and lozenges) are well defined and their use as collision proxies is common enough it is not covered further here.

Because the `NarrowPhase` event memorializes *when* the bounding volume proxies of the involved entities are to intersect, actual potential contact is nearly always further into the future. The “in the future” and additional futurity injected via use of the LCW in determining when to evaluate an event is why the events are commit to the `EventList` instead of evaluated immediately. There is a possibility that a `NarrowPhase` event may be pre-empted by another interloping event, and in so avoiding a potentially expensive computation or additional work to discard in accomplishing rework.

When the `NarrowPhase` is pulled from the `EventList` for evaluation the SMA-IDs for both involved entities are checked to see if they remained the same. If not, the changed SMA-ID indicates to exit the computation. This event-internal validation eliminates the need to endlessly search the `EventList` for potential `NarrowPhase` events every time an entity’s motion updates, thus eliminating admin work and increasing performance within the collision detection system. When both SMA are unchanged the detailed computations are executed to determine the time-of-contact, and a `EvaluateContact` event is put onto the `EventList`.

When the `EvaluateContact` event is pulled for evaluation it is verified as still relevant the in the same manner used for `NarrowPhase` events. If not superseded, any previous `StartMove` events for the involved entities are cancelled, the collision response forces and accelerations are computed for the involved entities and used to immediately evaluate new `StartMove` events for the involved entities. Optimizations for never-moving entities are simple and compatible with the presented flow of operations.

4.4.4.3.2 In-Region Localization Summary The goal of the localizing by `Region` is to provide a first order grouping of entities that are more likely to interact with each other, and handle that work within the single thread serving the `Region`. The `Region`’s

single thread executes events sequentially and this provides lock-free interactions within the `Region` which enhances overall efficiency. Entities are passed from one `Region` to another by placing the crossover-event into a drop-box style `inbox` implemented with a concurrent (lockless) queue. The `inbox` is emptied and the incoming events locally scheduled onto the `EventList` after each `CurrentEvent` evaluation.

4.4.4.3.3 Entities moving between Regions Under the `Region` with `inbox` scheme the average expected n of the worst case collision-detection driven $O(n^2)$ synchronization problem is dramatically limited by the small number of entities crossing across a single `Region` boundary at that point in time. Yes, this engineering assumption requires that the number of `Region` boundary crossings remains small and does not trigger extended cascades of synchronization requirements. The design space limits from the scope discussion of Chapter 1 are a general indication of circumstances where this architecture will suffer the $O(n^2)$ synchronization explosion difficulty. Examples of what to avoid are simulating the internals of an avalanche as it progresses down a mountainside or executing CFD simulations. These modalities were previously identified by Mirtich [7] as challenging for event-based simulations. Those types of dense simulation situations may be partitioned off into a `Region`, and that `Region` evaluated using the appropriately efficient techniques. This regionalization synchronizes with the event based simulation at the points where entities enter and/or leave this custom partitioned `Region`, events which will be sparse on the timeline compared to those fully internal computations required.

Liminal Entities. The entity that is crossing between `Regions` is termed a `Liminal`. How the `Liminal` entities are used is described in Section 3.1.4.3 and Section 3.1.6.2.2. There are no special techniques or algorithms employed to make `Liminal` entities work beyond their inheritance status as `Liminal` allowing a customization of the `startMove` event code to not generate a new event within the gaining `Region`. This way is a toll-free manner to differentiate the incoming `Liminal` entity actions from the standard in-`Region` entity actions.

The `Liminal` entity has its motion represented in the gaining `Region R*-Tree` tree as well as the motion represented in the `Region` it is departing from. `NarrowPhase` event computations are unchanged from descriptions above, but the `Liminal` version of scheduling the

NarrowPhase event includes triggering the notification back to the originating **Region** that a collision is projected and this helps manage the maximum amount of rework that might result from entity **Region** boundary crossings.

4.4.4.3.4 Performance Considerations for Region-Based Localizations Considering the circus simulation of Figure 4.9, Panel **D** represents a **Region** where both explicit and implicit event synchronization is occurring within the **Region** where all internal events are computed serially in the **Region**'s thread. As long as a clown car does not deposit an egregious number of additional jugglers into Panel **D** suddenly, so that the single thread is overtaxed and cannot keep up with the singular shared virtual timeline, that **Region** may progress independently and without external reference to any other **Region**. Plotting events against the shared virtual timeline is sufficient to ensure proper causal event and concurrent event resolution across the simulation.

Surprise and Rework. When a juggler in one **Region** (Panel **D**) decides to interact with juggler(s) in another **Region**, such as Panel **C**, the **Liminal** entity crossing event and deposit of it into the incoming **Region** inbox memorializes the positive synchronization between the two **Regions**. The discussion in Section 3.2.1.1 of what constitutes a *surprise* to the remainder of the simulation, thus requiring rework is straight-forward and there is no deep technical justification or algorithm necessary beyond that discussion.

It is important to reiterate that there are no cross-thread notifications or synchronizations necessary to execute the rework procedures, the rework is entirely local to any thread that determines rework is necessary. This is a significant difference from rollback mechanisms such as described by Jefferson for *Time Warp* [31] which requires additional communications in the form of antimessages where more than one thread are affected by the need to process rollback-related resolution work.

Detecting the Need to Rework. With each regional thread executing with respect to the wall clock, and events being pulled off the **EventList** queue for evaluation ahead-of-time by the magnitude of the **LCW** there is a narrowly limited between-events-interval where an event being pulled from the **inbox** might be scheduled before but be evaluated after the most recently completed **CurrentEvent**, this timing-interaction creates a causality issue.

This constitutes a *surprise*, and requires rework. Whether the surprise is generated by a transient network jitter causing late arrival of an transmit event or from an external input made at a remote host it is managed similarly.

Rework is only potentially necessary for local entities that have an event `startTime` later than the triggering event `startTime`, but are already evaluated due to being pulled from the `EventList` due to the LCW. Those entities' `MotionWitness`'s currently in the `R*-Tree` collision detection tree are a reflection a future interval which might not remain valid on the singular virtual timeline following the `startTime` of the surprising event. These `startTimes` identify a set of already locally evaluated entities which constitutes a minimum required subset of entities that must be processed to determine if there is an emergent causality issue to resolve, or not.

Assessing this minimum required subset of entity's previous `MotionWitness` to check for overlap with the incoming event `MotionWitness` is the primary work to be accomplished. The most likely result of this rework is that there is no entity overlap noted therefore no potential interaction, and that aspect of the rework would be finished. In the case one or more overlaps are discovered which could create a potential interaction, the earliest overlap possibility is processed and any resulting new `MotionWitnesses` checked in-turn against the remainder of the minimum required subset. It might be possible that an event was computed for an entity which depended on the entering entity's motion, but not involved in a collision. Imagine a troupe of clowns disgorging from their clown car in Panel D of Figure 4.9 running about the Region, hurtling to-and-fro conducting maximum hijinks for best comedic effect, but not actually affecting any of the jugglers, at all.

Importantly, *the minimum required subset can never grow while resolving it* because it is constrained in time between the `startTimes` of the entering event and the most recently evaluated `CurrentEvent`. Checking this interval identifies all entities that started new events requiring verification. While this reads like it could possibly cascade into a massive chain reaction of rework, that exploding cascade possibility approaches nil within the previously stated design space for this architecture (Section 1.3.2 refers.)

Just as in the real world, objects tend to be separated and move through space sparsely unless not moving and already in resting contact. Also, the potential motion distances in context of the LCW are small compared to the overall space and spacing between the moving

objects. As objects move faster, they tend to be in less densely occupied areas. Consider the space in front of an automobile on the highway at 96kph (60mph), even during a large LCW of 300ms where the auto will travel approximately 8.0m through unoccupied space, unless it is in extremis or during a collision, that future travel space is empty. And in general, the in-extremis or collision status is an exceptionally rare event across the whole of society, even if a personal assessment of other drivers is that they are out to hit us regularly, it turns out that they don't. Alternately walking through reasonable passageway traffic between classes and using a 100ms LCW the distance a student moves is only on the order of 4-5cm and if not walking where a blind door could swing open and create a great surprise, the students will self-navigate to maintain more personal standoff than that of the LCW transit magnitudes.

The hard limit on the future interval that may be affected, and the relative sparseness of how the space is filled with moving entities are what advantageously constrains the size of the minimum required subset of entities for rework. Region sizing design should avoid small Regions where normal entity velocities over LCW-sized (or less) motion durations end up crossing the majority of the Region volume and potentially interacting with the majority of the entities in the Region. With appropriately sized Regions that approximate real world sparseness, it is probable that the majority of events in the minimum required subset of entities are not near the incoming event region entry point and will not even interact with the incoming entity at all during the scope of the LCW. This is how the sparseness and a small LCW interval limits the potential for large cascades.

Incoming events will seldom all affect the same interior volume, which is an identical challenge for any simulation paradigm. With 6 faces and eight corners to a box-shaped Region, it is not likely that anything other than a specifically contrived situation such as multiple entities all simultaneously firing automatic weapons at the same corner the Region, or an infinite capacity rapid deployment clown car, will repeatedly place multiple entities into the same locality at nearly the same time to the point of bogging down the simulation so that causality maintenance cannot keep up with the wall clock. If the simulation designer knows the scenario presents this specific possibility, then Region sizing, placement planning, and appropriate behavior limits are necessary to address this custom situation.

4.5 Additional Design Considerations

There are additional interesting properties that result from the choices used in designing the *Lag Erasure* system, which themselves are not driving design factors, but do provide an indication that the *Lag Erasure* system may be more powerful than the ability to erase lag itself indicates. These topics include area of interest (AoI) management and use of computational tolerance.

