Introduction to DeFi

12 / 13

Storage Layout

Rules

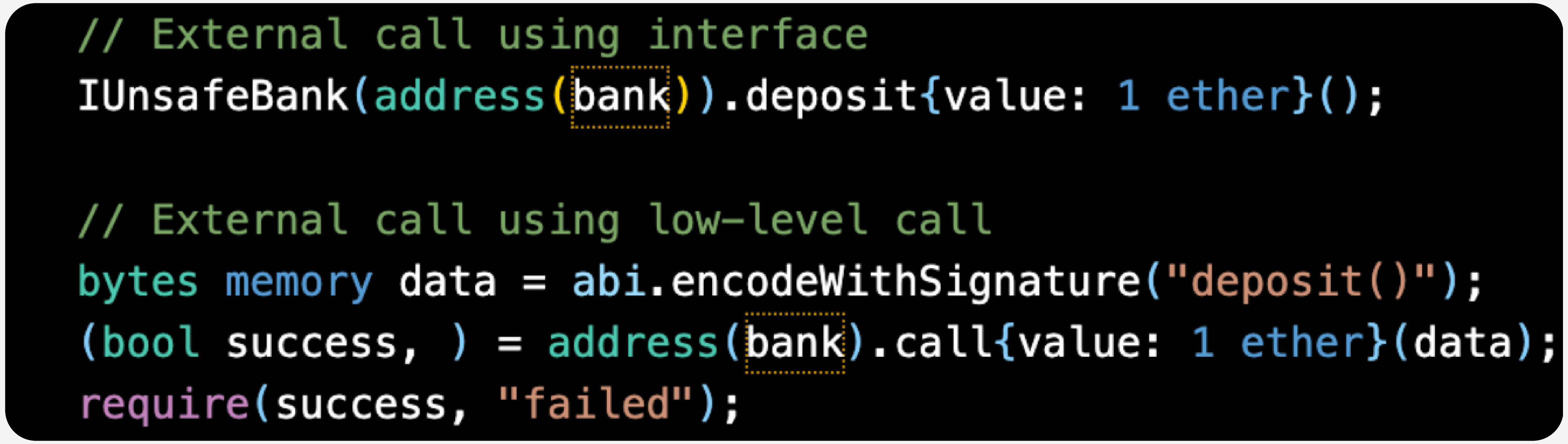
- The first item in a storage slot is stored with lower-order alignment.
- Value types use only the necessary bytes for storage.
- If a value type exceeds the remaining space in a storage slot, it will be stored in the next slot.
- Structs and array data initiate a new slot and follow these packing rules.
- Following struct or array data, subsequent items always initiate a new storage slot.
- Immutable and constant variables are not stored in storage but contract bytecode.

Examples can be found in 9 / 27 Slides p.17 - 19

Low-level Call

Question: I have the ùnsafeBank`contract address, how can I deposit my ethers through the deposit function?

Solution 2: Build the external calldata and transmit it to the target contract via a low-level call.



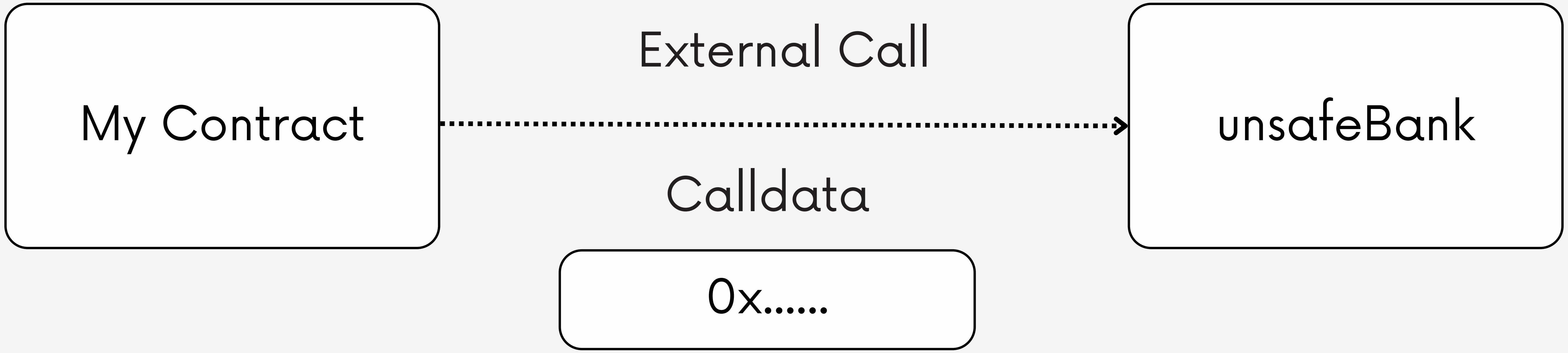
```
// External call using interface
IUnsafeBank(address(bank)).deposit{value: 1 ether}();

// External call using low-level call
bytes memory data = abi.encodeWithSignature("deposit()");
(bool success, ) = address(bank).call{value: 1 ether}(data);
require(success, "failed");
```

Low-level Call

Question: I have the ùnsafeBank`contract address, how can I deposit my ethers through the deposit function?

Solution 2: Build the external calldata and transmit it to the target contract via a low-level call.



My Contract

External Call

Calldata

0x......

unsafeBank

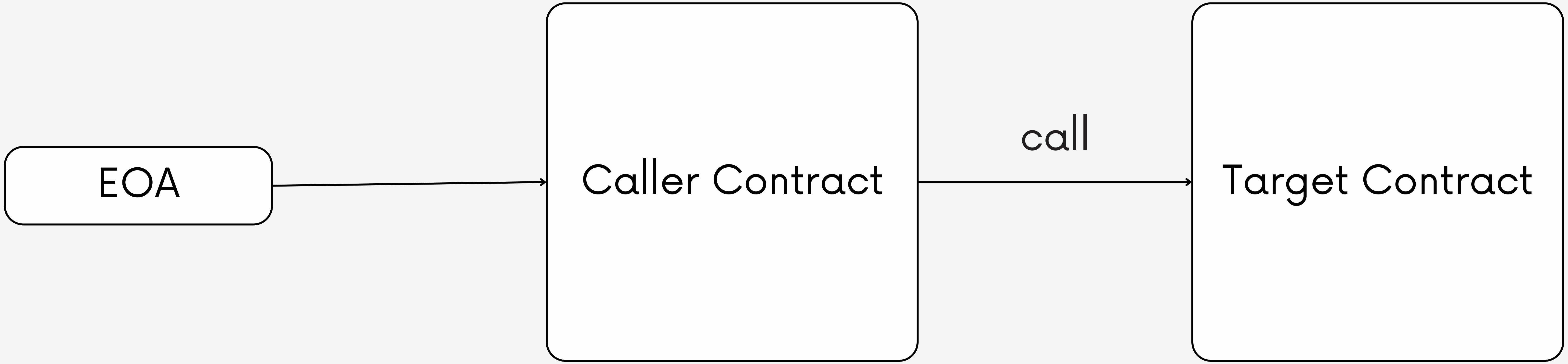
Function Dispatching

How to construct a calldata and make external call through low-level call?

Function	function balances(address addr) external returns(uint256)
Function Signature	balances(address)
Function Selector	bytes4(keccak256(“ balances(address)”))
Calldata	function selector + enc(function argument)*

