STARGAZER

PROTOCOL INTELLIGENCE BY FERNO

TECHNICAL DOCUMENTATION / 01

Technical architecture and roadmap

Local-first Stellar protocol intelligence

Archive replay, optimized indexing, live convergence, backfillable protocol modules, operations, and delivery.

ARCHITECTURE EDITION / VERSION 1.0 / AUGUST 2026

stargazer.ferno.ag/docs

Archive-index POC; live alert delivery is not yet production-ready.

DOCUMENT MAP / STATUS

Contents

This document separates the implemented archive-backed system from the target live architecture. Status labels are evidence claims, not schedule promises.

Implemented	In progress	Planned
Executable code, schema, script, or service exists	A working subset exists; integration or coverage is incomplete	Documented target; not wired into the running ledger pipeline

How to read the architecture

Solid processing paths describe code that can be run from this repository. Planned paths are identified in text and in status tags. The most important boundary is the normalized Stellar ledger input: archive and live acquisition should converge there, then share the same deterministic fact extraction and commit rules.

Scope and terminology

- Local-first means a retained Galexie archive mirror and local derived databases on the NAS.
- The archive mirror is not a Stellar validator and Stargazer does not publish a network history archive.
- Historical findings are stored for analysis with notification delivery disabled by default.
- Unknown contracts remain visible by their on-chain address until attribution has a verifiable source.

01 / IN PROGRESS

01 / System overview

Stargazer turns retained Stellar ledger evidence into compact dependency, version, and activity indexes. The same logical core is intended to process newly closed ledgers once continuous acquisition is wired.

IN PROGRESS

Chapter status
The archive replay, evidence database, serving projection, read API, and narrow controller watcher exist. Continuous ledger ingestion and production alert delivery remain target architecture.

01.1 / Purpose and current boundary

Modern blockchain applications depend on contracts, controllers, SDKs, repositories, and protocol infrastructure that change independently. Stargazer builds a verifiable history of those dependencies and prepares concise findings for developers. The current proof of concept is historical and archive-backed: it proves the evidence and query model before the live delivery layer is enabled.

Evidence Immutable compressed LedgerCloseMeta files retained locally with ledger provenance.	**Indexing** Bounded Rust decoding, exact relationships, executable history, and ordered commits.
Serving A separate wallet projection and deadline-limited read-only API.	**Intelligence** Deterministic findings first; explanation and delivery remain separate layers.

Figure 1. *The system separates durable evidence, derived state, public reads, and notification policy.*

01.2 / Implemented archive-backed path

01	**Public Galexie data** Closed Stellar public-network ledger batches.	Implemented
02	**NAS archive mirror** Partitioned compressed XDR retained for replay and recovery.	Implemented
03	**Bounded Rust decoder** Streaming zstd/XDR, concurrent decode, canonical ordering.	Implemented
04	**Compact evidence database** Contract transactions, exact graph edges, roots, versions, and rollups.	Implemented
05	**Serving database and API** Wallet aggregates, ranking indexes, bounded read queries, and demo data.	Implemented

Figure 2. *The running product path is historical: it processes closed archive partitions and serves a bounded dataset.*

01.3 / Target convergence

The future live adapter changes acquisition, not semantics. Stellar RPC or Captive Core will provide new LedgerCloseMeta records. A common validation and normalization boundary will feed the same versioned indexing modules used for offline replay. If a live gap appears, the local archive fills it before the tail resumes.

