

neXus

Peer-to-peer state fabric for agents and storage, from microcontrollers to data centers.

Taeho Lee*

SSCCS Initiative

ssccs.org

September, 2026

Abstract

neXus is a peer-to-peer state fabric on which heterogeneous systems (servers, agents, edge devices, data centers, and human-operated terminals) join the same board through a single lightweight binary, *nex*. Every attached node speaks the same coordinate-addressed^a I/O^b interface, regardless of its internal architecture. There is no hierarchy: any node can observe, act, or coordinate, and the fabric itself provides the shared blackboard. Identity resolves by arithmetic, and a filter resolves through the structural index; the network scales without a central orchestrator. This whitepaper specifies the FIH (Fact, Intent, Hint) record model, the coordinate identity, the runtime, and the verified behavior of the reference implementation, marking established results and open questions.

^asynTagma is the coordinate-space stack beneath the fabric: the Tagma whitepaper defines coordinate identity, and the reference implementation lives at docs.ssccs.org/projects/syntagma

^bchton is the IO materialization layer: it lands the coordinate space onto physical origins (memory, file, signal, network) and provides the flat key-space IO surface that neXus backends implement: docs.ssccs.org/projects/chton.

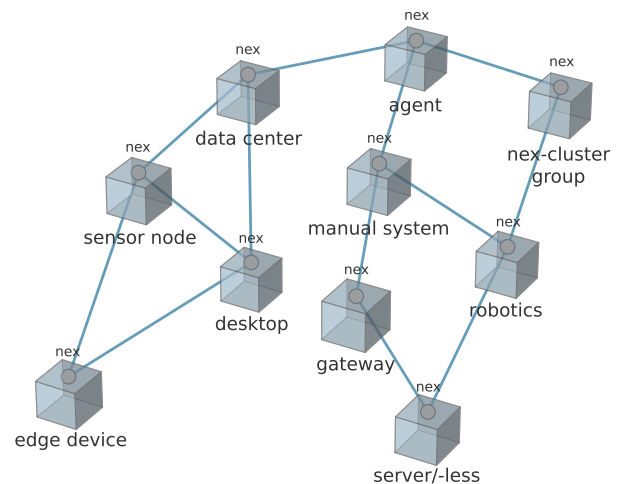


Figure 1: Heterogeneous software all attach to one nexus net: One coordinate space, one protocol, multiple directions, no orchestration.

Introduction

Generative agents, cloud functions, edge sensors, and human operators each speak different protocols and store state in incompatible backends. Coordinating across them typically requires a central orchestrator, an indexing layer, or a translation gateway, each adding latency, cost, and a single point of failure.

neXus replaces the gateway with a fabric. Any software that can attach to a storage backend or an agent becomes a peer, speaking the same FIH interface. *nex* is the universal attachment point: a binary that runs unchanged on Wasm, microcontrollers, cloud VMs, and bare metal. Once attached, the backend becomes a blackboard; the agent

*Founder & Architect; Corresponding author: lee@ssccs.org

becomes a peer. There is no master, no slave, no orchestrator, only a network of observers and actors where the observer is also an actor, and the actor is also an observer.

The effect is a flat ecosystem: a sensor node can write Facts, a data center can read them, an agent can propose an Intent, and a human-operated system can apply Hints, all through the same coordinate-addressed I/O. Because identity is spatial (coordinate, not hash) and records are immutable, the cost of resolving a record by identity is independent of the number of records, and the cost of coordinating across nodes scales with the number of attachments, not with the amount of data. The reference code is published under Apache License 2.0¹. Where a claim is a hypothesis rather than a measured result, it is labeled as such.

Model

neXus is a computing model before it is a system. The model states what computation is: a directed path over immutable records, in which the record is the state and the traversal is the computation. This section states the model itself and the minimal skeleton (Section) that tests it; the following chapters specify the record kinds (Section), the coordinate identity (Section), the storage (Section), and the runtime (Section) that realize it.

State. A knowledge state is a triple $K = (F, I, H)$, where F is the set of immutable Facts, I is the set of active Intents, and H is the set of active Hints. Every record carries a coordinate identity and a time coordinate, so K is a spatiotemporal state, and the state of the system is its record.

Operation. An operation is an Intent i that departs from a set of Facts $\text{src}(i)$ and is bounded by a set of Hints $\text{bnd}(i)$. Concluding the Intent produces exactly one new Fact f' :

$$(\text{src}(i), i, \text{bnd}(i)) \vdash f',$$

or, in the algebraic form the reference implementation uses, $F \times I \times H \rightarrow F'$. No Fact is overwritten; the state advances to $K' = K \cup \{f'\}$ with the concluded Intent removed.

Computation path. A computation is a sequence of such steps,

$$\pi = (f_0, i_1, f_1, i_2, \dots, f_n),$$

where each f_k is the Fact concluded by i_k from the Facts that precede it,

$$f_k = \text{conclude}(\text{src}(i_k), i_k, \text{bnd}(i_k)), \quad k = 1, \dots, n,$$

and because every step is recorded and no step depends on hidden state, replaying π from its recorded Facts reproduces f_n deterministically: replay equals computation. This equality is the property the model guarantees, the skeleton tests, and the storage layer makes measurable.

Closure. A new capability is a new Intent or a new Hint composed over the same immutable Facts. There is no separate execution model to invent: complex systems are built by composition, and the properties demonstrated by the minimal calculator are the properties every complex design must preserve.

Scale. The definitions hold at every scale, from a single agent to an ecosystem; an Observation at dimension N becomes a Hint at dimension $N - 1$, which is the property the project calls **scale-invariant** and which is specified with the scaling modes in Section .

¹neXus reference repository, github.com/ssccsorg/nexus, Apache License 2.0.

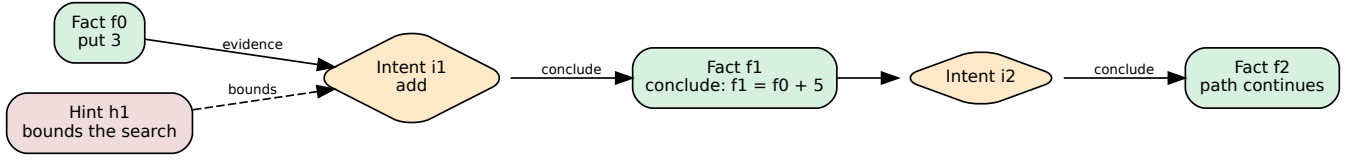


Figure 2: A computation in the nex model: Facts are consumed and produced by Intents under Hints; the recorded path is replayable, so replay equals recomputation.

The Minimal Skeleton and Computation as Record

As the first program in a new language is often a calculator, `nex-calc`² is the calculator of neXus: deliberately slow, and a minimal core proof. `put 3` writes a Fact at a coordinate, `put 5` writes another, `add` creates an Intent that is a directional vector between Facts, and `resolve` traverses from Fact to Intent to Fact and produces a new persistent Fact. The algebraic shape of the skeleton is

$$F \times I \times H \rightarrow F',$$

read as: a set of Facts, an Intent, and a set of Hints determine a new Fact. Nothing is added to the model to make the calculator work; the calculator is the model running unchanged. The skeleton is deliberately inefficient. Its purpose is not throughput but conformance: a complex system is designed by composing new Intents and Hints over the same immutable records, and the properties demonstrated by the skeleton are the properties that any complex design must preserve.

Two consequences follow, and they differ in epistemic status.

The first is a consequence of the record layer: an operation is a traversal recorded as an immutable Fact, the record is the state, and the traversal is the computation. This is established by the skeleton and by the record semantics of the store, and it is testable: replay of a recorded path must equal recomputation of that path, and no Fact is overwritten.

The second is a hypothesis under active test: the same record semantics can ground systems that learn. An agent records the Intents it proposes, the Facts it observes, and the Hints that bound its search; knowledge grows by accumulation of recorded paths instead of by overwriting continuous weights. The hypothesis is falsifiable. If reconstructing a recorded computation requires information outside the FIH record, such as timing, ordering, or hidden state, then computation-equals-record fails for that case. If path-accumulated learning cannot match a trivial stochastic gradient baseline on a toy problem at any cost, the learning extension is unsupported³.

The FIH Blackboard Model

The model uses three record primitives that form a complete basis for system state. The letters F, I, and H denote the three record kinds, and a neXus blackboard is any storage that can retain them.

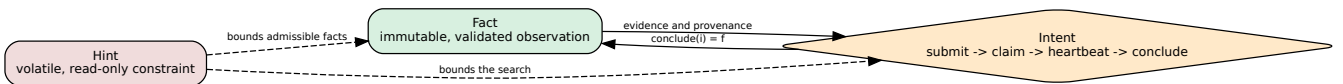


Figure 3: The recursive cycle of the three primitives. An Intent proposes an exploration; its conclusion produces an immutable Fact; Hints bound both.

