Whitepaper, version 1.0, September 2026

Epoch Nexus

A distributed intelligence engine for sub-second, asynchronously secure settlement.

Consensus

Multi-layered asynchronous BFT with a common-coin commit rule

Scaling

Dynamic sharding steered by localized neural nodes

Status

Pre-testnet. Every performance figure is a design target, not a measurement.

Contents

Reading guide

Sections 3 to 6 are technical. Sections 1, 2, 7 and 8 stand alone for readers who want the case, the economics and the plan.

Abstract

Epoch Nexus treats throughput as a scheduling problem as well as an agreement problem.

Epoch Nexus is a layer-1 network whose ordering core is a multi-layered asynchronous Byzantine fault tolerant (aBFT) protocol. It stays safe and live without any timing assumption, and tolerates up to one third of validators acting maliciously.

Above that core sits the **Distributed Intelligence Engine** (DIE): a set of localized neural nodes that forecast where load is about to appear and propose how computation should be dynamically sharded across the network. The engine can only improve performance. It cannot compromise safety, because every recommendation is a proposal that consensus checks against deterministic rules before it takes effect.

The goal is sub-second finality and near-linear scaling as shards are added, removing the transaction bottlenecks that limit legacy layer-1 networks. This paper describes the architecture, the protocol, the economic model and the milestones that will show whether the design works.

Design targets at a glance

Property	Target
Finality, median, geo-distributed	600 ms or less
Finality, 99th percentile	1.5 s or less
Fault tolerance	Fewer than one third Byzantine, no timing assumption
Shards	16 at mainnet beta, growing to 64
Aggregate throughput	50,000 or more simple transfers per second at 16 shards
Cross-shard transaction latency	About two epochs, 1.2 s or less

Targets are goals to be validated on the public testnet described in section 8. They are not measured results and may change.

> **What is new here.** Sharded networks usually fix their partition boundaries at genesis and rely on timeouts for safety. Epoch Nexus moves partition boundaries at runtime, guided by forecasts, while keeping the forecaster entirely outside the safety path. The next pages explain why that separation matters.

1 The bottleneck

Legacy layer-1 networks hit the same wall from three directions at once.

Serial agreement. Most networks push every transaction through a single ordering pipeline. Adding validators improves security but not capacity, so demand beyond the pipeline's limit becomes a queue and then a fee auction.

Static partitioning. Sharding fixes part of this by splitting state, but partition boundaries chosen at genesis assume load is spread evenly. Real demand is not: one popular application turns one shard into a hot spot while others sit almost idle. Users on the hot shard pay more and wait longer even though the network as a whole has room.

Timing assumptions. Partially synchronous protocols are safe only while messages arrive within a bound the operators guessed in advance. Under attack or during a regional outage, they stall or fork, and the recovery depends on someone retuning a timeout.

Applications end up designing around congestion instead of relying on the network: batching, queuing and retry logic that users never see but always pay for.

Constraint	Where legacy designs bind	Epoch Nexus response
Ordering	One serial pipeline; more validators add security, not capacity	Parallel dissemination, ordering runs on small certificate digests
Partitioning	Static shards fixed at genesis; a popular app overloads one shard	4,096 movable buckets; boundaries change at epoch boundaries
Timing	Partial synchrony; timeouts tuned by hand decide safety margins	Fully asynchronous; no clock is trusted
Congestion	Fees spike locally while global capacity sits idle	Per-shard fees feed the engine, which moves load toward spare capacity

2 Vision and design principles

Measure, forecast and rebalance continuously, without ever trusting the forecaster.

An **epoch** is a fixed consensus window in which the validator committee finalizes a set of certified batches. Epoch boundaries are the only moments at which the network may safely change its own configuration, including shard boundaries. Everything in this paper follows from taking that boundary seriously.

- **Safety is never learned.** Machine learning influences performance only. Correctness rests on aBFT rules that a formal model can check.
- **No clocks in the trust base.** Liveness and safety hold under arbitrary message delay, so a slow network degrades speed, not security.
- **Load follows the work.** Buckets of state move toward spare capacity ahead of demand, not after users complain.
- **Local first.** Neural nodes sit near the data they observe and share compact forecasts instead of raw telemetry.
- **Everything auditable.** Forecast inputs, outputs and the deterministic checks that accept or reject a plan are committed on-chain.

