

Smart LP Hook

Uniswap v4 hook plus single sided ladder vault for Robinhood Chain stock tokens.
Integration guidelines for the Hookr team.

Date: 2026-10-13
Repository commit: 810c9aaf668a | source hash: 11e14b4aeafc
Compiler: solc 0.8.26, cancun, via IR (Foundry)
Tests: 64 unit, 6 mainnet fork (Robinhood Chain 4663)
Publisher: SB (BVI) Ltd, StonkBrokers
Contact: simple@clutch.market, Telegram simplefarmer69

Status: LIVE on Robinhood Chain 4663
Hook, lens, USDG/NVDA pool and sINVDA ladder are deployed. Addresses in section 3 and hooks/deployments/smart-lp-hook.4663.json. Weekend equity feeds stay stale until Monday 00:00 UTC; trading still works under the stale fee path.

Source and ABIs
Contracts live under hooks/ in the StonkBrokers repository (Foundry): SmartLpHook.sol, SmartLpLadder.sol, SmartLpHookLens.sol. Byte exported ABIs ship beside this guide as SmartLpHook.abi.json, SmartLpLadder.abi.json and SmartLpHookLens.abi.json, also served from stonkbrosers.io/integration/hooks/.

Deployed bytecode sizes

Contract	Role	Runtime size
SmartLpHook	Dynamic fee, recapture skim, flush	16,116 B (65.6% of EIP-170)
SmartLpLadder	Single sided covered call ladder vault (ERC-20 shares)	18,780 B (76.4% of EIP-170)
SmartLpHookLens	One call read surface for UIs	6,809 B (27.7% of EIP-170)

Contents

- 1. What the hook does
- 2. Why this is not an owned root on Hookr
- 3. Addresses
- 4. Hook identity and hooklist entry
- 5. Pool keys
- 6. Swapping from the Hookr UX
- 7. Contract functions
- 8. Reading state (lens)
- 9. Ladder vault UX
- 10. Fee model and parameters
- 11. Events and revert selectors
- 12. Risks and limitations to show users
- 13. Verification ledger
- 14. Integration checklist (for both teams)

1. What the hook does

Stock tokens on Robinhood Chain are priced by Chainlink equity feeds that update during US market hours and go quiet from Friday close to Monday 00:00 UTC. A plain constant fee pool sells that information for free: whoever closes the gap between the pool price and the fresh Chainlink mark keeps the whole move and the LPs eat the adverse selection. The Smart LP Hook turns that gap into LP income.

Three mechanisms, all inside one hook:

- Oracle anchored asymmetric fee. Every swap is priced against the Chainlink mark converted to a pool tick. A swap that pushes the pool away from the oracle pays a higher fee that grows with the gap in ticks. A swap that pulls the pool toward the oracle pays the base fee. When the feed is stale the pool charges one flat symmetric stale fee.
- Opening bell recapture. When a fresh Chainlink round lands, the hook arms a short recapture window. During the window any swap that closes the pool to oracle gap pays a skim proportional to the gap it closed, taken from the swap output. That skim is held for in range LPs and donated to the pool on the next flush.
- Single sided ladder vault. Depositors put in the stock token only. The vault places the stock as sell rungs at fixed steps above the Chainlink mark (a covered call ladder), resets every epoch, and never redeploys the quote it earns. Depositors withdraw stock plus the quote the ladder sold into.

Fee policy: the base LP fee goes 100 percent to LPs. A 10 percent performance fee applies only to the add on income (recaptured arb and the fee surcharge above base). Out of that protocol share, partnerBps (5000 = half) is carved out for Hookr; the remaining protocol share splits 50 percent to the StockBooster and 50 percent to STONKBROKER buybacks. The hook never takes a cut of the base fee or of LP principal.

ERC-8056 note: Robinhood stock tokens carry a `uiMultiplier` that moves on reinvested dividends. The hook and the vault work in RAW units. Dividends accrue to the raw tokens the LP holds; nothing in the hook has to rebase.

2. Why this is not an owned root on Hookr

Hookr owned roots are unmodified Hookr roots spawned by `HookrOwnedRootFactory`. A custom hook cannot be an owned root, and `HookrRouter`, `HookrQuoter` and the Hookr SDK refuse pools whose hook is not a root. The Smart LP Hook is a standalone hook with its own address and flag word, so the integration path is the one Hookr documents for third party hooks:

- Analyze a Hook: the public source at a pinned commit, the test suite including failure paths, the exact `PoolKey` hook address, and a live readback of the hook state.
- Publish a Hook: the hooklist entry (section 4) plus the pool keys (section 5).
- Partner Integration Review: this document, the audit style notes in section 11, and a walkthrough call if useful.

