Merkle Trees

University of Athens
Dionysis Zindros, Christos Nasikas

Introduction

- A data structure: usually a binary tree
- Used for efficiently summarizing and verifying the integrity and validity of large sets of data
- Useful in peer-to-peer networks
  - Damaged / Altered blocks can be received and identified as such
  - Prevent malicious actors to send fake blocks
- Used in ipfs, git, BitTorrent, nosql databases, bitcoin, ethereum

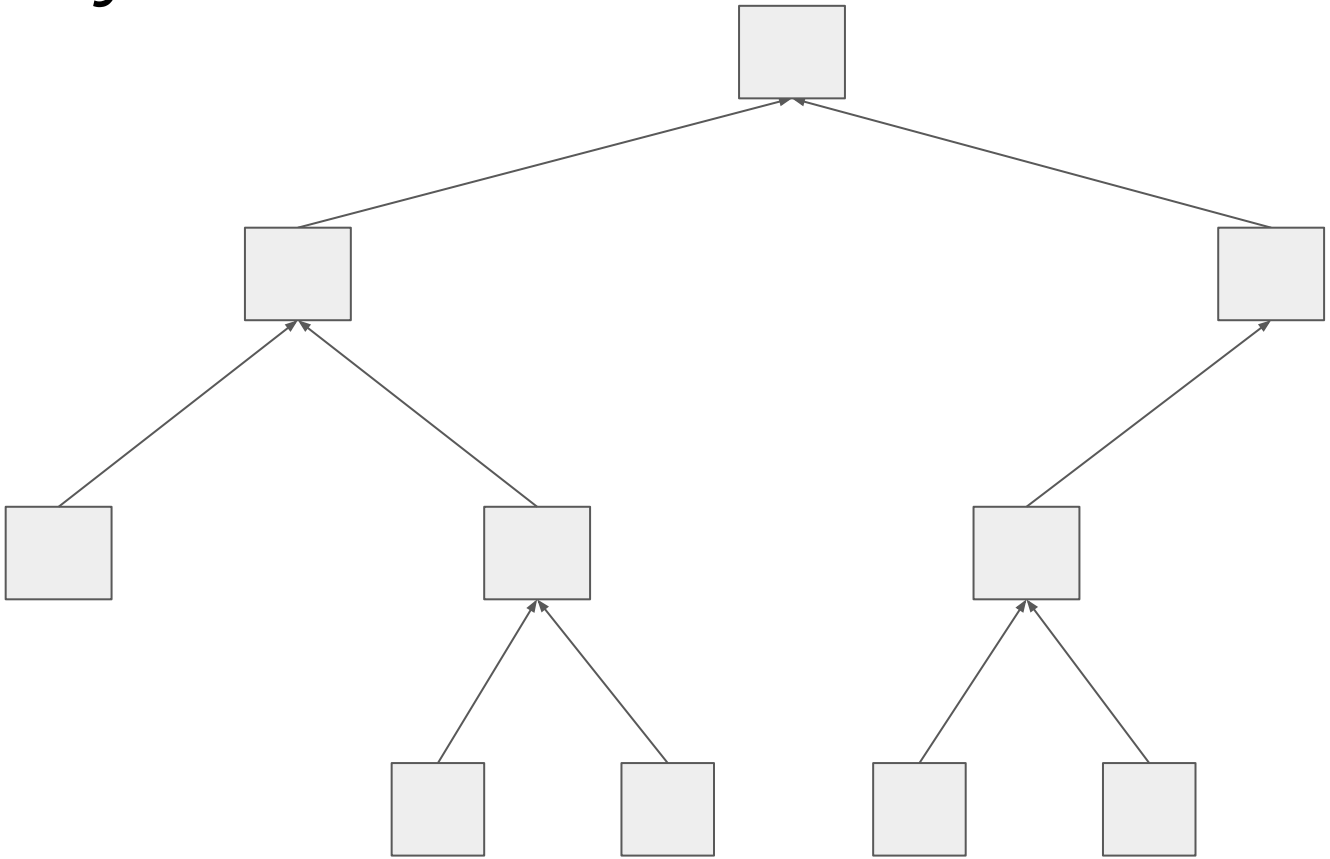
Introduction

- Proofs for data integrity and validity
- Proofs require little memory and space
- Proofs require tiny amounts of information to be transmitted across networks

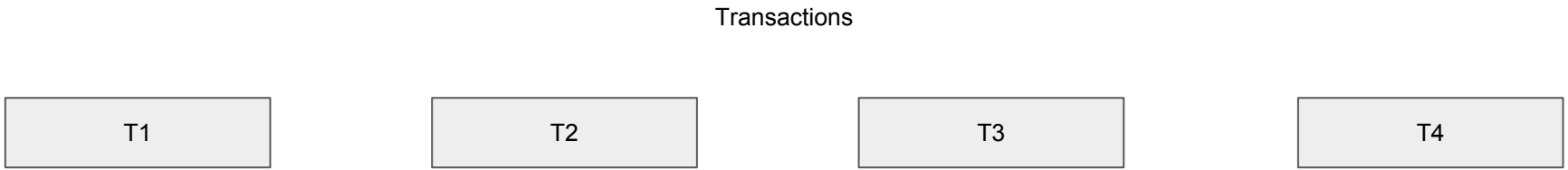
Introduction

- Used for keeping a summary of all the transactions in a block
- Digital fingerprint of the entire set of transactions
- Transaction haven't been altered or tampered within a block
- Fast proof of inclusion
- Extensively used by Simplified Payment Verification nodes

