POD 0010: Rotating Slot Ownership

COMMITTED

CATEGORY	**STATUS**
Protocol Specification	Committed
AUTHORS	**CREATED**
Vikrant Rathore <vikrant@insan.ai>, Paxos Odin Contributors	2026-09-16
LAST UPDATED	**LABELS**
2026-09-17	consensus, ownership, throughput

Abstract

`paxos-odin` can run its log with every voting member proposing at once. Under the `Node_Options.rotating_ownership` option, slot s belongs to the member at index $(s-1) \bmod N$ of the membership, the owner proposes there at a reserved round-zero ballot with no phase one, an idle owner fills its slots with the host's no-op, a stalled prefix is repaired by a bounded, per-decree phase one that fences the suspected owner out of the stalled slots only, and a suggestion that lost to such a revocation is queued for best-effort resubmission in a later own slot. This record specifies the rules, the messages and ledger records they touch (`Prepare_Scope`, `Nack_Message.slot`, `Write_Promise_At`), the state added to `Node`, each procedure in `src/ownership.odin` and its hooks in `src/consensus.odin` and `src/election.odin`, the safety argument (B1 per decree), and what `tests/test_ownership.odin` and the simulator's `--ownership` mode check. The design follows Mencius (Mao, Junqueira, Marzullo, 2008).

Status and Implementation Boundary

Rotating slot ownership is implemented in `src/ownership.odin` and enabled via `Node_Options.rotating_ownership`. The subsystem reuses the core acceptor, learner, and recovery machinery while establishing a round-zero suggestion channel, bounded range revocations, and best-effort resubmissions.

Execution safety is verified through unit tests in `tests/test_ownership.odin`, batch regression coverage in `tests/test_batch_review.odin`, and randomized schedules in `paxos-sim --ownership`. Multi-decree safety conditions hold per slot; clock-based leases and dynamic ownership reweighting remain future extensions.

Introduction

The single stable leader of Multi-Paxos makes every decision one round trip, but every proposal passes through one node. A member that receives a client request forwards it to the leader before the round trip begins, and the leader's link, disk, and CPU bound the group's throughput. When members are spread across sites, the forwarding hop is a wide-area latency on every request that did not originate at the leader.

Mencius observed that Paxos does not require one proposer per log, only one proposer per ballot per decree. If the decrees are dealt out round-robin and each coordinator uses a ballot that is, in its own decrees, the lowest possible, the coordinator can propose without a prepare, and the group's proposing capacity scales with its size. The cost is that the log now depends on every member: an idle member must

skip its instances and a crashed one must be revoked. This record adopts that design on the existing `Node`, keeping the acceptor, the learner, and the phase-one machinery intact and adding a second ballot regime alongside campaigns.

Terminology and Scope

- **Owner** of slot s: `owner_of(node, s)`, the member at index $(s-1) \bmod N$ of the membership in ascending id order established by `membership_init`.
- **Ownership ballot**: `ownership_ballot(owner) = ballot_make(0, 0, owner)`, round zero, priority zero.
- **Suggestion**: an `Accept_Message` at an ownership ballot in an own slot, sent by `propose_owned`. Mencius calls it SUGGEST.
- **Skip**: a suggestion of the host's no-op in an own slot below `highest_seen`, sent by `skip_idle_slots`.
- **Revocation**: a phase one at round one or above with `Prepare_Scope.Bounded`, started by `start_revocation`, that promises per decree over a stalled range and drives every slot in it.
- **Resubmission**: `propose_owned` of a value whose suggestion was decided away by a revocation, queued by `queue_resubmit` and sent by `drain_resubmits`.

In scope: everything in `src/ownership.odin` and the branches on `node.ownership` in `src/node.odin`, `src/consensus.odin`, `src/election.odin`, and `src/effects.odin`. Out of scope: reads, leases, and weighted or adaptive ownership, listed under open questions.

Design Overview

Six rules define the mode.

