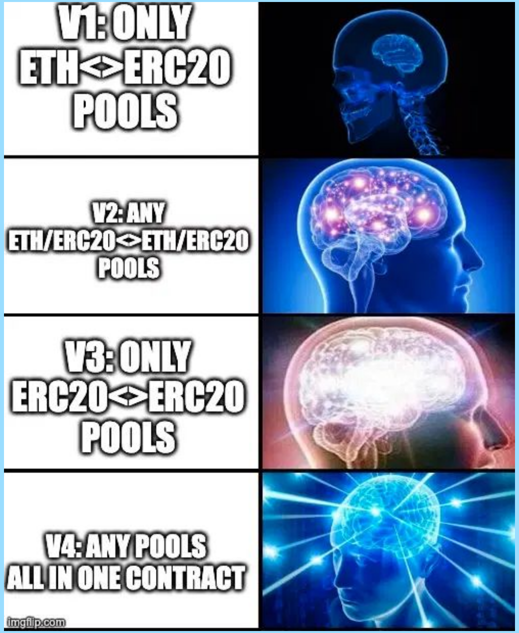
Uniswap V4: Road to Captain Hook

All the things you need to know to start hook engineering


V1: ONLY ETH<>ERC20 POOLS
V2: ANY ETH/ERC20<>ETH/ERC20 POOLS
V3: ONLY ERC20<>ERC20 POOLS
V4: ANY POOLS ALL IN ONE CONTRACT
imgflip.com

Ryan Wang

Core Contributor
@ Harvest Finance

Experience

- Contributor, Unirep Social TW
- PSE Summer Contribution Fellow
- Teaching Assistant, CSIC30161 Blockchain Dev. and FinTech
- PSE Bounty, ETH Taipei 2023

TABLE OF CONTENTS

01

WATS UNIV4

- Unique Feature Overview
- What UniV4 does?
- What can Hooks do?

02

WHY UNIV4

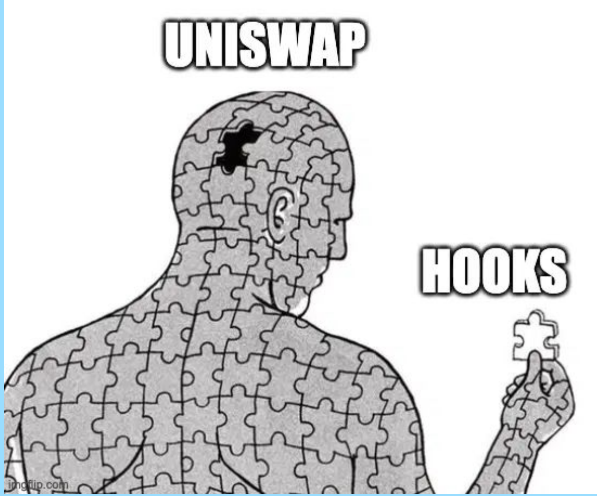
03

HOW UNIV4

- Wat with Hooks
- Kool Implementations

WATS UNIV4

Unique Feature / What UniV4 Does / What can Hook do



Before that, what´s UniV3

Uniswap V3

is a concentrated liquidity decentralized exchange.

Providing more capital efficient liquidity through the use of positions (ERC721) that provide liquidity within a limited price range, and introduced multiple fee tiers.

However, Uniswap v3 was not flexible enough to support new functionalities invented as AMMs and markets have evolved.

Now, UniV4

Uniswap V4

is Uniswap V3 with a Functionality Marketplace.

Turns Uniswap from a protocol into an engine and platform.

Bankless: V4 to Uniswap is like Rollups to Mainnet.

Wen?

Months to come, due to the Transient Storage (In Cancun fork)

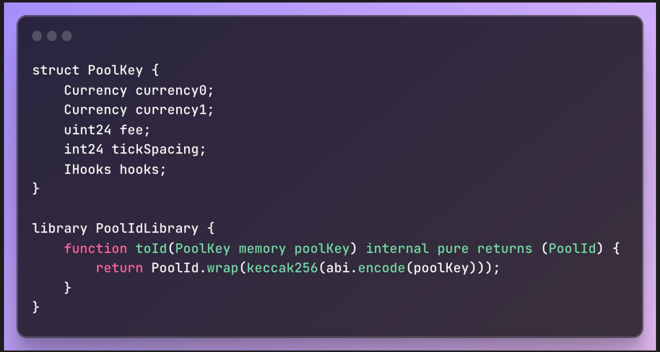
Unique Features

Hoooooks

Key feature of V4
Capable of giving custom functionality to a pool

Singleton

~~PoolFactory,~~ no more. Single contract using *struct PoolKey* to poolId mapping. And replaced ERC721 with ERC1155 accounting
Saving 99% gas



```
struct PoolKey {
    Currency currency0;
    Currency currency1;
    uint24 fee;
    int24 tickSpacing;
    IHooks hooks;
}

library PoolIdLibrary {
    function toId(PoolKey memory poolKey) internal pure returns (PoolId) {
        return PoolId.wrap(keccak256(abi.encode(poolKey)));
    }
}
```

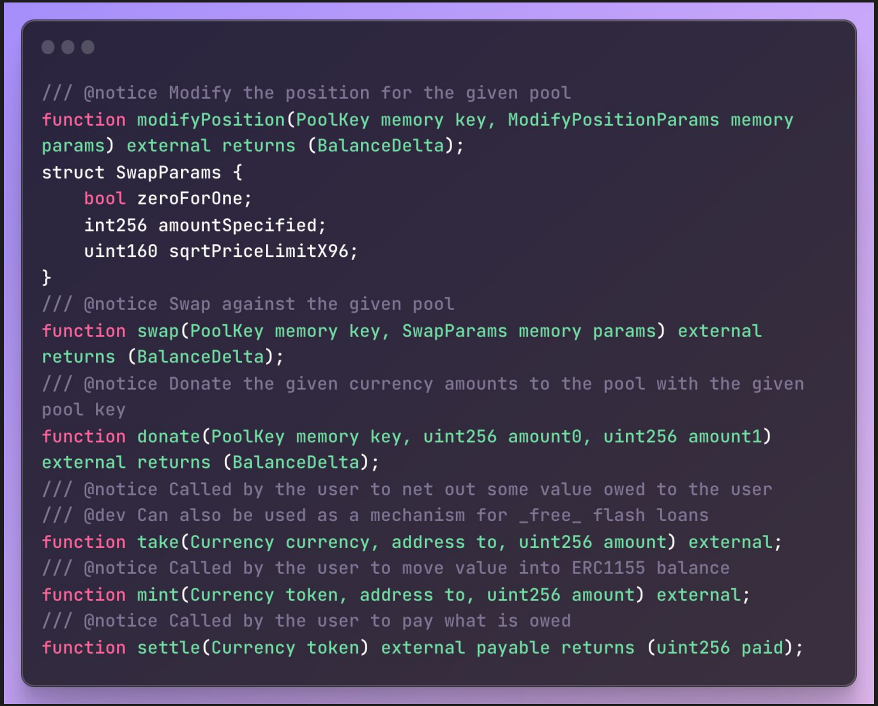
Flash Accounting

Introducing a lock mechanism. User need to acquire the lock before any actions. And before releasing the lock the accountDelta should be 0. Meaning user owes no token to the pool and the pool owes no token to the user. Related topics: Lock and Transient Storage

Native ETH

What UniV4 Does

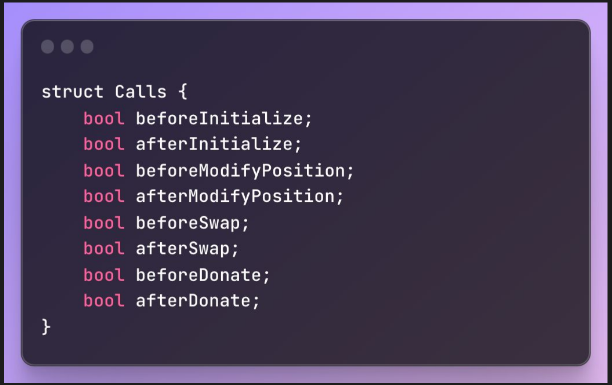
1. **Modify Position**
   Add / Remove Liquidity and Change Range
2. **Donate**
   Send Money to In-range LPs
3. **Mint**
   Mint ERC1155 token as LP, and the tokenId is from ERC20 token address
4. **Take**
   Borrow money from the pool (Free Flash Loan)
5. **Settle**
   Return the money to the pool
6. **Mint**



```
/// @notice Modify the position for the given pool
function modifyPosition(PoolKey memory key, ModifyPositionParams memory
params) external returns (BalanceDelta);
struct SwapParams {
    bool zeroForOne;
    int256 amountSpecified;
    uint160 sqrtPriceLimitX96;
}
/// @notice Swap against the given pool
function swap(PoolKey memory key, SwapParams memory params) external
returns (BalanceDelta);
/// @notice Donate the given currency amounts to the pool with the given
pool key
function donate(PoolKey memory key, uint256 amount0, uint256 amount1)
external returns (BalanceDelta);
/// @notice Called by the user to net out some value owed to the user
/// @dev Can also be used as a mechanism for _free_ flash loans
function take(Currency currency, address to, uint256 amount) external;
/// @notice Called by the user to move value into ERC1155 balance
function mint(Currency token, address to, uint256 amount) external;
/// @notice Called by the user to pay what is owed
function settle(Currency token) external payable returns (uint256 paid);
```

