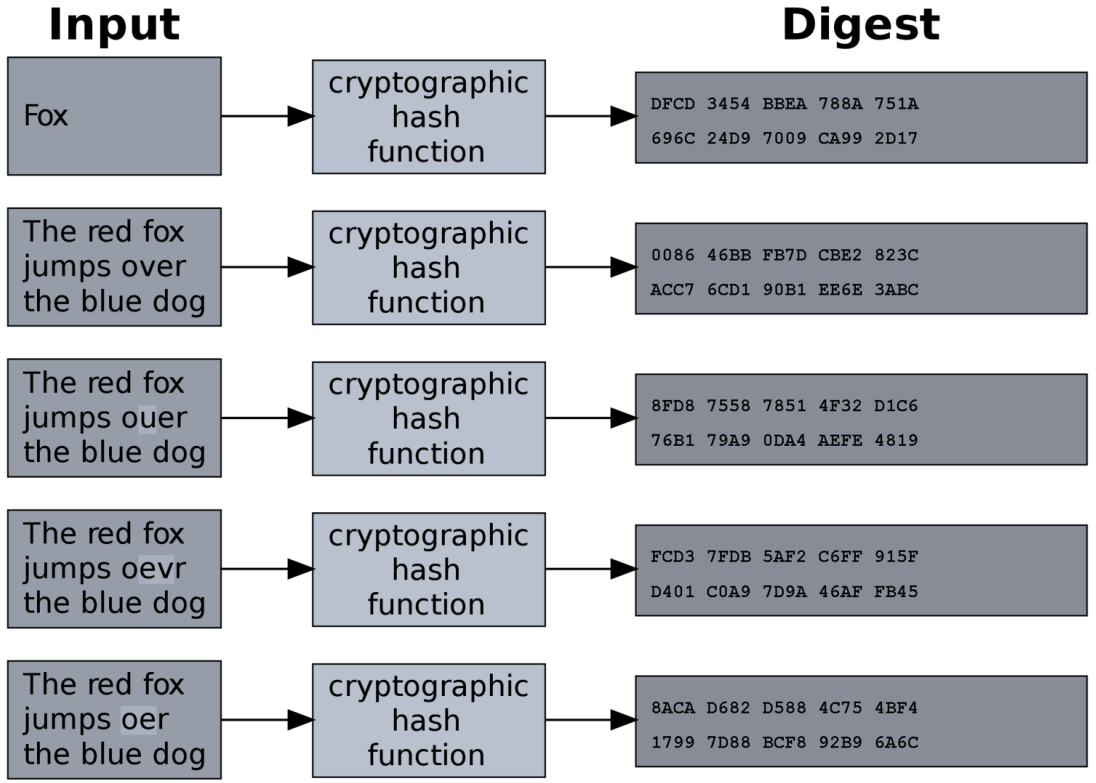
Blockchain

raed**RASHEED**

Hash Function

- A hash function is any function that can be used to map data of arbitrary size to data of fixed size.



Hash Function

- A hash function is any function that can be used to map data of arbitrary size to data of fixed size.

https://andersbrownworth.com/blockchain/hash

Structure of a Block

- The block is made of a header, containing metadata, followed by a long list of transactions that make up the bulk of its size.
- The block header is 80 bytes, whereas the average transaction is at least 250 bytes and the average block contains more than 500 transactions.
- A complete block, with all transactions, is therefore 1,000 times larger than the block header.

Structure of a Block

Size	Field	Description
4 bytes	Block Size	The size of the block, in bytes, following this field
80 bytes	Block Header	Several fields form the block header
1–9 bytes (VarInt)	Transaction Counter	How many transactions follow
Variable	Transactions	The transactions recorded in this block

Block Header

- The block header consists of three sets of block metadata.
  - First, there is a reference to a previous block hash, which connects this block to the previous block in the blockchain.
  - The second set of metadata, namely the difficulty, timestamp, and nonce, relate to the mining competition.
  - The third piece of metadata is the merkle tree root, a data structure used to efficiently summarize all the transactions in the block.

Block Header

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 bytes	Difficulty Target	The Proof-of-Work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the Proof-of-Work algorithm

Block Identifiers

- Block Header Hash
  - The primary identifier of a block is its cryptographic hash, a digital fingerprint, made by hashing the block header twice through the SHA256 algorithm.

