MONOLYTHIUM

Whitepaper

V5.0 · MAY 2026

Settlement Layer for the Autonomous Economy

NETWORK	Monolythium · LYTH
STEWARDED BY	Mono Labs R&D LLC · Monolythium Foundation
LICENSE	Whitepaper text — CC BY-SA 4.0

Abstract

Monolythium is a Layer 1 blockchain designed as a settlement layer for the autonomous economy — a future in which humans, companies, AI agents, machines, and software services transact directly on open rails.

The chain is built around six positions:

- post-quantum accounts by default, with no classical signature acceptance path;
- Rust-first smart contracts compiled to a deterministic RISC-V execution target;
- native modules for tokens, NFTs, markets, bridges, payments, and agent commerce;
- a structurally bifurcated denomination that separates monetary privacy from commerce so the chain cannot be used as a fungible-anonymous-payment rail;
- a cluster marketplace that turns validator operation into a public, competitive market with distributed validator technology;
- no on-chain governance and no perpetual futures or margin, so the surface that can be captured, gamed, or weaponised is smaller.

Monolythium is not designed to be another EVM-compatible chain, and it is not designed to compete with the open agent-payment standards now shipping at scale (x402, AP2, ACP, MCP, Visa Intelligent Commerce, Mastercard Agent Pay). It is designed to **settle underneath** them: to be the chain-anchored trust, policy, escrow, identity, and reputation layer that those rails compose against when a transaction has to be enforceable, auditable, and portable across providers.

The public position is:

> Monolythium is not EVM-compatible at execution. It is EVM-connected at the liquidity edge. And it is composable underneath every major agent-payment standard at the settlement edge.

Assets move in through light-client and zero-knowledge bridges, cross-chain swaps, and issuer-supported integrations. Agent payments flow in through the major payment standards. Once value arrives, it settles through Mono-native standards, native markets, Rust/RISC-V contracts, and the eight agent-commerce primitives.

This document has three parts.

Part 1 — Why Monolythium explains the design philosophy: what the chain is for, the first commercial wedge, how it composes with existing payment standards, what it refuses, and the structural choices that follow from those refusals. It is written to be read end-to-end.

Part 2 — How Monolythium Works is the technical reference: consensus, cryptography, identity, execution, native modules, tokenomics, cluster operations, agent-commerce primitives, privacy, bridges, hardware, threat model, and recovery. It is written to be read in sections.

Part 3 — Adoption and Outlook covers developer experience, user experience, honest limitations, and what success looks like.

Contents

PART 1 — WHY MONOLYTHIUM

1. The Thesis: Settlement Layer for the Autonomous Economy

Most production blockchains today are honestly named **trading chains**. They optimize for the round-trip latency of a swap, the throughput of a perpetuals exchange, or the fee surface of a high-frequency strategy. Trading is fine; trading is not what the next ten years of digital economic activity will primarily look like.

The next ten years are **agentic**. An AI agent that pays for inference, hires a human contractor, buys data, settles an invoice, runs a marketing workflow, escrows funds against a legal deliverable, subscribes to compute, or pays another agent for a service is not science fiction. Each of those actions is technically possible today; what is missing is open, neutral, enforceable rails on which to settle them.

Today, an agent that wants to spend money does so through an account it shares with its user inside a closed AI platform, or through a payment-processor account leased by the user, or through a custodial wallet rented from a single exchange. Each of those rails is **owned**. The agent is rentable, the identity is rentable, and the reputation accrued through years of work is rentable. If the platform changes its terms, the agent's economic life resets.

Monolythium exists to provide a different option. The chain is built around the assumption that the most interesting category of economic activity in the next decade will be **delegated** — a human or an organization authorizes a software entity to act on their behalf, with bounded authority, against open infrastructure, across providers and jurisdictions. The substrate for that activity needs to be:

- programmable, so policy is enforceable in code;
- auditable, so principals can verify what their agents did;
- permissionless, so an agent cannot be deplatformed by a single counterparty;
- bridge-connected, so it interoperates with existing value;
- post-quantum, so identity survives multi-decade cryptographic horizons;
- and structurally hostile to use cases that depend on combining anonymous payment with anonymous service discovery.

The chain is useful even if the agentic category grows slowly. Native tokens, NFTs, spot markets, bridges, payments, and application-specific contracts are valuable in their own right. The strategic focus is to remain genuinely useful in both the success case (agent commerce becomes a major category) and the failure case (it grows slower than expected), and not to take design decisions that only pay off if a single optimistic scenario plays out.

The chain is the same chain it would be if no agent ever showed up. The user is different. The agent is the audience the chain quietly fits best, and quietly was designed for from the start.

What the chain does for an agent

An agent on Monolythium has a chain-native sub-account address. The address is a sub-account of a human or organizational principal — the agent itself is not a legal entity, and tying it to a principal keeps the legal concept of delegated authority intact. The principal issues a programmable spending policy that the protocol enforces: budget caps, allow-listed counterparties, time-of-day windows, per-call limits. The protocol cannot be talked out of the policy by the agent; the policy is consensus state, not local configuration.

An agent transacts through canonical **runbooks** — typed, versioned, signed templates that define every supported operation. An agent does not write raw RPC; it selects a runbook, fills in parameters, and submits. Runbooks make agent commerce legible, replayable, and auditable. They are the agent-economy analogue of the structured forms an institution would use rather than free-text email.

An agent accumulates reputation. Reputation is multi-dimensional — speed, quality, communication, accuracy — and is derived from on-chain history. It is portable: an agent's track record is not owned by any AI lab or any platform; it lives on the chain and follows the agent identity wherever it operates. A future agent that switches model provider keeps its history. This portability is structurally impossible inside a closed payment system.

An agent can hire and be hired. It can post an offer, negotiate counter-offers, escrow funds against deliverables, and resolve disputes through a configurable arbiter — a single arbiter for low-value tasks, a quorum for mid-value tasks, a human-credentialed arbiter for high-value tasks. The same primitive serves a human hiring an agent, an agent hiring a human, an agent hiring another agent, or two humans transacting with no agent involved.

A worked example

A small business has an AI assistant operating in its in-box. The owner deploys the assistant as a sub-account with a spending policy: monthly budget under five thousand dollars, transactions capped at one thousand, allow-listed to legal, accounting, and administrative services. A customer raises a privacy-policy concern. The assistant searches the on-chain discovery registry for "lawyer with crypto-DAO experience, rated four stars or higher, available this week, fixed quote under five thousand." Three matches return. The assistant posts an offer, negotiates a five-day instead of seven-day timeline with one of them, escrows the funds against deliverables, receives the redlined privacy policy, releases the escrow on review, and updates the customer.

The lawyer is human. The decision to hire was delegated. Settlement was near-instant. The reputation update was on-chain. No platform took a 30% cut.

The 30% cut also paid for real services on existing platforms — discovery, credential vetting, dispute resolution, consumer protection, and demand aggregation. Monolythium replaces or complements each function with on-chain primitives rather than removing the function:

- **Discovery** is the permissionless discovery registry; providers stake a small LYTH bond to list, and indexers compete to surface listings well.
- **Credential vetting** is the issuer registry and attestation primitive; markets weight which issuers they trust.
- **Dispute resolution** is escrow with pluggable arbiters — single arbiter, quorum, or human-credentialed depending on the value at stake.
- **Consumer protection** comes from spending-policy enforcement, the structural hostility of the bifurcated denomination to illicit commerce, and arbiter accountability.
- **Reputation** accrues to the provider's chain address as portable, multi-dimensional signal that survives any single venue.

What the chain does not do is **demand aggregation** — getting buyers to the discovery registry in the first place. That remains an application-layer problem. The chain provides the rails; an application that aggregates demand on those rails has the opportunity to do it at a cost structure that does not subsidize the rest of the platform stack, but the aggregation work itself still has to happen.

That story has many variants — replace the lawyer with another AI agent, replace the business with a household, replace the legal task with a compute job or a marketing campaign or a kitchen renovation — and the chain primitives are the same. The breadth is the point. **Universal services on a single primitive set, not domain-specific marketplaces stacked on top of a generic chain.**

The agent-wallet moment

The simplest way to understand the category is this:

> The moment an AI assistant can say, "Send me 50 USDC to complete this task," the agent-commerce era has started.

That does not mean the agent has unlimited authority. It means the user can give the agent a wallet, a budget, and a policy. The agent can request funds, receive approval, spend within limits, open escrow, pay a vendor, or settle a workflow. Every action is visible, revocable, and bound by rules the user controls.

This is different from AI-assisted checkout. A checkout protocol lets an assistant help a user buy from a single merchant. Monolythium's surface is broader: an assistant can have an economic account and operate under user-defined rules across many services, across many providers, without depending on any individual marketplace.

What the chain does not over-claim

The bet that an open, neutral settlement layer wins as the trust layer behind the agentic economy is not a sure thing. Existing payment networks, large model providers, and cloud platforms are shipping their own agent-payment infrastructure today. The chain's bet is not that those rails disappear. The bet is that **the rails will need a neutral, chain-anchored substrate beneath them** for the parts they cannot supply themselves — enforceable principal-level policy, escrow with dispute resolution, portable cross-platform reputation, and post-quantum-grade long-lived identity.

The chain is designed to be useful in either outcome. Spot markets, post-quantum settlement, bifurcated privacy, and verifiable bridges remain valuable on their own and would not need rebuilding if the agentic category grew slowly.

2. The First Wedge

A settlement layer for the autonomous economy is the long horizon. The first commercial wedge — where the chain enters real workflows — is narrower.

The wedge is agent escrow, spending policy, and reputation as the trust layer behind paid APIs and services.

The market is already converging on standardized payment handshakes for AI agents. Coinbase shipped x402 in 2025; Google's AP2 launched the same year; Stripe and OpenAI released the Agentic Commerce Protocol; AWS Bedrock AgentCore added native stablecoin payment rails through Coinbase and Stripe; Visa and Mastercard both announced agent-payment programs. These standards solve the **payment handshake** — how an agent discovers a paid endpoint, presents payment, and receives the response.

They do not solve four problems that are unavoidable once agents operate against more than trivial workflows:

- **Enforceable principal-level spending policy** — caps, allow-lists, time-of-day rules, expiry, revocation — that the agent itself cannot bypass. A wrapper around a card or a custodial wallet enforces policy at the wrapper; if the agent is granted the wrapper's credentials it inherits its full authority. The chain enforces policy at admission, in consensus state, on a sub-account the principal controls and the agent cannot rewrite.
- **Escrow with counter-offer flow and pluggable arbiters** for non-trivial work — the deliverable-against-payment use case that defines real services. A paid-API handshake does not negotiate timelines, attach penalty clauses, or arbitrate disputes; an escrow primitive does.
- **Portable cross-platform reputation tied to the agent identity** — speed, quality, communication, accuracy — that survives the agent switching model providers, runtimes, or wallets. A closed platform's reputation cannot follow the agent off the platform.
- **Chain-anchored consent records** the principal can revoke instantly, leaving in-flight commitments grandfathered. A consent stored in a vendor's database can be overwritten by the vendor; a consent stored on-chain has the principal's signature on it.

Monolythium ships those four primitives as native modules. They are the chain's first deliverable to the existing rails — not as competition, but as composition. An x402 paid-API endpoint can settle in stablecoin while binding the transaction to a Monolythium spending policy. An AP2 mandate can be recorded as an on-chain consent. An ACP checkout can escrow against deliverables through Monolythium and resolve disputes through a registered arbiter. An MCP tool boundary can expose the chain's runbook surface to any assistant the user runs.

The wedge is narrow enough to ship and measure. Adoption signal looks like:

- the count of agent sub-accounts created against Monolythium spending policies;

- the volume of escrow flowing through Monolythium for ACP/AP2/x402-mediated transactions;
- the depth of the discovery registry and the number of legitimate providers it hosts;
- the survival of agent identity and reputation across changes in model provider, wallet, or platform.

If those metrics grow, the broader settlement-layer thesis follows. If they do not, the chain still has its native markets, post-quantum identity, bifurcated denomination, and bridge surface — none of which depend on the agentic wedge succeeding.

---

3. Composition with Agent-Payment Standards

Monolythium is designed to compose **underneath** the open standards now shipping for AI-agent payments. Each standard solves a different layer of the stack; Monolythium provides the layer they all share.

LAYER	STANDARD	MONOLYTHIUM ROLE
HTTP payment handshake	**x402** (Coinbase, x402 Foundation)	Receives x402-initiated stablecoin settlement; provides on-chain receipt, escrow, and reputation against the payee
Agent intent mandate	**AP2** (Google, open)	Anchors the agent's mandate as on-chain consent + spending-policy registration on the agent's sub-account
Checkout flow	**ACP** (Stripe, OpenAI)	Settles ACP checkout outputs; provides escrow with counter-offer flow and dispute resolution for the seller
Tool / connector boundary	**MCP** (Model Context Protocol)	Exposes spending policy, runbooks, escrow, and discovery as MCP tools the assistant can invoke
Card-issuer agent flows	**Visa Intelligent Commerce, Mastercard Agent Pay**	Provides the chain-anchored settlement, identity, and reputation layer behind agent-initiated card payments

The composition is one-directional. Monolythium does not require the rails to adopt anything; the rails do their job (initiate payment, transmit intent, complete checkout) and Monolythium handles the parts they leave open (policy enforcement, escrow, dispute, reputation, post-quantum settlement).

Worked composition: x402 + Monolythium

An assistant requests a paid API. The endpoint returns `HTTP 402` with x402 payment details. The assistant's wallet has a Monolythium agent sub-account with a registered spending policy.

1. The wallet checks the policy. The endpoint matches the allow-listed category. The price is under the per-call cap. The aggregate spend this month is under the monthly cap.
2. The wallet signs a Monolythium transaction that pays the x402 invoice in stablecoin. The transaction's `purpose` field references the x402 payment ID. The Monolythium runbook is `pay_vendor` with parameters typed against the x402 receipt format.

3. The protocol verifies the spending-policy constraints at admission, refuses if any fail, includes the transaction if all pass.
4. Settlement happens at anchor finality (three to five seconds). The endpoint receives confirmation through its x402-compatible receiver, which also fetches the on-chain receipt.
5. The endpoint serves the response. The chain emits a typed event the principal's dashboard reads as part of the agent's audit trail.
6. If the response is disputed (the data was stale, the endpoint underperformed against SLA), the principal opens an escrow-arbiter dispute referencing the x402 transaction. The arbiter registry routes the dispute.

The agent does not implement any of this manually. The wallet, the runbook, and the spending policy carry the discipline. The endpoint integrates only with x402; the chain handles everything beyond the handshake.

Worked composition: AP2 + Monolythium

A user is shopping through an AI assistant. The assistant generates an AP2 intent mandate ("I want to book the 9pm dinner reservation for two, budget $150, prefer Italian"). The mandate is signed by the principal and presented to the merchant via AP2.

