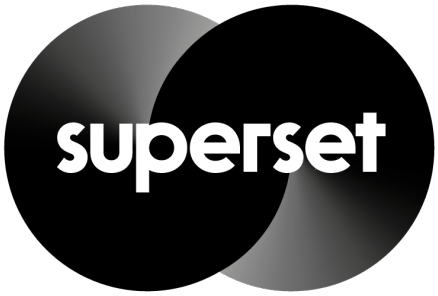
Superset Protocol V1
Whitepaper

Superset Technologies Inc

August 2025



Neil Staunton | CEO | neil@superset.finance
Ben Haslam | CTO | ben@superset.finance

Contents

1 Introduction

1.1 Motivation

The global foreign exchange market processes $9.6 trillion in daily volume, and this liquidity is now moving on-chain. Stablecoins represent the bridge between traditional finance and decentralized finance, with market projections reaching $2 trillion by 2028 according to Standard Chartered, and $3.4 trillion by 2030 according to Citi. As this transformation accelerates, the infrastructure supporting stablecoin trading must evolve to meet institutional standards of capital efficiency and execution quality.

The blockchain ecosystem is expanding rapidly, with over 300 new chains expected to launch in the next 24 months. While this growth creates opportunity, it also introduces a critical challenge: liquidity fragmentation. When a stablecoin is deployed across multiple chains, its liquidity becomes divided across separate pools on each chain, leading to:

- Higher slippage for large trades as liquidity depth decreases
- Increased costs for liquidity providers managing multiple positions
- Reduced capital efficiency as the same capital cannot serve multiple chains simultaneously
- Poor execution quality compared to centralized alternatives

Traditional DEX aggregators attempt to solve this by routing trades across multiple pools, but they remain constrained by the liquidity available on a single chain. Cross-chain aggregators offer another solution, but they introduce additional friction, cost, and security risk. Finally, Solver models can optimize trade execution, but they cannot create liquidity where it does not exist, as they rely of existing DEXs on a given chain.

1.2 SuperPools

Rather than accepting fragmented liquidity as an inevitable consequence of a multichain world, Superset virtualizes liquidity across chains through SuperPools — the industry's first unified omnichain stablecoin reserve protocol. SuperPools aggregates liquidity from all supported chains into a single global pool, allowing traders to access maximum depth from any chain, while liquidity providers deploy capital once and earn from activity across all chains where that pool is traded.

Thanks to cross-chain interoperability protocols like LayerZero, it is now easier than ever for token projects to deploy their tokens on multiple chains. Using token standards like LayerZero's OFT, tokens can be moved between chains by burning them on one chain and minting them on another. Although this allows token projects to reach a wider audience, the liquidity for that token is fragmented across all those chains, which leads to the problems described above.

A virtual pool is a liquidity pool spread over multiple blockchains, where they all share the same liquidity, and therefore the largest trade sizes can be executed with minimal slippage.

SuperPools used a hub and spoke mechanism, where the liquidity for trades is held on the spoke chains, whereas the liquidity aggregation and calculations of swaps happens on the hub chain. The hub chain virtualizes all existing liquidity, and builds on top of the battle tested Uniswap architecture, for efficient trading using concentrated pools of global liquidity. Each liquidity pool is a token pair, most often a stable coin or RWA (StableGBP, StableEURO, RWA...) paired with a liquidity token, a stablecoin with deep liquidity (USDT). For clarity, this token will only be denominated in USD.

When a user makes a swap between two assets using SuperPools, say StableGBP and StableEURO, they will in fact be doing two swaps: StableGBP $\rightarrow$ USD, USD $\rightarrow$ StableEURO. They will make swaps through a router contract which will then create the swap requests to be sent to SuperPools. Most of the tokens being traded are OFTs, including USDT via USDT0, so they can effectively be moved cross-chain by burning and minting.

When a user makes a swap request to be paid out to them on the same chain (not a cross chain swap), there is no need to actually transfer the tokens to the hub chain, rather the intent of what they want to do is sent to the hub chain, which will calculate the output of the swap request, and then send the output back to the user, who is then paid back using funds stored in the pool on the local chain.

This means that cross chain transfers of tokens are only needed when the user wants to be paid out on a different chain than the one they made the swap request on, or for when the hub chain needs to rebalance the liquidity in the pools. For this, OFT transfers will be used by the protocol.

This protocol is initially built for OFTs (the LayerZero standard for cross chain tokens), but will be extended to other cross chain token standards such as CCIP and NTT in the future.

1.3 SuperFactory

Superset also includes the SuperFactory, a tool that enables token issuers to deploy then manage multichain tokens with a single transaction, automatically configuring cross-chain connectivity through LayerZero's OFT standard. The SuperFactory abstracts away the complexity of deploying and managing tokens across multiple chains, allowing issuers to focus on building their projects. By using the SuperFactory, issuers can ensure that their tokens are compatible with SuperPools from day one, enabling them to access deep liquidity and capital efficiency across all supported chains.

2 SuperPools

SuperPools is a set of virtual liquidity pools spread over multiple blockchains, that will allow each chain to share the same global liquidity. This means that traders on any chain will be able to access all the token's liquidity when they make a trade, so they will be able to make large trades with minimal slippage.

The infrastructure of SuperPools consists of a hub chain and multiple spoke chains. The hub chain is the chain that will be used to aggregate liquidity from all the spoke chains, and will be used to process trades between the spoke chains. The hub chain has an internal AMM used to simulate trades using the virtual liquidity, which is how it processes user trades. The spoke chains are all the chains where the token is supported, each will have a local liquidity vault of supported tokens, which are used for liquidity onboarding, migration and rebalancing, as well as processing the swap result; paying a user out with the result of their swap.

A user can make a swap request on any supported chain, and the hub chain will process the swap request and send the swap response back to the spoke chain. The spoke chain will then pay the user out with the result of the swap. If the user requests to be paid out on a different chain than the one they made the swap request on, the hub chain will send the result of the swap to the spoke chain, where it will be paid out to the user, and send a message to the initial spoke chain to close the swap request. This means that cross chain swaps are possible with SuperPools.

2.1 Hub Chain virtual AMM

The hub chain has 'mirror tokens'', which mirror all the tokens contained in the spoke chains within the protocol. Whenever more tokens enter the protocol, via liquidity onboarding or a user making a swap, mirror tokens are minted to represent the new tokens on the hub chain. Mirror tokens are burned whenever tokens leave the protocol in a similar fashion. This means that the internal mirror tokens on the hub chain will perfectly match the tokens out the spoke chains, so any AMM operation applied to the mirror tokens can be used to get exchange tokens on the spoke chains.

The AMM in question is Uniswap v3, which is on the hub chain and is used to create pools of mirror tokens. When a swap request comes in from a spoke chain, the input tokens are mirrored (mirror tokens minted), then the output of the swap is calculated using Uniswap on the pools of mirror tokens (by just calling swap on the hub chain). The output mirror tokens are then burned, and the result (including the amount out) is sent back to the user on the spoke chain.

