Blockchains & Cryptocurrencies

Smart Contracts / Ethereum In Detail (III)

Instructor: Matthew Green
Fall 2024

News?

News?

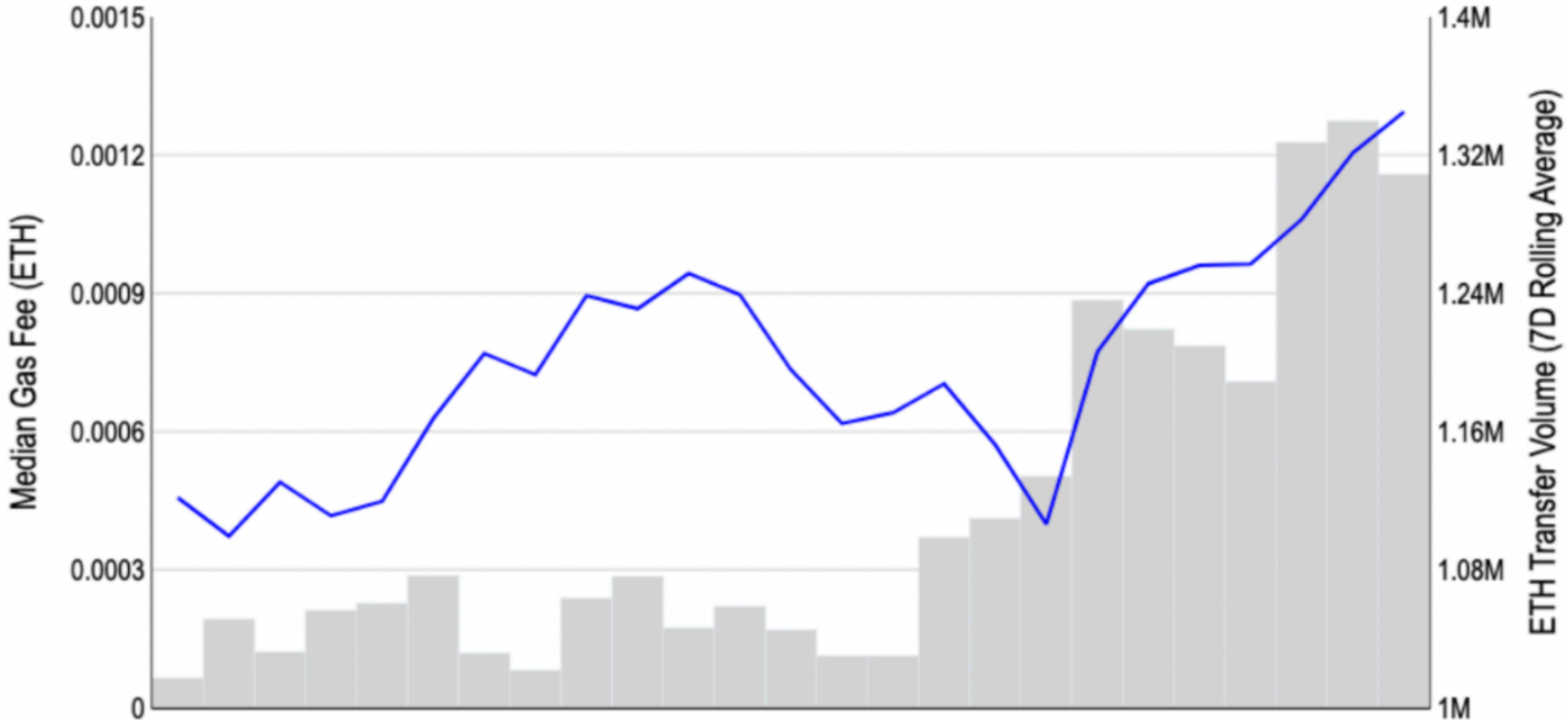
Don't Tell Anyone, but Private Blockchains Handle Over $1.5T of Securities Financing a Month

Permission-based repo ledgers are among the most successful applications o blockchain technology.

By Ian Allison Jun 24, 2024 at 5:46 a.m. EDT Updated Jun 24, 2024 at 5:49 a.m. EDT

News?

Coinbase's weekly report, published on Sept. 27, notes that the average Ethereum gas fees between Sept. 16 and Sept. 26 were 498% higher than the monthly average, with the median transaction cost rising from $0.09 at the beginning of the month to $1.69.



Ethereum Gas Tracker ⛽

Sponsored by: MetaMask

Next update in 6s

</> Gas APIs | Install Gas Extension

Ad

😁 Low	😀 Average	🙂 High
7.813 gwei	8.006 gwei	8.833 gwei
Base: 7.812 | Priority: 0.001	Base: 7.812 | Priority: 0.194	Base: 7.812 | Priority: 1.021
$0.40 | ~ 2 mins: 31 secs	$0.41 | ~ 1 min: 48 secs	$0.45 | ~ 30 secs

Additional Info

LAST BLOCK	PENDING QUEUE	AVG BLOCK SIZE	AVG. UTILIZATION
20877811	151463	171	48.29%

Last Refreshed: Wed, 02 Oct 2024 12:27:26 UTC

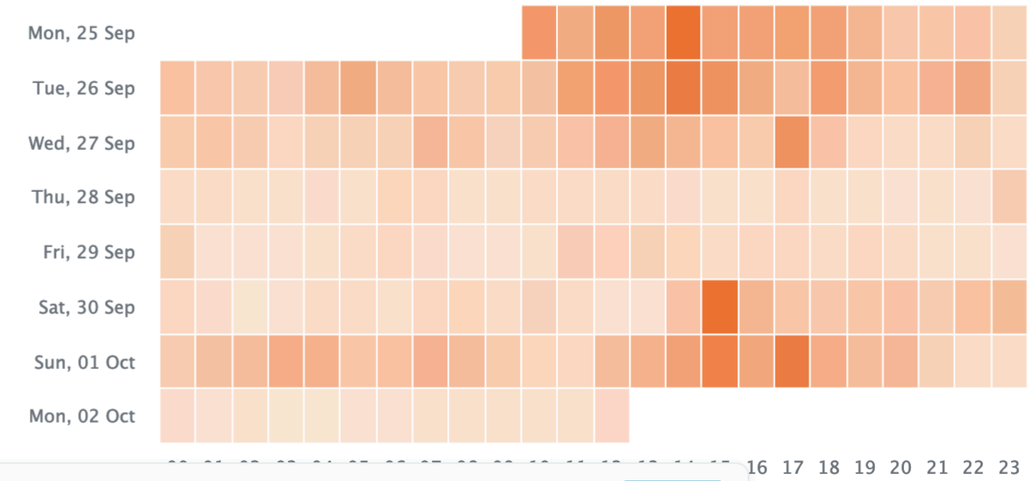
Featured Actions

Action	Low	Average	High
Swap	$6.84	$7.01	$7.71
NFT Sale	$11.56	$11.84	$13.04
Bridging	$2.20	$2.25	$2.48
Borrowing	$5.80	$5.95	$6.54

Custom Gas Limit

Gas Price Heatmap | Gas Price

Gas Price Heatmap



Top 50 Gas Guzzlers (Contracts / Accounts that consume a lot of Gas)

Dashboard | Last updated at Block 20877779

Rank	Address	Fees Last 3hrs	% Used 3hrs	Fees Last 24hrs	% Used 24hrs	Analytics
1	Uniswap: Universal Router	$26,671.83 (10.87 Eth)	10.59%	$483,154.57 (196.88 Eth)	11.66%	
2	Tether: USDT Stablecoin	$17,119.69 (6.98 Eth)	6.76%	$247,016.60 (100.66 Eth)	5.71%	
3	Uniswap V2: Router 2	$9,807.52 (4.00 Eth)	2.85%	$129,528.64 (52.78 Eth)	2.64%	
4	Maestro: Router 2	$10,625.01 (4.33 Eth)	2.36%	$129,377.93 (52.72 Eth)	1.96%	
5	0x1a93c356...74e270d01	$4,754.70 (1.94 Eth)	2.35%	$4,761.25 (1.94 Eth)	0.30%	
6	0xF3dE3C0d...172509127	$5,092.66 (2.08 Eth)	2.02%	$67,205.75 (27.39 Eth)	1.86%	
7	EigenLayer: EIGEN Token	$4,132.22 (1.68 Eth)	1.77%	$75,805.05 (30.89 Eth)	2.51%	
8	Circle: USDC Token	$4,387.47 (1.79 Eth)	1.76%	$83,721.57 (34.12 Eth)	1.96%	
9	Metamask: Swap Router	$4,426.02 (1.80 Eth)	1.68%	$71,576.21 (29.17 Eth)	1.73%	
10	Aggregation Router V5	$3,477.51 (1.42 Eth)	1.47%	$73,738.08 (30.05 Eth)	1.87%	
11	Banana Gun: Router 2	$7,129.98 (2.91 Eth)	1.39%	$148,304.80 (60.43 Eth)	1.80%	
12	EigenLayer: Delega					

From concept to practice

- Ethereum programs are in "EVM byte code"
  - This is great for running things in a VM, works across platforms
  - Not made for human comprehension

```
00000d8b  PUSH1   #2 {var_e0_25}
00000d8d  EXP      {var_c0_53}
00000d8e  SUB      {var_c0_53} {var_a0_34}
00000d8f  DUP4     {var_40_4} {var_c0_54}
00000d90  AND      {var_a0_35} {var_a0_34} {var_c0_54}
00000d91  PUSH1   #0 {var_c0_55}
00000d93  SWAP1    {var_a0_35} {var_a0_36} {var_c0_56}
00000d94  DUP2     {var_e0_26}
```

Source: https://blog.ret2.io/2018/05/16/practical-eth-decompilation/

From concept to practice

- **How does a developer see Ethereum?**
  - So far we have talked about:
    - Init function (at deploy)
    - A single stateUpdate function (triggered by message Tx)
    - Databases and VMs

From concept to practice

- Developers typically write programs in a high-level language
  - Technically any language can compile to EVM bytecode
  - And there are a few: Agoric (Javascript), Vyper
    - Some other chains (e.g., Solana) use rust
  - However, most Ethereum smart contracts are written in **Solidity**

Source: https://blog.ret2.io/2018/05/16/practical-eth-decompilation/

Solidity

- **How does a Solidity developer see Ethereum?**
  - Solidity is object-oriented
    - "contract" programs are like classes, with methods and variables
    - Each contract will have a constructor method that initializes any state variables
    - There are "view" (read-only) methods, and methods that (may) change state
    - Methods can have modifiers attached, that execute specific checks

