Zaxon Working Group
Request for Discussion and Implementation Record

ZDS 0011
Engineering Discussion

Global Slots and Certified Log Trimming

STATE
committed

INTENDED STATUS
Implemented for paxos-zig 0.6.0 and zaxonlite 0.6.0

CREATED
2026-08-26

AUTHORS
paxos-zig project

LAST UPDATED
2026-08-27

DISCUSSION

Remove the 2,044-commit rollover by separating Paxos progress, log retention, and SQLite state transfer

LABELS
consensus, paxos, zaxonlite, storage, verification

Status of This Memo

This document is an internal Zaxon discussion record authored in Typst and tracked in git. It intentionally follows RFC-style structure so the design scope, rationale, trade-offs, and operational constraints remain explicit. Documents that still use the placeholder number XXXXX are provisional drafts. Numbered ZDS documents are part of the project record.

Contents

1. Abstract

The current Paxos core allocates state for a compile-time maximum number of slots. Zaxonlite sets that limit to 2,048 entries and reserves the final four positions. Near slot 2,044, the host checkpoints SQLite, copies the complete database, synchronizes the copy, hashes every byte, seals the Paxos configuration, and restarts slot numbering in a new configuration.

That mechanism is safe but combines three independent concerns: bounded RAM, log reclamation, and state transfer. Its routine cost is proportional to database size while its frequency is proportional to writes. A multi-terabyte SQLite database therefore cannot use the current rollover as an ordinary maintenance operation.

This record proposes a deliberately conservative first release:

- one monotonic 64-bit global slot space;
- a fixed-size, slot-tagged in-memory consensus window;
- a streaming segmented durable journal indexed by absolute slot;
- distinct executed, durable-state, chosen-trim, and local-delete frontiers;
- an explicit trimmed-acceptor Phase-1 response and leader selection rule;
- conservative cleanup through the minimum durable-state frontier of every current data replica; and
- anchor-pinned, byte-exact state transfer only for bootstrap, repair, or voter replacement.

Membership configurations remain explicit, but no longer exist to recycle slots. Normal trimming never copies or hashes the SQLite database. The design keeps Paxos safety, bounds memory independently of database lifetime, and makes common-path reclamation depend on the amount of obsolete log data rather than the size of materialized application state.

Quorum-based trimming is not part of the first release. A failed data voter freezes conservative cleanup until it recovers or is replaced through ZDS 0008, matching the 2025 HoliPaxos design. A

later extension may choose a lower holder threshold, but only after agreement, state durability, and recovery availability are specified as three separate properties.

2. Decision Summary

Proposed direction. Replace epoch-capacity rollover with global slots, bounded windows, streaming journal segments, and conservative log trimming. Keep stop signs only for real membership changes.

Safety authority. Paxos chooses commands and trim records. A trimmed acceptor answers Phase 1 with a durable chosen-prefix anchor, and a leader never proposes into an anchored prefix. Accepted-only slots are never evicted or deleted.

Recovery authority. A synchronized local SQLite image plus its alternating durable state anchor is a checkpoint. Exact captured page images, including final page count, are streamed from segmented journals. A node behind the cluster trim installs an anchor-pinned raw image and then replays the suffix.

Performance intent. Ordinary writes retain the existing journal barrier. Durable-state anchoring is periodic and may add a small checkpoint/barrier; it never performs a full-file copy or full-file hash. Physical image copy is paid only for bootstrap, repair, or replacement.

3. Introduction

Paxos needs an unbounded logical sequence, not unbounded volatile memory. A finite implementation may retain a bounded moving window so long as old decisions remain recoverable and a reused physical cell cannot be confused with its former logical slot. Database systems make the same distinction: ARIES separates the log sequence number, the durable log, and the page state; Raft snapshots and Multi-Paxos trimming separate replicated progress from log retention; production systems such as Spanner, DynamoDB, Aurora, and Delos compose consensus with independently managed storage layers.

The present implementation instead treats a compile-time array bound as the end of a logical log. Zaxonlite repairs that exhaustion by creating a complete snapshot generation and a new Paxos configuration. This works for a small database, but the important failure is the periodic stop-the-world duty cycle, not merely the amortized mean.

Let λ be the write rate, E the fixed epoch length, and

$$T_{\mathrm{roll}}(D) = C_0 + \frac{D}{B_c} + \frac{D}{B_h}$$

where C_0 contains synchronization and transition latency. An epoch serves writes for approximately $\frac{E}{\lambda}$ seconds, so the stalled wall-clock fraction is

$$\varphi(D, \lambda) = T_{\mathrm{roll}} \frac{D}{\frac{E}{\lambda} + T_{\mathrm{roll}}(D)}$$

.

Ignoring C_0 and taking $B_c = B_h = B$, keeping $\varphi \le \varepsilon$ requires

$$D \le EB \frac{\varepsilon}{2\lambda(1-\varepsilon)}$$

.

For $E = 2044$, $B = 1$ GB/s, and $\varepsilon = 0.01$, the illustrative database-size ceilings are approximately:

Write rate	1% duty-cycle ceiling
100/s	103 MB
1,000/s	10.3 MB
10,000/s	1.03 MB
50,000/s	206 KB

These are not device-independent benchmark predictions; they expose the shape. Faster writes make a commit-count trigger occur more often.

The snapshot also causes at least $3D$ device traffic per epoch: read D, write D, then read D again for SHA-256. If a workload changes p bytes per commit, a simplified lower-bound traffic multiplier is

$$\mathrm{WA}_{\mathrm{snapshot}} \geq 1 + 3\frac{D}{Ep}$$

.

At $p = 4$ KB, this is about $37\times$ at $D = 100$ MB, $359\times$ at $D = 1$ GB, and $3.6 \cdot 10^5\times$ at $D = 1$ TB. The exact device amplification also includes SQLite WAL, payload, journal, metadata, and filesystem behavior; the formula isolates only the avoidable snapshot term.

The desired architecture has an unbounded logical history, bounded working memory, bounded retained recovery history, and rare state transfer. Those are compatible properties when each boundary has its own proof.

4. Terminology and Scope

- **global slot** s: a monotonically increasing `u64` Paxos instance number; it never resets during the lifetime of a database
- **configuration**: a versioned voter set; it changes only through an explicit membership decision
- **consensus window** W: the maximum number of unresolved or locally retained Paxos slots represented in fixed memory
- **segment**: an immutable, contiguous range of durable journal records
- **chosen frontier** C: the greatest slot for which every slot through C is locally known chosen
- **executed frontier** E_i: the greatest contiguous chosen slot reflected in replica i's currently open materialized image; it may be newer than the last power-loss-safe state anchor
- **durable-state frontier** A_i: the greatest contiguous chosen slot reflected in a synchronized SQLite image and its durable state-anchor record
- **persisted frontier** P_i: the greatest contiguous slot durably present in replica i's journal
- **cluster trim** G: the greatest chosen prefix that every current data replica had durably materialized when the trim record was chosen
- **local delete floor** T_i: the greatest slot through which replica i has physically deleted journal records; T_i may lag G
- **history anchor** H_s: a cryptographic commitment to the ordered command prefix through slot s

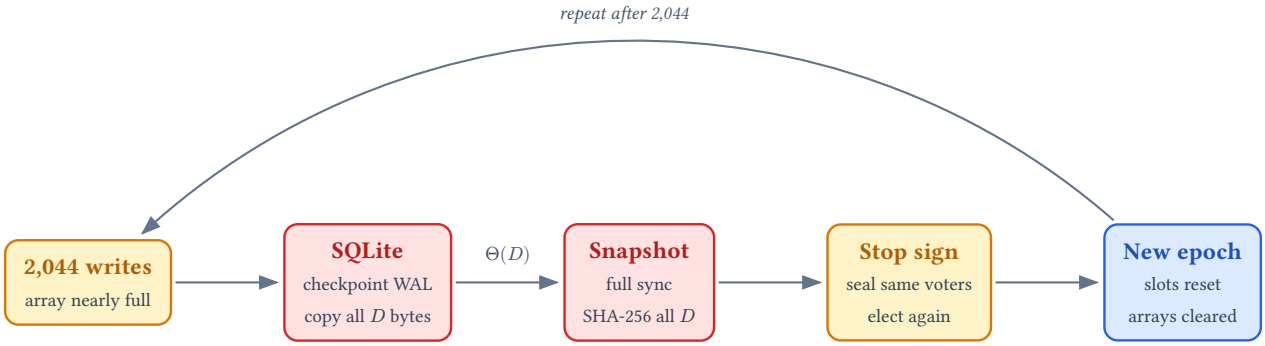
- **transfer lease**: a chosen or reconfiguration-bound lower cap S_l that prevents deletion of the suffix needed by an in-flight state receiver
- **retention horizon** R: extra physically retained history, measured by payload bytes and minimum time as well as slots, below the logical trim
- **state holder**: an independent failure domain with a verified materialized image and state anchor

This record is limited to the Paxos slot limit, journal retention, checkpoint authority, recovery, and the zaxonlite coupling that causes periodic full database copies. Vector indexing, `vec0`, ranking fusion, query execution, sharding, and cross-database transactions are explicitly out of scope.

5. Problem Statement

The implementation has four coupled bounds.

1. `src/protocol.zig` defines `Slot = u32` and allocates accepted, committed, recovered, proposal, acknowledgement, message, write, and bit-set storage from `options.max_slots`.
2. `src/replicated_log.zig` maps `max_entries` directly to that lifetime slot limit, and `src/learner.zig` retains a fixed array indexed from slot one.
3. `zaxonlite/src/types.zig` chooses `max_entries = 2048`. `epochNearlyFull()` reserves four entries, so ordinary writes stop near 2,044.
4. `zaxonlite/src/node.zig` resolves that stop by checkpointing SQLite, copying `current.db`, synchronizing the copy, hashing the whole copy, and completing a same-member configuration rollover.



The full snapshot is not required for Paxos agreement. It exists because the volatile representation cannot advance past its compile-time array. A membership primitive is therefore being used as a memory-reclamation primitive, and application state size appears in the normal consensus cost.

5.1. Why a larger epoch is insufficient

Let the epoch contain E commands and let the database reach size D. Even if E is raised by a factor of k, the amortized snapshot term is only divided by k. It is not eliminated. A sufficiently large D again dominates. A larger fixed array also increases every node's static memory and derived effect capacities, even when only a small pipeline is active.

The duty-cycle inequality makes the scale concrete. With the illustrative $B = 1$ GB/s model, sustaining $D = 1$ TB at 10^4 writes/s while holding rollover below 1% requires approximately $E = 2 \cdot 10^9$ commands. That is about one million times the present bound and is plainly not a reasonable compile-time array.

5.2. Why SQLite's size limit does not solve it

SQLite can address very large files, but that says nothing about how a host replication layer retains consensus metadata. SQLite's pager, WAL, and backup interfaces already support incremental operation. Zaxonlite introduces the 2,044-commit discontinuity above SQLite. The correction belongs at the replication and recovery boundary.