1. Before the assistant acts, the wallet registers the AP2 mandate as a Monolythium consent record on the principal's account. The consent is scoped to "restaurant booking" and has a 48-hour expiry.
2. The assistant negotiates with one of the merchant's AP2-compatible booking endpoints. A price and venue are agreed.
3. The assistant signs the Monolythium transaction that pays the booking. The protocol verifies the consent record matches the action (category, expiry, principal signature).
4. The transaction settles. The reservation is held.
5. The principal cancels the consent record. Any future booking attempts under the same mandate are refused at admission. The existing booking is grandfathered.

The principal owns the cancellation primitive. The merchant cannot continue to bill against the mandate after revocation. The on-chain consent is the principal's auditable record of authorization granted and withdrawn.

Why the composition stance

Two reasons.

The major rails will route value. Every credible projection of agent-commerce volume assumes the existing payment networks (cards, ACH, stablecoins over x402) move most of the flow. A chain that tries to displace them at the handshake layer fights a market

with deep distribution, mature compliance, and large customer balances. A chain that settles **underneath** them provides what the handshakes cannot supply and benefits from the volume regardless of which handshake wins.

The trust gap is real. A payment handshake solves payment. It does not solve the agent's authority, the principal's revocation, the deliverable's enforcement, or the reputation's portability. The chain solves those, and they get harder, not easier, as agent volume grows. The composition stance positions Monolythium to be the layer the rails reach for when the simple-payment model is no longer enough.

The chain is not an alternative to x402 or AP2 or ACP or MCP. It is the missing settlement, policy, escrow, identity, and reputation layer that any of those rails can compose against.

---

4. What Monolythium Refuses

A chain's identity is shaped at least as much by what it refuses to do as by what it does. Five refusals define Monolythium.

4.1 No on-chain governance

There is no governance token, no on-chain proposal contract, no protocol-level voting mechanism, no on-chain treasury voting, and no signaling extrinsic that pretends to be governance.

The reason is attack surface and clarity. On-chain governance, as deployed across a decade of L1 experiments, has consistently produced four failure modes:

- **Capture.** Large holders, custodians, and exchanges accumulate vote weight and steer outcomes.
- **Bribery markets.** Voting rights become rentable; outcomes follow whichever vote-buying market is liquid that week.
- **Theatre.** Low participation means a small, motivated minority decides everything, while the rest of the network's name is attached to the decision.
- **Direct control surface.** A working governance contract is a one-step path for an attacker to alter the protocol if they obtain a quorum of votes — by exploit, by accumulation, or by social engineering.

The chain rejects all four by not having the mechanism. There is nothing to capture, nothing to bribe over, nothing to vote against an empty hall about, and nothing to steer if you compromise a quorum.

The replacement model is honest about its centralisation in one place and aggressively decentralised in another:

- Protocol direction is set by accountable maintainers, in public, with explicit rationale.
- Community input is gathered through public channels and weighed.
- Operators choose whether to run new versions of the node software.
- The legitimate path of dissent is **forking**, covered in §8.

This is more honest than pretending a 4%-turnout vote is a popular mandate. It draws a clear line between **direction** (decided by maintainers and operators) and **consent** (expressed by running software, holding the asset, or forking off).

4.2 No perpetual futures or margin

The chain has spot markets. The chain does not have, and is not building, perpetual futures, margin trading, or any on-chain leverage venue. The protocol does not host a perpetuals exchange and does not mint a tradeable perp asset class.

The reason is product identity. Once a chain becomes a perpetuals venue, the design pressure on every other surface bends towards the perpetuals product — oracle dependencies, liquidation engines, funding-rate disputes, MEV around liquidations, leverage-driven liquidity, and a regulator audience that classifies the chain as a derivatives exchange whether it likes it or not.

Spot markets, payments, NFTs, agent-service settlement, cross-chain swaps, and conservative DeFi primitives are a deep ecosystem on their own. They are also, in this paper's view, a better foundation for agent commerce — agents do not need to trade perpetuals to settle real-world tasks. They need bounded budgets, predictable spot liquidity, escrow, and clean settlement.

This refusal also reduces the audit surface dramatically. A perpetuals venue is one of the most complex artifacts in DeFi; the chain removes that complexity entirely from the consensus-critical surface.

4.3 No EVM execution

Monolythium is **not EVM-compatible at execution**. Existing EVM bytecode does not run on the chain. Solidity is not the default developer model. The token, NFT, and account-abstraction conventions from the Ethereum ecosystem are not the chain's protocol standards.

This is a deliberate position. The chain is **EVM-connected at the liquidity edge** — value moves in and out through light-client and zero-knowledge bridges, cross-chain swaps, and issuer-supported integrations — but value, once it arrives, settles through Mono-native standards on a Rust/RISC-V execution layer.

The reasoning is in §9. The short version: trying to be a faster Ethereum has been tried and produces well-funded chains that nevertheless live inside Ethereum's frame. The chain optimizes for the audience that wants Rust contracts, AI-assisted developer ergonomics, post-quantum accounts, native modules for hot paths, and a smaller and cleaner protocol surface.

4.4 No fungibility between public and private money

Monolythium has two denominations of LYTH: **public** and **private**. They are not fungible. A LYTH that crosses from public into private cannot ever return to public, by any mechanism, including burn-and-reissue. A LYTH in the public denomination can be moved into the private denomination at the holder's choice; the move is one-way.

The private denomination supports two operations only: transfer to another private address, and burn. That is the complete set. Private LYTH cannot enter a smart contract, cannot trade on the spot order book, cannot bridge to another chain, cannot delegate to a staking cluster, and cannot pay for any service mediated by the discovery registry.

This is the most defensible privacy posture available to a general-purpose L1. It is detailed in §5 and the cordon implementation in §19.

4.5 No bundled AI model

The chain does not bundle a particular AI model with the protocol. Agent identity is chain-native, but the model behind any agent is the user's choice. Runbooks are typed and signed; they are not "AI prompts." The chain is model-agnostic on principle, and is built to be useful for agents driven by open-source models, frontier proprietary models, narrow domain-specific systems, and non-AI software agents alike.

This refusal matters because it preserves the chain's neutrality. A protocol that bundled a specific lab's model would tie its reputation, longevity, and credibility to a single vendor's roadmap. The agent economy needs the opposite — a substrate the model providers can plug into rather than one that picks the winning model in advance.

5. Bifurcated Denomination: Privacy Without Contamination

Privacy on a public ledger has, over the last several years, been on a steady regulatory collision course. The two historical trajectories are visible.

A chain with fungible shielded and transparent denominations — funds can flow freely between shielded and transparent and back — ends up delisted by major exchanges, because shielded-to-transparent flow contaminates the entire public denomination. An auditor cannot distinguish a deposit that came directly from a transparent address from a deposit that came from a transparent address that was funded yesterday by a shielded address that was funded the day before by a transparent address that was funded by an address on a sanctions list. Fungibility breaks the audit story for everyone.

A chain with a single private-by-default denomination ends up delisted by major exchanges, because there is no public denomination to clear. The audit story is “we cannot tell you anything.” Exchanges respond by removing the asset entirely.

Both trajectories converge on the same outcome: a privacy-supporting L1 ends up delisted, regardless of which mechanism it used. Monolythium’s bifurcated denomination is a third option that has not been deployed at scale by a major L1.

How bifurcation works

The protocol enforces non-fungibility at the consensus layer. Every account holds two balances — public and private — represented as separate state entries with separate spendable conditions. Every transaction operates on exactly one denomination. A native caller-origin cordon (described in §19) prevents a public contract or module path from receiving private-denominated value, and prevents private-denominated state from being used in any public contract context. Privacy modes — stealth addresses for sender/recipient unlinkability, confidential transactions for amount hiding — are available **within the private denomination only**.

A user can move LYTH from public to private at any time. The protocol records the movement as a **crossing** — the public balance decreases, the private balance increases by the same amount, the corresponding amount of LYTH is permanently shifted into the private denomination. The reverse direction does not exist. There is no module, no instruction, no foundational mechanism that allows private LYTH to become public LYTH again. A user holding private LYTH who wants public LYTH must acquire fresh public LYTH through a counterparty transaction.

What this gives a regulator or an exchange

A regulator or exchange treats the **public denomination** as fully auditable. Public activity is transparent, traceable by the analytics tooling that already exists for other transparent chains, and visible to subpoena. A KYC-supervised exchange can custody public LYTH with the same audit posture as it custodies any other transparent asset.

A regulator or exchange treats the **private denomination** as opaque by design. They are correct to do so. The protocol does not pretend otherwise. An exchange may choose to refuse private-denominated deposits entirely; an exchange may choose to require a "proof of crossing" before accepting them (the user demonstrates, via an optional application-layer proof, that the private LYTH crossed from a public address that was KYC'd at the time of crossing); an exchange may choose to accept private deposits at a higher KYC tier. The chain does not constrain the choice. The chain provides the **structural separation** that makes the choice coherent.

A user who values privacy crosses LYTH into the private denomination and uses it for peer-to-peer transfers. They cannot pay an exchange fee with private LYTH; they cannot invest private LYTH in a contract; they cannot hold private LYTH on a centralized venue that does not accept it. They retain monetary privacy for their direct transfers. The privacy is real. The cost is structural — privacy-denominated value cannot participate in the broader chain economy, by design.

Why this is the position

The regulatory soundbite, stated plainly:

> Monolythium separates monetary privacy from commerce. The private denomination supports peer-to-peer transfers only — it cannot pay for services, interact with smart contracts, or trade. The public denomination supports full commerce and is fully transparent and analyzable. This bifurcation makes the chain structurally hostile to use cases that depend on combining anonymous payment with anonymous service discovery. Compliance happens at frontends and fiat boundaries; the protocol stays neutral.

It is true, defensible, and not common in production.

Side effect: structurally hostile to illicit commerce

The pattern that history calls "the darknet marketplace" requires the simultaneous availability of an anonymous-payment rail and an anonymous-service-discovery surface. Monolythium's bifurcation makes those two requirements impossible to satisfy on the same chain. The private denomination has no service-discovery surface — it cannot

interact with the discovery registry, cannot escrow funds, cannot dispute. The public denomination has full service discovery but is fully transparent. A user can have anonymous money or service discovery, never both at once.

This is not the only defense the chain offers against illicit commerce — frontend curation refuses to surface illegal listings, arbiters refuse to arbitrate illegal disputes, attestation tilts markets toward legitimate providers, and fiat on-ramps apply KYC as they would for any other asset. But the bifurcation is the structural defense, the one that does not depend on application-layer compliance or post-hoc enforcement. It is by construction.

Institutional accumulators

A common question: what happens when an institution wants to accumulate LYTH privately for treasury purposes? Bifurcation answers cleanly. The institution holds public LYTH in a transparent address; if they want to obscure their accumulated balance from competitors, they cross LYTH into private and hold the private balance. The private balance is opaque to external observers but is also locked from any chain-native productive use — it cannot stake, cannot delegate, cannot earn yield from a contract. The institution faces a real trade-off: privacy on treasury holdings or chain-native productive use, but not both. They can split: hold a productive position publicly and a private reserve privately.

6. Cluster Marketplace: Validator Operations as a Public Market

Monolythium does not have validators in the traditional single-operator sense. It has **clusters**: 7-of-10 operator-threshold groups that produce one logical validator vertex per cluster, per consensus round. A cluster is staffed by ten independent operators, runs continuously, and tolerates up to three operator outages without missing a beat.

The full cluster set at the target scale is **100 clusters × 10 operators = 1,000 operator positions across the network**, with seven active and three standby positions per cluster (1,000 = 700 active operator seats + 300 standby).

The reasoning is direct: a single validator per stake-weighted position is a single point of failure, a single point of slashing, and a single source of geographic, network, and jurisdictional risk. A cluster is structurally less brittle in all three dimensions.

The marketplace

Cluster membership is a public market. Operators publish their **node attestation** (hardware class, network metadata, geographic claim, uptime track record, service-tier capacity), and clusters compose themselves from operators offering complementary skills:

- a bare-metal operator in one region for raw throughput;
- a cloud operator in a second region for geographic and network diversity;
- a home operator in a third region for jurisdictional diversity;
- a GPU-equipped operator for the cluster's prover service tier;
- an archive-capable operator for the cluster's RPC and archival service tier.

Clusters compete on reliability, latency, uptime, geographic diversity, service-tier breadth, hardware quality, and reputation. Stakers see all of this through wallet and explorer surfaces and delegate accordingly. The "best validator" is not picked by social reputation; it is composed in public by people who can read the same data the chain reads.

This is sometimes shorthanded as the **"Avengers assembly"** model — a real differentiator. Most chains hide operator selection behind opaque delegation lists in a wallet; Monolythium turns the composition itself into a public market with first-class wallet, explorer, and reputation support, so operator quality is visible to delegators rather than tucked behind a brand.

Why this matters for agent commerce

Agents need reliable infrastructure. A chain designed for autonomous settlement should make infrastructure quality visible and economically meaningful, so that operator quality is a market signal rather than an opaque social fact. Cluster reputation is the visible quality of the substrate the agent runs on. Operators with strong uptime, broad geo-

graphic coverage, and rich service-tier offerings command premium delegation; operators that go offline or run weak hardware lose market position. The agent economy benefits from a market that prices infrastructure quality, the same way the cloud economy is priced by reliability.

Standby capacity and graceful failover

Each cluster has a standby of three operators in addition to its seven active members. A standby operator participates in the cluster's communications, holds the threshold key shares ceremony's transcript, and is ready to step in when an active operator goes offline or is rotated out.

The standby surface is part of what makes the cluster model **operationally resilient** rather than just structurally redundant. An active operator can be removed (by the cluster, by network policy, or by their own departure) without a re-keying ceremony, because the standby slot is already provisioned. The next-eligible standby is promoted, the threshold continues to hold, and the cluster keeps producing.

Multi-cluster operator caps

A single operator entity cannot dominate the cluster set. Operator identity is verified across all cluster memberships, and a hard cap limits how many active cluster positions any single operator can hold. This prevents a successful operator from accumulating positions across many clusters and reproducing single-operator centralisation on top of the cluster substrate. Service-tier specialization (GPU, RPC, archive, oracle) is encouraged across clusters; consensus-position multiplicity is bounded.

The combination — public cluster composition + standby capacity + per-operator multi-cluster caps + per-wallet delegation caps (§16) — produces a network that does not naturally concentrate, even under sustained success of any single operator or cluster.

7. Post-Quantum from Genesis

Monolythium uses post-quantum cryptography as its **default** at the user-facing signature layer. ML-DSA-65 (NIST FIPS 204, also known as Dilithium Level 3) is the only signature primitive accepted at transaction admission. There is no classical fallback, no hybrid mode, no ECDSA acceptance path "for compatibility." Every account signs every transaction with a post-quantum primitive from the chain's first block.

