Uniswap v4 Core

August 2024

Hayden Adams
hayden@uniswap.org

Moody Salem
moody.salem@gmail.com

Noah Zinsmeister
noah@uniswap.org

Sara Reynolds
sara@uniswap.org

Austin Adams
me@aada.ms

Will Pote
willpote@gmail.com

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 additional customization for developers and architectural changes for gas efficiency improvements, building on the AMM model built by Uniswap v1 and v2 and the concentrated liquidity model introduced in Uniswap v3.

Uniswap v1 [2] and v2 [3] were the first two iterations of the Uniswap Protocol, facilitating ERC-20 <> ETH and ERC-20 <> ERC-20 swaps, respectively, both using a constant product market maker (CPMM) model. Uniswap v3 [4] introduced concentrated liquidity, enabling more capital efficient liquidity through positions that provide liquidity within a limited price range, and multiple fee tiers.

While concentrated liquidity and fee tiers increased flexibility for liquidity providers and allowed for new liquidity provision strategies, Uniswap v3 lacks flexibility to support new functionalities invented as AMMs and DeFi have evolved.

Some features, like the price oracle originally introduced in Uniswap v2 and included in Uniswap v3, allow integrators to utilize decentralized onchain pricing data, at the expense of increased gas costs for swappers and without customizability for integrators. Other possible enhancements, such as time-weighted average price orders (TWAP) through a time-weighted average market maker (TWAMM) [8], volatility oracles, limit orders, or dynamic fees, require reimplementations of the core protocol, and can 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. Additionally since Uniswap v2, Uniswap has required ETH to be wrapped into an ERC-20, rather than supporting native ETH. These design choices came with increased gas costs for end users.

In Uniswap v4, we improve on these inefficiencies 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 specific points in a call's lifecycle. These hooks can also manage the swap fee of the pool dynamically, implement custom curves, and adjust fees charged to liquidity providers and swappers though *Custom Accounting*.
- *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 allows a caller to lock the pool and access any of its tokens, as long as no tokens are owed to or from the caller by the end of the lock. This functionality is made efficient by the transient storage opcodes described in EIP-1153 [5]. 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.
- *Custom Accounting*: The singleton supports both augmenting and bypassing the native concentrated liquidity pools through hook-returned deltas, utilizing the singleton as an immutable settlement layer for connected pools. This feature can support use-cases like hook withdrawal fees, wrapping assets, or constant product market maker curves like Uniswap v2.

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. Optionally, hooks can also return custom deltas that allow the hook to change the behavior of the swap — described in detail in the *Custom Accounting* section (5).

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 [8]
- Onchain limit orders that fill at tick prices
- Volatility-shifting dynamic fees
- Mechanisms to internalize MEV for liquidity providers [1]
- Median, truncated, or other custom oracle implementations
- Constant Product Market Makers (Uniswap v2 functionality)

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 ten such hook callbacks:

- beforeInitialize/afterInitialize
- beforeAddLiquidity/afterAddLiquidity[1]
- beforeRemoveLiquidity/afterRemoveLiquidity
- 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.

[1]Having separate permissions for 'beforeAddLiquidity' and 'beforeRemoveLiquidity' reflects the difference in security assumptions between those two actions. Hooks that can affect minting but not burning of liquidity are safer for liquidity providers, since they are guaranteed to be able to withdraw their liquidity.

2.2 Hook-managed fees

Uniswap v4 allows fees to be taken on swapping by the hook.

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. 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. For example, a pool creator can choose whether a pool has a static fee (and what that fee is) or dynamic fees.

Governance also can take a capped percentage of swap fees, as discussed below in the Governance section (6.2).

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, most operations (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 or compound operations like swapping before adding liquidity.

Before the Cancun hard fork, the flash accounting architecture was expensive because it required storage updates at every balance change. Even though the contract guaranteed that internal accounting data is never actually serialized to storage, users would still pay those same costs once the storage refund cap was exceeded [6]. 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 [5].

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

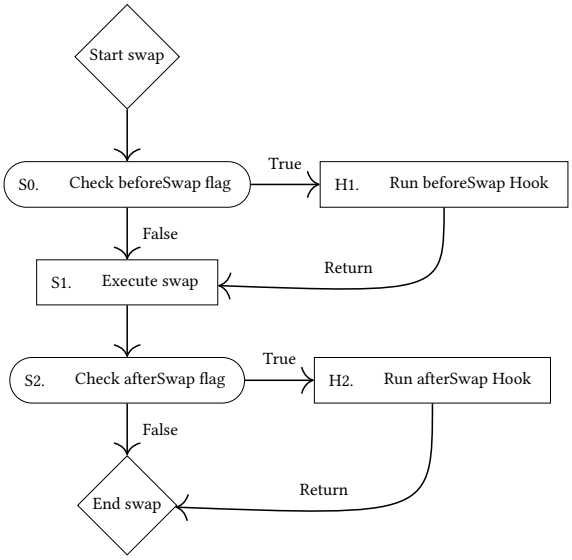

Figure 1: Swap Hook Flow

(unwrap) their ETH to (from) WETH before (after) trading on the Uniswap Protocol, requiring extra gas. According to transaction data, the majority of users start or end their transactions in ETH, adding this additional unneeded complexity.

