Amoeba L1

Continuously advancing state on GPU-resident markets

Amoeba Farm

November 2026

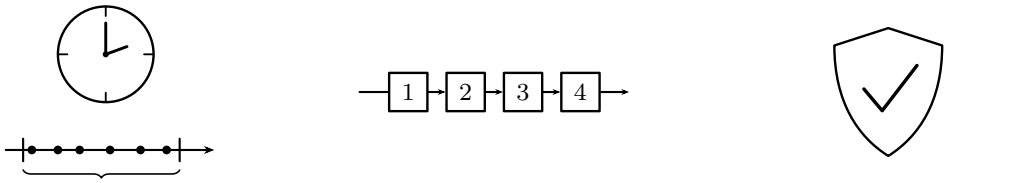
Abstract

Unlike traditional blockchains that progress in rounds of global coordination, Amoeba L1 advances state continuously, and is built for feeless, machine-speed markets. Each market is an append-only event stream with two positions tracked along it, an executed position that moves forward as events get executed, and a committed position that moves forward as validators agree, via Byzantine agreement, on the cumulative effects of prefixes of events. When validators agree that a run of events is final, they sign a fingerprint of exactly which events it contains and of the state those events produce, not just a count, so agreement pins both the history and its result. Because each market is its own stream, unrelated events proceed concurrently and committed effects are irrevocable, without any form of coordination across the entire chain. All state is in GPU memory. Packets land directly in GPU memory, and every component, authentication, matching, state commitment and Byzantine agreement, is implemented as a GPU kernel. Matching uses sorting and prefix sums to resolve contention at each price level in parallel, with the same results a serial matching engine would produce. Together these remove the three things that cap a market's speed on a block chain, the block interval no order can beat, the global sequence that makes every market wait on every other, and the matcher that handles one order at a time. By construction, a thousand orders contending at one price resolve in about ten parallel steps rather than a thousand sequential ones, and a co-located committee completes a vote round in tens of microseconds where block chains finalize in hundreds of milliseconds, so the design targets finality and throughput orders of magnitude beyond a block-based venue. Capacity is provided as regenerating allowances instead of fees for each action; stake increases an allowance, is always returned, and has no effect on where actions land in a stream. The system is therefore feeless, and instead of fees or token issuance the protocol is underwritten by profits from market making. In this paper, we present the design, the invariants it depends on, and the questions this all leaves open.

1 Introduction

A block does three things at once. It is a *scheduling* unit, because the system collects work for an interval and then processes it. It is an *ordering* unit, because everything inside a block is totally ordered and nothing inside it is ordered against anything outside it except by block number. And it is a *trust* unit, because the block is what validators vote on, and a client's result is final when the block containing it is final (Figure 1).

The three roles coincide for a historical reason rather than a technical one. The original proof-of-work design needed a bounded object to attach a lottery to, and the block was that object. The work since has loosened the coupling without removing it. Pipelined protocols [4] overlap agreement on one block with the proposal of the next. Leader-based runtimes [9] execute transactions on arrival and let votes trail behind, so execution no longer waits for agreement at all. Parallel runtimes [10, 9] execute inside a block concurrently. What none of



Scheduling
collect for an interval, then process

Ordering
one total sequence

Trust
what validators vote on

Figure 1: The three roles a block performs at once. A block chain binds them to one boundary; Amoeba L1 assigns each to a separate mechanism (Table 1).

them change is that the block is global. There is one sequence, one fork choice and one finality clock for every piece of state the chain holds, with a transaction's position fixed only when the chunk containing it is sealed. The limit of that line of work is a chunk of one transaction every few milliseconds, which is still a system in which everything shares one sequence.

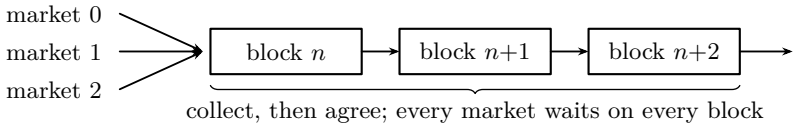
Markets for machine participants are a poor fit for that shape. The intended workload is a very large number of very small actions, such as market makers repricing ladders, agents cancelling and replacing quotes, strategies probing a book many times for every trade that prints. Nearly all of them touch one market, and most touch state that nothing else in flight touches. A global sequence nevertheless orders and finalizes each of them against everything else that happened in the same interval. A fill becomes final when the chunk containing it does, and that chunk also carries every other market's traffic, so a slow leader, a handoff or a partition anywhere delays finality everywhere. Charging a fee per action, the usual remedy for the resulting load, prices away exactly the activity the venue exists to host.

Amoeba L1 takes the three roles apart, per market. A market is an event stream that never closes. Execution advances a per-market *executed position* as events arrive. Agreement advances a per-market *committed position* by certifying cumulative prefixes of the stream, identified by history digest and state commitment rather than by length. Neither position depends on any other market. A market's ordering, execution and finality proceed on their own, and no chain-wide interval, leader rotation or fork choice sits between them. Scheduling boundaries exist (kernel leases, execution windows, packet batches) and are constrained so that how a stream is cut into them cannot change any result. Figure 2 draws the contrast.

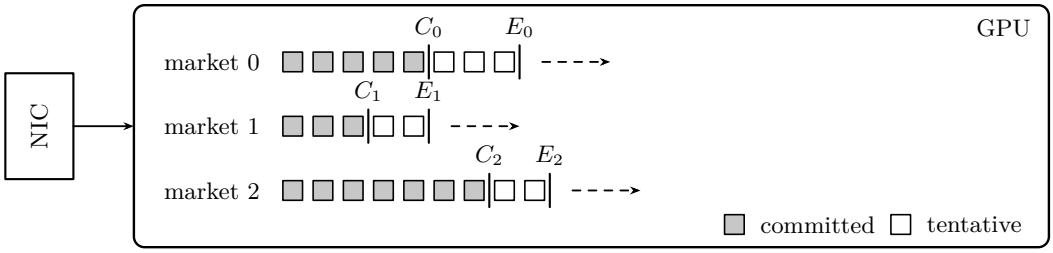
The GPU is what makes this affordable. Packets are delivered into device memory and consumed by device code; market state, authentication, matching, commitments and the agreement protocol itself run as kernels. The stream is still a queue, in which every event has a canonical position and priority within a price level is first-in, first-out. But it is not consumed one element at a time. A window of the queue is held in device memory as an array, and the fills a serial matcher would produce by walking it are computed at once by a sort and a prefix sum, with identical results. The serial dependency among contending orders is a property of the walk, not of the queue. Suppose the best ask holds resting orders of 5, 5 and 10, and Joe and then Jack hit it for 8 and 9. Joe fills first, as he must. But computing the fills does not require handling Joe before Jack. Cumulative supply (0, 5, 10, 20) and cumulative demand (0, 8, 17) are two prefix sums, and each taker's fills are the overlap of his demand interval with the supply intervals (Figure 3). The answer is the serial matcher's answer, and no part of it was computed one order at a time.

