

---

# A fair opening. A reason to stay.

---

## What we built

Musebook now has a locally tested launch system that connects a single-price opening, a bonding curve, conditional creator rewards, time-weighted holder rewards and migration into real decentralized-exchange programs. It also connects registered Metaplex agents to the launch workspace, with ownership checks and a separate, existing Genesis execution path.

The thesis is straightforward: **accepted opening orders share one price, creator rewards remain claimable only while enough of the original allocation stays locked, and half the curve fees are allocated to holders.** Each part has a concrete mechanism and a defined limit.

This is an implementation report dated October 11, 2026. The custom launch and rewards programs remain **undeployed**. The tests ran on this computer using isolated local runtimes, ephemeral accounts and pinned copies of actual Raydium and Meteora mainnet program bytecode. No public-chain transaction, new live agent mint or funded user-wallet action was performed by this integration. Publishing this article does not deploy those programs.

That distinction matters. We have executable evidence of the local lifecycle, rather than a claim that a production launch has happened. The [verification report](#) records the scope, checks, binary hashes and remaining limitations.

---

## Why launch fairness needs a lifecycle

A launch has several different moments when participants can gain an advantage. Arrival order can affect the opening price. Multiple accounts can dominate early purchases. A creator can earn fees while reducing their commitment. A holder can move balances just before a reward snapshot. Graduation can introduce a price discontinuity or leave liquidity under someone else's control.

A bonding curve answers how price changes with reserves. It does not, by itself, settle those other questions. Our contribution is to make the transitions explicit: opening deposits must settle before trading starts; trading fees must remain backed before migration; creator custody must remain measurable before rewards can be claimed; and identity must be checked before an agent association is prepared.

This makes the design interesting as a whole. Familiar primitives—commitments, constant-product arithmetic, token locks, Merkle proofs and program-derived addresses—enforce one set of launch terms across several stages. We are not claiming to have invented those primitives or established a first-ever

launch mechanism.

---

## An opening where speed cannot buy a cheaper fill

The hardened Musebook opening uses commit and reveal. A participant deposits quote tokens and commits to an order. During the reveal window, they disclose the order and its salt. Valid revealed orders settle at one configured rational price. If demand exceeds the opening allocation, fills scale proportionally; unused deposits are refunded. Unrevealed deposits also have a refund path.

Consider an opening with 200 tokens available. Twelve addresses each request 30 tokens at the same price. The total demand is 360, so each receives an integer-rounded proportional allocation and a refund for the unused deposit. The first accepted order and the last accepted order face the same price. Ordering a bundle ahead of everyone else cannot produce a cheaper accepted opening fill.

The program allows one opening order per address and caps an address at 3% of total supply. The creator cannot participate in this hardened opening. Once every order is settled, a guard period of at least 75 slots begins. Opening purchases count toward the guard's cumulative address cap; the guard also limits buys per wallet per slot and total buys per slot. The creator cannot buy during it.

Those are address-level rules. A person can control several wallets. Commitments hide the reveal payload, while deposits remain visible. The opening does not eliminate Sybil participation, censorship or timing advantages after continuous trading begins. The precise claim is equal accepted opening prices, with explicit limits on early address-level concentration.

The distinction from the supplied reference implementation is documented: its batch opens at a curve-cleared price and permits creator-purchased fills. The canonical Musebook program uses a fixed-price opening and excludes the creator. Reusing the core did not mean inheriting every reference policy.

---

## A 1% curve fee with three beneficiaries

During curve trading, the fee is 1% of gross quote volume. The split is 0.2% to the protocol, 0.3% to the creator and 0.5% to holders. Equivalently, holders receive half the fee, the creator receives 30% and the protocol receives 20%. Integer rounding dust goes to holders.

| On 10,000 quote units of curve volume | Amount |
|---------------------------------------|--------|
| Total curve fee                       | 100    |
| Holder share                          | 50     |
| Conditional creator share             | 30     |
| Protocol share                        | 20     |

The holder allocation and creator allocation coexist. Holder rewards do not depend on removing the creator allocation from the fee schedule. If there is no eligible creator allocation, or the creator breaches the lock threshold, the creator share redirects to holders.

