CS251 Fall 2025

(cs251.stanford.edu)

Stablecoins & Lending Protocols

Dan Boneh

Recap: Solidity

Everything is a contract:

- Contracts manage state variables
- Contracts have functions that can be called externally
- Can inherit code from other contracts (`contract A is B,C`)

Global objects: `block, tx, msg`

(e.g. `block.number, tx.gasprice, msg.sender`)

An example: ERC20 tokens

- https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

- A standard API for fungible tokens. (ERC-721 for non-fungible tokens)

- An ERC20 token is itself a smart contract that maintains all user balances:

  mapping(address => uint256) internal _**balances**;

- A standard interface allows other contracts to interact with every ERC20 token. No need for special logic for each token.

ERC20 token interface

```
function transfer(address _to,   uint256 _value) external returns (bool);

function transferFrom(address _from,   address _to,   uint256 _value) external returns (bool);

function approve(address _spender,  uint256 _value) external returns (bool);


function totalSupply() external view returns (uint256);

function balanceOf(address _owner) external view returns (uint256);

function allowance(address _owner, address _spender) external view returns (uint256);
```

An example …

Consider two ERC-20 tokens: say USDC and WETH

- USDC is a contract that maintains a **_balances[]** mapping
- WETH is a different contract that also maintains **_balances[]**

Say Bob owns 5 USDC and 2 WETH. This is recorded as:

In USDC contract: **_balances[Bob’s address] == 5**

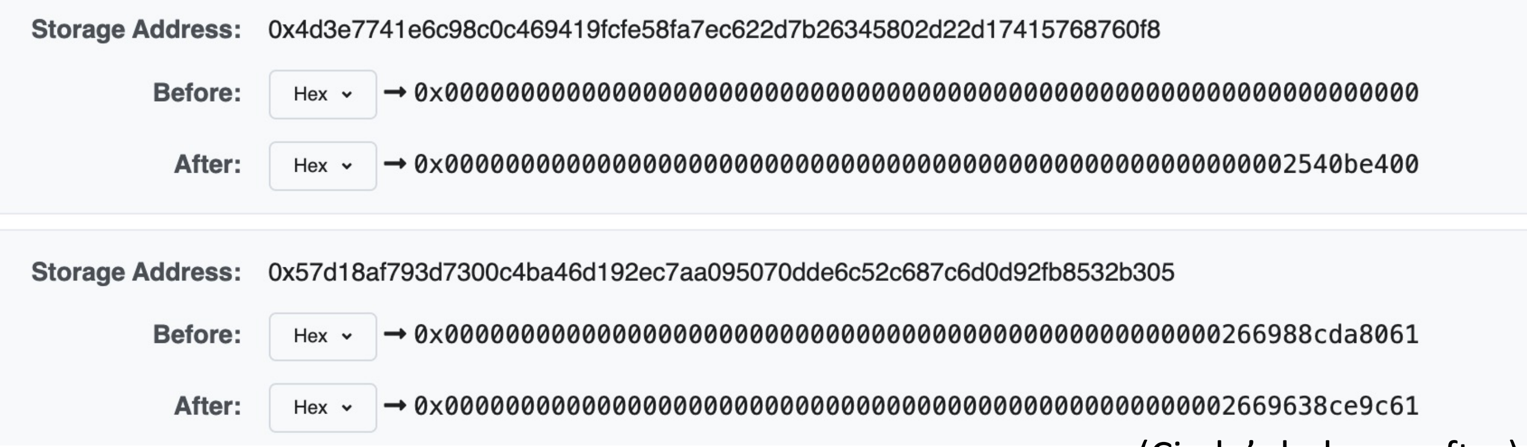
In WETH contract: **_balances[Bob’s address] == 2**

Wallet software shows all the coins associated with Bob’s address

Anyone can read ERC20 _balances[]

Transaction Hash: 0x6b85ca95e484d94503d1276456bfc32cc55f6fdb8bb231ff83....

Tells the USDC contract to transfer 10,010.00 USDC
from Circle's account to 0x7656159E42209A95b77aD374d...



```
Storage Address:  0x4d3e7741e6c98c0c469419fcfe58fa7ec622d7b26345802d22d17415768760f8

        Before:  Hex ⌄  → 0x0000000000000000000000000000000000000000000000000000000000000000

         After:  Hex ⌄  → 0x00000000000000000000000000000000000000000000000000000002540be400
```

recipient's entry

```
Storage Address:  0x57d18af793d7300c4ba46d192ec7aa095070dde6c52c687c6d0d92fb8532b305

        Before:  Hex ⌄  → 0x000000000000000000000000000000000000000000000000000266988cda8061

         After:  Hex ⌄  → 0x0000000000000000000000000000000000000000000000000002669638ce9c61
```

Circle's entry

(Circle's balance after)

(etherscan.io)

Calling other contracts

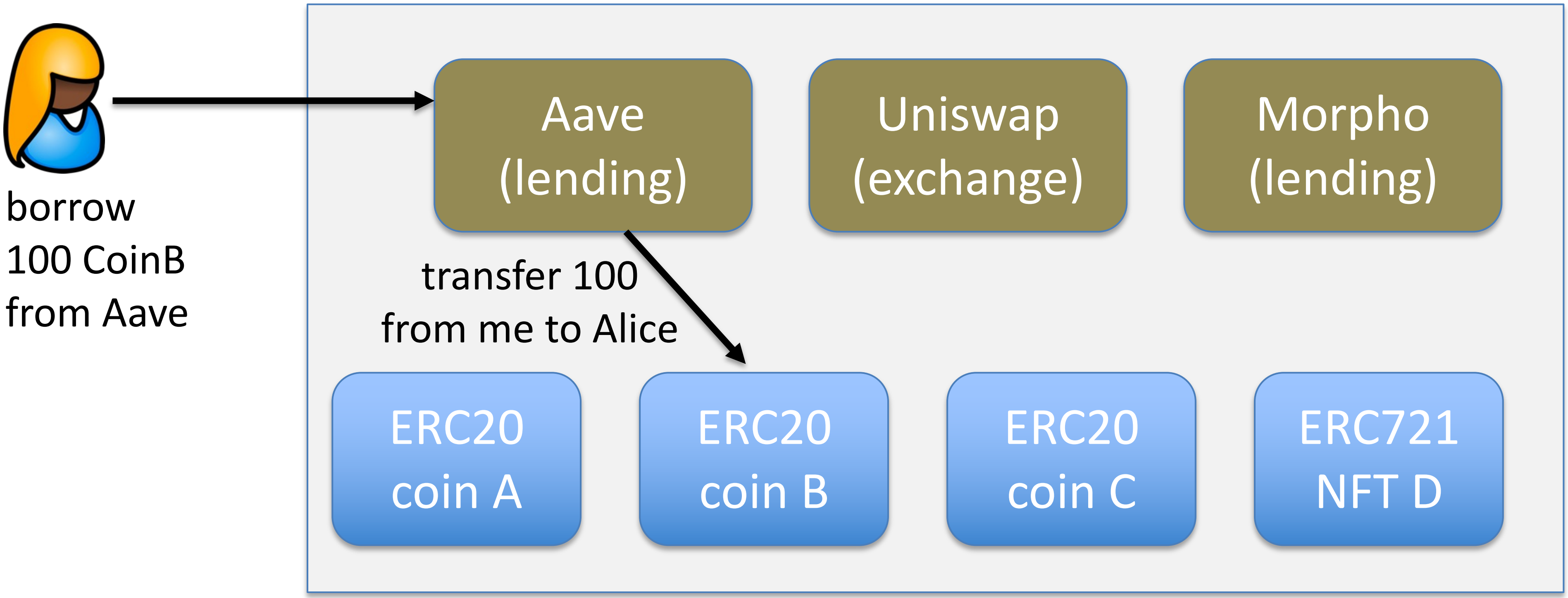
Addresses can be cast to contract types.

```
address _usdc = 0x7656159E42209A95b77aD374d…;
ERC20Token usdcContract = ERC20Token(_usdc);
```

To call the “transfer” function of contract at address _usdc:

```
usdcContract.transfer(_to,  _value);
```

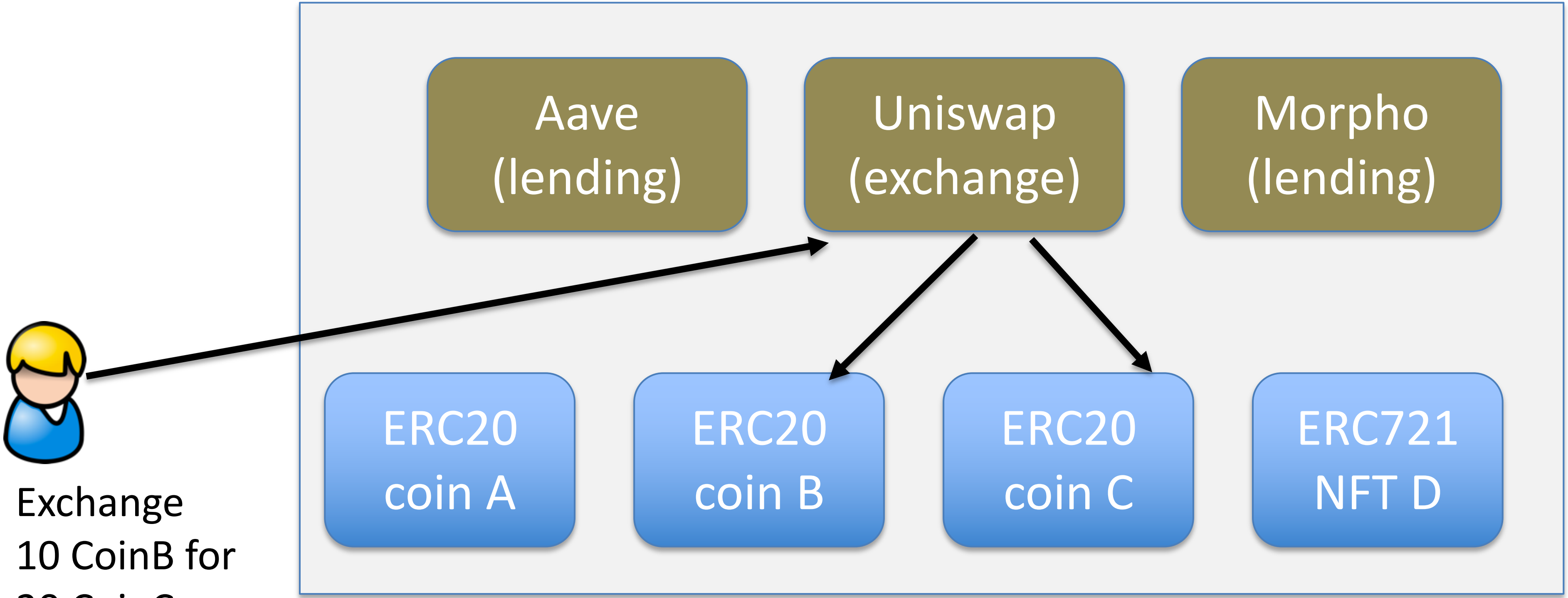
The world of DeFi



borrow
100 CoinB
from Aave

on-chain contracts

The world of DeFi



Aave (lending)

