Zaxon Working Group
Request for Discussion and Implementation Record

ZDS 0007
Engineering Discussion

paxos-zig 0.1.x Safety Hardening

STATE
committed

CREATED
2026-07-27

LAST UPDATED
2026-07-28

INTENDED STATUS
Implemented for 0.1.2

AUTHORS
paxos-zig project

DISCUSSION

A focused patch release that enforces the durability boundary in every build mode

LABELS
paxos, durability, verification, release

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

paxos-zig depends on one load-bearing promise from its host. The host must make each promise or vote durable before it releases a message that claims the write happened.

The API records this promise through `confirmWritesDurable`. Before this release, the misuse check was a debug assertion. `ReleaseFast` and `ReleaseSmall` could remove that assertion. An invalid call order could then continue in those builds.

This record defines a small 0.1.x hardening release. The default check will run in every build mode. Hosts that own group durability must use a separate and clearly named API. Invalid compile-time options will produce clear compiler errors. Tests will verify the contract in all four Zig optimization modes.

The normal host sequence does not change. Paxos messages, state transitions, storage records, and snapshots also remain unchanged.

2. Introduction

The protocol core performs no I/O. A transition returns an `Effects` value. It contains durable writes, outbound messages, and newly committed values.

This separation keeps the core deterministic. It also gives the host control over the durability boundary. The normal sequence is:

1. append every item in `writesSlice`;
2. run the required storage barrier;
3. call `confirmWritesDurable`;
4. release `messagesSlice`;
5. apply `committedSlice`.

`preDurableMessages` is the only named exception. It exposes phase-two accept requests. These requests ask another node to make a vote durable. They do not claim that the sender's own vote is already durable.

The sequence is already documented. zaxonlite already follows it. The problem is narrower. The default API guard is strong in some build modes and absent in others. A safety contract must have the same meaning in every build.

3. Terminology and Scope

- **durable-claim message**: a message that depends on a local promise or vote surviving power loss
- **effect order**: writes first, then the storage barrier, then confirmation, and finally message release
- **enforced host**: a host that uses the normal `paxos.Protocol` declaration
- **host-managed batching**: a host that owns an external write and message batch and releases no message before the shared barrier
- **optimization matrix**: Debug, ReleaseSafe, ReleaseFast, and ReleaseSmall

This record covers the core library and every repository consumer. It also covers examples, tests, simulations, benchmarks, documentation, and zaxonlite.

This record does not change the wire protocol or durable formats. It does not change Paxos ballots, quorums, slots, elections, recovery, or reconfiguration.

4. Problem Statement

`Effects.reset` and `Effects.messagesSlice` use `std.debug.assert` to protect effect order. Zig can remove this check in optimized builds. The process can then continue after a host integration error.

That behavior is unsafe. A node can send a promise or vote and lose it after power failure. Other voters may have already relied on the lost claim.

The current escape hatch is `assert_effect_order = false`. It is a boolean inside the normal `Options` struct. This makes the exception hard to audit. It is also easy to copy by accident.

The durable benchmark needs a real exception. In both of its modes it copies writes and messages into host-owned batches, performs its barrier outside `Effects`, and never calls `confirmWritesDurable`. The grouped mode also keeps the message queue private until one shared barrier completes. This host owns the safety obligation outside `Effects` in both modes, so the whole fixture moves to the exception namespace.

Compile-time validation has a similar build-mode problem. Several capacity and timer checks use debug assertions. Invalid type options should fail in every build mode.

5. Goals and Non-Goals

5.1. Goals

- Enforce the default durability boundary in all optimization modes.
- Keep the normal host call sequence unchanged.
- Put host-managed batching behind a separate and searchable declaration.
- Remove the effect-order escape hatch from normal `Options`.
- Produce clear compile errors for invalid type options.
- Test the enforced and host-managed paths.
- Run misuse tests in every optimization mode.
- Measure the release with the same workload, machine, and toolchain.
- Update all affected source comments and documentation in the same patch.

5.2. Non-Goals

- No change to wire or storage formats.
- No change to Paxos state transitions.
- No new storage abstraction.
- No new transport abstraction.
- No unrelated module reorganization.

6. Design Overview

The normal declaration always enforces effect order:

```
const paxos = @import("paxos");

const P = paxos.Protocol(u64, .{
    .max_members = 3,
    .max_slots = 512,
});
```

Normal `Options` will not contain `effect_order` or `assert_effect_order`. There is no boolean opt-out.

