Cascade
The Perpetual Liquidity Engine

Cascade Hypercorporation

November 6, 2025

Abstract

Cascade is a perpetuals venue and *perpetual liquidity engine*. It couples off-chain price discovery with deterministic on-chain settlement, and recycles its own cash flows into an on-chain *Liquidity Vault* that deepens future markets. Cascade is structured as a governance-minimized *hypercorporation*: an on-chain corporation whose default economics are expansive rather than extractive. Trading fees and other outputs fund the Liquidity Vault, which in turn supplies collateral and liquidity back to the exchange, compounding market capacity over time. The design preserves credibly neutral access for traders, vault depositors, and integrators, while enforcing a narrow set of invariants for risk, margin, and price integrity.

1 Motivation & Constraints

Motivation. Perpetual derivatives require *durable* liquidity: inventory that persists across market cycles, remains credibly neutral, and can be composed by many front ends. Cascade achieves this by internalizing its growth loop.

At each step, the Liquidity Vault grows by adding a portion of the protocol's residual fee share and any realized trading profits, while subtracting operational costs and grants:

$$\begin{aligned}\text{VaultBalance}_{\text{next}} = \text{VaultBalance}_{\text{current}} \\ + \text{ProtocolRevenueShare} + \text{RealizedProfitOrLoss} \\ - \text{OperationalExpenses}.\end{aligned} \tag{1}$$

As the vault grows, it supports higher open interest and tighter spreads, attracting more traders. This creates a *progressive liquidity* flywheel: activity drives fees, fees grow the vault, and the vault deepens liquidity, which in turn attracts more activity.

~~Hyperstructure~~ Hypercorporation. The Hypercorporation model focuses on protocols that are unstoppable, permissionless, and positive-sum by design.[1] Cascade adapts these principles to a derivatives system by:

1. maintaining settlement logic that is *upgradeable yet fully on-chain*, enabling transparent iteration without compromising determinism;
2. routing all fee flows to insurance buffers and the Liquidity Vault rather than to rent-seeking intermediaries; and
3. exposing a single, uniform interface that any originator (interface or third-party frontend) or auditor can integrate with, without privileged API keys.

System constraints. Three key constraints shape Cascade's architecture:

1. **Latency vs. custody.** Traders need millisecond-level execution, while collateral settlement must remain fully on-chain. Cascade separates these layers: a single matching engine handles price discovery and trade execution; the on-chain `Perpetual` contract enforces solvency, price bounds, and position rules at settlement (Sections 7, 3).
2. **MEV & price integrity.** Settlement is tied to a reference price with strict deviation limits. Margins are rechecked after every fill, and liquidation logic follows deterministic, auditable scripts (Sections 3, 4).
3. **Oracle reliability.** Market prices are sourced from authenticated feeds such as Stork, with decay-based fallbacks to ensure consistent references across all risk and funding calculations (Section 8).

All smart contracts are upgradeable under a controlled process governed by an on-chain Owner (DAO or multisig) without intermediary helper contracts. Governance will evolve

[1] Horne, *Hyperstructures: Crypto Protocols that Last*, sections *Definition* and *Expansive fees*.

under a model of progressive decentralization. The Liquidity Vault now lives as an internal ledger inside `Perpetual`, so capital providers deposit straight into managed accounts without wrapper contracts.

By default, nearly all residual fees continue to flow into the Liquidity Vault, and any change to this policy must be an explicit, transparent on-chain action.

2 Design Principles & Threat Model

Design Principles

1. **Credible neutrality.** Uniform rules for access, margin, and liquidation; no address-level whitelists in settlement paths. Interfaces (originators) may compete, but custody and rule enforcement remain shared and neutral.
2. **Progressive decentralization.** Contracts are upgradeable, with ownership migrating from the founding team to a DAO or multisig over time. No privileged escape hatches exist that bypass solvency or price checks.
3. **Expansive, not extractive, fees.** Fee splits reward aligned services while recycling residual protocol revenue into the Liquidity Vault to expand future capacity rather than pay platform rent.[2]
4. **Deterministic safety.** Every state transition revalidates lot sizes, price deviation, open-interest caps, and margin sufficiency (Section 3). Liquidations follow a two-tier ladder with insurance and ADL backstops (Section 4).
5. **Observability.** All significant actions—such as fills, funding updates, margin changes, and oracle commits—emit structured events to enable total replay and external verification (Sections 3, 8).
6. **Separation of concerns.** Off-chain services may optimize routing and UX but cannot mutate balances or override settlement rules (Section 7).