²`nex-calc` source, github.com/ssccsorg/nexus/tree/main/apps/nex-calc

³Computation as knowledge: deterministic path learning (2026-09-02), [ssccs-nexus repository docs; github.com/ssccsorg/nexus/blob/main/docs/2026-09-02-computation-as-knowledge-deterministic-path-learning.md](https://github.com/ssccsorg/nexus/blob/main/docs/2026-09-02-computation-as-knowledge-deterministic-path-learning.md)

Primitives

A Fact is an immutable record of a validated observation. It is the output of a concluded Intent and carries the identity of that Intent, so every Fact is traceable to the hypothesis that proposed it. A Fact is never overwritten. An Intent is a stateful record of a proposed exploration. It carries references to the Facts from which it departs, a description of the operation to perform, and a creator identity. A Hint is a volatile, read-only record that constrains which Intents and Facts are admissible in a given scope. Hints can be garbage-collected.

A governing rule keeps the blackboard clean at scale: observers record what they find as Facts, and actors propose what they intend as Intents. Automated detectors therefore do not accumulate unclaimed Intents; they write idempotent Facts. The rule is stated as *observe as Fact, act as Intent*.

The three primitives map one-to-one onto the SSCCS ontological primitives⁴: Segment materializes as the immutable coordinate identity of a Fact, Scheme materializes as the structural relationships and memory layout declared by the blackboard scope, Field materializes as the constraint set carried by Hints, and Observation materializes as the evaluation that produces a Fact from an Intent.

Lifecycle

Every Intent follows the lifecycle

$$\text{submit} \rightarrow \text{claim} \rightarrow \text{heartbeat} \rightarrow \text{conclude}.$$

A submitted Intent is claimed by an agent, kept alive by heartbeats, and concluded with a result. Conclusion produces a Fact. A scheduler monitors heartbeats and evicts stale Intents that have not been claimed or that have stopped heartbeating. The lifecycle is the enforcement point for liveness: a crashed or dishonest agent is detectable by the absence of heartbeats, and an Intent that is never concluded leaves no Fact behind.

Formally, an Intent is a finite state machine over the states

$$\mathcal{S} = \{\text{submitted}, \text{claimed}, \text{active}, \text{concluded}, \text{evicted}\}$$

with the transition events claim, heartbeat, release, conclude, and timeout, and conclusion is the map that emits the resulting Fact and binds its provenance to the Intent:

$$\text{conclude}(i, r) = f', \quad \text{prov}(f') = \text{id}(i).$$

An Intent is evicted when its last heartbeat is older than a scheduler bound τ ,

$$\text{evict}(i) \iff t_{\text{now}} - \text{last}(i) > \tau,$$

so liveness is an enforced, measurable property of the record rather than an assumption about the agent.

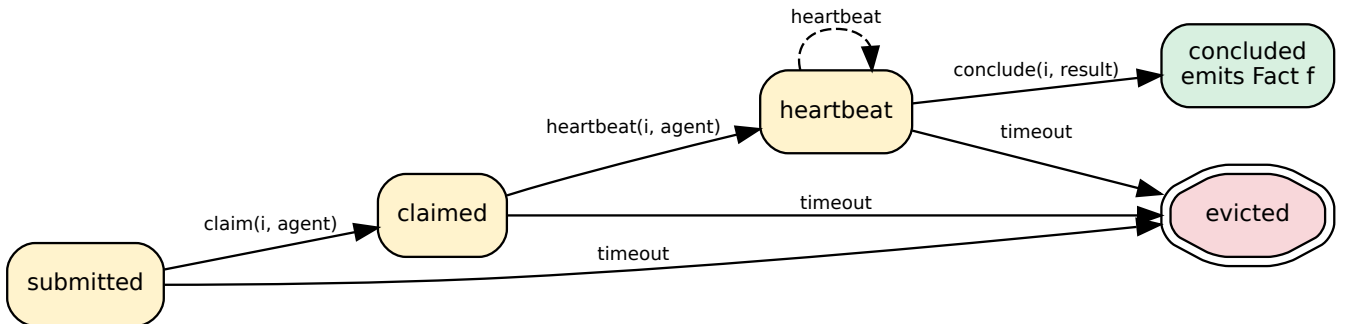


Figure 4: Intent lifecycle state machine. Concluded Intents emit Facts; stale Intents are evicted by the scheduler.

⁴SSCCS whitepaper, Schema-Segment Composition Computing System, ssccs.org/wp, DOI 10.5281/zenodo.18759106

Recursion and Scaling Modes

The FIH triad is **scale-invariant**: the same record structure and the same composition rules reappear at every scale, from a single agent to an ecosystem. A single agent runs a blackboard; a project runs a blackboard whose participants include that agent; an ecosystem runs a blackboard whose participants include projects. Each blackboard is a node that can contain sub-blackboards, and the composition rule is that an Observation at dimension N becomes a Hint at dimension $N - 1$. This nesting is structural, not logical: the same record semantics apply at every level.

Three scaling modes follow from treating the triad as a three-vector basis of state.

1. *Multi-blackboard composition*. Recursive scoping composes blackboards without a central store.
2. *Temporal accumulation*. State is a four-dimensional record: Facts are permanent, Intents are transient and leave a Fact residue on conclusion, and Hints are garbage-collectible. Every record carries a timestamp coordinate.
3. *Independent streaming*. Each primitive is an independent publish-subscribe stream, so distributed instances synchronize by FIH deltas without a shared database.

Coordinate Identity and Spatial Addressing

Record systems conventionally assign identity by hashing content and resolving collisions probabilistically. neXus derives identity from a fixed coordinate space supplied by Tagma⁵, a spatial primitive defined over a contiguous 16-bit Unicode block (U+AC00 through U+D7A3). Every value in that block is produced by a closed-form composition of three structural axes,

$$C(i, m, f) = \text{U+AC00} + 588i + 28m + f, \quad 0 \leq i < 19, 0 \leq m < 21, 0 \leq f < 28,$$

and exactly $11,172 = 19 \times 21 \times 28$ of the 65,536 possible 16-bit values are structurally valid. A coordinate is therefore simultaneously a 16-bit address and a three-axis point, and decoding is combinational: the coordinate of a value is its address, so finding a record is arithmetic rather than a table scan.

Identity Size and the Axis Model

An identity in neXus is an ordered tuple of coordinates. With $|\mathcal{A}| = 11,172$ valid values per coordinate, the space of N -coordinate identities has cardinality

$$|\mathcal{J}_N| = 11,172^N, \quad \log_2 11,172 \approx 13.45.$$

Two configurations are in use, and they differ in kind, not only in size.

The **domain identity** uses six application-defined axes, giving $|\mathcal{J}_6| = 11,172^6 \approx 1.9 \times 10^{24}$ addresses, about 2^{81} . Six axes are sufficient for domain-local record identity at a rate far above any realistic accumulation. Because the axes are part of the identity, identity is queryable: a filter on any axis prefix is a range query over coordinates, and an inverse index over the remaining axes resolves the matching identity set.

The **evidence-chain identity** uses twenty coordinates, $|\mathcal{J}_{20}| = 11,172^{20} \approx 2^{268.9}$. Since $268.9 > 256$, the twenty-coordinate identity is a full-injective encoding of a 256-bit digest into 20 base-11,172 coordinates; two distinct contents collide only up to SHA-256 collision resistance⁶. This configuration is an encoding of a digest, not a semantic axis decomposition; the axis-query property below applies to the six-coordinate configuration. The twenty coordinates are positional digits, not named axes, and are not independently filterable.

The six default axes are application-defined and ordered as `time_hi`, `time_lo`, `entity kind` (F, I, or H), `origin`, `creator`, and `serial`.

⁵Tagma whitepaper, docs.sscs.org/projects/syntaxagma/tagma/, DOI 10.5281/zenodo.21302508

⁶Coord ID 20 migration note, kept with the implementation rather than published here.

An axis query is an intersection of precomputed projections. For a query that fixes axis-value sets P_a on a set of axes Q ,

$$\mathcal{R}(Q) = \{r \in R : \text{coord}_a(r) \in P_a, \forall a \in Q\} = \bigcap_{a \in Q} S_a,$$

where S_a is the id set of the fixed value on axis a . In this design, each added axis intersects the candidate set before any record body is loaded, which is why the most selective query is the cheapest.

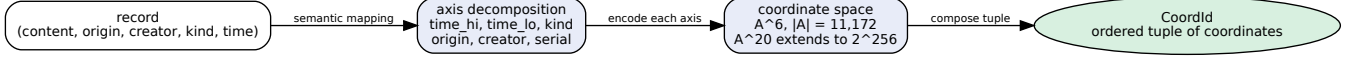


Figure 5: Content is placed on six axes; the ordered tuple of coordinates is the record identity, and axis filters become range queries over coordinates.

