Event–Historical Aggregation: Map–Reduce as Commitment in Spherepop

Flyxion

Abstract

This essay develops an event–historical semantics for distributed aggregation, recasting the classical map–reduce paradigm as a process of irreversible commitment rather than value computation. Working within the Spherepop calculus, computation is understood as a history of authorized, replayable transformations whose meaning lies in the constraints they impose on future possibilities. Mapping corresponds to the construction of local event histories that culminate in committed summaries, while reduction is realized as a sequence of MERGE events that irreversibly bind these local commitments into a durable global object.

The paper introduces a formal semantics for event–historical aggregation, derives algebraic correctness properties for provenance–guarded reducers, and shows how associativity, commutativity, and idempotence emerge only after quotienting histories by authorized collapse. Refusal is treated as a first–class semantic operator, rendering invalid aggregations ontologically inadmissible rather than merely erroneous. The resulting framework subsumes classical map–reduce and CRDT semantics as special cases while extending them with auditability, policy enforcement, and controlled forgetting.

1 Introduction

Map–reduce occupies a foundational role in distributed systems. Its enduring appeal lies in a simple abstraction: large computations can be decomposed into local mappings over shards of data and a reduction that aggregates partial results. Correctness, scalability, and parallelism are achieved by requiring the reducer to satisfy algebraic properties such as associativity and, in many settings, commutativity and idempotence.

Despite its practical success, the classical formulation of map–reduce is conceptually thin. Aggregation is treated as a function from values to values, and the identity of the result is exhausted by its final numerical or symbolic content. Provenance, policy constraints, and correctness conditions beyond pure algebra are relegated to informal assumptions or external mechanisms.

This paper argues that this value–centric framing obscures what aggregation actually does in practice. Real distributed aggregation is not merely the calculation of a number; it is the construction of a durable commitment about which inputs have been accepted, which combinations are permitted, and which alternatives have been ruled out. These commitments persist across time, shape future computation, and often carry normative or policy significance.

Spherepop provides a calculus in which such commitments are explicit. In Spherepop, computation is event–sourced: objects are not defined by their current state but by the irreversible history of authorized events that brought them into being. Within this setting, map–reduce admits a natural and precise reinterpretation. The map phase constructs many local histories, each culminating in a committed summary. The reduce phase composes these histories through authorized merges, producing a global object whose identity is inseparable from the commitments encoded in its lineage.

The contribution of this essay is threefold. First, it provides a rigorous event–historical semantics for map–reduce in Spherepop. Second, it derives the algebraic correctness conditions of reducers as emergent properties of provenance–guarded merge operations rather than as primitive assumptions. Third, it demonstrates how refusal and controlled forgetting can be integrated directly into the semantics of aggregation, yielding a framework that is both more expressive and more faithful to real distributed practice.

2 Computation as Event History

Traditional programming models identify computation with the evaluation of functions over values. A computation is correct if it produces the right output, and the internal steps leading to that output are, at best, an implementation detail. In contrast, Spherepop treats computation as the construction of a history. An object exists if and only if there is an authorized sequence of events that introduces it, transforms it, and commits it as a fact of the world.

This event–historical stance entails a fundamental shift in semantics. The meaning of an object is not given by its extensional content alone, but by the constraints its history imposes on future computation. A committed object rules out alternative possibilities: once a POP event has occurred, the system may no longer behave as though the committed fact were absent or different.

Within this framework, aggregation cannot be understood as a mere operation on values. Aggregation constructs an object whose identity records which inputs have been bound together and under what conditions. The question "what value did we compute?" is secondary to the question "what commitments have we made?"

Map–reduce, reinterpreted in this light, is a disciplined method for constructing global commitments from local ones. Mapping produces locally committed facts; reduction binds those facts together into a larger commitment that persists as a world–bearing artifact.

3 Mapping as Local Commitment

The map phase in Spherepop begins with the introduction of input shards as authorized objects. Each shard is established as a legitimate participant in the computation through a commitment event. A mapper then performs some local analysis over the shard. This analysis may be pure and reversible at the level of ordinary computation, but its output becomes meaningful only when it is committed.

Formally, a mapper produces a summary object whose existence is fixed by a POP event. A subsequent LINK event binds the summary to its originating shard, establishing provenance as a

structural property of the object rather than as external metadata. Once these events have occurred, the mapper output cannot be silently rewritten or replaced without creating a new history.

The result of mapping is therefore a collection of objects that are independent in the sense that their histories share no reduction commitments, yet grounded in the sense that each carries an explicit lineage to the data that produced it. These objects form the raw material for reduction, not as interchangeable values, but as historically situated facts.

4 Event–Historical Semantics

To reason rigorously about aggregation in Spherepop, we now introduce a formal semantics in which computation is modeled as the construction and extension of event histories. The purpose of this semantics is not to reproduce classical denotational or operational accounts, but to make precise the sense in which aggregation derives its meaning from irreversible commitment rather than from terminal values.

4.1 Histories and Authorization

Let $\mathcal{E}$ denote a fixed set of primitive event types, including POP, LINK, MERGE, COLLAPSE, and REFUSE. A history is a finite, totally ordered sequence

$$H = (e_1, e_2, \ldots, e_n),$$

where each $e_i \in \mathcal{E}$. Histories are constructed incrementally: at each step, a candidate event may or may not be admissible, depending on the history constructed so far.

Authorization is taken as a primitive relation

$$\mathsf{auth}(H, e),$$

which holds when the event e is permitted as an extension of the history H. Authorization subsumes typing constraints, policy rules, domain invariants, and resource conditions. Crucially, authorization is evaluated with respect to the entire prefix history, not merely the current state of some object.

Definition 1 (Admissible Extension)**.** *Given a history H and an event e, the extension $H \cdot e$ is admissible if and only if* $\mathsf{auth}(H, e)$ *holds.*

If an event is not authorized, no successor history exists. This absence is not modeled as an error state, but as a lack of admissible continuation.

4.2 Replay Semantics

The meaning of a history is given by its replay. Let

$$H$$

denote the object induced by replaying the events of H in order. Replay is a partial function from histories to objects: it is defined only for admissible histories. Two histories that differ syntactically may nevertheless replay to objects that are semantically equivalent under collapse, as discussed below.

Objects are therefore intensional: their identity depends on the history that produced them, not merely on an extensional value.

Remark 1. *This contrasts with state–based semantics, where histories are erased once their effects are incorporated into a state. In Spherepop, history is never erased implicitly; it can only be transformed or quotienting via explicit events.*

4.3 Commitment and Monotonicity

Commitment is introduced through the POP event. Informally, a POP asserts that an object now exists as a fact of the world and may be referenced by future events.

Definition 2 (Commitment)**.** *An object x is committed at history H if H contains a* $\textsc{pop}(x)$ *event.*

Commitment is irreversible in the following precise sense.

Proposition 1 (Monotonicity of Commitment)**.** *If an object x is committed at history H, then for any admissible extension $H' \succeq H$, x remains committed in H'.*

Proof. By definition, commitment is witnessed by the presence of a $\textsc{pop}(x)$ event in the history. Since histories are extended only by appending events, no admissible extension can remove or negate a prior $\textsc{pop}(x)$. Therefore, once committed, x remains committed in all admissible future histories. □

This monotonicity property formalizes the intuitive notion that commitment rules out alternative futures in which the committed fact did not occur.

4.4 Refusal as Semantic Partiality

Refusal plays a central role in the semantics of aggregation. Rather than being an exceptional outcome within the system, refusal is modeled as the absence of an authorized extension.

Definition 3 (Refusal)**.** *Given a history H and a candidate event e, refusal occurs when* $\mathsf{auth}(H, e)$ *does not hold. In this case, no history $H \cdot e$ exists.*

In particular, a REFUSE annotation may be used to document the reason for inadmissibility, but it does not create a successor object. Refusal is therefore ontological rather than operational: the forbidden aggregation simply cannot happen.

Theorem 1 (No Recovery from Refusal)**.** *If an event e is refused at history H, then there exists no history H' extending H in which the effects of e are realized.*

Proof. By definition, refusal means that $H \cdot e$ is not admissible. Since all histories are constructed by admissible extensions, and no alternative event can realize the same effects without violating authorization, there exists no admissible history extending H in which the effects attributed to e occur. □

This theorem captures the distinction between refusal and error handling. An error may be corrected or compensated for in later computation. A refusal precludes the existence of any history in which the forbidden act took place.

