POD 0002: Paxos-Odin: Architecture and Pure State Machine Design

COMMITTED

CATEGORY	**STATUS**
Architectural Specification	Committed
AUTHORS	**CREATED**
Vikrant Rathore <vikrant@insan.ai>, Paxos Odin Contributors	2026-09-16
LAST UPDATED	**LABELS**
2026-09-17	architecture, consensus, odin, data-oriented

Abstract

This document specifies the architecture of `paxos-odin` version `0.2.0` (`VERSION` in `src/paxos.odin`): a bounded, data-oriented implementation of Classic and Multi-Paxos written as a deterministic effect machine in Odin. The library performs no I/O, owns no threads or clocks, and allocates nothing on the heap during a consensus transition. A `Node` consumes one input (an envelope, a proposal, or a tick) and fills a caller-owned `Effects` value with everything the host must do next.

Version `0.2.0` is a ground-up redesign of the core. The acceptor's state is a `Ledger` whose columns are Lamport's variables (`maxBal`, `maxVBal`, `maxVal`, and the decision) laid out as struct-of-arrays over a power-of-two window; a `Ballot` is one packed 64-bit integer; values are referenced by pointer in every message, record, and released entry instead of being copied; and the node can run with rotating slot ownership (POD 0010) as an alternative to a single elected leader. This document describes the type surface, the data layout, the ten core files, the replicated log and learner layers, the error contract, and the verification that exists today. POD 0009 records the reasoning behind the layout; POD 0008 gives the safety argument.

Status and Implementation Boundary

This committed specification describes the implemented core. "Pure" indicates that the core performs no direct I/O; state transitions mutate the node's memory and append to caller-provided effects buffers. `recovery_index` checks the slot against the active range before subtracting `recover_base` and narrowing the result. Recovery scratch and each peer's seen bitmap have chunk capacity. `recovery_ready` freezes selection before phase two; backpressure retries cannot let a late report change a proposal at the same ballot. `resend_to` wraps at most once and does not repeat a used cell in one scan.

`start_campaign` records its own `Write_Promise` before broadcasting Prepare. `start_revocation` first runs `promise_bounded` locally. Both facts are needed for ballot uniqueness after a crash, including when pre-durable accepts are enabled. Ownership exposes `resubmits_dropped` on nodes and replicated logs: a bounded resubmission queue supplies best-effort delivery, not an at-least-once guarantee.

Architecture Overview

`paxos-odin` separates the algorithm from every runtime concern:

1. **Deterministic inputs.** The host calls `paxos.step(&node, envelope, &effects)`, `paxos.propose(&node, value, &effects)`, `paxos.campaign(&node, noop, &effects)`, or `paxos.tick(&node, noop, &effects)`. Ticks are the only clock; the host chooses their cadence.
2. **Pure transition.** The node mutates its own fixed-size arrays and bitmaps. No procedure in the core imports a socket, a file, or a timer. Every transition begins with `effects_reset(effects)`.
3. **Explicit effects.** The transition appends to four bounded lists inside `Effects`:
   - `writes`: durable records (`Write_Promise`, `Write_Promise_At`, `Write_Vote`, `Write_Chosen`, `Write_Trim`) the host must persist, in order, before it sends anything;
   - `messages`: `Envelope` values addressed to members;
   - `committed`: newly decided entries (`Committed{slot, value: ^Value}`), released in contiguous slot order (obligation L1 in `src/paxos.odin`);
   - `requests`: `Serve_Range_Request` values for history below the node's memory floor, which the host serves from its own journal or state image.

Nothing in a batch owns a value: writes, messages, and committed entries point into the node's ledger, and every pointer is valid until the next transition on the node that produced it. POD 0003 specifies the durability contract that binds the four lists together and the copy obligation that follows from the pointers.

The Ten Core Files

`src/paxos.odin` holds `VERSION`, the capacity and timer defaults, `MAX_SUPPORTED_MEMBERS`, `INVARIANT_CHECKS`, and the proc-group surface. Beneath it the core is ten files, each with one responsibility:

File	Responsibility
`ballot.odin`	`Node_Id :: u16`, `Slot :: u64`, the packed `Ballot`, `ballot_make` and its accessors, `cell_of` (the window mask), and `slot_add` (saturating slot arithmetic).
`bit_set.odin`	`Bit_Set(N)`: an array of native `bit_set[0..<64]` words providing `bit_set_insert`, `bit_set_remove`, `bit_set_contains`, `bit_set_count`, `bit_set_reset`, `bit_set_next`, and `bit_set_last`. Used by memberships and window bitmaps.
`membership.odin`	`Membership(MAX_MEMBERS)` with its stable member index (members sorted by id), `LINEAR_LOOKUP_LIMIT`, and quorum validation (obligation B2).
`ledger.odin`	`Ledger(Value, WINDOW)`, `Cell_State`, `Trim_Anchor`, the five `Write` records, cell access (`ledger_cell`, `ledger_vote_at`, `ledger_chosen_at`, `ledger_open`, `ledger_claim`, `ledger_record_vote`, `ledger_record_chosen`), and journal replay (`ledger_apply`, `ledger_replay_fold`).
`messages.odin`	The nine wire messages, `Prepare_Scope`, `Message(Value)`, `Envelope(Value)`, `message_value`, `Committed(Value)`, and `Host_Request`.
`effects.odin`	`Durability_Gate`, `Effects(Value, M, W, C, GATE)` with its exact capacities, the runtime gate, the accessors, `effects_requires_power_loss_barrier`, and the pre-durable iterator.
`node.odin`	`Role`, `Node_Options`, `Election_Peer`, the `Node` struct, lifecycle (`node_init`, `node_init_learner`, `node_restore`, `node_continue_at`, `node_restore_learner`, `node_begin_recovery`), the memory floor, trim installation, the queries, and `node_assert_valid`.

`election.odin`	Phase one: campaigns, `on_prepare` and `promise_bounded`, `on_promise` and `on_promise_range`, `quorum_fences`, `maybe_resolve_chunk`, `resolve_chunk` (obligation B3), `begin_next_chunk`, and `become_leader`.
`consensus.odin`	Phase two consensus transitions and steady-state execution: `claim_live`, `emit_contiguous`, `send_accept`, `on_accept`, `on_accepted`, `on_commit`, `record_commit`, heartbeats, `on_learn`, `on_nack`, proposals and batches, `node_tick`, `resend_to`, and `node_step`, the single dispatch point.
`ownership.odin`	Rotating slot ownership: `owner_of`, `ownership_ballot`, own-slot arithmetic, `propose_owned`, skips (`SKIP_BURST`), `start_revocation`, resubmission, and `tick_ownership`.

Three higher-level files layer directly on the core: `replicated_log.odin` (stop signs, sealing, `Log_Envelope`), `learner.odin` (the standalone non-voting window), and `errors.odin` (the `Error` enum and its explanations). Every file respects the structural limits of POD 0001 (`tools/check_style.py`: at most 1,408 lines per file, 108 columns per line, 70 lines of logic per procedure).

Types and Capacities

Shared type parameters

`Node` and `Effects` are declared with the same five parameters:

```
Node    :: struct($Value: typeid, $MAX_MEMBERS: int = DEFAULT_MAX_MEMBERS,
                  $WINDOW_SLOTS: int = DEFAULT_WINDOW_SLOTS,
                  $CHUNK_SLOTS: int = DEFAULT_CHUNK_SLOTS,
                  $GATE: Durability_Gate = .Enforced)
Effects :: struct($Value: typeid, $MAX_MEMBERS: int = DEFAULT_MAX_MEMBERS,
                  $WINDOW_SLOTS: int = DEFAULT_WINDOW_SLOTS,
                  $CHUNK_SLOTS: int = DEFAULT_CHUNK_SLOTS,
                  $GATE: Durability_Gate = .Enforced)
```

Every transition procedure takes `^Node(V, M, W, C, G)` and `^Effects(V, M, W, C, G)` with the parameters spelled once, so the compiler rejects an `Effects` whose parameters differ from its `Node` (the `effects_must_match_node` fixture in `tools/check_contracts.py`). `Node`, `Effects`, `Ledger`, `Replicated_Log_Node`, and `Learner` carry `where intrinsics.type_is_comparable(Value)`; equality is native `==`, so a `map` value is rejected at compile time (`non_comparable_value` fixture). The defaults are `DEFAULT_MAX_MEMBERS = 7`, `DEFAULT_WINDOW_SLOTS = 256`, `DEFAULT_CHUNK_SLOTS = 64`, `DEFAULT_MAX_METADATA_BYTES = 256`, and `DEFAULT_MAX_ENTRIES = 256`.

