CS251 Fall 2025

(cs251.stanford.edu)

Decentralized Exchanges

Dan Boneh

First: review of lending protocols

Over-collateralized loans: to borrow, borrower has to provide

$$0.8 \times \textit{value(collateral)} > \textit{value(debt)}$$

Loan-to-Value (LTV) ratio

Liquidation Threshold

Liquidation:

if $0.83 \times \textit{value(collateral)} < \textit{value(debt)}$

then collateral is liquidated to pay debt until inequality flips

(liquidation reduces both sides of the inequality)

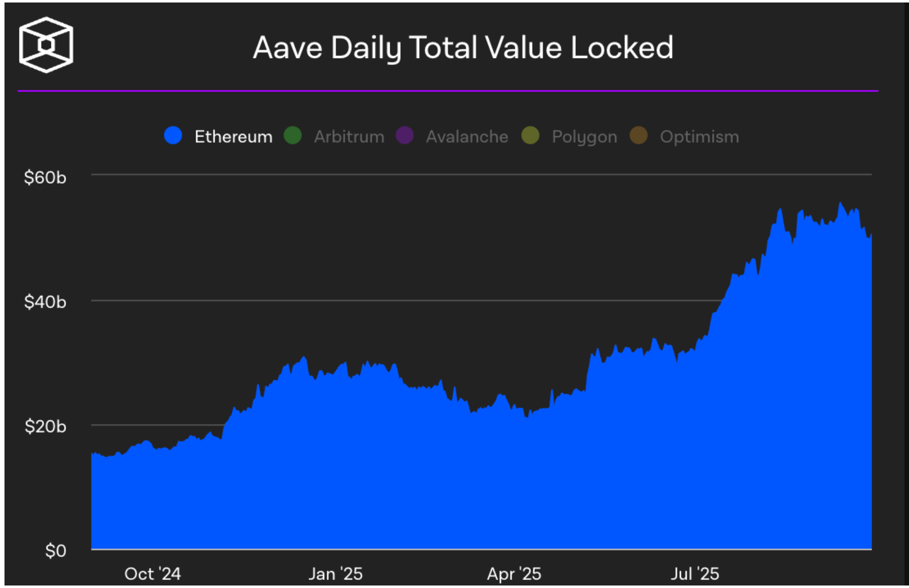
Lending protocols

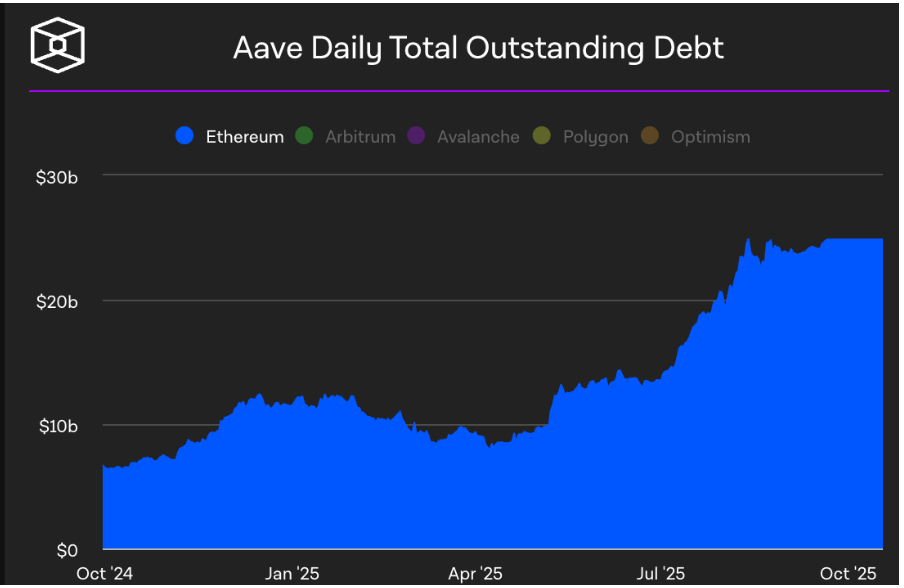
- **Liquidity providers** send asset to the lending smart contract
  - Get tokens representing their assets in the protocol
  - Can redeem tokens at any time to get back their assets

- **Borrowers** borrow from the smart contract
  - Interest payments flow to the liquidity providers
  - In case of a default, borrower loses (part) of its collateral
  - In case of a liquidation, borrower loses (part) of its collateral

Stats

DeFi lending usage (on Ethereum):



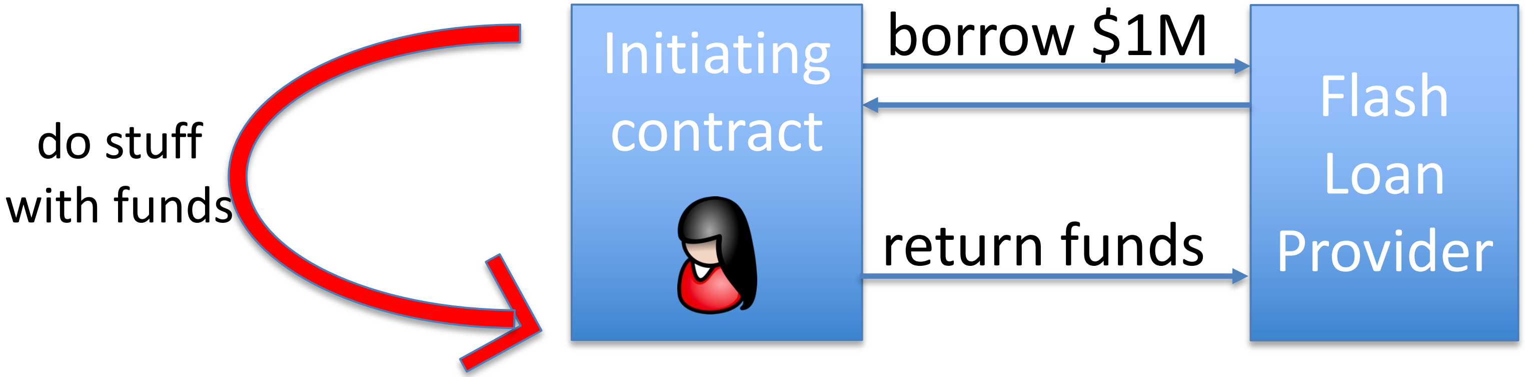


A unique concept: flash loans

What is a flash loan?

A flash loan is taken and repaid in a <u>single</u> transaction

$\Longrightarrow$ zero risk for lender $\Longrightarrow$ borrower needs no collateral



(Tx is valid only if funds are returned in same Tx)

"Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit"

