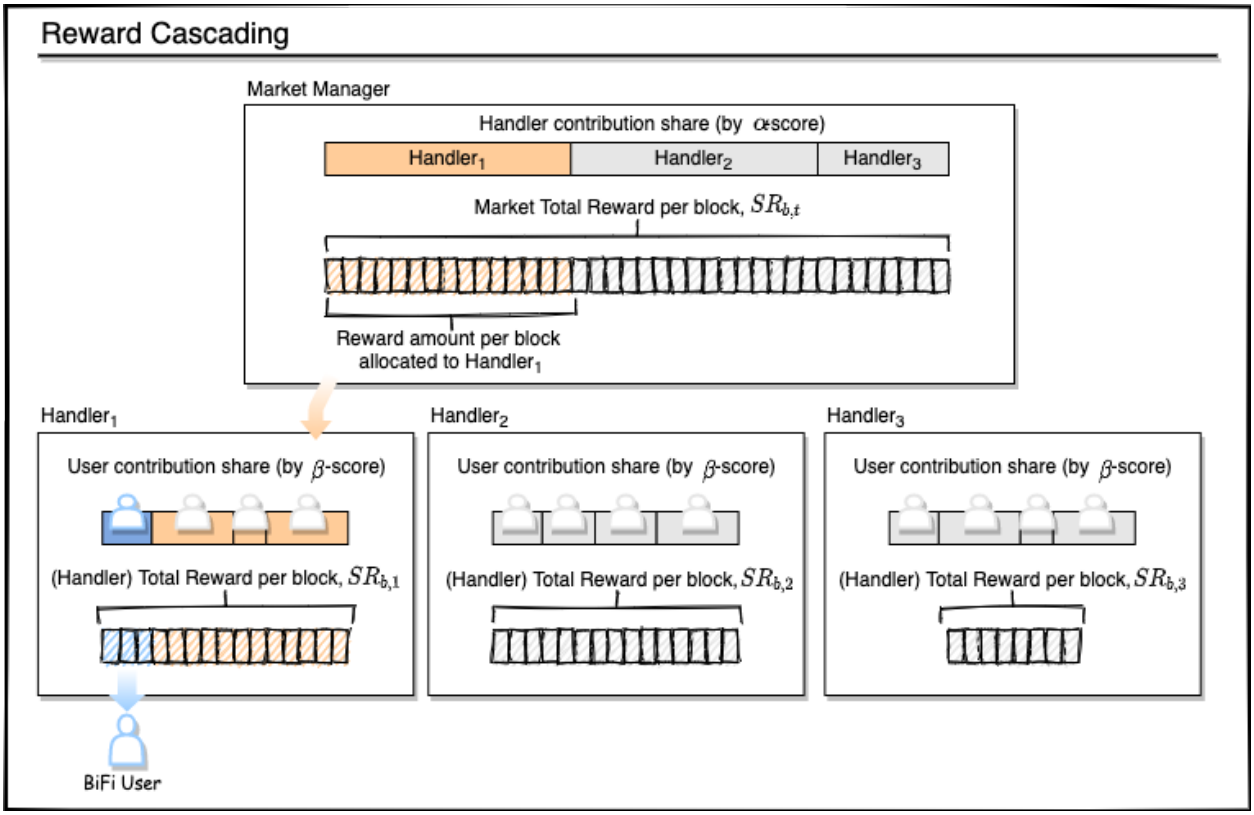
(ENG) Appendix I: Reward Method for Deposit and Borrow

1. Overview

BiFi calculates interest rates in real time based on the total amounts of deposits and borrows in the market. If there are more deposits, the deposit interest rates decrease and borrowing is incentivized. If there are more borrows, the borrowing interest rates increase and depositing is incentivized. Therefore, the deposits and borrows not only provide token liquidity, but also maintains the balance of the BiFi ecosystem.



BiFi rewards SI Tokens to users who deposit and borrow supported Tokens. Each Token has different amount of rewards allocated, set as Reward Parameters and determined by the α-score of each Token Handler (See *$Appendix\ III$: Reward Method for Reward Parameter Update*).

This allocation is then distributed to users according to the β-score, which is determined by their deposit and borrow amounts. For every block, the users receive rewards based on their β-score relative to the total β-score.

Terminology

The calculation and distribution of SI Token reward amounts is executed by Market Handler. Each Market Handler has the following variables.