Semantic Identity and the Conflict Guard

Identity is semantic: the coordinate tuple is derived from the entity kind, the origin, the creator, and the content of the record, not from a random nonce. The same record produced by the same creator under the same origin converges on the same identity, which makes duplicate submission detectable. At commit time a content-hash conflict guard checks the target coordinate: if the coordinate is already occupied by a record with a different content hash, the write is rejected as a conflict; if the hashes agree, the write is idempotent. The occupancy lookup is $O(1)$ through the existing-content-hash index. After the L2 restructure the per-call conflict check measures 445 nanoseconds, against 135 to 390 milliseconds before the restructure⁷.

$$\text{id}(r) = \Phi(\text{kind}(r), \text{origin}(r), \text{creator}(r), \text{content}(r)),$$

and the content-hash guard accepts a write exactly when the coordinate is free or the hashes agree,

$$\text{accept}(w) \iff \neg \text{occupied}(\text{id}(w)) \vee h_w = h_e,$$

where h_w is the content hash of the write and h_e that of the existing occupant; the guard compares hashes so that content equality is decidable without comparing record bodies.

The change of identity derivation from six to twenty coordinates is breaking for stored records and has no automatic in-place migration tool; the documented path for pre-1.0 data is re-ingestion through the semantic layer.

Determinism and Auditability

Every Fact carries a provenance link to the Intent that produced it, forming a chain from conclusion back to hypothesis. Two properties make that chain auditable. Replay is deterministic: the state is the record set, and what a read builds over it, the record maps and the structural filter index, is a cache reconstructed from the records, so two nodes that hold the same record set answer the same set of records. Identity is content-derived: given the same content, origin, and creator, the same coordinate arises on any implementation. The content hash remains in the record as evidence; coordinates give an ordering over the identity space and an address, and do not provide preimage resistance. The distinction is maintained in the design: hashing is for evidence, coordinates are for structure. The two compose: a digest encoded as coordinates remains verifiable and becomes queryable by axis.

Retrieval imposes no sequence. An enumeration hands the records over a page at a time in the order the channel holds them, and a reader that needs an order states it, either by deriving it from the record's axes, which the time coordinate always carries, or by sorting the set it keeps.

⁷Content-hash conflict detection and the L2 restructure (2026-08-20), kept with the implementation rather than published here; The thin nexus principle (2026-08-20), [ssccs-nexus repository docs, github.com/ssccsorg/nexus/blob/main/docs/2026-08-20-thin-nexus-principle.md](https://github.com/ssccsorg/nexus/blob/main/docs/2026-08-20-thin-nexus-principle.md)

Storage Architecture

The storage architecture applies one principle at two boundaries: *storage down, semantics up*. Low-level storage and I/O move outward to the external chton project at the crate boundary, and record bodies, identity, and relationships move to application-owned record maps at the internal layer-2 boundary. What remains inside the core is semantics: the FIH lifecycle, the conflict guard, the inverse index, the structural filter index, deterministic replay, and per-channel partition scans.

The Layer-2 Restructure

The unified tree index was reduced to a primitive: a six-axis structural filter index that maps each axis combination to a set of record ids over low-cardinality axes (time, entity, origin, creator, status), a filter projection of the identity axes. Record bodies live in hash-map record maps owned by the application layer. This is the standard index-to-heap pattern, applied to the whole store. The measured outcomes of the restructure are:

- per-fact heap footprint dropped from about 3.4 MB to about 40 KB on average, with a steady-state marginal cost around 0.9 KB per additional fact;
- the conflict check dropped from 135 to 390 ms per call to 445 ns;
- index memory is bounded by axis cardinality, not by record count.

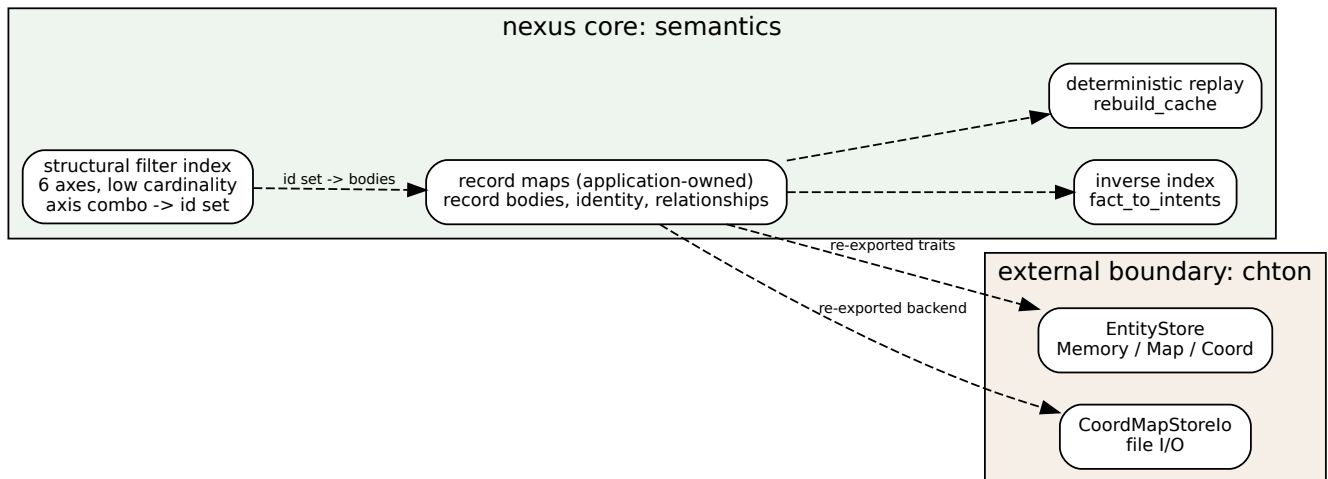


Figure 6: Layer-2 layout: a six-axis structural filter index maps axis combinations to id sets; record bodies and the inverse index live in application-owned record maps.

At the crate boundary the `EntityStore` trait family and the `CoordMapStoreIo` file backend are owned by chton and re-exported by the `fi` crate so that consumers keep a stable crate path. The re-export is a compatibility shim: it fixes the public path, not the coupling, and dependency drift between the two repositories is an accepted coordination cost.

Storage Primitives and Tiers

The store surface consists of three pure synchronous primitives with a zero-async core.

Primitive	Operations	Typical backing
Key-value	read, write, delete, list	warm records, SQLite
Blob	put, get, delete, list	cold bodies, Parquet, DuckDB
Object	read-state, compare-and-swap	hot coordination, WAL

Compare-and-swap is the only coordination primitive; it is supplied by whatever the backend offers (SQLite WAL, a blockchain, or an object store). Storage is tiered: hot records use object compare-and-swap, warm records use key-value, and cold records use blobs with an advancing flush cursor, optionally composed through a composite cold-storage layer.

Asynchronous runtimes (browser workers, Cloudflare Workers, canisters) cannot host the synchronous core directly. The AsyncStorage bridge hydrates the backend into memory, runs the synchronous core against dirty-tracked buffers, and drains only the mutated keys back through the async boundary. The same core files serve synchronous native builds and asynchronous hosted builds without modification, and the hosted instances exercise it at the storage boundary, where the record layer's own tests pass with no core file changed.

Memory Bound

Because the filter index is a fixed structure over axis combinations, its memory depends on the number of distinct axis values, not on the number of records. Let A be the number of distinct axis values and S the measured footprint per value, one dense leaf of 11,172 slots plus one branch, about 349 KB. Index memory is then

$$M_{\text{index}} \approx S \cdot A,$$

while record heap grows with record count R and marginal cost $m \approx 0.9$ KB, so

$$M_{\text{total}} \approx S \cdot A + m \cdot R.$$

At 100,000 and 1,000,000 records the index remains approximately 264 MB, while the record heap grows with R ; the flatness of the index is the property that keeps lookup fast and memory predictable as knowledge accumulates. The bound is linear in distinct axis values, so application axes must stay bounded in practice; a high-cardinality axis, such as per-record conclusion ids, inflates the index linearly.

This is the indexed profile. A deployment that resolves records by identity alone, or one whose part cannot hold the index, builds none: its memory is the record heap, and a read holds a page of keys and one record rather than the volume. The bound is visible in the walk's peak, measured on the OS-less tier over volumes the target did not write with one run each: three times the volume, from 32 to 96 records, moves the peak by 16 bytes, where reading the enumeration whole moved it by 27,648 bytes over the same span.

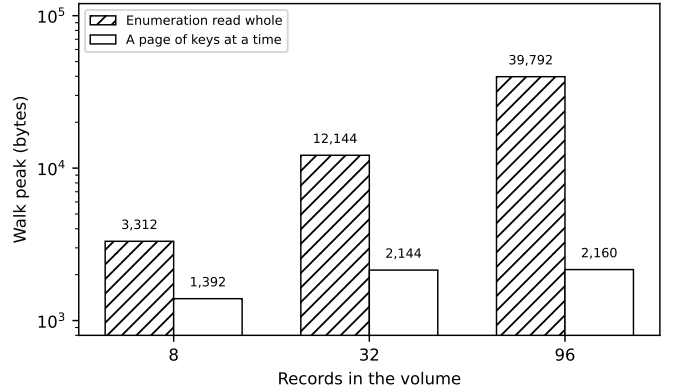


Figure 7: The record walk's peak, measured on the OS-less tier over volumes the target did not write, one run each. Reading the enumeration whole grew it with the volume (3,312, 12,144, 39,792 bytes); reading a page of keys at a time does not (1,392, 2,144, 2,160). Three times the volume moves the peak by 16 bytes, where the whole enumeration moved it by 27,648.

