Ethereum Basics

Gallersdörfer, U., Holl, P., & Matthes, F. (2020). "Blockchain-based Systems Engineering". Lecture Slides. TU Munich.

Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
wwwmatthes.in.tum.de

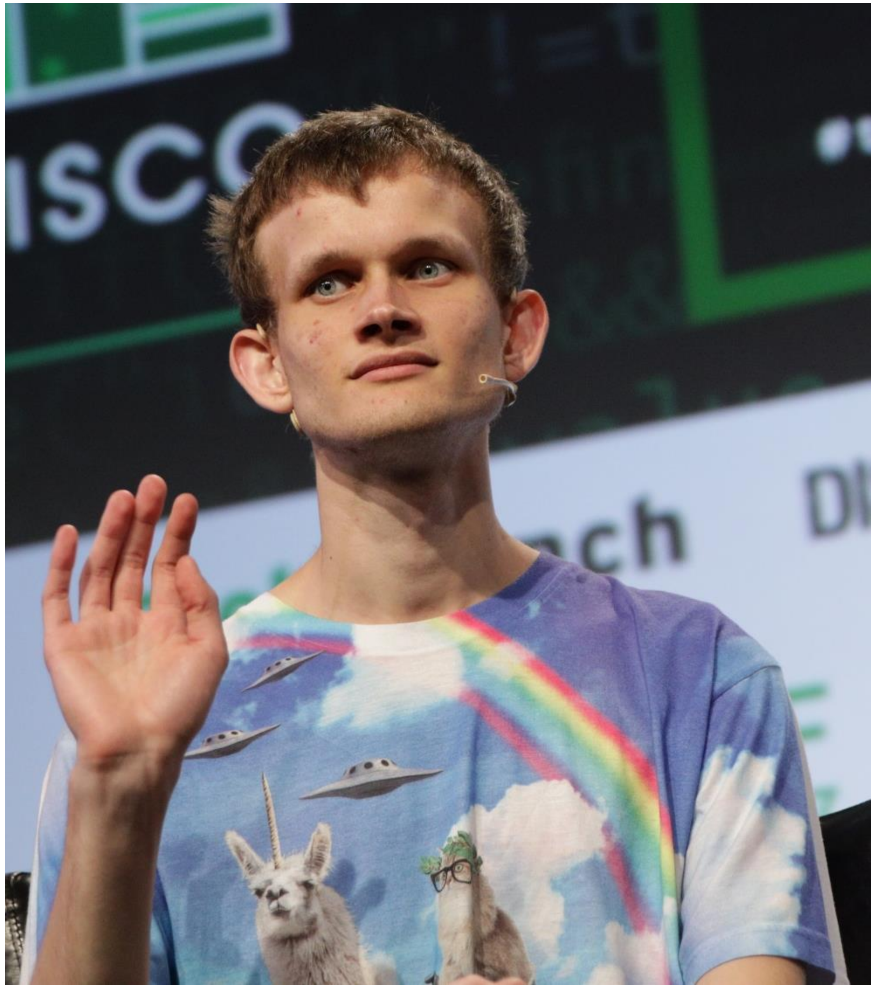
Outline

1. Ecosystem
   - Historical overview
   - Crowdsale statistics
   - Technical papers
   - Foundations
   - Network metrics

2. System architecture
   - Concept of a world computer
   - EVM
   - Accounts
   - Blockchain properties
   - Smart contracts

3. Network architecture
   - Overview
   - Node types
   - Clients
     - Geth
     - OpenEthereum

History



Vitalik Buterin at a Techcrunch conference

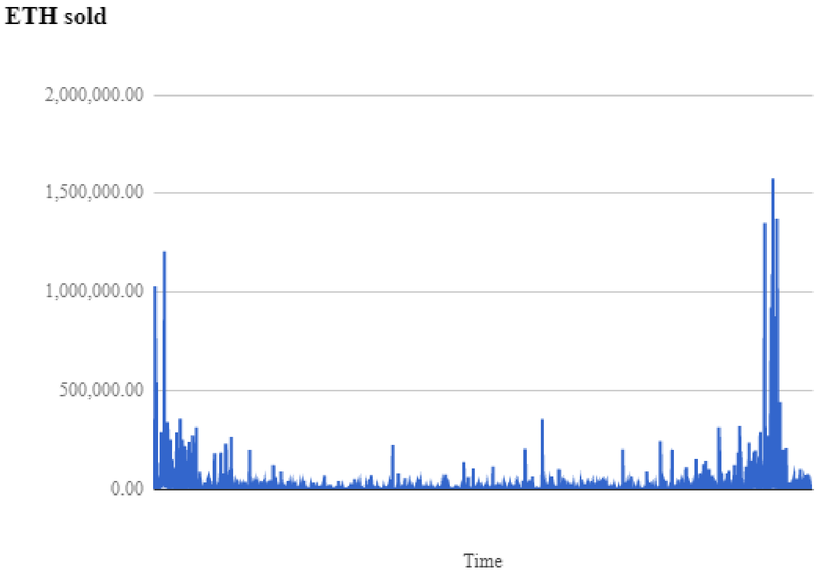
- In **November 2013**, Vitalik Buterin started working on the **first version** of the Ethereum **white paper**
- Buterin made the **first public announcement** of Ethereum on **24th January of 2014** at the Bitcoin conference in Miami
- On the **7th July of 2014**, Buterin **announced the start** of the **public crowd sale**
- The **sale lasted 42 days** until 2nd of September
  - For the first 14 days the price was 1 BTC for 2000 ETH
  - After that period the price went up to 1 BTC for 1337 ETH
- In total, **~60 million Ether** were sold in exchange for **31.591 Bitcoins**
  - Worth around 18.5 million USD at that time
  - Used by the Ethereum foundation

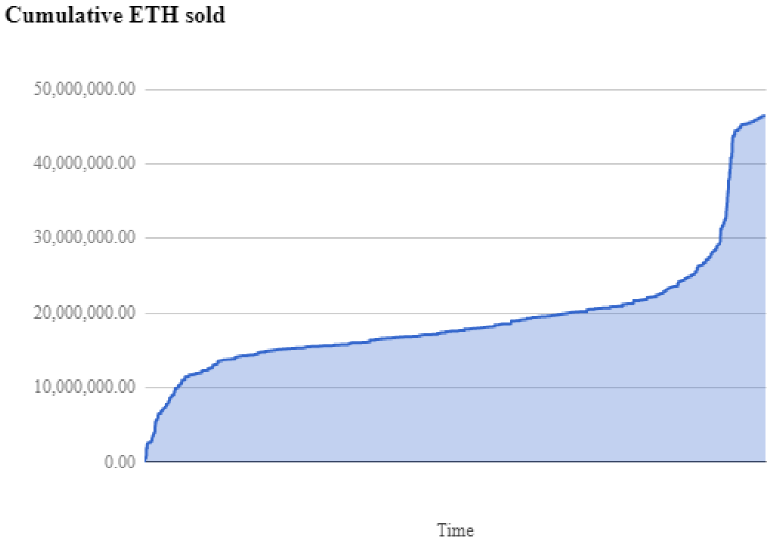
History of Ethereum http://ethdocs.org/en/latest/introduction/history-of-ethereum.html

CC BY-SA 4.0

Crowdsale statistics

First 14 day period

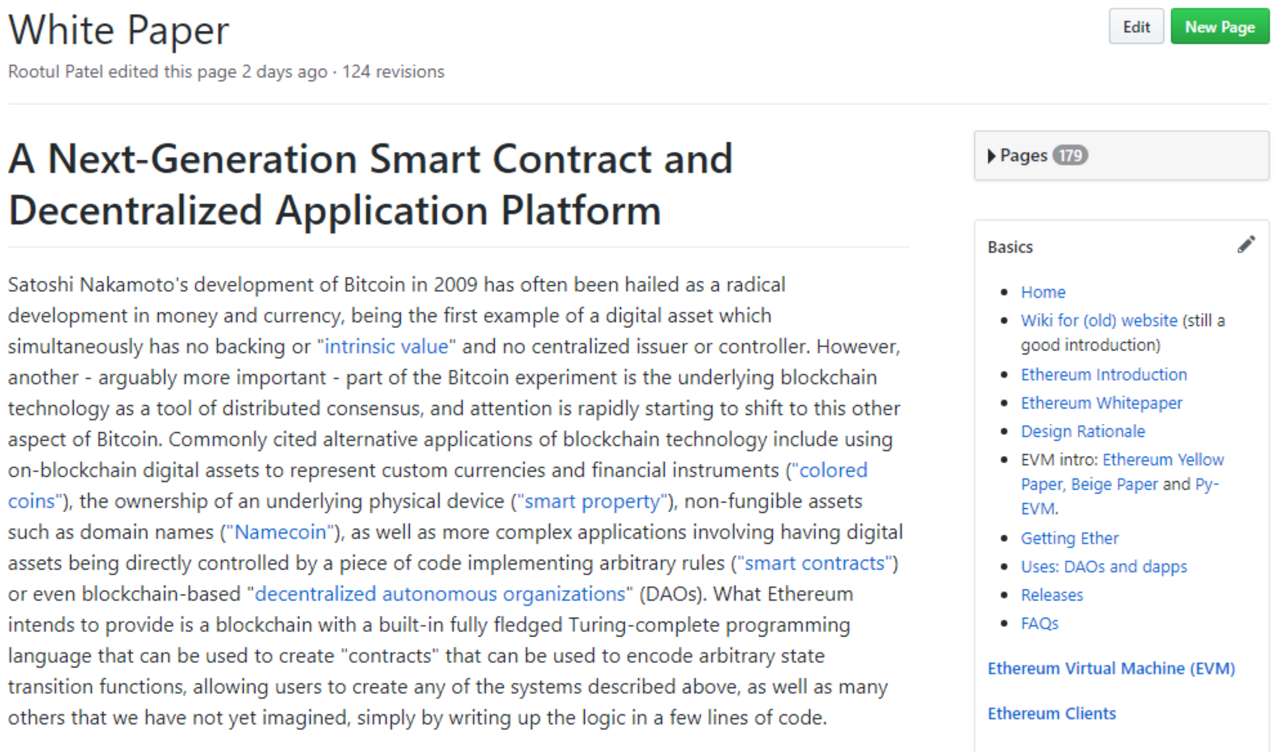




- Around **48 million ETH** were **sold during** the **first price period** of 14 days
- **Most ETH** were **sold** at the **beginning** and the **end** of the period
- **Biggest single purchase** during the first period was **500 BTC** which equals **1.000.000 ETH**
- **Smallest purchase** was **0.01 BTC**
- **43.6%** bought **2000 or more ETH**
- **0.8%** bought **200.000 or more ETH**
- **11,901,464.23948 ETH** to the **development team** (https://etherscan.io/address/0x5abfec25f74cd88437631a7731906932776356f9)