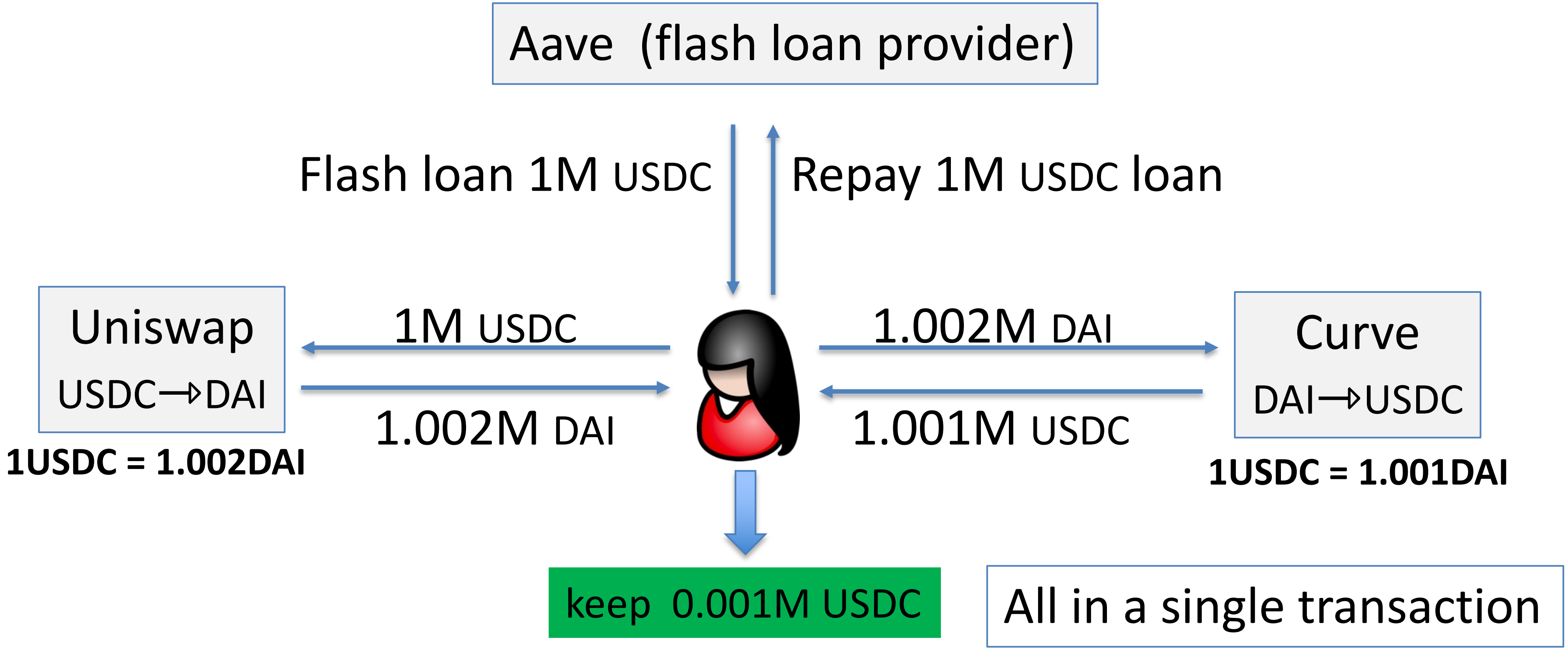
Use cases

- Risk free arbitrage
- Collateral swap
- DeFi attacks: price oracle manipulation

⋮

Risk free arbitrage

Alice finds a USDC/DAI price difference in two pools



All in a single transaction

Collateral swap

start:

Alice @Compound

-1000 DAI
+1 cETH

borrowed DAI using ETH as collateral

Take 1000 DAI flash loan
Repay 1000 DAI debt (@Compund)
Redeem 1 cETH (from Compound)
Swap 1 cETH for 1500 cUSDC
Deposit 1500 cUSDC as collateral
Borrow 1000 DAI
Repay 1000 DAI flash loan

(a single Ethereum transaction)

end goal:

Alice @Compound

-1000 DAI
+1500 cUSDC

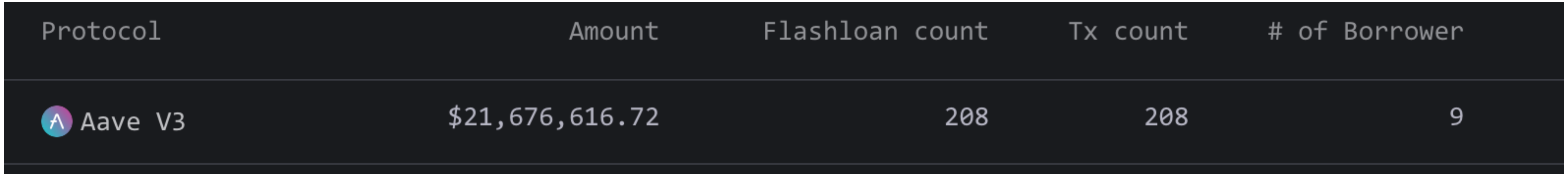
borrowed DAI using USDC as collateral

Aave v1 implementation

```
function flashLoan(address _receiver, uint256 _amount) {
    …
    // transfer funds to the receiver
    core.transferToUser(_reserve, userPayable, _amount);

    // execute action of the receiver
    receiver.executeOperation(_reserve, _amount, amountFee, _params);
    …
    // abort if loan is not repaid
    require( availableLiquidityAfter == availableLiquidityBefore.add(amountFee),
            "balance inconsistent");
}
```

Flash loans on Aave



Protocol	Amount	Flashloan count	Tx count	# of Borrower
Aave V3	$21,676,616.72	208	208	9

(24h period on 10/2025)

Decentralized Exchanges

(Dex)

Two types of exchanges (roughly speaking)

Spot exchange:

- Exchange (X of asset A) for (Y of asset B) based on the current A to B exchange rate (spot price)

Perpetual exchange:

- Place a bet on the future exchange rate between A and B
  (e.g., pay X now for soy beans to be delivered in three months)

Both exchange types are are implemented on-chain in smart contracts

- We will focus on spot exchanges

What is a spot exchange?

Many types of ERC-20 tokens on Ethereum:

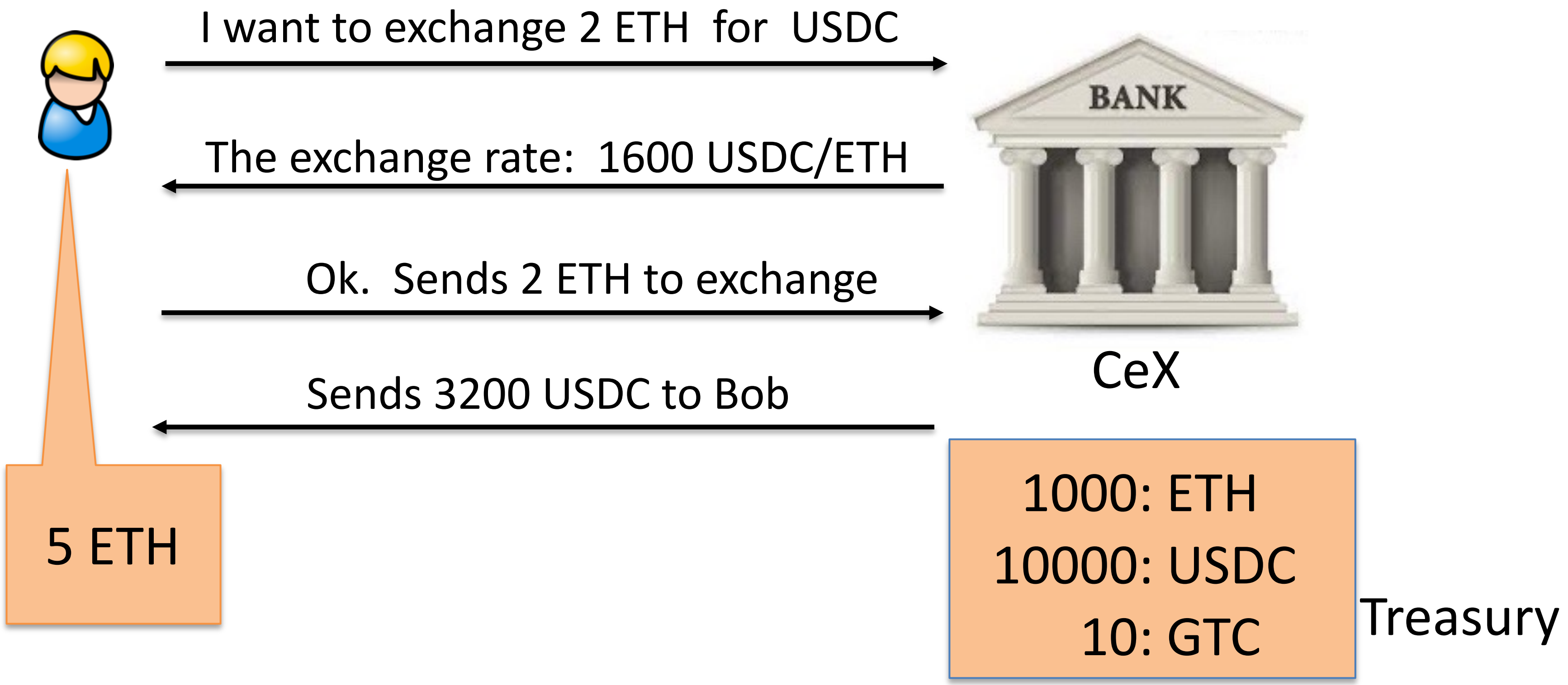
- WETH: ETH wrapped as an ERC-20, stETH: staked ETH
- USDC, USDT, DAI: USD stablecoins
- Governance tokens (e.g., GTC for Gitcoin),
- Gaming tokens

  …

An exchange: used to convert one token to another (e.g., USDC → GTC)

- What is the exchange rate?
- How to connect sellers and buyers?