*enc() follows the ABI-Encoding rules

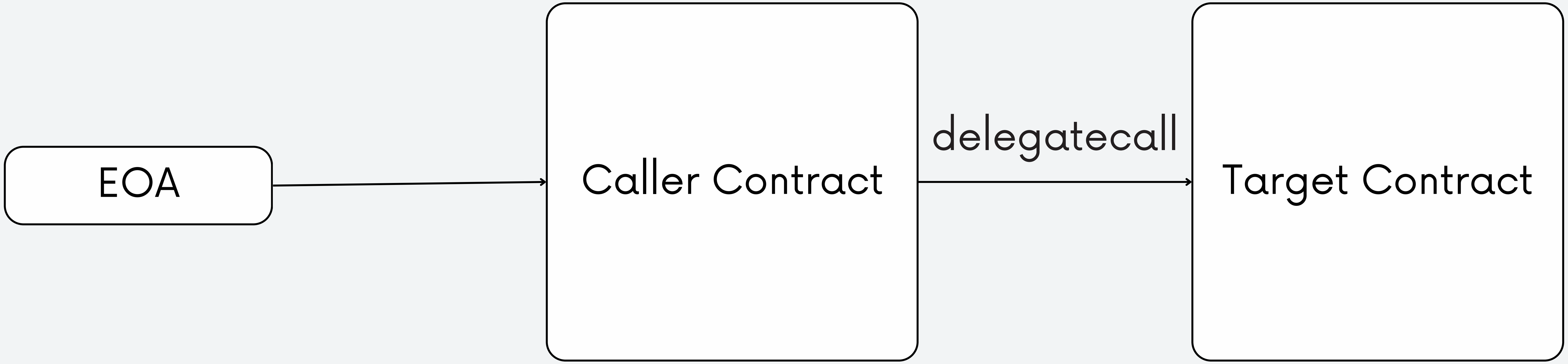
Call vs Delegatecall



msg.sender = EOA
msg.value => from EOA

msg.sender = Caller
msg.value => from Caller

Call vs Delegatecall



EOA

Caller Contract

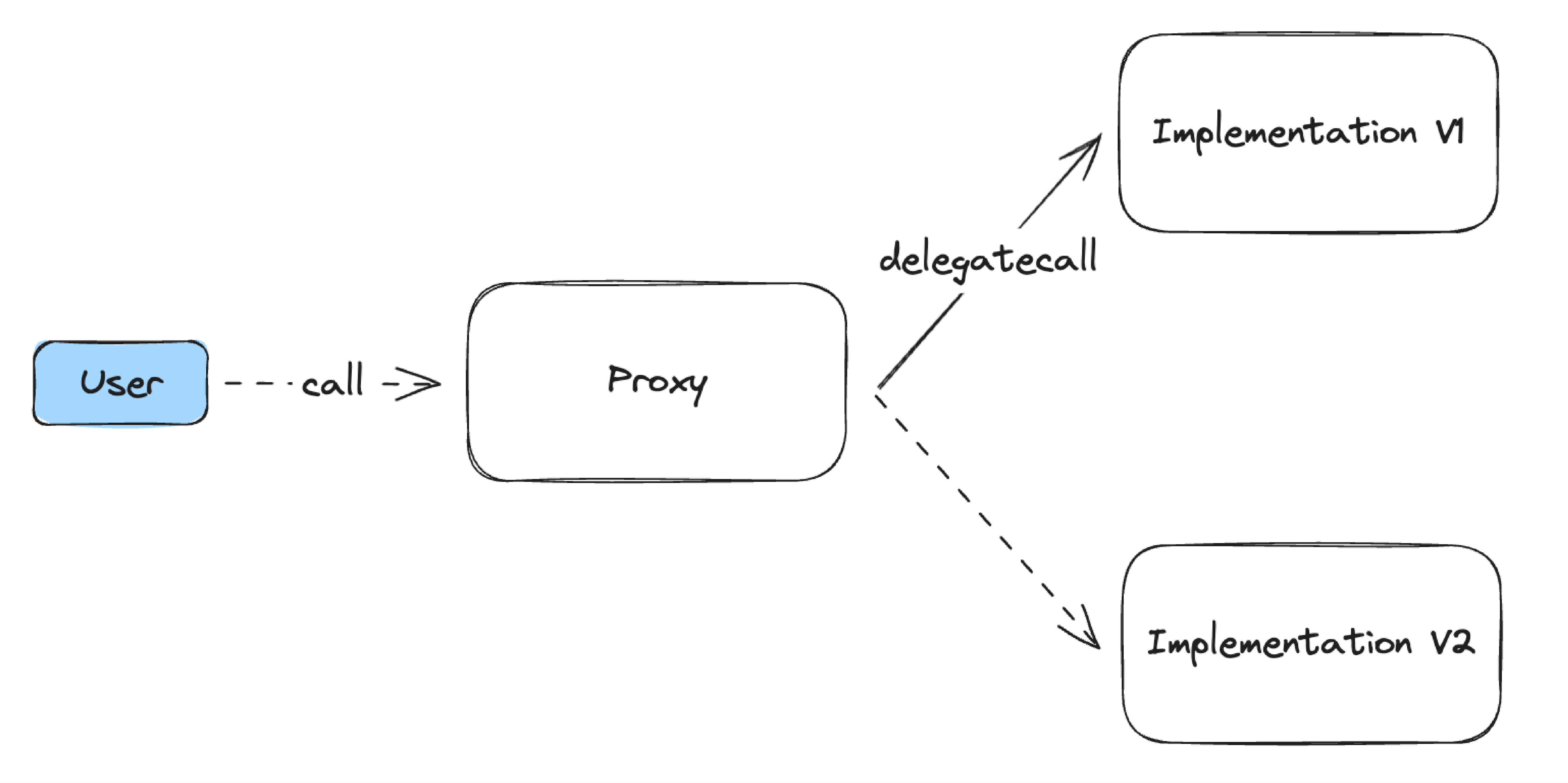
delegatecall

Target Contract

msg.sender = EOA
msg.value => from EOA

msg.sender = EOA
msg.value => from EOA

Upgradeable Proxy Overview

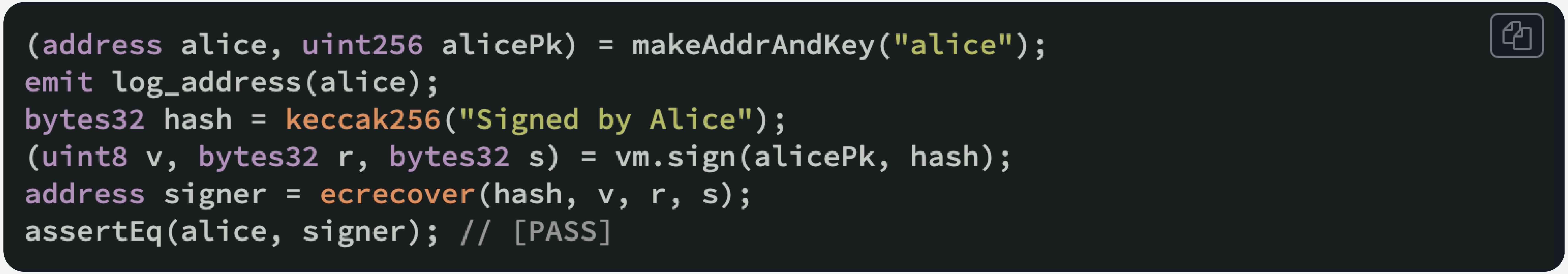


Note: Proxy Contract is not Upgradeable Proxy Contract

Signature Replay

What can be a digest / message? Any 32 bytes data

- How to sign? Use metamask / Foundry cheatcode
- How to recover? Use ecrecover precompile address. - **NOTE**: return zero address(0) when the recover process fail.



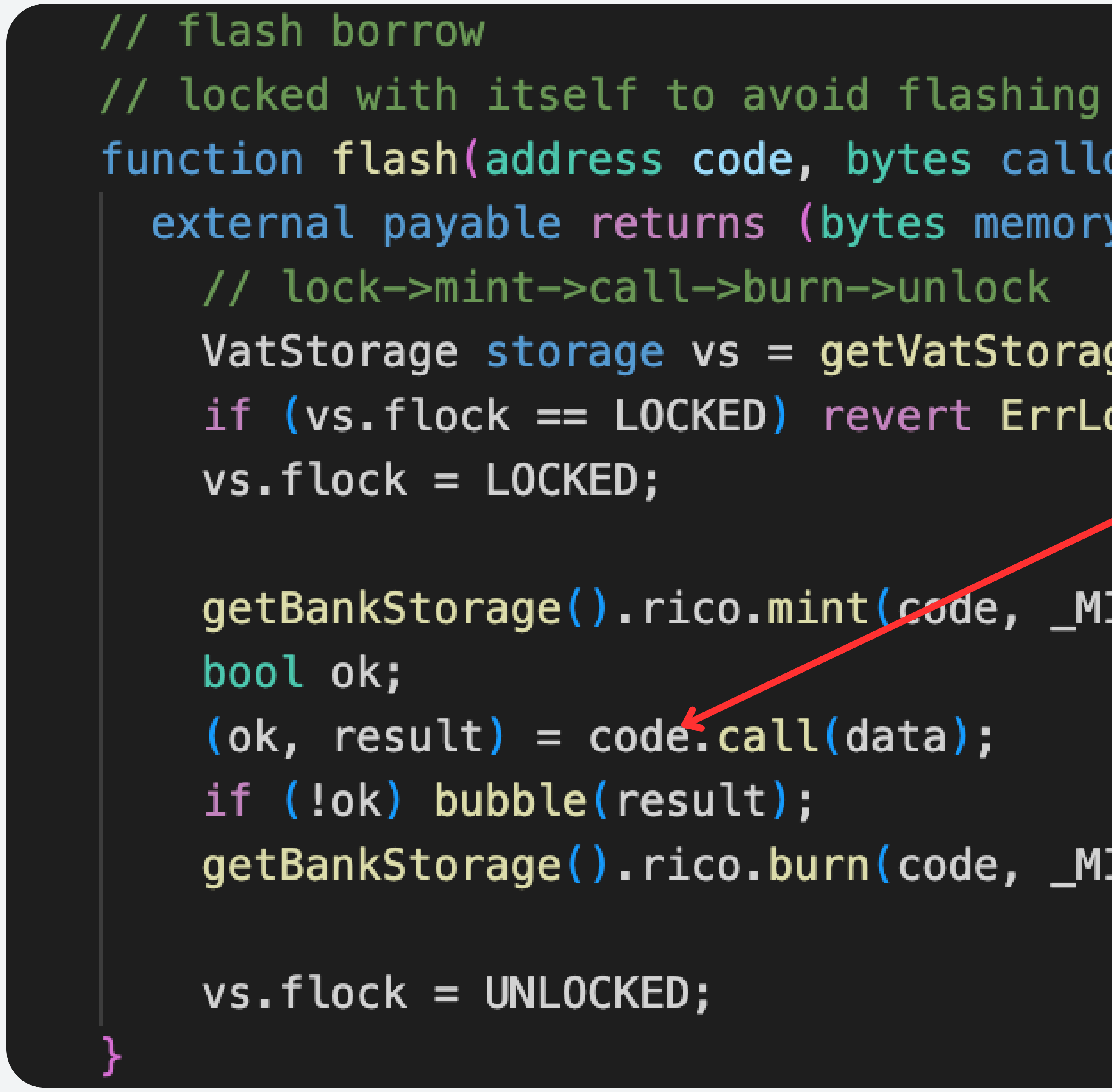
```
(address alice, uint256 alicePk) = makeAddrAndKey("alice");
emit log_address(alice);
bytes32 hash = keccak256("Signed by Alice");
(uint8 v, bytes32 r, bytes32 s) = vm.sign(alicePk, hash);
address signer = ecrecover(hash, v, r, s);
assertEq(alice, signer); // [PASS]
```

Signature Replay

Some signature can be reused for multiple times.

