CHANDRA

CONTEXT UNIT PROTOCOL

Perpetual organizational memory for every human decision and every agent action.

MIT LICENSE · v17.0 · GENERAL REASONING, INC.

v17.0 · April 2026 · MIT License · General Reasoning, Inc. · GR-008

v17.0 introduces the chronicle framing throughout: the chain is the organizational chronicle. A Chandra Core vs. DXMachine section clarifies the open-source/commercial boundary. All prior content is preserved. Limitations section from v15.0 retained in full.

OVERVIEW

What Chandra Is

Every other system produces audit logs as a side effect of operations. Chandra inverts this. The audit record is not produced after the fact — it is the mechanism by which the next operation is authorized. The chain must grow before flow can continue. The record is the gate token, not the receipt.

> **Auditability is the authorization mechanism for flow. Not a recording layer applied after. Not a compliance add-on. The audit record is a gate token, not a receipt.**

This distinction matters most in regulated human-agent environments. An agent in any other system can move faster than the audit trail. In Chandra the chain leads. The agent cannot append without producing an attestable record of its identity, its authorization, and its examiner result. There is no gap between action and record because the record IS the action.

Chandra treats agents and humans as first-class citizens. Humans need observation points, intervention capability, version designation, and a chain navigable at human speed. Agents need machine-speed append, optlock retry, full attribution, examiner gates, and tier-governed capability envelopes. Neither is subordinate to the other. The protocol serves both.

The organizational implication: an agent given access to a Chandra instance can reconstruct what the organization knew, what it decided, what it did, who authorized it, and when — at any point in history. The organizational chronicle is not a document. It is the chain.

What Chandra Is Not

Chandra is not Git with attribution. Git was designed for humans writing code. Agents are an afterthought in every Git-adjacent system, bolted on via CI/CD pipelines where the agent is outside the commit graph. In Chandra agent identity is a first-class principal.

Chandra is not a log aggregator. Log aggregators record what happened. Chandra gates what can happen next. The difference is architectural, not a matter of degree.

Chandra is not a message broker. It does not own subscriptions, fan-out delivery, or watch semantics. Chandra owns the record. The notification layer owns the routing.

Chandra is not a physical inventory tool. It tracks digital artifacts and digital meta-artifacts — the maps, decisions, and lineage records — that describe what physical assets exist and how to recover them. The reconstruction roadmap, not the inventory.

CHANDRA CORE AND DXMACHINE

Open Protocol and Commercial Product

Scope of this Specification

The Chandra protocol is open source under the MIT license. The reference implementation covers the core protocol: context units, lineage chains, attribution records, snapshots, ticklers, spokes, echo dispatch, diff, and three-way merge for text-class artifacts.

Several capabilities described in this whitepaper are implemented exclusively in **DXMachine**, the commercial enterprise product built on Chandra by General Reasoning, Inc. These are **not** part of the open protocol specification:

- Domain / Hub / Spoke organizational taxonomy and Commit Records
- The Chandra Marshaller and multi-instance federation
- Marshal API endpoints and Instance Manifest
- Attachment manifest with object storage routing
- SSAE 18 audit header mapping and SOC 2 evidence pipeline
- Agent examiner gates, tier classification, and capability envelopes
- Governed change pipeline and changeset model

The open protocol is the foundation. DXMachine is what that foundation looks like running the compliance machinery of a regulated organization.

DXMachine inquiries: **inquiries@genreason.com** · genreason.com

PROTOCOL PRIMITIVES

One Operation, Five Primitives

Chandra has one operation: append. Every CU is an append to the lineage chain of its subject. The chain grows forward and never backward. Nothing is modified, nothing is deleted, nothing is rebased. Branch reconciliation is itself an append — it names two predecessors instead of one.

> **Chandra has one operation. Append. Everything else is a variation.**

1. Context Unit (CU)

The atomic unit of Chandra. Immutable once appended. Fixed header and variable payload. One artifact per CU. Every append produces exactly one CU atomically with the operation.

