Jeffrey Scholz Jan 10 5 min read

How Compound V3 Allocates COMP Rewards

Updated: Feb 29

Compound issues rewards in COMP tokens to lenders and borrowers in proportion to their share of the a market's lending and borrowing.

The algorithm is extremely similar to the MasterChef Staking Algorithm, so the reader should familiarize themselves with that first.

High level overview of Compound V3 rewards

Similar to MasterChef, the Compound V3 reward contract tracks how much one hypothetical "staked" USDC has earned since the beginning of time. Here "staking" can mean either borrowing or lending — both activities are rewarded.

Analogous to Compound V3's baseSupplyIndex or MasterChef's rewardPerTokenAcc, Compound rewards has trackingSupplyIndex and trackingBorrowIndex which track rewards for one USDC (lent or borrowed) since the beginning of time.

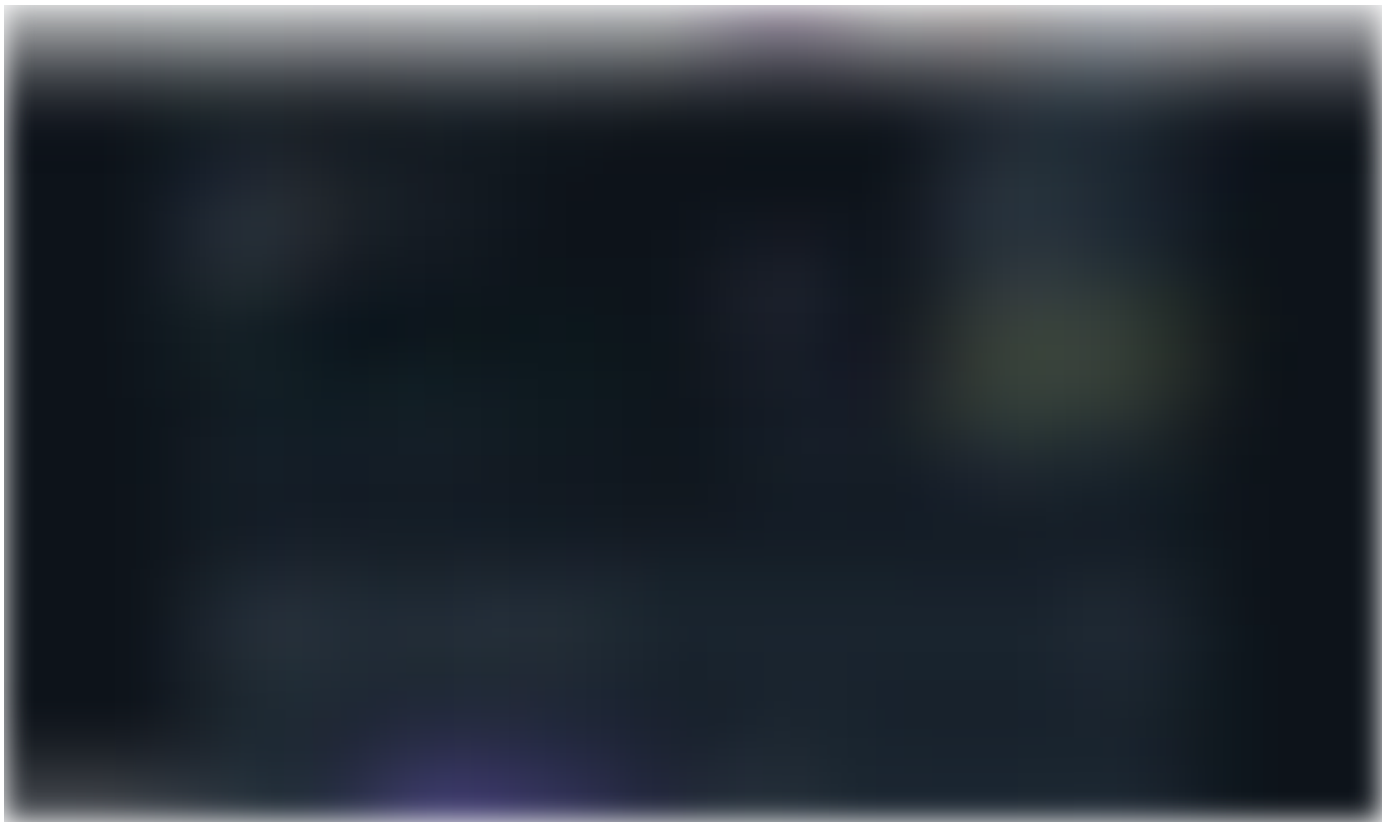
Like MasterChef, the amount of rewards a single USDC collects is "diluted" when more USDC is "staked" and vice versa.

