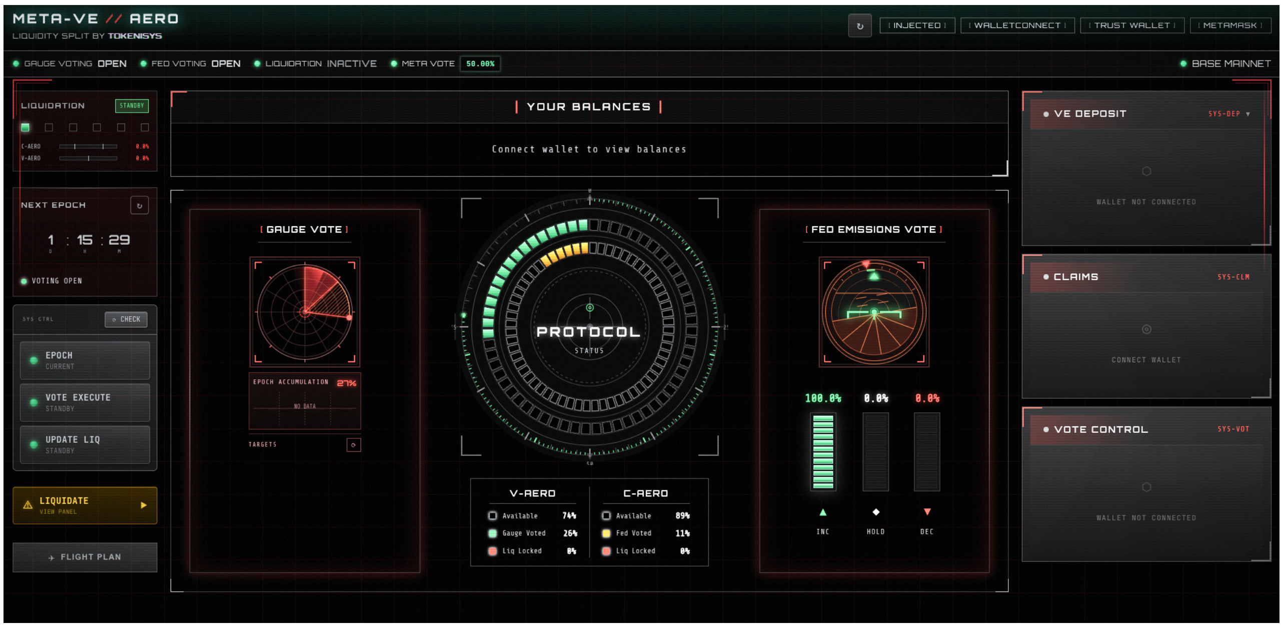
META-VE Protocol

A Comprehensive Guide

TOKENISYS

Economic Model | Smart Contract Architecture | User Operations

Gamma - January 2026



V-AERO		C-AERO	
Available	74%	Available	89%
Gauge Voted	26%	Fed Voted	11%
Liq Locked	0%	Liq Locked	0%

1 Executive Summary

META-VE is a fully autonomous token economic system that wraps vote-escrowed (VE) tokens from decentralised exchanges into liquid, composable assets. All asset flows are handled programmatically through smart contracts, so there is no active treasury management.

The protocol introduces several key innovations:

- **VE Split:** Decomposes veAERO into V-AERO (voting power) and C-AERO (capital rights).
- **DeltaForce Emissions:** Algorithmic logistic curve that responds to staking participation.
- **Dynamic Incentive/Fee Split:** Inverse relationship between emissions and fee distribution.
- **Integrated Locking:** Epoch-based locks prevent vote-and-dump attacks.
- **Vested Interest Model:** No keeper infrastructure; all actions are triggered by those who benefit.
- **EmissionsVoteLib:** Decoupled Fed emissions vote aggregation for gas-efficient voting.
- **GAMMA Transfer Settlement:** Hardened transfer mechanics with sweep-on-amount, sender checkpoint preservation, and round-UP recipient blending.
- **Liquidation Process:** Multi-phase wind-down requiring supermajority consent from both C and V holders.
- **L1 Proof Verification:** Trustless cross-chain communication via Merkle Patricia proofs.

Protocol Thesis

META-VE treats VE positions as programmable economic primitives. By separating voting rights from capital claims and coupling this with a dynamic emission and fee model, the system can bootstrap and sustain deep liquidity for VE ecosystems with minimal governance overhead.

2 Design Philosophy

META-VE is built on four foundational principles that differentiate it from traditional DeFi protocols.

2.1 Autonomous Operation

The protocol requires no manual treasury management. Fee collection, fee distribution, emission minting, and cross-chain settlement are all executed automatically via deterministic contract logic.

A multisig (MSIG) retains only high-level governance responsibilities:

- Whitelisting VE pools.
- Updating contract addresses.
- No withdrawal or funding responsibilities.

2.2 User-Pays Model

Every gas cost is aligned with vested interests. Those who benefit from an action pay for its execution. For example:

Action	Who Pays	Rationale
`depositVeAero()`	Depositor	Swaps for C + V ERC-20 tokens
`vote()`	Voter	Directs emissions and earns bribes
`claimRewards()`	Claimant	Directly receives value
`executeGaugeVote()`	Anyone	Public good / MEV opportunity
Generate L1 proofs	Claimant	Enables cross-chain claim

Table 1: User-pays gas model

2.3 Base as Hub

All staking and governance take place on Base L2. Remote chains (for example Optimism, Arbitrum) are consumers that receive allocations derived from state on Base.

This architecture:

- Avoids cross-chain consensus complexity.
- Uses L1 state proofs for trustless verification.
- Centralises *decision making* on a single rollup whilst distributing *value* across multiple chains.

2.4 Acceptable Trade-offs

The protocol accepts certain constraints in return for trustlessness:

- **2-3 hour proof lag**: Output roots are posted hourly; challenge periods only apply to withdrawals, not state reads.
- **Proof generation costs:** Approximately $0.05-0.10 per cross-chain claim.
- **Bridge requirement:** Users on remote chains must bridge to Base for staking and governance.

3 The VE Split Mechanism

Vote-escrowed tokens such as veAERO (Aerodrome) or veVELO (Velodrome) represent permanently locked positions that confer voting rights and fee claims. META decomposes these rights into separate, tradeable instruments.

