Jump Protocol: A PvE Batch Auction Extraction-Resistant Token Distribution in USDG

Jump Protocol Research
www.jump.family

Technical Note V0.5 — Auction Subsystem | August 2026

Abstract. Jump Protocol positions itself as a *PvE* (player-vs-environment) launchpad: participants compete against a fixed, transparent clearing rule rather than against each other's ordering, gas bids, or capital size. This note specifies the mechanism that enforces that property at the entry point of the token life-cycle. Jump Protocol replaces the free-form order-book style token sales common in memecoin and ICM launches with a single-price, sealed-participation *batch auction* denominated entirely in USDG. Users may deposit ETH or USDG during a fixed 60-minute window; ETH is converted to USDG at the point of deposit so that every downstream computation — the funding target, the per-address cap, the fill ratio, and any refund — is performed in one unit of account. The mechanism guarantees (i) no participant can be allocated more than a fixed percentage of total supply, regardless of capital deployed, (ii) the auction either fully clears or fully refunds, with no partial, ad-hoc "top-up" launches, and (iii) all integer rounding is resolved through a deterministic largest-remainder procedure so that token and currency conservation hold exactly on-chain. This note specifies the funding model, the two-regime waterfilling algorithm used to compute individual fills, the degenerate failure classification (MONEY vs. SEATS), and the Merkle-based settlement and refund path, together with the invariants that a conforming implementation must satisfy.

1 Introduction

Fixed-price and continuous bonding-curve launches share a common failure mode: the clearing price is discovered *after* capital has already been committed, which lets fast actors extract value from slower ones through pure ordering (front-running the first block, sniping the first tick, and so on). A batch auction removes this incentive by collecting all participation over a window, then computing a single allocation and a single implied price *simultaneously* for every participant, using only the aggregate demand observed at the end of the window. No participant's allocation depends on their position in a block or their gas bid — it depends only on how much they deposited relative to everyone else.

Jump Protocol's batch auction is the entry point of the token life-cycle (§2), and it is the only stage at which users commit capital before a liquidity pool exists. Because of this, the design goals are narrower and stricter than a general-purpose AMM: the mechanism must (a) settle in a single unit of account regardless of deposit asset, (b) bound the maximum allocation any one address can receive, (c) treat under-subscription as a hard failure rather than attempt to force a launch, and (d) make every unit of currency and every unit of token individually accounted for, with zero tolerance for negative refunds or over-issuance. Sections 3–6 formalize each of these requirements.

2 Life-cycle Placement

The auction is one state in the project state machine:

State	Trigger	Permitted actions
Created / Upcoming	Project registered	Preview only; no funds accepted
Batch Auction	Fixed-price snapshot taken	60-minute ETH/USDG deposit window
Snapshotting	`reachable` target met	Freeze inputs, compute fills, publish Merkle root
Graduating	Snapshot accepted	Convert USDG proceeds to Stock Token, seed LP
Trading	Canonical LP created	Swap, fee accrual, staking, earnings
Draft / Refundable	MONEY or SEATS failure	Full USDG refund only

Snapshotting, *Graduating*, *Trading*, *Draft*, and *Closed* are irreversible once entered, and every transition must emit an on-chain event so that indexers and front-ends converge to the same view within a bounded confirmation window. A failed auction closes permanently: the protocol does not support retroactive top-ups of an under-subscribed round; the only remedy is a new `roundId`.

