RAPID
INNOVATION

UPGRADING THE USER EXPERIENCE OF HUMANITY

WELCOME

Autofarm -Diamond

A Diamond Standard Implementation of Autofarm Smart Contracts

What is Diamond Standard?

The Diamond standard is a finalized Ethereum Improvement Proposal (EIP-2535) that aims to make it easier for developers to modularize and upgrade their smart contracts.

The core idea of the Diamond standard is that you can control many implementation contracts (i.e., logic contracts) from your single Diamond contract (i.e., proxy contract).

Features of Diamond Standard

- A single gateway to make proxy calls to n number of implementation contracts
- Upgrade a single or multiple smart contract atomically
- No storage limit to how many implementation contracts you can add to your Diamond
- A log history of all the upgrades made on the Diamond
- Can reduce gas costs (i.e., by reducing the number of external function calls)

Types of Diamonds

- Upgradeable Diamond: A mutable contract that can be upgraded
- Finished Diamond: An immutable contract due to the upgradeability feature being removed
- Single Cut Diamond: An immutable contract that can no longer be upgraded

Analysis of Diamond Standard

Diamond

A Diamond is a smart contract that uses other smart contracts (aka Facets) to make delegatecall calls.
{ delegate call is a special external function call where a proxy contract borrows code from another smart contract in its own context (i.e., its own state variables }.

Analysis of Diamond Standard

Facet

A Facet can be compared to an implementation contract or library as it holds the external function logic the proxy (e.g., Diamond) contract will make calls too. You can update multiple Facets (e.g., multiple implementation contracts) or an individual Facet.

Analysis of Diamond Standard

DiamondCut

DiamondCut is a standard function that must be implemented in a Diamond contract. It allows you to add, replace or move functions from your Diamond contract. DiamondCut can be considered the "upgrade" functionality where you update the mapping (e.g., **selectorToFacet**) held in storage for your diamond contract to point to a new function signature and/or address. Whenever you call the DiamondCut function, an event will be emitted, which can be used to track the history of your upgrades.

Analysis of Diamond Standard

DiamondStorage

The DiamondStorage technique consists of declaring your state variables inside structs. Each of those structs gets a specific position in contract storage.

AppStorage

AppStorage is an alternative method of managing your state variables and storage layout. It is more efficient because it helps you share state variables between facets. With AppStorage, you create a smart contract that has a struct of all your state variables. Once deployed, you can import that AppStorage into multiple Facets.

Analysis of Diamond Standard

DiamondLoupe

A Diamond contract implements the DiamondLoupe interface. DiamondLoupe helps us see what Facets, function selectors, and facet addresses are pointing to in the Diamond contract.

What is Autofarm ?

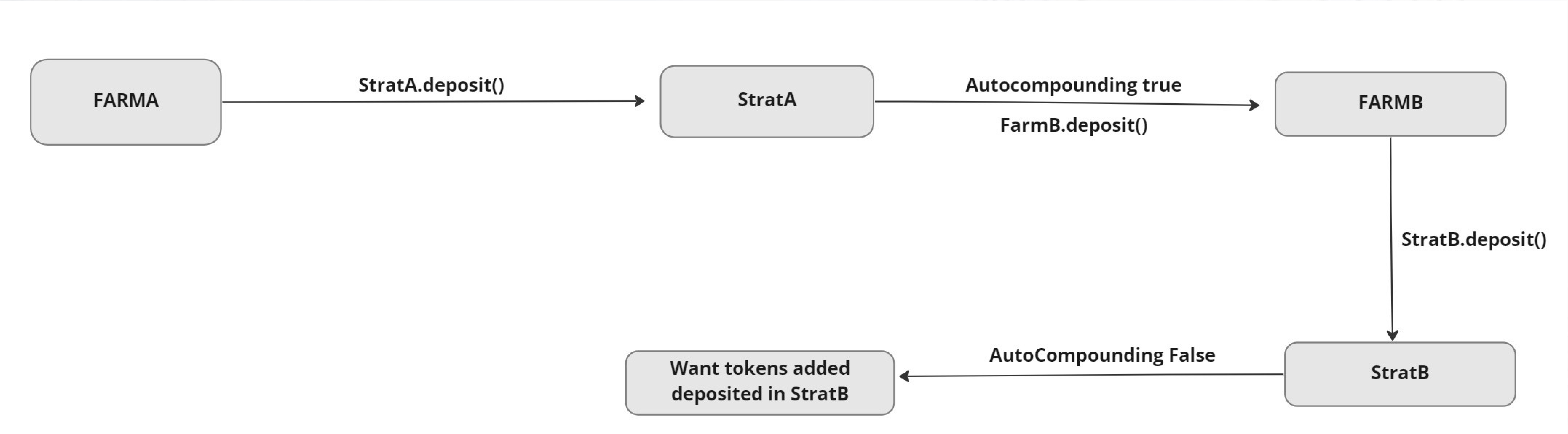
Autofarm is a decentralized finance app (Dapp) on Binance Smart Chain (BSC) for yield optimization.

It is a cross-chain yield aggregator that allows users to earn a return on their assets by staking them in Autofarm vaults.

How does Autofarm Works?

Autofarm operates in a straightforward manner. Its crypto yields automatically compound in the AutoFarm vaults, and the contracts compound at specified periods. This benefits consumers by saving them time and money on gas fees. Furthermore, users that utilize the Autofarm will receive the AutoFarm coin, $AUTO.

