POD 0008: Safety Argument: Axioms, Lemmas, and Proof Obligations

COMMITTED

CATEGORY	**STATUS**
Design Record	Committed
AUTHORS	**CREATED**
Vikrant Rathore <vikrant@insan.ai>, Paxos Odin Contributors	2026-09-16
LAST UPDATED	**LABELS**
2026-09-17	mathematics, safety, protocol

Abstract

This record fixes the model `paxos-odin` is argued against, the definitions the argument uses, and the theorem and lemmas that connect Lamport's conditions B1, B2 and B3 to the procedures in `src/`. It is the reference the book's chapter "The Safety Argument" expands with full proofs; here proofs are abbreviated and each lemma points at the code that discharges its premise. The record closes with the obligation-to-code map, the list of what is **not** proved, and two key insights surfaced by the analysis: the pre-durable Accept exception requires the candidate's own promise to be durable, and a stop sign seals at release rather than at choice, meaning any value chosen by the core above the seal is abandoned rather than prevented.

Status and Implementation Boundary

This is a committed paper argument over the implementation, not a machine-checked proof. The agreement argument concerns durable votes and decisions released only after their required writes are confirmed. Its induction is over decision events, including repeated events at one node, rather than over the set of nodes.

For B3, a complete read quorum is necessary before selection. Additional valid reports may contribute to the selected value. `recovery_ready` then freezes that selection before any phase-two proposal; retries reuse it. A later losing vote cannot contradict a known decision, whereas two reported decisions with different values are an error. Chunk-relative indexing must preserve this evidence until resolution finishes; POD 0009 records the boundary and retry tests.

Axioms

- **A1 (Processes).** A configuration has a fixed, finite membership. A process runs one transition at a time, may crash at any instant, and restarts with exactly the records it persisted: every field of `Node` outside `Ledger` is volatile and is rebuilt by `node_restore`.
- **A2 (Channels).** Envelopes may be lost, duplicated, reordered, and delayed without bound, but are never forged or corrupted. `node_step` refuses `.Not_Member` senders and `.Wrong_Recipient` envelopes; the host authenticates the transport.

- **A3 (Quorums).** A read quorum Q_1 is any `read_quorum_size` members, a write quorum Q_2 any `write_quorum_size` members, and `read_quorum_size + write_quorum_size > count`; `membership_init` refuses anything else with `.Non_Intersecting_Quorums`, and `replicated_log_init_from_stop` builds the next configuration through the same procedure.
- **A4 (Durability).** The host persists every `Write` of a transition in order, syncs, and calls `confirm_writes_durable` before any message of the transition leaves (except as Lemma 10 allows) and before the next transition runs; `effects_messages_slice` and `effects_reset` stop the process otherwise under `Durability_Gate.Enforced`. Confirmed records survive crashes. Restart folds the journal in order through `ledger_replay_fold` or `ledger_apply` into `node_restore` with the host's durably consumed floor, which never decreases. A `Committed` entry is applied only after its batch is confirmed.
- **A5 (No Byzantine behaviour).** Every process runs the library's code on its own ledger, never confirms a failed write, never edits a ledger by hand, and reports its state truthfully.

Definitions

- **D1 Ballot.** `Ballot :: distinct u64`, built by `ballot_make(round, priority, node)` as `round << 24 | priority << 16 | node`. The fields occupy disjoint bit ranges, so integer order is lexicographic order on $(\text{round}, \text{priority}, \text{node})$ and distinct triples are distinct ballots. The proposer of b is `ballot_node(b)`. A **campaign ballot** has round at least one; the **round-zero ballot** of member n is `ownership_ballot(n)`.
- **D2 Vote.** Acceptor a votes (b, v) in decree s when it durably records `Write_Vote{ballot = b, slot = s, value = v}`; the cell then holds `vote_ballot = b`, `value = v`, state `.Voted` (or `.Chosen`, which keeps the vote).
- **D3 Promise.** `promised` is the global promise (Lamport's maxBal; `Write_Promise`), `promised_at[c]` the per-decree promise (`Write_Promise_At`, or implied by a vote). The effective promise is $p_a(s) = \max(\text{promised}_a, \text{promised\_at}_a[s])$, that is `ledger_promise_for`.
- **D4 Chosen.** v is chosen in s at b when some Q_2 exists all of whose members voted (b, v) in s.
- **D5 Decided.** A node decided s when `record_commit` recorded `Write_Chosen{slot = s, value = v}`; the cell is `.Chosen`.
- **D6 Applied.** The host consumed the `Committed` entry for s from `committed_slice`; `delivered_through` is the greatest released slot, and `advance_memory_floor` reports what the host durably consumed.