3 Architecture

Three layers with one rule between them: information flows up as telemetry, and proposals flow down as plans that consensus may reject.

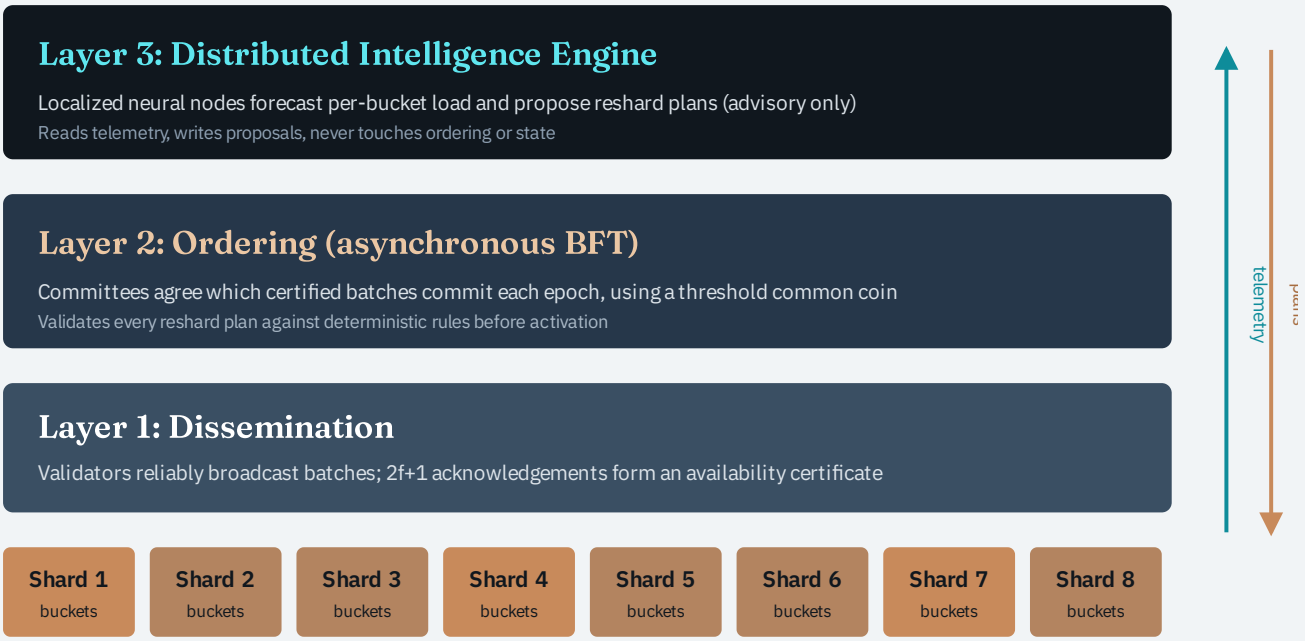


Figure 1. Telemetry rises from the shards to the intelligence layer (cyan arrow). Reshard plans return as proposals to the ordering layer (bronze arrow), which validates them.

3.1 Dissemination layer

Each validator gathers transactions into batches and reliably broadcasts them. When 2f+1 validators acknowledge a batch, its author assembles a certificate proving the data is available. Data moves in parallel across the whole validator set, and everything downstream works on 32-byte certificate digests instead of full batches. This is why ordering does not become the bottleneck as volume grows.

3.2 Ordering layer

Certificates reference certificates from the previous round, forming a directed acyclic graph. Committees run an asynchronous commit rule over that graph, using a threshold common coin to pick a leader certificate after the round's certificates are fixed. Committing the leader commits its entire causal history in a deterministic order. Section 4 gives the detail.

3.3 Intelligence layer

Neural nodes are ordinary participants that bond stake and run compact forecasting models. They observe mempool arrival rates, account contention and access patterns in their locality, and publish signed forecasts of load per **bucket**. State is hashed into 4,096 buckets, and a shard is simply a set of buckets. A reshard plan is a list of bucket moves. Section 5 covers how plans are formed, checked and executed.

