Emerging Technologies for the Circular Economy

Lecture 09: Ethereum and Smart Contracts Part 1

Prof. Dr. Benjamin Leiding (Clausthal)
M.Sc. Arne Bochem (Göttingen)
M.Sc. Anant Sujatanagarjuna (Clausthal)

License

- Updated versions of these slides will be available in our Github repository.

NEWS/UPDATES

Course Evaluation – QR Code and Link

- Link: Click Me

Disclaimer and Further Resources

- The lecture slides (Figures are often copied directly) are based on the course Blockchain-based Systems Engineering from TU Munich, which is distributed under a CC-BY-SA 4.0 license
- All their slides, exercises and further information are available online: https://github.com/sebischair/bbse

INTRODUCTION TO ETHEREUM

History

- Publicly announced in January 2014 by Vitalik Buterin
- Public crowd sale in July 2014
  - 60 million Ether sold for 31,591 Bitcoin
  - Worth around 18.5 million USD at that time
  - Funds controlled by the Ethereum foundation

“Founder of Ethereum Vitalik Buterin during TechCrunch Disrupt London 2015 - Day 2 at Copper Box Arena on December 8, 2015 in London, England.” by John philps is licensed under CC BY 2.0.

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

Ethereum 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.”*

Ethereum Foundation

- Founded in June 2014 in Zug, Switzerland
- Non-profit organization
- Foundation council consists of Vitalik Buterin and Patrick Storchenegger (legal affairs)
- Owns (or owned) at least 31,591 Bitcoins funding capital from the initial crowd sale

Ethereum White Paper

- 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
- Extensive summary of the Ethereum platform and technology
- Most current version is available via the public Git-repository: Link

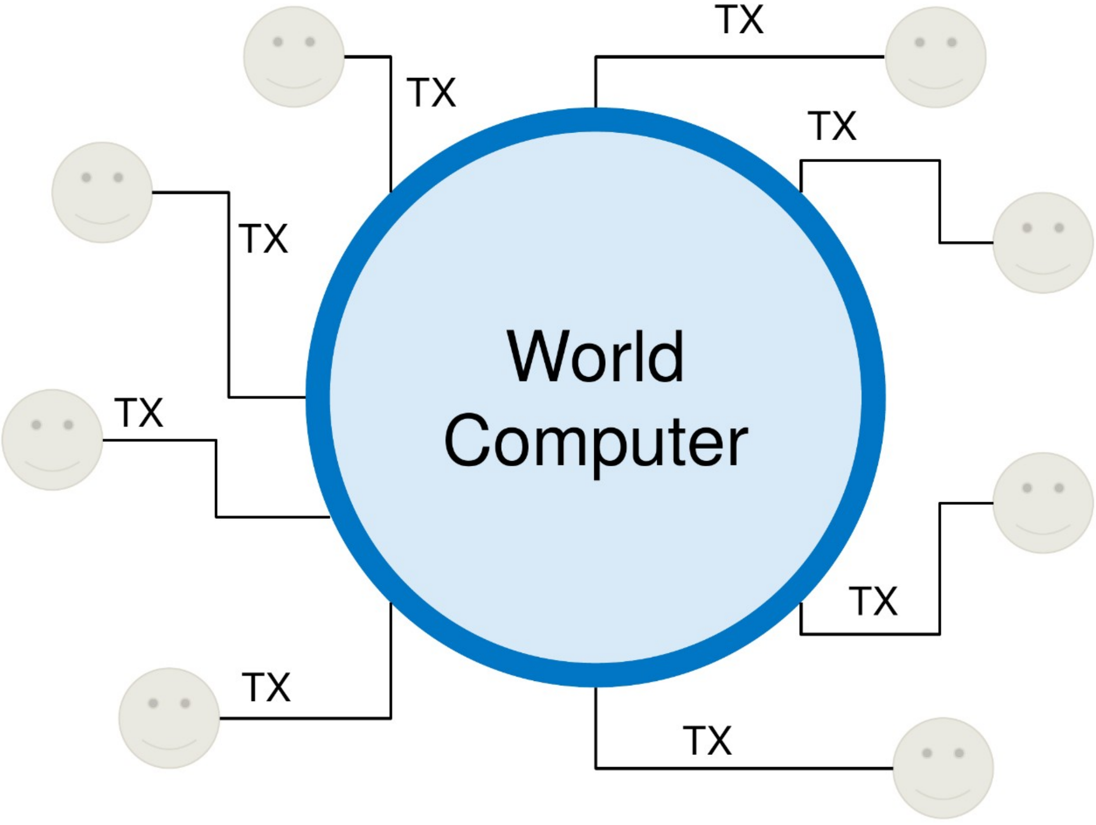
Ethereum Yellow Paper

- 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
- Yellow Paper

ETHEREUM SYSTEM ARCHITECTURE

The Concept of a World Computer

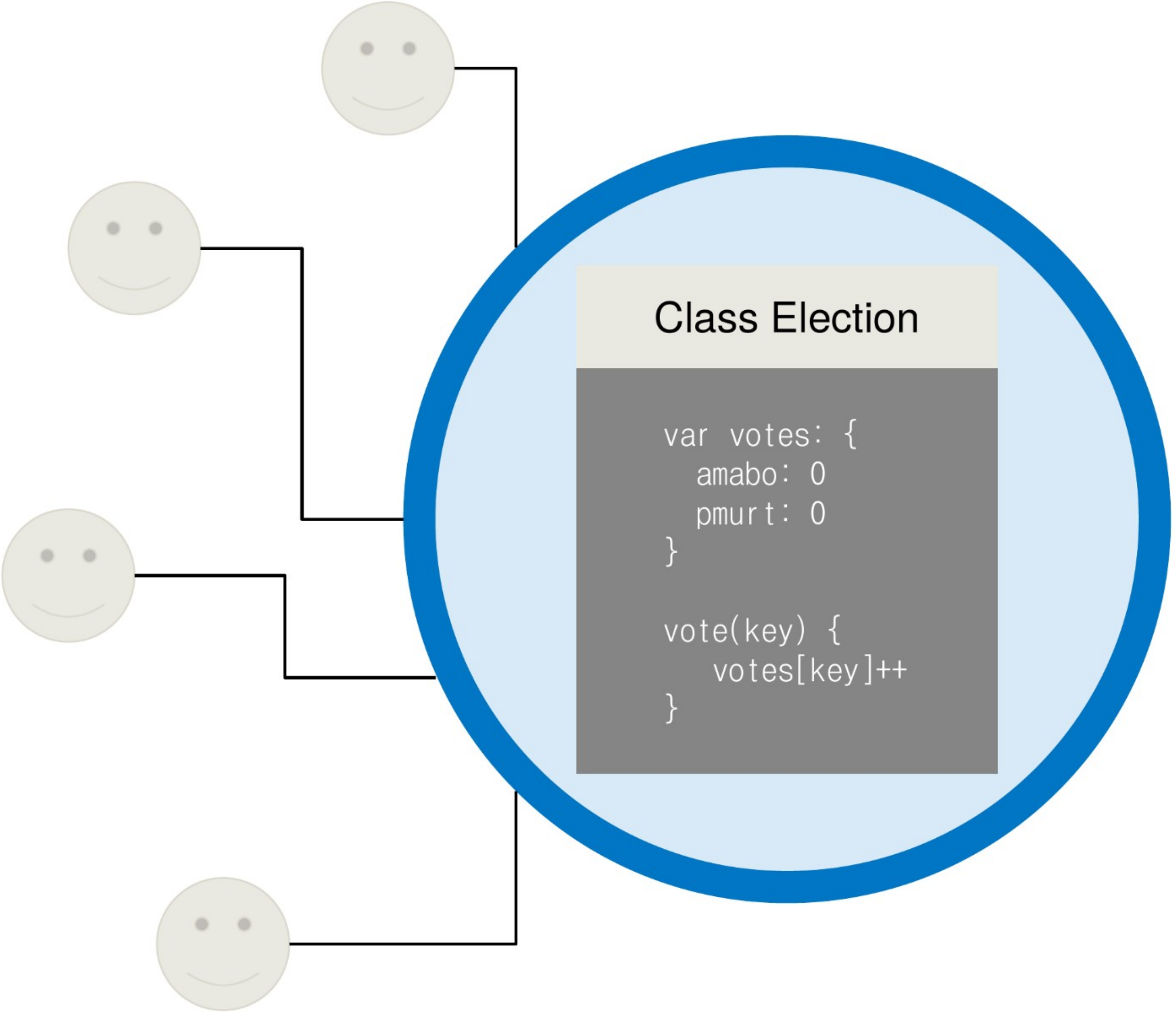
- All participants are using the same computer
- Users issue transactions to call programs on the computer
- Everyone shares the same resources and storage
- The computer has no explicit, single owner
- Using the computer’s resources costs money