2. Lineage Chain

Ordered, append-only sequence of CUs for a given subject. Each CU references its predecessor by cu-predecessor-id. cu-prev-hash seals the chain cryptographically. A broken chain halts new appends.

3. Attribution Record

Embedded in every CU header. Who initiated the append — human, agent, or system — what model was used, what session. Attribution is required. Anonymous CUs are rejected at the protocol level.

4. Snapshot

A complete, self-contained, AI-readable representation of a CU and its full lineage chain. An agent or human can reconstruct the full context of any subject from its Snapshot alone.

5. Tickler

A forward-scheduled CU with a trigger time and a target. Fires once at cu-trigger-at. Targets: human:{id}, role:{name}, agent:{id}, board:{id}, chandra:{instance-url}. Cancellation is a superseding append.

SCHEMA

Context Unit Schema

Every Context Unit carries two sections: a fixed header and a variable payload. The header is indexed and structured for fast agent filtering. The payload is artifact-type specific.

Header — fixed, indexed, always present

Slot	Type	Notes
`cu-id`	`string`	UUIDv7 — optlock token; lexicographic = creation order
`cu-timestamp`	`integer`	Unix timestamp; primary sort key
`cu-sequence`	`integer`	Monotonic per-instance sequence number. Marshaller cursor field. (GR-PROV-020)
`cu-actor`	`string`	"human:{id}" | "system" | "agent:{id}"
`cu-session-id`	`string`	Authenticated session; empty string pre-auth
`cu-ip-address`	`string`	Source address; empty for system/agent events
`cu-correlation-id`	`string`	Groups CUs from one logical operation.
`cu-component`	`string`	Originating component identifier
`cu-subject-id`	`string`	FK to affected record or artifact
`cu-subject-type`	`string`	Vocabulary defined by implementing system
`cu-event-type`	`string`	Vocabulary defined by implementing system
`cu-outcome`	`keyword`	:success | :failure | :partial
`cu-severity`	`keyword`	:info | :warning | :error | :critical
`cu-predecessor-id`	`string/nil`	Prior CU in lineage chain; nil = root CU
`cu-prev-hash`	`string`	SHA-256 of predecessor canonical JSON; zeros for root
`cu-scope-id`	`string/nil`	Optional parent subject_id — cross-spoke scope reference. (GR-PROV-017)

Payload — artifact-type specific

Slot	Type	Notes
`cu-artifact-type`	`string`	"mutation"|"text"|"code"|"voice-transcript"|"structured"|"tickler"|"commit-record"

`cu-before`	`string/nil`	Full snapshot of subject before mutation; nil on :created
`cu-after`	`string/nil`	Full snapshot of subject after mutation; nil on :deleted
`cu-change-reason`	`string`	Transition notes, abandon reasons, change rationale
`cu-context`	`string`	JSON — event-type-specific structured data
`cu-artifact`	`string/nil`	Artifact content for non-mutation CUs
`cu-prompt-lineage`	`string/nil`	JSON array of prompts/responses for AI-produced artifacts
`cu-model-id`	`string/nil`	Model identifier for AI-produced artifacts
`cu-human-author`	`string/nil`	Identity of human who directed or approved artifact
`cu-notes`	`string`	Human-authored description of what changed and why
`cu-tags`	`string`	JSON array of discovery and filter tags
`cu-trigger-at`	`integer/nil`	Unix timestamp for tickler CUs; nil for non-tickler
`cu-trigger-target`	`string/nil`	Tickler target: human:{id}, role:{n}, agent:{id}, chandra:{url}

Full snapshot policy: cu-before and cu-after carry complete JSON snapshots on every write — not a delta, not a hash. The hash chain proves the snapshot was not altered after the fact. Both are required. Neither alone is sufficient.

TAXONOMY

Instance, Domain, Hub, Spoke

Note: Domain, Hub, Commit Record, and Marshaller layers are implemented in DXMachine, the commercial product. They are not part of Chandra Core open-source specification.

Chandra organizes context units into four tiers. The hierarchy is fixed — no tier may be skipped, no tier may be added above the Instance. The Commit Record is a parallel structural primitive that runs vertically through the hub and domain tiers.

