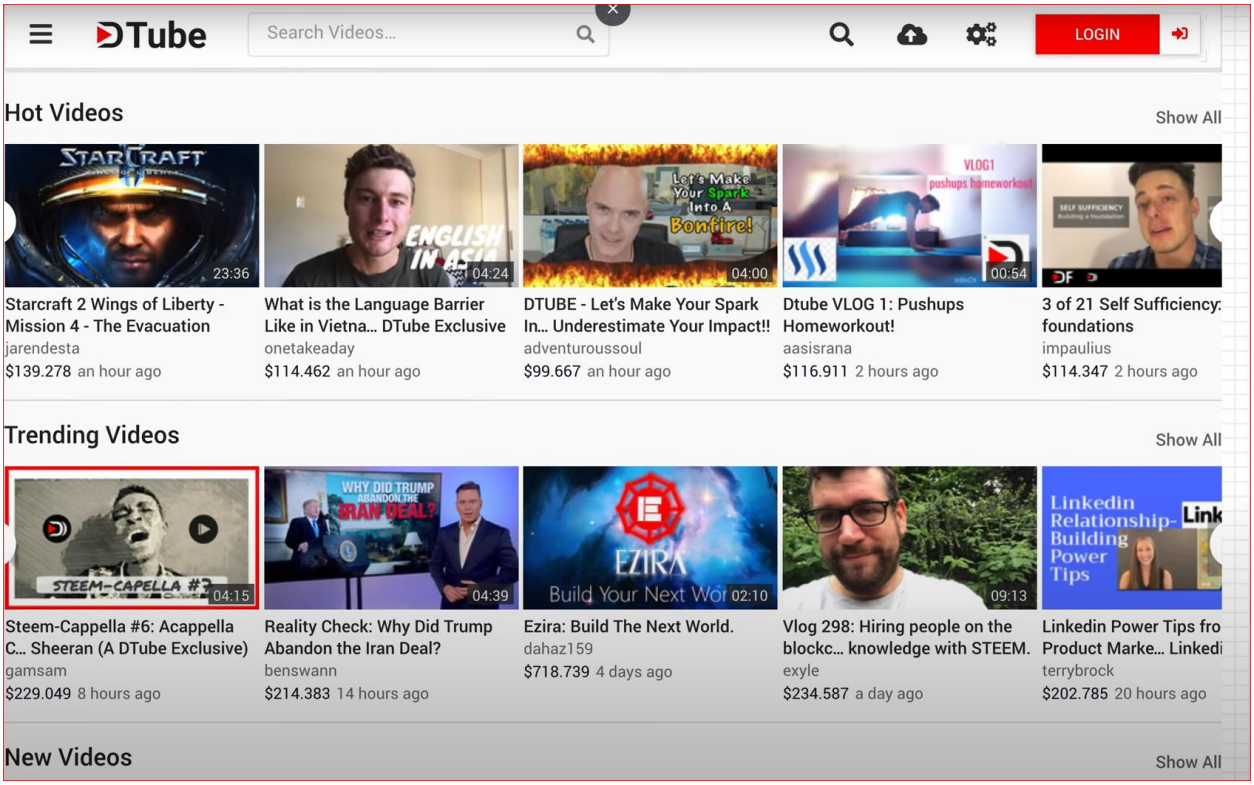
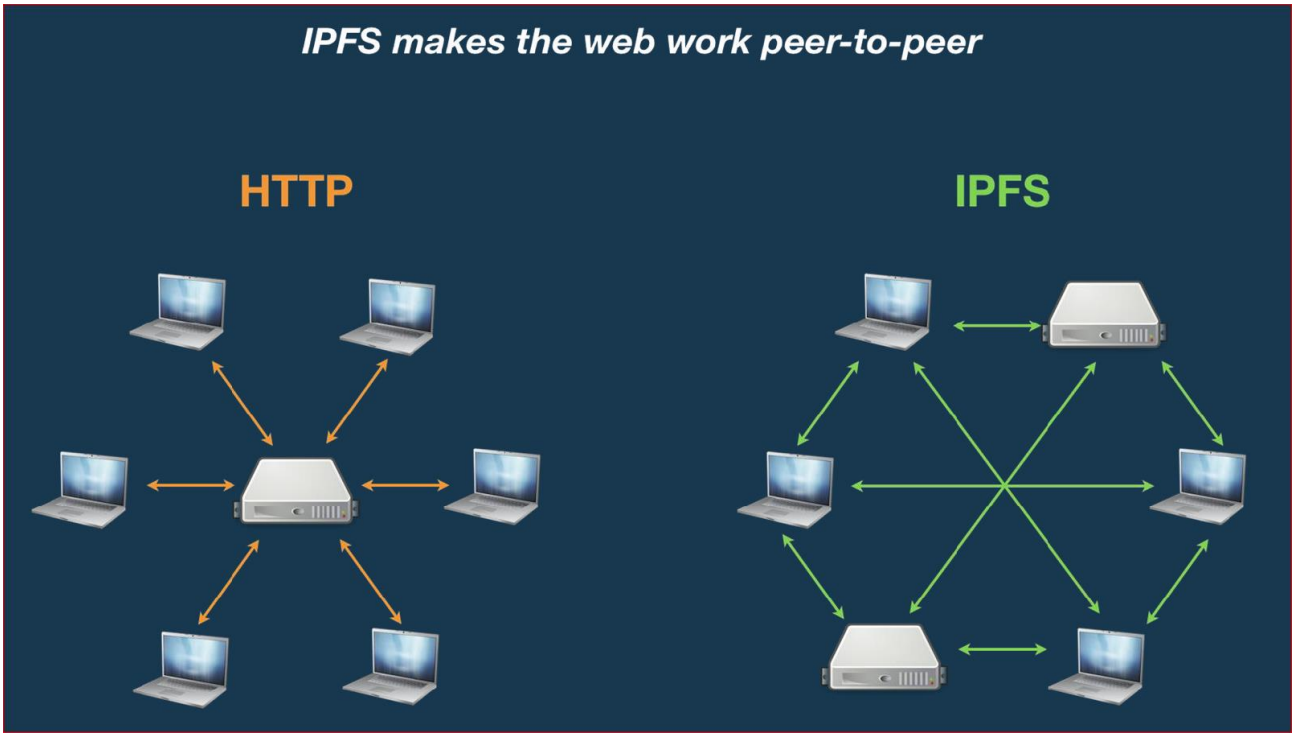
IPFS [InterPlanetary File System]

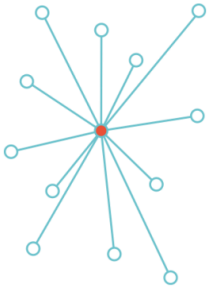
Definition and overview:-

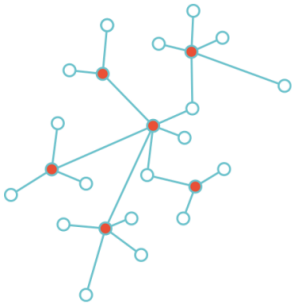
IPFS is a peer-to-peer (p2p) distributed storage network for storing and accessing files, websites, applications, and data.

