Decentralized Systems Engineering

CS-438 – Fall 2024

DEDIS

Pierluca Borsò-Tan and Bryan Ford

EPFL

Credits: C. Arad, ID2210/KTH, Consensys, M. Kleppman

Miscellaneous updates (HW, evals)

- Congrats, most of you scored 100% on HW0 !
- Be sure to work on the correct branch, i.e. “`hw1`” for homework 1 ☺

- HW1 code listing 1 had a mistake, the PDF was updated
- HW1 comes with benchmarks and hidden tests
  - Benchmarks count for 5%
  - Hidden tests for 10%

- Project deadline – this Sunday !

- Class evaluations this week: please participate!
  All (constructive) feedback is welcome, help us improve!

So far...

- Decentralized communication
- Unstructured & structured search
- Can we attack structured search systems?

Finishing up…

Chord DHT – Possible attacks

- Sybil attacks
- **Eclipse attacks**
- Churn attacks
- Adversarial routing
- Denial of Service

So far...

- Decentralized communication
- Unstructured & structured search
- Can we attack structured search systems?
  Yes, but they’re still useful!
- How do we handle the actual underlying data ?

Storing data (reliably)

- Local machine
  - RAID, FEC / ECC / erasure codes, …
- Distributed
  - Block- , filesystem- or object-level access (SAN, NAS, AWS S3)
  - Redundancy
  - Concurrency control
  - Sharding
- Decentralized
  - ??? → today's lecture

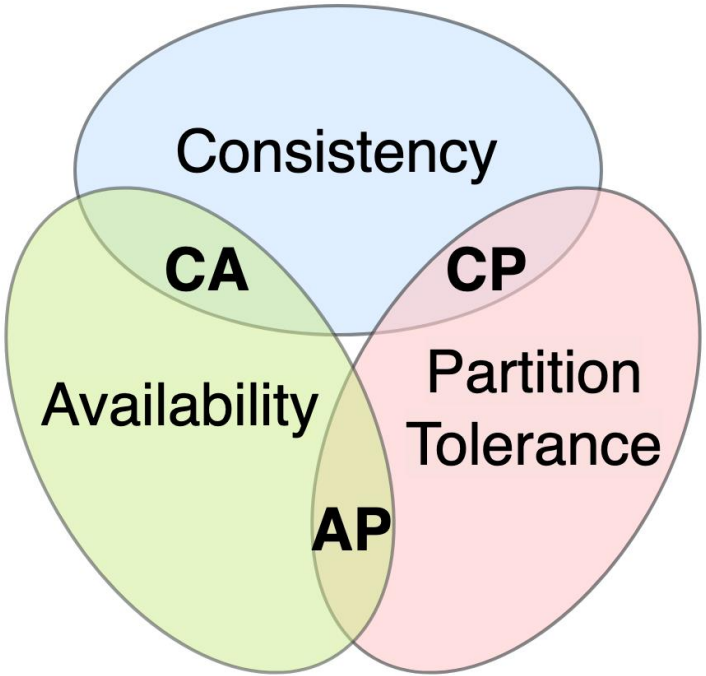
Decentralized Storage & Distribution

BitTorrent, IPFS and CRDTs

(Homework 2)

CAP Theorem – A reminder

a.k.a. Brewer's Theorem



Definitions:

- C = read last write (or error!)
- A = requests get non-error reply
- P = dropped/delayed packets

Trade-offs beyond CAP:

- Granularity of CAP
- *Latency* vs consistency
- Eventual consistency

Storage & distribution – Goals & Challenges (1/2)

- Availability — robust to churn, individual node failures, etc.
- Consistency — how do we stay in sync ? weak ? strong ?
- Scalability, load-balancing — efficiency in bandwidth & storage space
- Modifiability / mutability — (optionally) how do we manage multi-writers ?

Storage & distribution – Goals & Challenges (2/2)

● Malicious security	eclipse, tampering & rollback attacks
● InfoSec (CIA triad)	access control, logging, accountability
● (Logical) data organization	“flat”? files? directories? databases? graph?
● (Physical) data location	where should it be stored?

