1. Reversible Transactions

1.1 Mathematical Foundations

Adapted from sections of ./AE-Specifications/chapters/@GPT-reversibility.tex

In low-latency, high-throughput Layer 2 environments (e.g., Ethernet links), it's useful to model transactions as mathematical operations that can be precisely undone. This enables rollback, audit, and error recovery without heavyweight protocols.

1. **Model data as vectors.**
   Each Ethernet frame is viewed as a vector in $\mathbf{GF}(2)^n$, treating bits not as opaque payload but as elements in a vector space over a finite field.
2. **Transactions as invertible operations.**
   The sender and receiver maintain a shared state $S \in \mathbf{GF}(2)^n$. A transaction is an invertible linear transformation T applied to that state: $S' = T(S)$. Because T is invertible, the original state can always be recovered via T^{-1}.
3. **Reversibility via state updates.**
   To reverse a transaction, one sends a message (or derivable signal) allowing the application of T^{-1}. This guarantees deterministic rollback.

We consider a chain of $N+1$ nodes labeled $A_0 \to A_1 \to \cdots \to A_N$, where each node A_i maintains a local state vector $S_i \in \mathbf{GF}(2)^n$, typically initialized to the all-zero vector 0^n or some other agreed-upon state. Each link $(A_i \to A_{i+1})$ between adjacent nodes is associated with an invertible linear transformation $T_{i,i+1}$, which governs how state updates propagate along the chain.

1.1.1 Forward Execution

To execute a transaction spanning all links:

1. At each hop i, node A_i applies $T_{i,i+1}$ to its state S_i and transmits the transformation to A_{i+1}.
2. Node A_{i+1} applies the same $T_{i,i+1}$ to its own state S_{i+1}, maintaining link-local consistency.

The result is a chained sequence of transformations:

$$S_i' = T_{i,i+1} \cdot S_i \quad \text{for } i = 0, 1, \ldots, N-1, \quad S_N' = T_{N-1,N} \cdot S_N.$$

1.1.2 Rollback (Reverse Direction)

Reversibility is achieved by applying the inverse transformations in reverse order:

1. Node A_N applies $T_{N-1,N}^{-1}$ to revert S_N' to S_N.

2. It signals node A_{N-1}, which applies $T^{-1}_{N-1,N}$ and then $T^{-1}_{N-2,N-1}$, and so on.
3. This continues up the chain until A_0 applies $T^{-1}_{0,1}$, restoring the original S_0.

1.1.3 Example: XOR-Based Masks

If each $T_{i,i+1}$ is a simple XOR with mask $\Delta_{i,i+1}$, then:

$$S_i \mapsto S_i \oplus \Delta_{i,i+1}, \quad S_{i+1} \mapsto S_{i+1} \oplus \Delta_{i,i+1}.$$

Reversing just involves reapplying the same mask due to $\Delta \oplus \Delta = 0$.

1.1.4 Notes on Synchronization

- **Acknowledgments:** Each node should confirm that the next node has applied its transformation before committing its own.
- **Composite View:** The full transaction across N links is a composition:
$$T_{\text{total}} = T_{N-1,N} \circ T_{N-2,N-1} \circ \cdots \circ T_{0,1}.$$
- **Error Handling:** Any failure in transmission or transformation must be detected early, as desynchronization across nodes can compound. Redundant encodings, checksums, or commit/abort protocols may be used.

1.2 Atomic Transactions on Æ-Link

1.2.1 One-Phase Commit

1.2.2 Two-Phase Commit

1.2.3 Four-Phase Commit

1.3 Flow Control and Backpressure

1.4 Transactions on Trees

1.5 Reversible Transactions over Ethernet Links

GPT-Compressed adaptation of ./AE-Specifications-ETH/chapters/@Grok-reversibility-original.tex

Atomic Ethernet supports deterministic reversibility at the transaction level. This enables operations to be undone or rolled back by design—without ambiguity or loss—by embedding invertible transformations into the transmission semantics. Here, we outline a formal foundation for such reversibility using linear algebraic constructs, applied to unidirectional flows over a point-to-point Ethernet link.