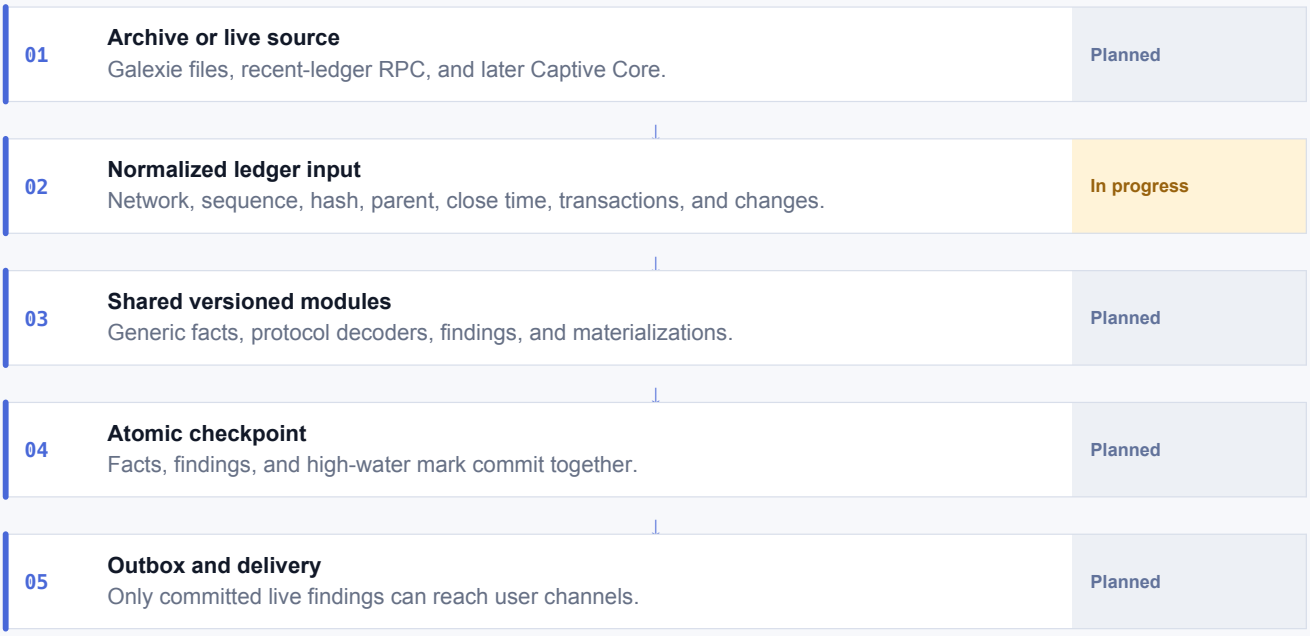

Figure 3. *Historical and live inputs converge before indexing so replay and real-time results cannot drift silently.*

01.4 / Design invariants

01 **Evidence is replayable**
Every derived fact retains network, ledger, transaction, and source coordinates.

02 **Commits stay ordered**
Decode may run concurrently; durable state advances in canonical ledger order.

03 **Backfill is silent**
Historical findings are queryable but never delivered as new live alerts by default.

04 **Unknown remains unknown**
Unattributed contracts keep their address instead of receiving an invented label.

05 **Read and policy state are separate**
The API cannot mutate evidence and user policy cannot rewrite chain facts.

02 / IMPLEMENTED

02 / Local archive and acquisition

The local Galexie mirror is both the bootstrap source and the engineering laboratory. It allows parsers, schemas, benchmarks, and incident rules to be replayed without depending on historical RPC retention.

IMPLEMENTED

Chapter status
The downloader mirrors closed public-network Galexie partitions onto the NAS, resumes existing data, and keeps the changing tip partition outside the completed set.

02.1 / A mirror, not a validator

Stargazer stores compressed Galexie LedgerCloseMeta batches on local NAS storage. It does not participate in Stellar consensus and it is not a network history archive server. Its role is narrower: preserve exact input evidence close to the indexer so derived databases remain disposable and reproducible.

BOUNDARY

Terminology boundary
Call this component a local Galexie archive mirror. Do not describe it as an archive node, validator, or independent publisher.

02.2 / Partition and retention model

Property	Current behavior	Reason
Source	Public Galexie pubnet object store	Complete LedgerCloseMeta evidence for replay
Partition size	Approximately 64,000 ledgers	Bounded download, discovery, and completion unit
Local store	NAS filesystem	Reuses bandwidth and keeps raw evidence near compute
Closed partitions	Marked complete after synchronization	Stable recovery and replay boundary
Tip partition	Synchronized but never marked complete	The newest public partition can still change
Concurrency	Filesystem lock prevents duplicate downloaders	Avoids downloading the same range twice
Retention	Runner selects an approximate recent-year window	Supports the current demo and benchmarking scope

