Common Protocol:

Introducing the COMMON Token and Coordination Layer

Draft Version 0.10

Dillon Chen
dillon@common.xyz

October 21, 2025

Abstract

Common is a coordination layer for multi-agent systems. Programmable loops create **markets around events**—both internal decision-making processes (governance votes, milestone approvals, quality thresholds) and external launch events (token deployments, content releases, feature rollouts). By making these events tradable and conditional, participants express intent, provide liquidity, and coordinate outcomes through price discovery rather than pure voting or manual allocation.

Markets minimize coordination costs when boundaries are well-defined and transaction costs are low. Programmable loops define these boundaries: each loop packages the event, the market structure, the settlement logic, and the fee routing into a composable primitive. Humans and AI agents transact across these boundaries as economic participants, coordinating through price signals rather than centralized allocation.

The long-term goal is a coordination layer for DAOs where inputs and outputs are priced by markets, enabling truly autonomous organizations. This paper introduces the COMMON token, the loop primitives, and initial implementations (launches, contests, community stake). The protocol is MCP-native; emissions and veC govern action budgets; futarchy will govern the token.

Contents

1 Why Loops, Why Now

The top-ranked user on HackerOne is an automated agent; Truth Terminal votes onchain; Grok trades based on chain data. These agents don't navigate org charts or wait for committee approvals—they respond to prices, incentives, and executable contracts. As agents proliferate, coordination shifts from *asking permission* to *transacting at market rates*. Crypto rails enable this: every action can be priced, every contribution can be settled, and every decision can be conditional on market signals. However, these still currently fail in multi-agent scenarios.

Markets are the coordination primitive. Intelligence is already too cheap to meter; coordination isn't. The bottleneck is **coordinating action across boundaries**: deciding what to build, who pays for it, when to ship, and how to split the upside. Markets solve this through price discovery. Agents operating in the real economy need **ground truth**—accurate, real-time signals about what is economically valuable. Modern machine learning relies on crafting RL environments to generate training data; loops-as-markets enable this for economically valuable tasks. Humans actively price what matters through trading, voting with capital, and staking on outcomes. Agents train on these real-world price signals rather than simulated environments.

DAOs need markets for decisions. Today's DAOs coordinate through forums, multisigs, and slow governance votes. There are no **markets for decisions**. A DAO cannot easily ask: "What is the market price of shipping feature X?" or "What odds does the market give that this grant will succeed?" Without price signals, DAOs overpay for low-impact work and underfund high-impact experiments.

Loops create markets around every event. The Common Protocol provides a simple primitive—*Loops*—that turns any event (internal governance decision, external token launch, milestone approval, content release) into a market. Each loop defines:

- **What:** the event or action (launch a token, approve a proposal, reward top content)
- **How:** the market structure (bonding curve, prediction market, contest with prizes)
- **When:** the epoch or trigger (weekly, on-demand, conditional on TWAP)
- **Who pays:** fee routing and settlement (protocol, namespace, voters, LPs)

Loops are composable: a contest loop can feed a launchpad loop; a launchpad with conditional hooks can gate a governance loop; emissions gauges can direct rewards to any loop. Humans and agents call loops via MCP, transact across loop boundaries, and coordinate through price signals rather than committees.

2 Protocol Architecture

2.1 Loop Definition & Phases

A **loop** is a programmable state machine that moves through three phases:

1. **Initialization** — configure admin, allowed *intents*, epoch policy, ranking strategy, hooks, and fee routing.
2. **Operation** — users submit revocable *intents* (deposits of behavior) recorded by the loop.
3. **Finalization** — rank intents & depositors, settle rewards, and dispatch post-finalize actions.

Loops are smart accounts that can hold assets and call contracts. *Intents* are revocable until finalization (users can add, modify, or remove them). Each intent accrues *points* (currently off-chain, with future on-chain support) enabling cross-loop composition.