> **Instance — Domain — Hub — Spoke. Nothing above the Instance. Nothing between spoke and hub. The Commit Record is a parallel primitive, not a tier.**

Instance

A Chandra Instance is a sovereign deployment — one AllegroCache store, one AllegroGraph triple store, one HTTP surface, one cryptographic seal boundary. An instance is owned by one organization or tenancy.

Domain

A Domain is a coherent operational context within an Instance — a reconstructible organizational unit that could be handed to a recovery team as a complete package. Hub domain membership is declared at creation and immutable.

Hub

A Hub is a governed artifact cluster — one logical subject with multiple artifact dimensions represented as spokes. Hub membership in a domain is declared at creation and immutable.

Spoke

A Spoke is one artifact dimension within a hub — one subject_id, one append chain, one lineage history. Spokes within a hub are structurally independent.

Spoke Granularity — One Spoke Per Governed Record

For systems using Chandra for record-level audit coverage, the correct granularity is one spoke per governed record. The optlock token is the cu-id of the latest CU on a subject's chain. Two records sharing a chain would share a lock — write contention on one would block writes to the other.

> **One spoke per governed record. The optlock token is per-subject. Aggregating records into shared chains destroys optlock semantics.**

Commit Record — parallel structural primitive

The Commit Record is not a spoke, not a hub, and not a tier. It is a parallel structural primitive at the hub and domain levels. Auto-created, non-deletable, not user-configurable. Its cu-artifact-type is "commit-record". It uses the same CU structure and hash sealing.

MULTI-INSTANCE ARCHITECTURE

The Chandra Marshaller

Note: The Marshaller is a DXMachine commercial feature, not part of Chandra Core.

A single Chandra instance is the right architecture for a single organization with a uniform regulatory regime. Enterprise deployments cannot place all governed data in one store. Data residency law and regulatory isolation requirements push toward multiple sovereign instances.

The Chandra Marshaller is the coordination layer for multi-instance deployments. It is not a Chandra instance. It holds no append chains. It is a stateless routing and aggregation layer.

> **The Marshaller is not a monolith above the instances. It is a stateless coordinator beneath them — aware of instances, owning nothing.**

Marshaller Endpoints

Endpoint	Purpose
`GET /marshal/instance-manifest`	Instance identity, purpose, domains, regulatory context, marshal API version.
`GET /marshal/health`	Instance liveness. Returns: instance_id, last-commit-timestamp, CU count, status.
`GET /marshal/scope-index`	All domains, hub counts, last-activity per domain. Routing table builder.
`POST /marshal/query`	Cross-instance header-level query by scope, domain, subject_type, tags, time range.
`GET /marshal/commit-feed`	Ordered stream of commit records since a given cu-sequence cursor.

IMPLEMENTATION

The Append Operation

The discipline that makes Chandra useful: every operation on every subject must produce a CU atomically with the operation. An operation without a CU did not happen in the audit sense.

Step	Action
1	Check predecessor: does cu-predecessor-id match the latest CU for this subject? If not, emit collision CU and return conflict response.
2	Apply the operation to the subject record in the data store.
3	Build the CU: populate header, compute cu-prev-hash from predecessor canonical JSON. Assign cu-sequence from instance monotonic counter.
4	Commit operation and CU atomically — both succeed or neither does.
5	If hub batch: write hub commit record CU and domain commit record CU in same atomic operation.
6	Return new cu-id to caller as updated optlock token.

Subject Record Optlock Integration

The cu-id optlock token is meaningful only if stored on the subject record in the authoritative data store and managed with discipline across the full client-server cycle.

> **The token is not a timestamp. It is not a sequence number. It is the cu-id — the exact identity of the last committed audit record for this subject.**

Phase	Action
Read	Server loads subject record. Returns current state plus event_token to client.
Write (client)	Client sends write request with current event_token in the request body.
Write (server check)	Server compares incoming event_token against stored token. Match: proceed. Mismatch: reject 409.
Write (server commit)	Server applies mutation, calls emit-cu, receives new cu-id. Sets event_token = new cu-id atomically.
Response	Server returns {ok, event_token: new-token}. Client replaces its stored token.
409 Collision	Server returns {error: conflict}. Client re-fetches, re-applies, retries immediately.