Contract examples

- Simple "custom token" contract (ERC20)

```
1 // ----------------------------------------------------------------------
2 // ERC Token Standard #20 Interface
3 // https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
4 // ----------------------------------------------------------------------
5 contract ERC20Interface {
6     function totalSupply() public view returns (uint);
7     function balanceOf(address tokenOwner) public view returns (uint balance);
8     function allowance(address tokenOwner, address spender) public view returns (uint remaining);
9     function transfer(address to, uint tokens) public returns (bool success);
10    function approve(address spender, uint tokens) public returns (bool success);
11    function transferFrom(address from, address to, uint tokens) public returns (bool success);
12
13    event Transfer(address indexed from, address indexed to, uint tokens);
14    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
15 }
```

```
contract TokenContractFragment {

    // Balances for each account
    mapping(address => uint256) balances;

    // Owner of account approves the transfer of an amount to another account
    mapping(address => mapping (address => uint256)) allowed;

    // Get the token balance for account `tokenOwner`
    function balanceOf(address tokenOwner) public constant returns (uint balance) {
        return balances[tokenOwner];
    }

    // Transfer the balance from owner's account to another account
    function transfer(address to, uint tokens) public returns (bool success) {
        balances[msg.sender] = balances[msg.sender].sub(tokens);
        balances[to] = balances[to].add(tokens);
        Transfer(msg.sender, to, tokens);
        return true;
    }

    // Send `tokens` amount of tokens from address `from` to address `to`
    // The transferFrom method is used for a withdraw workflow, allowing contracts to send
    // tokens on your behalf, for example to "deposit" to a contract address and/or to charge
    // fees in sub-currencies; the command should fail unless the _from account has
    // deliberately authorized the sender of the message via some mechanism; we propose
    // these standardized APIs for approval:
    function transferFrom(address from, address to, uint tokens) public returns (bool success) {
        balances[from] = balances[from].sub(tokens);
        allowed[from][msg.sender] = allowed[from][msg.sender].sub(tokens);
        balances[to] = balances[to].add(tokens);
        Transfer(from, to, tokens);
        return true;
    }

    // Allow `spender` to withdraw from your account, multiple times, up to the `tokens` amount.
    // If this function is called again it overwrites the current allowance with _value.
    function approve(address spender, uint tokens) public returns (bool success) {
        allowed[msg.sender][spender] = tokens;
        Approval(msg.sender, spender, tokens);
        return true;
    }
}
```

What can you trust?

- Contract/state variables will be isolated
- The **block** and **msg** data structures (and other globals)
  - Contain details of the TX and block you're running
- Your own contract address -> **address(this)**
- The hash of the execution code for a contract
  - Given a contract address, use the **EXTCODEHASH** opcode
  - Note: constructor code isn't included!

What can you trust?

- `block.basefee` (`uint`): current block's base fee (EIP-3198 and EIP-1559)
- `block.blobbasefee` (`uint`): current block's blob base fee (EIP-7516 and EIP-4844)
- `block.chainid` (`uint`): current chain id
- `block.coinbase` (`address payable`): current block miner's address
- `block.difficulty` (`uint`): current block difficulty (`EVM < Paris`). For other EVM versions it behaves as a deprecated alias for `block.prevrandao` (EIP-4399)
- `block.gaslimit` (`uint`): current block gaslimit
- `block.number` (`uint`): current block number
- `block.prevrandao` (`uint`): random number provided by the beacon chain (`EVM >= Paris`)
- `block.timestamp` (`uint`): current block timestamp as seconds since unix epoch
- `gasleft() returns (uint256)`: remaining gas

How to upgrade a contract?

How to upgrade a contract?

- Contracts are default immutable
- If there is no useful ongoing state, don't: just replace it
- If there is ongoing state (e.g., account balances) then:
  - Don't allow upgrades — and pray you got the code right
  - Call upgradeable/replaceable library code
  - Create a complex mechanism to transfer state from an old contract instance to a new contract instance

Concurrency and re-entrancy

- Ethereum transactions run sequentially and atomically
  - In principle this is good: there **appear** to be no concurrency issues (no threads) and your methods always run to completion or don't complete at all!

Example (ok)

```
function transfer(uint amount, address recipient) … {
    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }

    // Transfer the money
    balances[msg.sender] -= amount;
    balances[recipient] += amount;
}
```

Re-entrancy

```
function transferAndNotify(uint amount, address notifyContract) … {
    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }

    // Call the specified contract to notify it that a deposit is coming
    notifyContract.notify(amount, "You are getting a deposit!");

    // Transfer the money
    balances[msg.sender] -= amount;
    balances[notifyContract] += amount;
}
```

Re-entrancy

- There is still one scary "gotcha"!
- Ethereum is not re-entrancy "safe". *You can still have multiple calls to the same routine within any given call-stack.*

Re-entrancy

```
function transferAndNotify(uint amount, address notifyContract) … {
    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }
    balances[notifyContract] += amount;

    // Call the specified contract to notify it that a deposit is coming
    notifyContract.notify(amount, "You are getting a deposit!");

    // Transfer the money
    balances[msg.sender] -= amount;
}
```

Re-entrancy

```
function transferAndNotify(uint amount, address notifyContract) … {
    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }
    balances[notifyContract] += amount;

    // Call the specified contract to notify it that a deposit is coming
    notifyContract.notify(amount, "You are getting a deposit!");

    // Transfer the money
    balances[msg.sender] -= amount;
}
```

What if this contract call calls us?

Re-entrancy

```
function transferAndNotify(uint amount, address notifyContract) … {
    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }
    balances[notifyContract] += amount;

    // Call the specified contract to notify it that a deposit is coming
    notifyContract.notify(amount, "You are getting a deposit!");

    // Transfer the money
    balances[msg.sender] -= amount;
}
```

What if this contract call calls us?

Re-entrancy

```
function transferAndNotify(uint amount, address notifyContract) … {
    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }
    balances[notifyContract] += amount;

    // Call the specified contract to notify it that a deposit is coming
    notifyContract.notify(amount, "You are getting a deposit!");

    // Transfer the money
    balances[msg.sender] -= amount;
}
```

What if this contract call calls us?

Re-entrancy

```
function transferAndNotify(uint amount, address notifyContract) … {
    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }
    balances[notifyContract] += amount;

    // Call the specified contract to notify it that a deposit is coming
    notifyContract.notify(amount, "You are getting a deposit!");

    // Transfer the money
    balances[msg.sender] -= amount;
}
```

What if this contract call calls us?

DAO disaster (2016)

- Decentralized Autonomous Organization
  - "Like a VC fund" but decentralized
  - Implementation: a contract that controls money, and directs its disbursement according to "shareholder votes"
  - Shareholders buy in, pool their ETH (sending to contract)
  - Then vote on investments, which are made together
  - Users can "split" a DAO

“The DAO”

```
function splitDAO(
  uint _proposalID,
  address _newCurator
) noEther onlyTokenholders returns (bool _success) {

  ...
  // XXXXX Move ether and assign new Tokens.  Notice how this is done first!
  uint fundsToBeMoved =
      (balances[msg.sender] * p.splitData[0].splitBalance) /
      p.splitData[0].totalSupply;
  if (p.splitData[0].newDAO.createTokenProxy.value(fundsToBeMoved)(msg.sender)
== false) // XXXXX This is the line the attacker wants to run more than once
      throw;

  ...
  // Burn DAO Tokens
  Transfer(msg.sender, 0, balances[msg.sender]);
  withdrawRewardFor(msg.sender); // be nice, and get his rewards
  // XXXXX Notice the preceding line is critically before the next few
  totalSupply -= balances[msg.sender]; // XXXXX AND THIS IS DONE LAST
  balances[msg.sender] = 0; // XXXXX AND THIS IS DONE LAST TOO
  paidOut[msg.sender] = 0;
  return true;
}
```

Solutions?

```
function transferAndNotify(uint amount, address notifyContract) … {
    if (globalLock == true) {
        revert("Locked.");
    }
    globalLock = true;

    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }

    // Call the specified contract to notify it that a deposit is coming
    notifyContract.notify(amount, "You are getting a deposit!");

    // Transfer the money
    balances[msg.sender] -= amount;
    balances[notifyContract] += amount;

    globalLock=false;
}
```

```
function transferAndNotify(uint amount, address notifyContract) … {
   if (globalLock == true) {
      revert(“Locked.”);
   }
   globalLock =

   if (balances[msg.s
      // insufficient b
      revert(‘Someth
   }

   // Call the specifi
   notifyContract.no

   // Transfer the m
   balances[msg.sen
   balances[notifyContract] += amount;

   globalLock=false;
}
```

Drawbacks of (global) locks:

1. Extra gas (due to stores/loads)

2. Can get “stuck” if you’re careless

3. Sometimes re-entrant calls are useful!

Check-Effects-Interaction pattern

- Most common solution is to follow a code pattern:
  - First perform all contract checks (CHECKS)
  - Second, update contract state (EFFECTS)
  - Finally, make any contract calls (INTERACTION)

Check-Effects-Interaction pattern

```
function transferAndNotify(uint amount, address notifyContract) … {
   // CHECK
   require (amount < balances[msg.sender]);

   // EFFECTS
   balances[msg.sender] -= amount;
   balances[notifyContract] += amount;

   // INTERACTION
   notifyContract.notify(amount, "You are getting a deposit!");
}
```

Contract upgrades

- Ethereum contracts are not (natively) upgradeable
  - Once a contract is deployed, it can self-destruct
  - But its code cannot be changed
  - But some contracts need to be upgraded (bug fixes, etc.)
    - How are we going to handle this?

“Library” pattern

- Some contracts use a second contract to store their implementation

“Library” pattern

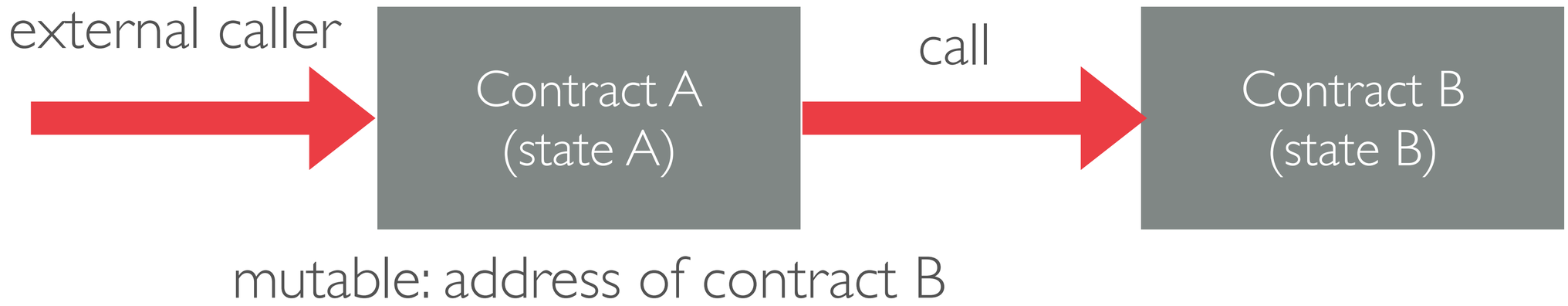
- Some contracts use a second contract to store their implementation
  - Enables code re-use (libraries)
  - Gets around code size limits

Proxy pattern

- Based on two standard features of Ethereum
  - An Ethereum contract (A) can call another contract (B) as a library