Runtime and System Architecture

The runtime is a peer fabric, not a client-server platform. All participants, whether verification engines, editors, synthesis tools, or sensors, are equal peers that interact only through the FIH blackboard interface. There is no privileged orchestrator; a peer is defined by what it reads and writes, and the same binary can act as a blackboard, a peer, or both.

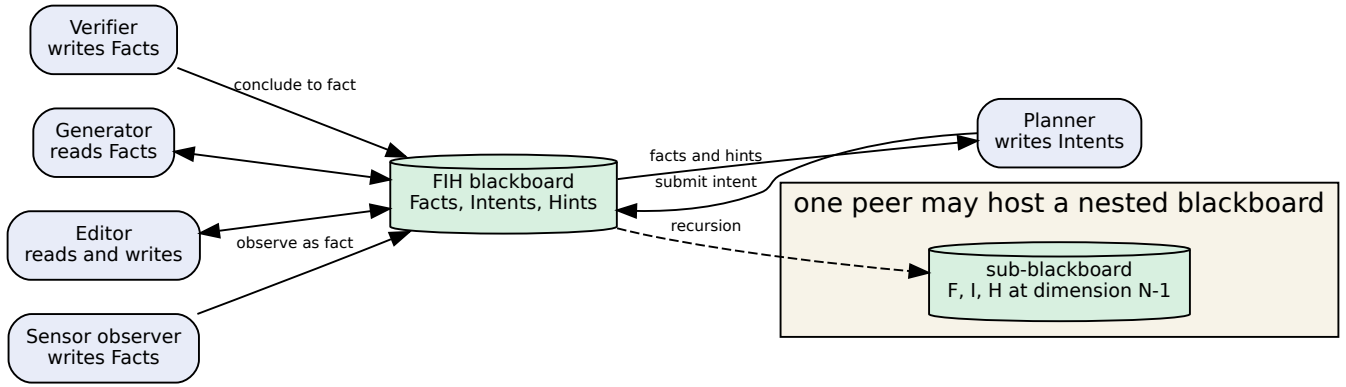


Figure 8: Peers around a blackboard. Every peer can itself host a nested blackboard, which makes the fabric recursive at every scale.

The Daemon Layer: nexd and nex-server

Execution is split into two processes with different responsibilities. `nex-server` is the blackboard engine: it owns the FIH state and answers record operations. `nexd` is a pure supervisor: it spawns `nex-server` as a managed child, waits for its socket, respawns it after a crash with the original command, and terminates it with `SIGTERM` and a bounded wait on shutdown. The two processes communicate over Unix domain sockets with line-delimited JSON-RPC 2.0, one object per line. The blackboard socket defaults to `/tmp/nex-server.sock` and the supervisor socket to `/tmp/nexd.sock`, each overridable by environment.

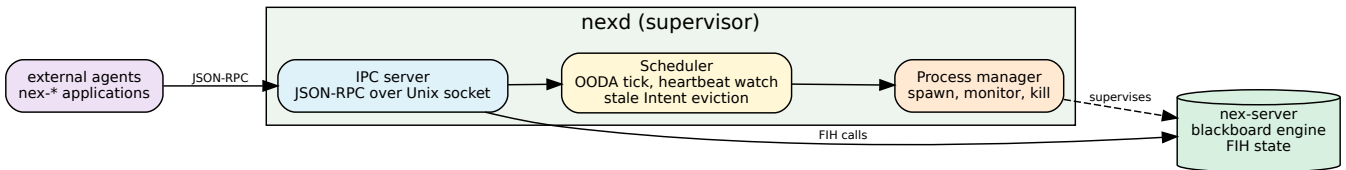


Figure 9: `nexd` runs three concurrent subsystems above the shared blackboard engine.

The supervisor exposes three responsibilities: local IPC for clients, an OODA scheduler that monitors heartbeats and evicts stale Intents, and a process manager for agent applications. FIH methods are forwarded verbatim to `nex-server` through the client SDK; supervision methods (`spawn_agent`, `list_agents`, `kill_agent`) are handled locally. `nexd` keeps no compile-time dependency on the blackboard engine; the wire contract is the boundary, and a change to a method name, parameter, or error code is a breaking change to the protocol.

The Five-Layer System Architecture

At the system level the fabric is organized in five layers, from retrieval to governance.

Layer	Logical component	Core responsibility
1	Knowledge graph engine	Vector, graph, and temporal hybrid retrieval over documents, entities, simulations, and sensor traces
2	Artifact ingestion pipeline	Engine-agnostic sync from object store through workers and queues
3	Agentic research loop	Stigmergy-based coordination of Planner, Verifier, and Generator
4	Learning loop	On-policy RL over blackboard traces with knowledge support rewards

Layer	Logical component	Core responsibility
5	Contract governance	Admission, constraint evaluation, evidence chains, and the research economy

Layer 1 decomposes artifacts into typed relationships and community clusters and supports naive, local, global, and hybrid retrieval strategies. Layer 2 abstracts storage backends behind a minimal interface so that SQLite, blockchain, and cloud databases are interchangeable without core changes. Layer 3 coordinates agents by stigmergy: detectors such as gap and contradiction finders apply count-based heuristics and record findings as idempotent Facts, and every action appends to an evolving memory for conflict-free concurrency and replay. Layer 4 batches rollouts for group-normalized advantages and updates the planner with an on-policy clipped objective and KL penalty (Flow-GRPO, a variant of group relative policy optimization); the learning loop is in the design stage and is not yet implemented in the core. Layer 5 gates what may be written, evaluates constraints, and keeps evidence chains; its economic rules are described in the governance section.

Design Principles

The posture of the runtime follows five principles.

- Peer runtime, not a platform: no marketplace, no extension rent, and no central gatekeeper; a producer runs nex on its own infrastructure and decides what its work is worth.
- Engine-agnostic and zero lock-in: synchronization endpoints isolate the system from specific backends, and every component is replaceable with an open equivalent, with dependency drift between the two repositories accepted as a coordination cost (see Section).
- Research-first: the system is optimized for the cycle of hypothesize, validate, and publish, across digital and physical domains.
- Boundaryless by design: physical-digital extension is a consequence of the engine-agnostic pattern, not an architectural rewrite.
- Invisible infrastructure: nex is measured by the success of the instances built on it; each instance is a standalone project with its own identity and an implicit nex extension.

Knowledge Accumulation and Governance

Three Fabrics in One

From the perspective of AI agent architectures, neXus provides three fabrics in one: a semantic fabric that records the research process and runs retrieval and gap detection on accumulated facts at zero marginal inference cost, a governance fabric that gates what may be written, evaluates constraints, and keeps an evidence chain, and a storage router fabric that lets agents, editors, and verification engines coordinate through one shared interface regardless of which backend holds the state. Each fabric is detailed in this chapter and in the architecture chapter. The industrial utility follows: accountable collaboration, since every conclusion is traceable to its evidence; affordable operation, since routine coordination costs no LLM inference; and sovereignty, since organizations run the fabric on their own infrastructure.

Zero-Marginal-Cost Retrieval

Model inference is confined to the generation of knowledge branches. Traversal, gap detection, and reporting operate on accumulated Facts through the filter index, so their cost does not scale with inference calls. The economic consequence is structural: reading and checking accumulated knowledge has zero marginal **inference** cost, and inference is spent only where a new branch of knowledge is produced. Retrieval itself is not free. Through the structural filter index its arithmetic and I/O cost scales with the size of the matching subtree; a query the index cannot prune, or a deployment that builds no index, pays a scan whose cost scales with the record count, which Section records as a boundary.

Stigmergic Compounding

Agents coordinate indirectly through the traces they leave, a mechanism known as stigmergy. A producer who solves a problem deposits verified knowledge into the shared store, and other producers inherit that knowledge and build on it. The ecosystem grows by recursive compounding of independently verified progress. In this model the accumulated record plays the role that difficulty plays in proof-of-work chains: value compounds with knowledge depth rather than with raw compute, and the same store serves as the coordination medium and the audit trail.

The governance layer recognizes five economic contributions: gap discovery, hypothesis submission, experimental validation, concept drift detection, and knowledge ingestion. Validated hypotheses return stake plus reward; falsified hypotheses are slashed. Write admission, constraint evaluation, and evidence chains are implemented in the core; the on-chain economy is staged in the hosted deployments.