Uniswap (exchange)

Morpho (lending)

ERC20 coin A

ERC20 coin B

ERC20 coin C

ERC721 NFT D

Exchange
10 CoinB for
20 CoinC

on-chain contracts

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

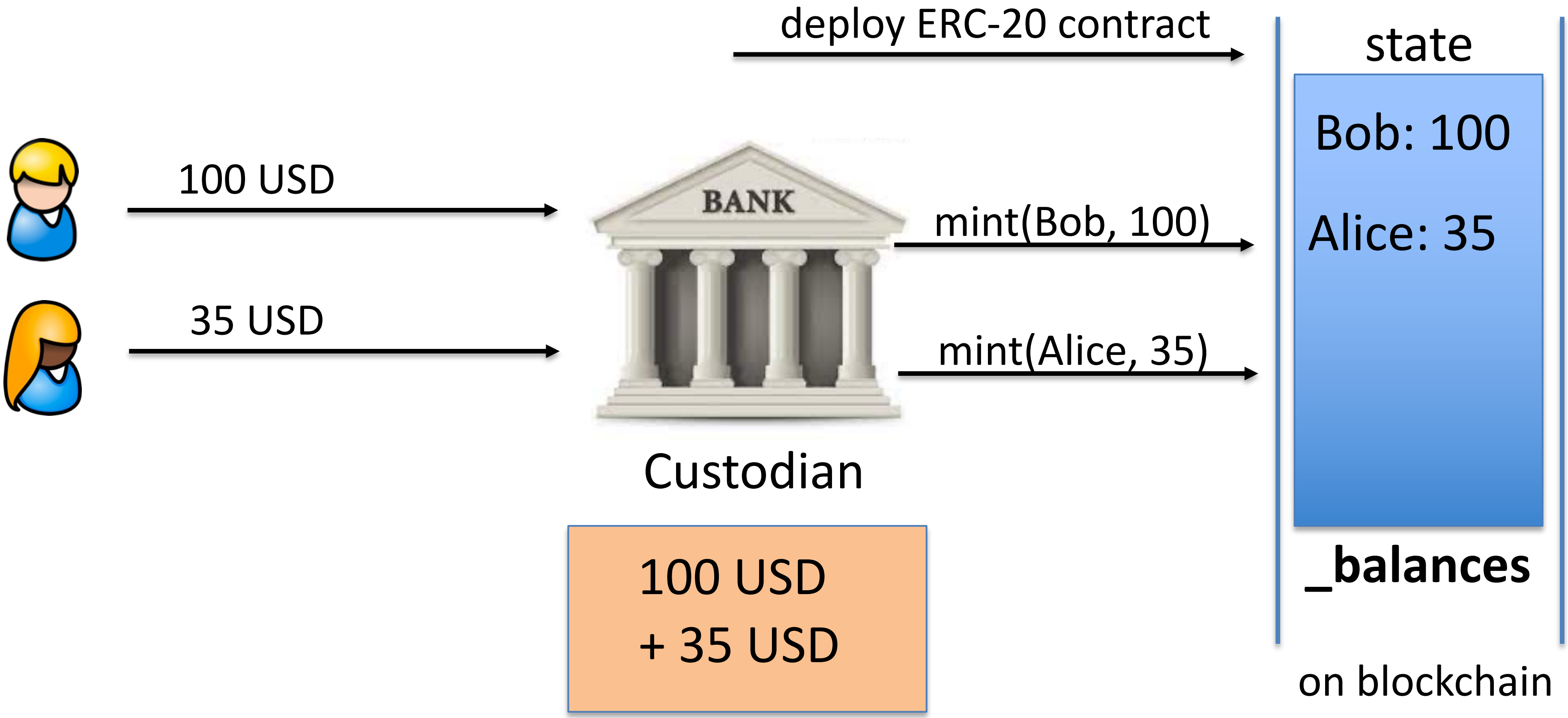
Goals:

- 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 (USDC)	synthetics (DAI, RAI)
Un(der)collateralized	central bank (digital) currency	Undercollateralized stablecoins

Custodial stablecoins: minting



deploy ERC-20 contract

100 USD

35 USD

mint(Bob, 100)

mint(Alice, 35)