Autofarm Deposit flow



FARMA

StratA.deposit()

StratA

Autocompounding true

FarmB.deposit()

FARMB

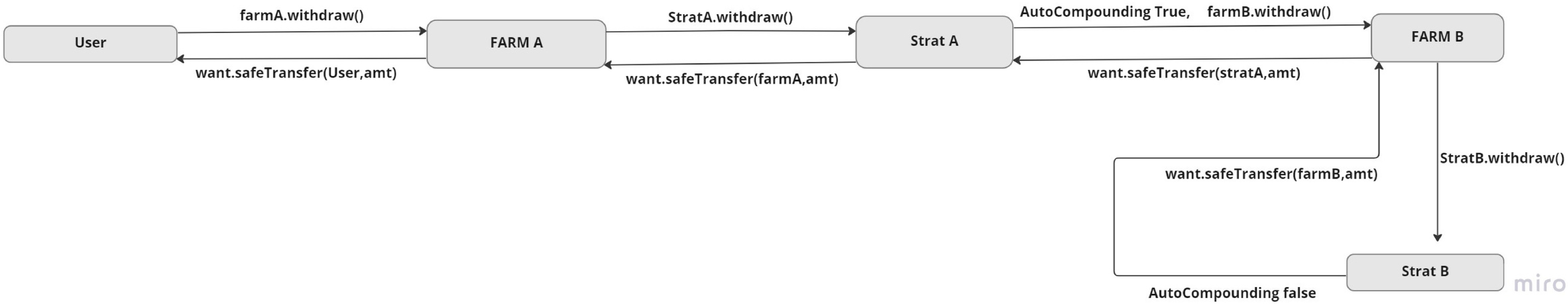
StratB.deposit()

StratB

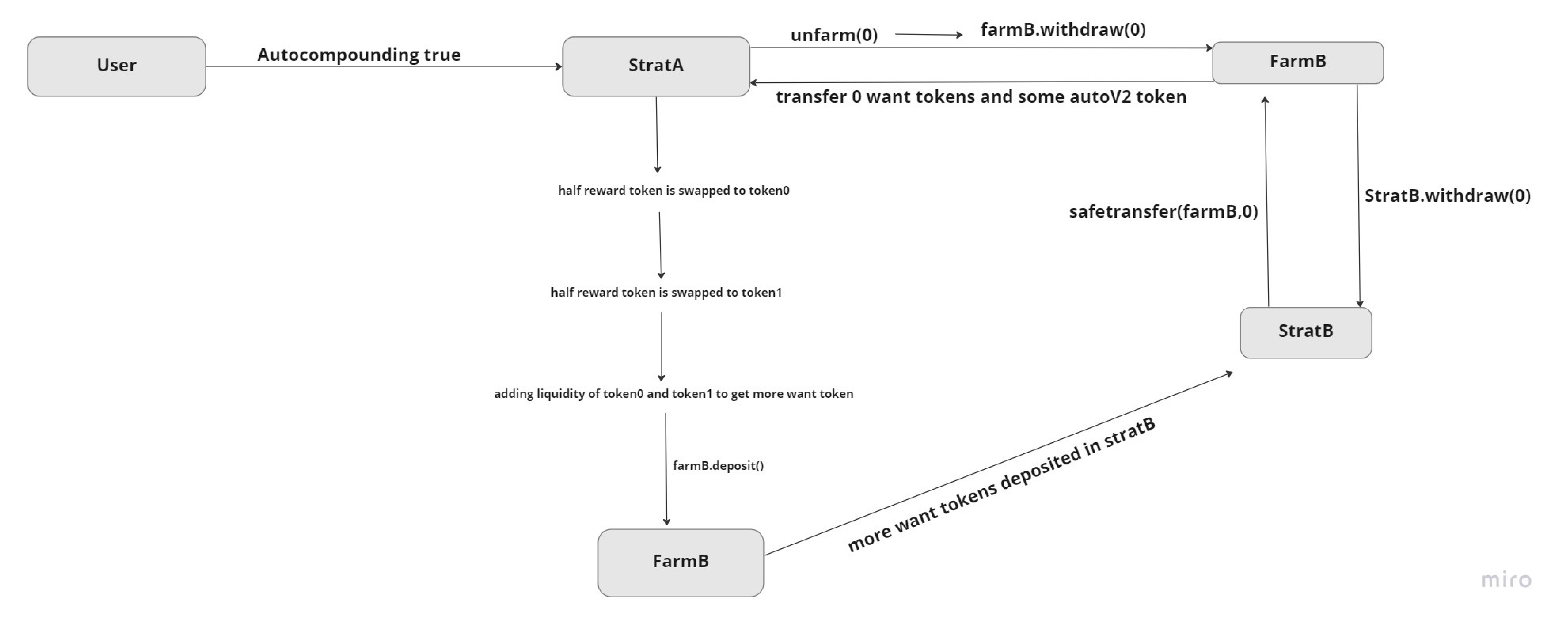
AutoCompounding False

Want tokens added deposited in StratB

Autofarm Withdraw flow



StratX2 Earn flow (AutoCompounding)


User
Autocompounding true
StratA
unfarm(0)
farmB.withdraw(0)
FarmB
transfer 0 want tokens and some autoV2 token
half reward token is swapped to token0
half reward token is swapped to token1
adding liquidity of token0 and token1 to get more want token
farmB.deposit()
FarmB
more want tokens deposited in stratB
StratB
safetransfer(farmB,0)
StratB.withdraw(0)
miro

RAPID
INNOVATION

Thank You !

Made by -Shivansh Shrivastava