1.5.1 Framing Transactions as Linear Operators

Let each transmitted message $m \in \mathbb{F}_2^n$ be a vector in a binary vector space. The sender applies a reversible transformation $T \in GL(n, \mathbb{F}_2)$—an invertible matrix over the Galois field $\mathbb{F}_2$—to produce the transmitted payload:

$$m' = T \cdot m$$

At the receiver, the inverse transform T^{-1} restores the original message:

$$m = T^{-1} \cdot m'$$

This formulation ensures that the transformation is loseless and decodable, and that every transaction can be reversed precisely, assuming both ends share T and agree on a consistent ordering.

1.5.2 Design Implications

Reversibility has several architectural consequences:

- **State Preservation:** Nodes maintain minimal internal state beyond the invertible transformation, supporting rollback without checkpointing.
- **Deterministic Rollback:** If a fault or cancellation occurs, the receiver may return m' along with T^{-1} or an encoded undo message, allowing the sender to revert application state.
- **Causal Provenance:** Every transformation T acts as a causal marker for the transaction's origin and structure. This ensures full verifiability of message lineage and intent.

1.5.3 Failure Recovery and Time Symmetry

In conventional systems, rollback is an afterthought—an external protocol layered above an irreversible transmission substrate. By contrast, Atomic Ethernet supports *built-in reversibility*, which permits time-symmetric protocols where forward progress and backtracking are mirror images.

A link failure mid-transaction results in an incomplete vector transformation. Since the transformation is linear and invertible, partial data receipt can trigger a retry or reversion without ambiguity. The system may encode a rollback transaction as $-T \cdot m$ or transmit a companion transaction to cancel the original.

1.5.4 Physical and Logical Layer Interface

Reversible transactions reside above the PHY layer, but the encoding scheme must be amenable to in-line streaming and backpressure. Slices are transmitted in fixed-size, entropy-bounded chunks, each marked with the transformation context. Intermediate nodes (in a multi-hop path) may propagate or transform T according to routing decisions, but the final receiver must be able to apply a composite inverse.

[MISSING FIGURE "linear-reversbile"]

1.5.5 Applications and Extensions

This mechanism enables new classes of semantics-aware networking:

- **Reversible Compute Fabric:** Transactions may represent computation steps. Reversal allows rollback of mispredictions or branch divergence in distributed computation.
- **Lossless Failure Handling:** Rather than assuming losses, the protocol treats all disruptions as reversible events, ensuring that no state is committed without an acknowledgment or its inverse.
- **Formal Auditing:** Because each transmission is encoded via known operators, a complete proof-of-history can be generated for compliance or replay.

1.5.6 Mathematical Construction

Let the application payload $d \in \mathbb{F}_2^n$ represent a fixed-size bit vector, e.g., 512 bits for a standard Ethernet frame. We model d as a column vector in a binary vector space.

To encode the transaction, the sender selects an invertible matrix $T \in GL(n, \mathbb{F}_2)$, producing the transmitted record:

$$r = T \cdot d$$

where:

- T is an $n \times n$ matrix, generated such that $\det(T) = 1$, i.e., T is invertible.
- The matrix T may be predefined, negotiated, or derived from a seed agreed by both parties.
- The receiver recovers d by applying the inverse: $d = T^{-1} \cdot r$

Example: Suppose $d = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$ and the transformation matrix is

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Then the transmitted vector becomes:

$$r = T \cdot d = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

This reversible encoding guarantees that no information is lost in transmission, and rollback is always feasible.

1.5.7 Encoding Strategies

The matrix T can be chosen to reflect transaction metadata:

- **Semantic Operators:** Specific rows or structures in T may denote specific transaction types (e.g., write, abort, checkpoint).
- **Forward-Only Compression:** Even when reversal is not expected, the matrix framework enables low-entropy encoding and structured inference.
- **Replay Protection:** Nonces or keys can be embedded into T to thwart reapplication of stale transactions.

1.5.8 Operational Summary

Encoding Sender maps data d to transmitted record $r = T \cdot d$

Transmission Frame r is sent with metadata identifying or referencing T

Reception Receiver decodes via T^{-1} to recover d

Rollback If reversal is needed, transmit inverse operation using $T^{-1} \cdot r$ or explicitly send a rollback instruction encoded via a known T_{undo}

This mechanism ensures that every transaction in the link pipeline has a symmetric undo path, forming the basis for reversible computing and auditable networking semantics.