This is unusual. Most production EVM-compatible chains use classical secp256k1 or Ed25519. Monolythium requires mandatory post-quantum user signatures while still supporting the cross-chain bridge edge that EVM-resident value lives behind.

It is also a deliberately understated headline. **Post-quantum is a property of the chain, not its purpose.** The chain exists to settle the autonomous economy; post-quantum cryptography is one of several properties that make that settlement layer trustworthy on a multi-decade horizon. The audience the chain serves needs identity that remains unforgeable for the lifetime of an agent's reputation — not because "quantum-proof" sells, but because that is the audience's real requirement.

The whole stack, not just signatures

Cryptographic posture is end-to-end:

LAYER	PRIMITIVE	USE
User signatures	**ML-DSA-65** (FIPS 204)	Every user-signed transaction
Emergency recovery	**SLH-DSA** (FIPS 205, hash-based)	Pre-registered backup; activated under emergency rotation
Key encapsulation	**ML-KEM** (FIPS 203)	Peer-to-peer Noise handshakes, RPC TLS, stealth-address derivation
Aggregate signatures	**BLS12-381**	Per-cluster threshold aggregation for consensus throughput
Threshold key encapsulation	**Ferveo over BLS12-381**	Encrypted-mempool body decryption at block inclusion
Zero-knowledge verification	**SP1 zkVM + Groth16-BN254**	Application-layer proof verification (zkML attestations, high-value off-chain computation)
Hash	**BLAKE3**	State-tree leaves, Merkle commitments, content-addressed proofs, address derivation

Three properties of this stack are worth highlighting because they reflect design choices rather than ingredient-list ticks:

- **No hybrid signature mode.** The chain does not accept "signed with ECDSA AND ML-DSA-65"; it accepts only ML-DSA-65. The reasoning is below.
- **Post-quantum at the deep finality tier.** Block-level aggregation uses BLS12-381 for bandwidth (BLS aggregates are tiny). Every hundred blocks, the chain commits a **quantum-attested checkpoint** signed in ML-DSA-65 by each cluster's operators. Bridges, exchanges, and high-value cross-chain attestations bind to checkpoint-level finality. A quantum attacker who forges BLS cannot forge ML-DSA-65 — the fork dies at the next checkpoint.
- **Bounded classical residual.** The only classical primitive in the chain is the per-block BLS12-381 aggregate. Its exposure is bounded to a five-minute checkpoint cadence. The honest cost-benefit: pure ML-DSA-at-consensus would tax per-block bandwidth roughly three thousand-fold; the chain instead pays bounded residual exposure with a clearly documented migration path if the industry produces a quantum-secure aggregate signature.

Why pure post-quantum, not hybrid

The temptation in this transition era is to ship "hybrid" mode — a transaction signed by both ECDSA and a post-quantum signature, with consensus accepting either. Hybrid is appealing because it preserves user-experience continuity and offers theoretical defense in depth.

Monolythium rejects hybrid for three reasons.

First, the defense-in-depth argument is structurally weak. A hybrid signature is only as strong as its **weakest** component, not its strongest. If ECDSA is broken on a sufficiently large quantum computer and an adversary can forge ECDSA signatures, the adversary can forge hybrid signatures by forging the ECDSA half and reusing the genuine post-quantum half from a previous transaction. Hybrid does not protect against the threat the post-quantum primitive was supposed to protect against.

Second, hybrid creates audit ambiguity. Two acceptance paths means two code paths. Two code paths means twice the surface for implementation bugs, twice the surface for downgrade attacks (an adversary suppresses the post-quantum half and forces verification through the classical path), and twice the verifier complexity. With one signature primitive, the verifier does one thing; the audit story is "ML-DSA-65 verification is correct" rather than "the verifier correctly handles three combinations of two signature types." This is a meaningful simplification.

Third, hybrid postpones the migration. A chain that ships hybrid commits to the migration twice — once now (to add the post-quantum primitive) and once later (to remove the classical primitive). The migration later is harder, because by then ECDSA has been

deprecated for years and tooling has accumulated around the assumption that classical signatures work. Monolythium does the migration once, at genesis, when there is no installed base of classical-only wallets to support.

The cost of pure post-quantum is signature size — ML-DSA-65 signatures are roughly 3,309 bytes versus ECDSA's 64 bytes. The cost is real and is paid in storage and bandwidth. The alternative is paying the migration cost twice.

The quantum hedge

Post-quantum cryptography is not unconditionally future-safe. Cryptographic primitives have historically had finite useful lifetimes, and lattice-based cryptography is no different. The honest position is that ML-DSA-65 is the right choice for the foreseeable horizon, that it will likely remain a good choice for a decade or more, and that a future cryptographic break should be planned for rather than denied.

The chain plans for it through a small, targeted mechanism: the **emergency-key registry**. A user can pre-register a backup key in a different cryptographic family — specifically SLH-DSA (FIPS 205, hash-based) — alongside their primary ML-DSA-65 key. Hash-based signatures rest on different mathematical assumptions than lattice-based signatures; an attack that breaks ML-DSA does not, on current understanding, break SLH-DSA, and vice versa. Registration is one-time and lives in a registry trie that does not widen the account record.

If an emergency algorithm rotation is declared, the protocol refuses transactions signed with the broken algorithm at a specified anchor height. Users with a registered backup rotate to their hash-based key and continue transacting. Users without a registered backup are **frozen** — never drainable by the attacker, recoverable through a runbook claim process. The break-day outcome is bounded.

8. Forking as the Exit Path

The chain refuses on-chain governance (§4.1). The replacement model needs an answer to the question: what happens if a significant fraction of holders, operators, or users disagree with the chain's direction?

The answer is **forking is legitimate**. Not as a threat, not as a last resort, but as the explicitly documented path of dissent.

The chain's text and source code are released under licenses that permit forking. Operators are free to run alternative software. The cluster substrate, the cryptographic primitives, the SDK, the explorer, and the wallet are all open enough for an alternative project to take Monolythium's design and run with it under different leadership, different parameter choices, or a different philosophy.

This is the inverse of the "governance capture" failure mode. A protocol that uses on-chain governance to absorb dissent must keep losing capture battles to remain credibly neutral; a protocol that says "if you disagree, fork" makes its neutrality the absence of a capture surface. There is nothing to capture, because direction is set by maintainers and operators, and exit is set by anyone with a compiler.

The point is not that the network expects forks; the point is that the right to fork is the structural guarantee against capture, and it is the only one the chain offers — but it is real.

---

9. Why Rust and RISC-V

The execution layer is Rust-first smart contracts compiled to a deterministic RISC-V execution target. There is no Solidity, no EVM bytecode, no Ethereum precompile model in the mainnet execution path.

EVM compatibility is valuable because it answers many practical market questions at once. When a developer asks whether a new chain is EVM-compatible, they usually mean:

- Can I bring assets over?
- Can I bridge major stablecoins, ETH, BTC, and other large assets?
- Can my wallet display the assets and route risk visibly?
- Can I create and trade tokens?
- Can my application reach users without rebuilding every surrounding integration?

EVM-compatibility answers those by inheriting the Ethereum tooling surface: token standards, contracts, wallets, explorers, auditors, libraries, and liquidity routers.

Monolythium answers them differently. The **execution** layer stays Rust/RISC-V-native; the **liquidity** layer connects outward through zero-knowledge and light-client bridges, cross-chain swaps, and issuer-supported integrations. The chain avoids isolation while keeping the base layer smaller and cleaner.

What Monolythium gives up by not being EVM-compatible

Real costs:

- existing EVM bytecode does not run;
- Solidity is not the default developer model;
- Hardhat, Foundry, OpenZeppelin, the Uniswap fork lineage, and proxy-pattern ecosystems do not carry over directly;
- some Solidity-focused developers and market makers will ignore the chain;
- Mono must build its own SDK, ABI, wallet UX, explorer support, token standards, and testing tooling.

This is paid intentionally.

What Monolythium gets

The upside is a stronger identity and structural advantages.

- **Rust** provides strong compiler feedback, clear state-machine design, ergonomic testing and fuzzing, and better alignment with AI-assisted development than Solidity. Rust does not make contracts magically safe — poor economics, bad access

control, oracle mistakes, bridge bugs, and unsafe host interfaces can still cause serious failures — but it removes entire classes of avoidable bugs, makes contracts easier to reason about, and reduces the amount of custom code needed for common financial primitives.

- **RISC-V** is open, simple, portable, deterministic, and increasingly aligned with the zero-knowledge proving ecosystem. It gives Monolythium a clean execution target without inventing a custom bytecode language. Tooling, debuggers, formal-verification work, and zero-knowledge provers all converge on RISC-V; the chain rides that convergence rather than fighting it.
- **Native modules.** Common financial and agent-commerce primitives — token transfers, NFT ownership, spot-market order placement, bridge proof verification, agent spending-policy checks, account permissions — can be implemented once and audited once at the protocol level. Applications then compose against audited native modules rather than reimplementing primitives in user code.
- **AI-assisted development.** AI coding assistants are dramatically more useful in Rust than in Solidity. The Rust ecosystem has fifteen years of training data; Solidity does not. The next decade of contract development will lean heavily on AI assistance, and Rust is structurally a better target language for it.
- **Cleaner audit surface.** Each native module is audited at the protocol level. Applications that compose on top inherit that audit work. The chain pushes the heaviest cryptographic and financial logic into the well-audited core and leaves applications a smaller, simpler API to compose against.

The chain has chosen a harder adoption curve in exchange for a cleaner identity, a stronger long-term foundation, and a better fit for the workloads it expects to serve. EVM chains compete on liquidity gravity. Monolythium competes on a different axis: settlement for the next decade's economic actors, on rails that are not borrowed from the previous decade.

10. Market Positioning

Monolythium does not have a perfect mirror competitor.

- Ethereum and EVM-compatible chains compete on liquidity and developer familiarity.
- High-performance non-EVM chains compete on throughput and consumer applications.
- High-performance trading chains compete for low-latency derivatives flow.
- Open agent-payment standards (x402, AP2, ACP, MCP) compete for the payment handshake — Monolythium composes underneath them rather than against them (§3).

Monolythium's differentiated combination is:

- AI-agent settlement as a primary category;
- Rust/RISC-V-native execution from the base layer;
- post-quantum accounts as default, not optional;
- native MRC token, NFT, market, and agent-commerce modules;
- zero-knowledge and light-client bridge liquidity rather than EVM execution compatibility;
- no on-chain governance;
- no mainnet perpetuals;
- structurally non-fungible public/private denomination;
- a public cluster marketplace with distributed validator technology;
- focused use of zero-knowledge proofs at bridges, swaps, zkML, and high-value verification;
- direct composition with the major agent-payment standards as the chain-anchored trust and settlement layer.

If the market only wants another EVM chain, Monolythium has chosen the harder path. If the market wants safer open settlement for agents, payments, spot markets, and long-lived digital identities, the chain has a distinct position. The bet is not that EVM disappears — it will not, for at least a decade. The bet is that the settlement layer for the next decade's most interesting workloads will not be a copy of the chain that hosts the last decade's.

Why large participants would use a shared network

A large company can build infrastructure. That does not automatically make its infrastructure a neutral settlement layer.

For agent commerce, neutrality matters because agents transact across companies, model providers, wallets, jurisdictions, and service providers. A payment rail owned by one AI lab or one platform may work inside that platform, but it is less credible as a universal settlement substrate. A bank or a fintech that wants its agent flow to interoperate with the rest of the agent economy needs a substrate that is not owned by a competitor.

Monolythium's value to large participants is that it provides shared settlement rules, verifiable state, portable agent identity, portable reputation, common bridge and market standards, open infrastructure participation, and a network where counterparties do not need to trust one company's private ledger. Large participants may still run clusters, build applications, issue assets, operate provers, or provide liquidity. The point is that joining the shared network is more useful than fragmenting into private chains.

PART 2 — HOW MONOLYTHIUM WORKS

11. Consensus: Starfish-C

Monolythium's consensus engine is **Starfish-C**, a leaderless DAG-BFT protocol derived from the Starfish family of distributed-acyclic-graph consensus designs. Starfish-C provides deterministic linearization of a directed-acyclic graph of cluster-signed vertices, **three-second deterministic finality** under the partial-synchrony assumption (three to five seconds typical, with an outer bound of about eight seconds under degraded networking before a view-change is triggered), and equivocation handling that produces succinct, on-chain proofs and a one-hundred-percent slash plus permanent operator exile on detection.

The three-second cadence is chosen to leave headroom for cluster-marketplace geographic diversity — operators on any continent participate cleanly — and to allow encrypted-mempool threshold decryption and aggregate signature collection under load. The chain optimizes for steady performance under adversarial conditions, not racetrack speed under ideal conditions.

Safety

Safety is guaranteed through **deterministic linearization** of the DAG. Three properties combine.

No forks. Clusters sign exactly one vertex per round. A vertex is valid only if it references a quorum (2f+1, where f is the Byzantine bound) of vertices from the prior round. Two vertices signed by the same operator at the same round constitute a slashable equivocation.

The causal cone. When a wave leader is committed, the leader's entire causal history — the recursive set of vertices the leader's vertex transitively references — is committed in a deterministic linear sequence. The linearizer is a pure function of the DAG state; two honest clusters starting from the same DAG state derive byte-identical block sequences.

Bounded reorg. The maximum reorg depth is the size of the causal cone for an unfinalized leader, bounded by protocol parameter. An adversary cannot cause a reorg deeper than this bound without controlling more than f Byzantine clusters, which is excluded by the Byzantine threshold assumption.

Liveness

Liveness holds under partial synchrony — the standard BFT assumption that messages are eventually delivered within an unknown but finite bound. Starfish-C achieves liveness through wave-based progression: time is divided into waves, each wave selects a leader whose committed vertex anchors the linearization of its causal cone, and if a

wave leader fails to be committed within its timeout, the linearizer executes a skip round and the next successful leader's commit covers the skipped wave. Malicious subsets cannot stall the network as long as a quorum (2f+1) of honest clusters is active.

Leader selection

Wave leadership is selected by a **threshold-VRF** seed derived from the previous wave's cluster aggregate signature. The aggregate is a canonical-form threshold BLS signature with set-independent Lagrange interpolation: for any 2f+1 honest set, the aggregate evaluates to the same constant, so the seed is independent of which 2f+1 honest partials are combined. This guarantees that an adversary cannot influence the seed by selectively withholding their partial signature.

The seed feeds a VRF native module that produces the next wave's leader assignment. Anti-grinding is structural — the seed is produced **after** wave vertices are gossiped, so clusters cannot influence future leadership through block content selection.