    000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
- Block Height
  - A second way to identify a block is by its position in the blockchain, called the block height. The first block ever created is at block height 0 (zero)

The Genesis Block

- The first block in the blockchain is called the genesis block and was created in 2009.

https://www.blockchain.com/btc/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

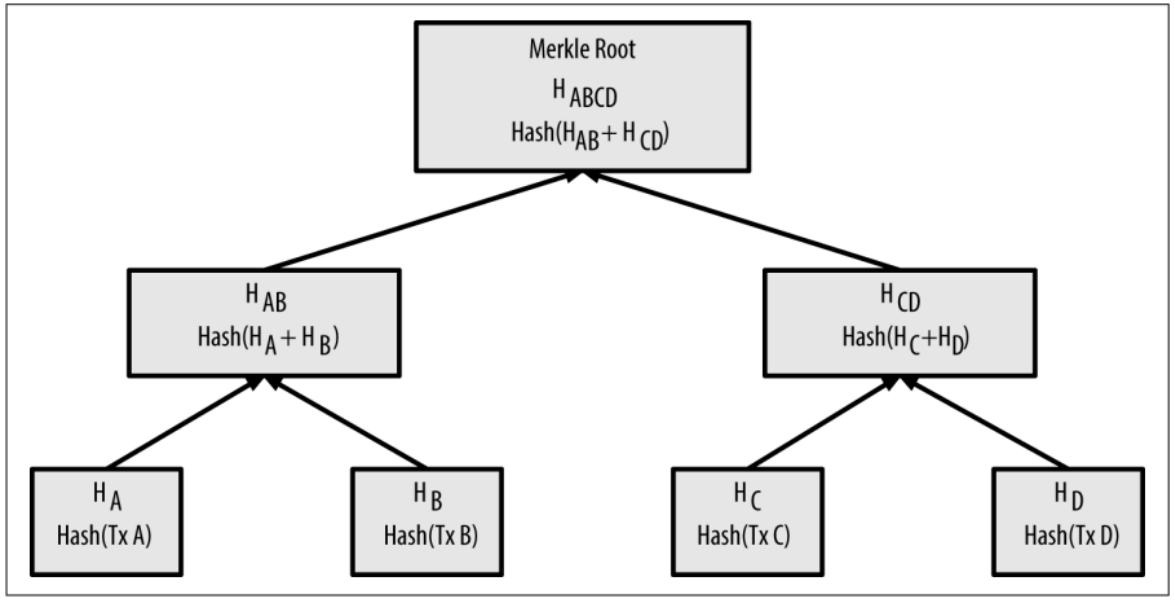
```
$ bitcoin-cli getblock
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

{
    "hash" : "000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
    "confirmations" : 308321,
    "size" : 285,
    "height" : 0,
    "version" : 1,
    "merkleroot" :
"4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",
    "tx" : [
        "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
    ],
    "time" : 1231006505,
    "nonce" : 2083236893,
    "bits" : "1d00ffff",
    "difficulty" : 1.00000000,
    "nextblockhash" :
"00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048"
}
```

Merkle Trees

- A merkle tree, also known as a binary hash tree, is a data structure used for efficiently summarizing and verifying the integrity of large sets of data.
- Merkle trees are binary trees containing cryptographic hashes.
- The merkle tree is constructed bottom-up.
- we start with four transactions, A, B, C, and D, which form the leaves of the merkle tree.
  - H_A = SHA256(SHA256(Transaction A))
  - H_{AB} = SHA256(SHA256(H_A + H_B))

