The Deliberate Collapse of Cognitive Multiplicity: Event-History, Architectural Constraint, and the Economics of Memory

Flyxion

December 2025

1 Introduction

Contemporary discussions of intelligence, cognition, and computation are dominated by a narrow set of metaphors: information as static representation, memory as passive storage, and systems as collections of interchangeable modules optimized locally for performance. These metaphors persist not because they are theoretically adequate, but because they align with existing institutional, economic, and engineering practices.

This paper argues that such metaphors systematically obscure the true locus of intelligence: not in states, representations, or outputs, but in *event history*. What a system has becomeâĂŤhow it was constructed, constrained, branched, and revisedâĂŤmatters more than any instantaneous configuration it may occupy. When this perspective is taken seriously, familiar distinctions between cognition, infrastructure, and economics begin to collapse. Memory is revealed as an active process, architecture as a constraint on lawful composition, and pricing models as implicit theories of what kinds of history are permitted to exist.

The core thesis is threefold. First, cognition is fundamentally event-driven rather than state-driven. Second, architectures that respect this fact necessarily privilege compositional histories over monolithic optimization. Third, contemporary platform economics actively suppress event-history preservation through artificial scarcity, despite the existence of mature technologies that make such scarcity unnecessary.

To make this argument concrete, the paper proceeds in three stages. The first establishes an event-historical account of cognition and architecture. The second examines how large-scale platforms, particularly cloud storage systems, violate this account through economic design choices. The third introduces PlenumHub as a counter-example: a cooperative, content-addressable storage architecture whose economic model aligns with the informational structure of event histories rather than exploiting user ignorance of it.

Throughout, the analysis draws on control theory, computer systems, information theory, and the emerging Spherepop calculus, which provides a formal language for reasoning about commits, collapses, and compositional history.

2 From States to Events

The dominant formal models of cognition and computation describe systems in terms of states and transitions between them. While mathematically convenient, this perspective hides a crucial asymmetry: multiple distinct construction histories can lead to indistinguishable states, yet those histories often differ dramatically in their future affordances.

An intelligent system does not merely occupy a state; it inherits a construction lineage. This lineage determines which transformations are accessible, which abstractions are stable, and which generalizations remain lawful. Two systems that appear identical when inspected synchronically may diverge irreversibly once perturbed, precisely because their event histories differ.

This observation motivates a shift from state-based to event-based ontology.

Definition 1 (Event)**.** *An* event *is a constrained transformation that irreversibly modifies a systemâĂŹs future space of possible actions. Events are not observations but commitments: once enacted, they restrict subsequent lawful behavior.*

On this view, memory is not a store of representations but an accumulation of constraints induced by past events. Forgetting, similarly, is not erasure but collapse: the deliberate or coerced reduction of accessible historical detail in order to satisfy external constraints.

This distinction becomes critical when considering large-scale systems. Architectures that respect event history must preserve the compositional structure of those events, allowing partial replay, branching, and recombination. Architectures that erase or tax history instead force systems into brittle, overfitted regimes that appear efficient locally but fail catastrophically under novelty.

Remark 1. *State-based abstractions are not incorrect; they are lossy projections. The error arises when these projections are treated as ontologically primary rather than as summaries derived from event history.*

3 Architecture as Commutativity

The essence of architecture is not optimization but *commutativity*. A well-designed architecture allows a complex global problem to be decomposed into subproblems whose independent solutions compose to the same result as a monolithic solutionâĂŤwere such a solution tractable.

This principle is well understood in control theory, where layered designs permit separation of concerns without sacrificing optimality under explicit constraints. It is equally evident in software systems, where compositional interfaces allow independent development without global coordination.

What is less often acknowledged is that this commutativity depends on strict discipline at interfaces. Components must do one thing, accept well-defined inputs, and produce lawful outputs. Violations of these contracts break injectivity, rendering composition unreliable.

The Unix pipeline provides a canonical illustration. Programs composed via the pipe operator function correctly precisely because each program respects a convention: read from standard input, write to standard output, and avoid side effects. The pipeline is not powerful because of any single program, but because the architecture enforces compositional legality.

Crucially, this low-level compositional success does *not* justify institution-scale demands for human task specialization. The effectiveness of pipelines arises from abstracted event composition, not from narrow functional isolation. Human systems become productive not when individuals are reduced to single-purpose components, but when higher-order eventsâĂŤevaluated, hierarchically abstracted commitmentsâĂŤare preserved and recombined.

Outputs, in this sense, are always side effects. They should be deferred until the last possible moment, after the relevant event history has been evaluated and stabilized. Systems that privilege output over history inevitably destroy the very structure that makes meaningful output possible.

Proposition 1. *Architectures that optimize outputs without preserving event history will appear efficient locally while degrading global adaptability.*

Proof. Local output optimization collapses historical degrees of freedom. Since future adaptation depends on those degrees of freedom, adaptability strictly decreases under repeated collapse. □ □

4 Event-Driven Memory and the Logic of Collapse

If cognition is fundamentally event-driven, then memory must be understood as the preservation of event structure rather than the retention of symbolic content. Memory, on this account, is not a warehouse but a scaffold: a set of constraints that delimit which future transformations remain lawful.

An event does not merely add information. It alters the topology of the systemâĂŹs possibility space. Once an event is committed, certain continuations become available while others are foreclosed. The cumulative effect of events is therefore not additive but structural.

This perspective clarifies why memory loss is so often catastrophic even when large quantities of data remain intact. What is lost is not content but connectivity: the relations between events that permit recomposition, reinterpretation, and counterfactual exploration.

Collapse occurs when a system is forced to reduce its event history to satisfy external constraints. These constraints may be physical, computational, or economic. Regardless of origin, collapse eliminates degrees of freedom by destroying distinctions that cannot be recovered later.

Definition 2 (Collapse)**.** *A* collapse *is an irreversible reduction of event-history resolution imposed to satisfy a constraint external to the systemâĂŹs internal logic.*

Importantly, collapse is not inherently pathological. Strategic collapse is essential for action: a decision is a collapse of alternatives. The problem arises when collapse is imposed prematurely or systematically, before the relevant event structure has been adequately explored.

Premature collapse yields brittle intelligence. Systems that collapse early must commit to interpretations before they are justified, leading to overfitting, superstition, and loss of adaptability. This dynamic is well documented in machine learning, where early stopping or aggressive pruning can produce models that perform well on training data but generalize poorly.