TX = Transaction

U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

Elections using a World Computer



Class Election

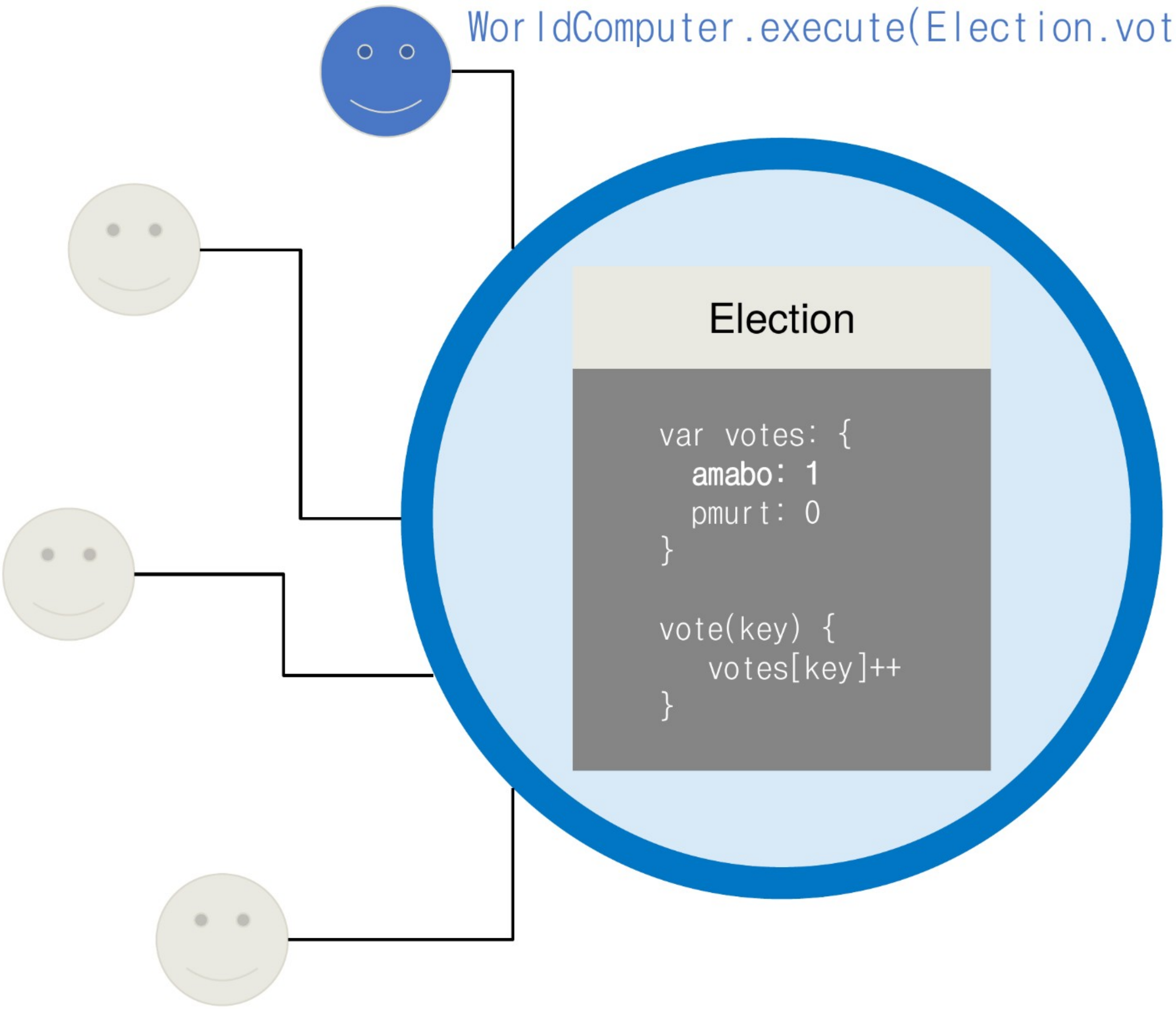
```
var votes: {
  amabo: 0
  pmurt: 0
}

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

State of the world

State 0 (initial)
(nothing happened yet)

U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

Elections using a World Computer

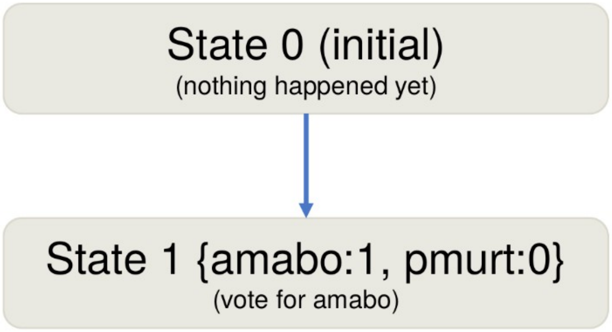


