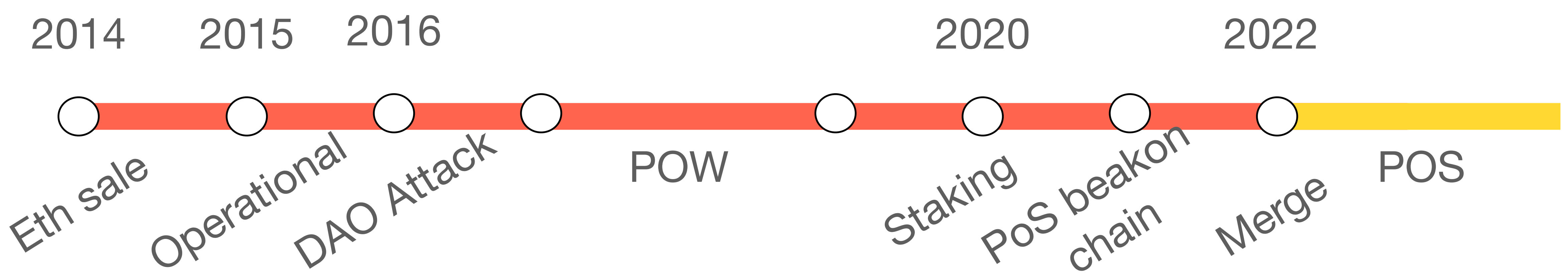
Ethereum

Introduction

Leander Jehl

Ethereum

Timeline



- 2014 Innitial Ethere sold on bitcoin
- 2016 DAO Attack: A Hacker expoited a smart contract bug, the community decided to undo attack, by discrading some blocks.
- 2020 Possible to stake ether and participate in PoS beakon chain, that votes on blocks created by PoW
- 2022 PoW depricated and Ethereum using PoS to create bocks

From PoW to PoS

Ethereum - PoW

- Fast blocks (every 12sec)
- Structured P2P network
- Uses PoW hashing function not suited for ASICs
- Uncles: Blocks lost in a fork still get some reward

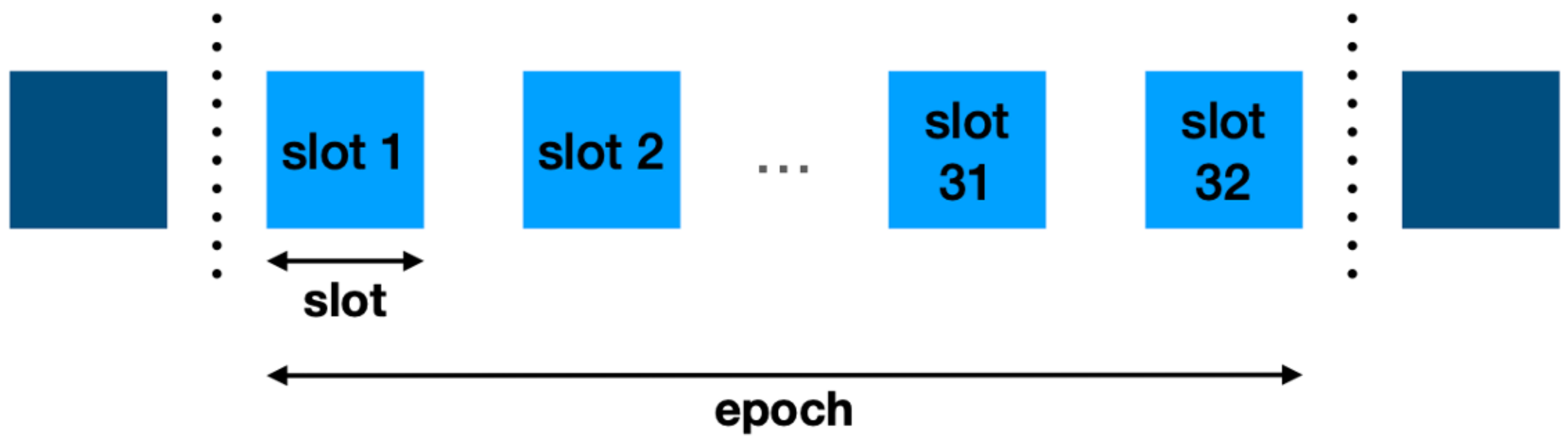
Ethereum - PoS

- One staker (validator) needs to deposit 32 ether (>75.000$ 2024)
- Stake can be slashed, but not increased
  -> 1.000.000 validators
- Validators need to run a node
- Can run one node for multiple validator ids.

Problem: Collecting signatures from 1mill nodes takes a lot of time.

Ethereum - PoS

Slots, Epoch, and Time

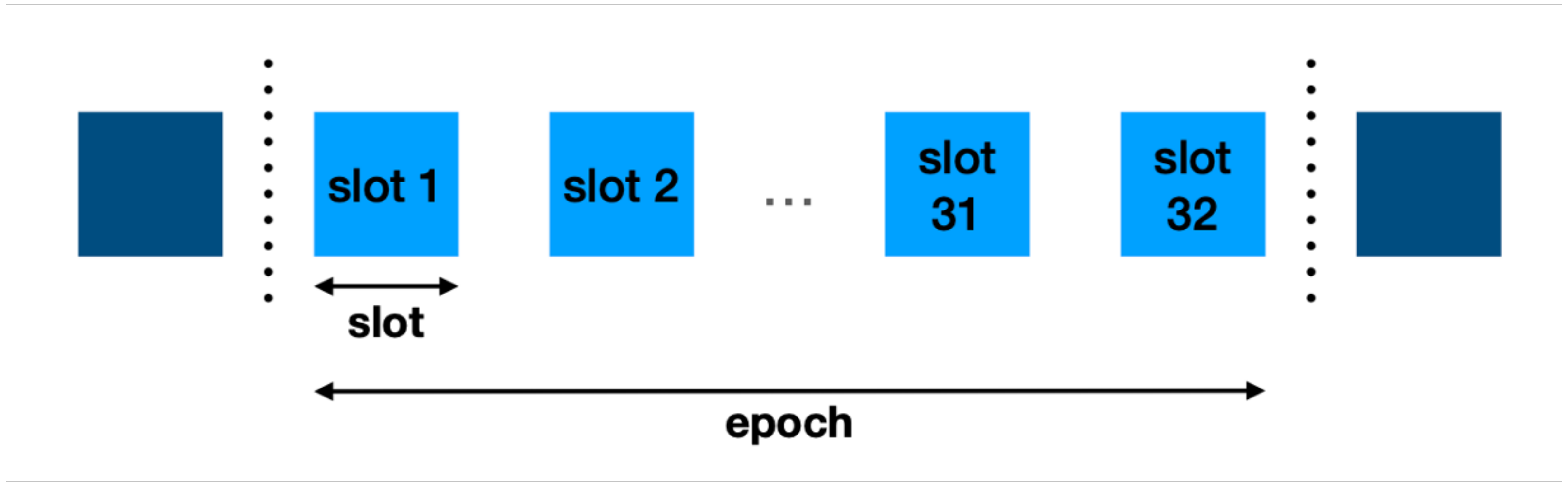


- One epoch is 6.4 minutes
- Each validator needs to vote once in each epoch (vote -> reward, no vote -> get punished)
- Epoch devided in 32 slots (each 12 sec)
- Validators devided in 32 committees
- Vote in your slot!

One block every 12 sec without requiring one vote every 12 sec!

Ethereum - PoS

Proposers



- At the start of the epoch
  32 random validators (proposers)
  are selectedto propose blocks

- Proposers also collect votes

- Proposers get extra reward

Ethereum - Forks

Two mechanisms:

- **LMD Ghost:** Ensure new blocks can be created.
- **Casper FFG:** Create checkpoints

