AEVA: An Asset-Native Ledger for Programmable Ownership, Financial Rights, and Settlement

Architecture and protocol design

AevaChain LLC · aevachain.com · Draft v0.1, September 2026 (testnet-0)

Abstract. Existing blockchains model accounts, balances and contract state; the assets that matter to finance are bundles of distinct rights: economic ownership, income, voting, redemption, control, together with the encumbrances and claims that sit on top of them. Representing such an asset as a fungible token forces every right, restriction and obligation into contract code that the ledger itself cannot see or verify. We propose a Layer 1 whose state machine models ownership directly. An asset is a root with a fixed schema of conserved right kinds; a position is a per-kind balance; a whole share is one unit of every bundled kind, and a strip is a kind held apart. The protocol enforces conservation of rights, bounds every encumbrance by the position it sits on, evaluates a closed transfer-policy language against the post-state of every transition, settles delivery versus payment atomically, and pays income through a constant-time accrual index. Identity is credential-based and carries no personal data on chain. Consensus is Byzantine fault tolerant with single-block finality; an Ethereum Virtual Machine is provided as a facade whose precompiles expose native rights as ERC-20-compatible tokens without ever holding their state. Every invariant is checked by validators at the end of each block and independently reconciled by an event-sourced indexer. The result is a ledger that cannot be wrong about itself and that can settle what it knows in one block.

1. Introduction

Tokenization has largely meant issuing an ERC-20 whose transfer function calls a whitelist. The share, the bond, or the fund unit becomes a balance, and everything that makes it a financial instrument (who may hold it, what it pays, what has been pledged against it, what happens on liquidation) is reconstructed in application code above the chain. Two consequences follow. The ledger has no notion that a dividend right and a voting right derive from the same underlying share, so nothing prevents the sum of derivative claims from exceeding the asset they derive from. And because the rules live in per-issuer contracts, every issuer re-implements custody, lockups, eligibility and settlement differently, and every counterparty must audit each implementation.

We start from a different premise: *ownership is the primitive*. Aeva's state machine understands that an asset has an issuer, a schema of rights with fixed supplies, holders of units of each right, locks that encumber some of those units in favour of a beneficiary, and a settlement asset in which income and proceeds are paid. Transactions mutate this structure directly, and the protocol refuses any transition that would violate its invariants. Smart contracts are not the mechanism of the design; where an execution environment is offered it is a guest that reads and moves native rights through checked interfaces.

This paper describes the protocol as implemented for the first public testnet: the data model (Sections 3 and 4), transfers and the policy language (5), identity (6), encumbrances (7), settlement (8), income (9), waterfalls and redemption (10), issuance and roles (11), the execution environment (12), consensus and validator design (13), observability (14), interoperability (15), and the limitations we consider honest to state (16).

2. Design principles

The design follows a small number of rules that were fixed before the first line of code and that decided every later trade-off.

Native state, guest execution. Assets, rights, locks, settlements and payouts are protocol objects in the state machine. The EVM (Section 12) reaches them only through precompiles that call the same keeper functions native messages call. No unit of any right is ever stored in contract storage.

Conservation. For every asset and right kind, the sum of all positions equals the supply. Supply changes only through issuance and redemption by authorised roles. Everything that looks like splitting an asset into parts is either a transfer of a subset of its kinds or a lock on some of its units; nothing is minted.

Post-state validation. A transfer policy is evaluated against the state that would result from the whole transition, not against the state before it. A settlement whose legs would each fail alone but succeed together (a holder-count cap, for instance) is accepted; order of legs never matters.

A closed policy language. Transfer rules are a fixed grammar with bounded depth and size, never a program. Evaluation cost is bounded, results are deterministic, and a denial names the rule that failed.

No personal data on chain. Identity is a set of typed attestations by named attesters, with expiry and revocation. Which attesters an asset trusts is declared by the asset, not by the network.

Nothing iterates holders. Distributions, liquidations, splits and invariant checks are constant or logarithmic in the number of holders. Income is pulled, never pushed.

Raw state, presentational scaling. A stock split changes a presentation multiplier, never the raw units in which conservation, locks and accrual are computed.

The change log is the events. Every state mutation emits exactly one typed event, and an independent indexer rebuilds the entire rights graph from those events alone, then reconciles it against the chain's own invariant queries every block. A disagreement is recorded, never corrected.

