Phuture v2 Core

September 2023

Oliver Mehr
oliver@phuture.finance

Alex Melnichuk
alex@phuture.finance

Abstract

Phuture v2 is a non-custodial, cross-chain asset management protocol compatible with any EVM enabled network. When compared with the first iteration of the protocol, Phuture v2 introduces cross-chain asset management and trading capabilities; improved unit minting and redemption efficiency and a more optimised contract architecture reducing overall gas consumption.

1 INTRODUCTION

Passive investing through index products is a mainstay of traditional finance that has grown at a compounded growth rate of 16% since 2015 and now accounts for circa 43% of assets held by U.S.-based ETFs and mutual funds.[1] The average actively managed equity fund has underperformed its respective benchmark for 12 consecutive years, mainly due to higher fees, liquidity constraints and a diminishing edge.[2] The average crypto investor currently employs active management strategies in their portfolio. Our thesis is a simple one. As crypto markets mature, more investors will allocate to passive index strategies as a result of more efficient markets, making it harder to achieve excess performance above the benchmark.

Index funds created on today's non-custodial asset management protocols are constrained in the range of assets they can effectively support. This limitation is created through a single chain approach, which lacks the capability to account for and manage assets across multiple chains, narrowing the spectrum of eligible assets and strategies.

In this paper, we present Phuture v2, a novel on-chain asset manager that builds on the first iteration of the protocol and introduces a number of significant new features and improvements:

- *Multi-chain accounting* - Index products have the ability to price and account for assets across any EVM enabled network through Proof of Asset (PoA) tokens.
- *Multi-chain trading* - Buying and selling batches of constituent assets can occur across any EVM enabled network through dex aggregators, ensuring absolute best execution.
- *Contract-driven rebalancing* - Multi-chain rebalancing is orchestrated from a *homechain*, where trades are generated and executed on each respective chain in a trustless manner.
- *Optimised minting and redemption* - Multi-chain indices are minted atomically, without waiting for cross-chain messages, and can be redeemed atomically when sufficient reserves are present. Additionally, the optimised approach allows for slippage-free minting and redemption. Gas costs related to index issuance and redemption have undergone drastic improvements, making larger indices more cost effective.

2 RELATED WORK

This section evaluates the leading participants in the crypto asset management industry, identifying their shortcomings and detailing how Phuture v2 effectively addresses and mitigates these weaknesses.

Set Protocol is a permissionless, non-custodial asset manager which allows for the tokenisation of active or passive fund strategies. Each instantiation of Set's protocol is siloed to the chain it is deployed onto, restricting the universe of available assets and strategies. Furthermore, unit issuance and redemption via the basic issuance module, the default module for an index, has higher associated costs, in all cases, due to the requirement to hold each constituent asset. This makes it cost prohibitive to mint and redeem indices with large numbers of assets.

Balancer is a decentralised automated market maker that offers a variety of multi-asset liquidity pool types. Its managed pools are used for fund management with features that allow for adding, removing and changing the weights of assets. Balancer pools are regularly rebalanced by arbitrageurs, which make it impossible for them to follow an index strategy that is only rebalanced on a periodic basis. In fact, this will cause the Balancer-based index to underperform in a unidirectional market when compared to a Phuture index.

Adding or removing large proportions of single sided liquidity from the pool is inefficient due to the reliance on

[1] 2021.Bloomberg.com.`https://www.bloomberg.com/professional/blog/passive-likely-overtakes-active-by-2026-earlier-if-bear-market/`

[2] 2022.CNBC.`https://www.cnbc.com/2022/03/27/new-report-finds-almost-80percent-of-active-fund-managers-are-falling-behind.html`

internal liquidity; internal liquidity - liquidity held within Balancer - is always less than or equal to external liquidity - liquidity held across all exchanges. Relying solely on internal liquidity does not guarantee the best trade execution when entering or exiting the pool. Moreover, Balancer pools only support assets native to the chain the pool is deployed onto, limiting the available asset universe.

Alongside is a permissioned asset management protocol that enables the management of multi-chain indices. Alongside utilise a centralised custodian, Coinbase Custody, to hold their index assets, requiring explicit trust in the members that have access to the account. As a result of this centralisation, there is no permissionless minting and redemption restricting access to the primary market. Furthermore, since trading is executed from the confines of a custody account, it cannot be as trustless as trading via smart contracts.

Enzyme is a permissionless and non custodial asset management protocol that provides fund managers with the tools to create tokenised investment strategies on a single chain. It offers a wide range of functionality with numerous integrations. Enzyme's approach allows the manager to allocate new capital in whichever way they want and are not forced to conform to specific assets and weights. This is necessary for active strategies, but introduces more trustpoints for passive indexing strategies where weights and assets are always known.

Additionally, tokenised funds deployed on Enzyme are tethered to a single chain which removes their ability to support multi-chain assets and strategies.

Phuture v2's most significant upgrade is the ability to trade and manage assets across disparate blockchains from a single homechain for a broader selection of assets and strategies. Barriers associated with primary market operations are drastically reduced through the introduction of a reserve which reduces gas costs and enables slippage-free issuance and redemption. Trading on v2 is bound to the target assets and weights of the index at all times, removing trust in the index manager.

3 SYSTEM OVERVIEW

While all multi-chain crypto indices that exist today are centralised, Phuture v2 aims to deliver the first DeFi native solution to cross-chain asset management. Its architecture is designed chiefly to support indices at scale, across many chains while minimising trustpoints.

Figure 1 showcases a complete overview of the Phuture v2 protocol, introducing the fundamental components that will be referenced throughout this paper.

3.1 Homechain, Remote Chain and Omni Layer

- Any multi-chain index launched on Phuture v2 will have an established *homechain*, where unit issuance, unit redemption and rebalancing originate. The homechain is responsible for pricing by storing the quantities and accessing the price oracles of each constituent asset.

- An index can have a number of *remote chains* depending on how many blockchains the index spans. Each remote chain contains vault contracts which hold balances of constituent assets. In addition, each remote chain has an orderbook contract that receives and executes rebalancing orders from the homechain. These orders establish the assets and quantities that should be bought or sold on that chain.

- *The Omni layer* connects the homechain with the remote chains and is made up of one or several cross-chain messaging protocols. Interfacing with these messaging protocols is managed through the messageRouter contract. Each chain has a messageRouter contract which allows it to send and receive cross-chain messages.

3.2 Index Token

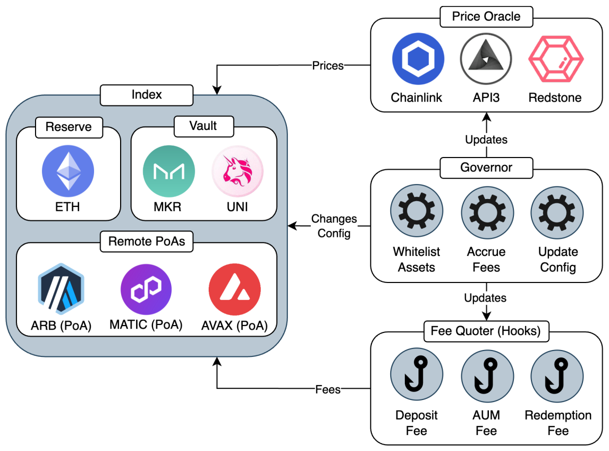


Figure 2: Index Contract Breakdown

The index token contract adopts a singleton architecture by inheriting from multiple smart contracts and embedding internal libraries. This achieves maximum efficiency by minimising the number of transfers and external calls required to execute key index functions like minting, redeeming and rebalancing. The contract keeps track of the Net Asset Value, underlying asset quantities on the home and remote chains, allows new assets and weights to be assigned, and sets the fees that the index charges.

