Ethereum

Introduction

Leander Jehl

Ethereum
Overview

Ethereum 1 (deprecated) was a Proof of Work blockchain that used several of the improvements discussed previously.

- Ethereum uses 12 sec block delay.
- Different P2P network.
- Ethereum uses a different Proof of Work function to protect agains ASICs.
- Ethereum uses uncles.
- Ethereum uses the GHOST rule, instead of longest chain rule.

Similar to Bitcoin, Ethereum uses *hashes of a public key as address*, and signatures for authentication.

Ethereum has a cryptocurrency, Ether.

Ethereum
Overview

Ethereum 2 is a Proof of Stake blockchain (the merge - 2022).

- Ethereum 2 is much more resource friendly
- Ethereum 2 is More efficient: it can handle more than 100,000 transactions per second (15 before)
- Ethereum 2 uses a committee of validators to confirm blocks
- Ethereum 2 has single slot finality

Similar to Bitcoin, Ethereum uses *hashes of a public key as address*, and signatures for authentication.

Ethereum has a cryptocurrency, Ether.

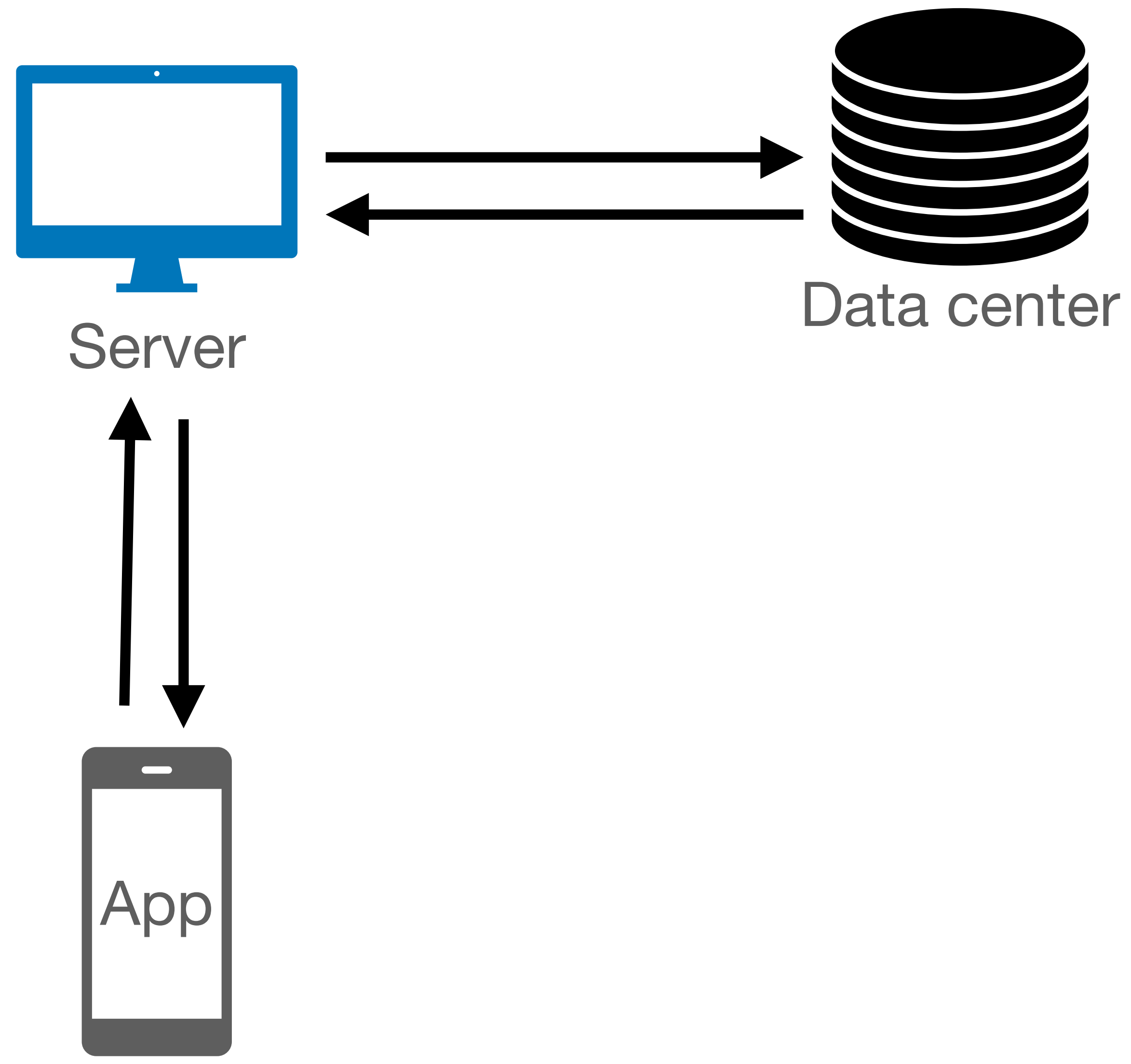
What are Smart Contracts

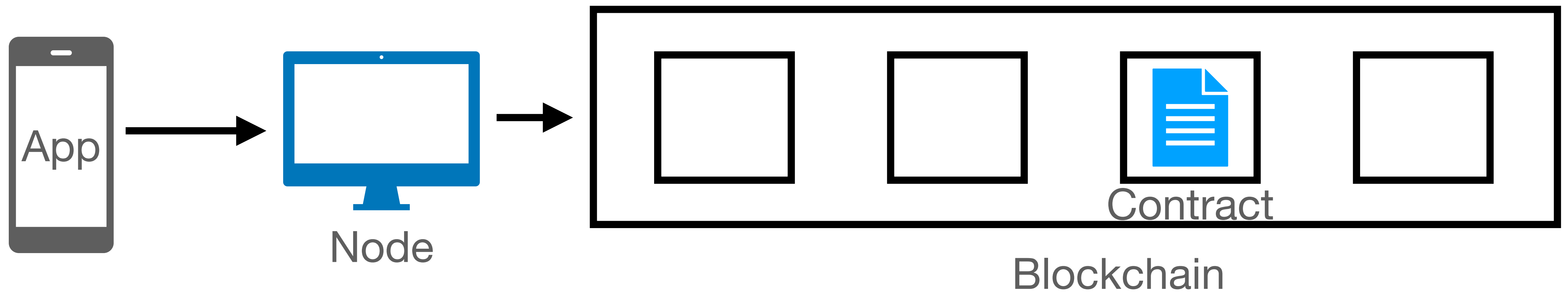
Ethereum

Smart Contracts

Smart contracts are codes stored on a blockchain that run when predefined conditions are met

- No central server, no need to trust any entity





Bitcoin scripts scripts

Spending conditions

Transactions:

$$tx = \langle \underbrace{[(id_1, rd_1), (id_2, rd_2)]}_{\text{Inputs}}, \underbrace{[(s_a, value_a), (s_b, value_b)]}_{\text{Outputs}} \rangle$$

- s_a a **spending condition**: output can be used if a value is supplied, that evaluates s_a to `true`
- rd_1 a **redeeming argument**:
  should ensure the script s_{id_1} returns `true`

UTXO scripts

Examples

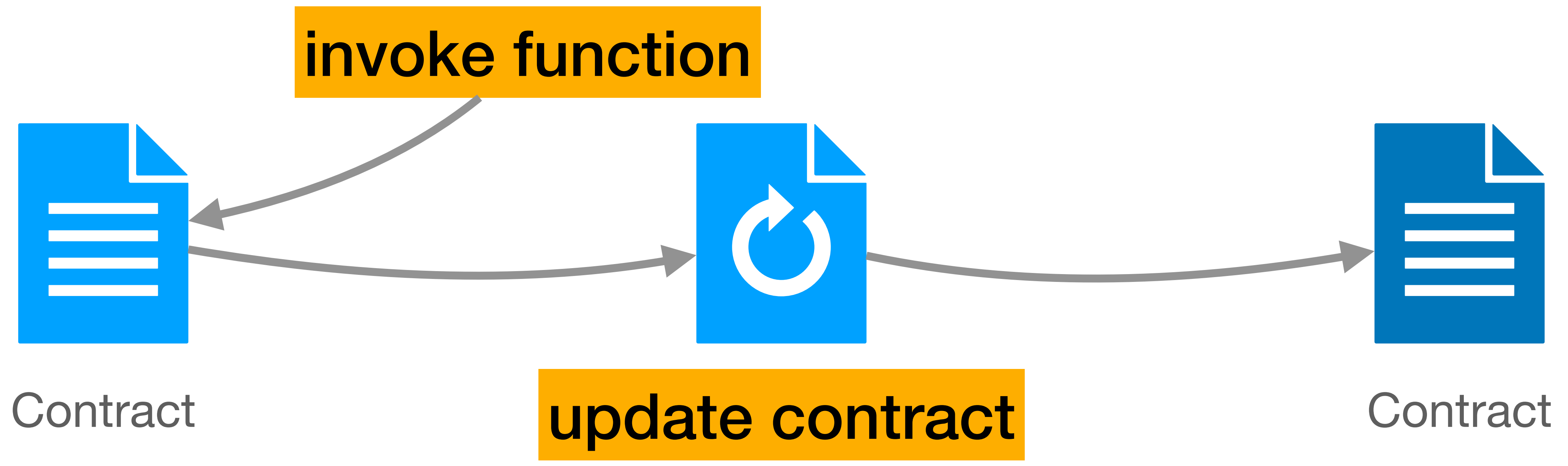
Name	Spending condition	Redeeming argument	
P2Pk Pay to public key	Public key	Signature	
P2PkH Pay to public key hash	Hash of Public key	Public key and signature	
Multisig	*m* public keys and parameter *k*	*k signatures*	

Ethereum

Smart Contracts

In ethereum, a contract is like a object from OOP, with fields and methods

- variables containing state (stored in account, mutable)
- functions



Ethereum

Example: Simple Storage

compiler version

```
pragma solidity ^0.5.11;
```

contract

```
contract SimpleStorage {
    uint256 public storedData;

    function get() public view returns (uint256){
        return storedData;
    }

    function set(uint x_) public {
        storedData = x_;
    }
}
```

state

functions

Ethereum

Example: Simple Storage

Simple online IDE: https://remix.ethereum.org/

Fun tutorial: https://cryptozombies.io/

- Constructors
- Basic types and collections
- Visibility (private, public)
- Inheritance
- Modifiers (view, pure)

Ethereum

Example: Simple Storage

Who can invoke functions?

- any user

Who can view values?

- anyone

Who can change the code?