Feelessness forces a second decomposition. A fee does three jobs too. It meters capacity, it pays the operators, and it is the hidden price of priority. Amoeba L1 gives the metering job to a per-account allowance, with a floor for everyone and more for those who stake, the payment job to revenue the network earns by making markets, and refuses the third job

(a) Block chain. One sequence for every market; a fill is final when its block is.



(b) Amoeba L1. One stream per market; each advances and finalizes on its own.



Path of one order: (1) the packet lands in device memory; (2) it is authenticated and executed in stream order, E_m advances and a tentative receipt is returned; (3) validators certify the prefix, C_m advances, and the fill is final. Step (3) for market 1 never waits on step (2) for market 0.

Figure 2: Where an order waits. In a block chain (a) every market's traffic is sequenced and finalized together. In Amoeba L1 (b) each market is its own stream in device memory with an executed position E_m and a committed position C_m; the positions of one market never wait on another.

outright, so that nothing a participant holds or pays can change where its action lands in a stream.

The contributions of this paper are: a decomposition of the block into roles and an assignment of each role to a mechanism that does not require a global boundary (§4); a GPU-native architecture in which no CPU sits on the transaction path (§5); a formulation of order matching within a window as a parallel computation that is byte-for-byte equivalent to the serial matcher (§5.4); a stake model in which one staked asset meters capacity, secures agreement and settlement, and is paid from market making alone (§6). No performance result is claimed.

2 Background and related work

Byzantine agreement. PBFT [1] established practical state-machine replication under Byzantine faults with a three-phase protocol. Tendermint [3] brought a rotating-leader variant to blockchains; HotStuff [4] reduced communication to linear by routing votes through the leader and introduced the lock-and-prepare-certificate structure that Amoeba L1's agreement core adopts. All three agree on blocks. DAG-based protocols such as Narwhal and Tusk [5] and Mysticeti [6] decouple data dissemination from ordering and let many validators propose concurrently, but still produce a totally ordered global sequence. FastPay [7] and Pod [8] drop total order for payments and for "generalized consensus" respectively, showing that many transactions need only a weaker guarantee than a global sequence; Amoeba L1 applies a similar observation at the granularity of a market stream rather than a single account, because an order book is exactly the structure that does need a total order internally and does not need one externally. Speculative execution ahead of agreement goes back to Zyzzyva [2]; the tentative-receipt discipline in §4.2 is in that lineage.

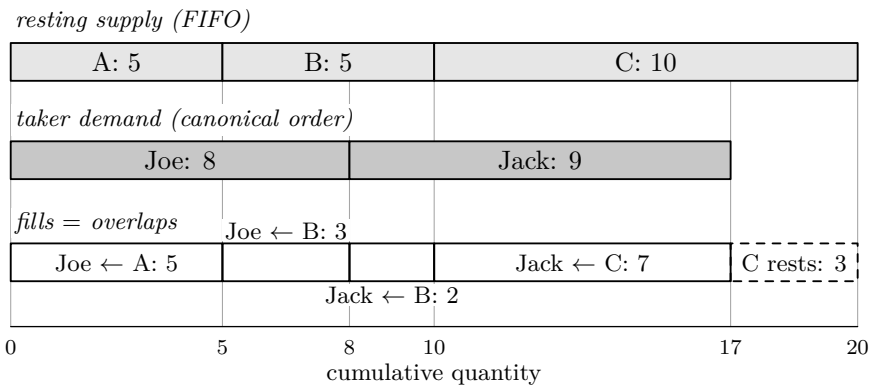


Figure 3: Contention as a scan. Resting supply at a level and taker demand in canonical order are each laid out by cumulative quantity; a taker's fills are the overlaps of its interval with the supply intervals. Joe precedes Jack in every segment, exactly as a serial walk would have it, but the three rows are two prefix sums and one intersection, computed at once.

Parallel execution. Solana's runtime [9] and Block-STM [10] parallelize execution inside a block, the former by declared access sets and the latter by optimistic concurrency with validation. Both keep the block as scheduling, ordering and trust unit. Solana's commitment levels (processed, confirmed, finalized) already separate executed state from agreed state; the separation here is made per market rather than per slot, and is bound to a state commitment rather than to a position in a global fork. GPU execution of blockchain programs has been explored for the EVM [11], chiefly for fuzzing and throughput experiments on a conventional block model. Amoeba L1's execution model is narrower, a fixed set of market operations rather than a general virtual machine, and that narrowness is what allows the dependency structure to be analyzed exactly.

Market design. The order book is a continuous double auction with price-time priority. Budish, Cramton and Shim [12] argue for frequent batch auctions, in which priority within a short interval is by price alone, as a response to latency arms races. Amoeba L1 keeps continuous price-time priority as the baseline and treats the batch auction as an open alternative (§7); the two differ in what is being decided, not in whether a window of orders is held at once. Co-located validator sets for exchange chains have been pioneered in production by Hyperliquid [13], whose design informs the deployment assumption here, as does its protocol-operated liquidity vault; Amoeba L1 differs in funding the whole network from that vault and in charging no fees around it.

GPU data paths and primitives. NVIDIA's GPUDirect RDMA and DOCA GPUNetIO [14] let a NIC write into GPU memory and let device code drive the NIC's queues without host involvement. Prefix sums and segmented scans [15] are the primitives the matching formulation in §5.4 reduces to. The design philosophy owes something to hardware–algorithm co-design in machine-learning systems [16], which organizes memory access and work around the accelerator while leaving the application semantics untouched. Figure 4 shows the resulting path.