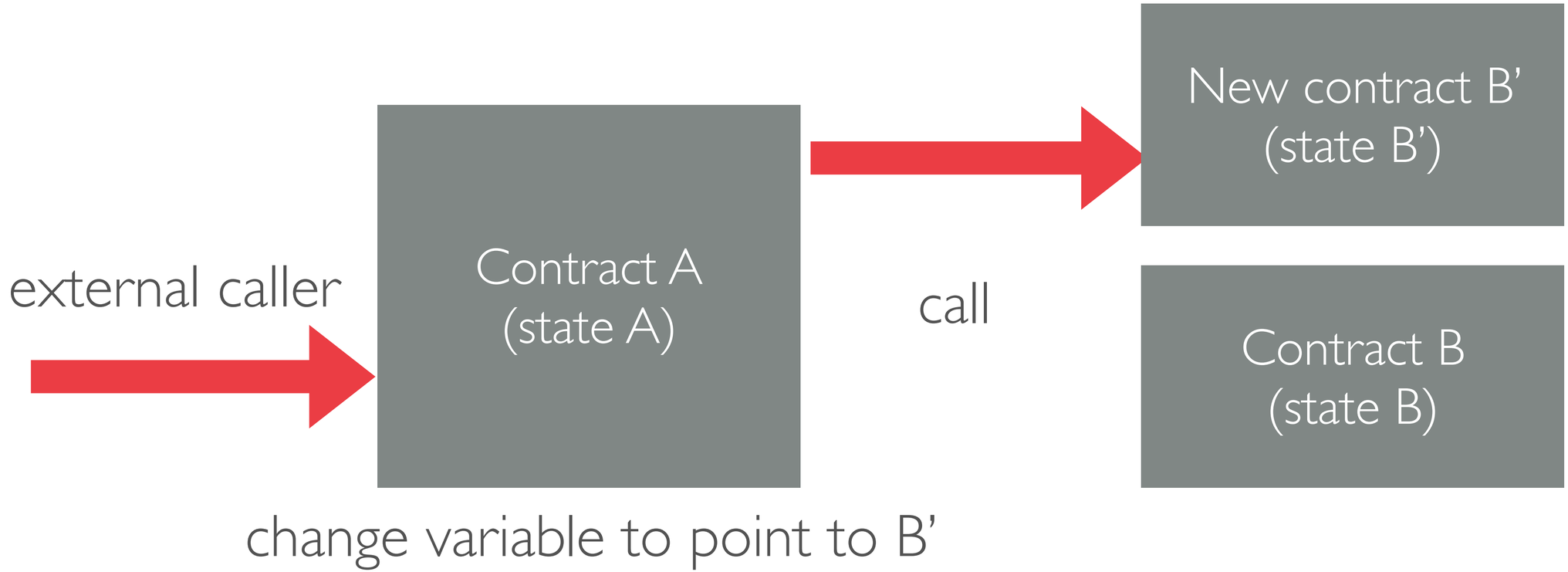
Proxy pattern

- Based on two standard features of Ethereum
  - An Ethereum contract (A) can call another contract (B) as a library
  - The location of that second contract can be stored in a mutable variable



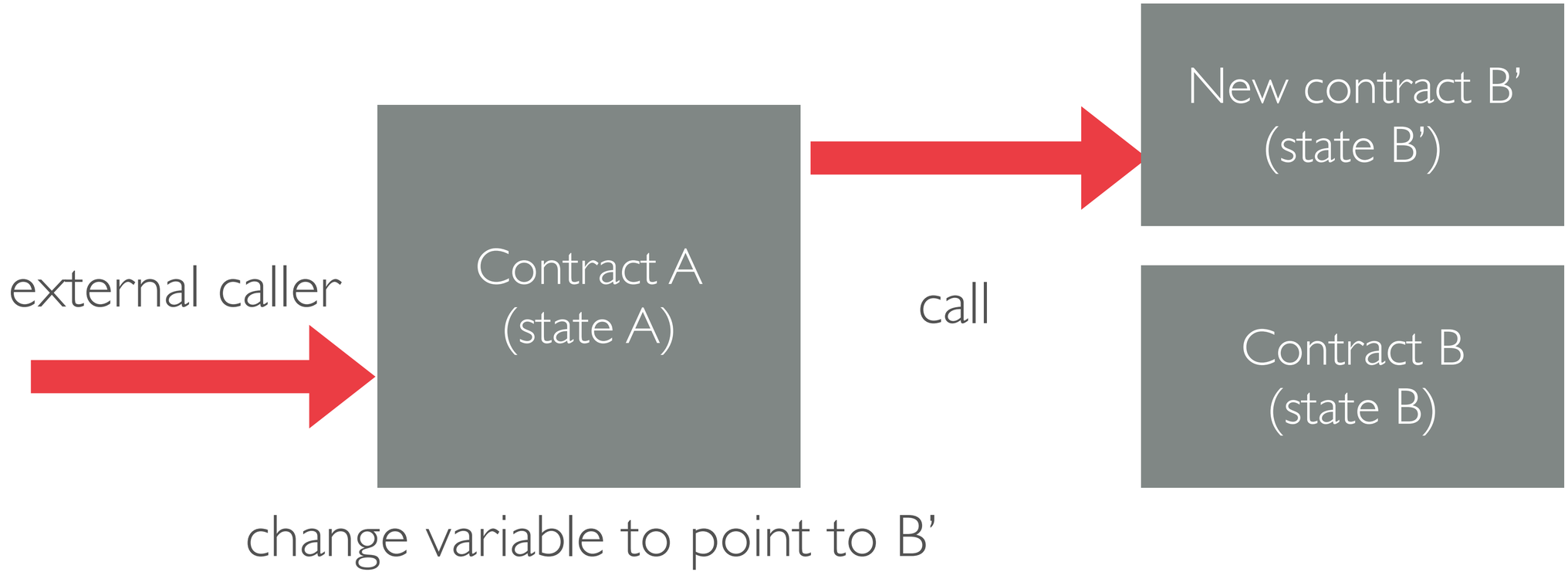
Proxy pattern

- Based on two standard features of Ethereum
  - We can always update that variable (using some special contract call we implement ourselves.)



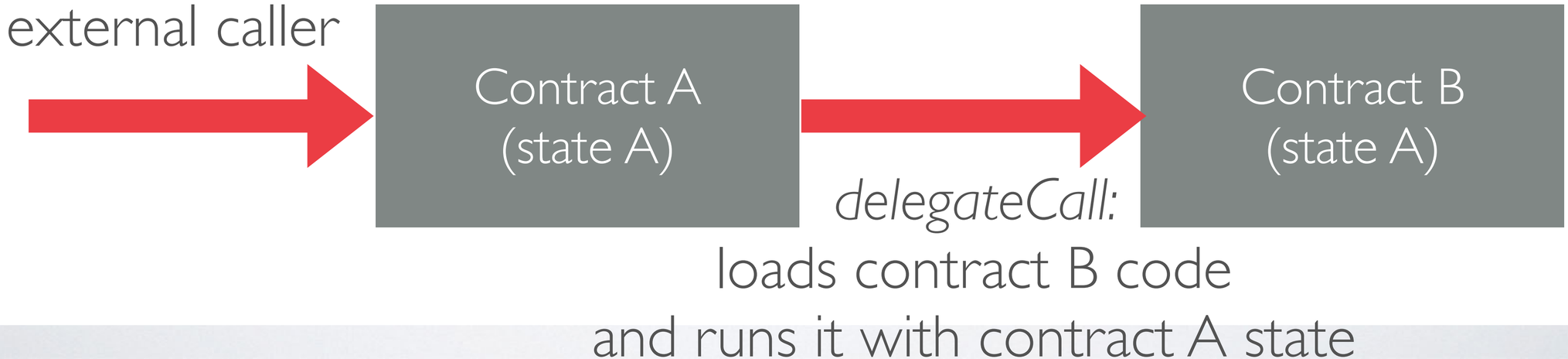
Proxy pattern

- **What are some problems with this solution (so far)?**

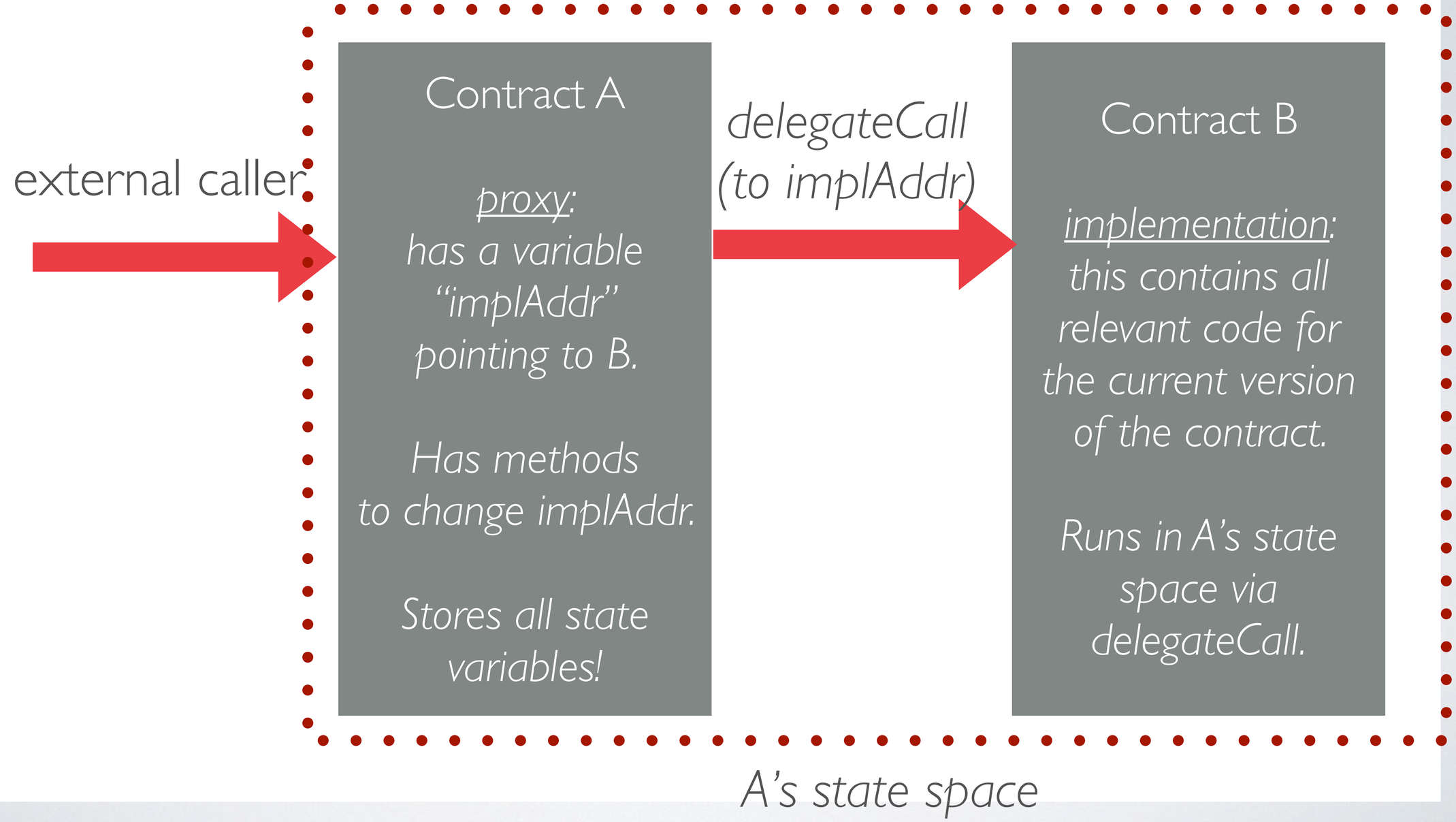


Delegate-Call

- Ethereum offers a nice "optimization"
  - A contract A can "load the code" from contract B into its own state context, run code B on A's state
  - This is called delegateCall
    - This is an exception to our normal isolation rules