- noone

```
pragma solidity ^0.5.11;

contract SimpleStorage {
    uint256 public storedData;

    function get() public view returns (uint256){
        return storedData;
    }

    function set(uint x_) public {
        storedData = x_;
    }
}
```

Ethereum

Smart Contract code

Smart contract code is immutable and public

- Anyone can trust smart contract (if it is not too complex)
  - No need to trust the creator of the contract
- No one can fix bugs in the contract
- Anyone can find and exploit bugs in the contract

Ethereum

Smart Contract code

- Assembly for Ethereum Virtual machine (EVM)
- Compiled from higher level language (Solidity)
- Stored in account (codeHash)

How does Ethereum enable Smart Contracts

Ethereum

Accounts

Ethereum uses accounts instead of UTXO.
Thus the state of Ethereum contains for every account:

- **address:** e.g. pub-key hash
- **balance**: amount of Ether the address owns
- **nonce**: sequence number of last transaction sent from this account
- **storage root**: *only for non-user accounts (contract account)*
- **code hash:** *only for non-user accounts (contract account)*

Ethereum

Accounts

Smart Contracts are also represented as accounts.
A contract account has:

- **address:** *e.g. hash from creator address & creation trancation nonce*
- **balance**: amount of Ether the address owns
- **nonce**: number of other contract created by this contract
- **storageRoot**: hash of data stored in this contract
- **codeHash:** hash of the code of this contract

Ethereum

Accounts

In a contract written in Solidity, you can access:

- The address of the current contract:

```
address contractaddress = address(this);
```

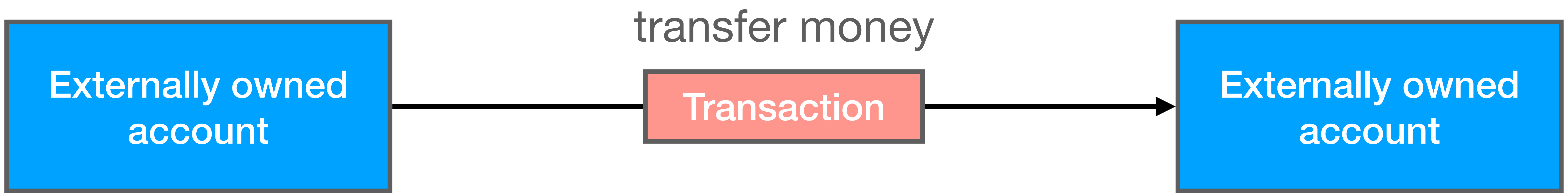
- The balance of the contract:

```
uint balance = contractaddress.balance;
```

Ethereum

Transactions and authenticaion

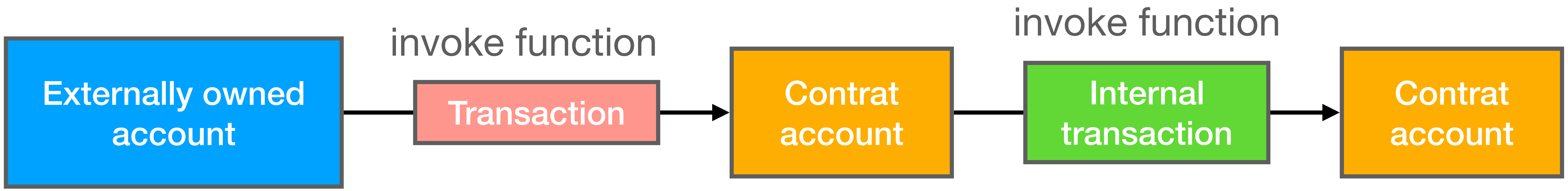
Transactions are used to transfer ether, invoke functions, and deploy new contracts.



Ethereum

Transactions and authenticaion

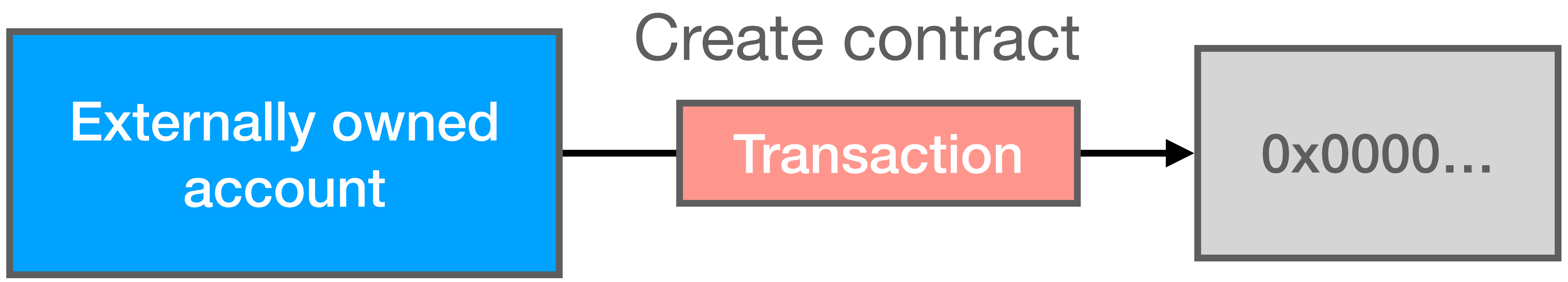
Transactions are used to transfer ether, invoke functions, and deploy new contracts.



Ethereum

Transactions and authenticaion

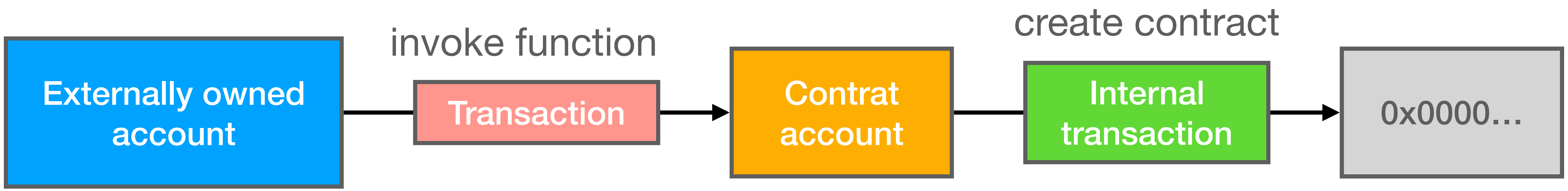
Transactions are used to transfer ether, invoke functions, and deploy new contracts.



Ethereum

Transactions and authenticaion

Transactions are used to transfer ether, invoke functions, and deploy new contracts.



Ethereum

Transactions and authenticaion

Transactions contain:

- *Nonce:* next sequence number for sender account
- *Gas price:* later
- *max gas:* later
- *Recipient:* destination Ethereum address
- *Value:* Amount of ether send to destination
- *Data:* Payload binay, e.g. function identifier and arguments
- *Signature:* Signature from sender, including his public key

Ethereum

Transaction validation

Transaction validation includes the following checks

- *Nonce:* is next sequence number for sender account
- Sender has sufficient balance to pay value and fees
- Transaction is correctly signed

When autheticating users in smart contract, we can rely on transaction validation!
Use *msg.sender* to access address invoking transaction.

Ethereum

Solidity example

```
contract SimpleBank {
    mapping(address => uint) private balances;
    address public owner;

    // function SimpleBank() deprecated syntax for
    constructor() public {
        owner = msg.sender;
    }

    function deposit() public payable returns(uint) {
        balances[msg.sender] += msg.value;
        return balances[msg.sender];
    }

    function withdraw(uint withdrawAmount) public returns (uint remainingBal){
        if (balances[msg.sender] >= withdrawAmount){
            balances[msg.sender] -= withdrawAmount;
            // this throws an error if fails.
            msg.sender.transfer(withdrawAmount);
        }
        return balances[msg.sender];
    }

    function balance() view public returns (uint) {
        return balances[msg.sender];
    }
}
```

Ethereum

Solidity example

What happens if data is empty?

- Money transfered to account. Default function run.

What is *msg.sender* for internal transactions?

- address of sending contract
  - a contract can have money in our bank!

Ethereum

Solidity exceptions

If a smart contract throws an exception, or error, state is reverted.

Ethereum
Gas

How to pay transaction fees in Ethereum?

- all bytecode instructions have a cost specified in Gas
- transaction has fixed cost in Gas
- especially: storing values is expensive

Transactions specify *Gas price* and *Gas limit*

- *Gas price* is ether given per gas
- *Gas limit* is how much the transaction may spend at most

Ethereum

Gas

Why specific gas per instruction:

- An infinite loop will cost infinitely much gas -> avoid denial of service

What happens if you hit the *Gas limit*?

- Exception is thrown and transaction reverted.
- Gas is still payed!

Which transactions are included?

- Miners will include transactions offering the highest gas price.

Ethereum 2 structure

Ethereum

Structure

Miners and electricity are replaced with validators and stake.
Becoming a miner in PoW:

- **Buying equipments:** Everyone can become a miner, just need to buy equipments
- **Investing electricity**: In case of attacks or malicious activity, electricity is wasted

Becoming a validator in PoS:

- **Staking in the system:** Everyone can become a validator, just need to stake
- **Investing the stake**: In case of attacks or malicious activity, the staked money is slashed

Ethereum
Validators

Validators need to stake 32 Ether in the beginning.

Investing more stakes creates more opportunities.

What happens to the stake in case of attacks or inactivity?

- The stake is slashed

Validators need to be active, have good connectivity.

Not everyone can afford being a validator

- The interested individuals would investigate more about downtime slashing

Ethereum

Validators

Validators

- Proposer
  - Create new blocks with valid transactions
- Attester
  - Verify the proposed blocks and vote for the valid ones

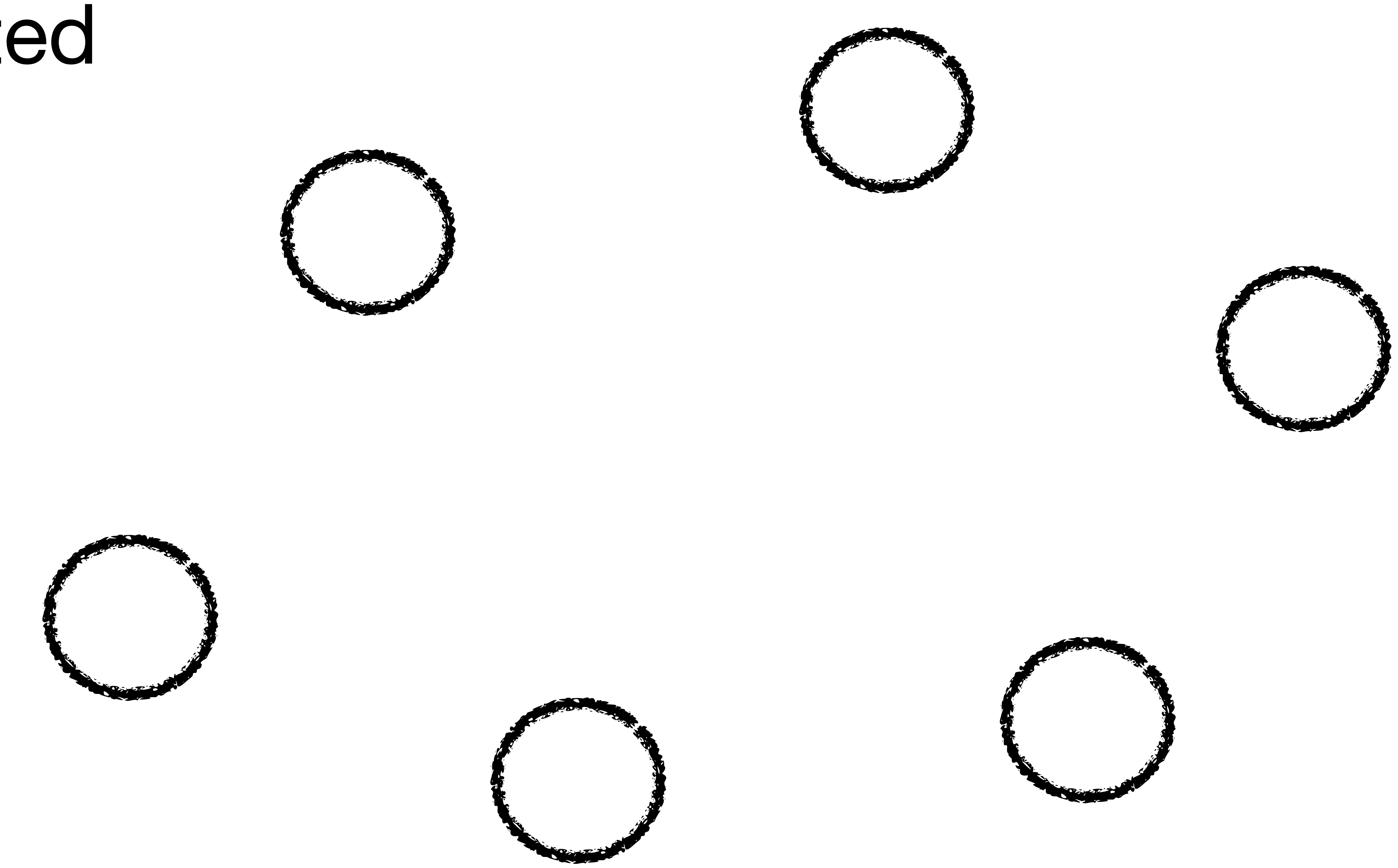
Ethereum

Propose - vote - commit

What happens in a normal committee-based blockchain?

