

# Swarm

Viktor Tron and Aron Fischer

August 30, 2016

## 1 A (very quick) overview of data in swarm

- Data in
- Data out

## 2 Incentive Structure

- A (simplified) history of WWW Incentivisation
- Incentivisation in Swarm

## 3 Status and Roadmap

- Status: Where are we at?
- Overview: Architecture
- Roadmap: What's next for Swarm?

# Outline

- 1 A (very quick) overview of data in swarm
  - Data in
    - Chunking
    - Encoding
    - Syncing
  - Data out
- 2 Incentive Structure
- 3 Status and Roadmap

## Data in

Saving data to the swarm.

# 1. Chunking: breaking up the data

# Chunks

Under the hood swarm does not deal in files but in *chunks*.



# Chunks

Under the hood swarm does not deal in files but in *chunks*.


- All data is broken into pieces of size 4kB: “chunks”.



# Chunks

Under the hood swarm does not deal in files but in *chunks*.

- All data is broken into pieces of size 4kB: “chunks”.

A “chunk:” 

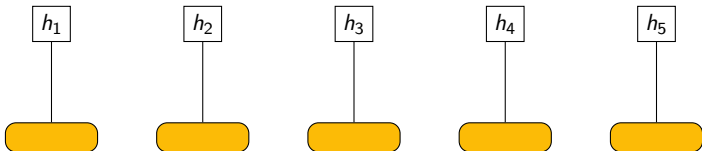




# Chunks

Under the hood swarm does not deal in files but in *chunks*.

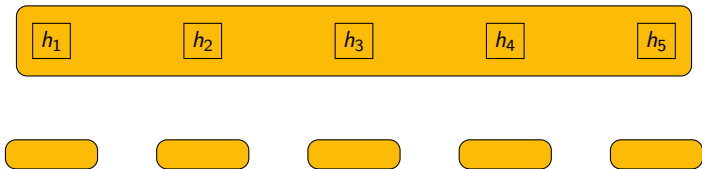
- All data is broken into pieces of size 4kB: “chunks”.
- Chunks are hashed and the hash is used as their ID/address.



## It's chunks all the way down...

Under the hood swarm does not deal in files but in *chunks*.

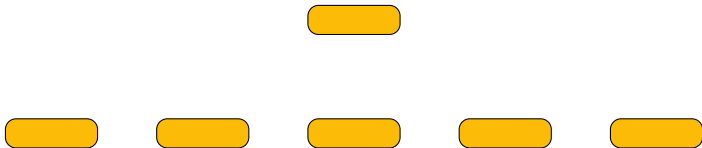
- All data is broken into pieces of size 4kB: “chunks” .
- Chunks are hashed and the hash is used as their ID/address.
- Chunk hashes are also packaged into 4kB chunks...



## It's chunks all the way down...

Under the hood swarm does not deal in files but in *chunks*.

- All data is broken into pieces of size 4kB: "chunks".
- Chunks are hashed and the hash is used as their ID/address.
- Chunk hashes are also packaged into 4kB chunks...



## 2. Encoding: Assembling the chunks into a merkle tree.

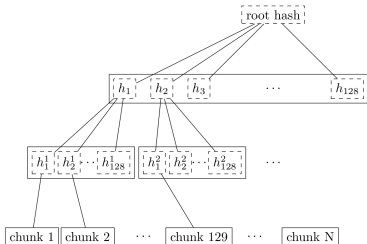
# Everything is merkle-ised.

When you want to save a document (or collection) to swarm, the swarm client

# Everything is merkle-ised.

When you want to save a document (or collection) to swarm, the swarm client

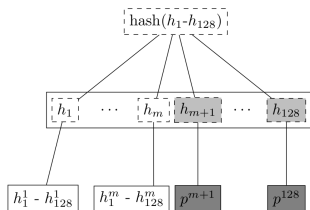
- 1 Assembles all chunks into a merkle tree...



# Everything is merkle-ised.

When you want to save a document (or collection) to swarm, the swarm client

- 1 Assembles all chunks into a merkle tree...
- 2 ...including extra redundancy (“parity chunks”).



# Everything is merkle-ised.

When you want to save a document (or collection) to swarm, the swarm client

- 1 Assembles all chunks into a merkle tree...
- 2 ...including extra redundancy (“parity chunks”).
- 3 Returns a single **root hash** for the entire collection.



# Everything is merkle-ised.

When you want to save a document (or collection) to swarm, the swarm client

- 1 Assembles all chunks into a merkle tree...
- 2 ...including extra redundancy (“parity chunks”).
- 3 Returns a single **root hash** for the entire collection.

**Note:** This means that you entire collection is accessible and retrievable from this single hash.

3. Syncing: getting the chunks to where they need to be.

# The syncing process.

The process of getting chunks to their storage destination is called **syncing**.

# The syncing process.

The process of getting chunks to their storage destination is called syncing.

Owner

# The syncing process.

The process of getting chunks to their storage destination is called syncing.

Owner

- Chunks are to be stored at the chunk ID.

chunk address

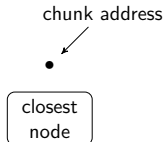


# The syncing process.

The process of getting chunks to their storage destination is called syncing.

Owner

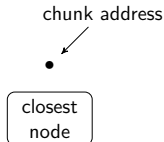
- Chunks are to be stored at **the node whose address is closest** to the chunk ID.



# The syncing process.

The process of getting chunks to their storage destination is called syncing.

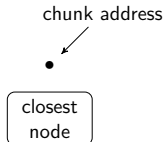
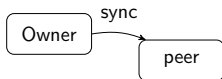
- Chunks are to be stored at the node whose address is closest to the chunk ID.
- Instead of directly connecting to that node and giving it the chunk in question, we pass the chunk along to one of our connected peers who is a little closer to the chunk ID than we are.



# The syncing process.

The process of getting chunks to their storage destination is called syncing.