Threat Model

We assume adversaries maximize profit subject to gas and latency limits.

Oracle manipulation. Attempt to skew the reference price to trigger liquidations or favorable fills. *Mitigations:* signed Stork payloads; max-age checks; matcher-supplied mark/index quotes validated on-chain; symmetric price-band guard on all fills (Section 8).

Under-margined growth. Traders try to accumulate exposure faster than margin scales. *Mitigations:* per-fill rechecks of initial margin requirement (IMR), maintenance margin requirement (MMR), and closeout margin requirement (CMR); per-market open-interest caps; reduce-only toggles (Sections 3, 4).

[2] Horne, *Hyperstructures: Crypto Protocols that Last*, *Expansive fees.*

Liquidation games. Front-running or withholding around distressed accounts. *Mitigations:* standardized liquidation orders, deviation-bound maintenance unwinds, deviation-free closeouts, protocol-vault sweeps, ADL for terminal deficits, and trusted operators accountable for running liquidation flows (Section 4).

Matcher malfeasance. Reordering or preferential matching off-chain. *Mitigations:* EIP-712 intents; on-chain revalidation of limits; maker/taker role determined at settlement; no ability to bypass checks; and matcher responsibilities held by vetted actors during progressive decentralization (Section 7).

Governance capture. Parameter griefing to extract rent or disable markets. *Mitigations:* immutable settlement bytecode; bounded parameter intervals enforced on-chain; evented updates; default policy routing residual fees to the Liquidity Vault rather than discretionary wallets (Section 6).

Delegated custody drift. Account owners can authorize depositors or managers; misaligned settings could expose capital. *Mitigations:* explicit allowance surfaces inside `Perpetual`, per-delegate limits, replayable events, and revocation paths that never bypass solvency checks (Section 6).

Liveness & DoS. Congestion or oracle delay during stress. *Mitigations:* minimal on-chain compute per fill; fallbacks for price; reduce-only switches; permissionless liquidation entry points (Sections 8, 4).

Taken together, these principles aim to realize a hyperstructure-like exchange: permissionless to use, defensible under stress, and *expansive*—its fee loop continually augments the Liquidity Vault, compounding market depth for the next block and the next cohort of participants.[3]

3 Market Mechanics

The user state in Cascade is expressed as a simple three-part record that mirrors the on-chain storage layout:

$$\text{Account} = (B, \mathcal{P}, M). \tag{2}$$

B freely withdrawable collateral balance.

$\mathcal{P}$ list of open positions, including size, cost basis, and fees paid.

M margin amounts locked against each position.

Whenever a user action changes one of these fields, Cascade re-runs a short solvency check to confirm the account can still cover its obligations.

3.1 Account Topology

Cascade supports co-existing cross-margin and isolated-margin regimes, allowing participants to choose between portfolio-wide and position-local risk segregation.

[3] Horne, *Hyperstructures: Crypto Protocols that Last*.

3.1.1 Cross-Margin Portfolio

The aggregate net asset value is computed as

$$\mathrm{NAV}_{\mathrm{cross}} = B + \sum_{p} M_p + \sum_{p} \mathrm{uPnL}_p. \tag{3}$$

M_p margin capital allocated to position p.

uPnL_p unrealized profit or loss associated with position p.

Losses in one market offset gains in another as long as $\mathrm{NAV}_{\mathrm{cross}} \geq 0$, enabling efficient collateral usage for diversified strategies.

3.1.2 Isolated-Margin Slots

Each isolated slot carries its own solvency test:

$$\mathrm{NAV}_{ip}^{\mathrm{iso}} = m_{ip} + \mathrm{uPnL}_{ip}. \tag{4}$$

m_{ip} collateral locked to isolated position ip.

uPnL_{ip} unrealized profit or loss of the same position.

The collateral in the pool B does not compensate for isolated exposures, ensuring that $\mathrm{NAV}_{ip}^{\mathrm{iso}} \geq 0$ remains a hard constraint per-position.