An audited batching host must select a separate namespace:

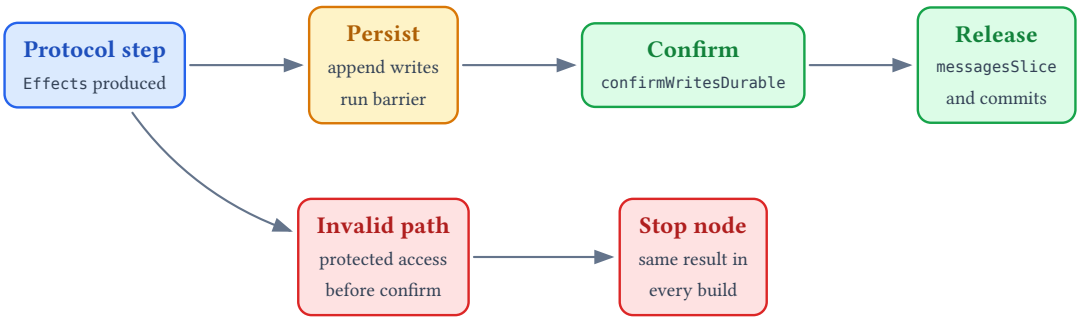
```
const paxos = @import("paxos");

const P = paxos.host_managed.Protocol(u64, .{
    .max_members = 3,
    .max_slots = 512,
});
```

The name `paxos.host_managed.Protocol` shows that the host owns the boundary. A repository search for `host_managed.Protocol` finds every exception. An engineer does not need to inspect every field in a large options value.

The namespace changes ownership only. It does not weaken the Paxos rule. The host must still keep all protected messages private until the write barrier completes.

6.1. Default Transition



7. Detailed Design

7.1. Always-on boundary

`Effects.messagesSlice` checks for unconfirmed writes. The check runs before it returns protected messages. `Effects.reset` runs the same check before it clears the current batch.

The normal declaration uses an always-on failure helper. The helper does not depend on `std.debug.assert` or `std.debug.runtime_safety`.

A violation stops the process. Continuing is not safe. A stopped node reduces availability. A node that forgets a promise or vote can break agreement.

The check stays narrow:

- messages are available at once when the transition has no writes;
- commit-only effects keep their documented weak-barrier treatment;
- `preDurableMessages` exposes only accept requests;
- internal array and state-machine assertions keep their current behavior.

7.2. Separate host-managed declaration

`src/host_managed.zig` defines the explicit exception namespace. `src/root.zig` exports it as `paxos.host_managed`.

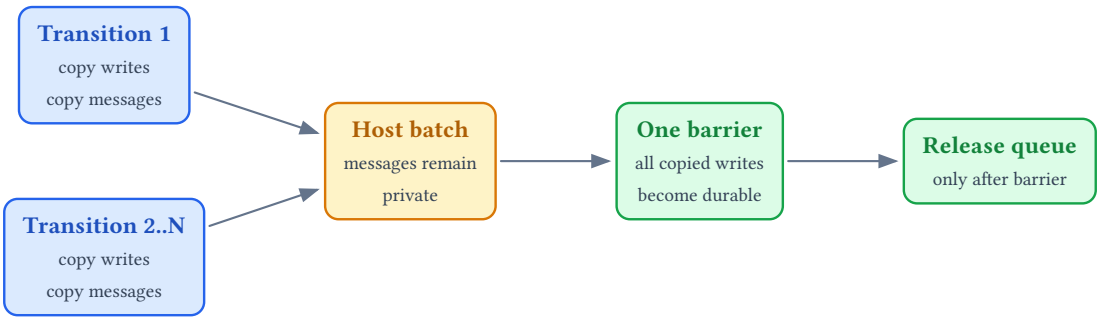
The namespace exposes a `Protocol` factory with the same capacity options as the normal factory. Internally, both factories share one implementation. The normal factory enables the ordering gate. The host-managed factory does not.

The regular `Options` type contains no ownership policy. Copying a normal options value cannot disable the gate. Every exception must spell `host_managed.Protocol` at the type declaration.

The API documentation must place a warning on the namespace and factory. The warning must state that the host now owns a load-bearing safety rule. It must also link to the four obligations below.