- Chunks are to be stored at the node whose address is closest to the chunk ID.
- Instead of directly connecting to that node and giving it the chunk in question, we pass the chunk along to one of our connected peers who is a little closer to the chunk ID than we are. “Syncing”.

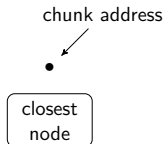
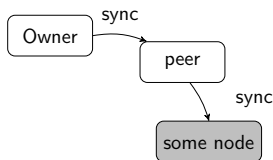




# The syncing process.

The process of getting chunks to their storage destination is called syncing.

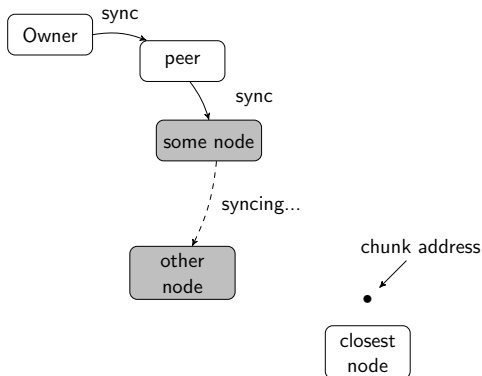
- Chunks are to be stored at the node whose address is closest to the chunk ID.
- Instead of directly connecting to that node and giving it the chunk in question, we pass the chunk along to one of our connected peers who is a little closer to the chunk ID than we are. "Syncing".
- This peer will repeat the process, thus the data is passed on



# The syncing process.

The process of getting chunks to their storage destination is called syncing.

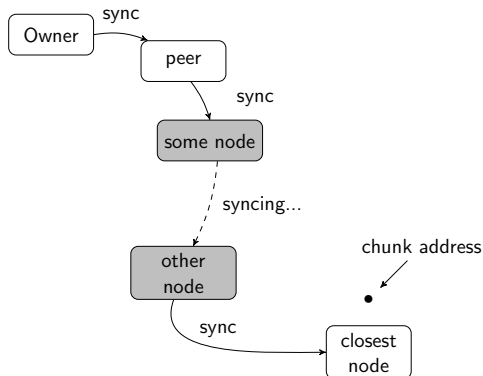
- Chunks are to be stored at the node whose address is closest to the chunk ID.
- Instead of directly connecting to that node and giving it the chunk in question, we pass the chunk along to one of our connected peers who is a little closer to the chunk ID than we are. "Syncing".
- This peer will repeat the process, thus the data is passed on from node



# The syncing process.

The process of getting chunks to their storage destination is called syncing.

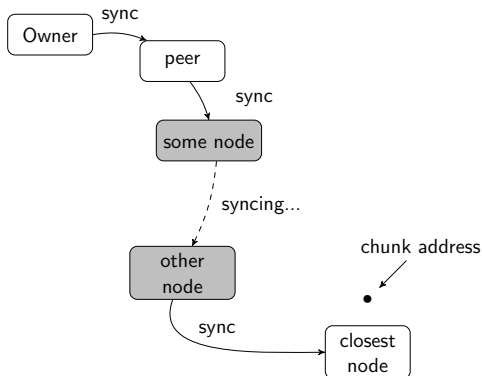
- Chunks are to be stored at the node whose address is closest to the chunk ID.
- Instead of directly connecting to that node and giving it the chunk in question, we pass the chunk along to one of our connected peers who is a little closer to the chunk ID than we are. "Syncing".
- This peer will repeat the process, thus the data is passed on from node to node.



# The syncing process.

The process of getting chunks to their storage destination is called syncing.

- Chunks are to be stored at the node whose address is closest to the chunk ID.
- Instead of directly connecting to that node and giving it the chunk in question, we pass the chunk along to one of our connected peers who is a little closer to the chunk ID than we are. "Syncing".
- This peer will repeat the process, thus the data is passed on from node to node.





## Data out

How to retrieve data stored in the swarm.

# Data Retrieval

When retrieving data from the swarm remember:

# Data Retrieval

When retrieving data from the swarm remember:

- All data is in chunks.



# Data Retrieval

When retrieving data from the swarm remember:

- All data is in chunks.
- All chunks are addressed by their hashes.

# Data Retrieval

When retrieving data from the swarm remember:

- All data is in chunks.
- All chunks are addressed by their hashes.
- Chunks are stored on nodes that are closest to the chunk hash.

