Consistency Models for Distributed Semantic Databases: An Operation-by-Operation Contract for CatDB

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

23 July 2026

Abstract

A distributed semantic database cannot be described honestly by one consistency adjective. A local transaction may be serializable while a remote materialization is stale. Provenance assertions may merge by set union while an active mapping, canonical identity decision, or branch head needs one serialized authority. A replayed message may be harmless inside one sink yet repeat an external side effect. These distinctions matter in CatDB, where schemas, mappings, migrations, identities, provenance, materializations, and workspace references are proposed as versioned executable objects.

This paper specifies a provisional operation-level consistency contract. A profile separates local isolation, replica visibility, session guarantees, convergence, semantic-version compatibility, replay effects, freshness, validation and authority, and historical retention. Thirty candidate operations are classified against six mechanisms: local transactions, causal broadcast, consensus or a serialized authority, idempotent replay, conflict resolution, and asynchronous validation. The classification permits available creation of immutable proposals and monotone evidence under stated conditions. It requires coordination or an explicit conflicting state for single-valued activation, canonical identity, membership, and irreversible retention decisions. Schema, mapping, and materialization consistency receive separate contracts. CAP and PACELC remain physical limits. CALM, CRDT, and invariant-confluence results apply only to a frozen admitted fragment.

The result is a research specification, not an implemented consistency model. The repository has no distribution runtime, causal transport, consensus-backed reference store, distributed checkpoint, consistency-profile type, fault harness, Haskell history oracle, Lean consistency library, or benchmark result. Paper 10's internally accepted research manuscript now contains a typed operation algebra and coordination classification, but it is not an accepted implementation interface and its executable dependencies remain unresolved. All CatDB runtime, convergence, availability, recovery, and performance claims in this paper are therefore OBSTRUCTED or OPEN. The paper's bounded propositions concern only the definitions presented here.

Contents

1 Introduction

1.1 A consistency failure that starts with reasonable edits

Consider two teams working from the same semantic commit. One adds a provenance annotation, approves two customer records as the same person, and publishes a refreshed revenue view. The other receives the annotation late, rejects the same identity match, strengthens a uniqueness constraint, and changes the active mapping from gross revenue to net revenue. A network partition lets both teams continue working. When the partition heals, every message is eventually delivered, some more than once and in a different order.

Calling this state "eventually consistent" does not answer the useful questions. The annotation may merge cleanly. The identity decisions cannot both become the one canonical answer without an authority or an explicit conflict. The new constraint may reject data that the other side accepted. The revenue view may be current for its source cursor but built under the old mapping. A compiled query plan may still run while interpreting a path with obsolete meaning.

Users need a concrete contract. Which actions remain available during the partition? Which block or fail? Which returned values are tentative or stale? Which messages must retain causal order? Which operations can be replayed without repeating an effect? Which conflict needs an administrator? What does "caught up" mean after the replicas reconnect?

This paper turns those questions into an operation matrix. It does not claim that the matrix removes the tradeoffs.

1.2 Research question

For each CatDB operation o, which combination of

$$\mathsf{LocalTxn}(o), \quad \mathsf{Causal}(o), \quad \mathsf{Consensus}(o),$$
$$\mathsf{Idempotent}(o), \quad \mathsf{Reconcile}(o), \quad \mathsf{AsyncValidate}(o)$$

is required to preserve a named invariant and observation contract under a stated fault model?

This question differs from asking which database consistency model is best. An append-only evidence assertion and a globally active mapping do not need the same mechanism. Neither does a local materialization checkpoint and a multi-source write-through.

1.3 Bounded thesis

The paper's thesis is an engineering hypothesis:

> A useful CatDB consistency model can be expressed as a versioned product of independent dimensions plus an operation policy. Immutable and monotone evidence may often use causal dissemination and idempotent replay. Single-valued authoritative references, noncommutative semantic changes, uniqueness-sensitive decisions, membership changes, and irreversible retention decisions require a serialized authority or an explicit conflict. Materialized reads require an additional semantic-version, frontier, and completeness contract.

The thesis is falsifiable. It fails if the profile cannot distinguish materially different histories, if an operation classified as coordination-free violates its invariant, if a replay repeats an admitted effect, if a strict read hides a missing causal dependency, or if the profile silently weakens during a partition.

1.4 Contributions

The provisional paper contributes:

1. a nine-dimensional consistency profile for semantic database operations and reads;
2. a 30-operation matrix over the six required mechanisms;
3. distinct contracts for schema activation, mapping activation, identity authority, materialization freshness, validation, replay, and retention;
4. bounded model results for idempotent replay, commuting operations, strict validation status, and linearizable compare-and-swap;
5. CAP, PACELC, CALM, CRDT, and invariant-confluence boundaries that prevent category-theoretic overclaiming;
6. executable history families and falsification criteria; and
7. an implementation and formalization audit that leaves every absent system behavior OPEN or OBSTRUCTED.

1.5 Current status

The repository's accepted executable slice covers a much smaller schema/path/mapping core. It does not provide distributed consistency. Paper 10's internally accepted manuscript proposes typed replicated operations and coordination regimes, but manuscript acceptance does not create a runtime or compiled interface. P11 imports its exact research contract as a design dependency only. The matrix must be revalidated when that algebra, state model, authority scopes, and delivery assumptions become executable interfaces.

No AGY peer review is included in this drafting pass. The manuscript status is OBSTRUCTED rather than accepted.

1.6 Organization

Section 2 fixes the evidence boundary. Section 3 presents the operation matrix. Section 4 gives named profiles. Sections 5 to 7 specify semantic-version contracts. Section 9 states the distributed-systems limits. Section 10 proves bounded model properties. Sections 12 and 13 describe implementation and falsification work. Sections 15 to 17 close with limitations, open questions, and the current result. The appendices audit the repository and prior art, state the running history and formal model, and define the six mechanisms in detail (Sections A to E).

2 Evidence vocabulary and repository boundary

2.1 Evidence labels

Label	Meaning in this paper
SUPPORTED BY PRIOR LITERATURE	A primary paper, standard, or official system artifact establishes the result under its own assumptions. It does not establish a CatDB behavior.
PROVED IN MODEL	A proposition follows from the finite definitions in this paper. It does not verify a runtime, network, backend, or recovery path.
ENGINEERING HYPOTHESIS	A proposed CatDB behavior requiring implementation and experiment.
OPEN	A definition, proof, test, implementation, or review remains incomplete.
OBSTRUCTED	A prerequisite semantic or executable boundary is absent, so the claim cannot yet be tested honestly.

3 Operation-by-operation matrix

The matrix is a provisional input to P10/P11 co-design. It describes requirements rather than current implementation. Every row is OPEN unless marked OBSTRUCTED. The strict outcome is the minimum observation contract for the row.

Table 2: Provisional CatDB consistency matrix.

ID	Operation	Strict outcome	Local	Causal	Order	Replay	Conflict	Async	Status
O01	Local instance transaction	Backend-declared isolation and atomic local commit.	REQ	NO	NO	COND	COND	NO	OPEN
O02	Create immutable semantic artifact	The same ID denotes the same bytes and semantic kind; collision rejects.	REQ	COND	NO	REQ	REQ	COND	OPEN
O03	Append monotone provenance or evidence assertion	Union preserves every distinct assertion.	REQ	REQ	NO	REQ	COND	COND	OPEN
O04	Retract or supersede assertion	History remains; dependent reads observe the status relation.	REQ	REQ	COND	REQ	REQ	COND	OPEN
O05	Publish semantic commit object	The immutable commit is complete and all referenced artifacts are available or explicitly remote.	REQ	REQ	NO	REQ	REQ	REQ	OBSTRUCTED
O06	Move branch reference by compare-and-swap	At most one success from one predecessor; strict readers see one authoritative ref.	REQ	INSUF	REQ	REQ	REQ	NO	OBSTRUCTED
O07	Create additive schema revision	Immutable revision validates against its parent.	REQ	REQ	NO	REQ	REQ	COND	OPEN
O08	Activate additive schema revision	Reads and writes follow the declared compatibility window.	REQ	REQ	REQ/ALT	REQ	REQ	COND	OBSTRUCTED
O09	Create destructive, type, or cardinality schema revision	Revision remains inactive; loss and compatibility are classified.	REQ	REQ	NO	REQ	REQ	REQ	OPEN
O10	Activate destructive schema revision	No admitted writer corrupts shared data; incompatible writers reject or isolate.	REQ	INSUF	REQ	REQ	REQ	NO	OBSTRUCTED
O11	Create mapping revision	Immutable typed mapping records coverage, loss, and equation checks.	REQ	REQ	NO	REQ	REQ	COND	OPEN
O12	Activate default mapping	One authoritative revision exists per single-default scope, or the conflict is explicit.	REQ	INSUF	REQ/ALT	REQ	REQ	NO	OBSTRUCTED
O13	Strengthen global constraint	No validated write violates the active constraint.	REQ	INSUF	REQ	REQ	REQ	COND	OBSTRUCTED
O14	Append identity candidate or evidence	Evidence remains available without asserting canonical identity.	REQ	REQ	NO	REQ	COND	REQ	OPEN
O15	Confirm identity equivalence	An authority-valid decision is current; strict readers do not see contradictory canonical identity.	REQ	REQ	REQ/ALT	REQ	REQ	COND	OBSTRUCTED
O16	Split or reverse identity decision	History remains, dependents invalidate, and one authoritative current decision exists.	REQ	REQ	REQ	REQ	REQ	COND	OBSTRUCTED
O17	Apply materialization delta and advance cursor	Effect and cursor agree atomically inside the sink boundary.	REQ	COND	COND	REQ	REQ	NO	OPEN

