TOKENS (CONT.) & SMART CONTRACT SECURITY

Kai Mast
CS639/839
Spring 2023

ANNOUNCEMENTS

- New mini project will be out sometime this week
- Start thinking about projects!
  - Feel free to meet with me to talk about ideas
- There will be an *optiona*l lecture series on blockchains with guest speakers
  - Usually on Monday's at noon
  - I will send out more information soon
- Lecture on 3/8 will be online

TODAY'S AGENDA

- Recap: Ethereum/DApps
- More Token Content
  - Decentralized Autonomous Organizations
- Decentralized Exchanges
  - Automated Market Makers
- Break
- Smart Contract Security
  - Reentrancy Attacks
  - The DAO hack

RECAP: BLOCKCHAIN EXECUTION

In Ethereum-like protocols

- Blocks are created periodically (we talk soon about how)
- Each node executes and validates blocks in order
  - We can only execute a block if we executed its predecessor

For each block

- Execute all transactions
  - Usually *in sequenc*e; there is not concurrency
  - Reject block if any transaction is invalid
- Compare resulting state and receipts with those in the block header
  - If not, reject block

TRANSACTION TYPES

(Externally-Owned) Account Creation

- Send Ether to an unused address

Contract Creation

- Send Code to an unused address
- (Optionally) also send some Ether

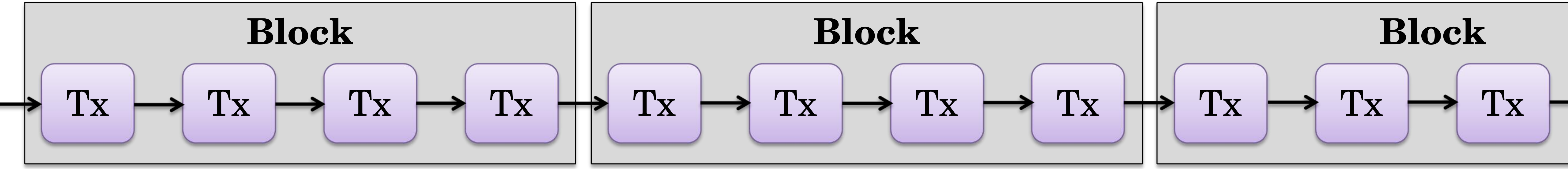
Simple Payments

- Send money to another account
- Data field is empty (or ignored by the recipient)

Function Invocation

- Data field contains function identifier and arguments
- (Optionally) also send some Ether

TRANSACTION EXECUTION (IN ETHEREUM)



Execution of transaction updates the *state* of the blockchain

- Transactions execute sequentially and as a single "thread"
- If a transaction aborts (is reverted), it will not make any changes to the state
- Each transactions sees the changes created by the previous transaction

BLOCKCHAIN STATE

In the UTXO-Model (e.g., Bitcoin):

- Blockchain state is the set of unspent transaction outputs

In the Account Model (e.g., Ethereum):

- Blockchain state contains all existing accounts and their data
- For each account, the ledger needs to store
  - Balance
  - Code (if any)
  - Custom Data (if any)