1. **Ownership.** Slot s is owned by `membership_get(membership, (s - 1) mod N)`. Ownership is a pure function of the membership and the slot; no message carries it.
2. **Ballot partition.** Round zero of the 40-bit round field is reserved. `ownership_ballot(owner)` is the only round-zero ballot an acceptor votes at in the owner's slots; `start_campaign` and `start_revocation` always choose a round of one or more. Within each decree the owner's ballot is therefore unique and the least ballot any acceptor votes at, so B1 holds per decree and B3 leaves the owner free to choose any value.
3. **Suggest.** An owner proposes in its next usable own slot through `send_accept` at its ownership ballot, with no prepare. Decided or revoked own slots are stepped over.
4. **Skip.** On every tick an owner suggests the no-op in each own slot at or below `highest_seen`, at most `min(CHUNK_SLOTS, SKIP_BURST)` per tick.
5. **Revoke.** A member whose `delivered_through` stays below `highest_seen` for `election_timeout_ticks` runs a bounded phase one over `[delivered_through + 1, min(chunk end, highest_seen)]`, promises are recorded per decree, every slot in the range is driven (a recovered vote by B3, otherwise the no-op), and the revoker returns to `.Follower` when the range has been driven; individual decisions may still be pending.
6. **Resubmit.** When a decision arrives for a slot this node suggested in at its ownership ballot and the decided value differs, the suggested value is queued for best-effort proposal in a later own slot.

`campaign` is refused with `.Campaign_Disabled`, and `proposal_gate` admits any voting member regardless of role.

Detailed Design

Messages and Records

- `Prepare_Scope :: enum u8 { Global, Bounded }` and `Prepare_Message.scope`. `.Global`, the zero value, is the Multi-Paxos takeover: promise every decree from `first` on and raise the ledger's single `promised`

ballot. `.Bounded` promises only `[first, last]`, one decree at a time. `start_campaign` leaves the scope at its zero value; `start_revocation` sets `.Bounded`, and `begin_next_chunk` sets `.Bounded` if and only if `node.ownership`.

- `Nack_Message.slot`. A refused accept names its decree; a refused prepare or heartbeat carries zero. `on_nack` reads only the two ballots (it raises `highest_observed_round` and steps a candidate down when `rejected == node.ballot`); the slot is there so a host or observer can see which decree fenced an owner.
- `Write_Promise_At{ballot, slot}`. The durable record of a per-decree promise. `ledger_apply` refuses a regression with `.Promise_Regression`; `ledger_replay_fold` keeps the maximum. `effects_requires_power_loss_barrier` treats it like `Write_Promise` and `Write_Vote`: it needs the full barrier.
- `Ledger.promised_at[cell]`. The per-decree promise; `ledger_promise_for(l, cell)` is `max(l.promised, l.promised_at[cell])` and is the promise `on_accept`, `send_accept`, and `next_usable_own_slot` compare against. `ledger_highest_ballot` folds it in, so a revocation's round always exceeds every per-decree promise in the ledger.

Node State

Ownership uses these fields: `ownership: bool` (copied from `Node_Options.rotating_ownership`), `own_next: Slot` (the next own slot to consider; seeded by `own_slot_from(node, 1)` in `node_init` and recomputed from `next_slot` in `node_resume_at`), `highest_seen: Slot` (the greatest slot suggested, accepted, or decided at this node, raised in `propose_owned`, `on_accept` after the ownership check, `record_commit`, and `node_resume_at`), `stall_ticks: u32`, `resubmit: small_array.Small_Array(CHUNK_SLOTS, Value)`, and the saturating `resubmits_dropped` counter. The existing `recover_base`, `recover_last`, `lead_slot`, `lead_ballot`, and `acknowledgements` are reused: `recover_last` bounds a revocation, and `lead_ballot` keys acknowledgement counting and retransmission to the ballot this node is driving in each slot, which under ownership is either its ownership ballot or its revocation ballot.