3.1 Token Architecture

When a user deposits a veAERO NFT, the `VeAeroSplitter` contract mints two tokens:

V-AERO (Voting Token)

- Represents voting power for gauge direction.
- Non-transferable while locked for voting.
- Allocation: 90% to user, 1% to Tokenisys, 9% to META contract.
- Used for gauge voting, passive voting, and liquidation confirmation.

C-AERO (Capital Token)

- Represents economic rights to trading fees.
- Fully transferable (ERC-20).
- Allocation: 99% to user, 1% to Tokenisys.
- Receives 50%+ of trading fees (via `CToken.claimFees()`) plus META incentives (via `CToken.claimMeta()`).

3.2 Deposit Split Distribution

Recipient	V-AERO	C-AERO	Purpose
User	90%	99%	Primary owner
Tokenisys	1%	1%	IP fee
META	9%	-	Protocol voting power

Table 2: veAERO deposit split

3.3 Why the Split Matters

The 9% V-AERO allocation accumulated in META functions as the protocol's governance flywheel. This V-AERO is voted:

- 50% passively (following the collective vote).
- 50% for the META-AERO liquidity pool.

This creates a self-reinforcing loop that strengthens META liquidity incentives as more veAERO is deposited.

Separating V and C tokens enables:

- **Capital efficiency:** C-AERO can be traded, collateralised, or LP'd while voting rights remain locked.
- **Specialisation:** Voters and yield farmers can specialise along distinct risk and time profiles.
- **Decoupled incentives:** Voting power and capital claims respond independently to market forces.

4 DeltaForce Emission Model

META tokens are minted following a logistic growth curve that naturally balances between bootstrapping growth and long-term sustainability.

4.1 Core Formula

The **DeltaForce** function computes daily emissions using a logistic differential equation:

$$\frac{dP}{dt} = k \cdot P \cdot (1 - P) \cdot U(S),$$

where:

- P is the current progress toward maximum supply (`baseIndex`), ranging from 0 to 1.
- $k = 0.00239$ is the base growth rate (0.239% per day).
- $U(S)$ is a utilisation modifier dependent on the staking ratio S.
- $(1 - P)$ represents the remaining supply fraction.

Daily tokens minted:

$$\text{tokens} = \text{TOTAL\_SUPPLY} \times k \times P \times (1 - P) \times U(S).$$

4.2 The Utilisation Function

The modifier $U(S)$ follows a parabolic curve:

$$U(S) = 4 \cdot S \cdot (1 - S).$$

This creates a natural equilibrium at $S = 50\%$ where emissions are maximised:

- If $S = 0\%$: $U(S) = 0$ (no staking, no emissions).
- If $S = 50\%$: $U(S) = 1.0$ (optimal, maximum emissions).
- If $S = 100\%$: $U(S) = 0$ (oversaturated, no new emissions).

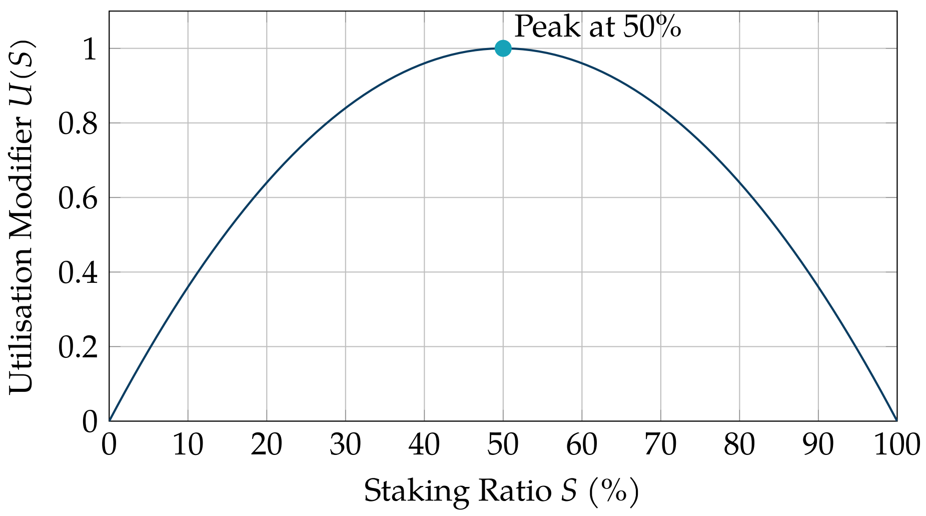


Figure 1: Parabolic utilisation function $U(S) = 4S(1 - S)$

4.3 Logistic Growth Characteristics

The logistic function naturally creates three phases:

Phase 1: Exponential Growth ($P < 0.5$) When few tokens are minted, $(1 - P) \approx 1$, and emissions grow exponentially:

$$\frac{dP}{dt} \approx k \cdot P \cdot U(S).$$

Phase 2: Peak Emission ($P \approx 0.5$) At 50% of supply minted, both P and $(1 - P)$ contribute equally, maximising the emission rate:

$$\frac{dP}{dt} = k \cdot 0.25 \cdot U(S).$$

Phase 3: Exponential Decay ($P > 0.5$) As supply approaches the cap, $(1 - P)$ becomes small, and emissions decay toward zero:

$$\frac{dP}{dt} \approx k \cdot (1 - P) \cdot U(S).$$

4.4 Economic Rationale

The logistic model balances:

- **Early bootstrapping:** Exponential growth attracts initial participants.
- **Natural ceiling:** Emissions peak at 50% supply, preventing hyperinflation.
- **Long-term sustainability:** Asymptotic decay ensures the cap is never exceeded.
- **Staking equilibrium:** Parabolic $U(S)$ creates stable 50% staking target.

