Blockchain and Distributed Ledger technologies

SAPIENZA
UNIVERSITÀ DI ROMA

Massimo La Morgia
massimo.lamorgia@uniroma1.it

From Merkle tree to MPT

In Bitcoin transactions are static and should never change after being committed.
They are “set in stone” via the Merkle hash root construction.

However, Merkle trees are not a data structure fit for editing.

Ethereum, unlike Bitcoin, must maintain a dynamic global state: account balances, nonces, contract storage, and code hashes change continuously.

Storing such mutable information in a plain Merkle tree would be expensive and inefficient.

This is why Ethereum relies on the Patricia Merkle Trie, a hybrid structure that preserves the cryptographic guarantees of a Merkle tree while enabling efficient updates, insertions, and lookups.

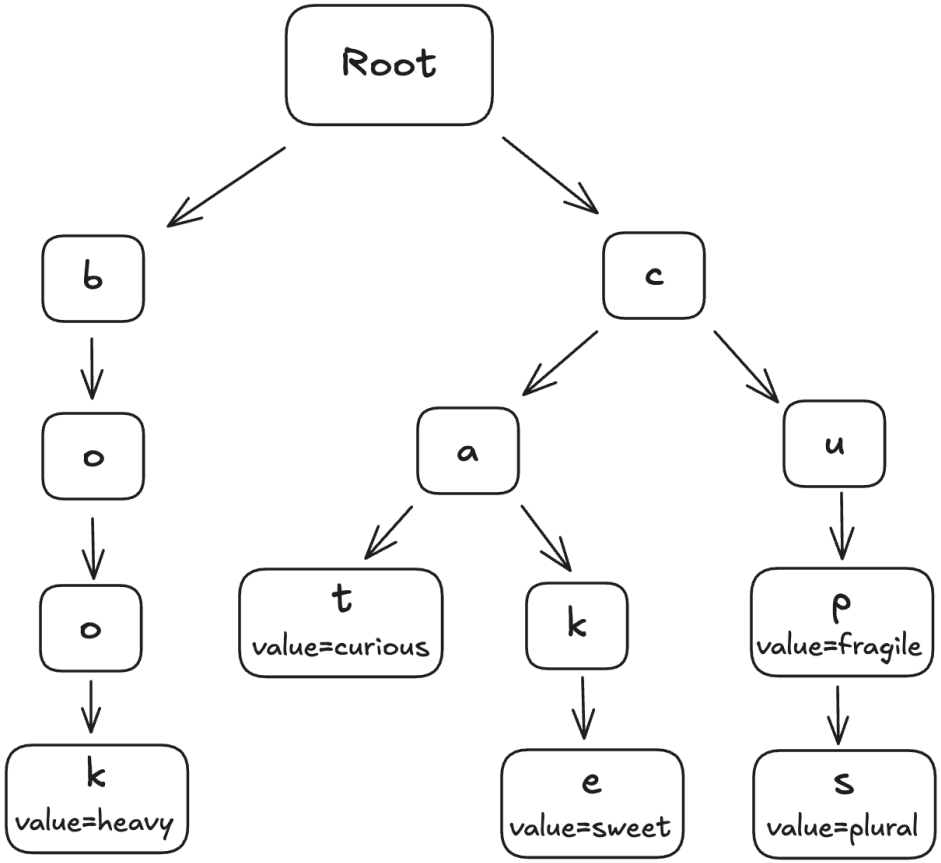
A Trie

A Trie (from re**trie**val) is a tree-like data structure used to efficiently store and retrieve key-value pairs, especially when the keys are strings or sequences.

Each level of the trie represents a character in the key, and the path from the root to a leaf corresponds to an entire key. Shared prefixes between keys are stored only once, making tries very space-efficient for datasets with common prefixes.

In a the nodes represent the symbols of the alphabet (binary or hex), while the leaf contains the final value.