Sovereign, with a swappable engine. The state machine speaks to consensus through the application blockchain interface, so the consensus engine can be replaced without touching the asset model.

3. Asset primitives

Aeva has exactly two kinds of fungible value. A *coin* is a plain bank denomination with no rights structure: the staking token AEVA and the settlement stablecoin. Everything else is an *asset*.

3.1 The asset root

An asset root is created by its issuer and carries: a content-addressed identifier derived from the issuer and a per-issuer nonce; the issuer account; optional roles (custodian, transfer agent, operator); a hash of the off-chain legal document; a settlement denomination; a fixed schema of right kinds; a transfer policy; a liquidation waterfall; a set of denominations in which income may be paid; and flags (issuance locked, liens allowed) with a status of ACTIVE, FROZEN, LIQUIDATING or LIQUIDATED.

3.2 Right kinds

Each right kind under a root has a class (ECONOMIC, INCOME, VOTE, REDEMPTION, CONTROL or CUSTOM), a conserved supply, a decimal precision, a *bundle* flag, an optional term, a transferable flag and an optional policy override. The class is a label for tooling; the protocol treats every kind identically. A bond may declare a principal kind and one kind per dated coupon; an equity may declare economic, income and voting kinds; a memecoin declares a single economic kind and an open policy.

3.3 Positions, bundles and strips

A position is the triple (asset, kind, holder) mapped to a units balance and an encumbered sub-balance. There is no separate token for "the share": holding one whole share means holding one unit of every kind flagged as part of the bundle, and

```
whole_units(holder) = min over bundled kinds of units(asset, kind, holder).
```

Transferring a share is a multi-kind transfer in one message. Transferring only the income kind creates a *strip*: the sender keeps a position with one kind fewer than a whole share, the recipient holds an income right on its own. Recombination needs no operation; a holder who reacquires the missing kind holds whole units again. This single decision removed the need for strip tokens, bundle-locking and a recombination invariant, and reduced conservation to one statement per kind.

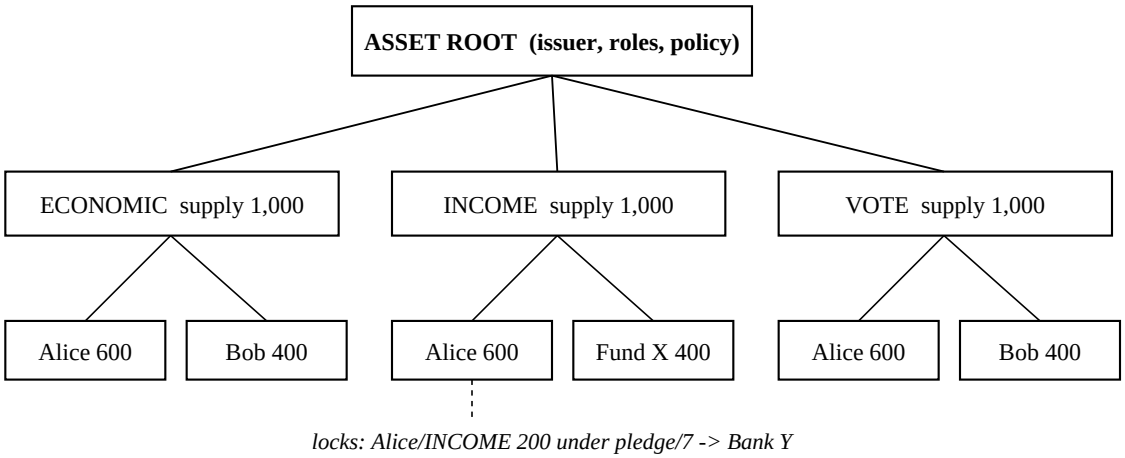

Figure 1. An asset root with three right kinds. Positions are per-kind balances; a whole share is one unit of each bundled kind; a lock encumbers part of a position in favour of a beneficiary.

