Technical Brief: Systematic Risk Controls & Execution Framework

Sonia Kolasinska, IndexMaker Labs

December 30, 2025

Contents

1 Executive Summary

The protocol utilizes a deterministic execution engine modeled on physical laws (Numerical Poisson Solver) to manage the convergence of user intent (Index orders) into physical asset positions. The system is governed by a multi-layered constraint architecture—incorporating individual **Max Order Size**, aggregate **Max Volley Size**, and a rigorously derived **Capacity Limit (CL)** — ensuring market stability, margin integrity, and “Best Execution” compliance.

2 Operational Architecture

2.1 Step A: Intent Vectorization & Trustless Keeper Facilitation

Users initiate orders by providing collateral and a buy/sell intent. To prevent systemic limit breaches, the transaction is capped by the market's remaining margin capacity (**M**) relative to current long (Δ_L) and short (Δ_S) exposures.

- **Trustless & Cryptographically Bound Keeper Service:** The Keeper service is a permissionless component that facilitates intent agreement. It acts as a neutral orchestrator, submitting a verified execution package that it cannot modify.
- **Asset Contribution Fractions (F_{ij}):** The Keeper derives the fractions F_{ij} (for asset i in order j) to ensure **fair capacity distribution** across a batch of m index orders.
  - **Execution Coefficient (K_j):** The scaling factor K_j for order j is:

$$K_j = \min\left(1, \min_i\left(\frac{L_i}{A_{ij}}\right)\right)$$

    where:

$$L_i \quad \text{asset liquidity of } i\text{-th asset}$$

    and:

$$A_{ij} \quad \text{asset quantity } i\text{-th asset } j\text{-th index order}$$

  - **Asset Contribution Fraction (F_{ij}):** Derived from the progress coefficient K_j:

$$F_{ij} = K_j \cdot \frac{A_{ij}}{L_i}$$

- **The Margin Vector (M) Derivation:** M is a per-asset vector derived from the **Max Volley Size** (V_{max}). For n assets, the allowed quantity M_i for each asset i is:

$$M_i = \frac{V_{\text{max}}/n}{P_i}$$

  Ensuring that the aggregate value $\sum_{i=1}^{n}(M_i \cdot P_i) = V_{\text{max}}$.
- **Margin Limit Vector (L):** The available capacity (**L**) is the sum of the system's available resources: the Inventory that can be closed (Δ_L or Δ_S) plus the Remaining Margin Headroom, capped by the equivalent vector for the scalar Market Capacity ($C \odot W$).
  - **For a Buy Order** (either reducing Δ_L or increasing Δ_S):

$$\mathbf{L} = \Delta_L \oplus \min(\mathbf{M} \ominus \Delta_S, C \odot \mathbf{W})$$

  - **For a Sell Order** (either reducing Δ_S or increasing Δ_L):

$$\mathbf{L} = \Delta_S \oplus \min(\mathbf{M} \ominus \Delta_L, C \odot \mathbf{W})$$

- **Capacity Limit (CL):** Derived from the Margin Vector **M** and Asset Weights **W**:

$$CL = \min\left(\mathbf{F} \odot \frac{\mathbf{L}}{\mathbf{W}}\right)$$

  where:

$$F \quad \text{asset contribution fractions}$$

  and:

$$L \quad \text{margin limit vector}$$

  and:

$$W \quad \text{asset weights}$$

2.2 Step B: The Vendor (AP) Solver Logic

The Vendor service acts as a "physics engine" observing the Delta vector. It is coded as a **numerical PDE (Poisson) solver** using a gradient descent method:

- **Observation:** The service monitors the current Δ and historical states to calculate the velocity ($d\Delta$) and acceleration ($d^2\Delta$) of the exposure gap.
- **State Aggregation:** The Delta is persistent across execution cycles. In any given cycle i, the new Delta state is defined by the previous state and the disturbance introduced by user activity:

$$\Delta_i = \Delta_{i-1} + \text{SupplyUpdate}_i$$

- **Iterative Gradient Computation:** The solver calculates the *SupplyUpdate* as a function of the historical Delta path to manage momentum and damping:

$$\text{SupplyUpdate}_i = f(\Delta_{i-1}, \Delta_{i-2}, \Delta_{i-3})$$

- **Discrete Kinematics & The Laplacian:** To approximate the physical properties of the field, the solver derives first and second derivatives from discrete snapshots across a three-period window.
  - **Velocity (First Derivative):** Calculated as the trend over the window:

$$d(\Delta_i) = \Delta_{i-1} - \Delta_{i-3}$$

  - **Acceleration (Second Derivative / Discrete Laplacian):** Modeled via a discrete central difference, identifying the curvature of the Delta field:

$$d^2(\Delta_i) = \Delta_{i-1} - 2\Delta_{i-2} + \Delta_{i-3}$$

- **Convergence Equation:** These discrete approximations are fed into the second-order stability equation:

$$a \cdot d^2(\Delta) + b \cdot d(\Delta) + c \cdot \Delta = 0$$

  Where $c \cdot \Delta$ acts as the restoring force pulling the system toward zero, and b provides the **Damping Factor** to ensure a stable, non-oscillatory landing. This ensures that the "SupplyUpdate" is not a jerky response, but a controlled, **intricate** approach to equilibrium.

