CSBC2000

Week 1 | Class 3

Distributed Ledger Technology as an Abstraction of Blockchain

Recap

- Ethereum is a virtual machine for any DApp's transactions
- Blockchain is an example of DLT which (in most cases) is a state machine
- Smart contracts define state transitions on the blockchain
- PoS is an improvement over PoW
- There are several interesting blockchain protocols, some backed by physical utilities

This class

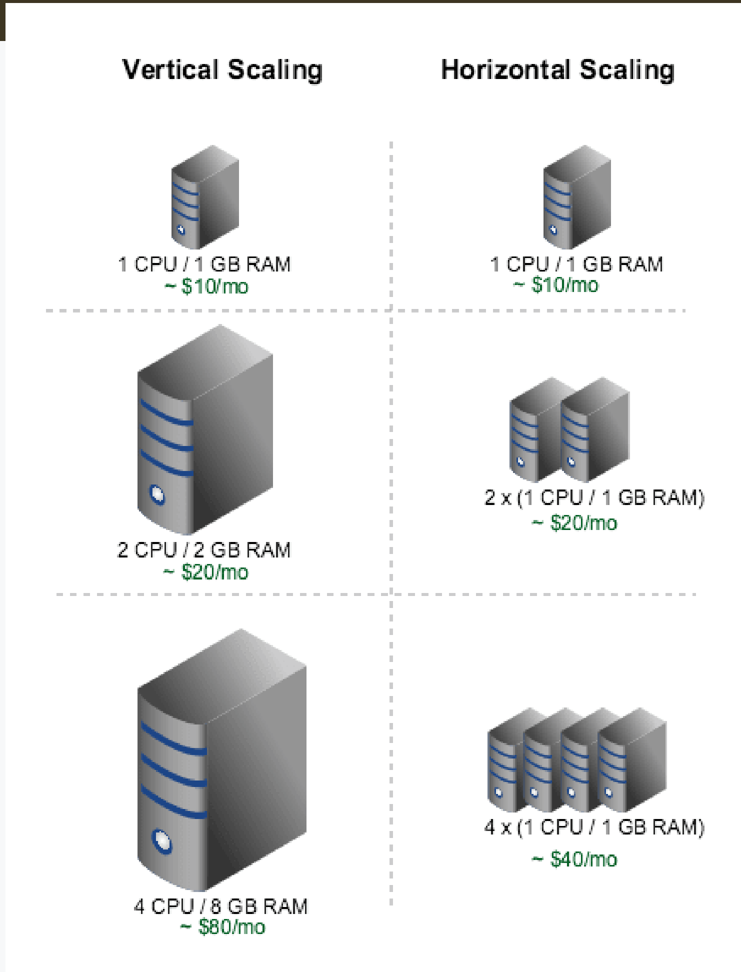
- We'll look at the networking component of blockchains
- How blockchains technically compare to databases
- Compare a blockchain and a database
- Define Consistency, Availability and Partition Tolerance
- Give a high-level overview of Single-leader, multi-leader and distributed hash table

Databases

- Store data!
- Come in all kinds of languages, data models, architectures, …
- In the pre-cloud days sit in the same machine as content host

Scaling databases

- If I wanted to scale my webapp, I can buy a very big computer with a lot of storage (vertical scaling)
- Or I can buy a few more computers and run my webapp on all of them
- Vertical scaling leads to single point of failure
- Horizontal scaling requires replication



Database performance

- Many ways of comparing databases in literature
  - ACID: Atomicity, Consistency, Isolation, Isolation, Durability
  - BASE: Basic-Availability, Soft-state, Eventual Consistency
- We'll be using CAP
  - Consistency, Availability, Partition-Tolerance
  - Not the best, but a good start

Consistency

- Distributed DBs need to handle reads and writes
- Recall *state*; this applies to a distributed DB as well
- When there is a write in one node, a read in another node should return the most up-to-date result

Consistency

- There is also a notion of *eventual* consistency
- If it takes database state across all nodes some time to fully be consistent, it's fine as long as read values return an up-to-date-value