First approach: a centralized exchange (CeX)



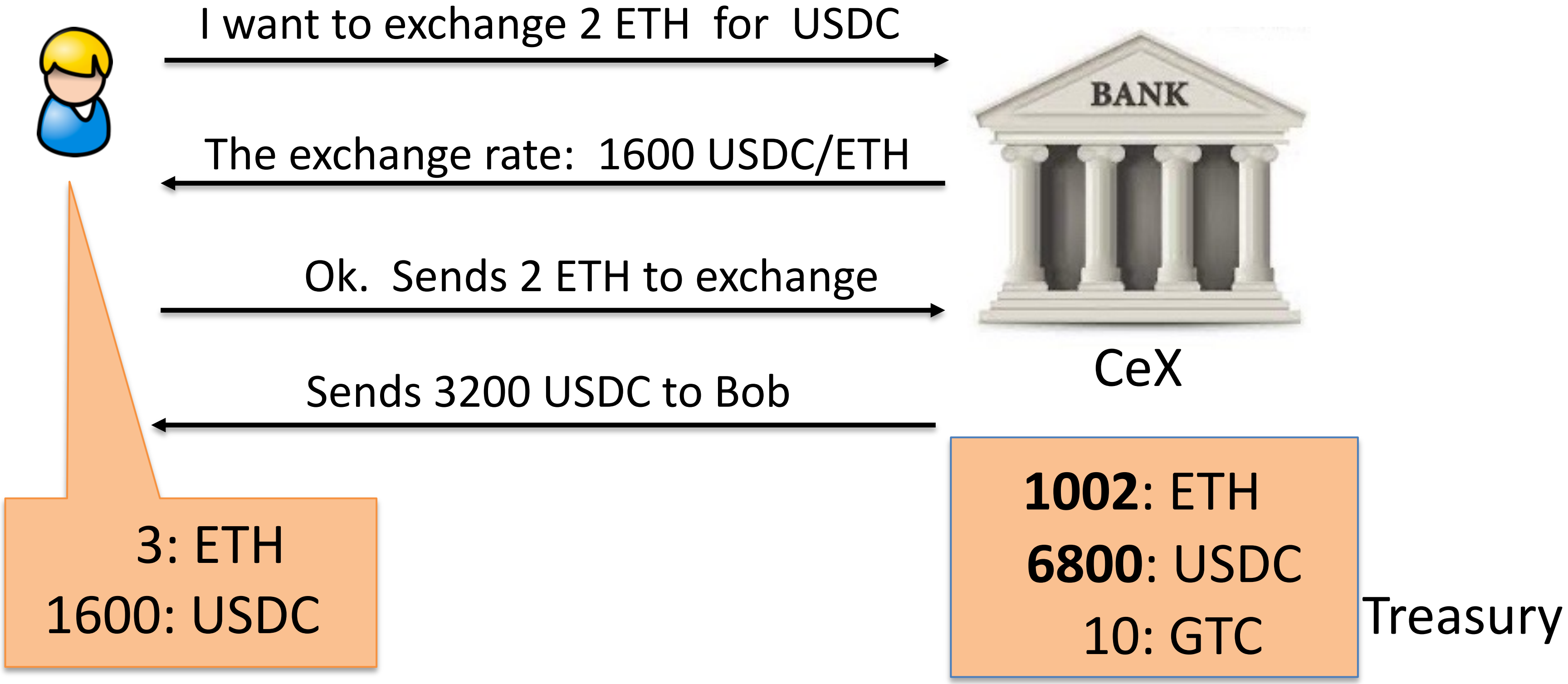
I want to exchange 2 ETH for USDC

The exchange rate: 1600 USDC/ETH

Ok. Sends 2 ETH to exchange

Sends 3200 USDC to Bob

First approach: a centralized exchange (CeX)



I want to exchange 2 ETH for USDC

The exchange rate: 1600 USDC/ETH

Ok. Sends 2 ETH to exchange

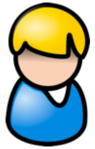
Sends 3200 USDC to Bob

3: ETH
1600: USDC

1002: ETH
6800: USDC
10: GTC

Treasury

First approach: a centralized exchange (CeX)



Many order types

Example: **Limit order**:

I am willing to buy

1 ETH for up to 1700 USDC

[for the next 24 hours]

CeX

The exchange either ”fills” the order, or not.

A list of such buy/sell orders is called an **order book**

Some issues …

How is exchange rate determined?

- By supply and demand at the exchange (not transparent)
- Competition with other exchanges (bad user experience)

Security: What if exch. takes Bob’s 2 ETH, but never sends USDC?

Censorship: What if exchange refuses to do business with Bob?

A more trusted solution: DeX

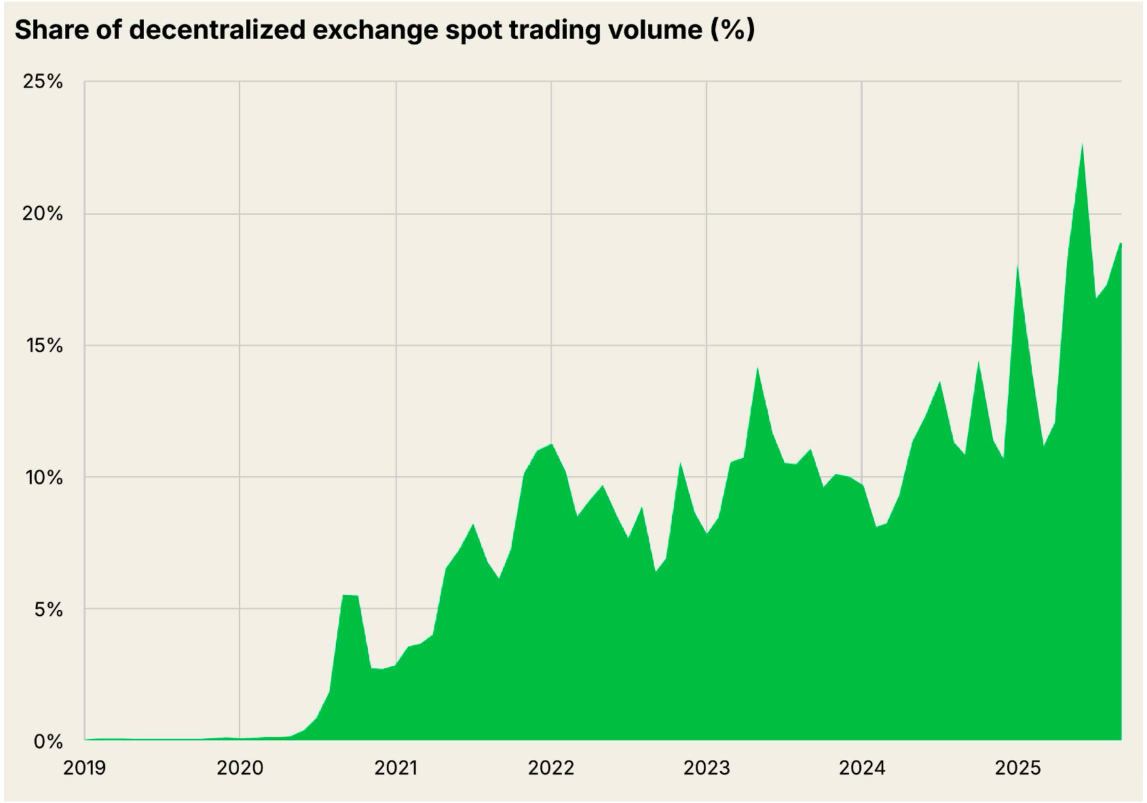
What is a DEX?