6. Goals and Non-Goals

6.1. Goals

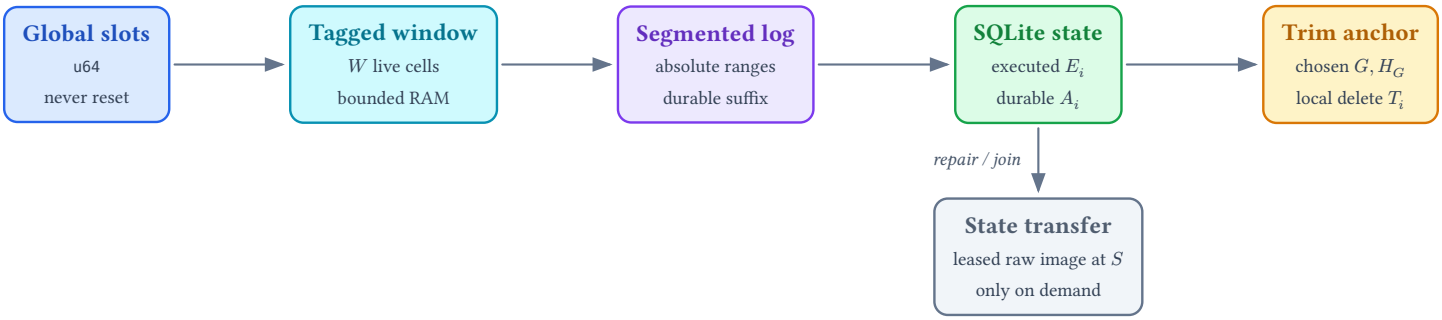
- Support a database lifetime of at least $2^{64} - 1$ logical slots without periodic slot reset.
- Bound consensus RAM by configured concurrency and recovery windows, not by lifetime commit count.
- Remove complete database copy, complete database hash, and same-member reconfiguration from normal log reclamation.
- Preserve Paxos agreement and prefix consistency across trimming, crash, recovery, and membership change.
- Keep exact captured SQLite page-image replay, including final file length, deterministic and idempotent.
- Permit slow or replaced replicas to recover by log suffix when possible and verified state transfer when necessary.
- Preserve one through nine voters and the present data-voter/witness roles.
- Specify a clean wire and durable format cut without decoder guessing or a backward-compatibility bridge.
- Make safety arguments executable through a small formal model and crash matrices.
- Hold the steady-state Paxos benchmark regression to a reviewed threshold and eliminate database-size-correlated rollover latency.
- Derive window, recovery-chunk, retention, and future holder thresholds from measurable workload and failure parameters rather than unexplained constants.

6.2. Non-Goals

- No vector or ANN index design.
- No Byzantine consensus or adversarial quorum certificates.
- No multi-writer SQLite execution.
- No automatic sharding or distributed SQL planner.
- No requirement to retain all historical values in RAM or on every node.
- No guarantee that a node offline beyond the retention horizon can recover without transferring state.
- No quorum or hard trim in the first implementation. A failed data voter is recovered or replaced before the conservative cluster trim advances.
- No replacement of SHA-256 as the existing integrity primitive.
- No mandatory disaggregated storage service.
- No Merkle inclusion or consistency proofs for third-party, Byzantine, or offline auditors in the first implementation.

7. Design Overview

The proposal separates five monotonic layers.



Slots identify decisions. The in-memory window accelerates active decisions. Segments retain a recoverable suffix. SQLite materializes the chosen prefix. The applied anchor proves how far that materialization is durable. A trim record authorizes deletion. None of those concepts changes the voter set.

8. System Model and Axioms

Let a configuration contain $N = 2f + 1$ voters and let a write quorum contain $q = f + 1$ voters. Nodes are non-Byzantine and may crash, restart, lose messages, duplicate messages, or receive them out of order.

Axiom 1 — Quorum intersection

Any two quorums of size q intersect because $2q = 2f + 2 > 2f + 1 = N$.

Axiom 2 — Ordered stable storage

If a node acknowledges durable write y after ordering durable write x before it with the platform's supported barrier, recovery cannot expose y while losing x. Torn or incomplete final records are detected by length, sequence, and checksum.

Axiom 3 — Deterministic page application

A committed zaxonlite payload contains page numbers and complete page images plus the final committed database page count. Applying the same valid payload twice writes identical bytes to identical offsets and sets the same final file length. It is therefore idempotent for both database growth and shrink.

Axiom 4 — Paxos safety

For a fixed global slot and configuration history, at most one command value is chosen. Phase one recovers a previously chosen or potentially chosen value before a leader may choose another.

Axiom 5 — Authenticated configuration

Only an authenticated current voter contributes to a Paxos quorum, durable- state report, chosen trim, or transfer lease. Membership transitions obey the stop-sign and decided-registry rules of ZDS 0008.

9. Detailed Design

9.1. Global slot space

`Slot` becomes `u64`. The first command in a new-format database uses slot one. Each successful allocation increments `next_slot`; no configuration transition resets it. Slot zero remains the genesis anchor and sentinel.

The database identity names one global slot line. Configuration ID and slot are carried separately:

```
Instance = { database_id, configuration_id, global_slot }
```

A membership stop occupies an ordinary global slot. The next configuration begins at the following global slot. Its phase-one recovery includes the retained suffix and anchor inherited from the sealed configuration.

Overflow is explicit. A database at `maxInt(u64)` returns `GlobalSlotExhausted`; it never wraps to zero. At one million commits per second, exhausting 64 bits requires more than 584,000 years.

9.2. Slot-tagged consensus window

The core option `max_slots` is replaced by `window_slots = W`, where W is a power of two. It limits concurrent unresolved and locally cached instances, not lifetime history. A physical cell is selected by

$$j(s) = s \wedge (W - 1)$$

and contains an explicit tag:

```
WindowCell(Value) {
    slot: u64,
    state: empty | accepted | chosen | delivered,
    ballot: Ballot,
    value: ?Value,
    acknowledgements: MemberSet,
}
```

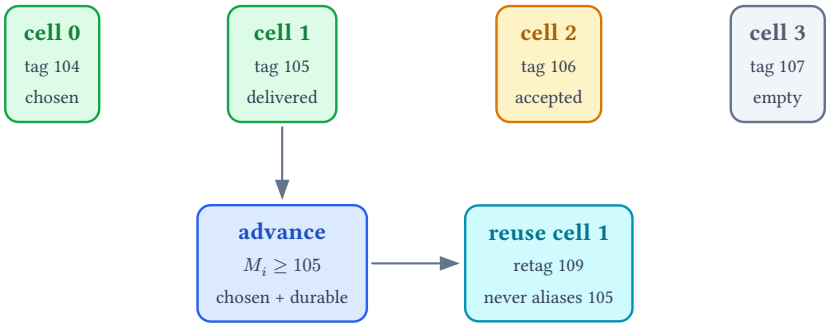
A lookup for s succeeds only when `cell.slot == s`. Physical-cell reuse and durable-log deletion are different events. Let M_i be replica i's memory floor. A cell for slot s may be cleared and retagged for $s + W$ only if

$$s \leq M_i \leq \min(P_i, C_i, \text{host\_consumed}_i)$$

and all of these are true:

- the slot is known chosen, not merely accepted;
- its accept/commit obligation is durable in the journal;
- every emitted effect through the slot has been durably consumed by the host;
- Phase 1 can retrieve the chosen/accepted state from the journal while $s > T_i$, or answer from the trim anchor while $s \leq T_i$.

An accepted-only cell is never evicted. A chosen cell may leave RAM before its journal segment is deleted.



Lemma 1 — Tagged-cell non-aliasing and obligation preservation

For distinct logical slots $s \neq t$ with $j(s) = j(t)$, a lookup for s cannot return state belonging to t; and reusing the physical cell for a chosen slot does not erase the acceptor's Phase-1 obligation.

Proof, identity. The lookup predicate requires both equal physical index and equal tag. The stored tag is one integer. It cannot equal distinct integers s and t simultaneously. Therefore reuse changes a cell's occupant but cannot change the identity of a successful lookup.

Proof, obligation. Reuse requires $s \leq P_i$ and $s \leq C_i$, so the value is chosen and its durable journal evidence exists. Before local deletion, Phase 1 reads that evidence from the journal. After local deletion, the durable trim anchor states that every slot through T_i is chosen, and the trimmed Phase-1 rule below forbids a leader from proposing into that prefix. Therefore only the storage location changes; the acceptor never answers a later ballot as if the slot were open. □

The leader applies backpressure before overwriting an active cell:

$$\text{next\_slot} - M_i \leq W$$

`WindowFull` is a transient flow-control result. It is not a terminal log limit and does not trigger a snapshot or configuration change.

The non-strict inequality is intentional. `next_slot` is the first unallocated slot, so the occupied interval $[M_i + 1, \text{next\_slot} - 1]$ contains `next_slot - M_i - 1` allocated slots. Allocation is allowed while the new occupancy will be at most W; there is no reserve cell.

9.3. Bounded Phase One and catch-up

The present promise can derive capacities proportional to `max_slots`. Version 2 replaces an all-history promise with a bounded range:

```
PromiseV2 {
    ballot
    anchor = { local_delete_floor, chosen_trim_slot, history_hash }
    accepted_range = [lo, hi]
    accepted[]
    chosen_through
    more
}
```

For an acceptor with chosen trim G_i, every complete promise states:

```
all slots 1...G_i are chosen under history anchor H_G_i
no vote is reported at or below G_i
every accepted or chosen record in the advertised range is returned
```

Amended: the interval frame itself echoes the leader's requested chunk for per-chunk counting under reordering; the invariant attaches to the reported votes, and the leader's F and K fences make the frame inert (see the implementation note "Promise ranges bound reported votes, not the interval frame"). The receiver drops overlapping or oversized counts, a hidden gap with `more = false` is caught by per-chunk counting, and a local delete floor never exceeds the chosen trim by construction of T_i.

After collecting a complete Phase-1 quorum Q_1, the prospective leader sets

$$F = \max_{i \in Q_1} G_i$$

.

It treats every slot $s \le F$ as permanently chosen. It never fills, proposes, or accepts a client value into that prefix. It recovers every accepted slot above F using the ordinary highest-ballot rule, fills only genuine gaps above F, and chooses its first new slot strictly above both F and the greatest recovered slot. Before serving application writes, a data leader must also possess materialized state through F or install it.

This is why bytes below the trim can disappear: the later answer is stronger— "chosen under this durable prefix anchor"—not weaker—"nothing accepted." The leader requests journal evidence in chunks of at most `recovery_chunk_slots`. `more` causes another bounded exchange; it does not authorize skipping a range.

Catch-up similarly requests `[from, min(from + chunk - 1, C)]`. Chunks are credit-based and may be pipelined; `ceil(W/R)` is a message-count bound, not necessarily that many sequential RTTs. Messages, effects, and writes are bounded by window and chunk sizes. No network frame or stack object is proportional to database lifetime.

Lemma 2 — Trimmed Phase-1 preservation

Let a complete Phase-1 quorum report anchors G_i and let $F = \max_{i \in Q_1} G_i$. If the leader follows the rule above, no later proposal can choose a value different from the already chosen value in any slot $s \le F$.

Proof. Each valid anchor is emitted only after its trim record is chosen, and that record asserts a contiguous chosen prefix. The leader proposes no value at or below the greatest such prefix. Cell absence and journal absence therefore cannot be interpreted as an empty Paxos instance. Any later value at $s \le F$ would require the leader to violate the selection rule. □

9.4. Segmented consensus journal

The one-file-per-configuration journal becomes a sequence of immutable data segments plus a small active segment:

```
consensus/
  MANIFEST
  0000000000000001-0000000000010000.zxj
  0000000000010001-0000000000020000.zxj
  active.zxj
```

```
  APPLIED.0
  APPLIED.1
  TRIM
```

Each v2 segment header contains:

```
magic = ZXS2
format_version = 2
database_id
first_global_slot: u64
previous_segment_digest: [32]u8
```

Each record retains canonical kind, sequence, length, bytes, and checksum. A sealed trailer contains `last_global_slot`, record count, segment digest, and a sparse table from every kth slot to byte offset. Segment names are hints; headers and hashes are authoritative.

Replay is streaming. The current v1 implementation allocates the entire journal file in `journal.zig:152–155`; that is acceptable only because an epoch is bounded. Journal v2 reads one fixed header and then at most one bounded record buffer, validates it, applies it, and advances the file offset. Sparse indexes are themselves bounded and paged. Recovery memory is $O(\text{max\_record\_bytes} + \text{index\_page\_bytes})$, never $O(\text{segment\_bytes})$.

Rotation writes and syncs the trailer, writes a new manifest generation, atomically selects it, syncs the directory, and only then opens the next active segment. A crash exposes either the old manifest plus old active file or the new complete generation. Recovery never infers a missing interior segment from filenames.

The segment digest chain is an integrity and gap-detection mechanism, not a Byzantine proof. Existing authenticated transport and quorum rules remain the source of consensus authority.

