

CSE446: Blockchain & Cryptocurrencies

Lecture - 14: Ethereum - 3

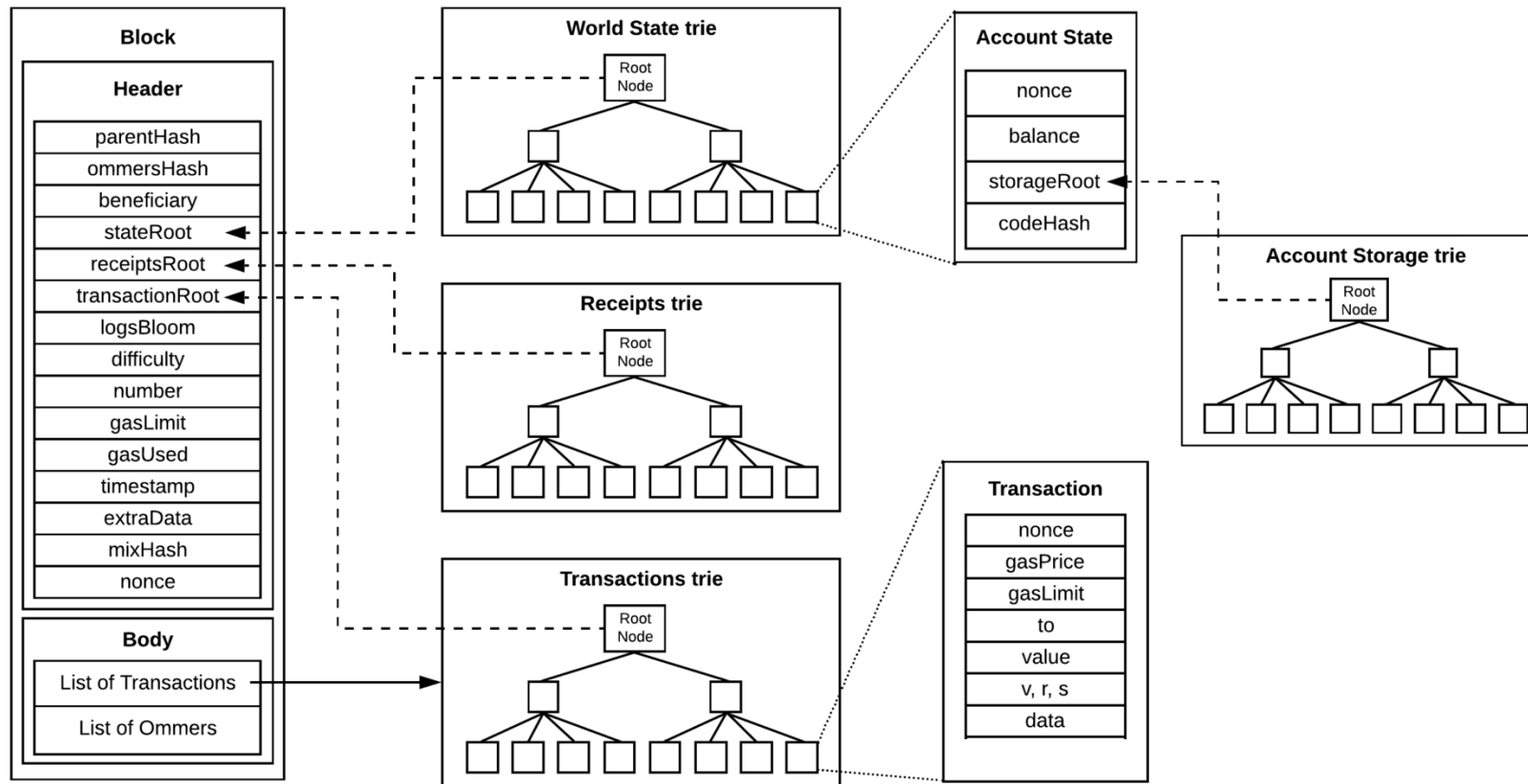


Inspiring Excellence

Agenda

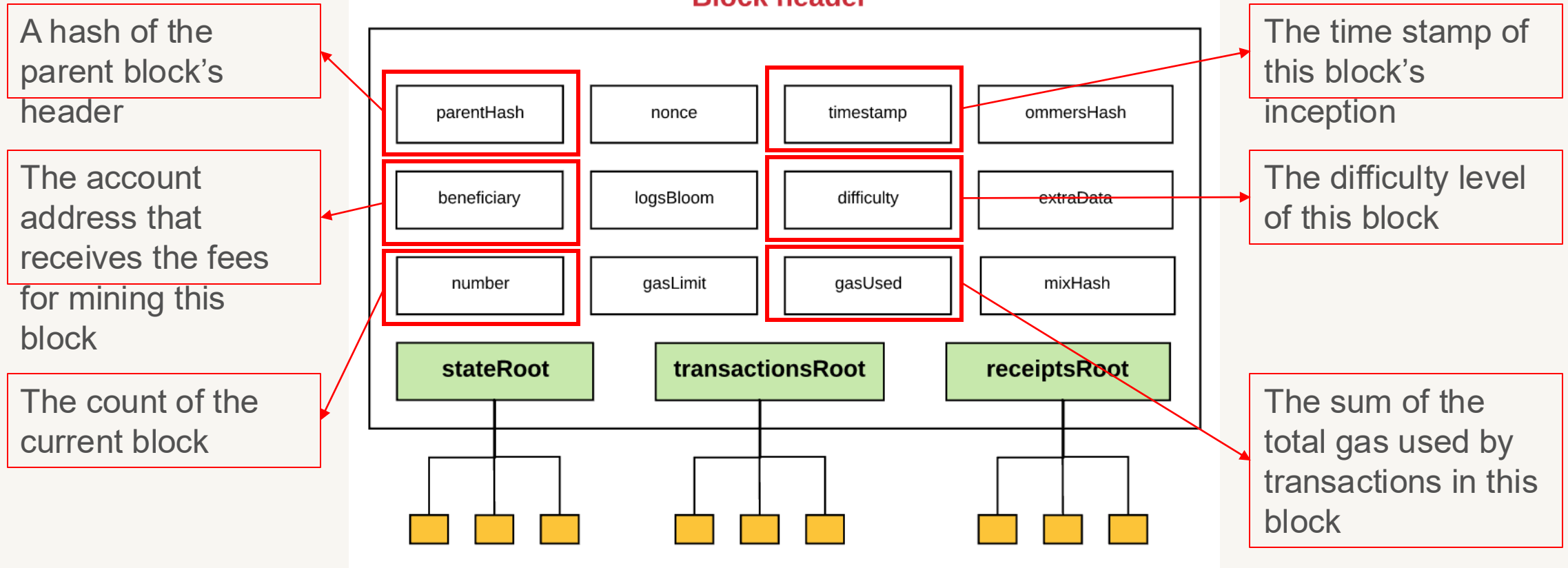
- Ethereum Block
- Ethereum Blockchain
- Ethereum Consensus