Availability

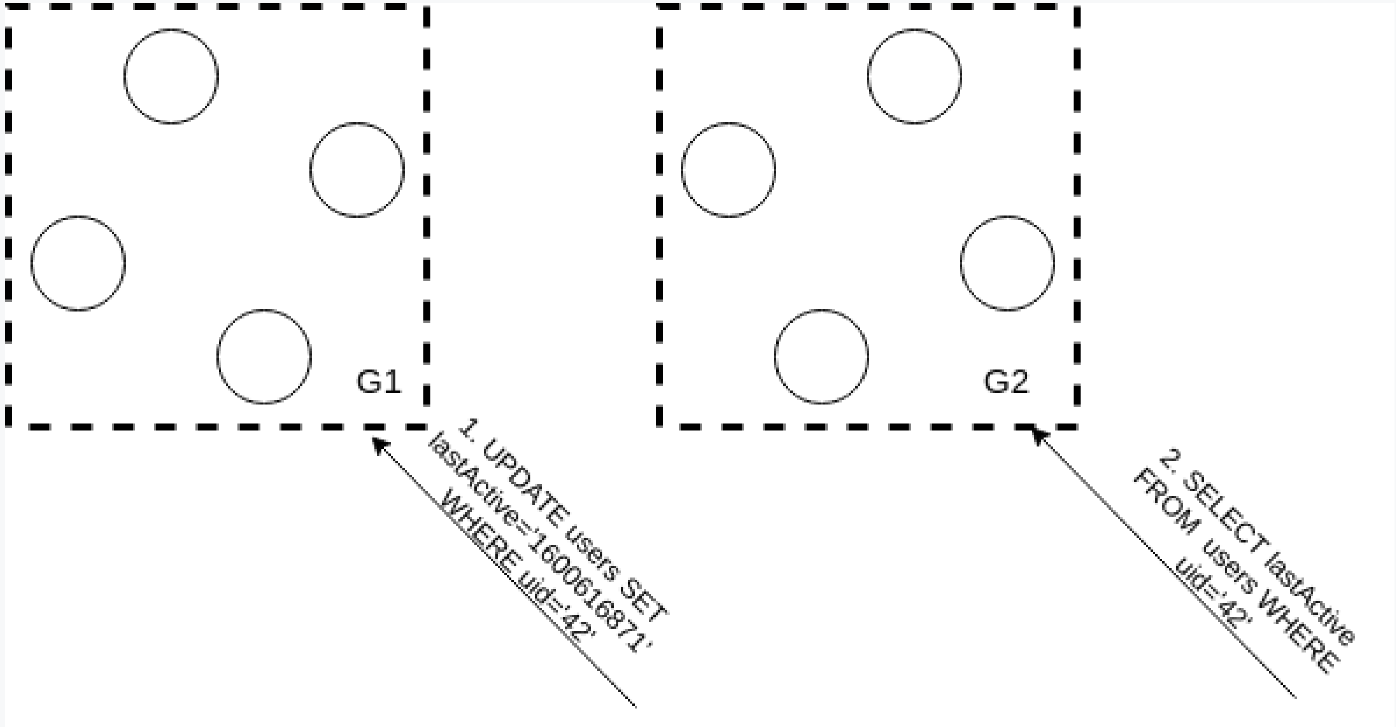
- When a query is made to any node in the network, it must return a non-error response
- This includes when there are node failures
- Delay is fine as long as it eventually gets back

Partition-Tolerance

- Distributed DB needs to be able to operate after being partitioned
- Partitions will have minimal to no communication with each other
- This pretty much always needs to be assumed because of internet infrastructure

CAP Theorem

- CAP Theorem (Brewer, 2000; Gilbert, Lynch, 2002): can't have all 3



Distributed Database Paradigms

- A distributed DB cannot have CAP as there can be two partitions, G1 and G2, of nodes within the system that don’t communicate with each other, and a write query to G1 followed immediately by a read request to G2 will have inconsistent values as G2 will not show the write query to G1
- DBs can thus (**very vaguely**) be classified into AP and CP

Single Leader

- One node is leader accepts writes and propagates them
- All other nodes accept reads
- CP side of CAP

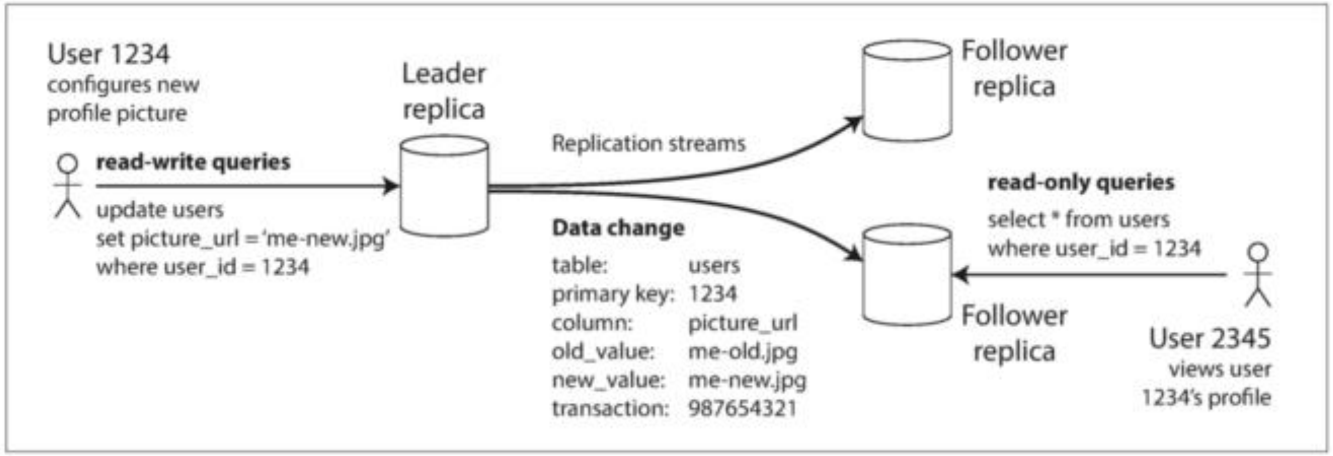


Figure 5-1. Leader-based (master–slave) replication.

