The rise of the third web

Viktor Tron and Aron Fischer

September 8, 2016

ethereum

whisper

swarm

Combining the power of the blockchain with a decentralised content storage and delivery network.

Outline

Data out

How to retrieve data stored in the swarm.

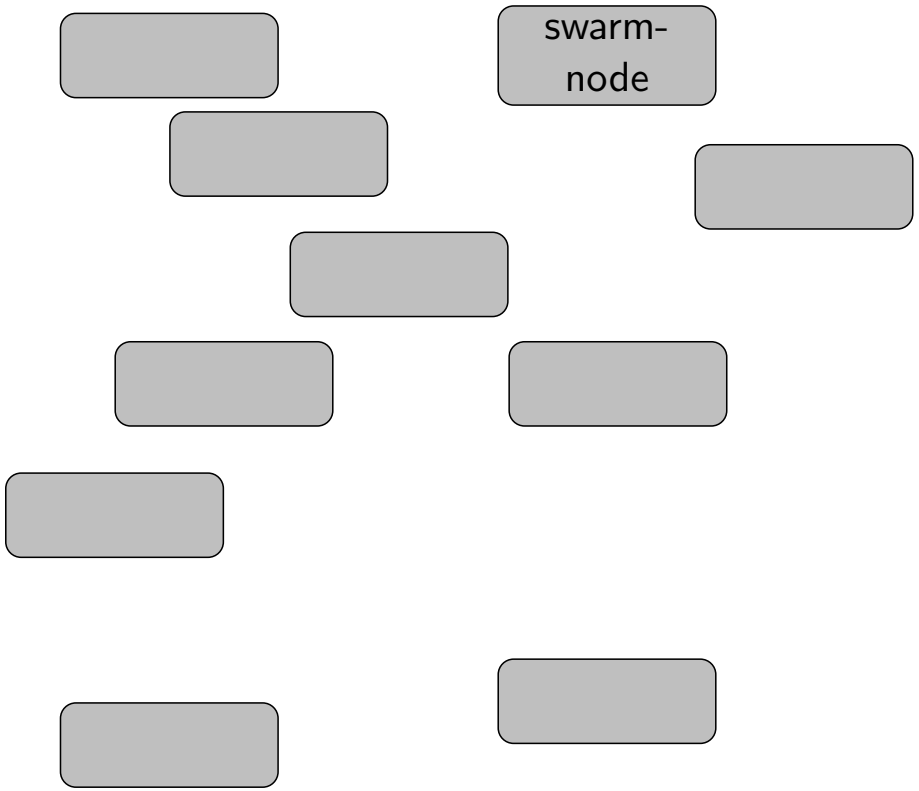
Data Retrieval

Everything on the swarm network has an **id**, every chunk of data, every node (even you!). The **id** also functions as an **address**.

Data Retrieval

Everything on the swarm network has an **id**, every chunk of data, every node (even you!). The **id** also functions as an **address**.

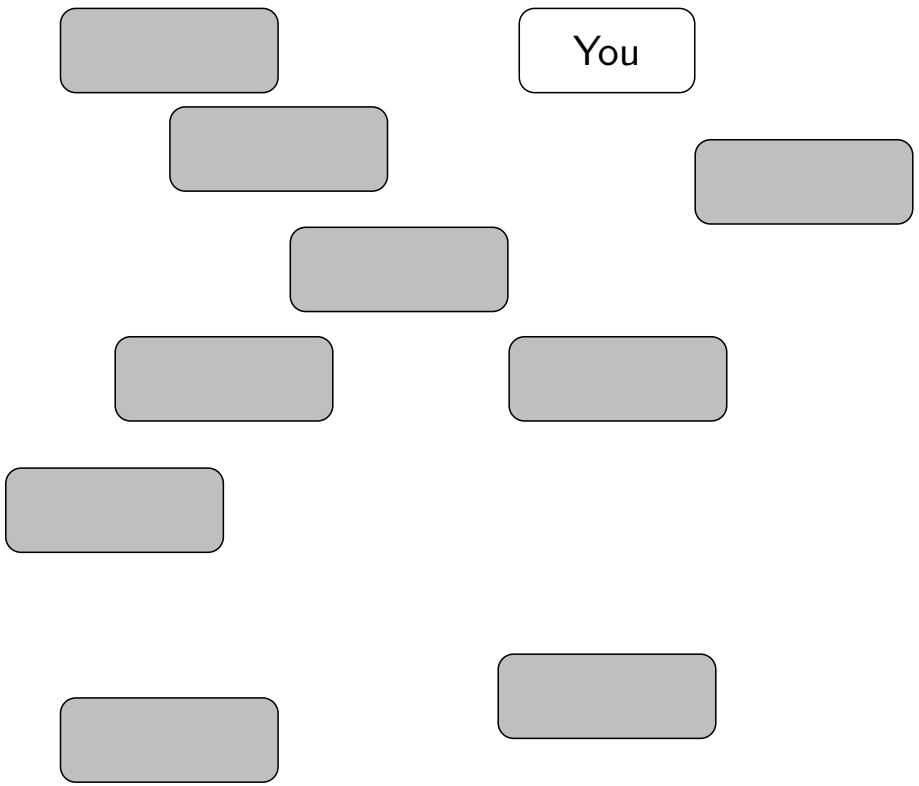
The Swarm Network:



Data Retrieval

Everything on the swarm network has an **id**, every chunk of data, every node (even you!). The **id** also functions as an **address**.

The Swarm Network:

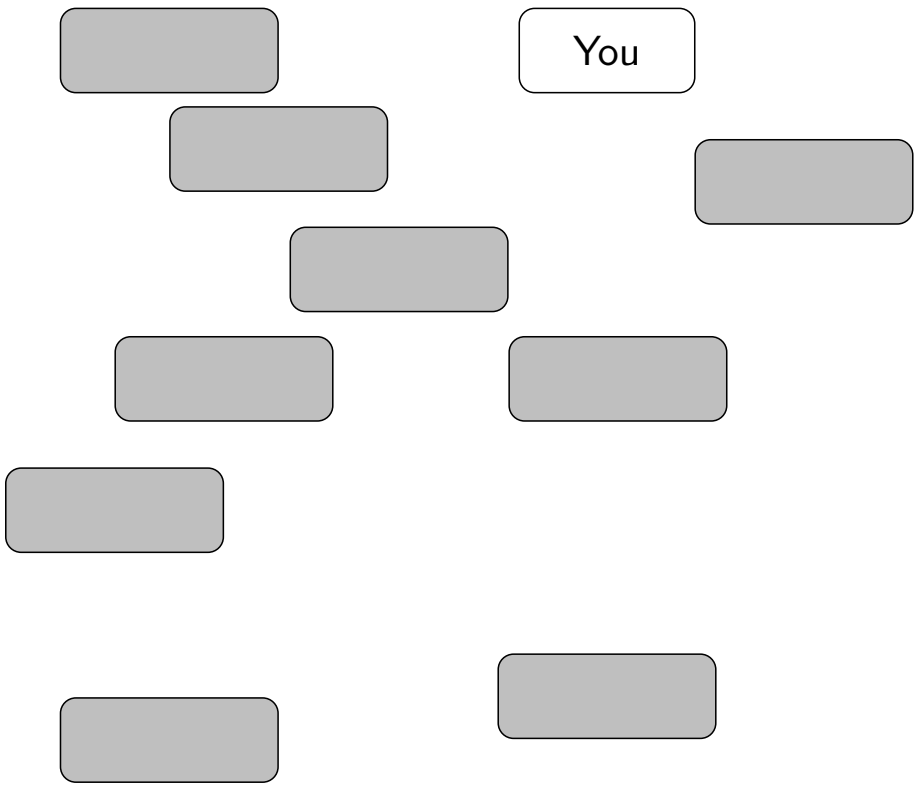


Data Retrieval

Everything on the swarm network has an id, every chunk of data, every node (even you!). The id also functions as an address.

When your swarm-enabled dapp wishes to retrieve "awesome-swarm-slides.pdf",

The Swarm Network:

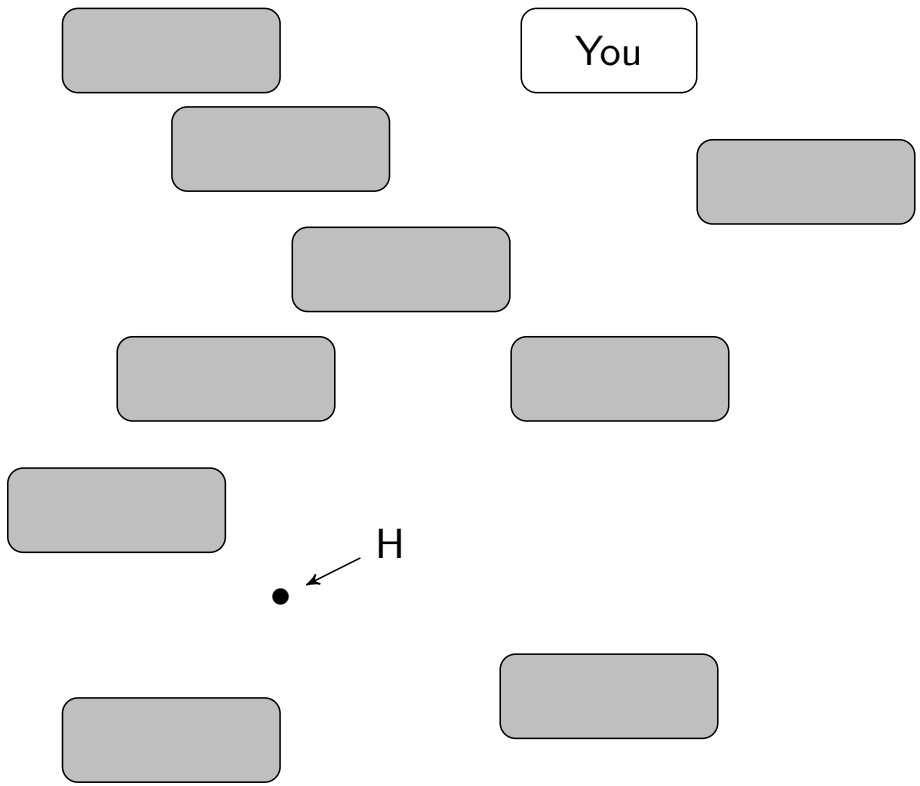


Data Retrieval

Everything on the swarm network has an id, every chunk of data, every node (even you!). The id also functions as an address.

When your swarm-enabled dapp wishes to retrieve "awesome-swarm-slides.pdf", it will attempt to retrieve it via its id "**H**".

The Swarm Network:



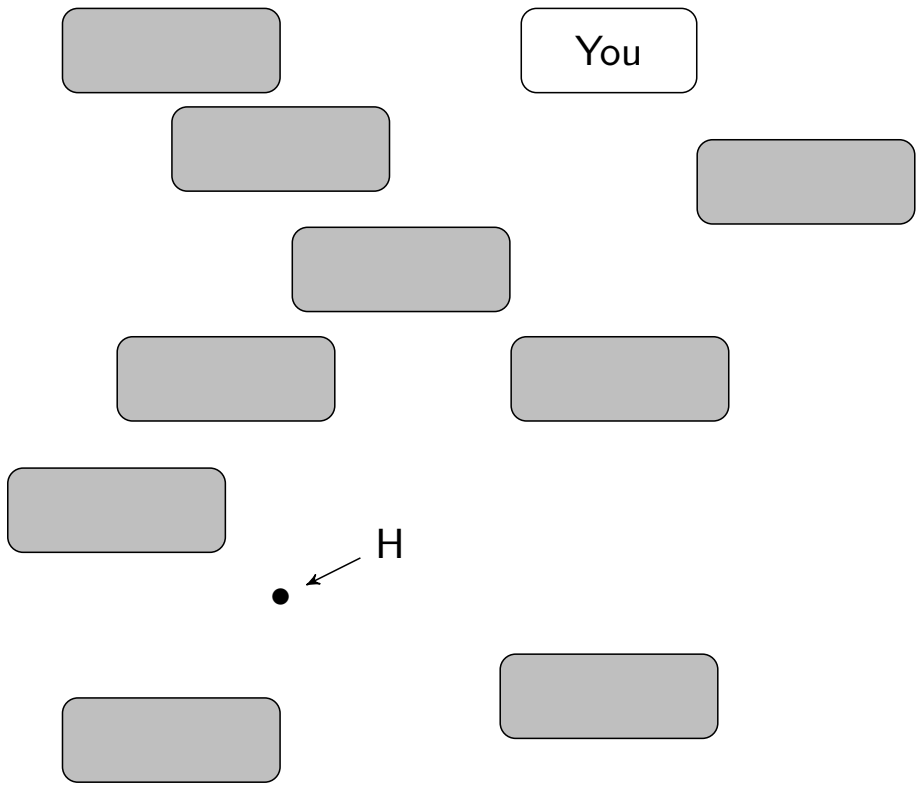
Data Retrieval

Everything on the swarm network has an id, every chunk of data, every node (even you!). The id also functions as an address.

When your swarm-enabled dapp wishes to retrieve "awesome-swarm-slides.pdf", it will attempt to retrieve it via its id "**H**".

In swarm, the content with address "**H**" is stored with the node whose own address is *closest* to **H**.

The Swarm Network:



Data Retrieval

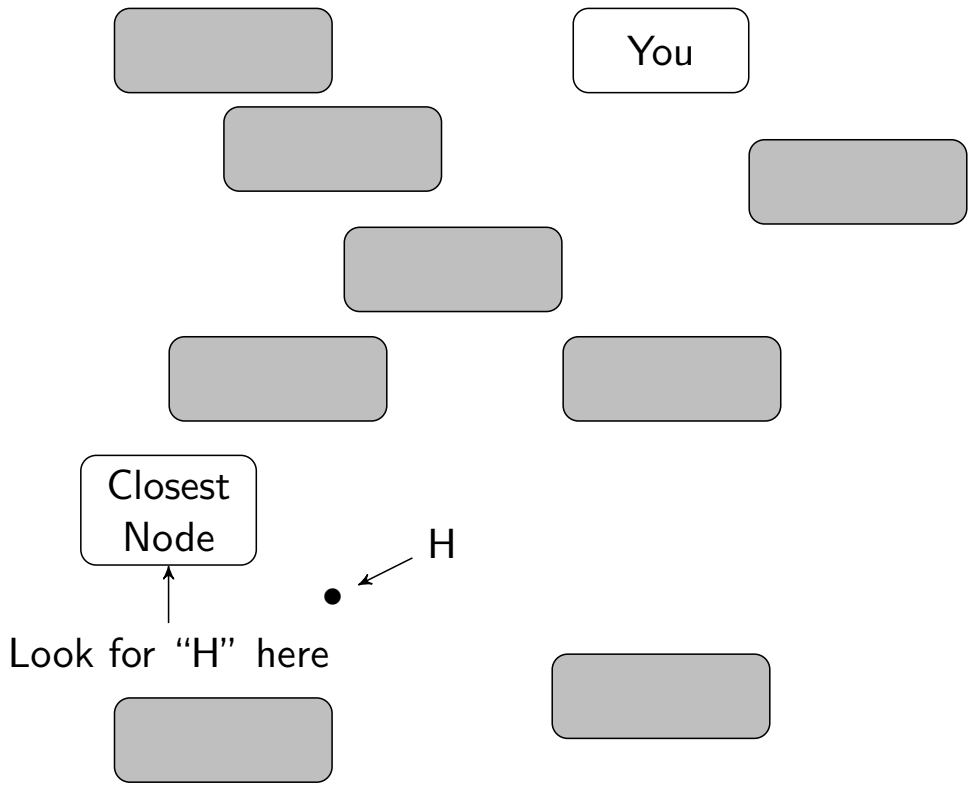
Everything on the swarm network has an id, every chunk of data, every node (even you!). The id also functions as an address.

When your swarm-enabled dapp wishes to retrieve "awesome-swarm-slides.pdf", it will attempt to retrieve it via its id "**H**".

In swarm, the content with address "**H**" is stored with the node whose own address is *closest* to **H**.

Swarm's **Retrieval Process** is responsible for deliviering the data to you.

The Swarm Network:

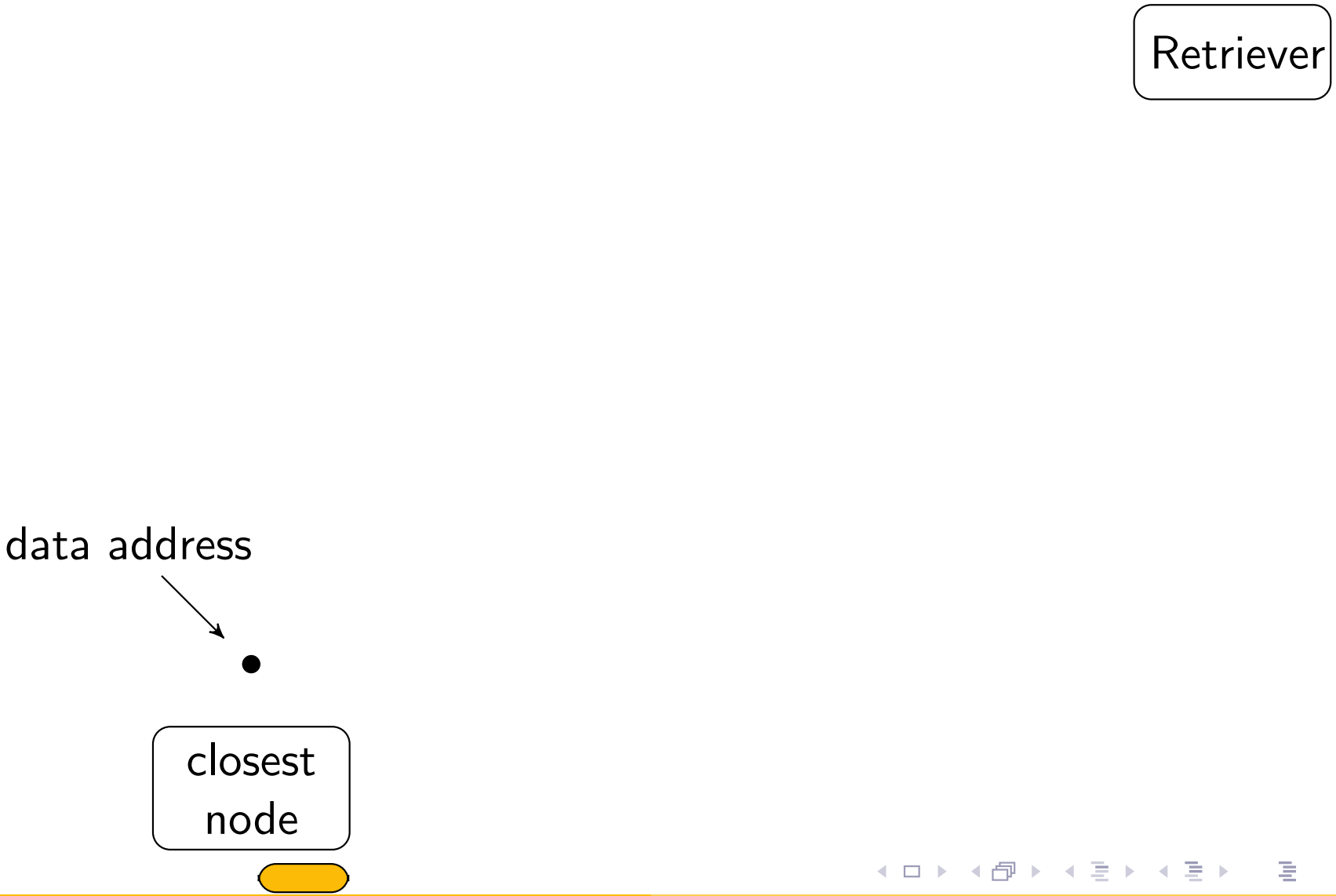


Swarm Retrieval Process

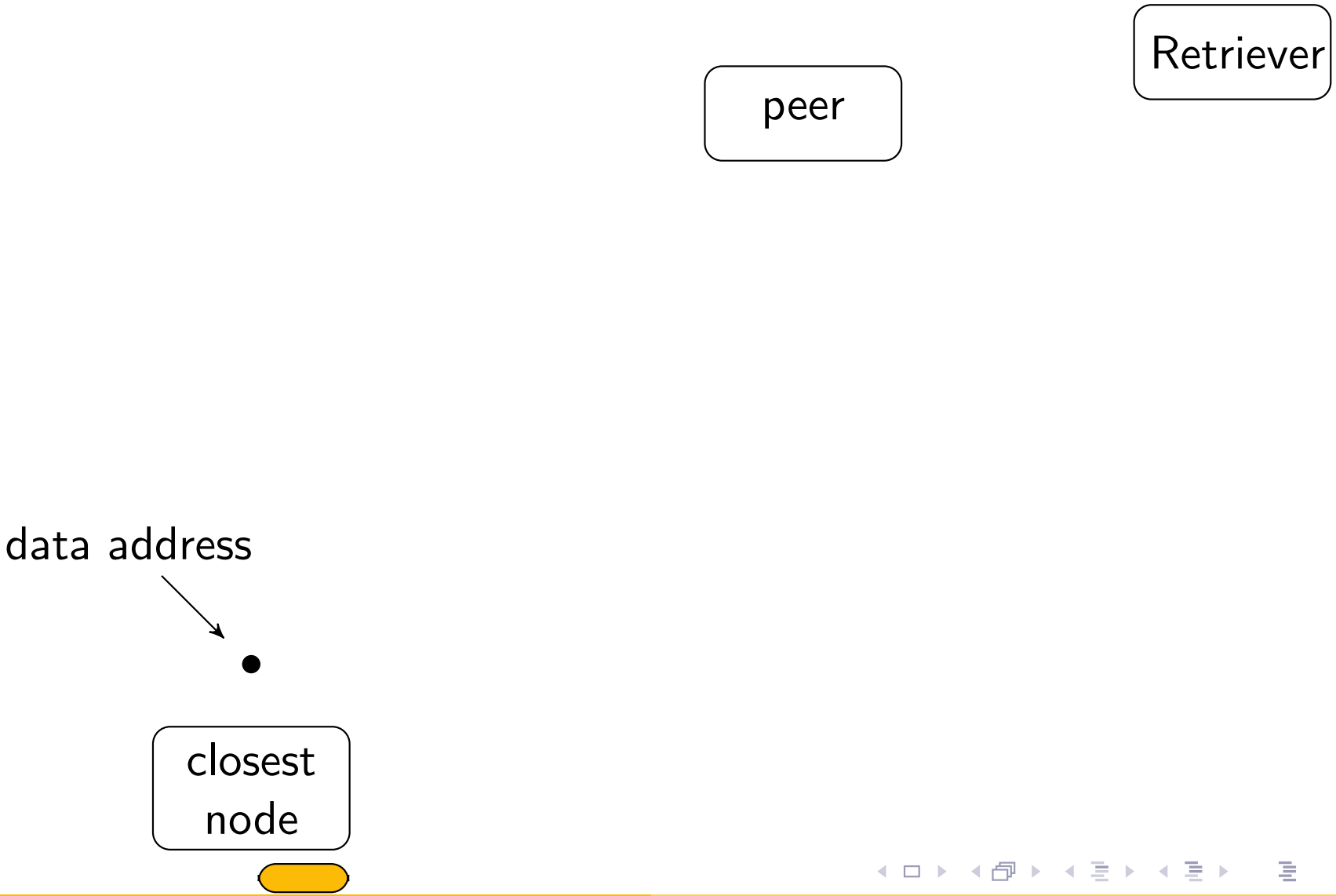
Retriever

Swarm Retrieval Process

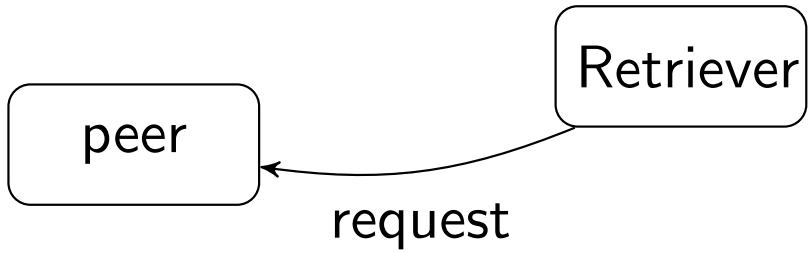
Swarm Retrieval Process

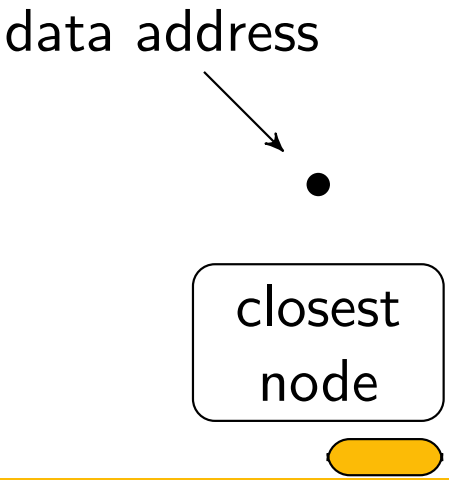


Swarm Retrieval Process

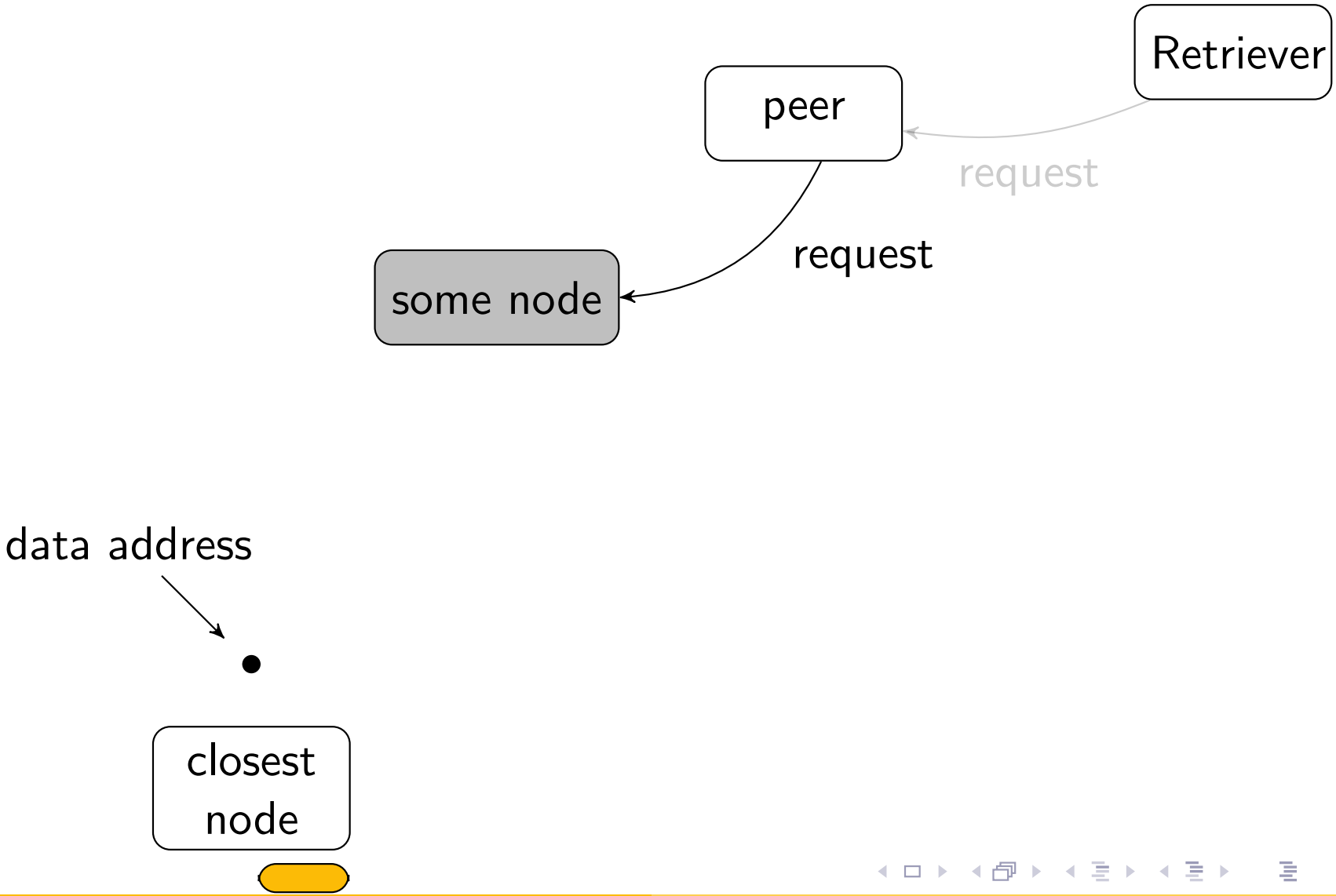


Swarm Retrieval Process

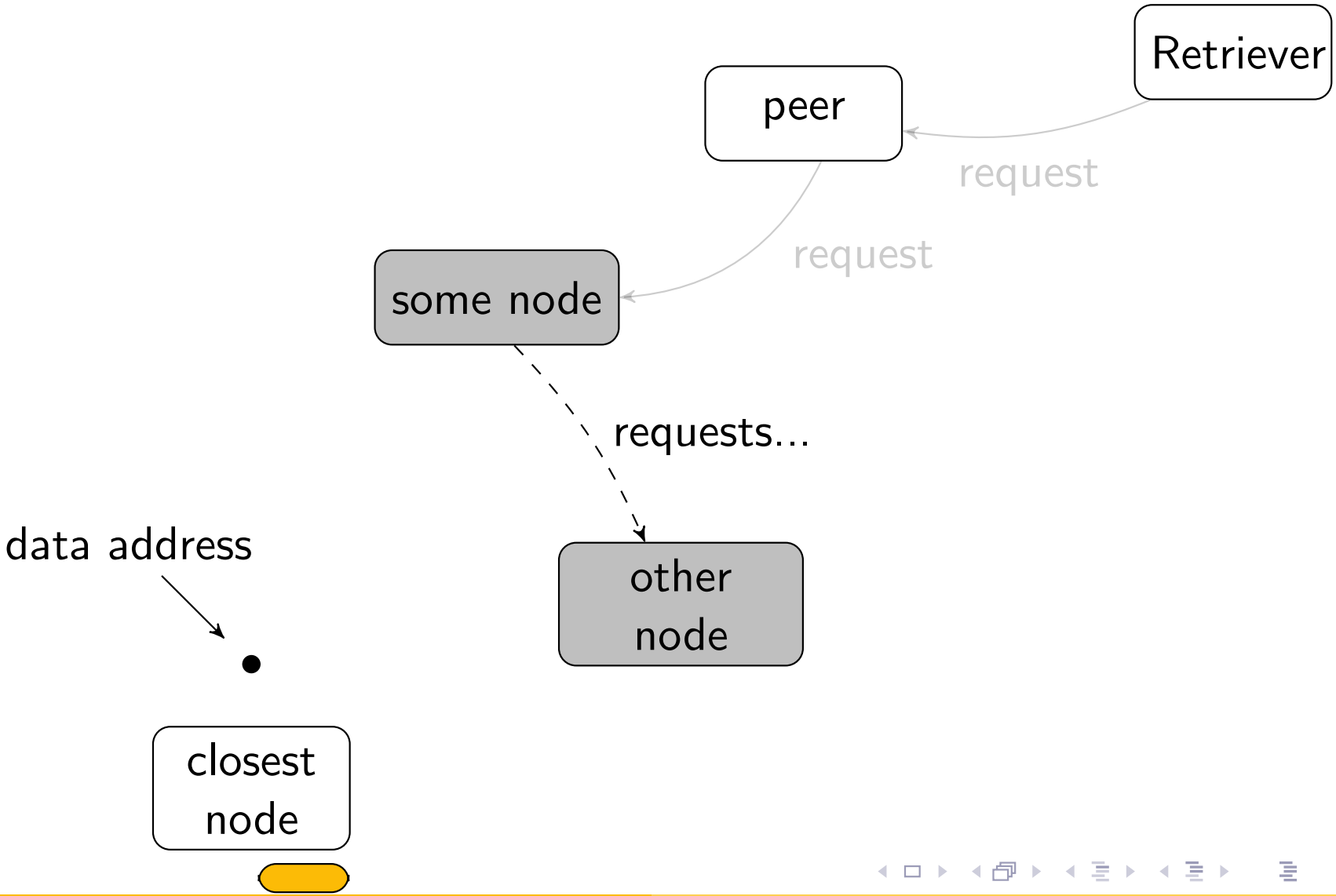




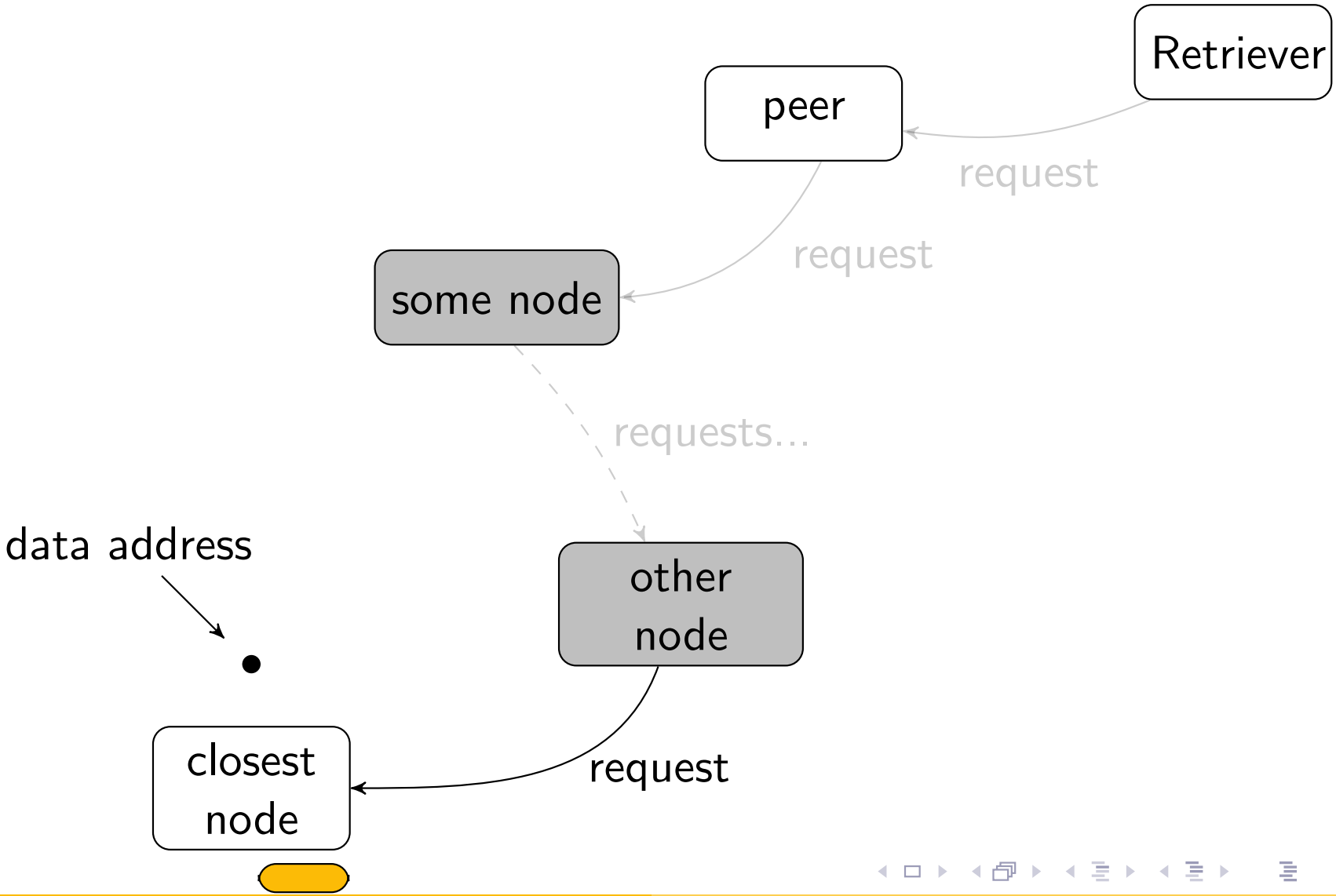
Swarm Retrieval Process



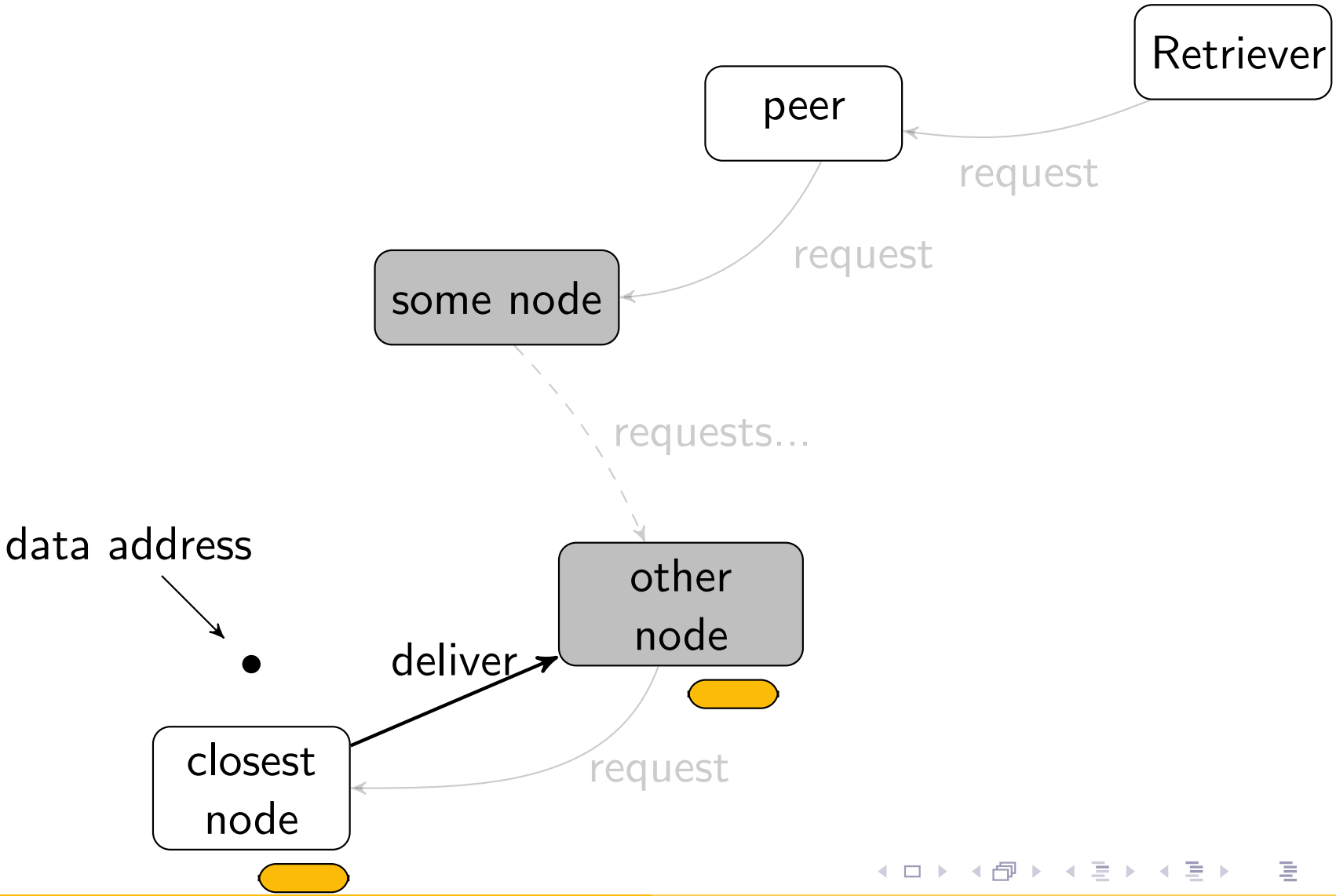
Swarm Retrieval Process



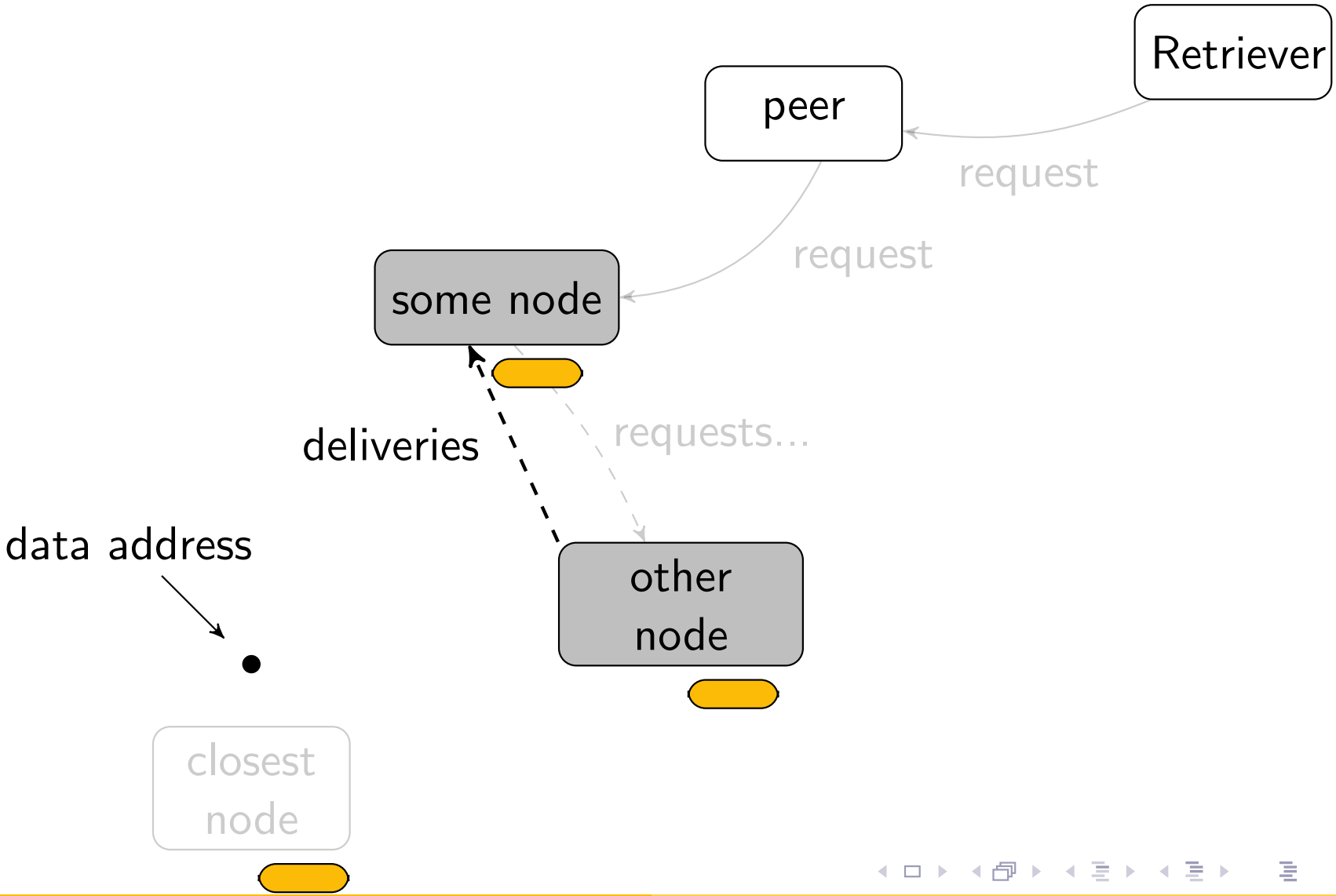
Swarm Retrieval Process



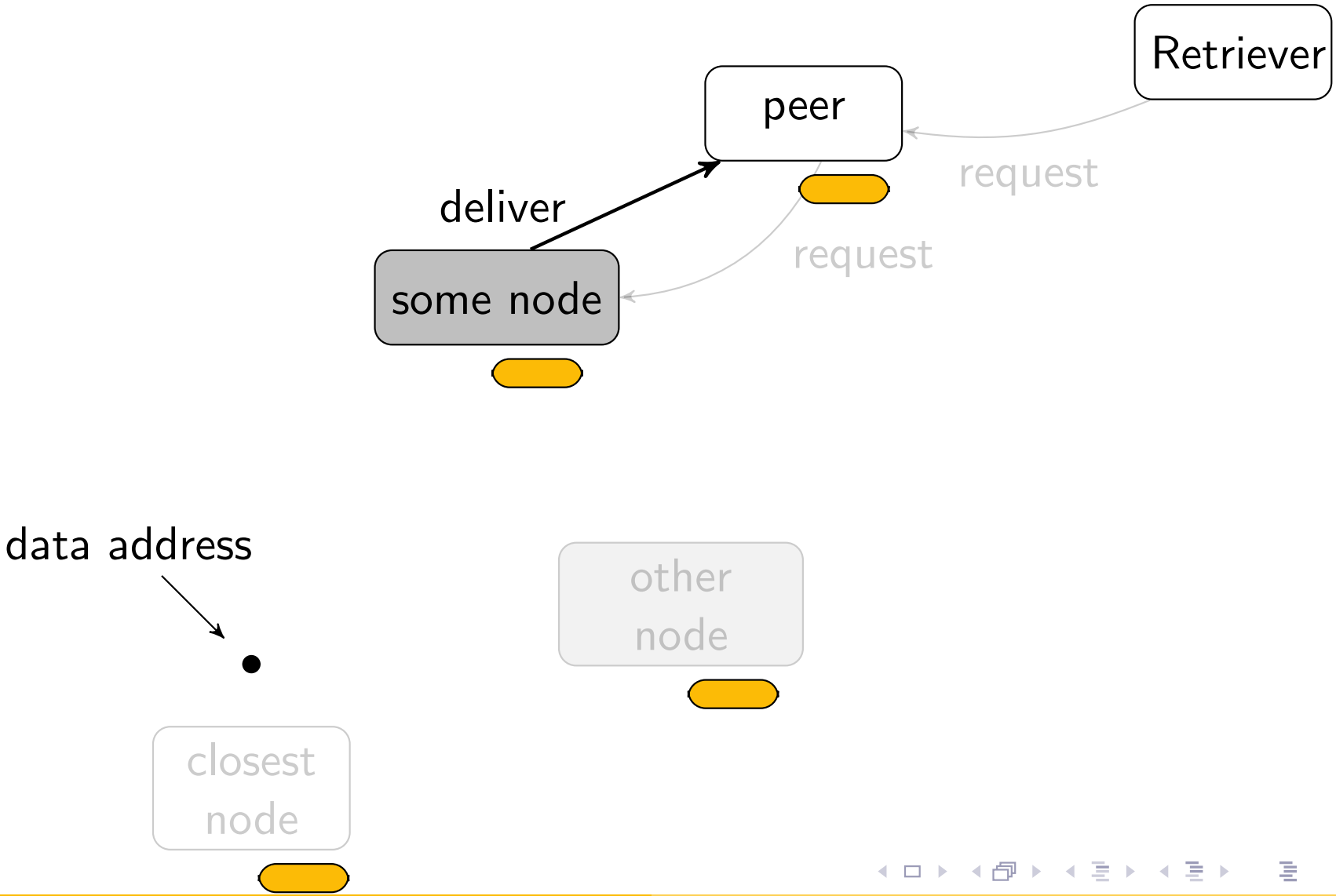
Swarm Retrieval Process



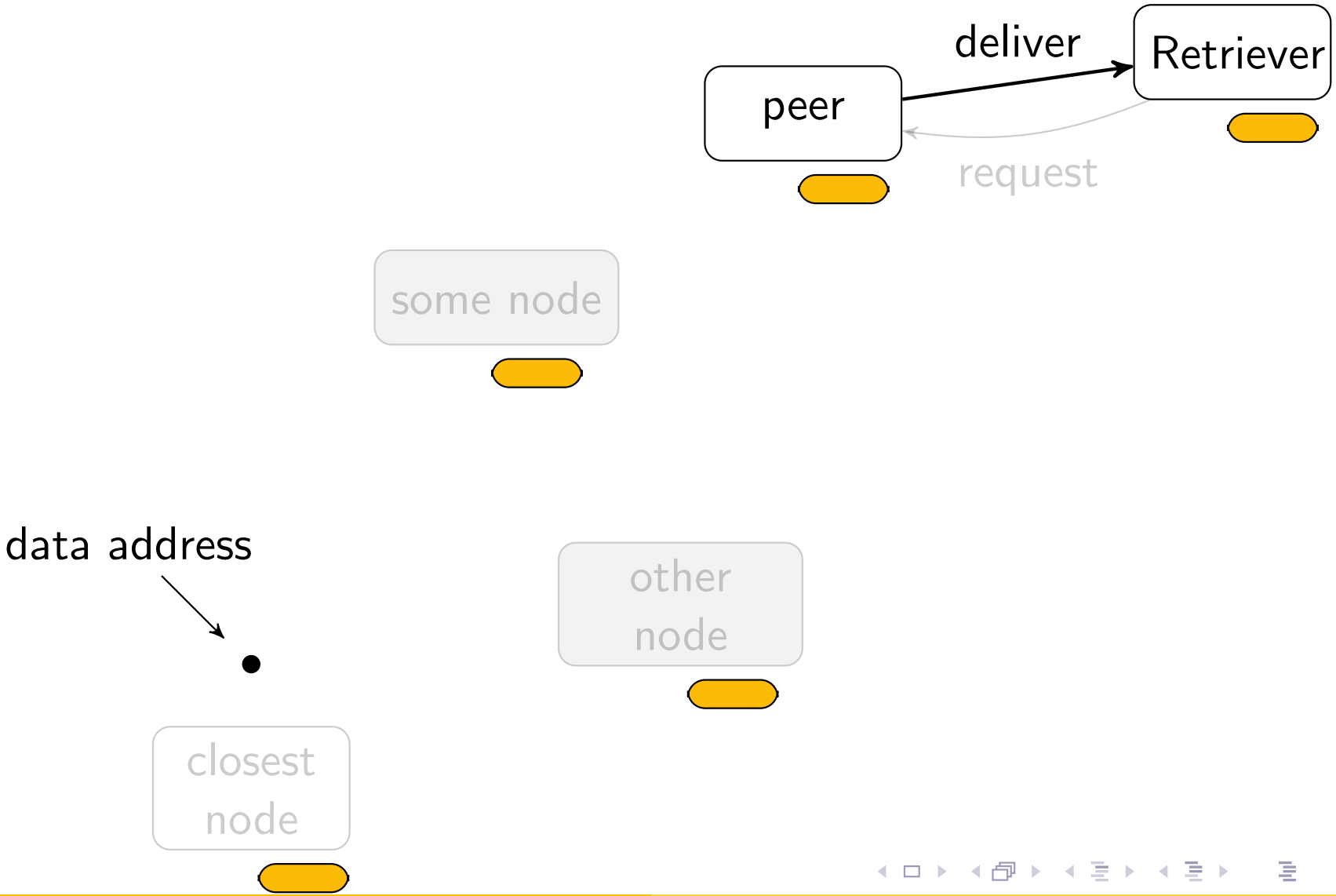
Swarm Retrieval Process



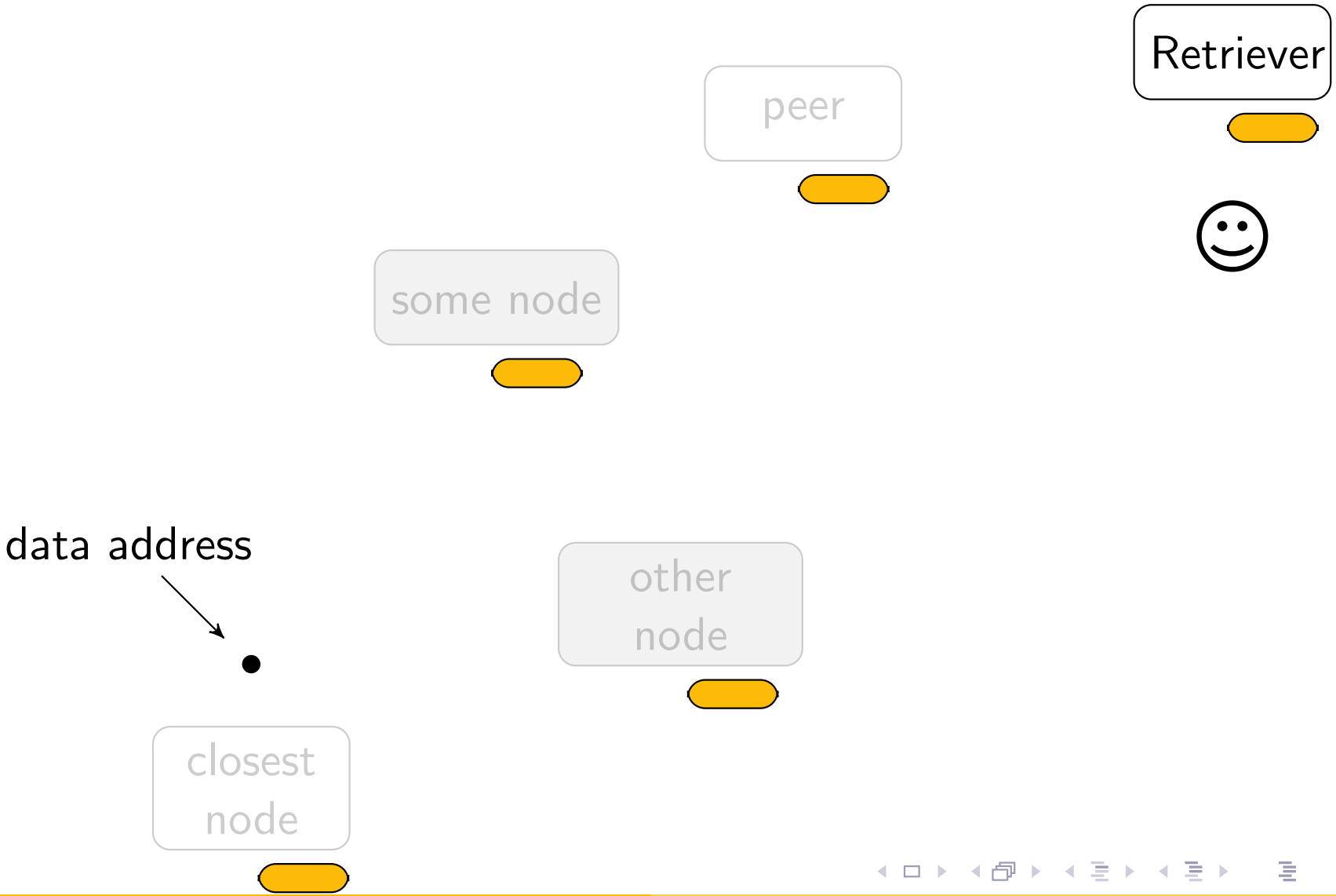
Swarm Retrieval Process



Swarm Retrieval Process



Swarm Retrieval Process



SWAP: **Sw**arm **A**ccounting **P**rotocol

SWAP: Swarm Accounting Protocol

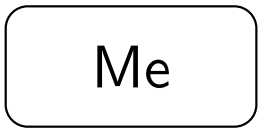
The **Swarm Accounting Protocol** keeps track of all data retrieved via the swarm retrieval process. It is used to facilitate automated payments between peers for the bandwidth they provide.

SWAP: Swarm Accounting Protocol

The **Swarm Accounting Protocol** keeps track of all data retrieved via the swarm retrieval process. It is used to facilitate automated payments between peers for the bandwidth they provide.
Note: Bandwidth accounting has to be per-peer.

SWAP: Swarm Accounting Protocol

The **Swarm Accounting Protocol** keeps track of all data retrieved via the swarm retrieval process. It is used to facilitate automated payments between peers for the bandwidth they provide.
Note: Bandwidth accounting is per-peer.



SWAP: Swarm Accounting Protocol

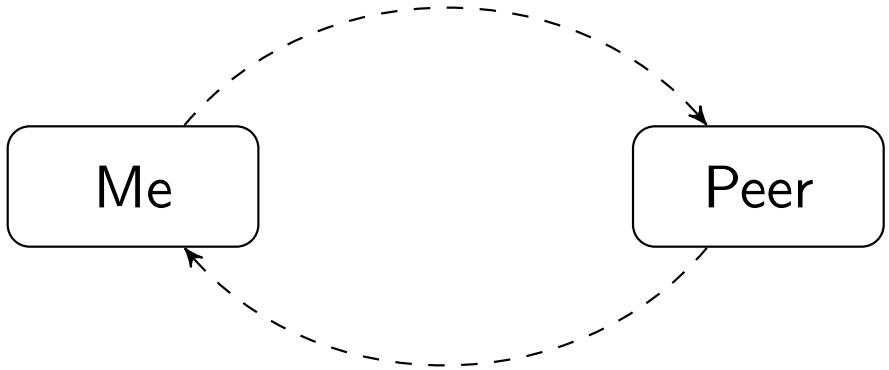
The **Swarm Accounting Protocol** keeps track of all data retrieved via the swarm retrieval process. It is used to facilitate automated payments between peers for the bandwidth they provide.
Note: Bandwidth accounting is per-peer.

SWAP: Swarm Accounting Protocol

The **Swarm Accounting Protocol** keeps track of all data retrieved via the swarm retrieval process. It is used to facilitate automated payments between peers for the bandwidth they provide.
Note: Bandwidth accounting is per-peer.

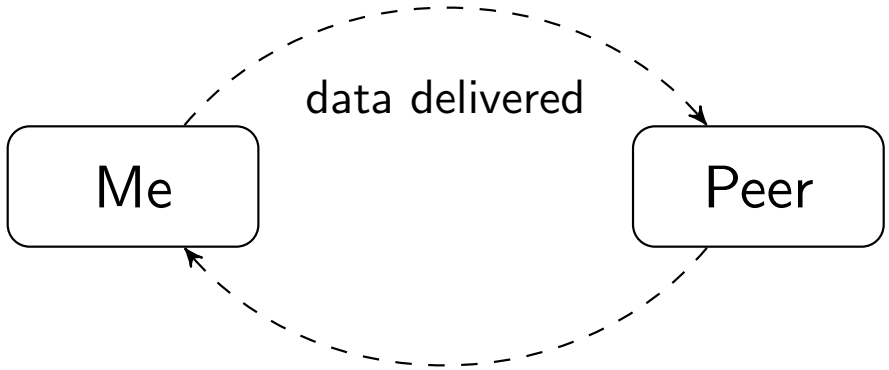
SWAP: Swarm Accounting Protocol

The **Swarm Accounting Protocol** keeps track of all data retrieved via the swarm retrieval process. It is used to facilitate automated payments between peers for the bandwidth they provide.
Note: Bandwidth accounting is per-peer.



