CS 5594: BLOCKCHAIN TECHNOLOGIES

Spring 2024

THANG HOANG, PhD

PROGRAMMABLE BLOCKCHAIN

Overview

Ethereum

Smart Contracts

Ethereum Virtual Machine

Decentralized Applications

ETHEREUM

Limitation of Bitcoin

Recall: UTXO contains (hash of) public key scripts

(simple) script: indicate conditions when UTXO can be spent

Lack of Turing-completeness

script does not nearly support everything

Lack of loop instructions

Value-blindness

UTXO is all-or-nothing – it must be spent completely as a whole

Cannot provide fine-grained control over the amount that can be withdrawn

Example – Hedging contract: A and B put in $1000 worth BTC; after 30 days sends $1000 worth of BTC to A and the rest to B

Limitation of Bitcoin

Lack of state

- UTXO can be either spent or unspent
- Script does not have their own internal persistent memory
  - Impossible for multi-stage contracts or enforce global rules on assets
  - Difficult to implement complex stateful contracts

Blockchain-blindness

- scripts cannot access some blockchain data such as nonce, timestamp – all are valuable sources of randomness
- Limit applications in gambling

Ethereum

A universal, programmable blockchain

Founder: Vitalik Buterin

Russian-Canadian programmer

Image from Wikipedia

Timelines

- 2013 — Idea / White paper
- 2014 — Yellow paper
- 2015 — Frontier Genesis block
- 2016 — Official release (Homestead)
- 2017 — Byzantium hard fork
- 2018 — Overtook BTC w.r.t # active users
- 2019 — Istanbul hard fork
- 2020 — ETH 2.0 phase 0
- 2021 — ETH 2.0 phase 1

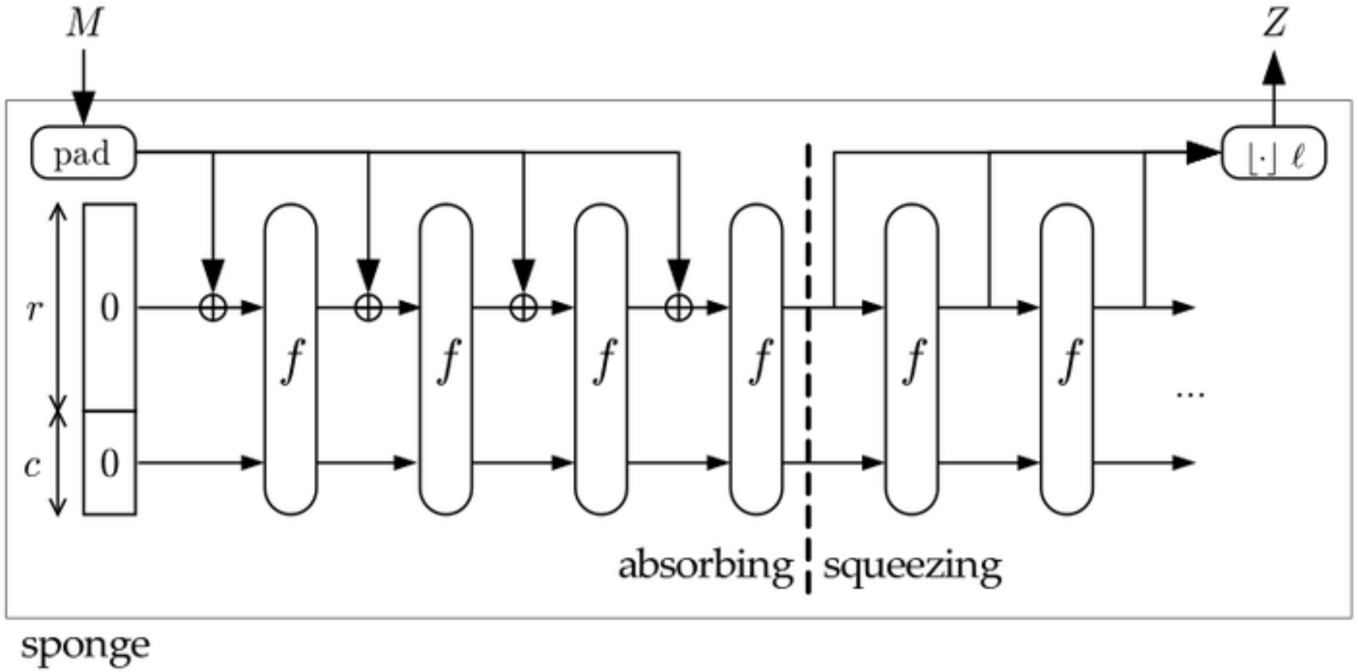
Ethereum Crypto

ECDSA for digital signatures (like Bitcoin)

Keccak-256 for hash functions (vs. SHA-256 in Bitcoin)

SHA-3

Sponge construction



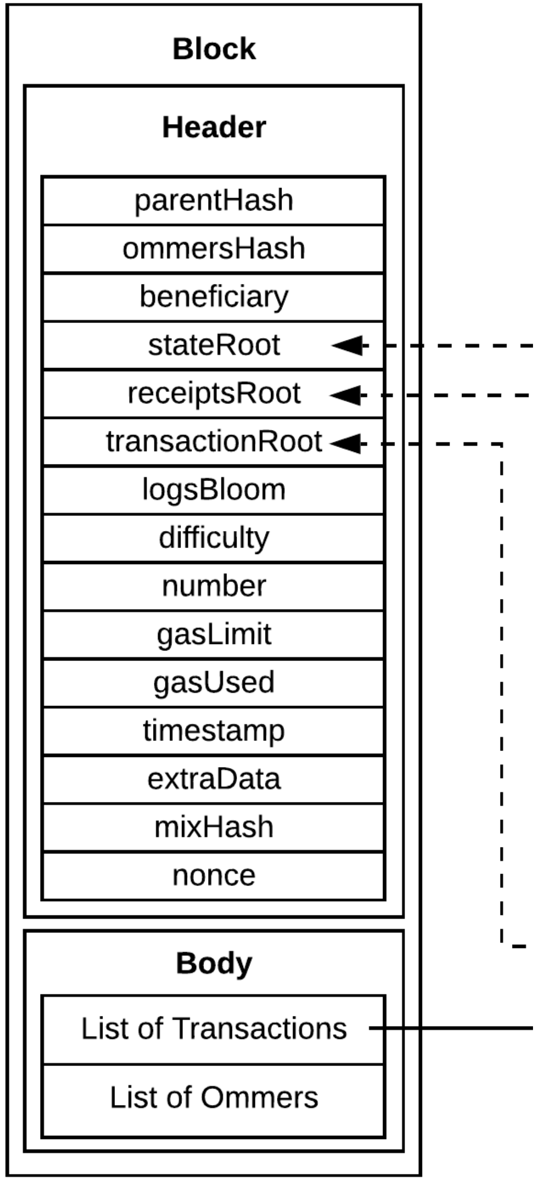
Ethereum Block

Keep track of account balance

- Not Unspent Transaction Outputs (UTXO) type like Bitcoin

An Ethereum block consists of two components

1. Block header with 15 elements
2. Block body
    1. List of Transactions
    2. List of Ommers



Ethereum Block

Block header

Consensus data: parent hash, difficulty, PoW solution, etc

Beneficiary: where TX fees will go (address)

World state root: updated world state

Merkle Patricia Tree hash of all accounts in the system

TX root: Merkle hash of all TXs included in block

TX receipt root: Merkle hash of log messages generated in block

Gas used: Tells verifier how much work to verify block

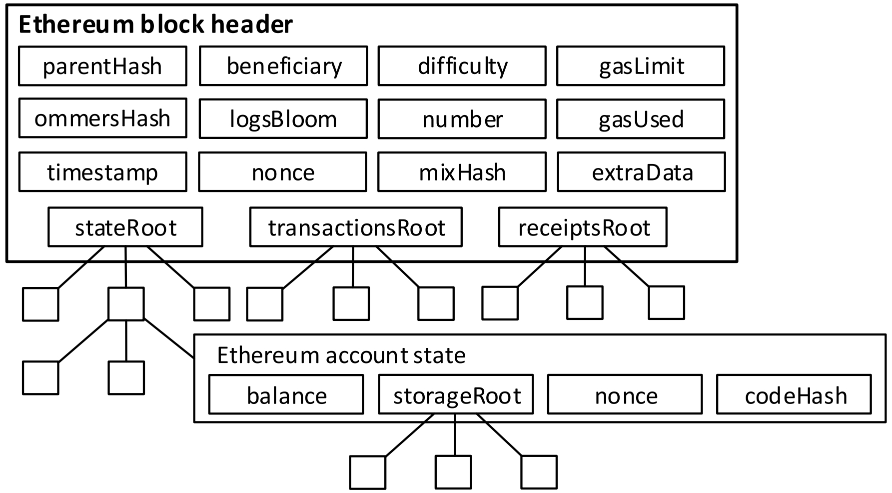


Image credit to Weber, Ingo, Qinghua Lu, An Binh Tran, Amit Deshmukh, Marek Gorski, and Markus Strazds "A platform architecture for multi-tenant blockchain-based systems." In *2019 IEEE International Conference on Software Architecture (ICSA)*, 2019.

Ethereum Block

Block header contains three Merkle trees for **Transactions**, **Receipts** and **States**

Enable light clients to conduct various types of queries

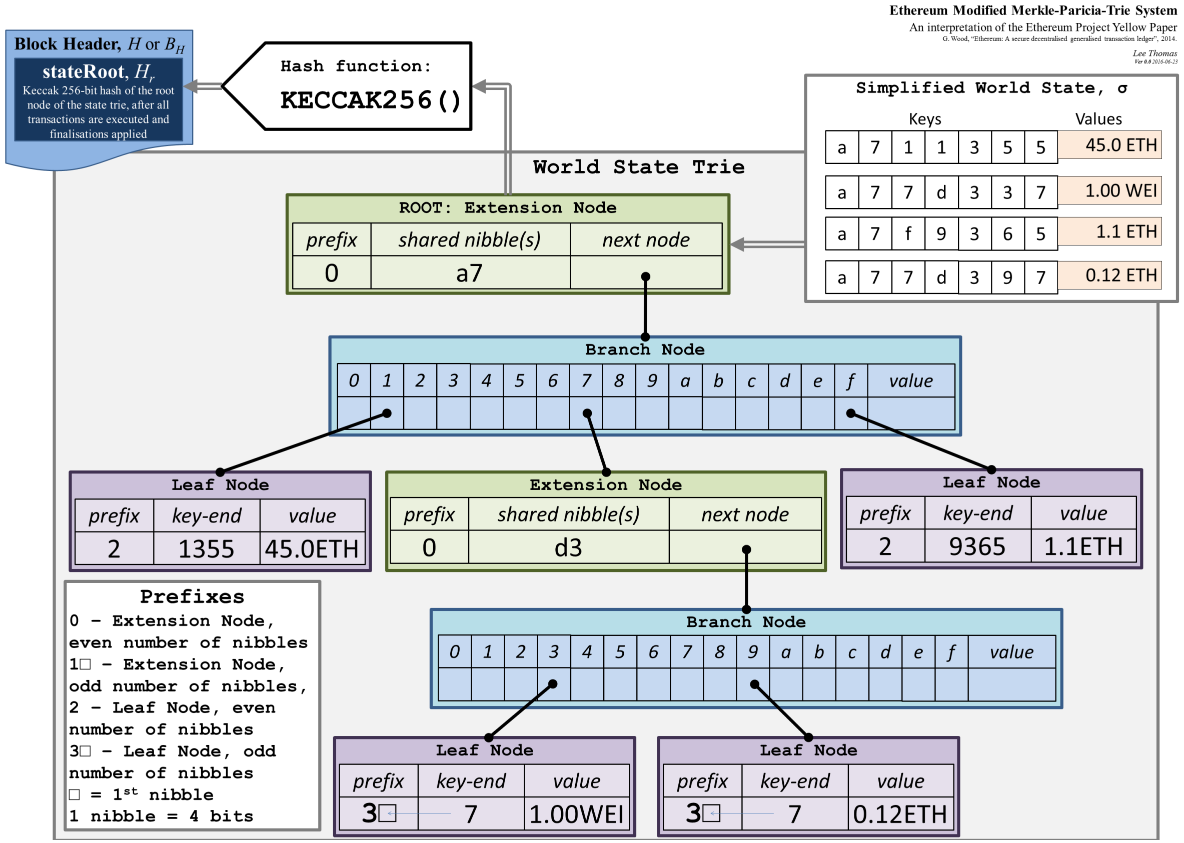
- Has this transaction been included in a particular block? (Transaction tree)
- Tell me all instances of an event of type X (eg. a crowdfunding contract reaching its goal) emitted by this address in the past Y days (Receipt tree)
- What is the current balance of my account? (State tree)
- Does this account exist? (State tree)

Modified Merkle Patricia Trie Tree

- Recap: ETH is account-based
  - Need a data structure for efficient account insert/delete/update
- Patricia: Practical Algorithm To Retrieve Information Coded In Alphanumeric