Continued on next page

ID	Operation	Strict outcome	Local	Causal	Order	Replay	Conflict	Async	Status
O18	Publish materialization version	Readers can identify semantic vector, source frontier, completeness, and validation.	REQ	REQ	COND	REQ	REQ	COND	OBSTRUCTED
O19	Serve bounded-staleness materialized read	The declared bound is measured and enforced; otherwise the request rejects or downgrades explicitly.	REQ	COND	COND	NO	NO	INSUF	OPEN
O20	Serve partial-source query	Missing sources and reconciliation consequences are explicit.	REQ	COND	NO	COND	REQ	REQ	OPEN
O21	Cache or reuse compiled plan	The cache key includes every semantic and capability revision affecting correctness.	REQ	REQ	NO	REQ	REQ	NO	OPEN
O22	Change authority or semantic policy	One authorized active policy exists per scope and audit history remains.	REQ	REQ	REQ	REQ	REQ	NO	OBSTRUCTED
O23	Publish validation result	The result binds exact subject and input revisions; stale validation cannot approve new state.	REQ	REQ	COND	REQ	REQ	NO	OPEN
O24	Create global checkpoint	The cut is consistent for declared replicas and artifacts; obligations are named.	REQ	REQ	COND	REQ	REQ	REQ	OBSTRUCTED
O25	Garbage collect history or deduplication state	No live replica, branch, replay, provenance, or historical query becomes falsely reconstructible.	REQ	INSUF	REQ	REQ	REQ	NO	OBSTRUCTED
O26	Change replica membership or configuration	No split authority or unsafe quorum transition occurs.	REQ	INSUF	REQ	REQ	REQ	NO	OBSTRUCTED
O27	Allocate globally unique human-visible key	No duplicate committed key exists in the declared scope.	REQ	INSUF	REQ	REQ	REQ	COND	OPEN
O28	Multi-source write-through	Target constraints and authority hold; outcome and compensation are explicit.	DEFER	DEFER	DEFER	REQ	REQ	DEFER	OBSTRUCTED
O29	Historical query replay	Retained inputs and semantic vector reproduce a normalized equivalent result.	REQ	NO	NO	REQ	REQ	NO	OBSTRUCTED
O30	Merge workspace branches	The candidate is isolated, conflicts are typed, and the target ref moves once after validation.	REQ	REQ	REQ	REQ	REQ	REQ	OBSTRUCTED

3.1 Reading the matrix

The "Order" column names consensus or an equivalent serialized authority. It does not require a geographically replicated Raft group for every row. A single administrator may serialize one scope, with different availability and fault tolerance. A NO entry does not remove local locking, persistence, dependency fetches, or connector coordination.

Causal order does not make concurrent operations commute. Idempotent replay does not provide an exactly-once external effect unless the effect and deduplication record share an atomic boundary or an end-to-end compensation protocol. An ALT cell requires the tentative or conflicting state to be part of the API. Silently choosing a winner is not an alternative.

3.2 Creation and activation

Rows O02, O07, O09, and O11 expose a recurring distinction. Creating an immutable proposal can remain available because it does not select one global interpretation. Activation rows O08, O10, and O12 are different. They change which semantics strict readers and writers use. A branch-local workspace may keep alternatives without consensus, but one global default needs a serialized authority or must return a conflict set.

3.3 Evidence and authority

Rows O03 and O14 permit monotone evidence append under narrow assumptions. Rows O15, O16, and O22 introduce an authority-sensitive current decision. The system can retain competing evidence while exposing one current result under a strict profile. Vector similarity may generate an identity candidate; it cannot supply authority.

3.4 Data plane and control plane

Rows O17 through O21 separate local data-plane execution from semantic publication. A sink can atomically couple a delta with a cursor without coordinating a global branch head. Publishing one current materialization or guaranteeing a cross-source freshness bound may require a stronger path. Policy, membership, checkpoint, and garbage-collection rows are control-plane operations whose failure can corrupt later authority or reproducibility.

4 Named consistency profiles

4.1 `local-transactional`

This profile declares local backend isolation and durability. It gives no distributed visibility guarantee. Retries need stable request IDs when effects are not naturally idempotent, and the semantic revision remains fixed for the transaction.

Candidate uses include artifact creation, local workspace edits, schema validation, and one-sink materialization updates. The profile cannot support a replica-convergence claim.

4.2 `causal-evidence`

This profile delivers causal parents before dependents. Concurrent monotone assertions may arrive in either order. It requires immutable identities, idempotent application, and eventual delivery to healthy replicas under the stated network assumption. It implies no single authoritative winner.

Candidate uses include provenance assertions, identity match candidates, validation observations, and nonexclusive annotations. The profile is unsafe for destructive mutation, uniqueness, active-reference selection, and canonical identity confirmation without another mechanism.

4.3 `available-tentative`

An operation may complete during a partition, but the result is explicitly tentative. Concurrent alternatives may coexist. Later validation or reconciliation may reject, compensate, or supersede them. Strict readers exclude the state, and no bounded recency is implied.

Candidate uses include branch-local schema or mapping proposals, possible identity matches, provisional materializations, and exploratory workspace interpretations. The profile does not describe a globally committed update.

4.4 `authoritative-linearizable`

One current value exists for the named authority object. Successful operations respect real-time order. Compare-and-swap or a consensus-backed state machine serializes updates. Minority partitions reject or block. Stale reads, if offered, use another API and profile.

Candidate objects include the `main` branch reference, active schema or mapping pointer, authority policy, canonical identity decision, membership, and authoritative checkpoint. Evaluation must measure quorum and cross-region latency.

4.5 `validated-checkpoint`

A consistent cut names replicas, operation frontiers, semantic revisions, materializations, validation results, unresolved conflicts, and retention policy. All dependencies exist before a serialized publication step. Garbage collection occurs only after the relevant obligations are discharged.

Candidate uses include reproducible release snapshots, global administrative checkpoints, and provenance-preserving archival boundaries.

4.6 `bounded-materialized-read`

The request and materialization semantic vectors match or satisfy a validated compatibility relation. Each source frontier is reported. Staleness and completeness bounds are explicit and enforced. Missing or late sources are visible, and the fallback or rejection policy is declared.

This profile is independent of causal consistency and eventual convergence. A causally ordered view can be too stale; a converged result can still use the wrong mapping revision.

5 Schema-version consistency

5.1 Creation, validation, activation

Schema evolution has three operations:

1. create an immutable proposed revision;
2. validate it against named predecessors, mappings, data, and writers; and
3. activate it for a named authority scope.

Creation can often remain available. Activation may require coordination even when the revision is additive, because an active default changes the context for writers, validators, caches, and readers.

Definition 5.1 (Schema-version consistency)**.** A request has schema-version consistency for revision s when every interpreted object, path, constraint, and value domain uses s or a named compatible revision; every mapping and migration is valid for those revisions; admitted writers cannot create a representation that admitted readers misinterpret; cached plans and materializations carry the relevant revision; validation results bind that exact revision; and the response reports the revision actually used.

5.2 Mixed versions

F1 demonstrates that asynchronous distributed schema change can be safe under a formal transition protocol and a bounded skew assumption [25]. It also shows why immutability is not enough. Ordinary mixed-version sequences can corrupt data.

A candidate CatDB transition is

$$\mathsf{proposed} \to \mathsf{validated} \to \mathsf{staged} \to \mathsf{active} \to \mathsf{deprecated} \to \mathsf{retired}.$$

Each transition must identify the authority, admitted readers and writers, mapping compatibility, enforced constraints, reusable materializations, rollback condition, replica acknowledgements, and representation-retention rule.

5.3 Forbidden schema histories

A correct strict profile must reject or prevent:

- a write under `customer@8` interpreted under `customer@7` with different meaning;
- a globally valid uniqueness constraint while a partition accepts conflicting writes;
- schema activation before required mappings and migrations are available;
- an old writer outside the safe mixed-version window;
- a cached plan for a removed path running under the new revision;
- a validation result for revision 7 approving revision 8; and

- rollback to a schema whose required representation was discarded.

Evidence status. OBSTRUCTED. P3 and P10 have not supplied an accepted mixed-version runtime protocol..

6 Mapping consistency

6.1 Revision context

Let $M^{v}_{s \to d}$ be mapping revision v from source schema revision s to domain schema revision d.

Definition 6.1 (Mapping-consistent request)**.** A request is mapping-consistent when it uses exactly M^{v} or a validated composition containing it; its source and target schemas are compatible; its coverage, loss, ambiguity, identity, and authority policies are applied; its plans and materializations are keyed by the complete mapping dependency vector; partial or ambiguous results are reported; and results from incompatible mapping revisions are not combined without an explicit reconciliation rule.