5 CUSTOM ACCOUNTING

Newly introduced in Uniswap v4 is custom accounting - which allows hook developers to alter end user actions utilizing hook-returned deltas, token amounts that are debited/credited to the user and credited/debited to the hook, respectively. This allows hook developers to potentially add withdrawal fees on LP positions, customized LP fee models, or match against some flow, all while ultimately utilizing the internal concentrated liquidity native to Uniswap v4.

Importantly, hook developers can also forgo the concentrated liquidity model entirely, creating custom curves from the v4 swap parameters. This creates interface composability for integrators - allowing the hook to map the swap parameters to their internal logic.

In Uniswap v3, users were required to utilize the concentrated liquidity AMM introduced in the same version. Since their introduction, concentrated liquidity AMMs have become widely used as the base liquidity provision strategy in the decentralized finance markets. While concentrated liquidity is able to support most arbitrary liquidity provision strategies, it may require increased gas overhead to implement specific strategies.

One possible example is a Uniswap v2 on Uniswap v4 hook, which bypasses the internal concentrated liquidity model entirely - utilizing a constant product market maker fully inside of the hook. Using custom accounting is cheaper than creating a similar strategy in the concentrated liquidity math.

The benefit of custom accounting for developers - compared to rolling a custom AMM - is the singleton, flash accounting, and ERC-6909. These features support cheaper multi-hop swaps, security benefits, and easier integration for flow. Developers should also benefit from a well-audited code-base for the basis of their AMM.

Custom accounting will also support experimentation in liquidity provision strategies, which historically requires the creation of an entirely new AMM. Creating a custom AMM requires significant technical resources and investment, which may not be economically viable for many.

6 OTHER NOTABLE FEATURES

6.1 ERC-6909 Accounting

Uniswap v4 supports the minting/burning of singleton-implemented ERC-6909 tokens for additional token accounting, described in the ERC-6909 specification [7]. 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 hooks who continually use the same tokens over multiple blocks or transactions, like frequent swappers, liquidity providers, or custom accounting hooks.

6.2 Governance updates

Similar to Uniswap v3, Uniswap v4 allows governance the ability to take up to a capped percentage of the swap fee on a particular pool, which are additive to LP fees. Unlike in Uniswap v3, governance does not control the permissible fee tiers or tick spacings.

6.3 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 base pools forgo the oracle altogether and save around 15k gas on the first swap on a pool in each block.

6.4 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.

7 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. Additionally, hook developers can elect to bypass the concentrated liquidity entirely, utilizing the v4 singleton as an arbitrary delta resolver. Some other improvements are native ETH support, ERC-6909 balance accounting, new fee mechanisms, and the ability to donate to in-range liquidity providers.

REFERENCES

[1] Austin Adams, Ciamac Moallemi, Sara Reynolds, and Dan Robinson. 2024. am-AMM: An Auction-Managed Automated Market Maker. *arXiv preprint arXiv:2403.03367* (2024).

[2] Hayden Adams. 2018. *Uniswap v1 Core.* Retrieved Jun 12, 2023 from https://hackmd.io/@HaydenAdams/HJ9jLsfTz

[3] Hayden Adams, Noah Zinsmeister, and Dan Robinson. 2020. *Uniswap v2 Core.* Retrieved Jun 12, 2023 from https://uniswap.org/whitepaper.pdf

[4] 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

[5] Alexey Akhunov and Moody Salem. 2018. *EIP-1153: Transient storage opcodes.* Retrieved Jun 12, 2023 from https://eips.ethereum.org/EIPS/eip-1153

[6] Vitalik Buterin and Martin Swende. 2021. *EIP-3529: Reduction in refunds.* Retrieved Jun 12, 2023 from https://eips.ethereum.org/EIPS/eip-3529

[7] JT Riley, Dillon, Sara, Vectorized, and Neodaoist. 2023. *ERC-6909: Minimal Multi-Token Interface.* Retrieved Aug 26, 2024 from https://eips.ethereum.org/EIPS/eip-6909

[8] Dave White, Dan Robinson, and Hayden Adams. 2021. *TWAMM.* Retrieved Jun 12, 2023 from https://www.paradigm.xyz/2021/07/twamm

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 and is not made on behalf of Uniswap Labs, Paradigm, or their affiliates and does not necessarily reflect the opinions of Uniswap Labs, Paradigm, their affiliates or individuals associated with them. The opinions reflected herein are subject to change without being updated.