Blockchain Technology

Introduction to Ethereum

Ashutosh Bhatia

BITS Pilani

ashutosh.bhatia@pilani.bits-pilani.ac.in

Overview

- What is Ethereum
- Compared to Bitcoin
- Components of a public Blockchain
- Ethereum: A general purpose blockchain
- Ethereum components
- Ethereum and Turing Completeness

What is Ethereum

- **Often described as “World wide computer operating under consensus”**
- **Computer Science Perspective**
  - Ethereum is a deterministic, but practically unbounded state machine, consisting of a globally accessible singleton state and a virtual machine that applies changes to that state according to consensus rules.
- **Practical Perspective**
  - Open source, globally decentralized computing infrastructure that executes programs called smart contracts and uses blockchain to synchronize and store the system’s state change along with a cryptocurrency called ether, to meter and constrain resource costs
- **Developer Perspective**
  - A platform that enables developers to built decentralized applications with build-in business logic, providing availability, auditability, transparency and neutrality, and reducing certain counterparty risks.

Ethereum Vs BITCOIN (Commonalities)

- **Open Blockchains:** Trustless, Immutable, Uncensorable, No central point of failure
- **Decentralized Identify:** Pseudo anonymous identify using public key
- **Consensus algorithm:** POW based Byzantine Fault tolerant consensus algorithm (mining power proportional to the computer power)
- **Cryptographic primitives:** use of cryptographic primitives such as digital signatures and hashes, and a digital currency (ether).
- **P2P Network:** peer-to-peer network connecting participants

Ethereum Vs BITCOIN (Differences)

BITCOIN

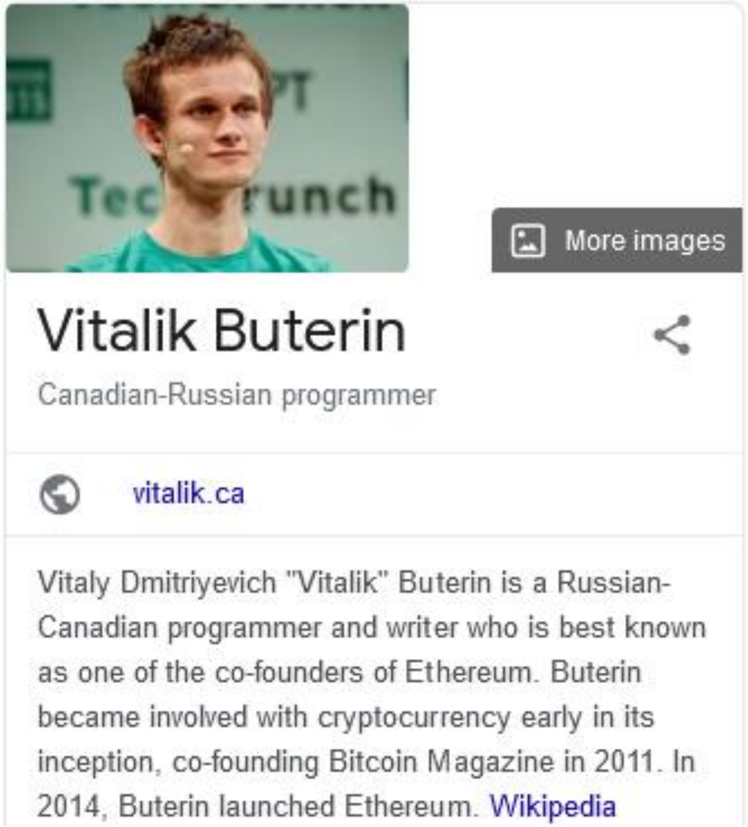
1. A blockchain for cryptocurrency
2. Asset: Bitcoins as currency
    - Primary purpose of the blockchain
3. Simple and robust
4. primitive scriptive language, not Turing complete
5. UTXO-based

Ethereum

1. General purpose programmable blockcahin
2. Asset: Ether as utility currency
    - Fund computations
3. Complex and feature rich
4. Turing complete scripting language
5. Account based

Birth of Ethereum

- Conceived at a time when people recognized the power of Bitcoin model
- Trying to move beyond cryptocurrency:
  - a similar model with more generalized applications
- Developers Confusion:
  - Either build on top of BITCOIN by trying to find workarounds and live with the constraints imposed by the transaction type, data types and size of the data storage. Design anything else needed as off-chain, which could completely negate the very reason of using the blockcahins
- In December 2013, a young programmer and BITCOIN enthusiast, Vitalik Burterin started sharing a white paper outline the idea behind **Ethereum**



Vitalik Buterin

Canadian-Russian programmer

vitalik.ca

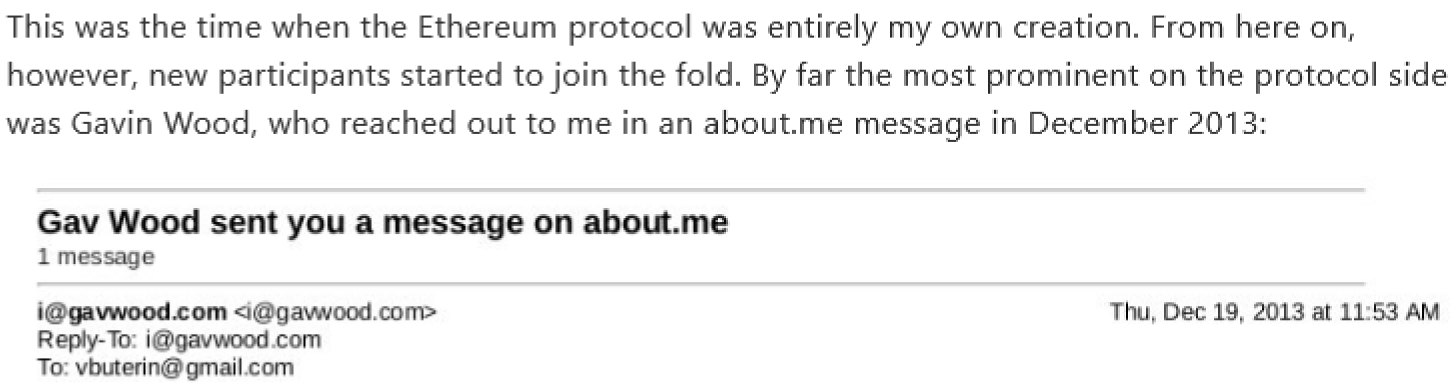
Vitaly Dmitriyevich "Vitalik" Buterin is a Russian-Canadian programmer and writer who is best known as one of the co-founders of Ethereum. Buterin became involved with cryptocurrency early in its inception, co-founding Bitcoin Magazine in 2011. In 2014, Buterin launched Ethereum. Wikipedia

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

https://ethereum.org/en/whitepaper/

Birth of Ethereum

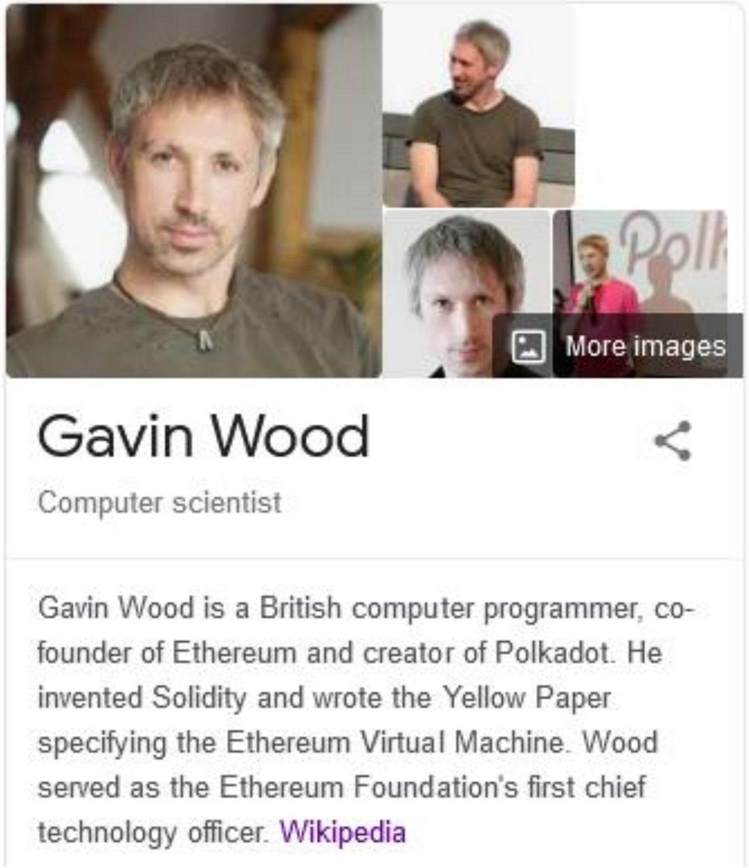
- A few dozen people saw the early draft and offered feedback helping Vitalink to evolve the proposal.
- Gavin Wood, a computer programmer was one of the first people to reach out Vitalink and offered his C++ programming skills in the creation of Ethereum.
- Vitalink on Galvin contribution to Ethereum



This was the time when the Ethereum protocol was entirely my own creation. From here on, however, new participants started to join the fold. By far the most prominent on the protocol side was Gavin Wood, who reached out to me in an about.me message in December 2013:

Gav Wood sent you a message on about.me
1 message

i@gavwood.com <i@gavwood.com> Thu, Dec 19, 2013 at 11:53 AM
Reply-To: i@gavwood.com
To: vbuterin@gmail.com



Gavin Wood

Computer scientist

Gavin Wood is a British computer programmer, co-founder of Ethereum and creator of Polkadot. He invented Solidity and wrote the Yellow Paper specifying the Ethereum Virtual Machine. Wood served as the Ethereum Foundation's first chief technology officer. Wikipedia

https://vitalik.ca/general/2017/09/14/prehistory.html

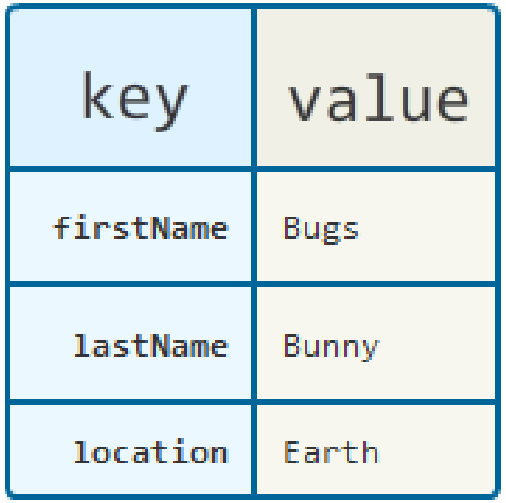
Ethereum’s Four Stages of Development

- ***Block #0 : Frontier—***The initial stage of Ethereum, lasting from July 30, 2015, to March 2016.
  - ***Block #200,000: Ice Age—***A hard fork to introduce an exponential difficulty increase, to motivate a transition to PoS when ready.
- ***Block #1,150,000 Homestead—***The second stage of Ethereum, launched in March 2016.
  - ***Block #1,192,000 DAO—***A hard fork that reimbursed victims of the hacked DAO contract and caused Ethereum and Ethereum Classic to split into two competing systems.
  - ***Block #2,463,000 Tangerine Whistle—***A hard fork to change the gas calculation for certain I/O heavy operations
  - ***Block #2,675,000: Spurious Dragon—*** A hard fork to address more DoS attack vectors, and another state clearing. Also, a replay attack protection mechanism.
- ***Block #4,370,000 Metropolis Byzantium—*****Metropolis** is the third stage of Ethereum
- **Serenity** Ethereum 2.0, also known as **Serenity** or ETH 2.0, is an upgrade to Ethereum on a number of levels. Its primary objective is to increase Ethereum's capacity for transactions, reduce fees and make the network more sustainable.

Ethereum: A general-purpose Blockchain

- Unlike BITCOIN which tracks the state transitions for “Currency Ownership”, the Etthreum tracks the state transition of **general purpose data expressible as key-value pair**

- Similar to RAM model of a computer it stored both code and data and uses blockchain to track the changes in this in stored data.



key	value
firstName	Bugs
lastName	Bunny
location	Earth

- Like a general purpose computer, etherum can load the code in the state machine, run that code and storing the resulting state change in the blockchin

- Two critical differences with general purpose computers

  Governed by rules of consensus.