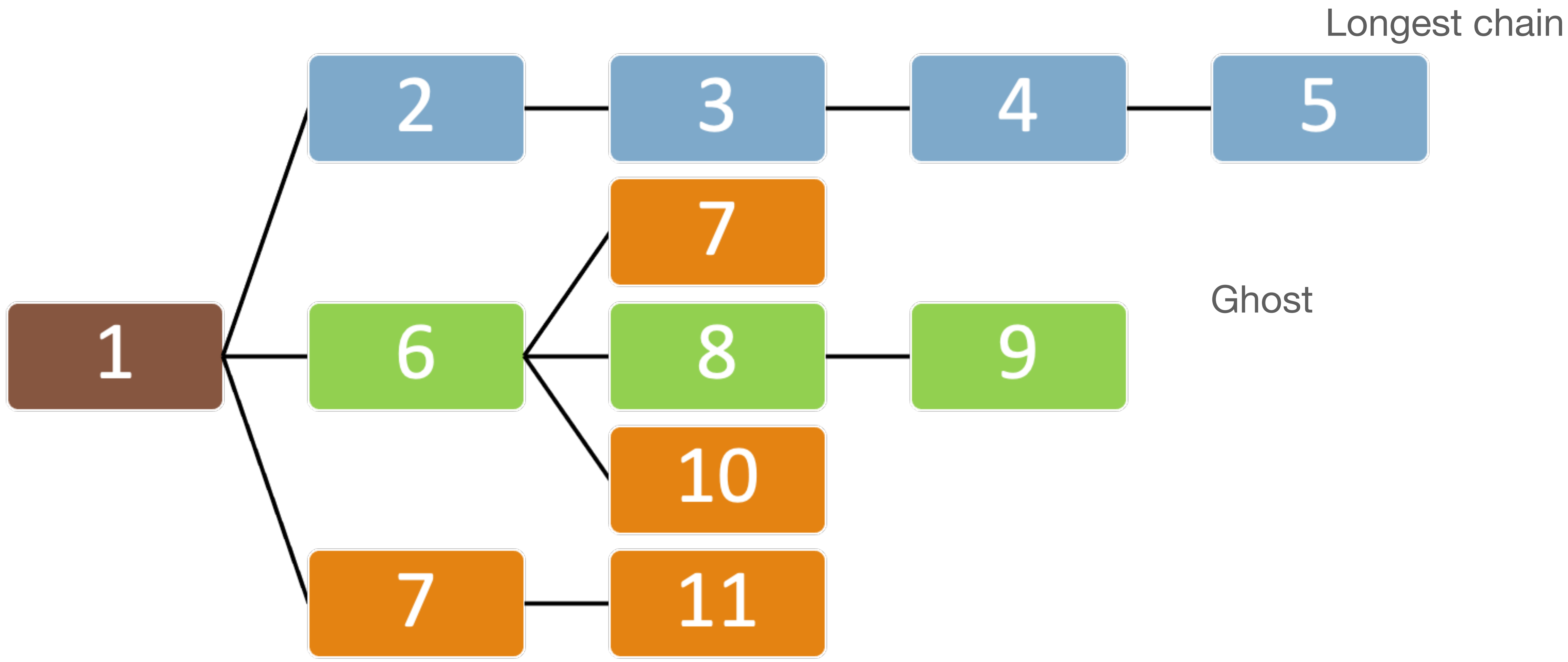
- Creating checkpoints is slow
- Forks can still arise between checkpoints

Ethereum - Forks

LMD Ghost

LMD Ghost is a fork choice rule (like longest chain):