1.6 From Clos to Graph: A Shift in Systems Thinking

Conventional datacenter networks built on Clos topologies and Kubernetes orchestration suffer from layered human dependencies: from manual switch configuration to YAML sprawl, from static policy declarations to failure-prone operational recovery. These fragilities compound superlinearly at scale.

Æthernet reimagines this by introducing an alternative: a Direct-Connected 8-Valency IPU mesh, running a decentralized Graph Virtual Machine (GVM). This system inverts operational complexity—delegating routing, scheduling, and security to topology-aware algorithms executed on a uniform graph structure. The goal is zero-trust-by-default, self-partitioning, and human-optional orchestration.

1.7 The Graph Virtual Machine

At its core, the GVM exposes programmable primitives for graph manipulation:

- `traverse(type, source, target)` – initiate message routing via BFS, DFS
- `partition(method, subgraphs)` – divide the network using k-means, MST, or spectral cuts
- `optimize(metric, scope)` – apply shortest path, max flow, or latency tuning globally or locally
- `deploy(tenant, subgraph, policy)` – install workloads subject to affinity, proximity, or isolation constraints

Every instruction is compiled into local actions at each node. No global controller is required.

1.8 Autonomy at 10,000 Nodes

Unlike Kubernetes, which becomes brittle beyond 1,000 nodes due to centralized control planes, the GVM scales linearly. Each IPU communicates only with 8 neighbors and executes a local GVM. Graph operations (e.g., BFS) run in $\mathcal{O}(\log n)$ time.

Maintenance, routing, and failure handling are executed as streaming graph updates. No YAML, no kubectl, no overlays.

1.9 Security via Confinement and Covers

GVM supports autonomous security through:

- **Graph Covers:** Define disjoint regions of the graph to isolate tenants.
- **Stacked Trees:** Construct independent spanning trees for redundant broadcast or failover paths.
- **Dynamic Partitions:** Tenants exhibiting anomalous behavior are moved to smaller or isolated subgraphs.

Attack surfaces are localized to 32 nodes at most. No blast radius extends across racks.

1.10 AI-Augmented Programming

AI assistants integrated with GVM support:

Policy Translation Turn human intent into graph ops (e.g., "keep tenant A near B but far from C").

Optimization Hints Suggest `optimize(flow)` or `partition(k-means)` based on real-time metrics.

Security Monitoring Isolate compromised nodes via AI-derived instruction chains.

Graph Debugging Visualize topologies and bottlenecks, offering suggestions for rerouting or partitioning.

The AI becomes the "network engineer," shifting the cognitive load away from humans.

1.11 Resilience via Topology

The 8-valent IPU mesh has no switches—only direct node-to-node links. This architecture achieves:

- **High Dynamic Laplacian Resilience (DLR):** With 8 neighbors, nodes tolerate up to 7 link failures before isolation.
- **No Leaf Bottlenecks:** Unlike Clos networks, where a failed top-of-rack switch can isolate 32 nodes, this mesh has no such point of failure.
- **Self-Healing:** Graph operations reconfigure routing paths in microseconds.

1.12 Mars-Scale Simplicity

In constrained environments—e.g., Martian colonies—human labor is scarce and risk is intolerable. GVM's autonomy makes it ideal:

- **Resilient:** Survives link loss, power failures, or electromagnetic damage
- **Minimal Ops:** Colonists specify goals; GVM and AI enact them
- **Self-Scaling:** From 100 to 10,000 nodes with no added complexity
  [MISSING FIGURE mars-network.pdf]

1.13 Why This Replaces Kubernetes

Metric	Kubernetes/Clos	GVM/IPU
Configuration	Manual YAML, CNI plugins	Self-partitioning graph ops
Operation	Human-tuned schedulers	Fully autonomous
Security	Declarative + fragile	Dynamic + confined
Latency	$10-30\,\mu$s typical	$\sim 50-100$ns
Failure Domain	Rack-wide (32+ nodes)	1–8 nodes max
Scaling	$n \sim 1,000$ ceiling	$n \geq 10,000$ trivial

1.14 Rethinking Atomicity: Counterfactual Transactions

From ./AE-Specifications-ETH/standalone/Transactions-maybe-dup.tex

This document challenges the Forward-In-Time-Only (FITO) assumptions behind conventional transactions in distributed systems. It argues that atomicity, as currently conceived, is a flawed abstraction and proposes a framework for reversible subtransactions as a more robust alternative.

