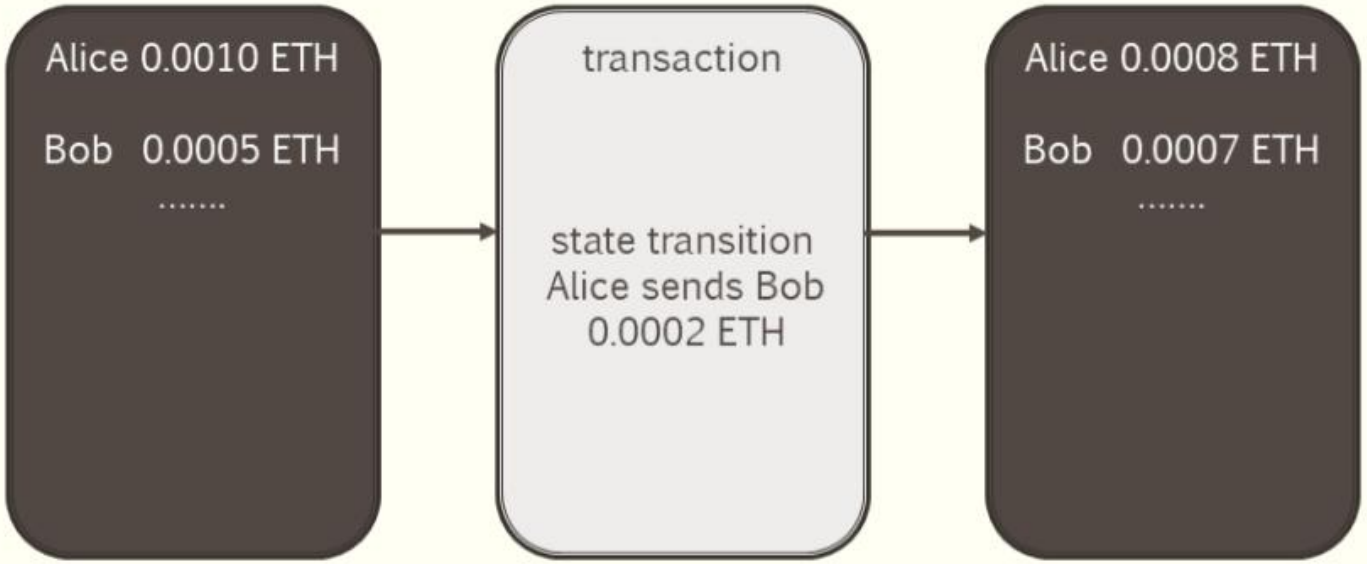
ETHEREUM

Blockchain technologies, **lecture 3**

Course overview

- Ethereum -- **transaction-based state machine**
- Ethereum accounts
  - Externally-owned accounts
  - Smart contracts
- Transaction structure
- Merkle-Patricia Trie
- EVM and code execution



Ethereum -- Bitcoin

ACCOUNT MODEL

- Key – Value.
- Direct access to balance.
- Fungible coins.
- Smaller transactions.
- Easy to scale with layer 1 solutions.

UTXO

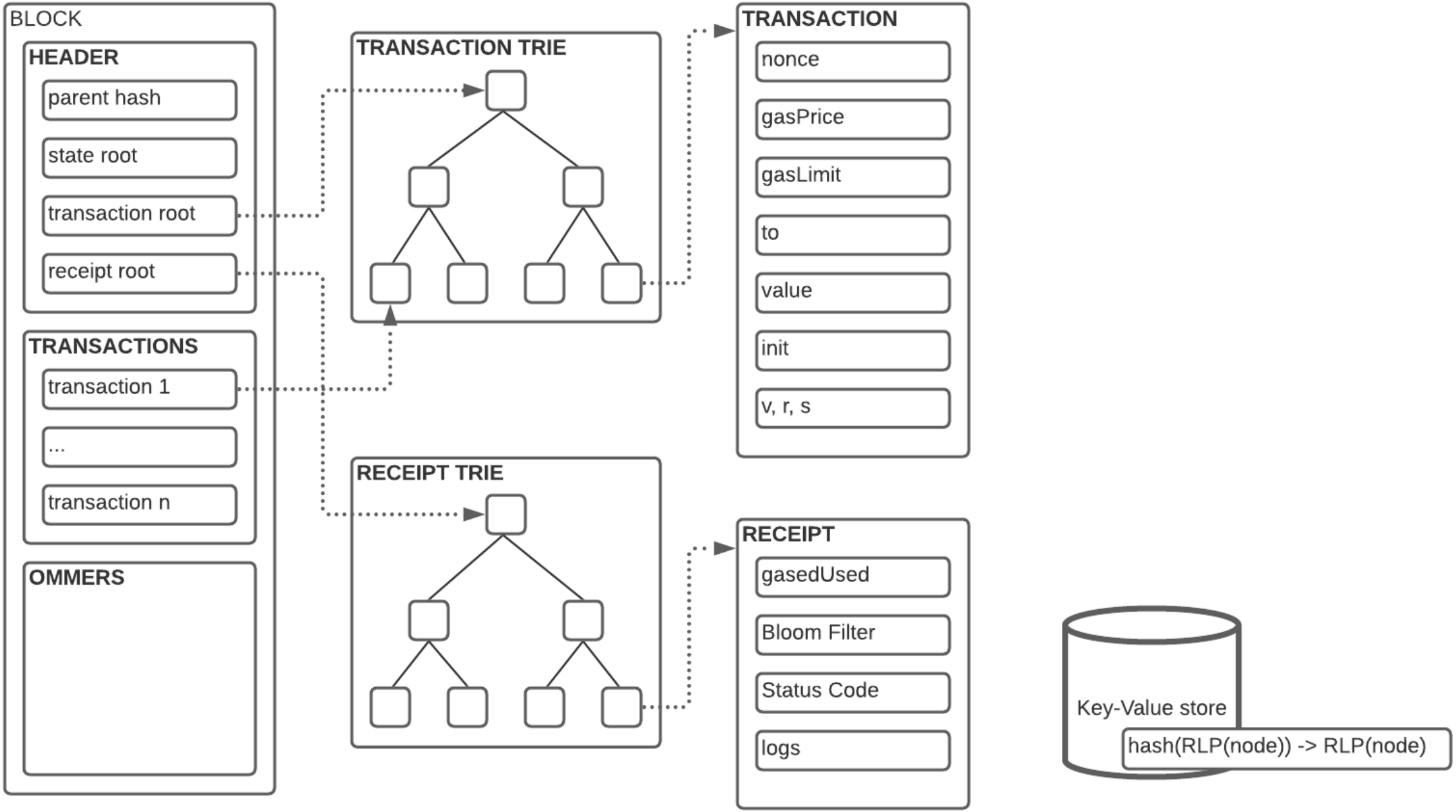
- DAG.
- Balance is not stored.
- Each UTXO token is unique.
- Memory consuming.
- Easy to scale with layer 2 solutions.