9.5. Ordered history anchor

Zaxonlite already maintains `TransactionBatch.base_chain_hash` and `result_chain_hash`. That chain validates transaction-batch causality and is checked on every apply, but it deliberately omits noops, read barriers, stop signs, trim records, configuration ID, and the Paxos slot. The global anchor does not replace or silently duplicate that invariant: it commits to the whole chosen log and folds the existing `result_chain_hash` into transaction leaves.

Every leaf uses one fixed-width canonical encoding:

```
HistoryLeafV1 {
    version: u16 = 1
    database_id: u128
    configuration_id: u64
    global_slot: u64
    kind: u8
    command_or_stop: fixed canonical bytes
    payload_digest: [32]u8       // zero when absent
    result_chain_hash: [32]u8    // zero for non-transaction entries
}
```

Let `leaf_bytes(s)` be exactly that encoding. Define

$$H_0 = \text{SHA256}(\text{0x00} \parallel \text{LE16}(1) \parallel \text{LE128}(\text{database\_id}))$$

$$L_s = \text{SHA256}(\text{0x01} \parallel \text{leaf\_bytes}(s))$$

$$H_s = \text{SHA256}(\text{0x02} \parallel H_{s-1} \parallel L_s)$$

.

The one-byte domains and fixed-width encoding remove concatenation ambiguity. They follow the same discipline as `command.chainStep` and the leaf/node domain separation in RFC 6962/9162.

Every applied anchor, segment boundary, trim record, and state-transfer manifest binds a pair (s, H_s). The hash chain commits to order and content; it does not claim to be a complete hash of the SQLite file or a Byzantine proof.

> **Lemma 3 — Prefix-anchor uniqueness**
>
> Assuming SHA-256 collision resistance and canonical encoding, two histories with the same database ID and the same anchor (s, H_s) contain the same ordered entry digests through s, except with negligible collision probability.
>
> **Proof sketch.** Choose the greatest position at which the histories differ. The inputs to that position's hash differ while its output, required by all later equal anchors, must be equal. This yields a collision in SHA-256 or in a canonical entry digest. □

A node does not prove an arbitrary old-to-new chain extension after deleting the intervening leaves. It obtains (s, H_s) from authenticated current voters, and a state receiver requires matching reports from a read quorum before it accepts the anchor. This is sufficient in the stated crash-fault model: current voters do not forge a history, and Paxos supplies the authority.

A Certificate-Transparency history tree or Merkle Mountain Range would add $O(\log n)$ inclusion and consistency proofs for offline or Byzantine verification. It would also require retained internal nodes or proof material, canonical peak/root rules, and worst-case $O(\log n)$ merge work on particular appends. Those audit proofs are useful but are not required to prevent a Paxos leader from reproposing below a chosen trim. They are deferred rather than smuggled into the first release as a supposedly constant-cost hash.

9.6. Durable SQLite applied anchor

Each data replica maintains two alternating fixed-size records, `APPLIED.0` and `APPLIED.1`:

```
magic = ZXAP
version = 1
generation: u64
database_id
global_slot: u64
history_hash: [32]u8
sqlite_page_size: u32
sqlite_page_count: u64
checksum: [32]u8
```

`current.db` is presently a rebuildable cache; ordinary offline page applies do not sync it after every command. This record does not pretend the existing `applied_slot` is already power-loss-safe. It splits volatile execution E_i from durable state A_i.

Ordinary writes continue to make payload and Paxos journal state durable before acknowledgement. Periodically, or under retention pressure, a data replica creates a state anchor while holding the node/writer mutex between commands:

```
verify contiguous execution through s and H_s
  -> finish or refuse any open transaction
```

```
-> checkpoint/truncate the SQLite WAL into current.db
-> synchronize current.db
-> write inactive APPLIED generation for (s, H_s)
-> synchronize the anchor and pathname transition
-> publish A_i = s
```

This adds a periodic state-durability operation; it does not add a database sync after every write. Its cadence is controlled by log-retention pressure and recovery objectives. It reads or writes dirty database/WAL pages but never copies or hashes all D bytes on the normal path.

The node selects the valid record with the greatest generation. It never trusts a marker whose database identity, file geometry, checksum, or history anchor fails validation. A corrupt or missing marker causes conservative replay or state transfer; it never advances the frontier.

The anchor is a sidecar rather than a SQLite metadata row because the Paxos slot is known only after the leader has captured the transaction's page images. Reapplying exact page images is idempotent under Axiom 3, so a crash after database persistence but before anchor persistence is harmless.

Lemma 4 — Applied-anchor crash safety

After recovery chooses durable anchor $\left(A_i, H_{A_i}\right)$, replaying the contiguous chosen suffix $A_i + 1$ through C produces the same materialized SQLite bytes as one failure-free application of the prefix through C.

Proof. Ordered storage prevents a selected anchor from being newer than the synchronized database pages and final file length. Any page writes newer than the selected anchor came from a suffix command whose journal record is already durable. Recovery reapplies that command. Exact page-image writes and final `setLength` are idempotent by Axiom 3. By induction over the contiguous suffix, the recovered image equals the failure-free image. □

9.7. Progress frontiers

The frontiers are distinct. For every active data replica i:

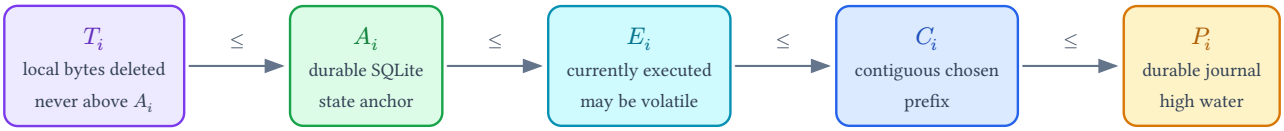
$$0 \le T_i \le A_i \le E_i \le C_i \le P_i < \text{next\_slot}$$

The cluster-wide chosen trim obeys $0 \le G \le C$. It is intentionally absent from the local chain above: a crashed or disk-repaired node may discover $G > A_i$. That condition triggers state installation before the node votes; it does not authorize local deletion. Physical deletion always sets

$$T_i \leftarrow \min\left(G, A_i, \text{retention\_cutoff}_i, \min_{l \in L} S_l\right)$$

where L is the set of active transfer leases. With no leases, the final term is $+\infty$.

Witnesses persist Paxos records but do not materialize SQLite; they report P_i and never claim A_i. A leader includes its observed chosen frontier in heartbeats. Data replicas piggyback `(durable_state_slot, history_hash)` only after the state anchor is durable; `executed_slot` is reported separately for lag monitoring and cannot authorize deletion.



9.8. Conservative cluster trim

Let D be the current data replicas and $n_d = |D|$. The first release uses the Global Last Executed rule from MultiPaxos Made Complete and HoliPaxos, but with the stronger durable-state frontier required by zaxonlite:

$$G_{\text{candidate}} = \min_{i \in D} A_i$$

.

Every report is authenticated and binds the same configuration and $H_{G_{\text{candidate}}}$. The leader may then propose one ordinary Paxos entry:

```
Trim {
    trim_id: u64
    through_slot: u64
    history_hash: [32]u8
    configuration_id: u64
    policy: all_data_replicas
}
```

Once chosen, G advances. The chosen record is the cluster-wide Phase-1 anchor; it is not a certificate that bytes have already been deleted. Each node independently advances T_i only to the minimum allowed by its durable state, retention horizon, and active transfer leases.

Segment deletion occurs after the new `TRIM` record and manifest generation are durable. Only complete segments are unlinked. A partial boundary segment is retained or rewritten as a separately synchronized generation; it is never truncated in place past uncertain bytes.

Lemma 5 — All-data-replica trim recoverability

Choosing $G = \min_{i \in D} A_i$ and deleting locally only through $T_i \le \min(G, A_i)$ cannot remove a command needed to reconstruct the materialized state of any current data replica.

Proof. By definition every current data replica has a synchronized image and durable anchor through G. By Lemma 4, recovery begins from its own $A_i \ge G$ and needs only a later suffix. The local rule independently prevents a node whose state anchor regressed after disk loss from deleting through the newer cluster trim. □

A transiently slow or offline data replica freezes G but does not stop Paxos choice until the retention budget is approached. A permanently failed data voter is replaced through ZDS 0008; the next configuration computes its trim over its own data-replica set only after the replacement's transfer lease and state installation are complete. This is the first-release answer to an indefinitely stuck frontier. It avoids a second recovery-quorum protocol.

9.9. Transfer leases

Reserved in v1. The chosen-lease lifecycle below is carried by the command codecs and the TRIM file, but no v1 path proposes a lease: the conservative trim (the minimum durable frontier over every current data replica, frozen while any report is stale) already provides the freeze Lemma 6 wants, and the sender's pinned private image copy keeps a running transfer safe. The lifecycle becomes live machinery only with the deferred quorum-trim variant; see the implementation note "Conservative trim subsumes transfer leases in v1".

The reserved lifecycle: an on-demand state transfer would create a lease before bytes are sent – carried by the decided stop announcement for a replacement, or as a chosen `TransferLease` entry for in-place repair – with the shape:

```
lease_id: u64
receiver_id
base_slot: u64
base_history_hash: [32]u8
configuration_id
expires_after_leader_ticks
```

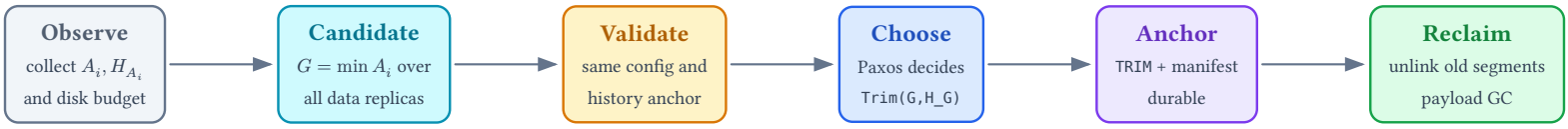
While lease l is active, every node caps physical deletion through $S_l = \text{base\_slot}$. Thus the required suffix $S_l + 1 \ldots C$ survives even if the original sender crashes. A receiver may resume from another authenticated data replica. Lease completion is chosen after the receiver durably installs the image and catches up; expiry is a leader-tick safety valve, never a wall-clock guess. An expired receiver starts again from a new anchor.

Lemma 6 — Transfer-suffix preservation

If every physical delete floor obeys $T_i \le S_l$ while lease l is active, a receiver that has durably installed the image at S_l can obtain every chosen entry required to advance beyond S_l from any surviving current node that retains the lease.

Proof. The first required entry is $S_l + 1$. The delete cap permits removal only through S_l, so every segment containing a later required slot remains retained. Because the lease is cluster-visible rather than sender-local, sender failure does not release the cap. □

9.10. Trim state machine



Trim IDs are monotonic and idempotent. A lower trim is ignored after validation. A same-ID different-anchor record is fatal corruption. Nodes may physically delete at different times. `chosen_trim_slot`, `local_delete_floor`, and `retained_first_slot` are reported separately.

9.11. Deferred quorum trim as one order statistic

Quorum trim is a future policy, not first-release machinery. Sort the current state holders' durable frontiers $A_1 \le A_2 \le \ldots \le A_{n_d}$. For a required holder count k define

$$G(k) = \max\{s : |\{i : A_i \ge s\}| \ge k\} = A_{n_d - k + 1}$$

.

The conservative rule is $k = n_d$. A future quorum trim might use a lower k, but the review must answer three independent questions:

- **Agreement.** Independent of k once every deleted slot is chosen and the trimmed Phase-1 rule is enforced.
- **State durability.** To survive f_{state} permanent independent state losses without external restore, require $k \ge f_{\text{state}} + 1$.

- **Recovery availability.** If holders are voters and recovery must be possible whenever any quorum of size q is reachable, require $k + q > N$. External archives help durability but do not satisfy this intersection condition.

Theorem 6 — State-durability threshold

A trimmed state prefix stored by k independent holders survives every set of at most f_{state} permanent holder losses if and only if $k \geq f_{\text{state}} + 1$.