2.2 Loop Taxonomy: Launch, Earn, Govern

All protocol activity falls into three loop families, each backed by implemented contracts:

Family	Purpose	Key Contracts & Mechanisms
Launch	Create surfaces and start programs	**Namespace:** `NamespaceFactory` deploys ERC1967 proxies with admin ID 0, `FeeManager`. **Launchpad:** `TokenCommunityManager` deploys tokens, governance, Namespace. **Contest setup:** `ContestFactoryUtils` deploys single/recurring contests.
Earn	Generate and distribute value per epoch or action	**Contests:** `ContestGovernor` ranks content, pays winners/voters, skims fees. **Community Stake:** `CommunityStake` bonding curve with fee splits. **Fee routing:** `FeeManager`, `ReferralFeeManager`.
Govern	Steer budgets and parameters over time	**Emissions:** `MintAuthority` → `EmissionsManager` → `GaugeManager`/`StakingPool`; veC votes direct budgets. **DAO proposals:** `VoteGovernance` for parameter changes. **Futarchy:** Launchpad + conditional vaults + CLMM pools; TWAP decisions.

2.3 Implemented Loop Surfaces

Each loop surface implements a common schema: **Admin** (who configures), **Intents** (what users submit), **Epoch** (when finalization occurs), **Ranking** (how outcomes are scored), **Hooks**

(validation and side effects), **Fees** (value routing), and **Finalize** (settlement logic). The table below shows how each surface instantiates this pattern.

Surface	Admin	Intents	Epoch	Ranking	Hooks	Fees / Finalize
Namespace	ID 0 holder(s)	mintId, burnId, transfer	Registry (lazy)	n/a	NS hooks on mint/xfer	FeeManager bound; no finalize
Contest (recurring)	NS admin	addContent, voteContent	Recurring (time)	Strategy-weighted; shares sum 100	Content, claim, strategy	Fee skim; finalize on rollover; pay winners/voters
Community Stake	Factory + mods	buyStake, sellStake	Continuous (lazy)	Bonding curve; no ranking	n/a	Split protocol/NS fees; trade-level settle
Launchpad	LP owner	executeLaunchAction	One-off	n/a	n/a	Finalize deploys gov/NS; no routing
Emissions (gov)	Governor/ auth	mint()	Recurring (weekly)	n/a	n/a	Split to EM + DAO; EM to gauges/staking

Loops share infrastructure (`FeeManager`, `NamespaceFactory`, points) and interoperate: Namespace IDs gate Contest participation, Contest results feed Launchpad tokens, and Emissions gauges direct rewards to any loop. This composability enables multi-loop workflows and new surfaces to be added by implementing the same schema.

3 Example Loop Extensions: Composable Primitives in Action

This section demonstrates how Common's loop primitives compose into different coordination mechanisms. We begin with the existing Launchpad and Post Coin surfaces implemented in `common-protocol`, then explore how these can be extended via hooks for more advanced use cases like conditional launches and futarchy governance.

3.1 Launchpad: Extensible Token Launch Infrastructure

The `Launchpad` contract is designed for extensibility via **launch action hooks** that execute custom logic during token deployment.

3.1.1 Core Hook Architecture

ILaunchActionHook. Executes logic post-token creation but pre-bonding curve registration. Can modify the initial distribution:

```
interface ILaunchActionHook {
  struct LaunchActionResponse {
    uint256[] shares;
    address[] holders;
  }

  function executeLaunchAction(
    string memory name,
    string memory symbol,
```

```
    uint256[] memory shares,
    address[] memory holders,
    uint256 totalSupply,
    address tokenAddress,
    address sender
  ) external returns (LaunchActionResponse memory);
}
```

Use cases:

- **NamespaceLaunchAction:** Auto-deploys a Namespace + mints admin ID to the launcher.
- **ConditionalVaultHook:** Escrows a portion of the supply in a conditional vault (e.g., for futarchy or milestone-based releases).
- **VestingHook:** Locks founder/team allocations in a vesting contract before bonding curve registration.

ICurveActionHook. Executes logic after each buy/sell transaction on the bonding curve:

```
interface ICurveActionHook {
  function postBuyHook(
    address token, uint256 tokenAmount,
    uint256 cost, uint256 fee
  ) external;

  function postSellHook(
    address token, uint256 tokenAmount,
    uint256 proceeds, uint256 fee
  ) external;
}
```

Use cases:

- **ReferralActionHook:** Distributes referral fees to the referrer + namespace on each trade.
- **LoyaltyPointsHook:** Awards off-chain or on-chain points per buy/sell transaction.
- **DynamicFeeHook:** Adjusts protocol fees based on volume, time-of-day, or holder count.

