Causality for the Cloudlets:

Offering Causality on the Edge With Small Metadata

Nuno Afonso, Manuel Bravo, Luís Rodrigues

inesc id lisboa

TÉCNICO LISBOA

Processing on the edge

There are many mobile applications that require the execution of resource demanding tasks.

- Face recognition
- Video-indexing
- Augmented reality

Processing on the edge

There are many mobile applications that require the execution of resource demanding tasks.

- Face recognition
- Video-indexing
- Augmented reality

These tasks need to be processed in the cloud.

Processing on the edge

There are many mobile applications that require the execution of resource demanding tasks.

- Face recognition
- Video-indexing
- Augmented reality

Latency constraints: 5-30 ms !!

Processing on the edge

There are many mobile applications that require the execution of resource demanding tasks.

- Face recognition
- Video-indexing
- Augmented reality

Small clouds near the edge.

Edge clouds

- Mobile edge computing
- Fog computing
- Cloudlets

Edge clouds

- Mobile edge computing
- Fog computing
- Cloudlets

Rough estimate

To ensure latency requirements,
more than 100 cloudlets should be needed
in Europe alone!

Causality on the edge

- Datacenters + cloudlets: high number of nodes
- Partial replication

Causality on the edge

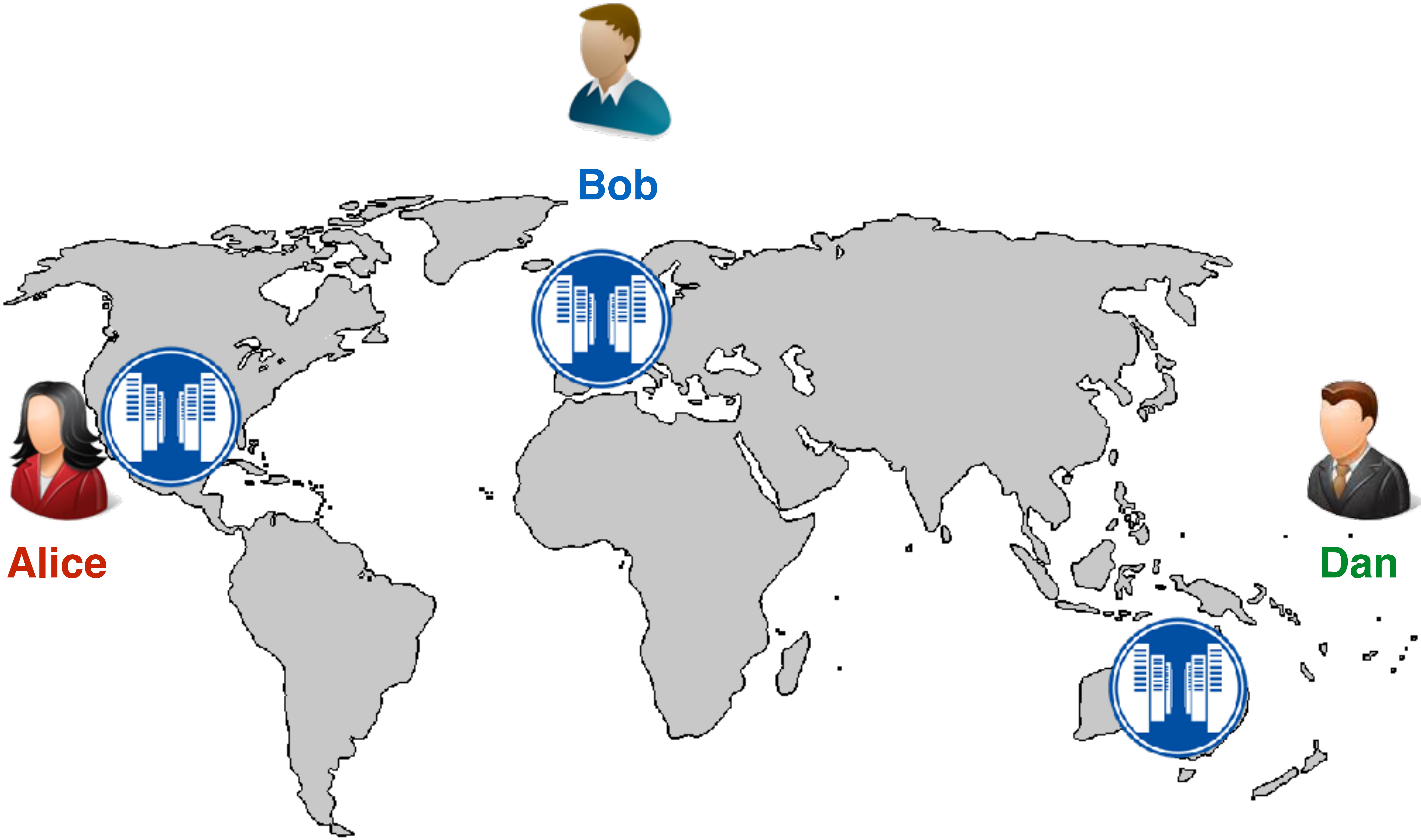
- Datacenters + cloudlets: high number of nodes
- Partial replication

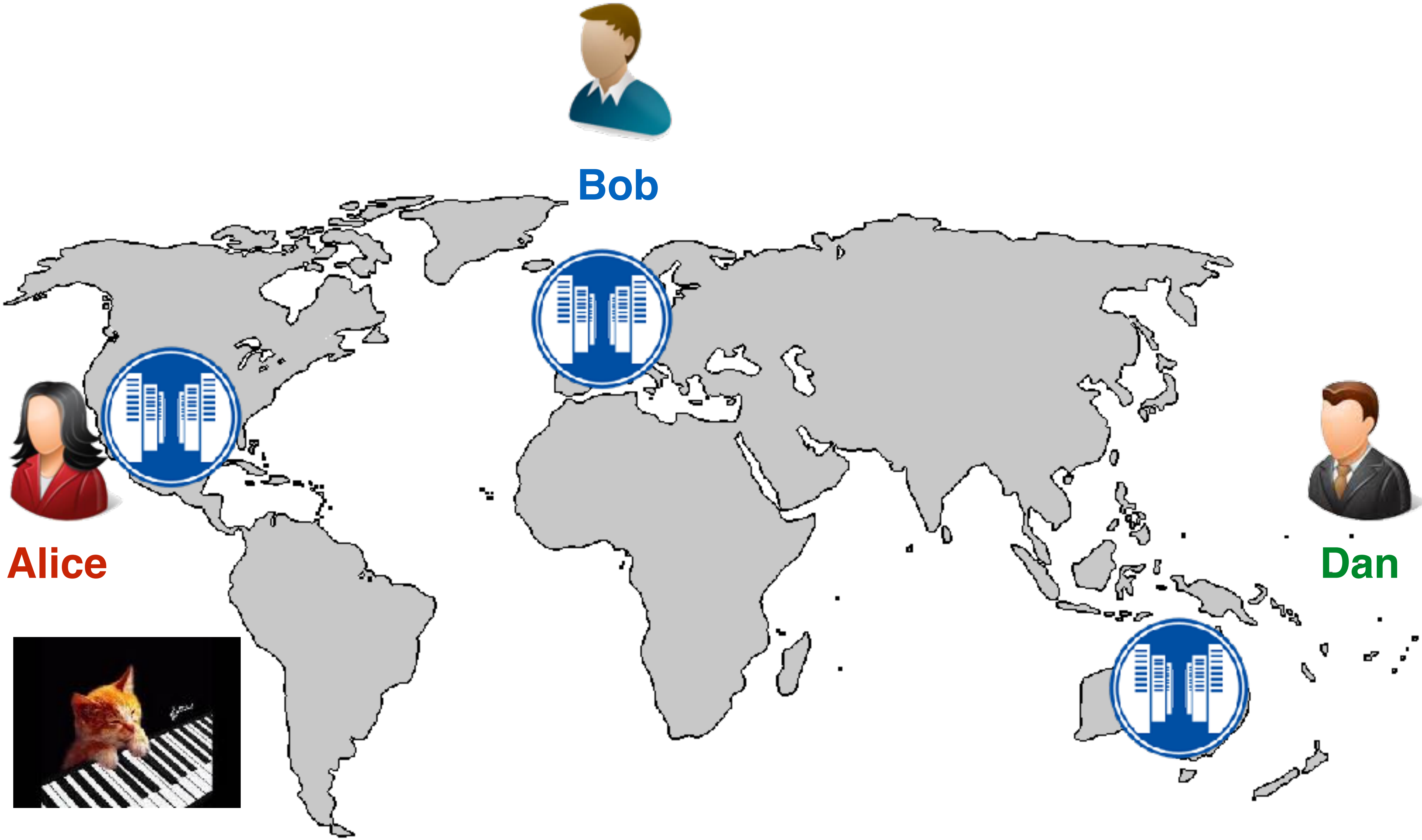
Traditional techniques to enforce causality, such as vector clocks, will not scale

