Limelight Whitepaper

A trust-first Solana launchpad: permanently-locked liquidity, fair openings, and holder-aligned fees

Version 1.1 | July 2026 | limelight.fun

Program: `342po67kbxoKXcjpfiQQcH47pd6WnX37Q2YrWaEkxfGq` (Solana devnet)

Abstract

LIMELIGHT is a token launchpad implemented as the Solana program launchpad_core (id 342po67kbxoKXcjpfiQQcH47pd6WnX37Q2YrWaEkxfGq, currently deployed on devnet). It addresses two recurring failure modes of public token launches — post-migration liquidity withdrawal (the "LP pull") and unfair openings — through mechanisms enforced entirely on-chain rather than by convention or trust. Tokens are priced on a constant-product bonding curve and may open via an optional blind commit auction that settles as a single batch buy into the curve at one shared price, eliminating the price gap that snipers exploit at launch; committers may withdraw their committed SOL at any time before the auction settles, so no capital is ever locked in against the committer's will. When a launch's real reserves reach a graduation threshold (default 85 SOL), the curve's liquidity migrates atomically into a Meteora DAMM v2 concentrated-liquidity pool and 100% of the resulting liquidity position is permanently locked by the AMM's own bytecode, making withdrawal structurally impossible. A 1% configurable trading fee is split among the platform, the creator, and holders (default 20/50/30), with holder rewards distributed through a MasterChef-style accumulator keyed to a program-tracked shadow balance. Creator token allocations are paid for off the live curve rather than granted for free, and vest linearly over min(bps/100, 12) months from graduation. Vesting launches may set an optional refund deadline that fails the launch and enables pro-rata SOL refunds if graduation is not reached. An on-chain referral program lets any wallet earn a share of a trade's platform-fee leg on the buys its link drives, carved from the platform's own share so the trader pays nothing extra; a creator may additionally opt to share a bounded part of their own creator fee to reward promoters more. This document specifies each mechanism, its exact formulas and constants, and an explicit statement of what the protocol structurally guarantees versus what it does not.

1 Introduction

Permissionless token launches on Solana have converged on the bonding-curve model, in which a token is priced by an on-chain reserve formula and trades until it accumulates enough liquidity to migrate to a decentralized exchange. This model lowers the barrier to launching a token, but two failure modes recur across the ecosystem and impose losses on participants.

The first is post-migration liquidity withdrawal — the "LP pull." After a launch migrates to an AMM, the party controlling the liquidity-provider (LP) position can withdraw the pooled reserves, collapsing the price and stranding token holders. Mitigations that rely on off-chain promises, multisig custody, or time-locks that eventually expire do not structurally prevent withdrawal; they only delay or socially discourage it.

The second is the unfair opening. When a curve opens at a price known in advance, the first transactions in the same slot can be front-run or sniped: an informed actor buys at the opening price and sells into later

buyers, capturing value that would otherwise accrue to ordinary participants. A gap between an initial seed price and the first market-clearing price creates a deterministic arbitrage window.

LIMELIGHT targets both failure modes with mechanisms enforced by program logic and by the AMM's own bytecode, not by trust in the creator or platform. This document specifies those mechanisms using the exact formulas and constants present in the deployed program, and states plainly the boundaries of what is and is not guaranteed.

2 Design goals

The protocol is designed around a small set of goals, each mapped to a concrete on-chain enforcement mechanism rather than to a policy or a promise.

- Structural liquidity lock: after graduation, 100% of the LP position is permanently locked by the AMM's own remove-liquidity guards; no key, authority, or admin path can withdraw it.
- Gap-free fair opening: the optional blind commit auction settles as a single batch buy into the standard curve, so the first Active-status trade continues from exactly where that buy left the price — every committer transacts at the same average price and the first buyer faces no gap.
- Aligned creator incentives: creator token allocations are purchased off the live curve at a ceil-rounded price (never underpaid) and vest post-graduation, rather than being minted free and dumped at open.
- Holder participation: a configurable share of every trading fee accrues to token holders through a program-tracked accumulator that cannot be inflated by raw token transfers.
- Principal solvency: refund, holder-reward, and fee ledgers coexist in one vault but are accounted separately, so no path can pay one bucket out of another's principal.
- Determinism and permissionlessness: the batch-buy settle, curve seed, migration price, and refunds are computed with no oracle and no discretion; liveness backstops (deadline failure, stuck-graduation recovery) are callable by anyone.

3 Protocol overview and lifecycle

A launch is created by calling initialize_launch. Before creation the token's mint authority and freeze authority must both be revoked (set to None); the program rejects any launch whose mint retains those authorities, so no new supply can be minted and no account can be frozen after launch. Creation costs a flat fee of LAUNCH_FEE_LAMPORTS = 50,000,000 lamports (0.05 SOL) paid to the platform fee wallet.

