Uniswap v4 Core [Draft]

June 2023

Hayden Adams
hayden@uniswap.org

Moody Salem
moody.salem@gmail.com

Noah Zinsmeister
noah@uniswap.org

Sara Reynolds
sara@uniswap.org

Austin Adams
austin@uniswap.org

Will Pote
pote@uniswap.org

Mark Toda
mark@uniswap.org

Alice Henshaw
alice@uniswap.org

Emily Williams
emily@uniswap.org

Dan Robinson
dan@paradigm.xyz

ABSTRACT

Uniswap v4 is a non-custodial automated market maker implemented for the Ethereum Virtual Machine. Uniswap v4 offers customizability via arbitrary code hooks, allowing developers to augment the concentrated liquidity model introduced in Uniswap v3 with new functionality. In Uniswap v4, anyone can create a new pool with a specified hook, which can run before or after predetermined pool actions. Hooks can be used to implement features that were previously built into the protocol, like oracles, as well as new features that previously would have required independent implementations of the protocol. Uniswap v4 also offers improved gas efficiency and developer experience through a singleton implementation, flash accounting, and support for native ETH.

1 INTRODUCTION

Uniswap v4 is an automated market maker (AMM) facilitating efficient exchange of value on the Ethereum Virtual Machine (EVM). As with previous versions of the Uniswap Protocol, it is non-custodial, non-upgradable, and permissionless. The focus of Uniswap v4 is on customization and architectural changes for gas efficiency upgrades, building on the AMM model built by Uniswap v1 and v2 and the concentrated liquidity model introduced in Uniswap v3.

Uniswap v1 [1] and v2 [2] were the first two iterations of the Uniswap Protocol, facilitating ERC-20 <> ETH and ERC-20 <> ERC-20 transactions, respectively, both using a constant product market maker (CPMM) model. Uniswap v3 [3] introduced concentrated liquidity, providing more capital efficient liquidity through the use of positions that provide liquidity within a limited price range, and introduced multiple fee tiers.

While concentrated liquidity and fee tiers increased flexibility for liquidity providers and allowed for new strategies to be implemented, Uniswap v3 was not flexible enough to support new functionalities invented as AMMs and markets have evolved.

Some features, like the price oracle originally introduced in Uniswap v2, allowed integrators to utilize decentralized onchain pricing data, at the expense of increased gas costs for swappers and without customizeability for integrators. Other ideas for enhancements, including time-weighted average price orders (TWAP) through a time-weighted average market maker (TWAMM) [6], volatility oracles, limit orders, or dynamic fees, would have required reimplementations of the core protocol, and could not be added to Uniswap v3 by third-party developers.

Additionally, in previous versions of Uniswap, deployment of new pools involves deploying a new contract—where cost scales with the size of the bytecode—and trades with multiple Uniswap pools involve transfers and redundant state updates across multiple contracts. And since Uniswap v2, Uniswap has required ETH to be wrapped into an ERC-20, rather than supporting native ETH. These come with gas costs.

In Uniswap v4, we improve on this through a few notable features:

- *Hooks*: Uniswap v4 allows anyone to deploy new concentrated liquidity pools with custom functionality. For each pool, the creator can define a "hook contract" that implements logic executed at key points in a call's lifecycle. These hooks can also manage the swap fee of the pool, as well as withdrawal fees charged to liquidity providers.
- *Singleton*: Uniswap v4 moves away from the factory model used in previous versions, instead implementing a single contract that holds all pools. The singleton model reduces the cost of pool creation and multi-hop trades.
- *Flash accounting*: The singleton uses "flash accounting," which requires that no tokens are owed to the pool or to the caller by the end of the lock. During the call, tokens can be used for any number of operations inside and outside the singleton. This functionality is made efficient by the transient storage opcodes proposed in EIP-1153 [4]. Flash accounting further reduces the gas cost of trades that cross multiple pools and supports more complex integrations with Uniswap v4.
- *Native ETH*: Uniswap v4 brings back support for native ETH, with support for pairs with native tokens inside v4 pools. ETH swappers and liquidity providers benefit from

gas cost reductions from cheaper transfers and removal of additional wrapping costs.

The following sections provide in-depth explanations of these changes and the architectural changes that help make them possible.

2 HOOKS

Hooks are externally deployed contracts that execute some developer-defined logic at a specified point in a pool's execution. These hooks allow integrators to create a concentrated liquidity pool with flexible and customizable execution.