The same phenomenon appears at the level of institutions and infrastructure. When systems impose economic or administrative limits on memory itself, they force users to collapse history not because it is cognitively optimal, but because it is financially or administratively required.

Remark 2. *Collapse induced by scarcity differs categorically from collapse induced by choice. The former erases structure indiscriminately; the latter preserves structure until evaluation is complete.*

5 Economic Constraints as Cognitive Constraints

Memory preservation requires resources. This banal observation acquires theoretical significance once it is recognized that pricing models implicitly encode assumptions about which kinds of memory matter.

When storage is priced per visible artifact rather than per informational entropy, systems penalize precisely those histories that are most valuable: long, branching, iterative records of exploration. Conversely, systems that reward compression and reuse implicitly encourage the preservation of deep event structure.

This distinction is not abstract. It manifests concretely in contemporary platform infrastructure, where users are charged repeatedly for duplicated content, incremental revisions, and shared cultural artifacts, despite the existence of mature technologies that render such duplication nearly free.

The result is a form of cognitive taxation. Users are compelled to delete, summarize, or externalize their own histories in order to remain within arbitrary limits. Memory becomes a subscription service rather than a right, and the capacity for long-term reasoning degrades accordingly.

Proposition 2. *Pricing models that scale with apparent volume rather than informational entropy systematically suppress event-history preservation.*

Proof. Incremental event histories consist primarily of small deltas over shared structure. Volume-based pricing charges repeatedly for shared structure, making long histories increasingly expensive. Users respond by deleting or collapsing history, reducing event resolution. □ □

This suppression is not an accidental byproduct of scale. It is a predictable consequence of economic models that prioritize rent extraction over cognitive infrastructure.

At this point, the analysis intersects directly with platform design. The question is no longer whether current systems could preserve event history, but why they systematically choose not to.

6 Architectural Simplification and Institutional Drift

Large-scale systems tend toward simplification under pressure of maintenance, coordination, and profit. Simplification, however, is ambiguous. It may refer to conceptual elegance, or it may refer to the elimination of degrees of freedom.

The former enhances intelligence; the latter degrades it.

In technical systems, productive simplification arises from abstraction that preserves compositional structure. In institutional systems, destructive simplification arises when complexity is displaced downwardâĂŤonto usersâĂŤwhile interfaces are narrowed to enforce compliance.

This pattern is visible across contemporary software platforms. Interfaces that once exposed compositional primitives gradually restrict user agency, removing access to typography, markup,

scripting, windowing, and interoperability. The stated justification is usability. The actual effect is dependency.

By constraining how users may act, platforms reduce the space of possible events users can generate. Over time, this trains users to operate within a narrow behavioral corridor, atrophying exploratory capacity and technical literacy.

The loss is cumulative. Once a population is trained to accept opaque systems and artificial limits as natural, the cognitive cost of alternatives becomes politically prohibitive. This is not because alternatives are technically difficult, but because users have been systematically denied the opportunity to understand how systems actually work.

Remark 3. *Institutional simplification that removes user-facing degrees of freedom functions as a long-term cognitive control mechanism, even when justified as convenience.*

The following sections examine this mechanism in detail through a specific case study: the evolution of consumer cloud storage, with particular emphasis on Gmail as a paradigmatic example of economic constraint masquerading as technical necessity.

7 Google and the Political Economy of Memory

The evolution of Gmail provides an unusually clear case study in how platform economics can directly suppress event-history preservation while presenting the resulting constraints as technical inevitabilities. Unlike many platform failures, this trajectory is well-documented, temporally bounded, and internally coherent. It therefore admits analysis not merely as corporate drift, but as a deliberate architectural inversion.

7.1 The Gmail Promise

When Gmail launched publicly in 2004, it positioned itself as a radical departure from prevailing email services. At the time, typical providers offered inbox capacities measured in megabytes. Gmail offered one gigabyte, an increase of two orders of magnitude, accompanied by a novel rhetorical frame: storage was not scarce, and users were no longer expected to curate their correspondence aggressively.

This framing was not incidental. GmailâĂŹs interface prominently displayed a running counter indicating that total storage capacity was continuously increasing. The message was explicit and pedagogical: storage was an expanding resource, and user participation contributed to economies of scale that benefited everyone.

The conceptual shift was profound. Email ceased to be treated as a volatile stream requiring constant pruning and became instead a persistent event history. Users were encouraged to retain correspondence indefinitely, to rely on search rather than deletion, and to treat their inbox as an externalized memory.

From the perspective developed earlier in this paper, Gmail briefly aligned with an event-driven conception of cognition. It lowered the cost of preserving historical detail, deferred collapse, and supported retrospective reinterpretation of past events.

7.2 The Enclosure

In 2013, this model was quietly abandoned. Google eliminated the public storage counter and introduced a unified quota of 15GB shared across Gmail, Google Drive, and Google Photos. Storage was reframed as a scarce personal resource, and users were compelled either to delete historical data or to purchase additional capacity.

This shift inverted every aspect of the original promise. Abundance was replaced by scarcity; mutual benefit by internal competition between data types; and transparency by opacity. Crucially, these changes occurred during a period in which the underlying costs of storage were declining rapidly due to advances in hardware density, compression algorithms, and distributed storage infrastructure.

The introduction of quotas was therefore not a response to technical pressure. It was an economic decision imposed after user dependency had been firmly established. Switching costs were high, export mechanisms were cumbersome, and users had reorganized their cognitive workflows around GmailâĂŹs search-centric model.

Remark 4. *The timing of GmailâĂŹs storage cap is diagnostic. Artificial scarcity was introduced only after user reliance rendered exit costly.*

7.3 Suppressed Infrastructure: Content-Addressable Storage

The most significant aspect of GmailâĂŹs pricing model is not the quota itself, but what it conceals. Modern distributed storage systemsâĂŤincluding those developed and operated by GoogleâĂŤemploy content-addressable storage and deduplication. In such systems, identical content is stored once and referenced many times. Storage cost scales with unique information, not with the number of users who reference it.

This architecture is neither experimental nor obscure. It underlies Git version control, distributed backup systems, and GoogleâĂŹs own internal file infrastructure. Its economic implications are straightforward: duplicated content is nearly free.

Despite this, Gmail charges users as though each copy of a file were stored independently. Popular images, shared documents, email attachments, and incremental revisions are all counted repeatedly against individual quotas, even when the underlying data already exists in the system.