What Can Hook Do

1. beforeInitiation
2. afterInitiation
3. beforeModifyPosition
4. afterModifyPosition
5. beforeSwap
6. afterSwap
7. beforeDonate
8. afterDonate



```
struct Calls {
    bool beforeInitialize;
    bool afterInitialize;
    bool beforeModifyPosition;
    bool afterModifyPosition;
    bool beforeSwap;
    bool afterSwap;
    bool beforeDonate;
    bool afterDonate;
}
```

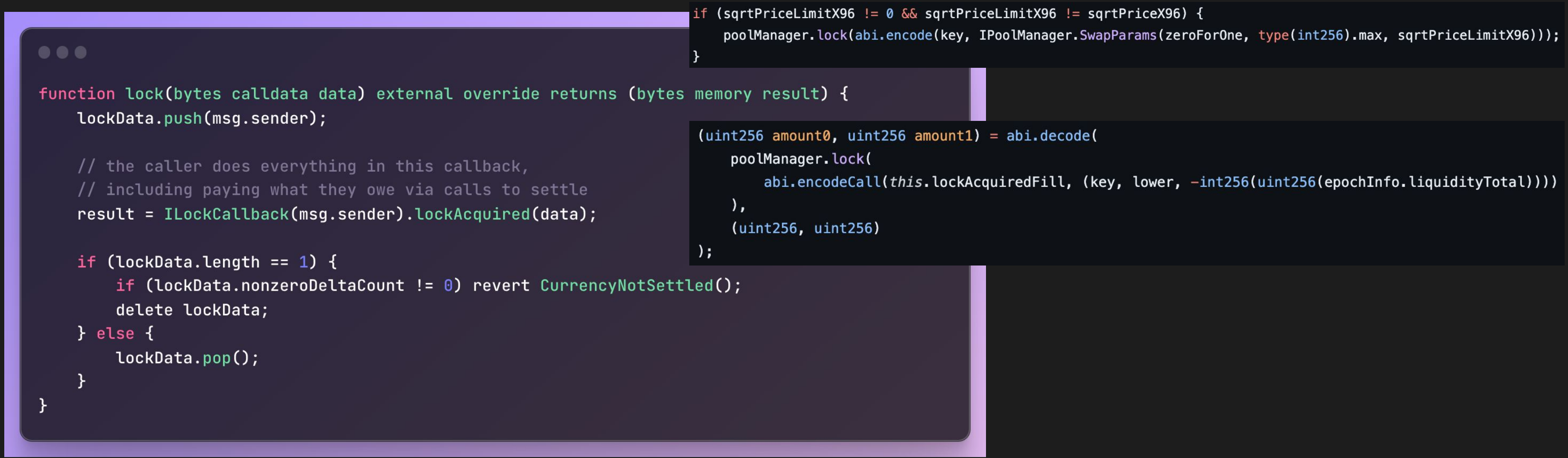
Full Cycle of UniV4 Actions

1. Get the lock
2. Activate Pre Hook
3. Fee Calculation
4. Execute action
5. Do account delta check
6. Activate Post Hook

Full Cycle of UniV4 Actions

1. Get the lock

All actions go through lock, and being executed with callback function



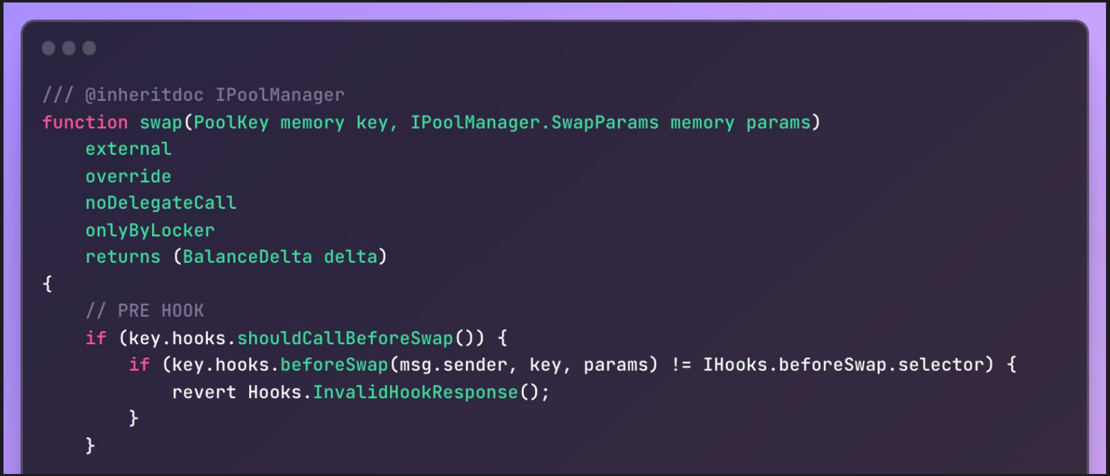
```
function lock(bytes calldata data) external override returns (bytes memory result) {
    lockData.push(msg.sender);

    // the caller does everything in this callback,
    // including paying what they owe via calls to settle
    result = ILockCallback(msg.sender).lockAcquired(data);

    if (lockData.length == 1) {
        if (lockData.nonzeroDeltaCount != 0) revert CurrencyNotSettled();
        delete lockData;
    } else {
        lockData.pop();
    }
}
```

```
if (sqrtPriceLimitX96 != 0 && sqrtPriceLimitX96 != sqrtPriceX96) {
    poolManager.lock(abi.encode(key, IPoolManager.SwapParams(zeroForOne, type(int256).max, sqrtPriceLimitX96)));
}
```

```
(uint256 amount0, uint256 amount1) = abi.decode(
    poolManager.lock(
        abi.encodeCall(this.lockAcquiredFill, (key, lower, -int256(uint256(epochInfo.liquidityTotal))))
    ),
    (uint256, uint256)
);
```

Full Cycle of UniV4 Actions

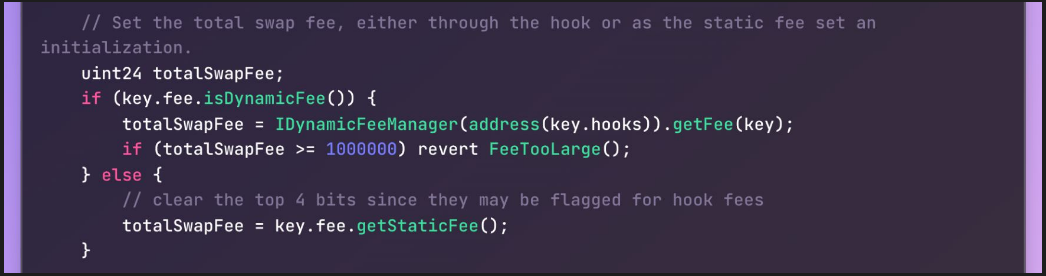
2. Activate Pre Hook



```
/// @inheritdoc IPoolManager
function swap(PoolKey memory key, IPoolManager.SwapParams memory params)
    external
    override
    noDelegateCall
    onlyByLocker
    returns (BalanceDelta delta)
{
    // PRE HOOK
    if (key.hooks.shouldCallBeforeSwap()) {
        if (key.hooks.beforeSwap(msg.sender, key, params) != IHooks.beforeSwap.selector) {
            revert Hooks.InvalidHookResponse();
        }
    }
```

Full Cycle of UniV4 Actions

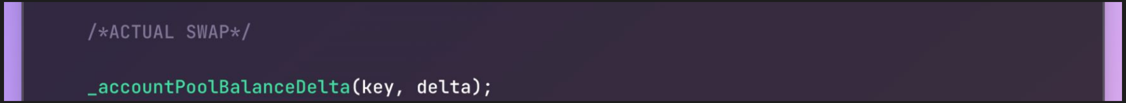
3. Fee Calculation



```
    // Set the total swap fee, either through the hook or as the static fee set an
initialization.
    uint24 totalSwapFee;
    if (key.fee.isDynamicFee()) {
        totalSwapFee = IDynamicFeeManager(address(key.hooks)).getFee(key);
        if (totalSwapFee >= 1000000) revert FeeTooLarge();
    } else {
        // clear the top 4 bits since they may be flagged for hook fees
        totalSwapFee = key.fee.getStaticFee();
    }
```