3.2 Margin Hierarchy

Every market specifies three collateral ratios: initial margin requirement (IMR), maintenance margin requirement (MMR), and closeout margin requirement (CMR) with the strict ordering $\mathrm{CMR} < \mathrm{MMR} < \mathrm{IMR}$. For exposure of notional size

$$N = |A| \cdot P_{\mathrm{mark}}, \tag{5}$$

the required collateral stack is given by

$$N \cdot \mathrm{IMR} \geq N \cdot \mathrm{MMR} \geq N \cdot \mathrm{CMR}. \tag{6}$$

A signed position amount in base units (long > 0, short < 0).

P_{mark} protocol-defined mark price for the market.

$\mathrm{IMR}, \mathrm{MMR}, \mathrm{CMR}$ initial, maintenance, and closeout margin ratios.

Liquidation checks activate when $\mathrm{NAV} < N \cdot \mathrm{MMR}$. If the equity deteriorates further such that $\mathrm{NAV} < N \cdot \mathrm{CMR}$, an escalated closeout routine or auto-deleveraging sequence initiates.

3.3 Order Specification

Execution intents are represented by the tuple

$$\text{Order} = (A, P_{\text{lim}}, f), \tag{7}$$

where f is a small set of binary flags. The pieces map to execution behaviour as follows:

A signed execution amount; the sign encodes buy versus sell intent.

P_{lim} limit price that caps the fill price for buys and floors it for sells.

f execution policy flags defined below.

3.3.1 Flag Semantics

`postOnly` safeguards maker intent by rejecting any fill where the match price P_{match} would cross the posted limit. `takerOnly` enforces aggressive execution and, when paired with a very high (for buys) or very low (for sells) P_{lim}, behaves like a market order. The `adl` flag designates orders sourced from the auto-deleveraging queue and guarantees that the counterparty provides maker-side liquidity during systemic closeouts.

Advanced order instructions such as stop triggers, OCO, or iceberg variants are orchestrated off-chain. Execution services decompose higher-level flows into valid on-chain orders, preserving deterministic settlement logic while keeping the contract state minimal.

3.4 Execution Invariants

Every fill passes three straightforward checks before it reaches final settlement:

1. **Maker or Taker, Not Both.** Exactly one side of the trade must provide liquidity. If an order is marked 'postOnly', it cannot cross existing liquidity. If it is marked 'takerOnly', it must cross immediately.
2. **Margin Sufficiency.** When a fill increases exposure, the engine estimates the post-trade buffer using
$$\text{NAV}_{\text{after}} = \text{NAV}_{\text{before}} - \text{Notional}_{\text{fill}} \times \text{IMR}. \tag{8}$$
The trade proceeds only if $\text{NAV}_{\text{after}} \geq 0$.
3. **Liquidation Consistency.** If the counterparty is already in liquidation, the engine confirms that its equity has fallen below the maintenance and closeout thresholds:
$$\text{NAV}_{\text{counterparty}} < N \times \text{MMR}, \qquad \text{NAV}_{\text{counterparty}} < N \times \text{CMR (for closeout and ADL)}. \tag{9}$$

$\mathit{Notional}_{\mathit{fill}}$ absolute dollar value of the position change created by the fill.

$\text{NAV}_{\mathit{before}}$, $\text{NAV}_{\mathit{after}}$ equity snapshot before and after reserving margin for the new exposure.

$\text{NAV}_{\mathit{counterparty}}$ equity of the account being liquidated.

Together these checks guarantee that each fill moves the account into another safe state without any out-of-band reconciliation.

3.5 Position Lifecycle

Positions evolve exclusively through order fills. With A_t denoting the signed position amount after fill t and B_t the cumulative basis, Cascade updates

$$A_{t+1} = A_t + \text{fillAmount}_t, \qquad B_{t+1} = B_t + \text{fillAmount}_t \times P_{\text{match},t}. \tag{10}$$

$\mathit{fillAmount}_t$ quantity executed in fill t.

$P_{\text{match},t}$ trade price delivered by the matching engine.

Unrealized PnL is tracked on-chain through

$$\text{uPnL} = A_{t+1} \cdot P_{\text{mark}} - B_{t+1} - F_{\text{trade}} - F_{\text{fund}}, \tag{11}$$

with symbol interpretations

F_{trade} cumulative execution fees debited from the position.