`node_init` carries three `#assert` checks whose messages end in a `Hint:`: `MAX_MEMBERS` in `1..=65535` (`MAX_SUPPORTED_MEMBERS`, because member indexes and the node field of a ballot are 16 bits), `WINDOW_SLOTS` a power of two (the cell of a slot is one mask), and `CHUNK_SLOTS` in `1..=WINDOW_SLOTS`. `tools/check_contracts.py` proves each rejection (`zero_window`, `window_not_power_of_two`, `zero_chunk`, `chunk_exceeds_window`, `too_many_members`, and `zero_members`).

Effects capacities

Each list in `Effects` is a `small_array.Small_Array` sized to the exact per-transition maximum:

- `writes`: `2 * CHUNK_SLOTS + 1` (one recovery chunk of votes, each possibly followed by a local decision, plus one promise);
- `messages`: `MAX_MEMBERS * CHUNK_SLOTS + 2 * MAX_MEMBERS + 1` (one chunk per peer plus prepares, heartbeats, and one catch-up request);

- `committed`: `WINDOW_SLOTS + 1` (one window plus one pass-through entry);
- `requests`: `MAX_MEMBERS`.

Overrunning any list is an `assert` inside the `effects_add_*` procedures, never silent truncation. Because a `Write(Value)` is 32 bytes and an `Envelope(Value)` is 72 bytes whatever the size of `Value` (the value travels by pointer), the capacities are independent of the payload type. The zero value of `Effects` is ready to use; `effects_init` clears a batch without checking the gate and `effects_reset` clears it with the check.

Node_Options

`node_init(node, id, membership, options := Node_Options{})` takes one options struct in which zero means the default: `priority: u8` (ballot tie-breaker, zero is lowest), `election_timeout_ticks`, `heartbeat_interval_ticks`, and `resend_interval_ticks` (zero selects `DEFAULT_ELECTION_TIMEOUT_TICKS = 10`, `DEFAULT_HEARTBEAT_INTERVAL_TICKS = 3`, `DEFAULT_RESEND_INTERVAL_TICKS = 10`), `gate_proposals_on_inherited_prefix`, `campaign_disabled`, and `rotating_ownership`. The last selects the ownership mode of POD 0010: campaigns return `.Campaign_Disabled`, `propose` goes to the node's next own slot without phase one, and stalls are repaired by bounded revocations.

Core Data Structures

Ballots

`Ballot :: distinct u64` packs three fields so that Lamport's B1 (a total order with unique ballots) is integer comparison and every message and record carries eight bytes:

- bits 63..24: `round` (40 bits, the campaign counter; `MAX_ROUND` is `1 << 40 - 1`, and `.Ballot_Exhausted` stops a node that would pass it);
- bits 23..16: `priority` (8 bits, breaks ties between rounds);
- bits 15..0: `node` (16 bits, the proposer, which makes every ballot unique).

`ballot_make(round, priority, node)`, `ballot_round`, `ballot_priority`, and `ballot_node` are `#force_inline` shifts and masks. Round zero is reserved: under rotating ownership `ownership_ballot(owner) = ballot_make(0, 0, owner)` belongs to a slot's owner alone, and no campaign ever uses it (`start_campaign` and `start_revocation` begin at `greatest + 1`, where `greatest` covers `highest_observed_round`, the node's own ballot, and `ledger_highest_ballot`). `BALLOT_ZERO` is the empty promise.

The ledger: Lamport's variables in columns

`Ledger(Value, WINDOW)` in `src/ledger.odin` is the acceptor's whole durable state and the node's durable per-slot storage:

```
Ledger :: struct($Value: typeid, $WINDOW: int = DEFAULT_WINDOW_SLOTS)
  where intrinsics.type_is_comparable(Value) {
  promised:    Ballot,
  anchor:      Trim_Anchor,
  slot:        [WINDOW]Slot,
  promised_at: [WINDOW]Ballot,
  vote_ballot: [WINDOW]Ballot,
  state:       [WINDOW]Cell_State,
  value:       [WINDOW]Value,
  used:        Bit_Set(WINDOW),
  chosen:      Bit_Set(WINDOW),
}
```

The mapping to "The Part-Time Parliament" is direct. `promised` is `maxBal` for every decree at or above the recovery base of the promise; `promised_at[c]` is a per-decree promise recorded by a bounded prepare (a revocation under ownership), and the effective promise for a cell is `ledger_promise_for = max(promised,`

promised_at[c]); vote_ballot[c] is maxVBal; value[c] is maxVal, or the chosen value once state[c] is .Chosen. Cell_State is Empty, Voted, or Chosen. There is no per-slot struct: a scan touches only the columns it needs, so a phase-one answer walks used, slot, and vote_ballot and never loads a value it does not report.

slot[c] tags which slot owns cell c, so a reused cell is never mistaken for an older one; cell_of(slot, WINDOW) is (slot - 1) & (WINDOW - 1). ledger_open retags a cell and clears everything but the value storage. node_assert_valid (compiled in when INVARIANT_CHECKS is true, which defaults to ODIN_DEBUG and can be overridden with -define:PAXOS_INVARIANT_CHECKS) checks that every tag is in its own cell, that no vote exceeds its cell's promise, and that both bitmaps agree with state.