Proxy pattern



Proxy upgrade

contract update
"please upgrade to
contract C"

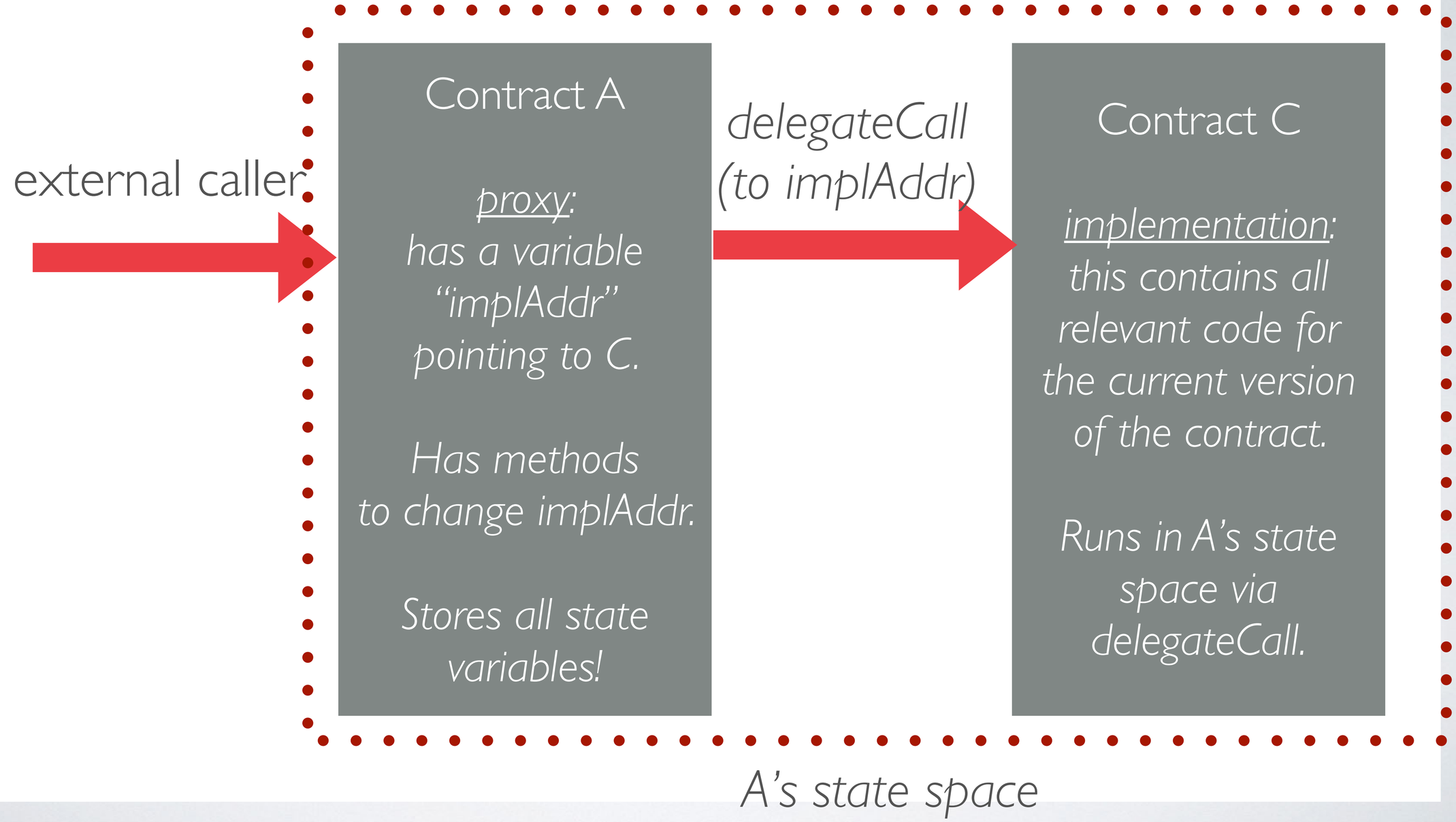
(e.g., called
by the "administrator"
of this contract)

Contract A

proxy:
check that upgrade
is authorized,
change "implAddr"
to point to new
Contract C

Contract C

After upgrade



Proxy costs

- There is always some overhead to proxy calls
  - Each call through the proxy involves (at least):
    1. A check to see if this is an upgrade call
    2. A variable load (lookup) to find *implAddr*
    3. A delegateCall

    Modern proxy contracts use EVM bytecode to minimize this

Proxy	Gas cost	Gas overhead
OpenZeppelin Transparent	29815	2770
Dharma Beacon	29752	2707
EIP-1882 UUPS	28679	1634
Storageless Beacon	28629	1584

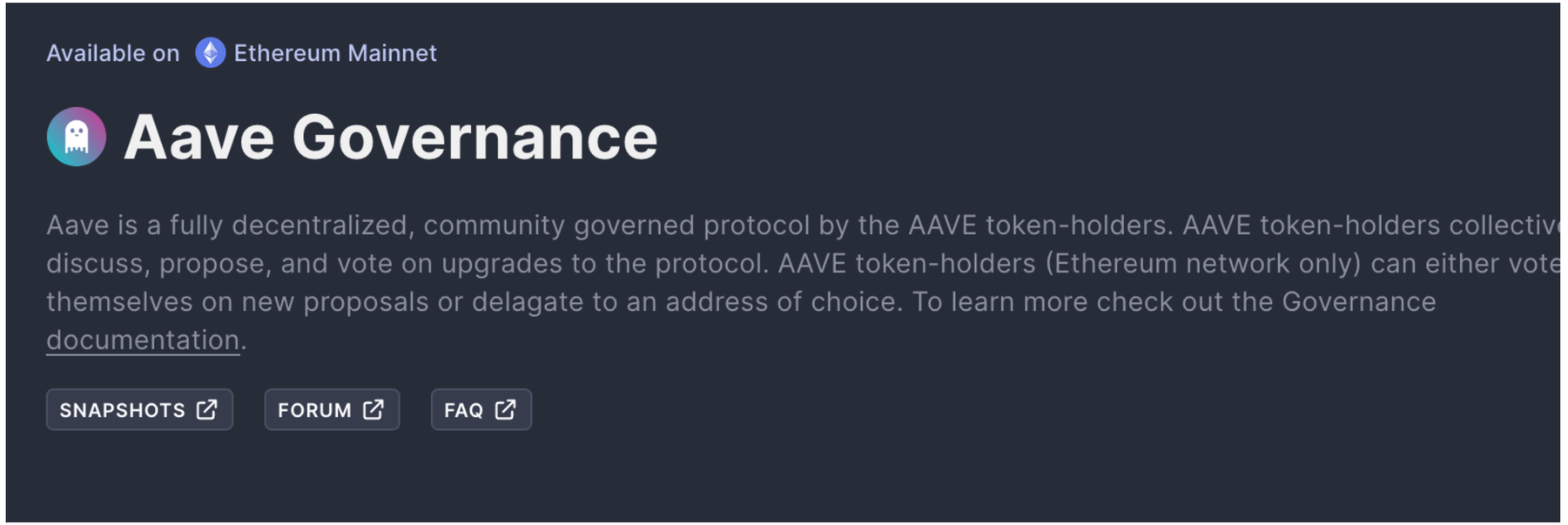
Proxies: advantages/disadvantages

- The ability to upgrade contract code is useful (bug fixes, etc.)
  - Counterargument: it can be very risky
  - If someone "hacks" your upgrade mechanism, they can steal all your money
  - What are some ways we can secure this upgrade process?
    - **Multi-sig** (require many signatures to upgrade contract)
    - **Voting** (require many token-holders to vote for an upgrade)
    - **Timelocks** (put a delay between start & end of an upgrade)

Contract governance

- The ability to upgrade contracts can be even more powerful
  - Enables the notion of "voting"-based contract governance
- **Idea**: issue a "governance token" (e.g., ERC20)
  - People might get the token in exchange for depositing real money, or for other reasons
  - This token gives holders a "vote" on contract governance
  - People can propose new features for the contract (in practice, this is essentially a contract upgrade proposal)
  - Voters can then review/vote/adopt the new features

Contract governance



Contract governance: dark side

- An observation in practice:
  - Contract governance is sometimes a form of "regulatory arbitrage"
  - Often: the contract developers want to argue that they are simply a software developer, and do not control this "decentralized protocol"
  - However they (or their friends/partners/VCs) may in fact control the bulk of the voting tokens
    - Hence one should always be a little bit skeptical of these claims

Contract governance: dark side II

News ▸ Technology

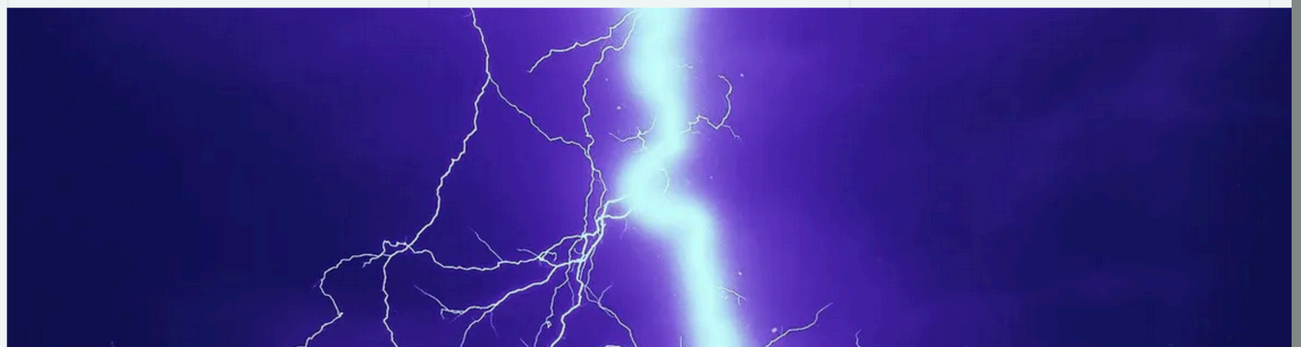
Group Uses Flash Loan to Game Maker Protocol Governance

BProtocol used a DeFi arbitrage tool to push through a governance vote it had proposed on the Maker protocol.

By Jeff Benson

Oct 29, 2020

2 min read



I intend to return to this (if there is time)…

Standards Track: ERC

ERC-2535: Diamonds, Multi-Facet Proxy

Create modular smart contract systems that can be extended after deployment.

