Distributed CatDB: Typed Operation Algebra and Coordination Boundaries

Matthew Long
The YonedaAI Collaboration
YonedaAI Research Collective
Chicago, IL
matthew@yonedaai.com · https://yonedaai.com

23 July 2026

Abstract

A database catalog is fairly easy to replicate while every site only adds unrelated, immutable objects. The hard cases begin when one site activates a schema while another edits a mapping that depends on the old schema, or when two sites move the same branch head during a partition. Delivering every update everywhere does not tell us whether the updates commute. It also does not protect an invariant or identify the authorized semantic choice.

This paper gives a provisional distributed contract for CatDB. The contract uses typed operation envelopes, explicit causal dependencies, component-level merge laws, conflict records, escrow rights, and named ordered transitions. It deliberately does not assign one merge operation to the whole catalog. Immutable content-addressed artifacts are candidates for join-based replication. Dependent additions may use causal delivery. Concurrent alternatives can converge as an explicit multi-value conflict. Active schema or mapping replacement, policy replacement, branch publication, membership, and unsafe checkpointing generally require conflict preservation or coordination.

We define the operation and state model, give a 25-operation classification, and work through twelve finite counterexamples. Ordinary add/remove semantics already fail under reorder. The later examples cover last-writer-wins branch heads, unchecked uniqueness, mapping composition, scalar logical clocks, and forking hash chains. None supplies a general semantic merge. The CAP result also remains unchanged: category-theoretic typing can expose preservation obligations, but it cannot move messages across a partition or make a noncommutative update commute.

The result is a research contract, not a distributed system. The repository has no distribution crate, causal metadata implementation, replicated operation log, conflict resolver, partition simulator, convergence fixture, or distributed benchmark. Its direct P9 workspace and P13 provenance dependencies are unresolved. Runtime, recovery, convergence, availability, and performance claims are therefore OBSTRUCTED. The proposed typed classification is an ENGINEERING HYPOTHESIS, and each unimplemented bounded component remains OPEN until executable and formal evidence exists.

Contents

1 Introduction

1.1 The problem in plain language

Suppose three CatDB replicas share a customer schema. Replica A creates a new schema version that renames `amount` to `total`. Replica B, which has not seen that change, edits a mapping so that `amount` is converted from cents to dollars. Replica C publishes a branch whose materialization still names the old schema and mapping. The network then partitions.

Each edit can be valid against the version its author saw. That does not make the edits interchangeable. If the rename is applied first, a name-based mapping edit may find nothing to change. If the unit edit is applied first, the renamed field carries the new scale. Once the partition heals, a system that merely sorts operations by timestamp will produce one answer everywhere, but the timestamp did not establish that answer's meaning or authority.

The same split appears elsewhere in the catalog. Adding an immutable schema version is different from making it active. Recording two semantic commits does not choose one branch head. Assertion and retraction events can both be retained without deciding which assertion is authoritative. Content sets, conflict sets, escrow counters, and consensus-backed pointers are all replicated objects, but they need different consistency mechanisms.

1.2 Research question and bounded thesis

The research question is:

> Which typed CatDB operations commute, converge, conflict, or require coordination under duplicate, reordered, delayed, and partitioned delivery?

The bounded thesis is an ENGINEERING HYPOTHESIS:

> CatDB may execute an operation without foreground coordination only when the operation's typed effect, preconditions, invariants, delivery assumptions, and conflict policy establish the required commutativity, idempotence, or join-semilattice conditions. Other operations preserve an explicit conflict, consume reserved rights, or use a named ordering mechanism.

This is not the claim that CatDB "is a CRDT." It is an operation-specific classification. The full state is decomposed into components that may use different protocols. A regime requested by an operation is not self-certifying; a checker must derive or validate it from the payload, preconditions, authority, invariants, and evidence for that operation class.

1.3 Contributions

This provisional paper contributes:

1. a typed operation envelope, causal context, application result, and decomposed replicated state;
2. state-based, operation-based, and delta-state compatibility conditions for a declared CatDB subset;
3. a 25-operation matrix spanning schemas, mappings, provenance, identity, policies, branch heads, materializations, checkpoints, and membership;
4. twelve finite counterexamples that preserve conflicts hidden by generic merge slogans;
5. a precise CAP and category-theory boundary;
6. a deterministic history, Haskell, Lean, Rust, backend, and benchmark roadmap with falsification gates; and
7. an implementation audit that keeps every current distributed claim OPEN or OBSTRUCTED.

1.4 Paper organization

Section 2 fixes the evidence boundary. Section 3 locates the proposed work in the distributed-systems literature. Section 4 gives a running example. Sections 5 to 7 define the formal contract. Section 8 gives the operation classification. Sections 9 and 10 examine semantic evolution and coordination. Section 11 proves the finite counterexamples. Sections 12 to 14 specify replay, implementation, and evaluation work. The paper ends with limitations, open questions, exact nonclaims, and reproducibility requirements.

2 Evidence vocabulary and current status

2.1 Status discipline

The manuscript separates external results, paper reasoning, and CatDB behavior.

Label	Meaning
ESTABLISHED EXTERNALLY	A primary source establishes the result under its own model. It is not evidence that CatDB implements the result.
PROVED IN MODEL	A definition, calculation, or finite counterexample in this paper establishes the bounded statement. It is not runtime evidence.
ENGINEERING HYPOTHESIS	A proposed design relation whose implementation and acceptance evidence do not yet exist.
OPEN	The claim has a specified test, but no accepted result.
OBSTRUCTED	A prerequisite definition, implementation, or fixture is absent, so the claim cannot yet be tested honestly.
REJECTED	The claim contradicts a theorem, counterexample, or the local evidence cut.

The operational labels requested for candidate CatDB behavior are ENGINEERING HYPOTHESIS, OPEN, and OBSTRUCTED. A paper proof does not promote an implementation claim. An external system result does not promote one either.

2.2 Local evidence cut

The repository revision inspected for this paper is:

`79b654fd456257865a1fe0e64a776807a012a277`.

The Rust workspace names exactly six crates: `catdb-core`, `catdb-ir`, `catdb-schema`, `catdb-mapping`, `catdb-migration`, and `catdb-cli`. There is no `catdb-distribution` workspace member or source directory.

Searches over the Rust, Haskell, Lean, and fixture surfaces found no accepted P10 operation log, causal metadata type, conflict resolver, partition simulator, convergence fixture, or benchmark. The accepted fixture profile covers eight bounded semantic categories but excludes distributed behavior. The accepted Lean coverage explicitly excludes distributed execution. The shared roadmap marks distribution slice 13 and `SYS-DIST-01` OPEN.

Local artifact	What it establishes	What it does not establish
Fixture schema v1	Eight bounded semantic categories and exact local structural or semantic outcomes	Replication, causal delivery, conflict, partition healing, or convergence
Rust semantic slice	Bounded schema, path, mapping, and migration behavior for its fixture profile	Distribution, replay, checkpointing, or consensus

Local artifact	What it establishes	What it does not establish
Haskell reference slice	Independent outcomes for the same bounded semantic profile	A distributed oracle or CRDT implementation
Lean F3 coverage	Typed paths and mapping identity or composition laws under stated assumptions	Operation commutativity, storage, liveness, or distributed execution
`SYS-DIST-01`	A future acceptance statement for a declared convergent subset	A passing implementation
Roadmap slice 13	Proposed distribution commands and test names	Existing crates, tests, or results
D6/F benchmark plan	Future distributed-history dimensions and properties	Measurements or accepted convergence evidence

2.3 Dependency boundary

P10 depends directly on P9 and P13. P9 must define semantic commits, workspace state, stable artifact identity, branch references, diffs, and merge conflicts. P13 must define provenance identity, causal and execution events, replay references, redaction, and evidence attached to conflict decisions. Neither prerequisite has supplied an accepted interface for P10. The operation envelope and state space are therefore provisional. Runtime and convergence claims are OBSTRUCTED.

P10 feeds P11, which will assign consistency guarantees to the operation classes. P11 cannot be treated as an already accepted global model without reversing that dependency.