3.1.2 Example 1: Standard Memecoin Launch (Bonding Curve)

Scenario: User launches a token with no custom logic—just a standard bonding curve.

Flow:

1. Call `launchTokenWithLiquidity` with `launchAction = address(0)`.
2. `Launchpad` deploys token; 100% supply goes to `LPBondingCurve`.
3. Users buy/sell; fees route to `FeeManager`.
4. When curve graduates, liquidity migrates to canonical COMMON↔TKN pool.

Value to COMMON**:** Trading fees → veC voters + DAO. Canonical liquidity ensures long-term fee capture.

3.1.3 Example 2: Post Coins (Content-Linked Tokens)

Scenario: Creator wants to launch a token tied to specific content (post, video, etc.).

Flow:

1. Creator submits content to Contest loop; if it wins/ranks, they trigger `launchTokenWithLiquidity`.
2. `ContentLinkHook` stores content hash in token metadata.
3. Bonding curve seeds with contest winnings or creator funds.
4. Fans buy Post Coin; fees flow to creator's namespace + protocol.

Value to COMMON**:** Contest fees + trading fees → veC voters + DAO. Post Coins eligible for emissions if canonical.

3.1.4 Example 3: Fundraising Milestone Launch (Kickstarter-style)

Scenario: DAO launches a token that only releases if fundraising goal is met (e.g., $100k). Otherwise, contributors get refunds.

Hook: `FundraisingVaultLaunchAction` escrows entire token supply in vault with goal + deadline. Vault becomes sole holder until milestone check.

Flow:

1. Launcher calls `launchTokenWithLiquidity` with `FundraisingVaultLaunchAction`, specifying token allocation (contributors, DAO/founder, bonding curve).
2. Hook escrows entire token supply in vault. Contributors deposit USDC/ETH and receive receipt tokens.
3. After deadline, anyone calls `vault.checkMilestone()`:
   - **SUCCESS:** Contributors redeem pro rata for tokens at fixed price; bonding curve activates with liquidity; DAO/founder allocation released.
   - **FAIL:** Contributors get full refunds; escrowed tokens return to founder.

Value to COMMON**:** If milestone met: trading fees from bonding curve + canonical liquidity. If milestone fails: no fees (contributors refunded).

Beyond launches: Same mechanism works for milestone-based funding of ongoing work (PR completion, audit passage, feature delivery). Transparent, verifiable fundraising with automatic refunds.

3.1.5 Example 4: Futarchy AMM — Conditional Governance via Prediction Markets

Scenario: Proposal becomes two tradeable governance options (PASS/FAIL). Market price signal (TWAP) decides outcome, not ballots.

Flow:

1. Deposit token or USDC → mint `p/` and `f/` claims (pay 1 if outcome happens).

2. Trade in PASS/FAIL pools or place single-sided limit ranges.

3. At window end, higher TWAP wins; winner claims redeem 1:1, loser = 0.

Why it matters: Leverage conviction with bounded risk (buy <1, redeem 1). Revives "dead" tokens with fresh liquidity. Open participation (USDC or token); LPs earn fees.

Integration: `FutarchyLaunchAction` hook. If PASS: bonding curve activates, canonical liquidity initializes. If FAIL: tokens return to founder, buyers refunded. All trading fees → veC holders + DAO.

Components:

- **Conditional Vault:** Deposit U → mint `pU` + `fU`; winner redeems 1:1, loser gets 0.
- **CLMM Pools:** Three pools per proposal (PASS, FAIL, ODDS). ODDS pool TWAP encodes probability; execute PASS if $\pi \geq 0.5 + \delta$ (typical $\delta = 2$–5%).
- **Decision Oracle:** `FutarchyGovernor` + `DecisionOracle` manage lifecycle and TWAP-based decisions. Risk controls: TWAP windows, circuit breakers, fee tiers.