4.5 Collapse and Equivalence of Histories

While histories are never implicitly erased, Spherepop permits explicit, authorized forgetting through COLLAPSE events. Collapse defines an equivalence relation on histories that preserves selected invariants while discarding others.

Definition 4 (Collapse Equivalence)**.** *Given a set of invariants I, a collapse operation induces an equivalence relation $\equiv_I$ on histories such that $H \equiv_I H'$ if and only if H and H' agree on all invariants in I.*

Collapse is irreversible and idempotent: once distinctions are forgotten, they cannot be recovered by further admissible extensions.

Proposition 2 (Idempotence of Collapse)**.** *If H collapses to H' under invariants I, then collapsing H' again under I yields H'.*

Proof. By definition, H' already satisfies the equivalence constraints induced by I. A second collapse under the same invariants cannot identify any further distinctions and therefore leaves H' unchanged. □

This completes the core semantic framework. With histories, authorization, commitment, refusal, and collapse formally defined, we can now turn to the semantics of aggregation itself and derive the algebraic properties of reducers as theorems rather than assumptions.

5 Merge Semantics and Reducer Construction

With the event–historical framework in place, we now give a precise semantics for reduction by merge. The central claim of this section is that reducer correctness properties traditionally assumed as algebraic axioms arise, in Spherepop, as consequences of authorization, provenance constraints, and collapse.

5.1 Reducer Objects and Provenance

A reducer object is an object whose history includes one or more merge events. Semantically, a reducer object represents a commitment that binds together a collection of mapper outputs. Each reducer therefore carries both a payload and a provenance component.

Definition 5 (Reducer Summary)**.** *A reducer summary is a pair (v, P), where v is a semantic payload and P is a finite set of shard identifiers indicating provenance.*

The interpretation of the payload v is domain–specific. For example, in a word–count application it may be a frequency table, while in numerical aggregation it may be a sum or vector. Provenance, by contrast, has a fixed semantic role: it constrains admissible composition.

5.2 Merge as an Authorized Extension

A merge is an event that takes two existing objects—typically a current reducer and a mapper output—and produces a new reducer object whose history extends the histories of its inputs.

Definition 6 (Merge Authorization)**.** *Let (v_1, P_1) and (v_2, P_2) be reducer summaries. The merge event* MERGE$((v_1, P_1), (v_2, P_2))$ *is authorized if and only if $P_1 \cap P_2 = \varnothing$.*

When authorized, the merge produces a new summary

$$(v_1, P_1) \oplus (v_2, P_2) = (v_1 \diamond v_2, P_1 \cup P_2),$$

where $\diamond$ is a binary operation on payloads. When the authorization condition fails, the merge is refused and no successor history exists.

Remark 2. *The authorization condition enforces a strong correctness guarantee: each shard may contribute at most once to a reducer history. Duplication is therefore not handled by idempotence at the value level, but excluded at the level of admissible histories.*

5.3 Monotonicity and Irreversibility of Reduction

Reduction by merge is monotone in a precise sense: each authorized merge strictly extends the set of commitments encoded in the reducer history.

Proposition 3 (Monotonic Growth of Provenance)**.** *Let r be a reducer summary with provenance P. If r' is obtained from r by an authorized merge with a mapper output whose provenance is Q, then the provenance of r' is $P \cup Q$, and $P \subsetneq P \cup Q$.*

Proof. By merge definition, $P \cap Q = \varnothing$ and the resulting provenance is their union. Since Q is nonempty for any nontrivial mapper output, the union is a strict superset of P. □

This strict growth property implies irreversibility.

Theorem 2 (Non–Reversibility of Merge)**.** *Let r' be obtained from r by an authorized merge. There exists no admissible history extending the history of r' in which the commitments introduced by that merge are undone.*

Proof. Undoing the merge would require either removing provenance elements from the summary or negating the effect of a prior POP event. Neither operation is authorized under the event–historical semantics, as histories may only be extended, not rewritten. Therefore no admissible extension can reverse the commitments introduced by the merge. □

5.4 Associativity up to Collapse

Classical reducers rely on associativity to permit reordering and parallel combination of inputs. In Spherepop, merge is not strictly associative at the level of histories, since different merge orders produce distinct event sequences. Associativity instead emerges at the level of collapsed objects.

Definition 7 (Associativity up to Collapse)**.** *Let m_1, m_2, m_3 be mapper outputs with pairwise disjoint provenance. Merge is associative up to collapse if the reducer objects obtained by*

$$((m_1 \oplus m_2) \oplus m_3) \quad \textit{and} \quad (m_1 \oplus (m_2 \oplus m_3))$$

are equivalent under an authorized collapse that preserves payload and provenance invariants.

Theorem 3 (Associativity of Provenance–Guarded Merge)**.** *If the payload operation $\diamond$ is associative and commutative, then merge is associative up to collapse.*

Proof. Both merge trees produce reducer summaries with identical payload $v_1 \diamond v_2 \diamond v_3$ and identical provenance $P_1 \cup P_2 \cup P_3$. The only difference lies in the internal structure of their histories. A collapse that preserves payload and provenance invariants identifies these histories, yielding equivalent reducer objects. □

This result justifies parallel and incremental reduction without erasing the historical information that distinguishes different execution paths.

5.5 Commutativity and Structural Symmetry

A similar argument establishes commutativity up to collapse.

Proposition 4 (Commutativity up to Collapse)**.** *If $\diamond$ is commutative, then for any two mapper outputs m_1 and m_2 with disjoint provenance, the reducer objects obtained by merging m_1 into m_2 and m_2 into m_1 are equivalent under collapse.*

Proof. The two merge orders produce summaries with identical payload and provenance. As in the associative case, a collapse preserving these invariants identifies the resulting histories. □

5.6 Idempotence via Refusal

Idempotence is often required to tolerate duplicate inputs. In Spherepop, this property arises from refusal rather than from algebraic absorption.

Theorem 4 (Idempotence by Inadmissibility)**.** *Let m be a mapper output with provenance P. Attempting to merge m into a reducer whose provenance already includes P is refused. Consequently, each mapper output can contribute to a reducer at most once.*

Proof. If $P \subseteq P_r$, then $P \cap P_r \neq \varnothing$, violating the merge authorization condition. The merge is therefore inadmissible, and no successor history exists in which the duplicate contribution occurs. □

This completes the derivation of the classical algebraic properties of reducers as consequences of event–historical semantics rather than as primitive axioms. Reduction is thus revealed as a process of constructing increasingly constrained objects through authorized composition.

6 Incremental and Streaming Reduction

A salient advantage of event–historical aggregation is that it supports incremental and streaming computation without additional machinery. Because each merge yields a new committed reducer object rather than mutating an existing one, aggregation naturally unfolds as a monotone sequence of commitments indexed by time.

Let $\{m_i\}_{i\geq 1}$ be a potentially unbounded stream of mapper outputs with pairwise disjoint provenance. An incremental reducer constructs a sequence of reducer objects $\{r_i\}_{i\geq 0}$ such that r_0 is an explicitly committed empty summary and

$$r_{i+1} = r_i \oplus m_{i+1}$$

whenever the merge is authorized. Each r_i is a distinct object with its own history, and the transition from r_i to r_{i+1} is marked by a MERGE followed by a POP.

Proposition 5 (Prefix Soundness)**.** *For any $i < j$, the reducer object r_i represents a sound aggregation of the prefix $\{m_1, \ldots, m_i\}$ of the input stream.*

Proof. By construction, r_i is obtained by a sequence of authorized merges starting from r_0 and incorporating exactly the mapper outputs m_1 through m_i. Authorization ensures that no mapper output outside this prefix can be included, and monotonicity of commitment ensures that no included output can be removed. □

This prefix property implies that at any moment the reducer exposes a coherent, auditable snapshot of the aggregation so far, without requiring global synchronization or quiescence.

7 Checkpointing and Rollback as Event Construction

Checkpointing in Spherepop does not require copying state or persisting mutable variables. To checkpoint a computation is simply to name a particular reducer object and record its identity. Because reducer objects are immutable once committed, a checkpoint is a stable reference to a specific history.

Rollback, correspondingly, is not a reversal of time. It is the construction of a new branch of history that resumes aggregation from an earlier checkpoint. Formally, let r_k be a previously committed reducer object. A rollback resumes computation by using r_k as the base reducer for subsequent merges, thereby producing a new sequence $\{r'_i\}$ whose histories extend that of r_k but not that of later reducer objects.

