Zaxon Working Group
Request for Discussion and Implementation Record

ZDS 0008
Engineering Discussion

zaxonlite Dynamic Voter Replacement

STATE
committed

CREATED
2026-07-27

LAST UPDATED
2026-07-28

INTENDED STATUS
Implemented for 0.1.2

AUTHORS
paxos-zig project

DISCUSSION

A bounded membership change for replacing one data voter without rebuilding the database

LABELS

zaxonlite, membership, operations, reconfiguration

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

zaxonlite builds its server registry from bootstrap arguments. That registry then stays fixed for the life of the database.

The underlying `ReplicatedLog` can seal one configuration and start the next. The zaxonlite host does not yet persist or recover a changed voter registry. Its transport and authentication paths also assume fixed membership.

This record defines one bounded operation. It replaces one data voter with one fresh data voter. The voter count and database identity stay the same.

The old configuration chooses the transition. The chosen value binds a checkpoint and the next registry. Surviving servers activate that registry at a controlled epoch boundary. The new voter must install and verify the state before it can vote.

2. Introduction

Replacing failed hardware should not create a new logical database. Operators should not copy membership files between servers. Consensus must remain the only source of membership truth.

The safe authority chain is:

1. Current voters choose a stop sign that names the next voter IDs.
2. Stop metadata binds the checkpoint and canonical next registry.
3. A read quorum from the old configuration confirms the sealed proof.
4. Surviving servers persist and activate the decided registry.
5. The replacement installs the bound checkpoint before it can vote.

The first implementation is intentionally narrow. Consensus chooses the transition online. Server transport is rebuilt at the epoch boundary. A brief write pause is acceptable. The result should be small and testable.

3. Terminology and Scope

- **sealed configuration**: the old configuration whose stop sign has been chosen
- **next configuration**: the configuration named by that stop sign
- **data voter**: a voting, campaigning node that stores the Paxos log and materializes database state
- **replacement**: one old data voter ID removed and one fresh data voter ID added
- **decided registry**: the canonical, durable mapping from configuration ID to node IDs, roles, and transport endpoints
- **stop proof**: evidence that the sealed configuration chose the transition and the checkpoint from which the next configuration starts
- **node-ID allocation fence**: the highest node ID ever admitted to a decided configuration
- **fresh ID**: a node ID greater than the allocation fence
- **operation ring**: the 32 most recent replacement operation records
- **activation**: the point after durable installation when a process may send or count votes in the next configuration

The first implementation supports network-hosted zaxonlite clusters. These clusters must use mutual authentication. The operation replaces exactly one data voter. The voter count and all surviving roles stay the same.

The cluster must contain at least three voters. The sealed configuration's read quorum must be satisfiable by the survivors alone, because the removed voter may be gone and the replacement never counts. With two voters that quorum is unreachable, so the request is rejected during validation.

4. Problem Statement

The protocol already carries a stop sign. It contains a next configuration ID, member IDs, and bounded metadata. The missing work is in the zaxonlite host:

- `server.zig` builds peers and senders from startup options. It reports membership as static.
- `node.zig` can complete a rollover. It does not make a changed server registry authoritative.
- Snapshot installation and checkpoint-proof validation use the local static member array, not the decided stop sign.
- The checkpoint proof contains one member list. Replacement needs both the old and next voter sets.

- Enrollment authorizes only the startup registry.
- Bootstrap membership helps derive the first database ID. Replacement must not derive a new database ID.

Changing only the `ReplicatedLog` member slice is not enough. Consensus could use one registry while transport uses another. Authentication, snapshots, and restart could also disagree.

This split is already present in the code. Rollover initializes the next consensus membership from the chosen stop sign, but the host keeps its startup member array. After a changed-member rollover, snapshot installation and proof validation would read the stale array. A node would reject its own checkpoint proof. The decided registry closes that gap.

5. Goals and Non-Goals

5.1. Goals

- Replace one data voter with one fresh data voter through a privileged, idempotent operation.
- Keep the database ID immutable across replacement.
- Make the consensus-decided registry the single authority after bootstrap.
- Bind the checkpoint, sealed configuration, next configuration, and next registry to one proof.
- Require old-configuration quorum evidence for the stop decision.
- Prevent the new voter from voting before verified state installation.
- Permanently reject the removed node ID after activation.
- Keep node-ID retirement and operation history in fixed memory.
- Recover deterministically from a crash at every durable transition.
- Reuse the existing stop-sign, checkpoint, manifest, and epoch rollover mechanisms.

