

Conditionals v2: Liquidity Architecture

Shared-collateral outcome shares, a four-outcome LMSR market maker, and a pooled liquidity vault for oracle-settled conditional prediction markets on Robinhood Chain.

Document	Technical design, revision 1 (proposal, not yet implemented)
Date	23 October 2026
Supersedes	Liquidity model of Conditionals v1 (parimutuel, four pools per market)
v1 deployment	0xc8cc656b71dd77ae1e29b0bc7890edd110e73255, Robinhood Chain, IMD launch 2278
Toolchain	Solidity 0.8.26, via-IR, Foundry; OpenZeppelin; Solady FixedPointMathLib (proposed)

Abstract

Conditionals v1 prices a market *if C, will Y?* through four independent parimutuel pools indexed by (world, side). Each conditional probability therefore requires counterparties on both sides of its world, odds are only determined at `closeTime`, the pool of the unrealised world is idle collateral, and positions cannot be exited before resolution. This document specifies v2, which replaces the pools with (1) four Arrow-Debreu shares over the joint outcome space $\{C, \text{not } C\} \times \{Y, N\}$ backed by a single unit of IMD collateral, (2) a Logarithmic Market Scoring Rule (LMSR) maker per market with worst-case loss $b \ln 4$, and (3) a vault that underwrites the LMSR subsidy across markets under explicit exposure caps. Oracle integration, attestation verification, and the no-admin trust model are carried over from v1 unchanged.

1. Problem statement

Let a market have condition C and outcome Y. v1 holds stakes $S_{w,s}$ for world w in $\{0 = C, 1 = \text{not } C\}$ and side s in $\{\text{YES}, \text{NO}\}$. On resolution with realised world w^* and side s^* , a staker of x on (w^*, s^*) receives:

```
payout = x + x * (S[w*][!s*] - fee) / S[w*][s*], fee = min(0.03 * pot, S[w*][!s*])
stakes in world !w* are refunded; a world with S[w][s] = 0 for some s is refunded
```

The quoted probability for Y given world w is $S_{w,\text{YES}} / (S_{w,\text{YES}} + S_{w,\text{NO}})$. This mechanism has the following liquidity failures:

#	Failure	Mechanism in v1	Consequence
F1	Fragmentation	Liquidity is partitioned into 4 pools; both conditional estimates require all 4 to be non-empty.	~4x the participation of a binary market for equivalent signal.
F2	No ex-ante price	Payout odds depend on S at closeTime; an empty pool has an undefined quote.	First mover bears full price uncertainty; markets fail to bootstrap.
F3	One-sided refund	If $S_{w,s} = 0$ the world is refunded with no transfer.	Early liquidity is uncompensated.
F4	Idle collateral	Stakes in world !w* return 1:1; collateral posted in both worlds is 2x the realised exposure.	Capital efficiency at most 50% for two-world positions.
F5	No exit	Stakes are locked until settlement or the 30-day void.	Duration risk; deters size.

Requirements for v2

- **R1** A well-defined price for every outcome from market creation onward.
- **R2** Continuous entry and exit until c loseTime.
- **R3** One unit of collateral spans both worlds (100% capital efficiency for conditional views).
- **R4** Bounded, ex-ante known loss for the liquidity provider.
- **R5** Solvency for every reachable state; no admin key over funds; same oracle trust assumptions as v1.

2. Choice of mechanism

Property	Parimutuel (v1)	CPMM per world ($x*y=k$)	LMSR, 4 outcomes (v2)
Price at t = 0	Undefined	Defined only after LP deposit	Defined: $p_i = 1/4$
Joint consistency of P(C)	None	None across the two pools	Yes: one simplex over 4 outcomes
LP loss bound	n/a	Unbounded impermanent loss toward 0/1	$b \ln n = 1.386 b$
Exit before close	No	Yes	Yes
On-chain math	Integer	Integer	exp/ln in fixed point
Liquidity source	Traders only	Passive LPs	Subsidy b from vault/creator

The LMSR is selected because it is the only option satisfying R1 and R4 simultaneously, and a single maker over the joint space keeps P(C) and both conditionals mutually consistent, which two independent binary pools do not.