Three node types

- Extension
- Branch
- Leaf



Ethereum Block

Ommer List

- Sometimes valid block solutions don't make to the main chain
  - Due to short mining time in ETH (~15 secs) where same blocks are mined within a short interval
  - Only block mined first added to the main chain, while others left off

- Goal: Provide small reward for miners when duplicate block solutions are found

- Two valid blocks (only header, not transaction) can be included in Ommer List
  - Valid blocks: within 6^{th} generation with valid PoW solution

Ethereum Block

Ommer Rewards

Ommer headers are included in main block for 1/32 of the main block miner's reward

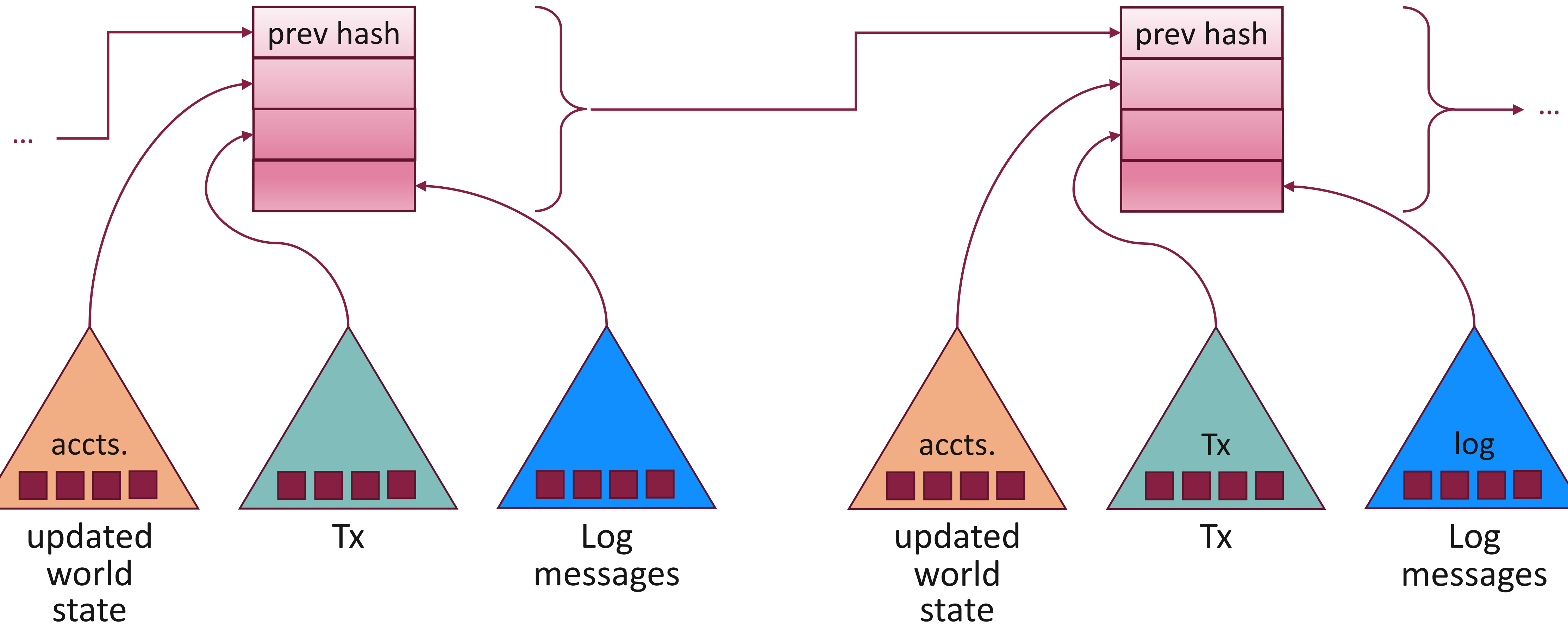
Reward equation

$$(On + (8 - Bn)) * 5 / 8$$

where On and Bn are ommer and block numbers, resp.

<u>Example:</u> (1333 + 8 - 1335) * ⅝ = 3.75 ETH

Ethereum Blockchain (Abstract)



Ethereum Denominations

Wei

Named after Wei Dai (author of b-money)

1/1,000,000,000,000,000,000 (quintillion)

Szabo

Named after Nick Szabo (author of Bit-Gold)

Finney

Named after Hal Finney

First bitcoin user after Nakamoto

Multiplier	**Name**
10	Wei
10^{12}	Szabo
10^{15}	Finney
10^{18}	Ether

Ethereum vs Bitcoin

	Bitcoin	Ethereum
Specification	Bitcoin Core client	Ethereum yellow paper
Consensus	SHA256 PoW	Ethash PoW / PoS
Contract Language	Script	EVM bytecode
Block interval	10 mins	10-20 secs
Block size limit	1 MB	1,500,000 Gas
Difficulty adjustment	After 2016 blocks	After every block
Currency supply	Fixed (21 million in total)	Varied (101 million as of 2018)
Currency units	1 BTC = 10^8 satoshi	1 ETH = 10^{18} Wei
Mining Reward	12 BTC (halves every 4 years)	5 ETH (main) + 2/32 (ommer)
Smart contract	Not supported	Supported

Gencer, A. E., Basu, S., Eyal, I., Van Renesse, R., & Sirer, E. G. (2018, February). Decentralization in bitcoin and ethereum networks. In International Conference on Financial Cryptography and Data Security (pp. 439-457). Springer, Berlin, Heidelberg.

Ethereum Nodes

P2P Network

Two types of nodes (like bitcoin)

- Full nodes: store a copy of the entire blockchain
  - Validate all transactions and new blocks
- Light nodes: store only block headers
  - Trust and request everything else from full nodes
  - Can only verify validity of data against state roots in block headers
  - Don’t execute transactions, used primarily for balance validation

Implemented in a variety of languages (Go, Rust, etc.)

Ethereum Accounts

Public/private key pair

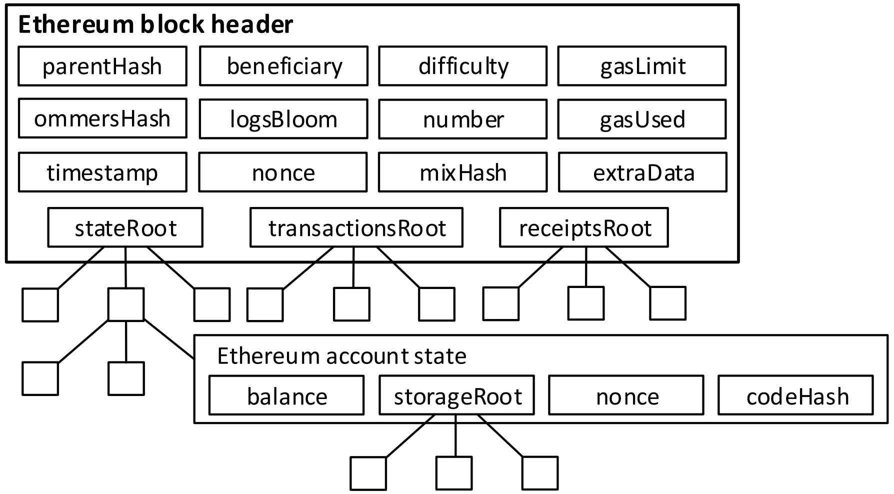
Two types of accounts

- External Owned Accounts (EOA) – most popular

  Controlled by anyone with private keys

- Contract Accounts

  Controlled by code (smart contracts)



Account info stored in World State nodes

Nonce: List of number of TX's from account

CodeHash: Hash of associated code (used in contract accounts) .

Computer program for a smart contract (hash of an empty string for EOAs)

StorageRoot: Merkle-Patricia trie tree root of account storage contents

Balance: Account balance

Ethereum Accounts

EOA Account Example

Private Key: 0x2dcef1bfb03d6a950f91c573616cdd778d9581690db1cc43141f7cca06fd08ee

64 hex characters

66 characters in total (with 0x appended). Case insensitive. Same derivation through ECDSA (like Bitcoin)

Address: 0xA6fA5e50da698F6E4128994a4c1ED345E98Df50

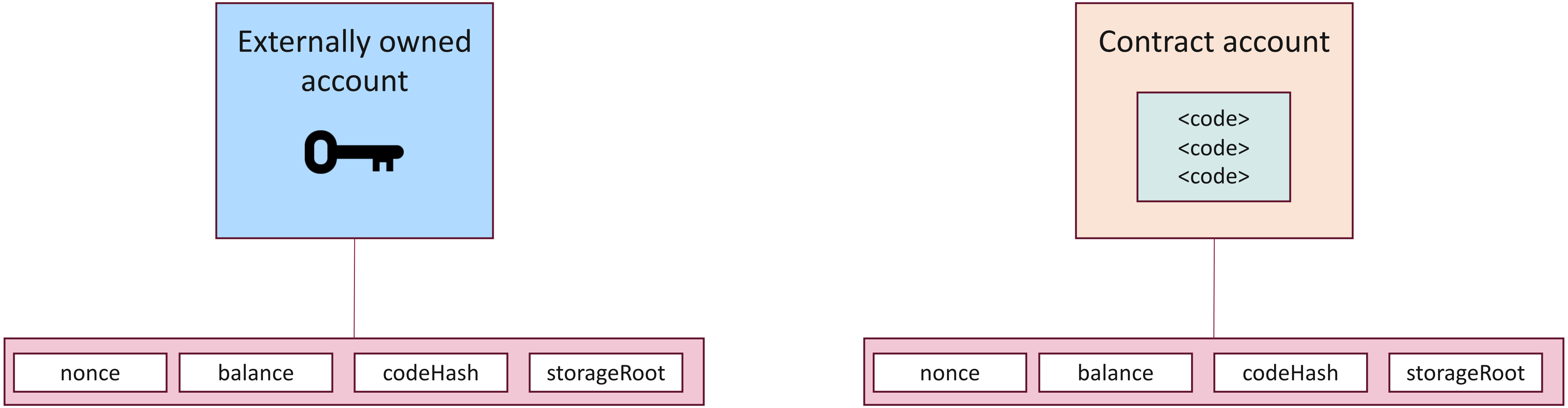
Last 40 characters (20 bytes) of the Keccak-256 hash of the ECDSA public key.

42 characters in total (append 0x to front for hexadecimal representation)

Ethereum Accounts

Contract Accounts

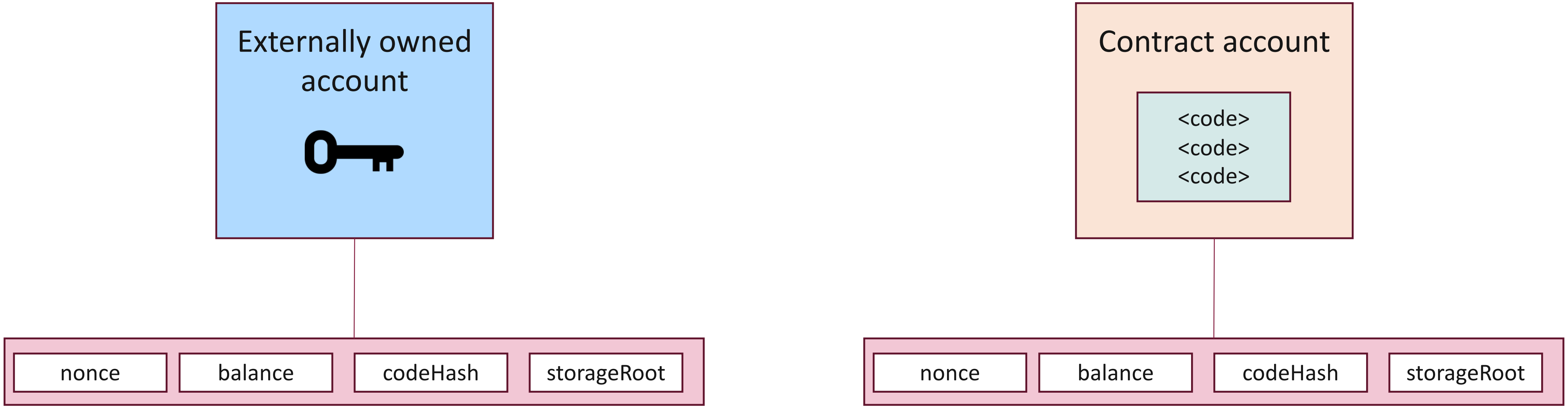
- Store and execute code – incur a fee/gas
- Code execution triggered by transactions or messages from other contracts
  - Perform operations with arbitrary complexity (Turing completeness)
  - Manipulate its own persistent storage (i.e., have its own permanent state)
  - Can call other contracts



Ethereum Accounts

Contract Accounts

- All actions is set in motion by <u>transactions fired from EOAs</u>
- Code in contract accounts is executed as instructed by input parameters included in the transaction
- Code executed by EVM running on Ethereum nodes