"Transactions begin and they end."
—Charlie Johnson, TMF Product News

This simple phrase conceals deep design hazards. Transactions appear to begin with a trigger and end with a commit, but in distributed systems, these bookends obscure severe internal inconsistencies.

At issue are the mechanisms we use to track and guarantee these transactional intervals: timestamps, logs, filesystems, and even our concepts of causality. Each introduces cracks in the facade of atomicity.

1.15 The Forward-In-Time-Only Fallacy

FITO: Forward-In-Time-Only thinking assumes linear causality.

Most distributed systems today adopt what we call **Forward-In-Time-Only (FITO)** thinking. That is:

1. **Open a transaction** with a timestamp.
2. **Apply a sequence** of operations.
3. **Close the transaction** with a commit or rollback.

But this approach breaks down under scrutiny.

Three FITO Hazards

1. **Timestamps are not unique.** Even on a single machine with GHz processors and nanosecond clocks, timestamp collisions occur. OS-level clock management does not guarantee uniqueness.
2. **Timestamps are single points of failure.** Any drift, packet loss, or sync error in NTP/PTP introduces false ordering assumptions.
3. **Simultaneity is an illusion.** Relativity tells us simultaneity is observer-dependent. Building global event orderings on timestamps is unsafe.

1.16 The False Comfort of Atomicity

We often say: "all or nothing." But our stack is built on sand:

- The database relies on the log.
- The log relies on the filesystem.
- The filesystem relies on `fsync`.
- `fsync` relies on storage hardware.

Each of these layers fails to guarantee true atomicity. When one fails, the recovery model becomes: **Smash and Restart**.

1.17 The Myth of Reliable Commit

Protocols like Two-Phase Commit (2PC) attempt to enforce distributed agreement. But they depend on:

- Log synchronization across nodes
- Network reliability
- Time-based coordination

When any assumption breaks, so does safety. Eventually, we replace consistency with survivability—and correctness with heuristics.

1.18 Toward Reversible Thinking

Suppose we reject FITO. Suppose we view the transaction as *reversible*.

If the forward protocol is correct, we can construct a reverse protocol.

This leads to **reversible subtransactions**: bounded operations that can be undone without global rollback.

Counterfactual Transactions

- A transaction can end, then begin again.
- Logs become braids, not linear sequences.
- Atomicity becomes a constraint, not an assumption.

Inspired by Marletto's counterfactual physics, this model embraces partial reversibility as an engineering practice.

1.19 Closing the Interval—Reopened

Simultaneity is not fundamental. Causality is.

"Closing the interval" with a commit only makes sense if we know the state is stable. In reality, it's a guess based on layers of non-atomic operations.

By rethinking transactions through reversible logic, we can:

- Define precise causal dependencies
- Undo partial effects
- Recover without restart

Reversibility isn't science fiction. It is what rollback always wanted to be.

1.20 Conclusion

The abstraction of atomicity has outlived its usefulness as a guarantee. In modern distributed systems, FITO thinking and timestamp dependency introduce hazards we can no longer ignore.

It is time to engineer **reversible protocols**, built on causal semantics—not illusions of simultaneity. Let us design transactions that don't just commit or roll back, but that can *unwind*.

1.21 Reversible Transactions over a Single Ethernet Link

Adapted from ./AE-Specifications-ETH/@GPT-reversibility.tex

Traditional transaction protocols over Ethernet assume a forward-only time model. Failures are handled through retries, timeouts, or speculative execution. This results in complexity, energy waste, and difficulty in verification. Inspired by physics, we propose a model in which every Ethernet transmission is reversible: a transaction is not committed until its inverse is impossible.

Vector-Based Framing

Rather than treating Ethernet frames as opaque byte sequences, we treat them as elements of a vector space over $\mathbb{F}_2^n$. This allows each transmission to be encoded as a linear transformation, and, crucially, inverted.

