Lesson 20 - Interacting with DeFi

Making a fork of mainnet

Hardhat

See also lesson 9
See hardhat documentation

You first need to have an account on Infura or Alchemy
This will give you a key so that you can use their RPC nodes.

Forking using ganache

`npx ganache-cli --f https://mainnet.infura.io/v3/<your key> -m "your 12 word mnemonic" --unlock <address> -i <chain ID>`

Fork from hardhat

```
npx hardhat node --fork https://eth-
mainnet.alchemyapi.io/v2/<your key>
```

In hardhat you can also specify this in the config file

```
networks: {
  hardhat: {
    forking: {
      url: "https://eth-mainnet.alchemyapi.io/v2/<key>",
    }
  }
}
```

Foundry

You can use:

`forge test --fork-url <your_rpc_url> -vv`

where your_rpc_url is your node details as above.

Or you can use Anvil (part of the Foundry suite) for more control:

```
anvil --fork-url <your_rpc_url>
```

With anvil you are able to use the `anvil_impersonateAccount` custom method to impersonate EOA's.

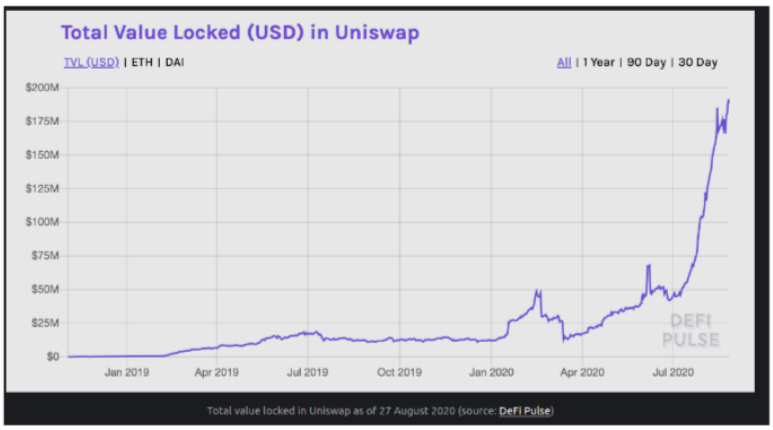
Please read here for more info

Uniswap

The first ideas came from Vitalik, Nick Johnson and Martin Koppelmann in 2016 in a Reddit post

It was followed by an implementation from Hayden Adams and launched in Nov 2018

- Launched in 2018, Uniswap is a DEX featuring an AMM
- Solves the problem of illiquid assets since anyone can set up a liquidity pool