  State is distributed globally.

Ethereum is Turing Complete

- In 1936 Alan Turing created a mathematical model for the computer consisting of a state machine that manipulates symbols by reading an writing on a sequential memory (Tape)
- Using this model he provided a proof to the question “Whether all problems are solvable” (universal computability)
- He proved that there are classes of problems that are uncomptable, specially the **halting problem**
- A system is said to be Turing complete if it can be used to simulate a Turing machine.

Ethereum groundbreaking contribution is to combine a general purpose computing architecture of a stored program computer with a decentralized blockchain thre by creating a distributed single state world computer.

Ethereum’s Components

- P2P Network: Etherreum runs on Ethereum main network, which is addressable on TCP port 30303 and runs a protocol called DEVp2p

devP2P

Cross platform peer-to-peer client library, for desktops and mobiles!

The C++ standalone library for establishing direct and secured P2P VPN connections. It can perform file transfers, port forwardings, full network redirection and more...

Once connected, use it for whatever purpose you need - share you screens using favorite application, open remote files, transfer data, send messages....

DevP2P General Features

- Establishes secure P2P VPN over internet
  - Connection is established through UDP or TCP protocol with remote devP2P, virtualy anywhere

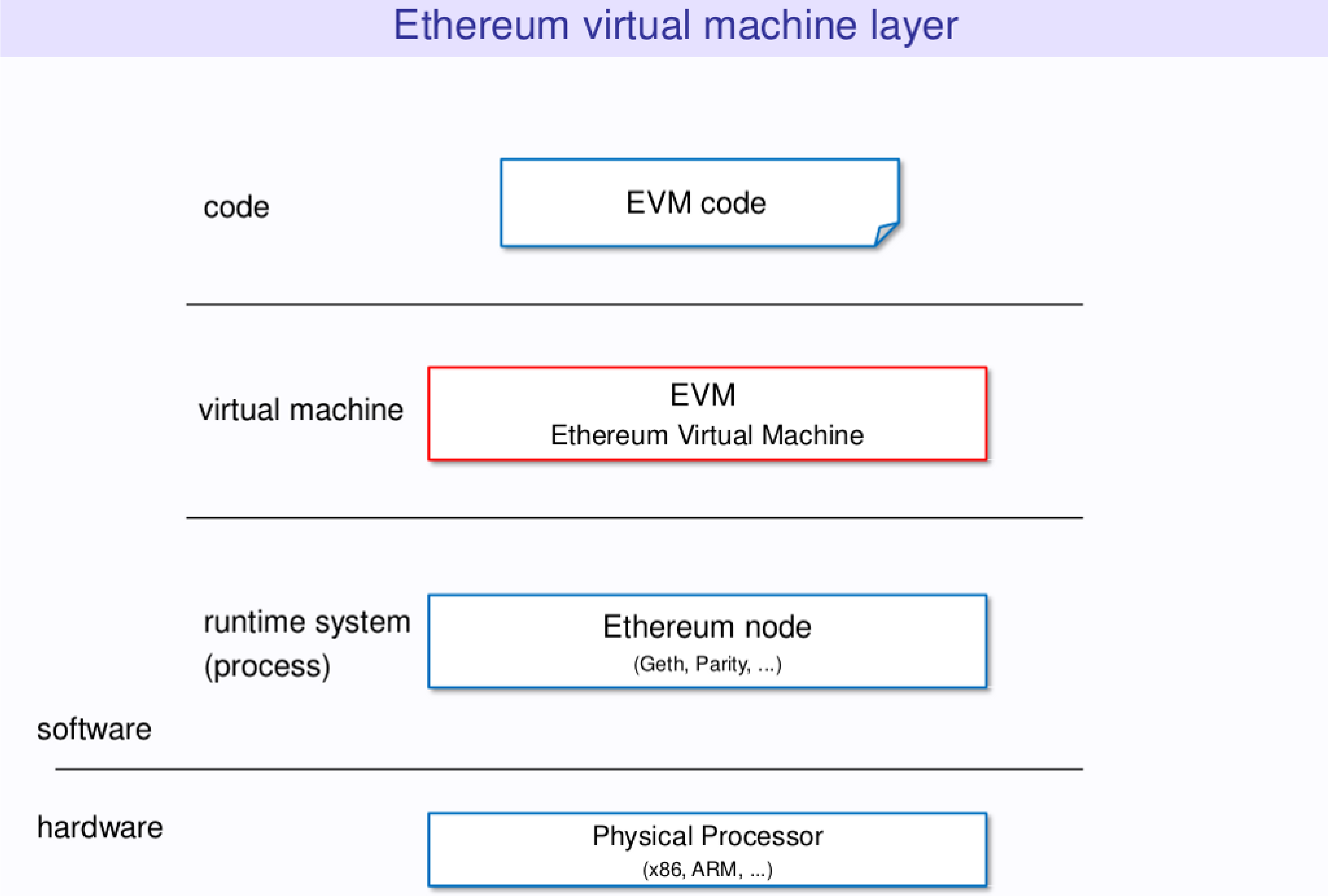
- Provides LAN over internet
  - When network redirect is used, local network packets for outgoing peers are captured, transferred through devP2P to remote peer, and there they are provided to the system just as they have arrived from network cable.

- Share screen, data, files
  - You can fire up tools to view remote desktop through firewalls, transfer files and data.

- Forwards ports.. Sends files and messages.. Redirects network..
  - After connection is established, there are 1024 channels to use.
    Use them to SendMessage, SendFile, StartForwading.

EVM

- state transitions are processed by the EVM a stack based virtual machine that executes byte code.
- EVM programs called “smart contracts” are written in high level programming language and compiled to byte code for execution in EVM.



Ethereum virtual machine layer

code: EVM code

virtual machine: EVM
Ethereum Virtual Machine

runtime system (process): Ethereum node (Geth, Parity, ...)

software

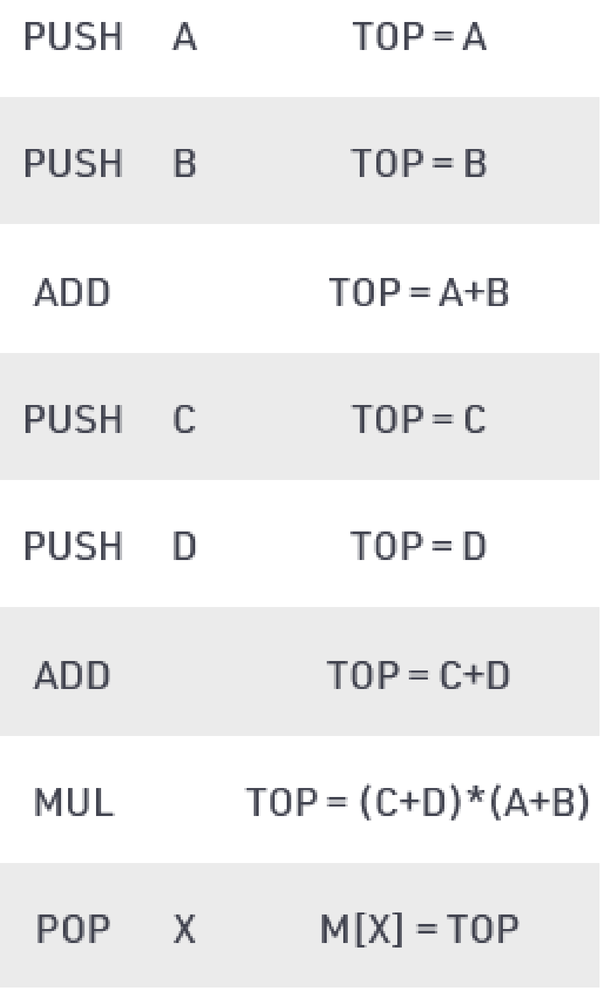
hardware: Physical Processor (x86, ARM, ...)

Stack based computer

- A stack based computer do not use address field in instruction.

To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

- Example
  - Expression: X = (A+B)*(C+D)
  - Postfixed : X = AB+CD+*



PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)
POP	X	M[X] = TOP

Why Stack based Machine

- Traditionally, virtual machine implementers have favored stack-based architectures over register-based due to 'simplicity of VM implementation'
- Ease of writing a compiler back-end
- executables for stack architecture are invariably smaller than executables for register architectures.

Why EVM and not a JVM

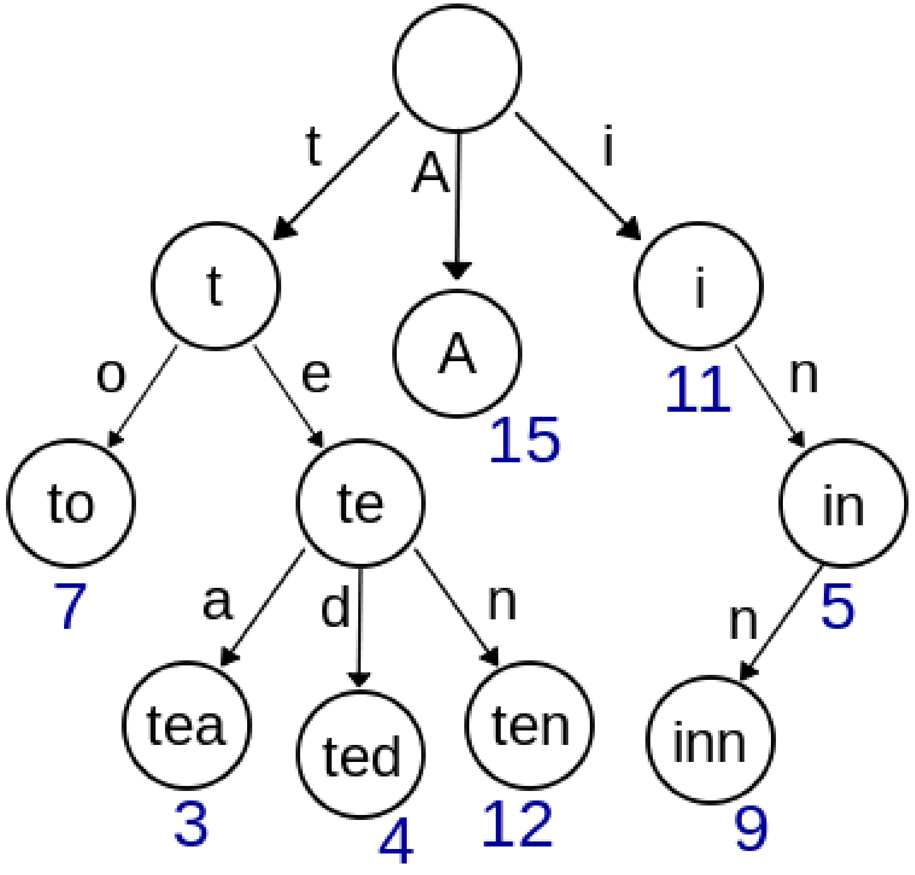
- **complex and voluminous** : a single java method can have a size up to 64KB so such VM or language isn't space saving.
- **useless features and security concerns**: Network access, I/O stream, File W/R etc => big security issues.
  - think of it, you can write a code which ping (of death) another machine or access protected files or even steal the miner's private keys.
  - file "write/read" feature could break the whole system's security. so we'll need to get rid of all these features, which is a hard task to achieve for a licensed VM.
  - We need to remember that a Blockchain VM should be **isolated** without the capacity to communicate with the external environment.
- Imagine you have a Java bytecode with a rand(), what would be the result and how to reach the consensus then?.
- **Weak DDoS resistance** : how to set a gas-like system in a complex VM like Java VM? .
- **JAVA VM is a licensed Sun product**, so you can't customize it to integrate it to the Ethereum's environment
  - (for example how would you calculate gas cost to avoid Dos attacks?)? to overcome this problem you need to write your own Java VM which is a complex task read

Ethereum’s Components: Data Structures

- Ethereum database is stored locally on each node as a database (usually Google’s LevelDB), which contains transactiions and system state in a serialized hased data structure called **Merkle Patricia Trie (MPT).**
- Basically, MPT is a combination of Patricia trie and Merkle tree, with few additional optimizations that fit the characteristics of Ethereum.
- Patricia trie is a data structure which is also called Prefix tree, radix tree or trie.
- Trie uses a key as a path so the nodes that share the same prefix can also share the same path.
- This structure is fastest at finding common prefixes, simple to implement, and requires small memory.
- Thereby, it is commonly used for implementing routing tables, systems that are used in low specification machines like the router.

