Signals Protocol v1: CLMSR-based Daily Bitcoin Range Markets

Signals Research Team
jaegun@signals.wtf, danny@signals.wtf
Signals Protocol

December 10, 2025

Abstract

Prediction markets promise an efficient way to aggregate dispersed information into prices, but most on-chain designs inherit a structural limitation from P2P binary markets: as soon as outcomes become continuous, liquidity fragments across coarse strikes and users lose the ability to express precise views. Range partitions, constant-product AMMs, and ad-hoc orderbooks alleviate parts of the problem, but none provide full-range betting with unified liquidity, bounded maker loss, and an operationally tractable capital stack.

Signals Protocol addresses this gap by implementing daily BTC range markets on top of a single cost-function market maker, CLMSR (Continuous LMSR). The price axis $[L, U)$ is discretized into ticks via an `OutcomeSpec`, and one global potential $C(q) = \alpha \ln Z(q)$ prices all ticks and ranges simultaneously. Traders buy arbitrary contiguous price ranges as atomic claims; the maker's P&L is path-independent and fully determined by the start and end CLMSR states, with a closed-form worst-case loss bound $O(\alpha \ln n)$ and exact range–binary equivalence.

Maker capital is packaged into a daily *Market-Cycle Batch* with an LP Vault. Each day's trading and settlement compresses CLMSR P&L and gross fees into a single scalar that is split across LPs, a Backstop Vault, and Treasury via a fixed Fee Waterfall. A batch mint/burn rule ensures that any two LP shares that underwrite the same sequence of daily markets earn exactly the same return, regardless of when deposits and withdrawals are requested within the day. On top of this, a Safety Layer ties depth to LP capital through entropy budgets, enforces a hard daily drawdown floor using Backstop Grants, and adds withdrawal lags plus a constrained LOLR-style role for Treasury. A Performance Layer then chooses depth, priors, and fee policy strictly inside this safe region to target a long-run Fee–Cost inequality while maintaining acceptable slippage for traders.

Finally, we describe an oracle and settlement architecture based on signed pull feeds with explicit timing windows, together with an implementation blueprint for an EVM rollup using immutable CLMSR math libraries, segment-tree-based distribution storage, and WAD-fixed arithmetic with explicit rounding invariants. The result is a solvency-aware, continuous-outcome prediction protocol that can be deployed as a real system and extended beyond single-asset daily BTC markets.

1 Introduction: Problem & Product

1.1 The P2P Binary Orderbook Model

Most on-chain prediction markets still rely on a **P2P (peer-to-peer) binary orderbook** model. The venue stays neutral by matching offsetting orders between participants so that directional risk is always transferred in a zero-sum fashion: one trader's loss is another's gain.

This structure works well for binary events with clearly defined YES/NO outcomes (e.g., "Will Bitcoin's price exceed $100,000 on December 31, 2024?"). Participants bet on one of two discrete outcomes, the engine pairs opposing orders on the same line, and the platform does not warehouse risk.

1.2 Structural Limitation on Continuous Outcomes

For **continuous variables** such as asset prices, the same P2P model runs into a structural limitation. The outcome space is a continuum; traders typically have views on specific ranges, not just on whether price ends up "above" or "below" a single strike.

Two effects appear:

- **Failure of coincidence of wants.** A trader may want to buy a narrow range such as \$60,100–\$60,200. Finding a counterparty who wants to sell that exact range, in the opposite direction, at the same time is unlikely. As the set of possible ranges grows, exact matching becomes harder.
- **Liquidity fragmentation.** To avoid this matching problem, venues usually list a small number of coarse strikes (e.g., "above \$50k", "above \$60k") and ignore the rest of the price axis. Liquidity is forced into these listed points, but information about the shape of the distribution between them is lost.

In practice, this means that P2P binary orderbooks do not scale well to full-range price discovery. They either fragment liquidity across many thin markets or compress information into a few wide buckets.

1.3 The Liquidity–Cost Trade-off (The CLMSR Solution)

Signals replaces the P2P matching model with a CLMSR-based automated market maker. Instead of searching for a counterparty for each range, a single cost-function maker takes the other side of all range trades.

This introduces an explicit trade-off:

- **Benefit.** A single CLMSR potential $C(q) = \alpha \ln Z(q)$ (Section 2) prices all ticks and ranges simultaneously. Traders can buy or sell arbitrary ranges as atomic claims, and liquidity is unified across the entire OutcomeSpec rather than fragmented across many orderbooks.
- **Cost.** The maker bears entropy cost and inventory risk. With a uniform prior, CLMSR's worst-case loss is bounded by $\alpha \ln n$ (Section 2). This bound increases with both depth α and resolution n; deeper and finer markets are more useful but more expensive to underwrite.

The design accepts this trade-off explicitly. Maker capital is packaged into an LP Vault (Section 3). A Safety Layer constrains worst-case loss and daily drawdown via entropy budgets, depth caps, and a Backstop Vault (Section 4). A Performance Layer then chooses depth, priors, and fee policy inside that region so that long-run fee flow is intended to cover entropy cost while maintaining usable slippage for traders (Section 5).

1.4 Maker Capital and the Role of LPs

A CLMSR maker must always stand ready to take the other side of range trades. Every time a trader buys a range R, the maker implicitly sells that range and accumulates the opposite position. Maintaining one global potential $C(q)$ with usable depth therefore requires **continuous maker capital** willing to bear entropy cost and inventory risk.

In principle, this capital could be provided entirely by a single protocol-owned pool. Signals instead packages the maker side into an LP Vault, allowing external LPs to supply capital and share in both risk and return. The reasons are primarily economic:

- **Depth scalability.** Daily CLMSR depth α must be large enough to keep slippage usable on the chosen OutcomeSpec. Solely protocol-owned capital would bottleneck depth and volume; sharing the maker role with external LPs lets depth scale with demand and capital inflow.
- **Explicit risk sharing.** By tokenizing maker capital as vault shares, we make the maker P&L path investable with a simple daily rule: each day's trading P&L, the portion of fees attributed

to LPs, and any tail-risk support from the Backstop are all reflected in the vault's net asset value before a single batch price is applied. Sections 3–4 formalize this accounting and the loss limits in precise terms. LPs opt in when they accept this risk–return profile under those constraints.

- **Separation of long-term insurance.** A dedicated Backstop Vault owns tail segments outside the LP design range, funded by a carved-out fee share, while Treasury plays a separate LOLR and operating-capital role (Section 4.2). This separation keeps the LP product economically clean while still allowing the protocol to manage systemic tail risk.

Concretely, LPs act as the CLMSR **equity tranche**: they primarily absorb entropy cost and routine volatility within the Safety Layer's design range ($\alpha_{\text{limit},t}$ and p_{dd}), and in return hold vault shares with a residual claim on the Fee Waterfall. Backstop truncates tails beyond that range by injecting capital on tail days when needed to respect the drawdown limit, and Treasury can intervene as LOLR in liquidity shocks. Section 3 formalizes how maker capital is packaged into a daily Market-Cycle Batch, and Section 4 specifies how the Safety Layer bounds the LP risk share.

1.5 Operational Framework and Organization

This document presents a comprehensive framework for implementing the CLMSR mechanism as an on-chain protocol.

Operational Framework Signals operates with **day-ahead** markets as the basic unit. To this end, the protocol defines the following core components:

- **Market Spec & Trading:** Discretization of the price axis $[L, U)$ and range trading interface.
- **Liquidity & Fees:** Ensuring sustainability through liquidity provision, the Fee policy, and the Fee Waterfall.
- **Settlement & Risk:** Oracle-based settlement process and risk guardrails for abnormal situations.

Terminology Throughout this paper we distinguish two separate concepts:

- **Fee policy** – the rule (and its parameters) that decides how much fee is charged on each trade and settlement during a day and aggregates those charges into a single gross daily fee figure.
- **Fee Waterfall** – a fixed-priority rule in the Safety Layer that takes that gross daily fee and splits it across loss compensation, Backstop reserves, LPs, and the protocol Treasury, and that may also route capital from the Backstop to the LP Vault on tail days when enforcing the drawdown floor.

The Waterfall's full specification (formulas and rounding rules) lives in the Safety Layer (Section 4.3 and Appendix C). The Performance Layer controls the Fee policy that generates the gross daily fee; it does not change the Waterfall priority itself.

Organization This whitepaper proceeds from mechanism foundations to governance and operations:

- **Sections 1–7:** Mechanism and implementation — Introduction; CLMSR engine; Market-Cycle Batch & LP Vault; Safety and Performance layers; Oracle & Settlement; and System Architecture.
- **Sections 8–10:** Governance and operations — Governance model & roles; change process, emergency measures, decentralization roadmap, and Conclusion & Outlook.
- **Appendices A–D:** Notation table, parameter catalog, numerics/rounding rules, and implementation/testing notes.

Scope and Reference This document specifies the **protocol/operational layer**. Mechanism-level definitions and proofs (e.g., CLMSR potential function, loss offset, rounding rules, segment tree invariants) use a separate CLMSR mechanism whitepaper[1] as the primary reference, and here we specify the procedures and parameters for operating that semantics on top of a daily BTC range market (single basis price batch, risk/oracle/governance layers).

Reading paths Readers with different roles can safely skip some sections:

- LPs and Backstop providers who are not concerned with CLMSR proofs may skim Section 2 and focus on Sections 1 and 3–5.
- Engineering-focused readers may read Sections 2–3 and 7 first, then return to Sections 4–5.
- Governance and operations participants may focus on Sections 1, 3–5, 8–9 and consult the appendices as needed.

The intermediate NAV and price variables used in the vault and Safety accounting ($N_t^{\text{raw}}, N_t^{\text{pre}}, P_t^{\text{raw}}, P_t^{\text{e}}$, etc.) are defined in Sections 3–5 and summarized in Appendix A.

2 CLMSR Pricing Engine

2.1 Theoretical Foundation and the Position of CLMSR

CLMSR is a structure that applies Hanson's LMSR to continuous outcomes by using **a single cost function discretized into tick units**. Since Section 1 established why we chose range-type markets and the necessity of a single potential function, here we clearly describe only the mathematical definitions and properties.

2.2 Outcome Space and Cost Function Definition

With **OutcomeSpec** (L, U, s, d), we divide $[L, U)$ into a uniform tick set $\mathcal{B} = \{0, \ldots, n-1\}$ where $n = (U-L)/s$. By design, OutcomeSpec is defined so that $U - L$ is divisible by s, making all tick widths equal. d is the number of fixed decimal places (decimals) for internal settlement input, determining the price unit (OutcomeUnit) in which L, U, s, and oracle observations x are expressed.

Each daily market t ultimately resolves to a single tick index $\tau_t \in \mathcal{B}$ (the settlement tick) obtained from the oracle via OutcomeSpec's tick mapping (Section 6). For any payoff vector $f \in \mathbb{R}^n$, interpret f_b as the amount paid if $\tau_t = b$, so the realized payout is f_{τ_t}. A range ticket with contiguous tick range $R \subseteq \mathcal{B}$ and quantity $x \geq 0$ corresponds to the payoff vector $f = x \cdot \mathbf{1}_R$ and has settlement payout

$$\text{Payout}(R, x; \tau_t) := x \cdot \mathbb{1}[\tau_t \in R].$$

The CLMSR state is a vector $\mathbf{q} = (q_0, \ldots, q_{n-1})$ representing cumulative issuance per tick. The cost function is

$$C(\mathbf{q}) = \alpha \ln Z(\mathbf{q}), \qquad Z(\mathbf{q}) = \sum_{i=0}^{n-1} e^{q_i/\alpha},$$

where $\alpha > 0$ is both the depth and the loss unit. Marginal prices/probabilities are

$$p_i(\mathbf{q}) = \frac{\partial C}{\partial q_i} = \frac{e^{q_i/\alpha}}{\sum_k e^{q_k/\alpha}}.$$

The partition sum Z binds all ticks together, enforcing $\sum_i p_i = 1$ and $p(B) = \sum_{b \in B} p_b$. If there is only one tick, it becomes classical LMSR. With this single equation, the meaning of "price = probability" is simultaneously defined across the entire range. The differential form is $dC = \sum_i p_i \, dq_i$, and since it is a conservative field generated by a single potential function, any closed path satisfies $\oint dC = 0$.

[1] CLMSR Mechanism Whitepaper, 2025.

Payoff pricing. The price of an arbitrary payoff vector $f \in \mathbb{R}^n$ with settlement payout f_{τ_t} is

$$\pi(f; \mathbf{q}) := \sum_b f_b \, p_b(\mathbf{q}).$$

For a range ticket (R, x), $f = x \cdot \mathbf{1}_R$ so

$$\pi(R, x; \mathbf{q}) = x \cdot \sum_{b \in R} p_b(\mathbf{q}).$$

Buying x of range R is exactly the range trade defined in Section 2.3.

2.3 Range-Binary Equivalence

Theorem 1 (Partition Reduction) Adding quantity x to any range $R \subset \mathcal{B}$ gives

$$\Delta C(R, x) = \alpha \ln \frac{Z_{\overline{R}} + e^{x/\alpha} Z_R}{Z_{\overline{R}} + Z_R},$$

($Z_R = \sum_{b \in R} e^{q_b/\alpha}$, $Z_{\overline{R}} = Z - Z_R$). This is exactly the same as the cost of buying x on one side in a binary LMSR that treats the event as only $\{R, \overline{R}\}$. If there is only one tick, it reduces to LMSR. Interpretation: Applying the multiplier $e^{x/\alpha}$ to R changes the partition sum to the numerator above, and this is exactly the binary LMSR cost formula. Since range orders are always reduced to a single binary market $\{R, \overline{R}\}$, the entire range is atomically updated and normalized. **Proof outline.** Let the weight sum outside R be $Z_{\overline{R}}$ and inside be Z_R, and multiplying R by $e^{x/\alpha}$ changes the partition sum to $Z_{\overline{R}} + e^{x/\alpha} Z_R$. The cost formula $\alpha \ln(Z_{\text{after}}/Z_{\text{before}})$ when one side's issuance is increased by x in a binary LMSR exactly matches the above. A more rigorous theorem is presented in the Partition Reduction theorem of the CLMSR mechanism whitepaper.

Price update. After buying x on R, the prices become

$$p_b(q') = \begin{cases} \dfrac{e^{x/\alpha} p_b(q)}{1 - \lambda_R + e^{x/\alpha} \lambda_R} & b \in R, \\ \dfrac{p_b(q)}{1 - \lambda_R + e^{x/\alpha} \lambda_R} & b \notin R, \end{cases}$$

where $\lambda_R = \sum_{b \in R} p_b(q)$. The softmax Jacobian gives local sensitivity $\frac{\partial p_b}{\partial q_j} = \frac{p_b}{\alpha}(\delta_{bj} - p_j)$.

2.4 Path Independence and Loss Bound

Path independence: Since C is a function of state only,

$$\Delta C = C(q_{\text{final}}) - C(q_{\text{initial}}).$$

The cost is determined solely by the start/end states, regardless of what order of trades occurred. Traders cannot create additional profit by reordering trades, and makers can design risk with just "initial state $\rightarrow$ target state," which becomes the basis for Section 4's risk framework (setting the initial state near the expected realization value). In other words, a trader's series of buy/sell P&L is also determined only by the start/end states, so they cannot create "round-trip arbitrage" through intermediate paths.

Bounded loss: Starting from uniform prior $q = 0$,

$$L_{\max} \leq \alpha \ln n.$$

The worst case is when mass concentrates on one tick so $Z \rightarrow 1$, dropping from $C(0) = \alpha \ln n$ to 0. Increasing n improves resolution but raises the bound by $\ln n$; increasing α reduces slippage but enlarges the loss unit (controlled by the Safety/Performance budgets in Section 4). This is the bound for a *uniform prior distribution*; for a non-uniform initial state q^0,

$$L_{\max}(q^0) \leq C(q^0) - \min_j q_j^0$$

(when mass concentrates on the minimum q_j^0 tick). In Section 4, we set q^0 near the expected realization value to reduce this exposure, acknowledging that loss can be larger if the prior misses.

2.5 Price Sensitivity and Slippage

When the current mass of range R is $\lambda_R = Z_R/Z$,

$$\Delta C^+(R, x) = \alpha \ln\big(1 - \lambda_R + e^{x/\alpha}\lambda_R\big).$$

The smaller λ_R is, the more nearly linear the cost; as $\lambda_R \to 1$, it grows exponentially, making it sharply more expensive to shift more weight to extreme ranges. Price impact is proportional to $\lambda_R(1-\lambda_R)$, most sensitive at mid-mass. This means we can adjust UX slippage and maker risk with α and tick resolution.