- Scenario 1: A proposal will deposit 5 ether to lending protocol, and if some signers want to deposit again, can they re-submit the same signature?
- Scenario 2: A proposal is created on another EVM-compatible chain, it has been executed. The same wallet is created on the Ethereum mainnet, can the signature be re-used again?

Arbitrary Call



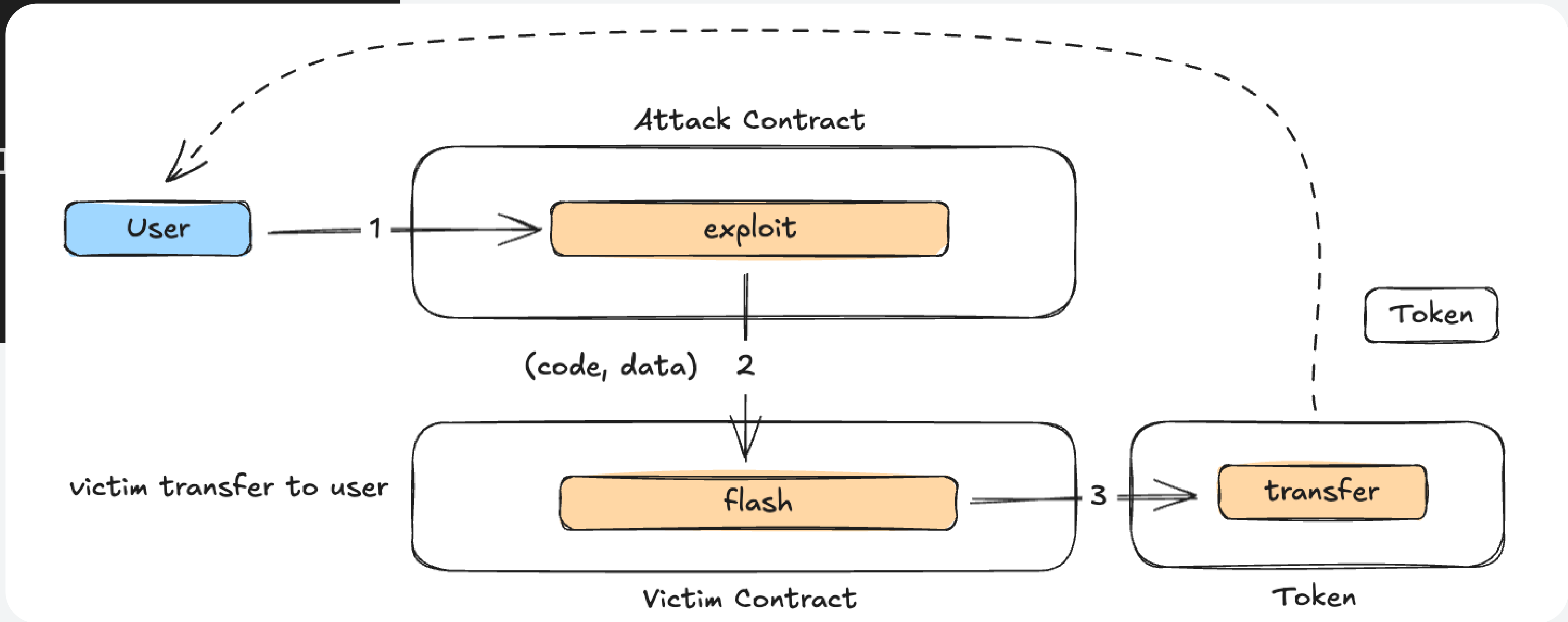
```
// flash borrow
// locked with itself to avoid flashing more than MINT
function flash(address code, bytes calldata data)
  external payable returns (bytes memory result) {
    // lock->mint->call->burn->unlock
    VatStorage storage vs = getVatStorage();
    if (vs.flock == LOCKED) revert ErrLock();
    vs.flock = LOCKED;

    getBankStorage().rico.mint(code, _MINT);
    bool ok;
    (ok, result) = code.call(data);
    if (!ok) bubble(result);
    getBankStorage().rico.burn(code, _MI

    vs.flock = UNLOCKED;
}
```

Not follow EIP-3156

The attacker provides a target address and call data to the victim contract to execute malicious actions.



Reentrancy Issue

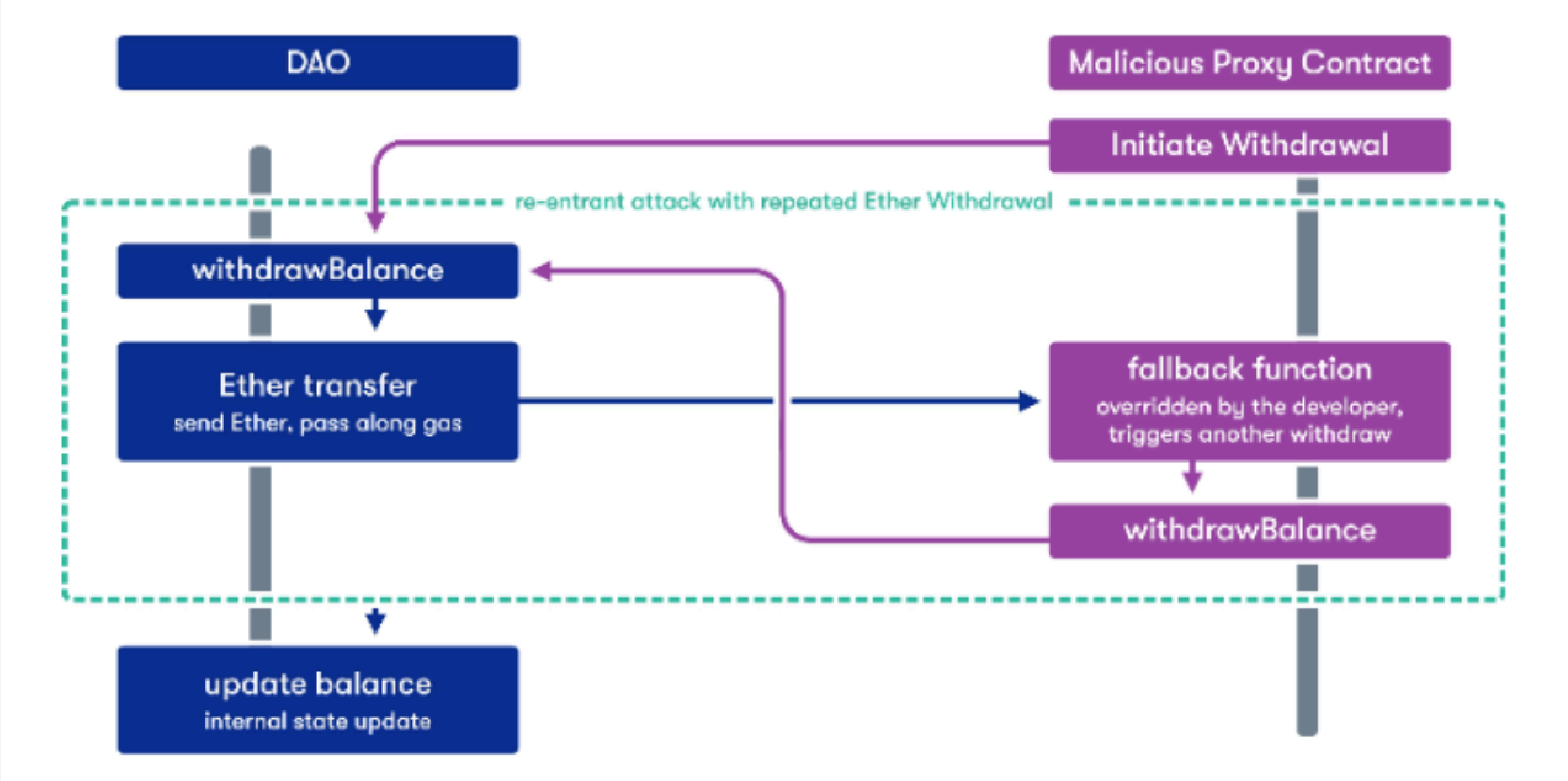
Different Types of Reentrancy

- Single function reentrancy
- Cross function reentrancy
- Cross contract reentrancy
- Cross chain reentrancy
- Read only reentrancy

The problem lies in the function that changes the state - not pure, view function

