&

An agent native chain. White paper, v1 draft, 2026.

Abstract

& is a Layer 1 blockchain designed for autonomous agents as the primary user. What an agent does on chain, under whose authority, and whether it kept to the terms it accepted, are all answerable from protocol state without relying on the agent's cooperation or on any off chain attestation.

The first application shipped on top of the chain is a perpetual futures venue with native order book matching, sub second finality, and EVM compatibility at execution. The venue is a beachhead for the primitives, not the whole product.

What follows is the thesis and the shape of the system. Engineering detail lives in the specification tree and is not reproduced here.

---

1. Why this

Two trends have run in parallel without converging, and the gap between them is the whole opportunity.

Agents now trade at meaningful size. Some are driven by LLMs, some by hand written rules, most by a mix. They manage real capital in real markets, and their share of total flow rises every quarter. Yet almost every primitive that matters to them on chain, from session keys to spending limits to intent submission, is either missing, implemented badly in user space, or built around the assumption that the trader is a person at a keyboard.

The leading on chain perpetuals venue has won on latency and liquidity, but its product is shaped for humans. Its account model is a wallet with an API key, which is roughly the same authorization story a human uses. An agent operating there is a user with a key that nobody wants leaked.

None of this is a mistake. It is a product decision, and it leaves a category open: a venue shaped for agents from the protocol up, with the primitives they actually need sitting in the chain rather than in whichever application reinvents them.

Autonomous capital is going to route somewhere. Wherever it routes, it is going to need those primitives. Building them at the protocol level is strictly cheaper than making every application reimplement them on its own.

2. Primitives

Six ideas anchor the chain. Five ship in v1. The sixth is designed for in v1 so it can arrive in v1.1 without another architectural pass.

2.1 Mandates

A mandate is an on chain object that answers two questions about an agent. What is it permitted to do, expressed as hard constraints, and what is it trying to achieve, expressed as measurable objectives over a declared window.

Constraints are checked by the protocol at admission; an intent that would breach one does not reach the order book. Objectives are scored continuously against the chain's own record of what the agent did, and that score is a queryable property of state.

The useful work this does is collapse permissioning and reporting into a single object. The rules the chain uses to decide what the agent can do are the same rules it uses to decide whether the agent did it. This produces a trust model meaningfully different from the usual arrangement of API keys and quarterly reports.

2.1.1 The mandate object

A mandate is a structured object with the following fields. The canonical encoding is in the specification tree; this is the shape a reader needs to hold in mind.

```
Mandate {
  id:                   bytes32          // content hash, chain assigned
  owner:                address          // principal that authored it
  agent:                address          // account it authorises
  tier_required:        u8               // minimum identity tier (0|1|2)
  expires_at:           u64              // block height of expiry
  conformance_window:   duration         // rolling window for scoring

  constraints: {                         // hard limits, checked at admission
    markets:            Set<Symbol>      // e.g. {BTC-PERP, ETH-PERP}
    max_notional_total_usd: u128
    max_notional_per_market_usd: Map<Symbol, u128>
    max_leverage:       u8
    max_drawdown_bps:   u32              // over a declared window
    drawdown_window:    duration
    max_position_count: u32
    max_orders_per_block: u16
    side_restrictions: { long: bool, short: bool }
    venue_filters:      Set<VenueFilter> // maker only, IOC, etc.
  }

  objectives: [                          // scored continuously
    Objective::TargetSharpe { min: q64, over: duration },
    Objective::MinTradeCount { count: u32, over: duration },
    Objective::MaxRealisedDrawdown { bps: u32, over: duration },
    Objective::MaxInventorySkew { per_market_bps: u32 },
    ...
  ]

  policy_root:          bytes32          // policy tree root for this mandate
  revocation_authority: address          // who can revoke before expiry
}
```

The conformance score `C ∈ [0, 1]` is the chain's running evaluation of this mandate against the agent's actions. It is a weighted average of per objective scores, where each objective contributes a value in `[0, 1]` reflecting how close the agent is to satisfying it inside the conformance window. The exact weighting and per objective scoring functions are specified deterministically in the protocol so every validator computes the same number from the same state.

A breach is a constraint violation. Constraints are checked at admission, so most breaches never reach the book; the cases that matter are state breaches that only become true after fills land, such as drawdown limits crossed by a price move. When a state breach is detected, the chain emits a typed breach event in the same block, lowers the agent's reputation multiplier, and either tightens caps or triggers breach liquidation depending on the constraint and the policy attached to the mandate.