2.6 Units and Semantics

External settlement currency is USDC 6 decimals; internal math is WAD (10^{18}). We allow *one conversion, one rounding* per direction and impose fees outside the cost function so p keeps its probability semantics. Full numeric/rounding rules are defined canonically in Appendix C (see Appendix C.1); this subsection is a summary only.

2.7 Implementation pointers (see Section 7)

Segment trees, precision, rounding, and other on-chain implementation guards are covered in Section 7 and Appendix D. This section focuses on CLMSR cost/price definitions and mathematical properties (normalization, range-binary equivalence, path independence, loss bound).

3 Market-Cycle Batch & LP Vault

Signals packages CLMSR maker capital into a daily **Market-Cycle Batch** with an LP Vault. Each daily BTC market t is one batch: traders move the CLMSR state from $q_{\text{start},t}$ to $q_{\text{end},t}$; the protocol computes maker P&L and fees for that day; and all LP shares that underwrote that market move together to a new basis price. Deposits and withdrawals are only executed once per batch, at that daily equity price.

The batch exists to formalize three simple facts:

- maker P&L for a day is only well defined after settlement,
- fees belong to the capital that actually carried that risk from the first trade to settlement,
- LP shares that experienced the same markets should see the same return, regardless of when their owner sent deposit or withdrawal requests.

We now specify the minimal vault state variables and the batch mint/burn rules that implement these entitlements.

3.1 Why the Market-Cycle Batch Exists

For a single-owner CLMSR maker, daily P&L is straightforward. Given CLMSR states

$$q_{\text{start},t} \to q_{\text{end},t},$$

the maker's P&L for daily market t is

$$L_t := C(q_{\text{end},t}) - C(q_{\text{start},t}),$$

fully determined by the endpoints of the day's trading path. Section 2 already established that C is path-independent; intermediate trade ordering does not matter.

Once multiple LPs share maker capital, a basic question must be answered:

> Which shares are entitled to which day's P&L and fees?

Signals takes a simple, economic stance:

- LPs **earn trading fees because they stand ready to settle traders at market close**. They supply collateral so the protocol can guarantee payouts to winning positions. That entitlement arises from underwriting the market, not from holding the token abstractly.
- Underwriting a daily market t means **being in the vault from the first trade of t until its settlement**. Only those shares should absorb L_t and receive the LP fee share F_t (the LP-attributed fee term produced by the Safety Layer's Fee Waterfall; see Section 4.3 for the formal definition). Shares that arrive after settlement see the new basis price but do not retro-claim fees from markets they never underwrote.
- Conversely, a share that was present while fees were being collected for market t must also absorb the corresponding settlement risk for that same market. There is no fee without the obligation to be settled.

Without a batch abstraction, a continuously minting/burning vault would face an unattractive choice:

- either let capital that arrives after most of the day's risk has already been realized "buy into" that day's P&L and fee stream, diluting earlier LPs, or
- track a separate exposure schedule for each deposit and withdrawal so that every unit of capital has its own list of markets it underwrote.

The first option violates fee entitlement; the second option rebuilds a complex clearing engine on top of a mechanism that was explicitly designed to be summarized by a single potential function $C(q)$.

The **Market-Cycle Batch** is the minimal structure that avoids both outcomes. Each daily market t is treated as one indivisible unit of maker risk and fee accrual:

- all shares that are already issued when t opens and are still present when t settles move together from basis price P_{t-1} to a new equity price P_t^e,
- deposits and withdrawals affecting which future markets a share underwrites are only executed *after* that move, all at P_t^e.

Everything else in this section is accounting detail to implement that principle.

3.2 One-Day Timeline for LPs

A single day t can be read as the lifecycle of one batch from an LP's point of view.

1. **Start of day t: previous batch closed**
   At the end of batch $t-1$, the LP Vault is summarized by
   $$(N_{t-1}, S_{t-1}, P_{t-1}), \quad P_{t-1} := \frac{N_{t-1}}{S_{t-1}}.$$
   This basis price P_{t-1} is the value per share after all P&L and flows from markets up to $t-1$ have been applied and after deposits and withdrawals for batch $t-1$ have cleared.
2. **Trading phase for daily market t**
   - The CLMSR engine opens with OutcomeSpec (L, U, s, d) and state $q_{\text{start},t}$.
   - Traders buy and sell ranges; the CLMSR state moves
     $$q_{\text{start},t} \to q_{\text{end},t},$$
     and gross trading fees $F_{\text{tot},t}$ accumulate according to the Fee policy.

- The vault accepts new deposit and withdrawal *requests* during this time, but

$$N_{t-1},\ S_{t-1},\ P_{t-1}$$

  do not change. Requests are only queued.

From the LP's perspective, submitting a deposit during day t means "start underwriting from the *next* batch that processes me"; submitting a withdrawal means "stop underwriting from the *first* batch that executes my request." Which concrete batch that is depends on the withdrawal lag in Section 3.6.

3. **Market close and daily maker cashflows**
When daily market t closes, the CLMSR endpoint state $q_{\mathrm{end},t}$ and the accumulated gross-fee meter $F_{\mathrm{tot},t}$ are fixed. Maker P&L is then computed from the endpoints:

$$L_t = C(q_{\mathrm{end},t}) - C(q_{\mathrm{start},t}).$$

These raw quantities, together with the current capital-stack state, are handed to the Safety Layer (Section 4), which computes a single net LP-vault credit Π_t and corresponding Backstop/Treasury flows for batch t.

4. **Apply P&L to existing shares**
The total maker-side P&L accruing to the LP Vault for day t is denoted Π_t. From the vault's standpoint, Π_t is a single scalar net credit produced by the Safety capital-flow mapping for market t (formalized in Section 4). This net amount is applied to the previous NAV:

$$N_t^{\mathrm{pre}} := N_{t-1} + \Pi_t,$$

while the share count is still S_{t-1}. The **batch equity price** that all shares which underwrote market t should jump to is

$$P_t^e := \frac{N_t^{\mathrm{pre}}}{S_{t-1}}.$$

5. **Execute withdrawals and deposits at the batch price**
At this point the day's P&L is fully reflected in P_t^e. The batch then:

   - executes all withdrawal requests whose lag has expired, burning shares at price P_t^e and paying out assets,
   - mints new shares for all deposits assigned to this batch, also at P_t^e.

   After processing all such requests, the vault ends at

$$(N_t, S_t, P_t), \quad P_t := \frac{N_t}{S_t}.$$

   The next daily market $t+1$ starts from this state.

After batch t closes, the Safety Layer recomputes drawdown DD_t and derives the next-cycle depth caps $\alpha_{\mathrm{base},t+1}$ and $\alpha_{\mathrm{limit},t+1}$ from capital $E_{t+1} := N_t$ and drawdown state (Sections 4.4–4.5). The Performance Layer then observes $(L_t, F_t, G_t, \psi_{\mathrm{fee},t}, \Delta p_t)$ to choose $(\alpha_{t+1}, q_{0,t+1}, \text{Fee policy})$ for the next day within those caps (Section 5).

From this timeline, each LP share can be described purely by **which sequence of daily markets it underwrote**. Any two shares that passed through the same list of batches see exactly the same sequence of equity prices P_t^e, regardless of when their owner clicked "deposit" or "withdraw" inside each day.

3.3 Vault State and Notation

At the end of each batch t the LP Vault is summarized by a small set of state variables:

- N_t: net asset value (NAV) of the vault, including CLMSR positions marked at cost, accumulated fees, and any Backstop Grants received.

- S_t: total LP shares outstanding.
- $P_t := N_t/S_t$: basis price per share.
- P_t^{peak}: running peak of P_t over all past batches.
- $\text{DD}_t := 1 - P_t/P_t^{\text{peak}}$: drawdown from peak, used by the Safety Layer to damp depth and size Backstop Grants.

During trading in daily market t:

- $(N_{t-1}, S_{t-1}, P_{t-1})$ stays fixed from the vault's standpoint; only the CLMSR state q and the gross fee meter $F_{\text{tot},t}$ evolve.
- Deposit and withdrawal requests are recorded in queues and assigned to future batches; they do not change N or S yet.

After market t closes and Safety has applied its capital flows, the LP-facing net LP-vault credit Π_t for batch t is known (see Section 4.3 for its decomposition into CLMSR P&L, LP fees, and Backstop Grants). The **pre-batch** NAV and equity price that apply to the shares that underwrote market t are defined as

$$N_t^{\text{pre}} := N_{t-1} + \Pi_t, \qquad P_t^e := \frac{N_t^{\text{pre}}}{S_{t-1}}.$$

3.4 Daily Maker P&L from CLMSR and Fees

This subsection connects CLMSR state changes and fee charging to the daily net vault credit Π_t used in LP-vault accounting.

CLMSR maker P&L. For daily market t, trading moves the CLMSR state

$$q_{\text{start},t} \to q_{\text{end},t}.$$

With cost function $C(q) = \alpha_t \ln Z(q)$ (Section 2), maker P&L is

$$L_t := C(q_{\text{end},t}) - C(q_{\text{start},t}).$$

When the cost decreases the maker loses; with a uniform prior $q_0 = 0$, the loss is bounded by $\alpha_t \ln n$. For general priors $q_{0,t}$, the conservative entropy budget $E_{\text{ent}}(q_{0,t})$ is defined in Section 4 and is used only by the Safety Layer to cap depth and Backstop exposure. Define realized entropy loss

$$L_t^- := \max(-L_t, 0).$$

Interface to Safety and vault accounting. The Fee policy aggregates per-trade and settlement fees into a gross daily fee $F_{\text{tot},t} \geq 0$ (Section 5). The Safety Layer maps raw market outputs $(L_t, F_{\text{tot},t})$ and the current capital-stack state into:

- a net LP-vault credit Π_t applied at batch finalization, and
- internal Backstop/Treasury flows and grants that maintain the Safety invariants.

By definition (Section 4.3),

$$\Pi_t = L_t + F_t + G_t,$$

where F_t is the LP-attributed fee term and $G_t \geq 0$ is the Backstop Grant used to enforce the daily drawdown floor. From the vault's standpoint, only Π_t enters the pre-batch NAV identity

$$N_t^{\text{pre}} - N_{t-1} = \Pi_t,$$

with the internal split into F_t and G_t treated as Safety-layer detail. This identity is the core accounting equation used throughout the document: Section 3.2 uses only the Π_t side when describing the vault's one-day timeline, while Section 4.3 specifies how Π_t decomposes into L_t, F_t, G_t.

3.5 Batch Mint/Burn Rules and Invariants

Once N_t^{pre} and $P_t^e = N_t^{\text{pre}}/S_{t-1}$ have been set for batch t, the vault must process all eligible withdrawals and deposits **without disturbing** this equity price. This subsection specifies the mint/burn rules that guarantee that property.

Price invariance under deposits and withdrawals. Consider any vault state (N, S) and a fixed basis price

$$P := \frac{N}{S}.$$

Deposits and withdrawals during a batch that uses price P are defined as:

- **Deposit** of settlement-token amount D:

$$(N, S) \longrightarrow (N', S') = (N + D,\ S + D/P).$$

- **Withdrawal** of x shares:

$$(N, S) \longrightarrow (N', S') = (N - xP,\ S - x).$$

For both operations,

$$\frac{N'}{S'} = \frac{N + D}{S + D/P} = \frac{N}{S} = \frac{N - xP}{S - x} = P,$$

so the basis price remains exactly P, regardless of the order in which deposits and withdrawals are applied.

In batch t, deposits and withdrawals are processed using $P = P_t^e$ and initial state $(N_t^{\text{pre}}, S_{t-1})$. Because every request uses the same price and the update rules above, any sequence of valid operations leads to the same final ratio $N_t/S_t = P_t^e$. Operational ordering does not create opportunities for timing arbitrage among LPs.

Rounding and integer arithmetic details (e.g., floor/ceil when converting between asset and share units) follow these formulas as closely as possible and are specified canonically in Appendix C. At the level of economic semantics, they preserve the invariance $N/S = P_t^e$.

Batch algorithm. Putting the pieces together, the Market-Cycle Batch for day t is one atomic procedure:

1. **Settle the market and compute raw P&L inputs**
   - Use the oracle to obtain settlementValue and settlementTick for market t (Section 6).
   - Compute maker P&L L_t from CLMSR state change $q_{\text{start},t} \to q_{\text{end},t}$.
   - Aggregate gross fees $F_{\text{tot},t}$ from trades and settlements.
2. **Apply Safety capital flows (Fee Waterfall + Grant rule)**
   - Run the Safety Layer's Fee Waterfall and grant rule (Sections 4.3, 4.6) on $(L_t, F_{\text{tot},t}, N_{t-1}, B_{t-1})$.
   - Record the resulting net LP-vault credit Π_t and the internal tranche flows $(F_t, G_t, F_{\text{BS},t}, F_{\text{TR},t})$.
   - Update the LP Vault's pre-batch NAV using Π_t:

$$N_t^{\text{pre}} := N_{t-1} + \Pi_t,$$

     leaving the share count at S_{t-1}.
   - In parallel, update Backstop and Treasury NAVs using $(F_{\text{BS},t}, F_{\text{TR},t}, G_t)$ as specified in Section 4.

3. **Fix the equity price for the batch**
   Compute
   $$P_t^e := \frac{N_t^{\text{pre}}}{S_{t-1}}.$$
   This is the only price used for processing withdrawals and deposits in batch t.

4. **Process matured withdrawals at P_t^e**
   For each withdrawal request of x shares whose lag has expired (Section 3.6), burn x shares and pay out assets corresponding to xP_t^e, subject to integer rounding. Update $(N, S) \mapsto (N - xP_t^e, S - x)$ for each such request. Requests whose delay has not elapsed remain in the queue for future batches.

5. **Process deposits assigned to this batch at P_t^e**
   Aggregate all deposit amounts D_t^{pend} assigned to batch t. Convert them into shares using the mint rule at price P_t^e, updating
   $$(N, S) \mapsto (N + D_t^{\text{used}},\ S + \Delta S_t),$$
   where ΔS_t is the number of shares minted and D_t^{used} is the portion of deposits actually consumed (Appendix C). Any residual deposit dust is refunded immediately; the vault does not keep it.

6. **Record end-of-batch state**
   Set
   $$N_t := N, \quad S_t := S, \quad P_t := \frac{N_t}{S_t}.$$
   Update the running peak P_t^{peak} and drawdown DD_t used by the Safety Layer. Expose any metadata such as last settled market ID and timestamp for monitoring.

By construction:

- every LP share that was present throughout daily market t moves from P_{t-1} to P_t^e,
- deposits and withdrawals affect only which future markets a share underwrites,
- no LP can change their outcome for market t by trying to time intraday deposit/withdrawal operations.

3.6 Withdrawal Lag and Which Markets You Underwrite

For LPs it is natural to want flexible exits. For a CLMSR-based maker vault, however, *instant* redemption at current basis price would let LPs try to escape anticipated losses while still enjoying already accrued fees, which contradicts the underwriting principle.

Signals therefore ties exits explicitly to the batch structure:

- Any LP share that is present at the start of daily market t and still present when the batch for t runs must fully absorb that day's (L_t, F_t, G_t).
- A withdrawal request only affects **which future markets** the share will underwrite; it does not erase exposure to markets that are already in progress when the request is lodged.

To enforce this and to slow collective exits, Signals v1 uses a fixed **withdrawal lag** D_{lag}:

- A withdrawal request at time T_{req} becomes eligible in the first batch t' whose cycle time $T_{\text{cycle},t'}$ satisfies
  $$T_{\text{cycle},t'} \geq T_{\text{req}} + D_{\text{lag}}.$$
- The request is executed at the batch equity price $P_{t'}^e$, *after* applying $(L_{t'}, F_{t'}, G_{t'})$ for market t'. There is no lock on the basis price at request time.

Consequences:

1. **No "run ahead of the loss" option**
   An LP cannot observe an unfavorable CLMSR state or oracle movement, submit a withdrawal, and be guaranteed to exit at today's price before tomorrow's loss is realized. They remain exposed to the P&L of all markets whose batches occur before their request clears, including any Backstop Grants.
2. **Controlled outflow speed and reaction time**
   Even if many LPs request withdrawals after a drawdown, capital leaves the vault over at least a D_{lag}-sized window. The protocol has time to:
   - compress depth α_t and adjust fee policy,
   - consider Treasury LOLR interventions (Section 4.5),
   - and de-risk or pause markets if needed under governance,

   before depth collapses.

From the accounting perspective, withdrawal lag ensures that every executed withdrawal is cleanly tied to a single batch price $P_{t'}^e$, preserving the invariant that **all shares that underwrote the same list of markets see the same basis-price path.**

3.7 Token Representation and Secondary Liquidity

LP equity and trading positions carry different rights and lifecycles, so Signals represents them as different token types.

