POD 0011: Paxodin: A Python SDK over the Odin Core

COMMITTED

CATEGORY	**STATUS**
SDK Design and Implementation	Implemented
AUTHORS	**CREATED**
Paxos Odin Contributors <team@insan.ai>	2026-09-17
LAST UPDATED	**LABELS**
2026-09-17	python, api, packaging, durability

Abstract

`paxodin` is a Python package that runs the existing Odin consensus core inside a Python process. A small native bridge preserves the core's bounded state and durability rules. Python supplies owned byte strings, typed results, useful exceptions, and explicit storage and transport interfaces. A `Session` and `AsyncSession` present compact, Requests-inspired entry points for applications; a lower-level `Node` lets experienced hosts drive the effect machine themselves.

The project lives at `python/paxodin/` and is managed with uv. The package name is published by the tag-driven release workflow after artifact qualification.

Status and Implementation Boundary

Implemented and verified on 2026-09-17. The C ABI, the typed `Node`, the durable `Session`, the asyncio `AsyncSession`, the nine typed message classes, the reference `FileJournal` and `FileHistory`, the wire codec, the in-process sync and async test clusters and the packaging pipeline all exist and are exercised by `make check-python` and `make python-wheel`. Every exception renders in the core's Elm-style shape, and a test enumerates every class for a title, a cause and a hint. A wheel builds from a source distribution outside the checkout and passes a three-node smoke test on CPython 3.12, 3.13 and 3.14 with no Odin compiler present.

Not implemented, and refused explicitly rather than half-supported. Reconfiguration, rotating slot ownership, learners, pre-durable message transmission and custom capacity profiles report a clear capability bit and return `Unsupported_Capability`. Leases and linearizable local reads are not present and must not be inferred; POD 0004 remains unresolved. No transport implementation ships: the package defines the interface and the application supplies the sockets, the framing policy and the peer authentication.

Evidence and its limits. The four-path measurement in this record was taken on one machine and is recorded with its revision, flags and raw samples. It establishes where time is spent across the boundary. It establishes nothing about other hardware, other workloads, or other implementations.

Problem and Scope

A Python user should not have to decode Odin unions or keep pointers into a moving ledger alive. Nor should a short `append` call hide the distinction between a chosen value, a released prefix and an applied command. The wrapper must make common operations easy while preserving these distinctions.

The first release targets fixed membership, single-leader replication, durable restart, bounded backpressure and catch-up. It wraps `Replicated_Log_Node` so that every network envelope already carries a configuration identity. Rotating ownership, reconfiguration, learners and custom compiled capacity profiles are later capabilities, enabled only after their wrapper contracts and tests exist. Unsupported options fail explicitly. Leases and linearizable local reads are not part of this proposal; POD 0004 remains unresolved.

One obligation was missed by the original draft and is now part of the contract. In single-leader mode there is no resubmission: `queue_resubmit` is reachable only under rotating ownership. A leader that loses its ballot after admitting a command can therefore see recovery choose a different value for that slot, and the command is dropped with no protocol signal. `append` consequently compares the decided entry against what it submitted and raises `ProposalLost` when they differ. A receipt that did not make that comparison would report success for a command that was never chosen.

This is forced in the suite rather than argued. A leader is cut off before it admits anything, so its accepts reach no quorum; a survivor wins a higher ballot with the quorum that never saw the vote and fills the same slot with its own command; the partition heals and the first leader asks for the history it missed. Two tests cover it: one establishes that the core really does decide another value in the admitted slot, and one establishes that `append` raises rather than returning a receipt. A third checks that a `CommitTimeout` describes the wait ending and never claims a cancellation.

API and Modern Python

Target **CPython 3.12 and later**, with an initial tested matrix of 3.12, 3.13 and 3.14. Declare `requires-python = ">=3.12"`; list only tested versions and platforms in release support documentation. New interpreters join the matrix before a support claim. Initially support ordinary GIL builds only; free-threaded Python, PyPy and subinterpreters require separate validation.

The governing rule is that Odin is the engine and Python is the product: nothing a Python developer touches is shaped by the C ABI or spelled in the engine's vocabulary. Concretely, this means the following, each of which the implementation now does.