4. Conservation of rights

The central invariant, checked by every validator at the end of every block, is

```
I1: for every (asset, kind): sum over holders of units(asset, kind, holder) =
supply(asset, kind).
```

Supply changes only through *Issue* and *Redeem*, both restricted to the issuer role or to a redemption pool the issuer funded, and the issuance record itself satisfies issued minus redeemed equals supply. An issuer may irreversibly lock issuance, after which supply can only fall. Explorers show "issuance locked" or "issuer can mint" as prominently as supply, because the difference is the difference between a fixed-supply instrument and an open one.

4.1 Splits

A forward stock split sets an integer multiplier on a kind. All state, including conservation, locks, settlement escrow and the income accrual index, continues to operate on raw units; queries, events emitted for presentation, the API and the ERC-20 facade report effective units (raw times multiplier), and message inputs are given in effective units and must be divisible by the multiplier. Reverse splits are not supported because they would require fractional raw units; issuers that need them redeem and reissue.

4.2 Lockups at issuance

An issuer may issue units under a lockup: the units are locked from the moment they exist, the holder may release them after the lockup date, and the issuer may waive the lockup early. This is an ordinary encumbrance (Section 7) and counts in the lock invariant like any other; it is not a special kind of unit.

5. Transfers and policy

Every operation that changes who holds units, whether a transfer, an issuance, a settlement leg, the exercise of a pledge or an ERC-20 call, is expressed as a set of *transitions* (asset, kind, from, to, units) and executed by one keeper function. The function first checks that every touched asset is ACTIVE, then applies all transitions to a cache of the state (rejecting any that exceed the sender's transferable units, that is units minus encumbered), then evaluates the effective policy for the recipient of every transition against the cached post-state, and only then writes. Any failure discards the cache. The caller of the function owns the cache, which is what lets a settlement with legs in both directions be checked as one batch.

5.1 The policy grammar

A policy is a tree of at most 32 nodes and depth at most 4 drawn from a closed grammar:

```
Policy := OPEN
| ALL[Policy, ...] all sub-rules must allow
| ANY[Policy, ...] at least one sub-rule must allow
| CREDENTIAL(type, attesters[]) recipient holds a live attestation
| ALLOWLIST(addresses[]) recipient is listed
| MAX_HOLDERS(n) post-state holder count <= n
```

The effective policy of a kind is its override if set, otherwise the asset's policy. A denial returns the path of the failing rule, for example `all[1].credential(investor.accredited)`, so that wallets and explorers can say why a transfer was refused rather than that it was. Policy changes by the issuer apply to future transitions only; existing holders are never re-validated, which is the only semantics under which tightening a policy cannot strand anyone.

Two properties of the grammar matter more than its vocabulary. Evaluation is bounded and deterministic, so a validator can price it and a counterparty can predict it. And because *MAX_HOLDERS* is checked on the post-state, a settlement in which one holder leaves as another arrives is accepted at the cap, where a pre-state check would refuse it. Jurisdiction rules, issuer-approval queues, transfer windows and lockup predicates are planned additions to the same grammar; none require a change to evaluation.

5.2 Freeze

An issuer may set an asset FROZEN. Freeze blocks issuance, transfers and new settlement proposals, and blocks nothing that releases value already owed: cancelling a settlement, releasing an encumbrance, claiming income and redeeming remain allowed. The rule is that a freeze can stop an asset from moving but can never make an escrow or a coupon unrecoverable.

6. Identity and credentials

Aeva stores no passports, names or documents. An *attestation* is the tuple (subject, type, attester, expiry, revoked, evidence hash), where the type is a short lower-case dotted string such as `investor.accredited` or `jurisdiction.de`, the attester is an account, and the evidence hash, if present, is exactly thirty-two bytes referring to material held off chain. The protocol forbids any other field; the proto definition is linted to make sure no free text can be added later. An attestation is *live* when it is not revoked and its expiry lies in the future by block time.

Attesting is permissionless. Trust is not a property of the network but of each asset: a credential rule names the attesters it accepts, so a regulated equity may trust a transfer agent's key and nothing else, while a testnet asset trusts the faucet. Only the original attester can revoke its attestation; re-attesting restores it. This keeps the network open, as the thesis requires: a permissionless asset with an OPEN policy coexists on the same ledger with a private-credit instrument that admits only accredited investors, and the difference between them is one rule on one asset.

Because attestations are public state, an indexer could trivially enumerate every account holding a given credential. Aeva's reference indexer exposes credentials per subject and per attester only, never by type; this is a product stance, stated as such, not a security boundary.