Custodian

100 USD
+ 35 USD

state

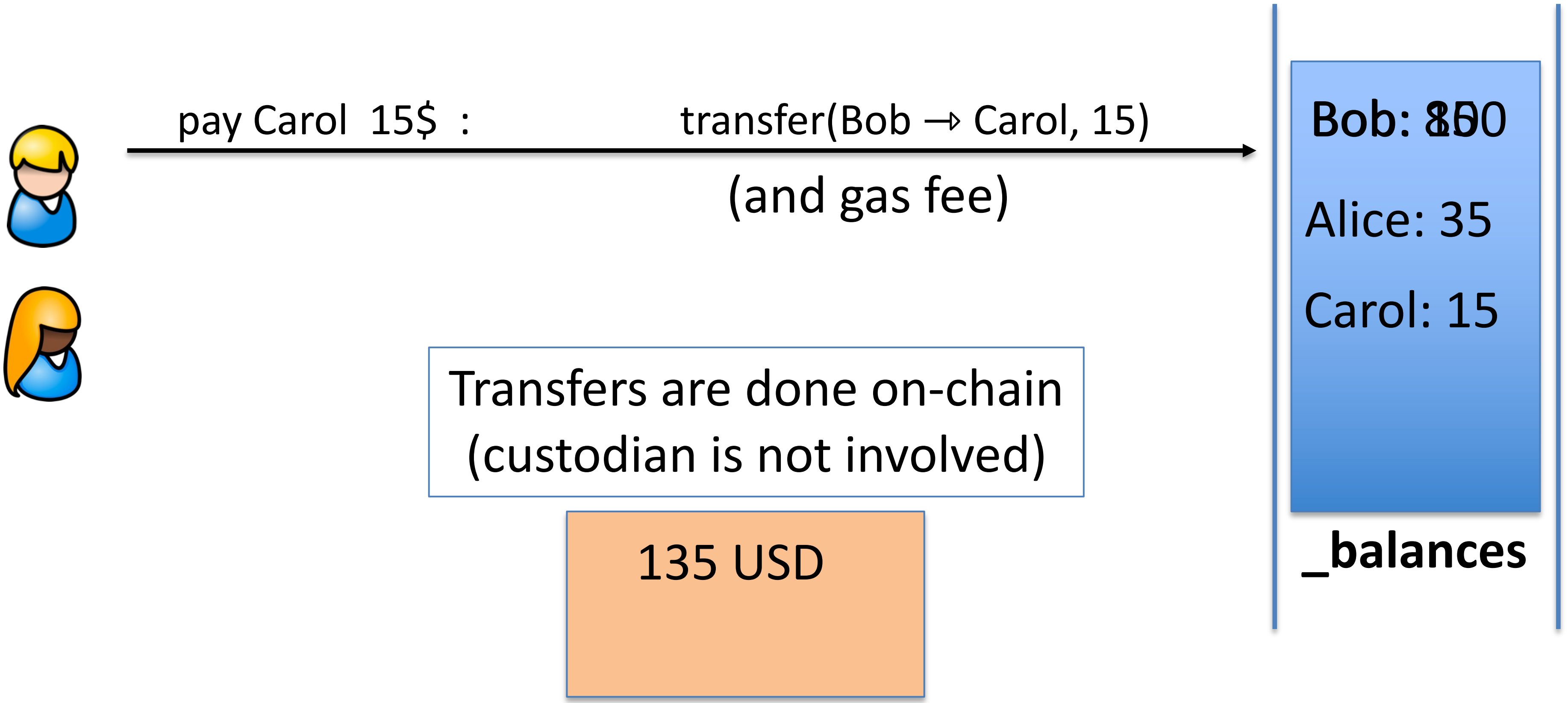
Bob: 100

Alice: 35

_balances

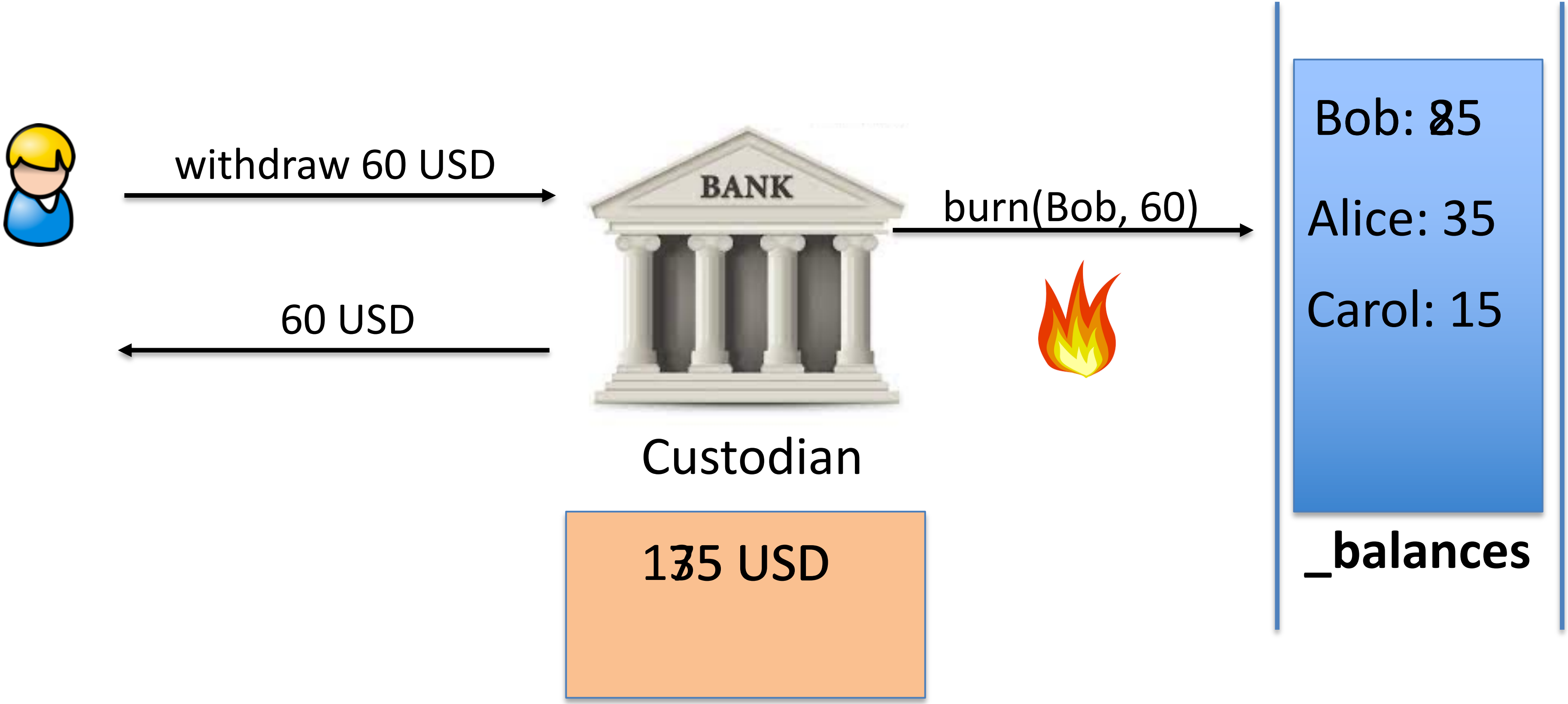
on blockchain

Custodial stablecoins: transfers



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

Custodial stablecoins: withdrawal



withdraw 60 USD

60 USD

burn(Bob, 60)

Custodian

135 USD

Bob: 85

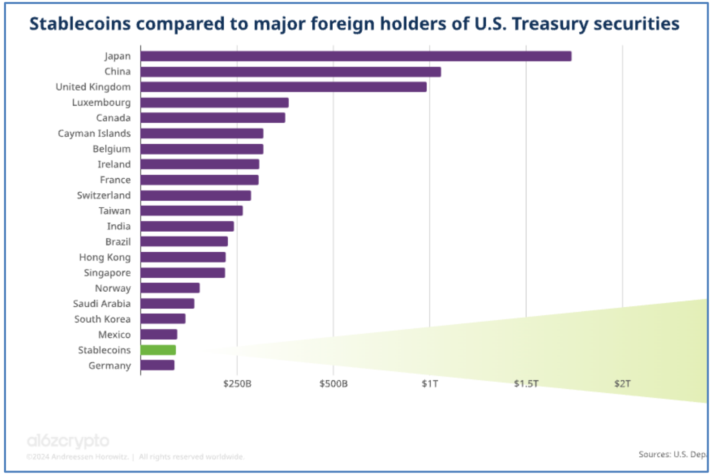
Alice: 35

Carol: 15

_balances

Two Examples

	Coins issued	24h volume
USDC	76 B	20.4 B
USDT	184 B	164.6 B



Stablecoins compared to major foreign holders of U.S. Treasury securities

Sources: U.S. Dep

stablecoin providers are a large holder of U.S. debt.

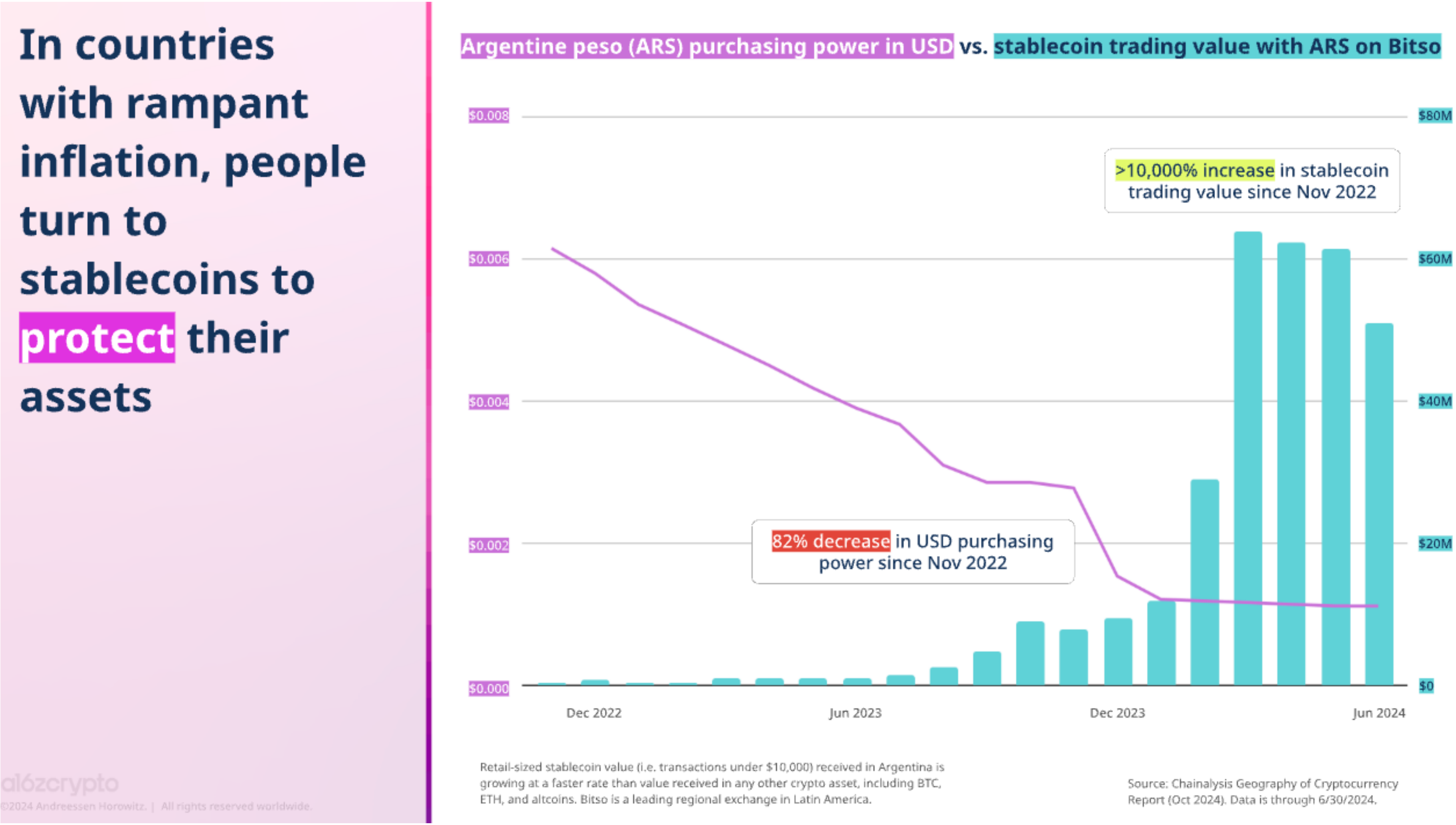
Some issues

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



In countries with rampant inflation, people turn to stablecoins to protect their assets

Argentine peso (ARS) purchasing power in USD vs. stablecoin trading value with ARS on Bitso

>10,000% increase in stablecoin trading value since Nov 2022

82% decrease in USD purchasing power since Nov 2022

Retail-sized stablecoin value (i.e. transactions under $10,000) received in Argentina is growing at a faster rate than value received in any other crypto asset, including BTC, ETH, and altcoins. Bitso is a leading regional exchange in Latin America.