Schema evolution can invalidate mappings, and adapting affected mappings is a semantic problem rather than a text edit [30]. CATDB adds a proposed distributed activation concern, not a new discovery that mappings evolve.

6.2 Concurrent interpretations

Two workspaces may retain M_{gross} and M_{net} as branch-local interpretations. This does not make both the one authoritative finance mapping. The consistency contract chooses a serialized default, an explicit conflict, or an explicit workspace, tenant, source, or consumer scope.

When active mapping changes, the system classifies compiled queries, plan-cache entries, materialization definitions and data, provenance explanations, reconciliation assumptions, write-back policies, compatibility views, historical queries, and downstream mappings. Each dependent is:

- reusable under a proved or validated rule;
- adaptable through a named transformation;
- stale but readable with the old revision label;
- invalid and hidden from strict reads; or
- unresolved.

6.3 Forbidden mapping histories

The strict profile forbids:

- two replicas answering the same request under different default mappings while reporting one current revision;

- reuse of a gross-revenue plan after net-revenue activation;
- data revision advancing without mapping provenance;
- a partial mapping silently becoming total after replay;
- a mapping conflict resolved by wall-clock order alone; and
- activation before the target schema reaches an admitted transition state.

Evidence status. OBSTRUCTED. Mapping revisions exist only in the smaller non-distributed research slice; no activation protocol or distributed cache invalidation exists..

7 Materialization consistency and freshness

7.1 Four independent questions

A materialized result answers:

1. Was it computed under the requested schema, mapping, query, identity, provenance, and policy revisions?
2. Which source changes are included?
3. Which expected sources or partitions are missing?
4. Is the version tentative, validated, current, superseded, or invalidated?

Freshness does not replace semantic validity, completeness, or validation.

Definition 7.1 (Materialization version)**.** A materialization version is

$$\mu = \langle \mathit{definition}, \rho, \mathbf{f}, \mathit{coverage}, \mathit{watermarks}, \mathit{publishedAt}, \mathit{validation}, \mathit{recoveryState} \rangle.$$

The publication time records when the version became visible. It does not prove that every source event with an earlier wall-clock time is present.

7.2 Visibility profiles

A `strict-current` read requires a matching semantic vector, the declared source frontier, no unresolved invalidation, and complete validation. Otherwise it blocks, recomputes, or fails.

A `bounded-stale` read supplies a measured staleness definition and bound, semantic compatibility, and a missing-source policy. If the bound cannot be certified, the request fails or downgrades only with caller consent.

A `best-effort` read may return available state, but attaches every frontier, missing source, semantic revision, and validation caveat. It makes no global-current claim.

A `historical` read binds an immutable checkpoint or retained frontier. Later data is intentionally excluded and reconstructibility is checked.

7.3 Cursor and effect

For source event e_k, materialization state m, and cursor c, the transition

$$(m, c) \xrightarrow{e_k} (m', c')$$

must make the effect and cursor durable together or use a recoverable intent protocol. If c' is durable without m', recovery can skip data. If m' is durable without c', replay can duplicate an effect.

This is a local atomicity requirement. It does not create an atomic transaction with the source unless source and sink participate in a named distributed protocol. MillWheel and PostgreSQL logical decoding provide framework-specific evidence, not a generic exactly-once theorem [3, 24].

7.4 Freshness dimensions

TACT demonstrates that numerical error, order error, and staleness are distinct dimensions [31]. A CatDB materialization evaluation should measure event-time lag, ingestion lag, processing lag, publication lag, source-cursor distance, order uncertainty, missing-source count, late-data allowance, invalidation age, semantic-version lag, and validation lag.

7.5 Forbidden materialization histories

- A cursor advances beyond an unapplied event.
- Retry applies a nonidempotent delta twice.
- A globally fresh read omits an unavailable source without disclosure.
- A current materialization uses a superseded identity decision.
- A new mapping reuses old data without an adaptation rule.
- A watermark excludes admissible late data while claiming completeness.
- Two publishers overwrite one another without version or CAS checks.
- A stale plan computes a new version under an old semantic vector.

Evidence status. OBSTRUCTED. P6 specifies a candidate checkpoint contract but the repository has no materialization runtime or distributed frontier service..

8 Identity, provenance, policy, and reads

8.1 Identity authority

Identity evidence may accumulate by set union, but an authoritative identity interpretation includes known equivalent, probable equivalent, possible match, known different, contradiction, superseded, reversed, and unresolved authority-conflict states.

A canonical-identity result reports the reconciliation profile, decision checkpoint, authority, confidence policy, unresolved contradictions, tentative-decision policy, invalidation status after reversal, and provenance for accepted equivalences. Concurrent $\mathsf{confirmedSame}(x, y)$ and $\mathsf{confirmedDifferent}(x, y)$ cannot be collapsed by similarity, arrival order, or a bare timestamp.

8.2 Provenance state

Provenance assertions are candidates for monotone append, but provenance answers also depend on retraction, source retention, redaction, trust, mapping and migration revisions, reconciliation, materialization checkpoints, and query semantics. A response distinguishes complete for the profile, partial, redacted, expired, unavailable, conflicted, and unvalidated.

8.3 Policy consistency

Policy changes are control-plane operations. A policy deciding who can approve mappings, identities, retention, or checkpoints cannot be an eventually reconciled annotation while dependent actions are called globally authoritative. An authority-valid operation binds the policy revision, actor, credential context, scope, and decision time. Revocation semantics state whether an action valid when committed remains valid after revocation.

8.4 Read request and response

A semantic read request contains

$$q = \langle \mathit{query}, \mathit{semanticContext}, \mathit{profile}, \mathit{sourceCoverage}, \\ \mathit{freshnessBound}, \mathit{validationPolicy}, \mathit{timeout}, \mathit{fallbackPolicy} \rangle.$$

The response contains

$$r = \langle \mathit{rows}, \rho, \mathit{sourceFrontiers}, \\ \mathit{missingSources}, \\ \mathit{validationState}, \mathit{conflicts}, \mathit{provenanceProfile}, \\ \mathit{servedBy}, \mathit{observedAt} \rangle.$$

If a strict request cannot complete during a partition, it may block until the caller's timeout, reject with a typed consistency-unavailable result, redirect to an authority, or return a separately requested weaker profile. It cannot silently return stale or tentative state.

8.5 Semantic fractures

A read is semantically fractured when its pieces are individually valid but mutually incompatible. Examples include schema revision 8 with a mapping compiled for revision 7, rows under identity checkpoint 21 with provenance under checkpoint 22, a plan under one source capability profile applied under another, and a constraint status from one checkpoint combined with data from another.

RAMP addresses multi-item read atomicity in its model [7]. CatDB must still define which semantic components form the atomic item group.

9 Distributed limits

9.1 CAP

Gilbert and Lynch formalize an impossibility for particular definitions of atomic consistency, availability, and partition tolerance in an asynchronous network [15]. The result does not assign every system a permanent two-of-three product label.

For each operation, P11 must name the modeled object, consistency property, availability definition, partition model, timeout or rejection behavior, permitted stale or conflicting return, and whether the claim concerns safety or liveness.

During a partition:

- an `authoritative-linearizable` reference update may reject or block on the minority side;
- a `causal-evidence` append may remain locally available under its delivery assumptions;
- an `available-tentative` workspace operation may proceed without claiming one global current state;
- a bounded-staleness read becomes unavailable once its bound cannot be certified;
- a best-effort read may return an explicit stale or partial result; and
- destructive global administration must not silently run in separate partitions.

Category theory does not bypass this result. Typing can expose preservation obligations; it does not move messages across a partition.

9.2 PACELC

PACELC adds the normal-operation latency and consistency tradeoff [1]:

if partition: availability versus consistency; else: latency versus consistency.

This is an analysis frame, not an architecture theorem. Evaluation must report local, same-region, and cross-region latency distributions, quorum or authority paths, read locality, acknowledgement point, replication lag, timeouts, consistency achieved, and failure behavior.

Semantic compilation may keep category-level reasoning outside row processing. It does not remove the latency of persistence, replication, ordering, and validation.

9.3 CALM

CALM relates monotonicity and coordination freedom in its formal model [4]. It motivates treating evidence accumulation differently from retraction, uniqueness, global negation, and winner selection.

It does not show that every categorical mapping is monotone, every migration is coordination-free, every append-only log converges semantically, source completeness is monotone, or authority decisions commute. P11 must freeze the computation model before importing a CALM conclusion.

9.4 CRDTs

A CatDB object is a CRDT only after the state space, partial order or operation semantics, merge or delivery assumptions, idempotence and commutativity conditions, invariants, deletion semantics, and metadata retention are specified and proved [27].

Sets of immutable content-addressed artifacts and grow-only evidence facts are possible candidates. Even these remain open until redaction, tombstones, identity collisions, and garbage collection are part of the datatype. An operation log is not a CRDT merely because operations can be replayed.

9.5 Invariant confluence

For invariant J, transactions T, and merge $\sqcup$, invariant confluence asks whether J-valid states reachable from a common ancestor merge to a J-valid state [6]. The answer is relative to the exact invariant, operations, merge, and system model.

Candidate CatDB analyses include content-addressed artifact union, stable-ID uniqueness, exclusive active references, schema-equation preservation, identity contradiction exclusion, global uniqueness, provenance retention, and frontier monotonicity. A change to the operation or invariant set requires a new analysis.

