PancakeSwap Infinity

March 2025

Chef Kids
kids@pancakeswap.com

Chef Jackson
jackson@pancakeswap.com

Chef Mist
mist@pancakeswap.com

Chef Momo
momo@pancakeswap.com

Chef Burger
burger@pancakeswap.com

ABSTRACT

PancakeSwap Infinity introduces a modular Automated Market Maker (AMM) architecture to promote adaptable and sustainable decentralized exchange design. The architecture facilitates multiple AMM implementations, including CLAMM and Liquidity Book, with customizable hook contracts that enable the execution of arbitrary logic at key points in a pool's operation. This modular design seamlessly integrates new pricing curves, ensuring PancakeSwap Infinity remains flexible and future-proof. Gas optimization techniques such as Singleton contracts and Flash Accounting reduce gas costs, while ERC-6909 improves efficiency and lowers costs for frequent users.

1 INTRODUCTION

PancakeSwap launched v3 in 2023, implementing the Concentrated Liquidity Automated Market Maker (CLAMM) model pioneered by Uniswap[1]. This iteration of PancakeSwap also incorporated on-chain liquidity farming within the v3 framework, partially resolving the difficulties of profitable liquidity provisioning by providing token incentives to the in-range liquidity positions. The CLAMM model revolutionized liquidity provisioning by allowing liquidity providers to supply assets within specific price ranges, thereby enhancing capital efficiency and facilitating deeper liquidity for traders. However, this innovation came with its own set of challenges, including a more complex user experience, increased impermanent loss, and higher gas costs.

Recognizing the necessity for a more adaptable and scalable solution, PancakeSwap Infinity further abstracts its architecture by decoupling Accounting logic from AMM logic. This approach ensures that PancakeSwap Infinity remains poised to accommodate evolving AMM paradigms without necessitating core protocol reimplementation.

Unlike PancakeSwap v3, which exhibited limited flexibility in supporting new functionalities atop liquidity pools, PancakeSwap Infinity adopts a more open-ended approach, giving pool creators and protocols greater customization capabilities. The integration of hooks is a key to this improvement, enabling pool creators to supplement the liquidity pools with custom features, including bespoke oracles, dynamic fees, active liquidity management strategies, diverse order types, and more. PancakeSwap Infinity maintains a non-upgradeable core, ensuring stability while allowing each pool to integrate a separate hook smart contract, determined at pool creation. Additionally, each AMM logic / Pool Type boasts an independent Singleton implementation, optimizing gas efficiency for pool creation and multi-hop transactions.

Flash Accounting complements Singleton contract design by consolidating transaction settlement by computing net balances for a batch of transactions and settling them collectively, thereby reducing gas consumption.

2 ARCHITECTURE

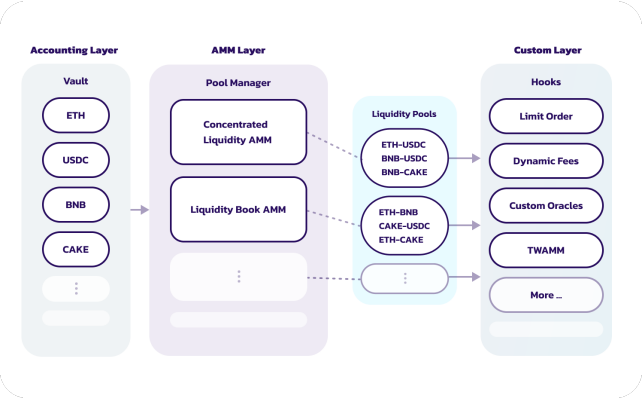

Figure 1: Three-Layered Architecture

PancakeSwap Infinity implements a three-layered modular architecture that consists of the Vault, Pool Managers, and Hooks.

This structured approach ensures a high degree of flexibility and scalability within the platform.

2.1 Vault

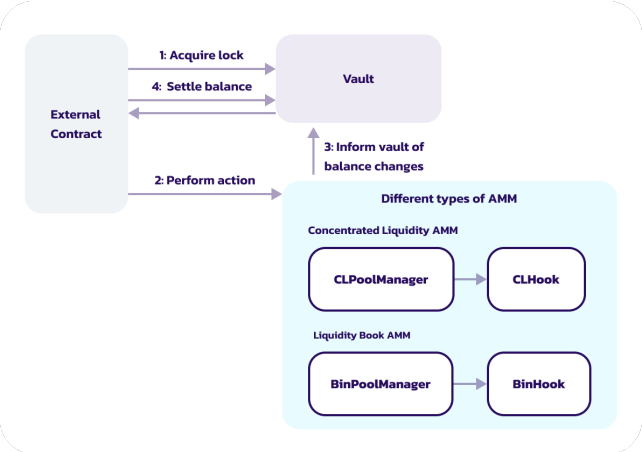

Figure 2: Vault Mechanism

At the core of this architecture is the Vault, which functions as an immutable accounting layer. It maintains a ledger of tokens that are either deposited or owed, facilitating a secure and efficient settlement process at the end of each transaction. The standard procedure for a transaction involves the following steps:

- Acquiring a lock from the vault
- Executing operations such as swaps, or adjusting liquidity levels
- For each operation, the pool manager updates the vault of the balance changes
- Reconciling the net balance with the vault (involving either depositing into or withdrawing tokens from the vault)

This method offers substantial savings on gas fees through implementation of Flash Accounting with Transient Storage[2], particularly evident in transactions involving multiple pools. For instance, in PancakeSwap v3's multi-hop swaps, like A→B→C, it's typically required to transfer all three tokens A, B, and C between the pools for A→B and B→C swaps. However, with this architecture, since token B's balance is offset during the swap, only tokens A and C need to be moved, streamlining the process significantly.

2.2 Pool Manager

Complementing the Vault are various Pool Managers, each a singleton contract that encapsulates the logic for different types of AMMs. These include but are not limited to, the CLPoolManager and BinPoolManager, as depicted in Figure 2.

CLPoolManager implements the CLAMM model, renowned for its widespread adoption and popularity. It operates on the principle of constant product invariant. On the other hand, BinPoolManager is built upon the Liquidity Book AMM model, introduced by Trader Joe[3]. It utilizes the constant sum invariant, offering a distinct approach to managing liquidity.

The AMM layer allows for the seamless integration of diverse pool types within PancakeSwap Infinity. When a new PoolManager contract is crafted, it is up to the governance body to cast its votes on whether to integrate this new manager into PancakeSwap Infinity.

2.3 Hooks

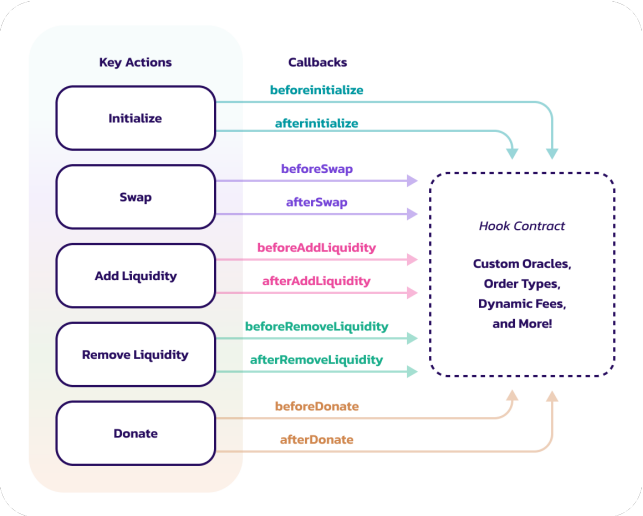

Figure 3: Hook Callbacks

Hooks are externally deployed contracts separate from the core AMM logics and Pool Managers. Anyone, including developers and protocols, can deploy these contracts, which execute predefined logic during key actions in a pool's operation, akin to widget-like add-ons, enriching liquidity pools with enhanced functionality and features.

During the initialization of a pool, the creator has the option to specify a hook contract linked to the pool. The key actions include, 'initialize', 'swap', 'addLiquidity', 'removeLiquidity', and 'donate'. Hooks empower creators to add custom logic before or after these actions, facilitating a myriad of use cases as discussed previously. PancakeSwap Infinity supports hook callbacks at ten specific points: beforeInitialize / afterInitialize, beforeAddLiquidity / afterAddLiquidity, beforeRemoveLiquidity / afterRemoveLiquidity, beforeSwap / afterSwap, and beforeDonate / afterDonate. These positions offer granular control over when hook logic is executed, enabling precise customization of pool behavior.

Hooks can allocate and distribute fees to various stakeholders, including Swappers, Hook Creator, and Liquidity Providers. The swap fee can either be static or dynamically adjusted, with hooks possessing the authority to modify them based on a predefined algorithm if they are dynamic. During pool creation, parameters including fee type (static/dynamic), hook callbacks are set. Once established, these flags remain immutable.

3 GAS OPTIMISATIONS

PancakeSwap Infinity integrates a comprehensive array of gas optimization techniques to boost efficiency and mitigate transaction costs. These optimizations encompass Singleton contract design,

Flash Accounting with Transient Storage, adherence to the ERC-6909 multi-token standard, support for native gas tokens, and exclusion of in-built features such as price oracles. The subsequent sections provide detailed explanations of these improvements.

