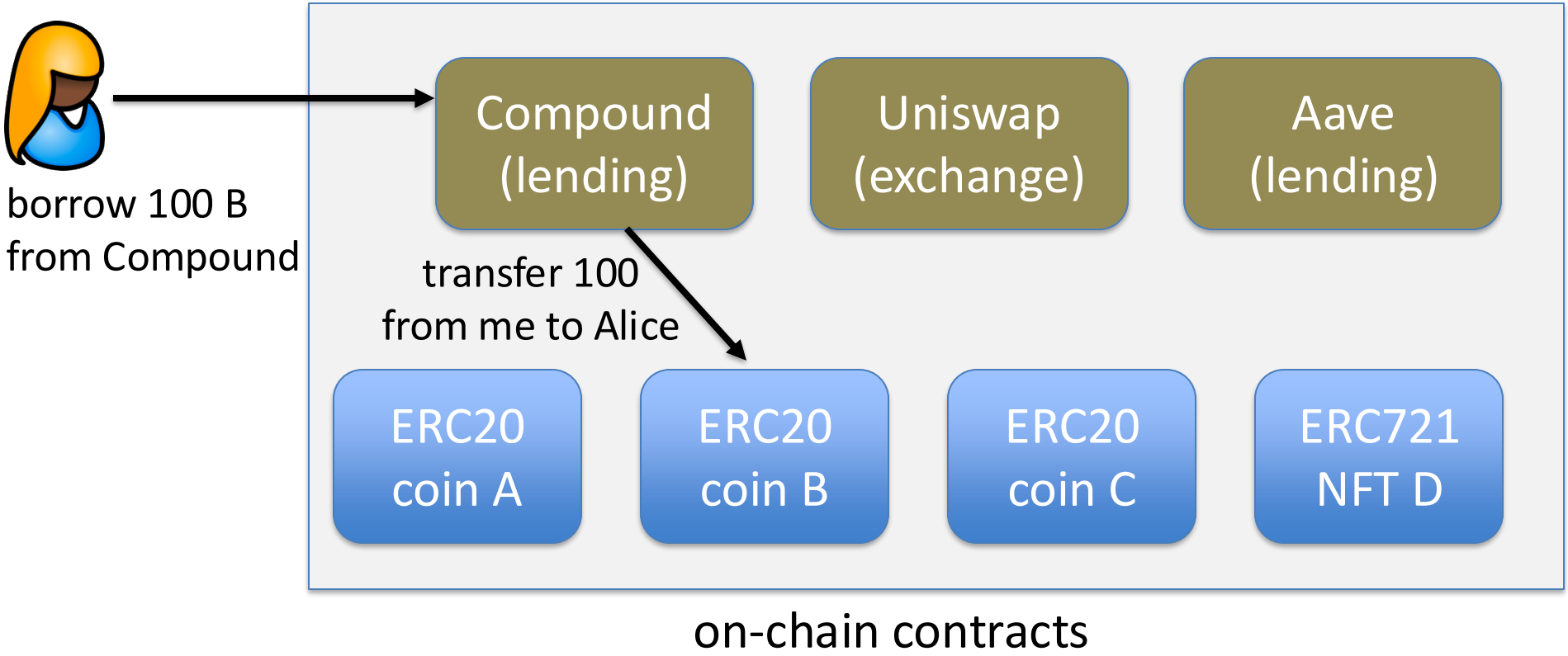
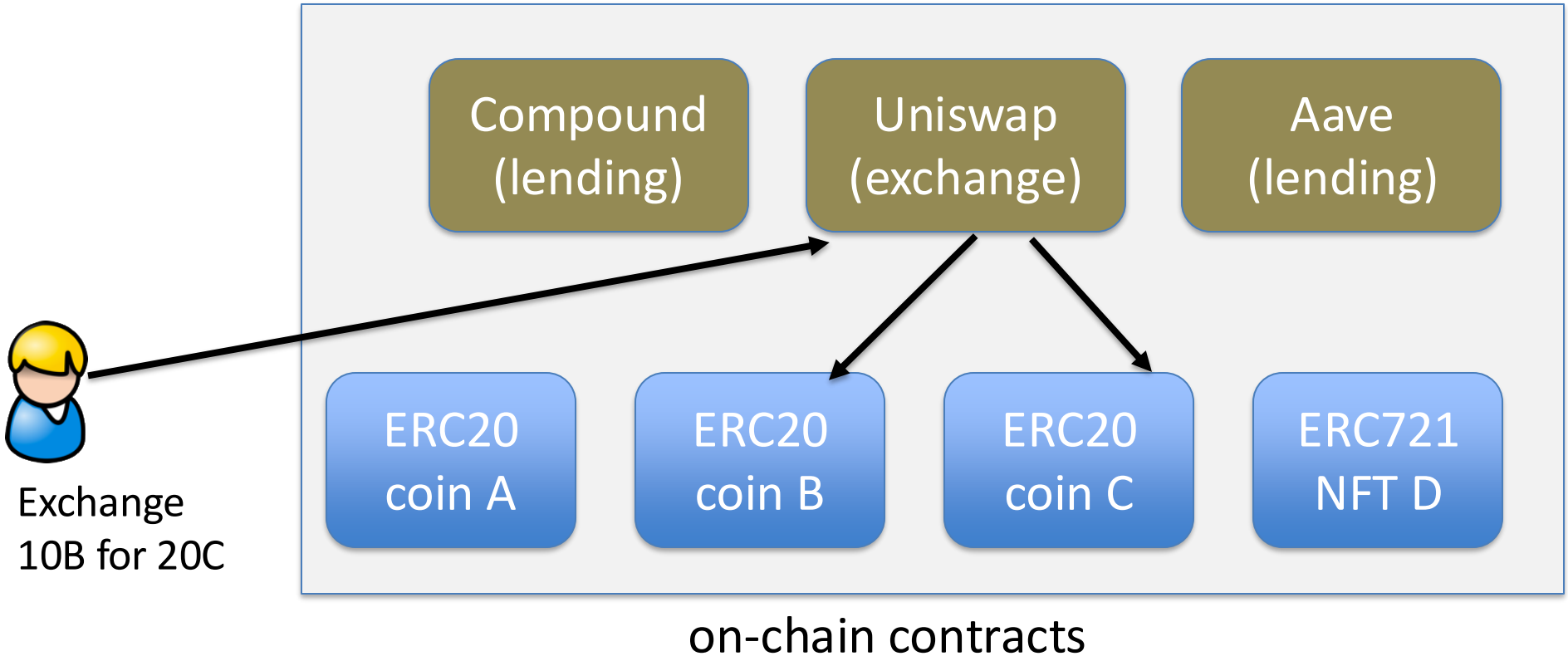
DeFi Lending Protocols, Swaps, DEXes

Slides from Dan Boneh

The world of DeFi



The world of DeFi



DeFi app #1: Stablecoins

Stable Coins

A cryptocurrency designed to trade at a fixed price

- Examples: **1 coin = 1 USD**, 1 coin = 1 EUR, 1 coin = 1 USDX

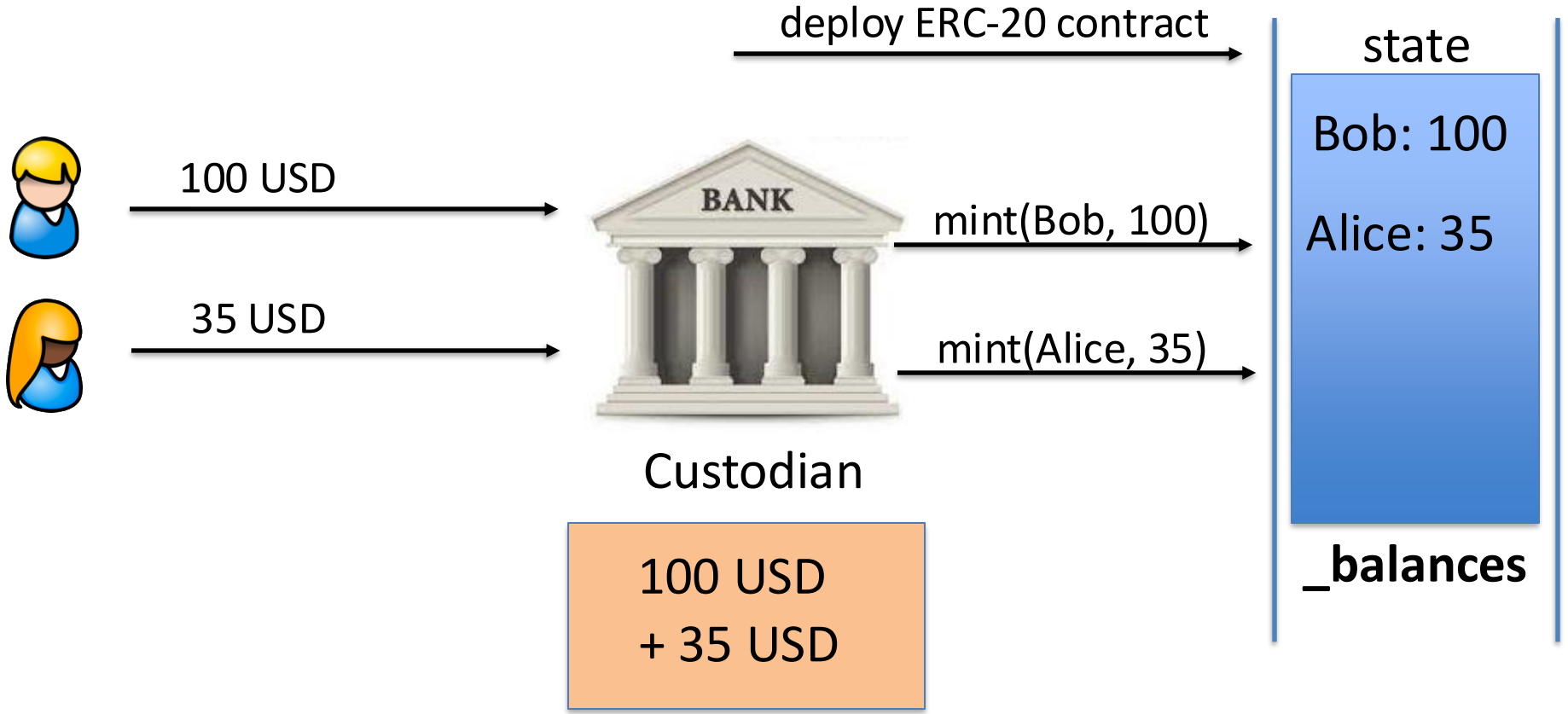
Goals:

- Integrate real-world currencies into on-chain applications
- Enable people without easy access to USD,
  to hold and trade a USD-equivalent asset