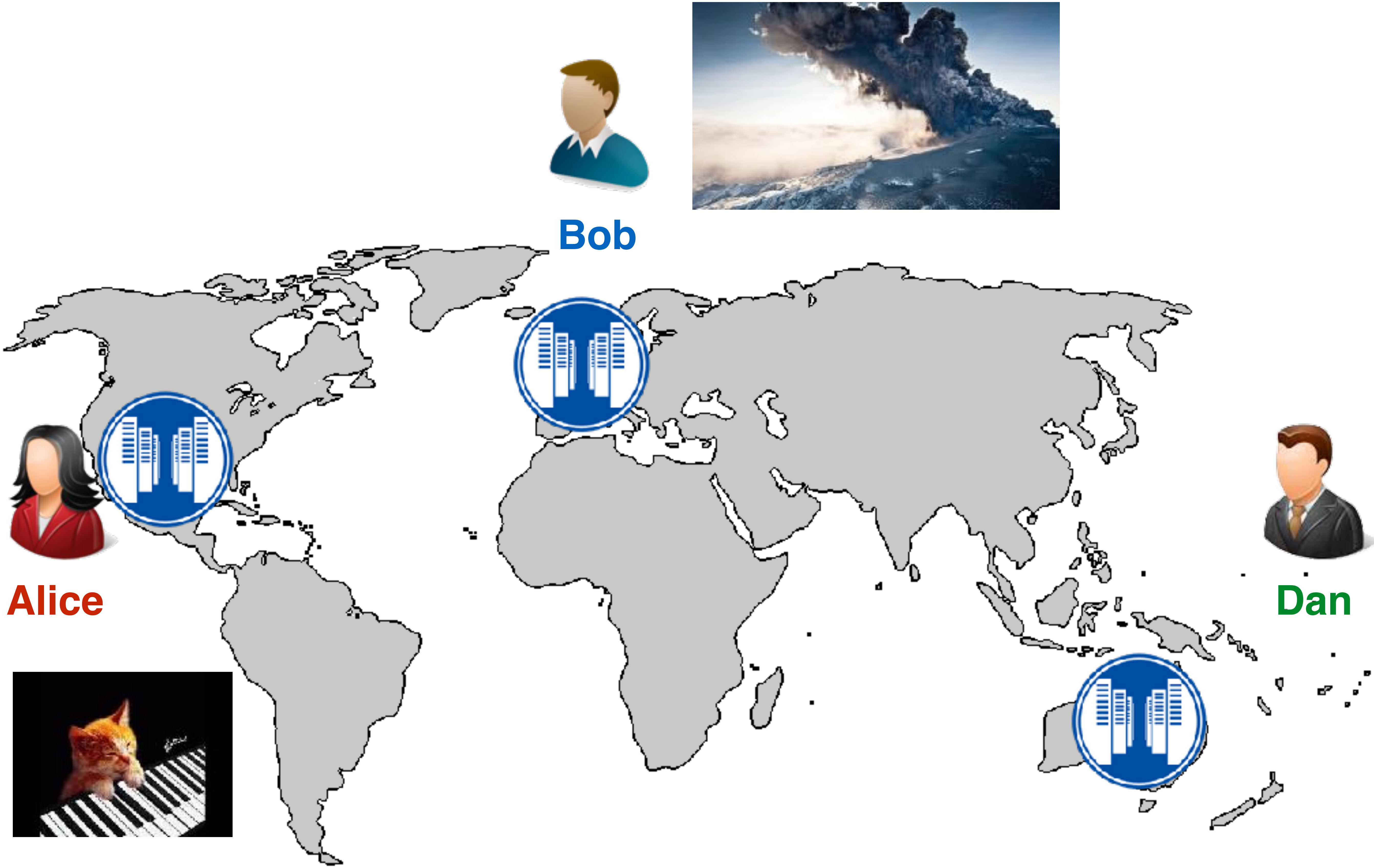
Causality on the edge

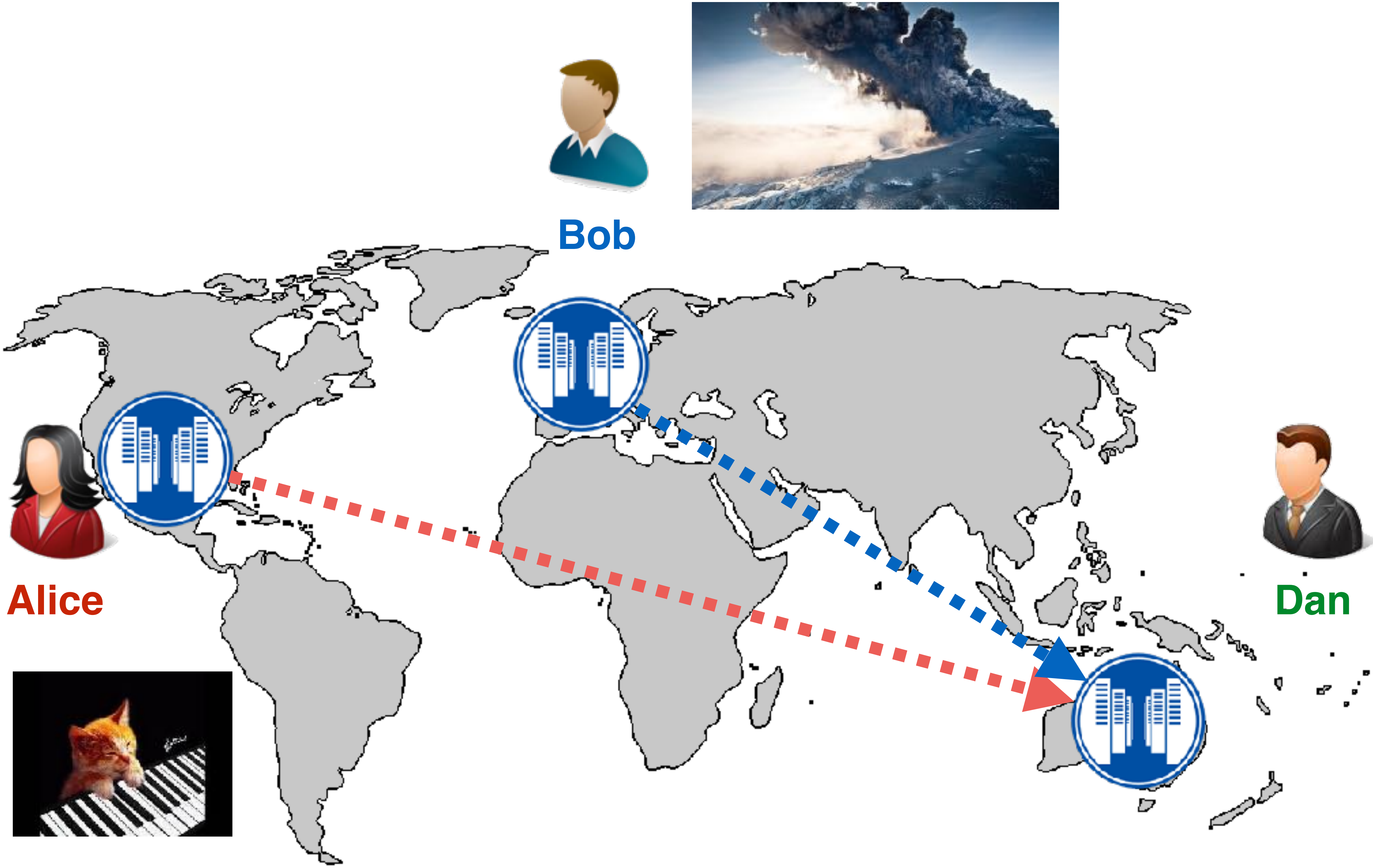
- Datacenters + cloudlets: high number of nodes
- Partial replication