USDT0 will be used as the base trading asset for the majority of tokens, i.e. pools will be setup for a given token against USDT0, but it will be possible to create different trading pairs for specific use cases, i.e. GBP - EURO.

2.2 Spoke chain contracts

As mentioned, the spoke chains hold the active liquidity in the protocol. For this we are using a singleton model, meaning there will only be one contract per chain, which will hold all the protocol assets for that chain. This means that for example if there are four different virtual pools that have USDT0 as a token that include a given chain, the spoke chain contract will hold all the USDT0 liquidity for that chain, rather than having four different contracts, one for each virtual pool.

Of course, in the hub chain these different pools are accounted for, but by sharing one pot of liquidity on the spoke chain, the effective liquidity available for trades is much higher. When there is un-even rates trading on the same pool across different chains, the protocol needs to rebalance the liquidity tokens to fix the ratio on each chain to what it should be as per the hub chain (this mechanism is described in more detail in section 3.3). With this singleton design, the protocol can effectively rebalance several pools on the spoke chain at once, rather than having to rebalance each pool individually.

Finally, rebalancing will be required less often, as the liquidity drain of a token from one pool on the spoke chain could be cancelled out by the liquidity added of that token to another pool on the same chain, so rebalancing that token is not necessary. These liquidity abstractions are needed to make the protocol as efficient for end users as possible.

2.3 SuperPools Onboarding

After a new or existing token has been deployed across several chains using the SuperFactory as shown above, it can be integrated into SuperPools. This involves deploying liquidity pools on each chain for that token, which then connect to the hub chain to aggregate liquidity from all the spoke chains.

When liquidity is added or removed from the pools, through onboarding or due to a swap, the new balance of the pool is sent to the hub chain, where it is aggregated with the other pools. This means that the hub chain will always have the most up to date reserves of the token, and will be able to process trades between the spoke chains.

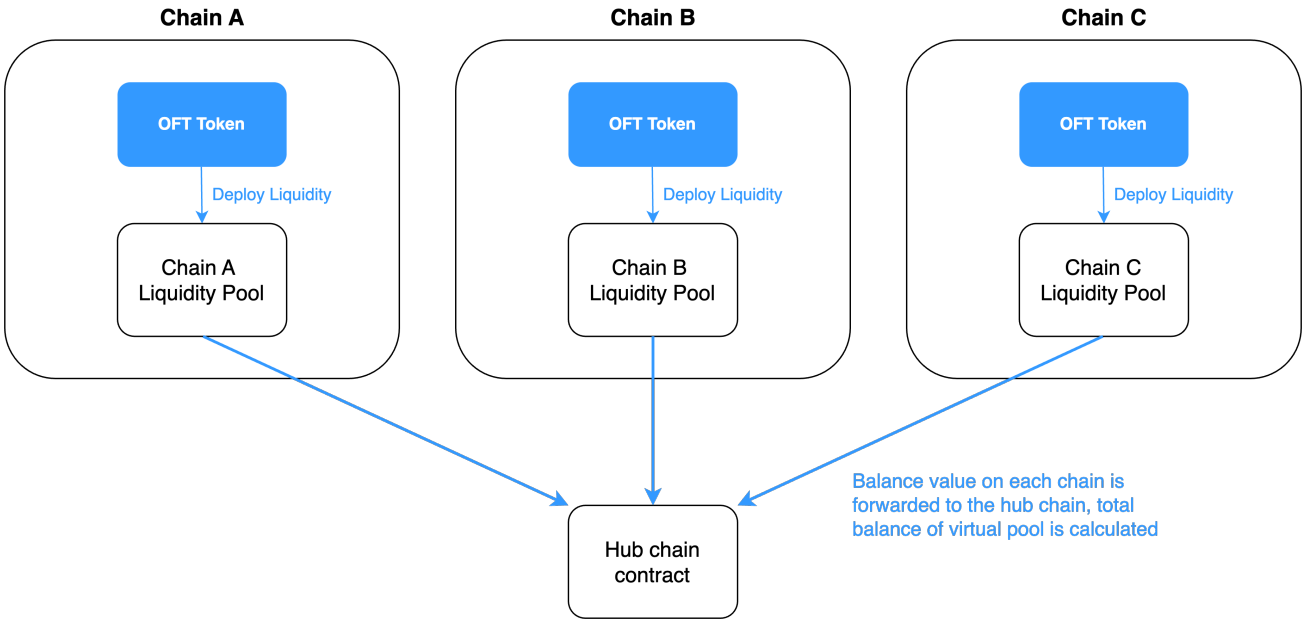


Figure 1: Token integrates with SuperPools: Liquidity is deployed locally on each chain, total reserves are calculated by the hub chain contract

2.4 SuperPools User Flow

2.4.1 Step 1 - User makes a swap request

The user makes a swap request between two tokens, which is passed from the router contract to SuperPools. The swap requests can be thought of as intents, to swap one token for a given range of another, which will then be received on a given chain. This swap request is sent to the hub chain, which will then be used to process the result of the swap request based on the total virtual liquidity pool available across all chains.

2.4.2 Step 2 - Swap is executed

The hub chain contract receives the swap request, and will then calculate the amount out using the total liquidity of the virtual pools of the two tokens being swapped. Generally there is one pool per token, which is a pair of the token and USD. So if a user wants to swap between two tokens, say TA and TB, the hub chain contract will first swap TA for USD, then swap USD for TB. As the USD being moved is the same for both of these, none actually needs to be moved, so the net result is that the balances of the pools of TA and TB are updated, and the user is paid out with the result of the swap.

The hub chain will then send the result of the swap request back to the spoke chain, where the user will be paid out with the result of the swap (TB). As the ratios of the tokens in the pools on each chain have now changed (and will be continuously changing with every swap), the protocol can rebalance the pools on individual chains, to ensure that there is enough liquidity in each pool for future swaps. We have an incentive mechanism in place to ensure that the pools are balanced and have sufficient liquidity for the trade volume, which will be described in more detail later.

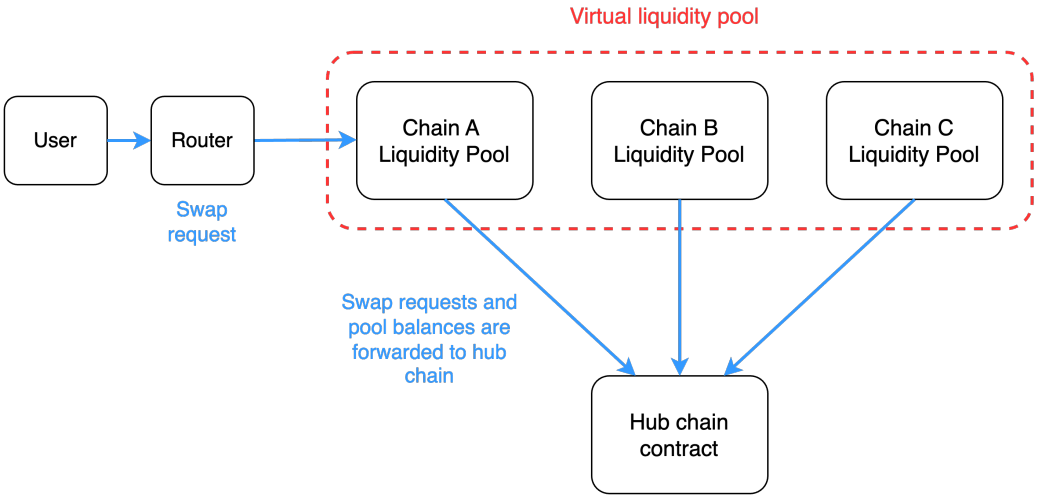


Figure 2: User makes a swap request to SuperPools. Request is forwarded to the hub chain where amount out can be calculated using global reserves