- a marketplace where transactions occur directly between participants, **without a trusted intermediary**

Properties:

- Programmable: can be used as a service by other contracts
- Transparent: code is available for everyone to see
- Permissionless: anyone can use
- Non-Custodial (never holds user assets)

Share of trading on DeX vs. CeX



source: state of crypto 2025 report

How to build a DeX?

First idea: on-chain order book

- Liquidity providers place buy/sell orders on chain
- Traders fill them on chain

Problem: gas costs.

- Orders cost gas: when placed, when filled, when canceled.
- Matching buy orders to sell orders takes lots of gas (but see here)
- Only feasible on chains with cheap gas … which exist!

How to build a DeX?

Next idea: off-chain order book

- Liquidity providers sign buy/sell orders off chain
    - Post orders on a centralized web site
- User signs an order it wants to fill and submits it on chain.
- Example: 0x Protocol

Problem: order book is not accessible to contracts (dAPPs)

How to build a DeX?

A very elegant idea: **Automated Market Maker** (AMM)

- Liquidity providers deposit assets into an on-chain pool
- Users trade with the on-chain pool
    - exchange rate is determined algorithmically
- Examples: Uniswap, Balancer, Bancor, …

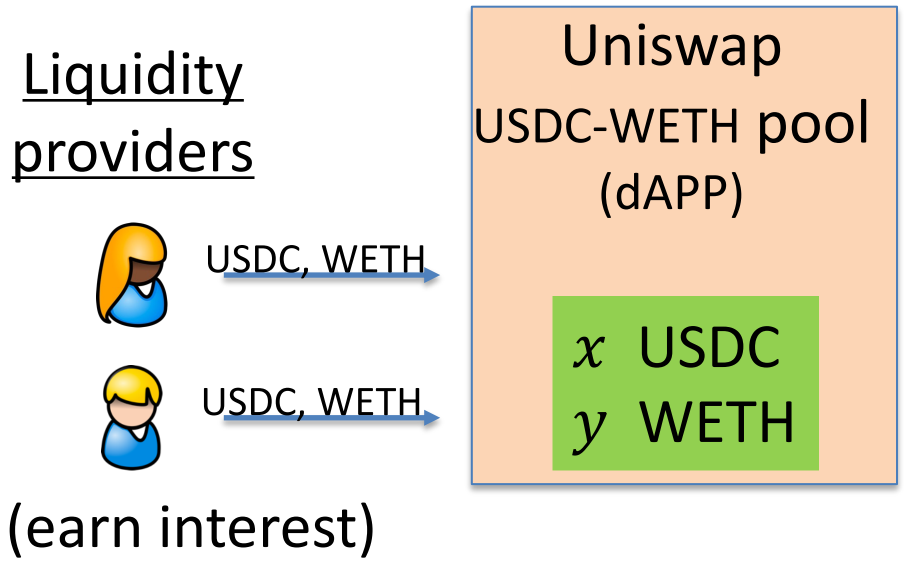
Benefits: Gas-efficient, accessible to contracts, easy to bootstrap

Automated Market Maker

Goal: People want to exchange **USDC ⟺ WETH**

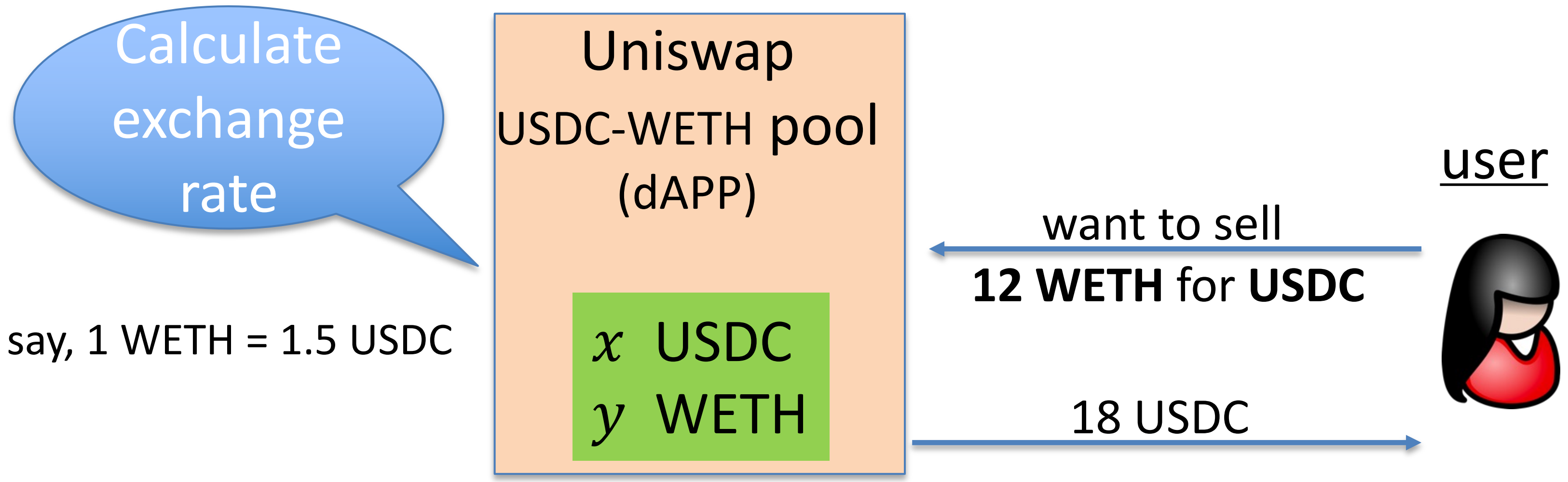
stable

volitile



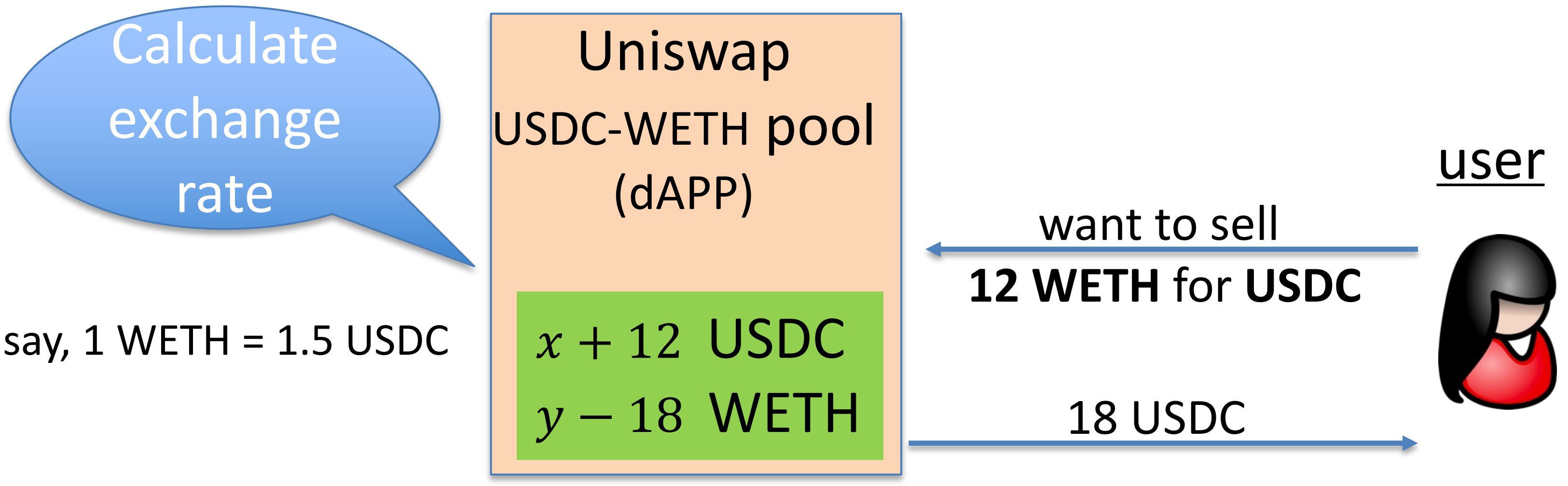
Automated Market Maker

Goal: People want to exchange **USDC ⟺ WETH**



Automated Market Maker

Goal: People want to exchange **USDC ⟺ WETH**



updated pool state

How to determine exchange rate?