Ethereum block



Block, transaction, account state objects and Ethereum tries

Ethereum block header

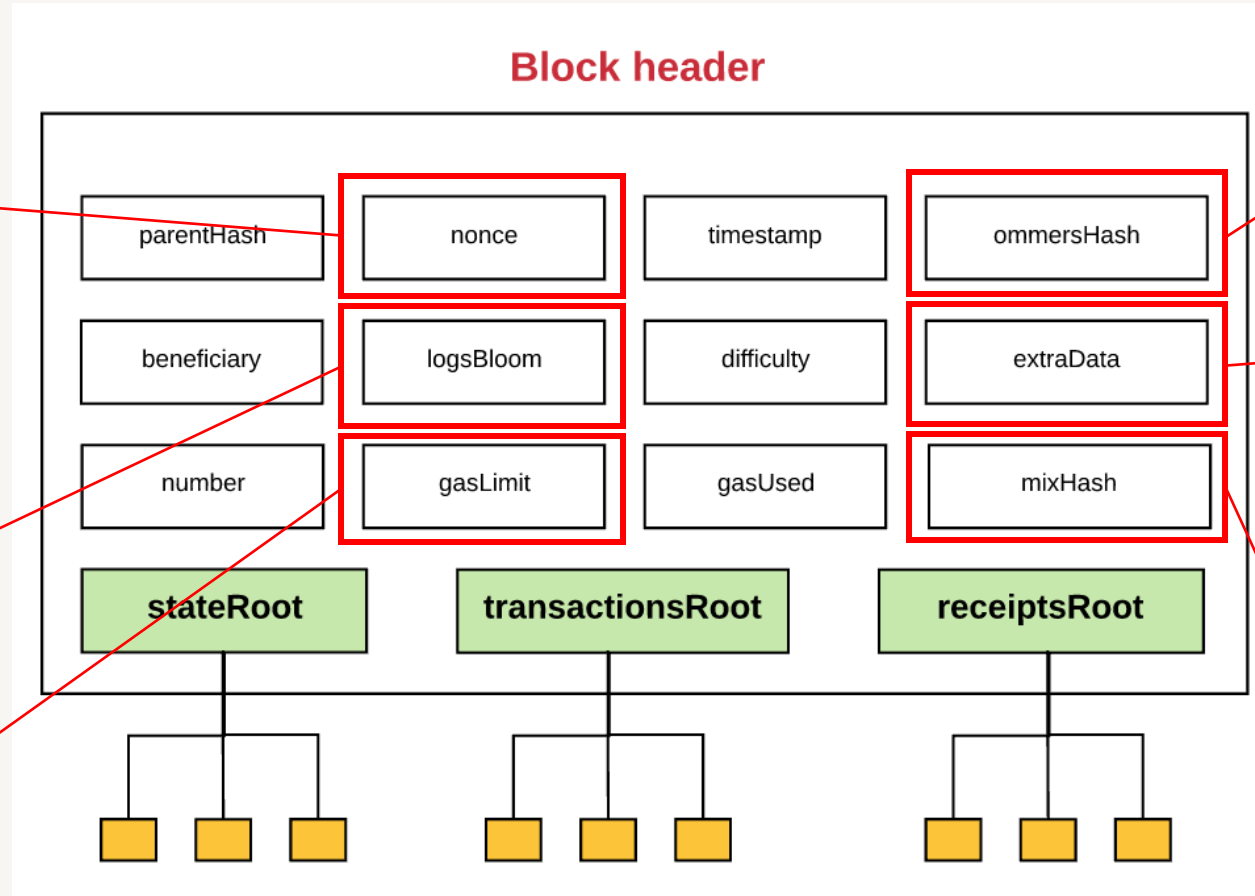


Ethereum block header

A value that, when combined with the mixHash, proves that this block has carried out enough computation