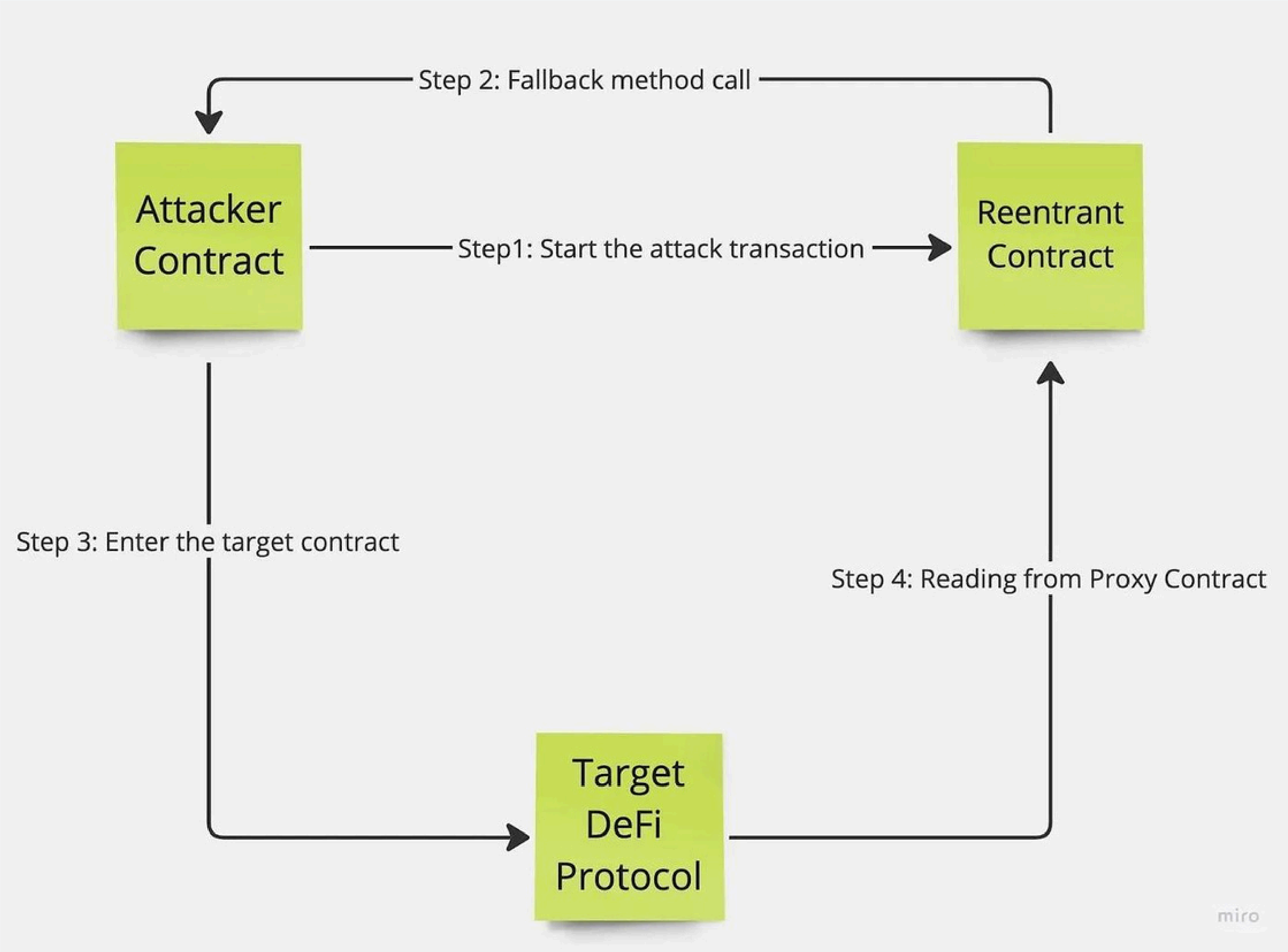
Note: the problem is in the `view`function

Reentrancy Issue



Example

Reentrancy Issue


Step 2: Fallback method call
Attacker Contract
Step1: Start the attack transaction
Reentrant Contract
Step 3: Enter the target contract
Step 4: Reading from Proxy Contract
Target DeFi Protocol
miro

The problem is in the `view` function

AMM

AMM (Automated Market Makers): Use algorithms to manage liquidity in an automated way, operating similarly to traditional market makers.

Uniswap

x * y = k

Balancer

$$\prod_{i=1}^{n} x_i^{w_i} = k$$

Curve

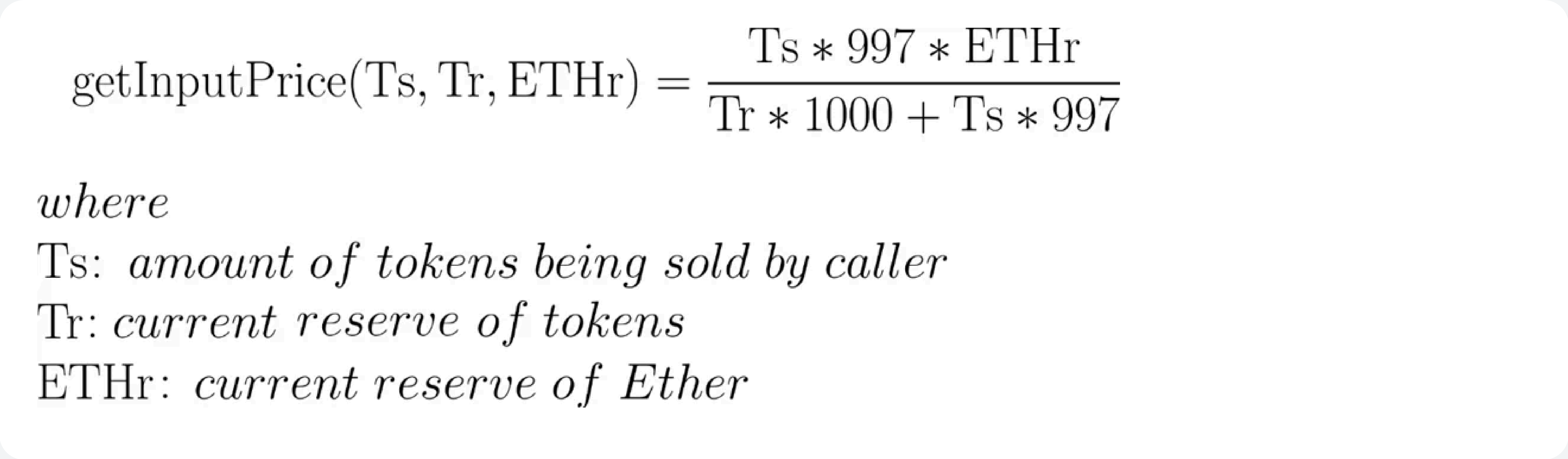
$$An^n \sum x_i + D = ADn^n + \frac{D^{n+1}}{n^n \prod x_i}.$$

Useful Article

Uniswap V1

Currently, no one use the version 1.

- The pool only allows ETH <> ERC-20 pair.
- There is **reentrancy** issue in the contract when interacting with ERC-777.



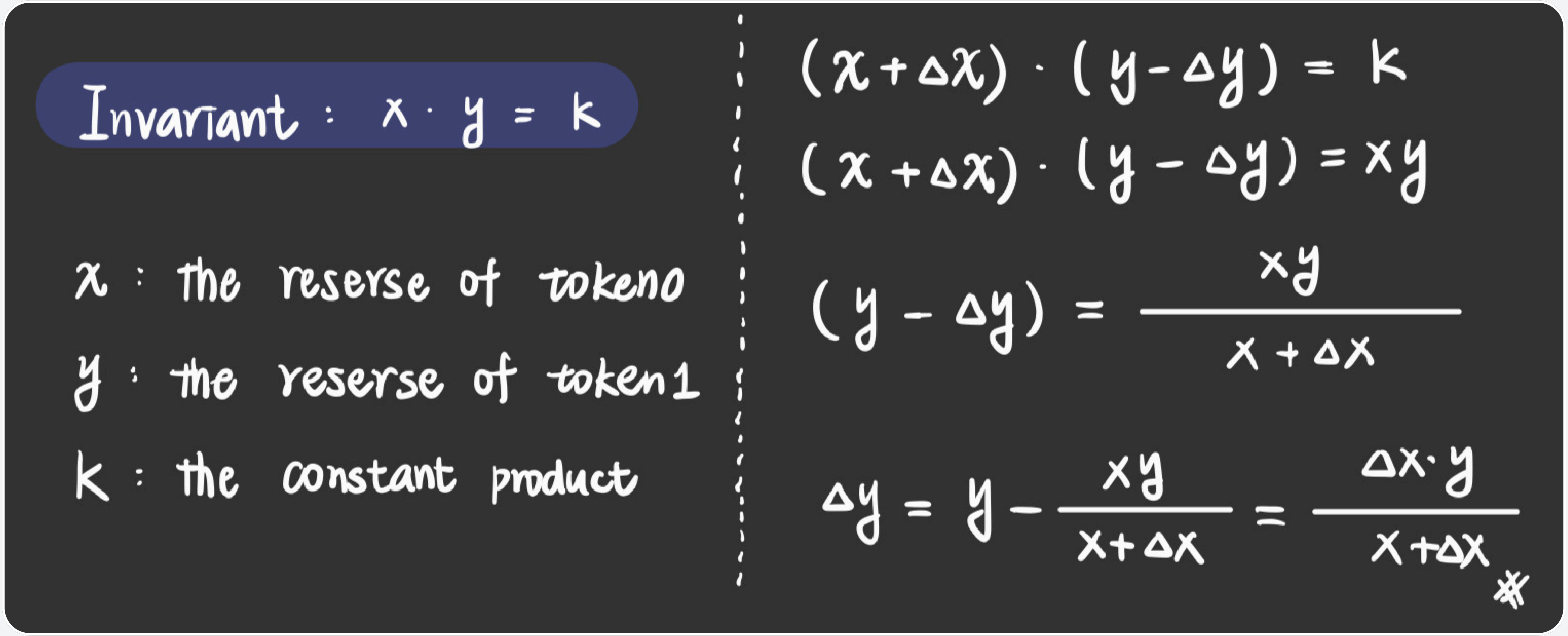
$$\text{getInputPrice}(\text{Ts}, \text{Tr}, \text{ETHr}) = \frac{\text{Ts} * 997 * \text{ETHr}}{\text{Tr} * 1000 + \text{Ts} * 997}$$

where
Ts: *amount of tokens being sold by caller*
Tr: *current reserve of tokens*
ETHr: *current reserve of Ether*

Transfer the ETH before updating the reserves

Swap Without Fee

Case 1: We want to swap x′ token0 for token1, how much will I get?