CC BY-SA 4.0

White Paper



White Paper

Rootul Patel edited this page 2 days ago · 124 revisions

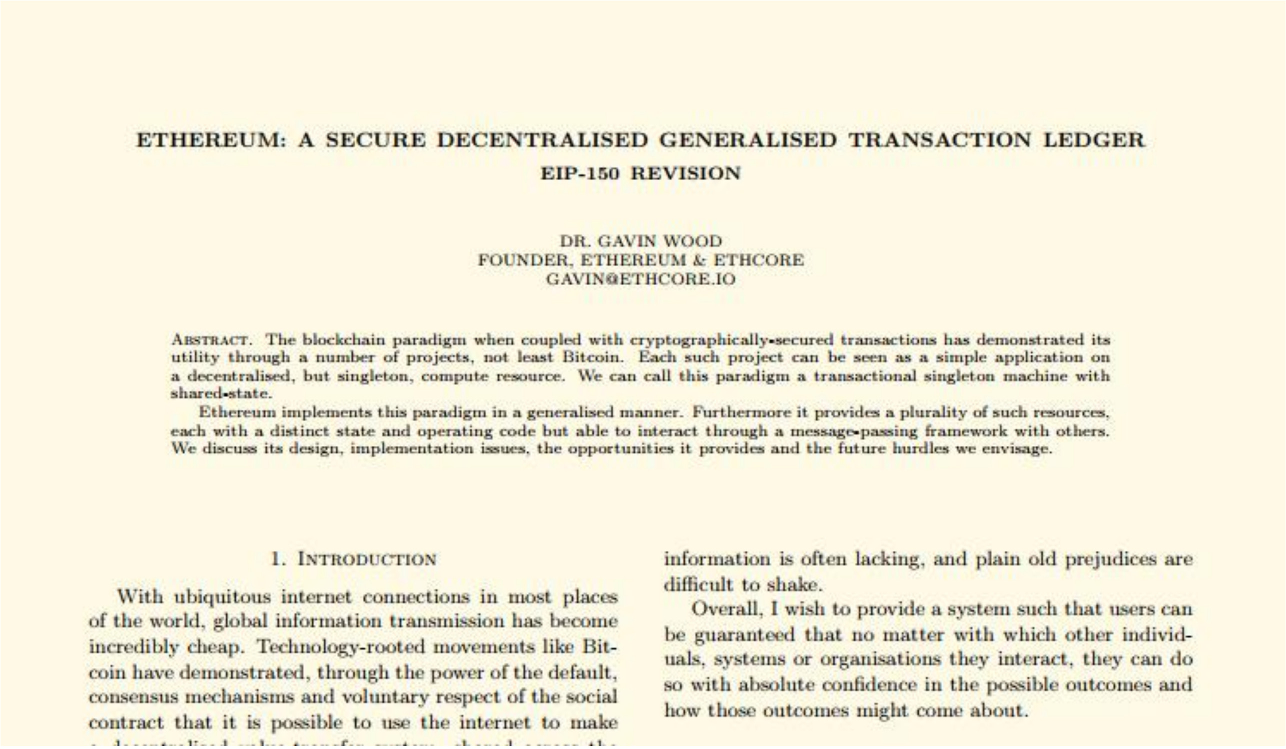
A Next-Generation Smart Contract and Decentralized Application Platform

Satoshi Nakamoto's development of Bitcoin in 2009 has often been hailed as a radical development in money and currency, being the first example of a digital asset which simultaneously has no backing or "intrinsic value" and no centralized issuer or controller. However, another - arguably more important - part of the Bitcoin experiment is the underlying blockchain technology as a tool of distributed consensus, and attention is rapidly starting to shift to this other aspect of Bitcoin. Commonly cited alternative applications of blockchain technology include using on-blockchain digital assets to represent custom currencies and financial instruments ("colored coins"), the ownership of an underlying physical device ("smart property"), non-fungible assets such as domain names ("Namecoin"), as well as more complex applications involving having digital assets being directly controlled by a piece of code implementing arbitrary rules ("smart contracts") or even blockchain-based "decentralized autonomous organizations" (DAOs). What Ethereum intends to provide is a blockchain with a built-in fully fledged Turing-complete programming language that can be used to create "contracts" that can be used to encode arbitrary state transition functions, allowing users to create any of the systems described above, as well as many others that we have not yet imagined, simply by writing up the logic in a few lines of code.

- **First draft** was written by Vitalik Buterin himself (**2013**)
- Contains **high level descriptions of** Ethereum's **core functionalities**
- **Living document** and **regularly updated** by Ethereum core developers (not only Buterin!)
- **Extensive summary** of the Ethereum platform and technology
- **Most current version** can be found the **public Git repository** of Ethereum: https://github.com/ethereum/wiki/wiki/White-Paper

CC BY-SA 4.0

Yellow Paper



ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER
EIP-150 REVISION

DR. GAVIN WOOD
FOUNDER, ETHEREUM & ETHCORE
GAVIN@ETHCORE.IO

Abstract. The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, not least Bitcoin. Each such project can be seen as a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state.

Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.

1. Introduction

With ubiquitous internet connections in most places of the world, global information transmission has become incredibly cheap. Technology-rooted movements like Bitcoin have demonstrated, through the power of the default, consensus mechanisms and voluntary respect of the social contract that it is possible to use the internet to make

information is often lacking, and plain old prejudices are difficult to shake.

Overall, I wish to provide a system such that users can be guaranteed that no matter with which other individuals, systems or organisations they interact, they can do so with absolute confidence in the possible outcomes and how those outcomes might come about.

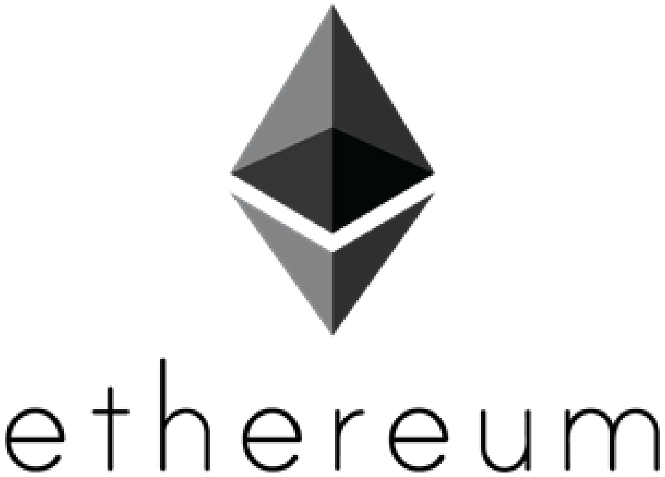
Ethereum yellow paper: https://ethereum.github.io/yellowpaper/paper.pdf

- **Published** in **April 2014** by **Dr. Gavin Wood**
- **Dr. Gavin Wood** is still listed as **the only author**
- Defines the **technical specification of Ethereum**
- **Very detailed**, contains mathematical function definitions and byte code mappings
- **Required to implement** a full node
- **Only updated** when **errors** are found **or** the **specification changes**

CC BY-SA 4.0

Foundation

*“The Ethereum Foundation’s **mission is** to **promote and support Ethereum platform** and **base layer research**, **development** and **education** to bring decentralized protocols and tools to the world that empower developers **to produce next generation decentralized applications** (dapps), and together build a more globally accessible, more free and more trustworthy Internet.”*



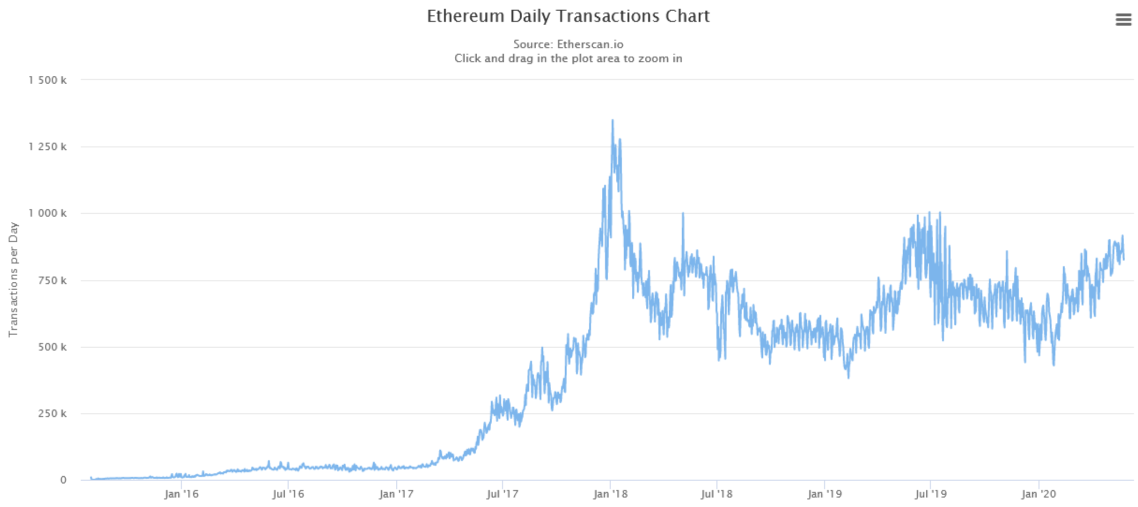
- **Founded** in **June 2014 in Zug**, Switzerland
- **Non-profit** organization
- **Foundation council** consists of **Vitalik Buterin** and **Patrick Storchenegger** who is responsible for all legal affairs
- **Owns** (or had owned) at least **31.591 Bitcoins** funding capital due to the crowdsale

Official website of the Ethereum foundation https://www.ethereum.org/foundation

CC BY-SA 4.0

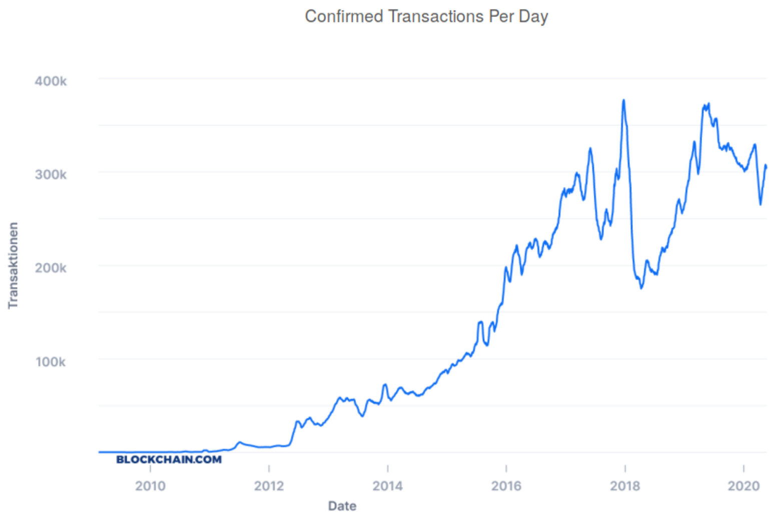
Like Bitcoin, Ethereum is in productive use

Network metrics: Transactions per day



Ethereum Daily Transactions Chart

Source: Etherscan.io
Click and drag in the plot area to zoom in



Confirmed Transactions Per Day

- **1.349.890** peak transactions per day
- Currently almost **3x** the number of **BTC transactions** each day

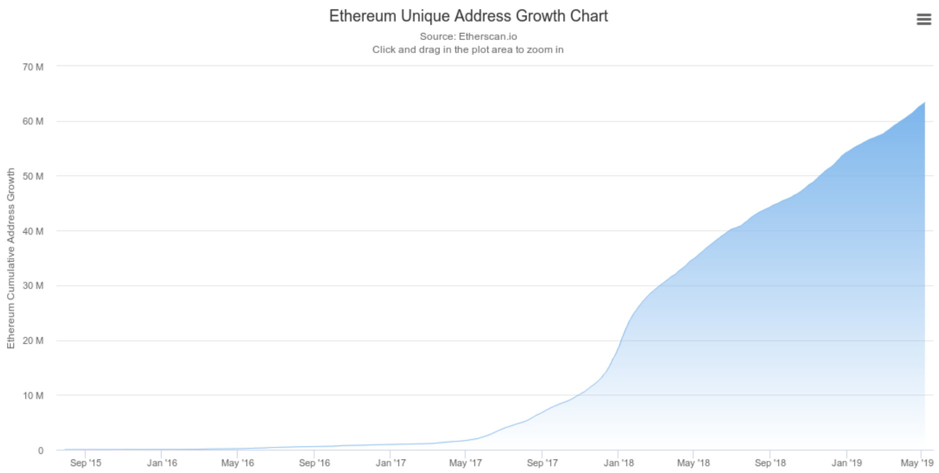
490.644 peak transactions per day

Ethereum transaction chart by Etherscan: https://etherscan.io/chart/tx
Bitcoin transaction chart by Blockchain.com: https://www.blockchain.com/charts/n-transactions