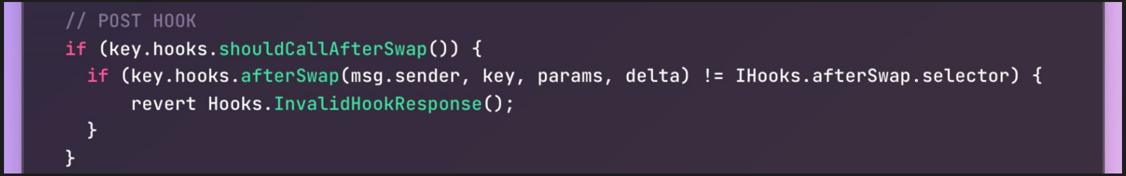
Full Cycle of UniV4 Actions

4. Execute action
5. Do account delta check



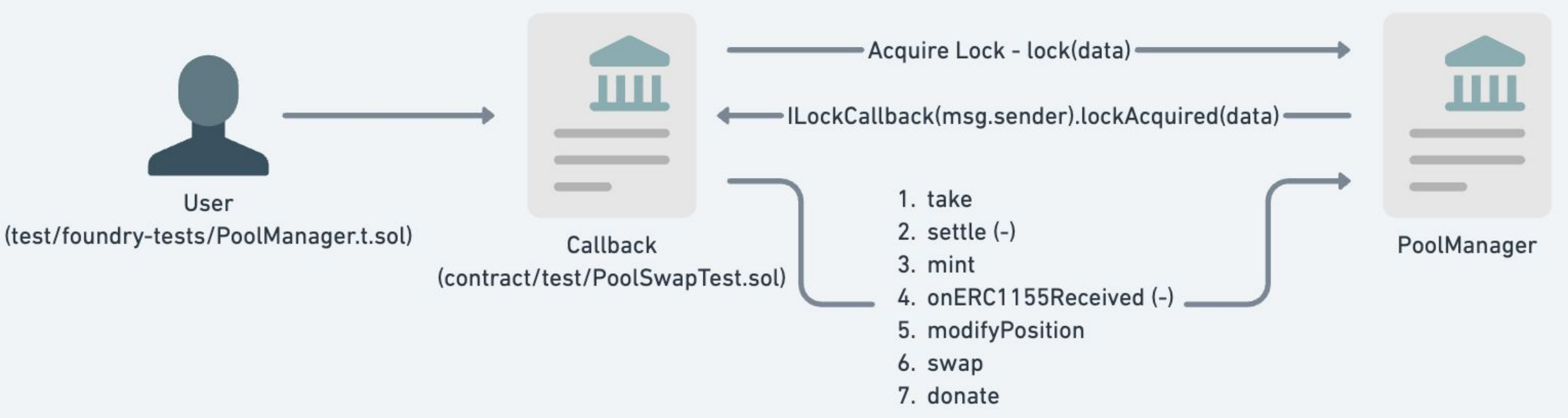
```
/*ACTUAL SWAP*/

_accountPoolBalanceDelta(key, delta);
```

6. Activate Post Hook



```
// POST HOOK
if (key.hooks.shouldCallAfterSwap()) {
  if (key.hooks.afterSwap(msg.sender, key, params, delta) != IHooks.afterSwap.selector) {
      revert Hooks.InvalidHookResponse();
  }
}
```

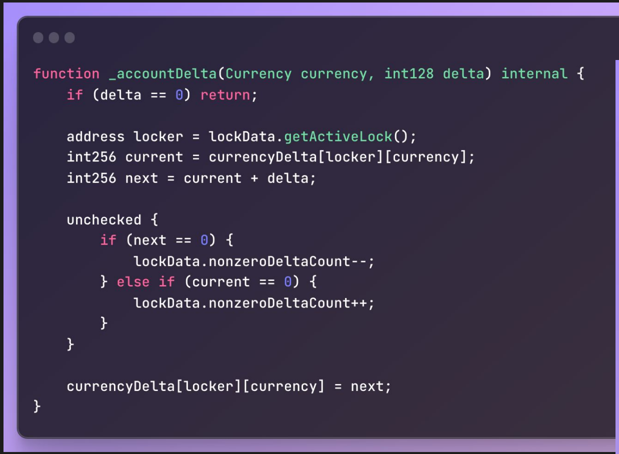
Flash Accounting


User
(test/foundry-tests/PoolManager.t.sol)
Callback
(contract/test/PoolSwapTest.sol)
Acquire Lock - lock(data)
ILockCallback(msg.sender).lockAcquired(data)
PoolManager

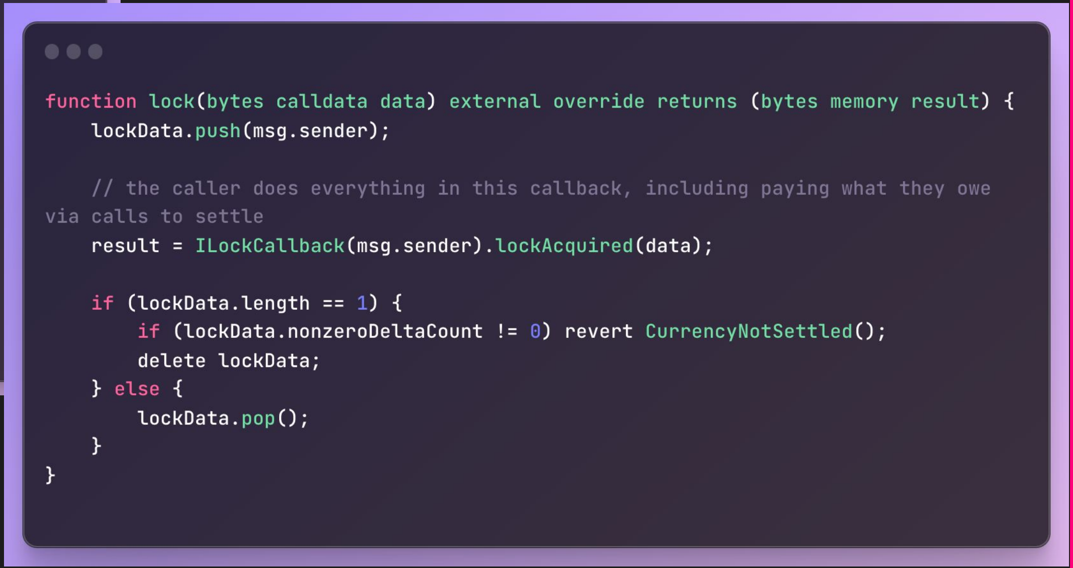
1. take
2. settle (-)
3. mint
4. onERC1155Received (-)
5. modifyPosition
6. swap
7. donate

Flash Accounting

Every moves go through _accountDelta



```
function _accountDelta(Currency currency, int128 delta) internal {
    if (delta == 0) return;

    address locker = lockData.getActiveLock();
    int256 current = currencyDelta[locker][currency];
    int256 next = current + delta;

    unchecked {
        if (next == 0) {
            lockData.nonzeroDeltaCount--;
        } else if (current == 0) {
            lockData.nonzeroDeltaCount++;
        }
    }

    currencyDelta[locker][currency] = next;
}
```



```
function lock(bytes calldata data) external override returns (bytes memory result) {
    lockData.push(msg.sender);

    // the caller does everything in this callback, including paying what they owe
via calls to settle
    result = ILockCallback(msg.sender).lockAcquired(data);

    if (lockData.length == 1) {
        if (lockData.nonzeroDeltaCount != 0) revert CurrencyNotSettled();
        delete lockData;
    } else {
        lockData.pop();
    }
}
```

Flash Accounting

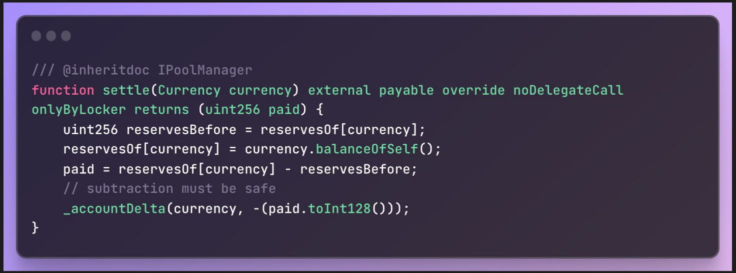
So every moves in UniV4 should end with

- take()
- settle()
- mint()
- onERC1155Received()

Flash Accounting

So every moves in UniV4 should end with

- take[]
  The only function in UniV4 with transfer
- settle[]
  Use balanceOf to check how much it received
- mint[]
- onERC1155Received[]



```
/// @inheritdoc IPoolManager
function settle(Currency currency) external payable override noDelegateCall
onlyByLocker returns (uint256 paid) {
    uint256 reservesBefore = reservesOf[currency];
    reservesOf[currency] = currency.balanceOfSelf();
    paid = reservesOf[currency] - reservesBefore;
    // subtraction must be safe
    _accountDelta(currency, -(paid.toInt128()));
}
```

WHY UNIV4

⚡ DeFi HASN'T INNOVATED SINCE UNISWAP V3
CHANGE MY MIND
LOUDER CROWDER

What´s unfinished with V3 ?

Oracle

Oracle in V3 makes swappers responsible for updating the observation. And it doesn't comes with customize oracle.

Fixed Fees

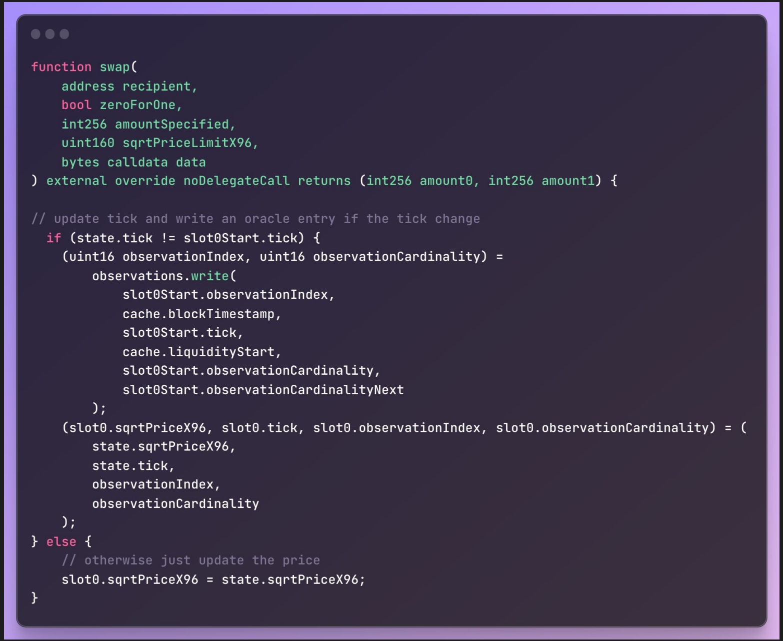
Fee in V3 comes in fixed fee tiers, 0.05%, 0.30%, and 1%. It's opinionated without customize possibility

Order Types

Impossible to implement different order types on-chain, like TWAP Order, Limit Order.