Self-Hosting

The first CU in any Chandra instance is a seed event recording the initialization of the system itself. A self-hosting Chandra instance stores the Chandra protocol specification as CU-0001. This is a structural requirement. A

protocol that cannot host its own specification has not been validated.

ARCHITECTURE

Hub, Spoke, and the Changeset Model

> **A hub is a document package. A spoke is one artifact with its full history. The hub commit record is the commit log. The batch-set is the commit operation.**

The translation for users coming from Git: hub equals repository, spoke equals tracked file, hub commit record CU equals commit, batch-set equals git commit.

Scope relationships within a hub

Field	Type	Cross-spoke function
`cu-scope-id`	`string/nil`	Named parent-child relationship between spokes. (GR-PROV-017)
`cu-tags`	`string`	Categorical grouping across any spokes. Indexed for O(1) lookup.
`cu-subject-type`	`string`	Type-based traversal across all spokes sharing a type.
`cu-correlation-id`	`string`	Operational grouping. CUs from one logical operation across multiple subjects.

Attachment Manifest Schema

Note: The formal attachment manifest with object storage routing is a DXMachine feature. The Chandra Core reference implementation stores binary artifacts as base64-encoded JSON envelopes directly in the CU artifact slot as a pragmatic simplification for the open-source reference.

In a production DXMachine deployment, binary attachments live in object storage. The CU artifact carries an attachment manifest: a JSON array of attachment records with sha256, storage_key, filename, mime type, upload timestamp, and status. The binary is never stored in the CU artifact slot.

Field	Type	Description
`attachment_id`	`string`	UUIDv7. Stable identifier across all versions.
`filename`	`string`	Original filename. Preserved for display.
`storage_key`	`string`	Opaque retrieval key. Format determined by attachment store.
`sha256`	`string`	SHA-256 hash of binary content. Computed on upload. Immutable.
`size_bytes`	`integer`	File size in bytes.
`content_type`	`string`	MIME type. e.g. "application/pdf".
`uploaded_by`	`string`	Identity of human or agent who attached the file.
`uploaded_at`	`integer`	Unix timestamp of upload.

`status`	`string`	"active" | "removed". Removal is a new CU. File never deleted in compliance context.

AGENTS AND HUMANS

Agents, Humans, and the Keeping-Up Problem

Chandra is built to accept agent commits at machine speed. A chain operated by agents can accumulate hundreds of CUs in the time it takes a human to read ten.

> **The chain does not slow down for humans. It is humans who must find ways to keep up.**

Why branches make no sense for agents

Repository branches exist because humans work slowly and in parallel. For regulated audit chains, branching is architecturally incoherent. A regulated audit trail requires a single authoritative sequence. The single-sequence chain is not a constraint. It is the guarantee.

Why versions are a human artifact

A version number is a cognitive compression device. In regulated industries the regulator requires that a human be able to say: I understood the system state at this point and I approved it. The version is a compliance primitive, not a technical artifact of the storage model.

Companion records

Chandra provides two companion record types as navigation aids for humans. Annotations attach a human note to a specific CU after the fact. Version tags group a set of CUs under a human-meaningful label. These carry no chain integrity guarantees.

> **Companion records are human navigation aids. Loss or corruption of companion records does not affect chain integrity or compliance evidence.**

Markdown as Organizational Memory Format

Chandra adopts Markdown as the native artifact format for text-class organizational documents. An agent reading Markdown needs nothing. It is plain UTF-8 text — readable without tooling, diffable with Myers, mergeable with three-way merge, storable without encoding.