Building BitTorrent : specifications

- Distribute a large, static (immutable) files

    … from a source node with limited bandwidth

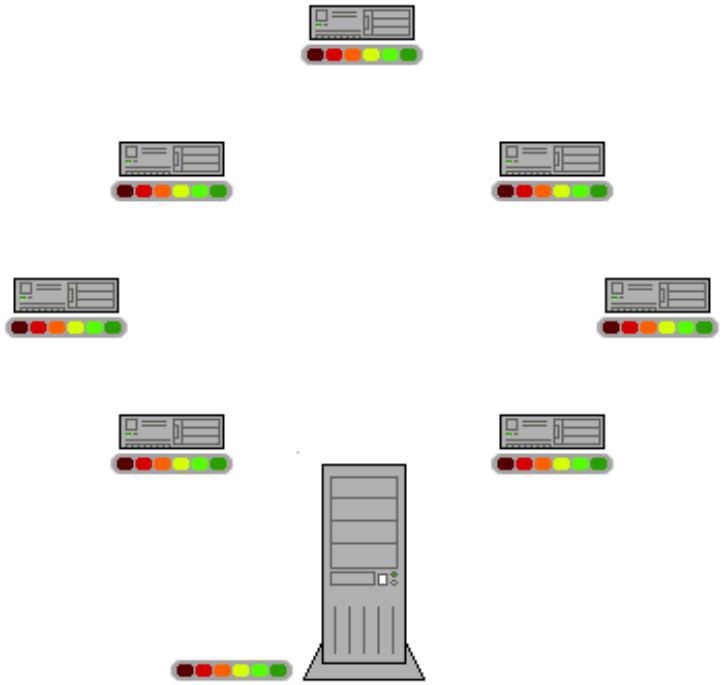
    … to a large number of users

    … as fast as possible !

- Scalable: 22% up- and 3% downstream of global internet traffic (Oct. 2023)
- Assume users are self-interested = don’t assume they want to help

How do we build this ? Core intuition ?

BitTorrent : distribution



Building BitTorrent : Sub-problems

- Advertising a file
- Finding peers to download from
- Verifying integrity of large files (or parts of them)
- Optimizing performance
- Aligning incentives (downloaders vs. uploaders)

Distribution & integrity of (large) files

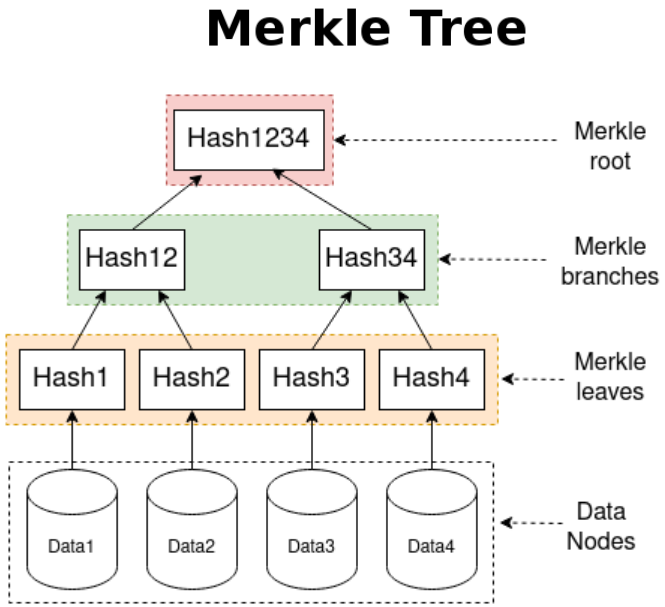
A client should be able to verify:

- parts of a (large) file, as they are downloaded
- the whole file (after download)

Solution ?

→ Chunking

→ Hash tree



BitTorrent : Bootstrapping / finding peers

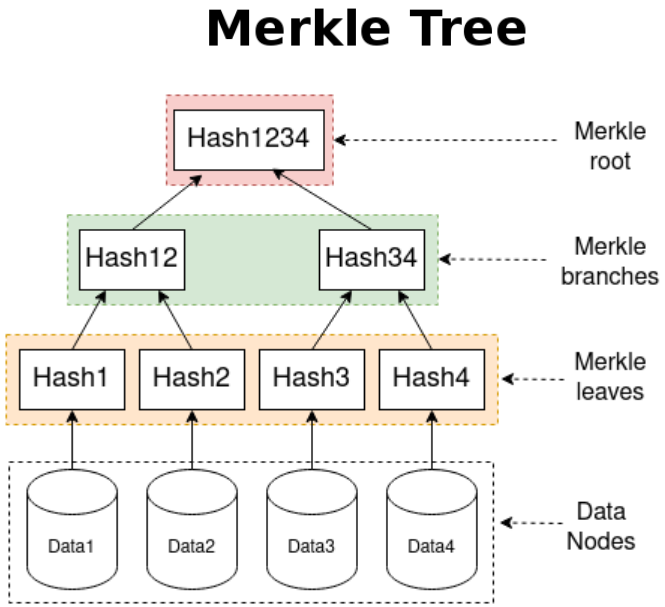
Two options:

- Trackers
- Mainline DHT (based on Kademlia)
  - Key: Merkle root
  - Value: the list of peers having (or downloading) the file

Join the swarm and connect to ~ 80 peers

BitTorrent : Publishing new content

- “Prepare” the file (chunk & build Merkle tree)
- Register with a “tracker”
- Publish a .torrent file or magnet (DHT) link



BitTorrent : Performance & Incentives

- Download rarest data blocks (“chunks”) first – entropy maximization
- Tit-for-Tat strategy (“choking” protocol)

  “chokes” (punishes) peers that are not uploading
  “unchokes” peers with the highest upload rates
  “optimistic unchoking” looks for better/bootstrapping peers

- Make the download rate proportional to the upload rate for each peer

BitTorrent-inspired solutions

Servers

- Twitter's "Murder" server deployment system
- Facebook's

Games

- "Blizzard Downloader" - World of Warcraft, Diablo III
- Wargaming's - World of Tanks, World of Warplanes, etc.

OS

- Windows Update
- Others have tried … and failed (e.g. "DebTorrent")

- Interplanetary Filesystem (IPFS)

BitTorrent limits

Why did DebTorrent fail to materialize ?

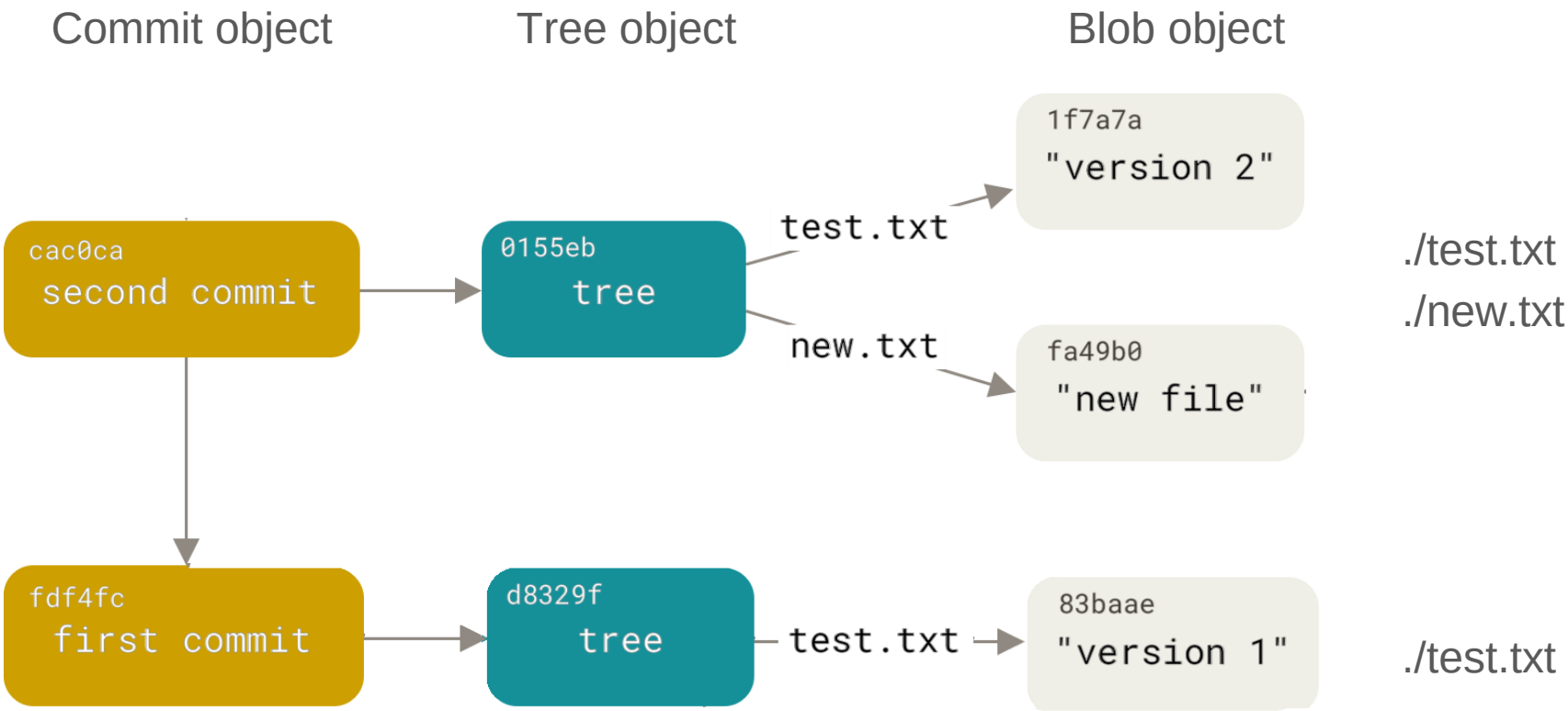
- Ill-suited for small files (overhead)
- Ill-suited for sharing overlapping sets of data (across torrents)
- Data is immutable (and not re-usable across torrents)
- Locality of peers is ignored – ISPs do traffic-shaping