7. Encumbrances

7.1 The lock primitive

Every encumbrance is built on one primitive in the rights module: a *lock* keyed by (asset, kind, holder, reference) holding a number of units. A lock may only be created on transferable units, raises the position's encumbered sub-balance, and can end in exactly two ways: *unlock*, which returns the units to the holder as free units, or *transfer-locked*, which moves the units to another party as free units after a policy check. The invariant

```
I2: for every position: sum of locks(position) = encumbered(position) <=
units(position)
```

is checked with I1. Locked units remain in the holder's position, so I1 is unaffected by any encumbrance, and locked units continue to count the holder as a record holder for policy purposes.

7.2 Kinds of encumbrance

Four objects use the primitive: *settlement escrow* (Section 8), *pledges* created by the holder in favour of a beneficiary, *liens* created without the holder's signature by the asset's operator role and only on assets whose issuer has opted in, and *lockups* created by the issuer at issuance. Delegations (Section 9.3) are locks with an income recipient. Each carries a release rule, an exercisable flag and an optional expiry.

Encumbrance	Created by	Released by	Exercise
Pledge	holder	beneficiary always; holder after the agreed date, if any	beneficiary, if exercisable
Lien	operator role (asset must allow liens; evidence hash required)	beneficiary or operator; never the holder	beneficiary, if exercisable
Lockup	issuer, at issuance	holder after the date; issuer early (waiver)	none

Escrow	settlement proposal	cancel, decline or expiry	accept moves the units
Delegation	holder	delegate always; holder if revocable; automatic at term end	none (income routed)

7.3 Disjoint units

Encumbrances on one position never overlap: each locks its own units, and creation requires the units to be transferable. There is therefore no priority ordering between a pledge and a lien on the same position and no invariant about it. What this does not express is a subordinate claim on already-locked units; that is a claim against proceeds, which belongs to the waterfall (Section 10), not to the position. Exercise, when permitted, transfers locked units to the beneficiary through the same policy-checked path as any transfer, so an ineligible lender cannot take a restricted asset by foreclosure. A partial exercise leaves the remainder encumbered.

7.4 Expiry

Encumbrances and settlements with an expiry are indexed by (expiry, id). At the end of every block the protocol sweeps the index up to a bounded number of entries and releases what has expired; an expired object touched by a transaction before the sweep is released in that transaction. A holder never needs anyone's cooperation to recover units whose lock has run out.

8. Settlement

Delivery versus payment is a native, two-phase object. A *settlement* names a proposer, a counterparty, up to sixteen legs, an expiry and an optional thirty-two-byte reference to an off-chain confirmation. A leg moves either rights (asset, kind, units) or coins (denomination, amount) in one of the two directions.

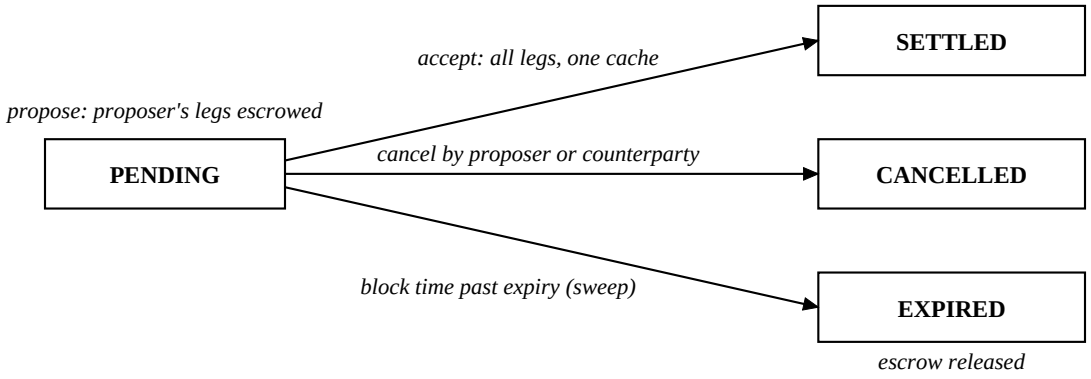

Figure 2. Settlement lifecycle. The proposer's legs are escrowed on propose; accept executes every leg of both directions in one cache and writes only if every policy check on the post-state passes.