- **Messages are types, not tags.** The nine protocol messages are nine frozen dataclasses - `Prepare`, `Promise`, `PromiseRange`, `Accept`, `Accepted`, `Commit`, `Learn`, `Nack`, `Heartbeat` - and `Message` is their union. An adapter writes `match envelope.message: case Accept(slot=s): ...`. The flat nineteen-field record that the ABI and the wire use exists in exactly one private place, and the type checker proves every `match` over the union exhaustive, so there is no fallback branch and no `kind` field to consult.
- **Time is in seconds.** The engine counts its timers in logical ticks and owns no clock; a `Session` accepts `election_timeout=0.5` and converts, so nobody reasons about ticks who did not ask to.
- **Errors are Elm-style, exactly as the Odin core's.** Every exception renders as a titled banner, the cause with its specific values, and a `Hint:` line with the corrective action. Protocol errors take their text from the core's own table, so there is no second copy that can drift. A test enumerates every exception class and fails on any that lacks a title, a cause or a hint, as the Odin suite enumerates its `Error` enum. Where a builtin already means the right thing, the class inherits it too: `CommitTimeout` is a `TimeoutError`, `ValueTooLarge` a `ValueError`, `StorageError` an `OSError`.

- **Reading is iteration.** `session.entries(start)` yields released entries, `session[slot]` indexes one, and `is_leader`, `leader` and `last_committed` are properties. `NodeState` remains for a host that wants the engine's full view.
- **There is an asyncio surface.** `AsyncSession` drives itself: a receiver task delivers frames, a ticker task keeps the timers moving, and `await session.append(...)` resolves on release. Ownership is explicit - every transition, the caller's or the driver's, runs under one `asyncio.Lock`, and journal syncs run in a worker thread so an `fsync` never stalls the loop. Cancellation is honest: cancelling `append` stops **waiting**; it does not un-propose, because Paxos has no cancel and the API will not pretend otherwise. The synchronous `Session` remains as the sans-I/O reference the async one is built on.

Beneath those: built-in generic annotations, `X | None`, keyword-only arguments, structural `typing.Protocol` adapters, immutable `@dataclass(frozen=True, slots=True)` results, Python 3.12 `type` aliases and PEP 695 generics, `TypedDict` with `Unpack` for option bundles so a call site reads as keyword arguments while staying checked. `py.typed` ships and an external consumer is verified under strict mypy. There are no compatibility shims for older Python, and no caller ever sees a native type.

```
from dataclasses import dataclass
from typing import Protocol

type Slot = int

@dataclass(frozen=True, slots=True)
class Receipt:
    configuration_id: int
    slot: Slot
    value: bytes

class Transport(Protocol):
    def send(self, *, peer: int, frame: bytes) -> None: ...
    def receive(self, *, timeout: float) -> bytes | None: ...
```

The public entry point is a local participant, not an HTTP client or a complete cluster. Its public use is:

```
from paxodin import FileHistory, FileJournal, Session

# transport is an application-supplied authenticated Transport.
with Session(
    node_id=1,
    members=[1, 2, 3],
    configuration_id=1,
    journal=FileJournal("state/node-1"),
    history=FileHistory("state/node-1"),
    transport=transport,
) as session:
    receipt = session.append(b"set counter 41", timeout=5.0)
    print(receipt.configuration_id, receipt.slot)
```

`append` drives local progress until the submitted value is durably known in the contiguous released prefix, or raises a specific exception. Success does not say that every peer has received it or that the application has applied it. A follower raises `NotLeader` with an optional leader hint; v1 performs no automatic forwarding. Other participants must continue serving traffic. `poll(timeout=...)` drives traffic, ticks and retries when there is no append. No hidden worker thread runs. Timeouts use a monotonic host clock and finite defaults; invalid, negative and non-finite durations are rejected before entering native code.

Two details of `append` are easy to get wrong and are fixed by the contract. It first drains whatever peers have already sent, so a member that has won a campaign is not refused as a follower for want of reading its own mail; this is local progress, which `append` is defined to drive, and not forwarding, which it never does. And the journal learns the participant's identity from the session rather than the caller repeating it: `Journal.open(node_id=, configuration_id=)` is called once, and a durable journal uses it to refuse a file that belongs to another member. The example above is a test in the suite, so the record cannot drift from the code it describes.