- Truly Decentralised
- Allows swap between any ERC20 pairs
- The code is robust

V2 Launched May 2020 allowing direct token swaps - halving gas fees

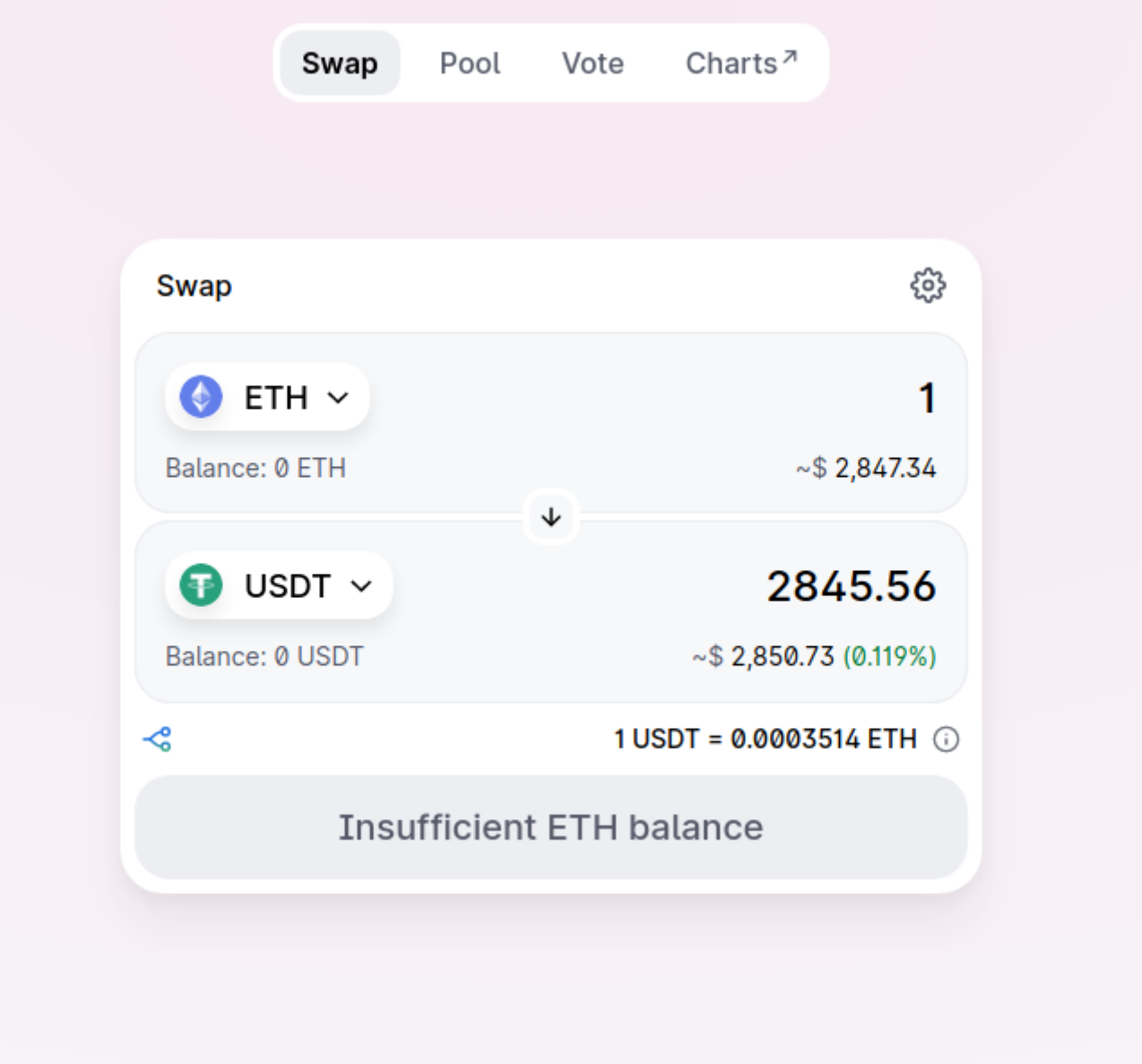
It solved many of the problems of the initial exchanges such as lack of incentives to provide liquidity for rarely traded assets.
It relies on a smart contract acting as an automatic market maker (AMM)

Incentivising Users

- Users deposit funds into a liquidity pool, for example ETH and USDT
- This pool (a token pair) allows users to exchange tokens
- Interacting with the exchange incurs fees
- These fees are paid to the liquidity providers

The AMM is more specifically a constant function market maker.

The term “constant function” refers to the fact that any trade must change the reserves in such a way that the product of those reserves remains unchanged (i.e. equal to a constant).



LP Tokens

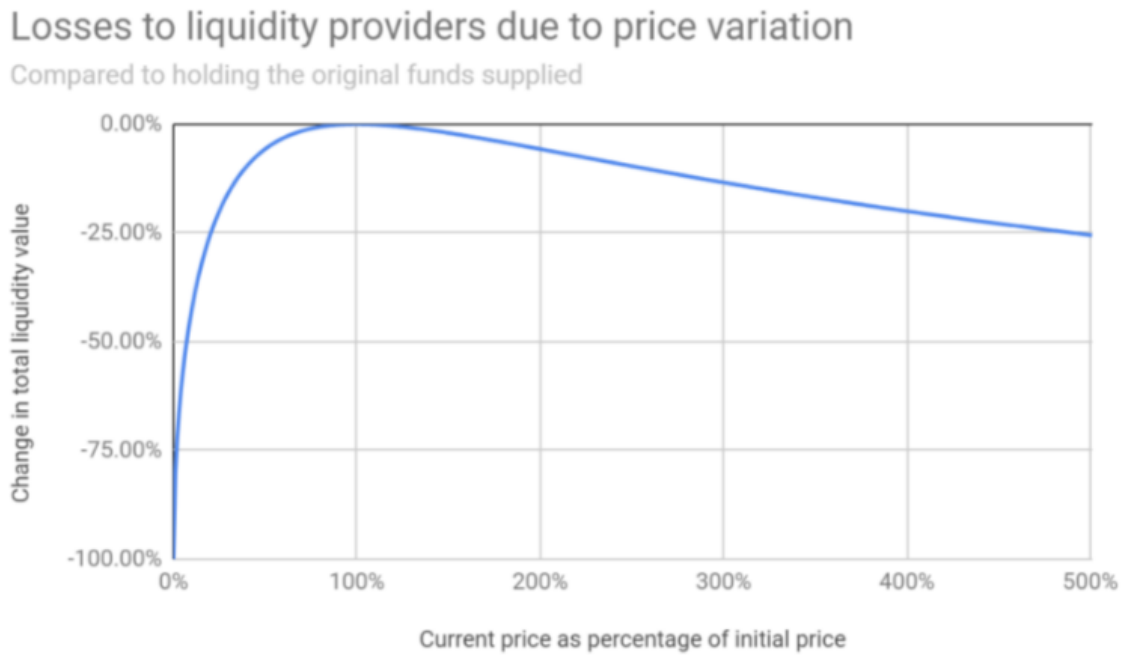
Typically the liquidity provider receives LP tokens when they add liquidity, say ETH and USDT