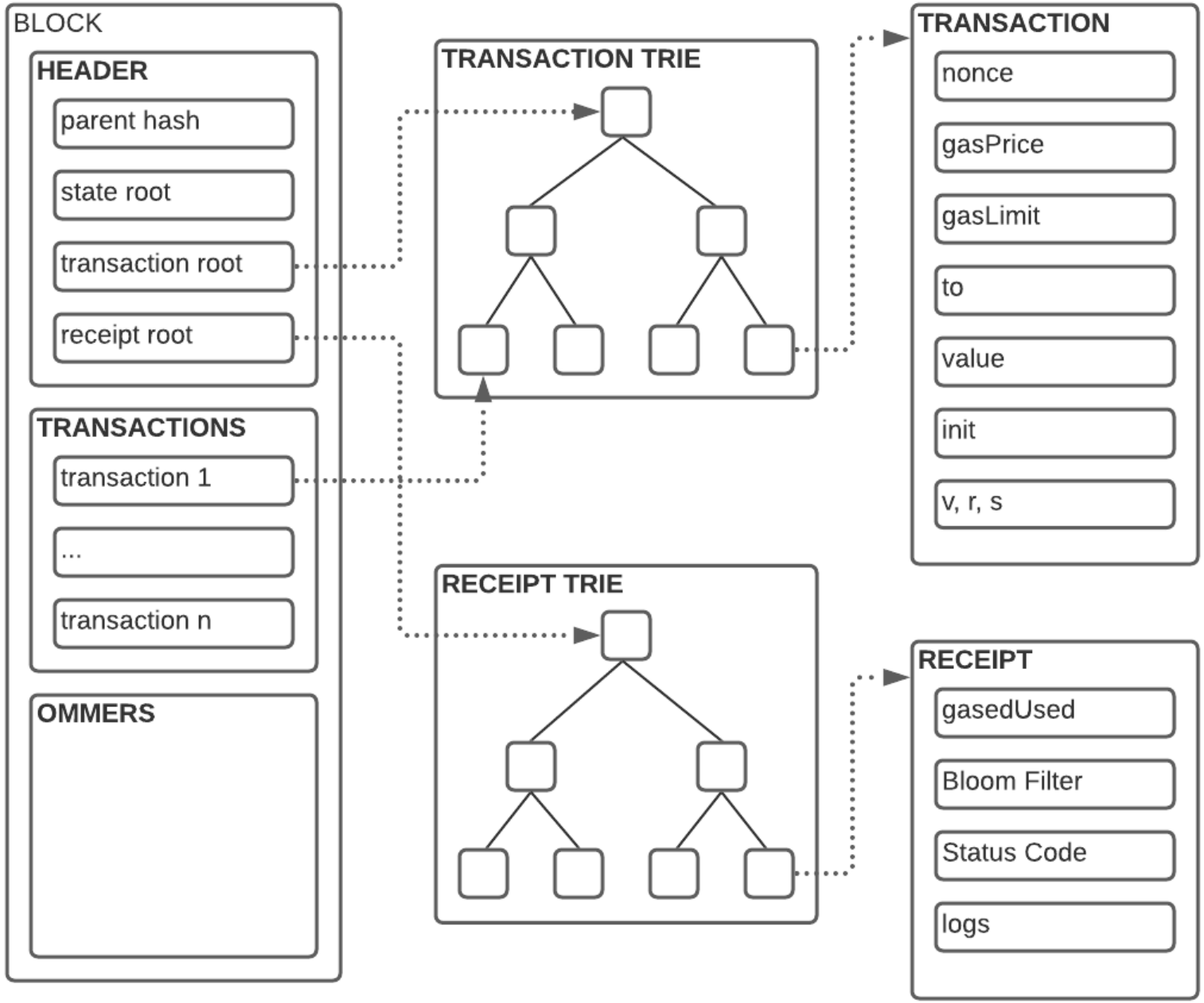
ETHEREUM STATE MACHINE

ETHEREUM STATE

Ethereum state machine

- Ethereum ACCOUNT MODEL – transaction-based state machine
- State Patricia Merkle Tries:
  - STATE ROOT stores accounts:
    - Externally-owned accounts
    - Smart contracts, account with storage and code
  - TRANSACTION ROOT
    - Transfer of ETH
    - Smart contract creation
    - Smart contract execution
  - RECEIPT ROOT – transaction inputs, logs, costs, block-number etc.


BLOCK
HEADER
parent hash
state root
transaction root
receipt root
TRANSACTIONS
transaction 1
...
transaction n
OMMERS
TRANSACTION TRIE
RECEIPT TRIE
TRANSACTION
nonce
gasPrice
gasLimit
to
value
init
v, r, s
RECEIPT
gasedUsed
Bloom Filter
Status Code
logs
Key-Value store
hash(RLP(node)) -> RLP(node)



Receipts – effect of transactions

check log hash
bloom filter – false positives

STATE TRIE

- Stores accounts.
- Modified Merkle Patricia Trie.

- The fields of an account are:

  - nonce: number of transactions initiated from the account, for externally-owned accounts, or the number of contracts created by a contract account.
  - balance: indicates the number of wei owned by an account (EOA or smart contract)
  - codeHash: empty for externally-owned accounts. Contains the code of a smart contract.
    immutable.
  - storageRoot: empty for externally-owned accounts. A 256-bit hash of the root node of a Merkle Patricia Trie that encodes the storage contents of the account.
    accessible for reads, not accessible for writes from another contract.

TRANSACTION TRIE

- Stores transactions.
- There are three types of transactions:
  - **Transfers:** the transaction changes sender and receiver balances.
  - **Contract creation:** the result of transaction is the creation of a new contract account.
  - **Contract call:** smart contract code execution.

Externally owned	**Contract**
created freely	cost: gas converted in eth (fee for transaction of type b or c.)
controlled by a pair of public-private keys	controlled by contract-code
have balance and nonce, code-Hash and storageRoot are empty	have balance, nonce, code-Hash and storageRoot
can initiate transactions: transfer funds, create contracts, execute contract code	can execute transfers or other types of transactions only as response to a transaction

Ethereum state machine

- Ethereum – **transaction-based state machine**
- State transition function: Y(S,T) = S'
- State transition function changes tries, in particular it changes storage tries.
- Turning complete (pseudo-complete due to gas limits)
- EVM code execution – stack machine
  - OP CODES: ADD, SUB etc. ADDRESS, BALANCE, BLOCKHASH
  - Each operation has a gas cost.

EVM - STORAGE

- Each account has a data area called storage.
- All contract data must be assigned to a location (storage or memory)
- A contract can only read or write to its own storage.
  - Key-value store. Maps 256-bit words to 256-bit words.
- Values stored in storage are stored permanently on the blockchain.
- Modifying storage is costly and should be avoided.

EVM – MEMORY AND STACK

- Memory -- Values only stored for the lifetime of a contract function's execution.
- Cheaper, not permanently stored.
- Writes can be either 8 bits or 256 bits wide.

.

- Stack maximum size of 1024 elements and contains words of 256 bits
  - Swap between top 16 elements and top, copy from top 16 elements, operation with topmost two elements.
- Each operation has a gas cost