Each launch proceeds through a small set of statuses. If a commit window is configured (commit_window_secs within [60, 86,400] seconds), the launch begins in Auction status; committers lock SOL blindly during the window, then settle_auction executes the committed pot as one batch buy into the curve and transitions the launch to Active. If commit_window_secs is 0, the launch is an Instant launch: it bypasses the auction entirely and is set to Active at initialization using a bootstrap curve seed.

In Active status the token trades on the constant-product bonding curve via buy and sell. No trading is permitted until launch.active_trading_starts_slot is reached, which is set to the settlement slot plus settle_cooldown_slots to blunt settle-and-buy sandwich attacks. On a buy, when real_sol_reserves reaches the graduation threshold, graduation fires: liquidity migrates to a DAMM v2 pool and the launch enters a Graduated terminal state.

Two terminal off-ramps exist. A vesting launch may carry an optional deadline; if graduation is not reached in time, a permissionless call moves the launch to Failed, after which only pro-rata refunds and creator-stake slashing remain enabled. Separately, a permissionless recovery instruction can move a launch stuck in Graduating (due to a pre-created pool griefing attack) to Failed so that refunds become available.

4 Bonding curve

Pricing uses a constant-product invariant over virtual reserves. The product k = virtual_sol_reserves * virtual_token_reserves is held constant across each trade, and the marginal price is defined as price = virtual_sol_reserves / virtual_token_reserves. Real reserves (real_sol_reserves, real_token_reserves) track the physical capital and token inventory in the curve vaults and back the virtual quotes.

On a buy, the fee is deducted pre-curve so that only the net input moves the reserves. The trader sends sol_in; the program takes fee_lamports and advances the SOL reserve by the remainder, then derives the new token reserve by division (flooring to an integer), and mints the difference to the trader.

On a sell, the fee is deducted post-curve from gross proceeds. The token reserve grows by the tokens returned, the new SOL reserve is derived by division, the gross SOL out is the reserve delta, and the fee is subtracted from that gross to yield the net paid to the seller. Fees never leave the curve vault during a trade; they accrue into pending fee and holder-reward buckets for later claim.

The fee floor guarantees that a buy-sell round trip always costs at least two lamports combined, closing a dust-drain vector where zero-fee micro-trades could otherwise siphon reserves. Trades are gated on active_trading_starts_slot, and the graduation check fires only on a buy that pushes real_sol_reserves to the threshold.

```
k = virtual_sol_reserves * virtual_token_reserves  (constant across a trade)
```

```
price = virtual_sol_reserves / virtual_token_reserves
```

```
fee_lamports = max(floor(amount * fee_bps / 10000), min_fee_lamports)
```

```
BUY: sol_in_net = sol_in - fee_lamports
```

```
BUY: new_virtual_sol = virtual_sol_reserves + sol_in_net
```

```
BUY: new_virtual_token = floor(k / new_virtual_sol)
```

```
BUY: tokens_out = virtual_token_reserves - new_virtual_token
```

```
SELL: new_virtual_token = virtual_token_reserves + token_in
```

```
SELL: new_virtual_sol = floor(k / new_virtual_token)
```

```
SELL: sol_out_gross = virtual_sol_reserves - new_virtual_sol
```

```
SELL: sol_out_net = sol_out_gross - max(floor(sol_out_gross * fee_bps / 10000), min_fee_lamports)
```

```
fee_bps = 100 (1%), min_fee_lamports = 1, BPS_DENOMINATOR = 10000
```

5 Fair opening: blind commit auction

A launch may open through a blind commit auction rather than trading directly on the seed price. During the commit window (60 to 86,400 seconds), wallets call commit_auction to lock SOL into the auction vault. Commits are blind: no price or running total is shown, and the tokens each commit will buy are not determined until settlement, so no arbitrage is possible during the window. No single wallet may commit more than max_single_committer_bps_of_total of the running total (the founding commit is exempt).

A committer is never locked in against their will. While the window is open (status Auction, before commit_deadline), withdraw_commit lets a committer pull back part or all of their committed SOL: it transfers the requested amount from the auction vault back to the committer, decrements both the committer's record and total_committed by that amount, and — on a full withdrawal — decrements the unique-committer count and resets the record so a later commit is counted as new again. Because the auction is blind and settles at one uniform price, the ability to withdraw cannot be used to game that price: an early commit is worth exactly as much as a late one, so entering and exiting carries no informational advantage.

At settle_auction the entire committed pot is executed as a single, fee-free buy into the standard bonding curve. The curve is first seeded at the ordinary bootstrap price — default virtual reserves, real_sol_reserves = 0, real token reserves = total_supply - creator_allocation_amount — exactly as an Instant launch opens. The whole of total_committed is then applied as one constant-product buy, and the tokens that buy yields, tokens_out, are the auction tranche committers split pro-rata. There is no separate clearing-price computation, no size or breadth gate, and no under-fill outcome: because it is one buy at one instant, every committer transacts at the same average price, total_committed / tokens_out, no matter when they committed.