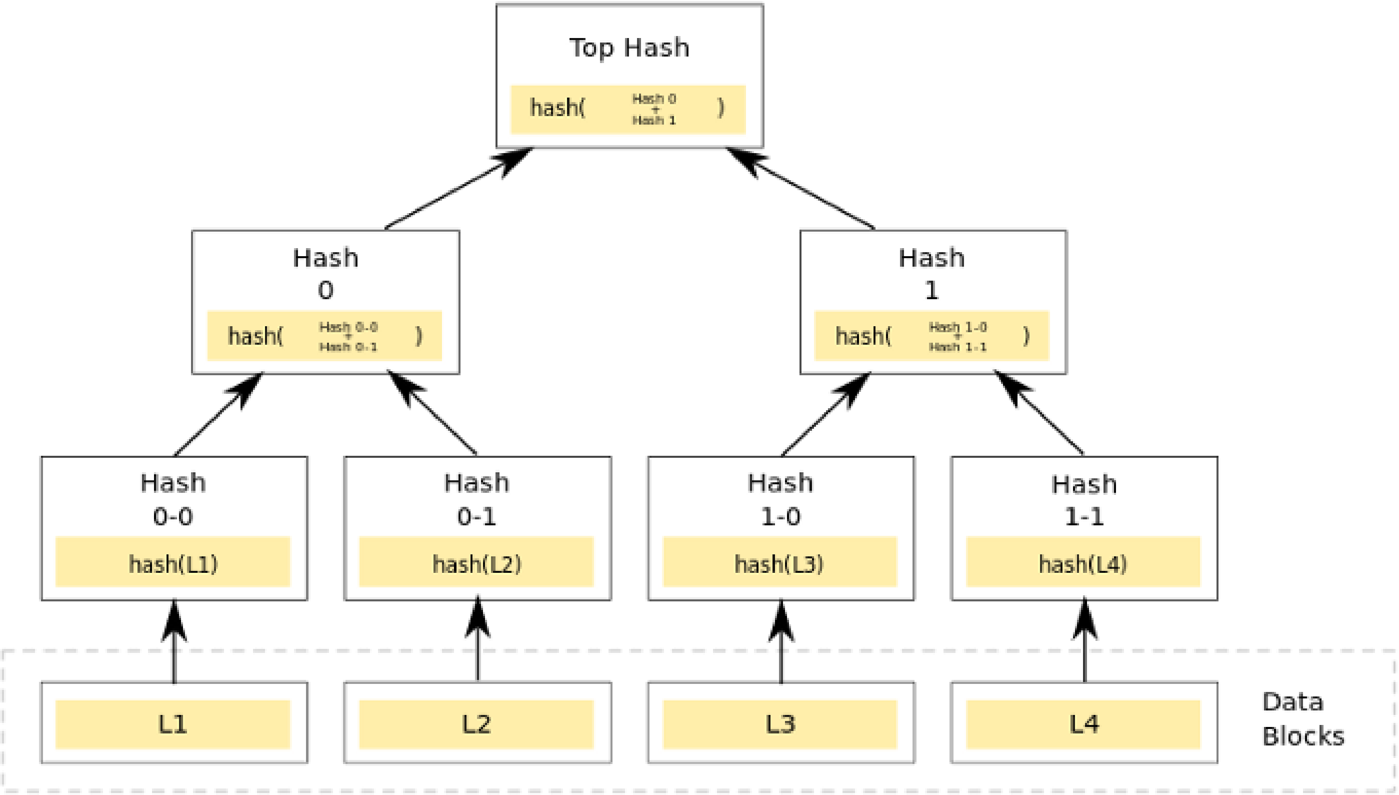
Patricia Trie: Example

- A trie for keys "A", "to", "tea", "ted", "ten", "i", "in", and "inn". Each complete English word has an arbitrary integer value associated with it.

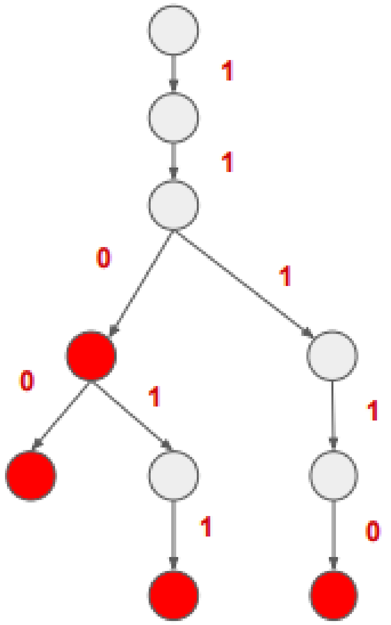


Merkle Tree

Merkle tree is a tree of hashes. Leaf nodes store data. Parent nodes contain their children’s hash as well as the hashed value of the sum of their children’s hashes. Since all the nodes except for leaf nodes contain a hash, the Merkle tree is also known as a hash tree.

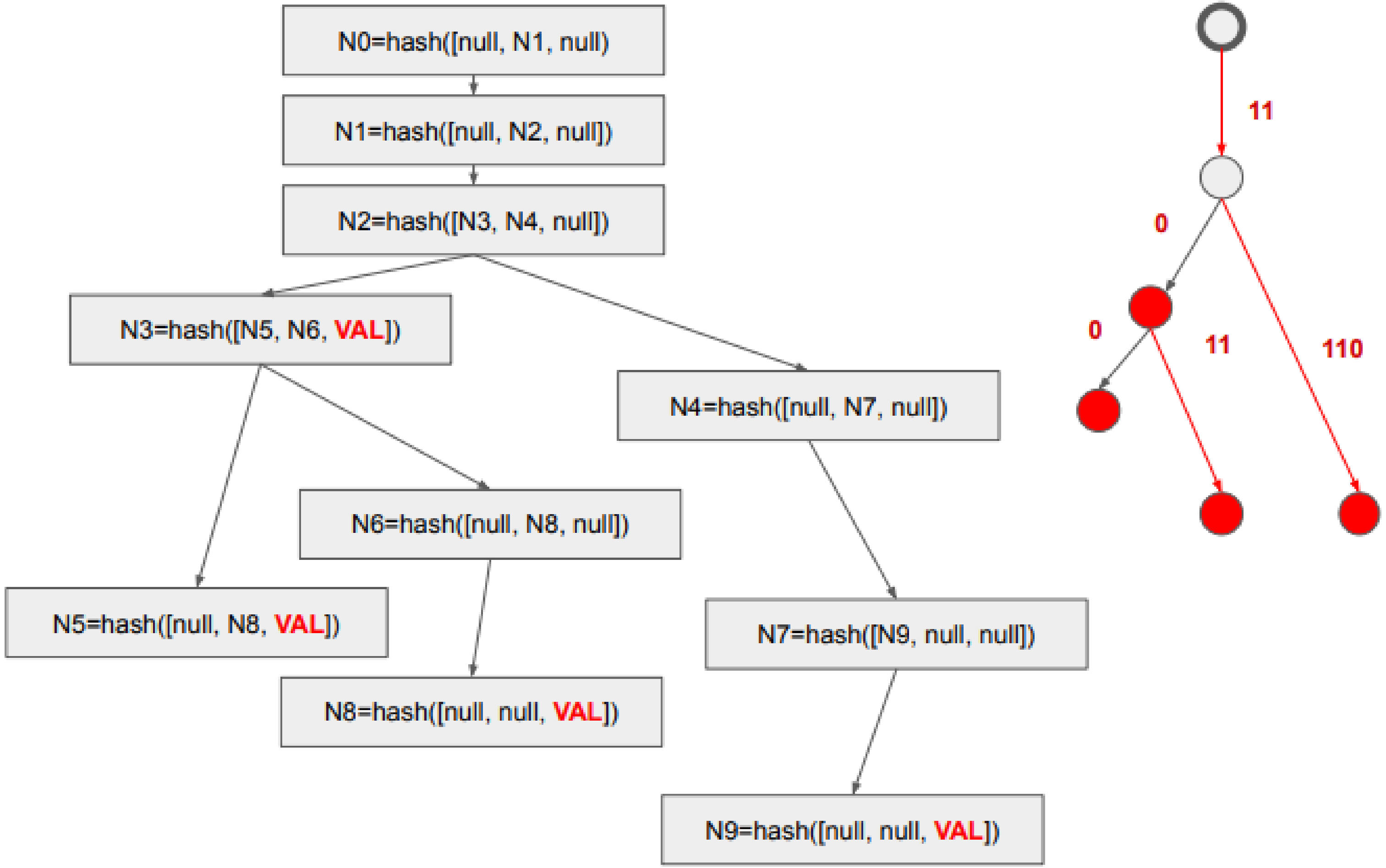


Prefix, Merkle and Petricia Tree



Keys:

110
1100
11011
11110



N0=hash([null, N1, null)

N1=hash([null, N2, null])

N2=hash([N3, N4, null])

N3=hash([N5, N6, VAL])

N4=hash([null, N7, null])

N6=hash([null, N8, null])

N5=hash([null, N8, VAL])

N7=hash([N9, null, null])

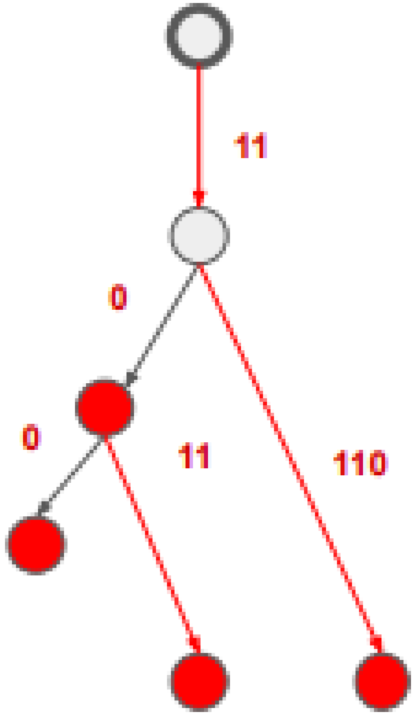
N8=hash([null, null, VAL])

N9=hash([null, null, VAL])

Merkle Patricia Trie

The Merkle Patricia Trie defines three types of nodes

- Branch – a 17-item node [$i0$,$i1$, ...,$i15$, $value$]
- Extension – A 2-item node [$path$, $value$]
- Leaf – A 2-item node [$path$, $value$]



Keys:

110
1100
11011
11110

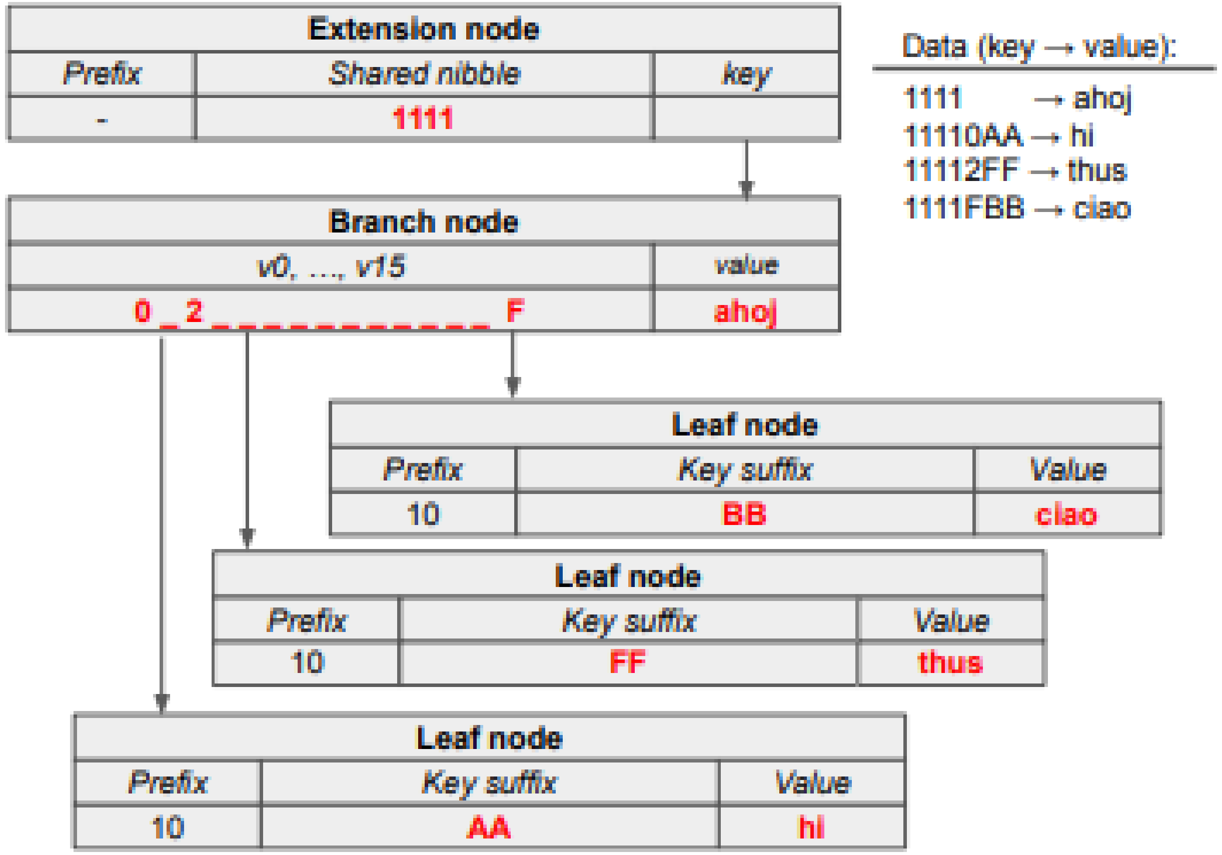
Merkle Patricia Trie

1. All the keys have the same prefix 1111 and for this reason the extension node is created as a root

2. Then the keys start to differ, which is captured by creating the branch node. This node branches for characters at the same position of all keys. In particular, branches are created for characters ‘0’, ‘2’, and ‘*F* ’.

3. The key having solely ‘1111’ terminates here, which means that the value for this key is stored in this branch node.

4. Rest of the keys form leaf nodes as there is no more branching. The leaf nodes store suffixes of each key and store values for the keys as well.



Extension node

Prefix	*Shared nibble*	*key*
-	1111	

Branch node

v0, …, v15	*value*
0 _ 2 _ _ _ _ _ _ _ _ _ _ _ _ F	ahoj

Leaf node

Prefix	*Key suffix*	*Value*
10	BB	ciao

Leaf node

Prefix	*Key suffix*	*Value*
10	FF	thus

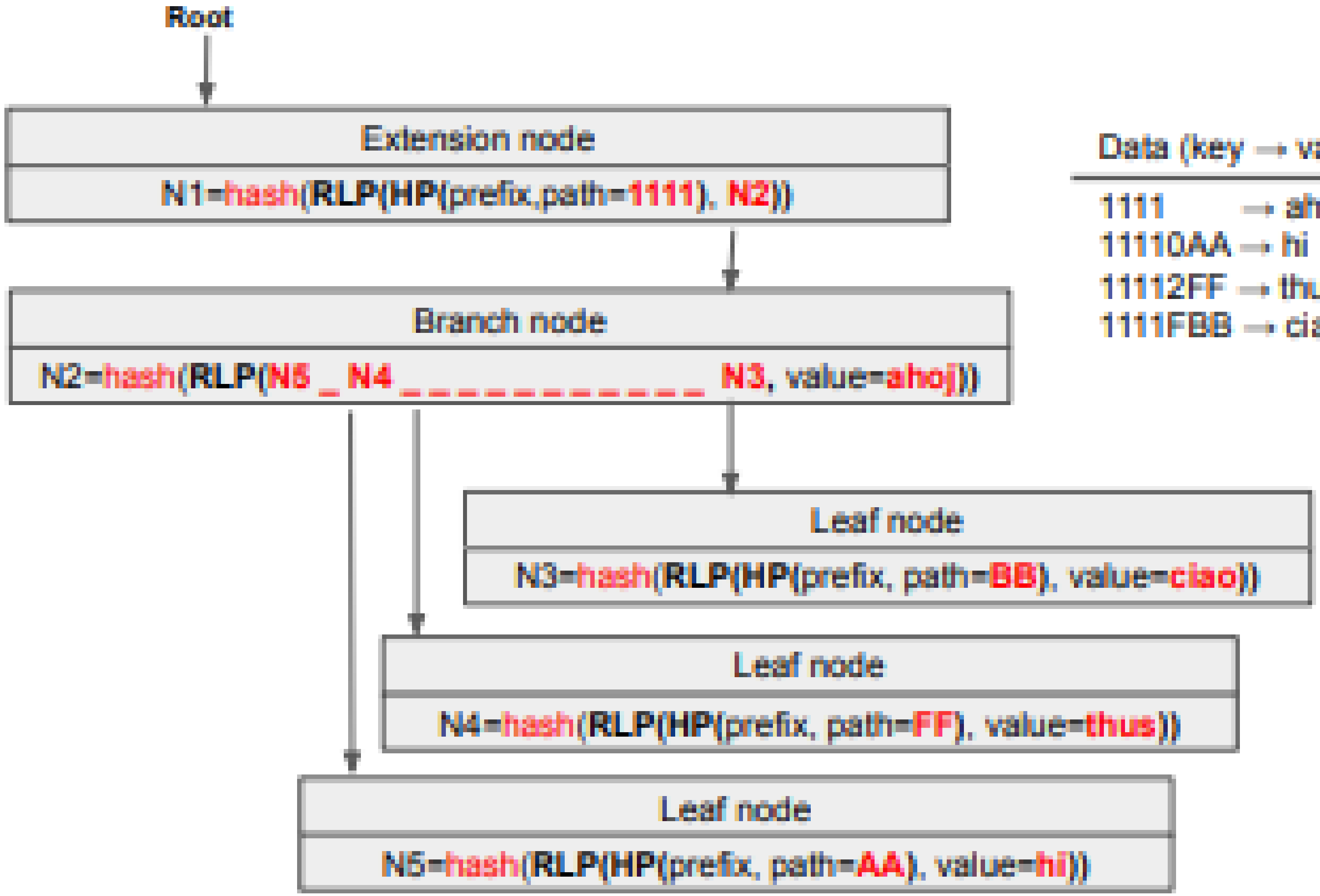
Leaf node

Prefix	*Key suffix*	*Value*
10	AA	hi

Data (key → value):

1111 → ahoj
11110AA → hi
11112FF → thus
1111FBB → ciao

Ethereum encoded merkle petricia Tree



Root

Extension node

N1=hash(RLP(HP(prefix,path=1111), N2))

Branch node

N2=hash(RLP(N5 _ N4 _ _ _ _ _ _ _ _ _ _ _ N3, value=ahoj))

Leaf node

N3=hash(RLP(HP(prefix, path=BB), value=ciao))

Leaf node

N4=hash(RLP(HP(prefix, path=FF), value=thus))

Leaf node

N5=hash(RLP(HP(prefix, path=AA), value=hi))

Recursive Length Prefix

HP: Hex Prefix Encoding

Data (key → value):

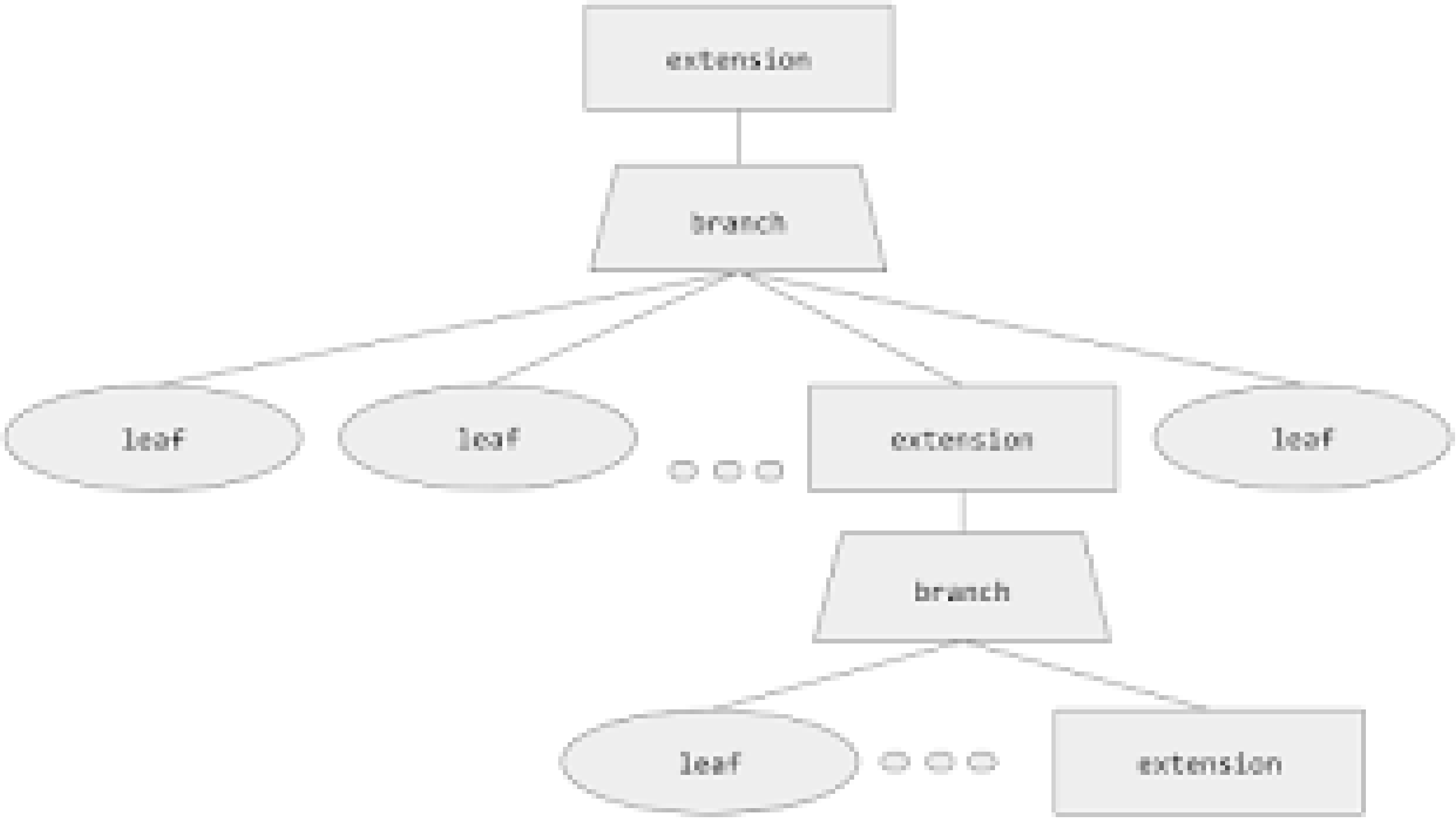
1111	→ ahoj
11110AA	→ hi
11112FF	→ thus
1111FBB	→ ciao

Prefix	Payload	Node Type	Path Length
[0, 0]	$[x_1x_2][x_3x_4]$	Extension	Even
$[1, x_1]$	$[x_2x_3][x_4x_5]$	Extension	Odd
[2, 0]	$[x_1x_2][x_3x_4]$	Leaf	Even
$[3, x_1]$	$[x_2x_3][x_4, x_5]$	Leaf	Odd

Merkle Patricia Trie

- In the MPT, there is one more type of nodes apart from the branch nodes and the leaf nodes. They are extension nodes.

- An extension node is an optimized node of the branch node.

