Introduction to DeFi

11 / 22

Table of Contents

Currency Exchange in Traditional Finance

Scenario

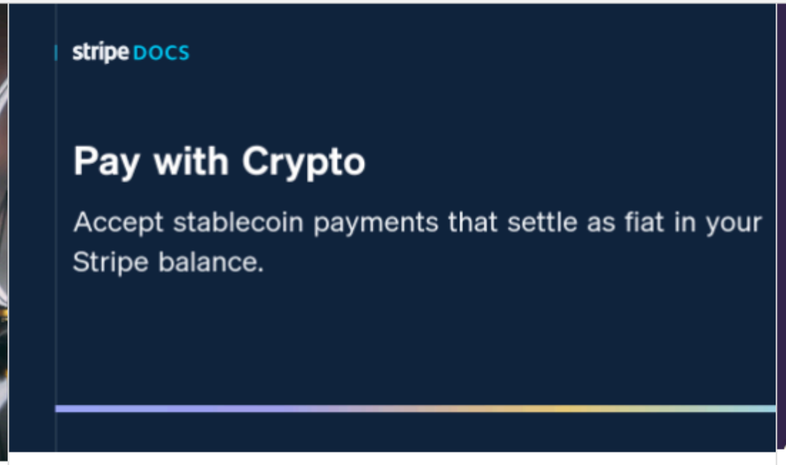
Why do we need on-chain exchange?

- If we launch meme tokens on-chain, could we use meme coins for daily transactions?

Grab's Singapore Users Can Now Use Crypto to Payments

The Grab super app is now making available the option of pa cryptocurrencies, The Straits Times reported Tuesday citing a Grab users.

CoinDesk / Mar 19, 2024



Pay with Crypto

Pay with Crypto lets you accept stablecoin payments that settle as fi your Stripe balance. You can accept USDC payments on Ethereum, Solana, and Polygon without the complexity of holding or converting.

stripe.com

MetaMask, Mastercard and Baanx Unveil Revolutionary Way to Pay with Crypto

MetaMask launches the world's first Mastercard debit card that enables instant spending directly from your self-custody wallet.

Consensys

Scenario

Why do we need on-chain exchange?

- We currently cannot pay with crypto; what are the alternatives?
- There might be two methods for you:
  - Centralized Exchange: Binance, Coinbase, Kraken, ...
  - Decentralized Exchange: Uniswap, PancakeSwap, Balancer, ...

Solution 1 - Centralized Exchange

I might deposit my USDT to **CEX**, and exchange USDT for ETH.

- Where can I exchange my USDT for ETH?
- How much ETH will I receive for exchanging 1000 USDT?
- Do exchange rates for USDT to ETH vary across platforms?
- Which entity regulates cryptocurrency exchange rates?
- What are the operating hours of these exchange services?

Solution 1 - Centralized Exchange

I might deposit my USDT to **CEX**, and exchange USDT for ETH.

- Advantages:
  - Quick transactions
  - Low fees per operation
  - User-friendly
- Disadvantages:
  - Assets are held by a centralized entity
  - Risk of collapse (e.g., FTX case)
  - Ongoing machine maintenance required

Solution 2 - Decentralized Exchange

I might deposit my USDT to **DEX**, and exchange USDT for ETH.

- Where can I exchange my USDT for ETH?
- How much ETH will I receive for exchanging 1000 USDT?
- Do exchange rates for USDT to ETH vary across platforms?
- Which entity regulates cryptocurrency exchange rates?
- What are the operating hours of these exchange services?

Solution 2 - Decentralized Exchange

I might deposit my USDT to **DEX**, and exchange USDT for ETH.

- Advantages:
  - Control Over Assets: Full control without relying on a central authority.
  - Transparency: Blockchain records ensure trust.
  - Privacy: No KYC required, maintaining anonymity.
- Disadvantages:
  - Complex Interface: Requires blockchain and wallet knowledge.
  - Limited Liquidity: Often lower than centralized exchanges.
  - Slippage Risk: Prone to price volatility and slippage.

Summary

Some users prefer on-chain exchanges without relying on centralized agencies.

- We launched our own token in HW1, can we list and exchange LiaoToken on a CEX?
- What if I need an atomic operation to exchange my token, like in flash loan scenarios?
- What if I want to interact with a DeFi protocol, or if the protocol itself requires token exchange?
- What if...?

Categories of AMMs in Cryptocurrency

DEX Solutions

Why don't we implement an order book mechanism directly on-chain?

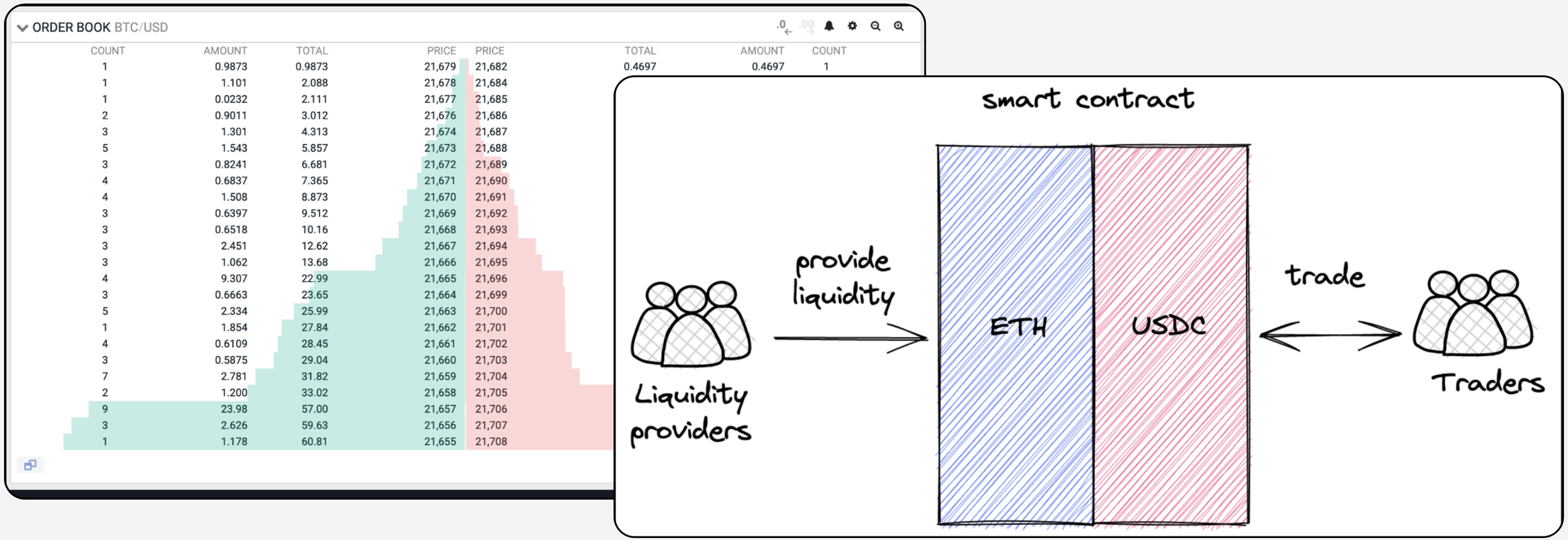
COUNT	AMOUNT	TOTAL	PRICE	PRICE	TOTAL	AMOUNT	COUNT
1	0.9873	0.9873	21,679	21,682	0.4697	0.4697	1
1	1.101	2.088	21,678	21,684	1.176	0.7071	2
1	0.0232	2.111	21,677	21,685	1.955	0.7784	2
2	0.9011	3.012	21,676	21,686	2.070	0.1152	1
3	1.301	4.313	21,674	21,687	2.301	0.2305	1
5	1.543	5.857	21,673	21,688	2.762	0.4610	1
3	0.8241	6.681	21,672	21,689	5.430	2.668	3
4	0.6837	7.365	21,671	21,690	6.591	1.161	3
4	1.508	8.873	21,670	21,691	7.355	0.7642	3
3	0.6397	9.512	21,669	21,692	7.896	0.5404	3
3	0.6518	10.16	21,668	21,693	7.993	0.0976	1
3	2.451	12.62	21,667	21,694	10.60	2.610	3
3	1.062	13.68	21,666	21,695	11.53	0.9287	3
4	9.307	22.99	21,665	21,696	14.72	3.190	3
3	0.6663	23.65	21,664	21,699	15.00	0.2746	3
5	2.334	25.99	21,663	21,700	15.00	0.0020	1
1	1.854	27.84	21,662	21,701	15.10	0.1010	2
4	0.6109	28.45	21,661	21,702	15.10	0.0030	2
3	0.5875	29.04	21,660	21,703	16.01	0.9020	5
7	2.781	31.82	21,659	21,704	17.25	1.240	5
2	1.200	33.02	21,658	21,705	26.81	9.560	4
9	23.98	57.00	21,657	21,706	28.07	1.259	5
3	2.626	59.63	21,656	21,707	28.07	0.0044	2
1	1.178	60.81	21,655	21,708	28.29	0.2178	2

- What data structures would need to be maintained?
- This could consume excessive gas on-chain.
- Imagine implement order book on Ethereum mainnet.
- It may not be a feasible idea at this point.

Reference: Uniswap V3 Book

DEX Solutions

Why don't we implement an order book mechanism directly on-chain?



Reference: Uniswap V3 Book

AMM

AMM (Automated Market Makers): Use algorithms to manage liquidity in an automated way, operating similarly to traditional market makers.