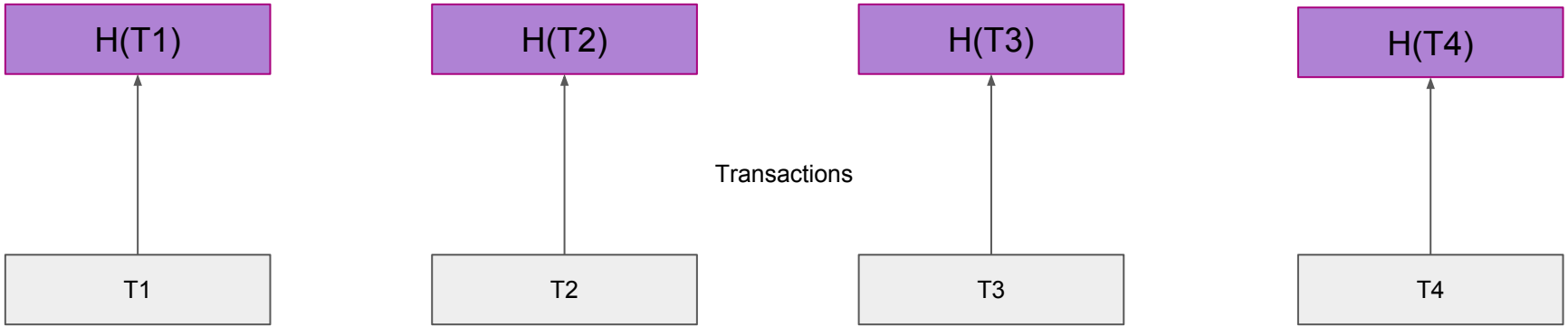
Binary tree



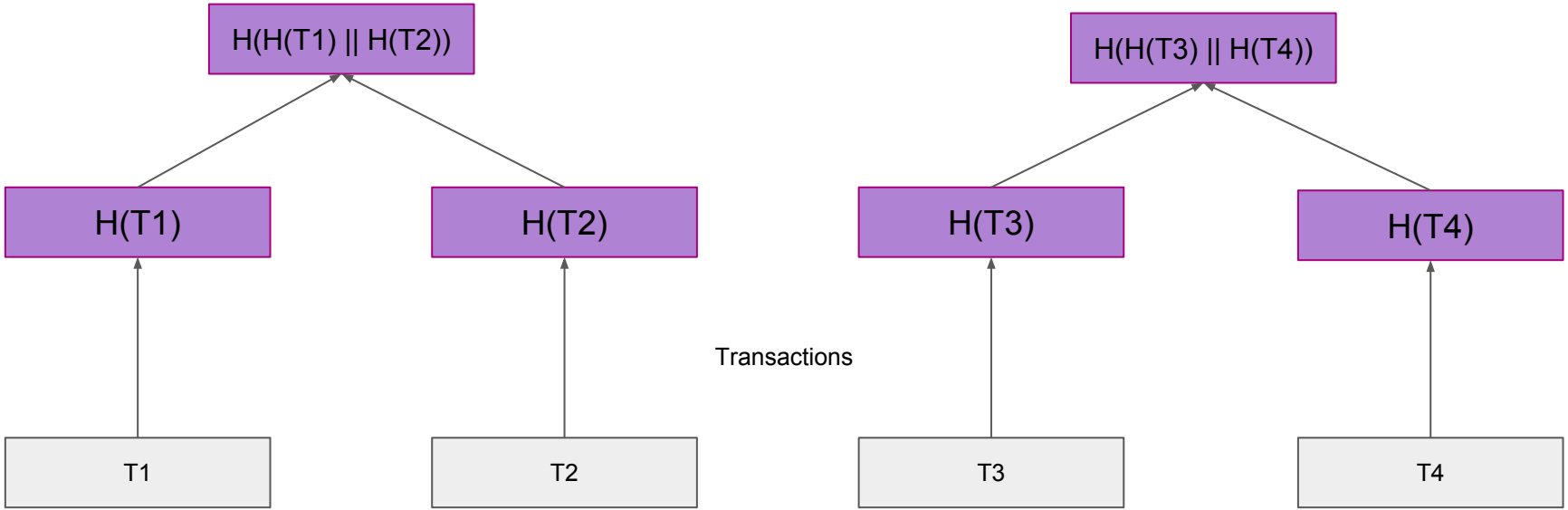
Merkle tree: construction



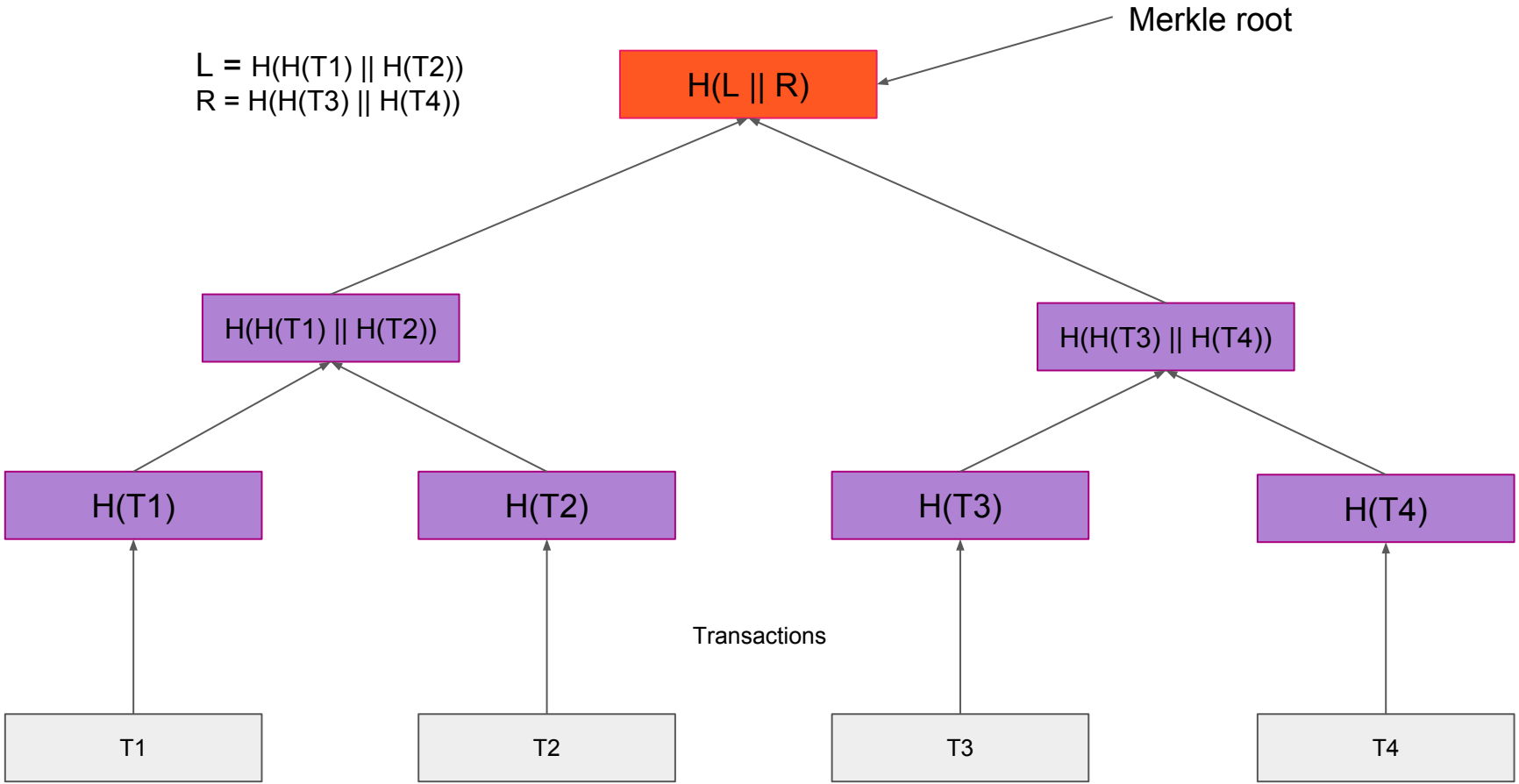
Merkle tree: construction



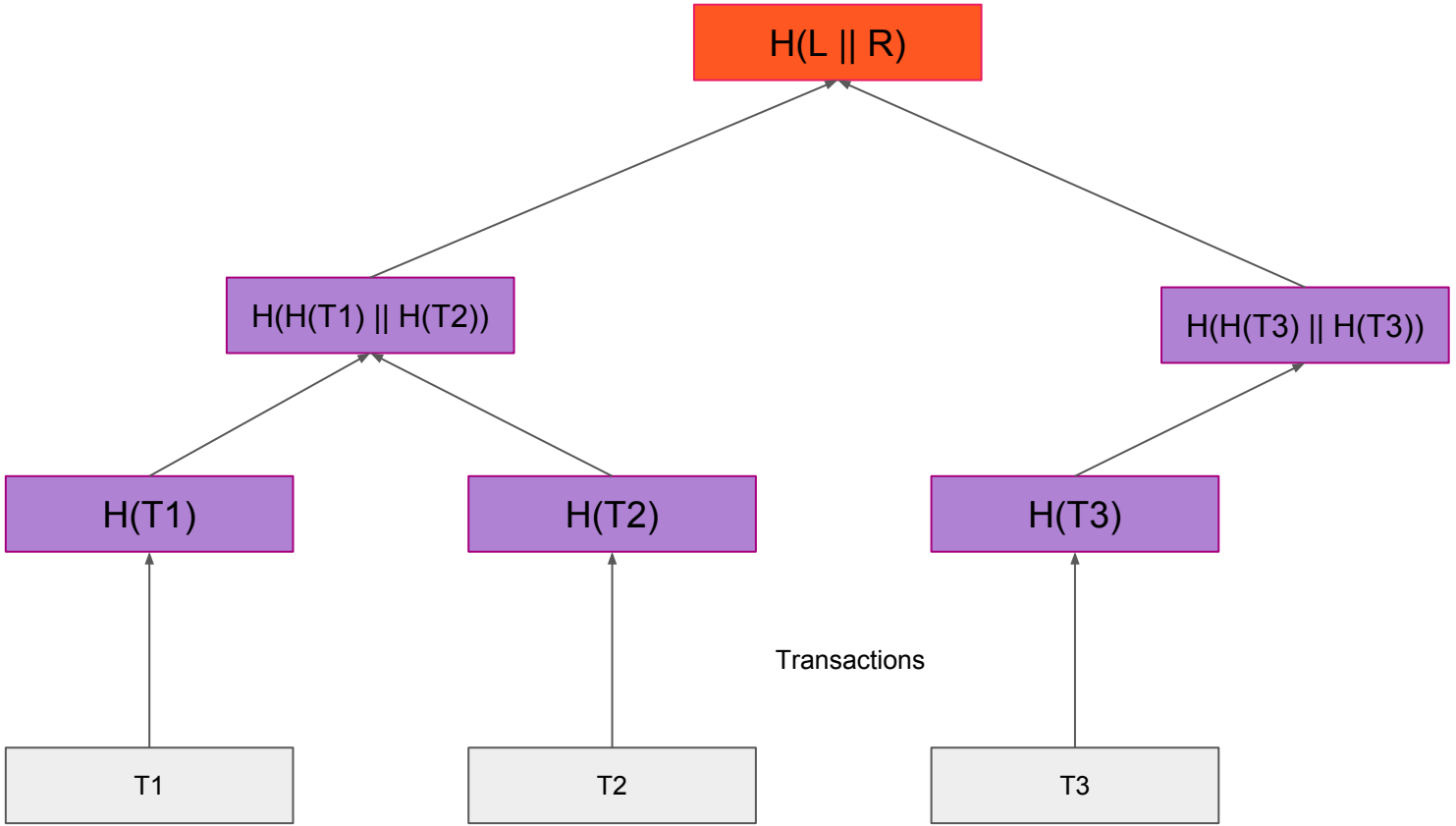
Merkle tree: construction



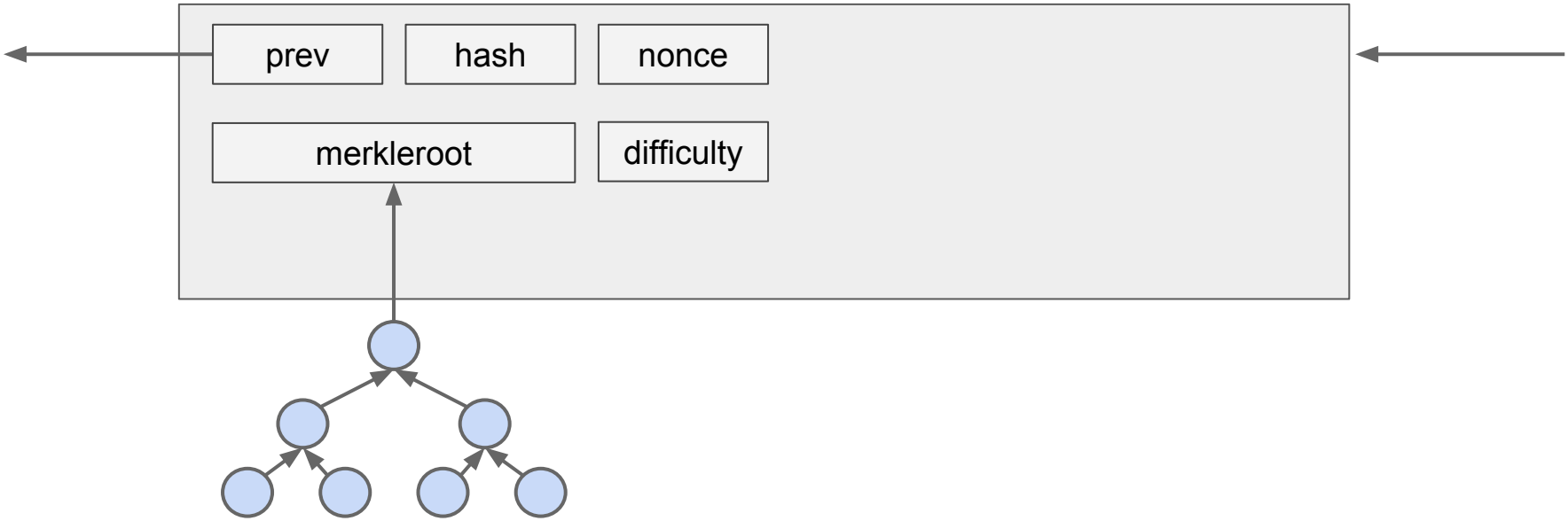
Merkle tree: construction