The result is a systematic overcharging that scales with ignorance rather than cost. Users pay for storage that is not actually consumed, while the platform captures rent on the illusion of scarcity.

Example 1. *If millions of users receive the same email attachment, a content-addressable system stores the file once and distributes pointers. Gmail, however, debits each userâĂŹs quota as though a separate copy were required, despite the fact that GoogleâĂŹs actual storage cost remains effectively constant.*

7.4 Event-History Suppression

The economic consequences of this design choice are not merely financial. By charging for historical accumulation, Gmail induces users to delete or prune their own event histories. Long-term corre-

spondence, iterative drafts, attachments, and contextual records are selectively erased to remain within arbitrary limits.

This enforced curation collapses history prematurely. Users are compelled to decide what will matter in the future without the benefit of hindsight. The system thereby externalizes the cost of architectural simplification onto cognition itself.

From the perspective of event-driven memory, this constitutes a direct interference with lawful abstraction. Events that could have been preserved, reinterpreted, or recombined are destroyed not because they lack value, but because their continued existence threatens a revenue model.

Proposition 3. *GmailâĂŹs quota system functions as an economic constraint on memory, forcing premature collapse of event history.*

Proof. Incremental event histories generate minimal additional entropy but occupy visible storage volume. Volume-based pricing makes such histories increasingly expensive, incentivizing deletion. Deletion irreversibly destroys event structure. □ □

7.5 Information Feudalism

The Gmail trajectory exemplifies what may be termed *information feudalism*. Infrastructure that is technically abundant is enclosed through proprietary control, and users are charged rent for access to resources whose scarcity is artificial.

In classical feudal systems, land was abundant but access was restricted by force and custom. In digital systems, storage is abundant but access is restricted by interface design, pricing models, and deliberate opacity.

The analogy is not rhetorical. Users are discouraged from understanding how storage works, prevented from exporting data in content-addressable formats, and trained to accept quotas as natural constraints. The knowledge required to contest these limits is systematically withheld.

Remark 5. *Git remains a niche tool not because its underlying technology is inaccessible, but because consumer-facing platforms have no incentive to teach users how little storage actually costs.*

7.6 The Cognitive Cost of Obscurity

The suppression of storage literacy has secondary effects. Users cease to think in terms of versions, deltas, branches, and reuse. Incremental reasoning is replaced by snapshot thinking. History becomes a liability rather than an asset.

This erosion of algorithmic intuition mirrors patterns observed in compulsory schooling, where dependence on authority replaces understanding of systems. Platforms, like schools, benefit from compliant users who accept constraints without questioning their origin.

The next section introduces a counter-example. PlenumHub demonstrates that storage infrastructure can be organized to reward contribution, preserve event history, and educate users about the systems they depend on, rather than extract rent from their ignorance.

8 PlenumHub: Cooperative Infrastructure and Economic Inversion

The critique developed thus far would be incomplete without a constructive counter-example. It is not sufficient to argue that contemporary platforms suppress event history through artificial scarcity; one must also demonstrate that alternative architectures are technically feasible, economically viable, and cognitively superior.

PlenumHub is proposed as such a counter-example. It is not a speculative technology but a recombination of existing, well-understood mechanismsâĂŤcontent addressable storage, deduplication, delta compression, and cooperative governanceâĂŤorganized around an explicitly event-driven conception of memory.

The central insight underlying PlenumHub is simple: storage cost is determined by entropy, not by reference count. When systems price storage as though each reference were independent, they charge repeatedly for structure that already exists. When systems price storage according to informational novelty, they align cost with actual resource consumption.

PlenumHub formalizes this alignment.

8.1 Entropy-Aligned Storage

In a content-addressable system, data is indexed by the cryptographic hash of its content. Identical content, regardless of who uploads it or when, resolves to the same address. The result is immediate deduplication: storage cost is incurred once, while references are effectively free.

Delta compression extends this principle to near-identical content. Successive versions of a document, incremental revisions, or branching variants share most of their structure. Only the differencesâĂŤthe deltasâĂŤrequire additional storage.

From an event-history perspective, this is precisely the desired behavior. Events typically modify existing structure incrementally. A system that charges full price for each revision treats events as independent artifacts, thereby penalizing the very process of iterative reasoning.

PlenumHub instead charges only for what is irreducibly new. The practical approximation to this notion is achieved compression length, which serves as an upper bound on Kolmogorov complexity.

Definition 3 (Entropy-Aligned Cost)**.** *Let x be a piece of content and S the existing store. The storage cost of x is proportional to the minimal compressed representation of x relative to S.*

Under this model, popular content becomes cheaper over time, not more expensive. As patterns accumulate, compression improves. The system rewards reuse, reference, and continuity rather than punishing them.

8.2 Compression as Collective Good

A further inversion distinguishes PlenumHub from platform models. When a user introduces content that improves compression for othersâĂŤby adding a reusable pattern, structure, or referenceâĂŤthe

system as a whole benefits. Conventional platforms capture this benefit entirely. PlenumHub returns a portion of it to the contributor.

This mechanism treats compression not merely as an optimization but as a form of infrastructural labor. Users who enrich the shared pattern space reduce collective storage costs. Compensating this contribution aligns incentives without requiring centralized planning.

The compensation is logarithmic rather than linear. Early contributions to new pattern spaces are rewarded more strongly, while diminishing returns prevent runaway accumulation. The result is a stable distribution that encourages diversity without concentrating power.

Remark 6. *Logarithmic compensation mirrors the structure of informational gain: the first instance of a pattern provides maximal compression benefit; subsequent instances refine but do not dominate.*

8.3 Event-History Preservation

Because incremental change is cheap, PlenumHub removes the economic pressure to delete history. Branching, revision, and long-term accumulation are no longer liabilities. They are natural operations supported by the underlying architecture.

This has direct implications for cognition. Event histories can be preserved in full resolution, enabling retrospective analysis, alternative interpretations, and counterfactual reasoning. Memory ceases to be a subscription and becomes a structural right.

Consider a collaborative document evolving over time. In conventional cloud storage, each version consumes full quota, incentivizing aggressive pruning. In PlenumHub, the shared base is stored once, and each revision contributes only its unique delta. The cost of preserving the entire history approaches the cost of preserving the final version alone.

