


Jeffrey Scholz  Jan 5 5 min read

# DeFi Interest Rate Indexes: Principal value and Present Value in Compound V3

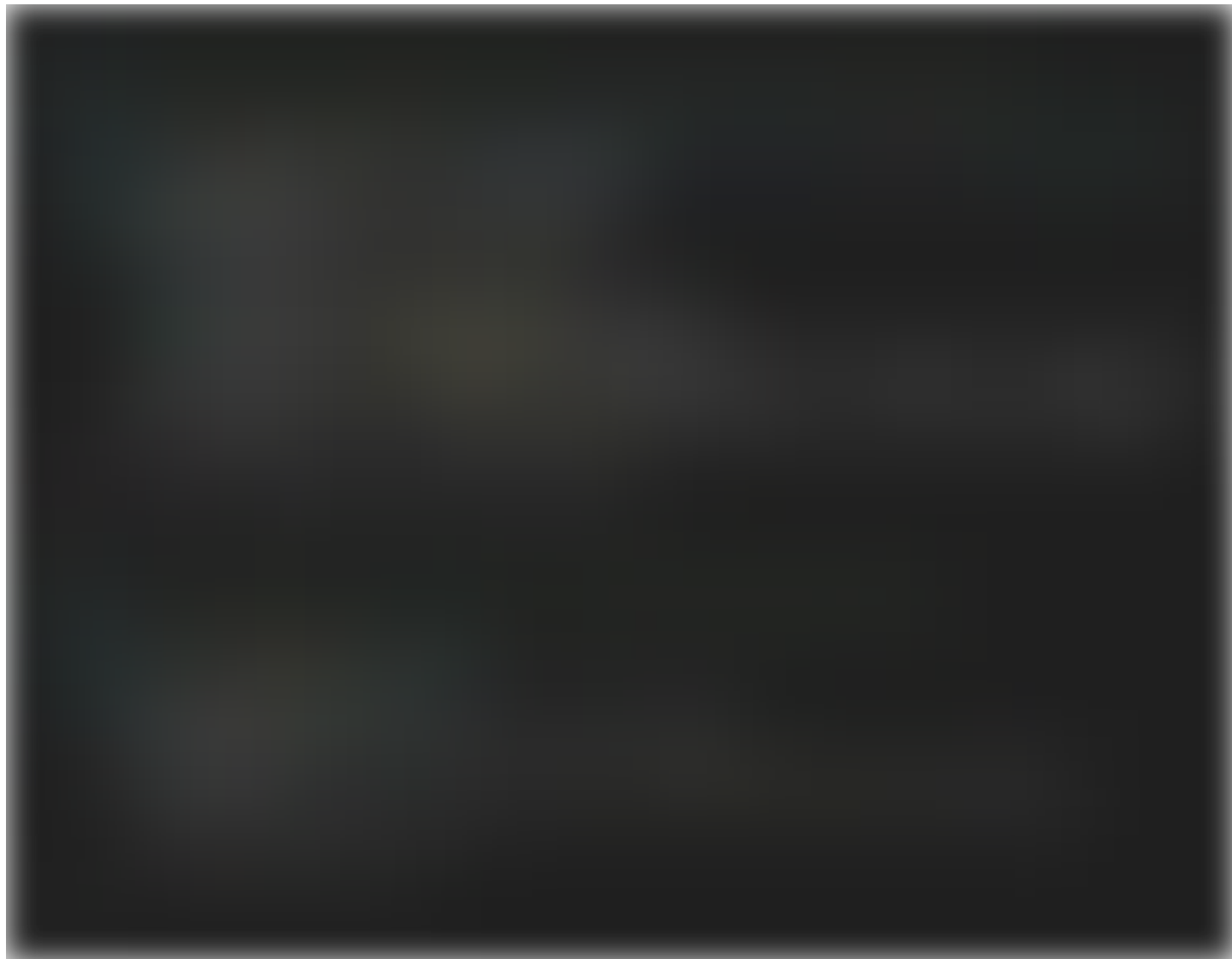
The intuitive way to track lender deposits is to record the amount of USDC they deposited and the time they deposited. **Compound V3 does not do this.**

Instead, similar to [SushiSwap Masterchef Staking Algorithm](#), Compound V3 tracks the hypothetical gain of one dollar lent “since the beginning of time.” (Readers not already familiar with this algorithm should read the linked resource).