Merkle tree: construction



Merkle tree: bitcoin block



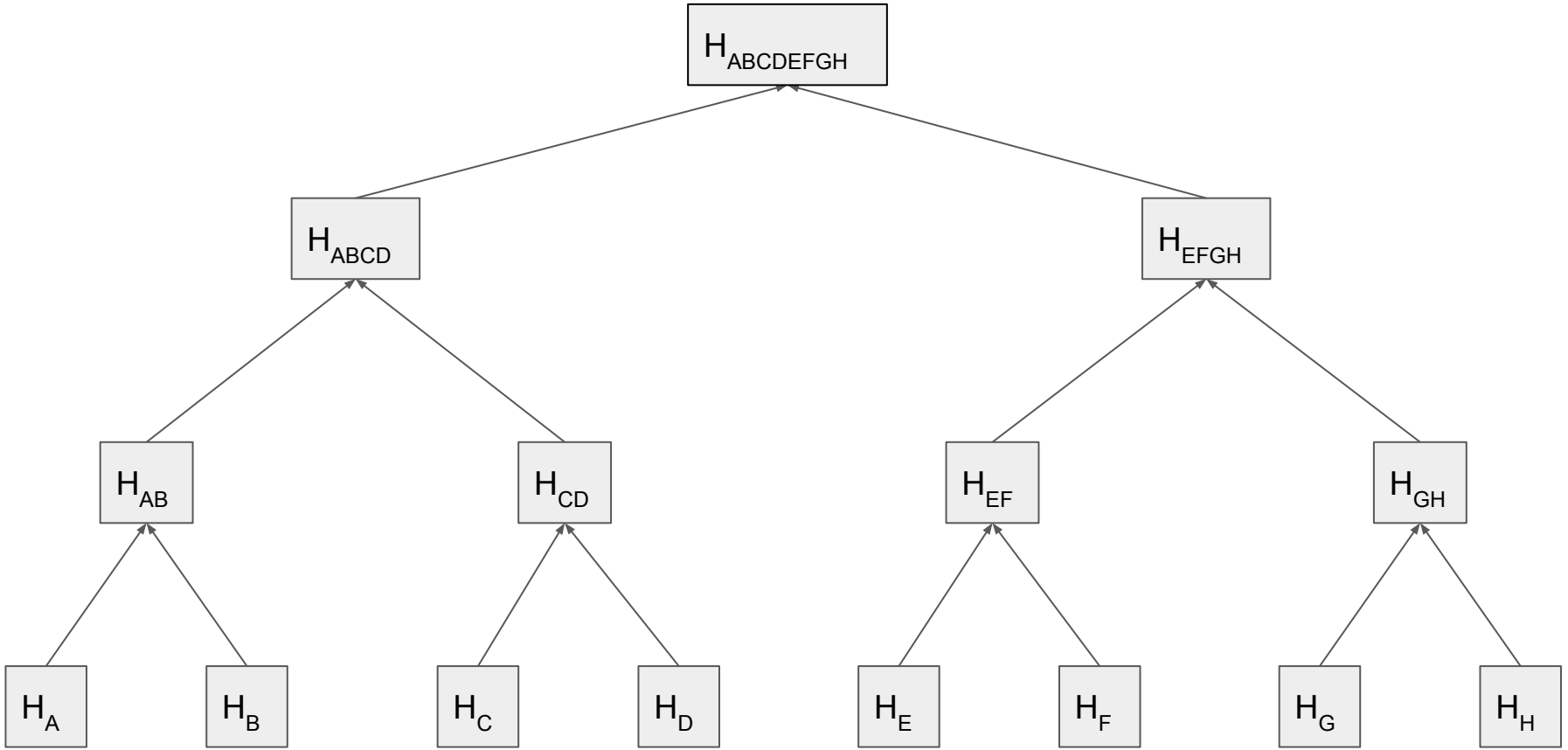
Merkle tree: proof of inclusion

- SPV nodes do not have data from all transactions in a block
- SPV nodes download only the block header
- It wants a proof that a transaction belongs to a certain block

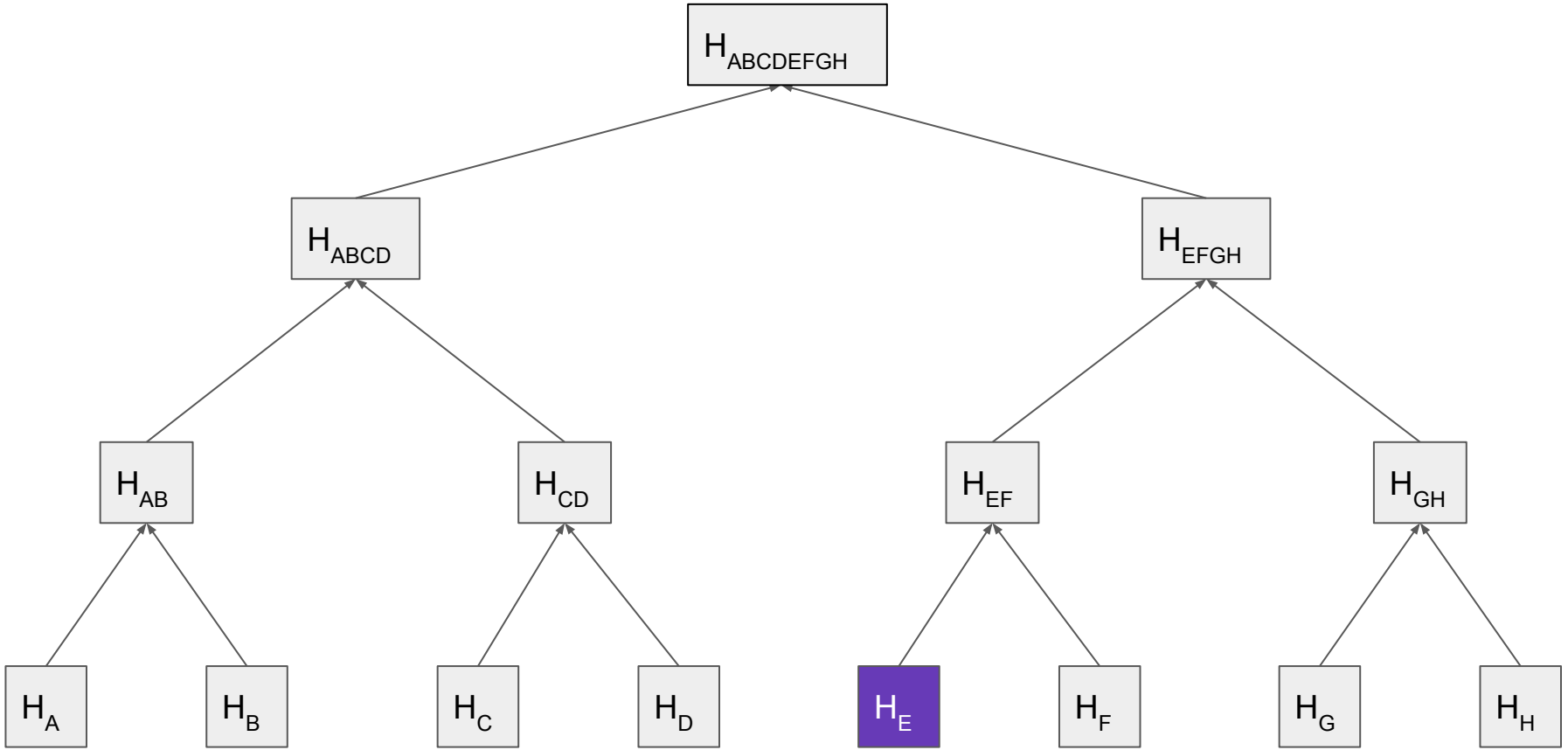
Merkle tree: proof of inclusion

- Merkle trees enable SPV nodes on the blockchain to check if miners have verified the transactions in a block without downloading all the transactions in a block
- Can verify parts of blocks individually and can check individual transactions using hashes of other branches of the tree