- Rule which fork to extend

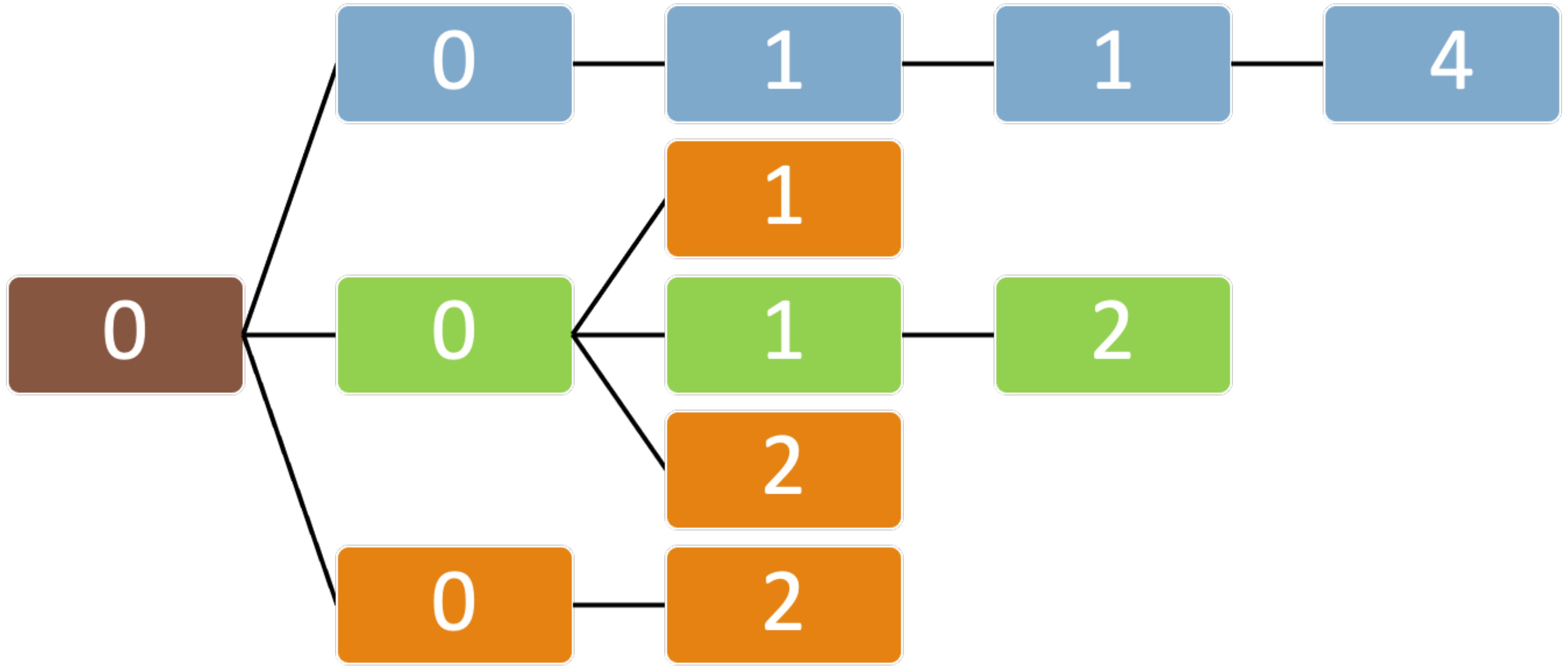


Ethereum - Forks

LMD Ghost

LMD Ghost: Extend fork with most votes

- Every validator can vote for one block

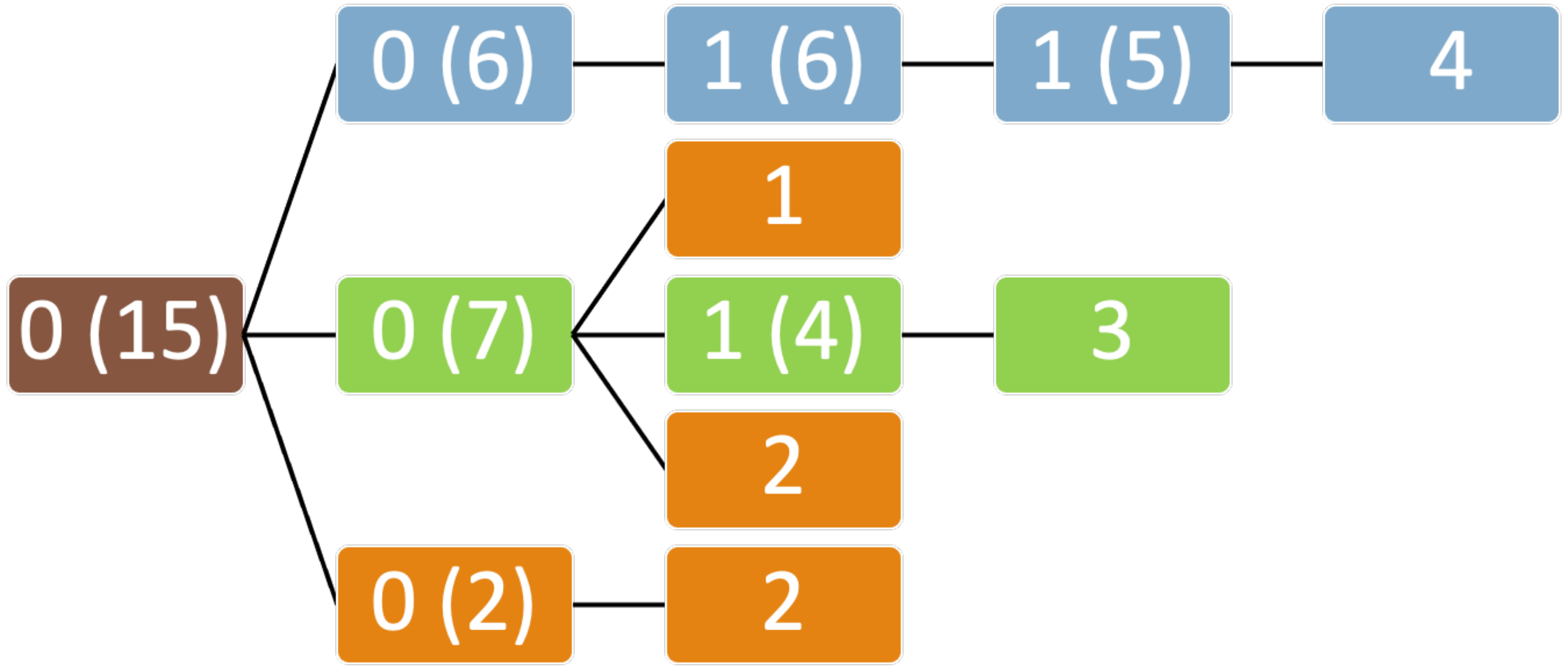


Ethereum - Forks

LMD Ghost

LMD Ghost: Extend fork with most votes

- Every validator can vote for one block
- Votes are summed for parents

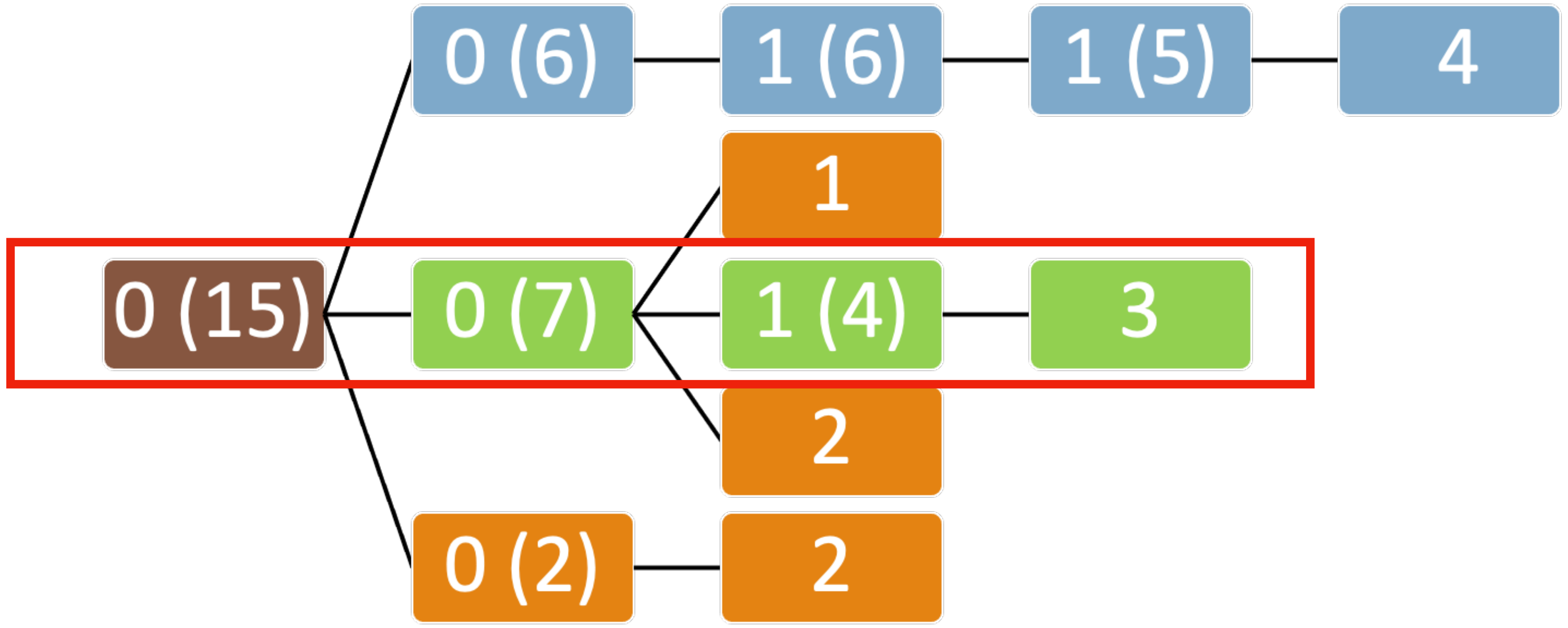


Ethereum - Forks

LMD Ghost

LMD Ghost: Extend fork with most votes

- Every validator can vote for one block
- Votes are summed for parents
- Choose fork with most votes

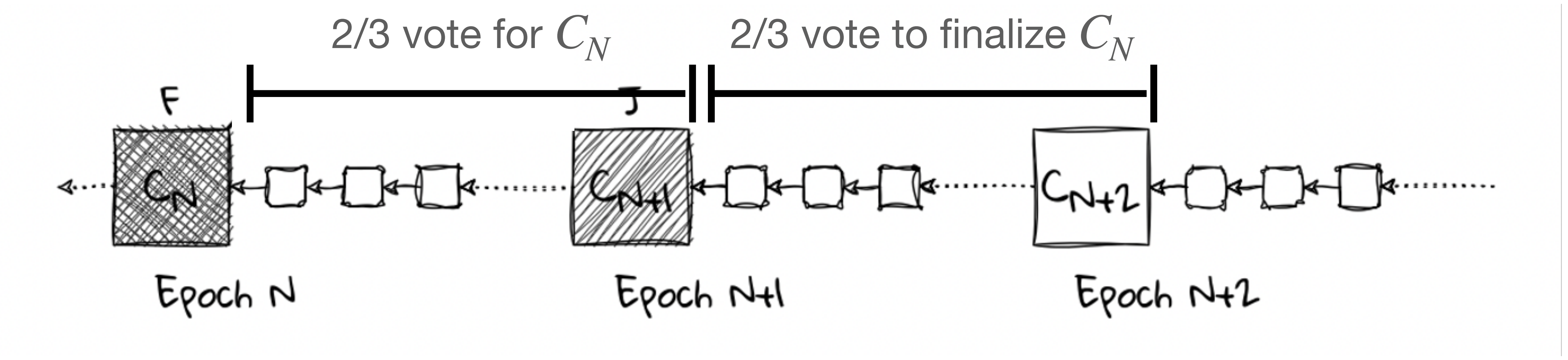


Ethereum - Forks

Casper FFG

Decide on checkpoint (consensus)

- Validators can vote to create checkpoint
- Need 2/3 of validators to vote for one checkpoint
- Needs to happen in 2 consecutive epochs

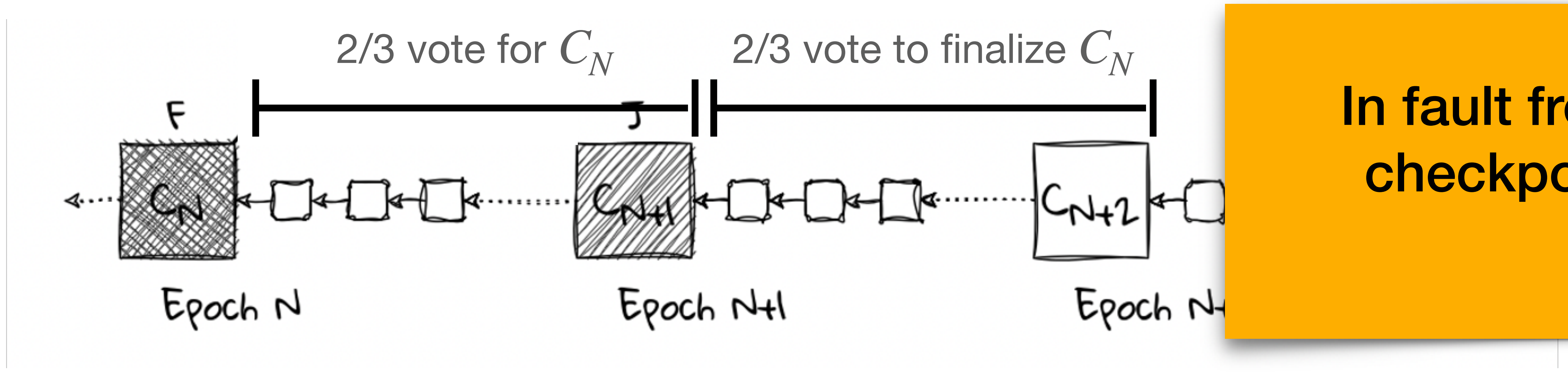


Ethereum - Forks

Casper FFG

Decide on checkpoint (consensus)

- Validators can vote to create checkpoint
- Need 2/3 of validators to vote for one checkpoint
- Needs to happen in 2 consecutive epochs



In fault free cases, creating a checkpoint takes 2 epochs
12 minutes

What are Smart Contracts

Bitcoin scripts scripts

Spending conditions

Transactions:

$$tx = \langle \underbrace{[(id_1, \sigma_1), (id_2, \sigma_2)]}_{\text{Inputs}}, \underbrace{[(pk_a, value_a), (pk_b, value_b)]}_{\text{Outputs}} \rangle$$

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

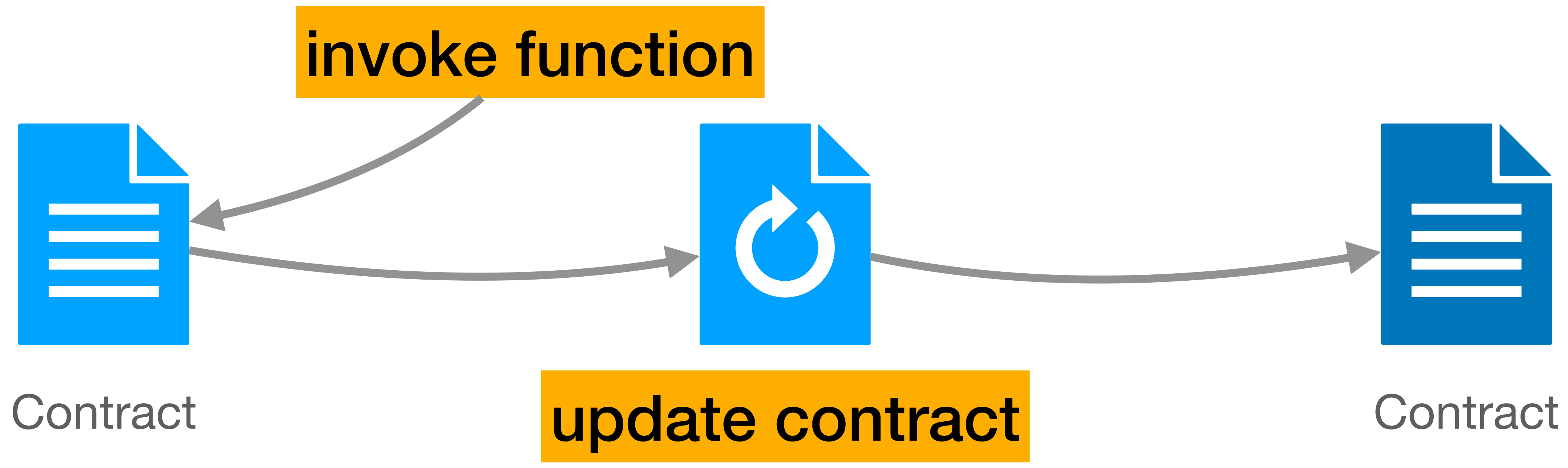
Name	Spending condition	Redeeming argument	
P2Pk Pay to public key	Public key	Signature	
P2PkH Pay to public key hash	Hash of Public key	Public key and signature	
Multisig	*m* public keys and parameter *k*	*k signatures*	

