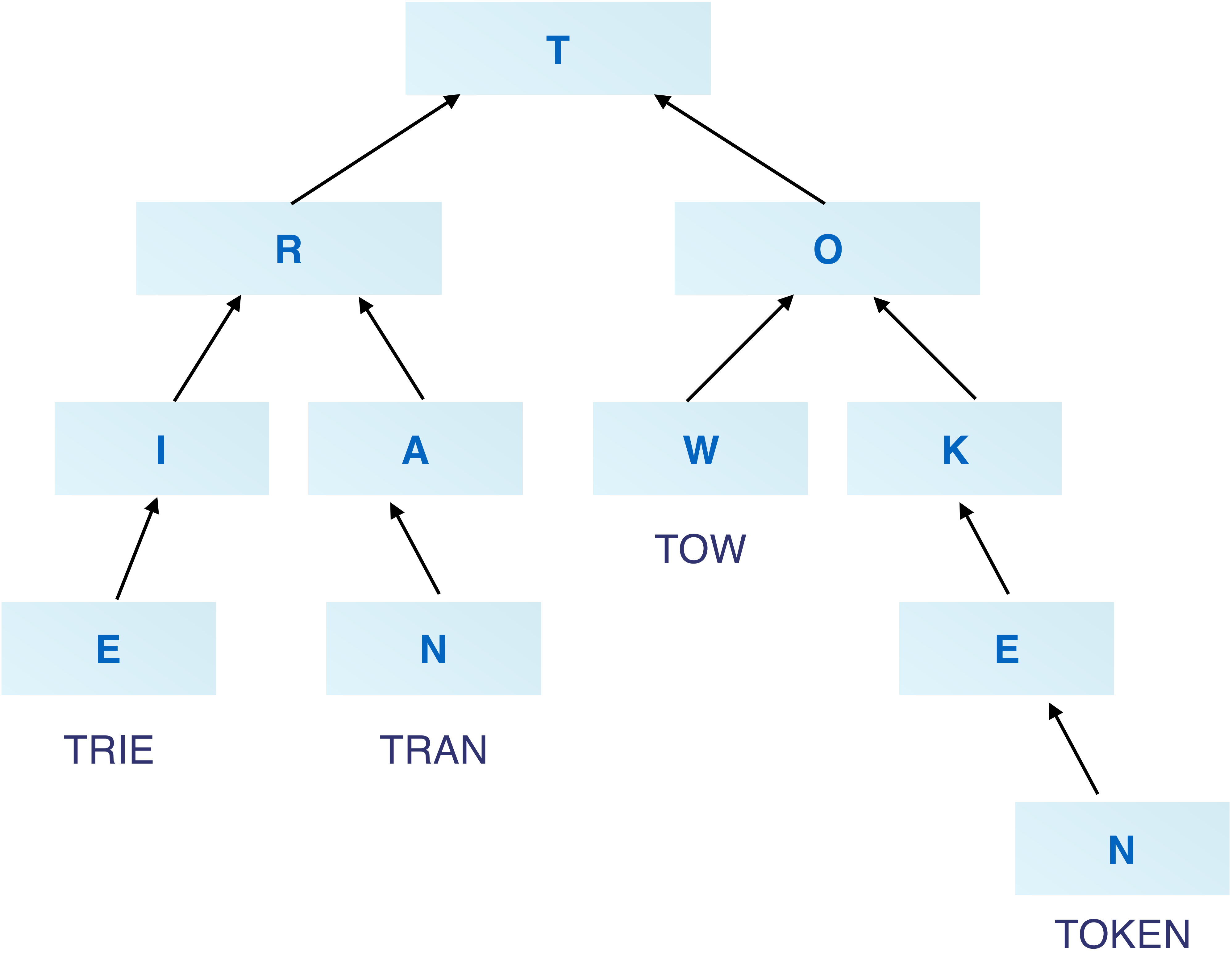
ETHEREUM MINI

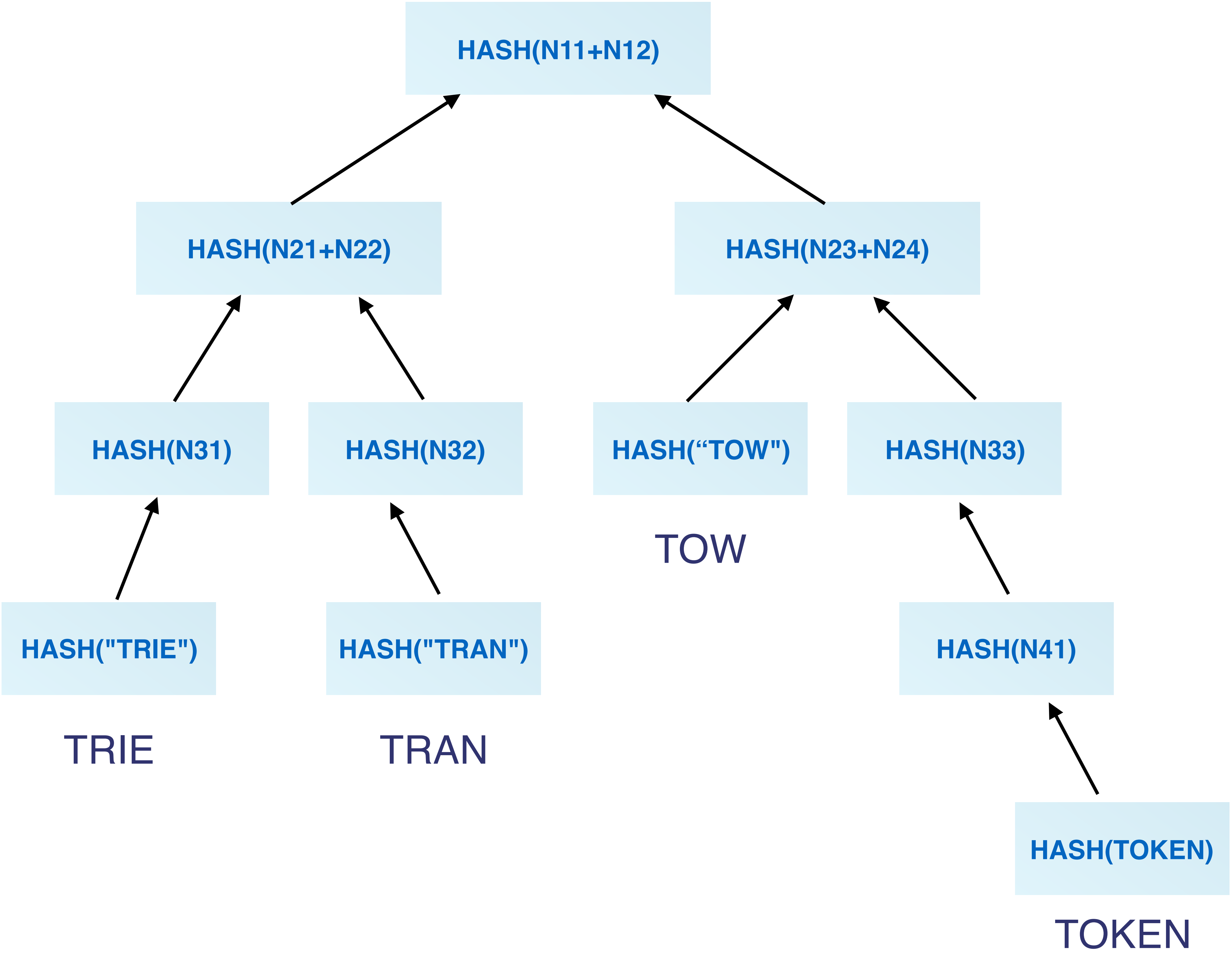