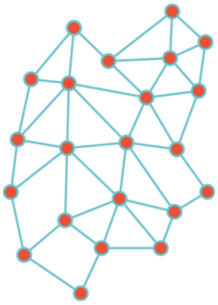


Http vs IPFS Protocol:-





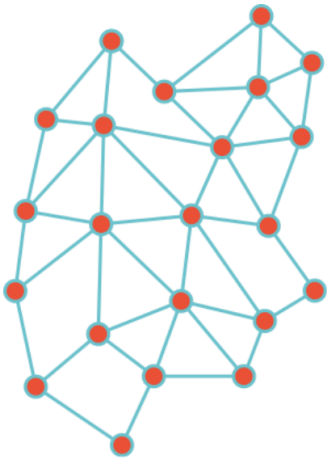




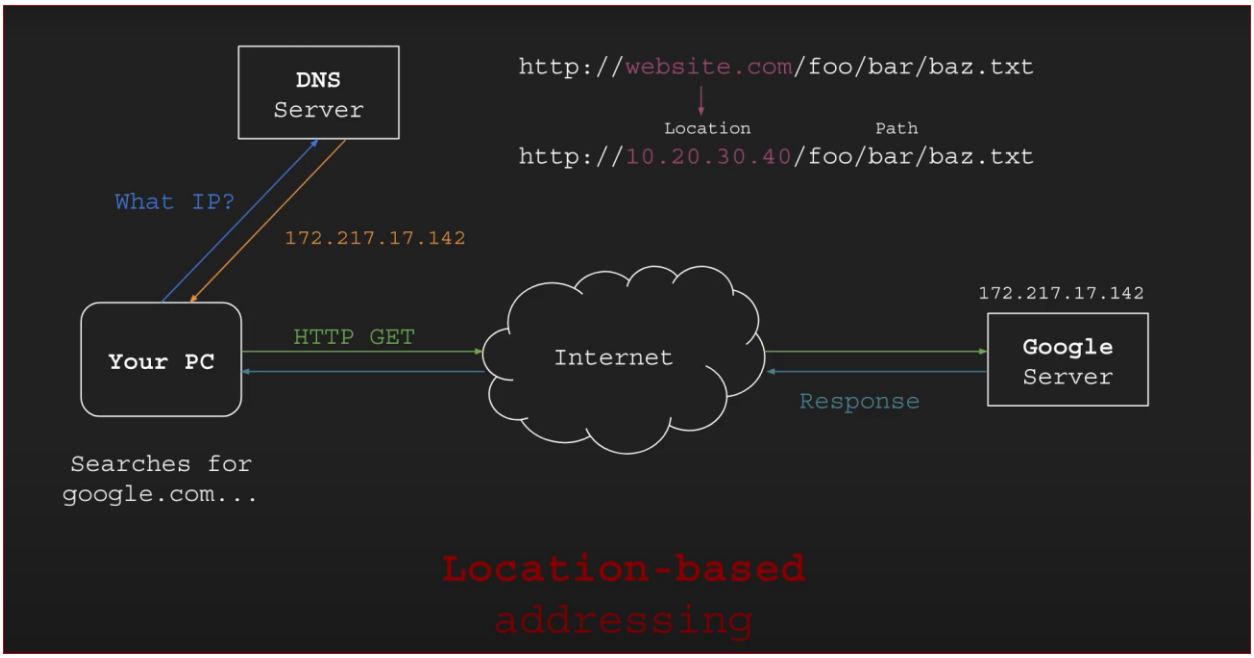
CENTRALIZED DECENTRALIZED DISTRIBUTED

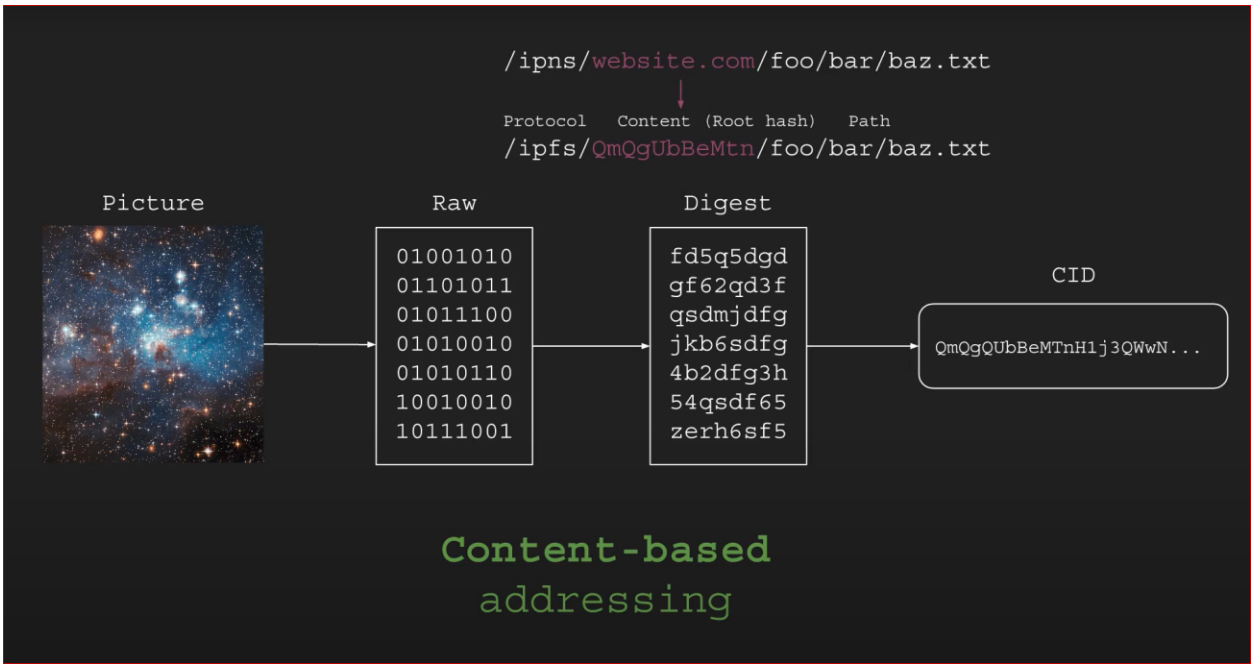
WHY DISTRIBUTED?

- Resilience / Offline-first
- Speed
- Scalability
- Security
- Efficiency
- Trustless



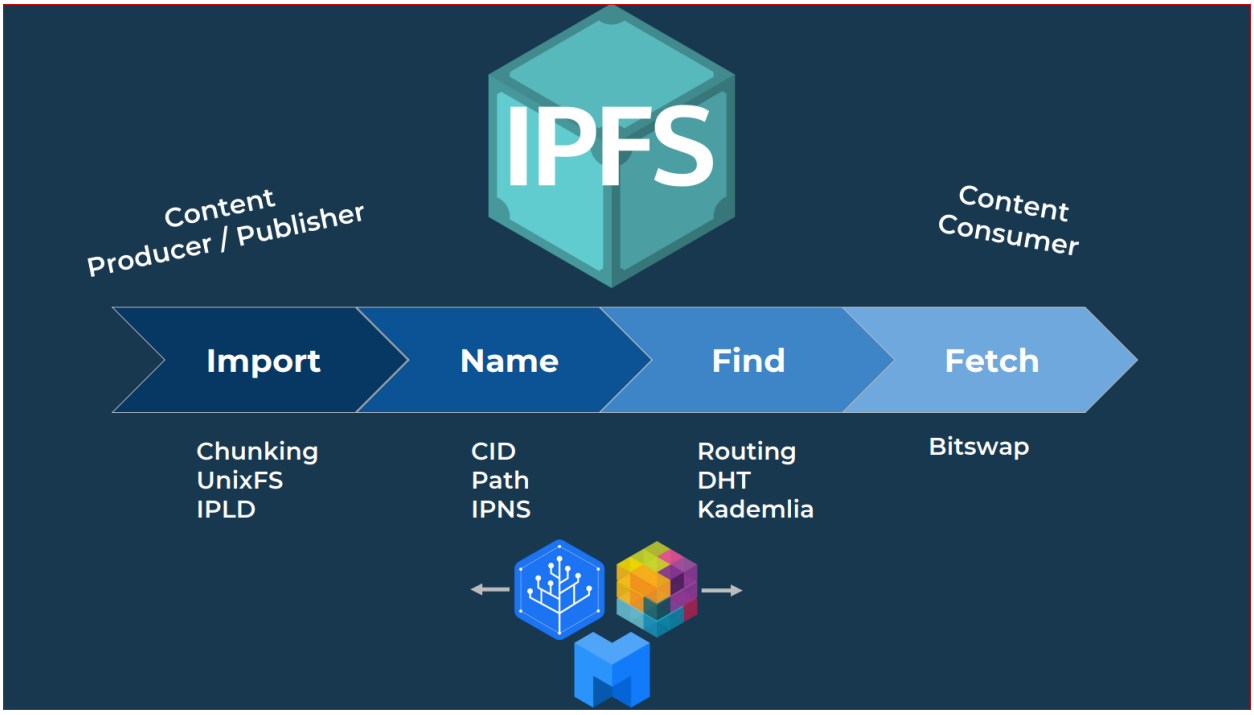
Location based addressing vs Content Based Addressing:-