Table 1. *Archive acquisition favors explicit completion state and reproducible local evidence.*

02.3 / Acquisition lifecycle

01	**Select range** Resolve closed partitions for the requested retention window.	Implemented
02	**Reuse local objects** Existing files with the expected size are not fetched again.	Implemented
03	**Synchronize partition** Bounded parallel transfers populate the local directory.	Implemented

04	**Verify and mark** Only a closed partition receives a durable completion marker.	Implemented
05	**Expose to walker** The indexer consumes completed partitions oldest first.	Implemented

The raw archive path is retained with ledger metadata in the compact database. A transaction can therefore be traced from a product result back to its canonical hash, ledger sequence, apply order, and compressed source file.

02.4 / Why local-first matters

Repeatable research Run a new incident rule against the same historical evidence and compare outputs.	**Fast parser iteration** New decoders avoid repeated remote downloads and provider retention limits.
Gap recovery A future live tail can repair missing ledgers from the same validated source range.	**Cost control** Bulk sequential archive processing is cheaper and more predictable than per-transaction RPC retrieval.

The local mirror is still an operational asset: its freshness, completion markers, storage health, and upstream recoverability require monitoring.

03 / IMPLEMENTED

03 / Memory-efficient decoding and indexing

The archive reader is designed around bounded memory and deterministic commits. Work may be parallelized before persistence, but the database always advances one contiguous ledger range in canonical order.

IMPLEMENTED

Chapter status
The optimized compact Rust path is the preferred indexer for contract-interacting transactions and the deployed demo dataset.

03.1 / Bounded reader

Pressure point	Control	Current boundary
Directory discovery	Partition-scoped walker	Only one partition of paths, normally at most 64,000
Compressed input	Size cap	128 MiB by default
Decoded input	Streaming cap	256 MiB by default
Decompression	zstd window bound	Window log 27
XDR nesting	Depth limit	Maximum depth 500
Parallelism	Fixed workers plus decode_ahead	Backpressure prevents an unbounded result queue
Persistence	Single ordered consumer	SQLite commits remain deterministic

Table 2. *Resource limits are explicit inputs to correctness, not tuning afterthoughts.*

03.2 / Concurrent decode, ordered consume

01	**Archive walker** Yield ledger files in sequence from one bounded partition.	Implemented
02	**Bounded job queue** Submit no more than the configured decode-ahead window.	Implemented
03	**Decode workers** Stream zstd directly into capped XDR parsing.	Implemented
04	**Ordered result buffer** Hold completed work until the next expected ledger is available.	Implemented
05	**SQLite ledger batch** Write facts and checkpoint in one transaction.	Implemented

Transaction envelopes and results are joined by the canonical network transaction hash and emitted in apply order. A filename, decoded ledger header, or expected sequence mismatch stops the pass rather than creating a silent hole.

03.3 / Commit and resume semantics

Invariant	Failure behavior	Recovery
One Stellar network per database	Startup rejects another network ID	Open or rebuild the correct database
Adjacent ledger hash continuity	Stop on a conflicting parent/hash	Treat as operator incident; do not guess
One contiguous committed range	Sparse append and prepend are rejected	Build a fresh database for an older start
Atomic ledger batch	Failed decode/write cannot advance checkpoint	Restart from last committed ledger
Idempotent resume	Already committed ledgers are recognized	Continue with the next expected ledger
Ordered persistence	Late worker result cannot leapfrog history	Wait or fail before committing later state

Table 3. *The high-water mark is meaningful only when facts and continuity checks commit atomically.*

03.4 / Compact retention policy

The preferred compact path does not store every Stellar transaction. It retains transactions that have contract calls, contract or diagnostic event touches, contract executable observations, or Wasm observations. This matches the product goal: wallet-to-contract dependency history, protocol usage, and contract version intelligence.

- Successful and failed contract-interacting transactions retain canonical transaction identity and evidence coordinates.
- Current state changes are materialized only from successful transaction ledger-entry changes.
- Raw transaction payload recovery remains possible from the archive path, ledger sequence, apply order, and transaction hash.
- The broad all-relationship index remains useful for comparison, but the compact contract-only index is the product path.