Ethereum

Smart Contracts

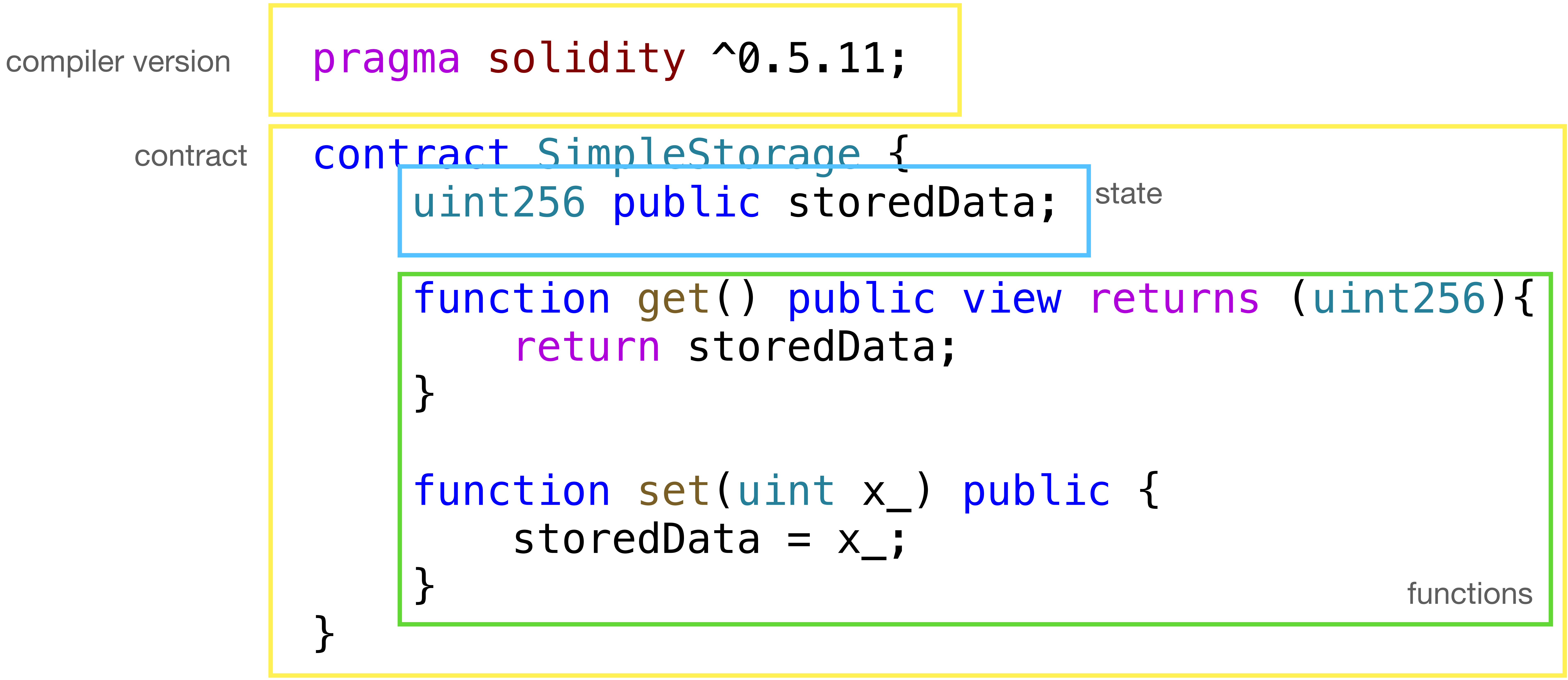
In ethereum, a contract is like a object from OOP, with fields and methods

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



Ethereum

Example: Simple Storage



```
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

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?

Who can view values?

Who can change the code?

```
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

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)

Ethereum

Smart Contract

Why use a smart contract

-

Ethereum

Smart Contract

Why use a smart contract

- No legal system to enforce paper contract
- Cheaper than paper contract

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

Can create different contracts with the same code.
(Like objects with the same type)

Ethereum

Accounts

In a contract written in Solidity, you can access:

- The address of the current contract:

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

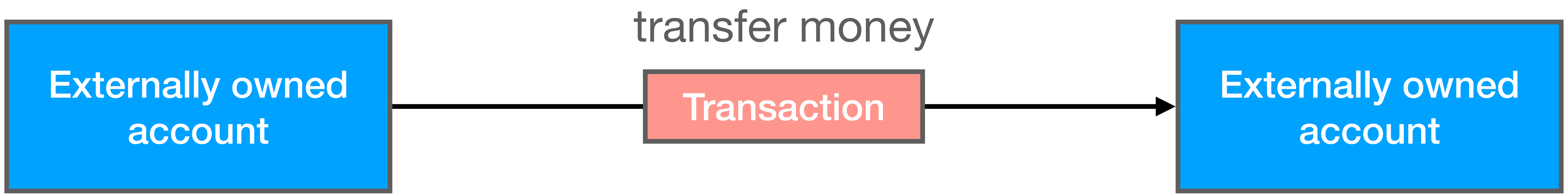
- The balance of the contract:

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

Ethereum

Transactions and authenticaion

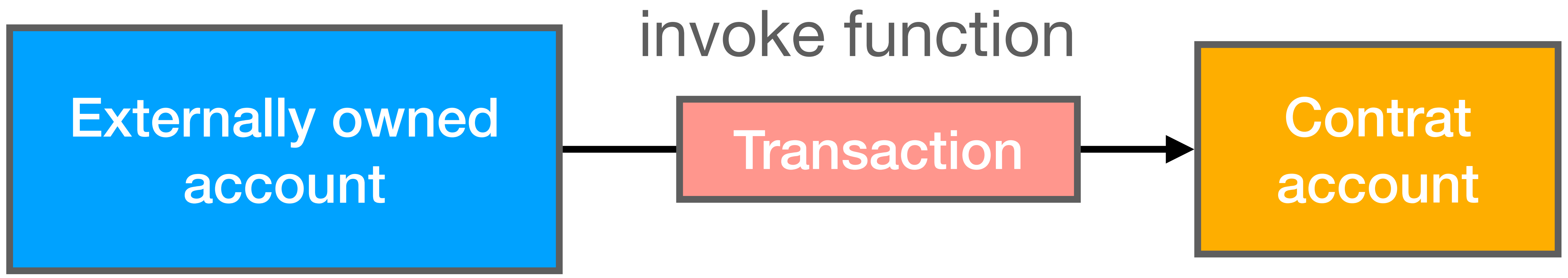
Transactions are used to transfer ether, invoke functions, and deploy new contracts.



Ethereum

Transactions and authenticaion

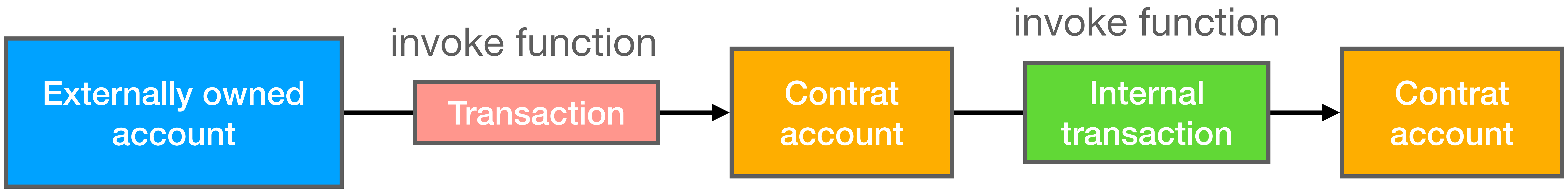
Transactions are used to transfer ether, invoke functions, and deploy new contracts.



Ethereum

Transactions and authenticaion

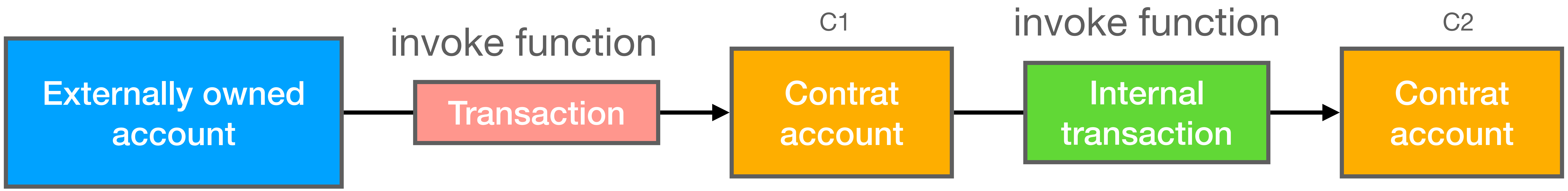
Transactions are used to transfer ether, invoke functions, and deploy new contracts.



