Lab 07:
Merkle Trees and Patricia Tries

Luke Strgar & Gary Li

LAB OUTLINE

0 ▶ CONSENSUS & STATE

1 ▶ MERKLE TREE REFRESHER

2 ▶ MERKLE PATRICIA TRIE REFRESHER

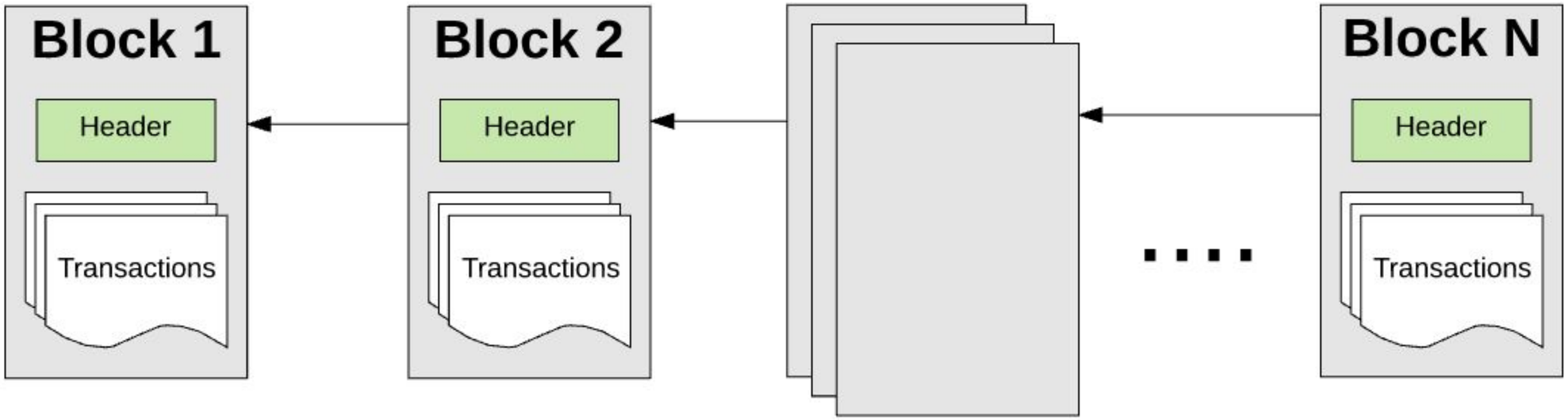
3 ▶ ASSIGNMENT

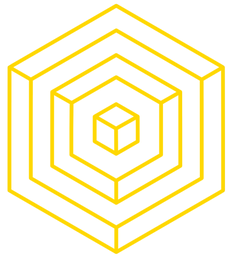
0 CONSENSUS & STATE

WHAT DOES CONSENSUS LOOK LIKE

INTERNAL TRANSACTIONS

In a blockchain, there's this idea of global truth, and everyone accepts this in order to make progress