`committed_since(slot, *, limit=...)` reads retained local history with a bounded result. It supplies no freshness guarantee. Application progress uses a separate durable cursor; receiving a `Receipt` never means "applied." A small `paxodin.testing.Cluster` may provide three in-process participants for examples, with memory-only storage visibly marked as unsuitable for durable deployments.

Architecture and Ownership

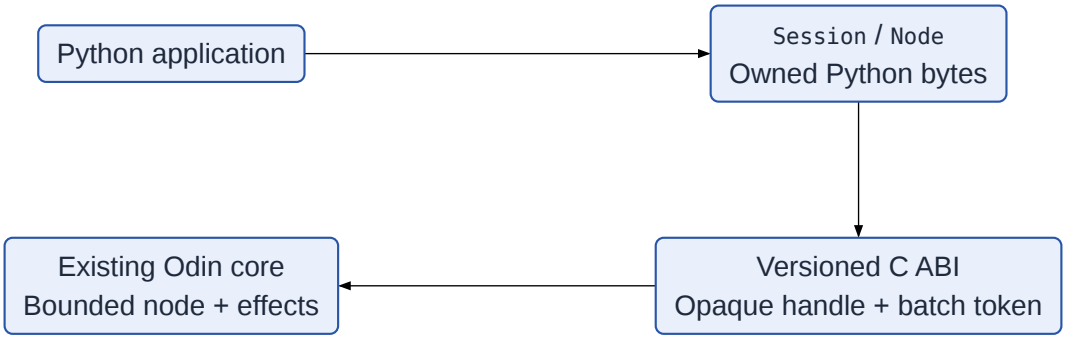

Figure 1: Proposed call boundary. Only the Odin layer implements Paxos.

`Node` owns an opaque native handle. Its constructor chooses a compiled profile and validates all ids, capacities and options. Context management and idempotent `close()` give deterministic release; finalization is only a fallback. Each handle has one lock around transitions, pending-batch access and close. Reentrant entry on that handle is rejected. The bridge does not invoke Python callbacks; storage and transport operations occur after returning across the ABI.

Use `ctypes.CDLL` initially. It provides a standard-library foreign function interface and releases the GIL during calls, so the handle lock remains necessary. Set every `argtypes` and `restype` explicitly. Do not infer safety from the GIL. Inherited handles are invalid after `fork`; record the creating process and raise an actionable error before access in a child. Initial concurrency support allows independent handles on independent threads, with tests for allocator/context safety.

The Odin bridge exports C-calling-convention procedures, initializes the required Odin context on entry, and owns its allocator and storage. The ABI uses fixed-width integers, explicit lengths, status codes and opaque handles. It exports an ABI version, capabilities, core source revision and capacity fingerprint. It never exports Odin struct layout, tagged unions, slices, strings or interior pointers. No Python exception crosses the ABI. Expected failures return status plus details; invariant failures make the handle unusable instead of pretending recovery succeeded.

The initial compiled profile proposes at most seven members, a 256-slot window, 64-slot recovery chunk and 1,024 application bytes per value. These are fixed packaging choices, not runtime-generic Odin instantiations. The native value is a comparable fixed-size record with a kind, a length and a zero-initialized payload array. Kind distinguishes an empty command from an internal no-op; canonical padding makes equal byte strings equal native values. Configuration metadata adds its own bounded overhead. Profile identifiers travel in journals and handshakes. Oversized values fail before mutation, with the supplied length and supported limit.

The Pending-Batch Contract

A naive wrapper could call Odin, then fail allocating a Python list and lose the writes needed before the next call. Therefore the bridge retains one native batch until the host explicitly finishes it. Buffer-size probes and copies are idempotent; allocation failure never repeats the underlying transition.

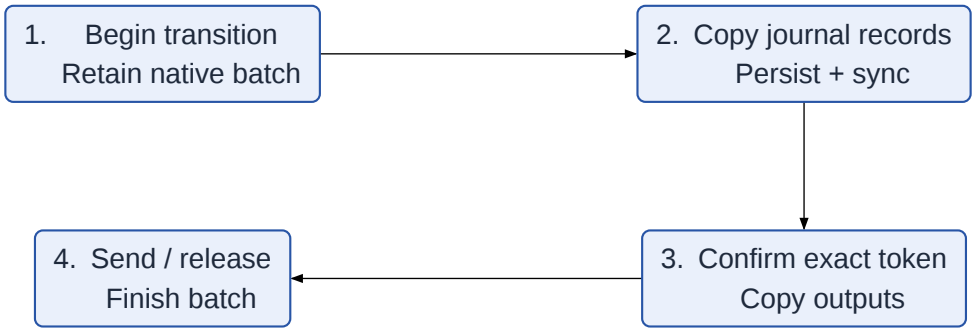

Figure 2: A batch is a durability obligation. The next transition is blocked until it is finished.