Ethereum

Transactions and authenticaion

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



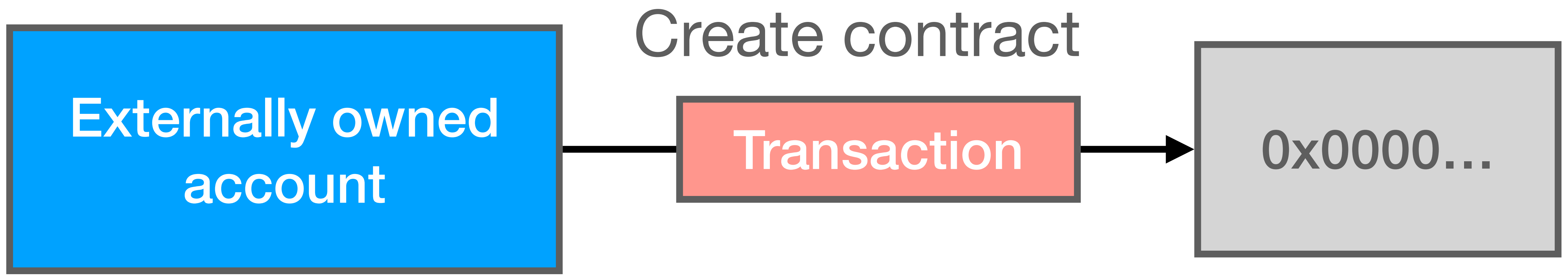
How does C1 know C2?

- C2 address supplied when deploying C1.
- C2 address supplied by transaction
- C2 address stored in contract state (can be updated)

Ethereum

Transactions and authenticaion

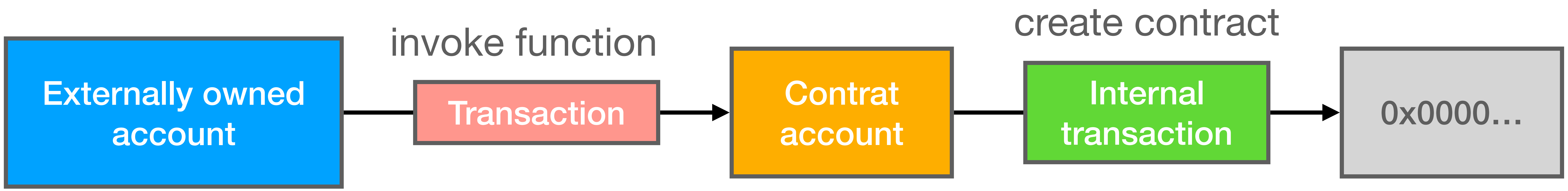
Transactions are used to transfer ether, invoke functions, and deploy new contracts.



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

Solidity view functions

View functions are read only

- Do not require fees
- Read contract state at one node

How can you know that a read from one node is correct?

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.

State of the blockchain (UTXO)
is not in the block.

Ethereum

Block structure

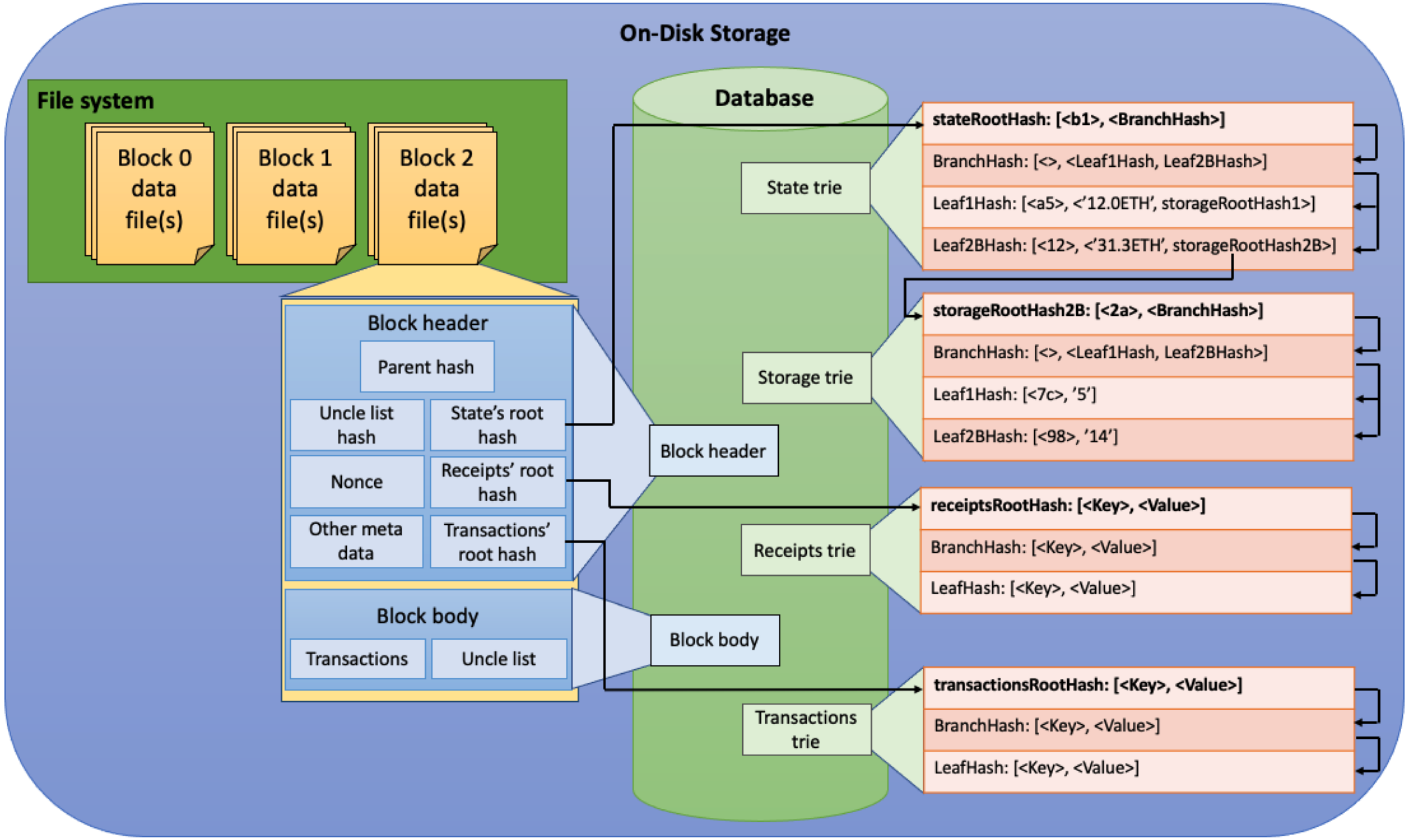
Header:

PrevBlockhash
Nonce
Timestamp

State root hash
Receipts root hash

Transaction data

Merkle tree


On-Disk Storage
File system
Block 0 data file(s)
Block 1 data file(s)
Block 2 data file(s)
Block header
Parent hash
Uncle list hash
State's root hash
Nonce
Receipts' root hash
Other meta data
Transactions' root hash
Block body
Transactions
Uncle list
Database
State trie
Storage trie
Block header
Receipts trie
Block body
Transactions trie
stateRootHash: [<b1>, <BranchHash>]
BranchHash: [<>, <Leaf1Hash, Leaf2BHash>]
Leaf1Hash: [<a5>, <'12.0ETH', storageRootHash1>]
Leaf2BHash: [<12>, <'31.3ETH', storageRootHash2B>]
storageRootHash2B: [<2a>, <BranchHash>]
BranchHash: [<>, <Leaf1Hash, Leaf2BHash>]
Leaf1Hash: [<7c>, '5']
Leaf2BHash: [<98>, '14']
receiptsRootHash: [<Key>, <Value>]
BranchHash: [<Key>, <Value>]
LeafHash: [<Key>, <Value>]
transactionsRootHash: [<Key>, <Value>]
BranchHash: [<Key>, <Value>]
LeafHash: [<Key>, <Value>]