The curve's opening health comes from the bootstrap seed, not from how much was committed, so a small auction is simply a small first buy on an otherwise-standard curve. After the buy, the committed SOL is swept from the auction vault into the curve vault as 1:1 real backing (real_sol_reserves = total_committed), and token custody across the two vaults is reconciled so the auction vault holds exactly tokens_out for committer claims and the curve vault holds the rest of the sellable supply. Because trading simply continues from the price the batch buy set, there is no gap and no sniper window between the last committer and the first buyer.

The launch then transitions to Active with active_trading_starts_slot = current_slot + settle_cooldown_slots. Committers call claim_auction_tokens exactly once to receive floor(commit_i * tokens_out / total_committed) tokens; the floor property guarantees the summed allocations never exceed the tranche. If total_committed is 0 the curve simply opens at the bootstrap price with an empty first buy (tokens_out = 0). Because every auction now settles into the curve and distributes tokens, there is no under-fill refund: claim_auction_refund is rejected once an auction has cleared and remains only for launches settled under the prior model. Instant launches (commit_window_secs = 0) skip this subsystem entirely.

```
batch buy at settle (fee-free), when total_committed > 0:
seed_virtual_sol   = default_virtual_sol_reserves
```

```
seed_virtual_token = total_supply + token_virtual_offset
seed_real_token    = total_supply - creator_allocation_amount
new_virtual_sol    = seed_virtual_sol + total_committed
new_virtual_token  = floor(seed_virtual_sol * seed_virtual_token / new_virtual_sol)
tokens_out         = seed_virtual_token - new_virtual_token
```

```
post-settle reserves:
virtual_sol_reserves   = new_virtual_sol
virtual_token_reserves = new_virtual_token
real_sol_reserves      = total_committed        (swept auction_vault -> curve_vault)
real_token_reserves    = seed_real_token - tokens_out
auction_token_supply   = tokens_out
```

```
per_committer_allocation  = floor(commit_i * tokens_out / total_committed)
effective_price_per_token = total_committed / tokens_out   (identical for every committer)
```

```
if total_committed = 0: curve opens at the bootstrap seed, tokens_out = 0
```

```
MIN_COMMIT_WINDOW_SECS = 60, MAX_COMMIT_WINDOW_SECS = 86,400
```

```
bootstrap fallback: virtual_sol_reserves = default_virtual_sol_reserves, real_sol_reserves = 0
```

6 Graduation and permanent liquidity lock

Graduation fires on a buy that brings real_sol_reserves to the graduation threshold, which defaults to 85 SOL (85,000,000,000 lamports) and is read from platform configuration rather than hardcoded. At migration the SOL leg is reduced by the configurable migration fee (migration_sol = real_sol_reserves - migration_fee_lamports; typically 2 SOL) while the token leg is unchanged. Before any transfer, both reserve figures are cross-checked against live vault balances to prevent accounting drift.

Liquidity migrates into a Meteora DAMM v2 (cp-amm) full-range pool. The pool address is a canonical PDA derived from the customizable-pool prefix and the ordered mint pair under the cp-amm program ID, so it cannot be spoofed. The initial sqrt-price is computed by a Q64.64 quadratic solver from the migration amounts, and the initial liquidity is min(liquidity_from_base, liquidity_from_quote); both are computed deterministically so migration introduces no price slippage. Pool creation, seeding, and the permanent lock execute as one atomic CPI sequence — either all succeed or the entire transaction reverts, so no partial migration state can exist.

The lock is the core liquidity-lock guarantee. After seeding, permanent_lock_position moves the entire position's liquidity into cp-amm's permanent_locked_liquidity. cp-amm's own bytecode enforces permanent_locked_liquidity as a floor that remove_liquidity, remove_all_liquidity, and close_position all refuse to breach — regardless of who holds the position NFT. The migration authority is a program PDA (seeds [b'damm_migration_authority', launch.key(), bump]) that is never exposed as a private key and is the sole signer for pool creation and lock. A sub-10-unit rounding dust may remain in the authority ATAs and is economically negligible and not swept.