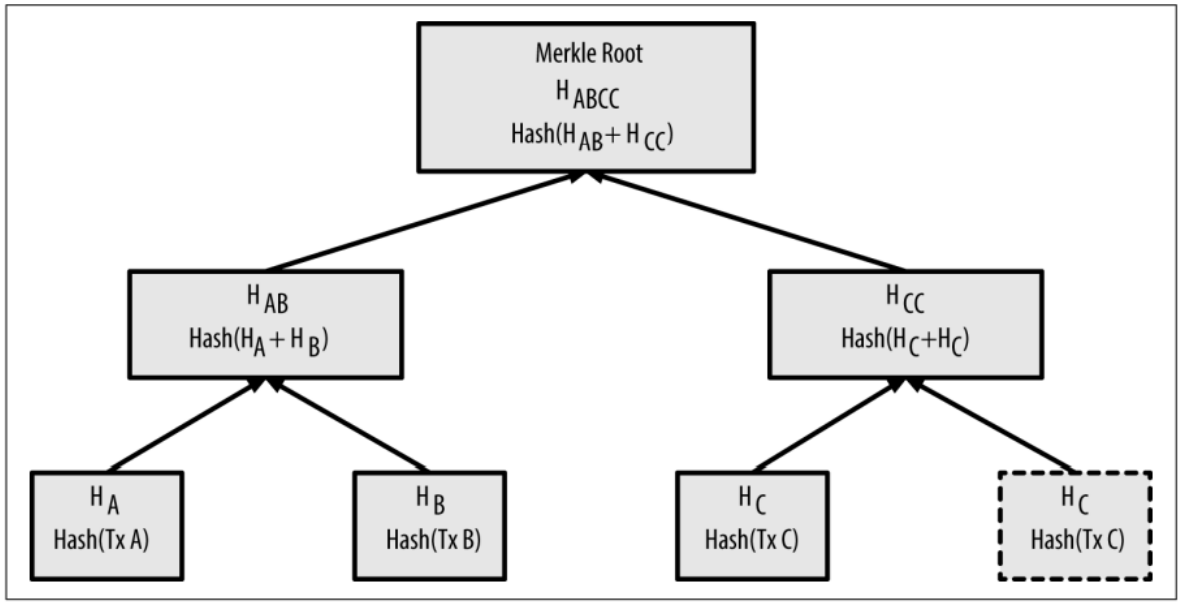
Merkle Trees

- A merkle tree



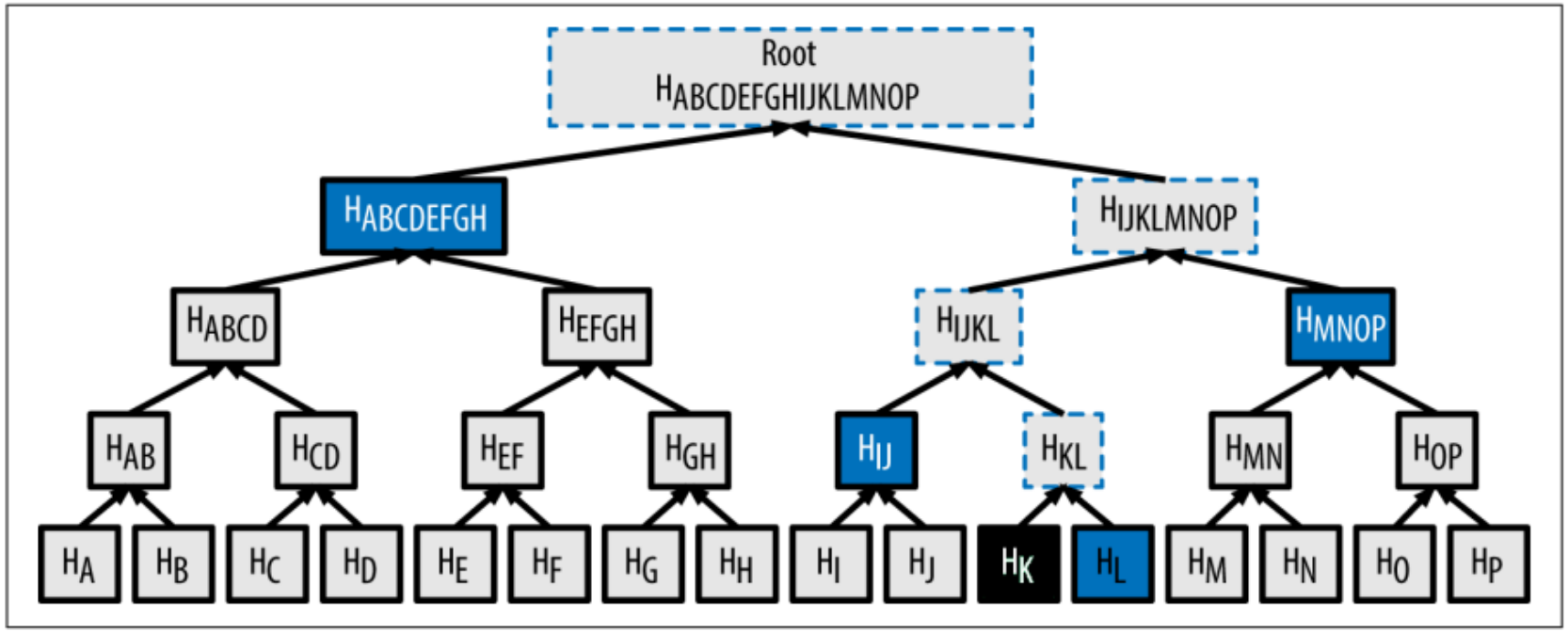
Merkle Trees

- If there is an odd number of transactions to summarize, the last transaction hash will be duplicated to create an even number of leaf nodes, also known as a balanced tree.



Merkle Trees

- A node can prove that a transaction K is included in the block by producing a merkle path that is only four 32-byte hashes long (128 bytes total). The path consists of the four hashes.



Merkle Trees

- Merkle tree efficiency

Number of transactions	Approx. size of block	Path size (hashes)	Path size (bytes)
16 transactions	4 kilobytes	4 hashes	128 bytes
512 transactions	128 kilobytes	9 hashes	288 bytes
2048 transactions	512 kilobytes	11 hashes	352 bytes
65,535 transactions	16 megabytes	16 hashes	512 bytes

Mining and Consensus

raed**RASHEED**

Mining

- The purpose of mining is not the creation of new bitcoin. That's the incentive system.
- Mining is the mechanism by which bitcoin's security is decentralized.
- Miners validate new transactions and record them on the global ledger.
- A new block, containing transactions that occurred since the last block, is "mined" every 10 minutes on average, thereby adding those transactions to the blockchain.