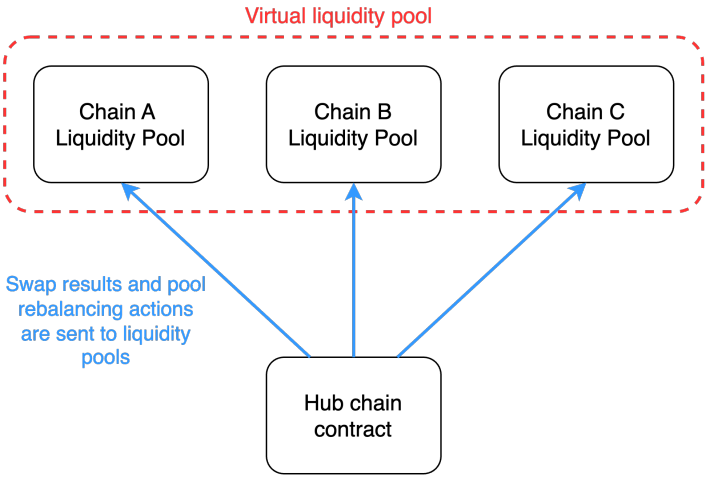


Figure 3: The hub chain contract executes the swap request, and sends the result back to the spoke chain

2.4.3 Step 3 - User payment and pool rebalancing

The output of the swap request is now sent to the user, and pools exchange liquidity to ensure that each one has enough for future swaps. There is no reason why a user would have to be paid out from the same chain they paid in. It would be simple to provide a destination chain and address, making cross chain swaps possible with SuperPools.

2.5 Additional Liquidity Onboarding

In order to add more liquidity to a given virtual pool, a user has a couple of options. The most simple option that does not shift the price is to provide an equal value of both tokens supported by the pool (TokenA / USD). This will increase the volume of both sides of the pool, but will not change the price/ratio. This is the standard for constant product AMMs.

Another option is to add an amount of the given token (TokenA) as well as the equal value of another token (TokenB) that is supported by SuperPools. This will internally swap TokenB for USD, then use that to add liquidity to the TokenA / USD pool. This will have the potentially unintended effect of increasing the TokenB / USD ratio, therefore decreasing the price of TokenB.

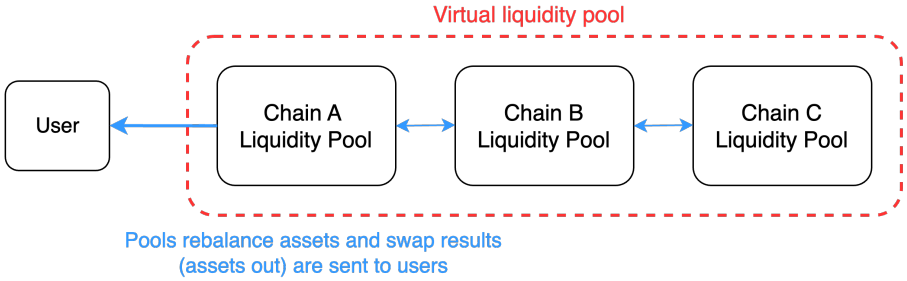


Figure 4: Pools exchange liquidity and the user receives their funds

The final option would be to just add TokenA with no other tokens for the other side of the pool. This approach would increase the ratio of TokenA / USD, therefore would decrease the price of TokenB.

For a new token that is not currently supported by SuperPools, a vote would be done by Superset token holders to whitelist the new token on all supported chains and create the new pools and virtual pool. From there, the liquidity must first be added using one of the first two methods described above to set the initial price for the new token.

2.6 Security Measures

SuperPools is built on top of LayerZero, as it was chosen as the most secure and efficient cross chain messaging protocol, with zero hacks to date. Despite this, we have implemented additional security measures to ensure that the system is secure and robust.

A concern a user could have is that their swap request is sent on the spoke chain, but something goes wrong so it does not reach the hub chain, or the response is not sent back to the spoke chain. In this case, the user would be left with no funds and no way to get them back. To prevent this, we have implemented a timeout mechanism, where if the swap response is not received after a certain block limit, the user can call a function on the spoke chain to get their funds back. This function will check the status of the swap request on the spoke chain, and if it was not executed, the user will be able to get their funds back.

2.7 Just In Time Liquidity

We also introduce the concept of Just In Time (JIT) liquidity, which allows the protocol to automatically move liquidity from the hub chain to one of the spoke chains if the required liquidity is not available on that chain. As this requires an OFT transfer to move the assets, it will cost more gas for the user, but it allows a user to make a trade on a chain that does not have enough liquidity. Ideally, through the migration and rebalancing mechanism described below, the active chains will always have enough liquidity to settle trades, but in the case that they do not, JIT liquidity will be used to move the liquidity from the hub chain to the spoke chain.

2.8 What About Atomic Swaps?

Due to the cross chain nature of SuperPools, there is a delay between the time a user makes a swap request and the time they receive their funds. This means that atomic swaps are not possible on the spoke chains, which could limit their use cases is places like DEX integrations with other protocols, and in high frequency trading strategies. We are implementing a solution to this however, with the following mechanisms:

1. **Hub Chain Atomic Swaps**: For users trading on the hub chain itself, the process for this will be the same as for spoke chains described above, except instead of making cross chain messages to the hub chain contract, it will simply be local calls to the hub chain contract. This means that the user will be able to make atomic swaps on the hub chain, and will be able to use the liquidity on the hub chain for their trades. This will enable high frequency trading strategies on the hub chain, in situations where any latency is unacceptable.
2. **Post-Swap Execution Logic**: We will allow users to pass in post swap execution logic with their swap request, which is additional logic that will be executed after the swap is executed, so when the user has

the liquidity in their wallet. This means that SuperPools can be used with other protocols, such as lending protocols, where the user can pass in the logic to lend out their funds after they have been paid out. This will require the user to add extra gas to pay for the execution of the post swap logic, which can be specified by developers using the ‘extraOptions‘ field for making a cross chain message. This will be a powerful tool for developers to use, as they can create their own logic to be executed after the swap is executed, and can be used to integrate with other protocols.

3. **Hooks**: Similar to Uniswap V4, we will allow issuers creating a new pool to add hooks to the pool, which are additional logic that will be executed before or after a swap is executed, or before or after liquidity is added or removed. This will allow issuers to create their own logic to be executed when users interact with their pools, which can be used to integrate with other protocols, or to add additional functionality to their pools. This is completely optional for issuers, but allows them to build on top of SuperPools in any way they choose.