Uniswap

Balancer

Curve

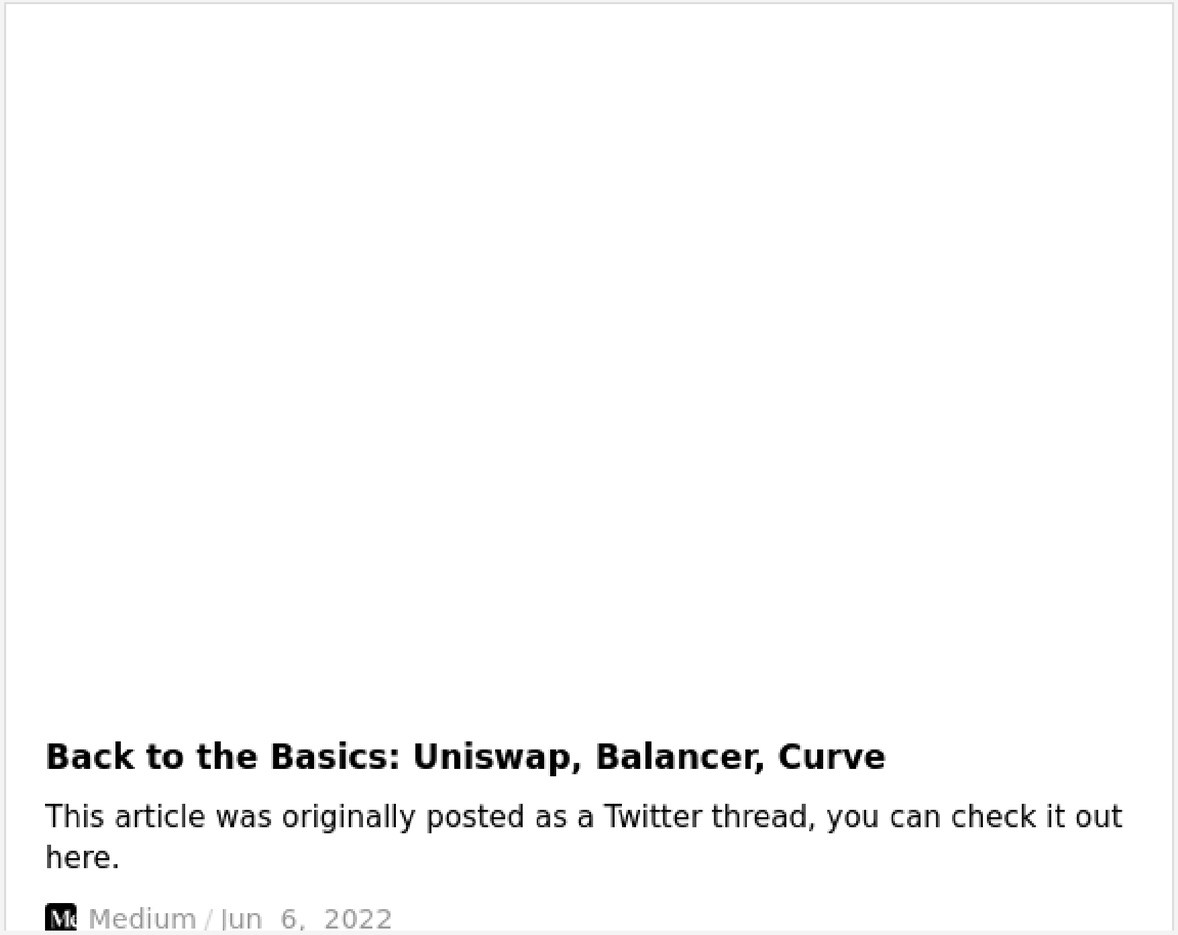
$x * y = k$

$$\prod_{i=1}^{n} x_i^{w_i} = k$$

$$An^n \sum x_i + D = ADn^n + \frac{D^{n+1}}{n^n \prod x_i}.$$

The good news is, we will only teach Uniswap
The bad news is, we will only teach Uniswap

The Evolution of AMMs



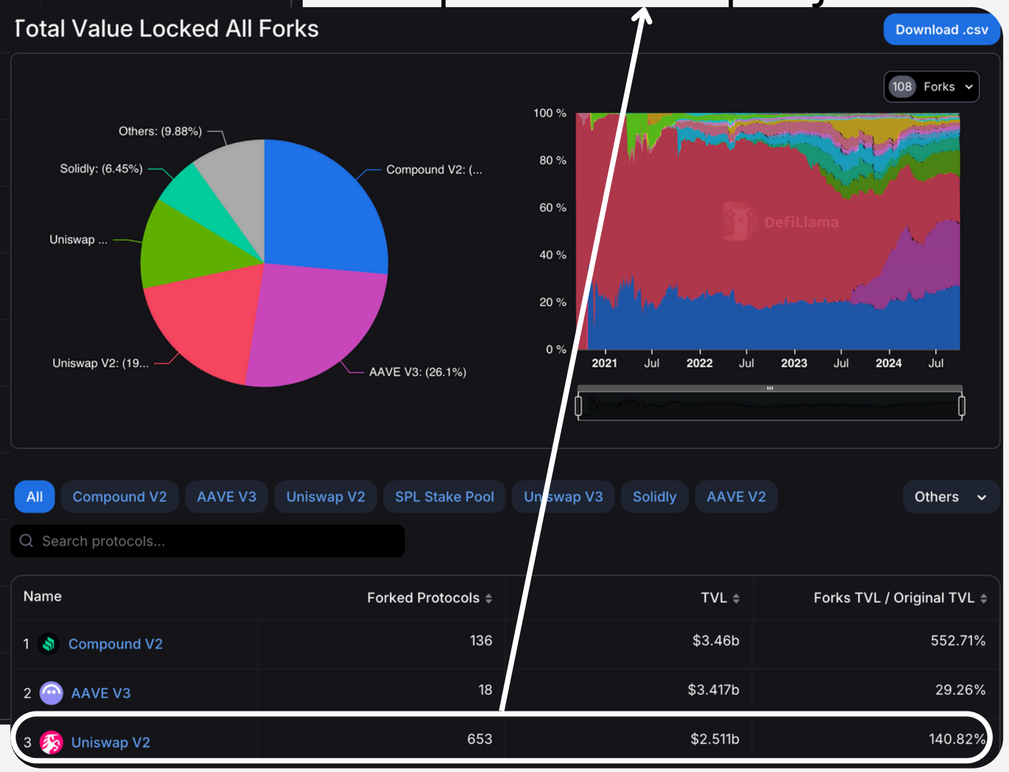
Uniswap

Why we start learning AMM with Uniswap?

Chain	Dexes	Basis Trading	Yield	Farm
1 Ethereum	Uniswap V3	Ethena	Pendle	Zircuit Staking
2 Tron	SUNSwap V1			FlatCoin DEX
3 Solana	Raydium		Sanctum Reserve	
4 BSC	PancakeSwap AMM	BounceBit Easy	Coinwind	
5 Arbitrum	Uniswap V3		Pendle	
6 Base	Aerodrome V1		Toros	
7 Avalanche	Joe V2.2		CIAN Automation	
8 Sui	Cetus AMM		Sparkling Finance	
9 Polygon	Uniswap V3		Aura	
10 Scroll	Ambient		Tranchess Yield	
11 Blast	Thruster V3		Ring Few	
12 Hyperliquid	Hyperliquid Spot			
13 Optimism	Velodrome V2		Toros	

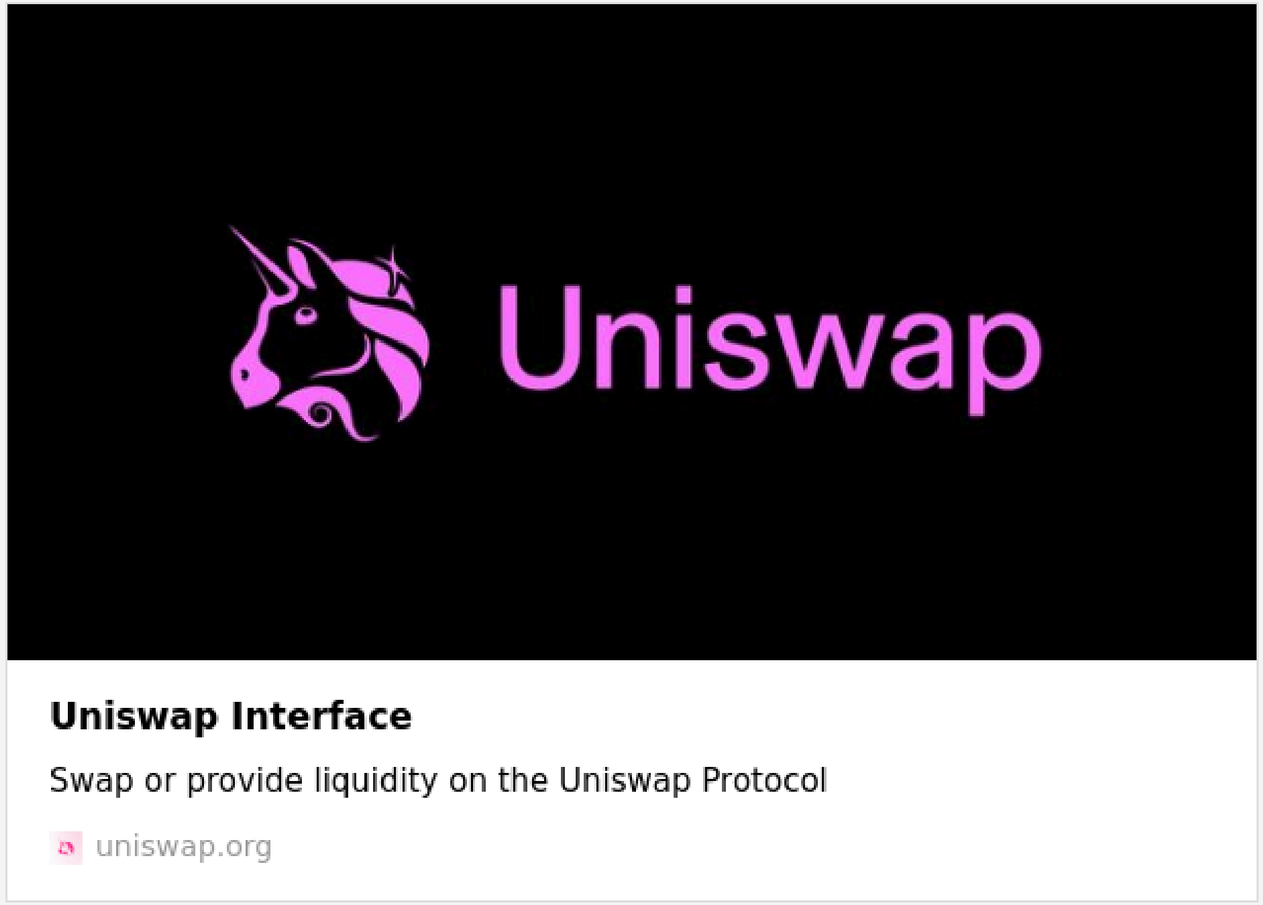
Use similar algorithm to Uniswap

Top 3 forked project



Uniswap

Before diving into the underlying mathematics and mechanisms, let's explore how to interact with Uniswap on their website.