Types of stable coins

	centralized	algorithmic
collateralized	custodial stablecoins (USD Coin)	synthetics (DAI, RAI)
Un(der)collateralized	central bank (digital) currency	Undercollateralized stablecoins

Custodial stablecoins: minting



Custodial stablecoins: transfers

pay Carol 15$: transfer(Bob → Carol, 15)

(and gas fee)

Transfers are done on-chain
(custodian is not involved)

135 USD

Bob: ~~85~~ 50

Alice: 35

Carol: 15

_balances

Custodial stablecoins: withdrawal

withdraw 60 USD

60 USD

Custodian

135 USD

burn(Bob, 60)

Bob: 85

Alice: 35

Carol: 15

_balances

Two Examples

	Coins issued	24h volume
USDC	33 B	5.3 B
USDT	110 B	29.3 B

Some issues

Custodian keeps treasury in a traditional bank

- Must be audited to ensure treasury is available
- Earns interest on deposits

Custodian has strong powers:

- Can freeze accounts / refuse withdrawal requests
- Custodian can remove funds from user balances

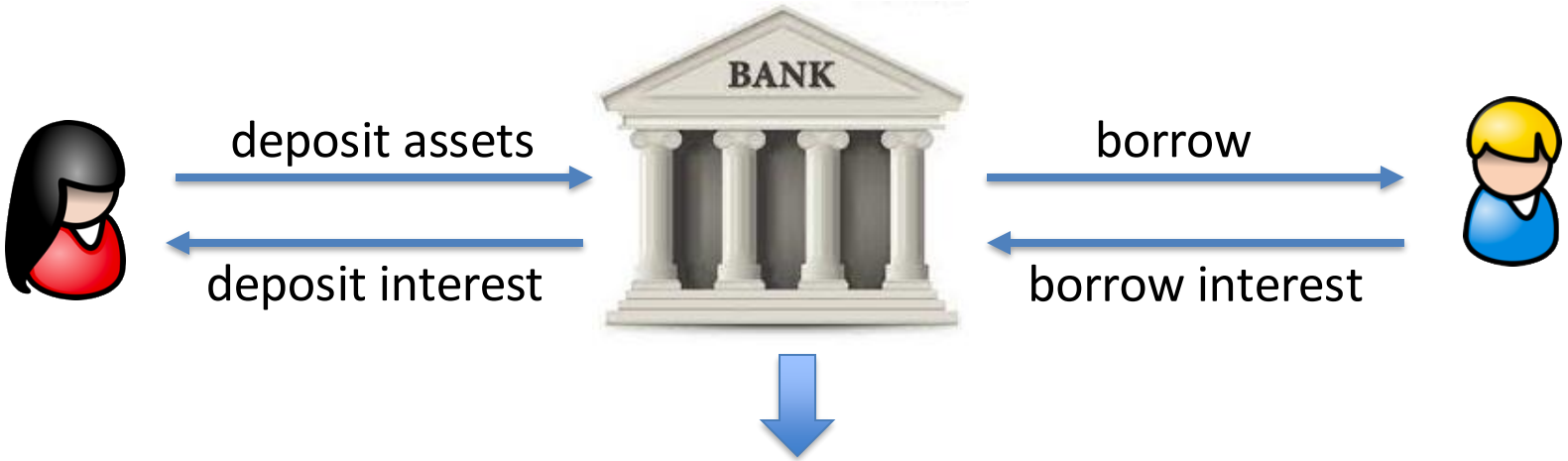
DeFi app #2: Lending Protocols

Goal: explain how decentralized lending works

This is not investment or financial advice

The role of banks in the economy

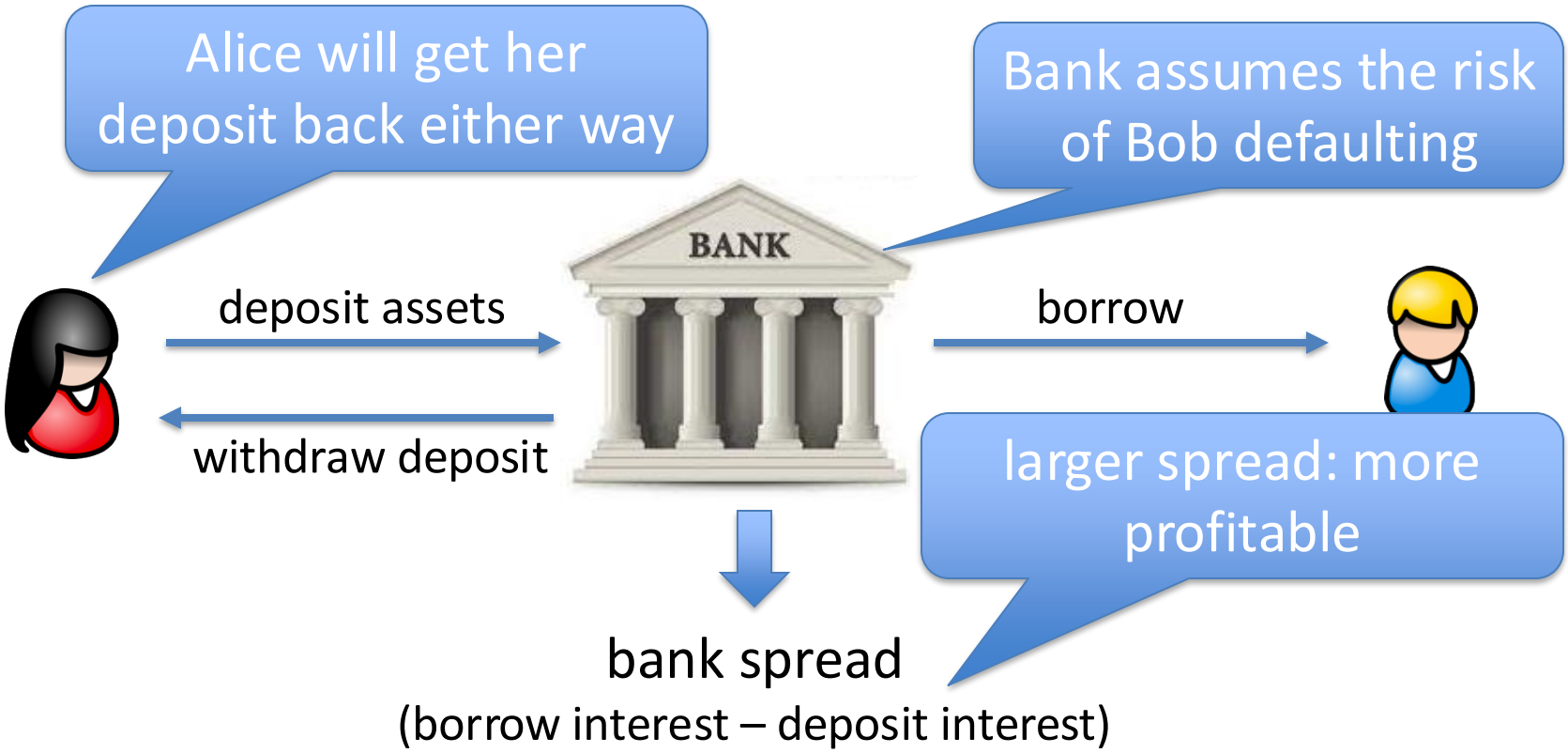
Banks bring together lenders and borrowers



bank spread

(borrow interest – deposit interest)

The role of banks in the economy



Crypto: CeFi lending

Same as with a traditional bank:

Alice gives her assets to the CeFi institution to lend out to Bob

Crypto: CeFi lending

From Wikipedia, the free encyclopedia

BlockFi was a digital asset lender founded by Zac Prince and Flori Marquez in 2017.[1] It was based in Jersey City, New Jersey.[2][3] It was once valued at $3 billion.[4]

In July 2022, it was announced that the cryptocurrency exchange FTX made a deal with an option to buy BlockFi for up to $240 million. The deal included a $400 million credit facility for the company.[5][6]

In November 2022, after the declaration of bankruptcy by FTX, the company halted withdrawals on its platform.[7] The firm disclosed "significant exposure" to FTX on November 14. The next day, the *Wall Street Journal* reported that BlockFi would likely file for bankruptcy, which was reaffirmed by reports compiled by Bloomberg News.[8][9][10] On November 28, BlockFi confirmed the reports and officially filed for Chapter 11 bankruptcy protection[11] with more than 100,000 creditors, according to filings.[12][13] In March 2023, following the collapse of Silicon Valley Bank, the U.S. Trustee watchdog overseeing BlockFi's bankruptcy revealed that BlockFi had around $227 million in uninsured funds at the bank.[14] BlockFi's CEO Zac Prince testified in the trial of Sam Bankman-Fried, and stated that he thought BlockFi would not have gone bankrupt if it had not lost access to its funds in the FTX collapse.[15] In October 2023, BlockFi emerged from Chapter 11 bankruptcy and announced that it would begin winding down all of its remaining operations and assets post-bankruptcy. Over the next few months

BlockFi

BlockFi

Industry	Cryptocurrency
Founded	2017; 8 years ago
Founders	Flori Marquez Zac Prince
Fate	Filed for Chapter 11 bankruptcy protection in November 2022
Headquarters	Jersey City, New Jersey, U.S.
Website	blockfi.com

The role of collateral

(1 ETH = 100 UNI)

CeFi's concern: what if Bob defaults on loan?