Theorem and Lemmas

Fix one decree s unless stated otherwise. Full proofs are in the book chapter; each entry here gives the statement, the procedures that discharge it, and the shape of the argument.

The Synod theorem

- **Lemma 1 (quorum intersection).** $Q_1 \cap Q_2 \neq \varnothing$. From $|Q_1| + |Q_2| > N$, enforced by `membership_init`.
- **Lemma 2 (a vote respects the promise).** A vote at b needs $b \geq p_a(s)$ and leaves $p_a(s) \geq b$; promises never decrease. `on_accept` nacks below `promised` or `promised_at`; `send_accept` refuses below `ledger_promise_for`; `on_prepare`, `on_heartbeat`, `promise_bounded` only raise; `ledger_apply` refuses `.Promise_Regression`; `ledger_replay_fold` folds with `max`.
- **Lemma 3 (one value per ballot).** At most one value is ever carried by an Accept for (b, s). Campaign ballots: `ballot_node` identifies the proposer, `start_campaign` and `start_revocation` take `greatest + 1` over `ledger_highest_ballot`, `send_accept` refuses a second value at a driven cell's ballot, `become_leader` puts `next_slot` above every used slot. Round-zero ballots: `on_accept` accepts round zero only from `owner_of(slot)` under `node.ownership`; the owner proposes once per own slot. Acceptors: `on_accept` and `ledger_apply` answer a second value at one ballot with `.Conflicting_Value`.
- **Lemma 4 (B3).** An Accept at a campaign ballot is issued only after a read quorum of complete reports (`maybe_resolve_chunk`, `on_promise_range`, `on_promise`), each member having promised the ballot before

reporting every used cell in the chunk (`on_prepare`, `promise_bounded`); `resolve_chunk` proposes the greatest reported vote, records a reported decision, or proposes the no-op for a hole. Fresh values from `node_propose` sit above every slot any member of the final chunk's quorum reported (Lemma 6).

- **Theorem 1 (Agreement).** If v is chosen at b and w at b' in s, then $v = w$. Let b_0 be the least ballot at which anything is chosen and v_0 its value. Every Accept in s at a ballot above b_0 carries v_0, by induction along the happens-before order of Accepts: the read quorum of the Accept's ballot meets the write quorum of b_0 (Lemma 1) in an acceptor a; a voted (b_0, v_0) before it promised (Lemma 2), so it reported a vote at a ballot at least b_0 (its cell is never overwritten downward, and cleared only when the slot is fenced, Lemma 8); the greatest reported vote is at b_0 (then v_0 by Lemma 3) or above it (then produced by an earlier Accept, which carried v_0 by hypothesis); Lemma 4 makes the new Accept carry it. The induction runs along time rather than ballots because a vote raises only `promised_at`, so a `.Global` prepare can collect a vote at a ballot above the candidate's; that is still safe.
- **Corollary 1 (decided implies chosen).** `record_commit` runs from `on_accepted` only at `membership_write_quorum` distinct acknowledgements of the driven ballot (`bit_set_insert` on `acknowledgements[cell]`) with the node's own vote still present (`.Missing_Proposed_Value` otherwise); from `send_accept` only when the write quorum is one; from `on_commit`, `node_learn_chosen` and `resolve_chunk` for a value another node had decided (induction over prior decision events, with host-certified learning covered by the host axioms). The implication applies after the required writes are durable: a quorum-one local decision produced within a transition is not externally released before its local vote is persisted.