Proposition 4. *In an entropy-aligned storage system, the marginal cost of preserving an additional event in a shared history approaches zero as shared structure increases.*

Proof. Each additional event introduces a delta over existing structure. As shared structure dominates, the size of the delta becomes small relative to the whole. Compression exploits this redundancy, reducing marginal storage cost. □ □

8.4 Educational Transparency

PlenumHubâĂŹs interface is designed to expose, rather than conceal, the mechanics of storage. Users see when content is deduplicated, how much compression is achieved, and how their contributions affect others.

This visibility is not cosmetic. It trains users to reason in terms of hashes, deltas, and shared structure. Over time, users internalize an understanding of information theory that contemporary platforms actively suppress.

By contrast, opaque quota meters teach only anxiety. They present storage as a mysterious substance that fills unpredictably, reinforcing dependency rather than understanding.

Remark 7. *An interface that explains itself is a form of education. An interface that obscures its own operation is a form of control.*

8.5 Why Platforms Cannot Adopt This Model

It is tempting to ask why existing platforms do not simply implement entropy-aligned pricing. The answer is structural. Transparent pricing reveals that most stored content is cheap. Deduplication-aware accounting exposes extreme markups. Compensation for compression undermines centralized value extraction.

Most importantly, data portability becomes trivial in a content-addressable system. Users can export their histories without loss, and cooperative systems can interoperate. Lock-in evaporates.

For a profit-maximizing platform built on rent extraction, these properties are not features but existential threats. The absence of such models in consumer platforms is therefore not an accident but an equilibrium.

8.6 Relation to Event-Driven Architecture

PlenumHub aligns naturally with the Spherepop calculus, in which commits, merges, and collapses are treated as first-class operations over event histories. Storage is not an afterthought but the substrate on which cognition operates.

By making history cheap, branching lawful, and recombination routine, PlenumHub provides the economic foundation required for event-driven cognition to scale beyond individual systems into collective infrastructure.

The final sections of this paper formalize this relationship explicitly, introducing the Spherepop kernel, its operational semantics, and the correspondence between event-history preservation and architectural commutativity.

9 The Spherepop Kernel

The arguments developed thus far require a formal substrate capable of representing event histories as first-class objects. Such a substrate must support incremental commitment, lawful recombination, and controlled collapse, while remaining agnostic to surface syntax, interface conventions, or application-level semantics. The Spherepop kernel is proposed as this substrate.

The kernel is not an application, nor a user interface, nor a storage system per se. It is an abstract operational core that specifies how event histories are created, composed, reduced, and preserved. Its purpose is to make explicit the minimal commitments required for event-driven cognition to remain lawful under constraint.

9.1 Kernel-Level Commitments

At the kernel level, the primitive object is not a state but a history. A history is a partially ordered sequence of events, together with the constraints induced by their execution. Importantly, histories are not assumed to be linear. Branching and merging are intrinsic, not exceptional.

Definition 4 (Kernel History)**.** *A* kernel history *is a directed acyclic graph of events equipped with a partial order that respects causal dependency.*

Events in the kernel are irreversible. Once committed, an event constrains the space of future events that may lawfully occur. This irreversibility is not a limitation but a requirement: without it, histories collapse into undifferentiated states, and compositional reasoning becomes impossible.

9.2 Primitive Operations

The Spherepop kernel admits a small set of primitive operations. These are not chosen for expressive completeness but for architectural necessity. Any richer behavior must be constructed by composition rather than by expanding the primitive set.

At minimum, the kernel supports committing an event, merging histories, and collapsing histories.

A *commit* introduces a new event, extending a history by adding a node and updating constraints accordingly. A *merge* combines two compatible histories, producing a new history that preserves the event structure of both. A *collapse* deliberately reduces historical resolution, replacing a subgraph with an abstraction that preserves selected invariants while discarding detail.

Definition 5 (Commit)**.** *A* commit *is an operation that appends a new event to a history, producing a strictly more constrained history.*

Definition 6 (Merge)**.** *A* merge *is an operation that combines two histories with compatible constraints into a single history that preserves both event structures.*

Definition 7 (Collapse)**.** *A* collapse *is an operation that replaces a sub-history with an abstracted event that preserves specified invariants while discarding internal structure.*

These operations are intentionally asymmetric. Commit and merge preserve information; collapse destroys it. Collapse is therefore treated as a privileged operation whose use must be justified by explicit constraints.

9.3 Kernel Minimalism

The kernel deliberately excludes many features commonly associated with programming environments. There are no global variables, no mutable state in the traditional sense, and no implicit side effects. All observable effects arise from committed events.

This minimalism mirrors the design of successful infrastructural kernels in other domains. Operating system kernels do not implement applications; version control systems do not interpret content; Unix pipelines do not enforce semantics beyond input-output contracts. The power of such systems lies in their refusal to speculate.

Remark 8. *Kernel minimalism is not asceticism. It is an architectural defense against premature collapse.*

By restricting the kernel to event-history operations, Spherepop ensures that complex behavior emerges through composition rather than through ad hoc mechanism. This property is essential if independent subsystems are to be developed, reasoned about, and recombined without global coordination.

9.4 Relation to Compositional Systems

The Spherepop kernel generalizes patterns already present in successful computational systems. Version control systems treat commits as irreversible events and merges as structured recombination. Unix pipelines treat programs as functions whose composition is lawful only if interfaces are respected. Window managers such as Byobu expose session histories as manipulable objects rather than ephemeral processes.

What distinguishes the kernel is that these patterns are not treated as implementation details but as formal commitments. The kernel insists that systems either respect event history or fail explicitly.

This insistence clarifies why many contemporary interfaces feel constraining. By hiding history, suppressing branching, and privileging output over process, they violate kernel-level principles. The resulting systems appear simpler but are, in fact, cognitively impoverished.

9.5 Why Event-History Must Be First-Class

Treating history as a secondary concern leads to architectures that cannot explain their own behavior. Debugging becomes forensic rather than structural. Learning becomes brittle because the path by which a conclusion was reached is lost.

By contrast, a system that preserves event history can reason about itself. It can replay, branch, compare alternatives, and attribute outcomes to specific commitments. This capability is the foundation of intelligence understood as adaptive constraint satisfaction rather than as static pattern recognition.

Proposition 5. *Any system capable of robust generalization must preserve sufficient event-history structure to distinguish alternative construction paths.*

Proof. Generalization depends on identifying invariants across varying instances. Without access to construction history, instances collapse into undifferentiated states, eliminating the information required to identify invariants. □ □

The remaining task is to show that the Spherepop kernel is not merely a conceptual framework but admits precise operational semantics. The next section provides this formalization, including reduction traces and concrete mappings to existing tools such as Unix pipelines, Byobu sessions, and AutoHotkey automation.