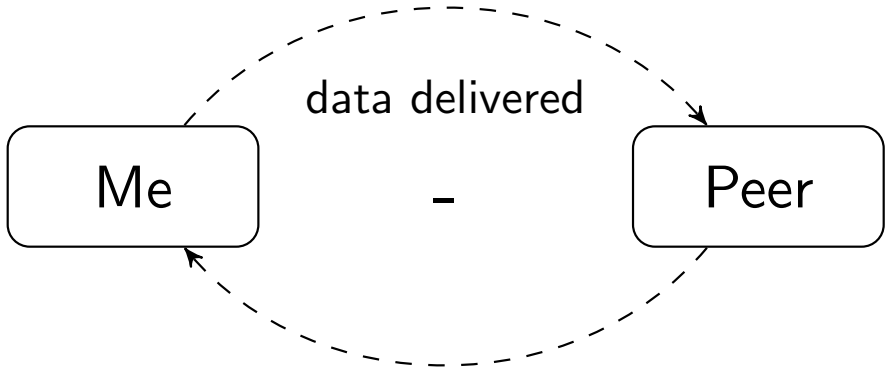
SWAP: Swarm Accounting Protocol

The **Swarm Accounting Protocol** keeps track of all data retrieved via the swarm retrieval process. It is used to facilitate automated payments between peers for the bandwidth they provide.
Note: Bandwidth accounting is per-peer.



SWAP: Swarm Accounting Protocol

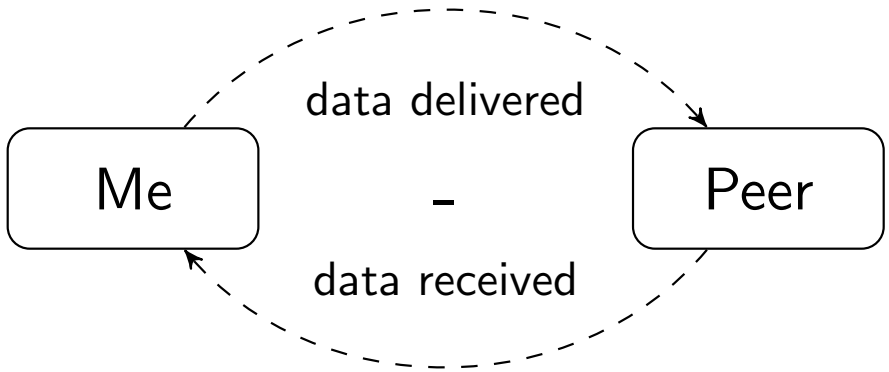
The **Swarm Accounting Protocol** keeps track of all data retrieved via the swarm retrieval process. It is used to facilitate automated payments between peers for the bandwidth they provide.
Note: Bandwidth accounting is per-peer.



SWAP: Swarm Accounting Protocol

The **Swarm Accounting Protocol** keeps track of all data retrieved via the swarm retrieval process. It is used to facilitate automated payments between peers for the bandwidth they provide.
Note: Bandwidth accounting is per-peer.

When the peer connection becomes too imbalanced, a *payment* is initiated.



The chequebook and the channel

- It is *impossible* to pay for every chunk of data delivered.

The chequebook and the channel

- It is *impossible* to pay for every chunk of data delivered.
- Even batch payments would constitute unacceptable blockchain bloat (and transaction cost).

The chequebook and the channel

- It is *impossible* to pay for every chunk of data delivered.
- Even batch payments would constitute unacceptable blockchain bloat (and transaction cost).

Instead of processing every payment on-chain, SWAP employs a *chequebook* smart contract:

The chequebook and the channel

- It is *impossible* to pay for every chunk of data delivered.
- Even batch payments would constitute unacceptable blockchain bloat (and transaction cost).

Instead of processing every payment on-chain, SWAP employs a *chequebook* smart contract:

- Cheques are passed between connected swarm nodes (peers) off-chain.
- Peers can cash in (process on-chain) the received cheques at any time.
- Issued cheques are *cumulative* and **only the last cheque has to be cashed**.

The chequebook and the channel

- It is *impossible* to pay for every chunk of data delivered.
- Even batch payments would constitute unacceptable blockchain bloat (and transaction cost).

Instead of processing every payment on-chain, SWAP employs a *chequebook* smart contract:

- Cheques are passed between connected swarm nodes (peers) off-chain.
- Peers can cash in (process on-chain) the received cheques at any time.
- Issued cheques are *cumulative* and **only the last cheque has to be cashed**.

SWAP will soon also be usable via *payment channels* (Raiden).