Procedures

- `owner_of(node, slot)` and `ownership_ballot(owner)`: public, pure, inlined.
- `own_slot_from(node, from)`: the first own slot at or after `from`, from the member's index and N.
- `next_usable_own_slot(node)`: from `own_next`, returns `.Global_Slot_Exhausted` at `max(Slot)` and `.Window_Full` when the slot is more than `WINDOW_SLOTS` above `memory_floor`; a held cell is usable only if it is not `.Chosen` and `ledger_promise_for` is at most the ownership ballot; otherwise `own_next` advances to the next own slot and the loop continues.
- `own_slots_available(node, wanted)`: probes every target with read-only `own_slot_probe`, stepping over chosen or revoked own slots. Admission runs on the ownership frontier before single-leader slot arithmetic. A rejected batch changes neither votes nor the frontier; `next_usable_own_slot` uses the same probe during execution.
- `propose_owned(node, value, effects)`: `next_usable_own_slot`, then `send_accept(node, slot, ownership_ballot(node.id), value, effects)`, then `own_next = own_slot_from(node, slot + 1)` and `highest_seen = max(highest_seen, slot)`. `node_propose` routes here when `node.ownership`.
- `send_accept(node, slot, ballot, value, effects)`: the phase-two sender, now with the ballot as a parameter. It returns `.Conflicting_Value` if this node is already driving the slot at the same ballot with a different value, records the local vote and `Write_Vote`, sets `lead_slot`, `lead_ballot`, and clears the acknowledgement set, commits immediately when the write quorum is one, and broadcasts.
- `on_accept`: before any promise check, a round-zero accept is dropped unless `node.ownership` and `ballot_node(msg.ballot) == owner_of(node, msg.slot)`. Then `highest_seen` is raised, the global and per-decree promises are checked (each refusal is a `Nack_Message` with the slot), the vote is recorded

with `Write_Vote`, and `observe_leader` runs only for round one and above, because a suggestion says nothing about leadership.

- `on_accepted`: counts an acknowledgement only if `lead_slot[cell] == msg.slot` and `lead_ballot[cell] == msg.ballot`.
- `record_commit`: raises `highest_seen`; before recording a new decision, if `node.ownership`, the cell is `.Voted` at `ownership_ballot(node.id)`, and the decided value differs, calls `queue_resubmit` with the suggested value.
- `SKIP_BURST :: 8` and `skip_idle_slots(node, noop, effects)`: while `own_next <= highest_seen` and fewer than `min(CHUNK_SLOTS, SKIP_BURST)` skips have left, `propose_owned(noop)`; `.Window_Full` ends the loop quietly.
- `start_revocation(node, noop, effects)`: selects a round above every observed and durable ballot, bounds the range by chunk, highest seen slot, and memory window, then calls `promise_bounded` locally before changing election state. If admission fails, the node stays a follower and resets its stall counter. Success stores the no-op, resets election state and broadcasts the bounded Prepare with its local promises in the same effects batch.
- `on_prepare` with `.Bounded`: `promise_bounded` must succeed or the acceptor sends nothing; on success `highest_observed_round` is raised, every used cell in the range is reported with `Promise_Message`, and `Promise_Range_Message` closes the answer. `observe_leader` is not called.
- `promise_bounded(node, msg, effects)`: returns false if `last - first >= CHUNK_SLOTS`, if any slot above `memory_floor` in the range has no live cell (`claim_live` fails), or if any such slot is promised above `msg.ballot`. Otherwise writes `Write_Promise_At` for each slot whose `promised_at` is not already the ballot.
- `on_promise`, `on_promise_range`, `maybe_resolve_chunk`: collect chunk-relative reports; selection freezes before phase two and remains fixed across retries. `resolve_chunk` sets `drive_all = any_more || node.ownership`, so a revocation drives every slot up to `recover_last`; a slot with a recovered `.Voted` cell is re-proposed at the revocation ballot, an empty one gets the no-op, a known decision is rebroadcast. `begin_next_chunk` under ownership caps the next `recover_last` at `max(highest_seen, recover_base)`.
- `become_leader` under ownership: role `.Follower`, `stall_ticks = 0`, `emit_contiguous`; no `leader_base`, no `leader_hint`.
- `queue_resubmit(node, value)`: `push_back` into `resubmit`; a value beyond the capacity is counted in `resubmits_dropped` (query `node_resubmits_dropped`) and left to the host's retry: resubmission is best effort. `drain_resubmits(node, effects)`: `propose_owned` each queued value in order, returning on `.Window_Full` with the rest still queued.
- `tick_ownership(node, noop, effects)`: while preparing, retries frozen recovery or starts a higher revocation on timeout. Otherwise resubmissions and skips share a budget of at most `CHUNK_SLOTS` proposals. Retransmission and a new revocation run only on a tick with no proposals, keeping effects within capacity. Stall accounting and learn requests track the unreleased prefix.
- `resend_to(node, peer, effects)`: unchanged, and now run by every member. It resends an `Accept_Message` for a `.Voted` cell only when `lead_slot[cell] == slot` and the vote's ballot equals `lead_ballot[cell]`, so a member resends its own suggestions and its own revocation accepts, never votes it merely cast.
- `node_campaign`: returns `.Campaign_Disabled` when `node.ownership`.
- `pre_durable_next`: yields only accepts with `ballot_round(accept.ballot) > 0`.