CC BY-SA 4.0

Like Bitcoin, Ethereum is in productive use

Network metrics: Active wallets

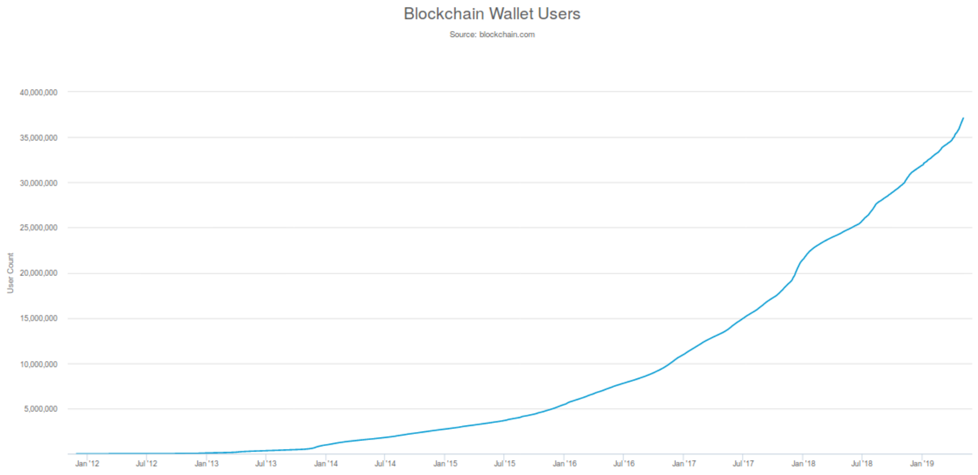


Ethereum Unique Address Growth Chart

Source: Etherscan.io

Click and drag in the plot area to zoom in

- Currently around **98 million unique wallets** with at least one incoming or outgoing transaction



Blockchain Wallet Users

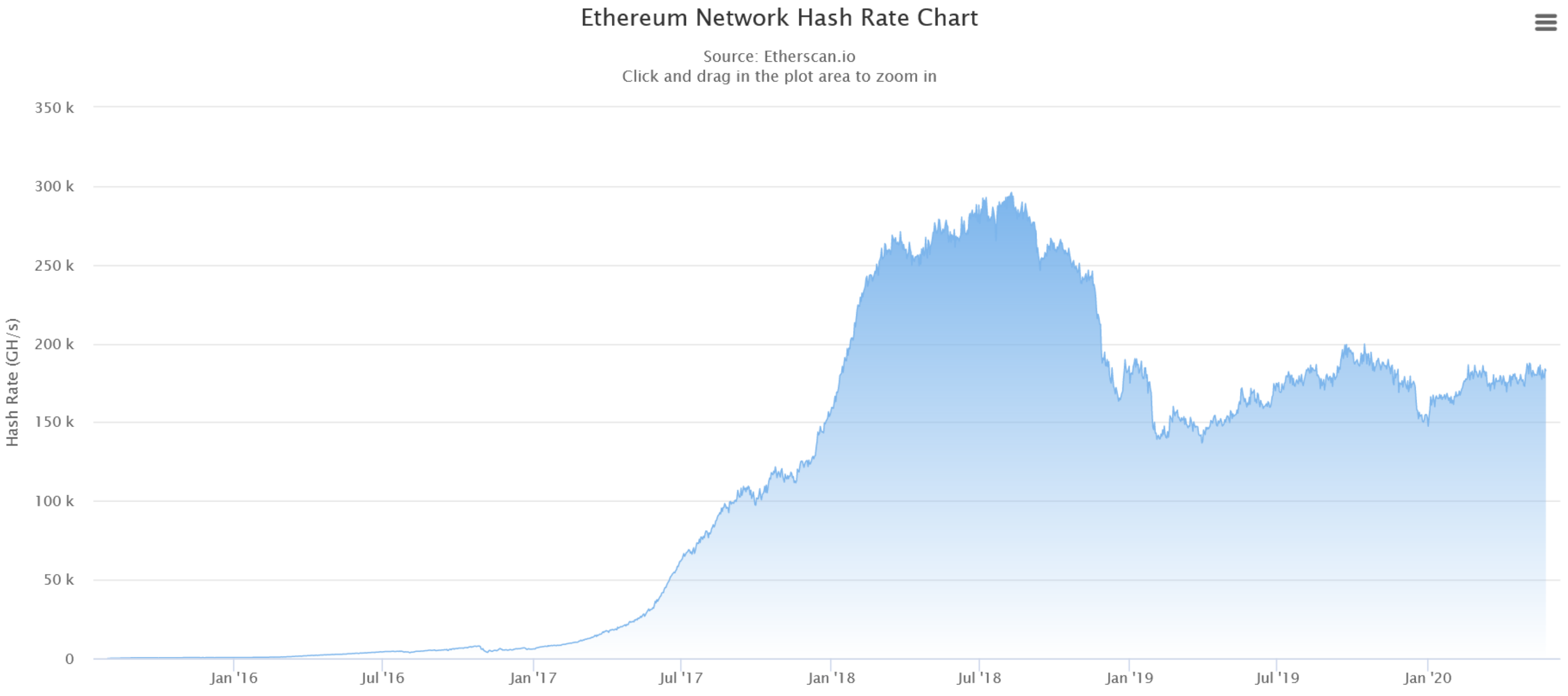
Source: blockchain.com

- Currently around **49 million unique wallets** with at least one incoming or outgoing transaction

Ethereum unique address chart by Etherscan: https://etherscan.io/chart/address
Bitcoin unique wallets chart by Blockchain.info: https://www.blockchain.com/de/charts/my-wallet-n-users

Like Bitcoin, Ethereum is in productive use

Also uses proof of work



- As of **May 2020**, the network **hash rate** is at around **182.000 GH/s**
- Mostly **GPUs** are used **for hashing (ASIC-resistant PoW)**
- Estimated **annual electricity consumption** at the current rate is **7.8 TWh**

Energy consumption estimation by Digiconomist: https://digiconomist.net/ethereum-energy-consumption
Ethereum network hash rate chart by Etherscan: https://etherscan.io/chart/hashrate

CC BY-SA 4.0

Outline

1. Ecosystem
   - Historical overview
   - Crowdsale statistics
   - Technical papers
   - Foundations
   - Network metrics

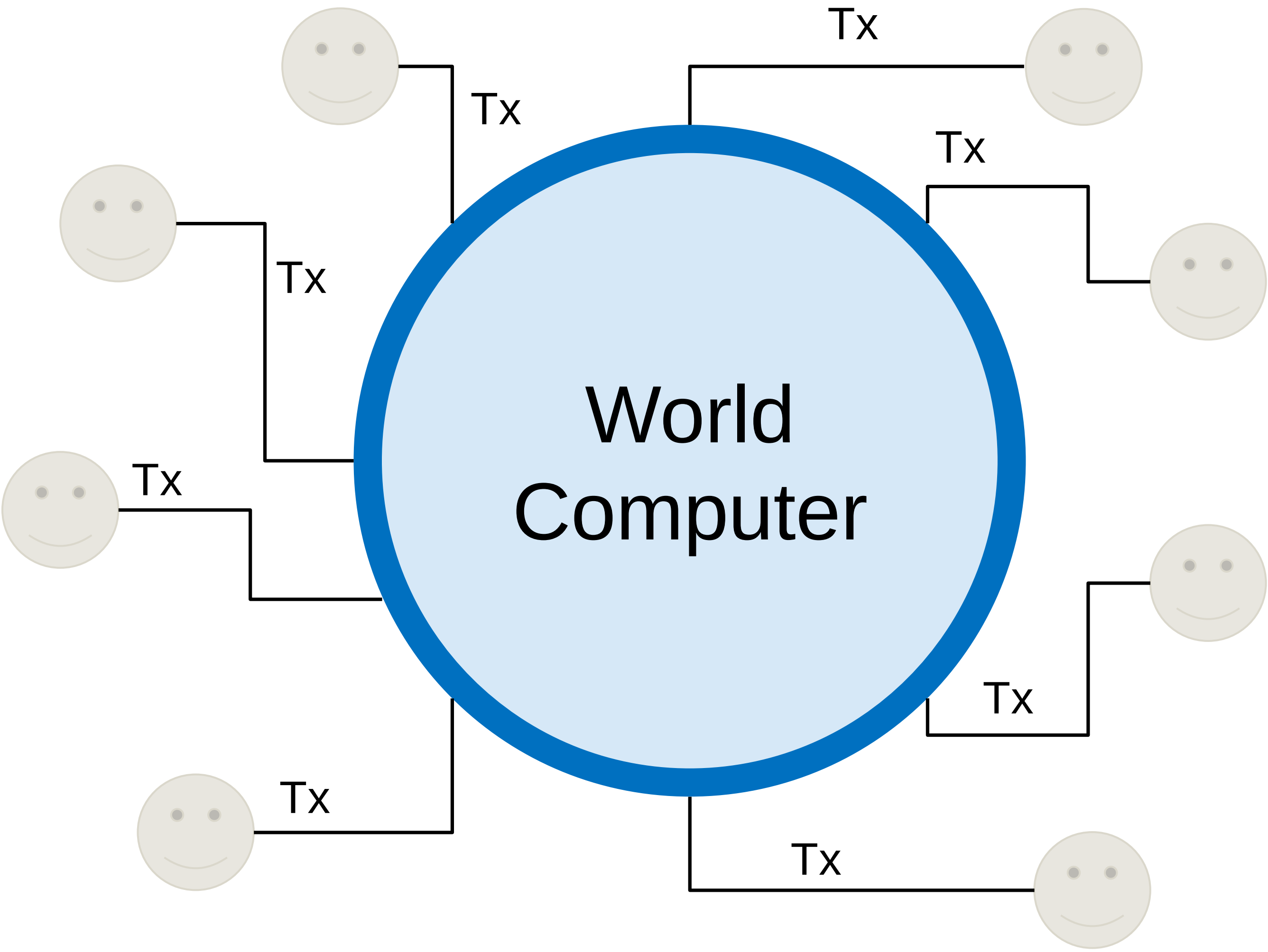
2. System architecture
   - Concept of a world computer
   - EVM
   - Accounts
   - Blockchain properties
   - Smart contracts

3. Network architecture
   - Overview
   - Node types
   - Clients
     - Geth
     - OpenEthereum

CC BY-SA 4.0

The concept of a world computer

State Machine



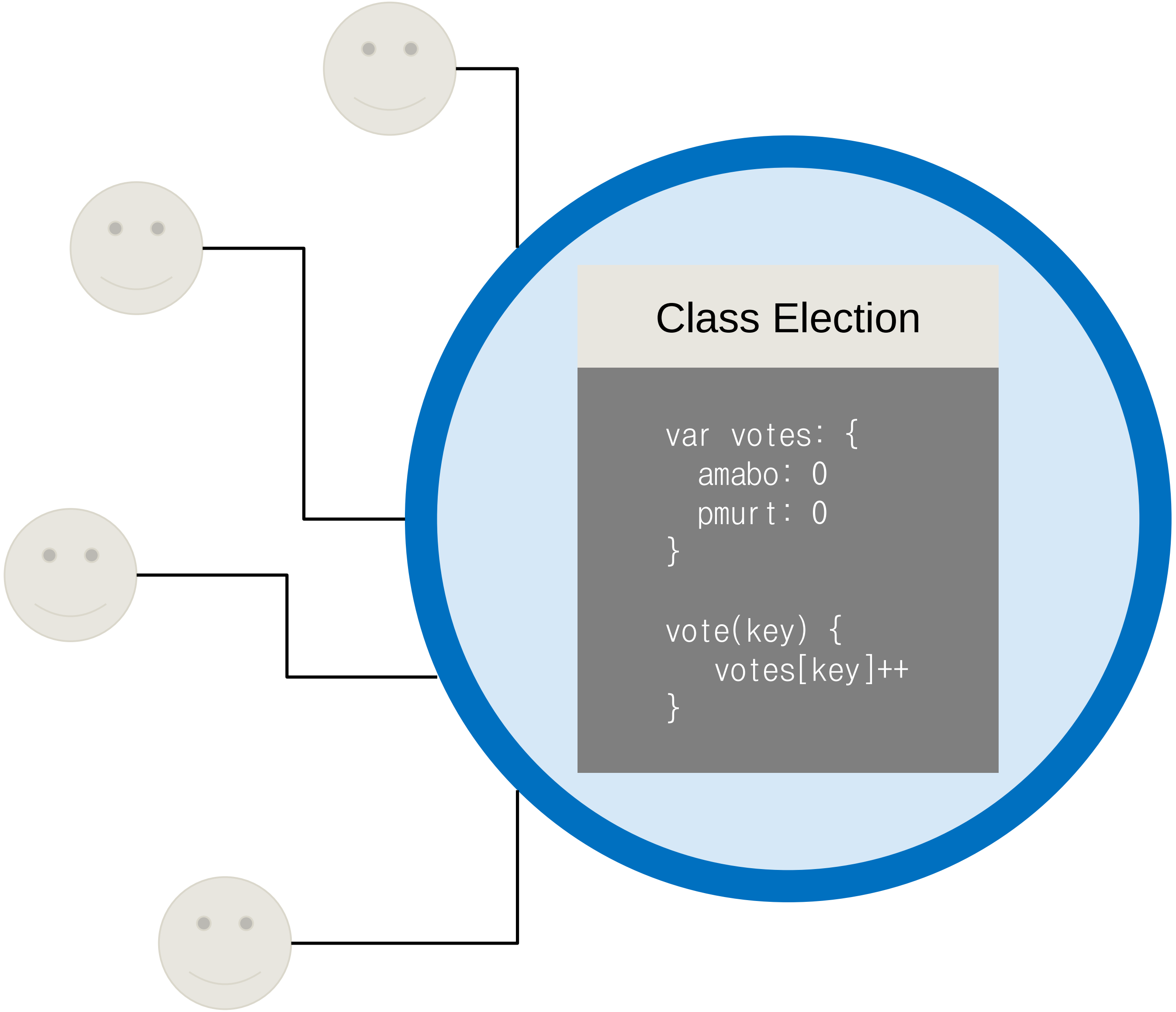
Tx = Transaction

Properties

- **All participants** are **using** the **same computer**
- Users issue **transactions to call programs** on the computer
- **Everyone shares** the same **resources** and **storage**
- The **computer has no explicit**, single **owner**
- **Using** the computer’s **resources costs money**