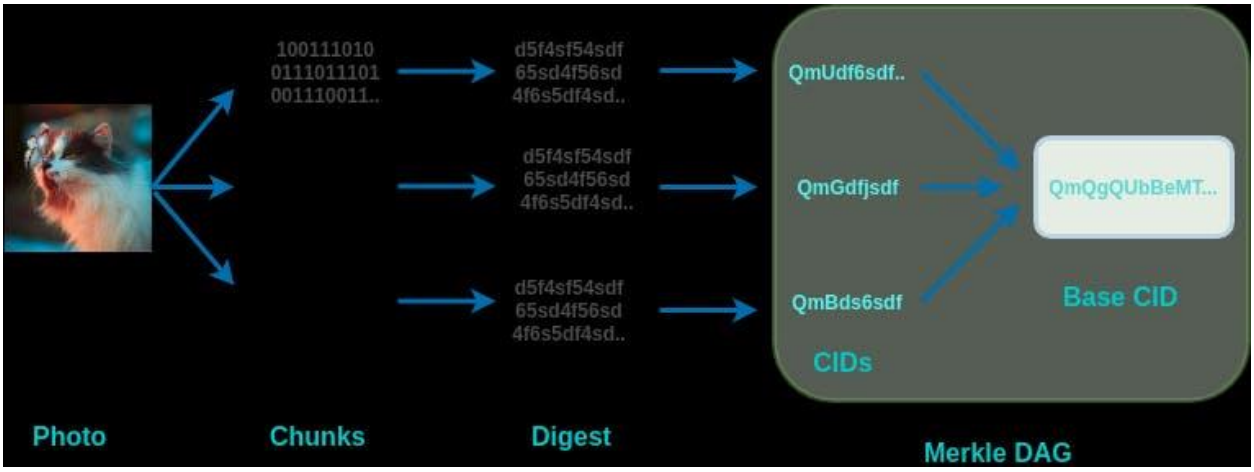
- Raw -> Digest => **SHA256 [Unique Identifier]**
- Digest -> CID => **Multi-Hash**

IPFS Lifecycle:



IPLD [InterPlanetary Linked Data]:

IPLD provides standards and formats to build Merkle-DAG data structures, like those that represent a filesystem.



Note: If the files are bigger than **256 kB**, then they are broken down into smaller chunks, so that all the part are equal or smaller than **256 kb.** Each of these chunks is first converted into a digest and then into CIDs.

IPFS uses **IPLD** (**IPLD** uses **Merkle DAG**, or directed acyclic graph) for managing all the chunks and linking it to the base CID.
IPLD (objects) consist of 2 components:

- Data — blob of unstructured binary data of size < 256 kB.

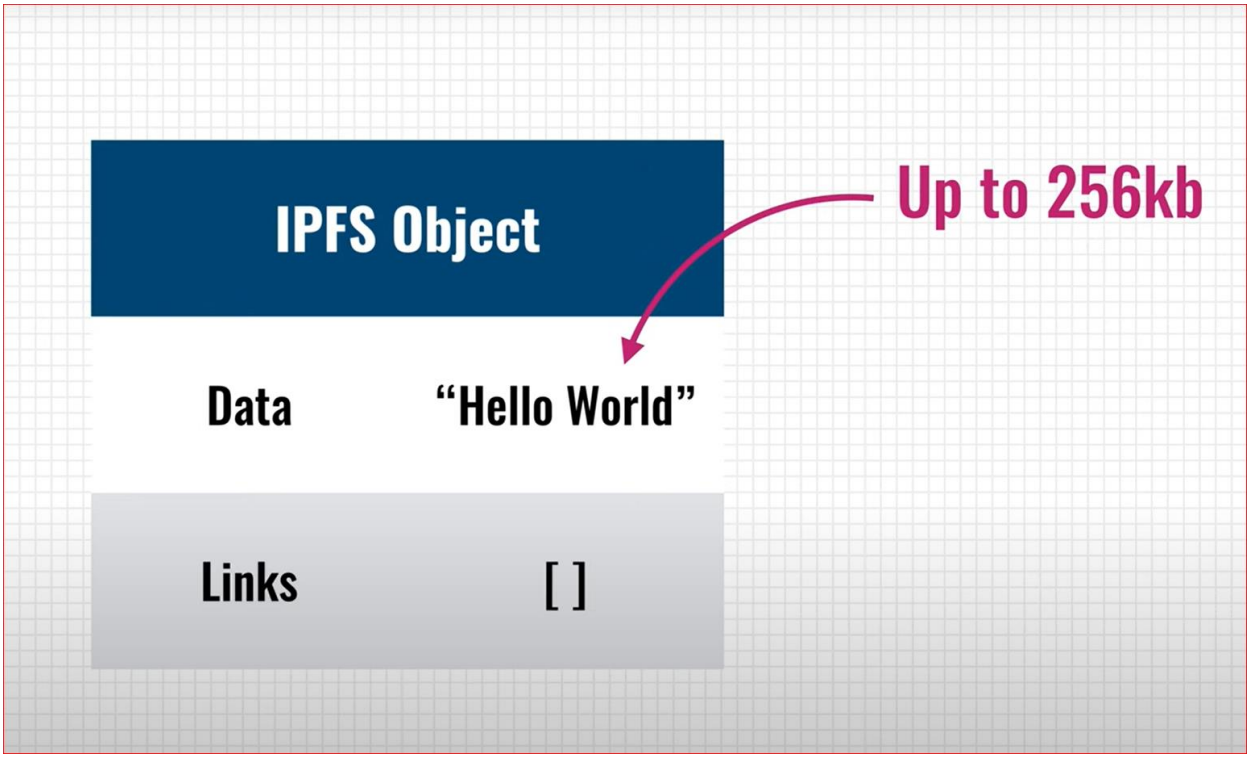
- Links — array of Link structures. These are links to other IPFS objects.

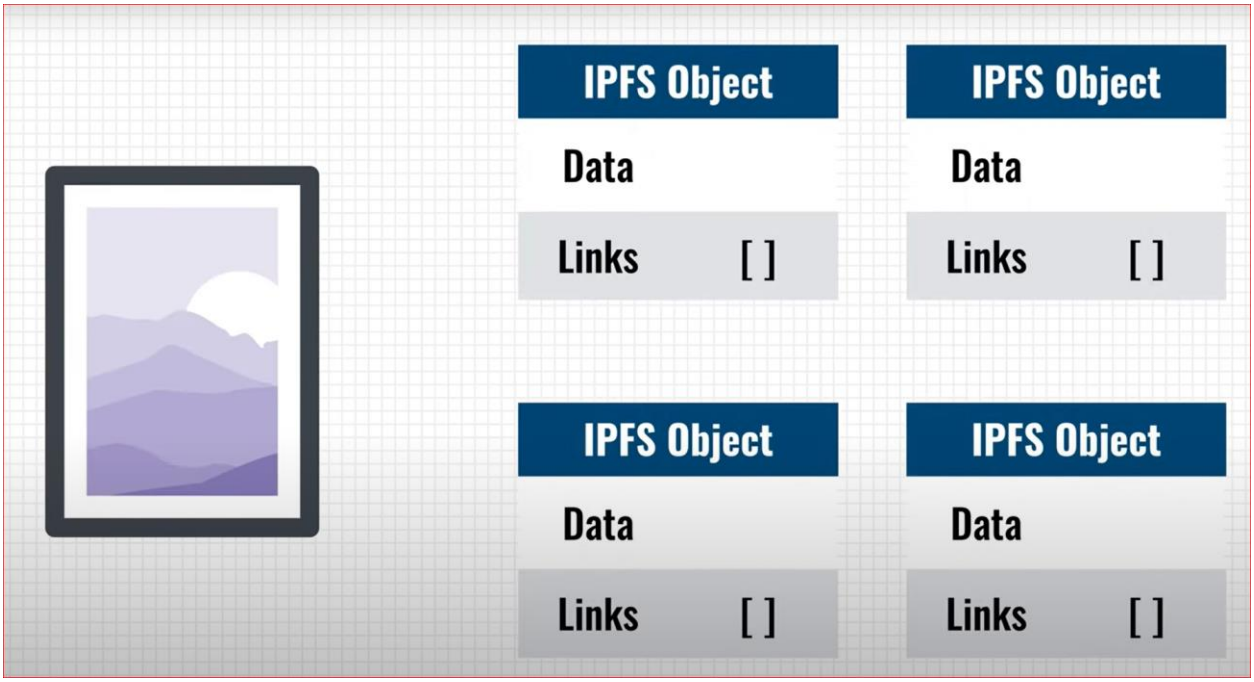
Every IPLD Link has 3 parts:

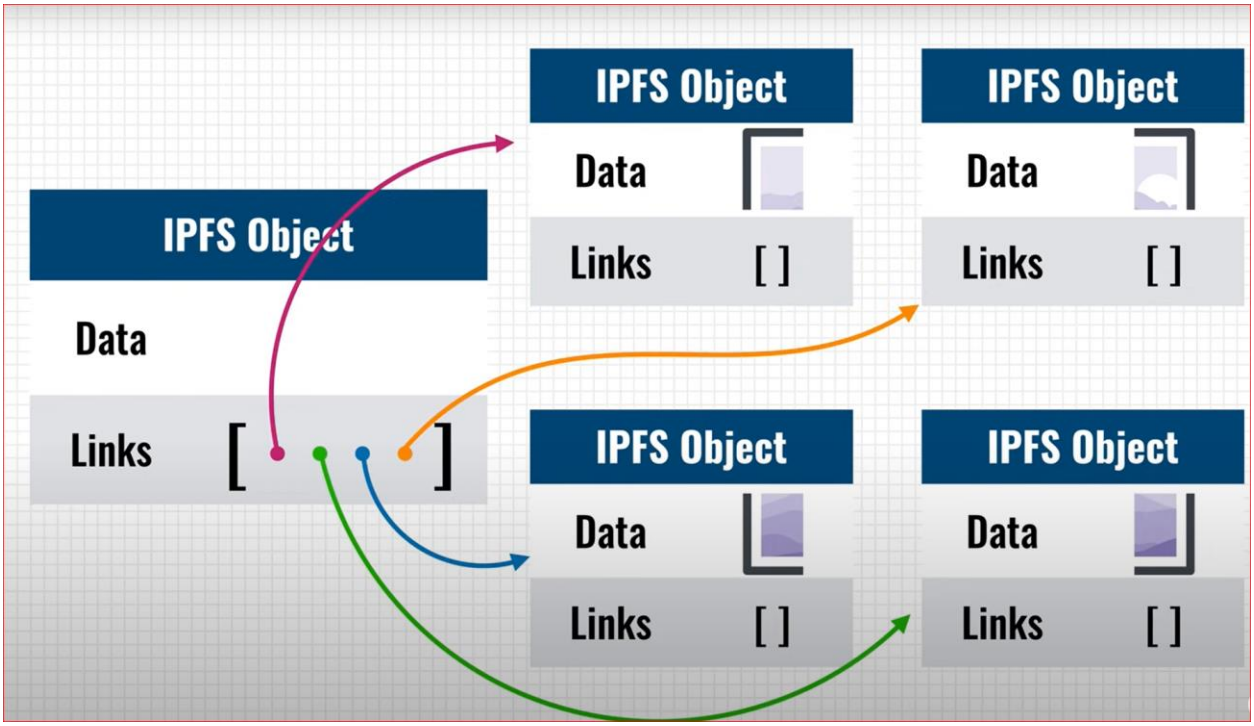
- Name — name of the Link
- Hash — hash of the linked IPFS object
- Size — the cumulative size of linked IPFS object, including following its links

Command: ipfs object get Qmd286K6pohQcTKYqnS1YhWrCiS4gz7Xi34sdwMe9USZ7u

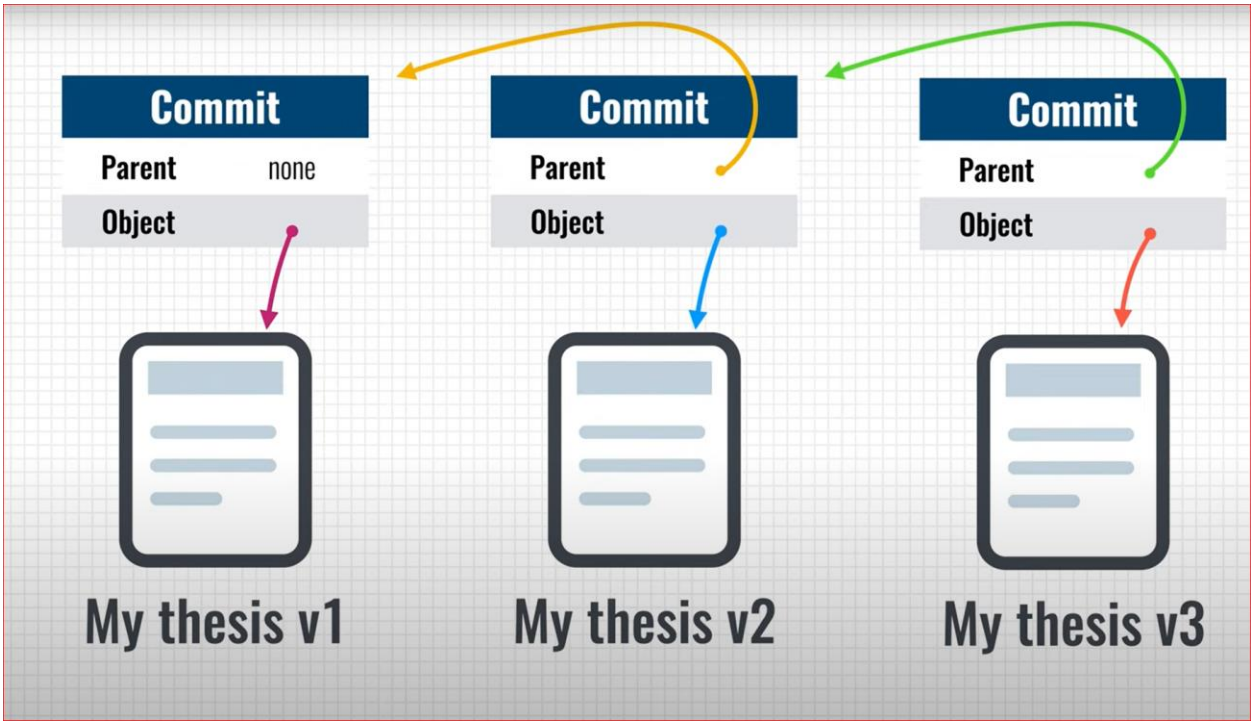

```
{
  "Links": [
    {
      "Name": "chunk-1",
      "Hash": "QmZ5RgT3jJhRNMEgLSEsez9uz1oDnNeAysLLxRco8jz5Be",
      "Size": 262158
    },
    {
      "Name": "chunk-2",
      "Hash": "QmUZvm5TertyZagJfoaw5E5DRvH6Ssu4Wsdfw69NHaNRTc",
      "Size": 262158
    },
    {
      "Name": "chunk-3",
      "Hash": "QmTA3tDxTZn5DGaDshGTu9XHon3kcRt17dgyoomwbJkxvJ",
      "Size": 262158
    },
    {
      "Name": "chunk-4",
      "Hash": "QmbsEhRqFwKAUoc6ivZyPa1vGUxFKBT4ciH79gVszPcFEG",
      "Size": 69716
    },
  ],
  "Data": "\b\u0002\u0018O\u0001 \u0010 \u0010 \u0010 \u0010
\u0010 \u0010 \u0010 \u0010 \u0010 \u0010 \u0010 \u0010
\u0010 \u0010 \u0010 O\u0004"
}
```


IPFS Object
Up to 256kb
Data
"Hello World"
Links
[ ]


IPFS Object
Data
Links
[ ]
IPFS Object
Data
Links
[ ]
IPFS Object
Data
Links
[ ]
IPFS Object
Data
Links
[ ]