10 Operational Semantics of the Spherepop Kernel

To move beyond metaphor, the Spherepop kernel requires an explicit operational semantics. This semantics specifies how histories evolve under kernel operations, how reductions occur, and under what conditions composition is lawful. The purpose is not to define a programming language, but to formalize the minimal execution model required for event-driven cognition.

The semantics is small-step and history-preserving. Each operation transforms a history into a new history, rather than transforming a state in isolation. Execution is therefore a trace over histories, not a sequence of states.

10.1 Configurations

A kernel configuration consists of a history and a frontier. The history records all committed events; the frontier identifies the active boundary at which new events may be committed.

Definition 8 (Kernel Configuration)**.** *A configuration is a pair (H, F) where H is a history graph and $F \subseteq H$ is the set of frontier nodes eligible for extension.*

Frontiers make branching explicit. Multiple frontier nodes indicate that several alternative continuations remain available. Collapse reduces the frontier by design.

10.2 Small-Step Semantics

Execution proceeds by applying one of the kernel operations to a configuration, producing a new configuration. We write:

$$(H, F) \longrightarrow (H', F')$$

10.2.1 Commit Rule

A commit introduces a new event e extending a frontier node $f \in F$.

$$\frac{f \in F \quad e \notin H}{(H, F) \longrightarrow (H \cup \{e\}, (F \setminus \{f\}) \cup \{e\})}$$

The effect is to replace the chosen frontier node with the newly committed event. This models irreversible extension: once committed, the new event becomes the active continuation.

10.2.2 Merge Rule

A merge combines two histories that share a common prefix and whose constraints are compatible.

$$\frac{H_1 \sim H_2}{(H_1 \cup H_2, F_1 \cup F_2) \longrightarrow (H_m, F_m)}$$

Compatibility requires that no event in one history violates constraints induced by events in the other. The merged history preserves both lineages rather than selecting one.

10.2.3 Collapse Rule

Collapse replaces a subgraph of history with an abstracted event c that preserves selected invariants.

$$\frac{S \subseteq H}{(H, F) \longrightarrow ((H \setminus S) \cup \{c\}, F')}$$

Where F' updates the frontier to reference c instead of elements of S. Collapse is irreversible and lossy by definition.

Remark 9. *Collapse is the only kernel operation that destroys information. Its use must therefore be justified by an explicit constraint external to the kernel, such as resource limits or decision deadlines.*

10.3 Reduction Traces

A computation in the Spherepop kernel is a reduction trace:

$$(H_0, F_0) \longrightarrow (H_1, F_1) \longrightarrow \dots$$

Unlike traditional operational semantics, the trace itself is a first-class object. Reasoning about a computation involves reasoning about the entire trace, not merely its endpoint.

10.4 Unix Pipelines as Kernel Reductions

The Unix pipeline exemplifies lawful composition under minimal semantics. Each program in a pipeline is a function from a stream to a stream, and the pipe operator enforces that composition respects this contract.

From the kernel perspective, each program invocation is a commit event. The pipeline as a whole is a linear history whose legality depends on interface compatibility.

If a program violates the conventionâĂŤby writing non-stream output, depending on global state, or performing hidden side effectsâĂŤthe pipeline ceases to be composable. Injectivity is lost, and reduction fails.

Remark 10. *Unix pipelines work not because programs are simple, but because the architecture enforces a narrow, compositional event interface.*

This is precisely why the pipeline metaphor does not justify institutional hyper-specialization of human labor. Human cognition operates at higher levels of abstraction, where events are evaluated, not merely executed. The success of pipelines depends on deferred collapse: interpretation happens at the end of the chain, not at each stage.

10.5 Byobu Sessions as Event-History Traces

Byobu and similar terminal multiplexers make session history explicit. Windows, panes, command sequences, and scrollback are not ephemeral states but preserved events. Users may detach, reattach, branch workflows, and revisit prior contexts.

From the Spherepop perspective, a Byobu session is a living history graph. Each command is a commit. Splitting a pane creates a branch. Synchronizing panes is a merge. Clearing scrollback is a collapse.

Crucially, Byobu does not privilege output over history. It preserves context until the user explicitly collapses it. This property explains its enduring value among technical users: it aligns with event-driven cognition rather than fighting it.

Proposition 6. *Tools that preserve interaction history enable higher-order reasoning by supporting replay, branching, and recombination of events.*

Proof. Higher-order reasoning requires comparison of alternative constructions. Preserved histories permit such comparison; ephemeral state does not. □ □

10.6 Why Platforms Eliminate Reduction Traces

Contemporary platforms systematically eliminate reduction traces. Interfaces collapse history into snapshots, suppress branching, and externalize context. The justification is simplicity; the effect is control.

By eliminating traces, platforms prevent users from understanding how outcomes were produced. This makes systems harder to leave, harder to replicate, and harder to contest. Event-history suppression thus functions as both cognitive and economic enclosure.

11 Conclusion

This essay has argued that many of the most pervasive failures of contemporary computational platforms, institutions, and cognitive tools do not arise from insufficient optimization, but from a systematic misunderstanding of what intelligence actually requires. Intelligence, whether biological, computational, or collective, is not a property of isolated states or outputs. It is a property of event histories: of how systems are constructed, constrained, revised, and recomposed over time.

When event history is treated as secondaryâĂŤwhen it is collapsed prematurely, taxed economically, or obscured architecturallyâĂŤsystems may appear simpler or more efficient in the short term, but they become brittle, opaque, and increasingly incapable of adaptation. This brittleness manifests at every scale. Individuals lose the ability to reason longitudinally. Software systems become unmaintainable. Institutions harden into hierarchies that mistake control for coordination.

The analysis began by reframing cognition as fundamentally event-driven rather than state-based. From this perspective, memory is not storage of content but preservation of constraint. Abstraction is not compression of data but lawful collapse of history under explicit justification. Architecture, in turn, is not a matter of local optimization, but of ensuring that decomposition and recomposition commuteâĂŤthat independently developed components can be integrated without loss of global coherence.

This reframing clarifies why certain technical architectures succeed. Unix pipelines, version control systems, and long-lived interactive environments work not because they are minimal, but because they preserve history up to the point where compatibility can be assessed. They defer collapse, enforce explicit interfaces, and make failure visible rather than implicit.