On successful migration the curve token vault is closed and its rent reclaimed to the platform fee wallet (the rent sink is checked to equal config.platform_fee_wallet, so a permissionless caller cannot redirect it), any unsettled creator stake is returned, and graduation_reward_eligible_supply is frozen to the current

total_reward_eligible_supply for post-graduation fee distribution. As a liveness backstop against a griefer pre-creating the canonical pool, the permissionless recover_stuck_graduation instruction detects an occupied pool on a Graduating launch and sets it to Failed to enable refunds; it is a no-op on healthy launches and the only path that can make this transition.

```
graduation_threshold_lamports = 85,000,000,000 (85 SOL, from PlatformConfig)
```

```
migration_sol = real_sol_reserves - migration_fee_lamports
```

```
migration_tokens = real_token_reserves
```

```
sqrt_price = solve_quadratic(migration_tokens, migration_sol, MIN_SQRT_PRICE, MAX_SQRT_PRICE)-
[Q64.64]
```

```
liquidity = min(liquidity_from_base, liquidity_from_quote)
```

```
MIN_SQRT_PRICE = 4,295,048,016
```

```
MAX_SQRT_PRICE = 79,226,673,521,066,979,257,578,248,091
```

```
migration authority PDA seeds = [b'damm_migration_authority', launch.key(), bump]
```

```
graduation_reward_eligible_supply = total_reward_eligible_supply  (frozen at graduation)
```

```
constraint: migration_sol >= 1 and migration_tokens >= 1, else ZeroMigrationAmount
```

```
constraint: migration_fee_lamports < graduation_threshold_lamports
```

7 Holder rewards

The holder-reward share of each trading fee is distributed through a MasterChef/Synthetix-style accumulator-per-share model. When new reward lamports arrive, accrue_holder_rewards adds delta_acc = floor((pending_rewards + new_reward) * REWARD_ACC_SCALE / total_reward_eligible_supply) to a per-share accumulator, where REWARD_ACC_SCALE = 1e12 preserves precision across integer divisions. If eligible supply is momentarily zero, rewards park in pending_rewards_lamports and roll forward rather than being lost.

Reward eligibility is read from a program-controlled shadow balance (UserRewardRecord.eligible_balance), never from the live SPL token account. This closes an accumulator-inflation vector: a raw token transfer into a wallet cannot increase its reward eligibility, because only program mutations update the shadow balance. Every balance-changing instruction — buy, sell, auction claim, vested claim, holder claim — first checkpoints pending rewards, credits them to unclaimed, then mutates the balance and re-seeds the reward debt, so no phantom reward is ever credited retroactively.

The eligible supply itself is an independent counter equal to circulating_supply minus locked creator allocation, where locked allocation = creator_allocation_amount - allocation_claimed_amount when allocation_purchased is true, else 0. Locked creator tokens therefore earn no holder rewards. Every payout is capped to the holder pool available_holder_pool = vault_lamports - rent_minimum - real_sol_reserves - pending_platform_fee_lamports, so a claim can never dip into principal or the platform fee ledger even under an arithmetic edge case.

After graduation the same accumulator carries the post-graduation LP fees. claim_damm_fees collects both fee legs from the DAMM pool, splits them by the same bps ratios used on-curve, unwraps the wSOL holder leg into the curve vault as native lamports, and accrues it against the frozen graduation snapshot denominator. The base-token holder share (which has no native-SOL representation) folds into the creator leg. Because post-graduation DAMM buyers have no UserRewardRecord and no transfer hook is available, only balances tracked at the graduation snapshot earn a share of post-graduation fees.

```
delta_acc = floor((pending_rewards_lamports + new_reward) * REWARD_ACC_SCALE / total_reward_eligible_supply)
```

```
reward_per_share_acc += delta_acc
```

```
accrued = floor((eligible_balance * reward_per_share_acc) / REWARD_ACC_SCALE)
```

```
pending_reward = accrued - reward_debt
```

```
reward_debt_new = floor((balance_new * reward_per_share_acc) / REWARD_ACC_SCALE)
```

```
total_reward_eligible_supply = circulating_supply - locked_allocation
```

```
locked_allocation = (allocation_purchased) ? creator_allocation_amount - allocation_claimed_amount : 0
```

```
available_holder_pool = vault_lamports - rent_minimum - real_sol_reserves - pending_platform_fee_lamports
```

```
payout = min(unclaimed_lamports + pending_reward, available_holder_pool)
```

```
REWARD_ACC_SCALE = 1,000,000,000,000 (1e12)
```

8 Creator allocation and vesting

Creator allocation is a pay-to-buy mechanism, not a free grant. At initialize_launch the creator specifies creator_allocation_bps in [0, MAX_CREATOR_ALLOCATION_BPS = 8000] (0-80% of supply), constrained to whole percents (bps % 100 == 0) and permitted only on an Instant launch — an auction opening (commit_window_secs > 0) must carry no creator allocation, so the auction's single fee-free settle buy always fits the sellable supply and can never overshoot it. The allocation tokens, creator_allocation_amount = floor(to-

tal_supply * bps / 10000), are moved to a locked allocation vault at init but are NOT counted in circulating_supply — unpaid-for tokens do not pollute early supply accounting.

To acquire them the creator calls purchase_creator_allocation, which prices the entire allocation off the live bonding curve. The constant-product math removes the allocation from virtual token reserves and computes the new virtual SOL reserve by ceil division, so the creator can never underpay relative to market. The fee is taken pre-curve on the gross input, the net SOL is added to real_sol_reserves, and the allocation tokens enter circulating_supply now backed by that deposited SOL. This purchase can itself trigger graduation if it pushes real_sol_reserves past the threshold. The purchased tokens remain locked in the allocation vault — they are not sent to the creator until vesting releases them.