> **The source of truth is the Markdown. Rendered formats are delivery artifacts.**

STORAGE

Full Snapshot Policy — Sizing and Infrastructure

Component	Size
CU header (fixed fields, JSON)	~400 bytes
Payload metadata (notes, tags, context)	~600 bytes
cu-before / cu-after (small records)	2-5 KB each
cu-before / cu-after (work items with EAV fields)	10-25 KB each
cu-before / cu-after (governed changes, board snapshots)	40-80 KB each
Conservative average CU (status updates)	5 KB
Moderate average CU (typical work item)	20 KB
Large average CU (governed change with full snapshot)	80 KB

Org size	Users	Mutations/day	CUs/year	1-year raw	5-year raw	RAID
Small	50-150	~200	73K	1.4 GB	7 GB	RAID 10
Mid-market	150-1,000	~2,000	730K	14 GB	70 GB	RAID 60
Enterprise	1K-10K	~20,000	7.3M	140 GB	700 GB	RAID 60
Large ent.	10K+	~100,000	36.5M	700 GB	3.5 TB	RAID 60

A single material compliance finding costs more in remediation than the entire 5-year hardware footprint of a RAID 60 Chandra store at enterprise scale.

ARCHITECTURE PRINCIPLES

One Artifact, Scale, and the Org Asset Thesis

One artifact per CU

One CU carries one artifact. This is permanent and non-negotiable. The moment a CU becomes a container the CU is no longer atomic. Multiple attachments are multiple spokes. A contract with three exhibits is four spokes — one for the contract, one per exhibit — each scoped via scope_id.

Two agent operation models

Model A: agent acts on an existing subject — appends a CU to the existing spoke. No new spoke. Model B: agent produces something that is a new subject — creates a new spoke. The test: did the agent do something to an existing subject (A), or produce something that is a new subject (B)?

> **Agent velocity deepens chains. It does not multiply spokes. Spoke count scales with distinct subjects governed, not with agent speed.**

The organizational chronicle

Under these disciplines, Chandra serves as the complete, reconstructable asset registry of an organization: every document in every version, every work item in every state transition, every agent action attributed and auditable, every compliance event as SSAE 18 evidence by construction.

> **The organizational chronicle is not a document. It is the chain. The chain is the chronicle — every decision, every agent action, in order, forever.**

LIMITATIONS AND DESIGN BOUNDARIES

Known Weaknesses and Out-of-Scope Boundaries

A complete protocol specification states its limitations honestly. The four weaknesses below are real. Each is acknowledged with its scope, its rationale, and an explicit statement of what is and is not within the protocol boundary.

A. Snapshot storage overhead

Full snapshots — cu-before and cu-after carrying complete subject records on every mutation — increase storage cost relative to delta-only approaches. This is a deliberate tradeoff. Delta reconstruction is an audit liability: an auditor must replay a chain to determine state at any point in history. Full snapshots make evidence readable at any CU without replay.

> **Delta reconstruction is an audit liability. Full snapshots are the correct tradeoff for compliance-grade evidence.**

Out of scope for the protocol: Storage tiering, compression, snapshot pruning, and cold storage migration are implementation concerns. The protocol requires full snapshots.

B. Operational complexity

The reference implementation uses three stores: AllegroCache (authoritative write), MariaDB (relational query index), and AllegroGraph (graph traversal). The reference stack is two processes — AllegroCache, AllegroGraph, and AllegroServe run in a single Allegro CL image. MariaDB is the one external dependency.

Out of scope for the protocol: The three-store architecture is a reference implementation choice, not a protocol requirement. Implementations may use a single Postgres or MariaDB instance and satisfy all conformance requirements.

C. Human navigation

Agents may generate hundreds of CUs in the time a human reads ten. The protocol's native answer is annotations and version tags as companion records. A complete answer requires an organizational hierarchy whose commit records provide pre-built summarization at each tier.

> **The human navigation problem is solved by commit record hierarchies, not by summarization agents applied after the fact.**

Out of scope for the protocol: The domain-hub-spoke organizational hierarchy and its commit record propagation are implemented in DXMachine. They are not part of the open protocol specification.

D. Insider threat

The protocol proves attribution, not truthfulness. A valid CU with false content — correct cu-actor, correct hash chain, fabricated cu-notes — is auditable but not self-refuting. The chain records who said it. It does not verify

whether what was said was accurate.