4.5.1 Regions for Area of Interest Management

Area of interest management is a long-standing topic for distributed simulations, especially as a means to determine what messages are required to be routed to which entities, and therefore hosts. AoI management methods and challenges received deep examination in a 1995 dissertation by Macedonia [72] and a 2000 dissertation by Morse [82]. The regional organization inherent in the *Lag Erasure* architecture's resembles both AoI management partitioning per Macedonia [72] and AoI management cells as defined in Morse [82], indicating that this architecture may be natively compatible with scaling AoI management networking methods rather than requiring AoI management to be bolted-on after-the-fact.

4.5.1.1 Potential Region Layouts

AoI Localization begins with an identification of the physical Regions within the simulation space, collectively representing a coarse 3D voxel-like grid. Additional Regions may also be contained within these basic Regions, for example a building could be defined within its own localized Region so that building-internal simulation events are evaluated together and on a separate thread from external events. Transitions between the internal and external Regions can be handled with a Doors-and-Portals style interface.

Of targeted interest to Morse's AoI management cells concept [82] is that not all Region boundaries are conceptually restricted to rectangular prism shapes. Alternately, a sensor volume may be defined and pre-computed into a data structure using a non-Cartesian Polar-coordinate system and geometry, defining a new non-rectangular Region which may be geo-located within the standard rectangular Cartesian space coordinates. When entities in rectangular Cartesian space intersect a geo-located proxy shape, a sphere in the case of a contained polar coordinate defined Region, a Region boundary crossing as described in Section 4.4.4.3.3 is triggered. The boundary crossing trigger executes the coordinate

transform operation and from there the polar coordinate data structure operates fully within its native coordinate system for maximum efficiency. This capability is expected to perform well for many physics-based sensor-related interactions, with additional potential efficiencies gained via offline pre-computation of sensor envelopes, providing look-up table-speed interactions for otherwise heavyweight computations if done at runtime. All this is potential benefit is accomplished with zero computational penalty to any entity that is outside the custom region boundaries.

These possibilities constitute an opportunity for significant future investigation. The ***Lag Erasure*** architectures flexible regionalized localization scheme provides the means for grouping entities that are likely to interact, and even more valuably, identify entities that will not interact due to lack of proximity or separation by physical barrier. Section 4.4.4.3 covers the operationalization of the basic regional organization into implementable multi-threaded code.

4.5.2 Thresholding and Error Tolerance

Threshold related checks such as *has a vertex of polygon A interpenetrated some polygon B?* compute in full binary fidelity to yield a true or false answer. This seems simple and the correct thing to do, but are the full depth of the mantissa's binary bits always useful? Or does naïve reliance on those deep bits handcuff human-scale simulation users and designers in a way they do not even realize?

Maintaining the absolute achievable binary precision as the threshold may end with unpredictable results if the true answer is between representable binary values, and the direction that the IEEE 754-2019 Standard for Floating-Point Arithmetic [83] tells the hardware to round up or down to the appropriate representable value is opposite the slope of the approximated curve. This can be a truly tiny change in magnitude, but if the slope between successive values flips unexpectedly and the algorithm is not built to catch this flip, an evaluation can unexpectedly “explode.” The “explosion” trigger may be of a magnitude that is irrelevant to the intended result, but binary math and decimal math are what they are, imperfect approximations of each other.

Intentional use of thresholding at values that constitute useful filters selected for fidelity necessary to answer the question posed allow simulation designers to avoid arbitrarily

invoking numeric methods to compute until deep-in-the-mantissa-bits indicate an absolute error tolerance has finally been met to terminate an evaluation. These absolute tolerances may be on the order of 1×10^{-15} or smaller resolving to femto-level dimensional differences. It must always be remembered that the real-world items described in a simulation designed for human interaction, or autonomous vehicle interaction do not all have mathematically perfect surfaces themselves, their surfaces are adequately represented at the 1×10^{-4} scale. Use of tolerance limits reliance on unrealistic expectations for model interactions, and for numeric algorithms that internally resolve iterative methods such as a Taylor series, provides a purpose-valid means for terminating a computation that is numerically good enough.

Objects that may mathematically resolve as if they are not in contact measured against the 100 nanometer scales of 1×10^{-7} , such as the classic stacked blocks problem for physics engines, can safely be evaluated as in-contact if the threshold is the thickness of paint (approximately 2×10^{-4} meters). Safely-in-contact positively results in no computed relative downward motion between stationary blocks, and the avoidance of the classic stack of blocks blowing up without artificial dampers or special case handling. Erleben [84] spent significant effort on resolving this stacked blocks problem in his 2004 dissertation *Stable, Robust, and Versatile Multibody Dynamics Animation*. The techniques used in the **Lag Erasure** architecture, including thresholding, fits well as a further constraint supporting Erleben's use of sequenced impulse dynamics in physical simulation, especially for ADES scheduled computations.

CHAPTER 5:

Use-Case Requirements

The following use cases collectively illustrate situations where the additional capabilities of the *Lag Erasure* architecture and algorithms provide either a more effective simulation capability than current traditional distributed simulation designs, or simulation capability that a current traditional distributed simulation design is not able to provide at all. Collectively these use-cases all exhibit significant challenges that are rooted in timing and lag breaking determinism and causality. All current designs capitulate to the problems of distributed state disagreements created by communications lag.

5.1 Perfect Playback

Perfect Playback provides something better than a stepwise event-by-event forwarding through the recorded simulation timeline, because of the continuous-over-time nature of the event stream (see Section 3.1.2.) The position, orientations and status of every entity in the simulation are deterministically available everywhere on the simulation timeline. The recorded simulation timeline also memorializes the action-related events which do not directly affect motion, but trigger updated entity state(s) or provide sensor-related information that an entity may use in the future. Simulation-related infrastructure interactions such as RNG pulls are also memorialized, a full breakdown of RNG-algorithm and entering arguments is not needed, the authoritative value of the pull itself is what is required. If the simulation depends on something, it is reflected in the recorded stream of events.

This allows the simulation playback to be examined using virtual slider controls in the “wiping or scrubbing” mode to target any particular moment on the simulation timeline, and if the browser provides freely positionable viewpoints, a best-view can always be configured for any moment. There is no re-computing interactions while scrubbing across the timeline, because all interactions and results are in the event stream.

Traditional fixed time step simulations do not have this perfect playback capability because it is infeasible to record every entity and every entity interaction in every time step. The *Lag Erasure* architecture’s capability to provide Perfect Playback derives from the choice to

use the longest appropriate event lengths based on computational and decision limits when the event interval is being determined. This dramatically reduces the number of potential-change points which must be archived on execution to exactly the necessary change points that actually evaluated during the simulation.

A deeper capability that emerges from Perfect Playback and the ability to scrub to any moment in the simulation timeline is the potential to seed a new simulation with the `EventList` as provided by Perfect Playback and induce an intentional difference which creates a uniquely new sequence of events, literally an alternate simulation timeline. This is a foundational capability necessary to perform deep *what-if* analyses in a methodical manner⁸. This capability may even allow use of the alternate timelines to directly examine the volatility effects of randomness at certain points in the simulation providing insight into choices that may lead to less volatile results.

5.2 The Digital Twin of Reality: Virtual-Space

There is a concept where technology and reality are so inter-twined and interdependent to society that the concept loses its capitalization as effectively as tissue is universally referred to as kleenex, despite Kleenex being a brand name. This is the case with virtual-space, or vir-space as an easier sound-bite. Vir-space is from the works of David Brin, *Earth* [86] and *Existence* [8], extrapolating that societally-shared AR becomes so pervasive in everyday life that it transcends AR. It just is, everywhere, and everyone wears the necessary vir-glasses. Vir-space is the mashup of multiple selectable information channels, and selectable information layers, all perfectly geolocated and displayed in 3D-space.

Vir-space is the necessary eventual domain of AI entities in which physical reality is digitized into an inherent digital twin that serves as the skeleton for all of the content and data incorporated into vir-space. This version of reality's digital twin allows machine-bound AI constructs to reason and interact with vir-space as the proxy for the physical world, including directly and indirectly influencing human interactions. This implies that the digitization, fidelity, and inter-actability between the virtual twin reality and physical reality is seamless and robust.

⁸How else is Doctor Strange able to properly examine all of the 14,000,605 timelines in *Avengers: Infinity War*? [85].

This seamless, robust, high fidelity version of reality's digital twin is not only what the literary AI constructs operate within. This level of detail is what real world present day AI entities need to be trained within. The current-day commercial model of train ever larger AI large language model (LLM) models and automagically-they-will-become-super-intelligent is a statistical fantasy. The ethereal concept of common sense that is necessary to invoke knowledge in a way that displays wisdom has eluded philosophical construction for the entirety of humankind's existence, yet it emerges ubiquitously, for humans, from interaction with the real world. If no logically constructed ruleset for common sense currently exists within the bounds of mathematics or philosophy, that ruleset cannot be directly programmed into a machine, or inferred by mere textual statistical analysis. The only option remaining is to provide the electronic entity a digital twin world that it can learn within.

A digital twin of reality useful for training a military AI tactical assistant is more complex and dependent on other remote-provisioned capabilities than is computable at a single site. Thus, the environment implementation must be distributed, and lag must be eliminated or the AI will operate with corrupted data.

The effort necessary to provide the breadth and depth of data necessary is vast. Commercial motivations are emerging for doing business in the VR-based metaverse, and emerging market niches are developing for AR, capturing meaningful chunks of data to continue building newer AR enabled capabilities. For example, the AR game *Pokémon Go* by Niantic has collected geospatial player data for years [87], even placing game elements to motivate player engagement in areas of interest for additional mapping. This data collection has resulted in what Niantic calls a "Large Geospatial Model" for an AI navigation system. Combinations of curated commercial data and open source public domain data, such as OpenStreetMap [88], are required to generate the digital twin's breadth and depth of data.

