IPFS - InterPlanetary File System

-- liuqian@cpchain.io

Contents - Background

- Distributed Hash Tables
- Block Exchanges - BitTorrent
- Version Control Systems - Git
- Self-Certified Filesystems - SFS

Contents - IPFS Design

Background - Distributed Hash Tables

What?

- A distributed system that provides a lookup service similar to a hash table.

Why?

- Autonomy and decentralization
- Fault tolerance
- Scalability

Background - Distributed Hash Tables

How?

- Keyspace Partitioning
    - e.g. A node with ID Ix owns all the keys Km for which Ix is the closest ID, measured according to Distance(Ix, Km).
- Overlay Network
    - Each node maintains a set of links to other nodes.
    - A node picks its neighbors according to a certain structure, called the network's topology.
    - For any key k, each node either has a node ID that owns k or has a link to a node whose node ID is closer to k.
    - It is then easy to route a message to the owner of any key k using greedy algorithm.

Background - Distributed Hash Tables

- *Performance*

Max Degree	Max Route Length	Note
O(1)	O(N)	worst lookup length
O(logN)	O(logN)	common choice
O(rootN)	O(1)	worst local storage needs

Background - Distributed Hash Tables

- *Real World Implementations*
- Kademlia and others
    - Wiki
    - Introduction

Background - Distributed Hash Tables

When searching for some value, the algorithm needs to know the associated key and explores the network in several steps.
Each step will find nodes that are closer to the key until the contacted node returns the value or no more closer nodes are found.

This is very efficient: Like many other DHTs, Kademlia contacts only O(log(n)) nodes during the search out of a total of n nodes in the system.

Background - Distributed Hash Tables

- *Real World Implementations*
  - Ethereum
  - IPFS
- The DHT is used in IPFS for routing, in other words:
  - to announce added data to the network
  - and help locate data that is requested by any node.

Background - Block Exchanges - BitTorrent

- Introduction to BitTorrent
  - Peer-to-peer networking with BitTorrent
  - Peer to Peer Content Delivery - BitTorrent, slides
  - The BitTorrent Protocol Specification, V2

Background - Block Exchanges - BitTorrent

With IPFS, the white paper mentions two BitTorrent features that IPFS uses:

- tit-for-tat strategy (if you don’t share, you won’t receive either)
- get rare pieces first (improves performance and more, see the first PDF above)
- BitSwap
    - ...

Background - Version Control System - Git - Summary

- Version Control Systems provide facilities to model files changing over time and distribute different versions efficiently.
- The popular version control system Git provides a powerful Merkle DAG object model that captures changes to a filesystem tree in a distributed-friendly way.

Background - Version Control System - Git - Features

1. Immutable objects represent Files (blob), Directories (tree), and Changes (commit).
2. Objects are content-addressed, by the cryptographic hash of their contents.
3. Links to other objects are embedded, forming a Merkle DAG. This provides many useful integrity and workflow properties.
4. Most versioning metadata (branches, tags, etc.) are simply pointer references, and thus inexpensive to create and update.
5. Version changes only update references or add objects.
6. Distributing version changes to other users is simply transferring objects and updating remote references.

Background - Version Control System - Git - Features

Know More

- Merkle Tree
- Git Internals for Curious Developers for more information.
- Merkle DAG
- Git for Computer Scientists

Background - Self-Certified Filesystems - SFS

This is used to implement the IPNS name system for IPFS. It allows us to generate an address for a remote filesystem, where the user can verify the validity of the address.

SFS introduced a technique for building Self-Certified Filesystems: addressing remote filesystems using the following scheme

```
/sfs/<Location>:<HostID>
```

where Location is the server network address, and:

```
HostID = hash(public_key || Location)
```

Thus the name of an SFS file system certifies its server.