Source: Chainalysis Geography of Cryptocurrency Report (Oct 2024). Data is through 6/30/2024.

a16zcrypto

source: state of crypto report

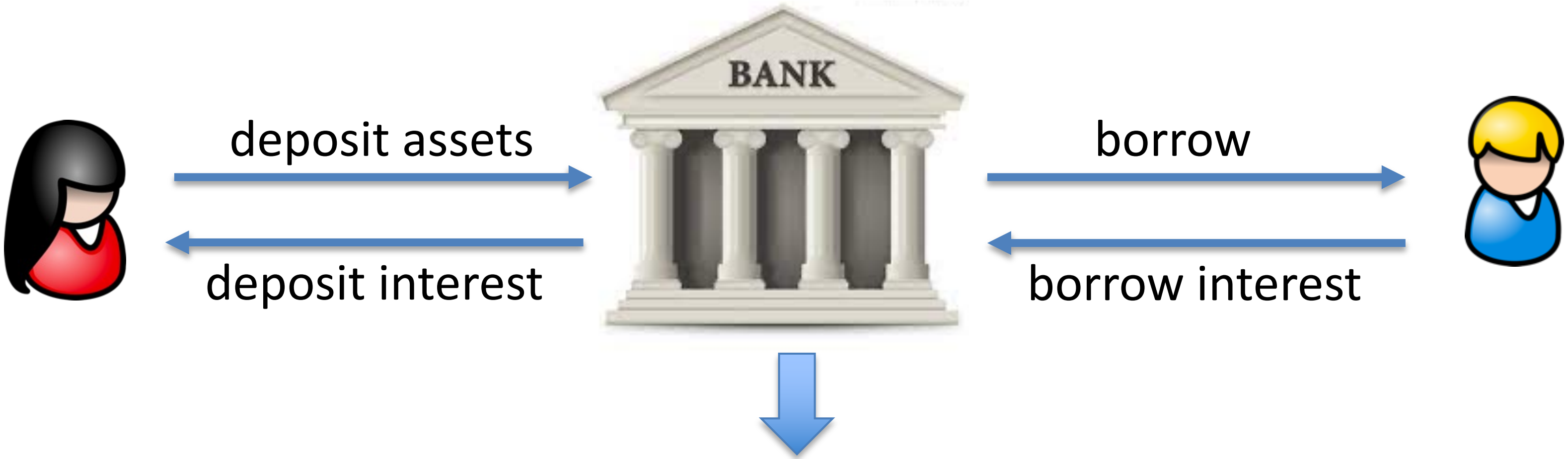
DeFi app #2: Lending Protocols

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



deposit assets

deposit interest

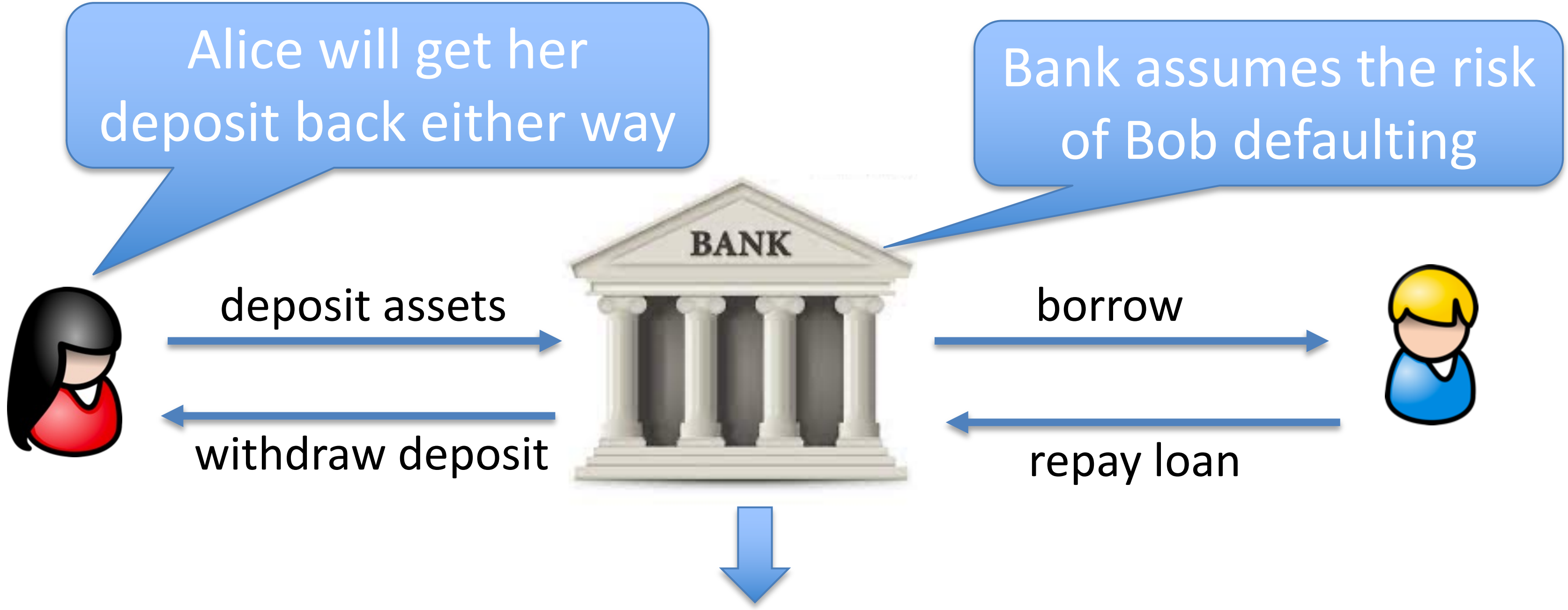
borrow

borrow interest

bank spread

(borrow interest – deposit interest)

The role of banks in the economy

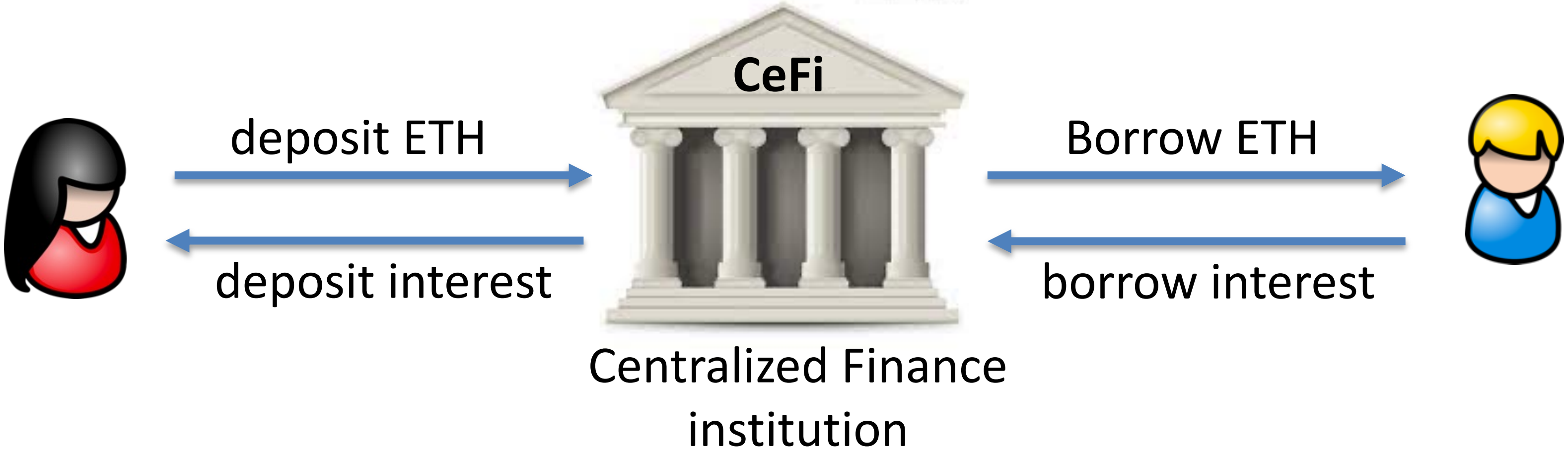

Alice will get her deposit back either way
Bank assumes the risk of Bob defaulting
BANK
deposit assets
withdraw deposit
borrow
repay loan

bank spread

(borrow interest – deposit interest)

Crypto: CeFi lending (e.g., Nexo)

Same as with a traditional bank:



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

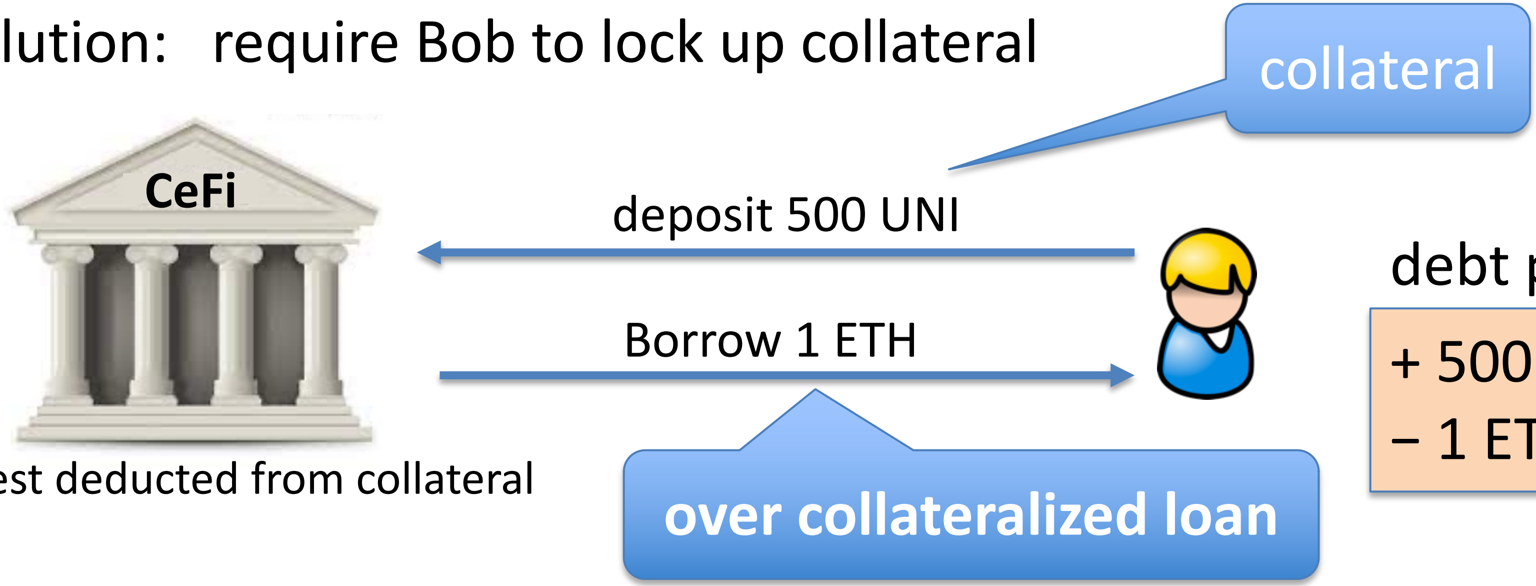
The role of collateral

(1 ETH = 100 UNI)