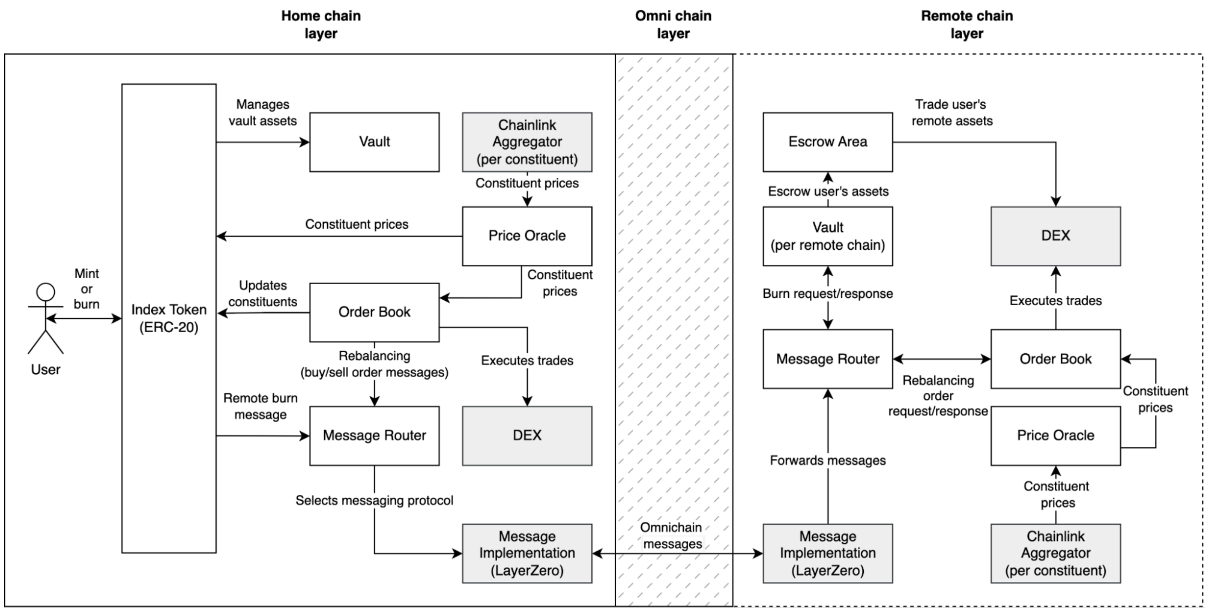


Figure 1: System Overview

3.3 Orderbook

The orderbook contract generates and executes trades during rebalancing. Each chain that the index operates on will have an orderbook contract. All orders originate from the homechain orderbook, which sends buy/sell orders to each remote chain.

4 CROSS-CHAIN MESSAGING

Phuture v2 plugs into a generalised cross-chain messaging protocol, which allows for the passing of messages and execution of arbitrary logic across blockchains.

4.1 Selecting a Messaging Protocol

A number of these protocols exist each with a different set of security and trust assumptions. We evaluate the most popular protocols including Layerzero, Wormhole, Axelar and Celer based on the following criteria:

- Security - How is message validity assured?
- Upgradeability - Can our security be impacted by protocol upgrades?
- Token Bridge - How liquid is the native token bridge and does it rely on wrapped assets?

Security. Broadly speaking each bridge analysed uses a set of validators or signers to attest to the existence of a transaction within a specific block. Axelar and Celer use Cosmos based blockchains whose validators, elected by delegated proof of stake (DPOS), attest to the validity of a transaction. Wormhole has 19 participants in its guardian network of which 13 are required to successfully validate a transaction. While Layerzero requires authentication from both a relayer and oracle which can each consist of multiple signers.

The native token for Axler and Celer have relatively low market caps, such that a sufficiently motivated actor could obtain a significant share of the votes, particularly in Celer, where quadratic voting is not implemented. However, unlike Wormhole and Layerzero, the slashing mechanism used in DPOS creates a tangible cost to signing invalid transactions outside of reputational damage.

When looking at the number of signers required to validate a transaction, Layerzero requires the fewest signatories in its default form, requiring 2/2, which makes it the most susceptible to collusion.

Upgradeability. Upgradability has been a longstanding source of significant and repeated issues within the existing messaging frameworks. Wormhole has paid out two separate bug bounties due to upgradeability issues and ultimately got hacked for over \$325mm in 2022 due to another bug found in the codebase. The most recent hack of the Nomad bridge which occurred in the summer of 2022, was due to an upgrade which introduced a bug that allowed anyone to forge arbitrary messages and resulted in the exploit of more than \$200mm.

All the aforementioned messaging protocols except Layerzero have upgradeable contracts that allow the owner to upgrade core components of the protocol moving all users and applications to these upgraded contracts. On the other hand, Layerzero allows applications to stipulate the specific libraries they intend to use and once set, cannot be modified by the Layerzero team. This separation provides time for new library updates to be checked before, if deemed necessary, including them into our production application.

Token Bridge. A token bridge will allow Phuture v2 to move funds from one chain to another. We reviewed Stargate, Portal, cBridge and Satellite, the native token bridges of Layerzero, Wormhole, Celer and Axelar, respectively. These bridges can be broken into two categories: native bridges (cBridge and Stargate) and wrapped bridges (Satellite and Portal). Simply put, native bridges use native assets as their input and outputs, while wrapped bridges take in native assets and output a wrapped variant of that asset, which can be swapped through a liquidity pool for the native token. In either case, high liquidity is of paramount importance because it reduces the slippage incurred when bridging assets.

Stargate is the most liquid bridge with a total of $380mm in total value locked, providing the best coverage of the primary connecting assets, ETH, USDC and USDT, across the major chains. While Wormhole has the second highest liquidity, the majority of this is held on Solana, which would not be a supported chain in the initial release of Phuture v2, and its support for bridging assets to Optimism and Arbitrum is non-existent. Satellite and cBridge have wide coverage of the major chains but have liquidity levels that are much lower and less uniformly distributed than Stargate. It should also be noted that because Stargate is a native bridge, it removes the necessity to perform an additional swap on the destination chain.

Summary. After analysing each generalised messaging protocol, we decided to use Layerzero because it allows us to take control of our bridge configuration, effectively separating it from any config updates pushed by the Layerzero team; we can drastically reduce any risk of collusion by running our own oracle or relayer in the future; and their token bridge uses native assets with the highest liquidity.

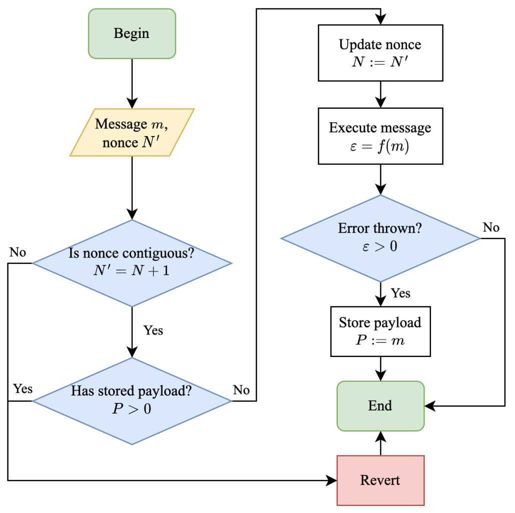
However, the analysis also highlighted that there is no clear winner amongst the cross-chain messaging protocols, and so locking Phuture v2 into a single vendor is not desirable. Therefore we built Phuture v2 to support any messaging protocol with the ability to switch should we need to. This functionality is encapsulated in the messageRouter contract, which we discuss in the next section.