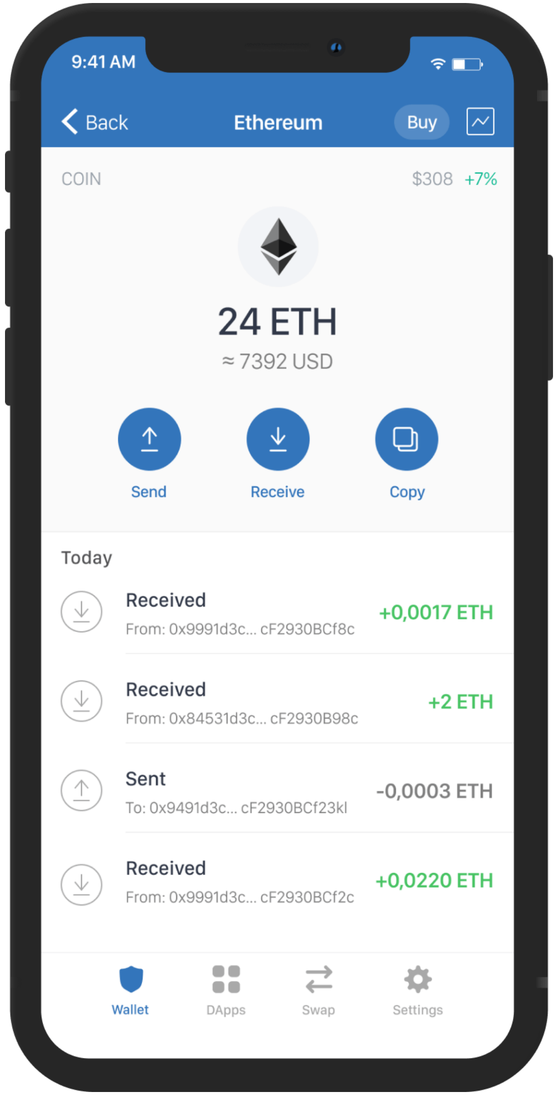


Ethereum Wallet

An app to interact with Ethereum accounts

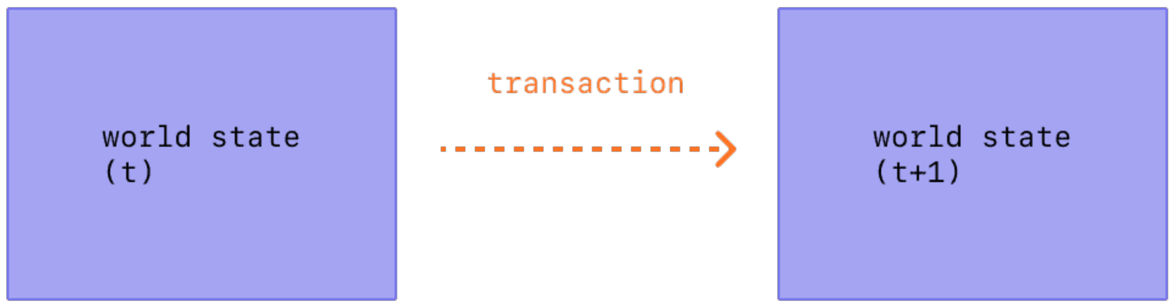
Manage a set of one or more external accounts

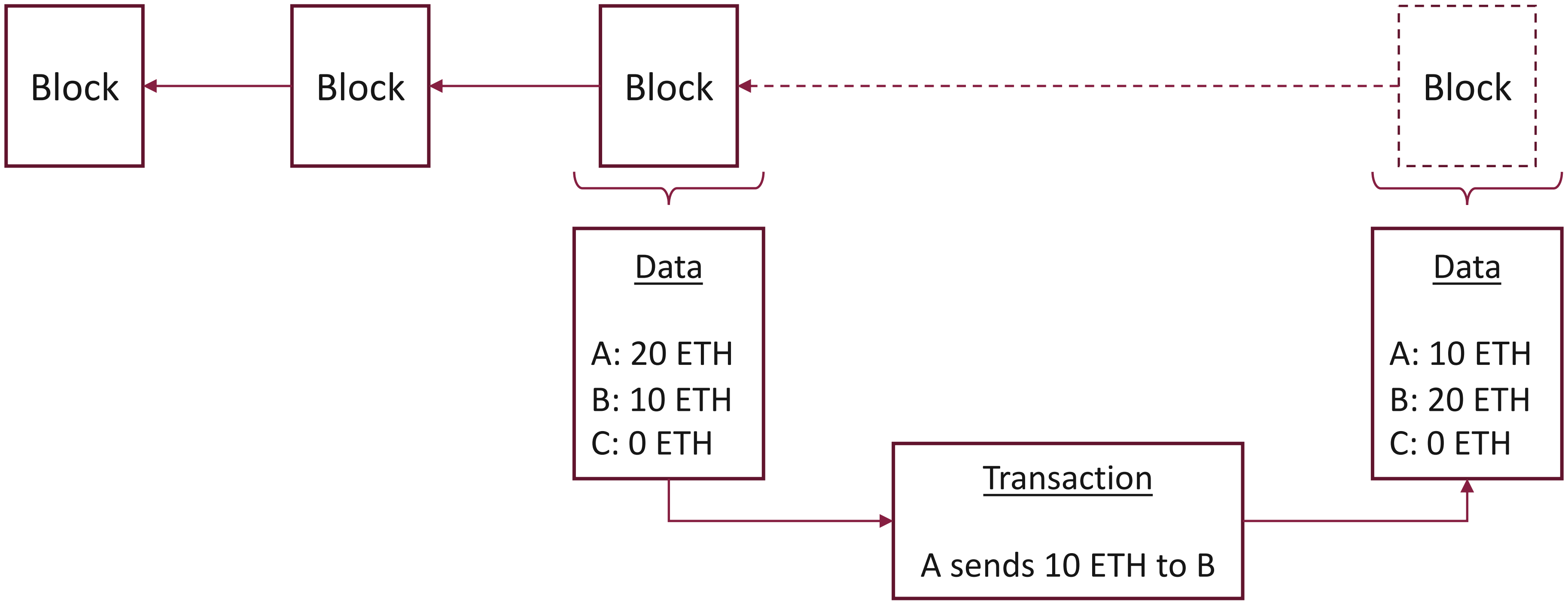
Used to store and transfer Ether



Ethereum Transaction

Ethereum can be considered as a transaction-based state machine



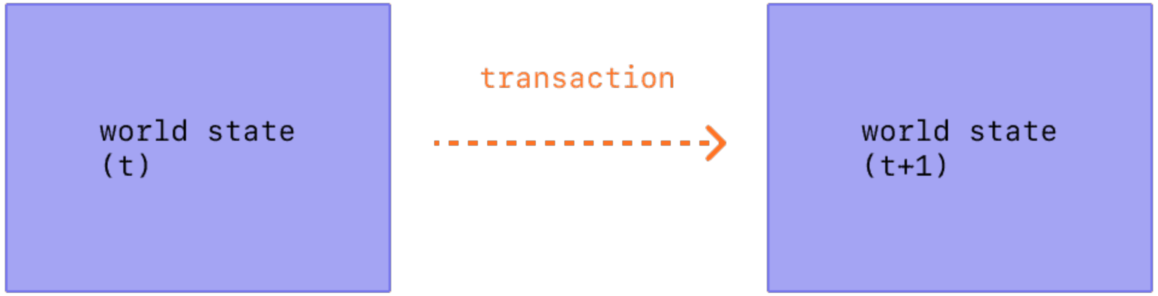


Ethereum Transaction

A request (initiated by EOA) to modify the state of the blockchain

- Can run code (contracts) to change global world state

Cryptographically signed by originating EOA



Transaction Types

- Send value from one account to another account
- Create smart contract
- Execute smart contract code

Ethereum Transaction

A submitted transaction includes the following information

Recipient: Receiving address

If EOA, will transfer value. If contract account, will execute contract code

Signature: Sender identifier

Value: Amount of ETH to transfer from sender to recipient (in WEI)

Data: optional field to include arbitrary data

gasLimit: Maximum amount of gas units consumed by transaction

Units of gas represent computational steps

gasPrice: The fee sender pays per unit of gas


```
{
from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
gasLimit: "21000",
gasPrice: "200",
nonce: "0",
value: "10000000000",
}
```

SMART CONTRACT

Smart Contracts

A collection of executable code (functions) and data (states) residing at a specific address on Ethereum blockchain

Live in Ethereum-specific binary format called EVM bytecode

Turing Completeness

Function like an external account

- Hold funds
- Can interact with other accounts and smart contracts via messages
- Contain code

Triggered by transactions

Smart Contract Programming

Solidity (javascript based)

- Originally proposed by Gavin Wood
- Object-oriented PL
- Most popular

Serpent (python based)

LLL (lisp based)

Mutan (Go based)

- Deprecated

Viper, Lisk, Chain, etc

Ethereum Smart Contract Programming

Solidity

JavaScript syntax

Support writing smart contracts and EVM bytecode compile

https://docs.soliditylang.org/en/v0.8.2/ (documentation)

Serpent

Python syntax

Support writing smart contracts and EVM bytecode compilation

Clean and simple clean

LLL as compiler

https://github.com/ethereum/serpent

Smart Contract Examples

Simple Storage

Store a single number accessible by anyone in the world

Anyone can call set again to overwrite number

The number will still be stored in the history of the blockchain

```
pragma solidity ^0.4.0;

contract SimpleStorage {

    uint storedData;

    function set(uint x) public {

        storedData = x;

    }

    function get() public view returns (uint) {

        return storedData;

    }

}
```

Smart Contract Examples

Subcurrency

Generate coins out of thin air, but can be done only by the one who created contract

Anyone can send coins to each other without registering username & password

```
pragma solidity ^0.4.21;
contract Coin {
        address public minter;
        mapping (address => uint) public balances;
        event Sent(address from, address to, uint amount);
        function Coin() public {
                minter = msg.sender;
        }
        function mint(address receiver, uint amount) public {
                if (msg.sender != minter) return;
                balances[receiver] += amount;
        }
        function send(address receiver, uint amount) public {
                if (balances[msg.sender] < amount) return;
                balances[msg.sender] -= amount;
                balances[receiver] += amount;
                emit Sent(msg.sender, receiver, amount);
        }
}
```

Keyword "public" makes those variables readable from outside

Events allow light clients to react on changes efficiently

This is the constructor whose code is run only when the contract is created

ETHEREUM VIRTUAL MACHINE

Most slides derived from the original ones by Takenobu T.

Ethereum Virtual Machine

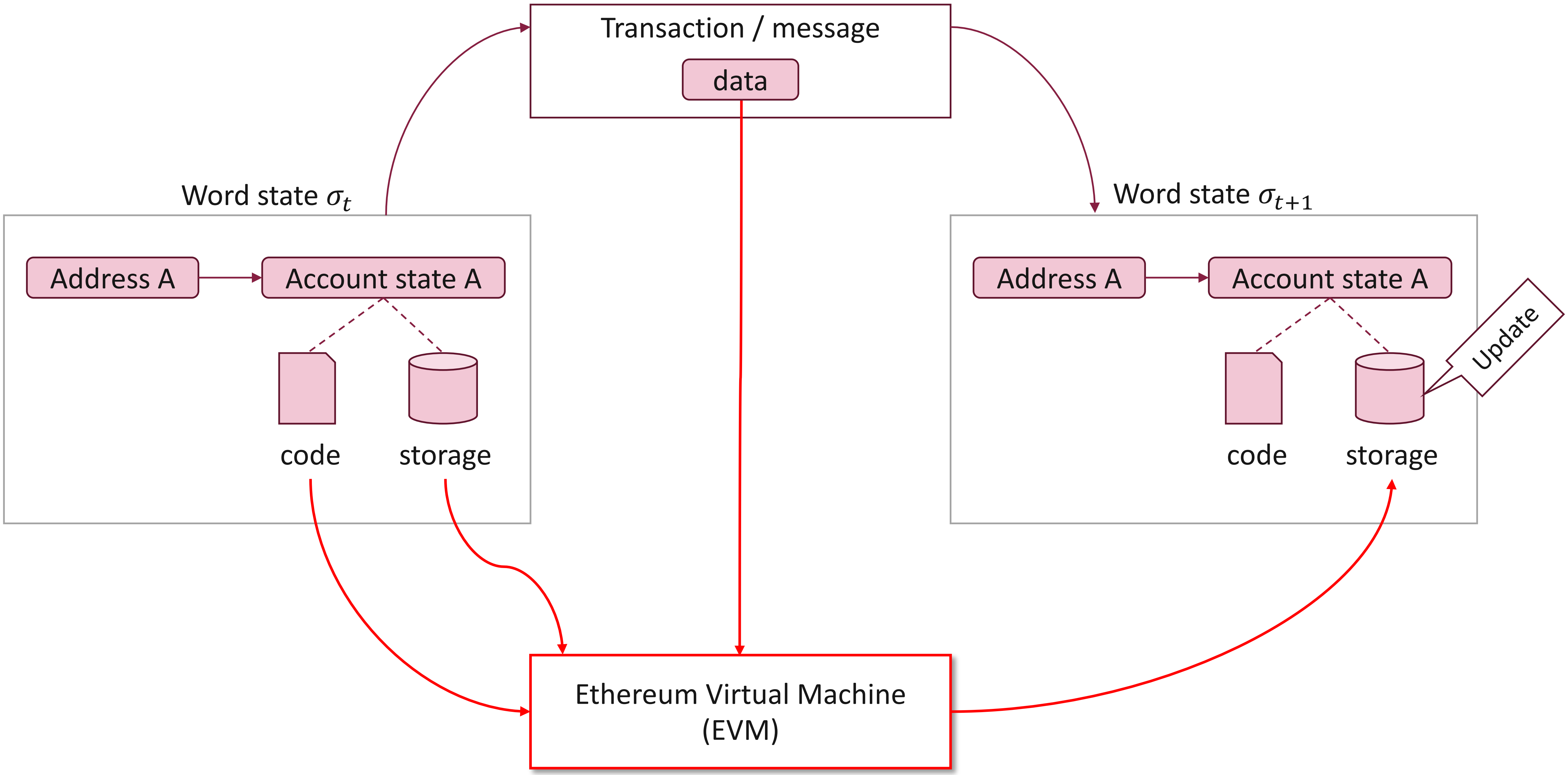
Smart contracts executed by nodes running Ethereum Virtual Machine (EVM)