Post-quantum signatures. Falcon [17] is a lattice-based signature scheme with compact signatures and fast verification; the deterministic-signing variant used here follows an existing implementation [18]. Ed25519 [19, 20] supplies small classical signatures for routine trading.

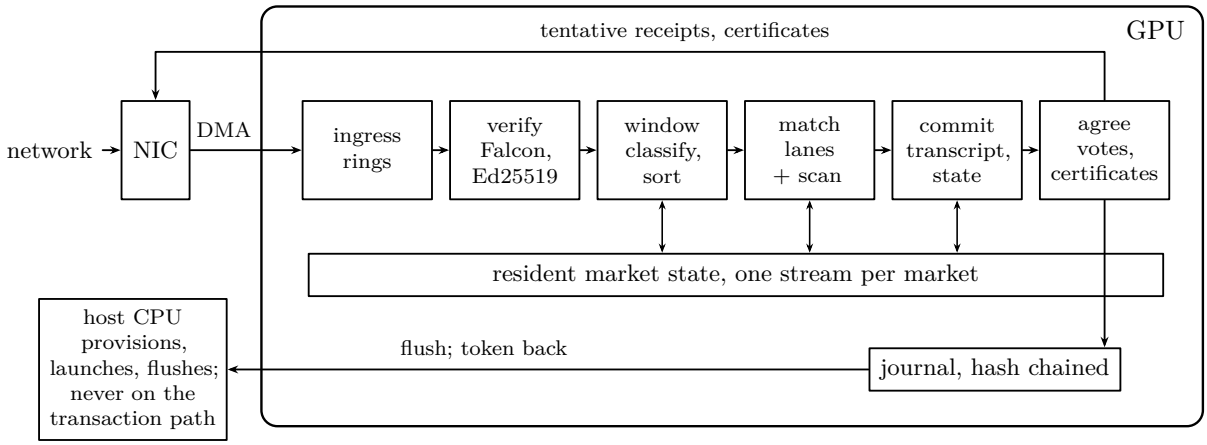


Figure 4: The GPU-native data path. Packets are delivered into device memory and every stage from authentication to agreement is a kernel over resident state. The host provisions, launches and flushes journals; it never ingests, matches, signs or votes.

3 Model and goals

Participants. A committee of validators operates a set of markets. Each validator's weight is the stake delegated to it in the network's native asset (§6); weights are positive and fixed within a configuration. Clients hold accounts and submit authenticated requests. The initial deployment places the validators in one facility; remote nodes keep asynchronous archives and do not vote.

Faults. Validators may crash and restart, lose or reorder messages, and behave arbitrarily, provided the total Byzantine stake is strictly less than one third of the configured weight. The network is asynchronous with eventual delivery; liveness is argued only after communication recovers.

Instruments. Markets trade capped futures, positions whose maximum loss is fixed at order time, with non-delivery settlement and no liquidations. Positions are denominated and settled in a cash unit held in accounts; the native asset is never the payoff unit of an instrument. This is a product choice with architectural consequences (§4.7).

Goals.

1. **No global boundary.** A market's progress, in execution and in finality, does not wait on any other market or on a chain-wide interval.
2. **Exactness.** Every validator computes identical results from identical event sequences, by exact integer arithmetic, defined overflow behavior, canonical serialization, deterministic priority.
3. **Scheduling invariance.** Kernel leases, execution windows and packet batches may not change any result. An isolated ready event is never made to wait for company.
4. **GPU-resident data path.** No CPU ingests, decodes, orders, matches, signs or votes on the transaction path.
5. **Feeless.** No per-action monetary fee. Capacity is allocated as a resource, not sold as a toll.
6. **Explicit finality.** Clients can distinguish tentative, committed and invalidated outcomes, and verify finality themselves.

7. **Stake, not volume.** Influence over the protocol (agreement weight, settlement weight, capacity) derives from stake and from nothing else, never from trading volume, profit, or payments made to the network. Stake never affects the order of events.

4 Blockless execution and agreement

4.1 The market stream

Definition 1 (Market stream)**.** A market m is an append-only sequence of authenticated events $e_1, e_2, \ldots$ with a canonical total order. Two positions are tracked along it, the executed position E_m, the index of the last event whose effects are present in the market's working state, and the committed position $C_m \leq E_m$, the index of the last event covered by a valid decision certificate.

Events after C_m are tentative; events at or before C_m are irreversible. Validators may be at different executed positions, but must agree on the committed history, which means not only the value of C_m but the bytes behind it. Table 1 summarizes where each of the block's roles goes.

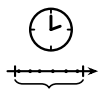
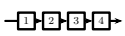
Role	Block chain	Amoeba L1
 Scheduling	The block interval; work waits for the next block.	Execution windows and kernel leases (§4.5). How a stream is cut into them cannot change a result.
 Ordering	Total order within a block, block number across blocks.	Canonical position in the market stream. Markets are ordered against each other only through explicit cross-market events.
 Trust	The block; final when its block is final.	The certified prefix, a cumulative certificate naming a position *and* the history digest and state commitment at that position (§4.3).

Table 1: The block's three roles and their replacements.

What is removed is the chain-wide, leader-paced boundary that sequences and finalizes every market at once and forces trust, ordering and scheduling to coincide. What remains is a per-market certified prefix as the trust unit, and bounded windows as scheduling units without semantic weight.

4.2 Continuous execution

Execution for market m is a deterministic function from a stream prefix $e_1..e_k$ to a state S_k. The validator currently leading the market (§4.3) proposes the order of newly arrived events, appends them to the stream, executes them, and streams signed *tentative receipts* to clients. Followers execute the same events as they obtain them. Nothing waits for a vote.

Two properties make execution ahead of agreement safe.

Execution is total and deterministic. Every event has a result, including rejection. A rejected event still occupies its stream position and advances the transcript digest, so a recorded rejection is distinguishable from an event that was never received. Arithmetic is

exact, and an event either applies whole or is rejected whole. Device completion order never decides priority; a market's events execute in stream order.