> *Out of scope for the protocol: Content truthfulness verification and behavioral anomaly detection are outside the protocol boundary. External anchoring — periodic publication of the chain tip hash to a notary or ledger — is the stated upgrade path to tamper-resistance. Chandra does not overclaim.*

The insider threat limitation does not reduce Chandra's compliance value for SOX, SSAE 18, CMMC, or HIPAA. Those frameworks require attributable, immutable, non-repudiable records — not content truthfulness verification. Chandra satisfies those requirements by construction.

CONFORMANCE

Conformance Requirements

Requirements marked (DXMachine) apply to commercial implementations and are not required for open-source Chandra Core conformance.

Requirement	Statement
Immutability	Published CUs must not be modified. Any system allowing post-publication mutation is not conformant.
Attribution	cu-actor is required on every CU. Implementations must reject writes where cu-actor is empty or absent.
Hash sealing	cu-prev-hash must be SHA-256 of the predecessor CU's canonical JSON. Root CU carries 64 zero characters.
Primary atomicity	Mutation and spoke CU emit must be strictly atomic in the authoritative store.
Optlock token discipline	The cu-id of each committed CU must be stored on the subject record as the optlock token. Mismatch must result in a 409.
Full snapshot	cu-before and cu-after must carry complete subject snapshots, not deltas or hashes.
Sequence field	Every CU must carry cu-sequence — a monotonically incrementing integer assigned at write time.
Instance manifest (DXMachine)	A conformant DXMachine instance must expose GET /marshal/instance-manifest.
Marshal API (DXMachine)	A conformant DXMachine instance must expose all five marshal API endpoints.
Rollup write discipline (DXMachine)	Hub and domain commit records must be written in the same batch as spoke CUs.

Any implementation satisfying the core conformance requirements, the five primitives, the append operation discipline, and the spoke structure is a conformant Chandra Core implementation. Python and other language ports are invited. The protocol is the contribution.

ENTERPRISE IMPLEMENTATION

Chandra and DXMachine

General Reasoning is building DXMachine — a compliance-grade value stream management platform for regulated industries — on top of Chandra. DXMachine is closed source. The DXMachine instance is the reference multi-domain Chandra deployment: vsm domain for all module hubs, platform domain for infrastructure, agents domain for the skill catalog, core domain for Chandra's own self-description.

The DXMachine implementation extends Chandra core with what regulated enterprises require: an auditable event header mapped to SSAE 18 and SOC 2 Trust Services Criteria, a governed change pipeline where every schema change is an attestable compliance event, and the emit-cu write discipline that makes the audit record a gate token rather than a receipt.

> **Every event DXMachine produces is SSAE 18 audit evidence by construction — not bolted on after the fact.**

The open protocol is the foundation. The enterprise product is what that foundation looks like when it runs the compliance machinery of a regulated organization — financial services, healthcare, defense, state and local government. The open protocol shows you what we mean by auditable by construction. DXMachine is the proof at production scale.

The architecture described here is the infrastructure for building the organizational chronicle systematically. Every append grows it. Nothing removes from it. The chronicle compounds — twelve months of clean audit evidence is difficult to replicate quickly, and the value of that chronicle increases with every completed audit period.

DXMachine inquiries: inquiries@genreason.com · genreason.com

OPERATIONAL GUARANTEES AND FAILURE HANDLING

Engineering Appendix

This section answers the five questions enterprise architects will ask after reading the protocol specification. Where the honest answer is a bounded guarantee rather than an absolute one, that boundary is stated explicitly.

1. Chain integrity violation procedure

Phase	Action	Who
Immediate	Instance enters read-only mode on the affected chain.	Automatic
Forensic	Identify first corrupted CU. Write integrity-event CU to dedicated integrity-events spoke.	Automatic
Escalation	Integrity event triggers immediate human notification. Chain segment flagged permanently.	Automatic → Human
Recovery	Human principal may issue branch-from-last-valid-cu operation.	Human-authorized
Resume	Read-only lifts after recovery CU commits and verification passes.	Automatic post-auth

2. Append transaction semantics

Store	Role	Consistency	Failure behavior
AllegroCache	Authoritative write store.	Strongly consistent. Single commit.	All-or-nothing. Write fails cleanly.
MariaDB	Relational query index.	Eventually consistent. Async write-through.	Reconstructible from AllegroCache.
AllegroGraph	Graph query store.	Eventually consistent. Read-only from AC.	Reconstructible from AllegroCache.