Against this backdrop, the behavior of contemporary platforms becomes legible. The imposition of artificial scarcity in cloud storage is not a technical necessity but an economic intervention that suppresses event-history preservation. By charging users for duplicated structure, incremental revision, and shared cultural artifacts, platforms force premature collapse of memory and externalize architectural simplification onto cognition itself. What is presented as convenience or sustainability is, in fact, a mechanism of control.

The Gmail trajectory exemplifies this dynamic with unusual clarity. An initial commitment to historical preservation and abundance was reversed only after dependency had been established,

revealing that scarcity was not discovered but introduced. The result is not merely overpricing, but a degradation of usersâĂŹ ability to treat their own histories as durable, revisitable, and recomposable objects of thought.

PlenumHub was introduced not as an idealized alternative, but as a proof of possibility. By aligning cost with informational entropy rather than visibility, and by treating compression as a collective good rather than a private revenue stream, it demonstrates that event-history preservation can be economically sustainable, technically straightforward, and educationally transparent. The fact that such models are absent from dominant platforms is therefore not an argument against their feasibility, but evidence of incompatible incentives.

At the architectural level, the Spherepop kernel was presented as a minimal formalization of these insights. Its insistence on commits, merges, and collapse as explicit operations is not a stylistic choice, but a structural necessity. Only systems that preserve enough history to test compatibility can support lawful composition. This requirement is not optional; it is the condition under which layered design, distributed work, and collective intelligence remain possible.

The broader implication is that many contemporary demands for simplification, specialization, and user passivity are not grounded in optimality, but in the convenience of managing systems that no longer remember how they were built. Low-level compositional success is mistakenly used to justify high-level fragmentation, even though the former depends precisely on deferred collapse and preserved context.

The choice facing designers, institutions, and users is therefore stark. Either we continue to accept architectures that profit from forgettingâĂŤtaxing memory, obscuring mechanisms, and enforcing premature decisionsâĂŤor we build systems that treat history as a first-class resource, align incentives with actual costs, and allow intelligence to emerge through lawful recomposition.

Event history is not a luxury. It is the substrate of reasoning, accountability, and freedom. Architectures that erase it do not merely simplify; they diminish. Architectures that preserve it make complexity navigable rather than unmanageable.

The future of intelligent systems will be determined not by how efficiently they produce outputs, but by how carefully they remember how they came to be.

A Appendix A: A Grammar for Spherepop Histories

This appendix presents a concrete grammar for the Spherepop kernel. The grammar is intentionally minimal. It does not attempt to encode application semantics, surface syntax, or domain-specific constructs. Its sole purpose is to specify the lawful structure of event histories and the primitive operations that act upon them.

The grammar is given in Backus–Naur Form (BNF) with informal annotations.

A.1 Lexical Elements

We assume a countable set of identifiers for events, histories, and invariants.

```
<id>        ::= [a-zA-Z_][a-zA-Z0-9_]*
<label>     ::= <id>
```

A.2 Core Constructs

A Spherepop program is a sequence of kernel operations acting on histories.

```
<program>   ::= <stmt> | <stmt> <program>
<stmt>      ::= <commit> | <merge> | <collapse>
```

A.3 Histories and Events

Histories are named collections of events. Events are opaque; their internal structure is irrelevant at the kernel level.

```
<history>   ::= <id>
<event>     ::= <id>
```

A.4 Commit Operation

A commit introduces a new event extending an existing history.

```
<commit>    ::= "commit" <event> "to" <history>
```

Semantically, this operation appends the event to the frontier of the specified history, producing a strictly more constrained history.

A.5 Merge Operation

A merge combines two compatible histories into a new history.

```
<merge>     ::= "merge" <history> <history> "as" <history>
```

Compatibility is a semantic condition, not enforced syntactically. If the histories contain incompatible constraints, the merge is undefined.

A.6 Collapse Operation

Collapse replaces a sub-history with an abstracted event preserving selected invariants.

```
<collapse>  ::= "collapse" <history> "preserving" <invariants>
<invariants>::= <label> | <label> "," <invariants>
```

The result of a collapse is a new history in which the specified invariants are preserved but internal structure is irreversibly discarded.

A.7 Remarks on Minimality

This grammar is deliberately sparse. There are no control structures, no variables, and no expression language. All complexity arises from composition of histories, not from syntax.

This reflects the architectural commitment of the Spherepop kernel: abstraction must emerge from event structure rather than from representational machinery.

Remark 11. *Any extension of this grammar should be conservative, preserving the kernelâĂŹs event-history semantics rather than introducing stateful shortcuts.*

B Appendix B: A Typed Variant of the Spherepop Kernel

The untyped grammar presented in Appendix A specifies the structural legality of Spherepop histories but leaves implicit several invariants required for safe composition. In practice, histories carry semantic commitments: constraints, interfaces, and guarantees that must be respected when histories are merged or collapsed.

This appendix introduces a typed variant of the Spherepop kernel. Types do not describe data values or representations. Instead, they describe admissible event histories and the constraints under which kernel operations are lawful.

B.1 History Types

Each history is associated with a type that encodes the invariants preserved by that history.

Definition 9 (History Type)**.** *A* history type τ *is a set of invariants that any realization of the history must satisfy.*

We write $H : \tau$ to denote that history H preserves invariants τ. Invariants may include interface contracts, causal guarantees, monotonicity properties, or domain-specific constraints.

Types are structural rather than nominal. Two histories share a type if they preserve the same invariants, regardless of their internal event structure.

B.2 Typing Judgments

Typing judgments have the form:

$$\Gamma \vdash H : \tau$$

where Γ is a typing context associating histories with types.

B.3 Typed Commit

A commit preserves the type of a history provided that the new event respects the existing invariants.

Theorem 1 (Commit Preservation)**.** *If* $\Gamma \vdash H : \tau$ *and event* e *respects invariants* τ*, then:*

$$\Gamma \vdash \texttt{commit}\ e\ \texttt{to}\ H : \tau$$

Proof. By definition, a commit extends a history without violating existing constraints. Since e respects τ, the extended history preserves τ. □ □

This rule enforces architectural discipline. Events that violate a historyâĂŹs invariants are rejected at commit time rather than producing latent failure.

B.4 Typed Merge

Merge is the most delicate kernel operation. Two histories may be merged only if their invariants are compatible.