The chequebook and the channel

Chequebook

Pro:

1. Offchain payments
2. Low barrier to entry (pay anyone)

Con:

1. Cheques can bounce (payment not guaranteed)

Channel

Pro:

1. Offchain payments
2. Secure - payments guaranteed

Con:

1. High barrier to entry (must first join channel network)

The chequebook and the channel

Chequebook

Pro:

1. Offchain payments
2. Low barrier to entry (pay anyone)

Con:

1. Cheques can bounce (payment not guaranteed)

Channel

Pro:

1. Offchain payments
2. Secure - payments guaranteed

Con:

1. High barrier to entry (must first join channel network)

The chequebook and the channel

Chequebook

Pro:

1. Offchain payments
2. Low barrier to entry (pay anyone)

Con:

1. Cheques can bounce (payment not guaranteed)

Channel

Pro:

1. Offchain payments
2. Secure - payments guaranteed

Con:

1. High barrier to entry (must first join channel network)

The chequebook and the channel

Chequebook

Pro:

1. Offchain payments
2. Low barrier to entry (pay anyone)

Con:

1. Cheques can bounce (payment not guaranteed)

Channel

Pro:

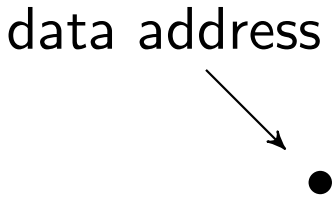
1. Offchain payments
2. Secure - payments guaranteed

Con:

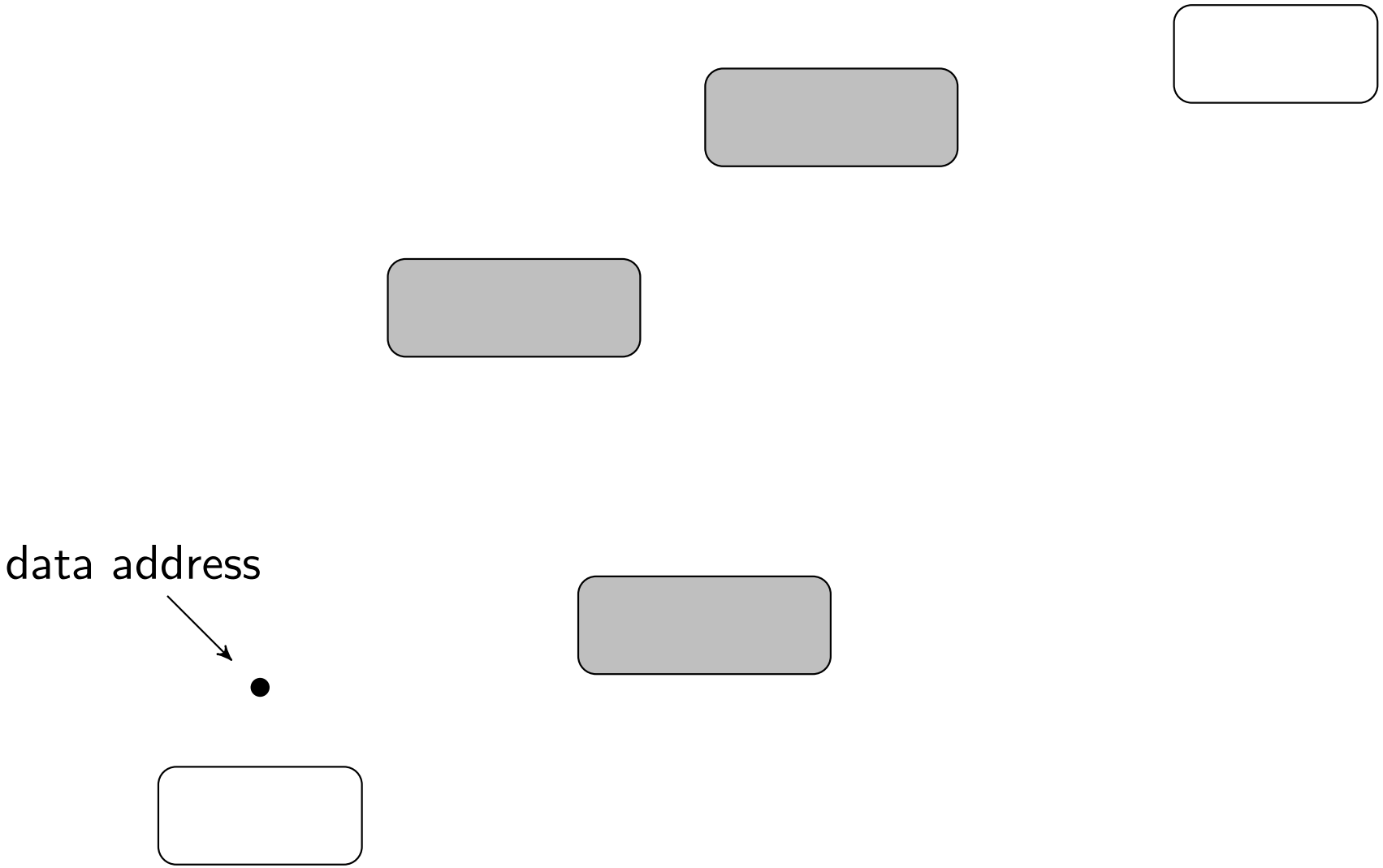
1. High barrier to entry (must first join channel network)

Regardless of *how* payments are processed, SWAP demonstrates how a swarm can have **programmable incentives**. In particular, a network of profit-maximising nodes using SWAP results in an automatically scaling content delivery network.