Validators
Stake, disseminate, certify and order. Sampled into shard committees each epoch.

Neural nodes
Forecast load and propose plans. Rewarded for accuracy, slashed for provable misbehavior.

Light clients
Verify finality with one aggregate signature and a Merkle proof per bucket.

4 Consensus: layered asynchronous BFT

4.1 Model

The network has n = 3f + 1 validators, of which up to f may be Byzantine. Messages between honest validators are eventually delivered, but the delay is unbounded and controlled by the adversary. No protocol step waits on a timeout to preserve safety, and the protocol terminates with probability one thanks to the common coin. The design follows published work on DAG-based asynchronous BFT (references 2 to 5) and adds the layering, sharding and reconfiguration described here.

4.2 One epoch in four steps

1 Disseminate
Each validator reliably broadcasts its batch to all others.

2 Certify
2f+1 acknowledgements form an availability certificate.

3 Link
New certificates reference 2f+1 certificates from the prior round.

4 Commit
The common coin picks a leader; its causal history commits in fixed order.

Because the coin is revealed only after the round's certificates are fixed, an adaptive adversary cannot know which leader to attack in advance. When the network behaves well, an optimistic path commits in a single wave; under attack it falls back to the coin path without any view-change timeout.

4.3 Latency budget

Finality is dominated by message delays, not computation. The arithmetic below assumes four message delays per commit and two network scenarios: a regional deployment with 30 ms one-way delay, and a global deployment with 75 ms.

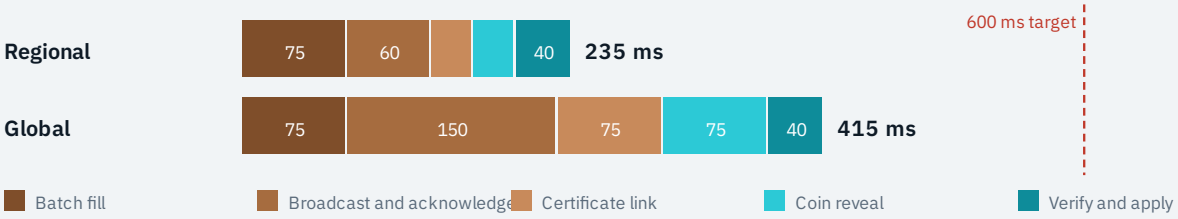


Figure 2. Design arithmetic in milliseconds, not measurements. The 600 ms median target leaves about 45% headroom over the global case for queuing, signature aggregation and stragglers.

4.4 Cryptography

Certificates use aggregate BLS signatures over BLS12-381, so a light client verifies finality with one signature. The common coin is a threshold BLS scheme. Account signatures default to Ed25519. All schemes are versioned protocol parameters, so a post-quantum migration is a governed upgrade, not a hard fork.

> **Why no timeouts matter.** In a partially synchronous protocol, a network partition can force a choice between safety and liveness. Here the choice never arises: during a partition the network makes progress only where a quorum exists, and it never finalizes conflicting histories.

5 The Distributed Intelligence Engine

A forecast that can be wrong is safe to use only if being wrong is cheap. The engine is designed so that it is.

5.1 Forecasting near the data

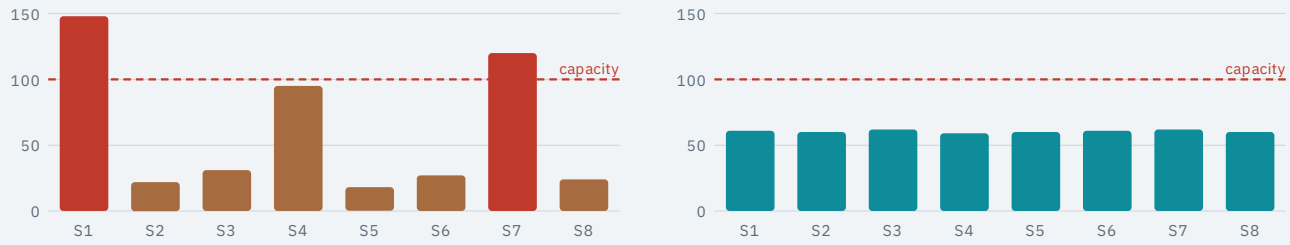
Each locality runs a small forecasting model, on the order of tens of megabytes, over local mempool arrival rates, contention on hot accounts and recent access patterns. Nodes publish a signed forecast of load per bucket for the next several epochs. A locality's forecasts are combined with a trimmed mean, so a minority of dishonest or faulty neural nodes cannot move the result far.