Variables for β-score Calculation

- SR_b: Amount of SI Tokens distributed per block, based on β-score
- R_u: Amount of SI Tokens distributed per block, per β-score
- R_{user}: Amount of SI Tokens distributed to a user in a particular block
- L: Cumulative sum of R_u for all blocks
- β: Weight used to calculate β-score
- S_t: Total sum of β-scores of all users
- S_{user}: β-score of a particular user
- H_L: Block height of the last block when an action occurred
- H_C: Block height of the current block

2. Distribution Method

2.1 User β-score

$$score = \beta * (Deposit\ amount) + (1 - \beta) * (Borrow\ amount)$$

The β-score of a user is defined as above. The impact of deposit and borrow on the β-score can be adjusted by changing the weight (β).

2.2 Reward Distribution Method

In every block, the SI Token reward amount of each user is calculated as follows:

$$R_{user} = SR_b * \frac{S_{user}}{S_t}$$

In each block, each Handler distributes SR_b tokens to each user, proportionate to the β-score of each user.

3. Implementation

Ideally, the parameters, such as R_u, should be calculated in every block; which is triggered by user-action transactions. However, the assumption that there are user interactions in every block is unrealistic. Furthermore, in order to calculate the reward for inactive periods, we should store the changes of β-score for every single block. This leads to a huge storage overhead in the smart contract. Instead, we keep the cumulative sum of 'reward amount per unit β-score,' referred to as `RewardLane` for each block and tightly approximate the user rewards based on it.

3.1 `RewardUnit`, R_u

Definition

The amount of SI Token distributed per 1 β-score in each block, $R_u = \frac{SR_b}{S_t}$

Usage

- Calculation of Reward Amount: $R_{user} = SR_b * \frac{S_{user}}{S_t} = S_{user}\frac{S_b}{S_t} = S_{user}\frac{s_b}{S_t} = S_{user} * R_u$
- Reward amount can be calculated by multiplying a user's β-score (S_{user}) and R_u of every block.

3.2 `RewardLane`, L

Definition

Cumulative sum of all previous R_u, $L = \sum_i R_u$.

Batch Update for Inactive Periods

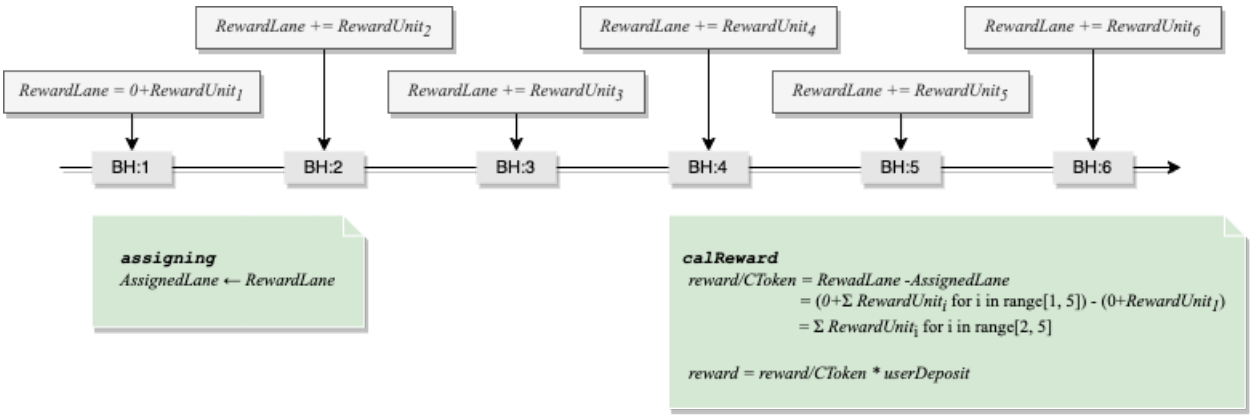
Before applying the current user action, the status of the service must be updated on the all previous user action. Fortunately it is simple for inactive periods; If there is no user action, R_u is not changed. Thus, we update L with the current value of R_u and the elapsed time (in block) between the previous and current actions.

Update Process

1. Calculate elapsed time (block height) : $\delta = H_C - H_L$
2. Update `RewardLane`: $L = L + \delta * R_u$
3. Update previous block height: $H_L = H_C$

- Note: If one block has multiple actions, it is sufficient for only the first action to update L.

3.3 `AssignedLane` and Reward Calculation



Handler assigns the current L to new users as well as users who already claimed rewards before on BiFi. The value of L for a user is referred to as `AssignedLane`. Using L and `AssignedLane`, Handler can calculate and distribute the accurate amount of rewards, whenever the user takes the an action.

$$reward = S_{user} * (L - AssignedLane)$$

The term "$L - AssignedLane$" represents the cumulative sum of R_u for all blocks between the previous and current actions. Therefore, users can claim all of their accrued rewards.