F_{fund} net funding transfers between longs and shorts.

Closure occurs once $A_{t+1} = 0$, after which the settlement routine flushes the residual uPnL into the cross-margin balance B.

Policy parameters enforce a minimum position size $A_{\min}$ (positions can only be smaller if they are fully closed) and require every fill amount to be an exact multiple of the configured lot size q_{lot}. These simple rules keep order sizes consistent between on-chain and off-chain execution venues.

3.6 Operational Observability

Cascade retains only the current snapshot $(A, B, F_{\text{fund}}, F_{\text{trade}})$ on-chain to conserve gas. Order submissions, fills, funding accruals, and margin movements emit structured events $\{\mathsf{Fill}, \mathsf{Funding}, \mathsf{MarginUpdate}\}$, enabling off-chain indexers to reconstruct comprehensive trading histories while maintaining the auditability expected of a perpetual derivatives venue.

4 Risk & Liquidations

Cascade couples its order mechanics with a layered risk envelope that is enforced per market and per account. An on-chain owner (DAO or multisig) curates these parameters directly, updating them through governed transactions instead of an intermediary coordinator.

4.1 Risk Envelope

Each market publishes a vector of limits that steer leverage, position sizing, and price control:

$\mathrm{IMR}, \mathrm{MMR}, \mathrm{CMR}$ margin ratios defining initial, maintenance, and closeout collateral levels.

$A_{\min}, q_{\text{lot}}$ minimum live position size and the discrete lot step that each fill must respect.

maxOI cap on aggregate open interest, measured as the sum of absolute position sizes across all accounts.

maxDeviation guardrail limiting how far a fill price can drift from the oracle index.

makerFee, *takerFee* fee rates that fund the protocol vault and aligned distribution partners.

Margin requirements scale linearly with notional size $N = |A| \times P_{\text{mark}}$:

$$\text{RequiredInitial} = N \times \text{IMR}, \tag{12}$$

$$\text{RequiredMaintenance} = N \times \text{MMR}, \tag{13}$$

$$\text{RequiredCloseout} = N \times \text{CMR}. \tag{14}$$

New exposure also updates the market-wide open interest tally:

$$OI_{\text{next}} = OI_{\text{current}} + |\text{fillAmount}_t|, \tag{15}$$

which must satisfy $OI_{\text{next}} \leq \text{maxOI}$ for the trade to clear. Orders that reduce exposure bypass this check to avoid trapping exits.

4.2 Liquidation Ladder

Every account continuously evaluates two simple solvency tests:

$$\text{NAV}_{\text{account}} < N \times \text{MMR} \quad \Rightarrow \quad \text{eligible for maintenance liquidation}, \tag{16}$$

$$\text{NAV}_{\text{account}} < N \times \text{CMR} \quad \Rightarrow \quad \text{eligible for closeout liquidation}. \tag{17}$$

The inequalities are applied per position, with N computed from that market's exposure. Maintenance liquidations aim to restore health using typical price protections, while closeout liquidations prioritize speed and can execute without deviation checks.

4.3 Liquidation Flow

Liquidations are permissionless and follow a standard script:

1. A keeper submits a taker order marked as 'liquidation' or 'closeout'.
2. The engine verifies that the targeted position shrinks in size and that the relevant NAV test (maintenance or closeout) is already breached.
3. Maintenance fills obey the usual price band ($|P_{\text{match}} - P_{\text{index}}| \leq \text{maxDeviation}$), while closeout fills skip that constraint to guarantee execution.

4.4 Insolvency Management

Breaching the closeout test $\mathrm{NAV}_{\mathrm{account}} < N \times \mathrm{CMR}$ pivots the position into a closeout liquidation. The keeper submits a taker-side unwind, while the matching engine sources the maker leg directly from solvent counterparties with opposing exposure. This is the same auto-deleveraging (ADL) mechanism—there is no separate second phase once closeout begins.