```
WorldComputer.execute(Election.vote("amabo"))
```

Election

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

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

State of the world

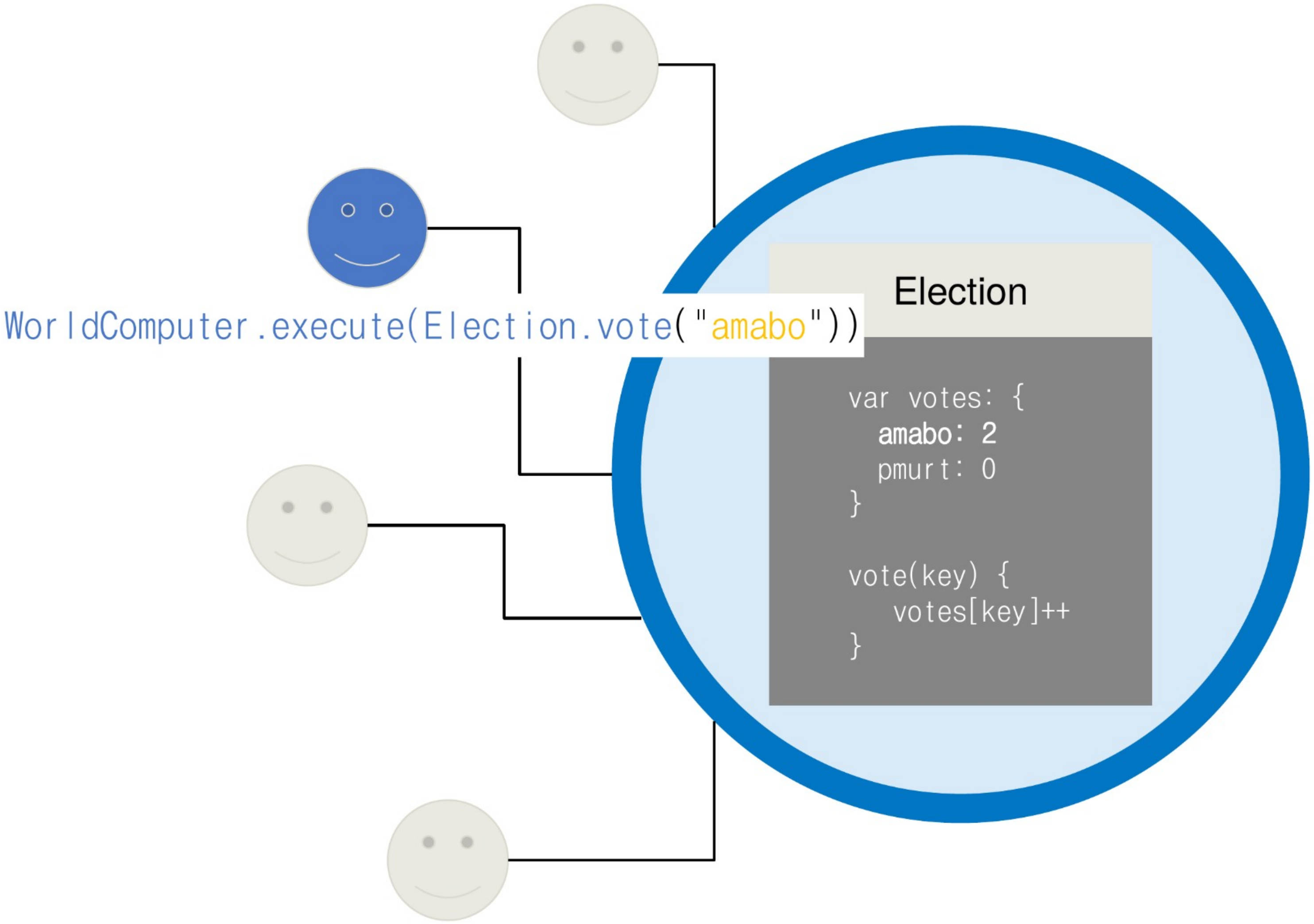


State 0 (initial)
(nothing happened yet)

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

U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

Elections using a World Computer



```
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)

↓

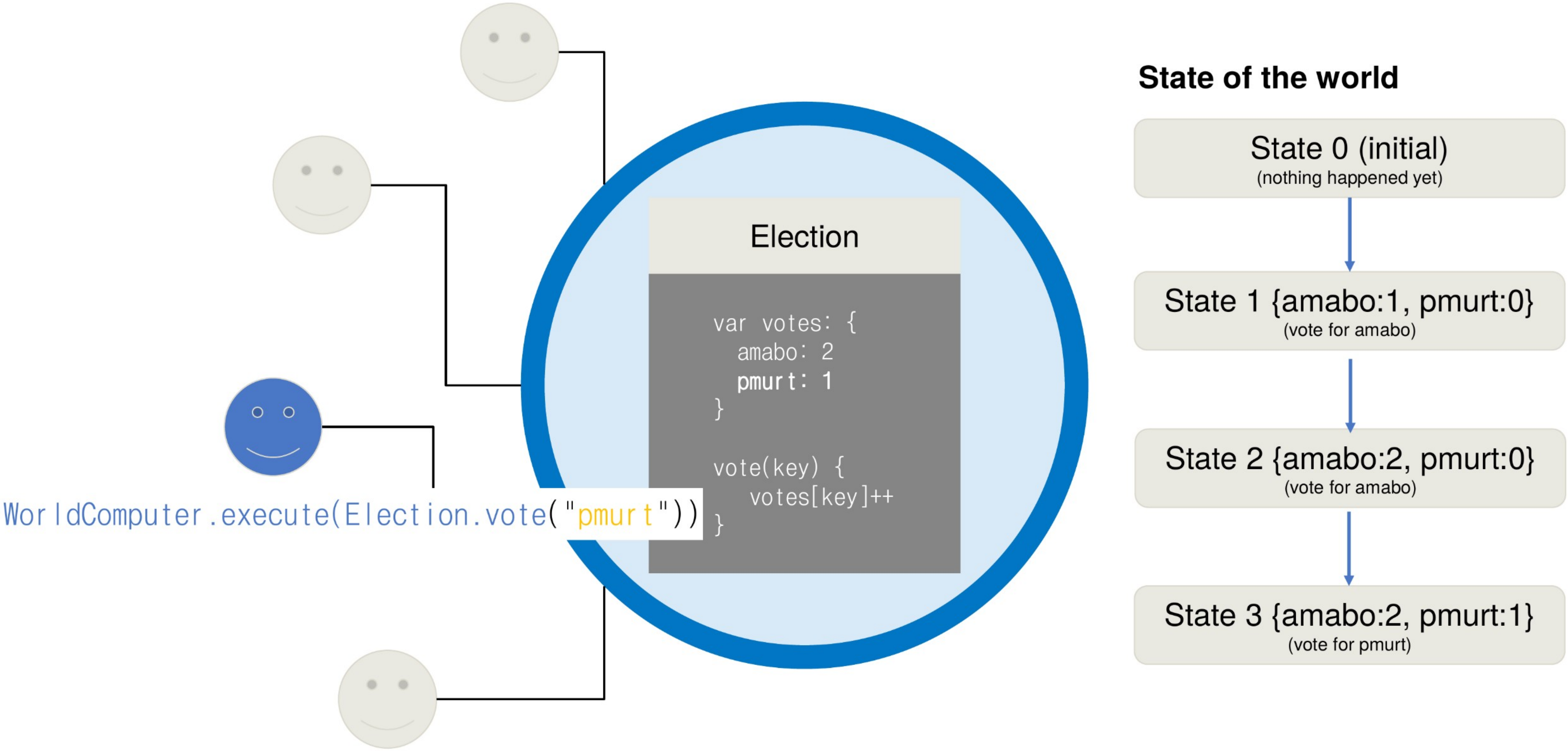
State 1 {amabo:1, pmurt:0}
(vote for amabo)