Retriever

Retriever

chunk address



Retriever

chunk address



closest  
node

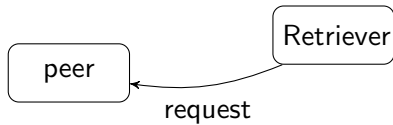
peer

Retriever

chunk address



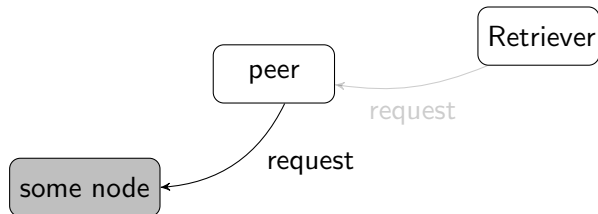
closest  
node



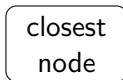
chunk address



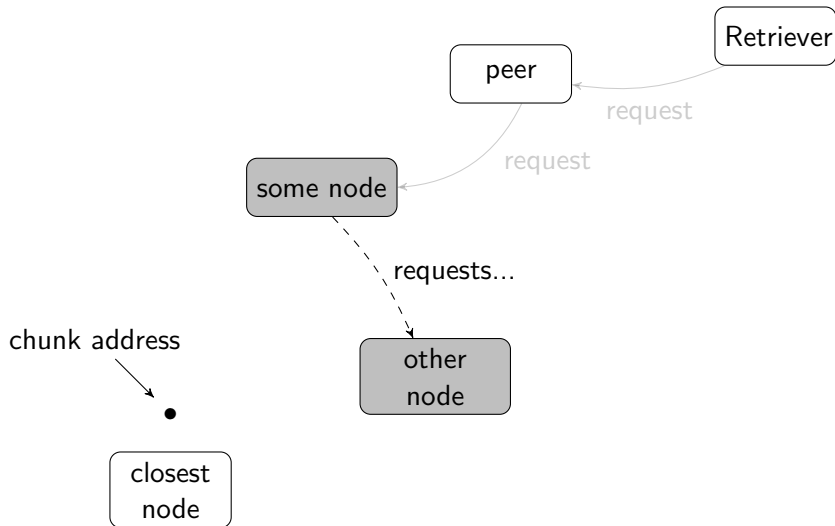
closest  
node

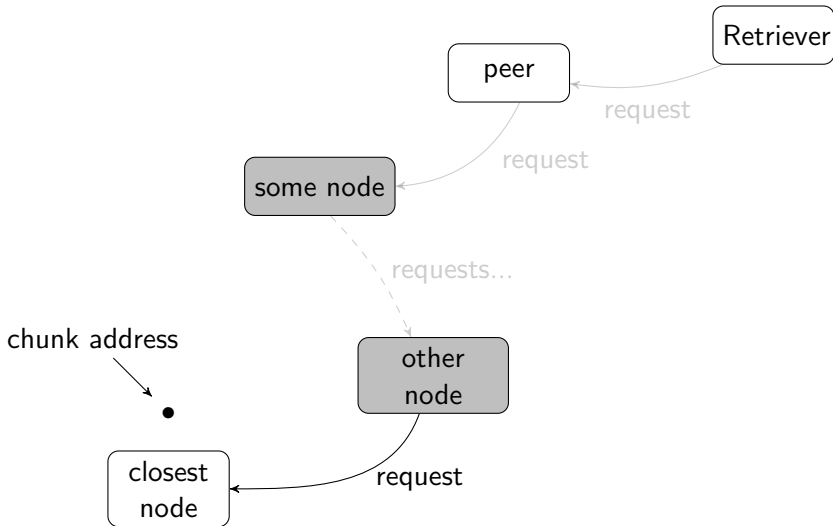


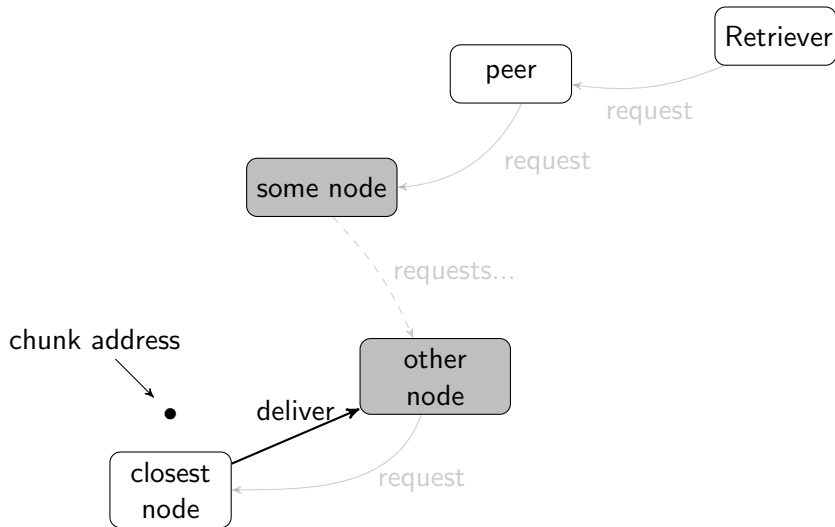
chunk address

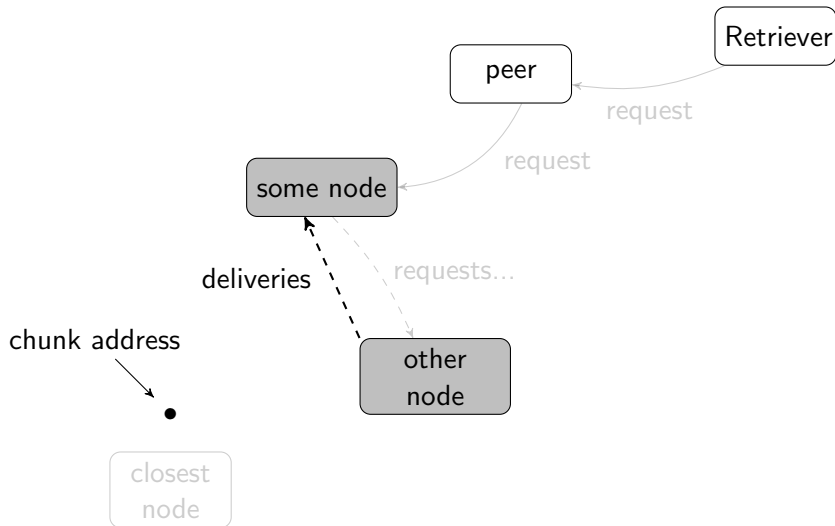


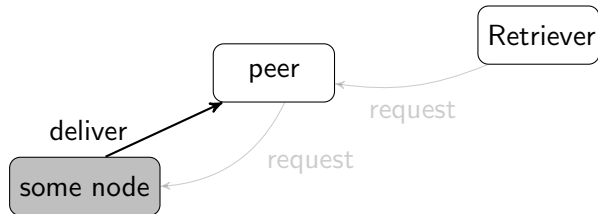












chunk address



other  
node

closest  
node



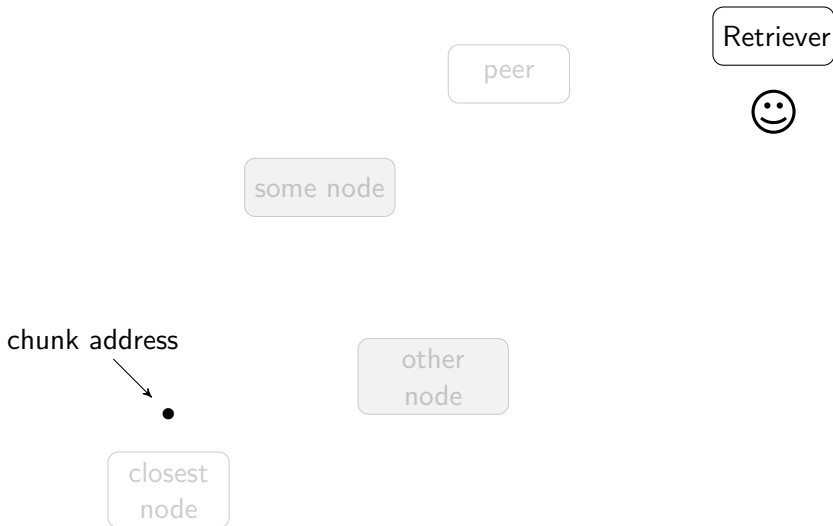
some node

chunk address



other node

closest node







# Outline

- 1 A (very quick) overview of data in swarm
- 2 Incentive Structure
  - A (simplified) history of WWW Incentivisation
    - Web 1.0
    - Web 2.0
    - Peer-to-peer (p2p)
  - Incentivisation in Swarm
    - Introduction
    - Bandwidth
    - Storage
- 3 Status and Roadmap

# Incentive structure of Web 1.0

# Incentive structure of Web 1.0

Back in the day...

# Incentive structure of Web 1.0

Back in the day...

- 1 Start up a webserver (or rent one)

# Incentive structure of Web 1.0

Back in the day...

- 1 Start up a webserver (or rent one)
- 2 Upload some content (FTP)

# Incentive structure of Web 1.0

Back in the day...

- 1 Start up a webserver (or rent one)
- 2 Upload some content (FTP)

**Content is unpopular**

# Incentive structure of Web 1.0

Back in the day...

- 1 Start up a webserver (or rent one)
- 2 Upload some content (FTP)

## **Content is unpopular**

- Pay running costs



# Incentive structure of Web 1.0

Back in the day...

- 1 Start up a webserver (or rent one)
- 2 Upload some content (FTP)

**Content is unpopular**      **Content becomes popular**

- Pay running costs





# Incentive structure of Web 1.0

Back in the day...

- 1 Start up a webserver (or rent one)
- 2 Upload some content (FTP)

**Content is unpopular**

- Pay running costs



**Content becomes popular**

- Bandwidth costs skyrocket



# Incentive structure of Web 1.0

Back in the day...

- 1 Start up a webserver (or rent one)
- 2 Upload some content (FTP)

## Content is unpopular

- Pay running costs



## Content becomes popular

- Bandwidth costs skyrocket
- Server crashes and goes offline.



# Incentive structure of Web 1.0

Back in the day...

- 1 Start up a webserver (or rent one)
- 2 Upload some content (FTP)

## Content is unpopular

- Pay running costs



## Content becomes popular

- Bandwidth costs skyrocket
- Server crashes and goes offline.



...but at least you owned your content.

# Incentive structure of Web 2.0

# Web 2.0

Today...

# Web 2.0

Today we just upload our content to “the cloud”.