Bitmaps and scans

used marks cells that hold a vote or a decision; chosen marks decisions. bit_set_next(bs, from) finds the smallest member at or after from by masking one 64-bit word and counting trailing zeros, then walks whole words. Every window walk in the core uses it: on_prepare reports the votes and decisions in a chunk by walking used; on_learn answers a catch-up request by walking chosen; resend_to rotates a per-peer cursor through used so a quiet peer cannot pin retries to the first cells; ledger_highest_used and replicated_log_pending_stop_sign walk used; replicated_log_observe_durable walks chosen. The tests test_bit_set_basic, test_bit_set_scans_across_words, and test_native_bit_set cover the primitive, including scans that cross a word boundary.

Values are referenced, not copied

Promise_Message, Accept_Message, and Commit_Message carry value: ^Value; Write_Vote and Write_Chosen carry value: ^Value; Committed carries value: ^Value. Outbound, a pointer refers into the sending node's ledger (&l.value[cell], or &node.pass_through for a decision released past the window edge) and stays valid until that node's next transition. Inbound, the host points it at the decoded value for the duration of the step call. message_value(message) returns the pointer and whether the kind carries one. An in-process transport that queues envelopes copies the value at enqueue time, as a codec would: the Packet type and packet_of/packet_envelope procedures in tests/harness.odin, examples/counter.odin, sim/simulation.odin, and bench/main.odin illustrate this pattern. Message(Value) is 64 bytes and Envelope(Value) 72 bytes for every Value.

Membership and quorums

Membership(MAX_MEMBERS) holds members (a small_array of Node_Id sorted by id whatever order the host gave; that position is the member's stable index in every per-member array of a node and the owner order under rotating ownership), read_quorum_size, and write_quorum_size. membership_index_of scans linearly for memberships of at most LINEAR_LOOKUP_LIMIT = 8 members and binary-searches the sorted members above that. membership_init validates non-zero unique ids, defaults both quorums to a majority when the overrides are zero, and rejects read + write <= total with .Non_Intersecting_Quorums (obligation B2). Validation builds a local candidate and assigns it only on success, so a failed call leaves the caller's value untouched. review_thousand_voters_reach_quorum exercises the binary-search path, while review_hundred_twenty_eight_voters verifies the acknowledgement bitmap across a 64-bit word boundary.

Volatile node state

Node holds the ledger plus volatile columns with two index domains. Phase one keeps, per recovery-chunk position, recovered_slot, recovered_ballot, recovered_state, and recovered_value (the greatest vote any acceptor reported for the decree), election: [MAX_MEMBERS]Election_Peer (what each peer said about the chunk), and promise_seen: [MAX_MEMBERS]Bit_Set(CHUNK_SLOTS). Phase two uses window-cell indices and keeps lead_slot, lead_ballot, acknowledgements: [WINDOW_SLOTS]Bit_Set(MAX_MEMBERS),

and `acknowledged: [WINDOW_SLOTS]u32`; the proposal itself is the leader's own vote in its ledger, so there is no separate proposal store. Ownership adds `own_next`, `highest_seen`, `stall_ticks`, and a bounded `resubmit` queue. `leader_hint` is `Maybe(Node_Id)`, and the remembered no-op is `noop: Maybe(Value)`; `maybe_resolve_chunk` returns `.Missing_Noop` if a campaign never supplied one.

Tagged unions

`Message(Value)` is a union of nine message structs (`Prepare_Message`, `Promise_Message`, `Promise_Range_Message`, `Accept_Message`, `Accepted_Message`, `Commit_Message`, `Learn_Message`, `Nack_Message`, `Heartbeat_Message`); `node_step` switches over it exhaustively. `Prepare_Message` carries a `scope`: `.Global` promises every decree from `first` on (the Multi-Paxos takeover) and `.Bounded` promises only `[first, last]`, recorded per decree. `Nack_Message.slot` names the decree whose accept was refused, or zero for a refused prepare. `Write(Value)` is a union of the five durable records. `Entry(Value, MAX_MEMBERS, MAX_METADATA_BYTES)` is a union of `Value` and `Stop_Sign`. `Host_Request` has one variant, `Serve_Range_Request`.