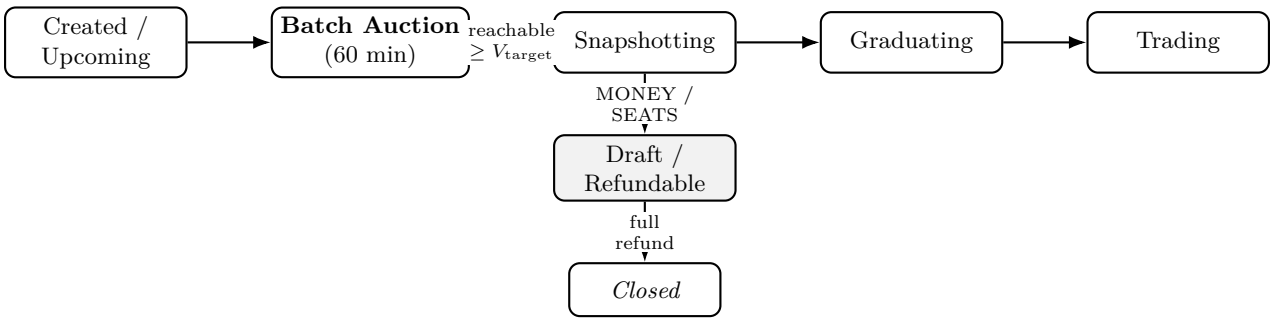

Figure 1: Auction life-cycle: a single clearing test at the close of the 60-minute window routes every round to exactly one of two irreversible paths — graduation into a trading pool, or full USDG refund into *Closed*. No intermediate or partial-success state exists.

3 Funding Parameters

Every project fixes a small set of parameters at creation time, immutable once the auction starts (`AuctionStarted`):

Symbol	Definition	Constraint
$R_{reserve}$	Creator / dev / author reserve, as a share of total supply	Meme $\leq$ 2.9%; APP/ICM $\leq$ 29%
R_{raise}	Share of supply sold through the auction	> 0
R_{pool}	Share of supply seeded into the initial DEX pool	> 0
—	$R_{reserve} + R_{raise} + R_{pool}$	$= 100\%$ exactly
totalSupply	Fixed total token issuance	10^{11} tokens

The three shares are constrained to sum to exactly 100%; this is enforced as an on-chain invariant, not merely a UI-level suggestion, and no combination of $R_{reserve}$, R_{raise}, and R_{pool} that violates it can be finalized at project creation.

Every deposit, regardless of entry asset, is normalized to a single funding denomination, USDG, before any auction arithmetic is performed:

- ETH deposits are routed through a controlled swap into USDG at deposit time; the amount credited to the user is the *actual USDG received*, not a quoted or expected amount.
- USDG deposits are credited directly.
- The funding target and the per-address cap are both denominated in USDG and are therefore invariant to any USDG/Stock-Token price movement that occurs during or after the auction window.

4 Auction Sizing and Clearing Condition

Let FDV_{U18} be the protocol-fixed internal fully-diluted valuation, expressed in USDG with 18-decimal fixed-point precision (currently 6,942 USDG). The two governing quantities are:

$$V_{\text{target}} = \text{FDV}_{U18} \times R_{raise} \tag{1}$$

$$A_{\max} = \text{FDV}_{U18} \times 2\% \tag{2}$$

both normalized to USDG precision. V_{target} is the USDG amount that must be *effectively* raised for the auction to clear; $A_{\max}$ is the maximum USDG that any single address may have counted toward that target, regardless of how much it deposits.

For address i, let z_i denote its cumulative deposit (multiple deposits from the same address are aggregated before any other computation). Define the address's *effective contribution* e_i and *reachable* aggregate demand:

$$0 \;\le\; e_i \;\le\; \min(z_i, A_{\max}) \tag{3}$$

$$\text{reachable} = \sum_i \min(z_i, A_{\max}) \tag{4}$$

The clearing condition is a single inequality evaluated once the 60-minute window closes:

$$\textbf{Clear: } \text{reachable} \ge V_{\text{target}} \quad\Longrightarrow\quad \text{proceed to Waterfilling (§5)}$$

$$\textbf{Fail: } \text{reachable} < V_{\text{target}} \quad\Longrightarrow\quad e_i = 0,\ \text{refund}_i = z_i\ \ \forall i,\ \text{ no LP is created}$$

A failed auction is further classified for user-facing reporting into one of two disjoint causes, both of which resolve to a full USDG refund and no on-chain state beyond that:

Class	Condition	Interpretation
MONEY	Aggregate deposits, uncapped, fall short of V_{target}	Insufficient total capital committed
SEATS	Aggregate deposits would clear if uncapped, but too few *unique addresses* exist beneath the $A_{\max}$ cap to reach reachable $\ge V_{\text{target}}$	Sufficient capital, insufficient distribution

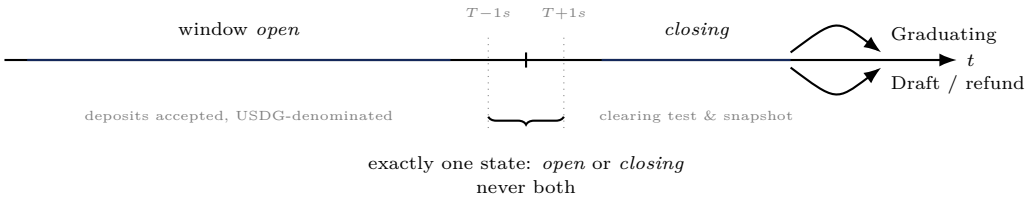

Figure 2: The 60-minute deposit window and the T boundary. The no-dual-state invariant (§7) requires the contract to be unambiguously in *open* or *closing* at every instant around T, including the adjacent $T{-}1s/T{+}1s$ checkpoints tested in acceptance criteria.

The SEATS classification is what makes the 2% per-address cap a genuine distribution constraint rather than a cosmetic one: a small number of large depositors cannot simply overfund their way past it, because only $A_{\max}$ of any address's deposit is ever counted toward `reachable`.

Once tokens are allocated, each address's token amount is a direct linear function of its effective contribution:

$$\text{token}_i = \text{tokenPool} \times \frac{e_i}{V_{\text{target}}} \qquad \text{where tokenPool} = \text{totalSupply} \times R_{raise} \tag{5}$$

Because $e_i \le A_{\max} = \text{FDV}_{U18} \times 2\%$, and by construction $\text{token}_i/\text{tokenPool} = e_i/V_{\text{target}} \le A_{\max}/V_{\text{target}}$, no address can ever receive more than 2% of R_{raise}'s token pool relative to its own cap ratio — and the protocol additionally enforces the stronger, supply-wide invariant that no address's final balance exceeds 2% of *total* supply across all three allocation buckets combined.

5 Waterfilling and Integer Settlement

When the auction clears, e_i must be computed for every participating address such that (1) no address exceeds either its own deposit or the global cap, and (2) the sum of all effective contributions equals V_{target} exactly. This is a constrained proportional-fill problem, solved by *waterfilling*.

5.1 Two regimes

Fast path. If, after applying a single global scaling factor k, no address would be pushed above $A_{\max}$, the fill is a pure linear scale:

$$e_i = z_i \times k_{\text{global}}, \qquad k_{\text{global}} = \frac{V_{\text{target}}}{\sum_i z_i} \tag{6}$$

Iterative path. If one or more addresses would be capped under a single global k, those addresses are pinned at $e_i = A_{\max}$ and removed from both the remaining funding pool and the remaining participant set; k is then recomputed over the reduced problem. This repeats until no further address is pinned, i.e. until convergence. Every intermediate and final value obeys the three-way clamp:

$$e_i = \min\left(z_i \times k,\ A_{\max},\ z_i\right) \tag{7}$$

which simultaneously forbids (a) allocating more than an address deposited, (b) allocating more than the per-address cap, and (c) any negative refund ($\text{refund}_i = z_i - e_i \geq 0$ follows directly from Eq. 7).

5.2 Largest-remainder rounding

Because e_i and token_i are computed over finite-precision integers, direct division introduces truncation error that, summed across potentially thousands of addresses, must not silently violate conservation. Jump Protocol resolves this with a deterministic largest-remainder procedure:

1. Compute every e_i and token_i by flooring (rounding down).
2. Compute the residual — the difference between the target sum and the sum of floored values — in both the USDG and token dimensions independently.
3. Distribute the residual one minimum unit at a time to the addresses with the largest fractional remainder, in descending order of remainder size.
4. Ties in remainder size are broken by ascending address order, giving a total order and therefore a unique, reproducible outcome for any given input set.