5.2. Non-Goals

- No change in voter count.
- No replacement of a witness or change of node role.
- No multi-node membership operation.
- No automatic replacement based on failure detection.
- No unauthenticated remote administration.
- No second membership algorithm beside the existing stop-sign transition.

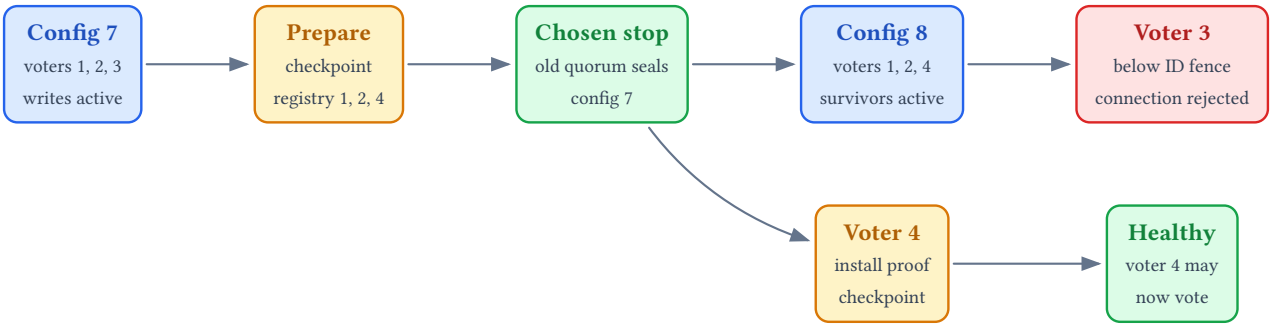
6. Invariants

The implementation must preserve these rules:

1. A command is never chosen after the chosen stop slot in the sealed configuration.
2. The next configuration is exactly the member ID set carried by the chosen stop sign.
3. The endpoint and role registry activated by a server has the digest bound into that stop sign.
4. A new voter cannot emit a Paxos message until the bound checkpoint and registry are durable.
5. Only IDs in the current decided registry can authenticate or vote.
6. The database ID never changes.
7. Recovery selects the old registry or the fully installed new registry. It never mixes the two.
8. The node-ID allocation fence never decreases or wraps.
9. The operation-ID high-water mark never decreases or wraps.
10. A retained operation ID returns the same result when its arguments match.
11. An operation ID older than the retained ring is rejected. It cannot start a new transition.

7. Design Overview

Assume the active data voters are `1`, `2`, and `3`. The operator replaces `3` with fresh ID `4`.



The transition has two independent gates.

- **decision gate**: only the sealed configuration can choose the stop sign
- **activation gate**: a node needs the decided registry and installed checkpoint before it can join the next configuration

The surviving voters can form the next quorum first. Service can then resume while the replacement transfers state. Configuration `8` has reduced fault tolerance until voter `4` activates.

8. Detailed Design

8.1. Replacement request

The administrative request contains:

```
operation_id: u64
expected_configuration_id
old_node_id
new_node_id
new_endpoint
role = data_voter
```

The active leader validates that:

- mutual TLS identifies a principal authorized for membership operations;
- the expected configuration is still active;
- the old ID is an active data voter;
- the new ID is greater than `highest_allocated_node_id`;
- the new endpoint is well formed and not assigned to another active node;
- old and new roles are both `data_voter`;
- replacing the old ID yields the same member count and a valid quorum;
- the cluster contains at least three voters;
- the new endpoint fits the bounded stop-metadata seed;
- no different replacement is already pending.

Operation IDs are monotonically increasing `u64` values. The server persists the record before it proposes the stop sign. A retained ID with the same arguments returns the recorded phase. The same ID with different arguments returns a conflict.

8.2. Canonical decided registry

A new module, `zaxonlite/src/registry.zig`, owns the decided registry. The existing `configuration.zig` keeps its current job, loading config files and secrets; it gains only the admin allow-list described under security.

Bootstrap options create configuration 1. The server then persists that registry. From that point, the persisted registry is authoritative. Startup flags cannot override it. A conflicting flag is a startup error, exactly as the identity file already rejects a changed node ID or role.

Each registry record contains:

```
database_id
configuration_id
predecessor_configuration_id
highest_allocated_node_id
nodes[] = { node_id, role, endpoint }
operation_records[32]
```

The encoding is canonical. Nodes are sorted by numeric ID. The format fixes integer widths and string lengths. An optional field has only one valid encoding.