3 Liquidity Optimization and Rebalancing

3.1 Introduction

Superpools aggregate liquidity across multiple chains, allowing for maximum capital efficiency for trading pairs. However, as trading activity is unlikely to be evenly distributed across all chains, we need an incentive mechanism that rewards real trading activity on each chain, while also ensuring that liquidity is situated where it is needed most.

For this reason, it is important to rebalance the liquidity and migrate positions between individual chains to ensure there is always enough liquidity to sufficiently settle trades on each chain. Therefore a fee mechanism must be used to incentivize LPs to reallocate liquidity between chains, to ensure that the pools are always well balanced.

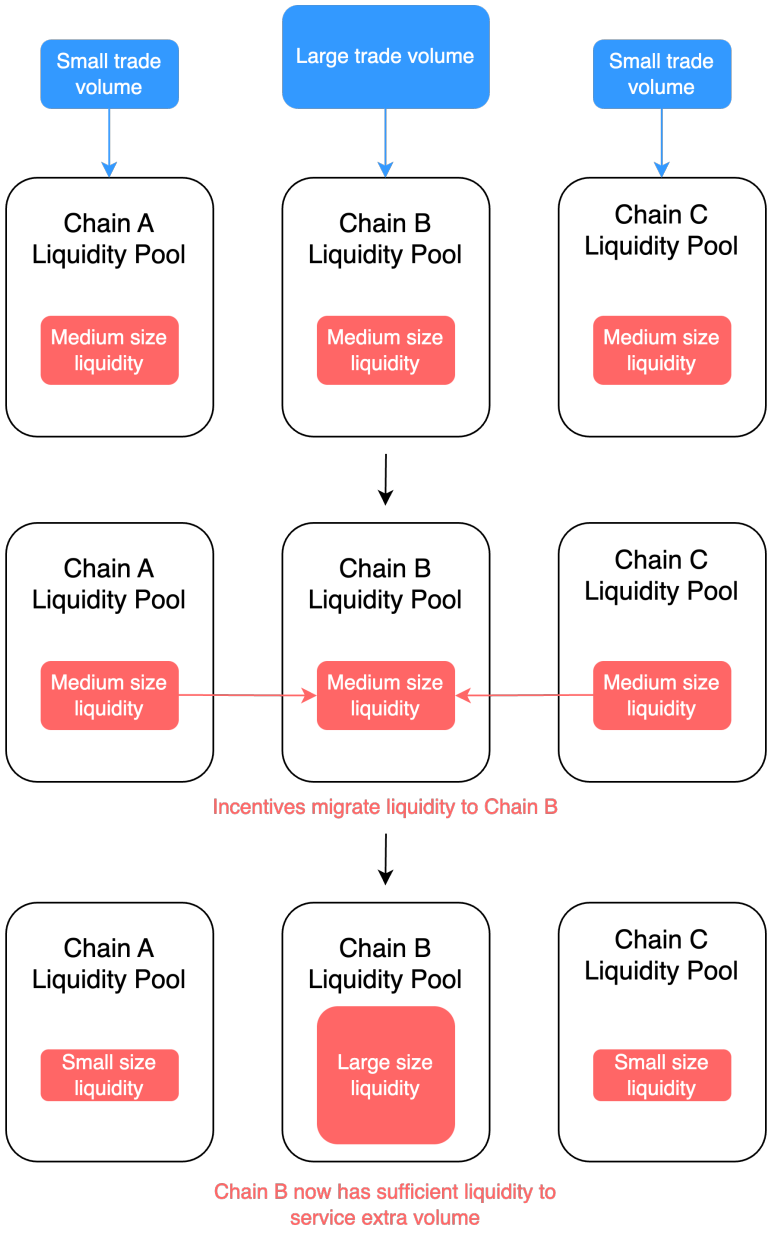


Figure 5: The protocol uses incentives to migrate the liquidity on each chain, to ensure that there is always enough liquidity to settle trades

3.2 Two Key Mechanisms for Liquidity Optimization

Superset's economic architecture is designed to optimize cross-chain liquidity through two key mechanisms:

- Rebalancing token ratios across spoke chains, described in section 3.3.
- LPs commit their LP position to a specific chain, which migrates their liquidity to that chain and allows them to earn rewards based on the usage of the liquidity on that chain. This process is described in more detail in section 3.4.

When tokens are being moved using either of the mechanisms described above, during the cross chain transfer the representation of the tokens to be moved on the hub chain go from 'src chain' to 'pending'. These tokens can no longer be used to pay out users from any chain (although they do still go into the total liquidity of the virtual pool). Once they are confirmed to have landed on the destination chain, the hub chain will update its internal representation and these tokens can then be used to pay out users on the destination chain.

3.3 Rebalancing Token Ratios

In addition to the LP staking mechanism, Superset implements a rebalancing mechanism to ensure that the token ratios across spoke chains are always in line with the hub chain. This is done by monitoring the token ratios on each chain, and if they deviate from the ideal ratios as described on the hub chain (the sum of LP positions committed to a given chain across pools operating on that chain), the protocol will rebalance the pools on each chain to bring them back in line.

As alluded to in section 2.2, when there is un-even rates trading on the same virtual pool across different chains, this will cause the token ratios to deviate from the ideal ratios as set by the hub chain. For example, say you have the same virtual pool of Token 1 and Token 2 on two chains, Chain A and Chain B, with the same token ratios. If a users on Chain A are on average buying more of Token 1, and users on Chain B are on average buying more of Token 2, this will cause a net flow of Token 2 to Chain A and Token 1 to Chain B.

This means that the token ratios on Chain A will be different from those on Chain B, so the protocol should rebalance the pools on each chain to bring them back in line, effectively cancelling out the net flow of tokens between the two chains. This action can be done by anyone for an incentive reward. The user will specify the chains and tokens they wish to rebalance, and the protocol will check that this action is needed, and if so, will execute the rebalancing action. The user will then be rewarded with Superset tokens, in proportion to the amount of liquidity rebalanced.

In practice, with the singleton model described in section 2.2, there will be multiple pools using the same tokens on each chain, so the protocol will effectively rebalance all these pools at once. It is also likely that there will be several different chains using a given token rather than just two as in the given example. A rebalancing action will define a single token moving between two chains, but multiple rebalancing actions can be bulk executed together, in the case the token ratios need to be fixed on multiple chains at once.

The incentive reward given to a user who is performing a rebalancing action will be proportional to the amount of liquidity being moved in relation to the total amount of liquidity for that token, with a check that the liquidity is being moved in the correct direction to bring the token ratios back in line.