On *propose*, the proposer's right legs are locked under an escrow reference and the proposer's coin legs move to a module account; the counterparty's side is not touched. Policy is not evaluated at proposal, since it would be stale by acceptance. On *accept*, all right legs of both directions are executed as one batch through the function of Section 5, coin legs are moved, and the result is written only if everything succeeded; otherwise nothing moves and the escrow stays in place. Either party may cancel a pending settlement (the counterparty's cancel is a decline), anyone may cancel an expired one, and the end-of-block sweep releases escrow the moment expiry passes. Cancel and expiry work while an asset is frozen. Income on escrowed units keeps accruing to the proposer until the units actually change hands.

What makes this more than an escrow contract is the combination with the rights model: a leg can be "40,000 units of the income right of this bond" against "1,200 units of the settlement denomination", the eligibility of the buyer to hold that strip is checked by the protocol at the moment of settlement, and both legs are final in the same block.

9. Income

9.1 The accrual index

Income is distributed without iterating holders. For each (asset, kind, denomination) the protocol keeps a cumulative index of income per unit, scaled by 10^{18}, and for each holder a checkpoint of the index at the holder's last change. When anyone deposits an amount `a` against a kind with supply `s`:

```
total_scaled = a * SCALE + carry_scaled
increment = total_scaled div s (rejected if zero)
carry_scaled = total_scaled mod s
index += increment
```

The remainder that integer division leaves is carried, not lost: over any sequence of distributions the sum of increments times supply equals the sum of deposits in scaled units. A holder's accrued income is units times (index minus checkpoint) divided by the scale, floored; the at-most-one-base-unit per holder per distribution that flooring leaves stays in the payout module account and is reported by a residue query rather than hidden. All arithmetic is performed in arbitrary precision and stored values are bounded; there is no floating point anywhere in the protocol, the indexer, or the reference clients.

9.2 Settle on change

Whenever a holder's units change (issuance, redemption, either side of a transfer, either side of a settlement or an exercise), the protocol first settles the holder's accrued income into a *pending* balance and then moves the checkpoint to the current index. Locking and unlocking do not change units and do not touch checkpoints. A holder who acquires units after a distribution therefore has no claim on it, and a holder who disposes of all units keeps a pending balance that survives the deletion of the position and is never forfeited. Income is claimed per asset, kind and denomination or in bounded batches; claiming zero is refused. Income accrues to the record holder, including on escrowed and pledged units, unless a route says otherwise.

9.3 Routed income

A lock may carry an income recipient. When such a lock is created, the holder is settled, and a routed checkpoint is opened for the locked units; from then on the holder accrues on units minus routed units and the recipient on the routed units, at the same index. Ending the lock settles the route into the recipient's pending balance. Two objects use this: a *delegation*, by which a holder assigns the income (and, as a record, the vote) of some units to another account for a term that reverts automatically, and a pledge whose holder agreed that income goes to the beneficiary while the pledge stands. The invariants are

```
I4: per denomination: module balance = sum over accruals of (distributed - paid)
I7: every routed checkpoint has a lock under its reference with at least its units
```

and the per-holder claimable amounts are recomputed by the indexer from the same events and compared against the chain's query for every holder.

10. Waterfalls, liquidation and redemption

A chain cannot liquidate a building. It can guarantee that whatever proceeds arrive on chain are routed according to a priority the issuer declared before the fact, and that nothing else can move while that happens. An asset's *waterfall* is an ordered list of at most sixteen tiers, each either a FIXED claim (claimant, amount, denomination) or a PRO_RATA tier over a right kind with an optional cap. It is settable while the asset is active or frozen and immutable afterwards.

Liquidate, by the issuer or the operator, moves the asset to LIQUIDATING for an exercise window: transfers, issuance, new proposals and new encumbrances stop; pending settlements are cancelled by the protocol with their escrow released; beneficiaries may still exercise and release. When the window closes the asset is LIQUIDATED, a terminal state. Anyone may deposit proceeds, and anyone may run the waterfall: FIXED tiers are paid in order up to their cumulative amounts, PRO_RATA tiers are paid through the accrual index of their kind (so holders claim their share exactly as they claim a coupon), and any remainder waits in the pool for the next run. A run costs time linear in the number of tiers, never in the number of holders. Redemption of units is not forced; units remain the claim on the pool.