Mining

- Miners receive two types of rewards in return for the security provided by mining:
  - new coins created with each new block, and
  - transaction fees from all the transactions included in the block.
- The solution to the problem, called the Proof-of-Work, is included in the new block and acts as proof that the miner expended significant computing effort.

Consensus Models

- Proof of Work Consensus Model.
- Proof of Stake Consensus Model.
- Round Robin Consensus Model.
- Proof of Authority/Proof of Identity Consensus Model.
- Proof of Elapsed Time Consensus Model.

Proof of Work Consensus Model

- In the proof of work (PoW) model, a user publishes the next block by being the first to solve a computationally intensive puzzle.
- The puzzle is designed such that solving the puzzle is difficult but checking that a solution is valid is easy.
- This enables all other full nodes to easily validate any proposed next blocks, and any proposed block that did not satisfy the puzzle would be rejected.
- The target value may be modified over time to adjust the difficulty (up or down) to influence how often blocks are being published.

Proof of Work Consensus Model

- For example, Bitcoin, which uses the proof of work model, adjusts the puzzle difficulty every 2016 blocks to influence the block publication rate to be around once every ten minutes.
- The adjustment is made to the difficulty level of the puzzle, and essentially either increases or decreases the number of leading zeros required.
- Adjustments to the difficulty target aim to ensure that no entity can take over block production.

Proof of Stake Consensus Model

- The proof of stake (PoS) model is based on the idea that the more stake a user has invested into the system, the more likely they will want the system to succeed, and the less likely they will want to subvert it.
- Stake is often an amount of cryptocurrency that the blockchain network user has invested into the system.
- With this consensus model, there is no need to perform resource intensive computations (involving time, electricity, and processing power) as found in proof of work.

Round Robin Consensus Model

- Within this model of consensus, nodes take turns in creating blocks.
- To handle situations where a publishing node is not available to publish a block on its turn, these systems may include a time limit to enable available nodes to publish blocks so that unavailable nodes will not cause a halt in block publication.
- This model ensures no one node creates the majority of the blocks.

Proof of Authority/Proof of Identity Consensus Model

- The proof of authority (also referred to as proof of identity) consensus model relies on the partial trust of publishing nodes through their known link to real world identities.
- Publishing nodes must have their identities proven and verifiable within the blockchain network.

Proof of Elapsed Time Consensus Model

- Within the proof of elapsed time (PoET) consensus model, each publishing node requests a wait time from a secure hardware time source within their computer system.
- The secure hardware time source will generate a random wait time and return it to the publishing node software.
- Publishing nodes take the random time they are given and become idle for that duration.
- Once a publishing node wakes up from the idle state, it creates and publishes a block to the blockchain network, alerting the other nodes of the new block; any publishing node that is still idle will stop waiting, and the entire process starts over.

The difficulty of mining

- The difficulty of mining a bitcoin block is approximately 10 minutes of processing for the entire network, based on the time it took to mine the previous 2,016 blocks, adjusted every 2,016 blocks.
- This is achieved by lowering or raising the target.

New Target = Old Target * (Actual Time of Last 2016 Blocks / 20160 minutes)

Solidity

raed**RASHEED**

Solidity

Solidity is an object-oriented, high-level language for implementing smart contracts.

Designed to target the Ethereum Virtual Machine (EVM).

Pragma

```
pragma solidity ^0.5.2;
pragma solidity >=0.5.2 < 0.9.0;
```

Importing other Source Files

```
import “filename";
import * as symbolName from "filename";
import "filename" as symbolName;
```

Comments

```
// This is a single-line comment.

/* This is
a multi-line
comment. */
```

Structure of a Contract

- Each contract can contain declarations of
  - Variables,
  - Functions,
  - Function Modifiers,
  - Events, Events are convenience interfaces with the EVM logging facilities.
  - Errors, Errors allow you to define descriptive names and data for failure situations.
  - Struct Types and
  - Enum Types.

State Variables

Variables whose values are permanently stored in a contract storage.

```
pragma solidity ^0.5.0;
contract SolidityTest {
   uint storedData;      // State variable
   constructor() public {
      storedData = 10;   // Using State variable
   }
}
```

Local Variables

Variables whose values are present till function is executing.

```
pragma solidity ^0.5.0;
contract SolidityTest {
   uint storedData; // State variable
   constructor() public {
      storedData = 10;
   }
```

Local Variables

Variables whose values are present till function is executing.

```
    function getResult() public view returns(uint){
        uint a = 1; // local variable
        uint b = 2;
        uint result = a + b;
        return result; //access the local variable
    }
}
```

Global Variables

Special variables exists in the global namespace used to get information about the blockchain.

Name	Returns
blockhash(uint blockNumber) returns (bytes32)	Hash of the given block - only works for 256 most recent, excluding current, blocks
block.coinbase (address payable)	Current block miner's address
block.difficulty (uint)	Current block difficulty
block.gaslimit (uint)	Current block gaslimit
block.number (uint)	Current block number

Global Variables

Special variables exists in the global namespace used to get information about the blockchain.