They progress in rounds

- A committee is selected
  - Based on stakes

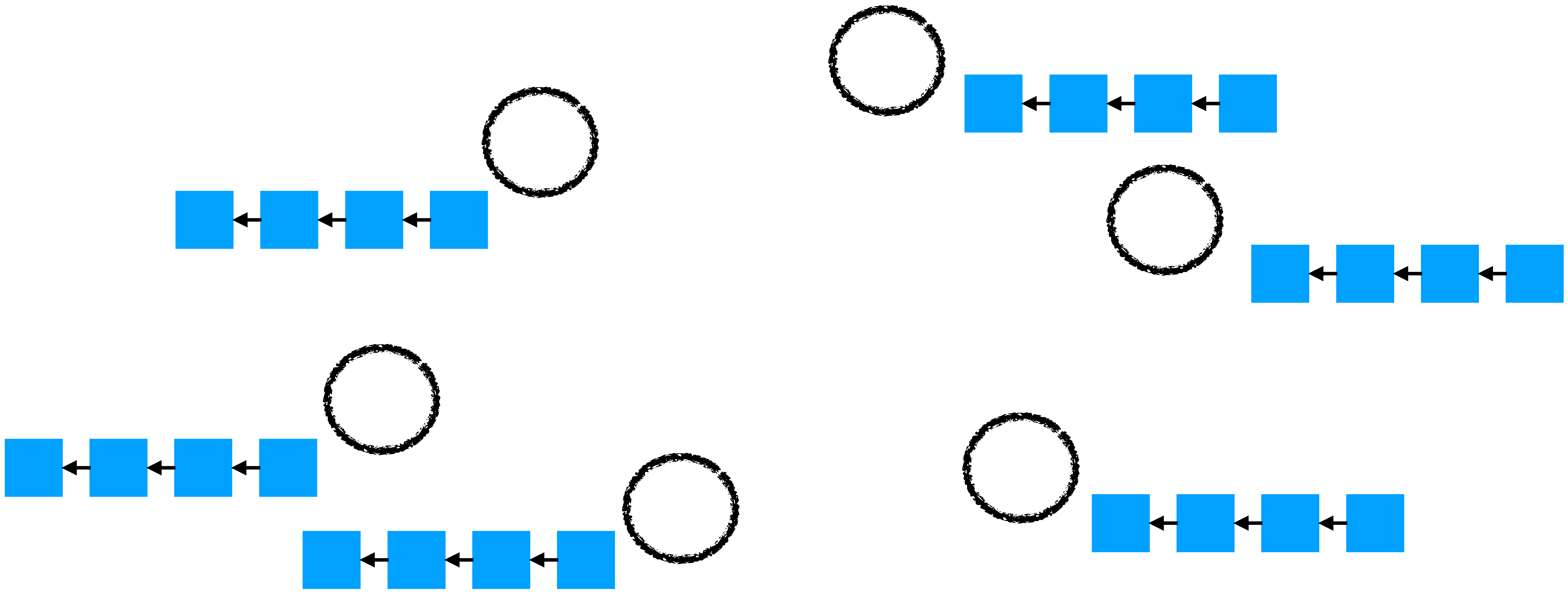


Ethereum

Propose - vote - commit

What happens in a normal committee-based blockchain?

- All committee members have access to the ledger

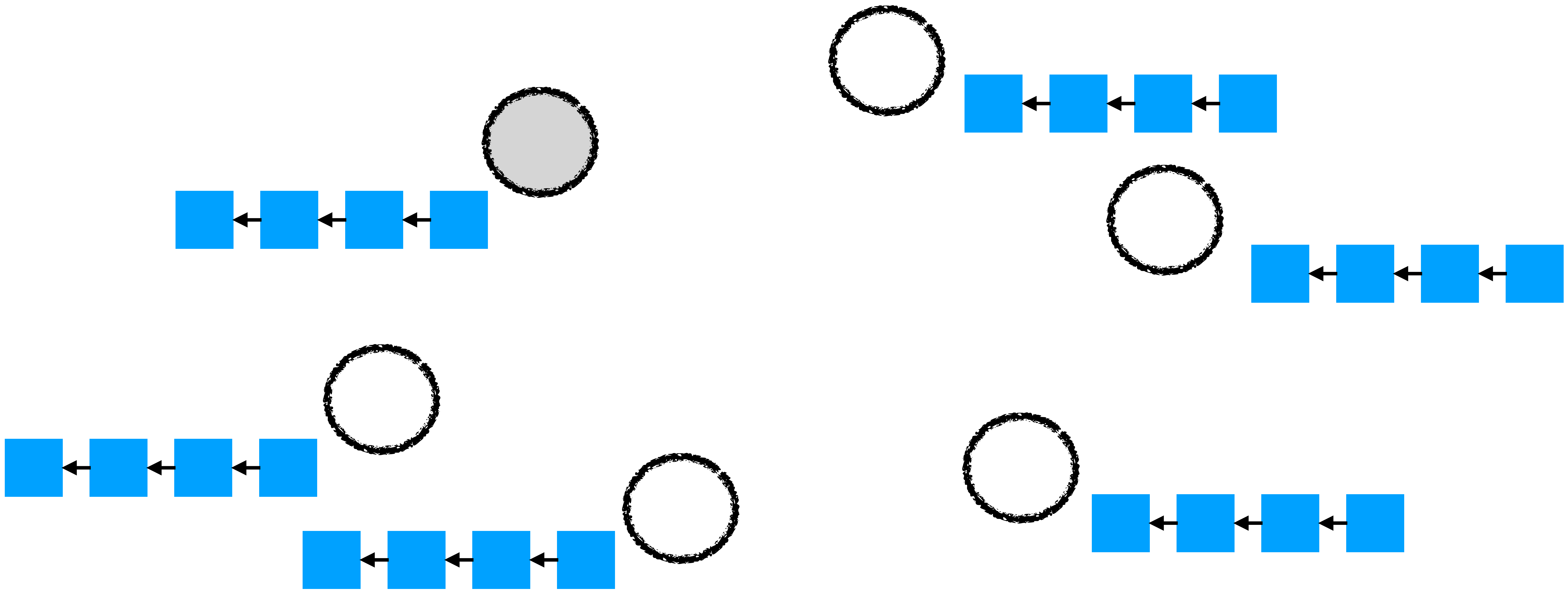


Ethereum

Propose - vote - commit

What happens in a normal committee-based blockchain?

- One of the committee members is selected as a proposer

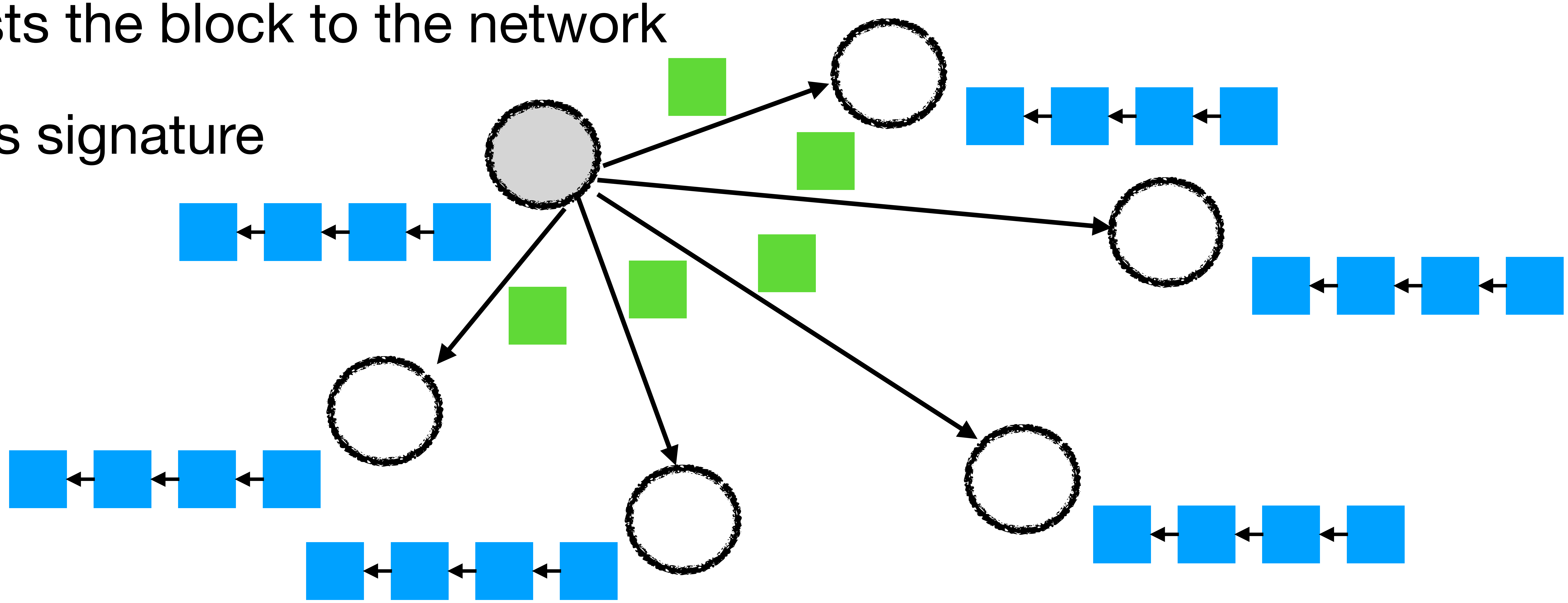


Ethereum

Propose - vote - commit

What happens in a normal committee-based blockchain?

- Proposer creates a block
- Broadcasts the block to the network
    - With his signature

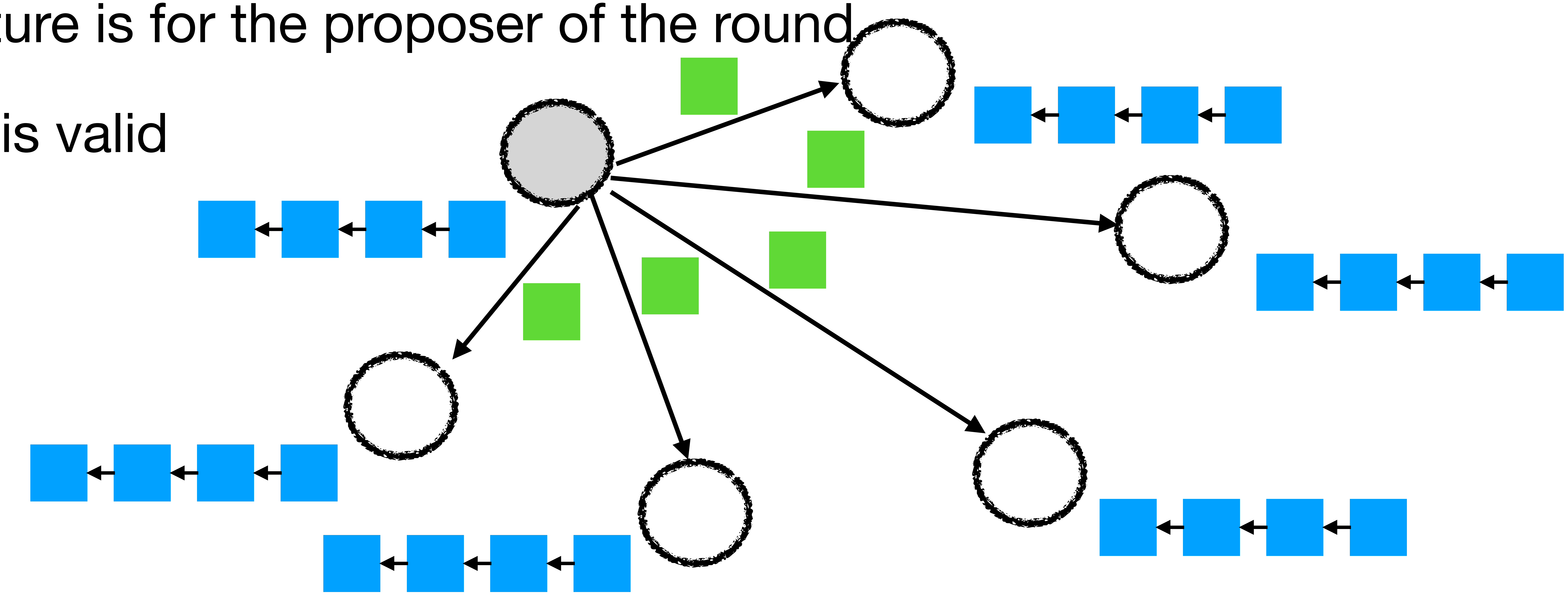


Ethereum

Propose - vote - commit

What happens in a normal committee-based blockchain?