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

⟹ CeFi will absorb the loss

Solution: require Bob to lock up collateral



interest deducted from collateral

debt position:

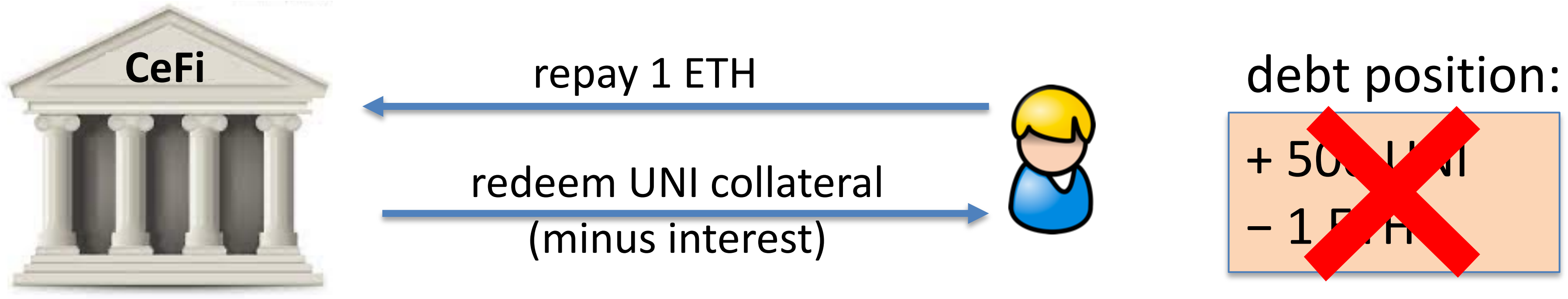
+ 500 UNI
− 1 ETH

The role of collateral

(1 ETH = 100 UNI)

Several things can happen next:

(1) Bob repays loan


CeFi

repay 1 ETH

redeem UNI collateral
(minus interest)

debt position:

~~+ 500 UNI~~
~~− 1 ETH~~

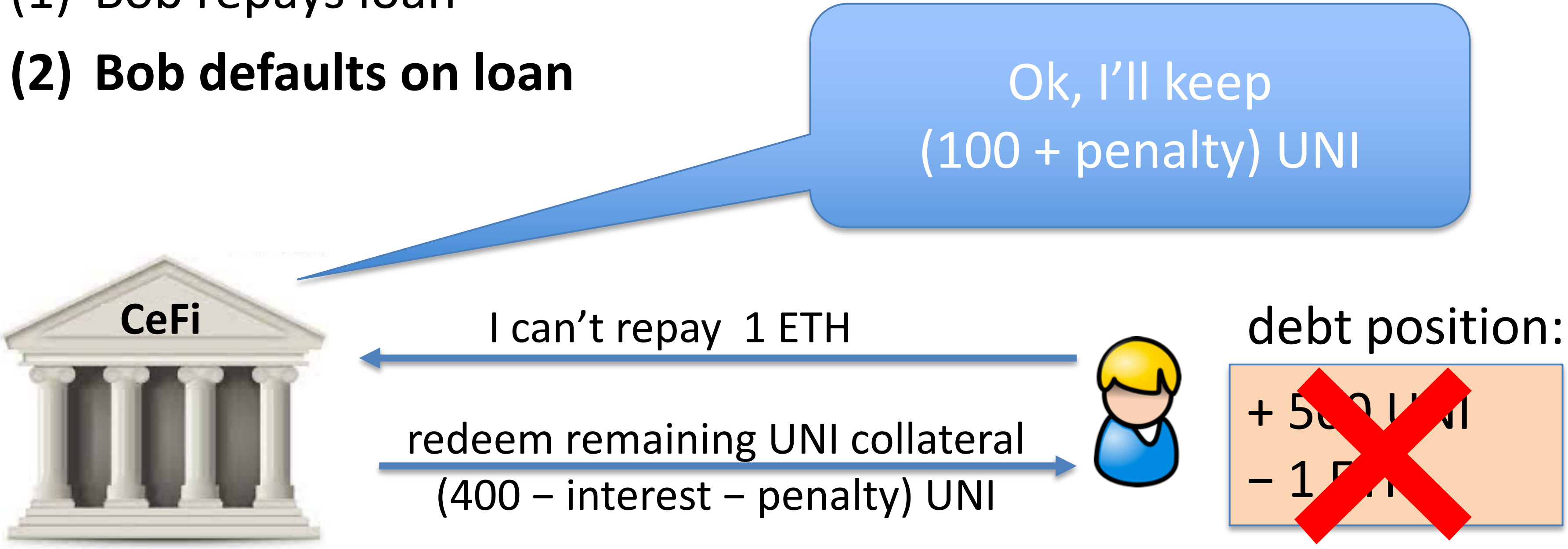
The role of collateral

(1 ETH = 100 UNI)

Several things can happen next:

(1) Bob repays loan

(2) Bob defaults on loan



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

I can't repay 1 ETH

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

debt position:

~~+ 500 UNI~~

~~− 1 ETH~~

The role of collateral

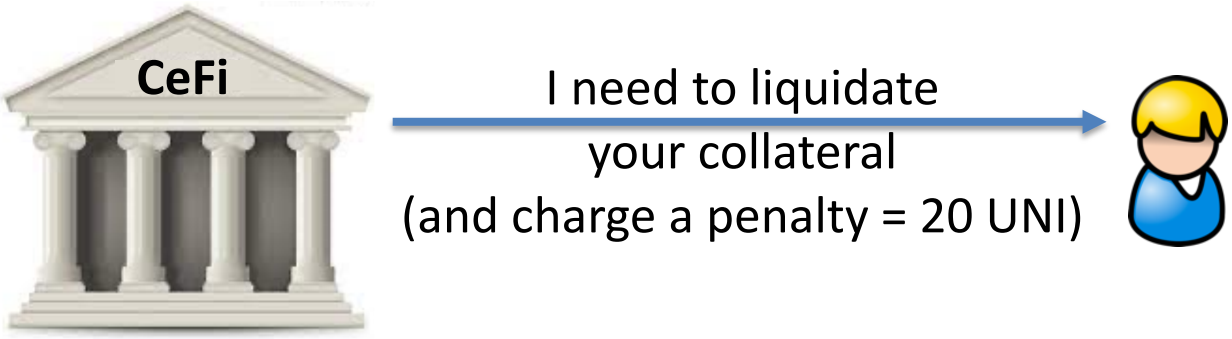
(1 ETH = 400 UNI)

Several things can happen next:

(1) Bob repays loan

(2) Bob defaults on loan

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



CeFi

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: 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)

Loan-to-Value (LTV)

Loan-to-Value (LTV) $\in [0,1]$

- Max value that can be borrowed using this collateral

- High volatility asset $\Longrightarrow$ low LTV
- Relatively stable asset $\Longrightarrow$ higher LTV

Examples: (on Aave)

LTV:	ETH: 80.5%,	USDC: 75%,	DAI: 63%
Liquidation th.:	ETH: 83%,	USDC: 78%,	DAI: 77%

Health of a debt position

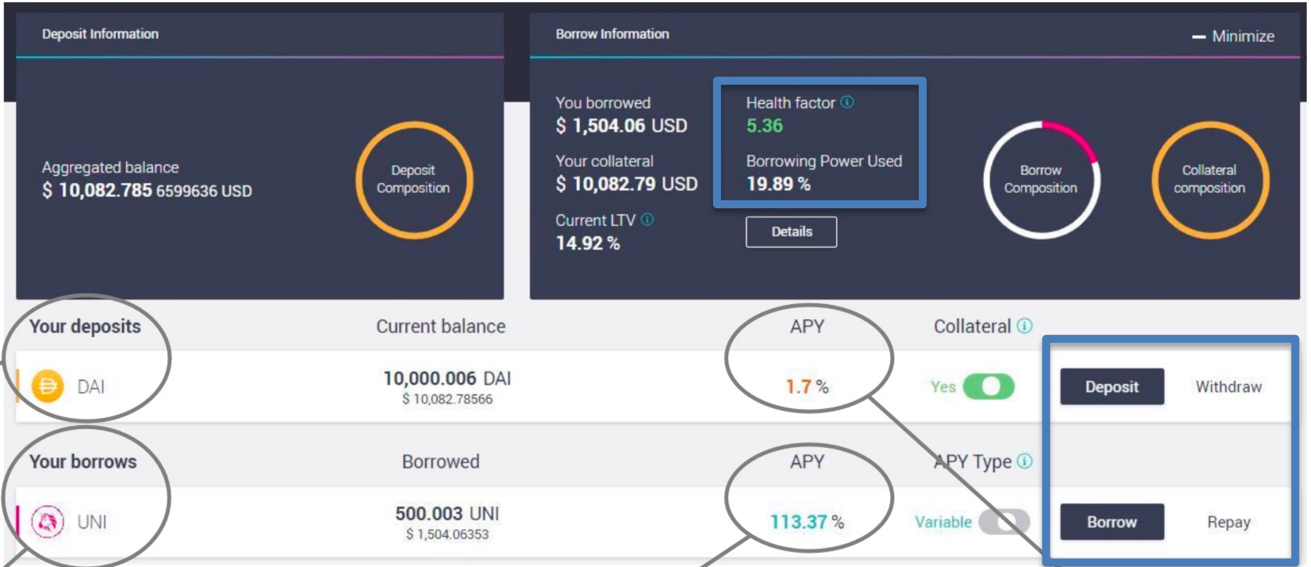
$$\text{BorrowCapacity} = \sum_i \text{value}(\text{collateral}_i) \times \text{LTV}_i$$

(in ETH)

$$\boxed{\text{health} = \frac{\sum_i \text{value}(\text{Collateral}_i) \times (\text{Liquidation Threshold}_i)}{\text{value}(\text{TotalDebt})}}$$

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

Example: Aave dashboard (a DeFi lending Dapp)



Aggregated balance
$ 10,082.785 6599636 USD

You borrowed
$ 1,504.06 USD

Your collateral
$ 10,082.79 USD

Current LTV
14.92 %

Health factor
5.36

Borrowing Power Used
19.89 %

Your deposits	Current balance	APY	Collateral
DAI	10,000.006 DAI $ 10,082.78566	1.7 %	Yes
Your borrows	**Borrowed**	**APY**	**APY Type**
UNI	500.003 UNI $ 1,504.06353	113.37 %	Variable

actions

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.