↓

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

U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

Elections using a World Computer



```
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)

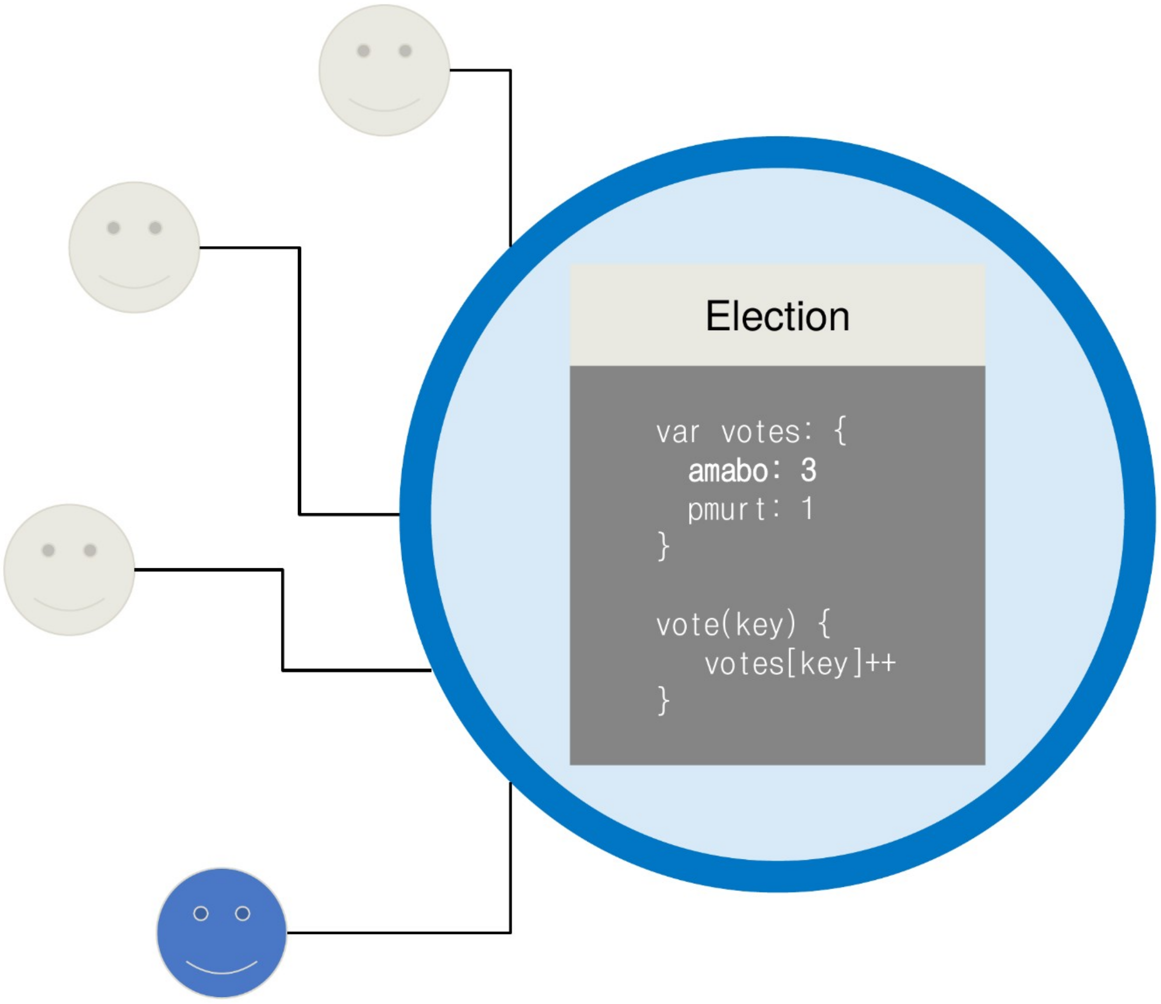
State 1 {amabo:1, pmurt:0}
(vote for amabo)

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

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

U. Gallersdörfer, P. Holl, and F. Matthes, "Blockchain-based Systems Engineering – Lecture Slides," Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

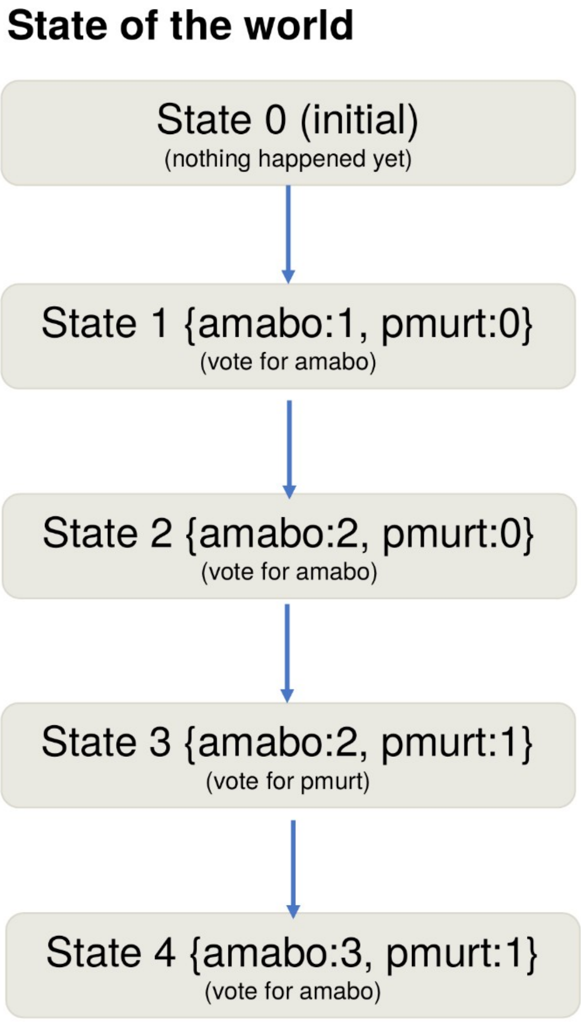
Elections using a World Computer



Election

```
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)