The hypothetical gain of one dollar lent since the beginning of time is tracked in the [baseSupplyIndex \(in CometStorage.sol\)](#). It behaves very similar to to the “rewardPerTokenAccumulator” from Sushiswap. It starts at 1.0 and each time a state-changing operation occurs (deposit, withdraw, borrow, etc), it is increased proportional to the time passed and the interest rate over that period. For example, if 100 seconds passed, and the interest rate was 0.001 per second (unrealistically high, but easy to reason about), then baseSupplyIndex will be updated to 1.1. Specifically, the following formula is used:

```
baseSupplyIndex += supplyInterestRatePerSecond(utilization) ×  
secondsElapsed
```

The only place baseSupplyIndex is ever changed is on line [403 in Comet.sol](#), inside `accrueIndexIndices`.



**Because interest rates are a direct function of utilization, baseSupplyIndex increases faster during times of high utilization and slower during times of low utilization.**

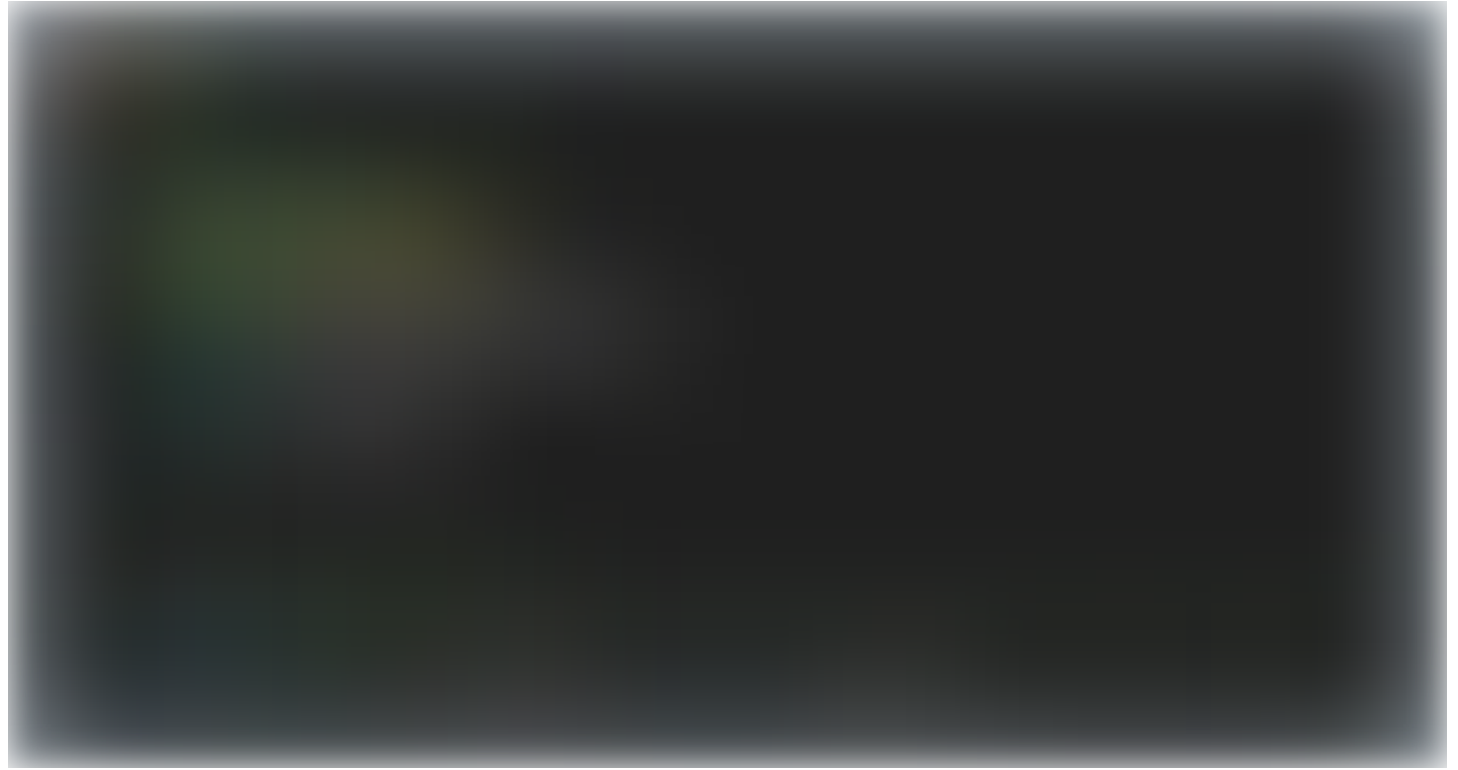
The following hypothetical plot shows the baseSupplyIndex and baseBorrowIndex increasing at different rates depending on the utilization. Generally, borrowers pay a higher interest than lenders are paid, so the baseBorrowIndex increases faster.



