SIMD Tokenomics Blueprint

Non-mintable settlement token for CFD simulations

December 24, 2025

Part 1 - Core Design

1. Goals

SIMD has a fixed supply (1 billion non-mintable token), serves as the settlement token for all simulation jobs (CPU/GPU operators are paid only in SIMD), converts real customer payments into SIMD at authorization time and escrows it for deterministic settlement, and captures protocol value via job settlement + staking distribution (not via buyback/burn assumptions, because we need innovation here).

2. Actors

- **Customer:** Pays for CFD simulations (fiat). Authorizes a budget cap job.
- **Operator:** Registers CPU/GPU resources, posts SIMD bond, performs compute, receives SIMD rewards.
- **SIMD Staker:** Deposits SIMD into a staking vault to earn a share of protocol rewards (paid in SIMD).
- **Protocol/Treasury:** Covers verification, dispute resolution, and operational costs; holds and deploys treasury funds.

3. Canonical routing and liquidity assumption

Primary liquidity for SIMD is the SIMD/SOL pool on PumpSwap. Customers pay in USD (via standard payment rails) which is mapped to USDC. To cover unavoidable swap fees and price impact on USDC→SOL and SOL→SIMD, the protocol charges a **conversion buffer** ρ (e.g., 2–3%) on top of the customer compute budget. The total charged amount is $\text{BudgetGrossUSD} = (1 + \rho)\,\text{BudgetCapUSD}$. The protocol acquires SOL via a canonical USDC→SOL route and finally acquires SIMD by swapping SOL→SIMD on PumpSwap, depositing SIMD into an escrow vault for deterministic settlement.

4. Price model (audit-friendly TWAP)

Let $X(t)$ be the SIMD reserve and $Y(t)$ be the SOL reserve of the PumpSwap SIMD/SOL pool at time t. The spot price (SOL per SIMD) is:

$$P_{\text{SIMD/SOL, spot}}(t) = \frac{Y(t)}{X(t)}. \tag{1}$$

To compute a manipulation-resistant time-weighted average price (TWAP), maintain a cumulative price accumulator:

$$C(t) = \int_0^t P_{\text{SIMD/SOL, spot}}(\tau)\, d\tau, \tag{2}$$

so over a window $[t_0, t_1]$:

$$P_{\text{SIMD/SOL, TWAP}}(t_0, t_1) = \frac{C(t_1) - C(t_0)}{t_1 - t_0}. \tag{3}$$

Implementation: the protocol program stores $(t_{\text{last}}, C_{\text{last}})$ and updates C whenever a new observation is written.

Let $P_{\text{SOL/USD, TWAP}}(t_0, t_1)$ be the SOL/USD TWAP from a robust oracle. Then:

$$P_{\text{SIMD/USD, TWAP}}(t_0, t_1) = P_{\text{SIMD/SOL, TWAP}}(t_0, t_1) \cdot P_{\text{SOL/USD, TWAP}}(t_0, t_1). \tag{4}$$

5. Customer billing ("Pay as You Go")

A customer authorizes **BudgetCapUSD**. The protocol acquires SIMD upfront using the execution rule in Part 2 and deposits the resulting amount S_{target} into a **Customer Credit Vault**. The job is billed purely as a fraction of pre-declared work completed.

Let N_{bill} be the total number of **billable steps** declared at job start (e.g., solver iterations). Let $n(t)$ be the number of billable steps completed by time t, with $0 \le n(t) \le N_{\text{bill}}$. Define progress:

$$p(t) = \frac{n(t)}{N_{\text{bill}}}. \tag{5}$$

Consumed SIMD at time t is:

$$S_{\text{consumed}}(t) = p(t) \cdot S_{\text{target}}. \tag{6}$$

At discrete checkpoints (every Δn steps), the incremental settlement released from the customer vault is:

$$\Delta S = \left(\frac{n_2 - n_1}{N_{\text{bill}}}\right) S_{\text{target}}, \tag{7}$$

where n_1 and n_2 are the completed billable steps at consecutive checkpoints. The job halts if $n(t)$ reaches N_{bill} or if the customer vault is exhausted. Any unused SIMD remains in the customer vault for future jobs.

