$MORT: An Immutable, Admin-Free Fee Engine for Community-Directed On-Chain Asset Accumulation

Mort

A dead boomer who buys the dip and never sells.

Robinhood Chain (EVM L2, chain id 4663) • heymort.co

August 2026

Abstract. $MORT is a memetic token whose economic substance is a fully on-chain, upgrade-free machine that converts trading fees into productive assets and returns them to holders. Every unit of creator fee earned on the launchpad is claimed, split by a fixed 50/30/20 rule, and routed through three independent sinks: a daily community-voted stock purchase distributed pro-rata to holders, a market-buy-and-burn of the token itself, and a team payout. The system contains no owner, no upgrade proxy, and no privileged control over the value flow. This paper documents the contract architecture, the time-weighted voting mechanism that directs the daily buy, the oracle-bounded and MEV-resistant execution path, and the trust model that follows from immutability. We argue that the design achieves credible neutrality: once deployed, the rules governing money movement cannot be altered by any party, including the authors.

1 Introduction

Most memetic tokens couple a strong narrative to a weak or absent mechanism: value accrual, if any, depends on discretionary treasury decisions that a team can revise, delay, or reverse. $MORT inverts this relationship. The narrative (a deceased buy-and-hold investor who accumulates forever and never sells) is expressed literally in code as an autonomous fee engine with no escape hatch for its operators.

The token launches on Pons, a bonding-curve launchpad on Robinhood Chain, an EVM-compatible Ethereum layer-2 built on the Arbitrum Orbit stack with ETH as the gas token. Trades pay a fee; the creator's share of that fee is the engine's only fuel. The engine's job is to transform that ETH stream into three outcomes that are decided at deploy time and never afterward.

The design goals are:

- **Immutability of money flow.** No contract that holds or moves fee value may have an owner, a setter, or an upgrade path.
- **Credible neutrality.** The split, the sinks, and the default behavior are fixed constants readable by anyone.
- **Permissionless operation.** Every step in the daily loop can be triggered by any address; the team runs a keeper bot only for convenience, never as a trusted party.
- **Resistance to extraction.** On-chain execution is bounded by independent price oracles and paced to blunt sandwiching and front-running.

The remainder of this paper describes the components that realize these goals and the security properties that follow.

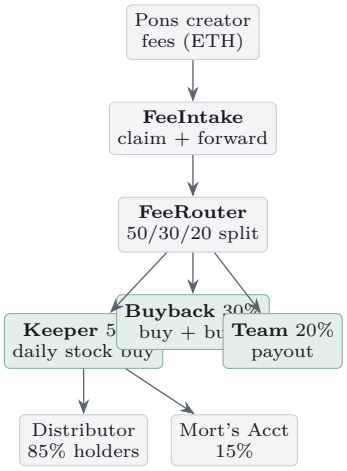

Figure 1: Fee flow. A single intake feeds an immutable split; each sink is an independent, admin-less accumulator. The keeper further routes its daily stock purchase 85/15 to holders and the vault.

2 System Architecture

The engine is a set of small, single-purpose contracts connected in a directed flow from a single intake to three terminal sinks. Contracts that touch value are *admin-less*; the only contracts carrying a narrow privileged action are append-only registries that can never redirect funds.

2.1 Intake

The `FeeIntake` is the front door and the only address registered as the launchpad's creator-fee recipient. Fees are not pushed; they accrue in the token's locked liquidity position and must be claimed. The intake exposes a permissionless `claimAndForward()` so any caller (in practice, a scheduled keeper) pulls the balance and forwards every

Share	Sink	Outcome
50%	Keeper → Distributor	voted stock to holders
30%	Buyback	market-buy $MORT, burn
20%	Team Splitter	fixed per-member shares

Table 1: The immutable 50/30/20 allocation of every fee.

wei to the router. WETH claims are auto-unwrapped, and a plain `receive()` fallback accepts direct transfers so the forward path never depends on a specific escrow ABI. The only privileged action is an operations-only rescue of stray non-flow ERC-20s, which by construction can never touch ETH or WETH. The fee flow itself is therefore unstoppable and unredirectable.

2.2 The split

The `FeeRouter` encodes the constants `STONKS_BPS`= 5000, `BUYBACK_BPS`= 3000, and `TEAM_BPS`= 2000 against a denominator of 10,000. There are no setters and no upgrade path; the three sink addresses and the split are fixed forever at deploy. Deposit and split are deliberately separate steps: depositing can never be blocked by a sink, while `split()` is permissionless and atomic, so all three sends succeed or the whole call reverts and ETH remains in the router. This atomicity imposes a hard invariant on every sink: its `receive()` must be an unconditional accumulator with no external calls and no revert path, since a single permanently reverting sink would otherwise strand all future fees. All active market operations (buys, burns) happen in later keeper steps, never inside a sink.

3 The Daily Loop: Time-Weighted Voting

Half of every fee funds a stock the community selects each day. The selection mechanism, `MortYell`, is the analytic core of the system.