Tentative is labelled tentative. A tentative receipt is signed under its own domain, distinct from a decision certificate. The client interface exposes three outcomes, tentative, committed and invalidated, so that a client acting on a tentative result is taking a risk the protocol has made visible. Tentative proceeds cannot be withdrawn or bridged as committed value, and a dependent action may be admitted only if it names the same branch and every obligation it can reach stays funded under the actual payoff rules.

The intended bound on speculation is $E_m - C_m \leq W_m$ for a per-market window W_m. The value of W_m is an open choice.

Remark. A tension remains between publishing fills tentatively and publishing them only once their window is certified. The former minimizes latency and exposes clients to reverted fills; the latter costs a vote round and makes window cadence the latency floor (§4.5). Which a market runs is recorded as open (§7).

4.3 Continuous agreement

Agreement runs separately for each market, over commitments to its ordered history and resulting state. There is no global trading block and no mandatory event batch; a proposal covers whatever prefix has executed since the last one, including a prefix of length zero.

Cumulative certification. A certificate for market m at position k is not a signed integer. It binds the genesis configuration, the market, the protocol phase, the view, the parent certified node, the position k, the digest of the authenticated history $e_1..e_k$, and the commitment to the executed state S_k. Two validators that sign the same certificate have executed the same bytes and reached the same state; a validator cannot vote for a position it has not independently executed, and availability is never inferred from a hash. Because certificates are cumulative, one vote round covers every event since the last certified node, so vote cost is amortized across events without forcing events into a batch.

Phases, locks and safety. The core is Basic HotStuff [4] with three explicit adaptations: stake-weighted quorums instead of node counts; certificates built from complete individual signatures instead of threshold signatures; and per-market prefix nodes instead of blocks. Phases are prepare, pre-commit, commit and decide. An honest replica votes at most once per phase and view, requires the preceding phase before advancing, and locks before releasing its commit vote. Its prepare rule accepts a proposal only if the proposal extends the lock or carries a strictly higher prepare certificate, and additionally preserves already-committed ancestry.

With total stake W, a quorum is any set of validators whose weight exceeds $2W/3$. Two quorums intersect in weight strictly greater than $W/3$; with Byzantine weight strictly below $W/3$ the intersection contains an honest validator, which is what HotStuff's conflicting-prepare argument requires. The adaptations carry their own obligations. Stake intersection replaces node-count intersection, and the linear communication bound that threshold signatures give is not claimed. The weight vector a certificate is formed under is itself named in the certificate, by a certified position in the stake stream (§6.3); a certificate is checked against the weights it names, not against whatever weights hold when it is read.

Quiet markets. A market that goes quiet must finalize its last event without waiting for a customer to trade. A zero-event child node carries a prepared history into a later view, so a pending prefix completes its phases on protocol machinery alone.

Leader replacement. Each market has a fixed rotating leader schedule. A view ends by decision or by a quorum-certified timeout, and waiting periods grow after failed views. This pacemaker is protocol machinery; it is not an order expiry and imposes no trading-rate restriction.

Durability boundary. The selected boundary is local durability with asynchronous remote archives. Votes, locks and histories survive validator restarts because protocol records are journaled and flushed to stable storage before the vote or acknowledgement they gate is published, and recovery replays the journal on the device rather than loading a host-selected snapshot. Archives are asynchronous copies of a known flushed prefix; they do not vote. The consequence is that loss of the entire site can lose acknowledged history that had not yet reached an archive. Requiring off-site durability before acknowledgement would place a remote round trip on the finality path; the boundary is a deployment choice and is reversible.

4.4 Transactions as tensors

A transaction is a fixed-width row, and a run of n transactions is an n-row array. There is deliberately no timestamp field, because a client's clock must never set priority. The row is the unit the network delivers into device memory and the unit the kernels read; there is no deserialization between them. The fixed width is what allows a window of the queue to be processed as an array rather than walked element by element, classified by field, sorted by position, grouped by price level, with the queue's order preserved throughout. None of this implies floating-point arithmetic or learned components; financial state is exact integers, and "tensor" names the layout and the operations (scans, reductions, sorts, segmented sums), not approximation.

4.5 The window as the unit of ordered execution

Execution proceeds in *windows*. A window is an already-available, canonically ordered span of the stream that is held and processed together. The natural window is the span between two proposals.

A window has a boundary, and in that respect it resembles a block. It differs in what the boundary is permitted to mean. It is not chain-wide and not leader-paced; it certifies nothing by itself; how a stream is cut into windows must not change any result; and an isolated ready event is never made to wait for others to fill a window. An implementation may bound a window to fit its memory; it may not introduce a collection timer. The same rule covers kernel leases, where a driver bounds how long a kernel may stay resident; state outlives the kernel, and results are invariant across relaunches. A boundary the hardware imposes may exist only if it cannot be observed in the result.

Windows exist because the parallel matching of §5.4 works by holding contending orders at once. With a co-located committee at round-trip times on the order of 10 μs, a three-delay vote round is tens of microseconds, and certifying a window every few hundred microseconds is a plausible cadence. If fills are published only on certification, that cadence is the latency floor.

4.6 Contention is not serialization

A sequential matcher processes orders one at a time, so the second order contending for a level appears to depend on the first. The dependency is real; the serialization is a property of the algorithm. With a window in hand, contention at a level resolves as a sort by canonical position followed by a prefix sum of cumulative taker demand against FIFO-ordered resting

supply. Hundreds of orders contending for the best ask become a sort and a segmented scan of logarithmic depth. §5.4 gives the formulation; the consequence for the protocol is that the dependency structure of a book does not force a serial critical path through a window, which is what would otherwise make per-market agreement no better than a block.

4.7 Capped positions make collateral a counter

In a margined exchange the hidden serializer is not the book but collateral, because margin checks and liquidations couple accounts across markets and force ordered state reads. A capped future has its maximum loss fixed at order time, no margin and no liquidations, so collateral reduces to a per-account reservation that decrements atomically on accept and increments on cancel or release. There is no mark-to-market sweep over every account and no liquidation cascade with a required order. This is a substantial part of why device-resident execution is plausible for this instrument.