Every node contains a virtual machine (similar to Java)

- Compile code from high-level language to bytecode
- Execute smart contract code and broadcast state

Every full-node on the blockchain processes every transaction and stores the entire state

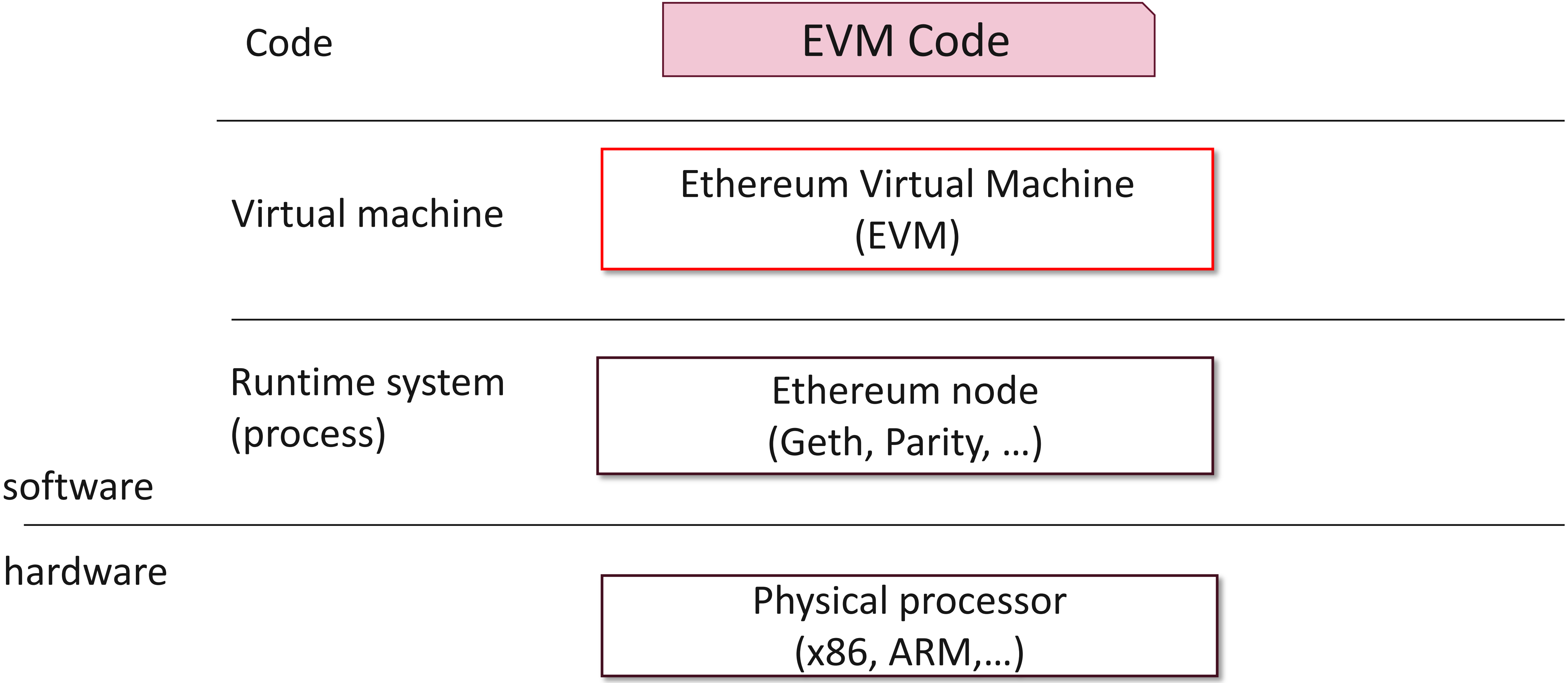
Ethereum Virtual Machine



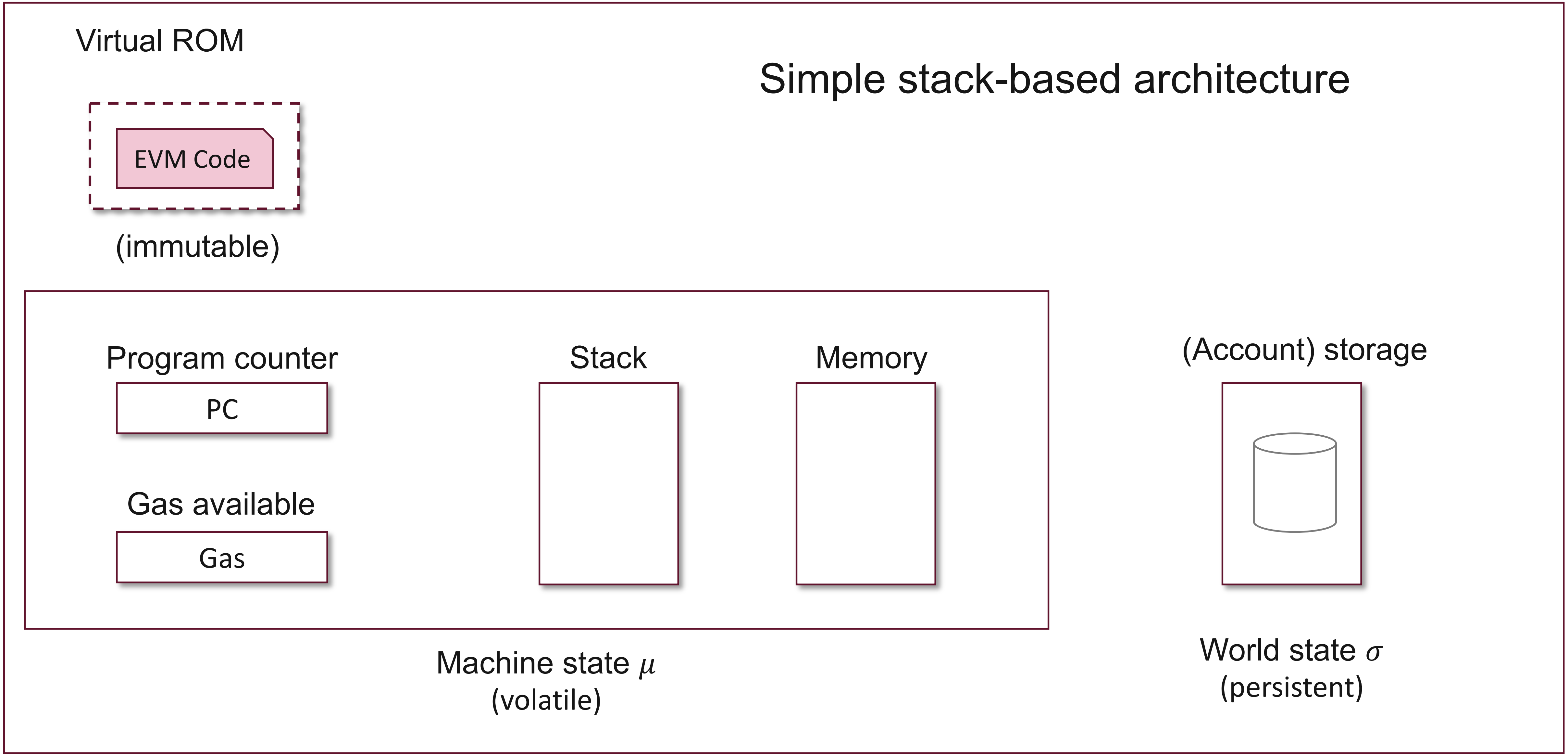
Ethereum Virtual Machine

EVM code is executed on EVM

EVM is the runtime environment for smart contracts in Ethereum



EVM Architecture



Machine space of EVM

Registers

Stack

Memory

(Account) storage

stack memory

volatile memory

Persistent memory

256 bits x 1024 elements

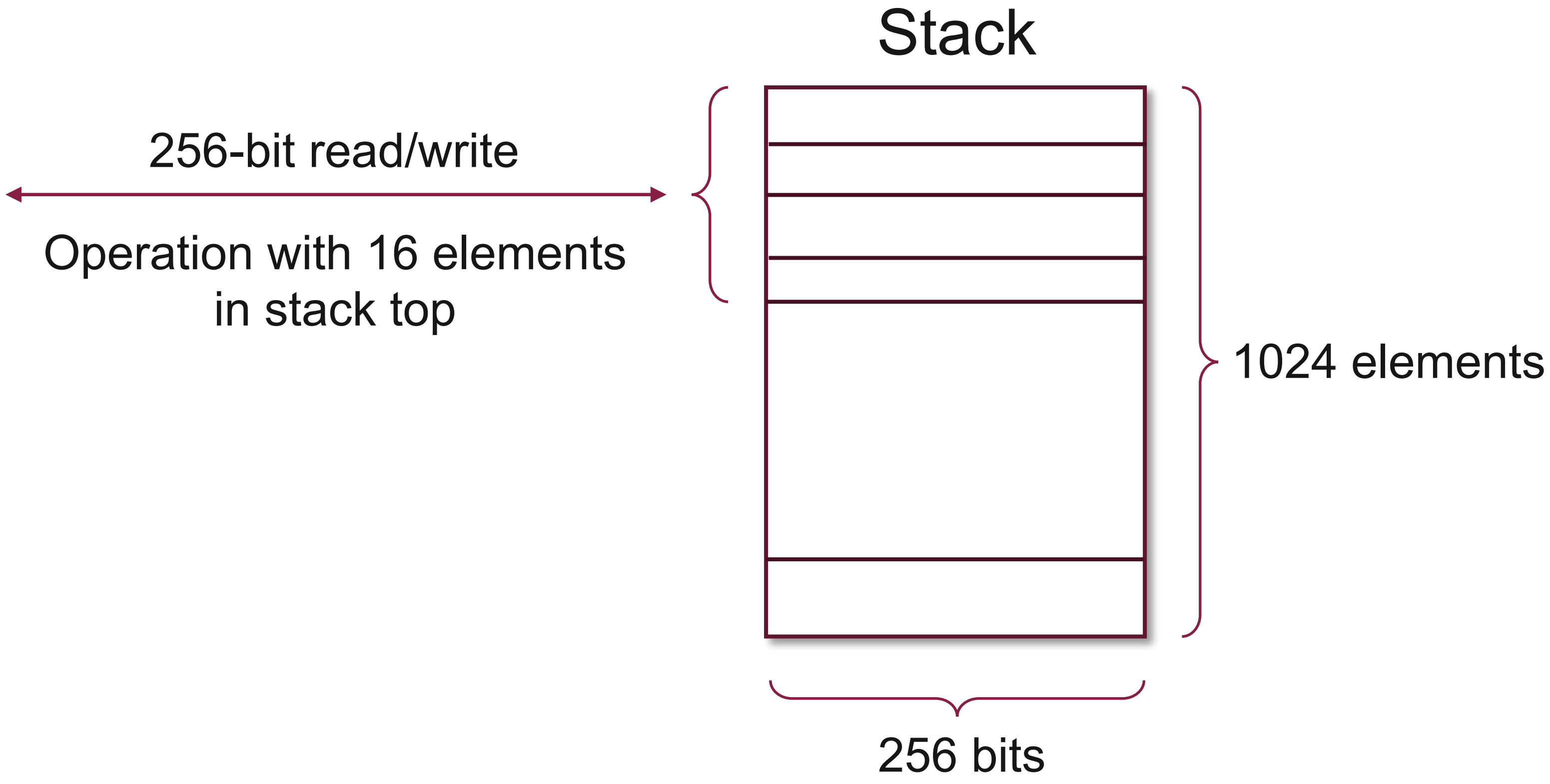
byte addressing
linear memory

256 bits – 256 bits
key-value store

Machine space of EVM

All operations performed on stack
Access with stack instructions such as PUSH/POP/COPY/SWAP/JUMP