Swaps against Smart LP pools route through the Uniswap Universal Router with empty `hookData`. The Hookr UX can quote them with the Uniswap V4 Quoter and surface them in the hook directory and pool pages while keeping its own router for root pools.

3. Addresses

Item	Chain	Address
SmartLpHook	Robinhood Chain 4663	0x47137a7733E2B4DBD0e25416d5D5B29C8A1Da0c4
SmartLpHookLens	Robinhood Chain 4663	0x29Eaf335fed9270ba98BD5dd99519bb3af4C3F00
SmartLpLadder sINVDA (USDG/NVDA)	Robinhood Chain 4663	0x3Ddf0412398D806AF1069B0EA29242b6BD22fd2B
Uniswap v4 PoolManager	Robinhood Chain 4663	0x8366a39CC670B4001A1121B8F6A443A643e40951
Uniswap v4 PositionManager	Robinhood Chain 4663	0x58daec3116aae6D93017bAAea7749052E8a04fA7
Uniswap V4 Quoter	Robinhood Chain 4663	0x8dc178efb8111bb0973dd9d722ebff267c98f94
Universal Router 2.1.1	Robinhood Chain 4663	0x8876789976dEcBfCbBbe364623C63652db8C0904
Permit2	Robinhood Chain 4663	0x000000000022D473030F116dDEE9F6B43aC78BA3
USDG (quote, 6 decimals)	Robinhood Chain 4663	0x5fc5360D0400a0Fd4f2af552ADD042D716F1d168
NVDA stock token (18 decimals)	Robinhood Chain 4663	0xd0601CE157Db5bdC3162BbaC2a2C8aF5320D9EEC

Item	Chain	Address
Chainlink NVDA/USD	Robinhood Chain 4663	0xC9d16E4f2569b9E3ea0468fD85844953713DC2a2
Chainlink AAPL/USD	Robinhood Chain 4663	0xBb11A21267cFD6b63d4935d99a499133DD1744ACb
Chainlink SPCX/USD	Robinhood Chain 4663	0xB265810950ba6c5C0F821c9963014a56fD8Bffb
Chainlink TSLA/USD	Robinhood Chain 4663	0x4A1166a659A55625345e9515b32adECea5547C38

The deploy rail (hooks/script/SmartLpHookDeploy.s.sol) writes deployments/smart-lp-hook.4663.json with hook, lens, owner, booster, buyback, partnerTo, partnerBps, the pool id, the ladder address and the CREATE2 salt.

USDG/NVDA pool id: 0x3b0a3db8c4d25fd428792aa6718796a3b87945e2b1d8b282500a753155a16e8b (currency0 USDG, currency1 NVDA, fee 0x800000, tickSpacing 60, hooks 0x47137a7733E2B4DBD0e25416d5D5B29C8A1Da0c4). Verified on chain as keccak256(abi.encode(PoolKey)). The pool id includes the hook address, so it differs from the superseded 0x17994a68...e573 pool of the previous hook generation.

Deploy txs: hook 0xe9e21b15073c5d5b465d33919e6a628c2ea8ca4de15dad1c0ce1ef5ea6027961, lens 0x8487ad742364dd1db7b869b17562cccec280c3263946ba19258e64693a3595f17, setPartner 0x193da34e191e36ad642e13ffb5dc69e461df67cc6a6c0fed64f012efd17cdbf9, register 0x1621d65cc06adf7479403a6ae98b8df81330436e889911dbbe92dd243002d1cf, initialize 0x007b2438d794f39827da91666742286a0e8b2b705dffee5b2ec2e7927aafbc40, ladder 0xa60ee6dfeac358be0ad4b52488a578390ec4707f1021720a381a191a2249b139.

3.1 Hookr fee leg

The performance fee (perfBps, 10 percent of the add on income) is the only protocol share this hook ever takes. Inside afterSwap the hook computes protocolShare, then carves partnerShare = protocolShare x partnerBps / 10000 out of it (5000 bps = half of the performance fee) and books both per side (accruedProtocol0/1, accruedPartner0/1). flush(key) pays the partner accrual to partnerRecipient when it is set and otherwise leaves it accrued (the payout helper returns 0 and the balance keeps growing), then splits the remaining protocol share between the StockBooster and the buyback wallet. Nothing is skipped: the Hookr share exists from the first swap, only the destination waits on the Hookr CA.

Wiring the Hookr CA: owner SmartLpHook.setPartner(<hookrCA>, 5000) from 0xD4aF10Bc56b08f8fc2990a3ADeeD4b757ae6A26C (reverts BadParams above 10000, emits PartnerSet), then anyone flush(key) per registered pool. Events to index: PartnerSet(to, bps), PartnerFlushed(poolId, partner0, partner1). Lens poolView exposes accruedPartner0/1, partnerRecipient and partnerBps.