Hooks can modify pool parameters, or add new features and functionality. Example functionalities that could be implemented with hooks include:

- Executing large orders over time through TWAMM
- Onchain limit orders that fill at tick prices
- Volatility-shifting dynamic fees
- Mechanisms to internalize MEV for liquidity providers
- Median, truncated, or other custom oracle implementations

We envision that future independent whitepapers will be drafted for selected hook designs, as many will be as complex as protocols themselves.

2.1 Action Hooks

When someone creates a pool on Uniswap v4, they can specify a hook contract. This hook contract implements custom logic that the pool will call out to during its execution. Uniswap v4 currently supports eight such hook callbacks:

- beforeInitialize/afterInitialize
- beforeModifyPosition/afterModifyPosition
- beforeSwap/afterSwap
- beforeDonate/afterDonate

The address of the hook contract determines which of these hook callbacks are executed. This creates a gas efficient and expressive methodology for determining the desired callbacks to execute, and ensures that even upgradeable hooks obey certain invariants. There are minimal requirements for creating a working hook. In Figure 1, we describe how the beforeSwap and afterSwap hooks work as part of swap execution flow.

2.2 Hook-managed fees

Uniswap v4 allows fees to be taken on both swapping and withdrawing liquidity.

Swap fees can be either static, or dynamically managed by a hook contract. The hook contract can also choose to allocate a percentage of the swap fees to itself. Withdrawal fees cannot be set natively in the pool. To set a withdrawal fee, pool creators must set a hook contract to manage that fee, and the collected withdrawal fees go to the hook contract. Fees that accrue to hook contracts can be allocated arbitrarily by the hook's code, including to liquidity providers, swappers, hook creators, or any other party.

The capabilities of the hook are limited by immutable flags chosen when the pool is created. The fee settings a pool creator can choose are:

- Whether a pool has a static fee (and what that fee is) or dynamic fees
- Permission for a hook to take a percentage of swap fees
- Permission for a hook to take withdrawal fees

Governance also can take a capped percentage of swap or withdrawal fees, as discussed below in the Governance section.

3 SINGLETON AND FLASH ACCOUNTING

Previous versions of the Uniswap Protocol use the factory/pool pattern, where the factory creates separate contracts for new token pairs. Uniswap v4 uses a *singleton* design pattern where all pools are managed by a single contract, making pool deployment 99% cheaper.

The singleton design complements another architectural change in v4: *flash accounting*. In previous versions of the Uniswap Protocol, each operation (such as swapping or adding liquidity to a pool) ended by transferring tokens. In v4, each operation updates an internal net balance, known as a `delta`, only making external transfers at the end of the lock. The new `take()` and `settle()` functions can be used to borrow or deposit funds to the pool, respectively. By requiring that no tokens are owed to the pool manager or to the caller by the end of the call, the pool's solvency is enforced.

Flash accounting simplifies complex pool operations, such as atomic swapping and adding. When combined with the singleton model, it also simplifies multi-hop trades.

In the current execution environment, the flash accounting architecture is expensive because it requires storage updates at every balance change. Even though the contract guarantees that internal accounting data is never actually serialized to storage, users will still pay those same costs once the storage refund cap is exceeded [5]. But, because balances must be 0 by the end of the transaction, accounting for these balances can be implemented with transient storage, as specified by EIP-1153 [4].

Together, singleton and flash accounting enable more efficient routing across multiple v4 pools, reducing the cost of liquidity fragmentation. This is especially useful given the introduction of hooks, which will greatly increase the number of pools.

4 NATIVE ETH

Uniswap v4 is bringing back native ETH in trading pairs. While Uniswap v1 was strictly ETH paired against ERC-20 tokens, native ETH pairs were removed in Uniswap v2 due to implementation complexity and concerns of liquidity fragmentation across WETH and ETH pairs. Singleton and flash accounting mitigate these problems, so Uniswap v4 allows for both WETH and ETH pairs.

Native ETH transfers are about half the gas cost of ERC-20 transfers (21k gas for ETH and around 40k gas for ERC-20s). Currently Uniswap v2 and v3 require the vast majority of users to wrap (unwrap) their ETH to (from) WETH before (after) trading on the Uniswap Protocol, requiring extra gas.