Liquidation fills continue until the distressed position reaches zero size. After the terminal fill the engine computes the account's residual balance

$$\Delta_{\mathrm{terminal}} = B_{\mathrm{terminal}} + \sum_{p} \mathrm{uPnL}_{p}^{\mathrm{realized}}, \tag{18}$$

which is the cash that remains once every slot is closed and all margin is unlocked. The account is then zeroed and $\Delta_{\mathrm{terminal}}$ is swept into the protocol vault:

$$\mathrm{Sweep}_{\mathrm{closeout}} = \Delta_{\mathrm{terminal}}. \tag{19}$$

A positive sweep records a closeout fee collected from the liquidated trader. A negative sweep withdraws insurance to repay insolvency on their behalf. Either direction resolves through the same vault accounting entry, keeping the system solvent without socialized loss.

5 Fees & Funding

Cascade prices every fill according to a layered fee stack that rewards liquidity partners and keeps the mark price aligned with the underlying index.

5.1 Fee Stack

For any fill with notional size

$$\mathrm{Notional}_{\mathrm{fill}} = |\mathrm{fillAmount}_{t}| \times P_{\mathrm{match},t}, \tag{20}$$

the base fee is determined by the order's liquidity role:

$$\mathrm{BaseFee} = \begin{cases} \mathrm{Notional}_{\mathrm{fill}} \times \mathrm{makerFee}, & \text{maker order}, \\ \mathrm{Notional}_{\mathrm{fill}} \times \mathrm{takerFee}, & \text{taker order}. \end{cases} \tag{21}$$

Orders may also include optional priority fees that prepay specific counterparties. The full fee credited to each participant combines their share of BaseFee and any attached priority component.

5.2 Distribution Path

The base fee routes across three deterministic rails:

ProtocolSweep deposits into the protocol vault, backstopping insurance and funding operations.

OriginatorShare rewards the interface or integrator that sourced the order flow.

MatcherShare compensates the matcher that submitted the bundle.

Let ϕ_{orig} and ϕ_{match} denote the fractional shares for the originator and matcher respectively; the residual accrues to the protocol vault. The cash allocation per fill is therefore

$$F_{\text{orig}} = F_{\text{trade}} \cdot \phi_{\text{orig}}, \tag{22}$$

$$F_{\text{match}} = F_{\text{trade}} \cdot \phi_{\text{match}}, \tag{23}$$

$$F_{\text{protocol}} = F_{\text{trade}} \cdot (1 - \phi_{\text{orig}} - \phi_{\text{match}}). \tag{24}$$

Each ϕ is specified in fixed-point form alongside fee rates, with the constraint $\phi_{\text{orig}} + \phi_{\text{match}} \leq 1$.

Recipient	Share	Amount (USD)
Originator	15%	675
Matcher	20%	900
Protocol vault	65%	2,925

Table 1: Illustrative fee allocation for \$1,000,000 of matched notional at a blended trade fee of 45 bps.

Optional priority amounts for originators stack on top of these percentages, enabling referral programs and competitive routing without altering solvency guarantees.

5.3 Maker and Taker Incentives

Maker fees are tuned below taker fees to reward resting liquidity. The configuration lives in the market parameter set and can be updated directly by the owner role, letting governance adjust incentives per market without downtime. Because fill logic determines the side dynamically, traders see the fee rate that matches the actual liquidity effect of their order.

5.4 Funding Alignment

Funding payments exchange value between longs and shorts so that the contract mark price shadows the oracle index. The engine measures a simple premium

$$\text{premium} = \frac{P_{\text{mark}} - P_{\text{index}}}{P_{\text{index}}}, \tag{25}$$

clips it to a governance-defined bound maxPremium, and scales the result by the elapsed time since the last funding update:

$$\text{fundingRate} = \text{clip}(\text{premium}, -\text{maxPremium}, \text{maxPremium}) \times \frac{\Delta t}{\text{fundingInterval}}. \tag{26}$$

Δt time in hours since the previous funding update (target interval is eight hours).

fundingInterval constant defaulting to eight hours.

clip$(\cdot)$ operation that limits the premium to the configured range.

When a position changes, the difference between the current funding accumulator and the position's last recorded accumulator determines the transfer:

$$\text{FundingPayment} = \text{PositionSize} \times (\text{Accumulator}_{\text{new}} - \text{Accumulator}_{\text{old}}), \tag{27}$$

crediting shorts when funding is positive and longs when funding is negative. The payment is booked inside the position's funding ledger until the trader closes or realizes the exposure.