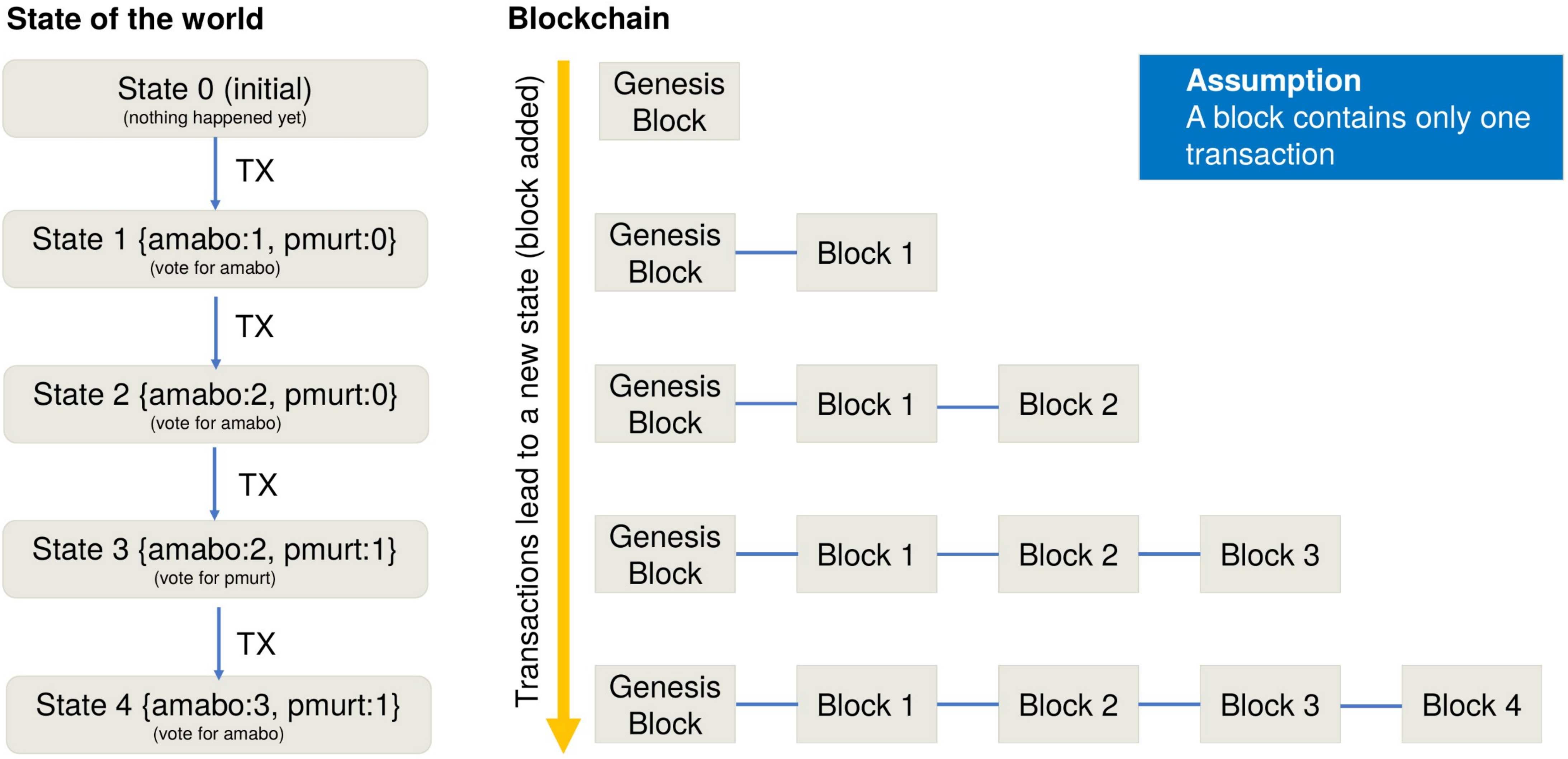
State 2 {amabo:2, pmurt:0}
(vote for amabo)

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

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

U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

The Blockchain State Machine

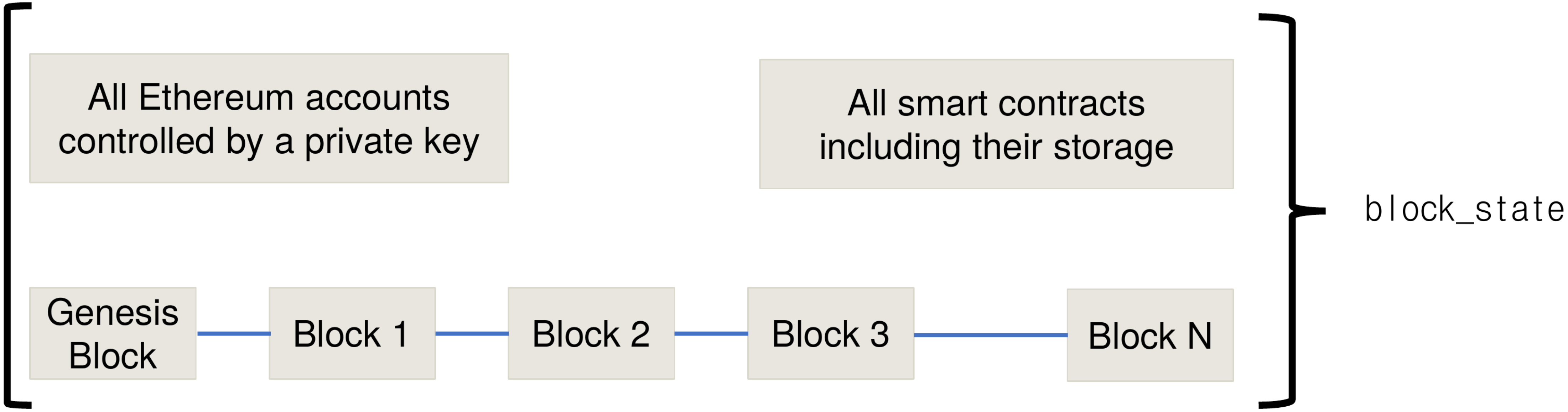


Assumption
A block contains only one transaction

U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

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

U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

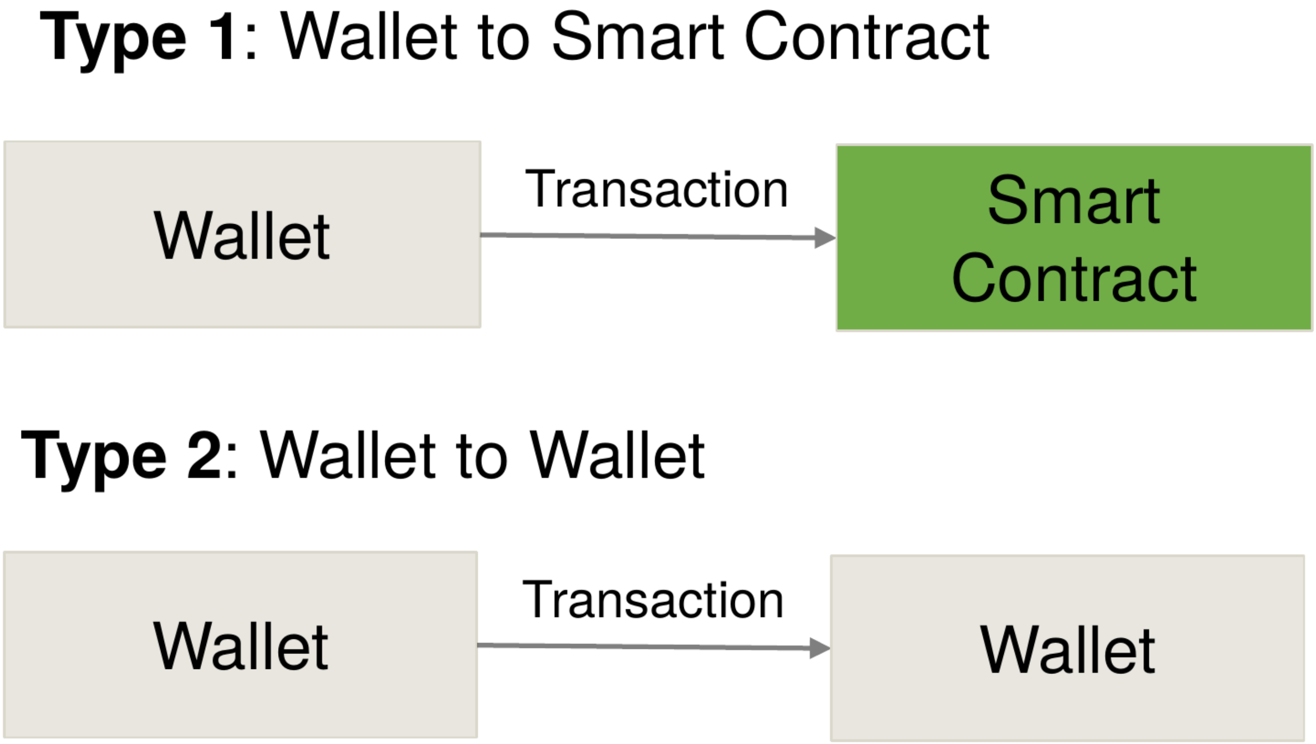
Transaction

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

- Recipient
- Sender’s signature
- Amount of ETH to transfer
- 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