That distributed digital twin of reality employed towards training a military AI tactical assistant must not be corrupted by lag effects if the training is to be deployable without substantial operational risk. *Lag Erasure* techniques are necessary to make this work.

5.2.1 Virtual-Space vs. The Metaverse

meta·verse—a virtual-reality space in which users can interact with a computer-generated environment and other users.—The Oxford English Dictionary. [89]

As the Oxford English Dictionary [89] defines it, and in agreement with common usage, the metaverse is proper VR, not AR, making the metaverse a digital realm virtually navigated by users that are not also navigating in the real world. As an example, consider the difference between the metaverse equivalent of flying a quality flight simulator and flying the actual aircraft in AR with a helmet mounted heads-up display (HUD). This differentiation of metaverse from vir-space limits the usefulness of the metaverse because locomotion around the real world not possible in VR. Mann [90] imagined a way around this by digitizing the real-world environment and projecting it inside the VR scene as part of his extended reality (XR) description, but until sensor and display latency is reduced to sub-perceptual delay timelines this is not a viable option.

Vir-space interactions have real-world limitations that do not affect a non-moving metaverse user. For instance, a user in vir-space would not be able to tap the correct brick behind the Leaky Cauldron to open the entrance to Diagon Alley and walk in [91]. Vir-space may be able to display a AR-style visual overlay view into Diagon Alley, but the user might break their vir-glasses walking into the real wall if they try to enter. While metaverse users can make the analogous inputs, their VR modality either has them motionless or on a specialized device such as an omni-directional treadmill, and the latter situation is rare.

What is presented in 1992 as the metaverse by Stephenson in *Snow Crash* [92] predates the term AR and head mounted display, the same with Brin's 1991 *Earth* [86]. Stephenson's follow-on works remained with the metaverse accessed as VR, but Brin's network-use concept evolved with society's newly emerged mobile device ecosystem and he presented the fully mobile, ubiquitous, interactive shared AR of vir-space in 2013 with *Existence* [8]. Considering the requirements for how shared AR environmental data must be managed from the context of independent users with independent displays (vir-glasses) to maintain a smooth and correct realtime interaction capability is one of the conceptual design filters that the development of **Lag Erasure** concepts and techniques is subject against.

5.2.2 Digital Twins are the Glue Between Virtual and Physical Realities

Lag Erasure is designed to work equally well in both the VR metaverse, and the AR-evolved vir-space. It is reasonable to consider that metaverse avatars may also be visible in vir-space in places where the metaverse depictions match the physical reality extents. This illustrates

the opportunity to construct reality's digital twin enabled for interactions, on a lag-free shared-basis, for human and machine alike using a virtual-to-real continuum allowing safe and realistic training before real world deployments. Any human or equipment-in-the-loop interaction scenario rests on these top-level conceptual design requirements.

Lag Erasure operating with the vir-space environment provides this smooth continuum between training and field, private entertainment to everyday physical interaction. ***Lag Erasure*** enables the necessary distribution of compute resources, and environmental consistency for both civilian and military contexts, and the ability to train an AI with enough world-context that it avoids task over-training brittleness (per Section 1.1.2.1.2).

5.3 Augmented Reality in the Cockpit

The F-35 Lightning II aircraft is the emerging example of AR not as training or a game, but as a intrinsic component of an operational weapons system. AN/AAQ-37 Distributed Aperture System is the formal designation for the RTX Corporation product line described as the *Electro-Optical Distributed Aperture System (EODAS)* [93]. EODAS provides the pilot a functional HUD depiction in the helmet visor and provides full spherical multi-spectral coverage in-visor that allows the pilot to see infrared (IR) imagery anywhere, including through the cockpit floor. The visual information is fused with the other HUD symbology providing gaze-relevant filtered weapons system information to the pilot wherever they look. This definitionally fits the AR definition of a multi-sensory display of virtual information overlaying the physical visual view of the real world.

Once beyond the realization that AR is not an entertainment toy or training-limited technology, what is the result of considering a vir-space digital twin of reality evolution of the F-35 system? Any relevant information can be rendered onto the pilots helmet mounted display (HMD) view, just the same as the EODAS imagery can be presented. Mission flight path plans and low-visibility tanker rendezvous paths can both can render as simple transparent 3D ribbons in the sky similar to automobile HUD navigation systems that are provided in current day 2025.

Adding tactical relevance leads to interesting *what-if?* questions. What-if pre-mission planning does not only identify threat avoidance locations with oversimplified stylized-cylinders or a Tupperware “jello mold” of death? A masters thesis by Tan, *Radar Envelope Visual-*

ization [55] illustrates computation of physically-based radar beam coverage in a 3D virtual scene, an example of the naïve “jello mold” is presented as Figure 5.1, and a physically modeled propagation representation under the effects of atmospheric refraction is presented as Figure 5.2. On the scale of these figures, an aircraft is an infinitesimally small dot, the beam-coverage holidays identifiable in Figure 5.2 are larger than that aircraft-sized dot.

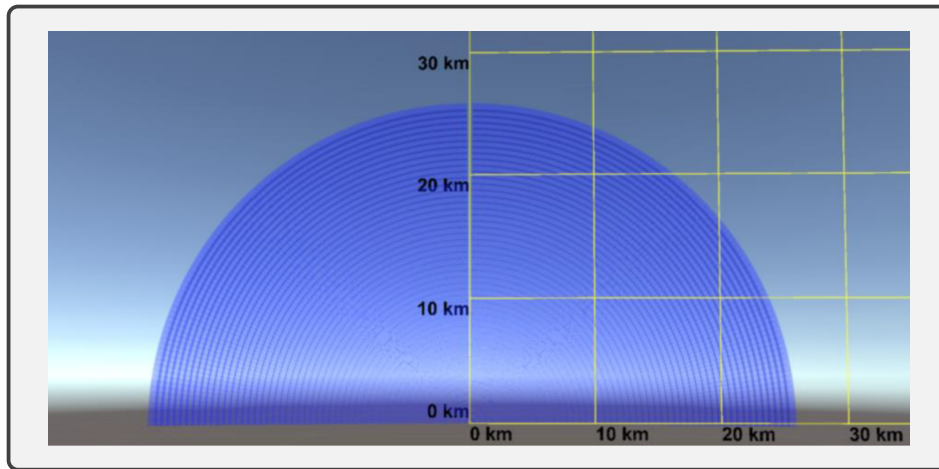


Figure 5.1. Notional target-tracking radar naïve volume (10km Grid Spacing). Source: [55].

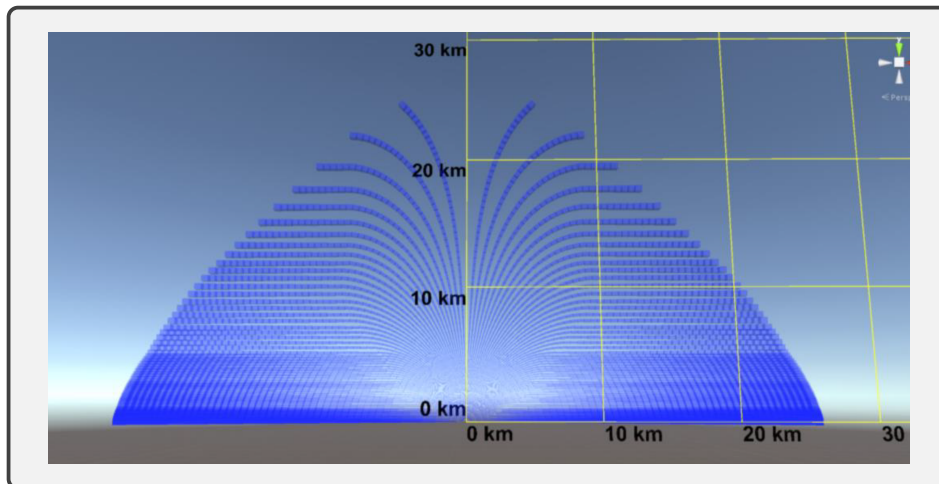


Figure 5.2. Notional target-tracking radar with standard atmospheric refraction (10km Grid Spacing). Source: [55].

This work allows users (pilots) to fully view the expected actual swept-volume of radio frequency (RF) energy based on beam shape and environmental factors, identifying the holidays between individual beam propagation paths. These propagation paths are rendered as transparent 3D shapes, a set of computations that can be completed before flight on a per-platform-of-interest basis using the expected environmental factors for the expected platform locations. These can provide low compute-cost to the aircraft inflight, and high information density to the pilot in use. Inflight RF sensors may be employed to add confirmatory information for whether or not a projected holiday is exploitable. This example of dense inflight information display based on massive offline data is just one of limitless possibilities once systems user interface designers begin thinking in true 3D AR and designing to take full advantage of spatial information instead of curated datapoints.

5.3.1 Live Aircraft vs. Constructive Forces

In an LVC simulated exercise, where L meets V and C can be problematic if the live platform is eventually supposed to have some form of visual contact with the virtually or constructively presented entities. Air combat is a prime problem area for this as virtual and constructed forces can be presented as C2 system contacts, but in the real-world engagement permission is often not providable without visual confirmation. There are decades of beyond visual range (BVR) philosophy vs. visual contact philosophy discussions and BVR engagements don't often receive liberal engagement permission. Rather, BVR engagements are often directed to evolve into visual contact, whether naked-eye or with electro-optical (EO) aided sensors. Virtually or constructively presented forces as currently handled cannot make this important training transition from BVR to visual confirmation, but AR-in-the-cockpit platforms do provide the baseline capability to provide mock visual targets in the pilots HMD, and the *Lag Erasure* system capabilities can make these visual presentations timing-realistic, especially if a "virtual" aircraft is controlled from a manned flight simulator.