FUNCTION CALLS IN CONTRACTS

Internal Calls

- Call a function within the same contract
- No context switch needed

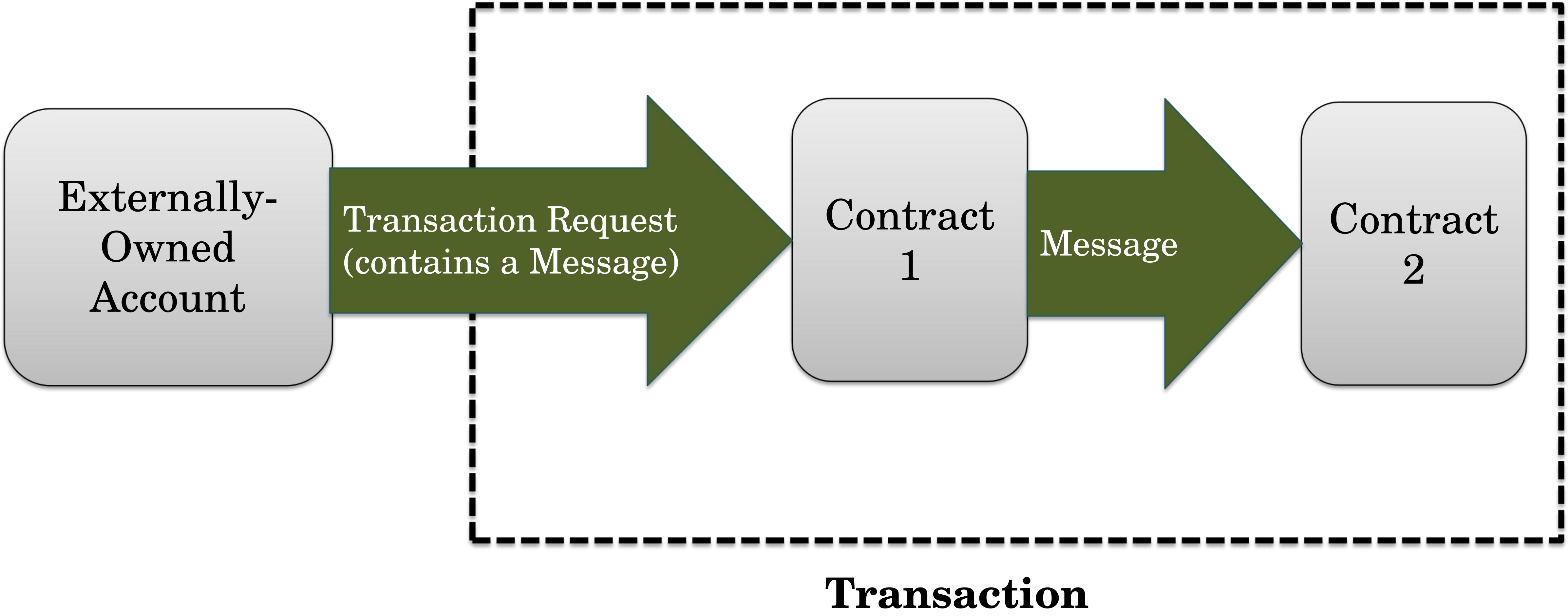
External Calls

- Call a function of another contract
- Requires a context switch

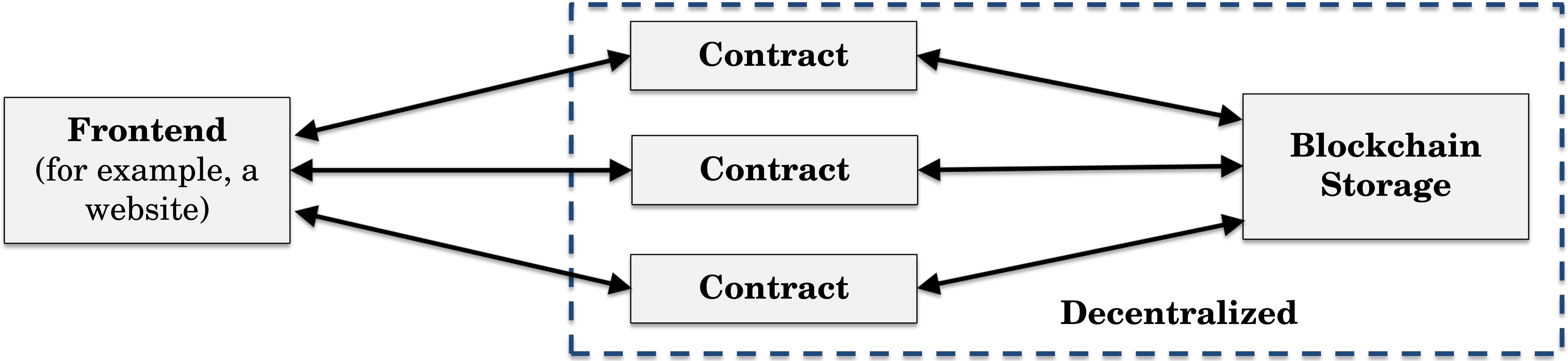
Library Calls (or “Delegate Calls”)