Theorem 5 (Rollback without Reversal)**.** *Rollback does not violate irreversibility: no committed reducer object is destroyed or undone by resuming from an earlier checkpoint.*

Proof. Rollback constructs new histories that extend an earlier committed history. Later histories remain intact and committed, but are no longer extended along the new branch. Since no history is shortened or rewritten, irreversibility is preserved. □

This treatment of rollback aligns aggregation with versioned data structures and persistent computation, while retaining explicit causal semantics.

8 Controlled Forgetting via Collapse

While histories are semantically significant, unbounded growth of reducer histories is neither necessary nor desirable. Spherepop addresses this through explicit collapse events that authorize forgetting under specified invariants.

A collapse operation takes a reducer object with history H and produces a new object with history H' such that $H \equiv_I H'$ for a declared set of invariants I. Typical invariants include payload totals and provenance sets, while internal merge structure is often dispensable.

Proposition 6 (Safety of Collapse)**.** *If a collapse preserves payload and provenance invariants, then any property of the reducer that depends only on these invariants is preserved under collapse.*

Proof. By definition of collapse equivalence, the replay of H and H' agree on all invariants in I. Any property definable solely in terms of these invariants therefore has the same truth value for both objects. □

Collapse is itself an irreversible commitment. Once a history has been collapsed, the distinctions it discards cannot be recovered by further admissible extensions.

Theorem 6 (Irreversibility of Forgetting)**.** *There exists no admissible history extending a collapsed history that restores discarded distinctions.*

Proof. Restoring discarded distinctions would require introducing information not derivable from the preserved invariants. Since admissible extensions may only operate on existing committed objects and events, and collapse explicitly forgets those distinctions, no authorized event can reconstruct them. □

This explicit treatment of forgetting distinguishes Spherepop from systems in which optimization silently erases causal structure.

9 Auditability and Lineage Guarantees

A principal motivation for event–historical aggregation is auditability. Because reducer objects retain their histories unless explicitly collapsed, it is always possible to reconstruct the lineage of an aggregate.

Theorem 7 (Lineage Completeness)**.** *Let r be a reducer object whose history has not been collapsed with respect to provenance. Then the set of mapper outputs contributing to r is uniquely recoverable from its history.*

Proof. Each authorized merge introduces explicit links from the new reducer object to both its prior reducer and the mapper output being incorporated. By traversing these links recursively, one

can enumerate all mapper outputs contributing to r. Authorization ensures that no mapper output appears more than once. □

Even after collapse, auditability can be preserved at a coarser granularity by retaining provenance invariants. The choice of invariants thus determines the resolution at which auditability is maintained.

Taken together, incremental reduction, checkpointing, rollback, and controlled forgetting demonstrate that Spherepop supports the practical requirements of distributed aggregation while preserving a principled semantic account of commitment and history.

10 Worked Examples of Event–Historical Aggregation

This section presents concrete instances of map–reduce expressed in the event–historical semantics of Spherepop. The goal is to demonstrate that common distributed aggregation tasks arise naturally from the merge calculus, while exhibiting properties—such as refusal and explicit provenance—that are difficult to express in value–centric models.

10.1 Word Frequency Aggregation

Consider a corpus partitioned into shards $\{s_i\}$. Each shard contains a finite sequence of tokens. A mapper processes shard s_i to produce a local frequency map $h_i : \Sigma \to \mathbb{N}$ together with provenance $P_i = \{i\}$.

The mapper output is committed via a POP event and linked to its shard. The reducer aggregates these outputs using merge. The payload operation $\diamond$ is pointwise addition of frequency maps, which is associative and commutative.

Theorem 8 (Correctness of Word Count Aggregation)**.** *Let r_n be the reducer object obtained by merging mapper outputs $m_1, \ldots, m_n$ with pairwise disjoint provenance. Then the payload of r_n assigns to each word $w \in \Sigma$ the total number of occurrences of w across shards $s_1, \ldots, s_n$.*

Proof. By induction on n. The base case $n = 0$ holds trivially for the empty reducer. Assume correctness for r_n. Merging m_{n+1} adds h_{n+1} to the payload via $\diamond$, yielding correct totals over all $n + 1$ shards. Provenance guarantees that no shard is counted twice. □

Attempting to re–merge a mapper output whose shard has already contributed is refused, preventing duplication at the semantic level.

10.2 Group–By Aggregation

Group–by operations can be expressed by lifting reducers to keyed families. A mapper emits pairs (k, m), where k is a key and m a mapper output for that key. The reducer maintains a map from keys to reducer objects, each governed by its own provenance constraints.

Proposition 7 (Independent Key Aggregation)**.** *Reducer histories for distinct keys are independent, and merges for one key do not affect the admissibility of merges for another.*

Proof. Provenance sets are disjoint across keys by construction, and merge authorization is checked per reducer object. Therefore histories evolve independently. □

This decomposition preserves parallelism while maintaining event–historical auditability per key.

11 Relation to CRDT Semantics

Conflict–free replicated data types (CRDTs) provide a widely adopted framework for distributed aggregation without coordination. At first glance, Spherepop reducers appear similar, as both rely on associative and commutative combination rules. However, the semantic commitments of the two frameworks differ fundamentally.

CRDT semantics are state–based or operation–based but ultimately value–centric. Correctness is expressed as eventual convergence: replicas that receive the same set of updates, possibly in different orders, converge to the same state. The causal history of updates is instrumental rather than meaningful.

Spherepop, by contrast, treats histories as first–class semantic objects. Merge is an irreversible commitment that constructs a new object whose identity records which inputs were accepted and how. Convergence, when desired, is realized via explicit collapse rather than implicit erasure of history.

11.1 Embedding CRDTs into Spherepop

Despite these differences, CRDTs can be embedded into Spherepop under suitable restrictions.

Theorem 9 (CRDT Embedding)**.** *Any state–based CRDT whose join operation is associative, commutative, and idempotent can be represented as a Spherepop reducer by imposing the following constraints: refusal rules are disabled, provenance is ignored or collapsed immediately, and collapse is applied eagerly after each merge.*

Proof. Under these constraints, each merge produces a reducer object that immediately collapses to a canonical representative of its payload state. Since collapse erases history, only the join–semilattice structure remains. Associativity, commutativity, and idempotence of the join operation ensure convergence equivalent to that of the original CRDT. □

This embedding shows that Spherepop strictly generalizes CRDT semantics.

11.2 Limits of CRDT Expressiveness

The converse embedding does not hold. CRDTs cannot express refusal, policy constraints that render certain merges inadmissible, or explicit forgetting as a committed act.

Theorem 10 (Strict Extension)**.** *There exist Spherepop reducers that cannot be faithfully represented as CRDTs.*

Proof. Consider a reducer with provenance–guarded merge that refuses duplicate shard contributions. CRDT idempotence absorbs duplicates silently, whereas Spherepop renders them inadmissible. No CRDT state transition can distinguish refusal from successful idempotent merge, so no faithful representation exists. □

This result highlights the semantic gain of event–historical aggregation: it allows distributed computation to encode normative and policy constraints directly in its semantics.

12 Attention as Aggregation on Graphs

The map–reduce interpretation developed thus far extends beyond classical batch aggregation. In this section, we show that the core mechanism of an attention head, as used in transformer architectures, can be reconstructed as a particular form of aggregation over a graph. This reconstruction clarifies the structural role of attention and prepares the ground for an event–historical reinterpretation.

Let $G = (V, E)$ be a directed graph whose nodes represent tokens, entities, or states, and whose edges represent potential interactions. Each node $i \in V$ carries an embedding vector $x_i \in \mathbb{R}^d$. A graph neural network (GNN) updates node representations by aggregating information from neighbors. In its most general form, a GNN layer computes

$$h_i = \phi\left(x_i, \bigoplus_{j \in \mathcal{N}(i)} \psi(x_i, x_j, e_{ij})\right),$$

where $\mathcal{N}(i)$ denotes the neighborhood of i, ψ computes a message from neighbor j to i, $\oplus$ is an aggregation operator, and ϕ produces the updated representation.

Attention mechanisms instantiate a particular choice of ψ and $\oplus$ in which the contribution of each neighbor is weighted by a learned relevance score. The critical observation is that attention is not fundamentally a sequence operation; it is a structured aggregation over a graph whose edges are dynamically reweighted.