Gas Cost

To adjust settings in pools requires redeployment, which is gas consuming.



```
function swap(
    address recipient,
    bool zeroForOne,
    int256 amountSpecified,
    uint160 sqrtPriceLimitX96,
    bytes calldata data
) external override noDelegateCall returns (int256 amount0, int256 amount1) {

// update tick and write an oracle entry if the tick change
  if (state.tick != slot0Start.tick) {
    (uint16 observationIndex, uint16 observationCardinality) =
        observations.write(
            slot0Start.observationIndex,
            cache.blockTimestamp,
            slot0Start.tick,
            cache.liquidityStart,
            slot0Start.observationCardinality,
            slot0Start.observationCardinalityNext
        );
    (slot0.sqrtPriceX96, slot0.tick, slot0.observationIndex, slot0.observationCardinality) = (
        state.sqrtPriceX96,
        state.tick,
        observationIndex,
        observationCardinality
    );
} else {
    // otherwise just update the price
    slot0.sqrtPriceX96 = state.sqrtPriceX96;
}
```

HOW UNIV4

WAT WITH HOOKS?

Official White Paper Cases
Official EthCC Cases
EthCC Hookathon Cases

Kool Implementations

Lock and Flash Accounting
Transient Storage
Upgradable Hook? Hook Whitelist?

WAT WITH HOOOOOKS

Official Whitepaper Case

Cases from the official white paper

Official EthCC Case

Cases from the official talks in EthCC

EthCC Hookathon Case

Yeah, ArrakisFinance and Uniswap Foundation
hold a hackathon about hooks in Eth CC

Official White Paper

1. TWAMM (tee-wham) - Executing large orders over time through TWAMM
2. Onchain limit orders that fill at tick prices
3. Volatility-shifting dynamic fees
4. Mechanisms to internalize MEV for liquidity providers
5. Median, truncated, or other custom oracle implementations

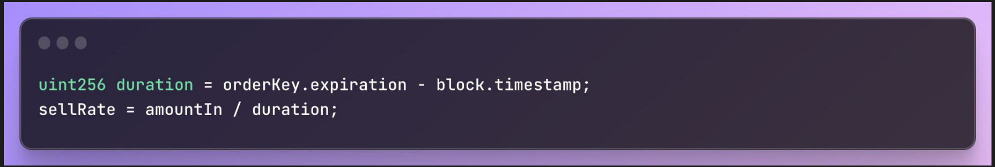
TWAMM

1. submitOrder()
2. executeOrder()
3. updateOrder()
4. claimTokens()

Submit Order

1. Calculate sellRate

Calculate the existence time of a order, and divided the swapping amount by the existence time. Thus get the swapping amount per second of a order

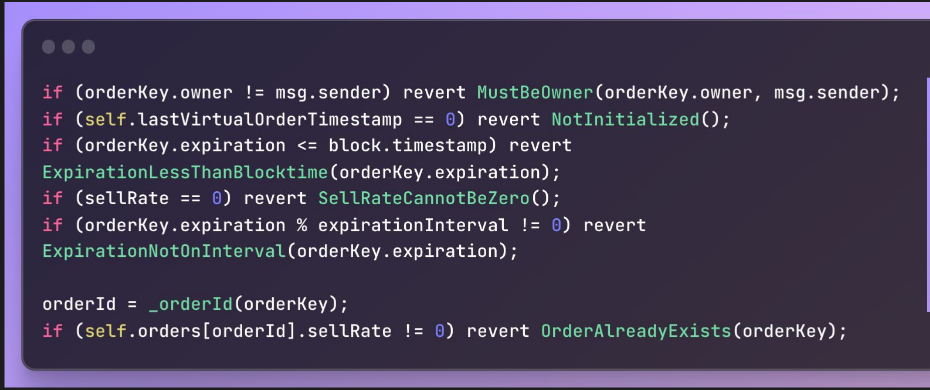


```
uint256 duration = orderKey.expiration - block.timestamp;
sellRate = amountIn / duration;
```

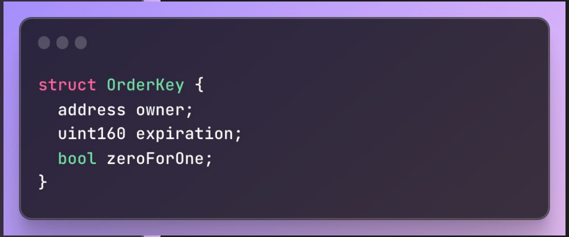
Submit Order

2. Variables Check

Check order owner, initialized, expiration time, and swap amount. And here a limitation, which is the expiration time must be at the expiration interval. Another limitation is that one user can only have one order on the same expiration date and the same input token. But this shouldn’t be a big issue, since user can update the order.



```
if (orderKey.owner != msg.sender) revert MustBeOwner(orderKey.owner, msg.sender);
if (self.lastVirtualOrderTimestamp == 0) revert NotInitialized();
if (orderKey.expiration <= block.timestamp) revert
ExpirationLessThanBlocktime(orderKey.expiration);
if (sellRate == 0) revert SellRateCannotBeZero();
if (orderKey.expiration % expirationInterval != 0) revert
ExpirationNotOnInterval(orderKey.expiration);

orderId = _orderId(orderKey);
if (self.orders[orderId].sellRate != 0) revert OrderAlreadyExists(orderKey);
```

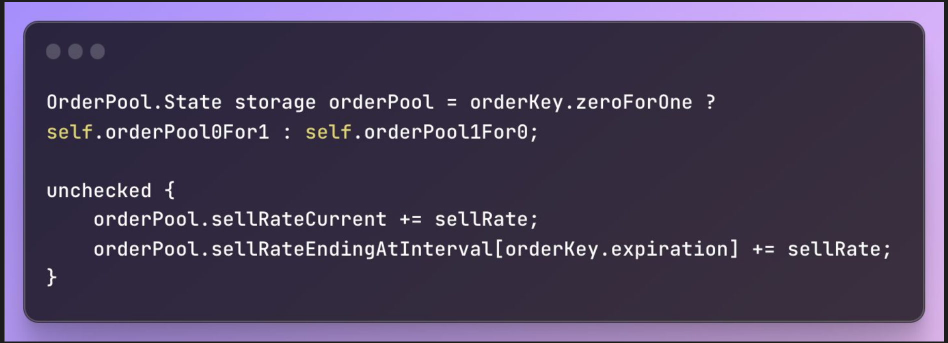


```
struct OrderKey {
  address owner;
  uint160 expiration;
  bool zeroForOne;
}
```

Submit Order

3. Update State

There are two variables being used to check if there are orders left to be executed. One is sellRate, the total amount of selling amount per sec, the second is sellRateEndingAtInterval, the total amount of selling amount per sec at certain timestamp.

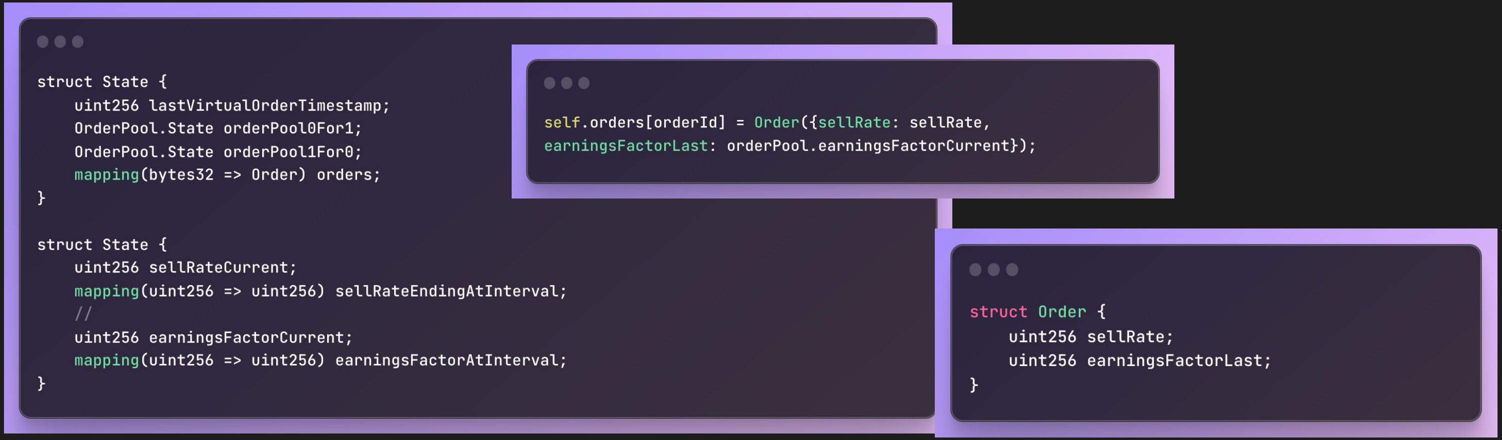


```
OrderPool.State storage orderPool = orderKey.zeroForOne ?
self.orderPool0For1 : self.orderPool1For0;

unchecked {
    orderPool.sellRateCurrent += sellRate;
    orderPool.sellRateEndingAtInterval[orderKey.expiration] += sellRate;
}
```