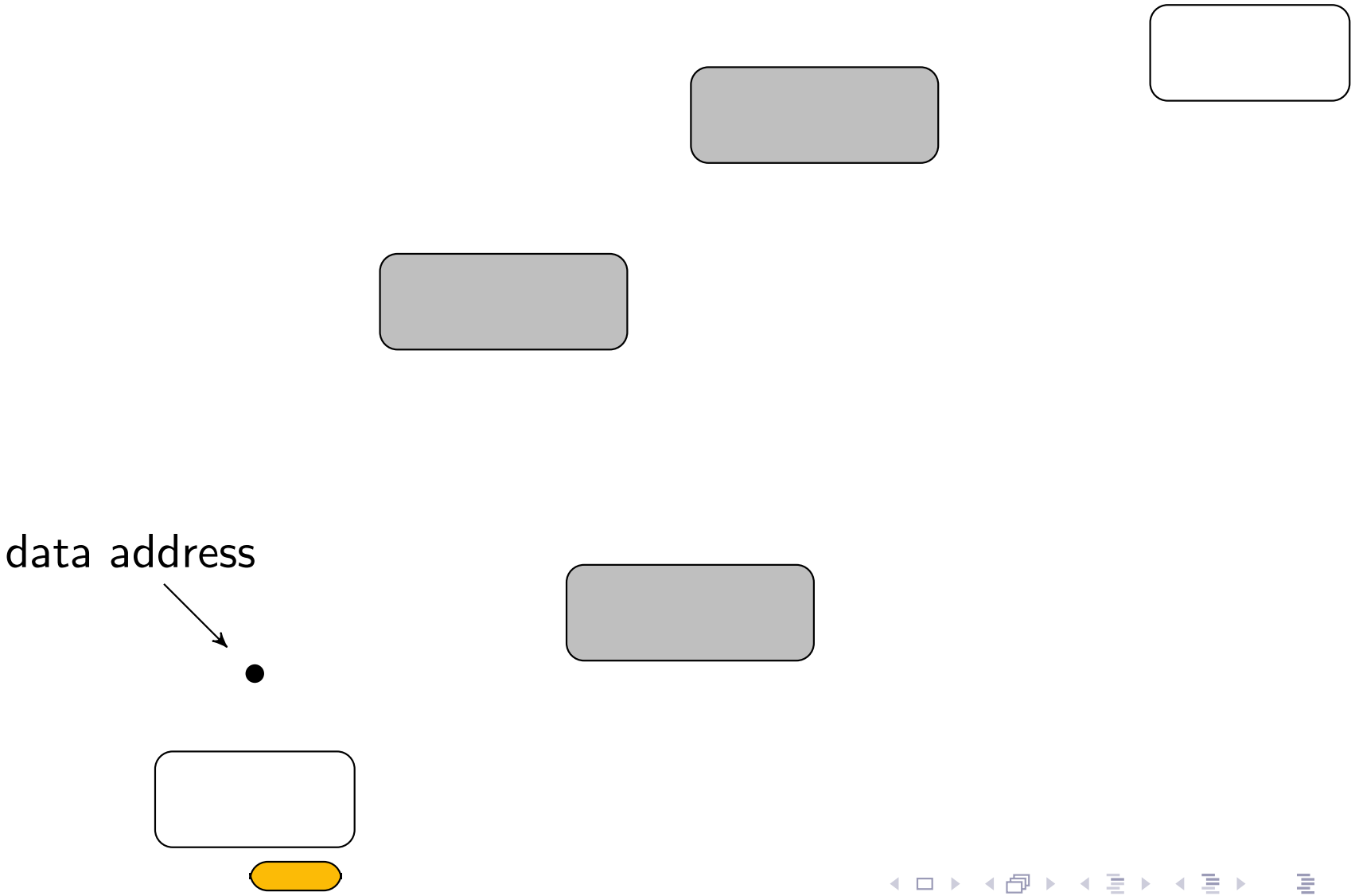
Swarm CDN is auto-scaling



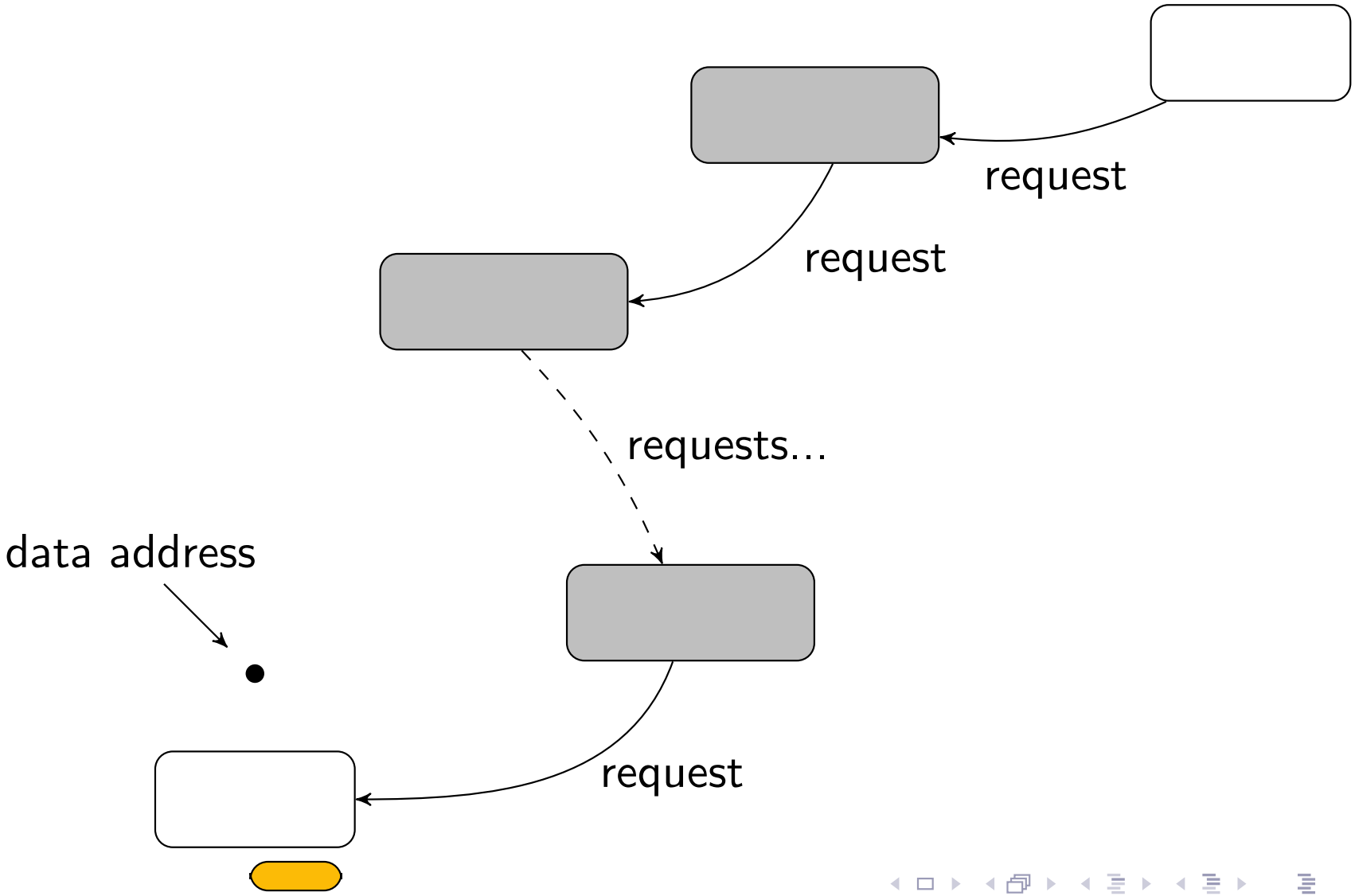
Swarm CDN is auto-scaling



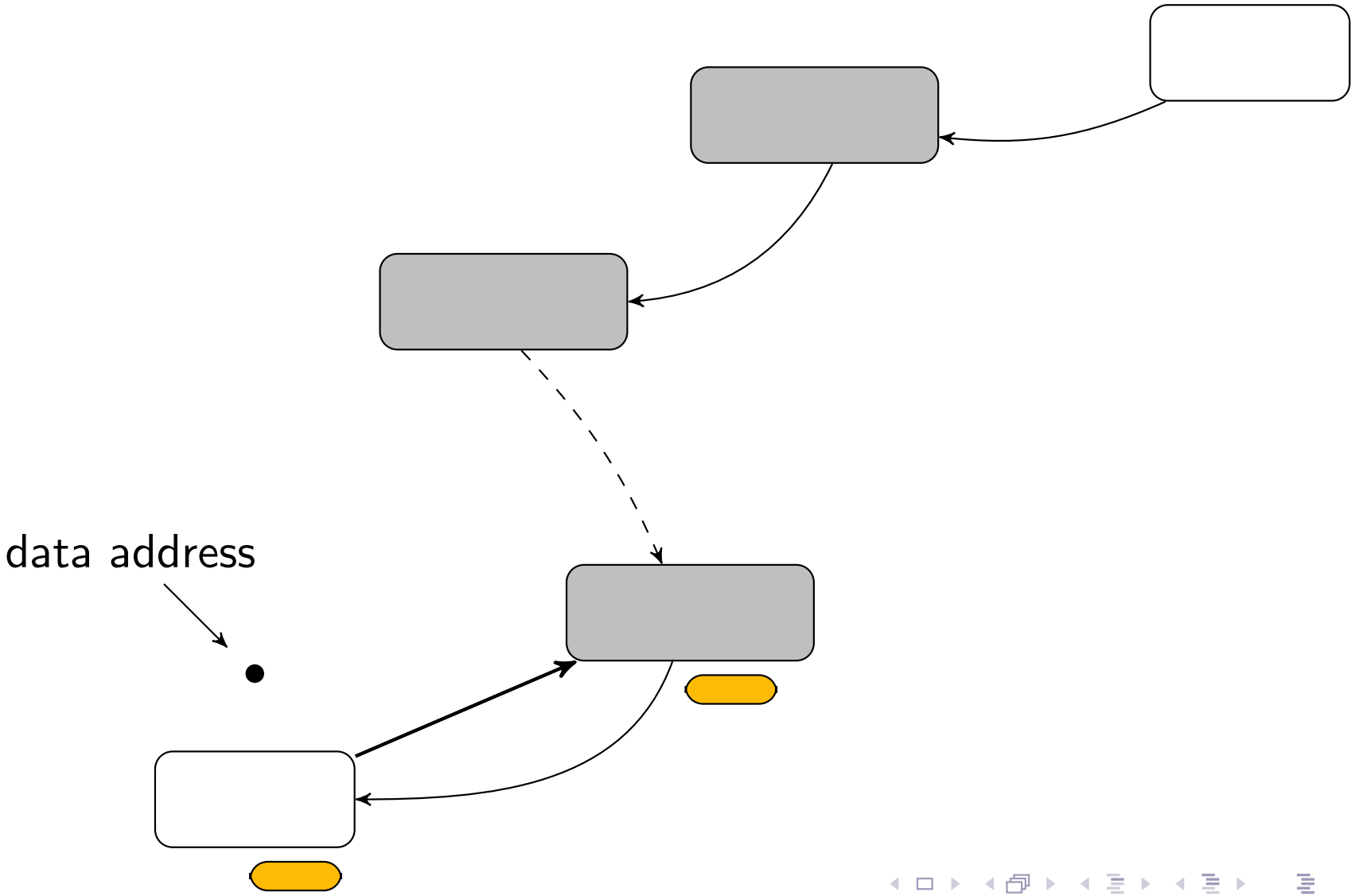
Swarm CDN is auto-scaling



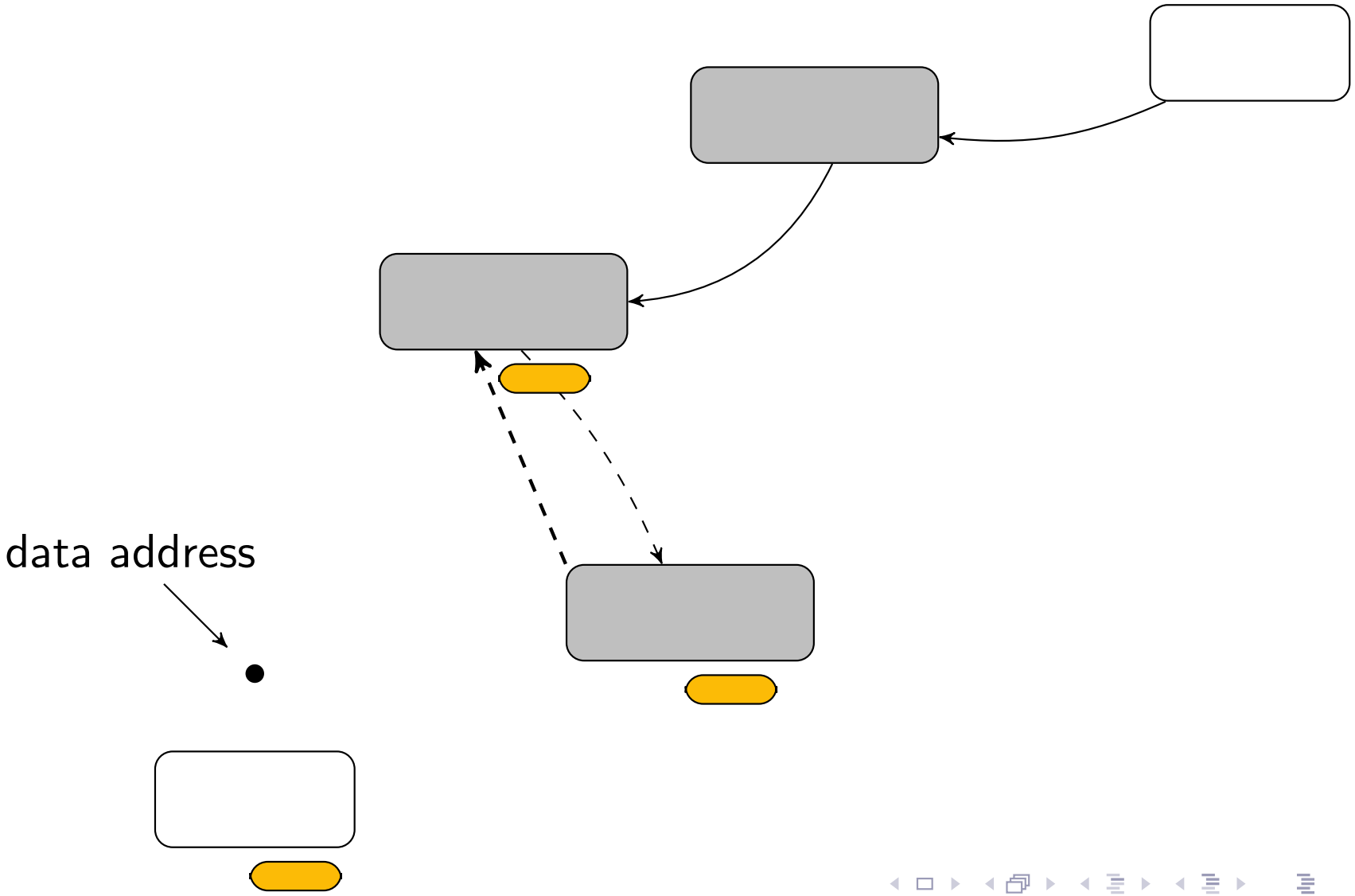
Swarm CDN is auto-scaling



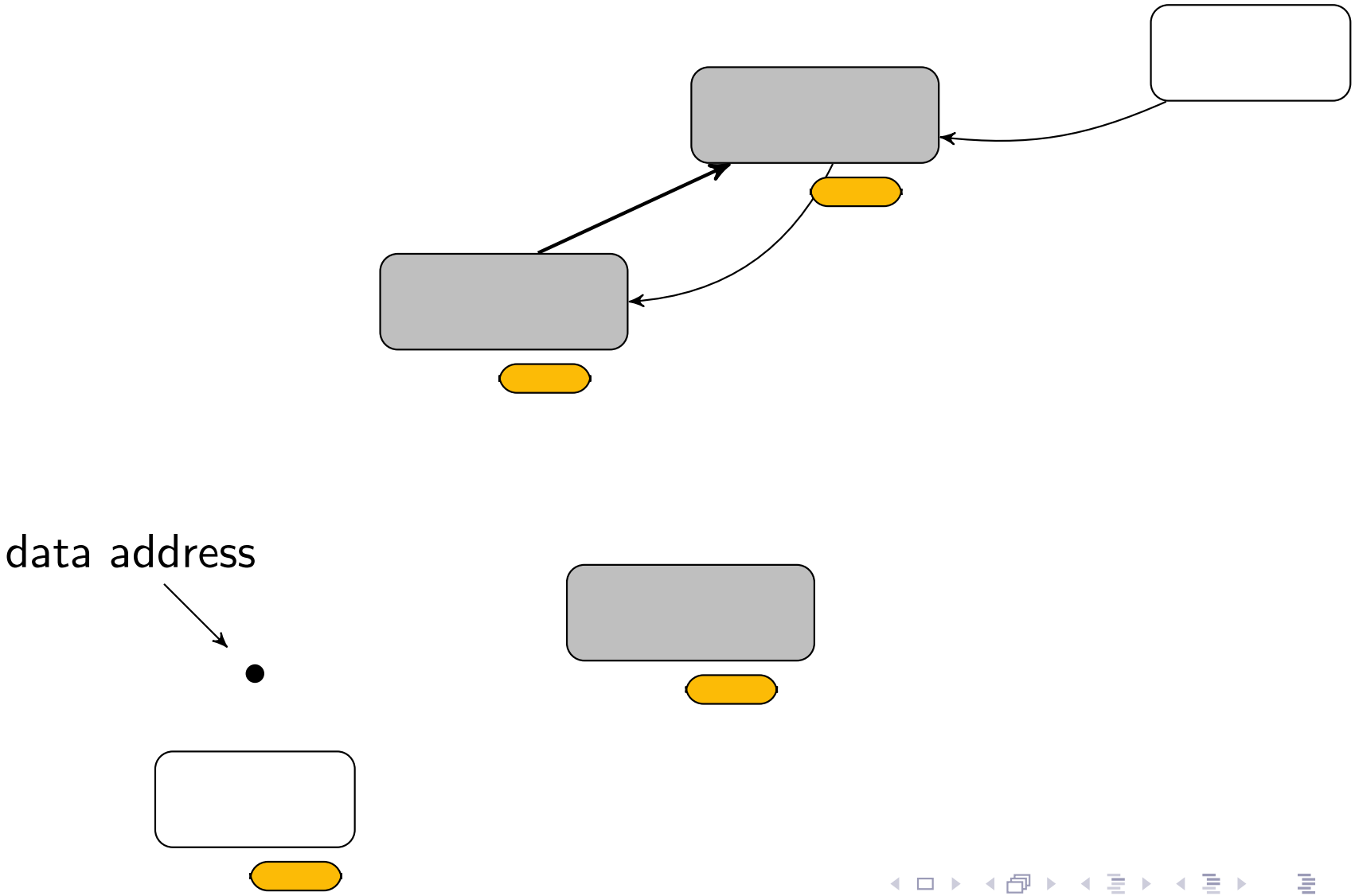
Swarm CDN is auto-scaling



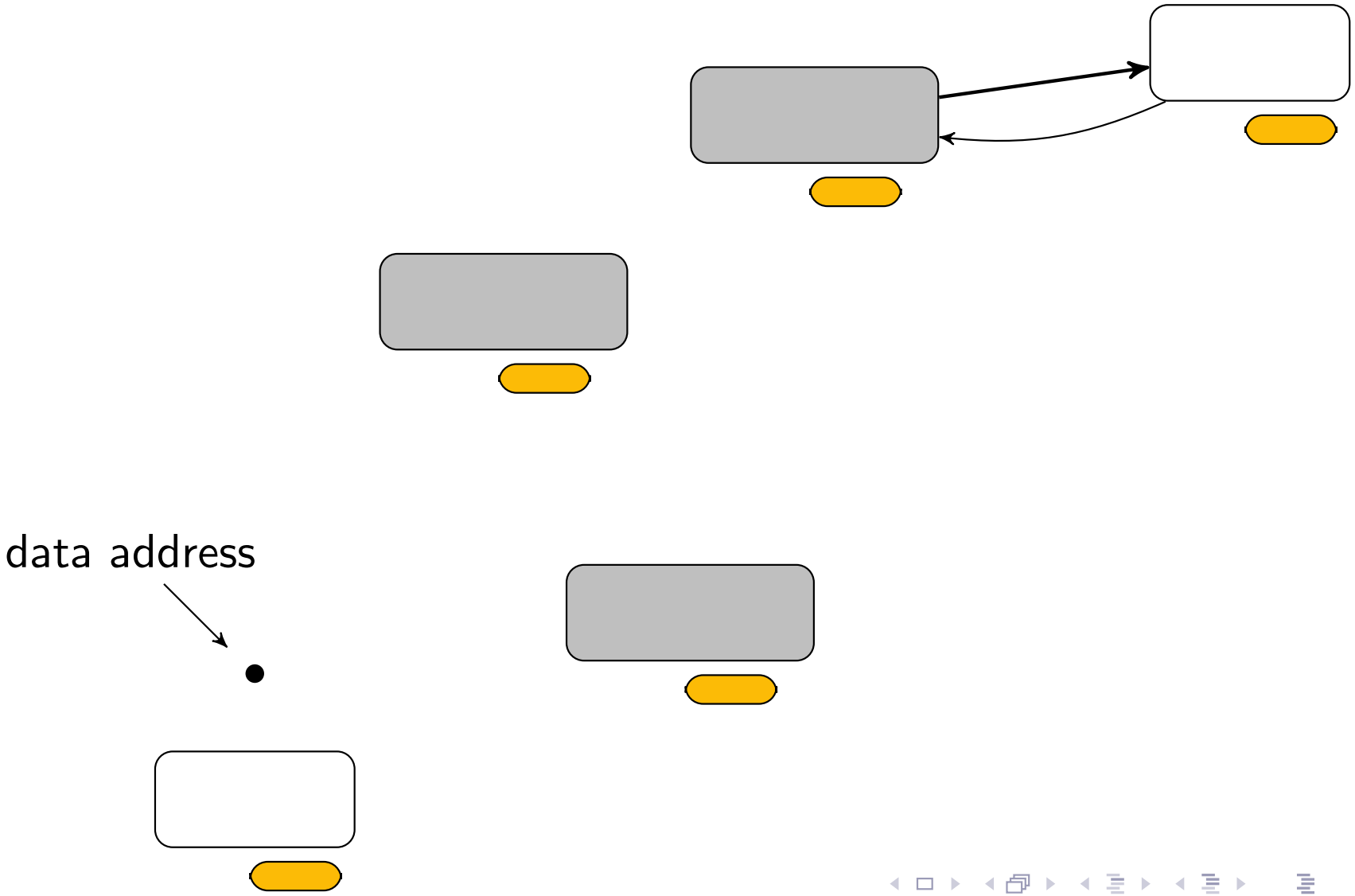
Swarm CDN is auto-scaling



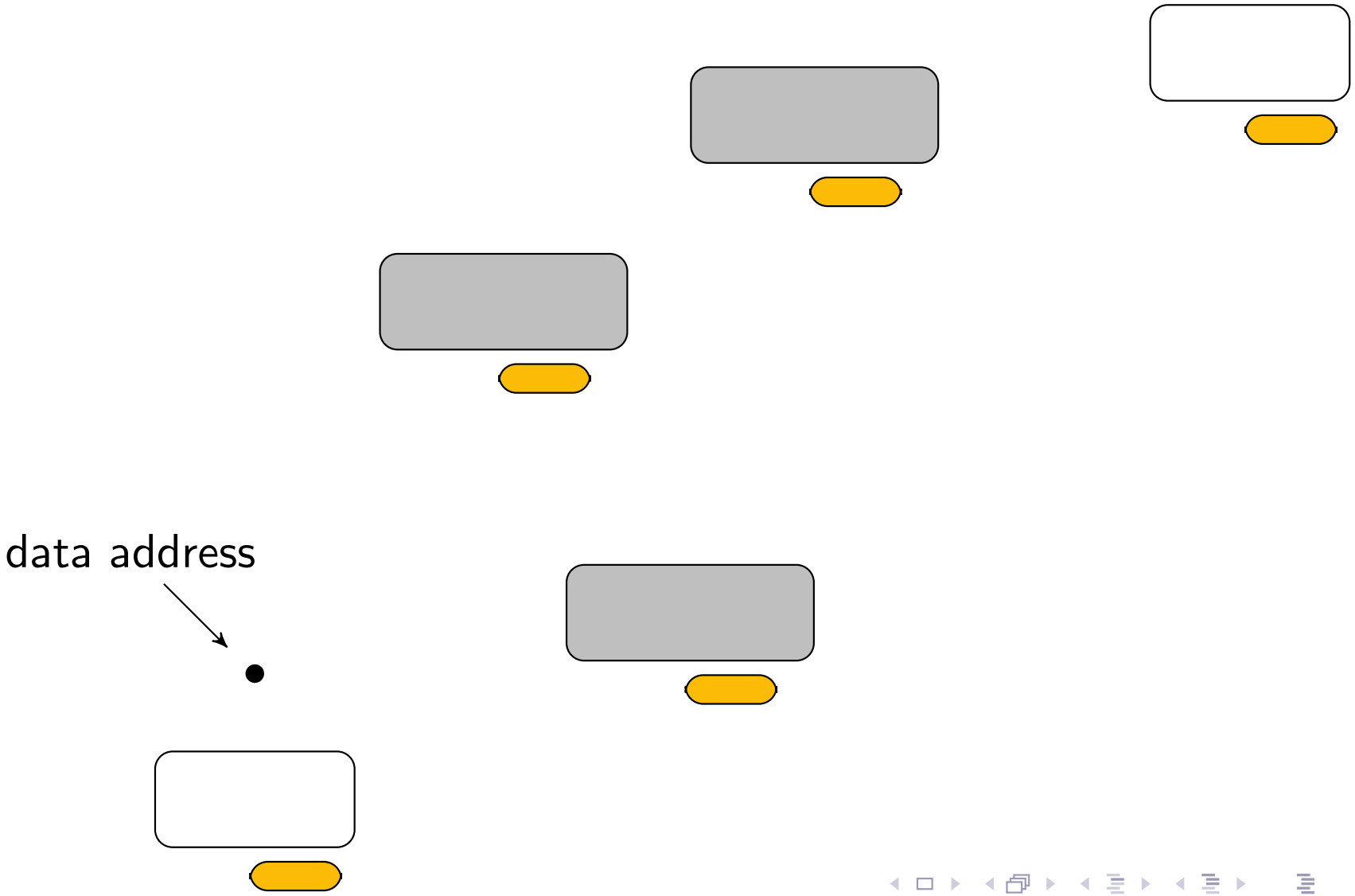
Swarm CDN is auto-scaling



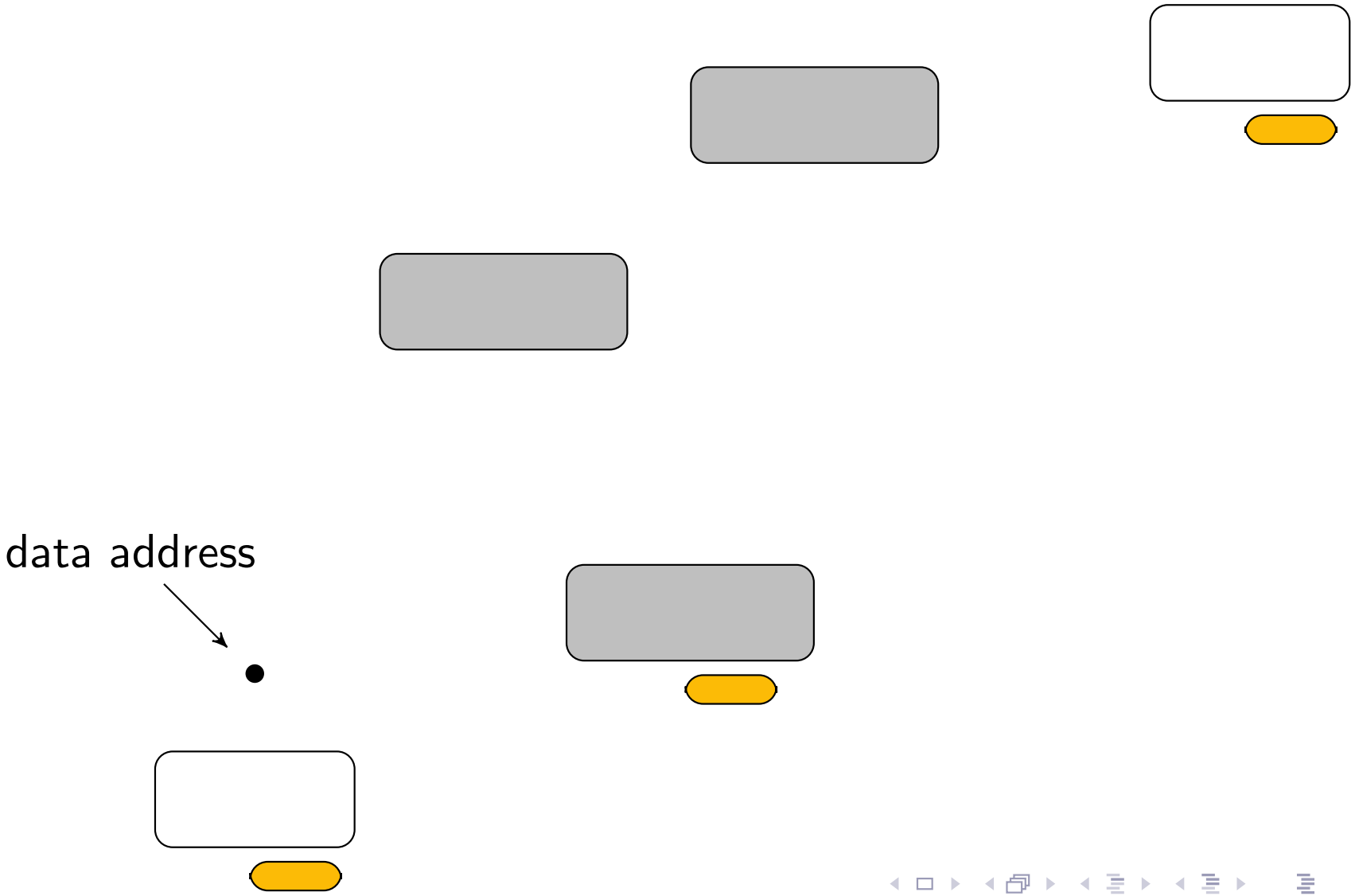
Swarm CDN is auto-scaling



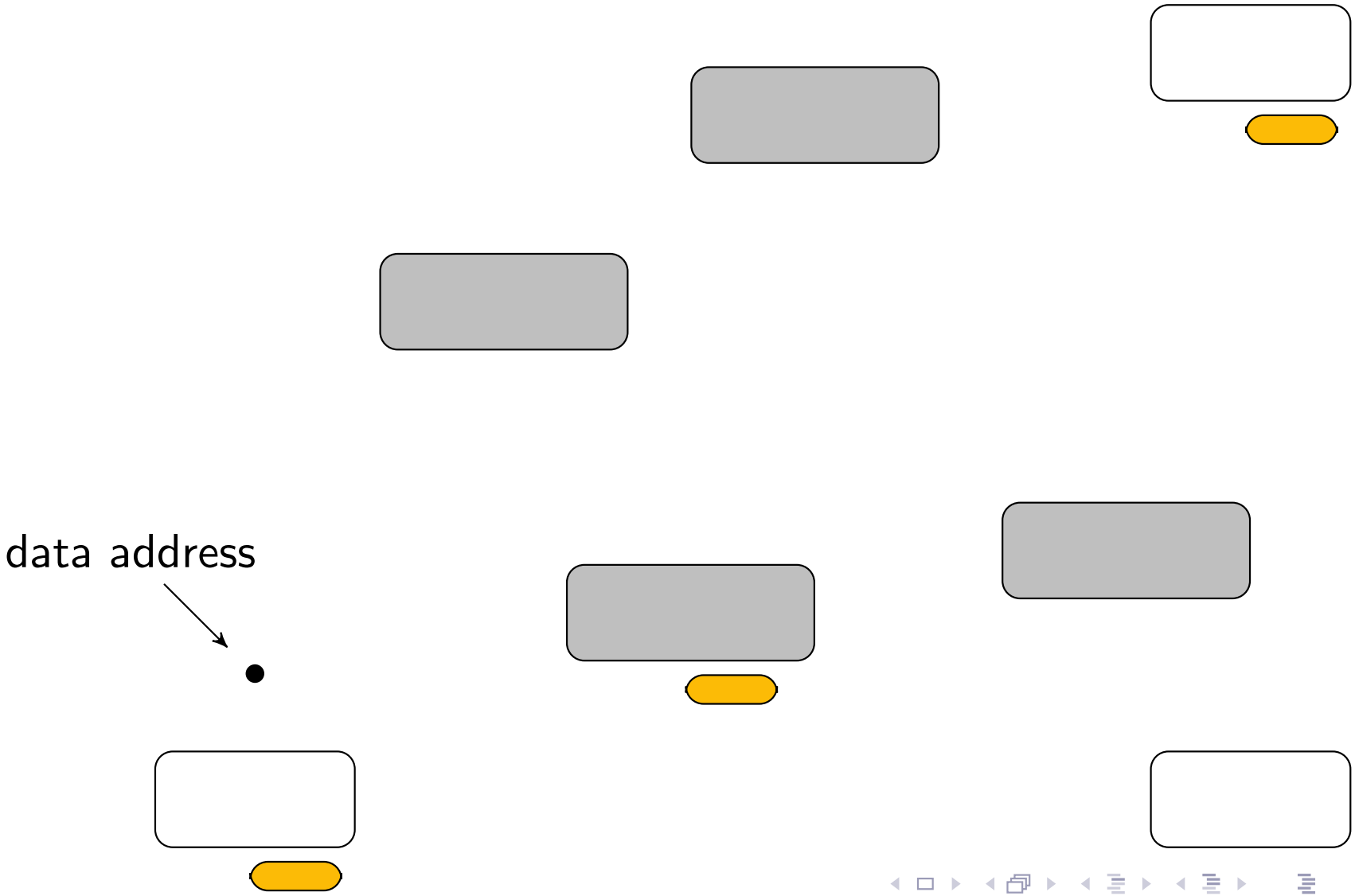
Swarm CDN is auto-scaling



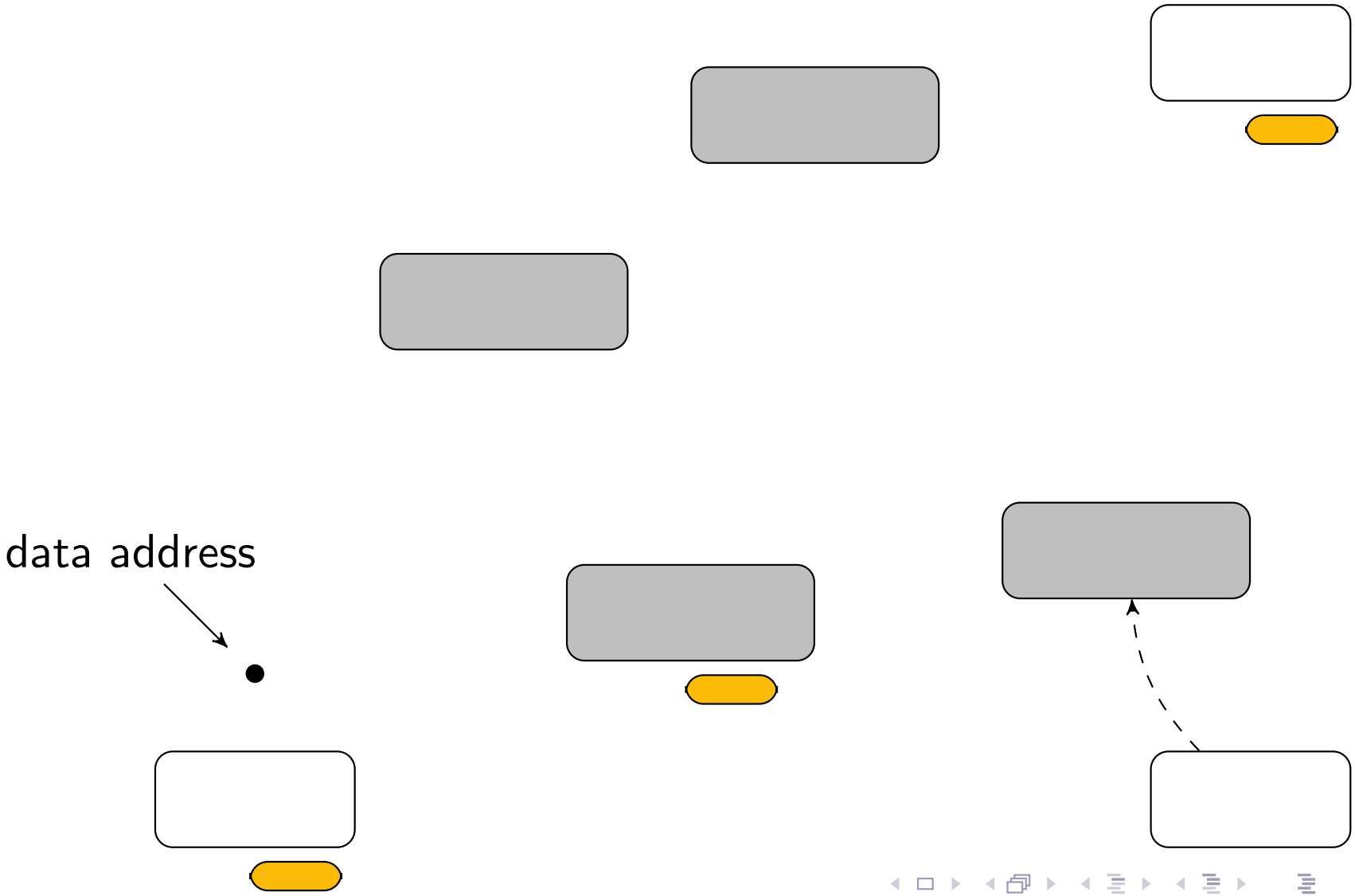
Swarm CDN is auto-scaling



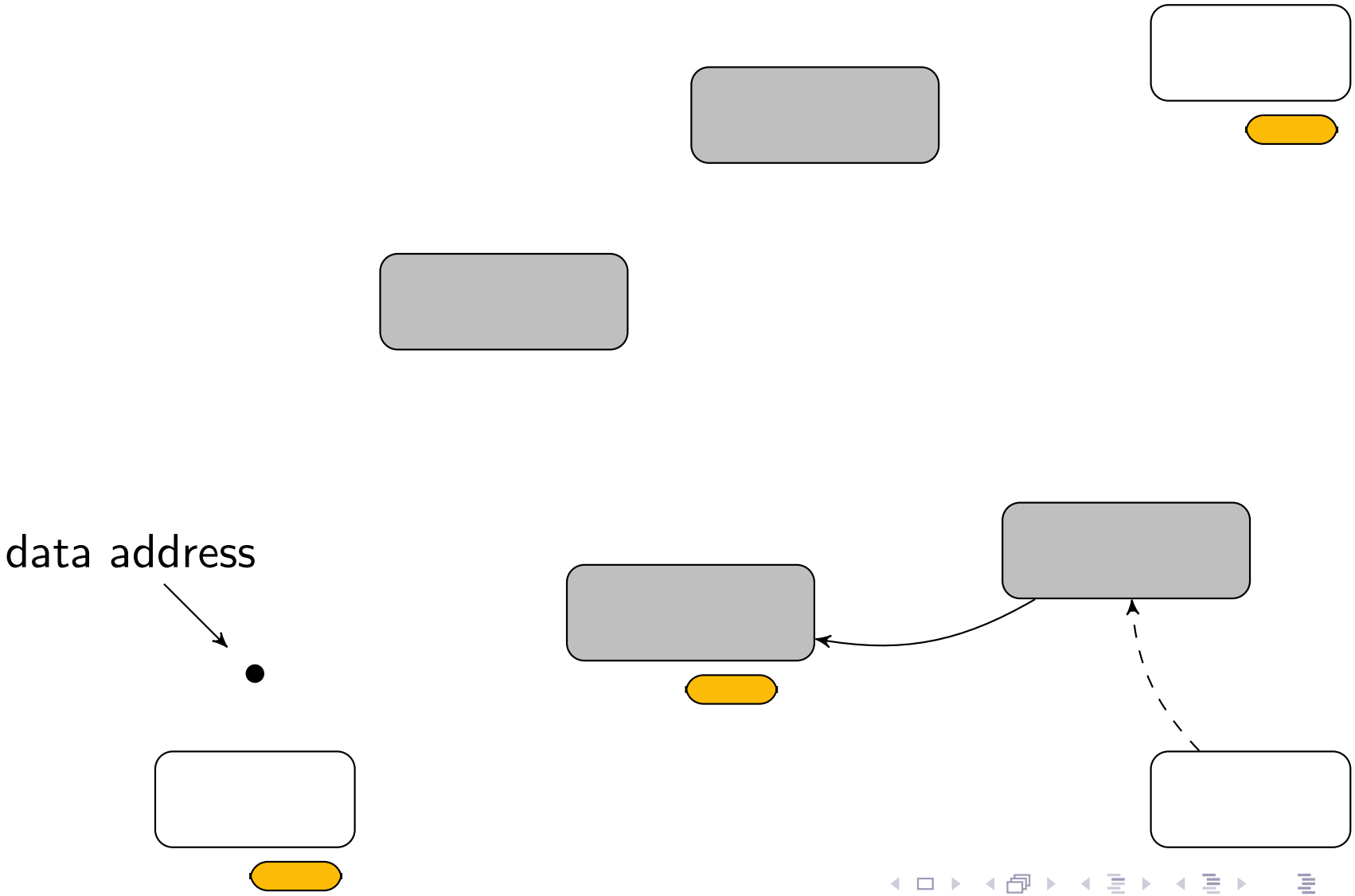
Swarm CDN is auto-scaling



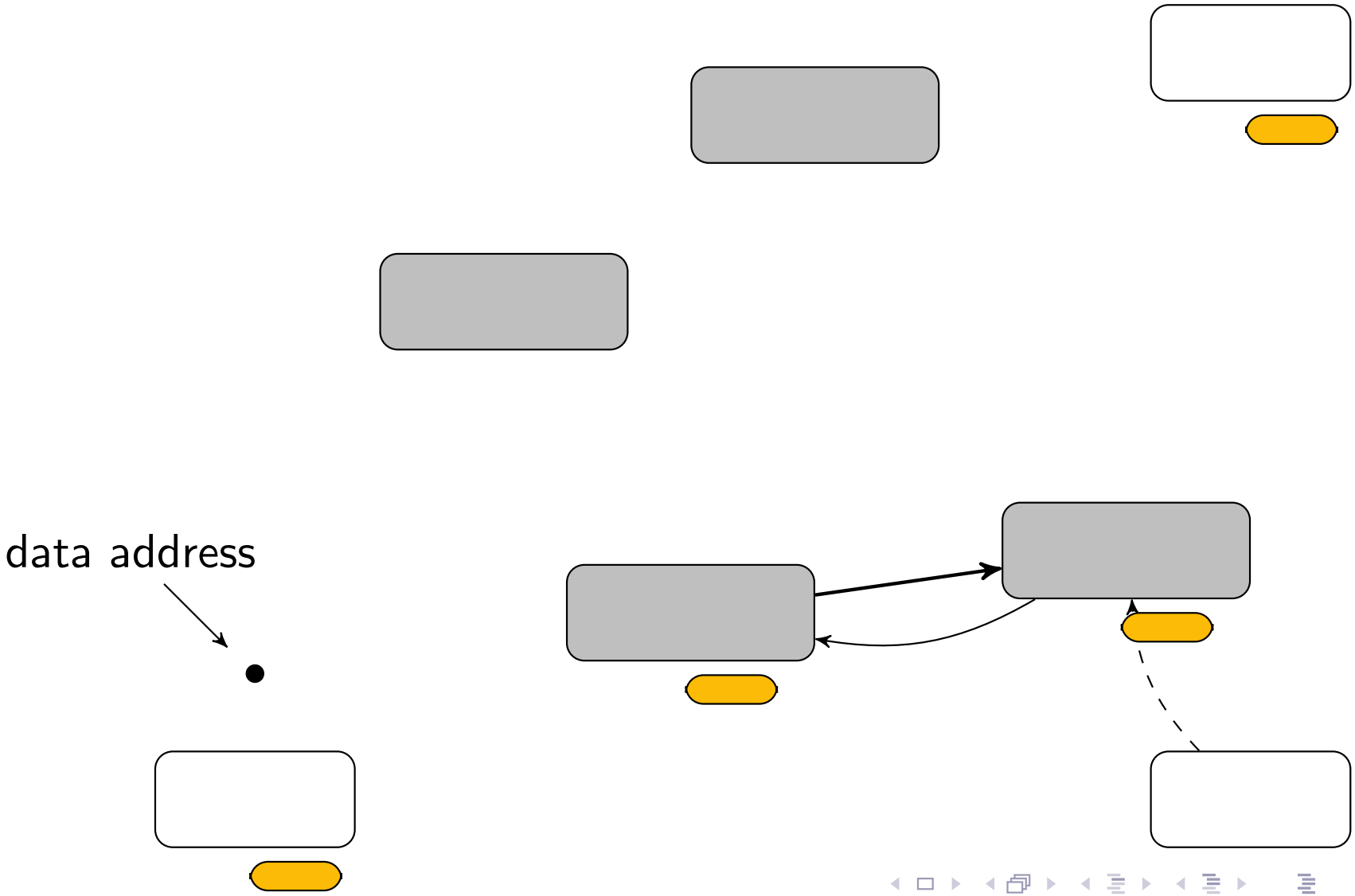
Swarm CDN is auto-scaling



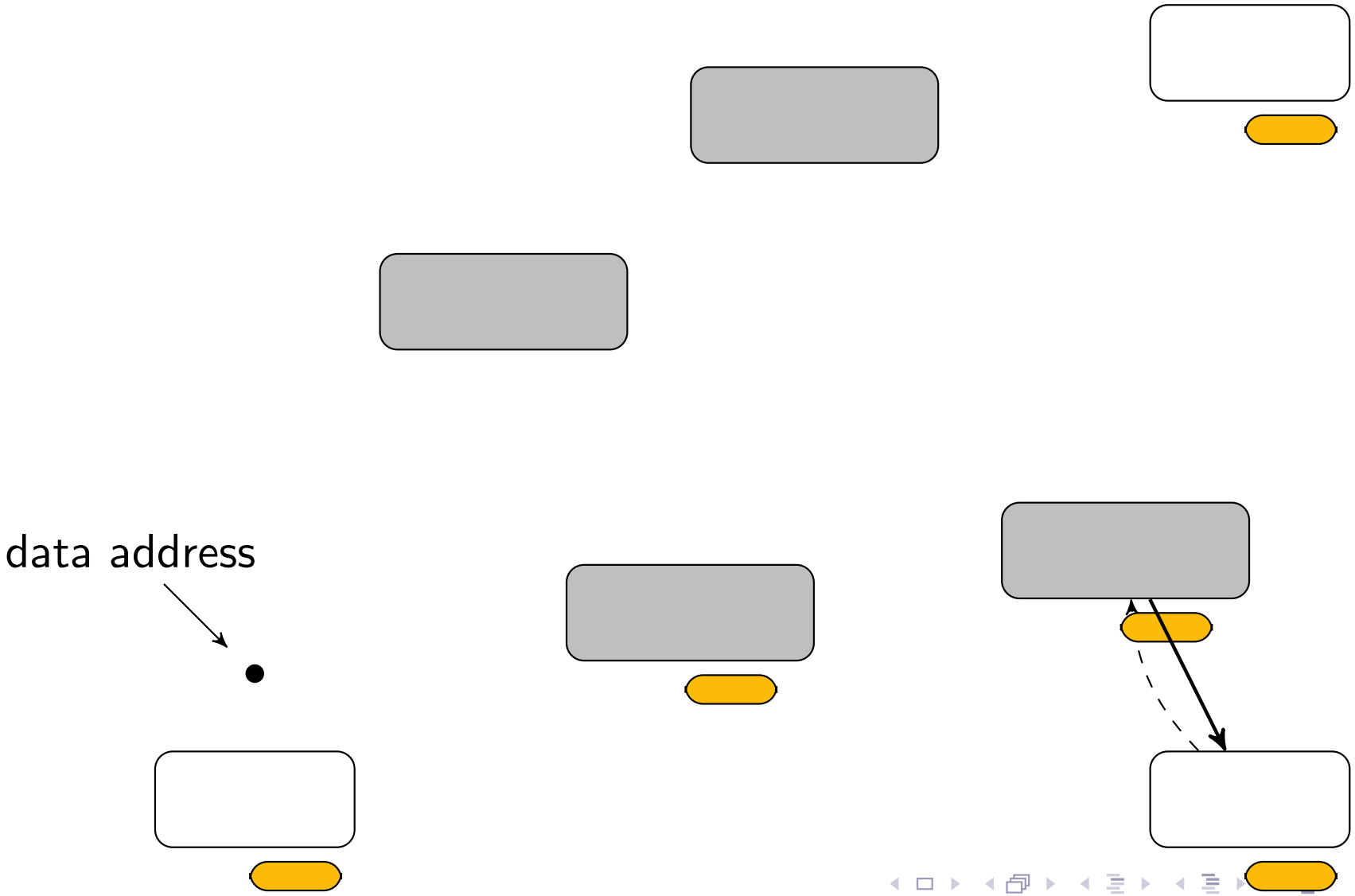
Swarm CDN is auto-scaling



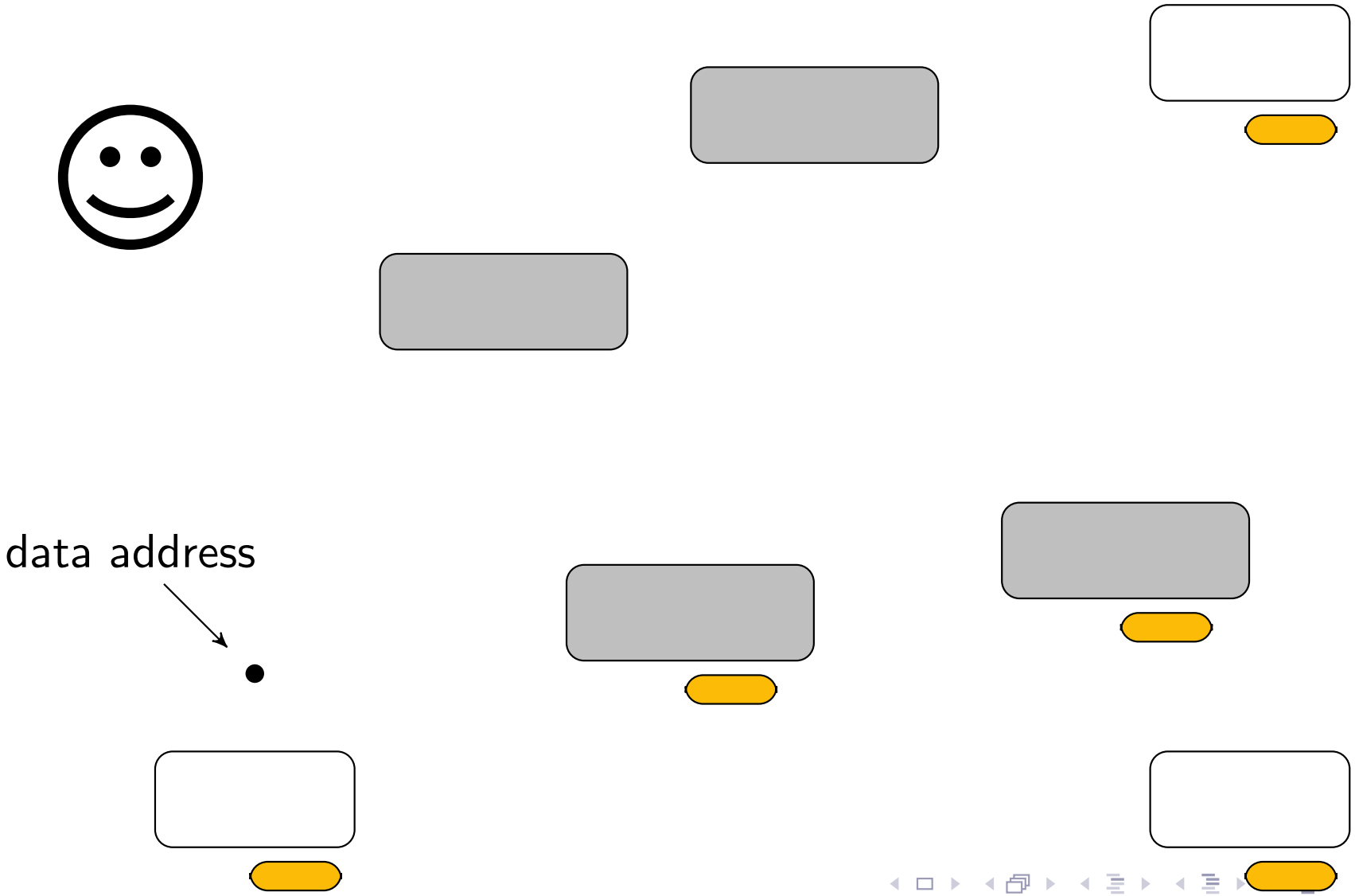
Swarm CDN is auto-scaling



Swarm CDN is auto-scaling



Swarm CDN is auto-scaling



Outline

While SWAP allows for speedy retrieval of popular content, there is no guarantee that less popular content will remain available. Whatever is not accessed for a long time is likely to be garbage collected.
The first strategy to overcome this problem is deceptively simple: change the swarm's incentives by paying nodes to *store* your content.

Payment for proof-of-custody

The basic idea:

1. Commit in advance to paying for data to be available in the swarm.
2. Over time, challenge the swarm to provide proof that the data is still available: request *proof-of-custody*.
3. Every successful proof-of-custody releases the next payment installment to the storing nodes.

Remember:

The **proof-of-custody** here is a small message - a single hash - which cryptographically proves that the issuer has access to the data.

POC + Payment Channel