Max stack depth = 1024
Program aborts if stack size exceeded; miner keeps gas



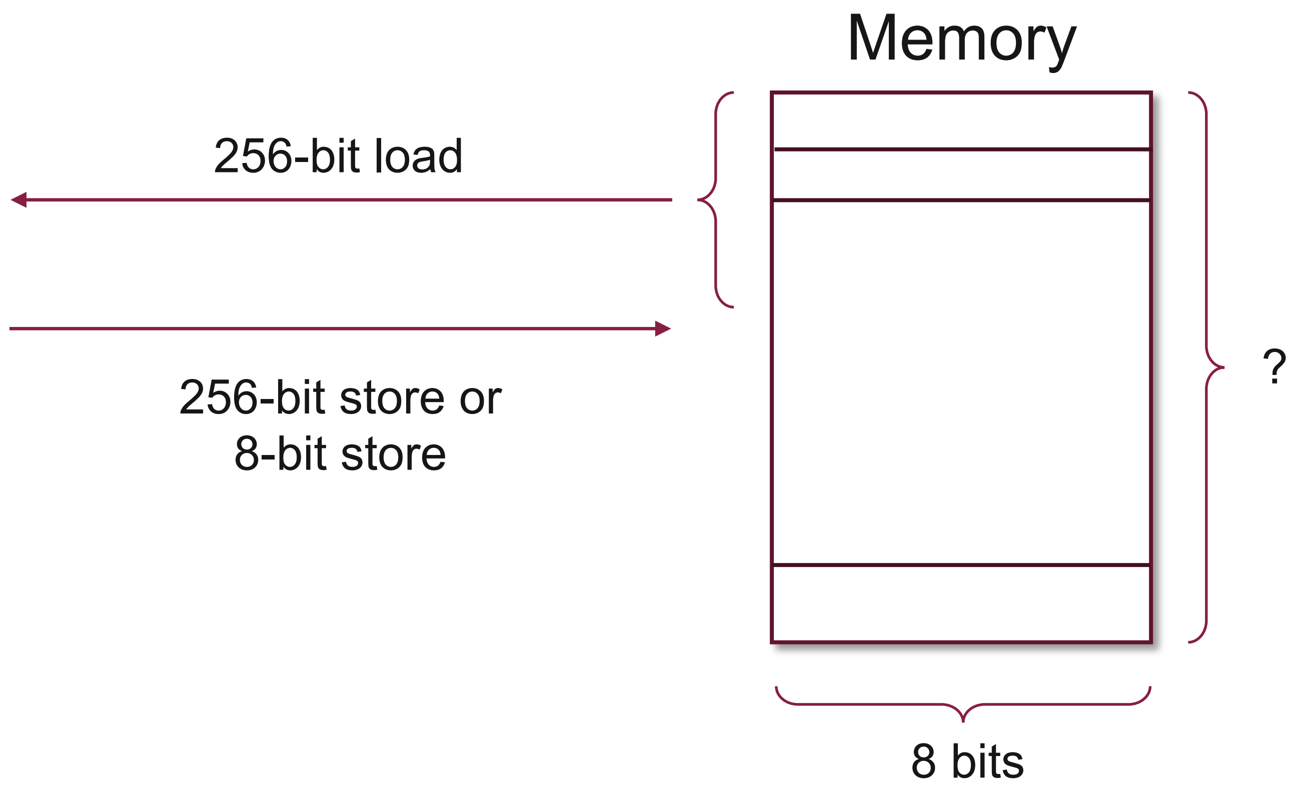
EVM Memory

Linear memory

Byte-level access

Access with MSTORE/MSTORE8/MLOAD instructions

All locations in memory are well-defined initially as zero



EVM Account Storage

Storage is key-value store mapping 256-bit words to 256-bit words

Access with SSTORE/SLOAD instructions

All locations in storage are well-defined initially as zero

256-bit load / store

(Account) storage

Key 1	Value 1
Key 2	Value 2
…	…
Key n	Value n

256 bits — 256 bits

EVM Code

Assembly view

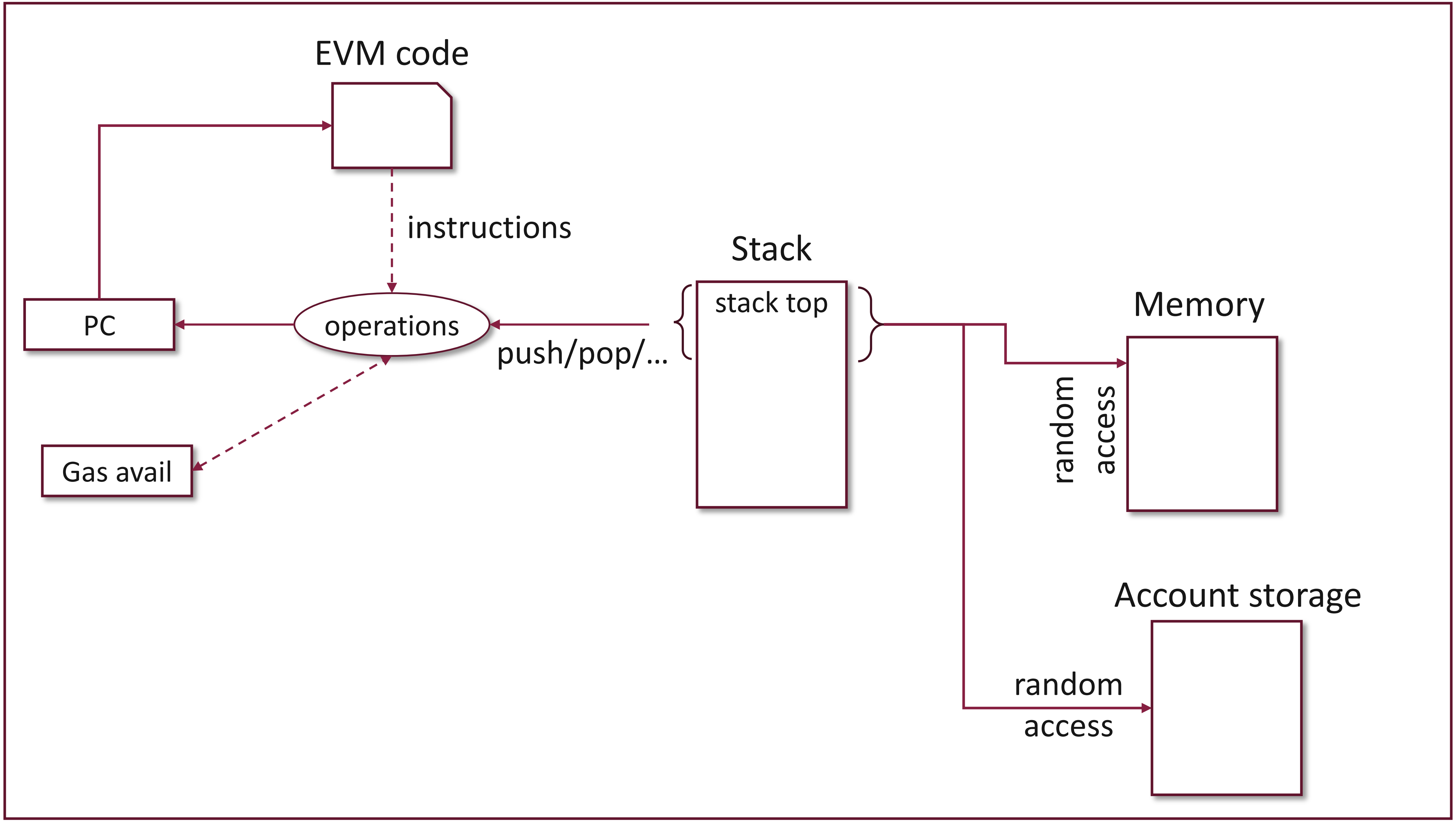
PUSH1 e0
PUSH1 02
EXP
PUSH1 00
CALLDATALOAD
…

Bytecode view

0x60e060020a600035

EVM Code is the bytecode that the EVM can natively execute

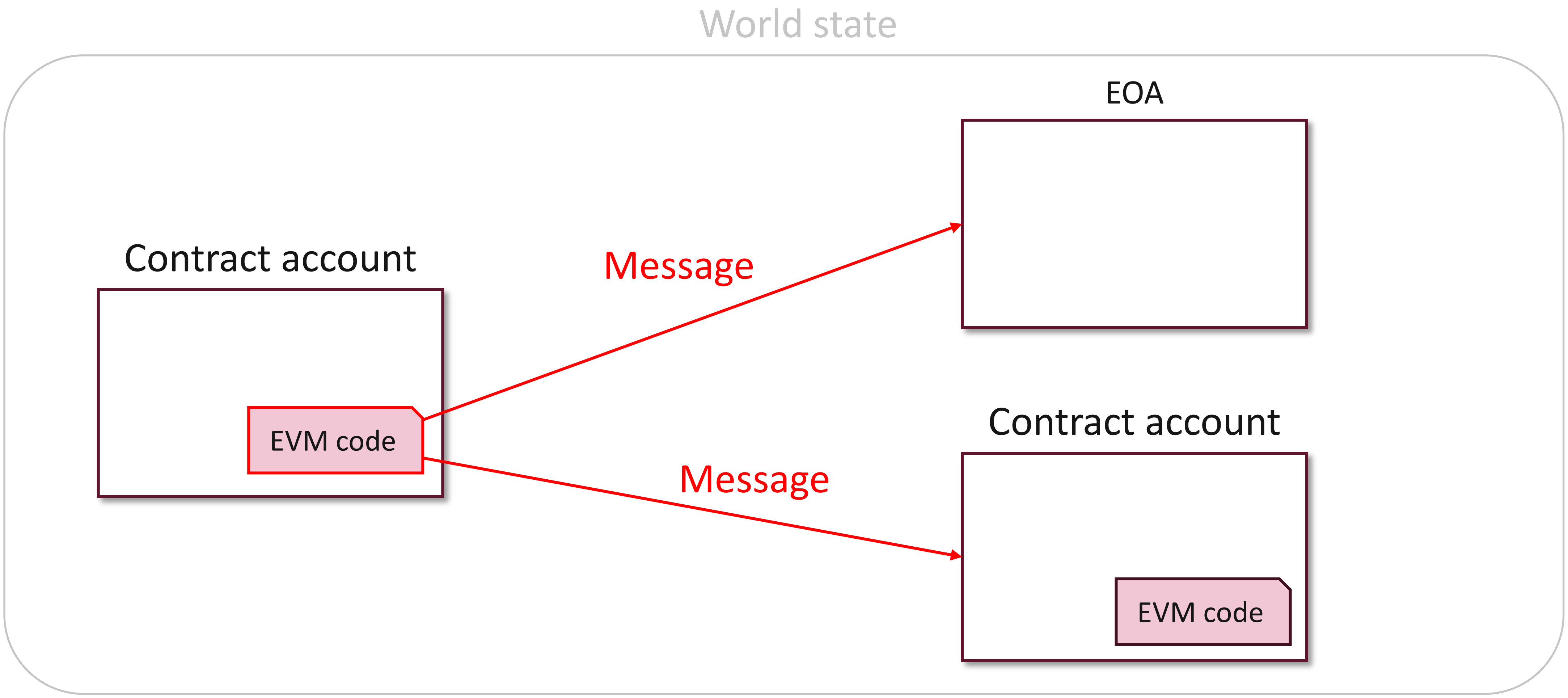
EVM Execution model



EVM Message Call

EVM can send a message to other account

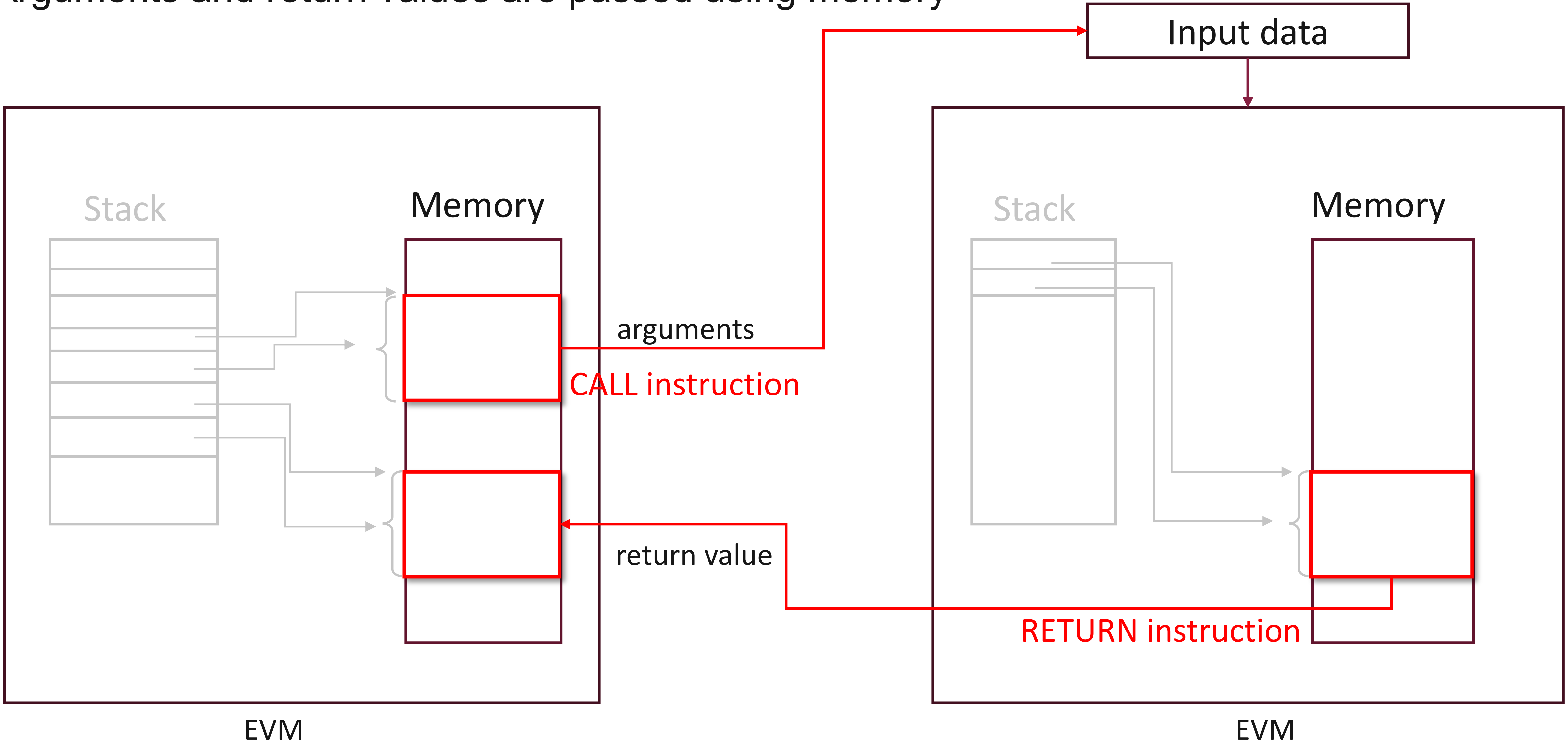
The depth of message call is limited to less than 1024 levels