Independently of liquidation, an issuer may fund a *redemption pool* for a kind at a price per unit until a date. A holder redeems by burning transferable units and receiving units times price from the pool; supply falls by the burnt amount and conservation holds. The issuer may withdraw the unspent remainder only after the date. The pool invariants (I5 for waterfalls, I6 for redemption) are of the same form as I4: balance equals funded minus paid minus withdrawn, per denomination, and are checked with the others.

11. Issuance and roles

Every power over an asset belongs to a named role, and every role is an account set by the issuer.

Role	Powers
Issuer	create the asset and its schema; issue and redeem; set and clear policies; freeze and unfreeze; set roles, the waterfall, the redemption pool and the income denominations; allow liens; lock issuance irreversibly; waive lockups; liquidate
Operator	create liens on assets that allow them, with an evidence hash; release liens; liquidate
Transfer agent	reserved for issuer-approval transfer flows; today a record
Custodian	reserved for custody attestations; today a record

Issuance is the only trust the protocol places in an issuer's honesty about the world: when an issuer issues units of a right, the chain records a claim that the issuer, not the chain, stands behind. Everything after issuance, including conservation, eligibility, escrow, income routing and priority of proceeds, is enforced by consensus. Aeva's promise is therefore precise: the ledger can never be inconsistent about itself, and what is on the ledger settles atomically; legal enforceability of a claim against the world remains a property of the issuer's wrapper and is stated as such.

12. Execution environment

Aeva provides an Ethereum Virtual Machine so that existing wallets, tooling and Solidity can use the chain, without letting the EVM define it. Accounts use secp256k1 keys of the Ethereum kind, so one key has both a native bech32 address and a 0x address; the EVM chain identifier is 2382 and the staking token has eighteen decimals so that no balance conversion is needed. Transactions submitted through the EVM pay gas in AEVA through a fee market; native transactions pay in either the staking token or the settlement stablecoin at a fixed ratio.

12.1 AE-20

Every (asset, kind) and every asset's bundle is exposed at a deterministic precompile address derived from the asset identifier. The interface is ERC-20 with extensions: `balanceOf` returns *transferable* units (so a DEX never quotes a balance the holder cannot move), while `unitsOf`, `encumberedOf` and `wholeUnitsOf` expose the rights view; `transfer` and `transferFrom` execute the function of Section 5, so a policy denial reverts with the rule path as the reason string and the same typed event is emitted as for a native transfer. The bundle token's transfer moves one unit of every bundled kind atomically. Allowances saturate at the protocol's amount bound rather than rejecting the unbounded approvals that routers send. There are no minting, burning or issuer selectors: an issuer's powers are native messages only.

Two implementation rules were learned by testing rather than assumed. Precompile actions call the batch function directly and never open a nested cache, because a write inside the EVM's snapshot store would commit every open snapshot of the transaction and break revert. And amounts and identifiers cross the boundary as exact integers; the EVM sees effective units after a split.

13. Consensus, validators and fees

13.1 Consensus

Aeva is a sovereign Layer 1. Blocks are produced by a Byzantine-fault-tolerant proof-of-stake consensus (CometBFT) among validators bonded in AEVA; a block is final when more than two thirds of bonded stake has signed it, in one round, with no probabilistic settlement and no dependence on any other chain for finality or data availability. Liveness requires more than two thirds of stake online; safety holds with any minority faulty. Validator misbehaviour (double signing, prolonged downtime) is punished by slashing bonded stake and jailing. The state machine communicates with the engine only through the application blockchain interface, which is the property that keeps the engine replaceable.