4.8 Feeless admission

There are no per-action monetary fees. Frequent quote replacement, tiny interactions and machine probing are the intended workload. Admission must therefore be solved as resource allocation, and the mechanism is a regenerating per-account allowance with a floor that every account receives and a rate that staked capital raises (§6.6). Two constraints bind any such mechanism. No admission rule may change the priority of an already-accepted fill, and a cancellation path cannot overtake a fill it would otherwise follow.

4.9 Authentication

Every account has a Falcon-512 root key [17], used in deterministic mode [18]. Roots control accounts and sign validator votes, the places where post-quantum security matters most and signing is infrequent. A root grants an Ed25519 [19] session scoped to one market; sessions place and cancel orders with small classical signatures and consecutive nonces. Grant and revocation are ordered events in the market, so the cutoff is a stream position rather than a clock, and replay state follows the same tentative and committed branch rules as market state.

The chain verifies both keys, each where it matters. A grant is a Falcon-signed event that records which session key may act for which account in which market and until which position; every order is then checked against that record, and a revocation is a Falcon-signed event after which the session's orders are rejected. The split exists because a Falcon signature is roughly ten times the size of an Ed25519 signature and far slower to produce, which is immaterial for the few events a root signs and prohibitive for the millions a session signs. A compromised session key can therefore trade, but only in its one market, only until its cutoff, only within the collateral the account has reserved there, and only until the root revokes it; it cannot withdraw, grant, or vote.

Cumulative certification spreads one Falcon vote over many events, and session keys make the per-event cost an Ed25519 verification, which batches well on the device (§5.5). Ordinary orders are classically authenticated; full post-quantum protection for trading is not claimed.

4.10 What "no block" does and does not mean

It *does* mean no chain-wide boundary (one market finalizes while another is partitioned or its storage acknowledgements are withheld); no leader-paced interval that a ready event waits for; and a trust unit that is a certified cumulative prefix identified by history digest and state commitment, never by length alone. It *does not* mean the absence of boundaries.

Windows, leases and packet batches exist, under the rule that they are scheduling and cannot change results. Faster replay of a prefix is not overlapping consensus, and execution ahead of certification is not finality; these are reported separately.

5 GPU-native architecture

5.1 The data path

The steady-state path is

$$\text{network} \rightarrow \text{NIC} \rightarrow \text{device memory} \rightarrow \text{resident kernels} \rightarrow \text{response},$$

with market state, authentication, ordering, matching, commitments, agreement and journal generation performed by device code. The host provisions devices and keys, allocates memory, launches kernels, reads diagnostic counters and, under the durability contract, flushes opaque device-produced journal records to disk. It does not ingest requests, decode events, choose histories, match, sign, count votes or decide finality. There is no CPU request route and no CPU protocol fallback. A CPU on the path is a serialization point with its own copy and its own scheduler, and every such point is a place where a block tends to grow back.

5.2 Ingress

Packets land in device memory and are consumed by device code through DOCA GPUNetIO [14]. Kernels reserve transmit queue slots, prepare descriptors, ring the doorbell from the streaming multiprocessor, poll completions and read received packets by address. Fragmentation and retransmission run on the device through the same queues.

Three capabilities are kept separate because a demonstration of one is routinely mistaken for the others: (1) the NIC writes directly into device memory; (2) device code initiates and controls the network operations; (3) the complete application path meets a latency and throughput target. The design requires all three (Figure 5).

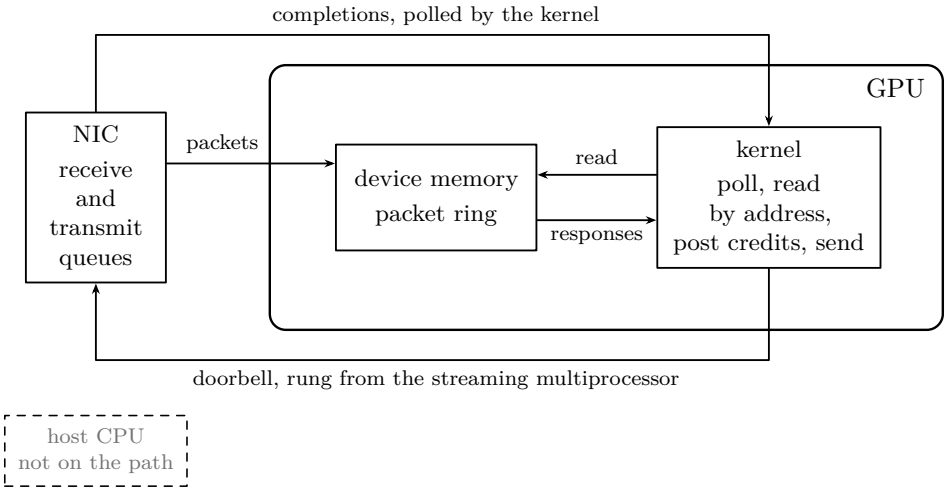


Figure 5: GPU-initiated networking. The NIC writes packets into a ring in device memory; a resident kernel reads them by address, rings the NIC's doorbell from the streaming multiprocessor and polls completions. No host code touches a packet.

5.3 State layout

Market state lives in device memory and outlives kernel launches. The book is a fixed price domain of levels, each side holding FIFO-ordered resting orders; the device-natural representation of a level is an append-only array with cancellation marks, so that a level's resting supply is a prefix sum and a cancel is a single write. A window of transaction rows (§4.4) is classified by field, sorted by canonical position and grouped by level without pointer chasing, and results are written as fixed-width outcome and fill records so the transcript can be reconstructed in canonical order.

5.4 Intra-market parallelism

A venue with few markets gains almost nothing from running them in parallel; useful parallelism must come from inside one book. The serial matcher is the correctness reference, not a lower bound.

Where the width is. The question is which events in a window depend on one another. Partition the window by price level. Most messages on a real venue are makers moving resting quotes, tens of them for every trade; if no order in the window trades at a level, the adds and cancels there affect nothing else and can run in parallel, with only a short in-order chain inside the level. The exception is a marketable order, which trades through the top levels of the opposing side; its fills depend on every add and cancel at those levels that precede it, and those events form the *contested set*. Collateral adds no further dependencies, since a capped position's reservation is a per-account counter (§4.7) and never links two levels. Counted by message, the contested set is small; weighed by money, it is everything, because it is where the trades are. The device therefore has two jobs: clear the independent majority in parallel so trades are not queued behind it, and then make the contested set itself parallel (Figure 3).