3 Prior art and the bounded CatDB delta

3.1 Order, causality, and snapshots

Lamport's happened-before relation and scalar logical clocks provide a consistent way to reason about partial order and to construct a total order when one is required [18]. Fidge and Mattern developed vector-time accounts that retain more causal information [10, 21]. Causal memory formalizes which writes an execution must observe [1]. Distributed snapshots record a consistent global state under explicit process and channel assumptions [7]. None of these results supplies a semantic winner for concurrent schema or mapping edits.

3.2 Convergence and replicated datatypes

State- and operation-based CRDTs have precise algebraic and delivery conditions [26, 25]. Delta-state replication changes dissemination granularity but preserves semilattice and anti-entropy obligations [3]. Replicated JSON shows that a structured datatype can preserve concurrent changes and converge under a formal semantics [16]. Mechanized work has

verified strong eventual consistency for selected algorithms and supplied sequential OpSet specifications [13, 17]. These are datatype results, not a proof that an arbitrary semantic catalog has one merge.

3.3 Availability and coordination

Gilbert and Lynch prove the atomic-consistency and availability incompatibility for an asynchronous network with partitions [12]. CALM-style results characterize monotone computations within a formal transducer model [4]. Invariant confluence asks whether a transaction set preserves a named invariant under merge [6], while highly available transaction results separate guarantees that are and are not achievable under a precise availability definition [5]. Abstract-datatype commutativity can depend on state, arguments, and results [29]. Escrow reserves rights for suitable bounded invariants [22]. RedBlue consistency demonstrates an operation-specific split between coordinated and uncoordinated work [19]. P10 must instantiate these models, not borrow their conclusions by analogy.

3.4 Systems precedents

Bayou uses dependency checks and application-specific conflict handling [27]. Gray and collaborators analyze hazards in update-anywhere replication [14]. Dynamo exposes versioning and application reconciliation in a service-specific highly available store [9]. COPS and Cure demonstrate causal storage under different models and system assumptions [20, 2]. Raft provides replicated-log consensus [23]; Spanner provides externally consistent transactions using a coordinated system design [8]. These systems are precedent for mechanisms, not evidence that CatDB implements them.

3.5 Schema, mapping, and replay

F1 performs online schema change through compatible intermediate states and bounded version skew [24]. Mapping adaptation under schema evolution can produce several valid candidates while preserving earlier choices where possible [28]. Event sourcing reconstructs state from a specified event sequence [11]; it does not define a concurrent merge. Linearizability gives a correctness condition for concurrent objects [15], but it does not become partition-available under the Gilbert-Lynch boundary.

3.6 Plausible contribution

The plausible CatDB-specific delta is an ENGINEERING HYPOTHESIS:

> A typed operation algebra over co-versioned schemas, mappings, semantic commits, identity decisions, provenance, authority, and materializations can emit a checkable per-operation coordination class, preserve finite counterexample conflicts, and reproduce its classifications across Lean, Haskell, Rust, and deterministic backend histories.

This is narrower than a new CRDT theory. Logical clocks, causal consistency, CRDT convergence, CAP, CALM, invariant confluence, escrow, consensus, staged schema rollout, mapping adaptation, and event sourcing are established prior art.

4 Running example

4.1 Three replicas and one semantic change

Replica A owns workspace branch `schema-modernization`. Starting from schema S_1, it creates immutable version S_2, where stable object `sales.amount` keeps its identity but its display name becomes `sales.total`. Replica B owns branch `currency-fix`. It creates mapping M_2, which changes the unit conversion on the stable object from cents to dollars. Replica C publishes materialization Q_1, derived from S_1 and M_1. Each version is immutable and content-addressed.

If the operations name stable identities and carry their version dependencies, the system can retain S_2, M_2, and Q_1 as immutable artifacts. Publication is the difficult step. Activating M_2 before receiving S_2 leaves a missing dependency. Moving the production branch to S_2 while another replica moves it to Q_1's parent creates two concurrent heads. A refresh of Q_1 is only meaningful if it records the schema and mapping snapshot it used.

The example uses three possible outcomes:

1. an immutable addition returns Applied or Duplicate;
2. a dependent activation returns Deferred until its declared versions are present;
3. concurrent publication returns Conflict or enters an ordered compare-and-swap protocol.

A timestamp does not decide the business meaning. A deterministic tie-break may make every replica display the same head, but it hides the discarded proposal. The conflict record must retain both candidates, their causal contexts, their authorities, and the evidence needed for a later decision.

4.2 Partition and healing history

At time t_0, all replicas have S_1, M_1, and branch head h_0. The partition isolates A from B and C.

$$\begin{aligned} A &: \mathrm{put}(S_2); \mathrm{moveHead}(h_0, h_A), \\ B &: \mathrm{put}(M_2); \mathrm{moveHead}(h_0, h_B), \\ C &: \mathrm{publish}(Q_1, S_1, M_1). \end{aligned}$$

When the partition heals, all three immutable artifacts can enter the shared artifact component. The two head moves have the same precondition h_0, so neither dominates the other. They become a multi-value conflict. The materialization remains valid only for the named S_1, M_1 snapshot and must not be relabeled as current under S_2, M_2.

Status. PROVED IN MODEL for the finite history under the definitions below; OBSTRUCTED for any CatDB execution of it.

5 Formal state and operation semantics

This section fixes a finite specification model. It does not assert a particular wire protocol, storage engine, or production membership system.

5.1 Primitive types

```
ReplicaId := stable identifier for a replica
Counter := nonnegative integer local to one ReplicaId
OpId := (ReplicaId, Counter)
Digest := collision-resistant content digest
ArtifactId := Digest
VersionId := Digest
ScopeKey := typed semantic conflict domain
AuthorityId := stable authority-policy reference
InvariantId := stable invariant reference
SnapshotId := source or catalog snapshot identity
```

`OpId` supplies duplicate identity, not semantic order. `ArtifactId` and `VersionId` identify immutable canonical content. A mutable display name is not a replica or artifact identity.

5.2 Causal context

Definition 5.1 (Finite causal context)**.** For a finite replica set R, a causal context is a version vector

$$V : R \to \mathbb{N}.$$

Define

$$V \preceq W \quad \Longleftrightarrow \quad \forall r \in R,\ V(r) \leq W(r),$$

and

$$(V \sqcup W)(r) = \max(V(r), W(r)).$$

Contexts are concurrent, written $V \parallel W$, when neither dominates the other.

Proposition 5.2 (Vector-context semilattice)**.** *For a fixed finite replica domain, $\preceq$ is a partial order and $\sqcup$ is an associative, commutative, idempotent least upper bound.*

Proof. Reflexivity, antisymmetry, and transitivity follow pointwise from the natural number order. Pointwise maximum is associative, commutative, and idempotent. It dominates each operand. Any vector that dominates both operands also dominates their pointwise maximum, so the maximum is the least upper bound. □

Status. PROVED IN MODEL for the fixed finite model; dynamic membership, compaction, dotted vectors, and Byzantine identities remain OPEN.

5.3 Typed operation envelope

```
Operation<K> = {
  op_id: OpId,
  kind: K,
  author_replica: ReplicaId,
  causal_context: CausalContext,
  dependencies: Set<VersionId>,
  semantic_scope: Set<ScopeKey>,
  precondition_digest: Digest,
  authority: AuthorityId,
  invariants: Set<InvariantId>,
  payload: Payload<K>,
  idempotence_key: Digest,
  requested_regime: Regime,
  provenance_ref: VersionId
}
```

The provisional regimes are:

Regime	Intended contract
JOIN	State-based inflationary update and component join
CAUSAL	Operation delivery after declared dependencies
MULTIVALUE	Preserve concurrent alternatives as a conflict
ESCROW	Consume preallocated invariant-preserving rights
ORDERED	Total-order or consensus-backed transition
ADMIN	Membership, checkpoint, or garbage-collection barrier

5.4 State decomposition

```
DistributedState = {
  artifacts: ArtifactJoinState,
  append_events: EventJoinState,
  causal_frontier: CausalContext,
  pending: Map<OpId, Operation>,
  conflicts: ConflictJoinState,
  coordinated_refs: OrderedReferenceState,
  rights: EscrowState,
  checkpoint: CheckpointState
}
```

There is deliberately no proposed total function

$$\mathsf{join} : \mathsf{State} \times \mathsf{State} \to \mathsf{State}$$

for this whole record. Only a component with a justified join receives one. Ordered references, escrow rights, and administrative state use different contracts.