Submit Order

4. Store Order

Store the order into the system state. The state of the whole system is consist of lastVirtualOrderTimestamp (Last time the orders were executed), OrderPool State 1 (Pending swapping amount of token0ToToken1), OrderPool State 2 (Pending swapping amount of token1ToToken0), and Orders.



```
struct State {
    uint256 lastVirtualOrderTimestamp;
    OrderPool.State orderPool0For1;
    OrderPool.State orderPool1For0;
    mapping(bytes32 => Order) orders;
}

struct State {
    uint256 sellRateCurrent;
    mapping(uint256 => uint256) sellRateEndingAtInterval;
    //
    uint256 earningsFactorCurrent;
    mapping(uint256 => uint256) earningsFactorAtInterval;
}
```

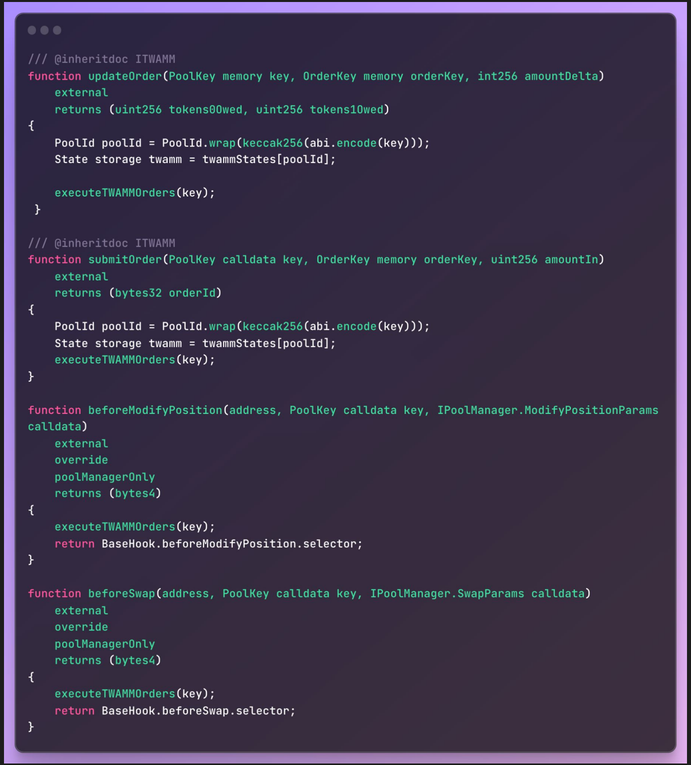
```
self.orders[orderId] = Order({sellRate: sellRate,
earningsFactorLast: orderPool.earningsFactorCurrent});
```

```
struct Order {
    uint256 sellRate;
    uint256 earningsFactorLast;
}
```

Execute Order

0. Execute Orders Everywhere

In the current design, executeOrders() is put in beforeModifyPosition, beforeSwap, submitOrder, and updateOrder. Every actions with the pool would require executing others order

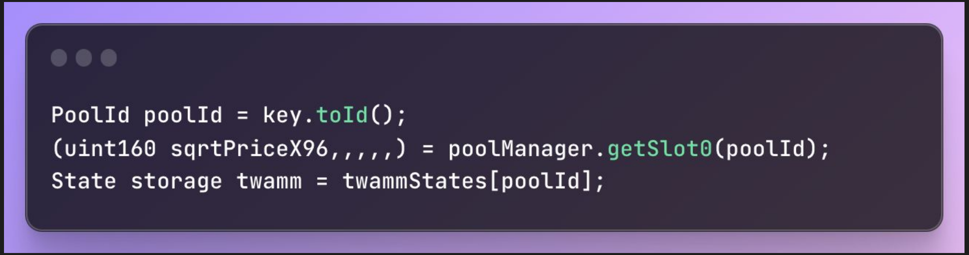


```
/// @inheritdoc ITWAMM
function updateOrder(PoolKey memory key, OrderKey memory orderKey, int256 amountDelta)
    external
    returns (uint256 tokens0Owed, uint256 tokens1Owed)
{
    PoolId poolId = PoolId.wrap(keccak256(abi.encode(key)));
    State storage twamm = twammStates[poolId];

    executeTWAMMOrders(key);
 }

/// @inheritdoc ITWAMM
function submitOrder(PoolKey calldata key, OrderKey memory orderKey, uint256 amountIn)
    external
    returns (bytes32 orderId)
{
    PoolId poolId = PoolId.wrap(keccak256(abi.encode(key)));
    State storage twamm = twammStates[poolId];
    executeTWAMMOrders(key);
}

function beforeModifyPosition(address, PoolKey calldata key, IPoolManager.ModifyPositionParams
calldata)
    external
    override
    poolManagerOnly
    returns (bytes4)
{
    executeTWAMMOrders(key);
    return BaseHook.beforeModifyPosition.selector;
}

function beforeSwap(address, PoolKey calldata key, IPoolManager.SwapParams calldata)
    external
    override
    poolManagerOnly
    returns (bytes4)
{
    executeTWAMMOrders(key);
    return BaseHook.beforeSwap.selector;
}
```

Execute Order

1. Get Pool Info

Before executing the order, retrieve the token price and the state of the whole system

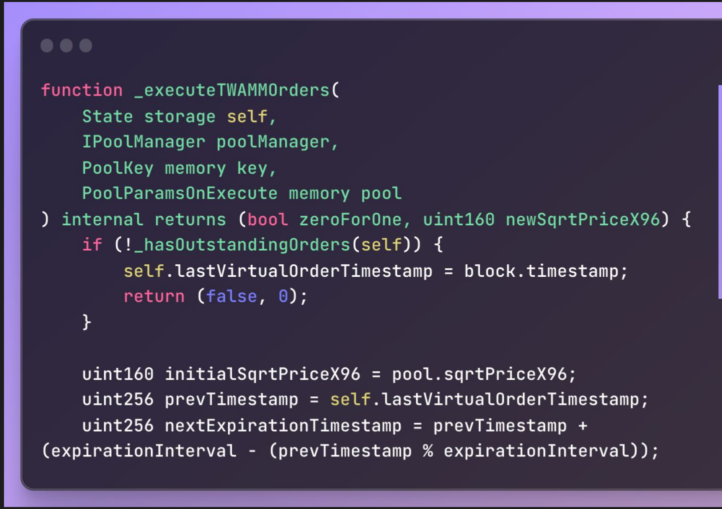


```
PoolId poolId = key.toId();
(uint160 sqrtPriceX96,,,,,) = poolManager.getSlot0(poolId);
State storage twamm = twammStates[poolId];
```

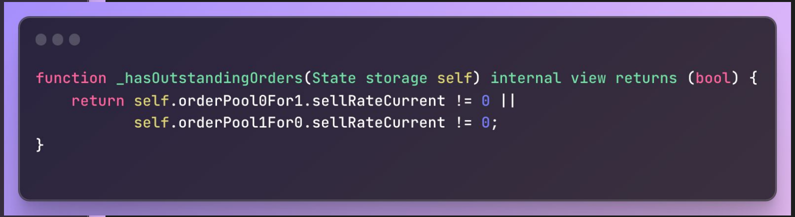
Execute Order

2. Variables Check

Check if there are outstanding orders. Then get the timestamp where the execution ends.



```
function _executeTWAMMOrders(
    State storage self,
    IPoolManager poolManager,
    PoolKey memory key,
    PoolParamsOnExecute memory pool
) internal returns (bool zeroForOne, uint160 newSqrtPriceX96) {
    if (!_hasOutstandingOrders(self)) {
        self.lastVirtualOrderTimestamp = block.timestamp;
        return (false, 0);
    }

    uint160 initialSqrtPriceX96 = pool.sqrtPriceX96;
    uint256 prevTimestamp = self.lastVirtualOrderTimestamp;
    uint256 nextExpirationTimestamp = prevTimestamp +
(expirationInterval - (prevTimestamp % expirationInterval));
```

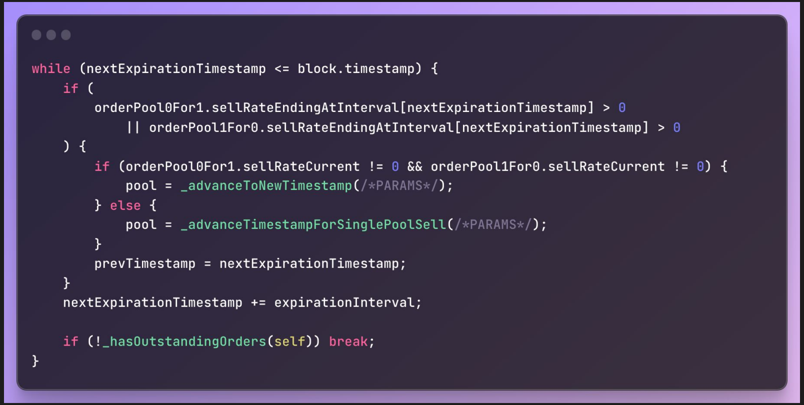


```
function _hasOutstandingOrders(State storage self) internal view returns (bool) {
    return self.orderPool0For1.sellRateCurrent != 0 ||
           self.orderPool1For0.sellRateCurrent != 0;
}
```