A Bloom Filter (data structure) that consists of log information

The current gas limit per block



A hash of the current block's list of ommers

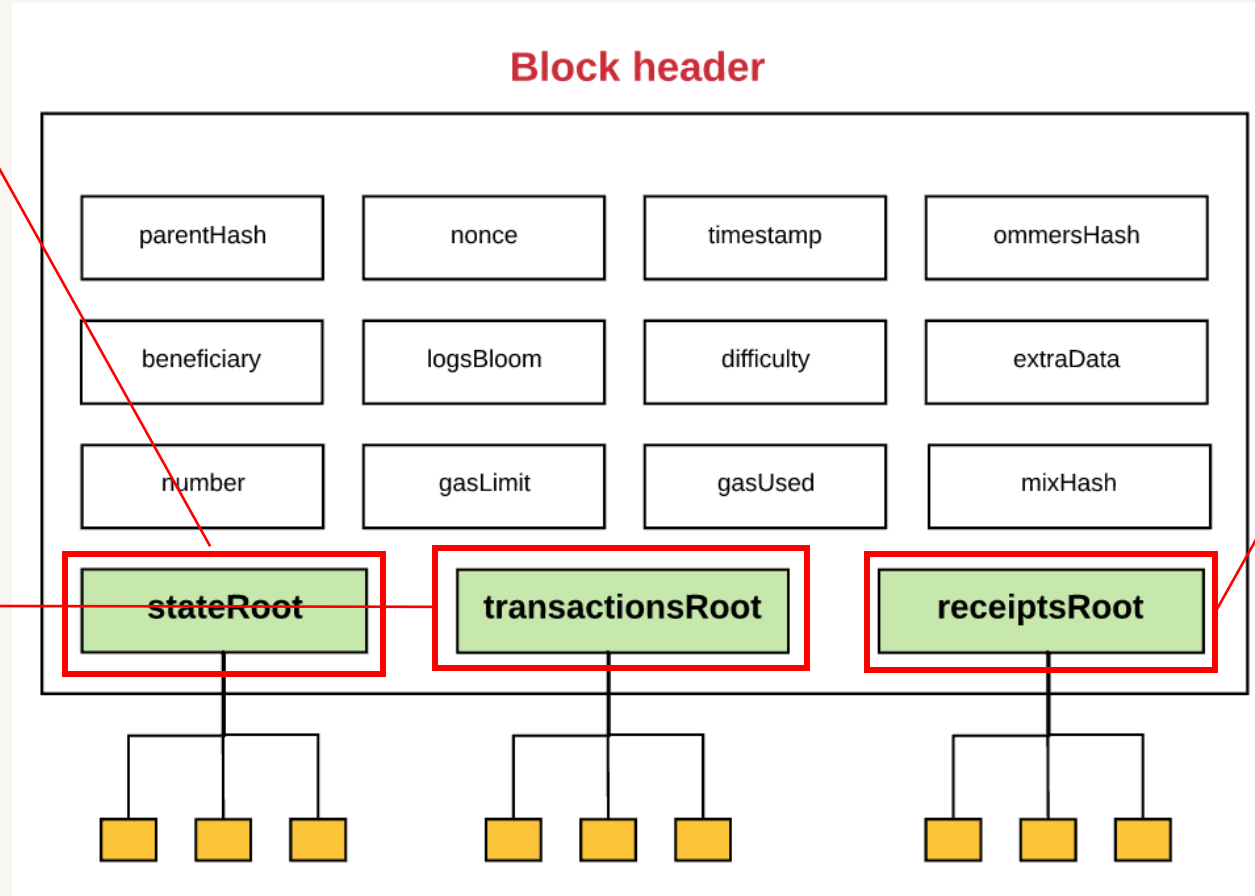
Extra data related to this block

A hash that, when combined with the nonce, proves that this block has carried out enough computation

Ethereum block header

The hash of the root node of the state tree

The hash of the root node of the tree that contains all transactions listed in this block



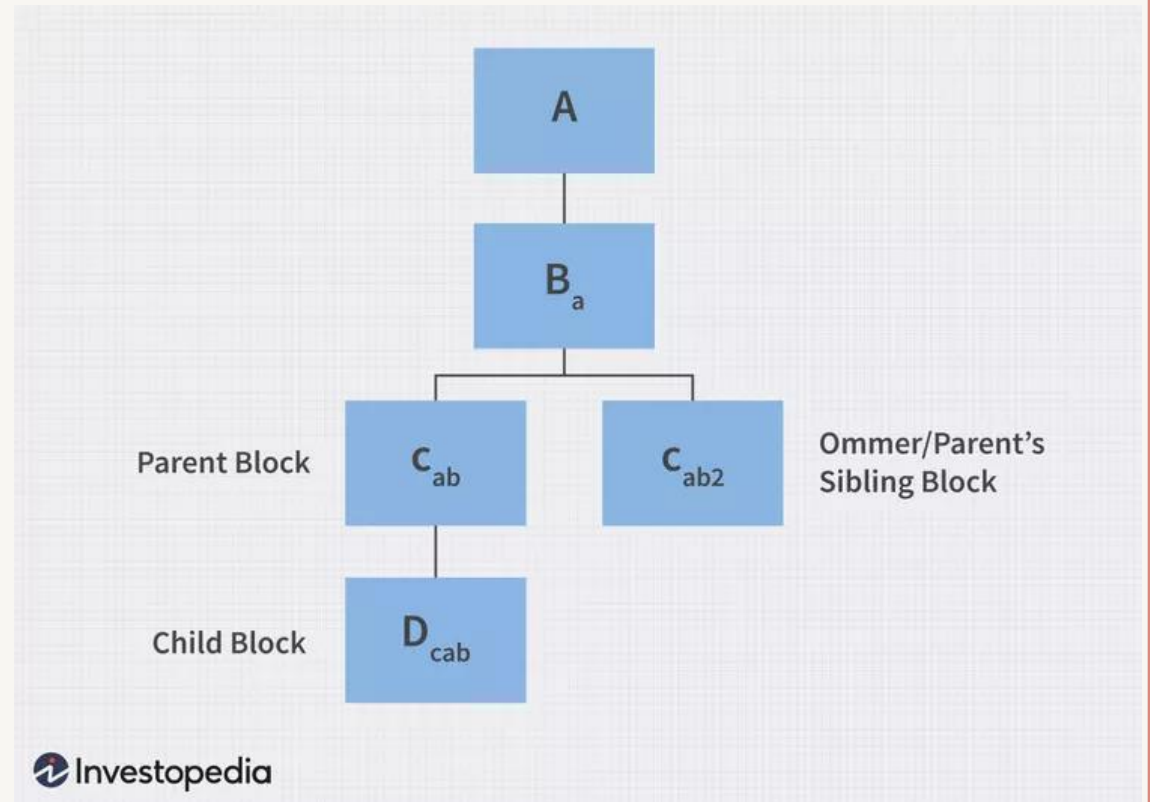
The hash of the root node of the tree that contains the receipts of all transactions listed in this block

Ethereum ommers

- It is possible for two blocks to be created simultaneously by a network
- When this happens, a fork happens and eventually one block is left out
- This leftover block is called an ommer block
- In the past, they were called uncle blocks
 - referring to the familial relationships used to describe block positions within a blockchain
- In Bitcoin, there is no reward for this omner block
 - Ethereum provides a minimum amount of reward to the omner miner