5.5 Referral Flow

Every order may specify an originator address. Originators earn a programmable share of the market fee alongside any optional priority fee provided by the trader. This structure lets front-ends, trading guilds, or routing networks monetize order flow while keeping settlement deterministic and transparent on-chain.

5.6 Net Trader Costs

Trader cashflows combine entry and exit fees with cumulative funding. For a long position that opens via taker flow and unwinds as a maker, the total outlay is

$$C_{\text{long}} = N_{\text{entry}} f_{\text{taker}} + N_{\text{exit}} f_{\text{maker}} + F_{\text{fund}}^{\text{long}}, \tag{28}$$

where $F_{\text{fund}}^{\text{long}} > 0$ when the mark trades rich to the index. For example, a 10 BTC long opened at \$50,000 with $f_{\text{taker}} = 50$bps, held for thirty days at an average $r_{\text{fund}} = 10$bps per period, and closed as maker at \$55,000 with $f_{\text{maker}} = 40$bps incurs

$$N_{\text{entry}} f_{\text{taker}} = 10 \cdot 50{,}000 \cdot 0.0005 = 250, \tag{29}$$

$$F_{\text{fund}}^{\text{long}} = 10 \cdot 50{,}000 \cdot 0.0001 \cdot 90 = 4{,}500, \tag{30}$$

$$N_{\text{exit}} f_{\text{maker}} = 10 \cdot 55{,}000 \cdot 0.0004 = 220. \tag{31}$$

The long thus spends \$4,970 over the holding window. A short facing identical fee terms would capture a funding rebate of \$4,500 when $P_{\text{mark}} < P_{\text{index}}$, turning fees into a net gain of \$4,030.

5.7 Protocol Revenue

Protocol revenue aggregates the residual fee share across all fills:

$$R_{\text{protocol}} = \sum_{\text{fills}} F_{\text{trade}} \cdot (1 - \phi_{\text{orig}} - \phi_{\text{match}}) \,. \tag{32}$$

This stream funds sequencer operations, oracle integrations, and insurance buffers via the protocol vault. Funding transfers, while crucial for price alignment, are strictly peer-to-peer and do not contribute to R_{protocol}.

6 Protocol Architecture

Cascade's design relies on a lean on-chain kernel paired with upgrade-safe coordination and optional vault tooling.

6.1 Core Contracts

`Perpetual` central contract that tracks positions, margins, and market parameters while enforcing every invariant during settlement.

`InvariantLib` rulebook invoked by `Perpetual` to gate signatures, price bands, margin sufficiency, and open-interest caps before any state change.

`MarginLib`, `PositionLib`, `FillLib` shared libraries that calculate notional values, update NAV, apply fills, and compute fees.

Together they provide deterministic custody: trades only post when all guards pass, and all collateral moves through a single accounting surface.

6.2 Parameter Administration

Per-market parameters live inside `Perpetual`. The governance lanes that tune them—including owner permissions, bounds, and escalation paths—appear in Section 9. This keeps the architecture overview focused on component roles while deferring policy specifics to the dedicated governance section.

6.3 Delegated Deposits

Delegation is native to `Perpetual`. Any account owner can whitelist depositors and managers, defining how much collateral each delegate may contribute or move. When a delegate deposits, funds credit the target account's balance immediately; trading authority follows the same allowance pattern without transferring ownership. Revoking a delegate freezes further deposits or position changes but leaves existing balances untouched, keeping capital composable without wrapper contracts.

6.4 Governance Token

The protocol token implements `ERC20Votes` with a capped exponential inflation schedule:

$$S_{\max}(t) = S_{\text{genesis}} \times e^{r_{\text{inflation}} \times \Delta t}. \tag{33}$$

$S_{genesis}$ token supply at launch.

$r_{inflation}$ annualized growth cap chosen by governance.

Δt elapsed time in years since launch.

Transfers are gated by default; governance can whitelist specific addresses or open transfers globally by allowing the zero address.

7 Orderbook & Settlement

Cascade separates price discovery from finality. Matchers and trading venues source liquidity off-chain, while settlement and risk enforcement stay fully on-chain.