$\Longrightarrow$ CeFi will absorb the loss

Solution: require Bob to lock up collateral

collateral

deposit 500 UNI

Borrow 1 ETH

debt position:

+ 500 UNI
− 1 ETH

interest deducted from collateral

over collateralized loan

The role of collateral

Several things can happen next:

(1) **Bob repays loan**

(1 ETH = 100 UNI)

repay 1 ETH

redeem UNI collateral
(minus interest)

debt position:

+ 50 UNI
− 1 ETH

The role of collateral

Several things can happen next:

(1) Bob repays loan

(2) Bob defaults on loan

(1 ETH = 100 UNI)

Ok, I'll keep
(100 + penalty) UNI

I can't repay 1 ETH

redeem remaining UNI collateral
(400 – interest – penalty) UNI

debt position:

+ 5[illegible] UNI
– 1 [illegible]

The role of collateral

(1 ETH = 400 UNI)

Several things can happen next:

(1) Bob repays loan

(2) Bob defaults on loan

(3) Liquidation: value of loan increases relative to collateral

I need to liquidate your collateral (and charge a penalty = 20 UNI)

debt position:

+ 80 UNI
− 0 ETH

lender needs to liquidate **before** value(debt) > value(collateral)

Terminology

Collateral: assets that serve as a security deposit

Over-collateralization: borrower has to provide
value(collateral) > value(loan)

Under-collateralization: *value(collateral) < value(loan)*

Liquidation:

collateral factor

if value(debt) > 0.6 × value(collateral)

then collateral is liquidated until inequality flips

(liquidation reduces both sides of the inequality)

Collateral factor

CollateralFactor $\in [0,1]$

- Max value that can be borrowed using this collateral
- High volatility asset $\Longrightarrow$ low collateral factor
- Relatively stable asset $\Longrightarrow$ higher collateral factor

Health of a debt position

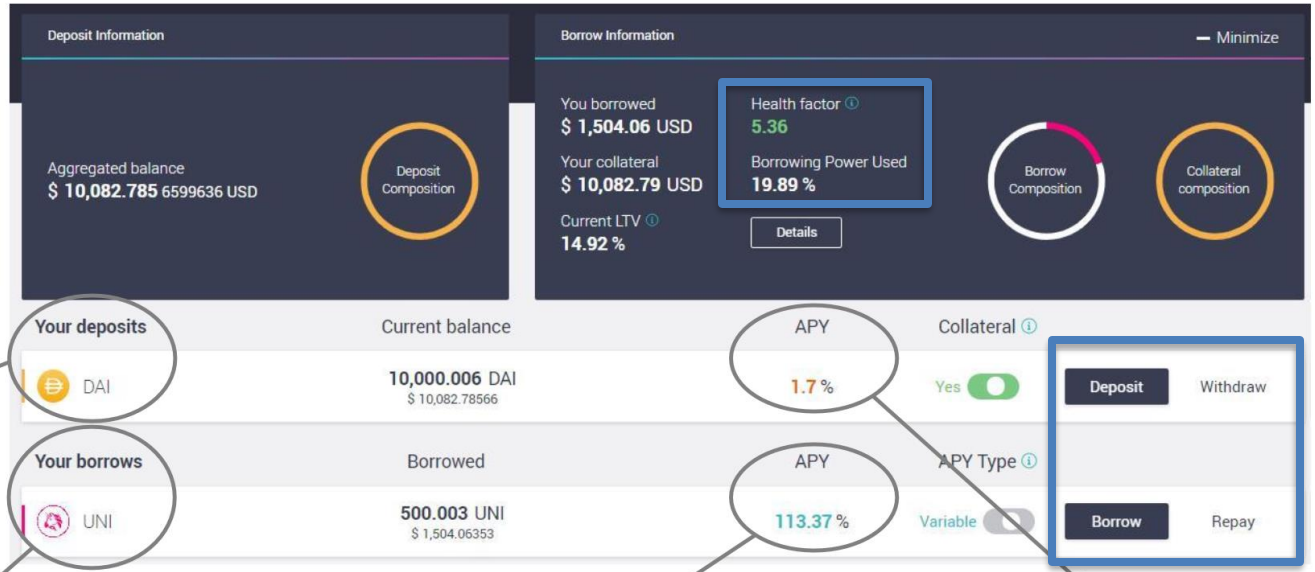
$$\text{BorrowCapacity} = \sum_i \text{value}(\text{collateral}_i) \times \text{CollateralFactor}_i$$

(in ETH)

$$\text{health} = \frac{\text{BorrowCapacity}}{\text{value}(\text{TotalDebt})}$$

helath < 1 $\implies$ triggers liquidation until (health ≥ 1)

Example: Aave dashboard (a DeFi lending Dapp)



DAI is deposited as collateral

UNI is borrowed

The borrowing interests the borrower needs to pay

In Aave, the collteral is also lent out. Hence the borrower can also earn interests.

actions

Credit: Arthur Gervais

Compound | Dashboard | Markets | Extensions | Vote | Connect Wallet

← Markets

USDC • Ethereum

Total Collateral
$898.08M

Total Borrowing
$409.59M

Market Stats

Total Earning	Available Liquidity	Total Reserves	Collateralization	Oracle Price
$522.89M	$128.47M	$15.21M	127.65%	$1.00

Market Rates Net of Rewards

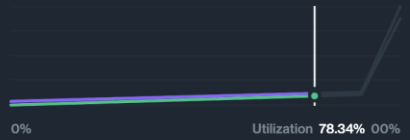
Net Borrow APR
4.50%

Net Earn APR
3.90%

Interest Rate Model

Borrow APR
4.76%

Earn APR
3.70%



Collateral Assets

Asset	Total Supply	Reserves	Oracle Price	Collateral Factor	Liquidation Factor	Liquidation Penalty
Wrapped Bitcoin WBTC	$596.92M	$0.91	$94,657.35	80%	85%	10%
Ether ETH	$142.28M	$0.82	$1,795.67	83%	90%	5%
Lido Wrapped Staked ETH wstETH	$91.09M	$0.29	$2,154.97	82%	87%	8%
Coinbase Wrapped BTC cbBTC	$36.58M	$0.10	$93,133.39	80%	85%	5%
Compound COMP	$17.92M	$0.14	$40.81	50%	70%	25%
Chainlink LINK	$5.69M	$0.62	$14.31	73%	79%	17%
tBTC v2 tBTC	$4.74M	$0.00	$93,824.76	76%	81%	10%
Uniswap UNI	$2.86M	$0.09	$5.31	68%	74%	17%

Why borrow ETH?

If Bob has collateral, why can't he just buy ETH?

- Bob may need ETH but he might not want to sell his collateral (e.g., WBTC or UNI)
- As an investment strategy: using UNI to borrow ETH gives Bob exposure to both

The problem with CeFi lending

Users must trust the CeFi institution:

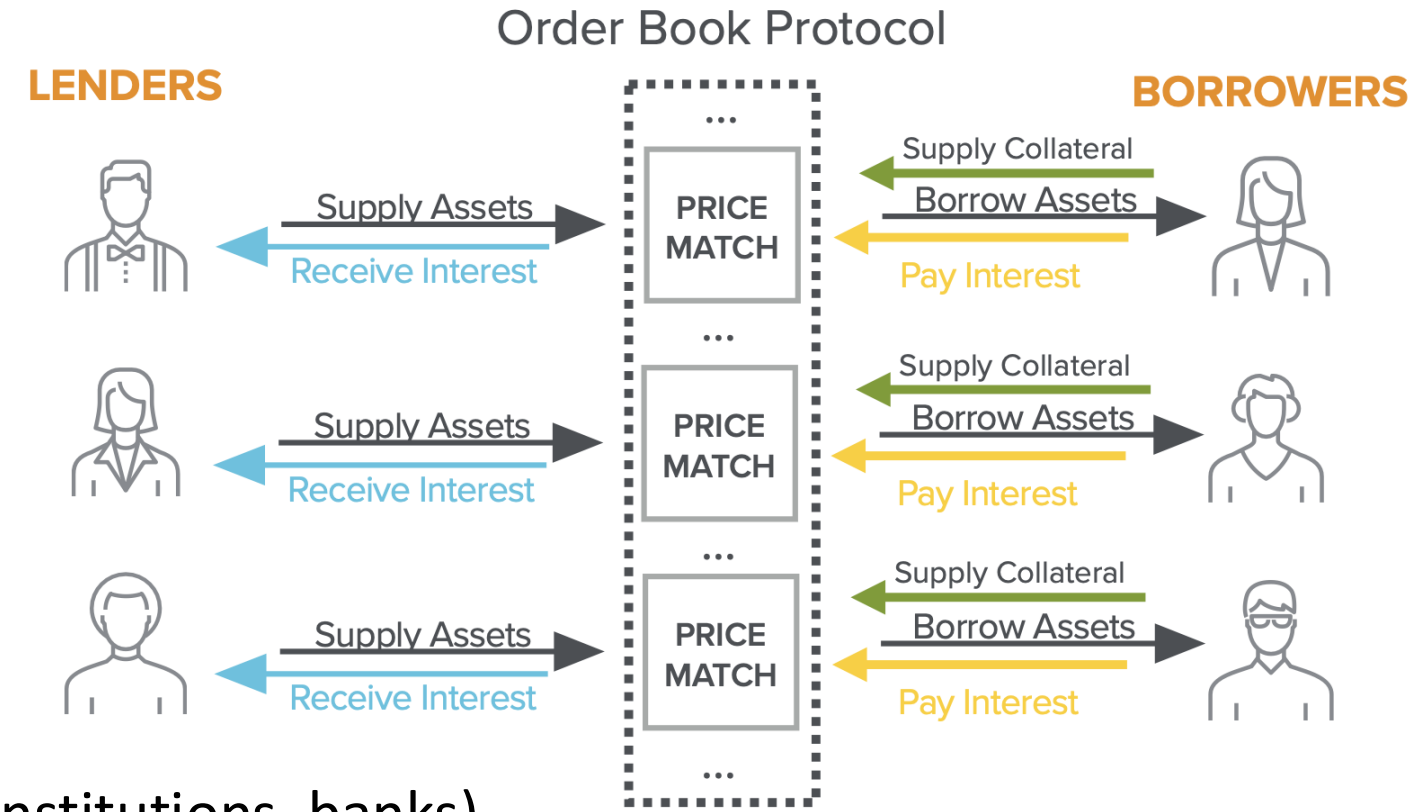
- Not to get hacked, steal assets, or miscalculate
- This is why traditional finance is regulated

- Interest payments go to the exchange, not liquidity provider Alice

- CeFi fully controls spread (borrow interest – deposit interest)

DeFi Lending

A first idea: an order book Dapp



(large institutions, banks)