Vesting begins at graduation and releases the allocation linearly over vesting_months = min(bps/100, 12) months, each month being SECONDS_PER_MONTH = 2,592,000 seconds. The 12-month cap prevents pathologically long tails — an 8000-bps allocation would otherwise vest over 80 months. claim_vested_creator_allocation is repeatable and pays the newly vested delta each call.

Allocation tokens are excluded from real_token_reserves at all times (they live in a separate vault, never in the curve token vault) but are included in virtual_token_reserves so they remain priceable off the curve. Vested tokens released to the creator do not become holder-reward-eligible, because the reward model froze the eligible denominator at graduation; only holdings tracked at the graduation snapshot earn post-graduation fees.

```
creator_allocation_amount = floor(total_supply * creator_allocation_bps / 10000)
```

```
constraint: bps % 100 == 0 AND bps <= MAX_CREATOR_ALLOCATION_BPS = 8000
```

```
purchase price: new_virtual_token = virtual_token_reserves - allocation_amount
```

```
purchase price: new_virtual_sol = ceil(k / new_virtual_token) = (k + new_virtual_token - 1) / new_virtual_token
```

```
vesting_months = min(creator_allocation_bps / 100, 12)
```

```
vesting_duration_secs = vesting_months * 2,592,000
```

```
elapsed = clamp(now - graduated_at, 0, vesting_duration_secs)
```

```
total_vested = floor(allocation * elapsed / vesting_duration_secs)
```

```
claimable = total_vested - allocation_claimed_amount
```

9 Refund deadline

A vesting launch (creator_allocation_bps > 0 and allocation_purchased) may set an optional active_deadline. deadline_secs must be either 0 (opt out) or at least MIN_DEADLINE_SECS = 86,400 seconds (24

hours). Non-vesting launches must use 0. A deadline of 0 means the launch trades indefinitely in the pump.fun style and the failure check can never fire.

If a deadline is set and the launch has not graduated by active_deadline (computed as auction_start + deadline_secs), anyone may call the permissionless check_deadline_and_fail to transition the launch to Failed. This is a liveness guarantee: it does not depend on the creator or platform acting.

Once Failed, buy and sell are permanently rejected; only claim_refund and creator-stake slashing remain enabled — a terminal state. Each holder calls claim_refund, burning burn_amount of their tokens to receive floor(burn_amount * real_sol_reserves / circulating_supply) in SOL, then both counters decrement by the refunded amounts.

The refund is drawn only from real_sol_reserves, which is isolated from pending_platform_fee_lamports and pending_rewards_lamports even though all three reside in the same curve vault — paying refunds out of the fee or reward buckets would leave those claims insolvent. The formula reads live real_sol_reserves and circulating_supply at claim time rather than a snapshot at failure, so the per-token ratio stays consistent as earlier claimants withdraw and later claimants receive the same price.

```
active_deadline = auction_start + deadline_secs  (0 disables)
```

```
constraint: deadline_secs == 0 OR deadline_secs >= MIN_DEADLINE_SECS = 86,400
```

```
refund_lamports = floor(burn_amount * real_sol_reserves / circulating_supply)  (both read live at claim)
```

```
post-refund: circulating_supply -= burn_amount; real_sol_reserves -= refund_lamports
```

```
Failed is terminal: buy/sell rejected; only claim_refund and slash_creator_stake enabled
```

10 Fee model and economics

There are three fee touchpoints. Launch creation costs a flat 50,000,000 lamports (0.05 SOL) to the platform fee wallet. Trading incurs a configurable fee of fee_bps = 100 (1%) applied uniformly to buy input and sell output with a 1-lamport floor. Graduation withholds a configurable migration fee (typically 2,000,000,000 lamports = 2 SOL) from the SOL leg before DAMM seeding; the exact amount charged is recorded in migration_fee_paid_lamports as the source of truth, and configuration requires migration_fee_lamports < graduation_threshold_lamports so post-fee reserves remain viable.

Each trading fee is split into legs. The default configuration is platform 20% / creator 50% / holder 30% (platform_bps 2000, creator_bps 5000, holder_bps 3000), and the platform config enforces that the three legs sum to exactly 10000. Each leg is tracked in its own counter: the platform leg (net of any referral carve, below) accrues into pending_platform_fee_lamports, the creator leg into pending_creator_fee_lamports, and the holder leg into the reward accumulator (parking in pending_rewards_lamports whenever eligible supply is zero). Keeping the creator leg in a counter separate from the platform leg is what lets the referral carve reduce the platform share without ever shrinking the creator's payout. Each leg also increments a monotonic cumulative counter for reporting.