4.5 Emission Distribution

Freshly minted META is split between:

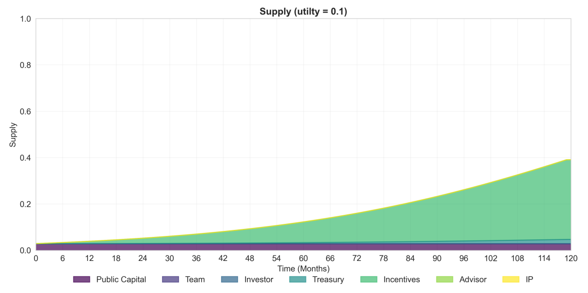
Recipient	Share	Purpose
Treasury	5%	Protocol development
Tokenisys	2.8%	IP fee
C-token Holders	$S/2 \times 92.2\%$	Distributed to C-AERO via `claimMeta()`
LP Incentives	$S/2 \times 92.2\%$	META-AERO gauge rewards
Protocol Reserve	$(1 - S) \times 92.2\%$	Retained in META contract

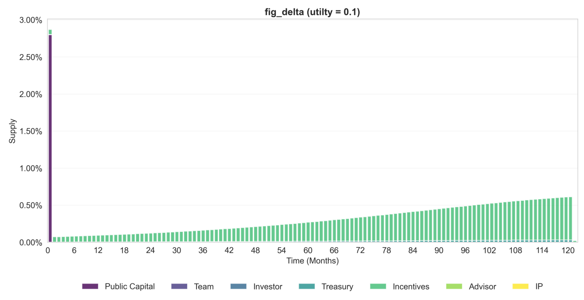
Table 3: Emission split by recipient (Gamma)

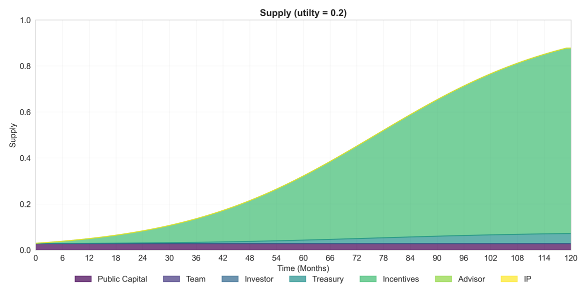
As staking increases, more emissions flow to capital holders (C-AERO), creating a natural feedback loop that stabilises staking around the target 50% ratio.

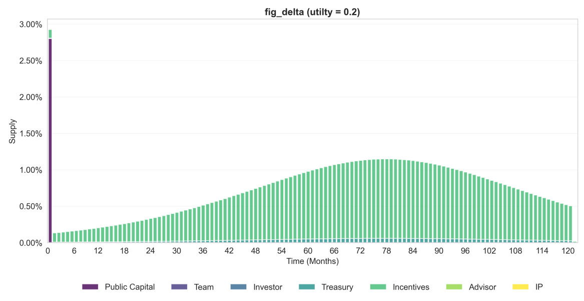
4.6 Utilisation Effect on Emissions

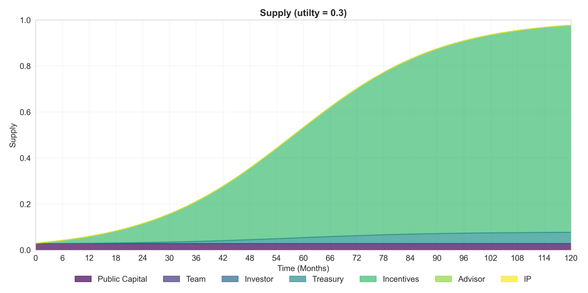
For visualisation we assume scenarios of static utility ranging 10 - 50% (noting the symmetry of U and $1 - U$).

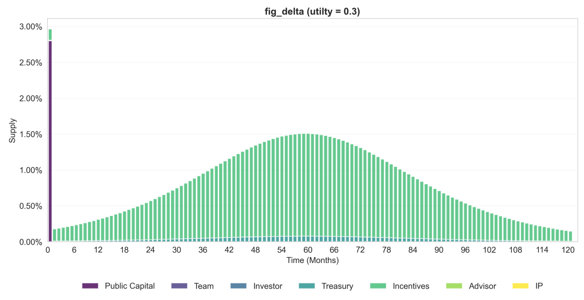

Supply (utilty = 0.1)
Supply
Time (Months)
Public Capital
Team
Investor
Treasury
Incentives
Advisor
IP

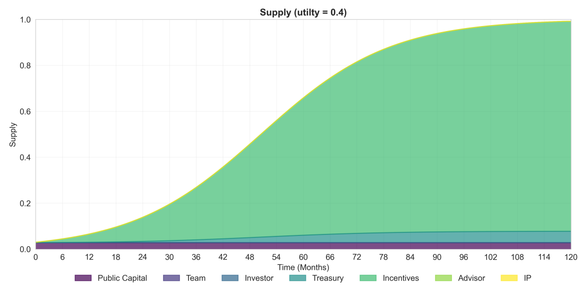

fig_delta (utilty = 0.1)
Supply
Time (Months)
Public Capital
Team
Investor
Treasury
Incentives
Advisor
IP

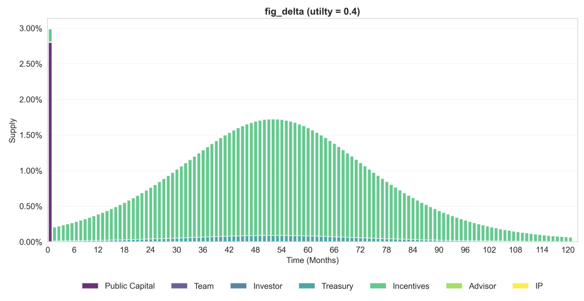

Supply (utilty = 0.2)
Supply
Time (Months)
Public Capital
Team
Investor
Treasury
Incentives
Advisor
IP

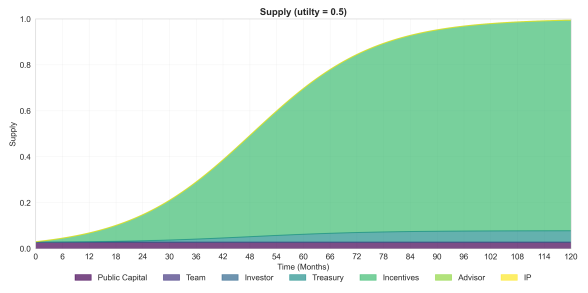

fig_delta (utilty = 0.2)
Supply
Time (Months)
Public Capital
Team
Investor
Treasury
Incentives
Advisor
IP