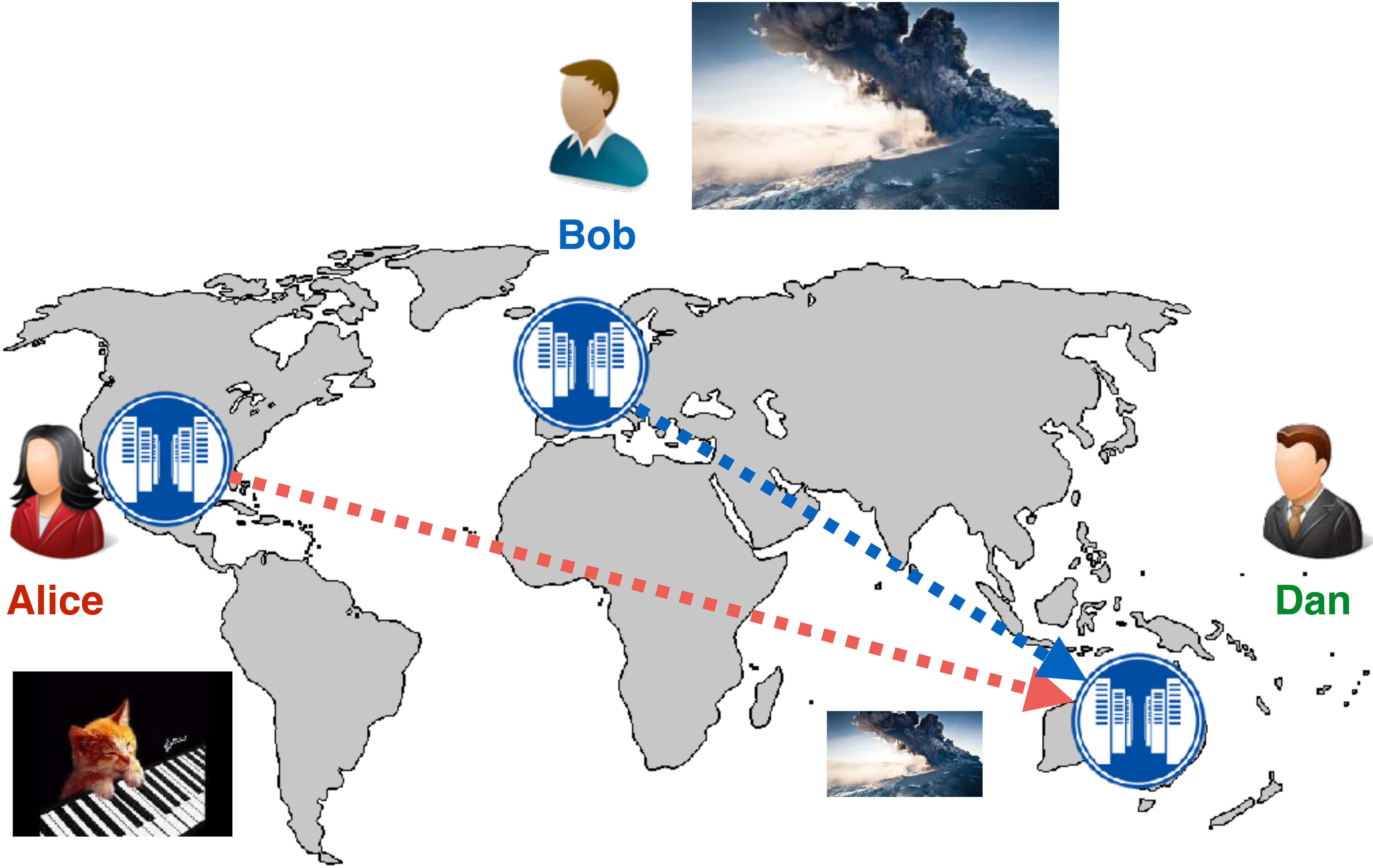
Naive techniques that use small metadata may generate false dependencies