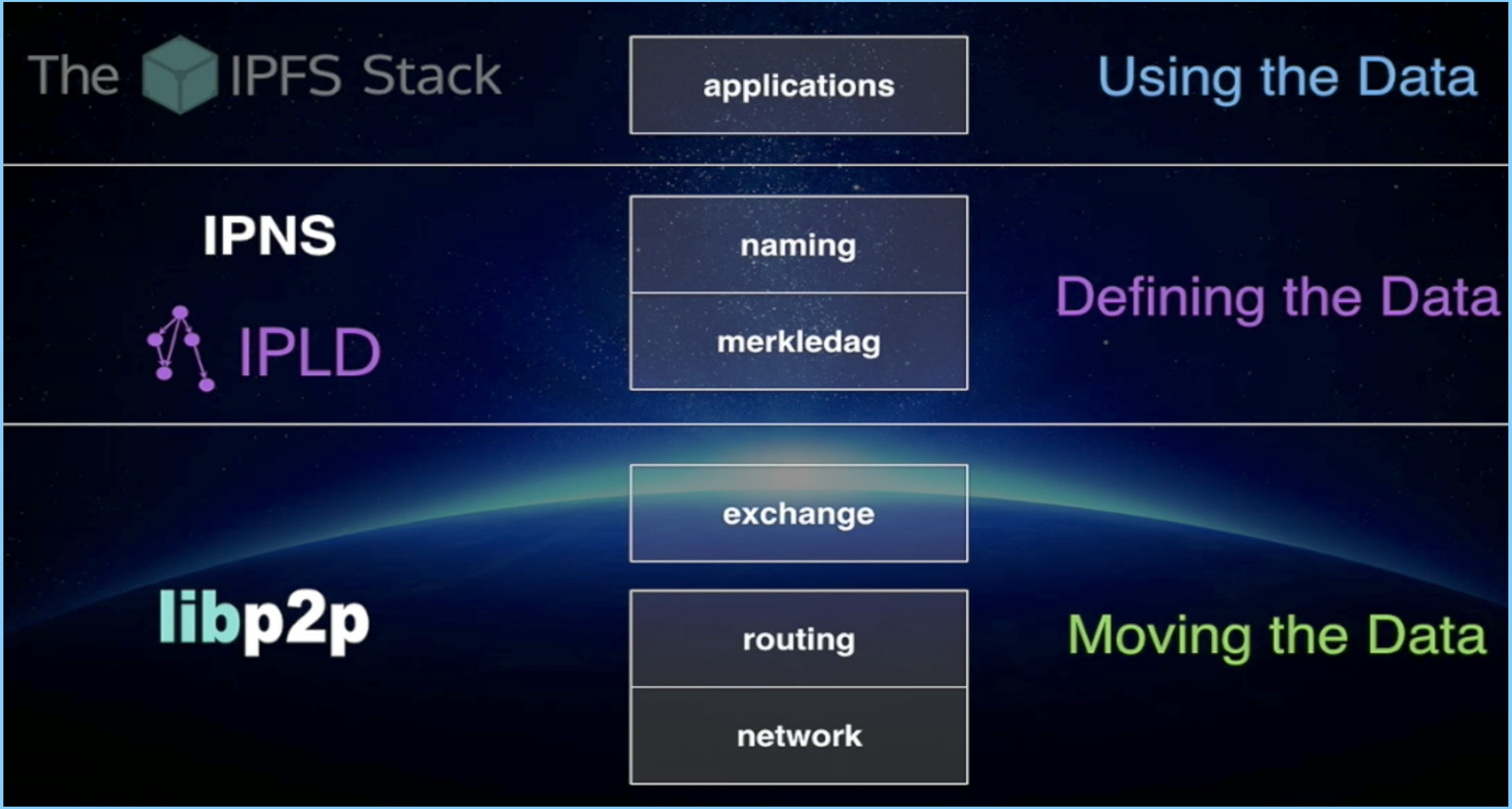
IPFS Design

1. Identities - manage node identity generation and verification.
2. Network - manages connections to other peers, uses various underlying network protocols.
3. Routing - maintains information to locate specific peers and objects.
4. Exchange - a novel block exchange protocol (BitSwap) that governs efficient block distribution.
5. Objects - a Merkle DAG of content-addressed immutable objects with links.
6. Files - versioned file system hierarchy inspired by Git.
7. Naming - A self-certifying mutable name system.