The registry digest is SHA-256 over the complete canonical bytes.

The database ID is copied from the existing durable registry. It is never derived from the replacement member set. The bootstrap derivation runs only when no registry file exists. A present but unreadable registry stops the server; it never falls back to derivation.

The registry lives beside the existing `identity` and `CURRENT` files:

```
registries/<configuration id, 16 hex>   canonical registry blob
REGISTRY                                16-hex pointer to the active blob
PENDING-OP                              the one in-flight replacement request
```

The blob directory is deliberately not named `registry`: common macOS and Windows filesystems are case-insensitive, so that name would collide with the `REGISTRY` pointer file.

All three use the node's existing atomic write primitive: write a temporary file, sync it, rename it into place, sync the pathname transition. The `REGISTRY` pointer has the same shape and the same strict length check as `CURRENT`. It is a separate file because the two pointers advance at different points in the rollover order. Superseded blobs are garbage collected with the same retention discipline as old snapshots.

8.3. Bounded identity and operation history

`highest_allocated_node_id` is a monotonic `u32` allocation fence. Bootstrap sets it to the largest initial node ID. A replacement ID must be greater than the fence.

The fence advances when the stop sign chooses the next registry. It never decreases. It must not wrap. A cluster at `maxInt(u32)` returns `NodeIdExhausted` and refuses another replacement.

Peer admission still checks the exact current registry. A removed ID is not in that registry, so it cannot connect or vote. The fence prevents that ID from being assigned again. This gives lifetime node-ID retirement with one four-byte value.

The pending request reserves its proposed node ID. Only one request may be pending. The reservation is released if the request ends before the stop is chosen. Enrollment credentials are not issued before choice.

Operation history is part of the canonical decided registry. It uses a fixed circular ring of 32 immutable records. Each record contains:

```
operation_id: u64
expected_configuration_id
old_node_id
new_node_id
request_digest
result_configuration_id
```

The candidate registry appends the pending request record. The record becomes authoritative only if the stop sign chooses that registry. Every survivor and replacement therefore receives the same ring.

A new operation ID must be greater than the newest decided record. A matching retained ID is an idempotent retry. A smaller ID that is no longer in the ring returns `OperationHistoryExpired`.

The operation ID must not wrap. A ring whose newest ID is `maxInt(u64)` returns `OperationIdExhausted` for a new request.

A new decided record overwrites only the oldest record. The newest record therefore acts as the operation-ID high-water mark. Memory stays fixed. A delayed request cannot become a new operation after its record expires.

The ring records immutable outcomes, not live transfer progress. `phase` and `installation_state` report current progress outside the registry digest.

8.4. Checkpoint and stop metadata

The leader first creates a durable checkpoint. It covers the applied command prefix through slot `S`. The leader writes the candidate registry beside the checkpoint. It does not activate that registry.

The leader then proposes the stop sign in slot `S + 1`. The sealed epoch accepts no later command proposal once reconfiguration is pending.

The stop sign carries:

- the next configuration ID;
- the exact next member ID array;
- bounded versioned metadata containing the checkpoint name, manifest digest, and next-registry digest.

The metadata tag moves from `zx1` to `zx2`, and the host raises its stop-metadata capacity from 256 to 512 bytes. The registry digest is now part of the safety boundary. For a replacement, `zx2` also carries a bounded reconstruction seed: the operation ID, the old node ID, the new node ID, and the new endpoint.

The seed makes the next registry a pure function of the current registry and the chosen stop sign. Every survivor rebuilds the candidate registry locally, verifies its digest against the stop metadata, and never needs the network to activate. Only the replacement node, which holds no predecessor registry, fetches the blob from a survivor and verifies the same digest. The endpoint length cap at request validation keeps the seed inside the metadata bound.

The decoder rejects an unknown version. It also rejects duplicate IDs and unsorted canonical lists. The registry ID set must exactly match the next member array in the stop sign.

8.5. Proof version 2

The checkpoint proof advances to version 2 and binds both sides of the transition:

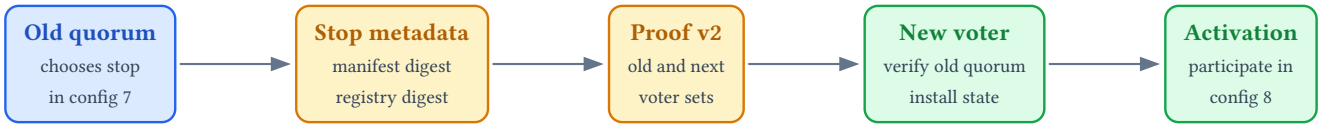
```
database_id
sealed_configuration_id
sealed_voter_ids[]
next_configuration_id
next_voter_ids[]
stop_slot
applied_slot
checkpoint_chain
manifest_digest
next_registry_digest
stop_metadata
```

Endpoints stay in the canonical registry. The registry digest commits to them, so the proof does not repeat them. The proof digest covers every field shown above.