3. Outcome space and shares

Each market defines four outcomes indexed $i = 2w + s$, with $w = 0$ if C occurred and $s = 0$ if Y occurred. A share of outcome i pays 1 IMD if and only if i is the realised index w^* .

i	Symbol	Event	Price interpretation
0	C+Y	C and Y	$P(C \text{ and } Y)$
1	C+N	C and not Y	$P(C \text{ and not } Y)$
2	!C+Y	not C and Y	$P(\text{not } C \text{ and } Y)$
3	!C+N	not C and not Y	$P(\text{not } C \text{ and not } Y)$

```

P(C) = p0 + p1
P(Y | C) = p0 / (p0 + p1)
P(Y | not C) = p2 / (p2 + p3)
effect of C = P(Y | C) - P(Y | not C)

```

- **Complete sets.** `mint(id, n)` takes n IMD and credits n of each share; `redeemSets(id, n)` burns n of each and returns n IMD. Both are fee-free. Since exactly one share pays 1, a complete set is worth exactly 1 IMD in every state, so this enforces $\sum p_i = 1$ by arbitrage and satisfies R3.
- **Internal balances.** Phase 1 records shares in `mapping(id => mapping(address => uint256[4]))`, not as transferable tokens. This keeps per-account cost basis exact for voids (section 7) and removes ERC-1155 receiver hooks and their reentrancy surface from the audit scope.

4. Market maker: LMSR

Let q in $\mathbb{R}^4$ be the maker's net shares sold per outcome (negative when the maker holds inventory bought from traders) and $b > 0$ the liquidity parameter fixed at creation.

```

C(q) = b * ln( sum_i exp(q_i / b) )
p_i(q) = exp(q_i / b) / sum_j exp(q_j / b) (= dC/dq_i, sum p_i = 1)
cost of trade dq = C(q + dq) - C(q) (path independent)
shares for spend x on outcome i, with S = sum_j exp(q_j / b):
k = b * ln( (S * exp(x / b) - S + exp(q_i / b)) / exp(q_i / b) )

```

Solvency and loss bound

The maker is seeded with a subsidy of exactly $C(0) = b \ln 4$. Because traders pay $C(q) - C(0)$ in aggregate, the maker's collateral is always `makerBalance = C(q)`. Since $b \ln \sum \exp(q_i/b) \geq \max_i q_i$, for every reachable q and every realised outcome w^* :

```

makerBalance - q_w* = C(q) - q_w* >= 0 (solvency)
subsidy - residual <= b ln 4 - 0 = 1.386 b (worst-case loss)

```

Shares created by `mint` are backed 1:1 outside the maker, so the total liability for outcome w^* is $q_{w^*} + \text{sets}$ and is covered by `makerBalance + sets`.

Depth versus subsidy

Price impact of a 10 IMD buy of C+Y into a fresh market ($q = 0$, fees excluded):

b (IMD)	Max maker loss $b \ln 4$	Shares bought with 10 IMD	Avg. fill price	$p(C+Y)$ after
20	27.73 IMD	25.59	0.3908	0.5451
50	69.31 IMD	31.71	0.3153	0.3860
100	138.63 IMD	35.11	0.2848	0.3214

Default $b = 20$ IMD (max loss 27.73 IMD); featured markets up to $b = 100$. A minimum of $b = 10$ prevents markets so thin that a single trade saturates the price.

Fixed-point implementation

- All quantities in WAD ($1e18$). `expwad` and `lnwad` from Solady FixedPointMathLib (int256).
- Log-sum-exp with max subtraction: $C(q) = m + b \ln \sum \exp((q_i - m)/b)$, $m = \max q_i$. Every exponent is ≤ 0 , so `expwad` cannot overflow, and the `lnwad` argument lies in $[1e18, 4e18]$.
- Rounding always favours the maker: buy cost rounded up, sell proceeds rounded down, an extra 1 wei per exp/ln call. Fuzz tests compare against an mpmath reference at 50 digits; tolerance $1e-12$ relative.
- Slippage bounds on every trade: `maxCost` on buys, `minProceeds` on sells, `minShares` on router stakes.