13 Constructing an Attention Head on a Graph

An attention head may be defined on a graph by introducing three linear maps:

$$Q : \mathbb{R}^d \to \mathbb{R}^{d_k}, \quad K : \mathbb{R}^d \to \mathbb{R}^{d_k}, \quad V : \mathbb{R}^d \to \mathbb{R}^{d_v}.$$

For each node i, the query, key, and value vectors are given by

$$q_i = Qx_i, \quad k_i = Kx_i, \quad v_i = Vx_i.$$

Given a directed edge $(i, j) \in E$, the compatibility between i and j is computed as

$$\alpha_{ij} = \frac{\exp(\langle q_i, k_j \rangle)}{\sum_{k \in \mathcal{N}(i)} \exp(\langle q_i, k_k \rangle)}.$$

The updated node representation is then

$$h_i = \sum_{j \in \mathcal{N}(i)} \alpha_{ij} v_j.$$

This computation is a weighted reduction over the neighborhood of i. The softmax normalization enforces a simplex constraint, ensuring that attention weights sum to one. From an algebraic perspective, attention is therefore a convex aggregation of neighbor values, parameterized by learned compatibility functions.

Remark 3. *Nothing in this construction depends essentially on linear order. Transformers can be understood as GNNs operating on complete graphs with positional or causal masking.*

14 Event–Historical Interpretation of Attention

The standard attention formulation is value–centric. It computes a weighted sum and discards the intermediate relevance structure once the output vector is produced. From an event–historical perspective, this erasure is semantically significant.

We reinterpret attention in Spherepop by treating each attention head as a reducer operating over a graph–indexed collection of mapper outputs. Each node j produces a mapper output consisting of its value vector v_j together with provenance identifying the source node. The attention head at node i performs a reduction over these mapper outputs.

The attention weights α_{ij} are not merely coefficients but encode a policy of admissible influence. In Spherepop terms, they define which merges are authorized and with what relative strength. A sharp attention distribution corresponds to a near–exclusive commitment to a small subset of inputs, while a diffuse distribution corresponds to a weaker, more distributed commitment.

Definition 8 (Attention as Weighted Merge)**.** *An attention head induces a merge operation in which the payload combination operator $\diamond$ is a weighted sum, and authorization is governed by the attention mask and normalization constraints.*

Under this interpretation, computing h_i corresponds to constructing a reducer object whose history records which neighbors were attended to and under what weights. Standard transformers immediately collapse this history, retaining only the final vector. Spherepop makes this collapse explicit and optional.

15 Attention, Refusal, and Masking

Attention masks, commonly used to enforce causality or sparsity, acquire a natural interpretation as refusal rules. If an edge (i, j) is masked, then the corresponding merge is inadmissible: node j cannot contribute to the reducer history of node i.

Proposition 8 (Masking as Refusal)**.** *Let (i, j) be a masked edge. Then any attempted merge incorporating v_j into the reducer at i is refused.*

Proof. Masking enforces $\alpha_{ij} = 0$ by construction. In event–historical terms, this corresponds to the absence of authorization for the merge event involving j. No admissible history exists in which j contributes to i. □

This reframing highlights a limitation of standard attention: refusal is encoded numerically rather than structurally. A masked edge contributes a zero vector, but the system cannot distinguish between principled exclusion and accidental irrelevance.

16 Multi–Head Attention as Parallel Reducers

Multi–head attention decomposes aggregation into multiple parallel reducers, each operating with its own query, key, and value projections. In Spherepop terms, each head constructs a distinct reducer history over the same set of mapper outputs.

Proposition 9 (Independence of Attention Heads)**.** *Reducer histories corresponding to distinct attention heads are independent and may be collapsed or audited separately.*

Proof. Each head uses a distinct payload operation determined by its projections (Q_h, K_h, V_h). Since no merge events are shared across heads, their histories are disjoint and evolve independently. □

The concatenation and projection step following multi–head attention may be understood as a higher–level merge that binds these parallel reducer objects into a single representation, again followed by collapse in standard practice.

17 Limits of Value–Centric Attention

From the event–historical viewpoint, standard attention mechanisms systematically erase the very information that makes aggregation meaningful: which inputs were considered, which were excluded, and how strongly each influenced the result. All such structure is collapsed immediately into a vector.

This explains a number of empirical phenomena observed in transformer models, including sensitivity to prompt phrasing, instability under long–horizon composition, and difficulty enforcing hard constraints. These are not merely optimization issues, but consequences of treating attention as a value computation rather than as a commitment–forming process.

Spherepop suggests an alternative research direction: attention mechanisms that retain, at least partially, their event histories, enabling explicit refusal, auditable influence, and controlled forgetting.

18 A Categorical Semantics of Event–Historical Aggregation

The event–historical semantics developed in the preceding sections admits a natural categorical formulation that clarifies the structural commitments of Spherepop and unifies map–reduce, streaming aggregation, and attention under a single formal lens. This categorical perspective does not replace

the event–level semantics; rather, it abstracts from it while preserving the essential asymmetries introduced by commitment, refusal, and collapse.

18.1 The Category of Event Histories

We define a category $\mathcal{H}$ whose objects are event histories modulo authorized collapse. Two histories are identified when they are equivalent under a declared set of invariants. Morphisms in $\mathcal{H}$ are authorized extensions of histories: a morphism $H \to H'$ exists if and only if H' can be constructed from H by a finite sequence of authorized events.

Identity morphisms correspond to empty extensions, and composition is given by concatenation of event sequences. Because authorization is history–dependent, composition in $\mathcal{H}$ is partial: not all pairs of morphisms are composable.

Remark 4. *This partiality is essential. It encodes refusal as the absence of a morphism, rather than as a morphism to an error object. Invalid aggregations therefore do not merely fail; they are unrepresentable.*

18.2 Merge as a Partial Monoidal Structure

Merge induces a monoidal structure on $\mathcal{H}$, but one that is partial and policy–dependent. When two objects H_1 and H_2 have compatible provenance, their tensor product $H_1 \otimes H_2$ is defined as the history obtained by merging them. The empty reducer object serves as the monoidal unit.

Associativity and symmetry of this tensor hold only up to collapse. That is, $\mathcal{H}$ is not a strict monoidal category but a monoidal category modulo a quotient functor induced by collapse. This precisely mirrors the earlier result that reducer correctness properties hold only after identifying histories that preserve the same invariants.

18.3 Refusal as Non–Existence of Morphisms

In this categorical setting, refusal corresponds to the non–existence of a morphism. If merging two objects violates authorization constraints, then no morphism exists that would compose them. This design choice has important consequences. It ensures that invalid constructions cannot be accidentally composed with subsequent operations, and it renders correctness a structural property rather than a runtime condition.

18.4 Collapse as a Quotient Functor

Collapse is formalized as a functor

$$C_I : \mathcal{H} \to \mathcal{H}_I$$

that maps histories to equivalence classes preserving a chosen set of invariants I. This functor is idempotent and non–invertible. Applying collapse forgets distinctions permanently, but the fact that forgetting occurred remains visible in the history as a committed event.

Different choices of I yield different quotient categories, corresponding to different trade–offs between fidelity and tractability.

19 Map–Reduce and Attention as Monoidal Folding

From the categorical perspective, map–reduce is a fold over a family of objects in $\mathcal{H}$. Mapping produces a collection of independent objects, while reduction folds them using the monoidal structure induced by merge. Unlike classical folds, this folding operation is constrained by authorization and may be partial.

Attention mechanisms fit naturally into this picture. An attention head is a parameterized fold over a graph–indexed family of objects, where the fold weights are determined by learned compatibility functions. Standard transformers instantiate a fold followed immediately by collapse, erasing the event structure.

Multi–head attention corresponds to parallel folds, whose results are themselves merged at a higher level. This hierarchical folding structure mirrors the layered architecture of modern neural networks while exposing the points at which semantic commitments are made and then forgotten.

20 Conclusion and Future Directions

This essay has developed an event–historical account of distributed aggregation, recasting map–reduce as a process of irreversible commitment rather than as a function on values. Within the Spherepop calculus, mapping constructs local histories that culminate in committed summaries, while reduction composes these histories through authorized merges. Classical algebraic properties of reducers emerge not as axioms but as theorems, holding only up to authorized collapse.

By treating refusal as semantic partiality and collapse as explicit forgetting, the framework accommodates policy constraints, auditability, and controlled abstraction as first–class concerns. Streaming aggregation, checkpointing, and rollback arise naturally from the immutability of committed histories.