The proof magic moves from `ZXP1` to `ZXP2`. Both voter counts widen to sixteen bits, and the encoded-size bound grows from 512 to 768 bytes. The rule that the next configuration ID equals the sealed ID plus one is kept. The wire protocol moves from version 6 to version 8 in the same release, covering the new proof, the registry fetch, and the replacement request. Exact-major acceptance is unchanged, and there is no dual-version bridge; the product has not launched.

Proof confirmation comes from a read quorum of the **sealed** voter set. The receiver authenticates each confirmer. It counts each sealed voter ID once. It never counts the proposed voter.

This proof answers one question. It shows that config `7` chose the stop. It does not show that the new voter is ready for config `8`.



8.6. Activation boundary

After the stop is chosen, each surviving server prepares one durable generation. It contains:

1. the new registry file;
2. the next configuration's initial checkpoint state;
3. the proof and operation ring.

An atomic current-generation pointer selects the prepared generation.

The primary implementation performs an in-process rebuild in `server.zig`. The server keeps client TCP connections open. The `Node` itself completes the rollover in place, exactly as it does for a same-member checkpoint today. What the server rebuilds is everything around it: the member table, the peer senders, and the peer admission rules, all derived from the decided registry.

The client connection layer must not retain a pointer to the old member table or sender set. Each request resolves the current in-memory generation before dispatch. An in-flight read either completes before quiescence or retries against the new generation.

This creates a bounded write pause. Existing reads may continue only when their normal linearizability checks succeed. Replacement adds no read bypass.

The activation order is:

```
stop chosen
  -> checkpoint and registry verified
  -> old Paxos work and peer dispatch quiesced
  -> in-flight requests completed or marked for retry
  -> next member table and peer senders built off to the side
  -> next registry and REGISTRY pointer durable
  -> active in-memory generation swapped
  -> removed peer connections closed
  -> Paxos traffic enabled
```

No process sends messages for the next configuration through the old sender set.

The durable `REGISTRY` pointer is the recovery authority for membership. It is written after the snapshot pointer and before the identity file, extending the rollover order the node already uses. A crash before its update restarts the old configuration. A crash after its update restarts the new configuration.

The in-process result must equal a clean restart from the same durable files. Tests compare configuration ID, registry digest, proof digest, journal state, and database state. If the in-process rebuild fails after the pointer update, the process stops. Restart then completes the new configuration. It never falls back to the old registry.

8.7. Replacement enrollment and catch-up

A proposal does not authorize the replacement to join. Enrollment opens only after the stop is chosen. An active survivor must also have the decided registry on durable storage.

The enrollment response binds:

- database ID;
- new node ID and role;
- next configuration ID;
- next-registry digest;
- the cluster certificate chain.

The replacement fetches proof v2, the registry, the manifest, the snapshot, and the required tail. The registry travels over a new fetch message pair and is checked against the digest bound in the enrollment response. The replacement verifies every digest and the sealed quorum. It then installs the state durably and starts in the next configuration.

Its certificate identity must match the decided node ID.

Receiving a connection from a next-config voter does not make it ready. Readiness is local durable state, not a transport handshake.

After installation and transport activation, the replacement sends an `installation_ready` frame. The frame carries the configuration ID and registry digest. A survivor accepts it only from the decided replacement. Connection state is still used for reachability, but never as installation evidence.

8.8. Removed voter behavior

The new registry removes the old ID and advances the node-ID fence. Current servers reject its enrollment, peer handshake, and Paxos messages. They reject the ID even when it presents an old valid certificate.

Certificate revocation may follow. Correctness does not wait for revocation to reach every server.

If the removed server returns, its old journal contains the chosen stop. It cannot append commands after that slot. The allocation fence prevents reuse of its node ID.

8.9. Status and operator interface

The server exposes two methods. Replacement is privileged. Status is read-only. The CLI presents them as:

```
zaxon replace-voter \
  --operation <u64> \
  --expected-config <id> \
  --old-node <id> \
  --new-node <id>@<host>:<port>

zaxon membership status
```

JSON and CLI status expose these fields separately:

- database and active configuration IDs;
- decided voter IDs, roles, and endpoints;
- registry digest and source `decided`;
- operation ID and phase, if a replacement is pending;
- `quorum_available`;
- `installation_state`.