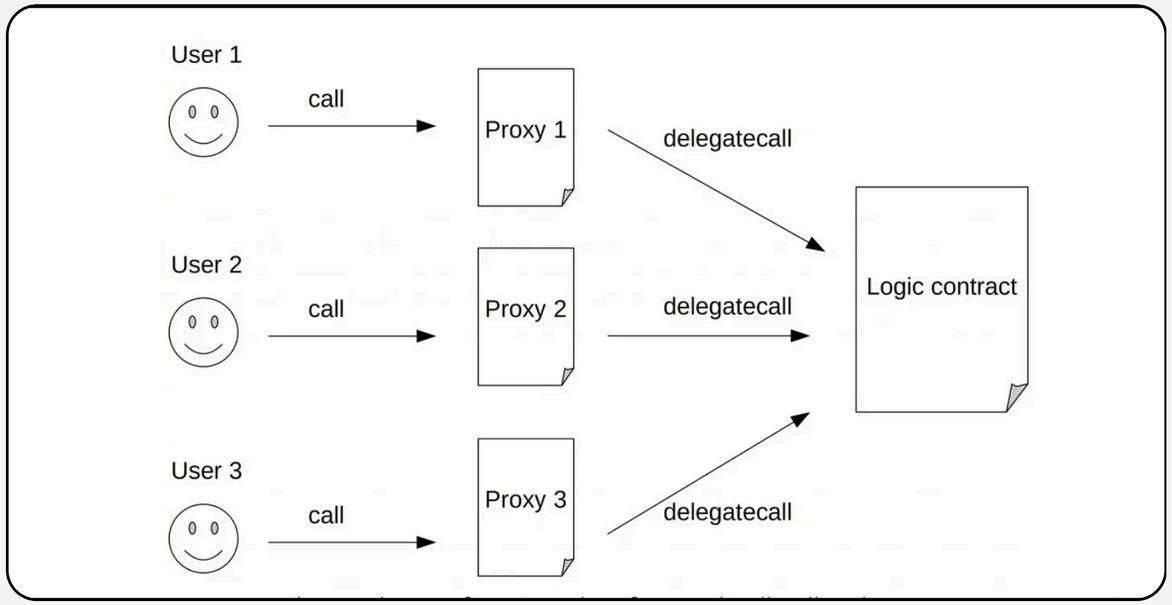


Uniswap:
Interaction & Operational Mechanics

Uniswap V1

2018: Uniswap launch their first product.

- There is a factory pattern that allows developer to create new pool
- It follows the minimal proxy pattern, where the same implementation is reused by creating a minimal proxy that delegates to the main implementation.



EIP-1167

- Reuse the implementation contract.
- Each proxy is an instance of the pool
- It is not upgradeable proxy as we do not need to upgrade our contract.

Uniswap V1

Currently, no one use the version 1.

- The pool only allows ETH <> ERC-20 pair.
- There is **reentrancy** issue in the contract when interacting with ERC-777.

$$\text{getInputPrice}(\text{Ts}, \text{Tr}, \text{ETHr}) = \frac{\text{Ts} * 997 * \text{ETHr}}{\text{Tr} * 1000 + \text{Ts} * 997}$$

where
Ts: *amount of tokens being sold by caller*
Tr: *current reserve of tokens*
ETHr: *current reserve of Ether*

Transfer the ETH before updating the reserves

Uniswap V2

2020: The version 2 was deployed and supported ERC-20 <> ERC-20 pair.

- Support more complex routing strategies across multiple pools.
  - Token A - Token B - Token C - Token D - ...
- Support flash swap, a service that is similar to flash loan
- Implement Time Weighted Average Price (TWAP) oracle.
- Algorithm:

$$x * y = k$$

Uniswap V2

Algorithm: x * y = k, where k is a constant

- x: the amount of token0 in the pair
- y: the amount of token1 in the pair

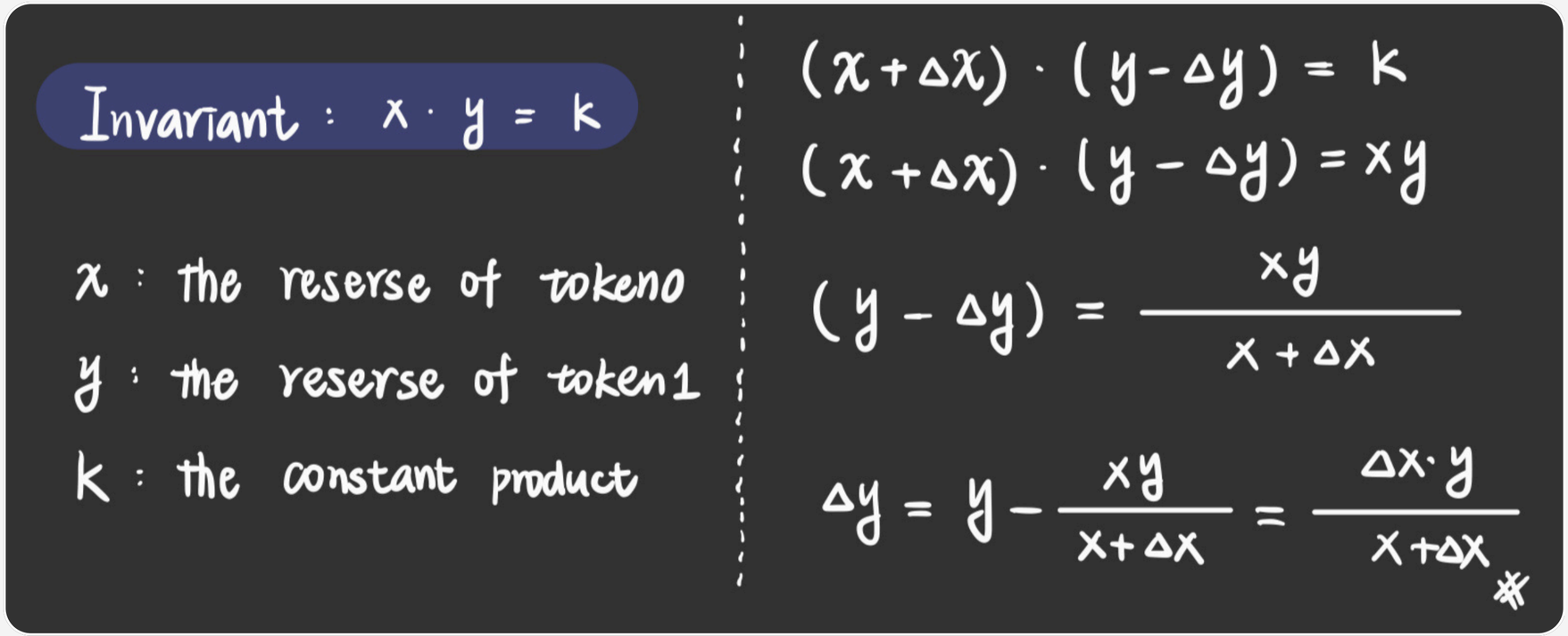
Assumption: There is no swap fee.

Case 1: We want to swap x' token0 for token1, how much will I get?

Case 2: We want to swap y' token1 for token0, how much will I get?

Swap Without Fee

Case 1: We want to swap x' token0 for token1, how much will I get?



Uniswap V2

Algorithm: $x * y \leq k$, where k is a constant

- x: the amount of token0 in the pair
- y: the amount of token1 in the pair

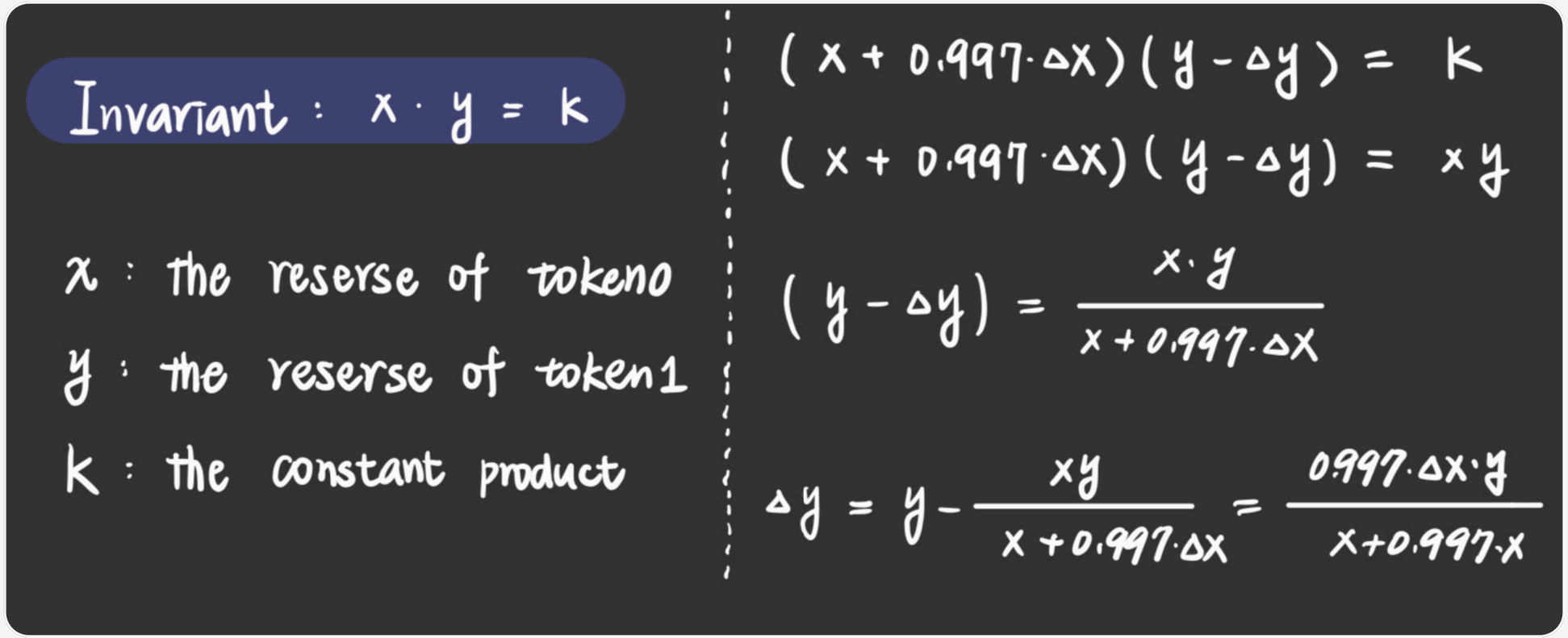
Assumption: There is **0.3%** swap fee.

Case 1: We want to swap x' token0 for token1, how much will I get?

Case 2: We want to swap y' token1 for token0, how much will I get?

Swap With Fee

Case 1: We want to swap x' token0 for token1, how much will I get?