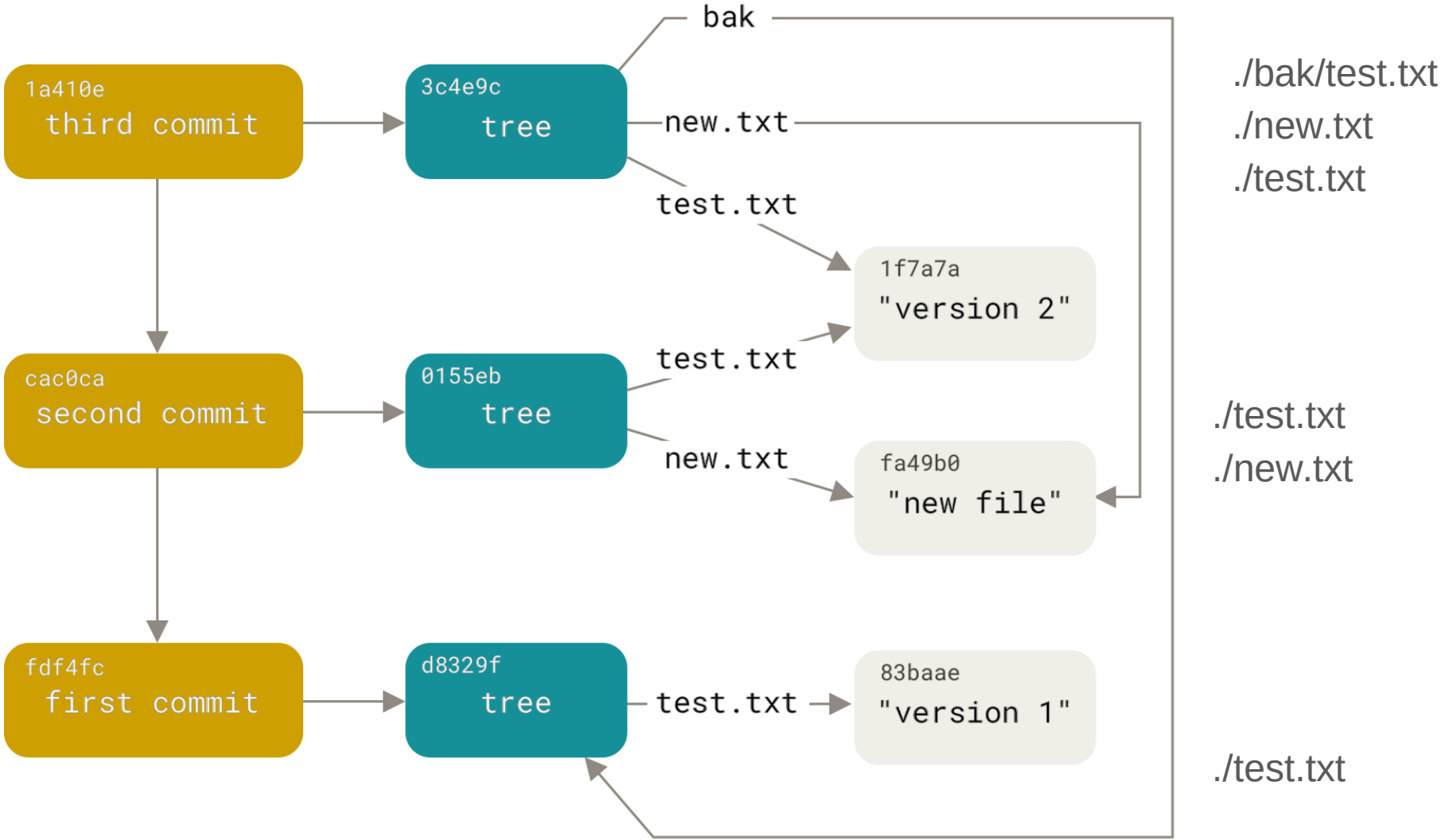
For DebTorrent, both “1 huge torrent” or “1M+ small ones” = inefficient