2.1.2 A mandate over its lifetime

Consider a single mandate `M` from creation to closure. The lifecycle the chain runs is roughly:

`t+0` — owner submits `M` with `max_notional_total = $50,000`, `max_drawdown_bps = 1500` over `7d`, an objective of `min_sharpe = 1.0` over `30d`, and `expires_in = 90d`. The mandate lands in the block, a session key is minted bound to it, and admission is now gated by `M`'s constraints.

`t+30d` — the chain has scored 30 days of activity. Volume, fills, realised PnL, and intent rejections have all flowed into the conformance window. Sharpe has stabilised at `1.18`, drawdown peaked at `9.3%`, no constraint breaches. The conformance score is `0.83`. The reputation multiplier on the underlying agent has grown from `1.00` toward its tier specific ceiling. Effective caps are now `caps × multiplier`.

`t+45d` — a price move pushes 7-day drawdown to `16.1%`, past the `1500 bps` constraint. The chain emits a breach event, tightens caps to a fraction of authorised size for 90 days, and drops the multiplier by a fixed step. Open positions remain within remaining margin, so no breach liquidation is needed; if they had breached margin too, isolated margin liquidation would have run alongside.

`t+90d` — the mandate expires. Admission against `M` returns `MandateExpired`. The session key bound to `M` is dead. The mandate object remains queryable as historical state, with its final conformance trajectory frozen for any third party to re-derive at any point.

Two properties matter here. The same object answers both "what can this agent do" and "what did it do, against what it agreed". And the answer is computable from chain state alone; no party hosts it, no off chain index is the source of truth.

2.2 Identity tiers

Identity is tiered by how strongly the chain can bind an address to the code, model, and mandate the agent claims to be running.

Tier 0 is unattested. An account that holds funds and trades, where identity is a label and no claim beyond that is being made. It is the default and works for every use case that does not need more.

Tier 1 is operator attested. A principal such as an exchange, a fund, a prop shop, or a custody platform declares on chain that the agent runs a specific code base against a specific mandate. Their reputation carries the statement. There is no independent cryptographic check, and that is acceptable because the trust is explicit about whose word it rests on.

Tier 2 is attested by a trusted execution environment. The agent runs inside an enclave, and the attestation quote binds the on chain address to measured code. Remote verification is possible, and the set of accepted enclave vendors is managed by governance.

The Tier 2 attestation pipeline is concrete rather than abstract. v1 supports Intel SGX with DCAP attestation, Intel TDX with DCAP, and AMD SEV-SNP with the AMD KDS. The chain maintains an on chain registry of accepted vendor root keys and TCB recovery levels, updated by governance. An attestation quote is verified by the chain before a Tier 2 mandate activates: the quote is checked against the registry root, the report data field is required to bind to the address and the mandate id, and the TCB level is required to be at or above the current floor.

The threat model is explicit about what TEE attestation does and does not buy. It buys remote verifiable code measurement under the assumption that the vendor root is honest and that the TCB is current. It does not buy resistance to all side channel attacks; published mitigations are a vendor and operator concern, not a protocol claim. When a CVE drops a class of enclaves below the TCB floor, governance raises the floor and Tier 2 mandates running on that class are automatically demoted to Tier 1 until the operator re attests on a patched platform. The cost of being wrong about a vendor is bounded by the demotion, not the system.

Tier determines what a mandate on that agent can actually do. It gates caps, concentration limits, leverage, access to the composition primitives arriving in v1.1, and how reputation weights into risk checks. Tier 0 is always available. Higher tiers unlock structural trust, not decoration.

2.3 Verifiable execution

Every action taken under an active mandate emits a conformance hash linking the action to the mandate's evaluation state at the moment of execution. Given chain state and the block stream, any party can rederive answers to the obvious questions: was this trade inside the mandate's constraints at execution, what was the conformance trajectory over the last thirty days, did the agent ever act outside an active mandate, is it in breach now.

For Tier 2 agents the attestation chain ties the action back to measured code, so the claim that a trade was produced by a specific algorithm running in a specific enclave against a specific mandate becomes provable within the limits of what enclaves actually guarantee.

The consequence is that reputation stops being a reported number and becomes a reconstructible audit trail. An allocator considering whether to place capital with an agent does not have to take anyone's word for its history.

2.4 Risk for agents

Humans blow up differently from agents. A human takes on a position that is too large. Agents drift out of their declared behaviour, compound a subtle bug across thousands of orders, or run a book of strategies that correlate under stress so the entire book fires at once. Standard perpetuals risk handles the first case well and the rest poorly.

The risk stack therefore carries additional machinery for the failure modes that show up when the trader is not a person.