EVM Message Call Instructions

Message call triggered by CALL instruction

Arguments and return values are passed using memory



Ethereum Gas

All programmable computation in Ethereum subject to fee (gas)

Gas Price: Current market price of a unit of Gas (in Wei)

https://ethgasstation.info/ for price

Set before a transaction by user

Gas Limit: maximum amount of Gas to use

All blocks have a Gas Limit

$$GasCost = gasLimit \times gasPrice$$

Help to regulate load on network

Ethereum Gas

Why Need Gas?

Halting problem (infinite loop)

Problem: Cannot tell whether a program will run forever from compiled code

Solution: Charge fee per computation step to limit infinite loop and stop flawed code from executing

Gas (TX fees) prevents submitting Tx that runs for many steps

Essentially a measure of how much user is willing to spend on a transaction even if buggy

Every EVM instruction costs gas

Every Tx specifies an estimate of gas to be spent

gasPrice: conversion: gas → Wei

gasLimit: max gas for Tx

Ethereum Gas Deduction

Tx specifies **gasPrice**: conversion gas → Wei

gasLimit: max gas for Tx

(1) if **gasLimit x gasPrice** > msg.sender.balance: abort

(2) deduct **gasLimit x gasPrice** from msg.sender.balance

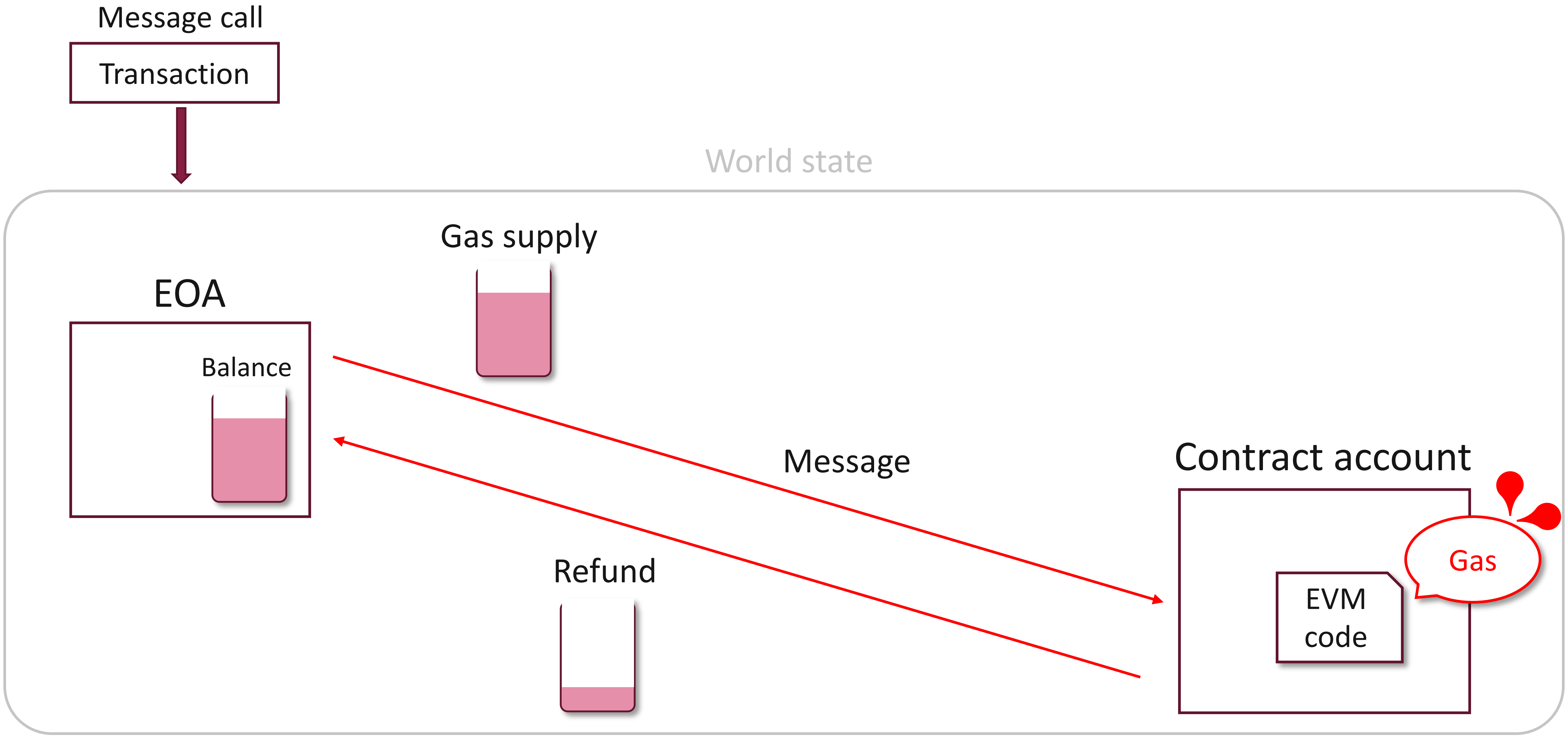
(3) set Gas = gasLimit

(4) execute Rx: deduct gas from Gas for each instruction

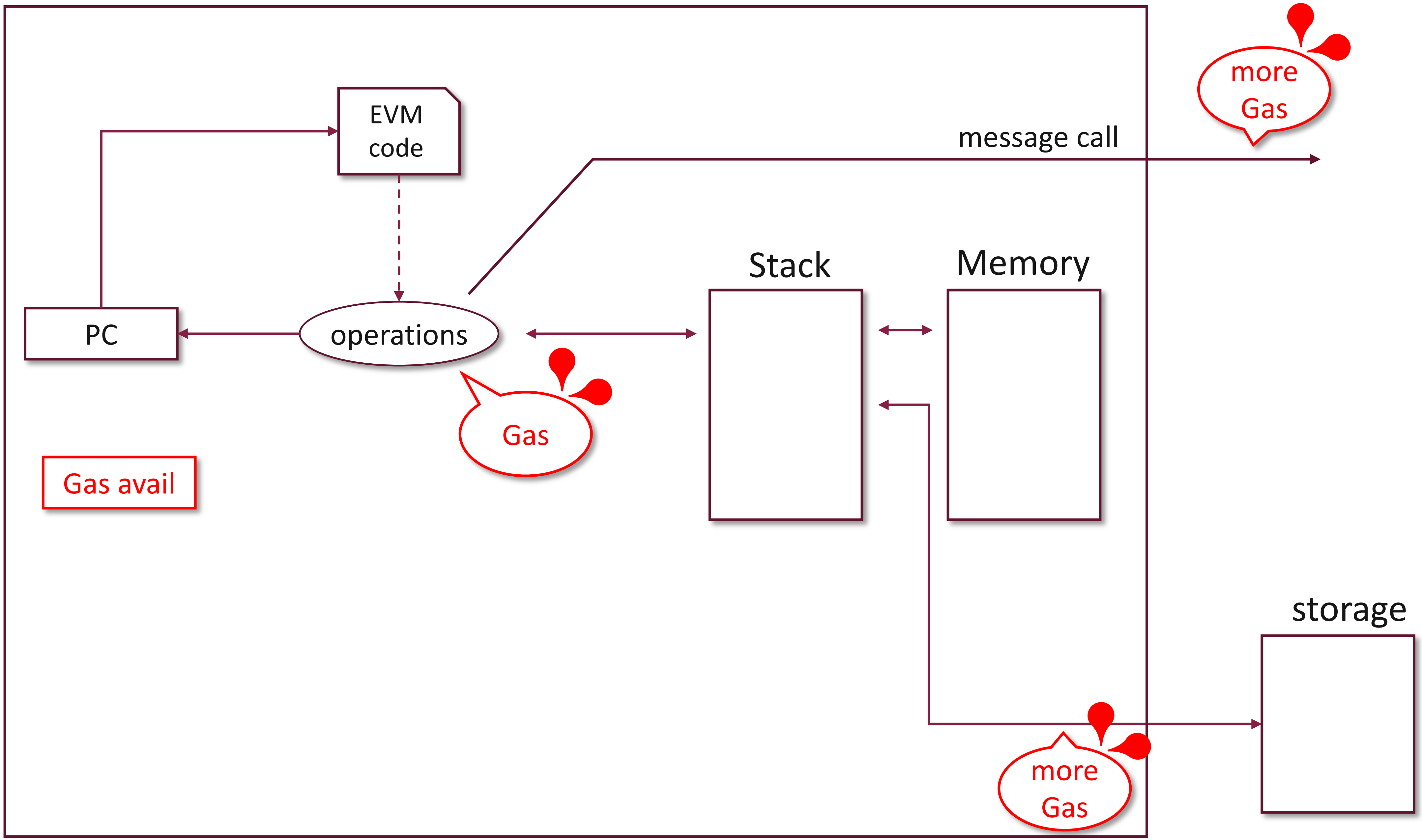
if (Gas < 0): abort, miner keeps **gasLimit × gasPrice**

(5) Refund **Gas x gasPrice** to msg.sender.balance

Ethereum Gas Deduction



Ethereum Gas Deduction



Ethereum Gas Prices: Example

SSTORE addr (32 bytes), **value** (32 bytes)

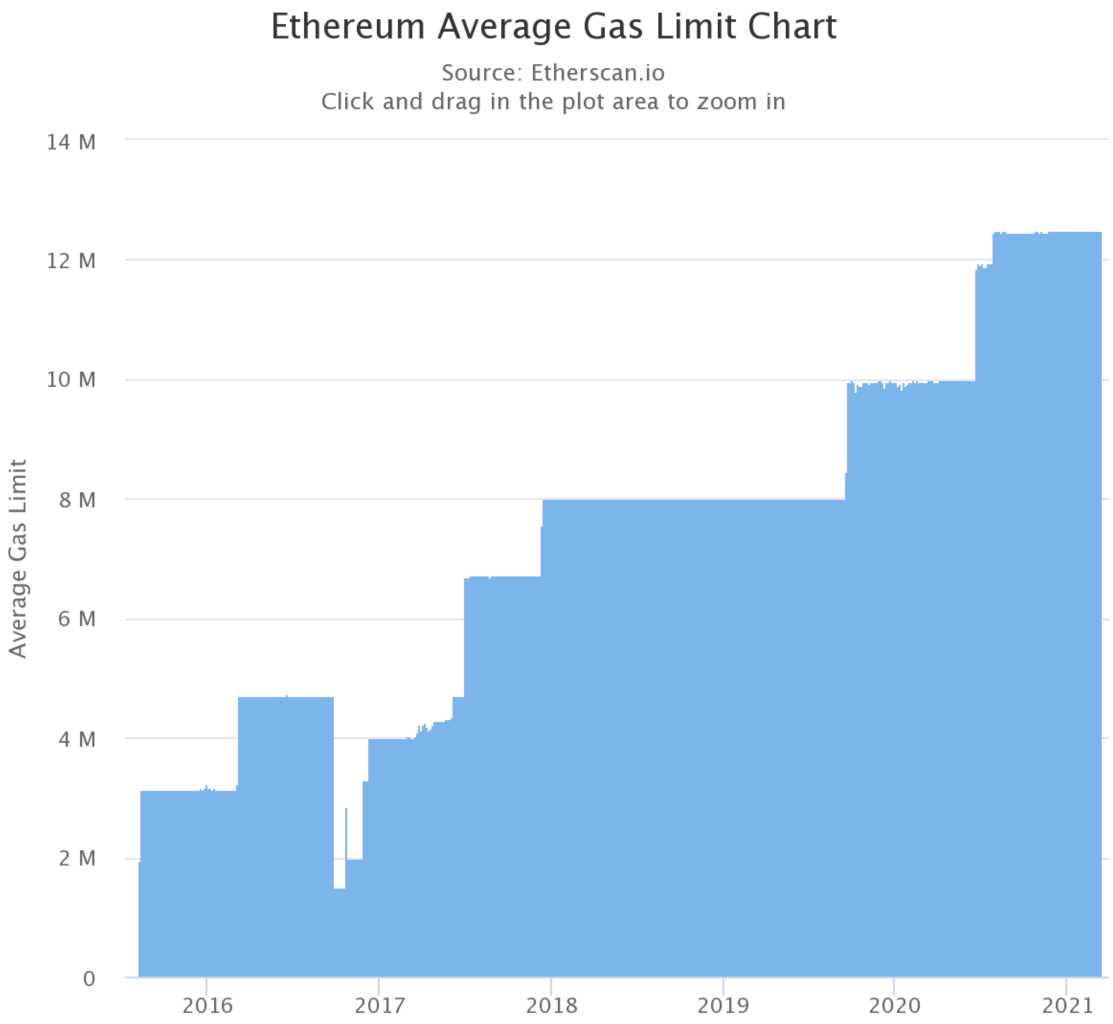
zero → non-zero:	20,000 gas
non-zero → non-zero:	5,000 gas
non-zero → zero:	15,000 gas refund