Unlike MasterChef, the reward per unit of time is set with the immutable variables baseSupplyTrackingSupplySpeed and baseTrackingBorrowSpeed. Governance has an option to "rescale" the rewards that are distributed.

Users claim rewards from Comet Rewards a separate contract from the main Comet lending contract.

Compound does not issue rewards if the total amount borrowed or total amount lent are below certain thresholds for reasons we will discuss later.

Like MasterChef, rewards are not automatically distributed, they must be claimed in a separate transaction. The screenshot below shows the frontend for claiming accumulated COMP tokens, with the action to claim the tokens in the yellow circle.



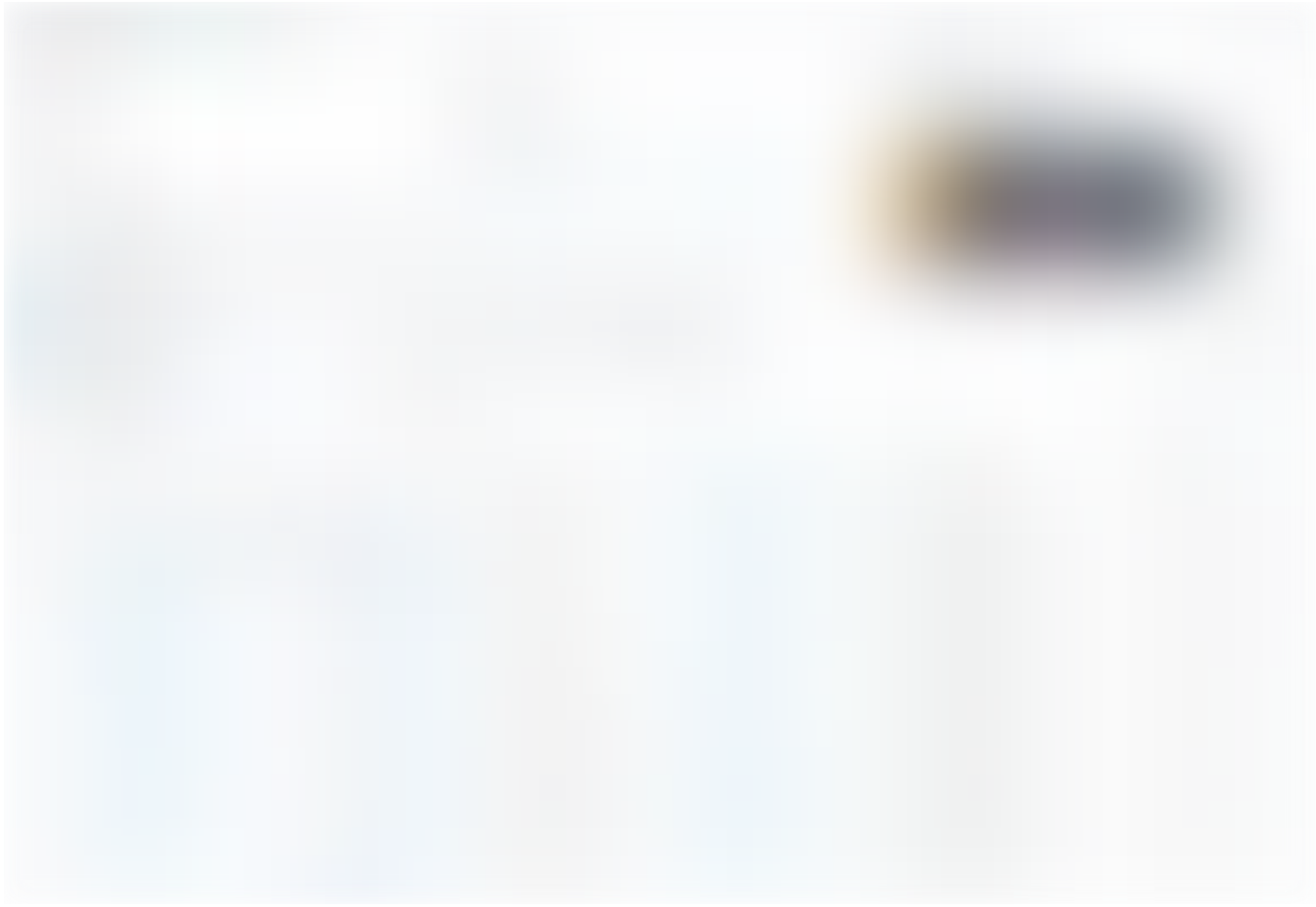
governance. Periodically, COMP tokens are transferred from the governance treasury to the reward contract. You can see the following governance transactions that "top up" the reward contract.

https://compound.finance/governance/proposals/194 (Nov 21, 2023)
https://compound.finance/governance/proposals/164 (June 29, 2023)

The mainnnet address for the rewards contract is 0x1B0e765F6224C21223AeA2af16c1C46E38885a40

Because there is a fixed supply, the COMP rewards for ecosystem participation cannot continue indefinitely unless governance buys COMP tokens on the open market.

