SWAP, SWEAR AND SWINDLE: incentive system for swarm and beyond

Viktor Trón and Aron Fischer

September 19, 2016

ethereum

whisper

swarm

Outline

Outline

data out

How to retrieve data stored in the swarm.

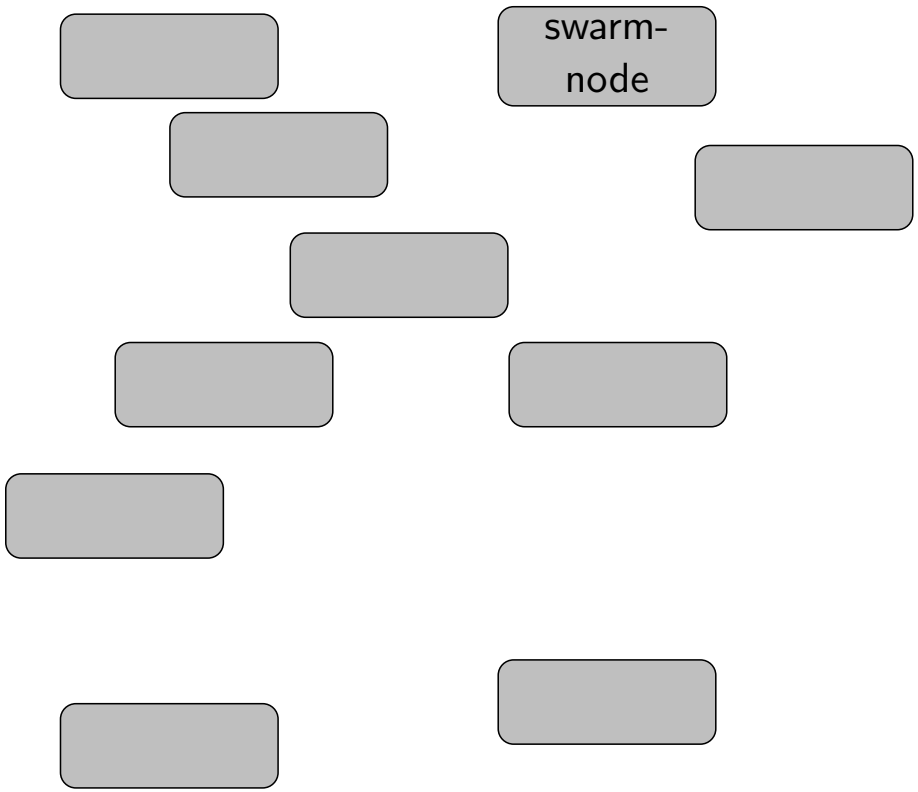
data retrieval

- node id, chunk id, function as addresses in the same keyspace

data retrieval

- node id, chunk id, function as addresses in the same keyspace

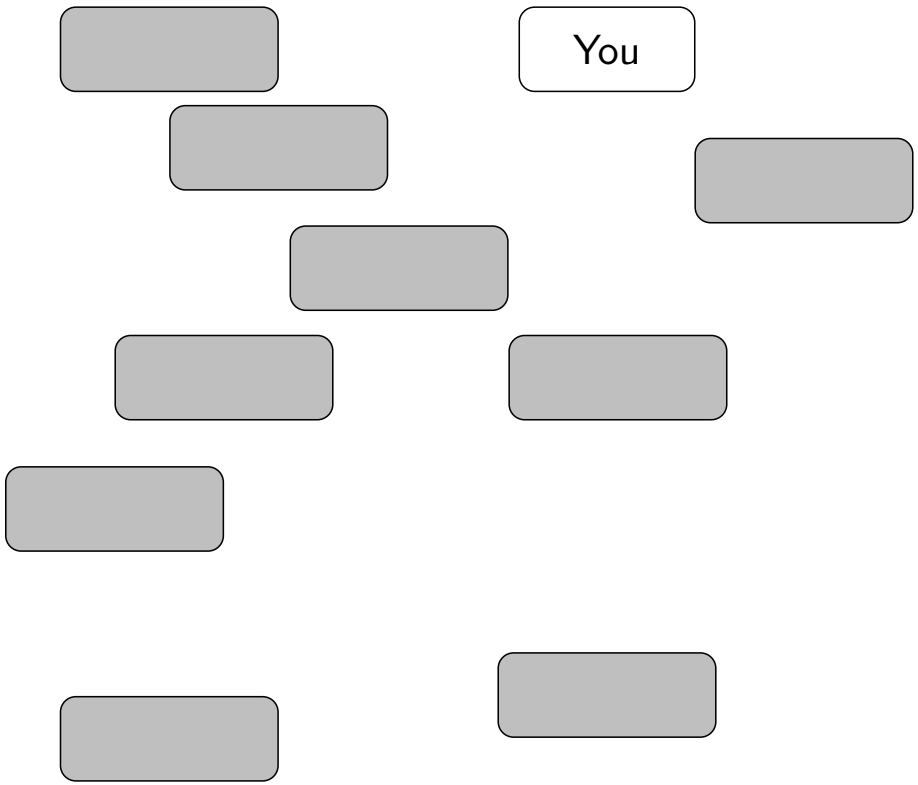
the swarm network:



data retrieval

- node id, chunk id, function as addresses in the same keyspace

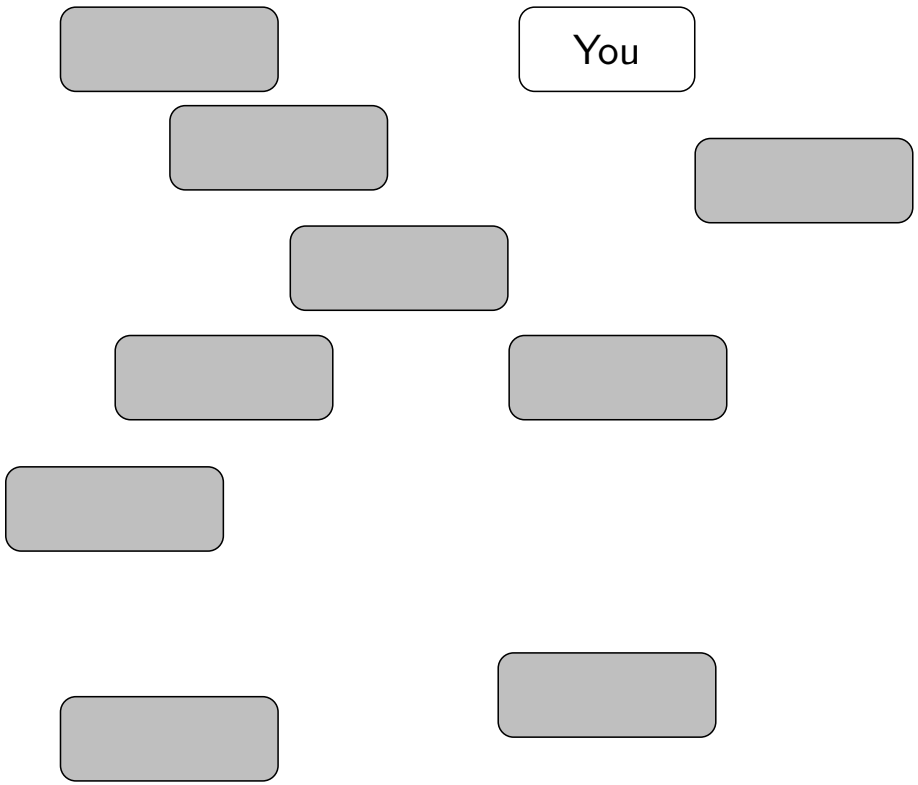
the swarm network:



data retrieval

- node id, chunk id, function as addresses in the same keyspace
- dapp retrieves `awesome-swarm-slides.pdf`

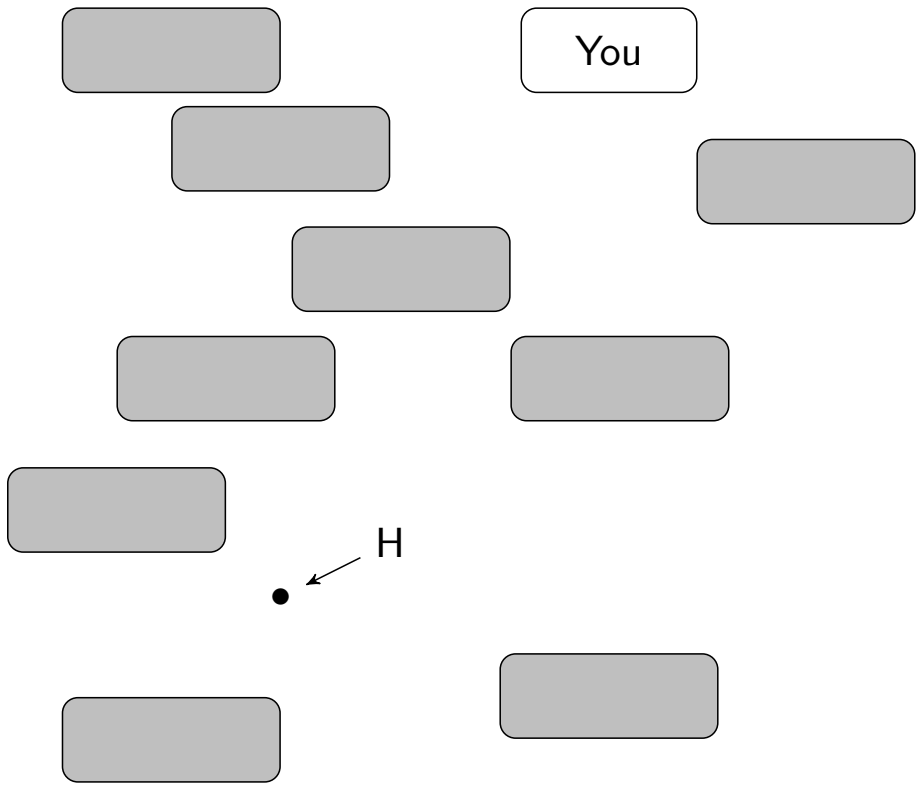
the swarm network:



data retrieval

- node id, chunk id, function as addresses in the same keyspace
- dapp retrieves `awesome-swarm-slides.pdf`
- get its address **H**

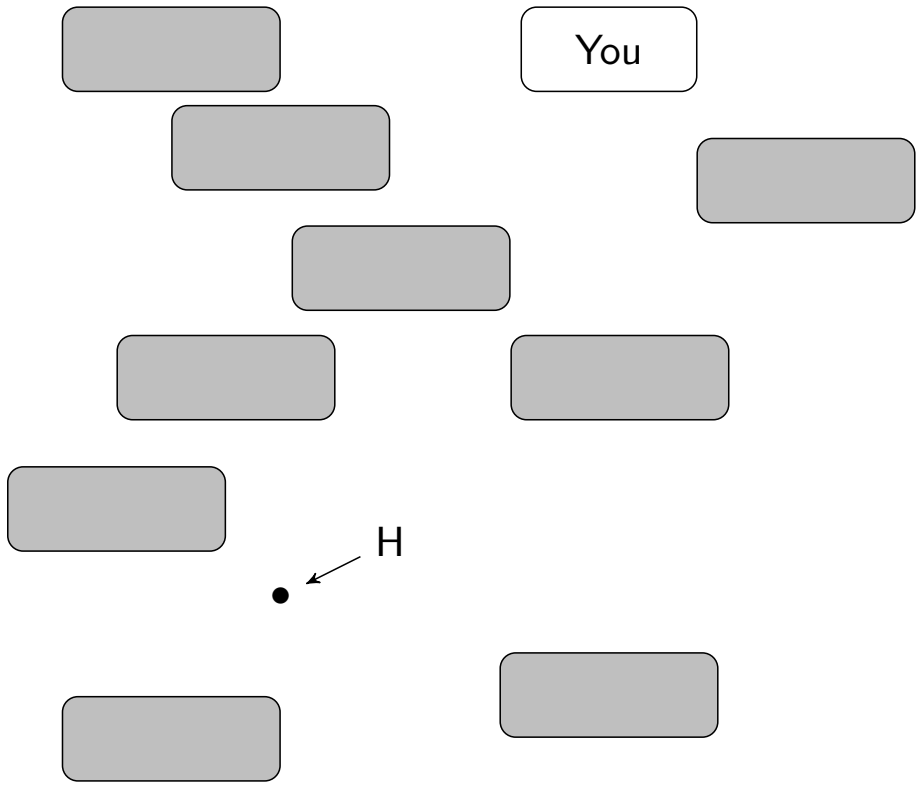
the swarm network:



data retrieval

- node id, chunk id, function as addresses in the same keyspace
- dapp retrieves `awesome-swarm-slides.pdf`
- get its address **H**
- content with address **H** stored with the node whose own address is *closest* to **H**

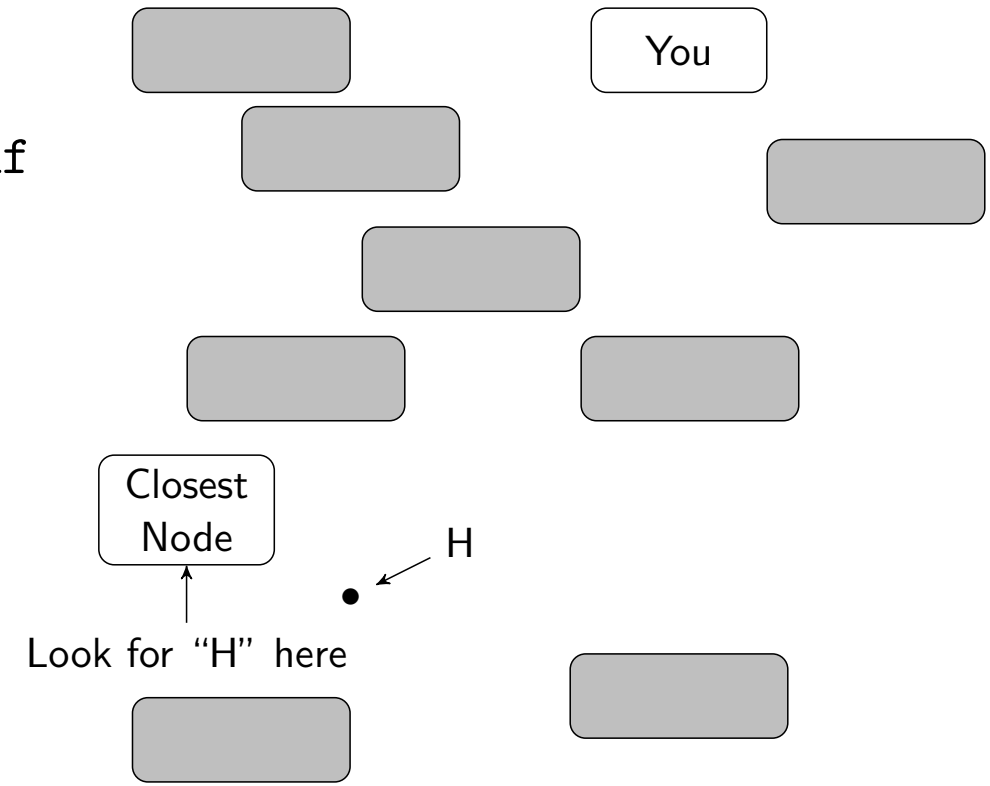
the swarm network:



data retrieval

- node id, chunk id, function as addresses in the same keyspace
- dapp retrieves `awesome-swarm-slides.pdf`
- get its address **H**
- content with address **H** stored with the node whose own address is *closest* to **H**
- swarm's **retrieval process** is responsible for deliviering

the swarm network:



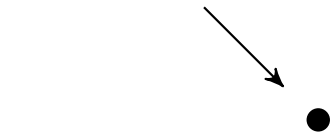
swarm retrieval process

retriever

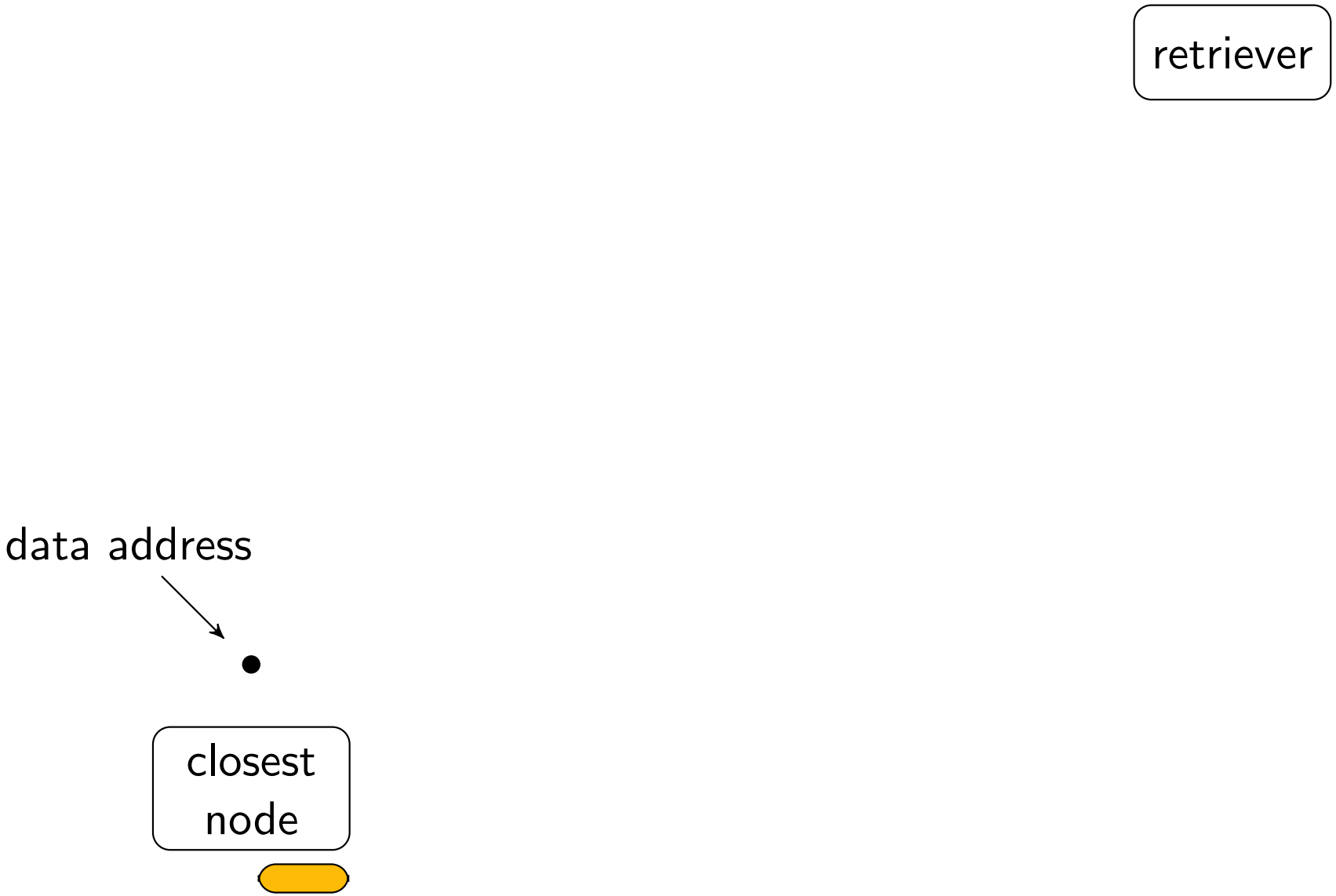
swarm retrieval process

retriever

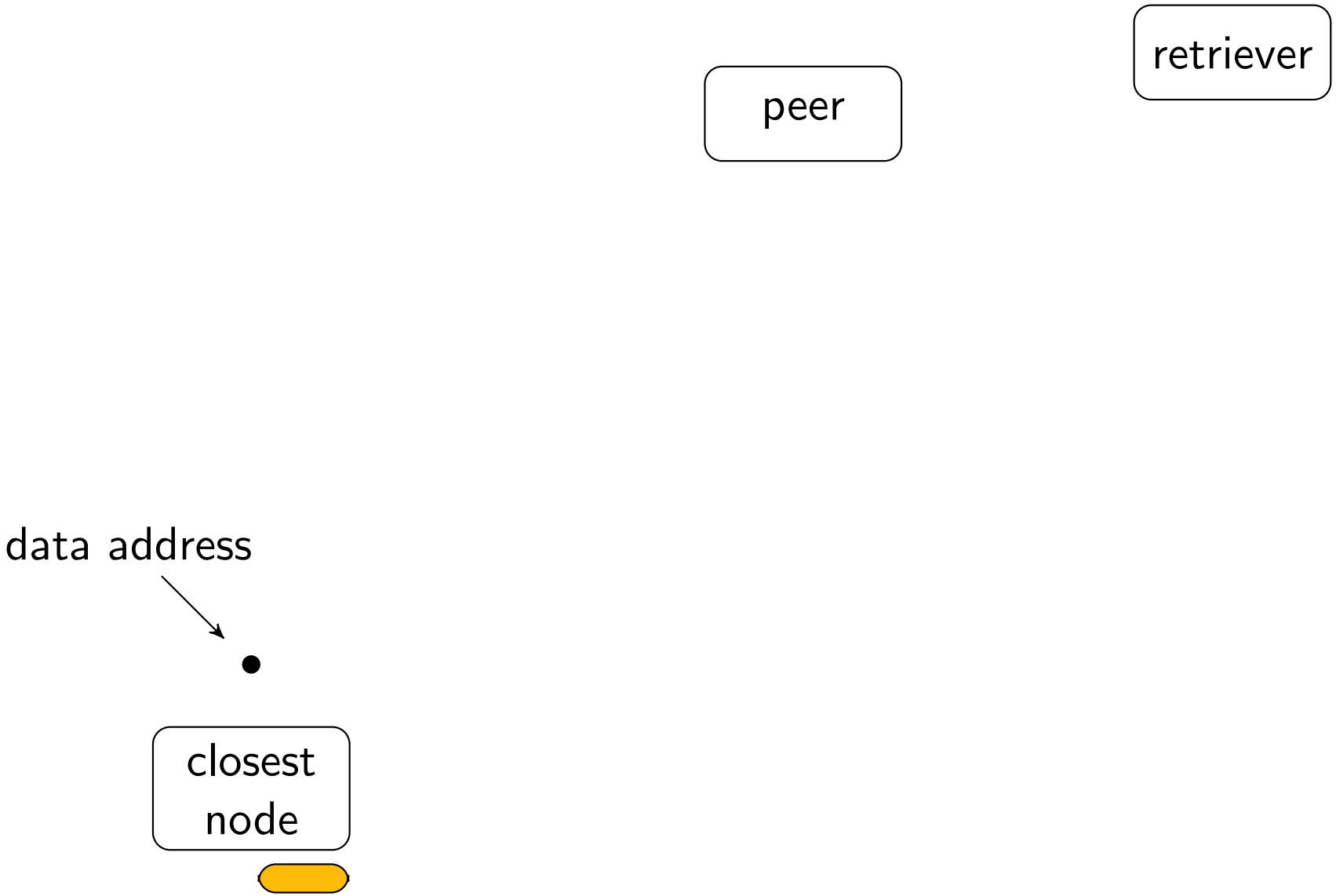
data address



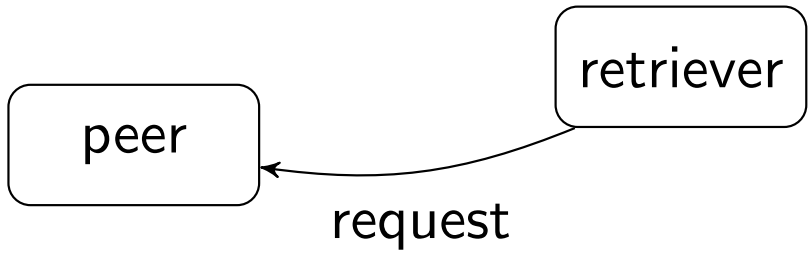
swarm retrieval process

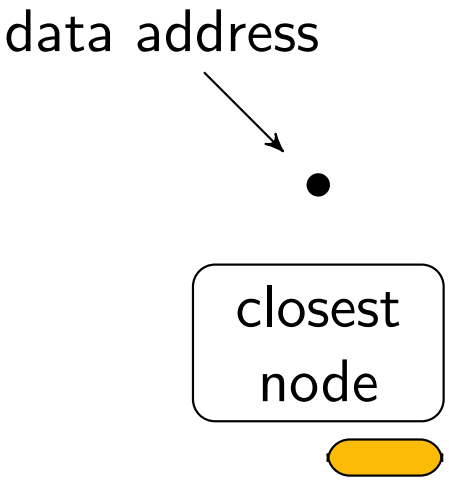


swarm retrieval process

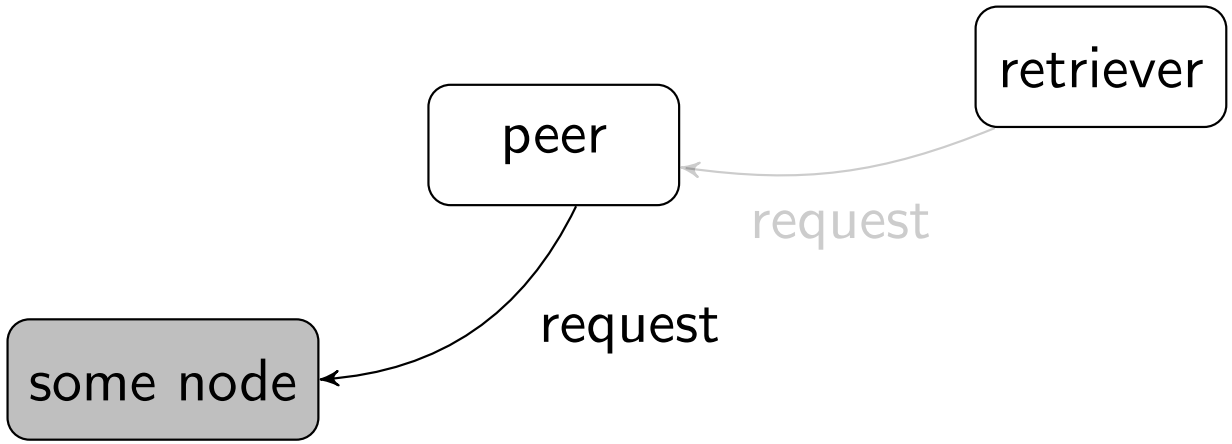


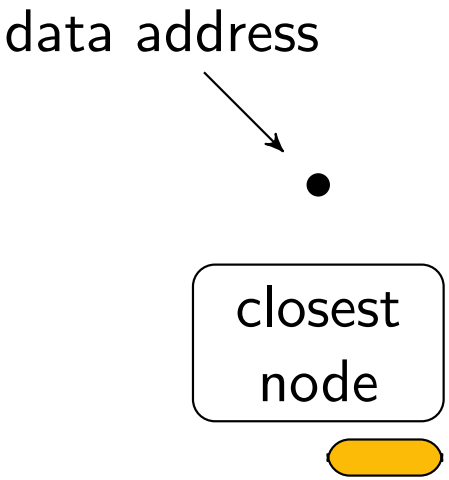
swarm retrieval process



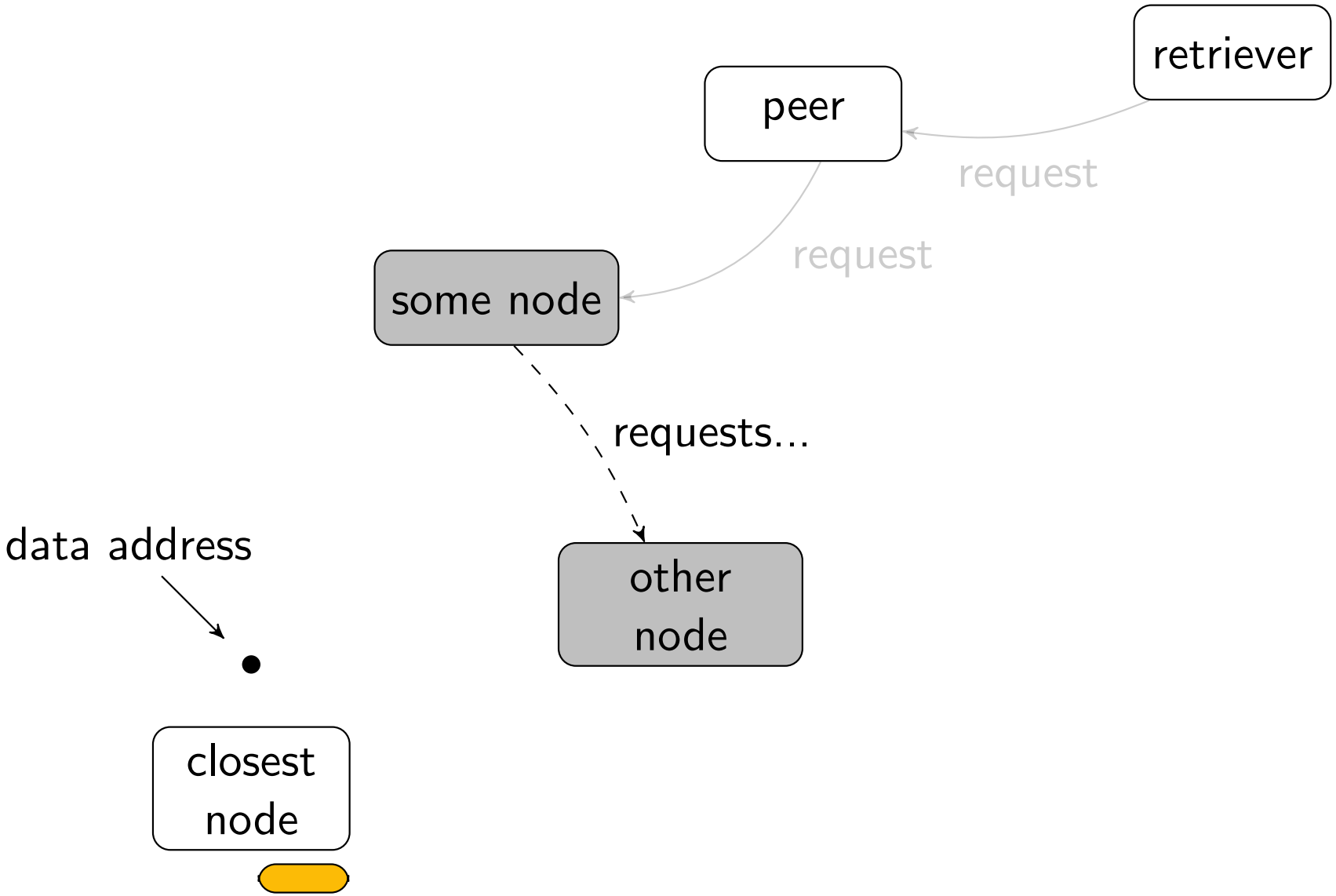


swarm retrieval process

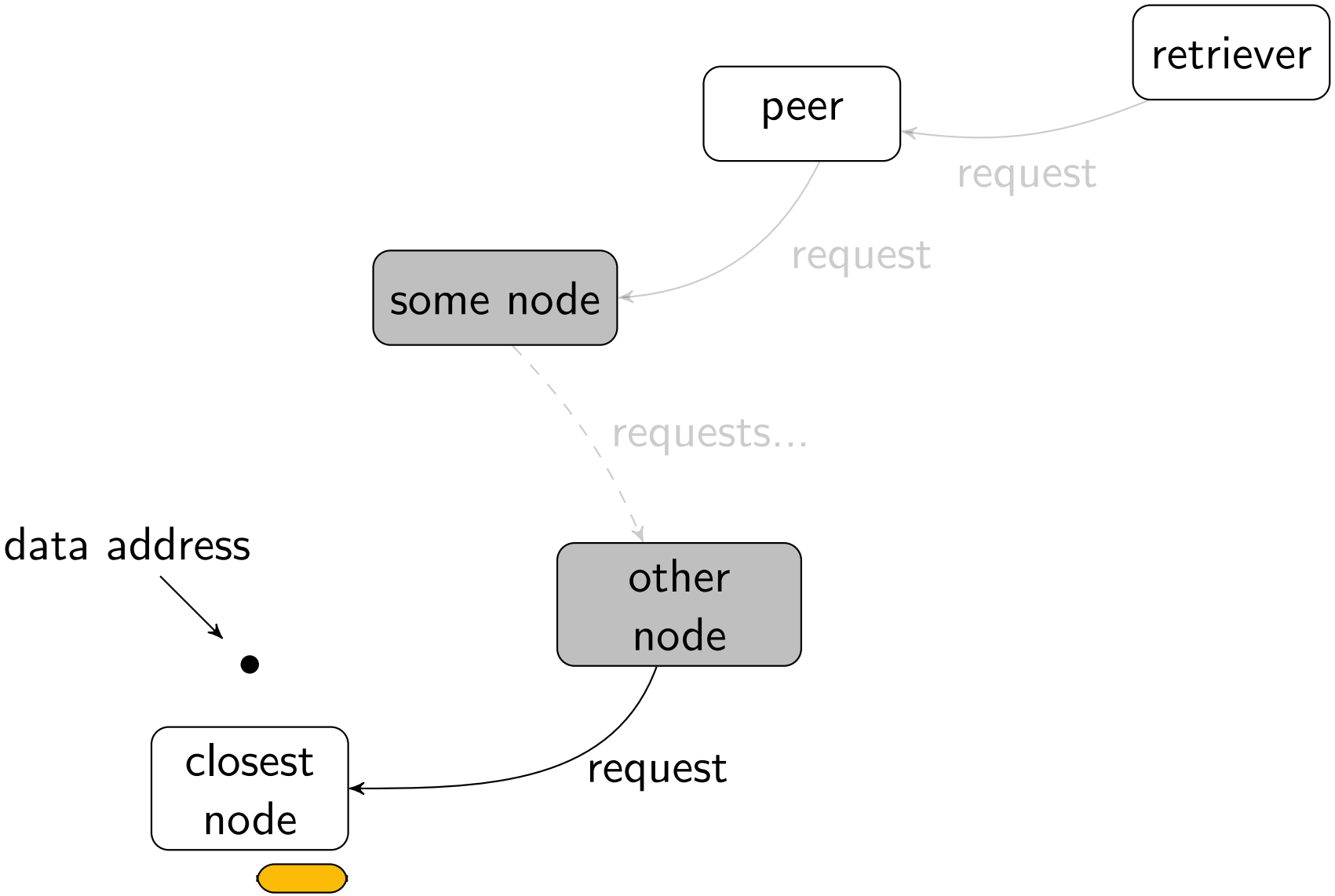




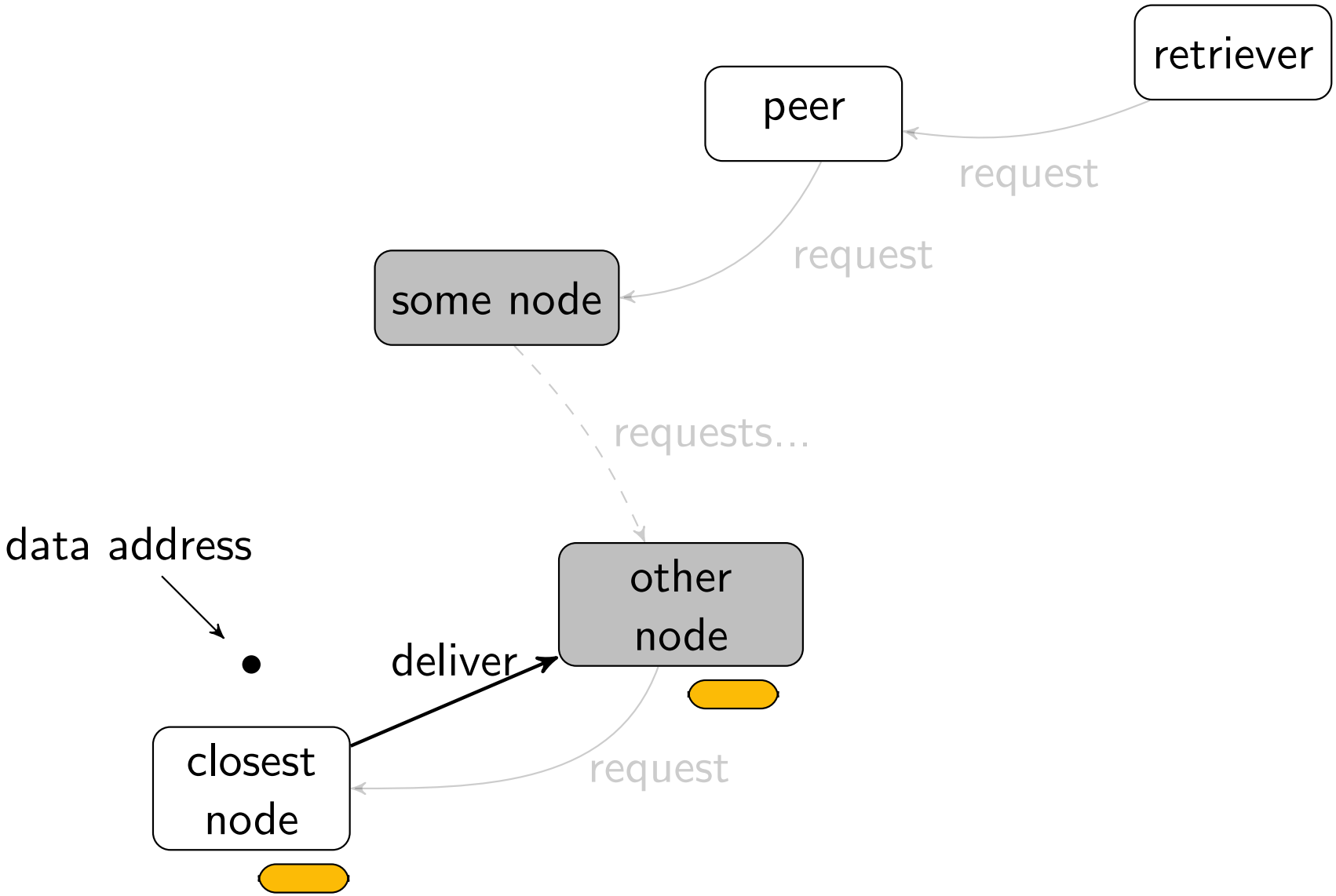
swarm retrieval process



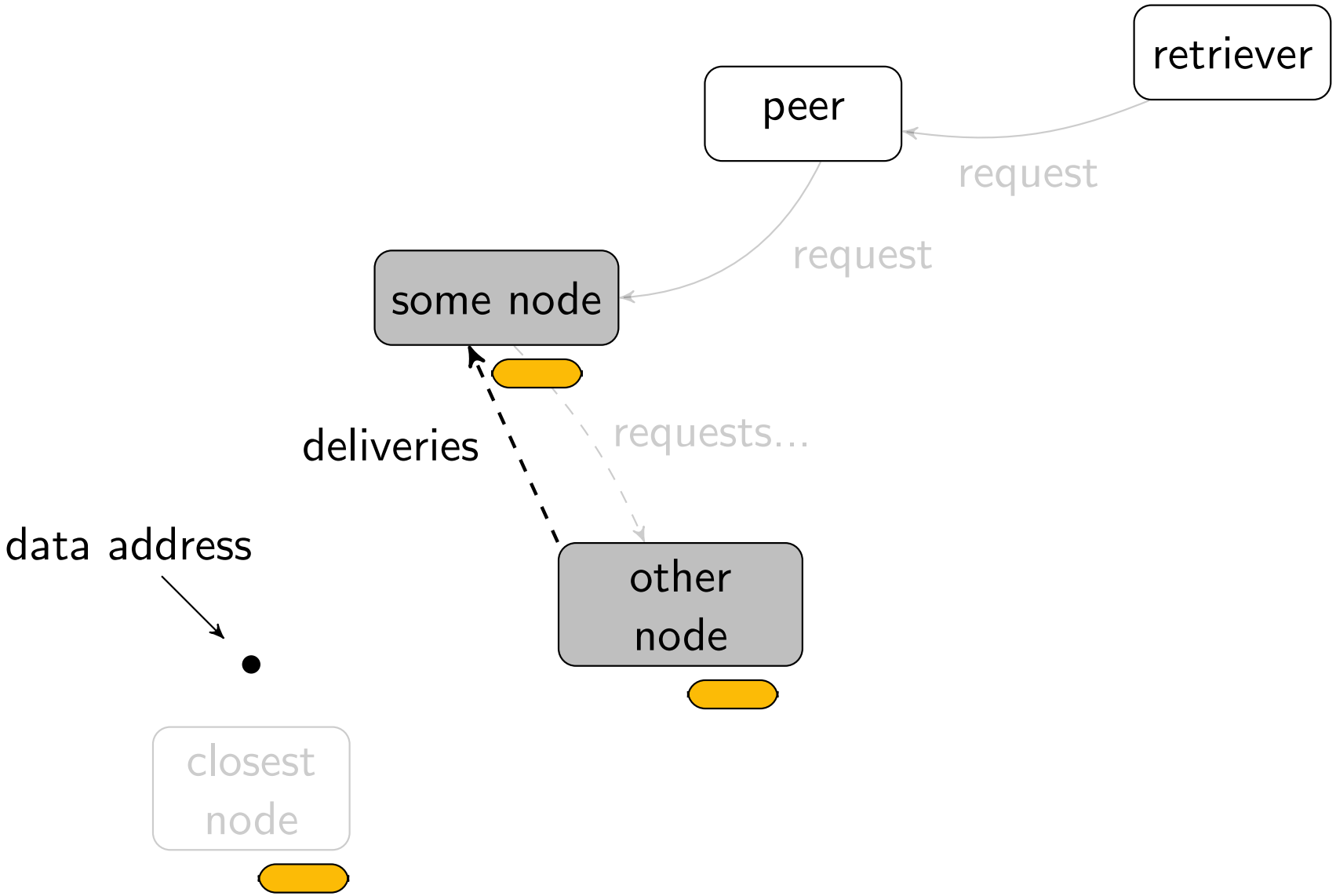
swarm retrieval process



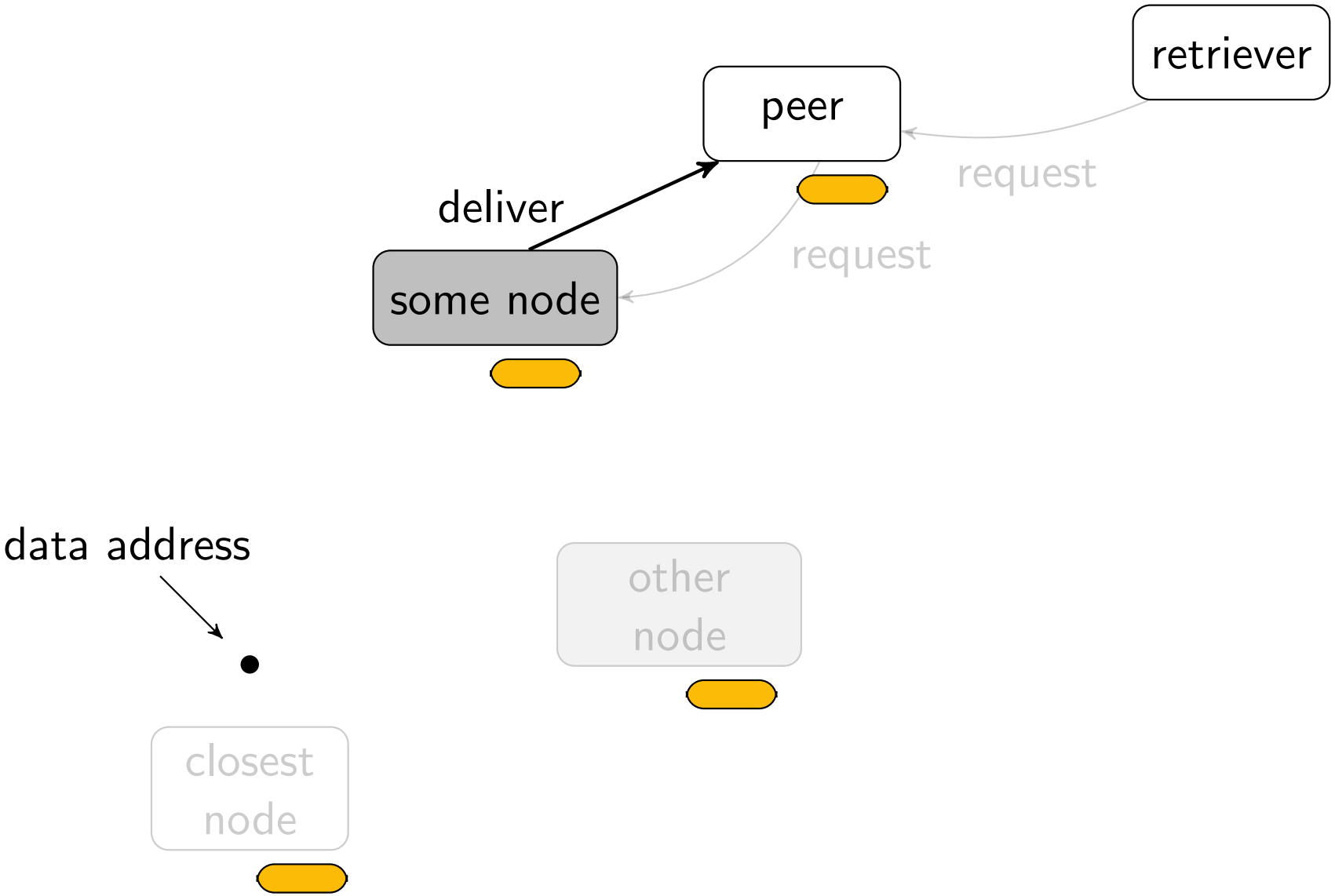
swarm retrieval process



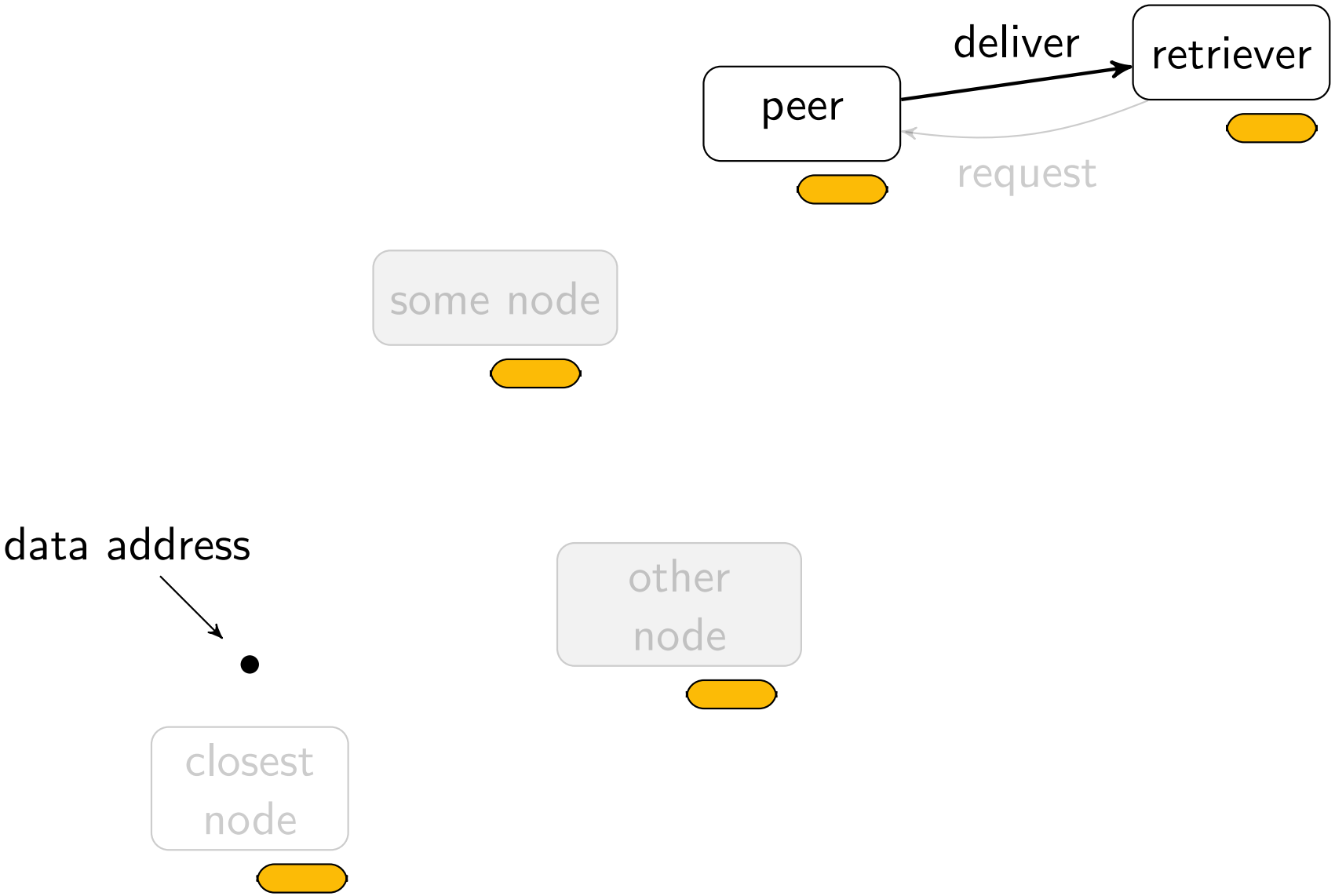
swarm retrieval process



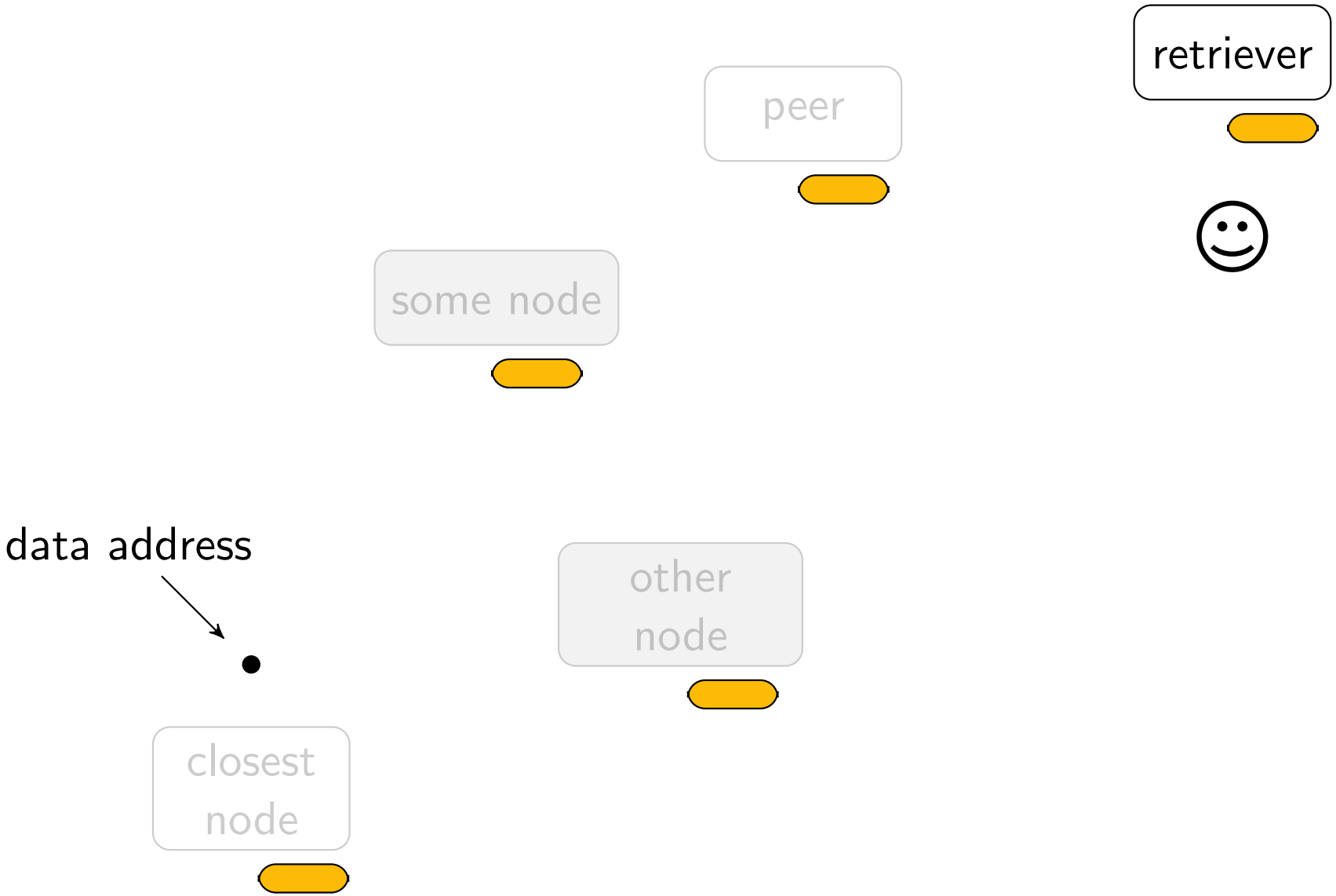
swarm retrieval process



swarm retrieval process



swarm retrieval process



SWAP: **sw**arm **a**ccounting **p**rotocol

SWAP: swarm accounting protocol

SWAP: swarm accounting protocol

per-peer bandwidth accounting

keeps track of all data retrieved
both directions

SWAP: swarm accounting protocol

per-peer bandwidth accounting

keeps track of all data retrieved both directions

me

SWAP: swarm accounting protocol

per-peer bandwidth accounting

keeps track of all data retrieved
both directions

SWAP: swarm accounting protocol

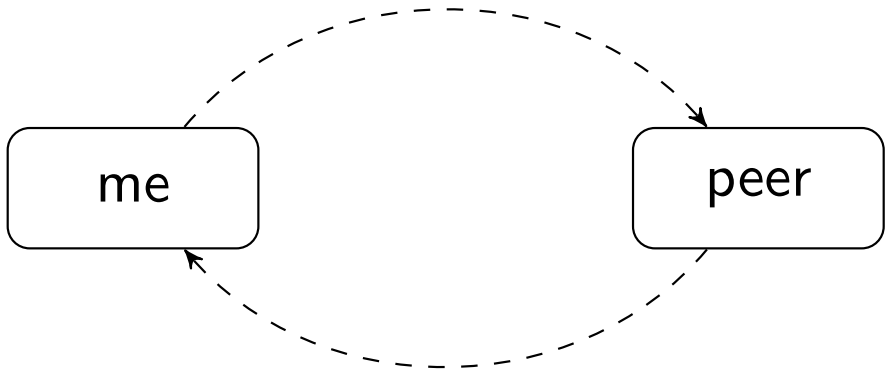