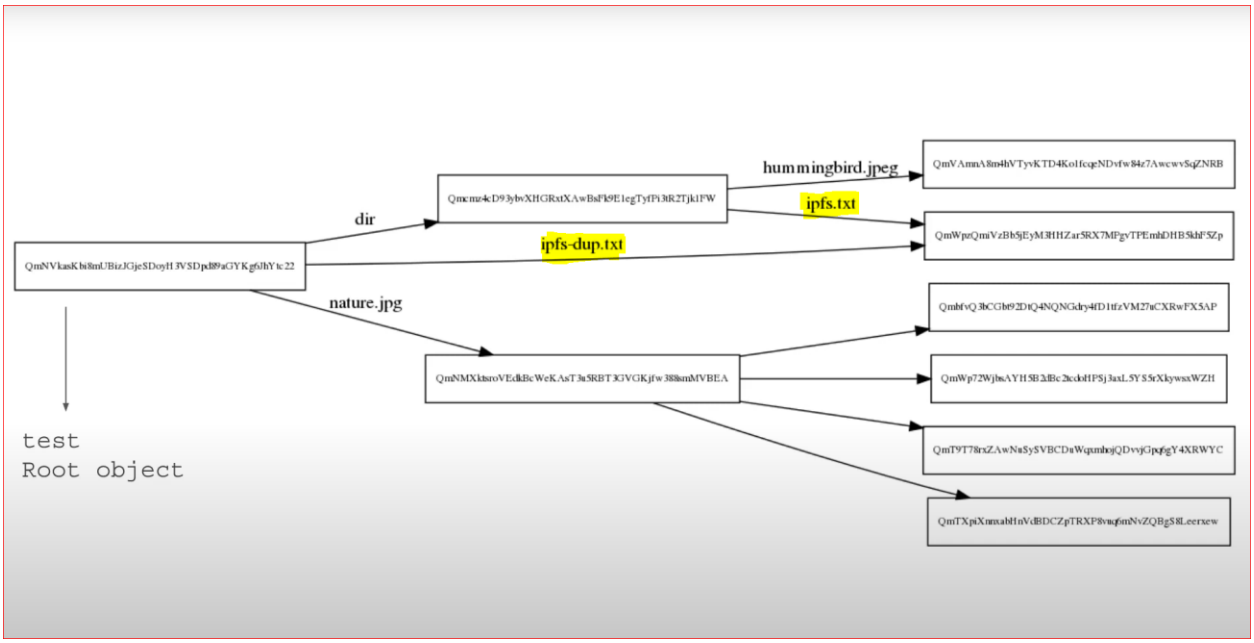

IPFS Object
Data
Links [ ]
IPFS Object
Data
Links [ ]
IPFS Object
Data
Links [ ]
IPFS Object
Data
Links [ ]
IPFS Object
Data
Links [ ]

Versioning:


Commit
Parent none
Object
Commit
Parent
Object
Commit
Parent
Object
My thesis v1
My thesis v2
My thesis v3

Deduplication:

```
[banerepublic] ~/P/Y/IPFS> ls test
dir/  ipfs-dup.txt  nature.jpg
[banerepublic] ~/P/Y/IPFS> tree test
test
├── dir
│   ├── hummingbird.jpeg
│   └── ipfs.txt
├── ipfs-dup.txt
└── nature.jpg

1 directory, 4 files
[banerepublic] ~/P/Y/IPFS> ipfs add -r test
added QmVAmnA8m4hVTyvKTD4Ko1fcqeNDvfw84z7AwcwvSqZNRB test/dir/hummingbird.jpeg
added QmWpzQmiVzBb5jEyM3HHZar5RX7MPgvTPEmhDHB5khF5Zp test/dir/ipfs.txt
added QmWpzQmiVzBb5jEyM3HHZar5RX7MPgvTPEmhDHB5khF5Zp test/ipfs-dup.txt
added QmNMXktsroVEdkBcWeKAsT3u5RBT3GVGKjfw388smMVBEA test/nature.jpg
added Qmcmz4cD93ybvXHGRxtXAwBsFk9E1egTyfPi3tR2Tjk1FW test/dir
added QmNVkasKbi8mUBizJGjeSDoyH3VSDpd89aGYKg6JhYtc22 test
 833.85 KiB / 833.85 KiB [=============================================================================================] 100.00%
```



IPNS [InterPlanetary Naming System]:

To share IPFS address such as `/ipfs/QmbezGequPwcsWo8UL4wDF6a8hYwM1hmbzYv2mnKkEWaUp` with someone, we would need to give the person a **new link every time you update the content**.

IPNS solves this issue by creating an address(hash of a public key) that can be updated (mutable link).

1. Start your IPFS daemon, if it isn't already running:

```
ipfs daemon
```

2. Open another command line window and create the file that you want to set up with IPNS. For the tutorial, we're just going to create a simple *hello world* file:

```
echo "Hello IPFS" > hello.txt
```

3. Add your file to IPFS:

```
ipfs add hello.txt

> added QmUVTKsrYJpaxUT7dr9FpKq6AoKHhEM7eG1ZHGL56haKLG hello.txt
> 11 B / 11 B [=============================================================] 100.00%
```

   Take note of the CID output by IPFS.

4. Use `cat` and the CID you just got from IPFS to view the file again:

```
ipfs cat QmUVTKsrYJpaxUT7dr9FpKq6AoKHhEM7eG1ZHGL56haKLG

> Hello IPFS
```

5. Publish your CID to IPNS:

```
ipfs name publish /ipfs/QmUVTKsrYJpaxUT7dr9FpKq6AoKHhEM7eG1ZHGL56haKLG

> Published to k51qzi5uqu5dkkciu33khkzbcmxtyhn376i1e83tya8kuy7z9euedzyr5nhoew: /ipfs/QmUVTKsr
```

   `k51...` is the public key or IPNS name of the IPFS you are running. You can now change the file repeatedly, and, even though the CID changes when you change the file, you can continue to access it with this key.

6. You can view your file by going to `https://gateway.ipfs.io/ipns/k51qzi5uqu5dkkciu33khkzbcmxtyhn376i1e83tya8kuy7z9euedzyr5nhoew` :

```
curl https://gateway.ipfs.io/ipns/k51qzi5uqu5dkkciu33khkzbcmxtyhn376i1e83tya8kuy7z9euedzyr5nh

> Hello IPFS
```

7. Make a change to your file, add it to IPFS, and update your IPNS:

```
echo "Hello again IPFS" > hello.txt
ipfs add hello.txt

> added QmaVfeg2GM17RLjBs9C4fhpku6uDgrEGUYCTC183VrZaVW hello.txt
> 17 B / 17 B [=============================================================] 100.00%

ipfs name publish QmaVfeg2GM17RLjBs9C4fhpku6uDgrEGUYCTC183VrZaVW

> Published to k51qzi5uqu5dkkciu33khkzbcmxtyhn376i1e83tya8kuy7z9euedzyr5nhoew: /ipfs/QmaVfeg2
```

8. You can now go back to `https://gateway.ipfs.io/ipns/k51qzi5uqu5dkkciu33khkzbcmxtyhn376i1e83tya8kuy7z9euedzyr5nhoew` to view your updated file using the same address:

```
curl https://gateway.ipfs.io/ipns/k51qzi5uqu5dkkciu33khkzbcmxtyhn376i1e83tya8kuy7z9euedzyr5nh

> Hello again IPFS
```

You can view the CID of the file associated with your `k5` key by using `name resolve` :

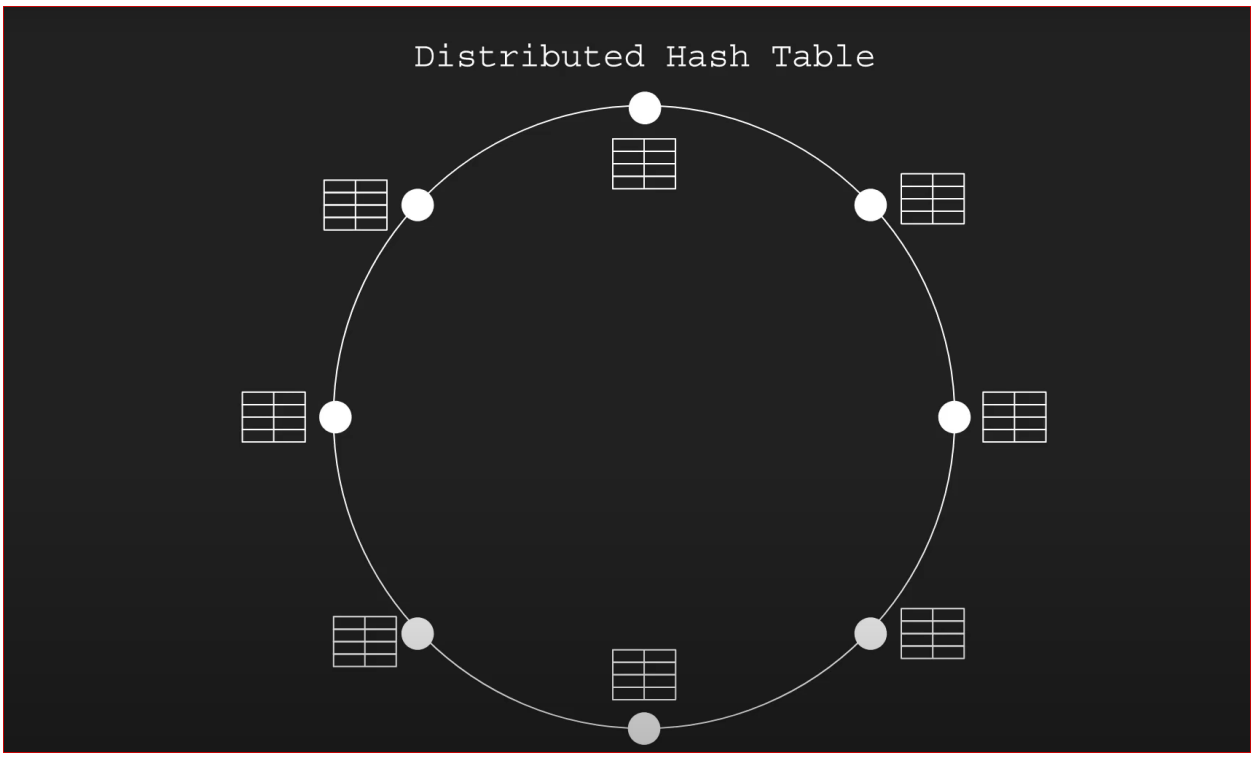
```
ipfs name resolve

> /ipfs/QmaVfeg2GM17RLjBs9C4fhpku6uDgrEGUYCTC183VrZaVW
```