9.6 Consensus assumptions

FLP prevents a deterministic consensus protocol from guaranteeing termination in the fully asynchronous crash-fault model [14]. A practical CatDB authority service must state its timing and crash-recovery assumptions, stable storage, leader or lease rules, quorum intersection, membership protocol, retry behavior, progress condition, and safety after loss of durable quorum state.

Raft, ZooKeeper, Chubby, and Spanner provide useful implementation baselines [10, 12, 17, 23]. None proves correctness of CatDB's proposed state-machine commands.

10 Bounded model results

The results in this section follow from the stated definitions. They do not establish a distributed runtime.

10.1 Profile dimensions do not collapse

Proposition 10.1 (Isolation does not determine replica visibility)**.** *There exist histories with the same local isolation component I and different replica visibility component V.*

Proof. Take two replicas whose local transactions are serializable. In history h_1, every committed update is synchronously replicated before completion, and strict reads observe a real-time-compatible order. In history h_2, each replica commits locally and exchanges updates asynchronously. Both histories have serializable local transactions. Their remote visibility differs: h_1 can satisfy a linearizable replicated-object specification, while h_2 admits a read at

one replica that does not yet observe a completed write at the other. Therefore I does not determine V. □

Proposition 10.2 (Visibility does not determine freshness)**.** *There exist materialized-read histories with the same replica visibility component V and different freshness component F.*

Proof. Let two replicas expose the same causally ordered sequence of materialization publications. In history h_1, publication μ includes every required source through the declared frontier. In history h_2, the identical publication order contains a μ' whose second source is missing. Replica visibility is causal in both histories. The completeness and freshness contracts differ. Hence V does not determine F. □

Corollary 10.3. *No scalar label derived only from local isolation or replica visibility determines the complete profile $\langle I, V, S, G, R, E, F, A, H\rangle$.*

Evidence status. PROVED IN MODEL. The statements concern the abstract profile, not a CatDB implementation..

10.2 Duplicate replay

Lemma 10.4 (Idempotent duplicate elimination)**.** *If o satisfies the idempotent replay definition, then any positive finite number of consecutive applications of o is observationally equivalent to one application.*

Proof. The two-application case is the definition. Assume n applications are equivalent to one. Applying o once more gives

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

Induction establishes the result for every $n \geq 1$. □

Remark 10.5. The premise covers the declared external-effect boundary. If an external charge is outside the equivalence or does not honor the operation ID, the lemma says nothing about duplicate charges.

10.3 Commuting operations

Theorem 10.6 (Permutation invariance for a pairwise commuting finite set)**.** *Let $O = \{o_1, \ldots, o_n\}$ be a finite set of deterministic operations. If every pair commutes on every admitted reachable state, then any two permutations of O yield the same final state.*

Proof. Any permutation can be transformed into any other by adjacent swaps. Each adjacent swap preserves state because the swapped operations commute on the reachable prefix state. Repeating the swaps preserves the final state. □

Remark 10.7. The theorem does not establish invariant preservation. Every intermediate and final state must still satisfy the named invariant, or the operation set needs coordination, reservation, rejection, or compensation.

Evidence status. PROVED IN MODEL. P10 classifies candidate commuting fragments in its accepted paper model, but no executable checker has established this theorem's all-reachable-state premise for a CatDB operation set..

10.4 Strict validation

Let validation states have a preorder in which

$$\mathsf{rejected}, \mathsf{authorityConflict}, \mathsf{tentative} \not\geq \mathsf{globallyValidated}.$$

Define a strict output as one whose dependencies all meet its required validation threshold.

Proposition 10.8 (Tentative-state nonleakage)**.** *Under the strict-output definition, an output depending on a tentative, rejected, or authority-conflicted input cannot be globally validated without a later validation operation that resolves that dependency.*

Proof. The dependency fails the strict output's threshold. The output is therefore not globally validated. Only a later operation that changes the dependency's status according to the validation state machine can make the predicate true. □

Evidence status. PROVED IN MODEL. The repository has no validation-state runtime..

10.5 Linearizable compare-and-swap

Proposition 10.9 (One successful predecessor)**.** *For a linearizable reference object, at most one compare-and-swap operation* $\mathsf{CAS}(x, C_0, C_i)$ *can succeed from predecessor* C_0 *before an intervening operation restores* C_0*.*

Proof. Linearizability places completed operations in a total order compatible with real time. The first successful compare-and-swap changes the reference away from C_0. Every later compare-and-swap that still expects C_0 fails until an intervening operation restores that value. □

Evidence status. PROVED IN MODEL for the abstract object. The repository has no linearizable CatDB reference store..

10.6 Checkpoint closure

Definition 10.10 (Causally closed checkpoint)**.** A checkpoint operation set X is causally closed when

$$o \in X \wedge p \to o \quad \Longrightarrow \quad p \in X.$$

Proposition 10.11 (Dependency completeness)**.** *A causally closed checkpoint cannot contain a dependent operation while omitting one of its causal parents.*

Proof. Immediate from the definition of causal closure. □

The proposition is deliberately small. It does not establish a consistent cut across dynamic membership, lossy channels, external sources, or retention. Chandy and Lamport's snapshot result applies only after the process and channel model is matched [11].

11 Replay, conflicts, and checkpoints

11.1 Replay identity

The operation ID must remain stable across client retry, be scoped to avoid collision, bind the payload digest and semantic vector, remain durable for the published replay window, be checked before the effect, and appear in generated provenance.

Crash point	Required recovery behavior
Before durable operation receipt	Retry may submit the same operation again.
After receipt, before effect	Replay applies the effect once.
After effect, before deduplication marker	Unsafe unless effect and marker are atomic or the effect is naturally idempotent.
After marker, before effect	Unsafe unless one transaction prevents the state or an intent record marks incomplete work.
After local effect, before remote effect	Retry or compensation names which side is authoritative.
After remote effect, before acknowledgement	The remote request ID or reconciliation prevents duplicate effect.
After materialization effect, before cursor	Recovery must not skip or duplicate the already applied input.
After cursor, before materialization effect	Forbidden by transactional coupling or repaired from an intent log.

The paper does not use unqualified "exactly once." It distinguishes at-most-once attempts, at-least-once delivery, deduplicated application, idempotent effect, transactional effect and cursor, and compensated external effect.

11.2 Conflict classes

Conflict records distinguish:

1. the same immutable ID with different content;
2. failed compare-and-swap predecessors;
3. concurrent active-reference choices;
4. schema versus schema and schema versus mapping incompatibility;
5. mapping disagreement or mapping versus data constraint failure;
6. identity contradiction and identity-driven materialization invalidation;
7. policy or authority conflict;
8. checkpoint or retention conflict;

9. incomparable source frontiers;
10. uncertain duplicate side effect;
11. validator disagreement; and
12. membership or quorum conflict.

Last-writer-wins is admissible only when the field permits arbitrary overwrite, the order source is trustworthy for that policy, losing values are retained when audit requires them, no invariant depends on the concurrent values, and the policy is visible. It is not the default for schema, mapping, identity, policy, checkpoint, or retention.

11.3 Validation state machine

The candidate validation sequence is

$$\mathsf{submitted} \to \mathsf{tentative} \to \{\mathsf{validated}, \mathsf{rejected}, \mathsf{authorityConflict}\} \to \mathsf{superseded}.$$

History is monotone even when current status changes. A later state does not erase the prior observation or evidence.

Rejection records the subject, rule and revision, observed evidence, authority, dependent artifacts, invalidation fanout, compensation, irreversible effects, and next admissible action. Asynchronous validation is unsafe when an effect cannot be hidden, reversed, compensated, or explicitly accepted as speculative.

11.4 Global checkpoints and retention

A checkpoint candidate is

$$\chi = \langle \mathit{replicaSet}, \mathit{operationFrontiers}, \mathit{semanticCommit}, \mathit{sourceFrontiers}, \\ \mathit{materializations}, \mathit{validationResults}, \mathit{conflicts}, \mathit{retentionPolicy} \rangle.$$

It is publishable only if the cut is consistent for the declared channels and all required dependencies are reachable.

An operation, tombstone, artifact, deduplication record, or provenance edge is collectible only when it is not needed by a recoverable replica, live branch, retained commit, promised retry, conflict resolution, materialization recovery, identity reversal, provenance query, audit policy, historical query, or pending validation. Absence of a recent read is not evidence of safety.

Garbage collection and redaction can be irreversible. A strict profile needs an authority and retention proof or must admit the loss of future reproducibility.

12 Implementation plan and present status

12.1 Prerequisite gate

Implementation should not begin until the program freezes:

1. P9 semantic commit, reference, and merge operations;
2. P10 operation envelope, operation algebra, fault model, and admitted convergence subset;
3. P6 materialization frontier and checkpoint semantics;
4. P7 identity decision and authority semantics;
5. the initial named consistency profiles; and
6. the history and fixture serialization.

The accepted P10 research manuscript closes the documentation part of this gate but does not supply the executable interfaces, history fixtures, or runtime behavior required to close it.