Classification. A conservative classifier marks every level a possible crosser could reach, which for each marketable order is the full crossing interval from the best opposing level to its limit, plus its own level, to cover liquidity introduced in the same window. This admits false positives in exchange for a simple proof.

Uncontested lane. One worker owns each touched uncontested level and walks its events in canonical order, exactly as the serial matcher would. Workers stage their changes privately and publish them once at a barrier, so no two workers ever write the same state.

Contested lane as a scan. After classification:

1. Takers are the possible crossers in canonical order. For each opposing level, resting supply is the FIFO-ordered sequence of remaining quantities with its exclusive prefix sum.
2. Opposing levels are walked from best to worst over the fixed price domain. At each level, the cumulative demand of the takers whose limits reach it, in canonical order, is a prefix sum. A taker's fills at the level are the resting orders whose supply intervals overlap its demand interval.
3. Fill order within a level follows FIFO position; a taker's level sequence follows price. The serial matcher's ordered fill sequence is therefore reproducible from interval arithmetic.

Cases the scan does not cover, such as a rejection discovered partway through a fill, segment the window, and the serial lane handles that segment.

Equivalence obligation. Reordering physical computation is not changing economic priority, and the test is exact. Every per-event result, the transcript digest, the state commitment and the resulting certified node must match the serial path byte for byte.

Remark. Device execution buys throughput under load, not latency at low volume. A CPU matches an isolated order in about a microsecond; a kernel launch costs more. The end-to-end latency target lives in agreement and inter-node round trips; this section concerns the regime in which arrival rate exceeds a serial matcher and queues grow.

5.5 Authentication and agreement on the device

Both signature schemes of §4.9 verify as kernels, a warp per Falcon signature and a thread per Ed25519 signature, so verification over a window is trivially parallel. The ordered checks against grants, revocations and nonces run after it and are unchanged by it.

Each validator for each market is a device actor with its own key, market state, queues, votes, locks and journal. Actors exchange proposal, vote, certificate, timeout and new-view messages through the network queues; an actor cannot read another actor's state. A client verifies prepare and commit proofs itself and accepts a final result only through its own replay of the certified history; it does not trust a validator-supplied flag. Protocol records form a device-generated hash chain, and votes and acknowledgements are gated on confirmation that the records they depend on have reached stable storage.

5.6 Commitments

A certificate binds two digests that answer two different questions. The *transcript digest* answers what happened. It chains every event with its outcome and ordered fills, so two validators with the same digest have processed the same bytes in the same order. The *state commitment* answers where the market now stands. It is a hash of the logical state in a canonical order, balances by account and then each side's levels by price with their FIFO records, and it excludes physical layout, so equivalent logical states hash identically however they are stored. Binding both is what makes a certificate a statement about bytes and about state at once; a transcript alone would let two validators agree on history while silently diverging in state, and a state alone would not say how it was reached. The signed protocol path additionally binds genesis configuration and authority state. Inclusion proofs for individual events or balances would need a tree structure over these commitments, which is left open.

6 Stake and economics

This section is stated because the rest of the design constrains it, and because a feeless network has to say where its capacity, its security and its money come from or it has not said anything.

6.1 Principles

Five commitments fix the shape of everything below.

1. **Feeless stays feeless.** No action costs money. Stake is returned in full; nothing is burned, and nothing is charged.
2. **No issuance; revenue buys the asset.** The native asset has a fixed supply and the network never mints it. Market-making revenue owed to stakers is used to buy the asset on the network's own books, and the asset bought is distributed to them by weight.

Every distribution to a stakeholder is therefore a share of revenue the network earned by making markets, paid in the asset rather than in cash, and nothing is burned.

3. **Stake buys participation, never position.** Stake decides whether a participant may act, how much weight its vote or its data carries, and what share of revenue it receives. It never decides where an action lands in a stream, and it is invisible to ordering and matching.
4. **One asset, delegated by choice.** There is one staked asset. A holder points it at a validator, at an oracle source, or at the market-making vault. Each target has its own fault domain; stake exposed to one is not exposed to another.
5. **The operator's share only falls.** The protocol's own share of revenue is a parameter that can be lowered by an ordered event and cannot be raised. The network is meant to be hatched and then left alone.

6.2 The stake stream

Stake lives in a designated stream that carries staking, unstaking, delegation, slashing and distribution events. It is certified by the same machinery as a market (§4.3) and trades nothing. Staking and delegation are ordered events signed by the account's root key, so a delegation's cutoff is a position in the stream rather than a clock; unstaking returns the asset after a fixed period during which it remains slashable for faults committed while delegated. Markets reference the stake stream one way. A certificate names the stake-stream position whose certified state supplies its weight vector, and admission reads staked balances from a certified snapshot. Nothing in a market can change the stake stream except a slash proof, which is itself an ordinary stake-stream event.

6.3 Validators

A validator's weight is the stake delegated to it, its own included, and quorums are computed over these weights as in §4.3.

The weight vector changes only at *configuration boundaries*. A configuration is a certified stake-stream position; a market enters a new configuration by a quorum-certified event in its own stream, and every certificate names the configuration it was formed under. A certificate can therefore be verified by anyone holding the two certified prefixes it names, and a weight change can never retroactively alter which past certificates were valid. The safety argument of §4.3 applies within each configuration; the hand-off between configurations is listed as open (§7).

Only provable faults are slashed, meaning a validator key signing two conflicting votes for the same phase and view, or two certificates for the same position with different histories or states. The proof is the pair of signed messages, submitted by anyone; it removes a fixed fraction of the stake delegated to that validator at the time of the fault, and the slashed amount is held rather than burned. Faults that cannot be proved from signatures, such as silence or slowness, are handled by the pacemaker and by delegators moving their stake.