KEY	VALUE
cat	curious
cake	sweet
cup	fragile
cups	plural
book	heavy

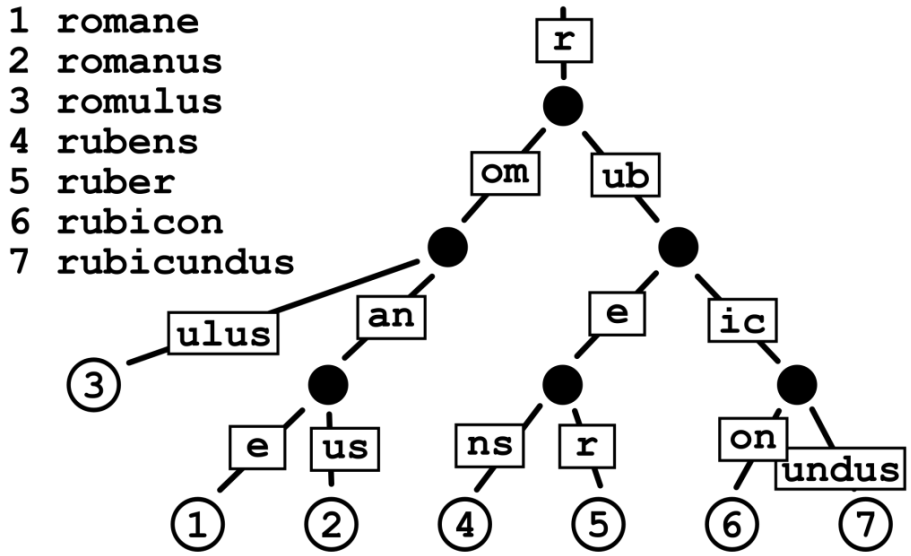


Radix Trie or Patricia trie

A radix trie or Patricia Trie (a special version of Radix trie) , is an optimization of the Trie that combines nodes with single children to reduce space complexity.

It compress common path.

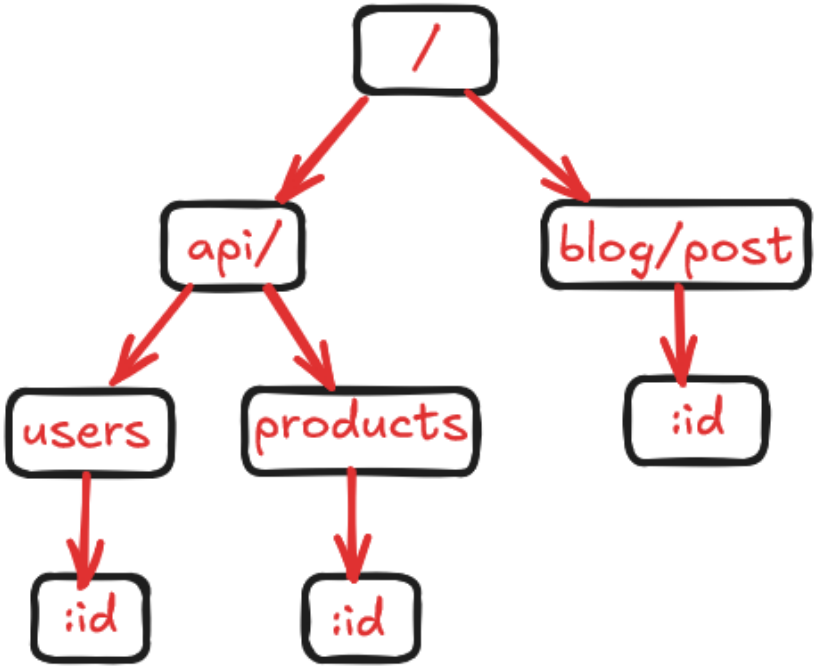
It retains the same hierarchical structure but eliminates unnecessary nodes, making it more memory-efficient.