WHAT DOES CONSENSUS LOOK LIKE

INTERNAL TRANSACTIONS

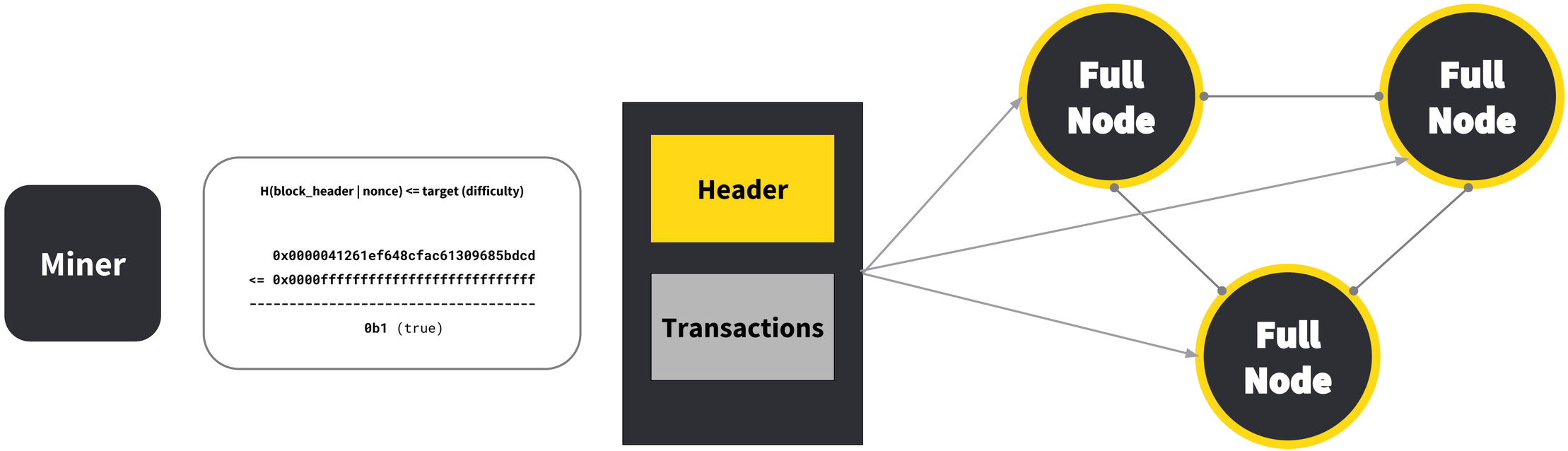
This truth is agreed upon via mining and peer validation

H(block_header | nonce) <= target (difficulty)

```
  0x0000041261ef648cfac61309685bdcd
<= 0x0000ffffffffffffffffffffffffff
-----------------------------------
  0b1 (true)
```