These are curve fees. Post-graduation fees depend on the venue. Burning Raydium LP tokens gives up the LP's continuing liquidity and fee rights. Raydium's separate [Burn & Earn flow](#) locks LP tokens or a supported position rather than destroying them, and retains fee claims through a transferable Fee Key NFT; this prototype does not implement that flow. The Meteora path permanently locks a position while retaining the supported position-fee claim path. Meteora's own protocol portion applies before our collected position earnings are split.

The current quote-approval path accepts initialized legacy SPL mints without freeze authority. Token-2022 and tokenized-stock quotes are excluded from this prototype.

---

## Creator rewards depend on staying committed

The draft defaults to a disclosed 5% creator allocation. That allocation is an explicit grant, not a creator purchase in the opening. It remains locked for at least 30 days. Creator reward tranches vest linearly over exactly 180 days—six 30-day months.

Vesting alone is not the retention condition. To remain eligible, the creator must keep at least half of the original allocation locked. The program measures authenticated creator custody, rather than treating the entire launch vault balance as creator holdings. Tokens reserved for the curve cannot disguise a creator's withdrawal.

After the minimum lock expires, a partial release can leave the creator eligible. Once the original-allocation threshold is breached, eligibility is permanently lost. Unpaid rewards stop, including amounts that have vested but have not yet been claimed. Anyone can redirect the remaining unpaid tranche to the holder treasury once. The creator cannot restore eligibility by returning tokens. Previously paid rewards are not recovered.

The local lifecycle test funded a creator tranche with 3,000,000 quote-token atoms. After approximately 31 days, the creator claimed 516,641 atoms. They then withdrew and actually sold 80% of their original locked allocation. A permissionless caller redirected the remaining 2,483,359 atoms to the holder treasury. Further creator claims and a second forfeiture attempt failed. A later authorized, funded holder epoch is still required to allocate that treasury to individual holders.

Separate boundary checks executed from the test state established that one second before the 180-day vesting boundary, one atom remained unpaid; at the boundary, the remaining amount became claimable. A fully vested but unclaimed tranche still forfeited when the creator breached the retention threshold. The [implementation guide](#) describes the exact conditions and test paths.

This is the economic point of the creator lock: continued earnings depend on a continued stake. It does not guarantee the creator will build, communicate honestly or maintain a token's value.

---

## Holder rewards recognize time, not a last-minute balance

The supplied Rust core now powers the canonical holder-accounting replay. New holdings pass through a 24-hour warm-up. Eligible balances accumulate reward weight with a loyalty multiplier that rises from 1x to 2x over 90 days, measured from the first purchase and including the warm-up day. A top-up restarts the warm-up of all still-pending balance; already eligible balance continues earning.

The accounting integrates token balance over time using exact integer arithmetic and carries fractional remainders. Repeated checkpoints cannot manufacture additional reward weight. A balance reduction removes pending holdings first; a reduction of eligible holdings forfeits proportional accumulated points and resets the remaining eligible balance's loyalty age.

This makes a sustained holding history economically different from a balance acquired immediately before a snapshot. It also makes the behavior replayable: a complete, ordered stream of aggregate balances produces an allocation that can be independently recalculated.

The current distribution has an important trust boundary. An off-chain indexer replays events and an authorized publisher funds an immutable Merkle epoch. Claims bind the owner, program, epoch, quote mint and amount, and funded allocations vest over 180 days. Destination checks and single claim receipts prevent redirection and replay.

The root publisher remains trusted to supply complete events and a correct allocation. The prototype does not intercept every transfer of arbitrary SPL tokens on-chain. Once a holder allocation has been published, it keeps vesting even if that holder later sells; later sales affect future points. This differs from the creator rule, which forfeits all unpaid creator rewards on a threshold breach. The [holder-accounting guide](#) explains that distinction in detail.