Pool has (x units of X) and (y units of Y)

Def: **marginal price.**

Suppose Alice sent dx (an infinitesimal) amount of X to pool; and the pool sent back dy amount of Y.

(dx, change in X is positive; dy, change in Y is negative)

Then the **marginal price** is defined as $p = -dy/dx$ (>0)

The price of a small amount Y in units of X

How to determine exchange rate?

A reasonable goal for the pool to maintain:

(value of X in pool) = (value of Y in pool)

Let's use the marginal price p to estimate value of assets in pool:

- (value of X in pool) in units of Y: $p \cdot x$
- (value of Y in pool) in units of Y: y

So, goal above requires: $p \cdot x = y \quad \Rightarrow \quad p = y/x$

Plugging in the def for p gives: $\boxed{-\frac{dy}{dx} = y/x}$

How to determine exchange rate?

The diff. eq. $\boxed{-\frac{dy}{dx} = y/x}$ has a unique solution:

$$\boxed{y = \frac{k}{x}\ , \quad \text{for a constant } k \in \mathbb{R}}$$

indeed:

$$\left(-\frac{dy}{dx} = \frac{k}{x^2} = \frac{1}{x} \cdot \frac{k}{x} = \frac{y}{x} \right)$$

or equivalently, the pool must maintain: $\boxed{x \cdot y = k}$

… the famous constant product formula

So what does $x \cdot y = k$ mean ??

The constant product market maker:

- Say: $x = 10$ WETH, $y = 12$ USDC

  10 × 12 = 120

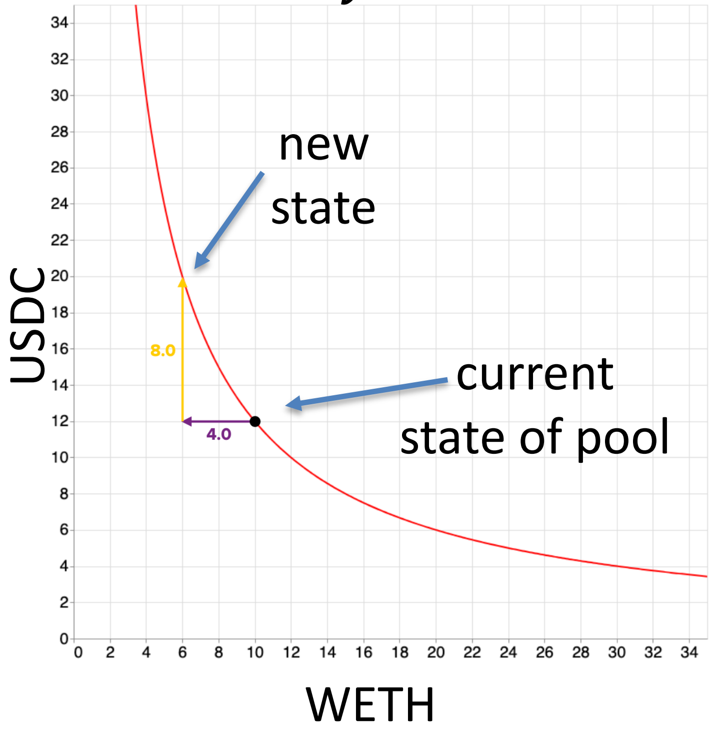
- Alice wants to buy 4 WETH from pool

  $x \rightarrow x - 4 = 6$

  To maintain $x * y = 120$ Alice needs to send 8 USDC to pool

  $y \rightarrow y + 8 = 20$

$x \cdot y = 120$



More generally: Uniswap v2

$x \cdot y = k$; Alice wants to buy $\Delta x \in (0, x)$ from pool.

How much Δy should she pay?

$(x - \Delta x) \cdot (y + \Delta y) = k \quad \Rightarrow \quad \boxed{\Delta y = \frac{y \cdot \Delta x}{x - \Delta x}}$ (solve for Δy and simplify)

But liquidity providers (LP's) take a fee $\phi \in [0,1]$ (say ϕ=0.9997)

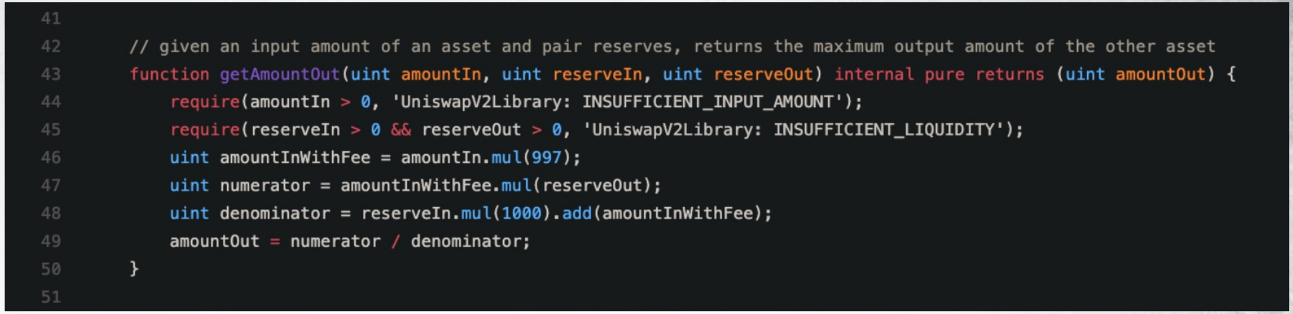
Alice pays Δy: pool gets $\phi \Delta y$, LP's get $(1 - \phi)\Delta y$

so: $(x - \Delta x) \cdot (y + \phi\Delta y) = k \quad \Rightarrow \quad \boxed{\Delta y = \frac{1}{\phi} \cdot \frac{y \cdot \Delta x}{x - \Delta x}}$

Buy and sell equations

Selling x for y $(x \rightarrow x + \Delta x)$

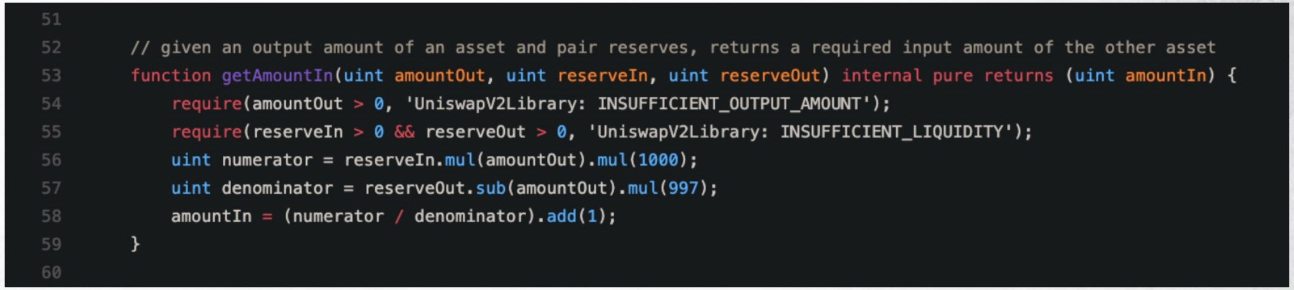
$$\boxed{\Delta y = \frac{y\phi\Delta x}{x + \phi\Delta x}}$$



```
    // 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;
    }
```

Buying x for y $(x \rightarrow x - \Delta x)$

$$\boxed{\Delta y = \frac{1}{\phi} \cdot \frac{y\Delta x}{x - \Delta x}}$$



```
    // given an output amount of an asset and pair reserves, returns a required input amount of the other asset
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) internal pure returns (uint amountIn) {
        require(amountOut > 0, 'UniswapV2Library: INSUFFICIENT_OUTPUT_AMOUNT');
        require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
        uint numerator = reserveIn.mul(amountOut).mul(1000);
        uint denominator = reserveOut.sub(amountOut).mul(997);
        amountIn = (numerator / denominator).add(1);
    }
```