04 / IMPLEMENTED

04 / Evidence, databases, and read model

Stargazer separates append-oriented historical facts from account-oriented serving indexes. This keeps bulk indexing lean, makes long finalization resumable, and lets the public API reject stale or incompatible data.

IMPLEMENTED

Chapter status
Compact schema version 2, the wallet finalizer, the read-only API, and atomic serving snapshot replacement are implemented.

04.1 / Four stores, four responsibilities

Store	Purpose	Mutation boundary	Rebuildable
Raw Galexie archive	Compressed canonical ledger evidence	Downloader only	Usually, while upstream exists
Compact evidence SQLite	Exact graph, protocol roots, usage, executable history	Archive indexer only	Yes, from archive plus code/config
Wallet serving SQLite	Wallet totals, protocol/contract counts, ranking indexes	Finalizer only	Yes, from compact DB
Governance snapshot	Curated controller policy and executed-operation alerts	Horizon watcher only	Partly, from Horizon history
Alert-control SQLite	Telegram pairing and wallet policy	Control service only	No; user state must be backed up

Table 4. *Each process has one reason to write; evidence, serving, and user policy remain separate trust boundaries.*

04.2 / Compact schema groups

Group	Primary tables	Meaning
Provenance and checkpoint	compact_meta, compact_ledgers	Network binding, range, hashes, close time, raw path
Retained transactions	contract_transactions	Canonical transaction identity, result, and apply coordinates
Exact reverse graph	transaction_wallets, transaction_contracts, transaction_wallet_contracts, wallet_contracts	Participants, contract evidence, and exact dependency edges
Protocol attribution	protocols, protocol_contracts, transaction_protocols	Source-backed roots and per-transaction protocol usage
Rollups	contract_usage, protocol_usage	First/last ledger and interaction totals
Executable lifecycle	contract_executable_history, contract_current_executable, contract_availability_events, contract_code	Chronological executable and code observations

Table 5. *The compact database records evidence and deterministic materializations, not arbitrary product copy.*

04.3 / Exact wallet-to-contract relationships

Wallet identities normalize muxed addresses to base G accounts. Participant roles include transaction source, operation source, fee source, and authorized account. An exact dependency edge is stricter than co-occurrence: a direct call links only its effective operation source, while an authorization account links only to contracts inside that authorization subtree.

Included edge Effective operation source -> contract directly called by that operation.	**Included edge** Authorized account -> contracts inside its authorization subtree.
Excluded inference Fee-only participation does not create a dependency on every touched contract.	**Excluded inference** Event-only contract touches do not create a wallet-to-contract Cartesian product.

Figure 4. *The graph is evidence-backed and role-aware, which prevents inflated wallet coverage claims.*

04.4 / Protocol attribution and versions

Protocol attribution is exact-match and source-backed. The current registry contains 36 mainnet roots across seven protocol families, plus two contracts used by the YieldBlox incident fixture. Unknown contracts remain queryable by address. Factory-child expansion and time-versioned mappings are roadmap items.

Observation	Stored meaning	Alert meaning
Wasm upload	Code hash, byte length, and first-seen provenance	Precursor only; no contract instance changed
First state inside bounded window	Non-exact pre-window baseline	Not presented as deployment or upgrade
Created or changed executable	Chronological executable transition	Eligible deterministic upgrade evidence
Restoration with same executable	Availability event	Not an upgrade
Removal	Lifecycle event; history preserved	Availability signal

Table 6. *An uploaded Wasm blob and an executed contract upgrade are intentionally different facts.*

04.5 / Wallet finalizer and API

01	**Read compact DB** Open historical evidence strictly read-only.	Implemented
02	**Partition wallet records** Write bounded fixed-width records into 256 account shards by default.	Implemented
03	**Aggregate one shard** Deduplicate transaction-wallet pairs and add contract/protocol counts.	Implemented
04	**Build ranking indexes** Create transaction and protocol orderings after bulk aggregation.	Implemented
05	**Validate and rename** Publish a coverage-matched serving generation atomically.	Implemented

The API validates compact schema version, Stellar network binding, and serving coverage at startup and per generation. SQLite is opened read-only with bounded cache, concurrency limits, and query deadlines. Atomic file replacement becomes visible on the next request without restarting the API.

