InterPlanetary File System (IPFS)

What is IPFS?

NFTs shouldn’t use **URLs**. URLs are **not secure**

- **links** can be broken
- what’s **behind the link** can change over time

Here, the InterPlanetary File System (IPFS) comes into play.

IPFS is a distributed system for storing and accessing data. It is based on a peer-to-peer network and content-addressing.

A peer-to-peer network is one in which the participants (referred to as **peers** or nodes) communicate with one another **directly**, on more or less “equal footing”. They do not require a set of “servers” which behave completely differently from their “clients”, as is the case in the client / server model.

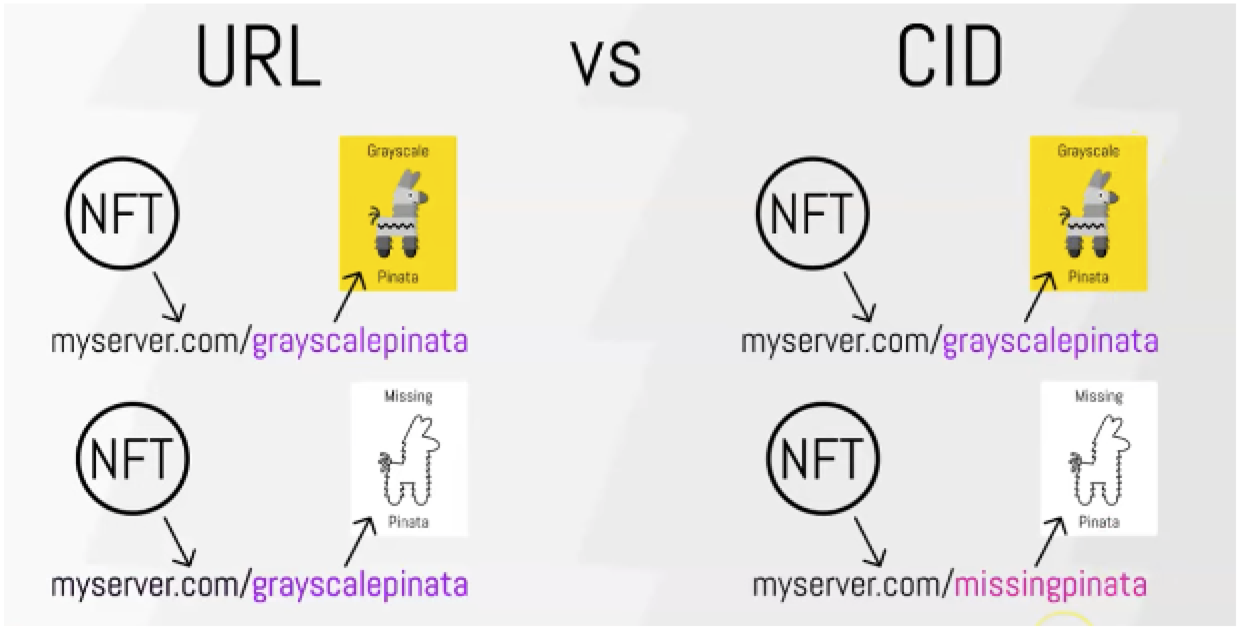
How does IPFS work?

Content Identifier (CID)

IPFS breaks up files into chunks of data called **blocks**. These blocks are identified by a content **identifier** (CID).

CID is a **content-addressed** identifier. It doesn't indicate where content is stored, but it forms a kind of address based on the content itself. CIDs are based on the **content’s** cryptographic **hash**. CIDs are short, **regardless of the size** of their underlying **content**.

URLs are different from CIDs:

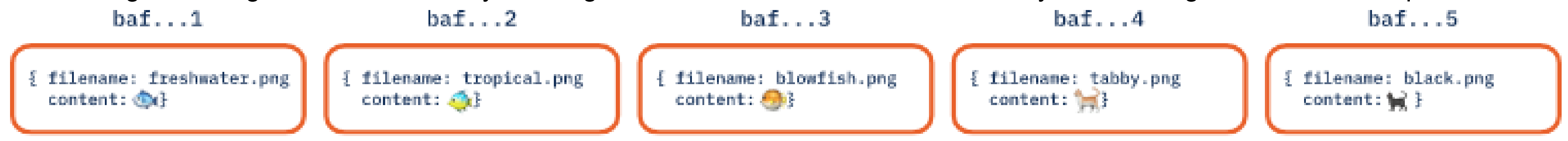

URL
VS
CID
NFT
Grayscale
Pinata
myserver.com/grayscalepinata
NFT
Grayscale
Pinata
myserver.com/grayscalepinata
NFT
Missing
Pinata
myserver.com/grayscalepinata
NFT
Missing
Pinata
myserver.com/missingpinata

IPFS lets you give CIDs to content and link that content together in a Merkle DAG.