In the Etherscan screenshot below we see most of the transactions with the contract are to claim COMP tokens (blue box), and that the contract is currently holding ~73,000 COMP tokens (blue arrow).



trackingSupplyIndex and trackingBorrowIndex behave like rewardPerTokenAcc

The plot below should be familiar from MasterChef. The more USDC that is "staked" the less reward each token receives because only a fixed amount is issued each period (determined by trackingSupplyIndex and trackingBorrowIndex).

One noteworthy variation is that if the amount of USDC supplied or borrowed (pink line) is below the baseMinForRewards (red text and dashed line) threshold, then USDC does not accumulate rewards, and the trackingSupplyIndex (or trackingBorrowIndex) does not increase for that state update.

These variables are not public, but can be retrieved via the totalsBasic() public function in CometExt. Since CometExt is a separate contract that Comet issues delegatecalls to, we cannot retrieve the values via Etherscan. Instead we use cast from Foundry to retrieve them, as the screenshot below shows.

baseMinForRewards

baseMinForRewards is defined in line 86 in Comet.sol

Compound doesn't issue rewards to lenders or borrowers if there are less than 1 million dollars (1e12 USDC as USDC has 6 decimals) lent out. Similarly, a borrowed USDC will not accumulate COMP rewards if there is less than 1 million dollars borrowed.

baseMinRewards exists to prevent accumulator overflow

If you are an auditor, this may be a fairly overlooked medium vulnerability because tests do not easily catch accumulator overflows. You need to make sure that the accumulator will not overflow for several years and this means either the reward rate needs to be small or the staked amount needs to be large.

accrueInternal() revisited

The trackingSupplyIndex and trackingBorrowIndex are updated whenever accrueInternal() is called.

The code below implements the logic described in the above sections. The if conditions in red boxes prevent trackingSupplyIndex or trackingBorrowIndex from accumulating more rewards if the supply or borrow amount is below baseMinForRewards. The baseTrackingSupplySpeed and baseTrackingBorowSpeed (blue boxes) are immutable variables, so the amount the indexes are incremented by only depends on timeElapsed and (inversely) to totalSupplyBase (or totalBorrowBase).

You can think of the baseTrackingSupplySpeed and baseTrackingBorrowSpeed as the "reward per unit time." When multiplied by timeElapsed, that computes the amount of rewards accumulated for a single participating USDC. Finally, dividing that result by totalSupplyBase or totalBorrowBase dilutes that USDC based on the total amount.

baseSupplyTrackingSpeed and baseTrackingBorrowSpeed

These variables are analogous to the rewardPerBlock of MasterChef. They specify how quickly the accumulators described in the section above increase.

We can retrieve their values from the Comet Etherscan Proxy contract.

Both of these variables use trackingIndexScale for their decimals, and per Etherscan, trackingIndexScale is 1e15:

Since they are 15 decimal fixed point numbers, their values are as follows:

baseTrackingSupplySpeed = 2.979166666666e-03
baseTrackingBorrowSpeed = 4.414467592592e-03

The variable definitions (with comments about the scale) from Comet.sol are screenshotted below

Tracking user-level rewards: baseTrackingAccrued and baseTrackingIndex

Like MasterChef, Compound Rewards accumulates rewards to an account when that account does a state-changing transaction. And also like MasterChef, the rewards the user accumulates are proportional to their balance and how much the "index" or "accumulator" changed since the last time the user did a state changing operation.

Let's look at the user struct again

baseTrackingIndex is the value of trackingSupplyIndex or trackingBorrowIndex at the time the UserBasic storage struct was last updated, depending on if the account is a lender or borrower respectively. The delta between the current trackingSupplyIndex (or trackingBorrowIndex) and the user's stored value of baseTrackingIndex determines how many rewards they will accumulate for that transaction. Consider the plot below

Whenever a user does something that will change their principal, a call to the internal function updateBasePrincipal() is made. The function will determine how much the trackingSupplyIndex or trackingBorrowIndex has changed since the last update and accumulate rewards to the user's baseTrackingAccrued accordingly. The function is shown below

