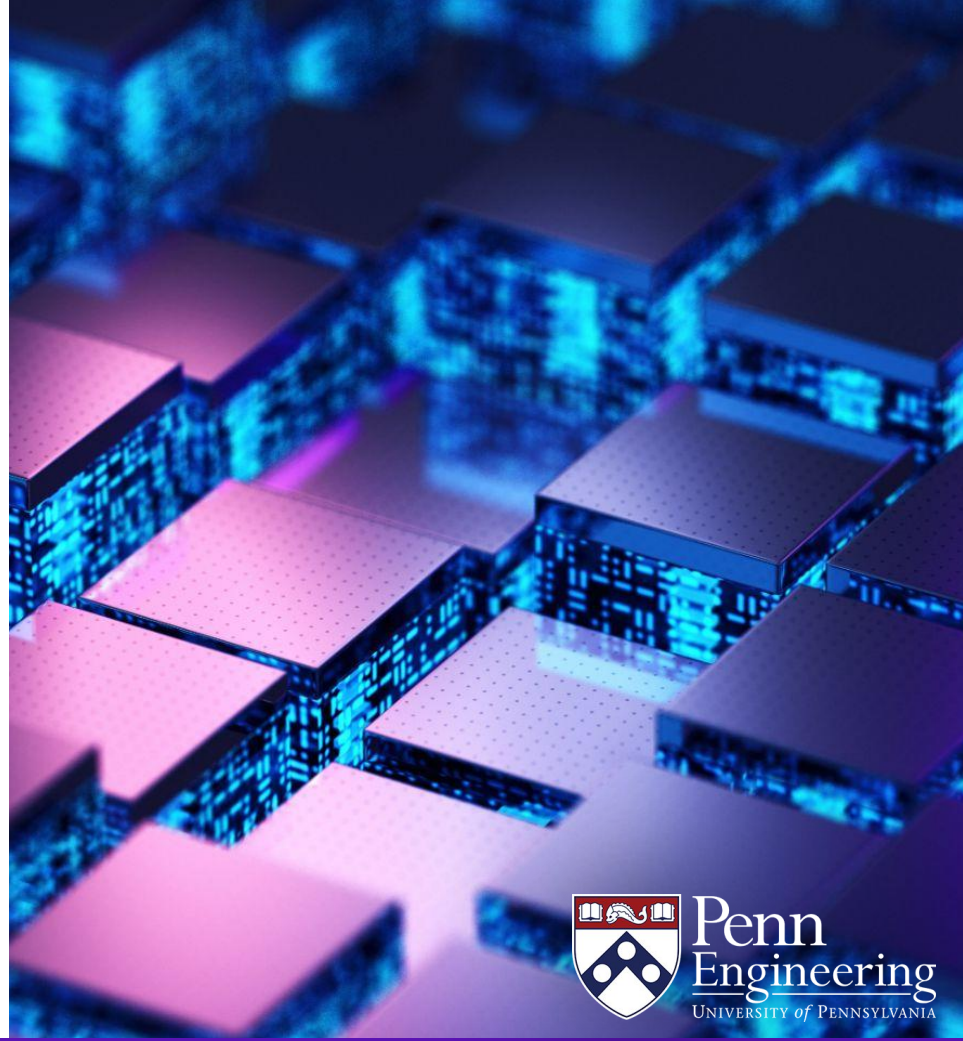
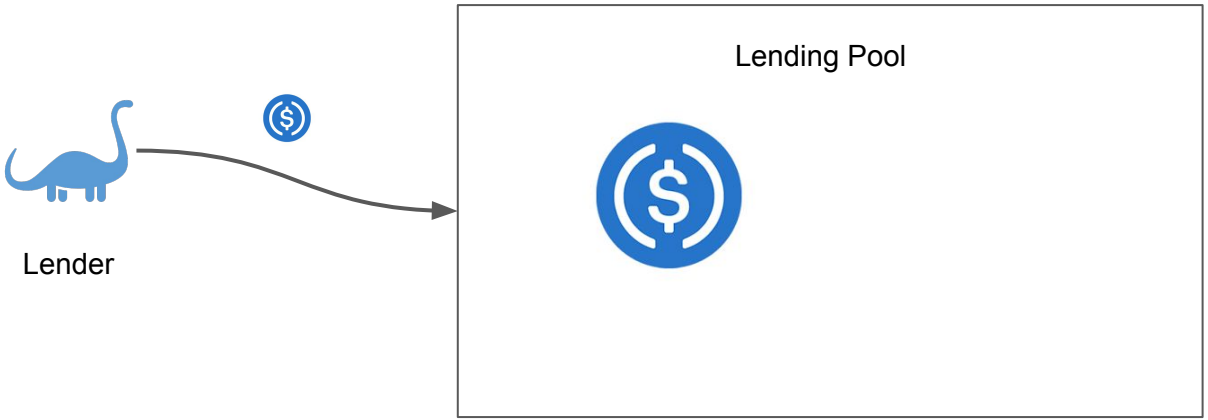
EAS 5830: BLOCKCHAINS