5. Conditional stakes (router)

`stakeConditional(id, w, s, x, minShares)` reproduces the v1 position (paid in world w on side s , refunded if world $!w$ occurs) at a price fixed at entry:

```
1. mint(id, x) // x of each share, cost x IMD
2. keep x of share (2(1-w)+0) and x of share (2(1-w)+1) // pays x if world !w: the refund leg
3. sell x of share (2w + (1-s)) to the maker -> proceeds r
4. buy k of share (2w + s) with r // closed form from section 4
=> payoff: x + k if (w, s) occurs; x if world !w occurs; 0 if (w, !s) occurs
```

Worked example ($b = 20$, fresh market, $x = 10$ IMD on YES in world C, fees excluded): step 3 returns $r = 2.0710$ IMD, step 4 buys $k = 6.6359$ C+Y. Payoff vector: **16.6359** IMD if C and Y, **10** IMD if not C, **0** if C and not Y, i.e. an average entry at $P(Y | C) = 0.6011$. Post-trade quotes: $p = [0.3484, 0.1516, 0.2500, 0.2500]$, $P(Y | C) = 0.6967$, $P(C)$ unchanged at 0.5000.

6. LiquidityVault

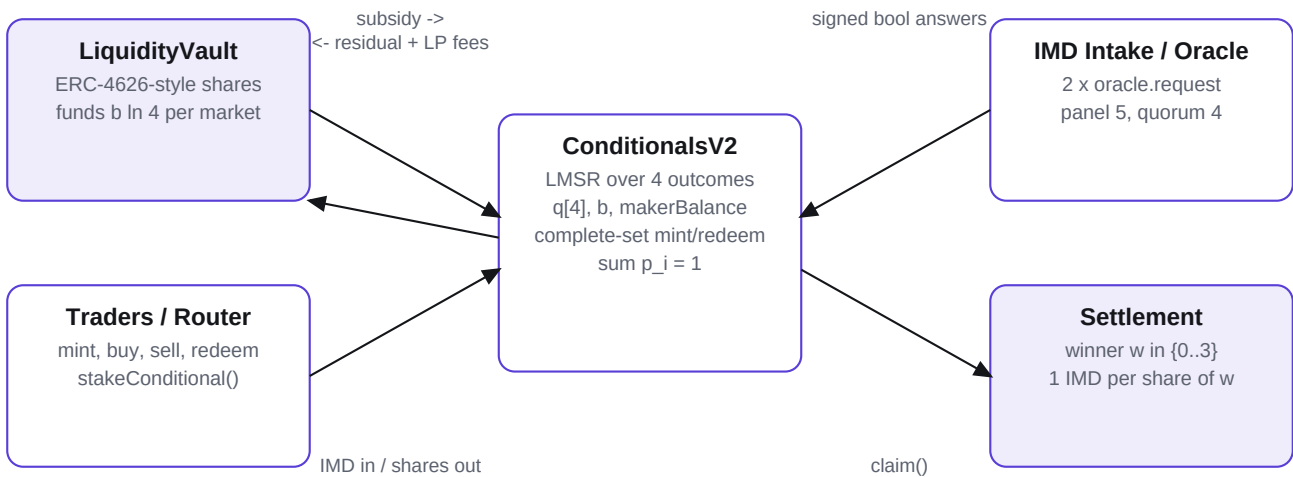


Figure 1. Collateral flows in v2.

The vault is a share-accounting contract holding IMD. It allocates subsidies to markets that request them and receives each market's residual $C(q) - q_{w^*}$ plus the LP fee stream at settlement.