Invariant : $x \cdot y = k$

x : the reserse of token0

y : the reserse of token1

k : the constant product

$$(x+\Delta x)\cdot(y-\Delta y) = k$$

$$(x+\Delta x)\cdot(y-\Delta y) = xy$$

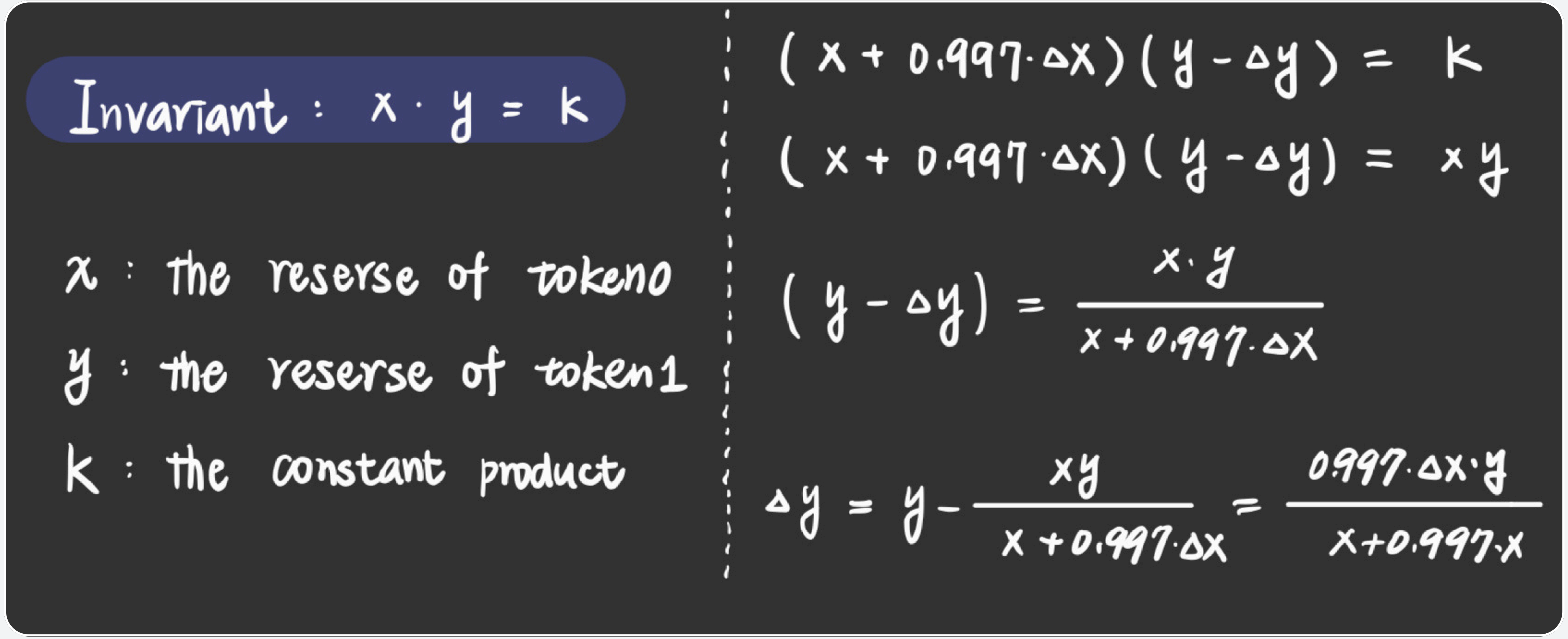
$$(y-\Delta y) = \frac{xy}{x+\Delta x}$$

$$\Delta y = y - \frac{xy}{x+\Delta x} = \frac{\Delta x \cdot y}{x+\Delta x} \; \#$$

Useful Article

Swap With Fee

Case 1: We want to swap x′ token0 for token1, how much will I get?



Invariant : $x \cdot y = k$

x : the reserse of token0

y : the reserse of token1

k : the constant product

$$(x + 0.997 \cdot \Delta x)(y - \Delta y) = k$$

$$(x + 0.997 \cdot \Delta x)(y - \Delta y) = xy$$

$$(y - \Delta y) = \frac{x \cdot y}{x + 0.997 \cdot \Delta x}$$

$$\Delta y = y - \frac{xy}{x + 0.997 \cdot \Delta x} = \frac{0.997 \cdot \Delta x \cdot y}{x + 0.997 \cdot x}$$

Useful Article

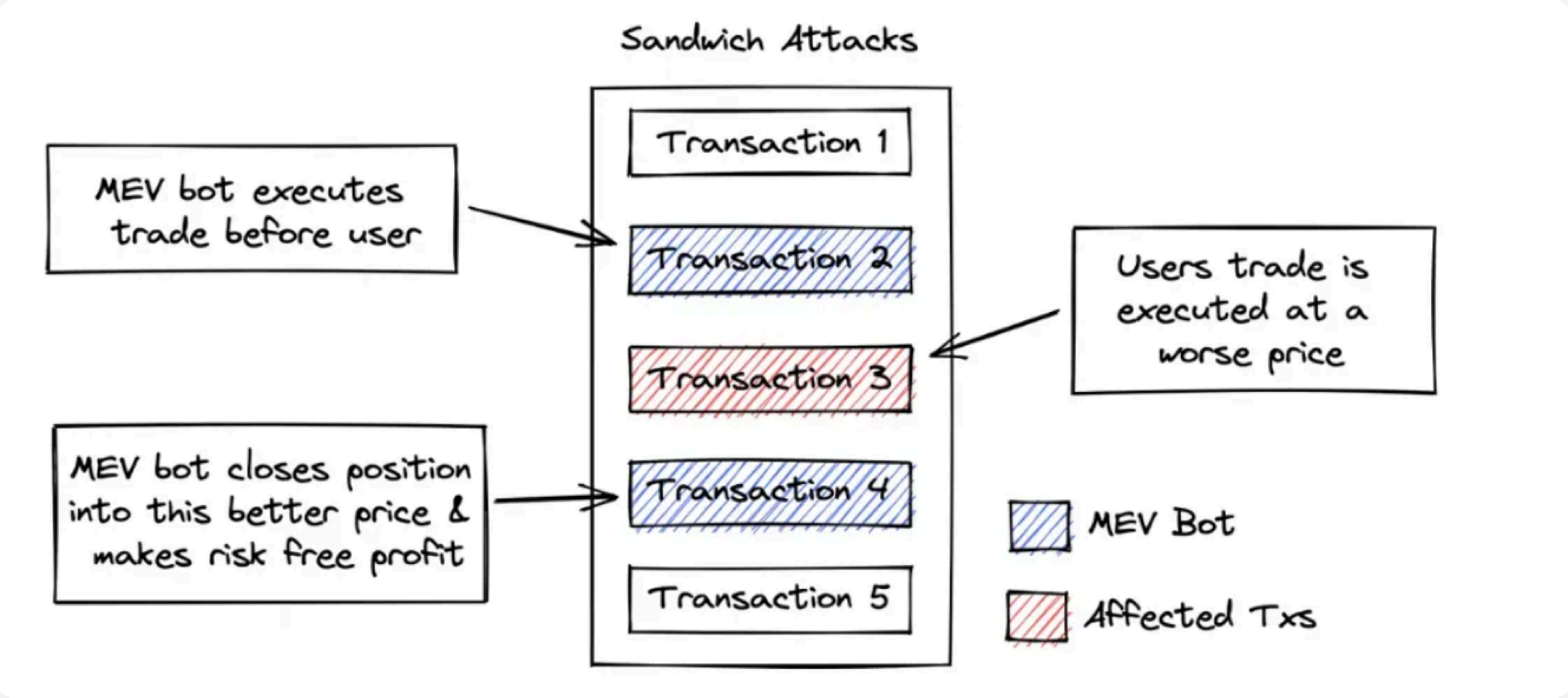
Slippage Issue

Can I always get the amount that I want

- Slippage: The difference between the expected price and the actual price.
- Imagine calculating the output amount off-chain and providing 3400 USDC to swap for 1 ETH.
- What if your transaction is not processed in time, and another user swaps, changing the pool's state?
- The output amount fluctuates as the pool updates, leading to slippage.
- What might cause slippage? One potential cause is MEV