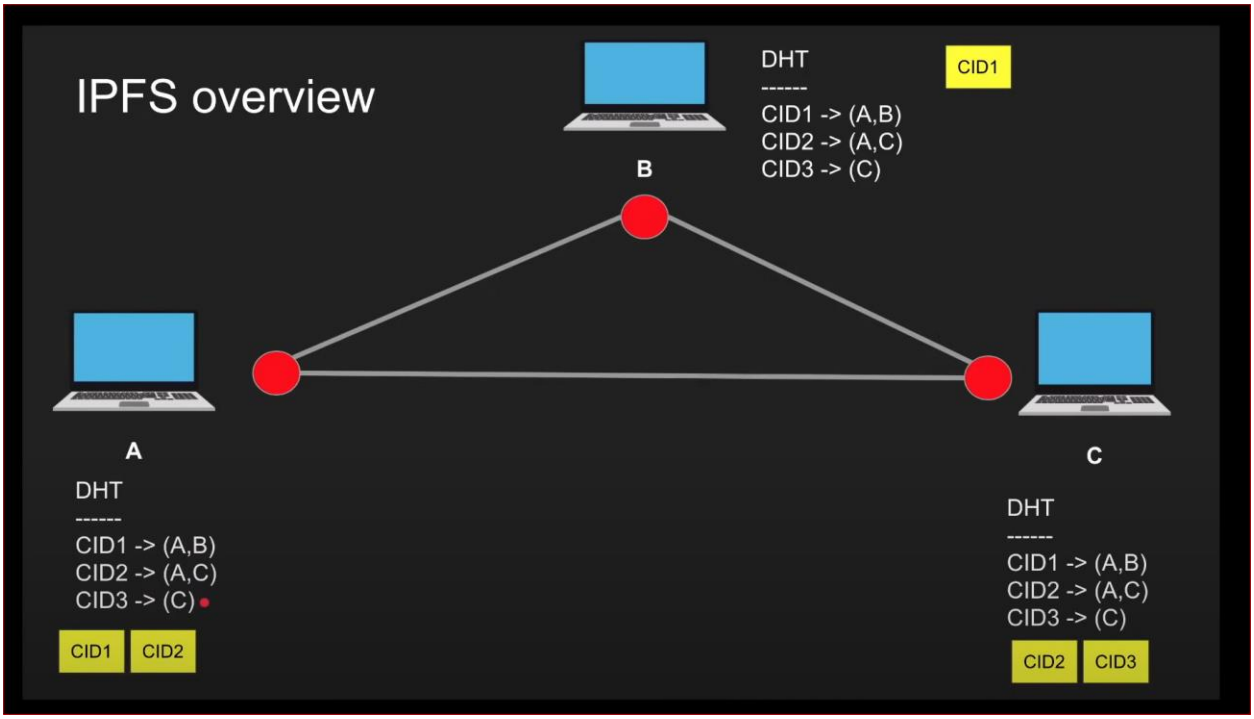
DHT [Distributed Hash Table]:

A Distributed Hash Table (DHT) provides a 2-column table (key-value store) maintained by multiple peers

The **DHT** in IPFS is used to provide:

- Content discovery (ContentID=PeerID)
- Peer routing (PeerID=/ip4/1.2.3.4/tcp/1234)

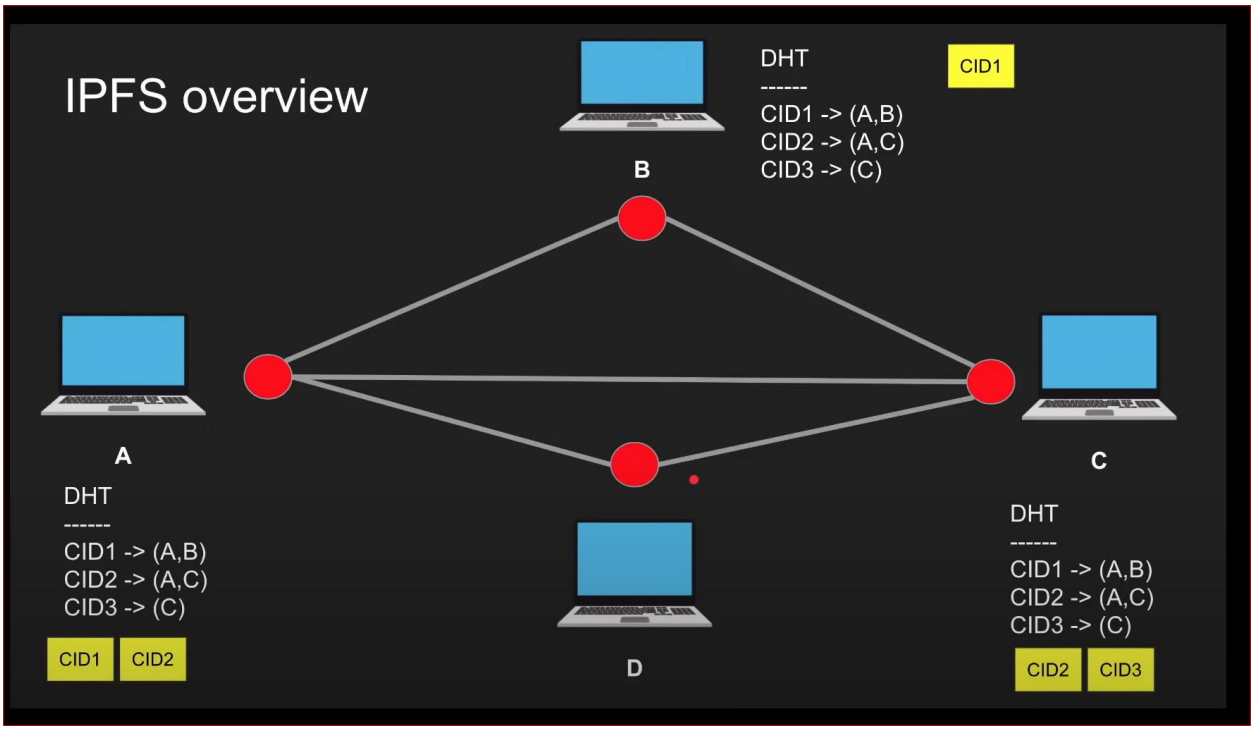

Distributed Hash Table


IPFS overview
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID1
B
A
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID1
CID2
C
DHT