12.2 Proposed Rust boundary

A future `catdb-distribution` crate would contain:

- `OperationEnvelope` and `SemanticVector`;
- `CausalContext` and `SessionContext`;
- `ConsistencyProfile`;
- `DeliveryState` and `ApplyOutcome`;
- `ConflictRecord`;
- `ValidationStatus` and `AuthorityDecision`;
- `ReplicaFrontier` and `MaterializationFrontier`;
- `CheckpointCandidate` and `RetentionObligation`; and
- adapter traits for an operation log, reference/consensus store, transport, durable local transaction, and clock.

The crate must not redefine schema or mapping semantics, treat consensus as a categorical law, hide source limitations, infer identity authority from similarity, label generic replay exactly once, silently weaken a profile, or garbage collect beyond declared retention.

12.3 Implementation order

1. Implement pure profile types, validation, and serialization.
2. Build a single-process deterministic replica simulator.
3. Implement dependency and idempotency checks.
4. Add conflict records and tentative validation.

5. Cross-check finite histories with Haskell.
6. Prove the smallest fragment in Lean.
7. Implement SQLite operation/effect/cursor transactions.
8. Implement PostgreSQL equivalents and logical-decoding tests.
9. Replicate the immutable evidence subset.
10. Add a linearizable reference through a reviewed consensus service.
11. Build the fault harness.
12. Implement schema and mapping activation.
13. Add materialization read profiles.
14. Add checkpoint and retention protocols.
15. Run multi-region experiments only after safety gates pass.

12.4 Haskell reference model

The Haskell oracle should parse operation envelopes, generate finite histories, compute causal closure, classify concurrency, apply an admitted convergent subset, enumerate delivery orders for small histories, check session guarantees and semantic-vector compatibility, simulate tentative validation, model abstract reference CAS, and check checkpoint closure.

Its results would be OPEN computational evidence, not proofs.

12.5 Lean targets

The first Lean targets are:

1. acyclicity of dependencies for well-formed finite histories;
2. dependency-before-apply under causal delivery;
3. duplicate neutrality for the admitted idempotent fragment;
4. permutation equality for pairwise commuting operations;
5. semilattice laws for each declared join;
6. strict validation nonleakage;
7. semantic-vector cache-key agreement for a frozen key function;
8. checkpoint causal closure; and
9. one-success behavior for the abstract linearizable CAS object.

Lean acceptance would not prove network liveness, a Raft implementation, backend durability, clock bounds, source behavior, external effects, performance, usability, or the complete engine.

12.6 Current implementation verdict

All implementation and formalization targets in this section are OPEN or OBSTRUCTED. The repository does not compute a P11 result.

13 Falsification and evaluation plan

13.1 History record

Every distributed test records event ID, process, replica, session, operation ID, invocation or completion, operation kind, argument digest, semantic vector, requested profile, observed result, validation state, source frontiers, logical time, wall-time interval, network fault epoch, and crash epoch.

Wall time is diagnostic unless the guarantee explicitly depends on clocks.

13.2 Required history families

Local transaction anomalies. Run conflicting transactions at every supported backend isolation level. Check dirty writes, dirty reads, fractured reads, lost updates, read and write skew, predicates, aborted-read leakage, duplicate retry, cursor/effect tearing, and stale-revision validation. Elle supplies a useful checker for part of this space but not predicates [18].

Causal dependency delivery. Create schema s_1, mapping m_1 targeting s_1, and plan p_1 targeting m_1. Deliver them in reverse order. The system withholds or marks the dependent state until parents arrive. Duplicate all operations and verify one effect.

Concurrent monotone evidence. Append independent provenance assertions at two replicas, duplicate and reorder delivery, heal the partition, and compare normalized sets. Reuse one assertion ID with different content and require a hard conflict.

Noncommutative identity. Concurrently issue the incompatible decisions

$$d_A = \mathsf{confirmedSame}(x, y) \qquad \text{and} \qquad d_B = \mathsf{confirmedDifferent}(x, y).$$

The tentative profile retains both and reports conflict. The authoritative profile returns at most one current decision. Dependent materializations stay tentative or invalidate.

Branch compare-and-swap. Two clients read `main` at C_0, create different commits, and issue $\mathsf{CAS}(\mathit{main}, C_0, C_i)$. Exactly one succeeds under the strict profile. A minority partition rejects or blocks. The losing commit remains an immutable artifact. A linearizability checker can evaluate the abstract reference history [9].

Mixed schema versions. Run old and new readers and writers through every transition state. Exercise additive, destructive, type-changing, and constraint-strengthening revisions. Crash between transition steps. An incompatible writer must reject before corruption.

Mapping and plan invalidation. Compile under mapping 12, activate mapping 13, and deliver activation before invalidation. Immutable semantic keys prevent old plans from serving strict queries. An explicit old workspace can still read under mapping 12.

Materialization recovery. Crash before and after operation receipt, effect, cursor, deduplication marker, and publication. Replay with duplicates. Compare data and provenance with full recomputation at the same frontier.

Bounded staleness. Delay each source independently, produce comparable and incomparable frontiers, cross the declared bound, and remove one source. Response, blocking, or failure must match the requested profile.

Asynchronous validation. Publish a tentative mapping or identity decision and build dependent artifacts. Resolve validation as accepted, rejected, and authority conflict. Strict readers never see a tentative dependency as validated.

Membership transition. Reconfigure the replica set and partition old and new members at each step. Attempt conflicting authoritative updates. No disjoint authority may succeed.

Checkpoint and garbage collection. Create branches, tombstones, retries, materializations, and provenance. Reject inconsistent cuts. Retain history while any live obligation exists. After obligations close, collect only the permitted state. Replay outside the promised window must return an explicit expired result.

CAP and PACELC profiles. Partition minority and majority sides, run every profile, record complete/reject/block/stale/conflict behavior, heal, and measure convergence. Repeat without partitions at increasing regional latency.

Historical replay. Freeze a validated checkpoint, replay with retained semantic and source inputs, and compare normalized data and provenance. Delete a permitted source artifact and require a later response to report partial reconstructibility or expiry.

13.3 Falsification conditions

The central thesis fails if:

1. two materially different guarantees serialize to the same profile;
2. a coordination-free row reaches an invalid or divergent state;
3. duplicate replay creates an additional admitted effect;
4. a strict read exposes a causal child without its parent;
5. a current read uses a different semantic vector than reported;
6. tentative or rejected state appears as validated;

7. two authoritative CAS updates both succeed from one predecessor;
8. a checkpoint permits unsafe deletion;
9. an availability profile blocks without declaring that behavior;
10. a strong profile returns stale or conflicting state as one current value;
11. a convergence claim depends on delivery order for operations declared commuting; or
12. semantic metadata and enforcement cost erase the claimed practical benefit.

13.4 Metrics and baselines

Safety comes before performance. A configuration enters the performance comparison only after passing local anomaly, dependency-delivery, duplicate/reorder replay, conflict-preservation, validation, reference, mixed-version, fracture, checkpoint, and convergence checks.

Baselines include a local backend transaction, a strong-everywhere authority path, causal evidence, available tentative state, the operation-specific classified path, best-effort and bounded-stale reads, strict current reads, metadata-disabled and full-profile execution, full recomputation, incremental recovery, and single-region versus multi-region topology.

Correctness metrics include anomaly counts, divergent states, invariant violations, semantic fractures, duplicate effects, tentative-state leaks, staleness-bound violations, false completeness, unsafe collection attempts, checker coverage, and incorrectly resolved conflicts.

Performance metrics include:

- operation latency percentiles and throughput;
- quorum or authority round trips;
- causal and semantic metadata bytes;
- deduplication lookup cost;
- conflict frequency and resolution time;
- validation, replication, and convergence lag;
- source, semantic, and materialization frontier lag;
- plan-cache hit rate under versioned keys;
- checkpoint and recovery time;
- retained metadata and storage cost; and
- profile-specific availability during faults.

The evaluation also needs a human-factors check. Operators should be able to predict partition behavior, interpret stale, partial, tentative, and conflict statuses, diagnose a semantic fracture, choose a suitable profile, and reconstruct the context of an observed result.

13.5 Statistical contract

Performance claims preregister the workload and comparison, report warmup and run duration, use independent runs, report distributions and uncertainty, retain raw samples, separate failure-free and faulted intervals, count rejected and blocked requests, avoid averaging different profiles, and disclose hardware, network, backend, and configuration.
Evidence status. OBSTRUCTED. No fault harness, distributed runtime, or P11 benchmark artifact exists..