5.2 Guardrails: plans are proposals

The engine proposes a reshard plan. The ordering layer accepts it only if a **deterministic evaluator**, which uses no learned component and reads only on-chain telemetry from past epochs, confirms all of the following:

- No more than **2%** of buckets move in one epoch, and no bucket moves twice within eight epochs.
- No shard falls below its minimum bucket count or exceeds its maximum.
- The plan reduces the measured imbalance score of the previous epochs by at least the governed threshold.

If any check fails, the plan is a no-op and the network continues unchanged. A broken model therefore costs performance for an epoch, never funds or correctness.



Static boundaries. Load as a percentage of shard capacity under a skewed workload. Shards 1 and 7 exceed capacity.

After rebalancing. Same total load with buckets redistributed at epoch boundaries.

Figure 3. Illustrative simulation input showing the intended effect. It is not a measured result; the milestone M2 exit criterion in section 8 defines how it will be measured.

5.3 Bucket migration protocol

1 **Announce** › 2 **Snapshot** › 3 **Verify** › 4 **Activate** › 5 **Release**

The source shard freezes a bucket at an epoch boundary and publishes a Merkle-committed snapshot. The destination verifies the root against the value committed on-chain, then activates the bucket in the next epoch. Transactions that touch the bucket during the handoff are re-queued rather than dropped. Once the destination has served an epoch, the source releases its copy.

5.4 Incentives

Neural nodes bond stake and earn rewards according to a proper scoring rule that compares each forecast with the load that actually arrived. Signing two conflicting forecasts in one epoch is provable misbehavior and is slashed. Forecast accuracy is the only way to earn, so honest, careful forecasting is the profitable strategy.

6 Execution, cross-shard atomicity and security

6.1 Execution

Each shard executes committed batches deterministically. Transactions declare the state they read and write, which lets a shard execute non-conflicting transactions in parallel and re-execute conflicting ones serially. Because inputs and order are fixed by consensus, every honest replica reaches the same state.

6.2 Cross-shard atomicity

A transaction that touches several shards uses certified receipts. The coordinator shard locks the input versions in epoch e and issues a receipt certificate. Each participating shard verifies the certificate and applies its part in epoch e+1 or later. If any participant rejects, the coordinator issues a rollback certificate that releases the locks. Time limits are counted in epochs, so the protocol keeps its asynchrony guarantees. The design target is completion in about two epochs.

```
tx.lock(inputs @ version v)         # epoch e, coordinator shard
receipt = certify(2f+1 sigs)        # availability plus validity
for shard in participants:
    apply(receipt) at epoch >= e+1    # verified against committed root
on any reject: rollback_cert -> unlock(inputs)
```

6.3 Security model

The security goal is that no honest history is ever reverted (safety) and that valid transactions eventually finalize (liveness), assuming fewer than one third of validators in each committee are Byzantine. The table lists the main threats and how the design contains each.

Threat	Mitigation
Byzantine validators (up to f)	aBFT safety; committees resampled every epoch from unbiasable common-coin randomness
Leader targeting and denial of service	No fixed leader; the coin picks after certificates are fixed
Forecast poisoning to force bad plans	Trimmed-mean aggregation, bonds and slashing, movement caps, deterministic evaluator
Shard takeover by concentrating stake	Committees of several hundred validators; size derived so that the chance of exceeding one third adversarial stake is negligible at an assumed global adversary, and re-derived at each reconfiguration
Cross-shard double spend	Versioned locks plus certified receipts; expiry measured in epochs, never wall-clock time
Tampered model weights	Model hashes committed on-chain; models are advisory, so a bad model can only trigger a rejected plan
Long-range attacks	Checkpoints every 1,024 epochs and weak-subjectivity sync for new nodes

> **Verification plan.** The epoch protocol, the migration protocol and the cross-shard protocol will each be specified in TLA+ and model-checked before the public testnet. A mechanized proof of the core safety property is a milestone M4 exit criterion.