The multi-decree log

- **Lemma 5 (independent decrees).** A cell belongs to the slot in `slot[cell]`; every read checks the tag (`ledger_cell`, `ledger_vote_at`, `ledger_chosen_at`, `claim_live`), and `ledger_open` clears promise, vote and state as it retags. Only `promised` is shared, and a shared promise can only refuse more.
- **Lemma 6 (one campaign covers all decrees above its base).** A `.Global` promise covers every decree; each chunk has its own complete read quorum (`begin_next_chunk` resets `Election_Peer` except the fences and clears `promise_seen`); the final chunk's quorum reported no vote above `last` (`more` false), so fresh proposals above it satisfy Lemma 4 (ii).
- **Lemma 7 (holes).** The no-op is proposed through `send_accept` like any value; `maybe_resolve_chunk` refuses `.Missing_Noop` without one. A hole has no vote in the chunk's read quorum, so nothing was chosen there at a lower ballot.
- **Lemma 8 (fences and the memory window).** A vote leaves a ledger only for a slot at or below the node's delivered prefix or trim anchor (`claim_live` requires `held <= memory_floor` and `.Chosen`; `node_resume_at` clears open votes at or below the restart base; `ledger_claim` reuses `.Chosen` or trimmed cells on replay). Both bounds travel in `Promise_Range_Message` as `chosen_through` and `anchor`; `quorum_fences` takes their maximum and `resolve_chunk` starts at `fence + 1`, `become_leader` above it. Fenced slots are chosen by Corollary 1 or by the definition of `Trim_Anchor`, so the fence forgoes only forced proposals. The fence is required for safety, not only for economy.

Durability

- **Lemma 9 (indelible ink).** `on_prepare`, `promise_bounded`, `on_heartbeat`, `on_accept` and `send_accept` add their write before their message; under `.Enforced`, `effects_messages_slice` and `effects_reset` call `host_order_violation` while `writes_pending`. A restarted ledger holds every promise and vote the node revealed, so Lemma 2 holds across restarts.
- **Lemma 10 (the pre-durable exception).** `pre_durable_next` yields only Accepts with round at least one. Sending them before the barrier loses at most the proposer's own vote, which cannot un-choose anything and is counted toward a quorum only in a later transition. Lemma 3 survives because the

restarted proposer campaigns at a higher round: `start_campaign` writes `Write_Promise` for its own ballot, and `start_revocation` runs `promise_bounded` on itself, in the same batch as the `Prepare`, so the ledger always holds a record at the round of every ballot the proposer announced, and `ledger_highest_ballot` lifts the next campaign above it. Round-zero Accepts are withheld because an owner's ballot is the same integer in every own slot and `next_usable_own_slot` would reissue the slot after a crash; the simulator's vote oracle ("ballot accepted two values in slot") found that counter-example.

Rotating ownership

- **Lemma 11 (ballot partition).** `owner_of` is a function of the slot and the membership sorted by ascending id; `on_accept` accepts round zero only from that owner; campaign and revocation rounds are at least one and occupy the top 40 bits, so the owner's ballot is the least in the decree and B1 holds per decree.
- **Lemma 12 (a revocation is a phase one).** `start_revocation` sends a `.Bounded` prepare; `promise_bounded` records `Write_Promise_At` for every slot of the range or answers nothing at all; `resolve_chunk` runs with `drive_all`, re-proposing recovered round-zero votes (B3) and filling holes; the owner is fenced by `promised_at` in `on_accept` and `next_usable_own_slot`; `become_leader` returns the revoker to `.Follower`.
- **Lemma 13 (skips).** `skip_idle_slots` calls `propose_owned` with the no-op, at most `min(C, SKIP_BURST)` per tick.
- **Lemma 14 (resubmission).** `on_accept` (when a higher ballot overwrites the cell's own round-zero vote with a different value) and `record_commit` (when the decided value differs from it) queue the owner's losing suggestion, keyed on the vote's ballot alone so a revocation the owner itself started cannot hide it; `drain_resubmits` proposes it in a later own slot through `propose_owned`. Best effort: the queue holds one chunk; overflow increments `resubmits_dropped` and requires host retry. A suggestion re-proposed by the revoker may decide twice, so commands need application-level deduplication.
- **Liveness (informal).** `tick_ownership` counts `stall_ticks` while `delivered_through < highest_seen`, asks the stuck owner every `heartbeat_interval_ticks`, and revokes at `election_timeout_ticks`; a timed-out revoker revokes again higher. Not proved.