7.3. Host-managed batching

The durable benchmark collects several transitions. It copies their writes and messages into an external batch. It then performs one barrier. Only then does it release the queued messages.



A host-managed consumer must meet four obligations:

1. Copy writes and messages before the next call resets `Effects`.
2. Keep the pending message queue private from the sender.
3. Release no message and report no client success after a failed append or barrier.
4. Rebuild the same durable state from the written prefix after restart.

The durable benchmark is the first consumer. Any production consumer needs the same level of crash testing.

7.4. Compile-time validation

Type factories will replace option-related debug assertions with `@compileError`. The compiler will reject:

- zero `max_members` or `max_slots`;
- member or slot bounds that do not fit their wire types;

- zero election, heartbeat, or resend intervals;
- zero `max_batch`;
- `max_batch` greater than `max_entries`;
- metadata lengths that do not fit their encoded count;
- bit sets with zero bits;
- derived effect capacities that overflow.

Quorum validation remains a runtime check. The member slice is supplied to `Membership.init` at runtime.

7.5. Diagnostics

Runtime diagnostics name the failed operation and the missing predecessor. They contain no values, payloads, or addresses.

```
paxos: messagesSlice before confirmWritesDurable
```

Reset has its own diagnostic:

```
paxos: reset discarded unconfirmed writes
```

Compile errors name the invalid option and the required relation:

```
paxos Protocol option max_slots must be greater than zero
```

Tests compare only the stable identifying substring. They do not compare the full compiler or panic output. Toolchain prefixes, source locations, and stack traces may change without changing the contract.

8. Change Surface

Area	Focused change
`src/protocol.zig`	Remove `assert_effect_order`. Add the always-on helper and shared internal factory. Convert option checks. Extend effect tests.
`src/host_managed.zig`	Add the explicit `host_managed.Protocol` factory and its safety warning.
`src/root.zig`	Export `host_managed`. Describe default enforcement in every build.
`src/replicated_log.zig`, `src/learner.zig`, `src/bit_set.zig`	Convert compile-time option assertions to clear compiler errors. Do not change runtime behavior. `ReplicatedLog` always builds on the enforced factory and gets no host-managed variant.
`README.md`, `examples/`, `sim/`, `integration/`	Keep the normal declaration and call order. Update all affected prose.
`build.zig`, continuous integration	Add the four-mode misuse matrix. Add compile-fail option fixtures.
`benchmarks/durable.zig`	Use `paxos.host_managed.Protocol`. Keep the grouped barrier and recovery self-check. Explain the transferred obligation.

Area	Focused change
`benchmarks/benchmark.zig`	Use the normal declaration. Record comparable before and after results.
`zaxonlite/src/node.zig`, `zaxonlite/src/types.zig`	Keep the normal declaration. Confirm that journal sync still comes before protected message release.
`docs/book`, `docs/zaxonlite/book`, `CHANGELOG.md`	Update the API, style, storage, engineering, and conformance text. Record the removed option, new namespace, diagnostics, and release evidence.

9. Verification Plan

9.1. Unit and misuse tests

- A normal write batch becomes readable after `confirmWritesDurable`.
- A no-write transition is readable immediately.
- Commit-only effects keep their documented barrier treatment.
- `preDurableMessages` returns accepts and no durable-claim message.
- `host_managed.Protocol` supports the grouped durability fixture.
- Normal `Options` has no effect-order escape hatch.
- Each invalid option has a compile-fail fixture.

The protected-read and reset misuse cases run as child processes. A correct test observes a nonzero exit. It also finds the stable diagnostic substring. It ignores the rest of stderr.

Each misuse fixture runs in Debug, ReleaseSafe, ReleaseFast, and ReleaseSmall. A successful ReleaseFast child process blocks the release.

9.2. Existing gates

The patch runs these gates without reducing counts or schedules:

```
zig build fmt
zig build test
zig build test-zaxonlite
zig build benchmark-zig
zig build benchmark-durable
```

Existing gates keep their safety expectations. The seeded fault simulation and the reconfiguration harness run under `zig build test`. The zaxonlite package runs its own crash, cluster, role, fault-network, and C ABI suites from its directory. The path-dependency integration consumer, previously an orphan step, now runs under `zig build test`. The conformance appendix is prose kept in sync with the handlers, not a build step.

9.3. Performance gate

Run the in-memory benchmark before and after the patch. Use the same machine, toolchain, build mode, workload, and sample count. Record the raw results and the median.