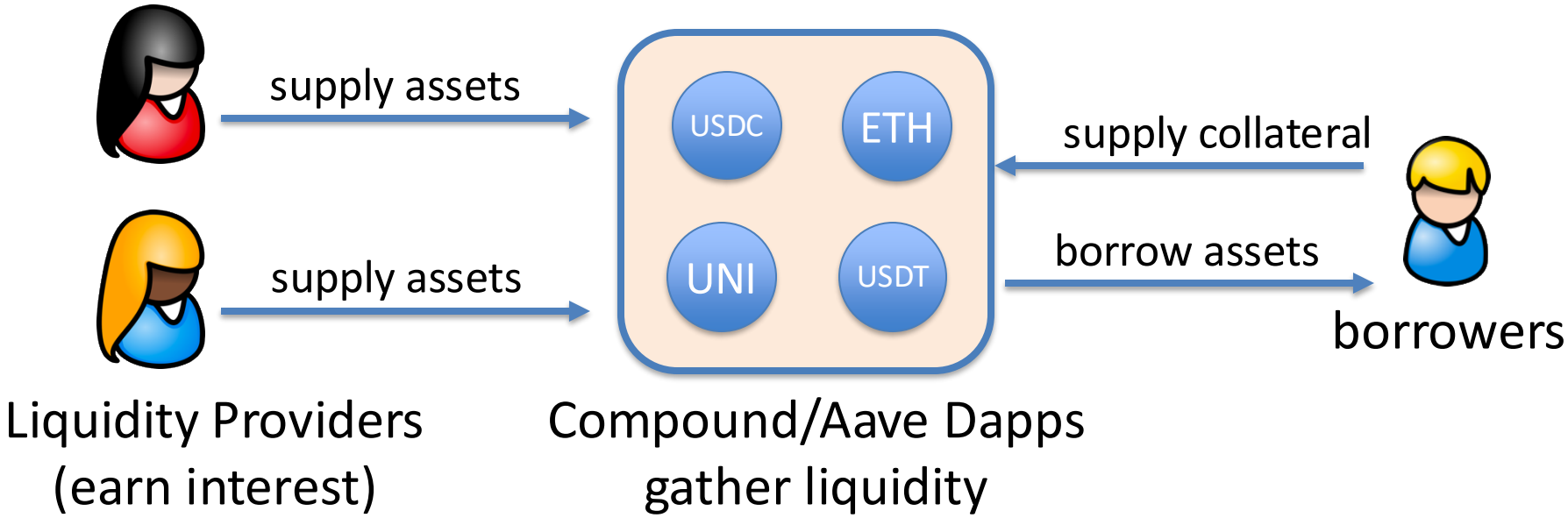
Credit: Eddy Lazzarin

Challenges

- **Computationally expensive**: matching borrowers to lenders requires many transactions per person (post a bid, retract if the market changes, repeat)
- **Concentrated risk**: lenders are exposed to their direct counterparty defaulting
- **Complex withdrawal**: a lender must wait for their counter-parties to repay their debts

A better approach: liquidity pools

Over-collateralized lending: Compound and Aave



Supply

The supply function transfers an asset to the protocol and adds it to the account's balance. This function can be used to **supply collateral, supply the base asset, or repay an open borrow** of the base asset.

If the base asset is supplied resulting in the account having a balance greater than zero, the base asset earns interest based on the current supply rate. Collateral assets that are supplied do not earn interest.

There are three separate methods to supply an asset to Compound III. The first is on behalf of the caller, the second is to a separate account, and the third is for a manager on behalf of an account.

Before supplying an asset to Compound III, the caller must first execute the asset's ERC-20 approve of the Comet contract.

Comet

```
function supply(address asset, uint amount)

function supplyTo(address dst, address asset, uint amount)

function supplyFrom(address from, address dst, address asset, uint amount)
```

- `asset`: The address of the asset's smart contract.
- `amount`: The amount of the asset to supply to Compound III expressed as an integer. A value of `MaxUint256` will repay all of the `dst`'s base borrow balance.
- `dst`: The address that is credited with the supplied asset within the protocol.
- `from`: The address to supply from. This account must first use the Allow method in order to allow the sender to transfer its tokens prior to calling Supply.
- `RETURN`: No return, reverts on error.

Withdraw or Borrow

The withdraw method is used to **withdraw collateral** that is not currently supporting an open borrow. Withdraw is **also used to borrow the base asset** from the protocol if the account has supplied sufficient collateral. It can also be called from an allowed manager address.

Compound III implements a minimum borrow position size which can be found as `baseBorrowMin` in the protocol configuration. A withdraw transaction to borrow that results in the account's borrow size being less than the `baseBorrowMin` will revert.

Comet

```
function withdraw(address asset, uint amount)

function withdrawTo(address to, address asset, uint amount)

function withdrawFrom(address src, address to, address asset, uint amount)
```

- `asset`: The address of the asset that is being withdrawn or borrowed in the transaction.
- `amount`: The amount of the asset to withdraw or borrow. A value of `MaxUint256` will withdraw all of the `src`'s base balance.
- `to`: The address to send the withdrawn or borrowed asset.
- `src`: The address of the account to withdraw or borrow on behalf of. The `withdrawFrom` method can only be called by an allowed manager.
- `RETURN`: No return, reverts on error.

Absorb

This function can be called by any address to liquidate an underwater account. It transfers the account's debt to the protocol account, decreases cash reserves to repay the account's borrows, and adds the collateral to the protocol's own balance. The caller has the amount of gas spent noted. In the future, they could be compensated via governance.

Comet

```
function absorb(address absorber, address[] calldata accounts)
```

- `absorber`: The account that is issued liquidator points during successful execution.
- `accounts`: An array of underwater accounts that are to be liquidated.
- `RETURN`: No return, reverts on error.

Liquidation

Historical DAI interest rate on Compound (APY):

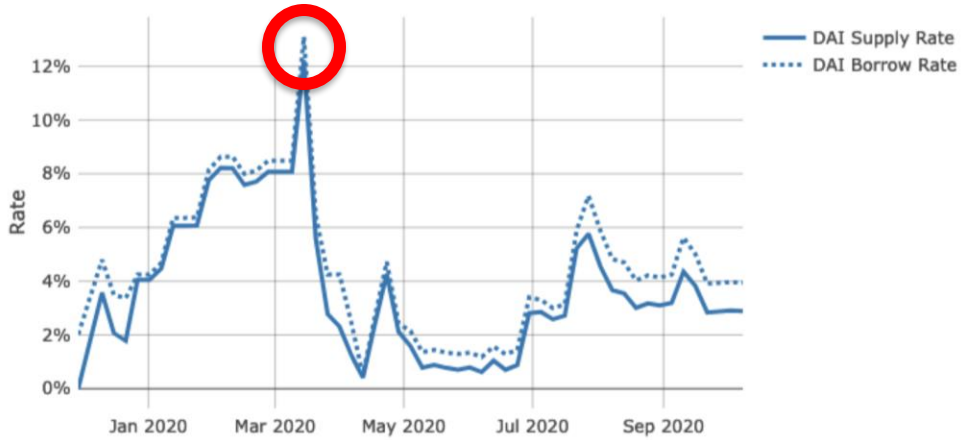
Demand for DAI spikes

$\Longrightarrow$ price of DAI spikes

$\Longrightarrow$ user's debt shoots up

$\Longrightarrow$ user's health drops

$\Longrightarrow$ liquidation …

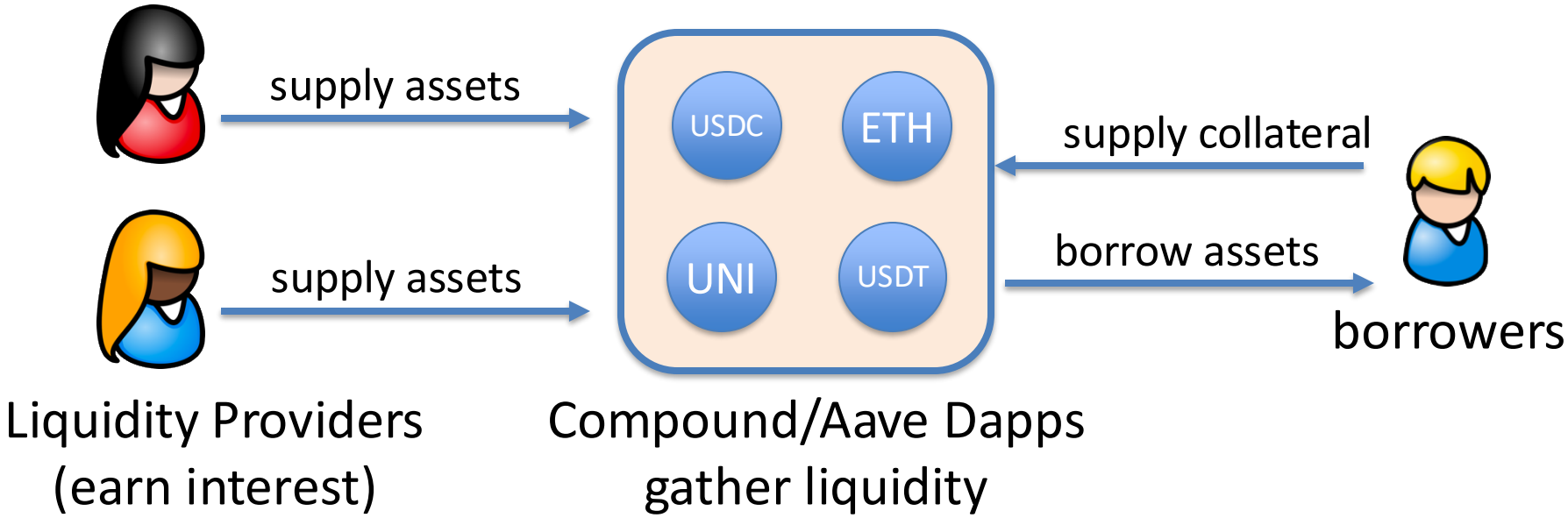


To use Compound/Aave, borrower must constantly monitor APY and quickly repay loans if APY goes too high (can be automated)

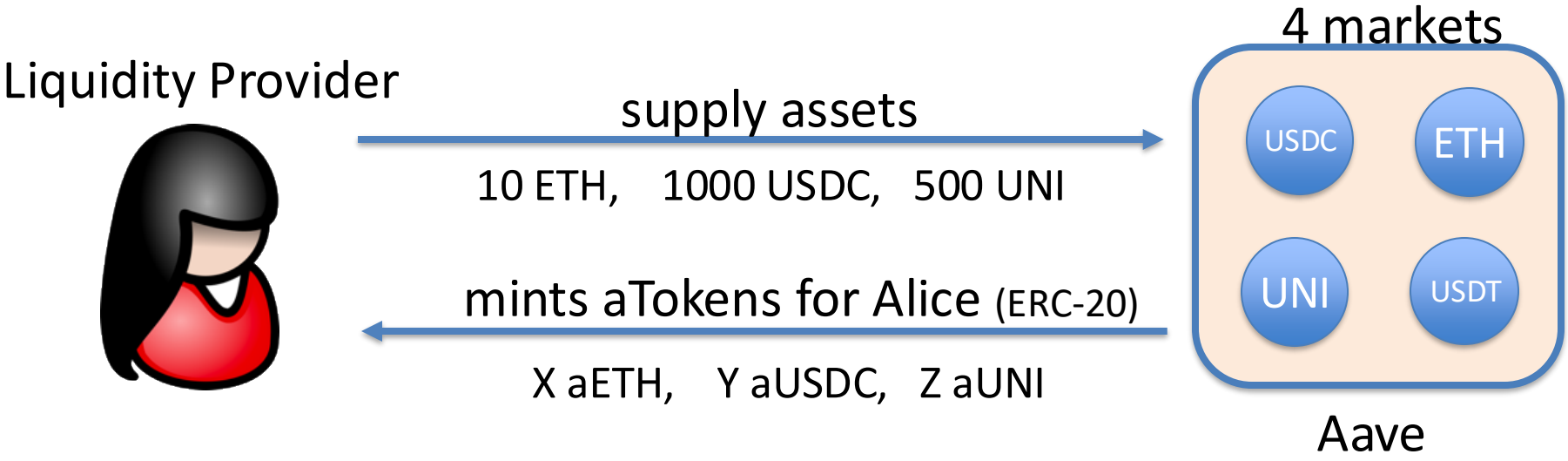
Recall: Lending

A better approach: liquidity pools

Over-collateralized lending: Compound and Aave

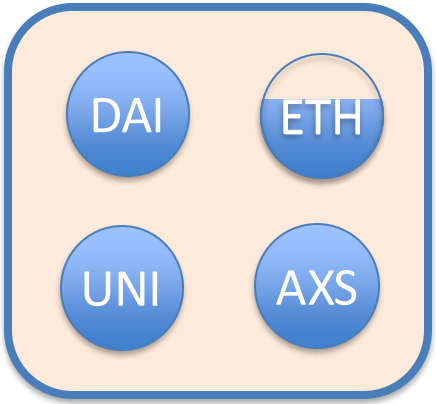


Example: Aave aTokens



Value of X, Y, Z is determined by the current exchange rate:
Token to aToken exchange rate is calculated every block