Stop signs and contiguity

- **Lemma 15 (sealing).** (i) `replicated_log_propose`, `replicated_log_propose_batch` and `replicated_log_propose_stop_sign` return `.Log_Sealed` while `replicated_log_is_sealed`; `replicated_log_recalculate_stop_pending` scans every used cell after every transition and after restore, so a vote for a stop sign seals as surely as a decision. (ii) A node that has decided the stop slot never releases, reports or reads a slot above it: `replicated_log_observe_effects` records the seal (`replicated_log_observe_stop`) and then `replicated_log_abandon_above_seal` cuts `effects.committed` at the first slot above `stop_slot`; releases are contiguous (Lemma 16), so the seal is on record before the cut; `replicated_log_observe_durable` rediscovers it on restore; `replicated_log_decided_through`, `replicated_log_read` and `replicated_log_read_decided` stop at the seal. A value the core chooses above the seal is abandoned, never prevented. (iii) `replicated_log_init_from_stop` continues at `stop_slot + 1` through `replicated_log_continue_at`; `replicated_log_step_checked` rejects `.Configuration_Mismatch` with no write and no message.
- **Lemma 16 (contiguity).** `emit_contiguous` releases only `delivered_through + 1` while the cell is tagged and `.Chosen`; the pass-through in `record_commit` releases exactly `delivered_through + 1`, at most once per transition (record and entry share `pass_through`); every released value is chosen. `claim_live` admits only slots in `(memory_floor, memory_floor + W]`, so cells and live slots are in bijection. `learner_learn_chosen` does the same with `released_through`.

Obligation-to-Code Map

Obligation	Procedures	Tests and oracles
B1 (Lemmas 3, 11)	`ballot_make`, `start_campaign`, `start_revocation`, `ownership_ballot`, `on_accept`, `send_accept`	`test_ballot_ordering`, `ownership_three_owners_propose_concurrently`, simulator vote oracle
B2 (Lemma 1)	`membership_init`	`test_membership_validation`, `review_negative_quorums_and_learner_campaign`, `review_flexible_quorums_and_window_reuse`
B3 (Lemmas 4, 6, 7, 12)	`on_prepare`, `on_promise`, `on_promise_range`, `maybe_resolve_chunk`, `resolve_chunk`, `promise_bounded`	`test_lamport_b3_max_vote_rule`, `election_matrix_preserves_chosen_values`, `ownership_revocation_keeps_a_seen_vote`, `review_multichunk_recovery_and_retry_progress`, simulator `AGREEMENT`, `VALIDITY`
Votes respect promises (Lemma 2)	`on_accept`, `send_accept`, `ledger_apply`, `ledger_replay_fold`	simulator `PROMISE REGRESSION`, `VOTE BELOW PROMISE`; `node_assert_valid`
Decided implies chosen (Corollary 1)	`on_accepted`, `record_commit`	`review_duplicate_acknowledgements_do_not_make_a_quorum`, `review_missing_proposal_is_error`, `test_three_node_cluster_agreement`
Window and fences (Lemmas 5, 8)	`claim_live`, `ledger_claim`, `quorum_fences`, `resolve_chunk`, `become_leader`	`review_recovery_preserves_fences_across_chunks`, `review_snapshot_preserves_votes_above_anchor`, `review_replay_reuses_certified_trimmed_vote`
D1 (Lemmas 9, 10)	`effects_messages_slice`, `effects_reset`, `effects_confirm_writes_durable`, `pre_durable_next`, `node_restore`	`test_effects_power_loss_barrier_flag`, `test_pre_durable_messages_iterator`, `test_host_managed_gate`, `test_node_restore_and_recovery`, simulator crash points, `CONVERGENCE`
Ownership (Lemmas 11 to 14)	`owner_of`, `propose_owned`, `skip_idle_slots`, `start_revocation`, `queue_resubmit`, `drain_resubmits`	`ownership_idle_owners_skip`, `ownership_revokes_a_crashed_owner`, `ownership_revoked_suggestion_is_resubmitted`, simulator `--ownership`
S1 (Lemma 15)	`replicated_log_is_sealed`, `replicated_log_propose`, `replicated_log_abandon_above_seal`, `replicated_log_step_checked`, `replicated_log_init_from_stop`	`test_replicated_log_stop_sign_seals_epoch`, `review_stop_seal_restore_and_completed_history`, `reconfiguration_cluster_handover_rejects_old_epoch`, `review_out_of_order_chosen_stop_blocks_proposals`, the `seal_expect_nothing_after` oracle of every `reconfiguration_sim_*` scenario
L1 (Lemma 16)	`emit_contiguous`, `record_commit`, `learner_learn_chosen`	`test_learner_contiguous_release`, `review_leader_fetches_decisions_from_ahead_follower`, simulator `CONTIGUITY`