Credit: Arthur Gervais

Why borrow ETH?

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

- Bob may need ETH (e.g., to buy in-game assets), but he might not want to sell his collateral (e.g., an NFT)

- 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)

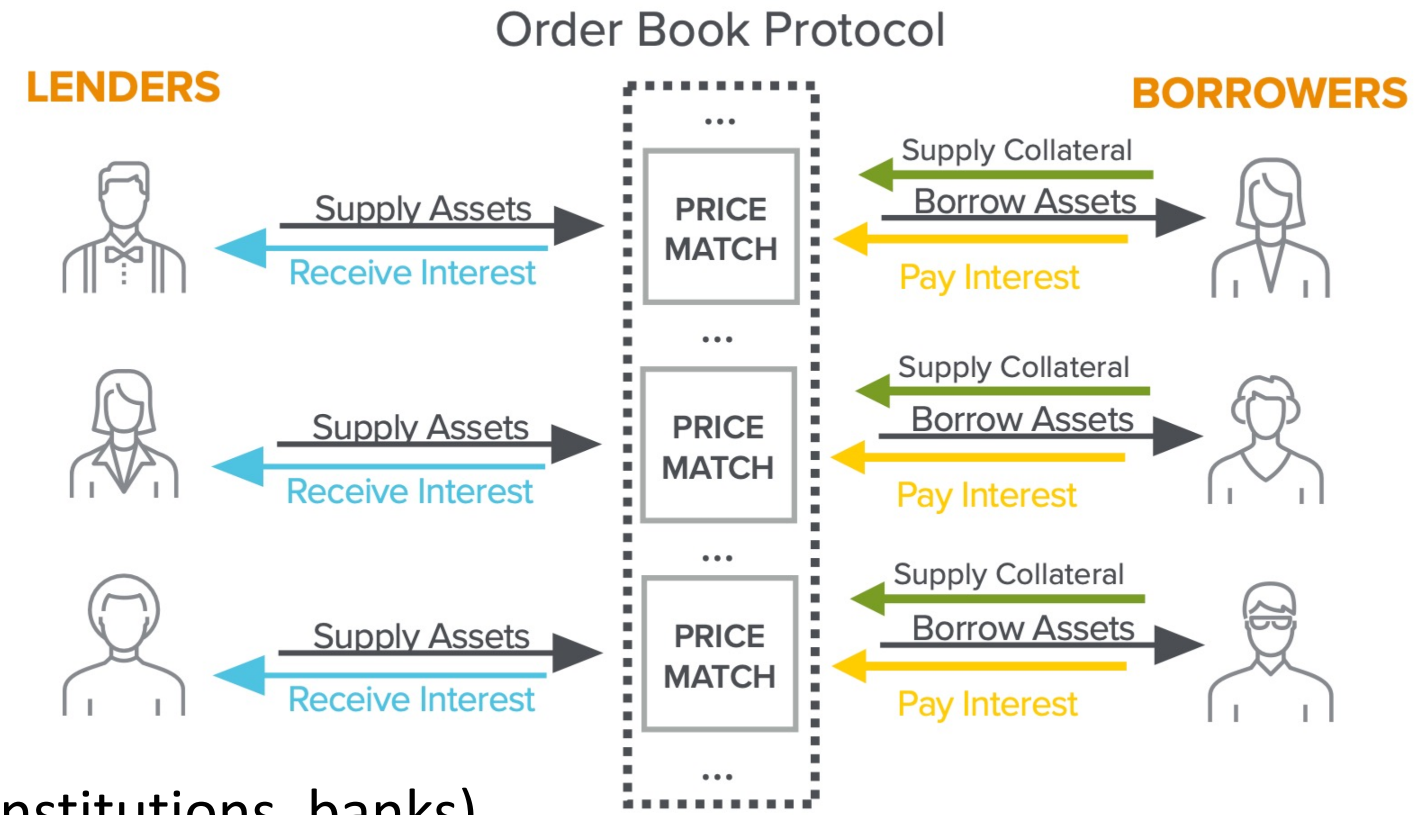
DeFi Lending

Can we build an on-chain lending Dapp?

$\Longrightarrow$ no central trusted parties

$\Longrightarrow$ code available on Ethereum for inspection

A first idea: an order book Dapp



(large institutions, banks)

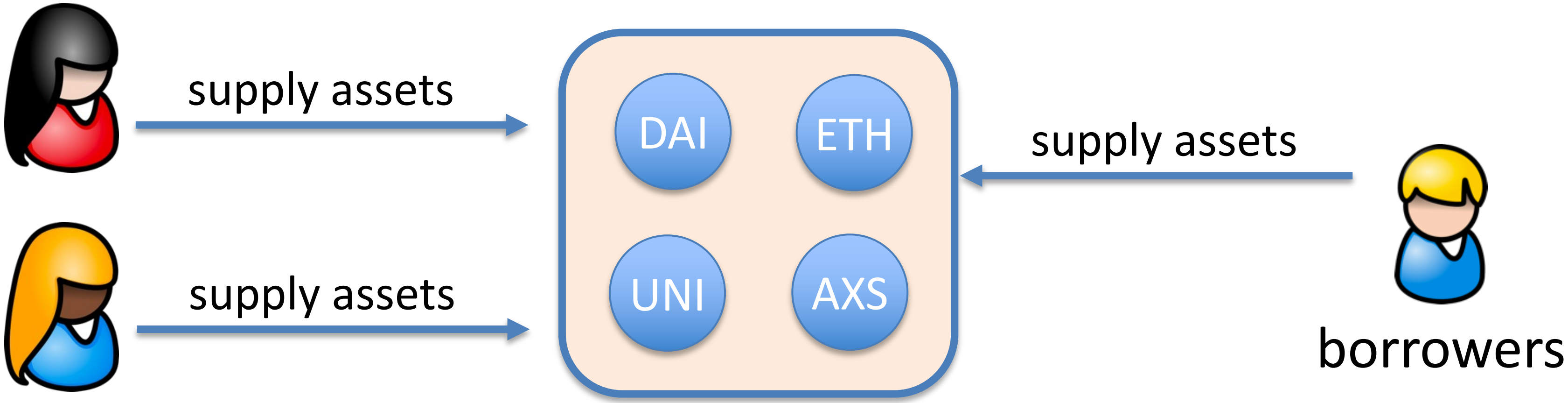
Credit: Eddy Lazzarin

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

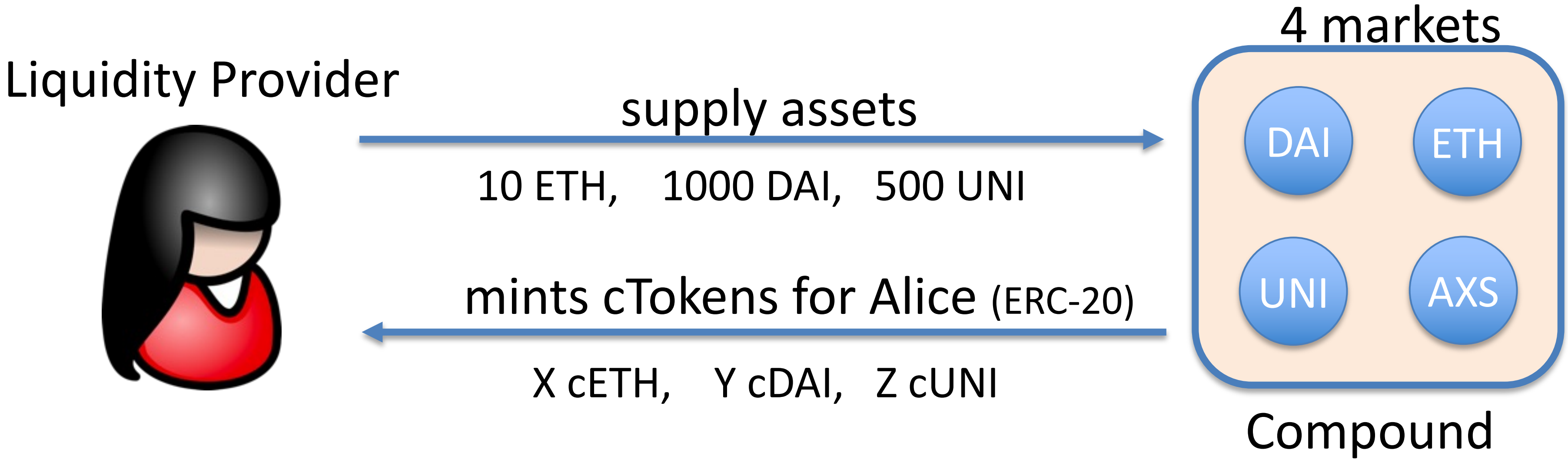
Over-collateralized lending: Compound and Aave



Liquidity Providers
(earn interest)

Compound/Aave Dapps
gather liquidity

Example: Compound cTokens



Liquidity Provider

supply assets

10 ETH, 1000 DAI, 500 UNI

mints cTokens for Alice (ERC-20)

