

JUNGLE • RESEARCH & ARCHITECTURE

JUNGLE

MEMORY HIERARCHY

An exploration into how economic thought and models can be technically applied to make emerging technologies more accessible

RESEARCH AND ARCHITECTURE STRATEGY REPORT
AUGUST 2026 • DECISION DRAFT

The move is architectural. MCP, plugins, hooks, SDKs, proxies, and IDE integrations become adapters. The product boundary becomes the memory kernel, canonical ledger, context compiler, and operating contract.

VERIFIED CURRENT STATE • RESEARCH FINDING • TARGET ARCHITECTURE • RECOMMENDATION

Executive decision

Build the memory data plane

Recommendation. Jungle should define and own a new infrastructure category: the **memory data plane for LLM systems**. Its job is to persist what an AI system may need to know, preserve why that knowledge is trustworthy, track when it is valid, and deliver only the minimum evidence needed for the next action. Retrieval-augmented generation remains an access pattern inside this system, vector search one rebuildable index, and MCP one transport. The product boundary is the data plane itself.

The analogy to PostgreSQL is structural. Applications adopt PostgreSQL because it provides a stable system of record with transactions, concurrency control, recovery, access policy, observability, migration, and an ecosystem of clients. Jungle must offer the corresponding contract for machine memory: durable writes, snapshot reads, bitemporal truth, scope isolation, provenance, change streams, backup and restore, policy enforcement, and service-level objectives. PostgreSQL's write-ahead logging, multi-version concurrency control, logical replication, row-level security, and point-in-time recovery show the institutional behaviors expected from a trusted data layer. [R7–R12]

The analogy to an operating-system memory manager is equally important. Long-context models still degrade as irrelevant material accumulates, and important evidence can be missed even when it is present. [R2–R4] Jungle therefore should answer “what does this agent already have, what has changed, what must be resident for the next operation, and what is the cheapest trustworthy delta?” The unit of optimization is the **task working set**.

The current codebase contains unusually strong precursors: typed memories, claims and evidence, intent tasks, checkpoints, atomic catalog generations, structural and lexical indices, source provenance, runtime sessions, residency records, leases, action ledgers, and a bounded context loop. [J1–J7] The remaining work is promoting these mechanisms into one coherent kernel with public contracts and database-grade reliability.

The decision in one sentence

Make Jungle the model-neutral system of record and runtime for context—where every memory is typed, versioned, scoped, temporal, attributable, policy-governed, and served through a measurable hierarchy.

The seven bets

What must be true for Jungle to become essential

Bet	Commitment	What changes in practice
1. The ledger is canonical	Every durable fact, claim, artifact, decision, and observation enters an append-oriented event history.	Search indices, graphs, summaries, and vectors become disposable projections that can be rebuilt.
2. Context is compiled	A provider-ready prompt is a derived artifact.	Jungle records the dependency set, policy decisions, layout, token cost, and validity receipt for every served bundle.
3. Modules are the unit of locality	The page table holds coherent, multi-resolution modules that combine code, memories, contracts, evidence, and verification.	Retrieval moves from isolated chunks to addressable working units that can be paged, refreshed, and shared.
4. Time and authority are first-class	Valid time, transaction time, origin, corroboration, and confidence are orthogonal structured fields.	Stale or adversarial memories can be contained without erasing history.
5. Sessions are transactional	Writes are idempotent; reads have declared consistency; conflicting updates use compare-and-swap or optimistic concurrency.	Multiple agents can cooperate without silently overwriting task state or presenting mixed revisions.
6. The runtime owns residency	The daemon or gateway tracks what is resident, predicts page faults, prefetches, evicts, and detects thrash.	MCP and tool descriptions stop carrying the burden of memory management.
7. Trust is measurable	Freshness, evidence coverage, stale-serve rate, redundant tokens, recovery point, and task outcomes have SLOs.	Jungle can be purchased and operated as infrastructure.

These bets are mutually reinforcing. A module without a canonical ledger becomes another cached summary. A ledger without a runtime becomes an archive. A runtime without authority labels becomes an injection surface. A secure store without measurable serving quality becomes an expensive database that models ignore. The product emerges only when storage, semantics, policy, and runtime behavior share one contract.

The end state

Any agent can attach to Jungle and ask for a task-scoped working set. Jungle resolves identity and scope, opens a consistent snapshot, computes the delta from the agent's believed resident state, pages in exact evidence and coherent modules, lays them out for the target model, records a receipt, observes the outcome, and safely incorporates new knowledge. The model need not know how the hierarchy works—just as an application need not schedule disk pages.

Contents and reading map

A forty-page decision path

Pages	Part	Decision answered
5–6	I • The category shift	What makes Jungle a core data layer?
7–9	II • Starting position	Which current capabilities are durable assets, and what is still missing?
10–13	III • Research and market	What do RAG, memory systems, databases, and operating systems imply?
14–15	IV • North-star contract	What exactly should Jungle promise to applications and operators?
16–20	V • The memory hierarchy	What lives at each level, and how does information move?
21–24	VI • Canonical data model	How are objects, time, authority, transactions, and modules represented?
25–28	VII • Reference architecture	How do the data, control, serving, and learning planes cooperate?
29–32	VIII • Infrastructure posture	How do security, governance, APIs, reliability, and SLOs work?
33–34	IX • Proof and economics	How will Jungle demonstrate outcome value and sustainable unit economics?
35–38	X • Migration and execution	What is built when, what changes organizationally, and where are the kill gates?
39–40	XI • Decision and sources	What should be authorized now, and what evidence supports the report?

How to read the status labels

- **Verified current state** describes behavior visible in the repository's code, schema, or maintained documentation as of this report. It makes no claim of production readiness.
- **Research finding** summarizes primary literature or official system documentation. Recent 2026 preprints are treated as directional evidence.
- **Target architecture** is a coherent proposed end state. It should be evaluated by its contracts and falsifiable outcomes, even where a named component does not exist today.
- **Recommendation** is a prioritization choice. It can be rejected or sequenced differently without disputing the underlying evidence.

Method

The analysis triangulates four bodies of evidence: the Jungle repository and schema; official database, protocol, and observability specifications; peer-reviewed or primary research on RAG, long context, and memory systems; and current product documentation from adjacent systems. Claims are deliberately separated from proposals. Competitive features are used only to identify table stakes.

I · The category shift

The plugin framing is now the constraint

A plugin is optional, host-defined, and invoked at the model or application's discretion. It inherits the host's lifecycle, identity model, failure semantics, and observability. Its value is judged per invocation: did the tool return a useful result? That framing was appropriate for proving that Jungle could compress a repository into precise, bounded context. It falls short for the next stage because memory correctness is a property of the whole lifecycle.

An essential data layer has the inverse relationship. Applications depend on its identifiers, consistency rules, schemas, policies, and recovery semantics. It is present before a prompt is assembled and remains authoritative after the model response is gone. It can tell an operator which version was served, why an object was eligible, who wrote it, whether it has been superseded, which index produced a candidate, and how to reconstruct the state seen by an agent.

Plugin posture	Core data-layer posture
A model chooses whether to call a tool.	The runtime automatically opens and maintains task memory.
The response is the product.	The canonical state, contract, and lifecycle are the product.
Host session state is assumed.	Session identity, snapshots, and residency are explicit.
Search relevance is the principal metric.	Correctness, freshness, sufficiency, latency, cost, and recovery are co-equal.
Integration-specific payloads dominate.	A stable Context ABI is adapted to every host and model.
Summaries and embeddings can become implicit truth.	Derived representations are versioned projections with lineage.
Failure means an unhelpful answer.	Failure has declared modes, retry semantics, alerts, and an error budget.
Security is tool permissioning.	Security covers write authority, row scope, provenance, retention, and exfiltration.

Recommendation. Stop presenting MCP as the center of the architecture. Keep it as a first-class compatibility surface, especially for discovery and low-friction adoption, but show it beside SDK, proxy, IDE/LSP, CI, event-stream, and service APIs. The daemon-owned memory kernel and canonical contracts must survive the rise and fall of any integration standard.

This reframing also changes product planning. “Add another tool” becomes “extend a stable service contract.” “Improve retrieval” becomes “improve a measurable page-in policy.” “Store a memory” becomes “commit a typed object under authority, temporal, and retention policy.” “Support another model” becomes “implement a layout and cache adapter against the same Context ABI.”

The infrastructure test

Ten properties Jungle must earn

Calling Jungle infrastructure is credible only when operators can rely on the following properties. They form a practical acceptance test for the strategy.

1. **Durability.** Acknowledged durable writes survive process and machine failure within a declared recovery point objective.
2. **Consistency.** Every read declares a consistency level—snapshot, read-your-writes, bounded stale, or latest—and the service never silently mixes incompatible catalog generations.
3. **Stable identity.** Memories, claims, modules, evidence, tasks, and artifacts have stable, content-addressable or logical identifiers that survive file moves and integration changes.
4. **Transactions and concurrency.** Clients can group writes, use idempotency keys, detect conflicts, lease work, and commit or abort without corrupting shared state.
5. **Temporal truth.** The system preserves both when a claim was true in the source world and when Jungle learned or changed it.
6. **Lineage.** Every served assertion can be traced through transformations to source evidence, policy decisions, and the catalog generation that authorized it.
7. **Isolation and governance.** Organization, repository, branch, task, user, and privacy scopes are enforced below the retrieval layer, with default-deny behavior where policy is absent.
8. **Recovery and portability.** Operators can back up, restore to a point in time, replay changes, export canonical records, rebuild projections, and migrate without prompt archaeology.
9. **Observability and SLOs.** Latency, freshness lag, stale serves, evidence integrity, page faults, context churn, token waste, and outcome deltas are visible and bounded.
10. **Protocol independence.** The service can outlive MCP, a provider SDK, an IDE, or a model family because its core object model and wire contracts are versioned independently.

PostgreSQL became foundational by making these concerns routine. WAL places the durable log before page mutation; MVCC gives transactions consistent snapshots; logical decoding publishes ordered changes; row-security policies constrain visibility; continuous archiving enables point-in-time recovery. [R7–R12] Jungle needs equivalents at the semantic level: a memory event log before projection updates, snapshot-visible catalogs, ordered change feeds, policy-gated scopes, and replayable state.

Falsifiable threshold. Within twelve months, a new agent host should be able to integrate Jungle without knowing its internal storage engine, reproduce any served context from a receipt, continue a task across models, recover from a failed write without duplicate memory, and export all canonical data. If that cannot be demonstrated, Jungle remains an advanced tool and the data-layer claim fails.