UniswapV2Library.sol

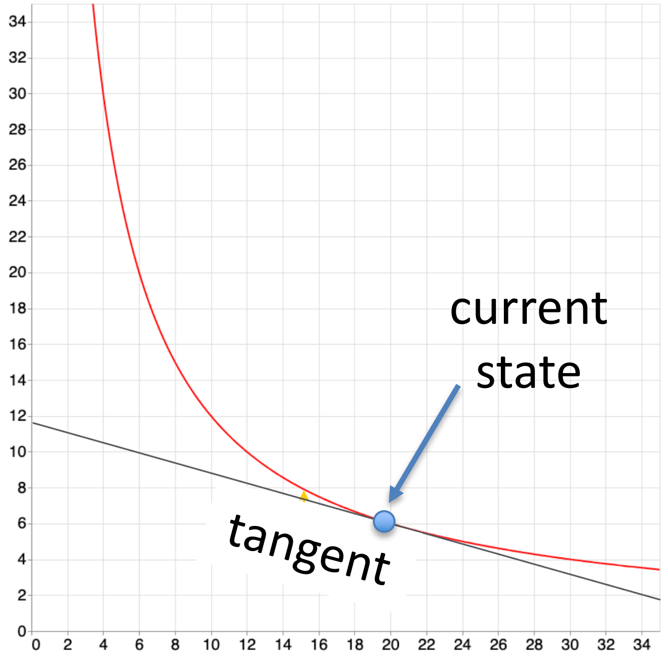
gas efficient calculations

The marginal price as a tangent

$$x \cdot y = k \quad \Rightarrow \quad y = k/x$$

The marginal price: $p = -\frac{dy}{dx} = \frac{\mathrm{y}}{\mathrm{x}}$

$\Rightarrow \quad -p$ is the slope of the tangent at the current state



A feature: automatic price discovery (assume ϕ=1)

Thm: the marginal price y/x converges to the market exchange rate

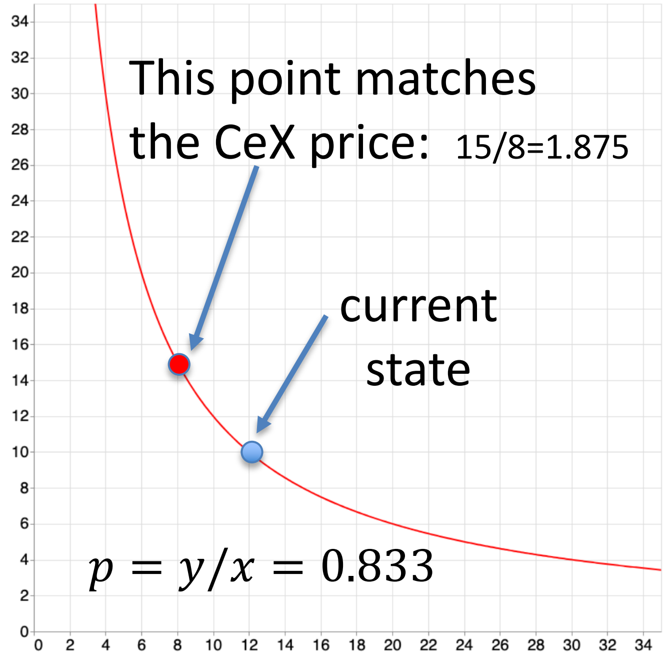
Proof by example: say, $x = 12, \;\; y = 10$

$\Rightarrow$ marginal price $p = y/x = 0.833$

Suppose a CeX offers a different price

$p_{market} = 1.875$

$\Rightarrow$ arbitrage opportunity!



A feature: automatic price discovery (assume ϕ=1)

Thm: the marginal price y/x converges to the market exchange rate

Arbitrageur will do:

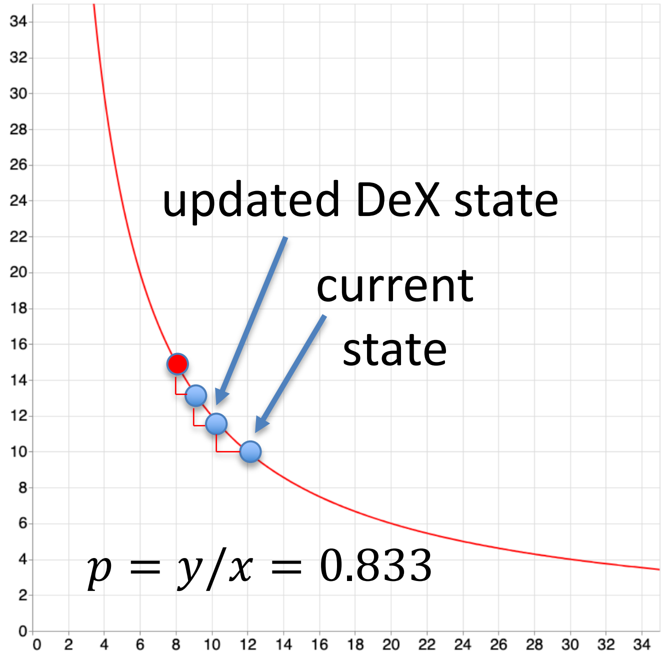
- borrow 1 token of type Y from Compound
- send 1 Y to DeX, get 0.77 X tokens back
- send 0.77 X to CeX, get $077 \times 1.875 = 1.44$ Y
- repay 1 Y to Compound, **keep 0.44 Y** !!

Iterate until DeX marginal price = CeX price

⇒ Arb. is providing a service, and making a profit

Where did the 0.44 Y come from? Who lost money?

Answer: LP's lost … we will see why



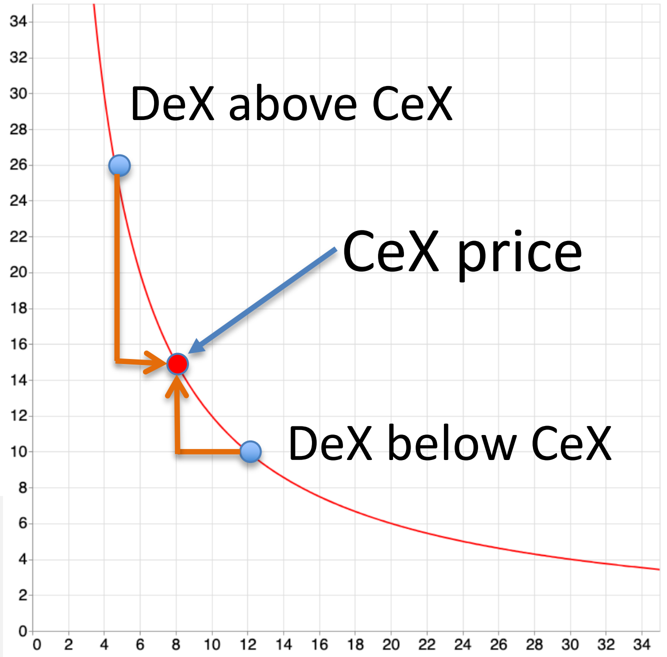
A feature: automatic price discovery (assume ϕ=1)

Thm: the marginal price y/x converges to the market exchange rate

To summarize:

- DeX state is below market rate
  $\Rightarrow$ arbitrageurs will move DeX up
- DeX state is above market rate
  $\Rightarrow$ arbitrageurs will move DeX down

DeX marginal price matches market price, without ever being told the market price !!