Transactions

- There are two types of transactions:



Type 1: Wallet to Smart Contract

Wallet → Transaction → Smart Contract

Type 2: Wallet to Wallet

Wallet → Transaction → Wallet

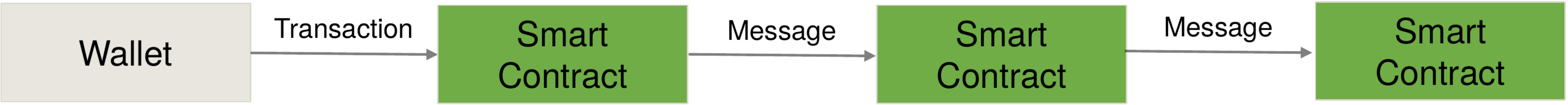
U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

Messages

- Similar to a transaction, but only sent by contracts and exist only virtually, i.e. they are not mined into a block like transactions
- Message contains:
  - Sender of the message (implicit)
  - Recipient
  - Amount of ETH to be transferred
  - Optional data field
  - *STARTGAS* value

Messages

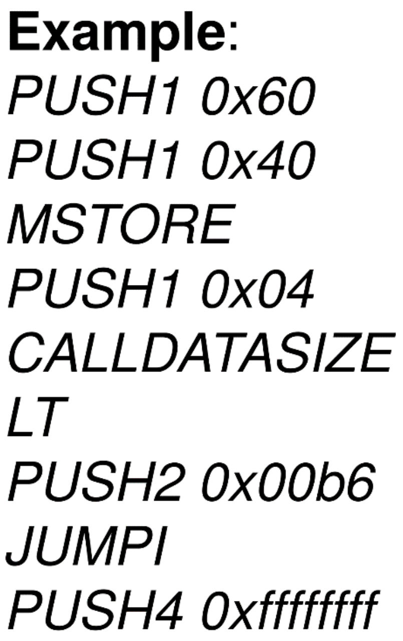
- Whenever a contract calls a method on another contract, a virtual message is sent.
- Whenever a wallet calls a method on a contract, a transaction is sent.



U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

Code and Memory

- Code:
  - Bytecode representation of a smart contract. EVM interprets smart contracts as a sequence of opcodes similar to assembly code.



Example:

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

U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

Code and Memory

- Code:
  - Bytecode representation of a smart contract. EVM interprets smart contracts as a sequence of opcodes similar to assembly code.
- Memory:
  - An infinitely expandable byte array that is non-persistent and used as temporal storage during execution.

Example:

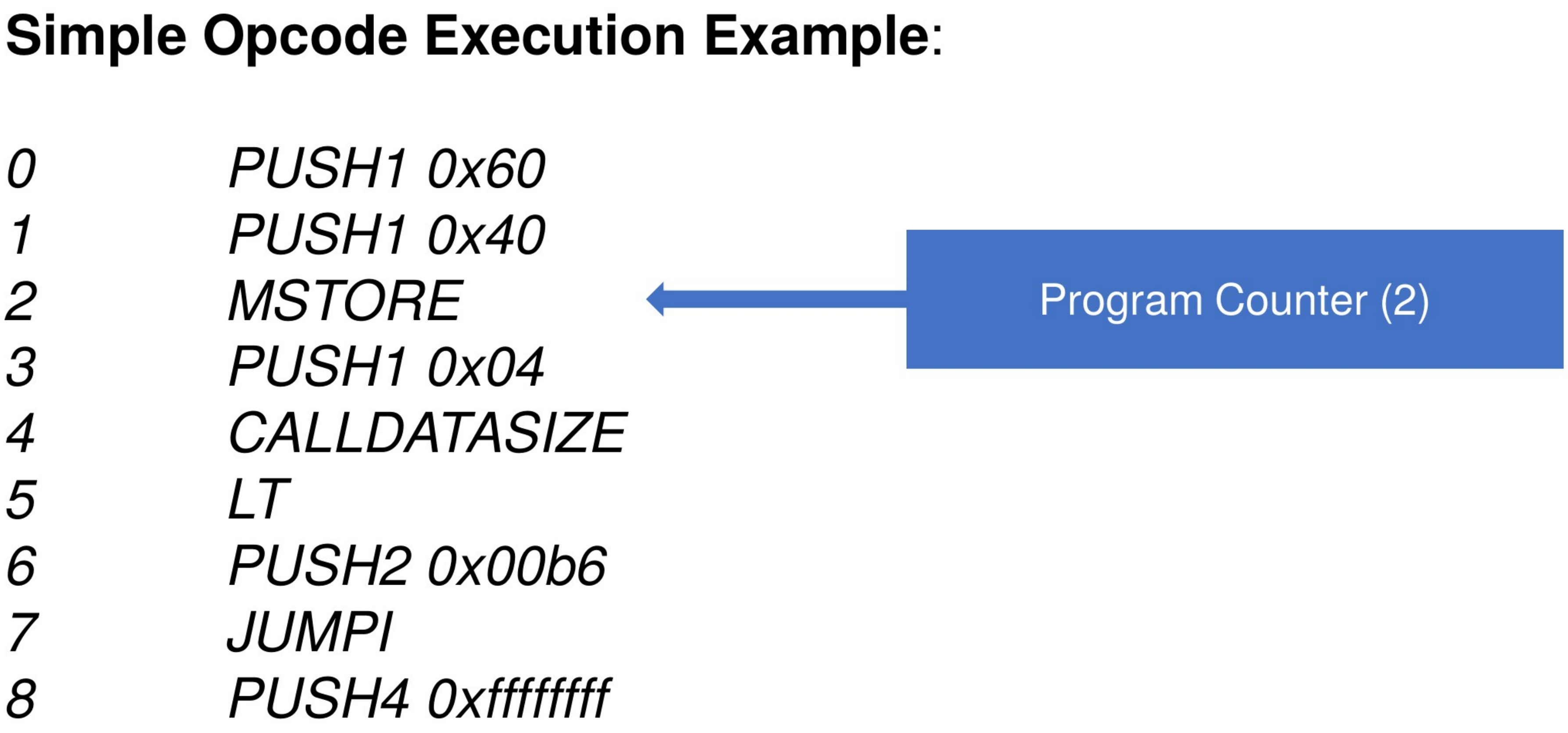
```
PUSH1 0x60
PUSH1 0x40
MSTORE
PUSH1 0x04
CALLDATASIZE
LT
PUSH2 0x00b6
JUMPI
PUSH4 0xffffffff
```

U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

Stack and 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:
  - Stands for “program counter”.
  - The program counter is always initialized with 0 and points to the position of the current opcode instruction.