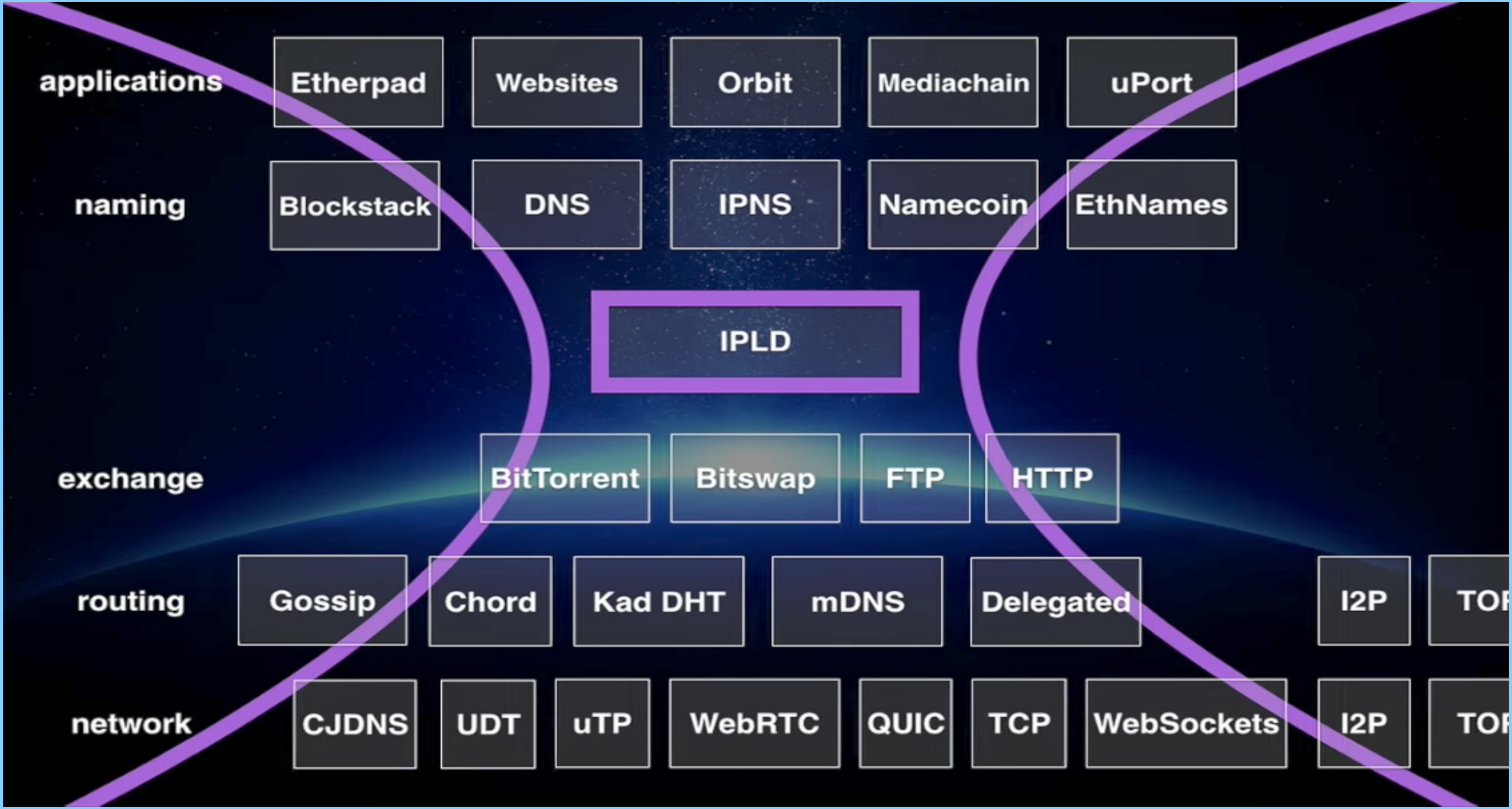
IPFS Design

- Identities: name those nodes
- Network: talk to other clients
- Routing: announce and find stuff
- Exchange: give and take
- Objects: organize the data
- Files: uh?
- Naming: adding mutability

IPFS Design


The IPFS Stack
applications
Using the Data
IPNS
IPLD
naming
merkledag
Defining the Data
exchange
libp2p
routing
network
Moving the Data

IPFS Design


applications
Etherpad
Websites
Orbit
Mediachain
uPort
naming
Blockstack
DNS
IPNS
Namecoin
EthNames
IPLD
exchange
BitTorrent
Bitswap
FTP
HTTP
routing
Gossip
Chord
Kad DHT
mDNS
Delegated
I2P
TOR
network
CJDNS
UDT
uTP
WebRTC
QUIC
TCP
WebSockets
I2P
TOR

Identities: name those nodes

- Its goal:
    - **manage node identity generation and verification**

When Preparing

- generate a PKI key pair (public + private key)
- hash the public key
- the resulting hash is the NodeId

Identities: name those nodes

When Handshaking

- exchange public keys
- check if: hash(other.PublicKey) == other.NodeId
- if so, we have identified the other node and can e.g. request for data objects
- if not, we disconnect from the "fake" node

Identities: name those nodes

More Details:

- IPFS favors self-describing values.
- For hash: `multihash` format, including the hash function and the hash bytes.

Like this `<function code><digest length><digest bytes>`.

- This allows the system to
  - choose the best function for the use case (e.g. stronger security vs faster performance)
  - evolve as function choices change.
- Self-describing values allow using different parameter choices compatibly.
  Check this multiformats.

Identities: name those nodes

```go
type NodeId Multihash

// self-describing cryptographic hash digest
type Multihash []byte

type PublicKey []byte

// self-describing keys
type PrivateKey []byte

type Node struct {
    NodeId NodeID
    PubKey PublicKey
    PriKey PrivateKey
}
```

Identities: name those nodes

S/Kademlia based IPFS identity generation:

```
n = Node{}
difficulty = <integer parameter>

for p < difficulty {
    n.PubKey, n.PrivKey = PKI.genKeyPair()
    n.NodeId = hash(n.PubKey)
    p = count_preceding_zero_bits(hash(n.NodeId))
}
```

Network: talk to other clients

- Goal:
    - **manages connections to other peers**
    - **uses various underlying network protocols**
    - **configurable**
- IPFS works on tops of any network, see multiaddr

  For example:

```
# an SCTP/IPv4 connection
/ip4/10.20.30.40/sctp/1234/

# an SCTP/IPv4 connection proxied over TCP/IPv4
/ip4/5.6.7.8/tcp/5678/ip4/1.2.3.4/sctp/1234/
```

Network: talk to other clients

- Features:
    - Transport: IPFS can use any transport protocol, and is best suited for WebRTC DataChannels (for browser connectivity) or uTP(LEDBAT).
    - Reliability: IPFS can provide reliability if underlying networks do not provide it, using uTP (LEDBAT) or SCTP.
    - Connectivity: IPFS also uses the ICE NAT traversal techniques.
    - Integrity: optionally checks integrity of messages using a hash checksum.
    - Authenticity: optionally checks authenticity of messages using HMAC with sender’s public key.

Routing: announce and find stuff

- Goal and features:
  - maintains information to locate specific peers and objects
  - responds to both local and remote queries.
  - defaults to a DHT, but is swappable.

Routing: announce and find stuff

- It is based on DHT and its purpose is to:
    - announce that this node has some data (a block as discussed in the next chapter), or
    - find which nodes have some specific data (by referring to the multihash of a block), and
    - if the data is small enough (=< 1KB) the DHT stores the data as its value.

Routing: announce and find stuff

```
type IPFSRouting interface {
    // gets a particular peer’s network address
    FindPeer(node NodeId)

    // stores a small metadata value in DHT
    SetValue(key []bytes, value []bytes)

    // retrieves small metadata value from DHT
    GetValue(key []bytes)

    // announces this node can serve a large value
    ProvideValue(key Multihash)

    // gets a number of peers serving a large value
    FindValuePeers(key Multihash, min int)
}
```

Routing: announce and find stuff

Note:

- Different use cases will call for substantially different routing systems (e.g. DHT in wide network, static HT in local network).
- Thus the IPFS routing system can be swapped for one that fits users' needs.
- As long as the interface above is met, the rest of the system will continue to function.

Exchange: give and take Block

- Data is broken up into `blocks`
- Not like BitTorrent, it's BitSwap
- BitSwap Protocol
  - A novel block exchange protocol that governs efficient block distribution.
  - Modelled as a market, weakly incentivizes data replication.
  - Trade strategies swappable.

Exchange: give and take Block

- BitSwap is modeled as a marketplace that incentivizes data replication.
- The BitSwap Strategy, also can be replaced by another strategy
- Check FileCoin for another strategy.