4.2 Message Router

The messageRouter forwards messages to the correct bridge implementation which is set based on the action type, such that different actions, like rebalancing or redeeming, can be channelled to different messaging protocols. For example, one action may use the non-blocking LayerZero application, while another uses the blocking LayerZero app. This gives us enough flexibility to decide how and through which protocol our messages get transported.

4.3 Blocking vs Non-blocking Messages

Layerzero supports both blocking (default) and non-blocking messaging formats. All messages m are nonce-ordered, meaning they will arrive from a source chain and source user application with contiguous nonces N'. Messages with non-contiguous nonces cannot be processed and are queued up for execution in order to keep the sequential/non-parallel nature of the message queue. In the blocking format, when a message fails due to a logical or out of gas error, it is registered as a stored payload P. This stored payload prevents the next message from the source chain from being executed on the destination chain. A stored payload will only block messages for a given pair of chains and will still allow other pairs to communicate. For example, if a message from Arbitrum to Optimism is registered as a stored payload, messages from Ethereum to Optimism and messages from Optimism to Arbitrum will still get executed successfully.



Begin
Message m, nonce N'
Is nonce contiguous? $N' = N + 1$
No
Yes
Has stored payload? $P > 0$
Yes
No
Update nonce $N := N'$
Execute message $\varepsilon = f(m)$
Error thrown? $\varepsilon > 0$
No
Yes
Store payload $P := m$
End
Revert

Figure 3: Blocking Queue Flow

The non-blocking messaging format allows messages to be passed freely, irrespective of failed messages. Failed messages are stored and can be retried but do not prevent subsequent messages from being executed.

Phuture v2 adopts the blocking format of messaging to ensure that messages are executed in the order they are confirmed on the destination chain. This simplifies the design as messages will either successfully complete or will block the message pathway preventing other messages from being executed.

4.4 Message Batching

Phuture v2 is built to support asset management at scale, and it therefore assumes the possibility that messages will carry a lot of data. For example, a single message to a remote chain may contain data to trade 15 assets. Messaging protocols like Layerzero enforce a 10,000 byte limit for each message. Therefore, message batching ensures that each message sent is below the byte limit by splitting the data across multiple messages.

4.5 Message Gas Configuration

Given the understanding that Phuture v2 uses blocking messages, it is highly important that we properly estimate the amount of gas used by a given message to ensure that it does not block the message pathway due to an out of gas error. Phuture v2's gas configurations ensure that a message is given the appropriate amount of gas.

Messages start on the source chain β_s. Two types of messages can be sent:

- *One-way messages* - Only send source-to-destination messages without expecting a callback.

- *Two-way messages* - Send source-to-destination message with intention to receive destination-to-source callback message.

Notation	Definition
T_V	Transaction value.
β_s	Source chain id.
β_d	Destination chain id.
x_d	Source-to-destination data.
x_c	Destination-to-source callback data.
g_{min}	Minimum gas required to execute source-to-destination message on the destination chain.
g_d	Extra destination gas optionally passed in addition to g_{min}.
g_c	Callback gas, required to execute destination-to-source message callback.
ϕ_σ	Markup added to g_c to account for gas price volatility.
$\Phi(\beta, g, x, \alpha)$	Returns LayerZero fee in source chain token required to send data x with gas g to the chain β with airdropped amount α.
Φ_d	Destination message fee in the source chain's native token.
Φ_c	Callback message fee in the source chain's native token.
p_s	Source chain native token price.
p_d	Destination chain native token price.

Table 1: Gas Configuration Parameters

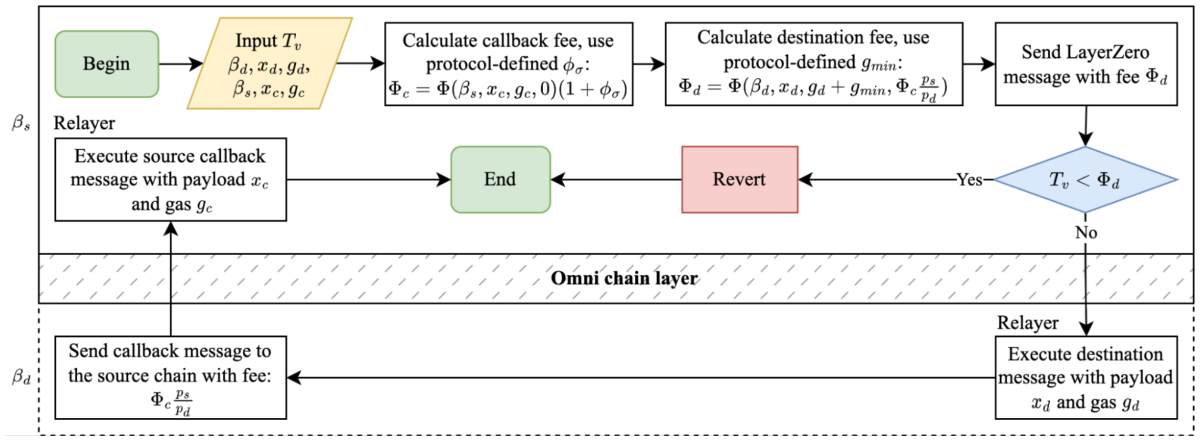

Begin
Input T_v $\beta_d, x_d, g_d,$ β_s, x_c, g_c
Calculate callback fee, use protocol-defined ϕ_σ: $\Phi_c = \Phi(\beta_s, x_c, g_c, 0)(1+\phi_\sigma)$
Calculate destination fee, use protocol-defined g_{min}: $\Phi_d = \Phi(\beta_d, x_d, g_d + g_{min}, \Phi_c \frac{p_s}{p_d})$
Send LayerZero message with fee Φ_d
β_s
Relayer
Execute source callback message with payload x_c and gas g_c
End
Revert
Yes
$T_v < \Phi_d$
No
Omni chain layer
Relayer
β_d
Send callback message to the source chain with fee: $\Phi_c \frac{p_s}{p_d}$
Execute destination message with payload x_d and gas g_d

Figure 4: Cross Chain Gas Lifecycle

LayerZero charges a fee Φ in the native token for each cross-chain message. Message data payload x, gas limit g and destination chain id β are required to calculate the fee.

The message sender fully covers the cost of the message. In order to calculate the total message fee Φ_d, the cost of executing the message on the destination chain φ_d and the cost of calling back the source chain from the destination chain φ_c must be found.