14 Claim register

ID	Claim	Evidence class	Current status
C01	The nine-dimensional profile distinguishes independent consistency obligations.	Model propositions and future API tests	OPEN
C02	A supported sink can couple effect, replay marker, and cursor atomically or recover equivalently.	Crash testing	OPEN
C03	Replication preserves declared semantic dependencies.	Causal history tests	OBSTRUCTED
C04	The admitted operation subset is replay-idempotent within its declared effect boundary.	Model proof plus crash and effect tests	OBSTRUCTED
C05	A declared commuting subset converges after duplicate, reordered, and partitioned delivery under stated assumptions.	Proof plus fault histories	OBSTRUCTED
C06	Operations outside that subset serialize, reject, branch, or expose typed conflicts.	Runtime and negative histories	OBSTRUCTED
C07	An authoritative reference supports linearizable CAS and safe membership.	Consensus adapter and checker	OBSTRUCTED
C08	Schema activation prevents incompatible mixed-version corruption.	Protocol proof and fault histories	OBSTRUCTED
C09	Strict reads exclude semantically fractured revision bundles.	Runtime checks and generated histories	OPEN
C10	Materialized reads report and enforce frontier, completeness, semantic, and validation state.	Runtime and delay tests	OBSTRUCTED
C11	Strict reads never expose tentative dependencies as validated.	Model result and dependency tests	OPEN
C12	Conflicting identity evidence remains while one authority-valid current decision is exposed under the strict profile.	Identity runtime and fault tests	OBSTRUCTED
C13	Validated checkpoints are consistent cuts and retention preserves every declared obligation.	Snapshot and retention model and tests	OBSTRUCTED
C14	Workspace sessions provide requested client-centric guarantees without claiming global consistency.	Session histories	OBSTRUCTED

ID	Claim	Evidence class	Current status
C15	Every distributed profile declares partition and normal-latency behavior without silent downgrade.	API audit and CAP/PACELC experiment	OPEN
C16	Operation-specific coordination improves latency or availability over a strong-everywhere baseline without invariant loss.	Controlled benchmark	OBSTRUCTED

No row currently has a CatDB implementation result. The model propositions in Section 10 support only their exact abstract statements.

15 Limitations and exact nonclaims

This manuscript is useful only if its limits remain visible.

First, the operation taxonomy is not an executable interface. Paper 10 supplies an internally accepted paper algebra, but P9 workspace operations, P6 materialization state, P7 identity authority, and P13 provenance retention are not accepted distributed interfaces. A changed constructor, invariant, merge, or authority scope can change a row's classification.

Second, a matrix is not enforcement. The proposed profiles need runtime types, a policy compiler, durable metadata, and response context. A document that says "causal" does not cause a transport to delay children until parents arrive.

Third, the paper-level results are small. They prove facts about definitions, pairwise commuting deterministic operations, validation thresholds, abstract linearizable compare-and-swap, and causal closure. They do not prove a network protocol, a storage engine, a source connector, or the complete operation matrix.

Fourth, weak and strong modes can be difficult to use. A small set of named profiles may still expose too many choices, while an aggressively simplified API may hide material tradeoffs. The proposed human-factors evaluation is necessary.

Fifth, source systems limit end-to-end guarantees. APIs may not accept stable request IDs, files may lack ordered change logs, PostgreSQL logical replication has version-specific restrictions, and external state may disappear. A local transaction cannot repair those boundaries.

Sixth, retention conflicts with privacy and cost. Durable provenance, tombstones, deduplication keys, and branch history improve explanation and replay but may retain data that policy requires the system to delete. Redacted history may cease to support reproduction.

The paper makes these exact nonclaims:

- CatDB does not eliminate CAP or PACELC tradeoffs.
- Category theory does not provide consensus, failure detection, or network availability.
- Functoriality does not imply commutativity or convergence.
- Not every CatDB operation is a CRDT.
- A causal log does not enforce global invariants.

- Local ACID does not imply distributed ACID.
- Consensus does not validate semantic meaning.
- Asynchronous validation is not strong consistency when tentative state is exposed as final.
- Session guarantees are not global consistency.
- One timestamp does not prove multi-source freshness.
- A watermark does not prove completeness outside its assumptions.
- Idempotent replay is not unqualified exactly once.
- Schema and mapping activation can require coordination.
- Similarity does not establish semantic identity.
- Explicit conflict availability is not one consistent value.
- Garbage collection can sacrifice reproducibility.
- Testing cannot prove the absence of every distributed bug.
- Machine-checked small-model invariants would not verify the production engine.
- Competitive latency, throughput, and availability remain experimental questions.

16 Open questions

16.1 Blocked by upstream semantics

1. Which exact P10 operations and state transitions will the runtime replicate?
2. What network, storage, crash, recovery, and membership model will P10 adopt?
3. Which P9 commit, reference, and merge operations survive review?
4. What is an authority scope: global, workspace, tenant, source, object, or policy domain?
5. Which exact subset is commuting and idempotent?
6. What equality defines replica convergence?
7. Which P6 source-frontier and checkpoint model is accepted?
8. Which P7 identity decision and authority model is accepted?
9. Which schema transitions and mapping adapters are admitted?
10. How do checkpoints interact with changing replica membership?
11. Which external effects are inside the CatDB atomicity boundary?
12. What retention obligations are mandatory?

16.2 Open design and evaluation questions

1. Which small set of profile names remains usable without hiding a material dimension?
2. Should semantic vectors travel inline or through immutable context references?
3. How much causal metadata do typical semantic dependencies require?
4. Can dependency graphs be compressed without creating false applicability?
5. Which operations are invariant-confluent under the frozen invariants?
6. Can additive schema activation avoid global consensus through a staged protocol?
7. Can mapping activation be partitioned safely by scope?
8. Which reservation or escrow structures apply to keys and capacities?
9. Which validation tasks may be asynchronous without irreversible leakage?
10. How should strict reads behave when source frontiers are incomparable?
11. Which staleness measure works for file and API sources without ordered logs?
12. How long must operation IDs and deduplication records remain?
13. How should privacy redaction interact with provenance reconstructibility?
14. Which consensus or coordination service is suitable for the first prototype?
15. Can the Haskell oracle enumerate histories at useful semantic complexity?
16. Which Lean targets give the best assurance for their cost?
17. What performance advantage remains after profile, validation, and provenance overhead?
18. Can users predict behavior from profile names and response metadata?

17 Conclusion

CatDB needs a consistency contract that follows the operation, not a label attached to the product. An immutable proposal, an evidence assertion, an active mapping, a canonical identity decision, a materialization cursor, and a garbage-collection authorization do different work. Treating them alike either coordinates too much or hides conflicts that matter.

The provisional model separates nine consistency dimensions and classifies 30 operations across local transactions, causal broadcast, serialized authority, idempotent replay, conflict handling, and asynchronous validation. It keeps schema, mapping, materialization, identity, provenance, policy, read, and retention semantics visible. It also keeps the limits visible. CAP and PACELC still apply. CALM, CRDT, and invariant-confluence results require a frozen model. Consensus orders decisions but does not decide whether their meaning is correct.

The repository does not implement this contract. Paper 10 now provides an accepted research vocabulary, but its operation algebra and dependent semantics have not passed

the executable gates needed for P11. The next useful result is not a stronger claim. It is a deterministic history model, a small accepted operation fragment, cross-checked Rust and Haskell behavior, selected Lean properties, and fault histories that try to break every matrix row.

Until those artifacts exist, the paper's systems thesis remains OBSTRUCTED.

A Repository evidence boundary

Evidence does not transfer between layers. A Raft safety argument does not make an unimplemented branch update linearizable. A CRDT theorem for one datatype does not make a destructive schema change converge. A SQLite transaction does not make a PostgreSQL and API write-through atomic. A causal log does not preserve every application invariant. A freshness timestamp does not prove completeness.

A.1 Repository audit

The current repository has no:

- `catdb-distribution` crate or replicated operation executor;
- causal clock, causal broadcast, anti-entropy, or transport implementation;
- consensus adapter, membership protocol, or linearizable reference store;
- consistency-profile, session-context, or distributed-conflict runtime type;
- distributed schema transition or mapping activation protocol;
- materialization frontier service or distributed checkpoint;
- replay window and durable deduplication contract;
- network fault harness or executable distributed history suite;
- Haskell distributed oracle or Lean distributed consistency model; or
- P11 benchmark result.

The P10 dossier and internally accepted manuscript now propose an operation classification. They report the same missing distribution surface and keep runtime claims obstructed. P11 therefore imports the paper model but no implemented guarantee from P10.

A.2 Dependencies

P11 depends directly on P10 for the replicated operation set, causal metadata, commutative subset, conflict boundary, membership model, and convergence criterion. Other dependencies are:

- P9 for semantic commits, workspace overlays, branch publication, merge, and rollback;

- P3 for immutable schema revisions and migration admissibility;
- P4 for mapping revisions, composition, coverage, and loss;
- P6 for materialization definitions, source frontiers, invalidation, cursor/effect checkpoints, and recovery;
- P7 for identity evidence, authority, contradiction, supersession, and reversal; and
- P13 for provenance completeness, trust, retention, and transformation lineage.

Several dependencies remain provisional or obstructed. P11 cannot resolve them by attaching a consistency label to an undefined operation.

B Prior results and the novelty boundary

B.1 Order and visibility

Lamport's happened-before relation supplies the basic partial order for causal reasoning [19]. Sequential consistency requires a legal common order consistent with each process's program order, but it does not impose real-time order [20]. Linearizability adds a real-time constraint and has a locality property for modeled concurrent objects [16]. These distinctions matter for CatDB. An immutable artifact set may not need one real-time global order. A branch compare-and-swap that claims one current head does.

Session guarantees improve client observations under weak replication. Read-your-writes, monotonic reads, monotonic writes, and writes-follow-reads do not imply global linearizability or bounded staleness [28]. COPS demonstrates a concrete causal wide-area store with explicit dependency tracking [22]. These systems establish prior vocabulary and implementation precedents. They do not classify CatDB's semantic dependencies.