Problem 1: Slippage

Slippage:

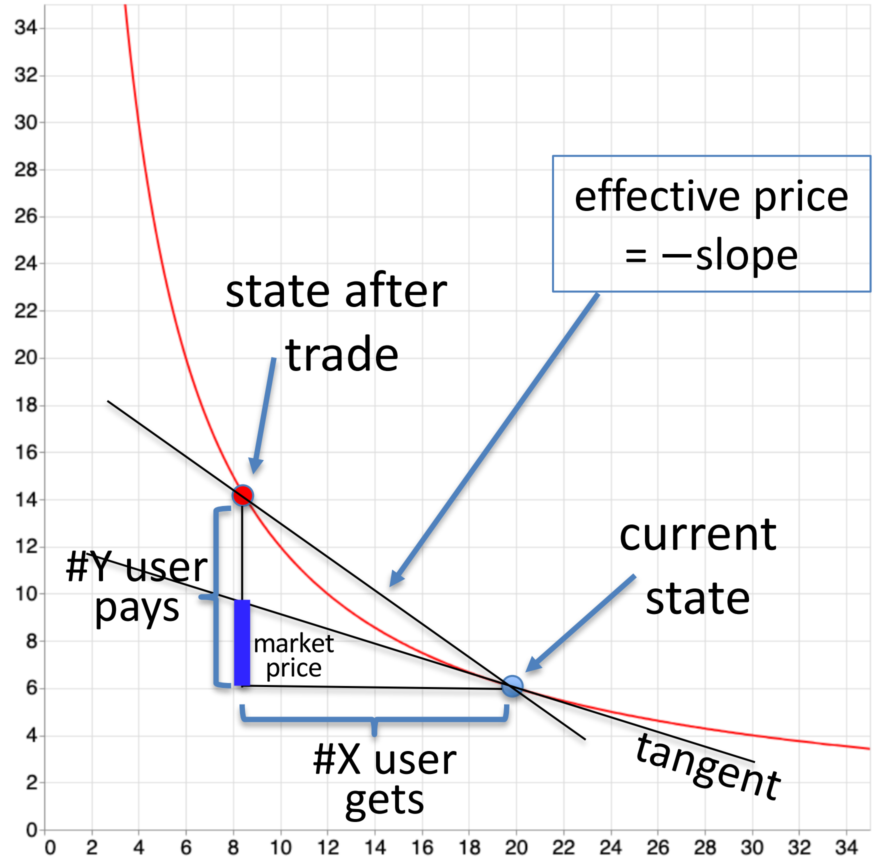
- the larger the trade, the worse the exchange rate is for the user

market price = −tangent slope

⇒ #Y user should pay = blue line

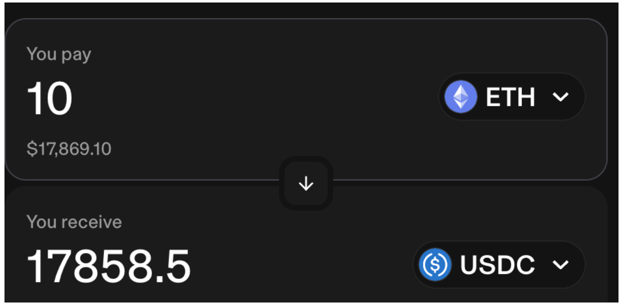
… but user pays more

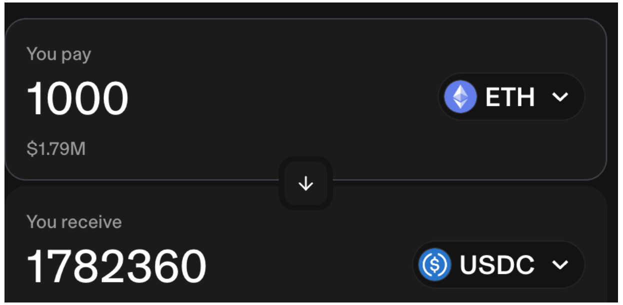
(uniswap bounds slippage at 0.5%)



Slippage: an example

$$\Delta y = \frac{y \cdot \Delta x}{x + \Delta x}$$


You pay
10
ETH
$17,869.10
You receive
17858.5
USDC


You pay
1000
ETH
$1.79M
You receive
1782360
USDC

1785.5 USDC/ETH $>$ **1782.36** USDC/ETH

Note: if $\Delta y = y$ (Alice wants to buy entire USDC pool)

then $\frac{\Delta x}{x+\Delta x} = 1$, so Alice must send $\Delta x = \infty$ ETH.

Alice cannot do $\Rightarrow$ pool will never run out of X or Y tokens.

Problem 2: the sandwich attack

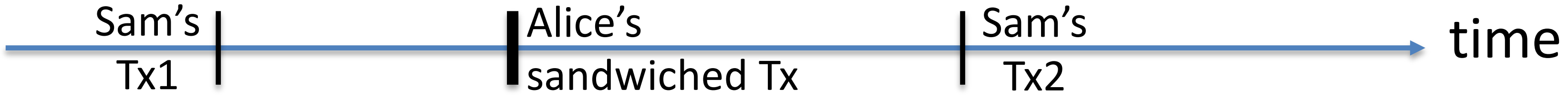
Consider the WETH-USDC pool:

- User Alice submits a Tx to sell Δx USDC to pool $(x \rightarrow x + \Delta x)$
- Normally, she gets back $\Delta y = y\,\Delta x/(x + \Delta x)$ WETH

Sam monitors the mempool, and sees Alice's Tx.

He immediately submits two of his own Tx:

- Tx1: Sam sells 5 USDC to pool, gets back s WETH (high tip)
- Tx2: Sam sells s WETH to pool, gets back s' USDC (low tip)



Problem 2: the sandwich attack

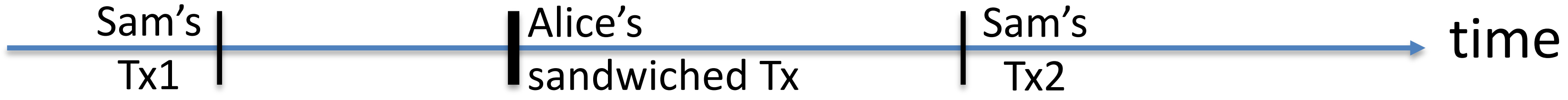
Now, Alice gets back $\Delta y' = \frac{(y-s)\Delta x}{(x+5)+\Delta x} < \Delta y$ WETH

$\Rightarrow$ she gets a worse exchange rate because of Sam's Tx1

$\Rightarrow$ For Sam, $s' > 5$ so he made $(s' - 5)$ USDC off of Alice

This is a frontrunning attack:

- Also happens in regular financial markets (see flash boys)
- We will come back to this when we discuss MEV (and show how to prevent)



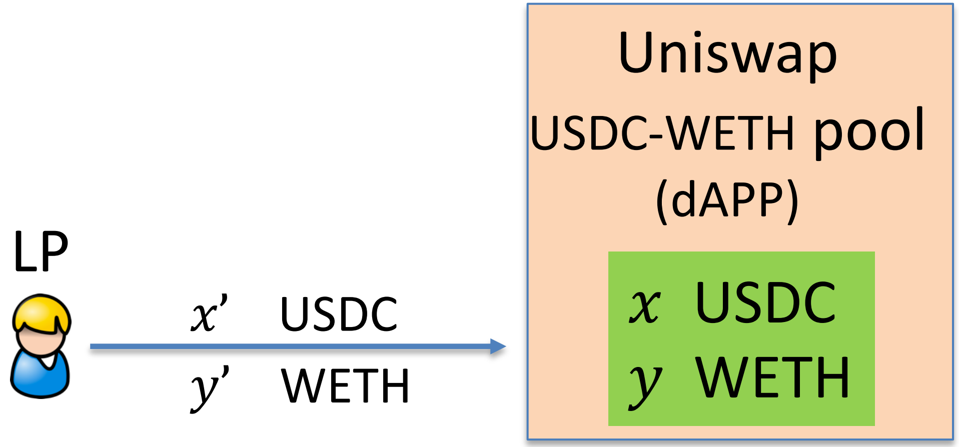
Incentives for liquidity providers

Recall: liquidity providers (LP's)

When LP contributes to pool: $y'/x' = y/x$

$\Rightarrow$ does not change marginal price of pool, namely $\frac{y+y'}{x+x'} = \frac{y}{x}$