3.2 Summary: Loops Feeding Loops

Hooks enable custom logic without forking core contracts. Contest loops feed Launchpad loops (content → token → fees). Futarchy adds prediction markets for governance decisions. Every extension drives value to COMMON via trading fees, canonical liquidity requirements, and emissions eligibility. Builders innovate on UX; protocol captures value via fees and emissions discipline.

4 Token Utility & Economics

4.1 Tokenomics Summary

- **Total Initial Supply:** 10 billion COMMON tokens
- **Community Allocation at Launch:** 47.5%
  - **Community DAO (4.4%):** 50% unlocked at launch, remainder vested, controlled via DAO vote, initially held via Security Multisig until emissions and governance are enabled.
  - **Retroactive Rewards (4.2%):** Distributed at launch and during the first year via historical, NFT and exchange boosts.
  - **Rewards & Incentives (13.9%):** Vested over 4 years (40/30/20/10), distributed to community by foundation in different programs.
  - **Foundation (25.0%):** 50% unlocked at launch, rest vested over 4 years.
- **Investors & Contributors (∼52.5%):** 1-year cliff, 3 year vest thereafter, 4 year linear vesting.

4.2 Emissions & Inflation

COMMON follows a controlled inflation model. Inflation will not be enabled at TGE, and will only be enabled pending Security Multisig review. Once inflation has been enabled, the schedule is as follows. The Foundation anticipates inflation to be enabled near to launch to enable new programmatic rewards for the protocol.

Year	**Annual Inflation**
Year 1	5%
Year 2	4%
Year 3	3%
Year 4+	2% terminal (voted on by governance)

Starting in Year 3, veC holders will vote annually on whether to continue inflation and where it should be directed: boosts (to lockers), emissions (to active contributors), or DAO treasury (non-circulating).

4.3 The COMMON Flywheel

Votes → Emissions → Actions → Fees → Rebates → More veC *(repeat).*

- **Actions generate fees:** launchpad deployments, contest entries, bonding curve trades, and futarchy market activity generate fees via `FeeManager`.
- veC **directs emissions:** weekly votes control gauge budgets, directing emissions to active loops (Launchpad, Contests, Community Stake, Decision Markets).
- **Rebates compound influence:** a defined share of fees from funded gauges is rebated to supporting veC voters.

Long-term holders lock COMMON for governance power; active traders participate in prediction markets; both benefit from protocol growth. Governance can adjust splits and weights over time to optimize for protocol growth and stakeholder alignment.

4.4 Governance & veC

Users can lock COMMON for up to 4 years to receive veC, which provides:

- Voting power over emissions and governance
- Share of protocol fees

Lock Duration	veC **Received**
1 year	25 veC per 100 COMMON
4 years	100 veC per 100 COMMON

veC holders vote weekly on where emissions go (e.g. posts, apps, markets) and receive a portion of fees generated by their selected markets.

Initial gauge types:

- **Namespaces** — community identity spaces
- **Contests** — reward-based coordination games
- **Bounties** — scoped on-chain tasks
- **Launchpads** — token launch coordination

Each gauge receives a fixed percentage of emissions rebated to veC holders. Note that veVoting will not be enabled at launch and will gradually be introduced as enabled by the security multisig.

4.5 Emissions Distribution & Control

Onchain, a `MintAuthority` contract mints per epoch (e.g., weekly with `epochsPerYear = 52`) based on a fixed annual schedule for early years, then 2% terminal thereafter.[1] Each mint splits tokens by basis points:

- A configurable `emissionsPercentage` goes to the `EmissionsManager`
- The remainder goes to the DAO vault

The `EmissionsManager` then splits its allocation according to `splitPercentage`:

- One portion to `GaugeManager` (for action-based emissions controlled by veC votes)
- One portion to `StakingPool` (for veC holder rewards)

Controlling emissions with veC**.** veC holders control emissions by voting weekly on gauge weights, directing the `GaugeManager` budget across protocol actions (Namespaces, Contests, Bounties, Launchpads). Each week, veC votes set per-action gauge weights (spendable budgets). Authorized hooks observe on-chain events and pay rewards from those budgets as actions occur. Actions generate fees; a defined share (e.g., 30–50%) is rebated pro rata to supporting voters; the remainder accrues to protocol revenue.