3.3.1 Rebalancing Eligibility and Rewards

The following specification defines how an address performing a rebalancing action is eligible to receive rewards:

All rebalancing is evaluated at a snapshot (point-in-time state) τ at the hub. We compute $M(\tau)$, $M_{\text{eff}}(\tau)$, $B_{\text{avail}}(\tau)$, and $\text{RPU}(\tau)$ only when a commit is proposed (not every block). The hub already mirrors/aggregates spoke balances and targets. At snapshot τ:

- We define $R_i(\tau)$ to be: the raw actual balance (pre-pending) per spoke i.
- We define $P^{\text{in}}_{i,T}(\tau)$ to be amount pending in from accepted, but unsettled commits.
- We define $P^{\text{out}}_{i,T}(\tau)$ to be amount pending out from accepted, but unsettled commits.
- The usable balance used for pricing (taking into account amount pendings from commits) is:

$$R_i^{\text{usable}}(\tau) = R_i(\tau) - P_{i,T}^{\text{out}}(\tau) + P_{i,T}^{\text{in}}(\tau)$$

- We define $R_i^*(\tau)$ as the hub chain's snapshot of what each token R's target inventory should be on spoke i. This is the sum of LPs and single-sided stakers for token R.

$$R_i^*(\tau) = \text{LPStakedToSpoke}_i(T) + \text{SingleSidedInventory}_i(T)$$

  This requires looping over LP positions of a given token, and summing the amounts committed to each spoke chain, so should only be done when the available rewards are more than the gas costs. As the rewards implementation is upgradable, this can be improved over time.

- The surplus at τ on spoke i, i.e. how far a spoke is over its target:

$$S_i(\tau) = \max\left(R_i^{\text{usable}}(\tau) - R_i^*(\tau),\, 0\right)$$

- The deficit at τ on spoke i, i.e. how far below a spoke is under its target:

$$D_i(\tau) = \max\left(R_i^*(\tau) - R_i^{\text{usable}},\, 0\right)$$

- We can then define M as the Global Misallocation Mass for token R across all spokes i:

$$M(\tau) = \frac{1}{2}\sum_i \left|R_i^{\text{usable}}(\tau) - R_i^*(\tau)\right|$$

  This implementation is inspired by Earth Mover's Distance (EMD), and measures the minimal tokens moved ('transport cost' in EMD) to align distributions. This gives us a global view for proportional rewards tied to system-wide importance (larger M lowers Rewards Per Unit (RPU) to spread budget).

  $M(\tau)$ here is the minimum token mass that must move at τ to align all spokes with their targets. We use snapshots as we want to pay for misallocation removed now, not for an entire epoch. This allows us to avoid continuous recomputation, prevents double-counting, and keeps gas predictable. New trading later creates a new snapshot τ' with a new $M(\tau')$.

- $M(\tau)$ **Calculation example:**

  Targets $R_i^*(\tau)$: 500/250/100 across 3 chains

  Actuals $R_i(\tau)$: 600/200/50

  Deviations $R_i(\tau) - R_i^*(\tau)$: 100/-50/-50

  Global Misallocation Mass $M(\tau)$: $(100 + 50 + 50)/2 = 100$

- We compute the global reserves, defined as the sum of all raw token balances across spokes for a given token, to set a mass floor:

$$R_{\text{global}}(\tau) = \sum_i R_i(\tau)$$

- And we then define a mass floor to protect us when we calculate Rewards per Unit when $M(\tau)$ is tiny:

$$M_{\text{floor},T} = \eta_T \cdot R_{\text{global}}(\tau)$$

  Where η_T is a small, token-specific constant (not snapshot-specific) representing the minimum fraction of global reserves to use as a floor when calculating rewards for token T.

A token T is eligible for Rebalance on snapshot τ if:

- **Absolute trigger (per spoke):**

$$\left|R_i^{\text{usable}} - R_i^*\right| \geq A_{\min,T}$$

where $A_{\min,T}$ is a token-specific constant (not snapshot-specific) representing the minimum absolute deviation required to trigger a rebalance for token T.

- Front end check: Global floor: $M \geq M_{\min,T}$
- Front end check: Relative trigger (per spoke to ignore noise):

$$|e_i| \equiv \left|\ln\left(\frac{R_i}{R_i^*}\right)\right| > \varepsilon_T$$

Funding:

- **Per-token Rebalance Pot:** $Pot[T]$, funded via 10% of fees for token T, and pays rebalancing rewards in T.
- **Reserve:**

  Define ρ to be the fixed fraction of the per-token Rebalance Pot that is never spendable for regular rebalances. This gives us a per-token permanent buffer so a single snapshot or cluster of commits can't drain the live pot. ρ protects solvency and gives us "dry powder" for later snapshots. In the below parameter settings, we choose a relatively small value for ρ. This was because once a rebalance happens, there should not be an imbalance until new trading occurs, which then generates new fees, and increases the Rebalance Pot. We use this smaller amount for edge cases.

$$\text{Res}[\tau] = \rho \cdot \text{Pot}[\tau]$$

  Reserved[T]: the sum of outstanding reservations for accepted commits.

  The purpose of this ρ parameter is because there may be large trades where the fee is only collected on one side, but by nature also causes a surplus/deficit in the paired token which needs rebalancing. This is to harden the protocol itself without having to dip into the Reserve Fund.

- **Unreserved and Available budget (per token T, at snapshot τ)**

  Budget B refers to how much of a token's pot is available for paying out rebalancers.

  Unreserved Budget: the pot after subtracting the permanent reserve and any open, unsettled commits:

$$B_{\text{unreserved}}(\tau) = \text{Pot}[T](\tau) - \text{Res}[T](\tau) - \text{Reserved}[T](\tau)$$

  Available Budget: the portion of the unreserved budget we expose that can actually be used at the current snapshot:

$$B_{\text{avail}}(\tau) = \min\left\{B_{\text{unreserved}}(\tau),\, \theta \cdot \text{Pot}[T](\tau)\right\}$$

  Where θ gives us how much of a token's total pot can be spent during any single snapshot, limiting how much of the budget per snapshot we can release.

  Note, we are using this method because we believe it is the most gas efficient way to do this, but we can get more sophisticated if we move off-chain.

  The goal here with the two levels is to prevent multiple rebalancers from overspending simultaneously, and keep rewards smooth over time.

Δ and Rewards Per Unit (RPU) Calculation:

- **Measured improvement (tokens):**

  We define $\Delta\text{Mass}(\tau)$ to be how many tokens were actually moved by the user who initiated the Rebalance, constraining for each source s and each destination d. For a proposed leg set $s \to d$:

$$\Delta_{\text{mass}}(\tau) = \sum_{(s \to d)} \min\{x_{s \to d}\, S_s(\tau),\, D_d(\tau)\}$$

  using snapshot values adjusted for pending holds.

  Using the min here simply limits each proposed leg to the smallest of the source surplus, the destination deficit, and the proposed amount so we don't exceed the amount that should be moved.

  Why Δ_{mass}? Builds on absolute triggers but pays on token units moved for dimensional consistency and proportionality to 'work done'.

- **Price per unit:**

  We define $M_{\text{eff}}(\tau)$ to be the larger of the actual misallocation and the floor, to prevent the unit price from spiking when an imbalance is small:

$$M_{\text{eff}}(\tau) = \max\{M(\tau),\, M_{\text{floor},T}\}$$

  where $M_{\text{floor},T}$ is a parameterized minimum to protect us when we calculate Rewards per Unit when $M(\tau)$ is tiny.