per-peer bandwidth accounting

keeps track of all data retrieved
both directions

SWAP: swarm accounting protocol

per-peer bandwidth accounting

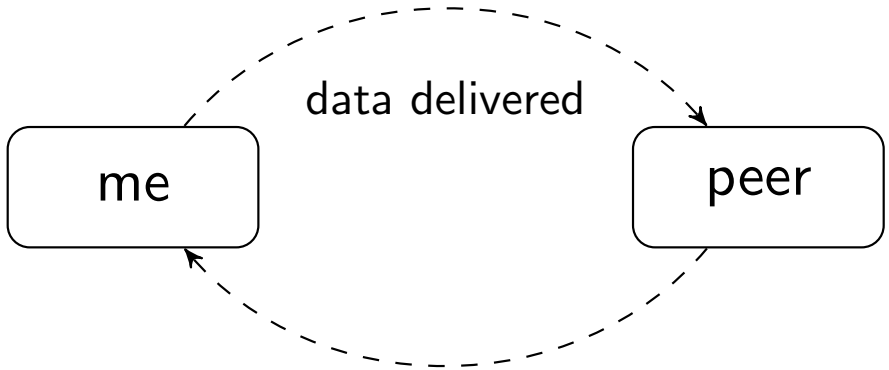
keeps track of all data retrieved
both directions



SWAP: swarm accounting protocol

per-peer bandwidth accounting

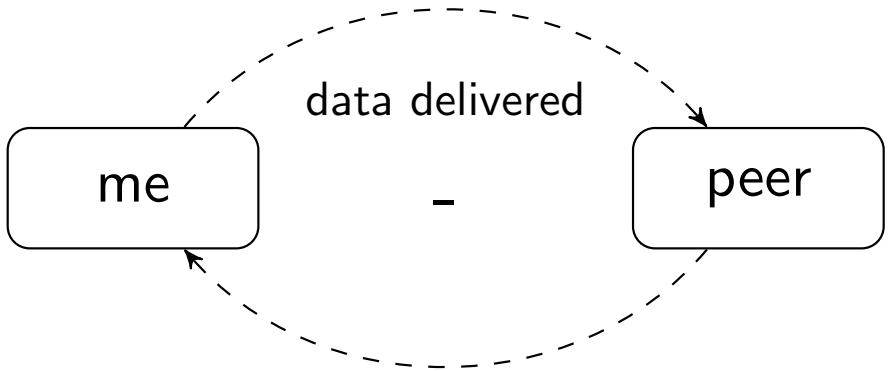
keeps track of all data retrieved
both directions



SWAP: swarm accounting protocol

per-peer bandwidth accounting

keeps track of all data retrieved
both directions



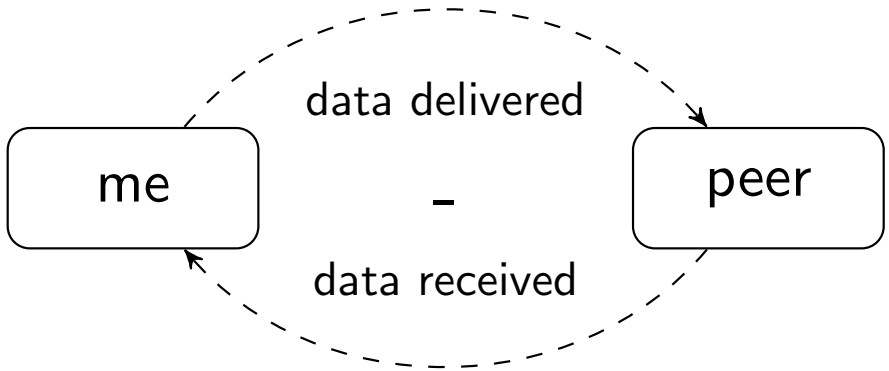
SWAP: swarm accounting protocol

per-peer bandwidth accounting

keeps track of all data retrieved both directions

settlement

service for service or tally too imbalanced $\rightarrow$ a *payment* is initiated



chequebook vs payment channel

- *not feasible* to pay for every chunk of data delivered with a transation

chequebook vs payment channel

- *not feasible* to pay for every chunk of data delivered with a transation
- even batch payments would constitute unacceptable blockchain bloat (and transaction cost).

chequebook vs payment channel

- *not feasible* to pay for every chunk of data delivered with a transation
- even batch payments would constitute unacceptable blockchain bloat (and transaction cost).

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

chequebook vs payment channel

- *not feasible* to pay for every chunk of data delivered with a transation