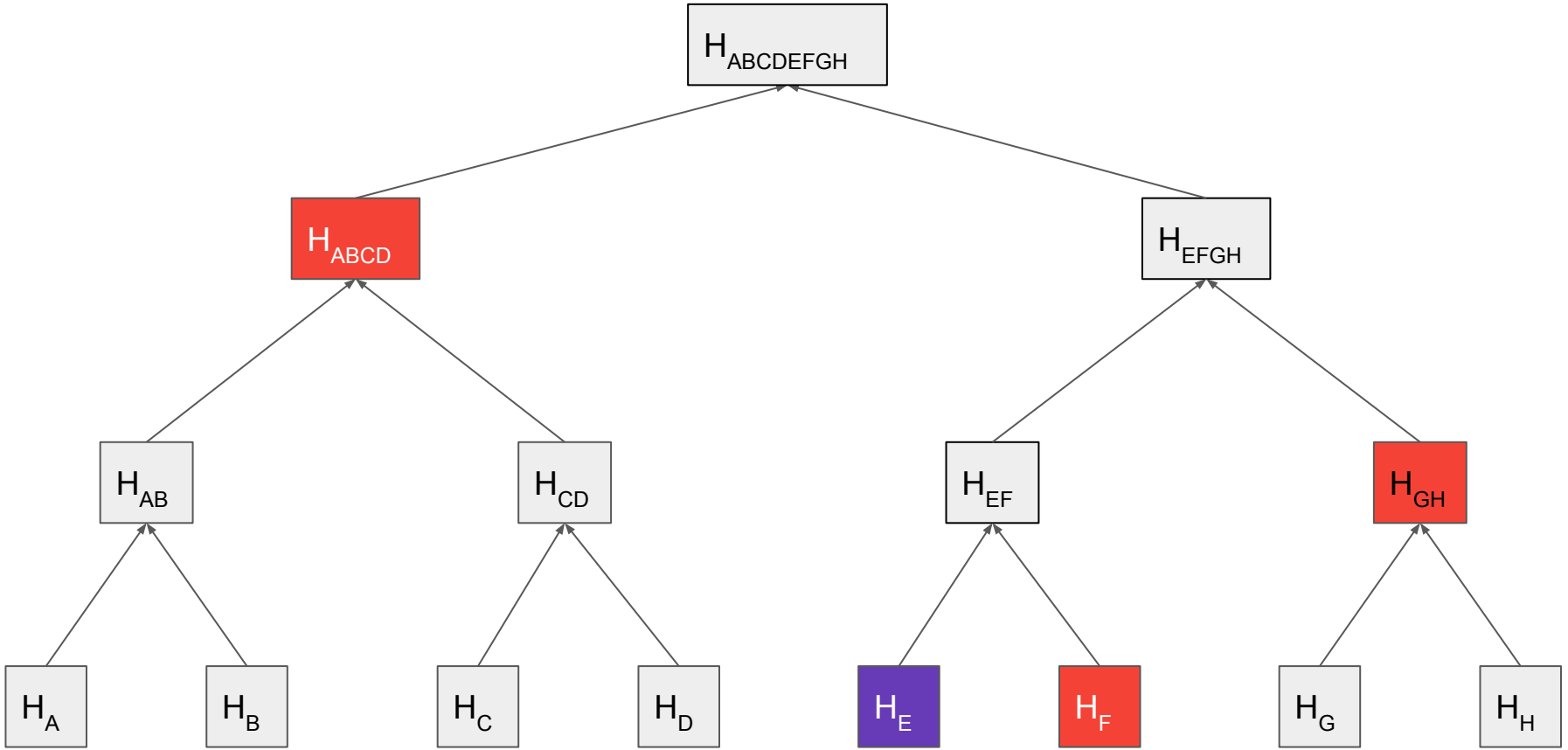
Merkle tree: proof of inclusion



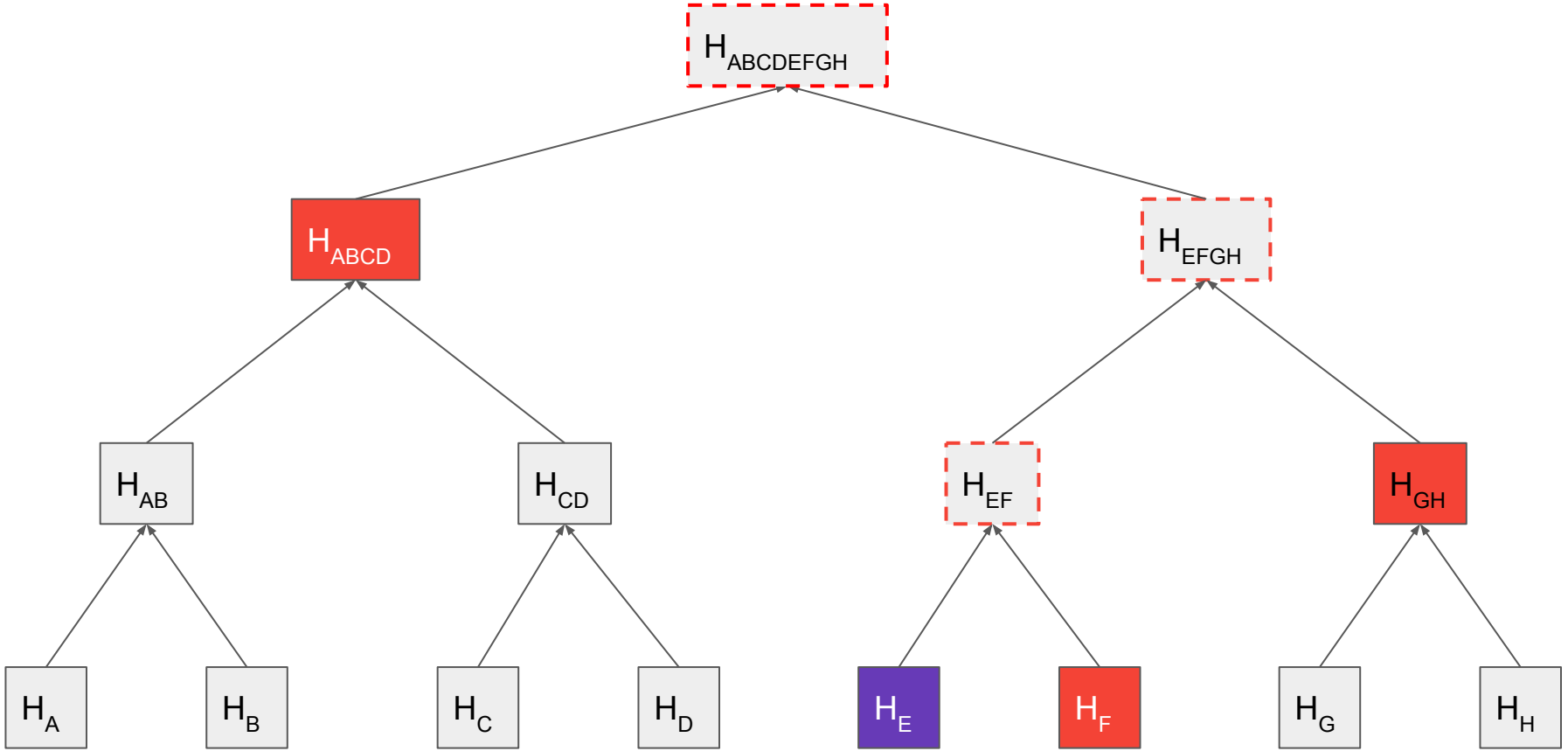
Merkle tree: proof of inclusion



Merkle tree: proof of inclusion



Merkle tree: proof of inclusion



Merkle tree

- Ethereum uses merkle trees
  - Transactions
  - State
  - Receipts
- More on future lectures