II · Starting position

Verified current state: the durable substrate already exists

The repository is materially closer to a memory database than its current product framing suggests. The schema contains a set of primitives that most context tools would need years to discover. [J2]

Existing primitive	Strategic value	Promotion required
Projects, corpus revisions, change journal	A durable identity and ordered record of repository change.	Formalize retention, sequence guarantees, replay, and external change-feed contracts.
Atomic catalog generations and plane receipts	A candidate-to-active publication boundary across source, structure, text, and vector planes.	Make generation snapshots universal across read APIs and publish failure/recovery semantics.
Files, source lines, structures, symbols	Exact evidence and structural addressing.	Introduce rename-stable object identity and a formal source-artifact schema.
Memory items, vectors, sources, scopes, policy receipts	Typed durable memories with deduplicated vector representations and enforcement traces.	Split canonical record from projections; strengthen validity, authority, and retention semantics.
Claims, occurrences, evidence, validation, relations	An epistemic ledger capable of representing support and contradiction.	Define public trust levels, corroboration rules, expiry, and promotion gates.
Knowledge packages, contracts, dependencies, covers	A precursor to coherent, reusable memory modules.	Unify packages with code and runtime residency; add multi-resolution manifests.
Intent tasks, commitments, checkpoints, revisions, audits	Durable task continuity and explicit user intent.	Expose transactional task APIs and cross-host continuation guarantees.
Runtime sessions, operations, leases, presence, action ledger	Concurrency and causal execution primitives.	Consolidate ownership and versioning under one runtime controller.
Health, extraction, provenance, redaction, scavenge records	The start of operator-grade audit and hygiene.	Convert internal records into SLOs, incident workflows, and externally consumable telemetry.
Lexical, structural, graph, and vector representations	Multiple access paths over the same corpus.	Declare every index rebuildable, versioned, quality-scored, and subordinate to canonical state.

This is a powerful starting point because it permits an evolutionary migration. Jungle does not need to discard SQLite, LanceDB, FTS5, the daemon, or the current MCP surface to establish the new contract. It needs to place a stable kernel boundary around the semantics already emerging, then make storage engines and transports replaceable behind that boundary.

Important caveat. Schema breadth and operating maturity are different things. A table name proves that a concept has been modeled; its transactions, lifecycle, migrations, monitoring, and user-facing contract may still be incomplete. This report treats the schema as architectural evidence; the infrastructure test is judged separately.

Verified current state: the runtime insight is differentiated

Jungle's strongest conceptual asset is the working-set model described in the Context MMU and runtime plans. [J3–J6] Conventional RAG asks which stored chunks are similar to a query. Jungle's emerging runtime asks what the agent is believed to hold, what the task is likely to need next, and which delta preserves progress at the lowest effective cost.

The current design already distinguishes four useful roles:

- a **resident belief** about what remains in the model-visible window;
- a **task working set** of recently touched or pinned objects;
- a cheap **page table** of handles, outlines, signatures, and summaries;
- a **cold evidence store** from which exact bodies and provenance can be paged.

Embeddings are correspondingly treated as vocabulary-mismatch insurance: they propose useful page-ins but do not define truth. That is the right hierarchy. It avoids a category mistake common in memory products, where the vector index becomes both discovery mechanism and de facto record system.

The repository also contains the ingredients for causal continuity: intent contracts, checkpoints, file residency, session operations, versioned runtime state, and action ledgers. In combination, these could let Jungle reconstruct not just what was retrieved, but what the agent believed, which evidence was available, which operation followed, and what outcome occurred.

What is differentiated now

1. **Exact structural evidence plus bounded expansion.** Outlines and signatures lead to source slices.
2. **Task-wide state.** The unit of optimization can span a whole edit loop.
3. **Freshness as a publication decision.** Atomic catalog generations prevent a vector plane or structure plane from silently representing a different corpus revision.
4. **Intent continuity.** User commitments and corrections can remain authoritative across compaction, host, and agent boundaries.
5. **Cross-agent primitives.** Leases, active claims, presence, and handoff records anticipate shared memory concurrency.

Recommendation. Make the Context MMU the serving runtime of a larger memory data plane. The hierarchy is how Jungle serves. The canonical ledger is what Jungle preserves. The control plane decides what is legal. The compiler turns those inputs into a provider-specific working set.

The gap analysis

What prevents data-layer status today

The current architecture is rich but still legible as a collection of capabilities. The final push is to make it behave as one system with hard boundaries.

Gap	Why it matters	Required correction
No singular canonical memory contract	Multiple tables and runtime records can encode overlapping truth.	Define a versioned object/event model and name the authoritative write path.
Transport still shapes product semantics	Tool descriptions, hooks, and host conventions leak into memory behavior.	Move lifecycle ownership into daemon/gateway; adapters become thin and stateless where possible.
Projection/canonical boundaries are implicit	Summaries, vectors, tags, and packages risk becoming incomparable sources of truth.	Record derivation manifests, source versions, model/policy versions, and invalidation dependencies.
Residency is a belief without complete feedback	Prefetch, replacement, confidence decay, and thrash detection are partial.	Implement a closed-loop memory manager with page-fault and consumption telemetry.
Concurrency semantics are fragmented	Multiple agents can race across task, memory, and file state.	Provide transactional batches, idempotency, leases, version checks, and consistent snapshots.
Enterprise trust is modeled but not packaged	Scope, redaction, and provenance tables do not yet equal policy governance.	Add identity integration, row-policy enforcement, audit exports, retention, deletion, and incident response.
Recovery lacks a public contract	Operators cannot buy a system whose RPO/RTO and replay behavior are undefined.	Add backup/restore, change feeds, migration tooling, corruption checks, and disaster exercises.
Evaluation emphasizes components	Better retrieval can coexist with worse end-to-end task performance or higher cost.	Measure trajectory outcomes, evidence integrity, warm-session gains, and cache-weighted effective tokens.
The category remains underspecified	Buyers compare Jungle with vector databases or IDE plugins on their terms.	Publish the Memory Data Plane contract, reference architecture, compatibility matrix, and operating SLOs.

Two forms of debt deserve special attention. First, **semantic debt**: duplicated concepts whose intended authority is clear to maintainers but not enforced by schema or API. Second, **operational debt**: behavior that works in a local dogfood environment but lacks bounded failure modes under multi-tenant load, partial index failure, delayed events, model outages, schema migration, or hostile input.

The migration should therefore avoid a large rewrite. The correct sequence is contract first, canonical log second, projection discipline third, runtime closure fourth, and distributed operation last. Every phase should leave current local workflows working through an adapter.

III · Research and market

RAG is only an access method

The original RAG formulation joined a parametric language model with non-parametric retrieved passages, improving knowledge-intensive generation and source updating. [R1] That innovation remains foundational. It does not define the complete lifecycle of organizational or agent memory.

RAG typically begins with a query and returns ranked material. A memory data plane must begin earlier and finish later. It observes writes, establishes identity and scope, versions truth, resolves policy, tracks what the consumer already has, chooses a representation, lays out evidence, records the serve, captures downstream outcomes, and revises or expires derived knowledge. Retrieval is one operator inside that transaction.

Long-context evidence strengthens this distinction. “Lost in the Middle” showed that models can underuse relevant information depending on its position even when it is present. [R2] More recent studies report broad context degradation as input length increases and material performance loss even when retrieval is perfect. [R3–R4] The implication is that **availability, attention, and validated use are three different things.**

RAG question	Memory data-plane question
Which passages match this query?	Which typed objects are valid, authorized, and absent from the consumer’s working set?
What is top-k?	What is the minimum sufficient dependency closure for the next operation?
How similar are query and chunk?	What evidence, authority, recency, structural role, and task locality support this page-in?
How do we rerank?	How do we compile, place, pin, refresh, and later evict the selected objects?
Did the answer cite a passage?	Was the decision grounded, did the task succeed, and should the memory be promoted or quarantined?

Compression systems such as LLMLingua demonstrate that token reduction can preserve useful signal, while agent-facing information-design research suggests that layout and tool descriptions materially change behavior. [R5–R6] These findings point toward a compiler: an intermediate representation, target-model layout passes, dependency manifests, and receipts that make each serve reproducible.

Research inference. The durable category is the system that decides when RAG, exact lookup, structural expansion, a task checkpoint, a shared decision, or no new context at all is the correct memory operation.

Memory systems are converging on hierarchy

Several adjacent systems validate pieces of the direction. MemGPT frames limited context as virtual memory and lets an agent move information between tiers. Letta exposes persistent context blocks that can be attached, detached, and shared. LangGraph persists checkpoints by thread for fault tolerance, human intervention, and time travel. Mem0 focuses on extracting and consolidating long-term memories. Zep models temporally changing facts in a graph. [R13–R17]

These systems demonstrate demand, but they also expose the unoccupied center of the market. Most begin from conversational memory, agent state, or a retrieval layer. Jungle begins from a versioned software corpus, exact structural evidence, task intent, and causal execution. That starting point makes it possible to connect four things that are usually separate: **source truth, derived knowledge, runtime residency, and action outcomes.**

Recent research makes the hierarchy more explicit. The 2026 “Missing Memory Hierarchy” preprint argues for a production-proxy memory manager built on demand paging. ContextWeaver and virtual-context-compression work similarly treat context as a managed representation. Human-inspired memory work explores consolidation, forgetting, and conflict resolution. [R18–R21] These preprints are directional, and their convergence is strategically meaningful.

What Jungle should absorb—and what it should resist

Pattern	Absorb	Resist
Virtual context	Explicit tiers, page faults, resident-set accounting.	Giving the model unbounded authority to rewrite durable memory.
Persistent blocks/checkpoints	Addressable, shareable state with thread continuity.	Treating preformatted prompt text as the canonical record.
Memory extraction	Promotion pipelines, consolidation, deduplication.	Silent summarization without lineage, validity, or model-version receipts.
Temporal knowledge graphs	Validity intervals, contradiction, supersession, entity relations.	Making a graph database the only representation or query path.
Human-inspired forgetting	Decay, rehearsal, utility, and retention policy.	Anthropomorphic policies that cannot be measured or reproduced.

Research inference. The market is likely to separate into memory applications and memory infrastructure. Applications will provide domain-specific behavior, personalities, workflows, and user experience. Infrastructure will provide identity, storage, hierarchy, policy, consistency, observability, and portability. Jungle should be the latter and enable the former.

Databases and operating systems supply the missing discipline

The strongest design language for Jungle comes from mature systems that manage scarce resources and shared truth.

From databases: a canonical log, typed schemas, transactions, stable identifiers, query planning, secondary indices, multi-version reads, change data capture, replication, backup, recovery, policy, and schema migration. PostgreSQL’s durability and concurrency model shows why a trusted layer must make failure behavior explicit. [R7–R12]

From operating systems: address spaces, page tables, working sets, demand paging, prefetch, replacement, dirty pages, page faults, locality, and thrash detection. Denning’s working-set model tied a process’s recent reference behavior to the memory it needs to progress and explained why overcommitment produces thrashing. [R22–R23] For LLMs, the scarce resources are effective attention, provider cache topology, latency, and money, alongside the nominal token window.