SUICIDE: kill current contract. 24,000 gas refund

Refund is given for reducing size of blockchain state

Current Ethereum Gas Limit

GasLimit is increasing over time ⇒ each Tx takes more instructions to execute



Ethereum Mining

Ethash Proof of Work

- Keccak-256 (SHA3 variant)
- Memory-hard computation
- Memory-easy validation
- Cannot exploit ASIC

Mining similar to Bitcoin

```
nonce = rand()
while (SHA3(block,nonce) * difficult > threshold
      nonce++
return nonce
```

Ethereum Mining

Difficulty adjustment

After every block (vs. after 2016 blocks in bitcoin)

```
block_diff    =  parent_diff + parent_diff / 2048 *
                 max(1 - (block_timestamp - parent_timestamp) / 10, -99) +
                 int(2**((block.number / 100000) - 2))
```

If the difference (`block_timestamp - parent_timestamp`) is

- < 10 secs, adjust upwards by `parent_diff / 2048 * 1`
- 10 - 19 secs, unchanged
- >= 20 seconds, adjust downwards from `parent_diff/ 2048 * -1` to `parent_diff / 2048 * -99`

Ethereum Mining

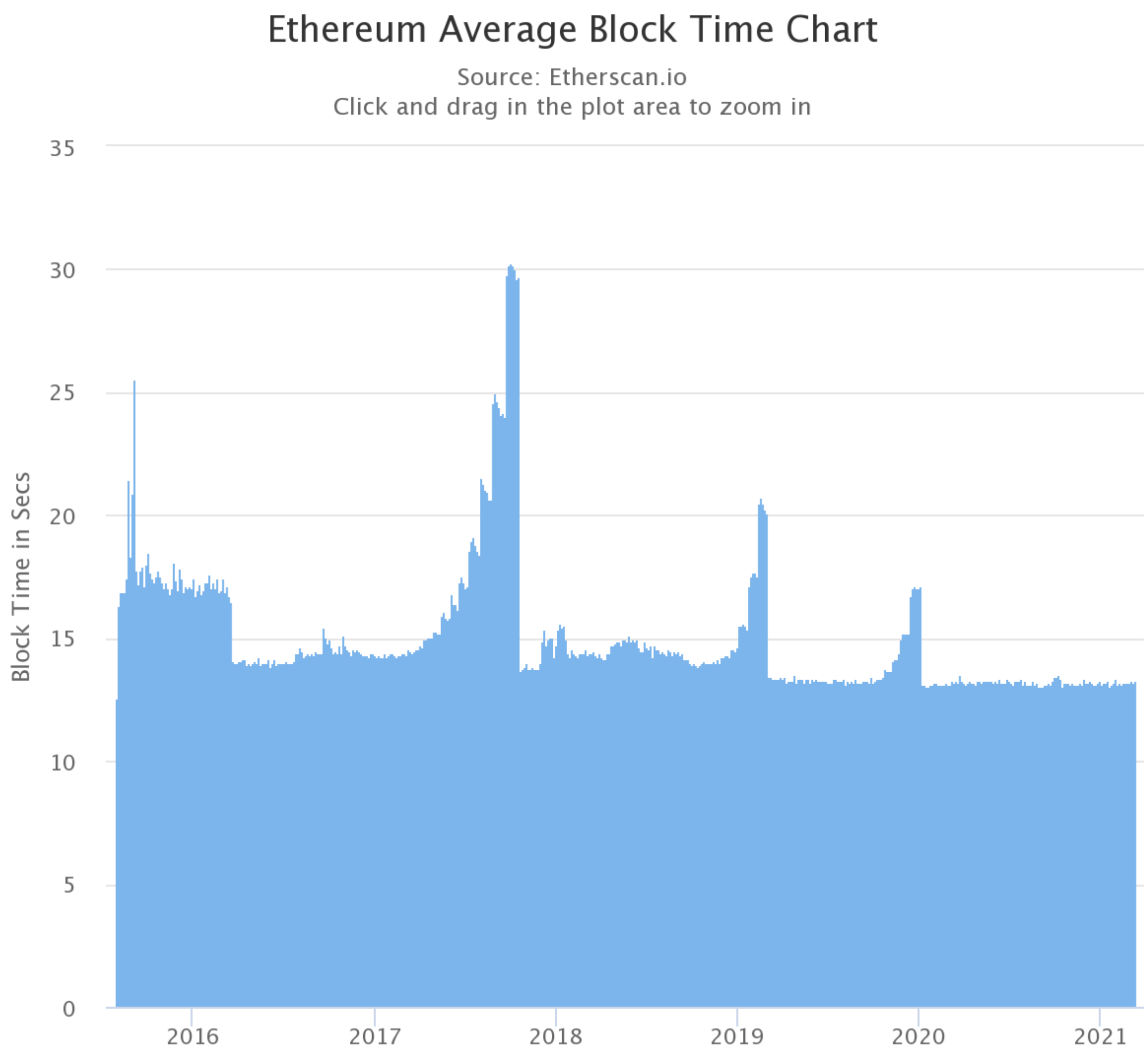
```
block_diff    =  parent_diff + parent_diff / 2048 *
                 max(1 - (block_timestamp - parent_timestamp) / 10, -99) +
                 int(2**((block.number / 100000) - 2))
```

Difficulty bomb

- Increases the difficulty exponentially every 100,000 blocks
- Goal: To reduce number of miners
  - Transition from PoW to Proof-of-Stake (PoS)
  - Shift in balance of power and profits away from miners into investors and users of the blockchain

Ethereum Mining

Impact of Difficulty Bomb



Ethereum PoS Transition

Ethereum is moving to Proof of Stake (PoS) consensus (ETH 2.0 phase 1)

- PoS does not incur huge computation resource and energy consumption
  - Also reduce 51% attack and fast TX validation
  - Disadvantage: may be more centralized
- Miners become “validators” and deposit to an escrow account
- The more escrow a miner deposit, the higher chance it will be chosen to mint next block
- Lose deposit if minting a block with invalid transactions

Decentralized Applications (DApp)

What is DApp?

Distributed application (and its data) running across multiple nodes

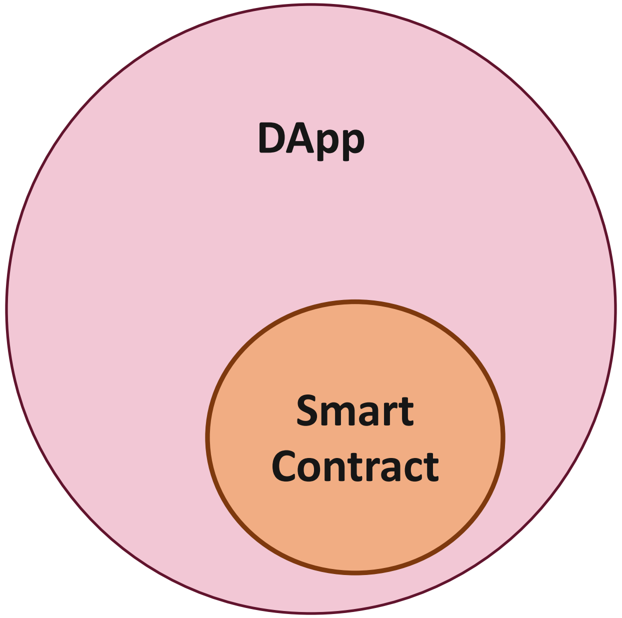
No single (central) point of failure, unkillable

DApp vs. Smart Contract

DApp is a <u>complete</u> application containing

Front-end (e.g., GUI)

Back-end (e.g., blockchain)