The following example illustrates how this variable is used.

## Example and Terminology

Alice deposits \$1,000 at a time when the `baseSupplyIndex` is 2.5. She is not credited with depositing \$1000, instead she is credited with depositing \$400, which is her deposit divided by the current `baseSupplyIndex` ( $\$1,000 \div 2.5$ ). Alice has a “principal value” of \$400 in her account (yellow box). This is the value Compound stores for users ([CometStorage.sol](https://www.cometstorage.sol)).



If she were to withdraw right away, Compound would calculate her balance as \$400 times the current baseSupplyIndex, which is 2.5, so she would withdraw \$1,000. Compound calls the principal value times the baseSupplyIndex the “present value”.

Compound V3 does not “remember” that her original real deposit was \$1,000. This is *implied* because the baseSupplyIndex is currently at 2.5 and her deposit is recorded as being \$400.

**The “downscaled” or “scaled backward” dollar value that Compound V3 stores is called the “principal value.” When we multiply the principal value by the current baseSupplyIndex, or “scale forward” we get the “present value.”**

Readers coming from a traditional finance background may find Compound’s use of the terms “principal value” and “present value” confusing – we suggest not trying to relate the terms to their traditional meanings and just accept Compound’s usage.

If she were to wait until the baseSupplyIndex increases to 3.0, the principal value would still be \$400, but the present value would increase to \$1,200 (\$400 x 3.0 = 1,200).



In [CometCore.sol](https://www.cometcore.sol), we see that:

1. the “principal value” is computed by **dividing** the “present value” by the baseSupplyIndex
2. the “present value” is computed by **multiplying** the “principal value” by the baseSupplyIndex.



So to summarize the important terms:

## Principal value

Principal value is the USDC deposited divided by the `baseSupplyIndex` value at the time of deposit. This is kept in a storage variable associated with the user's account and does not change unless the user deposits or withdraws. The principal value is generally less than the actual deposit because `baseSupplyIndex` is always greater than or equal to 1.