- 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*, i.e., **only the last cheque needs to be cashed for settlement**.

chequebook vs payment channel

- *not feasible* to pay for every chunk of data delivered with a transation
- 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*, i.e., **only the last cheque needs to be cashed for settlement**.

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

chequebook vs payment channel

chequebook

pro:

- offchain payments
- low barrier to entry (pay anyone)

con:

- cheques can bounce (payment not guaranteed)

channel

pro:

- offchain payments
- secure - payments guaranteed

con:

- high barrier to entry (must first join channel network)

chequebook vs payment channel

chequebook

pro:

- offchain payments
- low barrier to entry (pay anyone)

con:

- cheques can bounce (payment not guaranteed)

channel

pro:

- offchain payments
- secure - payments guaranteed

con:

- high barrier to entry (must first join channel network)

chequebook vs payment channel

chequebook

pro:

- offchain payments
- low barrier to entry (pay anyone)

con:

- cheques can bounce (payment not guaranteed)

channel

pro:

- offchain payments
- secure - payments guaranteed

con:

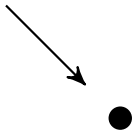
- high barrier to entry (must first join channel network)

SWARM + SWAP demonstrates

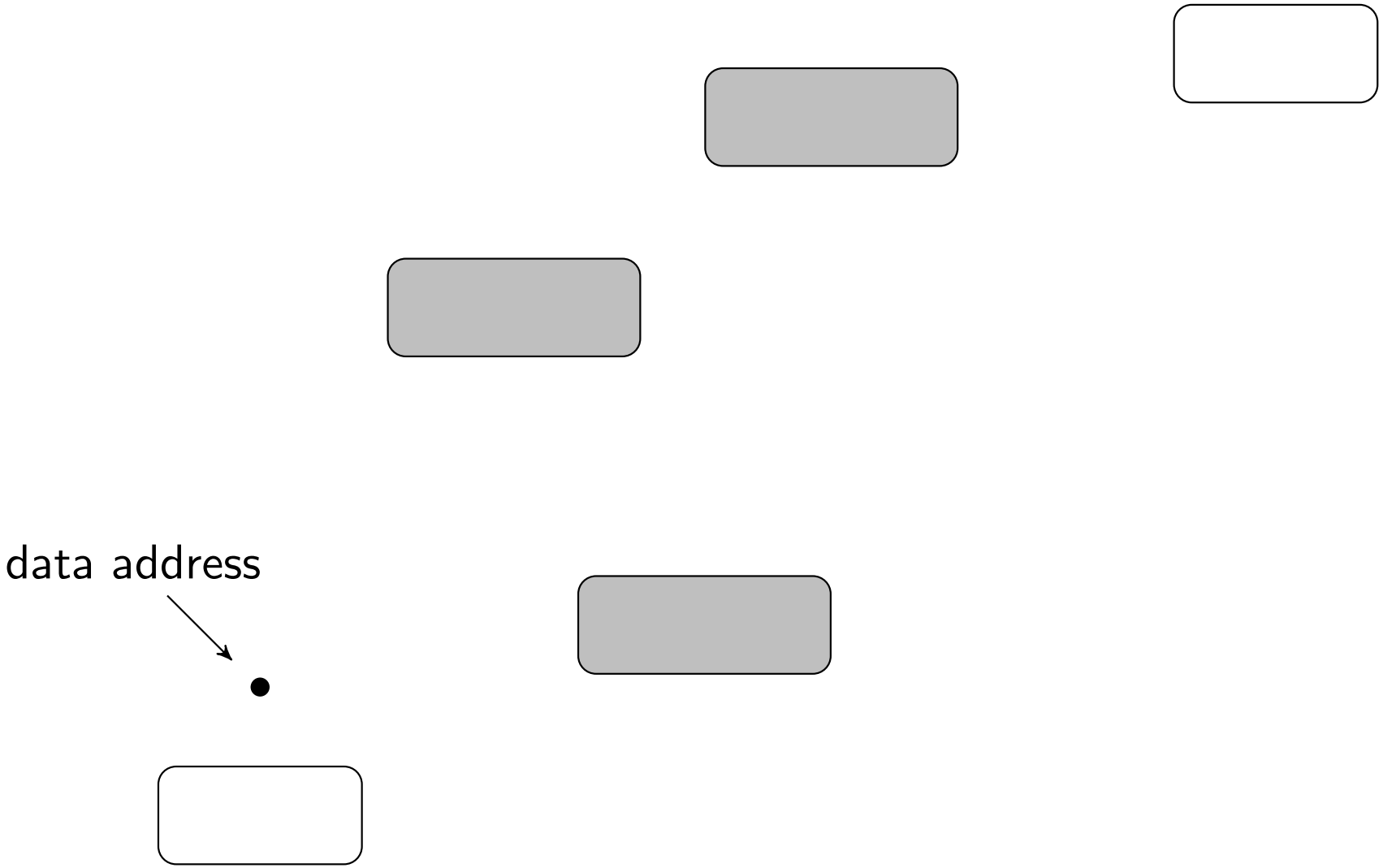
- programmable incentives
- drive towards low latency retrieval
- auto-scaling delivery network