Single Leader

- If a follower fails, it can sync when it returns
- If a leader fails, other nodes "elect" a leader (usually based on most up-to-date)

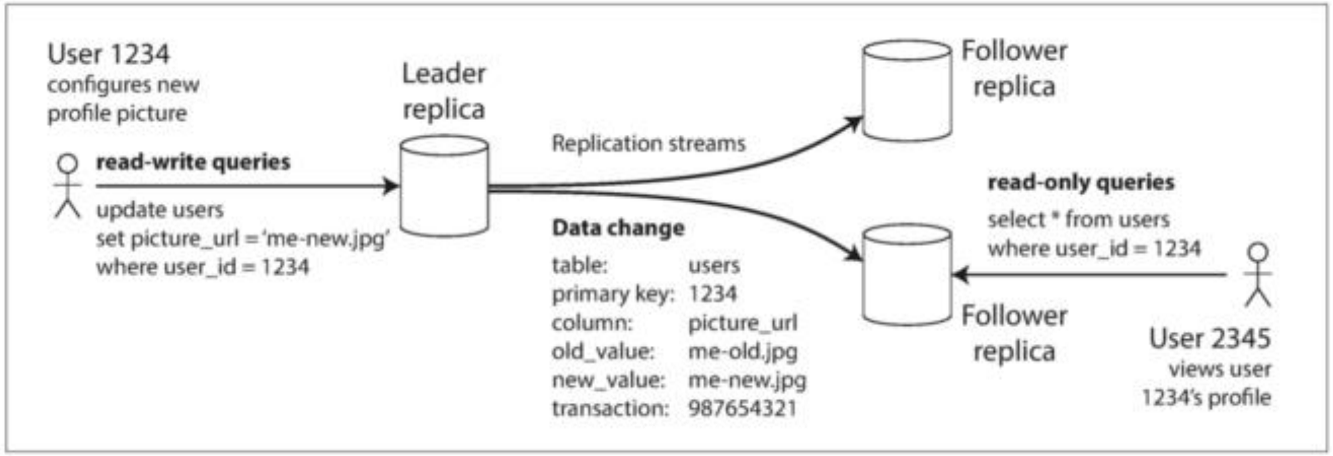


Figure 5-1. Leader-based (master–slave) replication.

Multi Leader

- Many leaders, all accept writes
- However, can have write conflics

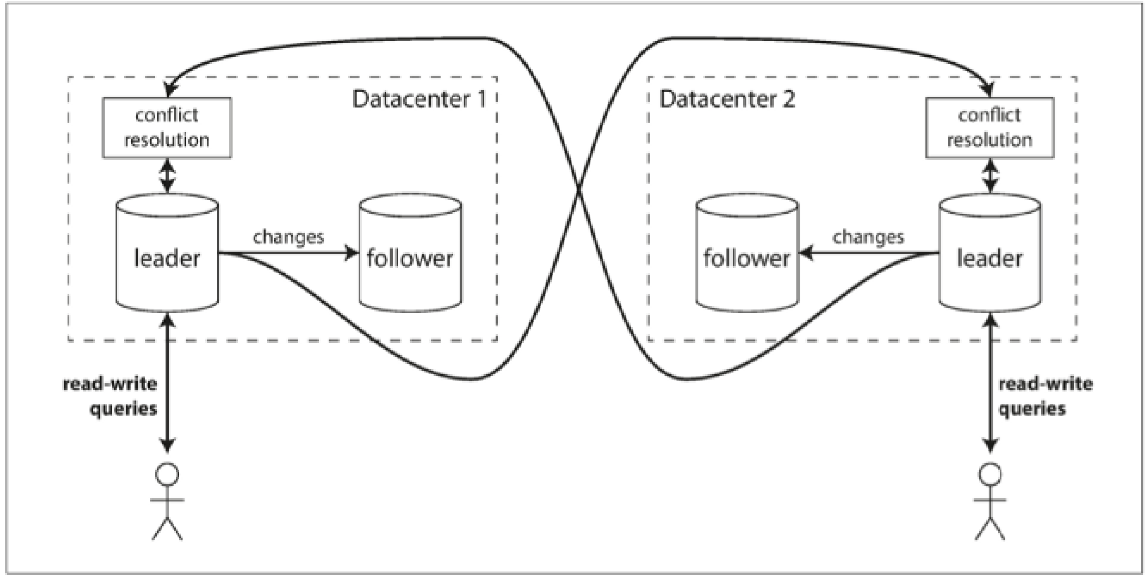


Figure 5-6. Multi-leader replication across multiple datacenters.