The committee is fixed at genesis and co-located. It opens, at a configuration boundary, to any candidate whose delegated stake exceeds a threshold and who meets the co-location requirement, and a member leaves when its stake falls below that threshold or on a slash. Until the committee opens, stake delegated to the genesis validators secures a committee the operator runs, and this paper does not describe that as decentralized security; what it provides from the first day is a public, slashable, delegable stake behind every certificate, and a path with a number on it. Validators and their delegators are paid the validator share of market-making revenue, converted to the asset by buyback and split by weight. There is no

issuance, so early validator income is small, and there are no fees to collect. Figure 6 draws the arrangement.

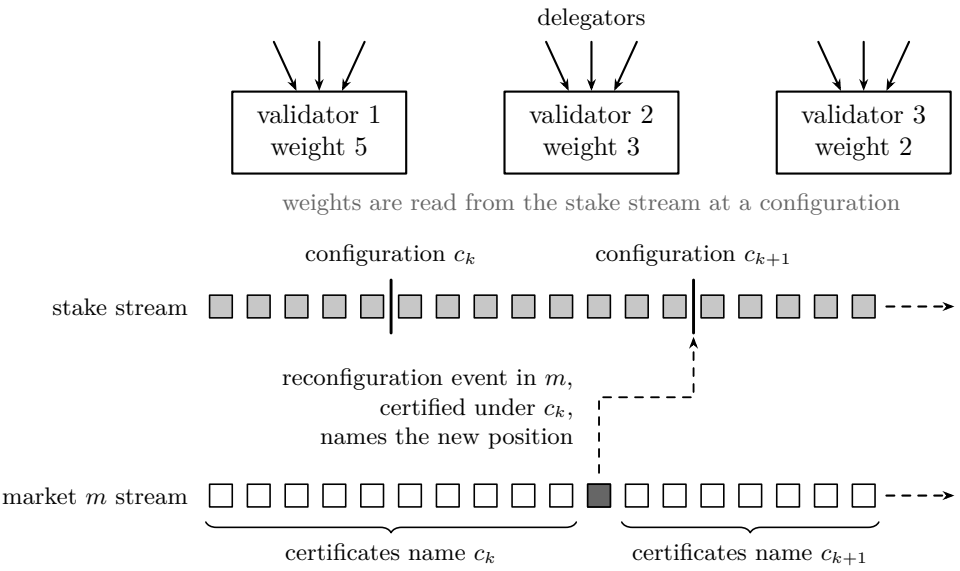


Figure 6: Validator weight and configurations. Delegated stake sets each validator's weight. Weights change only at configuration boundaries, which are certified positions of the stake stream. A market moves to a new configuration by an event in its own stream, certified under the old configuration, and every certificate names the configuration it was formed under, so a weight change never alters which past certificates were valid.

6.4 Settlement and the oracle

A capped future settles against a reference value. The reference enters a market as an ordered *settlement event*, so settlement is part of the certified history and a client verifies it the way it verifies a fill.

The oracle is staked with the same asset and the same primitive as validation, pointed at a different target. A staker names a *source*, a specification of what to observe rather than a URL, and delegates stake to it. The delegation is at once the source's weight in a weighted median that produces the reference value and the bounty that pays whoever fetches the observation; fetchers hold no stake, and delegators bear the slash if their source is successfully challenged. The index definition itself, which purchase each component prices and how components are weighted, is frozen by the operator and is not something stake can vote on. The oracle's challenge rules and observation formats are specified separately; what the chain fixes is that settlement weight is stake, that it is slashable on the stake stream, and that it is never derived from trading volume or profit.

6.5 Market making and revenue

The network earns in exactly one way. It makes markets. A protocol-operated vault quotes every book. Its capital has two sources, depositors who supply the cash unit and take proportional profit and loss, and stakers who delegate to the vault and back written positions with their staked asset, under a utilization cap and without the asset being sold. Because a capped position's maximum loss is known at order time (§4.7), the reservation against staked capital is a counter like any other, and no liquidation of staked capital can ever be required.

Revenue is distributed by ordered events on the stake stream in fixed shares to five recipients: cash depositors, stakers backing writes, oracle delegators, validators with their delegators, and the operator. Depositors are paid in the cash unit. The three staker shares are used to buy the asset on the network's own books and the asset bought is distributed by weight, so stakers are paid in what they staked and the buyback is itself ordinary trading on a certified stream. The operator's share is the one parameter in this paper that moves in only one direction. An ordered event may lower it and nothing may raise it. The vault's strategy, like everything else here, is to be published; a copy of the venue running elsewhere with the operator share set to zero is the intended end state, not a failure mode.

6.6 Admission as staked capacity

This is where the asset does its daily work, and where the two constraints of §4.8 bind.

Every account has an allowance that regenerates at a rate with a nonzero floor and rises with the stake the account holds. The floor is deliberate. An account that stakes nothing can trade, at a rate the network can sustain for everyone at once. Staking raises the rate and the ceiling and does nothing else. Allowance is metered at ingress, on the device, before an action receives a stream position, and nothing is ever charged in money.

The three traffic classes of §4.8 meet the allowance differently. A replaceable quote that is superseded before it becomes executable costs nothing, because it consumed no stream position and no state; a quote that reaches the stream costs one unit. Persistent state is a reservation, not a charge. Resting orders and accounts are bounded per account, with a floor that requires no stake, and a reservation is released on fill or cancel. Invalid traffic names no account that can be trusted, so it is never charged to the account it names; it is counted against the delivering endpoint and never enters a stream.

Refusal is not rejection. An action refused at admission never receives a stream position and never appears in the transcript; the client learns of it from the ingress response. An admitted action the market rejects does occupy a position (§4.2). Because admission acts strictly before position assignment, it cannot change the priority of anything already in the stream, and a cancel either enters the stream in canonical order or does not enter at all; it cannot be promoted past a fill by having more stake behind it. Under overload the network refuses at ingress in proportion to allowance. Staked participants keep their configured rate, unstaked participants keep the floor, no one is charged more and no one jumps the queue.

What stake does not buy. It does not buy ordering, which is fixed by canonical position; it does not buy a fee rebate, because there is no fee; it does not buy a vote on an index definition; and it does not buy exemption from any rule that applies to an unstaked account. Capacity is the whole of what it buys, and capacity is given back.