  $RPU_{\text{max},T}$ is a parameter (cap) per token class.

  We can then calculate our Reward per Unit, the unit price for token T that we pay per token of misallocation removed for that given snapshot:

$$RPU(\tau) = \min\left\{RPU_{\text{max},T},\, \frac{B_{\text{avail}}(\tau)}{M_{\text{eff}}(\tau)}\right\}$$

- **Payout:**

  Payout $= \Delta_{Mass}(\tau) \cdot RPU(\tau_{\text{commit}})$

  Paid in kind in token T at settlement.

3.4 Migrating Positions Between Chains

As mentioned in the introduction, for the most effective use of liquidity, it should be allocated to where it is needed most. This means that a mechanism is needed to reward LPs for providing liquidity to chains with higher trade volume in the tokens of their position. When a LP deploys assets into a given pool, they will receive a liquidity token representing their position in the pool. The LP will now start to accrue trading fees due on the usage of the virtual pool across all chains.

This LP token can then be committed in the hub contract to a specific chain. This will redirect the deployed liquidity to that chain, and will give the LP a proportional cut of the LP rewards for that chain. This mechanism ensures that the liquidity of the protocol is always efficiently managed, as users will be incentivized to commit their liquidity to whichever chains are best performing (highest tx volume) to maximize fees. The result of this is that due to MEV all the chains will have the correct amount of liquidity allocated to them and will give approximately equal incentive rewards, as users will move funds if an opportunity appears.

Post-TGE, when users commit their LP tokens as described below, they will start to earn superset tokens ($\boldsymbol{\lambda}$). The $\boldsymbol{\lambda}$ tokens represent ownership of the protocol, and therefore will be used for governance (adding new token protocols, chains), as well as earning a cut of the protocol revenue, generated through the protocol fee. They can start to earn from the protocol fee as soon as they commit their $\boldsymbol{\lambda}$ tokens, and will be able to vote on the protocol governance proposals. The $\boldsymbol{\lambda}$ token is an ERC20 token, and can be traded on any DEX that supports it.

The protocol fee is an additional fee taken on swaps (~0.1%), that is given to λ token holders. The λ token holders will be the team, investors, the public (through a token sale) and also is given as rewards for efficiently migrating LP tokens. This is how the investors in the protocol (both early VCs and LPs) realize profit from the protocol.

Pre-TGE, when we have not yet launched the λ token, the rewards will be given in the tokens that are being used in the pools. This means that LPs will be rewarded in the tokens they are providing liquidity for, which is a more direct way of rewarding them for their contribution to the protocol.

3.5 Incentive Structure

Due to the singleton model of the spoke chain contracts as mentioned in section 2.2, when a liquidity provider commits their LP tokens to a specific chain, they are effectively committing liquidity to all the pools on that chain that use the same tokens. This means that if there are four different virtual pools that use USDT0 on a given chain, the single vault of USDT0 on that chain will be used for all four virtual pools. This abstraction further de-fragments liquidity (on a single chain), as now it effectively increases the transaction size any user can make on that chain, as they can access the liquidity of all the virtual pools that use that token. This means that on a given spoke chain, the different virtual pools are not competing for liquidity, rather it is like they are all sharing the same pot of liquidity, which is more efficient for users.

However, this means that the incentive structure must be designed to ensure that LPs who stake their LP tokens to a specific chain are rewarded in proportion to the usage of the liquidity on that chain, across all the virtual pools that use that token, rather than just the usage on a single virtual pool. This means an LP who has a position of say USDT0 / EURO must be rewarded based on the usage of USDT0 across all the virtual pools that use USDT0 on that chain, as they are effectively providing liquidity used by all those pools to payout users from a swap on that chain.

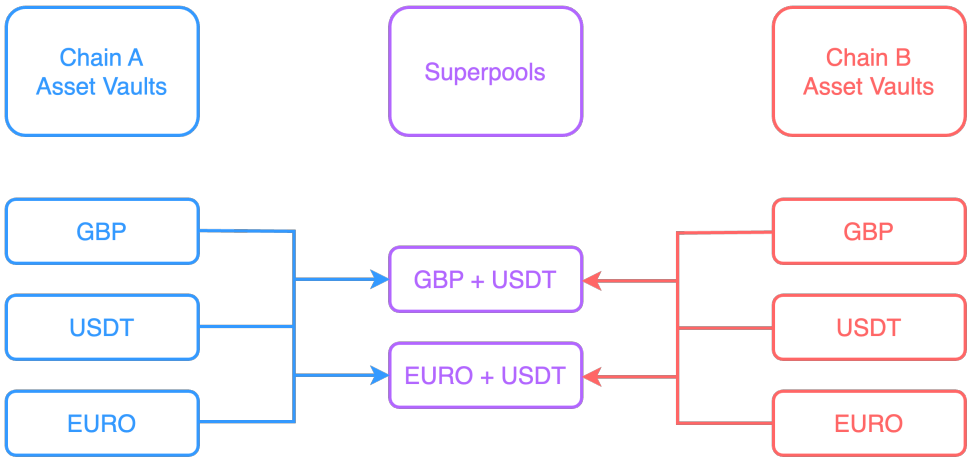


Figure 6: Virtual pools aggregate liquidity on pairs deployed across multiple chains, while spoke chain token vaults allow multiple virtual pools to share the liquidity on a single chain, to increase the maximum transaction size possible on that chain

3.6 Single Sided Liquidity Provision

You might see that this mechanism effectively allows single sided liquidity provision, as an LP can now provide a single token to a given spoke chain, to start earning rewards from the usage of that token on that chain, without actually providing liquidity to a specific virtual pool. This means they have not affected the price or slippage of any virtual pool, but they are still providing value to the protocol, as by providing liquidity to a given chain, they are ensuring that there is enough liquidity to settle trades on that chain, which also means there is less need for the protocol to rebalance tokens between chains, which saves gas and reduces the risk of failed transactions. Therefore, single sided LPs should be rewarded in the same way as LPs who provide liquidity to a specific virtual pool for the usage of the tokens they are providing to a given spoke chain.

This mechanism allows for a less risky form of liquidity provision, as the LP is only exposed to one token, rather than a pair of tokens, so there is no danger of impermanent loss. In this way, single sided liquidity provision is analagous to LPing into a lending protocol, but instead of lending to borrowers, the LP is essentially lending to the protocol so it can use the liquidity to settle trades on that chain.

3.7 Incentive Model

Pre-TGE, the rewards used for this incentive model are taken as a fixed rewards fee, from the *TokenOut* of a swap. This reward fee is then sent to a token pot for the *TokenOut* token, and the spoke chain where that token is being paid out from. This means that each token on each spoke chain will have its own reward pot. The protocol will save up these rewards over a fixed epoch (in the range of 8 hours), and at the end of the epoch, the rewards will be distributed to all the LPs who have committed that token to that spoke chain, in proportion to the amount of liquidity they have committed to that chain.