Stack and Program Counter



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
```

U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

Gas

- Executed opcode instruction use miner’s computational resources → requires a fee (called gas)
- Each opcode uses a certain amount of gas which may depend on the arguments of the operation
- Opcode for self-destruct(address) uses negative gas because it frees up space from the blockchain
- Sender must specify maximum amount of gas that he/she/it is willing to pay for the transaction
- Sender can set an arbitrary amount of Ether to be spent → called gas price
- Final costs for transaction → gas × gasprice
- If a transaction requires more gas as the maximum specified gas, the transaction fails
- If it takes less, the sender only pays the gas that was used

Account Types

Ethereum uses an account-based ledger → Each distinct address represents a separate, unique account.

1. Accounts controlled by private keys and owned externally
   - Can sign transactions, issue smart contract functions calls and send Ether from one account to another
   - Origin of any transaction is always an account controlled by a private key
2. Smart contract accounts controlled by their code
   - Smart Contracts are treated as account entities with their own, unique address
   - Can send messages to other accounts, both externally controlled and smart contracts
   - Can’t issue a transaction themselves.
   - Have a persistent internal storage to write and read data from

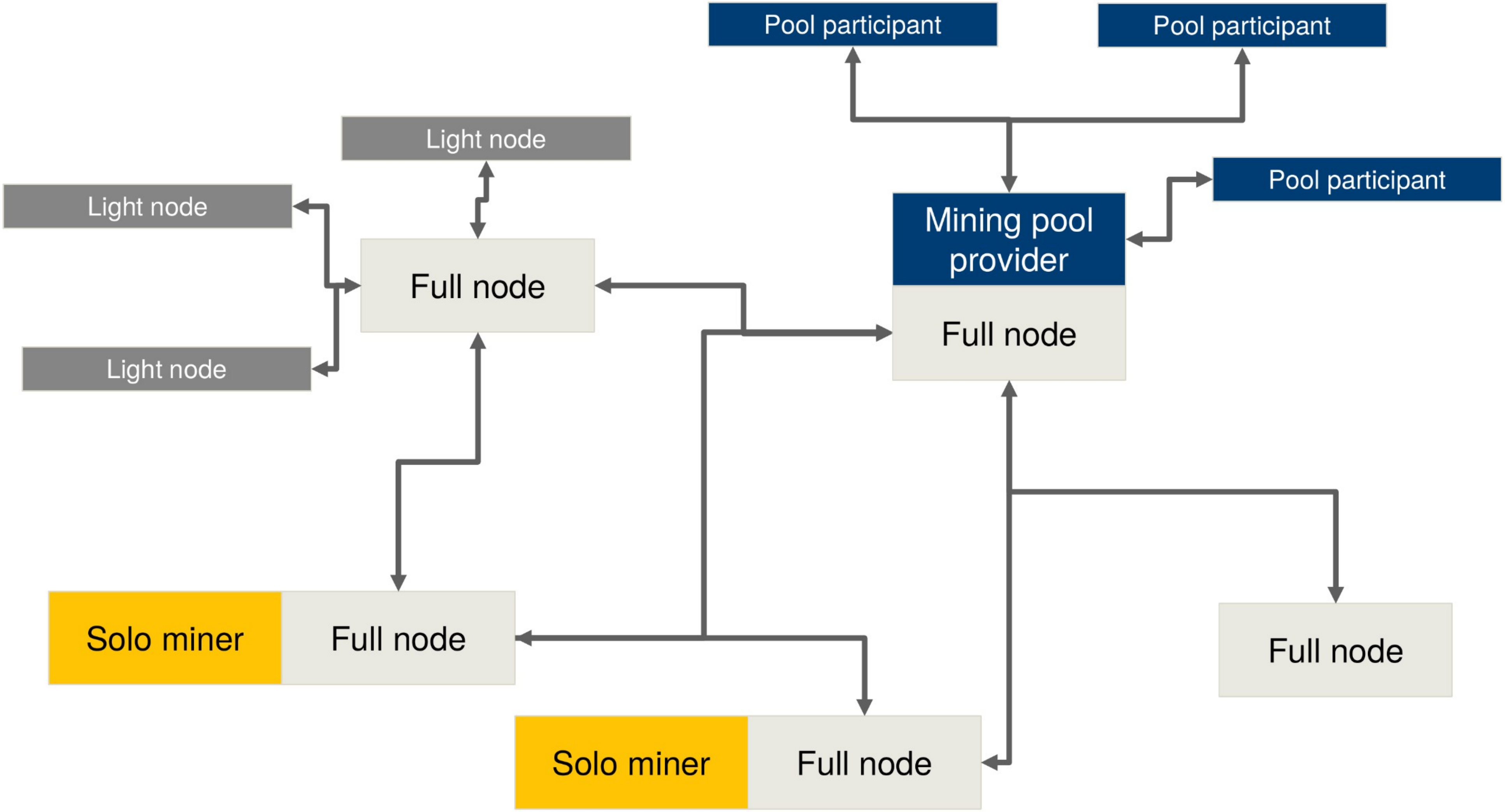
Account Properties

Ethereum account is a 4-tuple containing: (nonce, balance, contract code, storage)

- Nonce:
  - Increasing number that is attached to any transaction to prevent replay attacks and double spending.
- Balance:
  - Current account balance in ETH.
- Contract code:
  - Bytecode representation of the account. If no contract code is present, then the account is externally controlled.
- Storage:
  - Data storage used by the account – empty by default.
  - Only contract accounts can have their own storage.

ETHEREUM NETWORK ARCHITECTURE

Network Architecture Overview



U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

Node Types

- Full nodes:
  - Holds complete copy of the entire blockchain
  - P2P synchronization with other full nodes
  - Transactions must be sent to a full node to join the transaction pool
- Light nodes:
  - Clients connected to full nodes → instead of downloading the full blockchain on its own
  - Most commonly used by private users
- Solo miner:
  - An entity that tries to mine a block on its own
- Staking/Mining pools:
  - Coalition of entities combining their hash power / staking power
  - Revenue/Reward is shared among participants

ETHEREUM SMART CONTRACTS

What is a *Smart* Contract?

- A contract is a legal document
  - that binds two or more parties
  - who agree to execute a transaction immediately or in the future
- Smart contracts are digitization of legal contracts
- In Ethereum,
  - smart contracts are deployed, stored and executed within the Ethereum Virtual machine (EVM)

https://medium.com/coinmonks/https-medium-com-ritesh-modi-solidity-chapter1-63dfaff08a11

What is a *Smart* Contract?

- Also known as self executing contract or digital contracts
- It is like a vending machine, i.e. the ledger → You put money/data and you expect a finite item (e.g. your license, ..)
- Vitalik Buterin’s explanation:
  - An asset or currency is transferred into a program,
  - the program runs this code and at some point it automatically validates a condition
  - and it automatically determines whether the asset should go to one person or back to the other person,
  - or whether it should be immediately refunded to the person who sent it or some combination thereof.
  - In the meantime, the decentralized ledger also stores and replicates the document which gives it a certain security and immutability.

https://blockgeeks.com/guides/smart-contracts/

Smart Contract – Rent an Apartment

- You get a receipt which is held in our virtual contract;
- I give you the digital entry key which comes to you by a specified date
- If the key does not come on time, the blockchain releases a refund.
- If I send the key before the rental date, the function holds it releasing both the fee to me and key to you respectively when the date arrives.
- If I give you the key, I’m sure to be paid. If you send a certain amount in ETH, you receive the key.
- The document is automatically canceled after the time, and the code cannot be interfered with by either of us.

https://blockgeeks.com/guides/smart-contracts/

What is a *Smart* Contract?

- Smart contracts can also store data
- The data stored can be used to record
  - information, fact, associations, balances
  - or any other information needed to implement logic for real world contracts
- Smart contracts are similar to Object-oriented classes
  - A smart contract can call another smart contract just like an Object-oriented object to create and use objects of another class.
  - Think of smart contract as a small program consisting of functions.
  - You can create an instance of the contract and invoke functions to view and update contract data along with execution of some logic

https://medium.com/coinmonks/https-medium-com-ritesh-modi-solidity-chapter1-63dfaff08a11

Smart Contracts in a Nutshell

- Is a set of functions that can be called by other users or contracts
- Used to execute functions, send ETH, 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.

Smart Contracts in a Nutshell

- Is a set of functions that can be called by other users or contracts
- Used to execute functions, send ETH, 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.

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

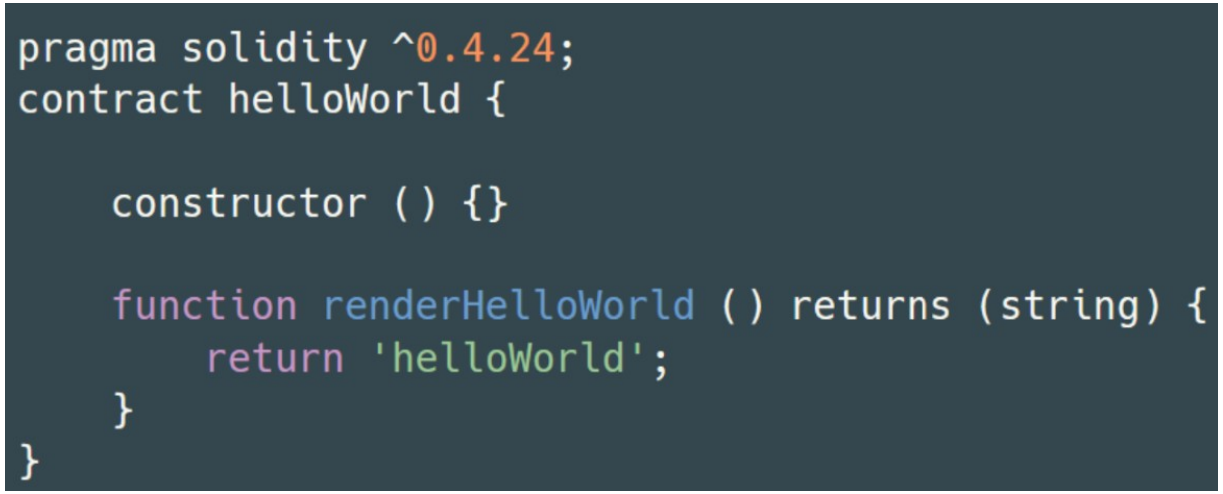
Ethereum Smart Contract Coding

- Solidity is a high-level language to write smart contracts for Ethereum
- Contracts can be defined as encapsulated units, similar to classes in traditional object-oriented programming languages like Java.
- A contract has its own, persistent state on the blockchain which is defined by state variables in the contract.
- Functions are used to change the state of the smart contract or to perform other computations.
- Solidity is compiled to bytecode which is persistent and immutable once deployed to the blockchain → Not patchable.

Brief Insight: Solidity

Solidity is a high-level language with a JavaScript-like syntax for writing Ethereum smart contracts.

- Language properties:
  - Statically typed
  - Object-oriented
  - Supports inheritance
  - Public & private methods
  - Dynamic binding
  - Compiles to EVM opcode instructions

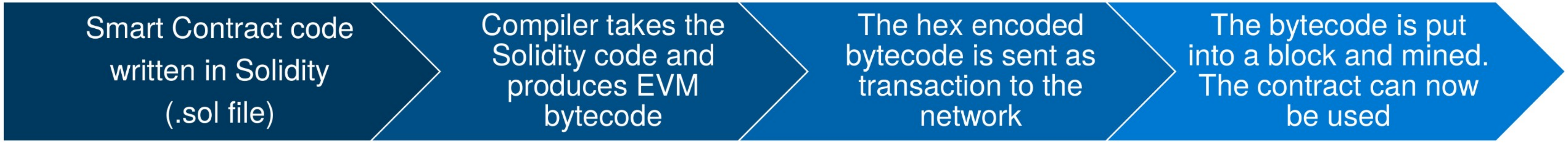