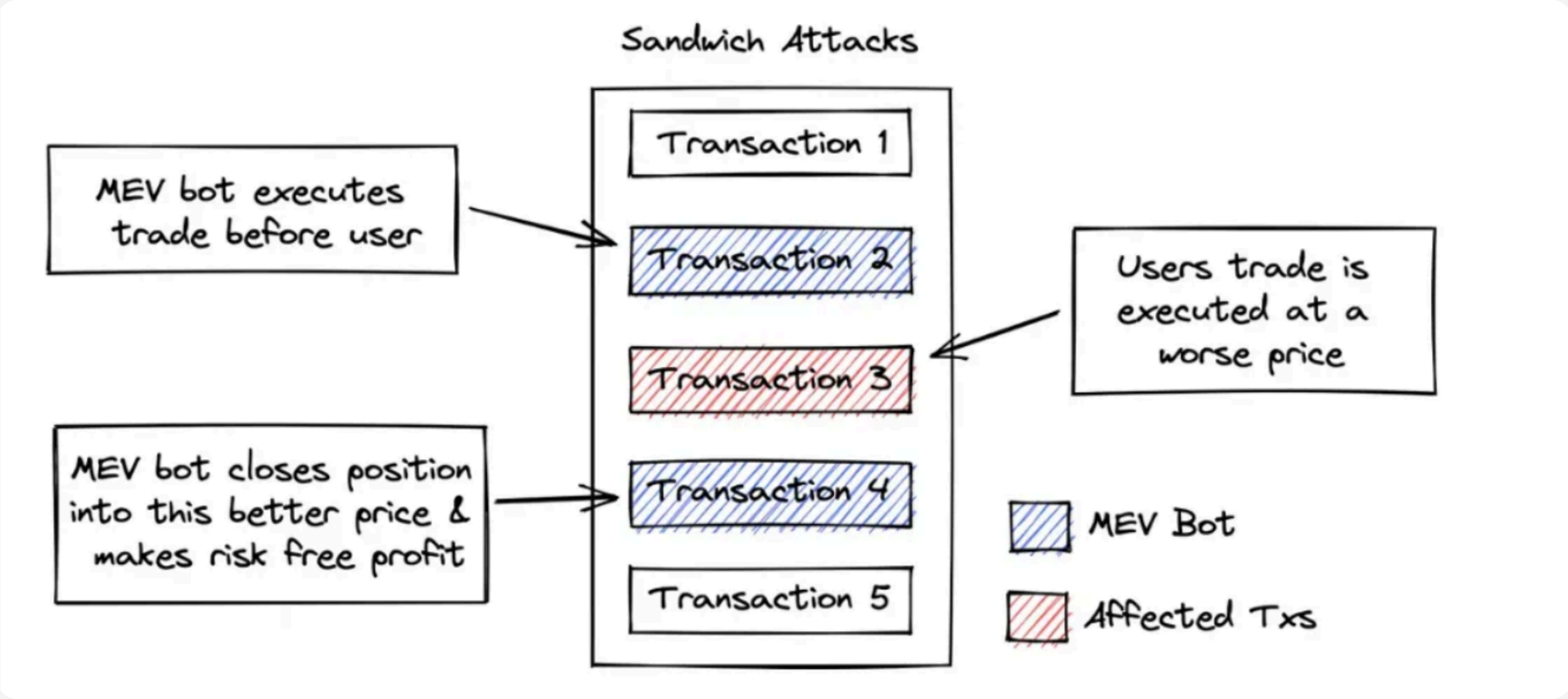
Slippage Issue

Can I always get the amount that I want

- Slippage: The difference between the expected price and the actual price.
- Imagine calculating the output amount off-chain and providing 3400 USDC to swap for 1 ETH.
- What if your transaction is not processed in time, and another user swaps, changing the pool's state?
- The output amount fluctuates as the pool updates, leading to slippage.
- What might cause slippage? One potential cause is MEV

Sandwich Attack

a form of front-running where an attacker places one transaction before and one after a target transaction to exploit price changes for profit.



Sandwich Attack

a form of front-running where an attacker places one transaction before and one after a target transaction to exploit price changes for profit.

- A user wants to swap USDC for ETH.
  - After the swap, the token balances in the pool will change:
  - USDC increases, while ETH decreases.
  - The price of ETH will rise because the USDC/ETH ratio increases.
- If an attacker notices a large swap is about to happen:
  - The attacker first swap USDC for ETH, expecting the price increase. (front-run)
  - The user's swap then executes, driving the price of ETH even higher.
  - The attacker sells their ETH at the inflated price, making a profit. (back-run)

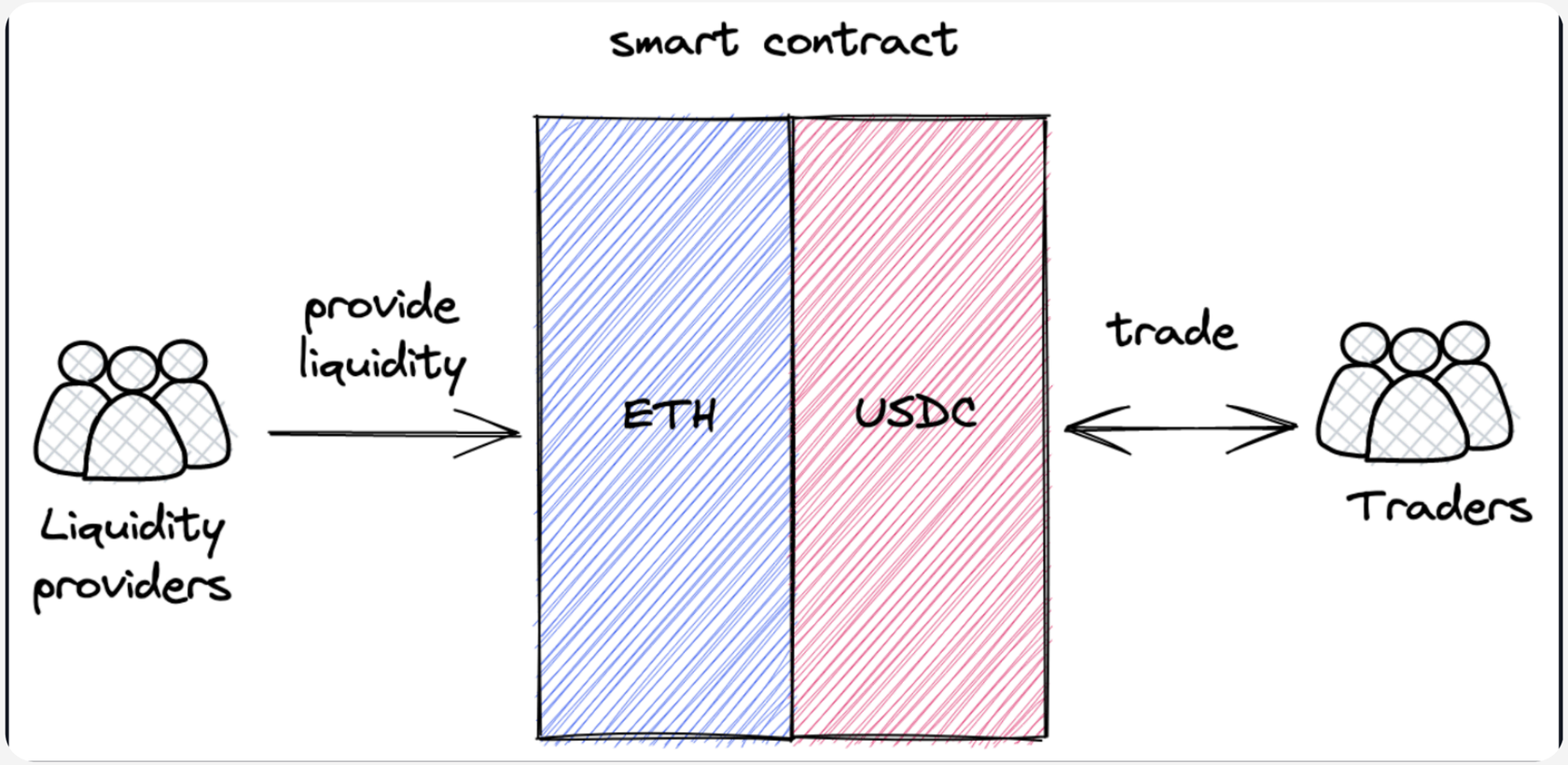
Liquidity Provider

Question: What is the incentive for LPs to provide their token as liquidity?

- Users can provide liquidity to any Uniswap v2 pair and, in return, receive a pool token representing their share.
- For each swap, a 0.3% fee is locked in the pair.
- Liquidity providers (LPs) earn more tokens proportional to the liquidity they contributed.
- This mechanism is similar to the ERC-4626 vault token standard, but it involves two tokens instead of just one.

Liquidity Provider

Workflow and the underlying mechanism



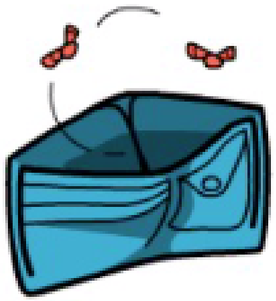
Impermenant Loss

It sounds good, is there any potential issue for providing liquidity?

- What if the token price is volatile?
- Providing liquidity to a Uniswap v2 pair means you are effectively holding the token.
- You might earn interest but risk losing the principal.

Impermenant Loss

Temporary loss of value experienced by liquidity providers when the price of tokens in a liquidity pool changes compared to simply holding the tokens outside the pool.



What is Impermanent Loss? DEFI Explained

Have you ever provided liquidity to a liquidity pool just to realise that some of your coins have gone...

Finematics /

Note: Impermenant vs Permenant Loss

Impermenant Loss

Temporary loss of value experienced by liquidity providers when the price of tokens in a liquidity pool changes compared to simply holding the tokens outside the pool.