Data structures

MERKLE PATRICIA TRIE

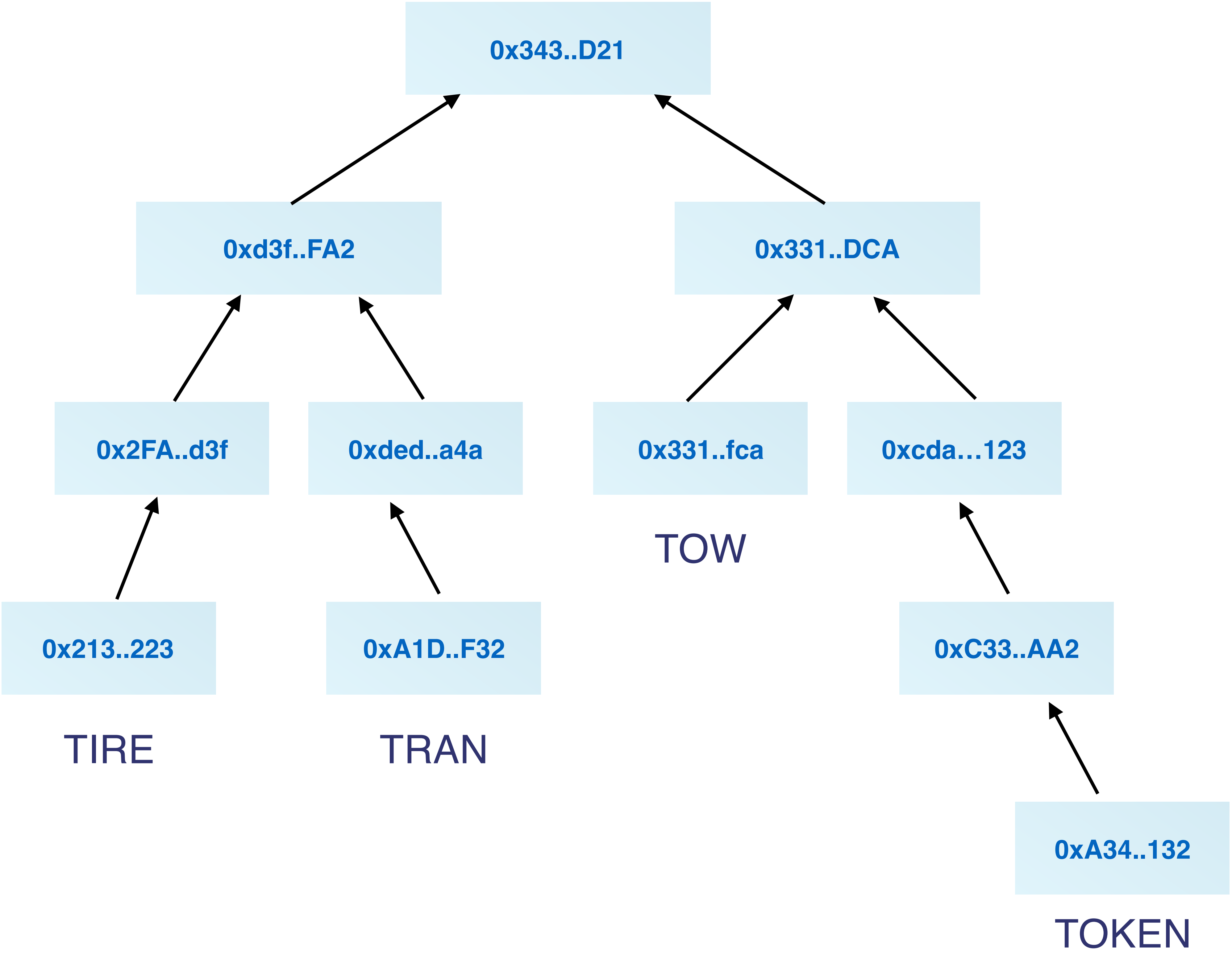
Patricia tree (Radix trie)