Vesting launches receive a boosted holder-and-creator tier funded from the platform leg. The boost is platform_bps / 2, redistributed half to creator and half to holders, so the default 20/50/30 becomes 10% / 55% / 35% (platform down 10, creator up 5, holder up 5). Because the platform and creator legs live in independent counters, claim_fee pays each out bit-exact from its own counter with no ratio reconstruction, so neither the vesting boost nor a referral carve can distort the creator's payout. On claim_fee the creator share is further split among registered fee recipients by weight_bps, subject to MAX_FEE_RECIPIENTS = 100 recipients and MAX_CREATOR_WALLETS = 5.

Referral program. Any wallet may register a random 16-byte referral code (register_referral_code) that binds the code to that wallet on-chain; the code, not the wallet, appears in a ?ref link, so the promoter's address stays private while only that wallet can claim what the code earns. When a buy or sell carries a valid referral, a cut is carved from that trade's platform leg — bounded by MAX_REFERRAL_BPS_OF_PLATFORM (5000 bps of the platform leg) and defaulting to DEFAULT_REFERRAL_BPS_OF_PLATFORM (2500) — and credited both to a per-(launch, referrer) ReferralEarnings account and to a launch-level pending_referral_lamports counter. The trader pays the same total fee; only the platform's own share is reduced. The referral path is fail-soft: if the passed referral accounts are missing, read-only, wrong-sized, aliased to a named account, or fail PDA re-derivation, the cut is zero and the full platform leg accrues, so a malformed or hostile referral can never block or misprice a trade. Referrers withdraw their earnings with claim_referral, whose payout is capped at min(owed, launch.pending_referral_lamports, vault principal platform creator rent); tying the cap to the launch's own tracked referral pool keeps the referral pool disjoint from the holder pool in the shared vault, so a referral claim can never draw from holder rewards, trader principal, or the other fee legs. The platform admin may additionally bind a code to a specific wallet at a custom rate (set_referral_rate) for KOL arrangements.

Creator-funded referrals stack on top of the platform-funded carve. A creator may opt their launch in (set_creator_referral_rate, gated on launch.creator only) to also route a bounded share of THEIR OWN creator leg — up to MAX_CREATOR_REFERRAL_BPS (5000 bps of the creator leg) — to the same referrer, recorded on a separate CreatorReferralConfig PDA so it never changes the TokenLaunch account layout. On a referred trade the program computes creator_cut = min(floor(creator_leg × creator_referral_bps / 10000), creator_leg) and nets BOTH legs before accrual: pending_platform += platform_leg platform_cut, pending_creator += creator_leg creator_cut, and pending_referral += platform_cut + creator_cut. Because each cut only moves lamports from its own leg's counter into the shared referral pool — and the creator leg is netted rather than double-booked — the four counters plus the holder leg still sum to exactly fee_lamports, so the vault invariant is preserved and the extra reward comes entirely out of the creator's own revenue. The whole creator-funded path is equally fail-soft: if the config account is absent, spoofed, or for a different launch, creator_cut is zero and the full creator leg accrues.

```
DEFAULT_REFERRAL_BPS_OF_PLATFORM = 2500, MAX_REFERRAL_BPS_OF_PLATFORM = 5000, MAX_CREATOR_REFERRAL_BPS = 5000
```

```
platform_cut = min(floor(platform_leg * referral_bps / 10000), platform_leg)
creator_cut  = min(floor(creator_leg  * creator_referral_bps / 10000), creator_leg)   (0 unless the creator opted in)
```

```
pending_platform += platform_leg - platform_cut ; pending_creator += creator_leg - creator_cut ; pending_referral += platform_cut + creator_cut
```

```
invariant per trade: (platform_leg - platform_cut) + (creator_leg - creator_cut) + (platform_cut
+ creator_cut) + holder_leg == fee_lamports
```

```
claim_referral payout = min(owed, launch.pending_referral_lamports, vault - real_sol_reserves
- pending_platform - pending_creator - rent)
```

A creator stake, computed as stake_lamports = graduation_threshold * stake_bps / 10000, is held during the launch and returned on graduation (or slashable on failure), aligning the creator with reaching graduation honestly. All economic parameters above — fee bps, splits, thresholds, and migration fee — are read from platform configuration rather than hardcoded, so exact values may differ per deployment while the structural invariants (legs summing to 10000, migration fee below threshold, mint and freeze authorities revoked) hold.

```
LAUNCH_FEE_LAMPORTS = 50,000,000 (0.05 SOL)
```

```
fee_lamports = max(floor(trade_amount * fee_bps / 10000), min_fee_lamports)
```

```
platform_leg = floor(fee_lamports * platform_bps / 10000)
```

```
creator_leg = floor(fee_lamports * creator_bps / 10000)
```

```
holder_leg = fee_lamports - platform_leg - creator_leg
```

```
default split: platform 2000 / creator 5000 / holder 3000 bps (20/50/30)
```

```
vesting boost = platform_bps / 2, split half/half -> tier 10% / 55% / 35%
```

```
constraint: platform_bps + creator_bps + holder_bps == 10000
```

```
migration_sol = real_sol_reserves - migration_fee_lamports  (typical 2 SOL)
```

```
constraint: migration_fee_lamports < graduation_threshold_lamports
```

```
stake_lamports = graduation_threshold * stake_bps / 10000
```

```
creator_leg_per_recipient = creator_share * recipient.weight_bps / 10000
```