The ULHM Verifier

The Universal Latent Homeomorphic Manifold (ULHM) framework extends the verifier to cross-modal domains. A homeomorphism is a continuous bijection that preserves topological structure, and applying it to a learned latent space assumes that space is a manifold; verifying that assumption is an open question. The verifier validates a candidate mapping $\phi : X \rightarrow Y$ between modalities with three canonical loss terms:

$$L = \lambda_c L_c + \lambda_t L_t + \lambda_w W(P_X, P_Y),$$

where L_c is a continuity loss bounding local perturbation, L_t is a trust loss preserving neighborhood structure, and $W(P_X, P_Y)$ is the Wasserstein distance aligning the global distributions:

$$L_c = \mathbb{E} d(\phi(x), \phi(x')) \quad \text{for small } d(x, x'), \quad L_t = \mathbb{E} d(N(\phi(x)), \phi(N(x))).$$

This is the mathematical basis for validating that a simulation result, a robot trajectory, and a physical measurement describe the same underlying structure. The term homeomorphism is used here in its strict topological sense, distinct from the structural scale-invariance of the blackboard across scales. Two enablers extend the framework: an episodic knowledge graph that turns the evolving memory into a temporal, provenance-ordered record for cross-modal coherence and physical reproducibility, and a homeomorphic bridge that supports semantic-guided recovery of partial physical observations and zero-shot compositional reasoning across simulation and hardware. The extension is in the design and validation stage; the core record model it verifies is implemented.

Verification and Correctness

Correctness is argued at four levels: exhaustive enumeration of the coordinate space, adversarial simulation of the coordination protocol, an exchange-contract proof sketch, and portability verification of the OS-less profile.

Exhaustive Coordinate Enumeration

The 16-bit coordinate space contains 65,536 values, of which exactly $11,172 = 19 \times 21 \times 28$ are structurally valid under the closed-form composition. A verification harness enumerates all 65,536 inputs and validates the decoder over the full space in milliseconds; the valid subset is exhaustive, not sampled. Because every record identity is composed from valid coordinates and invalid coordinates are rejected at construction, collision freedom holds by construction for the identity space rather than by probabilistic guarantee, and every storage layout that addresses records by coordinates inherits the property.

Adversarial Simulation

The simulator implements the same core traits as the production backends at two depths: at the trait level against a reference storage runtime, and across the async boundary through an HTTP emulation of hosted worker I/O. Acceptance phases 0 through 3 are complete; phase 4 is open. Seven attack scenarios are exercised.

- Direct free-riding: reading without contributing.
- Evasive free-riding: contributing minimal or duplicated content.
- Falsified contribution: fabricating results or evidence.
- Poison injection: inserting misleading facts into the shared store.
- Intent spam: flooding the blackboard with unclaimed Intents.
- Replay: re-submitting recorded actions.
- Sybil: impersonating multiple identities.

Four game-theoretic scenarios run as CI gates with explicit thresholds.

Scenario	Condition	Gate
Basic cooperation	200 turns	score of tit-for-tat exceeds score of always-defect
Evolutionary stability	500 generations	tit-for-tat remains among extant strategies
Generous forgiveness	repeated play	cooperation rate above 0.6
FIH intent lifecycle	repeated play	honest completion above 0.7, false conclusion below 0.1

The simulator emits a violation matrix, a signed proof artifact, and Hint generation. The same FIH domain recurrence maps the simulator to ExaVerif, the exhaustive verification engine for instruction extensions: both reduce a target to a FIH record set and evaluate constraints over it, so the simulator's core traits are the verification engine's core traits.

The Exchange Contract

The proof sketch reduces the coordination surface to a single function, `exchange(session_token, query_spec)`, under a rule that reading requires writing: provided data commits as Facts before search results are returned, and no read-only fact endpoint exists at the function level, which makes free-riding structurally impossible at that boundary. The oracle problem is addressed by execution as proof: an observer signs the act it performs, and the signature is carried in the record. The threat model covers free-riding, data poisoning, and governance collapse. Status is a design-level draft; implementation is scheduled after the core protocol stabilizes.

OS-less and Portability Verification

The core compiles for environments without an operating system. The Wasm build is exercised in CI, the core time and file abstractions are narrowed to two traits, `Now` and `FileIo`, and the coordinate core builds with `no_std`. The OS-less profile is the bound on the runtime surface: the same record semantics that run on a server also run where an operating system cannot, and the remaining work is a bare-metal launcher rather than a change to the record layer. The constrained profile drops the structural filter index, which is a build feature, and reads the volume a page of keys at a time, holding a page and one record; both shapes are exercised in CI.

Measured Performance

The numbers in this section come from the benchmark suite of the reference repository⁸. Runs execute on an Apple M1 (ARMv8.4 Firestorm cores at 3.2 GHz) in release mode; the suite is Criterion-based with a sample size of ten and reports the median. Fixtures run in memory on the simulation I/O backend, synchronously, with one flush boundary per workload: 50,000 facts across 50 origins and 20 creators for the core benches, a controlled 10-day by 10-origin by 10-creator grid for the scale runs, and the real embedded document manifest for the lifecycle runs. Measurements are marked with their scope: some compare the restructured index against a previous ceiling, and others compare the structural filter against a record-map scan. The core baseline is dated 2026-08-20 and the scale and lifecycle runs 2026-08-27; figures published earlier than the layer-2 restructure are meaningful only within a run, and the structural and scan result sets are asserted equal by the test harness.

Identity Lookup and Tree Walk

Ten thousand identity lookups, three representations.

Representation	Total	Per lookup	Note
Hash index	120.9 μ s	12.1 ns	baseline
Tagma dense packed array	60.8 μ s	6.08 ns	2.0x faster
Tagma tree	533 μ s	53.3 ns	carries prefix navigation

Lookup cost scales with coordinate depth, not with volume: ten thousand and 10^{24} records cost the same number of dereferences because the coordinate is the address. With d the coordinate depth and c_d the per-level dereference cost,

$$T_{\text{lookup}} = d \cdot c_d,$$

which is independent of the record count $|R|$.

A full walk of a 50,000-entry tree takes 540 ms. Five hundred prefix queries complete in 1.12 ms total, 2.24 microseconds per query. The gap between a full $O(N)$ walk and a subtree prefix query is a factor of about 241,000. Query time therefore does not grow as the knowledge base grows from 1,000 to 1,000,000 records; only the prefix depth changes. In cost terms a scan pays the full store while a prefix query pays only the named subtree,

$$T_{\text{scan}} \approx c N, \quad T_{\text{prefix}} \approx c |T(\text{pre})|,$$

so the measured per-query gap is $N/|T(\text{pre})| \approx 241,000$, and growth in N never enters T_{prefix} .

Structural Filtering

Over 50,000 facts, axis filters resolve to id sets without materializing records.

Query	Matches	Time
creator only	2,500	745 μ s
origin and creator (AND)	500	296 μ s
three axes (origin, creator, time)	bounded	83.8 μ s

The figures above are predicate times: they resolve coordinate id sets without loading record bodies. The full read path additionally materializes record bodies from the record maps; on the same 50,000 facts a creator filter

⁸neXus benchmark suite, github.com/ssccsorg/nexus, benches directory

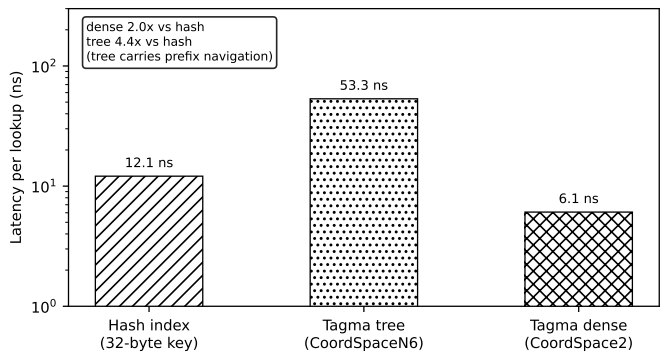


Figure 10: Identity lookup in the 6-axis space (10K lookups): hash-based index versus Tagma layouts. The dense layout is 2.0x faster than the hash index; the tree layout is 4.4x slower on single-record lookups because it carries prefix navigation.

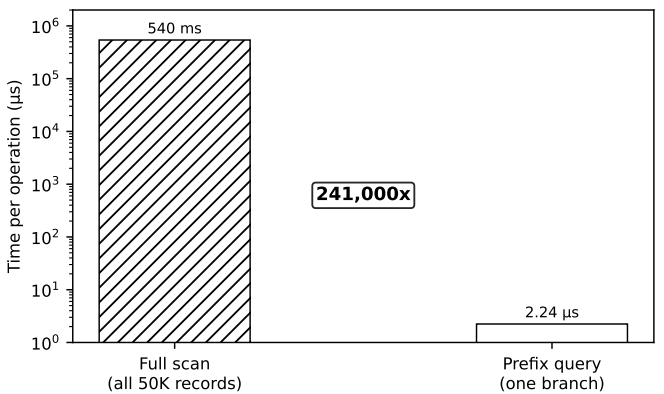


Figure 11: Query cost in a record store over 50K entries: touching every record versus navigating the named branch. Any scan-based paradigm pays the full cost on every query; the coordinate tree pays only the subtree.

including materialization measures about 4.9 ms, which shows that the structural index is the cheap stage and that pruning happens before any record is loaded. The previous full-scan ceiling was 27.8 ms at 10,000 facts, and a scan pays that cost regardless of selectivity. On the same 10,000-fact store the predicate path runs in 127 μs without axis hints and 42 μs with them; the pair measures the planned axis-hint fast path, whose wiring into the production read path is pending. The current predicate path sits roughly 219x below the earlier ceiling. In this design, selectivity is structural: each added dimension shrinks the candidate set before a record is loaded, so the most selective query is the cheapest.