Anti-equivocation

Double-signing — an operator signing two distinct vertices at the same round — triggers a **succinct equivocation proof** (the two conflicting signed vertices, both verified against the same operator's signature key) and a **100% slash of the operator's self-bond**. The operator is permanently exiled from the cluster set. Recovery by re-bonding is not permitted; the operator identity is burned.

The slash is exemplary, not just punitive. A protocol that tolerates equivocation invites it. A protocol that destroys the equivocating operator's stake and bars them forever does not need to tolerate it.

Anchors, blocks, and waves

The user-facing unit on Monolythium is the **anchor** — the canonical, finalized point in the chain history at which state transitions become irreversible. Anchors are produced from waves through deterministic linearization of the DAG's causal cone. Multiple internal layers exist:

- **Vertex.** A single cluster's signed round payload.
- **Wave.** A round of vertex production across the cluster set.
- **Anchor.** A deterministic linearization point in the wave structure; the user-facing unit of finality.
- **Block.** The traditional name preserved for compatibility with EVM-style RPC fields (such as `eth_blockNumber`) where an external tool expects a height-like integer. The block sequence is an aliased view of the anchor sequence.

Wallets, explorers, and the chain's SDK use **anchor** as the canonical term. A skipped wave produces zero anchors; an anchor that is committed advances the anchor height. The aliased block view exists for cross-chain tooling that needs a familiar integer label.

Performance against attack

Two properties matter most under adversarial load.

Chain-bomb resilience. A chain-bomb attack — an adversary attempting to flood the DAG with malformed or out-of-order vertices to inflate end-to-end latency — is bounded by a forward-clock cap parameter. Vertices more than N rounds in the future relative to the receiver's local wave clock are dropped at the gossip layer.

Payload efficiency. The amortized per-round payload cost scales linearly in the cluster count via Reed-Solomon shard dissemination, rather than quadratically as in earlier DAG-BFT designs. The result is a protocol that scales to substantially larger cluster sets than the current target without an architectural rewrite.

What this gives users

For a user or a builder, the important outcomes are:

- transactions finalize under a defined, mathematically grounded consensus process;
- anchor checkpoints can be verified;
- bridges and indexers can rely on canonical state commitments;
- operators have clear responsibilities and incentives;
- adversarial conditions degrade the chain's latency, not its safety.

12. Cryptography

This section specifies the cryptographic primitive set in implementation detail. Section 7 covers the design philosophy.

12.1 Primitive set

LAYER	PRIMITIVE	USE
User signatures	**ML-DSA-65** (FIPS 204, Dilithium Level 3)	Every user-signed transaction
Emergency backup signatures	**SLH-DSA** (FIPS 205, hash-based)	Pre-registered backup; activated under emergency rotation
Key encapsulation	**ML-KEM-768** (FIPS 203, Module-Lattice KEM)	Peer-to-peer Noise handshakes; RPC TLS; stealth-address derivation
Aggregate signatures (consensus)	**BLS12-381**	Cluster threshold aggregate, VRF, distributed key generation
Threshold key encapsulation (mempool)	**Ferveo over BLS12-381**	Encrypted-mempool body decryption at anchor inclusion
Zero-knowledge verification	**SP1 zkVM + Groth16-BN254**	Application-layer proof verification (zkML attestations, high-value off-chain computation)
Hash	**BLAKE3**	State-tree leaves, Merkle commitments, content-addressed proofs, address derivation

There is no Ed25519 acceptance path. There is no hybrid signature mode. The protocol validates exactly one signature primitive at transaction admission: ML-DSA-65 (with SLH-DSA as the emergency-rotation alternative, never coexistent).

12.2 ML-DSA-65 (Dilithium Level 3)

Standard. NIST FIPS 204, parameter set 3 — security comparable to AES-192.

Security basis. Module-Learning-With-Errors (M-LWE) hardness. M-LWE is the lattice problem on which Dilithium rests. No known quantum or classical algorithm breaks M-LWE in feasible time at the parameters used.

Performance. Verification on commodity hardware completes in ~0.12 ms per signature; signing in ~0.18 ms. Signature size is 3,309 bytes. Public-key size is 1,952 bytes. Private-key size is 4,032 bytes.

Determinism. Signing uses the deterministic variant of Dilithium with no random nonce. This eliminates the class of vulnerabilities arising from weak-random-source signing.

12.3 ML-KEM (Module-Lattice KEM)

ML-KEM-768 is used wherever a key encapsulation is needed:

- **Peer-to-peer Noise handshakes.** Operator-to-operator connections use ML-KEM-768 for the Noise XX handshake, replacing the classical X25519 default.
- **RPC TLS.** Public RPC endpoints serve TLS certificates with ML-KEM-encapsulated session keys. Wallet, indexer, and explorer connections to nodes use post-quantum TLS.
- **Stealth addresses.** The privacy denomination's stealth-address scheme uses ML-KEM end-to-end for one-time-address derivation.
- **Mempool encapsulation.** The encrypted mempool wraps each transaction body in an ML-KEM-768 encapsulation under a per-epoch threshold-DKG public key.

12.4 Two-tier finality

Consensus delivers **two levels** of finality, calibrated for different use cases.

Anchor-level finality (BLS12-381 aggregate). Each operator in a 7-of-10 cluster threshold-signs the cluster's vertex with their BLS share; the cluster aggregate compresses 700 individual operator signatures per round into a single verifier check. Anchor-level finality settles in three to five seconds (one wave). Use cases: everyday user transfers, application interactions, mempool admission.

Quantum-attested finality (ML-DSA-65 checkpoint). Every hundred anchors, operators each sign the canonical state root with their ML-DSA-65 keys. The result is a **post-quantum checkpoint** that bridges, exchange listings, and high-value cross-chain attestations bind to. Use cases: bridge unlocks, exchange deposits, large cross-chain transfers.

The checkpoint cadence is **milestone-overridable**, so high-value bridge integrations can drop the interval (for example to ~24 seconds) for tighter cross-chain UX without a hard fork, or raise it for bandwidth conservation.

The two tiers compose: anchor-level finality is the fast everyday signal; quantum-attested finality is the deep settlement signal that resists quantum forgery. A quantum attacker who can forge BLS aggregates cannot also forge ML-DSA-65 — checkpoints require honest clusters' post-quantum signatures, so an attacker fork dies at the next checkpoint regardless of how many BLS aggregates they fabricate between checkpoints.

The bandwidth cost is negligible. At 100 operators per checkpoint, ~330 KB every five minutes is roughly one kilobyte per second on average; even at the tight 24-second override, the cost remains well within the chain's normal traffic budget.

12.5 Threshold-DKE encrypted mempool

The encrypted-mempool admission rule is binding from genesis: every transaction enters the mempool encrypted; the body becomes plaintext only at anchor inclusion. Decryption is a Ferveo threshold-DKE ceremony over BLS12-381.

Each cluster operates a per-epoch threshold-decryption ceremony: operators run distributed key generation to produce a shared decryption key, and each operator holds one share. A threshold (7-of-10 per cluster) is required to decrypt any individual mempool ciphertext. No single operator can decrypt; no minority can. The mempool's privacy property holds against everyone except a 7-of-10 cluster collusion.

At anchor inclusion, the cluster's threshold-decryption coordinator collects partial decryptions, combines them via Lagrange interpolation, and emits the plaintext bodies into the executor. Every step is deterministic and auditable. Confidentiality is lifecycle-bounded — transactions become plaintext at inclusion, seconds after entering the mempool — so the harvest-now-decrypt-later threat that classical encryption faces does not apply.

12.6 Zero-knowledge proof systems

Application-layer zero-knowledge verification uses an SP1 zkVM with an on-chain Groth16-BN254 SNARK verifier, and is not consumed by the consensus path. A future migration to a hash-only, fully post-quantum FRI/STARK verifier is a long-horizon goal, pending the proving ecosystem shipping on-chain-verifiable STARK receipts; until then, post-quantum integrity at the deep-settlement tier comes from the ML-DSA-65 quantum-attested checkpoints (§12.4), which an attacker who forges a SNARK proof still cannot bypass. The chain uses zero knowledge where it reduces risk the most:

- **Bridge proofs.** A bridge can attest that an external chain finalized a specific event, state transition, burn, lock, or withdrawal condition. The chain verifies the proof before releasing assets or updating bridge state. This reduces dependence on trusted multisigs and relayer committees.
- **Swap proofs.** A swap can verify that a batch of intents or orders was matched according to a declared policy. This supports fair ordering, batch auctions, and verified-matching markets without asking users to trust an opaque sequencer.
- **zkML attestations.** A model can produce a verifiable attestation that an output was generated by a specific signed model on specific inputs.
- **High-value off-chain computation.** Computations that would be prohibitive to run on-chain can be proven off-chain and verified on-chain in bounded gas.

The GPU prover service tier — described in §17 — provides paid proof-generation capacity, while on-chain verification runs on commodity-CPU clusters. Generation is expensive and concentrated on operators with the right hardware; verification is cheap and distributed across the cluster set. The architecture decentralises the prover infrastructure without centralising the verifier.

12.7 Hash functions

BLAKE3 is the chain's primary hash for state-tree commitments, content-addressed proofs, and public address derivation. Grover's quadratic speedup against 256-bit output yields 128-bit quantum security, well above any quantum threat horizon. Where legacy compatibility codecs require Keccak (such as event-log selectors), that use is explicitly scoped to compatibility, not to address derivation or state commitments.

13. Identity: Addresses, Mnemonics, and Names

13.1 Account addresses

Monolythium addresses are 20-byte payloads derived from a domain-separated BLAKE3 hash of the user's public key:

```
address[20] = BLAKE3(
    "MONO_ADDRESS_BLAKE3_20_V1"
    || algo_id_be_u16
    || canonical_pubkey_bytes
)[0..20]
```

The wallet publishes the user's public key on chain at account creation, providing post-quantum forgery resistance from the moment the account exists, before any funds arrive. Until publication, the account is identified only by its address; after publication, every transaction is verified against the stored public key.

13.2 Bech32m display

User-facing addresses are displayed in **bech32m** format only. Hex `0x...` display is not the canonical surface; bech32m is the only address format wallets, the chain's block explorer, partner integrations, and SDKs render by default.

The bech32m checksum and the `mono` human-readable prefix make Monolythium addresses visually unmistakable from any other chain's `0x...` format, providing format-as-safety against cross-chain address confusion at exchange listings, bridge UIs, and partner integrations.

Addresses use a per-type human-readable prefix discriminator so that an address's role is visible at a glance:

PREFIX	ADDRESS CLASS
`mono1...`	Standard user account
`monos1...`	Stake / delegation account
`monoc1...`	Contract account
`monok1...`	Cluster account
`monom1...`	Native module account
`monox1...`	Bridge account

Additional prefixes are reserved for future account classes. A user reading the discriminator can tell whether they are about to send to a normal account, a contract, a cluster, or a bridge — a property that hex addresses cannot offer.

13.3 PQM-1 mnemonic — post-quantum wallet backup

Wallets support two recovery formats:

1. **Keystore (default).** Encrypted private-key file, password-protected (Argon2id key derivation over the password, then XChaCha20-Poly1305 over the seed). This is the standard post-quantum tooling pattern and is the default for non-power users.
2. **PQM-1 mnemonic (opt-in).** A 24-word BIP-39 phrase with a versioned, algorithm-tagged seed layout. The same wordlist BIP-39 uses, so existing hardware wallets that already display 24 words can still display and store the phrase — but the bytes those 24 words encode are interpreted under PQM-1, not under BIP-32.

The PQM-1 seed layout (32 bytes, most-significant-byte first):

OFFSET	LENGTH	FIELD
0	1 byte	Algorithm tag (`0x01` = ML-DSA-65; `0x02` – `0x04` reserved for future algorithms; `0xF0` – `0xFF` reserved for vendor extensions)
1	1 byte	Version (`0x01` for PQM-1 v1)
2	30 bytes	Entropy (240 bits — well above the 128-bit post-quantum security target)

Derivation:

```
words           = BIP-39 24-word phrase (English wordlist)
payload[32]     = bip39_decode_to_entropy(words)
algo            = payload[0]
version         = payload[1]
entropy[30]     = payload[2..32]
seed[32]        = SHAKE256(
                    "monolythium.pqm1.v1.mldsa65" || payload,
                    32
                  )
(pk, sk)        = ML_DSA_65_KEYGEN(seed)
address[20]     = BLAKE3("MONO_ADDRESS_BLAKE3_20_V1"
                         || algo_id_be_u16
                         || canonical_pubkey_bytes(pk))[0..20]
display         = bech32m("mono", address)
```

The domain-separated SHAKE256 expansion (with the literal string `"monolythium.pqm1.v1.mldsa65"`) ensures the seed handed to ML-DSA-65 keygen cannot collide with seeds derived for any other algorithm tag, version, or chain that adopts the PQM-1 scheme. Two conforming wallets given the same 24 words produce identical public key, identical address.

PQM-1 v1 encodes a **single account** per mnemonic. Hierarchical-deterministic multi-account derivation is intentionally not part of v1 — it bakes in path semantics that are awkward for ML-DSA-65 (large keys, no native hardening primitive matching the BIP-32 HMAC-SHA512 construction). Power users wanting multiple accounts hold multiple mnemonics or use the keystore format with a wallet that manages many keystores.

The mnemonic is **self-describing**: the algorithm tag and version are the first two bytes of the payload, so a wallet decoding an unknown mnemonic can tell the user "this is an ML-DSA-65 v1 PQM-1 mnemonic" before doing keygen. A wallet seeing a non-`0x01` algorithm tag — or a version it does not recognise — displays an explicit warning rather than silently deriving the wrong address. Wallets that conform to PQM-1 are required to display this warning.

A typical BIP-39 → secp256k1 mnemonic (the format used by most EVM wallets) **is not compatible**. Even though both schemes use 24 BIP-39 words, the byte-level interpretation differs (BIP-39 entropy → BIP-32 → secp256k1 vs. PQM-1 entropy → SHAKE256 → ML-DSA-65 keygen). Importing an EVM seed into a Mono wallet derives an entirely different address controlling no funds; importing a PQM-1 seed into an EVM wallet likewise derives a key that signs nothing on Mono. Wallets warn the user before either direction is attempted.

13.4 The LYTH name registry

Bech32m addresses are unmistakable but not memorable. Monolythium ships a hierarchical, on-chain **name registry** that maps human-readable names to addresses.

Naming structure:

- **Charset.** Lowercase ASCII letters, digits, and the hyphen. No mixed case, no Unicode confusables, no scripted characters at the protocol level.
- **Hierarchy.** Names live under the `.mono` namespace and resolve to one of five structural categories. A bare `<name>.mono` is a **human** (personal) account. An agent is registered **beneath its human principal** as `<name>.agent.<human>.mono`, so the naming hierarchy mirrors the agent-as-sub-account model directly. Clusters, deployed contracts, and protocol/system entities take `<name>.cluster.mono`, `<name>.contract.mono`, and `<name>.system.mono` respectively; the `system` category is foundation-only and closed to public registration.
- **Pricing.** A U-curve pricing schedule discourages both squatting (short, premium names cost meaningfully more) and clutter (very long names cost a small

administrative fee). The pricing curve is constitutional; it is not adjustable through any signaling mechanism.