Destination-to-source callback fee Φ_c is zero for one-way messages. Destination-to-source callback is executed after the source-to-destination message is delivered to the destination chain. Since the message delivery might take time, gas prices can change. A markup, ϕ_σ, is added to the estimated callback fee to ensure that the callback is executed successfully.

$$\Phi_c = \Phi(\beta_s, x_c, g_c, 0)(1 + \phi_\sigma) \tag{4.5.1}$$

Source-to-destination message requires the sender to pay the minimum gas requirement g_{min} for executing the message data on the destination chain. g_{min} ensures that the message will be executed successfully without blocking the message pathway. It is enforced at the contract level, such that a malicious actor cannot deliberately block the message pathway by sending a message with value less than g_{min}. Senders can add extra gas g_d to g_{min} if the destination transaction is expected to cost more gas than g_{min}.

$$\Phi_d = \Phi(\beta_d, x_d, g_d + g_{min}, \Phi_c \frac{p_s}{p_d}) \tag{4.5.2}$$

It is important that the gas consumption of the actions g_{min} covers can be reliably estimated, with little to no variation, to ensure that messages are consistently delivered successfully without blocking the pathway.

5 ACCOUNTING

In this section we discuss the mechanics behind index pricing, minting, redemption and fees.

5.1 Asset Pricing

Phuture v2 utilises a number of different price feed providers in order to price each asset within the index. Any asset added to an index must have a price feed on the homechain and its native chain. This ensures that the index can be priced atomically on the homechain, and slippage protection is applied during trading on the native chain.

5.2 NAV

An index's price, also known as the Net Asset Value (NAV), is required when minting or redeeming units of the index.

$$\text{NAV} = \frac{\text{Asset} - \text{Liabilities}}{\text{Total Number of Outstanding Shares}} \tag{5.2.1}$$

For the scope of this paper, we are going to focus on fund structures that do not have any liabilities, which simplifies the equation to:

$$\text{NAV} = \frac{\text{Asset}}{\text{Total Number of Outstanding Shares}} \tag{5.2.2}$$

First, we calculate the portfolio value V for n constituent assets with reserves C and prices p for each constituent asset i.

$$V = \sum_{i=1}^{n} C_i p_i \tag{5.2.3}$$

Phuture v2 uses a reserve C_r model that allows unit minting and redemption without buying or selling the constituent assets. This is analogous to a cash balance that the index holds, which accumulates when index tokens are minted and is withdrawn from during redemptions. Thus the portfolio value is now:

$$V = C_r p_r + \sum_{\substack{i=1 \\ i \neq r}}^{n} C_i p_i \tag{5.2.4}$$

Total supply S is the sum of outstanding index shares. The total supply increments each time new index shares s_m are minted for all mints M and decrements when shares s_b are redeemed for all redemptions B.

$$S = \sum_{i=1}^{M} s_{m,i} - \sum_{j=1}^{B} s_{b,j} \tag{5.2.5}$$

Index price P is the portfolio value V divided by the total supply S.

$$P = \frac{V}{S} \tag{5.2.6}$$

It is important to note that all components of P can be retrieved from the homechain such that the NAV can be found atomically.

5.3 Protocol Fees

Phuture v2 allows for the accrual of the following fees: minting, redemption and AUM.

5.3.1 Minting Fee

Minting fee ϕ_m is deducted from the minted units s_m. Minted shares received after deducting minting fee is

$$s' = s_m(1 - \phi_m) \tag{5.3.1}$$

The minting fee is settable by the index manager and has two purposes:

1. Compensate the index manager for administering the index.
2. Allow the index manager to add a premium to the NAV. A premium may be added to the price at which new units are minted to prevent the index from being exploited. Exploits can occur when the price at which the index is minted is below the actual NAV. This can occur due to delayed price feeds.

5.3.2 Redemption Fee

The redemption fee ϕ_b is deducted from the redeemed shares s_b. Redeemed number of shares after deducting redemption fee is

$$s' = s_b(1 - \phi_b) \tag{5.3.2}$$

The redemption fee is settable by the index manager and has two purposes:

1. Compensate the index manager for administering the index.
2. Allow the index manager to add a discount to the NAV. A discount may be added to the price at which new units are redeemed to prevent the index from being exploited. Exploits can occur when the price at which the index is redeemed is above the actual NAV. This can occur due to delayed price feeds.

5.3.3 AUM Fee

AUM fee is a function $\phi_a(t)$ of time t. ETFs typically deduct their management fee from the NAV on a daily or monthly basis. The expected decay in portfolio value V can be expressed with a rate r and t number of compounding periods.

$$V(1-r)^t \tag{5.3.3}$$

The AUM fee cannot be accrued once per year because portfolio value V can drastically change due to mints or redeems at any time. AUM should be calculated and charged before V changes. For example, if a user deposits without AUM accrual, the newly deposited assets would be accounted for as a part of the previous fee accrual time interval resulting in more fees taken than necessary. If a user redeems without AUM accrual, capital would leave the index without accruing fees for the last time interval.

Instead of separately accruing AUM from constituent quantities C_i, it is more gas efficient to dilute the total supply S. Each new accrual mints ΔS dilution. This dilution includes previous dilutions, resulting in the continuous compounding of S.

$$\frac{S}{(1-r)^t} = \frac{S}{e^{-xt}} \tag{5.3.4}$$

By simplifying the expression above and solving for x, a continuously compounded rate x is recovered and that will equal the simple rate of r, ensuring that the AUM fee is not overcharged due to compounding.

$$x = -ln(1-r) \tag{5.3.5}$$

A continuously compounded total supply dilution rate per second $\frac{x}{T}$ is precomputed off-chain and passed on-chain as a constant to avoid extra computations. Since the EVM calculates time in seconds, x should be normalised by dividing by the average number of days in a year, converted to seconds T.

$$T = 365.2425 \cdot 86400 = 31556952$$

Last AUM accrual timestamp t_l is reset each time the AUM fee is charged.

$$t'_l = \begin{cases} t, & \text{if}\, t > t_l. \\ t_l, & \text{otherwise.} \end{cases} \tag{5.3.6}$$

The final AUM fee function is thus given by

$$\phi_a(t) = e^{\frac{x}{T}(t-t_l)} \tag{5.3.7}$$

Minted total supply dilution ΔS is

$$\Delta S = S(\phi_a(t) - 1) \tag{5.3.8}$$

Total supply after AUM fee accrual

$$S' = S\phi_a(t) \tag{5.3.9}$$

The AUM fee is settable by the index manager.

5.4 Index Minting

Phuture v2 introduces atomic, slippage-free minting of multi-chain index funds. Instead of requiring each constituent asset to be bought during the minting process, a reserve asset c_r is deposited into the index, and new units are created based on the NAV. During this process, minting and AUM fees are charged. The amount of minted shares s for a given deposited amount net of minting fee c'_r is:

$$s(c'_r) = \begin{cases} c'_r p_r, & \text{if}\, S = 0. \\ c'_r p_r \frac{S'}{V}, & \text{otherwise.} \end{cases} \tag{5.4.1}$$

Once a given quantity of reserve asset has accumulated in the index, they will be converted into constituent assets via a process called reserve rebalancing. *Reserve rebalancing* converts the available quantity of C_r held by the index, into the current set of constituent assets using the current target weights.

The reserve model described above drastically reduces the gas costs of minting new index units and removes the need to execute swaps. Lowering costs promotes more arbitrage, ensuring the exchange price closely tracks the NAV and allows more people to access the primary market where they can take advantage of slippage-free execution.

It is worth noting that not immediately purchasing the constituent assets can cause the NAV to deviate from the benchmark that it is tracking. However, this deviation can largely be mitigated by having more frequent investment cycles or initiating a new cycle once the ratio of reserve asset to portfolio value reaches a certain threshold.