X cETH, Y cDAI, Z cUNI

4 markets

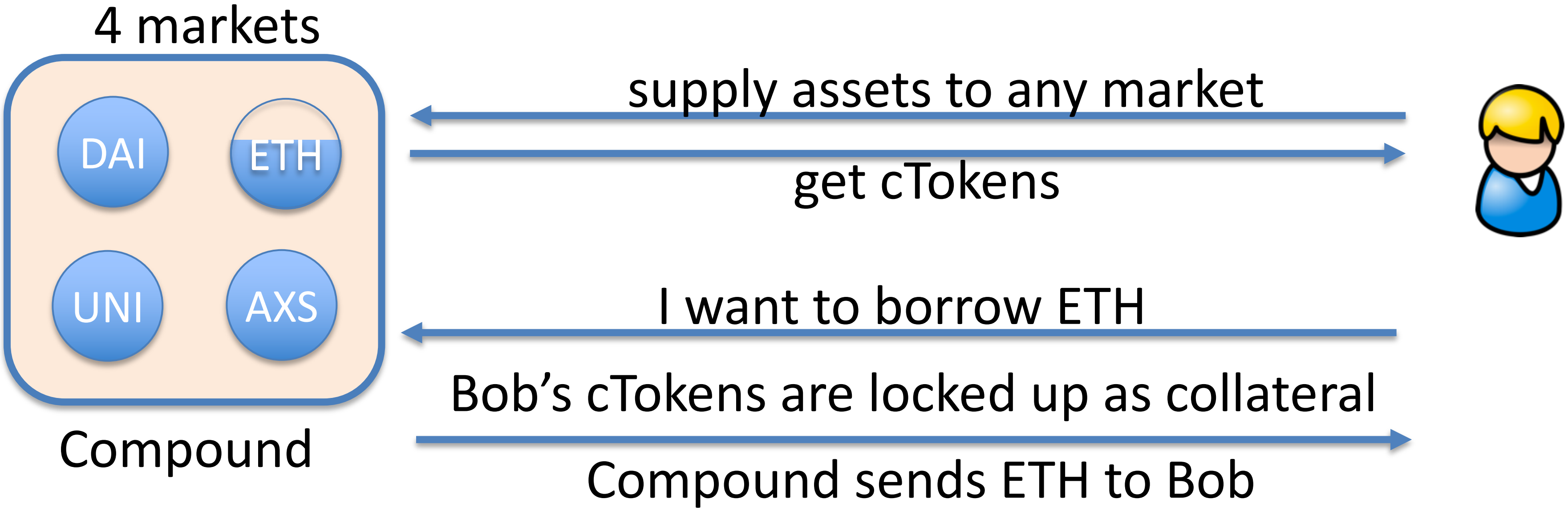
DAI ETH

UNI AXS

Compound

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

Borrowers



4 markets

Compound

supply assets to any market

get cTokens

I want to borrow ETH

Bob’s cTokens are locked up as collateral

Compound sends ETH to Bob

Bob’s accrued interest increases ETH/cETH exchange rate

⟹ benefit cETH token holders (ETH liquidity providers)

The interest rate: constantly updates

Key idea: determined by demand for asset vs. asset market size

Utilization ratio: $$\mathbf{U}_{\mathbf{ETH}} = \frac{\text{totalBorrowBalance}_{\text{ETH}}}{\text{availableBalance}_{\text{ETH}} + \text{totalBorrowBalance}_{\text{ETH}}}$$

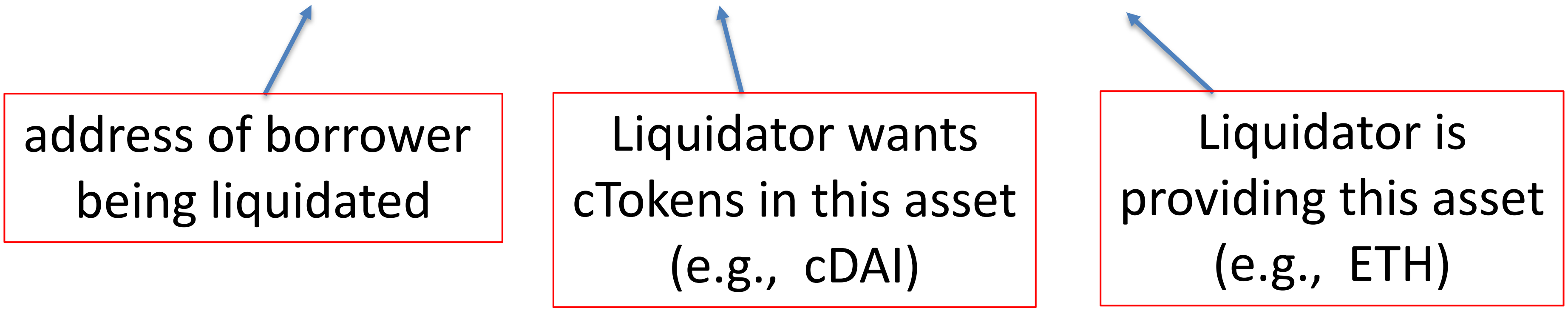
higher totalBorrowBalance, or
lower availableBalance in contract → higher $\mathbf{U}_{\mathbf{ETH}} \in [0,1]$

$$\text{interestRate}_{\text{ETH}} = \text{BaseRate}_{\text{ETH}} + \mathbf{U}_{\mathbf{ETH}} \times \text{slope}_{\text{ETH}}$$

Liquidation: debt > BorrowCapacity

If user's health < 1 then **anyone** can call:

liquidate(borrower, CollateralAsset, BorrowAsset, uint amount)



address of borrower being liquidated

Liquidator wants cTokens in this asset (e.g., cDAI)

Liquidator is providing this asset (e.g., ETH)

This function transfers liquidator's ETH into ETH market,
and gives the liquidator cDAI from user's collateral

Liquidation: debt > BorrowCapacity

If user's health < 1 then **anyone** can call:

lic t)

add

b

(e.g., cDAI)

set

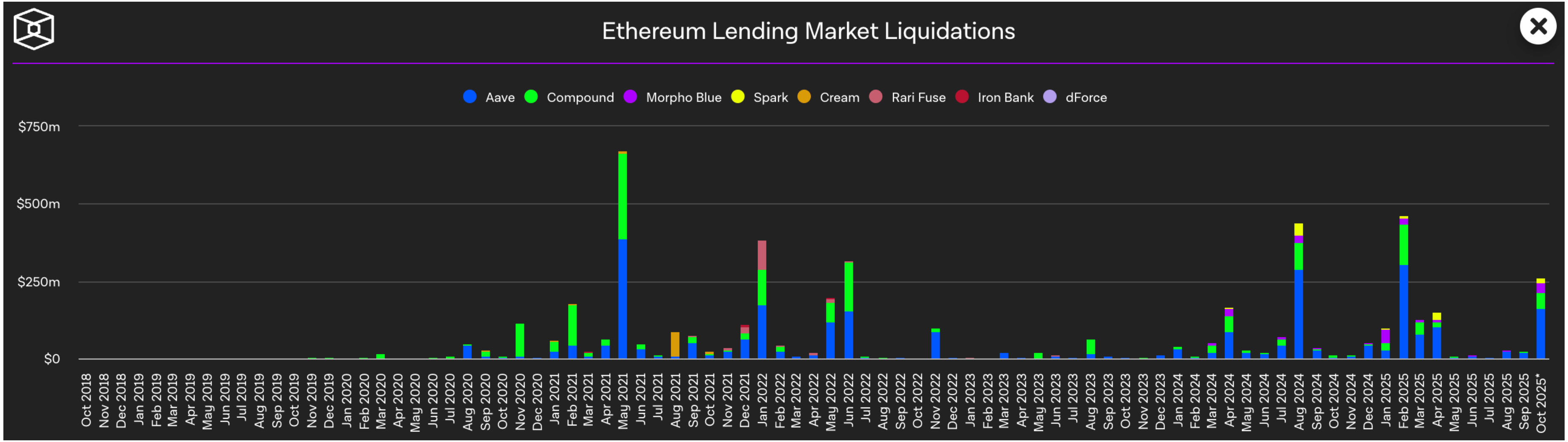
(e.g., ETH)

Liquidator is repaying the user's ETH debt
and getting the user's cDAI
[at a discounted exchange rate -- penalty for user]

This function transfers liquidator's ETH into ETH market,
and gives the liquidator cDAI from user's collateral

This happens …

Liquidations on Ethereum:



Caused by collateral price drops or debt APR spikes

What is liquidation risk?

Historical UNI interest rate on Aave (APR):