3.1 Singleton

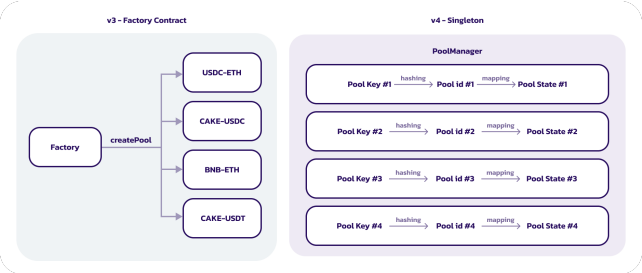

Figure 4: Singleton

In PancakeSwap v3, deploying a separate contract for each liquidity pool increased gas expenses for pool creation and multi-pool swaps. In contrast, PancakeSwap Infinity adopts a Singleton contract approach, consolidating all pools into a single contract. This architectural refinement eliminates the need to transfer tokens between different contracts during multi-hop transactions, significantly reducing gas costs.

Within the Singleton contract, data for each liquidity pool is efficiently stored in a mapping structure. The poolId, derived by hashing and converting the PoolKey struct, uniquely identifies each pool's state. The PoolKey struct encapsulates essential parameters such as token addresses, fees, hook address, poolManager address, and other pool-specific parameters like tick / bin spacing and flag for hooks, simplifying the creation and management of liquidity pools.

Furthermore, PancakeSwap Infinity supports different pool types, with each pool type featuring an independent Singleton pattern implementation.

3.2 Flash Accounting

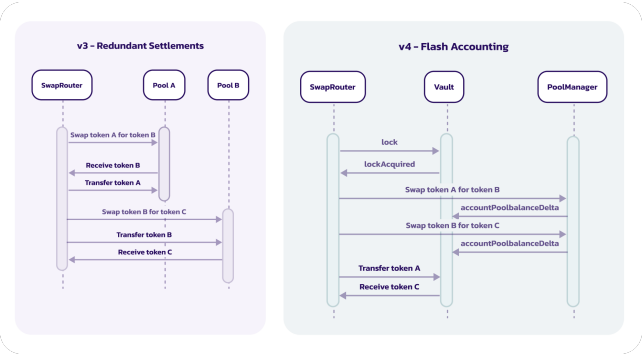

Figure 5: Flash Accounting

Flash Accounting streamlines the accounting process by computing net balances for a batch of transactions and settling them collectively. This approach significantly reduces gas consumption compared to the v3 model, where assets were transferred in and out of pools at the end of every sub-transaction.

This optimization is facilitated by the introduction of a lock mechanism. With this mechanism, when the lock is acquired each operation updates an internal net balance until all transactions are processed. Once completed, the lock is released to execute external transfers. Consequently, gas fees for multi-hop transactions are minimized.

Additionally, Flash Accounting benefits from "Transient Storage," enabled by EIP-1153. Transient storage provides a mechanism for temporarily storing data, similar to storage but with a crucial distinction: Transient storage values are discarded after every transaction, thus avoiding the serialization and deserialization processes required for permanent storage. As a result, transient storage operations are more cost-effective, as they do not incur the high gas costs associated with read and write operations to permanent storage.

3.3 ERC-6909

PancakeSwap Infinity uses ERC-6909[4] multi-token standard for accounting purposes. With the adoption of ERC-6909, users gain the ability to store tokens within the Vault contract, bypassing redundant token transfers to and from the contract.

In the usual process of swapping or managing liquidity, users either deposit or withdraw tokens from the vault upon completing their transaction. However, if there's no immediate requirement to move the tokens, users have the option to mint an ERC-6909 token instead of conducting a token transfer. In subsequent transactions, users can simply burn the ERC-6909 token to settle their account, eliminating the need to transfer tokens back to the vault. This approach can significantly reduce gas costs, especially for users who engage in frequent swapping and liquidity provisioning activities. Users can consolidate multiple token transfers across numerous trades into a single net settlement at their convenience.

Furthermore, ERC-6909 serves as a simplified alternative to the ERC-1155 standard, eliminating the requirements for callbacks and a single-operator permission model. Consequently, ERC-6909 stands out for its smaller code footprint and enhanced efficiency in gas usage.

3.4 Support for Native Gas Token

PancakeSwap Infinity also supports the use of native gas tokens to create liquidity pools. This allows for direct trading pairs with native gas tokens, eliminating the need for wrapping and unwrapping thereby reducing the gas cost further.

REFERENCES

[1] Uniswap v3 core. https://uniswap.org/whitepaper-v3.pdf, 2021.
[2] Eip-1153: Transient storage opcodes. https://eips.ethereum.org/EIPS/eip-1153, 2022.
[3] Joe v2.1 liquidity book. https://github.com/traderjoe-xyz/LB-Whitepaper/blob/main/Joev2LiquidityBookWhitepaper.pdf, 2022.
[4] Erc-6909: Minimal multi-token interface. https://eips.ethereum.org/EIPS/eip-6909, 2023.