Aave

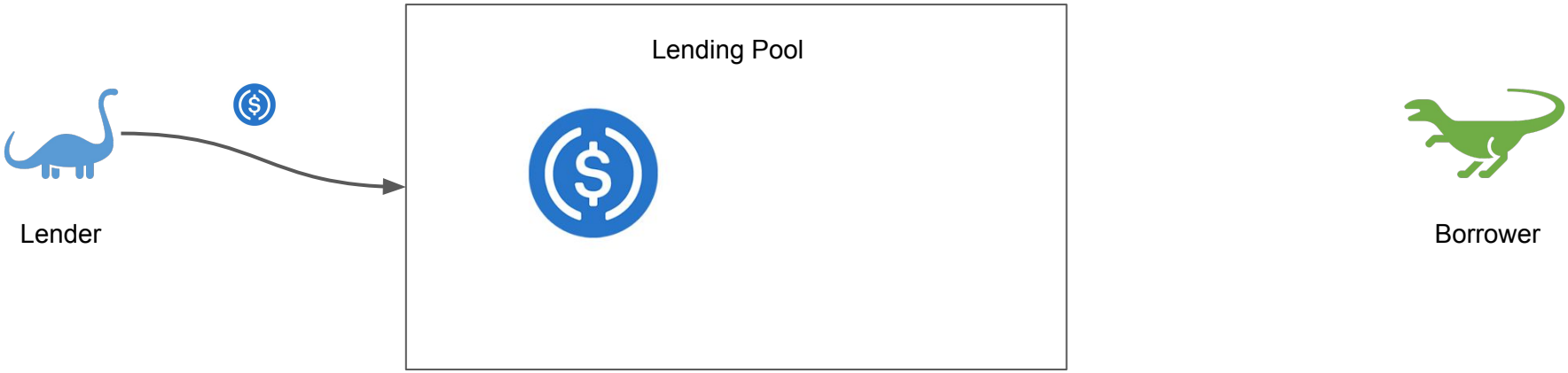
Professor Brett Hemenway Falk



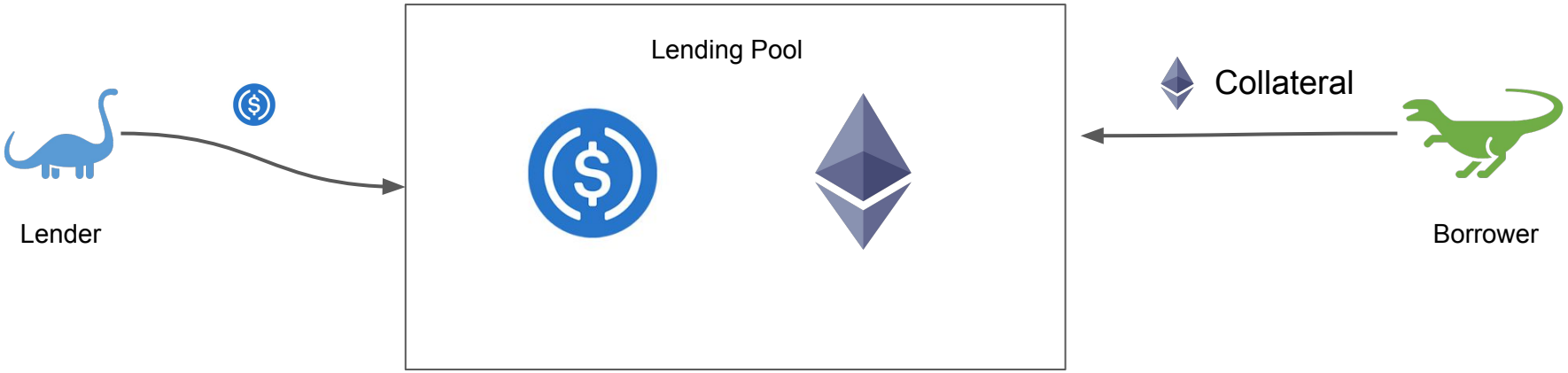
On-chain lending



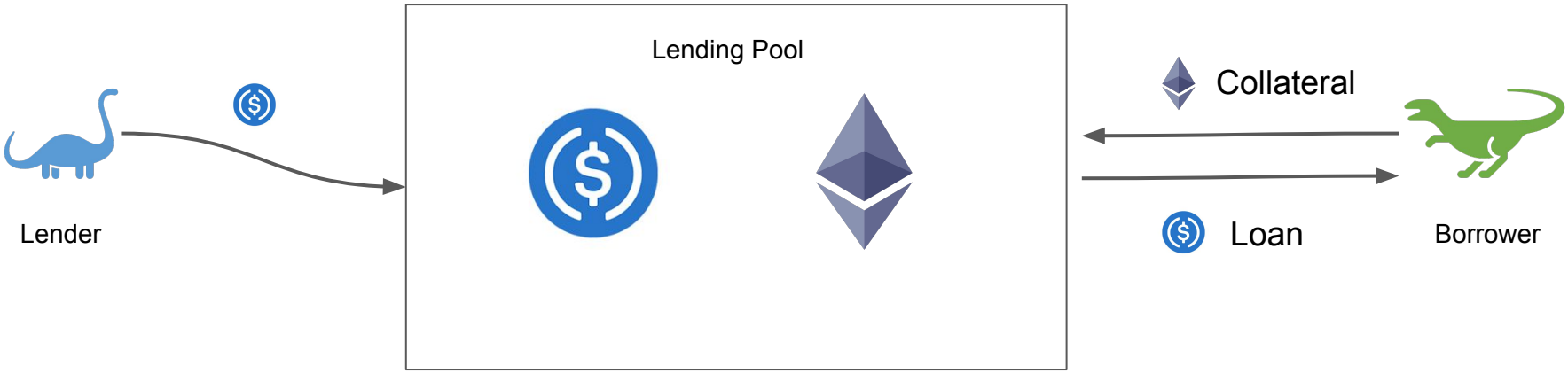
On-chain lending



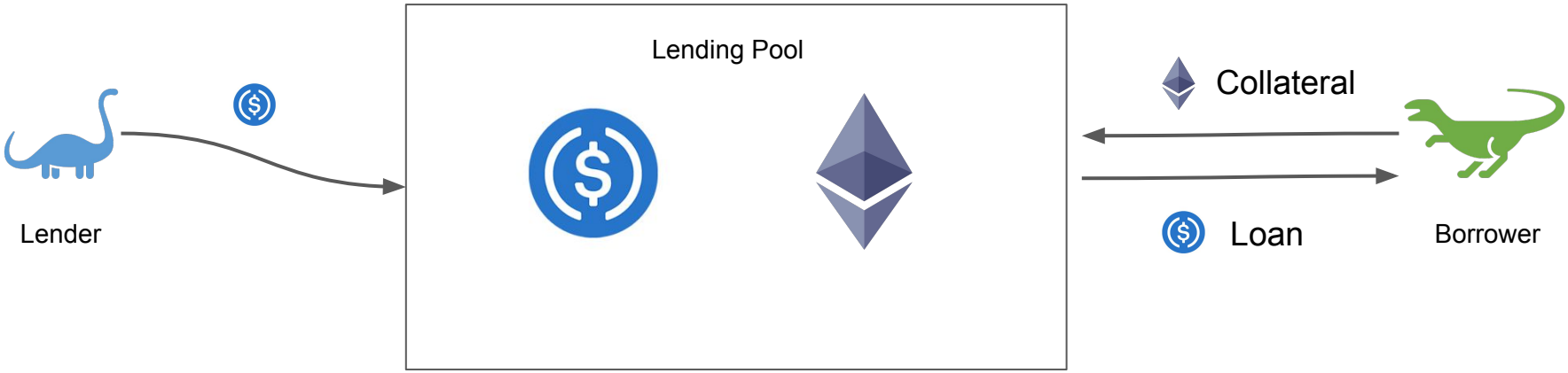
On-chain lending



On-chain lending



On-chain lending

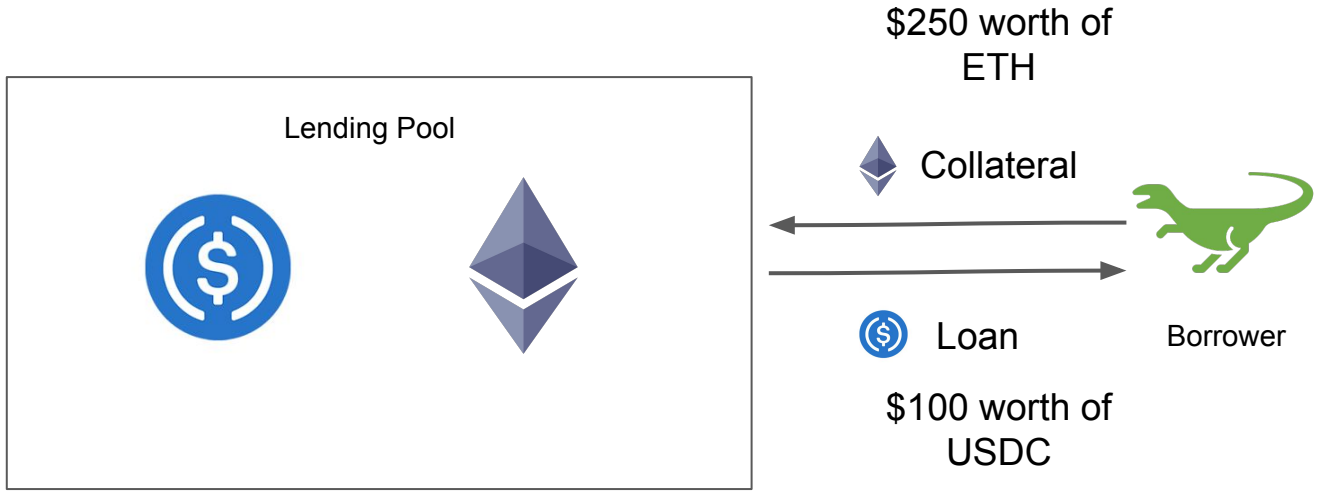