These deferred payments constitute a **conditional escrow**: payment must be made up-front for the storage, payment is held (escrow) and is only released when a successful proof-of-custody message is received (condition).

This procedure can be handled off-chain and can be directly **integrated into the payment channels.**

What this requires is that the payment channel has the ability to call a *judge contract* that can understand and verify promisory payments and proof-of-custody challenge/response messages.

If data goes missing...

The problem of using only positive incentives is that data loss has only limited consequences for the storing nodes. A node will lose potential revenue for no longer being able to generate proofs-of-custody, but there are no further consequences.

Therefore, to complete the storage incentive scheme, we introduce an *insurance system* the can punish offending nodes for not keeping their storage promises.

SWEAR: **SW**arm **E**nforcement of **A**rchiving **R**ules

SWEAR to store

SWEAR is a smart contract that allows nodes to register as long-term storage nodes by posting a security deposit.

Registered nodes can sell promisory notes guaranteeing long-term data availablilty – essentially insurance against garbage collection.

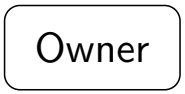
Implementation: Swarm + Receipts.

The syncing process.

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

The syncing process.

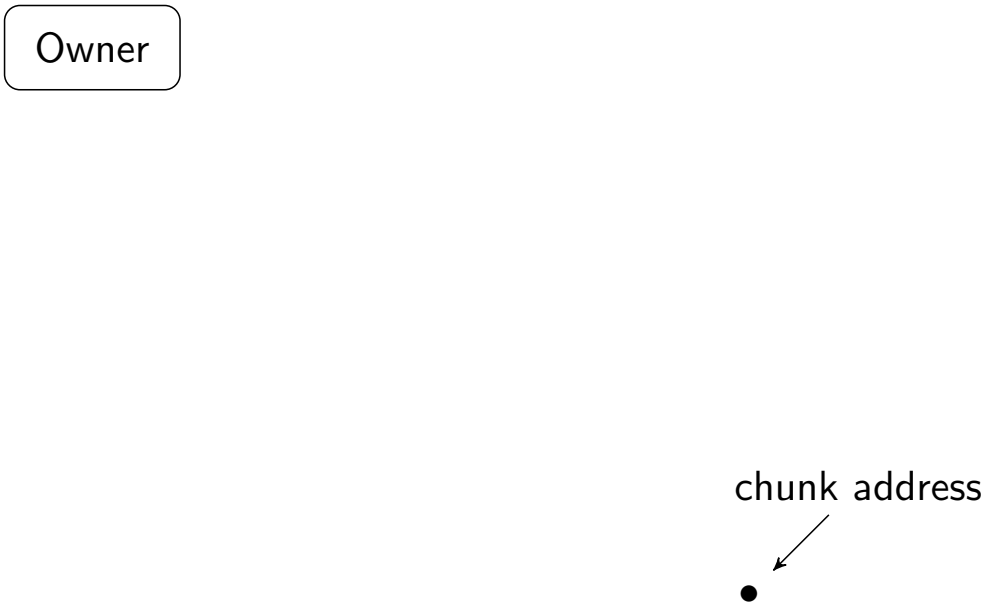
The normal process of getting chunks of data to their storage destination is called syncing.



The syncing process.

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

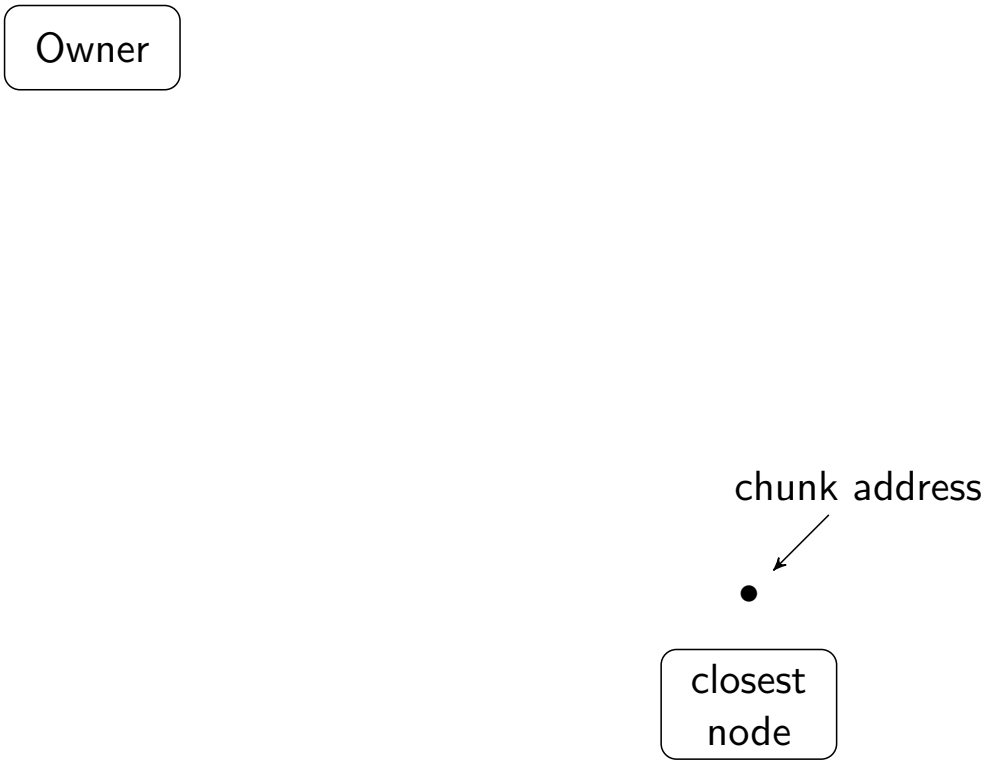
- Chunks are to be stored at the chunk ID.



The syncing process.

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

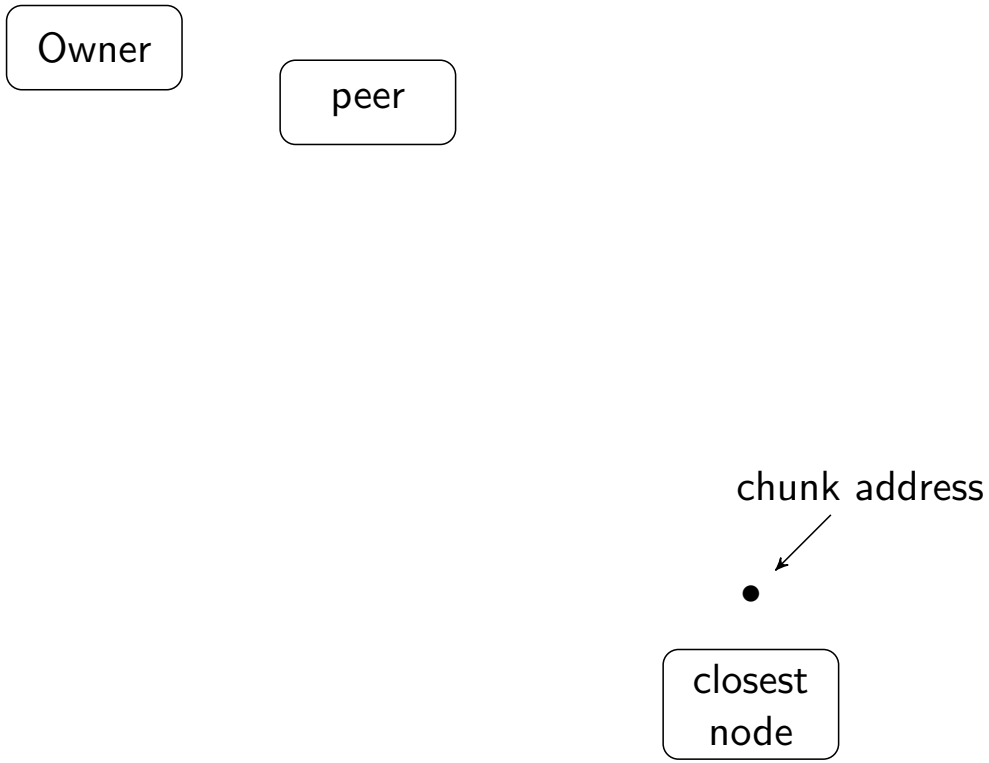
- Chunks are to be stored at the node whose address is closest to the chunk ID.