CC BY-SA 4.0

Election example using a world computer



Class Election

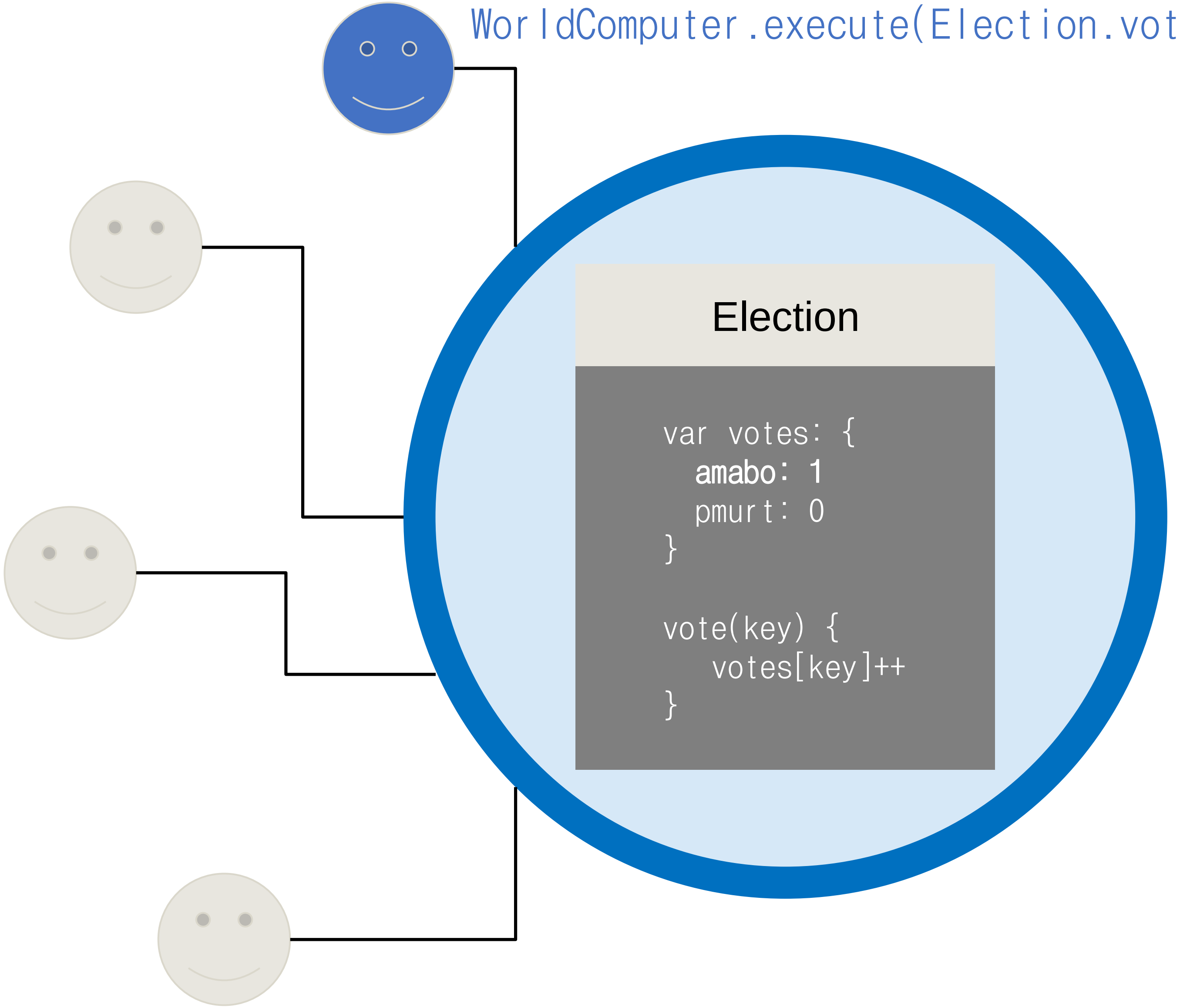
```
var votes: {
  amabo: 0
  pmurt: 0
}

vote(key) {
   votes[key]++
}
```

State of the world

State 0 (initial)
(nothing happened yet)

CC BY-SA 4.0

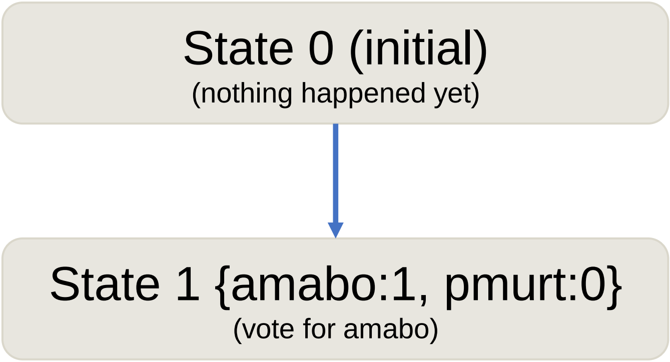
Election example using a world computer (cont.)



```
WorldComputer.execute(Election.vote("amabo"))
```

```
var votes: {
  amabo: 1
  pmurt: 0
}

vote(key) {
   votes[key]++
}
```

State of the world

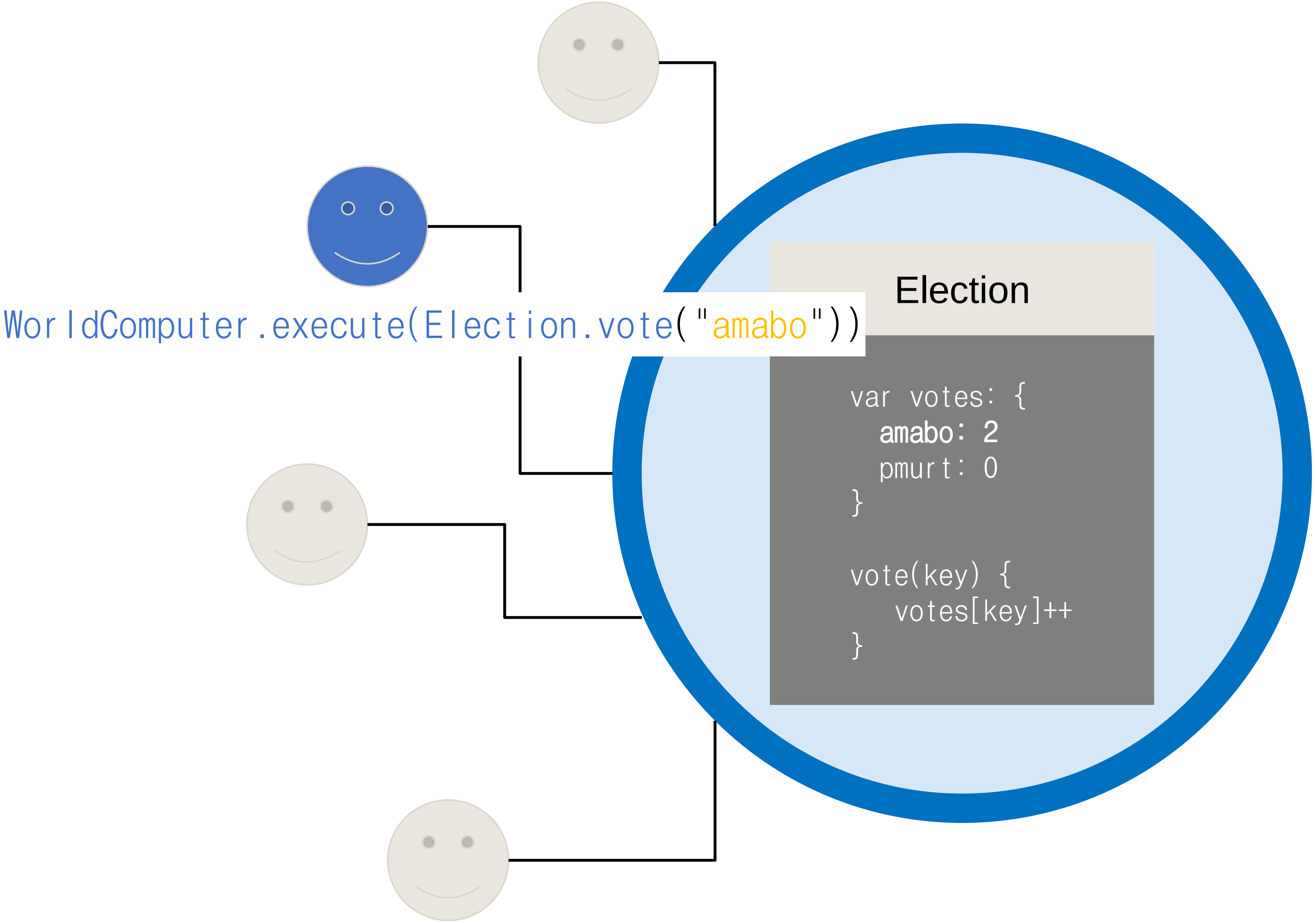


State 0 (initial)
(nothing happened yet)

State 1 {amabo:1, pmurt:0}
(vote for amabo)

CC BY-SA 4.0

Election example using a world computer (cont.)



```
WorldComputer.execute(Election.vote("amabo"))
```

Election

```
var votes: {
  amabo: 2
  pmurt: 0
}

vote(key) {
   votes[key]++
}
```