IPFS – Inter-Planetary File System

- Protocol for P2P distributed *file system*, fully decentralized
- Designed to address (perceived) flaws in HTTP
- Deployed at massive scale
  2024: >70k servers, millions of unique weekly users, 23 EiB capacity (2 stored)
- A decentralized file system inspired by:
  - Kademlia DHT
  - BitTorrent – block exchange
  - Git versioning
  - Self-certifying filesystems

Reminder: Git object database




1a410e
third commit
3c4e9c
tree
bak
new.txt
test.txt
1f7a7a
"version 2"
./bak/test.txt
./new.txt
./test.txt
cac0ca
second commit
0155eb
tree
test.txt
new.txt
fa49b0
"new file"
./test.txt
./new.txt
fdf4fc
first commit
d8329f
tree
test.txt
83baae
"version 1"
./test.txt

IPFS: Representing a filesystem in a DHT

- Everything is immutable
- All objects are self-certifying (files, links, folders, changes)
  ID is computed based on object's hash
- Any IPFS object (file, folder) is represented in the same way:

```
type IPFSObject struct {

    // array of links
    links []IPFSLink

    // opaque content data
    data []byte
}
```

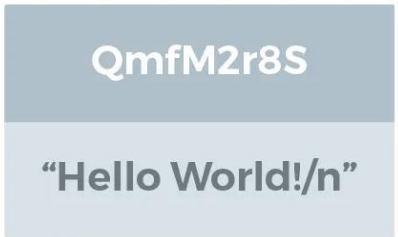
```
type IPFSLink struct {

    Name string    // target's name

    Hash Multihash // ... hash

    Size int       // ... size
}
```

IPFS: Representing a file < 256kB

Also known as:

a blob !

```
{
“Links”: [],
“Data”: “\u0008\u0002\u0012\rHello World!\n\u0018\r”
}
```

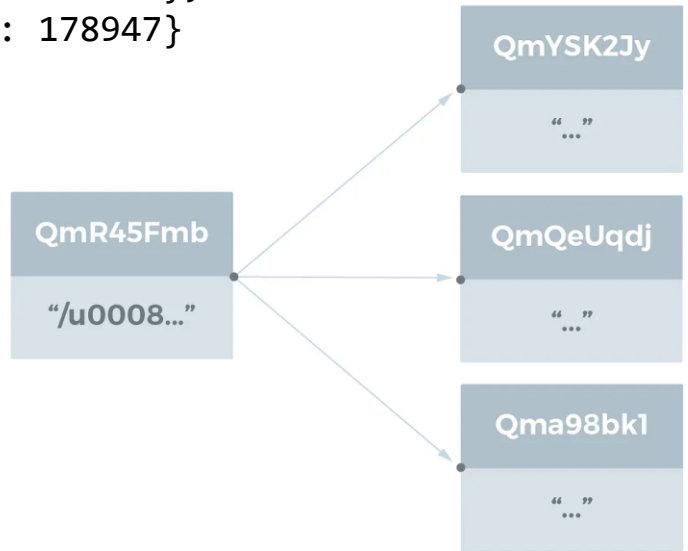


IPFS: Representing a file > 256kB

Also known as:

a list !