5.5 Application result

```
apply : State x Operation<K> -> ApplyResult

ApplyResult =
    Applied(State, Evidence)
  | Duplicate(State, ExistingEvidence)
  | Deferred(MissingDependencies)
  | Conflict(ConflictRecord)
  | Rejected(Violation)
```

An implementation must not translate Deferred, Conflict, or Rejected into silent success. A partial update before one of those outcomes is a correctness failure unless the operation contract declares a compensatable multi-stage transition.

5.6 Conditional commutativity

Definition 5.3 (Conditional commutativity)**.** Operations a and b commute at state s under observational equivalence $\approx$ when both orders are defined and

$$\mathsf{apply}(\mathsf{apply}(s, a), b) \approx \mathsf{apply}(\mathsf{apply}(s, b), a).$$

The relation can depend on state, parameters, authority, invariants, and results. An operation name such as `add`, `rename`, or `merge` is not enough. A conservative first-slice witness may require

$$\mathsf{scope}(a) \cap \mathsf{scope}(b) = \varnothing$$

plus proof that neither operation changes the typing context, invariant set, authority, or interpretation needed by the other. Disjoint serialized keys alone do not establish semantic noninterference.

5.7 Duplicate safety and causal readiness

Duplicate safety requires

$$\mathsf{apply}(\mathsf{apply}(s, o), o) \approx \mathsf{apply}(s, o).$$

It can follow from an idempotent effect or durable `OpId` deduplication. At-most-once transport is not assumed.

An operation is causally ready only when

$$\mathsf{dependencies}(o) \subseteq \mathsf{versions}(s) \quad \text{and} \quad \mathsf{causalContext}(o) \preceq \mathsf{frontier}(s).$$

This check prevents dependency inversion. It neither orders concurrent operations nor proves their semantic compatibility.

Proposition 5.4 (Order-independent finite replay)**.** *Let H be a finite set of deterministic operations. Suppose every replay order is a linear extension of the same finite dependency relation; every pair that is incomparable in that relation commutes whenever adjacent in an*

admitted replay; duplicate application is idempotent; and all preconditions remain enabled under permitted adjacent swaps. Then all duplicate-free linear extensions of H produce observationally equivalent states.

Proof. Any two linear extensions of a finite partial order are connected by a finite sequence of swaps of adjacent incomparable elements. Each permitted swap preserves observational equivalence by the commutativity hypothesis and preserves later definedness by the precondition hypothesis. Induction over the swap sequence gives equivalent final states. Duplicate insertions can be removed without changing the result by idempotence. □

Status. PROVED IN MODEL for the stated finite assumptions; identifying a nontrivial CatDB subset satisfying them is OPEN.

5.8 Convergence target

Let H_r be the admitted operation set delivered to replica r. For a declared subset C, the bounded strong-eventual-consistency target requires:

1. every nonfaulty replica eventually receives the same finite admitted set $H \subseteq C$;
2. dependencies precede dependent effects where the operation contract requires causal order;
3. duplicate delivery has no additional effect;
4. deterministic interpretation terminates; and
5. resulting states are observationally equivalent.

Thus

$$\forall r_1, r_2, \quad H_{r_1} = H_{r_2} = H \Longrightarrow s_{r_1} \approx s_{r_2}.$$

The implication begins after equal admitted delivery. It does not prove that partitions heal, messages arrive, data are fresh, or the converged state matches business intent.
Status. ENGINEERING HYPOTHESIS as a specification; OBSTRUCTED as a CatDB runtime claim.

6 CRDT compatibility contract

6.1 State-based component

A state component X is a candidate state CRDT only if:

1. $(X, \sqsubseteq)$ is a join-semilattice;
2. local mutations are inflationary;
3. merge computes the least upper bound;
4. replicas eventually exchange sufficient state or deltas; and

5. serialization preserves the order and join exactly.

Associativity, commutativity, and idempotence of join are mandatory. An immutable artifact set is a plausible first component if an equal digest must identify equal canonical bytes and any collision or mismatch is rejected as an integrity failure.

6.2 Operation-based component

An operation component is a candidate operation CRDT only if its effects are deterministic, concurrent effects commute under the declared equivalence, delivery satisfies the required reliability and causal assumptions, duplicates are suppressed or idempotent, and dependencies and preconditions survive delivery.

The state-based and operation-based assumptions differ [26]. P10 cannot borrow the smaller messages of operation replication while assuming the weaker dissemination conditions of state replication.

6.3 Delta-state component

A delta-state design still needs a join-semilattice and an anti-entropy protocol whose delta intervals preserve convergence and causal conditions [3]. A small payload is neither an operation nor a proof of idempotence.

6.4 Three meanings of monotonicity

The following statements are distinct:

1. a local transition is inflationary in a chosen semilattice;
2. a query is monotone under its input order; and
3. an operation set preserves a named application invariant under merge.

No item implies the other two without additional definitions. CALM has a specified network model [4]; invariant confluence is relative to transactions and invariants [6]. "Monotone" is not a universal coordination exemption.

7 Logical clocks and causal metadata

7.1 Scalar logical clocks

A Lamport clock can satisfy

$$a \to b \Longrightarrow C(a) < C(b).$$

The converse is false. $C(a) < C(b)$ does not establish $a \to b$, and a scalar clock alone cannot detect concurrency [18]. Adding a replica-ID tie-break produces a deterministic total order, but its extra order is policy, not causality.

7.2 Vector-style context

For the fixed finite model, $V_a \prec V_b$ can represent causal precedence, while incomparable vectors expose concurrency [10, 21]. The vector does not provide synchronized wall time, freshness, a single global snapshot, a conflict winner, membership compaction, or causal delivery unless the transport uses it correctly.

7.3 Causal consistency

Causal memory, COPS, and Cure show stronger ordering than bare eventual consistency without one global order over every operation [1, 20, 2]. Their data models and protocols differ. P10 may use causal delivery for dependent schema and mapping operations, but causal consistency is not serializability or linearizability.

8 Candidate operation and coordination matrix

The regime abbreviations are $J = \mathsf{JOIN}$, $C = \mathsf{CAUSAL}$, $M = \mathsf{MULTIVALUE}$, $E = \mathsf{ESCROW}$, $O = \mathsf{ORDERED}$, and $A = \mathsf{ADMIN}$. Every row is an ENGINEERING HYPOTHESIS. The last column records whether the corresponding CatDB result is OPEN or OBSTRUCTED.

Table 4: Candidate P10 operation classification.