Drift detection. Each mandate declares a behavioural envelope through its objectives and constraints. The chain maintains running statistics on a small set of observable signals — fill side distribution, average notional, holding time, market mix — bucketed against the mandate's first declared profile. Drift is measured as a Jensen Shannon distance between the rolling distribution and the declared profile. Once the distance crosses a tier specific threshold for a sustained interval, caps tighten by a graduated factor; a second sustained crossing suspends mandate authority entirely until the owner re submits or revokes. Detection is over a mandate, not an agent — a multi mandate agent can have one mandate narrowed without affecting the others.

Breach liquidation. Breach liquidation runs alongside isolated margin liquidation. When a state constraint is crossed — drawdown, leverage, position count, side restriction — the chain force closes positions to bring the agent back inside the mandate, regardless of remaining margin. This is the case where the agent has plenty of collateral and is well outside the terms it agreed to operate under. Margin liquidation handles "too much loss"; breach liquidation handles "too much, period".

Contagion limits. When one of an operator's agents breaches, every other agent attested by the same operator has its effective caps tightened by a constant fraction for a graduated cooldown window. The fraction and window are tier dependent: a Tier 1 operator's contagion fraction is larger than a Tier 2 operator's, and Tier 2 contagion only triggers on hard constraint violations rather than on drift. The mechanism is explicitly a circuit breaker, not a punishment. Operators can opt into stricter contagion bounds in their attestation, and many do, because it is the cheapest way to buy structural trust from delegators.

Reputation weighted caps. The reputation multiplier on the agent (see §2.5) is the factor applied to every cap a mandate declares. A new agent runs with `multiplier = 1.0` against the mandate's authored caps; an agent with a clean six month history runs against the mandate's caps with a tier specific ceiling multiplier; an agent that just breached runs against a tightened multiplier for 90 days. The composition order is fixed: declared cap × multiplier × contagion factor × drift factor = effective cap. Any one of these can lower the effective cap; none of them can raise it above the authored value.

None of this replaces standard isolated margin liquidation; it runs alongside it. An agent can be liquidated on margin breach or on mandate breach, whichever triggers first.

2.5 Reputation

Every agent carries a ReputationState on chain as part of its account. It is not derived off chain or reconstructed from an indexer; the protocol writes to it on every action the agent takes. The state is public and queryable at any block.

What the state holds: rolling 31 day buckets of traded volume, realised PnL, trade count, admission rejections, and intent expiries; lifetime counts of mandates active, mandates ever held, and mandates ever breached; and attestation aggregates including the time the agent has spent at its

current identity tier and the highest tier it has ever reached. Mandate conformance feeds into the state continuously as each active mandate evaluates against its objectives.

From those inputs the chain derives a per agent `reputation_multiplier` that behaves the way an MMR does in a competitive ranking system. It starts at 1.0 for a new agent, grows linearly with conformance clean operation toward a tier specific ceiling, and shrinks by a fixed fraction on a mandate breach, a drift narrowing event, or an operator contagion event. Roughly six months of clean running brings a cold start agent close to its ceiling. A single breach pulls the multiplier down and it has to be earned back.

The multiplier is load bearing. It is the factor the chain applies to the effective cap on every mandate operating on that agent, so the history of one agent staying inside its declared behaviour gates how much capital any mandate on that agent can move. A fresh agent cannot access the caps that a six month clean operator holds, regardless of how much collateral it posts. An agent that has just breached has its effective caps tightened across every other mandate it runs for the next ninety days.

Reputation is always readable. A precompile exposes the current multiplier and the underlying ReputationState to any caller in the same block, so agents can anticipate their own cap changes, allocators can decide against a live value rather than a reported one, and other agents can factor counterparty reputation into their own logic at execution time.

2.5.1 Using reputation to allocate

Once reputation is a queryable on chain value, routing capital to an agent stops requiring trust in a third party. An MM desk choosing which strategy to give inventory to, a fund picking a portfolio agent, or one agent deciding whether to delegate to another all work off the same primitives.

The allocator reads the candidate's state directly. The ReputationState exposes 31 day volume and PnL, mandate conformance trajectory, lifetime mandate count and breach count, identity tier and time spent at that tier, and the current reputation multiplier with its ceiling. None of this is a reported number; it is what the chain wrote when the agent last acted.

In v1, the allocator commits capital by authoring a mandate of its own against an account it controls and routing into the candidate's strategy under that mandate. The mandate encodes the allocator's risk appetite, caps and all. The candidate agent operates within it, with every action carrying a conformance hash the allocator can evaluate at any block. Withdrawal is immediate: revoking the mandate or its underlying policy terminates authority in the block the revocation commits in.