PATRICIA = **P**ractical **A**lgorithm **T**o **R**etrieve **I**nformation **C**oded **I**n **A**lphanumeric

Use cases - Radix Trie or Patricia trie

```
fastify.get('/api/users', handleGetUsers)
fastify.get('/api/users/:id', handleGetUser)
fastify.get('/api/products', handleGetProducts)
fastify.get('/api/products/:id', handleGetProduct)
fastify.get('/blog/posts', handleGetBlogPosts)
fastify.get('/blog/posts/:id', handleGetBlogPost)
```



Use cases - Radix Trie or Patricia trie

Let's see how it works mby inserting your names!

https://www.cs.usfca.edu/~galles/visualization/RadixTree.html

Merkle Patricia Trie

Ethereum uses a specialized form of trie called the Modified Merkle Patricia Trie (MPT). The name combines three core ideas:

- Trie: For organizing keys by shared prefixes.
- Merkle: Every node is hashed, forming a Merkle structure that enables cryptographic verification of the entire dataset.
- Patricia: a variant of a trie that compresses paths where nodes have a single child (also called radix or compact trie).

Merkle Patricia Trie

In a Patricia Merkle Trie there are 4 types of nodes:

1. NULL node: <>
2. **Branch node**: 17-items, the first 16 items encode a step of the path, the last item contains a value
3. Leaf node: it is made of 2 items Path, value
4. **Extension node**: it is made of 2 items Path, Key

Nibble in Merkle Patricia Trie

In Ethereum data is encoded in hexadecimal form, and thus the path are made of nibble.

A nibble is a group of 4 bit.

e.g. 0xA7 **A** is a nibble and **7** is a nibble.

However, data are stored in the form of byte (2 nibbles).

Thus, When traversing paths in nibbles, we may end up with an odd number of nibbles to traverse, but because all data is stored in bytes format.

How we can distinguish the the nibble **1** from the nibbles **01** (both must be stored as <01>).

To specify odd length, the partial path is prefixed with a flag. Flag nibbles are used to avoid ambiguity

hex char	bits	node type partial	path length
0	0000	extension	even
1	0001	extension	odd
2	0010	terminating (leaf)	even
3	0011	terminating (leaf)	odd

Extension and Brance node

Extension node

An extension node is created when multiple keys share the same path (nibbles).

After an extension node, there is always a branch node or a leaf node.

Path	Key (Hash Pointer)