The Procedure Surface

`src/paxos.odin` declares one proc group per verb. `paxos.init` dispatches to `node_init`, `effects_init`, `membership_init`, `replicated_log_init`, `learner_init`, and `stop_sign_init` on the type of its first argument; `campaign`, `propose`, `propose_batch`, `step`, `tick`, `reconnected`, `request_catch_up`, `learn_chosen`, `set_campaign_enabled`, `advance_memory_floor`, and `install_chosen_trim` dispatch between `^Node` and `^Replicated_Log_Node` (and, for `learn_chosen`, `^Learner`). Query groups such as `role`, `ballot`, `decided_through`, `leader_base`, `proposal_frontier`, `committed_at`, `read_decided`, `memory_floor`, `trim_anchor`, and `ledger` follow the same rule. The receiver-prefixed long spellings (`node_propose`, `replicated_log_step_checked`, `learner_learn_chosen`) remain for call sites that want to name their receiver, and `log_*` aliases shorten the replicated-log spellings. POD 0005 records the reasoning behind this surface.

Multi-Paxos Behaviour

A campaign (`node_campaign`, or a follower whose `election_ticks` reached `election_timeout_ticks`) picks a round above every round it has seen, clears the election columns, records its own `Write_Promise`, and broadcasts `Prepare_Message{ballot, first = delivered_through + 1, last = first + CHUNK_SLOTS - 1}` with `.Global` scope. An acceptor promises durably (`Write_Promise`) before replying, answers with one `Promise_Message` per used cell in the chunk (reporting `vote`, `state`, and a pointer to the value) and one `Promise_Range_Message` describing the chunk, its trim anchor, and its `chosen_through`. Once a read quorum has fully described the chunk, `resolve_chunk` re-broadcasts known decisions, re-proposes the highest-ballot vote per decree (obligation B3), fills holes with the remembered no-op, and asks the most advanced peer for a `Learn_Message` if it is ahead. Chunks continue while any peer reported `more`; then `become_leader` sets `leader_base` and the node runs phase two only.

A leader's `node_propose` claims `next_slot` and `send_accept` records the leader's own vote (`ledger_record_vote`, `Write_Vote`), counts its own acknowledgement, and broadcasts `Accept_Message`. An acceptor's `on_accept` votes when the ballot is at least its effective promise for that decree and nacks otherwise (`Nack_Message.slot` names the decree). `on_accepted` commits at the write quorum; `record_commit` emits `Write_Chosen` and `emit_contiguous` releases every slot now contiguous with `delivered_through`. Heartbeats carry `decided_through`; a follower that receives a heartbeat above its promise adopts the ballot (writing `Write_Promise`) rather than starting a needless election, and requests a `Learn_Message` when the leader is ahead. `message_decided_through` extracts a peer's progress from every message kind that reports it so `resend_to` can skip slots the peer has already decided.

Under rotating ownership the same procedures run with different inputs: `propose_owned` calls `send_accept` at `ownership_ballot(node.id)` in the node's next own slot, `tick_ownership` issues no-op skips and detects stalls, and `start_revocation` begins a bounded phase one at a fresh round: its `.Bounded` prepare makes `promise_bounded` record `Write_Promise_At` per decree, and the same `on_promise`, `on_promise_range`, and `resolve_chunk` procedures resolve the chunk. POD 0010 specifies the mode.

Replicated Log and Learner

`Replicated_Log_Node` wraps a `Node(Entry(Value, MAX_MEMBERS, MAX_METADATA_BYTES), ...)` with `configuration_id`, `stop_sign: Maybe(Stop_Sign)`, `stop_slot`, and `stop_pending`. Commands and stop signs share one slot line; `replicated_log_propose_stop_sign` (aliased `replicated_log_reconfigure`) proposes a validated `Stop_Sign` and `replicated_log_is_sealed` is true while a stop sign is pending or decided, so proposals during that interval fail with `.Log_Sealed`. A pending seal clears if recovery replaces the unchosen stop-sign vote; a decided seal persists. `Log_Envelope` stamps outbound envelopes with the configuration id and `replicated_log_step_checked` refuses a mismatch before the core sees it. `replicated_log_ledger` exposes the wrapped ledger. POD 0006 specifies the reconfiguration protocol.