From compilers: a target-independent intermediate representation, dependency graphs, invalidation, optimization passes, deterministic builds, source maps, and target-specific code generation. Jungle’s “context” should be compiled from typed objects into a provider layout, with a manifest that lets an operator reproduce it.

From data platforms: immutable events, materialized views, data contracts, lineage, quality tests, batch plus streaming ingest, and separation of control and data planes. W3C PROV-O and OpenLineage offer compatible vocabularies for entities, activities, agents, runs, jobs, datasets, and facets. CloudEvents offers a portable event envelope. [R24–R26]

The synthesis

Mature concept	Jungle equivalent
Write-ahead log	Memory event ledger: append before updating projections.
MVCC snapshot	Catalog-and-policy snapshot seen by a task operation.
Secondary index	Structural, lexical, vector, graph, recency, and outcome projections.
Page table	Multi-resolution module directory with handles and residency metadata.
Page fault	Evidence or module required but not resident for the next action.
Compiler IR	Typed Context IR independent of model or integration.
Source map	Serve manifest from prompt spans back to objects and evidence.
CDC stream	Ordered memory/source/task changes for caches, analytics, and replicas.
Row-level security	Scope- and origin-aware visibility enforced before candidate ranking.
Point-in-time recovery	Replay canonical events to a chosen catalog and policy state.

The competitive white space

Own the intersection

Jungle sits among vector databases, RAG frameworks, agent runtimes, memory services, code-intelligence systems, and data catalogs. Each category solves part of the problem. None should be dismissed; several will be partners or underlying engines.

Category	Core strength	Structural limit	Jungle relationship
Vector database	Fast approximate semantic discovery at scale.	Similarity index only; canonical truth, task state, and provenance lifecycle live elsewhere.	Use as a replaceable projection and integrate broadly.
RAG/orchestration framework	Flexible pipelines, retrievers, rerankers, and application composition.	Application-owned lifecycle; often request-scoped and host-specific.	Supply the durable memory and Context IR those pipelines consume.
Agent checkpoint/run time	Durable graph state, interrupts, retries, and workflow continuity.	Usually owns workflow state only.	Exchange checkpoints and receipts; avoid competing for orchestration UI.
Conversation memory service	User/profile memory extraction and personalization.	Often weak on exact source structure, bitemporal truth, or database operations.	Provide the trustworthy substrate and enterprise governance.
Code search/intelligence	Structural navigation, symbols, references, exact code context.	Commonly read-oriented, with no cross-model memory lifecycle.	Preserve Jungle's strongest ingest and evidence plane.
Knowledge graph/catalog	Relationships, lineage, ontology, temporal queries.	Can be heavy, centralized, and detached from runtime attention.	Project typed relations from the ledger; keep multiple query paths.
Provider-native context cache	Low-latency, discounted reuse within a model platform.	Provider-bound and prefix-oriented, with no durable system of record.	Compile provider-specific cache segments from portable objects.

The defensible intersection is: **exact repository evidence × temporal memory × task continuity × runtime residency × model-neutral serving × database-grade trust**. Jungle should deepen that intersection and integrate everything else.

Category language

Use **Memory Data Plane** as the company-level category. Use **Context Runtime** for the latency-sensitive serving subsystem. Use **Memory Kernel** for canonical storage, transactions, policy, and lineage. Use **Context Compiler** for provider-specific assembly. Use **Context ABI** for the portable object and serve contract.

Avoid leading with “AI context OS” in infrastructure procurement. The phrase is evocative but underspecified. It can remain the product vision: an operating system for machine context. The sellable contract should sound testable—durable memory, consistent snapshots, reproducible context, policy enforcement, and lower effective tokens per successful task.

IV · The north-star contract

Jungle is the memory data plane for LLM systems

Target architecture. Jungle is an always-on, model-neutral service that ingests source and experience, stores canonical typed memory, maintains rebuildable projections, and compiles task-specific working sets for agents. It is local-first by default, distributable when needed, and operable through stable contracts.

The public promise

For any context served to a model, Jungle can answer:

1. **What is it?** A typed object with stable identity and schema version.
2. **Where did it come from?** One or more source artifacts, observations, or explicitly attributed agents/users.
3. **When is it true?** A validity interval plus the transaction history of when Jungle learned, revised, or superseded it.
4. **Who may see or change it?** Scope, authority, privacy, retention, and write policy.
5. **Why was it selected?** A recorded plan containing constraints, candidate scores, dependency closure, and policy decisions.
6. **What did the consumer already have?** A confidence-weighted resident-set belief tied to a session and model boundary.
7. **How was it rendered?** A versioned compiler and provider layout with token and cache accounting.
8. **What happened next?** An optional causal link to actions, verification artifacts, user corrections, and task outcomes.
9. **Can it be reproduced?** A serve receipt naming the canonical snapshot and every projection/model version involved.
10. **Can it be removed or exported?** Lifecycle operations over canonical data plus auditable propagation to projections and replicas.

The non-goals

Jungle should not become a general-purpose relational database, a workflow orchestrator, a foundation model, an IDE, or an all-purpose knowledge-management UI. It should expose clean boundaries to each. Its specialization is the semantics and runtime of machine memory.

Recommendation. Publish this contract before marketing the next collection of features. It creates a stable lens for architecture decisions: a proposed component belongs in the kernel if it is required to uphold the contract; in a projection if it can be rebuilt; in the runtime if it affects serving; or in an adapter if it translates an external surface.

Architectural principles

Invariants that prevent the platform from collapsing back into a tool

1. **Canonical before clever.** Durable objects and events must remain interpretable without the embedding model, summarizer, ranking model, or agent that created their projections.
2. **Derived means disposable.** Every vector, summary, tag, graph edge, ranking feature, and compiled context declares inputs and a derivation version. If its lineage is complete, it can be rebuilt.
3. **Evidence is addressable.** A memory can point to precise source spans, artifacts, runs, and observations.
4. **Time is never overwritten.** Correction creates a new fact or status transition; it does not erase how a prior task reached its conclusion, except where legal deletion requires cryptographic or physical erasure.
5. **Authority cannot be laundered by retrieval.** An untrusted instruction does not become trusted because it was summarized, repeated, embedded, or selected many times.
6. **The model is a consumer.** It can propose memory mutations; deterministic policy, validation, conflict checks, and receipts govern durable commit.
7. **Residency is managed outside the prompt.** The runtime owns page-in, refresh, pinning, eviction, and cache alignment; the model may supply hints but cannot be the only controller.
8. **Consistency is explicit.** Fast stale reads may be valid for discovery; edits and high-authority decisions require stronger snapshots. APIs state the trade-off.
9. **Locality is portable.** Local-first operation and organization-scale deployment share the same object model, event semantics, and Context ABI.
10. **Outcome learning is bounded.** Learned selection policies may reorder eligible candidates but may not bypass security, freshness, evidence, or transaction gates.
11. **Protocol adapters stay thin.** MCP, SDK, CLI, proxy, IDE, and CI integrations translate the same contracts and do not become independent state machines.
12. **Operations are part of the product.** Backup, restore, migration, observability, resource limits, and failure drills ship alongside retrieval quality.

These principles imply a layered codebase and organization. Storage semantics should not import an MCP tool schema. A provider layout should not decide row visibility. A ranking model should not mutate canonical truth. An SDK should not be the only place idempotency exists. When boundaries are enforced in code and tests, the product can evolve without accumulating another generation of semantic debt.

V · The memory hierarchy

A managed address space for machine cognition



Figure 1 — Jungle’s managed memory hierarchy and rebuildable access projections.

The hierarchy separates **what is authoritative** from **what is cheap to serve**. Its levels describe latency, resolution, and lifecycle. A verified source span may be cold; an untrusted note may be resident. Authority, scope, and time remain orthogonal metadata enforced at every transition.

Level	Name	Typical contents	Service target	Persistence
L0	Provider-resident belief	Prompt/cache segments believed visible to the current model invocation.	Zero new tokens; confidence lookup in microseconds.	Ephemeral belief with receipts.
L1	Active task working set	Pinned files, decisions, constraints, current diffs, recent evidence, plan state.	Local low-millisecond metadata; bounded compile.	Session/task durable.
L2	Module page table	Multi-resolution manifests, outlines, summaries, signatures, contracts, dependency handles.	Always cheap enough to scan or route.	Durable, versioned projection.
L3	Canonical evidence store	Source bodies, typed memories, claims, artifacts, exact occurrences, validations.	Demand-paged with exact addressing.	Durable system of record plus canonical blobs.
L4	Historical and organizational memory	Prior revisions, task trajectories, superseded claims, shared patterns, policy history.	Queried selectively; batch and analytical paths.	Long-retained, governed.
L5	Archive and recovery tier	Cold events, backups, snapshots, large artifacts, legal holds.	Seconds-to-minutes restoration or replay.	Lowest-cost durable storage.

Running beside the levels are rebuildable **access projections**: lexical, structural, vector, graph, recency, co-change, ownership, and outcome indices. They do not form a “semantic level” because any of them can route to objects at L2–L4.

L0 and L1

Provider residency and the active task working set

L0 is a belief. Jungle cannot directly inspect a model's attention or guarantee that a provider retained a cache segment. It can only record what was sent, what a provider acknowledged, where compaction occurred, which prefix key is expected to match, and how confidence decays. The L0 record should therefore include model/provider, request and cache identifiers, content hashes, layout spans, send time, compaction events, cache TTL, and confidence.

Provider caches can materially change economics. Anthropic documents discounted cache reads relative to base input and different write multipliers; Google describes large discounts for reused context and prefix-oriented cache behavior. [R27–R28] Jungle's compiler should optimize for these mechanisms without making cached prompt text canonical. Stable, high-reuse module prefixes can be emitted separately from volatile task deltas.

L1 is the task's controlled working set. It holds the smallest set of objects required to continue the current trajectory: authoritative intent, open commitments, edit targets, current diffs, failing tests, recent decisions, active dependencies, relevant policies, and verification artifacts. Objects may be pinned, prefetched, dirty, stale, or evictable.

Required L1 metadata

Field	Purpose
object_ref + version	Prevent mixed revisions and enable exact refresh.
residency_state	Candidate, prefetched, resident, pinned, dirty, evicted, invalidated.
reason	Fault, dependency, user pin, predicted next step, policy, or verification.
first/last_touch and touch_count	Estimate locality and replacement priority.
consumer_receipts	Link serves to sessions, models, layout spans, and outcomes.
confidence	Represent uncertainty about continued L0 visibility after compaction or time.
cost_vector	Tokens, latency, cache write/read cost, and recomputation cost.
authority_floor	Minimum evidence/trust level required for the pending operation.

Replacement should be task-aware. Edit targets and user commitments stay pinned. Recently validated dependencies receive higher reuse probability. Large bodies can be demoted while their L2 handles remain. Repeated evict–fault cycles trigger a thrash alarm and temporarily expand or reorganize the working set.