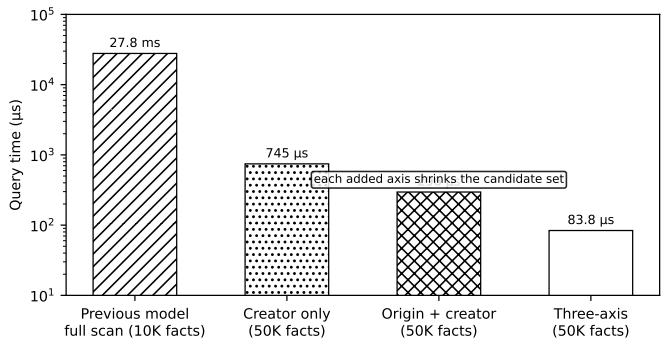


Figure 12: Filter queries: previous full-scan model (10K facts) versus current selective queries (50K facts). The previous model scanned the entire store regardless of selectivity, so the gap is conservative.

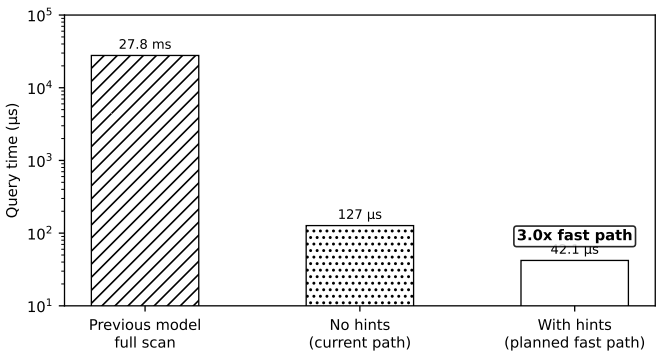


Figure 13: Filter path on 10K facts: previous full-scan model versus current predicate path and the planned prefix fast path (axis hints). Same data size throughout.

Knowledge Base and Write Path

The knowledge-base fixture moves the properties measured on the synthetic store into an application-shaped store: 10,000 documents across 10 projects and 20 authors, with Intents referencing Facts.

Query	Time
project and author (AND)	70.2 μs
project only	260 μs
project, author, time	70.4 μs

Query	Time
reverse lookup of Intents referencing a Fact	189 μ s per 100 calls (about 1.9 μ s per call)

Selectivity is structural here, as in Section : the single-axis project query is the slowest at 260 μ s, and adding axes does not add cost because each axis prunes the candidate set before any record is loaded, so the two-axis and three-axis queries both resolve in about 70 μ s. Reverse lookup of the Intents referencing a Fact resolves through the inverse index in 189 μ s per 100 calls, about 1.9 μ s per call; the earlier full-scan path cost 60.4 ms per call and is superseded by the inverse index.

A batch of 10,000 facts commits in 51.9 ms, or 5.19 microseconds per fact, within a single flush boundary. The same workload before the coordinate restructure took 1,776 ms, a 34x gap.

In rate form the single-flush batch sustains

$$\lambda = \frac{N}{\Delta} = \frac{10,000}{51.9 \text{ ms}} \approx 1.9 \times 10^5 \text{ facts/s},$$

where per-record I/O would multiply the flush cost by N .

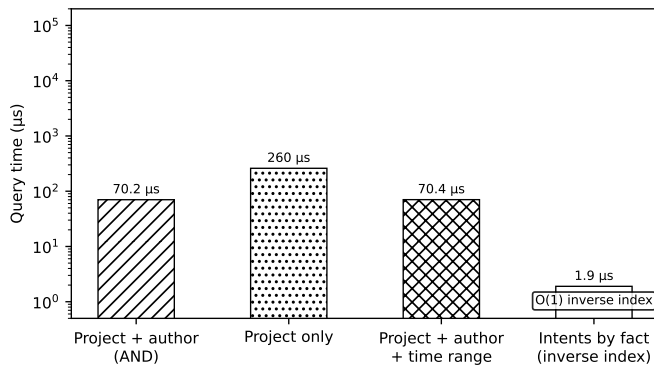


Figure 14: Knowledge-base scenario (10K documents, 10 projects, 20 authors): selective queries are sub-millisecond; reverse lookup resolves through the inverse index.

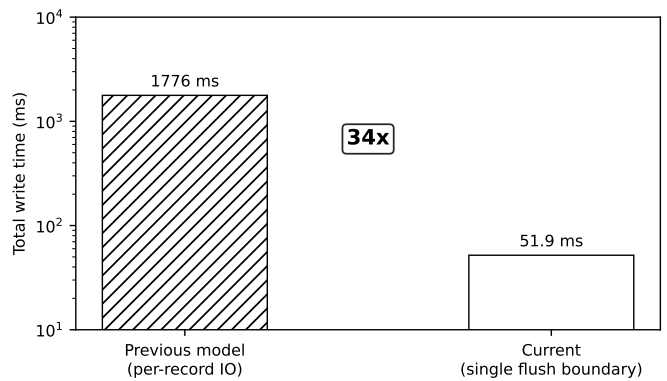


Figure 15: Batch write of 10K facts: pre-Tagma full-scan model versus current single-flush batch (34x).

Restructure Outcomes

The layer-2 restructure moved record bodies to application-owned maps. Conflict detection is sub-microsecond per call (445 ns against 135 to 390 ms before), the average per-fact footprint dropped from about 3.4 MB to about 40 KB, and a 10,000-event stress completes in about 9.0 seconds against 17.7 seconds previously.

Flat Index Cost

The index footprint is flat across record count: approximately 264 MB from 10,000 through 1,000,000 records, because the index is a fixed structure over axis combinations; the scale runs pin facts to fixed day buckets through a stepping clock, so the index structure is byte-identical across scales. The record layer costs about 901 bytes per fact, and live heap grows from 287 MB at 10,000 records to 354 MB at 100,000 and 1,165 MB at 1,000,000. Three-axis filters against a record-map scan are 8 to 38x faster at 100,000 records (scan 1,520 and 952 μ s for wide and narrow queries, tree 173 and 25 μ s) and 8 to 81x faster at 1,000,000 records (scan 25,700 and 18,300 μ s, tree 3,240 and 226 μ s). On a document set with real FIH lifecycles the tree advantage is 4.5x in phase 1 and 5.3x in phase 3. The flatness boundary is about 349 KB per distinct axis value, one dense leaf of 11,172 slots plus one branch.

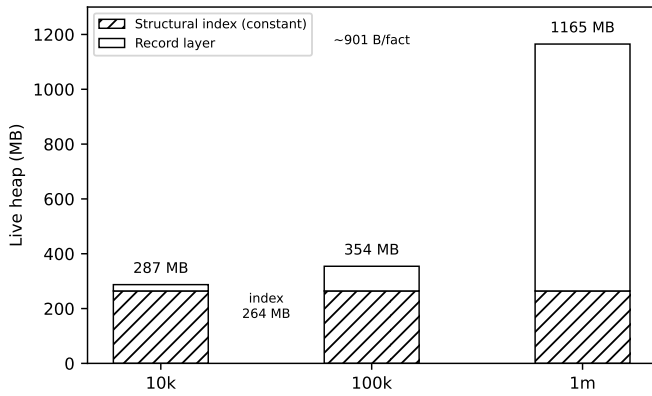


Figure 16: Flat indexing cost: the structural filter index holds a constant footprint as the record count grows. The index portion (hatched) is identical at 10k, 100k, and 1m facts over a fixed axis-combo space; only the record layer grows (~901 bytes per fact).

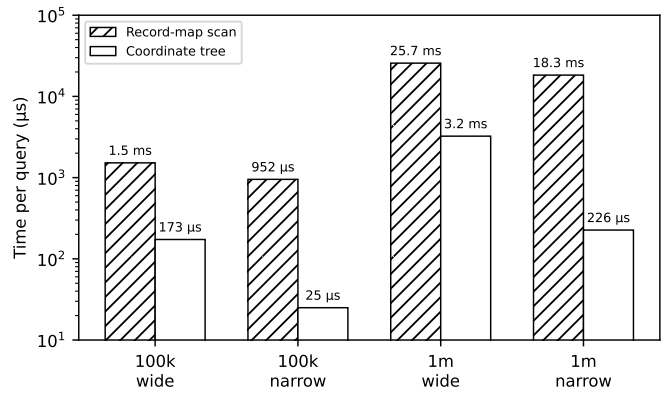


Figure 17: Multi-dimensional search at scale (origin + creator + time range): the coordinate-tree path (iter_prefix) versus the record-map scan. The gap widens as records accumulate: 9x to 38x at 100k, 8x to 81x at 1m facts.