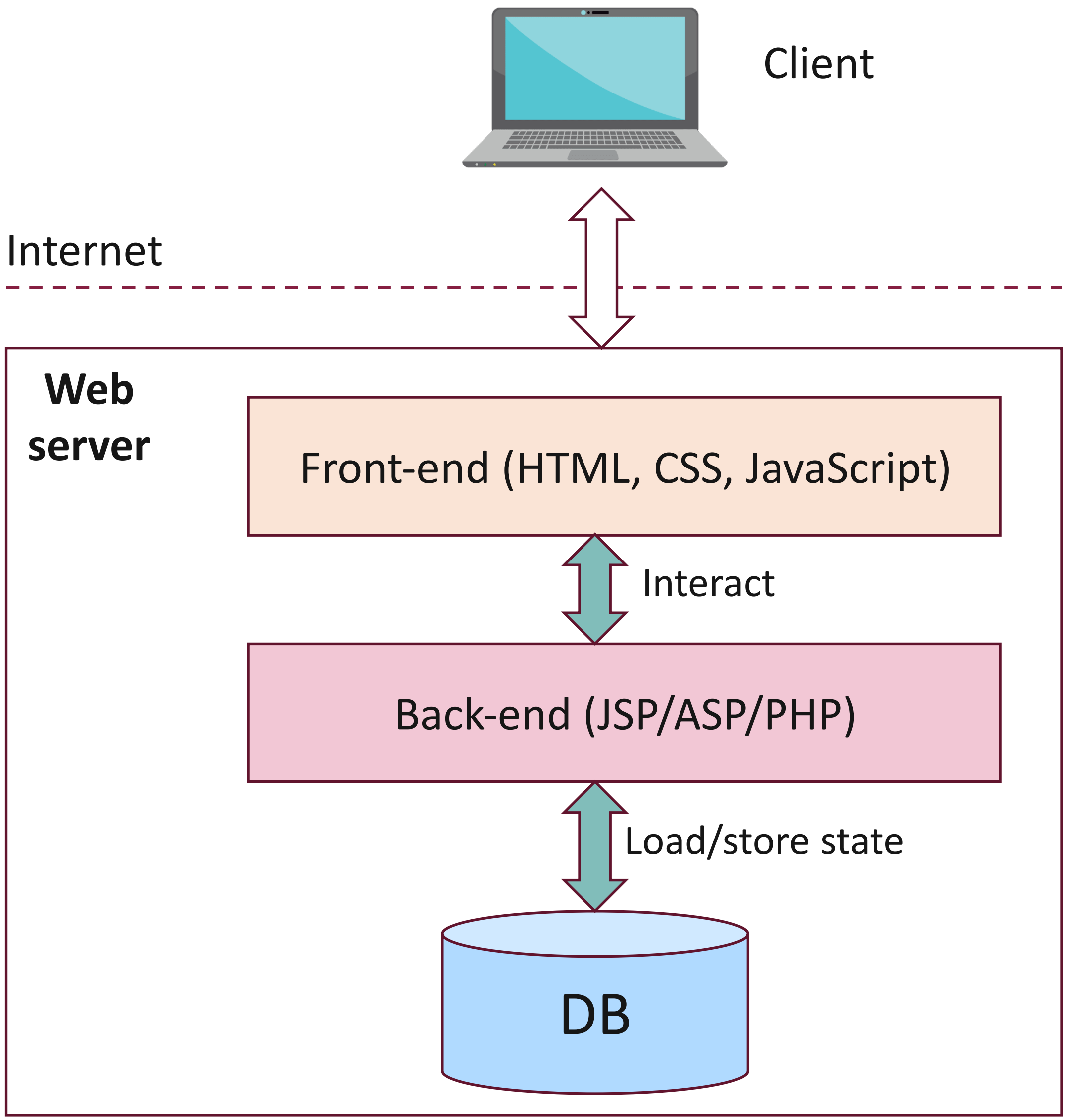


Smart contract is only a <u>part</u> of DApp that interacts with the blockchain

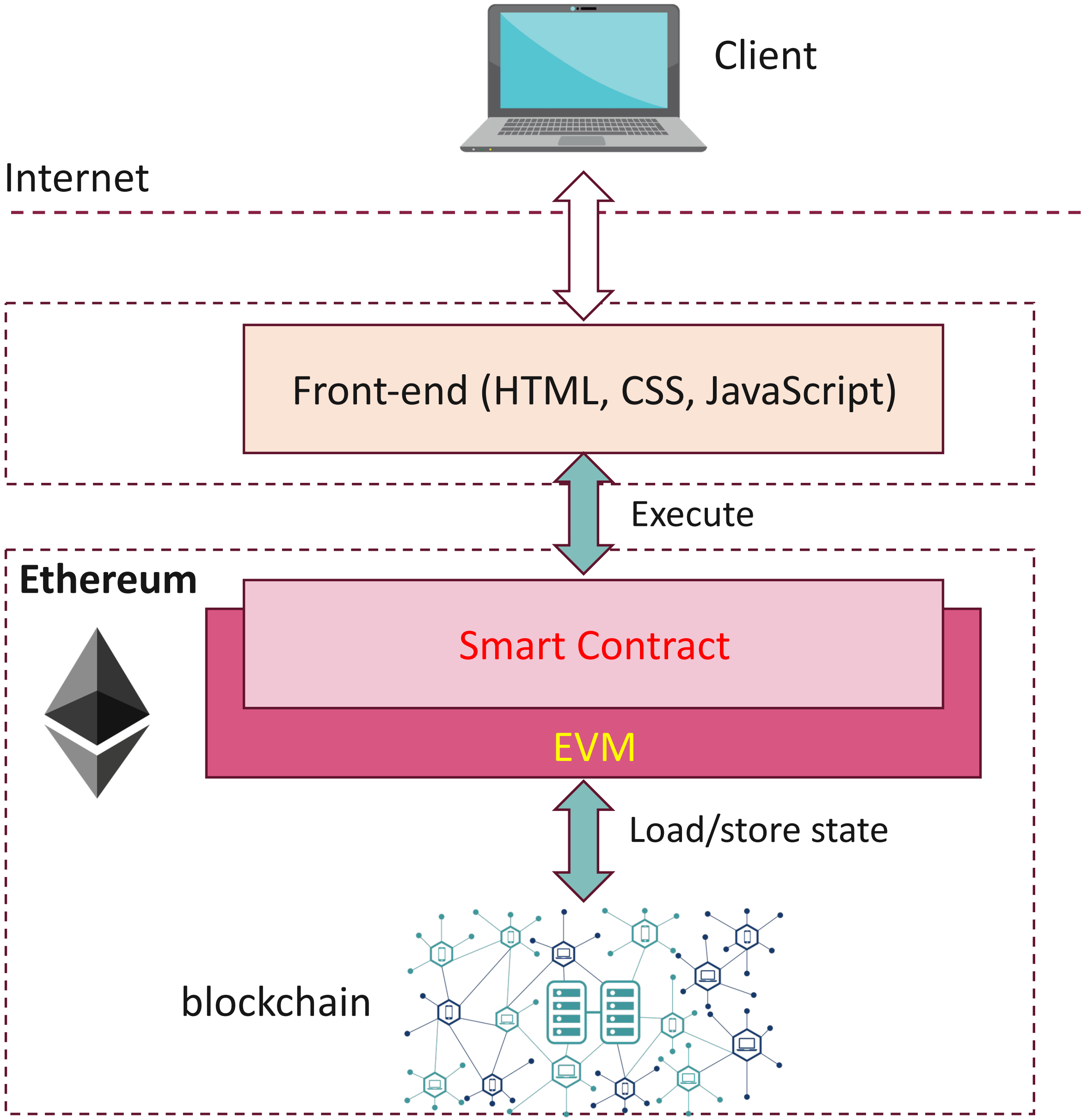
DApp vs. Centralized App

Architectural differences

Traditional Web Application



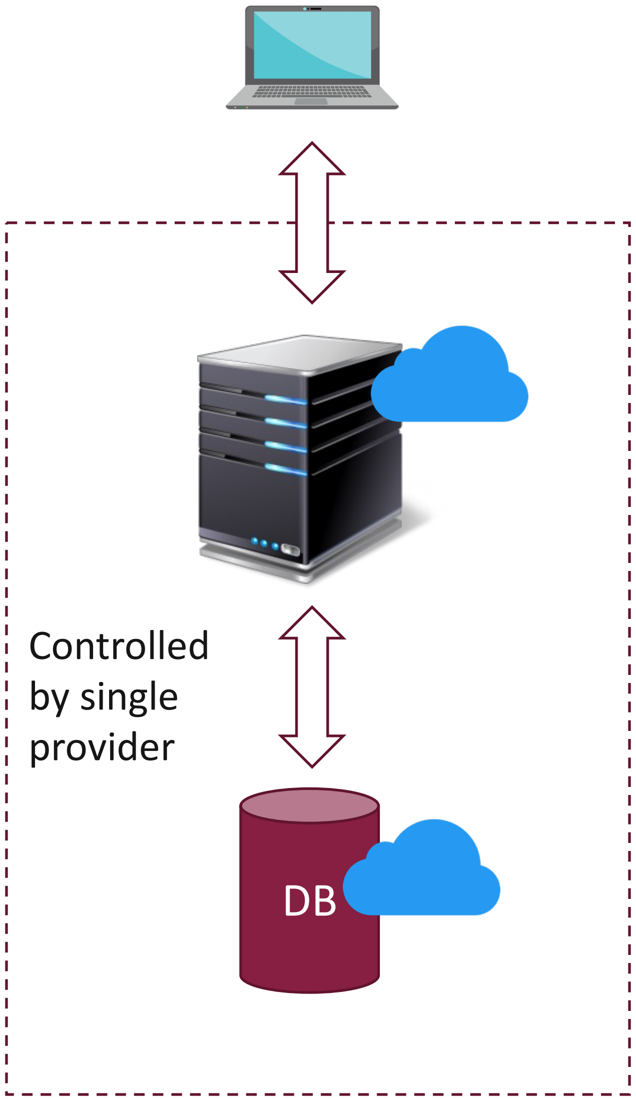
Decentralized Web Application



DApp vs. Centralized Application

Centralized applications follow standard client-server model

Front-end and back-end run by a single service provider



Advantages

- Low latency, high throughput
- Cost
- Easy to manage

Disadvantage

- Security, single point of failure
- Privacy
- Censorship

DApp vs. Centralized Application

Decentralized applications follow P2P model

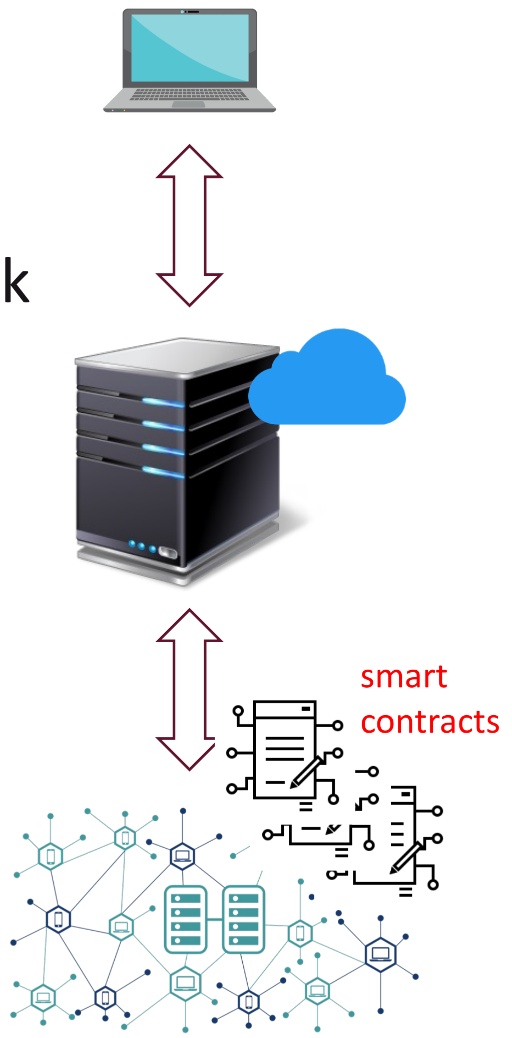
- Front-end run by some entities (P2P, static servers)
- Front-end talks to smart contracts using its API (via Wallets)
- Smart contracts execute code and store data on blockchain network

Advantages

- No censorship
- No single-point of failure

Disadvantage

- Cost
- High delay, low throughput
- Privacy: the right to be forgotten



Building DApp

Main principles to develop a DApp

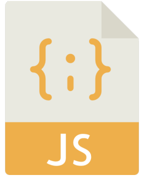
- **Develop Front-end:** create app's user interface

Front-end

- **Add library:** to connect front-end with wallet and blockchain

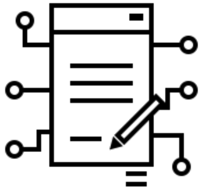
  User's wallet connect to the network and send TXs

library



- **Write smart contract:** contains your app's <u>core functions</u>, including anything that modifies user's wallet "contents"

Smart contract



- **Deploy:** deploy smart contract to the blockchain
  - Submit TX containing compiled smart contract without specifying any recipients

Blockchain

DApp Workflow Example

Life cycle of a voting application

- Voting TX is created by voter (via Web UI) via Voting Smart Contract
- TX validated and propagated throughout the network
- Voter gets confirmation once TX included in Blockchain

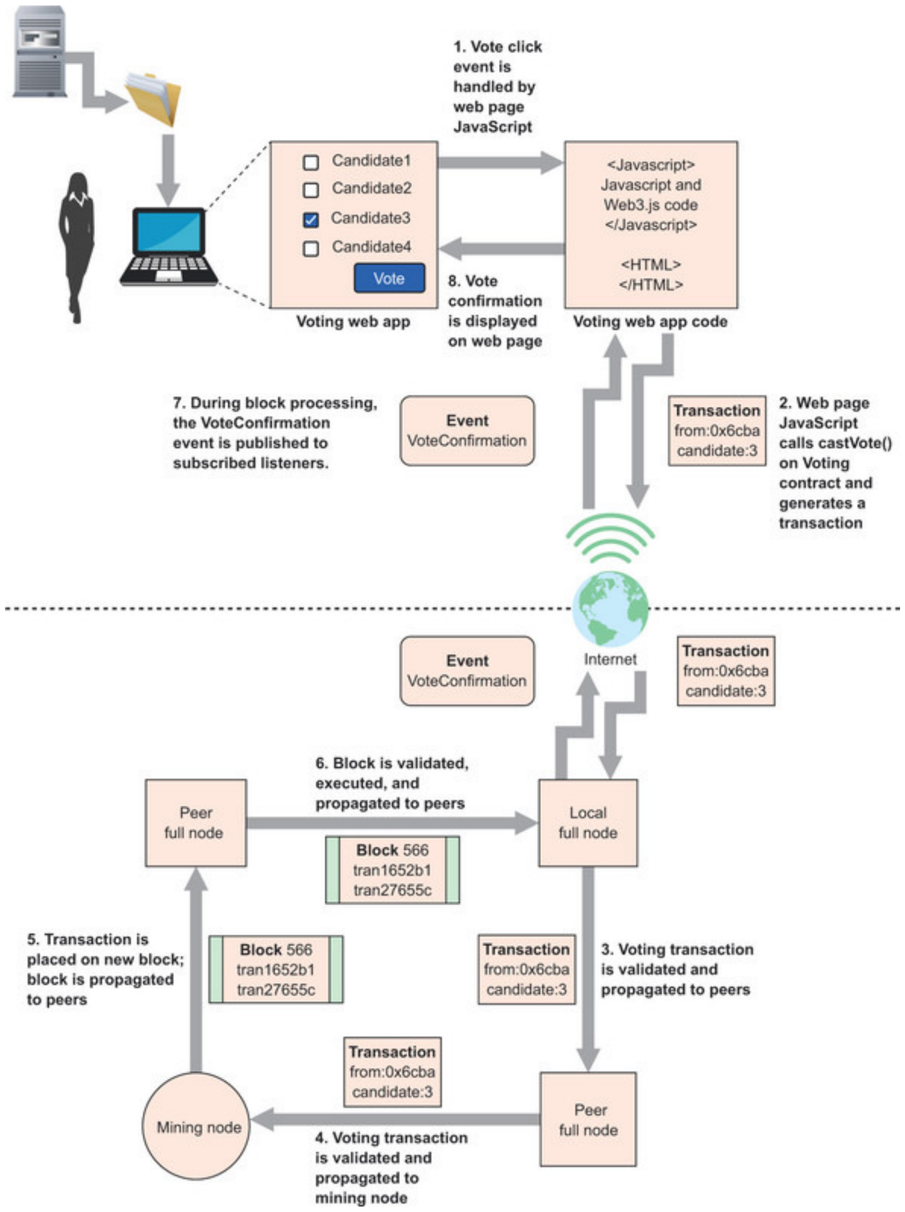


Image from https://livebook.manning.com/book/exploring-ethereum-dapps/chapter-1/94

Off-chain Storage

- Sometime data is too large to store directly on blockchain
  - Increase block size, computation (validation) and storage overhead on blockchain nodes
- **Solution**: store data content off chain, and its hash and address on chain
  - Example: IPFS, Swarm, Filecoin