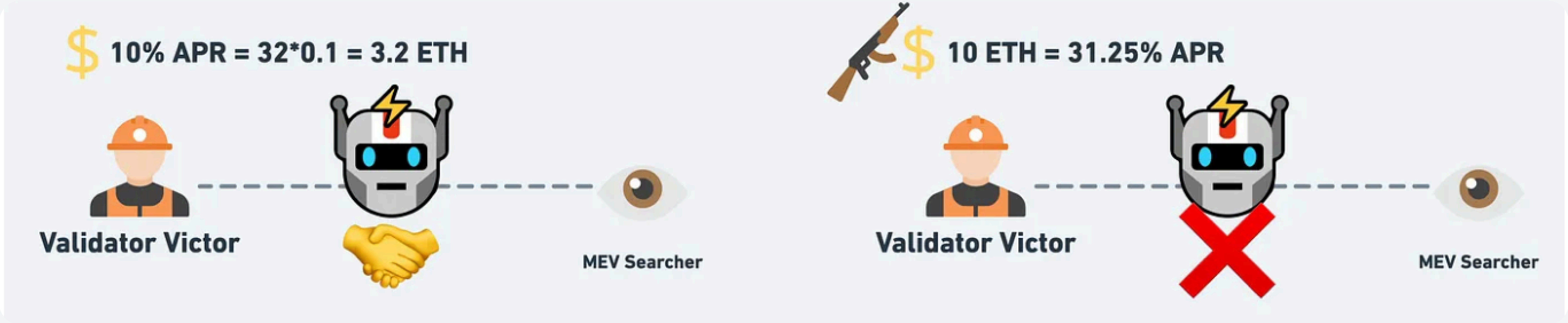
Sandwich Attack

a form of front-running where an attacker places one transaction before and one after a target transaction to exploit price changes for profit.


Sandwich Attacks
Transaction 1
Transaction 2
Transaction 3
Transaction 4
Transaction 5
MEV bot executes trade before user
Users trade is executed at a worse price
MEV bot closes position into this better price & makes risk free profit
MEV Bot
Affected Txs

More on MEV

From PoW to PoS, from miner-extractable-value to maximum-extractable-value



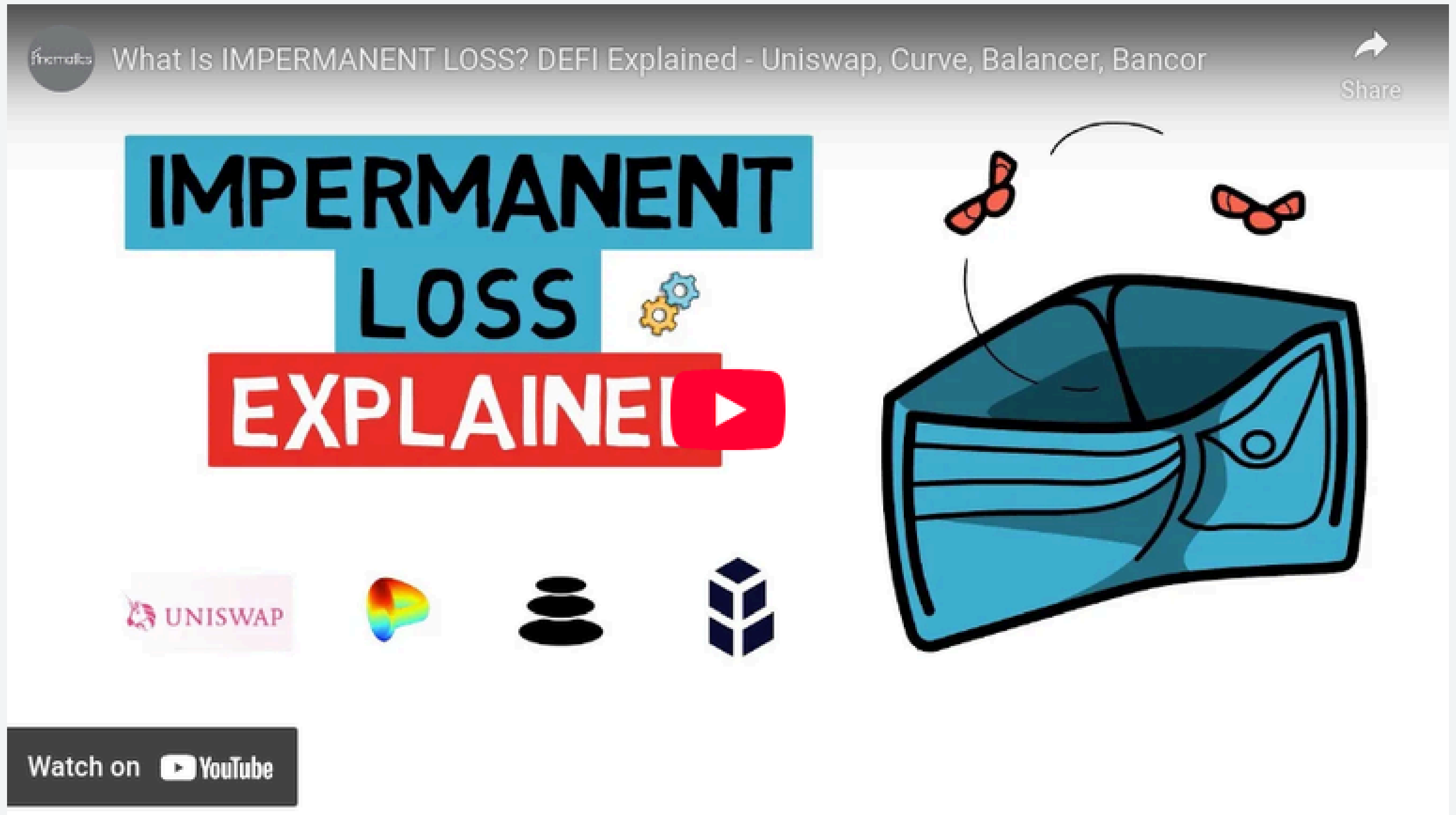
When transitioning to PoS, the validator might receive 10% APR in return.

What if the validator copies your transaction? Maybe it can earn more.

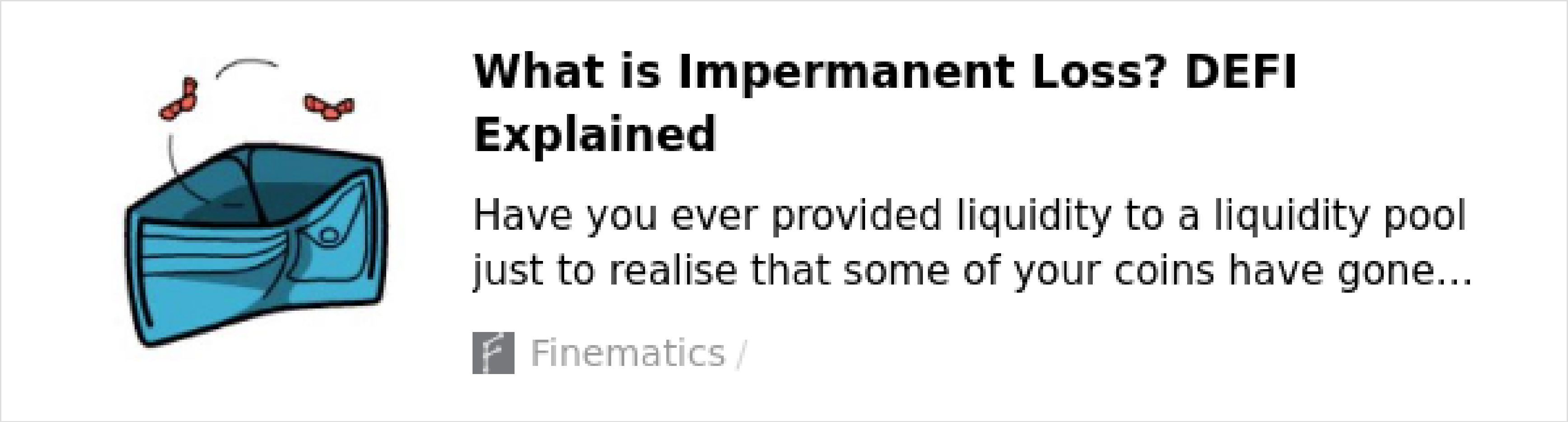
Reference

Impermanent Loss

I would recommend you check this video



What Is IMPERMANENT LOSS? DEFI Explained - Uniswap, Curve, Balancer, Bancor



What is Impermanent Loss? DEFI Explained

Have you ever provided liquidity to a liquidity pool just to realise that some of your coins have gone...

Finematics /

Lending Category

There are different types of lending based on the collateral provided

- Uncollateralized Loan: A loan issued without requiring the borrower to provide any collateral, such as a flash loan.
- Undercollateralized Loan: A loan where the collateral provided is worth less than the loan amount.
- Overcollateralized Loan: A loan where the borrower provides collateral that exceeds the value of the loan amount.

Leverage Long

In a lending protocol, you supply Token A as collateral and borrow Token B

- Step 1: Supply LiaoToken as collateral
- Step 2: Borrow USDC from the lending protocol
- Step 3: Use the borrowed USDC to buy more LiaoToken

Debt: The borrowed USDC.