WHAT DOES CONSENSUS LOOK LIKE

INTERNAL TRANSACTIONS



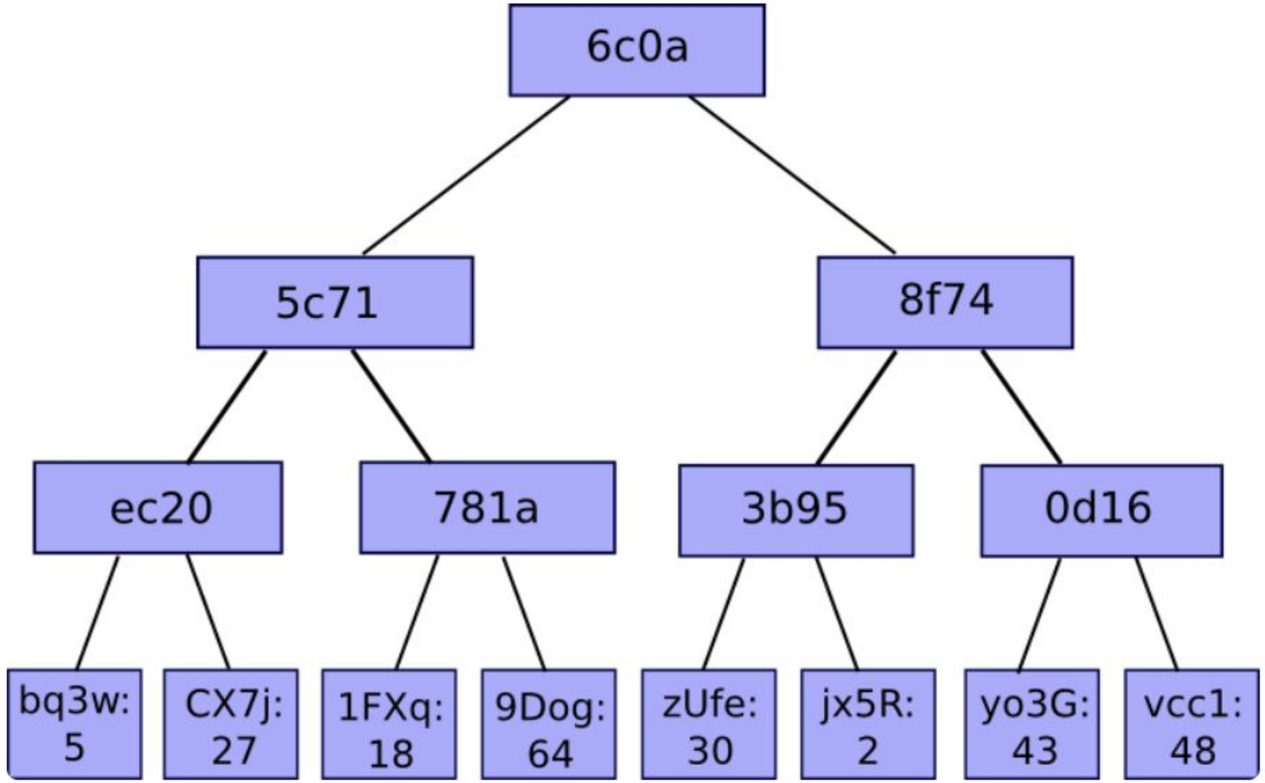
1 MERKLE TREE

WHAT IS A MERKLE TREE?

A merkle tree is a hash tree

Supports data integrity, efficient verification of data being included

```
For each (X,Y) in Pairwise(Data):
  Data.append(Hash(X+Y))
  Data.pop(X); Data.pop(Y)
  If Length(Data) == 1:
    Break
```



PROPERTIES

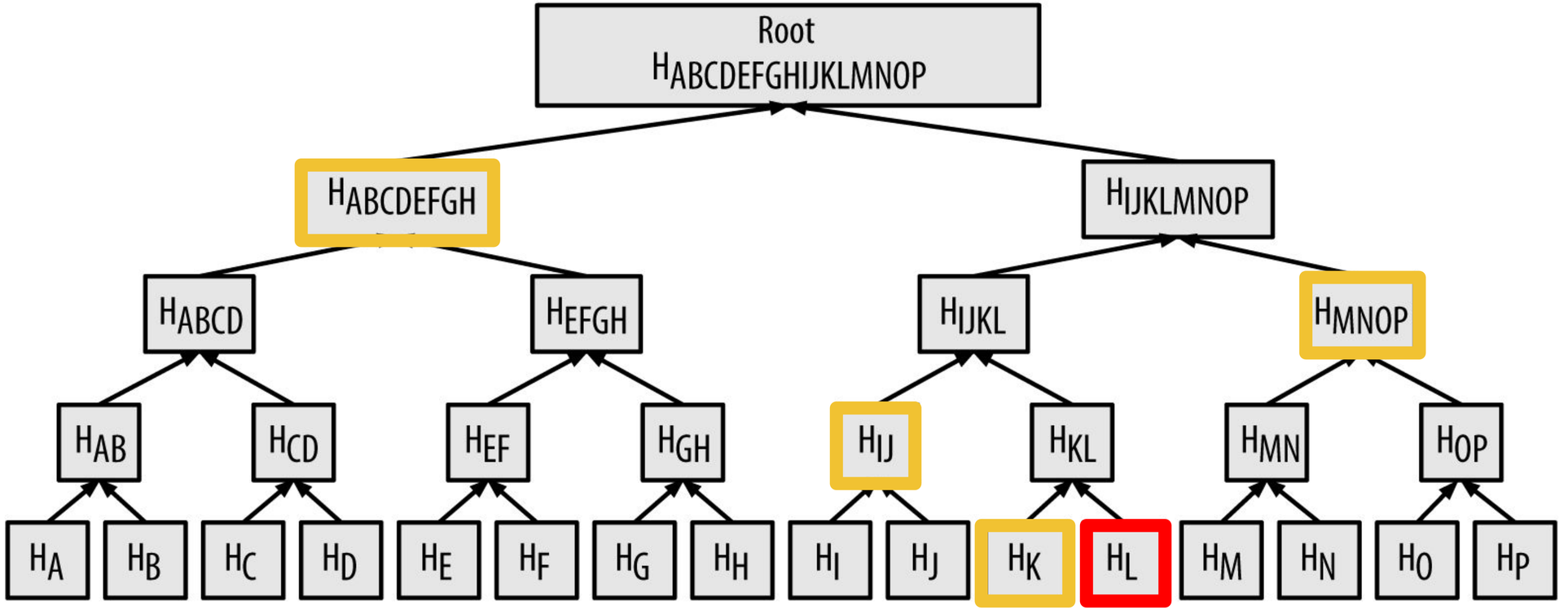
Easy to verify data integrity

- A single hash value (the root) is representative of all transaction data.
- If any of the data changes the root hash will change

Easy to verify inclusion of a transaction (merkle proof)

- This enables SPV (Simple Payment Verification).
- You no longer need to hold a copy of the entire chain.