Authors Nick Mudge (@mudgen)

Created 2020-02-22

Table of Contents

- Abstract
- Motivation
    - Upgradeable Diamond vs. Centralized Private Database
    - Some Diamond Benefits
- Specification
    - Terms
    - Overview
    - A Note on Implementing Interfaces
    - Fallback Function
    - Storage
    - Solidity Libraries as Facets
    - Adding/Replacing/Removing Functions
    - Inspecting Facets & Functions

As state space

Decentralized Exchanges

- Given multiple assets (ERC20s) on Ethereum, can we build platforms to exchange them?
  - Breaks down into two-subproblems:

    1. Price discovery / order matching
    2. Execution (swap)

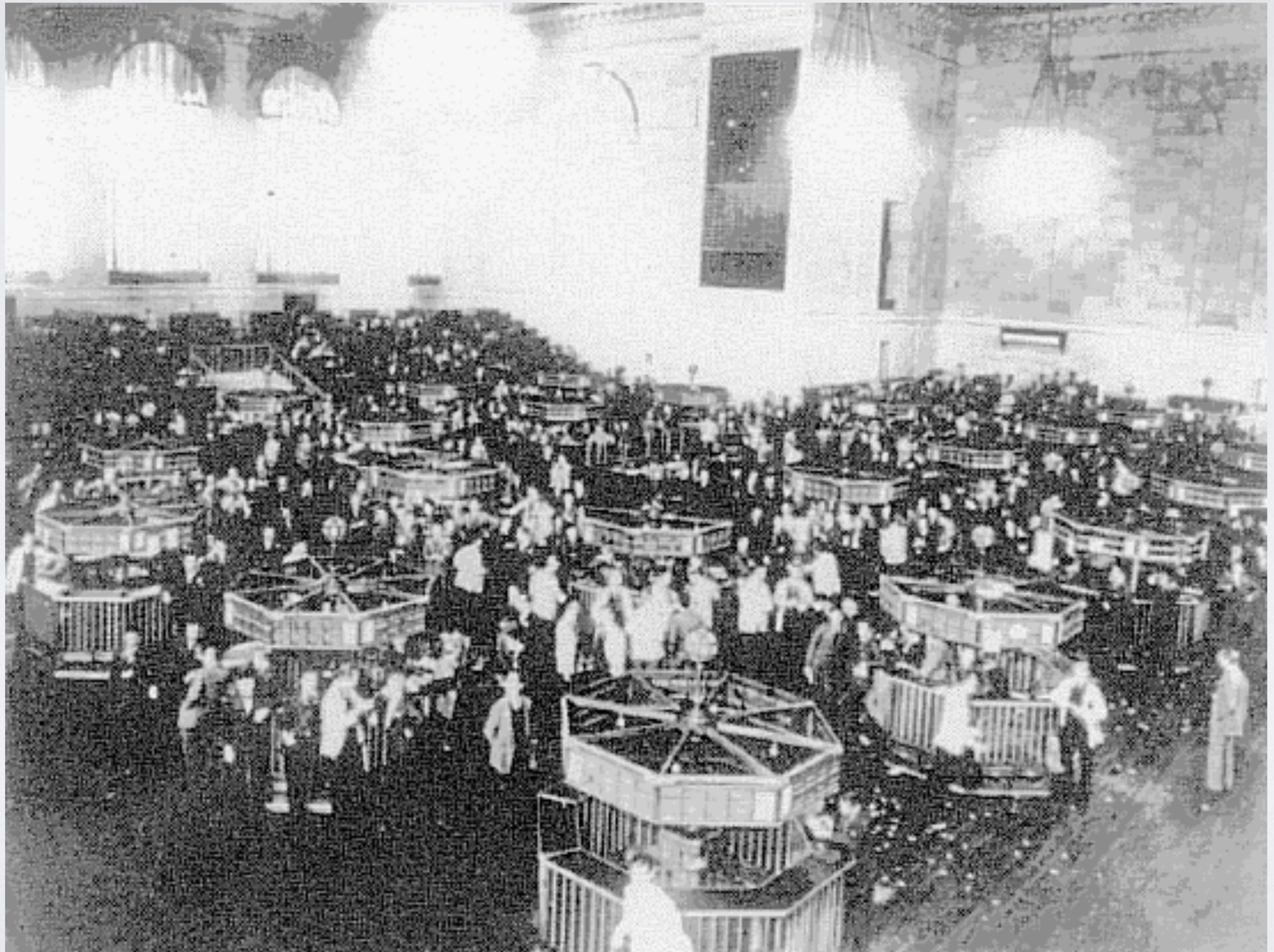
Decentralized Exchanges

- Given multiple assets (ERC20s) on Ethereum, can we build platforms to exchange them?
  - Breaks down into two-subproblems:

    1. Price discovery / order matching
    2. Execution (swap)

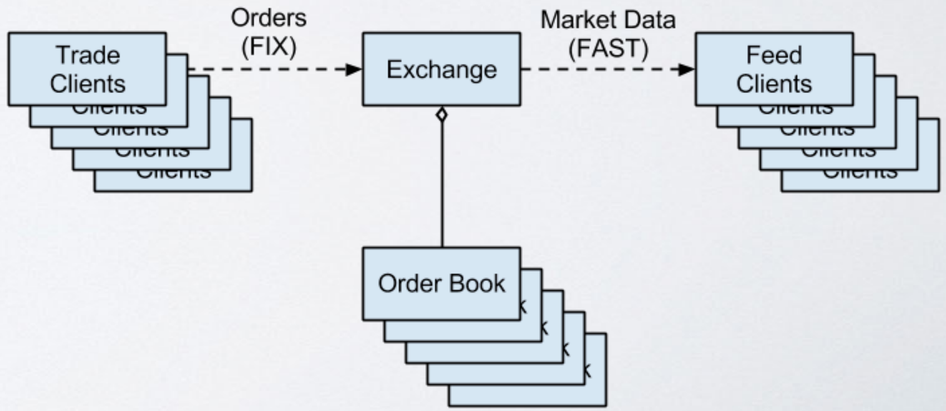
This part is relatively easy

Traditional (centralized) exchange

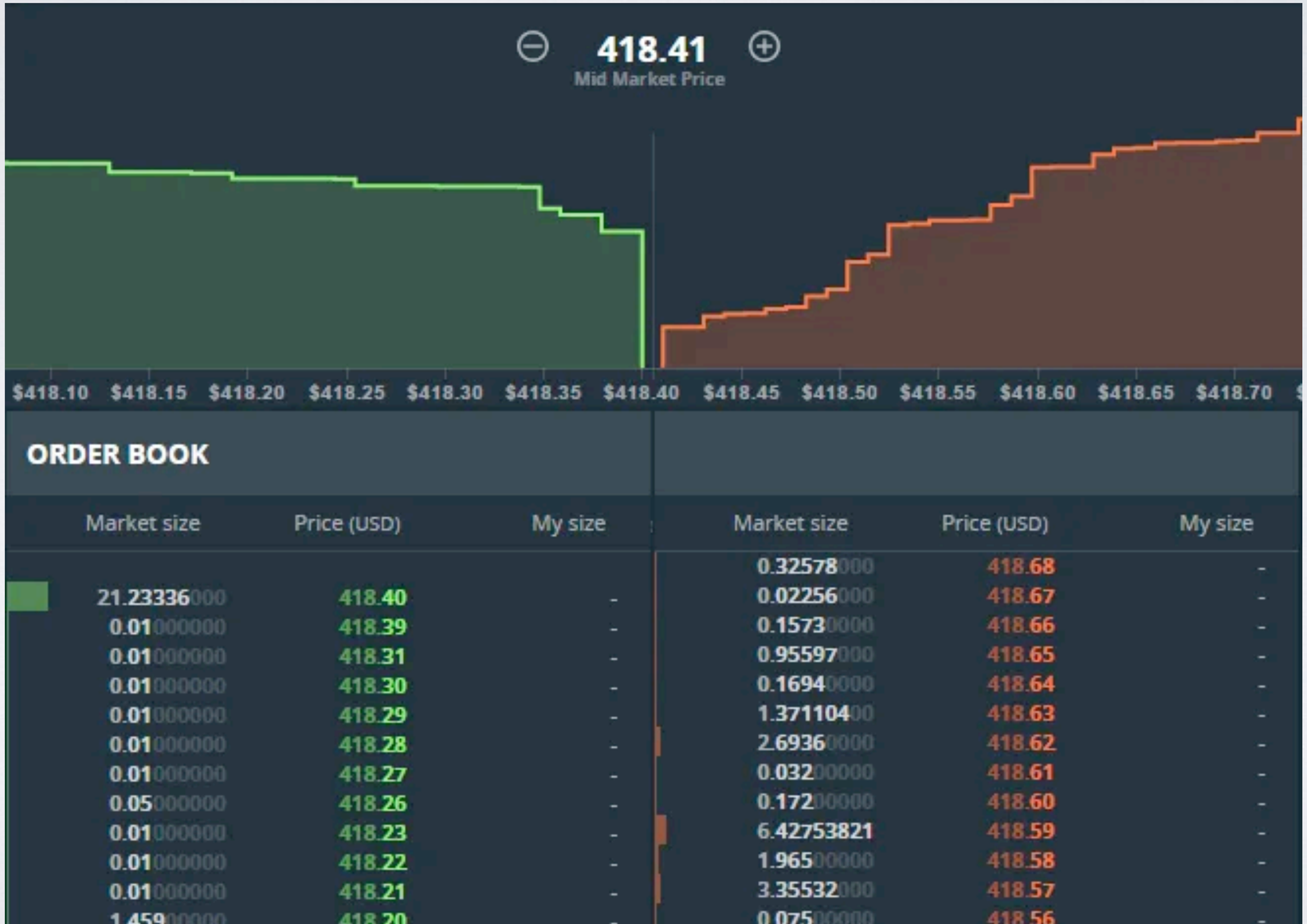


Electronic centralized exchange

- Maintains an "electronic order book"
  - Receives order of various types (e.g., market, limit, etc.)
  - Implements a matching engine to reconcile orders
  - Executes trades (swaps)
  - Produces asset price data (as a side effect)



Electronic order book

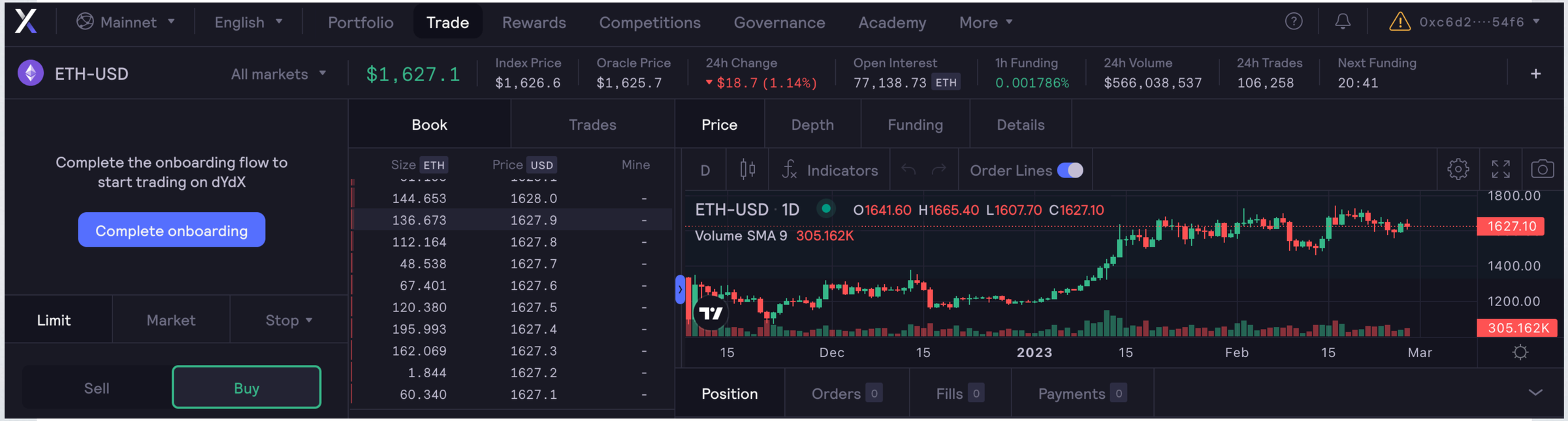


Electronic order book

- Can we do this purely on Ethereum?
  - Why/why not?