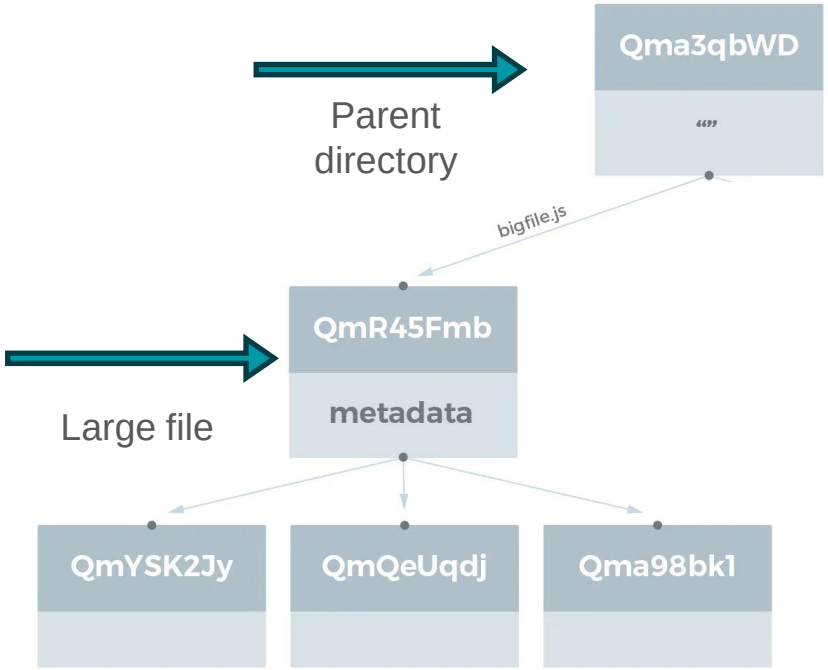
```
{
“Links”: [
          {“Name”: “”,“Hash”: “QmYSK2Jy...”,“Size”: 262158},
          {“Name”: “”,“Hash”: “QmQeUqdj...”,“Size”: 262158},
          {“Name”: “”,“Hash”: “Qma98bk1...”,“Size”: 178947}
],
“Data”: “\u0008\u0002\u0018* \u0010 \u0010 \n”
}
```



IPFS: representing a directory

Also known as:

a tree !



./bigfile.js
./my_dir/my_file.txt
./my_dir/Testing.txt
./hello.txt

e
e
locations)

IPFS: Versioning

Also known as:

a commit !

- Git-like
- Build a Merkle DAG (Directed Acyclic Graph)
- Build a “snapshot” of the current state
- Hash of both content and its parent commit’s hash
- Creates a Git-like log of versions

IPFS: Naming (mutable) data

Objects are immutable, so:

- Use a separate namespace for mutable data
- Use mutable, signed *pointers* to immutable data
- Not content-addressable: advertise link on routing system
- Built-in limit to rollback attacks

Shifting Paradigm: Local-First Software

What do these 3 applications have in common ?

- Git
- Google Docs
- Apple Notes

They work offline and you (nearly) get the full experience !

How?

- Multi-version concurrency control
  i.e. how do you “merge” versions that forked ?

Local-First Software

Goals

- Local client is first-class citizen
- Works offline
- Eventual consistency
- Ideally: can handle forks

What tool do we need to make this happen?

- Conflict-Free Replicate Data Types

  Data structure + Algorithm + Protocol

Conflict-Free Replicated Data Types (CRDTs)

Various types:

- Values
- Counters
- Sets
- Lists
- Log-based
- Text

Two main categories:

- Operation-based – commutative replicated data types (CmRDTs)
- State-based – convergent replicate data types (CvRDTs)

→ Theoretically equivalent

State-based CRDT – Formalism

Let U be the set of update operations, and V the set of values.

A state-based CRDT is a 5-tuple (S, s^0, q, u, m), where:

- S is the set of states;
- $s^0 \in S$ is the initial state;

- $q : S \rightarrow V$ is the **query** function
- $u : S \times U \rightarrow S$ is the **update** function
- $m : S \times S \rightarrow S$ is the **merge** function

Next steps - Readings

Mandatory:

- Incentives Build Robustness in BitTorrent
- Ivy: A Read/Write Peer-to-Peer File System

Recommended (Engineering):

- IPFS: Content Addressed, Versioned, P2P file system
- Peritext: A CRDT for Rich-Text Collaboration

... and a few others for the curious among you ...

→ Use Friday’s session to ask questions