5.4 Undersea Warfare (USW) Tactical Collaboration

If an RF "jello mold" can be rendered above ground (per Section 5.3), it can also be rendered underwater. The difference in transmission medium and wavelengths make acoustic propagation more shape-complex, and the relative slow motion of undersea warfare (USW) scenarios compared to jet combat provide an opportunity to leverage fog-of-war depictions

in a manner totally incapable in current weapons system information displays. The tool to evolve for this task is the venerable sand table.

With origins in the scratching of crude strokes with a stick in the sand, military sand tables evolved to abstract theater-level unit positions adjusted by croupier stick wielding operations specialists, and to highly detailed physical models of the location under study. Modern AR technology makes the digital sand table [94] a reality. Sand tables are routinely used as planning and training aides, but not yet in conjunction with massive real-time data from sensor or C2 systems. There are multiple possible implementation examples, here the focus is on one, the USW plotting table of a U.S. Navy (USN) destroyer combat direction center (CDC) in use for a submarine detection and localization exercise.

The Navy is moving on from paper charts on the plotting table to electronic versions such as the computer-aided dead-reckoning tracer, a flat-screened table top display that has the ability to move certain plotted points along a constant speed and course. This is a literal electronic version of the electro-mechanical under-table projection system used since World War 2, no colored pencils needed. While an improvement, it is evolutionary, not revolutionary. This remains a two-dimensional (2D) albeit movable representation, of a real-time 4D time-space evolution problem.

The AR sand table concept provides a full-motion 4D time-space projection of all the relevant tactical data, this includes visual depictions of sound travel paths derived from bathymetric data and the physics equations of underwater sound propagation. These depictions show both search space volumes without contact as well as highlighting any potential or confirmed contacts. The additional capability this AR sand table presents is the explicit and intuitively obvious visual representation of un-searched search-space volumes, and the time-based decay of searched volumes based on underwater target sound behaviors related to speed-of-advance. The un-searched volumes visual representation constitutes a real-time depiction of the fog-of-war during a search. The underwater sound propagation paths are complex and dependent on multiple independent factors making the fog-of-war a combination of tortured shapes and holidays that a small number of humans can mentally construct and track in their heads.

The visual depictions can be leveraged to plan how to eliminate holidays in coverage and manage the fog-of-war progress in un-sweeping the search area. What-if scenarios can be

constructed and played through to determine the relative utility of multiple potential next-steps, with replay-ability and adjustability. As the tactical depiction is just another AR scene, it can be shared with other participating platforms and integrate the searched space volume and fog-of-war status of the integrated team of participants. What ifs are seamlessly shared in collaborative multi-ship tactical discussions and reaching in to point or adjust the display is managed with the ***Lag Erasure*** techniques, making these interjections flow as naturally as they do during in-person communication. This collaborative approach leverages the many-heads-are-better-than-one principle and is superior to the current separate thoughts on separate-platforms with separate static views of the data, and sporadic communication between platforms.

5.5 Augmented Reality for Everyday Vehicle Safety

This Section (5.5, including Subsections 5.5.1 and 5.5.2) is excerpted from the author's article published in the January 2020 issue of IEEE Computer: *The Promise of Interactive Shared Augmented Reality* [95].

5.5.1 Motivating Example: The Difficult-to-See Bicycle [95]

Bicyclists often share the roadway with motor vehicles. Figure 5.3 depicts the unsolved problem for both human drivers and individual autonomous vehicles: the element of surprise in hazardous situations such as a bicycle entering an intersection from the side while the bicycle is blocked from view by an obstacle. It is easy to imagine a truck or bus as obstacles, but a fence, building, group of pedestrians, or tree, can also provide the same visual barrier. When the bicyclist enters the intersection from behind the obstacle there may be a critically short time for the bicyclist, motorist and/or self-driving automobile to react to avoid a collision. The uncontrolled placement of real-world visual obstacles hides critical information from multiple participants at the intersection.

Whether the driver of vehicle A is human or an autonomous control system, the time to sense the environment of the intersection, plus the time to make a decision and finally actuate the accelerator, brakes and/or steering (i.e., reaction time), may be longer than that needed to avoid a mishap. We believe that augmenting the automobile operator's individual

reality into a collective shared reality can dramatically reduce the occurrences of surprises, and in our scenario potential mishaps.

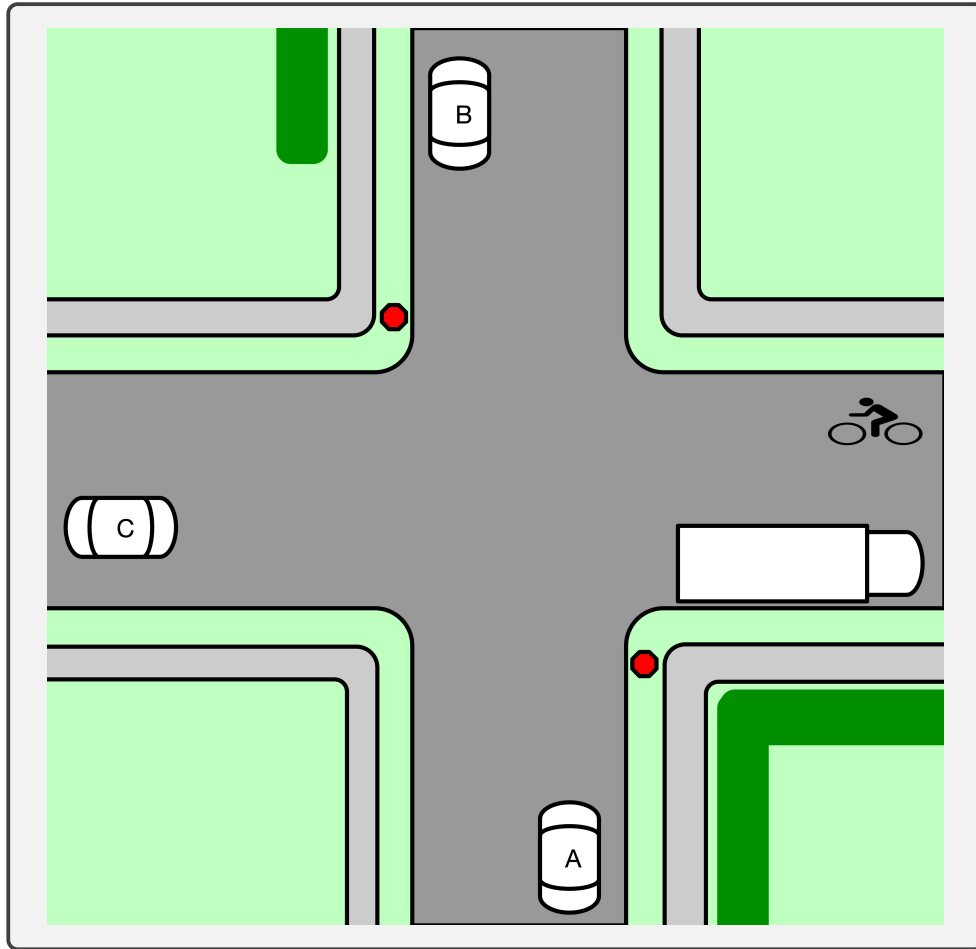


Figure 5.3. The crossing bicycle is hidden from vehicle A, and visible to vehicles B and C. Source: [95].

5.5.2 Shared Reality [95]

What is this shared reality? It is a form of AR where the augmenting information comes from a VR which itself is a distributed simulation. The base environment from a data/map system is dynamically combined with information obtained from tracking the behaviors of real-world objects via the participants' sensor arrays. Each participating vehicle shares its view of the world with other local vehicles, and this is merged into a common and consistent shared representation of the local world. We consider the result a form of shared reality.

Within an individual vehicle, the operator has access to both the information from on-board sensors, and additional information which may only be available via the network sharing. It is an accurate and consistent creation of this shared reality that is precluded by the network latency problems in current distributed simulations.

Back in our hidden bicycle scenario, the shared reality is built by communications between vehicles on the road, or even fixed sensors that may be part of a smart intersection management system. While Figure 5.3's vehicle A has a significant portion of its sensor field masked by the truck and tall hedges, other sensor-equipped autonomous vehicles (B and C) have clear line-of-sight and can not only detect the presence the non-networked bicycle and rider, but also accurately determine its relative position and velocity. When the bicycle's track information is passed along to other vehicles the receiving vehicles' decision systems cross-check and integrate this information into their local simulation. This is possible because fixed information from the map/data system is fused with sensor readings of fixed local landmarks such as traffic signal support posts and the parked truck. The common fixed-location information allows each vehicle in the network to determine high-confidence absolute position and track information for themselves and integrate the relative track and position of the non-networked bicycle. Now the vehicle's decision making systems each have a shared reality larger and/or more precise than what is possible from own-platform systems alone.

This shared reality is projected in each automobile to augment the platform operators' AR scenes. If the driver of vehicle A is human, an AR bicycle warning graphic can be projected to represent the bicycle and rider hidden behind the truck, and it can be moved so that it follows along the same bearing angles as the actual bicycle. If vehicle A is operated as an autonomous system, what-ifs/decision processes for determining how to safely approach and navigate the intersection can progress considering the bicycle's track, even without prior own-vehicle sensor confirmation. Either way, more relevant information presented to the operator generally enables globally better decisions, especially when the additional information unmask otherwise hidden threats.

5.6 Online Multiplayer Action Games

The first important takeaway for the online gaming community, both players and developers, is that a game is a simulation. *Everything* covered to this point of the dissertation applies. Pick a genre: multiplayer online battle arena (MOBA), massively multiplayer online game (MMOG), head-to head player vs. player (PvP), if the players are not geographically close enough that latency is less than a single frame delay, the game is not just another simulation, it is a distributed simulation.