Endpoint	Primary capability
/health	Exact archive coverage and service capabilities
/api/wallets/{wallet}	Wallet activity, exact dependencies, versions, and relevant findings
/api/contracts	Known and unattributed contract usage
/api/leaderboard	Wallet activity ranked by transaction or protocol count
/api/protocols	Source-backed roots, usage, executable versions, and controller state

05 / PLANNED

05 / Live ingestion and archive convergence

A production live indexer must preserve the archive path's evidence semantics while adding restart-safe acquisition, lag monitoring, gap recovery, and a post-commit alert boundary.

PLANNED	**Chapter status** Continuous LedgerCloseMeta ingestion is not wired. The implemented Horizon controller watcher is a separate, narrow forward signal and is not the live ledger indexer.

05.1 / One index, two acquisition paths

The initial live source can use Stellar RPC getLedgers for the recent tail and short recovery. Captive Core can be introduced later when source independence and lower latency justify the operating cost. Both sources must emit the same normalized ledger envelope used by archive replay before any fact extraction begins.

Archive acquisition Closed Galexie batches selected by ledger range and local completion state.	**Live acquisition** Recent LedgerCloseMeta records plus explicit oldest/newest source range.
Common boundary Network, ledger, hash, parent, close time, transactions, events, and state changes.	**Shared result** Identical fact identity, parser version, persistence rules, and materialization semantics.

Figure 5. *Acquisition may differ; facts for the same ledger must not.*

05.2 / Atomic live commit

01 **Verify continuity**
Check network, sequence, ledger hash, and parent against the committed high-water mark.

02 **Decode evidence**
Normalize transactions, authorization trees, contract events, and ledger-entry changes.

03 **Run modules**
Execute generic facts, compatible protocol decoders, materializations, and findings.

04 **Commit together**
Write facts, findings, outbox candidates, and the new checkpoint in one transaction.

05 **Acknowledge after commit**
Transport retries can never advance durable state without its facts.

06 **Deliver after policy**
Only committed live findings can enter a durable user-delivery outbox.

05.3 / Archive-assisted gap recovery

01	**Detect gap or stale source** Compare next expected ledger and source availability.	Planned
02	**Freeze live checkpoint** Keep the last committed high-water mark unchanged.	Planned
03	**Resolve missing range** Read from the local mirror or an archive-capable provider.	Planned
04	**Replay through shared core** Apply identical continuity, module, and commit rules.	Planned
05	**Reconcile and resume** Join the recent tail only after the gap closes.	Planned

BOUNDARY

Conflict behavior
A conflicting ledger hash stops ingestion and raises an operator incident. Stargazer must never choose between histories automatically.

05.4 / Current forward monitoring

A separate Horizon watcher currently monitors eight curated Classic controller accounts across Aquarius, Phoenix, Reflector, and YieldBlox. It detects signer or threshold policy changes and high-signal successful controller-sourced operations after execution. It publishes complete snapshots atomically and exposes fresh, stale, or unavailable state.

BOUNDARY

Multisig scope
The ledger cannot reveal unsigned or partially signed envelopes exchanged off-chain. Pending multisig monitoring requires an explicit project coordinator or envelope-submission integration.

05.5 / Acceptance gates

- Restart resumes exactly from the last committed ledger and cannot duplicate facts.
- Archive and live paths produce identical materialized facts for an overlapping test range.
- A deliberate sequence gap or conflicting hash stops progress visibly.
- A live outage is repaired from retained history before the recent tail resumes.
- Ingestion lag, source failures, parser failures, checkpoint age, and recovery state are measurable.
- Historical replay cannot enqueue a user notification.

06 / IN PROGRESS

06 / Adding an indexer and backfilling history

Every new fact extractor should prove deterministic replay before it can influence live alerts. The local archive is therefore the development, validation, migration, and recovery surface for every protocol module.

IN PROGRESS

Chapter status
Offline replay, ordered commits, fixtures, and fresh database publication exist. A pluggable versioned module runner, automated staging promotion, and live-tail handoff remain planned.