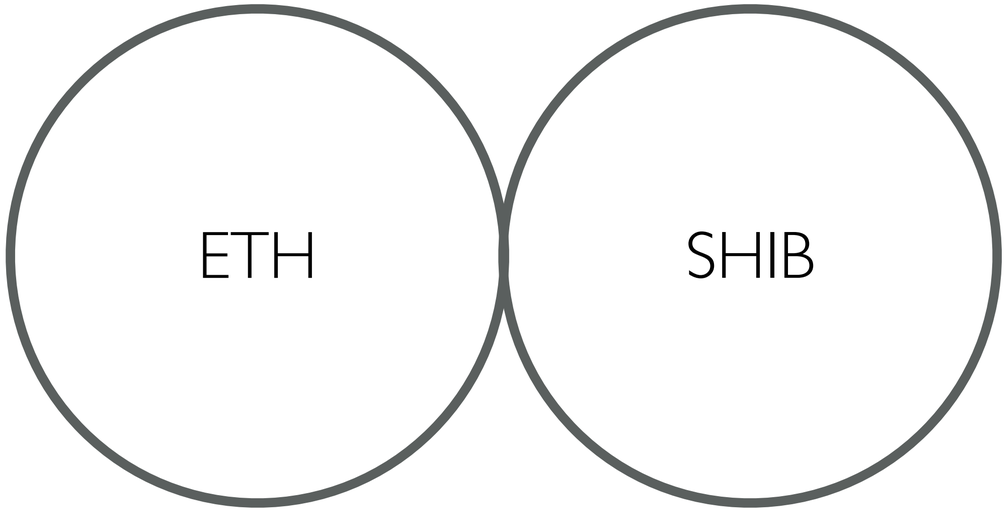
Electronic order book

- Can we do this on Ethereum?
  - Why/why not?



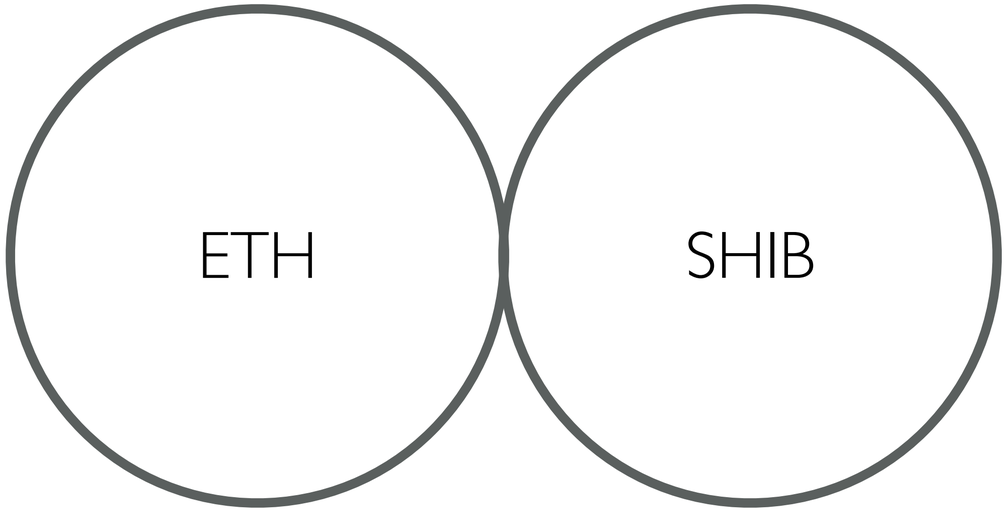
Constant Function Market Makers (AMM)

- A simpler (and much less efficient) type of exchange
  - Built from pairs of assets (e.g., ETH/SHIB)
  - Liquidity providers can deposit and withdraw asset pairs (in some ratio), forming a "liquidity pool"



Constant Function Market Makers (AMM)

- A simpler (and much less efficient) type of exchange
  - Built from pairs of assets (e.g., ETH/SHIB)
  - New LPs can increase/reduce the overall size of both pools, without changing the ratio

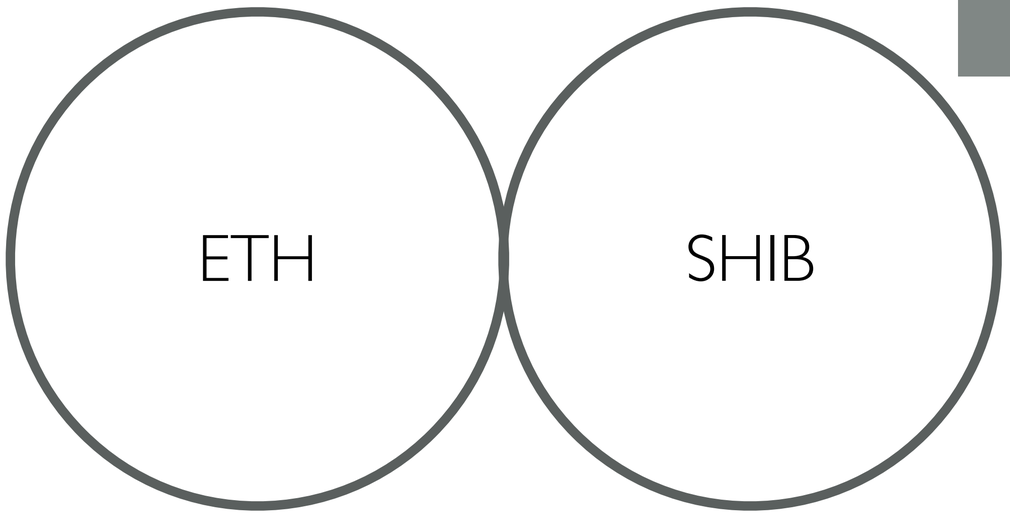


Constant Function Market Makers (AMM)

- A simpler (and much less efficient) type of exchange
  - Built from pairs of assets (e.g., ETH/SHIB)
  - Users can also "swap" (deposit one asset, withdraw the other), in exchange for fees

Overall goal:
Preserve a constant function (e.g., product, sum). E.g.,:

$$A * B = K$$

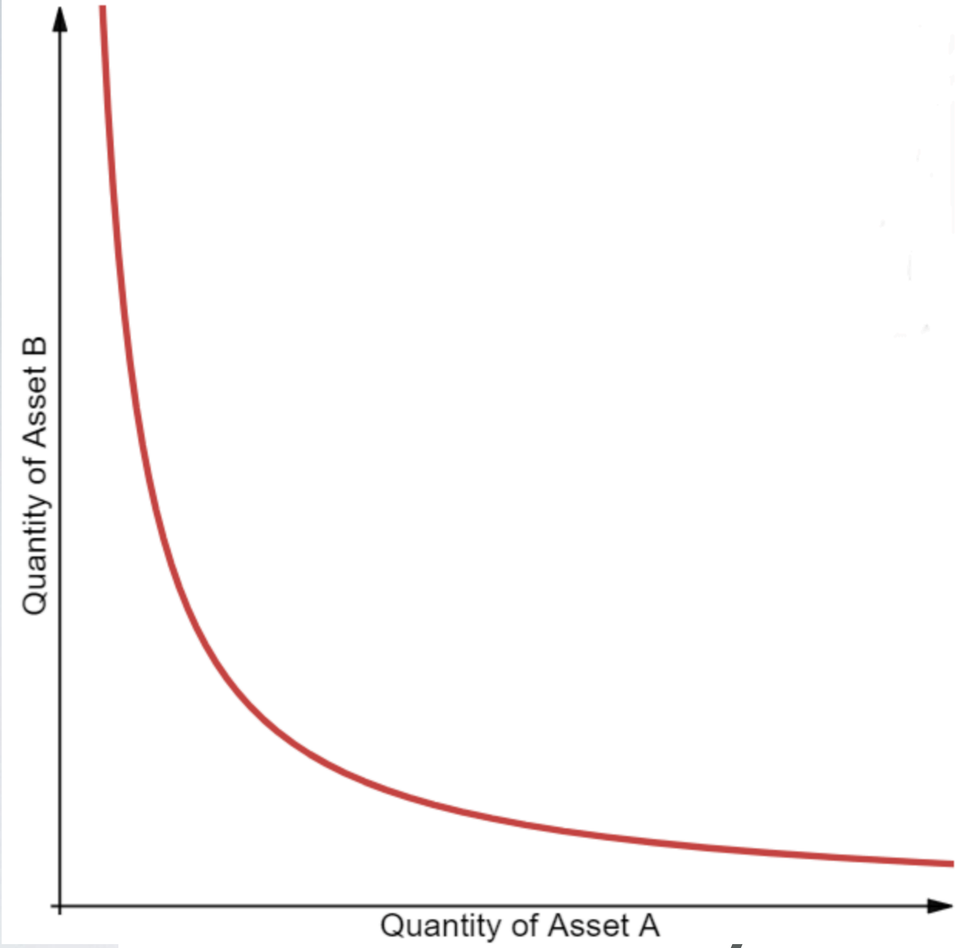


ɔn Market Makers (AMM)

ess efficient) type of exchange

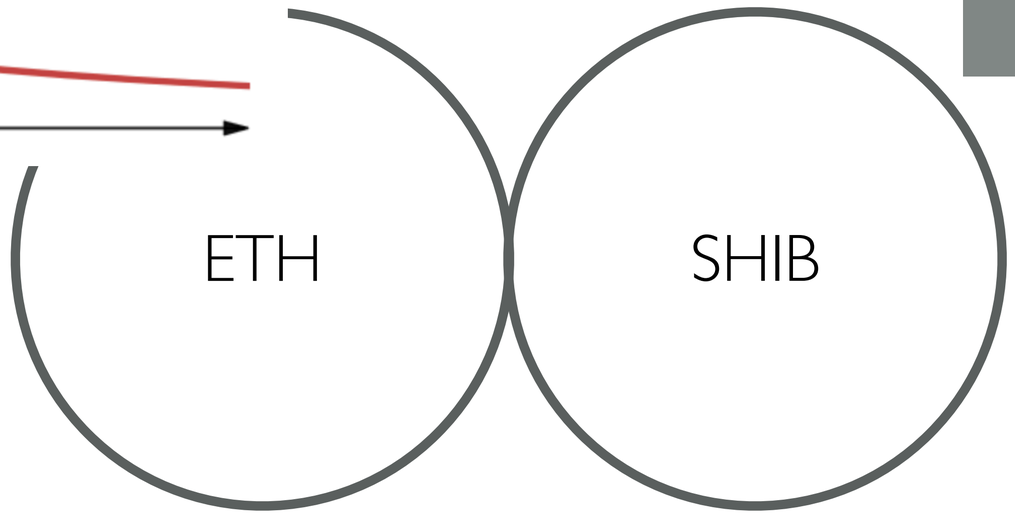
sets (e.g., ETH/SHIB)

d withdraw assets (in vari
ɔol”