There's no way to go after defaulters, so all loans are over-collateralized

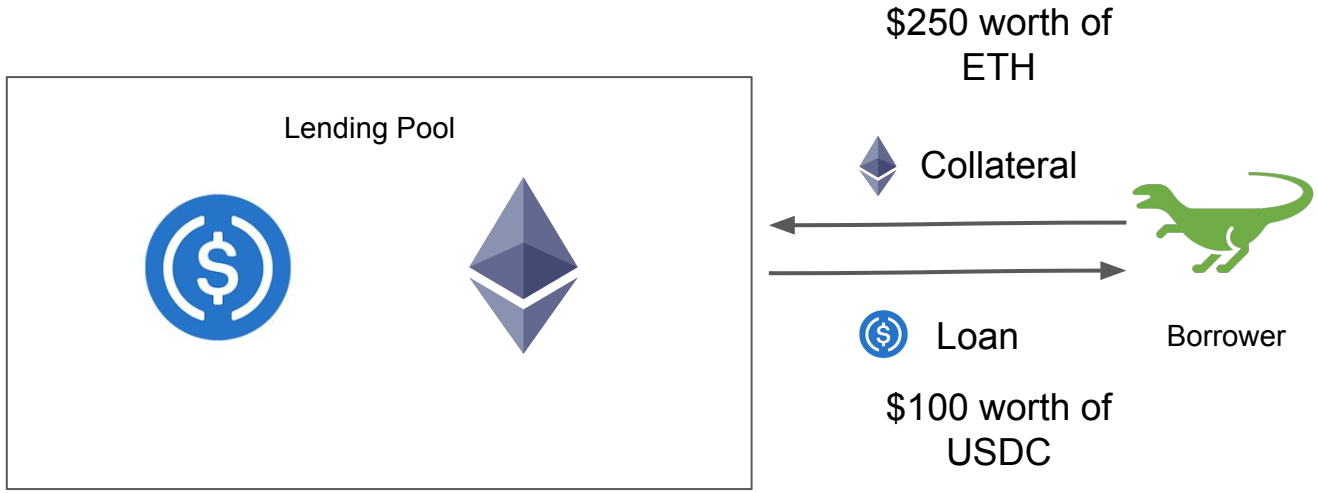
Collateralization and debt collection

- No background checks
- All loans are over-collateralized
  - Collateralization ratio is set by the protocol
- If your collateral drops below a liquidation threshold, your collateral is sold to repay your loan
- There is no other form of debt-collection