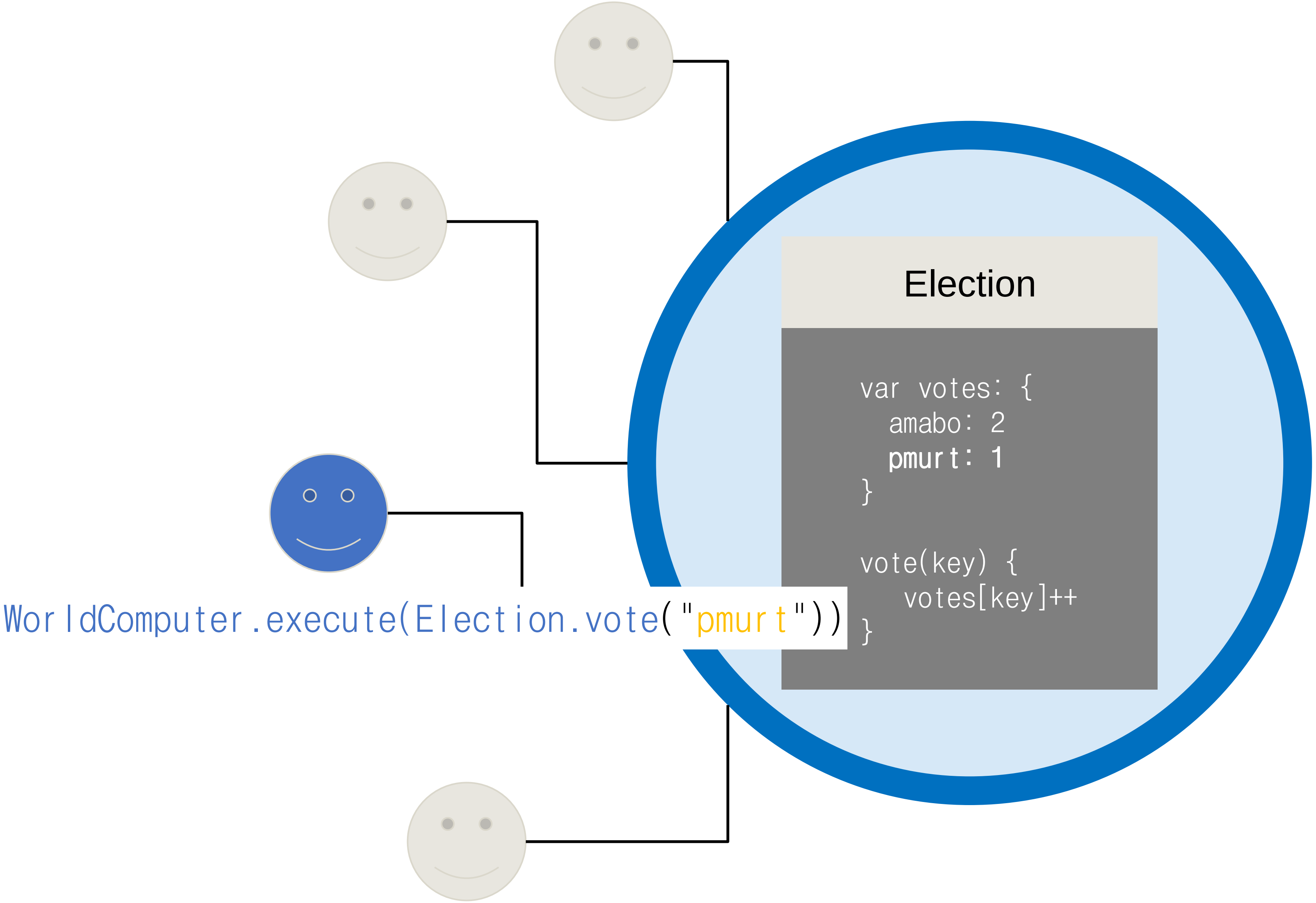
State of the world

State 0 (initial)
(nothing happened yet)

State 1 {amabo:1, pmurt:0}
(vote for amabo)

State 2 {amabo:2, pmurt:0}
(vote for amabo)

Election example using a world computer (cont.)



```
WorldComputer.execute(Election.vote("pmurt"))
```

Election

```
var votes: {
  amabo: 2
  pmurt: 1
}

vote(key) {
   votes[key]++
}
```

State of the world

State 0 (initial)
(nothing happened yet)

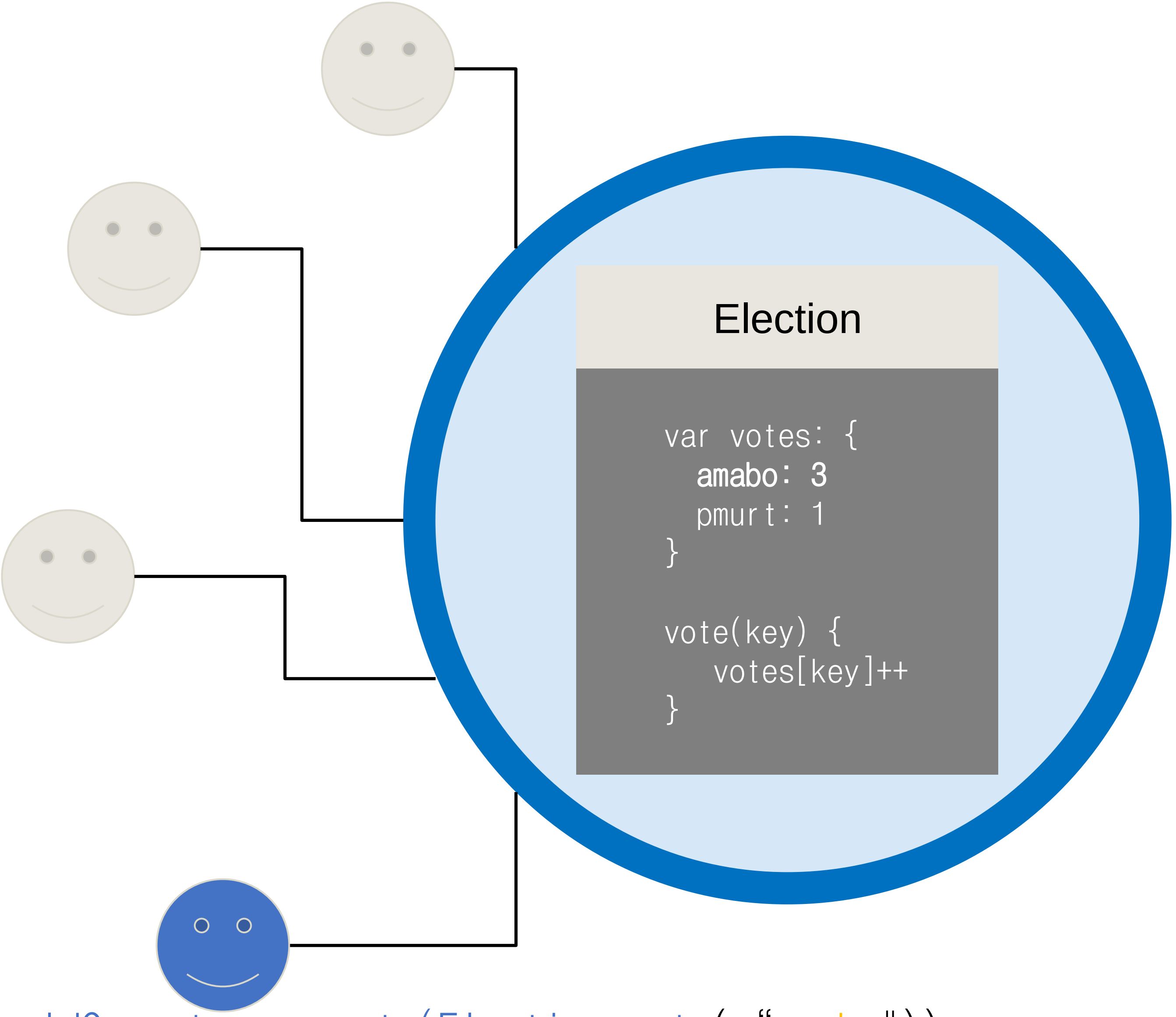
State 1 {amabo:1, pmurt:0}
(vote for amabo)

State 2 {amabo:2, pmurt:0}
(vote for amabo)

State 3 {amabo:2, pmurt:1}
(vote for pmurt)

CC BY-SA 4.0

Election example using a world computer (cont.)



```
var votes: {
   amabo: 3
   pmurt: 1
}

vote(key) {
    votes[key]++
}
```

`WorldComputer.execute(Election.vote( “amabo"))`

State of the world

State 0 (initial)
(nothing happened yet)

↓

State 1 {amabo:1, pmurt:0}
(vote for amabo)

↓

State 2 {amabo:2, pmurt:0}
(vote for amabo)

↓

State 3 {amabo:2, pmurt:1}
(vote for pmurt)

↓

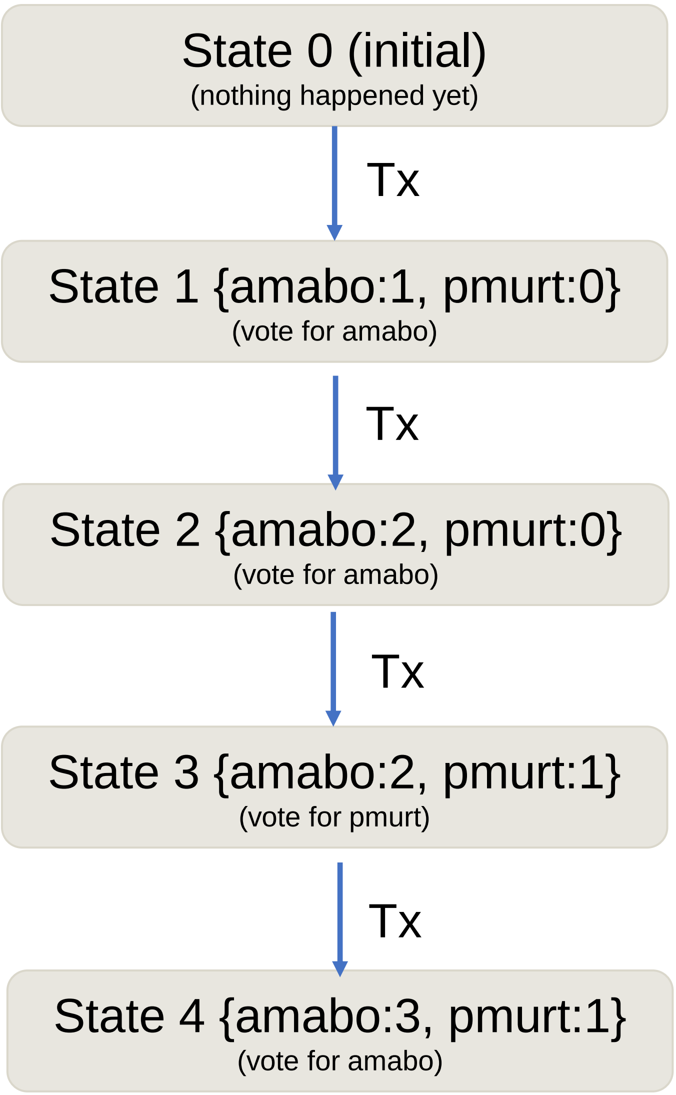
State 4 {amabo:3, pmurt:1}
(vote for amabo)

CC BY-SA 4.0

The blockchain as a state machine

Assumption
A block contains only one transaction

State of the world



State 0 (initial)
(nothing happened yet)

Tx

State 1 {amabo:1, pmurt:0}
(vote for amabo)

Tx

State 2 {amabo:2, pmurt:0}
(vote for amabo)