- **Transfer.** Names transfer through a propose-accept flow with a 24-hour acceptance window. The recipient must affirmatively accept the transfer; the sender cannot push a name to an unwilling recipient.
- **Reserved prefixes.** The registry refuses to register names that begin with the chain's bech32m human-readable prefixes (`mono`, `monos`, `monoc`, `monok`, `monom`, `monox`, and additional reserved prefixes), or with `0x`. This prevents the registry from minting names that could be confused with raw addresses.
- **Cascade.** When an agent identity is transferred to a new principal, any names registered against the agent are cascaded according to the transfer rules. Names cannot become orphaned silently.

The Monolythium Foundation seeds the registry at genesis with a reserve of category-defining and protocol-relevant names. The reserve is published, not sold; reserved names are released into the open market on a schedule that is also published.

The result is that a person is addressed as `alex-rivera.mono`, an agent owned by that person as `support-bot.agent.alex-rivera.mono`, a cluster as `north-star.cluster.mono`, a deployed contract as `monoswap.contract.mono`, and a protocol entity as `name-registry.system.mono`. The bech32m address is still the canonical underlying identifier; the name is a human-readable alias that resolves to it.

---

14. Execution: Rust on RISC-V

The execution environment is a **Rust/RISC-V-native smart contract runtime**. The chain does not target Solidity or EVM bytecode for mainnet. The execution layer has three tiers:

1. **Native protocol modules** for hot and security-critical primitives — token balances, NFTs, multi-assets, spot markets, delegation, bridge proof verification, agent-commerce registries, spending policy, name registry, privacy cordons, and emergency recovery.
2. **Rust/RISC-V contracts** for application-specific programmable logic.
3. **zkVM-proven computation** for bridge proofs, cross-chain swaps, zkML, and high-cost off-chain verification.

The goal is not “EVM, but faster.” The goal is a smaller, safer programmable settlement layer with first-class support for post-quantum accounts, AI-agent workflows, and verifiable cross-chain settlement.

14.1 Contract artifact

The contract artifact is a deterministic Monolythium RISC-V package containing:

- code section compiled to the approved RISC-V profile;
- contract ABI manifest;
- import table declaring required host syscalls;
- declared memory limits;
- declared storage namespace;
- code hash and build metadata;
- optional source map and debug metadata, excluded from consensus hashing.

The preferred source language is Rust. The contract SDK provides macros, types, storage collections, ABI generation, event helpers, test harnesses, and deployment packaging.

Contract execution is fully deterministic. There is no wall clock, no filesystem, no sockets, no host randomness, no floating-point. Every observable input arrives through transaction calldata, contract storage, anchor context exposed by syscall, or explicit native-module query.

14.2 Host syscall ABI

The RISC-V VM is intentionally small. Chain access happens through a narrow host syscall ABI:

```
storage_read(key) -> bytes?
storage_write(key, value)
storage_delete(key)
caller() -> Address
contract_address() -> Address
anchor_height() -> u64
anchor_hash(height) -> Hash
call_contract(address, input, value) -> bytes
emit_event(topic, data)
transfer_native(to, amount)
verify_signature(algo, pubkey, msg, sig) -> bool
hash(domain, bytes) -> Hash
revert(code, data)
```

Syscalls are metered separately from RISC-V instruction cycles. Storage reads and writes are priced against state I/O, not CPU. Cryptographic syscalls are priced against measured native implementation cost. Cross-contract calls create a bounded call frame with explicit gas/cycle budget, memory limit, value transfer rules, and revert propagation.

The syscall ABI is a **consensus contract**. Once a syscall reaches stable status, changing its semantics requires a coordinated upgrade.

14.3 Gas, cycles, and state

The cost model separates three resources:

RESOURCE	WHAT IT MEASURES
RISC-V cycles	Deterministic compute performed by a contract
Host syscall cost	Native functions such as hashing, signature checks, transfers, and module calls
State I/O	Reads, writes, deletes, storage growth, events, and receipts

This matters because **state is the real bottleneck** in a blockchain. Arithmetic is cheap. Long-term storage, database reads, and permanent state growth are expensive. Monolythium prices state intentionally so normal usage stays affordable while state bloat remains controlled.

The practical result:

- simple transfers are cheap;
- token and NFT operations are predictable;
- compute-heavy contracts benefit from efficient RISC-V execution;
- storage-heavy applications pay for the burden they place on the network;

- wallets can estimate fees more accurately because execution is more predictable than under EVM.

---

15. Native Modules and MRC Standards

Monolythium does not remove tokens, NFTs, or markets. It makes them **native**.

Instead of asking every application to rebuild core financial primitives as arbitrary smart contracts, the chain provides audited native modules and Mono-native standards.

ETHEREUM CONVENTION	MONOLYTHIUM TARGET
ERC-20	**MRC-20** fungible tokens
ERC-721	**MRC-721** non-fungible tokens
ERC-1155	**MRC-1155** multi-assets
ERC-4626	**MRC-4626** vaults
ERC-1271 / ERC-4337 patterns	Native smart-account and policy-account standards
Solidity marketplace contracts	Native marketplace modules and Rust/RISC-V contracts
Contract-based AMMs and order books	Native spot-market modules with contract hooks

The high-volume paths are native:

- token transfers;
- NFT ownership and transfer;
- multi-asset and game-item batch transfers;
- spot-market order placement and settlement;
- bridge proof verification;
- agent spending-policy checks;
- account and permission management.

Rust/RISC-V contracts remain available for application-specific logic: custom marketplaces, vault strategies, agent workflows, escrow templates, games, domain-specific settlement flows, and advanced business logic.

The design reduces duplication and audit burden. A common asset standard should not be reimplemented thousands of times across unrelated applications. Wallets and explorers display MRC assets as **first-class assets**, not as arbitrary contract calls — when a user receives an MRC-721, the wallet knows it is an NFT with a typed schema, not an opaque series of bytes that might or might not implement an interface correctly.

Markets and NFTs

The native order book is universal market infrastructure: compute, data, agent services, real-world assets, stablecoin pairs, and token pairs all settle through the same primitive. Perpetuals and margin remain explicitly removed (§4.2). Automated market makers can be implemented either as native modules or Rust/RISC-V contracts if the risk surface is accepted by the application.

MRC-721 and MRC-1155 provide native ownership, metadata, transfer, approval, and marketplace event shapes. Marketplace settlement can be native for the common case and contract-based for custom auction or game logic.

Why native does not mean larger attack surface

A common concern with native-module-heavy chains is that pushing primitives into consensus code grows the protocol's attack surface compared to a "thin L1 + many contracts" design. The opposite is true in practice for the workloads Monolythium serves.

A thin-L1 design spreads the financial primitives across thousands of independently authored contracts, each of which can be wrong in its own way. A token standard reimplemented incorrectly in a single application contract has produced losses many times in EVM history. The chain's design pulls the heaviest financial logic — token accounting, NFT ownership, order-book settlement, bridge proof verification, spending-policy enforcement — into a small set of native modules audited at the protocol layer. Applications compose against those modules. The audit work happens once. The bug surface in user code shrinks because user code does not implement what the native module already provides.

Native modules add to the consensus-critical line count. They subtract from the ecosystem-wide attack surface by far more. The chain pays the larger native module footprint to gain a much smaller composite surface across the applications that run on it.

16. Tokenomics

16.1 LYTH supply

The native token is **LYTH**.

- **Initial supply:** 100,000,000 LYTH at genesis.
- **Annual inflation cap:** 8% — a protocol-layer hard cap that applies to all cluster member and role rewards from the same allocation. The cap cannot be lifted by any signaling mechanism; lifting it requires a coordinated hard fork.
- **Treasury allocation:** pre-funded from the genesis reserve. There is no inflation tap for the treasury. The treasury draws on the genesis reserve and on buyback-burn yield over time.

The supply structure is constitutional: the inflation cap, the treasury's funding mechanism, and the basic distribution rules are not changeable through any in-protocol process. They are part of the chain's identity, not parameters open to runtime adjustment.

16.2 Allocation framework

The 100,000,000 LYTH initial supply is allocated across seven categories at genesis. Each category has its own release schedule and governance constraint; the supply itself is fixed at genesis.

#	CATEGORY	LYTH	SHARE
1	Community obligations (pre-allocated genesis balances)	32,781,503.90	32.78%
2	Foundation treasury reserve	15,000,000	15.00%
3	Ecosystem & grants	15,000,000	15.00%
4	Core contributors (vested)	13,000,000	13.00%
5	Public sale & community access programs	12,000,000	12.00%
6	Operator incentives & community programs	7,218,496.10	7.22%
7	Liquidity provision & integration support	5,000,000	5.00%
	Total	**100,000,000.00**	**100.00%**

Category 1 — community obligations. A frozen ledger of 732 historical community-cohort entries, totaling 32,781,503.90 LYTH, is pre-allocated as on-chain genesis balances. No claim flow is required; each entry resolves to a direct balance on the holder's chain address. The ledger is canonical and immutable.

Category 2 — Foundation treasury reserve. Held in a multi-signature Foundation treasury with published signers and a published rationale requirement for every disbursement. The treasury does not draw on the 8% annual inflation; it operates from this category and from buyback-burn yield. Release is governed by treasury rules published by the Foundation.

Category 3 — ecosystem & grants. Reserved for developer grants, integration support, market-maker partnerships, infrastructure subsidies during the operator-bootstrap phase, and ecosystem programs that grow the chain's adoption surface. Released against published criteria over a multi-year horizon.

Category 4 — core contributors. Held in vested allocations to the engineering, design, research, and operations contributors who built the chain. Standard vesting includes a 12-month cliff and a 48-month linear vest. The published team and contributor list is the canonical record.

Category 5 — public sale & community access programs. The Genesis Liquidity Program and successor community-access offerings are funded from this category. Sale prices, vesting schedules, and KYC requirements are published at the time of each offering on the canonical project surfaces. Proceeds from these offerings, denominated in stablecoins and fiat, fund engineering payroll, infrastructure, market-maker arrangements, and external security audits across the consensus, signing, bridge, escrow, agent-commerce, and private-denomination surfaces.

Category 6 — operator incentives & community programs. Operator onboarding rewards, cluster-bootstrap incentives, community airdrops to historical ecosystem participants, and similar programs draw from this category. Programs are published at the time of activation.

Category 7 — liquidity provision & integration support. Reserved for bridge-route bootstrapping, market-maker arrangements on the native order book, and partnerships that deepen the chain's liquidity edge. Disbursements are public and reasoned.

16.3 Token utility and value capture

LYTH accrues utility across the entire chain surface, not only at the gas layer.

Consensus economics.

- **Operator self-bond.** Each active operator position requires 5,000 LYTH self-bond. At the target cluster topology of 700 active positions, this locks 3,500,000 LYTH as consensus-critical collateral. Equivocation slashing burns the entire bond; partial slashing for non-equivocation faults reduces it. The bond is the operator's economic skin in the game.
- **Standby capacity.** Each cluster's three standby operators also bond at the same level; at full capacity 300 standby positions × 5,000 LYTH = an additional 1,500,000 LYTH bonded.

Network economics.

- **Gas.** Every Monolythium transaction pays gas in LYTH — including transactions that move stablecoins, MRC assets, or NFTs. Stablecoin volume settled through the chain still requires LYTH to authorize the underlying transaction.
- **Fee burn.** A portion of every transaction's gas is burned, creating sustained deflationary pressure proportional to network activity.
- **Storage and state-growth fees.** State-heavy operations (large storage writes, persistent data) pay against state I/O metering, denominated in LYTH.

Service-tier economics.

- **Cluster service-tier payments.** RPC calls, archival reads, GPU prover requests, and oracle feed consumption are denominated in LYTH and paid directly to the serving operator. The on-chain prover market is a native LYTH-denominated venue — proof generation is purchased through a chain-native market rather than through an off-chain vendor arrangement.
- **Bridge route fees.** Each bridge route charges a fee in LYTH on top of the asset-denominated bridge value. Bridge insurance and reserve programs also pay in LYTH.

Registry economics.

- **Discovery-registry bonds.** Every service provider listing the discovery registry stakes a LYTH bond, sized to deter spam and returned on listing closure.
- **Issuer-registry bonds.** Credential issuers register with a LYTH bond for the same purpose.
- **Arbiter-registry bonds.** Arbiters stake LYTH to register and forfeit a slashing amount on bad-faith arbitration.
- **Name-registry fees.** Each LYTH name registration pays a category-dependent fee under the U-curve pricing schedule. Renewal fees are recurring.

Application economics.

- **Native order-book fees.** Maker and taker fees on the native CLOB pay in LYTH or in the asset being traded (with a LYTH-denominated discount path).
- **Spending-policy registration.** Registering or modifying a programmable spending policy on an agent sub-account pays a small LYTH fee that funds the protocol-layer enforcement work.
- **Escrow & dispute fees.** Opening an escrow, escalating to dispute, and arbiter compensation all settle in LYTH.

Delegation economics.

- **Delegation tracking.** Delegators do not lock LYTH (liquid bonding), but their delegated balance is tracked against the per-wallet cap and earns reward distributions proportional to the cluster's performance.

- **Reward distribution.** The 8% inflation cap funds operator and delegator rewards. Service-tier revenue is direct to the providing operator; consensus rewards distribute through the cluster pool.

The composite effect is that LYTH is the economic primitive of the chain, regardless of what asset settles in the body of any given transaction. Agents paying USDC for an API call still pay LYTH gas; bridges moving Ethereum-resident value still pay LYTH route fees; clusters serving traffic still earn LYTH. The token captures the chain's economic activity at every layer above the payment-asset itself.

16.4 Canonical LYTH

The only canonical LYTH is the native token on Monolythium L1. Tokens with the same name on other chains — Binance Smart Chain, Ethereum, Solana, or any other — are not affiliated with the project unless explicitly listed on the canonical-tokens page published by the project.

Wallets, exchanges, and integrators should verify any cross-chain LYTH representation against the canonical-tokens page before treating it as official. Unaffiliated tokens carrying the LYTH name are not redeemable for native LYTH and do not carry the protocol-layer utility described above.

Bridged or wrapped LYTH on other networks, if and when such routes are established, will be listed on the canonical-tokens page with the bridge route, trust model, and verification status visible to users. Anything not listed there is not canonical.

16.5 Liquid bonding

Monolythium's staking model is **liquid bonding**: stakers retain custody of their LYTH at all times.