Part 2 - Distribution, Staking, Bonds, Slashing, and Execution Rules

1. Per-Job SIMD Distribution (Team / Locked Stakers / Compute Bucket + Conversion Buffer)

Let BudgetCapUSD be the customer compute budget and ρ be the conversion buffer (e.g., 0.02–0.03). The total charged amount is:

$$\text{BudgetGrossUSD} = (1+\rho)\,\text{BudgetCapUSD}. \tag{8}$$

Let S_{target} be the SIMD escrow required for the job (computed from the compute budget only). Let S_{recv} be the actual SIMD acquired on-chain from the gross budget after USDC→SOL and SOL→SIMD swaps. Define the **buffer remainder** in SIMD:

$$S_{\text{buffer}} = \max\left(0,\ S_{\text{recv}} - S_{\text{target}}\right). \tag{9}$$

We distribute the job settlement amount $S_{\text{job}} := S_{\text{target}}$ into three buckets:

$$S_{\text{Stakers}} = 0.30\,S_{\text{job}} + S_{\text{buffer}} \tag{10}$$

$$S_{\text{Team}} = 0.40\,S_{\text{job}} \tag{11}$$

$$S_{\text{ComputeBucket}} = 0.30\,S_{\text{job}}. \tag{12}$$

Operators are **not** paid as a percentage of the job. Instead, operator payouts are paid from $S_{\text{ComputeBucket}}$ according to measured CPU/GPU usage and a published rate card. Any unused portion of $S_{\text{ComputeBucket}}$ is routed to **Treasury/Insurance** (reruns, disputes, verification, and operational stability)..

2. Operator Payouts (Per CPU/GPU-hour, Not Percentage)

Let operator i report verified usage in CPU/GPU-seconds, CPU/GPUSeconds_i. Convert to CPU/GPU-hours:

$$H_i = \frac{\text{CPU/GPUSeconds}_i}{3600}. \tag{13}$$

Let $r_{\text{USD/hr}}(g_i)$ be the published USD/hour rate for the operator's CPU/GPU class g_i, and let $\mu \geq 0$ be a protocol premium (e.g., $\mu = 0.20$ for 20% above baseline) to retain operators. Define the operator's USD payout:

$$\text{PayUSD}_i = (1+\mu)\,r_{\text{USD/hr}}(g_i)\,H_i \cdot \alpha_{\text{reliability},i}, \tag{14}$$

where $\alpha_{\text{reliability},i} \in (0,1]$ reduces payout for low reliability and failed runs.

Convert USD payout to SIMD using the authorization-time TWAP (to keep settlement deterministic):

$$\text{PaySIMD}_i = \frac{\text{PayUSD}_i}{P_{\text{SIMD/USD, TWAP}}(t_0, t_0+\Delta)}. \tag{15}$$

The protocol enforces a hard cap that total operator payouts do not exceed the compute bucket:

$$\sum_i \text{PaySIMD}_i \leq S_{\text{ComputeBucket}}. \tag{16}$$

If a job would violate this cap, the job is underpriced and must be quoted at a higher BudgetCapUSD.

3. Locked Staker Rewards (Reward Only Locked SIMD, From Lock Time Until Next Job)

Only stakers who lock SIMD are eligible for rewards. Let job settlement events occur at times t_k. For the interval $(t_{k-1}, t_k]$, define each staker j's locked stake amount s_j and lock start time ℓ_j. Define stake-time weight:

$$W_j^{(k)} = s_j \cdot \max\left(0,\ t_k - \max(t_{k-1}, \ell_j)\right). \tag{17}$$

Let total weight be $W^{(k)} = \sum_j W_j^{(k)}$. Then the staker reward from this job is:

$$\text{Reward}_j^{(k)} = S_{\text{Stakers}}^{(k)} \cdot \frac{W_j^{(k)}}{W^{(k)}}, \tag{18}$$

where $S_{\text{Stakers}}^{(k)}$ is the staker bucket amount allocated from job k. Unstaking uses an unbonding delay so rewards cannot be gamed by instant deposit/withdraw around settlement.

4. Bonds and Lock Durations

Operators must lock at least $B_{\min}$ SIMD in a bond vault to register. Bond withdrawals require an unbonding delay D (e.g., $D = 21$ days) to allow slashing after disputes. Operator payouts may vest linearly over V days (e.g., $V = 30$ days); early unlock incurs a penalty redirected to Treasury/Insurance.