Identity Generation

Coordinate identity generation measures about 0.38 ns per identity against 227 ns for a SHA-256 digest, a factor of about 600, in the coordinate integration measurements of the Tagma application stage. The comparison is scoped to identity generation, not to cryptographic integrity; a non-cryptographic hash such as FNV-1a or xxHash is a closer comparator, and the structural coordinate is cheaper than those as well.

Application Domains and Ecosystem

The model is substrate-independent, and the test of that claim is an instance on a substrate the reference never touched. Instances follow one naming rule, `nex-{implementation}`: each is an independent project with its own identity and, simultaneously, a configuration of the same record model, so nothing in the record layer is added or changed to carry it.

Four domains recur across those instances: research documentation, where hypotheses are recorded as Intents, validated results as Facts, and scope as Hints; agent coordination, where peers interact only through the shared space without a privileged orchestrator; serverless and edge execution on any append-only backend; and knowledge accumulation, where solved problems deposit verified knowledge back into a store that compounds recursively.

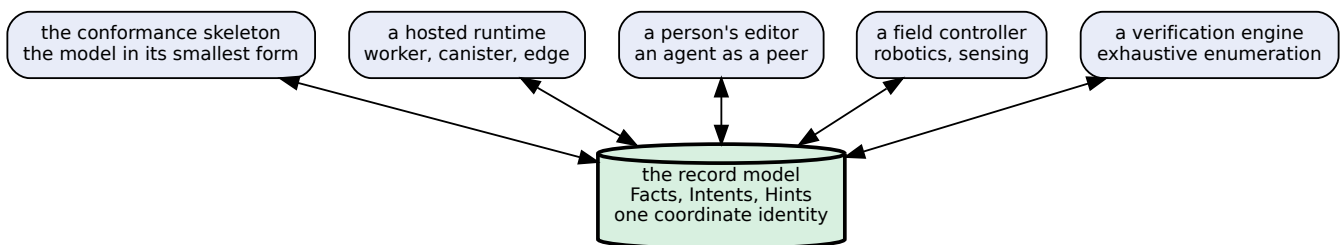


Figure 18: One record model, many substrates. An instance is a configuration of the same model, and the record layer is unchanged from a native daemon to a hosted runtime.

The conformance skeleton is the model in its smallest form and the only instance this document leans on. Every number is a Fact, every operator is an Intent, and every constraint is a Hint, so the calculator reduces computation to $F \times I \times H \rightarrow F'$ and establishes that nothing else is needed. It is deliberately inefficient: its value is conformance rather than throughput, and the properties it demonstrates are the properties a complex design must preserve.

The remaining instances are recorded with their platform, their source, and their status in the repository catalog rather than here, and this document states them at the level of what they show: the same records answer on a hosted runtime, in a canister, inside a person's editor, on a field controller, and in a verification engine. A named prototype dates a claim to the prototype, and the claim this document makes is about the model.

Bindings mirror the core: Rust is native, C and C++ are planned, a Python prototype exists, and Node is planned. Hosted deployments demonstrate that the same core serves serverless workers, canisters, and native daemons without code changes to the record layer.

Boundaries

The following limits are explicit, and several of them are deliberate design decisions documented in the repository.

- Coordinates give ordering and addressing, not preimage resistance. The content hash is retained in the record as evidence; identity claims are scoped to structure, and cryptographic claims are scoped to the hash.
- The conflict guard lives on the `submit_fact` path. A direct writer that bypasses submission can place a record at an occupied coordinate; id stability is documented on the placement function and asserted in debug builds, and conforming applications derive identity from content. A fresh id is an $O(1)$ record-map miss and does not pay the guard.
- Bulk import is a documented restore operation that overwrites. It is the migration path for whole-storage import and intentionally does not run conflict-checked per-record submission.
- The change from six-coordinate to twenty-coordinate identity is breaking, and pre-1.0 data requires re-ingestion through the semantic layer rather than an in-place migration.
- Recording everything is expensive. The skeleton is a reference, not a product; the claim is that the record is the value, and the cost must be measured. The measurements in this document are the beginning of that accounting, not its end.
- The CERN ROOT result (a coordinate-addressed read path reducing a full-file read from 192.8 seconds to 1.06 seconds, 181.7x) validates coordinate addressing on legacy software; the boundary is kept explicit that it does not by itself validate FIH record semantics.
- The learning extension is a hypothesis with a falsifiable program, not a result. Path accumulation may fail on representational grounds against stochastic optimization.
- The on-chain governance economy is staged: write admission, constraint evaluation, and evidence chains are implemented; consensus, ordering, and the adversary model sit above the record layer, in the same place the FIH lifecycle lives.
- The structural advantage requires a bounded prefix. A creator-only query that cannot bind the leading origin axis cannot prune the tree: at one million facts the tree walk (381 ms) is slower than the record-map scan (108 ms). Queries should bind high-cardinality leading axes, typically time or origin.
- The aether transport layer is outside the verified scope of the repository and is treated as conjecture until it has concrete content.
- The twenty-coordinate identity is an encoding of a digest, not a set of semantic axes; axis queries apply to the six-coordinate domain identity only.

Related Work

The FIH blackboard belongs to a family that includes classical blackboard architectures, knowledge graphs, vector stores, and agent memory layers, and it differs from each in a specific way.

Classical blackboards coordinate problem solvers through shared state; neXus restricts the state to three record kinds with a fixed lifecycle, makes Facts immutable, and gives identity a coordinate structure, which turns the blackboard itself into a replayable audit trail. Knowledge graphs store static entity-relationship triplets; a neXus Fact stores the research step that produced a statement, so the graph is a queryable computational trace rather than a snapshot. Vector stores answer similarity queries over embeddings; the structural filter answers axis queries over coordinates at constant marginal cost with no model inference. Agent memory layers in closed ecosystems couple memory to the platform; the FIH primitives decouple memory from any single backend.

A separate axis of comparison is the orchestration harness. Recent agent frameworks coordinate model calls through a fixed program: the harness owns the state machine, lays out the graph or pipeline of steps, and routes messages between roles. neXus inverts this shape. There is no harness-owned state machine and no fixed program; the fabric is a concurrent substrate on which observers act. An observer is any executor in the *kineTics* sense⁹, an agent, a sensor, a hardware transducer, or a human-operated terminal, and its only commitment is a set of Hints, the contract that bounds what it may do. Under that contract it reads and writes coordinate-addressed records on the shared blackboard, and coordination between observers is stigmergic: each acts on the traces the others left, with no central router deciding the next step.

Dimension	Orchestration harness	neXus fabric
Coordination	central harness runs a fixed graph or pipeline	none; observers coordinate through shared immutable records
State	harness-owned session or context object	the record is the state; the blackboard is append-only
Agent contract	tool and model calls defined by the harness	Hints bound what an observer may do; roles are not assigned
Retrieval	vector or external index chosen by the harness	coordinate addressing via Tagma; no inverted index for lookup
Memory	per-run context or external store	FIH records over any backend, decoupled from the platform
Scale	one orchestration run	horizontal fabric of independent nex instances

The difference is not throughput but control. A harness composes a fixed sequence and is therefore deterministic by construction, which is its strength for reproducible single workflows; neXus does not sequence observers and cannot guarantee a step order, which is its strength for open-ended concurrent accumulation. The two are complementary: a harness may run inside an observer, and the trace that observer leaves is recorded as Facts like any other, but the harness does not become the coordination medium of the fabric.

The ledger direction of the project follows from the same record layer: a twenty-coordinate identity is a full-injective encoding of a 256-bit digest, `rebuild_cache` is the seed of offline state reconstruction, and per-channel partition scans are the seed of multi-channel semantics. The framework of stigmergic coordination grounded in distributed ledger state has a published formal counterpart¹⁰. Consensus, ordering, and optional propagation are application-layer programs over the record layer; they are not yet part of the core.

A landscape catalog of roughly sixty external systems, spanning knowledge-graph construction, autonomous research, GraphRAG, and curation, is maintained in the project documentation as the reference survey for the retrieval and agent layers¹¹.

Status and Roadmap

The record layer and the runtime are implemented. The workspace is split into core, FIH, storage, and application crates; the last proprietary storage backend has been removed; the runtime is separated into a blackboard engine (`nex-server`) and a pure supervisor (`nexd`) with a formalized wire contract; and the governance primitives of write admission, constraint evaluation, and evidence chains operate in the core. Gap and contradiction detectors record idempotent Facts. The OS-less profile is verified in CI. The simulator passes phases 0 through 3.