The examples of LVC simulations, especially the sequence of Boids examples in Section 3.2.1 illustrate the same problems that every multiplayer networked action game experiences. The background coverage in Section 2.1.7 indicates 2003's lag compensation [9] is still the state of the art technique (see *Valorant* [14]) for dealing with lag in the game development community. The problems and asymmetric advantages that lag creates for the player base cannot be eliminated until the lag is erased from the game timeline. The *Lag Erasure* architecture is the only available methodology to eliminate the effects of lag.

5.6.1 Anti-Cheat Considerations

The breadth of potential cheats in a distributed game are broad, both on-host and off-host. The *Lag Erasure* system provides a ground-truth across the entire distributed system in the form of the consistent distributed state that no other distributed simulation methodology can leverage. Every previous attempt to identify the manipulating of game state with on-host cheats is hampered by the inevitability that there is no ground truth, there are different states at every participating host, and this inherently makes it difficult to determine whether the state differences are simply normal lag-created differences, or reflect a state corrupted by tampering.

The divergent states limit the existing anti-cheat technique of referees to employing statistical methods of cross-host state-difference examination and message success examination [96]. The lack of ground-truth drives reliance on reputation-based distributed methods [97] as part of the attempt to detect malicious state adjustment. None of these approaches have yielded widespread success and adoption.

Current state-of-the-art anti-cheat revolves around kernel-level operating system integrations to in an attempt to identify and monitor certain hardware-level suspect behaviors

including memory and bus manipulations, for example: Riot Games *Vanguard* [98] and Valve Steam's *Valve-Anti-Cheat (VAC) System* [99]. There are multiple other similar packages across the video gaming industry. By-definition, any software that installs operating system kernel-level modules carries the extra burden of maintaining security to preventing exploitation and privacy violations, an additional responsibility and expense that is not core to the games themselves, but is assessed as worth the resourcing and risk to provide a fairer playfield. Removing that security risk by examining only application-layer ground-truth data has long-term benefit.

5.6.2 Security Through Ubiquity

The ***Lag Erasure*** architecture's provision of consistent distributed state across the entire distributed system provides local application-layer automatic state comparison as an inherent component of maintaining the states for conducting lag erasure—no central referee server injecting additional delays into message deliveries, no kernel-level memory surveillance, no additional processing layers on top of the simulation engine operation. ***Lag Erasure*** already enforces distributed system wide state equivalence.

When ***Lag Erasure*** detects a state deviation that cannot be properly healed by the lag erasure portion of the system there are exactly two possibilities: there is a bug, or there is malicious manipulation of the state. State examination and comparison needed for executing ***Lag Erasure*** is entirely local once the state update message arrives, there is no consensus concept involved. This prevents teams of malicious hosts from claiming the definitive state, and asserting correct-state hosts are incorrect as is possible with reputation-based consistency methods. When high trustability or higher-authority dispute resolution is required, a passive referee server on trusted hardware running the same equivalent code as the participants can be employed to compute the simulation in designated Regions. All participants state update messages must resolve consistently on the referee the same as required on the other simulation hosts. No in-simulation possibility for the referee to be hijacked via consensus manipulation because as already stated, all state comparison and resolution is local when healing the temporary input-device generated state divergences.

Additionally, with a consistent shared state, designers have the necessary consistent data for generating statistical models identifying specific malicious player actions, such as de-

termining if *wallhacks* to see-through-the-walls are being employed. This form of hack can be executed both by maliciously examining game hardware memory, or by external video processing and recombination outside the purview of any in-host detection methods. Because every event and input are the same across the network and thus available to examine, modeling view and aiming reaction rate behavior differences between valid corner-camping and see-through-walls aiming and reaction rates is statistically viable, and independent from where the malicious actor gets the information they are reacting to.

This *Security Through Ubiquity* concept is new and untested, and constitutes future work with the ***Lag Erasure*** system. The concept is a profound change in how application behavior security can be achieved in distributed game clients.

5.7 Emergent System-of-Systems Analysis

A common thread across the Sections above is that the state consistency maintenance and communication of the ***Lag Erasure*** system is an emergent foundation for addressing multiple challenges that the System-of-Systems Engineering community has engaged with in recent decades. A consistent shared state across a networked system of actors provides a shared ground-truth that individual actors can leverage to make local decisions that are globally valid, such as in the collision-avoidance and warning in the automotive scenario.

The emergent result of locally determined and globally valid decision process is a form of global cooperation that does not require intensive inter-actor coordination. This same local-global decision-making relationship exists in the bicycle scenario (Section 5.5.1), anti-cheat (Section 5.6.1) and security through ubiquity (Section 5.6.2) scenarios which all match this same system-of-systems template but are not treated as such to date because previous to the ***Lag Erasure*** system, there is no form of consistent global data to access.

The consistent shared state of a ***Lag Erasure***-based simulation may further leverage the system-of-systems template into the ubiquitous daily shared interactions of tradesman raising a building in a chaotic worksite with blindspots even more common than the bicycle scenario (Section 5.5.1), or making route-guidance navigation applications organically integrated into supporting the driving, walking, or cycling tasks rather than the series of short distractions that they currently are. Consistent shared data supporting dynamic tasks across a wide network of actors has not existed previous to the ***Lag Erasure*** simulation archi-

ture, the shared realtime-tasks it can support are not limited by those presented in this Chapter, they are just stepping-off points for multiple communities that can leverage large scale distributed simulations to simplify and enable continuing work and new capabilities.

5.8 Realizing These Use Cases

The academic community answer is to reference Chapter 6 *Future and Enabled Work*. A program management relevant answer is that this dissertation is documenting the theoretical grounding for adding a new and previously impossible capability to distributed simulations, and it is supported by an in-progress application build towards a minimum viable product demonstration, which as of publishing validates the implementation of the speculative evaluation component of ***Lag Erasure***.

Completion of a minimum viable demonstration requires:

1. A minimal network connection protocol.
2. The event archiving portion of the *Perfect Playback* capability, per Section 5.1.
3. Post processing the event archive to assess the lag measuring metric presented in Section 2.1.3.

Accomplishment of this effort with verification that lag is successfully erased across the entire distributed simulation graduates ***Lag Erasure*** from being a high risk/high potential payoff project into a medium or lower risk category, and remaining as potentially high payoff. At that stage the risk no longer resides in “if it works,” the risk transitions to the capability of a program to fund and await delivery of an adequately provisioned physics-based simulation engine implementation that employs the ***Lag Erasure*** architecture. This stage is a significant investment in future capability unavailable in any other existing simulation ecosystem. The question of the simulation visualization component is not directly considered here as the ***Lag Erasure*** simulation modeling layer is cleanly separated from any visualization component, but additional integration work is required for any existing simulation visualizer specified. A working simulation engine alone is not enough, also required is an initial set of demonstration simulation scenarios which employ ***Lag Erasure*** for human participants and validate the participants are not perceiving lag effects.

With human participant validation and a running simulation engine the efforts transition to low-risk extensions of the simulation engine capabilities to support more elaborate training simulations. The majority of the remaining risk is in adoption-resistance revolving around any not-invented-here, or not-in-my-walled-garden viewpoints. Early acknowledgement of the potential resistance may include mitigation's involving the leveraging the new ***Lag Erasure*** capacity as an opportunity to simultaneously widen the supply base for simulation models that work under the ***Lag Erasure*** architecture, while financially motivating efforts for vendors to deliver and maintain simulation models rather than create and maintain walled-garden full-simulation solutions.

The capability for vendors to provide alternate implementations of a ***Lag Erasure*** simulation remains, the key issue to avoid walled-gardens is to ensure that the property of functionally equivalent code covered in Section 3.1.5.1 is maintained. In this context functionally equivalent code means the same models test to the same tolerances, there is wide latitude in technical implementations which can meet this requirement. The primary concern to guard against is for any implementation to unreasonably attempt to gate-keep which models are allowed to run, artificially constraining the functional equivalence threshold. This constitutes a stealthy attempt to reestablish a walled-garden full-simulation solution instead of leveraging the wider simulation model supply base mentioned in the above paragraph.

CHAPTER 6:

Future and Enabled Work

The future research directions of the *Lag Erasure* architecture fall into the direct future work of completing a functional demonstration system that enables testability of the shared distributed state claims. Enabled work begins with the assumption that the construction of a shared distributed state claim is possible and then describes new avenues of research that are opened for examination.

6.1 Future *Lag Erasure* Development and Testing

Initially, the multi-host capable version of the *Lag Erasure* system must be completed so that the claimed creation of a consistent distributed state can be rigorously tested. This breaks down into two main efforts.

- First, event stream archiving must be implemented in a manner supporting the rework functionality for accomplishing lag erasure on late arriving input-device events. This requires identifying when an event is no longer subject to potential rework and thus not required to be resident in main memory for efficient access during potential rework. Archivable events are placed in secondary storage for post-simulation reconstruction and replay.
- Second, the current single-machine multi-threading environment requires communications support for multi-host network operations, and this requires the addition of a minimum viable reliable ordered message networking protocol, plus associated message creation and receipt functionalities.

6.1.1 Post-Processing of Archived Event Stream Data

The addition of the archive enables the effort to treat the archived event stream as the source for Perfect Playback (see Section 5.1). The Perfect Playback capability is a cornerstone for full-scene quality assurance efforts and assistance with creating comprehensive post-simulation after-action reporting.

With the Perfect Playback capable event stream, the streams on each host in the simulation can be precisely compared with respect to the simulation's definitive virtual timeline. For Regions that are computed on multiple hosts, the states of each entity are comparable, and if *Lag Erasure* is working as designed should match for all archived events within the defined engineering tolerances as mentioned in Section 4.5.2.

Additional post-processing and summations of these measured deltas from the archived event streams, per the description in Section 2.1.3, constitute the remainder of the process generating the measurement values needed for construction of the overall percentage of the simulation runtime that any entity-exhibited state discrepancies between the distributed hosts of the simulation. These summed discrepancies constitute the total time that an entity is in a lagged state during the simulation execution.