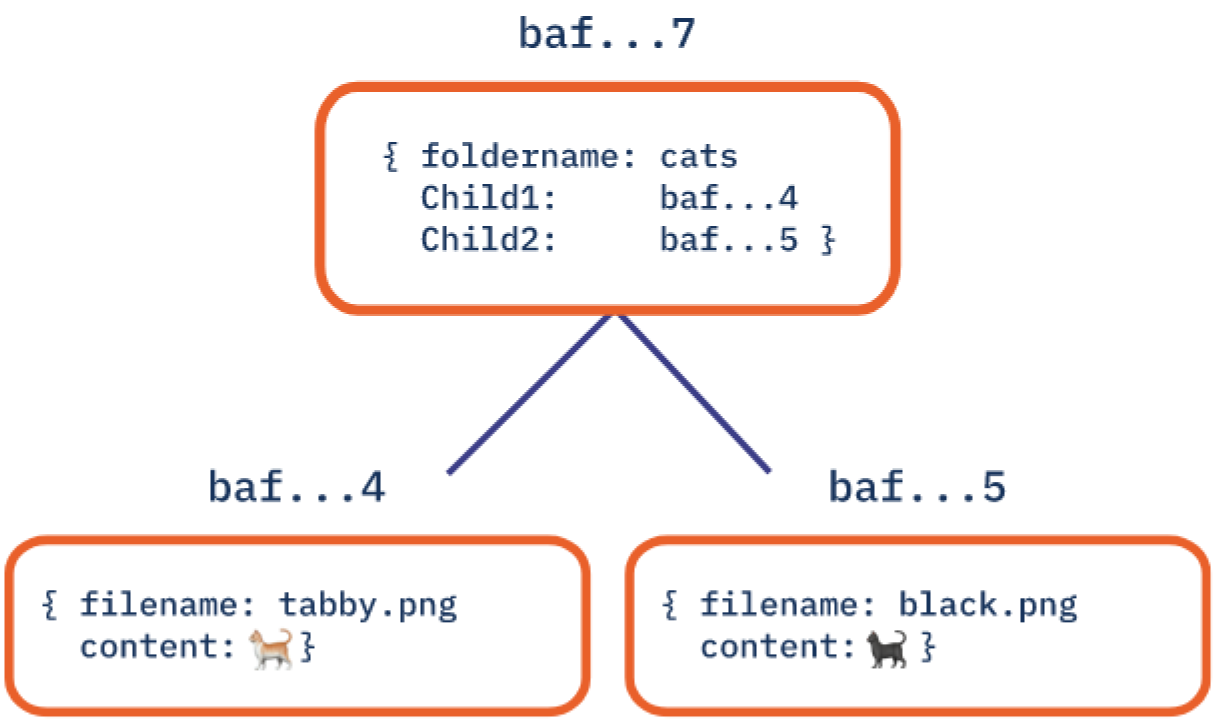
Merkle Directed Acyclic Graphs (DAGs)

In cryptography and computer science, a **hash tree** or **Merkle tree** is a tree in which every "**leaf**" (node) is labeled with the cryptographic hash of a **data** block, and every node that is **not a leaf** is labeled with the cryptographic hash of the labels of its **child** nodes. A hash tree allows efficient and secure verification of the contents of a large data structure.

We can begin building our Merkle DAG by creating its leaf nodes first, one node for every file, labeling each with its unique CID:


baf...1
{ filename: freshwater.png content: }
baf...2
{ filename: tropical.png content: }
baf...3
{ filename: blowfish.png content: }
baf...4
{ filename: tabby.png content: }
baf...5
{ filename: black.png content: }

The node structure for our intermediate nodes, the subdirectories, look like:



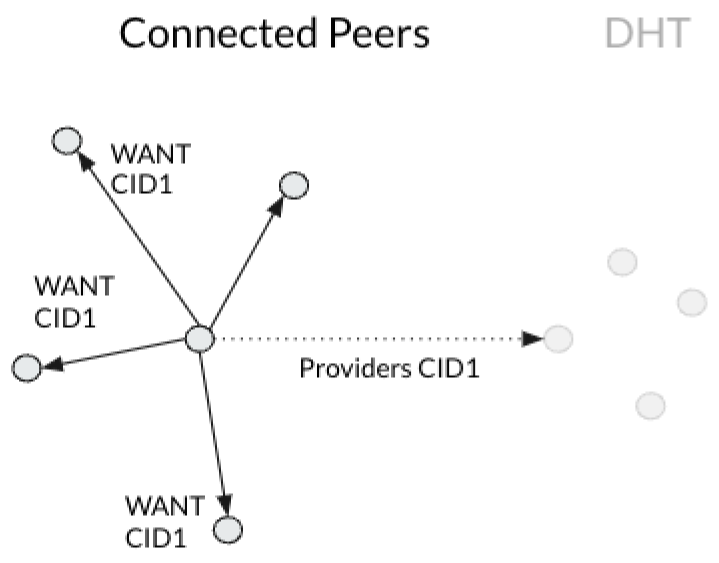
Now that we've derived representations for both types of nodes in our graph, we can continue to build the graph from the **bottom up**.

Any change to a node in a Merkle DAG is **propagated** to each of the changed node's **ancestors**.

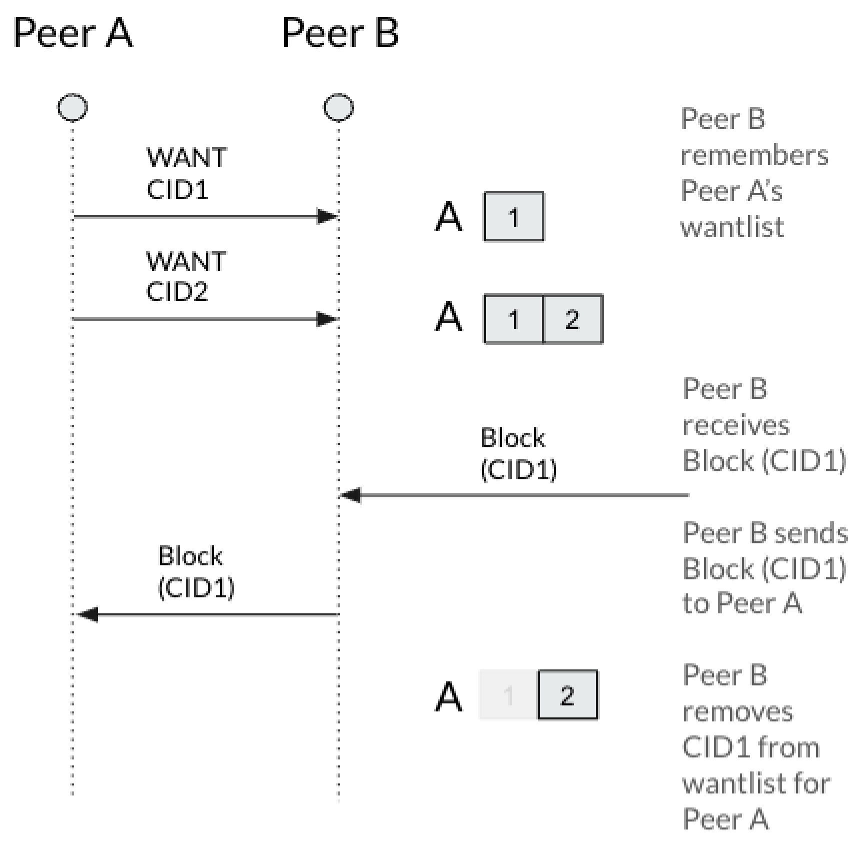
Bitswap

To **request** blocks from and **send** blocks to other peers, IPFS currently uses a module called Bitswap.