Extending the analysis to attention mechanisms reveals that attention is itself a form of aggregation whose semantic structure is obscured by value–centric implementations. Interpreting attention as event–historical folding exposes the cost of immediate collapse and suggests new directions for architectures that retain, reason over, and selectively forget their own commitments.

Several avenues for future work present themselves. One direction is the design of attention mechanisms and graph neural networks that preserve partial event histories to enable explicit refusal and auditability. Another is the integration of event–historical aggregation with learning dynamics, treating training itself as a sequence of commitments rather than as gradient descent in parameter space. Finally, the categorical framework developed here invites connections to traced monoidal categories, persistent data structures, and semantics of irreversible computation.

More broadly, event–historical aggregation offers a way to align distributed systems, machine learning architectures, and semantic computation around a shared notion of meaning as constraint. Aggregates are no longer ephemeral values, but durable objects whose identities encode how the world was allowed to become the way it is.

A Algebraic Properties of Provenance–Guarded Reducers

This appendix provides formal proofs of the algebraic properties attributed to Spherepop reducers in the main text. The objective is to show that associativity, commutativity, and idempotence arise as derived properties of event–historical constraints, rather than as axioms imposed on reducer functions.

A.1 Reducer Domain and Partiality

Let $\mathcal{S}$ denote a countable set of shard identifiers. A reducer summary is a pair (v, P) where $v \in V$ is a payload drawn from some carrier set V and $P \subseteq \mathcal{S}$ is a finite provenance set.

Let $\diamond : V \times V \to V$ be a binary operation on payloads.

Definition 9 (Admissible Merge)**.** *The merge operation $\oplus$ is a partial binary operation on reducer summaries defined by*

$$(v_1, P_1) \oplus (v_2, P_2)$$

if and only if $P_1 \cap P_2 = \varnothing$, in which case

$$(v_1, P_1) \oplus (v_2, P_2) := (v_1 \diamond v_2, P_1 \cup P_2).$$

If $P_1 \cap P_2 \neq \varnothing$, the merge is undefined.

Undefinedness corresponds exactly to refusal in the event–historical semantics.

A.2 Monotonicity and Strict Growth

Proposition 10 (Strict Provenance Growth)**.** *If $(v', P') = (v, P) \oplus (w, Q)$ is defined, then $P \subsetneq P'$.*

Proof. Since Q is nonempty for any mapper output and $P \cap Q = \varnothing$, we have $P' = P \cup Q$ with $P \subsetneq P'$. □

[No Cycles in Reducer Histories] Reducer histories are acyclic with respect to merge ancestry.

Proof. Each merge strictly increases provenance size. Since provenance sets are finite, no reducer can be its own ancestor. □

A.3 Associativity up to Collapse

Let $m_i = (v_i, P_i)$ for $i \in \{1, 2, 3\}$ with pairwise disjoint provenance.

Theorem 11 (Associativity up to Provenance–Preserving Equivalence)**.** *If $\diamond$ is associative, then*

$$(m_1 \oplus m_2) \oplus m_3 \quad \textit{and} \quad m_1 \oplus (m_2 \oplus m_3)$$

are both defined and equivalent under the equivalence relation

$$(v, P) \sim (v', P') \iff v = v' \ \wedge \ P = P'.$$

Proof. Both expressions are defined because provenance sets are pairwise disjoint. Associativity of $\diamond$ ensures

$$(v_1 \diamond v_2) \diamond v_3 = v_1 \diamond (v_2 \diamond v_3),$$

and set union is associative, yielding identical provenance $P_1 \cup P_2 \cup P_3$. □

Remark 5. *The histories producing these results differ syntactically, but collapse identifies them by quotienting over internal merge structure.*

A.4 Commutativity

Theorem 12 (Commutativity up to Collapse)**.** *If $\diamond$ is commutative, then for any two summaries with disjoint provenance,*

$$m_1 \oplus m_2 \sim m_2 \oplus m_1.$$

Proof. Commutativity of $\diamond$ implies $v_1 \diamond v_2 = v_2 \diamond v_1$, and set union is commutative. Provenance equality follows. □

A.5 Idempotence via Inadmissibility

Theorem 13 (Idempotence by Partiality)**.** *For any reducer summary $m = (v, P)$,*

$$m \oplus m$$

is undefined.

Proof. Since $P \cap P = P \neq \varnothing$, the admissibility condition fails. □

[Semantic Idempotence] Each mapper output contributes to a reducer at most once.

Remark 6. *Unlike algebraic idempotence, this property is enforced structurally. Duplicate inputs do not collapse to the same value; they are forbidden from occurring.*

A.6 Resulting Algebraic Structure

Theorem 14 (Reducer Quotient Structure)**.** *Let $\sim$ be the provenance–preserving equivalence relation. The quotient $(\mathcal{R}/\sim, \oplus)$ forms a commutative monoid.*

Proof. Associativity and commutativity hold up to $\sim$. The empty reducer $(v_0, \varnothing)$ serves as identity. Partiality disappears in the quotient because undefined merges correspond to excluded elements. □

Remark 7. *This monoid exists only after quotienting by collapse. At the level of histories, no total algebraic structure exists.*

B Embedding CRDT Semantics into Event–Historical Aggregation

This appendix formalizes the relationship between conflict–free replicated data types (CRDTs) and Spherepop reducers. The central results are twofold. First, CRDTs embed into Spherepop under specific semantic restrictions. Second, this embedding is strict: there exist Spherepop reducers that cannot be faithfully represented as CRDTs.

B.1 Preliminaries on CRDTs

We consider state–based CRDTs, as they admit the clearest algebraic characterization. A state–based CRDT is defined by a triple $(S, \sqcup, \leq)$ where $(S, \leq)$ is a join–semilattice and $\sqcup : S \times S \to S$ is the least upper bound operation.

Updates are monotone with respect to $\leq$, and replicas periodically exchange states, merging them via $\sqcup$. Correctness is expressed as convergence: replicas that have received the same set of updates converge to the same state, independent of order or duplication.

B.2 CRDTs as Collapsed Reducers

We now show that CRDT semantics arise as a degenerate case of Spherepop aggregation in which history is aggressively collapsed.

Definition 10 (CRDT–Compatible Spherepop Reducer)**.** *A Spherepop reducer is CRDT–compatible if it satisfies the following constraints:*

1. *The payload carrier V forms a join–semilattice.*
2. *The merge payload operation $\diamond$ coincides with $\sqcup$.*
3. *Provenance is either absent or collapsed immediately after each merge.*
4. *No refusal rules are enforced; all merges are admissible.*

Theorem 15 (CRDT Embedding)**.** *For every state–based CRDT $(S, \sqcup, \leq)$, there exists a CRDT–compatible Spherepop reducer whose collapsed semantics are equivalent to the CRDT's convergence semantics.*

Proof. Construct a Spherepop reducer with payload carrier $V = S$ and merge operation $\diamond = \sqcup$. Let each update be represented as a mapper output whose payload is the updated state. Because all merges are admissible and collapse is applied eagerly, reducer histories never retain internal structure. After each merge, the reducer collapses to the canonical join of all received states.

Associativity, commutativity, and idempotence of $\sqcup$ guarantee that the resulting payload is independent of merge order and duplication, matching CRDT convergence semantics. □

Remark 8. *Under this embedding, Spherepop contributes no additional expressive power. All event–historical structure is erased immediately, and aggregation reduces to value computation.*

B.3 Eager Collapse and Event Erasure

CRDT convergence may be characterized as a discipline of mandatory forgetting.

Proposition 11 (Convergence as Eager Collapse)**.** *CRDT convergence is equivalent to applying a collapse operation after every merge that preserves only the payload invariant.*

Proof. Since only the join–semilattice value matters for CRDT correctness, all other historical distinctions are semantically irrelevant. Eager collapse enforces this irrelevance by quotienting histories immediately. □

This observation clarifies the philosophical difference between the two frameworks: CRDTs treat history as an implementation detail, while Spherepop treats history as semantically primary.

B.4 Non–Embeddability of Refusal

We now establish that the CRDT embedding is strict.

Theorem 16 (Non–Embeddability of Refusal)**.** *There exists no faithful embedding of provenance–guarded Spherepop reducers into CRDT semantics.*

Proof. Consider a Spherepop reducer whose merge operation refuses to combine summaries with overlapping provenance. Suppose, for contradiction, that this reducer could be faithfully embedded into a CRDT. In CRDT semantics, duplicate updates are either absorbed by idempotence or merged redundantly without semantic effect.