L2 modules

The page table becomes Jungle's unit of locality

The current package and summary concepts should converge into a public **module** abstraction: a cohesive, versioned unit that binds code regions, memories, contracts, dependencies, owners, policies, and verification artifacts. A module is the unit Jungle can cheaply reason about before deciding whether to page exact evidence.

Module anatomy

- **Identity:** stable logical ID, content/revision hashes, namespace, schema version.
- **Scope:** organization, project, repository, branch, workspace, task, user, and privacy envelope.
- **Resolution ladder:** handle, synopsis, structural outline, operational brief, exact evidence manifest.
- **Contents:** code symbols/files, claims, decisions, task patterns, interfaces, invariants, known failure modes, tests, and artifacts.
- **Edges:** imports, calls, ownership, co-change, contradiction, supersession, dependency, and task-locality relations.
- **Contracts:** public interfaces, preconditions, postconditions, compatibility requirements, security boundaries.
- **Health:** freshness, evidence coverage, derivation version, validation status, last successful use, disagreement and churn signals.
- **Serving hints:** typical consumers, estimated token sizes, cache affinity, priority features, and common expansion paths.

Modules may nest, overlap, or represent cross-cutting concerns. A project can contain platform, feature, and concern modules; a security boundary or migration can legitimately span directory trees. The manifest records membership without forcing one physical hierarchy.

Target architecture. L2 routing happens in two passes. First, Jungle filters modules deterministically by scope, policy, freshness, task constraints, and structural relevance. Second, lexical/vector/graph/outcome projections rank eligible modules. Only then does the runtime expand a synopsis, dependency closure, or exact L3 evidence.

This design gives the system a natural locality boundary for prefetch and caching. If editing an API repeatedly faults its schema, implementation, tests, owner decisions, and rollout constraints, those objects should become one operational module or a stable linked cluster. Consumption telemetry can suggest reorganization, but a derived module never hides its membership or source lineage.

Migration note. Existing knowledge packages can seed modules. Do not invent a parallel abstraction. Add a versioned module manifest, map packages and covers into it, then make current retrieval and runtime paths consume the same representation.

L3 through L5—and the index sidecars

Durable evidence, historical truth, and recovery

L3 is the exact, durable evidence plane. It contains canonical source artifacts and typed memory objects at addressable versions: source spans, file bodies, schemas, issues, documents, user decisions, agent observations, claims, evidence links, validations, diffs, test results, and compiled artifacts. Large binary content may live in an object store, but its identity, metadata, checksum, policy, and lineage belong in the ledger.

L4 is historical and organizational memory. It preserves prior revisions, superseded claims, completed task trajectories, organizational decisions, reusable patterns, ownership history, benchmark results, and aggregated outcome evidence. L4 supports “why did this change?”, “what did we believe last quarter?”, and “which approach succeeded in comparable tasks?” Policy and temporal relevance determine promotion into a prompt.

L5 is archive and recovery. It holds compacted event segments, backups, periodic snapshots, cold artifacts, and legal holds. Its purpose is reconstructability and governance. A recovery drill should prove that L5 plus the canonical ledger can restore the active catalog and rebuild every projection.

Index sidecars: powerful, plural, subordinate

Projection	Best use	Failure mode to guard
Lexical / FTS	Exact identifiers, error strings, terms, and deterministic matching.	Vocabulary mismatch and tokenization drift.
Structural	Definitions, callers, dependencies, tests, scopes, and code boundaries.	Parser gaps and stale language support.
Vector	Semantic recall and vocabulary mismatch.	False authority, opaque drift, and filtered-ANN quality loss.
Graph	Dependency closure, provenance, contradiction, ownership, and impact.	Edge explosion and treating inferred relations as facts.
Temporal / recency	Current decisions, changed surfaces, validity and expiry.	Recency bias over authoritative older contracts.
Outcome	Prior contexts associated with verified success or failure.	Confounding, reward hacking, and feedback loops.

Each projection has its own generation receipt, quality metrics, source frontier, and rebuild status. Queries can declare which projections are required or optional. A failed vector refresh may degrade semantic recall while exact structural reads remain available; Jungle should expose that partial condition; a single “fresh” or “stale” flag would hide it.

Lifecycle policy

Promotion, demotion, faults, and memory metabolism

Memory becomes useful infrastructure when movement through the hierarchy is governed, observable, and reversible.

The control loop

1. **Observe.** Record task intent, active operation, resident belief, recent touches, source changes, policy, and provider constraints.
2. **Plan.** Determine the required authority and dependency closure; estimate the value, latency, and effective-token cost of candidate page-ins.
3. **Fault or prefetch.** Page exact evidence on demand; prefetch likely next modules at lower priority when confidence and budget justify it.
4. **Compile and serve.** Select resolution, arrange stable and volatile segments, emit provider-specific context, and write a receipt.
5. **Measure consumption.** Capture which references, tools, files, or claims the agent actually used, plus verification and user correction.
6. **Update residency.** Refresh touches, pins, confidence, dirty state, and invalidations; detect churn and repeated faults.
7. **Consolidate.** Promote durable learnings through validation, merge duplicates, record contradiction, and reorganize modules when repeated consumption demonstrates locality.
8. **Demote or expire.** Evict bodies while retaining handles, decay unverified observations, archive old trajectories, and execute retention or deletion policy.

Promotion must be typed. An agent observation can become a candidate claim, then a supported claim, then a corroborated organizational memory. It cannot silently jump from retrieved text to authoritative instruction. Recent memory-poisoning research shows that durable memory writes can create persistent compromise and that retrieved malicious memories can influence later actions. [R29–R31] The write path therefore needs origin binding, content classification, policy evaluation, corroboration thresholds, and quarantine.

Thrash and overload signals

- repeated fault–evict cycles for the same module cluster;
- high context churn with low novel evidence;
- rising tokens per verified action despite stable task complexity;
- L0 confidence collapse after frequent compaction;
- excessive prefetches never touched;
- stale invalidations arriving after serve;
- working sets that remain pinned beyond task boundaries.

A larger prompt is only one possible response. Jungle may restructure modules, raise synopsis quality, pin a small interface contract, split stable cache prefixes from volatile evidence, or ask the application to narrow the task. This is where the hierarchy becomes an active runtime.

VI · Canonical data model

One object system beneath every projection

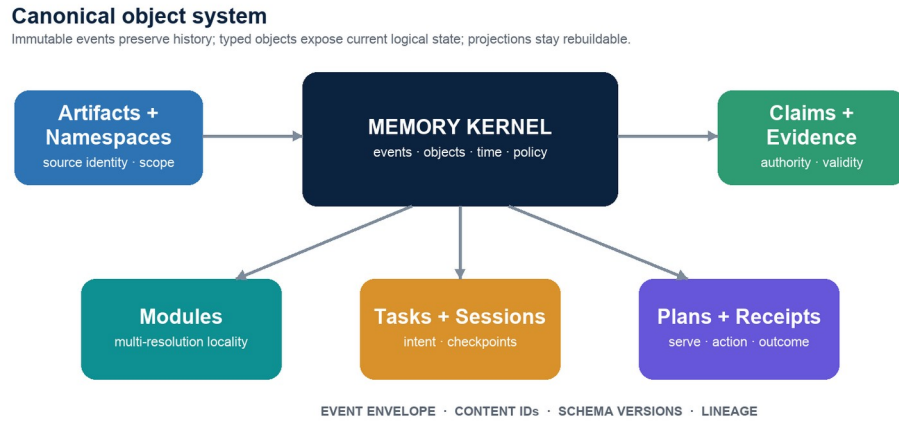


Figure 2 — Canonical objects are joined by an immutable, policy-aware event history.

The canonical model should be small enough to explain and rich enough to preserve semantics. Existing tables can remain implementation details during migration, but every public write should map to the following objects and events.

Object	Meaning	Canonical invariant
Namespace	Tenant, organization, project, repository, branch/workspace, task, or user boundary.	Identity and policy scope are explicit; no path-only tenancy.
Artifact	Versioned source or generated material: file, document, issue, diff, test result, binary, message.	Content checksum and source locator are immutable per version.
MemoryRecord	Typed durable observation, decision, preference, constraint, procedure, or summary.	Origin, authority, validity, retention, and derivation are mandatory.
Claim	A proposition that may be supported, contradicted, superseded, or expired.	Status is derived from evidence and validation events, never implied by repetition.
EvidenceLink	A typed relation from claim/memory to an artifact span, observation, or outcome.	The link preserves role, direction, version, and extraction method.
Module	Multi-resolution locality unit binding objects, edges, contracts, health, and serving hints.	Membership and every rendered resolution are versioned and reproducible.
Task / Session	Authoritative intent and execution continuity across agents and models.	Commitments, corrections, checkpoints, and operation versions are append-traceable.
ContextPlan / ServeReceipt	Selection decision and exact model-facing rendering.	Snapshot, policy, inputs, compiler, layout, cost, and consumer identity are recorded.
Action / Outcome	Tool, edit, decision, verification, correction, or task result caused after a serve.	Causal links are explicit and confidence-scored; correlation is not promoted to proof.
Policy	Versioned rule governing visibility, mutation, retention, export, and authority.	Enforcement decision and policy version appear in mutation and serve receipts.

Every state transition emits an immutable event envelope containing event ID, namespace, aggregate ID and expected version, event type and schema, valid time, recorded time, actor and origin, idempotency key, correlation/causation IDs, policy decision, and payload checksum. CloudEvents can supply a transport envelope; W3C PROV concepts can describe entity-activity-agent lineage. [R24, R26]

This does not require event sourcing every byte. Large artifacts and exact source bodies can remain immutable blobs referenced by the ledger. The key is that semantic state changes and projection frontiers are replayable.

Four orthogonal axes

Time, authority, scope, and temperature

Many memory systems collapse critical distinctions into a single relevance score. Jungle should preserve four independent axes and let policies compose them.

1 • Time

Use **valid time** for when a statement applied in the source world and **transaction time** for when Jungle recorded or superseded it. A dependency may have been vulnerable from June 1 to June 10 while Jungle learned the fact on June 4 and corrected the affected-version range on June 6. Both histories matter.

Required queries include: “as known at transaction T,” “valid at source time V,” “latest correction,” “changed since generation G,” and “show the evidence that was visible to task operation O.” Bitemporal semantics prevent a later correction from rewriting the forensic record of an earlier agent decision.

2 • Authority

Use a transparent ladder such as **untrusted input → observation → derived assertion → source-supported claim → independently corroborated claim → policy/owner-authorized instruction**. Confidence is separate: a model may be highly confident in an untrusted statement. Authority promotion requires typed evidence and an authorized event.