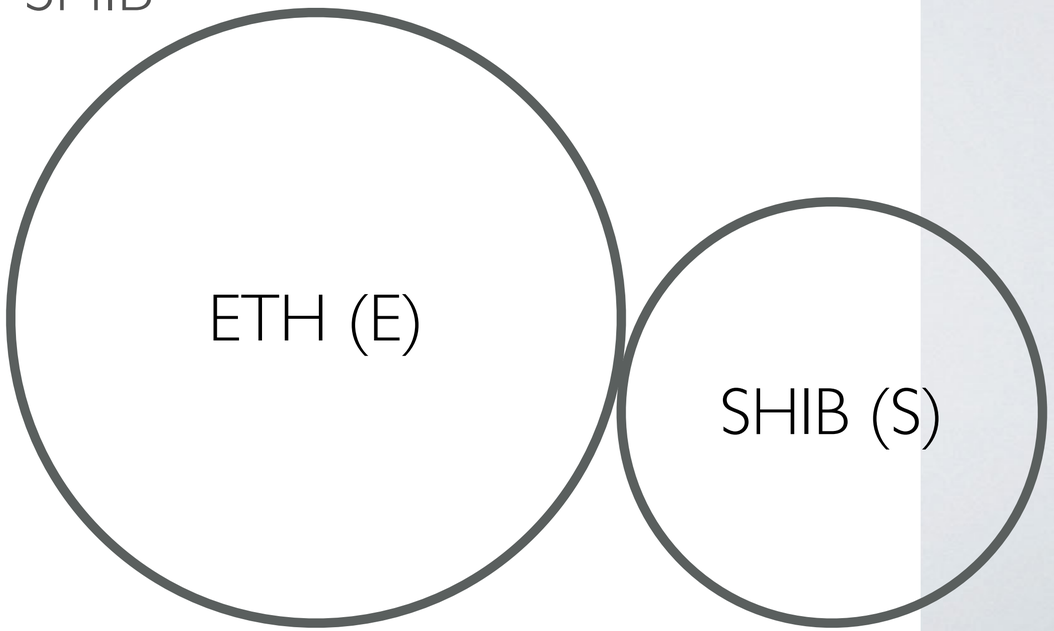
Overall goal:
Preserve a constant function (e.g., product, sum). E.g.,:

$A * B = K$

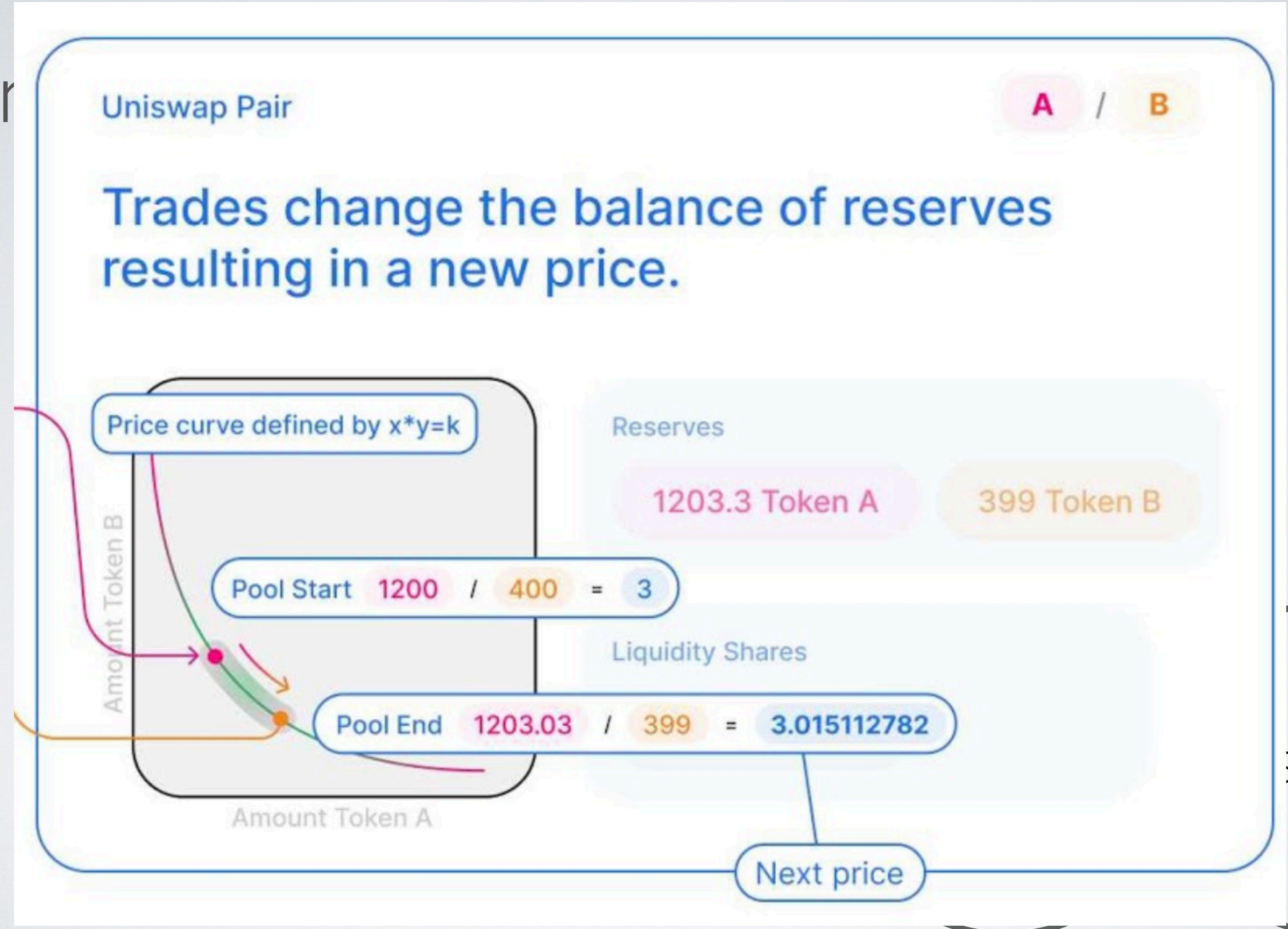


Constant Function Market Makers (AMM)

- User A deposits ETH, wants to “buy" SHIB
  - Initial equation: E * S = K
  - Defines an implicit price (ratio) between the assets
  - They can now deposit ETH & withdraw SHIB at that price (plus pay some fees)
    - This trade changes the market price
    - New price ratio for SHIB: (K / E’) / S’



Cor M)

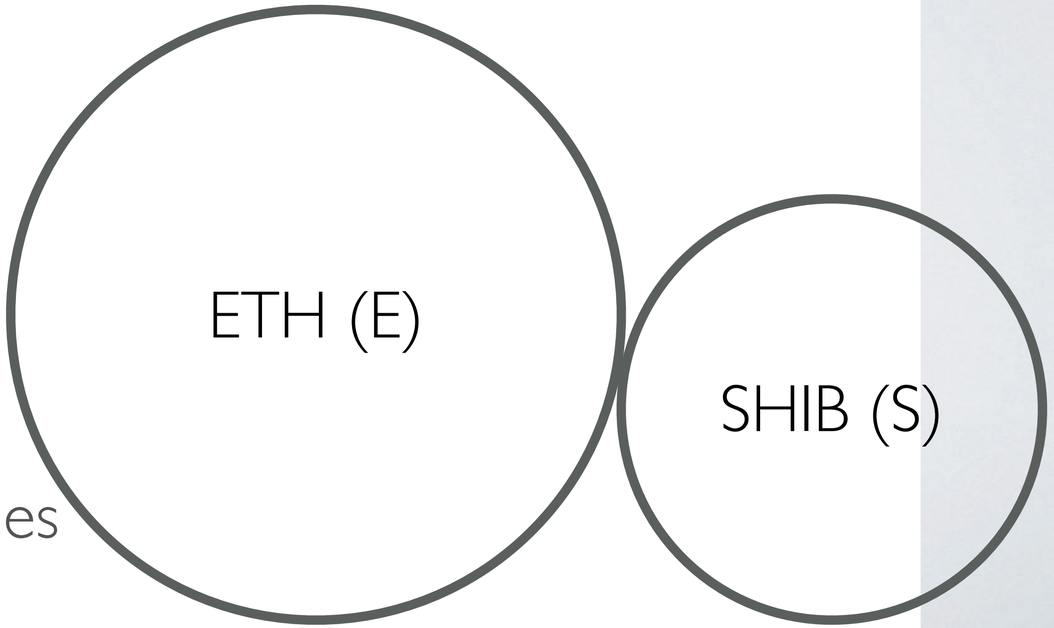


SDC (U)

https://academy.shrimpy.io/lesson/what-is-an-amm-automated-market-maker

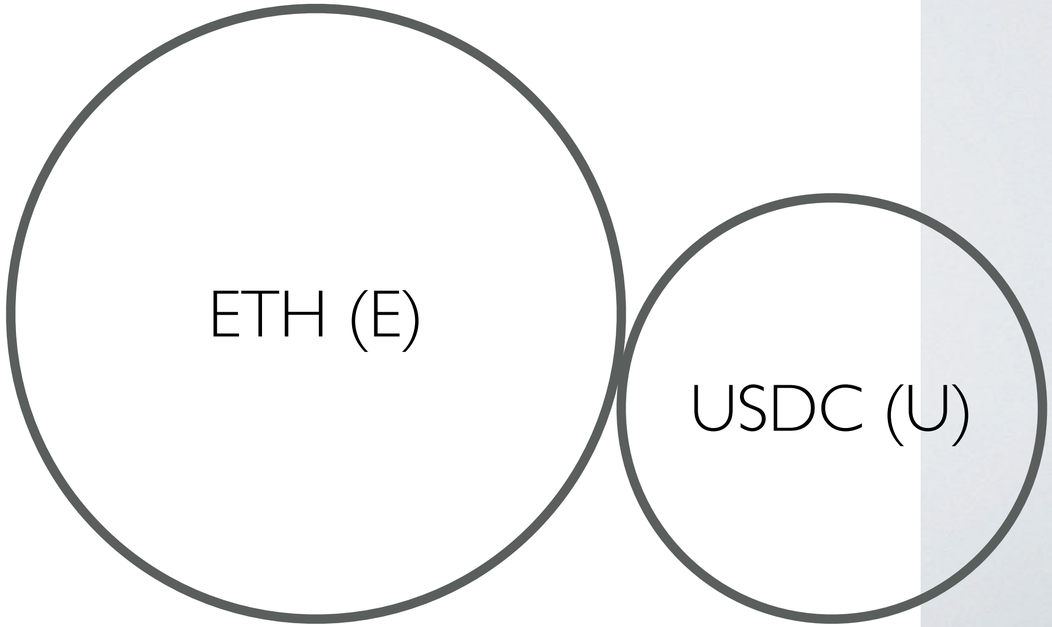
Arbitrage / MEV

- In principle, the price of assets should stay "close" to the true price (whatever that is)
  - If the ratio gets far out of whack, hungry traders will "adjust" the pool sizes by executing profitable swaps
  - If there is no pool for an asset pair, systems can "route" trades through multiple pools with more liquidity
    e.g., ETH->USDC, USDC->SHIB
  - Slippage can be quite large
  - Relatively easy to "front-run" public trades