7.1 Off-Chain Discovery

Orders are signed using EIP-712 digests so traders can authorize intent without paying gas. A typical flow is:

1. The trader signs a maker or taker order that embeds amount, price, expiry, flags, and optional fees.
2. Matchers collect signed orders, align maker and taker intents, and ensure both sides satisfy their stated limits.
3. Optional priority fees compensate the originator interface that sourced the order flow.

Matchers operate under a constrained trust model: they can choose matching sequences, but they cannot bypass the invariants that protect funds. Certain emergency paths, such as liquidations or ADL triggers, waive the signature requirement so any actor can restore solvency when accounts breach risk limits.

7.2 On-Chain Validation

When a matcher submits a trade bundle to `Perpetual`, the contract executes a deterministic checklist:

1. **Authenticate.** Verify signatures (unless the path is permissionless) and confirm the caller is an approved matcher if required.
2. **Price and Quantity Guards.** Ensure the maker-derived price respects both parties' limits and stays within the allowed deviation from the oracle index.
3. **Risk Checks.** Recompute margins, open-interest totals, and lot-size constraints to confirm the post-trade state remains valid.
4. **State Update.** Apply fills, adjust position basis, book fees, and emit events describing the change.

Liquidation-specific orders route through a dedicated branch that checks the breached maintenance or closeout threshold before permitting the unwind. Closeout flows skip solvency checks intentionally so the system can push bankrupt accounts into ADL resolution.

8 Oracles & Data Integrity

Cascade sources prices from the Stork oracle network and hardens them with deterministic validation. Every market caches the latest signed observation

$$\mathsf{OraclePrice} = \left(p_{\text{stork}}, t_{\text{stork}}\right) \tag{34}$$

for use in downstream guardrails. Parameter governance publishes per-market values for maxDelay so the system can enforce freshness without code changes.

8.1 Primary Oracle Flow

Incoming Stork updates travel through a short commit pipeline:

1. Decode the batch into feed-scoped payloads and reject mismatched lengths.
2. Confirm each payload's id matches the requested feed before state mutation.
3. Verify the ECDSA signature over $(\text{id}, t_{\text{stork}}, q_{\text{stork}}, \text{publisherRoot}, \text{algHash})$ using the published Stork key.
4. Apply the immutable decimal offset, convert nanosecond timestamps into seconds, and discard non-monotonic updates.

Only payloads that satisfy every check replace the cached observation and emit OraclePriceUpdated. The system rejects fills whenever $t_{\text{stork}} < t_{\text{now}} - \text{maxDelay}$, ensuring stale data never clears trades.

8.2 Matcher Pricing Inputs

Matchers compute a pair $(P_{\text{index}}, P_{\text{mark}})$ for every fill they submit. The algorithm is off-chain—typically blending the latest Stork observation with exchange-orderbook data and funding heuristics—but its outputs must survive on-chain validation:

1. P_{index} must be within the allowed deviation of the cached p_{stork}.
2. P_{mark} must respect the same deviation bound when compared to P_{index}.
3. Both quotes inherit the maxDelay freshness constraint from the underlying Stork timestamp.

When a bundle settles, `Perpetual` records the accepted $(P_{\text{index}}, P_{\text{mark}})$ pair and emits them inside the fill event so downstream risk engines can reproduce funding and NAV calculations. Because the matcher recomputes fresh values for every fill, the protocol no longer maintains exponential moving averages on-chain.

8.3 Safety Controls

The simplified price stack integrates directly with Cascade's invariants.

- **Deviation Guard.** Fill validation binds the clearing price to the matcher-supplied P_{index} through maxDeviation, rejecting trades where $\max(P_{\text{match}}, P_{\text{index}}) / \min(P_{\text{match}}, P_{\text{index}}) > 1 + \text{maxDeviation}$.
- **Margin & Solvency.** Maintenance and closeout requirements, unrealized PnL, and liquidation NAV tests all consume the recorded P_{index} and P_{mark}, keeping leverage calculus consistent across fills.

- **Auditability.** Fill events log only the matcher-supplied mark and index pair, reflecting the trusted publisher's view of the market. Indexers rely on those emissions (and any off-chain disclosures of the underlying oracle payloads) to reproduce funding and NAV calculations at any block height.

This combination of authenticated primary feeds and matcher-derived quotes keeps Cascade live during short-lived oracle outages while remaining tightly anchored to observed market conditions.