Operation	Conditional duplicate behavior	Concurrent relation	Regime	CatDB status
Put immutable artifact by content hash	Same hash and bytes are idempotent; same hash and different bytes is a fatal integrity failure	Commutes with distinct content hashes	J	OPEN
Append semantic commit with immutable ID and parent set	Duplicate ID must reproduce identical canonical bytes	Commutes as DAG-node addition if parents exist; branch publication is separate	C/J	OBSTRUCTED on P9
Append provenance event with stable ID	Duplicate must return existing evidence	Commutes as event-set addition; a single predecessor hash chain may fork	C/J	OBSTRUCTED on P13
Record immutable validation receipt	Idempotent by receipt digest	Commutes when it names fixed input versions and checker revision	J	OPEN
Add schema declaration under fresh stable ID	Duplicate must match bytes	Commutes only if namespaces, equations, types, and active invariants remain independent	C/M	OPEN
Add schema equation or constraint proposal	Duplicate proposal is idempotent	Proposal append may commute; activation can invalidate data or mappings	M/O	OBSTRUCTED
Replace active schema version	Compare-and-swap can reject a duplicate stale request	Generally noncommutative with schema, mapping, query, and data operations	O	OBSTRUCTED
Rename or delete schema object	Operation ID can deduplicate replay	Noncommutative with references, mappings, constraints, and queries	M/O	OBSTRUCTED
Add immutable mapping version under fresh ID	Duplicate must match canonical mapping	Version addition may commute after dependencies; activation is separate	C/J	OPEN
Replace or activate mapping for a scope	Compare-and-swap can reject stale replay	Generally noncommutative with schema evolution and other replacements	M/O	OBSTRUCTED
Compose mappings	Same derived-version request can be idempotent by digest	Composition order matters and the reverse may be ill-typed	C	OPEN
Add assertion under a fresh assertion ID	Duplicate must match assertion and evidence	Event addition commutes; authoritative current-state projection may not	C/J	OBSTRUCTED on P13
Retract or invalidate assertion	Replay can deduplicate by event ID	Concurrent assertion and retraction need observed-remove or authority policy	C/M	OBSTRUCTED
Add identity equivalence edge	Duplicate edge is idempotent	Set union converges, but closure can violate authority or cardinality policy	$C/M/O$	OBSTRUCTED on P7/P13
Reverse or split identity decision	Replay can deduplicate event	Nonmonotone against prior merges and dependent derivations	M/O	OBSTRUCTED
Increment unbounded counter	Increment needs deduplication or a per-replica component	Concurrent increments commute	J/C	OPEN
Decrement bounded quota	Duplicate debit is unsafe without deduplication	Concurrent debits can violate a lower bound	E/O	OPEN
Advance single-source cursor	Repeated exact position may be idempotent	Maximum works only in one comparable epoch with no fork	C/J	OBSTRUCTED on P6
Invalidate materialization version	Duplicate invalidation is idempotent	Set addition can commute when it names an immutable version	J/C	OBSTRUCTED on P6
Publish refreshed materialization	Duplicate result digest is idempotent	Noncommutative across snapshots, mappings, and active pointer	M/O	OBSTRUCTED
Change authority or semantic policy	Replay can deduplicate request	Grant, revoke, and replacement are order-sensitive	O	OBSTRUCTED
Approve a fixed immutable proposal	Duplicate approval can be idempotent	Approval-set addition may commute; thresholds and revocation may not	$C/J/O$	OBSTRUCTED

Operation	Conditional duplicate behavior	Concurrent relation	Regime	CatDB status
Move branch head	Compare-and-swap can make stale replay fail	Concurrent moves conflict; last-writer-wins discards a branch decision	M/O	OBSTRUCTED on P9
Checkpoint and truncate operation history	Same certificate may be idempotent	Unsafe against lagging replicas or unobserved tombstones	A/O	OBSTRUCTED
Change replica membership	Duplicate configuration ID can be recognized	Concurrent changes alter quorum and causal-metadata interpretation	O/A	OBSTRUCTED

The matrix is a research hypothesis. It is not accepted behavior. In particular, a slash lists candidate mechanisms to investigate; it does not assert that they are interchangeable.

9 Schema and mapping replication

9.1 Immutable versions and active references

The candidate design separates two layers:

1. immutable schema and mapping versions are content-addressed artifacts; adding independent versions may be a monotone replicated operation; and
2. active references identify the version that governs a workspace, branch, query, or write path; changing an active reference is a publication decision that may require conflict preservation or ordering.

Convergence of the version set does not imply convergence of the active semantic interpretation.

9.2 Checked independent schema addition

Adding declarations d_1 and d_2 under fresh stable identifiers is a candidate commuting pair only when:

1. neither reuses an identifier or namespace key;
2. neither changes the type or interpretation of an existing declaration;
3. their equations and constraints do not create a joint inconsistency;
4. existing mappings, queries, data, and policies remain well-typed;
5. activation is not implicit; and
6. the combined schema has a deterministic canonical representation.

The first slice should treat this list as a checked noninterference condition, not as a law for all operations named `Add`.
Status. ENGINEERING HYPOTHESIS; the checker and executable histories are OPEN.

9.3 Deletion, rename, and type change

Deletion, rename, and type change can invalidate mapping images, schema equations, constraints, queries, materializations, connectors, identity rules, provenance explanations, and policies. A tombstone can make replication state monotone, but it does not make the semantic transition commute. The system still needs a rule for concurrent references and for every derived artifact that used the old schema.

9.4 Mapping addition, activation, and composition

An immutable mapping version can be added after its source schema, target schema, and validation dependencies arrive. Activating it can change query answers, migration behavior, information-loss status, write authority, and provenance.

Mapping composition is associative where the typed categorical construction and preservation assumptions hold. It is not commutative:

$$G \circ F \neq F \circ G$$

in general, and the reverse composition may not be typed. Associativity of sequential transformations is not a concurrent merge rule.

9.5 Mapping adaptation as a search result

Schema evolution can invalidate mappings, and adaptation may produce several valid candidates [28]. A candidate P10 adaptation record is:

```
MappingAdaptation = {
  old_mapping: VersionId,
  source_schema_before: VersionId,
  source_schema_after: VersionId,
  target_schema_before: VersionId,
  target_schema_after: VersionId,
  candidate_mapping: VersionId,
  preserved_choices: Set<ChoiceId>,
  changed_choices: Set<ChoiceId>,
  obligations: Set<InvariantId>,
  authority: AuthorityId,
  status: proposed | validated | approved | rejected
}
```

Two valid candidates remain a conflict until an authorized policy chooses, composes, or rejects them. Deterministic enumeration does not create that authority.

9.6 Staged compatibility

F1 uses pairwise-compatible intermediate schemas and bounded version skew because abrupt schema changes can corrupt data or produce incorrect results while old and new servers coexist [24]. The result does not justify arbitrary asynchronous merge. Its shared-storage architecture, stateless-server assumptions, and schema semantics are not CatDB evidence.

For adjacent versions v_i and v_{i+1}, the provisional relation

$$v_i \ \mathsf{compatible}\ v_{i+1}$$

requires a witness:

```
CompatibilityWitness = {
  before: VersionId,
  after: VersionId,
  readers: Set<ComponentVersion>,
```

```
  writers: Set<ComponentVersion>,
  preserved_operations: Set<OperationKind>,
  blocked_operations: Set<OperationKind>,
  data_backfill: Optional<BackfillPlan>,
  mapping_adaptations: Set<VersionId>,
  invariants_checked: Set<InvariantId>,
  maximum_version_skew: NonnegativeInteger,
  evidence: Set<EvidenceRef>
}
```

Status. OBSTRUCTED because no such CatDB witness type or checker exists.

10 Partitions, availability, and coordination

10.1 Per-operation partition contracts

CAP terminology is useful only after its meanings are fixed. For each operation class K, P10 requires:

```
PartitionContract<K> = {
  safety_property: exact named property,
  availability_property: exact response/termination property,
  failure_model: crash | omission | partition | other,
  reachable_replicas_required: integer or quorum rule,
  accepted_partition_result: value | conflict | stale |
                          rejection | timeout,
  healing_rule: anti_entropy | replay |
              consensus_catch_up | repair,
  freshness_bound: optional and separately justified
}
```

There is no platform-wide answer. Immutable artifact addition, bounded quota debit, schema activation, and a linearizable branch-head update may use different contracts.

10.2 CAP boundary

Corollary 10.1 (Typing does not remove the CAP execution)**.** *Suppose a* CatDB *object and its clients satisfy the network, failure, availability, and atomic-consistency definitions of Gilbert and Lynch [12]. Replacing an untyped update with a typed morphism or a functorially checked operation does not permit both atomic consistency and availability in the partition execution covered by their theorem.*

Proof. The type annotation does not add a communication edge to the partitioned execution. If both sides must respond, neither can learn the other side's concurrent choice. If the responses must behave as operations on one atomic object, the indistinguishability argument remains. At least one side must wait, reject, or otherwise fail the theorem's availability condition. □

Status. ESTABLISHED EXTERNALLY for the impossibility result and PROVED IN MODEL for the model-preservation inference; any claim that CatDB "beats CAP" is REJECTED.