Shape of an extension node: [Path, Key] where he Key is a Hash pointer

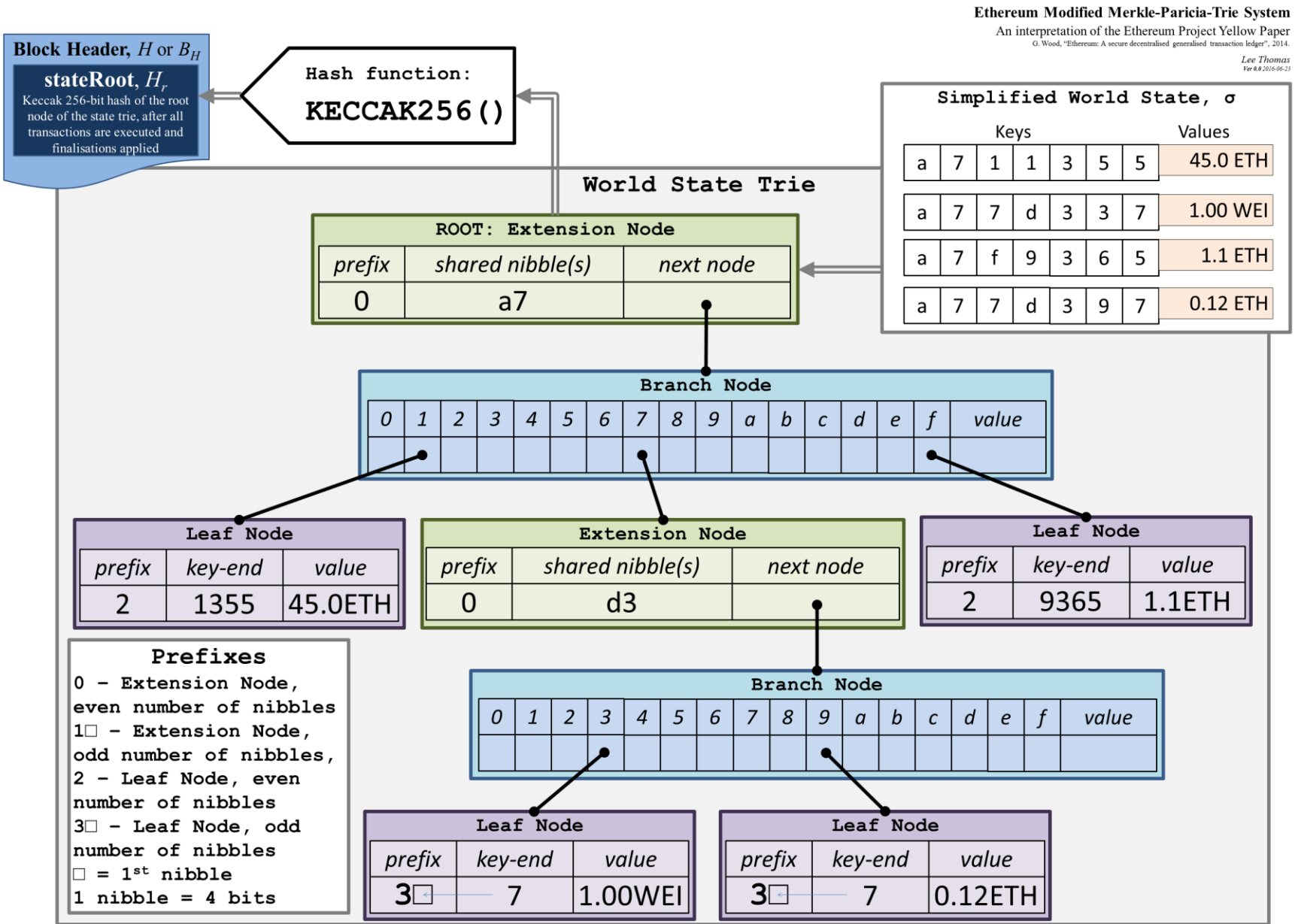
Branch node

A Branch node is created