11 Security model

The security posture rests on enforcement by program logic and by the AMM's own bytecode, plus explicit ledger isolation so that distinct pools of SOL held in one vault cannot cross-fund each other. This section states what the protocol structurally enforces, the threats it is designed against, and the limitations that remain.

Structural guarantees. Mint and freeze authorities must be revoked before a launch is accepted, so no new supply can be minted and no holder can be frozen. After graduation, 100% of the LP position is permanently locked by cp-amm's remove-liquidity guards, which no NFT authority, admin, or program path can override — this is the core liquidity-lock property. Migration is atomic, so no partial-migration state can strand funds. The blind auction settles as a deterministic, oracle-free batch buy of the whole committed pot into the standard curve, so trading continues from the price that buy set and no launch-gap arbitrage window opens. Holder-reward eligibility is read from a program shadow balance immune to raw-transfer inflation, and payouts are capped to the holder pool so principal and fee ledgers stay solvent. Refunds draw only from isolated principal (real_sol_reserves). Creator allocations are bought off the curve at a ceil-rounded price and vest, never granted free. Liveness backstops — deadline failure and stuck-graduation recovery — are permissionless.

Threat model. The design specifically addresses: post-graduation liquidity withdrawal (mitigated by the permanent lock); launch-open sniping and settle-then-buy sandwiches (mitigated by the gap-free batch-buy settle and the settle_cooldown_slots delay before Active trading); dust-drain micro-trade attacks (mitigated by the 1-lamport fee floor guaranteeing >= 2 lamports per round trip); reward-accumulator inflation via raw transfers (mitigated by the shadow balance); single-whale auction capture (mitigated by the per-committer bps cap); pre-created-pool graduation griefing (mitigated by the permissionless recovery path); and rent-redirection on vault closure (mitigated by requiring the rent sink to equal the configured platform fee wallet).

Honest limitations. The program has NOT undergone a formal third-party security audit; the guarantees above describe intended behavior of the current source, not audited-and-attested behavior. The program is deployed on devnet only — it has not been exercised under mainnet conditions, real economic incentives, or adversarial load at scale. The protocol makes NO price or return promise: bonding-curve and post-graduation prices are set entirely by market participants and can fall to zero. Permanent liquidity locking prevents LP withdrawal but does not prevent price decline — holders can still lose value as others sell. A blind auction always settles into the curve and distributes tokens; it has no under-fill refund, so a committer who changes their mind must withdraw before settlement — otherwise refunds exist only via the Failed-state flow. The refund deadline is opt-in and only available to vesting launches; a deadline of 0 means no automatic failure and no deadline-triggered refund. Finally, several parameters are platform-configurable, so a given deployment's exact thresholds, fees, and gates depend on its configuration, and the trust properties are only as strong as that configuration and the integrity of the deployed program and its cp-amm dependency.

- Enforced by cp-amm bytecode: permanent LP lock, unbreachable by any NFT/admin/program authority.
- Enforced by launchpad_core: authority revocation, atomic migration, gap-free batch-buy settle, shadow-balance rewards, ledger isolation, ceil-priced creator buy, permissionless liveness backstops.
- Not enforced / not guaranteed: audit status, mainnet-hardening, token price, protection from market decline, under-fill refund of committed SOL (auctions always settle into tokens; withdraw before settlement instead).

12 Status and roadmap

LIMELIGHT is implemented as the Solana program launchpad_core with program id 342po67kbxoKXcjpfiQQcH47pd6WnX37Q2YrWaEkxfGq, currently deployed on devnet. The full lifecycle described in this document — creation, optional blind auction, curve trading, graduation with permanent DAMM v2 lock, holder rewards, creator vesting, and refunds — is present in the deployed program.

The immediate priorities are a formal third-party security audit of the program and its cp-amm integration, and expanded adversarial and load testing under conditions approximating mainnet. Neither has been completed; until they are, the guarantees in Section 11 should be read as properties of the current source rather than externally attested facts.

Mainnet deployment is gated on the completion of the audit and testing above. This whitepaper will be revised as those milestones are reached and as any platform-configurable parameters are finalized for a production deployment; the structural invariants are expected to remain stable while specific numeric configuration may change.