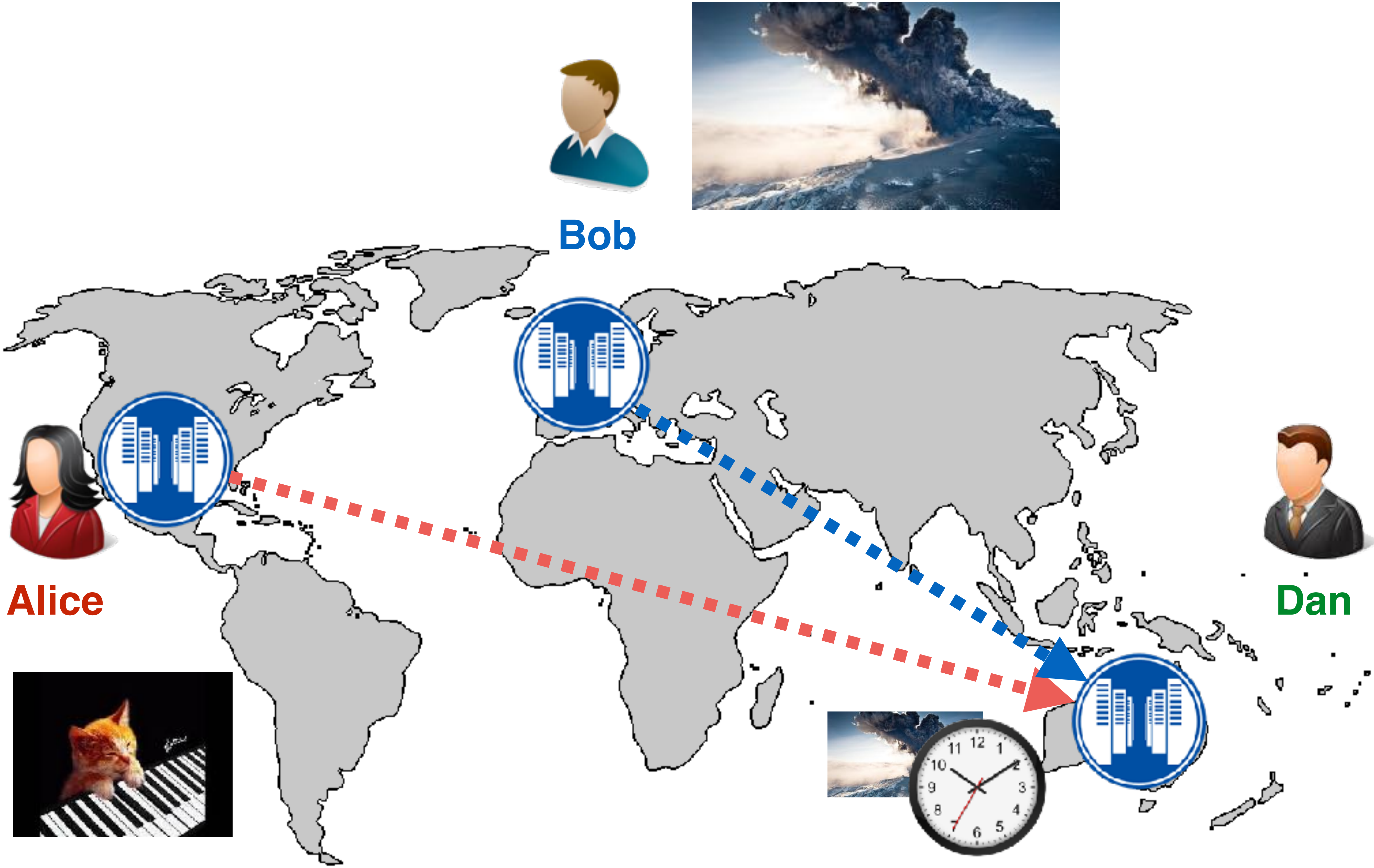

Bob
Alice
Dan

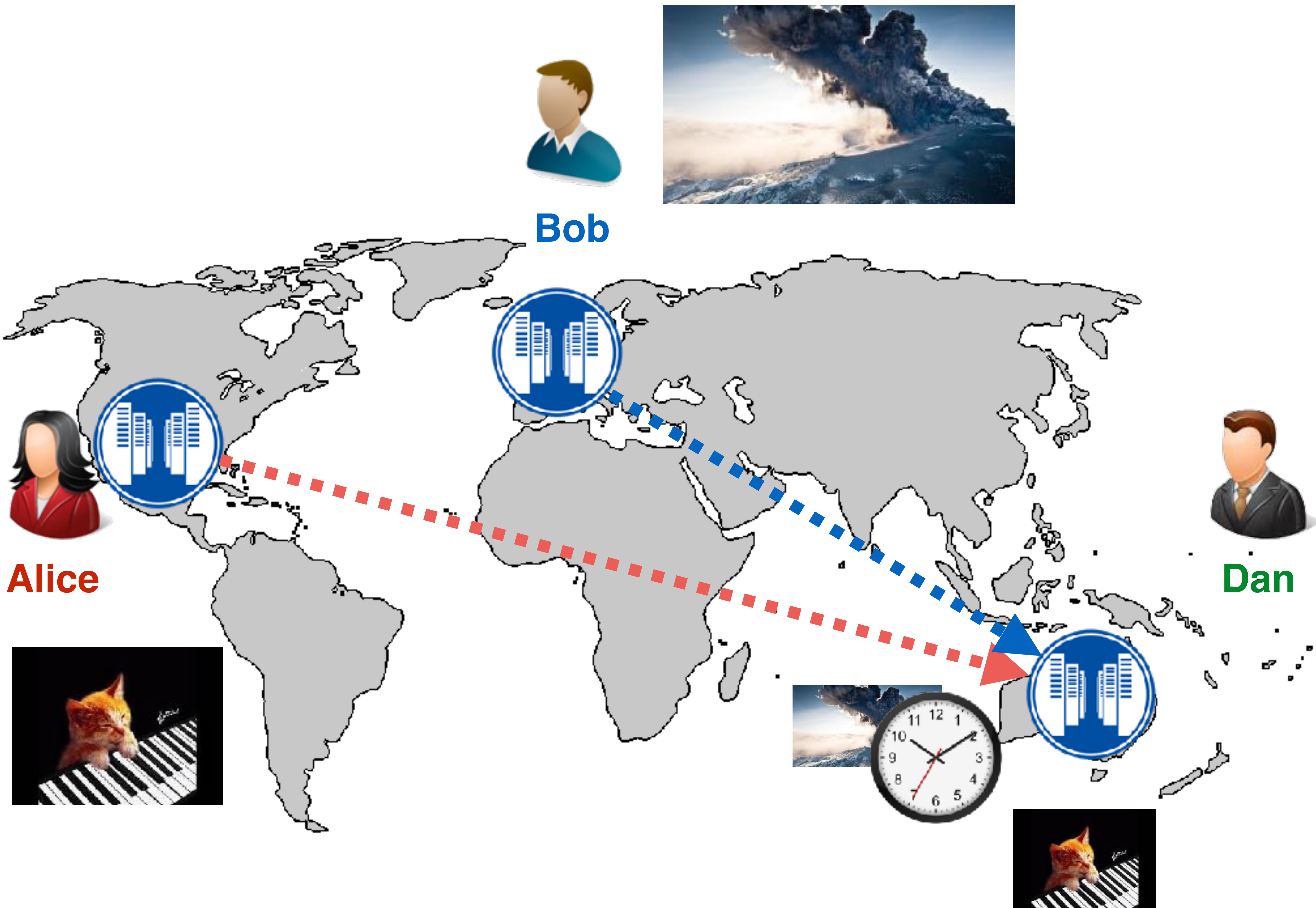

Bob
Alice
Dan

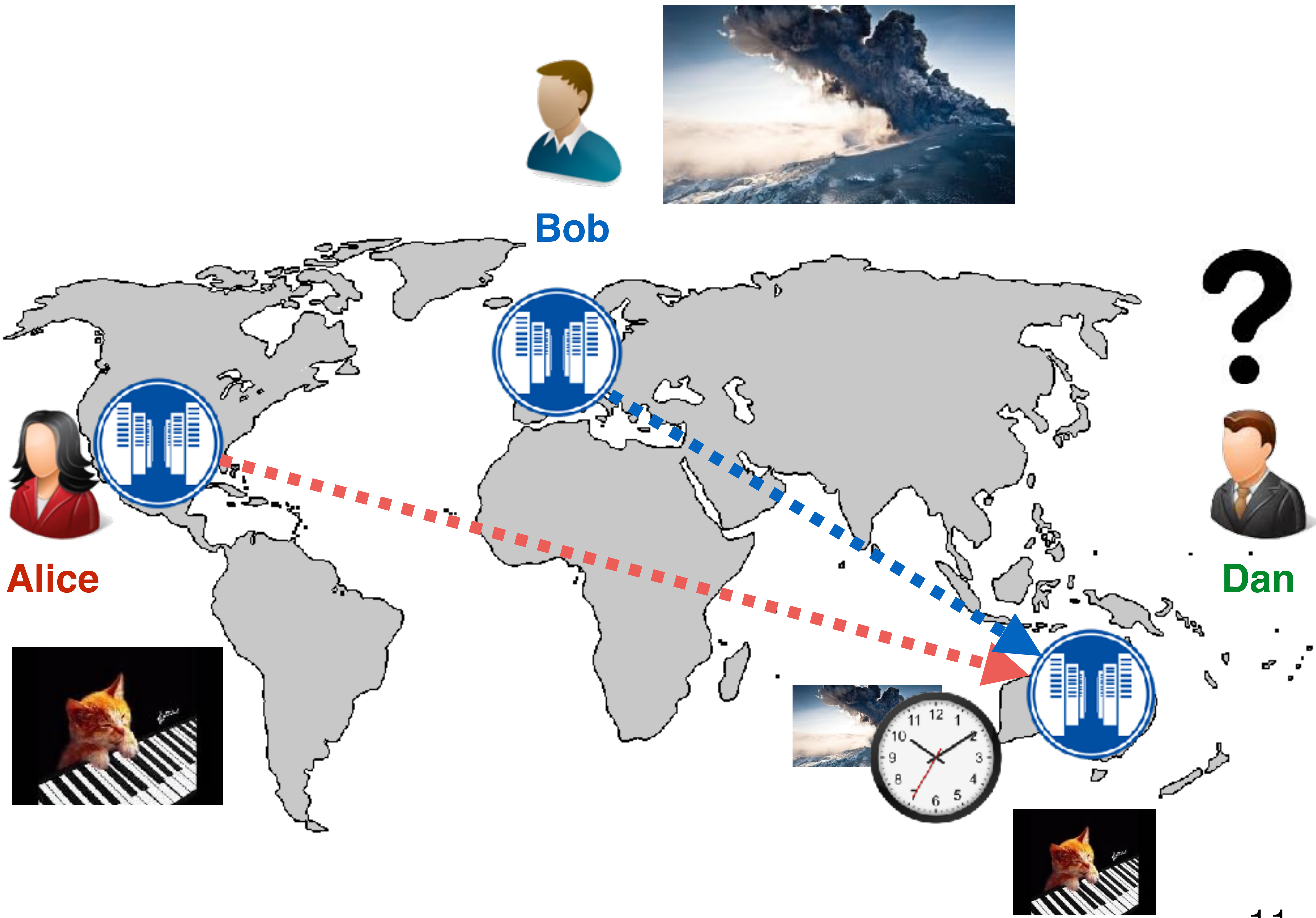

Bob
Alice
Dan

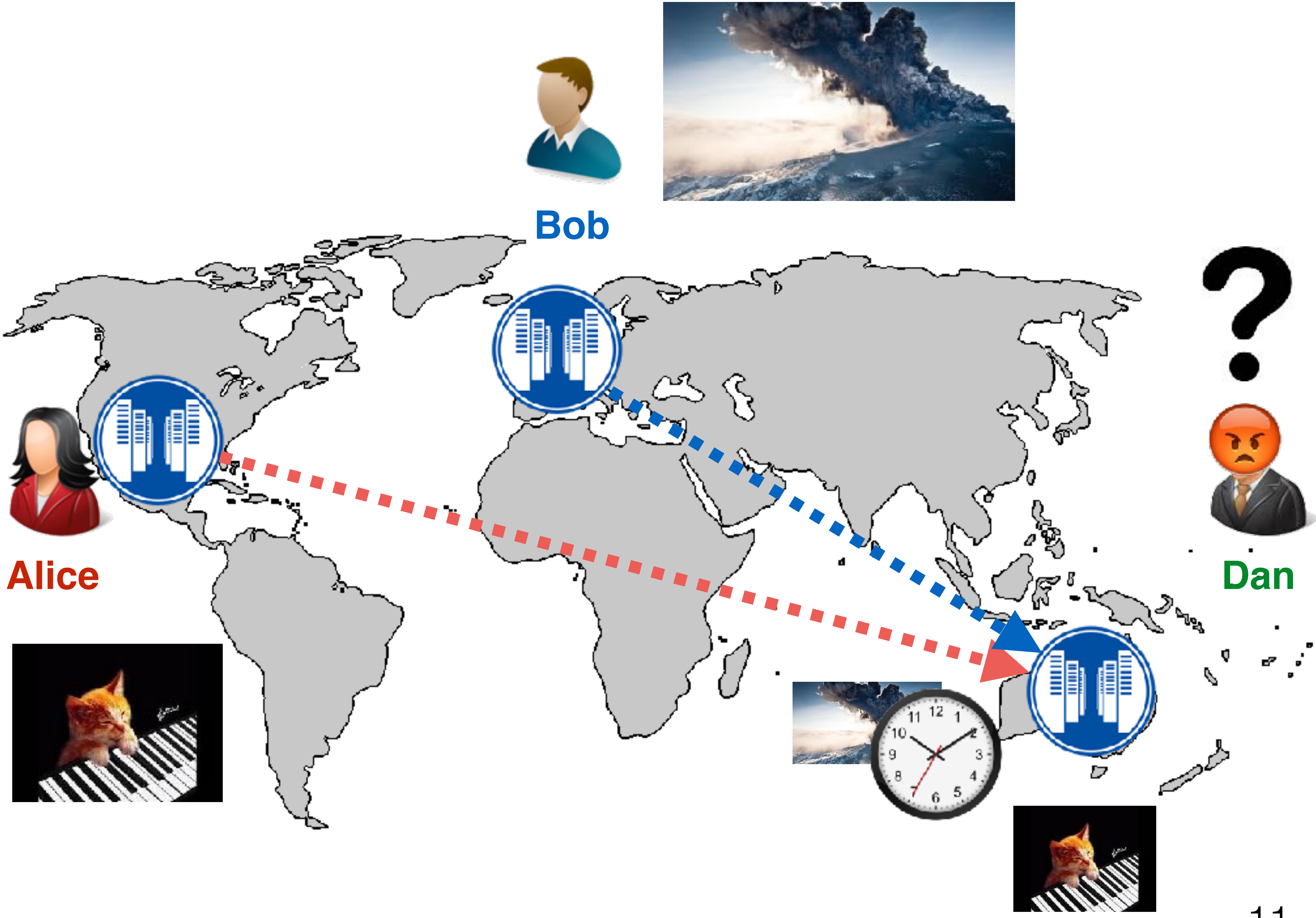

Bob
Alice
Dan


Bob
Alice
Dan


Bob
Alice
Dan


Bob
Alice
Dan


Bob
Alice
Dan
?


Bob
Alice
Dan

Our approach

- To leverage our previous work on Saturn
- Extend Saturn to operate on the edge

Saturn

God in ancient Roman religion,
that become the god of time

EUROSYS 2017

Distributed metadata service

- pluggable to existing geo-distributed data services
- handles the dissemination of metadata among data centers

Ensures that

- clients always observe a causally consistent state
- with a negligible performance overhead when compared to an eventually consistency system

Metadata

more metadata — less metadata

Metadata

Matrix/vector clocks

Metadata

Matrix/vector clocks

more metadata

less metadata

One vector per item.
One entry in each vector per DC.

Metadata

Matrix/vector clocks

more metadata　　　　less metadata

precise

expensive

Metadata

Matrix/vector clocks — **Lamport's clocks**

more metadata — less metadata

precise

expensive

Metadata

Matrix/vector clocks

Lamport's clocks

more metadata

less metadata

precise

expensive

One scalar.