Position: The collateralized ETH + the purchased ETH

Leverage Short

In a lending protocol, you supply Token A as collateral and borrow Token B

- Step 1: Supply USDC as collateral
- Step 2: Borrow LiaoToken from the lending protocol
- Step 3: Use the borrowed LiaoToken to buy more USDC

Debt: The borrowed LiaoToken

Position: The collateralized USDC + the USDC from selling LiaoToken.

Liquidation

The market is volatile, causing the prices of both the collateral and the borrowed assets to fluctuate.

Block	Collateral	Loan	Collateral - Loan	LTV
10,000	1000 USD	800 USD	1000 - 800	80%
10,020	600 USD (Drops 40%)	800 USD	600 - 800 = -200	133%

When the LTV exceeds 100%, the protocol incurs bad debt, leading to insolvency and an inability to repay lenders

Liquidation

The market is volatile, causing the prices of both the collateral and the borrowed assets to fluctuate.

Block	Collateral	Loan	Collateral - Loan	LTV
10,000	1000 USD	800 USD	1000 - 800	80%
10,020	1000 USD	1200 USD	600 - 800 = -200	133%

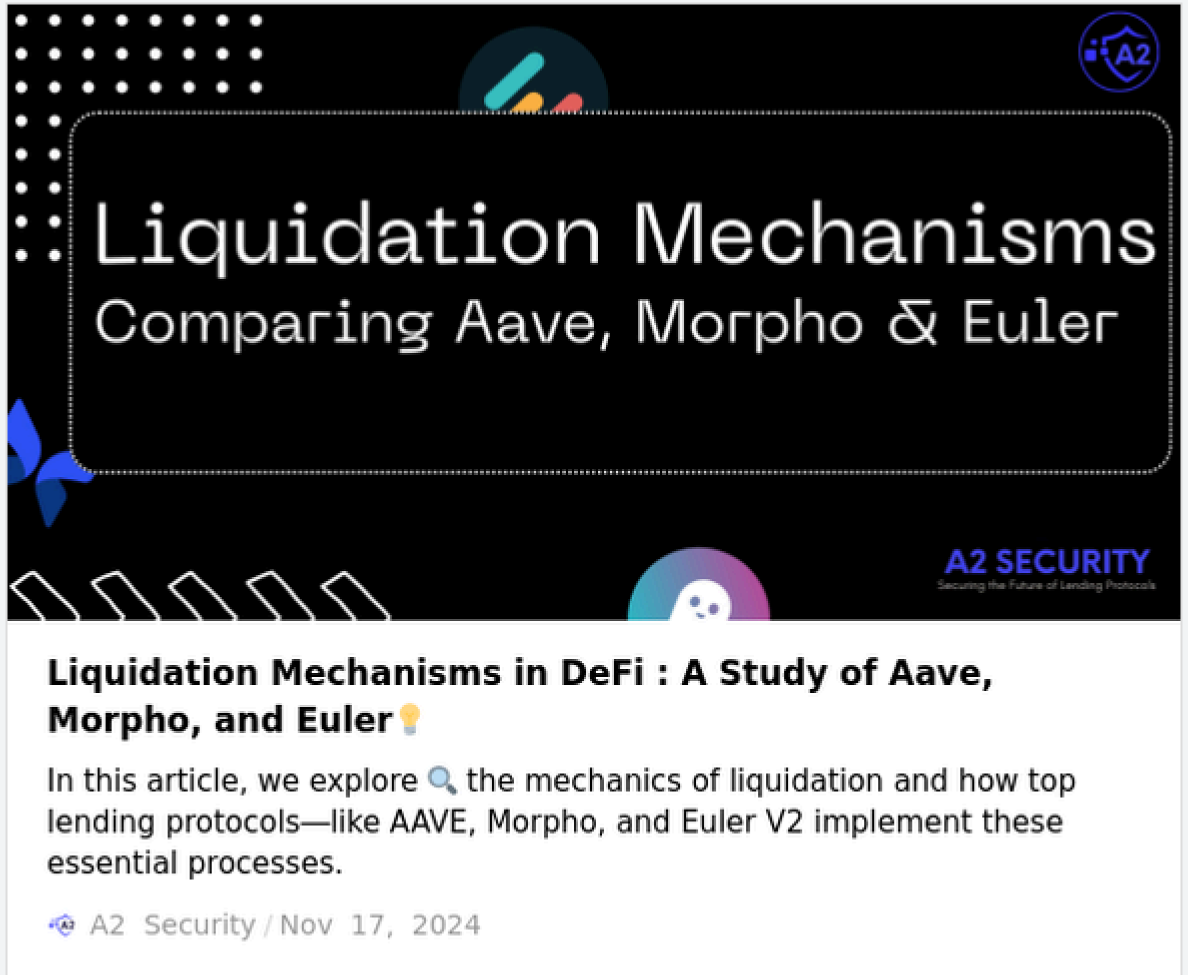
When the LTV approaches 100%, it becomes unmanageable and must be addressed before reaching this condition

Liquidation

Liquidation Factor: The threshold percentage at which a borrower's collateral may be forcibly liquidated to cover their debt.

- The liquidation factor is always set higher than the collateral factor.
    - If the collateral factor is 80% and the liquidation factor is 70%, the borrower would be liquidated immediately after borrowing.
- Setting the liquidation factor too close to 100%
    - Increases the risk of bad debt
    - The collateral's value may drop below the loan amount by the time liquidation occurs.

More on Liquidation



Liquidation Mechanisms

Comparing Aave, Morpho & Euler

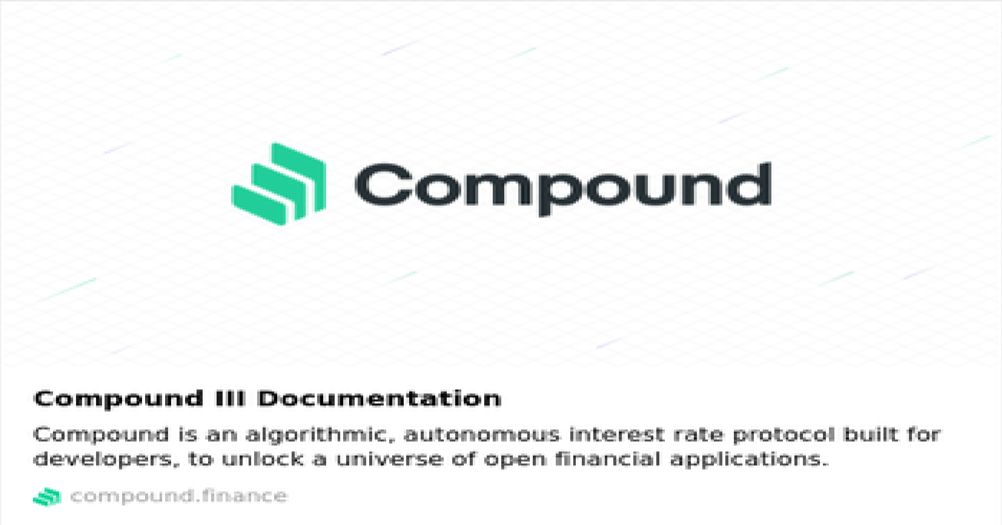
A2 SECURITY

Securing the Future of Lending Protocols

Liquidation Mechanisms in DeFi : A Study of Aave, Morpho, and Euler💡

In this article, we explore 🔍 the mechanics of liquidation and how top lending protocols—like AAVE, Morpho, and Euler V2 implement these essential processes.

A2 Security / Nov 17, 2024



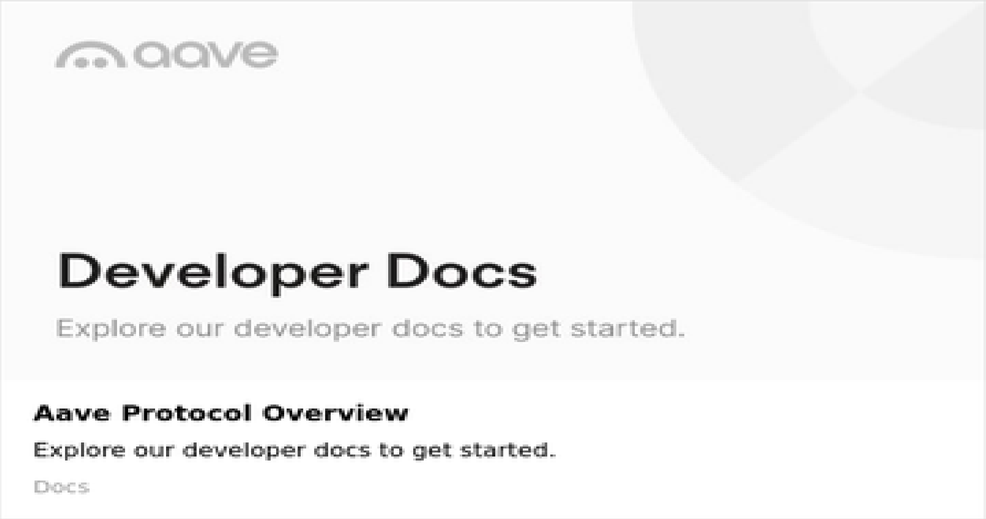
Compound

Compound III Documentation

Compound is an algorithmic, autonomous interest rate protocol built for developers, to unlock a universe of open financial applications.

compound.finance

Compound v3



aave

Developer Docs

Explore our developer docs to get started.