LP Vault shares (ERC-4626).

- ERC-4626 shares represent pro-rata claims on the vault NAV N_t.
- Each share entitles its holder to $1/S_t$ of the vault's assets at the end of batch t; its economic value is the basis price $P_t := N_t/S_t$.
- Deposits and withdrawals use the ERC-4626 interface but are executed according to the batch rules above: requests are queued during the day and processed once per batch at price P_t^e.
- LP shares are freely transferable and can form secondary liquidity on external DEXs or lending markets. The protocol does not enforce any relationship between secondary-market prices and on-chain basis price P_t; it only updates P_t once per batch.

Trading positions (ERC-721).

- When a trader buys range $R \subseteq B$ with quantity x in market t, the protocol issues a position NFT with metadata at least
  $$(\text{marketId},\ R,\ x).$$
- This NFT represents a single eventual payout determined by R, the settlement tick for that market, and the CLMSR engine.
- Position NFTs are freely transferable. A buyer of a position NFT steps into the claim on the underlying range; the vault's maker economics are unchanged.

A dedicated settlement entrypoint (e.g., `claim(positionId)`) checks that the associated market is finalized, computes the payout from R and settlementTick, pays it once, and marks the NFT as settled.

This separation keeps:

- **maker equity** composable via LP shares (structured products, vault-of-vaults, etc.), and
- **event outcomes** composable via NFTs (collateral, structured bets),

without entangling batch accounting with application-level use of those positions.

3.8 Rounding and Dust (Principles; Full Rules in Appendix C)

All vault accounting and fee splits are ultimately done in integer settlement-token units. Rounding follows three simple principles (Appendix C is the canonical reference):

1. **Trade-side dust**
   - Debits to users (costs, fees) round up.
   - Credits to users (proceeds) round down.
   - Per-trade dust (at most one base unit) stays with the maker and flows into L_t or F_t.
2. **Batch fairness**
   - For deposits, any residual amount that cannot be cleanly converted into shares at price P_t^e is refunded immediately; the vault does not keep deposit dust.
   - For withdrawals, any fractional remainder from converting shares to assets at P_t^e stays inside the vault and benefits remaining LPs via a slightly higher basis price.
   - When splitting residual fees between LP, Backstop, and Treasury, at most one unit of dust ($F_{\text{dust},t}$) appears, and it is always assigned to LPs.
3. **Backstop conservatism**
   - Backstop Grants (G_t) used to enforce the drawdown cap are rounded *up* so that Backstop over-covers by at most one unit.
   - Backstop never pays less than the WAD-level amount required by the Safety rule; it may pay one unit more.

These rules ensure that rounding never causes the protocol to pay out more than the underlying WAD-level economics, and that any bias from integer arithmetic is in favor of solvency and LPs, not against them.

3.9 Interface to the Safety Layer

For each daily market t, the interface between the LP Vault / Market-Cycle Batch and the Safety Layer is:

- **Inputs to Safety (from Sections 2–3):** CLMSR maker P&L L_t from $q_{\text{start},t} \to q_{\text{end},t}$ and depth α_t; gross fee amount $F_{\text{tot},t}$ from the active Fee policy; current capital-stack state $(N_{t-1}, B_{t-1}, T_{t-1})$ and configured Safety parameters $(\lambda, p_{dd}, \rho_{\text{BS}}, D_{\text{lag}}, k$, and related limits).
- **Outputs from Safety visible to the vault:** the net LP-vault credit Π_t applied before deposits and withdrawals, used to form $N_t^{\text{pre}} := N_{t-1} + \Pi_t$ and $P_t^e := N_t^{\text{pre}}/S_{t-1}$; updated Backstop and Treasury NAVs (B_t, T_t) reflecting fee allocations and any Backstop Grant G_t.

Section 4 formalizes how Safety computes Π_t and tranche-level flows from these inputs. The batch logic in Section 3 uses only Π_t and the updated capital-stack balances; it does not depend on Safety's internal decomposition.

4 Safety Layer: Solvency and Invariants

The Safety Layer is the part of the protocol whose only job is "the system does not die, no matter what path trading takes." It defines hard constraints on top of the CLMSR engine (Section 2) and the daily Market-Cycle Batch (Section 3). On any admissible path and for every daily market t, it guarantees:

- **Depth tied to LP capital.** The depth used satisfies $\alpha_t \le \alpha_{\text{limit},t} \le \alpha_{\text{base},t}$, where $\alpha_{\text{base},t}$ is derived from a single risk-budget parameter $\lambda \in (0,1)$ such that under a uniform prior the worst-case entropy loss consumes at most a fraction λ of LP capital.

- **Daily basis-price floor.** Let P_{t-1} be yesterday's LP basis price and P_t^e the pre-batch basis price after market t settles. Then
$$\frac{P_t^e}{P_{t-1}} - 1 \;\geq\; p_{dd}, \qquad p_{dd} := -\lambda,$$
enforced, if necessary, by a Backstop Grant $G_t \geq 0$.

- **Backstop never negative on admissible priors.** Opening prior $q_{0,t}$ defines an entropy budget $E_{\text{ent}}(q_{0,t})$ split into a core and a tail ΔE_t. Only priors with $\Delta E_t \leq B_{t-1}^{\text{eff}} \leq B_{t-1}$ are admissible, and the grant rule ensures $0 \leq G_t \leq \Delta E_t$, so
$$B_t = B_{t-1} + F_{\text{BS},t} - G_t \;\geq\; 0$$
on all admissible paths.

- **Same markets, same basis path.** Deposits and withdrawals are executed only once per day at the batch price P_t^e; withdrawal lag ensures any share present for market t participates fully in that market's P&L before it can exit.

- **Liquidity shocks cannot instantly collapse depth.** A fixed withdrawal lag D_{lag} slows outflows, and Treasury acts as a constrained Lender of Last Resort (LOLR) by buying LP shares in stress to restore depth capacity, never by paying Backstop Grants.

Sections 4.1–4.8 specify the state, parameters, and algorithms that implement these invariants.

4.1 Entropy Budget and the Single Risk Parameter λ

CLMSR uses
$$C(q) = \alpha \ln Z(q), \qquad Z(q) = \sum_b e^{q_b/\alpha}, \qquad p_b(q) = \frac{\partial C}{\partial q_b},$$
and maker P&L over a daily market is path-independent:
$$L_t = C(q_{\text{end},t}) - C(q_{\text{start},t}).$$

From the uniform state $q = 0$ on n ticks, $C(0) = \alpha \ln n$ and $p_b(0) = 1/n$. In the extreme where all mass collapses to one tick, $C(q) \to 0$, so a maker starting from $q = 0$ suffers at most
$$L_{\max}(0) \leq \alpha \ln n.$$
For a general opening state $q_{0,t}$,
$$L_{\max}(q_{0,t}) \leq C(q_{0,t}) - \min_j q_{0,t,j},$$
corresponding to "all mass concentrates on the tick with minimum initial log-weight."

Entropy budget and tail split. Define the conservative worst-case entropy-loss budget:
$$E_{\text{ent}}(q_{0,t}) := \begin{cases} \alpha_t \ln n, & q_{0,t} = 0 \text{ (uniform prior)}, \\ C(q_{0,t}) - \min_j q_{0,t,j}, & \text{general prior.} \end{cases}$$
For fixed α_t, separate core and tail:
$$\Delta E_t := E_{\text{ent}}(q_{0,t}) - \alpha_t \ln n \;\geq\; 0,$$
so $E_{\text{ent}}(q_{0,t}) = \alpha_t \ln n + \Delta E_t$.

Risk budget and static core. Let

$$E_t := N_{t-1}$$

be LP capital at the start of market t. Choose a single risk-budget parameter $\lambda \in (0,1)$ meaning "a worst-case uniform-prior day may burn at most λ of LP capital." Under a uniform prior enforce

$$\alpha_t \ln n = \lambda E_t,$$

which implies the static cap

$$\alpha_{\text{base},t} := \frac{\lambda E_t}{\ln n},$$

and set the drawdown floor

$$p_{dd} := -\lambda.$$

Even if Backstop were empty and only uniform priors were admissible, LPs would lose at most λE_t in a day and face the floor p_{dd}.

Prior admissibility. Let B_{t-1} be Backstop NAV at the end of batch $t-1$ and $B_{t-1}^{\text{eff}} \le B_{t-1}$ the portion reserved for tail coverage (v1 uses $B_{t-1}^{\text{eff}} = B_{t-1}$). A prior is admissible iff

$$\Delta E_t(q_{0,t}) \le B_{t-1}^{\text{eff}}.$$

If not, governance must flatten the prior, fall back to (near-)uniform, or recapitalize Backstop before opening market t. This condition, combined with the grant rule in Section 4.6, ensures Backstop is never asked to fund more tail risk than it has reserved.

4.2 Capital Stack and Tail Allocation

The entropy budget is split across a three-layer stack:

Tranche	Role	Loss segment	Source of return
LP Vault	Maker equity	Core entropy/volatility	LP fee residual F_t
Backstop Vault	Tail insurance	Tail ΔE_t beyond core	Backstop fee $F_{\text{BS},t}$
Treasury	LOLR & ops	Liquidity/ops shocks	Protocol revenue / policy

LP Vault (equity). NAV N_t aggregates CLMSR positions, fees, and Backstop Grants; share value $P_t := N_t/S_t$. LPs absorb the core segment within $\alpha_{\text{limit},t}$ and p_{dd} and receive $F_t := F_{\text{loss},t} + F_{\text{LP},t}$ (dust already absorbed) in the accounting identity $N_t^{\text{pre}} - N_{t-1} = L_t + F_t + G_t$. If LP-attributed fee inflow covers realized CLMSR losses over time (Section 5.1), basis price P_t tends to grow structurally.

Backstop Vault (mezzanine). Above LPs, it owns the incremental tail ΔE_t:

- Normal days: accumulates fee share $F_{\text{BS},t}$ (Section 4.3), carved out from gross fees.
- Tail days: pays Grant $G_t \ge 0$ when raw LP return would breach p_{dd}, recording $B_t - B_{t-1} = F_{\text{BS},t} - G_t$.
- Priors are admissible only if $\Delta E_t \le B_{t-1}^{\text{eff}}$; under this condition the grant rule never demands more than ΔE_t of Backstop capital.

Treasury (LOLR & ops). Treasury buys LP shares at batch price in stress to restore depth when withdrawals/losses would push α_t above $\alpha_{\text{limit},t}$ (Sections 3.6, 4.5). It funds audits/upgrades/incidents and may recapitalize Backstop under governance. Treasury never pays G_t directly; all grants are Backstop $\to$ LP.

4.3 Daily Fee Waterfall and Capital Flows

Gross fees $F_{\text{tot},t}$ and maker P&L L_t from market t map to daily fee shares, grants, and the net LP-vault credit Π_t via a fixed four-step priority that keeps the capital stack solvent. Starting balances are $(N_{t-1}, B_{t-1}, T_{t-1})$, and Safety parameters include $(p_{dd}, \rho_{\text{BS}}, \phi_{\text{LP}}, \phi_{\text{BS}}, \phi_{\text{TR}})$.

1. **Loss compensation.**
   Use fees to offset realized entropy loss $L_t^- := \max(-L_t, 0)$:
   $$F_{\text{loss},t} := \min(F_{\text{tot},t}, L_t^-),$$
   $$F_{\text{pool},t} := F_{\text{tot},t} - F_{\text{loss},t}.$$
   Define the raw post-loss NAV
   $$N_t^{\text{raw}} := N_{t-1} + L_t + F_{\text{loss},t}.$$
   This is the LP Vault NAV after CLMSR P&L and loss-offset fees but before any Backstop Grant.

2. **Drawdown floor and Backstop Grant.**
   Safety enforces the daily basis-price floor
   $$\frac{P_{e,t}}{P_{t-1}} - 1 \geq p_{dd}$$
   by sizing a Backstop Grant $G_t \geq 0$ according to the rule in Section 4.6. Let the floor NAV be
   $$N_t^{\text{floor}} := (1 + p_{dd}) \cdot N_{t-1}.$$
   Define the minimal integer grant subject to the admissible-tail constraint $0 \leq G_t \leq \Delta E_t$:
   $$G_t := \min\left\{\Delta E_t,\ \max\left\{0,\ \left\lceil N_t^{\text{floor}} - N_t^{\text{raw}} \right\rceil\right\}\right\}.$$
   If $G_t > 0$ then necessarily $F_{\text{tot},t} < L_t^-$ (hence $F_{\text{pool},t} = 0$), and no residual fee is distributed on that day.
   For bookkeeping it is convenient to write the intermediate balances after applying any grant as
   $$N_t^{\text{grant}} := N_t^{\text{raw}} + G_t, \qquad B_t^{\text{grant}} := B_{t-1} - G_t.$$

3. **Backstop coverage target.**
   When $F_{\text{pool},t} > 0$ (i.e., there is residual fee after loss compensation and no grant), remaining fees push Backstop toward its coverage target. Given a target ratio and shortfall
   $$B_t^{\text{target}} := \rho_{\text{BS}} \cdot N_t^{\text{grant}}, \qquad \Delta B_t^{\text{need}} := \max\{0,\ B_t^{\text{target}} - B_t^{\text{grant}}\},$$
   Safety dedicates part of $F_{\text{pool},t}$ to filling this gap:
   $$F_{\text{fill},t} := \min(F_{\text{pool},t},\ \Delta B_t^{\text{need}}), \qquad F_{\text{remain},t} := F_{\text{pool},t} - F_{\text{fill},t}.$$
   If Backstop is under-capitalized, most of $F_{\text{pool},t}$ flows into $F_{\text{fill},t}$; if coverage is adequate, $F_{\text{fill},t} = 0$ and $F_{\text{remain},t} = F_{\text{pool},t}$.

4. **Residual distribution and rounding.**
   The residual fee $F_{\text{remain},t}$ is split using weights $(\phi_{\text{LP}}, \phi_{\text{BS}}, \phi_{\text{TR}})$ (summing to 1). On-chain we choose non-negative integers
   $$F_{\text{LP},t}^{\text{core}},\ F_{\text{BS},t}^{\text{core}},\ F_{\text{TR},t}^{\text{core}}$$
   such that
   $$F_{\text{LP},t}^{\text{core}} + F_{\text{BS},t}^{\text{core}} + F_{\text{TR},t}^{\text{core}} \leq F_{\text{remain},t},$$
   and define the dust
   $$F_{\text{dust},t} := F_{\text{remain},t} - (F_{\text{LP},t}^{\text{core}} + F_{\text{BS},t}^{\text{core}} + F_{\text{TR},t}^{\text{core}}) \in \{0, 1\},$$
   which is always credited to LPs. The final fee shares are
   $$F_{\text{LP},t} := F_{\text{LP},t}^{\text{core}} + F_{\text{dust},t},$$
   $$F_{\text{BS},t} := F_{\text{fill},t} + F_{\text{BS},t}^{\text{core}},$$
   $$F_{\text{TR},t} := F_{\text{TR},t}^{\text{core}}.$$

Outputs and identities. Define total LP-attributed fees $F_t := F_{\text{loss},t} + F_{\text{LP},t}$. The net LP-vault credit applied at batch t is

$$\Pi_t := L_t + F_t + G_t.$$

Backstop and Treasury update by

$$B_t := B_{t-1} + F_{\text{BS},t} - G_t, \qquad T_t := T_{t-1} + F_{\text{TR},t} + (\text{other protocol flows}).$$

By construction,

$$F_{\text{tot},t} = F_{\text{loss},t} + F_{\text{LP},t} + F_{\text{BS},t} + F_{\text{TR},t},$$

and the tranche-level identities become

$$N_t^{\text{pre}} - N_{t-1} = \Pi_t = L_t + F_t + G_t,\ B_t - B_{t-1} = F_{\text{BS},t} - G_t,\ T_t - T_{t-1} = F_{\text{TR},t} + (\text{other protocol flows}).$$

Economically, loss is offset first; Backstop injects G_t only if needed to meet the drawdown floor. Remaining fees build Backstop toward coverage and then split residually across LPs/Backstop/Treasury.

4.4 Static α Cap from Entropy Budget

From $\alpha_t \ln n = \lambda E_t$ under a uniform prior, the static cap is

$$\alpha_{\text{base},t} := \frac{\lambda E_t}{\ln n}, \qquad E_t := N_{t-1}.$$

This does not depend on the choice of prior $q_{0,t}$; any incremental budget from a concentrated prior is treated as tail ΔE_t allocated to Backstop, not as a reason to shrink $\alpha_{\text{base},t}$. The Performance Layer must satisfy $\alpha_t \le \alpha_{\text{base},t}$ in every market.