EVM – EXECUTION

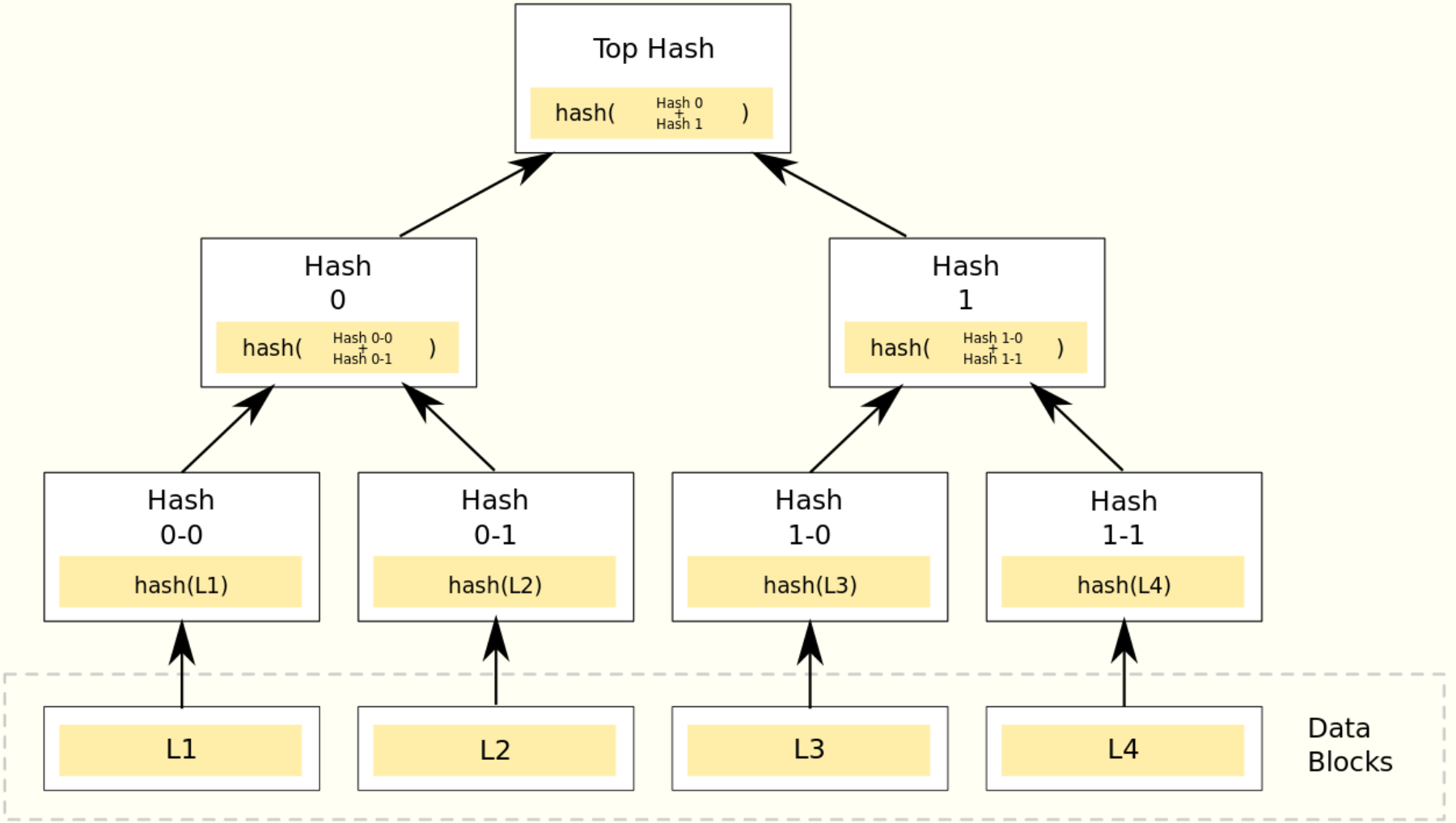
- bytecode is executed on the EVM as a set of EVM opcodes.
- opcodes can refer to read or writes to storage.
- Deterministic stack execution model.
- All Ethereum execution clients include an implementation of the EVM.
- Each operation has a gas cost.
- Full nodes validate and execute transactions.

MERKLE-PATRICIA TRIE

MERKLE-PATRICIA TRIE

- MERKLE TREE or hash trees are named after Ralph Merkle, 1979.
- Every leaf node is labelled with the cryptographic hash of a data block.
- Every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes.
- Allows efficient and secure verification of large data
- Example of **commitment scheme**.