10.3 Named coordination mechanisms

Need	Candidate mechanism	Evidence boundary
Same immutable content set	State join or delta anti-entropy	Needs a semilattice and eventual exchange
Causal dependency order	Causal broadcast or buffering	Does not choose a concurrent winner
Preserve concurrent alternatives	Multi-value register or conflict set	Converges on conflicts, not one semantic value
Maintain numeric bound	Escrow or reservation rights	Allocation and recovery are part of correctness
Select active schema or branch head	Consensus-backed compare-and-swap	Loses partition-time availability for that operation
Safe schema rollout	Staged compatible versions	Needs a proved compatibility relation and skew bound
Compact tombstones or history	Stable causal frontier and checkpoint certificate	Unsafe while unseen operations can arrive
Byzantine resistance	Outside the first P10 profile	Crash and partition evidence does not transfer

10.4 Event sourcing boundary

Event sourcing can reconstruct state by replaying one specified history [11]. It does not supply a merge law for concurrent histories, causal delivery, duplicate safety, invariant preservation, or agreement on one authoritative order. "Append-only" is not a CRDT proof.

11 Counterexample histories

Each history is a paper-level result under its displayed definitions. None is evidence of a CatDB runtime.

11.1 CE-01: naive add and remove diverge

Let $S_A = S_B = \varnothing$, and consider concurrent operations

$$a = \mathrm{add}(x), \qquad b = \mathrm{remove}(x)$$

with ordinary sequential set semantics. Then

$$b(a(\varnothing)) = \varnothing \quad \text{but} \quad a(b(\varnothing)) = \{x\}.$$

The operations do not commute. An observed-remove, add-wins, remove-wins, tombstone, or explicit-conflict design must change the semantics.
Status. PROVED IN MODEL; CatDB implementation OBSTRUCTED.

11.2 CE-02: schema rename and mapping-unit edit

Let

$$s = (name, scale) = (\texttt{amount}, 100).$$

Define

$$r(n,k) = \begin{cases} (\texttt{total}, k) & n = \texttt{amount}, \\ (n,k) & \text{otherwise}, \end{cases}$$

and

$$u(n,k) = \begin{cases} (n,1) & n = \texttt{amount}, \\ (n,k) & \text{otherwise}. \end{cases}$$

Direct calculation gives

$$r(u(s)) = (\texttt{total}, 1), \qquad u(r(s)) = (\texttt{total}, 100).$$

Both edits are valid against the base version, but they do not commute. Stable identifiers may help detect or translate the edit. The translation rule is the missing conflict protocol, not a consequence of replication.
Status. PROVED IN MODEL; CatDB implementation OBSTRUCTED.

11.3 CE-03: uniqueness is not merge-closed

Define

$$I(S) \iff \texttt{email} \text{ is unique in } S.$$

From one valid ancestor, replica A inserts customer 1 with `a@example.org`; replica B inserts customer 2 with the same email. Each local state satisfies I, while their set union does not. The operation and invariant pair is not invariant-confluent under this merge. Coordination, ownership, reservation, or an explicit conflicting state is required.
Status. PROVED IN MODEL; CatDB implementation OBSTRUCTED.

11.4 CE-04: last-writer-wins loses a branch

Let the initial branch head be h_0. During a partition:

$$A : h_0 \mapsto h_A, \qquad B : h_0 \mapsto h_B.$$

A timestamp and replica-ID tie-break can select h_B everywhere, but h_A disappears from the active reference without a merge or recorded conflict. Determinism establishes convergence, not preservation or semantic correctness. A multi-value head can retain $\{h_A, h_B\}$; choosing one then requires an authorized merge or ordered compare-and-swap.
Status. PROVED IN MODEL; CatDB implementation OBSTRUCTED on P9.

11.5 CE-05: activation before a schema dependency

Let mapping M_2 refer to schema S_2. Replica A receives activate(M_2) before S_2. Immediate application either creates a dangling active mapping or interprets it against the wrong schema. The operation must declare S_2 and return Deferred until the dependency is present. Receipt-time sorting does not repair the missing version.
Status. PROVED IN MODEL for the typed precondition; CatDB implementation OBSTRUCTED.

11.6 CE-06: checkpoint truncation breaks duplicate safety

Replica A applies operation o, records its `OpId`, checkpoints, and deletes both the log entry and the deduplication marker. Replica B, partitioned before the checkpoint, later replays o. If the effect is not intrinsically idempotent, A applies it twice.

A safe checkpoint must retain an idempotence summary or prove that every replica and durable message source has crossed a stable frontier.
Status. PROVED IN MODEL; CatDB implementation OBSTRUCTED.

11.7 CE-07: tombstone collection resurrects deletion

Replica A observes add(x), then remove(x), and garbage-collects the remove tombstone. Replica B, partitioned since before the remove, later sends old state containing add(x). Merge without the tombstone restores x. Collection therefore needs a stable causal frontier or another design that retains the remove information required by its semantics.
Status. PROVED IN MODEL; CatDB implementation OBSTRUCTED.

11.8 CE-08: identity union and split are not one join

Let a, b, c be source identities. Replica A records $a \sim b$; replica B records $b \sim c$. Union followed by equivalence closure yields $a \sim c$. A later authoritative decision that $a \not\sim c$ cannot be expressed as another equivalence edge. It retracts or qualifies earlier closure and may invalidate derived results.

Identity-edge union can be a CRDT component. Governed identity history, reversal, and current authoritative equivalence are not thereby solved.
Status. PROVED IN MODEL; CatDB implementation OBSTRUCTED on P7/P13.

11.9 CE-09: mapping composition is not concurrent merge

Given

$$F : S \to T, \qquad G : T \to U,$$

$G \circ F : S \to U$ may be defined. $F \circ G$ is not typed unless $U = S$, and even then it need not equal $G \circ F$. A replicated set containing both mappings does not specify their intended pipeline order.
Status. PROVED IN MODEL by typing; bounded mapping model OPEN and replication OBSTRUCTED.

11.10 CE-10: scalar logical order invents a winner

Concurrent operations a and b can receive scalar logical times 7 and 9 without $a \to b$. Ordering by `(clock, replica_id)` applies a before b everywhere, but the winner follows from tie-breaking policy rather than causality. For concurrent schema replacements, that choice requires authority.
Status. ESTABLISHED EXTERNALLY for the logical-clock boundary, PROVED IN MODEL for the finite policy example, and CatDB implementation OBSTRUCTED.

11.11 CE-11: policy grant and revoke invert authority

Assume actor p initially lacks merge authority. Concurrently:

$$g = \text{grant}(p, \mathit{merge}), \qquad r = \text{revoke}(p, \mathit{merge}).$$

Grant followed by revoke denies; revoke followed by grant allows. A last-writer rule needs a trusted order and still encodes a policy decision. Union of the policy events can preserve the conflict, but it cannot decide whether an intervening merge was authorized.
Status. PROVED IN MODEL; CatDB implementation OBSTRUCTED.

11.12 CE-12: a hash chain forks

Two replicas share event hash h_0. During a partition they append e_A and e_B, each naming h_0:

$$h_A = H(h_0, e_A), \qquad h_B = H(h_0, e_B).$$

Both links are valid, so the result is a fork. A Merkle DAG can preserve both branches; one canonical chain requires ordering or later selection. Hash chaining makes represented links tamper-evident. It does not prove that events were not omitted or that one global order exists.
Status. PROVED IN MODEL; CatDB implementation OBSTRUCTED on P13.

Proposition 11.1 (No universal merge follows from the twelve histories)**.** *There is no single operation-order-insensitive interpretation of the unmodified sequential operations in CE-01, CE-02, CE-04, CE-09, CE-11, and CE-12 that both preserves their stated sequential meanings and identifies every delivery order.*

Proof. CE-01, CE-02, and CE-11 each display two orders with different results under the stated sequential semantics. CE-09 shows that one reverse order may be ill-typed. CE-04 shows that selecting one value without retaining the other loses a valid concurrent branch. CE-12 shows that two valid successors of one hash do not form one chain. Any order-insensitive interpretation must therefore change at least one operation's semantics, preserve multiple outcomes, reject an order, or introduce coordination. □

12 Replay, checkpoint, and recovery model

12.1 A history is more than an event sequence

A deterministic P10 history contains:

```
History = {
  replica_set: MembershipEpoch,
  initial_states: Map<ReplicaId, Digest>,
  operations: Map<OpId, Operation>,
  deliveries: Sequence<DeliveryEvent>,
  partitions: Sequence<PartitionEvent>,
  crashes: Sequence<CrashEvent>,
  durable_boundaries: Sequence<DurabilityEvent>,
  checkpoints: Sequence<CheckpointEvent>,
  expected_results: Map<StepId, ApplyResultClass>
}
```

The delivery schedule, duplicate events, and durability boundaries are part of the input. Wall-clock races are not an adequate fixture. Replaying the same canonical history must expose the same result class at every step.