The syncing process.

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

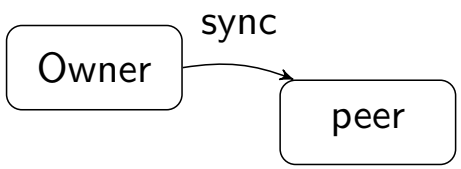
- Chunks are to be stored at the node whose address is closest to the chunk ID.
- Instead of directly connecting to that node and giving it the chunk in question, we pass the chunk along to one of our connected peers who is a little closer to the chunk ID than we are.

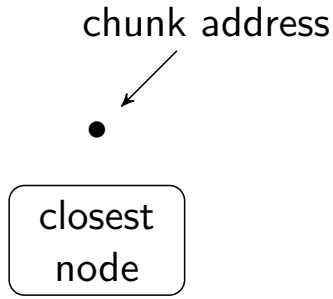


The syncing process.

The normal process of getting chunks of data 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".

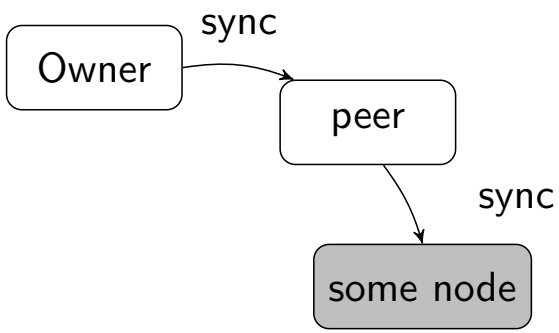


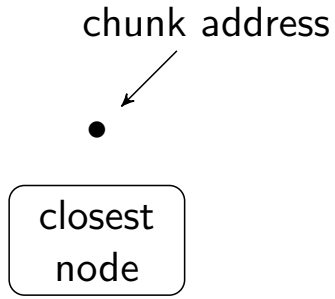


The syncing process.

The normal process of getting chunks of data 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

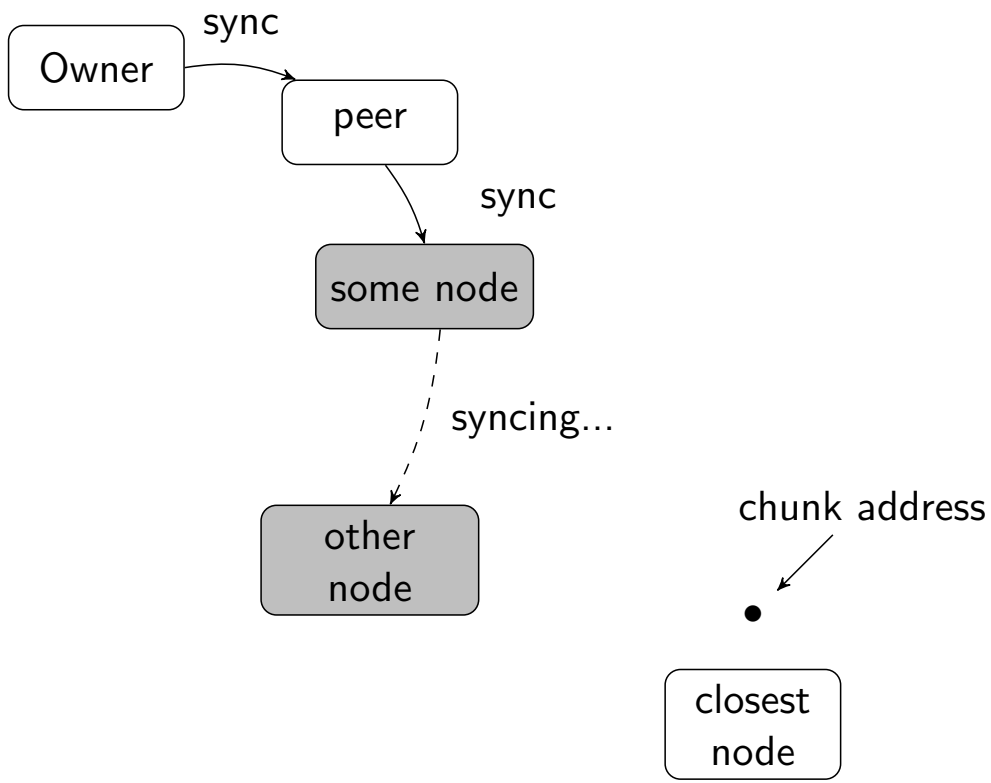




The syncing process.

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

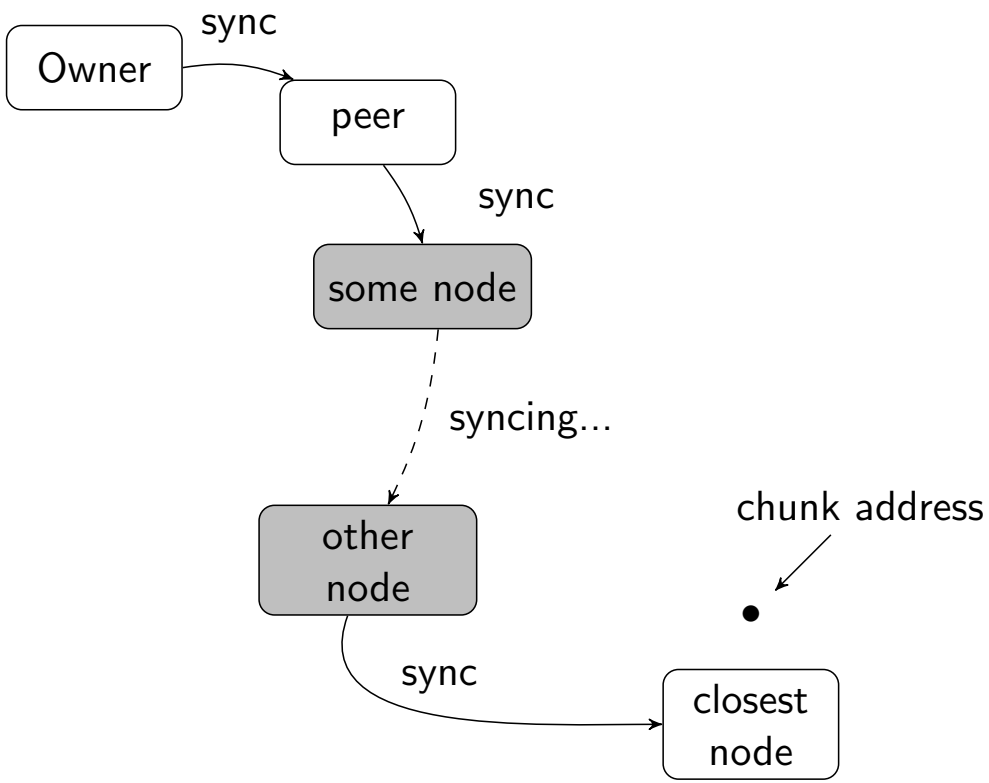
- Chunks are to be stored at the node whose address is closest to the chunk ID.
- Instead of directly connecting to that node and giving it the chunk in question, we pass the chunk along to one of our connected peers who is a little closer to the chunk ID than we are. "Syncing".
- This peer will repeat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



The syncing process.

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

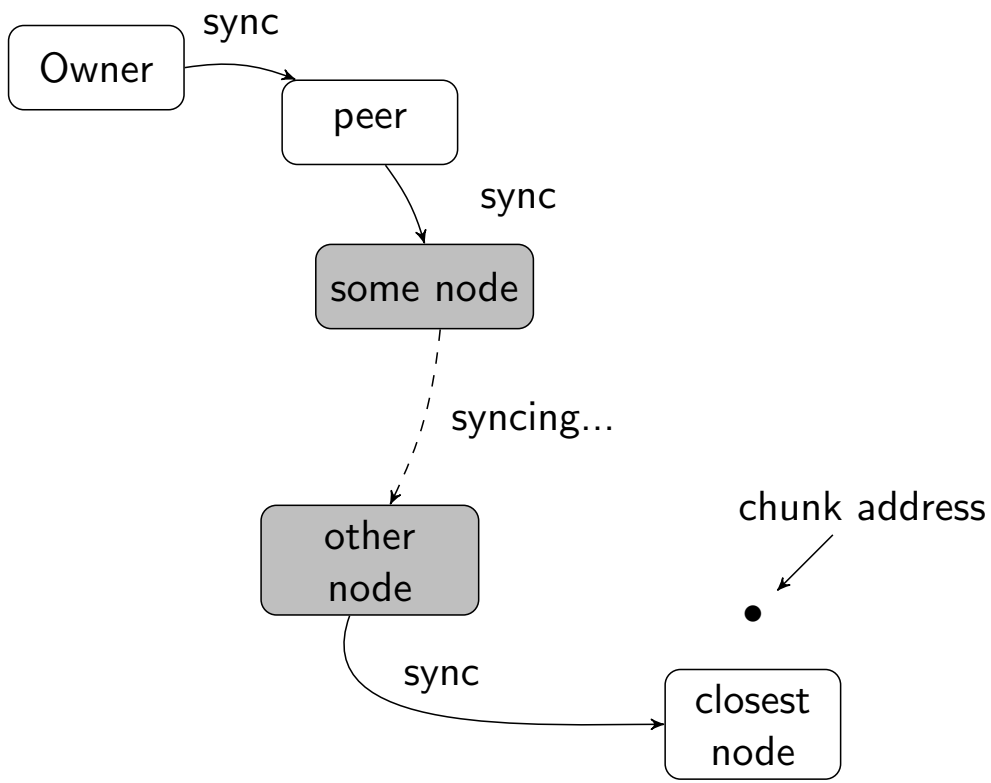
- Chunks are to be stored at the node whose address is closest to the chunk ID.
- Instead of directly connecting to that node and giving it the chunk in question, we pass the chunk along to one of our connected peers who is a little closer to the chunk ID than we are. "Syncing".
- This peer will repeat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



The syncing process.

The normal process of getting chunks of data 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.



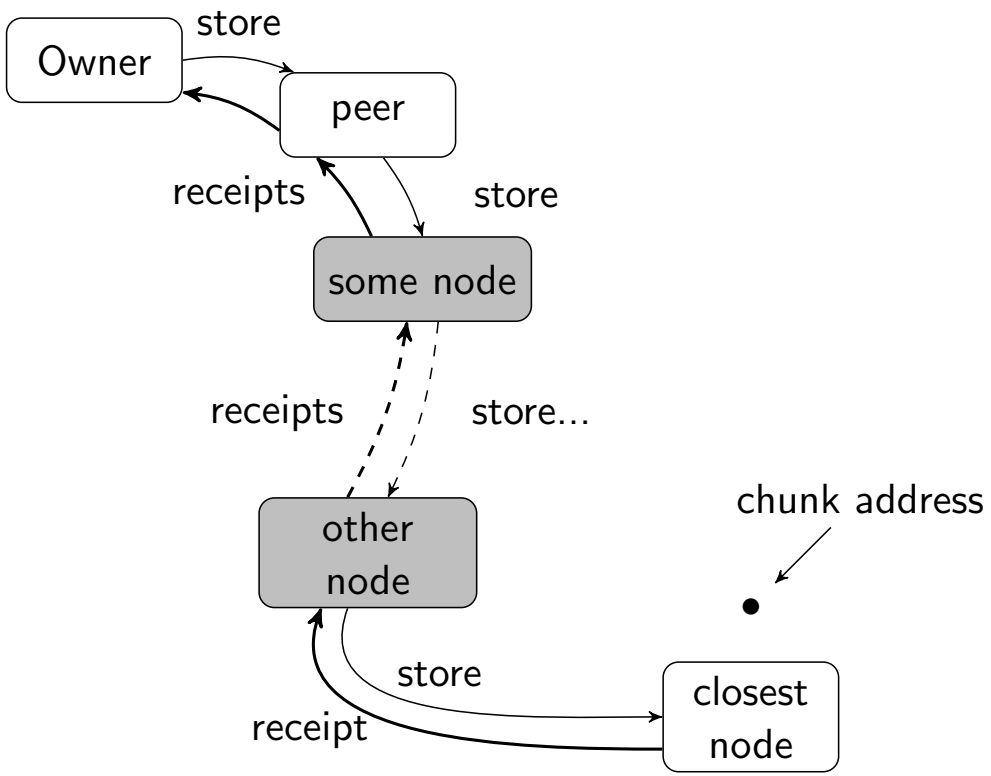
Save-to-swarm

"Saving data to the swarm", i.e. *insured storage*, uses the same basic mechanism as syncing, except that it involves only registered nodes and that every step is receipted.

Save-to-swarm

Insured storage:

- Owner passes data to a registered peer and receives an insurance receipt.
- This process is repeated until closest registered node has the data.
- All receipts are accounted and paid for.



SWINDLE: **S**ervice **W**ith **IN**surance **D**eposit **L**itigation and **E**scrow

SWINDLE

TL;DR

If insured data is lost, the storers lose their deposit.

Litigation upon data loss

If insured data is lost, anyone holding a valid receipt can launch the litigation procedure.
A node so challenged may defend itself by presenting

1. the data itself
2. proof-of-custody of the data
3. a storage receipt for the data, passing the blame and implicating another node as the culprit.

Only at the time of litigation is the chain from owner to storer explicitly determined. This is an important feature – litigation may take time but inital storage can be fast allowing you to 'upload and disappear'.

SWAP • SWEAR • SWINDLE

Outline

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.*

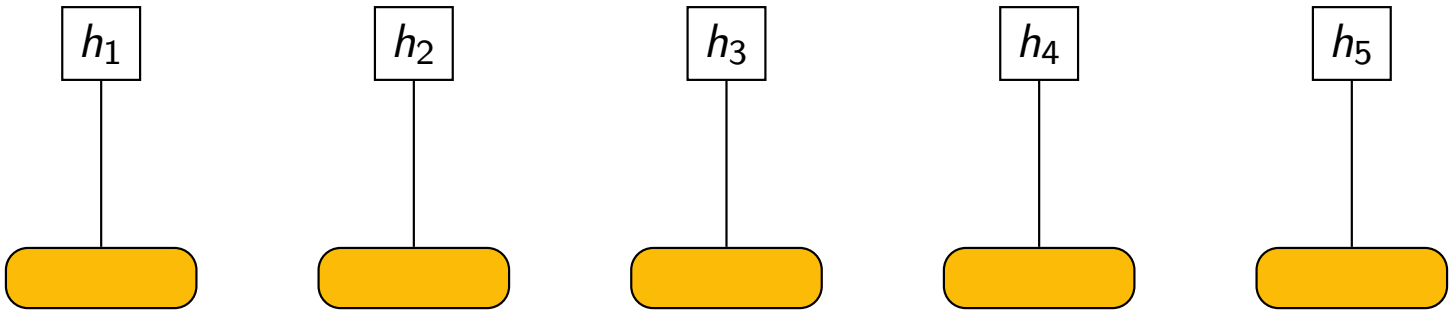
- All data is broken into pieces of size 4kB: “chunks”.

A “chunk:” 

Chunks

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

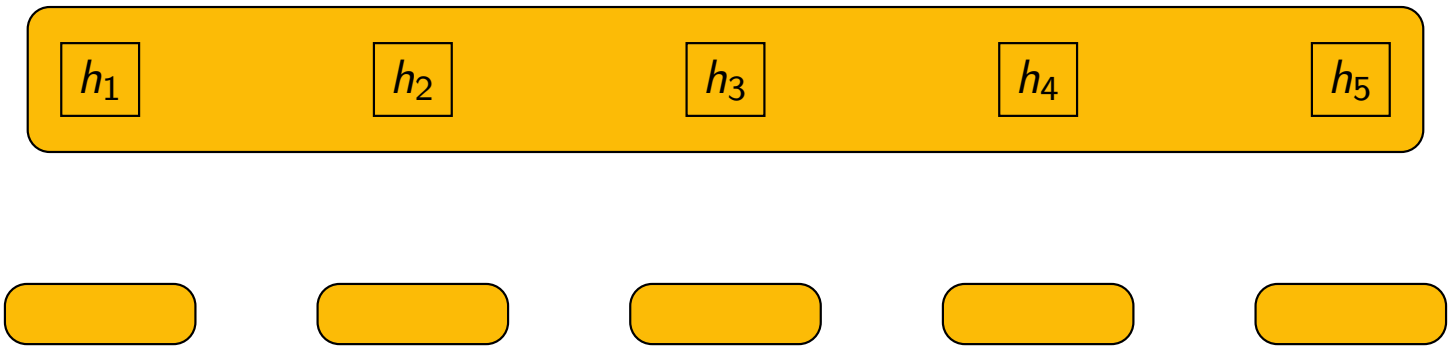
- All data is broken into pieces of size 4kB: “chunks”.
- Chunks are hashed and the hash is used as their ID/address.



It's chunks all the way down...

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

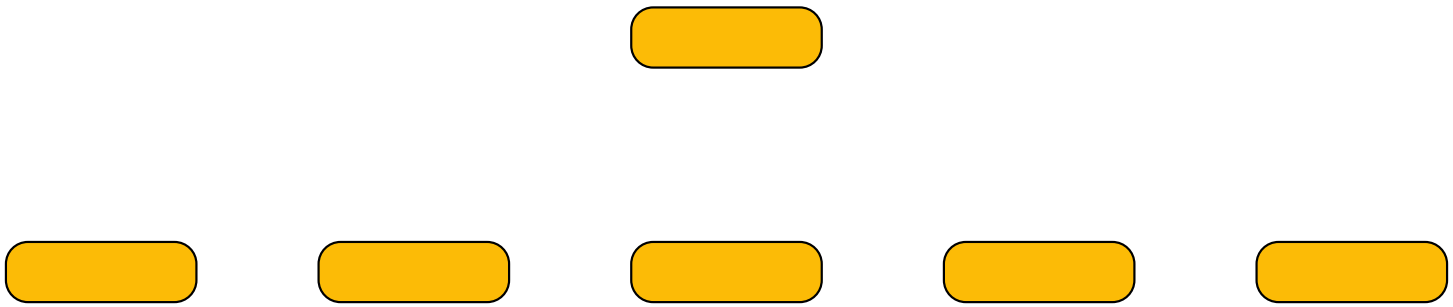
- All data is broken into pieces of size 4kB: "chunks".
- Chunks are hashed and the hash is used as their ID/address.
- Chunk hashes are also packaged into 4kB chunks...



It's chunks all the way down...

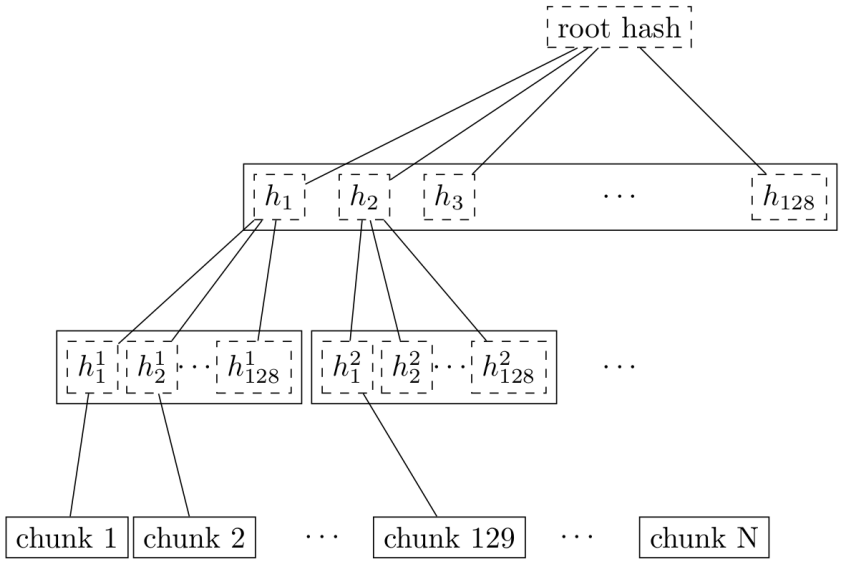
Under the hood swarm does not deal in files but in *chunks.*

- 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...



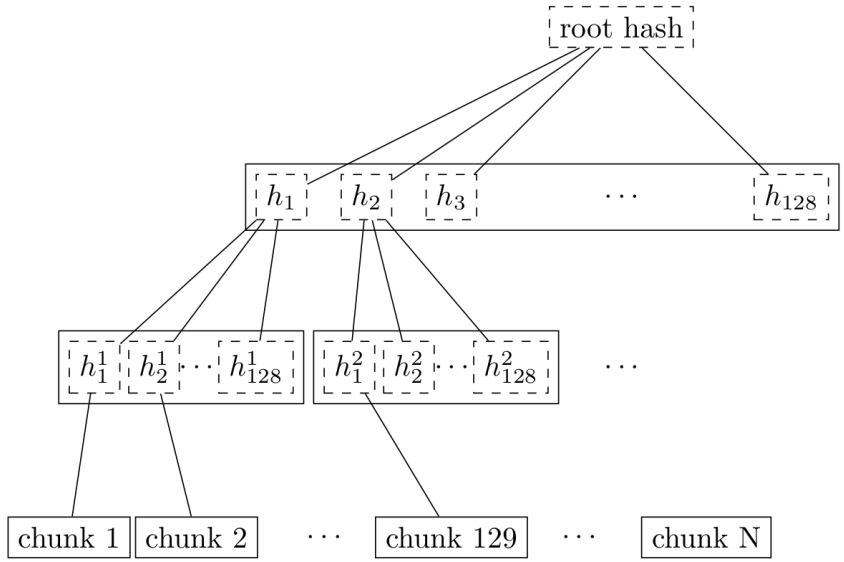
Chunks are assembled in a **Merke Tree**.

- Files are retrievable using a single 32byte hash.