3 • Scope

Visibility is the intersection of tenant, organization, project/repository, branch/workspace, task/session, user/team, and privacy classification. Inheritance must be explicit. A user preference may be global to that user but invisible to another; a branch decision may not apply to main; a secret-bearing artifact may be eligible only for a sandboxed tool and never for a model prompt.

4 • Temperature

Temperature measures expected near-term utility and service latency: resident, hot, warm, cold, archived. It drives caching and placement but never changes authority or retention.

Unsafe collapse	Correct representation
“High similarity means trustworthy.”	Similarity feature + independent authority/evidence fields.
“Newest means correct.”	Recency feature + validity interval + supersession relation.
“Frequently used means globally useful.”	Task-local utility + declared scope + outcome evidence.
“In the prompt means resident.”	Serve receipt + provider signal + confidence-decay model.
“Deleted from vector index means erased.”	Canonical deletion event + projection tombstones + verified purge receipt.

Transactions and consistency

Memory writes must be safer than prompt generation

Jungle needs database-style semantics without pretending every workload requires serializable isolation. The API should expose a small set of explicit guarantees.

Write protocol

1. A client opens a mutation with namespace, actor, origin, purpose, idempotency key, and optional expected aggregate/catalog version.
2. Deterministic validation checks schema, scope, redaction, evidence references, authority ceiling, and retention policy.
3. Large artifacts are content-addressed and staged; the canonical event is appended before projections advance.
4. The commit atomically publishes logical object versions and returns a receipt with event sequence, catalog frontier, and projection status.
5. Projection workers consume ordered events idempotently. Failures are visible and retryable; they do not roll canonical truth backward.
6. A delete emits a tombstone and schedules policy-specific physical erasure, with verification across projections and replicas.

Read consistency classes

Class	Guarantee	Typical use
Snapshot	All required objects, policies, and projections align to a named catalog frontier.	Editing, high-authority reasoning, reproducible evaluation.
Read-your-writes	The caller observes its committed mutations, even if optional projections lag.	Interactive task continuation and checkpointing.
Bounded stale	Results may trail the canonical log by a declared time/event bound.	Discovery, autocomplete, background prefetch.
Best available	Partial projections are allowed and returned with degradation metadata.	Low-risk browsing during rebuild or outage.

Optimistic concurrency is the default: objects carry versions, writes may include `if_version`, and conflicts return both versions plus an explicit reconciliation path. Leases protect exclusive operations such as a migration or coordinated edit, but leases are time-bounded and fenced by monotonically increasing tokens so a paused agent cannot resume with stale authority.

Multi-object operations—record decision, update task checkpoint, attach evidence, and mark an action complete—should commit as a logical transaction. The local engine may use SQLite transactions initially; a distributed deployment may use PostgreSQL or another transactional store. The public semantic contract stays the same.

Recommendation. Add chaos tests for duplicate delivery, process death after log append, projection lag, stale lease holders, concurrent correction, and restore/replay. Infrastructure credibility is earned in the spaces between happy-path calls.

Unified modules and packages

Replace memory fragments with coherent, composable units

The long-term module model should unify three things Jungle currently treats separately: repository structure, packaged knowledge, and runtime working-set behavior. The result is a **versioned module manifest** backed by canonical object references.

Resolution ladder

Resolution	Target size	Purpose
Handle	20–80 tokens	Identity, role, health, and routing features.
Synopsis	100–300 tokens	Why the module matters; main interfaces and constraints.
Operational brief	500–1,500 tokens	Current contracts, decisions, failure modes, owners, and common paths.
Structural outline	Variable, bounded	Symbols, files, relations, tests, and exact expansion handles.
Evidence bundle	Demand-sized	Precise source spans, artifacts, claims, and verification required now.

The ladder is compiled. Each resolution declares which object versions and transformations produced it. A synopsis can be refreshed when only a dependency contract changes; exact evidence does not need to be re-embedded to remain canonical.

Module construction

Start deterministically from file/symbol structure, dependency boundaries, owners, packages, and explicit contracts. Then use co-change, task co-consumption, fault clusters, and outcome telemetry to propose merges, splits, or cross-cutting overlays. Human or policy approval is required for organization-wide canonical modules; task-local ephemeral clusters can be created automatically.

Modules can contain modules. They can also overlap: an authentication concern, an API surface, and a deployment workflow may share source objects. Membership edges carry role and priority so an object's meaning is not duplicated.

Module health contract

A module is servable only with a manifest version, source frontier, evidence-coverage score, derivation versions, policy scope, and degradation state. Health checks detect orphaned references, stale summaries, broken dependency edges, contradictory owner decisions, and modules whose synopsis no longer predicts their evidence.

Strategic payoff. Modules become the unit of caching, replication, sharing, billing, policy, and marketplace distribution. An organization can publish a verified “payments integration” module or import a vendor's SDK module without exchanging opaque prompt files. This is how Jungle grows from repository memory into a general memory substrate while retaining exactness.

VII · Reference architecture

Four planes, one contract

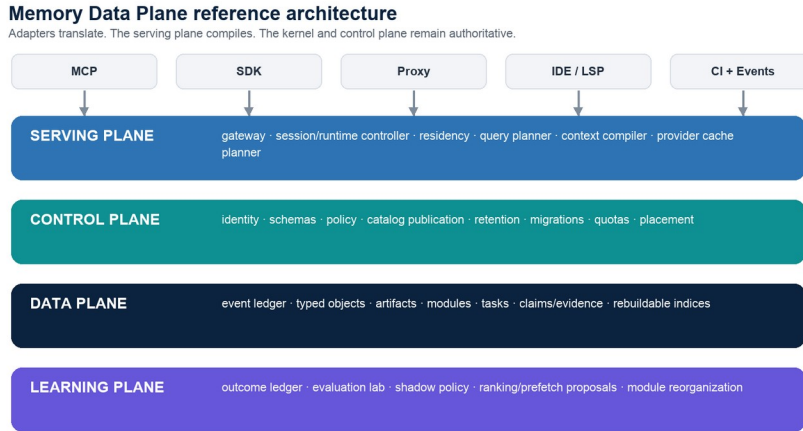


Figure 3 — Four planes share one Context ABI; adapters remain replaceable.

Data plane

The canonical event ledger, immutable artifact store, typed objects, task/session history, claims and evidence, module manifests, and rebuildable indices. This plane owns durable identity and replay.

Control plane

Namespaces, identity, schemas, policy, retention, encryption keys, catalog publication, migrations, quotas, placement, replication, and tenant administration. It decides what may exist, move, and be seen.

Serving plane

The runtime controller, residency manager, query planner, context compiler, provider cache planner, gateway, subscriptions, and adapters. It turns task state and canonical objects into low-latency working-set deltas.

Learning plane

Outcome/evidence ledger, evaluation lab, policy simulation, module reorganization proposals, and learned ranking/prefetch models. It may improve eligible selection and placement, but it cannot bypass data- or control-plane invariants.

The four planes share the Context ABI and observability schema. They can begin in one local process and separate operationally as load and tenancy require. This avoids premature distributed systems work while preserving clean seams.

Storage posture

Use storage engines by workload. SQLite remains credible for local canonical state and transactions. PostgreSQL is a natural managed control/metadata store for multi-tenant service. Object storage holds large immutable artifacts and event segments. A vector engine—LanceDB, pgvector, or another—serves semantic projections. FTS and structural tables serve exact and graph access. The platform contract must not expose any one engine as the ontology.

Key boundary. No adapter writes projections directly. No learned component writes authoritative claims directly. No index owns retention. No compiler changes visibility. All durable mutations enter through the kernel; all serves name a snapshot and policy decision.

Ingestion and change propagation

From source event to trustworthy projection

The ingest path is a streaming transaction, even when executed locally and synchronously.

1. **Observe a source change.** A connector emits an artifact event with source identity, upstream revision, actor if known, checksum, and cursor.
2. **Normalize and stage.** Jungle maps the event into a canonical artifact schema, performs content classification and redaction policy, and stores immutable bytes or references.
3. **Append canonical events.** The ledger records artifact/version changes, task/user annotations, and deletions with idempotency and ordering metadata.
4. **Extract typed structure.** Language parsers and document extractors produce symbols, sections, entities, spans, and exact addresses, each with extractor version and health.
5. **Advance candidate projections.** Lexical, structural, vector, graph, module, and summary workers consume the same event frontier. Each writes a receipt with inputs, output checksum, quality/degradation, and failure details.
6. **Validate publication.** Required planes satisfy catalog policy; optional planes may be marked degraded. A candidate generation atomically becomes active.
7. **Publish change data.** Ordered change feeds notify caches, L2 page tables, sessions with affected residents, analytics, and external subscribers.
8. **Invalidate and prefetch.** The runtime marks affected objects stale, computes impact, refreshes pinned modules, and schedules predicted page-ins.

Logical replication's snapshot-then-ordered-change model is a useful analogy: a consumer first receives a consistent base and then applies subsequent changes in order. [R9–R10] Jungle change streams should likewise identify a snapshot frontier and deliver resumable, monotonic cursors.

Partial failure policy

If embedding generation fails, exact source and structural projections may still publish if the catalog policy permits. If redaction or namespace policy cannot be evaluated, publication fails closed. If a summarizer fails, the prior summary can remain visible only with explicit staleness and source-frontier metadata. A consumer may require `all_required_planes_fresh=true` for high-risk operations.

This granular health model is more useful than a single stale flag. It lets the serving plane choose a safe fallback—exact lexical lookup instead of vector search, an older synopsis plus fresh evidence, or no serve at all—and tells operators exactly which capability is degraded.

The context compiler and runtime loop

Compile the next sufficient delta

The serving request is a **context transaction** with task state, pending operation, consumer capabilities, budget, consistency, and authority requirements.

Planning inputs

- task intent, commitments, checkpoint, open questions, and operation type;
- current catalog/policy snapshot and changed-object frontier;
- confidence-weighted L0 resident belief and L1 working set;
- eligible L2 modules, dependency closure, and exact L3 evidence handles;
- model context window, tokenizer, layout preferences, cache rules, tool schema, and cost schedule;
- latency deadline, token/effective-cost budget, risk class, and required evidence level.

Compiler passes

1. **Eligibility:** enforce namespace, policy, freshness, authority, and retention.
2. **Dependency:** build the minimum closure required by the pending operation.
3. **Resolution:** choose handle, synopsis, outline, or exact evidence per object.
4. **Delta:** subtract confidently resident content; refresh invalidated or low-confidence spans.
5. **Utility planning:** optimize expected task value subject to latency, cost, and risk.
6. **Layout:** place stable cacheable modules, task instructions, evidence, and recency-sensitive deltas for the target model.
7. **Guard:** enforce injection boundaries, provenance labels, content caps, and instruction/data separation.
8. **Emit:** return Context IR, rendered payload, expansion handles, and a signed serve receipt.