4. Hook identity and hooklist entry

The hook address encodes its permissions in the low 14 bits. The Smart LP Hook uses exactly four flags:

Flag	Set
BEFORE_INITIALIZE	yes
BEFORE_SWAP	yes
AFTER_SWAP	yes
AFTER_SWAP_RETURNS_DELTA	yes
everything else	no

Flag word: 0x20c4. Any candidate hook address must satisfy address & 0x3fff == 0x20c4; requiredFlags() on the contract returns the same word and getHookPermissions() returns the struct. The hook is deployed through the deterministic CREATE2 deployer 0x4e59b44847b379578588920cA78FbF26c0B4956C with a mined salt, so the address is fixed before the broadcast.

Hooklist entry (live):

```
{
  "name": "Smart LP Hook",
  "author": "StonkBrokers",
  "address": "0x47137a7733E2B4DBD0e25416d5D5B29C8A1Da0c4",
  "chainId": 4663,
  "flags": "0x20c4",
  "dynamicFee": true,
  "upgradeable": false,
  "requiresCustomSwapData": false,
  "vanillaSwap": true,
  "beforeSwapReturnsDelta": false,
  "afterSwapReturnsDelta": true,
  "source": "https://github.com/Clutch-L4bs/stonkbrosers/tree/main/hooks",
  "docs": "StonkBrokers-Smart-LP-Hook-Hookr-Integration.pdf",
  "tags": ["stock-tokens", "oracle-fee", "covered-call", "robinhood-chain"]
}
```

Notes for the directory card:

- Dynamic fee: yes. beforeSwap returns the fee with the override flag on every swap. The slot0 LP fee is never the truth; read quoteFee or the lens.
- Swap deltas: yes, afterSwap can take a skim from the unspecified currency. Routers must honor the returned delta (the Universal Router does).
- Custom swap data: none. Pass empty hookData on every swap and liquidity call. Non empty data is ignored.
- Upgradeable: no. No proxy, no delegatecall. Owner levers are parameter bounds only (section 9).

5. Pool keys

Every Smart LP pool is a stock token paired with a quote token, dynamic fee, tick spacing 60, hook = SmartLpHook.

Field	Value
currency0 / currency1	sorted by address (the stock can be either side)
fee	0x800000 (LPFeeLibrary.DYNAMIC_FEE_FLAG)
tickSpacing	60
hooks	SmartLpHook address

Pool creation sequence (owner only, once per pool):

1. hook.registerPool(key, feed, stockIs0) stores the Chainlink feed and token decimals for the pool id. Reverts WrongHook, NotDynamicFee, ZeroAddress, PoolAlreadyRegistered or BadParams.
2. poolManager.initialize(key, sqrtPriceX96) with the sqrt price read from hook.readMark(poolId) so the pool opens on the oracle. beforeInitialize reverts PoolNotRegistered for any key that skipped step 1, so a stranger cannot open a Smart LP pool on an unregistered pair.

Pool id is keccak256(abi.encode(key)) as in v4 core. Launch wave: USDG/NVDA first (the live hookless 0.30 percent pool is the reference), then AAPL, SPCX and TSLA against USDG.

6. Swapping from the Hookr UX

Smart LP pools are vanilla from the trader side. There is no hook data, no allowlist, no per wallet cap and no sandwich surface (Robinhood Chain has no public mempool).

Quote: call the Uniswap V4 Quoter quoteExactInputSingle with the pool key, zeroForOne, exactAmount and empty hookData. The quoter simulates beforeSwap and afterSwap, so the returned amount already includes the dynamic fee and any recapture skim.

Preview the fee before quoting (cheap, one call): hook.quoteFee(key, zeroForOne) returns (fee, oracleTick, poolTick, fresh, away). fee is in pips (1e6 = 100 percent), away says whether this direction moves the pool away from the oracle. Show it on the swap card as "pool fee right now".

Execute: Universal Router execute with the V4_SWAP command, actions SWAP_EXACT_IN_SINGLE, SETTLE_ALL,

TAKE_ALL, the same pool key and empty hookData. Token approvals go through Permit2 as for any v4 pool. Set amountOutMinimum from the quoter with the user slippage; the skim is deterministic per block so the quote and the fill match unless the pool moves between them.