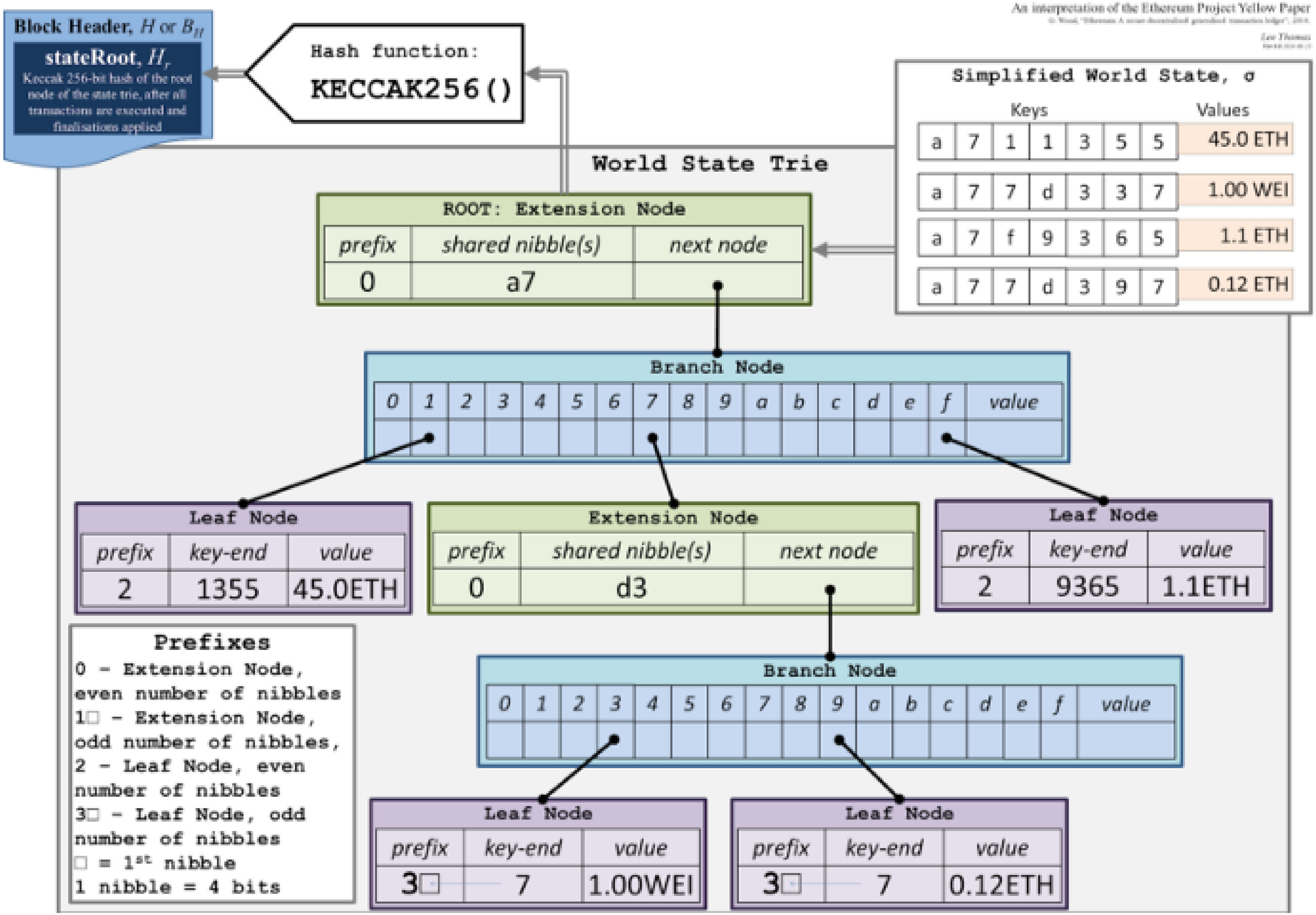


Ethereum Modified Merkle-Paricia-Trie System

An interpretation of the Ethereum Project Yellow Paper

G. Wood, "Ethereum: A secure decentralised generalised transaction ledger", 2014.

Lee Thomas



Block Header, H or B_H
stateRoot, H_r
Keccak 256-bit hash of the root node of the state trie, after all transactions are executed and finalisations applied
Hash function:
KECCAK256()
World State Trie

Simplified World State, σ

Keys							Values
a	7	1	1	3	5	5	45.0 ETH
a	7	7	d	3	3	7	1.00 WEI
a	7	f	9	3	6	5	1.1 ETH
a	7	7	d	3	9	7	0.12 ETH

ROOT: Extension Node

prefix	shared nibble(s)	next node
0	a7	●

Branch Node

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	value
	●						●								●	

Leaf Node

prefix	key-end	value
2	1355	45.0ETH

Extension Node

prefix	shared nibble(s)	next node
0	d3	●

Leaf Node

prefix	key-end	value
2	9365	1.1ETH

Branch Node

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	value
			●						●							

Leaf Node

prefix	key-end	value
3□	7	1.00WEI

Leaf Node

prefix	key-end	value
3□	7	0.12ETH

Prefixes

0 - Extension Node, even number of nibbles

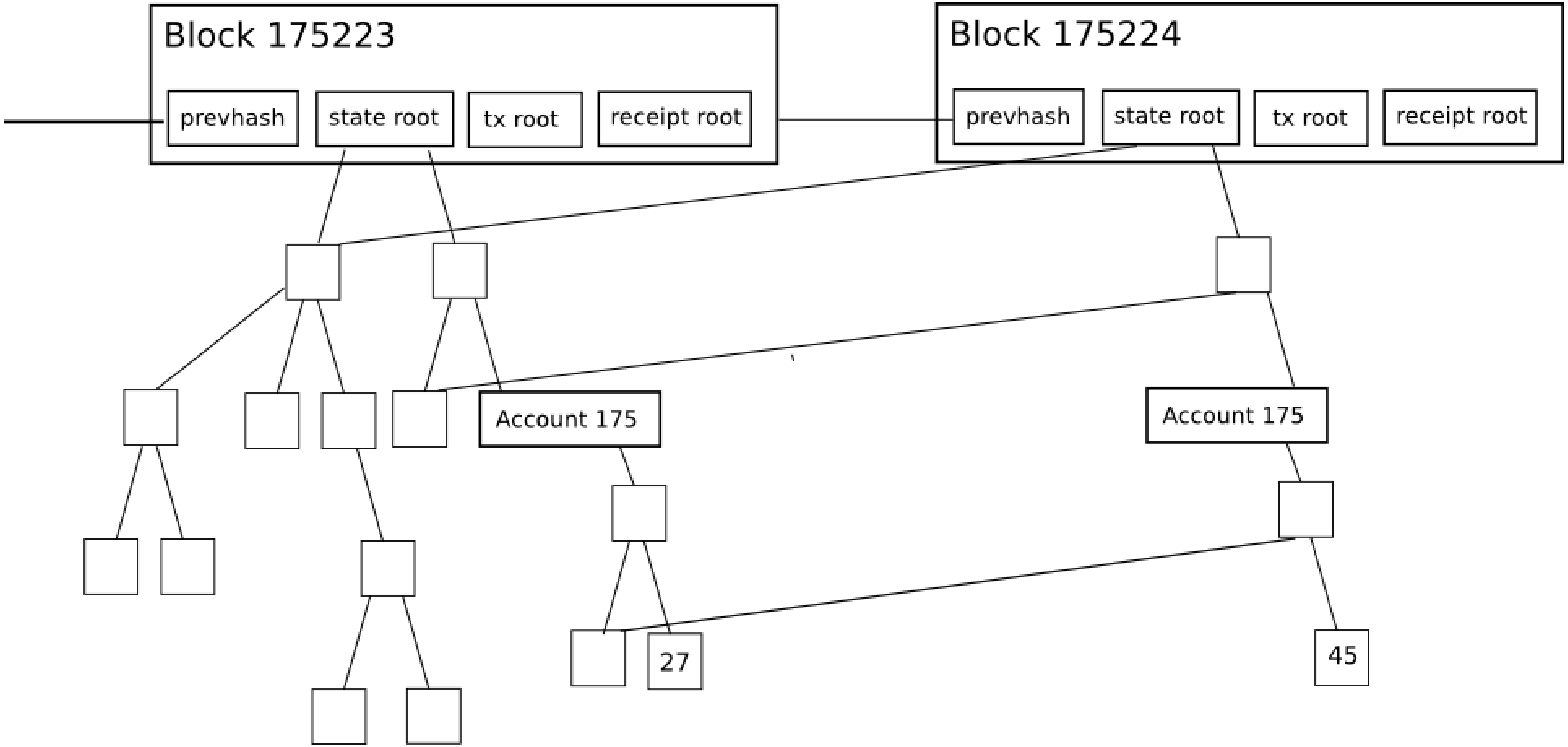
1□ - Extension Node, odd number of nibbles,

2 - Leaf Node, even number of nibbles

3□ - Leaf Node, odd number of nibbles

□ = 1st nibble

1 nibble = 4 bits


Block 175223
prevhash
state root
tx root
receipt root
Block 175224
prevhash
state root
tx root
receipt root
Account 175
Account 175
27
45

State Trie Architecture

The Ethereum State Trie has four types:

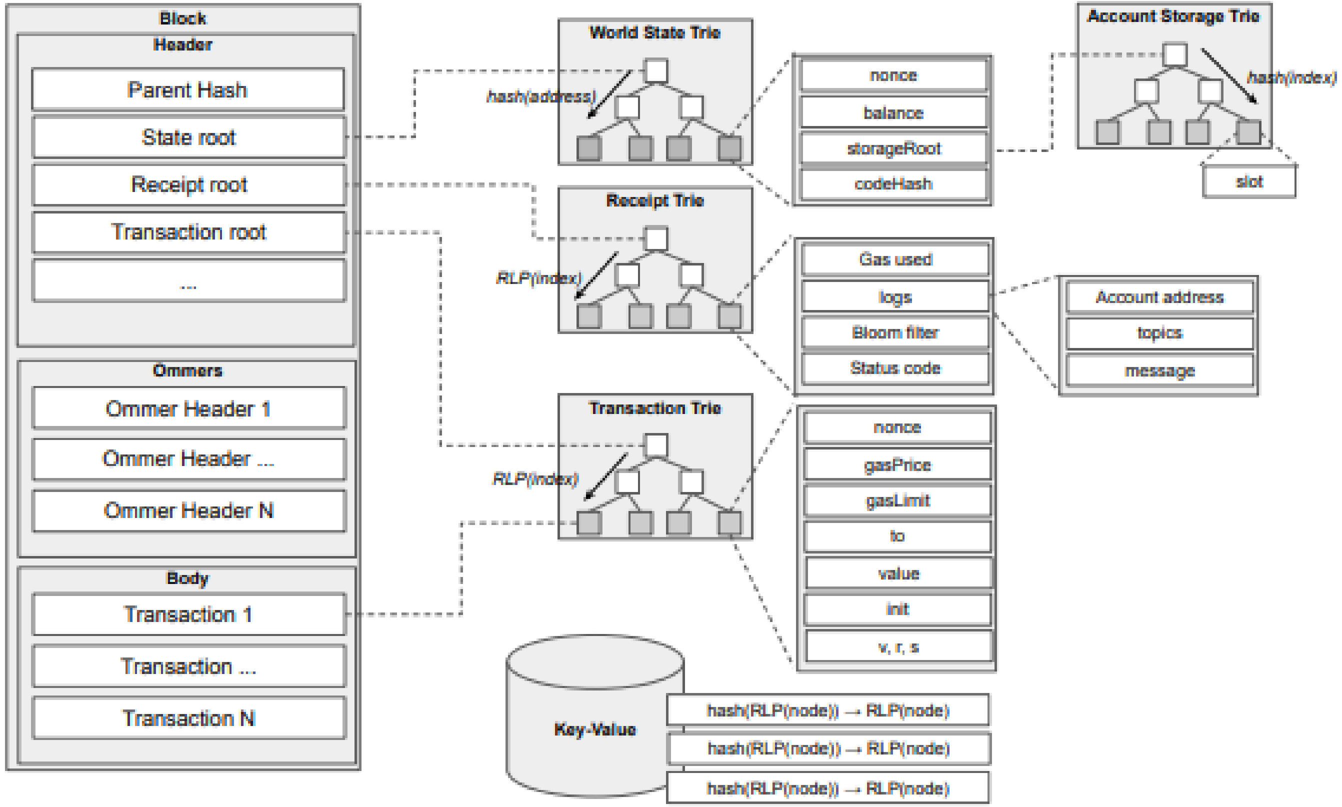
1. world state trie,
2. transaction trie
3. transaction receipt trie
4. and account storage trie.