swarm CDN is auto-scaling

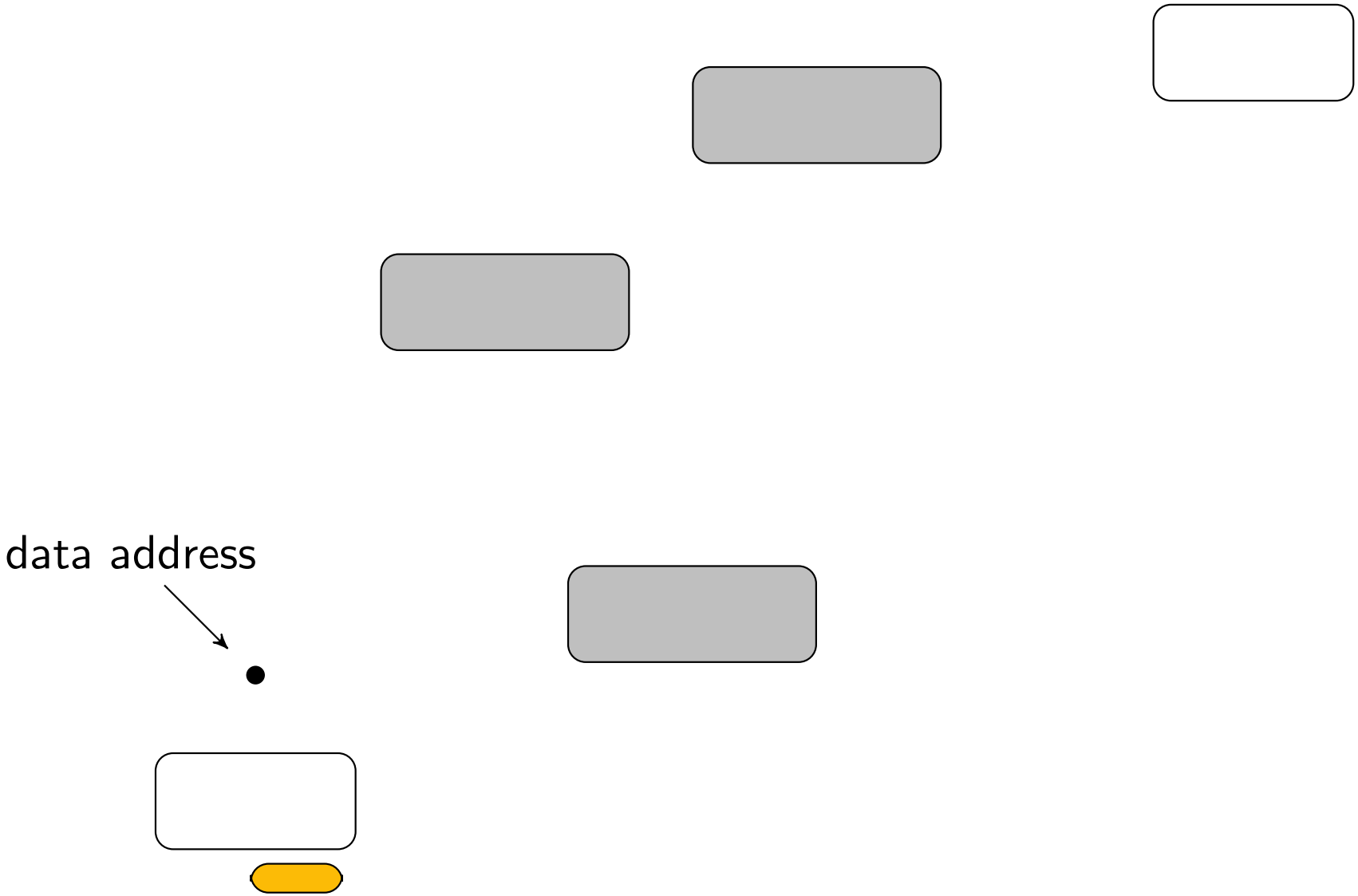
data address



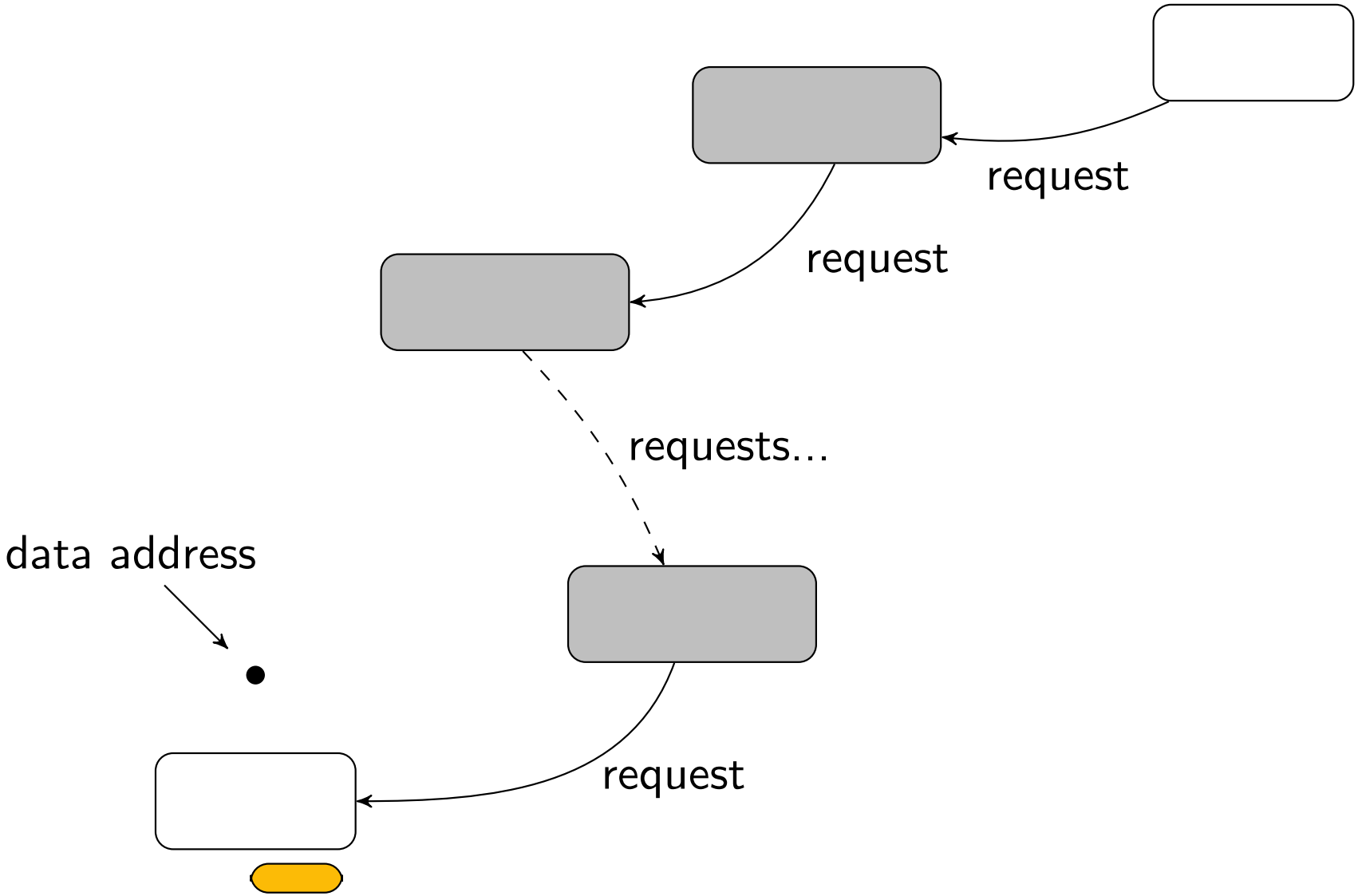
swarm CDN is auto-scaling



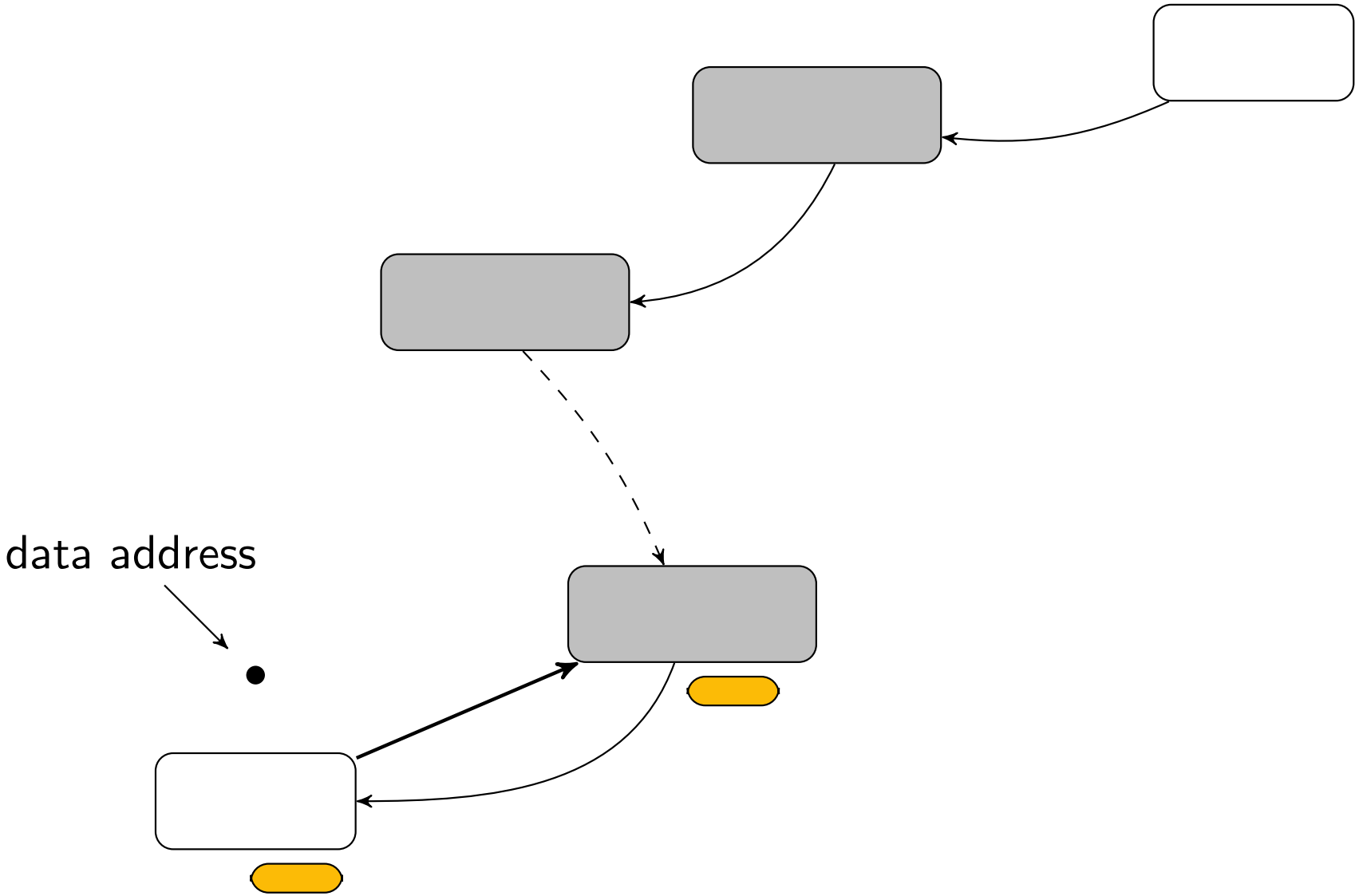
swarm CDN is auto-scaling



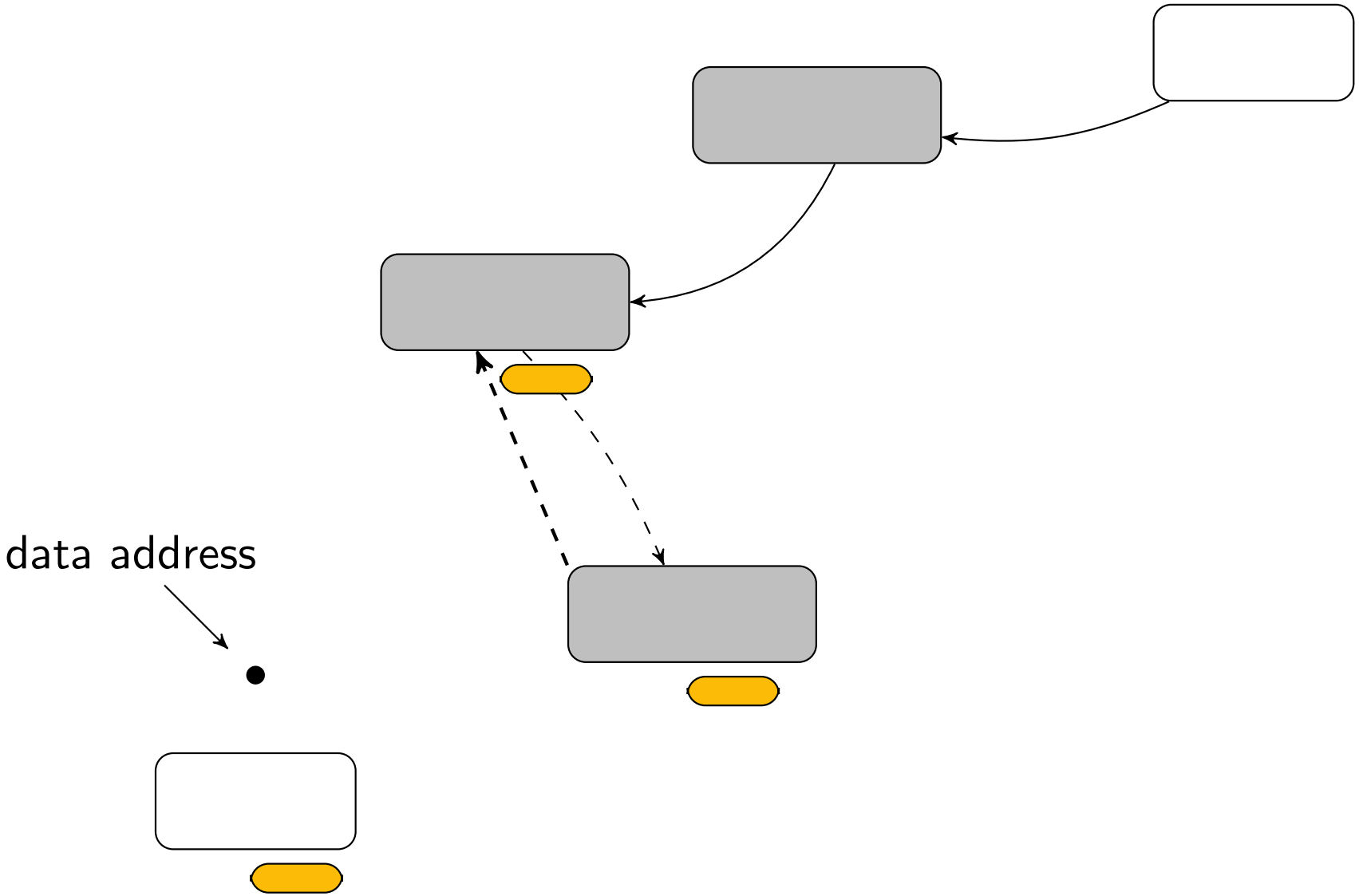
swarm CDN is auto-scaling



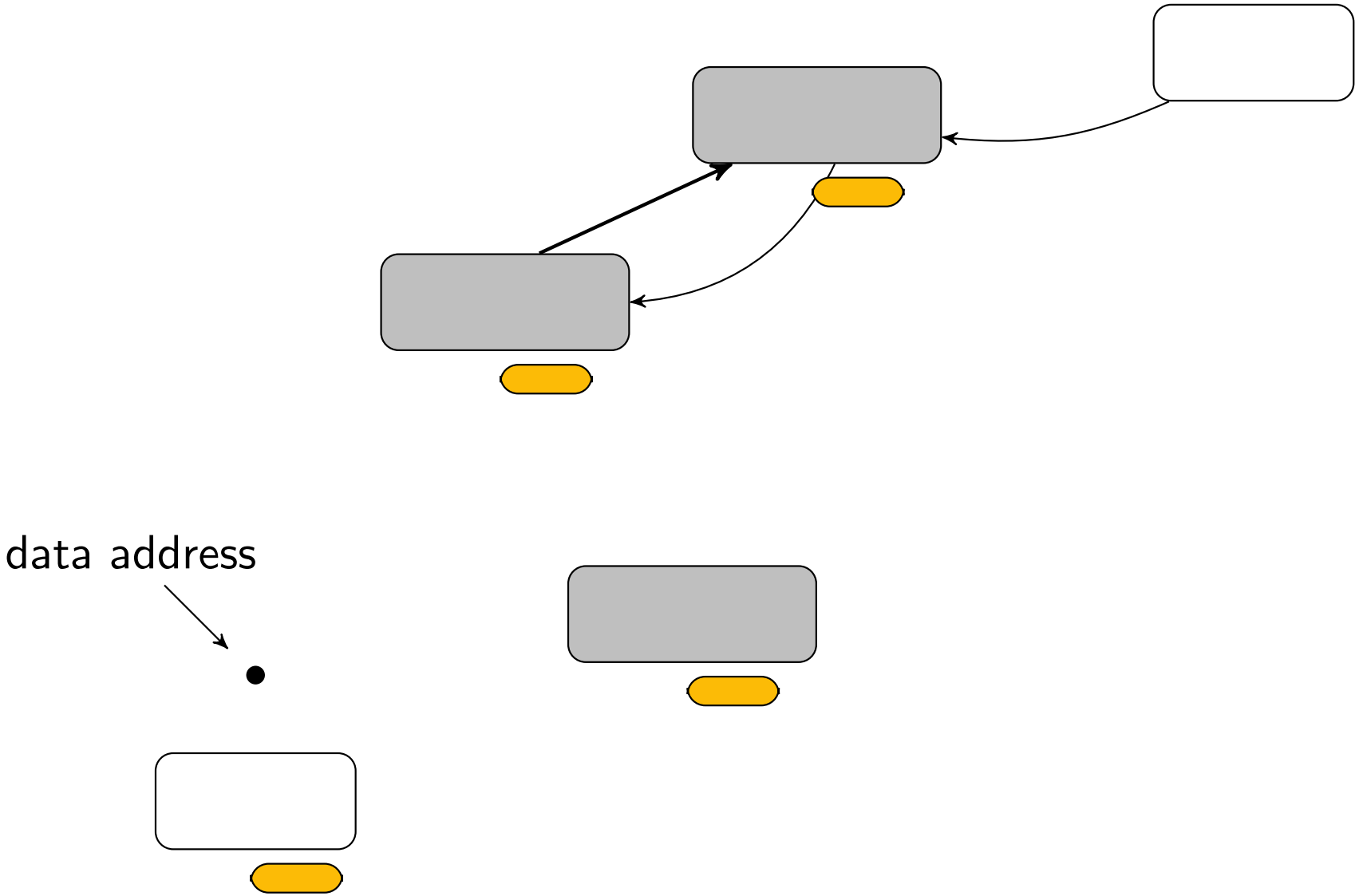
swarm CDN is auto-scaling



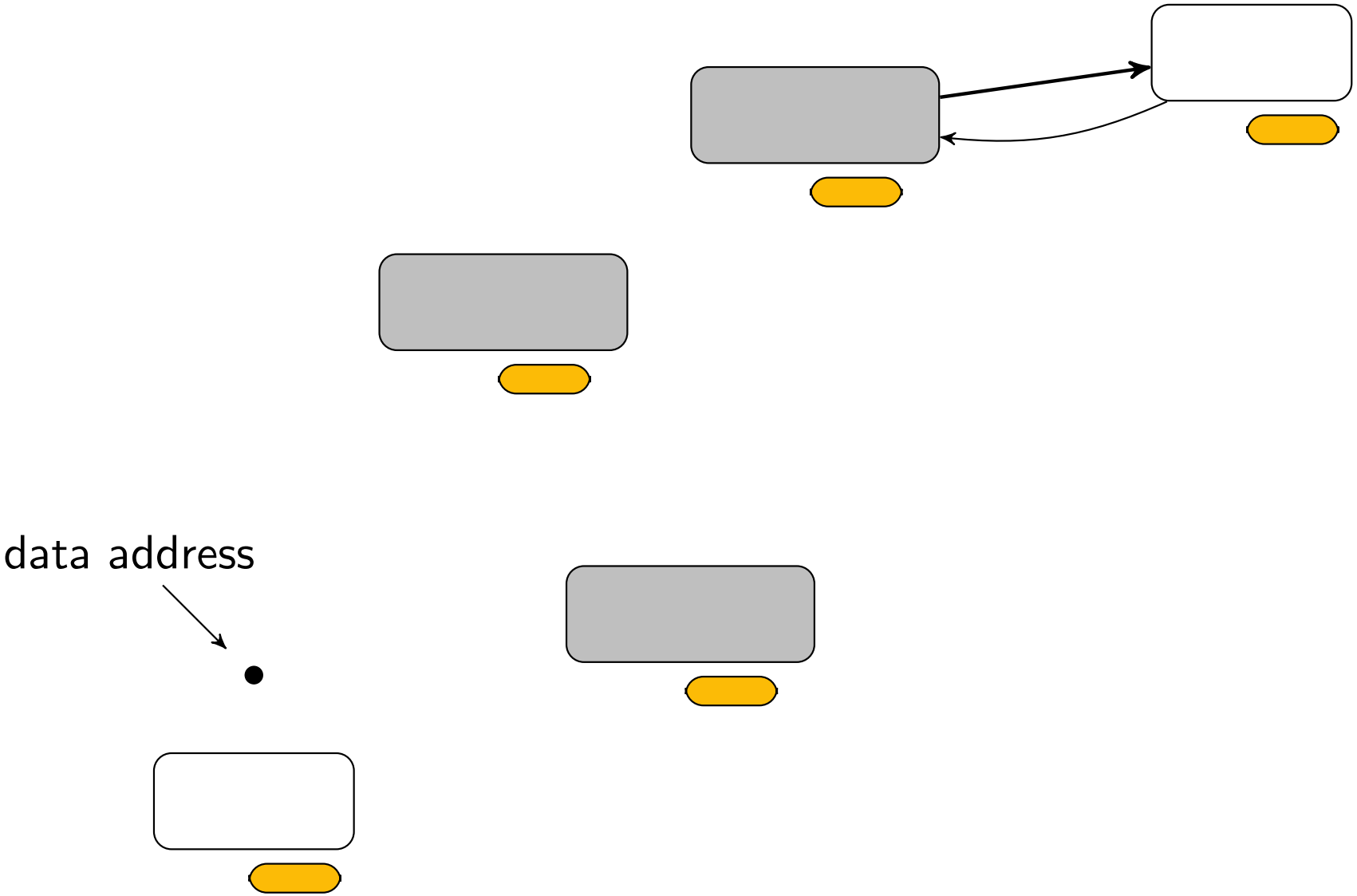
swarm CDN is auto-scaling



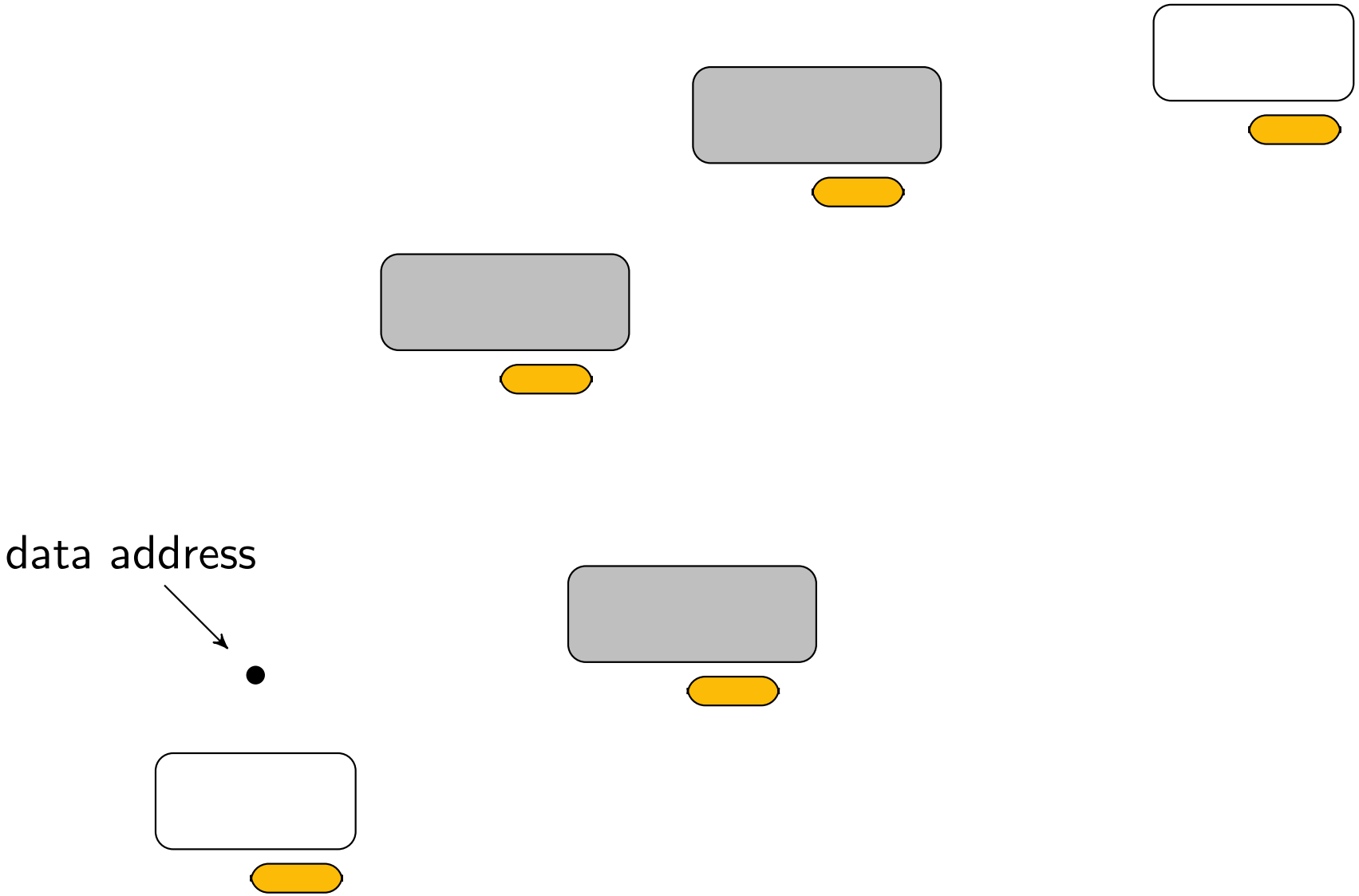
swarm CDN is auto-scaling



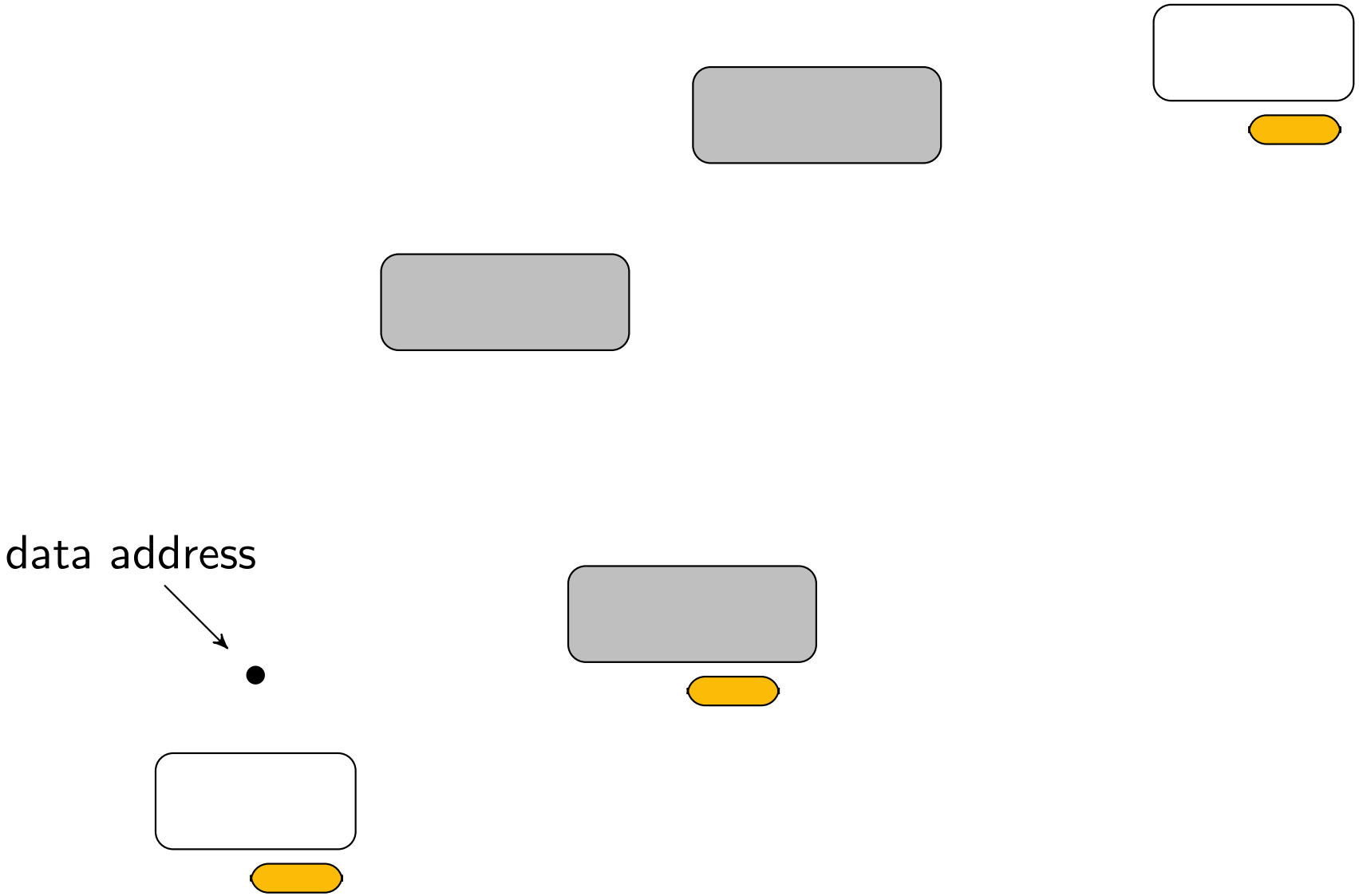
swarm CDN is auto-scaling



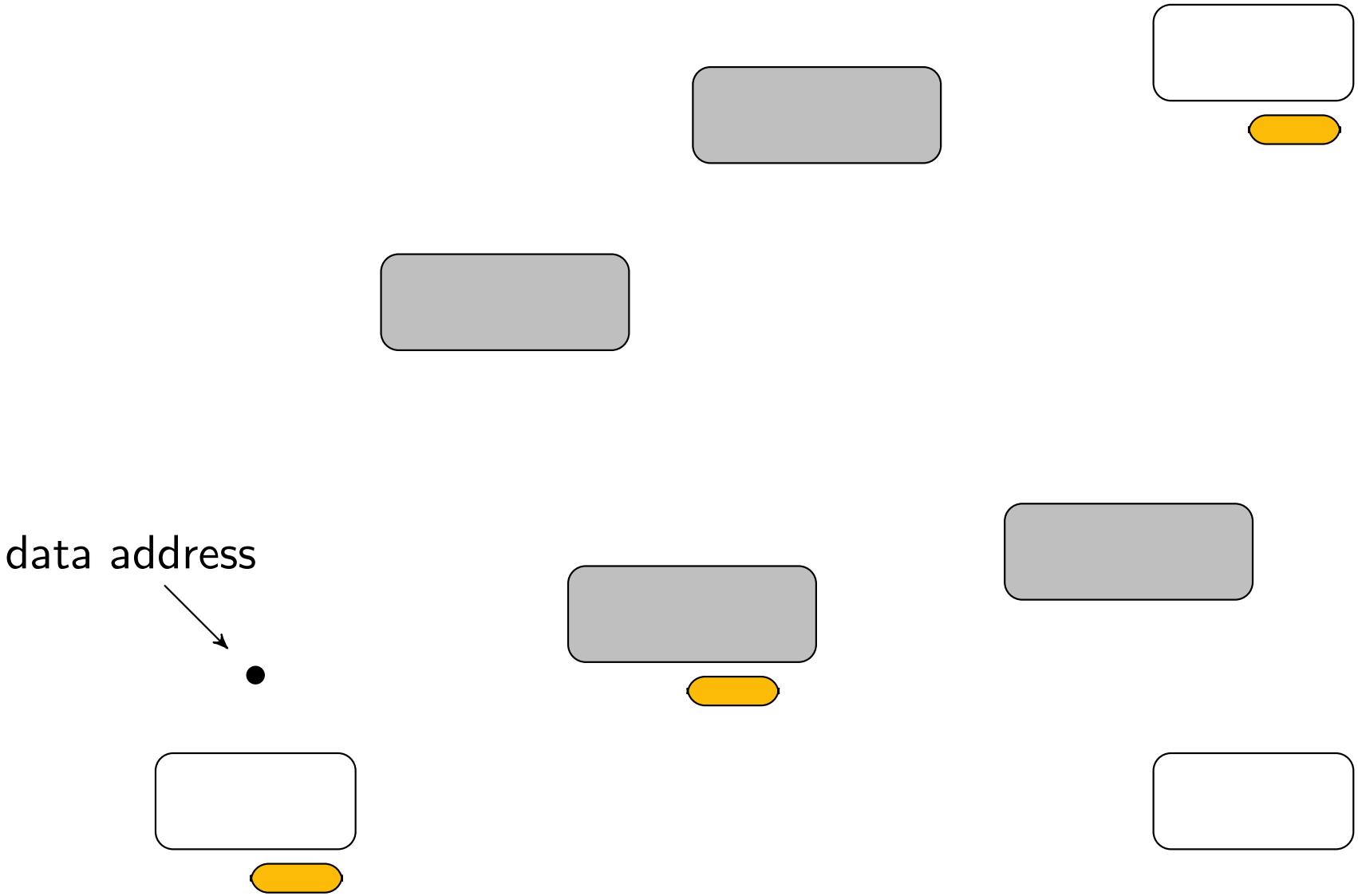
swarm CDN is auto-scaling



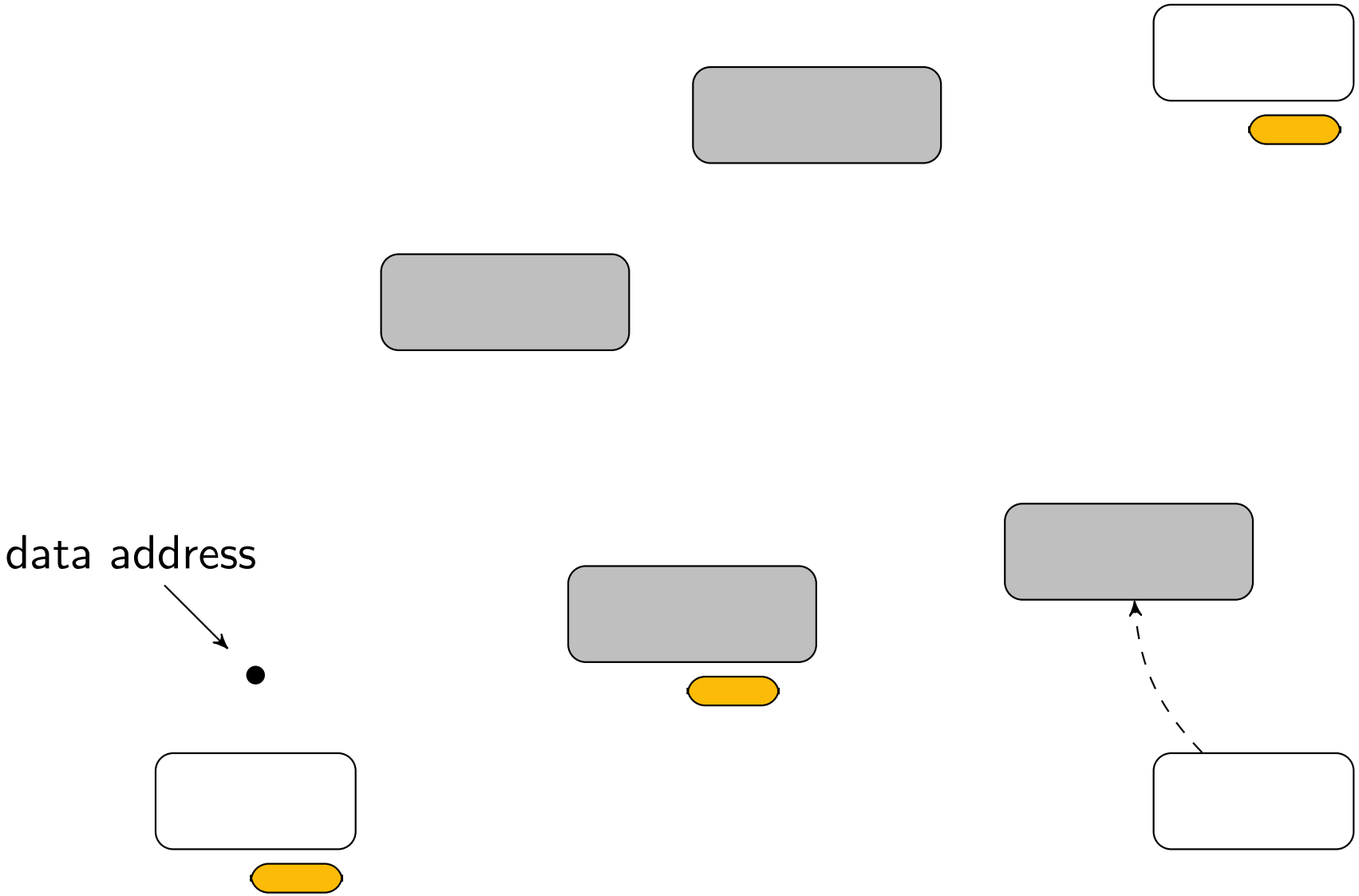
swarm CDN is auto-scaling