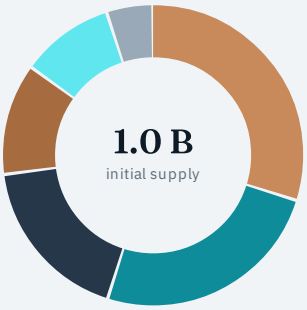
7 Token and economics

The token exists to pay for network work and to make honest work the profitable choice. It is not a claim on revenue or profit.

7.1 Roles

- **Fees.** Users pay a per-shard base fee plus an optional tip for execution and data availability.
- **Staking.** Validators stake to join committees; stake at risk backs the safety assumption.
- **Bonds.** Neural nodes bond tokens that are slashed on provable misbehavior.
- **Governance.** Holders vote on parameters and treasury spending (section 9).

7.2 Illustrative allocation



Ecosystem and grants	30%
Validator and neural-node rewards	25%
Core contributors	18%
Treasury	12%
Community and strategic sale	10%
Liquidity and market operations	5%

Figure 4. Illustrative starting parameters, subject to legal review and community governance. Core contributor tokens vest over four years with a 12-month cliff. Validator and neural-node rewards are emitted over ten years on a declining schedule.

7.3 Fee market and the intelligence engine

Each shard adjusts its base fee every epoch by up to 12.5% toward a 50% utilization target, in the manner of EIP-1559 (reference 7). A hot shard therefore prices congestion locally, and that price is one of the signals the engine reads. The engine moves buckets toward spare capacity; the fee market discourages load from piling up while it does. The two mechanisms push in the same direction and neither depends on the other for safety.

Where each base fee goes

The final 10% rewards neural nodes in proportion to forecast accuracy. Burning half of every base fee links supply to real network use.

> **No return is promised.** Token value can fall to zero. Nothing here is an offer or solicitation, and holding the token confers no equity, dividend or right to profit.

8 Milestones and roadmap

Each milestone ends with an exit criterion that a third party can check. A milestone is complete only when its criterion is met and the evidence is public. Dates are indicative and will move if evidence says they should.

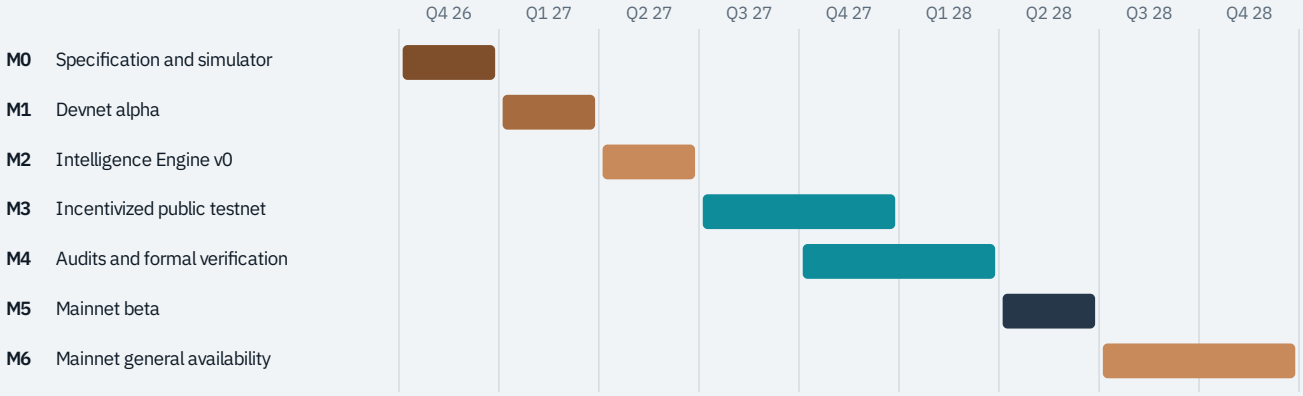


Figure 5. Indicative schedule, Q4 2026 to Q4 2028.