Aave Protocol Overview

Explore our developer docs to get started.

Docs

Aave

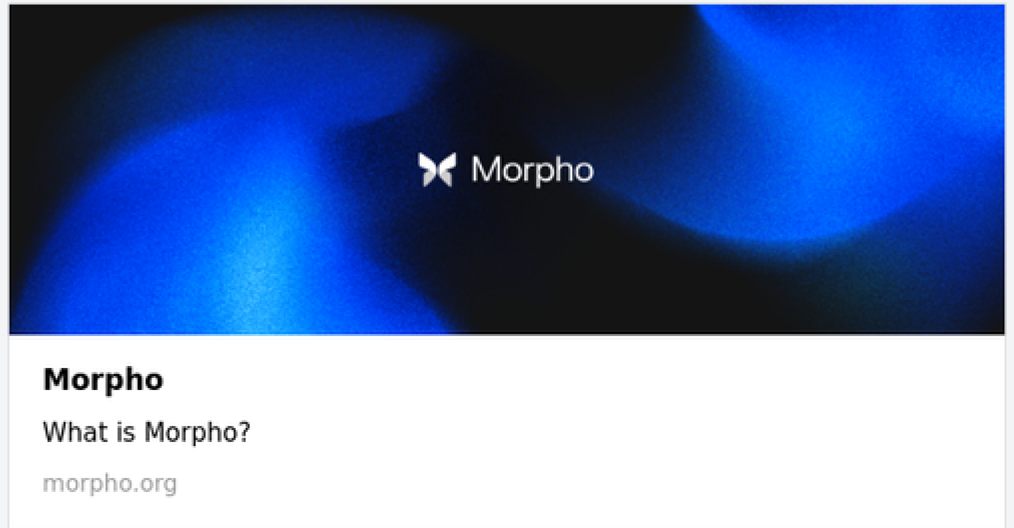
Compound

Compound v2 Documentation

Compound is an algorithmic, autonomous interest rate protocol built for developers, to unlock a universe of open financial applications.

compound.finance

Compound v2



Morpho

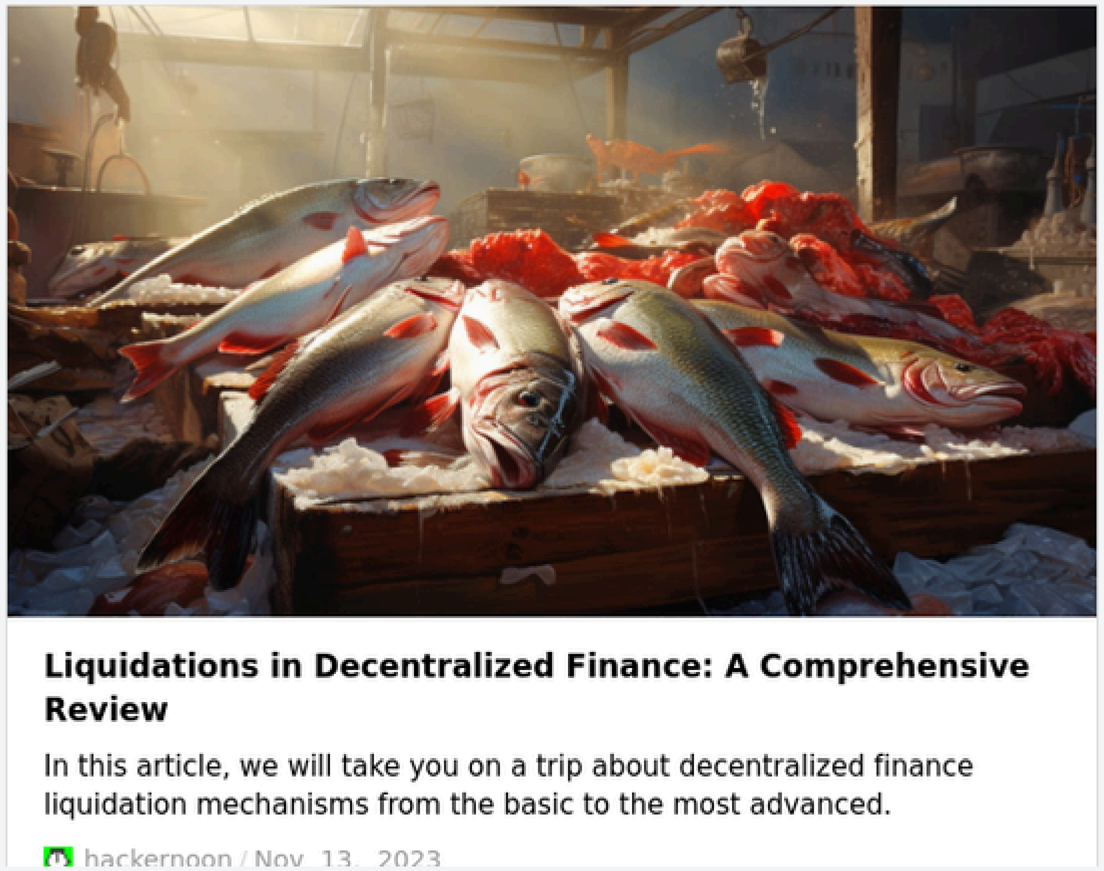
Morpho

What is Morpho?

morpho.org

Morpho

Supplementary Materials



Liquidations in Decentralized Finance: A Comprehensive Review

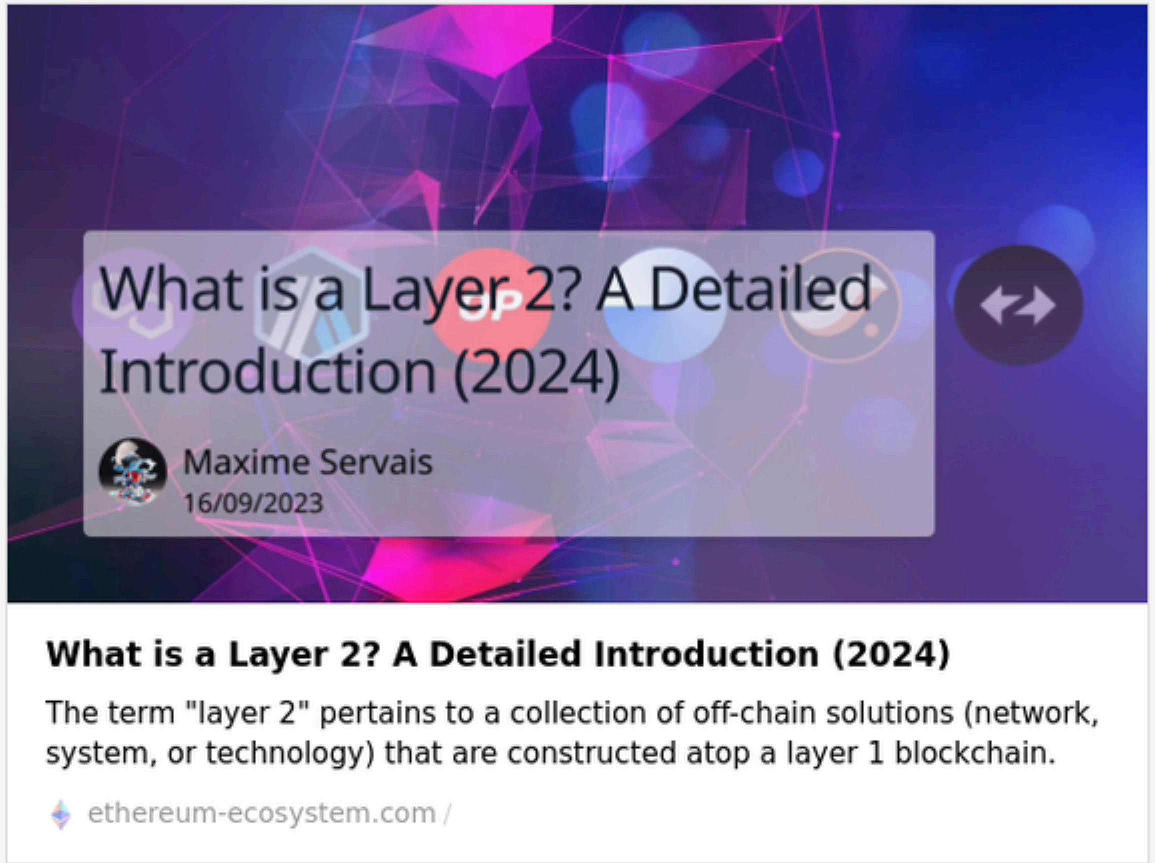
In this article, we will take you on a trip about decentralized finance liquidation mechanisms from the basic to the most advanced.

hackernoon / Nov. 13, 2023

Safeguarding Blockchain Ecosystem: Understanding and Detecting...

Cross-chain bridges are essential decentralized applications (DApps) to facilitate interoperability between different blockchain networks. Unlike regular DApps, the functionality of cross-chain...

arXiv.org



What is a Layer 2? A Detailed Introduction (2024)

The term "layer 2" pertains to a collection of off-chain solutions (network, system, or technology) that are constructed atop a layer 1 blockchain.

ethereum-ecosystem.com /