1. `begin_*` validates input and performs at most one transition. It returns a new generation token and protocol status. Status and effects are independent: an error can still leave writes that must be handled.
2. `copy_writes(token, ...)` copies versioned journal records into caller-owned buffers. The host persists records in order and performs the required barrier. The initial Session uses a full sync for every nonempty write batch.
3. `confirm(token)` calls the core durability confirmation only after persistence succeeds. A stale, duplicate or foreign token is rejected without changing state.
4. `copy_messages`, `copy_committed` and `copy_requests` copy each output kind separately into owned Python bytes and immutable records, in bounded chunks. A single `copy_outputs` was rejected during implementation: with a 1,320-byte inline value, a full message batch is two thirds of a megabyte in one allocation, which makes the very allocation failure this contract exists to survive more likely, and forces all three kinds to be re-copied to retry one. A failed copy of any kind can be retried at any offset.
5. `finish(token)` releases the pending batch after all required outputs have been copied. Only then can a new transition begin. Session retains copied outbound frames in a bounded queue; send failures never roll back a durable transition.

The guard the bridge enforces is the following, and three rows of it were added after review found the first draft incomplete. `copy_requests` is gated with the messages, because serving a range request means transmitting commits and a send is a send. `advance_memory_floor` requires no live batch, because the floor frees the very window cells the batch's released entries still point into, and a copy that then failed would lose a contiguously released prefix with nothing holding it. `confirm` requires that every record was actually copied, which turns the likeliest integration mistake into a status instead of an acknowledged but unwritten promise. A batch that produced no writes is born confirmed: a transition can release decisions without writing anything - a commit for an already chosen cell, or a configuration mismatch - and demanding a confirmation there would assert a durability fact about no records at all.

A read-only batch-state query reports whether a token is pending, confirmed or finished, together with how many records the host has copied, so an interrupted caller can resume at the correct phase without reconfirming or repeating the transition. Across a process crash the batch is volatile and gone; the only recovery is journal replay. Tokens are bound to a handle lifetime as well as a generation. A no-write batch still has a token and output lifetime. The low-level `Node` API exposes these phases through a `PendingBatch` context object with explicit `persisted()` and `finish()` methods; its exit never confirms automatically. Abandoning a batch blocks progress and reports recovery instructions. `Session` implements the sequence for callers. An I/O error with uncertain persistence closes normal progress: reopen and replay the journal

rather than guessing which writes reached stable storage. Cancellation after admission has the same rule. No pre-durable message optimisation is exposed in v1.

Storage, Wire Format and Recovery

`Journal` is a typed protocol for ordered batch append, sync, replay and close. `FileJournal` obtains an exclusive node-directory lock. Its header records node identity, configuration, format version and profile fingerprint. Records are length-delimited and checksummed. A demonstrably incomplete trailing record can be discarded after a crash; a checksum failure in a complete record is corruption and must stop recovery with the record offset and a repair hint. Never silently skip an interior record or invent a new node identity.

Replay feeds decoded records to the core's replay contract and restores with the last durably consumed floor. Store released entries in a durable history journal before advancing the native memory floor: this transfers responsibility for those entries from the bounded node to the host. Application acknowledgements update a separate durable cursor. History remains available for catch-up independently of the application cursor. Do not trim that history without a durable snapshot and trim-anchor policy that can answer `Serve_Range_Request`. V1 retains history on disk and refuses writes on quota exhaustion; snapshot-based compaction is deferred. This bounds RAM without claiming bounded disk use.

The transport carries versioned, length-delimited, configuration-stamped frames with fixed byte order and explicit variant tags. Validate total size, all lengths, ids, enum values and capabilities before constructing an Odin message. Authenticate the claimed sender through the transport adapter; a sender id in bytes is not authentication. No pickle or native-memory serialization is allowed. Unknown versions or incompatible profiles fail with upgrade or routing guidance.