Borrowers



supply assets to any market

get aTokens

I want to borrow ETH

Bob's aTokens are locked up as collateral

Aave sends ETH to Bob

Bob's accrued interest increases ETH/aETH exchange rate

$\Longrightarrow$ benefit aETH token holders (ETH liquidity providers)

Liquidation

The health of the Aave Protocol is dependent on the 'health' of the collateralised positions within the protocol, also known as the 'health factor'. When the 'health factor' of an account's total loans is below 1, anyone can make a `liquidationCall()` to the `Pool` or L2Pool (in case of Arbitrum/Optimism) contract, pay back part of the debt owed and receive discounted collateral in return (also known as the liquidation bonus).

This incentivises third parties to participate in the health of the overall protocol, by acting in their own interest (to receive the discounted collateral) and as a result, ensure borrows are sufficiently collateralize.

There are multiple ways to participate in liquidations:

1. By calling the liquidationCall() directly in the Pool or L2Pool contract.
2. By creating your own automated bot or system to liquidate loans.

> ! For liquidation calls to be profitable, you must take into account the gas cost involved in liquidating the loan. If a high gas price is used, then the liquidation may be unprofitable for you. See the Calculating profitability section for

Liquidation

```
function liquidationCall(address collateral, address debt, address user, uint256
debtToCover, bool receiveAToken)
```

Liquidate positions with a **health factor** below 1.

When the health factor of a position is below 1, liquidators repay part or all of the outstanding borrowed amount on behalf of the borrower, while **receiving a discounted amount of collateral** in return (also known as a liquidation 'bonus"). Liquidators can decide if they want to receive an equivalent amount of collateral *aTokens* instead of the underlying asset. When the liquidation is completed successfully, the health factor of the position is increased, bringing the health factor above 1.

Liquidators can only close a certain amount of collateral defined by a close factor. Currently the **close factor is 0.5**. In other words, liquidators can only liquidate a maximum of 50% of the amount pending to be repaid in a position. The liquidation discount applies to this amount.

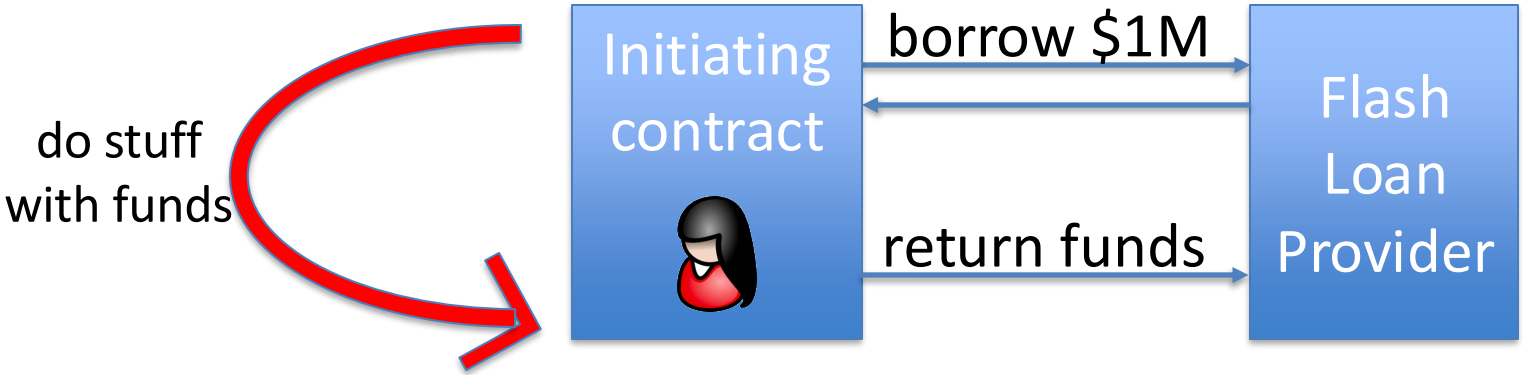
Liquidators must `approve()` the `Pool` contract to use `debtToCover` of the underlying ERC20 of the`asset` used

Flash loans

What is a flash loan?

A flash loan is taken and repaid in a single transaction

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



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

What is a flash loan?

flashLoanSimple

```
function flashLoanSimple( address receiverAddress, address asset, uint256 amount, bytes calldata params, uint16 referralCode)
```

Allows users to access liquidity of *one reserve or one transaction* as long as the amount taken plus fee is returned.

Receiver must approve the *Pool* contract for at least the *amount borrowed + fee,* else transaction will revert.

Does not waive few for approved *flashBorrowers* nor allow opening a debt position instead of repaying.

Referral program is currently inactive, you can pass `0` as `referralCode`. This program may be activated in the future through an Aave governance proposal

```
423       /// @inheritdoc IPool
424       function flashLoanSimple(
425         address receiverAddress,
426         address asset,
427         uint256 amount,
428         bytes calldata params,
429         uint16 referralCode
430       ) public virtual override {
431         DataTypes.FlashloanSimpleParams memory flashParams = DataTypes.FlashloanSimpleParams({
432           receiverAddress: receiverAddress,
433           asset: asset,
434           amount: amount,
435           params: params,
436           referralCode: referralCode,
437           flashLoanPremiumToProtocol: _flashLoanPremiumToProtocol,
438           flashLoanPremiumTotal: _flashLoanPremiumTotal
439         });
440         FlashLoanLogic.executeFlashLoanSimple(_reserves[asset], flashParams);
441       }
```

```
182     function executeFlashLoanSimple(
183       DataTypes.ReserveData storage reserve,
184       DataTypes.FlashloanSimpleParams memory params
185     ) external {
186       // The usual action flow (cache -> updateState -> validation -> changeState -> updateRates)
187       // is altered to (validation -> user payload -> cache -> updateState -> changeState -> updateRates) for flashloans.
188       // This is done to protect against reentrance and rate manipulation within the user specified payload.
189
190       ValidationLogic.validateFlashloanSimple(reserve);
191
192       IFlashLoanSimpleReceiver receiver = IFlashLoanSimpleReceiver(params.receiverAddress);
193       uint256 totalPremium = params.amount.percentMul(params.flashLoanPremiumTotal);
194       IAToken(reserve.aTokenAddress).transferUnderlyingTo(params.receiverAddress, params.amount);
195
196       require(
197         receiver.executeOperation(
198           params.asset,
199           params.amount,
200           totalPremium,
201           msg.sender,
202           params.params
203         ),
204         Errors.INVALID_FLASHLOAN_EXECUTOR_RETURN
205       );
206
207       _handleFlashLoanRepayment(
208         reserve,
209         DataTypes.FlashLoanRepaymentParams({
210           asset: params.asset,
211           receiverAddress: params.receiverAddress,
212           amount: params.amount,
213           totalPremium: totalPremium,
214           flashLoanPremiumToProtocol: params.flashLoanPremiumToProtocol,
215           referralCode: params.referralCode
216         })
217       );
218     }
```

```
/**
 * @title IFlashLoanReceiver
 * @author Aave
 * @notice Defines the basic interface of a flashloan-receiver contract.
 * @dev Implement this interface to develop a flashloan-compatible flashLoanReceiver contract
 */
interface IFlashLoanReceiver {
  /**
   * @notice Executes an operation after receiving the flash-borrowed assets
   * @dev Ensure that the contract can return the debt + premium, e.g., has
   *      enough funds to repay and has approved the Pool to pull the total amount
   * @param assets The addresses of the flash-borrowed assets
   * @param amounts The amounts of the flash-borrowed assets
   * @param premiums The fee of each flash-borrowed asset
   * @param initiator The address of the flashloan initiator
   * @param params The byte-encoded params passed when initiating the flashloan
   * @return True if the execution of the operation succeeds, false otherwise
   */
  function executeOperation(
    address[] calldata assets,
    uint256[] calldata amounts,
    uint256[] calldata premiums,
    address initiator,
    bytes calldata params
  ) external returns (bool);

  function ADDRESSES_PROVIDER() external view returns (IPoolAddressesProvider);

  function POOL() external view returns (IPool);
}
```