	PRICE	SUPPLIED	VALUE
	$1	10000	$10000
	$500	20	$10000

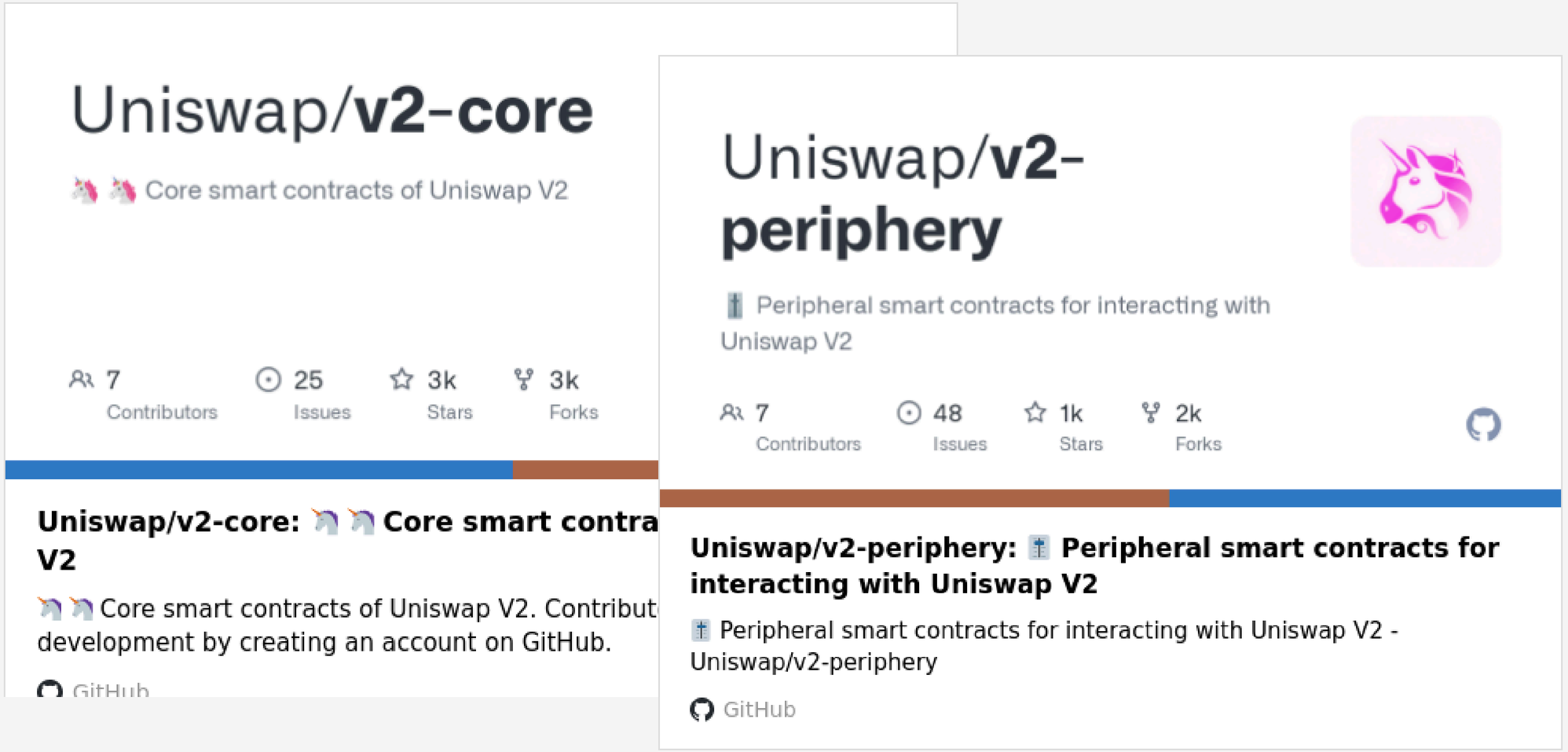
The original ratio

Ratio after arbitrage

							TOTAL
BEFORE ARB	10000	+	20	X	$500	=	$20000
AFTER ARB	10488.09	+	19.07	X	$550	=	$20976.59
			IMPERMANENT LOSS ←				$23.41
	10000	+	20	X	$550	=	$21000

Note: Impermenant vs Permenant Loss

The Codebase Walkthrough



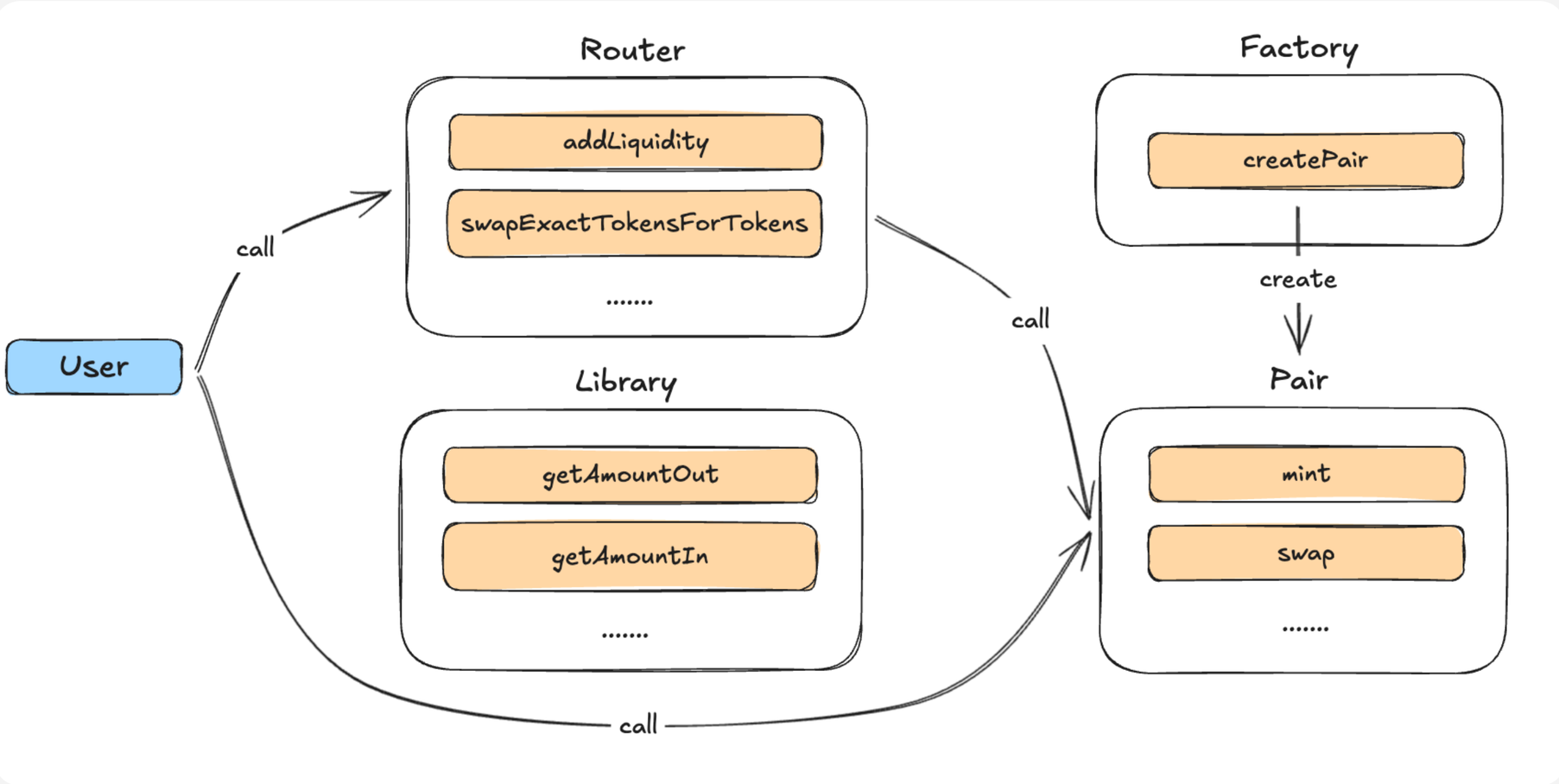
Design Pattern

Core / Periphery Pattern

- Core: The core logic of the protocol, such as the rules for token swaps, liquidity management, and other fundamental operations. It is immutable and ensures the protocol operates securely and consistently.
- Periphery: Includes the interfaces and helper contracts that interact with the core, allowing users and developers to integrate with the protocol in a flexible and upgradeable manner. These components can be modified or extended without changing the core.

Contract Architecture

Pair / Factory / Router



Version Difference

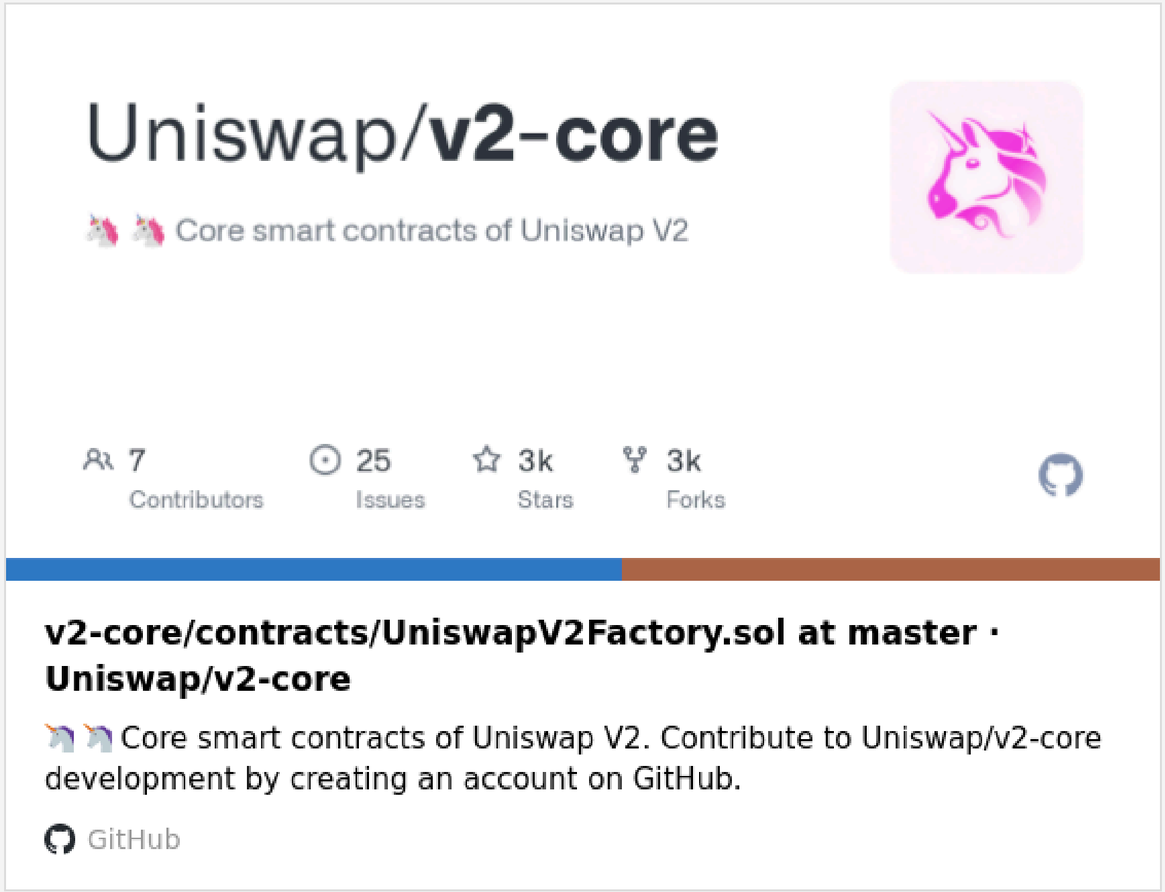
Before Solidity 0.8.0, there is no overflow / underflow checks

```
library SafeMath {
    function add(uint x, uint y) internal pure returns (uint z) {
        require((z = x + y) >= x, 'ds-math-add-overflow');
    }

    function sub(uint x, uint y) internal pure returns (uint z) {
        require((z = x - y) <= x, 'ds-math-sub-underflow');
    }

    function mul(uint x, uint y) internal pure returns (uint z) {
        require(y == 0 || (z = x * y) / y == x, 'ds-math-mul-overflow');
    }
}
```

Part 1: Core Component

Factory Contract

The developer can create a new Uniswap pair using the factory contract.



Factory Contract

Please review the codebase, and what is the major functionality in the factory

- Problem 1: Which state variable tracks the created pair?
- Problem 2: How is a new pair created? Using which method?
- Problem 3: If a (USDC, WETH) pair exists, can a (WETH, USDC) pair also exist?