Ethereum

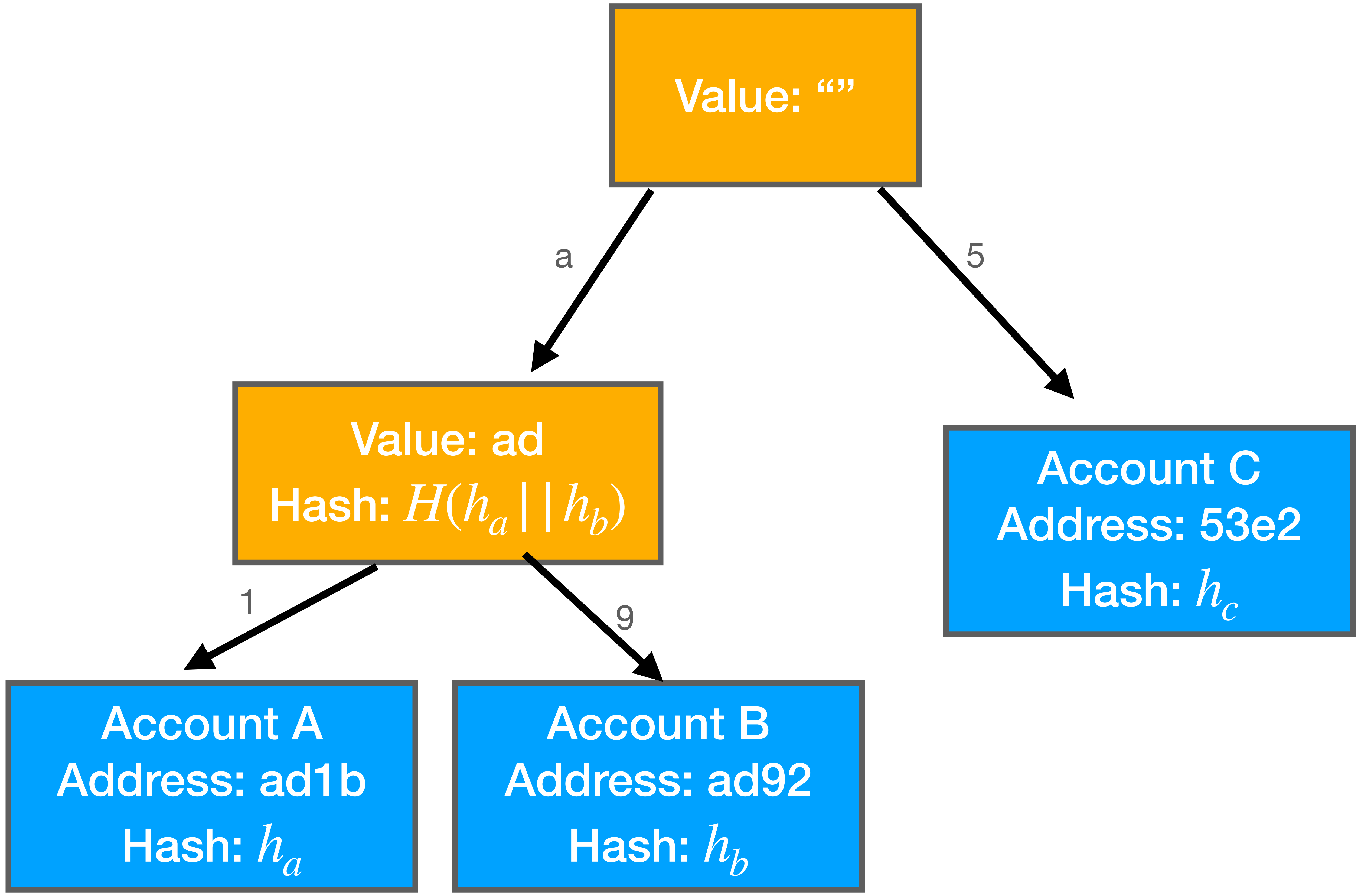
State trie

Stores accounts:

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

Trie:
Merkle tree that supports

```
update
lookup
proof
```



Root of state trie is in the block

Ethereum

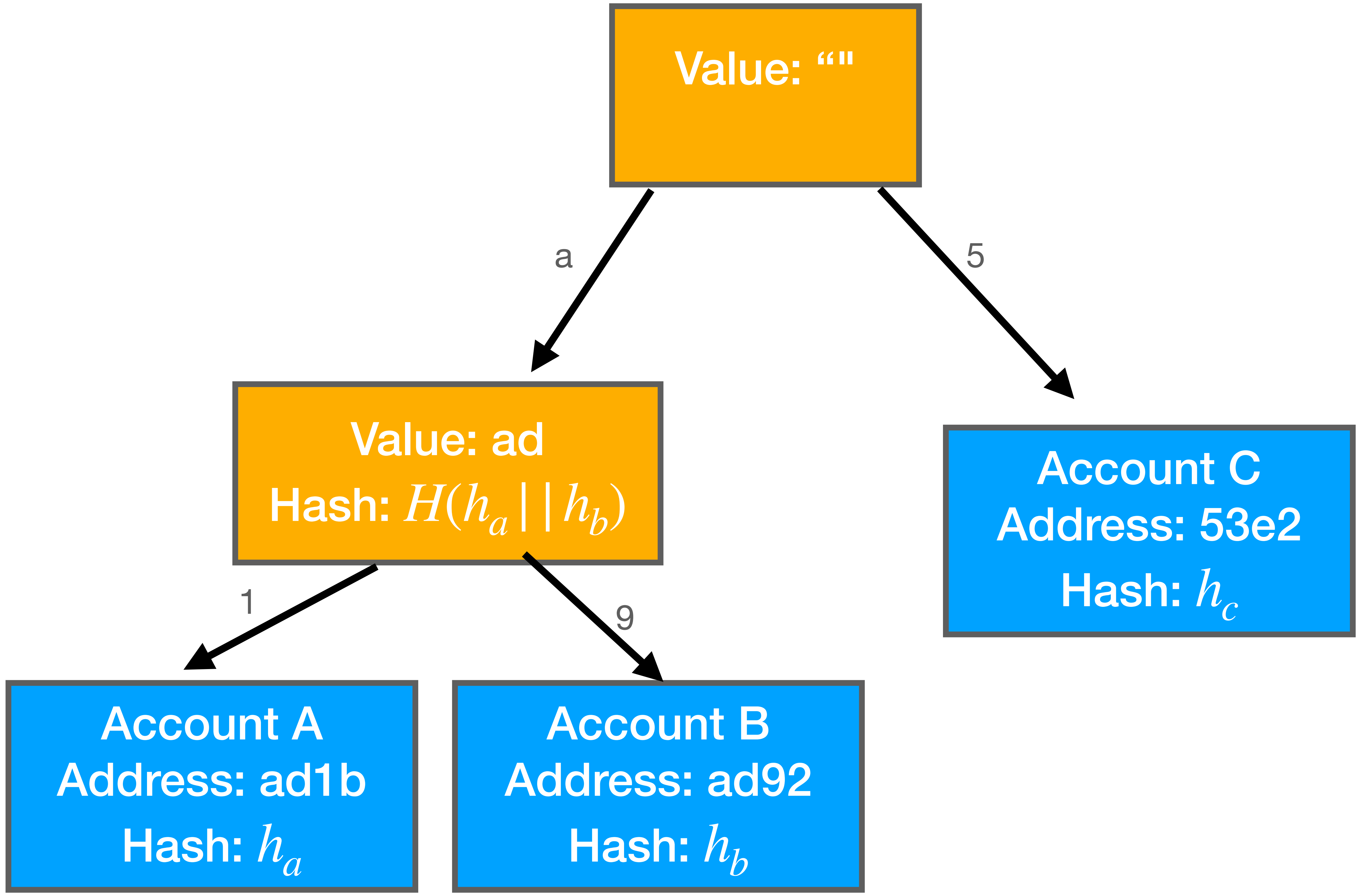
State trie

Stores accounts:

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

Trie:
Merkle tree that supports

```
update
lookup
proof
```



SC Variable changes
-> SC storage root changes
-> SC Account hash changes
-> State trie root changes