6.1.2 *Lag Erasure* Capability Expansion

Additional related work is necessary to broaden the *Lag Erasure* system from a minimum viable proof-of-concept implementation. Once the verification of the capability to erase lag from a simple distributed simulation is made, production level breadth of physics coverage must be implemented and full incorporation of non-motion actions that affect simulation state. This also requires positive management of pseudo-random number generation series to support the necessary multi-host rework capabilities. This may range between a brute-force approach such as managing a seeded pseudo-random number generator per entity and tracking pull sequence numbers to allow an inefficient but correct regeneration of the pseudo-random number sequence; through tracking pull-sequences from a reversible pseudo-random number generator.

With “auditable” randomness and a wider physics capability the baseline for usefulness across the spectrum of simulation communities is met.

6.2 Enabled Research and Implementation Work

6.2.1 Sensor Visualization: Augmented Reality in the Cockpit

Individual enabling capabilities for the AR in the cockpit case study in Section 5.3 are enabled with the sensor-swept volume display, the topic of a currently-in-work masters thesis.

The goal is an offline-computed expected propagation volume which is implemented as a real-time queryable data structure. The thesis is unclassified proof-of concept work using a parameterized notional antenna and basic standard day atmospheric. The physically-derived path of individual beam packets is approximated and represented by a series of Capsules, which are then wrapped by polar-AABB geometry that is part of an R*-Tree collision detection structure. A copy of the pre-computed R*-Tree can be geolocated in a simulated environment and trivially answer localization queries on whether an entity is inside the beam-propagation envelope. Because the sensor R*-Tree data is geometrically represented, the choice of whether to render the sensor volume in-scene or not is a dynamically adjustable runtime-decision.

6.2.2 Sensor Visualization: The Undersea Warfare Sand Table

Following the sensor approximations proof of concept, the implementations are readily adaptable to AR-based C2 display research and potential applications such as the Shared Tactical Collaboration topic of Section 5.4. There is additional work for the graphical depictions to treat the pre-computed R*-Tree geometry as “leaving behind” a fading, ever creeping, fog-of-war visualization to help tactical decision makers understand the intricacies of the real time 3D undersea warfare domain. The fog-of-war depiction is not only a visual aid to human operators, because it is geometrically derived and rate updated, it provides a human-understandable data input to an AI tactical assistant for training and tactical applications.

6.2.3 Online Gaming Distributed Simulation Security Enhancements

The *Security Through Ubiquity* concept introduced in Section 5.6.2 is new and entirely dependent on the capability of a distributed shared state across the entire set of participating hosts. There is no possible equivalent security method under current simulation construction paradigms. The shortfalls in usable data for previous state-surveillance methods caused by close but inconsistent state between hosts are removed. State comparisons on event streams are no longer subject to statistical evaluation to determine whether they are “manipulated enough” to be flagged. It is believed that this repeatable and reconstructable state data allows security-related interaction models to be developed and implemented that may be capable of identifying exploit behaviors that do not manipulate the state data that is transmitted over

the network, such as the general class of wallhacks. It is likely that security through ubiquity is only scratching the surface of what is possible in the space of securing online gaming distributed state.

6.2.4 Reliable *What-if?* Scenario Exploration

The record-ability and replay-ability of Perfect Playback as introduced in Section 5.1 are foundational to methodically exploring larger state spaces in a distributed simulation. Locally computed simulations have access to the Latin Hypercubes methodology, as described by Hernandez [100], to efficiently determine which portions of the state space effectively contribute to generating a usable answer. Inter-host state differences randomly injected via laggy network communications breaks the definitive state explorations desired when using the Latin Hypercubes methodology to partition the state space because there is a lag-created loss of deterministic control over that state space progression.

The ***Lag Erasure*** architecture removes the lag effects and non-determinism. This means methodologies for examining what-if questions may be able to employ Latin Hypercubes at multiple decision-points over the simulation timeline. Investigations into selecting the optimal choices and paths to examine are both enabled and necessary to generate the what-if insights within an efficient timeline. This time-series extension of Latin Hypercubes may be a central enabling technique for the training and employment of future AI tactical assistants.

AI tactical assistant simulation is not only about training. Situations are inevitable in which the AI is not already adequately trained for the presented task. The AI must not fail at this point, it must have the capability to generate and execute its own efficient what-if simulations to arrive at a reasonable answer in rapid fashion. This modality follows the general flow of Klein's *Recognition Primed Decision Making (RPD)* [101], a model which is:

[A] recognitional model of decision making that shows how people can use experience to avoid some of the limitations of analytical strategies. This model explains how people can make decisions without having to compare options. It fuses two processes—situation assessment and mental simulation—and asserts that people use situation assessment to generate a plausible course of action and use mental simulation to evaluate that course of action. [101]

This process cannot guarantee fully explainable AI decisions, but it may enable AI decisions that are as explainable as those generated by human experts. This is the level of decision trust currently employed within military chain-of-command task delegation for trained leaders, both Commissioned and Non-Commissioned Officers. Trust is attained via performance and evaluation through multiple standard training and qualification pipelines.

The results of an AI-assistants internal simulations are cross-checkable against both the external what-if scenario simulations, and real-world conditions, plus how well those agree or disagree. This process is analogous to the commander asking the unit leader in the field for conditions and their plan. Then, based on training, reputation of the field leader and the commanders experience, the commander makes their decisions for when to retain close control or delegate authority and transfer responsibility. No commander ever definitively knows every detail of the internal mental state supporting exactly how and why a human field leader makes a specific decision, but the process of repeated training and building trust is successful despite the lack of absolute verification capability.

Testability and repeatability gained from the availability of a consistent shared state among the networked participants provides the capability to understand the inputs to any tactical AI under-training, and by extension determine whether the eventual decisions are within doctrine and expectations. Deriving this level of AI trustability is a definite need in the emerging AI decision space and one that is currently not possible for large-scale distribution-required scenarios, indicating a fertile research space supporting AI over the long term.

6.2.5 Virtual-Space Applications

In Section 5.2 there are multiple use-cases presented for *Lag Erasure* enabled shared environments. Each of these constitutes newly enabled modes of sharing virtual information that is geo-registered with a physical reality. The breadth of research remaining to effectively create the virtual digital twin of reality that does not have lag-induced consistency failure modes is substantial.

This is not a purely aspirational entertainment toy. This digital twin of reality is necessary in the eventual training of AI-entities that are not brittle to random training gaps. This makes the vir-space realm not only shared between human and equipment-in-the-loop devices, but also shared with the eventual AI constructs. Exploring the ramifications of the digital twin

as a stand-alone VR as a means to provide safeties in the decision making of AI-entities before physical-reality is manipulated via hardware-connections is a necessary step before AI-entities are provided free access to the digital twin vir-space shared with humanity. The digital twin framework and research involving AI and *Recognition Primed Decision Making* mentioned in Section 6.2.4 may provide researchers and AI training personnel a level of explainability that approaches the level exhibited in regular conversation with human experts. This flavor of explainability is impossible in a classic vector-machine neural network that only holds outputs as coefficient vectors mapping to a specific “answer” based on some vectorized set of inputs. People do not maintain an internal monologue in terms of activation potentials in abstract vectors of neural columns. Extracting useful human meaning from a system that can only trace and communicate numeric vectors within its trained vector state machine is likely not possible.

The digital twin training may help the AI-trainee build abstractions of the world in a manner that is understandable and may be communicated to humans, in the same manner that humans make and communicate abstractions despite not having access to the vector coefficients of electrochemical potentials that generate those communicable abstractions. We trust other humans because they perform as expected in training exercises and can communicate in abstractions acceptable to the task’s trainers and leaders.

Following eons of evolution resulting in this performative means of generating trust it is difficult to imagine that humanity can successfully and without dramatic failure, invent a different mode of building trust on-the-fly. Therefore building that viable virtual digital twin of reality provides a bridge to using virtual equivalents of the real-world trust-building method humanity has evolved over its existence. The necessary virtual digital twin of reality is too large a set of computational models to execute locally. It requires the ability to be computed correctly in a distributed manner, and hence requires the capabilities provided and extended from the *Lag Erasure* architecture.

This is only the beginning. Yes that sentence is cliché, and remains accurate. The phenomena of lag is holding back innumerable applications of simulation technology, from the already tried and perpetually unsuccessful to the why-even-bother realm of ideas that

remain fantastical because nobody has figured out how to make the first group work. Lag, whether identified as such or not, is the reason because lag makes virtual social and shared interactions worse. Full stop.

Enabling shared-interaction to occur in augmented and virtual environments without the artificial and awkward interactions forced by lag removes a previously insurmountable barrier to designers and software artisans. Allowing interactions and scenarios to be designed free of the lag-created barrier enables far more than the above research and development ideas can communicate.