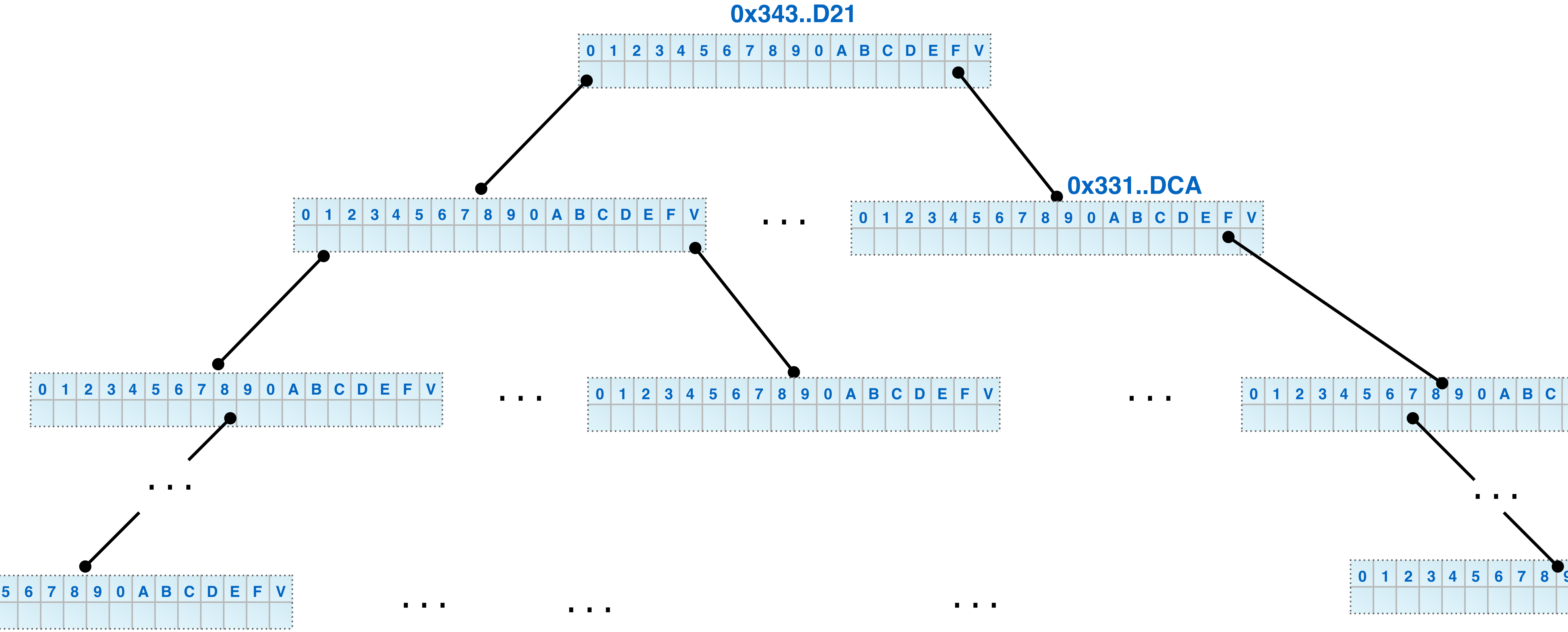
Merkle Tree


HASH(N11+N12)
HASH(N21+N22)
HASH(N23+N24)
HASH(N31)
HASH(N32)
HASH(“TOW")
HASH(N33)
TOW
HASH("TRIE")
HASH("TRAN")
HASH(N41)
TRIE
TRAN
HASH(TOKEN)
TOKEN

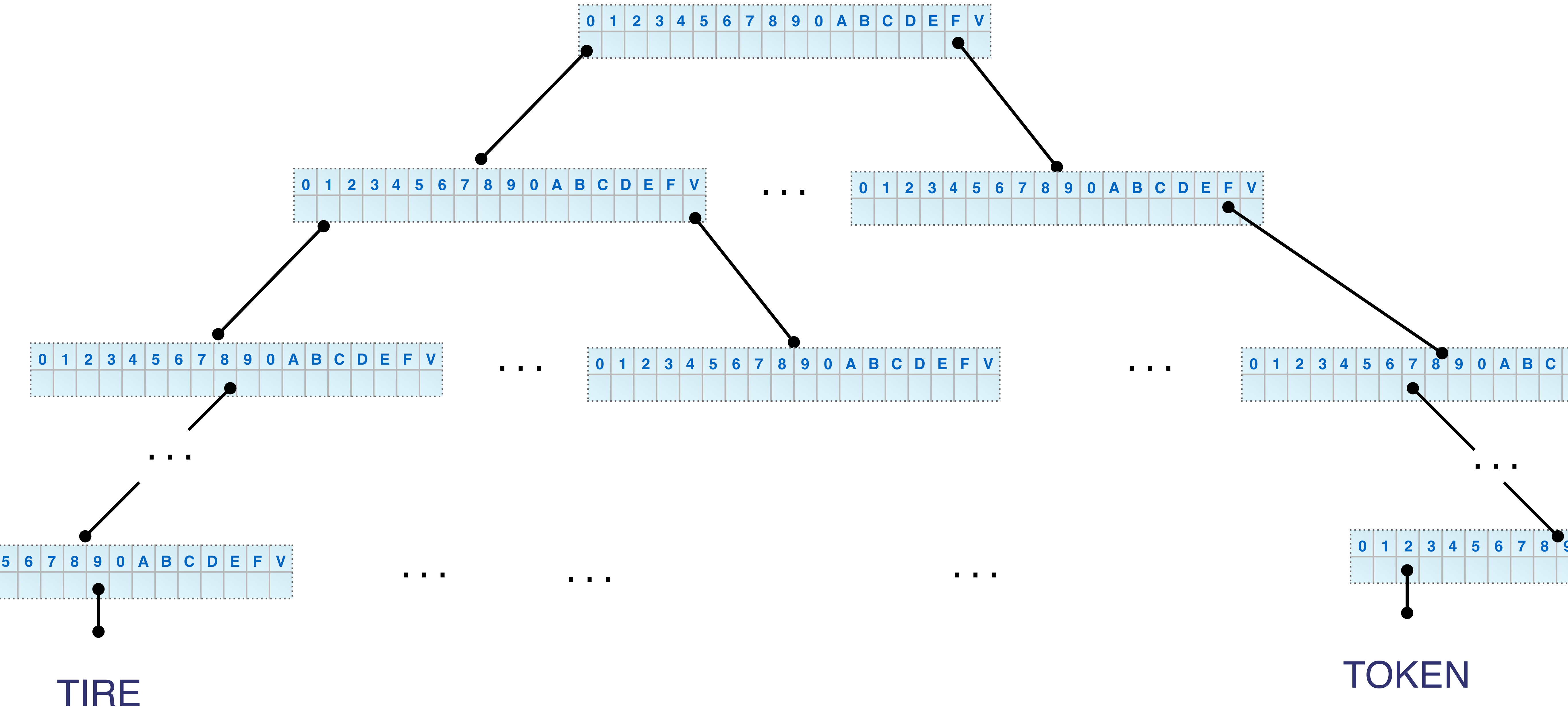
Merkle Tree


0x343..D21
0xd3f..FA2
0x331..DCA
0x2FA..d3f
0xded..a4a
0x331..fca
0xcda…123
TOW
0x213..223
0xA1D..F32
0xC33..AA2
TIRE
TRAN
0xA34..132
TOKEN