In Spherepop, by contrast, duplicate provenance renders the merge inadmissible. There exists no state transition corresponding to refusal: CRDTs cannot distinguish between forbidden composition and harmless idempotent merge. Therefore no embedding can preserve the refusal semantics. □

[Policy Non–Expressibility] Any aggregation policy that forbids certain combinations of inputs is inexpressible in CRDT semantics.

B.5 Non–Monotone Policies

CRDTs fundamentally rely on monotonic growth of state. Spherepop does not.

Theorem 17 (Non–Monotone Constraints)**.** *There exist Spherepop reducers whose admissibility conditions are non–monotone with respect to payload growth, and which therefore cannot be represented by CRDTs.*

Proof. Consider a reducer that refuses merges once provenance size exceeds a fixed threshold, enforcing a capacity constraint. This refusal depends on accumulated history rather than on monotonic payload growth. No CRDT state transition can encode such a constraint without violating convergence. □

B.6 Summary

CRDTs appear within Spherepop as a special case characterized by eager collapse, absence of refusal, and exclusive concern with payload convergence. Spherepop strictly generalizes this model by allowing history to remain meaningful, enforcing policy through inadmissibility, and treating forgetting as an explicit event rather than an implicit consequence of convergence.

C Categorical Semantics of Event–Historical Aggregation

This appendix presents a categorical formulation of the event–historical semantics underlying Spherepop aggregation. The purpose is to make precise the structural claims made in the main text, namely that aggregation is a form of monoidal composition constrained by authorization, refusal, and collapse, and that attention mechanisms correspond to traced folds within this structure.

C.1 The Category of Histories

Let $\mathcal{H}_0$ be the collection of all admissible event histories. We define a category $\mathcal{H}$ as follows.

Definition 11 (Objects)**.** *Objects of $\mathcal{H}$ are equivalence classes of histories under a chosen collapse equivalence relation $\equiv_I$, where I is a fixed set of invariants.*

Definition 12 (Morphisms)**.** *Given objects $[H]$ and $[H']$, a morphism $[H] \to [H']$ exists if and only if there exists a representative history $H'' \in [H']$ such that H'' is an authorized extension of a representative of $[H]$.*

Identity morphisms are given by empty extensions. Composition is given by concatenation of event sequences when authorization permits.

Proposition 12 (Well–Definedness)**.** *Morphisms in $\mathcal{H}$ are well defined with respect to the equivalence relation $\equiv_I$.*

Proof. If $H \equiv_I H'$ and $H \preceq K$ is an authorized extension, then there exists a corresponding extension $H' \preceq K'$ such that $K \equiv_I K'$, since collapse preserves admissibility of future events that depend only on invariants in I. □

C.2 Partiality and Refusal

Theorem 18 (Partiality of Composition)**.** *$\mathcal{H}$ is in general a partial category: composition of morphisms is not total.*

Proof. Authorization constraints depend on full history. There exist morphisms $[H] \to [H']$ and $[H'] \to [H'']$ such that the corresponding concatenation of events violates authorization. In this case, no composite morphism exists. □

Remark 9. *Refusal corresponds categorically to the absence of a morphism, not to a morphism into a distinguished error object.*

C.3 Merge as a Partial Monoidal Product

We now show that merge induces a monoidal structure on $\mathcal{H}$.

Definition 13 (Tensor Product)**.** *For objects $[H_1]$ and $[H_2]$, define $[H_1] \otimes [H_2]$ to be the object represented by the history obtained by merging H_1 and H_2, if such a merge is authorized. If the merge is not authorized, the tensor is undefined.*

Theorem 19 (Partial Monoidal Structure)**.** *$(\mathcal{H}, \otimes)$ is a partial monoidal category with unit object $[H_\emptyset]$, the empty reducer history.*

Proof. The empty history acts as a unit since merging with it introduces no provenance. Associativity holds up to collapse by the associativity–up–to–equivalence results proven in Appendix A. Partiality follows from authorization constraints. □

[Symmetry up to Collapse] If merge payload operations are commutative, then $\mathcal{H}$ admits a symmetry isomorphism $[H_1] \otimes [H_2] \cong [H_2] \otimes [H_1]$ up to collapse.

C.4 Collapse as a Quotient Functor

Definition 14 (Collapse Functor)**.** *Let $\mathcal{H}_I$ denote the category of histories collapsed with respect to invariants I. The collapse functor*

$$C_I : \mathcal{H} \to \mathcal{H}_I$$

maps each object to its equivalence class under $\equiv_I$ and each morphism to the induced morphism between equivalence classes.

Theorem 20 (Idempotence and Non–Invertibility)**.** *C_I is idempotent and non–invertible.*

Proof. Idempotence follows because collapsing twice with respect to the same invariants introduces no further identifications. Non–invertibility follows because distinct histories may be identified under collapse, and no functor can recover the discarded distinctions. □

Remark 10. *Different choices of I yield different quotient categories, corresponding to different semantic resolutions.*

C.5 Attention as Traced Monoidal Folding

We now formalize attention mechanisms within this categorical setting.

Let $G = (V, E)$ be a directed graph. For each node $i \in V$, let $\{H_{j \to i}\}_{j \in \mathcal{N}(i)}$ be histories representing mapper outputs from neighbors.

Definition 15 (Attention Fold)**.** *An attention head at node i is a fold*

$$\mathsf{att}_i : \bigotimes_{j \in \mathcal{N}(i)} [H_{j \to i}] \to [H_i]$$

parameterized by compatibility weights, followed by an optional collapse.

Theorem 21 (Attention as Traced Monoidal Operation)**.** *Multi–layer attention networks correspond to traced monoidal structures in $\mathcal{H}$, where residual connections induce trace operators.*

Proof. Residual connections feed outputs back as inputs in subsequent layers. In a monoidal category, such feedback corresponds to a trace. Partiality ensures that only authorized feedback loops exist. □

[Collapse in Standard Transformers] Standard transformer architectures correspond to applying collapse immediately after each attention fold.

Proof. Transformer implementations retain only the resulting value vectors and discard all intermediate relevance structure. This is precisely eager collapse with respect to payload invariants. □

C.6 Summary

The categorical semantics developed here unify event–historical aggregation, map–reduce, and attention mechanisms within a single partial monoidal framework. Merge corresponds to tensoring, refusal to the absence of morphisms, and collapse to quotienting by invariant–preserving equivalence. Attention emerges as a traced fold that is ordinarily collapsed in practice, obscuring its underlying commitment structure.

D A BNF Grammar for the Spherepop Calculus

This appendix presents a formal grammar for Spherepop, expressed in Backus–Naur Form (BNF). The grammar defines the syntactic structure of event–historical programs independently of any particular execution engine. Its purpose is to make explicit which constructs are primitive and how histories are formed, extended, and constrained.

The grammar is intentionally minimal. All semantic force arises from event authorization, refusal, and replay, rather than from complex syntactic forms.

D.1 Lexical Domains

We assume the following disjoint lexical domains:

$\langle \textit{Identifier} \rangle$	object names, shard IDs, variables
$\langle \textit{EventName} \rangle$	POP, LINK, MERGE, COLLAPSE, REFUSE
$\langle \textit{Invariant} \rangle$	named invariants (e.g. payload, provenance)
$\langle \textit{Literal} \rangle$	numbers, symbols, structured values

D.2 Programs and Histories

A Spherepop program is a history specification: a sequence of event statements.

$$\langle \textit{Program} \rangle ::= \langle \textit{History} \rangle$$

$$\langle History \rangle ::= \langle Event \rangle$$
$$\quad | \ \langle History \rangle \ \langle Event \rangle$$

Histories are ordered and append–only. There is no syntactic construct for deletion or mutation of prior events.

D.3 Events

$$\langle Event \rangle ::= \langle PopEvent \rangle$$
$$\quad | \ \langle LinkEvent \rangle$$
$$\quad | \ \langle MergeEvent \rangle$$
$$\quad | \ \langle CollapseEvent \rangle$$
$$\quad | \ \langle RefuseEvent \rangle$$

Each event form corresponds to a semantic primitive defined in the main text.

D.4 Commitment Events

$$\langle PopEvent \rangle ::= \text{POP} \ \langle Identifier \rangle$$

A POP event introduces an object as a committed fact. The grammar does not enforce authorization; admissibility is a semantic property.