- A single frame becomes a vector $v \in \mathbb{F}_2^n$.
- A transaction is a linear operation $T : v \mapsto Tv$.
- Inverse operations T^{-1} can be issued on failure, restoring prior state.

Two-Phase Semantics

Each transaction operates in two stages:

Phase 1: Tentative Forward Transmission
The initiator sends a vector transformation. The responder buffers the transformation but does not apply it semantically until Phase 2.

Phase 2: Acknowledgment or Inversion
If successful, a commit acknowledgment is returned. If failure or timeout, the initiator issues T^{-1}, instructing the responder to reverse the tentative state.

Benefits of Reversibility

- **Verifiability:** All transactions are algebraically closed and auditable.
- **Composability:** Compound transactions can be expressed as chains of linear transformations.
- **Energy Efficiency:** Rollbacks avoid speculative execution and wasted computation.
- **No Silent Failures:** The absence of acknowledgment implies an outstanding transformation requiring cleanup.

Implementation Implications

Æthernet links may cache a small rollback buffer per transaction and implement an algebraic acknowledgment loop. This loop behaves as a "circular snake" — with the head and tail of a transaction circulating on the link until either:

- The transaction is committed (the tail is consumed).
- The transaction is explicitly reversed (the tail eats the head).

This model supports deterministic, lossless computing over unreliable mediums by ensuring all state transitions are recoverable — a local time-symmetric alternative to global distributed consensus.

Mathematical Formulation

Let $\mathbf{v} \in \mathbb{F}_2^n$ represent the contents of an Ethernet frame, modeled as an n-dimensional vector over the binary field. Each transaction is a linear transformation:

$$T : \mathbb{F}_2^n \to \mathbb{F}_2^n, \quad \mathbf{v} \mapsto T\mathbf{v}$$

Here, T is an $n \times n$ binary matrix with full rank (i.e., $\det(()T) \neq 0$ in $\mathbb{F}_2$), ensuring invertibility:

$$T^{-1}T\mathbf{v} = \mathbf{v}$$

Each Ethernet transaction must store the pair $(T, \mathbf{v})$ temporarily, enabling the rollback operation:

$$\mathbf{v}' = T\mathbf{v} \quad \text{and} \quad \text{on failure: } T^{-1}\mathbf{v}' = \mathbf{v}$$

This structure creates a *reversible pipeline*, where only acknowledged transformations become final state.

Transaction Algebra and Composition

Multiple transactions may be composed sequentially:

$$\mathbf{v}_2 = T_2(T_1\mathbf{v}_0) = (T_2T_1)\mathbf{v}_0$$

To rollback to any previous stage, it suffices to apply the inverse of the composed transformation:

$$\mathbf{v}_0 = (T_2T_1)^{-1}\mathbf{v}_2 = T_1^{-1}T_2^{-1}\mathbf{v}_2$$

This algebraic chaining enables deterministic state replay and rollback — suitable for verifying transaction histories without logs.

Interpretation as Homology

A reversible link can be seen as enforcing *cycle closure*. That is, any sequence of transformations $T_1, T_2, \ldots, T_k$ across the link must eventually yield:

$$T_k \ldots T_2T_1 = I \quad \text{(identity)}$$

unless a semantic commit is agreed upon and the cycle is broken. This makes every transaction loop homologically equivalent to zero until commitment. The state machine on each end must therefore enforce that:

$$\sum_{i=1}^{k} T_i \in \ker(\text{Commit})$$

until an acknowledgment collapses the pending cycle into permanent change.

Reversibility as a Link Invariant

Just as error correction preserves information despite physical noise, reversible algebra ensures *semantic reversibility* across potentially unreliable hardware. This enables a protocol invariant:

$$\forall t, \quad \text{State}_{\text{link}}(t) \in \text{Image}(\mathcal{T}^{\pm})$$

where $\mathcal{T}^{\pm}$ is the group of invertible transformations. All observable state transitions fall within this reversible group action.

Acknowledgment as Projection

Acknowledgment is not merely a signal but a *projection operator* $\mathcal{P}$ that collapses the reversible superposition of possible futures into a committed state:

$$\mathcal{P}(T\mathbf{v}) = \mathbf{v}_{\text{committed}}$$

Until this projection is received, the state must be maintained in an entangled (reversible) buffer space — a form of local decoherence management.