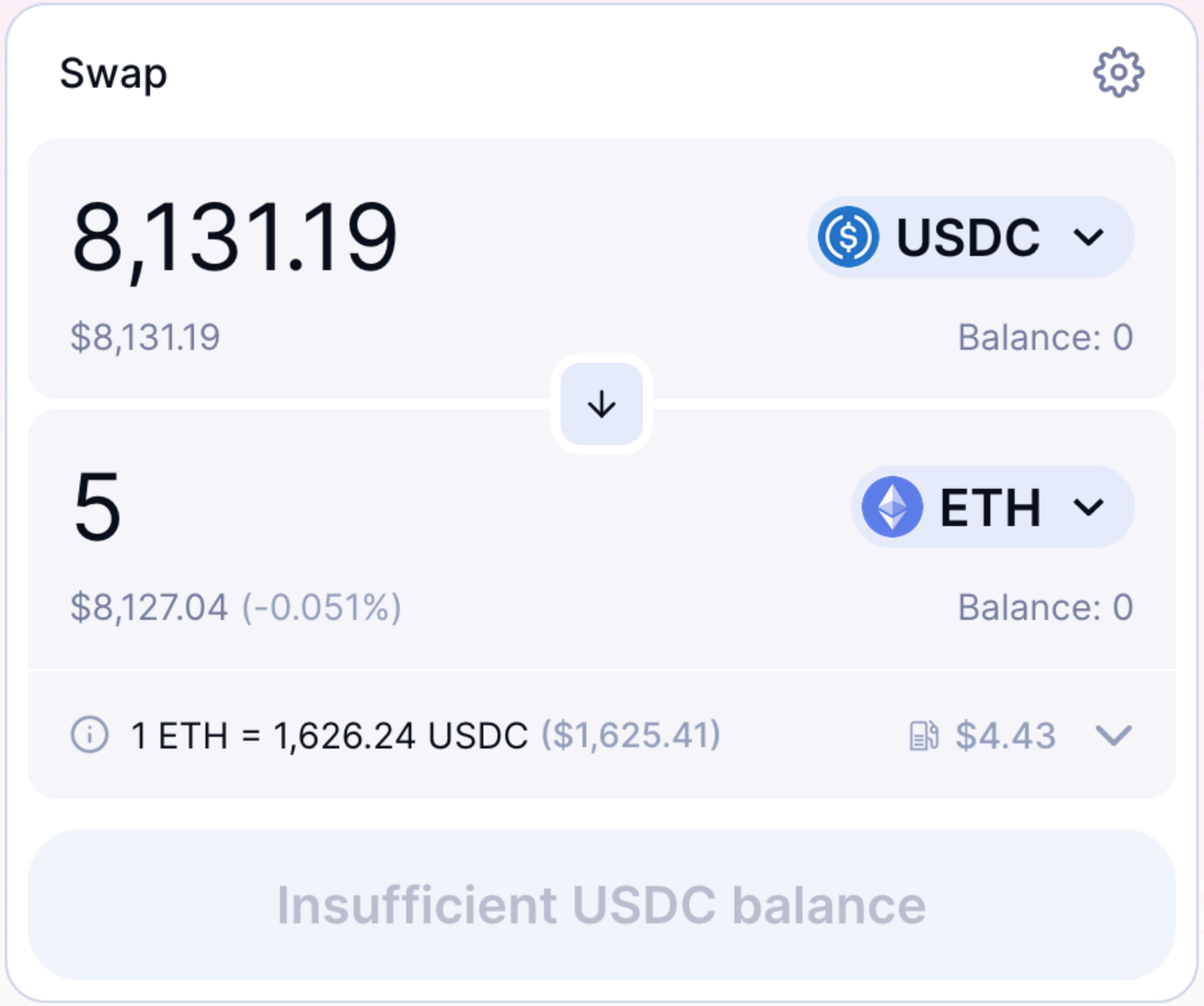


Why put your "liquidity" in these pools?

- The liquidity providers receive fees from the trade (usually split with the DeFi operators, developers)
  - They receive "Liquidity Provider Tokens" (yet another made-up ERC20) that they can later redeem to get money back out (in a new ratio)
  - **Criticism:** being an LP is dumb, because arbitrage is profitable and you are the loser in all those trades
  - Many LPs have been "incentivized" to participate by weird token schemes



W s?

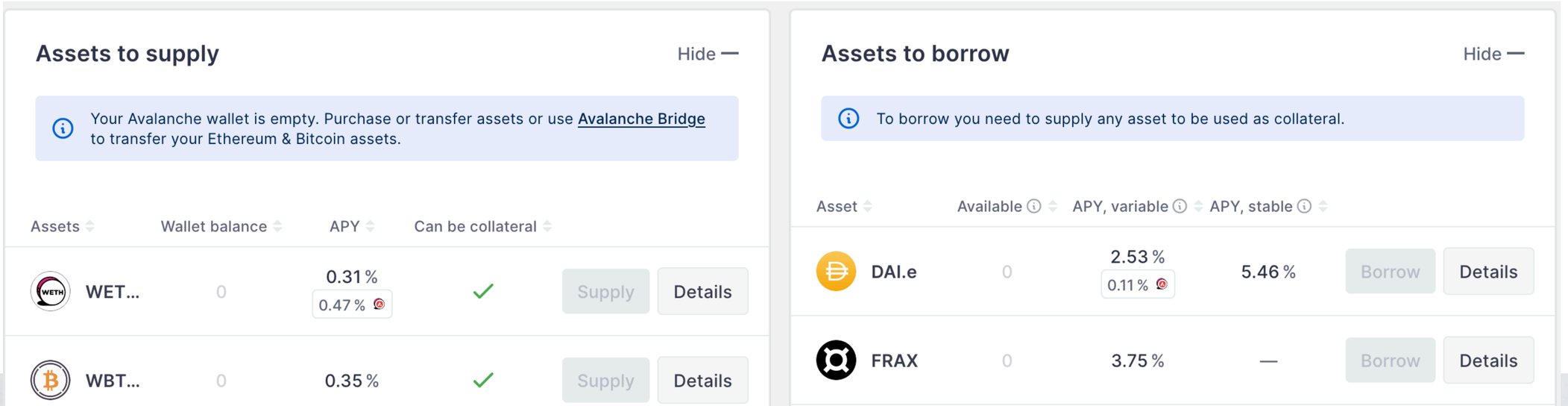

Swap
8,131.19
USDC
$8,131.19
Balance: 0
5
ETH
$8,127.04 (-0.051%)
Balance: 0
1 ETH = 1,626.24 USDC ($1,625.41)
$4.43
Insufficient USDC balance

Asset lending

- Another thing smart contracts can do easily is asset lending
  - **Problem:** smart contracts are not great at getting people to repay asset loans
  - This means you need some way to insure those loans
  - Answer: require (over)collateralization

Asset lending

- Another "popular" DeFi service is collaterized asset lending (borrowing)
  - You wish to borrow some amount of Asset B (for whatever)
  - You can deposit and "lock up" some amount of Asset A as collateral for the loan (and pay interest)



Asset lending

- If lenders don't repay the principal, their collateral is automatically liquidated to repay the lenders
  - Hence collateral value must (substantially) exceed loan value
  - If the value of the collateral drops, the system may automatically liquidate it without warning
  - **Question:** How do these smart contracts know the "value" of anything?

Asset lending

- If lenders don't repay the principal, their collateral is automatically liquidated to repay the lenders
  - Hence collateral value must (substantially) exceed loan value
  - If the value of the collateral drops, the system may automatically liquidate it without warning
  - **Question:** How do these smart contracts know the "value" of anything?
  - **Answer:** services provide price "oracles" to these systems, by sending Txes (to some contract) that contain current prices, e.g., ChainLink.
    (Or they can query AMM contracts!)

Asset lending

Assets to supply

Hide —

Your Avalanche wallet is empty. Purchase or transfer assets or use Avalanche Bridge to transfer your Ethereum & Bitcoin assets.

Assets	Wallet balance	APY	Can be collateral		
WET...	0	0.31 % 0.47 %	✓	Supply	Details
WBT...	0	0.35 %	✓	Supply	Details
WAVAX	0	1.65 % 1.15 %	✓	Supply	Details
AVAX	0	1.65 % 1.15 %	✓	Supply	Details

Assets to borrow

Hide —

To borrow you need to supply any asset to be used as collateral.

Asset	Available	APY, variable	APY, stable		
DAI.e	0	2.53 % 0.11 %	5.46 %	Borrow	Details
FRAX	0	3.75 %	—	Borrow	Details
MAI	0	3.51 %	—	Borrow	Details
USDC	0	2.31 % 0.17 %	5.43 %	Borrow	Details

Asset lending (horror stories)

- There are big risks here, if the collateral asset is badly priced (or thinly traded)
  - Show up with some fake nonsense-coin, that has been "wash traded" into appearing to have value
  - Borrow actual money with it
  - Walk away and never come back
- So these systems are not usually unsupervised… and lenders can lose money

Flash loans!

- It is possible to make loans that must be repaid within the course of a single transaction

Frontrunning

Re-entrancy

DAO disaster

- Decentralized Autonomous Organization
  - "Like a VC fund" but decentralized
  - Implementation: a contract that controls money, and directs its disbursement according to "shareholder votes"
  - Shareholders buy in, pool their ETH (sending to contract)
  - Then vote on investments, which are made together
  - Users can "split" a DAO

“The DAO”

```
function splitDAO(
  uint _proposalID,
  address _newCurator
) noEther onlyTokenholders returns (bool _success) {

  ...
  // XXXXX Move ether and assign new Tokens.  Notice how this is done first!
  uint fundsToBeMoved =
      (balances[msg.sender] * p.splitData[0].splitBalance) /
      p.splitData[0].totalSupply;
  if (p.splitData[0].newDAO.createTokenProxy.value(fundsToBeMoved)(msg.sender)
== false) // XXXXX This is the line the attacker wants to run more than once
      throw;

  ...
  // Burn DAO Tokens
  Transfer(msg.sender, 0, balances[msg.sender]);
  withdrawRewardFor(msg.sender); // be nice, and get his rewards
  // XXXXX Notice the preceding line is critically before the next few
  totalSupply -= balances[msg.sender]; // XXXXX AND THIS IS DONE LAST
  balances[msg.sender] = 0; // XXXXX AND THIS IS DONE LAST TOO
  paidOut[msg.sender] = 0;
  return true;
}
```

How to upgrade a contract?

How to upgrade a contract?

- Contracts are default immutable
- If there is no useful ongoing state, don't: just replace it
- If there is ongoing state (e.g., account balances) then:
  - Don't allow upgrades — and pray you got the code right
  - Call upgradeable/replaceable library code
  - Create a complex mechanism to transfer state from an old contract instance to a new contract instance

Future of Ethereum

- Proof of stake
- Rollups
- Sharding