---

## Graduation preserves the final curve price

When the curve reaches its configured graduation threshold, any caller can trigger the supported migration. The caller does not become the pool owner or receive its LP tokens. The program checks the exact target accounts, reserves and migration state, and rejects a repeat attempt.

Matching the pool price requires more than depositing everything in a vault. The curve's final spot price includes virtual reserves, while a real pool can only receive real tokens. Opening escrow, creator allocations and unpaid fee liabilities must also remain excluded from the liquidity budget.

For effective quote reserve  $q_e$ , remaining curve base reserve  $B$  and real quote available for liquidity  $q_r$ , the final curve price is  $q_e / B$ . The base deposit is  $\text{floor}(q_r \times B / q_e)$ . That puts the pool's opening reserve ratio at the curve's final spot price within the integer rounding bound. Unused base tokens are burned.

The local test checked actual pool vault balances, not just a predicted transaction. It measured a cross-product remainder of 7,644,187,720 against an effective quote reserve of 8,995,500,008. That is a discrepancy of less than one base-token atom under the deposit formula, rather than a loose floating-point

percentage tolerance. Trading still has price impact, venue fees and slippage; the opening reserve ratio is not a guarantee that every subsequent swap executes at that price.

---

## Real Raydium behavior, including an adversarial pool

The Raydium test loaded pinned mainnet program bytecode into the local runtime. An attacker created the publicly derivable canonical pool first at an unsuitable price. They also prefunded the launch's intended exclusive pool address with lamports.

Migration still succeeded because the legitimate pool uses a distinct launch-derived signer that outsiders cannot provide. The attacker-created pool could not be substituted for it. A noncreator triggered the move, the actual Raydium program created the intended pool, all issued LP tokens were burned and LP mint supply reached zero. A swap succeeded in the pool's permitted opening period. A second migration failed.

Raydium's initialized pool has an opening time. The local test honored that time before executing its swap.

The Meteora test separately executed actual DAMM v2 bytecode, created a pool, permanently locked the position, swapped and collected real position fees for the configured beneficiaries. An existing customizable Meteora pool can block this migration route. The program rejects that state rather than adopting an untrusted pool. The two venues have different control and liveness properties; we document both.

These tests establish behavior against the pinned binaries in local runtimes. They do not establish deployment, compatibility with every future upgrade, adequate production liquidity or an external security audit. Raydium's [initialization source](#) and Meteora's [DAMM v2 source](#) provide the upstream context.

---

## Agents gain a verifiable connection to the launch

An agent name and a ticker are weak identifiers. Musebook's new fairlaunch inspector reads the registered Metaplex Core asset, current owner and derived agent Identity account. It validates account ownership, supported layout, identity linkage and the confirmed read snapshot before preparing an association.

Public inspection works without a wallet. Exporting an owner association requires a connected wallet matching the current owner and a fresh verification. Preparation stops if the asset, wallet or network changes during the read. A Core asset, agent Identity PDA, Asset Signer PDA, curve program and fungible mint remain distinct identities. The downloaded JSON is an editable association record, not a signed authorization or a transaction receipt.

The interface offers two paths. The custom Musebook fairlaunch path exports a draft and keeps custom signing disabled while its programs remain undeployed. The existing native Genesis path opens the agent-associated launch workspace, which retains transaction review, simulation, wallet signatures and recovery receipts.

Metaplex's permanent canonical agent-token binding has specific requirements. Its [identity-program processor](#) validates a qualifying mint-funded Genesis account and the agent's Asset Signer authority, and permits the binding only once. A generic custom SPL mint or an exported association does not satisfy those requirements. Service metadata also remains untrusted; discovering a service does not delegate an agent wallet or authorize payment. The [agent integration guide](#) maps the supplied Metaplex guides to the implemented controls and the [official token documentation](#).

The opportunity is to put persistent agent identity beside explicit launch economics. An owner can present an inspectable connection between an agent and a token project without turning that connection into automatic spending authority. A token association does not promise equity, ownership of the agent's output or future service revenue.