Transaction isolation is another axis. Adya, Liskov, and O'Neil define implementation-independent isolation through dependency relations and anomalies [2]. RAMP transactions provide scalable atomic visibility in their model without supplying general serializability [7]. A materialized query may require atomic visibility of a semantic-version bundle even when its source transaction isolation is weaker. Conversely, local serializability says nothing about remote materialization freshness.

B.2 Weak replication and conflict

Bayou distinguishes tentative and committed updates and gives applications dependency checks and merge procedures [29]. Dynamo combines object versioning, quorum-like techniques, anti-entropy, and application-assisted conflict resolution [13]. Optimistic replication has a long design history with explicit reconciliation boundaries [26]. P11's tentative and conflict states are therefore not novel concepts.

The proposed distinction is semantic scope. A concurrent annotation, an active schema, a canonical identity decision, and a retention authorization have different admissible merge policies. Arrival order is not a neutral policy. Last-writer-wins is valid only where overwrite semantics, timestamp authority, and loss of the losing value are declared.

B.3 Availability and bounded inconsistency

The HAT analysis classifies transaction and replica guarantees that can and cannot remain highly available under its model [5]. TACT models numerical error, order error, and staleness as application-specific consistency dimensions [31]. These works directly preclude novelty claims for a consistency spectrum or a multidimensional consistency model.

CatDB's candidate delta would be a versioned profile tied to typed semantic objects and enforced end to end. The profile must include semantic revisions, validation and authority state, source frontiers, and historical retention. That combination remains an ENGINEERING HYPOTHESIS.

B.4 Coordination avoidance and convergent data

CRDT theory gives convergence conditions for declared state-based and operation-based datatypes [27]. CALM relates monotonicity and coordination in a relational-transducer model [4]. RedBlue consistency classifies operations so some use weaker ordering while others use stronger ordering [21]. Invariant confluence supplies a model-relative condition for safe coordination avoidance [6]. Indigo uses invariant awareness and reservations to recover more availability [8].

These are the closest precedents to the P11 matrix. The paper cannot claim novelty for "strong where needed, weak where possible." Its only plausible systems contribution is a precise semantic operation algebra, classifications justified by proof or adversarial histories, and faithful runtime enforcement. None exists yet.

B.5 Consensus and control planes

FLP shows a nontermination possibility for deterministic consensus in a fully asynchronous model with one crash fault [14]. Practical protocols add timing, failure-detector, randomness, or operational assumptions. Raft provides a replicated-log design with explicit leader and membership rules [23]. ZooKeeper gives linearizable writes and FIFO client order while allowing local reads with a weaker contract [17]. Chubby is a production coarse-grained lock and reliable low-volume metadata service [10]. Spanner provides external consistency through a substantially stronger replicated database and clock architecture [12].

Consensus establishes an order or selected value under a protocol. It does not determine whether a mapping is semantically correct, an identity decision is justified, or a materialization is fresh. P11 must specify the state machine and validator in addition to the ordering service.

B.6 Schema, mapping, and streaming precedents

F1's online schema-change protocol permits asynchronous mixed versions under a formal transition discipline and bounded version skew. The same paper shows that common schema changes can corrupt data without that discipline [25]. Mapping adaptation research shows that schema changes can invalidate mappings and that valid rewritings require semantic analysis [30]. MillWheel provides a concrete streaming framework with logical time, persistent state,

and fault-tolerance mechanisms [3]. Chandy and Lamport show how to obtain a consistent distributed cut under their process and channel assumptions [11].

These results motivate schema-transition, mapping-invalidation, materialization-frontier, and checkpoint contracts. They do not establish a generic CatDB implementation.

B.7 Executable history checking

Elle infers a broad Adya-style anomaly set from black-box observations for specified datatypes while excluding predicates from its published coverage [18]. Line-Up automates linearizability checking for supplied concurrent-object models [9]. P11 needs both kinds of checker plus CatDB-specific oracles for semantic-version compatibility, frontier completeness, validation propagation, provenance, and checkpoint closure. Passing sampled histories would remain experimental evidence rather than a proof of all executions.

B.8 Novelty statement

The paper does not claim novelty for:

- linearizability, sequential or causal consistency, or session guarantees;
- weak transactions, atomic visibility, tentative updates, or application conflict resolution;
- CRDTs, CALM, RedBlue consistency, or invariant confluence;
- consensus-backed metadata or online distributed schema change;
- mapping adaptation, bounded staleness, replay, deduplication, checkpoints, or consistent cuts; or
- CAP or PACELC analysis.

The candidate contribution is a typed and versioned consistency-profile system spanning schema, mapping, migration, identity, provenance, materialization, workspace, policy, and checkpoint objects. That candidate remains OPEN.

C Running history

Let replicas A and B start at semantic commit C_0. The commit references:

- schema revision `customer@7`;
- mapping revision `sap-customer@12`;
- identity policy `customer-match@4`;
- reconciliation set `customer-decisions@21`;
- provenance profile `row-field-origin@3`;
- materialization definition `customer-revenue@8`;

- source frontier `sap:lsn-145`;
- authority policy `finance-admin@5`; and
- branch reference `main` pointing to C_0.

During a partition, A appends provenance assertion p_1, creates $d_A = \mathsf{confirmedSame}(x, y)$, and computes materialization delta m_A through `sap:lsn-149` under mapping `@12`. Replica B creates $d_B = \mathsf{confirmedDifferent}(x, y)$, creates schema `customer@8` with a uniqueness constraint, and creates mapping `sap-customer@13` that changes gross revenue to net revenue. Both try to move `main` from C_0 to incompatible commits.

After healing, p_1, d_A, m_A, d_B reach both replicas with duplicates and different arrival orders. A valid design may union p_1 into an immutable evidence set. It must not infer that d_A, d_B, the branch reference, constraint activation, and materialization publication are also safe under unordered union.

The history creates five separate questions:

1. Which values can be stored without becoming authoritative?
2. Which causal parents must arrive before their dependents?
3. Which concurrent values commute or merge deterministically?
4. Which decision requires one serialized authority?
5. Which read can truthfully report freshness and validation?

The operation matrix in Section 3 answers these questions provisionally. It does not prescribe one protocol for the complete state.

D System and fault model

D.1 Processes, replicas, and operations

Let R be a finite set of replicas and P a finite set of client processes. A history contains invocations, completions, deliveries, applications, validations, publications, crashes, recoveries, and network fault events. Each semantic operation has the envelope

$$o = \langle \mathit{opId}, \mathit{kind}, \mathit{actor}, \mathit{authority}, \mathit{workspace}, \\ \mathit{semanticVector}, \mathit{parents}, \mathit{payloadDigest}, \mathit{precondition}, \mathit{issuedAt} \rangle.$$

The envelope is a proposal. P10 freezes paper-level constructors, preconditions, and authority scopes for research, but P11 cannot treat them as an implementation interface until serialization, validators, histories, and runtime behavior are accepted.

Definition D.1 (Causal dependency)**.** For operations o_1, o_2, write $o_1 \to o_2$ when:

1. both occur in one session and o_1 precedes o_2;
2. o_2 reads, references, validates, supersedes, derives from, or is authorized by o_1; or

3. the transitive closure of these rules relates them.

Definition D.2 (Strict applicability)**.** An operation is strictly applicable at replica r only if its causal parents are present, its precondition holds, its semantic vector is compatible with the local context, its authority is valid, and every synchronous validation required by its profile has succeeded.

D.2 Fault assumptions

The provisional test model includes:

- process crash and recovery with stable local storage;
- message delay, loss followed by later retransmission, duplication, and reordering;
- symmetric and asymmetric network partitions;
- delayed or unavailable source systems;
- ambiguous client timeout and retry;
- mixed semantic versions;
- leader change and replica membership transition; and
- late validation or conflicting authority decisions.

Byzantine faults are outside the first prototype. Liveness assumptions for any consensus service must be stated by that service. The paper assumes no perfect global clock. A profile that depends on bounded clock uncertainty must name and measure that assumption.

D.3 Consistency profile

Definition D.3 (Consistency profile)**.** A consistency profile is

$$\mathcal{K} = \langle I, V, S, G, R, E, F, A, H \rangle,$$

where:

- I is local transaction isolation and durability;
- V is replica visibility and order;
- S is the set of session guarantees;
- G is convergence and its preconditions;
- R is semantic-revision compatibility;
- E is the replay and effect contract;
- F is materialization freshness and completeness;
- A is validation and authority state; and

- H is historical retention and checkpoint state.

Candidate I values include no transaction, read committed, snapshot, serializable, and strict serializable. Candidate V values include local only, eventual, causal, sequential, linearizable, and explicit conflict set. The session component can request read-your-writes, monotonic reads, monotonic writes, and writes-follow-reads.

The convergence component must name an initial-state set, admitted operation set, delivery assumptions, deterministic apply or merge rule, invariant set, quiescence or fairness assumption, and state equality or observational equivalence. The replay component must name operation identity, deduplication scope and retention, atomicity boundary, crash points, and external side effects.

D.4 Semantic revision vector

Definition D.4 (Semantic revision vector)**.** For one request or effect,

$$\rho = \langle s, m, q, i, p, a, d, c \rangle$$

records schema s, mapping m, query q, identity/reconciliation state i, provenance profile p, authority policy a, materialization definition d, and source capability profile c.