Proof. If $k \geq f_{\text{state}} + 1$, removing at most f_{state} holders leaves at least one. Conversely, if $k \leq f_{\text{state}}$, the allowed failure set may contain all k holders, leaving no materialized recovery source. □

Theorem 7 — Recovery-availability intersection

Let state holders be a subset of the N voters. A holder is guaranteed to be present in every reachable voter quorum of size q if and only if $k + q > N$.

Proof. If $k + q > N$, two subsets of those sizes cannot be disjoint. If $k + q \leq N$, choose the q reachable voters entirely from the complement of the k holders; recovery then has a quorum but no holder. □

Theorem 1 supplies agreement and contains no k. Theorems 6 and 7 separately supply state durability and recovery availability. No one theorem is used as a substitute for the other two.

Flexible Paxos may change the relevant Phase-1 quorum size, but that trades a smaller holder intersection against election availability. It is not an automatic reason to lower k.

For capacity planning, independent per-holder loss probability p and a common-mode floor β give the illustrative model

$$P_{\text{state loss}}(k) = \beta + (1 - \beta)p^k$$

.

Increasing k cannot beat β; failure-domain independence and external backups matter more than replica count once the common-mode floor dominates. Witnesses never count as state holders. A topology with fewer than $f_{\text{state}} + 1$ independent materialized holders may still preserve Paxos agreement, but it must advertise the weaker state-durability tier rather than claim tolerance of arbitrary f_{state} state losses.

9.12. Payload garbage collection

Sealed segment trailers contain a sorted set or compact manifest of referenced payload digests. A payload object is reachable if any retained segment, unsealed active record, pending state transfer, or chosen-but-not-applied entry references it. It may be deleted only when:

$$\text{last\_reference\_slot} \leq T_i \wedge A_i \geq \text{last\_reference\_slot}$$

and no retained manifest names it. Garbage collection writes a candidate reachability generation, verifies it against retained segments, atomically publishes it, and then unlinks unreachable objects. Crashes may leak objects; they may not delete reachable ones.

9.13. Recovery protocol

Startup follows a strict ladder.

1. Select the valid highest `APPLIED` generation and its $\left(A_i, H_{A_i}\right)$.
2. Select the valid journal manifest and durable `TRIM` anchor.
3. If $A_i \geq G$, replay the contiguous journal suffix from $A_i + 1$.
4. If the suffix has a gap, stop before voting and request range repair.
5. If $A_i < G$ or the local SQLite image is lost, install state from a current authenticated data replica under a cluster-wide transfer lease. *Amended for v1:* this step is implemented for a joining replacement (empty consensus state); a damaged established voter fails closed at open and is repaired by ZDS 0008 replacement instead – see the implementation note "Damaged-voter repair is replacement in v1".
6. Verify the transferred database digest and manifest anchor at slot S, atomically select it, then replay $S + 1$ through the current chosen frontier.

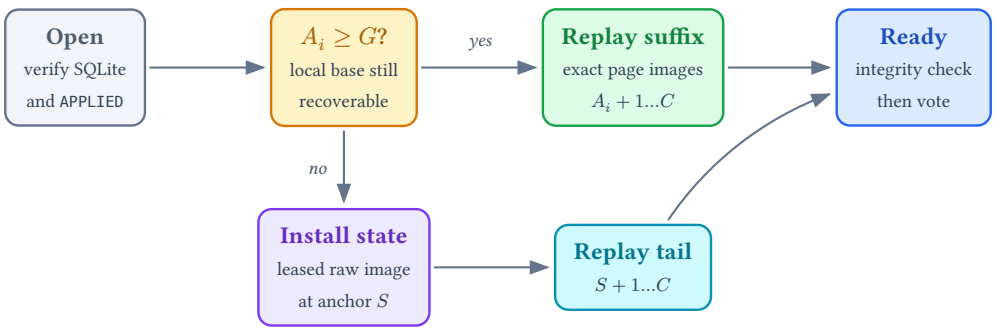
The first implementation uses an anchor-pinned raw copy, reusing the byte-exact invariant the existing follower snapshot path enforces. The source:

1. pins a fresh durable anchor at base S (v1; the reserved lease choice replaces this step under quorum trim);
2. holds the node/writer mutex between commands;
3. checkpoints/truncates the WAL, synchronizes `current.db`, and publishes durable state anchor (S, H_S);
4. creates a raw byte copy while the materialized image cannot change;
5. synchronizes and hashes that copy, writes a manifest containing file length, page geometry, S, H_S, database ID, and registry digest; and
6. releases the mutex and streams the immutable generation in bounded chunks.

A verified filesystem reflink may replace the physical copy when it provides a point-in-time copy-on-write image with the required durability semantics. If no such primitive exists, the raw copy holds the writer mutex for its duration. That can be expensive for a multi-terabyte image, but it occurs only during bootstrap, repair, or replacement and makes the cost visible at the recovery event instead of every 2,044 writes.

SQLite documents the completed Online Backup API as a consistent snapshot and, in relevant modes, a bitwise copy; therefore this record does not repeat the unsupported claim that backup necessarily rewrites header counters. It is nevertheless excluded from v1 because incremental backup permits source writes between `backup_step` calls and may restart or incorporate later writes. The current zaxonlite protocol has no reviewed mechanism that binds the completed image to one exact Paxos slot under that concurrency.

`VACUUM INTO` is explicitly forbidden for replicated state transfer. It may repack and renumber database pages; subsequent exact-page-number payload replay would then apply changes to a different physical layout.



9.14. Membership change

Stop signs remain the only membership-change primitive. A chosen stop sign names the next registry as in ZDS 0008, but it does not require a database snapshot and does not reset slots. Before a new voter activates, it installs a certified state and suffix through the activation frontier. Surviving voters carry the same global trim anchor into the next configuration.

The stop/reconfiguration announcement freezes physical deletion for the replacement: in v1 the conservative trim itself supplies the freeze (the joiner's zero frontier pins G), with the announcement-carried lease reserved for the quorum-trim variant. The next configuration does not advance G until the new data voter has installed the state and published its first durable-state anchor. Removed nodes do not count in the next configuration's conservative minimum.

9.15. Backpressure and storage budget

The system has flow-control limits, not a lifetime command limit:

$$\text{next\_slot} - M_{\text{leader}} \le W$$

$$P_{\text{leader}} - \min_{i \in \text{ write\_quorum}} P_i \le L_{\text{replication}}$$

$$C - \min_{i \in \text{ healthy\_data}} E_i \le L_{\text{execute}}$$

Crossing a soft threshold starts segment sealing, trim collection, or replica repair. Crossing a hard local disk threshold rejects new writes with `RecoveryRetentionExceeded` until a safe trim, state transfer, or operator capacity change succeeds. The node never deletes unproven history to remain available.

9.16. Deriving the window and recovery chunk

The record does not standardize `4096` and `256` as replacement magic numbers. Let d_p be the measured pth-percentile time from slot allocation until memory-floor eligibility, λ_p the corresponding peak admitted rate, and $\rho < 1$ the target window occupancy. Little's Law gives the lower bound

$$W_{\min} = \left\lceil \lambda_p \frac{d_p}{\rho} \right\rceil$$

.

Round up to a power of two for mask indexing, then enforce two upper bounds:

$$W b_{\text{cell}} \le M_{\text{consensus budget}}$$

$$\left\lceil \frac{W}{c \cdot R} \right\rceil \text{RTT} + W \frac{b_{\text{recovery}}}{B_{\text{network}}} \le L_{\text{failover}}$$

where R is records per chunk and c is the maximum pipelined chunk credit. If no W satisfies all bounds, the throughput, latency, memory, and failover objectives are mutually inconsistent and configuration must fail explicitly.

`recovery_chunk_slots` is then the greatest record count fitting the wire-frame bound and per-peer recovery-memory budget. It is not chosen independently of command size, c, or network bandwidth.

9.17. Sizing physical retention

Retention is accounted in total payload-store bytes plus journal and index bytes. `journal_bytes` alone is misleading because `Command` is a fixed 161 bytes while page-image payloads may be kilobytes or megabytes.

At minimum, a target no-transfer outage quantile X_p requires

$$R_{\text{slots}} \geq \lambda X_p$$

subject to the measured byte distribution per slot. A simple economic model makes the trade explicit. Let c be mean retained bytes per slot, X outage duration, ν expected outage events per second, D state-transfer bytes, and γ the operator's cost exchange between retained byte-seconds and transferred bytes. Then

$$J(R) = \gamma c R + \nu D \Pr\left(X > \frac{R}{\lambda}\right)$$

.

For a differentiable outage distribution, an interior optimum satisfies

$$f_{X(\frac{R}{\lambda})} = \gamma c \frac{\lambda}{\nu D}$$

.

For exponential outages with rate μ,

$$R^* = \frac{\lambda}{\mu} \cdot \ln\left(\frac{\nu D \mu}{\gamma c \lambda}\right)$$

when the logarithm is positive. The result is linear in write rate and mean outage duration but only logarithmic in database size. Heavy-tailed outage distributions produce materially larger horizons; operators must measure a tail or choose an explicit coverage quantile, not size from the mean alone.

Illustratively, $\lambda = \frac{5000}{s}$, mean outage $300s$, $D = 100$ GB, $c = 4.2$ KB, $\nu = \frac{2}{\text{week}}$, and $\gamma = \frac{10^{-6}}{s}$ give about $5.9 \cdot 10^6$ slots or 25 GB. The units and prices are policy inputs, not universal defaults. This design moves recurring full-state work out of the write dimension and deliberately spends bounded recovery space instead.

10. Safety Proof

Theorem 1 — Agreement survives window reuse and trimming

Under Axioms 1, 2, 4, and 5, no two different commands can be chosen for the same global slot after any sequence of window reuse, segment deletion, crash recovery, and unchanged-configuration trims.

Proof sketch. Before trimming, ordinary Paxos agreement follows Axiom 4. Window reuse cannot substitute one slot's accepted state for another by Lemma 1. A trim record is itself chosen by Paxos and binds the unique prefix anchor from Lemma 3. Every later Phase 1 applies Lemma 2: it treats the greatest quorum-reported anchor as chosen and proposes only above it, while recovering every accepted slot above it. The holder count is irrelevant to agreement. Thus deletion removes bytes but not the protocol fact that the prefix is closed. Any conflicting choice would violate ordinary Paxos agreement or the trimmed leader-selection rule. □

Theorem 2 — Recovered-state prefix correctness

A data replica that reaches `Ready` after the recovery ladder materializes exactly the ordered chosen command prefix through its reported applied frontier.

Proof. A local base is safe by Lemma 4. A transferred base is an immutable raw copy made while the source is pinned at S; its manifest digest is checked end to end and its (S, H_S) must match a current read quorum. Lemma 6 preserves its required suffix. In both branches the node replays one contiguous sequence of chosen, digest-bound page-image payloads. Induction on the suffix and Axiom 3 give the exact prefix state. A gap or mismatch stops activation, so no other path reaches `Ready`. □

Theorem 3 — Lifetime-independent volatile memory

For fixed member bound M, consensus window W, recovery chunk R, and maximum in-core value bytes B_v, volatile consensus memory is independent of lifetime slot count n.

Proof. The window holds W tagged cells and at most W acknowledgement sets of size $O(M)$. Phase one, catch-up, effects, and pending writes hold at most $O(R)$ records plus member metadata. All older values reside in segments or materialized state. Therefore memory is $O(W \cdot (B_v + M) + R \cdot B_v + M)$, with no n term. In zaxonlite the in-core command is currently 161 bytes and page images remain in the payload store; payload retention is a disk-byte budget, not hidden RAM. □

Theorem 4 — Routine reclamation is database-size independent

Excluding exceptional state transfer, the work to rotate and trim a journal segment does not depend on SQLite database size D.