Ethereum ommers

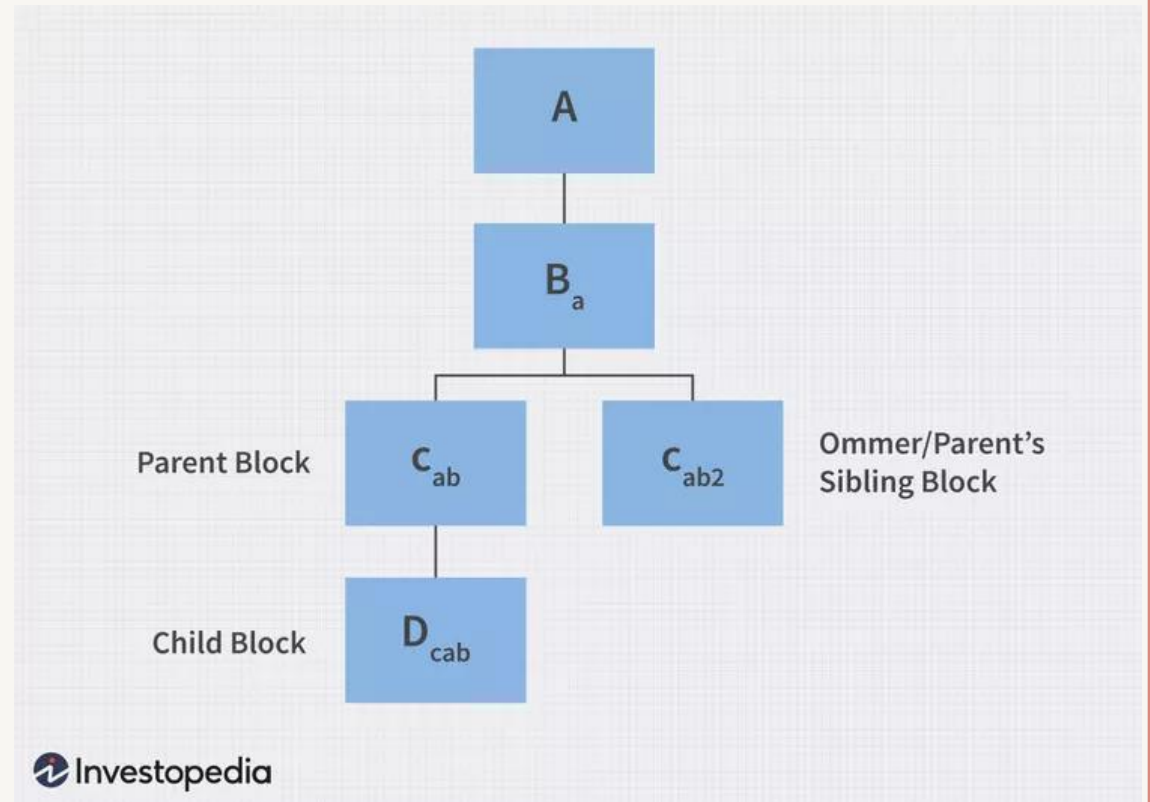
- An ommer is a block whose parent is equal to the current block's parent's parent
- Block times in Ethereum are around 15 sec
 - This is much lower than that in Bitcoin (10 min)
- This enables faster transaction



[https://www.investopedia.com/thmb/Bf315VzUb4nwViDSK36Piv55D1Q=/750x0/filters:no_upscale\(\):max_bytes\(150000\):strip_icc\(\):format\(webp\)/INV-uncle-block-cryptocurrency-fdc913eee3bb40aebba7e499bfceada.jpg](https://www.investopedia.com/thmb/Bf315VzUb4nwViDSK36Piv55D1Q=/750x0/filters:no_upscale():max_bytes(150000):strip_icc():format(webp)/INV-uncle-block-cryptocurrency-fdc913eee3bb40aebba7e499bfceada.jpg)

Ethereum ommers

- But there are more competing blocks, hence a higher number of orphaned blocks
- The purpose of ommers is to help reward miners for including these orphaned blocks
 - Compensating the miners for their computation



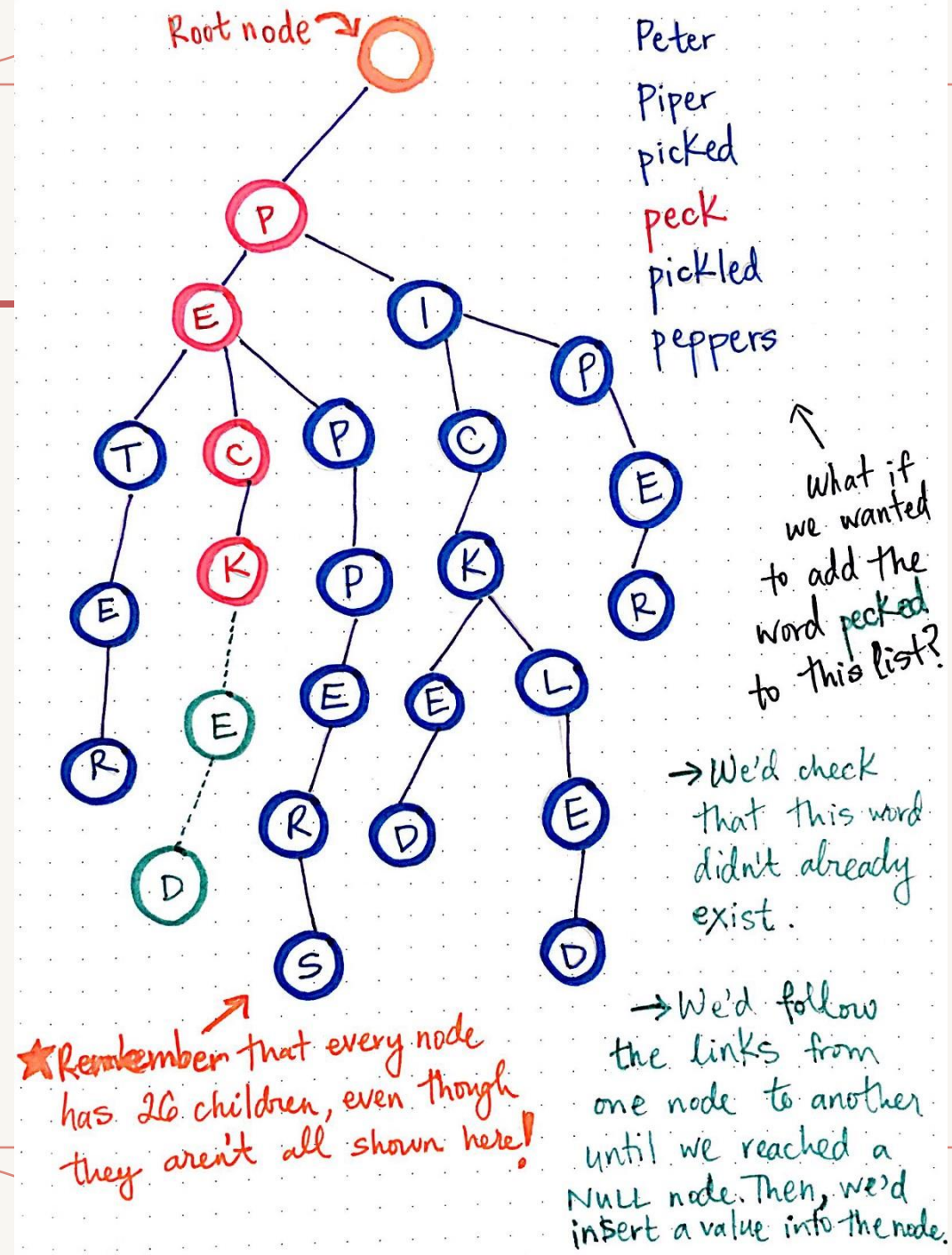
[https://www.investopedia.com/thmb/Bf315VzUb4nwViDSK36Piv55D1Q=/750x0/filters:no_upscale\(\):max_bytes\(150000\):strip_icc\(\):format\(webp\)/INV-uncle-block-cryptocurrency-fdc913eee3bb40aebba7e499bfceada.jpg](https://www.investopedia.com/thmb/Bf315VzUb4nwViDSK36Piv55D1Q=/750x0/filters:no_upscale():max_bytes(150000):strip_icc():format(webp)/INV-uncle-block-cryptocurrency-fdc913eee3bb40aebba7e499bfceada.jpg)

Trie

A **trie** is a tree-like data structure wherein the nodes of the tree store the entire alphabet, and strings/words can be **retrieved** by traversing down a branch path of the tree.

