IPFS makes the web work peer-to-peer
Problems

IPFS

Addressess

emerging networks
censorship

huge inefficiency
no offline use

bad security model
links break
IPFS: Distributed Web Protocol

IPLD: authenticated data model & formats

libp2p: modular p2p networking library

Multiformats: future-proofing & upgradability
IPFS: Lifecycle

Adding Files

Getting Files
IPFS:
Adding Files

```
λ: ipfs add -r docs
added QmZTR5bcpQD7cFgTorqxZDYaew1Wqgfbd2ud9QqGPAkK2V docs/about
added QmYCvbfNbCwFR45HiNP45rwJgvatpIw38D961L5qAhUM5Y docs/contact
added QmY5heUM5qgRubMDD1og9fhCPA6QdkMp3QCwd4s7gJsyE7 docs/help
added QmejvEPop4D7YUadeGqYWmZxHhLc4JBUczJJHWMzdcMe2y docs/ping
added QmXgqKTbdh83pQtKFb19SpMcpDDcKR2ujqk3pKph9aCNF docs/quick-start
added QmPZ9gcCEpqKTo6aq61g2nXGUhM4iCL3ewB6LDXZCtioEB docs/readme
added QmQ5vhrL7uv6tuoN9KeVBwd4PwfQkXdVVmDLUZuTNxqgvm docs/security-notes
added QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcz9Ba8nUH4uVv docs
5.97 KiB / 5.97 KiB [================================] 100.00%
```
IPFS:
Adding Files

λ: ipfs add -r docs
added QmZTR5bcpQD7cFgTorqxZDYaew1Wqgfbd2ud9QqGPAkK2V docs/about
added QmYCvbfNbcwFR45HiNP45rwJqvatp1W38D961L5qAhUM5Y docs/contact
added QmY5heUM5qRubMD1og9fhCPA6QdkMP3QCwd4s7gJsyE7 docs/help
added QmejvEPop4D7YUadeGqYWMzXh4JBCczJHWMzdcMe2y docs/ping
added QmXgqKTbdh83pQtKFB19SpMCpDDcKR2ujqk3pKph9aCNF docs/quick-start
added QmPZ9gCpEqKTo6aq61g2nXGUhM4iCL3ewB6LDXZCtioEB docs/readme
added QmQ5vhrL7u6tuoN9KeVBwd4PwfQkXdVVmDLUZuTNxqgvm docs/security-notes
added QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4uVv docs
5.97 KiB / 5.97 KiB [==================] 100.00%

-> CID: Content Identifier
-> IPFS Path: /ipfs/QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4uVv
-> Gateway URL: https://ipfs.io/ipfs/QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4uVv
IPFS:

Getting Files

λ: ipfs get -o docs /ipfs/QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4VuVv

Saving file(s) to docs
6.39 KiB / 6.39 KiB [==================================================] 100.00% 0s

CID
IPFS: Lifecycle

Adding Files

- Import
- Name
- Find
- Fetch

Getting Files
### Chunking
- UnixFS
- IPLD

### Import

### Name
- CID
- Path
- IPNS

### Find
- Routing
- DHT
- Kademlia

### Fetch
- Bitswap

### Contiguous File:

### Chunked File:
(each chunk is hashed)

- Deduplication
- Piecewise Transfer
- Seeking
Chunking
UnixFS
IPLD

CID
Path
IPNS

Routing
DHT
Kademlia

Bitswap

Contiguous File:

Chunked File:

- Deduplication
- Piecewise Transfer
- Seeking
**Import**
- Chunking
  - UnixFS
  - IPLD

**Name**
- CID
- Path
- IPNS

**Find**
- Routing
- DHT
- Kademlia

**Fetch**
- Bitswap

**Contiguous File:**
- Deduplication
- Piecewise Transfer
- Seeking

**Chunked File:**
**Chunking**
- UnixFS
- IPLD

**Name**
- CID
- Path
- IPNS

**Find**
- Routing
- DHT
- Kademlia

**Fetch**
- Bitswap

- **Contiguous File:**
  - Deduplication
  - Piecewise Transfer
  - Seeking

- **Chunked File:**
- **Deduplicated:**
### Import
- Chunking
  - UnixFS
  - IPLD

### Name
- CID
- Path
- IPNS

### Find
- Routing
- DHT
- Kademlia

### Fetch
- Bitswap

---

**Contiguous File:**
- Deduplication
- Piecewise Transfer
- Seeking

**Chunked File:**
Chunking
UnixFS
IPLD
CID
Path
IPNS
Routing
DHT
Kademlia
Bitswap
File Chunks:
**Import**
- Chunking
- UnixFS
- IPLD