Exchange: give and take Block

- The general working process:
    - when peers connect, they exchange which blocks they have (have_list) and which blocks they are looking for (want_list)
    - to decide if a node will actually share data, it will apply its BitSwap Strategy
    - this strategy is based on previous data exchanges between these two peers
    - when peers exchange blocks they keep track of the amount of data they share (builds credit) and the amount of data they receive (builds debt)

Exchange: give and take Block

- The general working process:
  - this accounting between two peers is kept track of in the BitSwap Ledger
  - if a peer has credit (shared more than received), our node will send the requested block
  - if a peer has debt, our node will share or not share, depending on a deterministic function where the chance of sharing becomes smaller when the debt is bigger
  - a data exchange always starts with the exchange of the ledger, if it is not identical our node disconnects

IPFS Design

The three layers of the stack we described so far (network, routing, exchange) are implemented in libp2p.

Objects: organize the data

- Object Merkle DAG
    - A Merkle DAG of content-addressed immutable objects with links.
    - Used to represent arbitrary data structures, e.g. file hierarchies and communication systems.

Objects: organize the data

To create any data structure, IPFS offers a flexible and powerful solution:

- organize the data in a graph, where we call the nodes of the graph objects
- these objects can contain data (any sort of data, transparent to IPFS) and/or links to other objects
- these links - Merkle Links - are simply the cryptographic hash of the target object

Objects: organize the data

This way of organizing data has a couple of useful properties (quoting from the white paper):

- Content Addressing: all content is uniquely identified by its multihash checksum, including links.
- Tamper resistance: all content is verified with its checksum. If data is tampered with or corrupted, IPFS detects it.
- Deduplication: all objects that hold the exact same content are equal, and only stored once. This is particularly useful with index objects, such as git trees and commits, or common portions of data.

Objects: organize the data

The IPFS Object format is:

```
type IPFSLink struct {
    // name or alias of this link
    Name string

    // cryptographic hash of target
    Hash Multihash

    // total size of target
    Size int
}
type IPFSObject struct {
    // array of links
    links []IPFSLink

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

Objects: organize the data

- List all object references in an object. For example:

```
ipfs ls /XLZ1625Jjn7SubMDgEyeaynFuR84ginqvzb
# XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x 189458 less
# XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5 19441 script
# XLF4hwVHsVuZ78FZK6fozf8Jj9WEURMbCX4 5286 template
# <object multihash> <object size> <link name>
```

Objects: organize the data

- Resolve string path lookups, such as `foo/bar/baz`. Given an object
    - IPFS resolves the first path component to a hash in the object’s link table
    - Then fetches that second object, and repeats with the next component.
    - Thus, string paths can walk the Merkle DAG no matter what the data formats are.

Objects: organize the data

- Resolve all objects referenced recursively:

```
> ipfs refs --recursive XLZ1625Jjn7SubMDgEyeaynFuR84ginqvzb
# XLLxhdgJcXzLbtsLRL1twCHA2NrURp4H38s
# XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x
# XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5
# XLWVQDqxo9Km9zLyquoC9gAP8CL1gWnHZ7z
...
```

Objects: organize the data - Paths

```
# format
/ipfs/<hash-of-object>/<name-path-to-object>
# example
/ipfs/XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x/foo.txt
```

Note this means that given three objects in path `<foo>/bar/baz`, the last object is accessible by all:

```
/ipfs/<hash-of-foo>/bar/baz
/ipfs/<hash-of-bar>/baz
/ipfs/<hash-of-baz>
```

Objects: organize the data - Local Storage

- IPFS clients require some local storage, an external system on which to store and retrieve local raw data for the objects IPFS manages.
- Ultimately, all blocks available in IPFS are in some node’s local storage. When users request objects, they are found, downloaded, and stored locally, at least temporarily. This provides fast lookup for some configurable amount of time thereafter.

Objects: organize the data - Build More

While it is easy to see how the Git object model fits on top of this DAG, consider these other potential data structures:

- key-value stores
- Traditional relational databases
- Linked Data triple stores
- Linked document publishing systems
- Linked communications platforms
- Cryptocurrency blockchains.

  These can all be modeled on top of the IPFS Merkle DAG, which allows any of these systems to use IPFS as a transport protocol for more complex applications.

Objects: organize the data - Object Pinning

- Nodes who wish to ensure the survival of particular objects can do so by pinning the objects.
- This ensures the objects are kept in the node’s local storage.
- Pinning can be done recursively, to pin down all linked descendent objects as well.
- All objects pointed to are then stored locally.
- This is particularly useful to persist files, including references.
- This also makes IPFS a Web where links are permanent, and Objects can ensure the survival of others they point to.

Objects: organize the data - Object-level Cryptography

```go
type EncryptedObject struct {
    // raw object data encrypted
    Object []bytes

    // optional tag for encryption groups
    Tag []bytes
}
type SignedObject struct {
    // raw object data signed
    Object []bytes

    // hmac signature
    Signature []bytes

    // multihash identifying key
    PublicKey []multihash
}
```

Now it's WIP.

Objects: organize the data - Know More

- Check this link for a detailed description.
- Intermission: IPLD and its implementation, a video about it.

Files - versioned file system hierarchy inspired by Git

On top of the Merkle DAG objects IPFS defines a Git-like file system with versioning, with the following elements:

- blob: there is just data in blobs and it represents the concept of a file in IPFS. No links in blobs
- list: lists are also a representation of an IPFS file, but consisting of multiple blobs and/or lists
- tree: a collection of blobs, lists and/or trees: acts as a directory
- commit: a snapshot of the history in a tree (just like a git commit).

Files - versioned file system hierarchy inspired by Git - File Object: blob

- The blob object contains an addressable unit of data, and represents a file.
- IPFS Blocks are like Git blobs or filesystem data blocks.
- They store the users’ data.
- Blobs have no links.

For example:

```
{
    // blobs have no links
    "data": "some data here",
}
```

Note that IPFS files can be represented by both lists and blobs.

Files - versioned file system hierarchy inspired by Git - File Object: list

- `lists` contains an ordered sequence of `blob` or `list` objects.