$\Rightarrow$ LP "owns" $\frac{x'}{x+x'}$ of the pool

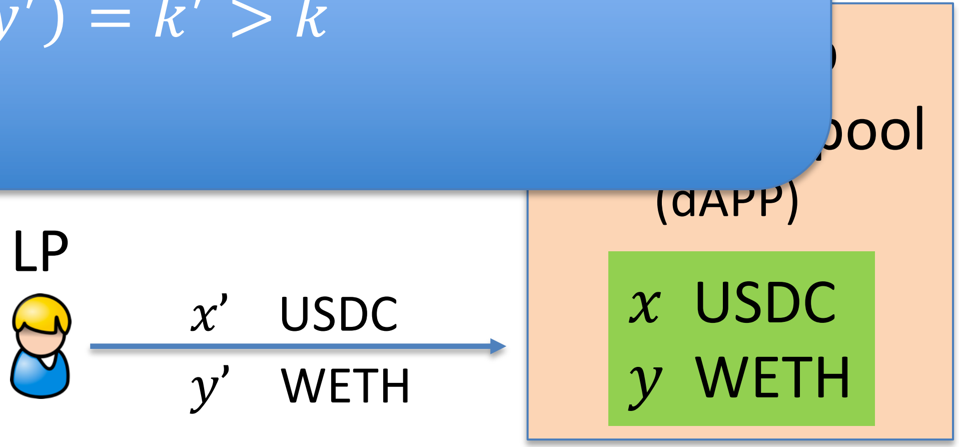


Recall: liquidity providers (LP's)

When LP contributes to pool: $y'/x' = y/x$

Note: LP contribution changes the constant k:

$$(x + x')(y + y') = k' > k$$

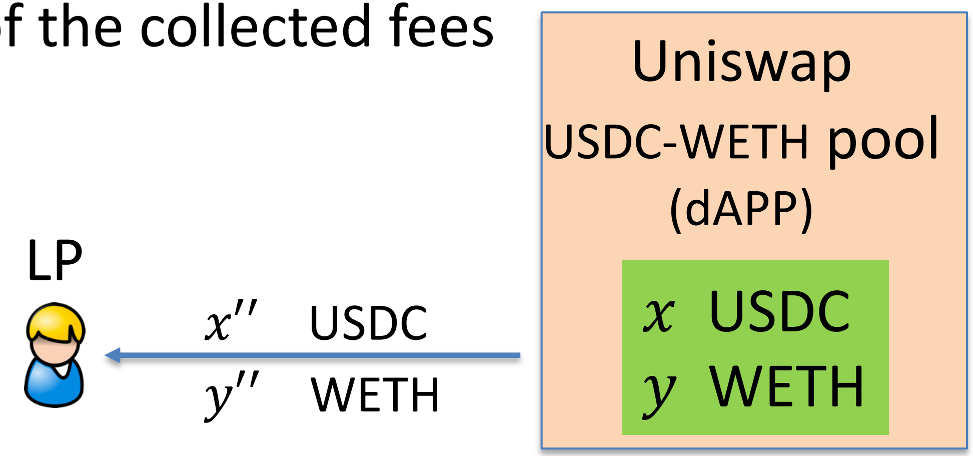


LP withdrawal

(x, y) is the current state of the pool. LP owns a β fraction of pool.

When LP withdraws from pool they get:

- (x'', y'') of (USDC,WETH) where $y''/x'' = y/x$ and $x''/x = \beta$.
- LP also receives a β fraction of the collected fees



Should LP's contribute to pool?

Suppose LP has (x', y') of (USDC,WETH).

- Should LP contribute them to USDC-WETH pool?
- Or is there a more profitable strategy for the LP?

AMM strategy:

contribute (x', y') to USDC-WETH pool at time t_0,

withdraw (x'', y'') from pool at time $t_1 > t_0$.

Loss vs. Hold (divergence loss)

HOLD Strategy: LP holds (x', y') of (USDC,WETH) between time t_0 and t_1.

Let $P(t)$ be the market price of WETH/USDC at time t.

Fact: if $P(t_0) = P(t_1)$ then at time t_1, LP's portfolio value is:

HOLD strategy: $P(\mathrm{t}_1) \cdot x' + y'$ WETH.

AMM strategy: $P(\mathrm{t}_1) \cdot x' + y'$ **+ fees** WETH.

Fact: Let $\Delta = P(t_1)/P(t_0)$. At time t_1, LP's portfolio value:

HOLD strategy: $P(\mathrm{t}_1) \cdot x' + y'$ WETH.

AMM strategy: $[P(\mathrm{t}_1) \cdot x' + y'] \cdot \boldsymbol{M}(\Delta - 1)$ **+ fees** WETH,

Loss vs. Hold

where $\boldsymbol{M}(0) = 0$ and $\boldsymbol{M}(z)$ increases with $|z|$.

Loss vs. Hold (divergence loss)

HOLD Strategy: LP holds (x', y') of (USDC,WETH) between time t_0 and t_1.

(1) Loss-vs-Hold increases as $\Delta = P(t_1)/P(t_0)$ deviates from 1.

$\Rightarrow$ the greater the change in price, the greater the LP's losses

(1) AMM vs. HOLD strategy makes sense only if fees > Loss-vs-Hold.

$\Rightarrow$ determines the pool's fee needed to attract liquidity

(3) Who gets the LP's losses? Arbitrageurs

Loss vs. Rebalancing (LVR)

Rebalancing Strategy:

- LP maintains its portfolio outside of the DeX
- LP does the same rebalancing on its portfolio as the DeX, but it does so by trading with a CeX.

A strategy that more accurately predicts LP's losses when providing liquidity to DeX

$\Rightarrow$ more accurately determines the fee needed to attract liquidity

https://arxiv.org/pdf/2208.06046.pdf

Loss vs. Rebalancing (LVR)

LVR meta theorem:

At periods of high exchange rate volatility, the rebalancing strategy greatly outperforms the investment in a DeX as an LP, unless fees are “very” high

$\Rightarrow$ At periods of high volatility, LPs withdraw funds from the pool (unless the pool charges very high fees)

Other functions

Constant Function Market Maker (CFMM)

Pool maintains $f(x,y) = k$ for some function $f(\cdot,\cdot)$

Examples:

$\text{val}(X) = \text{val}(Y)$

- **constant product**: $f(x,y) = x \cdot y$
- **constant weighted product**: $f(x,y) = x^{w_x} \cdot y^{w_y}$
  - Maintains an imbalanced portfolio $\text{val}(X)/\text{val}(Y) = w_x/w_y$
- **constant sum**: $f(x,y) = w \cdot x + y$ for some constant w.
  - marginal price is always $-dy/dx = w$ (never changes)
  - used when X-to-Y exchange rate does not change

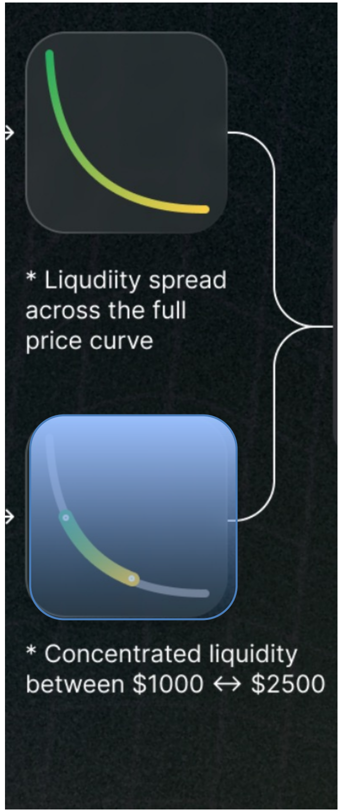
Uniswap v3: concentrated liquidity

In v2, LP’s liquidity is used on the entire price range.

In v3, LP can specify a price range where their liquidity will be used

⇒ protects LP from price swings. Results in a deeper pool when price is in the allowed range.

https://uniswap.org/whitepaper-v3.pdf


* Liqudiity spread across the full price curve
* Concentrated liquidity between $1000 ↔ $2500

Uniswap v4: hooks

Enables pool creator to specify hooks at pool creation time:

- code that executes at certain points during trade:
  e.g., BeforeSwap, AfterSwap hooks

Hooks enable: (more examples here)

- Sandwich protection (Angstrom)
- Dynamic trade fee (ϕ) based on state of the pool
- Limit orders (e.g., acceptable price for the next 24 hours)
- More sophisticated pricing strategies (e.g., average over last hour)

https://blog.uniswap.org/uniswap-v4

Summary: AMMs

- AMM is implemented as a simple smart contract
- Automatic price discovery (no off-chain oracles)
- No dependence on a central point of control
- Fully composable with other dAPPs

END OF LECTURE

Next lecture: MEV