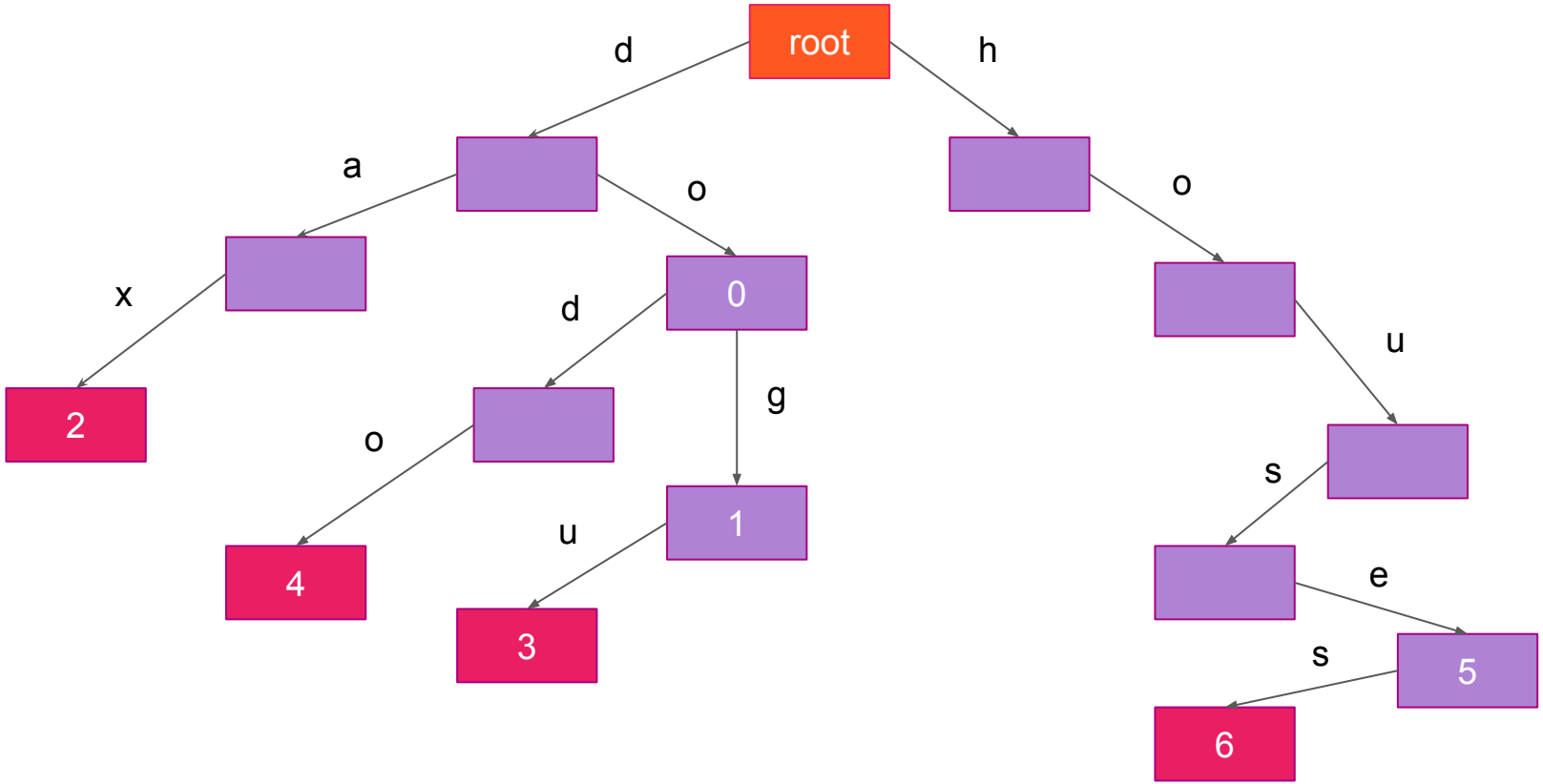
Tries

- Called also radix tree or prefix tree
- Search tree: ordered tree data structure
- Used to store a set or an associative array
- Keys usually are strings

Tries: example

{ **do**: 0, **dog**: 1, **dax**: 2, **dogu**: 3, **dodo**: 4, **house**: 5, **houses**: 6 }