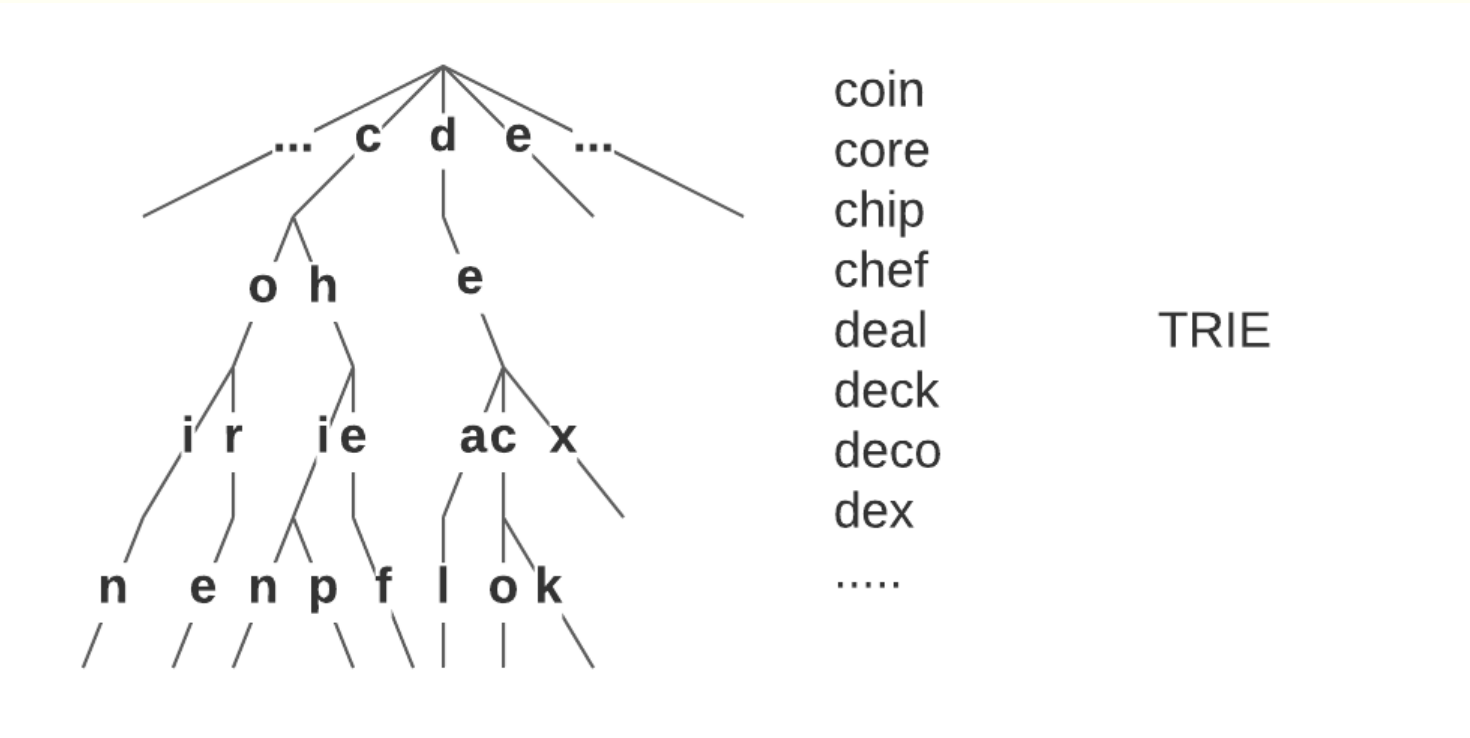
MERKLE-PATRICIA TRIE – hash trees



MERKLE-PATRICIA TRIE

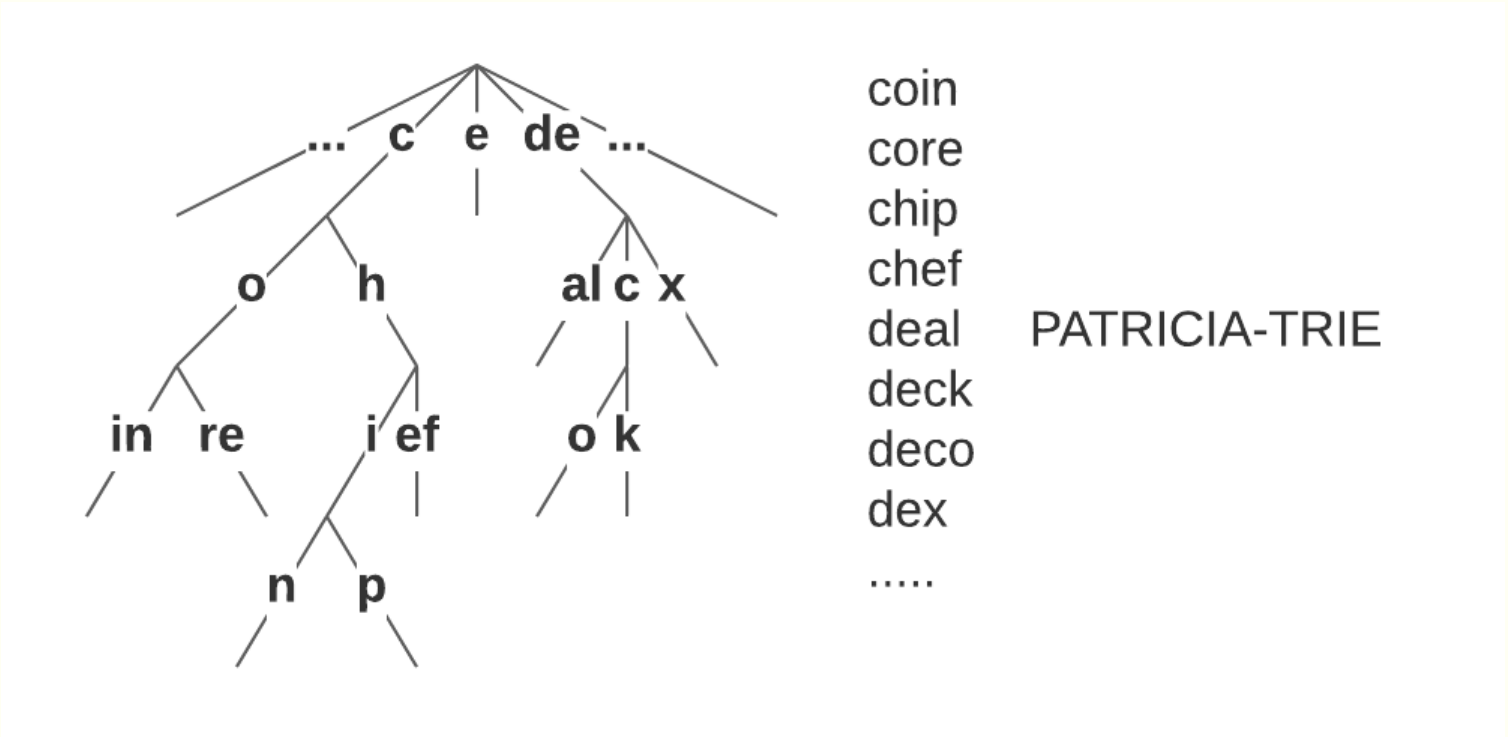
- PATRICIA derived from the Latin word patrician, meaning "noble".

Practical Algorithm to Retrieve Information Coded in Alphanumeric, D.R.Morrison (1968)



MERKLE-PATRICIA TRIE

- PATRICIA optimized trie - each node that is the only child is merged with its parent.

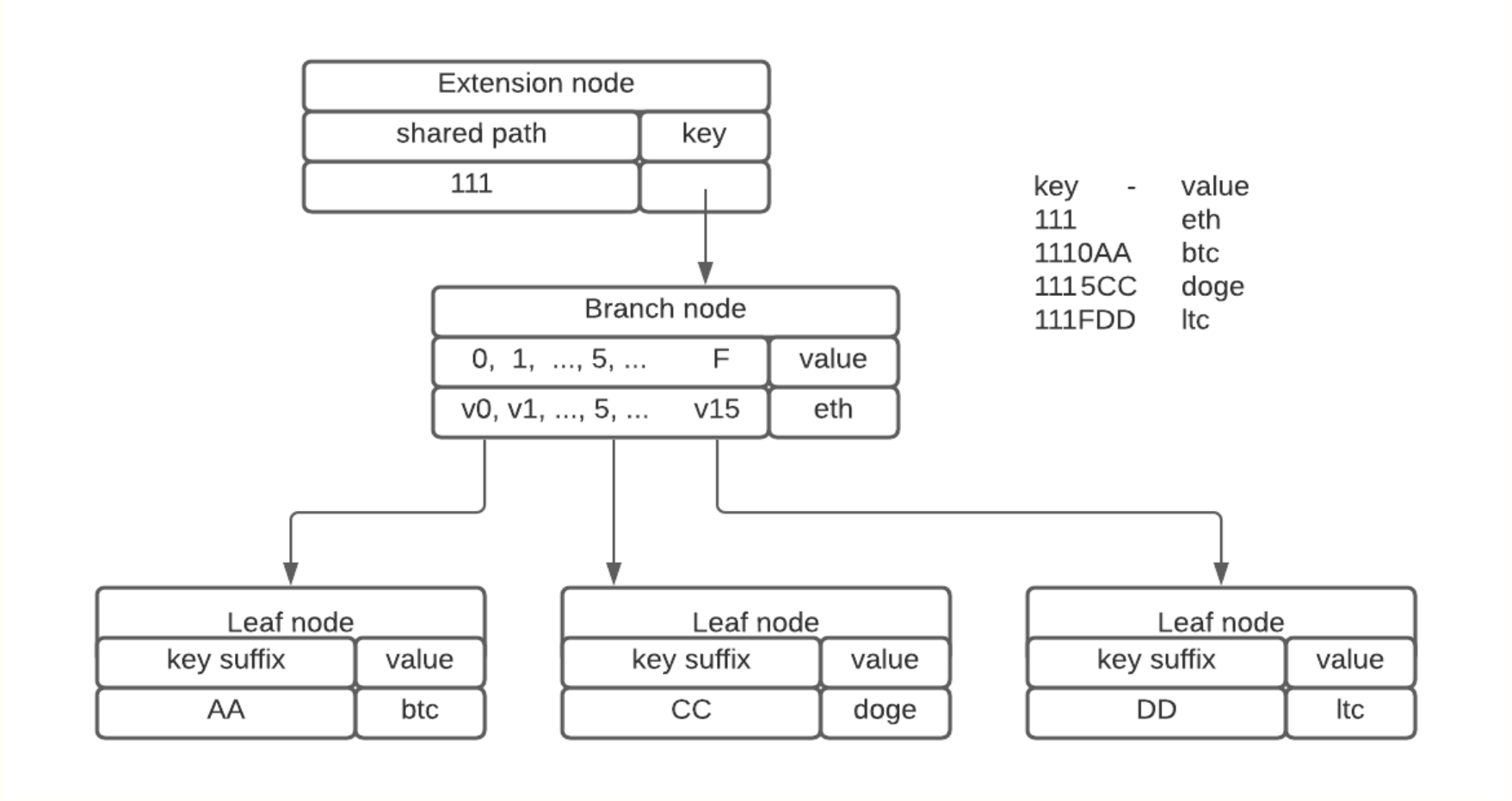


MERKLE-PATRICIA TREEs

- Hashing in the sense of MERKLE tree, a child node is referred by its cryptographic hash.
- Groups common prefixes in the sense of PATRICIA tries, branches only when branching is necessary.
- Allows 16 branches.
- Three types of nodes:
  - Branch node — $[i_0, i_1, \ldots, i_{15}, \text{value}]$ — The *i*–position contains a link to a child node. The 17^{th} item is used in the case of terminal nodes
  - Extension — [path, value] — Compression, stores common path for multiple keys, *value* links to a child node.
  - Leaf — [path, value] — Compression, terminal node.