Do not forget the future metal and plastic inhabitants of humanity's shared spaces, integrating autonomous robots of any form into the social fabric is a thing the metal and plastic inhabitants have not received a billion years of evolution to prepare for. Humans do not know enough about how the brain works internally to replicate it or its behavior in produced entities. Virtual socio-physical environments where training and education can occur safely before deployment and entering society in the real world is required, not optional. The breadth of required experience does not fit into a single box or even a local cluster, and the necessary distribution requires ***Lag Erasure***.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

- [1] S. Singhal and M. Zyda, *Networked Virtual Environments*. New York, NY, USA: Addison-Wesley Professional, 1999.
- [2] A. Steed and M. F. Oliveira, *Networked Graphics*. Burlington, MA, USA: Elsevier, 2010.
- [3] CAE Defense and Security, “Live-Virtual-Constructive (LVC) Training,” CAE Defense and Security, 2026. Accessed Jan. 23, 2026. Available: <https://www.cae.com/defense-security/what-we-do/training-systems-integration/live-virtual-constructive-lvc-training-1/>
- [4] S. J. Taylor, O. Balci, W. Cai, M. L. Loper, D. M. Nicol, and G. Riley, “Grand Challenges for Modeling and Simulation: Expanding Our Horizons,” in *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, Montréal, Québec, CA, 2013, pp. 403—408.
- [5] S. J. E. Taylor *et al.*, “Grand Challenges for Modeling and Simulation: Simulation Everywhere—From Cyber-infrastructure to Clouds to Citizens,” *SIMULATION*, vol. 91, no. 7, pp. 648–665, 2015.
- [6] D. Brutzman, “A Virtual World for an Autonomous Underwater Vehicle,” Ph.D. dissertation, Department of Computer Science, Naval Postgraduate School, Monterey, CA, 1994.
- [7] B. Mirtich, “Timewarp Rigid Body Simulation,” in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, New Orleans, LA, USA, 2000, pp. 193–200.
- [8] D. Brin, *Existence*, 1st ed. New York, NY, USA: Tor Science Fiction, 2013.
- [9] Y. W. Bernier, “Server In-game Protocol Design and Optimization,” Valve Developer Community, blog, Valve Software, 2001. Available: https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization
- [10] S. Zhou, W. Cai, B.-S. Lee, and S. J. Turner, “Time-Space Consistency in Large-Scale Distributed Virtual Environments,” *ACM Transactions on Modeling and Computer Simulation*, vol. 14, no. 1, pp. 31–47, 2004.

- [11] Y. Park, J. Kim, S. Kang, H. Ko, and Yonchul, “Dynamic Shared States Management For Distributed Interactive Virtual Environment,” in *14th International Conference on Artificial Reality and Telexistence*, 2004. Available: https://icat.vrsj.org/papers/2004/Workshop_3/W3-4.pdf
- [12] D. Delaney, T. Ward, and S. McLoone, “On Consistency and Network Latency in Distributed Interactive Applications: A Survey—Part I,” *Presence: Teleoperators and Virtual Environments*, vol. 15, no. 2, pp. 218–234, 2006.
- [13] J. S. Steinman, “Interactive SPEEDES,” in *[1991] Proceedings of the 24th Annual Simulation Symposium, New Orleans, LA*, Los Alamitos, CA, USA, 1991, pp. 149–158.
- [14] M. deWet, “Peeking into VALORANT’s Netcode,” Riot Games, blog, Riot Games, 7 2020. Available: <https://technology.riotgames.com/news/peeking-valorants-netcode>
- [15] S.-J. Kim, F. Kuester, and K. H. K. Kim, “A Global Timestamp-Based Approach to Enhanced Data Consistency and Fairness in Collaborative Virtual Environments,” *Multimedia Systems*, vol. 10, no. 3, pp. 220–229, 2005.
- [16] D. Roberts and P. Sharkey, “Maximising Concurrency and Scalability in a Consistent, Causal, Distributed Virtual Reality System, Whilst Minimising the Effect of Network Delays,” in *Proceedings of IEEE 6th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Cambridge, MA, USA, 1997, pp. 161–166.
- [17] D. Roberts and R. Wolff, “Controlling Consistency within Collaborative Virtual Environments,” in *Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications*, Budapest, HU, 2004, pp. 46–52.
- [18] O. Otto, D. Roberts, and R. Wolff, “A Review on Effective Closely-Coupled Collaboration Using Immersive CVE’s,” in *VRCIA ’06: Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and its Applications*, Hong Kong, CN, 2006, pp. 145–154.
- [19] V. K. Garg, *Elements of Distributed Computing*, 1st ed. New York, NY, USA: John Wiley & Sons, 2002.
- [20] K. P. Birman, *Reliable Distributed Systems*, 1st ed. New York, NY, USA: Springer, 2005.
- [21] G. Mühl, L. Fiege, and P. Pietzuch, *Distributed Event-Based Systems*, 1st ed. Heidelberg, DE: Springer, 2006.

- [22] G. Tel, *Introduction to Distributed Algorithms*, 2nd ed. Cambridge, UK: Cambridge University Press, 2000.
- [23] B. Liskov, M. Castro, L. Shrira, and A. Adya, “Providing Persistent Objects in Distributed Systems,” in *Proceedings of the 13th European Conference on Object-Oriented Programming*, Berlin, DE, 1999, pp. 230—257.
- [24] C.-H. Moh and B. Liskov, “TimeLine: A High Performance Archive for a Distributed Object Store,” in *NSDI’04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, San Francisco, CA, USA, 2004, p. 26.
- [25] M. Castro and B. Liskov, “Practical Byzantine Fault Tolerance,” in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, New Orleans, LA, USA, 1999, pp. 173—186.
- [26] B. Wester, J. Cowling, E. B. Nightingale, P. M. Chen, J. Flinn, and B. Liskov, “Tolerating Latency in Replicated State Machines Through Client Speculation,” in *NSDI’09: Proceedings of the Sixth Conference on Symposium on Networked Systems Design and Implementation*, 2009. Available: https://www.usenix.org/legacy/events/nsdi09/tech/full_papers/wester/wester_html/index.html
- [27] M. Mauve, “Distributed Interactive Media,” Ph.D. dissertation, Department of Computer Science, der Universität Mannheim, Mannheim, DE, 2003.
- [28] D. Delaney, T. Ward, and S. McLoone, “On Consistency and Network Latency in Distributed Interactive Applications: A Survey—Part II,” *Presence: Teleoperators and Virtual Environments*, vol. 15, no. 4, pp. 465–482, 2006.
- [29] O. Obleukhov and A. Byagowi, “How Precision Time Protocol is being deployed at Meta,” Engineering at Meta, blog, 11 2022. Available: <https://engineering.fb.com/2022/11/21/production-engineering/precision-time-protocol-at-meta/>
- [30] R. M. Fujimoto, *Parallel and Distributed Simulation* (Wiley Series on Parallel and Distributed Computing). New York, NY, USA: John Wiley & Sons, Inc, 2000.
- [31] D. R. Jefferson, “Virtual Time,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 7, no. 3, pp. 404–425, 1985.
- [32] L. Lamport, “Time, Clocks, and the Ordering of Events in a Distributed System,” *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.
- [33] *Network Time Protocol Version 4: Protocol and Algorithms Specification*, RFC 5905, 2010.

- [34] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Std 1588-2002, 2002.
- [35] R. M. Fujimoto and R. M. Weatherly, “Time Management in the DoD High Level Architecture,” *ACM SIGSIM Simulation Digest*, vol. 26, no. 1, pp. 60–67, 1996.
- [36] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*. Boston, MA, USA: McGraw-Hill Education, 4 2000.
- [37] A. H. Buss, “Discrete Event Simulation (DES) Modeling,” Course Notes, 2025, Naval Postgraduate School, Monterey, CA, USA, 2025. Available: <https://github.com/ahbuss/DESMModeling/blob/master/Discrete%20Event%20Simulation%20Modeling.pdf>
- [38] D. H. Eberly, *3D Game Engine Design*, 1st ed. San Francisco, CA: Morgan Kaufmann, 2007.
- [39] I. Millington, *Game Physics Engine Development*. Burlington, MA, USA: Morgan Kaufmann, 2010.
- [40] P. J. Denning, “The Locality Principle,” *Communications of the ACM*, vol. 48, no. 7, pp. 19–24, 2005.
- [41] R. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed. New York, NY, USA: Dover Publications, Inc., 1973.
- [42] Vectorku Studio, “batter-ball,” Vecteezy, 2023. Accessed June 23, 2025. Available: <https://www.vecteezy.com/vector-art/29218621-high-details-of-baseball-player-silhouette-minimal-symbol-and-logo-of-sport-fit-for-element-design-background-banner-backdrop-cover-vector-eps-10>
- [43] D. Adler, “7 Stars Entering ’25 with Bat Speed on the Rise,” *Major League Baseball*, Feb 2025. Available: <https://www.mlb.com/news/mlb-stars-who-added-bat-speed-in-2024>
- [44] *Official Baseball Rules, 2025 Edition*, Office of the Commissioner of Baseball, New York, NY, USA, 2002. Available: <https://mktg.mlbstatic.com/mlb/official-information/2025-official-baseball-rules.pdf>
- [45] *Continuous Collision Detection (CCD)*, Unity Manual, 2023. Available: <https://docs.unity3d.com/2023.1/Documentation/Manual/ContinuousCollisionDetection.html>
- [46] E. Kofman, “Discrete Event Simulation of Hybrid Systems,” *SIAM Journal on Scientific Computing*, vol. 25, no. 5, pp. 1771–1797, 2004.