- Run the code of another contract in the current context
- Allows reusing code

FUNCTION CALLS IN CONTRACTS



WEB3 AND DAPPS



Smart contracts: Small-ish programs executing on the chain

Decentralized Apps: Applications backed by smart contracts and a blockchain

Web 3.0: DApps replace conventional/centralized web services

EVENT LOGS

Contract Code

```
# A player won
event Winner:
    player: address

@external
def set(i: uint256, j: uint256):
    [...]
    if self.check_winner(from_player):
        self.winner = from_player
        log Winner(msg.sender)
```

Frontend Code

```
var ttt = web3.eth.contract(abi);
var c = ttt.at("0x1234...ab67");

// watch for changes in the callback
var c = ttt.Winner(
  function(error, result) {
     if(!error) {
        console.log(“Somebody won!”);
     }
  }
);
```

Transaction Receipts

- Certifies the output of a transaction
- Like state, they are not stored on the blockchain
- Unlike state, they *cannot be accessed by smart contracts*

What do they contain?

- `GasUsed`: The actual amount of gas consumed
- `Status`: Whether the transaction succeeded
- `Log`: Data logged by the transaction

CONTRACT INTERFACES

```
from vyper.interfaces import ERC20

implements: ERC20
```

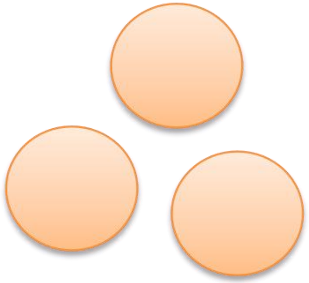
Interfaces provide a common API/ABI across smart contracts

Some interfaces are standardized, e.g.

- ERC20 for fungible tokens
- ERC165 for interface discovery
- ERC721 for non-fungible tokens

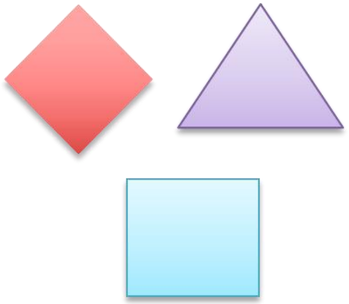
TOKENS

Two types of tokens exists



Fungible Tokens (or just "Tokens"):

- Tokens are interchangeable and separable
- E.g., shares of a company or a
- ERC20 is the standardized interface



Non-Fungible Tokens (NFTs):

- Each token is unique
- E.g., certificate of ownership of an asset
- ERC721 is the standardized interface for NFTs

FUNGIBLE TOKEN EXAMPLE: DAO

Decentralized **A**utonomous **O**rganizations

- First proposed in the Ethereum Whitepaper

People can buy *shares* (the token)

- DAO pools money of share holders

Potential Use Case of a DAO: Crowdfunding

- DAO accepts proposals for investments
- Shareholders vote on proposals
- Voting power is proportional to the number of shares held
- If vote succeeds, DAO invests

BUYING/SELLING SHARES: CODE

NFT EXAMPLE: OPENSEA

OpenSea is not an NFT, but an NFT marketplace

- Different tokens are traded on OpenSea

NFTs only contain a reference to the image/item

- Actual contents (images etc.) are stored elsewhere
- E.g., on IPFS

Token contract can generate information about the token

- For example, where the image is located
- Or, for example, what properties the “cryptokitty” has

EXCHANGES

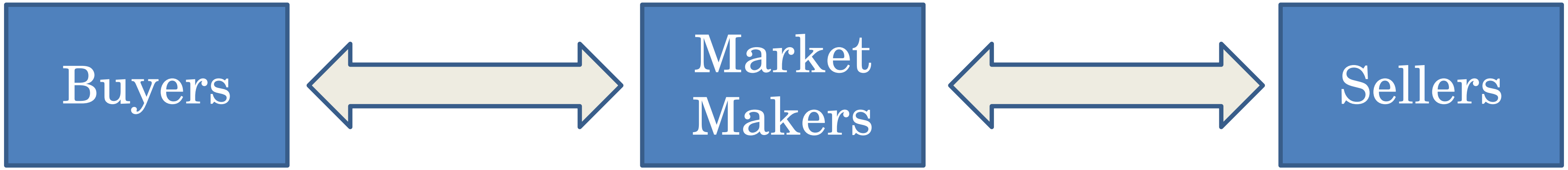
Why?