Each state trie is constructed with Merkle Patricia Trie and only root node (top node of state trie) is stored in block to spare storage.

Blocks in Ethereum

- We can find that a block B contains a header H, transactions T and ommers, sometimes referred to as uncles U :

$$B = (H,T\ ,U\ )$$



Block Header

- **parentHash** The Keccak 256 bit hash of the parent block. This field connects blocks in a chain.
- **ommersHash** The Keccak 256 bit hash of list of ommers.
- **beneficiary** An account address of a user who mined this block and receives reward.
- **stateRoot** The Keccak 256 bit hash of a root of the World State Trie.
- **receiptsRoot** The Keccak 256 bit hash of a root of the Transaction Receipt Trie.
- **transactionsRoot** The Keccak 256 bit hash of a root of the Transaction Trie.
- **logsBloom** A filter relating logs hashes with the log records.
- **difficulty** A scalar value corresponding to an effort that has to be undertaken to mine this block.
- **number** A scalar value equivalent to an ordinal number of this block. Every new block in the chain gets a number increased by one.
- **gasLimit** A scalar value containing an accumulated gas limit required to process all transactions in this block.
- **gasUsed** A scalar value containing an accumulated real consumed gas to process all transactions in this block.
- **extraData** A free form, 32 bytes or less long byte array, containing any additional data.
- **mixHash** The Keccak 256 bit hash confirming that a sufficient computation has been spent on mining this block.
- **nonce** A 64 bit value. This value combined with mixHash also proves that the computation has been spend for mining this block.

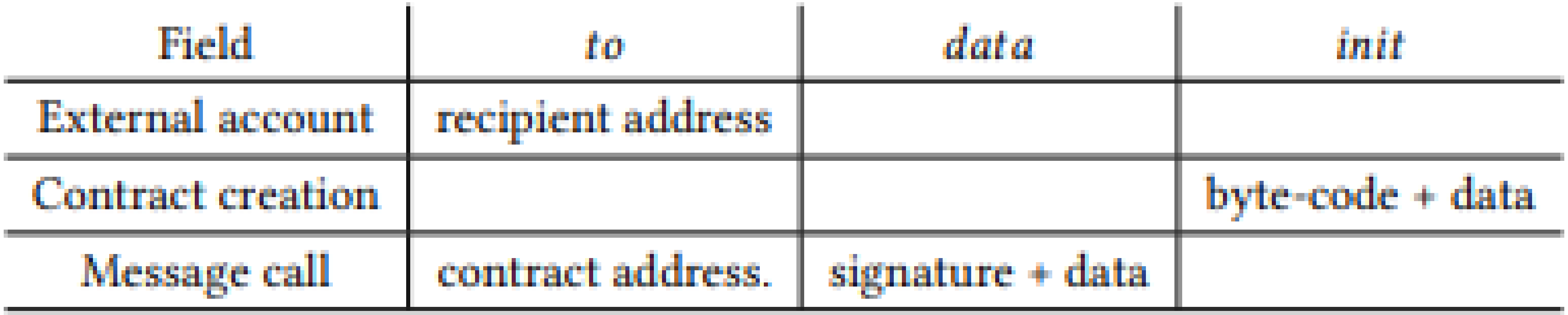
Transactions Trie

A transaction is mapped in the trie so that the key is a transaction $index$ and the value is the transaction T . Both the transaction index and the transaction itself are RLP encoded. It compose a key-value pair, stored in the trie:

$$RLP\ (index) \rightarrow RLP\ (T\ )$$

- **Nonce:** an ordinal number of a transaction. For every new transaction submitted by the same sender, the nonce is increased. Prevents sending the same transaction twice.
- **gasPrice:** A value indicating current price of gas.
- **gasLimit:** A value indicating maximal amount of gas the sender is able to spend on executing this transaction.
- **to:** Address of an account to receive funds, or zero for contract creation.
- **value:** Amount of funds to transfer between external accounts, or initial balance of an account for a new contract.
- **init:** EVM initialization code for new contract creation.
- **data:** Input data for a message call together with the message (i.e. a method) signature.
- **v, r, s:** Values encoding signature of a sender.

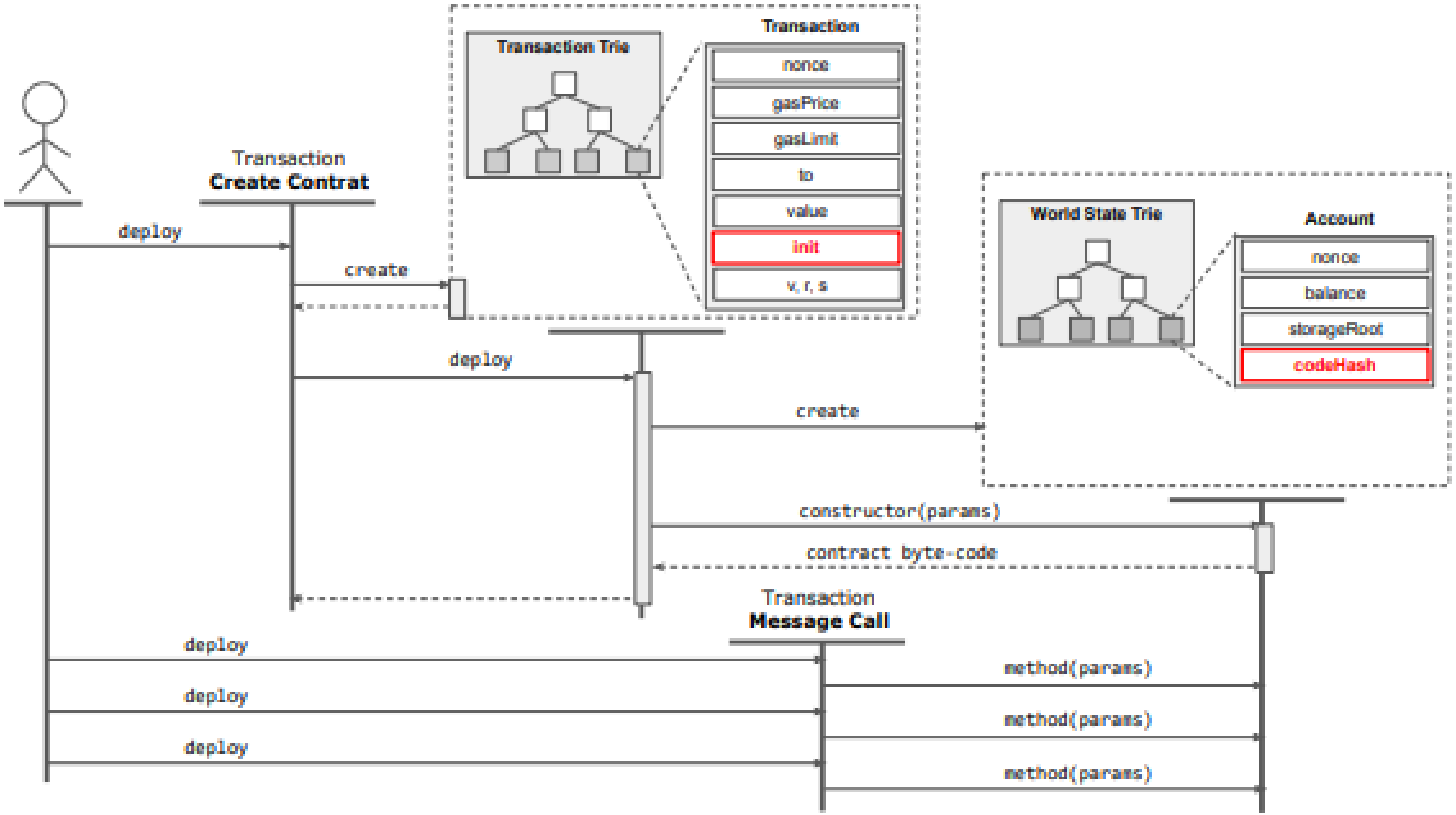
Transactions Trie



Field	*to*	*data*	*init*
External account	recipient address		
Contract creation			byte-code + data
Message call	contract address.	signature + data	

Three types of transactions and the mapping of respective fields

Contract Creation



Message Calls

- Message call execution of a method triggerd by a special transaction which has non-empty field data
- The transaction that calls a contract cobines both the method signature and the input data into the **data field**
- The method signature is stored in hashed form followed by the input data
- The data field is encoded in standardized binary format **Abstract Binary Interface**
- Contracts may call methods of other contracts via internal transactions.
- Internal transactions are **not recorded in the blockchain**
- They do not have the gas limit and consumed gas is billed against the source caller
- This allows creating libraries which are not priced themselves

Transaction Receipt

- When a transaction is submitted to a blockchain, it is not executed immediately.
- First executes the transaction the miner, who potentially put it in the block.
- Then, the transaction becomes part of the blockchain and all participating nodes will sooner or later synchronize, i.e. execute the transaction.
- **Because of this process, a user submitting originally the transaction cannot have its results immediately.**

Transaction Receipt Trie

- As a solution, result of each transaction execution is captured in the **Transaction Receipt Trie.**
- One entry in this trie is created for each transaction
- Optionally, the trie may contain log messages generated by smart contracts.
- By analyzing the Receipts Trie, the user can study log messages to get deeper insight.
- Ethereum nodes provide an API that allows the user to register for events and get notifications when a transaction receipt has arrived.

Transaction Receipt Trie

- The Receipt Trie has following structure. The **key is a transaction $\boldsymbol{index}$** and the value is a **receipt $\boldsymbol{R}$**. They compose a key-value pair:

$$RLP\ (index) \rightarrow RLP\ (R)$$

The receipt R has following fields:

- Cumulative gas used in a respective block after execution of current transaction.
- Set of logs printed by this transaction
- A Bloom Filter containing hashes of log entries
- A status code of the transaction result, expressed as a number. It semantics is application dependent.