Compatibility need not mean componentwise equality. A named and validated compatibility relation may admit adjacent schema versions or mapping adapters. If no relation is established, strict execution rejects, isolates, or exposes an explicit conflict.

D.5 Freshness vector

For sources $S_1, \ldots, S_n$, a materialization frontier is a vector

$$\mathbf{f} = \langle f_1, \ldots, f_n \rangle$$

with source-specific partial orders. Define

$$\mathbf{f} \preceq \mathbf{g} \quad \iff \quad \forall i.\ f_i \preceq_i g_i.$$

Two frontiers may be incomparable. A later publication timestamp does not make one frontier semantically newer than another.

D.6 Status and authority

Candidate validation states are unvalidated, locally validated, globally validated, rejected, superseded, and authority conflict. Candidate authority states include no authority, source authority, domain authority, a single-writer lease, quorum authority, and required human approval. The profile states whether unvalidated state may be returned and to which readers.

Historical state is independent. A result may be live, checkpointed, reconstructible, partially reconstructible, redacted, expired, or unavailable. A digest without retained evidence does not make a result reproducible.

E Mechanism definitions

E.1 Local transaction

A local transaction makes all effects inside one declared local atomicity domain visible together or not at all. It does not imply a distributed transaction, causal delivery, global serializability, or an atomic external API effect.

E.2 Causal broadcast

Causal broadcast ensures that if $o_1 \to o_2$, a replica delivering both delivers o_1 before o_2, or withholds o_2 until the dependency is available. Concurrent operations need not have the same observation order at every replica.

E.3 Consensus or serialized authority

The consensus column includes a replicated state machine, a linearizable compare-and-swap authority, or an equivalent single serialized authority for the decision's scope. A single administrator or primary can supply such an order with a different availability and failure model. The mechanism is required only when the contract needs one global choice or order.

E.4 Idempotent replay

Definition E.1 (Idempotent replay)**.** An operation o is idempotent under observation equivalence $\simeq$ when

$$\mathsf{apply}(\mathsf{apply}(s, o), o) \simeq \mathsf{apply}(s, o)$$

for every admitted state s. The equivalence includes durable external effects inside the declared boundary.

Stable operation identity, durable deduplication state, payload binding, and a retention window belong to this definition. Receiving one operation ID with different content is a conflict, not a retry.

E.5 Conflict resolution

Conflict resolution may mean deterministic merge, multi-value exposure, rejection, retry against a new predecessor, supersession with history, compensation, source-authority referral, or human approval. Every resolution is itself a versioned operation with provenance.

E.6 Asynchronous validation

Asynchronous validation admits an operation into a distinct tentative state before every check completes. It is safe only if the status is visible, strict consumers cannot mistake it for validated state, dependencies inherit tentativeness, rejection or compensation is defined, validation inputs are retained, and irreversible authority-sensitive effects do not occur early.

E.7 Matrix cell vocabulary

Cell	Meaning
REQ	Required for the row's stated strict contract.
COND	Required only under the named profile, invariant, or topology.
NO	Not required for the minimum stated contract, though an implementation may use it.
ALT	Alternative to blocking or coordination only if an explicit tentative or conflict state is exposed.
INSUF	The mechanism alone cannot establish the row's contract.
DEFER	Depends on a missing upstream semantic definition.

References

[1] Daniel J. Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *Computer*, 45(2):37–42, 2012. doi: 10.1109/MC.2012.33. URL https://doi.org/10.1109/MC.2012.33.

[2] Atul Adya, Barbara Liskov, and Patrick E. O'Neil. Generalized isolation level definitions. In *Proceedings of the 16th International Conference on Data Engineering*, pages 67–78, 2000. doi: 10.1109/ICDE.2000.839388. URL https://doi.org/10.1109/ICDE.2000.839388.

[3] Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. Millwheel: Fault-tolerant stream processing at internet scale. *Proceedings of the VLDB Endowment*, 6(11): 1033–1044, 2013. doi: 10.14778/2536222.2536229. URL https://doi.org/10.14778/2536222.2536229.

[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. doi: 10.1145/2450142.2450151. URL https://doi.org/10.1145/2450142.2450151.

[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. doi: 10.14778/2732232.2732237. URL https://doi.org/10.14778/2732232.2732237.

[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. doi: 10.14778/2735508.2735509. URL https://doi.org/10.14778/2735508.2735509.

[7] Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Scalable atomic visibility with RAMP transactions. In *Proceedings of the 2014 ACM SIGMOD*

International Conference on Management of Data, pages 27–38, 2014. doi: 10.1145/2588555.2588562. URL https://doi.org/10.1145/2588555.2588562.

[8] Valter Balegas, Diogo Serra, Sergio Duarte, Carla Ferreira, Marc Shapiro, Rodrigo Rodrigues, and Nuno Preguiça. Putting consistency back into eventual consistency. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 6:1–6:16, 2015. doi: 10.1145/2741948.2741972. URL https://doi.org/10.1145/2741948.2741972.

[9] Sebastian Burckhardt, Chris Dern, Madanlal Musuvathi, and Roy Tan. Line-up: A complete and automatic linearizability checker. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 330–340, 2010. doi: 10.1145/1806596.1806634. URL https://doi.org/10.1145/1806596.1806634.

[10] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *7th USENIX Symposium on Operating Systems Design and Implementation*, pages 335–350, 2006. URL https://www.usenix.org/conference/osdi-06/presentation/chubby-lock-service-loosely-coupled-distributed-systems.

[11] K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985. doi: 10.1145/214451.214456. URL https://doi.org/10.1145/214451.214456.

[12] 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. URL https://www.usenix.org/conference/osdi12/technical-sessions/presentation/corbett.

[13] 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. doi: 10.1145/1294261.1294281. URL https://doi.org/10.1145/1294261.1294281.

[14] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985. doi: 10.1145/3149.214121. URL https://doi.org/10.1145/3149.214121.

[15] 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. doi: 10.1145/564585.564601. URL https://doi.org/10.1145/564585.564601.

[16] 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. doi: 10.1145/78969.78972. URL https://doi.org/10.1145/78969.78972.

[17] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. ZooKeeper: Wait-free coordination for internet-scale systems. In *2010 USENIX Annual Technical Conference*, pages 145–158, 2010. URL https://www.usenix.org/conference/usenix-atc-10/zookeeper-wait-free-coordination-internet-scale-systems.

[18] Kyle Kingsbury and Peter Alvaro. Elle: Inferring isolation anomalies from experimental observations. *Proceedings of the VLDB Endowment*, 14(3):268–280, 2021. doi: 10.14778/3430915.3430918. URL https://doi.org/10.14778/3430915.3430918.

[19] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978. doi: 10.1145/359545.359563. URL https://doi.org/10.1145/359545.359563.

[20] Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, C-28(9):690–691, 1979. doi: 10.1109/TC.1979.1675439. URL https://doi.org/10.1109/TC.1979.1675439.

[21] 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. URL https://www.usenix.org/conference/osdi12/technical-sessions/presentation/li.

[22] 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. doi: 10.1145/2043556.2043593. URL https://doi.org/10.1145/2043556.2043593.

[23] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference*, pages 305–319, 2014. URL https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro.

[24] PostgreSQL Global Development Group. Logical decoding concepts and logical replication restrictions. https://www.postgresql.org/docs/current/logicaldecoding-explanation.html, 2026. Official documentation; version-sensitive. Restrictions: https://www.postgresql.org/docs/current/logical-replication-restrictions.html; accessed 23 July 2026.

[25] Ian Rae, Eric Rollins, Jeff Shute, Sukhdeep Sodhi, and Radek Vingralek. Online, asynchronous schema change in F1. *Proceedings of the VLDB Endowment*, 6(11):1045–1056, 2013. doi: 10.14778/2536222.2536230. URL https://doi.org/10.14778/2536222.2536230.

[26] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Computing Surveys*, 37(1):42–81, 2005. doi: 10.1145/1057977.1057980. URL https://doi.org/10.1145/1057977.1057980.

[27] 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. doi: 10.1007/978-3-642-24550-3_29. URL https://doi.org/10.1007/978-3-642-24550-3_29.

[28] Douglas B. Terry, Alan J. Demers, Karin Petersen, Mike J. Spreitzer, Marvin M. Theimer, and Brent B. Welch. Session guarantees for weakly consistent replicated data. In *Proceedings of the Third International Conference on Parallel and Distributed Information Systems*, pages 140–149, 1994. doi: 10.1109/PDIS.1994.331722. URL https://doi.org/10.1109/PDIS.1994.331722.

[29] 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 172–182, 1995. doi: 10.1145/224056.224070. URL https://doi.org/10.1145/224056.224070.

[30] 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. doi: 10.1016/B978-012722442-8/50058-6. URL https://doi.org/10.1016/B978-012722442-8/50058-6.

[31] Haifeng Yu and Amin Vahdat. Design and evaluation of a continuous consistency model for replicated services. In *Fourth Symposium on Operating Systems Design and Implementation*, pages 305–318, 2000. URL https://www.usenix.org/conference/osdi-2000/design-and-evaluation-continuous-consistency-model-replicated-services.