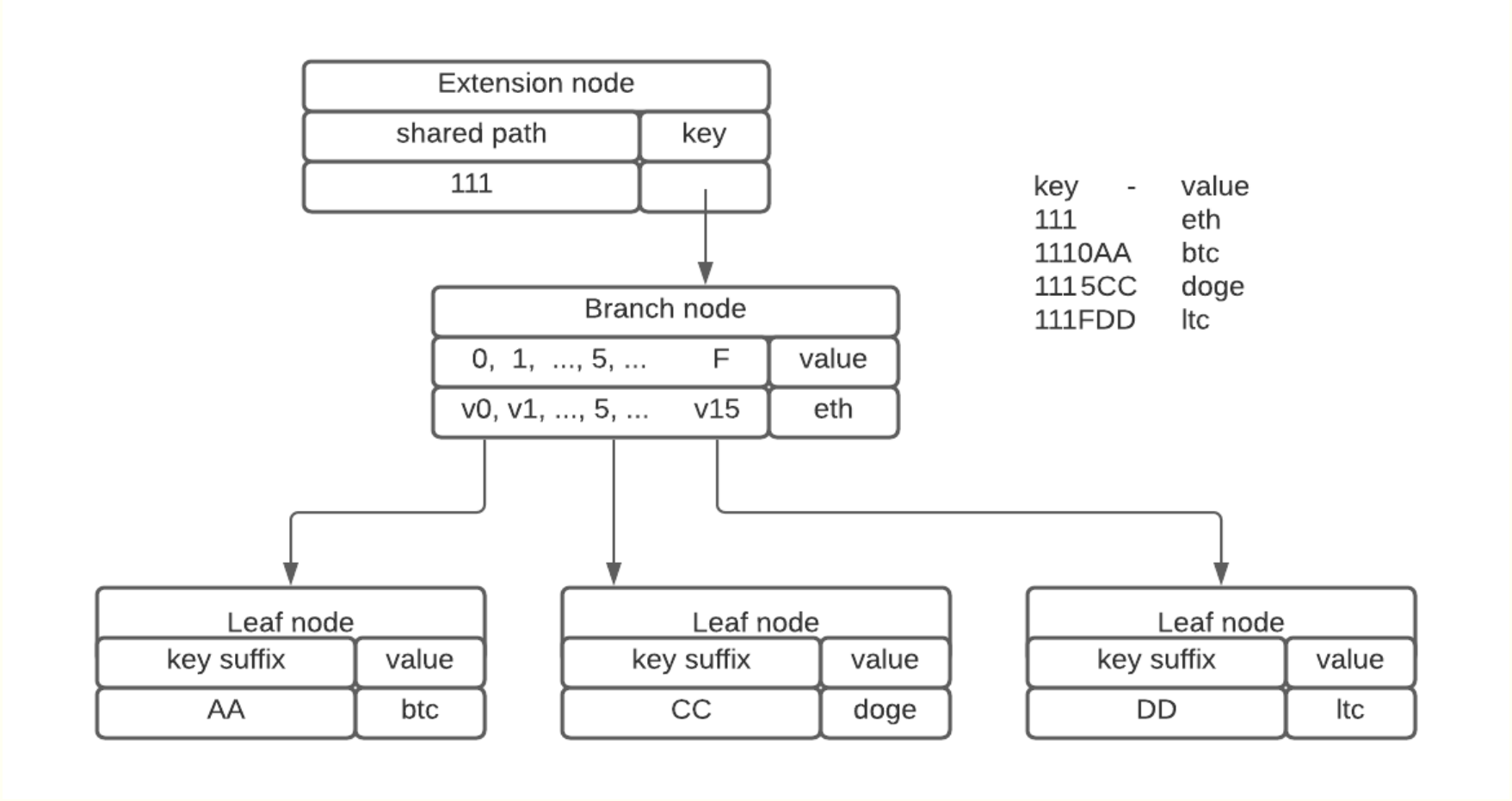
MERKLE-PATRICIA TREEs

- *MARKLE* PATRICIA TREE - Groups common prefixes in the sense of PATRICIA tries.
- use key-vale storages, RocksDB (Parity) LevelDB (Geth)



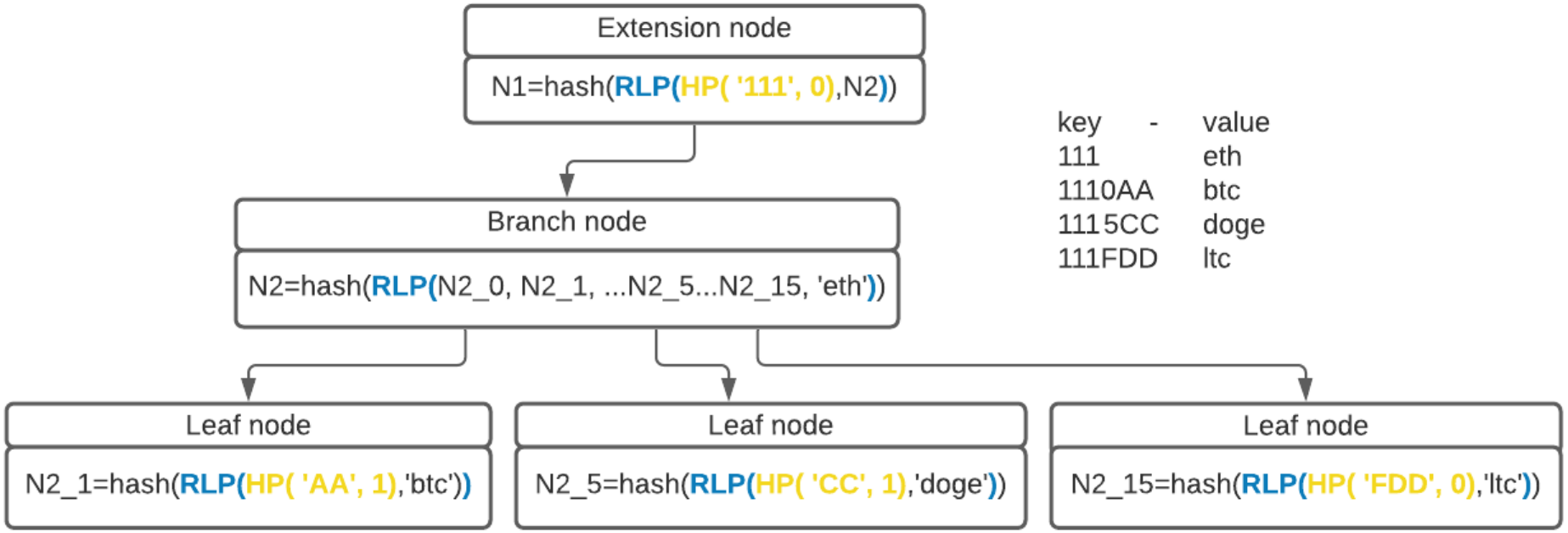
MERKLE-PATRICIA TREEs

- *MARKLE* PATRICIA TRIE - Groups common prefixes in the sense of PATRICIA tries.
- key-vale storages, RocksDB (Parity) LevelDB (Geth)



MERKLE-PATRICIA TREEs

- *MARKLE* PATRICIA TREE - a child node is referred by its cryptographic hash.
- RLP function, serializes a set of input arrays into one array.
- HP encodes node structures into compressed byte arrays.
- hash = keccak implementation of SHA3