- Others verify the received block
  - Signature is for the proposer of the round
  - Block is valid

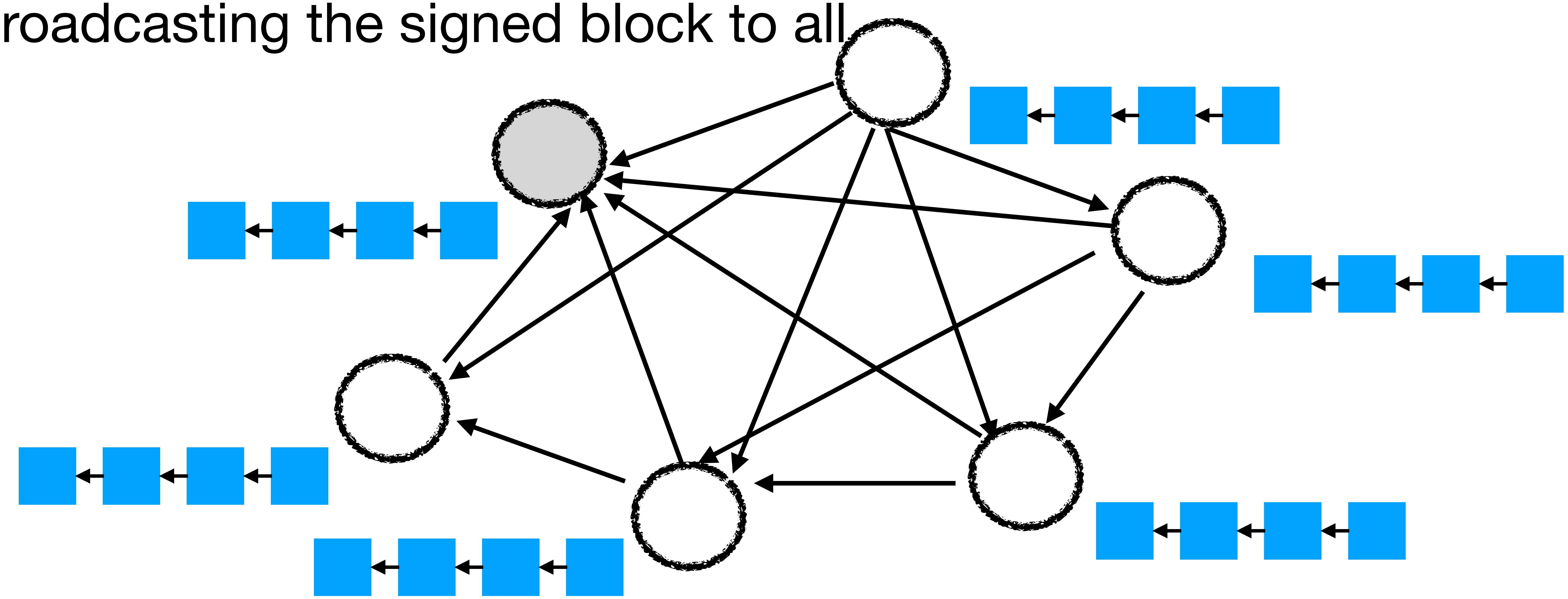


Ethereum

Propose - vote - commit

What happens in a normal committee-based blockchain?

- Others vote for the block if everything is correct
- Voting: Broadcasting the signed block to all

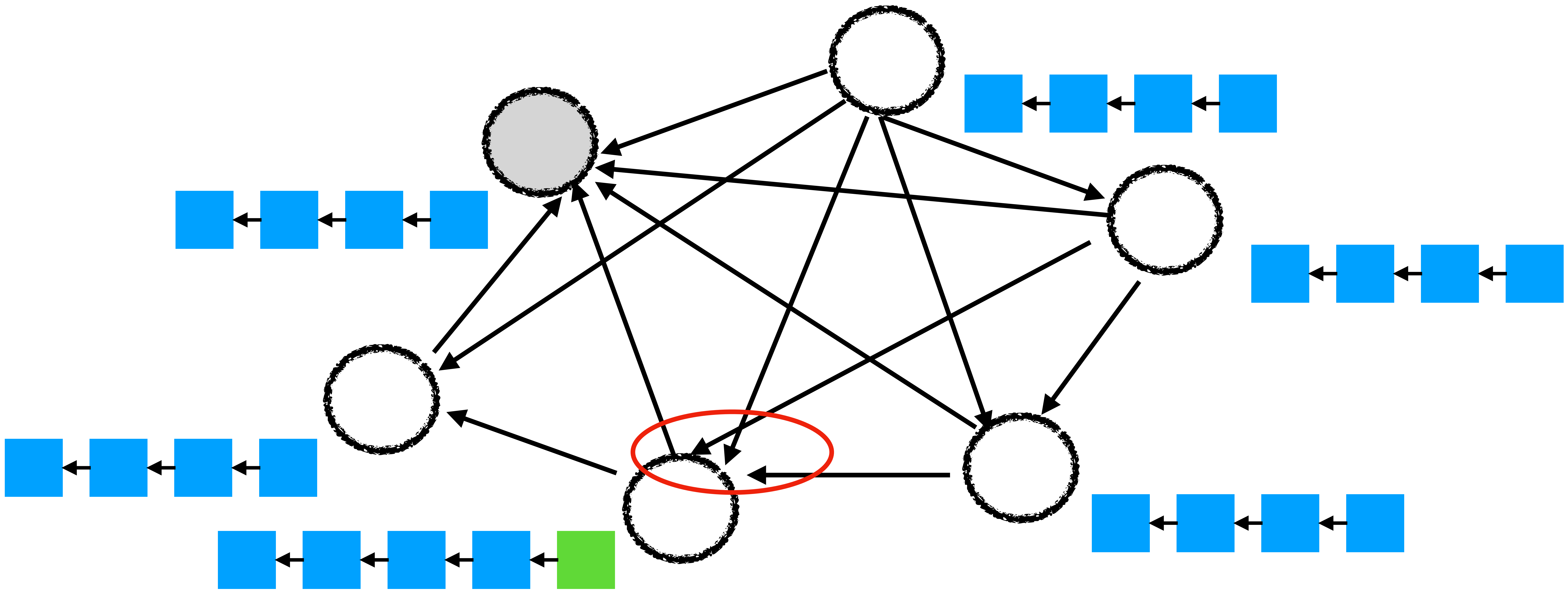


Ethereum

Propose - vote - commit

What happens in a normal committee-based blockchain?

- If a member receives a majority of votes for a block, it commits the block

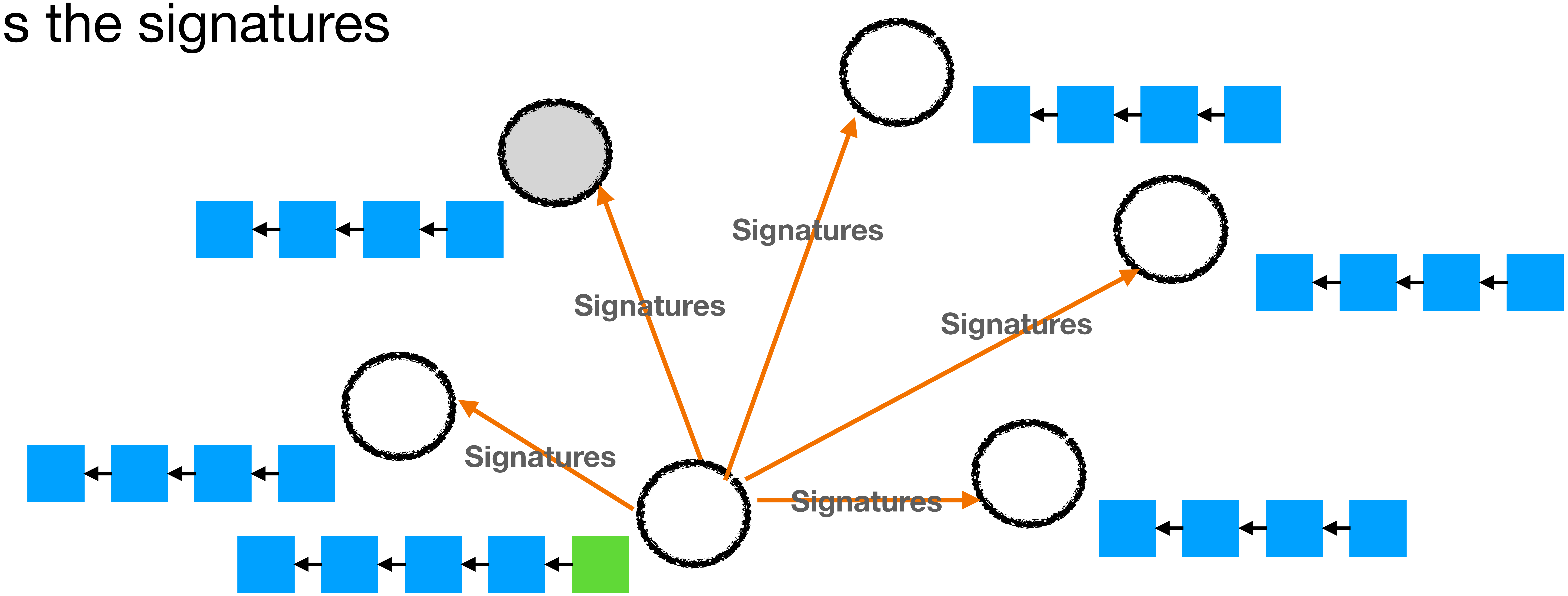


Ethereum

Propose - vote - commit

What happens in a normal committee-based blockchain?