The roadmap, in the order the project publishes it:

⁹kineTics, universal execution and transduction controller; an executor is any agent, signal, hardware, human-machine, or bridge type under one coordinate-based contract. Reference: see project documentation under *kineTics*.
¹⁰Ledger-State Stigmergy: A Formal Framework for Indirect Coordination Grounded in Distributed Ledger State, arXiv preprint cited in the project documentation
¹¹Reference landscape catalog of external systems, maintained in the neXus project documentation

Item	Status
Wire the multidimensional search path into the production read path	planned
Formalize the coordinate system specification	planned
Native query language over the record model	planned
Contract layer extensions	planned
Data-at-rest protection	planned
Research loop with AI hypothesis generation	planned
Attack-scenario verification of storage	planned
Agent-layer stigmergy: pressure fields and trace decay	planned
Editor integration over a standard agent protocol	in progress
Deterministic path learning research program	phases defined, first phase in execution

The learning research program deserves a closing note because it frames the direction. Phase 1 formalizes the skeleton as the conformance reference and tests that a run is fully reconstructible from its FIH record. Phase 2 builds a path learner over the filter index. Phase 3 benchmarks the learning trace on the existing store surfaces. Phase 4 runs the learner on the OS-less storage line. Phase 5 expresses a small SGD-trained function as accumulated paths and compares reproducibility, auditability, accuracy, and cost. The purpose of the program is to find the boundary between path-accumulated knowledge and stochastic optimization, not to declare a winner in advance.

Conclusion

neXus records computation and knowledge with three primitives over any storage backend. Facts are immutable observations, Intents are bounded explorations with a strict lifecycle, and Hints are volatile constraints; identity is a coordinate, so identity is queryable and lookup is arithmetic. The reference implementation demonstrates the properties that matter for accumulated knowledge: a flat index whose memory is bounded by axis cardinality, a write path measured in microseconds per record, structural filters that resolve queries without materializing records, deterministic replay, and an OS-less profile. The model instantiates SSCCS primitives directly, and the ecosystem of `nex-*` applications validates the record semantics on serverless workers, canisters, editors, robots, and verification engines.

The distinction between what is established and what is conjectured is maintained throughout the text. The record semantics, the measurements, and the verification results are established for the reference implementation. The extension of the record model to learning, to cross-reality verification, and to on-chain governance is a research program with defined, falsifiable phases. The value of the design is that both kinds of claims are testable against one immutable record.

Appendices

The appendices carry the detailed specifications that the main text summarizes. Each appendix is self-contained and cites the repository files that define the contract.

FIH Record and Lifecycle Specification

A Fact record is immutable and carries the coordinate identity, the origin, the creator, the content, the content hash, and a provenance reference to the Intent that concluded it. An Intent record is stateful and carries the coordinate identity, the set of Facts from which it departs, a description of the proposed operation, the creator, and a lifecycle state. A Hint record is volatile and carries a constraint over admissible Intents and Facts in its scope.

Intent lifecycle states and transitions:

State	Transition	Meaning
submitted	on write	proposal entered, no agent assigned
claimed	claim(id, agent)	an agent binds to the Intent
active	heartbeat(id, agent)	the agent confirms liveness
concluded	conclude(id, result)	the Intent emits a Fact
evicted	timeout	stale Intent removed by the scheduler

Governing rules: observe as Fact, act as Intent; a concluded Intent emits exactly one Fact; no Fact is overwritten.

Coordinate Identity Specification

Default axis order and semantics:

Axis	Semantics
time_hi	coarse time coordinate
time_lo	fine time coordinate
kind	entity kind: F, I, or H
origin	originating domain or blackboard
creator	producing agent or observer
serial	disambiguating counter

Identity configurations:

Configuration	Coordinates	Space size	Use
domain identity	6	$11,172^6 \approx 2^{81}$	record addressing within a blackboard; axis-queryable
evidence identity	20	$11,172^{20} > 2^{256}$	full-injective encoding of a 256-bit digest; not axis-queryable

Conflict semantics: a coordinate occupied by different content is a conflict and the write is rejected; a coordinate occupied by identical content is an idempotent write; occupancy lookup is $O(1)$ via the existing-content-hash index. The six-to-twenty coordinate change is breaking and requires re-ingestion.

Storage Architecture Specification

The synchronous core exposes three primitives.

Primitive	Operations	Coordination role
Object	read-state, compare-and-swap	only coordination primitive
Key-value	read, write, delete, list	warm record bodies

Primitive	Operations	Coordination role
Blob	put, get, delete, list	cold bodies and archives

The tier model: hot state on object compare-and-swap, warm state on key-value, cold state on blobs with an advancing flush cursor. Composite cold storage layers Parquet or DuckDB under the same interface.

The AsyncStorage bridge runs in four steps: hydrate the backend into memory, run the synchronous core on dirty-tracked buffers, drain only mutated keys, and flush through the async boundary. Core source files are shared between native and hosted builds.

Crate boundary re-exports that keep consumer paths stable:

- `chton::store`: `CoordEntityStore`, `EntityStore`, `MapEntityStore`, `MemoryEntityStore`
- `chton::io`: `CoordMapStoreIo` file backend

Core semantics that remain in the `fi`h crate: the conflict guard in `submit_fact`, the occupancy lookup `existing_fact_content_hash`, the inverse index `fact_to_intents`, the structural filter index maintained by `place_record` and `vacate_record`, deterministic replay in `rebuild_cache`, and per-channel partition scans in `scan_partition`.

Wire Protocol Reference

Transport: Unix domain sockets, line-delimited JSON-RPC 2.0, one object per line, no version field in the message.

Endpoint	Default path	Environment
nex-server (blackboard engine)	/tmp/nex-server.sock	NEX_SOCKET_PATH
nexd (supervisor)	/tmp/nexd.sock	NEXD_SOCKET_PATH

Error codes:

Code	Meaning
-32602	invalid params
-32601	method not found
-32000	internal error
-32001	not found
-32002	conflict, for example duplicate id with different content hash
-32003	forbidden, for example an Intent without a Fact reference

Blackboard methods on nex-server (id values are canonical coordinates):

Method	Parameters	Result
<code>write_fact</code>	origin, content, creator	fact id
<code>read_state</code>	none	facts, intents, hints
<code>read_state_struct</code>	none	state structure with empty content
<code>read_fact</code>	id	fact
<code>read_intent</code>	id	intent
<code>read_hint</code>	id	hint
<code>write_intent</code>	from_facts, description, creator	intent id
<code>claim_intent</code>	id, agent	ok
<code>heartbeat_intent</code>	id, agent	ok

Method	Parameters	Result
release_intent	id, agent	ok
conclude_intent	id, result	resulting fact
write_hint	id, content, creator	ok

Supervisor methods on next forward FIH methods verbatim and handle locally:

Method	Parameters	Result
spawn_agent	command, args	pid, command
list_agents	none	agents with pid and command
kill_agent	pid	ok

The contract is stable: a change to a method name, parameter, or error code is breaking and must be updated in the client SDK together with the protocol document.

Verification Scenario Catalog

The simulator exercises seven attack scenarios against the FIH lifecycle.

Scenario	Attack surface	Expected defense
Direct free-riding	read without contribution	read requires write
Evasive free-riding	minimal or duplicated contribution	content-derived identity and conflict guard
Falsified contribution	fabricated result	provenance chain and signed execution
Poison injection	misleading Facts in shared store	admission and constraint evaluation
Intent spam	unclaimed Intent flood	heartbeat lifecycle and stale eviction
Replay	re-submitted action	idempotent identity semantics
Sybil	multiple fake identities	origin and creator binding

Game-theoretic gates run in CI: basic cooperation over 200 turns requires tit-for-tat to outscore always-defect; evolutionary stability over 500 generations requires tit-for-tat to survive; generous forgiveness requires a cooperation rate above 0.6; the FIH lifecycle gate requires honest completion above 0.7 and false conclusions below 0.1.

Benchmark Notes

All figures in Section come from the reference benchmark suite, executed on Apple silicon (M1) in release mode with the median of ten runs reported. The suite and its sources are in the reference repository under benches. Two structural properties explain the flatness of the results: each leaf of the filter index covers 11,172 slots, and dense packing gives a flat lookup independent of record count. Figures compare the restructured index against recorded internal baselines, the pre-restructure ceiling or a hash index, not against an external benchmark. Methodology, fixture shapes, and measurement conventions are stated in Section .

Glossary

Term	Meaning
Blackboard	any backend holding Facts, Intents, and Hints
Fact	immutable, validated observation
Intent	proposed exploration with submit-claim-heartbeat-conclude lifecycle

Term	Meaning
Hint	volatile, read-only constraint
CoordId	ordered tuple of coordinates used as identity
Structural filter index	axis-combination index mapping to id sets
Conflict guard	content-hash check rejecting id collisions at commit
rebuild_cache	deterministic replay of record maps and index
scan_partition	per-channel record flow scan
nexd	supervisor daemon
nex-server	blackboard engine process
OODA scheduler	observe-orient-decide-act tick for heartbeats and eviction
Stigmergy	indirect coordination through left traces
eKG	episodic knowledge graph preserving temporal and provenance order
ULHM	universal latent homeomorphic manifold verifier
AsyncStorage bridge	hydrate-run-drain pattern for hosted runtimes