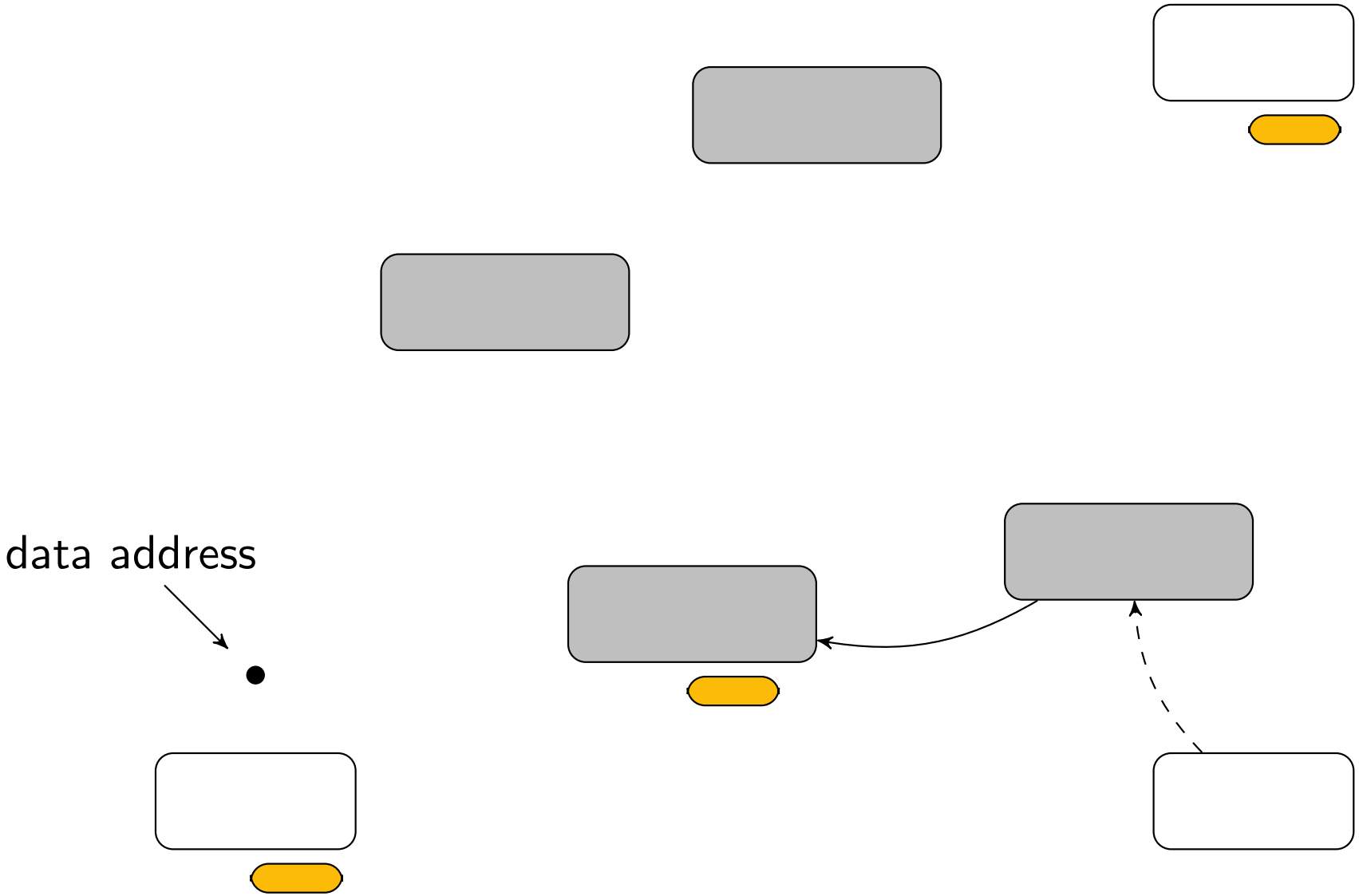
swarm CDN is auto-scaling



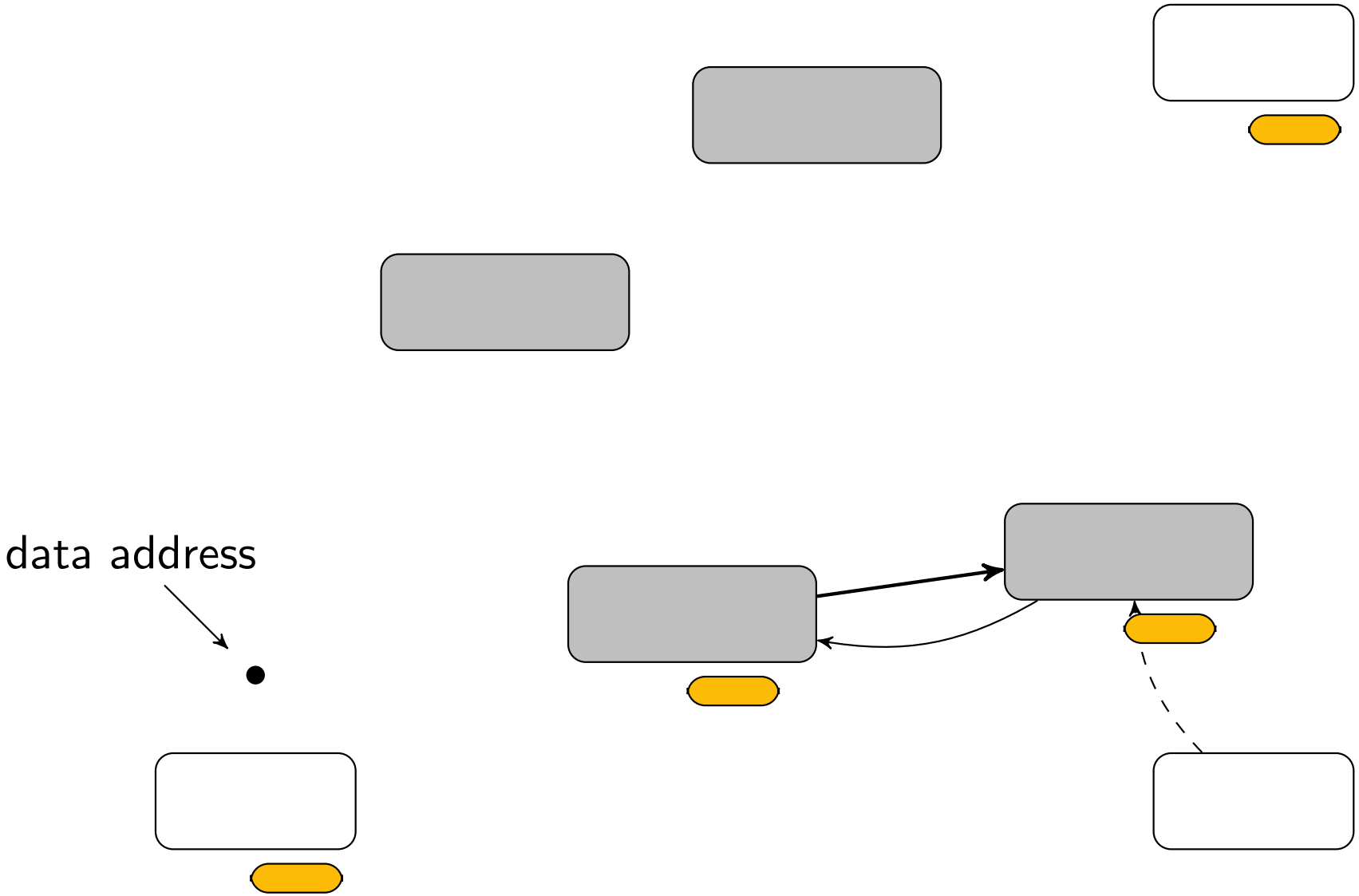
swarm CDN is auto-scaling



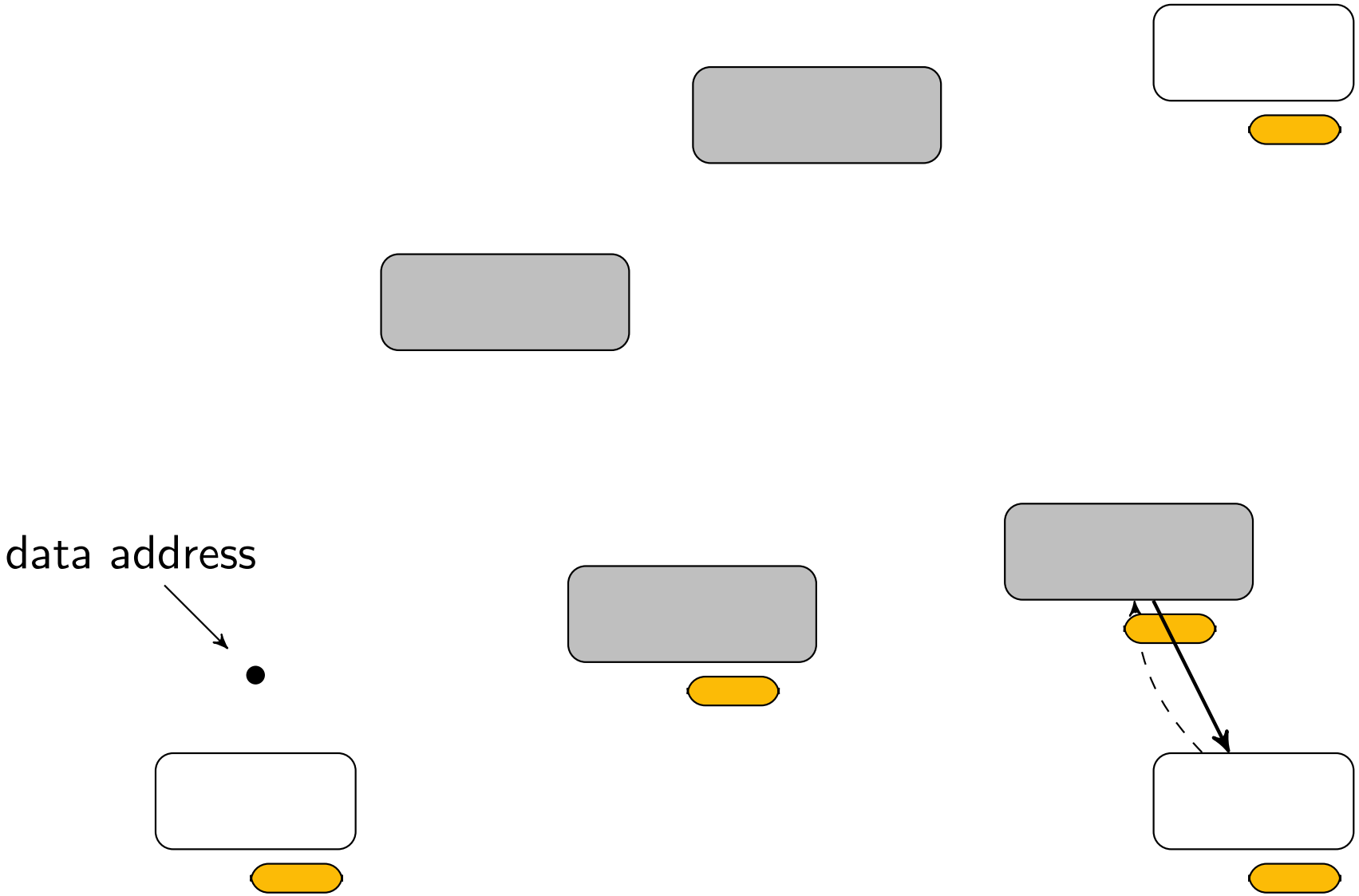
swarm CDN is auto-scaling



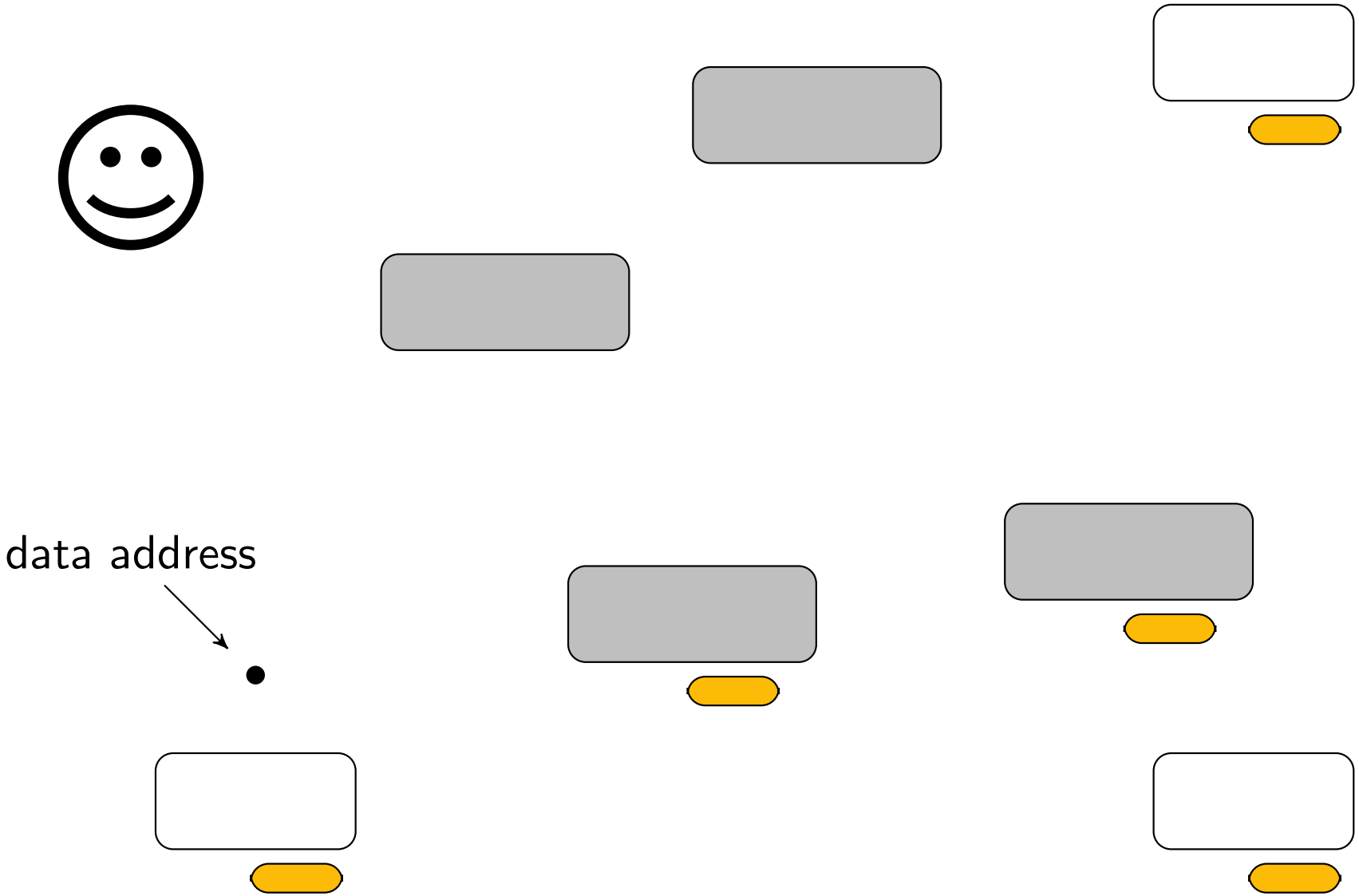
swarm CDN is auto-scaling



swarm CDN is auto-scaling



swarm CDN is auto-scaling



Outline

SWAP allows for speedy retrieval of *popular content*, but there is **no guarantee that less popular content will remain available**. Whatever is not accessed for a long time is likely to be deleted.

The first step: 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 valid 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.

proof of custody + payment channel

These deferred payments constitute a **conditional escrow**: payment is made up-front, payment is held (escrow) and is only released when a valid proof-of-custody is received (condition).

This procedure can be handled off-chain and can be directly **integrated into the payment channels.** All you need is a payment-channel *judge contract* that can understand swarm storage receipts.

If data goes missing...

If data goes missing nodes will lose potential revenue for no longer being able to generate proofs-of-custody, but there are *no further consequences* (yet).

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 promissory notes guaranteeing long-term data availablilty – essentially insurance against deleting.