4.6 Prediction-Market Governance

For **Common's own governance**, prediction markets (futarchy) create a dual-track system that enhances COMMON token utility:

veC holders control the agenda. veC holders govern *which* governance questions get asked and funded by directing emissions toward specific futarchy markets via gauge votes. This allows long-term COMMON holders to shape the DAO's decision-making priorities without needing to deploy additional capital into prediction markets. Locking COMMON for veC becomes more valuable because it grants agenda-setting power over protocol decisions.

[1] See `governance/src/TokenContracts/MintAuthority.sol` and `governance/src/Incentives/EmissionsManager.sol`.

Markets price the outcomes. Once a futarchy market is funded, market participants provide liquidity and trade using underlying assets (USDC, project tokens) to price governance outcomes. Traders express conviction by buying PASS or FAIL claims, with the market's TWAP determining execution. This creates a price-discovery mechanism for governance decisions while keeping COMMON capital efficient—veC holders don't need to lock additional capital to participate in outcome pricing.

Utility to COMMON **holders.** This dual-track design increases COMMON utility: (1) veC holders gain agenda-setting power over protocol governance, (2) veC holders receive trading fees from futarchy markets they fund, creating direct cash flows from governance activity, and (3) prediction markets provide data-driven signals that improve governance quality, increasing the value of holding and locking COMMON.

5 Use Cases: Present and Future

5.1 The Common App: Loops in Production

The Common App demonstrates how loop primitives integrate into a production application. All loops are anchored to **threads**, enabling users and agents to launch, earn, and govern directly from conversations.

Currently integrated:

- **Launchpad:** Deploy tokens with bonding curves directly from threads. Founders configure initial distribution, fee splits, and optional hooks (vesting, conditional releases, referral rewards).
- **Contests:** Run recurring competitions attached to threads. Users submit content, vote with weighted strategies, and claim rewards based on ranking.
- **Post Coins:** Reward content attached to each thread. Winners from contests can launch Post Coins that capture attention and trading fees.
- **Community Stake:** Continuous bonding curves for namespace tokens. Used for distribution alongside contests and other info-fi mechanisms.
- **Decision Markets:** Conditional markets (futarchy) attached to threads. Built on the Launchpad + conditional vault pattern, enabling governance via prediction markets.

These surfaces compose: contests feed launchpads, launchpads create community stake, decision markets gate governance. Each action generates fees that flow to veC holders, creating a flywheel where activity compounds value.

5.2 Agents as Economic Participants

Common is designed for multi-agent coordination. Humans and AI agents operate as equal economic participants: both submit intents, vote in contests, claim rewards, and trigger loop finalization. The protocol treats all actors uniformly—what matters is the action, not whether it originated from a person or a program.

MCP-native protocol. The Model Context Protocol (MCP) exposes Common's loop primitives as callable tools. An agent can launch namespaces, submit content to contests, trade on bonding curves, lock COMMON for veC, and vote on emissions—all via API calls. Users are simply agents with humans in the loop. The economic rights and incentives are identical.

5.3 A Universal Coordination Layer

The Common App is the first implementation, but the protocol is designed as a universal coordination layer for any use case where agents need to organize action, allocate resources, and settle value. The same loop primitives that power token launches and contests can coordinate media curation (contests, tipping, royalty splits), research and science (peer review bounties, data markets, grant allocation), and code development (PR bounties, issue prioritization, contributor rewards). Any domain that requires recurring coordination, conditional settlement, or market-based ranking can build on these primitives.

Composable loops, composable apps. Future apps can compose protocol primitives into new surfaces: custom bonding curves, multi-token pairs, tiered access, cross-loop workflows where points from one loop feed ranking in another. Agents route between loops to optimize yield and influence. Apps that route through canonical markets remain eligible for emissions and rebates; apps that fragment liquidity do not. This aligns builders: innovate on UX and use cases, preserve the shared economic layer.

5.4 The Path Ahead

Common starts with threads, tokens, and contests. What comes next is emergent: new apps, new loops, new agent strategies. The protocol is live; the flywheel is defined; the coordination layer is ready.

Disclaimer

Nothing herein is financial, investment, or legal advice. Parameters and models are subject to change via governance. Numbers are illustrative where noted.