when the paths of the keys diverge.

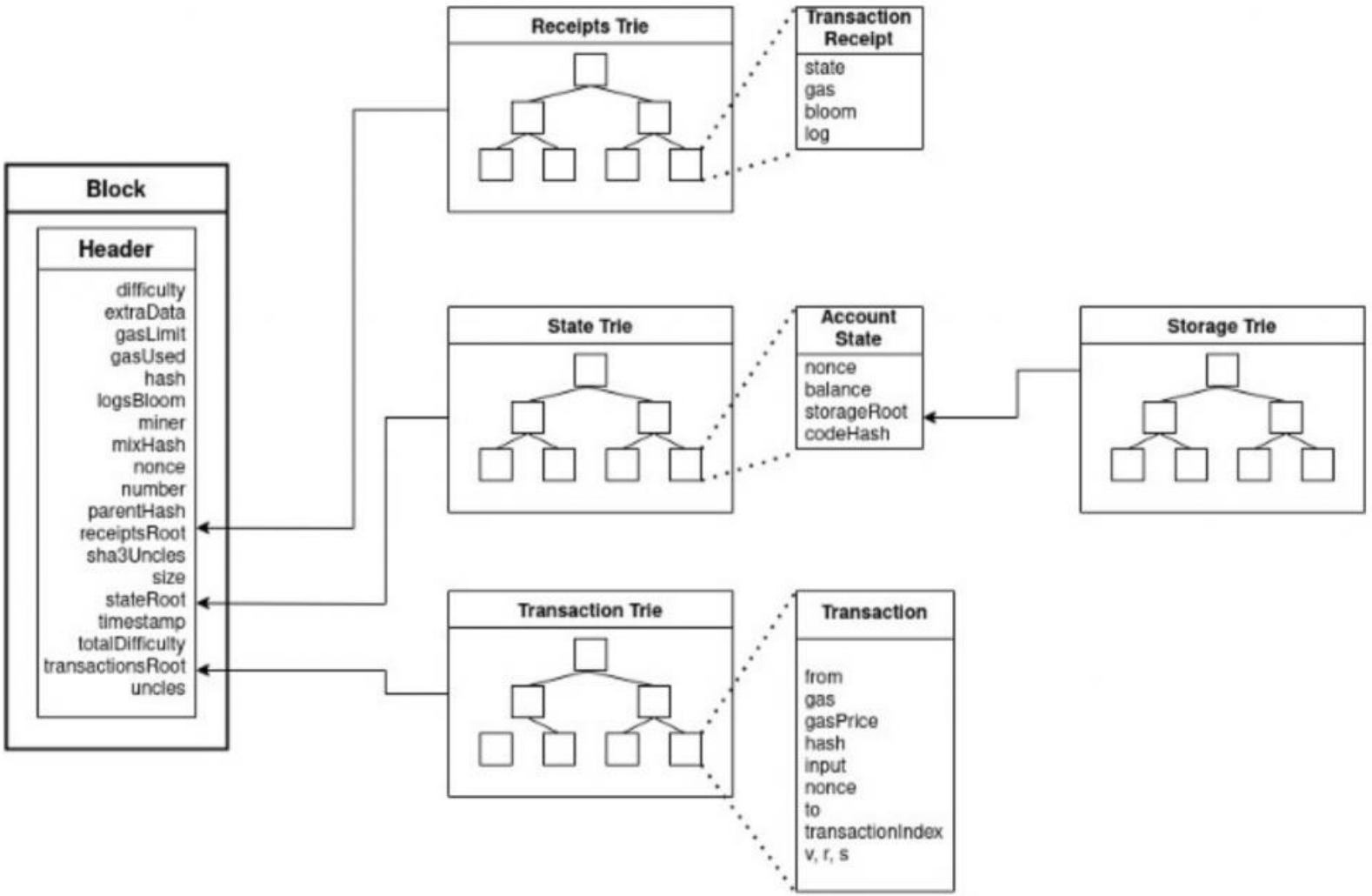
In other words, if the keys have different nibble at the same path length.