For example:

```
{
  "phase": "active-degraded",
  "quorum_available": true,
  "installation_state": "transferring"
}
```

`quorum_available` is true when recent authenticated peers can satisfy both the configured read quorum and write quorum. The count includes the local voter. This field is an operational observation. It does not authorize a membership decision. It also does not promise that this server is the leader.

`installation_state` describes the replacement voter. Its values are `not-applicable`, `not-started`, `transferring`, `verifying`, `installed`, `active`, and `failed`.

`installed` means the verified state is durable. `active` means the voter may send and receive Paxos traffic. `failed` includes a stable error reason in the status response.

`phase` remains a high-level lifecycle summary. `active-degraded` means the next configuration is active but the replacement is not yet an active voter. It does not imply that a quorum is currently available. Operators read `quorum_available` for that fact.

The leader handles mutation. A follower returns a structured leader hint.

8.10. Durable operation phases

Phase	Meaning and retry behavior
`prepared`	The request is valid. A checkpoint and candidate registry may exist. Membership has not changed. The operation may be cancelled or retried.
`proposed`	The stop sign is in Paxos. Query status by operation ID. A timeout does not mean failure.
`chosen`	The old configuration is sealed. The operation cannot be cancelled or replaced by a different target.
`activating`	Survivors durably install the registry and rebuild the server boundary. A crash resumes this phase from the chosen proof.
`active-degraded`	The next configuration is active, but the replacement is not yet active. See `quorum_available` and `installation_state` for current health.
`complete`	The replacement has installed state and is participating as a voter.

The coordinator writes `prepared` before building the checkpoint. It submits and durably consumes the Paxos proposal before writing `proposed`. On restart, a durable accepted stop promotes a remaining `prepared` record to `proposed`. Without that durable evidence, the exact request remains safe to retry.

9. Failure and Recovery

9.1. Before the stop is chosen

The old configuration remains authoritative if preparation fails. This includes checkpoint failure, an invalid target, a lost leader, or a rejected proposal.

Recovery may remove temporary files after it proves that no matching stop was chosen. The caller may retry the same operation ID and arguments. A prepared request enters the decided ring only when its stop sign is chosen. A different request must use a higher operation ID.

9.2. Stop chosen, response lost

The caller queries status by operation ID. Any old-config quorum can recover the chosen stop through normal Paxos phase one. The operation then proceeds to activation. It cannot roll back.

9.3. Replacement unavailable

The phase becomes `active-degraded` after the next configuration activates. Service resumes only when `quorum_available` is true. The absent replacement may join later with the same decided proof.

9.4. Crash during activation

Registry and snapshot installation use temporary files. Each file is validated and synced. An atomic generation pointer selects the active state.

Recovery ignores a partial generation that the pointer does not reference. Once the pointer selects the next registry, startup must use it. Recovery then continues in the next configuration.

The same rule applies during the in-process swap. A crash before the durable pointer update opens the old registry. A crash after the update opens the new registry. Volatile swap progress is never a recovery authority.

A corrupt proof, registry, manifest, or snapshot prevents activation. The server reports which digest failed. It remains stopped instead of mixing configurations.

9.5. Removed voter returns

Active servers reject any ID outside the current registry before message dispatch. The allocation fence also prevents reassignment of the old ID. The removed server cannot form an old-config quorum by itself. Any old-config phase-one quorum intersects the quorum that chose the stop.