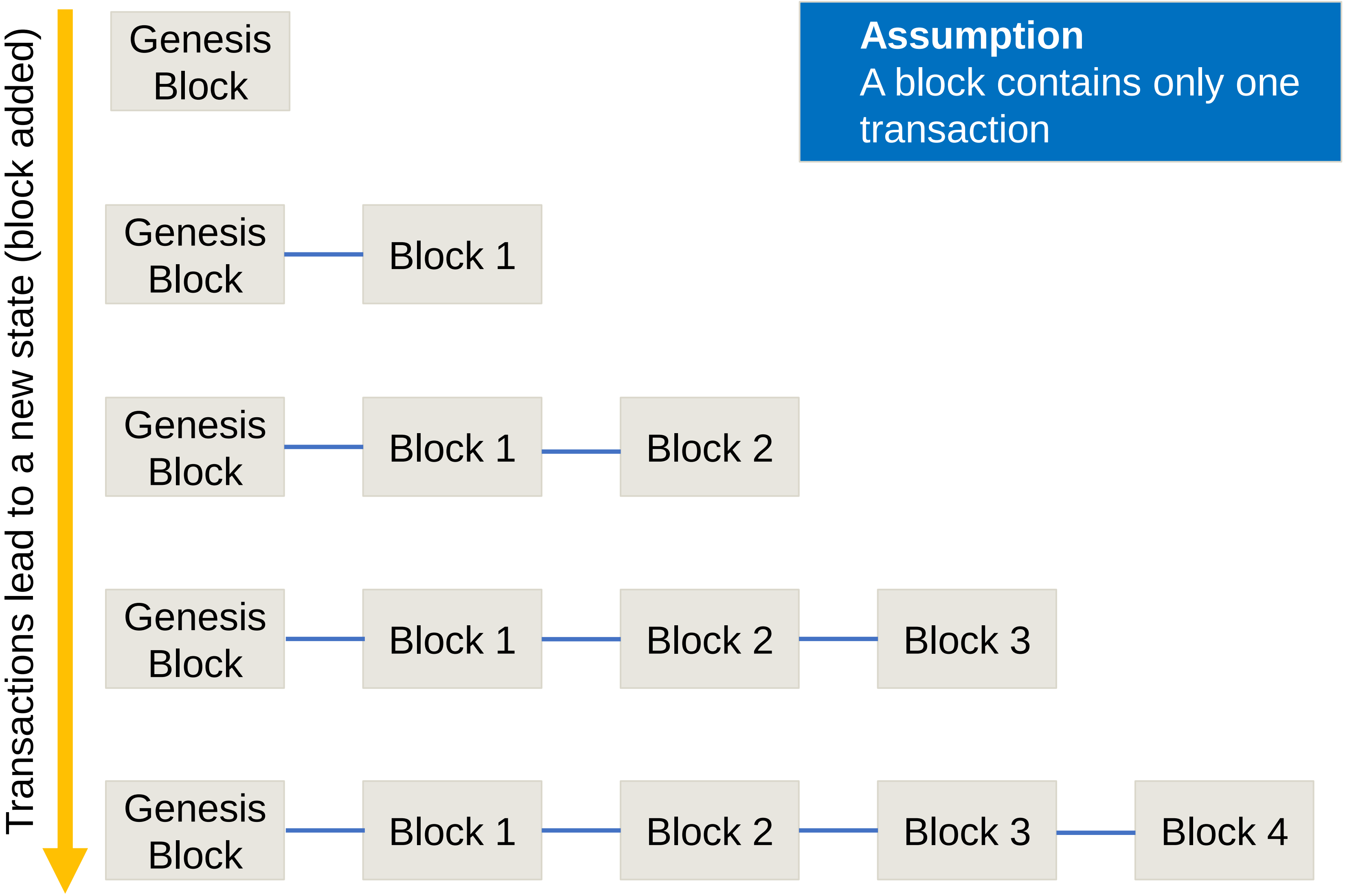
Tx

State 3 {amabo:2, pmurt:1}
(vote for pmurt)

Tx

State 4 {amabo:3, pmurt:1}
(vote for amabo)

Blockchain



CC BY-SA 4.0

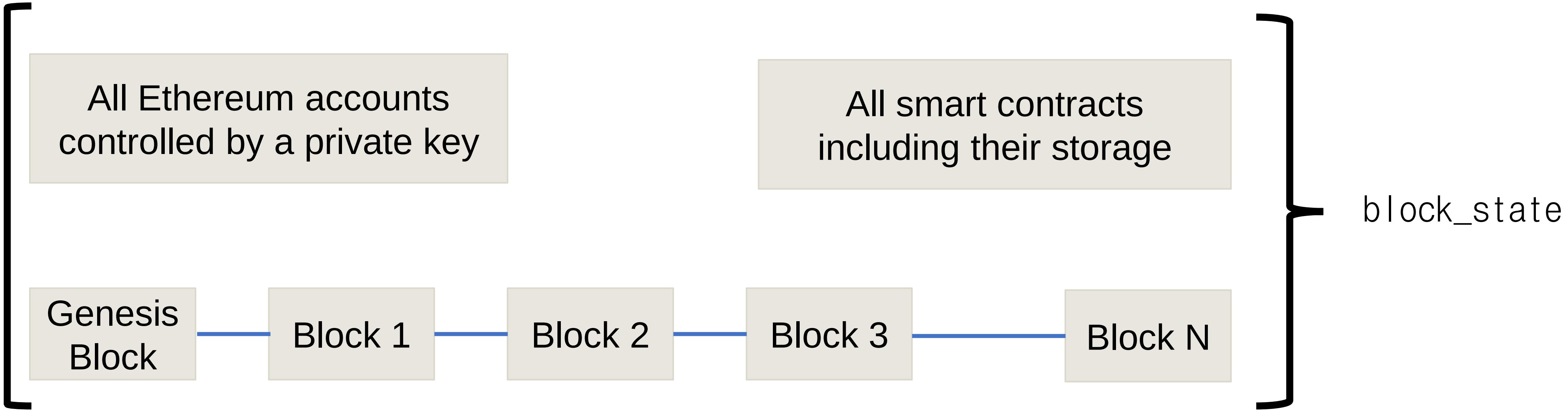
Ethereum Virtual Machine (EVM)

The concept of a state machine

The **EVM** specifies an **execution model for state changes** of the blockchain.

Formally, the **EVM** can be specified by the **following tuple**:
(block_state, transaction, message, code, memory, stack, pc, gas)

The *block_state* represents the **global state** of the whole blockchain including **all accounts, contracts and storage**


All Ethereum accounts controlled by a private key
All smart contracts including their storage
block_state
Genesis Block
Block 1
Block 2
Block 3
Block N

CC BY-SA 4.0

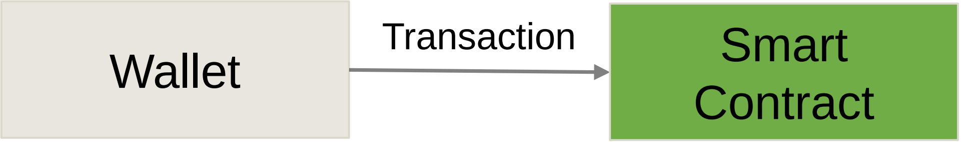
Ethereum Virtual Machine (EVM)

Transaction

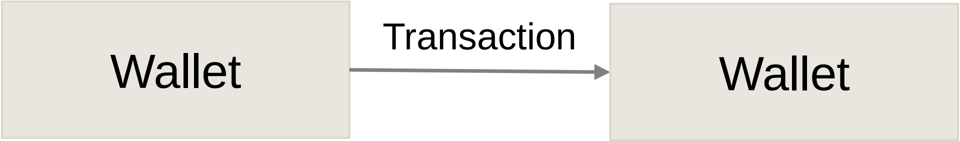
A **transaction** is a **signed data package** that is **always sent by a wallet** and contains the following data:

- The recipient of the message
- A signature identifying the sender
- The amount of ether to transfer from the sender to the recipient
- An optional data field
- A *STARTGAS* value, representing the maximum number of computational steps the transaction execution is allowed to take
- A *GASPRICE* value, representing the fee the sender pays per computational step
- There are two types of transactions: From wallet to wallet and from wallet to smart contract

Type 1: Wallet to Smart Contract



Type 2: Wallet to Wallet



CC BY-SA 4.0

Ethereum Virtual Machine (EVM)

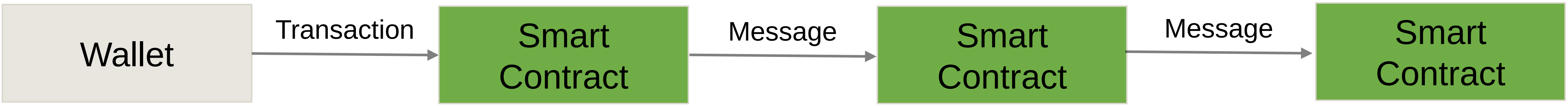
Message

A message is very similar to a transaction. Messages are only sent by contracts and exist only virtually, i.e. they are not mined into a block like transactions.

A message contains:

- The sender of the message (implicit)
- The recipient of the message
- The amount of ether to transfer alongside the message
- An optional data field
- A *STARTGAS* value

Whenever a **contract calls** a method on **another contract**, a virtual **message** is sent.
Whenever a **wallet calls** a method on a contract, a **transaction** is sent.



CC BY-SA 4.0

Ethereum Virtual Machine (EVM)

Code & memory

code

The code basically represents a smart contract as bytecode. For the EVM, a smart contract is a sequence of opcodes similar to assembly code.

Example:

```
PUSH1 0x60
PUSH1 0x40
MSTORE
PUSH1 0x04
CALLDATASIZE
LT
PUSH2 0x00b6
JUMPI
PUSH4 0xffffffff
```

memory

An infinitely expandable byte array that is non-persistent and used as temporal storage during execution.

CC BY-SA 4.0

Ethereum Virtual Machine (EVM)

Stack & program counter

stack

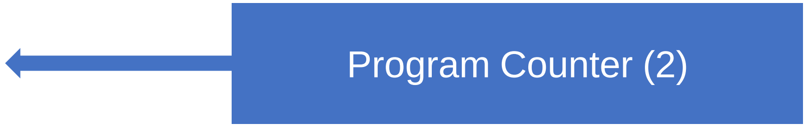
The stack is also used as a fast, non-persistent buffer to which 32 byte values can be pushed and popped during execution.

pc

PC stands for “program counter”. The program counter is always initialized with 0 and points to the position of the current opcode instruction.

Simple Opcode Execution Example:

```
0         PUSH1 0x60
1         PUSH1 0x40
2         MSTORE
3         PUSH1 0x04
4         CALLDATASIZE
5         LT
6         PUSH2 0x00b6
7         JUMPI
8         PUSH4 0xffffffff
```



Program Counter (2)

CC BY-SA 4.0

Ethereum Virtual Machine (EVM)

Gas

- **Every executed opcode instruction** uses a miner’s computational resources and therefore **costs a certain fee (called gas)**.

- Each opcode uses a certain amount of gas which may depend on the arguments of the operation, e.g., number of bytes to be allocated.

- The opcode for **selfdestruct(address)** uses **negative gas** because it frees up space from the Blockchain.

- At the beginning of a transaction the sender must specify a maximum amount of gas that he/she is willing to pay for the transaction to be executed.

- The sender can set an arbitrary amount of Ether he/she is willing to pay for each instruction called gas price.

- The final costs for each transaction are used gas * gas price.

- If a transaction requires more gas as the maximum specified gas, the transaction will fail. On the other hand, if it takes less, the sender only pays the gas that was used.

CC BY-SA 4.0

Ethereum account types

Compared to Bitcoin, **Ethereum** uses an **account-based ledger**. Each **distinct address** represents a separate, **unique account**.

Ethereum supports two types of accounts:

1. **Accounts that are controlled by private keys and owned externally**
   - Accounts that are controlled by a private key do not have any code stored on the blockchain. This type can be seen as the **default wallet of a user**. It can sign transactions, issue smart contract functions calls and send Ether from one account to another.
   - The **origin of any transaction** is **always** an account **controlled by a private key.**

2. **Smart Contract accounts which are controlled by their code**
   - Smart Contracts are treated as **account** entities with their **own**, unique **address**.
   - Contracts **can send messages** to other accounts, both externally controlled and smart contracts.
   - They **can't issue a transaction themselves.**
   - They **have** a persistent **internal storage** to write and read data from.

CC BY-SA 4.0

Account properties

On an abstract level, an Ethereum account is a 4-tuple containing the following data:
(nonce, balance, contract_code, storage)

nonce
An increasing number that is attached to any transaction to prevent replay attacks and double spending.

balance
The current account balance of the account in Ether.

contract_code
The bytecode representation of the account. If no contract code is present, then the account is externally controlled.

storage
The data storage used by the account and empty by default. Only contract accounts can have their own storage.

CC BY-SA 4.0

Properties of the Ethereum Blockchain

Ethereum (currently) is a Proof-of-Work Blockchain like Bitcoin.

- The **PoW algorithm** used by Ethereum is called **Ethash** and designed to be ASIC resistant.
- The **mining difficulty** is **adjusted** after **each block.**
- The actual **size of a block** is **not limited** like in Bitcoin. Instead, the miner sets a gas limit which defines how much computational resources he is willing to provide for each block.
- Correctly mined blocks that are **outpaced** by a **block** of another miner are **not orphaned** like in Bitcoin but **added as uncle blocks.**
  - The idea behind this is to counter mining centralization because miners who mine a correct block are still rewarded.
  - The **transactions** in the **uncle block** are considered **invalid.**

For further resources, see https://www.notion.so/BBSE-Additional-Content-85f1424258e64f679d3a6efac0d8c683

CC BY-SA 4.0

Smart contracts: Definition & peculiarities

- A **smart contract** is a **set of functions** that can be called by other users or contracts.
- They can be used to **execute functions, send ether or store data**.
- Each smart contract is an account holding object, i.e. **has its own address**.
- Smart contracts have some peculiarities compared to traditional software.