4.5 Drawdown-Damped α Limit

Static caps cannot react to cumulative losses. Let $P_{\text{peak},t}$ be the peak basis price up to batch t and P_t the current price; define drawdown

$$\text{DD}_t := 1 - \frac{P_t}{P_{\text{peak},t}} \in [0,1].$$

Given damping coefficient $k > 0$, define the next-market cap

$$\alpha_{\text{limit},t+1} := \max\{0,\ \alpha_{\text{base},t+1}(1 - k\,\text{DD}_t)\}.$$

At the end of batch t, Safety computes $E_{t+1} := N_t$, $\alpha_{\text{base},t+1}$, DD_t, and hence $\alpha_{\text{limit},t+1}$. During market $t+1$, Performance must choose depth satisfying

$$\alpha_{t+1} \le \alpha_{\text{limit},t+1}.$$

By construction $0 \le \alpha_{\text{limit},t+1} \le \alpha_{\text{base},t+1}$.

4.6 Drawdown Floor and Backstop Grants

Depth limits bound loss size but not the path of LP returns. Safety enforces a daily basis-price floor

$$\frac{P_t^e}{P_{t-1}} - 1 \ \ge\ p_{dd} = -\lambda,$$

using a Backstop Grant G_t when needed. This subsection is the canonical definition of the grant rule referenced in Section 4.3; all occurrences of G_t in capital flows use this rule.

After applying loss-offset fees, define raw NAV and price

$$N_t^{\text{raw}} := N_{t-1} + L_t + F_{\text{loss},t}, \qquad P_t^{\text{raw}} := \frac{N_t^{\text{raw}}}{S_{t-1}},$$

with raw return $r_t^{\text{raw}} := P_t^{\text{raw}}/P_{t-1} - 1$. Safety requires the final pre-batch price to satisfy

$$\frac{P_t^e}{P_{t-1}} - 1 \;\geq\; p_{dd}.$$

Since $F_{\text{LP},t} \geq 0$ (dust already in $F_{\text{LP},t}$) can only increase NAV, a sufficient condition is

$$N_t^{\text{raw}} + G_t \;\geq\; (1 + p_{dd})\, P_{t-1} S_{t-1}.$$

The minimal continuous grant satisfying this is

$$G_t^{\min} := (1 + p_{dd})\, P_{t-1} S_{t-1} - N_t^{\text{raw}},$$

and on-chain

$$G_t := \max\{0, \lceil G_t^{\min} \rceil\},$$

rounded up and never negative. If $G_t = 0$, the floor is satisfied by P&L and fees; if $G_t > 0$, Backstop injects capital to trim the tail.

Under the λ parametrization, $E_{\text{ent}}(q_{0,t}) = \lambda E_t + \Delta E_t$, so worst case

$$G_t \leq E_{\text{ent}}(q_{0,t}) - \lambda E_t = \Delta E_t.$$

Thus at most the tail ΔE_t is ever needed from Backstop to keep LP return above p_{dd}, and the admissibility rule $\Delta E_t \leq B_{t-1}^{\text{eff}}$ implies Backstop is never asked to pay more than its reserved tail capacity.

Interaction with fees:

- If $F_{\text{tot},t} \geq L_t^-$, then $F_{\text{loss},t} = L_t^-$, $N_t^{\text{raw}} \geq N_{t-1}$, $r_t^{\text{raw}} \geq 0 > p_{dd}$, and $G_t = 0$.
- If $G_t > 0$, necessarily $F_{\text{tot},t} < L_t^-$, so $F_{\text{loss},t} = F_{\text{tot},t}$ and $F_{\text{pool},t} = 0$: no residual fee is distributed that day; Backstop participation is via G_t only.

With the vault identity $N_t^{\text{pre}} = N_{t-1} + L_t + F_t + G_t$ and $P_t^e = N_t^{\text{pre}}/S_{t-1}$, G_t is uniquely determined by (p_{dd}, α limits) and realized ($L_t, F_{\text{tot},t}$). It is strictly one-way (Backstop $\to$ LP) and keeps LP daily return above p_{dd} while B_t stays non-negative on admissible paths.

4.7 Liquidity Guards: Withdrawal Lag & LOLR

Entropy budgets and drawdown caps control solvency on the capital side; liquidity guards control solvency on the cash-flow side.

- **Fixed withdrawal lag.** As in Section 3.6, a withdrawal requested at T_{req} executes only in the first batch with $T_{\text{cycle}} \geq T_{\text{req}} + D_{\text{lag}}$, always at that batch's single price P_t^e (no price lock at request time). Every share present for market t absorbs that market's (L_t, F_t, G_t) before it can exit; "same market, same basis path" is preserved.
- **Treasury LOLR.** When withdrawals and losses would otherwise push depth above $\alpha_{\text{limit},t}$, Treasury may buy LP shares at batch price to raise N_t and hence E_{t+1}, restoring future depth capacity. Triggers, price rules, and limits are policy parameters (Appendix B); Treasury never pays G_t directly.

4.8 Safety Summary & Interface to Performance Layer

Summarizing:

- **Entropy budget.** $E_{\text{ent}}(q_{0,t}) = \lambda E_t + \Delta E_t$, $E_t := N_{t-1}$; core λE_t (uniform prior), tail ΔE_t (prior concentration).
- **Tranching.** LPs hold the core segment and residual fees; Backstop holds tail ΔE_t with fees $F_{\text{BS},t}$ and grants G_t; Treasury handles liquidity/ops and may buy LP shares to preserve depth, never paying G_t.

- **Depth constraint.** $\alpha_t \le \alpha_{\text{limit},t} \le \alpha_{\text{base},t} = \lambda E_t / \ln n$, with $\alpha_{\text{limit},t}$ derived from DD_{t-1} and acting as a hard cap during market t.
- **Drawdown floor.** $\dfrac{P_t^e}{P_{t-1}} - 1 \ge p_{dd} = -\lambda$; grants satisfy $0 \le G_t \le \Delta E_t$ and $B_t = B_{t-1} + F_{\text{BS},t} - G_t \ge 0$ on admissible priors.
- **Backstop-thin regime.** If B_{t-1} is small, admissibility forces $\Delta E_t \approx 0$: only (near-)uniform priors are allowed; LPs still face at most λE_t and the drawdown floor p_{dd} holds.
- **Liquidity guards.** Fixed D_{lag} and constrained LOLR actions prevent instantaneous depth collapse and keep one basis-price system per batch.

The Performance Layer (Section 5) treats (L_t, F_t, G_t) as observed time series and chooses depth, priors, and Fee policies *inside this safety box*: $\alpha_t \le \alpha_{\text{limit},t}$, admissible priors, fixed p_{dd} and D_{lag}, and grants as defined above. Its goals are to satisfy the long-run fee–cost condition (Section 5.4), grow LP basis price, and keep trader slippage within UX targets without weakening Safety guarantees.

5 Performance Layer – Fees, Depth, and Prior

The Safety Layer fixes a narrow region in which the system must live:

- CLMSR depth is bounded by a drawdown-damped cap
$$\alpha_t \le \alpha_{\text{limit},t} \le \alpha_{\text{base},t} = \frac{\lambda E_t}{\ln n},$$
with $E_t := N_{t-1}$ and a single risk-budget parameter $\lambda \in (0,1)$.
- The LP basis price has a hard daily drawdown floor
$$\frac{P_{e,t}}{P_{t-1}} - 1 \ge p_{dd} = -\lambda,$$
implemented via Backstop Grants $G_t \ge 0$.
- Opening priors $q_{0,t}$ are admissible only if their entropy tail ΔE_t fits inside Backstop's reserved capacity.
- Withdrawal delay D_{lag} and Treasury as LOLR constrain how fast LP capital can leave and how quickly depth can collapse, even in a run.

On top of these invariants, the Performance Layer chooses how aggressively to use depth and where to place it, and how much fee to charge. Concretely, on each market t: Throughout this section, no new Safety semantics are introduced; the Performance Layer only chooses controller outputs inside the constraints defined in Section 4.

- **Inputs from Safety and accounting**
  - Previous LP state $(N_{t-1}, S_{t-1}, P_{t-1})$ with $P_{t-1} = N_{t-1}/S_{t-1}$.
  - Backstop balance B_{t-1} and coverage target ρ_{BS}.
  - Safety bounds $(\alpha_{\text{limit},t}, p_{dd}, D_{\text{lag}})$.
  - End-of-day realizations: CLMSR maker P&L L_t, gross fee $F_{\text{tot},t}$, LP-attributed fee F_t, grant G_t, and derived metrics such as Fee/Loss ratio $\psi_{\text{fee},t}$ and average slippage Δp_t.
- **Outputs chosen by Performance**
  - Depth $\alpha_t \le \alpha_{\text{limit},t}$ for market t (and α_{t+1} for the next day).
  - Opening prior $q_{0,t}$ (and $q_{0,t+1}$) subject to prior-admissibility.

– Fee policy choice for market t (which whitelisted FeePolicy implementation is active, and any operational configuration within its published rules) that determines gross fee $F_{\text{tot},t}$.

In terms of daily flow:

1. **Zero-Hour (before trading)**: Safety computes $(\alpha_{\text{base},t}, \alpha_{\text{limit},t})$ from $(N_{t-1}, B_{t-1}, \lambda)$. Performance chooses a prior $q_{0,t}$, a depth $\alpha_t \le \alpha_{\text{limit},t}$, and a Fee policy for market t.
2. **Trading**: traders interact with the CLMSR engine at fixed $(\alpha_t, q_{0,t})$; the CLMSR state moves $q_{\text{start},t} \to q_{\text{end},t}$ and fees are accrued according to the schedule.
3. **Settlement and Safety updates**: CLMSR P&L $L_t = C(q_{\text{end},t}) - C(q_{\text{start},t})$ and gross fees $F_{\text{tot},t}$ are known. The Fee Waterfall and Backstop rules allocate $F_{\text{tot},t}$ and compute a grant G_t to enforce the drawdown floor. LP pre-batch NAV and basis price $(N_t^{\text{pre}}, P_{e,t})$ follow from these flows (Section 3).
4. **Batch execution**: execute queued withdrawals/deposits at $P_{e,t}$, yielding (N_t, S_t, P_t) for day $t{+}1$.
5. **Feedback update**: observe $(L_t, F_t, G_t, \psi_{\text{fee},t}, \Delta p_t)$ and, given $(\alpha_{\text{limit},t+1}, p_{dd}, D_{\text{lag}})$, choose α_{t+1}, $q_{0,t+1}$, and the next Fee policy.

Sections 5.1–5.5 specify the Performance data view, Zero-Hour configuration, and daily control rules.

5.1 Daily P&L Decomposition and Monitoring Metrics

We first collect the objects that the Performance Layer treats as its "daily data view."

Daily maker P&L and pre-batch NAV. Over market t:

- L_t is the CLMSR maker P&L from Section 2.
- $F_{\text{tot},t} \ge 0$ is the gross fee collected on trades and settlements under the Fee policy.

The Safety Layer's Fee Waterfall (Section 4.3) splits $F_{\text{tot},t}$ across loss compensation, Backstop accumulation, LP residual, Treasury, and rounding dust. From the Performance Layer's perspective, we only need the aggregated LP fee term

$$F_t := F_{\text{loss},t} + F_{\text{LP},t},$$

which is the portion of the day's fee credited to the LP Vault, and the Backstop Grant $G_t \ge 0$, which is sized by the grant rule in Section 4.6 to enforce the daily drawdown floor.

We also define the realized entropy loss

$$L_t^- := \max(-L_t, 0),$$

used below in the Fee/Loss ratio. With these definitions, LP pre-batch NAV evolves as

$$N_t^{\text{pre}} := N_{t-1} + L_t + F_t + G_t,$$

which is exactly

$$N_t^{\text{pre}} - N_{t-1} = L_t + F_t + G_t.$$

The corresponding pre-batch basis price is

$$P_{e,t} := \frac{N_t^{\text{pre}}}{S_{t-1}}.$$

Basis return. The LP's daily return is

$$R_t := \frac{P_{e,t}}{P_{t-1}} - 1.$$

Safety guarantees $R_t \ge p_{dd} = -\lambda$ for all t, so the left tail of daily LP returns is truncated at a known floor by Backstop Grants.

Realized Fee/Loss ratio. To compare realized entropy cost and fee income, the Performance Layer tracks

$$\psi_{\text{fee},t} := \begin{cases} \dfrac{F_t}{L_t^-}, & L_t^- > 0, \\ +\infty, & L_t^- = 0. \end{cases}$$

On loss days this is fee per unit of realized CLMSR loss; on gain days it is set to $+\infty$. Governance sets a target $\psi_{\text{target}} > 1$. A moving average $\overline{\psi_{\text{fee}}}$ staying above ψ_{target} is a practical translation of the long-run Fee–Cost condition $\mathbb{E}[F_t] \gtrsim \mathbb{E}[L_t^-]$.

Two-sided average slippage. For each executed trade i in market t that respects the configured position-size limits (per-ticket and per-account caps; see Appendix B), let v_i be the execution price and m_i the contemporaneous CLMSR midprice; define relative slippage

$$\delta_i := \frac{|v_i - m_i|}{m_i}.$$

Let Δp_t be a volume-weighted average or high percentile (e.g., 90th) of $\{\delta_i\}$ over the day. Governance specifies a UX target $\Delta p_{\text{target}} > 0$. If $\Delta p_t \gg \Delta p_{\text{target}}$, typical allowed trades see excessive slippage; if $\Delta p_t \ll \Delta p_{\text{target}}$, depth may be unnecessarily high for the actually traded sizes. The Performance Layer uses $(R_t, \psi_{\text{fee},t}, \Delta p_t)$ as its primary signals for depth and fee adjustments.

5.2 Zero-Hour Configuration: Prior & Initial Depth

At the start of each daily market t, the CLMSR engine must be initialized at some state $q_{0,t}$ with depth α_t. Choosing both correctly reduces wasteful early-day P&L swings and makes efficient use of the entropy budget.

Exogenous prior embedding. Let p_t^0 be an exogenous estimate over ticks $b \in \mathcal{B}$. Given α_t, embed it via

$$q_{0,t,b} := \alpha_t \ln p_{t,b}^0 + \kappa_t, \quad b \in \mathcal{B},$$

where κ_t is an additive constant that leaves marginal prices unchanged. This choice affects

$$E_{\text{ent}}(q_{0,t}) = \alpha_t \ln n + \Delta E_t, \quad \Delta E_t \geq 0,$$

where $\alpha_t \ln n$ is the uniform-prior baseline and ΔE_t is the tail contribution Safety assigns to the Backstop segment. Prior admissibility requires

$$\Delta E_t(q_{0,t}) \leq B_{t-1}^{\text{eff}},$$

with B_{t-1}^{eff} the Backstop's effective tail capacity (Section 4). Priors that violate this bound must be flattened, combined with additional Backstop capital, or rejected.

Zero-Hour prior selection is therefore an optimization problem: approximate the settlement distribution as closely as possible while keeping ΔE_t within Backstop capacity. Estimation methods and blending rules are policy-level choices; the mechanism only requires prior admissibility.

Initial depth. Depth α_t at market open must respect Safety and UX:

- Safety gives a hard upper bound $\alpha_{\text{limit},t}$ from entropy budget and drawdown (Sections 4.4–4.6).
- UX is expressed as a bound on the slippage experienced by trades that respect the configured position-size limits. For any candidate depth α, the CLMSR curve together with those limits implies a predicted distribution of trade slippages. Let $\alpha_{\text{UX},t}$ denote the depth recommended by the off-chain UX policy for market t so that typical allowed trades are expected to see slippage near Δp_{target}.

Signals v1 uses

$$\alpha_t^{\text{init}} := \min\{\alpha_{\text{limit},t},\, \alpha_{\text{UX},t}\}.$$

If $\alpha_{\text{limit},t} \geq \alpha_{\text{UX},t}$, initial depth is UX-driven with Safety slack; if $\alpha_{\text{limit},t} < \alpha_{\text{UX},t}$, Safety dominates and slippage is temporarily higher until capital/drawdown improve. $\alpha_{\text{UX},t}$ itself is not an independent governance parameter; it is derived from UX policy, current regime, and the configured position-size limits. After Zero-Hour, α_t is updated daily by the feedback policy in Section 5.3.

5.3 Depth Control: Squeeze / Expand Policy

Depth determines both slippage and worst-case entropy cost. Holding everything else fixed, larger α_t improves UX but amplifies potential loss; smaller α_t conserves budget but increases slippage. The Performance Layer uses a simple daily feedback controller to move α_t inside the envelope set by Safety.