MERKLE-PATRICIA TREEs Hex Prefix encoding

- $f(t) = \begin{cases} 2, & t = 1 \\ 0, & t = 0 \end{cases}$ HP encodes a path stream so that two (4-bit) nibbles compose one (8-bit) byte.
- $\mathrm{HP}(x,t) = \begin{cases} (16f(t), 16x[0] + x[1], 16x[2] + x[3], \dots), & \|x\| \; is \; even \\ (16(f(t)+1) + x[0], 16x[1] + x[2], 16x[3] + x[4], \dots), & \|x\| \; is \; odd \end{cases}$

t	$\|x\|$	x	prefix	X[0] or 0	Grup 1	Grup 2
t=0 extension	even	0xa, 0xb, 0xc, 0xd	0000	0000	1010 1011	1100 1101
	odd	0x09, 0xa, 0xb, 0xc, 0xd	0001	1001	1010 1011	1100 1101
t=1, leaf	even	0xa, 0xb, 0xc, 0xd	0010	0000	1010 1011	1100 1101
	odd	0x09, 0xa, 0xb, 0xc, 0xd	0011	1001	1010 1011	1100 1101

MERKLE-PATRICIA TREEs Recursive Length Prefix

- RLP function, serializes (flattens) a set of input arrays into one byte array.
- Flattened byte array is persisted into a key-value storage.
- Length Prefix Every sub-sequence is prefix by a byte encoding its length and a bit-mask to determine the sub-subsequence type: array or string.
- Recursive Every flattened sequence is prefixed by total length of all its sub-sequences.

x	$\|\|x\|\|$	RLP(x)	RLP(x)	$\|\|$RLP(x)$\|\|$
XYZ	3	128 + $\|\|x\|\|$, X, Y, Z	10000011 X Y Z	4
AB	2	128 + $\|\|x\|\|$, A, B	10000010 A B	3
[XYZ, AB]	2	192 + $\|\|$RLP(x_1), RLP(x_2)$\|\|$, RLP(x_1), RLP(x_2)	11000111 10000011 X Y Z 10000010 A B	8

MERKLE PROOFS

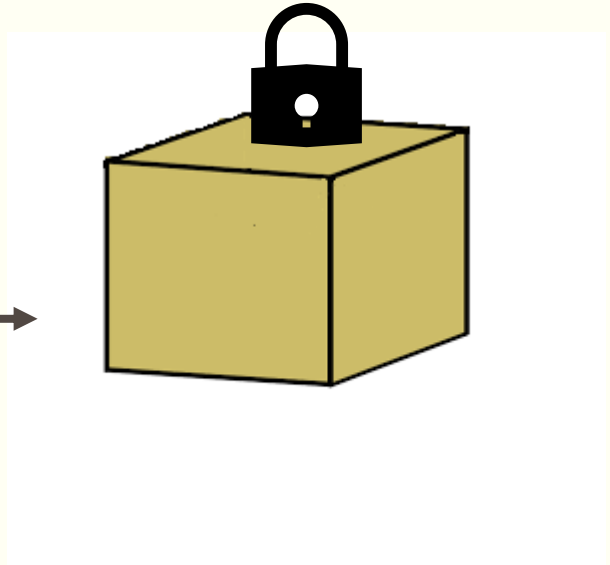
MERKLE-PATRICIA TRIE

- MERKLE TREE or hash trees are named after Ralph Merkle, 1979.
- Every leaf node is labelled with the cryptographic hash of a data block.
- Every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes.
- Allows efficient and secure verification of large data
- Example of **commitment scheme**. root hash represents the "commitment" to the value.

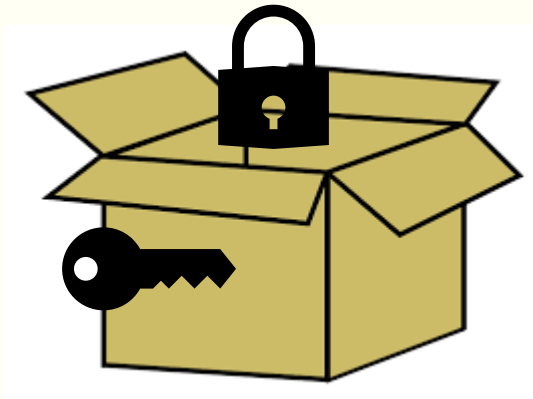
ALICE

BOB

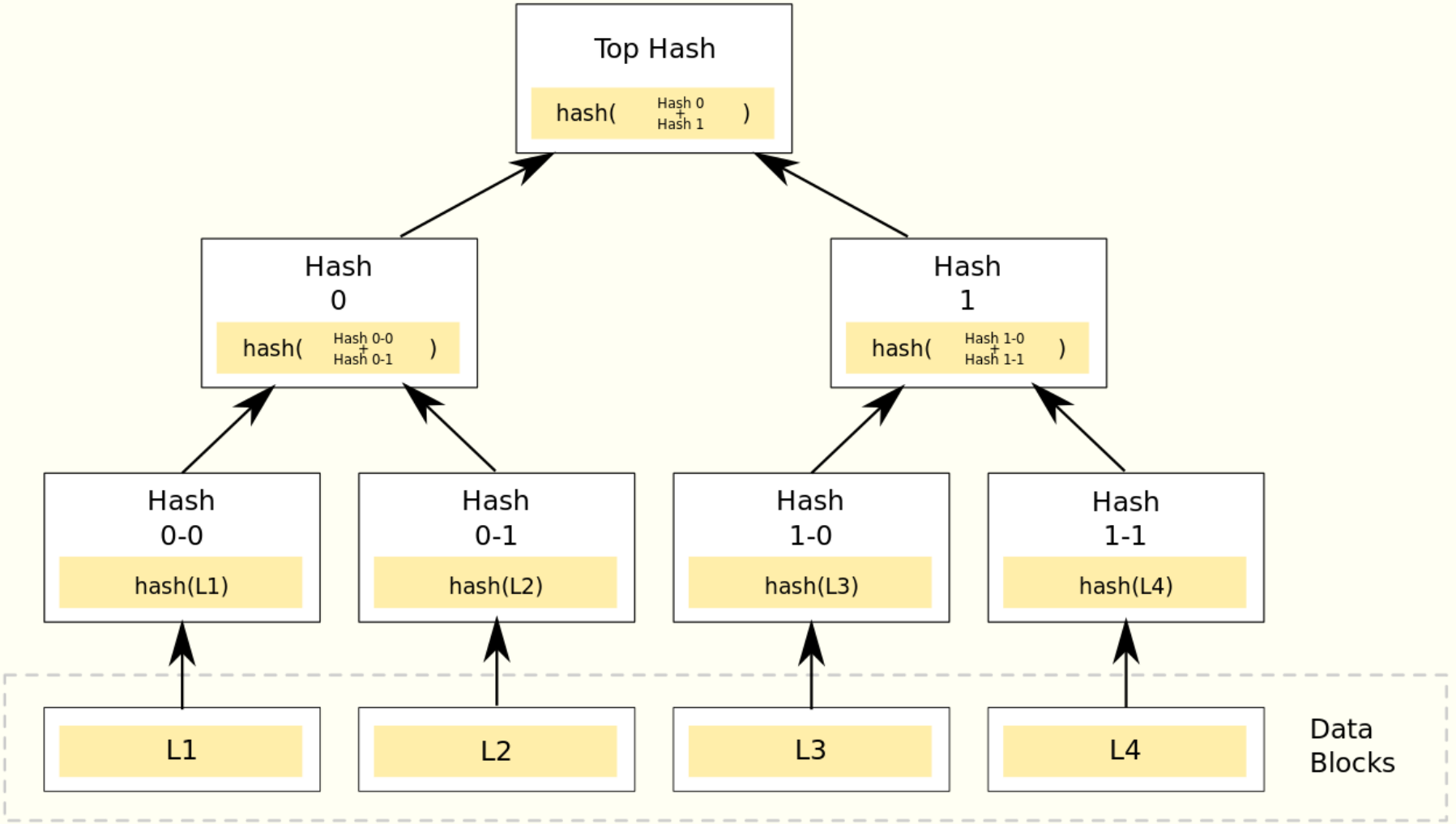
COMMIT (binding property)