```
{
    // lists have an array of object types as data
    "data": ["blob", "list", "blob"],

    // lists have no names in links
    "links": [
        { "hash": "XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x", "size": 189458 },
        { "hash": "XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5", "size": 19441 },
        { "hash": "XLWVQDqxo9Km9zLyquoC9gAP8CL1gWnHZ7z", "size": 5286 }
    ]
}
```

Files - versioned file system hierarchy inspired by Git - File Object: tree

- A `tree` object represents a directory, a map of names to hashes.
- The hashes reference `blobs` , `lists` , other `trees` , or `commits` .

Files - versioned file system hierarchy inspired by Git - File Object: tree

```
{
    // trees have an array of object types as data
    "data": ["blob", "list", "blob"],

    // trees do have names
    "links": [
        { "hash": "XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x", "name": "less", "size": 189458 },
        { "hash": "XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5", "name": "script", "size": 19441 },
        { "hash": "XLWVQDqxo9Km9zLyquoC9gAP8CL1gWnHZ7z", "name": "template", "size": 5286 }
    ]
}
```

Files - versioned file system hierarchy inspired by Git - File Object: commit

- The commit object in IPFS represents a snapshot in the version history of any object.
- It is similar to Git’s, but can reference any type of object.
- It also links to author objects.

Files - versioned file system hierarchy inspired by Git - File Object: commit

```
{
    "data": {
        "type": "tree",
        "date": "2014-09-20 12:44:06Z",
        "message": "This is a commit message."
    },
    "links": [
        { "hash": "XLa1qMBKiSEEDhojb9FFZ4tEvLf7FEQdhdU", "name": "parent", "size": 25309 },
        { "hash": "XLGw74KAy9junbh28x7ccWov9inu1Vo7pnX", "name": "object", "size": 5198 },
        { "hash": "XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm", "name": "author", "size": 109 }
    ]
}
```

Files - versioned file system hierarchy inspired by Git - File Object: Version control

- The commit object represents a particular snapshot in the version history of an object.
- Comparing the objects (and children) of two different commits reveals the differences between two versions of the filesystem.
- As long as a single commit and all the children objects it references are accessible, all preceding versions are retrievable and the full history of the filesystem changes can be accessed.
- This falls out of the Merkle DAG object model.

Files - versioned file system hierarchy inspired by Git - File Object: Filesystem Paths

- IPFS objects can be traversed with a string path API.
- The IPFS File Objects are designed to make mounting IPFS onto a UNIX filesystem simpler.
- They restrict trees to have no data, in order to represent them as directories.
- And commits can either be represented as directories or hidden from the filesystem entirely.

Files - versioned file system hierarchy inspired by Git - File Object: Splitting Files into Lists and Blob

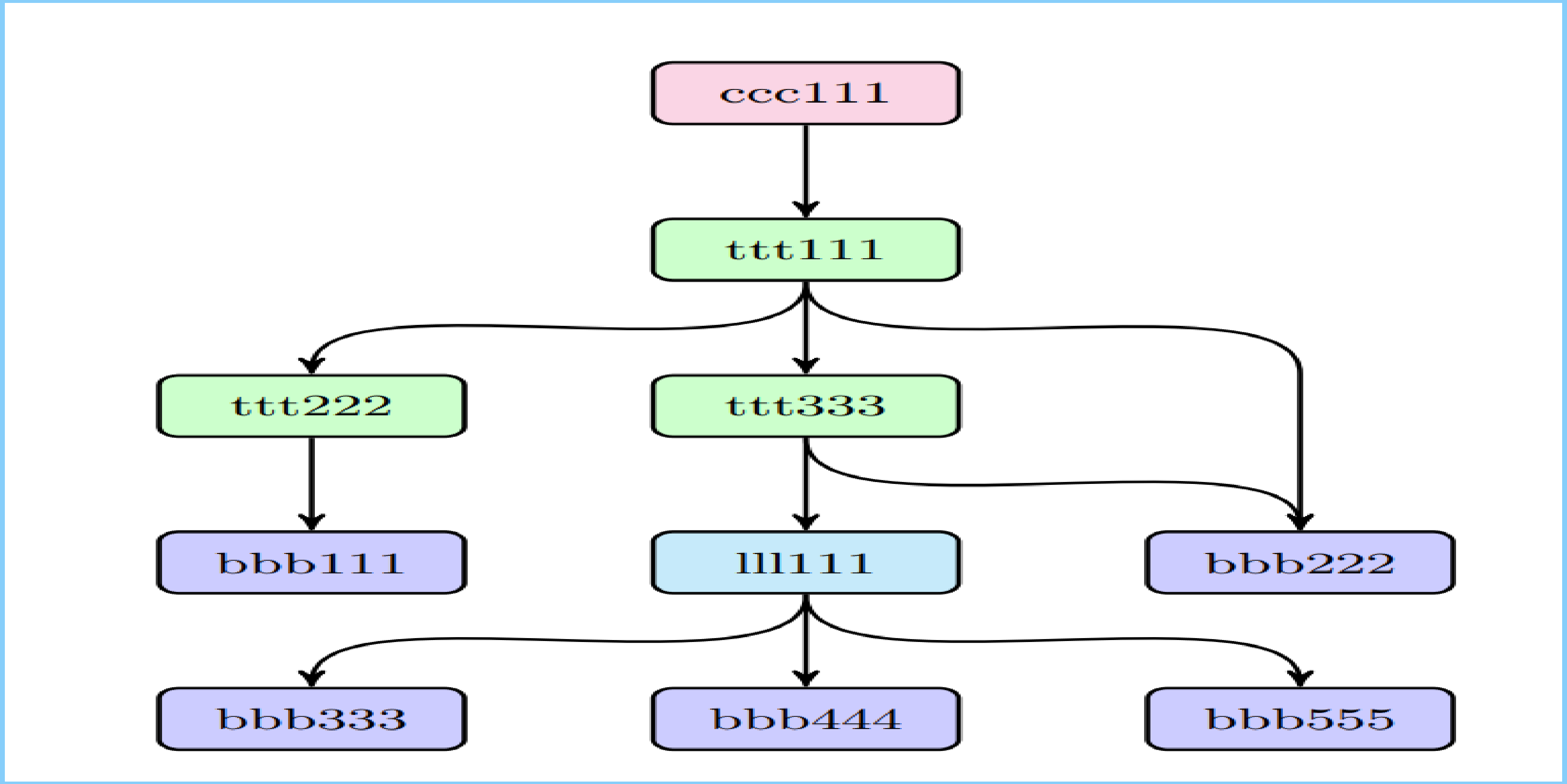
The Main Challenge:

- Finding the right way to split large files into independent blocks.

Rather than assume it can make the right decision for every type of file, IPFS offers the following alternatives:

- Use Rabin Fingerprints as in LBFS(Last Buffer First Served) to pick suitable block boundaries.
- Use the rsync rolling-checksum algorithm, to detect blocks that have changed between versions.
- Allow users to specify block-splitting functions highly tuned for specific files.

Files - versioned file system hierarchy inspired by Git - File Object: An Example



Files - versioned file system hierarchy inspired by Git - File Object: An Example