Ethereum Merkle Patricia Trie


0x343..D21
0 1 2 3 4 5 6 7 8 9 0 A B C D E F V
0 1 2 3 4 5 6 7 8 9 0 A B C D E F V
. . .
0x331..DCA
0 1 2 3 4 5 6 7 8 9 0 A B C D E F V
0 1 2 3 4 5 6 7 8 9 0 A B C D E F V
. . .
0 1 2 3 4 5 6 7 8 9 0 A B C D E F V
. . .
0 1 2 3 4 5 6 7 8 9 0 A B C
. . .
. . .
5 6 7 8 9 0 A B C D E F V
. . .
. . .
. . .
0 1 2 3 4 5 6 7 8 9

Merkle Patricia Trie


0 1 2 3 4 5 6 7 8 9 0 A B C D E F V
0 1 2 3 4 5 6 7 8 9 0 A B C D E F V
. . .
0 1 2 3 4 5 6 7 8 9 0 A B C D E F V
0 1 2 3 4 5 6 7 8 9 0 A B C D E F V
. . .
0 1 2 3 4 5 6 7 8 9 0 A B C D E F V
. . .
0 1 2 3 4 5 6 7 8 9 0 A B C
. . .
. . .
5 6 7 8 9 0 A B C D E F V
. . .
. . .
. . .
0 1 2 3 4 5 6 7 8 9
TIRE
TOKEN

Ethereum Merkle Patricia Trie

Node:

- NULL (represented as the empty string)
- branch A 17-item node [v0 ... v15, vt]
- leaf A 2-item node [encodedPath, value]
- extension A 2-item node [encodedPath, key]

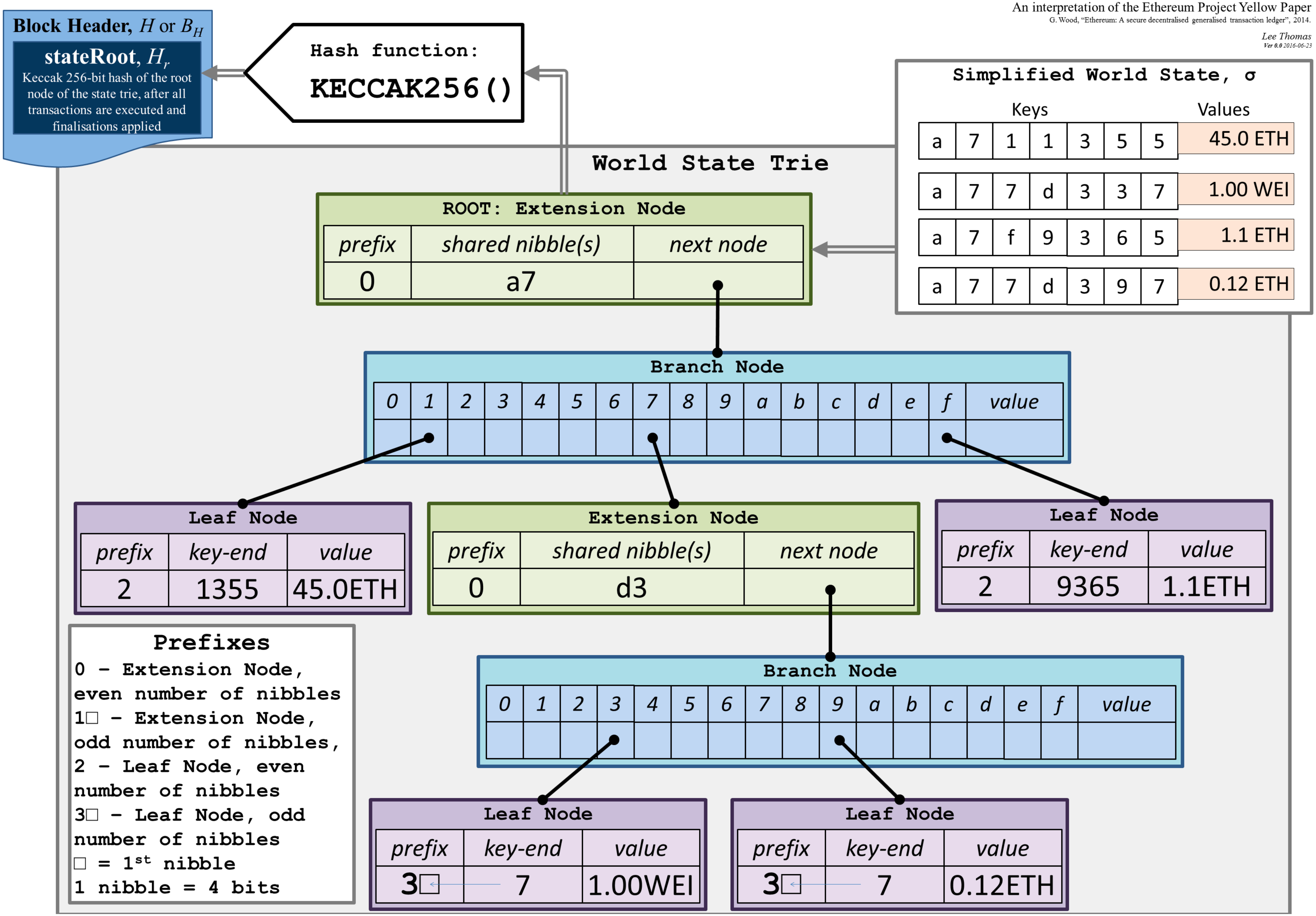
Ethereum Modified Merkle-Paricia-Trie System