- **Exposure caps.** $\text{subsidy}_m \leq 5\%$ of NAV per market; sum of open subsidies $\leq 60\%$ of NAV. A request that would breach either cap reverts.
- **Creator first loss.** A market's subsidy is $\text{creatorSeed} + \text{vaultPortion}$ with $\text{creatorSeed} \geq 10\%$ of the total. Realised loss $L = \text{subsidy} - \text{residual}$ is applied to `creatorSeed` first. LP fees are split pro rata with the creator's slice weighted $2x$.
- **Valuation.** Two marks per open market: `book` = committed subsidy; `floor` = $\text{makerBalance} - \max_i q_i$ (the guaranteed residual). Deposits mint at $\max(\text{NAV}_{\text{book}}, \text{NAV}_{\text{floor}})$ and withdrawals burn at $\min(\text{NAV}_{\text{book}}, \text{NAV}_{\text{floor}})$, so an LP cannot exit ahead of a loss that is already locked in.

- **Withdrawals** are paid from idle IMD immediately; any excess is queued and filled FIFO as markets settle.

7. Fees, settlement and void

Flow	v1	v2
Fee base	3% of winning pot, capped at losing pool	Fee on IMD notional of every LMSR buy/sell; mint, redeemSets and claim are fee-free
Base rate	300 bps	200 bps, linear ramp to 400 bps over the final 10% of [createdAt, closeTime]
Split	200 bps burn / 100 bps builder	100 LP / 67 burn / 33 builder (per 200; ramp scales proportionally)
Burn sink	0x...dEaD	0x...dEaD (IMD reverts on transfers to address(0))
Blocked recipient	Credited to owed[], withdrawOwed()	Unchanged

Settlement. Oracle handling is identical to v1: two oracle.request questions via the Intake, EIP-712 domain *IdentityMD Oracle* v2, attester EOA or ERC-1271, binding to the stored question hash, panel ≥ 5 , agreed ≥ 4 , expiry and 5-minute issuedAt tolerance. With answers (c, y): $w^* = 2(1 - c) + (1 - y)$. Each share of w^* redeems for 1 IMD; the maker residual $C(q) - q_{w^*}$ and the creator's unspent oracle budget are returned.

Void. After RESOLUTION_PERIOD (30 days) without both answers, a uniform 1/4 payout would transfer value from buyers at high prices to buyers at low ones, so v2 refunds cost basis instead. Each account tracks $\text{int256 net} = \text{IMD in} - \text{IMD out}$ for the market (mint, buy, sell, redeem, router). On void, accounts with $\text{net} > 0$ are refunded from the market balance, pro rata if it is insufficient; the subsidy funder is paid last. The vault's void exposure is therefore bounded by the subsidy, as in the settled case.

8. Contract interface (draft)

```

struct Market {
    // v1 fields: creator, condition, outcome, rules, closeTime, resolveTime, config, answers, asks
    int256[4] q;           // maker net shares sold per outcome (WAD)
    uint128 b;             // LMSR liquidity parameter (WAD)
    uint128 makerBalance; // == C(q) up to rounding in the maker's favour
    uint128 sets;          // complete sets outstanding
    uint128 subsidy;       // b * ln 4 at creation
    address funder;        // vault or creator
    uint8 state;           // Open, Closed, Asked, Settled, Voided
    uint8 winner;          // w* in {0..3}
}
mapping(uint256 => mapping(address => uint256[4])) internal _shares;
mapping(uint256 => mapping(address => int256)) internal _net;

function createMarket(string calldata condition, string calldata outcome, string calldata rules,
    uint64 closeTime, uint64 resolveTime, uint256 maxBudget, uint128 b, bool useVault) returns (uint256 id);
function mint(uint256 id, uint256 sets) external;
function redeemSets(uint256 id, uint256 sets) external;
function buy(uint256 id, uint8 i, uint256 shares, uint256 maxCost) external returns (uint256 cost);
function sell(uint256 id, uint8 i, uint256 shares, uint256 minProceeds) external returns (uint256 proceeds);
function stakeConditional(uint256 id, uint8 w, uint8 s, uint256 amount, uint256 minShares)
    external returns (uint256 shares);
function requestResolution(uint256 id) external returns (uint256 asked); // as v1
function onOracleResult(bytes32 requestId, Attestation calldata a, bytes calldata sig) external; // as v1
function onFailure(bytes32, uint8, bytes32, uint16, uint16, bytes calldata) external; // as v1
function settle(uint256 id) external;
function claim(uint256 id) external returns (uint256 amount);
function voidMarket(uint256 id) external;
function prices(uint256 id) external view returns (uint256[4] memory wad);
function conditionals(uint256 id) external view returns (uint256 pYesIfC, uint256 pYesIfNotC);
function quoteBuy(uint256 id, uint8 i, uint256 shares) external view returns (uint256 cost, uint256 fee);