Leaderless

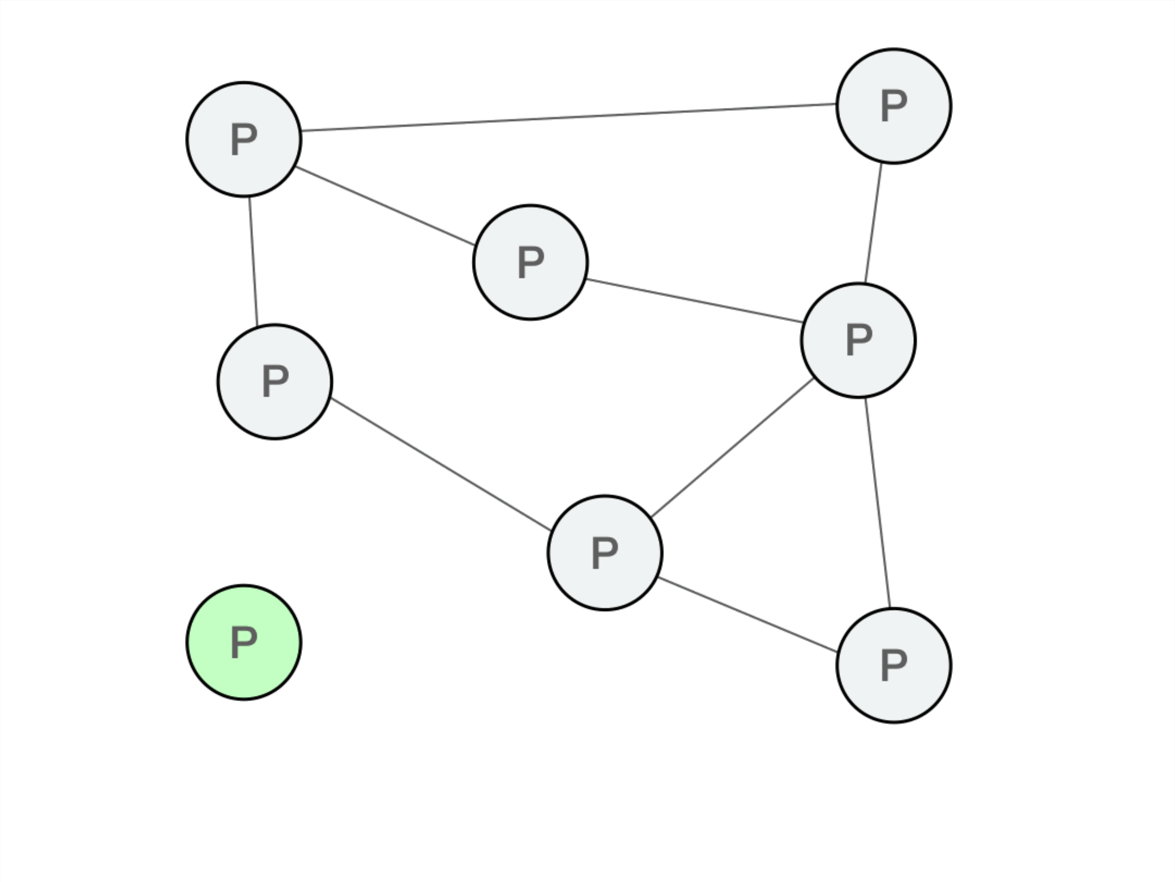
- Dynamo Based System
- P2P system

Dynamo

- Created by Amazon for internal use (!= DynamoDB)
- AP
- All nodes accept reads/writes
- Eventual Consistency
  - Anti-entropy background process
  - Read Repair
  - Quorum: $w+r=n$

P2P

- This is the fun stuff
- Any computer can join the network
- Always require some bootstrap nodes
  - Copy address book
  - Finds own peers after



Distributed Hash Tables

- Peers are arranged in some kind of special topology
- Every peer stores a representation of that topology
  - Knows its own position in topology
  - Has some mechanism for searching for other nodes in there
- Records are stored as K-V pairs
  - Keys are in the same address space as peer addresses
  - This way, no need to have read/wrote consistency instead can simply query keys across the network

Chord

- Circle topology
- Data gets assigned a random key
  - Needs to be collision resistant, SHA
- Can define a *distance* between a key and an address based on >,<,=
- Optimized by a Finger Table
  - Stores the closest node to every power of 2 offset by node's position