Execute Order

3. Execute Orders

Start the iteration from the nextExpirationTimestamp to the current timestamp. Use “sellRateEndingAtInterval” to decide if there are orders in the nextExpirationTimestamp. Then use “sellRateCurrent” to decide if there are orders in both pools or only in one pool

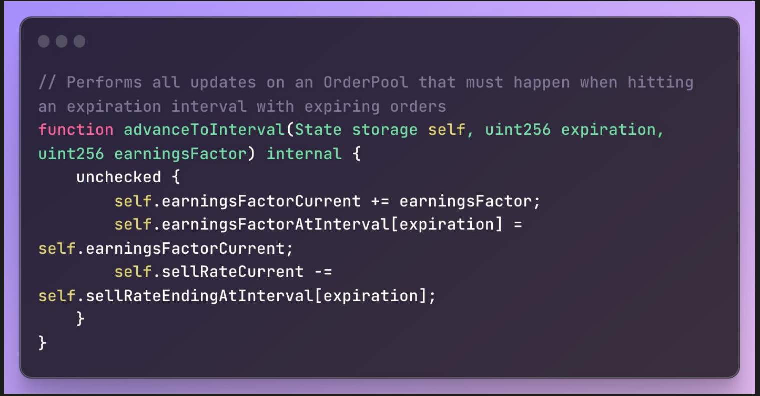


```
while (nextExpirationTimestamp <= block.timestamp) {
    if (
        orderPool0For1.sellRateEndingAtInterval[nextExpirationTimestamp] > 0
            || orderPool1For0.sellRateEndingAtInterval[nextExpirationTimestamp] > 0
    ) {
        if (orderPool0For1.sellRateCurrent != 0 && orderPool1For0.sellRateCurrent != 0) {
            pool = _advanceToNewTimestamp(/*PARAMS*/);
        } else {
            pool = _advanceTimestampForSinglePoolSell(/*PARAMS*/);
        }
        prevTimestamp = nextExpirationTimestamp;
    }
    nextExpirationTimestamp += expirationInterval;

    if (!_hasOutstandingOrders(self)) break;
}
```

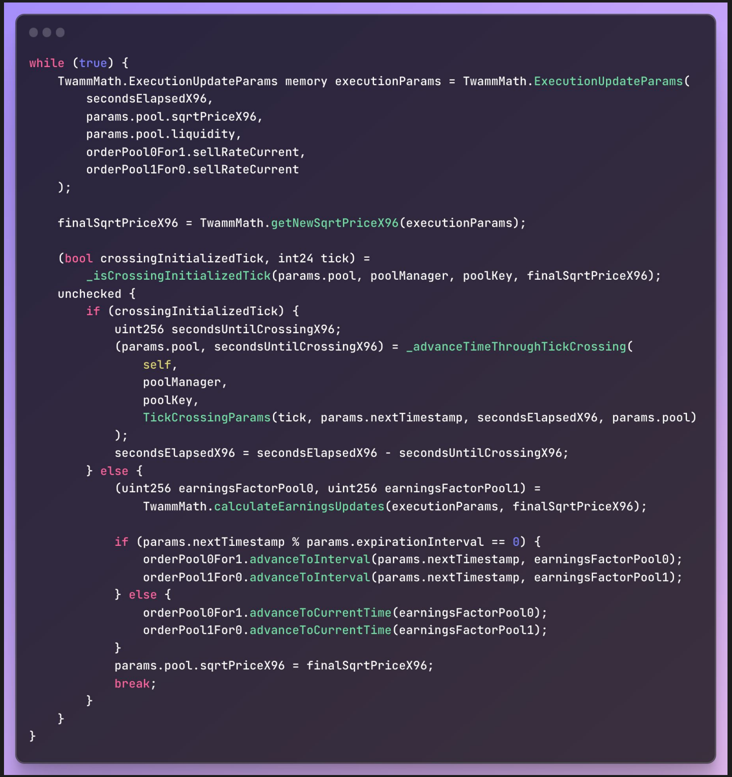
Execute Order

3. Execute Orders

Inside _advanceToNewTimestamp, and _advanceTimestampForSinglePoolSell. They continue to update the price till no crossing ticks. And call advanceToInterval or advanceToCurrentTime to adjust the sellRate



```
// Performs all updates on an OrderPool that must happen when hitting
an expiration interval with expiring orders
function advanceToInterval(State storage self, uint256 expiration,
uint256 earningsFactor) internal {
    unchecked {
        self.earningsFactorCurrent += earningsFactor;
        self.earningsFactorAtInterval[expiration] =
self.earningsFactorCurrent;
        self.sellRateCurrent -=
self.sellRateEndingAtInterval[expiration];
    }
}
```



```
while (true) {
    TwammMath.ExecutionUpdateParams memory executionParams = TwammMath.ExecutionUpdateParams(
        secondsElapsedX96,
        params.pool.sqrtPriceX96,
        params.pool.liquidity,
        orderPool0For1.sellRateCurrent,
        orderPool1For0.sellRateCurrent
    );

    finalSqrtPriceX96 = TwammMath.getNewSqrtPriceX96(executionParams);

    (bool crossingInitializedTick, int24 tick) =
        _isCrossingInitializedTick(params.pool, poolManager, poolKey, finalSqrtPriceX96);
    unchecked {
        if (crossingInitializedTick) {
            uint256 secondsUntilCrossingX96;
            (params.pool, secondsUntilCrossingX96) = _advanceTimeThroughTickCrossing(
                self,
                poolManager,
                poolKey,
                TickCrossingParams(tick, params.nextTimestamp, secondsElapsedX96, params.pool)
            );
            secondsElapsedX96 = secondsElapsedX96 - secondsUntilCrossingX96;
        } else {
            (uint256 earningsFactorPool0, uint256 earningsFactorPool1) =
                TwammMath.calculateEarningsUpdates(executionParams, finalSqrtPriceX96);

            if (params.nextTimestamp % params.expirationInterval == 0) {
                orderPool0For1.advanceToInterval(params.nextTimestamp, earningsFactorPool0);
                orderPool1For0.advanceToInterval(params.nextTimestamp, earningsFactorPool1);
            } else {
                orderPool0For1.advanceToCurrentTime(earningsFactorPool0);
                orderPool1For0.advanceToCurrentTime(earningsFactorPool1);
            }
            params.pool.sqrtPriceX96 = finalSqrtPriceX96;
            break;
        }
    }
}
```

Limitations

1. Unique Order by Owner, Expiration, Side
2. Order Expiration Interval is predefined

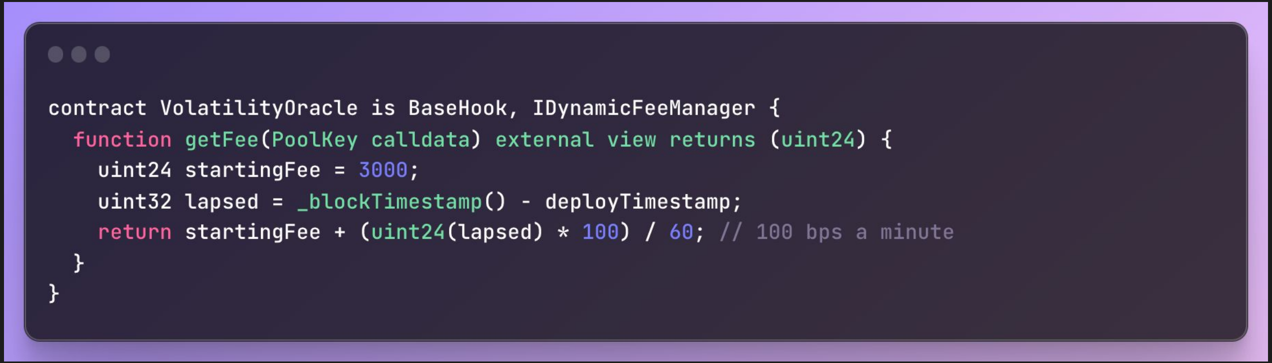
Official EthCC Talks

1. Improvements
   a. No op hook storage
   b. Singleton delta from hooks
   c. Arbitrary hook data passing
2. Cases
   a. Dynamic Fees via Auction
   b. Enforced V2 style full range LP
   c. Limit Order
   d. Arithmetic Oracle, Geomean Oracle
   e. Yield Bearing Hook, Fee Bearing Hook (Not sure what this means)
   f. Dollar Cost Averages (Not sure what this means)

Simple Case

1. Dynamic Fees

Inherit IDynamicFeeManager then write your own getFee() function

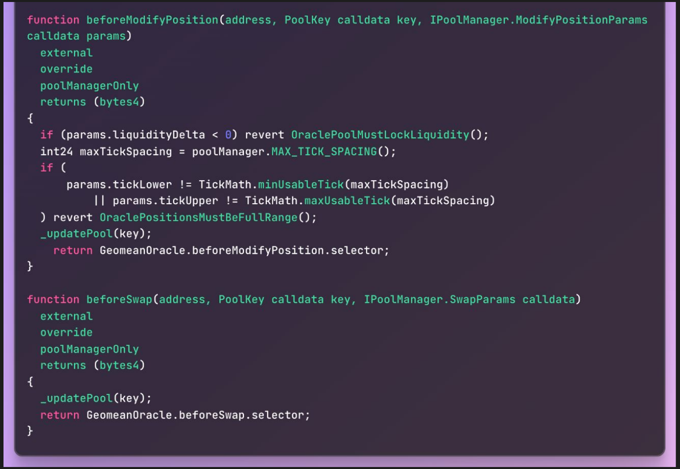


```
contract VolatilityOracle is BaseHook, IDynamicFeeManager {
  function getFee(PoolKey calldata) external view returns (uint24) {
    uint24 startingFee = 3000;
    uint32 lapsed = _blockTimestamp() - deployTimestamp;
    return startingFee + (uint24(lapsed) * 100) / 60; // 100 bps a minute
  }
}
```