# Web 2.0

Today we just upload our content to “the cloud”.

The cloud is:

- Cheap

## Web 2.0

Today we just upload our content to “the cloud”.

The cloud is:

- Cheap or even free



## Web 2.0

Today we just upload our content to “the cloud”.

The cloud is:

- Cheap or even “free”

## Web 2.0

Today we just upload our content to “the cloud”.

The cloud is:

- Cheap or even “free”
- Scalable

## Web 2.0

Today we just upload our content to “the cloud”.

The cloud is:

- Cheap or even “free”
- Scalable
- Reliable

## Web 2.0

Today we just upload our content to “the cloud”.

The cloud is:

- Cheap or even “free”
- Scalable
- Reliable

But...

## Web 2.0

Today we just upload our content to “the cloud”.

The cloud is:

- Cheap or even “free”
- Scalable
- Reliable

But...

- Content is **owned by the service providers.**

## Web 2.0

Today we just upload our content to “the cloud”.

The cloud is:

- Cheap or even “free”
- Scalable
- Reliable

But...

- Content is owned by the service providers.
- All users are **tracked and spied on**; providers profit off the data.

## Web 2.0

Today we just upload our content to “the cloud”.

The cloud is:

- Cheap or even “free”
- Scalable
- Reliable

But...

- Content is owned by the service providers.
- All users are tracked and spied on; providers profit off the data.
- Centralised control: **surveillance and censorship.**

What about p2p?



# Properties of the bittorrent network

Let's talk about Bittorrent

# Properties of the bittorrent network

Bittorrent

**Pros:**

- Content is distributed among peers.

# Properties of the bittorrent network

## Bittorrent

### Pros:

- Content is distributed among peers.
- Distribution scales automatically.

# Properties of the bittorrent network

## Bittorrent

### Pros:

- Content is distributed among peers.
- Distribution scales automatically.
- Hashing ensures data integrity.

# Properties of the bittorrent network

## Bittorrent

### Pros:

- Content is distributed among peers.
- Distribution scales automatically.
- Hashing ensures data integrity.
- No central point of failure (no servers).

# Properties of the bittorrent network

## Bittorrent

### Pros:

- Content is distributed among peers.
- Distribution scales automatically.
- Hashing ensures data integrity.
- No central point of failure (no servers).

### Cons:

- Downloads start slowly (high latency).

# Properties of the bittorrent network

## Bittorrent

### Pros:

- Content is distributed among peers.
- Distribution scales automatically.
- Hashing ensures data integrity.
- No central point of failure (no servers).

### Cons:

- Downloads start slowly (high latency).
- No incentive to provide content: “seeding”.