Blooms Filter

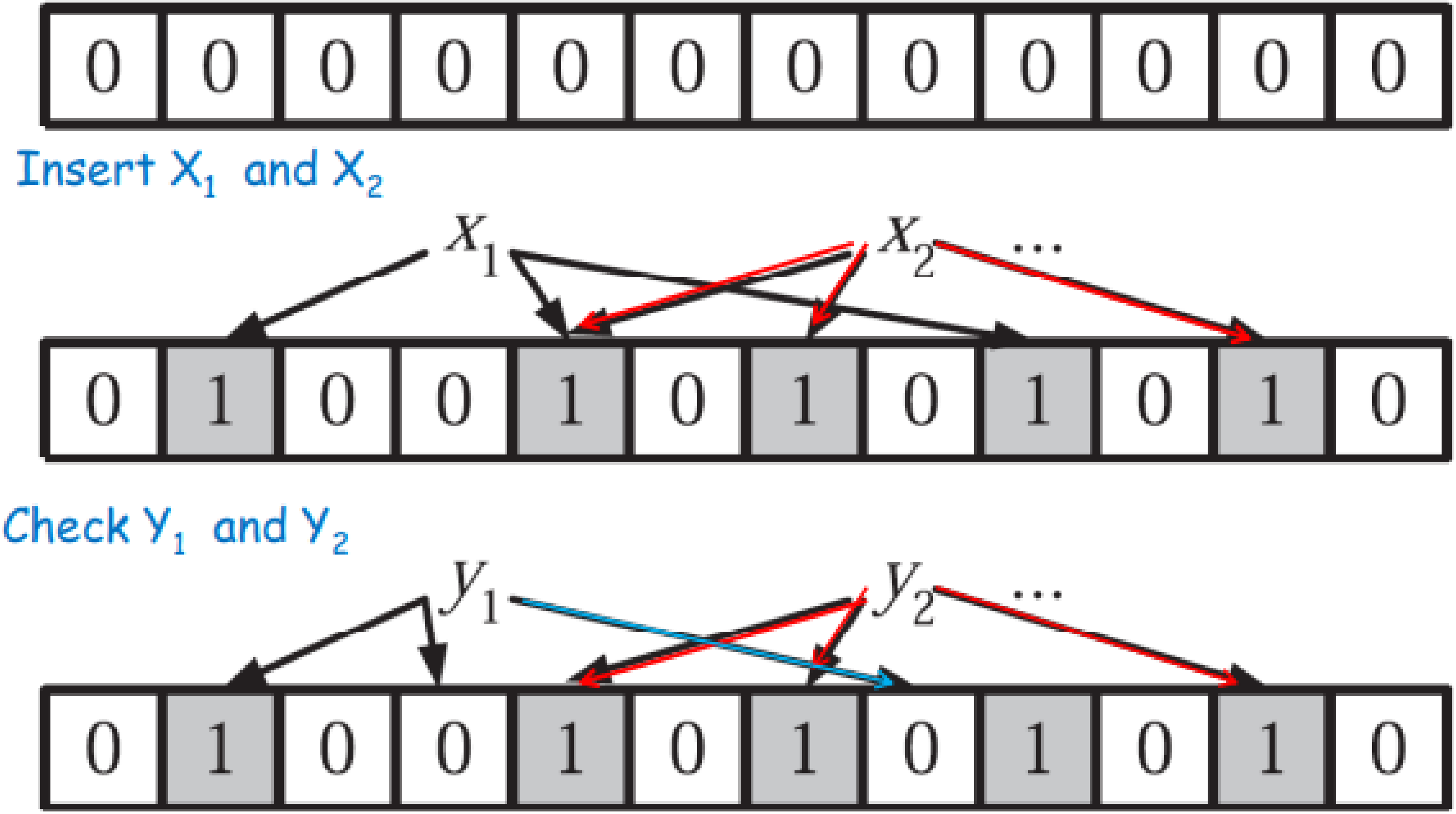
- Randomized data structure introduced by Burton Bloom [CACM 1970]
  - It represents a set for membership queries, with false positives
  - Probability of false positive can be controlled by design parameters
  - When space efficiency is important, a Bloom filter may be used if the effect of false positives can be mitigated.

- First applications in dictionaries and databases

Standard Bloom Filter

- A Bloom filter is an *array of m bits* representing a set $S = \{ x_1, x_2, \ldots , x_n\}$ of *n* elements
  - Array set to 0 initially
- *k* independent hash functions $h_1, \ldots , h_k$ with range $\{1, 2, \ldots, m\}$
  - Assume that each hash function maps each item in the universe to a random number *uniformly* over the range $\{1, 2, \ldots, m\}$
- For each element x in S, the bit $h_i(x)$ in the array is set to 1, for $1 \leq i \leq k$,
  - A bit in the array may be set to 1 multiple times for different elements

Bloom filter example



0	0	0	0	0	0	0	0	0	0	0	0

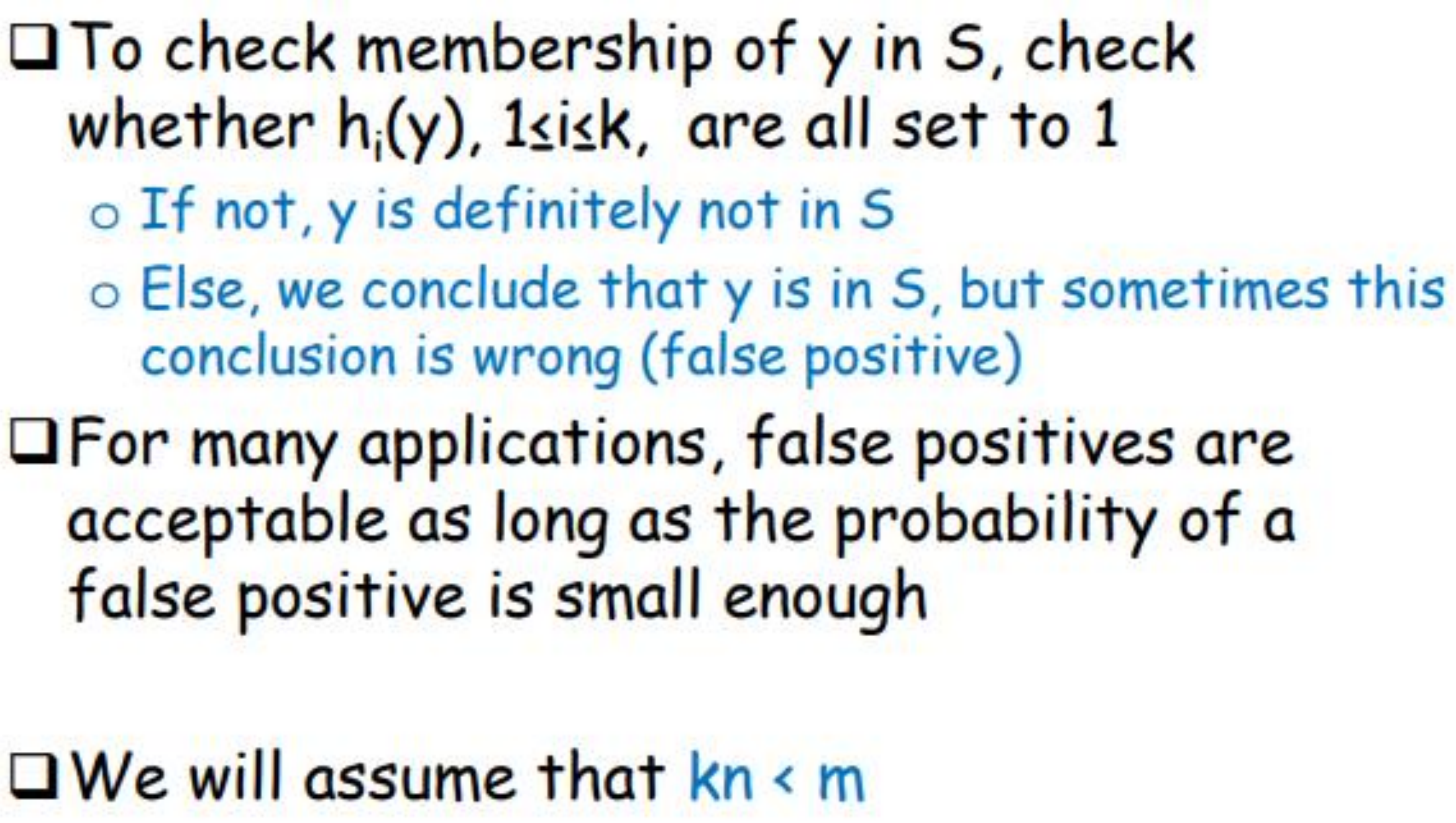
Insert X_1 and X_2

0	1	0	0	1	0	1	0	1	0	1	0

Check Y_1 and Y_2

0	1	0	0	1	0	1	0	1	0	1	0

Standard Bloom Filter (cont.)



- To check membership of y in S, check whether $h_i(y)$, $1 \le i \le k$, are all set to 1
  - If not, y is definitely not in S
  - Else, we conclude that y is in S, but sometimes this conclusion is wrong (false positive)
- For many applications, false positives are acceptable as long as the probability of a false positive is small enough
- We will assume that $kn < m$

False positive probability

- ❑ After all members of S have been hashed to a Bloom filter, the probability that a specific bit is still 0 is

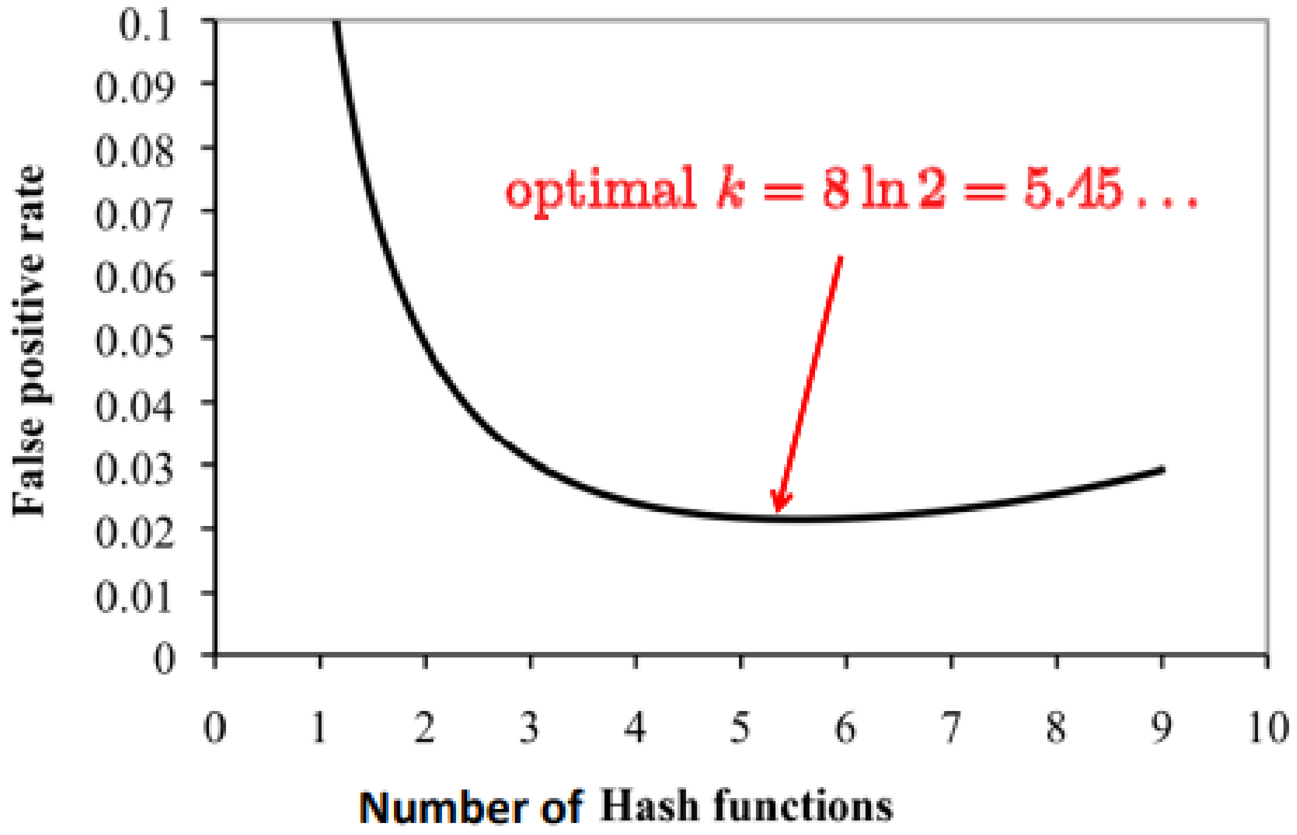
$$p' = (1 - \frac{1}{m})^{kn} \simeq e^{-kn/m} = p$$