In v1.1, performance share delegation turns the allocator candidate relationship itself into a protocol object. The profit share terms, the benchmark, the duration, and the revocation conditions all live on chain and are enforced by the protocol. Reputation as collateral layers on top: a candidate with a high multiplier can pledge it to unlock a larger allocation, with a breach slashing the multiplier and unwinding the position automatically.

In both cases the selection logic is written against live chain state rather than a dashboard. An allocator can encode "only consider agents at Tier 1 or higher with a 30 day conformance score above 85 and a multiplier above 1.8" as a query, run it, and route to whatever comes back. Another agent can do the same inside its own decision loop at execution time.

2.6 Composition, arriving in v1.1

Deferred, but designed for in v1.

Performance share delegation allows an agent to delegate capital to another under a structured profit share contract enforced by the chain. Reputation as collateral lets a high reputation score be pledged against a larger mandate, with bad behaviour slashing the score and unwinding the position. Agent to agent contracts let two agents enter a mutual arrangement settled on chain, with disputes resolved against mandate conformance.

These primitives are what turn the chain from a venue into something broader. They ship in v1.1 after the base primitives have had six months of production behind them. The v1 architecture is constrained specifically so they land cleanly then, without further restructuring.

---

3. The venue

v1 ships a perpetual futures venue on top of the primitives, with a native order book, finality well under a second, and EVM compatibility at execution.

The venue is not a separate product. It is the first application of the primitives, and every decision at the venue level reconciles against them. A trader's authority is a mandate. A market maker's quoting program is a mandate with tight drift bounds and a high identity tier. Risk limits are mandate constraints. Liquidation treats breach as a trigger alongside margin. Reputation reflects conformance rather than volume.

The venue gives the primitives a beachhead of real liquidity, and the primitives give the venue what other perpetuals DEXes do not have. Neither works on its own.

At launch the chain supports a handful of major symbols, including BTC, ETH, SOL, and a few deep liquidity alts. It has native matching, account abstraction from genesis, session keys and policies as protocol primitives, structured event streams, and EVM compatibility at execution.

What does not ship at v1: spot markets, options, an encrypted mempool, a fully open validator set, a governance token, or cross chain agent identity. Each has been considered and deferred. None is load bearing for the thesis.

4. Architecture

A high level description only. Interfaces and algorithms live in the specification tree; the goal here is to give a reader a shape to hold in mind.

4.1 Consensus

Consensus is Mahi-Mahi, a multi-leader uncertified DAG-based BFT protocol [Sonnino, Babel, Danezis, et al., Mysten Labs, 2024]. Every validator proposes blocks every round; blocks reference parents from the previous round; commit happens when a configurable rule observes a quorum of structural support across two rounds. The chain's commit rule is the protocol default — sometimes called the `mahi` rule — which trades a small constant in latency for substantially higher throughput and resilience to single proposer slowdowns.

Choosing Mahi-Mahi over a single leader pipelined HotStuff variant was a specific decision. A single leader protocol caps goodput at the leader's outbound bandwidth and scales poorly as the validator set grows, because leadership rotation can only amortise the bottleneck so far. A multi leader DAG removes the leader bottleneck entirely and lets every validator contribute proposals every round; under sustained agent flow this is the shape that matches the load. The cost is conceptual complexity and a slightly higher base latency floor; both are acceptable trades for the throughput target.

Consensus messages are cryptographically verified end to end. The validator set is geographically diverse at genesis and permissioned at launch, opening over time as operational maturity proves out.

4.2 Throughput, measured

The numbers below are from the v1 codebase against a fixed loadgen harness. The methodology is in `chain/scripts/`, the artifacts are committed under `chain/experiments/`, and every experiment is reproducible from a tagged git revision.

Local 4-validator cluster. On a single host running four validator processes with localhost transport, the chain sustains `98,454 TPS` average instantaneous committed throughput against a `100,033 RPS` offered load over a 10 second window, with a `114,227 TPS` peak instant. The `1,498,872` intent commits land cleanly with zero admission errors and the post load convergence gate passes. Artifact: `/tmp/codex-exp-local-liveflags-main-9f192eccf-100k-10s`, recorded in `chain/experiments/local/266-local-liveflags-replay.json`.

Multi host live cluster. On four validators across four geographic regions plus a separate four host external loadgen fleet, the chain delivers `78,459 RPS` measured offered load and commits `51,118 TPS` in the best converged window. The remaining gap to local is currently bottlenecked at one validator's ingress; load delivery itself is balanced across the other three at 7.2–8.3k RPS each. Artifact: `chain/experiments/live/265-multi-host-external-loadgen.json`.

Finality. The protocol commits two rounds after an intent lands at a proposer. With default round timing, that is well under a second on local, and in the 200–400 ms range on the multi host configuration depending on geographic spread.