13.2 Topology

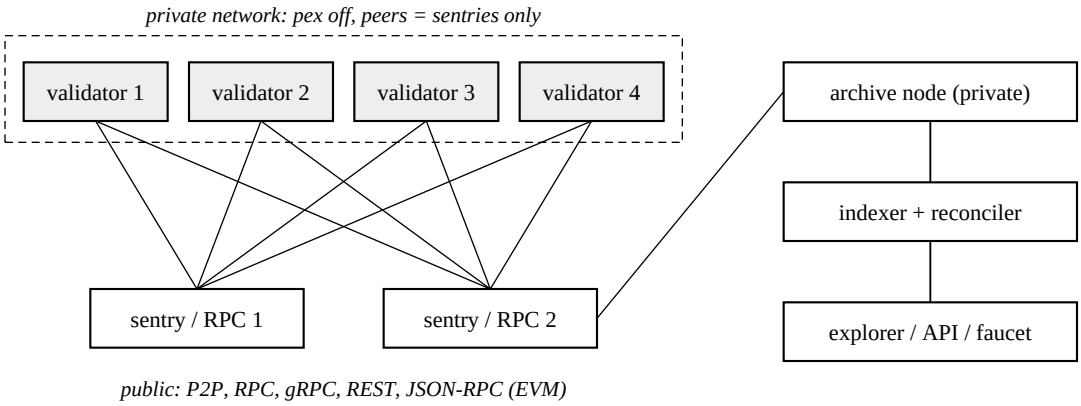

Figure 3. Validators keep peer exchange off and connect only to sentries. Sentries serve public peer-to-peer and RPC. A private archive node feeds the reconciling indexer that the explorer, API and faucet read.

Validators are never reachable from the public network. Sentry nodes take peer-to-peer traffic and serve RPC, gRPC, REST and the EVM JSON-RPC behind a reverse proxy; they also publish state-sync snapshots so that a new node can join in minutes. An archive node with full history is the only source for the indexer's reconciliation queries, which are pinned to heights. Every node runs behind an upgrade supervisor, and software upgrades are scheduled by governance vote at a height and switched automatically.

13.3 Invariants at the end of every block

The invariants of Sections 4, 7, 9 and 10 are not tests; they are executed by validators. The protocol maintains running sums per (asset, kind) and per denomination on every mutation and compares them against supply, locks and module balances at the end of each block, in time proportional to the number of asset kinds rather than holders. A full scan over all positions runs at a configurable period and on demand. A violation halts the chain: a ledger that has detected an inconsistency in its own accounting must not produce another block until humans have looked.

13.4 Fees

The unit of account for computation is AEVA, and AEVA is what secures the chain. Users, however, should not need a volatile token to move a tokenized bond. Nodes therefore accept fees in the settlement stablecoin at a governance-set ratio and validators receive fees in the denomination paid. The intended user experience is a fee quoted in cents. Whether stablecoin fees are converted into AEVA (and burned or paid to stakers) is an economic parameter, not a protocol constraint, and is left to the token design.

13.5 Measured performance

A load generator that submits a fixed mix of Aeva messages (transfers, single-kind splits, settlements, pledges and releases, income claims and ERC-20 transfers through the EVM) was run against a single tuned node on a laptop. Block time stayed at or below 1.5 seconds up to roughly 600 transactions per second (p95 latency 0.99 s at 584 tx/s), saturation was reached near 750 transactions per second, settlement acceptance committed within one block for every transaction at 100 tx/s and 84% at 500 tx/s, and read queries answered in about 10 ms at the 95th percentile. Profiling attributed 45% of CPU to two implementation costs outside the asset logic, both of which are being removed. These figures describe one laptop, not a network; production numbers will be measured on the testnet's hosts and published with their hardware. The honest claim today is: final in about a second, never reorganised, delivery versus payment in one block.

14. Observability and reconciliation

A chain that promises to be consistent about itself should be checked by something that does not share its code path. Aeva's indexer consumes the block results of any node, decodes the typed events, and derives the entire rights graph (assets, kinds, positions, locks, encumbrances, settlements, accruals, checkpoints and routed income) from events alone; replaying from genesis against the same node yields a byte-identical database. At every configurable interval it queries the chain's own invariant status and compares supply, position sums, lock sums, encumbrance sums, module balances and per-holder claimable amounts against its derived tables. A disagreement is recorded as drift, counted, alerted on, and displayed on every page of the reference explorer. It is never corrected: the remedy is to find the cause and replay. Checks are version-aware, so an invariant a running binary cannot yet report is marked not evaluated rather than drift.

The explorer is the human face of this: assets with their policy rendered in words, the rights graph with locks drawn into the bars, settlement timelines, encumbrance pages that state release authority in a sentence, distributions with residue, and an integrity strip on every page. Its arithmetic is exact (BigInt only) and its clocks are block time.