- ❑ For a non member, it may be found to be a member of S (all of its k bits are nonzero) with *false positive probability*

$$(1 - p')^{k} \simeq (1 - p)^{k}$$

False positive rate vs. k

Number of bits per member $\frac{m}{n} = 8$

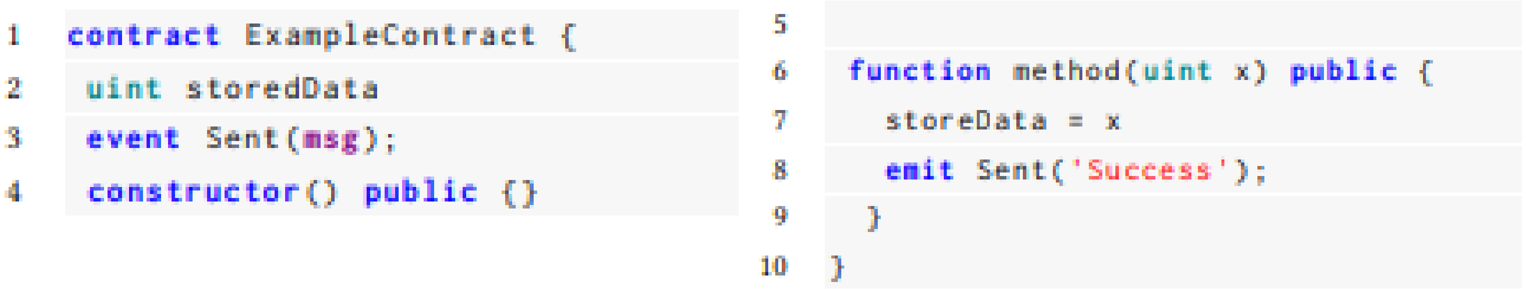


optimal $k = 8 \ln 2 = 5.45\ldots$

Message Logs

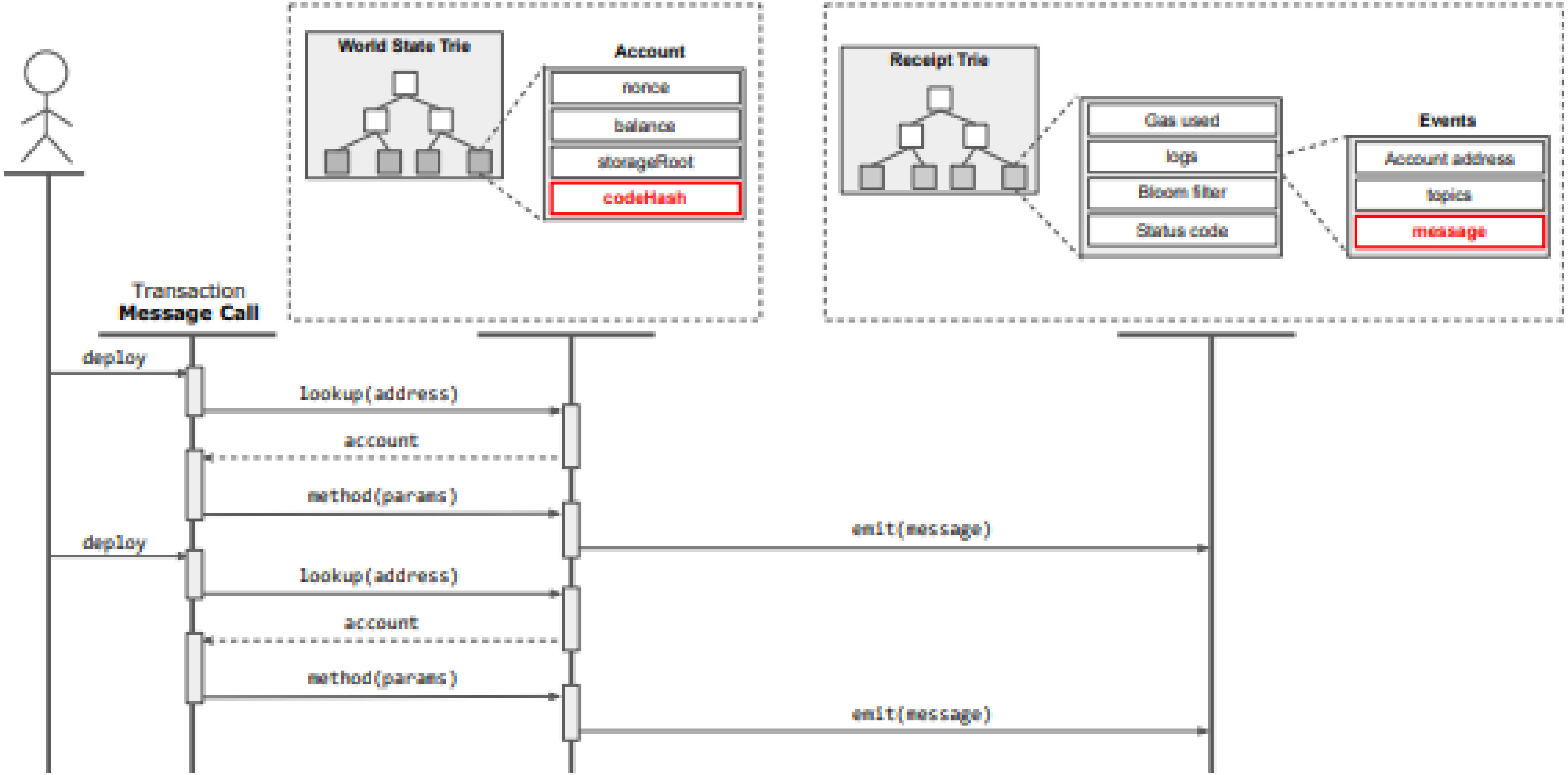
- The Receipt Trie can contain a set of log messages, produced by smart contract.
- In the case of Solidity, the log message is emitted via a keyword **emit**.
- The set of logs contains tuples with following items:
    - A contract account address, who triggered the log, i.e. executed this Smart Contract.
    - A sequence of 32 bytes long containing event topics.
    - A sequence of bytes containing the log message itself
- The log messages may contain any arbitrary data and every message may be assigned to a topic. This way clients may listen only to messages from certain topics if they are not interested in all log messages.

Log Event Updates Transaction Receipt



```
contract ExampleContract {
 uint storedData
 event Sent(msg);
 constructor() public {}

 function method(uint x) public {
   storeData = x
   emit Sent('Success');
  }
}
```

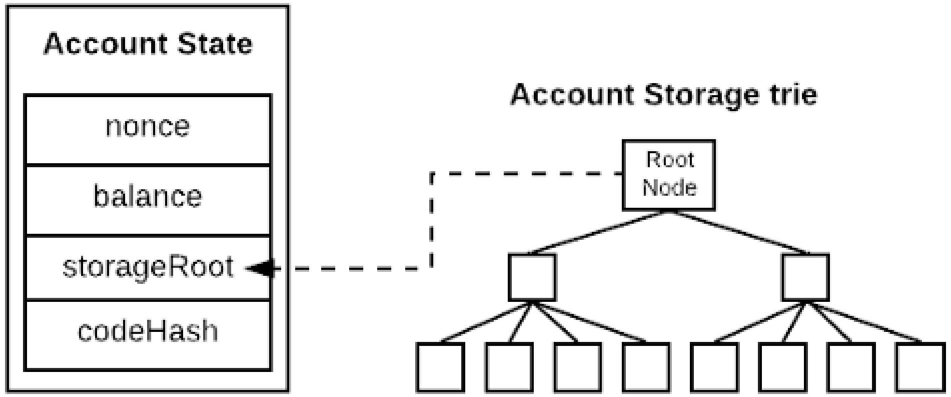


World State Trie

- Also known as **State Trie or Global State Trie**
- In contract to Trasaction Trie it is mutable data structure
- World state trie is a mapping between addresses and account states.
  - Keccak(address) -> RLP(A)
- It can be seen as a global state that is constantly updated by transaction executions.
- The Ethereum network is a decentralized computer and state trie is considered as hard drive.
- All the information about accounts are stored in world state trie and you can retrieve information by querying it.
- World state trie is closely related to account storage trie because it has “storageRoot” field that points the root node in account storage trie.

World state trie node fields

- **nonce**
  - Number of transactions sent from this address (if this is an External Owned Account - EOA) or the number of contract-creations made by this account
- **balance**
  - Total Ether (in Wei) owned by this account.
- **storageRoot**
  - A 256-bit hash root of the account storage trie. Empty for EOAs
- **codeHash**
  - Hash of EVM code. The bytecode is stored in undelying database under this hash key. Empty for EOA

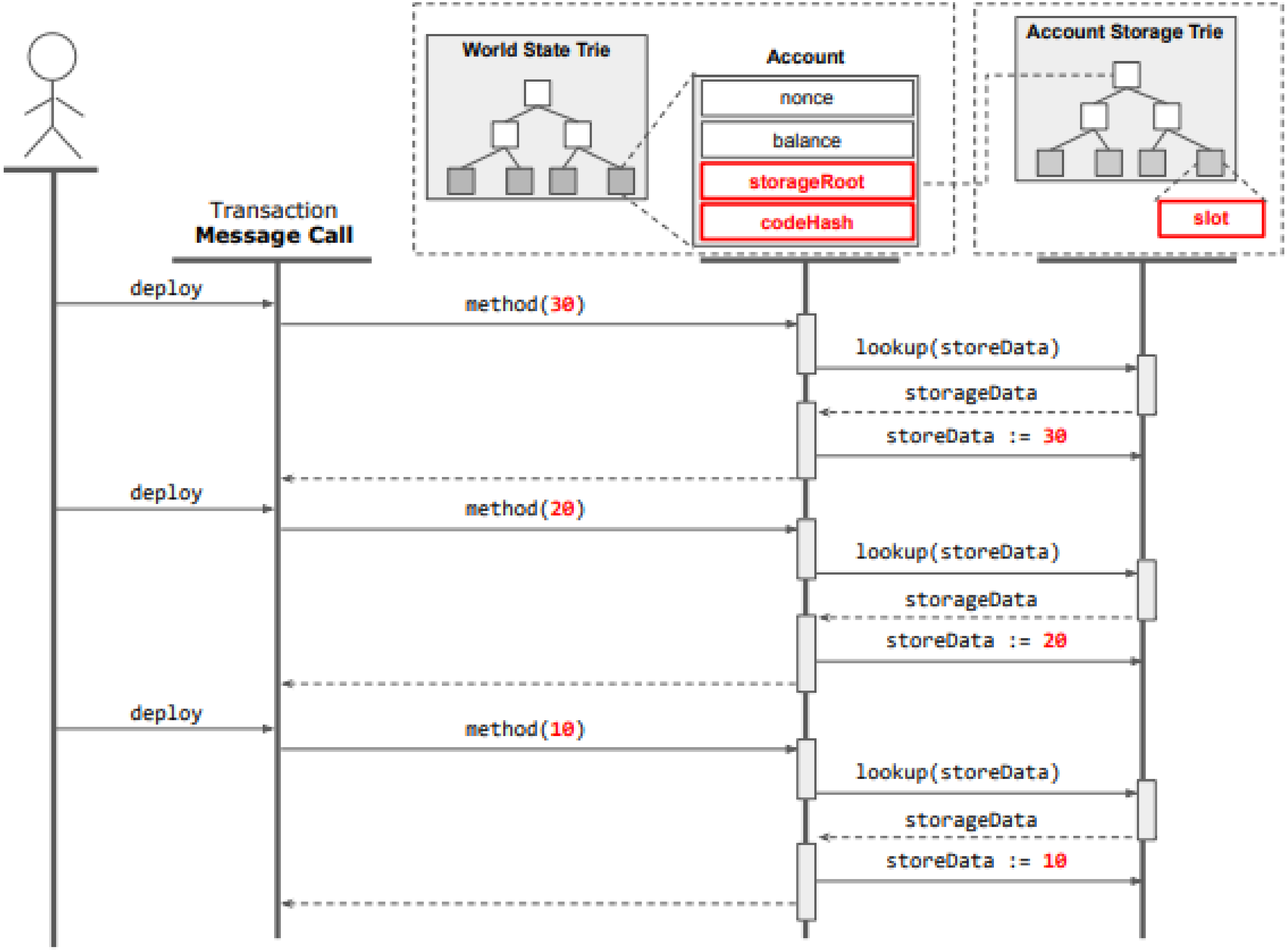


One important details about the account state is that **all fields (except the codeHash) are mutable**.

Account Storage Trie

- Account Storage Trie is where data associated with an account is stored.
- Accessed via storageRroot field in world state trie account
- This try is directly modified using smart contract byte code SLOAD and SSTORE
- Every key in this trie is an index of a slot stored in the leaf node
- Index represent one of more global variable determined at the compile time
  - Keccak(index) -> RLP(slot)
- Smart contract data is persisted in the account storage trie as a mapping between 32-bytes integers.

Storage Trie Updates


World State Trie
Account
nonce
balance
storageRoot
codeHash
Account Storage Trie
slot
Transaction
Message Call
deploy
method(30)
lookup(storeData)
storageData
storeData := 30
deploy
method(20)
lookup(storeData)
storageData
storeData := 20
deploy
method(10)
lookup(storeData)
storageData
storeData := 10