`Learner(Value, MAX_ENTRIES)` is a standalone non-voting window that stores values inline (`Learner_Cell{slot, value}`): `learner_learn_chosen(l, configuration_id, slot, value)` returns a `Learn_Result` (`Buffered`, `Advanced`, `Duplicate`) and an `Error`, refuses a foreign configuration with `.Configuration_Mismatch`, detects `.Conflicting_Chosen_Value`, and releases only the contiguous prefix through `released_through`. `learner_read_chosen` and `learner_chosen_at` read that prefix while it is still resident.

Error Contract

`Error` is one enum for the core, the log, and the learner, spelled in Ada_Case (`.Not_Leader`, `.Window_Full`, `.Configuration_Mismatch`). `explain_error(err)` returns a static string for every value: a banner title such as `"-- WINDOW FULL --"`, a blank line, a plain-language explanation, and a line beginning with `"Hint:"` that specifies the corrective action. The test `test_every_error_explains_problem_and_recovery` iterates the whole enum, so adding a value without an explanation fails the suite. No transition allocates or formats; the host calls `explain_error` at its own boundary.

Validation and Acceptance Gates

The evidence that exists in the repository today:

1. **Unit tests.** 79 procedures marked `@(test)` across `tests/` (`grep -c "@(test)" tests/*.odin`), run in both `-debug` and `-o:speed` builds. They cover ballots and quorums, the bit set, chunked recovery and retry progress, batches, inherited-prefix gating, trim identities, restoration from a replayed ledger, learner windows, sealing, the five `ownership_*` scenarios, and the repairs listed in POD 0007 (the `review_*` tests). `tests/harness.odin` supplies the journal and packet fixtures shared across all tests.
2. **Election matrix.** `election_matrix_preserves_chosen_values` enumerates every assignment of no vote / ballot 1 / ballot 2 to three voters, all six intersecting quorum pairs, and all six first-response orders, asserting exactly 972 cases and that any value chosen by an earlier write quorum survives.
3. **Seeded simulator.** `sim/simulation.odin` drives one, three, or five nodes under drops, duplication, link cuts, crashes, and restarts from a replayed journal, in both the single-leader and the `--ownership` mode. Crashes land at `Before_Writes`, `Partial_Writes`, or `Partial_Messages` inside the host commit sequence, and pre-durable accepts leave early. Oracles run after every transition: agreement against a golden log, validity, promise regression, vote-below-promise, one value per ballot and slot, contiguous release, and after quiescence liveness (a fresh decision) and convergence. `tools/check.py` runs `--`

`seeds` seeds (default 20) for each node count in each mode: 60 single-leader and 60 ownership runs, plus 120 focused small-window/chunk-3 runs covering majority and flexible quorums. The archived extended run used 100 seeds and completed 720 simulations (7.2 million steps).

4. **Reconfiguration scenarios.** Four seeded scenarios in `tests/test_reconfiguration_sim.odin`, each over 16 seeds, check seal agreement, nothing released past the seal, replay keeps the seal, and the next configuration decides on the same slot line; the fourth runs under rotating ownership and requires decisions other owners reach above the stop sign to be abandoned.
5. **Contract fixtures.** `tools/check_contracts.py` compiles nine programs that the compiler must reject and builds four durability fixtures in both profiles, asserting the named diagnostic.
6. **One entry point.** `tools/check.py` runs `tools/check_style.py` (the Zen constraints of POD 0001), `odin check -vet -strict-style` on every package and the example, both test profiles, the contracts, the simulations, the counter example, the benchmark JSON schema (eleven result rows), and CLI failure propagation in a temporary directory so a stale binary cannot mask a failure.

This is finite executable evidence. No refinement proof or coverage percentage is claimed; POD 0008 states the safety argument as axioms, lemmas, and proof obligations discharged by procedures.

References

- Lamport, Leslie. "The Part-Time Parliament." ACM TOCS, 1998.
- Lamport, Leslie. "Paxos Made Simple." ACM SIGACT News, 2001.
- Mao, Yanhua, Junqueira, Flavio P., and Marzullo, Keith. "Mencius: Building Efficient Replicated State Machines for WANs." OSDI, 2008.
- POD 0003: Durability Contracts, Window Reuse, and Trim Anchors.
- POD 0005: The Idiomatic Odin API Surface.
- POD 0006: Reconfiguration and Epoch Isolation.
- POD 0007: Review Findings and Verification Evidence.
- POD 0008: Safety Argument: Axioms, Lemmas, and Proof Obligations.
- POD 0009: The Data-Oriented Ledger.
- POD 0010: Rotating Slot Ownership.