- Buy/sell tokens at current market value

Two Types

- Centralized
  - backend by a legal entity, e.g., Coinbase
- Decentralized
  - backed by a blockchain

MARKET MAKERS



Market makers serves as an intermediate between sellers and buyers

- Market makers can profit from the "ask-bid spread" (difference between sell and buy offers)

Also called *"liquidity provider"*

- They offer a large quantity of the traded item(s) to facilitate continuous trade

UNISWAP

- Implements an **Automated Market Maker**
  - First proposed by Vitalik Buterin[1]

- Development started in 2017 by Hayden Adams

- Implemented as a set of Ethereum smart contracts

- Governed by the UNI token, which itself is one of the most traded tokens

[1] https://vitalik.ca/general/2017/06/22/marketmakers.html

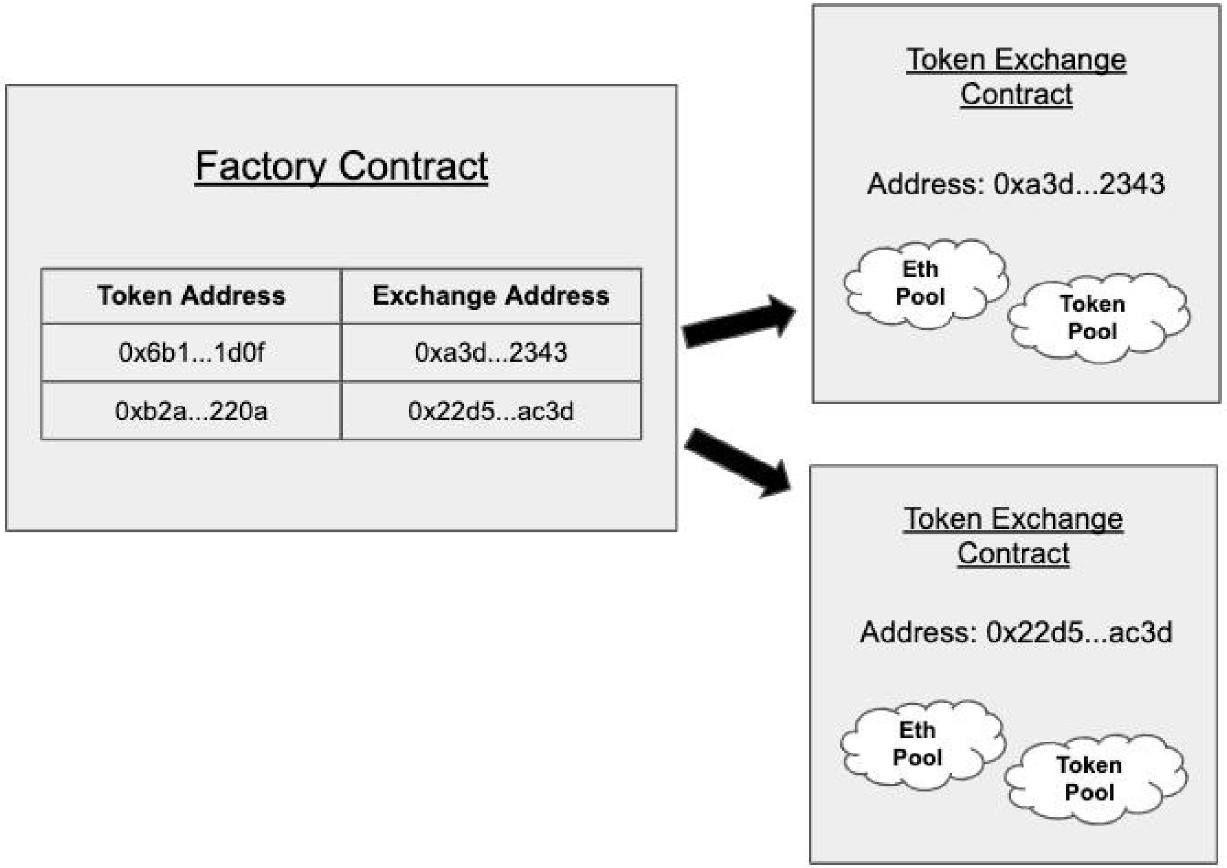
UNISWAP ARCHITECTURE

Factory contract

- Tracks mapping of all existing exchanges
- Creates new exchanges if needed

Exchange contracts

- Facilitate exchanges between Ether and a specific token
- Stores Ether and the token as needed