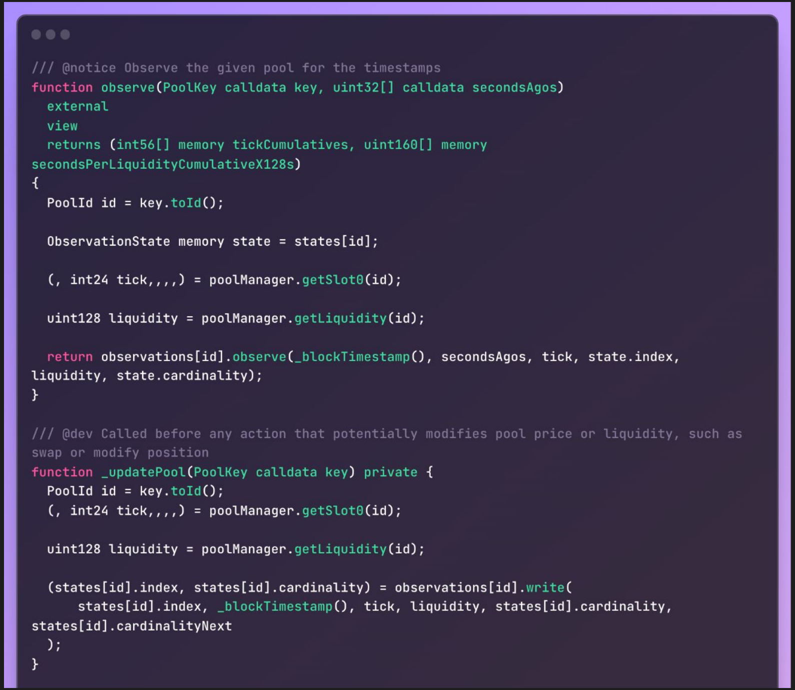
Simple Case

2. Customize Oracle

Have a observe function and decide how you want to return the price. And update the parameters in beforeModifyPosition and beforeSwap



```
function beforeModifyPosition(address, PoolKey calldata key, IPoolManager.ModifyPositionParams
calldata params)
  external
  override
  poolManagerOnly
  returns (bytes4)
{
  if (params.liquidityDelta < 0) revert OraclePoolMustLockLiquidity();
  int24 maxTickSpacing = poolManager.MAX_TICK_SPACING();
  if (
      params.tickLower != TickMath.minUsableTick(maxTickSpacing)
          || params.tickUpper != TickMath.maxUsableTick(maxTickSpacing)
  ) revert OraclePositionsMustBeFullRange();
  _updatePool(key);
    return GeomeanOracle.beforeModifyPosition.selector;
}

function beforeSwap(address, PoolKey calldata key, IPoolManager.SwapParams calldata)
  external
  override
  poolManagerOnly
  returns (bytes4)
{
  _updatePool(key);
  return GeomeanOracle.beforeSwap.selector;
}
```

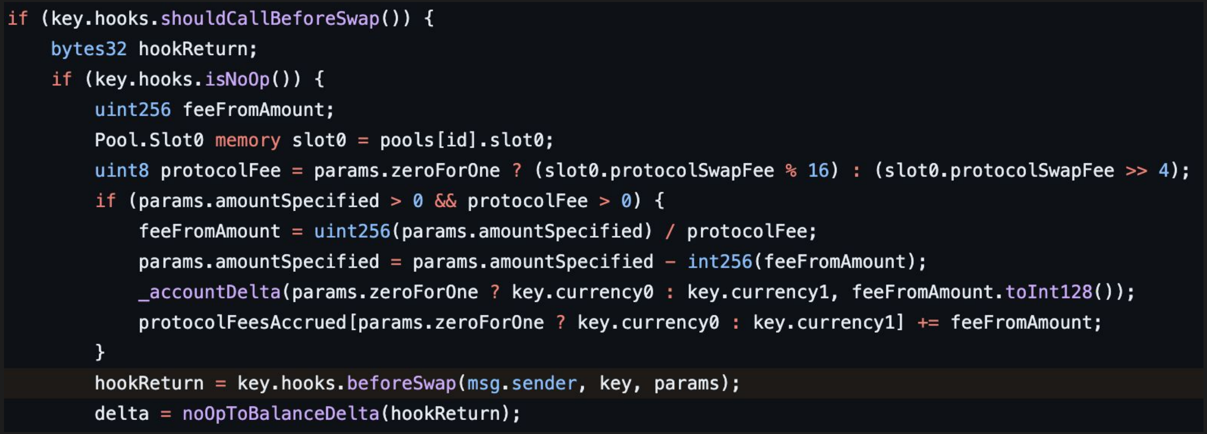


```
/// @notice Observe the given pool for the timestamps
function observe(PoolKey calldata key, uint32[] calldata secondsAgos)
  external
  view
  returns (int56[] memory tickCumulatives, uint160[] memory
secondsPerLiquidityCumulativeX128s)
{
  PoolId id = key.toId();

  ObservationState memory state = states[id];

  (, int24 tick,,,,) = poolManager.getSlot0(id);

  uint128 liquidity = poolManager.getLiquidity(id);

  return observations[id].observe(_blockTimestamp(), secondsAgos, tick, state.index,
liquidity, state.cardinality);
}

/// @dev Called before any action that potentially modifies pool price or liquidity, such as
swap or modify position
function _updatePool(PoolKey calldata key) private {
  PoolId id = key.toId();
  (, int24 tick,,,,) = poolManager.getSlot0(id);

  uint128 liquidity = poolManager.getLiquidity(id);

  (states[id].index, states[id].cardinality) = observations[id].write(
      states[id].index, _blockTimestamp(), tick, liquidity, states[id].cardinality,
states[id].cardinalityNext
  );
}
```

No op hook storage

Potential: Non Constant Product AMM

Add a no op state, and forget about Uniswap.
Calculate the price, delta whatever you want



```
if (key.hooks.shouldCallBeforeSwap()) {
    bytes32 hookReturn;
    if (key.hooks.isNoOp()) {
        uint256 feeFromAmount;
        Pool.Slot0 memory slot0 = pools[id].slot0;
        uint8 protocolFee = params.zeroForOne ? (slot0.protocolSwapFee % 16) : (slot0.protocolSwapFee >> 4);
        if (params.amountSpecified > 0 && protocolFee > 0) {
            feeFromAmount = uint256(params.amountSpecified) / protocolFee;
            params.amountSpecified = params.amountSpecified - int256(feeFromAmount);
            _accountDelta(params.zeroForOne ? key.currency0 : key.currency1, feeFromAmount.toInt128());
            protocolFeesAccrued[params.zeroForOne ? key.currency0 : key.currency1] += feeFromAmount;
        }
        hookReturn = key.hooks.beforeSwap(msg.sender, key, params);
        delta = noOpToBalanceDelta(hookReturn);
```

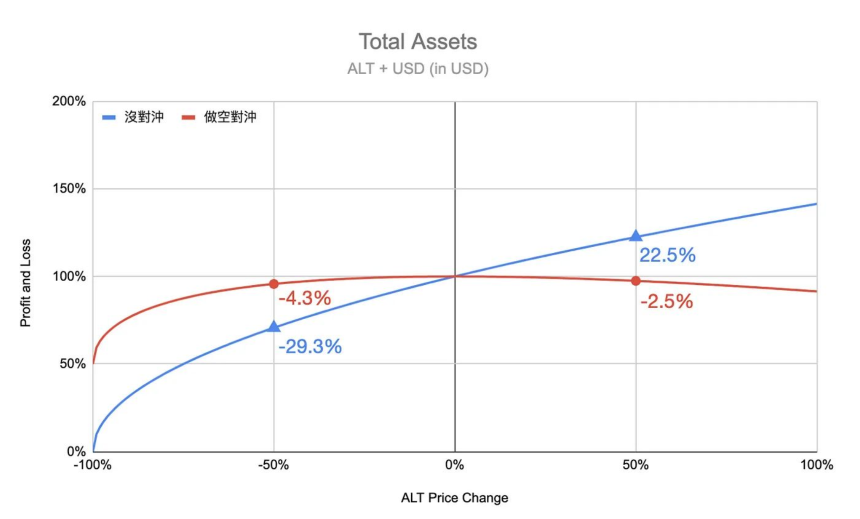
EthCC Hookathon

1. Delta Hedge LP
2. Median Oracle
3. Trader Hedge Swap Tool
4. Trading Day with Market Close and Open

Delta Hedge LP

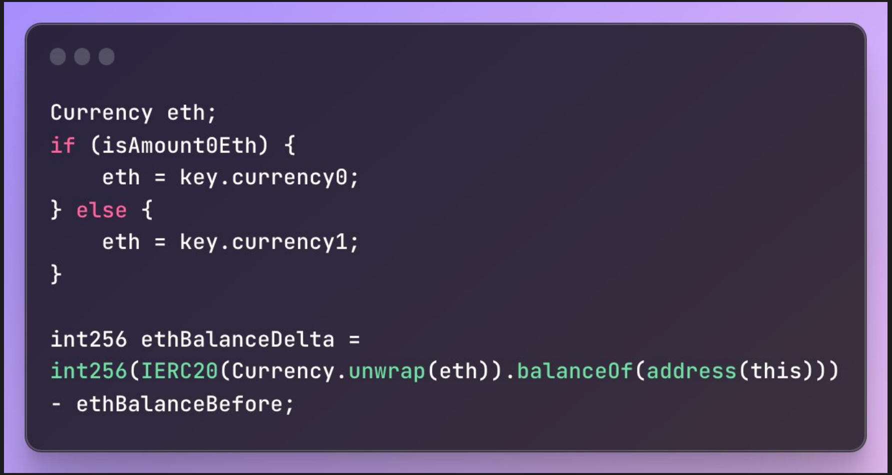
0. What Is Delta Hedge

Hedging delta risk, the risk of the total value of the pool influenced by the price change in underlying asset.
Hedging the impermanent loss.