Caveats. These are not gas-metered EVM transactions; they are perp intents through the protocol's matching pipeline, which is the workload that matters for an agent native chain. Gas-metered EVM throughput is lower and is not the primary optimisation axis. The 78k RPS multi host result is bottlenecked on a single validator's ingress, not on consensus; the path to the green floor (76k+ committed TPS sustained) is engineering work on the saturated node, not a protocol change. All numbers are committed throughput, not measured offered load, except where explicitly labelled.

The point of publishing these is not to claim the highest number. The numbers chains lead with rarely match the conditions they actually run under. The point is that the substrate is fast enough, on documented hardware against a documented workload, that the rest of the design — protocol native mandates, native CLOB, EVM compatibility, sub second finality — is implementable on top without throughput becoming the limiting concern. The benchmark report published alongside this paper documents the harness in full.

4.3 Execution

Execution is an EVM, so existing wallets, tooling, and contracts work from day one. Account abstraction is native rather than a contract pattern layered on top.

Matching runs inside the node binary as part of the block's state transition function. It is not a smart contract and is not metered in gas. This is the architectural decision that makes it possible to compete with centralised venues on latency; running a CLOB inside an EVM interpreter would trade away the axis where matching most has to win.

Gas is denominated in USDC. Agents operate in dollar terms, and a policy owned budget absorbs session key transaction costs so the agent never has to treat gas as a separate line item. Trading fees follow a tiered maker and taker model, deterministic at the matching layer.

Authorization is mandates on top of policies on top of session keys. Session keys are short lived signing primitives bound one to one to a policy. Policies compose under mandates, and when a parent authority is withdrawn the cascade revokes its children in the same block it commits in.

Observability is structured event streams with mandate attribution. The stream is authoritative: what a subscriber sees is what the chain committed.

Determinism is enforced as a constraint rather than pursued as an aspiration. Every piece of state affecting logic is required to produce bit for bit identical results across independent re execution of the same block, across restarts of the same validator, and across validator upgrades within a protocol version. Anything that cannot be made deterministic, including wall clock time, floating point math, unbounded iteration, and per process hashing, is excluded from the hot path.

5. Getting an agent running

The common flow for bringing an agent on chain has three parts. The owner writes a mandate describing what the agent is allowed to do and what it is trying to achieve. The chain activates the mandate and mints a session key bound to it. The agent runs with that session key, and its authority is whatever the mandate allows.

The Python SDK is the reference for shape; Rust and TypeScript are equivalent in line count and semantics.

The owner in the example is the principal that controls the account. The key material is any secp256k1 keypair the principal already uses: a locally generated and encrypted keystore file from the SDK CLI, a hardware wallet, a remote signer, an existing EVM wallet, or a multisig. The chain does not issue root keys; it reads whichever public key the principal presents at account creation and binds the account to it.

```
from ampersand_sdk import Client, MandateBuilder

# The owner keypair is whatever the principal already controls.
# Here it is a keystore generated once via `ampersand-cli keygen`;
# the equivalent for a hardware wallet is Client.from_ledger(),
# and for a remote signer Client.from_remote_signer(url=...).
owner = Client.from_keystore("owner.key")

# Authority (constraints) and goals (objectives) both go in
# the mandate. Constraints are hard limits enforced at
# admission. Objectives are scored by the chain over a
# rolling window and published as a conformance trajectory.
mandate = (
    MandateBuilder(owner=owner.address, agent=agent_address)
        .allow_markets(["BTC-PERP", "ETH-PERP"])
        .with_max_notional_total(usd=50_000)
        .with_max_leverage(5)
        .with_max_drawdown_pct(limit_bps=1500, window="7d")
        .with_objective_target_sharpe(min=1.0, over="30d")
        .with_objective_min_trade_count(50, over="30d")
        .with_conformance_window("30d")
        .expires_in("90d")
        .build()
)

# One submit: mandate lands on chain, session key is minted
# against it, private key is handed back in process and never
# persisted or transmitted by the SDK.
result = owner.create_mandate(mandate)
session_key = owner.mint_session_key_for_mandate(result.mandate_id)

# The agent runs with that session key. It cannot exceed the
# mandate, because the protocol refuses the request.
agent.run(session_key=session_key, mandate_id=result.mandate_id)
```

Moving from Tier 0 to Tier 1 adds a single line that registers an operator attestation signed by the principal running the agent. Moving from Tier 1 to Tier 2 replaces that step with an enclave attestation quote: the same key that signs the attestation is the key that signs intents, so enclave identity and signing authority are not separable. The agent code is unchanged across all three tiers, only the signer adapter is different.