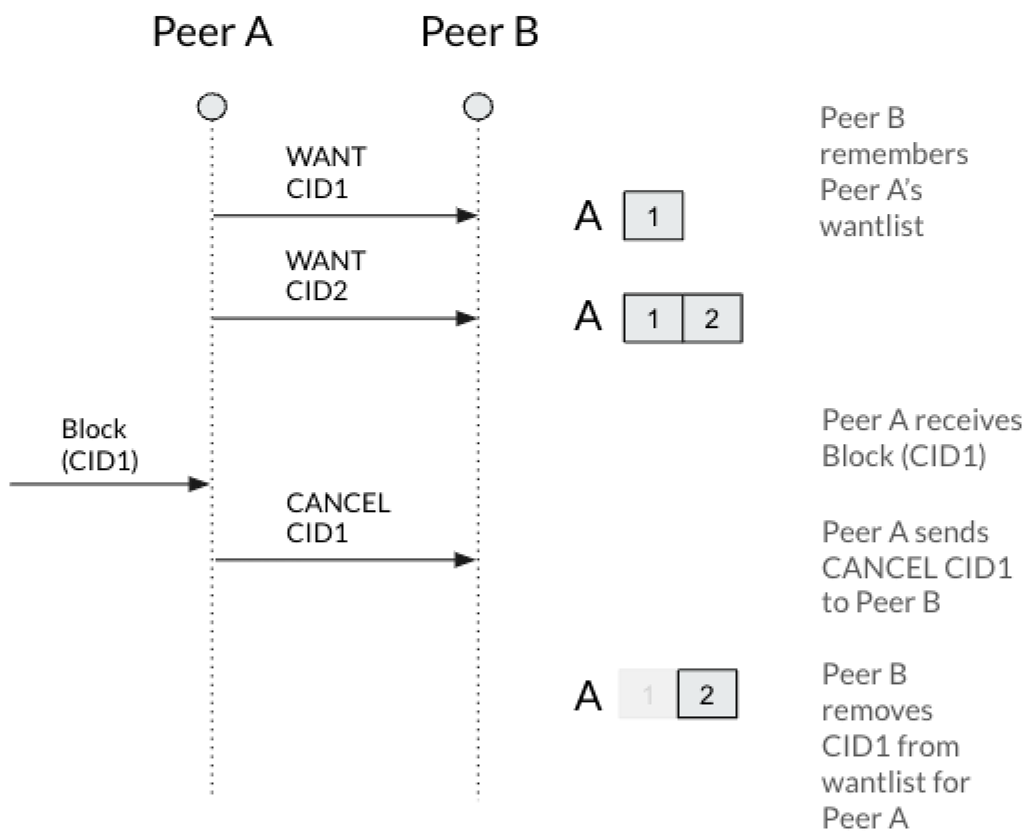
- Discovery
  - Broadcast WANT to connected Peers
  - If there's no response, ask DHT who has root CID



- Want lists
    - Each node remembers the want list for each of its connected peers

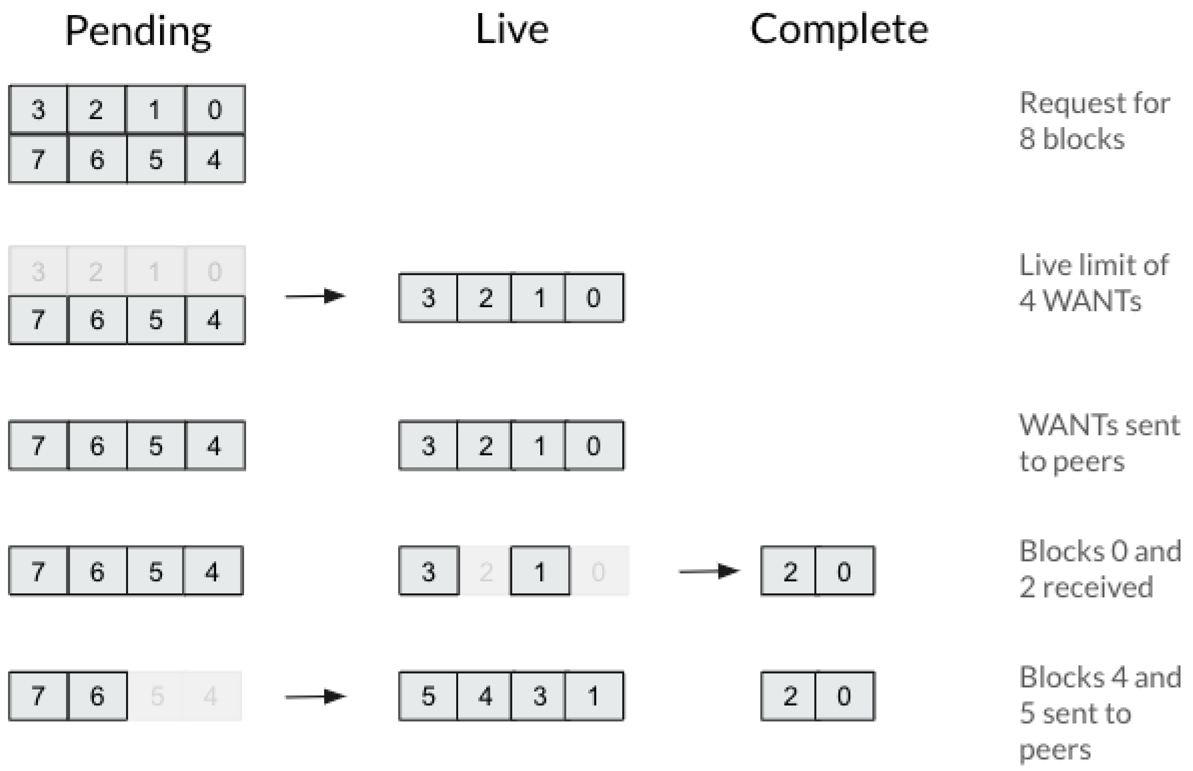

Peer A
Peer B
WANT CID1
A 1
Peer B remembers Peer A's wantlist
WANT CID2
A 1 2
Block (CID1)
Peer B receives Block (CID1)
Block (CID1)
Peer B sends Block (CID1) to Peer A
A 1 2
Peer B removes CID1 from wantlist for Peer A