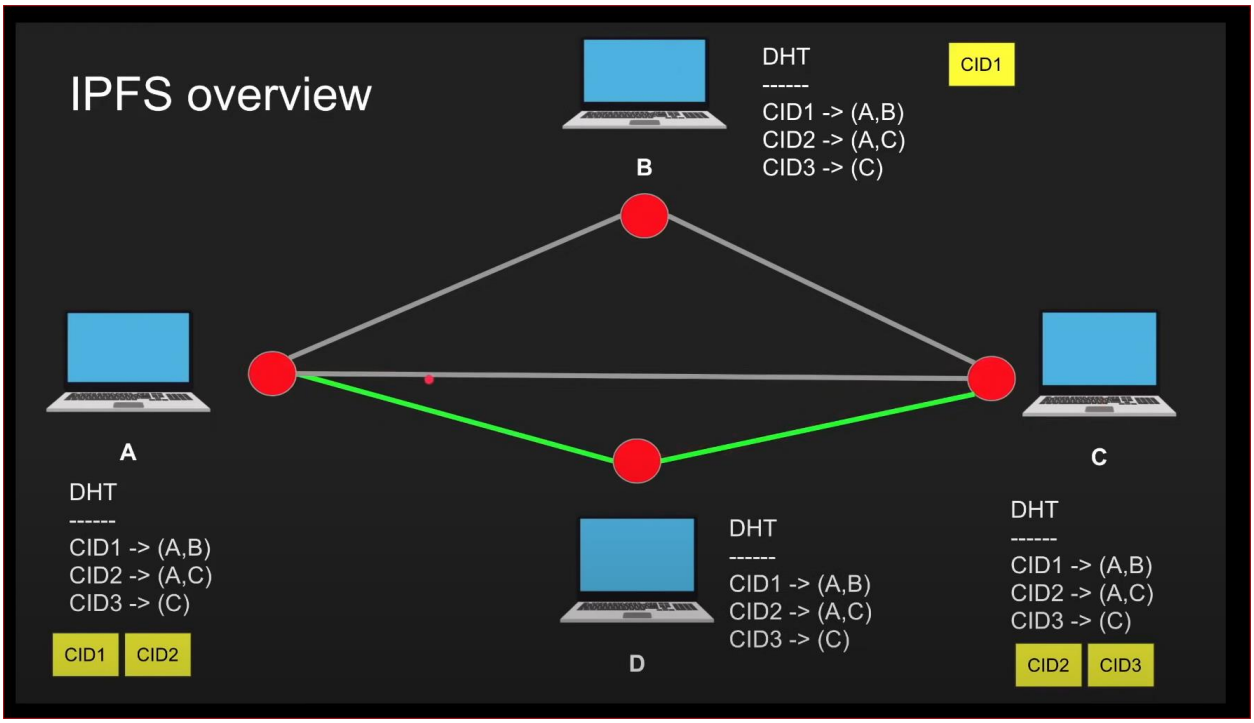
CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID2
CID3


IPFS overview
B
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID1
A
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID1
CID2
C
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID2
CID3
D

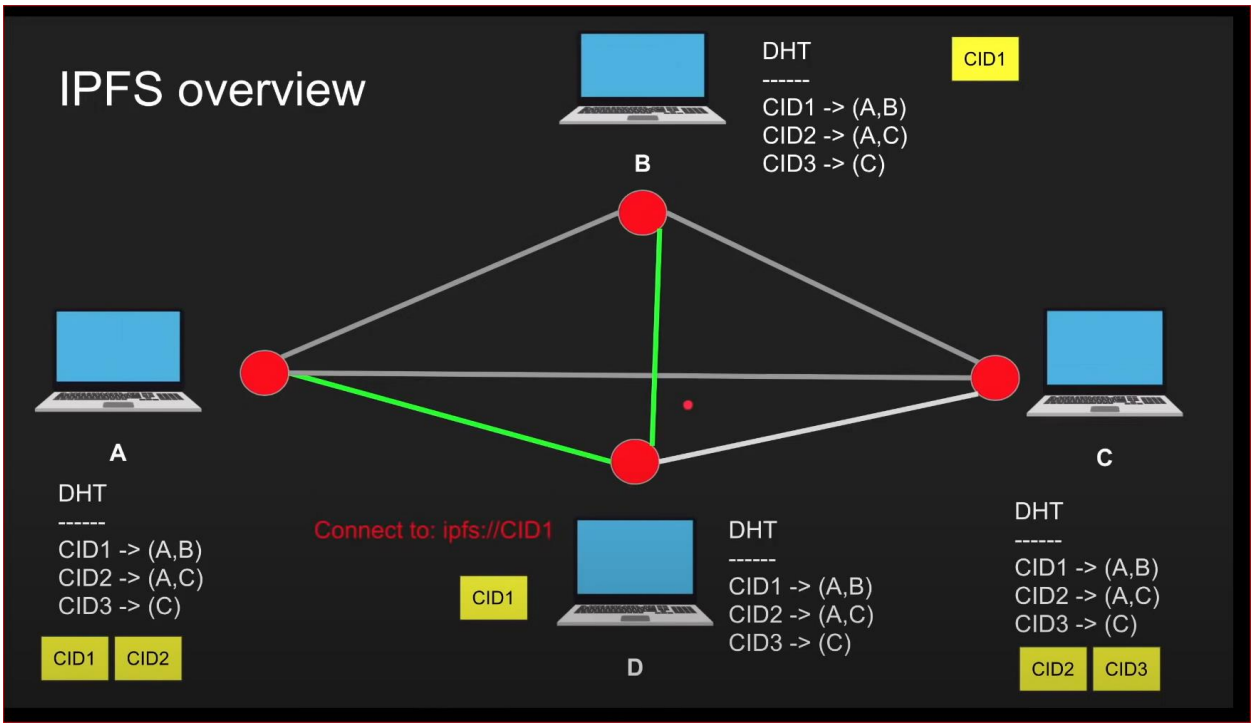

IPFS overview
B
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID1
A
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID1
CID2
C
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID2
CID3
D
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)

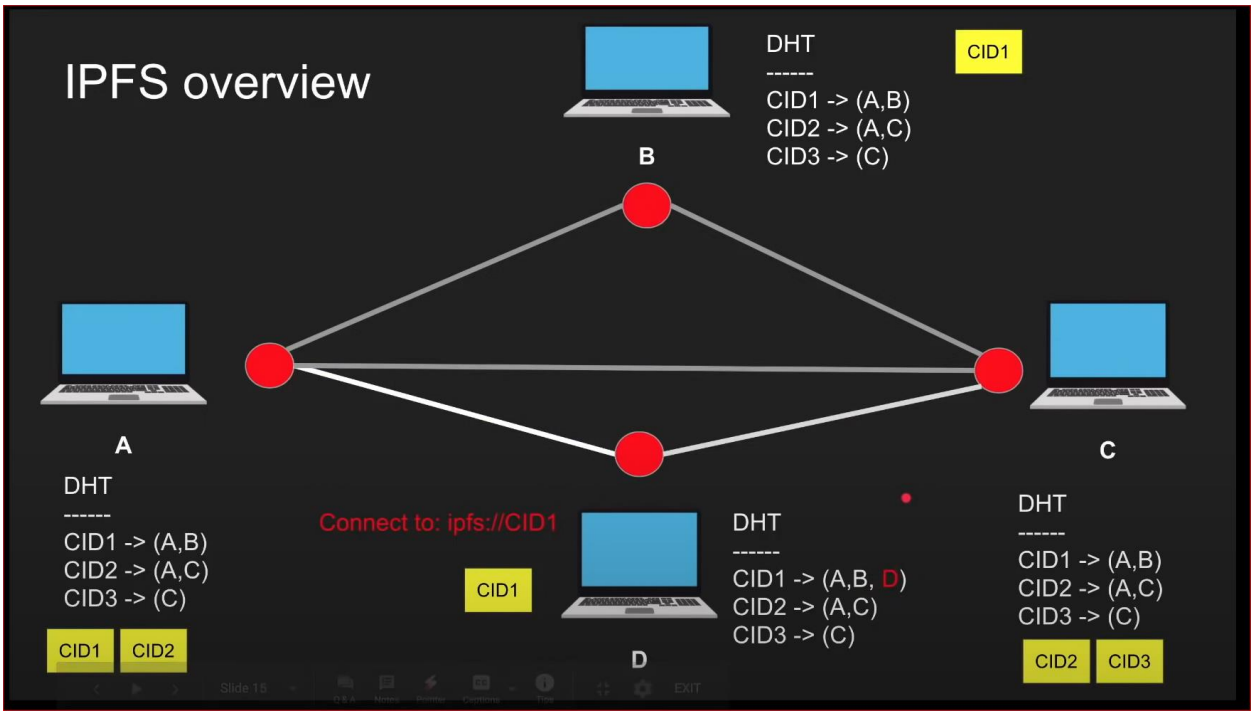

IPFS overview
B
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID1
A
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID1
CID2
Connect to: ipfs://CID1
CID1
D
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
C
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID2
CID3