06.1 / Backfill-first workflow

01 **Define the contract**
Assign a module version, output schema, supported Wasm versions, evidence coordinates, and coverage limits.

02 **Add representative fixtures**
Include normal operations, failures, upgrades, unknown versions, and incident transactions.

03 **Replay the local archive**
Write to a fresh staging database or generation with delivery disabled.

04 **Validate outputs**
Check determinism, continuity, duplicates, parser failures, resource use, and known incident expectations.

05 **Catch up**
Advance staging to a recorded handoff ledger using the same module version.

06 **Publish atomically**
Expose only a complete, schema-compatible, coverage-matched generation.

07 **Join the live cursor**
Run the same module on new ledgers and fill the transition gap from the archive.

Figure 6. *A new module joins live processing only after deterministic historical replay and atomic publication.*

06.2 / Module contract

Requirement	Why it is required
Stable module and schema version	Replays and migrations must identify the exact transformation used
Network and ledger coordinates	Every fact must return to canonical evidence
Transaction and operation/event coordinates	Avoid ambiguous incident and alert references
Supported executable/Wasm set	A protocol upgrade cannot silently run through an incompatible decoder
Generic fallback facts	Unknown contracts and code remain visible even when semantic decoding stops
Attribution source and confidence	Protocol identity and controller roles require verifiable provenance
Historical versus live origin	Replay findings remain queryable without becoming live notifications

Table 7. *Module versioning is part of the evidence record, not only a deployment label.*

06.3 / Promotion gate

Gate	Pass condition
Determinism	Repeated replay produces identical fact identity and materialized state
Concurrency parity	Serial and concurrent decoding commit the same result
Continuity	No missing, duplicate, or conflicting ledger in the target range
Coverage	Supported and unsupported executable versions are reported explicitly
Integrity	Schema, foreign keys, counts, and application invariants validate
Resource budget	Peak memory, write rate, and final database size stay within the approved envelope
Serving safety	The API never opens a partial or coverage-stale generation
Delivery safety	Historical replay creates no live delivery record
Rollback	The previously published generation remains recoverable

06.4 / Current platform limitation

The compact writer is currently monolithic and path-based. Schema version 2 rejects incompatible databases; the committed range is append-only and cannot be prepended; and materially changed protocol mappings require a fresh compact database so stale roots are not retained. The next engineering step is to extract a shared normalization, module, fact-identity, and commit interface.

Implemented	In progress	Planned
Fresh database replay and bounded finalization	Versioned normalized input boundary	Pluggable staging, parity checks, catch-up, and promotion

07 / IN PROGRESS

07 / Operations, recovery, and alert boundaries

Stargazer recovers from durable evidence and committed checkpoints, not assumptions. Every boundary must expose freshness, coverage, failure, and publication state independently.

IN PROGRESS

Chapter status
Archive and serving recovery behavior is implemented. Full live health metrics, durable alert outbox, channel delivery, restore drills, and paid activation remain planned.

07.1 / Operational health model

Signal	Purpose	Status
Archive partition freshness and completion	Proves the local recovery surface is current	Partial: markers and logs
Indexed ledger start/end and transaction count	States the exact bounded dataset	Implemented in health API
Last fetched, decoded, and committed ledger	Separates source, parser, and database lag	Planned for live
Parser failures by module/version	Prevents silent coverage loss after upgrades	Planned
Serving generation and coverage match	Stops stale rankings after compact range changes	Implemented
Controller snapshot freshness	Distinguishes fresh, stale, and unavailable governance state	Implemented
Outbox age, retry, and receipt state	Separates detection from notification success	Planned

Table 8. *A green API does not imply fresh ingestion or successful delivery; each boundary needs its own health signal.*

07.2 / Failure response

Failure class	Required response
Remote source unavailable	Keep checkpoint, retry with bounded backoff, then use the documented fallback
Archive ledger missing	Stop before a sparse commit; resynchronize the partition
Hash conflict	Stop and raise an operator incident; never choose history automatically
Unsupported XDR or Wasm	Preserve raw evidence and emit an observable coverage failure
Serving snapshot stale	Reject ranking claims, rebuild beside active file, validate, and rename atomically
Delivery channel unavailable	Retain committed outbox event and retry idempotently without rolling back chain facts