5.5 Index Redemption

When units of the index are redeemed, the protocol first tries to fully cover the value of redeemed shares s with the value held in reserve token C_r. In cases where C_r has sufficient value, the redemption can complete without selling any constituent assets.

Constituents are sold only if there is insufficient value in C_r. This is done for optimisation purposes:

- *Gas costs* - Withdrawing multiple constituents requires more gas to be paid for transfers and internal accounting.
- *Cross-chain overhead* - Cross-chain messages charge bridging fees and take longer to execute due to their non-atomic nature.
- *Fees* - Swapping multiple assets incurs trading fees.
- *Performance* - Reserve growth causes the index to stray away from its benchmark. Having less reserve ensures that the index more strictly follows its intended strategy.

Total number of reserve shares S_r available for redemption is recovered by expressing the available reserve tokens C_r in terms of index shares.

$$S_r = C_r p_r \frac{S'}{V} \tag{5.5.1}$$

Redeemed shares cannot exceed the total supply $s \in \{0, ..., S\}$. The amount of reserve shares s_r withdrawn cannot exceed the total amount of reserve shares S_r.

$$s_r = min(s', S_r) \tag{5.5.2}$$

Withdrawn reserve assets c_r convert from shares to reserve asset.

$$c_r = \begin{cases} 0, & \text{if } S_r = 0. \\ s_r \frac{C_r}{S_r}, & \text{otherwise.} \end{cases}$$

In case the reserve was insufficient to cover redeemed shares s', the remaining shares $(s' - s_r)$ would be used to proportionally withdraw from each index constituent i, with total quantity C_i. Total supply S must be reduced to remove the previously redeemed shares s_r.

Withdrawn constituent quantity c_i for constituent i:

$$c_i(s') = \begin{cases} 0, & \text{if } S' - s_r = 0. \\ (s' - s_r)\frac{C_i}{(S' - s_r)}, & \text{otherwise.} \end{cases} \tag{5.5.3}$$

Withdrawn asset value v is, therefore, a combination of withdrawn reserve token plus the sum of each withdrawn index constituent.

$$v = c_r p_r + \sum_{\substack{i=1 \\ i \neq r}}^{n} c_i p_i \tag{5.5.4}$$

5.5.1 Redemption Escrow

Each user is given their own escrow account on each remote chain when they redeem index tokens. The escrow account is a holding area for funds which were not successfully sent back to the homechain. This can happen because of a trading related error or an out of gas error. The escrow account allows for transactions to gracefully fail by transferring funds to the escrow and preventing the error from blocking the message pathway. Funds held in the escrow can be returned to the homechain upon the user's request.

The minimum gas to invoke the redeem function, g_{min}, is set to ensure there is always enough gas to escrow the assets on each chain the index operates on.

5.5.2 Two Stage Redeem

Redemptions should complete in a single transaction. However, if another mint or redeem transaction is in the same block, there is a case where it will span two transactions. This is attributed to the fact that redemptions first utilise the reserve before selling constituents. If the reserve changes intra-block then the quantity of constituents sold can increase or decrease. However, the data passed in from the exchange aggregators will not adjust for this change in quantity and, therefore, will try to sell too little or too much of each constituent.

To handle this case, the redeem function takes in the ratio of reserve value to total constituent value as an additional parameter. If the ratio is the same as what is computed on-chain, then it means that the state of the index is unchanged, and a single transaction will suffice. If the ratio has changed, the swap data passed in will be incorrect. To avoid wasting gas, the first transaction withdraws each constituent asset to the user's escrow account without executing any swaps. The second transaction converts and bridges these assets back to the homechain.

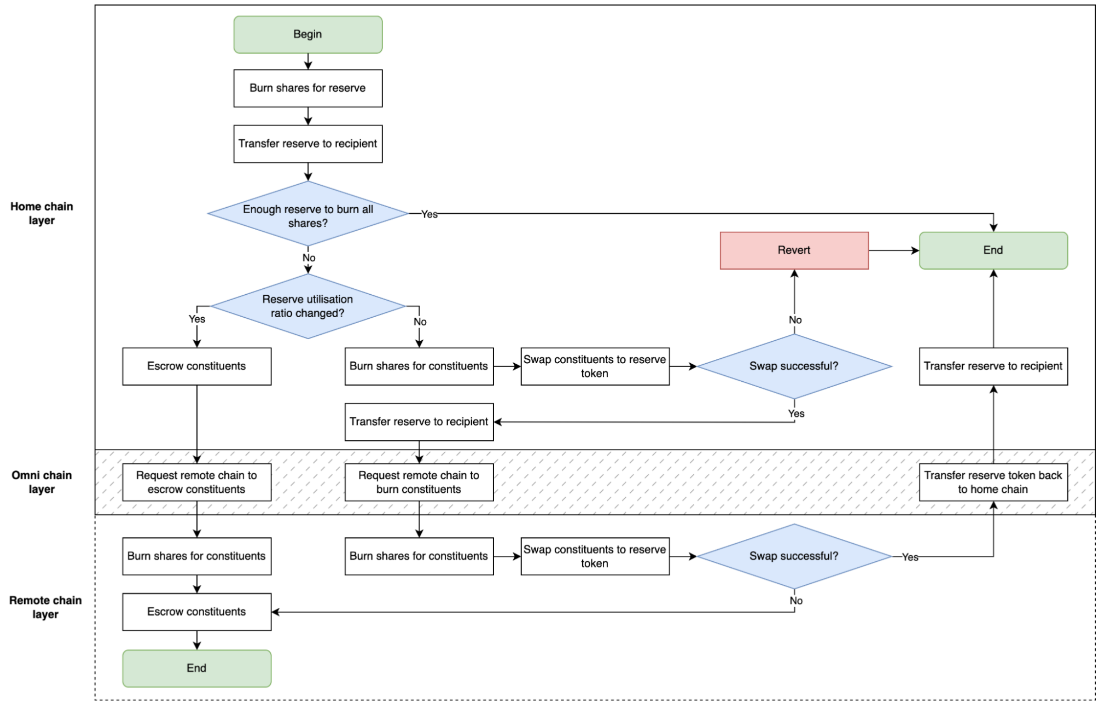

Home chain layer
Omni chain layer
Remote chain layer
Begin
Burn shares for reserve
Transfer reserve to recipient
Enough reserve to burn all shares?
Yes
No
Reserve utilisation ratio changed?
Yes
No
Escrow constituents
Burn shares for constituents
Swap constituents to reserve token
Swap successful?
Revert
End
Transfer reserve to recipient
Transfer reserve to recipient
Request remote chain to escrow constituents
Request remote chain to burn constituents
Transfer reserve token back to home chain
Burn shares for constituents
Burn shares for constituents
Swap constituents to reserve token
Swap successful?
Escrow constituents
End

Figure 5: Index Redemption Flow

6 PROOF OF ASSETS

Proof of Assets (PoAs) are accounting representations of the native tokens held on the remote chains. PoAs are stored on the homechain to account for the quantity of each asset held on the remote chains. PoAs are not liquid - they are used for accounting purposes only and are not designed to be traded or utilised outside of the protocol.

PoAs are updated by sending remote-to-home chain messages. PoA balances are refreshed:

- Upon completion of rebalancing.