Definition 10 (Type Compatibility)**.** *Two types τ_1 and τ_2 are* compatible *if there exists a type τ_m such that $\tau_m \subseteq \tau_1 \cap \tau_2$.*

Compatibility expresses the existence of a common refinement that preserves shared guarantees without contradiction.

Theorem 2 (Merge Typing)**.** *If $\Gamma \vdash H_1 : \tau_1$, $\Gamma \vdash H_2 : \tau_2$, and τ_1 and τ_2 are compatible, then:*

$$\Gamma \vdash \texttt{merge}\ H_1\ H_2\ \texttt{as}\ H_m : \tau_m$$

for some $\tau_m \subseteq \tau_1 \cap \tau_2$.

Proof. Merge preserves exactly those invariants shared by both histories. The merged history cannot preserve invariants present in only one input, nor can it violate invariants common to both. □ □

This rule formalizes why compositional architectures require narrow, explicit interfaces. Broad, implicit invariants reduce compatibility and make merging intractable.

B.5 Typed Collapse

Collapse is explicitly lossy. The typed kernel therefore treats collapse as a type-changing operation.

Definition 11 (Collapse Signature)**.** *A collapse has the form:*

$$\texttt{collapse} : \tau \to \tau'$$

where $\tau' \subset \tau$.

The post-collapse type τ' specifies which invariants are preserved. All others are discarded.

Theorem 3 (Collapse Safety)**.** *If $\Gamma \vdash H : \tau$ and $\tau' \subset \tau$, then:*

$$\Gamma \vdash \texttt{collapse}\ H\ \texttt{preserving}\ \tau' : \tau'$$

Proof. Collapse is defined precisely as the elimination of structure not required to preserve τ'. Since τ' is a subset of τ, preservation is well-defined. □ □

Typed collapse makes explicit what is usually implicit in system design: every abstraction discards information, and that loss must be accounted for.

B.6 Failure as First-Class Outcome

In the typed kernel, illegal operations do not silently coerce histories into valid states. Instead, they fail explicitly.

Attempting to commit an event that violates invariants, merge incompatible histories, or collapse without specifying preserved invariants results in rejection rather than implicit coercion.

Remark 12. *Explicit failure is an architectural virtue. Silent coercion is a form of unacknowledged collapse.*

B.7 Relation to Existing Systems

Typed Spherepop histories generalize familiar notions from software systems. Type signatures correspond to interface contracts in Unix pipelines. Merge compatibility mirrors version control constraints. Collapse signatures resemble abstraction barriers in programming languages.

What distinguishes the kernel is that these constraints are enforced at the level of event history rather than at the level of representation or execution state.

B.8 Typed Histories and Cognitive Guarantees

From a cognitive perspective, types encode what a system knows it will not violate. They define safe regions of reasoning. Typed histories permit modular thinking without global recomputation, enabling complex systems to scale while remaining intelligible.

The next appendix grounds these abstractions in concrete practice through a worked translation from AutoHotkey scripts to Spherepop histories, illustrating how everyday automation already conforms to kernel principles when viewed through an event-historical lens.

C Appendix C: A Worked Translation from AutoHotkey to Spherepop

This appendix demonstrates that the Spherepop kernel is not an abstract imposition on practice but a faithful formalization of workflows already employed by expert users. AutoHotkey (AHK) scripts, particularly when used as personal automation infrastructure, already instantiate event-driven, history-sensitive computation. What Spherepop provides is an explicit semantics for what AHK users implicitly rely upon.

The translation proceeds by interpreting AHK constructs as kernel-level events and histories, rather than as mere text substitution or imperative control flow.

C.1 AutoHotkey as Event Infrastructure

AutoHotkey scripts are typically written incrementally, accumulated over time, and rarely deleted. Hotstrings and hotkeys function as durable commitments: once defined, they alter the future space of possible actions. The script as a whole is therefore best understood as an event history rather than a program in the conventional sense.

Each binding represents a commitment that persists across sessions. Reloading the script does not reset cognition; it reactivates a previously committed history.

Remark 13. *AHK users do not "run programs" so much as inhabit evolving environments of commitments.*

C.2 Example: Batch Renaming Hotstrings

Consider the following AutoHotkey hotstring, representative of routine file system automation:

```
::nonew::
(
for file in new_*.png; do mv "$file" "${file/new_/}"; done
)
```

This construct introduces a durable transformation triggered by a textual event. From the Spherepop perspective, the definition itself is the primary event; individual executions are instantiations.

We translate this as follows.

C.2.1 Kernel Interpretation

Defining the hotstring corresponds to a commit:

$$\texttt{commit } e_{\texttt{nonew}} \texttt{ to } H_0$$

where $e_{\texttt{nonew}}$ encodes the invariant:

> Files prefixed with `new_` may be safely normalized by prefix removal.

Each invocation of the hotstring is not a new commit but an application of an existing event to new context. The history records the availability of the transformation, not each execution.

Remark 14. *This distinction explains why deleting and retyping commands is cognitively inferior to defining reusable transformations. Spherepop records the latter as history; the former evaporates.*

C.3 Example: Incremental Media Processing

Consider a more complex sequence:

```
::compresspdf::
convert wandering.pdf -compress jpeg -quality 40 wandering-creativity.pdf
```

This introduces a lossy transformation. In kernel terms, this is a collapse, not a commit, because it irreversibly discards information.

C.3.1 Typed Collapse

We represent this as:

$$\texttt{collapse } H_{\texttt{pdf}} \texttt{ preserving } \{\texttt{visual\_content}\}$$

The collapse explicitly preserves visual readability while discarding high-frequency detail. The typed kernel makes this loss explicit rather than implicit.

Remark 15. *Treating lossy transforms as collapse rather than commit prevents accidental destruction of high-resolution histories.*

C.4 Loops and Repetition as Single Events

AHK users often employ loops to apply a transformation across many artifacts:

```
::invrt::mogrify -negate *.png
```

Although this operates over many files, it is a single event at the kernel level: the commitment that color inversion is an acceptable operation for the selected domain.

This maps to:

$$\texttt{commit } e_{\texttt{invert\_images}} \texttt{ to } H$$

The kernel does not record each file mutation individually unless required. This reflects a crucial principle: events are defined by constraint, not by cardinality.

C.5 Interruptible Loops and Explicit Control

Consider the following AHK loop with an interrupt mechanism:

```
Loop
{
  If stop = 1
      Break
  Sleep, 500
}

^q::
If stop = 0
  stop = 1
Else
  stop = 0
```

This pattern introduces a control invariant: execution remains reversible until explicitly collapsed by interruption.