MERKLE PROOF

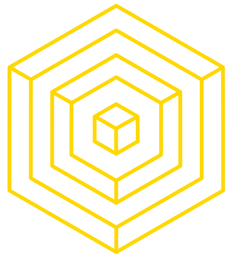


EXTENSIONS

Single merkle tree enables simple light client (SPV)

Does NOT tell you anything about current state (i.e. how many bitcoin I hold, my account balance on ethereum)

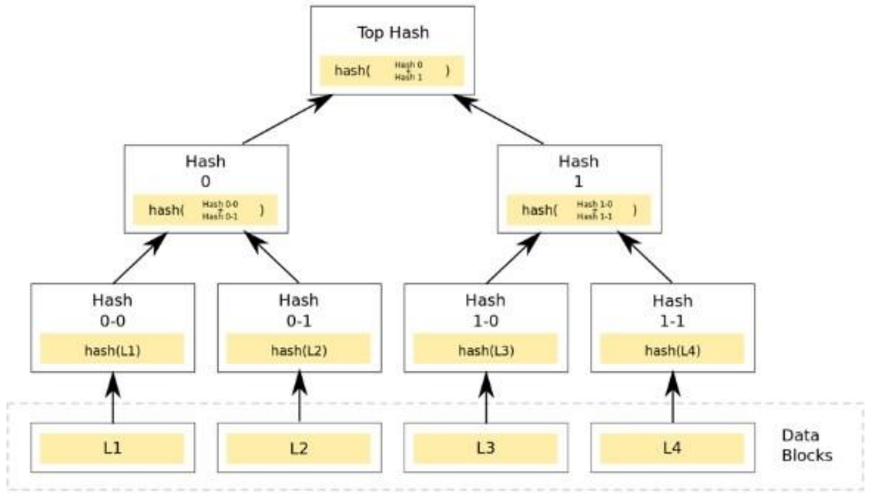
Ethereum blocks include THREE merkle tries -- transactions, receipts, state



2 MERKLE PATRICIA TRIE

WHY THE NEED FOR A MPT?

- Unlike in Bitcoin, a block header in Ethereum contains not just one merkle tree, but three trees for three different objects:
  - Transactions
  - Receipts
  - **State**
- Merkle Trees good for transactions
  - A list of transactions in a block never changes
  - Edit time doesn't matter



WHY THE NEED FOR A MPT?

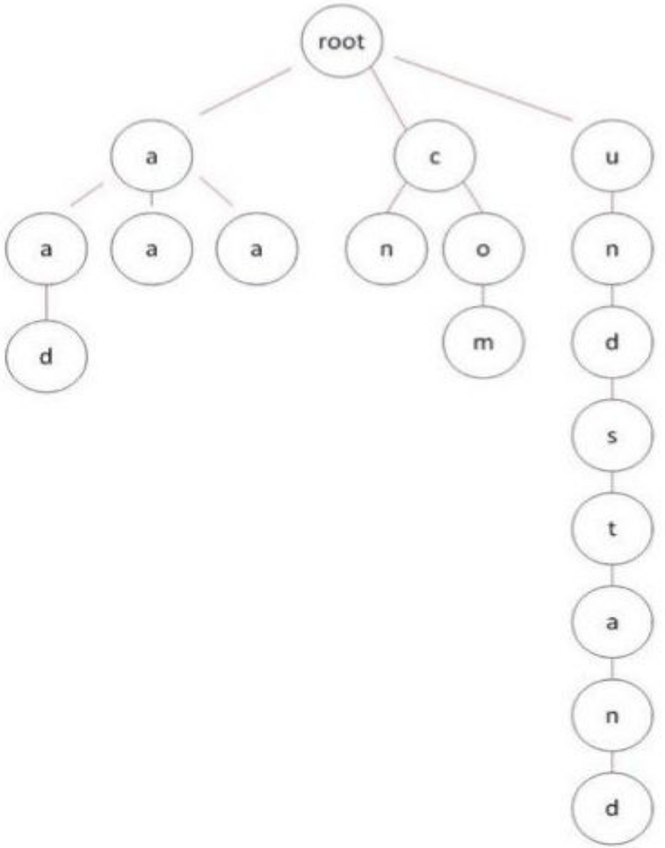
- **State Tree**
  - The State in Ethereum is essentially a **key-value mapping**, where keys are account addresses and values are account declarations, including account balance, nonce, code (if contract account)
  - Changes frequently
    - New accounts are created, balances change, account nonces update
- Transaction Trie
  - Also key-value mapping where key is the transaction ID