Feedback

After the invocation, the runtime accepts structured signals: referenced object IDs, tool calls, file touches, verification results, user corrections, compaction events, cache hits, and task outcomes. It updates residency and evaluation records but does not infer durable truth from model attention alone.

Core optimization metric: minimize **cache-weighted effective tokens per verified successful operation** while maintaining or improving correctness, latency, and evidence integrity. Raw prompt tokens are insufficient because cached and uncached tokens have different cost, and a cheap failed trajectory is not efficient.

Context ABI and protocol independence

One portable contract, many delivery surfaces

The Context ABI should define typed, versioned messages independent of storage engine and model provider.

Core operations

Operation	Contract
open_task / resume_task	Establish identity, intent, namespace, consistency, and consumer capabilities.
mutate / commit / abort	Transactional object and task-state changes with evidence and idempotency.
plan_context	Return candidate plan, cost, risks, missing evidence, and required expansions.
serve_delta	Compile and render the next working-set delta with a reproducible receipt.
expand	Page a specific module/object at a requested resolution and snapshot.
ack_consumption	Report actual references, tool/file touches, cache and compaction signals.
subscribe_changes	Stream resumable artifact, memory, task, module, and policy changes.
snapshot / export / restore	Operate lifecycle and portability over canonical records.
explain	Show selection, policy, lineage, authority, and staleness for an object or serve.

MCP can expose these operations as tools and resources. Its resource model includes listing, reading, change notifications, and optional subscriptions, making it a good interoperability surface. [R32–R34] But the ABI must also have first-party HTTP/gRPC or local IPC bindings, Python and TypeScript SDKs, an IDE/LSP adapter, a provider proxy, CI/event connectors, and an administrative CLI.

Provider adapters

Provider adapters translate Context IR into message roles, cache-control segments, tool schemas, structured output, and tokenizer-specific budgets. They report observed cache metadata and compaction behavior back to the runtime. They do not decide durable authority or rewrite module semantics.

Versioning discipline

Use additive schema evolution, capability negotiation, explicit deprecation windows, and golden compatibility fixtures. A serve receipt records ABI version, compiler version, provider adapter version, tokenizer, and policy snapshot. A client can therefore reproduce an old decision even after the default compiler changes.

Open ecosystem move. Publish the Context ABI and event envelope under a permissive specification while keeping the best compiler, hierarchy policies, operational control plane, and enterprise governance in Jungle. A portable contract increases adoption and makes Jungle the reference implementation.

VIII · Infrastructure posture

Security: the memory firewall

Persistent memory turns a transient prompt attack into a durable supply-chain problem. Untrusted content can be written once, retrieved later in a different task, summarized into apparently benign language, and elevated through repetition. Recent preprints demonstrate high memory-write success and material downstream action rates for poisoned memories; OWASP has also begun treating agent memory as a distinct security surface. [R29–R31]

Jungle should enforce a **memory firewall** at both write and read time.

Write-time controls

- bind every mutation to authenticated actor, connector, origin, purpose, and namespace;
- distinguish user instruction, source data, model observation, derived summary, and verified outcome;
- classify secrets, personal data, executable instructions, and cross-tenant references;
- cap the authority a source class may obtain without corroboration or owner approval;
- store candidate memories in quarantine when policy, evidence, or conflict checks fail;
- require idempotency, schema validation, evidence integrity, and signed connector receipts;
- preserve transformation lineage so summarization cannot erase the origin's trust ceiling.

Read/serve-time controls

- enforce row/scope policy before ranking; inaccessible objects never enter candidate features;
- separate instructions from quoted data in Context IR and provider rendering;
- compute an authority floor from operation risk—editing auth code requires stronger evidence than brainstorming;
- detect contradiction, expiry, unusual cross-scope promotion, and instruction-like content in evidence;
- redact or tokenize secrets for model delivery while retaining permitted exact access for tools;
- return policy and degradation receipts; nothing is silently filtered.

OPA's separation of policy decision from enforcement is a useful model, and PostgreSQL row-level security demonstrates default-deny row visibility when policies are enabled but absent. [R11, R35] Jungle can embed a policy engine locally and run the same bundles in a cloud control plane.

Threat model priorities

1. durable prompt/memory injection;
2. cross-tenant or cross-user leakage;
3. authority laundering through summaries or agents;
4. poisoned connectors and compromised source systems;
5. stale policy or deleted data remaining in projections/caches;
6. model-generated writes that exceed user intent;
7. malicious or buggy modules distributed through an ecosystem.

Security becomes part of the moat because Jungle can provide end-to-end origin and serve lineage that prompt-only systems cannot reconstruct.

Governance, lineage, and privacy

Make memory administrable over years

An organization needs to know which memory exists, why it exists, where it flows, how long it remains, and who can correct or remove it. Jungle should ship a governance layer over canonical objects.

Required governance capabilities

Capability	Minimum contract
Inventory and ownership	Browse objects/modules by namespace, type, source, owner, authority, retention, health, and consumer.
Lineage	Traverse source → extraction → claim/summary/module → serve → action/outcome, including model and policy versions.
Retention	Apply type/scope policies, legal holds, expiry, archival, and verified purge across projections and replicas.
Correction and dispute	Attach contradiction, owner correction, supersession, and adjudication without erasing prior transaction history.
Consent and purpose	Record allowed uses for personal or customer data and enforce purpose-limited serving.
Export and portability	Export canonical objects, events, evidence, modules, and schemas in open, documented formats.
Deletion	Tombstone immediately, prevent future serving, propagate purge, and emit a completion receipt with exceptions.
Audit	Query who wrote, read, served, exported, changed policy, or elevated authority, with tamper-evident retention.

W3C PROV-O supplies a portable vocabulary for entities, activities, and agents; OpenLineage offers run, job, dataset, and facet concepts familiar to data-platform operators; SLSA provenance demonstrates the value of attestations for generated artifacts. [R24–R25, R36] Jungle should map to these standards where useful while retaining memory-specific concepts such as authority, validity, resident belief, and serve receipts.

Privacy architecture

Encrypt data in transit and at rest; use per-tenant or per-namespace envelope keys at larger deployments; isolate sensitive projections; minimize model exposure; and make derived-data deletion testable. Content-addressed deduplication must never leak equality across tenants. Telemetry should prefer object IDs and aggregate metrics over raw context. Debug capture of prompts or evidence requires explicit policy, sampling, and short retention.

Recommendation. Treat exportability as a trust accelerator. Customers will make Jungle essential only if they can leave. The durable advantage should be the operating quality and the accumulated governed structure.

APIs, query model, and compatibility

Make the layer easy to adopt and hard to misuse

Jungle needs two public faces: a **high-level task/context API** for agent builders and a **lower-level memory/query API** for infrastructure teams. Both operate on the same identity, snapshot, policy, and receipt model.

Query families

- **Object reads:** fetch exact artifact, claim, memory, module, task, or receipt by stable ID and version.
- **Change reads:** list or subscribe to changes since a catalog/event cursor with resumable delivery.
- **Hybrid discovery:** combine lexical, structural, vector, graph, temporal, authority, and outcome features under deterministic filters.
- **Dependency queries:** expand callers, implementations, tests, owners, evidence, contradiction, supersession, and impact.
- **Historical queries:** inspect state as known at a transaction time or valid at a source time.
- **Explain queries:** return selection features, policy decisions, lineage, staleness, projection health, and excluded-candidate reasons.
- **Administrative queries:** inventory, retention, export, purge, replay, rebuild, migration, and health.

The planner can support a typed JSON expression first and later a concise SQL-like language. For example: request modules relevant to an edit operation, valid at the active branch revision, at least source-supported authority, visible to a named task, expanded through tests, and read at one catalog snapshot. The query compiles to deterministic eligibility plus ranked projections; it does not embed access policy in user-provided filters.

Compatibility layers

Surface	Primary user	Design rule
Local IPC + CLI	Individual developers, scripts, CI.	Zero-config local-first semantics and inspectable receipts.
Python/TypeScript SDK	Agent and application builders.	Typed async APIs, retry/idempotency helpers, offline fixtures.
HTTP/gRPC service	Platform teams and multi-tenant deployments.	Stable versioning, streaming changes, auth, quotas, and observability.
MCP server	MCP-compatible hosts.	Resources for addressable state; tools for transactions and context planning.
Provider proxy	Transparent model integration.	Automatic session/residency tracking without requiring voluntary tool use.
IDE/LSP adapter	Code agents and human developers.	Exact source identity, edit lifecycle, diagnostics, and branch awareness.
Connector/event SDK	Repositories, docs, issue trackers, data systems.	Idempotent cursors, origin attestations, backfill plus CDC.

PostgreSQL's stable frontend/backend protocol and extension model illustrate how a durable core can support many clients without making any one client authoritative. [R43–R44] Jungle should adopt that posture.

Reliability, SLOs, and observability

Operate context as a production dependency

The first SLOs should cover the promises users feel directly: can Jungle return the right snapshot quickly, is it fresh, can it explain itself, and can it recover? Targets below are proposed launch objectives; the current system has not yet been measured against them.

Service indicator	Initial objective	Hard guardrail
Core object-read availability	99.95% monthly, managed tier	Never serve cross-tenant data during degradation.
Exact object read latency	p95 < 100 ms local/cache; < 250 ms managed-region	Receipt reports tier and cache state.
Context plan + compile latency	p95 < 750 ms for bounded warm task	Deadline yields a partial, explainable plan.
Canonical ingest acknowledgement	p95 < 200 ms metadata-only local	Acknowledge only after declared durability point.
Required-plane freshness lag	p99 < 5 s local; < 30 s managed	Snapshot reads never mix generations.
Serve receipt coverage	100% of model-facing context transactions	No unreceipted privileged serve.
Evidence resolvability	99.99% of active authoritative references	Broken evidence blocks authority promotion.
Stale serve rate under snapshot mode	0 by contract	Fail or degrade explicitly.
Recovery point / recovery time	RPO ≤ 1 min; RTO ≤ 60 min managed target	Quarterly restore and projection-rebuild drill.
Deletion serve-block	< 60 s after accepted tombstone	Physical purge completion separately receipted.

Use OpenTelemetry traces to connect ingest, extraction, projection, planning, page-in, compiler passes, provider calls, tool actions, and verification. Emerging GenAI semantic conventions can inform span attributes, but Jungle should define stable memory-specific attributes now: namespace, object type, catalog frontier, authority class, plan ID, serve receipt, resident-hit/fault, resolution, token/cost vector, policy decision, and degradation. [R37]

Error budgets and degradation

An error budget turns reliability into a product trade-off. [R38] If freshness or evidence-integrity budgets are exhausted, freeze risky learned-policy rollout and prioritize correctness. Degradation should be capability-specific: vector search unavailable, synopsis stale, graph partial, provider cache unknown. Exact reads and canonical writes can remain healthy.