2.3 Step C: Execution & On-Chain Settlement

- **Exchange Interaction:** Computed IOC orders are routed to a centralized crypto exchange (e.g., Bitget).
- **Inventory Update:** Upon receiving fills, the Vendor submits the updated inventory state to the smart contracts, immediately recalculating the Δ vector.

3 The Castle: Security Architecture & Governance

The protocol utilizes a Diamond-pattern architecture known as the Castle to manage authorization and upgrades via a multi-tiered Access Control List (ACL). This modular approach ensures that the specific duties are delegated.

3.1 Security & Governance

1. **Admin Role:** Sovereign control of the Castle rests with the *Admin Role*, which can be assigned to multiple addresses. Admins hold the power to grant or revoke roles within the system.
2. **The Constable:** Typically, an Admin appoints a *Constable*—a specialized smart contract representing the definitive setup, configuration, and upgrade logic of the Castle. The Constable orchestrates the mapping of roles to functions.
3. **Role-Based Invocation:** Every external function requires a specific security role (e.g., *Keeper Role* for execution facilitation, *Vendor Role* for solver logic). Facets without an explicit ACL entry are private and only invocable internally.
4. **Integrity Safeguards:** Mathematical errors in F_{ij} by a Keeper only shift the scalar CL ($CL > 0$) and cannot distort user-signed weights. Malicious or inefficient Keepers can have their roles revoked by the Admin via the Castle ACL.

3.2 Trading Functions

1. **The Factor:** Facilitates trade execution. The Factor receives orders containing sparse market data from the Keeper and invokes high-dimensional vector math.
2. **The Banker:** Manages the rigorous bookkeeping for Vendor supply and inventory states, ensuring the financial integrity of the vault assets.
3. **The Guildmaster:** Oversees the governance layer, facilitating the creation of new Index Vault contracts and managing the voting processes.

3.3 Computational Functions

1. **The Clerk:** The mathematical engine of the Castle. Both the Factor and the Banker send specific formulas to the Clerk to perform the actual computations.

3.4 Auxiliary Functions

1. **The Worksman:** A specialized smart contract appointed for the deployment of Vault contracts (representing Index ITP tokens). The Worksman focuses exclusively on the construction and initialization of new Vaults.
2. **The Scribe:** A specialist contract dedicated to signature verification. The Scribe ensures that all user intents and vendor pricing payloads meet rigorous cryptographic standards before execution.

4 Key Risk Controls & Compliance

The protocol implements a suite of safety measures designed to protect the system from market volatility, manipulative behavior, and internal errors.

Control Component	Technical Mechanism	Compliance Function
Price Non-Repudiation	Vendor-signed Prices	Prevents Keeper/User price manipulation.
Intent Integrity	User-signed Orders	Ensures execution matches verified intent.
Castle ACL Security	Admin & Constable Roles	RBAC for Keepers, Vendors, and Upgrades.
Deployment Security	Worksman Contract	Isolated and audited Vault construction.
Exposure Ceiling	Max Volley Size (V_{max})	Determines the per-asset Margin Vector **M**.

Table 1: Systematic Hierarchical Control Summary

4.1 Order Execution Limits

The system's safety is rooted in these distinct mathematical boundaries, each serving a specific purpose.

1. **Max Order Size (The Individual Guardrail):** This is a per-user limit defined in USD. If a user's request exceeds this limit, the system realizes the order up to the **Max Order Size** and automatically queues the remainder for subsequent iterations (the "poke" mechanism). This ensures that the protocol maintains **systemic fairness** by preventing single actors from monopolizing liquidity and ensuring a **limited impact on pricing** at the external exchange level

2. **Max Volley Size (V_{max}) (The Aggregate Guardrail):** The V_{max} acts as an upper cap on the total value of the **Delta (Δ)** vector. It represents the absolute maximum volume of **unrealized liquidity** resulting from user orders that are yet to be settled by the Vendor. Importantly, this limit is not an absolute sum of all transaction values, as Sell intents can nullify Buy intents, reducing the net exposure. V_{max} strictly governs the net open position awaiting realization.

3. **Capacity Limit (CL) (The Derived Mathematical Ceiling):** The CL is the most complex of the limits derived from current **Delta (Δ)** vector, configured Margin (**M**) vector, and current Index Capacity (**C**) scalar. It is a dynamically calculated value that scales the actual execution of orders based on the underlying liquidity (L) of each asset in the index. Even if a user's order is within the *Max Order Size*, the CL will scale down the execution if the market cannot support the required volume within limited slippage. It ensures a "smooth landing" by verifying that the smallest liquidity bottleneck determines the speed of the entire index convergence.

5 Conclusion

The framework presented herein replaces heuristic financial execution with a deterministic system of control. By implementing a suite of limit constraints — ranging from individual order caps to the aggregate V_{max} ceiling — the protocol provides an absolute guarantee of risk safety. These guardrails ensure that no single disturbance, whether stochastic or systemic, can compromise the core integrity of the vault.

Central to this stability is the Numerical PDE (Poisson) Solver. By treating the Delta vector as a physical field and applying second-order kinematics, the system transforms execution into a sophisticated process of convergence. The solver's utilization of the discrete Laplacian and iterative damping ensures that the system avoids volatility during high-volume periods, approaching equilibrium with meticulous precision.

This controlled, intricate approach to liquidity management ensures that the protocol remains in a stable state across all execution cycles. By programmatically managing the rate of change and the restoring force within the error field, the system guarantees a compliant, asymptotic, and highly efficient resolution of exposure gaps.