Principles:

1. Depth never exceeds the Safety cap:

$$\alpha_{t+1} \leq \alpha_{\text{limit},t+1}.$$

2. By default, depth slowly decreases when conditions do not justify expansion, reducing structural entropy cost over time.
3. Depth increases only when both Fee–Cost and UX indicators are favorable.

Let $\gamma \in (0,1)$ be the squeeze rate, $\eta > 0$ the expand rate, and $(\psi_{\text{target}}, \Delta p_{\text{target}})$ the targets.

Squeeze: Default Cost Reduction.

$$\alpha_{t+1}^{\text{squeeze}} := (1-\gamma)\alpha_t.$$

This nudges depth down slowly when neither Fee–Cost nor UX signals justify expansion, conserving entropy budget. In practice, UX lower bounds are enforced by the combination of this policy and the per-market position-size limits; there is no separate on-chain $\alpha_{\min}$ parameter.

Expand: Fee–Cost & UX Conditions. Expansion is considered only when

$$\psi_{\text{fee},t} > \psi_{\text{target}} \quad \text{and} \quad \Delta p_t > \Delta p_{\text{target}}.$$

Given $\eta > 0$,

$$\alpha_{t+1}^{\text{expand}} := \min\{\alpha_{\text{limit},t+1},\ (1+\eta)\alpha_t\},$$

so expansion stays gradual and inside the Safety cap.

Update Rule.

$$\alpha_{t+1} = \begin{cases} \alpha_{t+1}^{\text{expand}}, & \psi_{\text{fee},t} > \psi_{\text{target}} \text{ and } \Delta p_t > \Delta p_{\text{target}}, \\ \alpha_{t+1}^{\text{squeeze}}, & \text{otherwise.} \end{cases}$$

Parameters $(\gamma, \eta, \psi_{\text{target}}, \Delta p_{\text{target}})$ are calibrated via simulation and live data and can be tuned by governance without changing CLMSR or Safety semantics.

5.4 Fee Policy and Long-run LP Economics

Fees close the loop between CLMSR entropy cost and LP returns. From the Performance Layer's point of view there are only two moving pieces:

1. the **Fee policy**, which charges individual trades and settlements and aggregates them into a single daily gross fee $F_{\text{tot},t}$; and

2. the **Safety-fixed Fee Waterfall**, which takes $(L_t, F_{\text{tot},t}, N_{t-1}, B_{t-1})$ as input and produces the LP-facing quantities (F_t, G_t) plus Backstop/Treasury flows, according to the rules in Section 4.3.

The Performance Layer controls only the first; the second is treated as an immutable Safety mechanism.

Fee policy (how fees are collected). The Fee policy is any rule that, given the day's trading and settlement activity, produces a non-negative gross fee amount $F_{\text{tot},t}$. In practice this includes:

- maker/taker rates on range trades,
- settlement and claim fees, and
- any volume-, size-, or time-dependent adjustments (e.g., higher fees during stressed periods),

aggregated over the day.

All such choices must satisfy three constraints:

1. **Outside the cost function.** Fees are imposed *outside* $C(q)$ so that $p(q)$ keeps its probability semantics and CLMSR path-independence is preserved.
2. **Well-defined daily aggregate.** For every market t, the policy yields a unique $F_{\text{tot},t}$ that can be consumed by the Fee Waterfall in Section 4.3.
3. **Compatibility with long-run Fee–Cost inequality.** Together with depth and priors, the policy should be chosen so that the realized Fee/Loss ratio $\psi_{\text{fee},t}$ stays above a governance target $\psi_{\text{target}} > 1$ on average, making the structural condition

$$\mathbb{E}[F_t] \gtrsim \mathbb{E}[L_t^-]$$

attainable in practice.

Within these constraints the policy space is deliberately wide: Signals v1 does not fix a single fee curve. Governance instead curates a whitelist of FeePolicy implementations (contract addresses) whose published code encodes maker/taker tiers, minimum/maximum rates, and adjustment rules; the mechanism only requires that any such implementation compress its per-trade and per-settlement charges to a unique daily $F_{\text{tot},t}$.

Interaction with Safety, Fee Waterfall, and Grants. Once $F_{\text{tot},t}$ is known, the Safety Layer takes over. The Fee Waterfall and grant rules in Sections 4.3–4.6 map

$$(L_t,\, F_{\text{tot},t},\, N_{t-1},\, B_{t-1}) \longmapsto (F_t,\, G_t,\, F_{\text{BS},t},\, F_{\text{TR},t}),$$

with a fixed priority "loss compensation → Backstop accumulation → residual distribution."

From the Performance Layer's perspective:

- F_t is the LP-attributed fee term produced by the Fee Waterfall; it enters the daily P&L identity

$$N_t^{\text{pre}} - N_{t-1} = L_t + F_t + G_t,$$

and drives basis-price growth together with L_t and G_t.

- $G_t \geq 0$ is a one-way Backstop Grant fixed by Safety parameters (λ, p_{dd}, entropy budgets) and realized $(L_t, F_{\text{tot},t})$; it is not Performance-controlled.
- Backstop and Treasury flows $(F_{\text{BS},t}, F_{\text{TR},t})$ never feed back into the controller directly; they are tracked for capital-stack accounting and governance, not for daily tuning.

In other words, Performance chooses which Fee policy implementation to apply (and any operational calibration it allows) together with depth/prior policy so as to shape the joint process $(L_t, F_{\text{tot},t})$ and the derived metrics $(\psi_{\text{fee},t}, \Delta p_t)$, but it does *not* change:

- the Waterfall priority or rounding rules (Section 4.3, Appendix C),
- the grant sizing and drawdown-floor semantics (Section 4.6), or
- the Backstop coverage targets (Section 4.3) and α-limit logic (Section 4.5).

All of those belong to the Safety Layer.

Long-run LP economics under a given Fee policy. Given a fixed Safety configuration, a family of Fee policies defines a family of possible LP basis-price paths. For any such policy, the LP pre-batch NAV evolves as

$$N_t^{\text{pre}} = N_{t-1} + L_t + F_t + G_t, \qquad P_{e,t} = \frac{N_t^{\text{pre}}}{S_{t-1}},$$

and the LP's daily return is

$$R_t := \frac{P_{e,t}}{P_{t-1}} - 1 \;\geq\; p_{dd} = -\lambda.$$

The Performance Layer's objective is to choose:

- depth α_t,
- priors $q_{0,t}$, and
- the Fee policy applied to each market t (choice of whitelisted FeePolicy implementation and any operational calibration within its published rules),

so that, under real trading behavior:

1. the long-run Fee–Cost inequality $\mathbb{E}[F_t] \gtrsim \mathbb{E}[L_t^-]$ holds,
2. empirical $\psi_{\text{fee},t}$ and slippage Δp_t stay near their governance targets, and
3. Safety invariants (entropy budgets, α-limits, drawdown floor, liquidity guards) remain strictly respected.

Signals v1 intentionally leaves the exact Fee policy functional form and calibration to governance and operations; Section 8 describes how these parameters can be adjusted over time as more data on $(L_t, F_{\text{tot},t}, \psi_{\text{fee},t}, \Delta p_t)$ is observed.

5.5 Summary and Interface

The Performance Layer sits strictly inside the Safety envelope and operates as a daily feedback controller:

- **Contract with Safety:** depth bounded by $\alpha_t \leq \alpha_{\text{limit},t}$; daily return floor $p_{dd} = -\lambda$ via grants; priors obey $\Delta E_t \leq B_{t-1}^{\text{eff}}$; withdrawal lag and LOLR rules are fixed.
- **Control levers:** Zero-Hour picks $(q_{0,t}, \alpha_t)$ and Fee policy; end-of-day $(L_t, F_t, G_t, \psi_{\text{fee},t}, \Delta p_t)$ drives Squeeze/Expand and Fee policy tuning inside Safety bounds.
- **Long-run LP economics:** $N_t^{\text{pre}} - N_{t-1} = L_t + F_t + G_t$ with batch mint/burn rules; Safety trims tails and caps depth; Performance targets $\overline{\psi_{\text{fee}}} \gtrsim \psi_{\text{target}} > 1$ and $\Delta p_t \approx \Delta p_{\text{target}}$.

Future versions can introduce richer controllers—for example, multi-vault depth allocation and LP-selectable risk tranches that more finely slice which capital bears which parts of the CLMSR risk/return surface, together with LP-weighted preference aggregation and on-chain learning for priors, depth, and fee curves—so long as they respect the same Safety contract and Market-Cycle Batch accounting layer.

6 Oracle & Settlement

Signals' daily BTC range markets each have a fixed **settlement reference time** T_{set}. The BTC/USD price at this time is mapped to OutcomeSpec ticks and confirmed as the **settlement tick**, and position payouts are calculated based on this tick. CLMSR maker P&L L_t is determined separately by the state transition $q_{\text{start},t} \to q_{\text{end},t}$ via $C(q_{\text{end},t}) - C(q_{\text{start},t})$ (Section 2).

In v1, settlement is divided into two layers.

- **Primary Rule**: Select one sample closest to T_{set} from the signed pull oracle feed to determine the settlement price.
- **Secondary Rule**: For markets where the primary rule fails, calculate a separate settlement price according to *pre-announced manual settlement rules* and reflect it on-chain within that day.

On-chain contracts only fix `settlementValue` and `settlementTick` after primary/secondary rules resolve. Reference CEX prices, indices, etc. for secondary settlement stay off-chain.

6.1 Signed Pull Oracle: On-chain Consumer

Signals uses a **signed pull oracle feed**. Oracle providers continuously sign and provide BTC/USD prices, and users attach this signed data to their settlement transactions and deliver it to contracts.

Oracle Packet. Settlement transaction calldata includes an oracle data packet containing the following fields.

- `feedId`: ID identifying the BTC/USD feed.
- `price`: Normalized BTC/USD price with decimals d_{feed}.
- `priceTimestamp`: UNIX timestamp when that price was observed.
- Signature and public key ID (for verifying oracle provider's signature).

Contracts parse the packet via the oracle adapter (e.g., `getOracleNumericValueFromTx(feedId)`) and verify the signature against the registered public key.

Format Constraints. After passing signature verification, contracts check the following.

- Whether `feedId` matches that market's BTC/USD feed.
- Whether `price`>0 and can be converted to OutcomeUnit scale without overflow/underflow.
- Whether $|\texttt{priceTimestamp} - T_{\text{set}}| \le \Delta_{\max}$ (reject samples that are too old or early).

`price` satisfying these conditions is normalized to OutcomeUnit integer $x \in \mathbb{Z}$ according to OutcomeSpec's decimals d. This x becomes a `settlementValue` candidate.

6.2 Outcome Domain and Tick Mapping

Outcome domain and tick mapping use the definitions from Section 2 as-is. Each market has OutcomeSpec (L, U, s, d), where L, U are OutcomeUnit scale lower/upper bounds, s is tick interval, and d is raw oracle decimal places. $U - L$ is always divisible by s, giving tick count $n = (U - L)/s$.

For settlement price x, tick mapping is defined as

$$\text{toTick}(x) = \min\left\{n - 1,\ \max\left\{0,\ \left\lfloor \tfrac{x-L}{s} \right\rfloor\right\}\right\}.$$

If $x < L$, clamp to tick 0; if $x \ge U$, clamp to tick $n - 1$, so edge ticks absorb prices outside the range.

On-chain, two values are ultimately stored.

- `settlementValue`: OutcomeUnit scale settlement price (for audit/logging).
- `settlementTick`: settlement tick index obtained from the above `toTick`. Used for position payout. Maker P&L L_t is determined solely by the CLMSR state transition via $C(q_{\mathrm{end},t}) - C(q_{\mathrm{start},t})$ (Section 2).

6.3 Settlement Timeline and State Machine

Each market m has the following parameters at creation.

$$T_{\mathrm{set}}(m), \quad \Delta_{\mathrm{settle}}, \quad \Delta_{\mathrm{ops}}, \quad \Delta_{\mathrm{claim}}.$$

Signals v1 uses default values

$$\Delta_{\mathrm{settle}} = 10\mathrm{min}, \quad \Delta_{\mathrm{ops}} = 5\mathrm{min}, \quad \Delta_{\mathrm{claim}} = 15\mathrm{min}.$$

Here we define

$$T_{\mathrm{settle,end}} := T_{\mathrm{set}} + \Delta_{\mathrm{settle}}, \qquad T_{\mathrm{ops,end}} := T_{\mathrm{settle,end}} + \Delta_{\mathrm{ops}}.$$

The flow on the time axis is as follows.

- $t < T_{\mathrm{set}}$: Only position trading is possible (`Trading` state).
- $T_{\mathrm{set}} \le t < T_{\mathrm{settle,end}}$: Anyone can call `settleMarket` to submit oracle samples (`SettlementOpen` state).
- $T_{\mathrm{settle,end}} \le t < T_{\mathrm{ops,end}}$: No longer accept new settlements, only maintain stored candidate settlementPrice (`PendingOps` state). During these 5 minutes, operations can determine fail status with sanity check.
- $t \ge T_{\mathrm{ops,end}}$: If no fail call, primary rule settlement is confirmed (`FinalizedPrimary`); if fail is called, enter secondary rule manual settlement waiting state (`FailedPendingManual`).

Market states are summarized as follows.

- `Trading`
- `SettlementOpen`
- `PendingOps`
- `FinalizedPrimary` (market confirmed by primary rule)
- `FinalizedSecondary` (market confirmed by secondary rule manual settlement)

Position claims are possible in both Finalized states, the only difference being where the settlement price came from (primary vs secondary).

State transitions are transaction-driven. A state-advancer moves `SettlementOpen` → `PendingOps` after $T_{\mathrm{settle,end}}$, and `PendingOps` → `FinalizedPrimary` after $T_{\mathrm{ops,end}}$ if not failed. Time-gated moves stay explicit even without on-chain timers.

6.4 `settleMarket`: Permissionless Primary Settlement

Settlement transaction `settleMarket(marketId, oraclePayload)` can be called by *anyone*. It is not limited to Keepers; any address can submit a packet containing a valid oracle signature. On-chain authority control is done only through signature verification.

Time and State Constraints. `settleMarket` calls must satisfy all of the following conditions.

- `block.timestamp` $\geq T_{\text{set}}$,
- `block.timestamp` $< T_{\text{settle,end}}$,
- Market state is `Trading` or `SettlementOpen`.
- Oracle payload timestamp also satisfies `priceTimestamp` $\leq$ `block.timestamp` $+ \delta_{\text{future}}$ (default $\delta_{\text{future}} = 0$) so future-dated samples are rejected.

Calls outside this range revert with `SettlementWindowClosed` type errors.

Closest Sample Selection. With a valid oracle packet, contracts convert it to (x, t_{px}) and compare with stored $(x^{(0)}, t^{(0)})$.

- If candidate is empty, store $(x^{(1)}, t^{(1)})$ as-is.
- If candidate already exists, replace with new candidate only when $|t^{(1)} - T_{\text{set}}| < |t^{(0)} - T_{\text{set}}|$.
- In case of tie, keep the more past sample (smaller t).

This way, even if update intervals are uneven around the settlement reference time, one sample closest to T_{set} is always selected.

Promotion at Window End. When `block.timestamp` $\geq T_{\text{settle,end}}$ and state transitions from `SettlementOpen` to `PendingOps`, stored candidate $(x^\star, t^\star)$ is promoted as follows.

$$\begin{aligned} \texttt{settlementValuePrimary} &:= x^\star, \\ \texttt{settlementTickPrimary} &:= \text{toTick}(x^\star), \\ \texttt{settlementPriceTimestamp} &:= t^\star. \end{aligned}$$

After this, `settleMarket` is no longer accepted and the primary settlement price does not change. Operations can determine off-chain whether this price passes sanity during the `PendingOps` period (≈ 5 minutes).

6.5 Claim Gating and Position Settlement

Whether primary or secondary rule, once the final settlement price is set and state is `FinalizedPrimary` or `FinalizedSecondary`, position holders claim via `claim(positionId)`. Claims must satisfy the following conditions.

- Market state is `FinalizedPrimary` or `FinalizedSecondary`.
- `settlementTick` is set.
- `block.timestamp` $\geq$ `settlementFinalizedAt` $+ \Delta_{\text{claim}}$.

`settlementFinalizedAt` is the transaction time that sets `FinalizedPrimary` or `FinalizedSecondary`. Claim gating is always relative to this time. With $\Delta_{\text{claim}} = 15$ minutes, claims open 15 minutes later.