An interpretation of the Ethereum Project Yellow Paper

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

Lee Thomas

Ver 0.0 2016-06-23



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

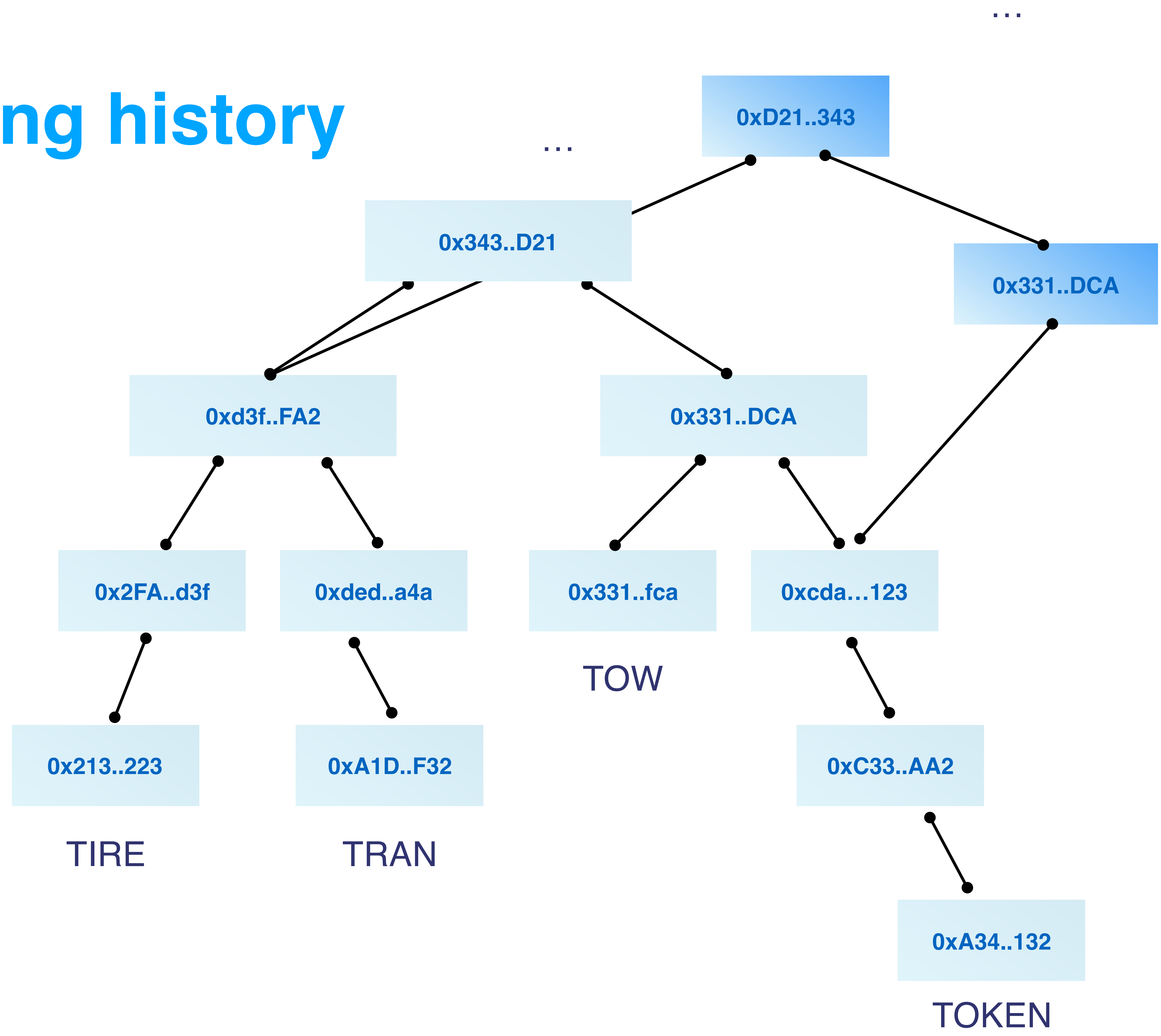
prefix	*key-end*	*value*
3□	7	1.00WEI

Leaf Node

prefix	*key-end*	*value*
3□	7	0.12ETH

Source: https://ethereum.stackexchange.com/questions/12941/in-the-ethereum-modified-merkle-particia-tree-what-do-the-prefix-key-and-value

Storing history



Ethereum block header tries

- State Trie

  Path: ethereumAddress (sha3)