Proof. Rotation writes one bounded trailer, sparse index, manifest, and active header. Trimming persists one bounded anchor and unlinks complete segment files plus unreachable payloads. No operation reads, copies, or hashes `current.db`. Its work is $O(\text{records\_in\_segment})$ when sealing or $O(\text{segments\_deleted} + \text{payloads\_reclaimed})$ when trimming, with no D term. □

10.1. Conditional liveness

Theorem 5 — Progress under bounded lag

If a stable leader communicates with a healthy write quorum, durable storage eventually completes, application throughput eventually exceeds offered write throughput, and the configured windows admit at least one new slot, then commands continue to be chosen without snapshot rollover.

Argument. Stable Multi-Paxos chooses each proposed slot. Eventual persistence and execution advance P, C, and E; periodic state anchors advance A. Their advance frees tagged cells and enables conservative trim. A permanently failed data voter must recover or be replaced before G advances. No normal trim requires copying the database. As usual, this is conditional liveness: permanent overload, unreplaced failure, or exhausted storage invokes explicit backpressure rather than unsafe deletion. □

11. Durable and Wire Formats

This change is format-breaking and must be versioned as one coherent feature.

Boundary	Old	New	Reason
Paxos slot	`u32`	`u64`	Global non-resetting instance number.
Wire protocol	`8`	`9`	64-bit slots, range promises, durable-state reports, trim anchors, and transfer leases.
Journal	`1`	`2`	Segment headers, absolute ranges, digest chain, sparse index, and trim anchor.
Applied state	implicit	`ZXAP` v1	Crash-safe SQLite recovery frontier.
Checkpoint proof	`ZXP2`	`ZXP3`	Global anchor, state-transfer lease, raw-image digest, and 64-bit fields.
Snapshot manifest	`1`	`2`	State transfer rather than epoch reset; global slot and history anchor.

All integer encodings are canonical little endian. Decoders reject unknown versions, non-canonical ranges, integer overflow, slot zero where prohibited, and anchors beyond a locally proven chosen frontier. Wire version remains exact-major after activation.

12. API Changes

The core's conceptual options become:

```
pub const Options = struct {
    max_members: usize,
    window_slots: usize,          // derived from measured pipeline bounds
    recovery_chunk_slots: usize, // derived from frame and memory bounds
    max_batch: usize = 16,
};
```

The core adds host-owned progress callbacks or explicit inputs:

```
advanceMemoryFloor(through: Slot) !void
installChosenTrim(anchor: TrimAnchor) !void
beginRecovery(anchor: TrimAnchor) !void
requestRange(peer: MemberId, first: Slot, count: u32, effects: *Effects) !void
```

`decidedThrough()`, proposal results, effects, promises, catches-up, learner messages, status, C ABI surfaces, and client JSON widen slots to `u64`. Historical random access below the memory floor moves out of the core into the host journal API. A call for an unavailable old slot returns `Trimmed` with the current anchor; it never indexes a reused cell.

13. Specific Change Surface

The narrowed first release is expected to touch approximately 24 to 32 files. Six to eight contain substantial algorithm or storage work; the rest are format, tests, benchmark, status, and documentation propagation. This estimate is a review aid, not permission to broaden implementation silently.

File or area	Required change
`src/protocol.zig`	Widen `Slot`; replace lifetime arrays and bit sets with tagged window cells; bound phase one and effects; add anchor/range validation and transient window backpressure.
`src/replicated_log.zig`	Replace `max_entries` lifetime semantics with window semantics; retain stop signs only for membership; expose memory-floor and trim-anchor integration.
`src/learner.zig`	Use a tagged delivery window and absolute learned/released frontiers.
`src/host_managed.zig`, `src/root.zig`, `src/errors.zig`	Export `u64` slots and new host contract; distinguish `WindowFull`, `Trimmed`, anchor mismatch, and true exhaustion.
`zaxonlite/src/types.zig`	Replace `max_entries = 2048` with consensus-window, recovery-chunk, segment, and retention-budget options.
`zaxonlite/src/journal.zig`	Implement journal v2 segments, active recovery, sparse index, digest chain, manifests, atomic rotation, bounded-buffer streaming replay, and safe segment deletion.
`zaxonlite/src/applied_anchor.zig` (new)	Canonical alternating `ZXAP` records, generation selection, geometry and history verification, and durable publication.

File or area	Required change
`zaxonlite/src/trim.zig` (new)	All-data-replica frontier observation, chosen trim encoding, transfer leases, retention policy, and idempotent durable trim state.
`zaxonlite/src/node.zig`	Remove capacity-triggered snapshot rollover; maintain global frontiers and history anchor; apply/persist ordering; range recovery; trim and payload-GC orchestration; on-demand state transfer. Retire `epochNearlyFull()` and `ensureEpochCapacity()` rather than leaving dead epoch semantics in the public/internal surface.
`zaxonlite/src/wal.zig`	Document and enforce the page-write/apply-anchor durability order while preserving deterministic exact-page replay.
`zaxonlite/src/checkpoint_proof.zig`	Add proof v3 for the global raw-image anchor and transfer lease; retain strict database and registry binding.
`zaxonlite/src/payload_store.zig`	Move reachability from epoch journals to retained segment manifests and active transfer leases.
`zaxonlite/src/wire.zig`	Protocol v9; widen every slot; add bounded range, durable-state frontier, chosen-trim, transfer-lease, and transfer-anchor frames.
`zaxonlite/src/server.zig`, `client.zig`, `main.zig`, `cli/render.zig`	Propagate 64-bit status and errors; expose global slot, trim, retained range, apply lag, segment bytes, and transfer state.
`zaxonlite/src/*test.zig`, `src/*test coverage`, `sim/`, `specs/`	Replace reset-based expectations; add long-run, wrap-around, trim, crash, transfer, reconfiguration, corruption, and model-based coverage.
`benchmarks/benchmark.zig`, durable and zaxonlite benchmarks	Measure a moving window over millions of slots, segment boundaries, trimming, lag, and database-size-independent tail latency.
`docs`, ZDS 0002/0004/0008 amendments, release notes	Describe the new limits and formats only after acceptance and implementation. Existing committed records are not edited by this prediscussion draft.

14. Format Cut and Rollout

Backward compatibility is not required. This record defines a clean pre-release format cut:

- wire v9 accepts only wire v9;
- journal v2 readers reject journal v1;
- checkpoint proof v3 readers reject proof v2;
- state-anchor and segmented-manifest files have no legacy fallback; and
- the implementation contains no bridge mode, dual writer, rolling mixed- version voting, or downgrade path.

Existing development and benchmark directories are recreated. If a developer needs to preserve sample data, a separate offline export/import utility may be written, but it is not part of the runtime consensus protocol and carries no rolling-upgrade guarantee.

A newly created database starts at global slot one with canonical genesis

$$H_0 = \mathrm{SHA256}(0\mathrm{x}00 \parallel \mathrm{LE16}(1) \parallel \mathrm{LE128}(\mathrm{database\_id}))$$

.

All voters in a test cluster are stopped and upgraded together. Startup fails closed on any old durable format. This materially reduces decoder states, crash cases, implementation files, and verification burden.

14.1. Implementation stages

1. Add `u64` slot types, tagged windows, and property tests behind an experimental core option.
2. Add journal v2, applied anchors, and replay equivalence without trimming.
3. Add conservative all-data-replica trim and segment/payload reclamation.
4. Add anchor-pinned raw state transfer (cluster-wide leases reserved; the conservative trim freeze stands in for them in v1).
5. Integrate ZDS 0008 replacement with the frozen trim frontier.
6. Make the direct format cut and add strict rejection/conformance tests.
7. Run formal, crash, long-duration, and performance gates.
8. Remove the old capacity rollover and both capacity helper APIs only after the new recovery path passes every gate.

15. Verification Plan

15.1. Formal model

Add `specs/GlobalTrim.tla` and a TLC configuration for $N \in \{3, 5\}$ with small windows $W \in \{2, 3\}$ so reuse is exercised frequently. Model:

- Paxos promises, accepts, choice, and leader change;
- global slot allocation and tagged cell reuse;
- persisted, chosen, executed, durable-state, cluster-trim, memory, and local delete frontiers;
- conservative trim and trimmed Phase-1 replies;
- crashes between every durability step;
- local replay, transfer leases, sender loss, and raw state transfer;
- one membership transition; and
- witness versus data-replica roles.

Check these invariants:

```
Agreement
ChosenPrefix
TagNonAliasing
TrimNeverExceedsCertifiedAnchor
AppliedNeverExceedsDurablePages
AcceptedOnlySlotNeverEvicted
PromiseRangeStartsAboveAnchor
LeaderNeverProposesAtOrBelowAnchor
LocalDeleteNeverExceedsDurableState
TransferLeasePreservesSuffix
RecoveryReadyImpliesExactPrefix
```

`NoWitnessClaimsMaterializedState`
`GlobalSlotNeverDecreases`

The model must include an action-to-code map in `specs/README.md`. Safety is checked without fairness. Conditional liveness checks weak fairness only for message delivery and successful durable writes.

15.2. Deterministic and property tests

- run at least ten million global slots with a tiny W and prove bounded RSS;
- force physical-cell wrap on every few proposals and compare against an unbounded reference model;
- split phase-one replies at every chunk boundary and reorder chunks;
- make an entire Phase-1 quorum contain only trimmed acceptors and verify that the new leader never fills the anchored prefix;
- inject gaps, duplicates, stale tags, stale trim IDs, conflicting anchors, and near-`u64` overflow;
- compare in-memory continuation with crash/restart at every segment rotation;
- vary data voters and witnesses; prove witnesses never report durable materialized-state frontiers;
- perform membership change without resetting the global slot;
- kill the original state-transfer sender and complete from another data replica (in v1 the conservative trim freeze, not a lease, keeps the base retained);
- retain and reclaim payloads across shared segment references;
- verify old-history API calls return `Trimmed`, never a newer value.

15.3. Crash matrix

Crash before and after each of:

1. payload persistence;
2. chosen journal persistence;
3. page-image write;
4. database durability barrier;
5. inactive `APPLIED` write;
6. applied-anchor barrier;
7. segment trailer write and sync;
8. manifest generation and pointer publication;
9. durable-state report persistence and transmission;
10. trim choice;
11. local `TRIM` persistence;
12. segment unlink and directory sync;
13. payload reachability publication and unlink;
14. pinned copy (mid-copy and complete), transfer chunk, staged image, install rename, and published anchor – the shipped stateless-phase ladder; the lease-choice and lease-completion points are reserved with the lease lifecycle itself.

Every restart must either expose the previous complete generation or the next complete generation. It must never vote with a gap, an unverified image, or an anchor newer than durable page state.

15.4. Performance gates

Compare on the same host, power profile, Zig version, and commit. Before measurement, estimate coefficient of variation and compute sample size for 80% power at the 5% non-inferiority margin. The normal approximation

$$n \approx 2(z_{0.975} + z_{0.8})^2 \frac{\mathrm{CV}^2}{\delta^2}$$

requires about 64 samples per arm when `CV = 10%` and $\delta = 5\%$; 30 is not a sufficient universal gate.

- For stable-leader `u64-3n`, report the Hodges–Lehmann location shift and a bootstrap 95% confidence interval. The upper regression bound must remain at or below 5% unless review explicitly accepts a measured trade-off.
- The number of steady-state durability barriers per committed group must not increase.
- Throughput, p99, p99.9, maximum, and maximum/median ratio are measured across at least one million slots and many physical-window wraps, not only a fresh first window.
- Per-operation latency is checked for periodicity at window, segment, and state-anchor frequencies using a predeclared autocorrelation or periodogram threshold. This directly tests removal of the rollover cliff.
- Segment rotation p99 must remain independent of SQLite database size within measurement noise.
- Runs use at least 1 MB, 1 GB, and a sparse or generated large database; no normal-path event may read or copy the full database.
- A lagging replica is tested within the retained suffix and behind the trim frontier. Only the latter may pay full state-transfer cost.
- RSS is measured at 10 thousand, 1 million, and 10 million chosen slots and must remain within the configured window plus allocator tolerance.