- It also broadcasts the committed block with proof to all
    - Proof is the signatures

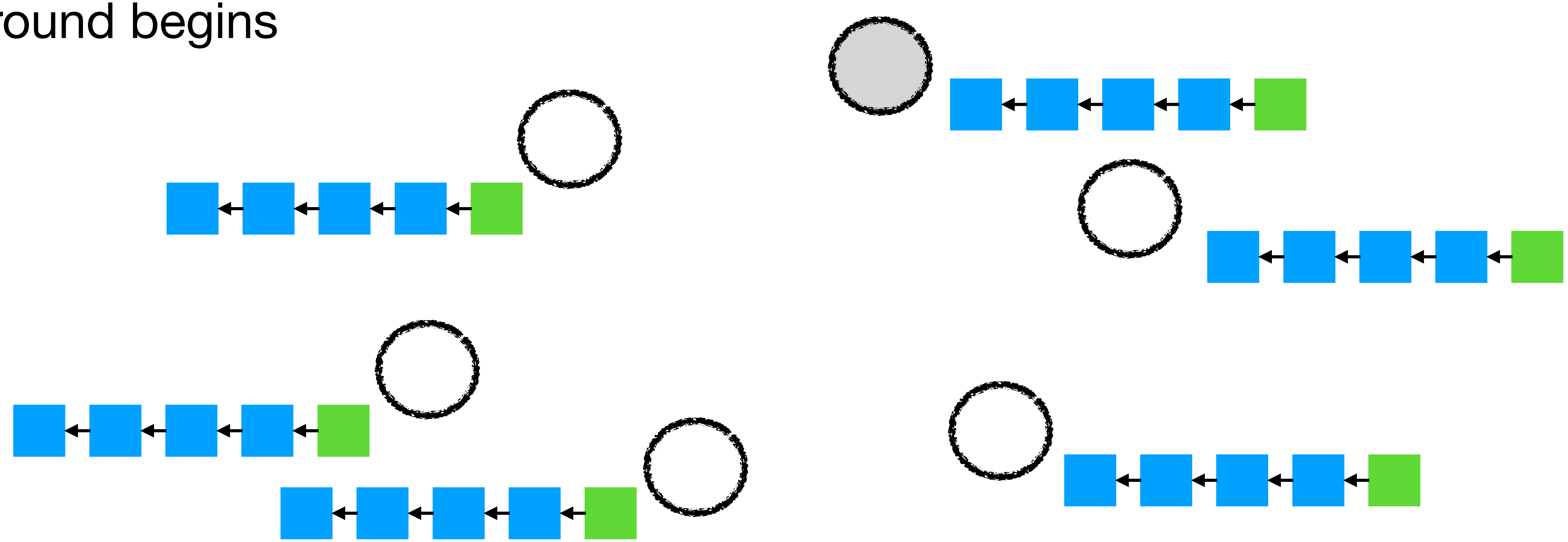


Ethereum

Propose - vote - commit

What happens in a normal committee-based blockchain?

- Everyone commits the block
- New round begins



Ethereum

Round failure

What happens if someone does not receive enough votes, or proof for a block?

- Waits for some pre-specified time
- Sends a next-round message to all
  - “Round r is failed, let’s move to round r+1”’
- If someone has the proof, sends it so they can catch up

Ethereum

Round failure

What happens if a majority send next-round message?

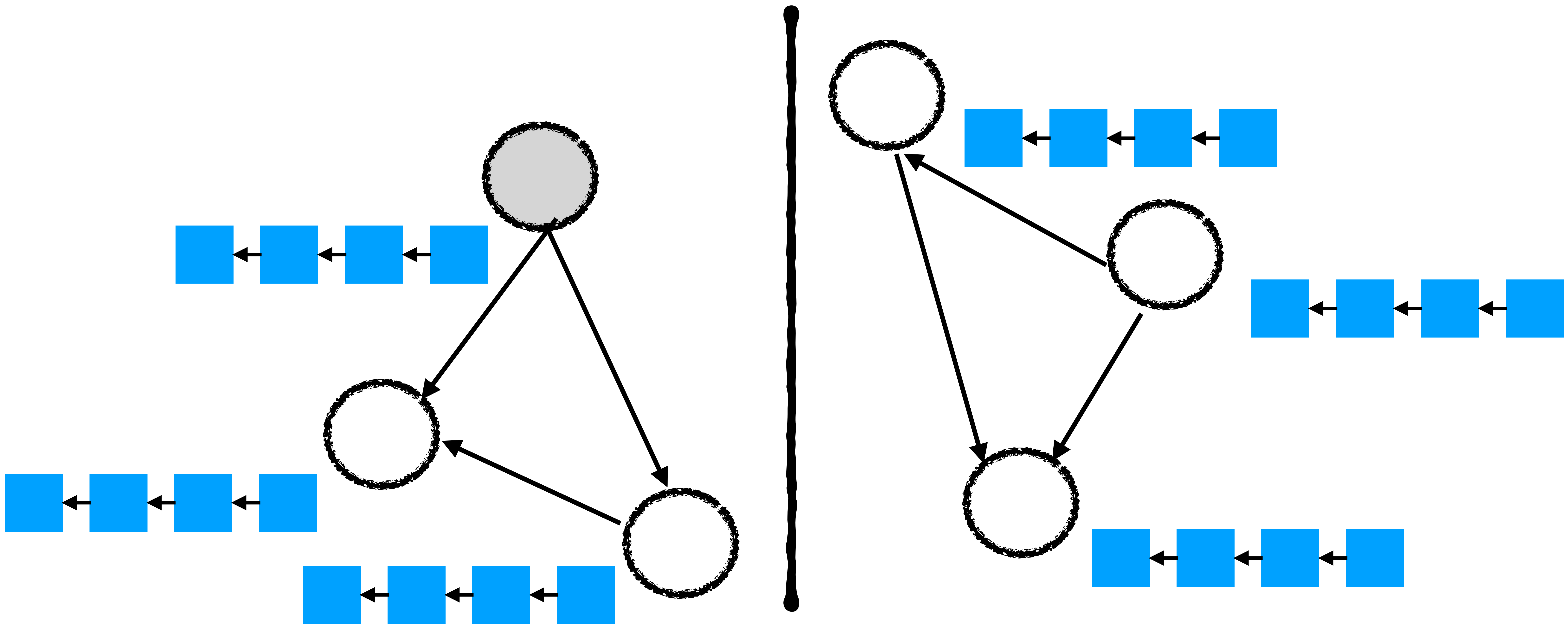
- Round is considered failed
- They all go to the next round and discard the proposed block

Connectivity is important!

Ethereum

Network partitions

What if a network partition happens?

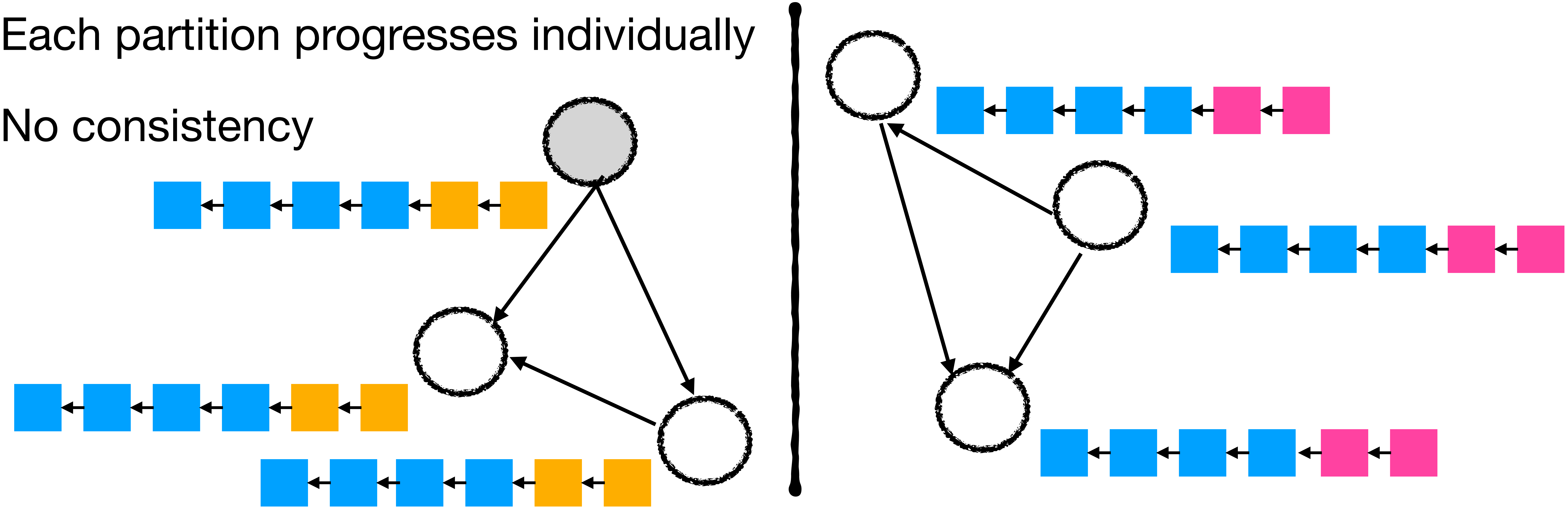


Ethereum

Network partitions

What if a network partition happens?

- Case 1: We only need at least 3 votes for block approval
  - Each partition progresses individually
  - No consistency

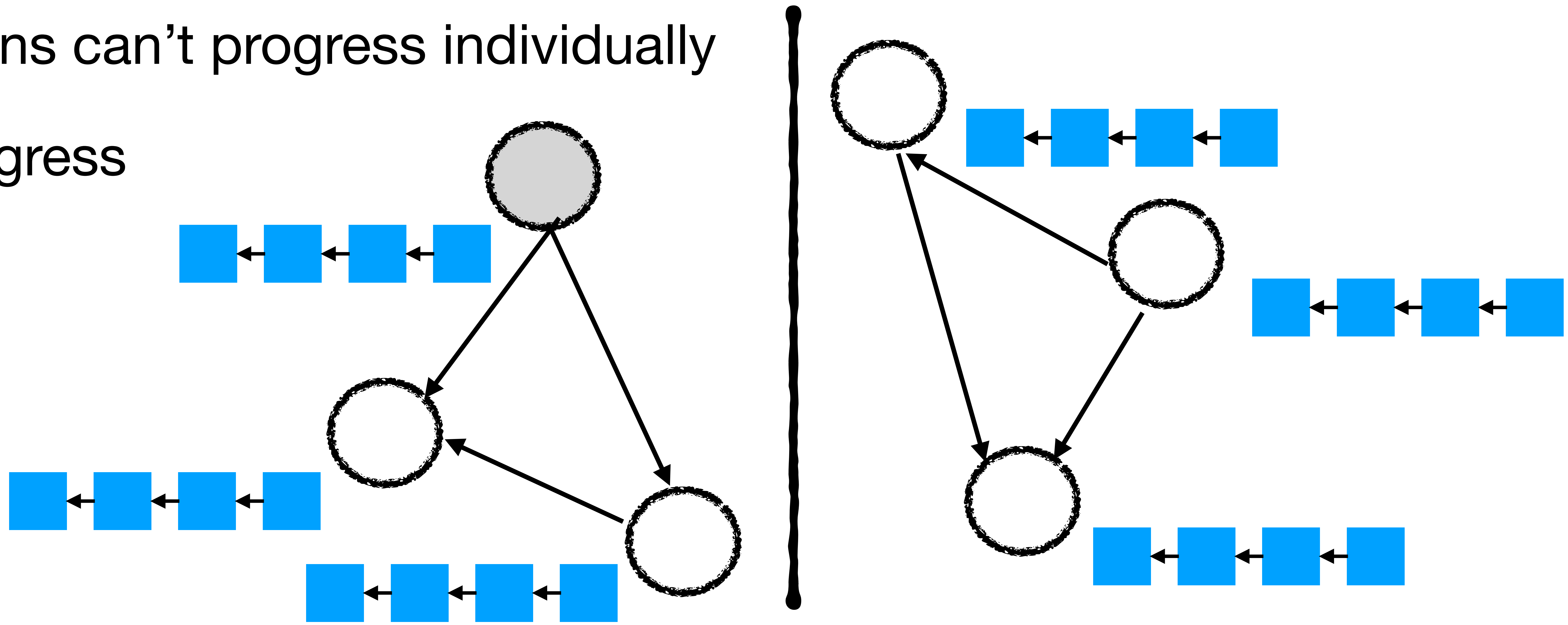


Ethereum

Network partitions

What if a network partition happens?

- Case 2: We need at least 4 votes for block approval
  - Partitions can't progress individually
  - No progress



Ethereum

CAP Theorem

In case of network partitions, it is impossible to have both consistency and progress.

- Consistency is more important
- Most schemes require majority vote, disallow progress in case of network partitions
- Once a block gains enough votes, it is considered final

Ethereum

CAP Theorem

Ethereum intends to have both consistency and progress!