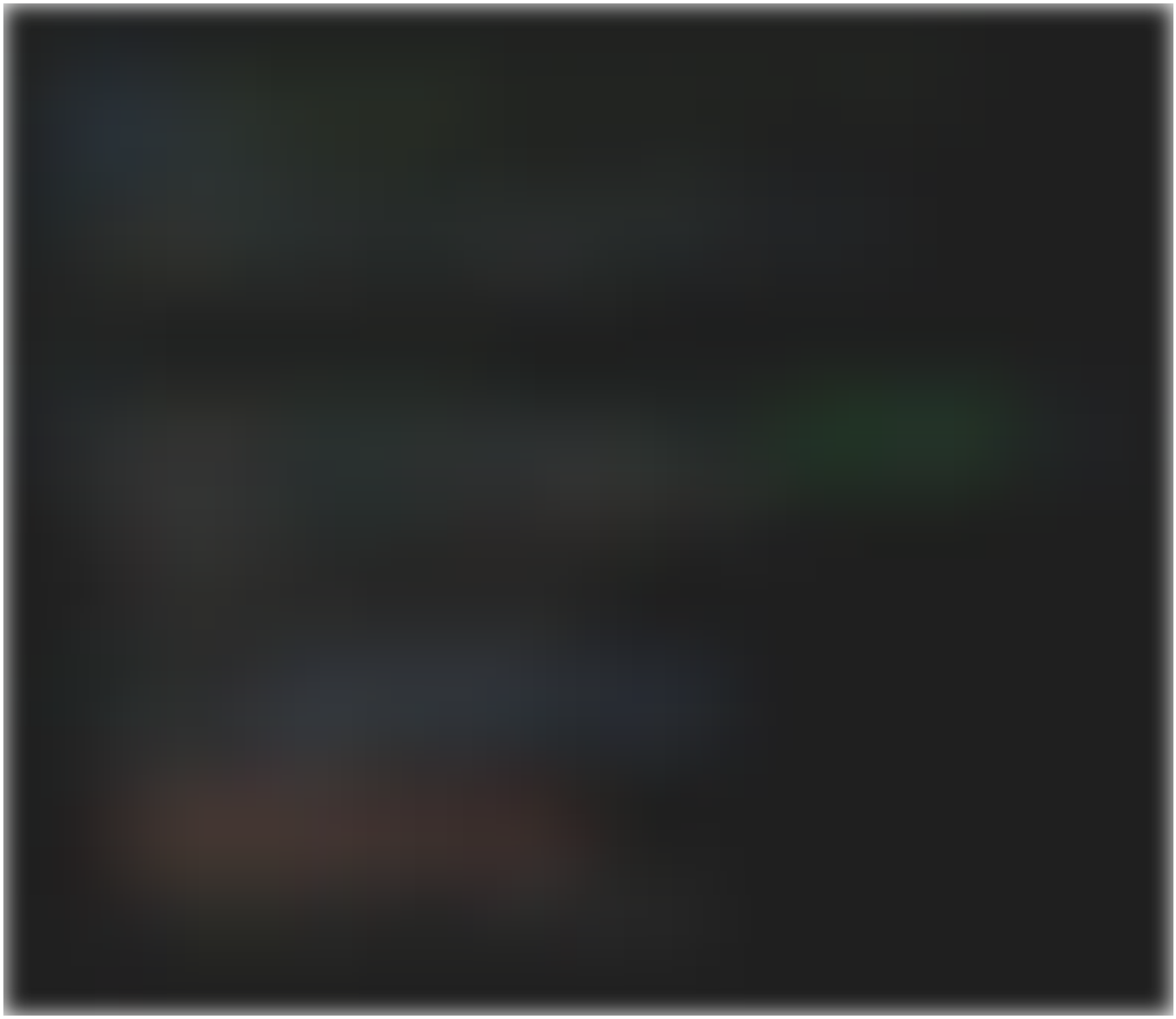
In summary baseTrackingIndex is the value of the index when the user last updated. baseTrackingAccrued is the total rewards owed to the user since they participated in the protocol (regardless of past claims which are negated with reward debt tracked in the reward contract.

What is accrualDescaleFactor?

baseTrackingAccrued is divided by 1e12 so that it effectively has the same number of "decimals" as USDC. This allows baseTrackingAccrued to track both assets on the same scale.

Claiming rewards

To claim rewards, a user simply calls the claim() function in CometReward.sol. The rewardsClaimed mapping (red box) behaves like the rewardDebt from MasterChef.



what is the shouldAccrue argument for?

If someone is claiming rewards as the only action in a transaction, then shouldAccrue (green box) should be true. However, if it is after other function calls, then other state-changing function calls will call accrueAccount() making the another call unnecessary.

getRewardAccrued() (CometRewards.sol)

In the blue box above, getRewardAccrued determining how much to pay the user. This simply queries the baseTrackingAccrued from the user struct in Comet. CometRewards then subtract it by their reward debt (rewardsClaimed) and pay the user the difference.

Recent Posts See All

Understanding the Function Selector in Solidity

16 0

How ERC721 Enumerable Works

175 0 2

Subscribe to our newsletter Subscribe Now

We do not sell your information to anyone. Period.

Web3 Blockchain Bootcamp