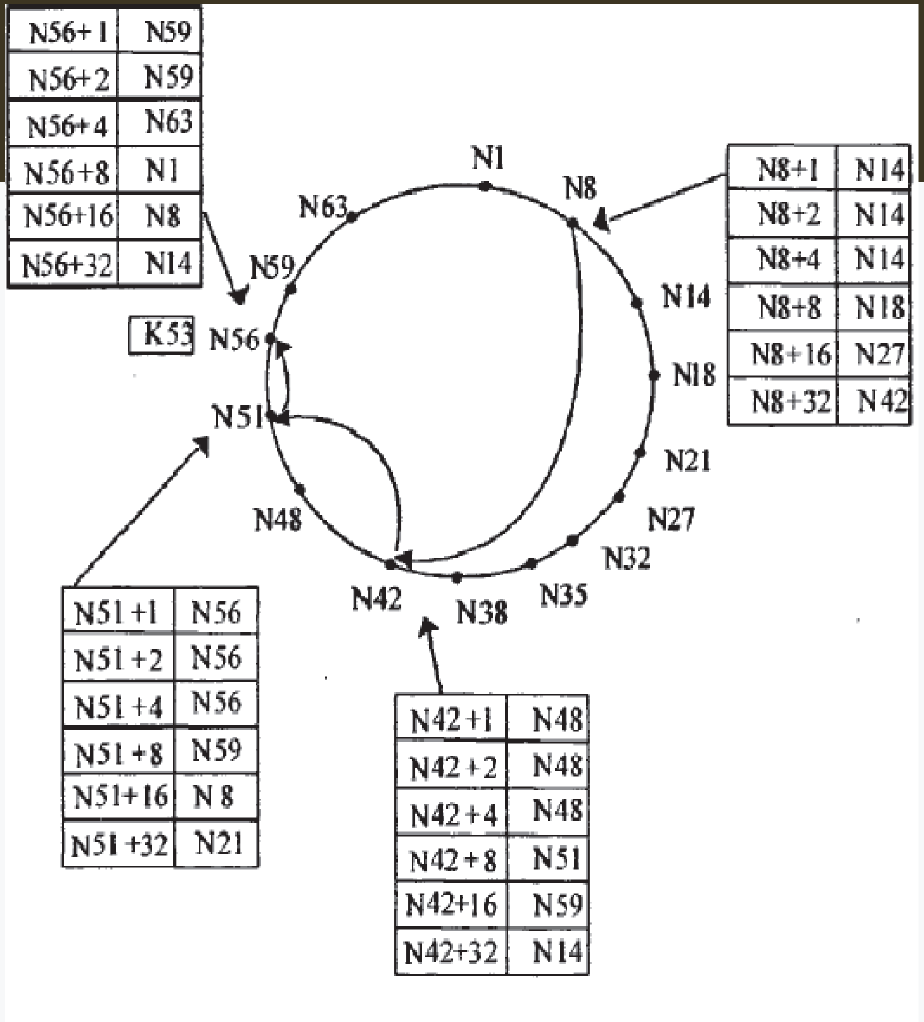
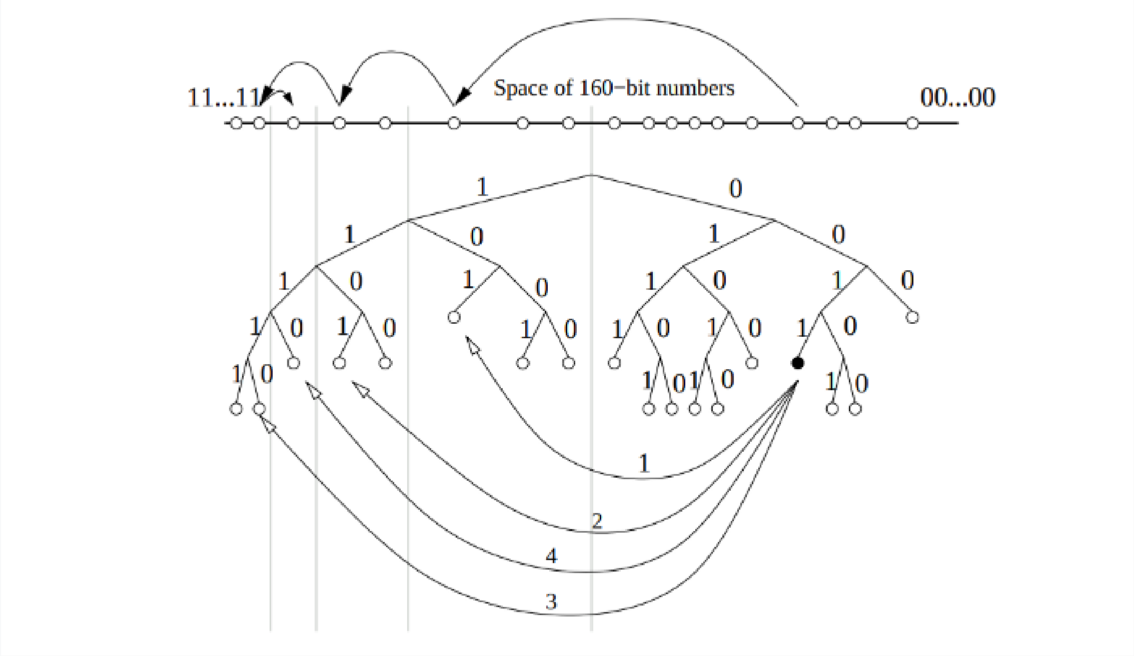