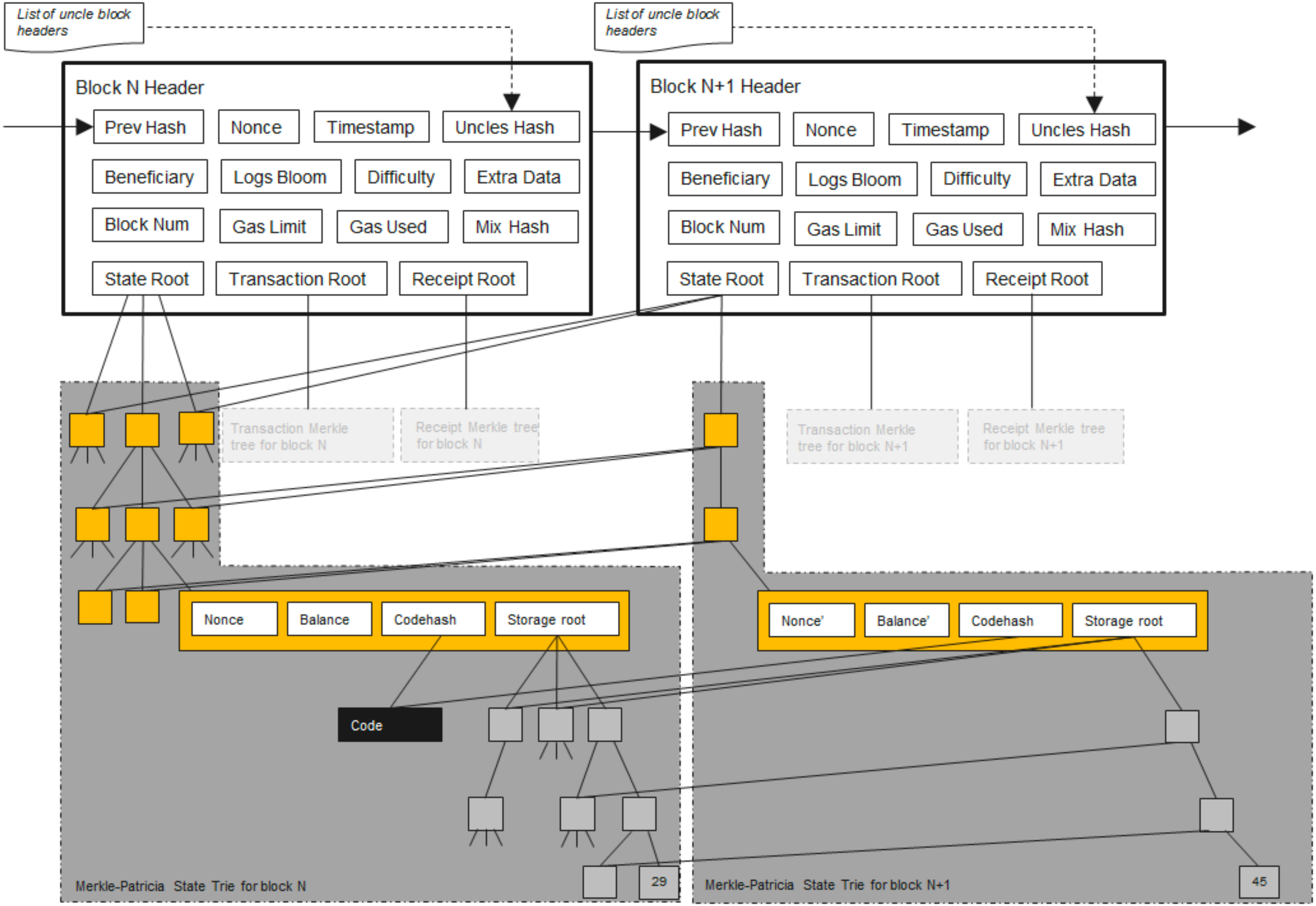
  Value: [nonce, balance, storageRoot, codeHash] (rlp)

- Transactions Trie

  path: transactionIndex - within the block (rlp)

- Receipts Trie

  path: transactionIndex - within the block (rlp)



Source: https://ethereum.stackexchange.com/questions/359/where-is-the-state-data-stored

Storage Trie

- Part of account
- Where all contract data lives
- Separate storage trie for each account

Ethereum node types

- Full - (geth) node processes the entire blockchain and replays all transactions that ever happened
- Prune - store just most recent blocks (parity - 1024)

  NOTE: You can have full - prune node
- Light - use external node

Read more: https://dev.to/5chdn/the-ethereum-blockchain-size-will-not-exceed-1tb-anytime-soon-58a

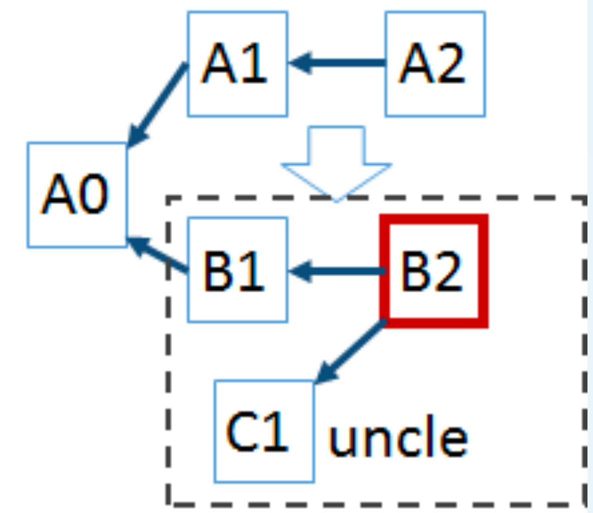
Block

- block.coinbase (address): current block miner’s address
- block.difficulty (uint): current block difficulty
- block.gaslimit (uint): current block gaslimit
- block.number (uint): current block number
- block.timestamp (uint): current block timestamp as seconds since unix epoch
- **block.blockhash**(uint blockNumber) returns (bytes32): hash of the given block