- Decouples finalizing blocks from proposing blocks
- Two different voting
  - Vote for proposed blocks
  - Vote for finalizing blocks
- If network partition happens
  - node can still propose and vote for blocks
  - After going back online they can finalize the blocks

Ethereum

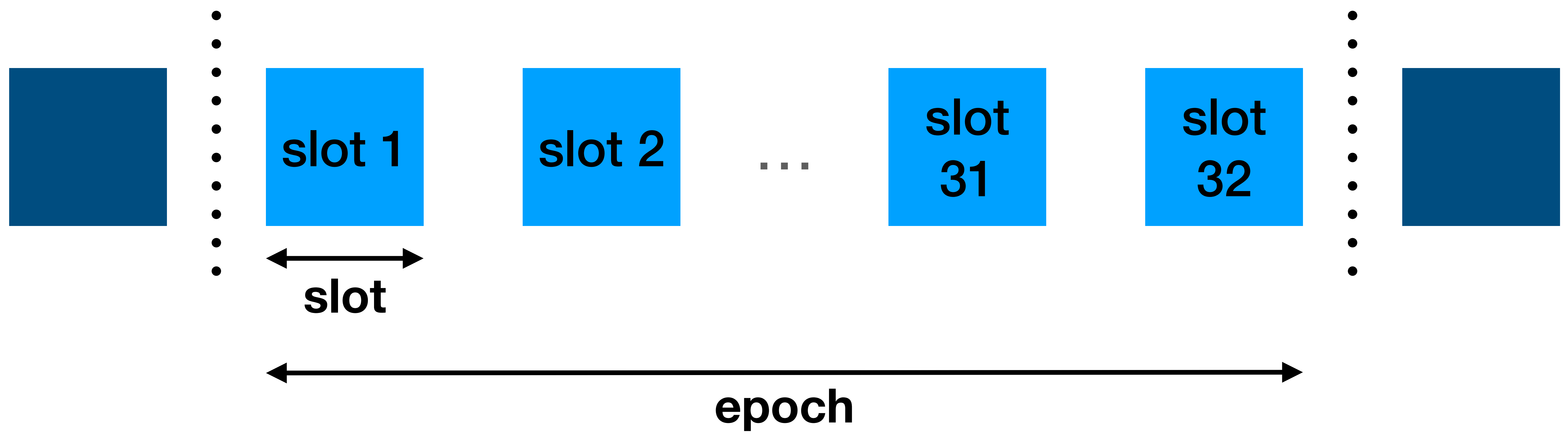
Epoch

Ethereum works in epochs

Each epoch consists of 32 slots

Each slot is 12 seconds

In each slot, at least one block is proposed



Ethereum

Validators

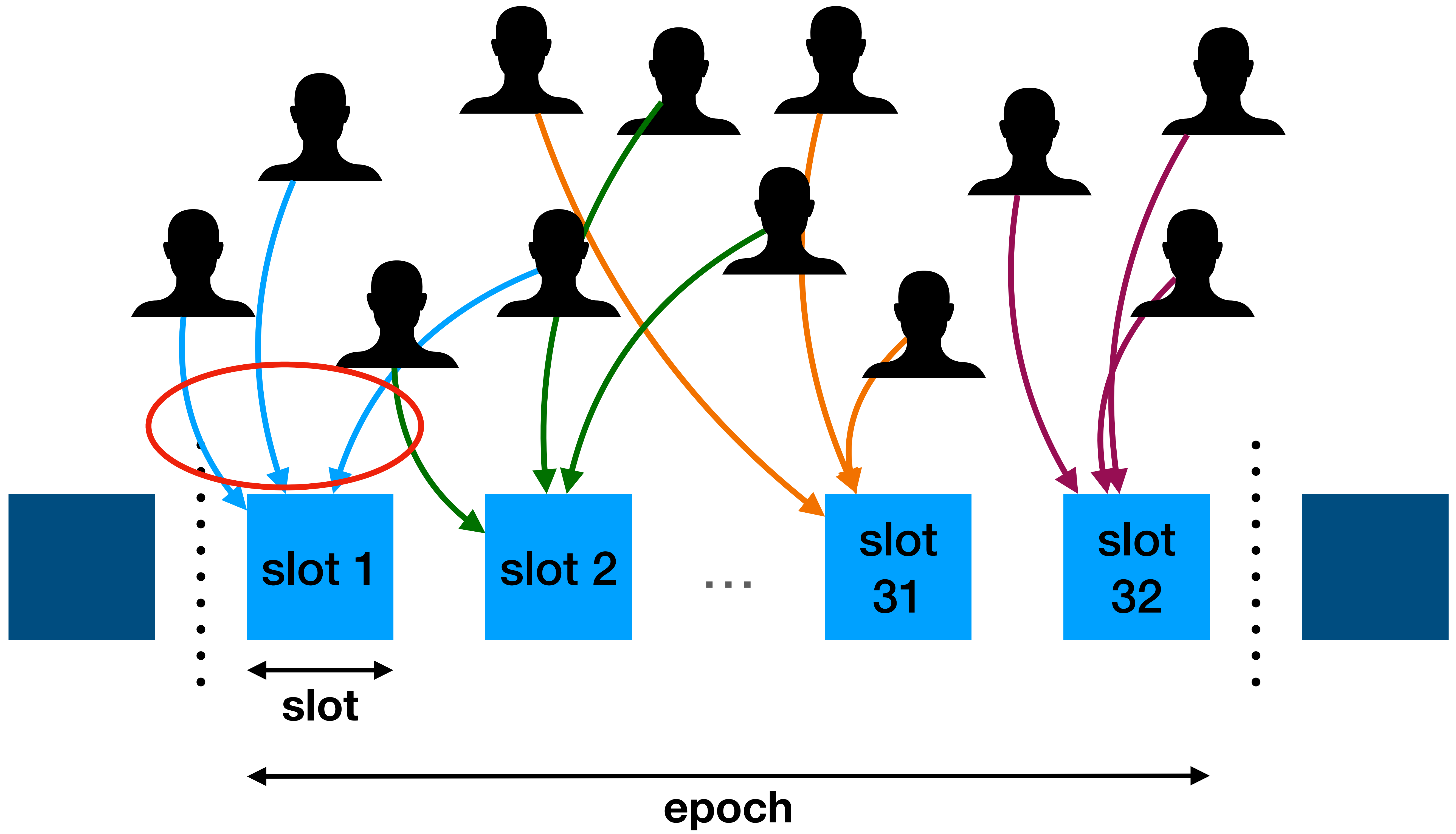
Validators are divided into committees in each epoch

- One committee is assigned to one slot
  - 32 committees in total
- Each validator can only be in one committee

Ethereum

Epoch

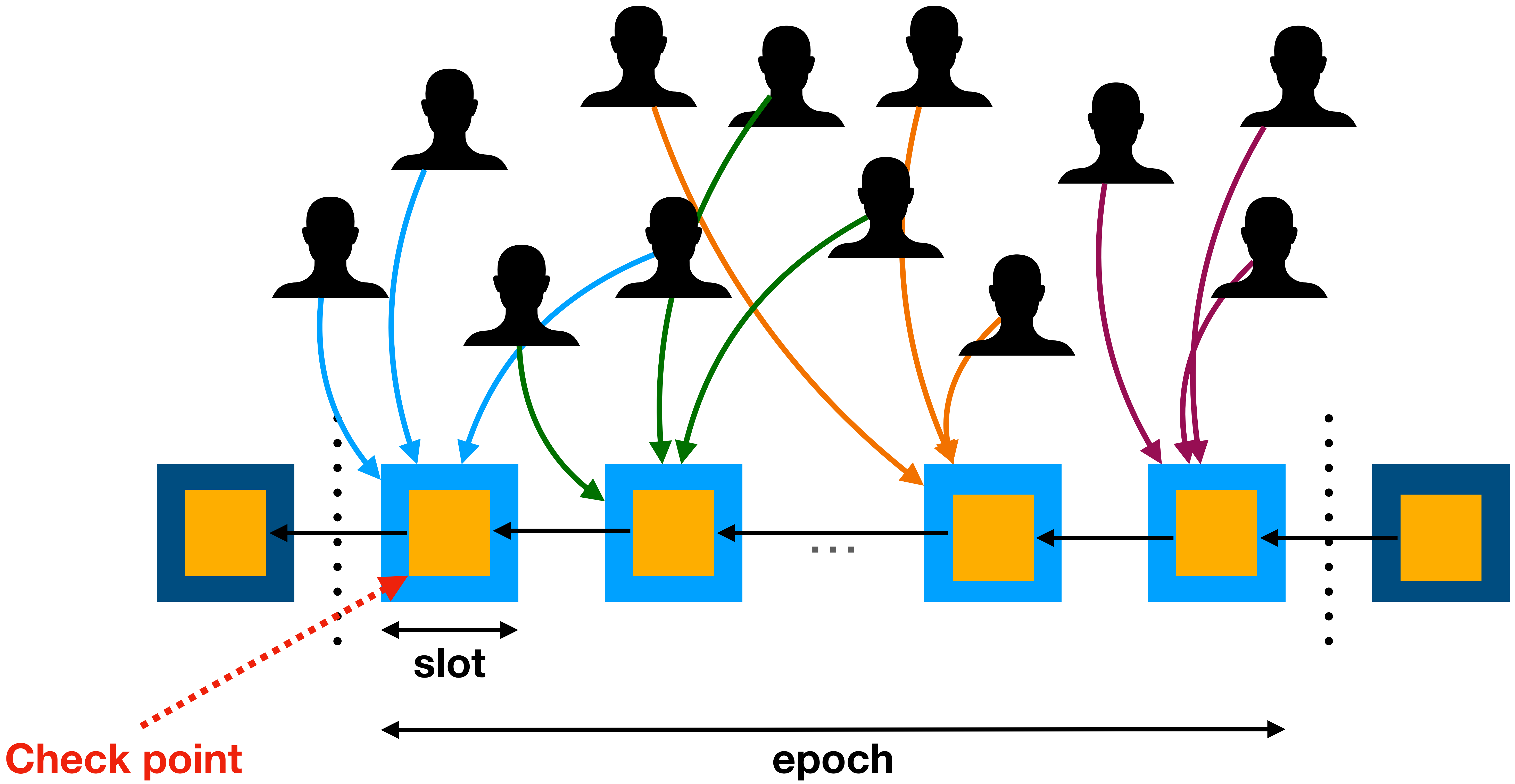
During a whole epoch, a validator either attests in one slot, or proposes a block in one slot.



Ethereum

Checkpoint

A pair (b, e) where b is block produced in the first slot of epoch e



Ethereum

CAP Theorem

Ethereum intends to have both consistency and progress!

- Decouples finalizing blocks from proposing blocks
- Two different voting
  - Vote for proposed blocks
  - Vote for finalizing blocks
- If network partition happens
  - node can still propose and vote for blocks
  - After going back online they can finalize the blocks

Ethereum

CAP Theorem

Two different voting

- Vote for proposed blocks
  - In each slot, validators vote for a block and grow the chain
- Vote for finalizing blocks
  - At each epoch, validators vote for the last valid checkpoint
  - A checkpoint is final if 2/3 of validators vote for it
  - Once an epoch is final, all blocks in that epoch are final

State stored in Ethereum blockchain

Bitcoin

Block structure

Header:

```
PrevBlockhash
Nonce
Timestamp
```

Transaction data

```
Merkle tree
```

Merkle tree allows to easily proof that a transaction is included in a block.