Factory Contract

Token Address	Exchange Address
0x6b1...1d0f	0xa3d...2343
0xb2a...220a	0x22d5...ac3d

Token Exchange Contract

Address: 0xa3d...2343

Token Exchange Contract

Address: 0x22d5...ac3d

Source: https://docs.ethhub.io/guides/graphical-guide-for-understanding-uniswap/

LIQUIDITY PROVIDERS IN UNISWAP

Problem: How does the exchange get liquidity (=money to trade)

Solution: “Crowdfund” liquidity

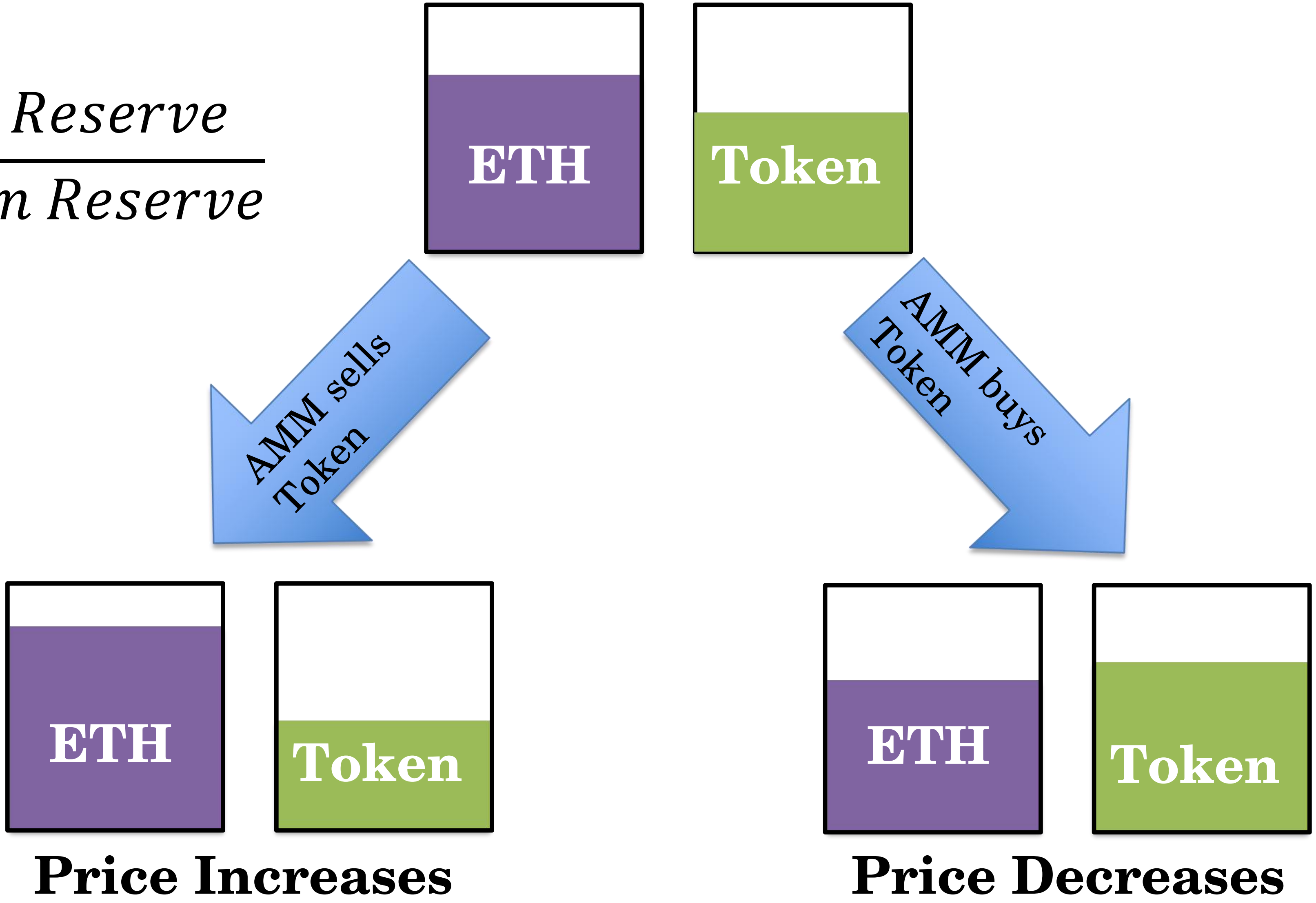
- Participants pool currency/tokens in a smart contract
  - Participants receive some number of exchange-specific tokens in return