WHY THE NEED FOR A MPT?

- **We want a tree data structure that:**
  - Quickly recalculates tree root after an insert, edit, or delete operation
    - New accounts created, balances updated, etc.
  - Has bounded depth
    - Prevents Denial-of-Service attacks where attacker crafts transactions to make tree as deep as possible, making updates very slow
  - Has a tree root dependent only on the data, not on the order in which updates are made
    - Making updates in different orders should result in the same state and tree root
    - Roots of regular merkle trees depend on the order of leaves

WHAT IS A MERKLE PATRICIA TRIE?

- A **trie**, also known as a radix tree or a prefix tree, is an ordered tree data structure that is used to store key-value mappings where the key is the actual path taken through the tree to get to its corresponding value
  - Allows keys that begin with the same sequence to have values that are closer together in the tree
  - Drawbacks
    - Can be inefficient when there is a long key where no other key shares a common prefix



WHAT IS A MERKLE PATRICIA TRIE?

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

- Improvements made in Merkle Patricia Trie
  - Nodes are referenced by their hash
    - Thus, root node can act as a cryptographic fingerprint for the data structure (where the Merkle comes in)
  - Four node types are added to improve efficiency
    - **Empty Nodes:** Simply blank nodes
    - **Leaf Nodes:** Standard node with a key and a value
    - **Branch Nodes:** List of length 17; first 16 elements correspond to the 16 hex chars that can be in a key and the last element represents the value if there is a key-value pair where the key ends at the branch node
    - **Extension Nodes:** Key-value node where the value is the hash of another node

WHAT IS A MERKLE PATRICIA TRIE?

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

- Keys in MPTs are encoded with a special Hex-Prefix (HP) encoding
  - Recall that both extension and leaf nodes have their format as a key-value mapping
  - Nibble: one hex character
  - A nibble is appended to the beginning of the key to signify both parity (even or odd length key) and terminator status (whether the node is a extension node or leaf node)
    - The lowest significant bit encodes parity, while the next lowest encodes terminator status
    - If the original key was of even length, an extra zero nibble is appended to achieve overall evenness, so the key can be properly represented in bytes

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

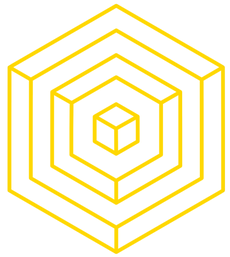
Leaf Node

prefix	*key-end*	*value*
3□	7	0.12ETH

WHAT IS A MERKLE PATRICIA TRIE?

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

- In general, when inserting into a MPT
  - If you stopped at an empty node, you add a new leaf node with the remaining path and replace the empty node with the hash of the new leaf node
  - If you stopped at a leaf node, you need to convert it to an extension node and add a new branch and a new leaf node
  - If you stopped at an extension node, you convert it to another extension node with shorter path and create a new branch and leaves



3

ASSIGNMENT

ASSIGNMENT

MERKLE TREE AND MERKLE TRIE DEMO

github.com/Blockchain-for-Developers/merkle-tree

github.com/Blockchain-for-Developers/merkle-patricia-trie

SEE YOU NEXT TIME

NO LECTURE TOMORROW

HAVE A GREAT SPRING BREAK!!!!!!!!!!!!!!!