Security

The **development process** of smart contracts **requires special attention on security**.
Once **deployed**, a **contract** is **publicly accessible** by anyone on the network with the following information:

- Address of the smart contract
- OPCODE
- Number of public functions and their hash signature

- Furthermore, the whole transaction history is accessible (function calls + actual arguments).
- **Smart contracts** – once **deployed** – **cannot** be **changed** or **patched** anymore.

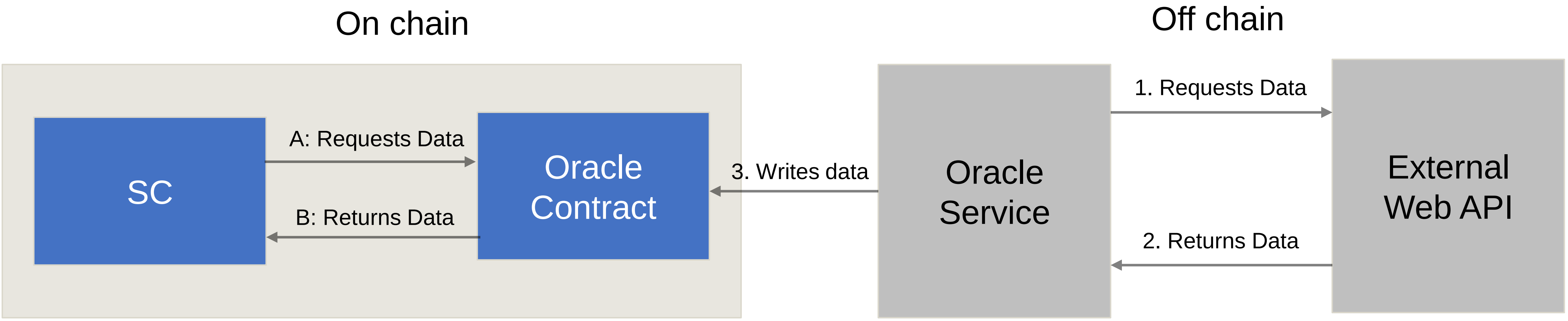
➔ All contracts deployed on the Ethereum blockchain are publicly accessible and can’t be patched.

CC BY-SA 4.0

Smart contracts are closed systems

Smart contracts can’t access any **data** from **outside** the **blockchain** on their own. There are no HTTP or similar network methods implemented to call external services. This is on purpose to **prevent non-deterministic behavior** once a function is called (there are also no functions to generate random values).

Currently, the **only way** to write smart contracts **using external data** (e.g. weather data, traffic data etc.) is to **use oracles**. Oracles are basically third-party services that verify data from web services and write the data via a special smart contract to the blockchain. Other smart contracts can now call the oracle contract to get the data.



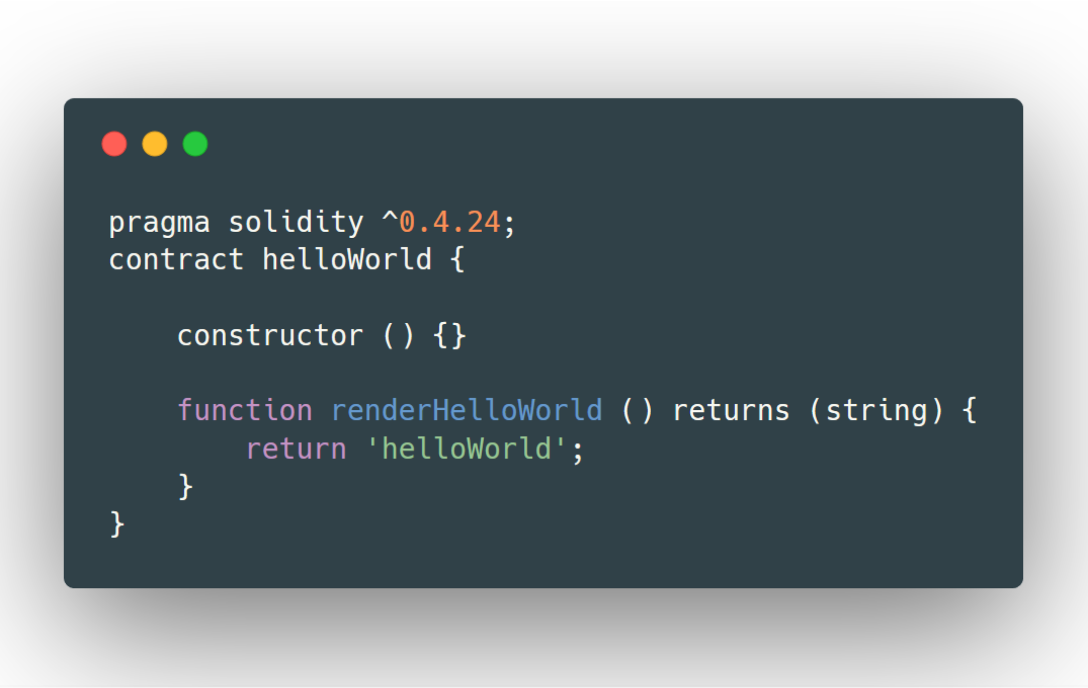
CC BY-SA 4.0

Brief insight: Solidity

Usually, smart contracts are not written as a sequence of opcodes instructions directly. **Solidity** is a high-level language with a JavaScript-like syntax and the de facto **standard** for writing **Ethereum smart contracts**. However, unlike JavaScript, Solidity is **statically typed**.

Language properties

- Statically typed
- Object-oriented
- Supports inheritance
- Complex, user-defined types
- Public & private methods
- Dynamic binding
- Compiled to EVM opcode instructions



```
pragma solidity ^0.4.24;
contract helloWorld {

    constructor () {}

    function renderHelloWorld () returns (string) {
        return 'helloWorld';
    }
}
```

CC BY-SA 4.0

Use case examples for smart contracts

Token systems
Token Systems are currently the largest use case for smart contracts, mostly used to collect money via initial coin offerings (ICOs). Usually, tokens work as a sub-currency of Ethereum and represent a certain asset such as a stock.

Identity and reputation systems
Smart contracts can be used as a decentralized identity management system like uPort.

Decentralized Autonomous Organization (DAO)
DAOs are basically a generalization of multi-signature wallets. The members vote to trigger certain methods in the smart contract like the transfer of money.

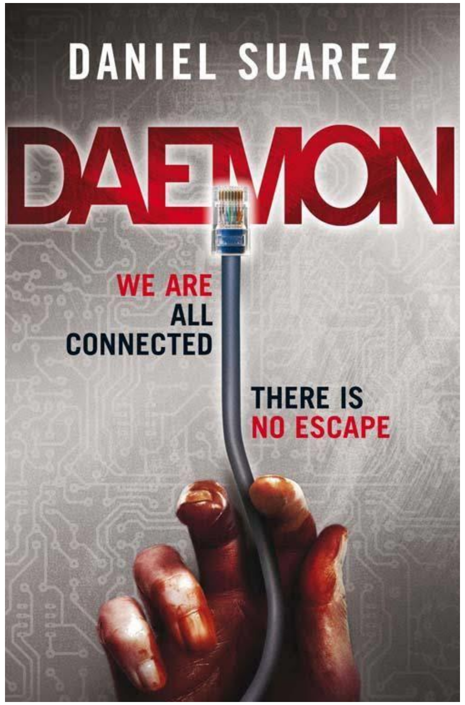
Election and voting systems
The blockchain provides a tamper-proof data structure for storing votes. A smart contract can ensure that a specific wallet can only vote once.

CC BY-SA 4.0

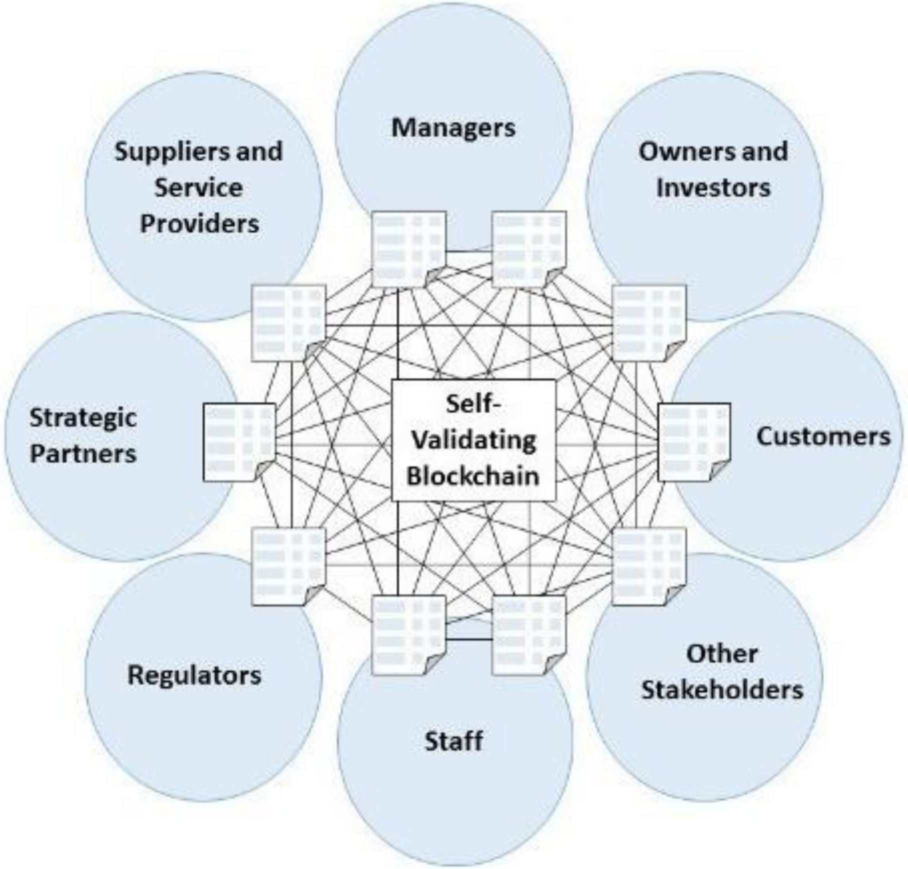
Decentralized Autonomous Organizations (DAOs)

Vision

- Create a fully digital (virtual) organisation.
- The organisation exclusively uses Smart Contracts to interact with its shareholders, employees, customers, suppliers, partners and public authorities.
- These stakeholders can be humans or organizations in the “real world” or other DAOs.