This guarantees the two settlement identities hold *exactly*, not approximately:

$$\sum_i e_i = V_{\text{target}}, \qquad \sum_i \text{token}_i = \text{tokenPool} \tag{8}$$

5.3 Worked example

For a health-check case of 5 addresses depositing 200 USDG each and 45 addresses depositing 80 USDG each, the *iterative* path is exercised: since $A_{\max} = \text{FDV}_{U18} \times 2\% = 138.84$ USDG is below the 200 USDG deposit size, the 5 larger addresses are pinned at $e_i = A_{\max} = 138.84$ and removed from the pool, after which k is recomputed over the remaining 45 addresses alone. This yields $\sum_i e_i = 3471$ with each of the 45 smaller depositors receiving $e_i \approx 61.706\overline{6}$ USDG — illustrating both that a subset of addresses can be capped while the rest still resolve by simple proportional scaling, and that the largest-remainder step, not floor truncation, determines the final integer allocation.

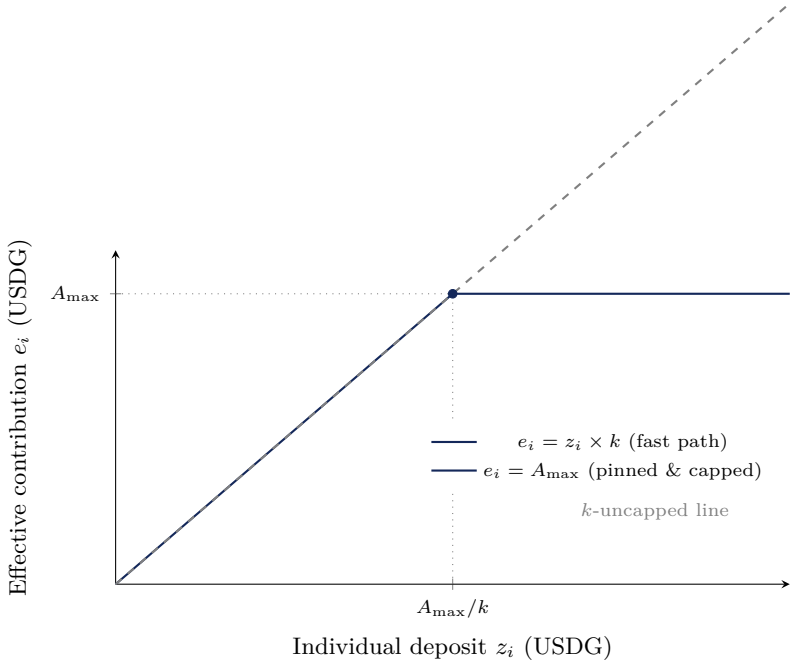

Figure 3: The waterfilling fill function $e_i(z_i)$. Below the cap threshold $A_{\max}/k$, contribution scales linearly with deposit size (Eq. 6); addresses depositing beyond that threshold are pinned at $A_{\max}$ (Eq. 7), and the freed-up allocation is redistributed by recomputing k over the remaining uncapped set — the iterative path of §5.

A boundary case with $N \in \{24, 25, 26\}$ addresses, each depositing far in excess of $A_{\max}$ (so that every address's effective contribution is pinned at the cap, $e_i = A_{\max} = 138.84$), demonstrates the discreteness of the clearing condition itself: $N = 24$ gives reachable $= 24 \times 138.84 < V_{\text{target}}$ and refunds in full; $N = 25$ clears with *zero* slack ($25 \times 138.84 = V_{\text{target}}$ exactly, so every unit of `reachable` is consumed); $N = 26$ clears and the marginal address receives a partial fill via the largest-remainder step, with the remainder refunded — confirming the mechanism has no hidden minimum over-subscription buffer beyond what the arithmetic itself requires.

6 Snapshot, Merkle Settlement, and Claim

At scale, iterating every address's fill and refund in a single on-chain transaction is gas-infeasible and reintroduces exactly the ordering-dependence the auction was designed to avoid. Jump Protocol instead computes the settlement off-chain over the frozen deposit set and commits only a single Merkle root on-chain.