In Spherepop terms, this is a history with multiple active frontiers. The interrupt toggles collapse eligibility rather than forcing termination.

C.6 Translation Summary

The essential correspondences are as follows:

AutoHotkey Construct	**Spherepop Interpretation**
Hotstring definition	Commit (durable event)
Hotkey binding	Commit (interface invariant)
Script reload	History reactivation
Loop	Iterated application of single event
Lossy transform	Typed collapse
Interrupt	Frontier reduction

C.7 Why This Matters

AutoHotkey survives precisely because it preserves event history. Users accrete capabilities rather than overwrite them. Scripts become personal operating systemsâĂŤkernels of cognition adapted to individual constraint landscapes.

This explains the persistence of AHK in the Windows ecosystem despite repeated attempts to replace it with declarative automation tools. Those tools optimize for state snapshots; AHK optimizes for historical continuity.

Remark 16. *Windows survives as a viable expert platform largely because AutoHotkey restores event-driven agency that the native interface suppresses.*

The final appendix formalizes the correspondence between the Spherepop kernel and architectural commutativity, establishing that the kernel is not merely compatible with layered design, but is in fact its minimal formal expression.

D Appendix D: The Correspondence Theorem

This appendix formalizes a claim that has been implicit throughout the paper: that event-historyâĂŞpreserving systems are not merely compatible with good architecture, but constitute the minimal conditions under which architectural layering is possible at all.

The result clarifies why certain systems scale cognitively and institutionally, while others degrade into brittle hierarchies of control.

D.1 Statement of the Problem

Consider a system subject to real constraints: limited resources, bounded latency, and incomplete information. Such a system may be optimized in two conceptually distinct ways.

The first approach attempts to solve the full constrained optimization problem monolithically. The second decomposes the problem into layers, solves each layer locally, and composes the solutions.

In practice, the first approach is typically intractable. The second succeeds only when the decomposition is lawful. The question is therefore: under what conditions does layered optimization commute with global optimization?

D.2 Architectural Commutativity

We say that an architecture is commutative if solving the system as a whole and then decomposing yields the same result as decomposing first and solving each component independently.

Definition 12 (Architectural Commutativity)**.** *An architecture is* commutative *if the following diagram commutes:*

$$Solve_{global} \equiv Compose \circ Solve_{local}$$

where $Solve_{local}$ acts on decomposed subsystems and Compose recombines their results.

This notion appears implicitly in control theory, software engineering, and systems biology. It explains why layered designs can be optimized piecemeal without sacrificing global correctness.

D.3 Event-History as the Missing Condition

State-based formulations obscure a crucial requirement for commutativity. When systems are described purely by instantaneous states, decomposition destroys information about construction paths. Independent solutions cannot be reliably recombined because compatibility depends on history, not on final state alone.

Event-historyâĂŞpreserving systems avoid this failure mode.

Definition 13 (History-Preserving Decomposition)**.** *A decomposition is* history-preserving *if each subsystem retains enough event history to determine compatibility with other subsystems at composition time.*

This requirement explains why Unix pipelines, version control systems, and layered control architectures succeed: they preserve interface contracts and construction histories rather than collapsing them prematurely.

D.4 The Correspondence Theorem

We can now state the central result.

Theorem 4 (Correspondence Theorem)**.** *Architectural commutativity holds if and only if the system preserves event history up to the level required to test compatibility at composition time.*

Proof. (*Only if*) Assume architectural commutativity holds in a system that does not preserve event history. Then subsystems must be composable based solely on state. But state alone cannot encode construction-dependent constraints. Therefore, composition may succeed locally while violating global invariants, contradicting commutativity.

(*If*) Assume the system preserves sufficient event history. Local solutions retain their construction constraints. Composition can test compatibility explicitly via preserved invariants. Since no

information relevant to compatibility is lost, the composed solution matches the global solution. □

□

D.5 Interpretation

The theorem explains why layered systems fail when history is erased. When interfaces are underspecified, or when systems collapse histories into opaque states, independent development becomes impossible. Integration devolves into trial-and-error rather than principled composition.

Conversely, when histories are preserved, layers can evolve independently. Engineers, users, or subsystems can work locally, confident that their work will compose lawfully.

This is the sense in which event history is not an implementation detail but an architectural prerequisite.

D.6 Examples Across Domains

In Unix pipelines, each programâĂŹs adherence to input-output conventions is a preserved invariant. Violating these conventions breaks commutativity.

In version control, commits preserve construction history. Merges rely on this history to detect and resolve conflicts. Systems that squash history lose this capability.

In Byobu and similar environments, preserved session history allows workflows to be suspended, resumed, and recombined. Systems that discard context force users into fragile linear workflows.

In platform cloud storage, by contrast, history is monetized and suppressed. Users are forced to collapse event histories prematurely, destroying the conditions under which architectural commutativity could emerge.

D.7 Consequences for Human Institutions

A common error is to infer from the success of low-level compositional systems that specialization at the human or institutional level is inherently optimal. The theorem shows why this inference fails.

Low-level specialization works because abstraction boundaries preserve event history and defer collapse. Institutional specialization typically does the opposite: it collapses context early, forcing individuals into narrow roles without access to the histories required for recomposition.

The result is not efficiency but fragility.

D.8 Spherepop as Minimal Architecture

The Spherepop kernel satisfies the conditions of the Correspondence Theorem by construction. Commits preserve history. Merges test compatibility explicitly. Collapse is permitted only as a typed, acknowledged loss of information.

This minimality is not accidental. Any architecture that satisfies the theorem must, implicitly or explicitly, implement the same structure.

Remark 17. *Spherepop does not invent architectural commutativity; it exposes it.*

D.9 Conclusion

The suppression of event history is not merely a technical inconvenience. It is the root cause of architectural failure across computational, cognitive, and institutional systems.

Platforms that tax memory, obscure infrastructure, and enforce premature collapse do not merely extract rent. They destroy the conditions under which lawful composition, genuine learning, and collective intelligence can occur.

By contrast, systems that preserve history, expose constraints, and align economics with entropy enable both individual agency and scalable cooperation.

The choice is therefore not between simplicity and complexity, but between architectures that remember how they were built and those that insist on forgetting.

References

[1] John C. Doyle. *Toward a Theory of Control Architecture*. Unpublished manuscript / lecture transcript.