Extending Reversibility to Graphs

While the reversible transaction model began with point-to-point (1D) links, it naturally generalizes to multi-node graphs.

Spanning Tree for Determinism To maintain deterministic reversibility in a graph of n nodes, define a spanning tree T_s rooted at a node r. Each transaction along edge (i, j) becomes a matrix T_{ij} with:

$$T_{ij}^{-1} T_{ij} = I$$

The system maintains a log of traversed edges to support backward computation if recovery is triggered.

Multi-path and DAG Tensions Acyclic paths support clean reversibility. In contrast, fully connected graphs (K_n) or cyclic DAGs may complicate rollback semantics unless time-bounded checkpoints are defined. Reversibility can be ensured by constraint:

$$\text{Path}(r \to t) \in \text{Tree}_{\text{epoch}}$$

Where $\text{Tree}_{\text{epoch}}$ rotates to provide redundancy without losing invertibility guarantees.

Valency Constraints and Physical Limits

Most physical Ethernet or SerDes ports limit a node's degree to 8 or fewer. As such, highly connected graphs (K_n for $n > 9$) are infeasible. Instead, the protocol supports:

- **Hypercube Topologies:** Efficient for uniform, fault-tolerant routing under degree constraints.
- **Torus and Mesh Graphs:** Enable deterministic reversibility and spatial locality.
- **Tree-Stacked Subgraphs:** Spanning trees with redundant shadows for failover support.

Dynamic Laplacian Resilience (DLR)

We define a resilience metric for reversible Ethernet graphs under link failures:

$$\mathrm{DLR}(G) = \frac{1}{k} \sum_{i=1}^{k} \lambda_2(G_i)$$

where G_i is the graph after i failures and λ_2 is the second-smallest eigenvalue of the Laplacian matrix (algebraic connectivity). High λ_2 implies robust path redundancy and invertibility.

Snake-Based Circulation Model

In the reversible framing model, the "packet" and its acknowledgment together form a single logical object — a snake of bits that must eventually return to its source or be rolled back. Let:

- $\mathrm{Head}(T\mathbf{v})$ be the transmitted transformation.
- $\mathrm{Tail}(T^{-1})$ be the reversible closure.

Completion occurs when both ends of the transformation exist and collapse:

$$\mathrm{Commit}(T\mathbf{v}) \iff \text{Tail returns within timeout and matches } T^{-1}$$

This model avoids deadlocks and waste. Only transformations that complete the loop can semantically alter state.

Time Symmetry in Protocol Design

All operations are temporally bidirectional unless explicitly committed. This mirrors time symmetry in physics and allows for:

- **Exact Rollbacks:** Even at multi-hop or multi-frame granularity.
- **Auditable Transactions:** Linear algebra ensures closure and reconstructibility.

- **Semantic Isolation:** No side effects are propagated until loop closure.

Implications for Reversible Hardware

The reversibility model has strong implications for hardware design:

- **Frame Buffers:** Temporarily store $(T, \mathbf{v})$ until acknowledgment or rollback.
- **Commit Logic:** Frame commits must be locally verifiable by algebraic identity.
- **Link Loops:** The head and tail of a transaction can circulate on the same physical link, forming a tight loop.

In short links (e.g., chiplets), the "snake" can be fully contained in the round-trip delay of the cable. This permits sub-microsecond full transaction turnaround without global arbitration.

Conclusion: Reversibility as a First-Class Network Property

Reversible computation has long been a topic in theoretical computer science, but with Æthernet, it becomes a foundational property of the link layer. The implications are:

1. **All state transitions are reversible until observed.**
2. **Transactions are modeled as group operations with inverse semantics.**
3. **Deterministic rollback replaces speculative retry.**

This model positions Æthernet as the first Ethernet protocol that enforces reversibility as a *physical layer invariant*, enabling full-stack semantic safety — from API to transceiver.

1.22 Linearizability vs. Hypertransactions

From DAE-Technical/Mulligan Stew.pages

There is no such thing as a real-time order. Wall clocks are illusions; physics knows no absolute now.