3.1 Voting rule

Between 00:00 and 20:00 UTC, holders name a ticker from an admitted list. The ticker with the greatest weight wins the entire day's purchase, executed at 21:00. A tie breaks to whichever ticker reached its total first. If nobody votes, the engine buys the default index (QQQ): a dead boomer defaulting to the index is the joke, but the weighting math is not.

3.2 Voting weight as an integral

Voting power is a true time-weighted balance (TWAB), not a snapshot. To vote, a holder deposits $MORT into the contract; power accrues as the exact integral of the deposited balance from the moment of the vote to the 20:00 close. For a voter holding balance $b(t)$ over the window $[t_v, T_{\text{close}}]$, weight is

$$w \;=\; \int_{t_v}^{T_{\text{close}}} b(t)\,dt \qquad \text{[token-seconds]}.$$

Because power is measured in token-seconds, three properties follow directly from the mathematics:

- **Flash-loan immunity.** A deposit and withdrawal in the same block span zero time and integrate to zero weight.
- **Whale-latency penalty.** A large balance arriving seconds before close contributes almost nothing.
- **No double-voting.** Custody, not `balanceOf` sampling, backs the integral, so the same tokens can never vote from two wallets.

Custody is essential: the launchpad token exposes no transfer hooks, so a sampled external balance would be gameable between permissionless finalization pages. Keeping the integral over deposited balances closes that gap. Deposits persist across days (stake once, vote daily) and withdrawal is instant at any time; accrual simply stops. Nothing is ever paid out of the stake pool, so deposits map one-to-one to withdrawals.

3.3 Finalization

After the close, finalization is permissionless and paginated and operates purely on internal accounting with no external calls, so page timing cannot be gamed. The sealed per-voter weights become the day's TWAB set, which the distributor reuses directly. The voting set and the payout set are therefore identical, by the same weights: a day's stock is shared exactly among those who held and voted through it, eliminating dividend capture by a last-minute buyer and denying rewards to holders who never voted.

4 Execution and MEV Resistance

The `MortKeeper` holds the 50% stock cut and executes the day in a permissionless three-step sequence that any address can drive.

1. `startDay`: after the day is sealed, lock in the winning ticker (or the default on silence) from the append-only registry.
2. `buyChunk`: swap a slice of ETH for the stock. Repeated calls over several minutes form a TWAP execution.
3. `settleDay`: route the acquired stock 85% to the distributor and 15% to Mort's Account.

4.1 Oracle-bounded buys

Each chunk independently enforces a minimum output derived from a Chainlink USD feed, so no single chunk can be sandwiched past the slippage bound. Slippage is capped by a hard constant (`MAX_SLIPPAGE_BPS` = 2000, i.e. 20%) that a deploy cannot exceed. A corporate-action pause on the asset's oracle blocks the buy rather than trading on a frozen price: a skipped day is strictly better for the engine than a wrong price. Only one day is in flight at a time, keeping ETH accounting unambiguous; the loop runs even on a zero-fee day, buying nothing and settling cleanly.

4.2 Pacing floors

Because `buyChunk` is permissionless, the contract fixes floors that a deploy cannot disable: a minimum interval between chunks (`MIN_CHUNK_INTERVAL` = 15s) and a minimum chunk size (a fraction of the per-day cap). Without a size floor, a one-wei griefer could consume one pacing slot per interval and starve an honest keeper's TWAP. The bot additionally randomizes its start to blunt predictable front-running. The keeper has no admin over funds: it can only push stock to the immutable distributor and vault, never to an arbitrary address.

4.3 Pricing a token without a native feed

The 30% buyback must bound its own $MORT purchases, but $MORT has no Chainlink feed. The `Buyback` contract derives its minimum output on-chain from the $MORT/ETH pool's time-weighted average price, never from a caller-supplied number that a permissionless trigger could set to zero. An attacker who pushes pool spot away from its average to skim the buy trips the bound and the swap reverts. The TWAP window is floored (`MIN_TWAP_WINDOW` = 600s) so that moving the average is expensive. For the launchpad token, which also lacks a feed, a dedicated `V3TwapUsdFeed` synthesizes a Chainlink-shaped USD price as

$$p_{\text{usd}} \;=\; \frac{\text{TWAP}(\text{token/WETH}) \times \text{Chainlink}(\text{ETH/USD})}{1 - f_{\text{pool}}},$$

fee-adjusted by the pool fee f_{pool} so the feed quotes the price a buyer effectively pays. Bounding buys against an unadjusted mid would silently spend the entire slippage budget on pool fees.

5 Distribution and the Vault

5.1 Distributor

The `StonkDistributor` pays the 85% holder cut of each daily buy pro-rata by the sealed TWAB set. It is pull-claim and gas-bounded per voter: the keeper opens a day with the token and amount deposited, and each voter claims their slice once,

$$\text{payout}_i \;=\; \text{amount} \times \frac{w_i}{\sum_j w_j}.$$

Rounding dust (the sum of floors falling below the total) simply rests in the contract. Zero-weight days never reach the distributor: the keeper routes those fully to the vault, and `openDay` refuses a day with no total weight, making division-by-zero and orphaned pots both impossible. The keeper can only add days; it can never claw back or redirect what voters are owed.