D.5 Provenance and Dependency

$$\langle LinkEvent \rangle ::= \text{LINK} \ \langle Identifier \rangle \ \texttt{->} \ \langle Identifier \rangle$$
$$\quad | \ \text{LINK} \ \langle Identifier \rangle \ \texttt{->} \ \langle Identifier \rangle \ \texttt{:} \ \langle Identifier \rangle$$

Links encode provenance, causal dependency, typing, or policy constraints. The optional label refines the semantic role of the link but does not affect syntax.

D.6 Merge Events

$$\langle MergeEvent \rangle ::= \text{MERGE} \ \langle Identifier \rangle \ \langle Identifier \rangle \ \texttt{->} \ \langle Identifier \rangle$$

A merge event binds two existing objects into a new object. The target identifier must be fresh. Authorization depends on provenance and policy rules external to the grammar.

D.7 Controlled Forgetting

$$\langle CollapseEvent \rangle ::= \text{COLLAPSE} \ \langle Identifier \rangle$$
$$\quad | \ \text{COLLAPSE} \ \langle Identifier \rangle \ \texttt{keep} \ \langle InvariantList \rangle$$

$$\langle InvariantList \rangle ::= \langle Invariant \rangle$$
$$\quad | \ \langle Invariant \rangle \ \texttt{,} \ \langle InvariantList \rangle$$

Collapse events specify which invariants must be preserved. All other distinctions are discarded irreversibly.

D.8 Refusal

$$\langle \textit{RefuseEvent} \rangle ::= \textsc{refuse}\ \langle \textit{Event} \rangle$$
$$\mid \textsc{refuse}\ \langle \textit{Event} \rangle\ \texttt{:}\ \langle \textit{Identifier} \rangle$$

A refusal documents an inadmissible event. Semantically, refusal prevents the existence of any successor history in which the refused event occurs.

D.9 Values and Literals (Optional Layer)

The core calculus is value–agnostic. When needed, literals may be introduced through external binding mechanisms:

$$\langle \textit{LiteralBinding} \rangle ::= \langle \textit{Identifier} \rangle\ \texttt{:=}\ \langle \textit{Literal} \rangle$$

Such bindings are interpreted as shorthand for constructing objects whose payloads are given by the literal values. They do not alter the event semantics.

D.10 Well–Formedness Conditions (Informal)

The grammar permits syntactically valid but semantically inadmissible programs. The following conditions are enforced semantically rather than grammatically:

- An identifier must be introduced by POP before it can be merged.
- Merge targets must be fresh identifiers.
- Authorization constraints determine whether merges and collapses are admissible.
- Refusal prevents the existence of extended histories.

These constraints are intentionally excluded from the grammar to preserve a clean separation between syntax and event–historical semantics.

D.11 Discussion

This grammar emphasizes that Spherepop is not a term–rewriting language but a history–construction calculus. Programs specify not how values are transformed, but which events are attempted and which commitments are made or refused. Execution is replay, and correctness is admissibility.

The simplicity of the grammar reflects a design principle: expressive power should reside in event semantics and authorization, not in syntactic complexity.

E Small–Step Operational Semantics

This appendix derives a small–step operational semantics for Spherepop directly from the core event grammar. The semantics is intentionally minimal: it defines how histories are extended one event at a time, how refusal corresponds to stuckness (absence of a successor configuration), and how replay materializes objects as persistent artifacts.

E.1 Configurations

We model execution as a transition system over configurations. A configuration is a triple

$$\langle H,\ \Gamma,\ \Sigma \rangle,$$

where H is the committed event history, Γ is an object environment, and Σ is a semantic store. The environment Γ maps identifiers to object references; the store Σ maps object references to semantic records. A semantic record minimally contains a payload component and a provenance component, but the semantics is parameterized by the payload algebra.

Definition 16 (Semantic Record)**.** *A semantic record is a pair $\Sigma(o) = (v(o), P(o))$ where $v(o)$ is the payload and $P(o) \subseteq \mathcal{S}$ is a finite provenance set.*

The semantics assumes a partial payload merge operator $\diamond$ on payloads and a partial authorization predicate Auth that may depend on H, Γ and Σ.

E.2 Judgments

We write a small step as

$$\langle H, \Gamma, \Sigma \rangle \xrightarrow{e} \langle H', \Gamma', \Sigma' \rangle,$$

meaning that event e is admissible and produces a successor configuration. If no successor exists, the configuration is stuck; refusal is represented by non–existence of a step.

We will also use a helper judgment for freshness: $\mathsf{fresh}(x, \Gamma)$ holds when identifier x is not bound in Γ.

E.3 Rules for pop

A POP introduces a new committed object name. Operationally, it allocates a fresh object reference o and binds it to the identifier. The payload and provenance of a POP may be supplied externally (e.g. via literal binding) or may default to a unit value.

Proposition 13 (Pop Rule)**.** *If $\mathsf{fresh}(x, \Gamma)$ and $\mathsf{Auth}(H, \Gamma, \Sigma, \textsc{pop}\ x)$ then there exists a fresh object reference o such that*

$$\langle H, \Gamma, \Sigma \rangle \xrightarrow{\textsc{pop}\ x} \langle H \cdot \textsc{pop}(x),\ \Gamma[x \mapsto o],\ \Sigma[o \mapsto (v_0, P_0)] \rangle.$$

Remark 11. *The choice of (v_0, P_0) is parameterized. For mapper outputs, (v_0, P_0) will already encode the shard provenance; for empty reducers, $P_0 = \varnothing$.*

E.4 Rules for link

Links do not change payloads; they register dependency structure. The semantics may record links as part of the store, or treat them as history–only. We present a conservative semantics that records them in Σ as adjacency metadata $\mathsf{Links}(o)$.

Proposition 14 (Link Rule)**.** *If* $\Gamma(x) = o_x$*,* $\Gamma(y) = o_y$*, and* $\mathsf{Auth}(H, \Gamma, \Sigma, \textsc{link}\ x \to y)$ *then*

$$\langle H, \Gamma, \Sigma\rangle \xrightarrow{\textsc{link}\ x \to y} \langle H \cdot \textsc{link}(x \to y),\ \Gamma,\ \Sigma'\rangle$$

where Σ' *is* Σ *updated so that* o_x *records a link to* o_y*.*

E.5 Rules for merge

Merge is the only primitive that necessarily changes payload and provenance. It constructs a fresh target object whose semantic record is computed from the sources. Authorization includes (at minimum) provenance disjointness and may include policy.

Let x, y, z be identifiers. Define $o_x = \Gamma(x)$, $o_y = \Gamma(y)$, and assume z is fresh. Let $\Sigma(o_x) = (v_x, P_x)$ and $\Sigma(o_y) = (v_y, P_y)$.

Theorem 22 (Merge Rule)**.** *If* $\mathsf{fresh}(z, \Gamma)$*,* $P_x \cap P_y = \varnothing$*, the payload merge* $v_x \diamond v_y$ *is defined, and* $\mathsf{Auth}(H, \Gamma, \Sigma, \textsc{merge}\ x\ y \to z)$*, then there exists a fresh object reference* o_z *such that*

$$\langle H, \Gamma, \Sigma\rangle \xrightarrow{\textsc{merge}\ x\ y \to z} \langle H \cdot \textsc{merge}(x, y \to z),\ \Gamma[z \mapsto o_z],\ \Sigma'\rangle$$

where $\Sigma'(o_z) = (v_x \diamond v_y,\ P_x \cup P_y)$ *and* Σ' *also records links from* o_z *to* o_x *and* o_y*.*

Proof. By the stated conditions, the semantic record for the merged object is defined. Freshness ensures a new identity. Recording ancestry links is admissible and preserves lineage. □

E.6 Rules for collapse

Collapse replaces an object by a collapsed representative that preserves a chosen set of invariants. Operationally, collapse may either produce a new object name or rewrite the semantic record of an existing object reference. To preserve the append–only character, we adopt a persistent semantics: collapse allocates a new object reference o' and rebinds the identifier to it.

Let Keep_I denote an invariant projection operator, mapping semantic records to a canonical representative preserving invariants I.