UniswapV2Factory::createPair

There are two arguments in the function: token0 and token1

- Token validation: (1) No identical address (2) No zero address (3) No existing pair
- The token is sorted based on the address > uint256(uint160(address(addr)))
- If there is already an (USDC, WETH) pair, the (WETH, USDC) pair will not exist.

```
getPair[token0][token1] = pair;
getPair[token1][token0] = pair; // populate mapping in the reverse direction
```

This will be used if identical token pair is passed as argument

UniswapV2Factory::createPair

How do the factory create a new pair? There is no “new” keyword.

```
bytes memory bytecode = type(UniswapV2Pair).creationCode;
bytes32 salt = keccak256(abi.encodePacked(token0, token1));
assembly {
    pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
}
```

What is the difference between create and create2?

Create vs Create2

There are two methods for creating contract

create

```
address = keccak256(rlp([sender_address,sender_nonce]))[12:]
```

create2

```
initialisation_code = memory[offset:offset+size]
address = keccak256(0xff + sender_address + salt + keccak256(initialisation_code))[12:]
```

Create vs Create2

There are two methods for creating contract

- **Predictable Contract Address**: CREATE2 allows the contract address to be determined in advance based on the sender's address, salt, and contract bytecode, rather than only on the sender's nonce.
- **Re-deployment at Same Address**: Contracts can be re-deployed at the same address as long as the same salt and bytecode are used, even if the original contract was destroyed.

Create2 in Uniswap

Why we use create2 instead of create here?

```
// calculates the CREATE2 address for a pair without making any external calls
function pairFor(address factory, address tokenA, address tokenB) internal pure returns (address pair) {
    (address token0, address token1) = sortTokens(tokenA, tokenB);
    pair = address(uint(keccak256(abi.encodePacked(
            hex'ff',
            factory,
            keccak256(abi.encodePacked(token0, token1)),
            hex'96e8ac4277198ff8b6f785478aa9a39f403cb768dd02cbee326c3e7da348845f' // init code hash
        ))));
}
```

```
// this low-level function should be called from a contract which performs important safety checks
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
```

Pair

Please review the codebase, and what is the major functionality in the factory

- There are several operations in the contract, including
  - Swap: token exchange
  - Mint: Provide liquidity and mint LP token (share)
  - Burn: Provide LP token and receive the tokens as return
  - Sync / Skim: update the reserve / return the excessive tokens

UniswapV2Pair

Please review the codebase and check the state variable and functions

- MINIMUM_LIQUIDITY: later used for dead share, preventing inflation issue.
- SELECTOR: Used in the _safeTransfer internal function

```
function _safeTransfer(address token, address to, uint value) private {
    (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
    require(success && (data.length == 0 || abi.decode(data, (bool))), 'UniswapV2: TRANSFER_FAILED');
}
```

Do you remember the weird ERC-20, and the properties of USDT and USDC?

UniswapV2Pair

Please review the codebase and check the state variable and functions

- reserve0, reserve1: The cached token balance stored in the pair contract. It does not use **token.balanceOf(address(this))** directly.
- blockTimestampLast: Tracking the last updated timestamp for the TWAP price

```
uint112 private reserve0;           // uses single storage slot, accessible via getReserves
uint112 private reserve1;           // uses single storage slot, accessible via getReserves
uint32  private blockTimestampLast; // uses single storage slot, accessible via getReserves
```

Why using uint112 instead of uint256?

UniswapV2Pair

What if someone transfers tokens into the pair during a swap or other situations?

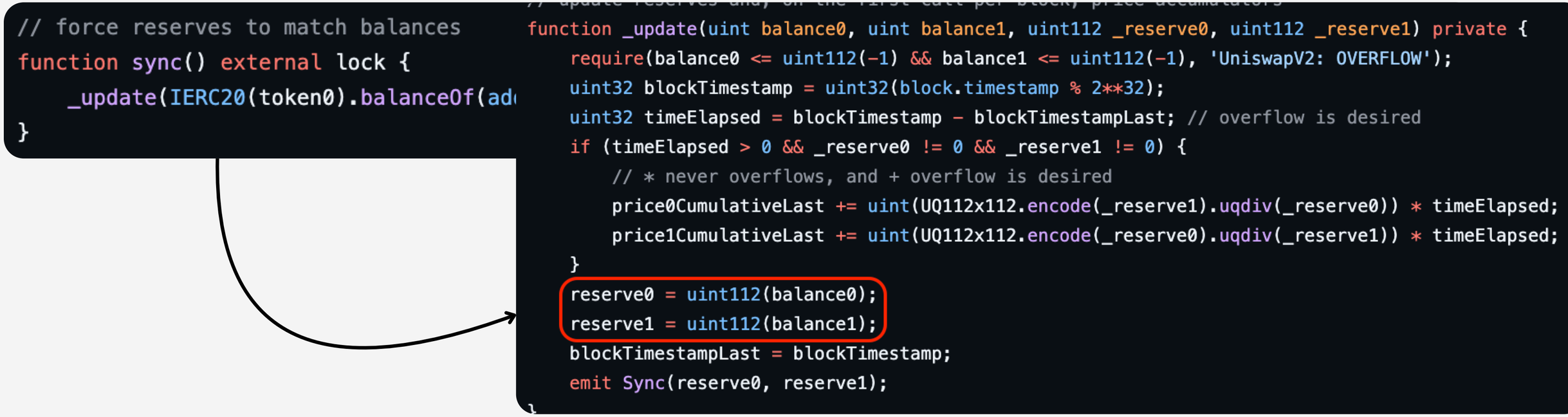
- reserve0, reserve1: The cached token balance stored in the pair contract. It does not use **token.balanceOf(address(this))** directly.

```
// force balances to match reserves
function skim(address to) external lock {
    address _token0 = token0; // gas savings
    address _token1 = token1; // gas savings
    _safeTransfer(_token0, to, IERC20(_token0).balanceOf(address(this)).sub(reserve0));
    _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).sub(reserve1));
}
```

UniswapV2Pair

What if someone transfers tokens into the pair during a swap or other situations?

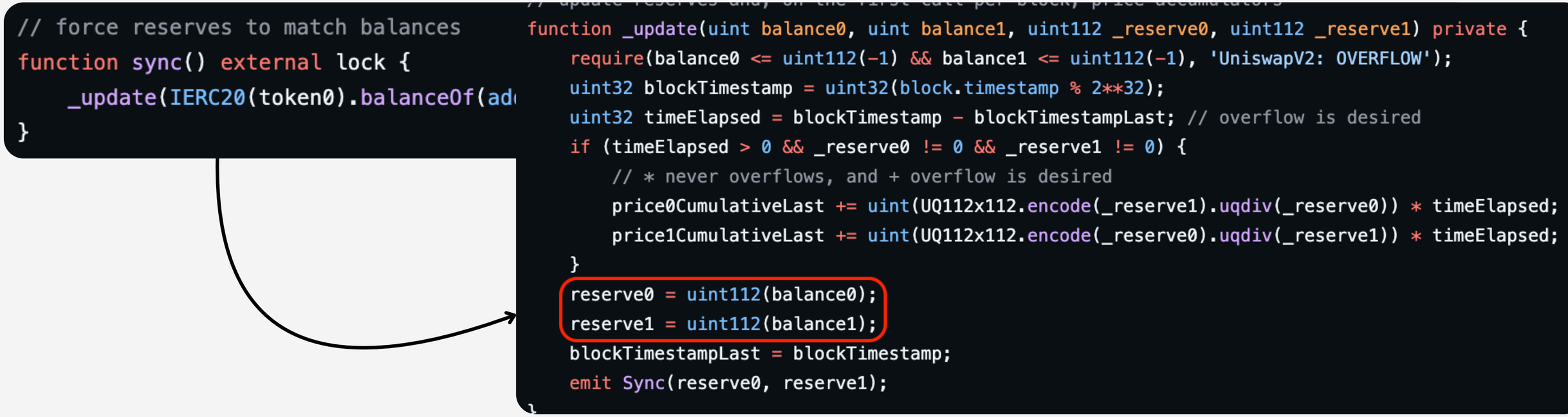
- reserve0, reserve1: The cached token balance stored in the pair contract. It does not use **token.balanceOf(address(this))** directly.



UniswapV2Pair

What if someone transfers tokens into the pair during a swap or other situations?

- reserve0, reserve1: The cached token balance stored in the pair contract. It does not use **token.balanceOf(address(this))** directly.



UniswapV2Pair::swap

How does the swap operation works?

- Verify input amount
- Cache current token reserve
- Ensure the token reserve is enough for the swap

```
// this low-level function should be called from a contract which performs important safety checks
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
    require(amount0Out > 0 || amount1Out > 0, 'UniswapV2: INSUFFICIENT_OUTPUT_AMOUNT');
    (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
    require(amount0Out < _reserve0 && amount1Out < _reserve1, 'UniswapV2: INSUFFICIENT_LIQUIDITY');
```

UniswapV2Pair::swap

How does the swap operation works?

- Cached the token for gas saving
- Validate the destination address
- Direct transfer the token to the destination address Flash swap callback - Identical to ERC3156

```
    uint balance0;
    uint balance1;
    { // scope for _token{0,1}, avoids stack too deep errors
    address _token0 = token0;
    address _token1 = token1;
    require(to != _token0 && to != _token1, 'UniswapV2: INVALID_TO');
    if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer tokens
    if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer tokens
    if (data.length > 0) IUniswapV2Callee(to).uniswapV2Call(msg.sender, amount0Out, amount1Out, data);
```

UniswapV2Pair::swap

How does the swap operation works?

- Calculate the input token amount
- Validate the input token amount

```
    balance0 = IERC20(_token0).balanceOf(address(this));
    balance1 = IERC20(_token1).balanceOf(address(this));
    }
    uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
    uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
    require(amount0In > 0 || amount1In > 0, 'UniswapV2: INSUFFICIENT_INPUT_AMOUNT');
```

UniswapV2Pair::swap

How to calculate the input amount?

```
uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
```

amount0Out -> the token that gives away

balance0 -> the final token balance stored in the pair

(1) balance0 > _reserve0 - amount0Out

If the current balance is greater than the remaining balance after deducting the output amount from the reserve.

It means there is some token coming in

(2) balance0 > _reserve0 - amount0Out

If not, it means there is no input token, that is, amount0In would be 0

UniswapV2Pair::swap

How to calculate the input amount?

- Deduct the fee and multiply by 1000 - Precision loss issue
- Validate the k value with a require statement
- Update the reserve balance

```
    uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
    uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
    require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000**2), 'UniswapV2: K');
    }

    _update(balance0, balance1, _reserve0, _reserve1);
    emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
```

UniswapV2Pair::mint

How does the mint operation works?

- There is no input token amount
- The input amount is the difference between the actual token balance and the reverse stored

```
// this low-level function should be called from a contract which performs important safety checks
function mint(address to) external lock returns (uint liquidity) {
    (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
    uint balance0 = IERC20(token0).balanceOf(address(this));
    uint balance1 = IERC20(token1).balanceOf(address(this));
    uint amount0 = balance0.sub(_reserve0);
    uint amount1 = balance1.sub(_reserve1);
```

UniswapV2Pair::mint

How does the mint operation works?