Metadata

Matrix/vector clocks | **Lamport's clocks**

more metadata | less metadata

precise | **false positives**

expensive | **cheap**

Problems of the previous state-of-the-art

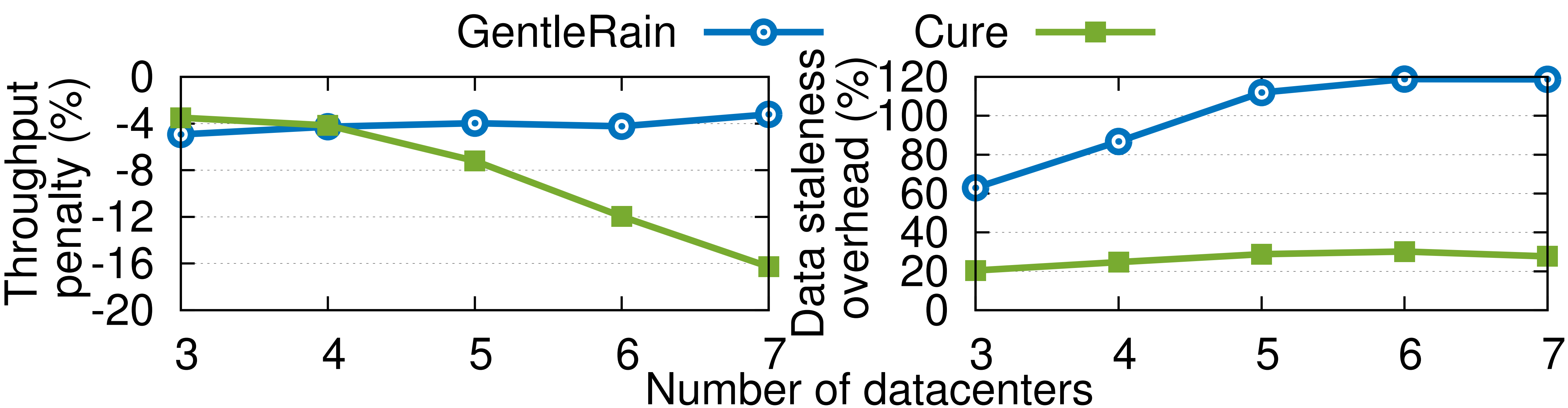
Throughput vs. data staleness tradeoff

GentleRain [SoCC' 14]: Optimizes throughput

Compresses metadata into a scalar

Cure [ICDCS' 16]: Optimizes data freshness

Relies on a vector clock with an entry per data center



Problems of the previous state-of-the-art

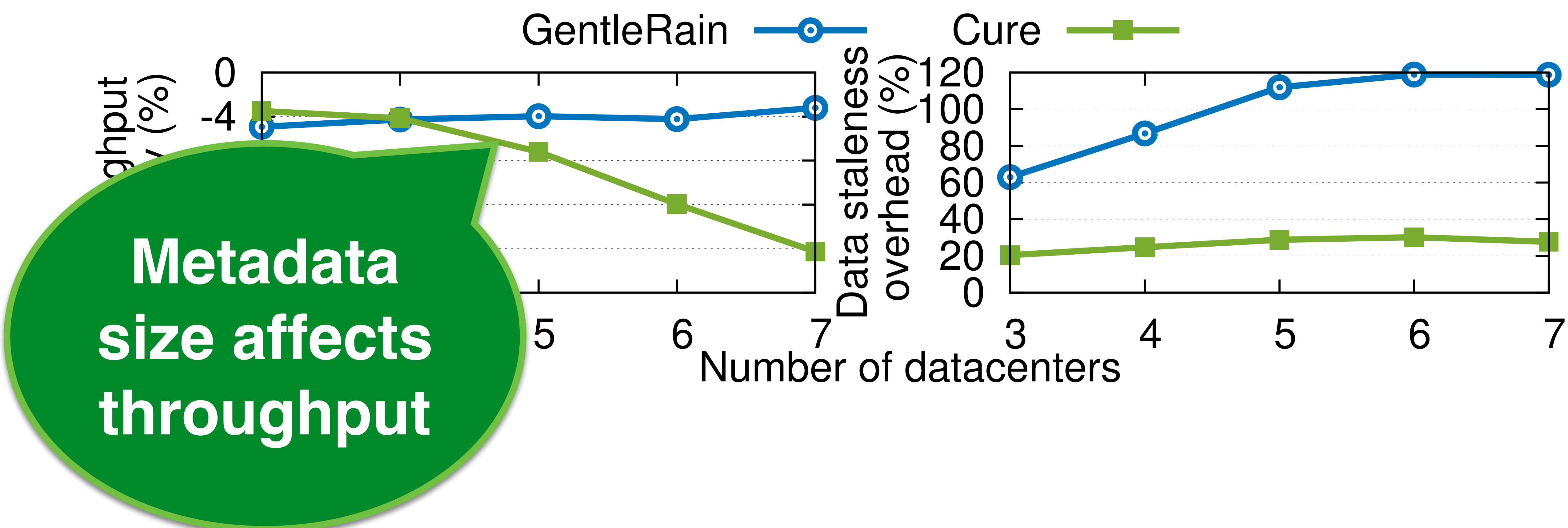
Throughput vs. data staleness tradeoff

GentleRain [SoCC' 14]: Optimizes throughput

Compresses metadata into a scalar

Cure [ICDCS' 16]: Optimizes data freshness

Relies on a vector clock with an entry per data center



Problems of the previous state-of-the-

Throughput vs. data staleness tradeoff

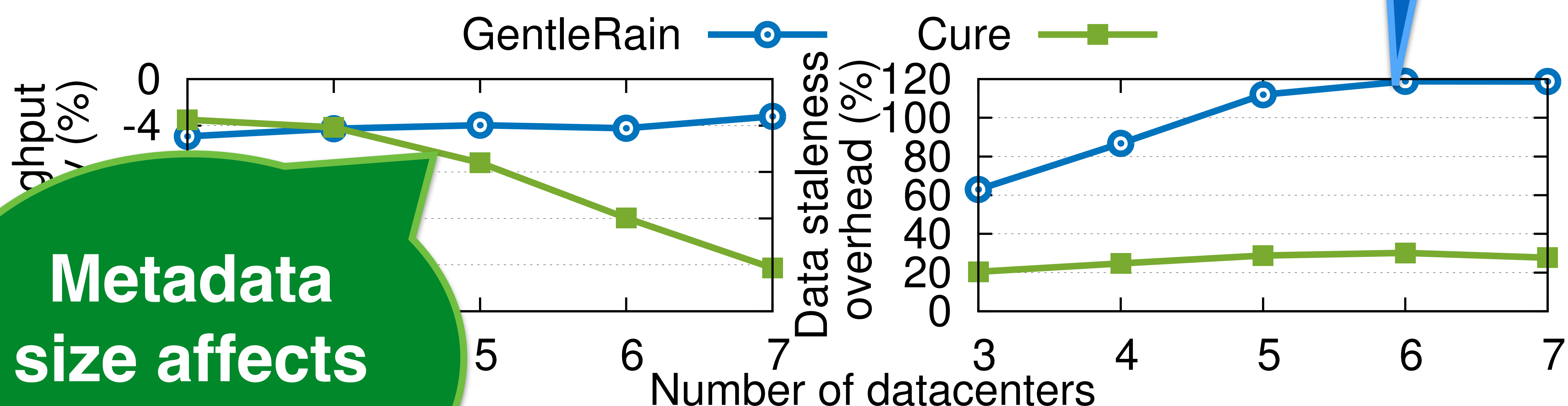
False dependencies damage data freshness

GentleRain [SoCC' 14]: Optimizes throughput

Compresses metadata into a scalar

Cure [ICDCS' 16]: Optimizes data freshness

Relies on a vector clock with an entry per data center



Metadata size affects throughput

key features

key features

Requires a **constant and small** amount of metadata
regardless of the system's scale (servers, partitions, and locations)

key features