Theorem 23 (Collapse Rule)**.** *If* $\Gamma(x) = o$*,* $\mathsf{Auth}(H, \Gamma, \Sigma, \textsc{collapse}\ x\ \texttt{keep}\ I)$*, and* $\mathsf{Keep}_I(\Sigma(o))$ *is defined, then there exists a fresh object reference* o' *such that*

$$\langle H, \Gamma, \Sigma\rangle \xrightarrow{\textsc{collapse}\ x\ \texttt{keep}\ I} \langle H \cdot \textsc{collapse}(x; I),\ \Gamma[x \mapsto o'],\ \Sigma[o' \mapsto \mathsf{Keep}_I(\Sigma(o))]\rangle.$$

E.7 Rules for refuse

A REFUSE statement is documentation of inadmissibility. Operationally, it does not advance the computation, because it asserts that a particular event has no admissible successor. We therefore treat it as a judgmental annotation, not a transition.

Definition 17 (Refusal and Stuckness)**.** *A configuration is stuck on event e if there is no configuration C' such that $C \xrightarrow{e} C'$. A* REFUSE *statement is well formed exactly when its embedded event would be stuck at the current configuration.*

Remark 12. *This matches the essay's semantics: refusal is absence of a continuation, not an error value.*

F A Static Type and Authorization System

This appendix gives a static discipline that approximates the dynamic authorization predicate Auth and prevents large classes of inadmissible histories from being written. The intent is not to decide authorization completely, but to separate (i) syntactic well–formedness, (ii) static admissibility, and (iii) dynamic policy checks.

F.1 Typing Environments and Types

We define a typing environment Δ mapping identifiers to types. Types track at least two aspects: the payload kind and the provenance footprint.

Let κ range over payload kinds (e.g. histogram, sum, set, embedding). Let π range over provenance descriptors. For static checking, provenance may be approximated as either a concrete finite set (when known) or an abstract region label.

Definition 18 (Types)**.** *A Spherepop object type is a pair $\tau = \kappa[\pi]$, where κ is a payload kind and π is a provenance descriptor.*

We write $\mathsf{disjoint}(\pi_1, \pi_2)$ for a decidable predicate that conservatively approximates disjointness of provenance. If it returns true, dynamic provenance disjointness is guaranteed; if false, disjointness is unknown and must be decided dynamically or refused conservatively.

F.2 Typing Judgments

Typing judgments have the form

$$\Delta \vdash e : \Delta',$$

meaning that under environment Δ, event e is statically admissible and produces an updated environment Δ'.

F.3 Rules

Theorem 24 (Pop Typing)**.** *If $x \notin \mathrm{dom}(\Delta)$ and τ is an assigned type for x, then*

$$\Delta \vdash \textsc{pop}\ x : \Delta[x \mapsto \tau].$$

Remark 13. *The grammar does not assign τ; a front–end may infer it from literal bindings or annotations, or leave it abstract.*

Theorem 25 (Link Typing)**.** *If $\Delta(x) = \tau_x$ and $\Delta(y) = \tau_y$, then*

$$\Delta \vdash \textsc{link}\ x \to y : \Delta.$$

Theorem 26 (Merge Typing)**.** *If $\Delta(x) = \kappa[\pi_x]$, $\Delta(y) = \kappa[\pi_y]$, $z \notin \mathrm{dom}(\Delta)$, and $\mathsf{disjoint}(\pi_x, \pi_y)$ holds, then*

$$\Delta \vdash \textsc{merge}\ x\ y \to z : \Delta[z \mapsto \kappa[\pi_x \cup \pi_y]].$$

Proof. Static disjointness ensures the dynamic disjointness precondition. Payload kinds must match to ensure that $\diamond$ is defined at the kind level. Provenance union is tracked in the resulting type. □

Theorem 27 (Collapse Typing)**.** *If $\Delta(x) = \kappa[\pi]$ and I is an invariant set admissible for κ, then*

$$\Delta \vdash \textsc{collapse}\ x\ \texttt{keep}\ I : \Delta.$$

Remark 14. *Collapse does not change the kind, but it may coarsen provenance. A refined system may update π to a less precise descriptor to reflect forgetting.*

F.4 Soundness Relative to Dynamic Authorization

Let Auth include at least (i) kind compatibility, (ii) provenance disjointness, and (iii) any additional policy constraints.

Theorem 28 (Static Soundness)**.** *If $\Delta \vdash e : \Delta'$ and the runtime configuration realizes Δ, then either e is dynamically authorized or it is rejected only due to policy constraints not expressible in Δ.*

Proof. The typing rules enforce the structural preconditions of authorization: freshness, kind compatibility, and (conservative) provenance disjointness. Any remaining rejections must therefore arise from constraints intentionally left to dynamic policy. □

G A Reference Interpreter Consistent with the Grammar

This appendix specifies a minimal reference interpreter that executes Spherepop programs by replaying their event histories under the small–step semantics. The interpreter is described as an algorithm rather than an implementation, so that multiple concrete runtimes can be validated against the same reference behavior.

G.1 Interpreter State

The interpreter maintains a state (H, Γ, Σ) as in Appendix E, together with an allocation counter for fresh object references.

The interpreter takes as input a parsed program, i.e. a sequence of event statements $(e_1, \ldots, e_n)$.

G.2 Execution Algorithm

Execution proceeds sequentially. For each event e_i, the interpreter checks static admissibility (optional), then checks dynamic authorization, then applies the corresponding transition rule. If no transition rule applies, the interpreter halts and reports refusal (inadmissibility) with an explanation when available.

```
Algorithm Replay(program):
    initialize H := empty
    initialize ÎŞ := empty
    initialize Îč := empty
    for each event e in program:
        if optional_static_check_enabled:
            require ÎŤ âŁć e : ÎŤ'  (else reject)
        if no small-step rule applies to â§ÍH,ÎŞ,Îčâ§Ľ with label e:
            report REFUSAL(e, H, ÎŞ, Îč)
            halt
        else:
            apply the unique rule to obtain â§ÍH',ÎŞ',Îč'â§Ľ
            set (H,ÎŞ,Îč) := (H',ÎŞ',Îč')
    return â§ÍH,ÎŞ,Îčâ§Ľ
```

G.3 Determinism

Determinism holds when allocation is deterministic (e.g. fresh references chosen by a counter) and when payload merge $\diamond$ is deterministic.

Theorem 29 (Determinism of Replay)**.** *Assume deterministic allocation and deterministic payload operations. Then for any program, replay produces either a unique final configuration or a unique first refusal event.*

Proof. By construction, each event statement has at most one applicable transition rule given the current configuration. If no rule applies, refusal is determined at that point. Otherwise the successor is uniquely determined. □

G.4 Conformance Criteria

A concrete implementation conforms to this reference interpreter if, for every program, it agrees on (i) the first point of refusal, when refusal occurs, and (ii) the final replayed semantic records for all committed identifiers, modulo the chosen collapse invariants.

Definition 19 (Interpreter Conformance)**.** *An implementation conforms if for all programs P, either both reference and implementation refuse at the same event index, or both terminate with equivalent stores under invariant–preserving equivalence.*

References

[1] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[2] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[3] M. Fowler. Event sourcing. *martinfowler.com*, 2005. Online essay.

[4] M. Shapiro, N. PreguiÃğa, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In *Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2011.

[5] C. Baquero, P. SÃĺrgio Almeida, and A. Shoker. Making operation-based CRDTs operation-based. In *Proceedings of the 21st International Conference on Principles of Distributed Systems (OPODIS)*, 2017.

[6] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[7] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[8] S. Abramsky and B. Coecke. A categorical semantics of quantum protocols. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS)*, 2004.

[9] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, 1996.

[10] P. Selinger. A survey of graphical languages for monoidal categories. In *New Structures for Physics*, Springer, 2011.

[11] C. H. Bennett. The thermodynamics of computationâĂŤA review. *International Journal of Theoretical Physics*, 21(12):905–940, 1982.

[12] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, 1961.

[13] S. Mac Lane. *Categories for the Working Mathematician*. Springer, 2nd edition, 1998.

[14] J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2nd edition, 2009.

[15] A. Vaswani et al. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[16] P. Battaglia et al. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261*, 2018.

[17] M. M. Bronstein et al. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv:2104.13478*, 2021.

[18] J. A. Barandes. Unistochastic quantum theory. *Physical Review A*, 108(3), 2023.

[19] K. Friston. The free-energy principle: A unified brain theory? *Nature Reviews Neuroscience*, 11:127–138, 2010.

[20] G. Dodig-Crnkovic. Information, computation, cognition. *Entropy*, 16(1):245–256, 2014.

[21] G. Winskel. Event structures. In *Advances in Petri Nets*, Springer, 1987.

[22] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[23] C. Hewitt. A universal modular actor formalism for artificial intelligence. In *Proceedings of IJCAI*, 1973.