## Incentivisation in Swarm

We want all the benefits of p2p

We want all the benefits of p2p ...  
while using ethereum to pay and get paid

We want all the benefits of p2p ...  
while using ethereum to pay and get paid, aligning everyone's  
individual incentives with those of the network.

# Swarm Incentive System

# Swarm Incentive System

Bandwidth

Storage

# Swarm Incentive System

## Bandwidth

- Account for all bandwidth used (p2p).

## Storage

# Swarm Incentive System

## Bandwidth

- Account for all bandwidth used (p2p).
- Compensate nodes based on provided bandwidth.

## Storage



# Swarm Incentive System

## Bandwidth

- Account for all bandwidth used (p2p).
- Compensate nodes based on provided bandwidth.

## Storage

- Allow for long term storage of data.

# Swarm Incentive System

## Bandwidth

- Account for all bandwidth used (p2p).
- Compensate nodes based on provided bandwidth.

## Storage

- Allow for long term storage of data.
- Provide proper compensation for nodes storing the data.

# Dealing with Bandwidth

## Bandwidth

# Dealing with Bandwidth

Bandwidth accounting is peer-to-peer.

# Dealing with Bandwidth

Bandwidth accounting **has to be** peer-to-peer.

# Dealing with Bandwidth

Bandwidth accounting is peer-to-peer.



Me

## Dealing with Bandwidth

Bandwidth accounting is peer-to-peer.

Me

Peer

## Dealing with Bandwidth

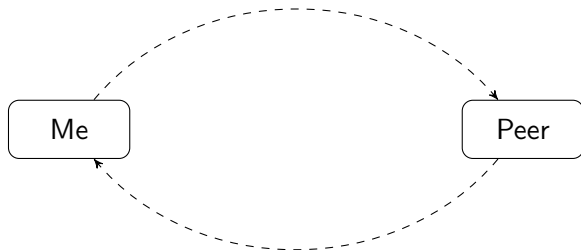
Bandwidth accounting is peer-to-peer.





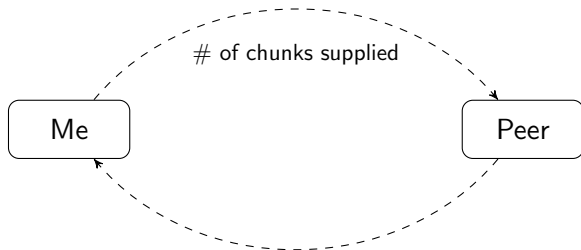
## Dealing with Bandwidth

Bandwidth accounting is peer-to-peer.



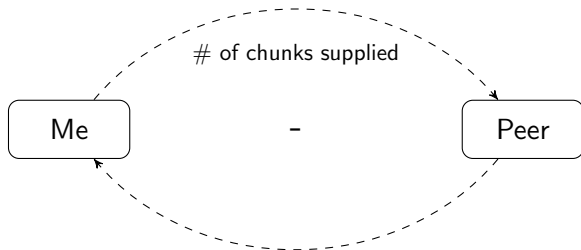
## Dealing with Bandwidth

Bandwidth accounting is peer-to-peer.



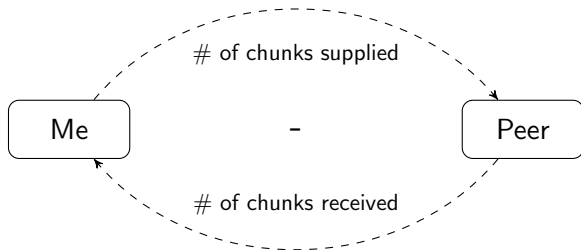
## Dealing with Bandwidth

Bandwidth accounting is peer-to-peer.



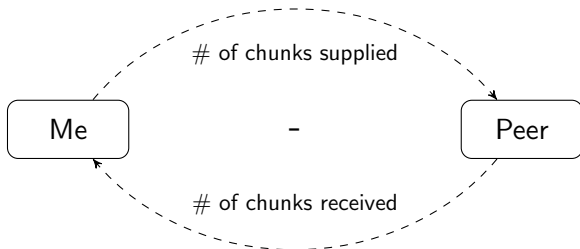
## Dealing with Bandwidth

Bandwidth accounting is peer-to-peer.



## Dealing with Bandwidth

Bandwidth accounting is peer-to-peer.



Note: Only involves Request/Deliver, not syncing.

## SWAP: **S**warm **A**ccounting **P**rotocol

# SWAP: Swarm Accounting Protocol

## The Swarm Accounting Protocol

# SWAP: Swarm Accounting Protocol

## The Swarm Accounting Protocol

- Keeps track of chunks provided/received (per peer)



# SWAP: Swarm Accounting Protocol

## The Swarm Accounting Protocol

- Keeps track of chunks provided/received (per peer)
- Can trade chunk-for-payment

# SWAP: Swarm Accounting Protocol

## The Swarm Accounting Protocol

- Keeps track of chunks provided/received (per peer)
- Can trade chunk-for-payment and chunk-for-chunk

# SWAP: Swarm Accounting Protocol

## The Swarm Accounting Protocol

- Keeps track of chunks provided/received (per peer)
- Can trade chunk-for-payment and chunk-for-chunk

## Payments

# SWAP: Swarm Accounting Protocol

## The Swarm Accounting Protocol

- Keeps track of chunks provided/received (per peer)
- Can trade chunk-for-payment and chunk-for-chunk

**Payments** use the swarm chequebook smart contract.

# SWAP: Swarm Accounting Protocol

## The Swarm Accounting Protocol

- Keeps track of chunks provided/received (per peer)
- Can trade chunk-for-payment and chunk-for-chunk

**Payments** use the swarm chequebook smart contract. Cheques are *cumulative* and are sent off-chain. Only the last cheque needs to be cashed. This saves transaction costs and blockchain bloat.