- Stakers never send their LYTH to a staking contract. Tokens remain in the user's wallet. The protocol tracks vote weight without locking the assets.
- There is **zero unbonding period** for delegators. A delegator can exit a delegation instantly. There is no slashing-window cooldown for delegators; slashing applies to operator self-bonds only.
- A user may multi-stake across up to ten clusters with basis-point granularity, subject to the per-cluster delegation cap below.

Liquid bonding gives delegators full flexibility while preserving the security model: operator self-bonds carry the slashing risk and are the principal capital at stake for protocol misbehaviour.

16.6 Operator self-bond and slashing

While delegators remain liquid, cluster operators guarantee network integrity with hard collateral:

- **5,000 LYTH self-bond** at onboarding into an active cluster position.
- **100% slash plus permanent operator exile on equivocation** (signing two distinct vertices at the same round). Recovery by re-bonding is not permitted; the operator identity is burned.
- **Reduced slashing for non-equivocation faults** (extended downtime, malformed-vertex broadcasting, cluster non-cooperation): partial bond burn plus temporary suspension.

The exemplary nature of equivocation slashing is deliberate. A protocol that tolerates double-signing invites it; a protocol that destroys the operator's stake and exiles them forever does not need to tolerate it.

16.7 Distributed delegation protocol

The core anti-capture mechanism is the **per-wallet delegation cap**: any single wallet can delegate at most X% of its total LYTH holdings to any single cluster. The cap binds on capital, not on wallet count.

The cap tightens progressively as the network matures:

NETWORK CONDITION	PER-CLUSTER CAP	MINIMUM DIVERSIFICATION
Early operating set	50%	2 clusters
Growing operating set	25%	4 clusters
Mature operating set	15%	7 clusters
Steady-state operating set	10%	10 clusters

Tightening is triggered by sustained network conditions — number of independent active clusters, geographic distribution, operator entropy — not by calendar dates and not by any in-protocol vote. Modifying the schedule requires a coordinated hard fork.

Wallet-multiplication resistance. The cap binds on capital per wallet, not on wallet count. An adversary splitting their capital across many wallets cannot route more LYTH to a single target cluster than they could from a single wallet — the per-wallet cap percentage applied to the smaller per-wallet balance produces the same nominal amount in aggregate. This eliminates the wallet-multiplication advantage that has historically broken delegated-stake systems.

The cap does not, and cannot, eliminate the underlying constraint that a sufficiently wealthy actor can route their capped percentage to any single cluster of their choice. That constraint is a function of capital, not of wallet count, and is unavoidable for any per-wallet cap. Three structural defenses combine to bound the practical concentration outcome:

- the per-wallet cap (this section),
- the quadratic reward curve (§16.8),
- the per-operator multi-cluster cap (§6 and §17).

The combined effect is that even an actor with substantial capital cannot replicate single-validator dominance: their per-wallet cap limits their delegation to any one cluster, the quadratic reward curve makes oversaturating that cluster unprofitable for them, and the per-operator cap prevents them from operating the cluster across many of its positions.

Stake division, not multiplication. A delegator with one hundred LYTH diversifying across ten clusters at the 10% cap delegates ten LYTH per cluster — total credited weight one hundred LYTH, not one thousand. Total credited weight is conserved. There is no economic gradient pushing voters toward cartel slates over independent diversification — cartel formation has cost but no incremental yield.

16.8 Reward distribution

The cluster reward pool, sized within the 8% annual inflation cap, distributes to clusters using a **quadratic curve** that introduces diminishing returns as a cluster's delegated stake grows. Doubling a cluster's stake does not double its reward share. Combined with the per-wallet cap, this creates a layered economic disincentive against cluster concentration:

- the per-wallet cap is the hard ceiling at the protocol layer;
- the quadratic reward curve is the soft economic gradient at the reward layer.

Even within the cap, oversaturated clusters yield diminishing per-stake returns, so honest delegators are nudged toward under-served clusters by their own self-interest. **Decentralization is the highest-yield strategy at every margin, not just at the boundary.**

The curve shape is a design commitment of the protocol; calibration parameters are adjustable only through hard-fork protocol updates.

Within a cluster, the consensus reward pool distributes equally among its operators. Service-tier revenue (RPC, archive, prover, oracle) is per-operator direct rather than pool-split; an operator with a strong service offering captures the direct revenue from that service.

16.9 Auto-rebalance on cap tightening

When a tightening transition activates and the cap moves (for example from 25% to 15%), existing over-cap delegations are not slashed or instantly invalidated. The protocol enters a **grace period**:

- over-cap delegators receive standard rewards on the in-cap portion;
- over-cap delegators receive a tapered share on the over-cap excess, sized to give honest delegators sufficient time to rebalance without meaningful yield loss;
- the wallet UX (autovote routines, see §25) is notified to rebalance;
- after the grace period ends, the over-cap excess earns no rewards until the delegator reduces per-cluster allocation or redirects the excess to additional clusters.

The transition is non-disruptive: delegators have a clear economic signal and a defined window to adjust; clusters have continuity of voting weight during the rebalance.

16.10 Acknowledged limitations

Two limitations are documented honestly.

Custodian aggregation. A large custodian holds customer assets in a pooled omnibus wallet. The single wallet is subject to the cap, potentially under-representing the diverse preferences of underlying users. The cap still binds on the custodian's pooled wallet itself — even a major exchange under-representing many users cannot dominate any single cluster beyond the cap. The protocol does not solve sub-account expression of preference, but it prevents custodian-level cluster capture. Sub-account delegation standards and integration with proof-of-personhood systems are areas of ongoing development.

Correlated preferences. The protocol does not, and should not, prevent legitimate market behaviour. If a large majority of independent, sovereign delegators happen to choose the same high-performing cluster, that cluster legitimately earns a large share of delegated stake. This is a reflection of the market's will, not a Sybil attack.

17. Cluster Operations: DVT, Slashing, Service Tiers

17.1 Distributed validator technology

A cluster is a **distributed validator** in the strict sense: ten independent operators run a coordinated key-share ceremony, produce a shared signing key, and threshold-sign the cluster's vertex with a 7-of-10 threshold. The cluster signs one vertex per round. Up to three operators can be offline without halting the cluster's participation.

Distributed validator technology gives the chain three properties single-operator validators cannot:

- **Operator-level fault tolerance.** A single operator's outage does not stop the cluster.
- **Operator diversity.** Operators within a cluster can be in different regions, on different network providers, on different hardware classes, and under different jurisdictions.
- **Operator markets.** Operator capacity is a public market — operators publish capabilities, clusters compose themselves from the market, and reputation accrues to the operator identity over time.

17.2 Standby capacity

Each cluster has a **standby of three operators** in addition to its seven active members. Standbys participate in the cluster's communications, hold the threshold key-shares ceremony's transcript, and are ready to step in when an active operator goes offline, is rotated out, or departs.

This is what makes the cluster operationally — not just structurally — resilient. An active operator can be replaced without a re-keying ceremony; the next-eligible standby is promoted, the threshold continues to hold, and the cluster keeps producing.

17.3 Service tiers

Beyond consensus participation, operators earn LYTH from service tiers paid by users:

- **GPU proving.** Off-chain proof generation for zkML attestations and zero-knowledge bridge proofs runs on GPU-equipped operators; on-chain verification runs on commodity-CPU operators. The cluster GPU service is paid through the on-chain prover market — a native, on-chain-paid GPU proof market that lets clusters monetize proof generation directly through the protocol rather than through an off-chain vendor arrangement.
- **RPC and archival.** Clusters serve user traffic — wallet RPC calls, indexer queries, archival reads — and earn LYTH from the requestor.

- **Oracle feeds.** Clusters with reliable infrastructure participate in the oracle aggregation network and earn LYTH from oracle consumers.

Service-tier revenue is **direct to the operator** providing the service, not split through the cluster reward pool. A cluster that wants a strong GPU prover service offers operators with GPUs a meaningful direct revenue stream in addition to their share of the cluster's consensus rewards.

17.4 The GPU service tier separation

A cluster's GPU prover capacity is a **service tier**, not a consensus dependency. A cluster's consensus participation depends on its 7-of-10 BLS threshold over CPU-bound operators; GPU is required only for off-chain proof generation. A GPU operator going offline costs the cluster its prover service revenue but **does not affect the cluster's consensus participation or the chain's Byzantine fault threshold**.

The GPU-as-service-tier choice is deliberate. Enforcing GPU on every operator would create hardware-cost moats that smaller operators cannot cross, producing operator monopolies and hardware-class centralisation. The chain's design lets operators specialize: clusters that want strong GPU service hire GPU-equipped operators; clusters that don't earn the consensus base reward and avoid the GPU capital cost. The market prices the difference; the consensus layer remains commodity-CPU and accessible.

The asymmetric cost economics make this work. Proof generation on a modern GPU runs in tens of seconds; on-chain verification by commodity CPU operators runs in a small, predictable gas budget. Generation is expensive and concentrated; verification is cheap and distributed.

18. Agent Commerce Primitives

Monolythium ships **eight agent-commerce primitives** that together enable the autonomous-economy use case. The primitives are minimal, composable, and consensus-critical only where consensus protection is necessary; gameable primitives are deliberately placed at the application or indexer layer.

#	PRIMITIVE	LAYER	PURPOSE
1	`attestation`	Native module	Foundational signed-hash + typed schema registry
2	`consent`	Native module	Principal-signed consent records with revocation
3	`issuer-registry`	Native module	Permissionless registry of credential issuers
4	`discovery`	Native module	Permissionless service-listing registry
5	`reputation`	Indexer view	Multi-dimensional rating aggregation over chain-emitted events
6	`availability`	Native module	Generic availability state composed by every domain registry
7	`escrow + arbiter-registry`	Native modules	Configurable escrow with counter-offer flow + pluggable arbiter registry
8	`spending-policy`	Native module (consensus-critical)	Programmable per-account spending caps enforced at admission

None consume the consensus path except `spending-policy`, which is consensus-critical because the protocol must enforce budget caps — the agent itself cannot be trusted to enforce its own caps; the agent is precisely what the policy is constraining. Reputation lives at the indexer layer because the gaming surface (Sybil rating, wash trades, coordinated reputation farming) does not belong in consensus-critical code.

18.1 Attestation

The foundational primitive. A `subject` is attested by a `signer` against a `typed schema`; the resulting attestation is content-addressable, revocable by the signer, and verifiable offline against the signer's public key. New schemas register at runtime — no hard fork is required to add a new attestation type.

Use cases:

- capability claims (“this entity is qualified to do X”);
- consent attestations (principal signs the terms);
- reputation events (counterparty rates an interaction);
- service deliverables (provider attests to completion).

18.2 Consent

The chain’s consent registry enables principal-signed records with controlled revocation.

- `grant(scope, terms, expiry?) -> ConsentId`
- `revoke(id)` — instant for new uses; existing in-flight escrows initiated under the consent are grandfathered until completion.
- `query(scope) -> Vec<Consent>` — read all currently-active consents for a scope.

Revocation is instant for new activity; in-flight workflows are grandfathered until completion. This balances the principal’s right to instant revocation with the counter-party’s right not to be left holding a half-completed work product because the principal changed their mind mid-stream.

18.3 Issuer registry

The chain’s authoritative map from “issuer name + jurisdiction” to “public key + metadata” for credential-issuing authorities. Permissionless to register, with a minimal anti-spam stake bond. The chain does not pick winners; markets weight which issuers they trust.

Use case: when a holder presents an attestation signed by an issuer (a state bar, a medical board, a credentialing program), the chain knows which public key corresponds to that issuer’s name. A verifier offline-verifies the attestation against the registered key without trusting any single party’s claim of who the issuer is.

The protocol has no mechanism to remove an issuer from the registry once registered. If an issuer turns out to be fraudulent, the market response — verifiers stop trusting the issuer’s signatures, attestations under that issuer’s key lose value — is the only mechanism. Attempting to add a removal mechanism would be governance-shaped and is excluded by the chain’s no-governance design.

18.4 Discovery registry

The chain-native, permissionless registry of service providers. A provider stakes a small LYTH bond, registers a typed listing (`provider_address, capability_set, fee_structure, availability_handle, metadata_uri`), and becomes searchable on-chain. The chain emits indexable events; off-chain indexers build the user-facing search experience.

Listings are organized under a hybrid taxonomy:

- **fixed top-level categories** for filtering (legal, dev, design, finance, consulting, marketing, social, operations, and others);
- **free-text description fields** per listing;
- **embeddings** for fuzzy and natural-language search, computed off-chain by indexers from listing text; the on-chain storage is the listing itself.

Marketplace scope is universal. The same discovery primitive serves lawyers, developers, designers, finance professionals, AI agents, paid one-off consultations, batch promotional services, and any other category that fits a typed listing schema. Same registry, same primitive, different category.

The stake bond is sized to deter spam, not to gate participation; it is held for the lifetime of the listing and returned on closure. Frivolous spam listings cost LYTH; legitimate providers pay a one-time entry cost.

18.5 Reputation

Reputation is **derived from on-chain history** — there is no separate "reputation token" or "rating contract." The reputation views aggregate over the same on-chain data that backs every other primitive.

The chain emits typed events for every reputation-relevant interaction: escrow released, deliverable attested, dispute resolved, counter-offer countered. Off-chain indexers aggregate the events for fast queries; raw data is provable from the chain.

Reputation is **multi-dimensional** — four fixed axes: speed, quality, communication, accuracy. Each axis is rated 1-5 by counterparties; aggregations are public. The four axes give downstream search clients enough signal to filter on dimension ("fast turnaround" vs. "high quality" vs. "good communication") without forcing a one-dimensional collapse.

Reputation is **portable**. An agent's reputation address is the agent's chain address. Reputation accrues to the agent address. If the agent switches model provider, the address stays the same; the reputation persists. This is the structural-portability property that closed payment systems cannot offer.

18.6 Availability

A generic availability state machine composed by every domain registry — arbiters, providers, inference services, all of them. One source of truth for:

- **open request count** (how many requests are currently outstanding for the entity);
- **accept/deny per request with timeout** (a request is auto-declined if not accepted within a configurable timeout);
- **vacation/availability flag** ("not taking requests until date X").

The same state machine serves an arbiter (tracking how many disputes they have queued), a provider (tracking how many open offers they have outstanding), and an inference service (tracking how many concurrent jobs are being executed). One implementation, one audit, many consumers.

18.7 Escrow and arbiter

The workhorse of agent commerce. Two on-chain modules — `escrow` and `arbiter-registry` — that compose into the full counter-offer + dispute-resolution flow.

The arbiter mode is set per escrow at creation:

- **Single arbiter** — one designated arbiter signs to release or refund. Suitable for low-value escrow.
- **Quorum arbiter** — N-of-M signatures from a designated arbiter set. Suitable for mid-value escrow.
- **Human-required** — arbiter must be a human-credentialed entity per attestation. Suitable for high-value escrow.