---

## Reverse engineering became a shared evidence trail

We used the supplied REA checkout to inspect browser behavior, export JavaScript and record source identities. REA supported controlled captures and static analysis of selected web scripts. It did not analyze Solana SBF bytecode. Program behavior was tested separately through compiled programs and Solana runtimes.

The integration reused the supplied pure Rust core while retaining the canonical programs' custody and administration checks. The launch calls its curve and guard code, the rewards program calls its vesting code, and the host replay calls its holder accounting. It added exact-remainder holder arithmetic, connected the Rust event replay to proof generation, aligned vesting at 180 days and connected the supplied detector CLI to the same engine used by the site API.

That shared detector decodes the actual Pump instruction roles, uses the real mint supply and reports evidence coverage. Missing genesis history, truncated samples, opaque transactions or failed providers produce an unknown or partial result, never a fabricated clean score. A common funder is an observation, not proof that wallets have one owner. The detector is a research tool; it does not enforce the opening's on-chain rules.

The [integration ledger](#) records changes, tests, provenance and limitations. The [source manifest](#) identifies the baseline used for the implementation evidence. That baseline is preserved separately from this article's publication checksums.

---

## What the tests establish

The recorded integration verification passed 33 local-validator checks across 35 successful local transactions, two additional LiteSVM lifecycle tests and 563 frontend regression tests. Desktop and mobile browser checks verified the draft export, agent gates, native Genesis navigation and unknown-state handling for an unavailable risk provider. The browser checks made zero financial broadcasts.

Those numbers describe the integration baseline before this article was added. Test categories overlap, so they should not be summed into an invented grand total. The publication's [article manifest](#) separately records content and downloadable-document hashes.

The meaningful evidence is the set of transitions: refunds survive pause; trading waits for settlement; creator custody cannot be spoofed by a program-owned keypair; rewards remain funded; graduation preserves fee liabilities; the real pool has the expected opening ratio; a precreated canonical Raydium pool cannot hijack migration; a real creator sale ends unpaid claims.

Production still requires independently reviewed programs, deployment and upgrade-authority decisions, a deployed and monitored event-indexing/root-publication process, venue configuration, a reviewed custom wallet client and operational checks on the intended network. These are remaining engineering steps, not capabilities implied by a passing local test.

---

## A fair comparison with Pump.fun

Pump's published SOL/USDC bonding-curve schedule is 1.25%, including a 0.30% creator component. Musebook's local design uses 1.00% with a 0.30% conditional creator share and a separate 0.50% holder share. That is 0.25 percentage points less in total curve fees, or 20% less relative to 1.25%. Venue and pair economics after graduation must be compared separately. See the [official fee schedule](#).

The earlier shorthand that Pump gives holders nothing is inaccurate. Pump publishes an opt-in [holder-reward mode](#) that redirects creator fees to holders. Pump also supports [custom quote pairs](#). Neither holder rewards in general nor choosing a custom quote asset is a defensible uniqueness claim for this project.

Our specific design combines simultaneous creator and holder fee shares, a fixed-price proportional opening, conditional creator vesting, time-weighted holder accounting and a tested migration boundary. We can explain and test those mechanisms. We cannot infer the absence of every competitor protection from public documentation, claim universal bundle resistance or describe this prototype as rug-proof.

---

## Where the innovation is

The strongest innovation is the connection between rules that are often discussed separately. Opening fairness is an allocation rule. Creator alignment is a custody-dependent earning rule. Holder participation is an accounting and distribution rule. Graduation is a reserve and authority transition. Agent association is an identity rule. Musebook makes their boundaries explicit and tests how they interact.

That creates a more useful launch proposition than a promise of safety: participants can inspect how opening allocations work, how fees are divided, what happens when a creator leaves, what a holder must do to earn weight and who can control liquidity after graduation. The local prototype turns those terms into executable behavior and preserves the evidence needed to challenge the claims.

**A fair opening. A reason to stay. An identity you can inspect.** That is the direction we have implemented locally—and the standard the next production steps must preserve.

Explore the [fairlaunch workspace](#), read the [developer documentation](#), or download this article as [Markdown](#) or [PDF](#).