The transport contract preserves frame boundaries and bounds queued bytes. A failed send may mean the peer received the frame; protocol retransmission tolerates duplicates. Catch-up service reads retained history on a bounded schedule. Adapter callbacks must not recursively drive the same Session. Closing stops local work and frees resources; it does not withdraw a proposal that peers may still choose.

Errors and Developer Experience

Expose a `PaxodinError` hierarchy with stable `code`, concise `message`, actionable `hint`, and structured context. Preserve core error code and hint where relevant. Validation failures identify the argument, actual value and expected bound. Native loader errors report the platform, missing library or ABI mismatch and the supported installation path. Do not include command contents or credentials in errors by default.

```
ValueTooLarge: append received 1536 bytes; this profile allows 1024.
Hint: store a reference to the larger object, or install a compatible larger profile
on every member before creating the configuration. No proposal was admitted.

CommitTimeout: slot 28 was admitted, but its decision was not observed within 5s.
Hint: continue polling and inspect local history. A timeout does not cancel Paxos.
Retry a command only with an application id and deduplication policy.
```

`CommitTimeout` includes `admitted`, configuration and slot when known. Admission itself does not guarantee the value will win recovery. Do not automatically retry an admitted command into a new slot. Exactly-once application is an application protocol, not a property inferred from `append`. `WindowFull` explains which retention or durable-progress condition prevents window reuse. `StorageError` reports that progress stopped and directs the operator to reopen/replay.

Project and Build Design

```
python/paxodin/
  pyproject.toml          # PEP 621 metadata and tool settings
  uv.lock                 # development and test dependency lock
  .python-version         # pinned development interpreter
  hatch_build.py          # build hook; invokes a pinned Odin compiler
  native/
    paxodin.h             # versioned C ABI
    bridge.odin           # adapter over the existing core
  src/paxodin/
    __init__.py
    _native.py
    node.py
    session.py
    models.py
    errors.py
    protocols.py          # Journal, Transport and Clock interfaces
    codec.py              # frame and envelope encoding
    storage.py            # FileJournal reference adapter
    testing.py            # in-process cluster for examples and tests
    py.typed
  tests/                  # unit, integration and packaging checks
  examples/
```

Use uv for environment management and locked development dependencies, with hatchling as the PEP 517 backend and a build hook that invokes the Odin compiler directly. `uv_build` currently targets pure Python packages, so it is not the native build backend here.

This supersedes the earlier proposal of scikit-build-core with a CMake custom command, on the 2026-09-17 evidence that the project contains no C or C++ source: CMake would have configured a toolchain only to shell out to one `odin build` invocation, and would have added `cmake` and `ninja` to every build environment to do it. The hook is sixty lines and calls the compiler as the Makefile already does. Neither choice implies a second consensus implementation; the point of recording the change is that the shorter path was taken deliberately.

Ruff is both the formatter and the linter, mypy checks the typed API, pytest runs scenarios, and Hypothesis explores byte/frame and lifecycle inputs. The structural limits of POD 0001 apply to Python as they do to Odin: ruff carries every rule except file length and procedure-body length, which `tools/check_style.py` adds through the `ast` module so one set of constants governs both languages. Pin tool versions in the lockfile and build requirements separately: an isolated PEP 517 build does not automatically inherit every uv development constraint.

The editable project compiles against the repository's existing `src/`. Release staging copies that exact core revision into the source distribution with a source manifest and licenses. Do not maintain a second editable copy of the algorithm. CI must build a wheel from the sdist in a directory outside the repository; paths such as `../../src` cannot be required by an installed source distribution. Source builds require the declared Odin toolchain; wheel users do not need Odin.

Bundle the platform shared library as package data. Load it with `importlib.resources`, keeping any extracted-resource context alive for the loaded library's lifetime. Rebuilding an editable library requires a new interpreter; a process must not hot-replace an already loaded binary. The artifact uses a platform wheel tag, normally `py3-none-<platform>` for a ctypes library; it is neither a platform-independent `any` wheel nor a CPython `abi3` extension.

Start wheel qualification on Linux x86-64, then add macOS arm64 and Windows amd64 only after their packaging and durability suites pass. Use cibuildwheel with platform repair tools where required; inspect bundled dependencies and exported symbols. Build for a documented portable CPU target, not the builder's native microarchitecture. Record Python, backend, Odin and source versions in artifacts. Package versioning is independent of the core; ABI compatibility and bundled core revision are queryable. Publishing is a separate release action.