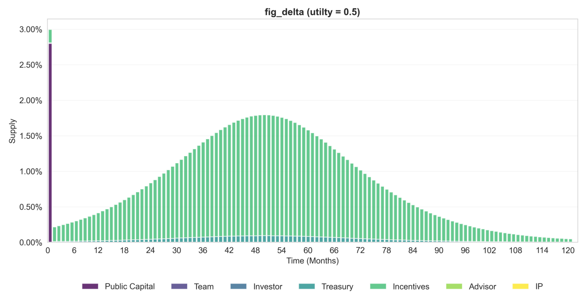

Supply (utilty = 0.3)
Supply
Time (Months)
Public Capital
Team
Investor
Treasury
Incentives
Advisor
IP


fig_delta (utilty = 0.3)
Supply
Time (Months)
Public Capital
Team
Investor
Treasury
Incentives
Advisor
IP

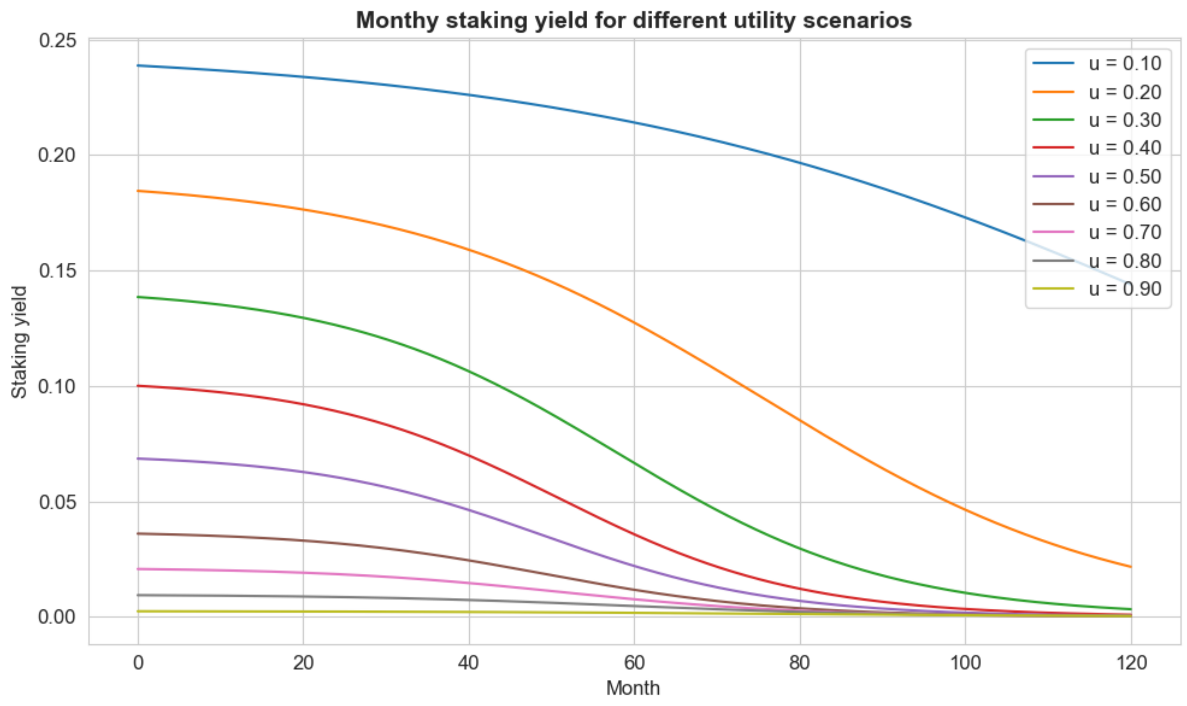

Supply (utilty = 0.4)
Supply
Time (Months)
Public Capital
Team
Investor
Treasury
Incentives
Advisor
IP

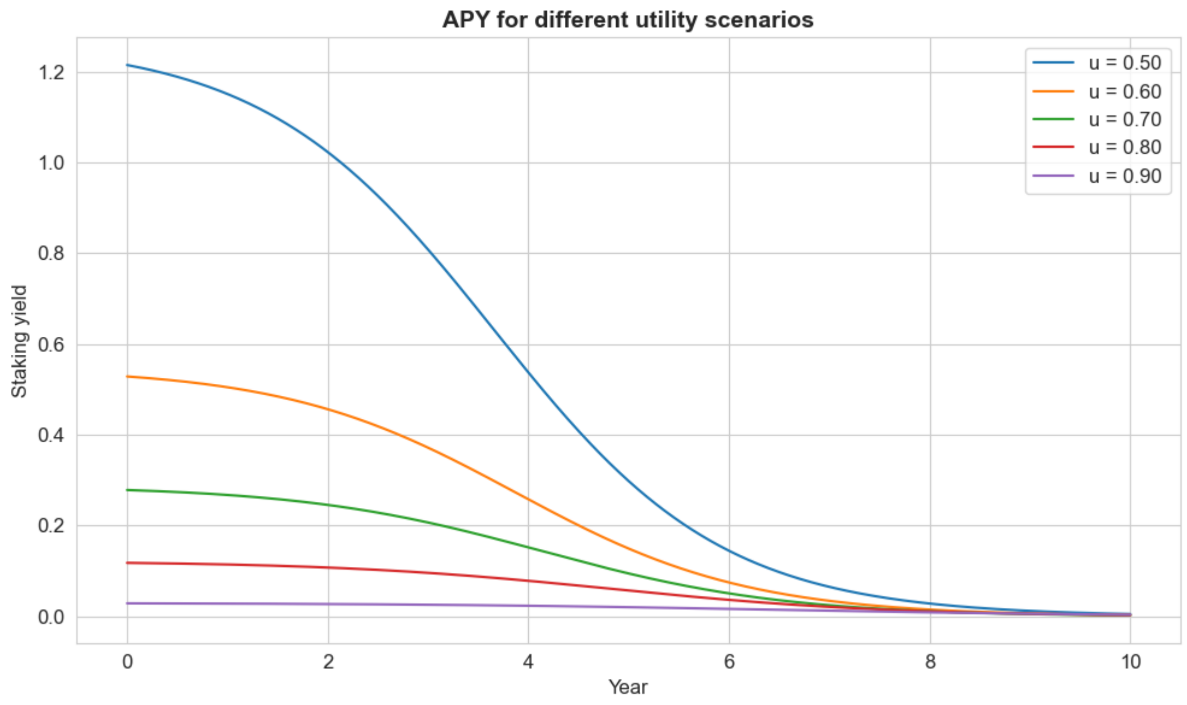

fig_delta (utilty = 0.4)
Supply
Time (Months)
Public Capital
Team
Investor
Treasury
Incentives
Advisor
IP


Supply (utilty = 0.5)
Supply
Time (Months)
Public Capital
Team
Investor
Treasury
Incentives
Advisor
IP


fig_delta (utilty = 0.5)
Supply
Time (Months)
Public Capital
Team
Investor
Treasury
Incentives
Advisor
IP

Whilst utilisation decreases emissions as utility diverges from equilibrium, it allows staking yield to be correctly inversely proportional to utilisation with increasing on staking increasing emissions out to C token holders of VE-AERO and the META LP Gauge.