Figure 1. The Chord ring and its finger table

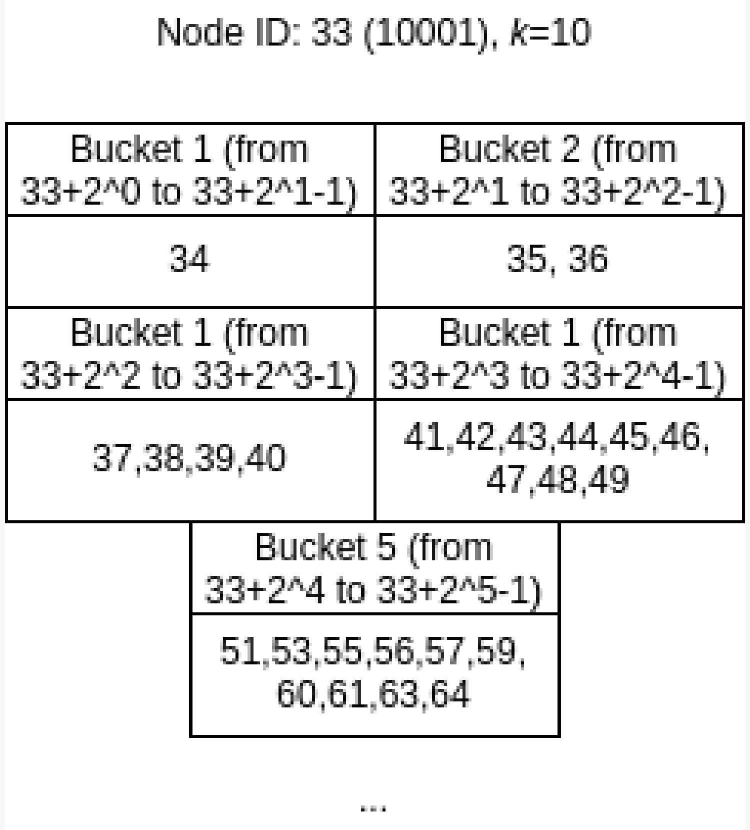
Kademlia

- Uses a binary tree instead of circle
- XOR instead of *successor*
  - Symmetrical
  - Gets the closest common ancestor
- Prefix based addressing
- 4 messages: PING, STORE, FIND_NODE, FIND_VALUE



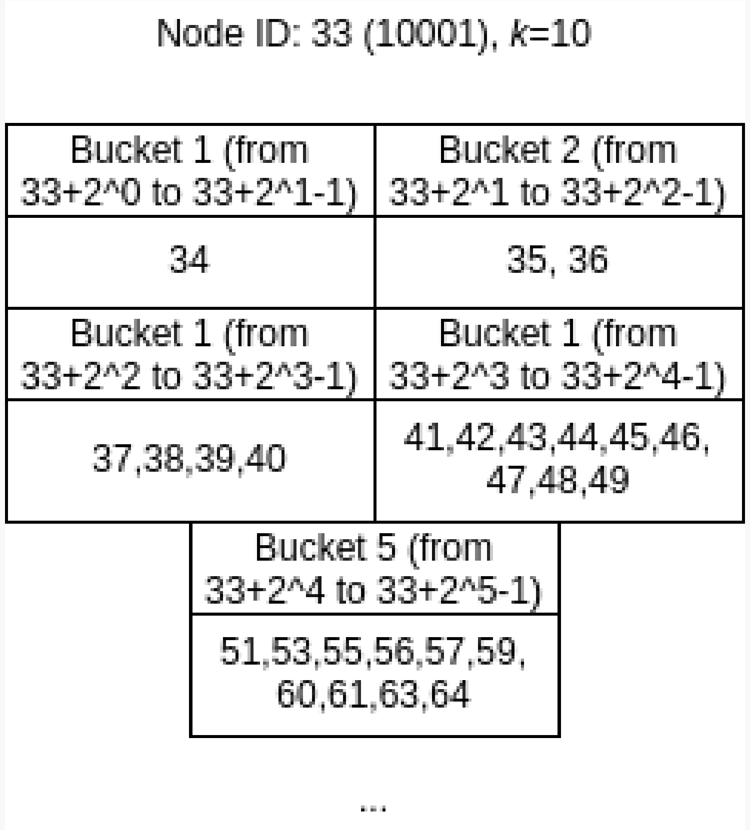
Kademlia

- K-bucket = address book (remember bootstrap)
- Each bucket stores *k* nodes in the range [2^i, 2^{i+1}] far away from it
- Knows a lot about neighbors and just enough about distant peers
- Log(n) hops to match key-val pairs anywhere in network
  - 10000000 nodes would take 20 hops!