- [47] W. H. Press, S. A. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. New York, NY, USA: Cambridge University Press, 2007.
- [48] L. Schruben, “Simulation Modeling With Event Graphs,” *Communications of the ACM*, vol. 26, no. 11, pp. 957–963, 1983.
- [49] R. G. Sargent, “Event Graph Modeling for Simulation with an Application to Flexible Manufacturing Systems,” *Management Science*, vol. 34, no. 10, pp. 1231–1251, 1988.
- [50] T. K. Som and R. G. Sargent, “A Formal Development of Event Graphs as an Aid to Structured and Efficient Simulation Programs,” *ORSA Journal on Computing*, vol. 1, no. 2, pp. 107–125, 1989.
- [51] A. H. Buss, “Modeling With Event Graphs,” in *Proceedings of the 28th Conference on Winter Simulation*, Coronado, CA, USA, 1996, p. 153–160.
- [52] C. Böhm and G. Jacopini, “Flow Diagrams, Turing Machines and Languages With Only Two Formation Rules,” *Communications of the ACM*, vol. 9, no. 5, pp. 366–371, 1966.
- [53] E. L. Savage, L. W. Schruben, and E. Yücesan, “On the Generality of Event-Graph Models,” *INFORMS J. on Computing*, vol. 17, no. 1, pp. 3–9, 2005.
- [54] D. Kozen and W.-L. D. Tseng, “The Böhm–Jacopini Theorem Is False, Propositionally,” in *Mathematics of Program Construction*, Berlin, DE, 2008, pp. 177–192.
- [55] D. A. Tan, “Radar Envelope Visualization,” M.S. thesis, Naval Postgraduate School, Monterey, CA, 2018.
- [56] Y. P. Cheng, “Hybrid High Fidelity Modeling of Radar Scenarios Using Atemporal, Discrete Event, and Time Step Simulation,” Ph.D. dissertation, Modeling, Virtual Environments, and Simulation Institute, Naval Postgraduate School, Monterey, CA, 2016.
- [57] K. Erleben, “Velocity-Based Shock Propagation for Multibody Dynamics Animation,” *ACM Transactions on Graphics*, vol. 26, no. 2, p. 12, 2007.
- [58] J. Koenig, I. Karamouzas, and S. J. Guy, “Object-Centric Parallel Rigid Body Simulation With Timewarp,” in *Proceedings of Motion on Games*, Dublin, IE, 2013, pp. 203–212.

- [59] J. Nutaro, “Parallel Discrete Event Simulation with Application to Continuous Systems,” Ph.D. dissertation, Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ, 2003.
- [60] L. Numeroff, *If You Give a Pig a Pancake*, 1st ed. New York, NY, USA: Harper Collins, 1998.
- [61] R. M. Fujimoto, “Parallel Discrete Event Simulation,” *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, 1990.
- [62] K. M. Chandy and J. Misra, “Asynchronous Distributed Simulation via a Sequence of Parallel Computations,” *Communications of the ACM*, vol. 24, no. 4, pp. 198–206, 1981.
- [63] A. Piccione, P. Andelfinger, and A. Pellegrini, “Hybrid Speculative Synchronisation for Parallel Discrete Event Simulation,” in *SIGSIM-PADS ’23: Proceedings of the 2023 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, New York, NY, USA, 2023, pp. 84–95.
- [64] J. S. Steinman, “SPEEDES: Synchronous Parallel Environment for Emulation and Discrete-Event Simulation,” *Advances in Parallel and Distributed Simulation (PADS ’91)*, vol. 23, pp. 95–103, 1991.
- [65] R. B. Osborne, “Speculative Computation in Multilisp,” in *Proceedings of the 1990 ACM Conference on LISP and Functional Programming*, Nice, FR, 1990, pp. 198–208.
- [66] *IEEE Standard for Distributed Interactive Simulation–Application Protocols*, IEEE Std 1278.1-2012 (Revision of IEEE Std 1278.1-1995), 2012.
- [67] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, “The R*-tree: An Efficient and Robust Access Method for Points and Rectangles,” in *Proceedings of the 1990 ACM SIGMOD international conference*, Atlantic City, NJ, USA, 1990, pp. 322–331.
- [68] H. Samet, *Foundations of Multidimensional and Metric Data Structures*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann, 2006.
- [69] *NIST Reference on Constants, Units and Uncertainty*, National Institute of Standards and Technology, 2025.
- [70] P. J. Denning, “The Art of Operating Systems: An On-Line Book of Slides,” Denning Institute, 2025. Accessed July 02, 2025. Available: <https://denninginstitute.com/pjd/ArtOS2/>

- [71] C. W. Reynolds, “Flocks, Herds and Schools: A Distributed Behavioral Model,” in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, Anaheim, CA, USA, 1987, pp. 25—34.
- [72] M. Macedonia, “A Network Software Architecture for Large Scale Virtual Environments,” Ph.D. dissertation, Department of Computer Science, Naval Postgraduate School, Monterey, CA, 1995.
- [73] G. Fiedler, “Reliable Ordered Messages,” Gaffer on Games, blog, 09 2016. Available: https://gafferongames.com/post/reliable_ordered_messages/
- [74] G. E. Krasner and S. Pope, “A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk80 System,” ParcPlace Systems, Inc., Mountain View, CA, USA, Rep., 1988. Available: https://web.archive.org/web/20100921030808/http://www.itu.dk/courses/VOP/E2005/VOP2005E/8_mvc_krasner_and_pope.pdf
- [75] A. Reed. *Archer, SIE1*. (Sept 2009).
- [76] I. Newton, *Philosophiae Naturalis Principia Mathematica*, 1st ed. London, UK: Smith, 1687.
- [77] NASA Glenn Research Center, “Newton’s Laws of Motion,” NASA Glenn Research Center: Beginners Guide to Aeronautics, July 2024. Available: <https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/newtons-laws-of-motion/>
- [78] “Ping Time Between New York and Los Angeles - WonderNetwork,” WonderNetwork, Apr 2025. Available: <https://wondernetwork.com/pings>
- [79] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, “Fast Proximity Queries with Swept Sphere Volumes,” University of North Carolina, Chapel Hill, Chapel Hill, NC, USA, Rep. TR99-018, 1999. Available: <http://gamma.cs.unc.edu/SSV/ssv.pdf>
- [80] L. Peitso and D. Brutzman, “Defeating Lag in Network-Distributed Physics Simulations: An Architecture Supporting Declarative Network Physics Representation Protocols,” in *Proceedings of the 23rd International ACM Conference on 3D Web Technology*, 2018. Available: <https://doi.org/10.1145/3208806.3208826>
- [81] L. Peitso and D. Brutzman, “Defeating Lag in Network-Distributed Physics Simulations,” *Graphical Models*, vol. 111, p. 101075, 2020. Available: <https://doi.org/https://doi.org/10.1016/j.gmod.2020.101075>
- [82] K. L. Morse, “An Adaptive, Distributed Algorithm for Interest Management,” Ph.D. dissertation, Department of Computer Science, University of California, Irvine, Irvine, CA, 2000.

- [83] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2019 (Revision of IEEE 754-2008), 2019.
- [84] K. Erleben, “Stable, Robust, and Versatile Multibody Dynamics Animation,” Ph.D. dissertation, Department of Computer Science, University of Copenhagen, Copenhagen, DK, 2004.
- [85] J. Russo and A. Russo. *Avengers: Infinity War*. (Apr 2018).
- [86] D. Brin, *Earth*, 1st ed. New York, NY, USA: Spectra/Ballantine Books, 1991.
- [87] B. Edwards, “Niantic Uses Pokémon Go Player Data to Build AI Navigation System,” *ars Technica*, Nov 2024. Available: <https://arstechnica.com/ai/2024/11/niantic-uses-pokemon-go-player-data-to-build-ai-navigation-system/>
- [88] “OpenStreetMap,” OpenStreetMap.org, 2025. Accessed Dec. 03, 2025. Available: <https://www.openstreetmap.org>
- [89] *The Oxford English Dictionary*. Accessed January, 12, 2026.
- [90] S. Mann and C. Wyckoff, “Extended Reality,” working paper, Massachusetts Institute of Technology, Cambridge, MA, USA, 1991. Available: <http://wearcam.org/xr.txt>
- [91] J. K. Rowling, *Harry Potter and the Sorcerer’s Stone*, 1st ed. New York, NY, USA: Scholastic, 1997.
- [92] N. Stephenson, *Snow Crash*, 1st ed. New York, NY, USA: Bantam Books, 1992.
- [93] RTX, “Electro-Optical Distributed Aperture System,” RTX, 2025. Accessed Dec. 03, 2025. Available: <https://www.rtx.com/raytheon/what-we-do/air/eodas>
- [94] B. Knobloch, “Military Mission Planning Using Augmented Reality Technology,” M.S. thesis, Naval Postgraduate School, Monterey, CA, 2020.
- [95] L. E. Peitso and J. B. Michael, “The Promise of Interactive Shared Augmented Reality,” *Computer*, vol. 53, no. 1, pp. 45–52, 2020.
- [96] S. D. Webb, S. Soh, and W. Lau, “RACS: A Referee Anti-Cheat Scheme for P2P Gaming,” in *Network and Operating System Support for Digital Audio and Video (NOSSDAV) 2007*, 2007. Available: <https://www.nossdav.org/2007/files/file-15-session2-paper2-webb.pdf>
- [97] M. Véron, O. Marin, S. Monnet, and Z. Guessoum, “Towards a Scalable Refereeing System for Online Gaming,” in *2012 11th Annual Workshop on Network and Systems Support for Games (NetGames)*, Venice, IT, 2012, pp. 1–2.

- [98] Riot Games, “VALORANT Support: What is Vanguard?” Jan 2024. Available: <https://support-valorant.riotgames.com/hc/en-us/articles/360046160933-What-is-Vanguard>
- [99] Valve Corporation, “Valve Anti-Cheat (VAC) System,” Steam Support, 2026. Accessed July 02, 2025. Available: <https://help.steampowered.com/en/faqs/view/571A-97DA-70E9-FF74>
- [100] A. S. Hernandez, “Breaking Barriers to Design Dimensions in Nearly Orthogonal Latin Hypercubes,” Ph.D. dissertation, Department of Operations Research, Naval Postgraduate School, Monterey, CA, 2008.
- [101] G. Klein, “A Recognition Primed Decision (RPD) Model of Rapid Decision Making,” in *Decision Making in Action: Models and Methods*, G. A. Klein, J. Orasanu, R. Calderwood, and C. E. Zsombok, Eds. Norwood, NJ, USA: Ablex Publishing, 1993, pp. 138–147.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
Fort Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California



DUDLEY KNOX LIBRARY

NAVAL POSTGRADUATE SCHOOL

WWW.NPS.EDU

WHERE SCIENCE MEETS THE ART OF WARFARE