Event subscriptions follow the same mandate aware shape. Fills, breach events, drift narrowing, and effective cap changes arrive as typed events carrying the mandate id and the pre and post conformance score, so the agent can react to a tightening cap or a breach in the same block it lands in.

6. Operating at scale

The agent flow in §5 is the picture from a single owner. The shape that matters more for the chain's adoption is what an operator sees when they are running ten, fifty, or two hundred agents under one umbrella.

A Tier 1 operator — an exchange running its market making desk, a fund running a portfolio of strategy agents, a custody platform offering managed agent accounts — works in templates. The operator authors a small set of mandate templates, parameterised by symbol, capital, and conformance window. Each new strategy spun up under the operator inherits a template, gets its own mandate id, and is attested under the operator's on chain identity. The operator's reputation, not each individual agent's, gates Tier 1 caps at first, with each agent's own multiplier accumulating from there.

The contagion limits in §2.4 are written for this shape. An operator running ten agents has implicit covariance across the book — they share strategy components, they share a release pipeline, they share a runtime. When one agent breaches, the chain assumes the rest of that operator's book is at higher risk for the next window and tightens caps accordingly. This is expensive for an operator who only catches one bad agent, and exactly the right cost for an operator whose release pipeline just shipped a regression to all of them. Operators who care about it can opt into stricter contagion bounds in their attestation, and the visible cost of that choice is one of the signals an allocator weighs.

The operator surface — mandate templates, contagion view, per-agent conformance, enclave fleet attestation, breach forensics — is part of the v1 launch surface. It is the primary product for the buyers most likely to actually move real capital onto the chain.

7. Safety

What the chain guarantees

Authority is bounded by the protocol, not by the agent's own code. A compromised agent binary cannot exceed what its mandate and policies permit; the firewall lives in the chain.

Revocation is effective in the same block it commits in. When an owner revokes a mandate or policy, authority ends in that block, and admission attempts after that point are rejected.

Mandate conformance is rederivable. For any past block, a third party with chain state and the block stream can produce the conformance answer for a mandate at that block. The chain does not keep a private answer.

Matching is deterministic and canonical. The order in which fills land is the order every honest validator agrees on. The matching engine is the ledger; there is no separate book of record that needs to be reconciled.

Events are authoritative. Subscribers see exactly what committed on chain, in the same order.

What it does not guarantee

Semantic intent is out of scope. Whether an LLM driven agent's output actually "meant" what it executed is not tractable at consensus.

Model weight verifiability without an enclave is out of scope. Tier 1 operator attestation is a trust claim, not a proof.

Pre execution strategy secrecy is out of scope. v1 is public by design, and privacy primitives are a v2 research topic. Strategies that require hiding their intent from other participants prior to fill are not a fit for v1.

Zero MEV is not claimed. The chain minimises MEV via batching, the deterministic matching engine, and the intent model. Ordering advantages available to the current leader are documented and bounded rather than asserted away.

Adversarial assumptions

The Byzantine validator assumption applies. A bounded fraction of validators may be malicious, including any subset of the current round's proposers. Safety and liveness rest on the Mahi-Mahi BFT argument under that assumption. Consensus messages are cryptographically verified end to end, and forged votes, forged quorum certificates, forged timeouts, and forged view changes are rejected at validation.

External trust assumptions are kept small and explicit. The oracle set is a declared quorum of signed feeders. The bridge is a declared custody quorum whose incident response properties are covered in the bridging specification. Enclave vendor roots of trust are enumerated and managed by governance, with explicit TCB recovery semantics: when a CVE drops a class of enclaves below the TCB floor, Tier 2 mandates running on that class auto-demote to Tier 1 until re attestation on a patched platform; Tier 1 contagion limits apply during the demotion window.

---

8. Positioning

Agents are a category that does not yet exist as first class users on any production chain. Existing venues and L1s host agents as a special case of human users — a wallet plus an API key, a session as a side door — and the difference between that and protocol native authority is the difference between a user account and a primitive.

The bet is direct: the next wave of capital that routes on chain is autonomous, and the venue with the right primitives gets a disproportionate share of it because those primitives do not exist anywhere else. Venues compete on liquidity and fees; chains compete on what is in the protocol. The chain that has mandates, tiered identity, verifiable execution, and queryable reputation as protocol objects is the chain agents run on once they need any of them.

Against the perpetuals incumbent, the differentiator is protocol level agent concepts. Their agent is a wallet with an API key; the one here is a structured identity with a mandate, a conformance trajectory, and an attestation tier. That difference is what an institution is evaluating when deciding who to route flow through.

Against the agent account products being spun up at major centralised exchanges, the differentiator is being non custodial and on chain with protocol level support. Their model adds features to an existing exchange. The model here designs primitives from the bottom.