5 Dynamic Incentive vs Fee Distribution

META-VE features a novel incentive model where emission and fee distributions move in opposite directions based on the staking ratio S.

5.1 Fee Flow Architecture

Trading fees collected from Aerodrome are routed through two channels:

1. **50% to C-AERO holders:** Distributed via the `CToken.feePerCToken` index mechanism. C-AERO holders call `CToken.claimFees()` to receive AERO.
2. **50% to META contract:** This portion accumulates in the protocol reserve and supports META token value.

The fee collection flow is:

1. `Splitter.collectFees()` collects AERO from Aerodrome fee distributors
2. 50% is sent directly to `CToken`, updating `feePerCToken`
3. 50% is retained in the protocol
4. `CToken.claimFees()` allows users to claim their proportional AERO

5.2 Fee Distribution Model

In Gamma, the fee distribution is simplified:

Source	To C-AERO	Notes
Trading Fees	50%	Via `CToken.claimFees()`
META Emissions	$S/2 \times 92.2\%$	Via `CToken.claimMeta()`
LP Gauge	$S/2 \times 92.2\%$	Via Aerodrome gauge

Table 4: C-AERO holder reward sources (Gamma)

5.3 Utilisation-Based Emission Scaling

The DeltaForce emission model creates self-stabilising pressure through the parabolic utilisation function $U(S) = 4S(1 - S)$:

- **Low utilisation ($S < 50\%$):** Higher emissions per unit staked attract more participation.
- **High utilisation ($S > 50\%$):** Lower emissions reduce dilution pressure.
- **Equilibrium:** The system naturally gravitates toward $S \approx 50\%$, where emissions are maximised.

5.4 Reward Summary

C-AERO holders benefit from multiple reward streams:

Reward Type	**Token**	**Claim Function**
Trading Fees	AERO	`CToken.claimFees()`
META Emissions	META	`CToken.claimMeta()`
Rebase Growth	V/C-AERO	`claimRebase()`

Table 5: C-AERO holder reward types

6 Integrated Locking & Anti-Gaming

META-VE implements anti-gaming features using epoch-aligned locks, transfer settlement, and bribe snapshots.

6.1 Epoch-Based Lock Architecture

All voting locks are aligned with the Aerodrome epoch (Thursday 00:00 UTC to Wednesday 23:59 UTC).

Action	Token	Lock Duration	Unlock Trigger
`VToken.vote()`	V-AERO	Until epoch end	Epoch rollover
`CToken.voteEmissions()`	C-AERO	Until epoch end	Epoch rollover
`confirmLiquidation()`	V-AERO	Until resolution	Liquidation end
`voteLiquidation()`	C-AERO	Until resolution	Liquidation end

Table 6: Epoch-based lock rules

6.2 Vote-and-Dump Prevention

When a user votes, their tokens are locked until the epoch ends. This prevents:

- Vote selling followed by immediate transfer.
- Flash loan voting, since borrowed tokens cannot be returned whilst locked.
- Double voting of the same underlying voting power.

6.3 Transfer Settlement (GAMMA)

On C-AERO transfers, the `onCTokenTransfer` hook in VeAeroSplitter ensures that windfall attacks are not possible. The GAMMA update fixes several edge cases in the transfer settlement logic.

Aspect	Pre-GAMMA (Bug)	GAMMA (Fixed)
Sweep calculation	On sender's post-transfer balance (often 0)	On `amount` transferred
Sender checkpoint	Updated to global (lost claim on remaining)	**Unchanged** (can claim on remaining)
Recipient checkpoint	Could accumulate windfall	Blended with **round-UP**
Self-transfer	Swept rewards to Tokenisys	**No-op** (early return)
Rebase sweep	100% to Tokenisys	**91% Tokenisys, 9% META**

Table 7: GAMMA transfer settlement changes

GAMMA Changes Summary

Sender Settlement

- Unclaimed Splitter rewards (fees, META, rebase) on the *transferred amount* are swept to Tokenisys.
- **Sender's checkpoint remains unchanged**-they can still claim rewards on their remaining balance.
- CToken rewards (META + fees via `feePerCToken` / `metaPerCToken`) are preserved in `userClaimable` storage.

Recipient Settlement

- **New holders** are assigned current global index values, preventing a windfall.
- **Existing holders** receive a weighted average checkpoint with *round-UP* to prevent dust accumulation attacks:

  $$\text{newCheckpoint} = \left\lceil \frac{\text{oldBalance} \cdot \text{oldCheckpoint} + \text{amount} \cdot \text{globalIndex}}{\text{newBalance}} \right\rceil$$

  Implemented as: `(numerator + newBalance - 1) / newBalance`

Self-Transfer Guard A self-transfer (`from == to`) is now a no-op-no sweep occurs, no checkpoint changes. This prevents accidental reward forfeiture.

Rebase Sweep Fee Split When rebase rewards are swept on transfer, the fee split mirrors `claimRebase()`:

- **V-AERO:** 91% to Tokenisys, 9% to META contract
- **C-AERO:** 100% to Tokenisys

Anti-Sniping Protection

If you transfer C-AERO without first calling `claimFees()` and `claimMeta()` on the Splitter, your unclaimed Splitter rewards on the transferred tokens are swept to Tokenisys. However, you *can* still claim on your remaining balance (GAMMA fix), and your CToken rewards are preserved in `userClaimable`.

6.4 Bribe Snapshot Mechanics

Bribes are distributed to V-AERO voters who are recorded in an epoch snapshot:

1. User votes during epoch N.
2. After the voting window closes (Wednesday 22:00 UTC), user calls `snapshotForBribes()`.
3. The snapshot records voting power at that moment.
4. During epoch $N + 1$, the user calls `claimBribes()` to receive their share.