Stakers lock SIMD into a staking vault and face an unbonding delay. The staking vault does not grant the protocol permission to spend staked SIMD; it only tracks eligibility and accounting for rewards.

5. Slashing Rules

- **Liveness:** missed heartbeat windows ⇒ reliability decay and potential slash.
- **Abandonment:** accepted job not completed ⇒ slash and reduced reliability.
- **Invalid result / fraud:** heavy slash + suspension.

Slashed SIMD is routed to Treasury/Insurance to fund reruns, disputes, and stability operations.

6. Swap and Execution Rule (No Ambiguity, Buffer Covers USDC→SOL Slippage)

We enforce one deterministic method: **Exact SIMD escrow at authorization time** with a conversion buffer ρ.

Compute the target SIMD escrow from the compute budget (excluding the buffer):

$$S_{\text{target}} = \left\lceil \frac{\text{BudgetCapUSD}}{P_{\text{SIMD/USD, TWAP}}(t_0, t_0 + \Delta)} \right\rceil. \tag{19}$$

Charge a gross budget that includes the conversion buffer:

$$\text{BudgetGrossUSD} = (1 + \rho)\,\text{BudgetCapUSD}. \tag{20}$$

Execute the routing USD→USDC (off-chain), then USDC→SOL (canonical route), then SOL→SIMD (PumpSwap). The swaps must satisfy a hard guard:

$$S_{\text{recv}} \geq S_{\text{target}}, \tag{21}$$

otherwise the transaction reverts and the quote must be refreshed. The conversion buffer is what makes this guard reliably satisfiable under normal market conditions. Any SIMD acquired beyond S_{target} is:

$$S_{\text{buffer}} = \max(0,\ S_{\text{recv}} - S_{\text{target}}), \tag{22}$$

and is allocated to locked stakers as defined in the distribution schedule.

7. Worked Example: $1000 Job, 7 Hours (with Conversion Buffer)

Assume a job is quoted with a compute budget BudgetCapUSD = 1000 and a conversion buffer $\rho = 0.03$ (3%), so the customer is charged:

$$\text{BudgetGrossUSD} = (1+\rho)\,\text{BudgetCapUSD} = 1.03 \times 1000 = 1030. \tag{23}$$

Assume the authorization-time price is $P_{\text{SIMD/USD, TWAP}} = 0.50$ USD per SIMD. The protocol targets:

$$S_{\text{target}} = \left\lceil \frac{1000}{0.50} \right\rceil = 2000 \text{ SIMD}. \tag{24}$$

Using the gross budget (1030 USD) to route USDC→SOL→SIMD, suppose the protocol actually receives $S_{\text{recv}} = 2050$ SIMD. The buffer remainder is:

$$S_{\text{buffer}} = \max(0,\ S_{\text{recv}} - S_{\text{target}}) = 50 \text{ SIMD}. \tag{25}$$

The settlement amount is $S_{\text{job}} := S_{\text{target}} = 2000$ SIMD and is allocated as:

$$S_{\text{Stakers}} = 0.30\,S_{\text{job}} + S_{\text{buffer}} = 0.30 \times 2000 + 50 = 650 \text{ SIMD} \tag{26}$$

$$S_{\text{Team}} = 0.40\,S_{\text{job}} = 0.40 \times 2000 = 800 \text{ SIMD} \tag{27}$$

$$S_{\text{ComputeBucket}} = 0.30\,S_{\text{job}} = 0.30 \times 2000 = 600 \text{ SIMD}. \tag{28}$$

The simulation runs for 7 hours. Operators are **not** paid as a percentage of the job; they are paid from $S_{\text{ComputeBucket}}$ based on verified CPU/GPU-hours and a published rate card (market rate plus a small premium). Operator payouts (converted into SIMD using the authorization-time TWAP) must satisfy:

$$\sum_i \text{PaySIMD}_i \le S_{\text{ComputeBucket}}. \tag{29}$$

Any unused amount in $S_{\text{ComputeBucket}}$ is routed to Treasury/Insurance, while S_{buffer} is added to locked-staker rewards.