On-chain lending

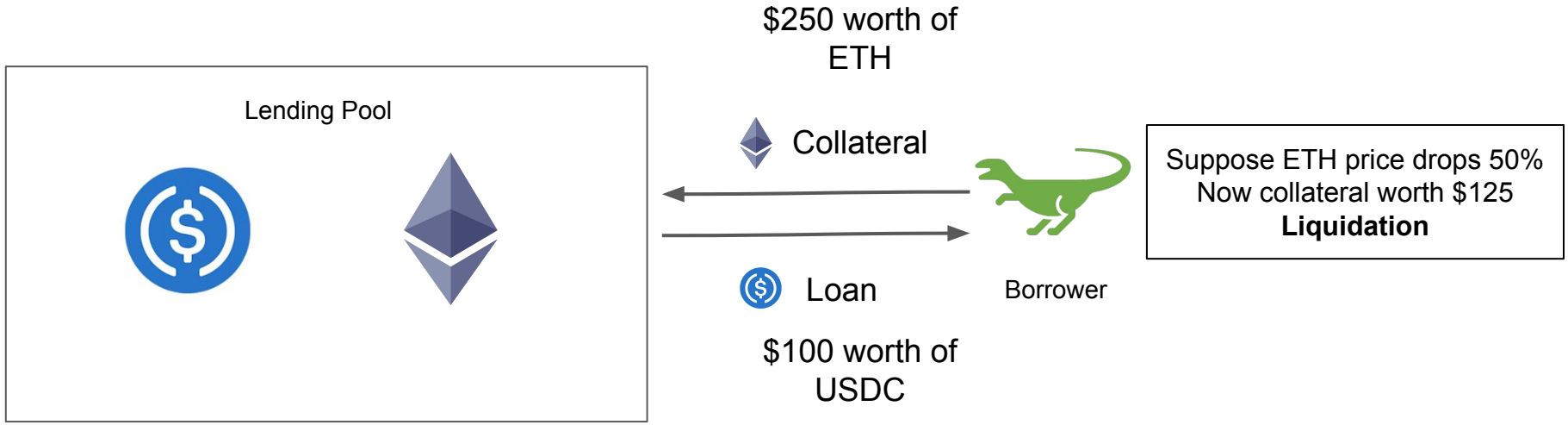


On-chain lending

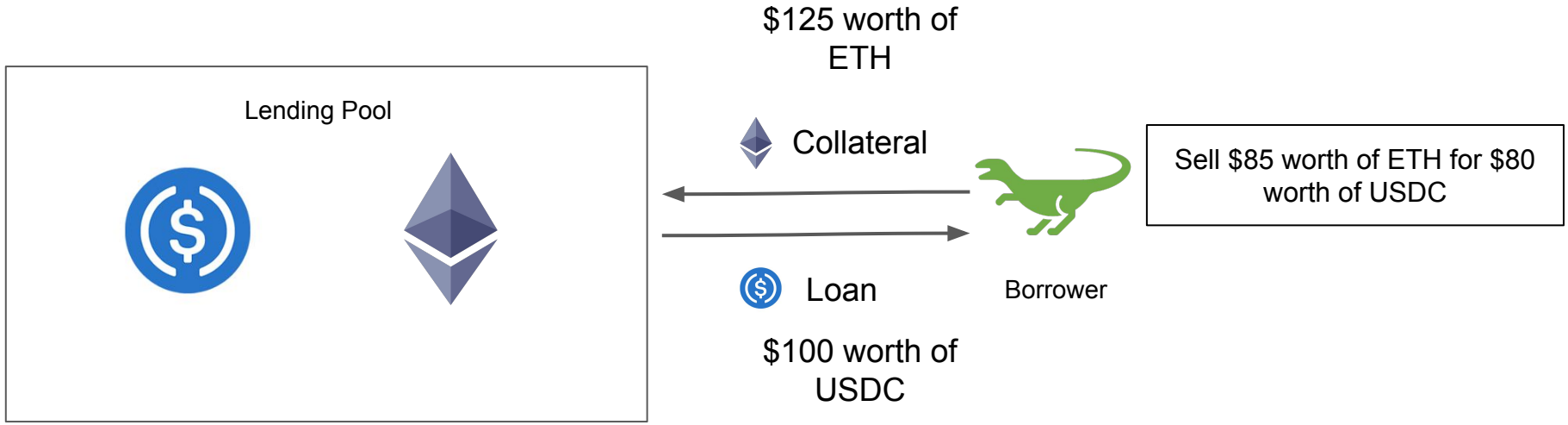


Minimum collateralization ratio is 2:1

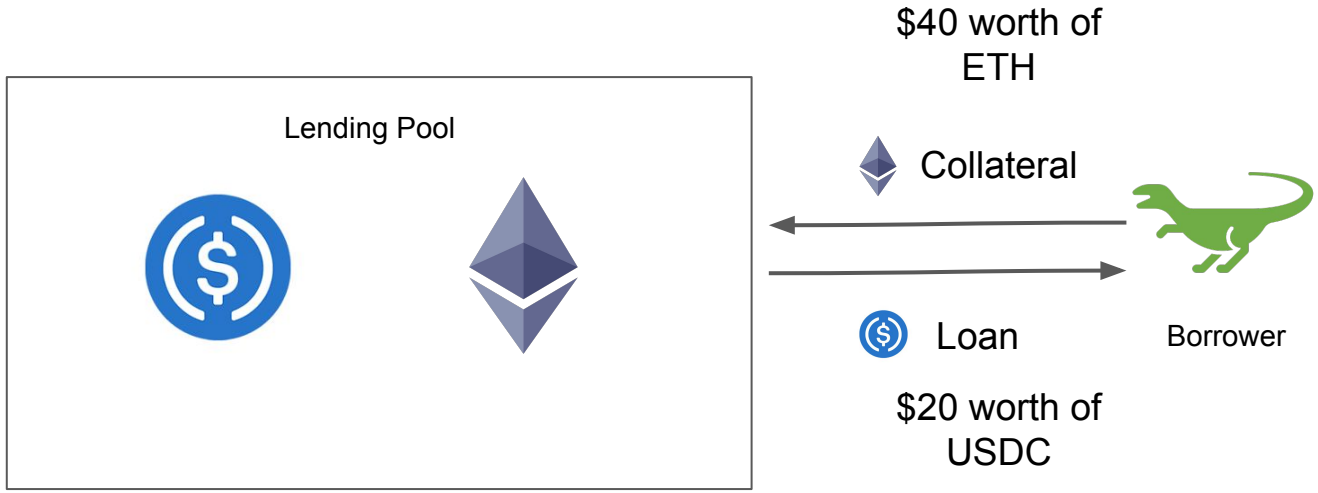
On-chain lending



On-chain lending



On-chain lending



Protocol risk

- Protocol loses money if collateral value drops faster than it can be sold
- Very rarely happens
  - Black Thursday 2020

Aave lending

- Loans are indefinite
- Interest is only paid when you repay your loan
  - No monthly payments
- Interest is paid in borrowed asset

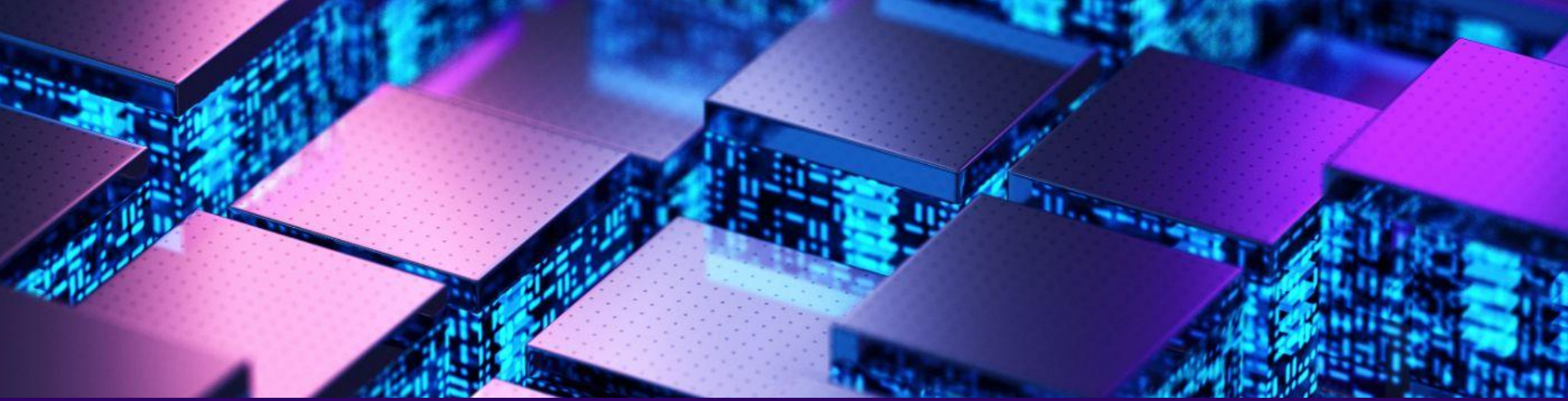
Collateralization

Collateral usage ✓ Can be collateral

Max LTV	Liquidation threshold	Liquidation penalty
80.50 %	83.00 %	5.00 %

Collateral usage ✓ Can be collateral

Max LTV	Liquidation threshold	Liquidation penalty
77.00 %	80.00 %	4.50 %



Why Borrow?

Why Borrow

- Get cash from illiquid assets
  - The “Buy, Borrow, Die” strategy is also common outside the blockchain space
- Get leverage

Curve Finance CEO and wife purchase two mansions in Australia worth around $40 million

by Yogita Khatri

PEOPLE • MAY 29, 2023, 8:06AM EDT

Egorov now has $132 million worth of collateral and $42 million in debt across all other DeFi lenders.