Security and Correctness Considerations

B1 per decree. Lamport's B1 requires distinct ballots within one decree. The ownership ballot `(0, 0, owner)` recurs across the owner's decrees, which B1 permits, and within each decree it is the only round-zero ballot any acceptor votes at, because `on_accept` drops every other round-zero accept before

consulting its promise. `send_accept` refuses to reuse a ballot with a second value in a slot this node is driving (`.Conflicting_Value`), and `on_accept` refuses to vote a second value at a ballot it has already voted (`.Conflicting_Value`). Campaign and revocation ballots are compared as integers exactly as before, and their round is always above zero, so the owner's ballot is the least ballot in each of its decrees and B3 constrains it to nothing. The Synod proof therefore applies unchanged.

Why revocation is phase one. A revoker proposes in another member's slot, at round one or above. Doing so without a prepare would let the owner's round-zero vote and the revoker's vote form two quorums with different values. A bounded prepare collects a read quorum of per-decree promises, which fences the owner's ballot out of those decrees and reports every vote in them, so `resolve_chunk` can apply B3. Promising per decree rather than globally is what keeps the owner usable in every slot outside the range; promising and driving the whole range is what keeps a promised slot from being fenced without ever being decided. `promise_bounded` fails closed rather than partially, so a candidate never counts a promise an acceptor did not record.

The pre-durable exception. A campaign accept may leave before the sender's writes are durable because a restarted proposer campaigns at a fresh round. An owner's suggestion has no fresh round: after a crash before `Write_Vote` is durable, `node_resume_at` finds the slot unused and the owner would suggest a different value at the same ballot in the same decree. `pre_durable_next` therefore never yields a round-zero accept; the host sends suggestions only after `confirm_writes_durable`. The simulator's `persist_sim_write` oracle fails a run when two members hold different values under one ballot in one slot, which is the observable form of this violation.

Operational Considerations

- **Skip burst.** `SKIP_BURST` is 8; the per-tick bound is `min(CHUNK_SLOTS, SKIP_BURST)`. An owner that was partitioned while the others advanced k slots needs about $\frac{k}{N \cdot 8}$ ticks to catch up its own slots; until then the prefix is held at its first unfilled slot.
- **Stall timeout.** The revocation timeout is `election_timeout_ticks` (default `DEFAULT_ELECTION_TIMEOUT_TICKS`, 10), and the catch-up request to the stuck slot's owner runs every `heartbeat_interval_ticks` (default 3) of stall. There are no heartbeats under ownership, so a host tuning these values is tuning suspicion and catch-up only.
- **Message cost.** Each decision costs $N-1$ accepts, up to $N-1$ acknowledgements, and $N-1$ commits, for suggestions and skips alike. Every member runs `resend_to` for every peer every `resend_interval_ticks`, sending at most `CHUNK_SLOTS` messages per peer per scan.
- **Window backpressure.** `next_usable_own_slot` returns `.Window_Full` when the next own slot is more than `WINDOW_SLOTS` above `memory_floor`. A stalled prefix that the host has not consumed therefore stops every owner until a revocation fills the hole; hosts should treat `.Window_Full` as they do behind a leader.
- **Membership order is ownership order, and membership order is ascending id.** `membership_init` sorts the ids it is given, so hosts may list them in any order and every node derives the same `owner_of`. Ownership changes only with the membership, at a stop sign; the next configuration recomputes `owner_of` from its own member list and `Log_Envelope` keeps the old configuration's suggestions out. Because other owners may get suggestions decided above the stop sign before they learn of it, `Replicated_Log_Node` abandons every decision above a decided stop sign: never released, not readable, re-decided by the next configuration (POD 0006, scenario `reconfiguration_sim_ownership_abandons_decisions_above_the_seal`).
- **Identity survives restart.** The ownership ballot is `(0, 0, id)`. Within a configuration, a replacement process must restore that member's durable state rather than reuse its id with an empty journal. Reuse across configurations requires the checked configuration boundary and state handover of POD 0006.

- **Mixed groups are not supported.** A member without the option ignores every round-zero accept and campaigns on its own timer; a member with it refuses to campaign. All or none.