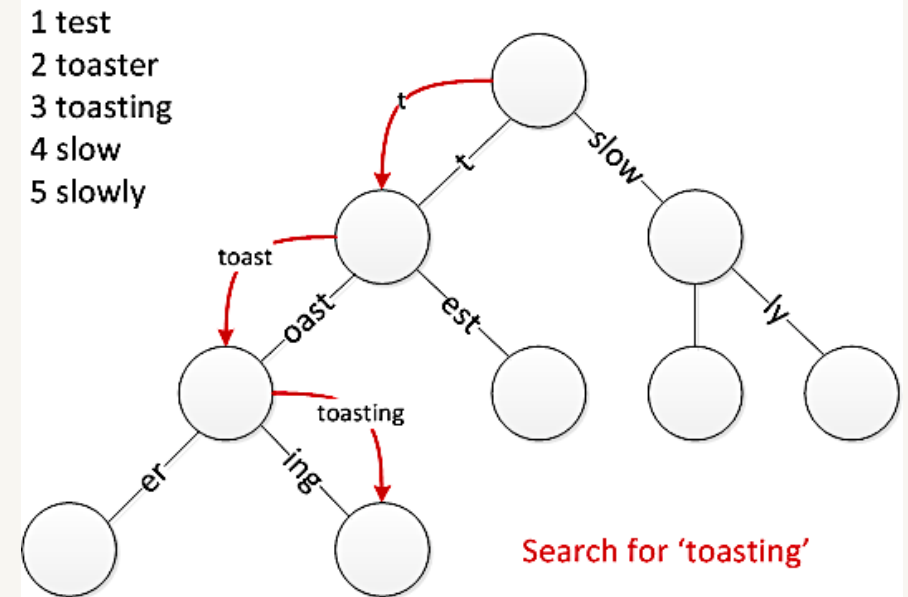
https://cdn-images-1.medium.com/max/1600/1*rkanFIU4G_tmuC939_txhA.jpeg

https://cdn-images-1.medium.com/max/1200/1*sZOrNXzQICVv5ePpav1-g.jpeg



Patricia trie

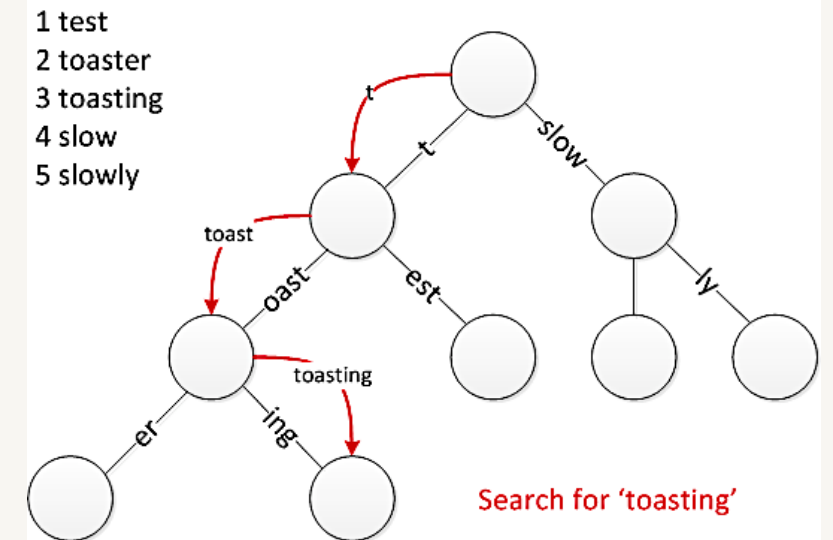
- A Patricia (Practical Algorithm To Retrieve Information Coded In Alphanumeric) trie is a binary radix trie
 - binary choice at each node when traversing the trie
- It is a data structure which uses a key as a path so the nodes that share the same prefix can also share the same path



<https://i.stack.imgur.com/d2w07.png>

PT

- This structure is fastest at finding common prefixes, simple to implement, and requires small memory
- It is commonly used for implementing routing tables, systems that are used in low specification machines like the router



Merkle Patricia Trie (MPT)

- In Ethereum the concept of PT is modified to Merkle Patricia trie
 - the root node becomes a cryptographic fingerprint of the entire data structure, just like a Merkle tree
- An MPT is a data structure for storing key value pairs in a cryptographically authenticated manner
- Three different node types: extension, branch and leaf
- A node that does not have a child node is called a leaf node

MPT

- Nodes in MPT can have 16 child nodes
 - Plus it has its value, totalling 17 fields
- In Ethereum, hexadecimal is used - a 16 characters "alphabet"
- Note a hex character is referred to as a "nibble"

MPT

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 ()

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

World State Trie

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

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

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

MPT

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

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

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

- This represents a simplified world state of accounts
- Instead of storing each account in the blockchain, MPT is used

MPT

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 ()

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

World State Trie

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

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

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

- Prefix determines the type of node
- 0/1 indicates an extension node
- If there are even number of nibbles then 0, otherwise 1

MPT

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 ()

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

World State Trie

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

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

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

- 2/3 indicates a leaf node
- If there are even number of nibbles then 2, otherwise 3

MPT

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 ()

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

World State Trie

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

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

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

- All these accounts share a common prefix: a7
- That is why a7 remains in the root

MPT

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()

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

World State Trie

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

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

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

- Next node field in the extension node points to a branch node
- A branch node has 16 hexadecimal characters and a value field

MPT

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 ()

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

World State Trie

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

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

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

- Each field except the value field represents a key character

MPT

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()