Name	Returns
block.timestamp (uint)	Current block timestamp as seconds since unix epoch
gasleft() returns (uint256)	Remaining gas
msg.data (bytes calldata)	Complete calldata
msg.sender (address payable)	Sender of the message (current caller)
msg.sig (bytes4)	First four bytes of the calldata (function identifier)
msg.value (uint)	Number of wei sent with the message

Global Variables

Special variables exists in the global namespace used to get information about the blockchain.

Name	Returns
now (uint)	Current block timestamp
tx.gasprice (uint)	Gas price of the transaction
tx.origin (address payable)	Sender of the transaction

Functions

```
pragma solidity >=0.7.1
contract SimpleAuction {
    function bid() public payable { // Function
    // ...
    }
}
```

Function Modifiers

```
pragma solidity >=0.4.22
contract Purchase {
    address public seller;
    modifier onlySeller() { // Modifier
        require(msg.sender == seller, "Only seller can call this." );
        _;
    }
    function abort() public view onlySeller { // Modifier usage
        // ...
    }
}
```

Struct Types

```
pragma solidity >=0.4.0
contract Ballot {
    struct Voter { // Struct
        uint weight;
        bool voted;
        address delegate;
        uint vote;
    }
}
```

Enum Types

```
pragma solidity >=0.4.0
contract Purchase {
    enum State { Created, Locked, Inactive } // Enum
}
```

Types

- bool
  - !, &&, ||, ==, and !=
- uint8 to uint256, int8 to int256
  - Comparisons: <=, =, > (evaluate to bool)
  - Bit operators: &, |, ^ (bitwise exclusive or), ~ (bitwise negation)
  - Shift operators: << (left shift), >> (right shift)
  - Arithmetic operators: +, -, unary - (only for signed integers), *, /, % (modulo), ** (exponentiation)
  - Bit operations
    - not (~), shift (x << y [x * 2**y], x >> y [x / 2**y])

Types

- Fixed-Point-Numbers refers to a method of representing fractional numbers (not integers).
- fixed / ufixed: Signed and unsigned fixed point number of various sizes.
- Keywords ufixedMxN and fixedMxN, where M represents the number of bits taken by the type and N represents how many decimal points are available.
- M must be divisible by 8 and goes from 8 to 256 bits.
- N must be between 0 and 80, inclusive.
- ufixed and fixed are aliases for ufixed128x18 and fixed128x18, respectively.

Types

- address:
  - address: Holds a 20 byte value (size of an Ethereum address).
  - address payable: Same as address, but with the additional members transfer and send.
- bytes32
- string
- bytes

Variable Size

- Fixed-size
  - bool, uint, address, and bytes32
- Variable-size
  - string, bytes, uint[], and mapping(address => uint) users
- User-define
  - struct, enum

Variable Location

Storage	Memory
Permanent Storage	temporary
It contains keys and value like a hashtable	Byte Array
Uses state variables	Local variables defined in function calls
Contract state variable	memory variables
Expensive gas operation	Does not cost gas and inexpensive operation
keyword in solidity	keyword in solidity

Function

function (<parameter types>) {internal|external} [pure|view|payable] [returns (types)]

Function Visibility

- private called from inside the contract _getValue().
- internal called from other contract inherit from contract.
- external only called from outside the contract.
- public can be called from any where.

Function Types

function (<parameter types>) {internal|external} [pure|view|payable] [returns (types)]

- Function that only reads but doesn’t alter the state variables defined in the contract is called a view function.
- Function that only do some computation is called pure function.
- Functions declared payable can receive ether into the contract.