Validation and Acceptance Gates

1. Native boundary

Implement a single profile, exported header, handle lifecycle, validation and pending-batch protocol. Compile a standalone C consumer against the header and run create/close/restore/transition scenarios. Verify integer bounds, canonical payload padding, empty-command/no-op distinction and stale tokens. Compare every bridge output with a native Odin run of the same trace before adding Session.

Outcome. The bridge is nine Odin files under `python/paxodin/native/`. The guard table is exercised by a phase matrix covering every (phase, call) pair. The whole suite runs a second time against the `.Enforced` twin, which is the standing proof that the host-managed bridge never trips the core's own gate.

2. Typed Node

Implement ctypes declarations, owned-copy outputs, lock/reentrancy rules and error mapping. Test forced allocation failures before and after mutation, retries of copy operations, concurrent close, use after close, post-fork access and repeated creation/destruction. Test that native memory remains bounded across moving windows and that previously returned Python bytes survive many later transitions.

Outcome. 27 hazard tests cover oversized values, stale, foreign and post-reopen tokens, use after close, idempotent close, concurrent close, reentrancy, access after `fork`, abandoned batches, no-write batches, a floor advance refused while a batch is live, 1,200 values through a 256-slot window, and a `bytes` that stays intact after 600 later transitions have reused its cell.

3. Durable Session

Implement journal, authenticated transport protocol, progress loop, history and application cursor. Run three- and five-node integration scenarios with drops, duplicates, reordered frames, leader loss and restart. Inject crashes before writes, partway through records, after sync, before confirm and during output delivery. Check agreement, no reused ballot/value pair, contiguous release, replay equivalence and preservation of unapplied history. Add disk-full, torn-tail, corrupt-record, wrong-configuration and mismatched-profile tests. No success receipt may escape before its required writes are durable.

4. Packaging and usability

Build sdist then wheel in isolation. Install wheels in clean environments without Odin, import the native library and run replication/restart smoke tests on every claimed platform/Python pair. Check type information from an external consumer, licenses, resource loading, ABI diagnostics and examples. Keep all detailed docs in this POD and the book; a package README is justified only as its GitHub entry. Run proposed examples as tests once the package exists; until then label them as design examples. Do not report this gate complete from documentation compilation.

Outcome. `make python-wheel` builds the sdist, then builds the wheel from it in a temporary directory where `../../src` does not resolve, and installs the result into a clean environment with Odin removed from `PATH`. Three-node consensus runs on CPython 3.12, 3.13 and 3.14. The artifact is `py3-none-linux_x86_64`; the check fails an `any` or `abi3` tag. An external consumer typechecks against `py.typed` under `mypy --strict`.

macOS and Windows are not claimed. The API reference is generated from docstrings by mkdocstrings, so it cannot drift; the prose stays here and in Part IX.

5. Performance and memory evidence

Measured 2026-09-17 on an AMD Ryzen 7 5800H, Linux 7.0.0, CPython 3.13.5, Odin `dev-2026-09-nightly`, core `0.2.0`, built `-o:speed -no-bounds-check -microarch:x86-64-v2`. One voting member, one value per transition, the full batch lifecycle discharged and the memory floor advanced each time. Nine samples per row, paired bootstrap interval, raw samples in `bench/results/paxodin-paths-20260917.json`. Commands are recorded in that file.

Path	ns/value	vs native	ABI crossings	What it adds
native Odin	826	1.00	-	the transition alone
C ABI	7,008	8.52	7	ctypes crossings and copies
Python `Node`	23,787	28.78	7	owned Python objects
`Session` (memory)	29,739	35.98	8	framing, journal, ordering
`Session` (fsync)	48,091	58.29	8	a durable barrier per batch

The rows are **not** interchangeable and the ratios are diagnostic, not a ranking: comparing a Python `fsync` against a native in-memory transition would say nothing. What the table establishes is where the cost accrues. The boundary itself is **883 ns per crossing** at seven crossings per value, so the C ABI accounts for 6.2 of its 7.0 microseconds. Building owned Python objects costs a further 17 microseconds, which is the price of the ownership contract: a `bytes` returned today stays correct after any number of later transitions. Payload size moves the native and ABI rows by under 5% across 8, 64 and 1,024 bytes, because values are stored inline at a fixed size and a larger one costs the engine no allocation. The Python rows move by up to 12% at 1,024 bytes, and that is the copy into an owned `bytes` - again the ownership contract, paid once per released entry. The table was re-taken after the typed message classes and the asyncio surface landed; the typed boundary added no measurable cost.