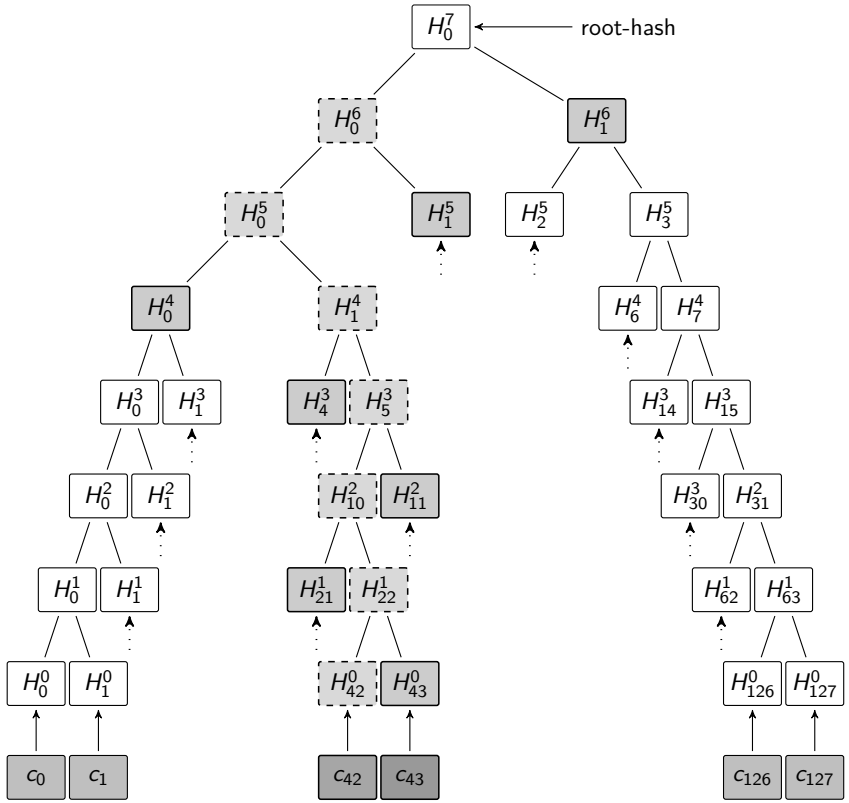
Chunks are assembled in a **Merke Tree**.

- Files are retrievable using a single 32byte hash.
- Built-in integrity protection and random access.



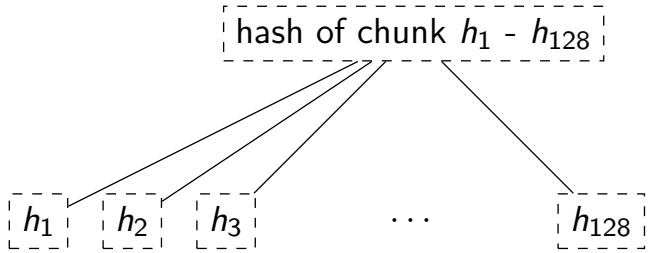
Chunks are assembled in a **Merke Tree**.

- Files are retrievable using a single 32byte hash.
- Built-in integrity protection and random access.
- Merkle-proofs enable proof-of-custody schemes.



Chunks are assembled in a **Merke Tree**.

- Files are retrievable using a single 32byte hash.
- Built-in integrity protection and random access.
- Merkle-proofs enable proof-of-custody schemes.
- Treversible using ASCII charactes due to branching factor of 128.



Manifests

We can take this one step further, be tying together various swarm assets under a new root-hash by generating a new tree: A **Manifest**.

A Swarm Manifest...

...is a Merkle tree whose leaves are root-hashes of other swarm assets (files, collections, manifests, chunks...)

The only difference between this and the chunk-tree of a file, is that it is not balanced and has metadata.

For example,

For example, the Swarm landing page

`swarm-gateways.net/bzz:/swarm/`

SWARM

SERVERLESS HOSTING INCENTIVISED PEER-TO-PEER STORAGE AND CONTENT DISTRIBUTION

orange paper series
talks on swarm
code and status
contact
online press

Swarm is a distributed storage platform and content distribution service, a native base layer service of the ethereum web 3 stack. The primary objective of Swarm is to provide a sufficiently decentralized and redundant store of Ethereum's public record, in particular to store and distribute Ðapp code and data as well as block chain data. From an economic point of view, it allows participants to efficiently pool their storage and bandwidth resources in order to provide the aforementioned services to all participants.

From the end user's perspective, Swarm is not that different from WWW, except that uploads are not to a specific server. The objective is to peer-to-peer storage and serving solution that is DDOS-resistant, zero-downtime, fault-tolerant and censorship-resistant as well as self-sustaining due to a built-in incentive system which uses peer to peer accounting and allows trading resources for payment. Swarm is designed to deeply integrate with the devp2p multiprotocol network layer of Ethereum as well as with the Ethereum blockchain for domain name resolution, service payments and content availability insurance.

orange paper series

The ΞTHΞЯSPHΞЯΞ orange paper series is an attempt to provide an umbrella for sharing and publishing cutting edge research about various aspects of the ethersphere. Our aim is to foster synergy between groups and individuals by creating a frictionless, collaborative editing platform with reputation, endorsement system based an ontology of skill categories, peer review, promotion, meme provenance tracking.

Swarm incentive system research papers. Call for peer review, proposals for improvement, criticism, encouragement and general feedback.

- **Viktor Trón, Aron Fischer, Dániel Nagy A and Zsolt Felföldi, Nick Johnson: swap, swear and swindle: incentive system for swarm. May 2016**

Manifests

...is loaded from this 4-entry manifest:


```
{"entries":[{
"path":"Swarm_files/",
"hash":"0294e48456a49fe7c02162c83b068075ff9ae6aaafb46439dba32da7de548379",
"contentType":"application/bzz-manifest+json",
"status":0},
{"path":"ethersphere/orange-papers/"...
{"path":"i"...
{"path":"talks/"...
{"path":"",
"hash":"6fac0b0c1f118f7f383792c0f01c80d1b2dc94f0e166d62ff4f999a926e9d94a",
"contentType":"text/html;charset=utf-8","status":0}]}
```

Manifests

...is loaded from this 4-entry manifest:


```
{"entries":[{
"path":"Swarm_files/",
"hash":"0294e48456a49fe7c02162c83b068075ff9ae6aaafb46439dba32da7de548379",
"contentType":"application/bzz-manifest+json",
"status":0},
{"path":"ethersphere/orange-papers/"...
{"path":"i"...
{"path":"talks/"...
{"path":"",
"hash":"6fac0b0c1f118f7f383792c0f01c80d1b2dc94f0e166d62ff4f999a926e9d94a",
"contentType":"text/html;charset=utf-8","status":0}]}
```

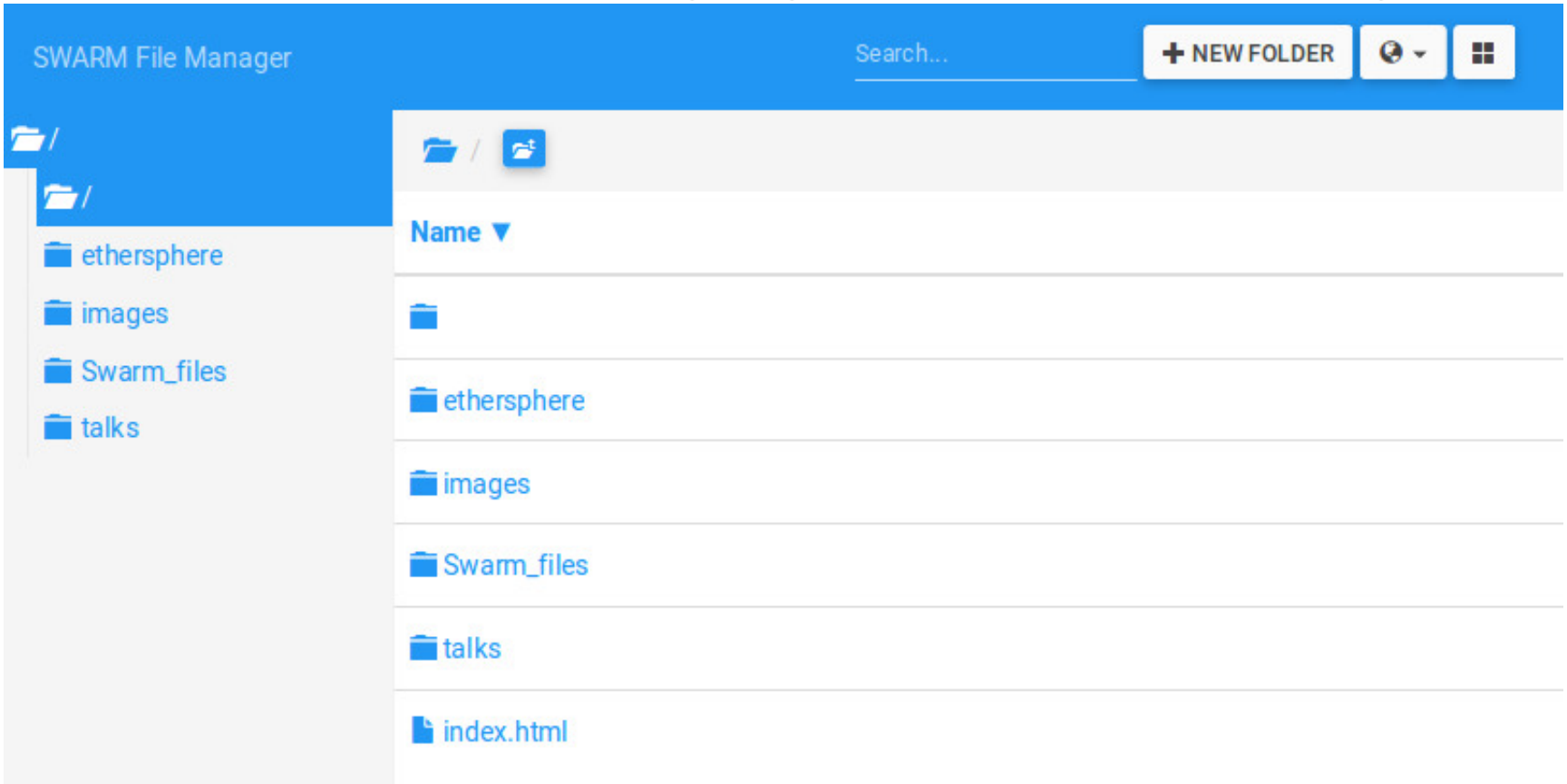

Manifests translate a URL path into swarm hashes (URL defines manifest merkle-tree traversal).

When combined with the Ethereum Name Service (ENS) to register a name for the manifest's own root hash, we can **serve any and all swarm data directly to your browser using human readable names**.

With manifests, you can navigate swarm just like you would navigate your own filesystem.

With manifests, you can navigate swarm just like you would navigate your own filesystem.
Let us open the swarm landing page in the swarm file manager:

With manifests, you can navigate swarm just like you would navigate your own filesystem.
Let us open the swarm landing page in the swarm file manager:



Using Manifests...

- Two-way translation possible from directories to manifests
  - Filesystem driver (fuse)
  - Filesystem API
  - Dropbox, rsync, ...
- Only root hashes must be registered (ENS) on blockchain. Beyond this the whole site is integrity protected.
- Metadata for any manifest entry can include
  - copyright information
  - access control
  - payment triggers
  - auto-play continuation
  - subscription information
- Version control system (mango = git over swarm)

Outline

How does information (dynamic content) move around?

Using the same routing and incentive system as storage and retrieval.

Internode communication

What kinds of interactions are we used to?

- tweets, status updates (public or restricted)
- chatroom, discussion forum, stackoverflow
- pager & fax, phonecall, videocall
- audio-video broadcast, tv, radio, podcast (live or recorded)
- rss, subscription, notifications, newsletters
- file transfer, download

Internode communication

What kinds of interactions are we used to?

- tweets, status updates (public or restricted)
- chatroom, discussion forum, stackoverflow
- pager & fax, phonecall, videocall
- audio-video broadcast, tv, radio, podcast (live or recorded)
- rss, subscription, notifications, newsletters
- file transfer, download

Question:

Why are these services provided by private entities and are not part of the basic public infrastructure? After all, everything is just pulling, pushing and storing data.

In order to recover the services we are used to, we must classify data according to storage and delivery criteria.

- Who is it for?
- How should it be stored and transported?
- Encryption?
- What is the context?

In order to recover the services we are used to, we must classify data according to storage and delivery criteria.

- Who is it for?
- How should it be stored and transported?
- Encryption?
- What is the context?

- Is it addressed to specific recipients?
- Should it be (re-)delivered to specific recipients?

In order to recover the services we are used to, we must classify data according to storage and delivery criteria.

- Who is it for?
- How should it be stored and transported?
- Encryption?
- What is the context?

- Should it be stored (at content address) or is it ephemeral?
- Does it have high priority, is it urgent, is latency a factor?
- Should it be archived? is it insured? expiring?
- Is data access recorded/receipted?

In order to recover the services we are used to, we must classify data according to storage and delivery criteria.

- Who is it for?
- How should it be stored and transported?
- Encryption?
- What is the context?

- Is it confidential? Private?

In order to recover the services we are used to, we must classify data according to storage and delivery criteria.

- Who is it for?
- How should it be stored and transported?
- Encryption?
- What is the context?

- reaction to previous communication, content, topic? Comments, answers, corrections
- Existing asset (reference), streaming data, real time feed?
- How should the data be displayed? Timeline or thematic/threaded view

The Vision

The message routing and content delivery system that swarm uses, can also be used to provide for all the above services.

The Vision

The message routing and content delivery system that swarm uses, can also be used to provide for all the above services.

Tools at our disposal

- incentivised message relay (store requests sent towards non-content address must be paid for)
- deterministic routing and message delivery (to complement Whisper)
- priority queues
- insured storage
- taking receipts
- multicast broadcast

PSS: **P**ostal **S**ervices **S**uite (BZZ-whispered)

How can this handle live multimedia sessions?

Multimedia live broadcast: multibitrate low-latency streaming

- leech - continuous data stream from peers
- non-multiplexed multi-bitrate stream offered
- RTSP/MPEG-DASH standard - allows html5 video tag, compatible with most browsers out of the box
- webRTC or FFMEG to generate streams
- multicast tree - solves scalability of media server solutions
- peer to peer symmetry client = server, the same technique for videoconference or even one-on-one AV session
- data goes to viewers via pairwise transmission channels
- peers sit on the multicast chain and get promoted, demoted depending on payment and latency/throughput
- the same framework can drive historical syncing

SWATCH: **S**treaming **W**ith **A**daptive **T**ransmission **CH**annels

Where is information (dynamic content) pulled from?

1. the blockchain, ethereum state & contract storage. (expensive and slow)
2. local storage private to user, cookies (limited to data only client uses)
3. distributed database on swarm? (cheap and verifiable)

How are database services organised?

- Structure
- Security
- Scalability
- Sustainability

How?

- manifests implement key-value store (note as oppose to traditional DHT)
- supports various indexes and iteration (range queries)
- conventions for index entry
- db table manifest

How are database services organised?

- Structure
- Security
- Scalability
- Sustainability

How?

- verifiable on the blockchain by challange so it does not have to be on-chain
- verifiable authentication and request and notification

How are database services organised?

- Structure
- Security
- Scalability
- Sustainability

How?

- sql resolver (reql of rethinkdb) sitting on top
- parallel processess walk the indexes and merge results
- index updates, derivative data (full text search indexes, aggregate statistics) supplied by a computational market
- query caching and accelerated retrieval for real-time low latency experience supplied by specialised nodes

How are database services organised?

- Structure
- Security
- Scalability
- Sustainability

How?

- due to verifiabiliy of computations, *swap swear and swindle* pattern is applicable

Examples

- example: decentralised markets
- example: storing the blockchain and state on swarm

SWORD: **S**tate **W**ith **O**n-demand **R**etrieval of **D**ata

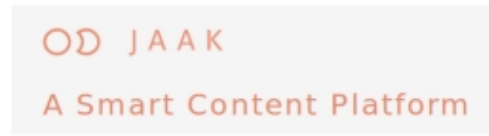
Can we put the ethereum blockchain and state on swarm

- light client, LES protocol abstraction - flexible transition from remote, light, full and archival nodes
- solves the scalability problem of too big state data, receipts, contract storage, fast syncing
- decentralised blockchain explorer

PSS

SWATCH

ENS



Ethereum: status and roadmap

Current Status

'Homestead' up and running. Multiple client implementations.

Roadmap

'Metropolis' release milestone

'Mist browser' - Dapp browser with swarm support.

Swarm: status and usage

What is the developlent status of swarm?

1. golang implementation: proof-of-concept iteration 2 release 4, code has been merged to go-ethereum develop branch
2. Microsoft Azure hosting a testnet of 100+ nodes
3. expanding team, come join or contribute

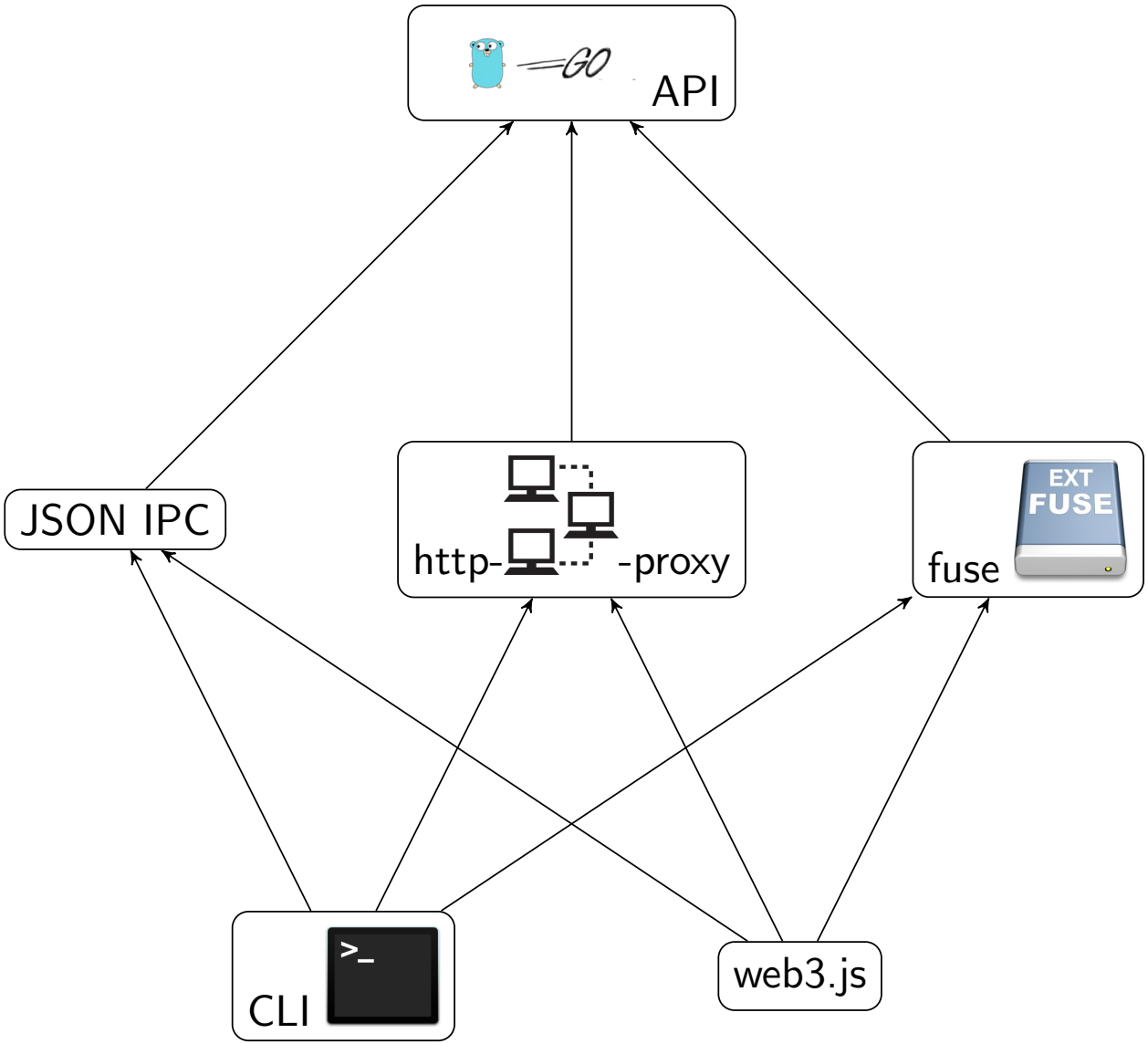
Swarm: status and usage

What is the developlent status of swarm?

1. golang implementation: proof-of-concept iteration 2 release 4, code has been merged to go-ethereum develop branch
2. Microsoft Azure hosting a testnet of 100+ nodes
3. expanding team, come join or contribute

How can swarm be used?

- bzzd - swarm daemon, communicates with ethereum via IPC, so any ethereum client works
- APIs: JSON RPC (via websockets, http, or ipc), http proxy, cli, fuse driver (planned)
- API bindings: web3.js and CLI


GO
API
JSON IPC
http-
-proxy
fuse
EXT
FUSE
CLI
>_
web3.js

Join us

Contact and contribute

- swarm channel: `https://gitter.im/ethereum/swarm`
- orange papers `http://web3.download/bzz:/swarm/`
- join our research channel, reading group and write the orange papers `https://gitter.im/ethersphere/orange-lounge`

The Team

- Daniel A. Nagy, Nick Johnson, Viktor Trón (Zsolt Felföldi) (core team)
- Ram Devish, Bas van Kervel, Alex van der Sande (Mist integration)
- Felix Lange (integration, devp2p)
- Igor Shadurin (file manager dapp)
- Aron Fischer & Ethersphere orange lounge group
- Nick Johnson, Alex van der Sande (Ethereum Name Service)
- Gavin Wood, Vitalik Buterin, Jeffrey Wilcke (visionaries)