UX copy suggestions:

- When away is true: "This trade moves the pool away from the Chainlink mark. Fee X percent." and show the oracle price beside the pool price.
- When fresh is false: "Chainlink feed paused (market closed). Flat Y percent fee until the feed resumes."
- When recaptureActive is true (lens): "Opening bell window. Trades that close the gap share part of the move with LPs."

7. Contract functions

What each public surface does. Arguments and return values match the shipped ABIs. Pool keyed views take either a PoolId (bytes32, keccak256(abi.encode(key))) or a full PoolKey (currency0, currency1, fee, tickSpacing, hooks).

7.1 SmartLpHook

Identity and permissions:

Function	Who	What it does
getHookPermissions()	anyone	Returns the Uniswap Hooks permissions struct (beforeInitialize, beforeSwap, afterSwap, afterSwapReturnDelta). Pure.
requiredFlags()	anyone	Returns 0x20c4, the low 14 bits every valid CREATE2 address must encode. Pure.
defaultFeeParams()	anyone	Returns the default FeeParams written into a pool at registerPool. Pure.

Owner setup (two step Ownable; owner today is the interns ops deployer 0xD4aF...A26C):

Function	Who	What it does
registerPool(key, feed, stockIs0)	owner	Binds a PoolKey to a Chainlink feed and which side is the stock. Reads token decimals on chain. Writes default fee params. Emits PoolRegistered and FeeParamsSet. Reverts WrongHook, NotDynamicFee, ZeroAddress, PoolAlreadyRegistered, BadParams. Call once before poolManager.initialize.
setFeeParams(id, params)	owner	Replaces fee policy for one pool. Bounds enforced on chain (see section 10). Emits FeeParamsSet.
setRecipients(booster, buyback)	owner	Sets the two protocol payout addresses for flush. Neither may be zero. Emits RecipientsSet. Live: booster = Directed Clock In 0x1f12fe...81c9, buyback = MM wallet 0xd267...bde8.
setPartner(to, bps)	owner	Hookr partner leg: bps of the protocol share for to. Live: to = 0x0 (pending the Hookr CA), bps = 5000. Reverts BadParams above 10000. Emits PartnerSet.
transferOwnership / acceptOwnership	owner / pending	Standard two step Ownable. No path to LP principal or accrued skims.

Read surface (swap cards and keepers):

Function	Who	What it does
poolConfig(id)	anyone	Feed address, stockIs0, stock/quote decimals, registered. Live USDG/NVDA: feed NVDA/USD, stockIs0=false, decimals 18/6.
feeParams(id)	anyone	Full fee policy tuple for the pool (base, slope, max, stale, windows, shares, deadband).
poolState(id)	anyone	Accrued LP, protocol and partner amounts in currency0/1 (accruedPartner0/1), last feed updatedAt, recaptureUntil.
partnerRecipient() / partnerBps()	anyone	Live Hookr partner recipient and share of the protocol leg.
oracleMark(id)	anyone	(answer, updatedAt, tick, fresh). Converts the Chainlink answer into a pool tick. fresh is false when age > staleAfter. Live audit: answer ~230.36, tick 221924, fresh flips with the equity calendar.
readMark(id)	anyone	Same mark as oracleMark but reverts BadFeedAnswer / feed errors instead of returning soft failure. Used for initialize sqrt price.

Function	Who	What it does
quoteFee(key, zeroForOne)	anyone	(feePips, oracleTick, poolTick, fresh, away). The fee the next swap in that direction would pay right now. When stale, both directions return staleFee (live default 10000 = 1%). Cheap preview for UX.

Swap path (PoolManager only; traders never call these):

Function	Who	What it does
beforeInitialize	PoolManager	Reverts PoolNotRegistered if the key was not registered. Blocks stranger initialize on this hook address.
beforeSwap	PoolManager	Prices the swap: reads oracle, arms recapture when a fresh round lands, returns dynamic fee with override flag. Emits SwapPriced (and RecaptureArmed when the window opens). Empty hookData required for vanilla UX.
afterSwap	PoolManager	Optionally skims from the unspecified currency (recapture + surcharge), splits LP vs protocol accrual, carves partnerBps of the protocol share into the partner accrual, returns delta. Emits Skimmed. Cap = maxSkimBps of output.
flush(key)	anyone	Permissionless. Donates accrued LP skim into the pool (needs nonzero liquidity), pays the partner accrual to partnerRecipient when set (PartnerFlushed), then pays the remaining protocol halves to booster and buyback. Emits Flushed. Reverts NothingToFlush when every accrual is zero (live empty pool returns that selector 0xeb694a3c).
unlockCallback	PoolManager	Internal settle path used by flush. Reverts NoUnlockInProgress if called outside that flow.

Liquidity and donate callbacks revert HookNotImplemented. The hook does not gate mint/burn of LP positions.