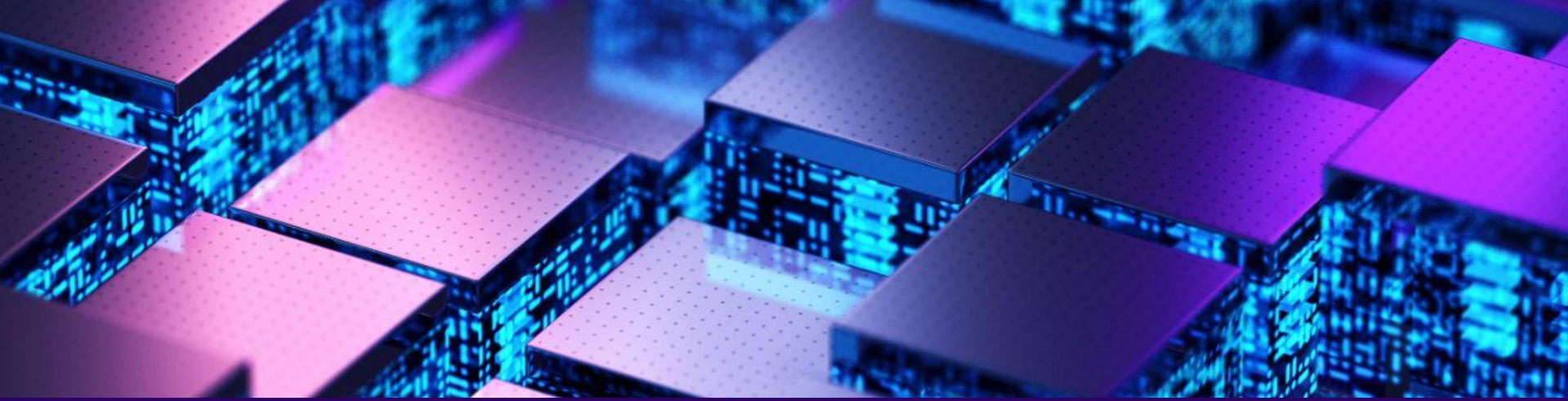
By Oliver Knight Sep 27, 2023 at 8:05 a.m. EDT Updated Sep 28, 2023 at 11:33 a.m. EDT

Leverage

1. Deposit ETH
2. Borrow USDC
3. Used the USDC to buy ETH
4. Wait for the price of ETH to increase

Leverage

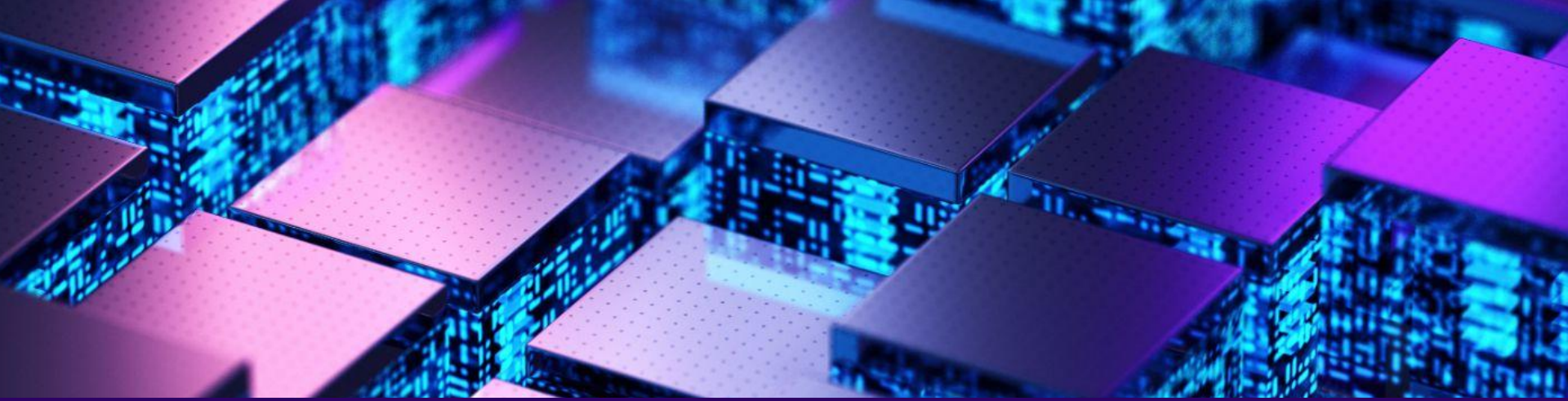
- Put up $100 worth of ETH
- Borrow $50
- Buy $50 worth of ETH
- Now you have $150 worth of ETH exposure
- 1.5x leverage
- If price of ETH doubles, sell ETH for $300
- Pay off $50 loan
- $150 Profit (compared to $100 without loan)



Price Oracles

Price oracles

- Common value needed for
  - Loan-to-value
  - Liquidations
- Aave uses Chainlink to provide price feeds
  - V1
  - V2
  - V3



Interest rates

The interest rate R_t follows the model:

$$if\ U \leq U_{optimal}: \qquad R_t = R_0 + \frac{U_t}{U_{optimal}} R_{slope1}$$

$$if\ U > U_{optimal}: \qquad R_t = R_0 + R_{slope1} + \frac{U_t - U_{optimal}}{1 - U_{optimal}} R_{slope2}$$

The interest rate R_t follows the model:

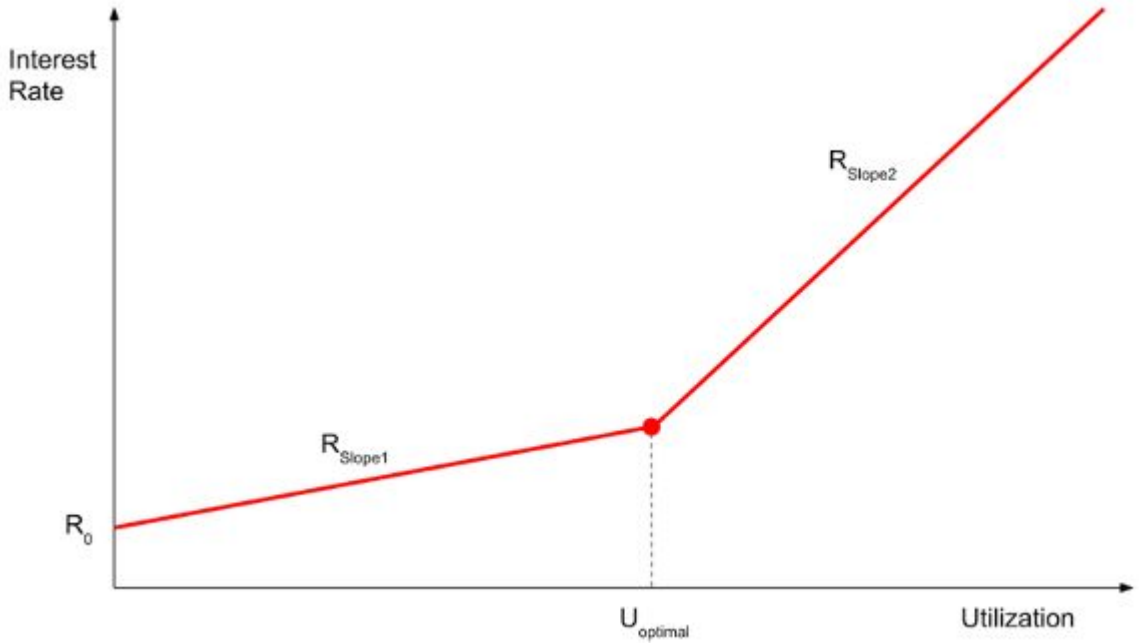
$$if\ U \leq U_{optimal}: \qquad R_t = R_0 + \frac{U_t}{U_{optimal}} R_{slope1}$$

$$if\ U > U_{optimal}: \qquad R_t = R_0 + R_{slope1} + \frac{U_t - U_{optimal}}{1 - U_{optimal}} R_{slope2}$$