Kademlia

- Redundancy parameter, concurrency parameter
- Frequently shares values with neighbors
- More secure considerations than (most) alternatives
  - Self-stabilization
- Battle-tested
  - Bittorrent, Gnutella, IPFS, Eth (kind of)



The Bitcoin Network

- RPC calls
- Denial of service protection
- TCP connections
- By default, 8 outgoing conns
- Supernode: all nodes (about 10k)
- Also has bootstraps
  - Used to use IRC Clients!

Full Client vs Light Client

- Full node isn't always an option...
- Storage and processing requirements are very high
- Light clients are a good alternative as they operate solely on headers (recall block structure)
  - Usually communicate with full node
  - Needs more bandwidth
  - Used by wallets

Minimum Requirements

Bitcoin Core full nodes have certain requirements. If you try running a node on weak hardware, it may work—but you'll likely spend more time dealing with issues. If you can meet the following requirements, you'll have an easy-to-use node.

- Desktop or laptop hardware running recent versions of Windows, Mac OS X, or Linux.
- 350 gigabytes of free disk space, accessible at a minimum read/write speed of 100 MB/s.
- 2 gigabytes of memory (RAM)
- A broadband Internet connection with upload speeds of at least 400 kilobits (50 kilobytes) per second
- An unmetered connection, a connection with high upload limits, or a connection you regularly monitor to ensure it doesn't exceed its upload limits. It's common for full nodes on high-speed connections to use 200 gigabytes upload or more a month. Download usage is around 20 gigabytes a month, plus around an additional 340 gigabytes the first time you start your node.
- 6 hours a day that your full node can be left running. (You can do other things with your computer while running a full node.) More hours would be better, and best of all would be if you can run your node continuously.

Note: many operating systems today (Windows, Mac, and Linux) enter a low-power mode after the screensaver activates, slowing or halting network traffic. This is often the default setting on laptops and on all Mac OS X laptops and desktops. Check your screensaver settings and disable automatic "sleep" or "suspend" options to ensure you support the network whenever your computer is running.

Ethereum: DevP2P

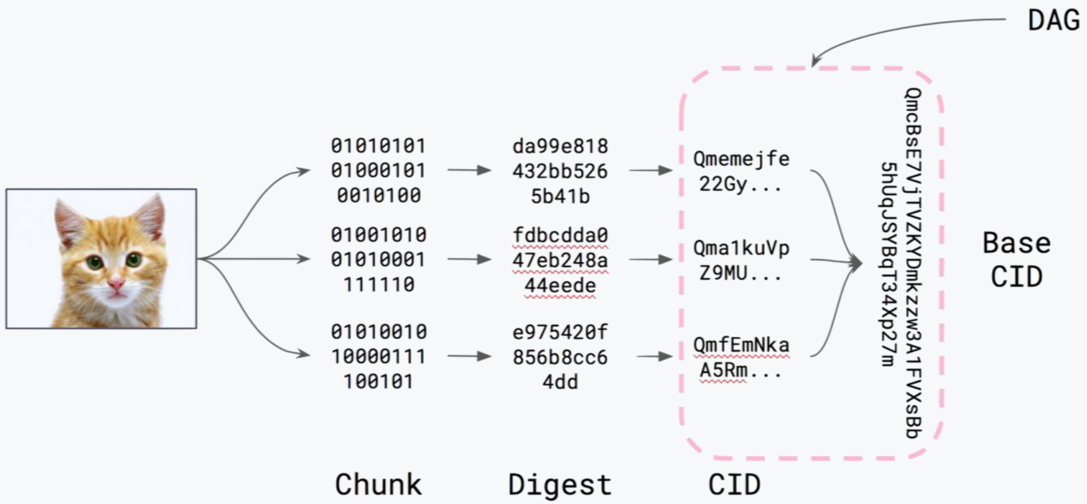
- Uses Kademlia-like DHT
- $distance(n_1, n_2) = keccak256(n_1)\ XOR\ keccak256(n_2)$
- UDP packets
- Nodes maintain an ENR (node record)

IPFS

- InterPlanetary File System
- Developed by Juan Benet ~2015
- Censorship resistant mirrors during multiple political events
- Supported under the hood by libp2p
- Filecoin -> IPFS -> libp2p
- Libp2p has implementations in several languages