7.2 SmartLpHookLens

Stateless helper. No storage, no owner, no value path. Point it at any deployed hook or ladder.

Function	Who	What it does
poolView(hook, key)	anyone	One call pool page: slot0 (sqrtPriceX96, poolTick, liquidity, lpFeeSlot0), oracle (answer, updatedAt, tick, fresh), gapTicks, directional fees and away flags, recapture window, accrued LP and protocol amounts. Live USDG/NVDA readback matches hook views wei for wei.
ladderView(ladder)	anyone	One call vault page: share supply, idle stock/quote, NAV, position amounts, epoch clocks, pause flag, rung array, config, oracle mark + freshness, pool tick, staleAfter.

Prefer the lens from the UX; fall back to direct hook/ladder views if a fork has no lens deployed.

7.3 SmartLpLadder (sINVDA)

One vault per pool. ERC-20 share token. Immutables: poolManager, feed, stockIs0, stockDecimals, quoteDecimals, poolId. Live sINVDA poolId equals the USDG/NVDA pool id above.

Identity and config:

Function	Who	What it does
name / symbol / decimals / totalSupply / balanceOf	anyone	Standard ERC-20 share surface. Transfer event only (no allowance API; shares move via deposit/withdraw).
poolKey()	anyone	Reconstructs the PoolKey from immutables.
stockToken() / quoteToken()	anyone	The ERC-20s the vault pulls and pays. Live: NVDA / USDG.
config()	anyone	Rung ladder and gates. Live defaults: rungBps = [250, 500, 750, 1000], widthSpacings = 2, epochSecs = 604800 (7d), staleAfter = 10800 (3h), depthGateMult = 0 (off), minDepositQuote = 10e6 (10 USDG), capQuote = 0 (uncapped).
setConfig(c)	owner	Replaces config with on chain bounds. Emits ConfigSet.
setPaused(p)	owner	Blocks deposit and deploy only. Withdraw always stays open. Emits PausedSet.

Mark and accounting:

Function	Who	What it does
mark()	anyone	(Mark, fresh) from the vault feed. Same freshness rule as the hook.

Function	Who	What it does
nav()	anyone	Vault NAV in quote units at the current mark (idle stock marked + idle quote + in range position value).
totalAssets()	anyone	(stockAmt, quoteAmt) idle balances held by the vault contract.
positionAmounts()	anyone	Stock and quote locked inside live rungs (PoolManager positions).
rungs()	anyone	Array of (tickLower, tickUpper, liquidity) for the open ladder. Empty until first deploy.
epoch() / epochStartedAt / epochEndsAt()	anyone	Current epoch index, start timestamp, and when roll becomes available.
previewDeposit(stockIn)	anyone	Shares that would mint for stockIn at current NAV. Does not enforce freshness (deposit does). Live: 1e18 NVDA previews ~230e6 share units when mark is ~230 USDG.
previewWithdraw(shares)	anyone	(stockOut, quoteOut) pro rata including a burn of rung liquidity.

Lifecycle (depositors and keepers):

Function	Who	What it does
deposit(stockIn, to)	holder	Pulls stock (approve vault first), mints shares to to at NAV. Reverts Paused, ZeroAmount, FeedStale (live weekend probe returns selector 0x0a1d52c0), DepositTooSmall, CapExceeded, DepthGate. Idle stock sits until deploy.
deploy()	anyone	Permissionless. First call of an epoch builds rungs above the oracle mark; later calls top them up with idle stock only. Never deploys quote. Emits Deployed. Reverts Paused, FeedStale, NothingToDeploy.
roll()	anyone	Permissionless after epochEndsAt. Burns every rung, keeps quote idle forever, redeploys remaining stock on a fresh mark into a new epoch. Emits Rolled then the next Deployed. Reverts EpochNotOver, FeedStale.
withdraw(shares, to)	holder	Burns shares, pulls pro rata rung liquidity, pays stock + quote to to. Always open when paused or stale. Reverts ZeroAmount, InsufficientShares.
unlockCallback	PoolManager	Internal modifyLiquidity settle. Not a user endpoint.

Owner has no rescue of stock, quote, or shares. The only owner levers are setConfig and setPaused.