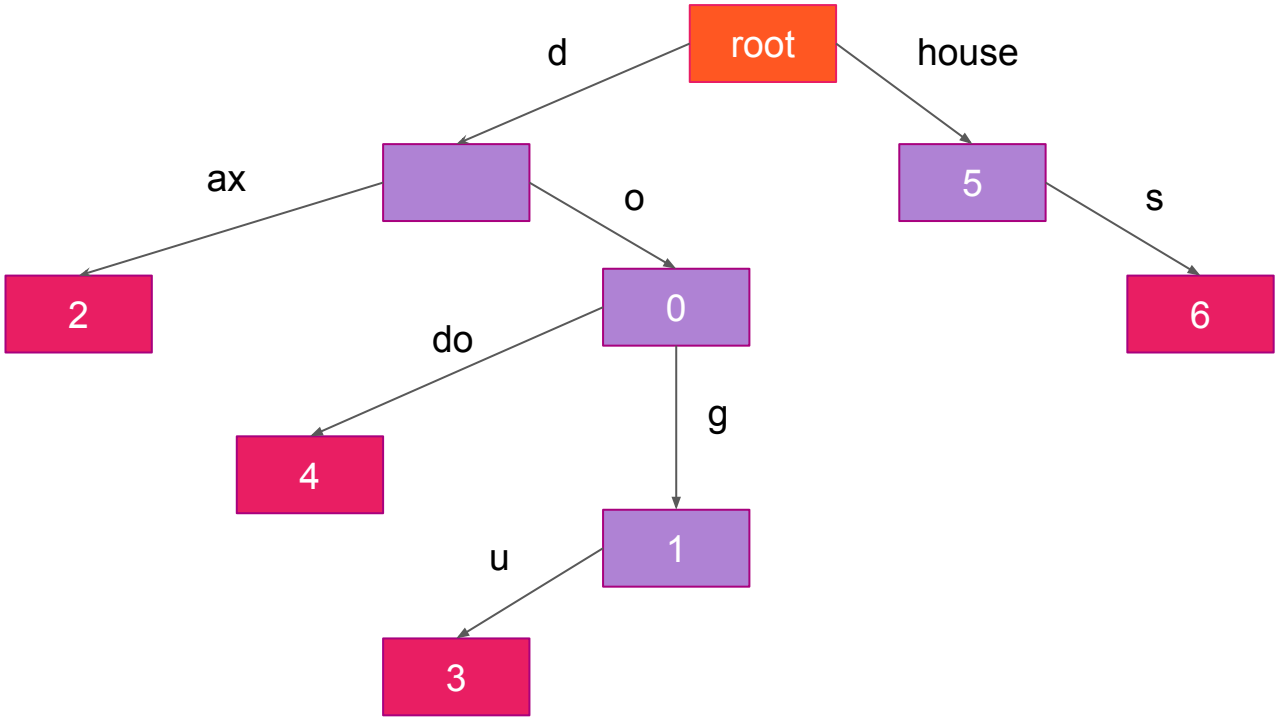
Tries



Patricia (or radix) trie

- Space-optimized trie
- Each node that is the only child is merged with its parent

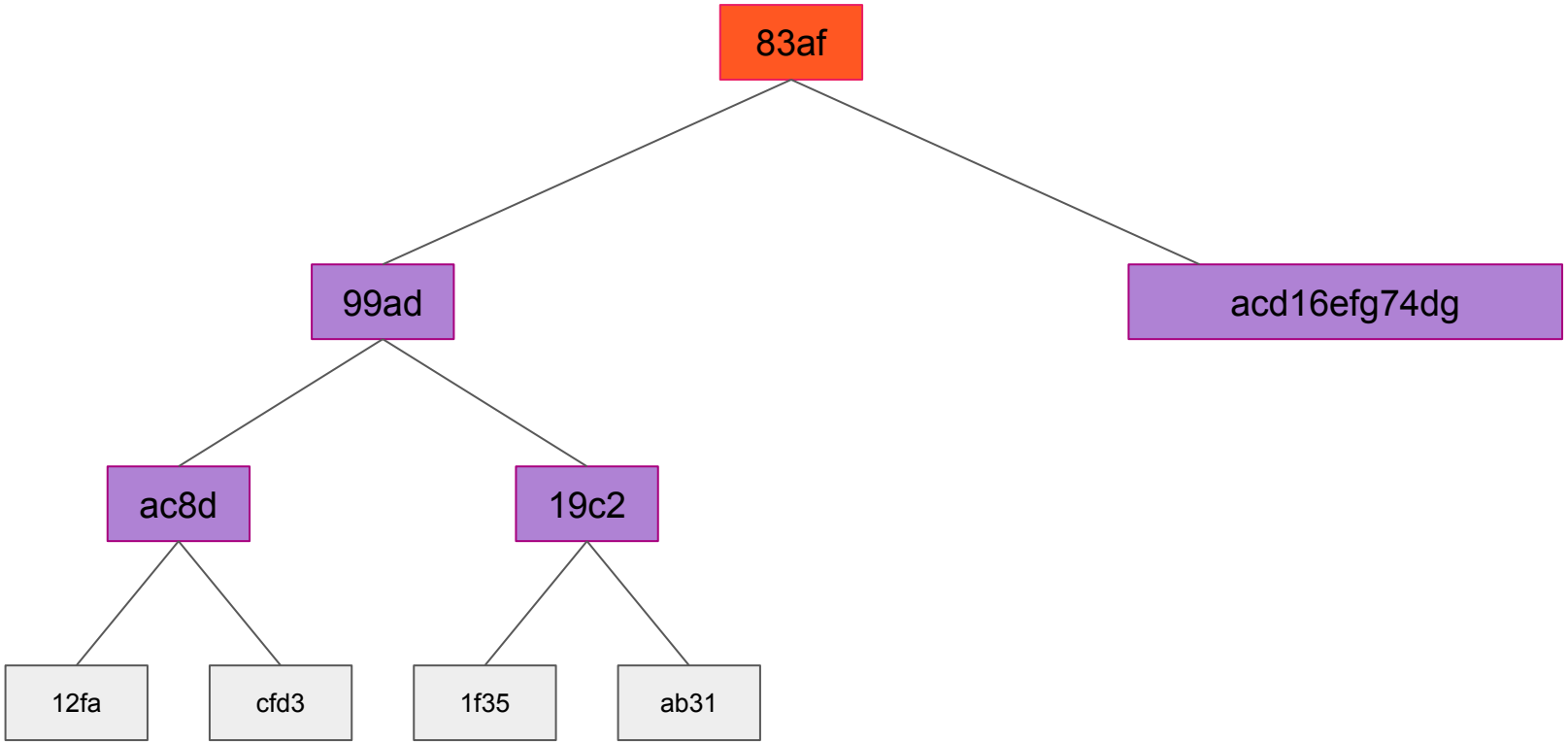
Patricia trie



Merkle patricia trie

- A combination of merkle tries and patricia tries
- First implemented in Ethereum
- Allows proof of inclusion and state look up:
  - Balance
  - Account existence
  - Smart contract values