Ethereum
Block structure

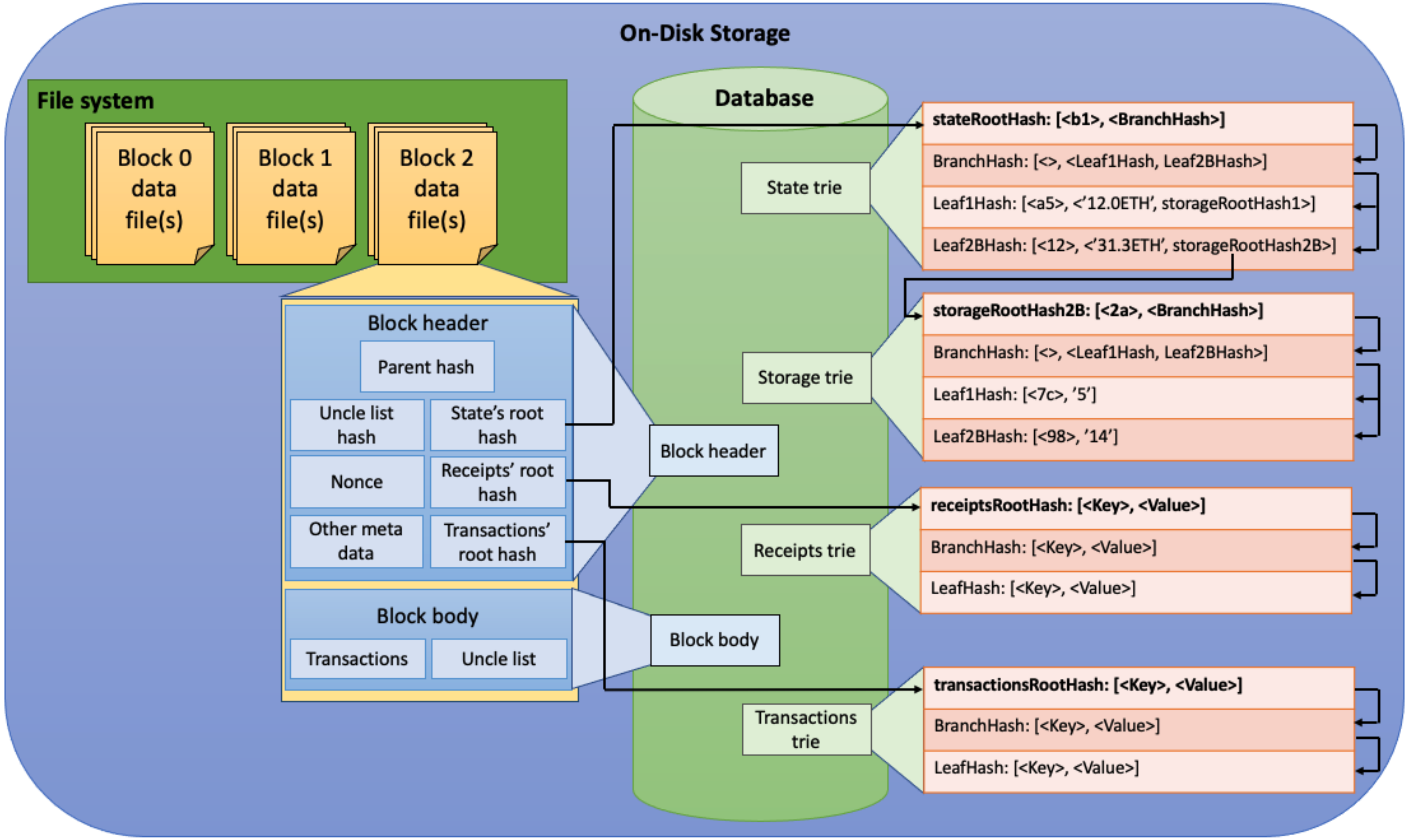
Header:

```
PrevBlockhash
Nonce
Timestamp

State root hash
Receipts root hash
```

Transaction data

```
Merkle tree
```



Ethereum

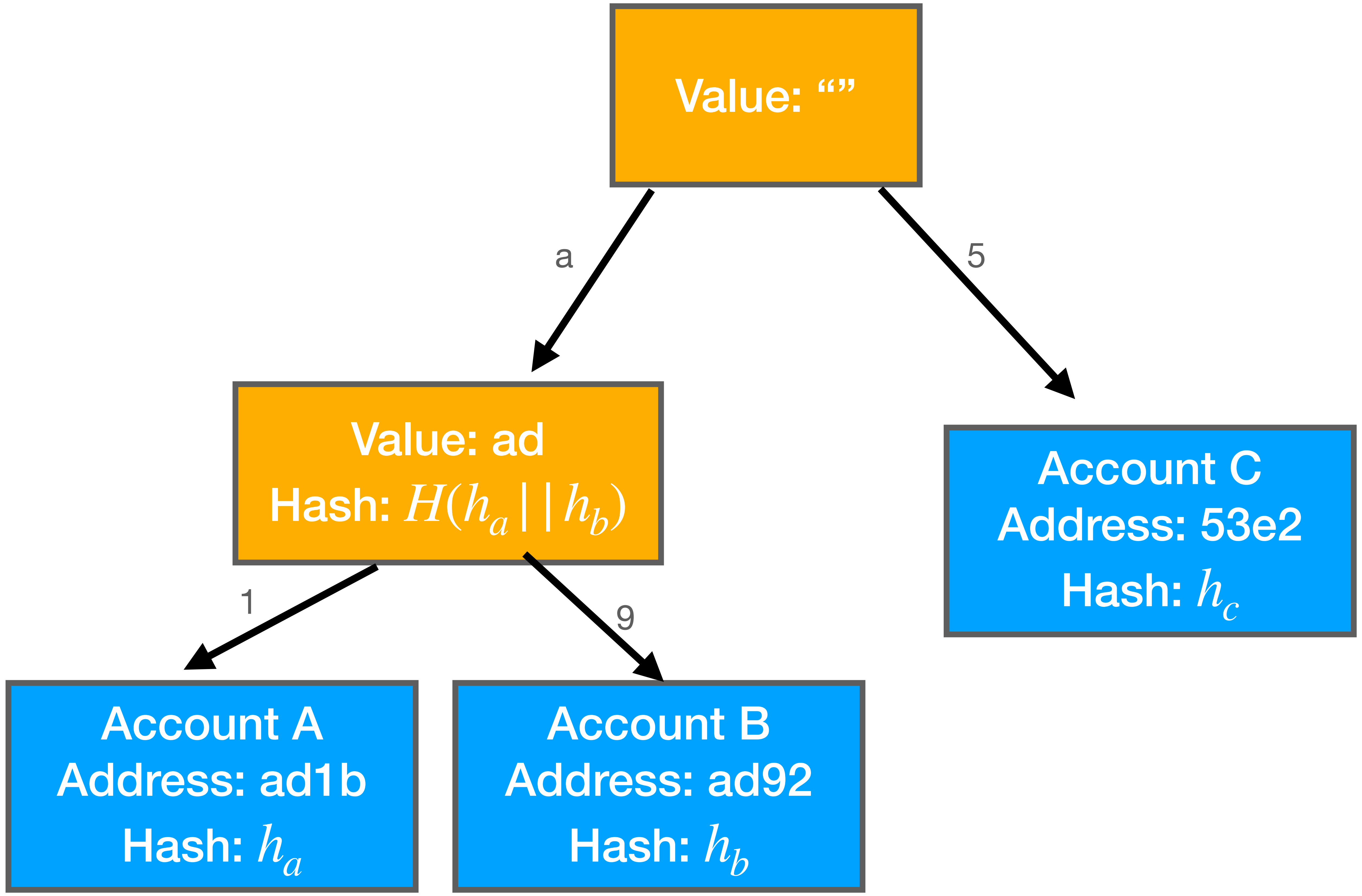
State trie

Stores accounts:

```
Address:
  [Value,
   Nonce,
   StorageRoot,
   CodeHash]
```

Trie:
Merkle tree that supports

```
update
lookup
proof
```



On new block, only changed nodes get added.

Ethereum

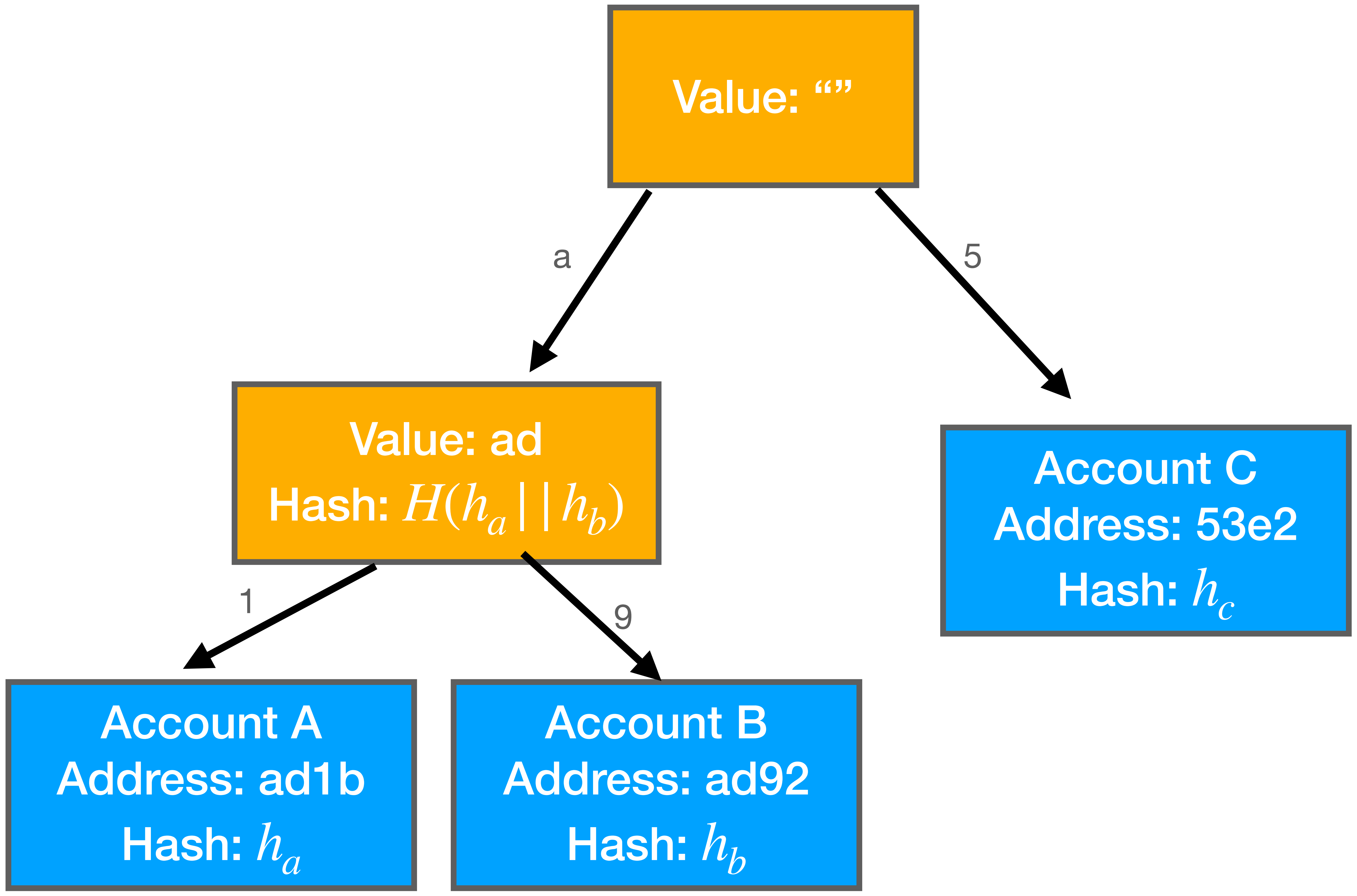
State trie

Stores accounts:

```
Address:
  [Value,
   Nonce,
   StorageRoot,
   CodeHash]
```

Trie:
Merkle tree that supports

```
update
lookup
proof
```



On new block, only changed nodes get added.