8. Reading state (lens)

SmartLpHookLens.poolView(hook, key) returns everything a pool page needs in one call:

Field	Meaning
sqrtPriceX96, poolTick, liquidity	slot0 and total liquidity from the PoolManager
lpFeeSlot0	the stored slot0 fee (informational only, dynamic fee overrides it)
oracleAnswer, oracleUpdatedAt, oracleTick, oracleFresh	Chainlink mark converted to a pool tick
gapTicks	poolTick minus oracleTick
feeZeroForOne, feeOneForZero	fee in pips per direction right now
awayZeroForOne, awayOneForZero	which direction is away from the oracle
recaptureActive, recaptureUntil	opening bell window state
accruedLp0, accruedLp1	skim held for LPs, donated on the next flush
accruedProtocol0, accruedProtocol1	performance fee held for booster and buyback (after the partner carve)
accruedPartner0, accruedPartner1	Hookr partner share held until flush
partnerRecipient, partnerBps	live Hookr recipient and share of the protocol leg

Direct hook views: poolConfig(id), feeParams(id), poolState(id), oracleMark(id) and readMark(id).

Suggested pool page widgets: pool price vs Chainlink price with the gap in ticks, the two directional fees, a "market open / closed" chip driven by oracleFresh, and an "LP recapture held" tile summing the accrued LP amounts.

9. Ladder vault UX

One SmartLpLadder per pool. It is an ERC-20 share token (name, symbol, balanceOf, totalSupply, Transfer event) and it holds only the stock token and the quote token of its pool.

Lifecycle:

1. deposit(stockIn, to) pulls the stock token (approve the vault first), mints shares at NAV, keeps the stock idle. Reverts Paused, ZeroAmount, FeedStale (no deposits while the market feed is closed), DepositTooSmall (below minDepositQuote, default 10 USDG), CapExceeded, DepthGate.
2. deploy() is permissionless. It builds the rungs on the first call of an epoch (Deployed event with the anchor tick) and tops them up on later calls. Only stock is ever deployed.
3. The rungs sit above the mark at rungBps steps (default 2.5, 5, 7.5 and 10 percent), each widthSpacings wide (default 2 x 60 ticks). As price rises through a rung the stock sells into quote and collects swap fees.
4. roll() is permissionless once epochEndsAt() has passed (default 7 days). It pulls every rung, keeps the quote idle forever, and rebuilds the ladder from the stock side on the fresh mark. Reverts EpochNotOver, FeedStale.
5. withdraw(shares, to) is always open, even when paused or stale. It burns the pro rata slice of every rung and pays stock plus quote.

Reads: previewDeposit(stockIn), previewWithdraw(shares), nav() (quote units), totalAssets() (stock, quote), positionAmounts(), rungs(), epoch(), epochEndsAt(), mark(), config(), paused(). SmartLpHookLens.ladderView(ladder) bundles all of it with the feed and pool ticks.

UX flows:

- Stake card: stock balance, amount input with Max, "You receive N shares" from previewDeposit, a one line note that the vault sells stock into quote as price climbs and keeps the quote.
- Position card: shares, value in quote from nav, stock and quote split from previewWithdraw, rung bars from rungs() against the current pool tick, epoch countdown from epochEndsAt.
- Keeper buttons: "Deploy idle stock" when totalAssets().stock minus the amount already in rungs is above the minimum, "Roll epoch" when block.timestamp >= epochEndsAt(). Both are safe for anyone to press.
- Closed market state: when mark() returns fresh = false, disable Stake, Deploy and Roll with the closed market copy. Withdraw stays enabled.

10. Fee model and parameters

Per pool FeeParams (owner set, bounded on chain):

Parameter	Default	Bound
baseFee	3000 (0.30 percent)	<= maxFee
slopePerTick	25 pips per tick past the deadband	any
maxFee	30000 (3 percent)	<= 100000
staleFee	10000 (1 percent)	<= 100000
staleAfter	10800 s (3 h)	!= 0
recaptureSecs	900 s	any
recaptureShareBps	5000 (half of the gap closed)	<= 10000
maxSkimBps	500 (5 percent of output)	<= 1000
perfBps	1000 (10 percent)	<= 2000
deadbandTicks	10	any

Fee per swap: if the feed is stale, staleFee. Otherwise gap = poolTick - oracleTick; if the swap direction is away from the oracle and |gap| > deadbandTicks, fee = baseFee + (|gap| - deadbandTicks) x slopePerTick capped at maxFee; else baseFee.

Skim per swap (taken from the swap output, only when fresh):

- Recapture: during the window, if the swap reduced the gap, output x gapClosedBps x recaptureShareBps / 1e8.
- Surcharge: when the applied fee was above base, output x (fee - baseFee) / 1e6 x perfBps / 1e4.