The current benchmark that executes only 1,000 measured zaxonlite operations cannot validate this design because it never reaches the existing rollover. It remains useful for request-path comparison but is not a compaction test.

16. Security Considerations

Durable-state reports, chosen trims, and transfer leases influence recoverability and therefore require the same authenticated peer identity and configuration binding as consensus traffic. A stale voter, learner, gateway, or witness cannot claim materialized state. Duplicate replica IDs count once.

History hashes and segment hashes detect accidental corruption and bind protocol evidence. They do not turn crash-fault Paxos into Byzantine Paxos. A malicious quorum can still certify false state; protecting against that requires a different fault model and is out of scope.

State transfer exposes the database contents. Production transfer remains inside mutually authenticated TLS, validates database and registry identity, enforces chunk and total-size bounds, and writes only beneath a newly created temporary generation. Paths from remote metadata are never used directly.

Resource exhaustion is bounded. Range requests, promises, state reports, sparse indexes, manifests, transfer chunks, and outstanding leases all have explicit limits. Invalid anchors, overlapping ranges, decompression bombs, integer overflow, or excessive retries close the peer request and increment structured diagnostics.

Deletion follows least authority. The trim subsystem receives explicit opened directory handles and validated segment identities. It never constructs a recursive deletion target from network input.

17. Operational Considerations

Status adds:

```
decided_slot
applied_slot
durable_state_slot
memory_floor
chosen_trim_slot
retained_first_slot
journal_segment_count
journal_bytes
payload_retained_bytes
trim_mode
installation_state
```

Amended to the shipped surface: the list above is what v1 status exposes, under the implementation's field names. `local_delete_floor` is derivable as `retained_first_slot - 1`; apply lag as `decided_slot - applied_slot`. `retention_age_seconds` needs per-slot timestamps the journal does not record, and `transfer_lease_base` belongs to the reserved lease lifecycle; both return with the quorum-trim variant.

Alerts distinguish consensus unavailability, execution lag, durable-state lag, retention pressure, and a frozen transfer lease. A slow replica first delays conservative trim. It does not force a snapshot. Near the disk budget, the leader attempts state repair; a permanently failed data voter is replaced through ZDS 0008. If neither completes safely, writes stop before storage exhaustion with a stable operator error.

Operators choose retention by bytes and minimum time, not by total database size. Defaults should preserve enough suffix for routine outages while limiting disk amplification. Segment size is a performance parameter, never a safety parameter.

Backups remain necessary. Consensus replication protects availability, not operator deletion, correlated storage loss, or application-level corruption. An external backup is outside the first-release trim minimum. It remains the recovery authority for correlated loss and operator error.

18. Expected Benchmark Effects

The core hot path adds a 64-bit tag comparison and absolute-index arithmetic. Power-of-two masking avoids division. Clearing an entire 2,048-slot epoch and same-member election disappear. Bounded phase one may use more messages after a long leader outage, but each frame is smaller and bounded.

Zaxonlite adds periodic durable-state-anchor maintenance. The current journal barrier occurs before offline page application, so the design does not claim it can make later database writes durable for free. The anchor path may add a checkpoint and barrier at its measured cadence; it must not add one full filesystem barrier per write. Segment trailers and manifests add occasional small writes.

Expected qualitative effects are:

Workload	Current	Proposed	Risk
Fresh-window core	Very fast fixed arrays	Tag check and `u64` slots	Small constant regression to measure.
Long-running core	Stops at bound	Constant-memory continuation	Chunk recovery complexity.
Routine reclamation	Full DB copy/hash	Segment metadata/unlink	Filesystem directory latency.
Lag within retention	Epoch-specific recovery	Range suffix	More wire states.
Lag beyond retention	Snapshot generation	On-demand state transfer	Rare cost remains $\Theta(D)$.
Membership change	Snapshot + slot reset	Transfer if needed; slots continue	Cross-config proof complexity.

No claim of improvement is accepted without the verification workload above. The mathematical result is narrower but firm: normal reclamation removes the unavoidable $\frac{D}{E}$ term. Actual constants remain device and implementation properties.

19. Alternatives Considered

19.1. Raise `max_entries`

This postpones exhaustion and increases static memory. It preserves the database-size-dependent rollover term and remains a lifetime limit.

19.2. Scale epoch size with database size

This can keep amortized copy cost below a target, but makes memory and recovery bounds dynamic, still copies the database periodically, and couples unrelated dimensions. It is a mitigation, not a separation of concerns.

19.3. Reflink or copy-on-write snapshots

Filesystem clones can make the initial copy cheap, but are platform-dependent. Hashing, later copy-on-write amplification, snapshot retention, and the fixed slot bound remain. Reflinks are valuable for optional state transfer staging, not the consensus abstraction.

19.4. Incremental or Merkle-page snapshots

Changed-page snapshots reduce transfer bytes and may be added later. They do not by themselves make a finite Paxos array unbounded. The proposed global slot and trimming model is still required.

19.5. Quorum trim in the first release

Quorum trim avoids a frontier frozen by one failed data replica, but mixes agreement, state durability, and recovery availability. HoliPaxos explicitly presents it as a higher-snapshot-rate alternative and uses reconfiguration for an extended failure in its conservative design. Zaxonlite already has voter replacement in ZDS 0008. The first release therefore uses $k = n_d$ and leaves $G(k)$ as the reviewed extension point.

19.6. Merkle Mountain Range as the history authority

History trees and MMRs provide logarithmic append-consistency and inclusion proofs. They are valuable if an offline or Byzantine verifier must prove that one root extends another. Zaxonlite's current fault model instead obtains a history anchor from authenticated non-Byzantine voters and constrains leaders with the trimmed Phase-1 rule. An MMR would add canonical peak handling, internal-node retention, proof formats, and non-constant worst-case merges without strengthening Paxos agreement. It is deferred to an auditability ZDS.

19.7. SQLite Online Backup for concurrent state transfer

SQLite's backup API creates a consistent snapshot and can allow concurrent writes. Incremental backup may restart or incorporate later source writes, however, and this design needs a reviewed exact mapping from the finished page image to one global slot. The v1 raw pinned copy has a simpler proof and preserves existing byte-exact page layout. A future implementation may adopt the backup API after it atomically captures the final source slot and proves subsequent exact-page replay compatibility.

19.8. `VACUUM INTO` for state transfer

Rejected. Vacuuming may repack and renumber pages. Zaxonlite replication applies later payloads by physical page number and final page count, so logical SQL equivalence is insufficient.

19.9. ARIES-style physiological redo as the consensus algorithm

ARIES provides excellent recovery principles—monotonic LSNs, write-ahead logging, checkpoints, and repeating history—but it does not replace distributed agreement. This design adopts those separation principles while Paxos remains the authority for chosen order.

19.10. Replace Paxos with Raft

Raft also requires snapshots or log compaction and a bounded implementation. Changing the consensus family does not remove the storage-layer problem and would discard the existing verified core. The required change is compatible with Multi-Paxos.

19.11. Matchmaker Paxos, Delos, or disaggregated log storage

Matchmaker Paxos improves reconfiguration; Delos composes virtual consensus logs; Aurora separates database compute from replicated storage. Each offers useful future directions, especially for elastic membership or remote archives. None is required to fix the local slot lifetime and routine snapshot tax. This record chooses the smallest architecture that establishes clean boundaries now.

20. Open Questions and Recommendations

Area	Shipped / Recommended default	Rationale & code mapping
Q1: Window & chunk profile	$W = 4096$ slots, $R = 256$ slots, pipelining credit $c = 4$, target occupancy $\rho = 0.75$.	Derived via Little's Law for $\lambda_p = 5 \cdot 10^4$ writes/s at "p99" commit latency $d_{99} =$ 15 ms ($W_{\min} = 1000 \to 4096$). In-core window memory is under 1.3 MiB. At 161 bytes/command, $R = 256$ forms compact ≈ 41 KB wire frames that fit within standard TCP burst MTU windows below the 64 MiB limit.
Q2: Retention horizon	Soft retention: 15 minutes or 20 GB of payload/journal bytes. Hard ceiling: 60 minutes or 64 GB. *v1 ships:* the 64 GiB hard ceiling over journal plus retained payload bytes is the shipped default (`journal_cap_bytes`), refusing writes with	Covers $X_{99.9} = 900$ s transient restarts (VM migrations, OS reboots). At $\lambda = \frac{5000}{\text{s}}$ and $c =$ 4.2 KB/slot, 15 minutes requires $4.5 \cdot 10^6$ slots (≈ 18.9 GB). The hard ceiling engages backpressure (`RecoveryRetentionExceeded`) on the admitting leader if a replica remains

Area	Shipped / Recommended default	Rationale & code mapping
	`RecoveryRetentionExceeded`; the soft horizon is the opt-in slot-based `retention_slots` (the time/byte form is deferred with quorum trim). The ceiling is a leader-side admission bound with a proven one-write reserve, so no admitted write lands above it; it is not a cluster-wide volume guarantee – consensus never refuses chosen entries, so a lagging follower installs what was chosen and fails loudly on its own write path if its volume fills first.	down; follower volumes are protected by the same cap only insofar as they track the leader's admitted growth.
Q3: Reflink & CoW support	Linux XFS (`FICLONE`), Btrfs (`BTRFS_IOC_CLONE`), macOS APFS (`clonefile(2)`), and Windows ReFS (`FSCTL_DUPLICATE_EXTENTS_TO_FILE`). Fall back to synchronized stream copy on NTFS and ext4.	Runtime probing via `probeReflinkSupport` (matching `durability.zig:probePathnameSemantics`). Durability requires an immediate `syncFile` on the clone followed by `syncDirectory` on the parent to ensure point-in-time barrier safety.
Q4: Segment size & sparse index	Segment capacity: 64 MiB (`64 * 1024 * 1024` bytes). Sparse-index stride: $k = 64$ slots.	64 MiB matches modern NVMe erase-block allocation, bounds directory entry counts (a 1 TB log is 16,384 files), avoids long rotation syncs, and matches the 64 MiB wire frame ceiling. A 64-slot stride requires under 4 KB of index per segment (one memory page) for sub-microsecond binary search.
Q5: Applied state anchor cadence	Periodic checkpoint every 30 seconds, or every 10,000 committed slots, or when uncheckpointed WAL reaches 64 MiB. *v1 ships all three triggers;* the 80%-soft-retention acceleration is deferred with the time/byte horizon it keys on.	Bounds the uncheckpointed recovery lag $E_i - A_i$ while holding SQLite WAL checkpoint and `APPLIED.0/1` barrier overhead to under 1% of write duty cycle on NVMe drives.
Q6: CI performance gate	Shipped contract: absolute Hodges–Lehmann bounds at $n = 64$ raw samples – $\leq 3\%$ for in-memory workloads, $\leq 5\%$ for durable ones, no fsync normalization (recorded runs pin the host). *Future, labelled for shared CI:* a $\leq 10\%$ non-inferiority margin normalized against a baseline `fsync` calibration loop, for when the gate runs on virtualized runners with noisy storage.	Pinned-host recording makes absolute bounds meaningful today. Virtual CI runners exhibit high storage variance, which is what the labelled future normalization exists for; the steady-state invariant of one barrier per transaction group holds under either policy.
Q7: Audit log retention	Strictly decouple operational consensus log retention from compliance audit history. Stream sealed `.zxj` segments asynchronously to an external archive sink (e.g., S3/cold storage).	Local disk retention serves only crash-recovery and catch-up (hours/days). Local consensus nodes must never stall cluster log trimming for multi-year audit compliance.
Q8: Audit proofs vs. anchors	Quorum-certified history hash chain (s, H_s) is authoritative and sufficient for crash-fault consensus. Defer RFC 9162 MMR / Merkle tree proofs to a dedicated auditability ZDS.	Lemma 3 proves unique prefix binding under collision-resistant SHA-256. A full Merkle tree adds $O(\log n)$ CPU and peak-merging overhead to the transaction hot path without strengthening Paxos agreement.
Q9: $G(k)$ quorum-trim trigger	Retain conservative all-data-replica trim ($k = n_d$) as the first-release contract; use ZDS 0008 voter replacement for permanently failed nodes. Implement $G(k)$ only for geo-distributed topologies with long-disconnected edge replicas.	ZDS 0008 voter replacement safely retires dead nodes without introducing the multi-tier failure-domain complexity of Theorem 6 ($k \geq f_{\text{state}} + 1$) and Theorem 7 ($k + q > N$).