10. Change Surface

Area	Focused change
`src/replicated_log.zig`	Retain the stop-sign algorithm. Add changed-member rollover coverage. Expose the decided stop slot, a by-reference stop-sign accessor, and public member validation for host decoders.
`zaxonlite/src/registry.zig` (new)	Define canonical decided-registry encoding, validation, hashing, durable generations, the node-ID fence, the operation ring, and bootstrap precedence. `configuration.zig` keeps only config and secret loading.
`zaxonlite/src/checkpoint_proof.zig`	Add proof v2 with sealed and next voter sets. Add the next-registry digest. Verify confirmations against the sealed set.
`zaxonlite/src/node.zig`	Prepare replacement checkpoints, bind stop metadata, install the decided next membership, and recover rollover from proof v2.
`zaxonlite/src/server.zig`	Add the privileged operation and status query. Implement the in-process generation swap. Preserve client sessions. Rebuild peer senders and admission from the decided registry.
`zaxonlite/src/enrollment.zig`	Authorize a fresh decided ID only after the stop is chosen. Bind its identity and bootstrap material to the next registry.
`zaxonlite/src/wire.zig`	Version replacement metadata and proof exchange. Reject ambiguous and cross-configuration payloads.
`zaxonlite/src/main.zig`, `zaxonlite/src/client.zig`	Add CLI request, orthogonal quorum and installation fields, structured errors, and leader hints. `membership status` is the CLI's first nested subcommand.
`zaxonlite/src/cluster_test.zig`, `cli_test.zig`, crash and fault tests, `sim/`	Replace static-output expectations where needed. Add deterministic membership, crash, authentication, and end-to-end coverage.
`docs/zds/records/0002-*`, `0003-*`, `0004-*`	Not edited. The amendments section of this record supersedes the affected clauses, so their section numbers and conformance anchors stay valid.

Area	Focused change
`specs/`	Add the bounded host-level replacement model beside `Paxos.tla`, its TLC configuration, and a second action-to-code map in `specs/README.md`.
`docs/zaxonlite/book`	Update the front matter, learning guide, quickstart, CLI, product, storage, cluster, embedding, C ABI, client and gateway, examples, operations, formats, verification, reference, and conformance chapters. Reword the database-identity derivation as bootstrap-only wherever it appears. Keep embedded-node membership explicitly fixed.
`zaxonlite/README.md`, `README.md`	Replace the static-membership and no-replacement limits once the feature lands, and record the new operator commands.

11. Verification Plan

11.1. Model and deterministic simulation

Add a bounded host-level model, `specs/VoterReplacement.tla`, beside `specs/Paxos.tla` with:

- old-quorum stop choice;
- immutable database identity;
- checkpoint and registry digest binding;
- separate sealed and next voter sets;
- activation only after durable install;
- monotonic node-ID and operation-ID fences;
- bounded operation history;
- in-process swap and restart equivalence;
- retired-ID rejection;
- crash and restart at each phase.

The model checks agreement and log-prefix consistency. It allows one active registry per durable generation. It also checks that a new voter cannot vote before installation. No command may appear after the stop.

The deterministic simulator runs `1,2,3 -> 1,2,4`. It varies the leader, message loss, duplication, reordering, and restart schedule.

11.2. Unit tests

- canonical registry encoding is stable across input order;
- registry, manifest, metadata, and proof corruption is rejected;
- duplicate IDs, reused IDs, endpoint conflicts, and role changes are rejected;
- proof confirmation counts distinct sealed voters only;
- a next voter confirmation cannot help satisfy the sealed quorum;
- stop members and registry IDs must match exactly;
- database ID remains byte-identical across replacement;
- registry generation selection survives partial writes;
- the node-ID fence never decreases or wraps;
- IDs at or below the fence cannot be allocated;
- an unchosen candidate ring never replaces the active ring;

- every survivor installs the same decided operation ring;
- retained operation retries are idempotent;
- changed arguments for a retained operation ID are rejected;
- an evicted operation ID returns `OperationHistoryExpired`;
- the ring keeps exactly the newest 32 decided records;
- retired peer and enrollment identities are rejected;
- `quorum_available` follows the configured read and write quorum sizes;
- each installation transition produces the documented status value.

11.3. Crash matrix

Crash and restart are exercised after each of:

1. checkpoint creation;
2. candidate registry sync;
3. stop proposal;
4. stop choice;
5. proof persistence;
6. registry generation sync;
7. active-generation pointer update;
8. Paxos write and peer-dispatch quiescence;
9. next node and peer-set construction;
10. in-memory generation swap;
11. removed-peer connection close;
12. replacement snapshot transfer;
13. replacement snapshot installation;
14. replacement activation.

Before the stop is chosen, restart must recover the old configuration. After the stop is chosen, restart must recover the decided next configuration. A test may not repair state by editing files.

11.4. End-to-end acceptance scenario

1. Start voters `1`, `2`, and `3`; keep one client read connection open.
2. Submit replacement `3 -> 4` and inject an ambiguous client timeout.
3. Retry with the same operation ID and observe one transition.
4. Let voters `1` and `2` activate config `8` while `4` is offline.
5. Read through the existing client connection after the in-process swap.
6. Enroll `4`, install its checkpoint and tail, then activate it.
7. Compare the live in-process result with a clean restart from the same durable files.
8. Bring back `3`; verify authentication and protocol traffic are rejected.
9. Verify `quorum_available` and every `installation_state` transition.