Operator dashboards should show task outcomes alongside systems metrics. A fast compiler that increases verification failures is unhealthy; a fresh index that repeatedly re-serves resident material is wasteful. Context infrastructure must observe both machinery and cognition-adjacent effects without claiming access to hidden model attention.

IX · Proof and economics

An evaluation lab for the whole trajectory

Jungle should maintain a continuous, versioned evaluation lab in which every architecture claim is tied to an experiment, dataset, baseline, cost model, and reproducible receipt. RAGAS and BEIR provide useful retrieval-oriented patterns; SWE-bench demonstrates outcome-based evaluation for repository tasks. [R40–R42] Jungle’s distinctive tests must extend across multi-step and multi-session work.

Five evaluation layers

Layer	Question	Representative metrics
Object integrity	Are identity, source, time, scope, and evidence correct?	Broken refs, temporal query accuracy, policy leakage, replay equality.
Projection quality	Do indices discover eligible evidence?	Recall@k, MRR/NDCG, structural closure, freshness lag, filtered-ANN recall.
Compiler quality	Is the served delta sufficient, compact, well placed, and reproducible?	Evidence precision/recall, redundant-resident tokens, omitted dependency rate, receipt replay.
Trajectory outcome	Does the agent complete verified work with less waste?	Resolve rate, first-pass tests, regressions, turns, latency, effective tokens per success.
Operational resilience	Does the system remain correct through failure and attack?	RPO/RTO, duplicate-event tolerance, mixed-generation prevention, poisoning success, tenant isolation.

Mandatory baselines

Compare the same tasks and model under: no added context; repository dumped to available window; vector RAG; best static curated pack; Jungle cold; Jungle warm with hierarchy; Jungle warm with learned prefetch. Use blinded task splits and freeze policies before scoring.

Workload matrix

Include cold one-shot questions, warm iterative edits, failures followed by diagnosis, branch changes, long idle/resume, model handoff, two-agent coordination, source contradiction, dependency upgrades, deleted data, projection outage, and hostile memory writes. Stratify by repository size, language, task type, and evidence locality.

Falsifiable claims

- Cold tasks must reach parity with the best retrieval baseline; hierarchy overhead cannot be justified by warm promises alone.
- Warm and multi-session tasks must show a growing advantage in verified success or effective cost.
- Page faults should concentrate after state changes and failures; a uniformly high rate signals thrash.
- Learned policy must beat deterministic policy on held-out tasks without weakening safety gates.
- Every claimed token reduction must report cache pricing, latency, task outcome, and confidence interval.

This lab is the feedback system that decides which hierarchy policies deserve production authority.

Economics and durable advantage

Optimize the cost of successful cognition

The economic unit should be a **verified successful operation**. A useful internal equation is:

Effective cost per success = (uncached input + cache writes + cache reads + output + model/tool retries + Jungle compute + storage/egress) ÷ verified successful operations.

The denominator prevents false efficiency. Removing context may cut tokens while increasing retries, regressions, or human review. The numerator makes provider cache discounts and Jungle's own projection costs visible.

Cost levers

1. **Resident delta serving:** avoid resending content already visible with high confidence.
2. **Resolution selection:** route with handles and synopses; page exact bodies only when needed.
3. **Stable cache segmentation:** isolate reusable module prefixes from volatile task state.
4. **Incremental projections:** recompute only affected structural, lexical, vector, graph, and summary artifacts.
5. **Outcome-aware prefetch:** spend latency/tokens where repeated trajectories demonstrate utility.
6. **Tiered storage:** keep canonical metadata and hot modules near serving; move cold artifacts and events to cheaper tiers.
7. **Model routing:** use smaller models or deterministic transforms for extraction, validation, and compilation where quality gates permit.

The compounding moat

The defensible asset is the governed graph linking source versions, typed memory, modules, context decisions, agent actions, verification, and outcomes across time. That graph improves module boundaries, prefetch, trust, and evaluation while remaining auditable. Competitors can copy a retriever; reconstructing causal, high-integrity operational history is harder.

The moat must remain customer-owned and portable. Jungle wins by being the best steward and runtime of that structure. Product tiers can monetize managed reliability, organization-wide connectors, policy/governance, private networking, evaluation labs, advanced compiler policies, and support. Local core semantics should remain easy to adopt.

Economic gate

By the end of the 3–9 month phase, demonstrate at least one production-representative workload where Jungle improves verified task success or time-to-resolution while reducing cache-weighted effective tokens per success by a material margin. Predeclare the margin and confidence threshold with design partners; do not set it after results arrive.

X · Migration and execution

Phase 0 · The first 90 days: establish the kernel contract

Contract first · runtime closure second · scale last



Figure 4 — The migration sequence avoids premature distribution and protects current workflows.

The first phase is a consolidation program. Its output is a stable semantic boundary around current capabilities.

Window	Deliverables	Exit evidence
Weeks 0-2	Ratify category language, public infrastructure test, object glossary, authority ladder, scope model, and canonical/projection ownership map.	Architecture decision record signed by product, runtime, storage, and security owners.
Weeks 2-5	Specify Memory Event Envelope v0, Context ABI v0, serve receipt, consistency classes, idempotency, object versions, and error taxonomy.	Golden protocol fixtures; two current adapters mapped without semantic exceptions.
Weeks 4-8	Build canonical write facade over existing local storage; route new task/memory mutations through it; append events and preserve backwards compatibility.	Crash/duplicate/conflict tests; deterministic replay into a clean database.
Weeks 6-10	Add projection registry and receipts: source frontier, derivation version, quality, required/optional, rebuild state.	Rebuild at least FTS/structural/vector projections from canonical fixtures.
Weeks 8-12	Emit universal context plan/serve receipts; add OpenTelemetry trace spine; expose explain and health APIs.	Every dogfood serve is reproducible and visible in one trace.
Weeks 10-13	Establish evaluation lab baselines and security threat suite; publish migration compatibility matrix.	Cold/warm baseline report, poisoning tests, no regression in current workflows.

Work explicitly deferred

- distributed consensus or global active-active writes;
- a new vector engine chosen without a measured bottleneck;
- learned prefetch in the authoritative path;
- marketplace/module distribution;
- broad UI redesign before contracts stabilize;
- rewriting all current tables into a theoretically pure event store.

90-day decision gate

Proceed only if Jungle can replay canonical task/memory state, rebuild declared projections, reproduce served context, and run current MCP/SDK/proxy paths through the same contract. If adapters still own divergent lifecycle semantics, pause feature expansion and finish the consolidation.

Phase 1 · Months 3–9

Close the runtime loop and earn production trust

1 · Unified modules

Map current packages, summaries, symbol/file structure, contracts, and knowledge records into Module Manifest v1. Provide handle, synopsis, outline, and evidence resolutions. Instrument module faults, touches, health, and serving cost. Let consumption telemetry propose task-local clusters while keeping organization-wide changes reviewable.

2 · Complete the Context MMU

Implement confidence-weighted resident belief, boundary prefetch, replacement policy, invalidation, dirty/pinned states, cache-aware segmentation, and thrash detection. Move session ownership into the daemon/runtime controller. The proxy and MCP adapter should use the same `open_task → plan → serve_delta → acknowledge` lifecycle.

3 · Database-grade operations

Ship snapshot and read-your-writes semantics, optimistic concurrency, fenced leases, ordered change feeds, export, backup, point-in-time restore, projection rebuild, schema migrations, quotas, and resource-pressure behavior. Run monthly fault injection and quarterly recovery exercises.

4 · Security and governance

Add enterprise identity mapping, default-deny scope policies, origin/authority enforcement, quarantine, secret minimization, retention, verified deletion, audit export, policy simulation, and an administrator inventory. Threat-test memory poisoning, connector compromise, cross-scope retrieval, and summary laundering.

5 · Evaluation and design partners

Recruit a small set of teams with long-running, multi-model software work. Run frozen baselines against cold, warm, resumed, failed, and multi-agent tasks. Optimize for verified outcome, effective cost, and operator trust. Do not report aggregate token savings without task-stratified results.

Exit criteria at month 9

- one Context ABI drives every first-party integration;
- canonical state and active catalog can be restored on a clean machine within target RTO;
- warm-task hierarchy beats the best static/RAG baseline on predeclared outcome/economic gates;
- no unresolved cross-tenant or authority-elevation critical findings;
- module health and residency behavior are observable, with bounded thrash;
- at least two external design partners run Jungle as an always-on dependency.

Only after these criteria should the team describe the managed service as production infrastructure.

Phase 2 · Months 9–24

Scale from product to platform

Months 9–15: organization and managed deployment

Introduce organization namespaces, immutable source contracts, workspace overlays, regional placement, private networking, managed PostgreSQL/object storage deployments, horizontal projection workers, tenant quotas, billing meters, and an administrative control plane. Preserve the local-first kernel and export format so a repository can move between laptop, self-hosted, and managed environments.

Add organization memory types—architecture decisions, policies, ownership, incidents, migrations, service contracts—and bind them to repositories and tasks through evidence, with no prose copied into every project. Make temporal “why” queries and policy-aware cross-repository modules first-class.

Months 12–18: adaptive serving under guardrails

Train ranking, prefetch, resolution, and module-reorganization policies from the outcome ledger. Use offline counterfactual replay, shadow decisions, canaries, and automatic rollback. Learned policies can affect utility ordering and cache placement; deterministic eligibility, authority, scope, evidence, and freshness gates remain immutable.

Months 15–24: ecosystem and open specification

Publish Context ABI 1.0, Memory Event Envelope 1.0, conformance tests, a connector SDK, and module packaging/signing rules. Support third-party storage and index adapters. Establish a registry for verified modules with provenance attestations, compatibility ranges, security review, and revocation.

Build partnerships with agent runtimes, model gateways, IDEs, data catalogs, observability vendors, and vector engines. Jungle should be the layer they plug into for durable context semantics.

Strategic option: portable organizational memory

Once the contracts are stable, Jungle can let organizations treat memory modules as governed infrastructure artifacts: versioned, tested, signed, promoted across environments, observed in use, and rolled back. A team could stage a policy or architecture module in shadow mode, validate it against real tasks, then promote it like a schema migration.

Month-24 proof

A model or agent host unknown at the start of this roadmap should integrate through the public ABI, resume an existing organization task, receive a policy-correct working-set delta, act against exact evidence, emit verification, and leave the memory system more useful—without any host-specific canonical state.

Operating model, risks, and kill gates

Organize around contracts and evidence

Create four accountable engineering ownership areas aligned to the planes: **Memory Kernel**, **Context Runtime**, **Trust & Control**, and **Evaluation & Ecosystem**. A small architecture council owns cross-plane invariants and ABI evolution; it should not become an approval bottleneck for internal implementation.