Later they can take liquidity by providing LP tokens to the contract and will receive back ETH and USDT. Ideally they will make a profit

Risks when swapping or providing liquidity

- Slippage
  Large trades can move the price

- Impermanent loss
  As a result of volatility


Losses to liquidity providers due to price variation
Compared to holding the original funds supplied
Change in total liquidity value
0.00%
-25.00%
-50.00%
-75.00%
-100.00%
0%
100%
200%
300%
400%
500%
Current price as percentage of initial price

Uniswap Details

Core Repo
Periphery Repo

Router Interface
https://docs.uniswap.org/protocol/reference/periphery/interfaces/ISwapRouter
https://github.com/Uniswap/v3-periphery/blob/main/contracts/SwapRouter.sol

Uniswap V3 Router
https://etherscan.io/address/0xe592427a0aece92de3edee1f18e0157c05861564
Code
https://etherscan.io/address/0xe592427a0aece92de3edee1f18e0157c05861564#code

Guide to single swaps

From Uniswap docs

Swaps are the most common interaction with the Uniswap protocol.
The `exactInputSingle` function is for performing *exact input* swaps, which swap a fixed amount of one token for a maximum possible amount of another token. This function uses the `ExactInputSingleParams` struct and the `exactInputSingle` function from the ISwapRouter interface.

When trading from a smart contract, the most important thing to keep in mind is that access to an external price source is required. Without this, trades can be front run for considerable loss.

Exact Input Swaps

The caller must `approve` the contract to withdraw the tokens from the calling address's account to execute a swap. Remember that because our contract is a contract itself and not an extension of the caller (us); we must also approve the Uniswap protocol router contract to use the tokens that our contract will be in possession of after they have been withdrawn from the calling address (us).

To execute the swap function, we need to populate the `ExactInputSingleParams` with the necessary swap data. These parameters are found in the smart contract interfaces, which can be browsed here.

The function parameters:

- `tokenIn` The contract address of the inbound token
- `tokenOut` The contract address of the outbound token
- `fee` The fee tier of the pool, used to determine the correct pool contract in which to execute the swap

- `recipient` the destination address of the outbound token
- `deadline`: the unix time after which a swap will fail, to protect against long-pending transactions and wild swings in prices
- `amountOutMinimum`: we are setting to zero, but this is a significant risk in production. For a real deployment, this value should be calculated using our SDK or an onchain price oracle - this helps protect against getting an unusually bad price for a trade due to a front running sandwich or another type of price manipulation
- `sqrtPriceLimitX96`: We set this to zero - which makes this parameter inactive. In production, this value can be used to set the limit for the price the swap will push the pool to, which can help protect against price impact or for setting up logic in a variety of price-relevant mechanisms.

Calling the function from Solidity

```
// Naively set amountOutMinimum to 0. In production, use an
oracle or other data source to choose a safer value for
amountOutMinimum.
// We also set the sqrtPriceLimitx96 to be 0 to ensure we swap
our exact input amount.

                ISwapRouter.ExactInputSingleParams memory params
=
            ISwapRouter.ExactInputSingleParams({
                tokenIn: DAI,
                tokenOut: WETH9,
                fee: poolFee,
                recipient: msg.sender,
                deadline: block.timestamp,
                amountIn: amountIn,
                amountOutMinimum: 0,
                sqrtPriceLimitX96: 0
            });

        // The call to `exactInputSingle` executes the swap.
        amountOut = swapRouter.exactInputSingle(params);
    }
```