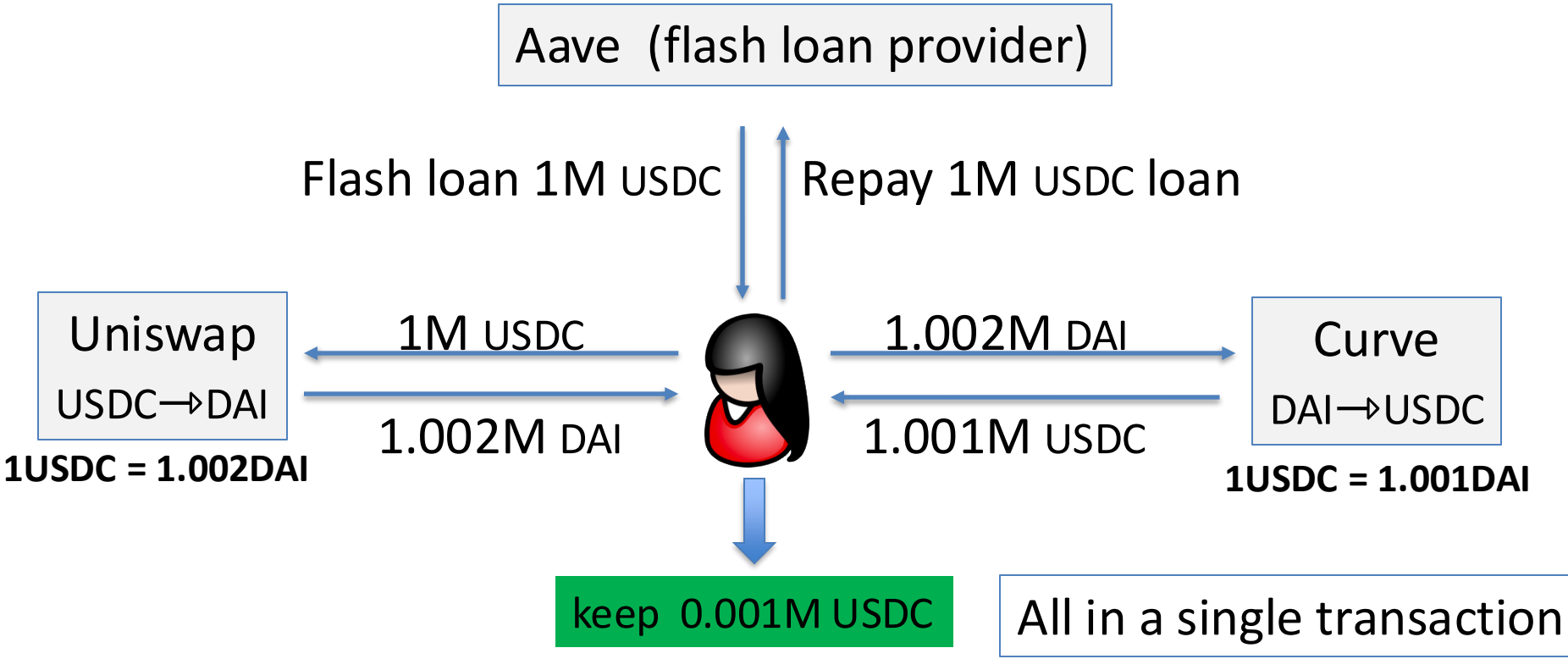
Use cases

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

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

Risk free arbitrage

Alice finds a USDC/DAI price difference in two pools



Collateral swap

start:

Alice @Compound

end goal:

Alice @Compound

-1000 DAI
+1 cETH

borrowed DAI using ETH as collateral

Take 1000 DAI flash loan
Repay 1000 DAI debt
Redeem 1 cETH
Swap 1 cETH for 3000 cUSDC
Deposit 3000 cUSDC as collateral
Borrow 1000 DAI
Repay 1000 DAI flash loan

(a single Ethereum transaction)

-1000 DAI
+3000 cUSDC

borrowed DAI using USDC as collateral

Decentralized Exchanges

What is an 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

CeX

5 ETH

1000: ETH
10000: USDC
10: GTC

Treasury

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

CeX

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?

Remember FTX?

DeX

DeX idea:

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

Properties of a DEX:

- Programmable: can be used as a service by other contracts
- Transparent: code is available for everyone to see
- Permissionless: anyone can use
- Non-Custodial

How to build a DeX?

First idea: on-chain order book

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

Problem: on some chains, this is gas inefficient.

- Orders cost gas: when placed, when filled, when canceled.
- Matching buy orders to sell orders takes lots of gas
- **Feasible on chains with cheap gas!**

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.
- Examples: 0x, dydx, etc.

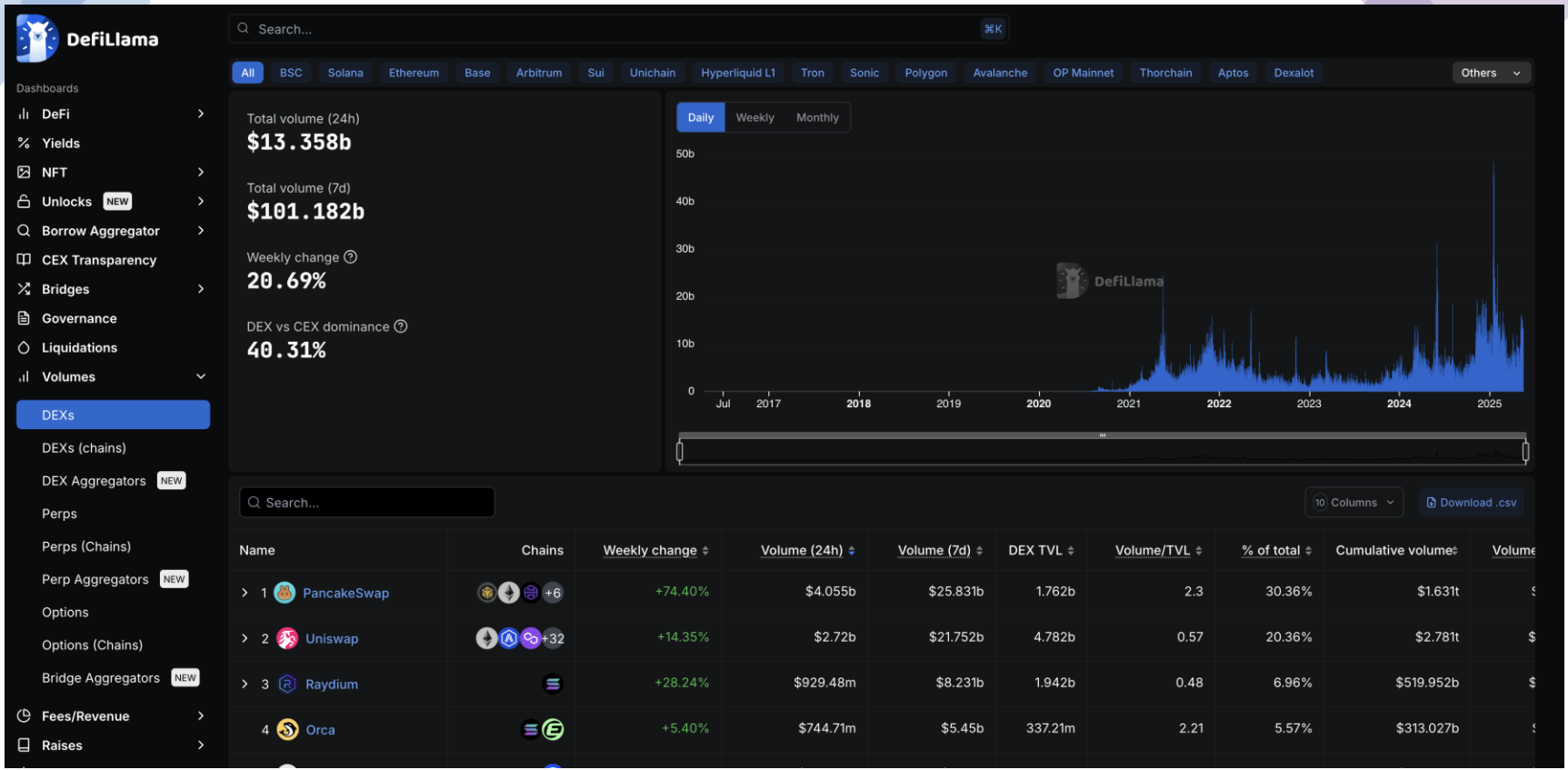
Problem: order book is not accessible to contracts

How to build a DeX?

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, SushiSwap, Balancer, Curve

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


DefiLlama
Search...
⌘K
All
BSC
Solana
Ethereum
Base
Arbitrum
Sui
Unichain
Hyperliquid L1
Tron
Sonic
Polygon
Avalanche
OP Mainnet
Thorchain
Aptos
Dexalot
Others
Dashboards
DeFi
Yields
NFT
Unlocks NEW
Borrow Aggregator
CEX Transparency
Bridges
Governance
Liquidations
Volumes
DEXs
DEXs (chains)
DEX Aggregators NEW
Perps
Perps (Chains)
Perp Aggregators NEW
Options
Options (Chains)
Bridge Aggregators NEW
Fees/Revenue
Raises
Total volume (24h)
$13.358b
Total volume (7d)
$101.182b
Weekly change
20.69%
DEX vs CEX dominance
40.31%
Daily
Weekly
Monthly
50b
40b
30b
20b
10b
0
Jul
2017
2018
2019
2020
2021
2022
2023
2024
2025
DefiLlama
Search...
10 Columns
Download .csv
Name
Chains
Weekly change
Volume (24h)
Volume (7d)
DEX TVL
Volume/TVL
% of total
Cumulative volume
Volume
1 PancakeSwap +6 +74.40% $4.055b $25.831b 1.762b 2.3 30.36% $1.631t
2 Uniswap +32 +14.35% $2.72b $21.752b 4.782b 0.57 20.36% $2.781t
3 Raydium +28.24% $929.48m $8.231b 1.942b 0.48 6.96% $519.952b
4 Orca +5.40% $744.71m $5.45b 337.21m 2.21 5.57% $313.027b

Automated Market Maker

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

stable

volitile

Liquidity providers

USDC, WETH

USDC, WETH

(earn interest)

Uniswap

USDC-WETH pool

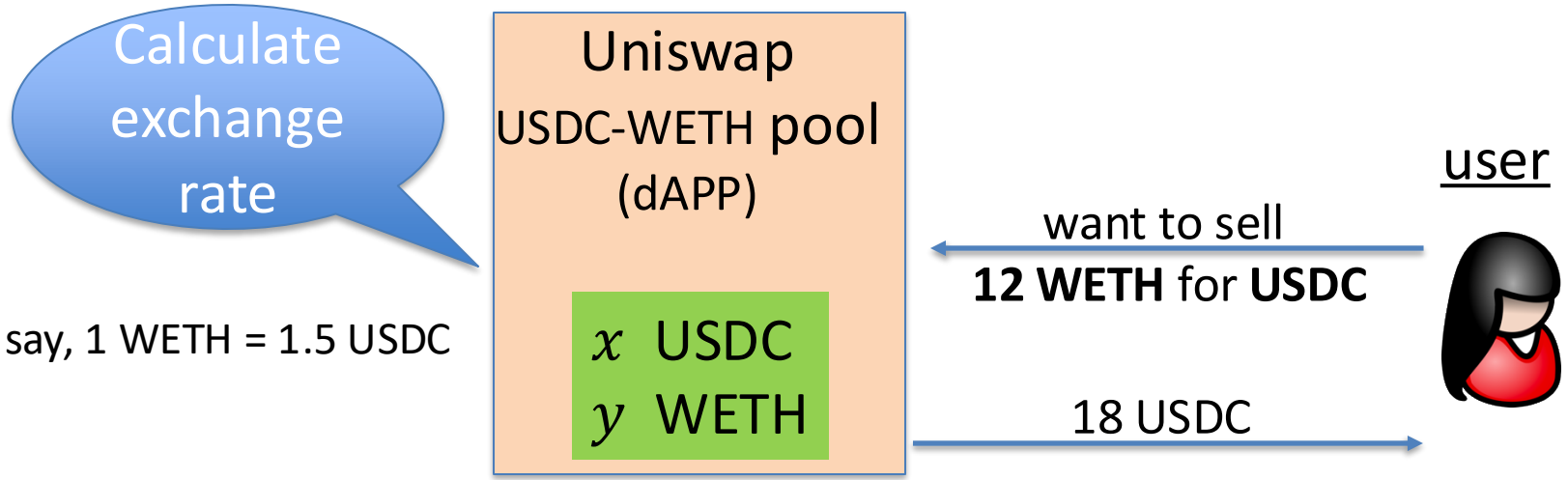
(dAPP)