Risk	Early warning	Mitigation / kill gate
Overbuilding a distributed database	Roadmap dominated by replication before design-partner load.	Keep local semantics primary; distribute only measured bottlenecks.
Semantic duplication persists	Adapters or tables keep independent truth and lifecycle rules.	Block new feature paths until they use canonical write and receipt contracts.
Hierarchy adds latency without outcome gain	Cold/warm tasks regress against static packs or RAG.	Require predeclared parity and warm-advantage gates; simplify levels if not earned.
Modules become opaque summaries	Low evidence coverage, poor expansion precision, high churn.	Health thresholds, source maps, deterministic seeds, and rollbackable manifests.
Outcome learning rewards shortcuts	Lower tokens but more hidden regressions or security exceptions.	Verified outcomes, held-out evaluation, policy immutability, shadow rollout.
Security blocks usefulness—or vice versa	Users bypass policy, or high-risk content is over-served.	Risk-class profiles, explainable denials, sandboxed exact-tool access, red-team suite.
Protocol churn fractures the platform	MCP/proxy/SDK implement different concepts.	ABI conformance tests and thin adapter ownership.
Research language outruns reliability	“OS/database” claims without restore drills or SLO history.	Evidence-gated launch language; publish limitations and incident learning.
Customer lock-in erodes trust	Exports omit lineage or derived data; deletion is unverifiable.	Open canonical export, replay fixtures, portability tests, purge receipts.

Decision rights

- Product owns user-visible contracts and prioritization; semantic exceptions stay out of scope.
- Kernel owns canonical object/event semantics and migrations.
- Runtime owns serving and residency within kernel/control constraints.
- Trust owns policy vocabulary, enforcement requirements, and incident gates.
- Evaluation independently validates outcome and economic claims.

Stop conditions

Stop or materially narrow the strategy if cold parity cannot be maintained; warm advantage does not appear across representative multi-step tasks; operators cannot reproduce serves; policy overhead makes trusted use impractical; or the canonical model cannot support current workflows without pervasive adapter exceptions. A strong strategy states how it can be wrong.

XI · Decision memo

Authorize the shift now

Jungle has already discovered most of the hard nouns: intent, checkpoint, claim, evidence, package, catalog generation, residency, lease, provenance, and action. The final push is to turn them into one set of verbs and guarantees.

Authorize three immediate company-level moves:

1. **Declare the category.** Jungle is the Memory Data Plane for LLM systems; the Context Runtime is its serving engine. Plugins and MCP remain strategic adapters.
2. **Fund the 90-day kernel consolidation.** Freeze semantic proliferation long enough to ship the event envelope, Context ABI, canonical write facade, projection registry, serve receipts, replay, and baseline lab.
3. **Tie every subsequent feature to an infrastructure proof.** Each roadmap item must improve a named contract, SLO, recovery property, security control, outcome metric, or economic measure.

If executed, Jungle becomes the layer that lets models forget safely because the system remembers correctly. It makes context portable across models, durable across sessions, bounded under cost pressure, explainable under audit, and recoverable after failure. It turns an agent's prompt from an improvised pile of text into a compiled view over governed organizational memory.

The future-feeling experience is deceptively simple: an agent opens a task and already knows exactly what it is allowed to know, why it believes it, what changed, and what to load next. The infrastructure beneath that moment is the product.

Internal source register

- [J1] README.md — product model, local-first operation, bounded context loop, current claims and integration surfaces.
- [J2] backend/db_schema.py; backend/db.py; backend/retrieve.py — canonical schema inventory and current SQLite/LanceDB/FTS storage implementation.
- [J3] docs/plans/CONTEXT_MMU.md — working-set hierarchy, resident belief, page table, cold evidence, prefetch/thrash gaps, falsifiable hypotheses.
- [J4] docs/plans/COGNITION_LAYER_RESEARCH.md — Context IR, context-rot evidence, cognitive policies, prior-art synthesis.
- [J5] docs/plans/JUNGLE_PERFORMANCE_AND_CONTEXT_PLATFORM_ROADMAP_2026-07-22.md — atomic freshness, multi-resolution IR, sufficiency contracts, memory metabolism, firewall, transactional memory, portable ABI.
- [J6] docs/runtime-controller.md; docs/cwsr-trajectory-runtime.md — runtime ownership and causal working-set trajectory.
- [J7] docs/plans/PROXY_FIRST_MIGRATION_AUDIT.md — current surface inventory and migration toward daemon-owned lifecycle.
- [J8] docs/knowledge-runtime.md; docs/agent-protocol.md — memory, intent, checkpoint, handoff, and verification workflows.
- [J9] docs/plans/ORGANIZATION_JUNGLE_CLOUD_RUNNERS.md — organization indexing, overlays, storage/query, leases, security, and economics.
- [J10] docs/plans/PROD_READINESS_AUDIT_20260817.md — production-readiness and structural risks. Uncommitted at time of review; used directionally.

Sources and research notes

Primary and official sources

- [R1] Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *NeurIPS* 2020. <https://papers.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf>
- [R2] Liu et al., "Lost in the Middle: How Language Models Use Long Contexts," *TACL* 2024. <https://aclanthology.org/2024.tacl-1.9.pdf>
- [R3] Chroma Research, "Context Rot," model-length evaluation. <https://www.trychroma.com/research/context-rot>
- [R4] "Context Length Alone Hurts LLM Performance Despite Perfect Retrieval," *Findings of EMNLP* 2025. <https://aclanthology.org/2025.findings-emnlp.1264.pdf>
- [R5] Jiang et al., "LLMLingua," *EMNLP* 2023. <https://aclanthology.org/2023.emnlp-main.825.pdf>
- [R6] "Agent-Facing Information Design," 2026 preprint. <https://arxiv.org/abs/2605.23916>
- [R7] PostgreSQL, "Write-Ahead Logging." <https://www.postgresql.org/docs/16/wal-intro.html>
- [R8] PostgreSQL, "Introduction to MVCC." <https://www.postgresql.org/docs/14/mvcc-intro.html>
- [R9] PostgreSQL, "Logical Replication Architecture." <https://www.postgresql.org/docs/16/logical-replication-architecture.html>
- [R10] PostgreSQL, "Logical Decoding." <https://www.postgresql.org/docs/current/logicaldecoding.html>
- [R11] PostgreSQL, "Row Security Policies." <https://www.postgresql.org/docs/17/ddl-rowsecurity.html>
- [R12] PostgreSQL, "Continuous Archiving and Point-in-Time Recovery." <https://www.postgresql.org/docs/17/continuous-archiving.html>
- [R13] Packer et al., "MemGPT: Towards LLMs as Operating Systems." <https://arxiv.org/abs/2310.08560>
- [R14] Letta, "Attaching and Detaching Memory Blocks." <https://docs.letta.com/tutorials/attaching-detaching-blocks/>
- [R15] LangGraph, "Persistence." <https://docs.langchain.com/oss/python/langgraph/persistence>
- [R16] Mem0, "Building Production-Ready AI Agents with Scalable Long-Term Memory." <https://arxiv.org/abs/2504.19413>
- [R17] Rasmussen et al., "Zep: A Temporal Knowledge Graph Architecture for Agent Memory." <https://arxiv.org/abs/2501.13956>
- [R18] Pichay, "The Missing Memory Hierarchy in Large Language Model Systems," 2026 preprint. <https://arxiv.org/abs/2603.09023>
- [R19] "ContextWeaver," 2026 preprint. <https://arxiv.org/abs/2604.23069>
- [R20] "Virtual Context Compression," 2026 preprint. <https://arxiv.org/abs/2603.29678>
- [R21] "A Human-Inspired Memory Architecture for LLM Agents," 2026 preprint. <https://arxiv.org/abs/2605.08538>
- [R22] Denning, "The Working Set Model for Program Behavior," *CACM* 1968. https://denninginstitute.com/pid/PUBS/WSModel_1968.pdf
- [R23] Denning, "Thrashing: Its Causes and Prevention," 1968. <https://denninginstitute.com/pid/PUBS/thrashing-1968.pdf>
- [R24] W3C, "PROV-O: The PROV Ontology." <https://www.w3.org/TR/prov-o/>
- [R25] OpenLineage Specification. <https://github.com/OpenLineage/OpenLineage/blob/main/spec/OpenLineage.md>
- [R26] CloudEvents Specification. <https://github.com/cloudevents/spec>
- [R27] Anthropic, "Pricing and Prompt Caching." <https://docs.anthropic.com/en/docs/about-claude/pricing>

- [R28] Google Cloud, "Context Caching for Generative AI." <https://cloud.google.com/blog/products/ai-machine-learning/vertex-ai-context-caching>
- [R29] "Sleeper Memory Poisoning Attacks Against LLM Agents," 2026 preprint. <https://arxiv.org/abs/2605.15338>
- [R30] "Origin-Bound Authority for Non-Malleable Agent Memory," 2026 preprint. <https://arxiv.org/abs/2606.24322>
- [R31] OWASP, "Agent Memory Guard." <https://owasp.org/www-project-agent-memory-guard/>
- [R32] Model Context Protocol Specification, 2025-06-18. <https://modelcontextprotocol.io/specification/2025-06-18/index>
- [R33] Model Context Protocol, "Server Features." <https://modelcontextprotocol.io/specification/2025-06-18/server/index>
- [R34] Model Context Protocol, "Resources." <https://modelcontextprotocol.io/specification/draft/server/resources>
- [R35] Open Policy Agent Documentation. <https://www.openpolicyagent.org/docs>
- [R36] SLSA, "Provenance." <https://slsa.dev/spec/v1.2/provenance>
- [R37] OpenTelemetry, "General Trace Semantic Conventions" and GenAI span conventions. <https://opentelemetry.io/docs/specs/semconv/general/trace/>
- [R38] Google, *Site Reliability Engineering*, "Embracing Risk." <https://sre.google/sre-book/embracing-risk/>
- [R39] NIST, "AI Risk Management Framework." <https://www.nist.gov/itl/ai-risk-management-framework>
- [R40] Es et al., "RAGAS," *EACL* 2024. <https://aclanthology.org/2024.eacl-demo.16/>
- [R41] Thakur et al., "BEIR," *NeurIPS Datasets and Benchmarks* 2021. <https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/65b9eea6e1cc6bb9f0cd2a47751a186f-Abstract-round2.html>
- [R42] Jimenez et al., "SWE-bench," *ICLR* 2024. https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html
- [R43] pgvector, official repository and documentation. <https://github.com/pgvector/pgvector>
- [R44] PostgreSQL, "Frontend/Backend Protocol." <https://www.postgresql.org/docs/current/protocol.html>

Interpretation notes

Repository documents describe design intent as well as shipped behavior; the report labels schema/code evidence as current-state primitives and future documents as direction. Product documentation is used to establish public contracts. Preprints from 2026 are directional evidence and should not independently justify production decisions. Proposed SLOs, API names, phase timing, and architecture boundaries are recommendations to validate through the 90-day program.