Peer to Peer

Layers

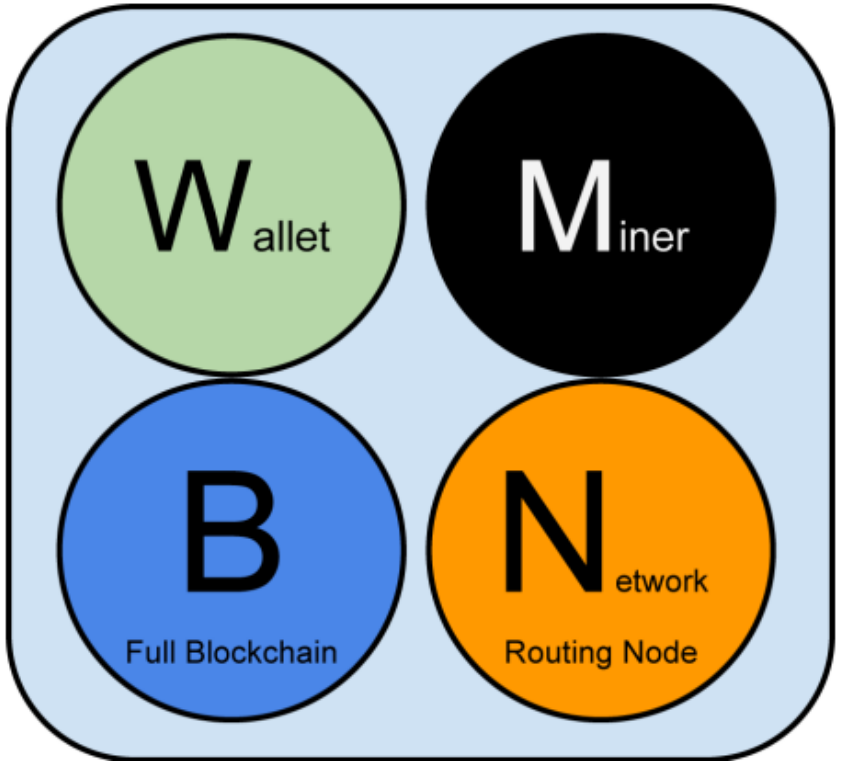
Token xShop, EPSCoin

Protocol Bitcoin, Etherum, NEO

Blockchain

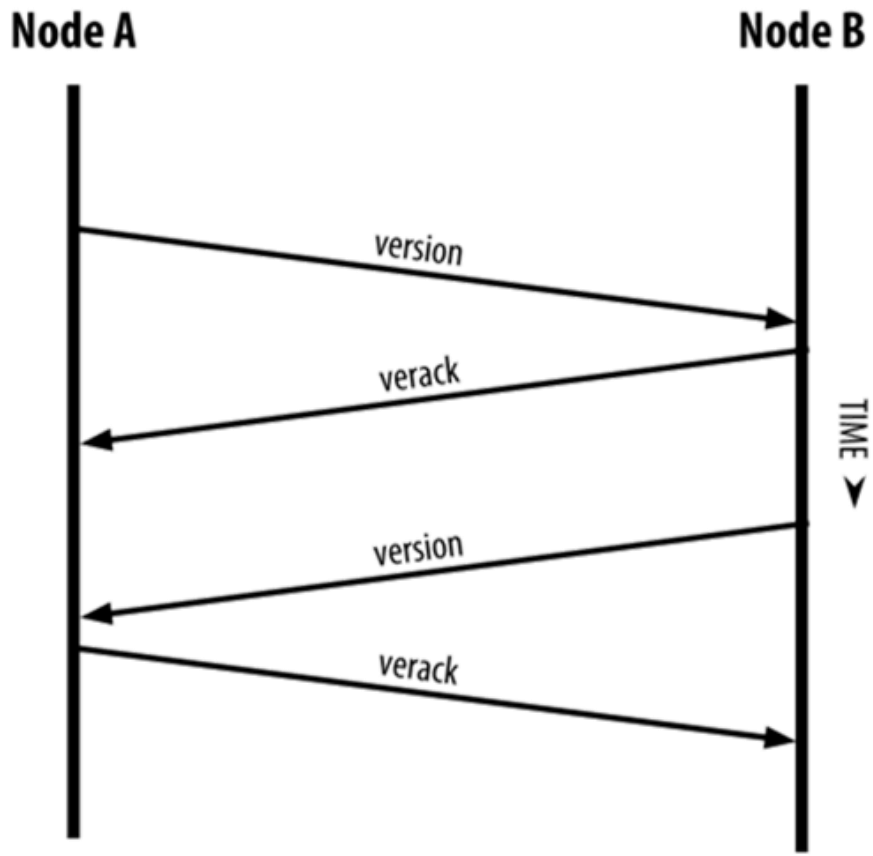
Node Types and Roles

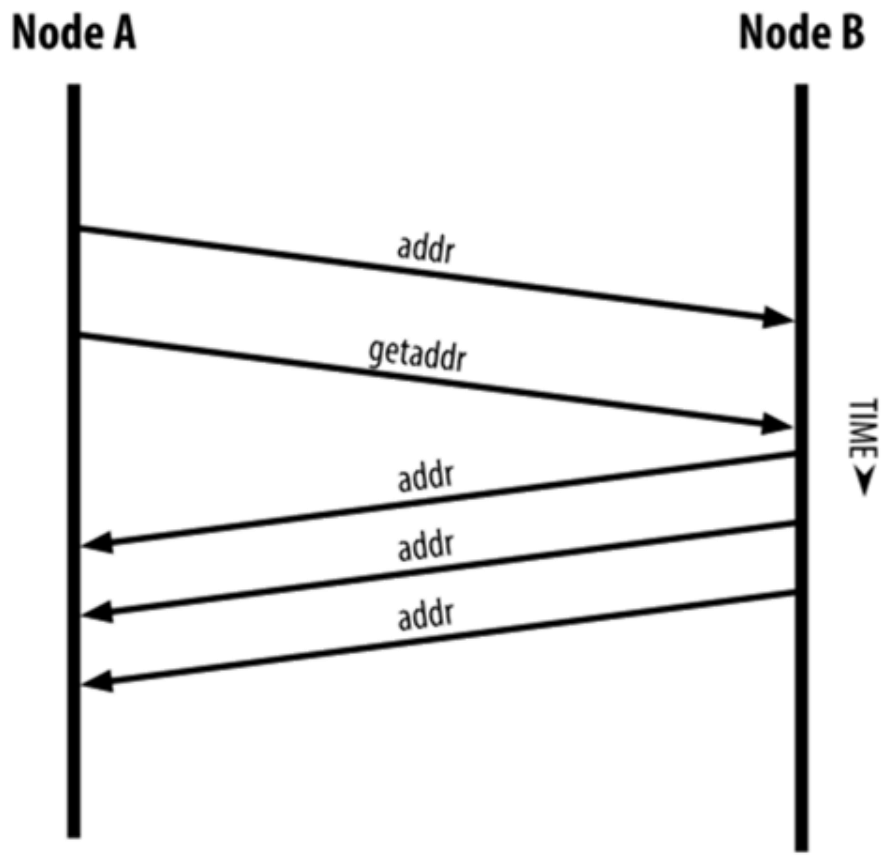
- depending on the functionality
  - routing,
  - the blockchain database,
  - mining, and
  - wallet services.