- When people trade through the contract, a fee is charged
  - Each pool participants gets share of the fee proportional to their pool contribution

- Exchange tokens can be converted back into liquidity *at the current exchange rate*

AUTOMATED MARKET MAKERS

$$TokenPrice = \frac{Eth\ Reserve}{Token\ Reserve}$$



BREAK

SMART CONTRACT SECURITY

- Smart contracts, like any software, are prone to bugs
- Attackers can exploit bugs
  - By issuing malicious transactions
  - Through other smart contracts
- Due to the immutability of smart contracts, bugs cannot easily be fixed
- Some vulnerabilities are “the usual suspects”
  - e.g., integer overflow, missing assertions,...
- Some are blockchain-specific
  - e.g., *re-entrancy attacks*

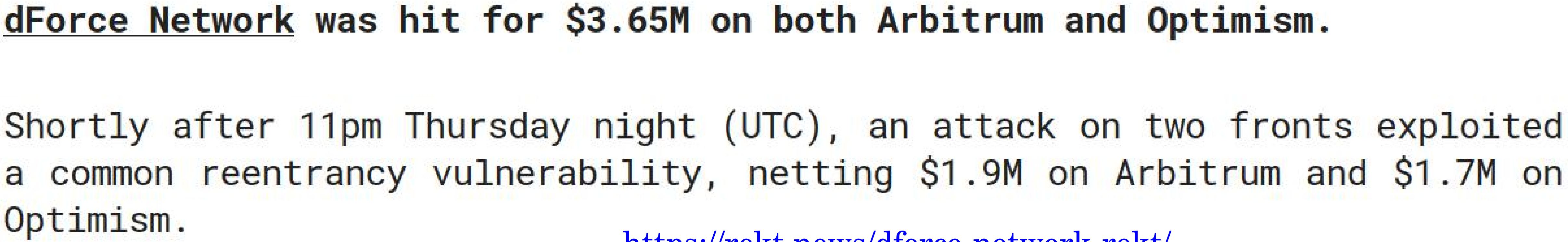
RE-ENTRANCY ATTACKS

Idea: Call back into a contract while it is still executing

Why does this work?

- Contract might have partially updated its state before calling another function
- Contract might not re-check assertions after an external call returns

First discovered 7 years ago, but still happens!



dForce Network was hit for $3.65M on both Arbitrum and Optimism.

Shortly after 11pm Thursday night (UTC), an attack on two fronts exploited a common reentrancy vulnerability, netting $1.9M on Arbitrum and $1.7M on Optimism.

https://rekt.news/dforce-network-rekt/

FALLBACK FUNCTIONS

Fallback functions are the default behavior of a smart contract

They are called when:

- Money is sent to the contract without any calldata
- The contract receives a function call, but the function does not exist

They *can perform arbitrary logic*

- Including function calls!