Position payouts are determined by comparing Outcome range R with final `settlementTick`, and each position ID is marked in internal state so it can only be settled once. `PositionSettled` events record position ID, owner, payout, and win/loss together so subgraph/indexer can reconstruct state.

6.6 Failure Modes and Secondary Rule

Since the primary settlement rule is not always reliable, Signals' principle is to complete manual settlement **within the same day** using *pre-announced secondary settlement rules* when it fails. Here "secondary rule" means decision rules specified in off-chain documents (governance parameters, operations manual, etc.) for BTC/USD data sources, sampling window, and aggregation method (VWAP/median, etc.).

Major failure modes and processing flows are as follows.

(i) Oracle Sample Absence. This is when no valid oracle samples are submitted during the settlement window, so no `settlementValuePrimary` candidate is created.

- In this case, the market enters `PendingOps`, but since the primary price is empty, primary settlement cannot be confirmed.
- The operations account calls `markFailed(marketId)` during or right after the `PendingOps` period to mark that market as Failed and transition state to `FailedPendingManual`.
- Then calculate manual settlement price $x^{(2)}$ and timestamp $t^{(2)}$ off-chain according to the secondary rule. Within the same settlement day, call `manualSettleFailedMarket(marketId, x(2), t(2))` to reflect it on-chain.

(ii) Serious Data Divergence. This is when a primary candidate price exists, but divergence from multiple CEX/indices clearly exceeds the pre-set threshold $\delta_{\max}$.

- The operations account calls `markFailed(marketId)` within the `PendingOps` period (about 5 minutes) to mark as Failed. This call is not allowed for anyone, only for operations accounts (or multisig designated by governance).
- The subsequent procedure is the same as (i). According to the secondary rule, calculate new settlement price $x^{(2)}$ off-chain and reflect on-chain with `manualSettleFailedMarket` within that day.

On-chain Reflection of Secondary Settlement. `manualSettleFailedMarket(marketId, x(2), t(2))` can only be called once for Failed state markets and performs the following.

- Verify market state is `FailedPendingManual` and reject markets where manual settlement is already done.
- Verify $x^{(2)}$ as an OutcomeUnit integer and apply `toTick` to calculate `settlementTickSecondary`.
- Set `settlementValue` to $x^{(2)}$, `settlementTick` to that tick, and record `settlementFinalizedAt` as `block.timestamp`.
- Transition state to `FinalizedSecondary`.

Operations SHOULD submit the manual settlement within the same settlement day (e.g., $[T_{\text{set}}, T_{\text{set}} + 24\text{h})$) so markets do not remain stranded in Failed state. After this, claims work the same as primary settlement; only `settlementTick` comes from the secondary rule price.

(iii) Keeper Failure. Since settlement transactions themselves are permissionless, even if a specific Keeper fails, another party can call `settleMarket` with the same oracle packet. If no one sends `settleMarket` within the settle window, it results in (i), and then manual settlement is done with the secondary rule.

Manual Settlement Within the Day. Fail means "primary price not trusted; settle manually the same day." Data sources/windows/aggregation for the secondary rule live in governance docs; this spec fixes only on-chain invariants (time gates, state transitions, single execution). `settlementTick` is set once and drives CLMSR P&L and position payouts.

7 System Architecture

This section keeps the high-level object model, unit system, and state machines. Detailed math libraries, oracle adapters, upgradeability, and testing notes are collected in Appendix D.

7.1 Implementation Scope & Object Model

This section covers the on-chain objects that implement the semantic objects from Sections 2–6:

- **Market.** Semantic market specification (OutcomeSpec, depth α, fee policy, timing) → `SignalsCoreStorage.Market` fields (tick bounds/spacing, n, `liquidityParameter`, status flags, settlement values).
- **CLMSR state q.** Semantic q vector and partition function $Z(q)$ → per-market `LazyMulSegmentTree.Tree` storing $f_b = e^{q_b/\alpha}$ as factors plus cached sums.
- **LP Vault.** Semantic $(N, S, P, P_{\text{peak}}, \text{drawdown})$ → ERC–4626 LP vault plus `SignalsCoreStorage.VaultState` fields used by the Market-Cycle Batch in Section 3.
- **Risk state.** Semantic $\alpha_{\text{base},t}$, $\alpha_{\text{limit},t}$, drawdown caps, exposure caps → `SignalsCoreStorage.RiskState` consumed by `RiskModule`.
- **Positions.** Semantic range ticket (R, x) and entry cost → ERC–721 `SignalsPosition` token plus per-position metadata in core (market, ticks, quantity, settled flag).
- **Oracle/settlement.** Semantic settlement value/tick → `OracleModule` payload checks and final `Market.settlementTick/settlementValue`.

Each mapping line also marks a responsibility boundary: math libraries own the semantics (CLMSR updates, batch accounting), modules own validation and token/state plumbing, while off-chain infra (frontends, oracle operators, indexers) is out of scope for this layer.

7.2 Global Unit System & Fixed-Point Arithmetic

Units. External token transfers use the settlement token's native decimals (e.g., USDC 6 decimals). Internal CLMSR math uses WAD (10^{18}). Outcome/settlement values use the integer scale 10^d from `OutcomeSpec` (L, U, s, d) (Section 6).

Conversions. Token ↔ WAD conversions occur once on entry to and exit from CLMSR math. The rule is *one conversion, one rounding* per direction to avoid rounding arbitrage.

Rounding policy. Trade costs/fees debit users with rounding *up*; trade proceeds credit users with rounding *down*. Deposit/withdrawal, fee split, and grant rounding follow the vault rules in Section 3.8; rounding cannot make the protocol pay more than the WAD-level economics.

Arithmetic. WAD helpers (`wMul`, `wDiv`, nearest/up variants) are used throughout. Division by zero and overflow are guarded explicitly. Full numeric/rounding rules are fixed in Appendix C; invariant and testing surfaces live in Appendix D.

7.3 Market and Vault State Machines

Core + modules. `SignalsCore` (UUPS proxy) owns storage and exposes all externals. Modules are delegate-only. The main entrypoints are `MarketLifecycleModule` (create/settle/fail/manual settle), `TradeModule` (open/increase/decrease/close/claim), `LPVaultModule` (daily batch), `RiskModule` (limits/drawdown hooks), and `OracleModule` (payload validation/toTick).

Market state machine. Mirrors Section 6: states progress through `Trading`, `SettlementOpen`, then `PendingOps` or `FailedPendingManual`, and finally `FinalizedPrimary` or `FinalizedSecondary`. Each transition is owned by a specific function, and settlement is one-time per market and immutable once set.

Batch execution. The daily LP batch is permissionless. `LPVaultModule.batch()` (or equivalent) may be called once per market-day after that market has reached a finalized state; calls revert if the market is not yet `FinalizedPrimary`/`FinalizedSecondary` or if the batch for that day already ran. Claim and Δ_{claim} delays gate withdrawals/claims, not batch execution.

Upgrade and role boundaries. Math libraries (CLMSR core, segment tree, rounding helpers) are immutable. Modules are replaceable under governance+timelock but are delegate-only; storage and role gating live in `SignalsCore`. All state-changing entrypoints are guarded by roles and pausability scopes sized to lifecycle/config changes, not to math semantics.

Vault batch. Section 3's N_t, S_t, P_t update runs in one daily batch: apply the net LP-vault credit Π_t (derived from L_t, F_t, G_t by the Safety Layer), compute pre/post NAV, process the withdraw queue then deposit queue, and record (N_t, S_t, P_t). Invariants: $P_t = N_t/S_t$ within rounding; shares that rode the same markets see identical basis changes; rounding dust stays internal except the explicit deposit refunds in Section 3.8(b1).

Risk hooks. Drawdown cap, α limits, and exposure checks are enforced via `RiskModule` gates around vault actions and trade sizing.

8 Governance Model & Roles

Parameter catalog details are in Appendix B.

Sections 1–7 defined the CLMSR pricing engine, the daily Market-Cycle Batch and LP Vault, the Safety and Performance layers, and the oracle and settlement architecture. This section describes how those components are governed:

- which parts are fixed as mechanism semantics, - which parts are adjustable on-chain parameters, and - which parts are off-chain operational policies.

It also specifies the roles and authorities of the main actors (LPs, Backstop providers, Treasury, oracle operators, frontends/integrators).

8.1 Governance Surface and Variability

From a governance perspective, Signals v1 has three distinct layers:

1. Mechanism layer (immutable semantics, changed only by versioned upgrades). 2. On-chain configuration layer (governance parameters). 3. Off-chain operational policy layer.

(1) Mechanism layer (immutable semantics) The following elements define the core product and are not affected by routine governance actions. They can change only via explicit protocol upgrades (new deployed versions):

- **CLMSR cost function and price definition.**
  Single potential function

$$C(q) = \alpha \ln Z(q), \quad Z(q) = \sum_b e^{q_b/\alpha}, \quad p_b(q) = \partial C/\partial q_b,$$

with range–binary equivalence, path independence, and loss bounds (Section 2).

- **OutcomeSpec and tick mapping.**
  OutcomeSpec (L, U, s, d), uniform tick set $B = \{0, \ldots, n-1\}$ with $n = (U-L)/s$, and `toTick`(x) that clamps out-of-domain prices to edge ticks (Sections 2, 6).

- **Market-Cycle Batch accounting invariants.**
  Vault state (N, S), basis price $P := N/S$, pre-batch NAV update

$$N_t^{\text{pre}} - N_{t-1} = L_t + F_t + G_t,$$

  single batch price $P_t^e = N_t^{\text{pre}}/S_{t-1}$, and mint/burn rules

$$(N', S') = (N + D, S + D/P), \quad (N'', S'') = (N - xP, S - x)$$

  that preserve $N'/S' = N/S = P$ (Section 3.3).

- **Settlement state machine.**
  Market lifecycle states (Trading, SettlementOpen, PendingOps, FailedPendingManual, FinalizedPrimary,
  FinalizedSecondary), time gates $(T_{\text{set}}, \Delta_{\text{settle}}, \Delta_{\text{ops}}, \Delta_{\text{claim}})$, and the one-time settlementTick semantics (Section 6).

- **Token representation.**
  ERC-4626 LP Vault shares and ERC-721 position NFTs with "single settlement per position" semantics (Section 3.7).

- **Numeric and rounding semantics.**
  WAD math, unit system, and rounding rules in Appendix C and D, including one conversion + one rounding per direction, maker-side dust conventions, and fee split/grant rounding invariants.

- **Fee Waterfall priority and Backstop Grant rule.**
  Four-step Safety mapping (loss compensation → Backstop coverage fill → residual split across LPs/Backstop/Treasury) and the grant sizing rule for G_t in Sections 4.3–4.6, including the constraints $0 \le G_t \le \Delta E_t$ and $B_t = B_{t-1} + F_{\text{BS},t} - G_t$ on admissible priors.

These items are treated as part of the protocol's contractual semantics for daily BTC range markets.

(2) On-chain configuration layer (governance parameters) The following values are configurable parameters that appear in the Safety, Performance, and oracle layers. They are stored in dedicated configuration contracts and can be changed by governance subject to the processes in Section 9.

- **Safety parameters** (Section 4):
  risk-budget parameter λ (with drawdown floor $p_{\text{dd}} := -\lambda$ derived from it), drawdown damping coefficient k for $\alpha_{\text{limit},t}$, withdrawal lag D_{lag}, Backstop coverage targets, and any position limits (per-account and per-ticket exposure caps) enforced by the Risk module.

- **Performance parameters** (Section 5):
  squeeze rate γ, expand rate η, Fee/Loss target ψ_{target}, slippage target Δp_{target}, the whitelist of FeePolicy implementations (contract addresses), and fee-split weights for LP/Backstop/Treasury.

- **Market/product parameters** (Sections 2, 3, 6):
  OutcomeSpec templates (L, U, s, d) for BTC markets, market calendar (e.g., UTC time for T_{set}, trading close), and optional per-account notional or directional limits.

- **Oracle/settlement parameters** (Section 6):
  settlement window Δ_{settle}, PendingOps window Δ_{ops}, claim delay Δ_{claim}, allowed timestamp deviation $\Delta_{\max}$, divergence threshold $\delta_{\max}$, and the set of oracle feed IDs and provider public keys.

Core contracts read these values from configuration contracts; they do not allow arbitrary direct writes. Only governance/operations keys can update configuration, and such updates are controlled by timelocks and access control described in Section 9.

(3) Off-chain operational policy layer Some rules are operational rather than on-chain invariants. They are documented in governance and operations manuals and must remain consistent with the invariant surface above:

- data sources and algorithms for secondary settlement (CEX set, indices, sampling windows, aggregation method such as VWAP/median/TWAP),
- models and procedures for prior estimation at Zero-Hour (combining options markets, historical distributions, internal forecasts) used to build $q_{0,t}$,
- off-chain criteria for `markFailed` decisions (e.g., divergence beyond $\delta_{\max}$, oracle outages),
- Treasury capital policy for LP share purchases (LOLR) and Backstop recapitalization,
- UX policies for displaying settlement windows, withdrawal delay, drawdown caps, and market states in frontends.

These policies can evolve more frequently than on-chain configuration, but they may not override or circumvent contract-level invariants.

8.2 Roles and Authorities

Signals v1 involves several categories of participants. This subsection summarizes, for each one, the main exposures, sources of return, and authorities.

(1) LP Vault LPs LPs supply capital to the LP Vault (Section 3) and hold ERC-4626 shares representing proportional claims on NAV N_t.

- **Exposure.**
  LPs are exposed to the daily pre-batch P&L
  $$\Pi_t = L_t + F_t + G_t$$
  where L_t is CLMSR maker P&L, F_t is LP-attributed fees, and $G_t \geq 0$ is the Backstop Grant (Sections 3.4, 5.1). Under the Safety Layer's depth limits, drawdown cap, and prior-admissibility rule, daily basis-price returns are truncated at p_{dd} or better (Section 4.5).
- **Compensation.**
  LPs participate in basis-price growth $P_t = N_t/S_t$. Fees that remain after loss compensation and Backstop accumulation accrue to N_t via F_t; if the long-run fee–cost condition holds (Section 5.4), P_t grows structurally over time.
- **Rights.**
  LPs may deposit into and withdraw from the Vault via ERC-4626 interfaces, have withdrawals executed after delay D_{lag} at the batch price P_t^e (Sections 3.5, 3.6), and transfer LP shares freely or create secondary markets on external venues (Section 3.7).

(2) Backstop Vault providers Backstop providers supply capital to the Backstop Vault (Section 4.2), which holds mezzanine tail risk.

- **Exposure.**
  Backstop pays Grants $G_t > 0$ on days when raw LP returns would violate the drawdown cap p_{dd}. Under the prior-admissibility rule $\Delta E_t \leq B_{t-1}^{\mathrm{eff}} \leq B_{t-1}$ (Section 4.1), the worst-case grant per day is bounded by ΔE_t, so the Backstop Vault remains solvent on admissible paths (Sections 4.1, 4.6).
- **Compensation.**
  Backstop receives a fee share $F_{\mathrm{BS},t}$ from the daily Waterfall (Section 4.3). Economically, this is a portion of LP fee income carved out to build insurance reserves. When tails do not materialize, this fee share accumulates; when tails hit, reserves are drawn down via G_t.

- **Rights.**
  Backstop providers have claims on Backstop NAV B_t and may, depending on governance design, participate in parameter decisions that primarily affect tail risk (e.g., Backstop fee share, coverage targets).

(3) Treasury Treasury is a protocol-level capital account that plays two roles (Section 4.2):

- **Lender of Last Resort (LOLR).**
  In stress scenarios where withdrawals and losses cause N_t to fall and depth would otherwise exceed $\alpha_{\text{limit},t}$, Treasury can buy LP shares at the batch price, increasing N_t, restoring $E_{t+1} := N_t$ and future depth capacity (Sections 4.5, 9).
- **Operating capital.**
  Treasury funds audits, upgrades, incident response, and may recapitalize Backstop if its NAV falls below target. These flows are governed decisions, not automatic reactions.

Treasury never pays G_t directly; all Backstop Grants are internal transfers from Backstop to LP Vault. In v1, entities controlling Treasury typically also control configuration contracts and upgrade proxies, subject to the constraints in Section 9.

(4) Oracle providers, keepers, and operations Oracle and settlement responsibilities are split as follows (Section 6):

- **Oracle providers** maintain signed BTC/USD feeds that satisfy freshness and quality constraints. They are responsible for data integrity and availability within the allowed timestamp window $\Delta_{\max}$.
- **Keepers** (any addresses) can call `settleMarket` during the settlement window. They are not trusted for content, only for liveness; all critical checks (signature, feedId, timing, normalization) are enforced on-chain.
- **Operations accounts** are governance-controlled multisigs with limited powers:
  - call `markFailed` during PendingOps for markets whose primary price fails sanity checks,
  - call `manualSettleFailedMarket` once with the secondary-settlement price computed according to published rules (Section 6).