# SWAP: Swarm Accounting Protocol

## The Swarm Accounting Protocol

- Keeps track of chunks provided/received (per peer)
- Can trade chunk-for-payment and chunk-for-chunk

**Payments** use the swarm chequebook smart contract. Cheques are *cumulative* and are sent off-chain. Only the last cheque needs to be cashed. This saves transaction costs and blockchain bloat.

**Soon: Raiden payment-channel integration!**

# SWAP: Swarm Accounting Protocol

Big picture:

# SWAP: Swarm Accounting Protocol

Big picture:

- If you download a lot of content, you **pay your peers** for providing it.



# SWAP: Swarm Accounting Protocol

Big picture:

- If you download a lot of content, you pay your peers for providing it.
- If you host popular content, you will **earn fees from your peers** for making the content available.

# SWAP: Swarm Accounting Protocol

Big picture:

- If you download a lot of content, you pay your peers for providing it.
- If you host popular content, you will earn fees from your peers for making the content available.
- The Swarm is **auto-scaling!**

# SWAP: Swarm Accounting Protocol

Big picture:

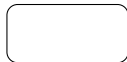
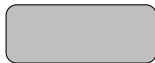
- If you download a lot of content, you pay your peers for providing it.
- If you host popular content, you will earn fees from your peers for making the content available.
- The Swarm is auto-scaling!
  - interplay of routing protocol and per-chunk payment between peers means that popular content will be widely distributed thereby increasing available bandwidth while decreasing latency



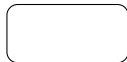
chunk address

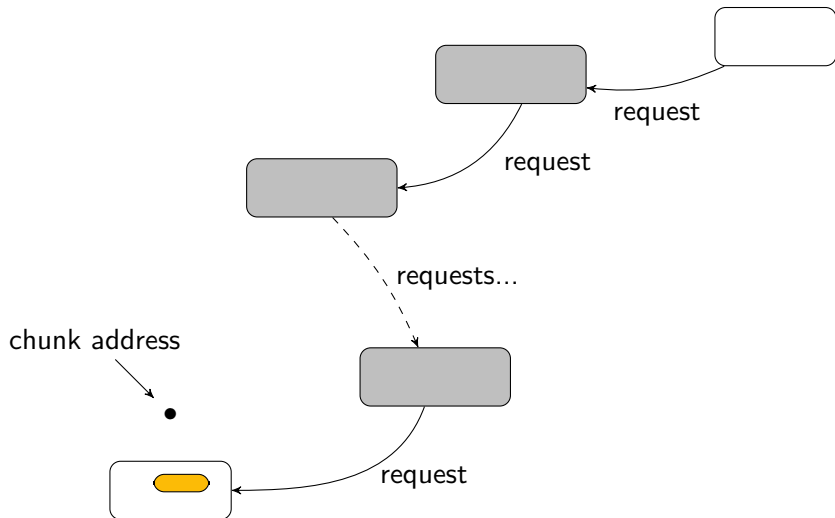


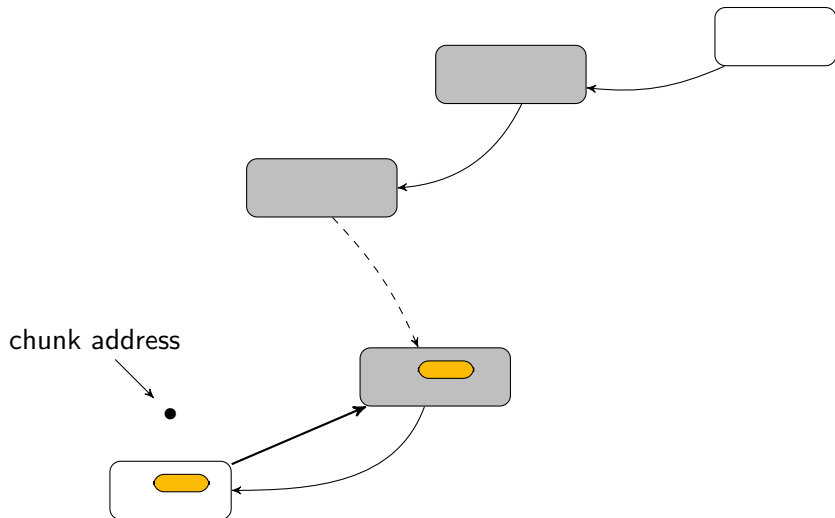
chunk address



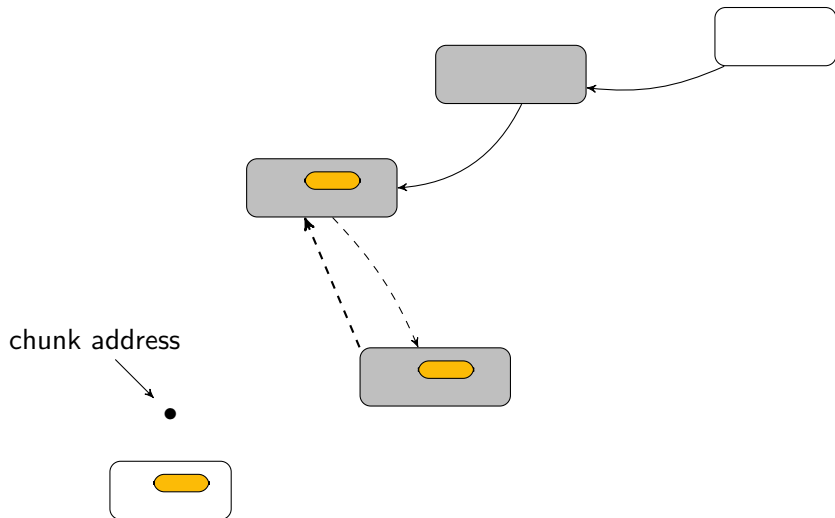
chunk address

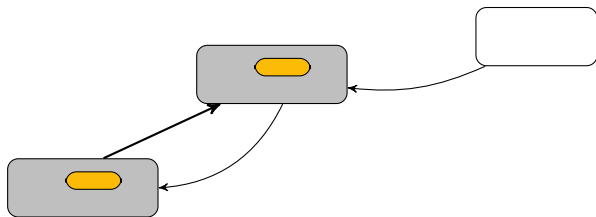




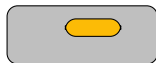
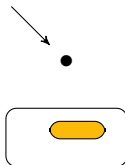