- Upon completion of reserve rebalancing.

Updating the PoAs stored on the homechain each time balances on the remote chains change allows the homechain to accurately calculate the NAV of the index.

7 SUB INDICES

Internally, multi-chain indices built on Phuture v2 consist of smaller *sub-indices*, each *sub-index* contains the asset vault for a particular remote chain. A sub-index has both a homechain and remote chain representation. On the homechain, each sub-index holds a set of *Proof of Assets* (*PoAs*), while the remote chain sub-index holds the native asset vault.

Sub-index shares represent percentage ownership of the underlying asset vault. Therefore, a user will own a percentage of the sub-index shares proportional to their ownership of the index's outstanding tokens.

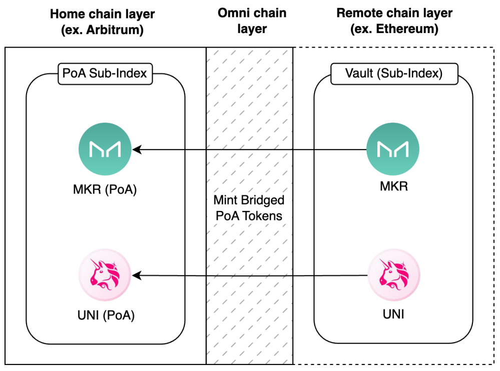


Figure 6: Sub-Index

Structuring multi-chain indices as an index comprised of a number of sub-indices is beneficial for two mains reasons:

- *Reducing storage costs through aggregation of operations* - Sub-index-based approach reduces the algorithmic complexity from $O(\text{constituents})$ to $O(\text{remote chains})$. Accounting for all index constituents is costlier than accounting for sub-indices.

- *Reducing cross-chain bandwidth* - Operating with sub-indices helps to reduce cross-chain bandwidth. Since each remote chain is aware of all the assets it holds, no explicit information about constituents and their quantities need to be passed to the remote chain.

7.1 Sub-Index Snapshots

Snapshots have been designed to keep track of a sub-index's assets and balances over time. Each time any form of rebalancing is conducted a new snapshot is created with the sub-index's new state.

Snapshots were developed to solve for a lack of transaction atomicity in cross chain systems, due to the possibility that a message can successfully send on the homechain but fail on the destination chain due to an error. Therefore, snapshots allow Phuture v2 to recover a sub-index's previous state and properly account for failed transactions when they are eventually retried.

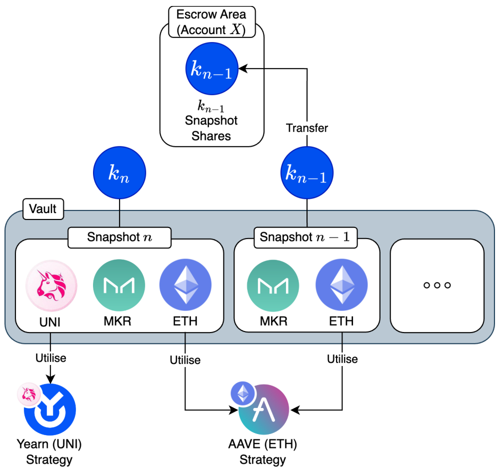


Figure 7: Sub-Index Snapshots

To showcase snapshots, we will take a look at a failed redemption transaction. When the redemption transaction fails the shares of the current sub-index snapshot, k_{n-1} are transferred to the user's escrow account. The user does not immediately retry the transaction and a new snapshot is created, n, as a result of a rebalancing. When the new snapshot is created all funds owed to the k_{n-1} snapshot shares held by the index are transferred over, leaving the portion of assets owed to escrowed k_{n-1} snapshot shares. When the user eventually redeems they will access the funds remaining in the $n-1$ snapshot, ensuring they only receive the assets and quantities they would have originally received had their redemption transaction initially succeeded.

7.2 Sub-Index Accounting

The sub-index architecture scales the underlying PoA balances in order to recover the actual quantities held on the remote chain. Initially, a sub-index's numerator, k, is set to the same value as the denominator K.

$$k := K \tag{7.2.1}$$

Therefore, the initial scale factor of the sub-index is 1, which means the PoA balances are equal to the quantities held on the remote chain.

$$\textit{Scale Factor} = \frac{k}{K} = 1 \tag{7.2.2}$$

When the protocol processes a redemption, instead of decrementing the balance of each PoA held by the sub-index, the numerator k of the scaling factor is decreased by the percentage of the index redeemed.

$$k := k(1 - \frac{s' - s_r}{S' - s_r}) \tag{7.2.3}$$

By adjusting the scaling factor, we can decrement the balances of all PoAs held through a single storage write, reducing the accounting related gas costs during redemptions.

$$k\frac{C_{PoA}}{K} \tag{7.2.4}$$

Whenever a sub index's PoA balances are refreshed through rebalancing or reserve rebalancing, k is reset to equal K.

8 REBALANCING AND ORDER EXECUTION

The rebalancing and order execution capabilities instilled in Phuture v2 bring about the ability to shift an index fund to a new set of target weights and assets by generating and executing orders across all necessary chains. Below is a diagram outlining the rebalancing process from start to finish, and in the subsequent sections, we will dive into this further.

An important consideration to be aware of during cross-chain order execution is that minting and redemption functionality is blocked. This is to ensure that value cannot be syphoned from the index by actors taking advantage of the stale NAV. A stale NAV occurs primarily due to latency between the remote chain and the homechain. An example will provide further clarification.

An index manager initiates rebalancing at a NAV of 100 USD per token. The rebalancing requires selling all the held AVAX for a new asset ETH. Once this trade is executed, the homechain must be notified of the composition change. However, there is a delay in this message being received. In the meantime, the price of ETH is increasing rapidly. The actual value of the index is now 110 USD, but the homechain is still reporting a value of 100 USD. This allows a trader to mint new units of the index at 100 USD, wait for the homechain to update with the new composition and then redeem their tokens at 110 USD, profiting 10 USD.