9 Governance & Upgradability

Cascade's control plane partitions immutable settlement logic from configurable market levers. Governance therefore centers on parameter selection rather than contract replacement, keeping execution deterministic while allowing administrators to tune risk appetite and incentive surfaces.

9.1 Contract Roles

Three contracts form the on-chain governance perimeter. The relationships are summarized in Table 2.

Component	Responsibility	Mutability
`Perpetual.sol`	Position accounting, margin checks, settlement, delegation	Immutable
Delegation registry	Per-account deposit/trade allowances embedded in `Perpetual`	Mutable by account owners
Owner (DAO / multisig)	Parameter custody, event emission, upgrade gating	Configurable signers

Table 2: Governance footprint of the core components. Administrative surfaces live inside `Perpetual` and the owner address.

The `Perpetual` contract owns all user collateral, maintains the account state $(B, \mathcal{P}, M)$ from Section 3, and enforces invariants via a bundled `InvariantLib`. Parameter updates now execute directly through the owner address, which can be implemented as a DAO or multisig with optional timelock. Delegated deposits and trading rights are simply entries in the embedded registry, so capital can route between participants without auxiliary wrappers.

9.2 Parameter Surface

Every listed market advertises a parameter vector Θ_i identified by feed i. Parameters are encoded in fixed-point form with six decimals of precision and can be grouped as

$$\Theta_i = (\theta_i^{\text{margin}}, \theta_i^{\text{size}}, \theta_i^{\text{fund}}, \theta_i^{\text{price}}, \theta_i^{\text{fee}}, \theta_i^{\text{state}}), \tag{35}$$

where

$$\theta_i^{\text{margin}} = (m_{\text{init}}, m_{\text{maint}}, m_{\text{close}}, m_{\text{default}}), \tag{36}$$

$$\theta_i^{\text{size}} = (q_{\min}, q_{\text{lot}}, q_{\max\text{-}oi}), \tag{37}$$

$$\theta_i^{\text{fund}} = (T_{\text{fund}}, C_{\text{fund}}), \tag{38}$$

$$\theta_i^{\text{price}} = (\delta_{\max}), \tag{39}$$

$$\theta_i^{\text{fee}} = (f_{\text{taker}}, f_{\text{maker}}, \phi_{\text{orig}}, \phi_{\text{match}}, \alpha_{\text{orig}}, \alpha_{\text{match}}), \tag{40}$$

$$\theta_i^{\text{state}} = (\chi_{\text{cross}}, \chi_{\text{reduce}}). \tag{41}$$

Here α_{orig} and α_{match} denote the payout addresses associated with the fee shares, while $\chi_{\text{cross}}, \chi_{\text{reduce}} \in \{0, 1\}$ toggle cross-margin eligibility and reduce-only pauses. The settlement engine consults Θ_i on every fill and margin event; absent administrator intervention, Θ_i remains static.

9.3 Governance Controls

A single owner address $\mathcal{O}$ authorizes every mutation of Θ_i. In production deployments $\mathcal{O}$ is expected to be implemented as a DAO-controlled multisig or governor with an optional timelock so stakeholders can observe queued transactions. Each setter clamps proposed values to closed intervals

$$p \in [p_{\min}, p_{\max}], \tag{42}$$

baked into `Perpetual`; attempts outside the allowed range revert on-chain. This keeps operational desks within pre-approved risk envelopes while retaining an emergency path for $\mathcal{O}$ to force actions such as $\chi_{\text{reduce}} = 1$ when markets require rapid de-risking.

9.4 Event Transparency & Upgradability

Every accepted update emits a `FeedUpdated` event containing the feed identifier, parameter slot, and new value. Off-chain risk engines and discovery services subscribe to this feed to rebuild the latest Θ_i snapshot, guaranteeing that quote surfaces stay synchronized with on-chain truth. Because the core contracts are immutable, protocol evolution occurs through three levers:

(a) onboarding new markets by instantiating additional Θ_i,

(b) refining governance processes around $\mathcal{O}$ and $\mathcal{U}$, including multi-signature or timelock wrappers, and

(c) coordinating full contract migrations via opt-in upgrades, preserving legacy state until adoption thresholds are met.

This architecture confines upgrade risk to a narrow, auditable surface while preserving the deterministic guarantees that underpin Cascade's settlement core.