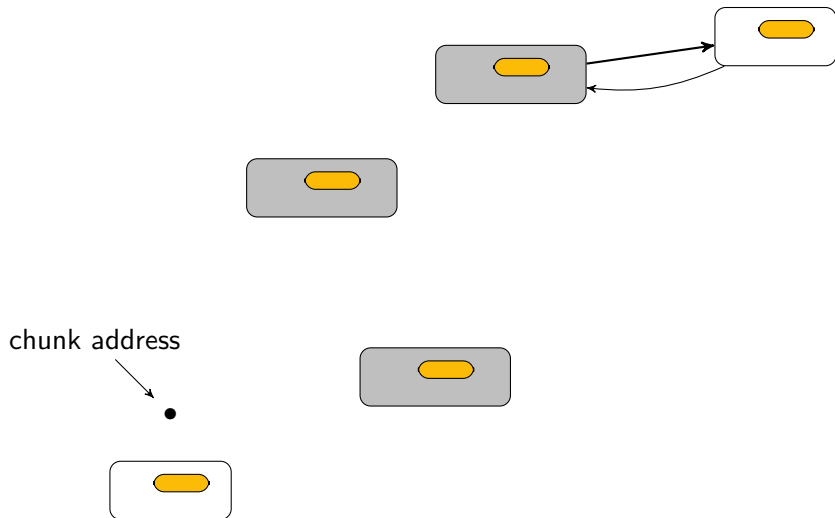




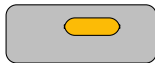
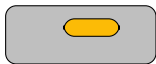


chunk address

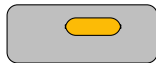
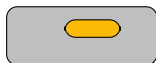




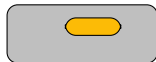
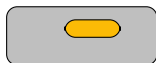
chunk address



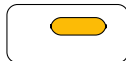
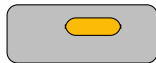
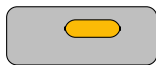
chunk address

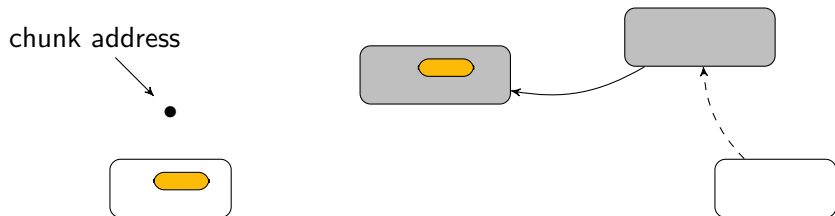


chunk address

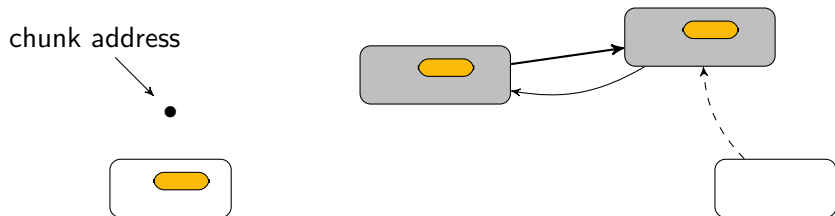


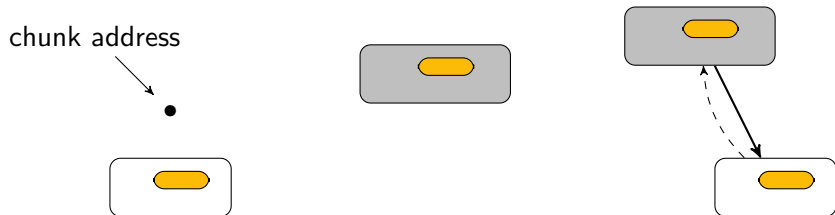
chunk address

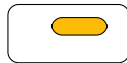
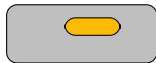




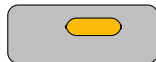
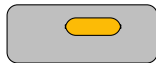








chunk address





# Dealing with storage

Storage

# Dealing with storage

Alas... we don't have the time

Ok. Very quickly:

Ok. Very quickly:  
The **SWEAR** contract takes security deposits.



Ok. Very quickly:

The **SWEAR** contract takes security deposits.

Swear registered nodes can sell promises of long term data storage.

Ok. Very quickly:

The **SWEAR** contract takes security deposits.

Swear registered nodes can sell promises of long term data storage.

The **SWINDLE** contract enforces these promises with  
proof-of-custody litigation engine.

# Outline

- 1 A (very quick) overview of data in swarm
- 2 Incentive Structure
- 3 Status and Roadmap
  - Status: Where are we at?
    - Swarm clients
    - Testnet
    - Incentives
  - Overview: Architecture
  - Roadmap: What's next for Swarm?
    - Scalability
    - Streaming Media

## Status: Swarm Clients

Currently there is a swarm enabled geth client only (github, swarm branch → develop).

## Status: Swarm Clients

Currently there is a swarm enabled geth client only (github, swarm branch → develop).

Soon swarm will be a standalone daemon.

## Status: Swarm Testnet

There is a testnet up and running.