**Name**
- CID
- Path
- IPNS

**Find**
- Routing
- DHT
- Kademlia

**Fetch**
- Bitswap

---

**UnixFS File:**

(merkle-tree-dag) - directed acyclic graph

---

**File Chunks:**

- 0-200
- 200-350
- 0-100
- 100-200
- 100-200
- 200-250

(merkle-link)
UnixFS Directory:

UnixFS File(s):

File Chunks:
Meta-format for understanding, encoding, and decoding merkle-linked data.
Linked Data

http://b.com/Bar.json -> {
  "foo": http://a.com/Foo.json
}

http://a.com/Foo.json -> {
  "content": "I am foo"
}
Linked Data

http://b.com/Bar.json -> {
    "foo": http://a.com/Foo.json
}

http://a.com/Foo.json -> {
    "content": "I am foo"
}
**Merkle-Linked Data**

```json
QmBar -> {
    "foo": QmFoo
}
QmFoo -> {
    "content": "I am foo"
}
```

- Immutable
- Authority Less
Content Identifier

- Used for **content addressing**
- Are **self describing**
- Used to name every piece of data in IPFS/IPLD
- Are basically a **hash** with some **metadata**
Digression:

Content Addressing / Location Addressing
Location Addressing

"My cat, Ozzy, is here."
Content Addressing

"This is my cat, Ozzy."
Location Addressing

Digression: Content Addressing
Location Addressing

Digression: Content Addressing
Chunking
UnixFS
IPLD
CID
Path
IPNS
Routing
DHT
Kademlia
Bitswap
Chunking
UnixFS
IPLD
CID
Path
IPNS
Routing
DHT
Kademlia
Bitswap

Digression: Content Addressing

Location Addressing

"That's the wrong cat!"
Digression: Content Addressing

Location Addressing

"That's the wrong cat!"

(But you can't know that!)
Routing

DHT

Kademlia

Verifiable, Immutable, Trustless

Permanent

Import

Name

Find

Fetch

Chunking
UnixFS
IPLD

CID
Path
IPNS

Routing
DHT

Bitswap

Import

Name

Find

Fetch

Permanent

No
Digression:
Multiformats: Self Describing Data
Digression: Multiformats

- **Multicodec**: a non-magic number to uniquely identify a format, protocol, etc.
- **Multihash**: a self-describing hash digest.
- **Multibase**: a self-describing base-encoded string.
**Digression: Multiformats**

Multicodec: a non-magic number.

<table>
<thead>
<tr>
<th>name, identity, ip4, dcpp, dnsaddr, protobuf, cbor, raw, ...</th>
<th>tag, multihash, multiaddr, multiaddr, multiaddr, serialization, serialization, ipld, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>code, description</td>
<td>raw binary</td>
</tr>
<tr>
<td>0x00,</td>
<td>Protocol Buffers</td>
</tr>
<tr>
<td>0x04,</td>
<td>CBOR</td>
</tr>
<tr>
<td>0x21,</td>
<td>raw binary</td>
</tr>
<tr>
<td>0x38,</td>
<td></td>
</tr>
<tr>
<td>0x50,</td>
<td></td>
</tr>
<tr>
<td>0x51,</td>
<td></td>
</tr>
<tr>
<td>0x55,</td>
<td></td>
</tr>
</tbody>
</table>

[github.com/multiformats/multicodec](https://github.com/multiformats/multicodec)
Digression: Multiformats

Multihash: a self-describing hash digest:
- Hash Function (*multicodec*)
- Hash Digest Length
- Hash Digest

[github.com/multiformats/multihash](https://github.com/multiformats/multihash)
Digression: A bit of metadata

Multibase: a self-describing base encoding.

- A multibase prefix.
  - b - base32
  - z - base58
  - f - base16
- Followed by the base encoded data.

bafybeibxm2...
Self Describing

- **CIDv0**: `QmS4u...`
  - Base58 encoded sha256 `multihash`
- **CIDv1**: `bafybei...`
  - Multibase encoded (ipld format `multicodec`, `multihash`) tuple.
- **Why CIDv1?**
  - Can be encoded in arbitrary bases (base32, base58, etc.).
  - Can link *between* merkle-dag formats using the `ipld format` `multicodec`. 
IPFS uses **paths**, not **URIs/URLs**:

Like URLs, paths are **namespaced**:

/ipfs/QmFoo/welcome.txt
/ipns/QmBar/index.html

Unlike URLs, paths are **recursive**:

/dns/github.com/tcp/22/ssh.git

Versus:

**Not Composable!**

/git+ssh://github.com:22
IPNS maps **Public Keys** to *paths*

```
/ipns/QmMyKey -> /ipfs/QmFoo (signed)
```

**IPNS is mutable**

```
/ipns/QmMyKey -> /ipfs/QmSomethingNew
```

**IPNS can point to arbitrary paths**

```
/ipns/QmMyKey -> /ipns/QmYourKey
```
Content Address (CID) → Location Address (Peer)
Solution: Keep a "routing table"

<table>
<thead>
<tr>
<th>What</th>
<th>Who</th>
</tr>
</thead>
<tbody>
<tr>
<td>QmFoo</td>
<td>Ozzy</td>
</tr>
<tr>
<td>QmBar</td>
<td>Izzy</td>
</tr>
</tbody>
</table>
But the table is too **BIG**!

<table>
<thead>
<tr>
<th>What</th>
<th>Who</th>
</tr>
</thead>
<tbody>
<tr>
<td>QmFoo</td>
<td>Ozzy</td>
</tr>
<tr>
<td>QmBar</td>
<td>Izzy</td>
</tr>
<tr>
<td>... <em>millions of lines later</em> ...</td>
<td></td>
</tr>
<tr>
<td>QmXXXXXXXXXXXXXXXXXXXXXXXX</td>
<td>Seth (not a cat!)</td>
</tr>
</tbody>
</table>
Solution: Distribute the routing table and give a little bit to each peer.

Ozzy Knows

<table>
<thead>
<tr>
<th>What</th>
<th>Who</th>
</tr>
</thead>
<tbody>
<tr>
<td>QmBar</td>
<td>Izzy</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Izzy Knows

<table>
<thead>
<tr>
<th>What</th>
<th>Who</th>
</tr>
</thead>
<tbody>
<tr>
<td>QmFoo</td>
<td>Ozzy</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
How do we know who has what piece of the routing table?
How do we know who has what piece of the routing table?

**Solution:** *Deterministically* distribute the routing table.
**Distance Metric:** Is peer X closer to content C than peer Y?

**Query Algorithm:** Given the distance metric, how do we find the peers closest to C.
**Distance Metric:** \( \text{XOR} (\text{HASH}(C), \text{HASH}(\text{Peer})) \)

**Query Algorithm:**

1. Ask the closest peers you know for closer peers.
2. Remember the closest peers.
Distance Metric: "Is this closer?"
Query Algorithm: "How do I get closer?"
- Chunking
- UnixFS
- IPLD
- CID
- Path
- IPNS
- Routing
- DHT
- Kademlia
- Bitswap
Izzy wants QmTreats and QmToy.

Ozzy wants QmCuddles, QmFood, and QmAttention.

Chunks: UnixFS, IPLD

Names: CID, Path, IPNS

Routing: DHT, Kademlia

Fetch: Bitswap

Import: Portions of the diagram include 'Chunks UnixFS IPLD', 'Names CID Path IPNS', 'Routing DHT Kademlia', and 'Fetch Bitswap'.
Izzy wants:
- QmTreats
- QmToy

Ozzy wants:
- QmCuddles
- QmFood
- QmAttention
**Izzy** Wants
- QmTreats
- QmToy

**Ozzy** Wants
- QmCuddles
- QmFood
- QmAttention

**Import**
- Chunking
- UnixFS
- IPLD

**Name**
- CID
- Path
- IPNS

**Find**
- Routing
- DHT
- Kademlia

**Bitswap**
Routing
Kademlia

Izzy
- QmTreats
- QmToy

Ozzy
- QmCuddles
- QmFood
- QmAttention

Izzy Wants
- QmToy

Ozzy Wants
- QmCuddles
- QmFood
- QmAttention

Bitswap

Import
- Chunking
- UnixFS
- IPLD

Name
- CID
- Path
- IPNS

Find
- Routing
- HT
- Kademlia

Fetch

Import
- Fetch
- Find
- Name
- Bitswap

QmToy

QmFood

QmAttention
Routing

DHT

Kademlia

Izzy Wants

● QmTreats

Ozzy Wants

● QmCuddles

Izzy

Ozzy

Chunking

UnixFS

IPLD

CID

Path

IPNS

Routing

DHT

Kademlia

Bitswap
IPFS

- **Import**
  - Chunking
  - UnixFS
  - IPLD

- **Name**
  - CID
  - Path
  - IPNS

- **Find**
  - Routing
  - DHT
  - Kademlia

- **Fetch**
  - Bitswap
How IPFS Works

(approximately)

Sawood Alam (@ibnesayeed)
Old Dominion University

Original slides by Steven Allen (@Stebalien)