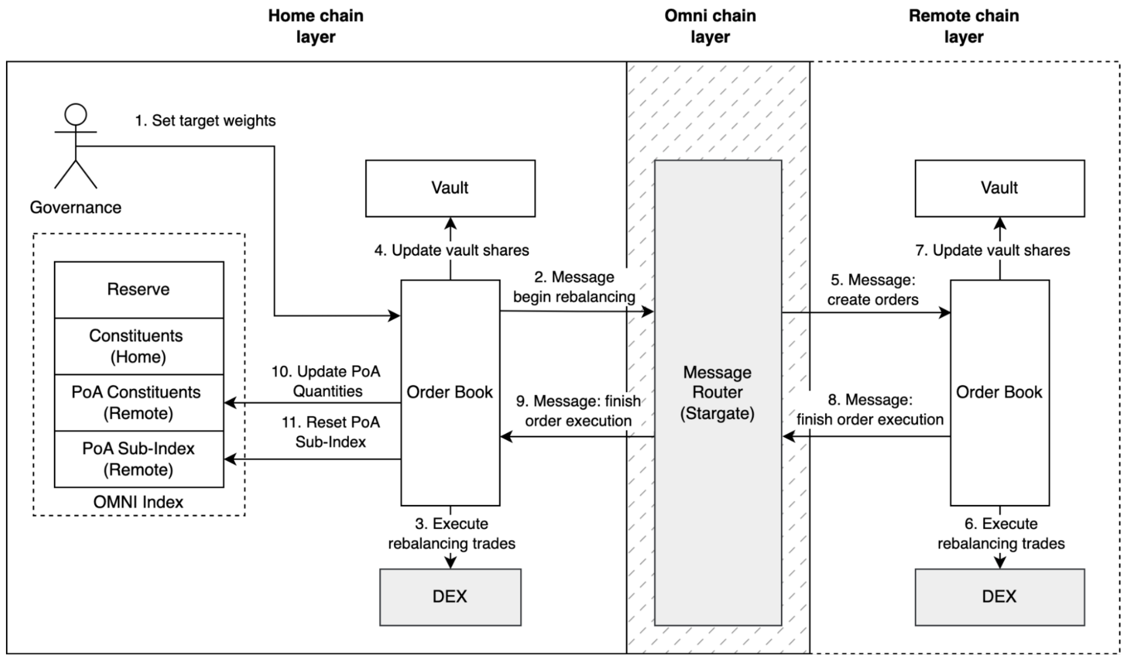

Home chain layer
Omni chain layer
Remote chain layer
1. Set target weights
Governance
Vault
4. Update vault shares
2. Message begin rebalancing
Message Router (Stargate)
5. Message: create orders
Vault
7. Update vault shares
Reserve
Constituents (Home)
PoA Constituents (Remote)
PoA Sub-Index (Remote)
OMNI Index
10. Update PoA Quantities
11. Reset PoA Sub-Index
Order Book
9. Message: finish order execution
8. Message: finish order execution
Order Book
3. Execute rebalancing trades
DEX
6. Execute rebalancing trades
DEX

Figure 8: Rebalancing Flow

The profitability of the above scenario can be reduced by increasing the minting and redemption fees. However, this was considered to be a less robust solution in the event of extreme price volatility.

8.1 Generating Gaps

The current weight w of each constituent i is a ratio of its value divided by the total index valuation. Normalising by V ensures weights are mapped to $[0, 1]$ range.

$$w_i = \frac{C_i p_i}{V} \tag{8.1.1}$$

When rebalancing is initiated, the fund manager supplies a new set of weights w'_i for each constituent i. Sum of all target weights must add up to 1.

$$\sum_{i=1}^{n} w'_i = \sum_{i=1}^{n} w_i = 1 \tag{8.1.2}$$

Gap orders ΔC_i would then be generated for each constituent i by converting the portfolio weight delta into constituent terms. Constituents must be sold if $w'_i < w_i$ and bought if $w'_i > w_i$.

$$\Delta C_i = \frac{V(w'_i - w_i)}{p_i} \tag{8.1.3}$$

Sell/buy gaps ΔC are then generated and orders are sent to the respective remote chains. Rebalancing swaps fill gaps to best move current weights w_i to target weights w'_i.

8.2 Order Types

Order $(C_{k,s}, b_k, \beta_k)$ consists of quantity $C_{k,s}$ for sold constituent s, bought asset b_k and destination chain β_k for order k.

Phuture v2 has 3 order types:

- *Local orders* - Sell $C_{k,s}$ worth of constituent s on the same chain as the current chain $\beta_k = \beta_c$.

- *Outgoing orders* - Sell $C_{k,s}$ worth of constituent s to buy constituent b on the destination chain β_k, where $\beta_k \neq \beta_c$. Outgoing order messages are batched and sent once all local orders are executed.

- *Incoming orders* - Incoming orders are outgoing orders that have arrived at their destination chain β_k. Structurally they are the same as local orders - their destination chain β_k matches current chain β_c.

Each chain will be in one of the following mutually exclusive states:

- *Net outflow* - Funds will be sent to other chains via outgoing orders.

- *Net inflow* - Funds will be received from other chains via incoming orders.

- *Net neutral* - Funds will not be sent or received and only local orders will be executed.

8.3 Outgoing Orders

Given that b_k is the asset to be purchased on the destination chain, the protocol still needs to execute an intermediary buy to the asset that will be bridged to that chain. The bridged asset depends on the destination chain β_k and can be recovered from a mapping held on-chain. This allows the protocol to select the best bridging asset for each destination chain. Bridging assets are chosen based on their liquidity to maximise the output value on the destination chain.

8.4 Incoming Orders

The number of incoming orders I' is passed to each chain when rebalancing begins. The expected count of incoming orders is necessary to not end rebalancing prematurely.

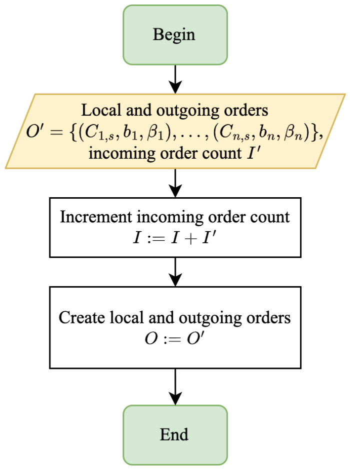

Begin
Local and outgoing orders
$O' = \{(C_{1,s}, b_1, \beta_1), \ldots, (C_{n,s}, b_n, \beta_n)\}$,
incoming order count I'
Increment incoming order count
$I := I + I'$
Create local and outgoing orders
$O := O'$
End

Figure 9: Update Orderbook Flow

Once incoming orders arrive at their destination, they are added to the order set O. The expected incoming order count I is decremented upon message arrival.

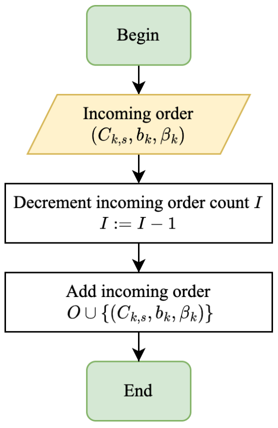

Begin
Incoming order
$(C_{k,s}, b_k, \beta_k)$
Decrement incoming order count I
$I := I - 1$
Add incoming order
$O \cup \{(C_{k,s}, b_k, \beta_k)\}$
End

Figure 10: Process Incoming Orders Flow

8.5 Trading

Phuture v2 allows trustless trading through exchange aggregators by prescribing several checks and balances to ensure the correct trades are being executed, at the real market price, within acceptable slippage tolerances. Firstly, the protocol sanitises the order by ensuring it exists within the set of orders on that chain, preventing incorrect orders from being executed.