x USDC

y WETH

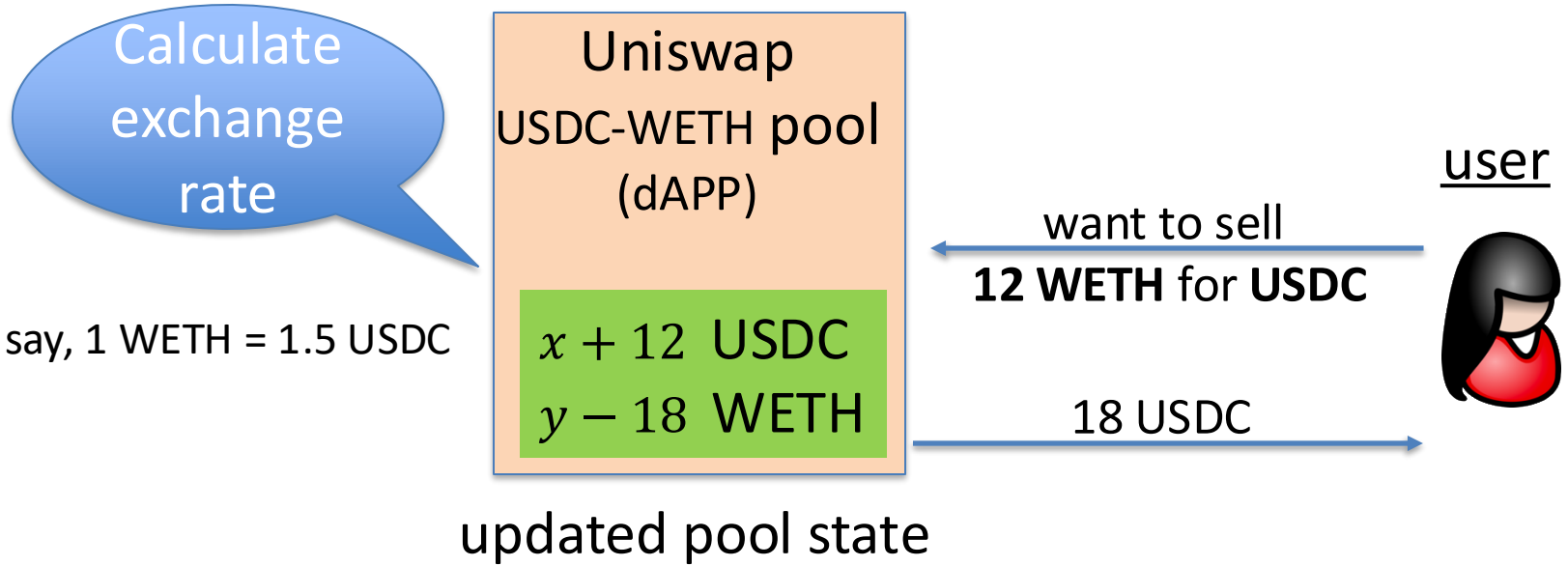
Automated Market Maker

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



Automated Market Maker

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



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 of 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: $-\frac{dy}{dx} = y/x$

How to determine exchange rate?

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

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

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

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

The constant product market maker:

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

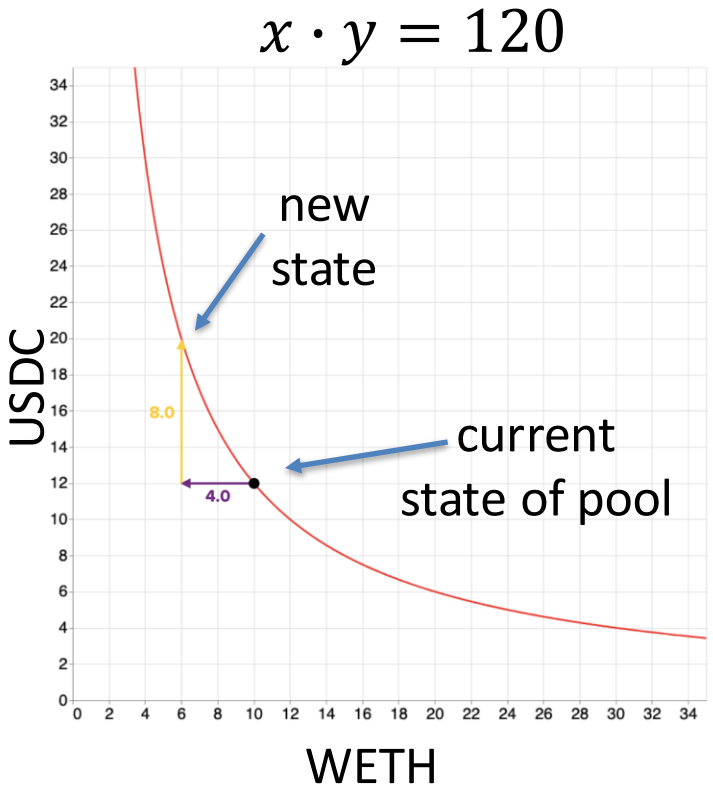
  $10 \times 12 = 120$

- 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

$y \rightarrow y + 8 = 20$



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?

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

But liquidity providers (LP's) take a fee $\phi \in [0,1]$ (say ϕ=0.97)

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

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

Buy and sell equations

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

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

```
41
42      // given an input amount of an asset and pair reserves, returns the maximum output amount of the other asset
43      function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) internal pure returns (uint amountOut) {
44          require(amountIn > 0, 'UniswapV2Library: INSUFFICIENT_INPUT_AMOUNT');
45          require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
46          uint amountInWithFee = amountIn.mul(997);
47          uint numerator = amountInWithFee.mul(reserveOut);
48          uint denominator = reserveIn.mul(1000).add(amountInWithFee);
49          amountOut = numerator / denominator;
50      }
51
```

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

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

```
51
52      // given an output amount of an asset and pair reserves, returns a required input amount of the other asset
53      function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) internal pure returns (uint amountIn) {
54          require(amountOut > 0, 'UniswapV2Library: INSUFFICIENT_OUTPUT_AMOUNT');
55          require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
56          uint numerator = reserveIn.mul(amountOut).mul(1000);
57          uint denominator = reserveOut.sub(amountOut).mul(997);
58          amountIn = (numerator / denominator).add(1);
59      }
60
```

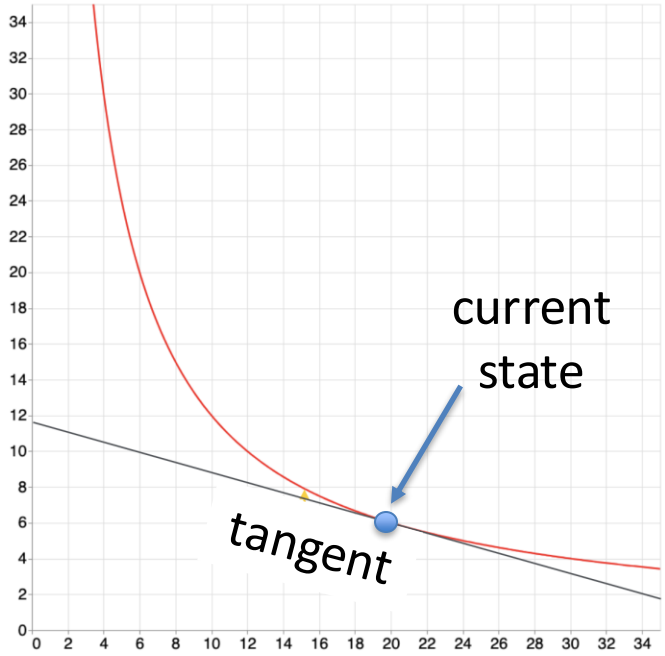
UniswapV2Library.sol

The marginal price as a tangent

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

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

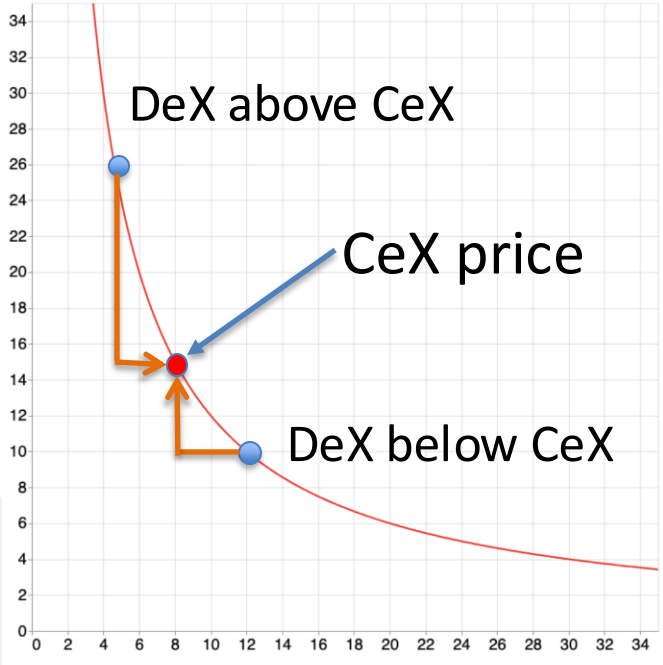
$\Rightarrow \quad -p$ is the slope of the tangent at the current state



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

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

- DeX state is below market rate
  ⇒ arbitrageurs will move DeX up
- DeX state is above market rate
  ⇒ arbitrageurs will move DeX down



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

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

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

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

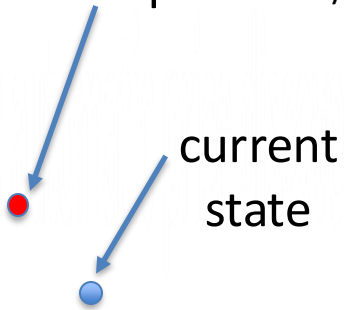
$\Rightarrow$ marginal price $p = y/x = 0.833$

Suppose a CeX offers a different price

$p_{market} = 1.875$

$\Rightarrow$ arbitrage opportunity!