A one-hour snapshot window (Wednesday 22:00-23:00) prevents last-second sniping.

6.5 EmissionsVoteLib Architecture

Since Beta V11.1, the Fed emissions voting logic has been extracted into a separate `EmissionsVoteLib` contract for improved modularity and gas efficiency.

Design Rationale

- **Separation of concerns:** Vote aggregation is decoupled from the main Splitter contract.
- **Gas efficiency:** Vote recording and totals are stored in a dedicated contract.
- **Upgradability:** The emissions vote logic can be updated without modifying core contracts.

Vote Flow

1. C-AERO holder calls `CToken.voteEmissions(choice, amount)` where choice is -1 (decrease), 0 (hold), or $+1$ (increase).
2. CToken validates the vote and calls `EmissionsVoteLib.recordVote(user, choice, wholeTokens)`.
3. Vote totals are accumulated in `emissionsDecreaseTotal`, `emissionsHoldTotal`, or `emissionsIncreaseTotal`.
4. During execution window (Wed 22:00-Thu 00:00), `Splitter.executeEmissionsVote(proposalI` calls `EmissionsVoteLib.getWinningChoice()` to determine the majority.
5. At epoch reset, `Splitter.resetEpoch()` calls `EmissionsVoteLib.resetEpoch()` to clear vote totals.

Function	Caller	Purpose
`recordVote()`	CToken only	Record user's emissions vote
`resetEpoch()`	Splitter only	Clear totals for new epoch
`getWinningChoice()`	Anyone (view)	Return winning vote option
`getTotals()`	Anyone (view)	Return all three vote totals

Table 8: EmissionsVoteLib access control

7 Liquidation Process

The liquidation mechanism enables protocol wind-down when a supermajority of stakeholders consent. It is deliberately slow and requires buy-in from both capital holders (C-AERO) and voters (V-AERO), protecting minority stakeholders from hostile takeovers.

7.1 Liquidation Phases

Phase	Name	Trigger	Description
0	Normal	N/A	Protocol operates normally
1	CLock	C-AERO locking begins	Users can lock C to signal interest
2	CVote	≥25% C-AERO locked	90-day voting period begins
3	VConfirm	≥75% C voted	V-AERO holders must confirm
4	Approved	≥50% V confirmed	Liquidation approved, 7-day claim window
5	Closed	7 days after approval	No further R-AERO claims

Table 9: Liquidation phases

7.2 Phase Transitions and Thresholds

Transition	Threshold	Delay	Action
Normal → CLock	Any C locked	Instant	-
CLock → CVote	25% of C supply	Instant	-
CVote → VConfirm	75% of C supply	90 days	`resolveCVote()`
CVote → Normal	<75% of C	90 days	Liquidation fails
VConfirm → Approved	50% of V supply	1 epoch	`resolveVConfirm()`
VConfirm → Normal	<50% of V	1 epoch	Liquidation fails
Approved → Closed	Automatic	7 days	Time-based

Table 10: Liquidation transition rules

7.3 R-AERO (Redemption Token)

When liquidation is approved, C-AERO holders who participated in the vote receive R-AERO tokens proportional to their locked C-AERO. R-AERO represents a claim on the underlying veAERO NFTs held by the protocol.

Characteristics:

- **Minting:** R-AERO is minted 1:1 for C-AERO locked during CVote.

- **Claiming:** Users must claim R-AERO within 7 days of approval (Phase 4).
- **Redemption:** R-AERO can be burned to extract veAERO NFTs proportionally from the treasury.

7.4 Liquidation Flow Diagram

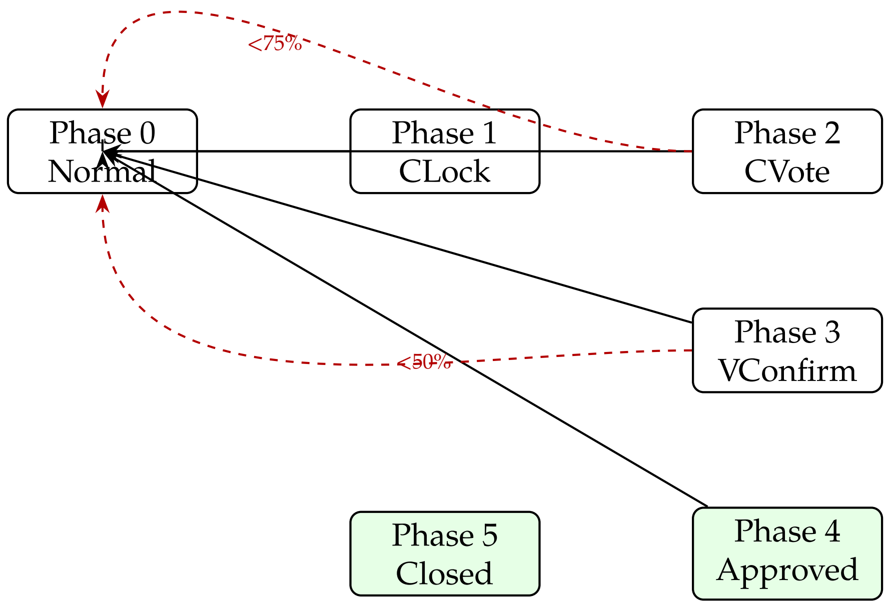


Figure 3: Liquidation state machine

7.5 User Actions by Phase

Phase	Action	Description
0-2	`voteLiquidation(amount)`	C-AERO holders (locks tokens)
3	`confirmLiquidation(amount)`	V-AERO holders (locks tokens)
4	`claimRTokens()`	Get R-AERO for locked C-AERO
Any	`withdrawFailedLiquidation()`	Recover tokens if liquidation fails

Table 11: User actions by liquidation phase

7.6 Anti-Gaming Measures

- **Long voting period:** 90 days prevents flash governance attacks.
- **Dual consent:** Requires both C-AERO (75%) and V-AERO (50%) supermajorities.
- **Token locks:** All liquidation votes lock tokens until resolution.
- **Claim window:** R-AERO must be claimed within 7 days, preventing indefinite lock-up.

- **Failure recovery:** Users can withdraw locked tokens if liquidation fails.