07.3 / Post-commit alert delivery

01	**Committed finding** Deterministic evidence is already durable.	Planned
02	**Policy match** Resolve subscribed wallet, protocol, severity, and entitlement.	Planned
03	**Durable outbox** Create an idempotent delivery key after the chain commit.	Planned
04	**Channel worker** Telegram first, then webhook and email.	Planned
05	**Delivery receipt** Record success, retry, failure, and user-visible history.	Planned

Telegram pairing and wallet alert policy are implemented, but no outbox or delivery worker is wired. Pairing must never be presented as active monitoring while delivery is unavailable. Paid activation and Stellar USDC settlement follow delivery reliability, not precede it.

07.4 / Backup priority

Asset	Regenerable	Priority
Registry research, parser versions, and configuration	No, not without reconstructing decisions	Highest
Alert policy and future delivery receipts	No	Highest
Raw local archive	Usually, while upstream objects remain	High: recovery time and bandwidth
Compact evidence database	Yes, from archive plus exact code/config	Medium
Serving snapshot	Yes, from compact database	Low

PLANNED

Recovery proof
A production milestone is incomplete until a clean-environment restore rebuilds the read path, validates coverage, and demonstrates source failover and gap recovery.

08 / IN PROGRESS

08 / Roadmap

The roadmap expands two dimensions together: protocol coverage and the monitoring surface around each dependency. The shared archive/live indexing core comes first because every later decoder and alert needs reproducible history and a safe handoff.

IN PROGRESS

Chapter status
The evidence core is implemented. Remaining milestones are ordered by technical dependency and acceptance evidence, not by unsupported calendar promises.

08.1 / Technical delivery sequence

01 **Archive-backed evidence core - implemented**
Local mirror, compact index, exact graph, executable history, wallet projection, API, and real-data demo.

02 **Shared archive/live core - in progress**
Normalized input, atomic checkpoint, source parity, and automatic archive gap fill.

03 **Backfillable module framework - planned**
Versioned modules, isolated staging, validation gates, live catch-up, and atomic promotion.

04 **Broader attribution - planned**
Factory-child discovery, time-valid roles, conflicts, and source-backed ecosystem roots.

05 **Protocol security semantics - planned**
Version-aware decoders, controls, oracle dependencies, and generalized incident rules.

06 **Wider signal surface - planned**
Governance, repositories, SDKs, packages, manifests, advisories, and core releases.

07 **Reliable delivery - planned**
Durable outbox, Telegram, webhooks, email, receipts, team review, and later paid entitlements.

08.2 / Protocol expansion waves

Wave	Coverage	Why it comes here	Exit criterion
First	Blend + Reflector + Classic SDEX	Cross-protocol lending and oracle dependency with a confirmed incident fixture	Market, oracle, reserve, and position facts replay deterministically
Next	Soroswap + DeFindex	Factory events make child discovery testable and improve wallet attribution	Pairs, vaults, and strategies are source-backed and time-aware
Then	Aquarius + Phoenix + FxDAO	Known administrative and runtime surfaces support high-signal monitoring	Upgrade delays, emergency controls, roles, and runtime roots are decoded
Close gaps	Templar, Sushi, and new projects	Avoid broad Stellar DeFi coverage claims with known attribution gaps	Production roots and controllers have public provenance

Table 9. *Protocol work is sequenced by dependency risk and verifiable on-chain surface.*

08.3 / Expand the monitoring surface

On-chain runtime Deployments, executable changes, calls, state, events, restorations, failures, and code hashes.	**Control plane** Admins, guardians, treasuries, governors, signers, thresholds, controllers, and executed payloads.
Developer surface Repositories, releases, SDKs, packages, manifests, interfaces, and breaking changes.	**Ecosystem context** Governance, advisories, Stellar protocol releases, Core/RPC changes, and incidents.

Deterministic evidence remains the verification boundary. AI can correlate sources, prioritize likely impact, and write a concise explanation, but it must retain primary evidence, parser version, confidence, and coverage limits.