1.23 Linearizability and Its Limits

1.23.1 What is Linearizability?

Linearizability is a consistency model for concurrent systems that provides a real-time ordering of operations, making the system's behavior appear as if operations were executed instantaneously at some point between their invocation and response.

Real-Time Order: Operations appear to take effect at a single point between invocation and response.

Consistency and Atomicity: Guarantees transition between consistent states as if operations were sequential.

Indistinguishability: Observers cannot distinguish between the actual execution and a hypothetical, linearized one.

1.23.2 Physical Reality

While this may be intuitive for computer scientists, modern physics rejects any notion of absolute real-time order. Special and General Relativity, as well as Quantum Mechanics, demonstrate that causality is indefinite, and entanglement is non-local.

1.23.3 FITO Thinking

Linearizability is a **Forward-In-Time-Only (FITO)** concept. It presumes monotonic progress, ignoring the need to reverse operations. In reality, reversing steps (like backing out of a one-way street) is essential to ensure availability and resilience.

1.24 Hypertransactions

1.24.1 Reversible Subtransactions in FPGA

Hypertransactions are *reversible subtransactions executed entirely within the FPGA substructure*.

> **Moss defines a transaction as:** *"collections of primitive actions that are indivisible, ensuring consistency even with concurrency or failure."*

Despite decades of effort, the industry has failed to achieve true indivisibility. Instead of enforcing an irreversible timeline, we introduce reversibility into the design.

1.24.2 Nested Transactions and Synchronization

Nested transactions enable nested universes of synchronization and recovery:

- Subtransactions can fail independently without aborting the parent.
- Define "reversibility points" to backtrack from local failures.

We use 'invocation' and 'response' as reversible events to guide ancilla bits in managing forward/backward state machine evolution.

1.24.3 From Locking to Multiway Systems

Moss relies on locking, distributed commit, and deadlock detection. We move beyond this to MultiWay Systems, allowing multiple possible valid outcomes in superposition. Determinism is achieved *on demand*.

1.25 HyperIsolation: A New Atomicity Primitive

1.25.1 Atomicity Without Compromise

DAE overcomes the performance/strictness tradeoff of conventional atomicity:

- Violations become visible and recoverable.
- Cell agents monitor liveliness and reroute stalled links.
- Recovery occurs in 2–4 FPGA cycles (sub-microsecond), appearing as "unbreakable" to host processes.

1.25.2 Reversible Definitions of Atomicity

Where Moss defines atomicity in FITO terms, we extend it:

- Proofs hold in both forward and reverse directions.
- Reverse path is mathematically modeled (e.g., invertible functions in linear algebra).
- Alternative rollback paths to safe states are enabled.

1.26 Consistency Reimagined

1.26.1 Mathematical Consistency

Our consistency primitive is rooted in formal mathematical completeness, compositionality, and simulation. Verified in Mathematica, compiled into Petri nets, and synthesized to Verilog.

1.27 HyperIsolation: A New Isolation Primitive

We introduce an entangled, two-phase commit which:

- Transfers reversible tokens of varying size across FPGAs.
- Overcomes limitations of Serializability and S2PL.
- Prevents host-side blocking via TPI (Transaction Processing Interface).

1.28 Physics and Networking: A Rebuttal to FITO

1.28.1 Real-Time Order is an Illusion

Physics denies the concept of instantaneous global time. All observations are relative, and latency is inevitable.

1.28.2 Consistency and Atomicity Are Observer-Relative

- Transitions appear different to each observer.
- Faults expose inconsistent sub-states that cannot be masked.
- Redundancy and availability metrics do not guarantee determinism.

Entanglement means maintaining consistent-but-complementary states between peers.

1.28.3 Linearization Fails under Fault

Packet drops, reordering, and duplication break coherence between senders and receivers. Indistinguishability is violated. Sequence numbers alone cannot recover lost structure.

1.29 Sequence Numbers as Complex Modulo Counters

- Sequence numbers must be reversible modulo counters.
- Bounded interaction buffers (2, 4, ... N interactions) are provably atomic.
- We use complex-number representations, stored in lookup tables.

Note: Increasing counter width increases failure likelihood and recovery complexity.