Existing zaxonlite tests remain required. This includes unit, integration, fault-cluster, role-cluster, conformance, and durability tests. Their safety expectations do not change.

12. Security Considerations

Replacement changes who may vote. Only an authorized administrator may start the operation. The request must use mutual TLS. Development PSK mode does not expose this operation.

Administrator identity is a certificate, not a node. Admin certificates use a distinct common-name namespace, `zaxon-admin-<name>`, issued by the same cluster authority as node certificates. The server holds an explicit allow-list of admin names; an empty list denies every privileged

request. A node certificate can never authorize a membership operation. PSK connections carry no certificate at all, so the operation is structurally unreachable in development mode.

The chosen stop sign binds the next member IDs. The registry digest binds their roles and endpoints. Proof v2 rejects a snapshot or registry from another database or configuration.

The allocation fence prevents identity reuse. Exact-registry admission stops an old credential from becoming a current voter.

The operation is always operator initiated. Reachability does not authorize a replacement. A failed health check or expired timer is also insufficient.

13. Operational Considerations

The old configuration needs a leader and write quorum. Without them, it cannot choose the replacement.

The next configuration needs a reachable quorum to resume before catch-up. In a three-voter cluster, the other two voters must be healthy when replacing an unavailable voter.

Operators should expect:

- one checkpoint transfer to the new voter;
- a bounded write pause at the epoch boundary;
- client read TCP connections to stay open across the in-process swap;
- reduced fault tolerance until the replacement reports `complete`;
- permanent retirement of the old node ID;
- stable database identity and client endpoint semantics.

The server refuses a second membership operation while one is pending. Status must show progress and the safe retry action. Operators should not need to inspect the journal.

`quorum_available` is a monitoring signal. It may change as peers connect or time out. It must not be used as proof that consensus chose a replacement.

14. Rollout

1. Land canonical registry storage, the node-ID fence, and the operation ring.
2. Land proof v2 and validate same-member checkpoints through the new verifier.
3. Extend the existing changed-member `ReplicatedLog` simulation and add the deterministic host schedule.
4. Add the administrative request and stop metadata binding.
5. Add the in-process generation swap and restart-equivalence tests.
6. Add decided-ID enrollment, snapshot installation, and retired-ID rejection.
7. Add the orthogonal status fields.
8. Run the crash matrix and end-to-end scenario under mutual TLS.
9. Enable the CLI command only after all gates pass.

Deployment has not launched. The persisted registry, metadata, and proof can move directly to their new versions. No compatibility decoder or migration shim is required.

15. Acceptance Criteria

- One data voter is replaced without changing voter count or database ID.
- The old configuration alone chooses the stop sign.
- The new registry's member IDs exactly equal the chosen next member IDs.

- A replacement cannot vote before durable verified installation.
- A removed or reused node ID is rejected after activation.
- `highest_allocated_node_id` never decreases or wraps.
- Operation memory remains fixed at 32 records.
- Operation IDs never decrease or wrap.
- Retained retries are idempotent and expired operation IDs are rejected.
- Startup uses the decided registry, not conflicting bootstrap arguments.
- Every crash point converges to one complete configuration.
- The primary activation path uses an in-process generation swap.
- Existing client read TCP connections remain open across that swap.
- In-process activation and clean restart produce identical state.
- Survivors can resume with a next-config quorum before catch-up completes.
- Status exposes `quorum_available` and `installation_state` separately.
- `active-degraded` remains a summary and does not hide quorum health.
- Full restart preserves state, registry digest, proof digest, and configuration ID.

16. Alternatives Considered

16.1. Edit startup membership and restart

Independent edits can give servers different voter sets. Consensus does not choose those edits. A partial rollout then has no safe point or authoritative answer.

16.2. Let the new voter participate while catching up

A node without the chosen prefix has incomplete state. It must not acknowledge or campaign. Durable installation is a prerequisite for voting.

16.3. Activate endpoints directly from stop member IDs

The stop sign contains IDs. It does not contain authenticated endpoints and roles. The canonical registry digest binds that product-level mapping to the consensus decision.

16.4. Mutate transport in place without an epoch boundary

An in-place mutation could reduce the pause. It also creates mixed states between the node, senders, admission, and recovery. A controlled rebuild is easier to specify and crash-test.