(5) Frontends and integrators Frontends and external protocols are not part of the core mechanism but materially affect user risk perception:

- Frontends must accurately display settlement times, windows, the withdrawal delay D_{lag}, drawdown caps, and market states.
- Integrations that use LP shares or position NFTs as collateral must clearly indicate any additional leverage, liquidation logic, or risk transformation beyond what Signals itself defines.

These actors do not have protocol-level authority, but their design choices influence how effectively users can interpret the on-chain semantics.

9 Change Process & Decentralization

Governance must be able to adjust parameters and, when necessary, take emergency actions without breaking the mechanism semantics or surprising LPs and traders. This section describes the normal change process, emergency tools, and the relationship to secondary settlement. Section 9.4 then sketches a decentralization roadmap, and Section 9.5 summarizes trust assumptions and the threat model.

9.1 Normal parameter changes

In v1 it is useful to distinguish between:

- **controller outputs**, which are expected to move every market-day (for example the next-day depth α_t chosen inside Safety caps, the Zero-Hour prior $q_{0,t}$, and the Fee policy selected for each market from a whitelisted family); and
- **configuration parameters**, which define the Safety envelope and these controllers (for example λ, k, D_{lag}, Backstop coverage targets, depth and slippage targets, oracle windows, and the whitelist of allowed FeePolicy implementations and product templates).

Daily tuning of controller outputs inside the Safety envelope — choosing $(\alpha_t, q_{0,t}, \text{Fee policy})$ for each market t with $\alpha_t \leq \alpha_{\text{limit},t}$ and admissible priors — is treated as normal operation and does not itself go through the governance proposal process.

By contrast, changing configuration parameters (adjusting λ or k, modifying Backstop coverage targets or fee-split weights, altering oracle windows, or changing the set or ranges of allowed FeePolicy implementations and OutcomeSpec templates) follows a standard governance flow:

Routine changes to Safety, Performance, oracle, and product configuration parameters follow a standard flow:

1. **Proposal.**
   Governance participants publish a proposal specifying:
   - the parameters and before/after values,
   - the rationale and expected impact on Safety, Performance, and UX,
   - any dependencies (e.g., if one change assumes another has been executed).
2. **Off-chain review.** A minimum review period allows LPs, Backstop providers, and other stakeholders to comment via forums or other governance channels.
3. **On-chain timelock.** Approved proposals are queued in configuration contracts with a timelock. Safety-critical parameters (e.g., λ, D_{lag}, Δ_{settle}) may have longer timelocks and higher approval thresholds than purely UX-related ones.
4. **Execution.** After the timelock, authorized addresses call configuration setters (e.g., `setRiskParams`, `setFeeParams`, `setOracleParams`).
   Events record the change and the proposal ID for auditability.

9.2 Emergency measures and circuit breakers

In exceptional conditions (oracle failures, discovered vulnerabilities, Backstop depletion, governance compromise risk), an emergency multisig may have limited authority to take fast, scoped actions without going through the full proposal cycle. Typical measures include:

- temporarily pausing new market creation or new trading for specific markets,
- temporarily pausing settlement or position claims,
- forcing depth control into "squeeze only" mode (disabling expansion) for new cycles,
- lowering per-account limits or disabling leverage in connected products where applicable.

Emergency actions must be strictly scoped (which functions/markets are affected), be time-bounded (with explicit sunset or requirement for formal ratification), and be reflected in on-chain state and off-chain communications so users can detect them.

9.3 Secondary settlement process

Section 6 defines secondary settlement for markets where the primary rule fails. Governance and operations apply it as follows:

1. During PendingOps, operations may call `markFailed` for a market whose primary candidate price is missing or obviously invalid (e.g., divergence beyond $\delta_{\max}$ from reference data).
2. Off-chain, operations compute a secondary settlement price $x^{(2)}$ and timestamp $t^{(2)}$ according to a published secondary rule (data sources, sampling window, aggregation method).
3. Within the same settlement day, operations call `manualSettleFailedMarket(marketId, x, t)` once to set `settlementValue` and `settlementTick` from that secondary price and transition the market to `FinalizedSecondary`.

Claim gating and single-settlement semantics remain unchanged: positions can be claimed only after the claim delay Δ_{claim}, and each position can be settled exactly once.

9.4 Progressive Decentralization Roadmap

Signals v1 is not fully permissionless at launch. Instead, decentralization of both operations and governance is staged as the mechanism and its implementation are hardened.

- **Phase 0 – Public testnet validation.**
  Core contracts are first deployed to a public testnet with upgradeable proxies. Governance and operations keys are already held by disclosed multisigs, but all capital is team/partner capital and markets run with test or tightly capped incentive balances. The main focus is validating CLMSR semantics, batch accounting, oracle integration, and Safety invariants under realistic order flow.
- **Phase 1 – Guarded mainnet launch with internal capital.**
  The full mechanism is deployed on mainnet. LP Vault and Backstop balances are funded entirely with protocol-owned or partner capital; permissionless LP deposits are not yet enabled. Configuration contracts and upgrade proxies are controlled from the outset by disclosed multisigs with timelocks on critical changes. Emergency procedures and circuit breakers from Section 9 are wired and documented but used conservatively. The goal is to operate the production system with real traders while keeping balance-sheet risk and attack surface tightly bounded.
- **Phase 2 – Public LP access under caps.**
  Permissionless LP deposits into the Vault are opened, subject to caps on LP/Backstop TVL, supported assets, and listed markets. Risk parameters, Fee policy, and oracle settings remain governed by core-team multisigs with on-chain timelocks, but the proposal and review process from Section 9 is fully public so that LPs and integrators can monitor changes and their rationale.
- **Phase 3 – Community governance layer.**
  Authority over configuration parameters (fees, depth-control targets, oracle parameters, outcome templates) begins to shift to community-based processes (e.g., token voting, elected councils, or delegated guardians). LPs, Backstop providers, and other stakeholders can formally participate in Safety and Performance parameter setting. Oracle provider selection and secondary settlement rules are progressively brought under community governance.
- **Phase 4 – Mechanism hardening and authority minimization.**
  Core mechanism contracts (CLMSR engine, Market-Cycle Batch, Vault accounting) are locked as non-upgradeable. Only configuration parameters remain adjustable, via transparent community governance. Emergency powers are narrowly scoped, time-limited, and fully observable on-chain; users can always exit over a bounded time window without relying on off-chain assurances.

Throughout this roadmap, the goal is to converge to a state where:

- core semantics are immutable and independently auditable, - parameter changes are governed by open, predictable processes, and - no single party can seize user funds or silently rewrite economic rules.

9.5 Trust Assumptions and Threat Model

This subsection summarizes the main trust assumptions and threat categories implicit in the design.

Trust assumptions Participants implicitly assume:

- **Chain security.**
  The underlying blockchain executes contracts correctly and provides finality.
- **Mechanism implementation correctness.**
  The deployed contracts correctly implement the CLMSR cost function and Market-Cycle Batch semantics from Sections 2–3 and have undergone sufficient testing and external audit.
- **Governance good faith.**
  Holders of configuration/upgrade/operations keys act to preserve Safety and Performance objectives and do not attempt to expropriate LP or trader assets.
- **Oracle integrity.**
  Oracle providers do not systematically manipulate prices over time, and operations use `markFailed` and the secondary rule according to published criteria.

Major threats and mitigations Representative threat classes and the corresponding mitigations are:

- **Market risk.**
  Extreme BTC moves, regime shifts, or mis-specified priors can cause large losses for LPs and Backstop. Mitigations: entropy budgets and depth caps ($\alpha_{\text{limit},t}$), drawdown cap p_{dd}, Backstop/Treasury tranching, and withdrawal lag (Section 4).
- **Oracle and data manipulation.**
  Adversaries may try to distort prices around T_{set} or inject bad data. Mitigations: signed pull oracle, timestamp window $\Delta_{\max}$, multiple external references and secondary settlement rules, and `markFailed` + manual settlement path (Section 6).
- **Liquidity and bank runs.**
  After drawdowns, LPs may rush to exit. Mitigations: fixed withdrawal lag D_{lag}, Market-Cycle Batch processing with a single price per day, and optional Treasury LOLR interventions (Sections 3.6, 4.5).
- **Smart contract and governance risk.**
  Implementation bugs or key compromise could break invariants. Mitigations: external audits, bug bounties, multisig + timelocks, monitoring of configuration changes, and the ability for users to exit over time.

Non-goals Signals v1 explicitly does *not* aim to:

- eliminate all losses for LPs or Backstop providers,
- guarantee principal or returns,
- protect users from losses due to their individual trading decisions.

The governance parameters and policy levers in this section are intended to express risk within explicit, transparent constraints that can be monitored and adjusted in a predictable way on top of the Safety and Performance structure.

10 Conclusion

This document has specified Signals Protocol v1 as an implementation of daily BTC range markets built around a CLMSR pricing engine, a batched LP Vault, and an explicit capital stack with tail protection.

At the mechanism level, a single potential function $C(q) = \alpha \ln Z(q)$ over a discretized price axis prices all ticks and ranges, enforces normalized probabilities, and yields a closed-form bound on entropy loss. Traders interact with this engine through range tickets: arbitrary contiguous subsets of ticks purchased as atomic claims. Maker P&L is path-independent and determined solely by the start and end CLMSR states for each daily market, so "what happened in between" does not affect the final accounting.

The Market-Cycle Batch and LP Vault structure turns this maker P&L into daily equity returns for LPs via a single batch price. A pre-batch NAV identity

$$N_t^{\text{pre}} - N_{t-1} = L_t + F_t + G_t$$

summarizes, for each day t, CLMSR P&L L_t, LP-attributed fees F_t from the Fee Waterfall, and any Backstop Grant G_t used to enforce the daily drawdown floor. Deposits and withdrawals are processed once per batch at that day's price, with a withdrawal lag that ties each share's economics to the exact list of markets it underwrote.

On top of this accounting spine, the Safety Layer defines hard invariants. Depth is tied to LP capital via a single risk-budget parameter λ, producing a static cap $\alpha_{\text{base},t}$ and a drawdown-damped cap $\alpha_{\text{limit},t}$. LP basis-price returns are bounded below by a daily floor $p_{\text{dd}} := -\lambda$, enforced when necessary by Backstop Grants that truncate tails while keeping the Backstop Vault non-negative on admissible priors. Liquidity guards — withdrawal lag and a constrained LOLR role for Treasury — prevent instantaneous depth collapse even under stressed outflows.

The Performance Layer operates strictly within this envelope. It chooses priors, depth, and Fee policy each day so that empirical Fee/Loss ratios and slippage stay near governance targets while respecting all Safety constraints. In effect, Safety fixes the box in which the system is allowed to live, and Performance is the controller that moves the protocol within that box in response to realized trading, P&L, and liquidity conditions.

The oracle and implementation architecture show how to realize these semantics on an EVM rollup with signed pull oracles, bounded settlement windows, module-based core contracts, immutable CLMSR math libraries, and explicit numeric and rounding invariants. Settlement is handled through a primary rule (closest signed sample to the reference time) with a constrained, auditable secondary rule for failure cases, so that every daily market can be finalized and settled exactly once.

Open directions and next steps include:

- **Mechanism and modeling.** Richer models for priors and depth control, better understanding of trader behavior under CLMSR with range tickets, and empirical calibration of the single risk-budget parameter λ across regimes.
- **Product surface.** Extending from single-asset BTC to multi-asset and multi-tenor markets, and designing cross-market risk management that keeps the same CLMSR/Safety interface while allowing more complex payoff structures.
- **LP product design.** Evolving from a single homogeneous LP Vault to LP-selectable risk tranches, multi-vault depth allocation, and LP-weighted preference aggregation for priors, depth, and fee curves, without breaking the Market-Cycle Batch invariants.
- **Governance and decentralization.** Progressively decentralizing control over Safety and Performance parameters and oracle policy, moving configuration and secondary-settlement rules from team-operated multisigs toward open, on-chain governance while keeping core semantics stable.
- **Engineering and operations.** Hardening the implementation, extending test and invariant coverage, improving monitoring of Safety and Performance metrics, and keeping "spec-as-code" aligned with the CLMSR, batch, and capital-stack semantics described here.

We intend this specification to serve as a reference design for continuous-outcome prediction markets: a pattern for composing cost-function market makers, vaults, risk tranching, and oracle infrastructure into a solvency-aware protocol that can be safely operated, audited, and extended.

A Notation & State Variable Table

This appendix summarizes the core symbols used throughout the paper. Each symbol is defined where it first appears; this table is a cross-reference only.

A.1 CLMSR and Outcome Space

Symbol	Meaning
(L, U, s, d)	OutcomeSpec: lower/upper bounds, tick spacing, decimals (Sections 2, 6)
B, n	Tick set $B = \{0, \ldots, n-1\}$, tick count $n = (U-L)/s$
q, q_i	CLMSR state vector and cumulative issuance per tick
$C(q)$	CLMSR cost function $C(q) = \alpha \ln Z(q)$
$Z(q)$	Partition sum $Z(q) = \sum_b e^{q_b/\alpha}$
$p_b(q)$	Tick prices/probabilities $p_b(q) = \partial C/\partial q_b$
$R \subseteq B$	Range (set of ticks) for a range ticket

A.2 Daily P&L and Fees

Symbol	Meaning
L_t	CLMSR maker P&L for market t, $L_t = C(q_{\text{end},t}) - C(q_{\text{start},t})$
L_t^-	Realized entropy loss $L_t^- := \max(-L_t, 0)$
$F_{\text{tot},t}$	Gross trading/settlement fees generated in market t
$F_{\text{loss},t}$	Portion of $F_{\text{tot},t}$ used immediately to offset CLMSR loss
$F_{\text{pool},t}$	Remaining fee pool after loss compensation: $F_{\text{pool},t} := F_{\text{tot},t} - F_{\text{loss},t}$
$F_{\text{LP},t}$	Residual fee share allocated to the LP Vault (after Backstop target filling)
$F_{\text{BS},t}$	Fee share allocated to the Backstop Vault (coverage fill + residual share)
$F_{\text{TR},t}$	Fee share allocated to Treasury
$F_{\text{dust},t}$	Rounding dust from residual fee distribution, $F_{\text{dust},t} \in \{0, 1\}$, credited to LPs
F_t	LP-attributed fee term: $F_t := F_{\text{loss},t} + F_{\text{LP},t}$ (dust absorbed into $F_{\text{LP},t}$)
$\psi_{\text{fee},t}$	Realized Fee/Loss ratio: $\psi_{\text{fee},t} := F_t / L_t^-$ if $L_t^- > 0$, else $+\infty$
ψ_{target}	Governance target for long-run Fee/Loss ratio

A.3 Capital Accounts and Batch States

Symbol	Meaning
N_t	LP Vault NAV at end of batch t
S_t	LP Vault share count at end of batch t
P_t	LP Vault basis price $P_t := N_t/S_t$ at end of batch t
P_t^{peak}	Peak basis price to date, $P_t^{\text{peak}} := \max_{\tau \le t} P_\tau$
DD_t	Drawdown $DD_t := 1 - P_t/P_t^{\text{peak}}$
E_t	LP capital at start of market t ($E_t := N_{t-1}$)
N_t^{pre}	Pre-batch NAV after daily P&L: $N_t^{\text{pre}} := N_{t-1} + L_t + F_t + G_t$
P_t^{e}	Batch equity price applied to existing shares: $P_t^{\text{e}} := N_t^{\text{pre}}/S_{t-1}$
B_t	Backstop Vault NAV at end of batch t
T_t	Treasury NAV at end of batch t
G_t	Backstop Grant (Backstop $\to$ LP Vault) for drawdown-cap enforcement
Π_t	LP Vault daily P&L (pre-batch): $\Pi_t := N_t^{\text{pre}} - N_{t-1} = L_t + F_t + G_t$

Raw NAV/price variables used only in Safety/Waterfall definitions:

Symbol	Meaning
N_t^{raw}	Raw NAV after CLMSR P&L and loss-offset fees: $N_t^{\text{raw}} := N_{t-1} + L_t + F_{\text{loss},t}$
P_t^{raw}	Raw basis price before grant: $P_t^{\text{raw}} := N_t^{\text{raw}}/S_{t-1}$

These appear only in Section 4.6 to size G_t.