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

World State Trie

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

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

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

- Each leaf node has a prefix indicating its even or odd number of nibbles
- A key-end to store the last values of the key
- Finally the corresponding balance for the account

MPT

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()

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

World State Trie

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

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

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

MPT

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()

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

World State Trie

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

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

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

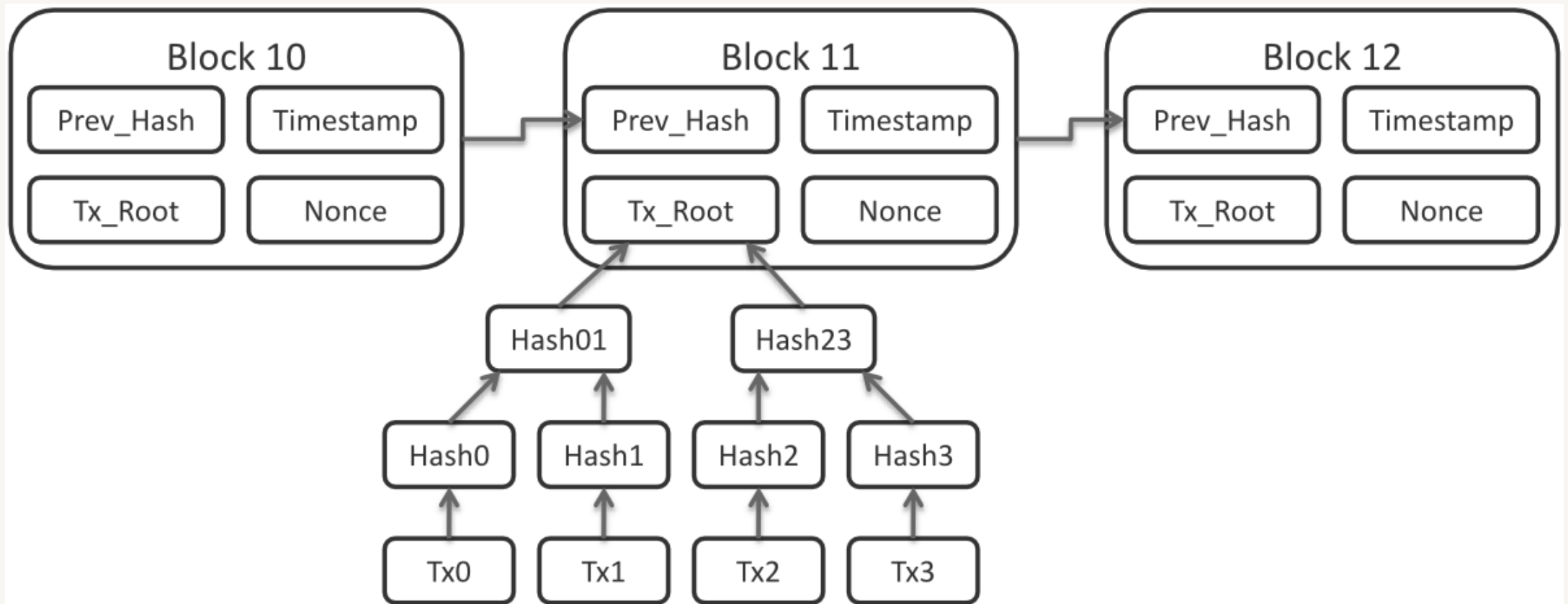
Leaf Node

prefix	key-end	value
3□	7	0.12ETH

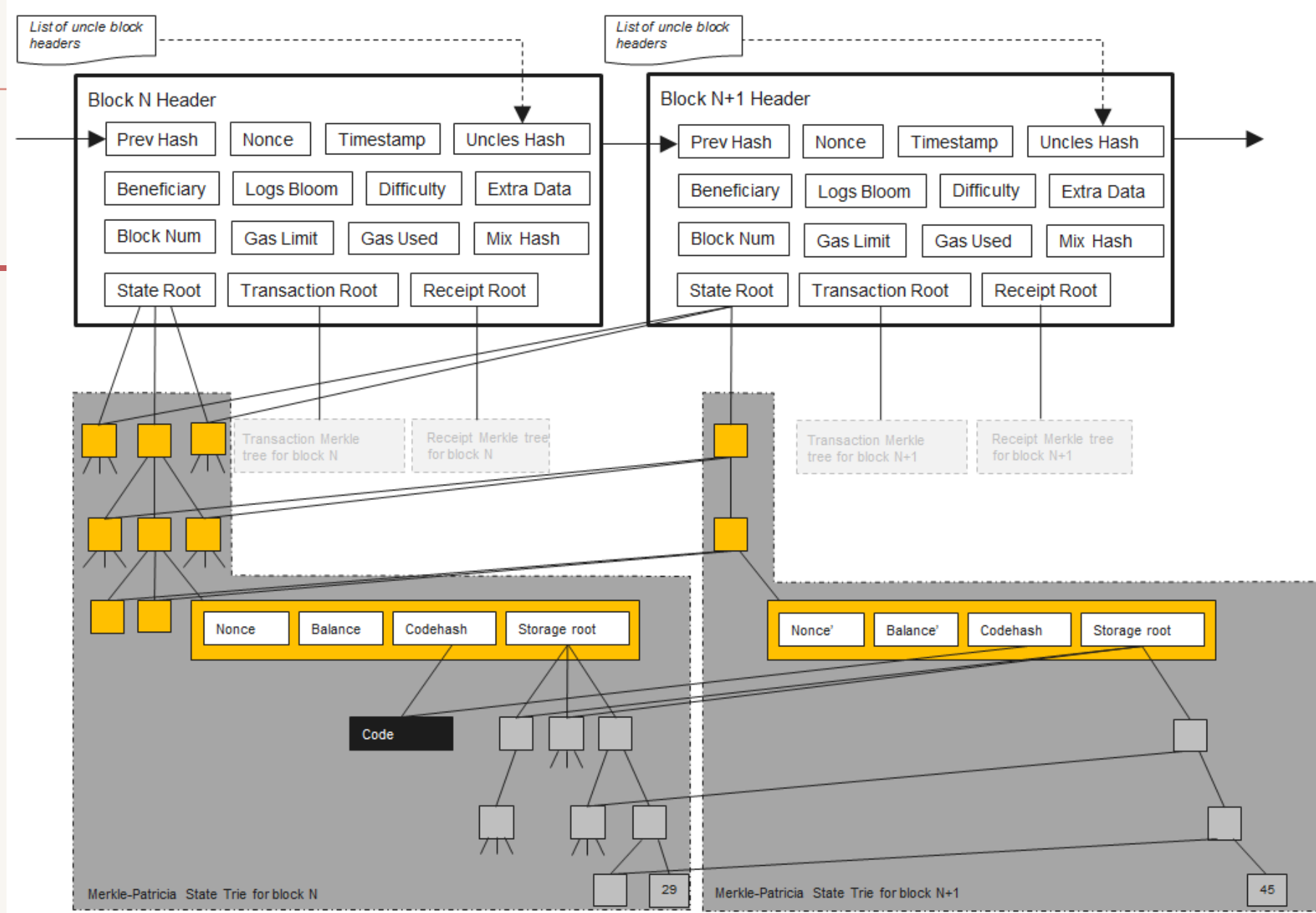
Ethereum tries

- There are four different tries used in Ethereum
 - State Trie
 - Contains an account information with respect to their address
 - Transaction Trie
 - Contains transaction information
 - Transaction Receipt Trie
 - Contains information regarding transaction receipt
 - Account storage Trie
 - Contains storage information with respect a smart contract