`http://web3.download` is a direct http proxy to a swarm node

There is an ENS (name service) running on the swarm. For example you can access the page named 'swarm' at:

`http://web3.download/bzz:/swarm/`

# Features and Implementation

## Feature

- 1 Efficacy & Efficiency
- 2 Reliability
- 3 Data Integrity
- 4 User Authentication
- 5 Attribution

## Why?

## How?

# Features and Implementation

## Feature

- 1 **Efficacy & Efficiency**
- 2 Reliability
- 3 Data Integrity
- 4 User Authentication
- 5 Attribution

## How?

SWAP incentive structure:  
popular content is readily  
available  
Payment-channel integration  
(Raiden): fast and cheap

## Why?

So that all content is accessible,  
popular content has low latency  
(comp. Web2)



# Features and Implementation

## Feature

- 1 Efficacy & Efficiency
- 2 **Reliability**
- 3 Data Integrity
- 4 User Authentication
- 5 Attribution

## How?

- Redundant storage
- Built-in erasure coding
- Proof-of-custody based insurance
- Automatic scan-and-repair

## Why?

So that stored content does not get lost

# Features and Implementation

## Feature

- 1 Efficacy & Efficiency
- 2 Reliability
- 3 **Data Integrity**
- 4 User Authentication
- 5 Attribution

## How?

- Everything is stored in Merkle trees
- Merkle proof compatible file 'manifests'
- Chunk traversal follows Merkle-proof logic

## Why?

So that we aren't given bad data.

# Features and Implementation

## Feature

- 1 Efficacy & Efficiency
- 2 Reliability
- 3 Data Integrity
- 4 User Authentication**
- 5 Attribution

## How?

Data is encrypted and signed.  
Identity is managed on user side.

## Why?

So that only people with proper  
authorisation can access content

# Features and Implementation

## Feature

- 1 Efficacy & Efficiency
- 2 Reliability
- 3 Data Integrity
- 4 User Authentication
- 5 **Attribution**

## How?

ENS + smart contracts

## Why?

So that content producers are recognised

# Features and Implementation

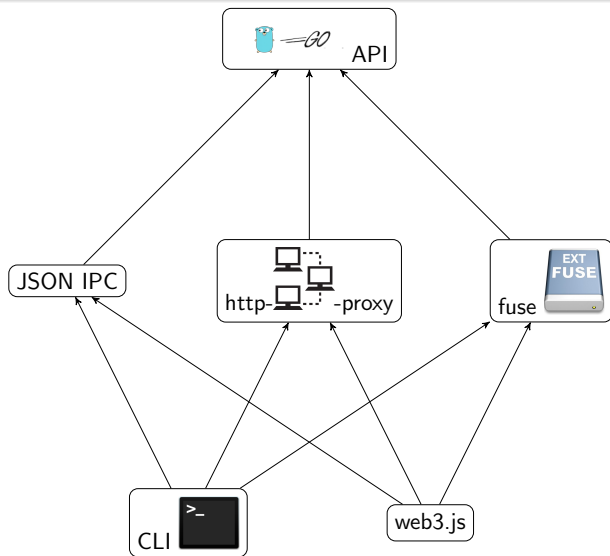
## Feature

- 1 Efficacy & Efficiency
- 2 Reliability
- 3 Data Integrity
- 4 User Authentication
- 5 Attribution

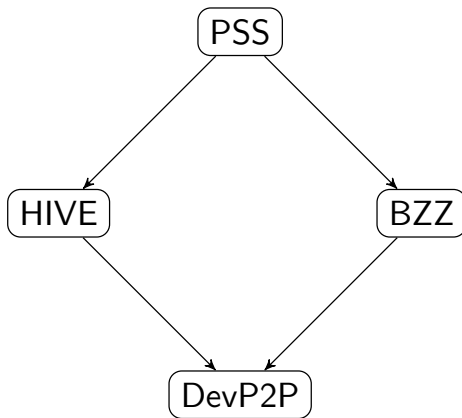
## How?

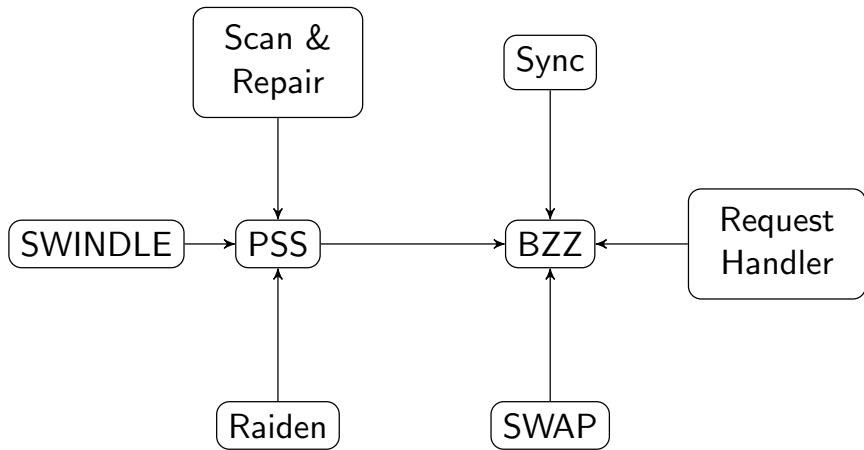


## Why?



# Network Layer







What else?

- Payment-channel integration into swarm (Raiden)

## What else?

- Payment-channel integration into swarm (Raiden)
- Swarm-routing integration into Raiden

## What else?

- Payment-channel integration into swarm (Raiden)
- Swarm-routing integration into Raiden
- Streaming Video – project swatch to stream devcon over swarm

## What else?

- Payment-channel integration into swarm (Raiden)
- Swarm-routing integration into Raiden
- Streaming Video – project swatch to stream devcon over swarm
- Stable testnet and scalability testing (azure cloud)

## What else?

- Payment-channel integration into swarm (Raiden)
- Swarm-routing integration into Raiden
- Streaming Video – project swatch to stream devcon over swarm
- Stable testnet and scalability testing (azure cloud)
- Dropbox and archiver
- ...and so much more.