Delta Hedge LP

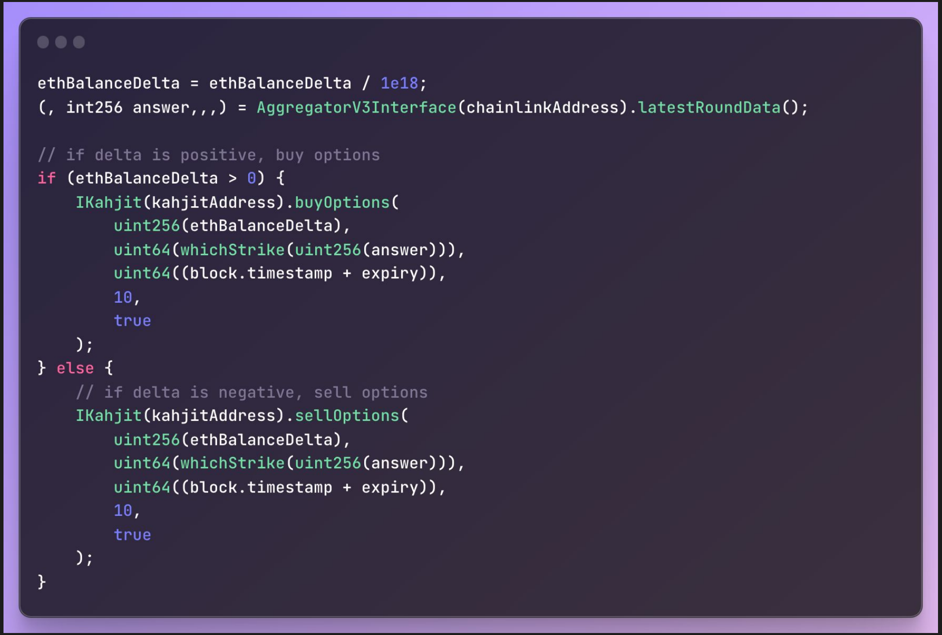
1. Calculate delta



```
Currency eth;
if (isAmount0Eth) {
    eth = key.currency0;
} else {
    eth = key.currency1;
}

int256 ethBalanceDelta =
int256(IERC20(Currency.unwrap(eth)).balanceOf(address(this)))
- ethBalanceBefore;
```

Delta Hedge LP

2. Buy Option from Lyra



```
ethBalanceDelta = ethBalanceDelta / 1e18;
(, int256 answer,,,) = AggregatorV3Interface(chainlinkAddress).latestRoundData();

// if delta is positive, buy options
if (ethBalanceDelta > 0) {
    IKahjit(kahjitAddress).buyOptions(
        uint256(ethBalanceDelta),
        uint64(whichStrike(uint256(answer))),
        uint64((block.timestamp + expiry)),
        10,
        true
    );
} else {
    // if delta is negative, sell options
    IKahjit(kahjitAddress).sellOptions(
        uint256(ethBalanceDelta),
        uint64(whichStrike(uint256(answer))),
        uint64((block.timestamp + expiry)),
        10,
        true
    );
}
```

Kool Implementations

Lock and Flash Accounting

One of the core innovations in their white paper

Transient Storage

The key EIP that makes this upgrade possible

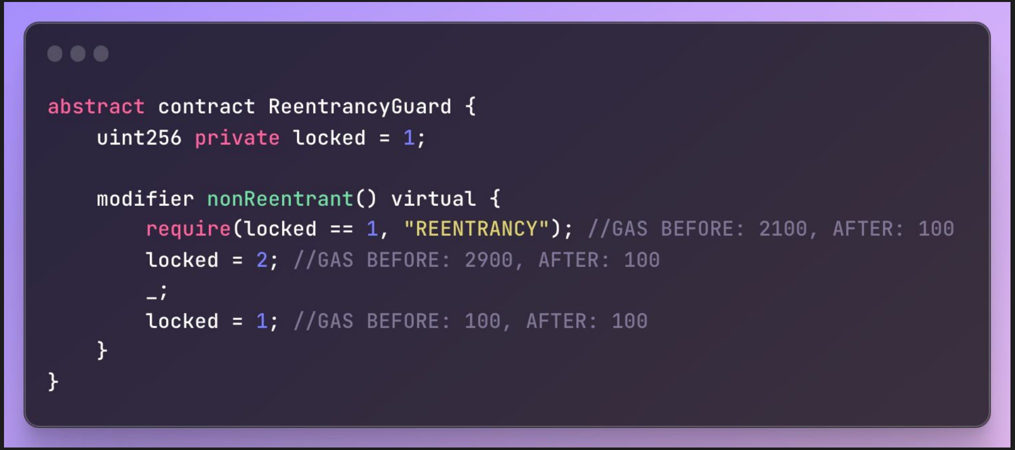
Security: Upgradable / Whitelist

Some questions that occur to me

Transient Storage

EIP-1153

Expected in Cancun Fork, it allows *storage* variable to exist only in transaction. A memory-liked persistent storage space, which is cheap and clean.



```
abstract contract ReentrancyGuard {
    uint256 private locked = 1;

    modifier nonReentrant() virtual {
        require(locked == 1, "REENTRANCY"); //GAS BEFORE: 2100, AFTER: 100
        locked = 2; //GAS BEFORE: 2900, AFTER: 100
        _;
        locked = 1; //GAS BEFORE: 100, AFTER: 100
    }
}
```

@garythung

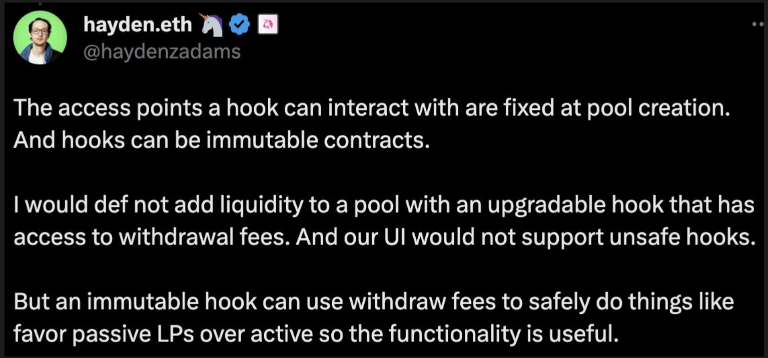
Security Issue

Is the hook upgradable?

Yes, the hook can be upgradable. Which gives more customizability to the hooks but also increases the security risks.

What is the solution?

Censoring the hooks through UI. In their EthCC talk, whitelist is mentioned.



hayden.eth 🦄
@haydenzadams

The access points a hook can interact with are fixed at pool creation. And hooks can be immutable contracts.

I would def not add liquidity to a pool with an upgradable hook that has access to withdrawal fees. And our UI would not support unsafe hooks.

But an immutable hook can use withdraw fees to safely do things like favor passive LPs over active so the functionality is useful.

RESOURCES

If I can only recommend one source

- https://github.com/fewwwww/awesome-uniswap-hooks#-tools

RESOURCES

Related

- Our Vision for Uniswap v4
- Alice Henshaw & Sara Reynolds - Introducing Uniswap v4
- Uniswap V4 ANNOUNCED By Founder Hayden Adams
- https://www.paradigm.xyz/2021/07/twamm#summary
- * Austin Adams - The Evolution of On-chain Trading
- Common Errors Uniswap Doc
- Token Integration Issues
- Callback Function
- Paradigm TWAMM
- Hookathon Delta Hedge
- No Ops Hook

Non-Related

- Hayden Adams - Onchain trading
- Xin Wan - Hyper Fragmented Liquidity, Fully Adversarial Mempool, Wat Do?

Tweets

- https://twitter.com/garythung/status/1668716556489203713
- https://twitter.com/hensha256/status/1680708957558456321
- https://twitter.com/UniswapFND/status/1682344172995457027
- https://twitter.com/haydenzadams/status/1668723133233127426
- https://twitter.com/Bob_Baxley/status/1669381297330855937
- https://twitter.com/LearnWeb3DAO/status/1682780499335364609
- https://twitter.com/UniswapFND/status/1683983199872122881