Leaf construction. Each leaf commits to the seven-tuple `(chainId, projectId, roundId, snapshotVersion, account, tokenAmount, refundUsdgAmount)`. Binding `chainId` and `roundId` into the leaf prevents cross-chain or cross-round replay of a proof; binding `snapshotVersion` means that a corrected root is a distinct, non-ambiguous commitment rather than a silent overwrite.

Immutability and correction. Once a root is activated it cannot be silently replaced. If an error is discovered post-publication, the only remedy is to pause the round and publish a new root under an incremented `snapshotVersion` — there is no code path that mutates an already-active root in place.

Claim semantics. Token allocation and USDG refund may be claimed together or separately; the protocol treats them as independent leaves of the same commitment; a user who has not yet claimed one is never blocked from claiming the other, and no claim of one can invalidate the other. All claims are *pull*-based: the contract exposes a claim function that the user calls with a Merkle proof, guarded by checks-effects-interactions ordering and re-entrancy protection. The protocol does not push refunds or allocations to users in bulk, which both bounds gas cost per transaction and removes any scenario in which a failed push to one malicious or non-standard address blocks settlement for everyone else.

7 Invariants and Acceptance Criteria

A conforming implementation must satisfy the following properties, all of which are treated as release-blocking:

Invariant	Statement
No dual state	At the boundary of the auction window ($T-1s$, T, $T+1s$) the contract must be in exactly one of {open, closing} — never both.
Cap enforcement	$e_i \leq A_{\max}$ for every address, for every round, with no implementation path that allocates above the cap even transiently.
Fill conservation	$\sum_i e_i = V_{\text{target}}$ and $\sum_i \text{token}_i = \text{tokenPool}$ exactly, post largest-remainder correction.
Non-negative refund	$\text{refund}_i = z_i - e_i \geq 0$ for every address, always.
Entry-conversion integrity	A failed ETH→USDG conversion at deposit time must not be credited to the user's z_i; the deposit simply does not enter the auction.
Proof integrity	A tampered proof, a proof against a stale root, a replayed claim, or a re-entrant claim call must all fail deterministically.
Failure atomicity	MONEY and SEATS failures both resolve to $e_i = 0$ for all i with no LP creation — there is no partial-success state between "cleared" and "fully refunded."

Regression testing additionally requires a randomized fuzz suite — on the order of 10,000+ runs over 1 to 120 synthetic addresses with randomized deposit sizes — in which the counted incidence of negative refunds, over-cap allocations, over-principal allocations, and any conservation violation must each be exactly zero.

8 Discussion

The batch auction's central design bet is that collapsing all participation into a single USDG-denominated clearing computation removes the two dominant sources of unfairness in on-chain token launches: intra-block ordering advantage and denomination risk from volatile entry assets. The per-address cap converts this into a genuine distribution mechanism rather than a price mechanism — the auction does not attempt to discover a market price at all; it fixes FDV_{U18} exogenously and instead solves purely for a fair *allocation* of a fixed token pool against demand, deferring all subsequent price discovery to the DEX pool created once the round graduates (§2). This is a deliberate scope reduction relative to continuous or dutch-auction-style bonding mechanisms: the auction subsystem is optimized for distributional fairness and settlement correctness under a fixed valuation, not for price discovery, which is left entirely to the trading phase that follows it.

This is also the concrete sense in which the auction is *PvE* rather than *PvP*: a participant's outcome is a deterministic function of how much they deposited and how many other addresses deposited, evaluated once, after the window closes — never a function of who else was in the same block, who paid more gas, or who reacted fastest to on-chain information. The environment (the fixed clearing rule) is the only counterparty; there is no mechanism by which one participant's outcome can be improved at another's direct expense.

This note describes protocol mechanism design and is not investment, financial, or legal advice. Parameters (FDV, cap ratios, reserve bounds) reflect the V0.5 specification and are subject to governance and audit review prior to mainnet activation.