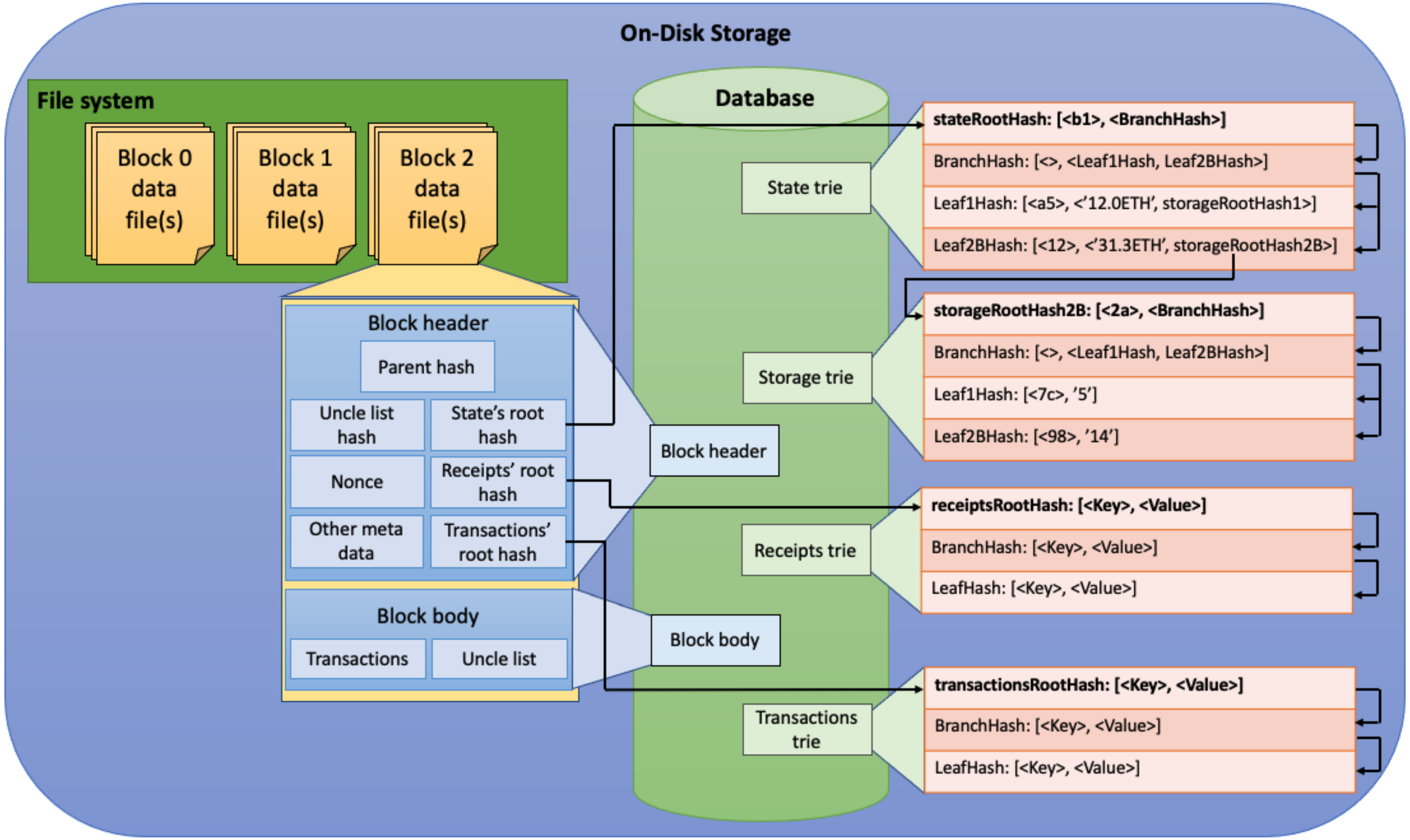
Ethereum
State trie

Stores accounts:

```
Address:
  [Value,
   Nonce,
   StorageRoot,
   CodeHash]
```

Trie:
Merkle tree that supports

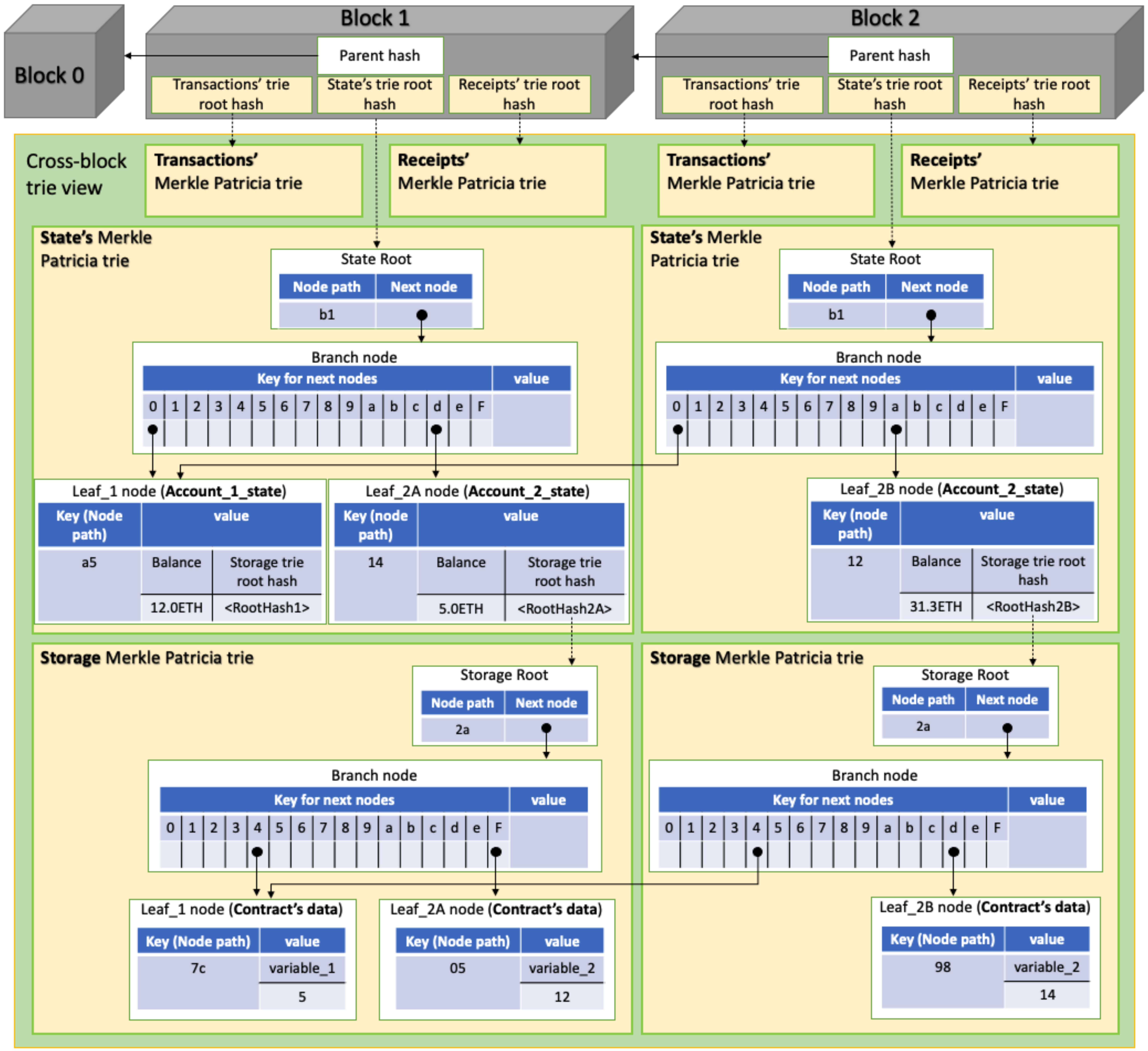
```
update
lookup
proof
```



`StorageRoot` is the root of a different trie.

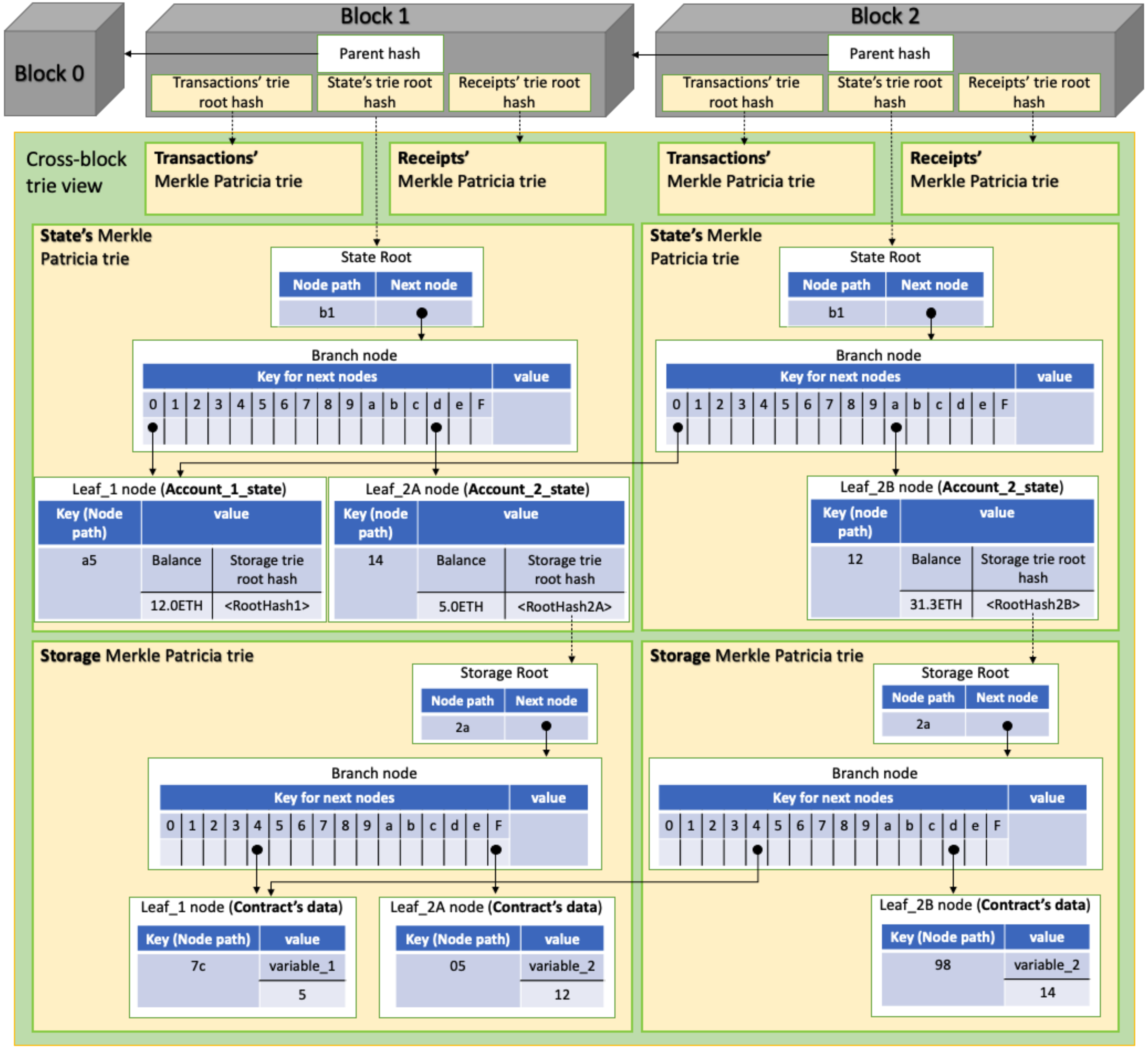
Ethereum

State trie



Ethereum

State trie



Ethereum

Read contract state

1. ask trusted node

2. receive inclusion proof for

   stateRoot: storageTrie
   account state: stateTrie

   and block header

Ethereum

Receipts trie

Stores transaction results:

```
From: address
To:   address
Status: … // aborted?
Logs: events
ContractAddress address
  // new contract address,
  // if created
```

Return transaction results, by emitting Events, which are added to the `logs`.