This point matches the CeX price: 15/8=1.875



$p = y/x = 0.833$

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

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

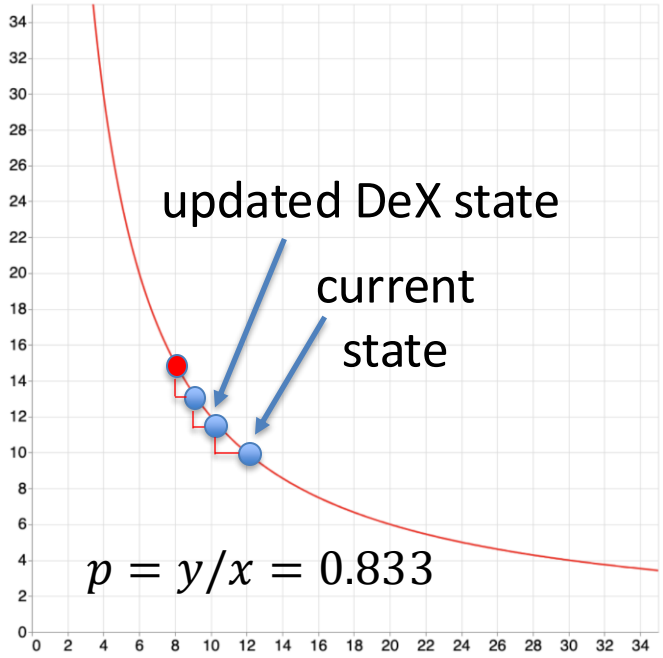
Arbitrageur will do:

- 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

$\Rightarrow$ Arb. is providing a service, and making a profit

Where did the 0.44 Y come from? LP's lost money



Problem 1: Slippage

Slippage:

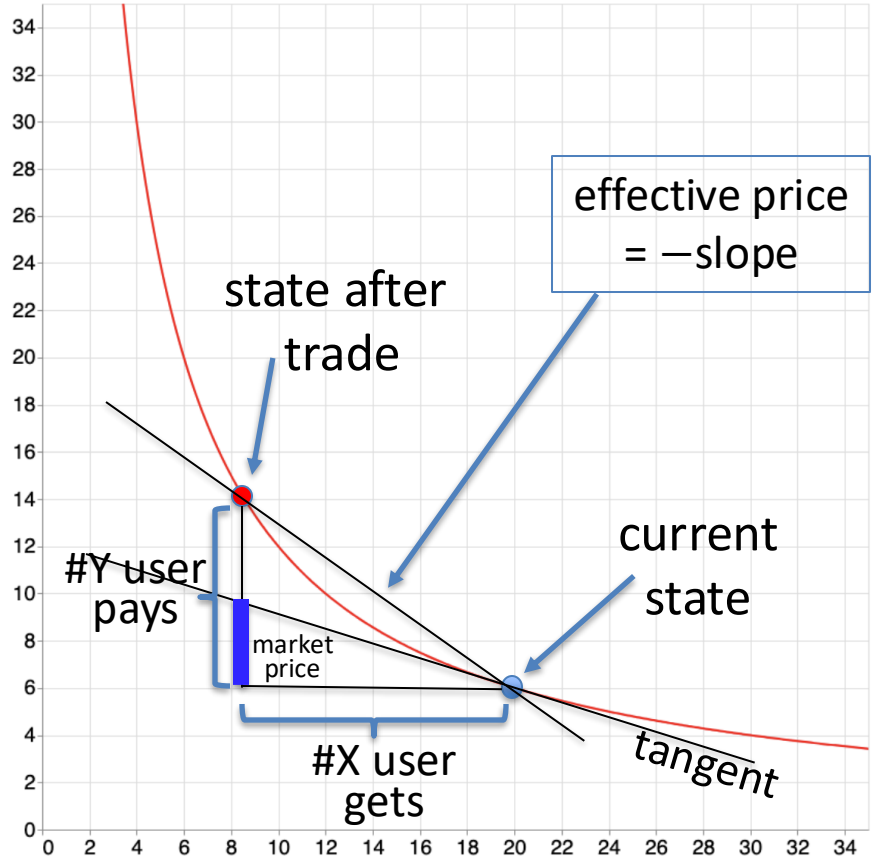
- the larger the trade,
  the worse the exchange rate
  is for the user

market price = −tangent slope

⇒ #Y user should pay = blue line

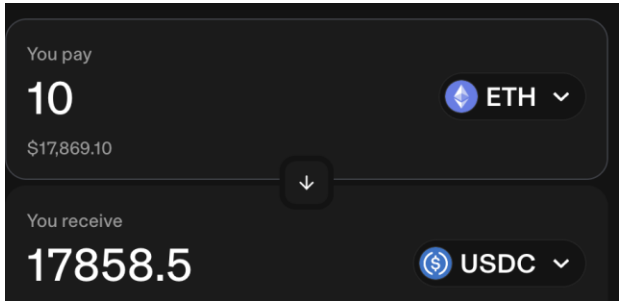
… but user pays more

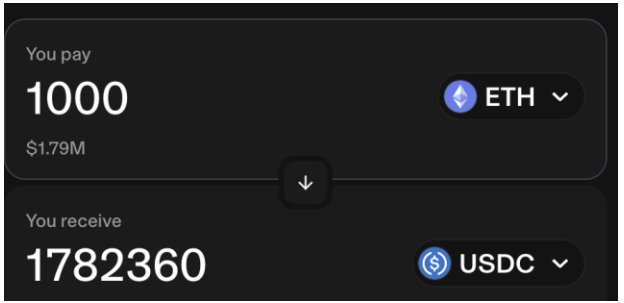
(uniswap bounds slippage at 0.5%)



Slippage: an example

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





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

Note: if $\Delta x = x$ (Alice wants to buy entire pool) then price is ∞

$\Rightarrow$ Pool will never run out of X or Y tokens.

https://app.uniswap.org/swap

Problem 2: the sandwich attack

Consider the WETH-USDC pool:

- User Alice submits a Tx to sell Δx USDC to pool.
- 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

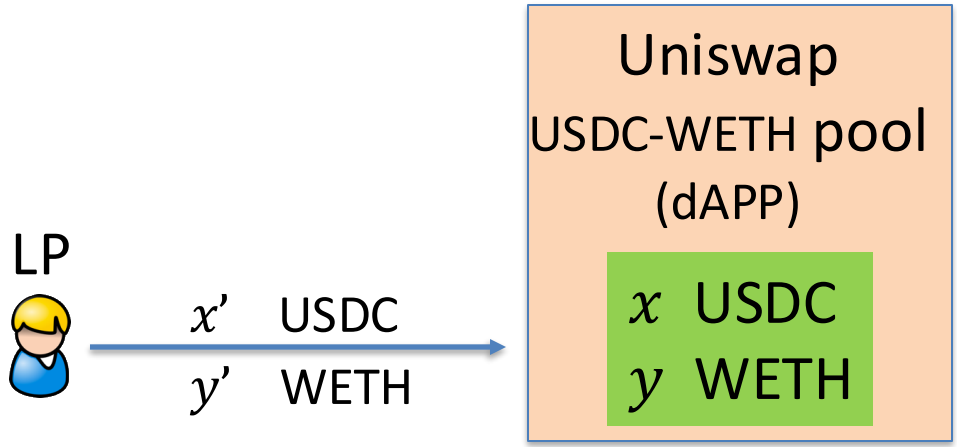
Incentives for liquidity providers

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

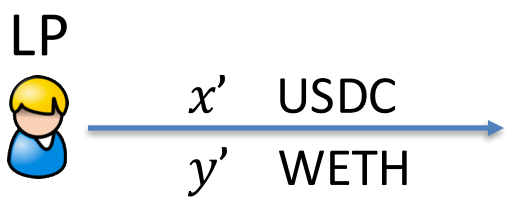


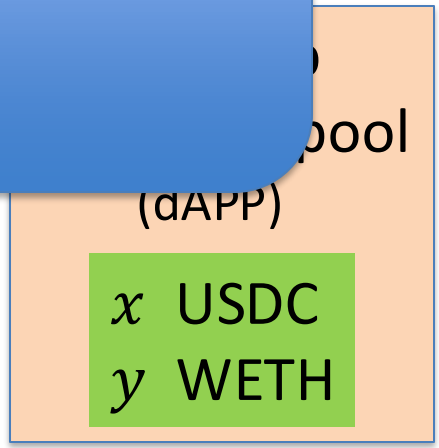
Recall: liquidity providers (LP's)

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

Note: LP contribution changes the constant k:

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



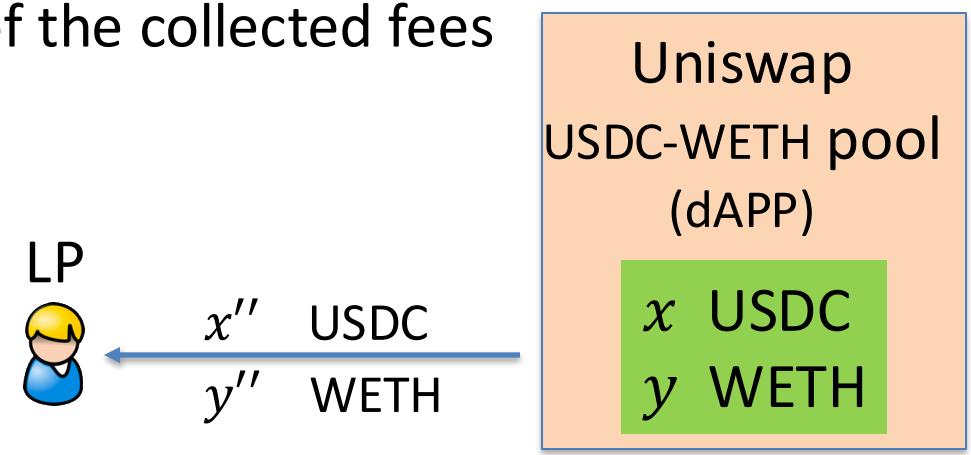


LP withdrawal

(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: $P(t_1) \cdot x' + y'$ WETH.

AMM strategy: $P(t_1) \cdot x' + y'$ **+ fees** WETH.

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

HOLD strategy: $P(t_1) \cdot x' + y'$ WETH.

AMM strategy: $[P(t_1) \cdot x' + y'] \cdot \boldsymbol{M}(\Delta - 1)$ **+ fees** WETH,

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

Loss vs. Hold

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

⇒ 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

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

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

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)

- 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 (proj #4)
- Automatic price discovery (no off-chain oracles)
- No dependence on a central point of control
- Fully composable with other dAPPs