No claim follows from this about other hardware, other workloads or other implementations, and none is made. The measurement's purpose was to decide whether a more complicated binding is warranted before building one. It shows that batched FFI would address 6.2 microseconds and a compiled extension would address the 17 spent on object creation; neither is undertaken here.

Compare native Odin, the C ABI, Python `Node`, and `Session` under matched member counts, payload sizes, capacities, depths and completion criteria. Include 3/5 members, 8/64/1024-byte values and depths 1/8/64. Measure transition-only work and full durable host work separately. Count FFI crossings, allocations, copied bytes, queue high-water marks and sync calls. Profile before considering batched FFI, CFFI or a compiled CPython extension.

Use paired repeated samples with recorded revisions, compiler flags, CPU and raw results. Report throughput, latency distribution and uncertainty. Distinguish native inline bytes, Python allocations, queue memory and process RSS. Match serialization, durability and release semantics before comparing another package or the existing Zig harness; never compare Python fsync latency with native in-memory transition cost. Set regression budgets from the first reproducible baseline, not an invented universal target. Publish evidence in Typst and raw data under the benchmark results tree. No "fastest Python Paxos" claim follows from native-only results.

6. Optional capabilities

Status: none of these are enabled. Each reports a clear capability bit and returns `Unsupported_Capability` rather than half-working.

Expose ownership only after testing bounded resubmission loss and duplicate commands through Python. Expose reconfiguration only after state handover, configuration fencing, stop release and crash recovery are tested end to end. Each capability needs its own documentation, feature bit and negative tests on builds that omit it. Asynchronous support shipped with the ownership and cancellation model described above; it is not a wrapper of blocking calls in tasks, which would have left two coroutines free to transition one node at once.

Alternatives Considered

A direct CPython extension may reduce conversion overhead but adds interpreter ABI and reference-management work. CFFI supplies another FFI path but adds build or runtime dependencies. An out-of-process service isolates crashes but creates a network protocol and operational service. Begin with ctypes and a small stable boundary, then let matched profiles justify additional machinery.

Open Questions

The open questions of the draft are now settled. The stock profile is seven members, a 256-slot window, a 64-slot recovery chunk and 1,024 payload bytes; the measured cost is 417,304 bytes for one node and 41,840 for one effects batch, both reported through `paxodin_profile`. The release matrix qualifies Linux x86-64, Windows x86-64, and macOS Apple Silicon with native builds and installed-wheel tests. The journal format is a locked node directory holding a header that binds node identity, configuration, format version, and profile fingerprint, followed by length-delimited CRC-32 records whose values travel inline, because replay dereferences each record's value. The Python distribution is named `paxodin`; the Odin package remains `paxos`. Tagged releases publish only after the verification and artifact jobs succeed.

Two decisions changed during implementation, and the reasons are recorded rather than the outcomes alone. The build backend is hatchling with a sixty-line hook instead of scikit-build-core with CMake, because the project contains no C or C++ and CMake would have configured a toolchain solely to invoke one `odin build`. The shipped library compiles the core with `Host_Managed` instead of the default gate, because `host_order_violation` calls `os.exit` and a host inside a Python interpreter cannot accept a process kill with no traceback and no chance to close its journal; the bridge enforces the same four rules itself and a second library compiled with `Enforced` runs the whole test suite as a standing proof.

What remains open is the clock and authentication contract of each transport adapter, which is per-adapter review rather than a property of this package, and the trim-anchor policy that snapshot-based history compaction would need.

References

- POD 0003: durability, borrowed effects, replay and memory floors.
- POD 0006: replicated log configuration identity and stop signs.
- POD 0009: memory accounting and matched performance evidence.
- uv build backends.
- uv project configuration.
- scikit-build-core: packaging ctypes libraries.
- Python ctypes reference.
- Odin overview: foreign interfaces and context.
- Requests sessions and context management.

- cibuildwheel documentation.