12.2 Dependency buffering

When an operation's declared dependency is absent, the interpreter places the operation in `pending` and returns Deferred. Receipt of a missing version rechecks only operations indexed by that dependency. A dependency cycle, unknown membership epoch, failed authority check, or invalid precondition remains distinct from temporary absence.

Proposition 12.1 (Dependency readiness is not conflict resolution)**.** *If two operations a and b are both causally ready and their contexts are concurrent, readiness alone does not imply that they commute.*

Proof. In CE-02, both the rename and unit edit are defined against the same base state, so each can be causally ready after that base version is present. The calculation still gives different outcomes under the two orders. □

12.3 Deduplication and durable effects

The abstract apply protocol requires a durable relationship among:

1. canonical operation-log append;
2. state effect or conflict record;
3. deduplication identity;
4. response evidence; and
5. replay after crash.

A backend that writes the effect before the deduplication marker can apply an operation twice after a crash. A backend that writes the marker before the effect can lose the effect. An implementation may use one atomic database transaction, a write-ahead protocol, or a proved idempotent effect, but it must name the choice.

12.4 Checkpoint certificate

A candidate certificate records:

```
CheckpointCertificate = {
  checkpoint_id: SnapshotId,
  state_digest: Digest,
  causal_frontier: CausalContext,
  membership_epoch: VersionId,
  retained_dedup_summary: Digest,
  retained_tombstone_summary: Digest,
  acknowledged_replicas: Set<ReplicaId>,
  operation_classes: Set<OperationKind>,
  authority: AuthorityId,
  evidence: Set<EvidenceRef>
}
```

The certificate is not proof of safety until the membership, lagging-replica, durable-message, tombstone, and deduplication assumptions are established. Checkpoint and garbage collection remain OBSTRUCTED.

12.5 Tamper evidence

Canonical hashes can detect later alteration of represented bytes. A predecessor link can make a path tamper-evident. Neither property proves completeness, one global order, or protection against deletion. P13 must define what an evidence reference attests before P10 can use it in a recovery claim.

13 Implementation status and staged roadmap

13.1 Current result

There is no distributed CatDB implementation. This paper introduces no Rust, Haskell, Lean, fixture, backend, or benchmark artifact. The candidate operation algebra is an ENGINEERING HYPOTHESIS. The runtime claim that a bounded subset converges after duplicate, reordered, and partitioned delivery is OBSTRUCTED.

13.2 Gate R0: freeze prerequisites

Implementation begins only after:

1. P9 supplies accepted semantic commit, workspace, branch, and conflict types;
2. P13 supplies accepted operation and provenance identity;

3. P3 and P4 supply bounded schema and mapping version semantics;
4. P7 supplies identity authority and reversal semantics;
5. P6 supplies cursor and materialization semantics;
6. P11 names guarantees for the regimes P10 exposes; and
7. the crash, partition, membership, and delivery models are fixed.

Status. OBSTRUCTED.

13.3 Gate R1: language-neutral fixtures

The proposed closed, versioned artifacts are:

```
fixtures/distribution/operation-v1.schema.json
fixtures/distribution/history-v1.schema.json
fixtures/distribution/expected-v1.schema.json
fixtures/distribution/cases/*.json
```

Each fixture records the membership epoch, initial state digests, typed operations, causal dependencies, delivery schedule, partitions, healing, duplicates, crashes, restarts, expected step results, final canonical states, and conflict sets. It distinguishes `conflict`, `deferred`, `rejected`, and `unsupported` from a skipped test.
Status. OPEN after R0.

13.4 Gate R2: independent Haskell reference semantics

The proposed modules are:

```
haskell/catdb-reference/src/CatDB/Distribution/Types.hs
haskell/catdb-reference/src/CatDB/Distribution/Causal.hs
haskell/catdb-reference/src/CatDB/Distribution/Apply.hs
haskell/catdb-reference/src/CatDB/Distribution/History.hs
haskell/catdb-reference/src/CatDB/Distribution/Properties.hs
```

The interpreter should compute canonical states, conflicts, pending operations, and step results for deterministic schedules. Property tests should check conditional commutativity and idempotence and shrink failing histories. Shared generated decision logic with Rust would weaken the oracle and must be disclosed.
Status. OPEN; Haskell tests would be computational evidence, not theorem proof.

13.5 Gate R3: bounded Lean core

ID	Target statement	Required assumptions
DIST-L01	Version-vector dominance is a partial order	Finite replica identifiers and pointwise natural order

ID	Target statement	Required assumptions
DIST-L02	Pointwise maximum is associative, commutative, and idempotent	Same replica domain
DIST-L03	Immutable artifact-set union is a join-semilattice	Collision-free or integrity-rejecting identity
DIST-L04	Reapplying admitted immutable-artifact insertion is idempotent	Canonical bytes and exact digest equality
DIST-L05	Pairwise commuting deterministic operations give order-independent finite replay	Swapped pairs remain enabled and preserve preconditions
DIST-L06	Causal topological replays agree for the declared subset	Finite acyclic dependencies and concurrent commutativity
DIST-L07	Ordinary add and remove do not commute	CE-01 sequential set semantics
DIST-L08	Rename and unit edit do not commute	CE-02 state and functions
DIST-L09	Uniqueness is not closed under merge	CE-03 union and uniqueness invariant
DIST-L10	Checkpoint safety needs retained deduplication or stable frontier	Explicit replay and truncation model

The build must report axiom assumptions and reject `sorry`, `admit`, declaration-level axioms, and untracked classical assumptions. These theorems would not prove transport liveness, backend durability, CAP escape, or production performance.
Status. OPEN and blocked on a shared operation model.

13.6 Gate R4: Rust distribution crate

The first proposed Rust slice is:

```
crates/catdb-distribution/
  operation.rs
  causal.rs
  apply.rs
  conflict.rs
  replay.rs
  checkpoint.rs
  history.rs
  canonical.rs
```

Its initial admitted subset should include only immutable artifact addition, immutable commit or event addition with dependency buffering, durable `OpId` deduplication in the simulator, explicit multi-value conflicts, deterministic replay, and stable canonical serialization. Schema activation, mapping activation, identity reversal, policy replacement, membership, and checkpoint truncation should return `UNSUPPORTED_OPERATION_CLASS` until their protocols exist.
Status. OBSTRUCTED on R0 through R3.

13.7 Gates R5 through R8

R5 adds a deterministic simulator with controlled delivery, duplicate, delay, omission, partition, healing, crash, durable-state, membership, checkpoint, and seed events. R6 compares Haskell and Rust at every step. R7 adds backend failpoints before and after every durability boundary. R8 runs system acceptance commands:

```
cargo test -p catdb-distribution --test duplicate_reorder_replay
cargo test -p catdb-distribution --test partition_heal_convergence
cargo test -p catdb-distribution --test coordination_required
cargo test -p catdb-distribution --test operation_log_recovery
cargo test -p catdb-distribution --test stale_materialization_visibility
cargo test -p catdb-distribution --test causal_dependency_buffering
cargo test -p catdb-distribution --test schema_mapping_noncommutativity
cargo test -p catdb-distribution --test conflict_set_convergence
cargo test -p catdb-distribution --test checkpoint_frontier
cargo test -p catdb-distribution --test membership_epoch_rejection
cabal test catdb-reference
lake build
```

These commands are acceptance targets. They do not currently run because the referenced distribution artifacts do not exist.
Status. OBSTRUCTED.