A.4 Safety and Performance Parameters

Symbol	Meaning
λ	Dimensionless risk-budget parameter; under a uniform prior $\alpha_t \ln n = \lambda E_t$ and the drawdown cap is $p_{dd} := -\lambda$ (Sections 4.1, 4.4)
α_t	Depth used in daily market t
$\alpha_{\text{base},t}$	Static depth cap from entropy budget/capital (Section 4.4)
$\alpha_{\text{limit},t}$	Drawdown-damped depth cap, $\alpha_{\text{limit},t} \le \alpha_{\text{base},t}$
$E_{\text{ent}}(q_{0,t})$	Entropy loss budget for market t from opening state $q_{0,t}$ (Section 4)
ΔE_t	Incremental entropy-tail budget: $\Delta E_t := E_{\text{ent}}(q_{0,t}) - \alpha_t \ln n$; allocated to Backstop (Sections 4.1, 4.2)
p_{dd}	Daily drawdown cap for LP basis-price return, defined as $p_{dd} := -\lambda$
D_{lag}	Fixed withdrawal lag (request $\to$ batch execution)
ρ_{BS}	Backstop coverage target ratio (Backstop NAV vs LP NAV), Appendix B
Δp_t	Two-sided average slippage indicator for trades in market t that respect configured position-size limits (Section 5.1)
Δp_{target}	Governance slippage target

The full parameter catalog (fee-schedule parameters, oracle windows, market templates) is in Appendix B.

B Parameter Catalog

This appendix lists only governance-tunable parameters. Formal definitions and formulas are in Sections 4–5 (Safety/Performance) and Section 6 (Oracle); implementation-level use lives in Section 7.

Safety Parameters

λ Effective single-market worst-case loss ratio under a uniform prior; $\alpha_{\text{base},t} = \lambda E_t / \ln n$ and $p_{dd} := -\lambda$ are derived from this.

p_{dd} Daily drawdown cap for LP basis price (defined as $p_{dd} = -\lambda$).

k Drawdown decay coefficient used in $\alpha_{\text{limit},t}$.

D_{lag} Fixed withdrawal delay.

Backstop coverage target Target ratio of Backstop NAV to LP Vault NAV.

Performance Parameters

γ Depth squeeze rate.

η Depth expand rate.

ψ_{target} Long-run Fee/Loss ratio target.

Δp_{target} Slippage target for typical allowed trades (see Section 5.1).

Fee policy whitelist Addresses of whitelisted FeePolicy implementations that map trades/settlements to $F_{\text{tot},t}$.

Fee Waterfall weights Residual split ratios for LP/Backstop/Treasury after loss compensation and Backstop coverage fill (e.g., parameters $(\phi_{\text{LP}}, \phi_{\text{BS}})$ with $\phi_{\text{TR}} := 1 - \phi_{\text{LP}} - \phi_{\text{BS}}$).

Market/Product Parameters

(L, U, s, d) OutcomeSpec (domain bounds, tick spacing, decimals).

T_{set} Settlement reference time (trading closes just before T_{set}; see Section 6).

Position limits Per-account exposure caps and per-ticket maximum notionals, used both as risk limits and as UX levers to keep single-trade slippage bounded (Section 5.1).

Oracle/Settlement Parameters

Δ_{settle} Settlement window length.

Δ_{ops} PendingOps window length.

Δ_{claim} Claim delay after Finalized.

Δ_{max} Allowed price timestamp deviation.

δ_{max} Divergence threshold for `markFailed`.

C Numerics & Rounding Rules

External settlement currency is USDC 6 decimals, internal operations are processed in WAD (10^{18}). For a single transaction, only **one conversion, one rounding** of "external→internal→cost/price calculation→external" is allowed to prevent rounding arbitrage. Fees are imposed and distributed outside the cost function to preserve the normalization/additivity semantics that interpret p as "probability". Trade/LP/fee/grant rounding and dust rules are summarized in Section 3.8; this appendix and Section 7.2 are the canonical specifications.

C.1 Rounding and Dust Handling in the Vault

All vault accounting is in integer settlement-token units; rounding and dust ownership are fixed as follows so that the continuous invariant $N'/S' = N/S$ from Section 3.3 is respected up to at most one base unit of error.

(a) Trade rounding. CLMSR costs/proceeds are computed in WAD, then converted to the 6-dec token with *debits rounded up, credits rounded down*. Per-trade dust is at most one base unit and stays on the maker side, flowing into L_t or F_t (see Section 7.2).

(b1) Deposits (asset→share). At batch basis price $\widetilde{P}_t$,

$$S_{\text{mint}} := \left\lfloor \frac{A}{\widetilde{P}_t} \right\rfloor, \qquad A_{\text{used}} := S_{\text{mint}} \cdot \widetilde{P}_t.$$

The residual $A - A_{\text{used}} \in \{0, 1\}$ (at most one base unit) is *immediately refunded* to the depositor in the same transaction. The vault never retains deposit dust.

(b2) Withdrawals (share→asset). Burn x shares and pay

$$A_{\text{wd}} := \left\lfloor x \cdot \widetilde{P}_t \right\rfloor.$$

Rounding dust $x\widetilde{P}_t - A_{\text{wd}} \in [0, 1)$ stays in the vault and accrues to remaining LPs via a slightly higher basis price. Credits are always truncated, so users never receive more than the WAD-level value.

(c) Fee waterfall splits. Let $F_t^{\text{remain}} \geq 0$ be the integer fee pool remaining after loss compensation and any Backstop-coverage fill in the Fee Waterfall (Steps 1–3 of Section 4.3). Given governance weights $(\phi_{\text{LP}}, \phi_{\text{BS}}, \phi_{\text{TR}})$ with $\phi_{\text{LP}} + \phi_{\text{BS}} + \phi_{\text{TR}} = 1$, the on-chain split chooses non-negative integers

$$F_{\text{LP},t}^{\text{core}},\ F_{\text{BS},t}^{\text{core}},\ F_{\text{TR},t}^{\text{core}}$$

such that

$$0 \leq F_{\text{LP},t}^{\text{core}}, F_{\text{BS},t}^{\text{core}}, F_{\text{TR},t}^{\text{core}}, \qquad F_{\text{LP},t}^{\text{core}} + F_{\text{BS},t}^{\text{core}} + F_{\text{TR},t}^{\text{core}} \leq F_t^{\text{remain}}.$$

The rounding dust

$$F_{\text{dust},t} := F_t^{\text{remain}} - \left(F_{\text{LP},t}^{\text{core}} + F_{\text{BS},t}^{\text{core}} + F_{\text{TR},t}^{\text{core}}\right) \in \{0, 1\}$$

is credited entirely to the LP Vault; Backstop and Treasury never participate in this dust. The residual-distribution components are then

$$F_{\text{LP},t} := F_{\text{LP},t}^{\text{core}} + F_{\text{dust},t}, \qquad F_{\text{BS},t}^{\text{res}} := F_{\text{BS},t}^{\text{core}}, \qquad F_{\text{TR},t} := F_{\text{TR},t}^{\text{core}}.$$

The full Backstop fee share including coverage-target accumulation is $F_{\text{BS},t} := F_{\text{fill},t} + F_{\text{BS},t}^{\text{res}}$ where $F_{\text{fill},t}$ is defined in Section 4.3. The LP fee term entering daily P&L is

$$F_t := F_{\text{loss},t} + F_{\text{LP},t},$$

consistent with Sections 3.4 and 5.1; the dust $F_{\text{dust},t}$ is already absorbed into $F_{\text{LP},t}$. Backstop and Treasury fee shares update as $B_t - B_{t-1} = F_{\text{BS},t} - G_t$ and $T_t - T_{t-1} = F_{\text{TR},t} + (\text{other flows})$; rounding never causes payouts to exceed WAD-level economics.

(d) Backstop Grants G_t (tail cut, one-way). From the continuous recommendation

$$G_{\min,t} = (1 + p_{dd}) P_{t-1} S_{t-1} - N_t^{\text{raw}}, \qquad N_t^{\text{raw}} := N_{t-1} + L_t + F_{\text{loss},t},$$

on-chain uses

$$G_t := \max\{0, \lceil G_{\min,t} \rceil\},$$

decreasing Backstop and increasing the Vault by the same integer G_t. This conservatively enforces the drawdown cap with at most one base unit of extra cost. All Backstop upside and recovery is modeled via its fee waterfall share $F_{\text{BS},t}$.

D Implementation Blueprint & Testing

D.1 CLMSR Engine: Tree, Cost, and Complexity

State representation. CLMSR weights $f_b = e^{q_b/\alpha}$ are stored in a lazy range-multiplication segment tree. Leaves store f_b (WAD), internal nodes cache subtree sums, and pending factors accumulate lazy updates. The root sum is $Z(q)$.

Range updates/queries. Buying x on range $[l, h]$ applies factor $e^{x/\alpha}$ to that interval via `applyFactor` in $O(\log n)$. Range sums `sumRange` return Z_R. Both operations honor pending factors and keep `cachedRootSum` synchronized.

Safe exponent and chunking. Exponent input $z = (\Delta q/\alpha)$ is constrained to $z \le z_{\max}$. If a trade would exceed this, the library computes a safe chunk size deterministically from inputs and applies the trade in multiple chunks, each within domain, summing costs so that path-independence holds at WAD precision.

Invariants/tests. Tree invariants: $f_b > 0$; buys on positive-mass ranges yield $Z' > Z$; root sum matches composed factors. Property tests include round-trip checks (buy then sell same range/size approximately restores distribution) and partition normalization.

D.2 Oracle Adapter and Settlement

OracleModule. Stores per-market oracle feed/config, verifies signatures and staleness, checks $|t - T_{\text{set}}| \le \Delta_{\max}$, converts price to outcome units, and exposes a single interface to obtain a candidate settlement price.

`settleMarket`. Permissionless; validates the payload, picks the closest sample to T_{set}, and records `settlementValue` and `settlementTick`.
It then transitions to `PendingOps` or `FinalizedPrimary` per Section 6. The secondary path is handled by `markFailed` and `manualSettleFailedMarket`; in both paths, `settlementTick` is set exactly once.

Snapshots/indexing. Post-settlement, `requestSettlementChunks` emits bounded chunk events to let off-chain indexers process positions without per-position on-chain events; claims use on-chain view math so on/off-chain payouts match.

D.3 Upgradeability, Config, and Security Guards

Upgrade pattern. Only `SignalsCore` and `SignalsPosition` are UUPS proxies. Modules are replaceable logic targets called via `delegatecall`. Storage layout is centralized in `SignalsCoreStorage` and `SignalsPositionStorage`.

Config layer. Config/env contracts hold governance parameters (fees, α caps, λ, $\Delta_{\max}$, D_{lag}, exposure limits). Modules read these values when enforcing the invariants from Sections 4–6.

Guards. External entrypoints use `nonReentrant`/`whenNotPaused` and role checks. Oracle/settlement functions validate feed IDs, timestamps, and signature keys. Token interactions are constrained to whitelisted assets. Pausing and owner hooks are scoped to lifecycle/config changes, not to math semantics.

D.4 Invariants, Testing, and Spec-as-Code

Invariants (summary). CLMSR: non-negative weights, path-independent costs, loss bound via $\alpha \ln n$, partition normalization. Market/Vault: one settlement per market; $N_t^{\text{pre}} - N_{t-1} = L_t + F_t + G_t$ (pre-batch); shares riding the same markets see the same basis changes; withdrawal lag and drawdown caps are enforced; Backstop balance follows $B_{t+1} = B_t + F_{\text{BS},t} - G_t$. Oracle: single `settlementTick` with time gates and $\Delta_{\max}$.

Testing surface. Unit tests for validation/state machines; parity tests for CLMSR math vs. reference engine; property tests for tree consistency and round-trips; batch/NAV tests for vault accounting; ora-

cle/settlement edge-case tests. v0 parity tests cover SDK semantics from earlier internal versions where applicable.

Spec-as-code. The whitepaper (Sections 2–7) and code-level invariant docs/tests are kept in sync; changing mechanism libraries or their invariants is treated as a protocol version change, not an implementation detail.

References

[1] R. Hanson. Logarithmic market scoring rules for modular combinatorial information aggregation. *Journal of Prediction Markets*, 1(1):3–15, 2007.

[2] J. Abernethy, Y. Chen, and J. Wortman Vaughan. An optimization-based framework for automated market-making. In *Proceedings of the 14th ACM Conference on Electronic Commerce*, pages 297–314, 2013.

[3] J. Wolfers and E. Zitzewitz. Prediction markets. *Journal of Economic Perspectives*, 18(2):107–126, 2004.

[4] J. Moody et al. EIP-4626: Tokenized Vault Standard. Ethereum Improvement Proposal 4626, 2022.

[5] RedStone Team. RedStone Oracles Documentation. https://docs.redstone.finance/, accessed 2025.

[6] Chainway Labs. Citrea: Bitcoin-native ZK Rollup. https://citrea.xyz/, accessed 2025.

[7] Signals Research Team. CLMSR Mechanism Whitepaper. Technical report, Signals Protocol, 2025.

Disclaimer This section is non-normative and does not define protocol behaviour; it is provided for legal and risk-disclosure purposes only.

This document ("Whitepaper") describes the design and intended operation of the Signals protocol, an on-chain system for daily Bitcoin range markets built on a CLMSR-based automated market maker, a batched LP Vault, and associated Safety, Performance, oracle, and governance components.

1. **Informational / technical document only**
This Whitepaper is provided solely for informational and technical discussion purposes. Nothing in this document constitutes, or should be interpreted as, financial, investment, legal, accounting, or tax advice. Readers should make their own independent assessment and, where appropriate, consult qualified professional advisers before providing liquidity, trading, integrating with, or otherwise using any implementation of the Signals protocol.

2. **No offer or solicitation**
This Whitepaper is not a prospectus, offering memorandum, or similar disclosure document and is not intended to form the basis of any investment decision. It does not constitute an offer to sell, or a solicitation of an offer to buy, any token, security, derivative, or other financial instrument in any jurisdiction. Any future issuance of tokens, governance rights, or other instruments, if any, will be governed by separate documentation and may be subject to additional legal and regulatory requirements.

3. **Experimental protocol, market risk, and possibility of loss**
The Signals protocol, including the CLMSR engine, Market-Cycle Batch, LP Vault, Safety Layer (entropy budgets, depth caps, Backstop Grants), Performance Layer, oracle system, and implementation blueprint, is experimental. Smart contracts, fixed-point math, rounding rules, data structures (such as segment trees), oracles, and surrounding infrastructure may fail, be exploited, or behave in unexpected ways.

Participation as an LP, Backstop provider, trader, integrator, or governance participant may result in partial or total loss of capital. Losses may arise, among other things, from CLMSR maker losses exceeding expectations, depletion of Backstop funds or failure to enforce drawdown limits, oracle errors or manipulation, implementation bugs in on-chain or off-chain components, large or rapid moves in the underlying asset price, mis-specified priors and depth settings, fee policies that fail to cover realized losses, and correlated withdrawals or other concentrated participant behaviour. Safety mechanisms including

entropy budgets, α-limits, drawdown caps, the Backstop Vault, withdrawal lags, and LOLR actions are intended to bound certain risks but do not eliminate the possibility of significant or total loss.

4. **Regulatory and eligibility considerations**
The legal and regulatory treatment of on-chain prediction markets, automated market makers, vaults, and related tokens varies by jurisdiction and is evolving. The Signals protocol, as described in this Whitepaper, has not been reviewed or approved by any regulator. There is no guarantee that it complies with the laws or regulations of any particular jurisdiction, or that future legal or policy changes will not adversely affect it.

Access to or use of any implementation of the protocol may be restricted or prohibited for certain persons (for example, based on nationality, residence, sanctions status, or regulatory classification). Each reader and potential user is solely responsible for determining whether they are permitted to access or use the protocol and for complying with all laws and regulations that apply to them.

5. **No warranties; limitation of liability; priority of code**
This Whitepaper and any related materials are provided "as is" and "as available", without any representation or warranty of any kind, whether express, implied, or statutory, including, without limitation, any warranties as to accuracy, completeness, fitness for a particular purpose, or non-infringement.

To the maximum extent permitted by applicable law, the authors of this Whitepaper, contributors to the Signals protocol, and any related persons or entities disclaim all liability for any direct, indirect, incidental, consequential, special, exemplary, or punitive damages (including loss of profits, loss of data, or loss of opportunity) arising out of or in connection with (a) the use of or reliance on this Whitepaper, (b) any interaction with or use of the Signals protocol or related contracts, interfaces, or infrastructure, or (c) any third-party services, integrations, or dependencies.

In the event of any inconsistency between this Whitepaper and the behaviour of deployed smart contracts, the behaviour of the deployed contracts prevails. This Whitepaper is not a binding specification or contract and does not create any contractual rights or obligations.