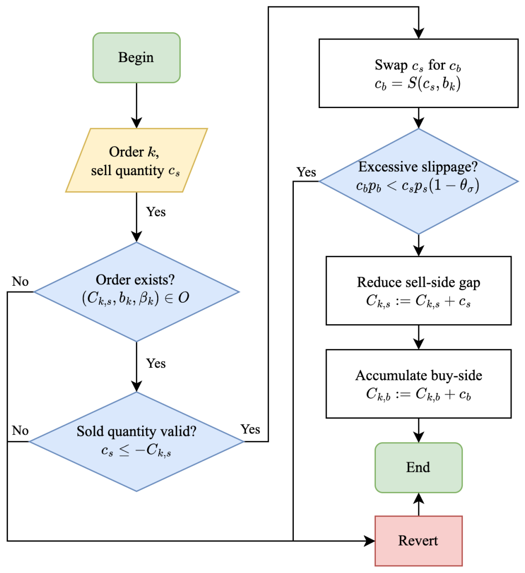


Figure 11: Execute Trade Flow

Sold order amount is verified - caller cannot sell more c_s than available order quantity $-C_{k,s}$ (sell-side orders are negative).

$$c_s \le -C_{k,s} \tag{8.5.1}$$

Swapping constituent quantity c_s with swap function S is only allowed if the value of the bought asset quantity c_b is within the allowed tolerance margin θ_σ. This prevents trades with excessive slippage from being executed and eroding the value of the index.

$$c_b p_b < c_s p_s(1 - \theta_\sigma) \tag{8.5.2}$$

Order sizes are then updated to reflect the amounts traded. Orders move towards zero when trades are executed, reflecting a tightening of gaps. Negative sell side quantity $C_{k,s}$ moves towards zero by getting incremented by c_s. Buy-side quantity is accumulated and $C_{k,b}$ is incremented by c_b.

$$C_{k,s} := C_{k,s} + c_s \tag{8.5.3}$$
$$C_{k,b} := C_{k,b} + c_b \tag{8.5.4}$$

8.6 Completing Rebalancing

Rebalancing is complete on each chain when all $-C_{k,s}$ are below the minimum value threshold θ_m for all orders and the incoming orders count I is equal to zero.

$$\sum_{k=1}^{|O|}[-C_{k,s} \cdot p_s < \theta_m] = 0 \tag{8.6.1}$$

Outgoing order messages M_o with bought quantity $C_{k,b}$ are sent to each destination chain β_k before rebalancing completion.

When all orders are executed, message M_h is sent to the homechain β_h to signify rebalancing completion on the remote chain. Each M_h will refresh the PoA balances held by the sub-index for that particular chain.

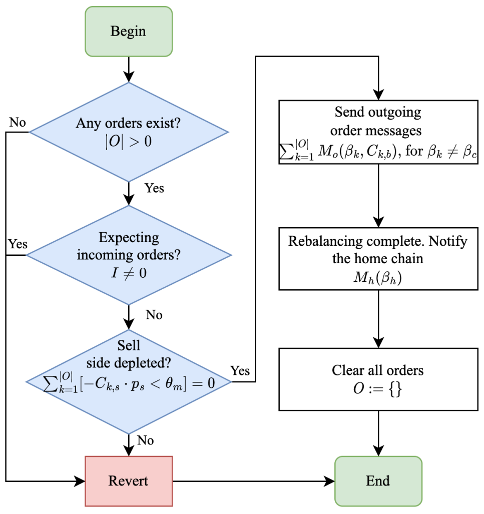

Begin
Any orders exist? $|O| > 0$
No
Yes
Expecting incoming orders? $I \neq 0$
Yes
No
Sell side depleted? $\sum_{k=1}^{|O|}[-C_{k,s} \cdot p_s < \theta_m] = 0$
Yes
No
Revert
Send outgoing order messages $\sum_{k=1}^{|O|} M_o(\beta_k, C_{k,b})$, for $\beta_k \neq \beta_c$
Rebalancing complete. Notify the home chain $M_h(\beta_h)$
Clear all orders $O := \{\}$
End

Figure 12: Finalise Rebalancing Flow

Once the homechain receives rebalancing completion messages from all chains, the sub-index scalar is reset and minting and redemption functionality is reinstated.

8.7 Reserve Rebalancing

Reserve rebalancing is an operation initiated on the homechain, which converts the funds held in the reserve, C_r, into the constituent assets of the index according to the current set of target weights, w_i.

$$C_{k,r} = -C_r w_i \tag{8.7.1}$$

The sold quantity $C_{k,r}$ computed for reserve asset r, order k is injected into local or outgoing orders, depending on which chain the constituent asset resides. Orders $(C_{k,r}, b_k, \beta_k)$ are local when $\beta_k = \beta_h$ and outgoing when $\beta_k \neq \beta_h$. In all cases, b_k is the constituent asset to be bought by the order.

The process for trading and reinstating minting/redeeming is identical to that described in section 8.5 and 8.6 respectively.

9 OPTIMISATIONS

The optimisations in Phuture v2 centre around compression in order to reduce the operational costs of the index and allow the protocol to support index products with potentially 100s of assets.

Optimisations fall into two key areas:

- *Storage costs* - Storing redundant data or making unnecessary transfers for multiple assets between smart contracts.

- *Bandwidth costs* - Passing data for multiple assets in between smart contracts (making external calls) and sending cross-chain messages. Bandwidth-related gas overheads can be, in some cases, comparable to making multiple SSTORE operations.

Throughout this paper we have discussed several optimisations including the singleton index contract architecture, reserve accounting and sub-index accounting. Next we will take a look at other optimisations included in Phuture v2.

9.1 Index Config Hashing

The index config contains the data that is static in between rebalancing periods such as assets, weights, index balances, fee parameters, etc. Only the 32-byte hash of the index config is stored on-chain. During minting or redemption a hashed version of the config is generated off-chain and passed in as a parameter where it is compared to the stored config hash. This removes the need to SLOAD all the required variables during minting and redemption.

9.2 Bit Packing

Phuture v2 tightly packs multiple variables into a single EVM word to reduce bandwidth when making external calls and sending cross-chain messages. Libraries for multiple packed variables were created:

- *A160U96* - Packs an address and uint96 balance variable into a single word.

- *U16X15* - Packs 15 uint16 variables into a single word with one variable allocated for length encoding.

- *U96X2* - Packs 2 uint96 balance variables into a single word.

9.3 Bit Sets

Bit sets can store up to 256 unique, contiguous integers in a single EVM word. Phuture v2 internally assigns a unique integer ID for each asset's address, which reduces bandwidth from N address elements to $\lceil N/256 \rceil$ address ID elements.

10 CONCLUSION

Phuture v2, as described in this paper, will allow for the creation and management of cross-chain, non-custodial index funds with permissionless unit issuance and redemption. The innovations that Phuture v2 delivers will significantly increase the asset universe that crypto index funds have access to, allowing for the formation of comprehensive market and sector-based indices.

11 FUTURE WORK

Building on the foundations set out in this paper, the protocol can be extended to include:

- *Support for non-EVM chains* - Providing access to assets on non-evm chains to further broaden the available asset universe.
- *Improved pricing* - Provide pricing support for longer tail assets and improve price feed aggregation on the homechain.
- *Improved bridge pathing* - Expand the number of bridges used and optimise pathing between them.
- *ZKP-driven scaling use-cases* - Perform complex calculations for 100s of assets off-chain (e.g. NAV), dynamic fee calculation based on market risk modelling and advanced rebalancing/trading strategies.

12 DISCLAIMER

This paper is for general information purposes only. It does not constitute investment advice or a recommendation or solicitation to buy or sell any investment and should not be used in the evaluation of the merits of making any investment decision. It should not be relied upon for accounting, legal or tax advice or investment recommendations. This paper reflects current opinions of the authors which are subject to change without being updated.

REFERENCES

[1] Bloomberg.com, *Passive likely overtakes active by 2026, earlier if bear market*, 2021, `https://www.bloomberg.com/professional/blog/passive-likely-overtakes-active-by-2026-earlier-if-bear-market/`.

[2] CNBC, *ETF EDGE New report finds almost 80% of active fund managers are falling behind the major indexes*, 2022, `https://www.cnbc.com/2022/03/27/new-report-finds-almost-80percent-of-active-fund-managers-are-falling-behind.html`.