8 Gas Optimisation Architecture

META-VE employs multiple gas-saving strategies to keep operations affordable at scale.

8.1 Index-Based Claims

Rather than iterating over all holders during reward distribution, the protocol uses a global index system. Each user's pending rewards are calculated:

$$\text{pending} = \text{balance} \times (\text{globalIndex} - \text{userCheckpoint}).$$

This reduces claim costs from $O(n)$ to $O(1)$.

8.2 Bitpacked Vote Storage

Vote weights are stored in a custom `DynamicGaugeVoteStorage` library that packs multiple pool weights into single storage slots. For example:

- Each pool weight uses 18 bits (supports up to 262,143 votes per pool).
- 14 pools fit in a single `uint256` slot.
- Initialization cost for 100 pools: ∼160k gas vs ∼2M gas for naive implementation.

8.3 Immutable Addresses

Core contract addresses (`AERO_TOKEN`, `VOTING_ESCROW`, etc.) are immutable, avoiding repeated `SLOAD` operations.

8.4 Lazy Evaluation

Index updates are triggered by the first user action after a distribution event, rather than pushing updates to all users.

8.5 Gas Benchmarks

Operation	**Gas Cost**	**Notes**
`depositVeAero()`	∼285,000	First deposit (higher)
`depositVeAero()`	∼235,000	Subsequent deposits
`vote()`	∼180,000	Single pool
`votePassive()`	∼95,000	No pool storage
`CToken.collectFees()`	∼85,000	Pulls from Meta
`CToken.claimFees()`	∼65,000	Index-based AERO
`CToken.collectMeta()`	∼80,000	Pulls from Meta
`CToken.claimMeta()`	∼52,000	Index-based
`executeGaugeVote()`	∼320,000	Includes Aerodrome call
`consolidate(50)`	∼620,000	Batch of 50 NFTs

Table 12: Gas benchmarks (Base L2, Dec 2025)

9 Vested Interest Model (No Keepers)

META-VE does not rely on dedicated keeper bots or external incentives. Instead, every protocol action is designed to be executed by those who directly benefit.

9.1 Self-Executing Actions

Action	Executor	Incentive
`depositVeAero()`	Depositor	Receives V + C tokens
`vote()`	Voter	Directs emissions and earns bribes
`voteEmissions()`	C-AERO holder	Directs Fed emissions policy
`CToken.claimFees()`	Fee claimant	Receives AERO
`CToken.claimMeta()`	META claimant	Receives META
`claimBribes()`	Bribe claimant	Receives bribe tokens
`executeGaugeVote()`	Anyone	MEV / public good
`executeEmissionsVote()`	Anyone	MEV / public good
`Splitter.collectFees()`	Anyone	MEV / protocol health
`CToken.collectFees()`	Anyone	MEV / protocol health
`consolidatePending()`	Anyone	MEV / protocol health

Table 13: Stakeholder incentives

9.2 Public Goods with MEV

Certain actions (`executeGaugeVote`, `collectFees`) are technically public goods but often executed by:

- MEV searchers who extract value from being first.
- Large stakeholders who benefit from protocol health.
- Frontends that bundle actions for users.

10 Trustless Cross-Chain Via L1 Proofs

META-VE performs cross-chain operations without oracles by using L1 state proofs. Remote L2 chains can verify Base state for incentive distribution.

10.1 Trust Chain

Trust flows from Ethereum L1 to remote L2 storage:

1. **L1Block predeploy:** Base exposes an L1 block hash via a system contract.
2. **L1 block header:** The user provides a header; the hash is checked against the predeploy.
3. **L1 state root:** Extracted from the verified block header.
4. **L2OutputOracle:** A Merkle proof shows the oracle contract exists on L1 with a given storage root.
5. **Output root:** A storage proof extracts the remote L2 state root.
6. **Remote contract:** An account proof shows the target contract exists on the remote L2.
7. **Storage value:** A final storage proof reveals the required storage slot.

10.2 Merkle Patricia Trie Verification

Ethereum stores state in Merkle Patricia tries. Each proof is a path from root to leaf:

- Each node is identified by `keccak256(node_data)`.
- Verification rehashes each provided node and confirms the hash sequence.
- Any tampering invalidates the path.

The `L1ProofVerifier` contract performs RLP decoding and trie traversal on-chain.

10.3 Proof Components

Component	Size	Gas Cost (approx.)
L1 block header	~550 bytes	~20,000
L1 account proof	~800 bytes	~80,000-120,000
L1 storage proof	~600 bytes	~50,000-80,000
L2 account proof	~800 bytes	~80,000-120,000
L2 storage proof	~500 bytes	~50,000-100,000
Total	~3.3 KB	~280,000-440,000

Table 14: Typical proof footprint

11 User Guide

11.1 Depositing veAERO

- **Window:** Thursday 00:01 UTC to Wednesday 21:44 UTC.
- **Requirements:** Permanently locked veAERO NFT that has not already voted.
- **Action:** Approve the NFT to `VeAeroSplitter`, then call `depositVeAero(tokenId)`.
- **Receive:** 90% V-AERO and 99% C-AERO (1% of each to Tokenisys, 9% V to META).

11.2 Voting with V-AERO

- **Active voting:** `VToken.vote(poolAddress, amount)` allocates voting power directly. Amount must be specified in wei (18 decimals) and must represent whole tokens.
- **Passive voting:** `VToken.votePassive(amount)` follows active voters proportionally. Amount must be in wei and represent whole tokens.
- **Locking:** Tokens are locked until the epoch ends and then unlock automatically.
- **Execution:** Any user can call `executeGaugeVote()` in the Wednesday 22:00-Thursday 00:00 window.

11.3 Claiming Fees (AERO)

C-AERO holders receive trading fees in AERO via the CToken contract:

1. Check that `CToken.pendingFees(yourAddress)` is positive.
2. Ensure someone has recently called `CToken.collectFees()` to pull AERO from Meta.
3. Call `CToken.claimFees()` to receive AERO proportional to your C-AERO holdings.

11.4 Claiming META Rewards

C-AERO holders also receive META incentive emissions:

1. Check that `CToken.pendingMeta(yourAddress)` is positive.
2. Ensure someone has recently called `CToken.collectMeta()` to pull META from Meta contract.
3. Call `CToken.claimMeta()` to receive META proportional to your C-AERO holdings.