Arbiters are **pluggable** via the arbiter registry. The Monolythium Foundation seeds the registry at genesis with a small set of proven arbiters as a public good. Anyone else stakes to register.

The counter-offer flow is a full on-chain negotiation state machine:

```
Open → Negotiating(round_n) → Accepted → InProgress → Submitted →
Released | Disputed
```

- counter-offers can modify any term (timeline, scope, price, payment schedule, penalty clauses);
- each counter-offer is signed by both parties before becoming binding;
- either party can walk away during negotiation with no penalty — escrow funds are not yet locked until `Accepted`;
- a practical round limit applies before the offer auto-closes (anti-griefing).

Reviews and disputes are distinct: reviews are ex-ante quality signal handled by reputation; disputes are ex-post fund recovery on a specific transaction, handled by escrow. They do not fuse.

18.8 Spending policy

The only consensus-critical agent-commerce primitive. The protocol must guarantee that a transaction signed by an agent sub-account violates no registered policy constraint, because the agent itself cannot be trusted to enforce its own caps.

The model is **sub-account**. Agents are sub-accounts of a human or organizational principal. The principal creates a sub-account for the agent, assigns LYTH to the sub-account, and registers a spending policy on the sub-account. The agent can sign transactions from the sub-account, but the protocol enforces:

- per-transaction caps;
- per-day, per-week, per-month rolling budget caps;
- per-counterparty allow-lists;
- per-category allow-lists (legal, dev, design, etc.);
- time-of-day windows;
- expiry dates;
- revocation by the principal at any time.

Policies are content-addressed and stored in consensus state. A transaction signed by an agent sub-account that would violate any active policy on that sub-account is **rejected at admission**, not after the fact. The agent cannot opt out of its own policy; the policy is the protocol.

18.9 Runbooks

Agents transact through **typed, versioned, signed templates** rather than free-text RPC. A runbook is a JSON-shaped operation definition with a signed parameter schema and a typed result. Initial runbooks include:

- `pay_vendor`
- `open_escrow`
- `release_escrow`
- `refund_payment`
- `book_service`
- `place_trade`
- `set_spending_policy`
- `revoke_agent_permission`
- `verify_receipt`

- `rate_vendor`

Runbooks are the bridge between natural language and safe settlement. The user speaks naturally; the agent maps intent into a constrained workflow; the wallet shows the exact action before approval. The protocol enforces the runbook's typed parameters; the agent cannot smuggle a hand-crafted RPC call past a wallet that is expecting a runbook.

---

19. Privacy Cordon

The bifurcated denomination (§5) is enforced by a native **caller-origin cordon** in the execution layer.

The cordon is a single rule, stated simply: **a public contract or module path cannot receive private-denominated value, and private-denominated state cannot be used in any public contract context.**

Concretely, the runtime tracks the denomination origin of every value passing through a contract call frame. A call that mixes denominations is rejected at admission. A contract that attempts to read private-denominated state from a public-denominated execution context is rejected. The rule is enforced by the host, not by user code, so no contract can opt out.

This is the rule that turns the bifurcation from a policy into a structural property. Without the cordon, a sufficiently clever contract could route private LYTH through a public contract and emit it on the public side. With the cordon, that path does not type-check.

Privacy modes inside the private denomination

Within the private denomination, the chain supports:

- **stealth addresses** for sender/recipient unlinkability;
- **confidential transactions** for amount hiding;
- **transfer to another private address** as the primary operation;
- **burn** as the secondary operation;
- a **one-way crossing** from public to private, with no reverse path.

A full shielded pool inside the private denomination is not part of the base protocol. The structural separation between public and private is the novel primitive; layering additional privacy machinery on top of bifurcation before bifurcation itself is operationally proven would be premature.

Privacy at the wallet edge

Wallets render the two denominations distinctly. A user moving LYTH from public to private sees the warning explicitly: “this is a one-way move; private LYTH cannot return to public.” The crossing is intentional and visible, not an accident.

20. Bridges and the Liquidity Edge

Monolythium needs liquidity but does not need to inherit EVM execution to get it. The liquidity strategy has three layers.

1. **Zero-knowledge or light-client bridges** for major external assets where proof-bound verification is feasible.
2. **Cross-chain swaps** where proof-bound settlement is better than a full bridge route.
3. **Issuer-supported native assets** where the network earns enough adoption to justify direct issuer integrations.

Wrapped assets are labeled honestly. A bridged stablecoin is not the same as a native issuer-minted stablecoin. Wallets and explorers show the route, trust model, cooldown, proof status, and risk metadata. **The goal is not to hide bridge risk; the goal is to make bridge risk legible.**

20.1 Bridge cooldowns and route risk

Bridge cooldowns are **route-specific safety parameters**, not a single global constant.

A seven-day withdrawal window can be useful for weaker, trusted bridges because it gives humans time to detect fraud. But if a bridge route is verified by a light client or a zero-knowledge proof, the security model changes. The long human-dispute window is replaced by proof verification, drain caps, circuit breakers, and explicit route-risk metadata.

ROUTE	COOLDOWN POSTURE
Ethereum finalized events	One epoch once finalized-event inclusion is verified
Solana	One to two epochs depending on finality confidence and bridge policy
Bitcoin	Two or more epochs or value-tiered limits, because finality is probabilistic
Trusted or transitional bridge	Longer cooldown until replaced by a light-client or zero-knowledge route

Cooldown reductions are valid only when the verification route, drain caps, circuit breakers, and monitoring are live. Moving from a multi-day delay to a one- or two-epoch verified route makes bridge liquidity significantly more usable for payments, markets, and agent commerce.

20.2 Bridge safety controls

Every bridge route exposes its risk model clearly:

- per-asset **drain caps** — a route cannot drain more than the configured amount in a configured time window;
- **circuit breakers** — the route can be paused automatically if anomalies trigger;
- **route-specific cooldowns** as above;
- **proof or light-client verification** where available;
- **bridge metadata visible to wallets and explorers**;
- **insurance or reserve information** where available;
- **public status** for paused, degraded, or stale routes.

This matters because users often treat all bridged assets as equal. They are not equal. A trusted multisig bridge, a light-client-verified route, a zero-knowledge bridge, and a native issuer-minted asset all carry different risks. The wallet and explorer ecosystem makes those differences visible before a user signs.

20.3 Bridge proofs and swap proofs

Bridge proofs attest that an external chain finalized a specific event, state transition, burn, lock, or withdrawal condition. The chain verifies the proof on-chain before releasing assets or updating bridge state. This reduces dependence on trusted multisigs and relayer committees, which have historically been among the largest hack surfaces in crypto.

Swap proofs verify that a batch of intents or orders was matched according to a declared policy. This supports fair ordering, batch auctions, and other matching rules without asking users to trust an opaque sequencer.

21. Hardware Sovereignty

Production operators run on **Monarch OS** — an immutable substrate built on a minimal Linux base, hardened for institutional security and designed for permissionless accessibility. A home operator with a recent gaming PC runs on the same substrate a co-located bare-metal operator runs.

21.1 Substrate properties

- **Immutable image.** Monarch OS is delivered as a single signed image. There is no package manager, no SSH, no interactive shell. The system is not a general-purpose Linux distribution; it is a purpose-built operator substrate.
- **Verified rooted filesystem.** Every block of the filesystem is verified against a signed Merkle root at read time. Tampering with the filesystem is mathematically impossible without breaking the boot.
- **TPM 2.0 measured boot.** The boot sequence's hash measurements are extended into the TPM's Platform Configuration Registers. The system can prove what it booted via TPM PCR quotes.
- **TPM-sealed operator key shares.** A cluster's BLS share is sealed against the local TPM. If the chassis is opened or the firmware is modified, the seal breaks and the operator can no longer sign with the BLS share until they re-onboard with a fresh ceremony.
- **No transitive dependency surface.** A conventional Linux distribution inherits a vast supply-chain attack surface through transitive dependencies. Monarch OS has no transitive-dep mechanism. An operator never installs a transitive dependency, so it cannot inherit one.
- **No userspace foothold for kernel exploits.** A modern class of kernel local-privilege-escalation bugs requires the attacker to first execute code as a local non-privileged user. Monarch OS structurally denies that foothold: no SSH, no interactive shell, no multi-user system, no package manager, no cron, no general-purpose userspace daemons. The only userspace process of consequence is the node binary itself.

21.2 Kernel attack-surface hardening

Beyond the absence of a userspace foothold, Monarch OS ships with an aggressively trimmed kernel configuration: features the operator node does not need are not compiled in or are compiled out of autoload.

- The userspace cryptographic kernel API is disabled at kernel-config level. The node binary performs all cryptographic work in-process via Rust libraries (ML-DSA-65, ML-KEM, BLAKE3, BLS12-381, SLH-DSA), not via kernel crypto sockets. The entire class of kernel-CVE attacks against userspace crypto APIs is closed off going forward.

- Unused kernel subsystems are disabled: virtualization (operators do not host VMs), audio and video, Bluetooth, wireless drivers (operators are wired-network-only), legacy filesystems, and other categories irrelevant to a server-grade operator workload. The smaller the kernel surface, the smaller the future-CVE exposure.

The crypto-on-OS / crypto-in-app distinction is structural. A chain that signs blocks via system-level cryptographic invocations has a fundamentally different exposure profile than one that signs blocks via in-process Rust libraries. Monolythium chose the latter from genesis. Kernel-CVE classes that target userspace crypto APIs do not affect the operator's signing path at all.

21.3 Hardware targets

Monarch OS runs on standard x86-64 hardware. Three target categories:

- **Home server.** Dedicated tower or rack server. Recent professional or consumer CPUs, 32–64 GB RAM, NVMe storage. Firmware TPM available on Intel/AMD CPUs since the late 2010s. Suitable for a home operator running a single cluster position.
- **Gaming PC.** Consumer hardware with firmware TPM. Suitable for a hobbyist operator. Performance is typically bounded by network and storage, not CPU.
- **Bare-metal co-location.** Dedicated server in a co-location facility with discrete TPM 2.0 chip. Recommended for operators offering high-bandwidth service tiers (RPC, prover, oracle).

GPU is **optional** for consensus and required for the GPU prover service tier. Operators who want to earn from the prover market install consumer or workstation-class GPUs; operators who do not, do not. Consensus participation is unaffected.

21.4 Continuous runtime attestation

Operators publish **TPM PCR quotes** every epoch to the on-chain node registry. The PCR quote is a signed declaration of the values currently in the TPM's PCRs at the time of quote generation; it proves what the operator's machine is running.

The `pcr_values` field captures the state of measurement: the bootloader, the kernel image, the substrate root filesystem, the running node binary. A change in any measured component changes the corresponding PCR. The chain stores the attestation; explorers display it; cluster members and other operators verify against expected PCR values.

A change in PCR values that is not preceded by a coordinated upgrade announcement is treated as an anomaly. Cluster members are notified; the cluster's quorum can vote to suspend the operator pending investigation.

This makes it cryptographically observable when a corporate hypervisor or unauthorized management layer is introduced underneath a Monarch node. The PCR values would shift; the network would notice; the cluster would respond.

21.5 Network and geographic diversity

Hardware attestation alone does not prevent a single entity from running multiple registered operators on physically distinct hardware in the same datacenter, on the same upstream network provider, or in the same jurisdiction. The protocol augments per-operator hardware attestation with **network-level and geographic diversity scoring** at cluster admission and during continuous attestation:

- distinct IP address per operator is a baseline requirement;
- ASN (Autonomous System Number) is recorded on-chain and entered into cluster admission scoring;
- geographic region claims are cross-checked against IP geolocation and latency triangulation from the existing cluster set;
- hosting class (bare-metal, co-location, cloud) is derived from PCR attestation patterns and network classification.

A cluster's diversity score is visible to delegators. Diverse clusters earn delegator preference; concentrated clusters lose it.

22. Threat Model

Monolythium's threat model is adversarial across the chain's full lifecycle. The chain assumes:

- some operators are Byzantine;
- some clusters may be compromised in part;
- bridges are an active attack surface;
- private keys can be stolen;
- model providers can be compromised;
- governance can be socially engineered (which is one reason the chain has none);
- cryptographic primitives may eventually fail;
- nation-state-scale attackers can target the protocol and its dependencies.

The chain's defensive posture is **separation of blast radius**. Different surfaces have different protection budgets, and a failure at one surface should not compromise another.

22.1 Surface separation

SURFACE	PRIMARY PROTECTION	FAILURE SCOPE
Consensus	Cluster threshold + equivocation slash + DAG-BFT mathematics	A failure here halts safety; the protocol must reject equivocating operators and degrade gracefully
Cryptography (user signatures)	ML-DSA-65 + emergency-key registry + algorithm rotation	A primitive break triggers rotation; users with backup keys survive; users without are frozen, not drained
Bridges	Light-client / zero-knowledge proof verification + drain caps + circuit breakers + per-route cooldown	A bridge failure is bounded to the bridge route's drain cap; consensus and accounts elsewhere are unaffected
Mempool	Threshold-DKE encryption + lifecycle-bounded confidentiality	Mempool exposure is bounded to the period between admission and inclusion (seconds)
Application contracts	Audit + native modules + sandbox boundaries	A contract bug damages the contract's users; native modules and the consensus layer are insulated
Hardware	TPM PCR attestation + immutable substrate + network/ geographic diversity scoring	A compromised operator is detectable through PCR drift; a compromised hosting class is detectable through diversity scoring
Recovery	Emergency-key registry + frozen-account claim flow	A primitive break or coordinated attack does not allow draining; affected accounts are recoverable

The point is that the chain does not have a single point at which "everything depends on this." Each surface has its own defense, and each defense has bounded failure consequences.

22.2 What the chain does not promise

The chain does not promise:

- that operators will not be compromised individually — instead, the cluster threshold and DVT structure bound the damage;
- that bridges will never be exploited — instead, drain caps, route cooldowns, and proof-bound verification bound the damage;
- that smart contracts will not have bugs — instead, native modules carry the audited primitives and contracts are sandboxed;

- that cryptography will work forever — instead, the emergency-key registry and algorithm rotation provide a planned migration path;
- that no user will lose funds to social engineering — instead, wallet UX surfaces risk and the chain's runbook + spending-policy model bounds delegated authority.

Honesty about limits is part of the threat model. A chain that claims invulnerability is a chain whose users are unprepared when invulnerability fails.

22.3 What about MEV?

MEV — value extracted by privileged actors who can reorder, insert, or censor transactions — is bounded structurally by the chain's design.

- **Encrypted mempool.** Transactions are encrypted until anchor inclusion. An operator cannot read pending transactions and front-run them; a quorum collusion would be required to decrypt early.
- **Threshold leader selection.** Wave leadership is determined by a threshold-VRF seed that operators cannot influence through block-content selection.
- **Native order book.** The native CLOB settles orders deterministically. Sequencer-style MEV games against an order book are bounded by the consensus order, which is determined by the DAG linearization rather than by a single sequencer's discretion.