Requires a **constant and small** am
regardless of the system's scale (servers

to avoid impairing
throughput

key features

Requires a **constant and small** am

regardless of the system’s scale (server

to avoid impairing
throughput

Mitigates the impact of **false dependencies**

by relying on a tree-based dissemination

key features

Requires a **constant and small** am
regardless of the system's scale (servers

to avoid impairing
throughput

Mitigates the impact of **false depe**
by relying on a tree-based disseminatio

to enhance
data freshness

key features

Requires a **constant and small** am
regardless of the system's scale (servers

to avoid impairing
throughput

Mitigates the impact of **false depe**
by relying on a tree-based dissemination

to enhance
data freshness

Implements **genuine partial replication**
data centers only manage data and metadata of the items replicated locally

key features

Requires a **constant and small** am
regardless of the system's scale (servers

to avoid impairing
throughput

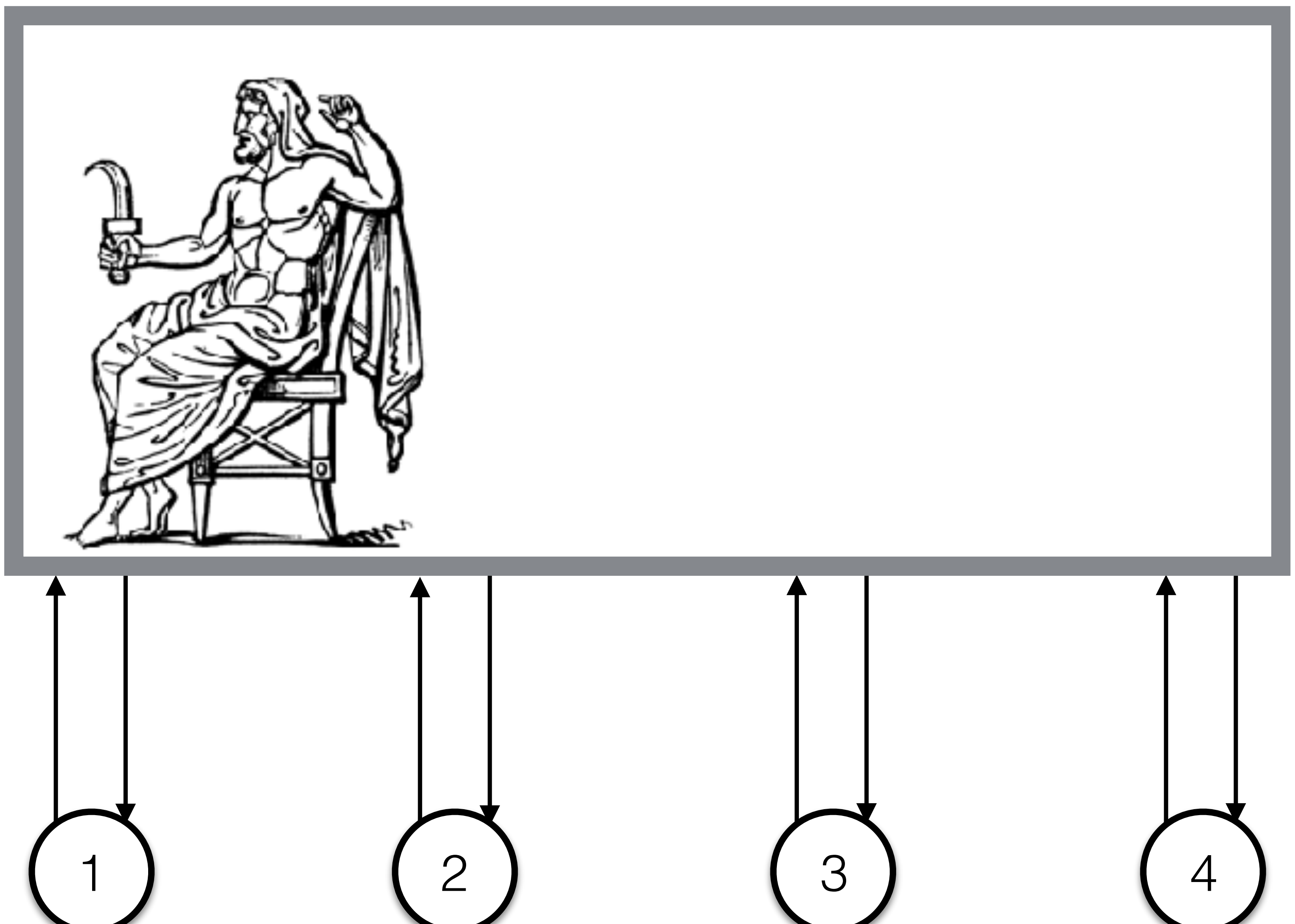
Mitigates the impact of **false depe**
by relying on a tree-based dissemination

to enhance
data freshness

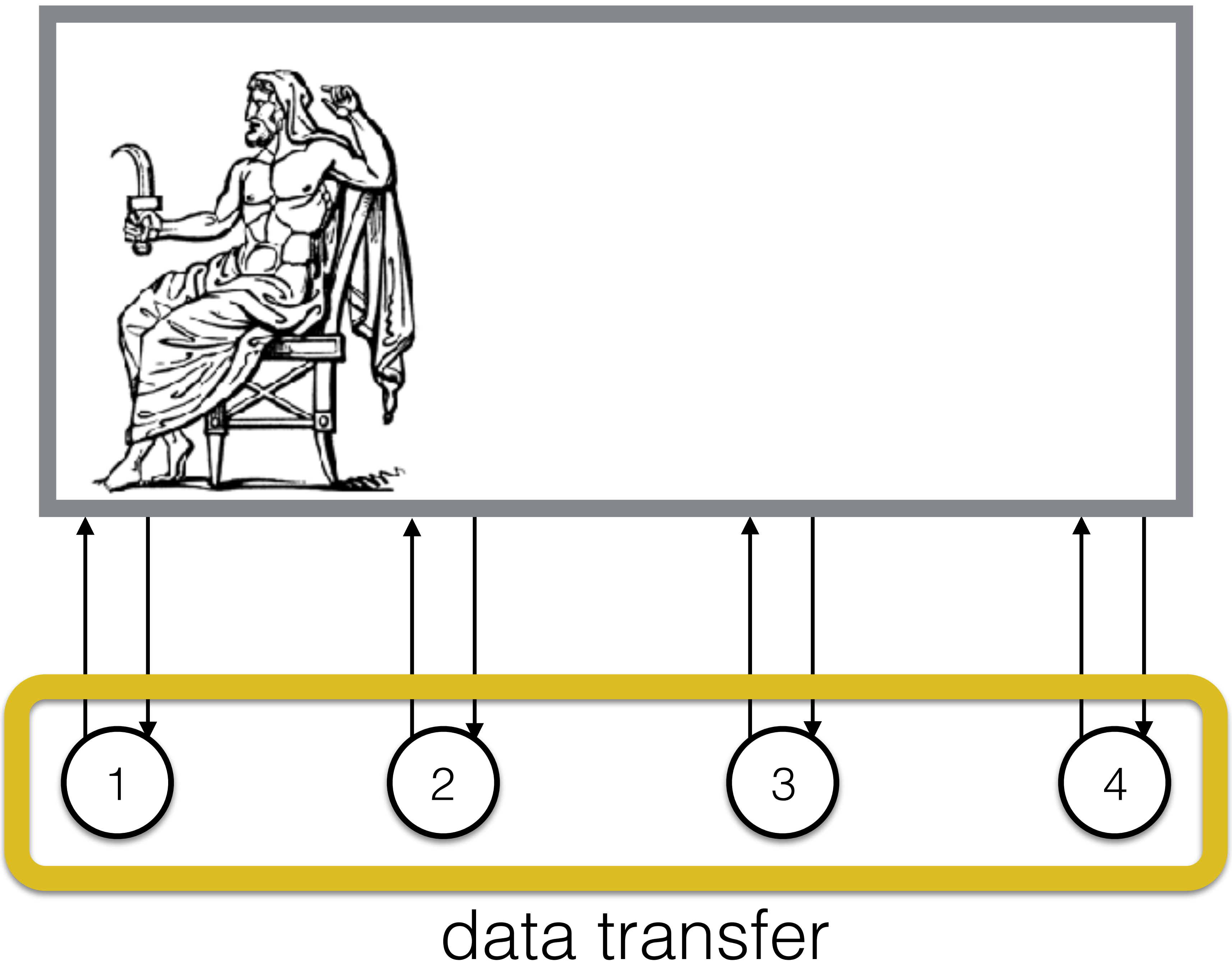
Implements **genuine partial replica**
data centers only manage data and meta
replicated locally