- Total capped at output x maxSkimBps / 1e4.
- Split: recapture skim goes (1 - perfBps) to LPs and perfBps to protocol; the surcharge skim is 100 percent protocol.
- Partner carve: partnerBps (5000) of every protocol share is booked for Hookr; the StonkBrokers halves are computed on what remains.

flush(key) is permissionless: it donates the LP share into the pool (only when the pool has liquidity, so it lands on in range LPs) and pays the Hookr partner accrual to partnerRecipient when it is set, and sends the protocol halves to boosterRecipient and buybackRecipient. Reverts NothingToFlush. A keeper calls it daily; the UX can expose it as "Distribute recapture".

Owner levers: setFeeParams, setRecipients, setPartner, registerPool, two step transferOwnership and acceptOwnership. There is no pause, no fee switch on the base fee, no path to LP principal and no upgrade.

11. Events and revert selectors

Hook events:

Event	When
PoolRegistered(poolId, feed, stockIs0, stockDecimals, quoteDecimals)	registerPool
FeeParamsSet(poolId, params)	registerPool defaults and setFeeParams
RecipientsSet(booster, buyback)	setRecipients
SwapPriced(poolId, fee, oracleTick, poolTick, fresh, away)	every beforeSwap
RecaptureArmed(poolId, until, feedUpdatedAt)	first swap after a fresh feed round
Skimmed(poolId, currency, lpShare, protocolShare, gapClosedTicks)	afterSwap with a nonzero skim
Flushed(poolId, lp0, lp1, protocol0, protocol1)	flush
PartnerSet(to, bps)	setPartner
PartnerFlushed(poolId, partner0, partner1)	flush with a set partner recipient and a nonzero partner accrual

Ladder events: Deposited(from, to, stockIn, shares, navBefore), Withdrawn(from, to, shares, stockOut, quoteOut), Deployed(epoch, anchorTick, stockDeployed, rungs), Rolled(epoch, stockBack, quoteBack), ConfigSet(config), PausedSet(paused), Transfer.

Revert selectors:

Selector	Error	Where
0xae18210a	NotPoolManager()	hook callbacks called by anyone else
0x0a85dc29	HookNotImplemented()	liquidity and donate callbacks
0x739f4185	PoolNotRegistered()	initialize on an unregistered key
0xb3e8301e	PoolAlreadyRegistered()	registerPool twice
0xb07a5703	WrongHook()	key.hooks is not this hook
0xcf587f14	NotDynamicFee()	key.fee is not 0x800000
0x019af637	BadParams()	setFeeParams out of bounds, ladder config
0xeb694a3c	NothingToFlush()	flush with zero accruals
0x11dc47aa	NoUnlockInProgress()	unlockCallback outside flush
0xd92e233d	ZeroAddress()	zero owner, feed or recipient
0x68033bc8	BadFeedAnswer()	feed answer <= 0
0x30cd7471	NotOwner()	owner only levers
0x1f2a2005	ZeroAmount()	ladder deposit or withdraw of 0
0x0a1d52c0	FeedStale()	ladder deposit, deploy, roll while closed
0xceee81e7	DepthGate()	ladder would own too much of the pool
0xa4875a49	CapExceeded()	ladder capQuote
0x6ba4a1c7	DepositTooSmall()	below minDepositQuote

Selector	Error	Where
0x61b708dd	EpochNotOver()	roll before epochEndsAt
0x39996567	InsufficientShares()	withdraw more than held
0x8cec92fc	NothingToDeploy()	deploy with no idle stock
0x9e87fac8	Paused()	ladder deposit or deploy while paused

Wrapping: when a hook callback reverts inside a swap, the PoolManager surfaces it as WrappedError(address hook, bytes4 selector, bytes reason, bytes details) (selector 0x90bfb865) with HookCallFailed() (0xa9e35b2f) as the details. Decode the inner selector against the table above for user facing copy. PoolManager errors you will meet while wiring: PoolAlreadyInitialized() 0x7983c051, HookAddressNotValid(address) 0xe65af6a0, CurrenciesOutOfOrderOrEqual(address,address) 0x6e6c9830, TickSpacingTooLarge(int24) 0xb70024f8.