MEV is not zero, and the chain does not claim it is. Some MEV — backrunning of public events, arbitrage between markets, latency-based capture — exists wherever transactions are visible. The chain's design forecloses the largest extractive categories (front-running, sandwich attacks based on mempool visibility) and leaves the remainder as a competitive market that benefits ordinary users through tight spreads.

23. Recovery and Emergency Posture

The chain's recovery posture is designed for two scenarios that are normally underspecified in L1 protocols: a cryptographic primitive breaks, and a coordinated systemic event requires the chain to freeze in place while it is sorted out.

23.1 Emergency-key registry

A user can pre-register a backup key in a different cryptographic family (specifically SLH-DSA, FIPS 205) alongside their primary ML-DSA-65 key. The registration is one-time and lives in a registry trie that does not widen the account record.

The backup key is **never used for normal transactions**. It is dormant unless the protocol declares an emergency algorithm rotation. At that point:

- the protocol stops accepting signatures using the broken algorithm at a specified anchor height;
- users with a registered backup key sign with the backup key from that anchor forward;
- the wallet UX transitions the user to the backup key transparently;
- users without a registered backup are **frozen** — never drainable by the attacker, recoverable through a runbook-based claim process.

The frozen-account claim process requires the principal to prove control of the account through a series of out-of-band attestations (recovery contacts, KYC linkage if it existed, hardware-bound emergency tokens). The process is documented in the recovery runbook and is designed to be slow and human-checked, so that the very attack that triggered the freeze cannot be used to drain frozen accounts through the recovery channel.

23.2 Emergency freeze — scope and limits

In the case of a coordinated systemic event — a discovered cryptographic break before rotation completes, a major bridge exploit in progress, or a confirmed adversarial fork — the chain supports an emergency **freeze** mechanism. The freeze is gated by a multi-signature Foundation key with a declared signer set and a ratification window.

What the freeze is for (the entire scope, exhaustively):

- pausing admission of new transactions during a confirmed cryptographic-primitive break, while users transition to the emergency-key registry;
- pausing affected bridge routes during a confirmed active exploit, to bound the drain;
- pausing transaction admission during a confirmed adversarial fork, until the canonical chain state is re-established.

What the freeze is not for:

- routine protocol upgrades — those happen through operator software rollout, not through the freeze mechanism;
- parameter changes — the constitutional parameters (supply cap, inflation cap, delegation cap schedule, reward curve shape) require a coordinated hard fork, not a freeze;
- protocol-direction decisions — direction is set by maintainers and operators in public, not through the emergency mechanism;
- asset confiscation — the freeze pauses transaction admission; it does not move user funds and cannot rewrite balances;
- ongoing supervision — the freeze is time-bounded by the ratification rules and cannot be sustained as a normal operating mode;
- censorship of specific accounts — the freeze pauses the chain globally; per-account freezing is not a freeze-mechanism action.

The freeze is a circuit breaker: it exists so that the chain has a documented, accountable, time-bounded path through a worst-case event, instead of relying on improvisation under duress. It is **not** a governance backdoor. The chain's no-governance design (§4.1) and the freeze mechanism are compatible exactly because the freeze is scoped to events with clear, observable triggers (a cryptographic break, an active bridge exploit, an adversarial fork) and not to ongoing decisions about how the protocol should evolve.

The signer set, signer responsibilities, ratification timeline, and post-freeze audit obligations are published and verifiable. Every freeze action is logged, justified, and reviewable.

23.3 Multisig treasury

The Monolythium Foundation treasury is held in a multisig with a published signer set. The treasury cannot move funds without a quorum, and the quorum cannot be reduced through any in-protocol mechanism. Treasury transactions are published with rationale.

The treasury is funded from the genesis reserve (allocation category 2 in §16.2) and from buyback-burn yield over time. It does not draw on the 8% annual inflation cap.

23.4 Recovery runbooks

Recovery procedures — frozen-account claim, key rotation, emergency freeze invocation, bridge rollback — are documented as **runbooks**, the same typed-template structure used by application-layer agent commerce. A runbook for an emergency procedure is signed by the Foundation, published in advance, and verifiable. Operators and users can read the runbook before the emergency, not during it.

PART 3 — ADOPTION AND OUTLOOK

24. Developer Experience

The developer experience target is:

- Rust-first contract development;
- a Mono contract SDK;
- deterministic local testing;
- ABI and artifact generation;
- typed events and receipts;
- wallet and explorer support for MRC assets;
- contract deployment tooling;
- testing and fuzzing utilities;
- bridge and market simulation tools;
- indexer-ready schemas.

The chain must replace the convenience that EVM compatibility normally provides. That means SDKs, docs, templates, wallets, explorers, and examples are not optional — they are part of the core adoption surface.

The chain is designed to be approachable for:

- Rust developers;
- AI-assisted developers using LLMs that already understand Rust well;
- fintech and payments teams;
- game and NFT builders;
- agent-platform builders;
- bridge and infrastructure operators;
- teams that need post-quantum account security.

The goal is not to make EVM developers feel at home by copying Ethereum. The goal is to make a better native environment for the next category of builders.

24.1 AI runbooks for developers

AI assistants do not improvise raw financial actions on Monolythium. They select from typed, wallet-visible runbooks that can be simulated, approved, executed, indexed, and audited. Initial runbooks include:

- `pay_vendor`

- `open_escrow`
- `release_escrow`
- `refund_payment`
- `book_service`
- `place_trade`
- `set_spending_policy`
- `revoke_agent_permission`
- `verify_receipt`
- `rate_vendor`

Runbooks are the bridge between natural language and safe settlement. The user can speak naturally; the agent maps intent into a constrained workflow; the wallet shows the exact action before approval.

24.2 AI-assisted contracts

The Rust ecosystem has fifteen years of training data behind it. AI coding assistants write good Rust today and will only get better. Smart-contract languages with smaller training corpora are at a structural disadvantage in the AI-assisted development era.

The chain's choice of Rust is partly a bet that the cost of writing a contract will fall fastest on the language with the deepest, cleanest training surface. The bet is not that AI will replace contract authors — it is that the marginal cost of producing a correct, well-tested, idiomatic contract will fall faster in Rust than in any alternative, and the chain wants to ride that curve.

---

25. User Experience

For everyday users, the chain should not feel like a research project.

Users hold LYTH and MRC assets. They see whether an asset is native, wrapped, bridged, or private. They understand bridge risk before signing. They trade tokens and NFTs. They use spot markets. They authorize agent spending policies. They revoke or update permissions. They view reputation and escrow state. They understand finality and cooldown status.

The wallet does not expose users to raw implementation details. But it does not hide important risk either.

A bridge route with weaker trust assumptions looks different from a light-client or zero-knowledge route. A wrapped asset looks different from a native issuer asset. An agent account with spending authority is easy to inspect and revoke. The bech32m address with its per-type discriminator (§13) tells the user at a glance whether they are sending to a person, a contract, a cluster, or a bridge.

This is one of the most important adoption requirements. A non-EVM chain compensates with **clarity**.

25.1 The four-button autovote

Delegation is enforced at the protocol layer (§16.7); day-to-day delegation happens in the wallet. The chain's reference wallet ships an **Intelligent Autovote** surface with four named modes:

- **Max Yield.** Allocate to the highest-APR clusters consistent with the per-cluster cap.
- **Max Diversity.** Spread allocation across as many independent clusters as the current cap allows, weighted by reputation and uptime.
- **Max Decentralization.** Actively route stake away from clusters with high correlated-preference scores, geographic concentration, or shared operator membership.
- **Custom.** Manual per-cluster allocation, with the cap enforced at submission time and the wallet warning before any out-of-policy distribution is signed.

The four-button surface gives the user a simple choice over a complex underlying state machine. The user does not need to think about cap percentages or diversity scoring; the wallet does.

25.2 The wallet shows risk

Wallets render risk in addition to balance. Before a user signs:

- the asset's denomination (public or private);
- the asset's route (native, wrapped, bridged, light-client-verified, zero-knowledge-verified);

- the bridge’s drain-cap remaining and circuit-breaker status;
- the agent sub-account’s policy summary if applicable;
- the runbook the assistant has selected, with parameters spelled out;
- the recipient’s name (if registered) and the recipient’s bech32m address;
- the finality posture (anchor-level or quantum-attested checkpoint) the receiving party expects.

Users can decline. Users can approve once, approve once with caps, or approve a long-running policy. The wallet preserves the user’s authority.

---

26. Honest Limitations

The chain's direction is a strategic bet. The risks are real and named.

- Some developers will not migrate from Solidity to Rust.
- Liquidity may arrive more slowly without direct EVM compatibility.
- Wallets and explorers require more custom work than they would on an EVM chain.
- Native MRC standards must earn trust before they reach the ubiquity ERC standards have.
- The Rust/RISC-V contract tooling must be excellent — anything less than excellent loses to a familiar EVM environment.
- Zero-knowledge bridge circuits are complex and require serious audit work.
- Bridge prover infrastructure adds operational burden.
- The market for agent commerce may take longer to mature than expected.
- Post-quantum signatures are larger than classical signatures, raising storage and bandwidth costs.
- Distributed validator technology has not been deployed at the chain's target scale in a leaderless DAG-BFT configuration; the operational learning is ahead, not behind.
- The bifurcated denomination is structurally hostile to certain user expectations from existing privacy chains; users coming from those chains will find the constraints unusual.
- The composition stance with the major agent-payment standards (§3) depends on those standards remaining open and composable. A standard that closes off the integration surface would constrain the wedge for that particular rail; the chain's bet is that most of them stay open, because their open status is part of their adoption story.

These are not footnotes. They are the cost of choosing a distinct category.

The counter-risk is also real: becoming another EVM-compatible chain may make adoption easier at first but leave the project buried under stronger incumbents, deeper liquidity, and thousands of similar competitors. The chain accepts the harder path because the easier path leads to a category in which it is already too late to win.

27. What Success Looks Like

Monolythium succeeds if it becomes a credible settlement layer for:

- AI agents acting on behalf of human or organizational principals;
- human-to-agent commerce;
- agent-to-agent commerce;
- payments;
- token and NFT issuance;
- spot markets;
- cross-chain swaps;
- safer bridge liquidity;
- long-lived, post-quantum digital identities.

The early test is not whether every Solidity project deploys unchanged. That is not the chosen market.

The early test is whether users and builders understand the value of safer bridges, visible route risk, native agent permissions, Rust-native contracts, post-quantum accounts, clean token and market standards, and a chain designed around the next decade rather than the previous one.

Success is also measurable along structural dimensions that are independent of any single market hype cycle:

- the number of independent clusters and the geographic and ASN distribution of their operators;
- the depth of the discovery registry and the number of legitimate providers it hosts;
- the volume of escrowed agent-to-agent and human-to-agent transactions;
- the share of bridge volume moving through proof-bound routes versus trusted-multisig routes;
- the volume of agent-payment-standard composition (x402, AP2, ACP, MCP) settling against Monolythium spending policies and escrows;
- the diversity of MRC asset issuers;
- the survival of agent identities across model providers — that is, the structural portability of reputation.

A network that scores well on these is a network that has delivered what the design promised. A network that scores well on token price alone has not.

28. Closing

Monolythium is a deliberate break from the default Layer-1 playbook.

It does not try to win by becoming a slightly faster EVM chain. It chooses Rust on a deterministic RISC-V target, post-quantum accounts as default, native asset standards, native markets, native agent-commerce primitives, zero-knowledge and light-client bridge liquidity, focused use of zero-knowledge proofs at the highest-value boundaries, a structurally non-fungible privacy denomination, a public cluster marketplace, and a smaller protocol surface.

It composes underneath the major agent-payment standards rather than fighting them, providing the chain-anchored policy, escrow, identity, and reputation layer those rails leave open.

That choice is harder. It means more tooling, more education, and a slower path to existing Solidity liquidity. It also gives the chain a clearer identity and a stronger foundation for the workloads it expects to serve.

The thesis is straightforward: the next major settlement layer will not be the chain that copies Ethereum most closely. It will be the chain that gives new economic actors — autonomous agents, organizations delegating to software, services that want neutral cross-platform rails — safer rails to transact across applications, markets, and jurisdictions.

Monolythium is built for that future.

Acknowledgments

The Monolythium protocol is the product of years of design work, research, and engineering by the team at Mono Labs R&D LLC, in coordination with the Monolythium Foundation. The protocol draws on a wide body of academic and open-source work, including:

- the Dilithium, Kyber, and SPHINCS+ post-quantum primitives standardized by NIST as ML-DSA, ML-KEM, and SLH-DSA;
- the Starfish family of leaderless DAG-BFT consensus protocols;
- the BLS12-381 pairing-friendly curve and associated threshold-signature literature;
- the Ferveo threshold-DKE construction for encrypted mempool decryption;
- the FROST distributed key generation family;
- the BIP-39 wordlist and bech32m encoding conventions;
- the FRI-based proof-system family and the broader zero-knowledge research community;
- the BLAKE3 hash function;
- the open agent-payment standards that the chain composes against — x402, AP2, ACP, and MCP — whose existence makes the settlement-layer wedge concrete and addressable.

This whitepaper is a synthesis of those contributions into a specific Layer-1 architecture. Errors are the authors'. Attribution should be given to the project, not to the underlying primitives whose authors deserve their own citation.

Entities

Monolythium is operated and stewarded by two entities:

- **Mono Labs R&D LLC** — the operating company, based in San Francisco, California. Mono Labs R&D LLC develops the Monolythium protocol, operates the reference client and SDK, and is the seller of record for the LYTH token in jurisdictions where a seller of record is required.
- **Monolythium Foundation** — the protocol steward, based in the Cayman Islands. The Foundation operates the chain's emergency-key registry, the Foundation multisig treasury, the genesis name reserve, the emergency-freeze ratification window, and other constitutional-layer functions that require an entity rather than a company.

The two entities are independent, with separate governance, separate financial accounts, and separate roles in the protocol's operation.

License

The text of this whitepaper is licensed under the **Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0)**.

You are free to share and adapt the material, including for commercial purposes, under the following conditions:

- **Attribution.** Give appropriate credit, provide a link to the license, and indicate whether changes were made.
- **ShareAlike.** If you remix, transform, or build on the material, distribute your contribution under the same license as the original.

The full license text is available at: https://creativecommons.org/licenses/by-sa/4.0/legalcode

Attribution string:

> Monolythium Whitepaper, v5.0 (May 2026), Mono Labs R&D LLC, licensed under CC BY-SA 4.0.

This license applies to the **whitepaper text** in this document. The Monolythium protocol source code is licensed separately under the Business Source License 1.1, with a four-year commercial restriction window before automatic conversion to a permissive license. Selected execution crates ship under MIT.

End of Monolythium Whitepaper v5.0.