Investigate either of these changes:

- throughput falls by 3% or more;
- latency rises by 3% or more.

The 3% threshold starts an investigation. It is not an automatic rejection. The investigation must check measurement noise, generated code, branch placement, and benchmark setup.

The safety check is non-negotiable. A performance result cannot justify removing it. Any optimization must preserve the always-on default.

9.4. Release evidence

The existing benchmark archives are development observations. Both record a dirty worktree. The older run also predates unrelated protocol changes. They do not form a clean before-and-after pair for this hardening change.

Before tagging 0.1.2, record a clean comparison. Use the commit immediately before the hardening change as the baseline. Use the 0.1.2 release commit for the second run. Keep the machine, toolchain, build mode, workload, and sample count the same. Store both raw result files and the calculated deltas.

The release owner must investigate every throughput drop or latency increase of 3% or more. Until the clean pair exists, the performance gate is pending. The functional safety implementation does not depend on that measurement.

10. Security Considerations

This change reduces the effect of a host integration bug. An optimized binary will stop instead of releasing an unsafe claim.

The library still cannot prove that a storage device completed a correct barrier. `confirmWritesDurable` remains a statement made by the host.

`paxos.host_managed` is a trust boundary. Its use must be easy to search. Each use needs written ownership notes and a crash-recovery test. The release audit must list every use.

11. Operational Considerations

Correct hosts keep the same runtime flow. Incorrect optimized hosts now stop at the first protected access.

Operators should treat either diagnostic as a software integration failure. They should preserve the journal and logs. They should restart only after the host bug is fixed.

This is a pre-launch 0.1.x release. We do not need a compatibility alias for `assert_effect_order`. Removing the old option keeps the API clear.

12. Rollout

1. Add the shared internal factory and always-on helper.
2. Add the `paxos.host_managed` namespace.
3. Remove `assert_effect_order`.
4. Convert compile-time option checks.
5. Migrate the durable benchmark.
6. Update every repository consumer and affected document.
7. Run the four-mode matrix and full repository gates.
8. Record benchmark data and investigate any 3% regression.
9. Tag 0.1.2 after every release gate passes.

Documentation must never claim always-on enforcement while an optimized build still omits the check.

13. Acceptance Criteria

- Default misuse stops in all four optimization modes.
- Normal `Options` contains no effect-order switch.

- The old option is absent from compilable source, active configuration examples, and public API tables. Historical prose may name it when explaining its removal.
- Every exception uses the searchable `paxos.host_managed` namespace.
- Each exception has ownership notes and crash-recovery coverage.
- Tests compare only stable diagnostic substrings.
- Invalid compile-time options fail with readable diagnostics.
- Existing protocol and zaxonlite safety tests pass.
- Durable replay matches live state in both benchmark modes.
- A clean before-and-after benchmark pair is recorded for 0.1.2.
- A 3% or greater throughput or latency regression in that pair is investigated.
- The safety check remains enabled regardless of benchmark results.
- README, books, API comments, and changelog describe the same contract.

14. Alternatives Considered

14.1. Return an error from `messagesSlice`

This error would represent a programming violation. Every correct caller would still need to handle it. A caller could also ignore it. The node cannot safely continue, so a fatal result is clearer.

14.2. Keep a policy field in normal Options

This would make the exception easy to miss inside a large options value. It would also make accidental copying more likely. A separate namespace is easier to review and search.

14.3. Keep the old boolean

The boolean names an implementation detail. It does not show who owns the safety rule. It also keeps the escape hatch in the normal API.

14.4. Require ReleaseSafe

ReleaseSafe remains the recommended deployment mode. Build mode is not a substitute for a boundary check. The contract must remain the same in every mode.

15. Resolved Decisions

- Host-managed batching uses the separate `paxos.host_managed` namespace.
- A 3% throughput drop or latency increase requires investigation.
- Tests compare only stable diagnostic substrings.

No design question remains open for this hardening release.

16. References

- `src/protocol.zig` - `Options`, `Effects`, and the durability boundary
- `src/root.zig` - public package declarations
- `src/replicated_log.zig` - replicated-log option validation
- `benchmarks/durable.zig` - grouped host-managed durability fixture
- `zaxonlite/src/node.zig` - journal, barrier, confirmation, and message order
- `docs/book/04_style.typ` - errors, assertions, and optimized-build policy
- `README.md` - public host integration sequence