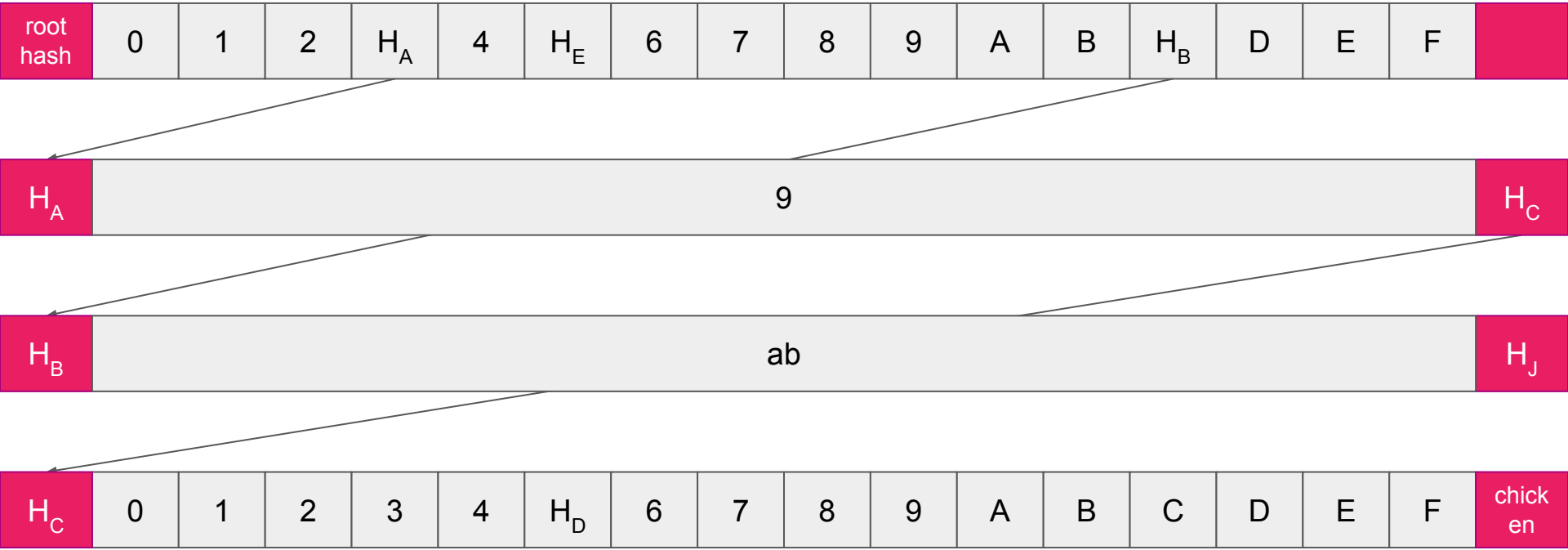
Merkle Patricia trie

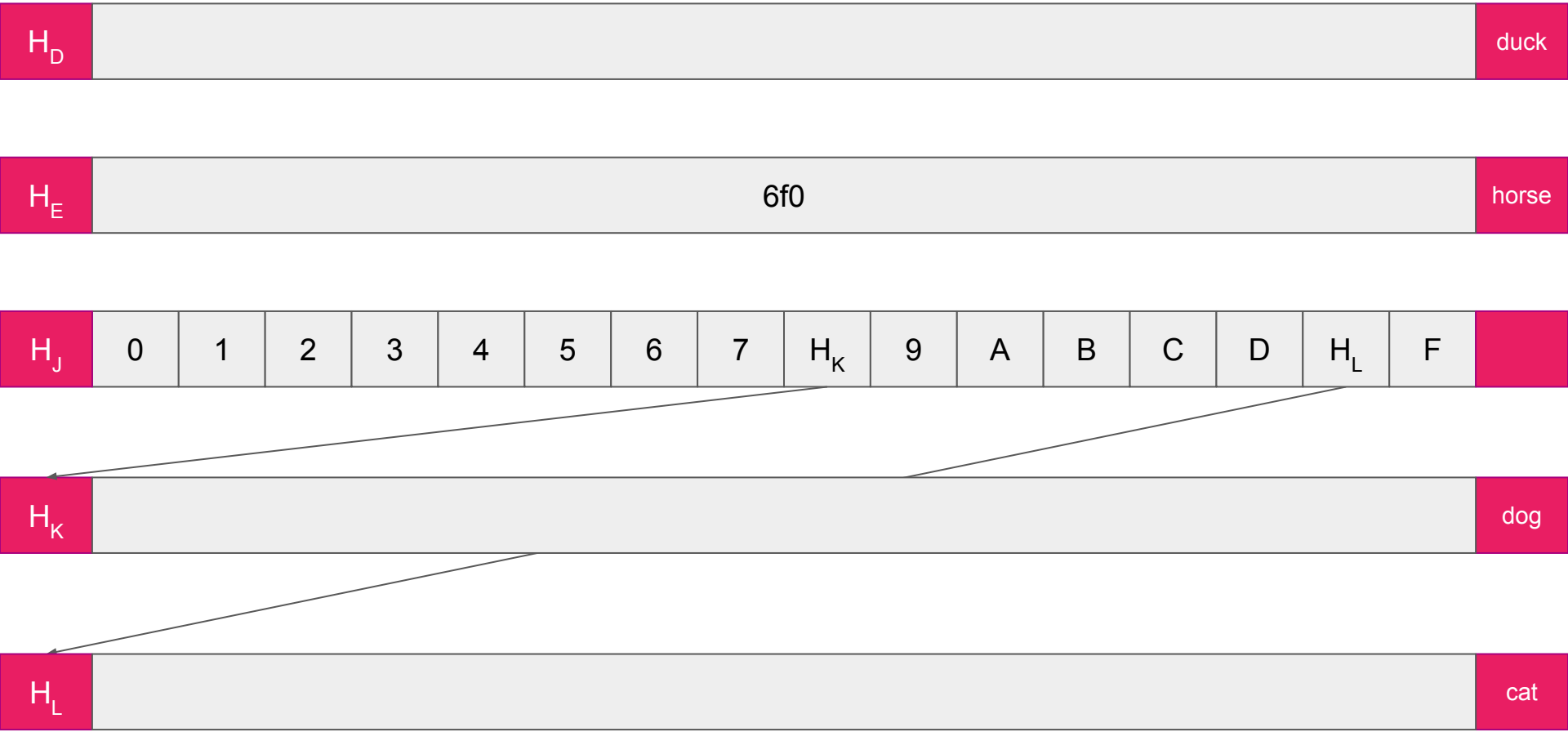


Merkle patricia trie: node

Merkle patricia trie: example

{ **'cab8'**: 'dog', **'cabe'**: 'cat', **'39'**: 'chicken', **'395'**: 'duck', **'56f0'**: 'horse' }


root hash
0
1
2
H_A
4
H_E
6
7
8
9
A
B
H_B
D
E
F
H_A
9
H_C
H_B
ab
H_J
H_C
0
1
2
3
4
H_D
6
7
8
9
A
B
C
D
E
F
chick en


H_D
duck
H_E
6f0
horse
H_J
0
1
2
3
4
5
6
7
H_K
9
A
B
C
D
H_L
F
H_K
dog
H_L
cat

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 0.9 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


BY