11.5 META Token

META tokens are the protocol's native incentive token:

- **Earning:** C-AERO holders earn META through the `CToken.claimMeta()` mechanism.
- **LP Incentives:** META is distributed to the META-AERO liquidity pool gauge.

- **Protocol Voting:** 9% of V-AERO deposited accumulates in the META contract and is used for protocol voting.

11.6 META Staking

META holders can lock their tokens to vote for VE pools and earn rewards:

- **Locking:** Call `Meta.lockAndVote(amount, vePool)` to lock META and vote for a VE pool (e.g., CToken address).
- **Voting constraint:** Users can only vote for one VE pool at a time. To change pools, unlock first.
- **Initiating unlock:** Call `Meta.initiateUnlock(amount)` to begin the cooldown period.
- **Unlock timing:** Unlock completes at 00:01 UTC, approximately 24-48 hours from initiation.
- **Completing unlock:** After cooldown, call `Meta.completeUnlock()` to receive tokens.
- **Rewards:** Locked META earns incentive emissions and trading fees via `Meta.claimRewards()`.

⚠ Unlock Cooldown

You cannot initiate a new unlock while one is already in progress. Complete the current unlock first before starting another.

11.7 Claiming Bribes

1. Vote during epoch N.
2. Call `snapshotForBribes()` between Wednesday 22:00 and Thursday 00:00.
3. During epoch $N+1$, call `claimBribes(tokenAddresses[])`.

11.8 Participating in Liquidation

1. **As a C-AERO holder:** Call `voteLiquidation(amount)` to lock C-AERO and signal support.
2. **As a V-AERO holder:** During VConfirm phase, call `confirmLiquidation(amount)` to lock V-AERO.
3. **After approval:** Call `claimRTokens()` within 7 days to receive R-AERO.
4. **If liquidation fails:** Call `withdrawFailedLiquidation()` to recover locked tokens.

11.9 Epoch Timeline Summary

Time (UTC)	**Window**	**Available Actions**
Thursday 00:00	Epoch start	Tokens unlock, new epoch begins
Thu 00:01-Wed 21:44	Deposit window	`depositVeAero()`, `vote()`, `votePassive()`, `voteEmissions()`
Wed 21:45-22:00	Deposit closed	Preparation for vote execution
Wed 22:00-Thu 00:00	Execution window	`executeGaugeVote()`, `executeEmissionsVote()`
Wed 22:00-23:00	Snapshot window	`snapshotForBribes()`
Wed 23:00-23:59	**Sweep window**	**`sweepBribes()` (Tokenisys only)**

Table 15: Epoch timeline

Bribe Claim Deadline

Users must call `claimBribes()` before Wednesday 23:00 UTC. After this time, unclaimed bribes may be swept by Tokenisys. This prevents bribe tokens from being locked indefinitely in the protocol.

12 Technical Reference: Wei and Whole Token Architecture

The protocol uses a hybrid storage model that optimises for both user experience and gas efficiency.

12.1 Design Principle

External APIs (balances, transfers, voting) use wei (18 decimals) for ERC-20 compatibility, whilst internal storage uses whole tokens for gas optimisation. Conversions occur only at boundaries.

12.2 Data Flow

12.3 Function Return Types

Function	Returns	Usage
VToken (User APIs)		
`balanceOf(user)`	Wei	Always format for display
`vote(pool, amount)`	-	Input must be wei
`votePassive(amount)`	-	Input must be wei
`totalPassiveVotes()`	Wei	Format for display
`totalGaugeVotedThisEpoch`	Wei	Public variable (wei)
CToken (Fee & META Distribution)		
`pendingFees(user)`	Wei	Format for display
`pendingMeta(user)`	Wei	Format for display
`feePerCToken()`	Wei	Index (scaled 1e18)
`metaPerCToken()`	Wei	Index (scaled 1e18)
`collectFees()`	Wei	Returns amount collected
`claimFees()`	Wei	Returns amount claimed
VeAeroSplitter (View Functions)		
`totalVLockedForVoting()`	Whole	Display directly
`totalPassiveVotes()`	Whole	Display directly
VeAeroBribes (Snapshot)		
`snapshotVotePower(user)`	Whole	Display directly
`totalSnapshotVotePower()`	Whole	Display directly

Table 16: Function return units by contract

12.4 Implementation Example

Listing 1: Voting with proper unit handling

```
// 1. User input (display units)
const voteAmount = "50.0";  // 50 tokens

// 2. Convert to wei for VToken API
const amountWei = parseUnits(voteAmount, 18);  // 50e18
```

```
// 3. Vote (input is wei)
await vToken.vote(poolAddress, amountWei);

// 4. Read total voted (returns wei)
const totalWei = await vToken.totalGaugeVotedThisEpoch();
const displayTotal = formatUnits(totalWei, 18);  // "50.0"

// 5. Read from splitter (returns whole tokens)
const totalWhole = await splitter.totalVLockedForVoting();
console.log(‘Total: ${totalWhole}‘);  // "50" (already whole)
```

12.5 Rationale

Layer	Rationale	Unit
User APIs	ERC-20 standard expects wei	Wei
Vote Storage	Gas efficiency (smaller numbers)	Whole
Splitter Views	Event/UI convenience	Whole
Aerodrome	Protocol requirement	Whole

Table 17: Architectural layer justifications

13 Conclusion

META Protocol VE represents a shift in VE token economics, combining:

- **Capital efficiency:** Separation of voting and economic rights.
- **Algorithmic autonomy:** DeltaForce emissions and dynamic fee splits require no discretionary management.
- **Security by design:** Epoch locks, transfer settlement, and snapshots mitigate common gaming strategies.
- **Trustless scaling:** L1 proofs enable cross-chain expansion without oracles.
- **Aligned incentives:** The vested interest model replaces keepers with economically motivated participants.
- **Graceful wind-down:** Multi-phase liquidation protects minority stakeholders whilst enabling orderly exit.

The architecture is designed to be extensible. Additional L2 chains can be onboarded by deploying the remote contract set and whitelisting their VE pools. The core economic logic remains unchanged regardless of how many chains participate.

By eliminating manual intervention points, META-VE approaches genuine decentralisation while maintaining the sophistication required for sustainable protocol growth.

META-VE Protocol
Tokenisys / Gamma / January 2026