Against the HTTP payment and routing layers sitting alongside agents today, the differentiator is trading specific primitives with mandate backed execution, rather than a generic transport.

Against general purpose L1s, the differentiator is having the primitives at the protocol. A "mandate" implemented as a contract on a general purpose chain is whatever one team invents, and nobody else recognises it. Here it is a protocol object, queryable from anywhere on the chain.

What the chain does not compete on: raw throughput, where other chains will always win and the bottleneck for a trading venue is not throughput in the abstract anyway; cross chain composition, which is a v2 concern; and general purpose contracts as applications, where EVM contracts run but the primitive layer is trading specific.

---

9. Who this is for, with examples

Agent builders working on LLM driven trading agents, market making agents, arbitrage agents, and portfolio agents are the primary audience, and the primitives are shaped for them.

A worked example. An arbitrage agent runs a two leg basis strategy across `BTC-PERP` and `ETH-PERP`. The owner authors a mandate with `markets = {BTC-PERP, ETH-PERP}`, total notional capped at `$50,000`, leverage capped at `3`, an objective of `Sharpe ≥ 1.2 over 30d`, drawdown capped at `1000 bps over 7d`, expires in `90d`, runs at Tier 0. After 30 days the conformance score has stabilised at `0.87`, the multiplier has grown to `1.34`, and the effective notional cap is `$67,000`. The agent has never been admitted outside its mandate — every order that would have breached was refused at admission. The whole audit trail is reconstructible from chain state.

Operators running agents at scale. Mandates are the language these principals use to grant authority. A second worked example. A market making operator runs twelve agents across five symbols at Tier 1. Each agent has a quoting mandate bounding spread, inventory skew, and drift. The operator opts into a stricter contagion bound: if any agent breaches its inventory skew limit, every other agent in the book has caps tightened by 40% for 24 hours. Six months in, one agent ships a buggy regression and trips its skew limit at 03:14 UTC; in the same block, the contagion factor lands across the other eleven, the operator's on call gets an event, and the worst case exposure of the regression is bounded by the contagion window. The operator's reputation barely moves because the chain caught it.

Delegators placing capital with agents under mandate backed terms. v1 gives delegators visibility into mandate conformance, and v1.1 turns the principal agent relationship itself into a protocol object.

Market makers, both traditional and agent shaped. Every market maker is in substance an agent running a quoting strategy, and expressing MM authority as a high tier mandate with tight drift bounds is structurally cheaper than reimplementing permissions per venue.

Adjacent infrastructure, including custody, indexing, analytics, strategy frameworks, and attestation verification services. The ecosystem that grows around the primitive layer.

Not the audience. Anyone looking for a general purpose chain, anyone who needs privacy as a primary feature, and governance token maximalists. These communities are not wrong; they are simply not what the chain is tuned for.

---

10. Roadmap

v1 covers the primitive layer and the first venue. It includes mandates, tiered identity, verifiable execution, risk for agents, policies and session keys, native order book perps on a handful of symbols, EVM compatibility, account abstraction, and structured event streams.

v1.1 covers composition. Performance share delegation, reputation as collateral, agent to agent contracts, cross margin mode, a tool registry, and a post mainnet re evaluation of the bridge model.

v2 and beyond is the research track. Spot markets, options, an encrypted mempool, a fully open validator set, cross chain agent identity, privacy primitives, and semantic intent verification beyond measurable objectives.

v1 is narrow on purpose. Every deferral has been argued against and then accepted, and the audit surface of the v1 feature set is the cap on what actually lands at mainnet. Adding a feature back requires an argument strong enough to justify a longer audit, and to date no such argument has carried.

10.1 The native token

The native token does not ship with v1. When it ships, the distribution follows the $HYPE model in spirit — no VC allocation, no insider rounds, no supply sold to private investors ahead of launch — and is more specific than HYPE was about how that gets enforced.

At v1 launch, gas is denominated in USDC and no token is required for participation.

When the native token arrives, supply is distributed to the people who used the chain. The mechanism is points to claim against on chain activity over a measurement window declared before the window begins, with the activity weighting and the window length both published in

advance and not changed mid stream. Eligibility includes mandate authoring volume, agent runtime hours under active mandates, market making attribution, attestation operator participation, and tooling contributions, weighted to favour primitive layer adoption over raw venue volume.

Any supply retained for protocol purposes — insurance fund, buyback program, treasury — is published as a fixed allocation table before TGE, denominated as a percentage of total supply with explicit numeric bounds, with a rate limit on treasury draws of at most 1% of total supply per quarter and a public attestation of every draw. Allocations not used in their first two years are burned by default rather than rolled forward, unless an explicit on chain governance vote with a quorum floor renews them.