14 Evaluation and falsification plan

14.1 Correctness before timing

The study begins with deterministic histories over 2, 3, 5, and 9 replicas. Operation streams range from 10 to 100,000 operations where feasible and mix data, schema, mapping, identity, policy, provenance, cursor, and materialization changes. Schedules include overlapping semantic scopes, multiple partitions, duplicate and reordered delivery, crash and restart, version skew, stale materializations, administrative operations, checkpoints, and intentionally unsafe garbage collection.

14.2 Baselines

ID	Baseline	Purpose
P10-BL01	Single serial interpreter	Canonical sequential result where one exists
P10-BL02	Consensus-ordered log	Strong-order correctness and coordination cost
P10-BL03	State-based full-state CRDT	Convergence and message-size comparison
P10-BL04	Delta-state CRDT	Incremental dissemination comparison
P10-BL05	Causal operation CRDT	Operation payload and causal-metadata comparison

ID	Baseline	Purpose
P10-BL06	Multi-value conflict register	Conflict preservation without semantic classification
P10-BL07	Last-writer-wins register	Negative baseline for hidden conflicts and lost proposals
P10-BL08	Event-log replay without concurrent merge	Negative baseline for event-sourcing overclaim
P10-BL09	Invariant-confluence classifier	Coordination-decision baseline
P10-BL10	Escrow bounded counter	Rights-based invariant baseline
P10-BL11	Staged F1-style schema profile	Compatibility baseline for supported online changes

Brand names are not equivalent baselines. Each implementation must disclose its operation, conflict, failure, and delivery semantics.

14.3 Exact correctness measurements

The correctness record includes:

- canonical-state disagreement after equal admitted delivery;
- conflicts missed, invented, or silently resolved;
- duplicate operations with an additional effect;
- lost accepted operations and forbidden causal inversions;
- invariant violations;
- stale results reported as current;
- checkpoint resurrection and replay errors;
- cross-language semantic disagreements; and
- minimized counterexample size.

Any nonzero value in an excluded category is a correctness failure, not a performance tradeoff.

14.4 Performance after correctness

Only a passing correctness profile may report local and coordinated latency, partition-time rejection or unavailability, time to convergence after healing, messages and bytes per operation, causal metadata size, pending buffer residence, conflict rate and record size, checkpoint and replay cost, CPU, memory, disk writes, network transfer, classification overhead, and throughput.

Performance runs need exact source revisions, environment capture, repeated runs, cold or warm labels, and uncertainty intervals. A fast finite history does not prove all histories, and a large random campaign does not replace a bounded proof.

14.5 Falsification anchors

The thesis fails for any admitted profile if:

1. a pair classified as commutative yields observably different states under the two orders;
2. an operation classified as idempotent changes state on replay;
3. a join violates associativity, commutativity, or idempotence;
4. a schema or mapping operation is accepted without its dependencies;
5. Rust, Haskell, and Lean use incompatible equality or error semantics;
6. replicas receive the same admitted set and do not converge;
7. partition healing loses an accepted operation;
8. a lagging replica resurrects collected state;
9. recovery reapplies an effect or loses deduplication;
10. a causal dependency appears in a forbidden order;
11. conflict sets differ after equal delivery;
12. membership or checkpointing admits a history excluded by the proof;
13. a noncommutative semantic transition silently chooses a winner;
14. uniqueness, authority, or quota is violated by a coordination-free classification;
15. an available API blocks indefinitely on an unreachable quorum;
16. a linearizable API returns mutually incompatible completed results;
17. escrow rights are duplicated, lost, or overspent;
18. the paper reports performance without raw histories and environment evidence;
19. a theorem omits a delivery, failure, finiteness, or equality assumption used by its proof;
20. category theory is used to claim escape from CAP or consensus;
21. an external system result is reported as CatDB behavior; or
22. a last-writer-wins result is called conflict-free without recording the discarded value.

Every failure should become a minimized immutable regression fixture.

15 Limitations

There is no distributed runtime to evaluate. The paper can reject invalid universal claims and state a typed design, but it cannot show that the proposed boundary is practical. No history has passed through a CatDB network. No crash has tested a durable deduplication path, and no benchmark has measured the cost of causal or semantic metadata.

The finite replica model is deliberately small. Dynamic membership, long-lived partitions, replica retirement, causal-context compaction, and checkpoint certification are unresolved. The initial failure model excludes Byzantine behavior. Content hashes are treated as integrity identities under a collision-rejecting assumption; the paper does not supply a cryptographic threat model.

The operation matrix also depends on unfinished semantic work. P9 must say what a branch head and semantic commit mean. P13 must say what event identity and provenance evidence attest. P7, P6, and P11 affect identity reversal, materialization state, and consistency contracts. Those dependencies can change the payloads or move an operation between candidate regimes.

Explicit conflicts do not resolve themselves. A multi-value result can converge and still leave a difficult human or policy decision. A controlled study is needed to learn whether typed conflict records improve diagnosis; richer metadata by itself is not evidence.

16 Open questions

1. What is the smallest useful semantic scope language that is stronger than byte or key overlap but still decidable and cheap?
2. Which immutable schema and mapping additions admit a compositional noninterference proof?
3. Can compatibility witnesses support staged rollouts across mapping, provenance, and authority versions as well as schemas?
4. Which causal-context representation supports dynamic membership without weakening the replay proof?
5. How should a checkpoint certificate account for offline replicas and durable messages outside the service?
6. Which invariants admit escrow or ownership partitioning, and which require one ordered authority?
7. Can a conflict record remain compact while preserving both proposals, dependencies, policy, and provenance?
8. How should redaction interact with hash-linked replay evidence and conflict reproduction?
9. Can Lean, Haskell, and Rust share a fixture vocabulary without sharing decision logic that undermines oracle independence?

10. What false-positive and false-negative rates result from semantic operation classification compared with manual invariant-confluence or RedBlue classification?
11. When is deterministic last-writer-wins an explicitly authorized policy rather than a hidden default?
12. Which P11 guarantees should be exposed per operation without collapsing them into one platform label?

17 Conclusion

A distributed CatDB design has to classify operations before it can promise a merge. Immutable content addition, causal dependency delivery, explicit conflict preservation, escrow, ordered publication, and administrative barriers solve different problems. The typed envelope proposed here records those differences and gives each claim a test.

The counterexamples need only a few operations. Ordinary add and remove do not commute. Two locally valid inserts can break uniqueness, and a branch tie-break can erase a valid proposal. Mapping composition has a direction. A logical clock is not causality. A hash chain can fork. Categorical language does not change any of these results.

The next credible result is a narrow fixture-backed slice: immutable artifacts, dependency buffering, durable duplicate identity, explicit multi-value conflicts, and deterministic replay. An independent executable model, a bounded formal model, Rust, and backend recovery histories must agree on that slice. Until they do, the design remains an ENGINEERING HYPOTHESIS, its testable components remain OPEN, and its runtime claims remain OBSTRUCTED.

A Exact nonclaims

The following statements are part of the paper's claim boundary.

1. CatDB does not currently implement distributed replication.
2. CatDB has no accepted CRDT operation subset.
3. CatDB has no accepted causal delivery or logical-clock implementation.
4. CatDB has no accepted consensus, membership, checkpoint, or garbage-collection protocol.
5. CatDB has no measured partition-time availability, convergence time, throughput, or latency.
6. Category theory does not eliminate CAP-style tradeoffs.
7. Category theory does not make arbitrary schema or mapping changes commute.
8. Functor composition is not a concurrent merge law.
9. Event sourcing does not by itself guarantee convergence.

10. Append-only storage does not by itself guarantee idempotence or causal order.
11. A Lamport timestamp is not a concurrency detector, wall clock, or freshness proof.
12. Vector clocks do not choose a winner for concurrent semantic updates.
13. Causal consistency is not serializability or linearizability.
14. Deterministic last-writer-wins resolution is not proof of correct business meaning.
15. Replica convergence does not imply invariant preservation.
16. Replica convergence does not imply that source data or materializations are fresh.
17. A tombstone does not repair mappings, queries, or policies invalidated by schema deletion.
18. A content hash does not establish one global append order.
19. A hash chain does not prevent event omission or concurrent forks.
20. An immutable version set does not determine one active schema, mapping, or branch head.
21. Exactly-once delivery is not assumed.
22. Network partitions are not assumed to heal within a fixed time.
23. Byzantine faults, malicious replicas, and cryptographic consensus are outside the initial P10 profile.
24. Haskell property tests would be computational evidence, not theorem proof.
25. Lean proofs over a finite model would not prove transport liveness, backend durability, or production performance.
26. External CRDT, database, and consensus systems are precedent, not CatDB implementation evidence.
27. No universal consistency label describes the proposed platform.
28. No automatic conflict resolver may silently infer business authority.