```
pragma solidity ^0.4.24;
contract helloWorld {

    constructor () {}

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

U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

Solidity to Smart Contract

- Solidity code is stored in files with the special file extension .sol
- A good practice is to have one separate .sol file per contract
- The Solidity compiler takes a .sol file as input and generates the corresponding sequence of EVM opcode instructions
- The opcode instructions are then encoded as hex bytecode
- The contract is deployed via a special transaction containing the bytecode as payload
- Once the transaction is mined, a new contract account on the Ethereum network is created
- The contract is now usable



Smart Contract code written in Solidity (.sol file)

Compiler takes the Solidity code and produces EVM bytecode

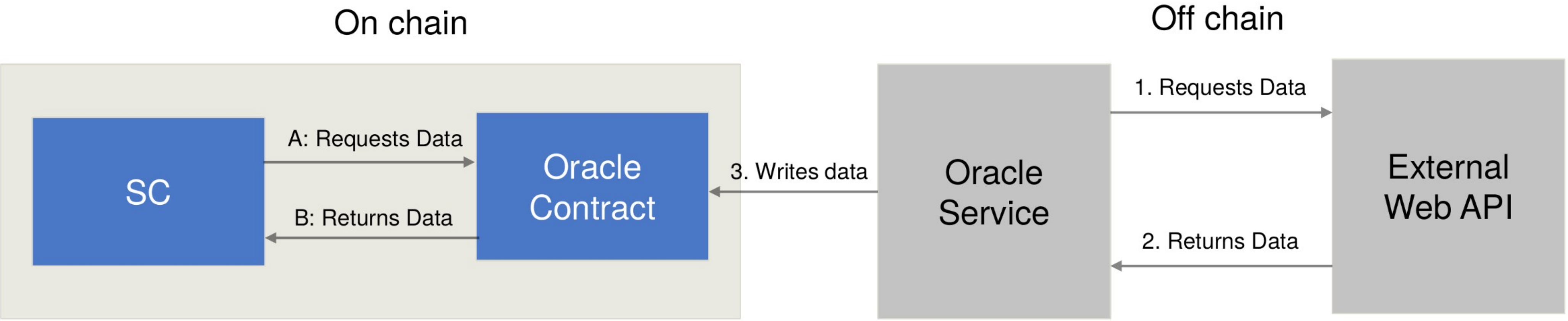
The hex encoded bytecode is sent as transaction to the network

The bytecode is put into a block and mined. The contract can now be used

U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

Oracles

- Smart contracts cannot access any data from outside the blockchain on their own → on purpose to prevent non-deterministic behavior
- Also no functions to generate random values
- Solution → Oracles
  - 3rd-party services that verify data from web services and write the data via a smart contract to the chain



U. Gallersdörfer, P. Holl, and F. Matthes, “Blockchain-based Systems Engineering – Lecture Slides,” Oct 2019. [Online]. Available: https://github.com/sebischair/bbse

Questions?