15. Interoperability

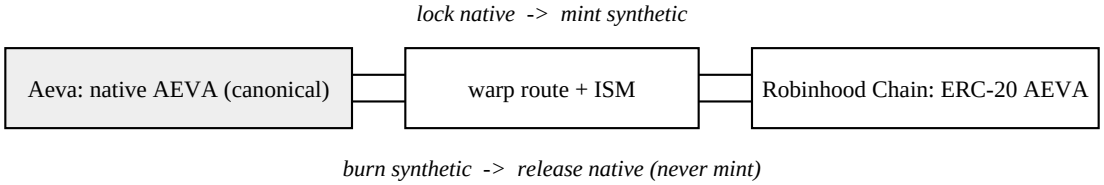

Figure 4. Native AEVA is canonical. The bridge mints only the synthetic representation on the other chain and releases, never mints, on Aeva.

Aeva's first ecosystem is Robinhood Chain. Interoperability uses a permissionless message-passing bridge with warp routes and an interchain security module whose signers are Aeva's validators. The rule that governs every route is that the native side is canonical: locking native AEVA mints its representation elsewhere; burning the representation releases native from escrow; nothing that crosses the bridge can ever mint native stake. A bridge that could mint the staking token would make the bridge, not the validators, the security of the chain. The same routes carry the settlement stablecoin in, and, subject to the

issuer's consent, Robinhood's stock tokens in as Aeva assets with rights, and Aeva strips out as ERC-20s tradable on Robinhood Chain. A yield-bearing dollar is, in this model, an asset with a principal right and an income right; the split into a fixed claim and a residual claim with a maturity is the same operation as stripping a bond.

16. Limitations and future work

Subordination. Two claims on the same units cannot be expressed as encumbrances; the junior claim must be a waterfall tier over proceeds. This is deliberate but limits some collateral structures.

Record dates. Distributions accrue to holders at the block of payment. Record-date distributions need a snapshot mechanism, planned as copy-on-write at the snapshot height.

Reverse splits and voting. Reverse splits are unsupported; voting power is exposed as a query and there is no on-chain vote execution.

Legal enforceability. The chain enforces the consistency of claims and the atomicity of their settlement. Whether a claim binds anyone off chain depends on the issuer's legal wrapper; the protocol records the hash of that wrapper and nothing more.

Decentralisation. The first public network is operated by a small validator set under one operator. It is sovereign from genesis; it becomes decentralised when validators the operator does not control hold a majority of stake. Token economics, an external audit of the execution environment, and validator onboarding are prerequisites for a production network and are not properties any milestone in this paper produces.

Throughput. The measured ceiling is that of one laptop and one tuned configuration. If production requirements exceed what the current engine delivers on real hardware, the interface boundary allows a successor engine without changing the asset model.

17. Conclusion

We have described a ledger whose native objects are assets, rights, locks, settlements and payouts rather than balances and contracts. Conservation of rights, post-state policy evaluation, bounded encumbrances, atomic two-phase settlement and constant-time income distribution are enforced by validators at the end of every block and reconciled by an independent replica; the execution environment is a guest that can read and move rights but never hold them. The design trades generality for guarantees: a closed policy grammar instead of programs, disjoint encumbrances instead of priority ordering, raw units instead of rebasing. In exchange the ledger cannot be wrong about who owns what, and it can settle what it knows in one block. Everything in this paper is implemented and tested on the first testnet; the parts that remain are the parts no protocol can supply: issuers, validators, and law.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[2] E. Buchman, J. Kwon and Z. Milosevic, "The latest gossip on BFT consensus," arXiv:1807.04938, 2018 (Tendermint / CometBFT).

[3] Cosmos SDK documentation, module system and application blockchain interface (ABCI), interchain.io.

[4] Cosmos EVM, an EVM execution layer for Cosmos SDK chains with ERC-20 precompiles, github.com/cosmos/evm.

[5] Hyperlane, permissionless interchain messaging and warp routes; hyperlane-cosmos module.

[6] Ethereum Improvement Proposal 20, "ERC-20 Token Standard," 2015.

[7] W3C, "Verifiable Credentials Data Model," for the attestation model this design simplifies.

[8] AevaChain, "Aeva protocol specification v1, architecture decision records ADR-001 to ADR-014, and event catalogue," repository documentation, 2026.