- There is a feeOn flag that indicates whether the mint fee should be chareged.

```
bool feeOn = _mintFee(_reserve0, _reserve1);
uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can update in _mintFee
if (_totalSupply == 0) {
    liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
   _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY tokens
} else {
    liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply) / _reserve1);
}
require(liquidity > 0, 'UniswapV2: INSUFFICIENT_LIQUIDITY_MINTED');
_mint(to, liquidity);
```

UniswapV2Pair::mint

How does the mint operation works?

- Two cases for minting liquidity
  - Case 1: It is the first mint, the totalSupply is zero
  - Case 2: There is already liquidity in the pair - Why take the minimum?

```
bool feeOn = _mintFee(_reserve0, _reserve1);
uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can update in _mintFee
if (_totalSupply == 0) {
    liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
   _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY tokens
} else {
    liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply) / _reserve1);
}
require(liquidity > 0, 'UniswapV2: INSUFFICIENT_LIQUIDITY_MINTED');
_mint(to, liquidity);
```

UniswapV2Pair::mint

Why is take the minimum value for liquidity minting?

- We compare two values when calculating the liquidity
  - amount0 / reserve0 - the ratio of token0
  - amount1 / reserve1 - the ratio of token1

```
bool feeOn = _mintFee(_reserve0, _reserve1);
uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can update in _mintFee
if (_totalSupply == 0) {
    liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
   _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY tokens
} else {
    liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply) / _reserve1);
}
require(liquidity > 0, 'UniswapV2: INSUFFICIENT_LIQUIDITY_MINTED');
_mint(to, liquidity);
```

UniswapV2Pair::burn

How does the burn operation works?

- Calculate the amount based on the ration of liquidity provided and total liquidity supply.

```
bool feeOn = _mintFee(_reserve0, _reserve1);
uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can update in _mintFee
amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata distribution
amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata distribution
require(amount0 > 0 && amount1 > 0, 'UniswapV2: INSUFFICIENT_LIQUIDITY_BURNED');
```

UniswapV2Pair::burn

How does the burn operation works?

- Burn the share, transfer the token, and update the reserve with current token balance

```
_burn(address(this), liquidity);
_safeTransfer(_token0, to, amount0);
_safeTransfer(_token1, to, amount1);
balance0 = IERC20(_token0).balanceOf(address(this));
balance1 = IERC20(_token1).balanceOf(address(this));

_update(balance0, balance1, _reserve0, _reserve1);
if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
emit Burn(msg.sender, amount0, amount1, to);
```

White Paper Revisit

Uniswap V2 Core White Paper

Uniswap v2 Core

Hayden Adams
hayden@uniswap.org

Noah Zinsmeister
noah@uniswap.org

Dan Robinson
dan@paradigm.xyz

March 2020

Abstract

This technical whitepaper explains some of the design decisions behind the Uniswap v2 core contracts. It covers the contracts' new features—including arbitrary pairs between ERC20s, a hardened price oracle that allows other contracts to estimate the time-weighted average price over a given interval, "flash swaps" that allow traders to receive assets and use them elsewhere before paying for them later in the transaction, and a protocol fee that can be turned on in the future. It also re-architects the contracts to reduce their attack surface. This whitepaper describes the mechanics of Uniswap v2's "core" contracts including the pair contract that stores liquidity providers' funds—and the factory contract used to instantiate pair contracts.

Part 1: Periphery Component

Uniswap V2 Periphery

There are several contracts in periphery, as a helper to mint, burn and swap

- There are two major contracts: UniswapV2Router01 & UniswapV2Router02
- UniswapV2Router01: All the fundamental operations
- UniswapV2Router02: Extend the v1 version with fee-on-transfer tokens
- Before diving into these contracts, we should take a look at the library

UniswapV2Library::sortToken

Do you remember the token address is sorted before the pair is created?

- Sort the token based on the uint256 value of the address type

```
// returns sorted token addresses, used to handle return values from pairs sorted in this order
function sortTokens(address tokenA, address tokenB) internal pure returns (address token0, address token1) {
    require(tokenA != tokenB, 'UniswapV2Library: IDENTICAL_ADDRESSES');
    (token0, token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
    require(token0 != address(0), 'UniswapV2Library: ZERO_ADDRESS');
}
```

UniswapV2Library::pairFor

When interacting with the router, we pass the token addresses, not the pair address.

- The pair address is deterministic, as it is created through create2 opcode.

```
// calculates the CREATE2 address for a pair without making any external calls
function pairFor(address factory, address tokenA, address tokenB) internal pure returns (address pair) {
    (address token0, address token1) = sortTokens(tokenA, tokenB);
    pair = address(uint(keccak256(abi.encodePacked(
            hex'ff',
            factory,
            keccak256(abi.encodePacked(token0, token1)),
            hex'96e8ac4277198ff8b6f785478aa9a39f403cb768dd02cbee326c3e7da348845f' // init code hash
        ))));
}
```

UniswapV2Library::quote

Given a certain amount of one asset and the pair's reserves, it returns the equivalent amount of the other asset.

- reverse0: reserve1 = amount0: amount1

```
// given some amount of an asset and pair reserves, returns an equivalent amount of the other asset
function quote(uint amountA, uint reserveA, uint reserveB) internal pure returns (uint amountB) {
    require(amountA > 0, 'UniswapV2Library: INSUFFICIENT_AMOUNT');
    require(reserveA > 0 && reserveB > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
    amountB = amountA.mul(reserveB) / reserveA;
}
```

UniswapV2Library::getAmountOut

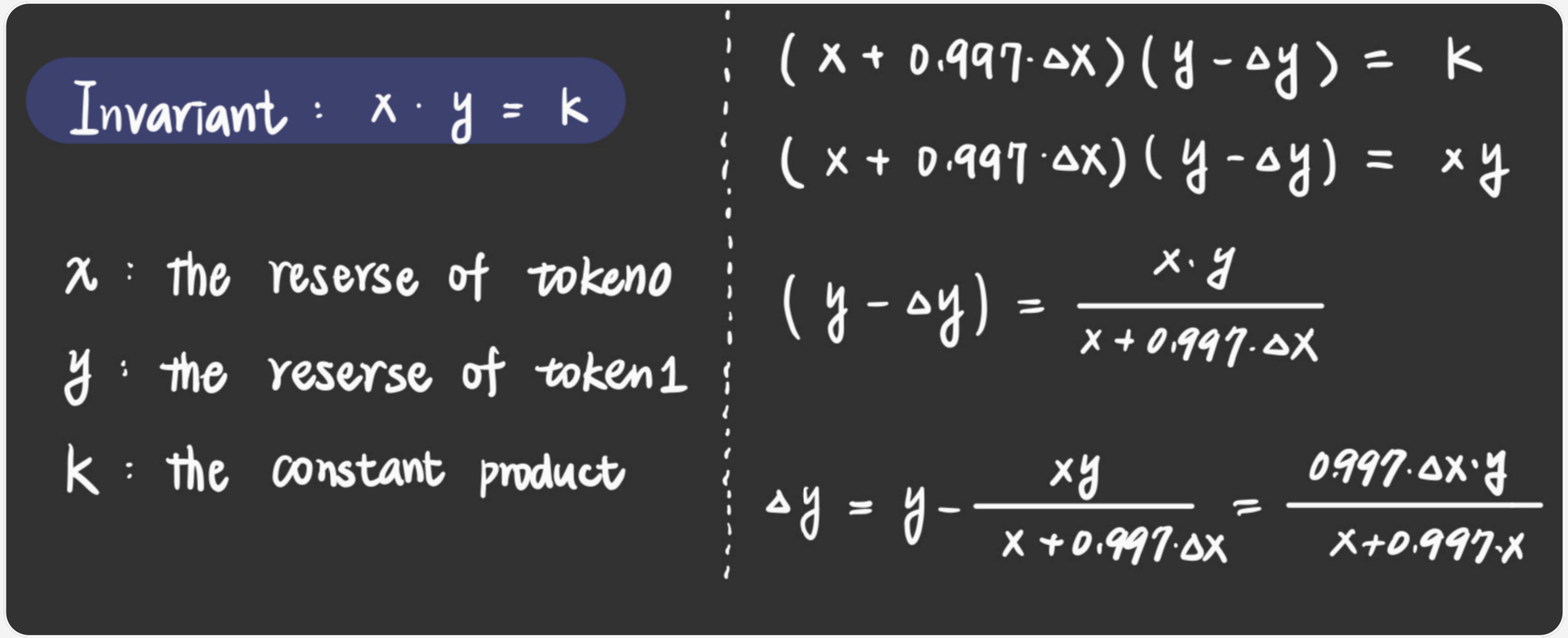
Do you remember the swap equation with 0.3% swap fee?

- getAmountOut: Given input amount and reserves, return output amount
- getAmountIn: Given output amount and reserves, return input amount

```
// given an input amount of an asset and pair reserves, returns the maximum output amount of the other asset
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) internal pure returns (uint amountOut) {
    require(amountIn > 0, 'UniswapV2Library: INSUFFICIENT_INPUT_AMOUNT');
    require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
    uint amountInWithFee = amountIn.mul(997);
    uint numerator = amountInWithFee.mul(reserveOut);
    uint denominator = reserveIn.mul(1000).add(amountInWithFee);
    amountOut = numerator / denominator;
}
```

UniswapV2Library::getAmountOut

Do you remember the swap equation with 0.3% swap fee?



UniswapV2Library::getAmountsOut

Do you remember the swap equation with 0.3% swap fee?

- Given a path, conduct swap on each pair in the array
- The output amount in each swap is the input amount in the next swap

```
// performs chained getAmountOut calculations on any number of pairs
function getAmountsOut(address factory, uint amountIn, address[] memory path) internal view returns (uint[] memory amounts) {
    require(path.length >= 2, 'UniswapV2Library: INVALID_PATH');
    amounts = new uint[](path.length);
    amounts[0] = amountIn;
    for (uint i; i < path.length - 1; i++) {
        (uint reserveIn, uint reserveOut) = getReserves(factory, path[i], path[i + 1]);
        amounts[i + 1] = getAmountOut(amounts[i], reserveIn, reserveOut);
    }
}
```

UniswapV2Library::getAmountsOut

Do you remember the swap equation with 0.3% swap fee?