5 OTHER NOTABLE FEATURES

5.1 ERC1155 Accounting

Uniswap v4 will support the minting/burning of singleton-implemented ERC-1155 tokens for additional token accounting. Users can now keep tokens within the singleton and avoid ERC-20 transfers to and from the contract. This will be especially valuable for users and

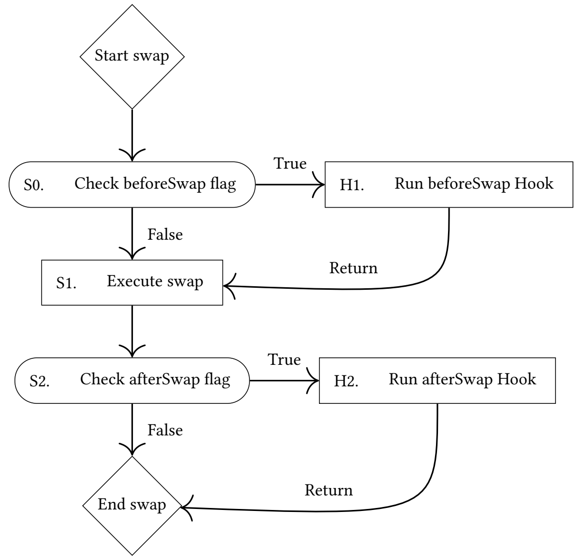

Figure 1: Swap Hook Flow

hooks who continually use the same tokens over multiple blocks or transactions, like frequent swappers or liquidity providers.

5.2 Governance updates

Uniswap v4 has two separate governance fee mechanisms, swap fees and withdrawal fees, each with different mechanisms. First, similar to Uniswap v3, governance can elect to take up to a capped percentage of the swap fee on a particular pool. With v4, if hooks initially choose to turn on withdrawal fees for a pool, governance also has the ability to take up to a capped percentage of that withdrawal fee. Unlike in Uniswap v3, governance does not control the permissible fee tiers or tick spacings.

5.3 Gas reductions

As discussed above, Uniswap v4 introduces meaningful gas optimizations through flash accounting, the singleton model, and support for native ETH. Additionally, the introduction of hooks makes the protocol-enshrined price oracle that was included in Uniswap v2 and Uniswap v3 unnecessary, which also means some pools could forgo the oracle altogether and save around 15k gas on the first swap on a pool in each block.

5.4 donate()

`donate()` allows users, integrators, and hooks to directly pay in-range liquidity providers in either or both of the tokens of the pool. This functionality relies on the fee accounting system to facilitate efficient payments. The fee payment system can only support either of the tokens in the token pair for the pool. Potential use-cases could be tipping in-range liquidity providers on TWAMM orders or new types of fee systems.

6 SUMMARY

In summary, Uniswap v4 is a non-custodial, non-upgradeable, and permissionless AMM protocol. It builds upon the concentrated liquidity model introduced in Uniswap v3 with customizable pools through hooks. Complementary to hooks are other architectural changes like the singleton contract which holds all pool state in one contract, and flash accounting which enforces pool solvency across each pool efficiently. Some other improvements are native ETH support, ERC-1155 balance accounting, new fee mechanisms, and the ability to donate to in-range liquidity providers.

REFERENCES

[1] Hayden Adams. 2018. *Uniswap v1 Core.* Retrieved Jun 12, 2023 from https://hackmd.io/@HaydenAdams/HJ9jLsfTz
[2] Hayden Adams, Noah Zinsmeister, and Dan Robinson. 2020. *Uniswap v2 Core.* Retrieved Jun 12, 2023 from https://uniswap.org/whitepaper.pdf
[3] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. 2021. *Uniswap v3 Core.* Retrieved Jun 12, 2023 from https://uniswap.org/whitepaper-v3.pdf
[4] Alexey Akhunov and Moody Salem. 2018. *EIP-1153: Transient storage opcodes.* Retrieved Jun 12, 2023 from https://eips.ethereum.org/EIPS/eip-1153
[5] Vitalik Buterin and Martin Swende. 2021. *EIP-3529: Reduction in refunds.* Retrieved Jun 12, 2023 from https://eips.ethereum.org/EIPS/eip-3529
[6] Dave White, Dan Robinson, and Hayden Adams. 2021. *TWAMM.* Retrieved Jun 12, 2023 from https://www.paradigm.xyz/2021/07/twamm

DISCLAIMER