Ethereum

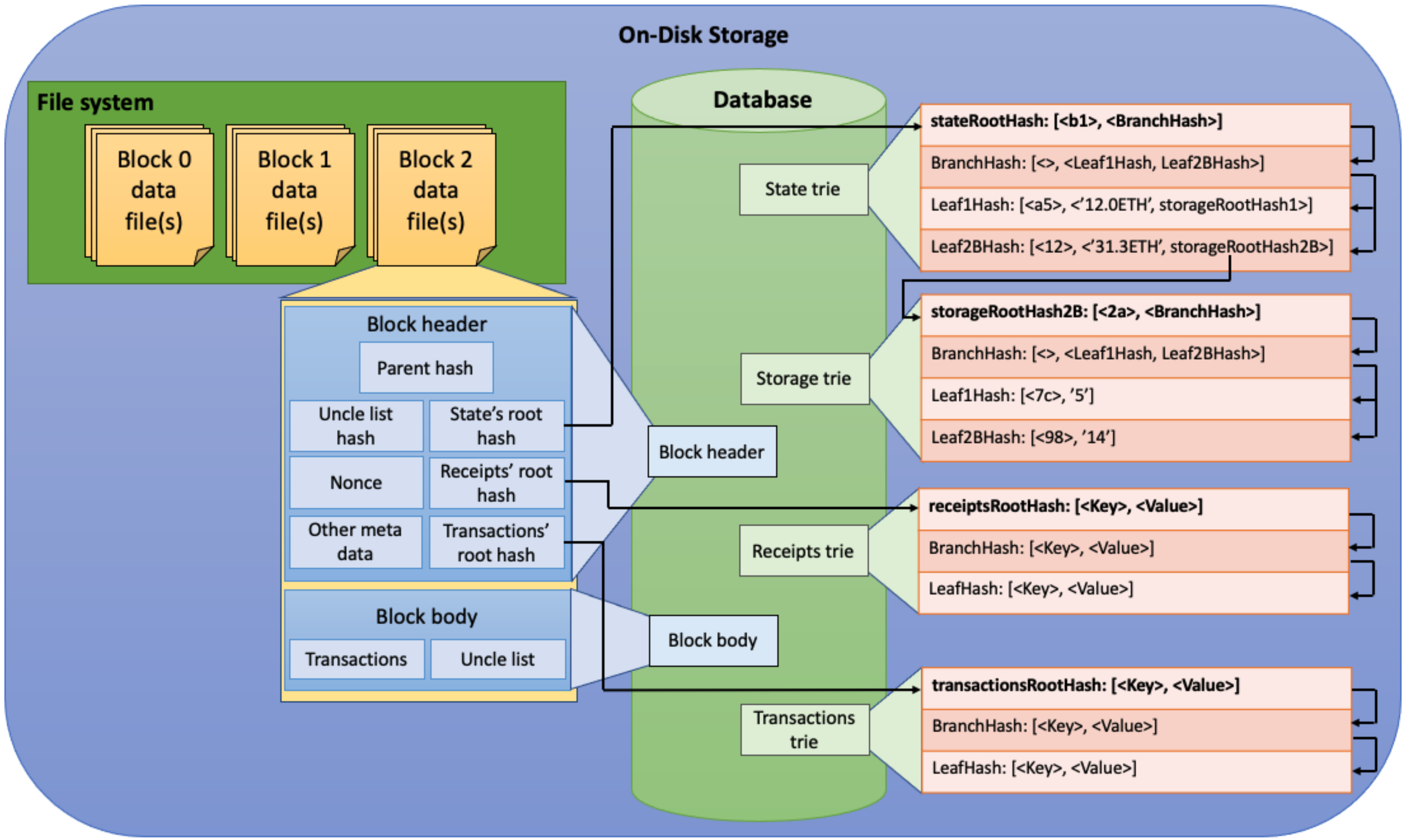
State trie

Stores accounts:

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

Trie:
Merkle tree that supports

```
update
lookup
proof
```



On-Disk Storage
File system
Block 0 data file(s)
Block 1 data file(s)
Block 2 data file(s)
Block header
Parent hash
Uncle list hash
State’s root hash
Nonce
Receipts’ root hash
Other meta data
Transactions’ root hash
Block body
Transactions
Uncle list
Database
State trie
Storage trie
Block header
Receipts trie
Block body
Transactions trie
stateRootHash: [<b1>, <BranchHash>]
BranchHash: [<>, <Leaf1Hash, Leaf2BHash>]
Leaf1Hash: [<a5>, <‘12.0ETH’, storageRootHash1>]
Leaf2BHash: [<12>, <‘31.3ETH’, storageRootHash2B>]
storageRootHash2B: [<2a>, <BranchHash>]
BranchHash: [<>, <Leaf1Hash, Leaf2BHash>]
Leaf1Hash: [<7c>, ’5’]
Leaf2BHash: [<98>, ’14’]
receiptsRootHash: [<Key>, <Value>]
BranchHash: [<Key>, <Value>]
LeafHash: [<Key>, <Value>]
transactionsRootHash: [<Key>, <Value>]
BranchHash: [<Key>, <Value>]
LeafHash: [<Key>, <Value>]

`StorageRoot` is the root of a different trie.

Ethereum

State trie

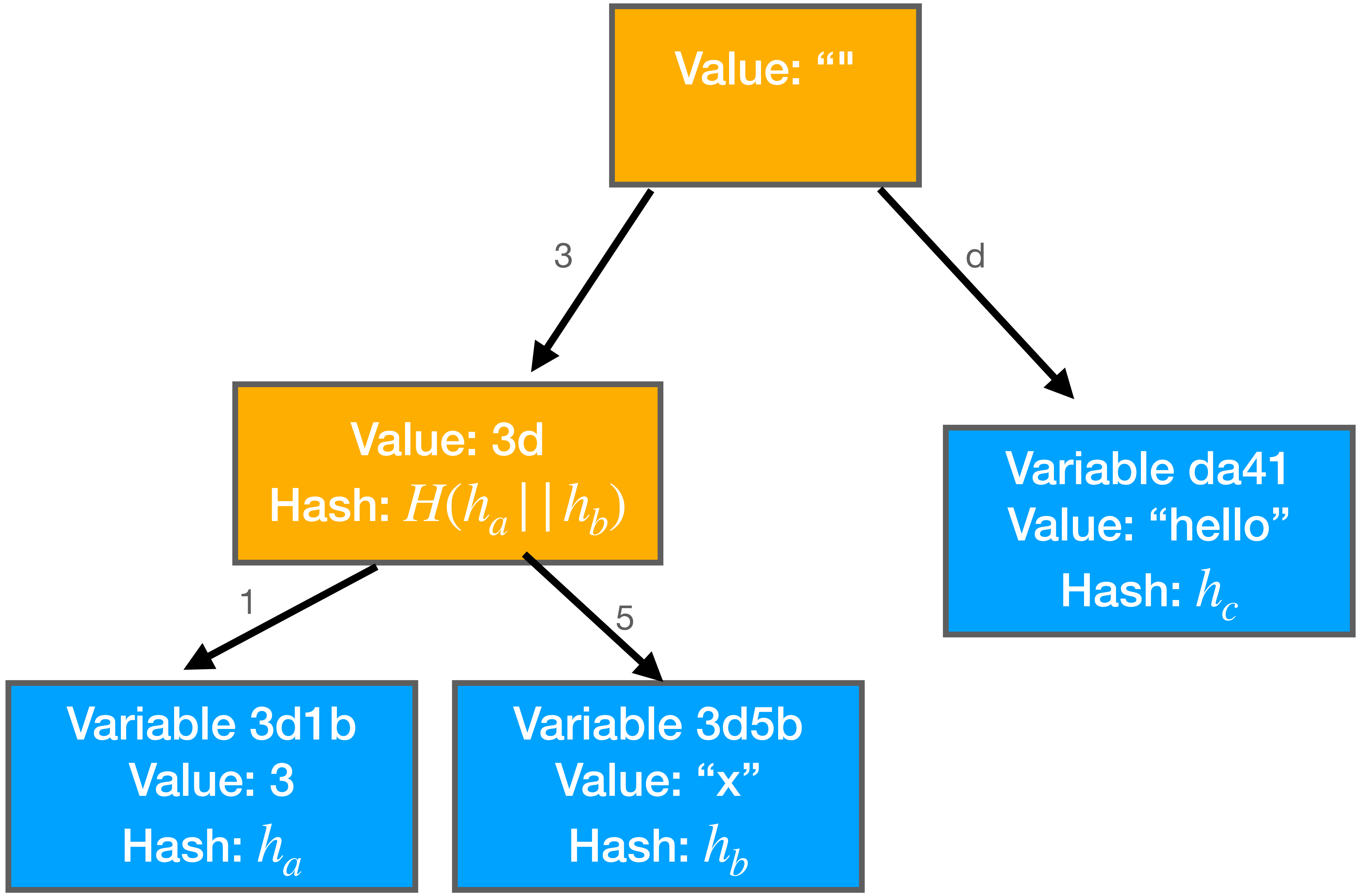
Stores accounts:

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

Trie:
Merkle tree that supports

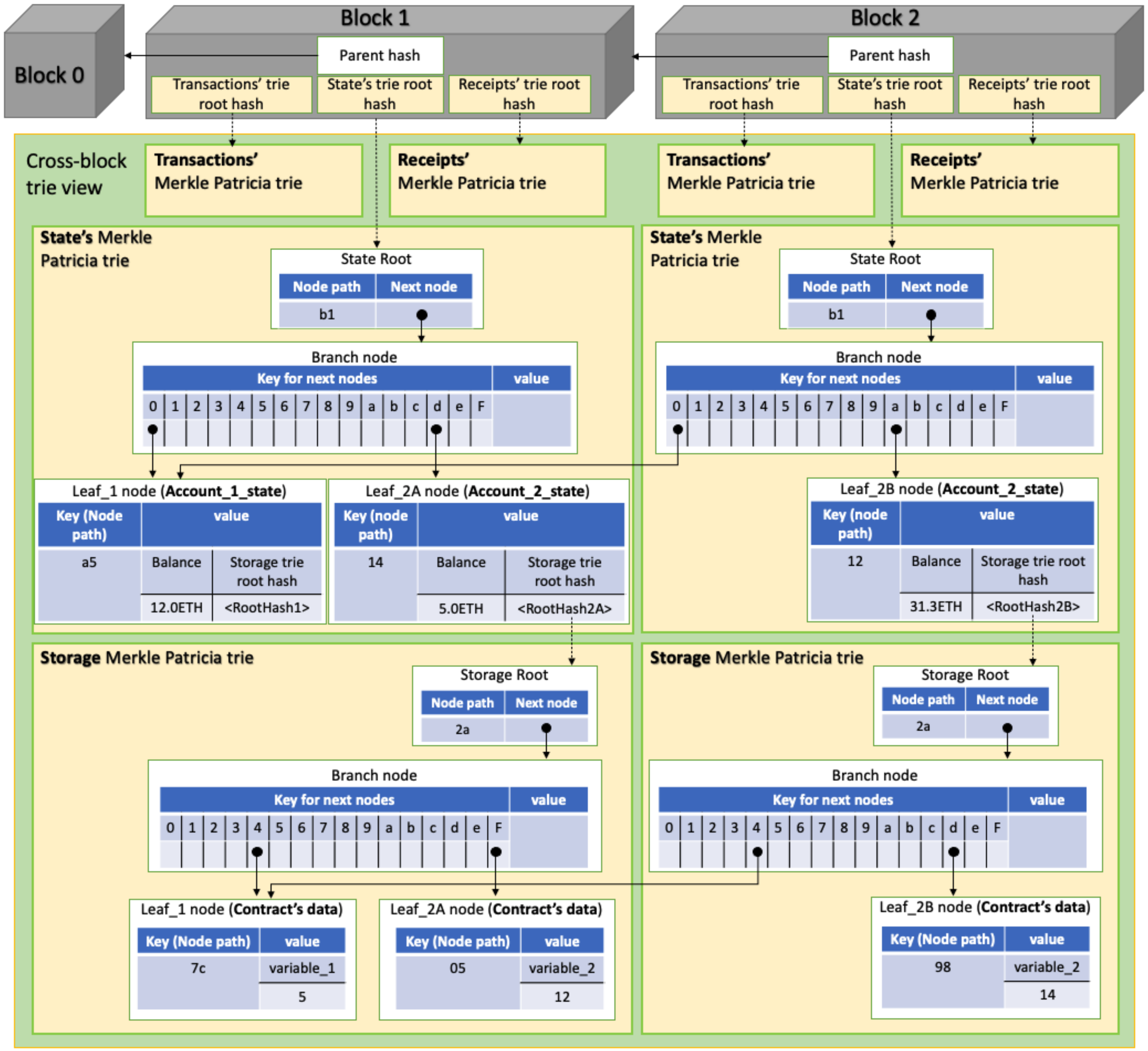
```
update
lookup
proof
```

`StorageRoot` is the root of a different trie.



Ethereum

State trie



Block 0

Block 1

Transactions' trie root hash	Parent hash State's trie root hash	Receipts' trie root hash

Block 2

Transactions' trie root hash	Parent hash State's trie root hash	Receipts' trie root hash

Cross-block trie view

Transactions' Merkle Patricia trie

Receipts' Merkle Patricia trie

Transactions' Merkle Patricia trie

Receipts' Merkle Patricia trie

State's Merkle Patricia trie

State Root

Node path	Next node
b1	●

Branch node

Key for next nodes																value
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	F	
●													●			

Leaf_1 node (**Account_1_state**)

Key (Node path)	value	
a5	Balance	Storage trie root hash
	12.0ETH	<RootHash1>

Leaf_2A node (**Account_2_state**)

Key (node path)	value	
14	Balance	Storage trie root hash
	5.0ETH	<RootHash2A>

State's Merkle Patricia trie

State Root

Node path	Next node
b1	●

Branch node

Key for next nodes																value
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	F	
●										●						

Leaf_2B node (**Account_2_state**)

Key (node path)	value	
12	Balance	Storage trie root hash
	31.3ETH	<RootHash2B>

Storage Merkle Patricia trie

Storage Root

Node path	Next node
2a	●

Branch node

Key for next nodes																value
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	F	
				●											●	

Leaf_1 node (**Contract's data**)

Key (Node path)	value
7c	variable_1
	5

Leaf_2A node (**Contract's data**)

Key (Node path)	value
05	variable_2
	12

Storage Merkle Patricia trie

Storage Root

Node path	Next node
2a	●

Branch node

Key for next nodes																value
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	F	
				●									●			

Leaf_2B node (**Contract's data**)

Key (Node path)	value
98	variable_2
	14

Ethereum

Read contract state

1. ask trusted node

2. receive inclusion proof for

   stateRoot: storageTrie
   account state: stateTrie

   and block header

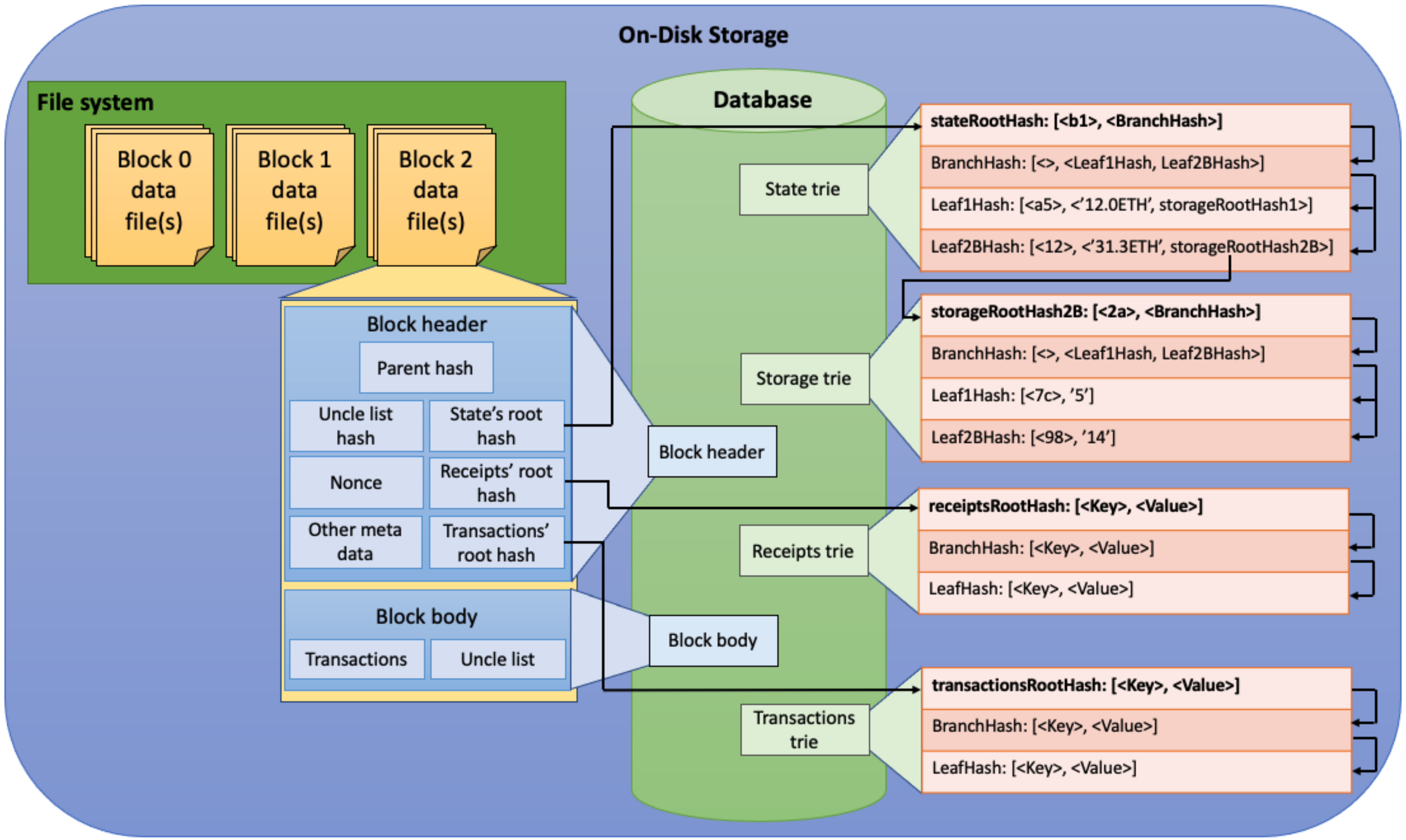
Ethereum

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`.


On-Disk Storage
File system
Block 0 data file(s)
Block 1 data file(s)
Block 2 data file(s)
Block header
Parent hash
Uncle list hash
State's root hash
Nonce
Receipts' root hash
Other meta data
Transactions' root hash
Block body
Transactions
Uncle list
Database
State trie
Storage trie
Block header
Receipts trie
Block body
Transactions trie
stateRootHash: [<b1>, <BranchHash>]
BranchHash: [<>, <Leaf1Hash, Leaf2BHash>]
Leaf1Hash: [<a5>, <'12.0ETH', storageRootHash1>]
Leaf2BHash: [<12>, <'31.3ETH', storageRootHash2B>]
storageRootHash2B: [<2a>, <BranchHash>]
BranchHash: [<>, <Leaf1Hash, Leaf2BHash>]
Leaf1Hash: [<7c>, '5']
Leaf2BHash: [<98>, '14']
receiptsRootHash: [<Key>, <Value>]
BranchHash: [<Key>, <Value>]
LeafHash: [<Key>, <Value>]
transactionsRootHash: [<Key>, <Value>]
BranchHash: [<Key>, <Value>]
LeafHash: [<Key>, <Value>]

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

Actually: Gas price is divided by:
Base price + Tip
Base price is burned
Tip is given to validators

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.
- Blocks have maximum amount of gas.

Ethereum

Gas - London upgrade 2021

Gas price is divided in Base price + Tip

- Base price is burned
- Tip is given to the validators

Blocks can be bigger than target size, but
included transactions have to pay a larger base price