REVEAL (hiding property)



MERKLE-PATRICIA TREEs – hash trees

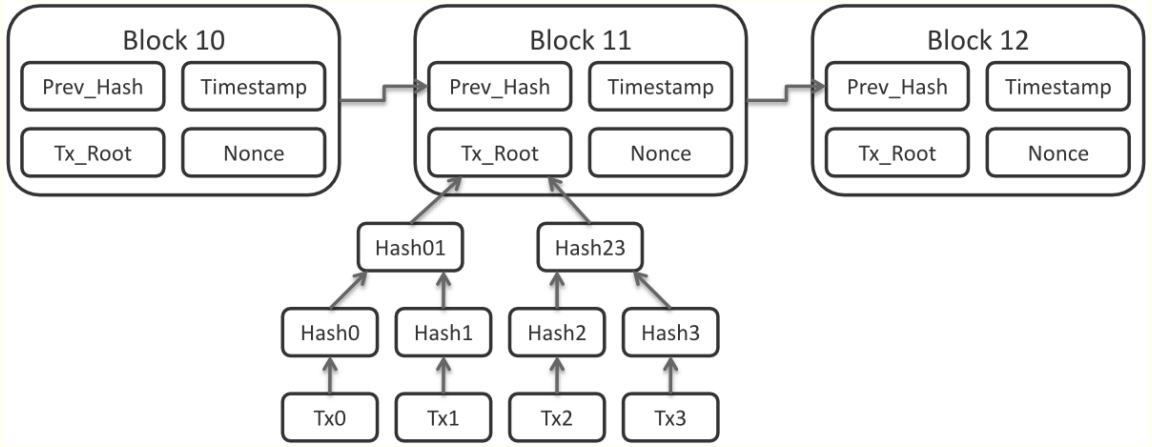


MERKLE PROOFS BITCOIN -- LightClients

Light clients download only block headers.

80-bytes chunks of data for each block that contain:

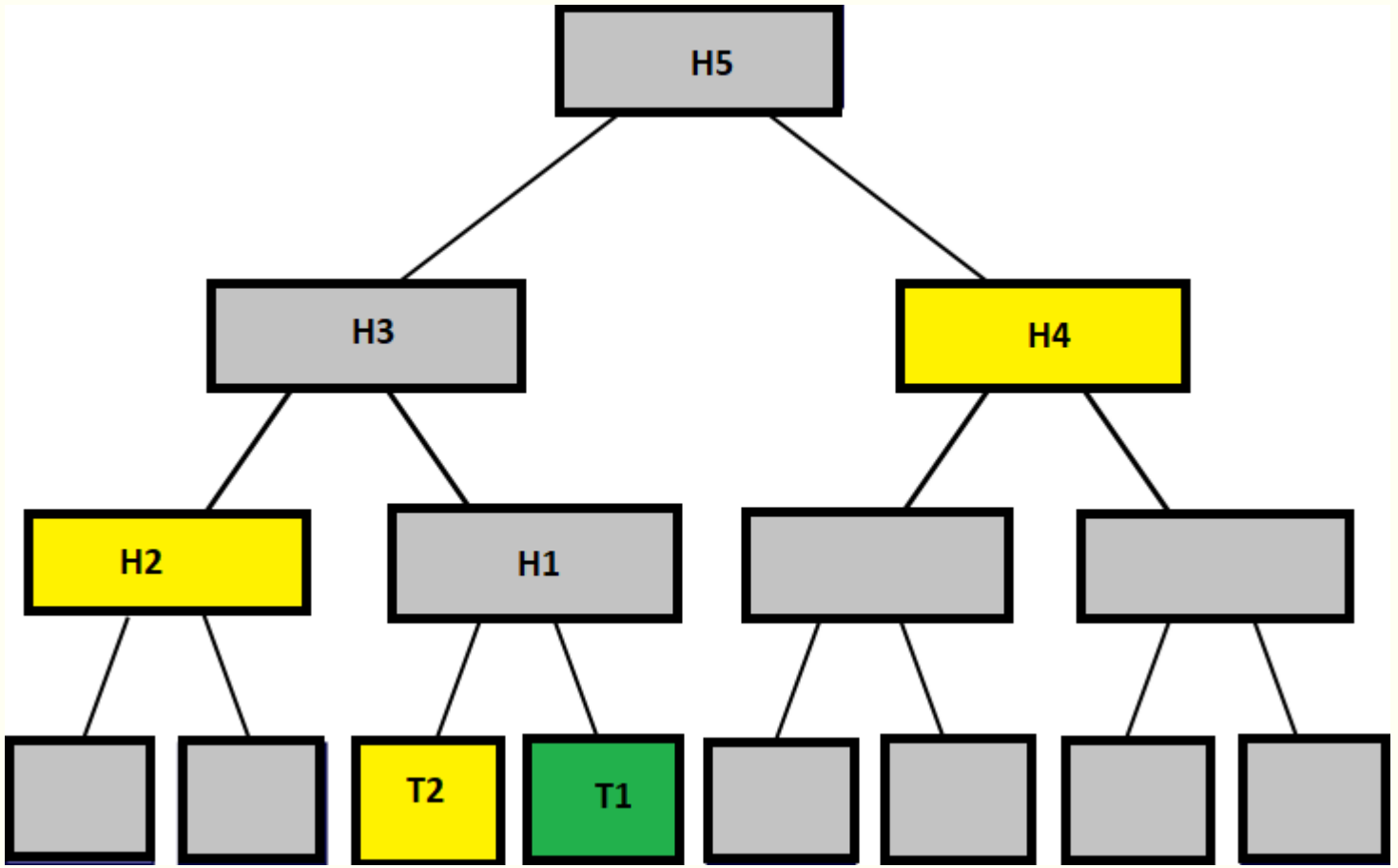
- Previous block hash
- Timestamp
- Mining difficulty
- PoW None
- Root hash for Merkle tree of transactions



MERKLE PROOFS BITCOIN -- LightClients

Light clients can be used to determine the status of a transaction:

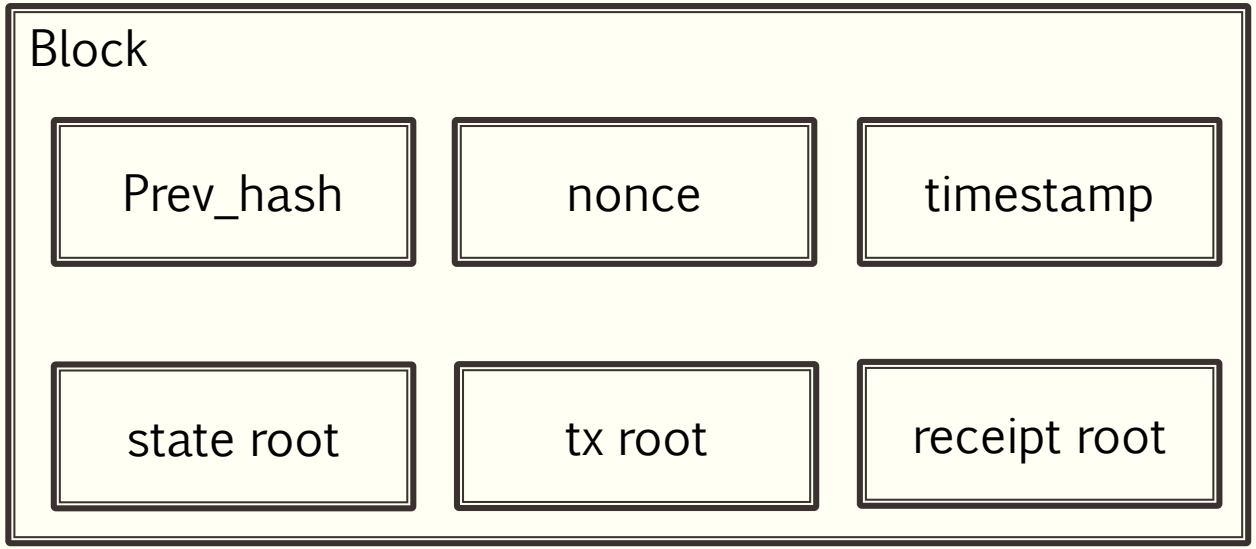
- Ask for a Merkle proof showing that a transaction is in one of the Merkle trees.
- A Merkle proof consists of a chunk, the root hash of the Merkle tree, and the "branch" consisting of all of the hashes on the path from the chunk to the root.
- Bitcoin proof not enough to know current state (balance, asset current holder etc.). One must authenticate all transactions.