```
> ipfs file-cat <ccc111-hash> --json
{
    "data": {
        "type": "tree",
        "date": "2014-09-20 12:44:06Z",
        "message": "This is a commit message."
    },
    "links": [
        { "hash": "<ccc000-hash>", "name": "parent", "size": 25309 },
        { "hash": "<ttt111-hash>", "name": "object", "size": 5198 },
        { "hash": "<aaa111-hash>", "name": "author", "size": 109 }
    ]
}
```

Files - versioned file system hierarchy inspired by Git - File Object: An Example

```
> ipfs file-cat <ttt111-hash> --json
{
    "data": ["tree", "tree", "blob"],
    "links": [
        { "hash": "<ttt222-hash>", "name": "ttt222-name", "size": 1234 },
        { "hash": "<ttt333-hash>", "name": "ttt333-name", "size": 3456 },
        { "hash": "<bbb222-hash>", "name": "bbb222-name", "size": 22 }
    ]
}
```

Files - versioned file system hierarchy inspired by Git - File Object: An Example

```
> ipfs file-cat <bbb222-hash> --json
{
    "data": "blob222 data",
    "links": []
}
```

Files - versioned file system hierarchy inspired by Git - File Object: Path Lookup Performance

- Path-based access traverses the object graph.
- Retrieving each object requires looking up its key in the DHT, connecting to peers, and retrieving its blocks.
- This is considerable overhead, particularly when looking up paths with many components.

Files - versioned file system hierarchy inspired by Git - File Object: Path Lookup Performance

This is mitigated by:

- tree caching: since all objects are hash-addressed, they can be cached indefinitely. Additionally, trees tend to be small in size so IPFS prioritizes caching them over blobs.
- flattened trees: for any given tree, a special flattened tree can be constructed to list all objects reachable from the tree. Names in the flattened tree would really be paths parting from the original tree, with slashes.

Files - versioned file system hierarchy inspired by Git - File Object: Path Lookup Performance

For example, flattened tree for ttt111 above:

```
{
    "data": ["tree", "blob", "tree", "list", "blob" "blob"],
    "links": [
        { "hash": "<ttt222-hash>", "size": 1234, "name": "ttt222-name" },
        { "hash": "<bbb111-hash>", "size": 123, "name": "ttt222-name/bbb111-name" },
        { "hash": "<ttt333-hash>", "size": 3456, "name": "ttt333-name" },
        { "hash": "<lll111-hash>", "size": 587, "name": "ttt333-name/lll111-name"},
        { "hash": "<bbb222-hash>", "size": 22, "name": "ttt333-name/lll111-name/bbb222-name" },
        { "hash": "<bbb222-hash>", "size": 22, "name": "bbb222-name" }
    ]
}
```

IPNS: Naming and Mutable State - A self-certifying mutable name system.

- Since links in IPFS are content addressable (a cryptographic hash over the content represents the block or object of content), data is immutable by definition. It can only be replaced by another version of the content, and it, therefore, gets a new "address".
- The solution is to create "labels" or "pointers" (just like git branches and tags) to immutable content. These labels can be used to represent the latest version of an object (or graph of objects).

IPNS: Naming and Mutable State

In IPFS this pointer can be created using the Self-Certified Filesystems. It is named IPNS and works like this:

- The root address of a node is `/ipns/<NodeId>`
- The content it points to can be changed by publishing an IPFS object to this address
- By publishing, the owner of the node (the person who knows the secret key that was generated with ipfs init) cryptographically signs this "pointer".
- This enables other users to verify the authenticity of the object published by the owner.
- Just like IPFS paths, IPNS paths also start with a hash, followed by a Unix-like path.
- IPNS records are announced and resolved via the DHT.

Use Cases

1. As a mounted global filesystem, under `/ipfs` and `/ipns`.
2. As a mounted personal sync folder that automatically versions, publishes, and backs up any writes.
3. As an encrypted file or data sharing system.
4. As a versioned package manager for all software.
5. As the root filesystem of a Virtual Machine.
6. As the boot filesystem of a VM (under a hypervisor).

Use Cases

7. As a database: applications can write directly to the Merkle DAG data model and get all the versioning, caching, and distribution IPFS provides.
8. As a linked (and encrypted) communications platform.
9. As an integrity checked CDN for large files (without SSL).
10. As an encrypted CDN.
11. On webpages, as a web CDN.
12. As a new Permanent Web where links do not die.

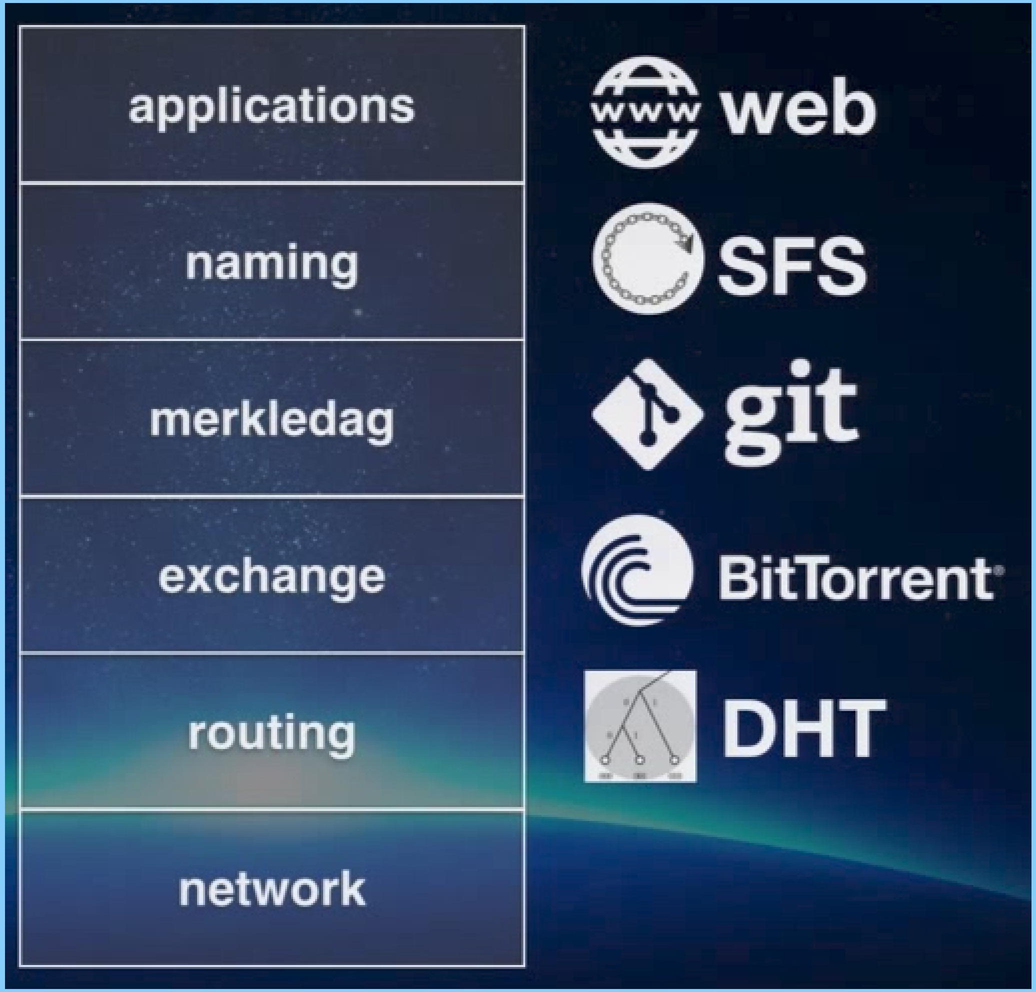
The IPFS Implementations Target

- an IPFS library to import in your own applications.
- commandline tools to manipulate objects directly.
- mounted file systems, using FUSE or as kernel modules.

Review - Background

- Distributed Hash Tables
- Block Exchanges - BitTorrent
- Version Control Systems - Git
- Self-Certified Filesystems - SFS

Review


applications
web
naming
SFS
merkledag
git
exchange
BitTorrent
routing
DHT
network

Review - IPFS Design

- Identities: name those nodes
- Network: talk to other clients
- Routing: announce and find stuff
- Exchange: give and take
- Objects: organize the data
- Files: uh?
- Naming: adding mutability

Know More

- Quick Start Video
- Tutorial
- Github Repository
- White Paper
- Detailed Reading List

Thank You

Using IPFS

```
# download ipfs installation package
wget https://dist.ipfs.io/go-ipfs/v0.4.21/go-ipfs_v0.4.21_linux-amd64.tar.gz

# unzip the package
tar -vxzf go-ipfs_v0.4.21_linux-amd64.tar.gz

# install it to some path
cd go-ipfs && sudo ./install.sh
```

Using IPFS

```
# create a playground directory
mkdir x && ...
tree
# x/
# ├── bar
# │   ├── bar.txt
# │   ├── bax.txt
# │   └── baz.txt
# └── foo
#     └── foo.txt
#
# 2 directories, 4 files
```

Using IPFS