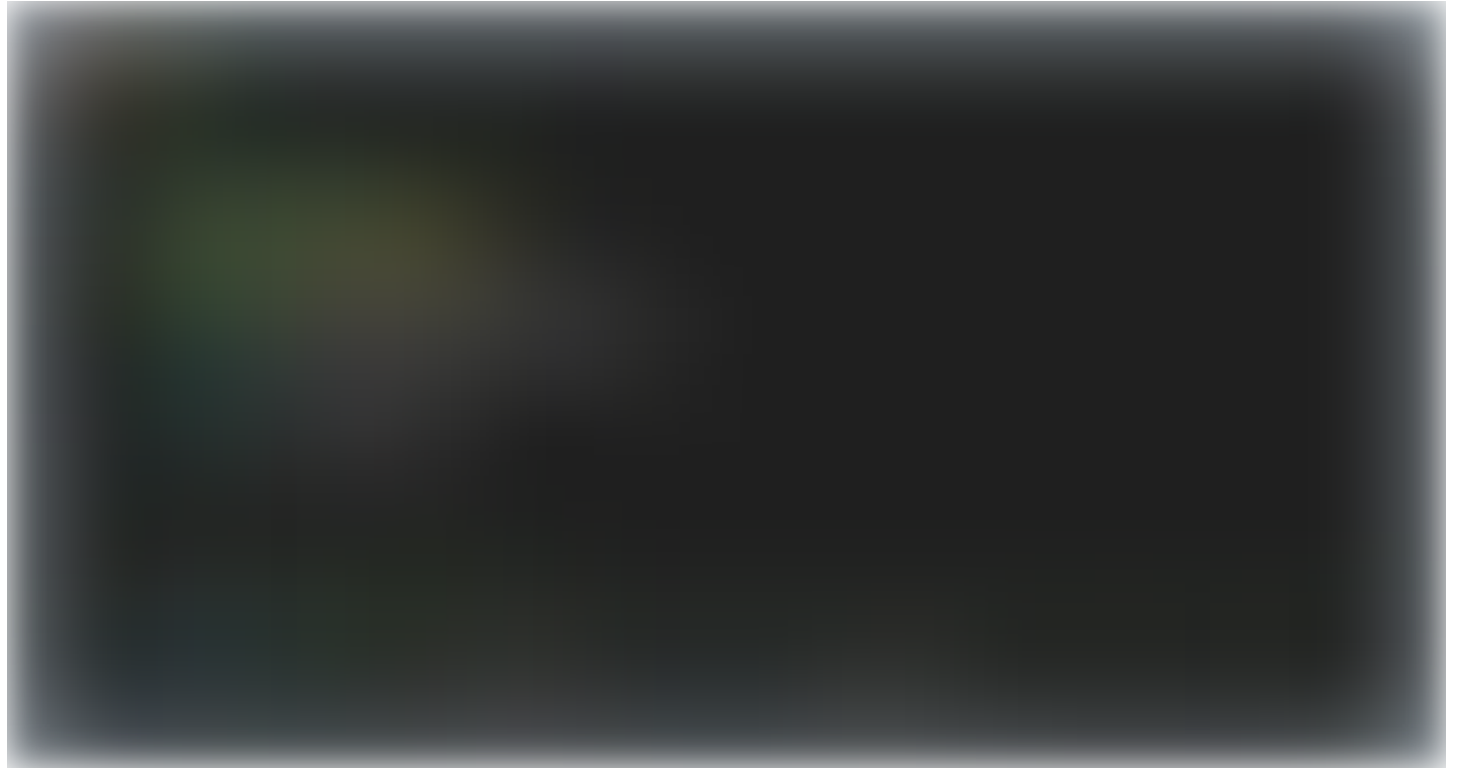
## Present value

The present value is the principal value times the **current value** of the `baseSupplyIndex`. This value is not stored anywhere but is computed on the fly.

**Principal value and present value are two of the most critical concepts for understanding Compound V3.**

## Principal is the only variable tracked for lenders

Let's revisit the struct we screenshotted above:



The variable `baseTrackingAccrued` is used by `CometRewards` to determine how much `COMP` to award the account for participation in the protocol. The variable `baseTrackingIndex` is related to that but currently unused. The variable `assetsIn` is only used for borrowers as an indicator for whether certain collateral assets are used. The variable `_reserved` is unused.