Post-TGE, the rewards will be given in the Superset Token ($\boldsymbol{\lambda}$), rather than the tokens being used in the pools, which will then be used for governance and earning a cut of the protocol revenue, from the above mentioned protocol fee. The rewards will be distributed in the same way as pre-TGE, but instead of being given in the tokens being used in the pools, they will be given in $\boldsymbol{\lambda}$ tokens.

3.7.1 LP eligibility

The following specification defines how an address providing liquidity through either methods is eligible to receive rewards:

- **Eligibility**: Address must be locked from $t_0 \to t_1$ with either/both:
  1. Single-sided staked token T, and/or
  2. Staked Uniswap v3 LP NFTs whose pools use token T.
- **Weights** (per address, per token T):
  1. **Single-sider**: $\text{weight}_T(\text{addr}) += \min(\text{balance}_{T@t_0}, \text{balance}_{T@t_1})$
  2. **LP NFT contribution**: For each NFT in a pool that uses T, derive the embedded token amounts at t_0 and t_1 from the position's tick range, liquidity, and the pool's price snapshots (sqrtPriceX96@t_0, sqrtPriceX96@t_1); add $\min(\text{amount}_{T@t_0}, \text{amount}_{T@t_1})$ to the address's weight_T.
- **Payout** (per address, per token):
  1. After the finalize walk, compute $\text{rewardPerWeight}[T] = \text{TokenPot}[T]/\text{totalWeight}[T]$.
  2. Address claims $\text{payout}_T(\text{addr}) = \text{weight}_T(\text{addr}) \times \text{rewardPerWeight}[T]$.
- Inventory is rewarded in-kind per token, shared across all pools that consume that token on the spoke.

3.7.2 Anti-Gaming Controls

- **Hard epoch lock**: Stake before t_0; withdraw after t_1. Staked NFTs cannot be edited mid-epoch.
- **Dust floor**: Ignore balances below a small spoke-level threshold to reduce spam.
- **Out-of-range LPs**: Earn nothing from the LP Bonus (no fees), but still earn from the Token Pot only if locked for the epoch.

3.7.3 Epoch Lifecycle

1. **Stake (pre-t_0)**
   - Pools snapshot sqrtPriceX96@t_0.
   - Single-siders snapshot $\text{balance}_{T@t_0}$.
   - LP NFTs snapshot $\text{feeGrowthInside}_{\text{side}}\text{X128}@t_0$, and lock ticks/liquidity.
2. **Accrue ($t_0 \dots t_1$)**

- Pools accrue swap fees and owner protocol fees internally.

3. **Finalize (post-t_1)**
   - Collect owner protocol fees from each pool (per side) into RewardVault.
   - Split: 50% → LP Bonus (that side), 50% → Token Pot (that token).
   - Aggregate Token Pots per token across all pools using that token on the spoke.
   - Snapshot sqrtPriceX96@t_1 per pool; snapshot single-sider balances @t_1.
   - Chunked finalize walk for Token Pots:
     - For each single-sider: $+\min(\text{balance}_{T@t_0}, \text{balance}_{T@t_1})$.
     - For each staked NFT and each token used by its pool: compute amounts at t_0 and t_1; add $+\min(\text{amount}_{T@t_0}, \text{amount}_{T@t_1})$.
     - Ignore any address with $\text{weight}_T(\text{addr}) < \text{dustFloor}$; such weights do not contribute to totalWeight[T] and receive 0 for that epoch.
     - After all items for token T: if totalWeight[T] > 0, set rewardPerWeight[T] = TokenPot[T]/totalWeight[T] and mark finalized; otherwise, carry TokenPot[T] forward to the next epoch.
4. **Claim (anytime after finalize)**
   - LP Bonus: $\text{payout}_{\text{Side}}(\text{NFT}) = \text{rewardPerFeeSide} \times \text{feesEarned}_{\text{Side}}(\text{NFT})$.
   - Token Pot: $\text{payout}_T(\text{addr}) = \text{weight}_T(\text{addr}) \times \text{rewardPerWeight}[T]$.
   - If rewardPerFeeSide or rewardPerWeight[T] is unset because the corresponding sum was 0 at finalize, the associated pot was carried forward and there is no payout for that epoch.

3.8 Additional Chain Incentives

The protocol will also allow users or chains to add additional incentives for the LPs to move their liquidity to a different chain. This will be done in with a plugin, that will allow users to add additional incentives to the LPs to move their liquidity to a specific chain. The plugin will be added to the hub chain, as that is where the liquidity and volume is aggregated. The nature of the incentive is up to the user, but it could be a token or a stablecoin. This is completely optional for chains who want to incentivize LPs to move their liquidity to their chain. They could achieve a similar effect by just LPing themselves to that chain, or using a market maker to do so, and not migrating the liquidity position from that chain.

4 The SuperFactory

4.1 Introduction to the SuperFactory

In addition to SuperPools, we have created the SuperFactory, which allows any token project or issuer to create their own OFT token, which can be used in the SuperPools system. The SuperFactory is a smart contract and SDK that can be used to automatically deploy and setup multi-chain tokens with a single transaction to one chain, then manage the token across all chains after deployment. The user will then be able to use this OFT token in the SuperPools system, and will be able to transfer it between chains using the LayerZero protocol.

The SuperFactory can deploy and configure OFT tokens and adapters, but is also an OApp itself. This means that a factory on one chain is able to send and receive messages from factories on other chains. The reason for adding this OApp capability is so that a user is able to deploy a new OFT token to multiple chains with just one transaction to one chain. The initial factory will receive the transaction, and split it into sub-transactions for each selected chain, and will then forward those messages to factories on the other chains, where they will then deploy the OFT contracts on each chain. When all is finished, through one transaction a user will have deployed a multichain token across as many chains as they like, with all the setup taken care of.

The SuperFactory therefore is an abstraction layer that allows users to deploy and manage their own OFT tokens, without having to worry about the underlying complexity of the LayerZero protocol. The SuperFactory is a powerful tool that will enable users to easily create and manage their own OFT tokens, and will also onboard users into the SuperPools protocol, by setting them up with the correct standard.