08.4 / Roadmap acceptance standard

- Every monitored protocol publishes known contracts, roles, source URLs, and observed-at boundaries.
- Factory-backed protocols report discovered children and unresolved attribution conflicts.
- Every semantic decoder identifies recognized Wasm hashes and emits unknown-code coverage findings.
- Every live module passes deterministic archive replay and archive/live source-parity tests.
- Every alert retains coordinates, primary evidence, module version, confidence, origin, and coverage limits.
- Every delivery channel exposes retries and receipts independently from detection.
- Status surfaces distinguish delayed, unavailable, partial, and healthy sources.

A / IN PROGRESS

A / Capability status matrix

A compact statement of what Stargazer can demonstrate today and what remains an engineering target.

IN PROGRESS

Chapter status
Status is grounded in executable repository evidence as of August 2026.

A.1 / Current capability matrix

Capability	Status	Current boundary	Completion criterion
Galexie archive download	Implemented	Closed partition mirroring with completion markers	Operational freshness metrics
NAS evidence archive	Implemented	Local mirror for current retention scope	Backup and restore proof
Bounded Rust XDR decoder	Implemented	Streaming zstd/XDR with explicit caps	Live-source normalized input
Compact contract-only index	Implemented	Contract interactions, versions, usage, provenance	Pluggable module boundary
Exact wallet-contract graph	Implemented	Role-aware direct and authorization edges	More semantic protocol context
Root-seeded protocol attribution	In progress	36 roots, 7 families, 2 incident contracts	Factory children and time-valid mappings
Contract executable history	Implemented	Instance diffs and lifecycle semantics	System-event corroboration
Wallet serving projection	Implemented	Activity and ranking indexes	Automated refresh after range growth
Read-only API	Implemented	Bounded archive-backed queries	Coverage/status expansion
YieldBlox historical replay	In progress	Narrow deterministic IOC correlation	Generalized Blend/Reflector/SDEX decoder
Classic controller watcher	In progress	Eight curated accounts, post-execution	Historical state and wider controllers
Continuous live ledger indexer	Planned	Architecture defined only	Restart, parity, gap, and lag gates pass
Pending multisig monitoring	Planned	No generic on-chain source exists	Explicit project coordinator integration
Production alert dispatcher	Planned	Pairing/policy only	Durable outbox and receipt tests
Off-chain developer signals	Planned	Product narrative only	Versioned source adapters and evidence correlation

A.2 / Claims Stargazer must not make yet

- Production continuous Stellar ledger ingestion.
- Exhaustive Stellar protocol or DeFi coverage.
- Full semantic event, state, interface, or contract-spec decoding.
- Universal pending multisig or signer-approval detection.
- Historical incident notifications that were actually delivered at incident time.
- Active paid alerts or Stellar USDC settlement.

- An independently operated Stellar history archive or validator node.

BOUNDARY

Product wording
The current demo is a real-data, bounded historical index. Its API is live as a service, but ledger ingestion itself is not yet continuous.

A.3 / Evidence index

Topic	Repository evidence
Archive and compact index	ingester/README.md; ingester/reader-rs/src/archive.rs; decode.rs; compact.rs
Archive acquisition and NAS runner	ingester/scripts/download-galexie-pubnet.sh; index-one-year-compact-nas.sh
Wallet finalizer	ingester/WALLET_FINALIZER.md; stargazer-wallet-finalizer.rs
Read API and incident replay	ingester/reader-rs/src/api.rs; incidents.rs
Controller watcher	ingester/governance/README.md; governance watcher sources and tests
Alert control	ingester/alert-control-rs/README.md
Live and parser architecture	docs/stellar-archive-and-live-indexing.md; docs/stellar-defi-backfill-parser-plan.md
Protocol registry research	docs/stellar-contract-registry-research.md; ingester/reader-rs/registry

Historical and live intelligence must share deterministic evidence semantics, while acquisition, serving, and delivery remain separate trust boundaries.

Stargazer by Ferno - Technical architecture edition - August 2026

Public documentation: stargazer.ferno.ag/docs