Therefore, principal is the only essential variable for lender accounting. Note that this is a signed variable – for borrowers it is negative. For borrowers, we also need to track how much collateral they have deposited, but that is for another article.

## **balanceOf() – checking the lender’s positive balance**

To illustrate that Compound V3 stores the principal value, but values the account at the present value, consider the [balanceOf\(\) function in Comet.sol](#) shown below.

First it will read the updated `baseSupplyIndex` without updating it, as this is a view function. Then it read’s out the principal balance of the lender and multiplies it by the `baseSupplyIndex`.

Interest accrued is a function of time and utilization, so as long as the utilization is not zero, each time `balanceOf` is queried, it will return a higher value.



The following screenshot shows the relationship between the lender's balance on the Compound V3 dapp and the balanceOf return value from Etherscan imposed on it for that same address. Both values highlighted in an orange box.



## Visual Example of baseSupplyIndex

When a lender deposits, the USDC balance of their deposit is divided by the baseSupplyIndex to produce the principal value, which is what gets stored. The later they deposit, the larger baseSupplyIndex will be, and they will be credited with a lower principal value.

The principal value is static (unless they deposit or withdraw), but the present value is the principal value \* baseSupplyIndex, and that is constantly increasing.

In the graphic below, Alice deposited \$1 when the baseSupplyIndex equaled 1.01, so her principal value is 0.99 ( $1 \div 1.01$ ). Bob also deposited \$1, but at a later time when the baseSupplyIndex had a value of 1.03 ( $1 \div 0.97$ ). Therefore, his principal value was 0.97.

**Bob and Alice both deposited the same amount: \$1. But because Bob deposited later, his principal value is lower.**



## supplyBase() and withdrawBase()

When a user supplies (or withdraws) the base asset, the principal value is reset. For example, if Alice deposited \$10 when the baseSupplyIndex was 10, her principal value would be \$1. Now suppose at a time when the baseSupplyIndex index increases to 20, her present value for her account would be \$20. She adds \$10 more. Her new *present value* should be \$30.

What should her principal value be? (Think about it for a second!)

The baseSupplyIndex is currently 20, the present value is currently \$30, so her principal value should be \$1.5 ( $\$30 \div 20$ ).

With this example in mind, we show the supplyBase() function from Comet.sol line 829 which is called when a lender deposits.





When Alice deposits, we read her principal value (orange box), convert it to present value (green box), and add the amount just deposited to it (blue box). This is converted from a present value to a principal value stored in `dstPrincipalNew` (yellow box) which we store to be the new principal value for that user (red box)

The function `updateBasePrincipal()` (red box) above simply updates the `UserBasic` struct associated with the user, overwriting the old principal with `dstNewPrincipal`.

The opposite function, `withdrawBase()` (Comet.sol line 1051), applies the same logic in reverse.

## What about baseBorrowIndex?

Similar to how `baseSupplyIndex` tracks the gain in one dollar lent since the beginning of time, `baseBorrowIndex` tracks the accumulation of debt for one dollar borrowed since the beginning of time. Lenders and borrowers have different interest rate curves, so two separate variables are needed to track them.

## When will the indexes overflow?

Both `baseSupplyIndex` and `baseBorrowIndex` are treated as fixed point numbers with 15 decimals, so `1e15` is treated as 1.0. The largest number a signed 104 bit number can hold is `1.014e31`. Therefore, the largest number the accumulator can hold is `1.014e16` (with 15 decimals).

**Let's assume that the lending protocol will never be over 100% APR. It would take 53 years for the index to overflow. Using a more realistic 10% interest rate, it would take 386 years for one dollar to compound to 10 trillion dollars.**

**It's pretty reasonable to assume that Compound will upgrade to version 4 before that day comes.**

## **Learn More with RareSkills**

**Please see our [Solidity Bootcamp](#) to learn more advanced Solidity concepts.**