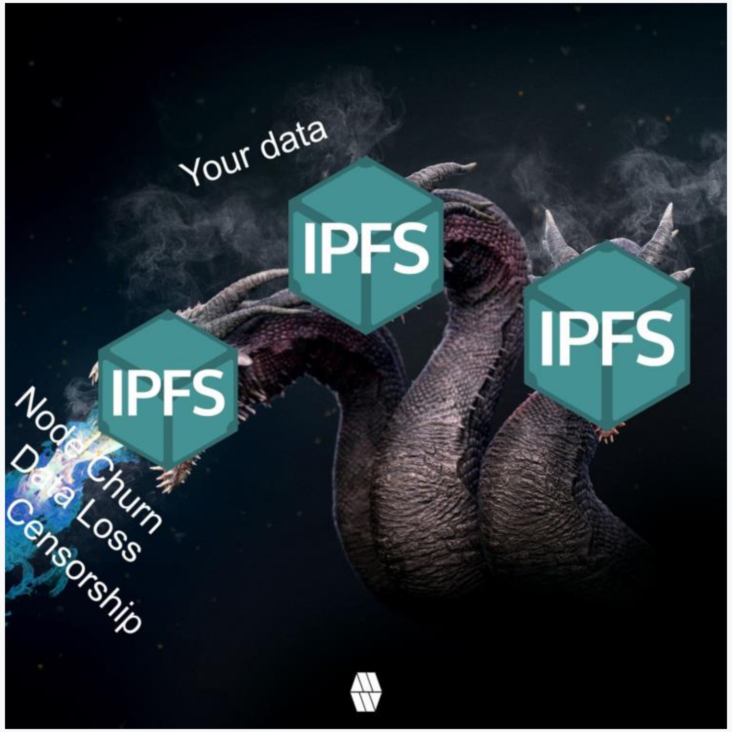
IPFS - MerkleDAG

- Content is chunked (512 bytes) and hashed
- Each hash has some extra information added which represents the CID: Content IDentifier
- Dag.ipfs.io has a visualizer



IPFS Recovery

- "Erasure coding (EC) is a method of data protection in which data is broken into fragments, expanded and encoded with redundant data pieces and stored across a set of different locations or storage media"*
- It is worth consuming some extra storage to obtain better data resiliency and routing performance
- Redundant data can be spread across the network, as hydra heads, for better delivery guarantees and longer lifespan



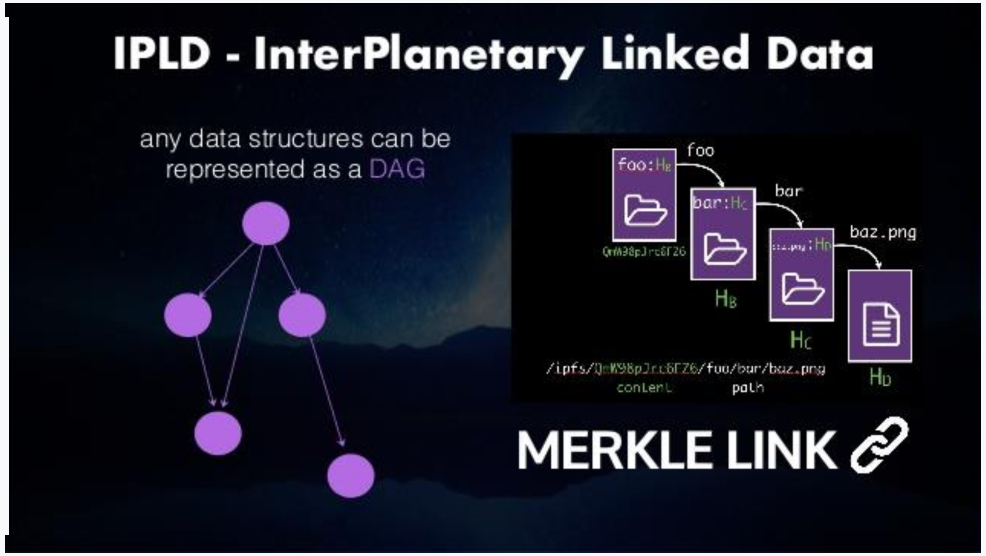
IPLD

- InterPlanetary Linked Data
- "An ecosystem of formats and data structures for building applications that can be fully decentralized"

```javascript
import { encode, decode } from '@ipld/dag-json'
import { CID } from 'multiformats'

const obj = {
  x: 1,
  /* CID instances are encoded as links */
  y: [2, 3, CID.parse('QmaozNR7DZHQK1ZcU9p7QdrshMvXqWK6gpu5rmrkPdT3L4')],
  z: {
    a: CID.parse('QmaozNR7DZHQK1ZcU9p7QdrshMvXqWK6gpu5rmrkPdT3L4'),
    b: null,
    c: 'string'
  }
}

let data = encode(obj)
let decoded = decode(data)
decoded.y[0] // 2
CID.asCID(decoded.z.a) // cid instance
```



Questions/Comments?

Let's code!