The interest rate R_t follows the model:

$$if\ U \leq U_{optimal}: \quad R_t = R_0 + \frac{U_t}{U_{optimal}} R_{slope1}$$

$$if\ U > U_{optimal}: \quad R_t = R_0 + R_{slope1} + \frac{U_t - U_{optimal}}{1 - U_{optimal}} R_{slope2}$$


Interest
Rate
R Slope2
R Slope1
R 0
U optimal
Utilization

DAI

Parameters	Value
Optimal Usage	90%
Base Variable Borrow Rate	0
Variable Rate Slope 1	4%
Variable Rate Slope 2	60%
Base Stable Borrow Rate	2%
Stable Rate Slope 1	0.5%
Stable Rate Slope 2	60%
Optimal Stable to Total Debt Ratio	20%

DAI

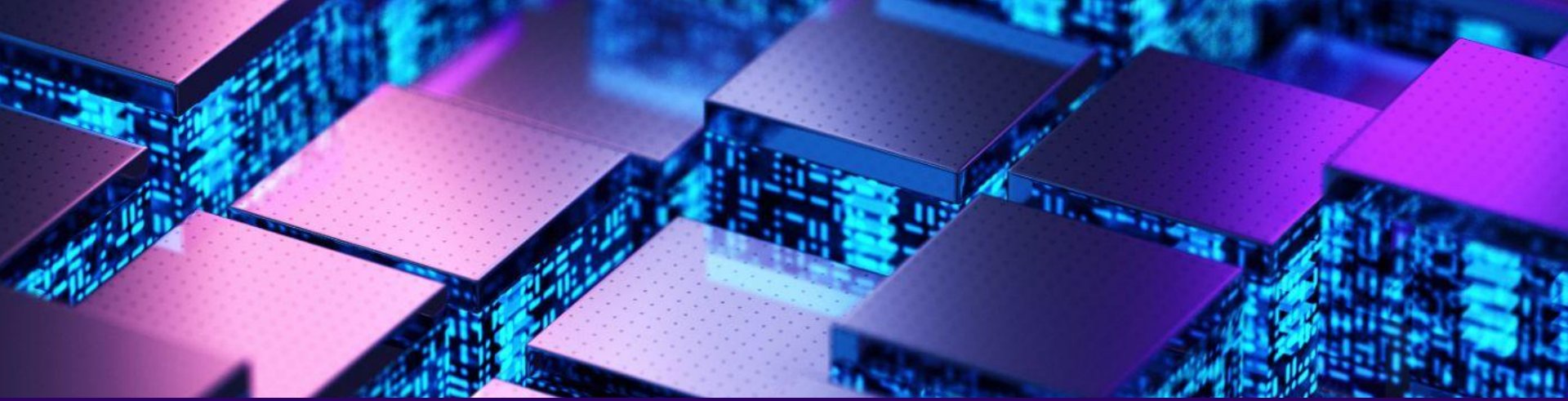
Parameters	Value
Optimal Usage	90%
Base Variable Borrow Rate	0
Variable Rate Slope 1	4%
Variable Rate Slope 2	60%
Base Stable Borrow Rate	2%
Stable Rate Slope 1	0.5%
Stable Rate Slope 2	60%
Optimal Stable to Total Debt Ratio	20%

DAI

Parameters	Value
Optimal Usage	90%
Base Variable Borrow Rate	0
Variable Rate Slope 1	4%
Variable Rate Slope 2	60%
Base Stable Borrow Rate	2%
Stable Rate Slope 1	0.5%
Stable Rate Slope 2	60%
Optimal Stable to Total Debt Ratio	20%

DAI

Parameters	Value
Optimal Usage	90%
Base Variable Borrow Rate	0
Variable Rate Slope 1	4%
Variable Rate Slope 2	60%
Base Stable Borrow Rate	2%
Stable Rate Slope 1	0.5%
Stable Rate Slope 2	60%
Optimal Stable to Total Debt Ratio	20%

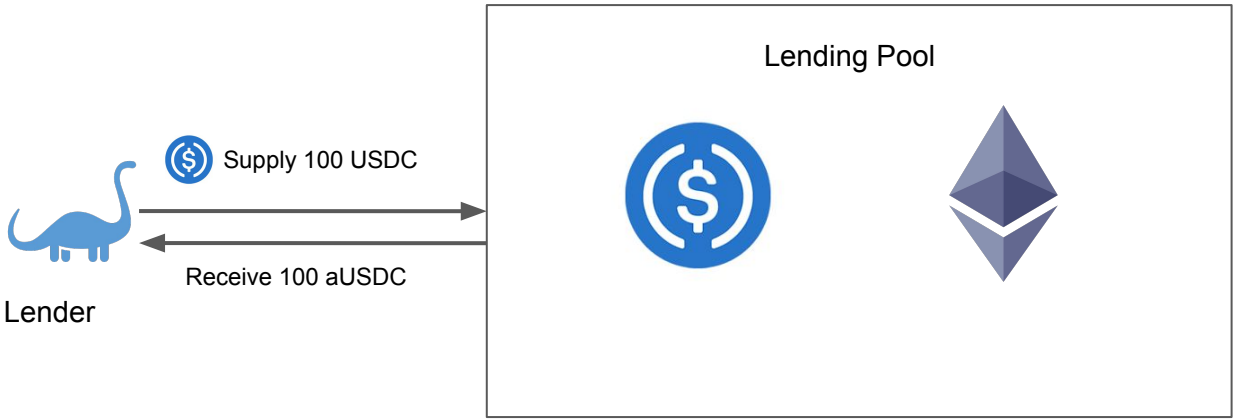


Accounting

Accounting with aTokens

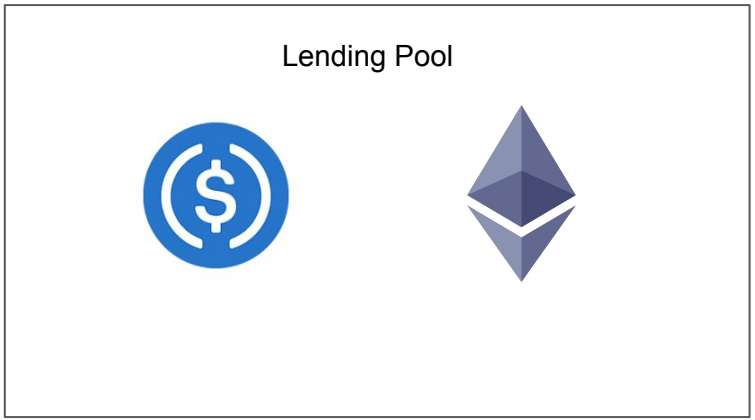
- When you supply, you get aTokens
  - Deposit WETH, get aWETH
  - Deposit USDC, get aUSDC
- When you withdraw, you burn aTokens and get underlying asset
- aTokens are ERC-20 tokens, so you can transfer them
  - Price should track underlying asset