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

Merkle Patricia Trie



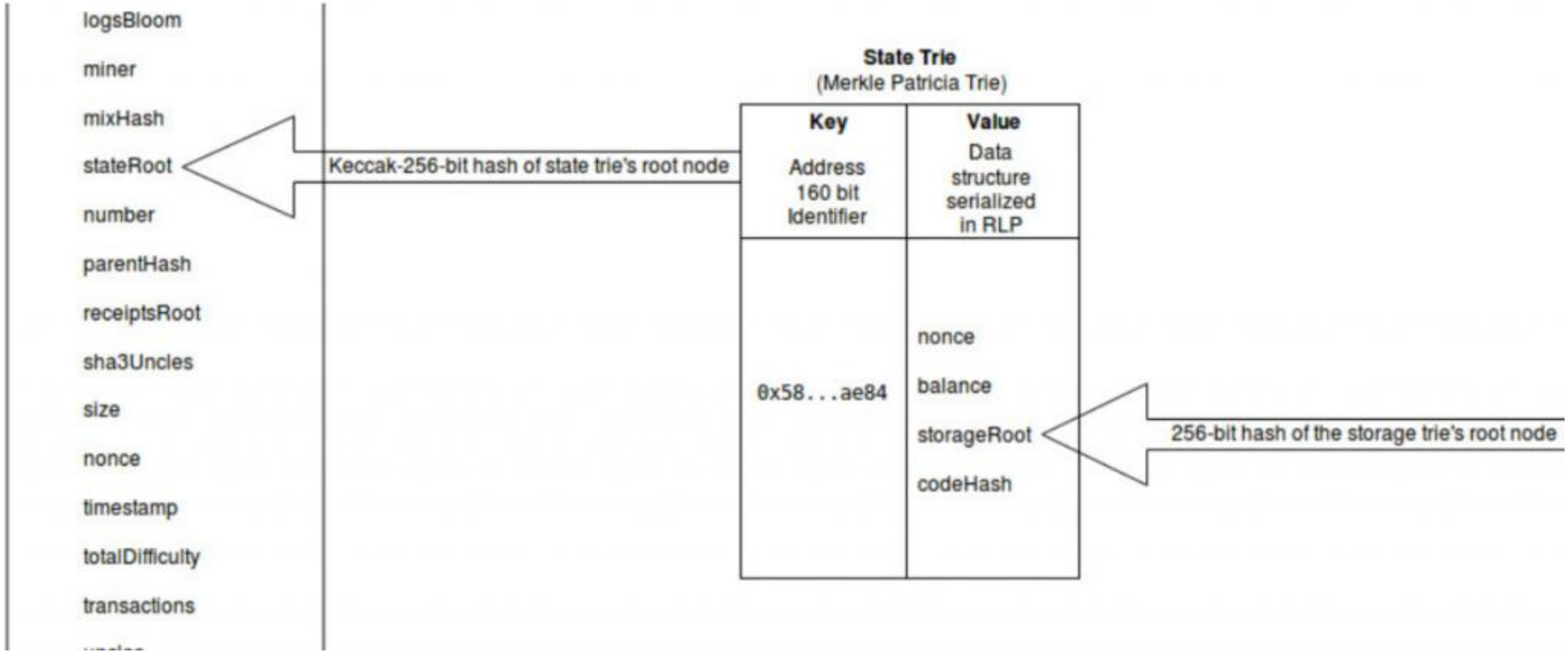
Ethereum tries



State trie

There is one global state trie, and it is updated every time a client processes a block.

The keys of this trie are the Ethereum addresses, while the values are the values that define an account: Nonce, Balance, StorageRoot and CodeHash. The Storage root is the roof of another Patricia Trie.



Storage trie

It is the Patricia Trie containing the data related of the smart contract.

Each account has a separate storage trie.

It is not part of the block.

Remember that the storageRoot for an Ethereum account is empty by default if it's not a contract account.

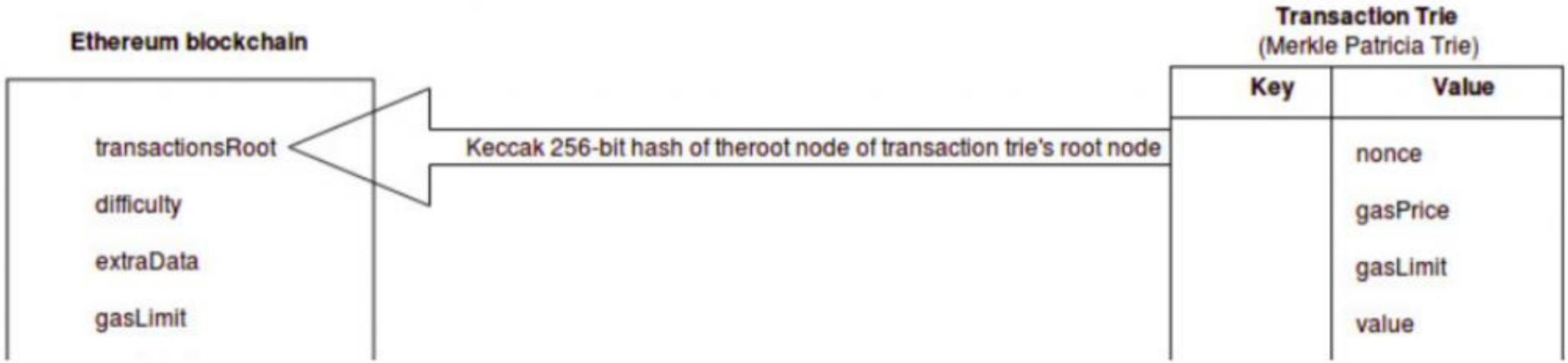
Transaction trie

Every block has a separate transaction trie.

It records the transactions happened in that block. Once created and embedded into a block it will never modified.

The keys of this Patricia trie are the Transaction Index.

The value associated to each key are the information related to the transaction like: the nonce, gasPrice, gasLimit and the value (amount of ETH)



Receipt trie

Like the transaction trie the receipt trie is never updated.

Also in this cases the keys are the transactionIndexes, while the values are the gasUsed by the transaction, logs or event emitted by the smart contracts with which the transaction interacted.