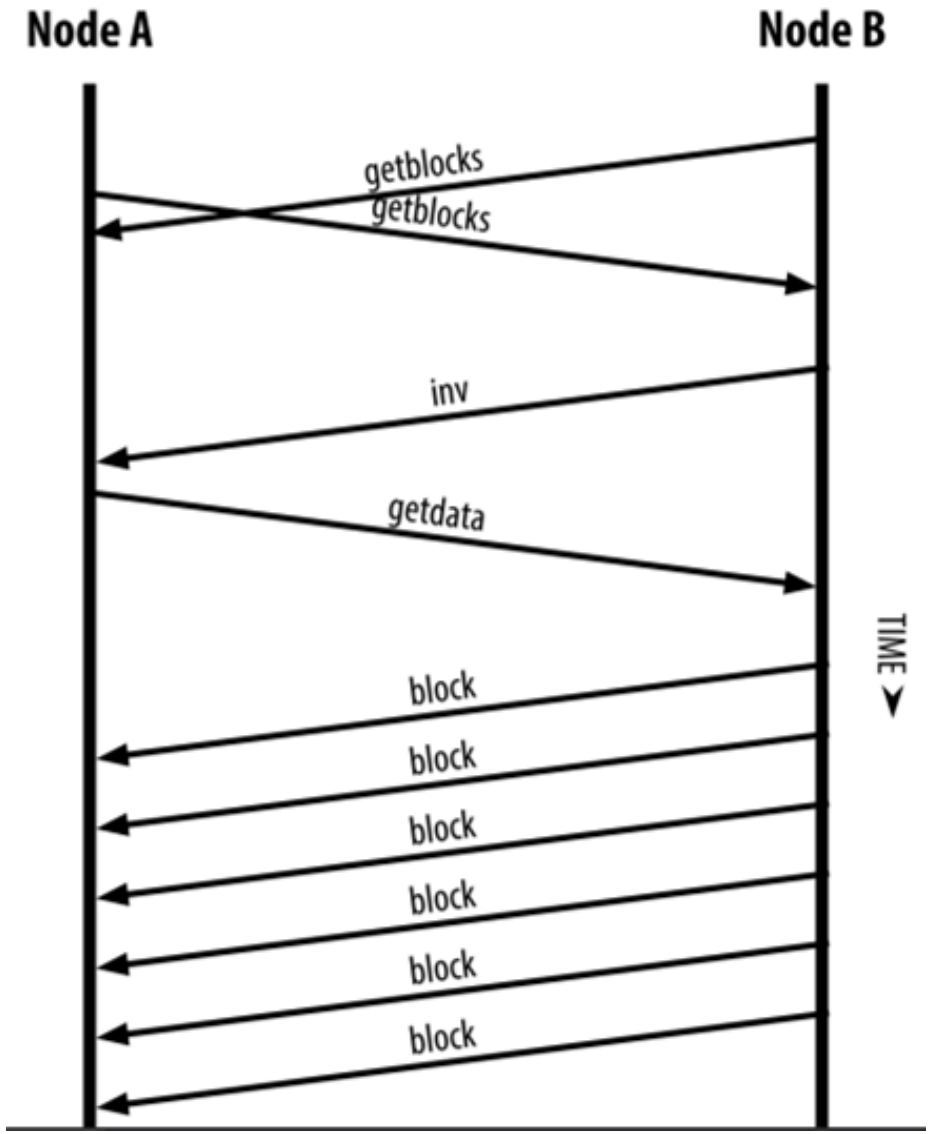


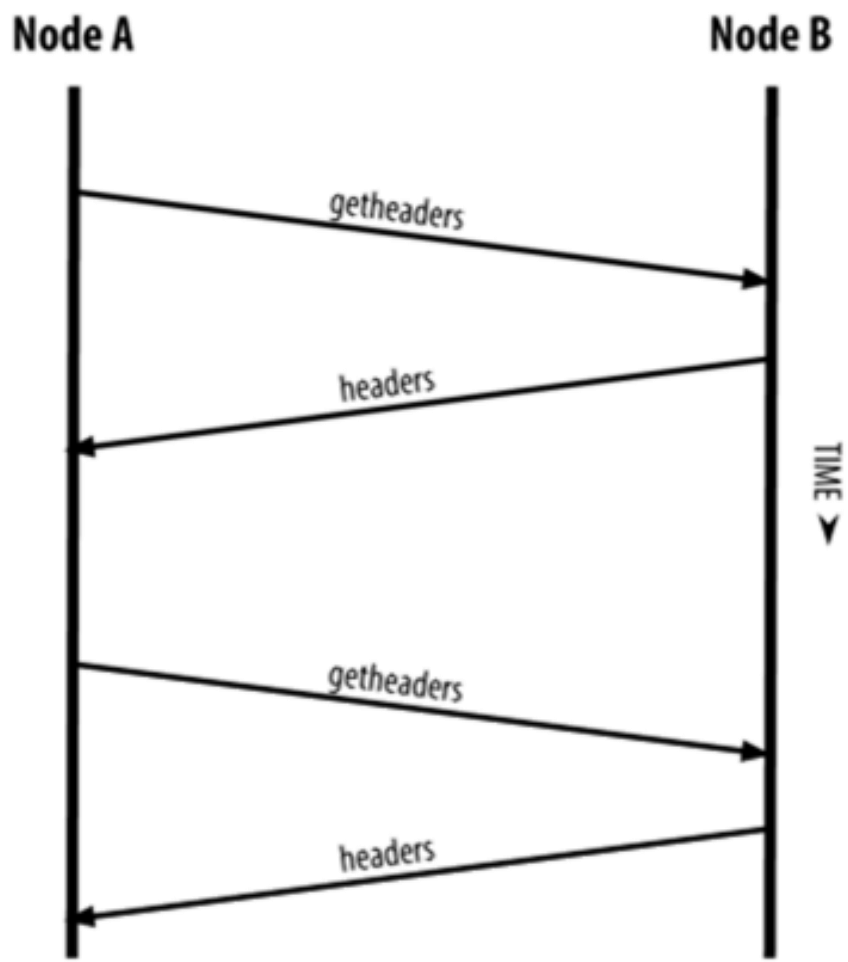
How does a new node find peers?

- DNS seeds (static list of IP addresses)
- Once one or more connections are established, the new node will send an address message containing its own IP address to its neighbors
- The neighbors will, in turn, forward the address message to their neighbors


Node A
Node B
version
verack
TIME
version
verack


Node A
Node B
addr
getaddr
addr
addr
addr
TIME


Node A
Node B
getblocks
getblocks
inv
getdata
block
block
block
block
block
block
TIME


Node A
Node B
getheaders
headers
TIME
getheaders
headers

Code (Python)

```
import socket
import threading

HEADER = 64
PORT = 5050
NODE = socket.gethostbyname(socket.gethostname())
ADDR = (NODE, PORT)

FORMAT = 'utf-8'
DISCONNECT_MESSAGE = "!exit"
```

Code (Python)

```
node = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    node.bind(ADDR)
    print("[STARTING] Node starting...")
    start()
```

Code (Python)

```
def start():
    node.listen()
    print(f"[LISTENING] Node is listening on {NODE}")
    while True:
        conn, addr = node.accept()
        thread = threading.Thread(target=handle_myPeer, args=(conn, addr))
        thread.start()
        print(f"[ACTIVE CONNECTIONS] {threading.activeCount() -1}")
```

Code (Python)

```
def handle_myPeer(conn, addr):
    print(f"[NEW CONNECTION] {addr} connected.")
    connected = True
    while connected:
        msg_length = conn.recv(HEADER).decode(FORMAT)
        if msg_length:
            msg_length = int(msg_length)
            msg = conn.recv(msg_length).decode(FORMAT)
            if msg == DISCONNECT_MESSAGE:
                connected = False
            print(f"[{addr}] {msg}")
```

Code (Python)

```
def send(msg):
    message = msg.encode(FORMAT)
    msg_length = len(message)
    send_length = str(msg_length).encode(FORMAT)
    send_length += b' ' * (HEADER - len(send_length))
    myPeer.send(send_length)
    myPeer.send(message)
    print(myPeer.recv(2048).decode(FORMAT))
```

Code (Python)

```
try:
    myPeer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    myPeer.connect(ADDR)
    send("version")
    peerconnected = True
    while peerconnected:
        msg = input("Msg>>")
        send(msg)
        if msg == DISCONNECT_MESSAGE:
            peerconnected = False
```