20.1. Detailed Rationale for Open Question Resolutions

- **Q1: Workload and window profiling.** In `zaxonlite`, each in-core command is 161 bytes. With member acknowledgement bitsets and metadata, $b_{\text{cell}} \approx 300$ bytes. Setting $W = 4096$

requires only 1.2 MB of volatile RAM, well within any embedded or server budget. Setting recovery chunk $R = 256$ ensures that recovery frames (≈ 41 KB) can be streamed without exceeding packet buffers or the protocol's 64 MiB wire frame ceiling.

- **Q2: Sizing physical retention.** Sizing retention for a 15-minute outage quantile ($X_{99.9} = 900$ s) provides ample margin for node reboots, OS updates, and network partitions. At 5,000 writes/sec, this bounds local log storage to ≈ 20 GB, while the hard 64 GB ceiling prevents volume exhaustion by asserting write backpressure (`RecoveryRetentionExceeded`) rather than compromising safety.
- **Q3: Point-in-time reflink mechanics.** Reflink support avoids copying large SQLite databases during state transfer staging. Probing filesystem capabilities at startup allows `zaxonlite` to leverage `FICLONE` on Linux (XFS/Btrfs) and `clonefile` on macOS (APFS), falling back to synchronized sequential streaming on non-CoW filesystems (ext4, NTFS).
- **Q4: Segment and sparse-index geometry.** A 64 MiB segment size balances file handle count and allocation granularity on NVMe storage. A sparse index with stride $k = 64$ slots fits inside one 4 KB operating system page per segment, allowing binary search in L1/L2 cache and streaming disk recovery.
- **Q5: Applied state anchor frequency.** Running `APPLIED` checkpointing at 30-second intervals or 10,000 slots maintains sub-second restart recovery times while keeping WAL checkpoint latency below 1% of total transaction duty cycle.
- **Q6: Robust CI gating.** By testing the consensus core in-memory with strict 3% regression margins and durable storage at an absolute 5% bound on pinned recording hosts (barrier normalization reserved for shared CI runners), the gate stays sensitive to code regressions without failing on disk jitter.
- **Q7: Archival vs. Operational separation.** Offloading compliance archiving to asynchronous segment export ensures that local operational databases retain only the active recovery window.
- **Q8: Cryptographic proofs in scope.** Quorum-signed hash anchors (s, H_s) provide unambiguous history integrity (Lemma 3). Deferring full RFC 9162 Merkle trees preserves $O(1)$ append latency in the high-throughput write path.
- **Q9: Operational quorum trim.** Keeping $k = n_d$ simplifies the consensus and durability invariant in v1, relying on established voter replacement (ZDS 0008) for long-term node retirements.

21. Acceptance Criteria

- At least ten million decisions complete in one database and one unchanged configuration with no snapshot rollover and no slot reset.
- Consensus RSS remains bounded by configured windows within measured allocator tolerance.
- No two values are chosen for one slot in the formal model or deterministic simulation.
- A Phase-1 quorum containing only trimmed acceptors never causes a proposal at or below its greatest chosen prefix anchor.
- An accepted-only slot is never evicted; every reused chosen cell remains answerable from journal evidence or the durable trim anchor.
- A crash at every specified failpoint recovers the exact chosen SQLite prefix or refuses to vote.
- Cluster trim never advances past the minimum durable-state frontier of all current data replicas.
- Local deletion never advances past local durable state, the retention cutoff, or an active transfer lease.
- A witness alone never satisfies state recoverability.

- Segment and payload deletion cannot remove a retained reference.
- Membership change preserves monotonically increasing global slots.
- Routine segment rotation and trimming perform no full database copy or full database hash.
- A replica within retention uses range recovery; a replica behind trim uses a byte-exact pinned state image and suffix replay.
- Killing the original transfer sender does not strand the receiver: in v1 the conservative trim freeze retains every slot the receiver still needs, and any surviving data replica pins a fresh image and completes the send.
- Wire v9 and durable v2/v3 decoders reject every legacy format; no bridge or mixed-version path exists.
- Stable-leader core performance meets the reviewed regression gate, and long-run p99.9, maximum/median ratio, and periodicity checks contain no database-size-correlated rollover spike.
- ZDS 0004′s format contract and ZDS 0008′s membership contract receive explicit amendments when this record is accepted; they are not silently contradicted.

22. Implementation Notes and Deviations

The v1 implementation landed as specified, with the deviations and discoveries below. Each deviation keeps an invariant of this record intact while simplifying the mechanism that enforces it.

22.1. Segments are record-capped without payload manifests

Journal segments seal at a fixed record count (16,384) rather than a byte budget, and sealed trailers carry no per-segment payload-digest manifest. Payload garbage collection instead streams the retained journal to build the reachable set, and runs only when reclamation removed history or the chosen trim advanced — never on every pump. The trailer keeps the `max_promised` ballot rollup, which is the part correctness needs: without it, trimming a promise-bearing segment would let the acceptor promise backwards after replay.

22.2. No trim hysteresis

Trim proposals are not rate-limited by a hysteresis band. The conservative candidate only moves when a data replica publishes a new durable anchor, so the anchor cadence already bounds trim frequency; a separate band added a tunable without adding a property.

22.3. The first anchor publishes promptly

A node with no durable anchor recovers from genesis, so `maybeCreateStateAnchor` fires as soon as anything is applied, and the slot-interval cadence (10,000 slots) governs afterwards. This also makes trimming live: the conservative trim needs every data replica to have reported a nonzero durable frontier.

22.4. Conservative trim subsumes transfer leases in v1

The chosen `transfer_lease` and `lease_complete` entries and the TRIM-file lease table exist as specified, but no v1 path proposes them. The v1 trim is the minimum durable frontier over all data replicas, and that minimum is itself the freeze this record wanted from leases: a joining replacement reports a zero frontier until its first anchor, and a lagging replica pins the candidate at its own last anchor, so history a transfer target needs cannot be reclaimed while it catches up. Killing a transfer sender strands nothing: the receiver re-requests from any data replica, and the frozen minimum keeps the base retained. Leases become necessary only with the deferred quorum-trim variant, and the plumbing for them is in place.

22.5. The checkpoint proof is removed, not revised

The planned proof v3 artifact is not produced. The stop-sign proof existed to let a receiver verify a snapshot generation attributed to a sealed configuration; the anchor-pinned transfer has a live cluster to ask instead. The receiver confirms `(anchor_slot, history_hash)` with a read quorum of the current voters — each voter vouches from a small ring of recent per-slot history hashes, its own durable anchor, or the chosen trim anchor — and then verifies the image digest from the transfer manifest. `checkpoint_proof.zig` and the `ZXP2` format are deleted with the format cut.

22.6. Raw image copy is a buffered stream

The transfer sender pins the anchored image with an ordinary buffered copy; the reflink probe (`FICLONE`, `clonefile`) from Q3 is not implemented. The copy is process-private, priced O(database), and happens once per transfer, which is already the rare recovery path.

22.7. Lifetime-journal replay folds across ballot lines

A discovery, not a design choice: one journal now spans configuration changes, and elections in a successor configuration legitimately begin at round one, below ballots promised in the sealed line. Strict replay declared that history corrupt. `DurableState.replayFold` folds the lifetime journal instead — promises fold to the maximum so the acceptor still never promises backwards, accepts whose cells were reused by newer slots are dead history and are skipped, and commits and trim anchors keep their strict rules. The voter-replacement suite found this by killing and restarting a survivor after the handover.

22.8. Stop signs are configuration-scoped observations

A replayed or journal-served stop entry naming a configuration the node already runs is completed history: it neither re-arms the membership handover nor seals the log. Without this, a restarted survivor looped forever trying to complete a handover that had already completed.

22.9. The closed prefix is enforced at the acceptor

An external audit found that `onAccept` and the commit-install path guarded only the memory floor, while `GlobalTrim.tla` guards `Vote` and `Learn` with `slot > anchor[n]`. The certified trim anchor can run ahead of the memory floor until the host consumes, and in that gap a late accept could re-tag a released cell with an accepted-only occupant that eviction can never clear. Both handlers now drop messages at or below the acceptor's own trim anchor, restoring the model's closed-prefix rule; a protocol test pins the floor-behind-anchor shape directly.

22.10. In-place transfer preserves acceptor obligations

The same audit found that installing a transferred image restored the core from an empty durable state, discarding the receiver's promise and its votes above the transfer anchor – state the lifetime journal still held, so a restart would resurrect what the live node had forgotten. The model never covered this: `InstallJoiner` requires an empty consensus state. Installation now carries the current durable state into the restored core – the promise and every vote above the anchor survive; votes at or below it are discharged by the anchor and the reported `chosen_through` fence – so a transfer can never let an older ballot win a slot the receiver already helped choose. Restores scrub accepted-only cells at or below the restore floor for the same reason they may be dropped: the slot is inside a certified chosen prefix, and a wedged accepted-only occupant would block its cell forever. The in-place install is modeled as `InstallVoter`, carrying exactly this preservation rule; its validation pair is queued on the verification machine behind the exhaustive base sweep.

22.11. Promise ranges bound reported votes, not the interval frame

This record's PromiseV2 shape asked for `accepted_range.lo` to start above the responder's trim anchor. As implemented and as modeled, the invariant attaches to the votes: `onPrepare` answers the leader's requested chunk interval verbatim but reports no vote at or below its anchor (the anchor itself travels in the reply, and `PromiseRangeStartsAboveAnchor` checks exactly this in the model), and the leader's F and K fences make the interval framing inert. The interval is transport framing for per-chunk counting under reordering; canonicalizing it to the anchor would add a second copy of a fact the fences already own. This note amends the wire-shape clause to the reported-vote invariant the model verifies.

22.12. Damaged-voter repair is replacement in v1

The recovery ladder's step 5 promised an in-place install for an established voter whose anchor no longer reaches retained history or whose image is lost. v1 fails closed there instead: `Node.open` refuses with `StateUnavailable` and the operator hint names the replacement path. The reduction is deliberate. An imageless established voter cannot restart into a truthful consensus state: its journal may hold votes far above any floor it can claim (the window invariant forbids holding them over floor zero), and any claimed `chosen_through` above what it can apply would overstate the K fence. Serving either would trade a loud, lossless failure for a quiet unsound one. Replacement loses nothing – the conservative trim freezes on the damaged voter's stale report, so every slot the successor needs stays retained – and the preserving in-place install remains implemented, unit-tested, and modeled (`InstallVoter`) for the quorum-trim variant where an established voter can genuinely fall behind retention.

22.13. The in-place voter transfer path is dormant under conservative trim

Same-configuration transfer preserves the promise and votes above the installed anchor, with the rule unit-tested and modeled as `InstallVoter`. No end-to-end scenario drives an established voter through it, and that is a consequence of v1 policy rather than missing test work: the conservative trim is the minimum durable frontier over every current data replica and freezes whenever a report goes stale, so a data voter that exists can never fall behind physical retention – its own frontier holds the trim. The path stays implemented and preserving as defense in depth, and becomes reachable only with the deferred quorum-trim variant, which is when an end-to-end scenario for it becomes constructible.