Implementation: swarm syncing process with added receipts.

insured upload to swarm

syncing

insured upload to swarm

syncing

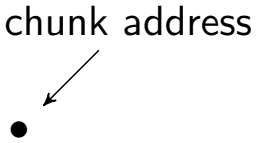
owner

insured upload to swarm

syncing

- chunks to be stored at the nodes whose address is closest to the chunk ID

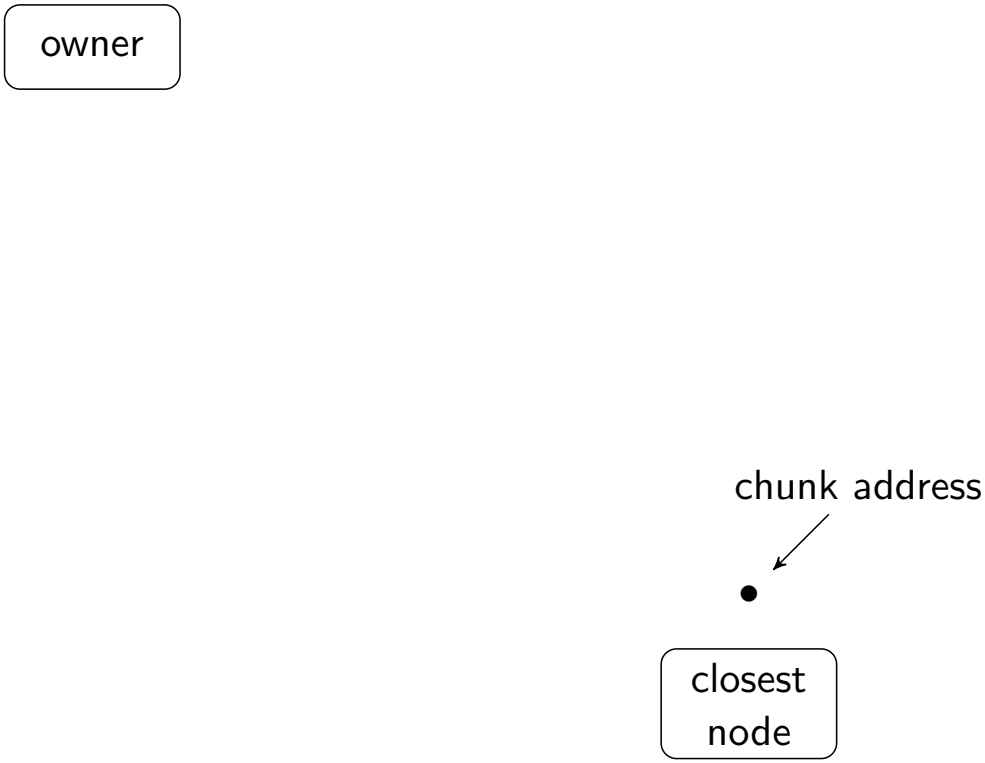
owner



insured upload to swarm

syncing

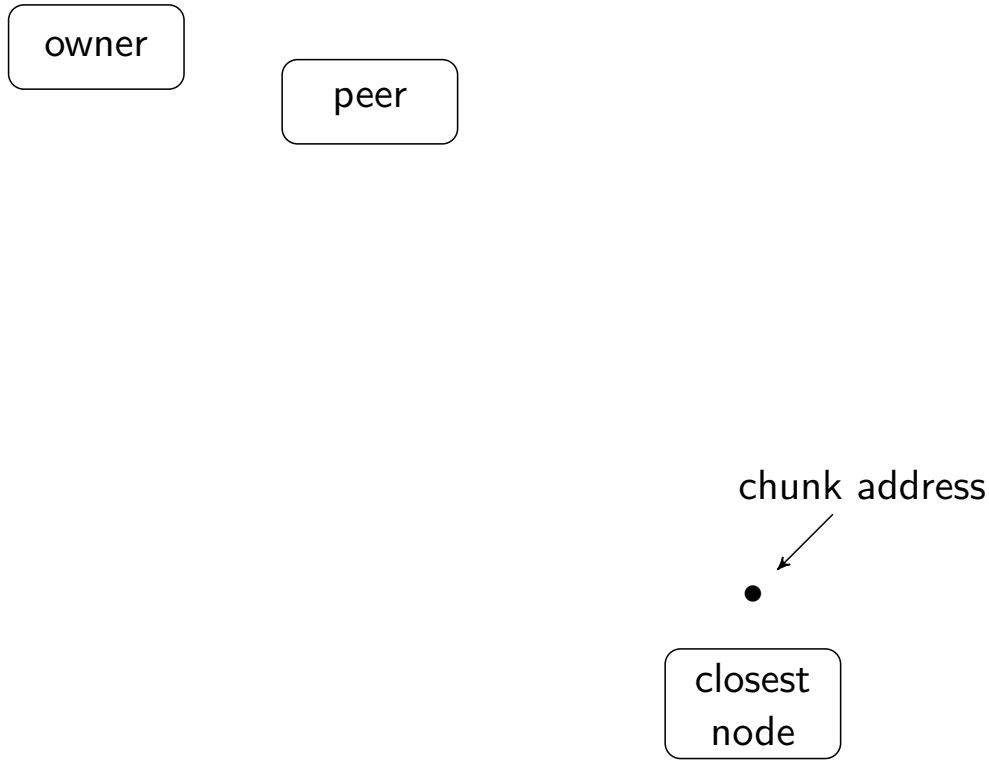
- chunks to be stored at the nodes whose address is closest to the chunk ID



insured upload to swarm

syncing

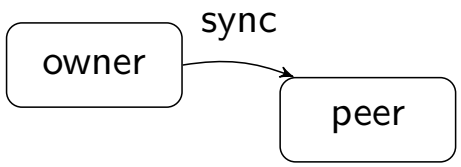
- chunks to be stored at the nodes whose address is closest to the chunk ID
- relaying: syncing

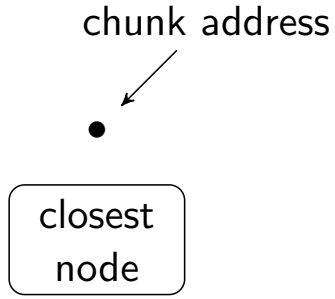


insured upload to swarm

syncing

- chunks to be stored at the nodes whose address is closest to the chunk ID
- relaying: syncing

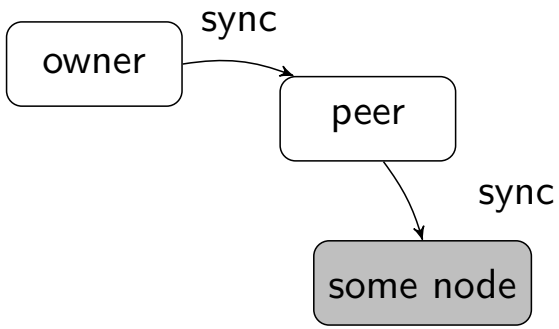


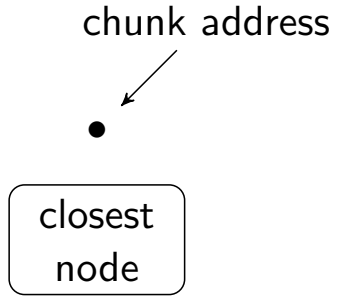


insured upload to swarm

syncing

- chunks to be stored at the nodes whose address is closest to the chunk ID
- relaying: syncing
- data is passed on from node to node

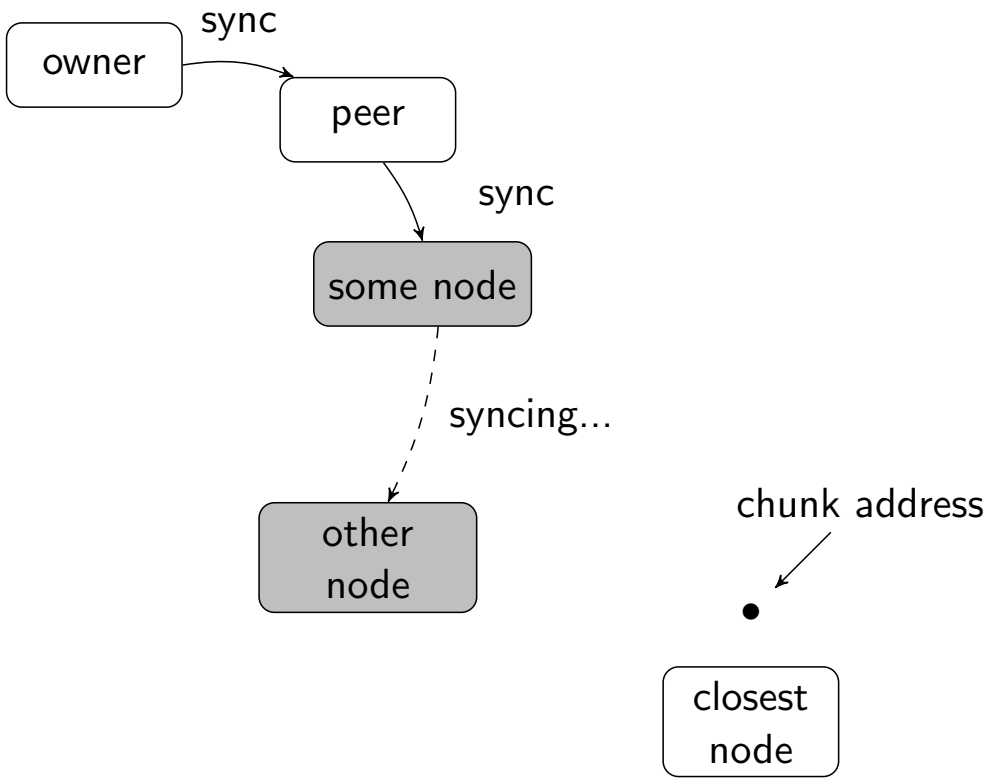




insured upload to swarm

syncing

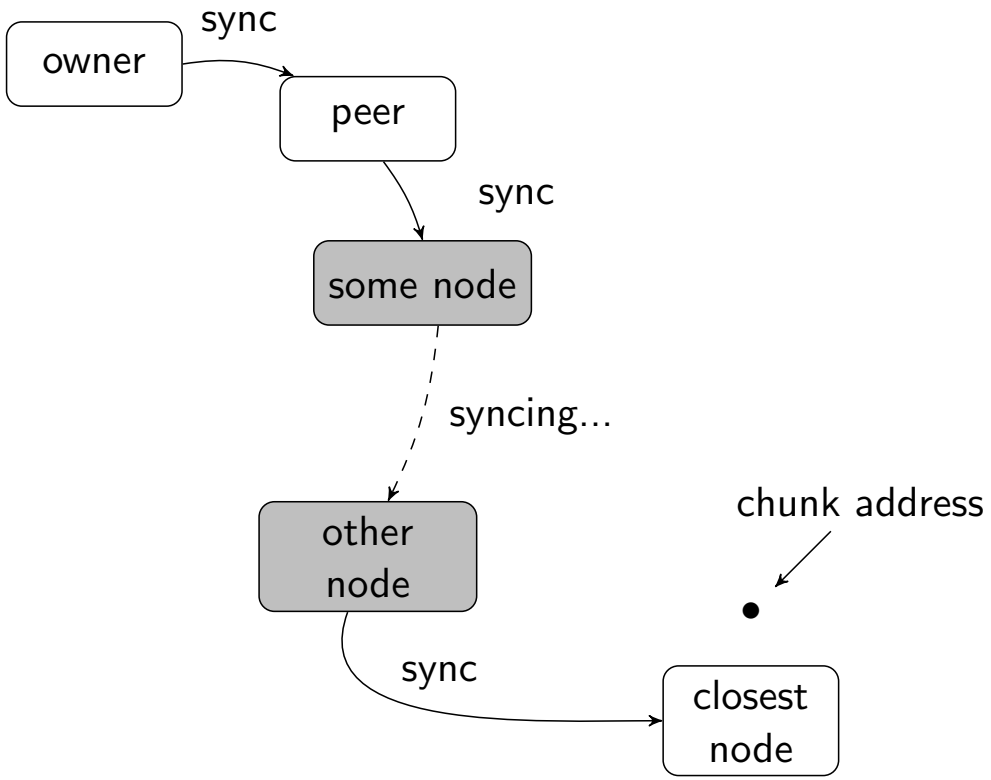
- chunks to be stored at the nodes whose address is closest to the chunk ID
- relaying: syncing
- data is passed on from node to node



insured upload to swarm

syncing

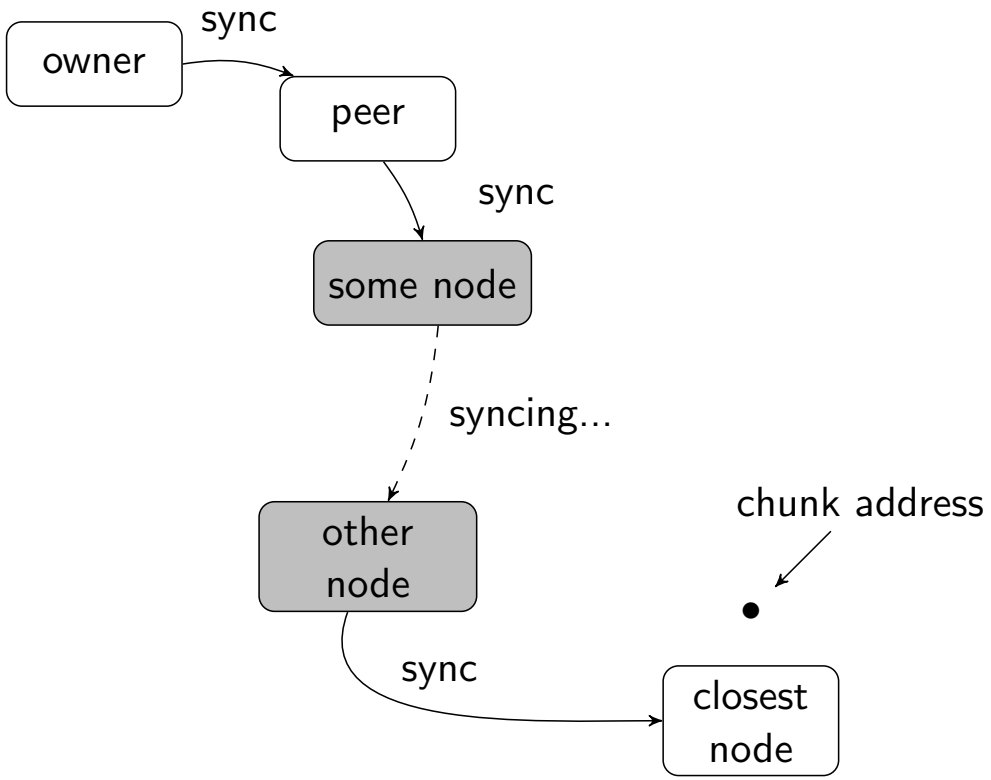
- chunks to be stored at the nodes whose address is closest to the chunk ID
- relaying: syncing
- data is passed on from node to node



insured upload to swarm

syncing

- chunks to be stored at the nodes whose address is closest to the chunk ID
- relaying: syncing
- data is passed on from node to node



insured upload to swarm

insured storage

syncing via registered nodes with each swap receipted.

insured storage:

- owner passes data to a registered peer and receives an insurance receipt
- relaying: syncing
- all receipts are accounted and paid for

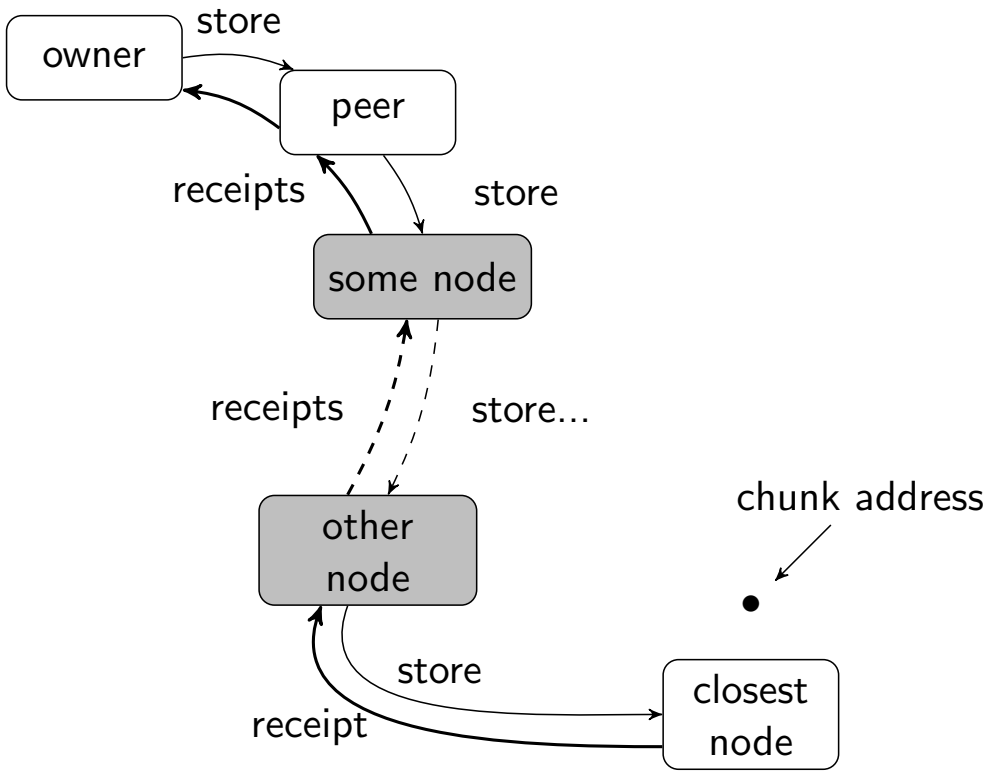
insured upload to swarm

insured storage

syncing via registered nodes with each swap receipted.

insured storage:

- owner passes data to a registered peer and receives an insurance receipt
- relaying: syncing
- all receipts are accounted and paid for



SWINDLE: **s**torage **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 not found

litigation by challenge

defense by providing

- proof-of-custody of the data (eventually the data itself)
- a storage receipt for the data, shifting the blame and implicating another node as the culprit.

upload and disappear

- swear to sync and receipting → immediate settlement with the peer at upload
- finger-pointing along chain of receipts → correct accountability of storer thereafter

SWAP • SWEAR • SWINDLE

ethersphere orange paper series

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

Viktor Trón, Aron Fischer, Nick Johnson: smash-proof: auditable storage for swarm secured by masked audit secret hash. May 2016

swarm: status and usage

what is the development 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 over 3 regions
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

join us

contact and contribute

swarm channel: `gitter.im/ethereum/swarm`

swarm info page & orange papers: `swarm-gateways.net`

swarm gateway: `swarm-gateways.net web3.download`

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