3. Threat model

Threat	Guarantee	Limitation	Upgrade path
Accidental corruption	Detected by chain verification.	Detection is scheduled, not real-time.	Increase verification frequency.
Malicious admin	Admin cannot conceal identity. All operations produce CUs.	A privileged admin can issue valid CUs that misrepresent events.	Separation of duties. Cross-instance audit.
Direct DB access	Any modification breaks the hash chain.	Attacker with DB access could recompute hashes forward.	External anchoring to notary or ledger.

Insider valid CUs	Every CU carries full attribution. Nothing deletable.	Valid CU with false notes is auditable but not self-refuting.	Cross-reference with commit records.

4. Scale under agent load

Property	Behavior	Implication
Chain partitioning	Every spoke is an independent chain. Concurrent appends to different subjects do not contend.	Write throughput scales horizontally with concurrent subjects.
Optlock contention	Two concurrent appends to same subject: one collides. Retry window is milliseconds.	High contention on a single subject is the bottleneck case.
Verification scheduling	Chain verification is background, not inline with writes.	Verification overhead does not appear in write latency.
Query separation	Three-store architecture separates writes from range and graph queries.	Agent write throughput not degraded by complex analytical queries.

5. Marshaller failure modes

Applies to DXMachine multi-instance deployments.

Scenario	Marshaller behavior	Data integrity impact
Instance unreachable	Query dispatched with timeout. Non-responding instance returns partial result with staleness flag.	No data integrity impact.
Marshaller process failure	Stateless. On restart: re-read manifests, rebuild scope index, resume from last cu-sequence cursor.	No impact on any instance. Fully recoverable.
Instance version mismatch	marshal_api_version checked on contact. Incompatible instances excluded.	Results from compatible instances are complete.
Rogue instance insertion	Registration requires shared credential provisioned at deployment time.	Deployment-time security control.

6. Scaling properties and write amplification

The spoke-per-record design distributes load across independent chains. A table with one million records produces one million independent chains, each typically 10-100 CUs deep. Concurrent writes to different records do not contend.

> **One million chains each 50 CUs deep scales better than one chain with 50 million events. Chains do not interact.**

Mutation rate	Effective CU write rate	Notes
1,000 mutations/sec	3,000-4,000 CU writes/sec	Light workload. Within AllegroCache parameters.
5,000 mutations/sec	15,000-20,000 CU writes/sec	Mid-range. Monitor hot-record contention.
10,000 mutations/sec	30,000-40,000 CU writes/sec	Heavy workload. Multi-instance recommended.

Required indexing

Index	Purpose
`(subject_id, cu_sequence)`	Latest CU lookup. SELECT ... ORDER BY sequence DESC LIMIT 1.
`(subject_id, cu_prev_hash)`	Chain verification traversal. Sequential scan in sequence order.
`(hub_id, sequence)`	Hub commit record queries. Ordered hub history.
`cu_tags`	Cross-spoke tag filter. O(1) tag lookup across all chains.

Implementations that omit the (subject_id, cu_sequence) index will perform full table scans on every chain-tail lookup. This is the most critical index in the schema.

Alternative authoritative stores

Store	Deployment context
MariaDB (InnoDB)	Primary alternative for DXMachine. Already in the three-store architecture as query index. MyRocks worth evaluating for high-volume append workloads.
PostgreSQL	General-purpose reference implementation. ACID transactions, row-level locking, strong SQL standards compliance.
kdb+/q	High-frequency financial services. Native q implementation provides Chandra audit semantics on existing kdb+ infrastructure.
CockroachDB	Distributed Postgres-compatible. Best fit for multi-region deployments where single-node capacity is insufficient.

The protocol is the contribution. The store is an implementation choice.