Validation and Acceptance Gates

Verification coverage. `tests/test_ownership.odin`, on `Node(u64, 3, 16, 4)` with an in-process queue and a `silent` member whose traffic is dropped: `ownership_three_owners_propose_concurrently` (slots 1 to 4 decided from three owners, `owner_of` values, `campaign` refused); `ownership_idle_owners_skip` (slots 2 and 3 decide the no-op after three ticks); `ownership_revokes_a_crashed_owner` (slot 3 of the silent owner decides the no-op after thirty ticks and both survivors are `.Follower`); `ownership_revocation_keeps_a_seen_vote` (a suggestion heard by one survivor is re-proposed by the revoker and decided); `ownership_revoked_suggestion_is_resubmitted` (a suggestion heard by nobody is revoked to the no-op and, once the owner is reconnected, decided in a later own slot).

Simulator verification. `paxos-sim --ownership` initialises every node with `rotating_ownership = true`, skips the bootstrap campaign, proposes from any live node, and applies every oracle of the single-leader mode: agreement and validity of every durable decision against the golden log, the durable-vote quorum detector and the one-value-per-ballot check, promise regression for the global promise, contiguity of released entries, convergence of every node, and a liveness probe after healing. Crashes land before writes, after a prefix of writes, or after a prefix of messages, and restarts go through `restore` with the same option. The oracle does not cross-check `Write_Promise_At` against later votes; that check is left to the ledger's own `.Promise_Regression` on apply.

Alternatives Considered

- **Single stable leader.** The library's default. One round trip per decision from the leader, an idle member costs nothing, and a crashed follower costs nothing. Rejected as the only mode because every proposal from another member pays a forwarding hop and the leader bounds throughput.
- **Per-instance leader election.** Run a prepare for every slot from whichever member has the value. Correct and leaderless, but it restores the second round trip Multi-Paxos removed and makes every slot contended. Rotating ownership keeps one round trip by making the prepare unnecessary rather than cheap.
- **Dependency-tracking protocols (EPaxos-style).** Fast-path commits with no fixed owner, at the cost of per-command dependency sets, conflict detection, and a separate execution order. The bounded, allocation-free `Node` has no room for dependency graphs, and the log's contract is a single total order released contiguously. Rejected for this library.
- **Leases.** A time-based lease could let an owner serve reads or skip acknowledgements, but it would import a clock assumption into a core that has none. Leases remain the subject of POD 0004 and are orthogonal to ownership.

Open Questions

1. **Piggybacked skips.** Mencius carries skips inside other messages; here each skip is a full round-zero suggestion with its own quorum. Piggybacking a "skipped through" mark on `Accept_Message` or `Accepted_Message` would cut idle-owner traffic but adds fields to every message.
2. **Learning skips without a quorum.** In Mencius a skip needs no consensus, since only the coordinator can suggest in its instance and a skip is the coordinator's word. The library decides skips like values so that the learner never trusts a single member; whether an acceptor may treat an owner's no-op suggestion as decided on receipt, safely, is open.
3. **Resubmission is best effort and can duplicate commands.** A differing replacement queues the lost suggestion; an equal replacement does not. A resubmission is a new proposal, not a deduplicated

application operation; the API supplies no exactly-once command guarantee. Overflow of the one-chunk queue is counted in `resubmits_dropped`; host retries and application ids remain necessary. The SDK must expose this limitation before enabling ownership.

4. **Adaptive ownership shares.** Ownership is uniform round-robin. A busy member cannot take more slots and a quiet one cannot give them up, so a workload concentrated on one member pays $N - 1$ skips per decision. Weighted ownership would need the weights agreed through the log, for example in a stop sign's metadata.

References

- Mao, Yanhua, Junqueira, Flavio P., and Marzullo, Keith. "Mencius: Building Efficient Replicated State Machines for WANs." OSDI, 2008.
- Lamport, Leslie. "The Part-Time Parliament." ACM TOCS, 1998. Section 2.2, conditions B1 to B3.
- Lamport, Leslie. "Paxos Made Simple." ACM SIGACT News, 2001.
- POD 0002: Paxos-Odin: Architecture and Pure State Machine Design.
- POD 0003: Durability Contracts, Window Reuse, and Trim Anchors.
- POD 0004: Fast-Path Leases.
- POD 0006: Reconfiguration and Epoch Isolation.
- POD 0007: Review Findings and Verification Evidence.