```
# init
ipfs init

# add
ipfs add -r x
# added QmbFMke1KXqnYyBBWxB74N4c5SBnJMVAiMNRcGu6x1AwQH x/bar/bar.txt
# added QmcNWfP4SFXje7nZDPiT9Y46tvxStvzAoGpyQdYU3CB92W x/bar/bax.txt
# added QmT78zSuBmuS4z925WZfrqQ1qHaJ56DQaTfyMUF7F8ff5o x/bar/baz.txt
# added QmT78zSuBmuS4z925WZfrqQ1qHaJ56DQaTfyMUF7F8ff5o x/foo/foo.txt
# added QmdKr2Dvn3yKeBoX6UU85AQ7GdfY9sW4N8DUPmhp5JeRXF x/bar
# added QmR2RTyLMEYKUdAKjG6jA74VzpVdDgFEJxrj4iL4nGWyWi x/foo
# added Qmd4pWC75EE9awSegDZxZfNqSsq1YXrcKzNzrAsuBYTne1 x
# 40 B / 40 B [=======================================] 100.00%
```

Using IPFS

```
# go to another directory and fetch the data
mkdir y && cd y

# get
ipfs get Qmd4pWC75EE9awSegDZxZfNqSsq1YXrcKzNzrAsuBYTne1
tree
# y
# └── Qmd4pWC75EE9awSegDZxZfNqSsq1YXrcKzNzrAsuBYTne1
#     ├── bar
#     │   ├── bar.txt
#     │   ├── bax.txt
#     │   └── baz.txt
#     └── foo
#         └── foo.txt
#
# 3 directories, 4 files
```

Using IPFS

```
# cat
ipfs cat /ipfs/QmT78zSuBmuS4z925WZfrqQ1qHaJ56DQaTfyMUF7F8ff5o
# hello world

# ls
ipfs ls /ipfs/Qmd4pWC75EE9awSegDZxZfNqSsq1YXrcKzNzrAsuBYTne1
# QmdKr2Dvn3yKeBoX6UU85AQ7GdfY9sW4N8DUPmhp5JeRXF - bar/
# QmR2RTyLMEYKUdAKjG6jA74VzpVdDgFEJxrj4iL4nGWyWi - foo/

# refs
ipfs refs /ipfs/Qmd4pWC75EE9awSegDZxZfNqSsq1YXrcKzNzrAsuBYTne1
# QmdKr2Dvn3yKeBoX6UU85AQ7GdfY9sW4N8DUPmhp5JeRXF
# QmR2RTyLMEYKUdAKjG6jA74VzpVdDgFEJxrj4iL4nGWyWi
```

Using IPFS

```
# block
# block get
ipfs block get QmcNWfP4SFXje7nZDPiT9Y46tvxStvzAoGpyQdYU3CB92W
#
# 你好, 世界

# block stat
ipfs block stat QmcNWfP4SFXje7nZDPiT9Y46tvxStvzAoGpyQdYU3CB92W
# Key: QmcNWfP4SFXje7nZDPiT9Y46tvxStvzAoGpyQdYU3CB92W
# Size: 24

# object data
ipfs object data QmcNWfP4SFXje7nZDPiT9Y46tvxStvzAoGpyQdYU3CB92W
# 你好, 世界
```

Using IPFS

```
# object object
ipfs object get QmcNWfP4SFXje7nZDPiT9Y46tvxStvzAoGpyQdYU3CB92W
# {"Links":[],"Data":"\u0008\u0002\u0012\u0010你好，世界\n\u0018\u0010"}

ipfs object get Qmd4pWC75EE9awSegDZxZfNqSsq1YXrcKzNzrAsuBYTne1
# {"Links":[{"Name":"bar","Hash":"QmdKr2Dvn3yKeBoX6UU85AQ7GdfY9sW4N8DUPmhp5JeRXF","Size":201},{"Name":"foo","Hash":"QmR2RTyLMEYKUdAKjG6jA74VzpVdDgFEJxrj4iL4nGWyWi","Size":73}],"Data":"\u0008\u0001"}

ipfs object links Qmd4pWC75EE9awSegDZxZfNqSsq1YXrcKzNzrAsuBYTne1
# QmdKr2Dvn3yKeBoX6UU85AQ7GdfY9sW4N8DUPmhp5JeRXF 201 bar
# QmR2RTyLMEYKUdAKjG6jA74VzpVdDgFEJxrj4iL4nGWyWi 73  foo
```

Using IPFS

```
ipfs object stat Qmd4pWC75EE9awSegDZxZfNqSsq1YXrcKzNzrAsuBYTne1
# NumLinks:       2
# BlockSize:      95
# LinksSize:      93
# DataSize:       2
# CumulativeSize: 369

# ipfs dag
ipfs dag get QmcNWfP4SFXje7nZDPiT9Y46tvxStvzAoGpyQdYU3CB92W
# {"data":"CAISEOS9oOWlve+8jOS4lueVjAoYEA==","links":[]}
```

Using IPFS

```
# ipfs dag
ipfs dag get QmcNWfP4SFXje7nZDPiT9Y46tvxStvzAoGpyQdYU3CB92W
# {"data":"CAISEOS9oOWlve+8jOS4lueVjAoYEA==","links":[]}

ipfs dag get Qmd4pWC75EE9awSegDZxZfNqSsq1YXrcKzNzrAsuBYTne1
# {"data":"CAE=","links":[{"Name":"bar","Size":201,"Cid":{"/":"QmdKr2Dvn3yKeBoX6UU85AQ7GdfY9sW4N8DUPmhp5JeRXF"}},{"Name":"foo","Size":73,"Cid":{"/":"QmR2RTyLMEYKUdAKjG6jA74VzpVdDgFEJxrj4iL4nGWyWi"}}]}
```