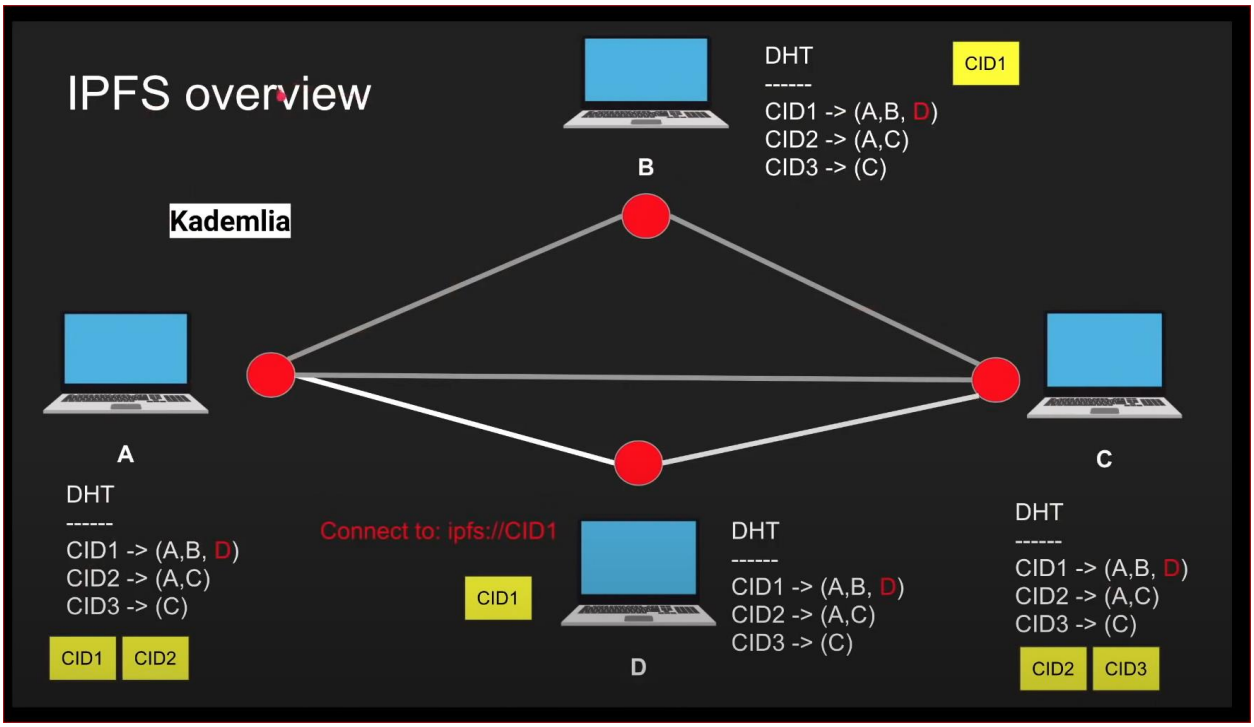

IPFS overview
B
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID1
A
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID1
CID2
Connect to: ipfs://CID1
CID1
D
DHT

CID1 -> (A,B, D)
CID2 -> (A,C)
CID3 -> (C)
C
DHT

CID1 -> (A,B)
CID2 -> (A,C)
CID3 -> (C)
CID2
CID3



Bitswap:

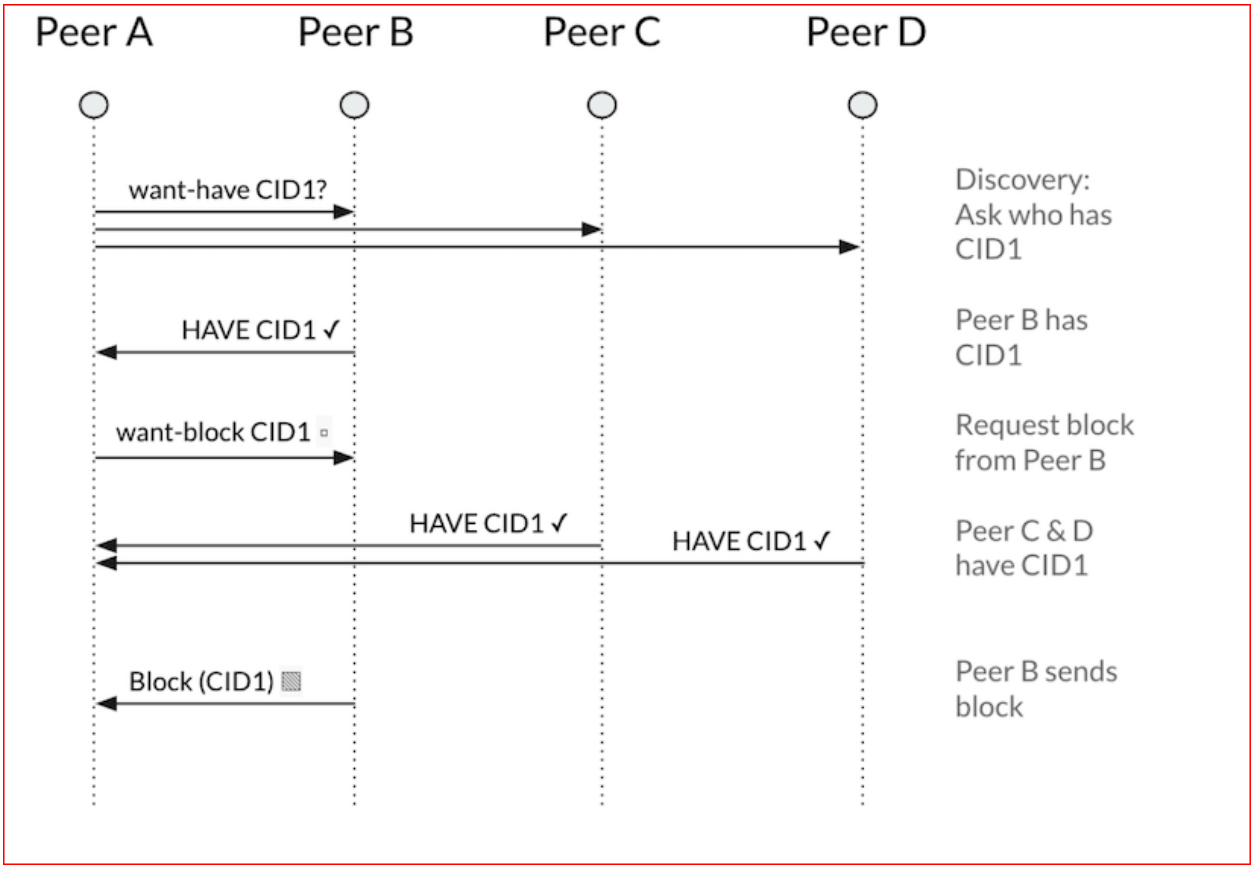
Bitswap is a **core module of** IPFS for exchanging blocks of data. It directs the requesting and sending of blocks to and from other peers in the network.

Bitswap has two main jobs:

- Acquire blocks requested by the client from the network.
- Send blocks in its possession to other peers who want them

IPFS breaks up files into **chunks of data called *blocks***. These blocks are identified by a content identifier (CID). When nodes running the **Bitswap protocol** want to fetch a file, they send out want-lists to other peers. A want-list is a list of CIDs for blocks a peer wants to receive.

```
Want-list {
  QmZtmD2qt6fJot32nabSP3CUjicnypEBz7bHVDhPQt9aAy, WANT,
  QmTudJSaoKxtbEnTddJ9vh8hbN84ZLVvD5pNpUaSbxwGoa, WANT,
  ...
}
```



Pining in IPFS:

Pinning is a very important concept in IPFS.
Pinning is the mechanism that allows you to tell **IPFS to always keep a given object somewhere** — the **default being your local node (pinning)**.

IPFS has a fairly aggressive caching mechanism that will keep an object **local for a short time** after you perform any IPFS operation on it, but these objects may get **garbage-collected regularly**.

To prevent that garbage collection, **simply pin the CID** you care about, or add it to **MFS**. Objects added through ipfs add are pinned **recursively by default**.

Default behavior for IPFS pinning is to pin to local IPFS node , but it's also possible to pin your files to a *remote pinning service*. These third-party services give you the opportunity to pin files not to your own local node, but to nodes that they operate.

Problems in IPFS:

- Need pinning services to pin to other IPFS nodes
- All IPFS nodes with pinned data might get shut down