  Note: only works for 256 most recent blocks excluding current

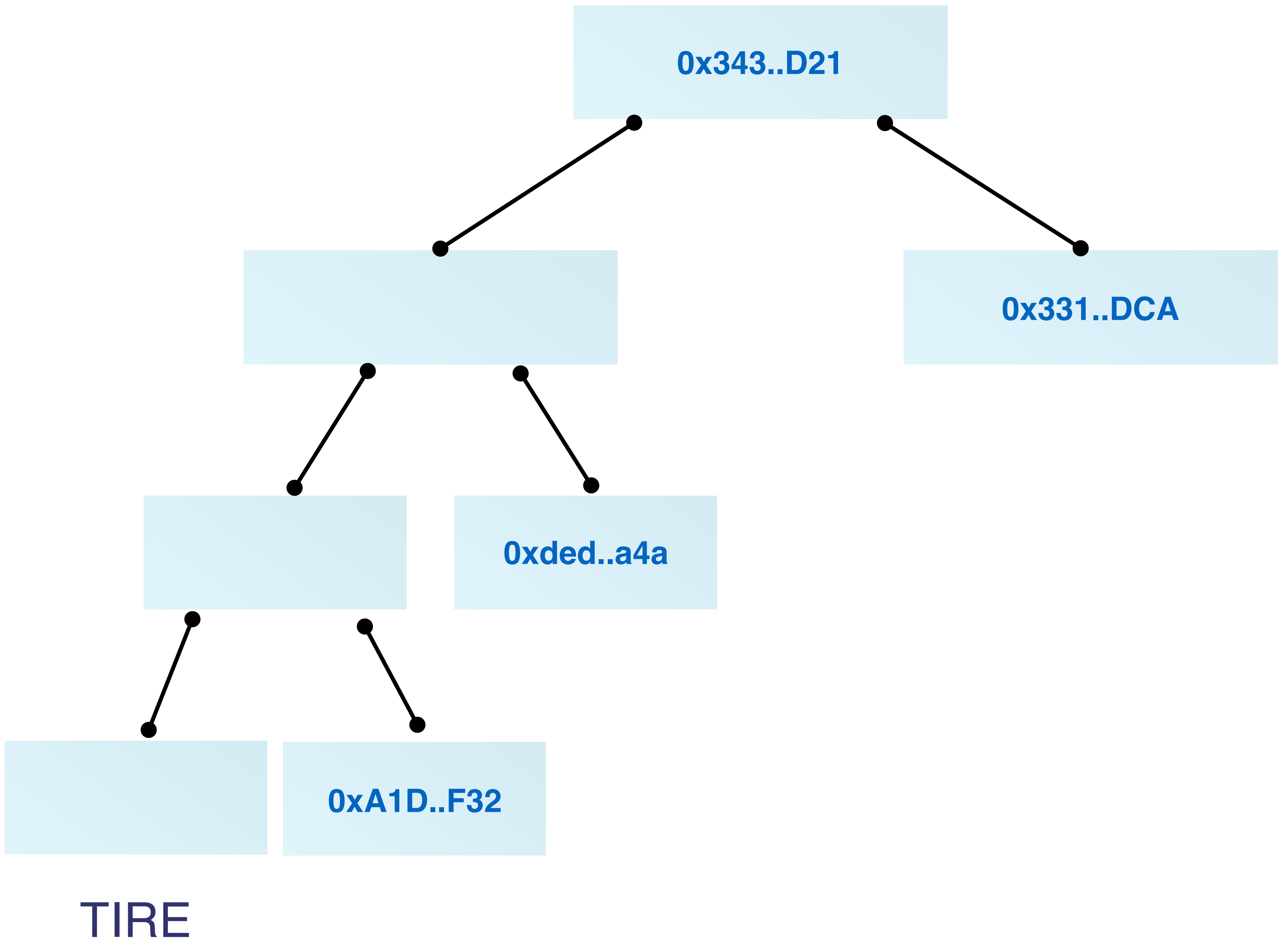
GHOST protocol

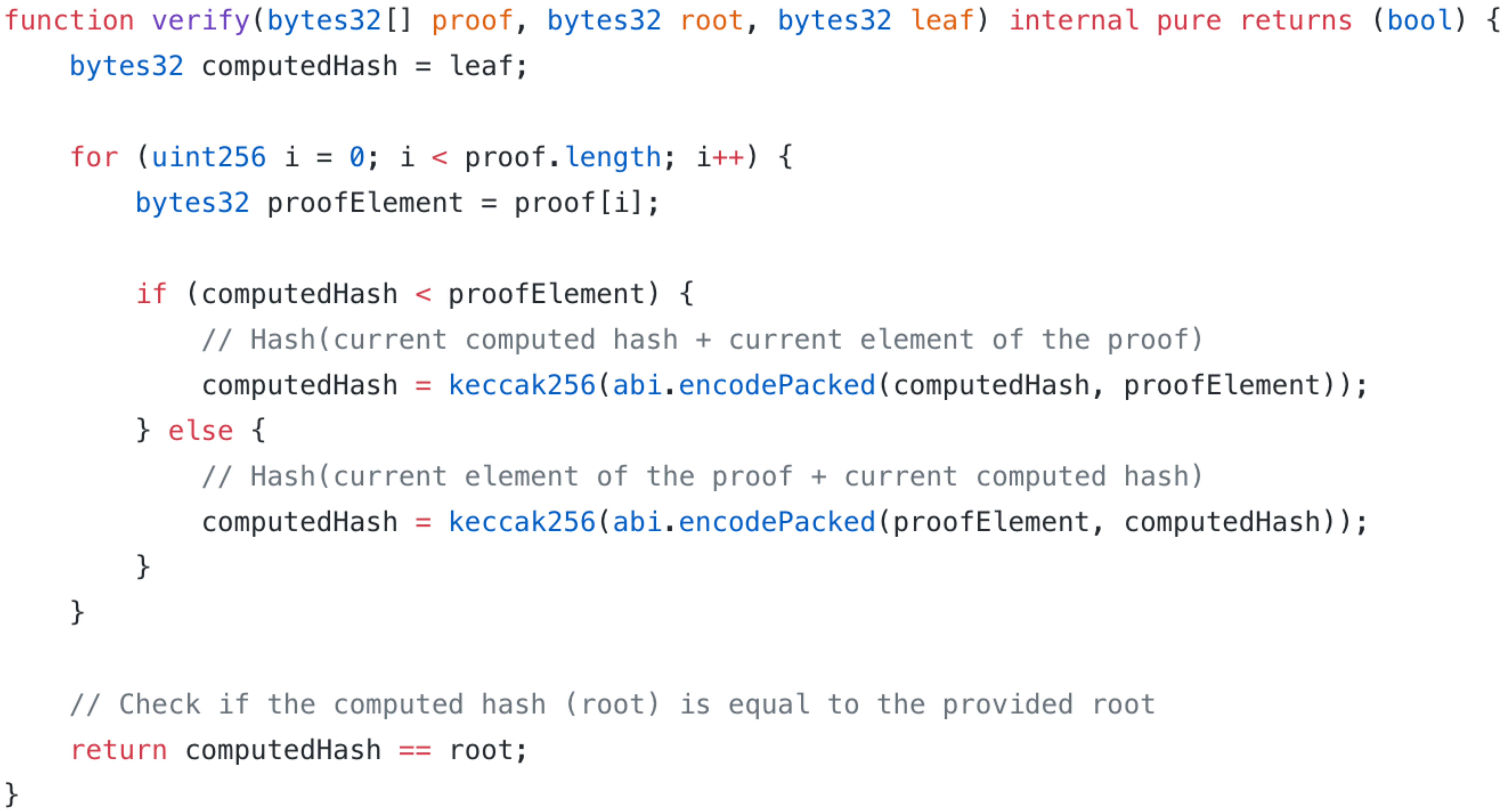
- Greedy Heaviest Observed Subtree
- Enables faster block time
- Reduce the incentive for pooled mining
- Miners includes stale blocks as uncles