JOACHIM RADKAU
GESCHICHTE DER ZUKUNFT
Prognosen, Visionen, Irrungen in Deutschland von 1945 bis heute
HANSER



CC BY-SA 4.0

Outline

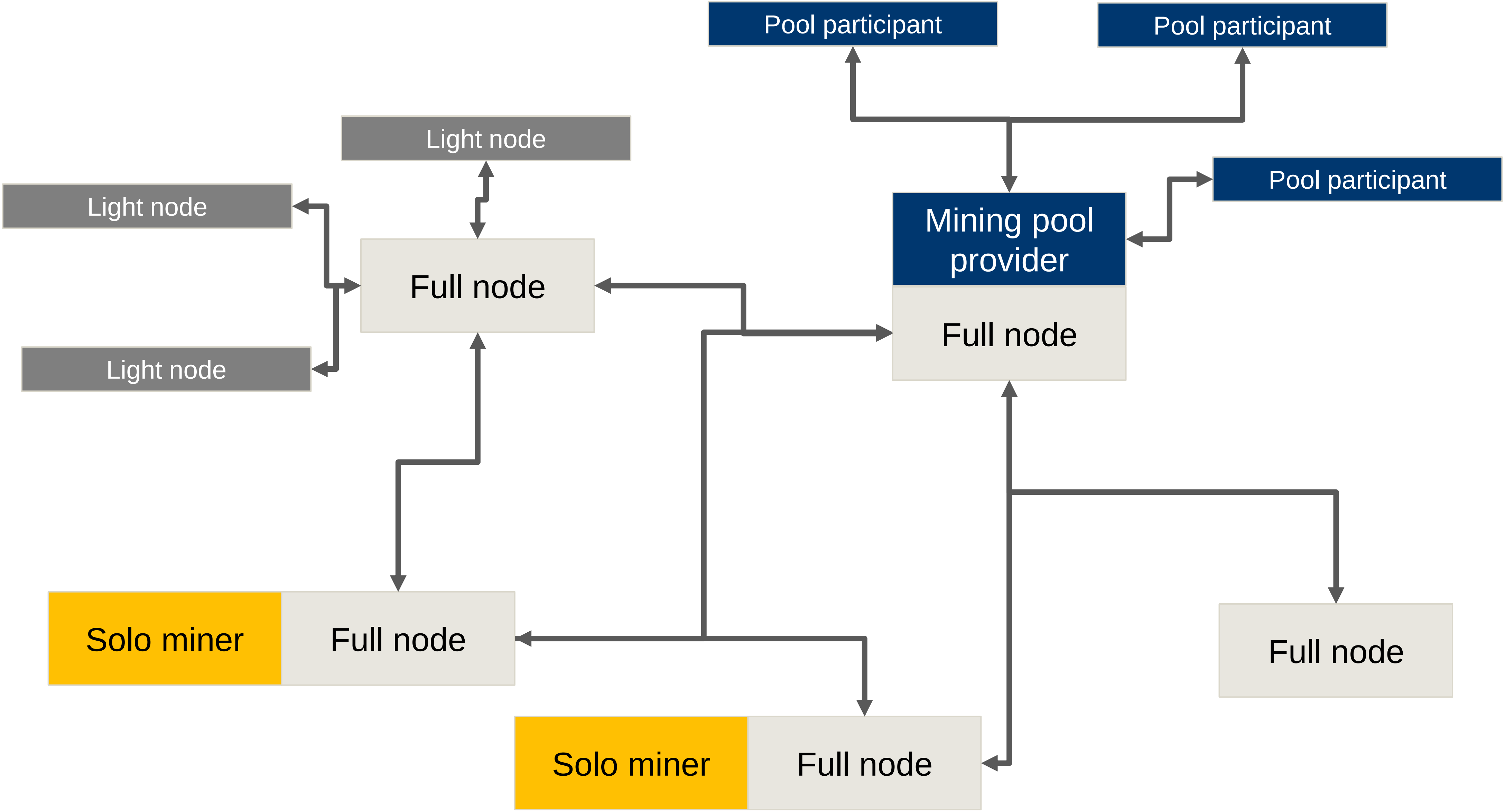
1. Ecosystem
   - Historical overview
   - Crowdsale statistics
   - Technical papers
   - Foundations
   - Network metrics

2. System architecture
   - Concept of a world computer
   - EVM
   - Accounts
   - Blockchain properties
   - Smart contracts

3. Network architecture
   - Overview
   - Node types
   - Clients
     - Geth
     - OpenEthereum

CC BY-SA 4.0

Network architecture overview



CC BY-SA 4.0

Node types

Full nodes
Full nodes are the foundation of the Ethereum network. Each full node holds a copy of the entire blockchain and syncs it with other nodes. Transactions must be sent to a full node which distributes it among the network participants.

Light nodes
A light node is a client that is connected to a full node for the sake of not having to sync and download the entire blockchain. For most private people, light nodes are the most comfortable way of interacting with the Ethereum blockchain. One of the most common light nodes is https://myetherwallet.com (always triple check the domain).

Solo miner
A solo miner is an entity that tries to mine a block on its own. At the current network hashing rate this is practically impossible. However, in order to mine a block it is required to have a synced copy of the full blockchain.

Mining pools
Mining pools are a coalition of entities combining their hash power to solve a mining problem. A pool consists of a **controller** that **splits** and **distributes** the **mining puzzle** among the participants.

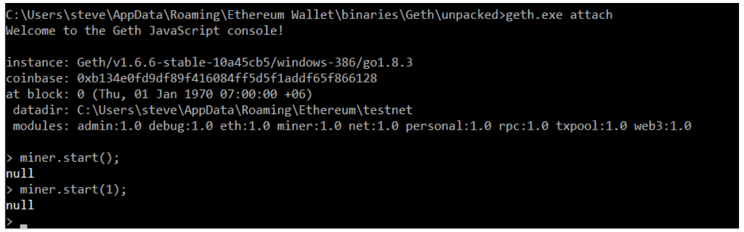
CC BY-SA 4.0

Popular Ethereum implementations (nodes with clients)

Not all Ethereum nodes are using the same code base. Since the specification for an Ethereum node is open source, basically anyone could create a different implementation. The two major Ethereum implementations are:

Geth

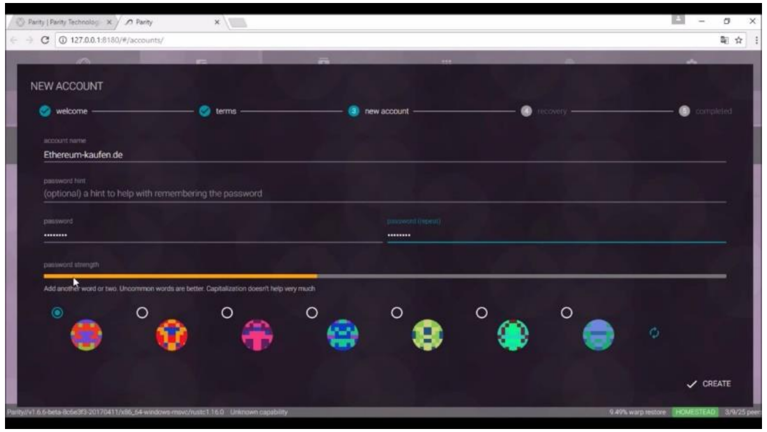
The most commonly used and **official Ethereum implementation**. Geth is implemented in Go and provides a command-line interface for running a full node. Geth comes with a JavaScript console and a JSON RPC server. Through which – if publicly exposed – other (light) nodes could connect to the network.



```
C:\Users\steve\AppData\Roaming\Ethereum Wallet\binaries\Geth\unpacked>geth.exe attach
Welcome to the Geth JavaScript console!

instance: Geth/v1.6.6-stable-10a45cb5/windows-386/go1.8.3
coinbase: 0xb134e0fd9df89f416084ff5d5f1addf65f866128
at block: 0 (Thu, 01 Jan 1970 07:00:00 +06)
 datadir: C:\Users\steve\AppData\Roaming\Ethereum\testnet
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

> miner.start();
null
> miner.start(1);
null
> 
```

OpenEthereum (former: Parity)

OpenEthereum is another popular Ethereum implementation in Rust with the goal to be “*the fastest, lightest, and most secure Ethereum client*“. OpenEthereum ships with a browser-based UI which is considered as a very user-friendly way to interact with Ethereum. However, the multi-signature wallet used by OpenEthereum was responsible for the *biggest (based on USD) hack in Ethereum’s history*. The company Parity transitioned the Parity codebase and maintenance to a DAO to allow for a continued development.



NEW ACCOUNT

account name
Ethereum-kaufen.de

password hint
(optional) a hint to help with remembering the password

password
•••••••••

password (repeat)
•••••••••

password strength
Add another word or two. Uncommon words are better. Capitalization doesn't help very much

Parity//v1.6.6-beta-8c6e3f3-20170411/x86_64-windows-msvc/rustc1.16.0   Unknown capability   9.49% warp restore   HOMESTEAD   3/9/25 peers

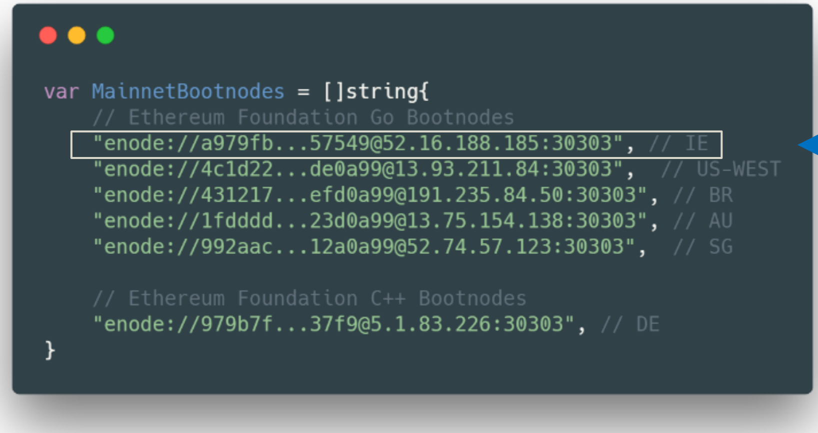
The Geth node is available at https://geth.ethereum.org/
The OpenEthereum node is available at https://github.com/openethereum/openethereum
Announcement by Parity Technologies: Transitioning Parity Ethereum to OpenEthereum DAO

CC BY-SA 4.0

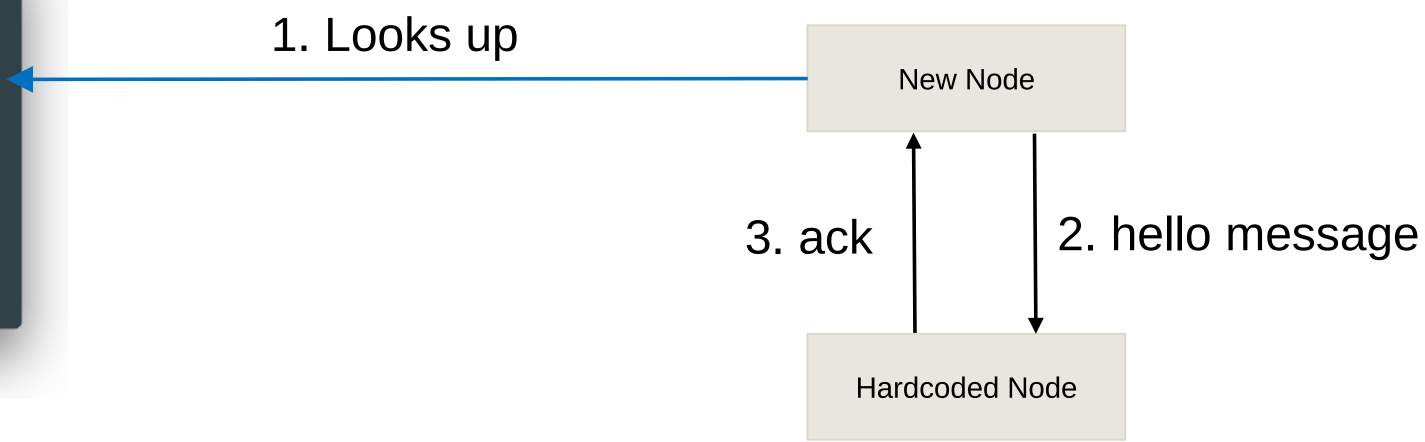
Peer discovery

Both Geth and OpenEthereum have a maintained list of default peers hardcoded into their source code. Otherwise, it would be possible that no nodes are found, and the sync will always fail.

The Geth client comes with 6 hardcoded peers:



```
var MainnetBootnodes = []string{
    // Ethereum Foundation Go Bootnodes
    "enode://a979fb...57549@52.16.188.185:30303", // IE
    "enode://4c1d22...de0a99@13.93.211.84:30303",  // US-WEST
    "enode://431217...efd0a99@191.235.84.50:30303", // BR
    "enode://1fdddd...23d0a99@13.75.154.138:30303", // AU
    "enode://992aac...12a0a99@52.74.57.123:30303",  // SG

    // Ethereum Foundation C++ Bootnodes
    "enode://979b7f...37f9@5.1.83.226:30303", // DE
}
```



Once a node is selected, a hello message is sent to make an initial connection with the node.

CC BY-SA 4.0