What Is Not Proved

- **No mechanised model.** The argument is a paper proof over the code as read; there is no model checker, proof assistant, or refinement mapping from the procedures to an abstract specification. The

simulator (`sim/simulation.odin`) and the tests are evidence, not proof: they check the oracles on the executions they generate.

- **Liveness.** Election, heartbeat, retransmission, stall detection, and revocation are described, not proved. Dueling candidates and dueling revokers are possible; `priority` and the host's timers are the only remedies, and none is a safety argument.
- **Host-side obligations.** A4 and A5 are assumed, not enforced beyond the `Enforced` gate's two checks. In particular the host must: replay its whole journal in order; pass a monotone consumed floor to `node_restore`; apply released entries idempotently across a restart; serve `Serve_Range_Request` from a state image that agrees with the trimmed log; bind `trim_id` to that image; and never confirm a failed write.
- **The pre-durable exception (Lemma 10).** The exception is sound only because the restarted proposer's ledger holds a record at the round of the in-flight ballot, and the library discharges that itself: `start_campaign` and `start_revocation` promise their own ballot in the batch that carries the `Prepare`. An earlier draft of this record left it to the host to deliver a candidate's self-addressed `Prepare` before any peer's reply; that rule is no longer needed, and a host must not rely on message order for safety.
- **Chosen above the seal (Lemma 15).** The core can still choose a value above the stop slot in the sealed configuration: `resolve_chunk` re-drives a reported vote with no seal check, and under rotating ownership an owner that has not learned of the seal can have a suggestion decided in its own slot (`reconfiguration_sim_ownership_abandons_decisions_above_the_seal`). The log abandons such a decision: it is never released and `replicated_log_read`, `replicated_log_read_decided` and `replicated_log_decided_through` hide it. Nothing is proved about an abandoned decision beyond that. A client value abandoned this way is not reported to the host and must be proposed again in the next configuration, and a host that inspects the core ledger through `replicated_log_ledger` sees the decision and must not act on it.
- **Learners.** `node_learn_chosen` and `learner_learn_chosen` trust the host's certification that a value is chosen for the named configuration; the argument covers voting members.

References

- Lamport, Leslie. "The Part-Time Parliament." ACM Transactions on Computer Systems 16(2), 1998. Conditions B1(B), B2(B), B3(B) in section 2.2; Theorem 1 (consistency); the basic protocol in section 2.3; the multi-decree parliament in section 3.
- Lamport, Leslie. "Paxos Made Simple." ACM SIGACT News 32(4), 2001. P1, P2, P2a, P2b, P2c.
- Mao, Yanhua, Junqueira, Flavio P., and Marzullo, Keith. "Mencius: Building Efficient Replicated State Machines for WANs." OSDI 2008. Coordinators, suggest, skip, revoke; coordinated Paxos.
- Lamport, Leslie, Malkhi, Dahlia, and Zhou, Lidong. "Reconfiguring a State Machine." ACM SIGACT News 41(1), 2010.
- Howard, Heidi, Malkhi, Dahlia, and Spiegelman, Alexander. "Flexible Paxos: Quorum Intersection Revisited." OPODIS 2016.
- POD 0003: Durability Contracts, Window Reuse, and Trim Anchors.
- POD 0006: Reconfiguration and Epoch Isolation.
- POD 0007: Review Findings and Verification Evidence.