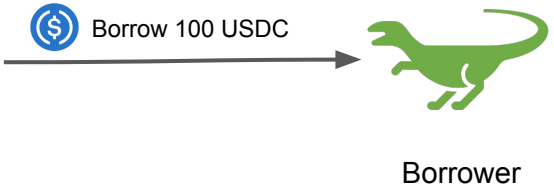
On-chain lending



On-chain lending

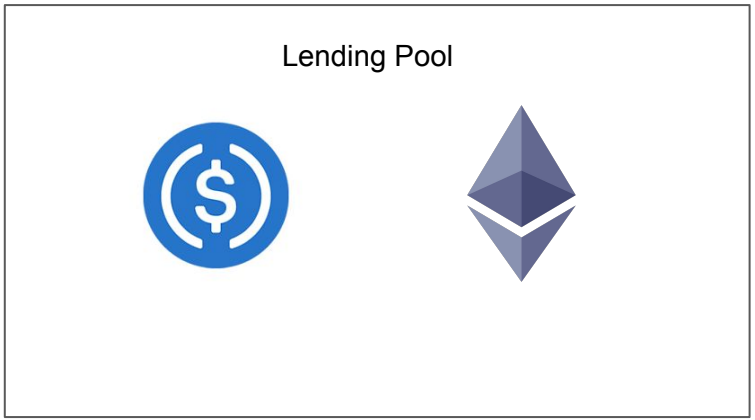
Lender

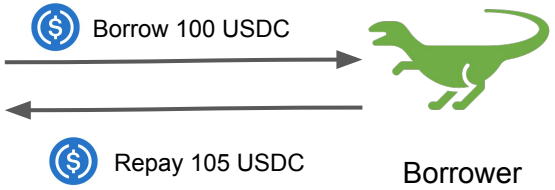




On-chain lending

Lender

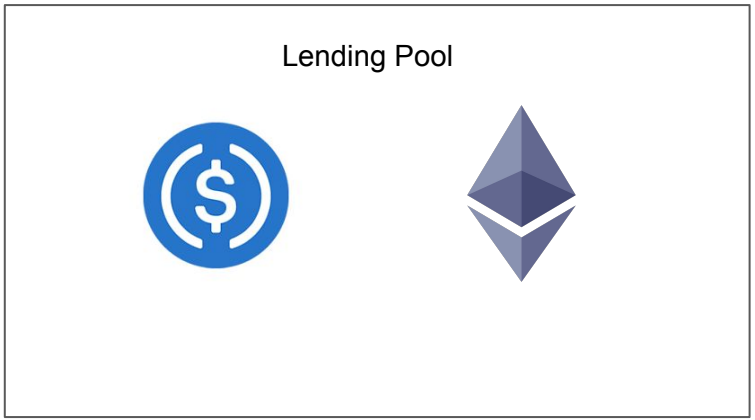




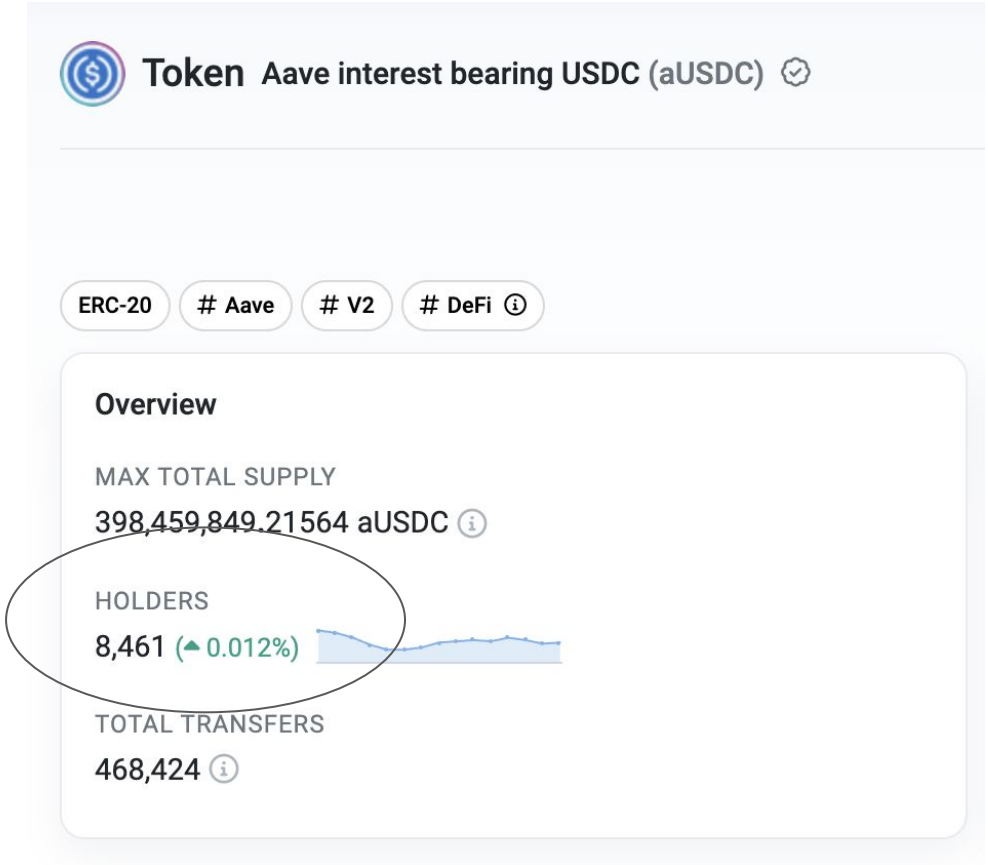
On-chain lending

Lender

102
aUSDC



Borrower


Token Aave interest bearing USDC (aUSDC)
ERC-20
Aave
V2
DeFi
Overview
MAX TOTAL SUPPLY
398,459,849.21564 aUSDC
HOLDERS
8,461 (0.012%)
TOTAL TRANSFERS
468,424

Paying interest on-chain

- Paying interest to thousands of aToken holders would be extremely gas intensive
  - Pay out interest infrequently?
    - Different token holders deposited at different times
- Fix a start time t_o
  - Keep a running tally of the percentage paid since t_o
  - When someone supplies, rescale their deposit, as if they deposited at t_o
    - If they supply more, this is also rescaled
    - One value per lender
  - When someone withdraws pay out the amount with interest
  - No on-chain interest payments!