Structural guarantees

- Mint authority and freeze authority must be revoked (None) before a launch is accepted, so no additional supply can be minted and no holder account can be frozen post-launch.
- After graduation, 100% of the LP position is permanently locked by Meteora cp-amm's own remove_liquidity/remove_all_liquidity/close_position guards; no position-NFT authority, admin, or program path can withdraw it.
- Graduation migration is atomic (pool creation + seeding + permanent lock in one CPI sequence): it fully succeeds or the whole transaction reverts, so no partial-migration state can exist.
- The DAMM v2 pool address is a canonical PDA derived from the customizable-pool prefix and ordered mint pair under the cp-amm program ID, so it cannot be spoofed.
- Migration price is computed deterministically (Q64.64 sqrt-price solver, liquidity = min(from_base, from_quote)) so migration introduces no price slippage; MIN_SQRT_PRICE <= sqrt_price <= MAX_SQRT_PRICE is enforced.
- The blind auction always settles into the curve as a single fee-free batch buy of the whole committed pot — there are no size or breadth gates and no under-fill outcome.
- Every committer pays the same average price total_committed / tokens_out (deterministic, oracle-free), and trading continues from the price that batch buy set — no launch-open gap or sniper window.
- No single wallet may commit more than max_single_committer_bps_of_total of the running total (founding commit exempt), and per-committer allocations use floor division so summed allocations never exceed the auction tranche.
- Active trading is blocked until active_trading_starts_slot = settlement_slot + settle_cooldown_slots, blunting settle-then-buy sandwich attacks.
- The 1-lamport fee floor guarantees every buy-sell round trip costs at least 2 lamports combined, closing the dust-drain vector.
- The constant-product invariant k = virtual_sol_reserves * virtual_token_reserves is exactly preserved across each trade after fee application.
- Holder-reward eligibility is read from a program-controlled shadow balance, never from the live SPL token account, so raw token transfers cannot inflate reward eligibility.
- Every balance-changing instruction checkpoints pending rewards before mutating supply, so no phantom reward is credited retroactively.

- Holder-reward payouts are capped to available_holder_pool = vault_lamports - rent_minimum - real_sol_reserves - pending_platform_fee_lamports, so principal and platform-fee solvency are guaranteed.
- real_sol_reserves (refundable principal) is accounted separately from pending_platform_fee_lamports and pending_rewards_lamports even within one vault; refunds draw only from real_sol_reserves.
- Pro-rata refunds use live real_sol_reserves and circulating_supply at claim time, so every claimant receives the same per-token price.
- Creator allocation is purchased off the live curve at a ceil-rounded price (never underpaid) and is not counted in circulating_supply until paid for; it vests linearly over min(bps/100, 12) months.
- creator_allocation_bps must be a whole percent (bps % 100 == 0) and at most MAX_CREATOR_ALLOCATION_BPS = 8000 (80% of supply).
- Trading-fee legs must sum to exactly 10000 bps, and migration_fee_lamports must be strictly less than graduation_threshold_lamports.
- Liveness backstops are permissionless: check_deadline_and_fail (for vesting launches past their >= 24h deadline) and recover_stuck_graduation (pre-created-pool griefing), the latter being the only path that can move Graduating to Failed.
- On vault closure the rent sink is required to equal config.platform_fee_wallet, so a permissionless caller cannot redirect reclaimed rent.

What is not guaranteed

- The program has NOT undergone a formal third-party security audit; the stated guarantees describe intended behavior of the current source, not externally attested behavior.
- The program is deployed on devnet only and has not been exercised under mainnet conditions, real economic incentives, or adversarial load at scale.
- The protocol makes NO price or return promise: bonding-curve and post-graduation prices are set entirely by market participants and can fall to zero.
- Permanent liquidity locking prevents LP withdrawal but does not prevent price decline — holders can still lose value as other participants sell.
- A blind auction always settles into the curve and distributes tokens; it has no under-fill refund. A committer who changes their mind must withdraw before settlement — otherwise refunds are available only through the Failed-state claim_refund flow.
- The refund deadline is opt-in and available only to vesting launches; a deadline of 0 means no automatic failure and no deadline-triggered refund (the launch may trade indefinitely).
- Many economic parameters (fee bps, splits, graduation threshold, migration fee, per-committer auction cap, commit-window bounds, cooldown) are platform-configurable, so a given deployment's exact values depend on its configuration.
- Trust properties depend on the integrity of the deployed launchpad_core program and its Meteora cp-amm dependency; a compromised or mis-configured deployment, or a defect in cp-amm, could weaken the guarantees.
- Small rounding dust (sub-10 raw token units / sub-10 lamports) can remain in migration-authority ATAs after graduation and is not swept; it is economically negligible but non-zero.

- Post-graduation LP-fee rewards accrue only to balances tracked at the graduation snapshot; DAMM buyers after graduation have no tracked reward record and earn no holder share.