12. Risks and limitations to show users

- Market hours. Chainlink equity feeds on Robinhood Chain are classed us_equities_24/5. They publish nothing from Friday close to Monday 00:00 UTC and the worst measured gap was about 80 hours (long weekends). Trading never stops; the pool charges the flat stale fee and the ladder refuses deposits, deploys and rolls until the feed resumes. Withdrawals always work.
- Stale fee is symmetric by design. During a closed market the pool price can drift from the eventual Monday mark; the recapture window is what pays LPs back on reopen. Show the closed market chip prominently.
- Issuer powers on the stock token. Robinhood stock tokens are beacon proxies with pause, blocklist and adminBurn. A paused token blocks swaps, deposits and withdrawals of that token for everyone, including the vault. This is an RWA property, not a hook property.
- ERC-8056 multiplier. Balances shown in UI units can change on dividend reinvestment while raw units do not. The vault accounts in raw units; display with the token's uiMultiplier if the Hookr UX shows share values in stock units.
- Oracle dependence. A wrong Chainlink answer misprices the fee and the ladder anchor for as long as it stands. The hook never market sells anything on an oracle move, so the exposure is fee level and rung placement, not a liquidation.
- Ladder is a covered call. Depositors give up upside above the rungs in exchange for quote proceeds and fees. Say so on the stake card.
- Routing allowlist. Hooks with dynamic fees and swap deltas need Uniswap Labs routing review before the Uniswap interface will route through them. Hookr can route via the Universal Router regardless; the review only affects third party front ends that allowlist hooks.
- Not an owned root. Hookr router, quoter and SDK calls against a Smart LP pool revert by design; use the Uniswap V4 Quoter and Universal Router for these pools.

13. Verification ledger

Item	Value
SmartLpHook runtime size	16,116 B (65.6 percent of EIP-170)
SmartLpLadder runtime size	18,780 B (76.4 percent of EIP-170)
SmartLpHookLens runtime size	6,809 B
Compiler	solc 0.8.26, cancun, via IR
Unit tests	63 (41 hook incl. 6 partner leg tests, 22 ladder), all green; whole hooks/ suite 142/142
Mainnet fork tests	6 against the live 4663 PoolManager and the live Chainlink NVDA/USD feed, all green
Live chain audit (post deploy)	Wiring, permissions, poolId, fee path, lens and ladder gates confirmed at head

Fork run facts (block of the run, NVDA/USD answer 230.36): oracle tick 221924; fee away from oracle 30000 pips vs 3000 toward it at a gap of 2469 ticks; weekend replay with the pool parked at tick 220057 against a Monday oracle tick of 221629 moved the gap from 1572 to 918 ticks on the arb swap, the arb output on a plain pool was 10729.721169 USDG vs 10366.013109 USDG on the hook pool, the skim held for LPs was 315.384850 USDG and the protocol share 35.042761 USDG; a full ladder deposit, deploy, roll and withdraw cycle on the live manager cost 4,839,846 gas in total.

What the tests pin: pool must be registered before initialize, vanilla swaps with empty hookData, asymmetric fee by direction, stale fee when the feed is quiet, recapture arming on a fresh round and expiry after the window, skim caps and split, flush donation and protocol payout, ladder NAV math, deposit gates, permissionless deploy and roll, withdraw while paused.

Live head audit (Robinhood Chain 4663, after deploy):

- Hook code size 16116, flags address & 0x3fff == 0x20c4, owner / poolManager / booster / buyback / partnerBps 5000 / partnerRecipient zero match the state file.
- poolConfig and feeParams match defaults; pool initialized at oracle tick 221924; poolId equals keccak256(abi.encode(PoolKey)).
- While the equity feed is stale, quoteFee returns staleFee 10000 both directions and oracleMark.fresh is false. Ladder deposit reverts FeedStale (0x0a1d52c0). flush on an empty accrual reverts NothingToFlush (0xeb694a3c).
- Lens poolView matches hook views. Ladder immutables and config match the launch defaults above. Pool liquidity is still zero until seed (expected); swaps and flush donation need that seed.

14. Integration checklist (for both teams)

Done on StonkBrokers side: PHASE=deploy (with setPartner(0x0, 5000)), PHASE=register for USDG/NVDA, PHASE=ladder for sINVDA. Addresses and txs are in section 3 and hooks/deployments/smart-lp-hook.4663.json. The 2026-10-09 generation (hook 0xFa68...E0C4, lens 0xA7aF...a669, sINVDA 0xeC8A...aFF8, pool 0x17994a68...e573, never seeded) is archived as smart-lp-hook.4663.superseded-2026-10-10b.json.

Still open:

1. StonkBrokers: seed the USDG/NVDA pool with full range liquidity, run one live swap per direction, one flush after a skim, one ladder deposit and deploy once the feed is fresh, and send Hookr the transaction hashes.
2. Hookr: Analyze a Hook against the pinned commit, add the hooklist entry, list the pool keys, wire the lens reads on the pool page, swap through the Universal Router with the V4 Quoter, surface the ladder vault card, and send the contract address that should receive the Hookr fee leg (owner then runs setPartner(<hookrCA>, 5000)).
3. Both: a 48 hour watch over the first weekend close and Monday reopen, reading SwapPriced, RecaptureArmed and Skimmed events.

Seed and first live swaps wait on an explicit StonkBrokers go. Hookr listing can start from this document and the live addresses now.