- Given a path, conduct swap on each pair in the array
- The output amount in each swap is the input amount in the next swap

```
// performs chained getAmountOut calculations on any number of pairs
function getAmountsOut(address factory, uint amountIn, address[] memory path) internal view returns (uint[] memory amounts) {
    require(path.length >= 2, 'UniswapV2Library: INVALID_PATH');
    amounts = new uint[](path.length);
    amounts[0] = amountIn;
    for (uint i; i < path.length - 1; i++) {
        (uint reserveIn, uint reserveOut) = getReserves(factory, path[i], path[i + 1]);
        amounts[i + 1] = getAmountOut(amounts[i], reserveIn, reserveOut);
    }
}
```

Now, let's go back to the router contract

Router::swapExactTokensForTokens

Given input token, amount and the path, conduct the swap and get output result

- amountOutMin: specify the minimum output token amount
- deadline: specify the last timestamp the transaction should be executed

```
function swapExactTokensForTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external override ensure(deadline) returns (uint[] memory amounts) {
    amounts = UniswapV2Library.getAmountsOut(factory, amountIn, path);
    require(amounts[amounts.length - 1] >= amountOutMin, 'UniswapV2Router: INSUFFICIENT_OUTPUT_AMOUNT');
    TransferHelper.safeTransferFrom(path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0], path[1]), amounts[0]);
    _swap(amounts, path, to);
}
```

Slippage Check, this is very important when integration

Router::swapExactTokensForTokens

Given input token, amount and the path, conduct the swap and get output result

- First validate the amountOutMin and deadline
- Transfer the token to the router: the user should first approve the router

```
function swapExactTokensForTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external override ensure(deadline) returns (uint[] memory amounts) {
    amounts = UniswapV2Library.getAmountsOut(factory, amountIn, path);
    require(amounts[amounts.length - 1] >= amountOutMin, 'UniswapV2Router: INSUFFICIENT_OUTPUT_AMOUNT');
    TransferHelper.safeTransferFrom(path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0], path[1]), amounts[0]);
    _swap(amounts, path, to);
}
```

Router::swapExactTokensForTokens

Given input token, amount and the path, conduct the swap and get output result

- Loop over the path and conduct swap for each token pair

```
// requires the initial amount to have already been sent to the first pair
function _swap(uint[] memory amounts, address[] memory path, address _to) private {
    for (uint i; i < path.length - 1; i++) {
        (address input, address output) = (path[i], path[i + 1]);
        (address token0,) = UniswapV2Library.sortTokens(input, output);
        uint amountOut = amounts[i + 1];
        (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) : (amountOut, uint(0));
        address to = i < path.length - 2 ? UniswapV2Library.pairFor(factory, output, path[i + 2]) : _to;
        IUniswapV2Pair(UniswapV2Library.pairFor(factory, input, output)).swap(amount0Out, amount1Out, to, new bytes(0));
    }
}
```

Router Swap Operation

There are other variation for router to conduct swap operation

- swapTokensForExactTokens: Given output token, amount, and path, returns the token required for the swap.
- swapExactETHForTokens: Given **ETH**, amount and path, the router will first convert ETH to WETH and later conduct the swap.
- swapExactTokensForTokensSupportingFeeOnTransferTokens

Router::addLiquidity

Given input token, amount and the path, conduct the swap and get output result

- (amountADesired, amountBDesired)
- (amountAMin, amountBMin)

```
function addLiquidity(
    address tokenA,
    address tokenB,
    uint amountADesired,
    uint amountBDesired,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
) external override ensure(deadline) returns (uint amountA, uint amountB, uint liquidity) {
    (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired, amountBDesired, amountAMin, amountBMin);
    address pair = UniswapV2Library.pairFor(factory, tokenA, tokenB);
    TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
    TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
    liquidity = IUniswapV2Pair(pair).mint(to);
}
```

Router::addLiquidity

If the pair does not exist, the pair will be created first.

- (amountADesired, amountBDesired)
- (amountAMin, amountBMin)

```
// create the pair if it doesn't exist yet
if (IUniswapV2Factory(factory).getPair(tokenA, tokenB) == address(0)) {
    IUniswapV2Factory(factory).createPair(tokenA, tokenB);
}
```

Router::addLiquidity

If the pair does not exist, the pair will be created first.

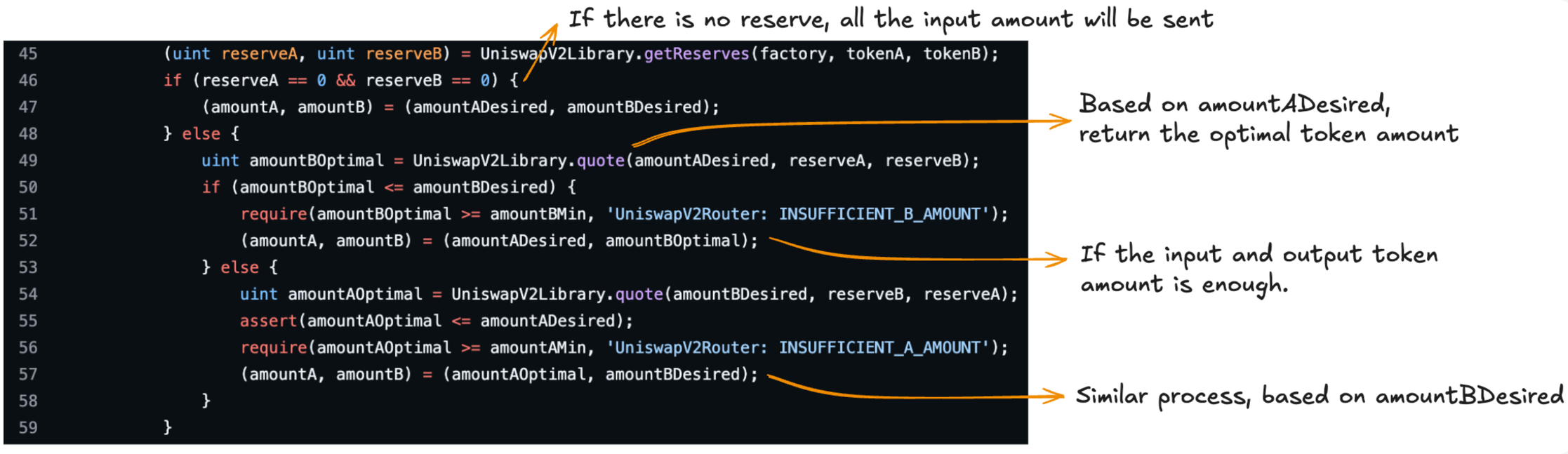
- For each condition, let's check how it is validated.

```
45            (uint reserveA, uint reserveB) = UniswapV2Library.getReserves(factory, tokenA, tokenB);
46            if (reserveA == 0 && reserveB == 0) {
47                (amountA, amountB) = (amountADesired, amountBDesired);
48            } else {
49                uint amountBOptimal = UniswapV2Library.quote(amountADesired, reserveA, reserveB);
50                if (amountBOptimal <= amountBDesired) {
51                    require(amountBOptimal >= amountBMin, 'UniswapV2Router: INSUFFICIENT_B_AMOUNT');
52                    (amountA, amountB) = (amountADesired, amountBOptimal);
53                } else {
54                    uint amountAOptimal = UniswapV2Library.quote(amountBDesired, reserveB, reserveA);
55                    assert(amountAOptimal <= amountADesired);
56                    require(amountAOptimal >= amountAMin, 'UniswapV2Router: INSUFFICIENT_A_AMOUNT');
57                    (amountA, amountB) = (amountAOptimal, amountBDesired);
58                }
59            }
```

Router::addLiquidity

If the pair does not exist, the pair will be created first.

- For each condition, let's check how it is validated.




```
45            (uint reserveA, uint reserveB) = UniswapV2Library.getReserves(factory, tokenA, tokenB);
46            if (reserveA == 0 && reserveB == 0) {
47                (amountA, amountB) = (amountADesired, amountBDesired);
48            } else {
49                uint amountBOptimal = UniswapV2Library.quote(amountADesired, reserveA, reserveB);
50                if (amountBOptimal <= amountBDesired) {
51                    require(amountBOptimal >= amountBMin, 'UniswapV2Router: INSUFFICIENT_B_AMOUNT');
52                    (amountA, amountB) = (amountADesired, amountBOptimal);
53                } else {
54                    uint amountAOptimal = UniswapV2Library.quote(amountBDesired, reserveB, reserveA);
55                    assert(amountAOptimal <= amountADesired);
56                    require(amountAOptimal >= amountAMin, 'UniswapV2Router: INSUFFICIENT_A_AMOUNT');
57                    (amountA, amountB) = (amountAOptimal, amountBDesired);
58                }
59            }
```

Router::addLiquidity

There are other variation for router to conduct swap operation

- **addLiquidityETH**:
- **addLiquidity**:

Router::removeLiquidity

There are other variation for router to conduct swap operation

- (tokenA, tokenB, liquidity)

```
102         // **** REMOVE LIQUIDITY ****
103         function removeLiquidity(
104             address tokenA,
105             address tokenB,
106             uint liquidity,
107             uint amountAMin,
108             uint amountBMin,
109             address to,
110             uint deadline
111         ) public virtual override ensure(deadline) returns (uint amountA, uint amountB) {
```

Router::removeLiquidity

There are other variation for router to conduct swap operation

- (amountAMin, amountBMin)
- deadline

```
102         // **** REMOVE LIQUIDITY ****
103         function removeLiquidity(
104             address tokenA,
105             address tokenB,
106             uint liquidity,
107             uint amountAMin,
108             uint amountBMin,
109             address to,
110             uint deadline
111         ) public virtual override ensure(deadline) returns (uint amountA, uint amountB) {
```

Router::removeLiquidity

There are other variation for router to conduct swap operation

- (amountAMin, amountBMin)
- deadline

```
102         // **** REMOVE LIQUIDITY ****
103         function removeLiquidity(
104             address tokenA,
105             address tokenB,
106             uint liquidity,
107             uint amountAMin,
108             uint amountBMin,
109             address to,
110             uint deadline
111         ) public virtual override ensure(deadline) returns (uint amountA, uint amountB) {
```

Router::removeLiquidity

There are other variation for router to conduct swap operation

- Get the pair token through the pairFor helper
- Privde the liqudity and burn the share

```
112             address pair = UniswapV2Library.pairFor(factory, tokenA, tokenB);
113             IUniswapV2Pair(pair).transferFrom(msg.sender, pair, liquidity); // send liquidity to pair
114             (uint amount0, uint amount1) = IUniswapV2Pair(pair).burn(to);
115             (address token0,) = UniswapV2Library.sortTokens(tokenA, tokenB);
116             (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1, amount0);
117             require(amountA >= amountAMin, 'UniswapV2Router: INSUFFICIENT_A_AMOUNT');
118             require(amountB >= amountBMin, 'UniswapV2Router: INSUFFICIENT_B_AMOUNT');
```

Router::removeLiquidity

There are other variation for router to conduct swap operation

- **removeLiquidityETH:**
- **removeLiquidityWithPermit:**
- **removeLiquidityETHSupportingFeeOnTransferTokens**
- **removeLiquidityETHWithPermitSupportingFeeOnTransferTokens**