22.14. Recovery authorities are reconciled at open

Open now cross-checks the durable trim authorities instead of trusting each independently: the TRIM file and the journal's replayed anchor must agree (a file one record ahead re-installs, since it is written first; twins under one id and a journal ahead of the file fail closed), manifest segment ranges must chain contiguously with the oldest retained segment allowed to straddle the anchor (reclamation removes only whole segments, so the straddle is the common cluster shape – requiring an exact anchor boundary made any lagging delete floor unopenable), sealed segments must chain their predecessor digests and match the manifest's ranges, a resumed active segment must name this database and chain onto the last sealed digest, and an anchored image whose retained journal no longer reaches it fails closed instead of serving stale state. Suffix replay applies commits in one streaming pass through a window-sized reorder ring (the core only journals in-window commits, so the reorder distance is bounded), and folds each entry's history leaf under the configuration that chose it by advancing a cursor across replayed stop entries – restoring the byte-exact hash the live handover produces. The voter-replacement suite now ends by driving a

trim round after the handover and the survivor restarts: every data voter anchors, and the chosen trim slot must advance on all of them – exactly the observable a diverged hash freezes.

22.15. Transfer crash coverage is a boundary ladder, not a fuzzed matrix

The fifteen-case single-node crash matrix covers the write, anchor, trim, reclamation, and payload-GC ladders. The transfer path carries seven deterministic failpoints (mid-copy and post-copy on the sender's pin; a received chunk, the staged image, either side of the install rename, the published anchor, and the in-memory core resume on the receiver), and the end-to-end scenario below crashes every stateless-phase boundary plus the sender. What it deliberately does not do: crash during the post-install suffix replay (that is the ordinary restart path the storage matrix and cluster suites already cover), crash after the core resume as a separate rung (`after_transfer_resume` differs from `after_transfer_anchor` by in-memory state only, so one join can crash at exactly one of them), or fuzz interleaved multi-process schedules. The transfer's crash safety still rests on its verify-then-install shape – staged private file, digest match, quorum-vouched anchor binding, disposable sender pin – and the ladder proves each boundary of that shape once. Every rung waits under a deadline and proves the process died at exactly the armed failpoint by reading the crash line from its own log; the sender rung arms node 1 through its process environment on a restart, the mechanism that cannot be lost to timing. Building the ladder also exposed a peer-connection defect: the per-peer inbound limit rejected the newest connection while a crashed predecessor's not-yet-reaped entries held the slots, so a rapid crash-restart cycle churned reconnects for minutes. Admission is now newest-wins – at the limit the oldest same-credential connection is shut down and reaped, and a fresh reconnect always seats.

22.16. The beyond-retention transfer runs end to end under a crash ladder

Forcing the transfer requires filling and reclaiming whole journal segments cluster-wide, so `zaxon serve` gained a test-only `--segment-records` flag (failpoints-gated, never above the production capacity) that shrinks rotation to test scale. The `test-transfer-cluster` scenario drives three voters past several rotations, certifies a trim, reclaims below it, replaces a voter, and forces the replacement through the anchor-pinned transfer: the receiver is crashed at each stateless-phase failpoint in sequence (a received chunk, the staged image, before the install rename, after the rename, and after the anchor publish), the sender is killed mid-copy while pinning its image and another peer completes the send, and the final clean start must converge from the installed anchor, match its peers' row multiset and application chain hash, accept writes, and survive its own restart. The whole join runs on an otherwise idle cluster, so the leaderless recovery probes below are themselves under test.

Building the scenario found two liveness gaps, neither a safety one. First, a stateless joining voter could win the election, and catch-up and snapshot escalation both ran against the leader, so leadership starved its own recovery forever; a joining data voter with nothing applied now withholds campaigning (it still votes) until state applies. Second, a held joiner could not learn the leader on an idle cluster – heartbeats above its zero promise are ignored until a first accepted vote arrives, and an idle cluster sends no accepts – so recovery depended on an application write. The joiner now probes the decided registry's peers directly on a rotating cursor, pairing range catch-up with a snapshot request; the sender side arbitrates which applies, and no application write is required.

22.17. The ten-million-decision run is banked

The full acceptance run completed on 27 August 2026: the long-run gate (`test-longrun`) drove 10,000,000 fsync-backed writes on one database in one unchanged configuration on a 32-core

AMD Ryzen 9 5950X (tmpfs-hosted data directory), finishing in 22,426 seconds with exit 0. The gate's own verdict covers the criteria: hundreds of segment rotations with physical reclamation below the certified chosen trim, retention bounded to the suffix plus the active segment throughout, configuration id 1 from the first write to the last, and an anchored restart that recovered exactly ten million rows with the correct checksum, a clean integrity report, and a fresh write accepted. The supporting evidence remains alongside it: the core moving-window benchmark's 262,144 decisions across 256 window wraps with flat batch percentiles, and the nightly 100,000-write CI gate.

22.18. Statistical benchmark gate

`zig build bench-gate` compares two recorded result files with a Hodges-Lehmann shift estimate and a deterministic bootstrap 95% confidence interval (the plain percentile bootstrap this record's verification plan specified; BCa is deliberately not required), failing at +3% of the baseline median for in-memory workloads and +5% for durable ones. Enforcement requires both runs to carry at least 64 raw `samples_ns_per_value` observations, the bar the record's power derivation set; the in-memory matrix, the moving workload, and both durable workloads now emit them (durable groups are individually timed, with headline aggregates still taken from whole-run clocks), so durable rows gate. Below the bar, or with summary quantiles only, a pair is report-only. A fixture field carried by only one record makes the pair incomparable and skipped; mismatched host or toolchain metadata downgrades a pair to report-only. Within-run observations share one process lifetime and are autocorrelated, so the interval understates variance; repeated same-fixture runs in separate files remain the gold standard, and the tool says so. Periodicity runs on the recorded `batch_ns_series`: the moving workload's full per-batch series covers the window-wrap lag, and the zaxonlite write benchmark drives real durable anchors with two series – inter-completion time, whose correlation at the forced anchor cadence is a positive control proving the instrument sees anchor cost (it fails loudly if anchors become invisible), and bare per-exec latency, whose anchor-lag correlation is the gate property: whether anchoring degrades the writes around it, measured flat. Existing archived results predate the sample fields; the next recorded run banks the first enforceable pair.

23. Amendments to Prior Records

This section follows the amendment pattern of ZDS 0006. The prior records are not edited; where a clause below and this record disagree, this record wins.

23.1. Amendments to ZDS 0004

The bounded-epoch format contract is replaced by the global-slot contract of this record. The wire protocol moves from version 8 to version 9 with exact-major acceptance unchanged; slots are 64-bit everywhere. The per-configuration journal (`paxos-<configuration>.log`) is replaced by the lifetime `consensus/` directory: first-slot-named `.zxj` segments (`ZXS2` headers, sealed `ZXT2` trailers with the `max_promised` rollup), the `ZXM2` manifest generations, the alternating `APPLIED.0`/`APPLIED.1` durable state anchors, and the `ZXTR` trim state. Snapshot generations, the `CURRENT` pointer, and the `ZXP2` checkpoint proof are removed. Stop metadata moves from `zx2` to `zx3` and carries only the next registry digest and the replacement seed. Legacy artifacts fail closed at open; there is no bridge, permitted because deployment has not launched and there are no shipped bytes to migrate.

23.2. Amendments to ZDS 0008

The decided one-for-one replacement operation, its authorization, the decided registry, the allocation fence, and the idempotent operation ring all stand. Its integration with the epoch

rollover is superseded: the stop sign no longer names a snapshot generation or manifest digest, survivors complete the handover in place and continue the same global slot line (delivered and floor at the stop slot, inherited trim anchor) instead of initializing a fresh epoch, and the journal is not replaced at the boundary. The joining replacement fetches and verifies the decided registry against its enrollment descriptor, then catches up through ordinary range recovery from the retained journal; only a gap beyond retention uses the anchor-pinned state transfer of this record. The read-quorum attestation of ZDS 0008's install path is carried forward as the history probe over `(anchor_slot, history_hash)` rather than a proof digest over a sealed stop sign.

24. References

- Leslie Lamport, "The Part-Time Parliament" and "Paxos Made Simple" — the quorum and agreement foundation: `https://lamport.azurewebsites.net/pubs/lamport-paxos.pdf`
- Robbert van Renesse and Deniz Altinbuken, "Paxos Made Moderately Complex" — operational Multi-Paxos structure: `https://www.cs.cornell.edu/courses/cs7412/2011sp/paxos.pdf`
- Zhiying Liang, Vahab Jabrayilov, Aleksey Charapko, and Abutalib Aghayev, "MultiPaxos Made Complete" — lightweight trimming without repeated snapshots: `https://arxiv.org/abs/2405.11183`
- Zhiying Liang et al., "HoliPaxos: Towards More Predictable Performance in State Machine Replication" — Global Last Executed, reconfiguration for a stuck frontier, transfer freeze, and quorum-trim comparison: `https://www.vldb.org/pvldb/vol18/p2505-charapko.pdf`
- Heidi Howard, Dahlia Malkhi, and Alexander Spiegelman, "Flexible Paxos: Quorum Intersection Revisited" — only cross-phase consensus-quorum intersection is required: `https://doi.org/10.4230/LIPIcs.OPODIS.2016.25`
- Leslie Lamport and Mike Massa, "Cheap Paxos" — main versus auxiliary processors and the distinction between voting and state-bearing roles: `https://www.microsoft.com/en-us/research/publication/cheap-paxos/`
- Scott Crosby and Dan Wallach, "Efficient Data Structures for Tamper-Evident Logging," plus RFC 9162 — history-tree inclusion and consistency proofs, explicitly deferred from v1: `https://www.usenix.org/conference/usenixsecurity09/technical-sessions/presentation/efficient-data-structures-tamper-evident`, `https://www.rfc-editor.org/rfc/rfc9162.html`
- Tushar Chandra, Robert Griesemer, and Joshua Redstone, "Paxos Made Live" — the gap between protocol and production system: `https://research.google/pubs/paxos-made-live-an-engineering-perspective/`
- C. Mohan et al., "ARIES: A Transaction Recovery Method Supporting Fine- Granularity Locking and Partial Rollbacks Using Write-Ahead Logging" — LSN, repeating-history, and checkpoint separation: `https://www.cs.cmu.edu/~15849g/readings/mohan92.pdf`
- Diego Ongaro, "Consensus: Bridging Theory and Practice" — log compaction, snapshots, membership, and implementation proof obligations: `https://github.com/ongardie/dissertation/raw/master/stanford.pdf`
- James C. Corbett et al., "Spanner: Google's Globally-Distributed Database" — replicated state machines composed with storage and reconfiguration: `https://research.google/pubs/spanner-googles-globally-distributed-database/`
- Mostafa Elhemali et al., "Amazon DynamoDB: A Scalable, Predictably Performant, and Fully Managed NoSQL Database Service" — production replication, storage nodes, repair, and operational isolation: `https://www.usenix.org/conference/atc22/presentation/elhemali`
- Alexandre Verbitski et al., "Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases" — separation of database and replicated

storage recovery: https://www.amazon.science/publications/amazon-aurora-design-considerations-for-high-throughput-cloud-native-relational-databases

- Mahesh Balakrishnan et al., "Virtual Consensus in Delos" — composable log abstraction and reconfiguration: https://www.usenix.org/conference/osdi20/presentation/balakrishnan
- Michael Whittaker et al., "Matchmaker Paxos" — reconfiguration without fixed configuration sequencing assumptions: https://arxiv.org/abs/2007.09468
- SQLite, "Write-Ahead Logging," "Online Backup API," and "VACUUM INTO" — local page state, concurrent backup semantics, and why vacuumed logical copies are not the v1 physical replay base: https://www.sqlite.org/wal.html, https://www.sqlite.org/backup.html, https://www.sqlite.org/lang_vacuum.html#vacuuminto
- docs/zds/records/0001-zds-process.typ — lifecycle and authoring rules
- docs/zds/records/0002-zaxonlite-product-plan.typ — product architecture
- docs/zds/records/0004-zaxonlite-format.typ — current format contract
- docs/zds/records/0008-zaxonlite-voter-replacement.typ — membership and checkpoint transition contract
- src/protocol.zig, src/replicated_log.zig, src/learner.zig — current bounded consensus representation
- zaxonlite/src/node.zig, journal.zig, wal.zig, payload_store.zig, and checkpoint_proof.zig — current rollover, persistence, and recovery boundary