MERKLE PROOFS ETHEREUM -- LightClients

- Is the transaction included in a block (tx root)?
- Does account X exists?
- What is the current balance of the account X?
- All events of type X?
- What will be the outcome of a certain transaction, effect on balances?

- Transaction tree immutable
- State-tree updates: new accounts, update balance, update storage for smart contracts etc.



TRANSACTION STRUCTURE

ETHEREUM TRANSACTION STRUCTURE

1. **transactionHash**: the hash of the transaction.
2. **type**: Ethereum delimits transactions for creating a new contract (0x0) and any other type of transaction (0x2).
3. **chainId**: indicates the network, 1 from mainnet, 1337 for private chains etc.
4. **blockHash**: the hash of the block the transaction is contained in.
5. **blockNumber**: the number of the block the transaction is contained in.
6. **transactionIndex**: the index of the transaction within the block.

ETHEREUM TRANSACTION STRUCTURE

7. **from**: the account that sends the transaction.
8. **to**: the receiver of the transaction.
9. **value**: the amount of ether being send.
10. **nonce**: the number of the transactions initiated by sender.
11. **(r, s, v)**: the signature (ECDSA) of the creator of the transaction.
12. **gas**: units of gas used by the transaction.
13. **gasPrice**: amount payed per unit of gas.
14. **maxFeePerGas**: Maximum amount of wei per gas the creator is willing to pay.
15. **maxPriorityFeePerGas**: Maximim amount of wei per gas the creator is willing to pay as tip to the miner.

SMART CONTRACT CREATION

/

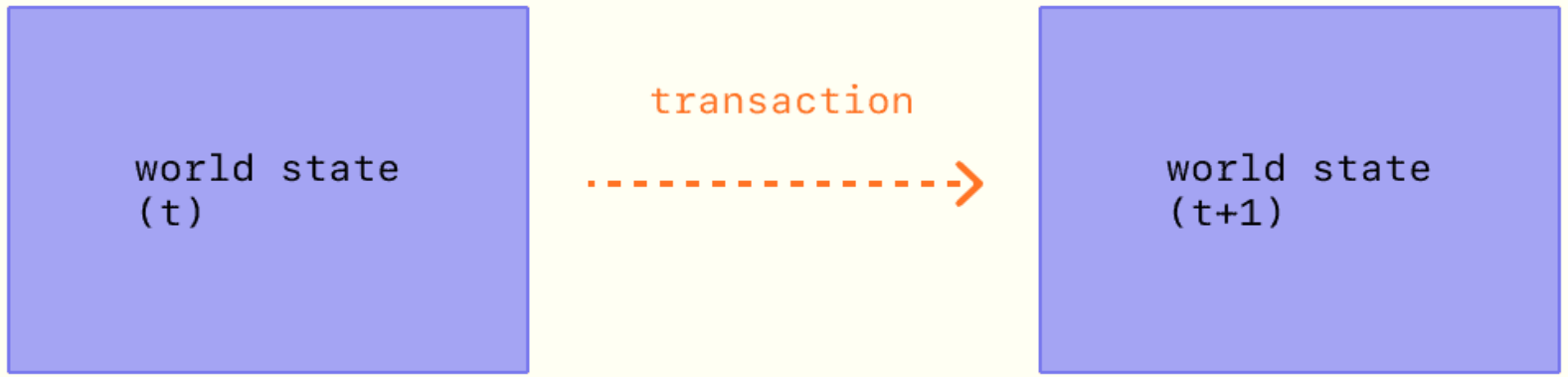
TRANSACTIONS

data: unlimited size byte array EVM code/input data.

ETHEREUM TRANSACTIONS AND GAS

ETHEREUM TRANSACTION

- A transaction is initiated by an externally-owned account
- The result of the transaction is changing word state.
- Lifecycle:
  - Generate transaction hash
  - Broadcast to the network, include transaction in transaction pool
  - Miner include transaction in a block
  - Transaction receive confirmations



EVM - GAS

- Measures the amount of computational effort required to execute operations.
  - “Fuel” for Ethereum network – not currency!
- Each operation costs a specific amount of gas.
- For each transaction sender specify a “gas limit”, maximum amount he is willing to pay for the transaction to be processed.
- Gas limit is implicitly purchased from the sender’s account balance.
- The transaction is considered invalid if the account balance cannot support paying gas limit.
- Gas can be partially return to the sender.
- If transaction runs out of gas, then it’s reverted back to its original state.
- “Block gas limit” determines that amount of gas that can be spent per block.

ETHEREUM TRANSACTION

- Transaction validity

- Transaction is well formed RLP, no additional bytes.
- Transaction signature is valid.
- Transaction nonce is valid, equivalent to the sender’s current nonce.
- Gas limit is smaller than gas used by the transaction.
- The sender account balance contains at least the cost required to pay transaction.

- Gas limit * gas cost
- Gas limit * Base fee + tip

EVM - GAS EIP 1559

- Before EIP 1559 Gas cost set by **first price auction system**
- Everyone submit bid, miners select transactions with the highest fee.
- Often users pay more than 5x than necessary.

- Before EIP 1559 block size had fixed-size
- After EIP 1559 block size is variable and will increase or decrease depending on the network demand up until the block limit (30 million gas)

- After EIP 1559 block has a **base fee** determined by the blocks before. If block size is greater than block limit increase fee, if is less then the block size, decrease fee.

EVM – GAS EIP 1559

- **Ethereum Improvement Proposals** (EIP) describe standards for the Ethereum platform
  - Core improvements requiring consensus fork, miner startegy changes.
  - Networking
  - ERC – contract standards (ERC-20, ERC-721) Ethereum Request for comment.
- (London Upgrade) August 5th, 2021
  - Better transaction fee estimation, predictable transaction fees
  - Quicker transaction inclusion
  - counteract the release of ETH by burning a percentage of transaction fee.
- Replace first auction system with **base fee**.
- **maxPriorityFeePerGas** amount of gas paid as tip

Bibliography

- Ethereum yellow paper https://ethereum.github.io/yellowpaper/paper.pdf
- Merkling in Ethereum, Vitalik Buterin https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/
- Ethereum Patricia-tree https://eth.wiki/en/fundamentals/patricia-tree
- HP implementation and examples https://hexdocs.pm/hex_prefix/HexPrefix.html
- Ethereum EVM https://ethereum.org/en/developers/docs/evm/
- EIP standards https://eips.ethereum.org/
- EVM memory https://docs.soliditylang.org/en/latest/introduction-to-smart-contracts.html?highlight=memory#storage-memory-and-the-stack

- https://eth.wiki/en/concepts/ethash/ethash
- https://commons.wikimedia.org/wiki/File:Hash_Tree.svg
- https://commons.wikimedia.org/wiki/File:Bitcoin_Block_Data.png