Scaling balances

Deposit 100
USDC

t_o

Scaling balances

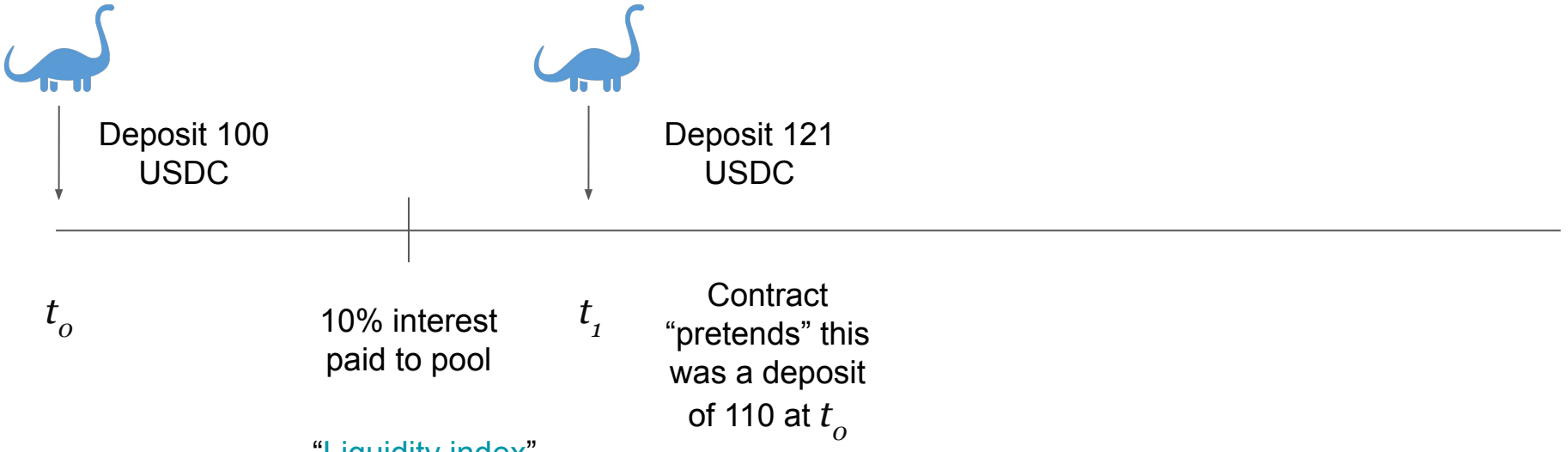
Deposit 100
USDC

t_o

10% interest
paid to pool

"Liquidity index"
is 1.1

Scaling balances



```
66      function _mintScaled(
67        address caller,
68        address onBehalfOf,
69        uint256 amount,
70        uint256 index
71      ) internal returns (bool) {
72        uint256 amountScaled = amount.rayDiv(index);
73        require(amountScaled != 0, Errors.INVALID_MINT_AMOUNT);
74
75        uint256 scaledBalance = super.balanceOf(onBehalfOf);
76        uint256 balanceIncrease = scaledBalance.rayMul(index) -
77          scaledBalance.rayMul(_userState[onBehalfOf].additionalData);
78
79        _userState[onBehalfOf].additionalData = index.toUint128();
80
81        _mint(onBehalfOf, amountScaled.toUint128());
82
83        uint256 amountToMint = amount + balanceIncrease;
84        emit Transfer(address(0), onBehalfOf, amountToMint);
85        emit Mint(caller, onBehalfOf, amountToMint, balanceIncrease, index);
86
87        return (scaledBalance == 0);
88      }
```

```
66      function _mintScaled(
67        address caller,
68        address onBehalfOf,
69        uint256 amount,
70        uint256 index
71      ) internal returns (bool) {
72        uint256 amountScaled = amount.rayDiv(index);
73        require(amountScaled != 0, Errors.INVALID_MINT_AMOUNT);
74
75        uint256 scaledBalance = super.balanceOf(onBehalfOf);
76        uint256 balanceIncrease = scaledBalance.rayMul(index) -
77          scaledBalance.rayMul(_userState[onBehalfOf].additionalData);
78
79        _userState[onBehalfOf].additionalData = index.toUint128();
80
81        _mint(onBehalfOf, amountScaled.toUint128());
82
83        uint256 amountToMint = amount + balanceIncrease;
84        emit Transfer(address(0), onBehalfOf, amountToMint);
85        emit Mint(caller, onBehalfOf, amountToMint, balanceIncrease, index);
86
87        return (scaledBalance == 0);
88      }
```

"Ray math" is to prevent rounding errors

```
66      function _mintScaled(
67        address caller,
68        address onBehalfOf,
69        uint256 amount,
70        uint256 index
71      ) internal returns (bool) {
72        uint256 amountScaled = amount.rayDiv(index);
73        require(amountScaled != 0, Errors.INVALID_MINT_AMOUNT);
74
75        uint256 scaledBalance = super.balanceOf(onBehalfOf);
76        uint256 balanceIncrease = scaledBalance.rayMul(index) -
77          scaledBalance.rayMul(_userState[onBehalfOf].additionalData);
78
79        _userState[onBehalfOf].additionalData = index.toUint128();
80
81        _mint(onBehalfOf, amountScaled.toUint128());
82
83        uint256 amountToMint = amount + balanceIncrease;
84        emit Transfer(address(0), onBehalfOf, amountToMint);
85        emit Mint(caller, onBehalfOf, amountToMint, balanceIncrease, index);
86
87        return (scaledBalance == 0);
88      }
```

```
66      function _mintScaled(
67        address caller,
68        address onBehalfOf,
69        uint256 amount,
70        uint256 index
71      ) internal returns (bool) {
72        uint256 amountScaled = amount.rayDiv(index);
73        require(amountScaled != 0, Errors.INVALID_MINT_AMOUNT);
74
75        uint256 scaledBalance = super.balanceOf(onBehalfOf);
76        uint256 balanceIncrease = scaledBalance.rayMul(index) -
77          scaledBalance.rayMul(_userState[onBehalfOf].additionalData);
78
79        _userState[onBehalfOf].additionalData = index.toUint128();
80
81        _mint(onBehalfOf, amountScaled.toUint128());
82
83        uint256 amountToMint = amount + balanceIncrease;
84        emit Transfer(address(0), onBehalfOf, amountToMint);
85        emit Mint(caller, onBehalfOf, amountToMint, balanceIncrease, index);
86
87        return (scaledBalance == 0);
88      }
```

```
127       /// @inheritdoc IERC20
128       function balanceOf(
129         address user
130       ) public view virtual override(IncentivizedERC20, IERC20) returns (uint256) {
131         return super.balanceOf(user).rayMul(POOL.getReserveNormalizedIncome(_underlyingAsset));
132       }
```

Compound's cTokens

- When you supply liquidity in Compound you get cTokens
  - Supply USDC, get cUSDC
- If you hold 5% of the cUSDC supply you are entitled to 5% of the USDC pool
- **Price** of cTokens goes up as interest is paid into the pool
  - Price of cToken does not match price of underlying