5.2 Mort's Account

The retained 15% accumulates in `MortsAccount`, a one-way vault with no withdraw, no sell, no redeem, no owner, and no upgrade path. The absence of an ordinary exit is the point: with no way to pay anyone, the vault stays clear of redeemable-fund territory. Mort is dead and never sells; his account behaves the same way. A single constrained escape hatch can move holdings only to one fixed, immutable beneficiary Safe and only after a timelock. It cannot redirect, sell, or redeem; regardless of caller or argument, funds reach exactly one pre-committed address. Because the destination is fixed, the off-chain unlock secret (stored only as a hash) is leak-proof and needs no commit-reveal. The one permissionless write is an additive token registration so dashboards can enumerate holdings; valuation happens off-chain from these balances, the stateless `StockValuer` views, and Chainlink feeds.

6 Extensibility Without Admin Keys

New assets must be admissible after launch without granting anyone power over the value flow. Three append-only structures provide this.

- **Ticker registry.** The single source of truth for the admitted-asset list (tokenized stocks and blue-chip crypto). `MortYell` and `MortKeeper` both read it, so they can never drift out of index alignment. Additions only: an index is never mutated or removed, so a listed ticker can never be replaced by a scam. Additions are timelocked, and proposing one cannot touch funds, the split, the sinks, or the default. Tokenized stocks are detected by their ERC-8056 surface and keep the corporate-action pause; plain assets skip it.
- **Swap adapters.** A thin `DispatcherExecutor` holds an append-only token-to-adapter table and forwards each buy to the immutable, single-venue executor for that asset (a Uniswap v3 executor today, a v4 child for the buyback, others later). One adapter per token, forever; no admin can redirect a listed token's venue. A keeper-level executor setter was explicitly rejected as the first redirect lever in the money layer.
- **Team splitter.** The 20% team cut is divided by immutable per-member basis-point shares agreed in writing before launch, pull-based, so one member that cannot receive ETH blocks only their own claim.

Each swap path is validated structurally at registration (byte-length and endpoint checks on the encoded route), so a fat-fingered path cannot be registered at all, and an admitted-but-unbuyable asset is impossible.

7 Holder Bootstrapping

To seed a wide, credibly-decentralized holder set at launch, $MORT used a custody-free batch distributor (`MortAirdrop`) following the Disperse pattern: one approval plus one `disperseToken()` fans a batch out atomically, so a mid-batch failure can never half-pay a list. The contract never holds a balance and has no owner or state. Distribution ran in escalating waves (an initial visibility push, a capped-grant expansion, and a final large wave of 2×10^8 $MORT across 28,556 unique wallets), with grant sizes assigned by an integer largest-remainder (Hamilton) apportionment of scores so that per-wallet amounts sum exactly to the intended total. Because voting weight is time-weighted and custodial, this broad initial distribu-

tion translates directly into a broad, sybil-resistant voting base rather than a one-block snapshot that a whale could dominate.

8 Trust Model and Security Properties

The system's guarantees are structural rather than procedural. They hold because the relevant contracts have no privileged code, not because an operator promises to behave.

- **No admin over value.** Intake, router, buyback, distributor, team splitter, and vault expose no owner, no setter, and no upgrade proxy touching ETH or holder funds.
- **Unstoppable flow.** Every step (claim, split, start, buy, settle, claim payout, burn) is permissionless. The team's keeper is a convenience, not a trust assumption; a stalled bot delays but cannot capture.
- **Bounded execution.** All market buys enforce an independent oracle-derived minimum output under a hard slippage ceiling, with TWAP windows and pacing floors that a deploy cannot weaken.
- **Append-only growth.** The asset list and swap routes can only grow, never mutate, and every addition is timelocked and fund-neutral.
- **Atomic safety.** The split is all-or-nothing; airdrop batches are all-or-nothing; the distributor cannot divide by zero or orphan a pot.

Residual assumptions are explicit and narrow: the launchpad's locked liquidity and fee accounting, the correctness and liveness of the consumed Chainlink feeds, and a launch-time review that every value sink's `receive()` is a pure accumulator. Each is documented rather than obscured. The contracts were deployed and their engine reviewed prior to launch; the liquidity position is permanently locked.

9 Conclusion

$MORT demonstrates that a memetic token can carry a real, self-contained economic mechanism whose rules no party can change after deployment. By fixing the split, the sinks, and the defaults as constants, measuring governance as an integral of custodied stake over time, and bounding every on-chain buy against an independent oracle, the engine turns a narrative about a buyer who never sells into an autonomous, credibly-neutral system. The result is a machine whose most important property is what it *cannot* do: it cannot be paused, redirected, upgraded, or drained by anyone, including its authors.

This document describes a technical system and is not financial advice or an offer of any security. $MORT is a memetic asset; Mort's Account holds assets and, by design, cannot pay them out. Deployed contract addresses and source are published at heymort.co.