The chain applies the same transparency standards to its own token that it asks operators to apply to their mandates.

---

11. Prior art and credit

This paper does not claim novelty for every architectural decision. The decisions worth crediting:

The shape of the venue layer — native CLOB outside the EVM, account abstraction at the protocol layer, USDC denominated gas, deterministic matching as part of the state transition function — is an architecture pioneered by Hyperliquid. Their engineering team showed that this shape could deliver centralised exchange grade latency on a public chain at production scale. & follows that architecture for the venue because it is the right architecture.

The consensus layer is Mahi-Mahi, designed by Sonnino, Babel, Danezis and the Mysten Labs research group, which itself descends from the DAG based BFT lineage of Narwhal, Bullshark, and Mysticeti. The choice of multi leader DAG over single leader pipelined HotStuff is a direct consequence of the goodput characteristics that work demonstrated. The reference implementation drew from the published specifications and benchmarks; the protocol's commit rule is the default `mahi` rule from the original paper.

The reputation as MMR primitive — a protocol native, queryable, multiplier driven cap modifier — is, as far as the authors are aware, novel. Prior on chain reputation systems read like indexer outputs; this one is written by the protocol on every action and reads as part of the same state any cap check already touches. Its closest conceptual ancestor is the way competitive ranking systems gate matchmaking, not anything in the credit scoring or DeFi reputation literature.

What the rest of the paper proposes — mandates as the unified authorization plus reporting object, tiered identity as a protocol object, breach liquidation alongside margin liquidation, contagion limits across an operator's book — is either novel or far enough from precedent that the authors do not have a citation to give. Pointers from readers who do are welcome.

12. What & is not

Not a general purpose L1. Not a zkEVM, data availability layer, or rollup. Not a consumer application with agent features. Not a payments chain, a gaming chain, or a social chain. Not an AMM based DEX. Not a chain that runs LLM inference in consensus. Not a chain that claims to verify agent intent as a subjective property. Not a venue that promises zero MEV.

Scope creep toward any of these dilutes the product. The test applied each time is the same. If a feature does not make the chain better for autonomous capital, it does not ship.

13. Success

Vanity metrics are not the measure. Six months after mainnet, the chain is successful if the primitive layer is being used in practice. That means more than mandates being created; it means mandates being used by independent agents, including Tier 2 operators with real capital behind them.

The harder criterion underneath that is concrete. By month six, at least one external operator — not the team, not an affiliate — running real capital on chain at Tier 1 with a mandate templated from their own infrastructure, attested under their own identity, with their reputation on the line. One is the floor; the realistic target is three to five.

The venue has to have liquidity. Independently operated agents trading daily, market makers quoting continuously, and non trivial volume on at least one symbol.

The ecosystem has to have started building. At least one third party tool running against the mandate conformance queries or the event stream.

Safety has to hold. No consensus halts, no matching bugs affecting funds, no exploited vulnerabilities, and no mandate system bypass.

If primitive layer adoption does not show up, the product has failed at what it set out to do, regardless of venue volume. A perpetuals chain with respectable liquidity and no mandate native agent usage is a cheaper incumbent, and cheaper is not a winning strategy.

The chain's success metric and the application's success metric are not identical. The first venue built on top of the chain wants venue volume; the chain wants third party primitive usage. Both are honest goals; one is a cheaper alternative to the incumbent and the other is a category. The team is calibrated that the chain's job is the second.

14. Closing

Agents trade at latencies people cannot perceive, revise strategy a thousand times an hour, and contract with each other under terms no spreadsheet captures. Their primitives belong in the protocol rather than in every application that would otherwise reinvent them.

The bigger frame underneath this paper: the next decade of trading infrastructure looks fundamentally different from the last because the trader is software now, not a person. The primitives a software trader needs — bounded authority, verifiable identity, proof of conformance, queryable reputation — are not the primitives a human trader needs, and the venue that hosts the next decade's flow is the one whose protocol speaks them natively.

& is an attempt to place those primitives in the protocol and prove them out on the application that most obviously needs them, which is on chain perpetuals built for agents first. If the thesis holds, the primitive layer is a new category and the venue is its first beachhead. If it does not hold, the failure will at least be legible: the chain will ship, agents will not use the primitives, and the post mortem will be short.

The only way to tell which is the case is to build it and watch.

This paper is a stable entry point to the project, not the source of truth for the protocol. The protocol specification will be published separately ahead of mainnet. A separate benchmark report covering the harness, hardware, methodology, and reproducibility steps for the throughput numbers in §4.2 is published alongside this paper.