Merkle Proof

```
function verify(bytes32[] proof, bytes32 root, bytes32 leaf) internal pure returns
(bool)
```



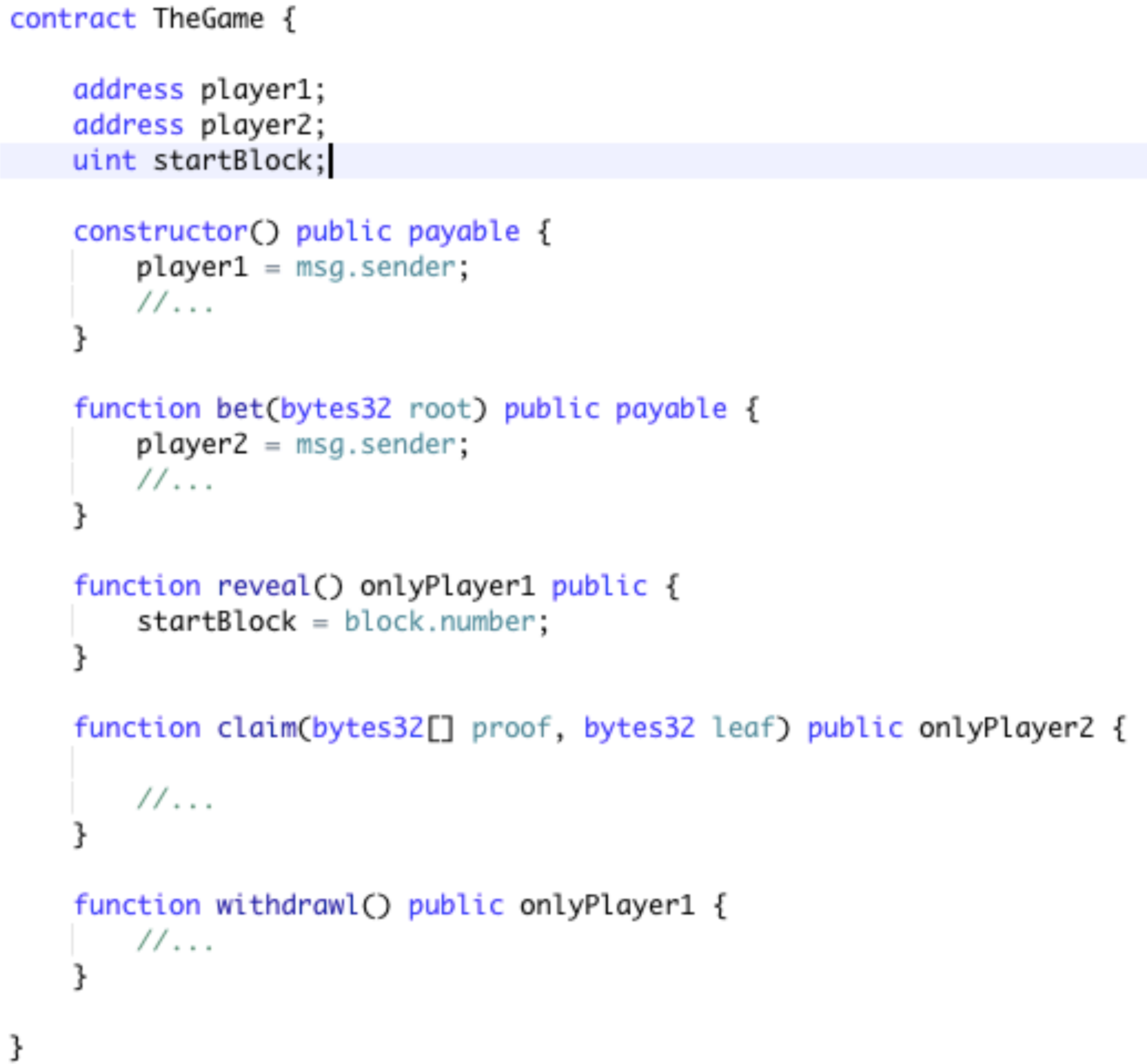


```
function verify(bytes32[] proof, bytes32 root, bytes32 leaf) internal pure returns (bool) {
    bytes32 computedHash = leaf;

    for (uint256 i = 0; i < proof.length; i++) {
        bytes32 proofElement = proof[i];

        if (computedHash < proofElement) {
            // Hash(current computed hash + current element of the proof)
            computedHash = keccak256(abi.encodePacked(computedHash, proofElement));
        } else {
            // Hash(current element of the proof + current computed hash)
            computedHash = keccak256(abi.encodePacked(proofElement, computedHash));
        }
    }

    // Check if the computed hash (root) is equal to the provided root
    return computedHash == root;
}
```

Task: The game

- **Player1**: Stakes 2 ether
- **Player2**: Provides Merkle root of the tree with 1024 numbers (between 0 and 2048).
- **Player1**: Provides a number (between 0 and 2048)
- **Player2**: Can withdraw if he can prove there is a tree supplied earlier (within 256 blocks)
- **Player 1**: After 256 blocks Player1 can withdraw (if Player2 didn’t)

- Contract + tests



```
contract TheGame {

    address player1;
    address player2;
    uint startBlock;

    constructor() public payable {
        player1 = msg.sender;
        //...
    }

    function bet(bytes32 root) public payable {
        player2 = msg.sender;
        //...
    }

    function reveal() onlyPlayer1 public {
        startBlock = block.number;
    }

    function claim(bytes32[] proof, bytes32 leaf) public onlyPlayer2 {

        //...
    }

    function withdrawl() public onlyPlayer1 {
        //...
    }

}
```

Marek Kirejczyk

Follow us on twitter
@ethworks

ETHWORKS