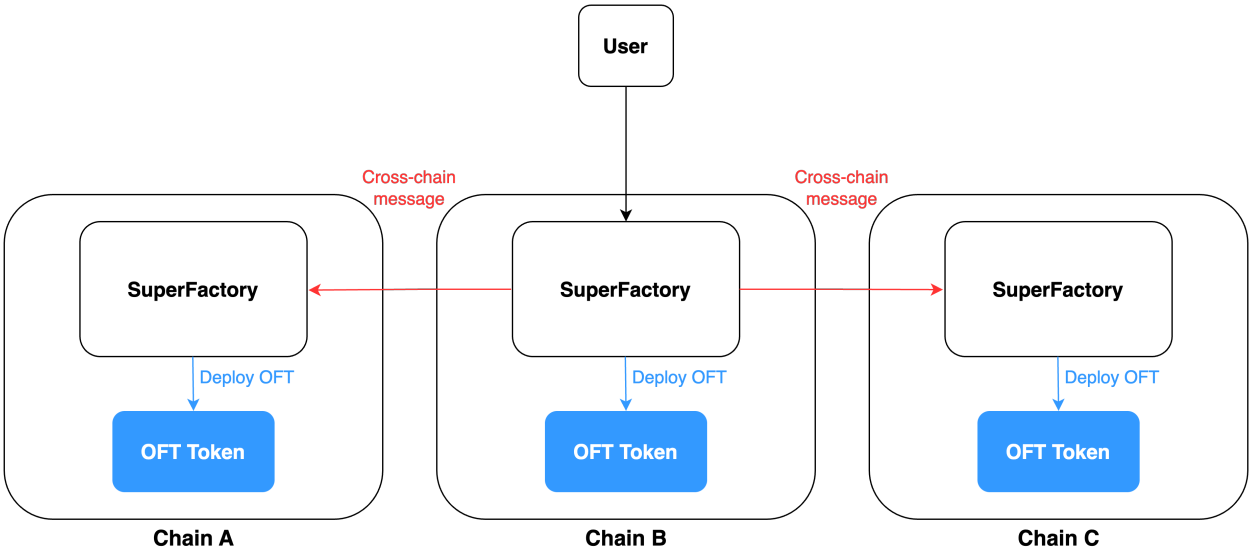


Figure 7: The factory on the initial chain receives the transaction, and splits it into sub-transactions for each selected chain, to deploy and configure the OFT token all at once

4.2 Local OFT Deployments

4.2.1 Create Functions

The SuperFactory has functions `_CreateOFT` and `_CreateOFTAdapter`, which both use the `create2` opcode to deploy an OFT contract to a deterministic address. To load the bytecode, external libraries are used, which are set in the constructor along with other default values or can be optionally passed into the functions for a custom implementation.

The factory automates the deployment, setup and peering of an OFT token by calling LayerZero contracts and functions directly. This can be seen in the `configure` function, which is called by both `_CreateOFT` and `_CreateOFTAdapter`. These functions take the parameters used to create the token, as well as an array of `peer` structs, which define the peers on different networks that will automatically be added.

4.2.2 Compute Address

The factory has an overloaded function `computeAddress`. This takes the same parameters as `_CreateOFT` and `_CreateOFTAdapter`, and will return the address that those functions would deploy to. This means that it can be used to get the address of an OFT token before it is deployed, so if an OFT token is deployed to two different chains at the same time, they can automatically peer with each other on deployment. This functionality is exposed with the SDK, so using any one factory the user can find the deployment address of an OFT on any other factory. If the parameters for this being passed to `create2` are consistent (factory address, token name, symbol, etc.), then the token will be deployed to the same address on all the different chains.

4.2.3 Token Bytecode

For deploying a new OFT or OFTAdapter, the SuperFactory uses libraries to get the deployment bytecode. These libraries must be deployed first and then passed in as constructor arguments for deploying the factory. This repo provides default libraries and the SDK will automatically deploy them before deploying the factory, however, these libraries can be swapped out for custom implementations of OFT and OFTAdapter for a specific use case. The SDK exposes this functionality so any implementation of an OFT can be deployed across multiple chains as long as bytecode libraries for those are pre-deployed on each chain.

4.2.4 OAPP Admin

The OAPP factory (and therefore its implementations like the SuperFactory) maintains a mapping of deployed OAPP addresses to user addresses called admins. This admin can be thought of as the owner of the deployed OAPP (OFT), who can control the OFT through the factory. When `_CreateOFT` and `_CreateOFTAdapter` are called, the factory itself is set as the owner and delegate of the new OFT contract, but sets the sender as the admin. This means that the factory can be used in the future to remotely or locally add new peers or update the ULN config for that OFT, but will always check that the sender == admin for that OFT.

If the admin wants to take full control of a new OFT themselves without the factory, they can call `transferOwnership` on the factory to set themselves as the owner and delegate. Conversely, if a user wants to bring an already existing OFT into the Factory ecosystem, so the factory can be used to manage future deployments and changes, they can call `setAdmin`, then set the factory as the owner and delegate for that OFT.

4.2.5 Configure an OFT

The `_configureOAPP` function on the factory is called to setup a new or existing OFT, to connect it to multiple peers on other networks, so they will be able to send OFT transfers cross chain in the future. This function takes an array of `Peer` structs, which define the endpoint ID, address and ULN config for that peer. The `_configureOAPP` function iterates through this array and peers the local OFT with its multichain counterparts, and sets up the configuration so they will then be able to send messages to each other with LayerZero. This function can be called locally by the admin with the public function `_configureOAPP`, or it is called remotely when the factory receives a cross chain message.

4.3 Remote OFT Deployments

4.3.1 Remote Create

In order to deploy OFT tokens to other chains, the OAPP Factory (abstract contract that is implemented by the SuperFactory) has a `_remoteCreate` function, which calls the LayerZero endpoint contract to send a cross chain message. This message includes standard LZ data for cross chain messages, and instructions of what the receiving factory should do; i.e. deploy a new OFT token or adapter and add new peers, or configure an existing OFT contract (to add new peers). For sending messages to other EVM chains, the `_remoteCreate` function will test the cross chain message that is about to send, to check that the format is correct to execute on the receiving chain.

4.3.2 Create Multiple Contracts at Once

The `_remoteCreate` function is internal and cannot be called directly. Instead, it is called by one of two external functions, `createOappMulti` or `configureMulti`. These functions first complete an operation on the local chains

(creating or configuring an OFT), then will iterate through an array of remote messages to call `_remoteCreate` for each one, to either deploy new contracts or to configure contracts on other chains.

These functions are what allows the SuperFactory (and any other contract implementing the OAPP Factory) to deploy or update a multichain token at once with only one transaction. That transaction is split into operations that are sent to each destination chain, to deploy and setup all the individual smart contracts at once.

In order to achieve this, the single transaction that is sent to the initial chain must include all the data to tell each factory on every other chain what to do. This means, for deploying a new token to multiple chains for example, each cross chain message should include all the peers that the new token should add, and optionally what ULN config to use for each one. This would be a lot of orchestration to expect end users to do themselves, so we have implemented functionality in the SDK to do this automatically, with the functions `createOftMulti` and `configureOappMulti`.

4.3.3 Receiving a LayerZero Message

In order to receive a cross chain message, the SuperFactory overrides the `_lzReceive` function (as it is implementing the `OAPP` abstract contract). This takes the encoded data, and decodes it to `LzReceiveData` struct, which contains information on the sender and what operation the factory should carry out, as well as more encoded data, which will be used to carry out that operation, either calling `_createOFT`, `_createOFTAdapter` or `_configureOAPP`.

5 Conclusion

The development of natively cross chain token standards such as LayerZero's OFT have made is seamless to move tokens between blockchains. This solves one of the main challenges with blockchain interoperability, but it opened up a new one, that of fragmented liquidity for trading assets. For crypto and blockchain technology to reach its full potential in bringing financial services such as FX on chain, this liquidity fragmentation must be solved.

With Superset's SuperPools, we have created a system that aggregates liquidity across multiple chains, allowing for maximum capital efficiency for stablecoin trading pairs. With our incentive mechanism, we ensure that liquidity is always situated where it is needed most, so traders on any blockchain can trade on the deepest pools with no fragmentation.