Milestone	Deliverables	Exit criterion
M0 Q4 2026	**Specification and simulator.** Protocol spec v0.9, TLA+ models, deterministic simulator, open reference repository.	Simulator upholds safety and liveness over 10,000 randomized schedules with f Byzantine nodes; two external research reviews.
M1 Q1 2027	**Devnet alpha.** Single-shard aBFT core on 64 validators; benchmark methodology and raw data published.	Median finality 400 ms or less on a regional network; 72-hour soak with zero safety violations.
M2 Q2 2027	**Intelligence Engine v0.** Neural nodes, forecast aggregation, deterministic evaluator, bucket migration on an 8-shard devnet.	At least 40% lower worst-shard overload than static shards on a skewed replay; zero unsafe migrations under fault injection.
M3 Q3 to Q4 2027	**Incentivized public testnet.** Permissionless validators and neural nodes, chaos-testing campaign, bug bounty.	200+ validators and 50+ neural nodes across 5+ regions; 30-day median finality 700 ms or less.
M4 Q4 2027 to Q1 2028	**Audits and formal verification.** Two independent audits (consensus and cryptography; economics and contracts), mechanized safety proof.	All critical and high findings fixed; audit reports and proof published.
M5 Q2 2028	**Mainnet beta.** Guarded launch with validator cap, per-shard circuit breakers and value limits.	90 days without a critical incident to lift caps; measured median finality 600 ms or less.
M6 Q3 to Q4 2028	**Mainnet general availability.** On-chain governance, treasury and grants, TypeScript and Rust SDKs, cross-shard app framework.	First parameter change ratified by on-chain vote; SDKs released with reference apps.

Evidence policy. Benchmarks are published with methodology, configuration and raw data, so anyone can reproduce or dispute a result. A milestone whose exit criterion is missed stays open and is reported as such.

Gates. Mainnet beta (M5) cannot begin until M4 audit reports are public and every critical finding is fixed. Caps are lifted only after 90 incident-free days.

9 Governance, risks and legal notice

9.1 Governance

Three bodies share control. **Token holders** vote on protocol parameters and treasury spending. A **validator council** must ratify consensus-critical upgrades by a two-thirds supermajority. Every approved change passes a **14-day time-lock** before it activates, so users and operators can exit or object. An emergency circuit breaker may pause cross-shard transfers and resharding only. It cannot alter balances, and it expires 18 months after mainnet general availability.

9.2 Principal risks

Risk	Description and mitigation
Novel combination	Sharding, asynchronous consensus and learned control have not been combined at scale. Mitigation: advisory-only engine, staged milestones, formal models.
Missed performance targets	Finality and throughput are targets. Physical network delay sets a floor. Mitigation: publish methodology and raw data at M1, M3 and M5.
Economic miscalibration	Rewards, fees and slashing may need retuning. Mitigation: parameters governed on-chain with time-locks.
Regulatory uncertainty	Rules for tokens and validators differ by jurisdiction and change over time. Mitigation: legal review before any distribution.
Adoption	A fast network without applications has little value. Mitigation: grants, SDKs and reference apps in M6.

9.3 Legal notice

This document is for information only. It is not an offer or solicitation to sell any security, token or other instrument, and it is not financial, legal or tax advice. Statements about performance, features, schedules and token parameters describe intentions and design targets, not commitments, and they may change. Digital assets are volatile and may lose all value. Nothing here creates any right to revenue, profit, ownership or governance in any entity. Regulatory treatment varies by jurisdiction; readers are responsible for complying with the laws that apply to them and should seek independent professional advice.

9.4 References

1 Castro, M. and Liskov, B. Practical Byzantine Fault Tolerance. OSDI 1999.

2 Miller, A. et al. The Honey Badger of BFT Protocols. ACM CCS 2016.

3 Keidar, I. et al. All You Need is DAG. ACM PODC 2021.

4 Danezis, G. et al. Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus. EuroSys 2022.

5 Spiegelman, A. et al. Bullshark: DAG BFT Protocols Made Practical. ACM CCS 2022.

6 Zamani, M. et al. RapidChain: Scaling Blockchain via Full Sharding. ACM CCS 2018.

7 Buterin, V. et al. EIP-1559: Fee market change for ETH 1.0 chain. 2019.