B Claim-status ledger

ID	Claim	Evidence	Status
P10-C01	State-based CRDT convergence has semilattice and inflationary-update conditions	Literature	ESTABLISHED EXTERNALLY
P10-C02	Operation-based convergence needs effect commutativity and delivery assumptions	Literature	ESTABLISHED EXTERNALLY
P10-C03	Atomic consistency and availability conflict during partitions in the Gilbert-Lynch model	Theorem	ESTABLISHED EXTERNALLY
P10-C04	The inspected repository has no P10 distribution implementation	Local audit	OBSTRUCTED implementation

ID	Claim	Evidence	Status
P10-C05	Digest-addressed immutable insertion can be an idempotent join component under integrity assumptions	Design	OPEN
P10-C06	The whole distributed state should not receive one unproved join	Safety restriction	ENGINEERING HYPOTHESIS
P10-C07	The stated rename and unit edit do not commute	Counterexample	PROVED IN MODEL
P10-C08	Mapping composition is not an order-independent concurrent merge	Typing	PROVED IN MODEL
P10-C09	Causal dependencies can buffer mapping activation until its schema arrives	Design	OPEN
P10-C10	Scalar Lamport clocks cannot detect concurrency by themselves	Literature	ESTABLISHED EXTERNALLY
P10-C11	Unique-key inserts are not invariant-confluent under set union	Counterexample	PROVED IN MODEL
P10-C12	A bounded CatDB subset converges through duplicate, reorder, and partitions	Runtime	OBSTRUCTED
P10-C13	Noncommutative operations always conflict or use named coordination	Runtime	OBSTRUCTED
P10-C14	Recovery preserves operations and deduplication facts	Runtime	OBSTRUCTED
P10-C15	Causal metadata overhead is acceptable	Performance	OBSTRUCTED
P10-C16	Semantic classification reduces coordination or improves throughput	Performance	OBSTRUCTED
P10-C17	Lean can prove the bounded core	Formal target	OPEN
P10-C18	Haskell can provide an independent finite-history oracle	Computational target	OPEN
P10-C19	Rust and backend histories match the reference model	Runtime	OBSTRUCTED
P10-C20	Category theory eliminates CAP-style tradeoffs	Broad claim	REJECTED
P10-C21	Event sourcing alone guarantees convergence	Broad claim	REJECTED
P10-C22	Convergence implies correct business meaning	Broad claim	REJECTED

C Reproducibility contract

C.1 Manuscript reproduction

The manuscript source and bibliography are the only publication artifacts created for this author pass. A clean build copies those two files to a new temporary directory and runs:

```
latexmk -cd -pdf -interaction=nonstopmode -halt-on-error paper.tex
```

The repository contains no generated PDF, cover, auxiliary file, review artifact, distribution source, or benchmark result from this pass. The paper-local `reproducibility.md` records the exact temporary build directory, tool versions, hashes, warning inventory, page count, and all-page render inspection.

C.2 Future execution bundle

Every future distributed run archives:

```
benchmarks/runs/<run_id>/
  pre-registration.md
  source.json
  build.json
  environment.json
  baseline-manifest.json
```

```
history-schema.json
histories/
delivery-schedules/
raw-observations/
canonical-states/
conflicts/
minimized-failures/
statistics.json
provenance.json
```

Seeds, source revisions, membership epochs, delivery schedules, crash points, durability boundaries, canonical states, and minimized failures are required. Convergence and invariant checks are exact per finite history. Performance claims require repeated runs and uncertainty reporting.

References

[1] Mustaque Ahamad, Gil Neiger, James E. Burns, Prince Kohli, and Phillip W. Hutto. Causal memory: Definitions, implementation, and programming. *Distributed Computing*, 9(1):37–49, 1995.

[2] Deepthi Devaki Akkoorath, Alejandro Z. Tomsic, Manuel Bravo, Zhongmiao Li, Tyler Crain, Annette Bieniusa, Nuno Preguiça, and Marc Shapiro. Cure: Strong semantics meets high availability and low latency. In *2016 IEEE 36th International Conference on Distributed Computing Systems*, pages 405–414, 2016.

[3] Paulo Sérgio Almeida, Ali Shoker, and Carlos Baquero. Delta state replicated data types. *Journal of Parallel and Distributed Computing*, 111:162–173, 2018.

[4] Tom J. Ameloot, Frank Neven, and Jan Van den Bussche. Relational transducers for declarative networking. *Journal of the ACM*, 60(2):15:1–15:38, 2013.

[5] Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Highly available transactions: Virtues and limitations. *Proceedings of the VLDB Endowment*, 7(3):181–192, 2013.

[6] Peter Bailis, Alan Fekete, Michael J. Franklin, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Coordination avoidance in database systems. *Proceedings of the VLDB Endowment*, 8(3):185–196, 2014.

[7] K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.

[8] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner:

Google's globally-distributed database. In *10th USENIX Symposium on Operating Systems Design and Implementation*, pages 251–264, 2012.

[9] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, pages 205–220, 2007.

[10] Colin J. Fidge. Logical time in distributed computing systems. *Computer*, 24(8):28–33, 1991.

[11] Martin Fowler. Event sourcing. Author-maintained pattern description, 2005. Accessed 2026-07-23.

[12] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):51–59, 2002.

[13] Victor B. F. Gomes, Martin Kleppmann, Dominic P. Mulligan, and Alastair R. Beresford. Verifying strong eventual consistency in distributed systems. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):109:1–109:28, 2017.

[14] Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The dangers of replication and a solution. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 173–182, 1996.

[15] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, 1990.

[16] Martin Kleppmann and Alastair R. Beresford. A conflict-free replicated JSON datatype. *IEEE Transactions on Parallel and Distributed Systems*, 28(10):2733–2746, 2017.

[17] Martin Kleppmann, Victor B. F. Gomes, Dominic P. Mulligan, and Alastair R. Beresford. OpSets: Sequential specifications for replicated datatypes. *CoRR*, abs/1805.04263, 2018. Mechanized artifact: https://www.isa-afp.org/entries/OpSets.html.

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

[19] Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues. Making geo-replicated systems fast as possible, consistent when necessary. In *10th USENIX Symposium on Operating Systems Design and Implementation*, pages 265–278, 2012.

[20] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. Don't settle for eventual: Scalable causal consistency for wide-area storage with COPS. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 401–416, 2011.

[21] Friedemann Mattern. Virtual time and global states of distributed systems. In Michel Cosnard, Yves Robert, Patrice Quinton, and Michel Raynal, editors, *Parallel and Distributed Algorithms*, pages 215–226. North-Holland, 1989.

[22] Patrick E. O'Neil. The escrow transactional method. *ACM Transactions on Database Systems*, 11(4):405–430, 1986.

[23] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference*, pages 305–319, 2014.

[24] Ian Rae, Eric Rollins, Jeffrey Shute, Sukhdeep Sodhi, and Radek Vingralek. Online, asynchronous schema change in F1. *Proceedings of the VLDB Endowment*, 6(11):1045–1056, 2013.

[25] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. A comprehensive study of convergent and commutative replicated data types. Technical Report RR-7506, INRIA, 2011.

[26] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Stabilization, Safety, and Security of Distributed Systems*, volume 6976 of *Lecture Notes in Computer Science*, pages 386–400. Springer, 2011.

[27] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 172–183, 1995.

[28] Yannis Velegrakis, Renée J. Miller, and Lucian Popa. Mapping adaptation under evolving schemas. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 584–595, 2003.

[29] William E. Weihl. Commutativity-based concurrency control for abstract data types. *IEEE Transactions on Computers*, 37(12):1488–1505, 1988.