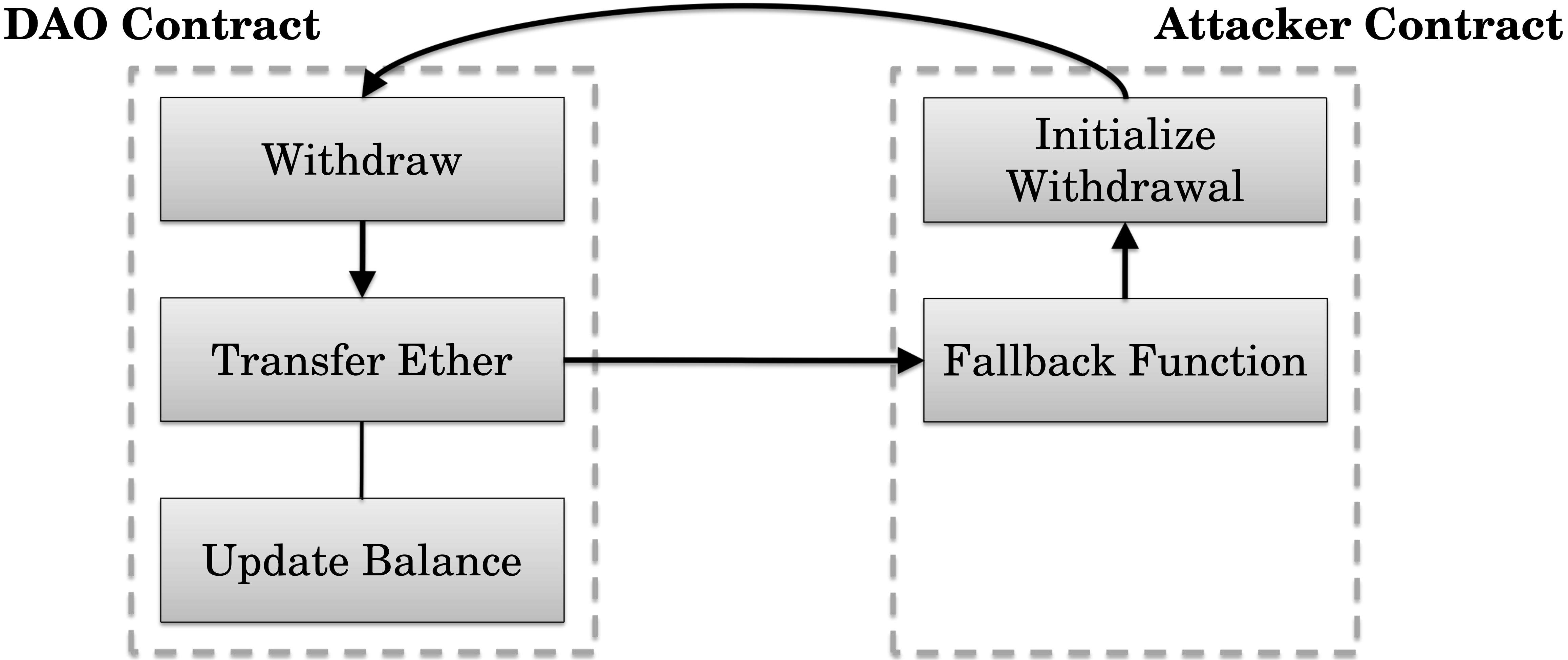
```
@external
@payable
def __default__():
    # Do something here
```

Fallback Function in Vyper

THE DAO HACK

- '"The DAO" was the first Decentralized Autonomous Organization
  - Held about $150 million in total
  - Never actually invested in anything; it got hacked before it had a chance to
- First large scale attack on a smart contract
  - Happened in 2016
  - About $50 million stolen
- At is core, just a re-entrancy attack

THE DAO HACK



Attacker loops until all money is drained

THE DAO HACK: CODE

DAO Hack: Aftermath

Rollback and Fork

- Ethereum developers reverted chain state to undo the hack
  - Shareholders of the DAO were refunded
- Caused “Ethereum Classic” to split off Ethereum

Language/Protocol Changes

- Ethereum limited gas available to the fallback function when calling send()
- Some languages, e.g., Vyper, do not allow integer over- and underflows
- *Re-entrancy locks* are now considered good practice for all smart contracts

RE-ENTRANCY LOCKS

```
@external
@nonreentrant("my_lock")
def make_a_call(_addr: address):
    # this function is protected from re-entrancy
    ...
```

Re-entrancy locks ensure mutual exclusion:

- Grab a lock while entering a function
- If lock is already held, revert(=abort) transaction

THAT'S ALL (FOR TODAY)

- Next time: Blockchain Protocols