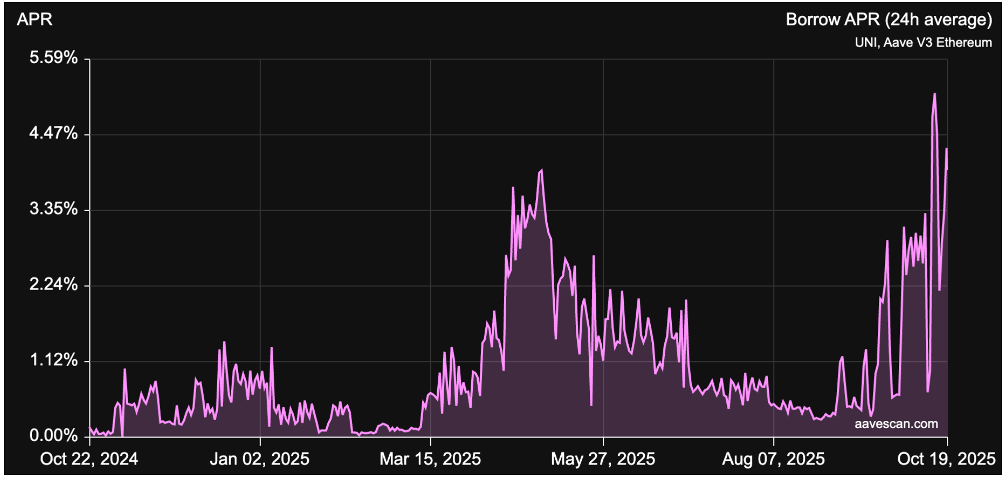
Demand for UNI spikes

⟹ price of UNI spikes

⟹ user's debt shoots up

⟹ user's health drops

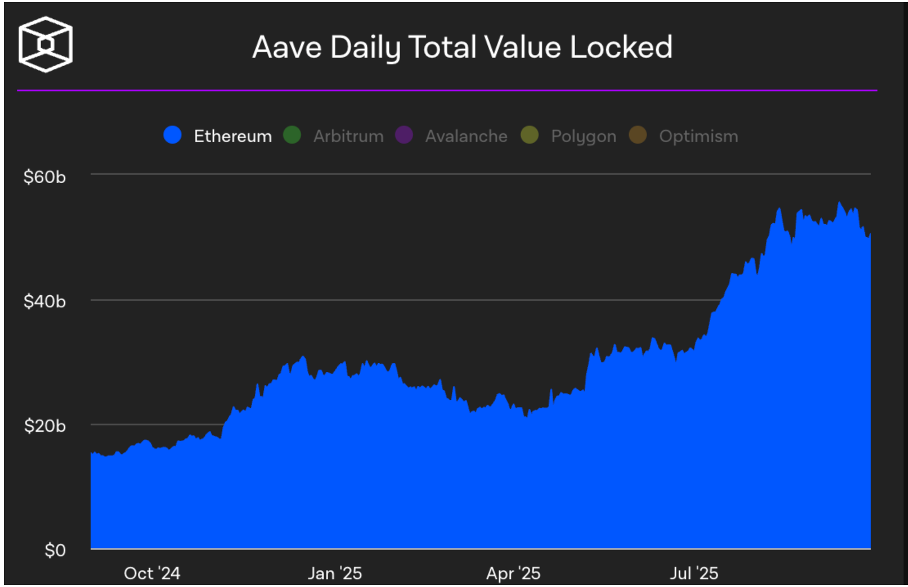
⟹ liquidation …

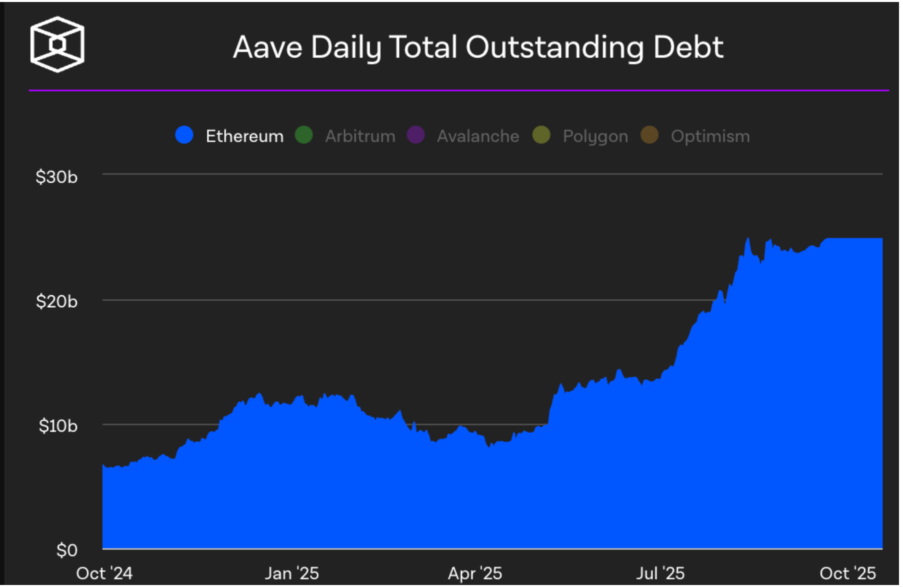


Borrowers must constantly monitor the health of their position, and quickly repay loans if it drops (several services automate this)

Summary & stats

- Liquidity providers can earn interest on their assets
- DeFi lending usage (on Ethereum):


Aave Daily Total Value Locked

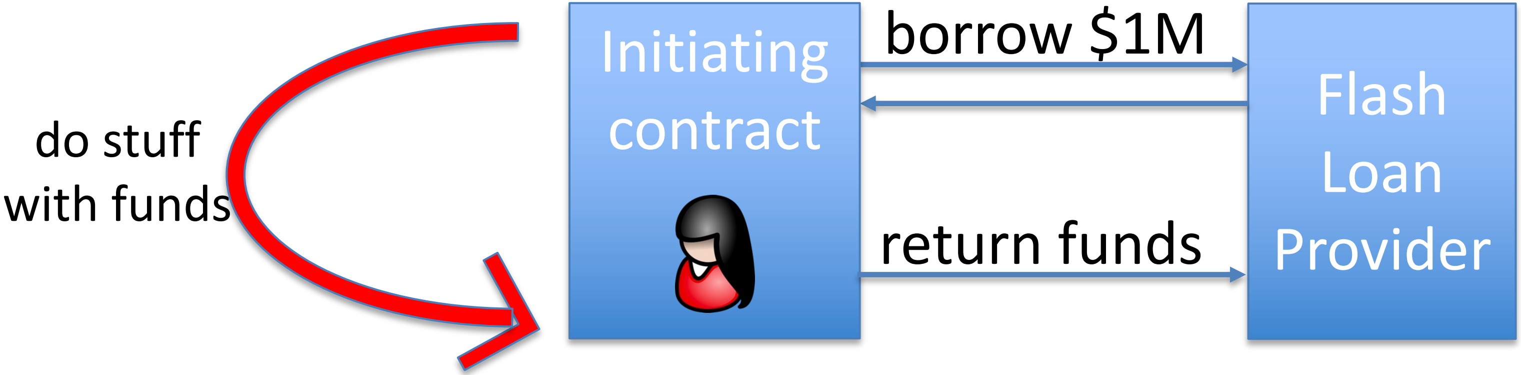

Aave Daily Total Outstanding Debt

Flash loans

What is a flash loan?

A flash loan is taken and repaid in a single transaction

⟹ zero risk for lender ⟹ 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"

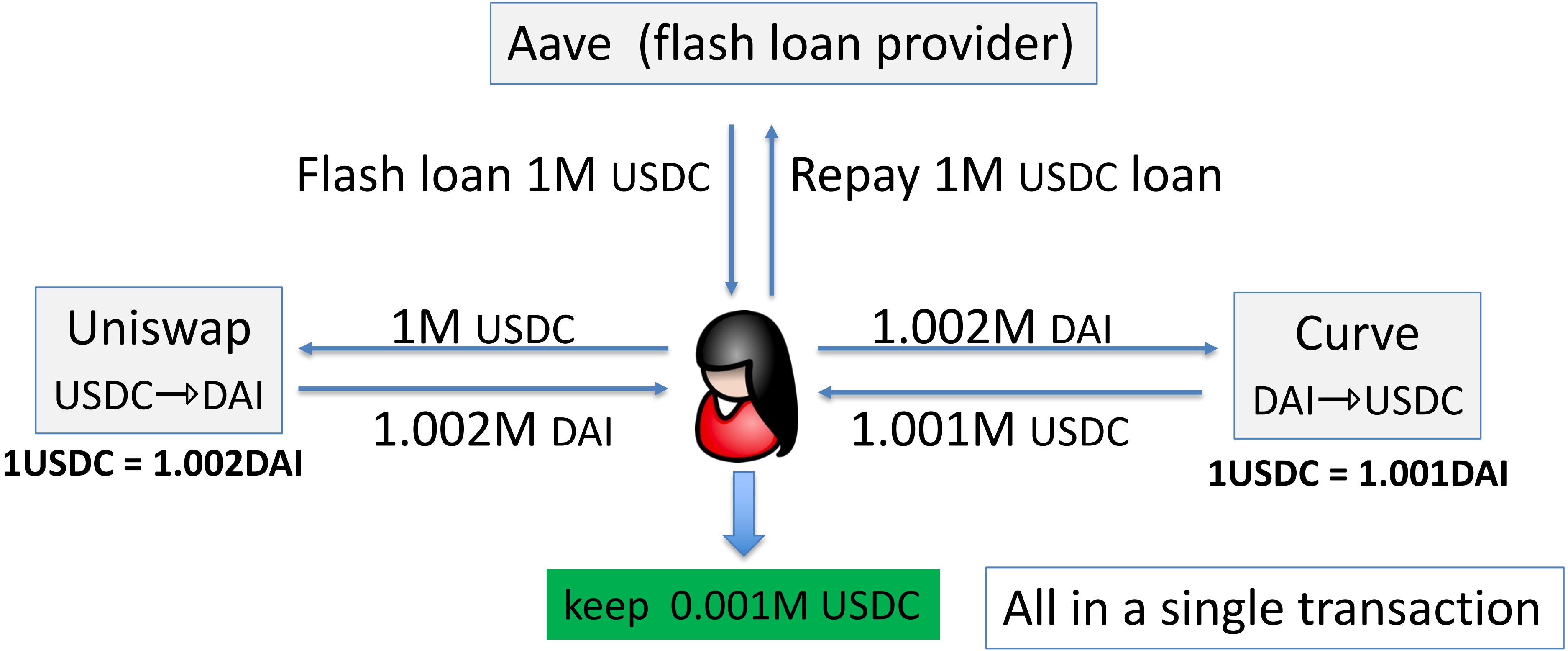
Use cases

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

⋮

Risk free arbitrage

Alice finds a USDC/DAI price difference in two pools



Aave (flash loan provider)

Flash loan 1M USDC

Repay 1M USDC loan

Uniswap

USDC→DAI

1USDC = 1.002DAI

1M USDC

1.002M DAI

1.002M DAI

1.001M USDC

Curve

DAI→USDC

1USDC = 1.001DAI

keep 0.001M USDC

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
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)

end goal:

Alice @Compound

-1000 DAI
+3000 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");
}
```

END OF LECTURE

Next lecture: Decentralized Exchanges (DeX)