```

9. Invariants and verification plan

ID	Invariant	Test
I1	IMD.balanceOf(this) >= sum over markets of (makerBalance + sets + unrefunded void credit) + sum owed	Foundry invariant, random actors
I2	makerBalance >= max _i q _i for every open market (solvency in all four states)	Invariant + fuzz
I3	For each i: sum over accounts of shares[i] == q _i + sets	Invariant
I4	sum p _i - 1e18 <= 4 wei	Fuzz on random q
I5	Round trip buy(k) then sell(k) never returns more than was paid	Fuzz
I6	Total fees <= 400 bps of notional; burn : builder = 2 : 1	Invariant
I7	Void refunds <= market balance; funder paid last; sum refunds + funder = balance	Unit + fuzz
I8	Callback gas of onOracleResult < 200,000 (Intake limit)	Unit, as in v1

- Differential tests of C(q), p(q) and the closed-form buy against a 50-digit Python reference.
- Fork test against the live Robinhood Chain Intake, carried over from v1's `ConditionalsFork.t.sol`, plus the protocol oracle conformance vector.
- Gas: per-trade cost to be measured; target < 150k gas for buy/sell and < 250k for stakeConditional.

10. Threat model

Threat	Mitigation
Informed trading after C or Y becomes public	Loss capped at b ln 4 per market and 5% of NAV; creator first loss; closeTime must precede the earliest public resolution (UI-enforced, as v1); late-fee ramp to 400 bps.
Fixed-point error exploited through many small trades	Maker-favoured rounding per operation; minimum trade 0.001 IMD; I5 fuzzing.
Front-running / sandwiching	maxCost, minProceeds and minShares on every entry point bound a sandwich's extraction to the victim's own slippage tolerance; the 2% fee on both attacker legs makes small sandwiches unprofitable.
Vault bank run	60% commitment cap; floor-priced withdrawals; FIFO queue filled at settlement.
Oracle refusal or silence	v1 retry (re-ask after failure or after 1 day) and 30-day void with cost-basis refunds.
Reentrancy	No token callbacks in phase 1 (internal balances); nonReentrant on all IMD-moving functions; checks-effects-interactions.
Oracle price increase	Per-market price snapshot and maxBudget, as v1.

11. Parameters

Parameter	Value	Mutable
FEE_BPS / LATE_FEE_BPS	200 / 400	Constant
LP_BPS / BURN_BPS / BUILDER_BPS	100 / 67 / 33	Constant
MIN_B / DEFAULT_B / MAX_B	10 / 20 / 100 IMD	Constant
MIN_TRADE	0.001 IMD (1e15), as v1 MIN_STAKE	Constant
VAULT_MARKET_CAP / VAULT_TOTAL_CAP	500 / 6000 bps of NAV	Constant
CREATOR_FIRST_LOSS	>= 1000 bps of subsidy	Constant
RESOLUTION_PERIOD	30 days, as v1	Constant
Oracle config	Intake, signer, price, panel 5, quorum 4, action id	Owner, 3-day timelock (as v1)