- When a node receives a block it wanted, it sends a CANCEL message to all peers it has requested the block from



Peer A
Peer B
WANT CID1
WANT CID2
Block (CID1)
CANCEL CID1
A 1
A 1 2
A 1 2
Peer B remembers Peer A's wantlist
Peer A receives Block (CID1)
Peer A sends CANCEL CID1 to Peer B
Peer B removes CID1 from wantlist for Peer A

- Sessions
  - Peers who respond are added to the Session
  - Peers in the session are ordered by latency; request is split across peers
  - Rate Limiting: Sessions are limited to 32 “live” wants, a WANT request that has not yet received a response



There are issues caused by the basic protocol above, for example, rate-limited per-session rather than rate-limited per-peer.

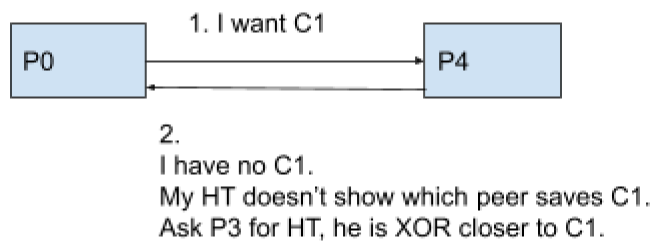
There is an extension of the Bitswap protocol to solve this problem (See more).

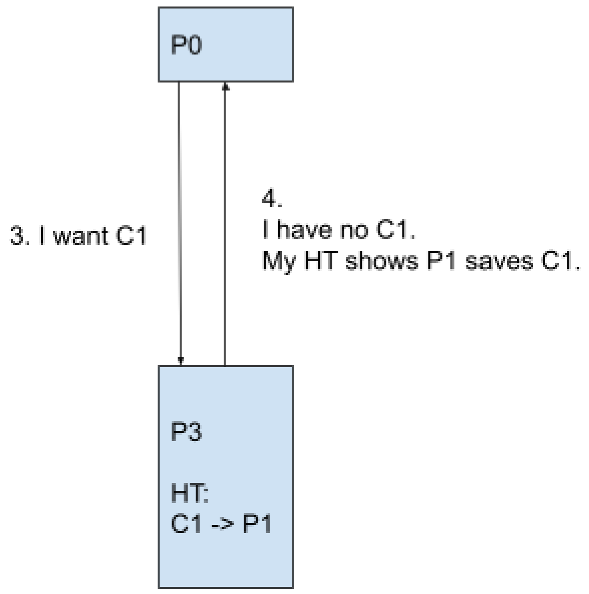
Distributed Hash Tables (DHTs)

A DHT is a **distributed** system for mapping keys to values. It is widely used to coordinate and maintain **metadata** about **peer-to-peer** systems. It provides **logically centralized access** to **content physically distributed** across the Internet.

In the case of IPFS, the **keys** are CIDs (of data blocks), and the **values** are the set of peers who have each block. Each peer has a globally unique “name”, PeerId, which also allows anyone to retrieve the **public key** for the identified peer.

CIDs and PeerIds have the same format and length. To assign key-value pairs to particular peers, it relies on the notion of an exclusive or (XOR) distance between two identifiers.


1. I want C1
P0
P4
2.
I have no C1.
My HT doesn’t show which peer saves C1.
Ask P3 for HT, he is XOR closer to C1.



For example, there are 4 peers: P0, P1, P2, P3.

P0 is connected to P4. P4 is connected to P3.

The DHT looks like: (C1 → P1, C2 → P2, C3 → P3, C4 → P4)

P0 asks P4 for C1.
P4 has no C1, suggests P3 may know who has C1.

P0 connects to P3 and asks for C1.
P3 has no C1, but his HT shows P1 has C1.

Imagine P0 connects to P1 and gets C1 successfully…
The libp2p project is the part of the IPFS ecosystem that provides the DHT.

*References

https://minima.global/blog/do-you-know-where-your-nft-art-is-stored
https://proto.school/tutorials
https://ipfs.io/
https://en.wikipedia.org/wiki/Merkle_tree
Technical overview of the Go implementation of Bitswap
https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf
https://docs.libp2p.io/