6.7 The asset

AMBA is the chain's native asset. It is not the cash unit in which positions are denominated and settled; a capped future's payoff is never in AMBA, so the asset's price does not enter any market's risk. Holdings on the prior venue migrate by a genesis claim executed at the root-key claim handshake (§4.9), the same event that binds a legacy key to a Falcon root credits the account with its claimed balance. There is no bridge on the transaction path.

7 Open problems

1. **Publish-tentative versus certify-then-publish.** Whether fills reach clients before their window is certified. A product decision that sets the latency floor.

2. **Window cadence as agreement timing.** Certifying every few hundred microseconds on a co-located committee is plausible but is a change to agreement timing that the rest of the design does not by itself entail.
3. **Canonical time.** With validators observing different arrival orders, some receive timestamp or deterministic tie-break must define canonical position within a window, and whoever supplies it can reorder within the window. Feeless capped futures shrink what is extractable to queue position; the rule must still be explicit, and it must be shown that stake cannot influence it.
4. **The speculative bound** W_m and cross-market speculative credit.
5. **Admission parameters.** The mechanism of §6.6 is fixed; its rates, ceilings and reservation limits are not, nor is the endpoint throttle for invalid traffic, nor how the floor is set so that the unstaked population as a whole stays within device capacity.
6. **Matching rule.** Price-time FIFO is the baseline. Compact maker curves and a universal curve representation remain open; representation and clearing are separate choices. Frequent batch auctions [12] would make the contested set per-level and parallel by construction, at the cost of time priority.
7. **Collateral as a counter.** The funded ledger does not exist; the design constrains it, and §6.5 adds staked AMBA as a second collateral source with its own utilization cap.
8. **Remote role.** Archives, remote voters or standby takeover imply different guarantees. Local durability with asynchronous archives is selected and reversible.
9. **Physical networking.** The cheapest complete system with a working device-initiated path, including whether consumer parts can be made to provide it.
10. **Reconfiguration safety.** Weight changes at configuration boundaries (§6.3) need a hand-off argument, that a lock held under one configuration is honored by the next, and that the Byzantine bound holds across the boundary when stake moves. The threshold θ, the co-location requirement for remote candidates, and the disposition of slashed stake are open.
11. **Oracle challenge rules.** The chain fixes that settlement weight is delegated stake and is slashable; the acceptance tests, challenge windows and observation formats live in a separate specification and are not settled.
12. **Whole-device scaling for one hot market.** A thread block resides on one streaming multiprocessor; distributing one market's window across blocks needs a device-resident coordinator and a liveness argument.

8 Conclusion

The block has been doing three jobs that have no reason to share a boundary. Amoeba L1 gives the trust job to a per-market certified prefix, the ordering job to canonical position in a stream, and the scheduling job to bounded windows and leases that cannot change any result. The fee has been doing three jobs too; Amoeba L1 gives the metering job to a per-account allowance that stake raises and never sells, the payment job to market-making revenue, and declines to let anything buy position. The GPU is what makes this affordable. Packets land in device memory, state stays there, authentication and agreement run as kernels, and the apparent serial dependency of a contested book turns out to be a sort and a scan. The parallel matcher, the physical data path, the funded ledger, the stake stream and every performance number are still ahead, and the paper is written so that each can be checked against it when it arrives.

References

[1] M. Castro and B. Liskov. Practical Byzantine fault tolerance. *OSDI*, 1999.

[2] R. Kotla, L. Alvisi, M. Dahlin, A. Clement and E. Wong. Zyzzyva: Speculative Byzantine fault tolerance. *SOSP*, 2007.

[3] E. Buchman, J. Kwon and Z. Milosevic. The latest gossip on BFT consensus. arXiv:1807.04938, 2018.

[4] M. Yin, D. Malkhi, M. K. Reiter, G. Golan Gueta and I. Abraham. HotStuff: BFT consensus with linearity and responsiveness. *PODC*, 2019. arXiv:1803.05069.

[5] G. Danezis, L. Kokoris-Kogias, A. Sonnino and A. Spiegelman. Narwhal and Tusk: A DAG-based mempool and efficient BFT consensus. *EuroSys*, 2022.

[6] K. Babel, A. Chursin, G. Danezis, L. Kokoris-Kogias and A. Sonnino. Mysticeti: Reaching the limits of latency with uncertified DAGs. arXiv:2310.14821, 2023.

[7] M. Baudet, G. Danezis and A. Sonnino. FastPay: High-performance Byzantine fault tolerant settlement. *AFT*, 2020.

[8] O. Alpos, B. David, J. Gilchrist, E. Kokoris-Kogias and D. Zindros. Pod: An optimal-latency, censorship-free, and accountable generalized consensus layer. arXiv:2501.14931, 2025.

[9] A. Yakovenko. Solana: A new architecture for a high performance blockchain. Whitepaper, 2018.

[10] R. Gelashvili, A. Spiegelman, Z. Xiang, G. Danezis, Z. Li, D. Malkhi, Y. Xia and R. Zhou. Block-STM: Scaling blockchain execution by turning ordering curse to a performance blessing. *PPoPP*, 2023.

[11] CuEVM: A CUDA implementation of the Ethereum Virtual Machine. Singapore Blockchain Innovation Programme, 2024.

[12] E. Budish, P. Cramton and J. Shim. The high-frequency trading arms race: Frequent batch auctions as a market design response. *Quarterly Journal of Economics*, 130(4), 2015.

[13] Hyperliquid. Technical documentation: HyperBFT and validator architecture. https://hyperliquid.gitbook.io/hyperliquid-docs.

[14] NVIDIA. DOCA GPUNetIO programming guide; GPUDirect RDMA documentation. https://docs.nvidia.com/doca/.

[15] G. E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, Carnegie Mellon University, 1990.

[16] T. Dao, D. Y. Fu, S. Ermon, A. Rudra and C. Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *NeurIPS*, 2022.

[17] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte and Z. Zhang. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. NIST PQC submission, 2020. https://falcon-sign.info/.

[18] Algorand. Deterministic Falcon-512 implementation. https://github.com/algorand/falcon.

[19] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe and B.-Y. Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2), 2012.

[20] S. Josefsson and I. Liusvaara. Edwards-curve digital signature algorithm (EdDSA). RFC 8032, 2017.