16.5. Use a full process restart as the primary path

A restart gives a simple recovery boundary. It also closes healthy client connections. The in-process generation swap preserves those connections and uses the same durable pointer as restart. Restart remains the recovery oracle and failure path.

17. Resolved Decisions

- `server.zig` uses an in-process generation swap as the primary activation path. It must produce the same result as crash and restart.
- `highest_allocated_node_id: u32` retires node IDs with constant memory. A fixed ring in the decided registry retains 32 operation records. Node and operation IDs are monotonic and never wrap.
- Status exposes `quorum_available` and `installation_state` as separate fields. `active-degraded` remains a high-level phase.
- The decided registry lives in a new `registry.zig` module with its own `REGISTRY` pointer file beside `CURRENT`, reusing the node's atomic write primitive and recovery ladder.

- Survivors reconstruct the next registry deterministically from `zx2` stop metadata. Only the replacement fetches the blob, and it verifies the decided digest.
- Administrators authenticate with `zaxon-admin-<name>` certificates checked against an explicit allow-list.
- Versions move together: wire protocol 6 to 8, proof `ZXP1` to `ZXP2`, stop metadata `zx1` to `zx2` with a 512-byte bound.

No design question remains open for this replacement operation.

18. Amendments to Prior Records

This section follows the amendment pattern of ZDS 0006. The prior records are not edited. Their section numbers, and the conformance clauses anchored to them, stay valid. Where a clause below and this record disagree, this record wins once it reaches the `committed` state.

18.1. Amendments to ZDS 0002

The non-goal of automatic voter replacement stands; this record adds an operator-initiated operation. ZDS 0002 already names the safe form in its rqlite review: stop-sign epoch transition, learner catch-up proof, and operational fencing. That row, and the roadmap paragraph on runtime voter reconfiguration, are superseded for the one-for-one replacement case. The reserved `members add` and `members remove` commands are not implemented; the operator surface is `replace-voter` and `membership status`. The TLA+ obligation ZDS 0002 records for dynamic reconfiguration is discharged by the model in this record.

18.2. Amendments to ZDS 0003

The statements that membership stays operator-configured and static, that eviction is an offline procedure, and that a membership-changing snapshot is rejected by the transfer path are superseded for this operation. The forward-looking clause of that record is now in force: a decided stop sign with a new member set atomically replaces the transport allowlist and closes sockets for removed nodes, and consensus membership overrides certificate validity. Its open question, which quorum attests a membership-changing snapshot, is answered: a read quorum of the sealed voter set. The security matrix gains one row: the privileged replacement request, authenticated by an admin certificate against an explicit allow-list, absent from development PSK mode. Enrollment now authorizes a decided next-registry identity after the stop is chosen; it still never chooses membership itself.

18.3. Amendments to ZDS 0004

The release limit of one through nine static voters is amended: the voter count stays fixed, and one data voter can be replaced through the decided operation in this record. Enrollment's scope is amended the same way: it remains a narrow bootstrap that grants no role, but the registry it checks is the decided registry, not the startup flags. The identity section gains the decided registry, the `REGISTRY` pointer, and the node-ID allocation fence; the rule that a node ID is never reused is unchanged and is now enforced by that fence. The wire protocol moves from version 6 to version 8 with exact-major acceptance unchanged. Version 8 adds an explicit durable installation announcement. The checkpoint proof moves from `ZXP1` to `ZXP2`. The stop metadata moves from `zx1` to `zx2` with a 512-byte bound. The database ID derivation becomes bootstrap-only. These are direct replacements, permitted because deployment has not launched and there are no shipped bytes to migrate.

19. References

- `src/replicated_log.zig` - chosen stop sign and next-configuration rollover
- `zaxonlite/src/node.zig` - checkpoint, stop handling, durable transition, and snapshot installation
- `zaxonlite/src/checkpoint_proof.zig` - checkpoint proof encoding and quorum confirmation
- `zaxonlite/src/registry.zig` - decided registry, fence, and operation ring (new module)
- `zaxonlite/src/server.zig` - peer registry, transport, administration, and database identity
- `zaxonlite/src/enrollment.zig` - mutually authenticated node enrollment
- `docs/zds/records/0002-zaxonlite-product-plan.typ` - product-host verification boundary
- `docs/zds/records/0003-zaxonlite-security-remediation-plan.typ` - trust model, enrollment, and eviction rules amended above
- `docs/zds/records/0004-zaxonlite-format.typ` - persisted and wire format registry
- `docs/zds/records/0006-windows-durability.typ` - the amendment pattern for committed records