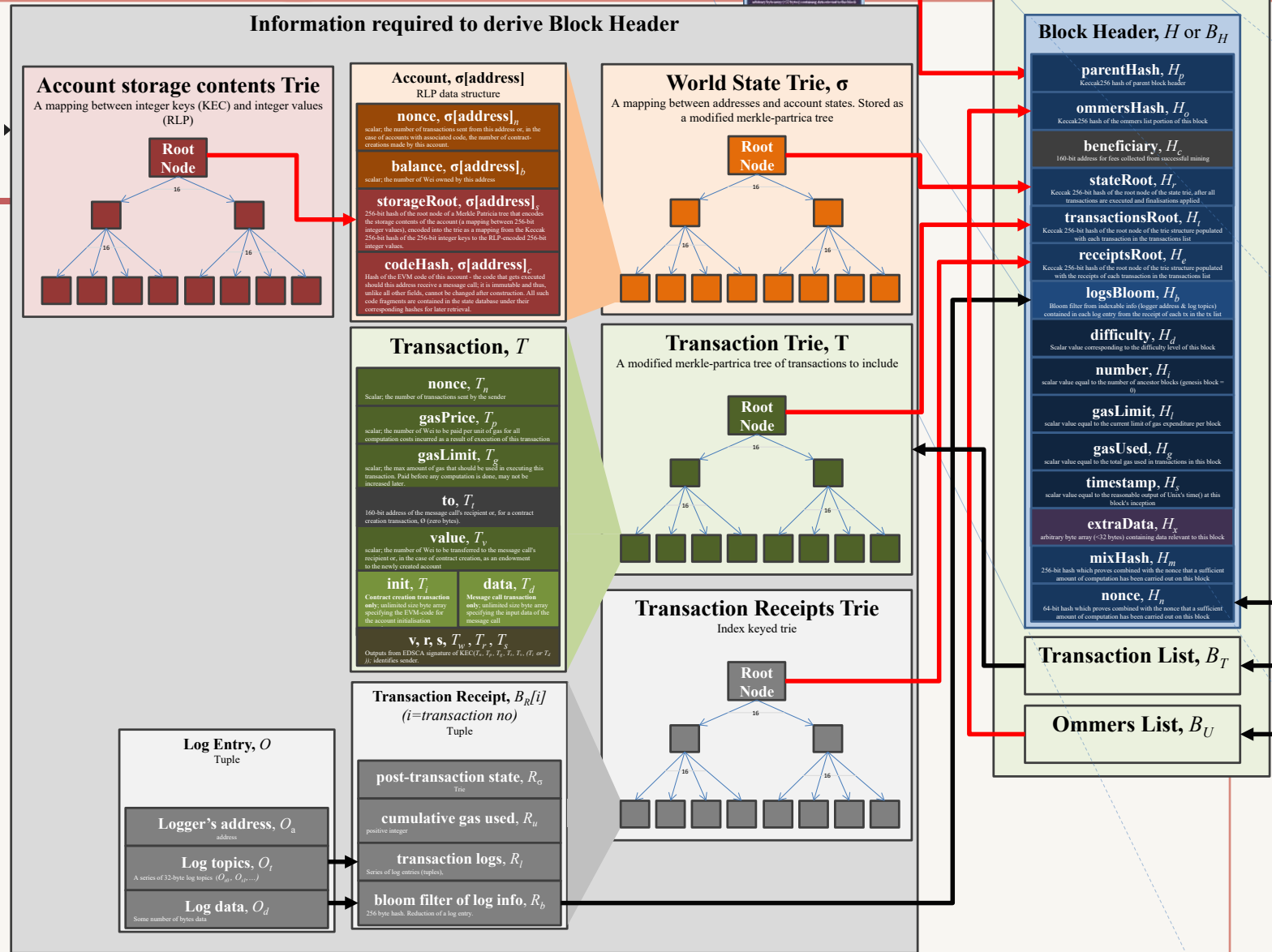
Bitcoin blockchain



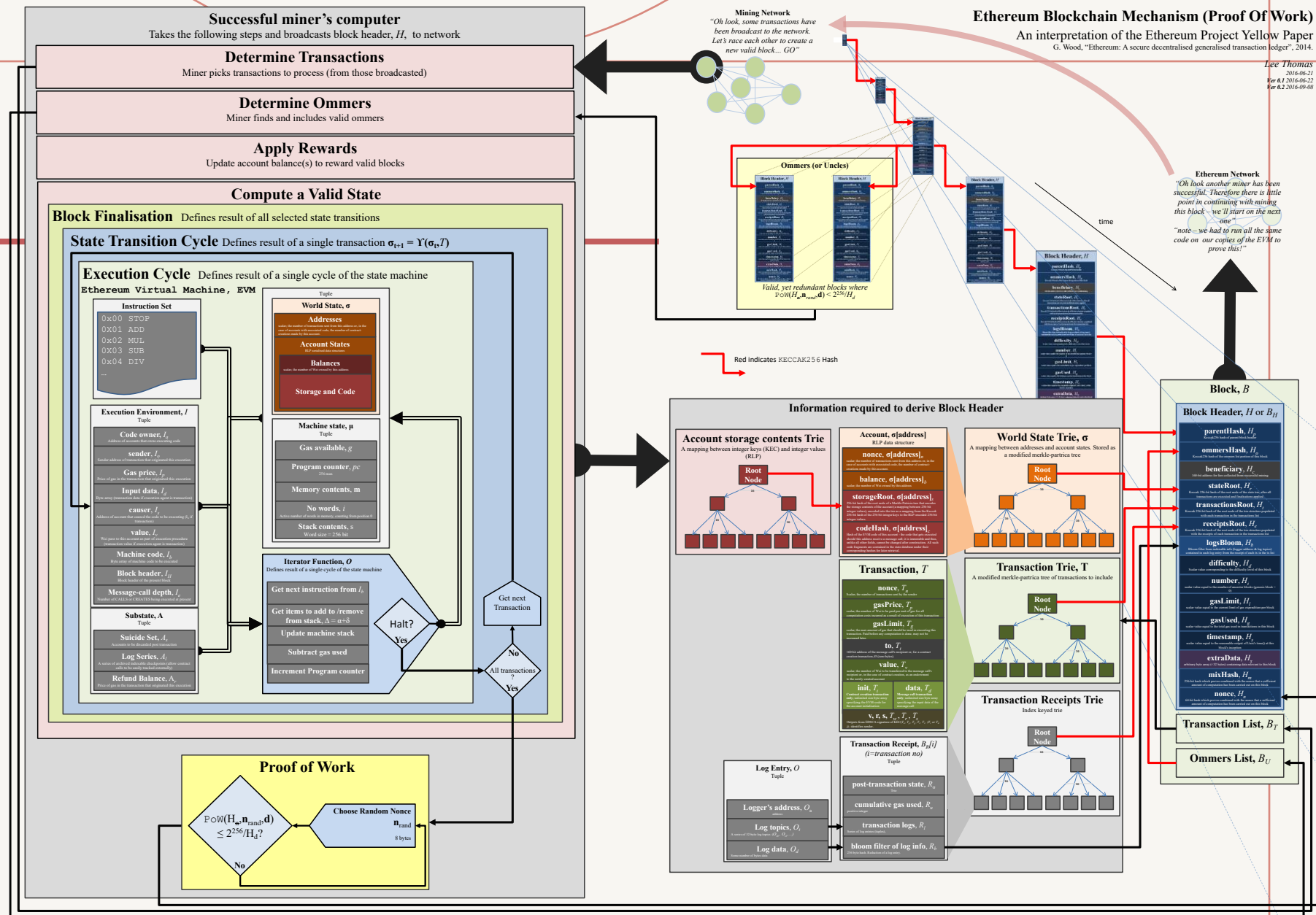
Ethereum blockchain



Ethereum blockchain



Ethereum blockchain



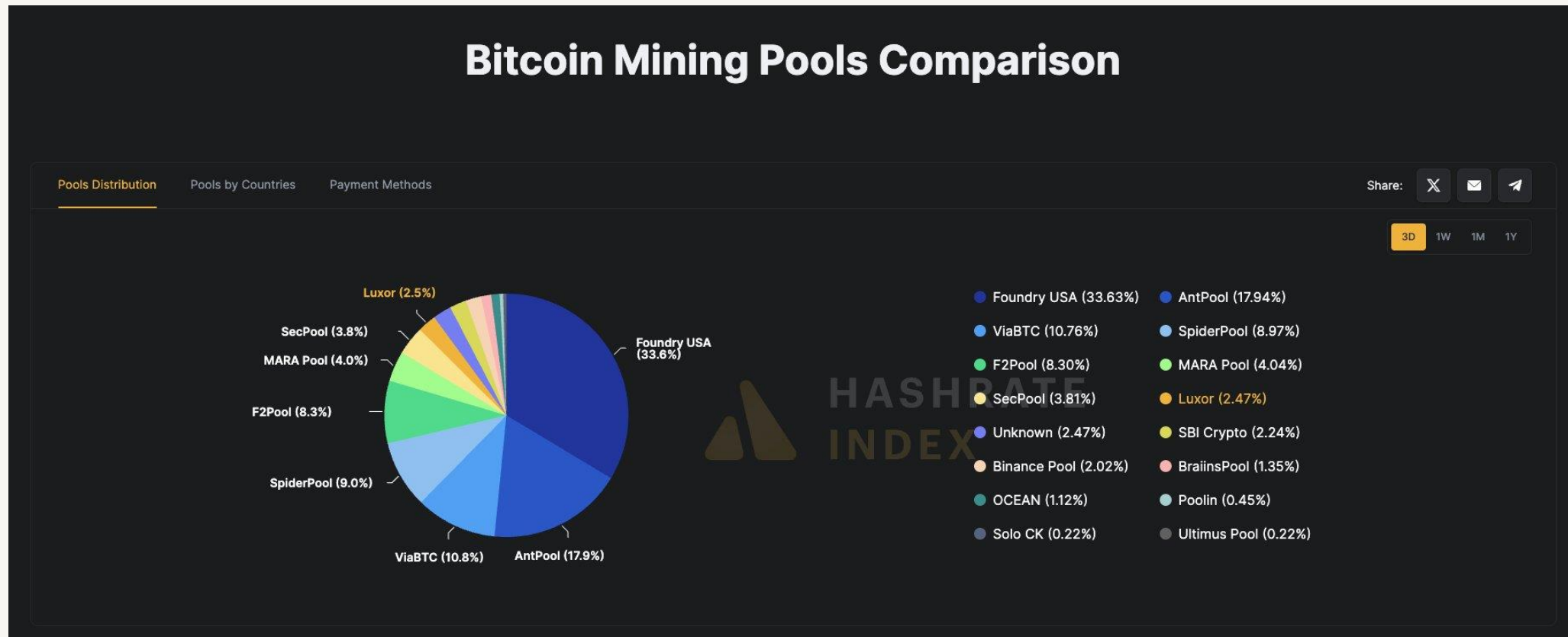
Bitcoin consensus

- The PoW algorithm utilised in Bitcoin is called a Compute-bound consensus algorithm
- A Compute-bound PoW, also known as CPU-bound PoW, employs a CPU-intensive function
 - that carries out the required computational task by leveraging the capabilities of the processing units (e.g., CPU/GPU)
 - and it does not rely on the main memory of the system
- These particular characteristics can be massively optimised for faster calculation by using Application-specific Integrated Circuit (ASIC) rigs

Bitcoin consensus

- This is not an ideal scenario as now general people with their general purpose computer cannot participate in the mining process
- The mining process is mostly centralised among a group of mining nodes
- Many crypto-currency enthusiasts suggest that this is not a democratic process and facilitates the “rich getting richer” scenario

Bitcoin consensus



Question?