to take full advantage of
partial replication

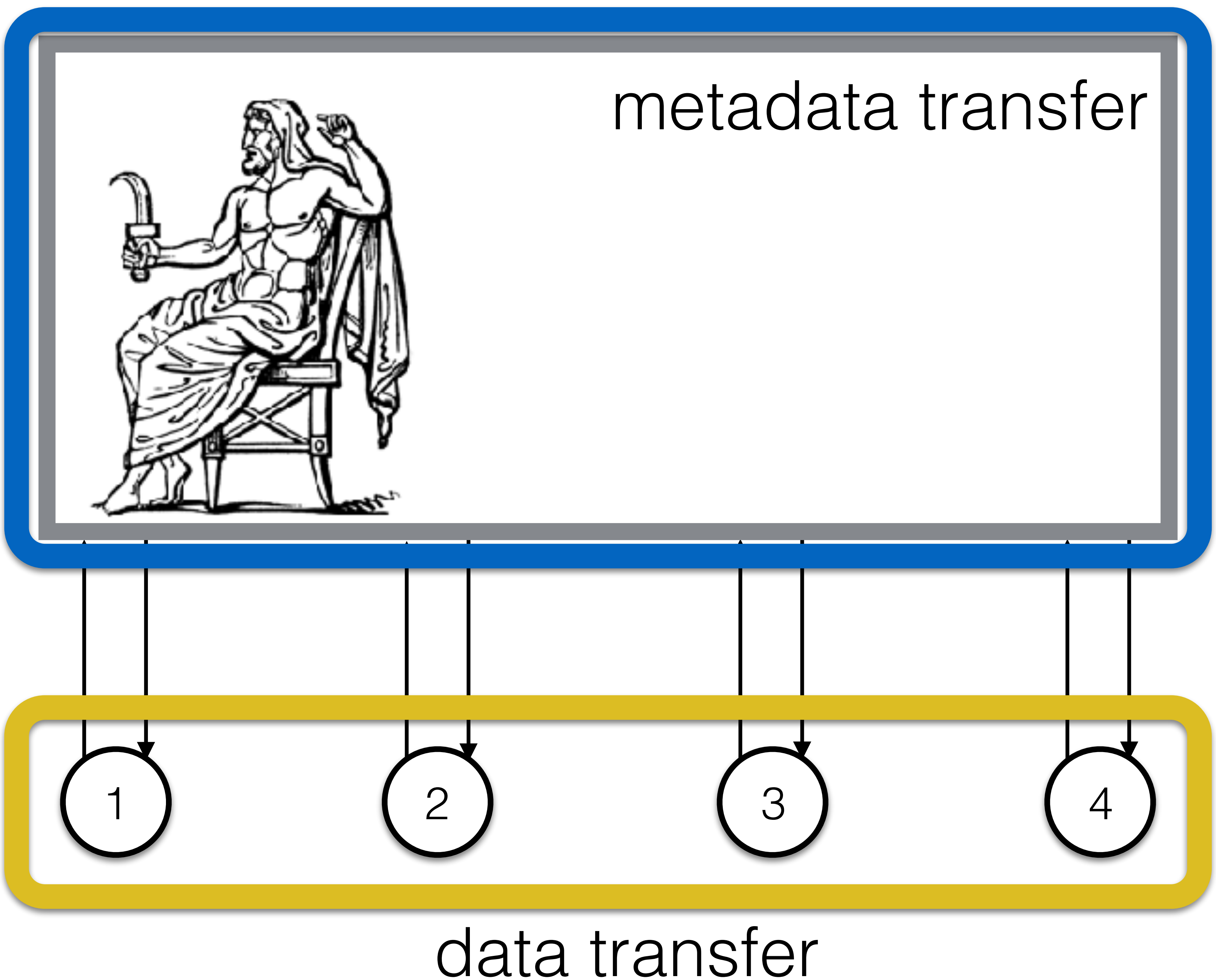
Decoupling data and metadata



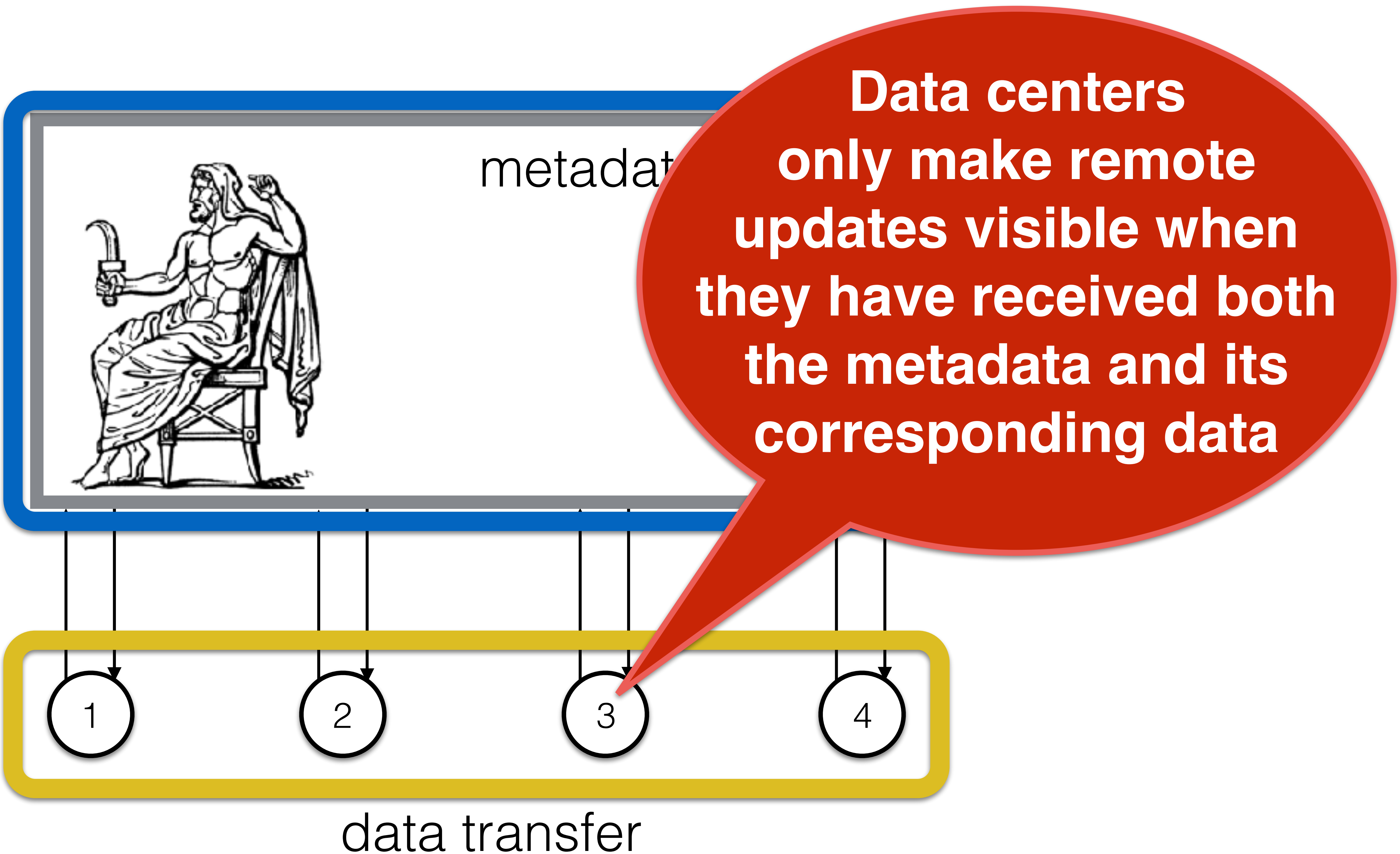
Decoupling data and metadata



Decoupling data and metadata

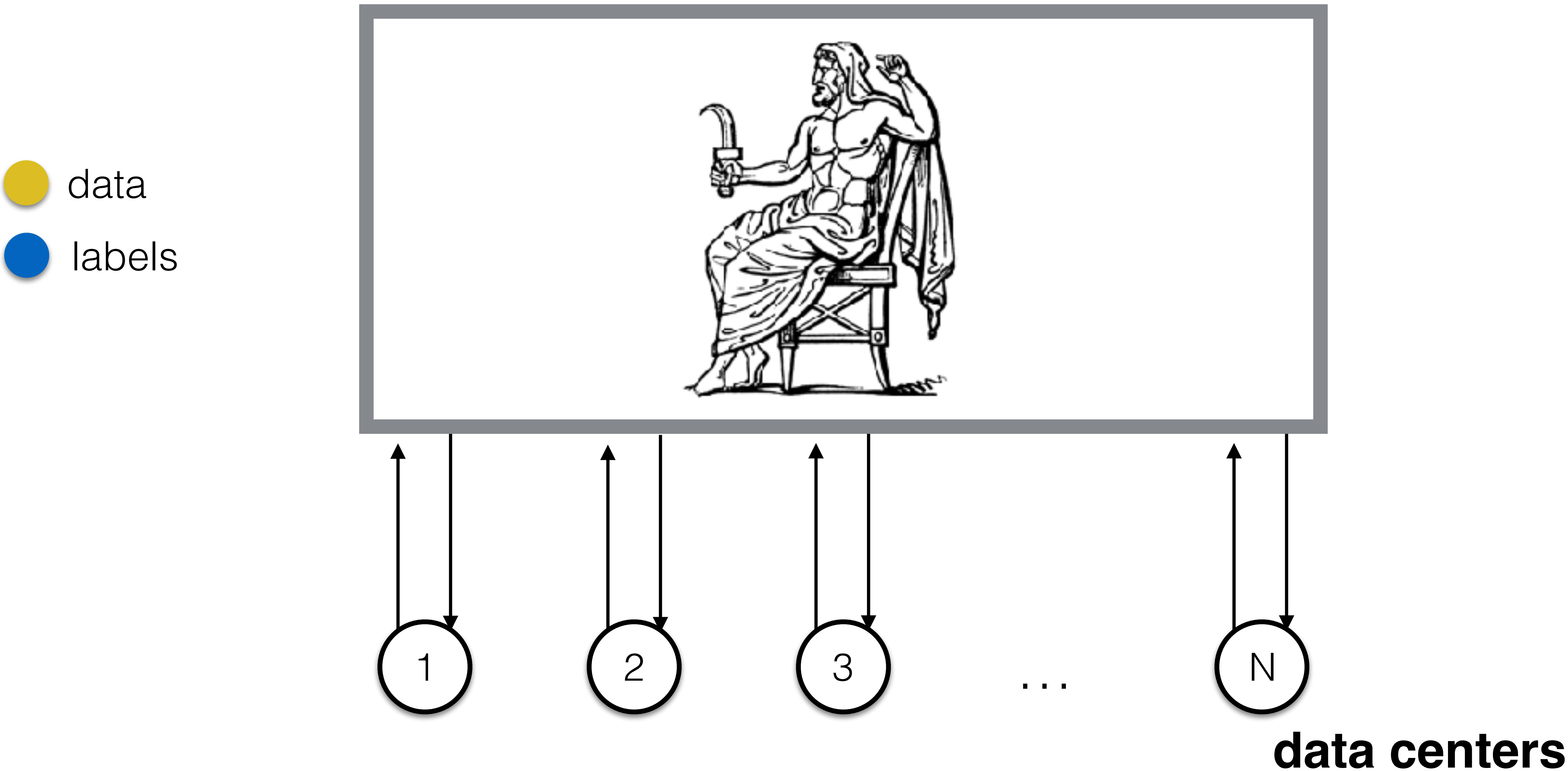


Decoupling data and metadata

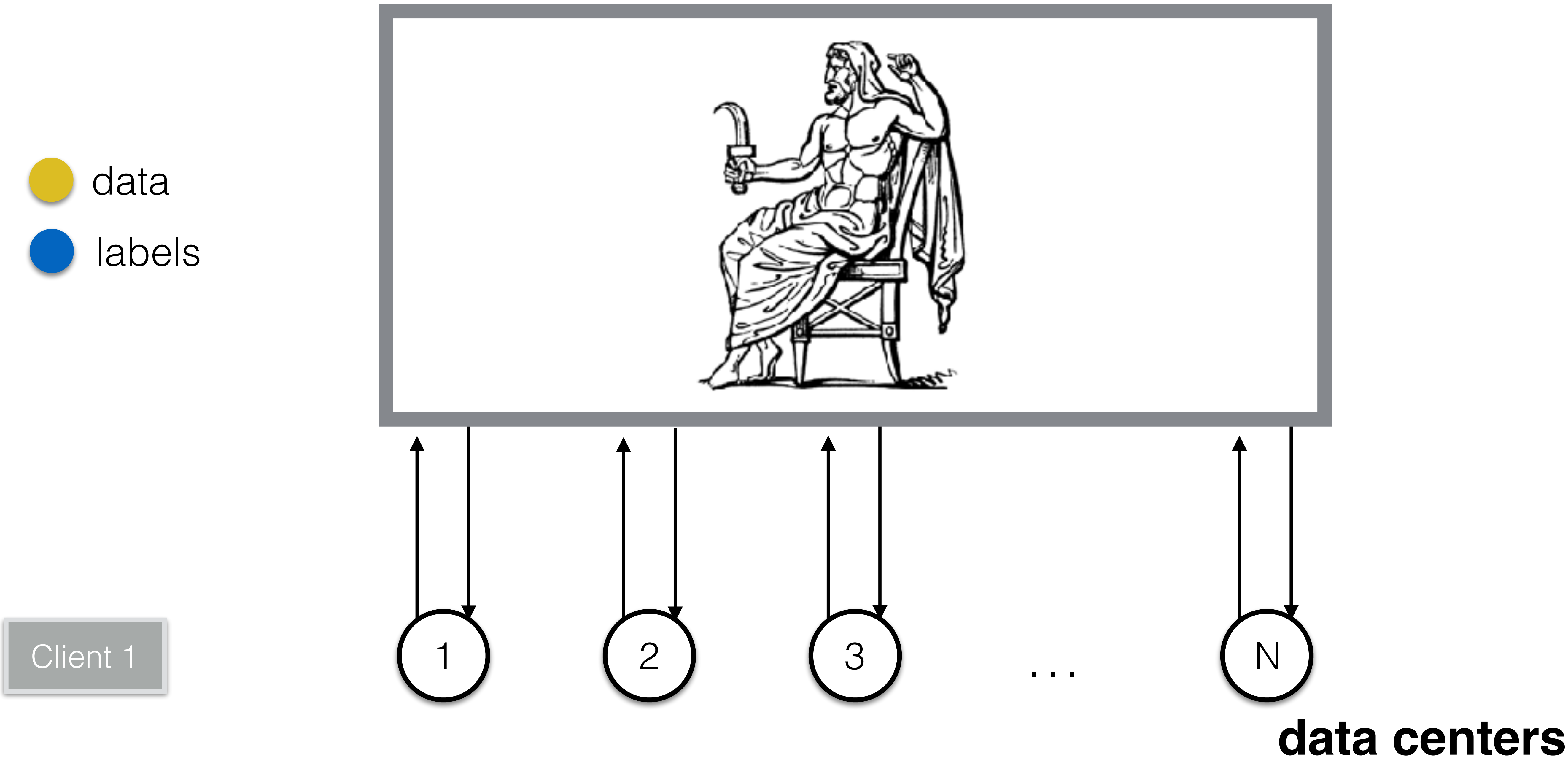


Example: write request

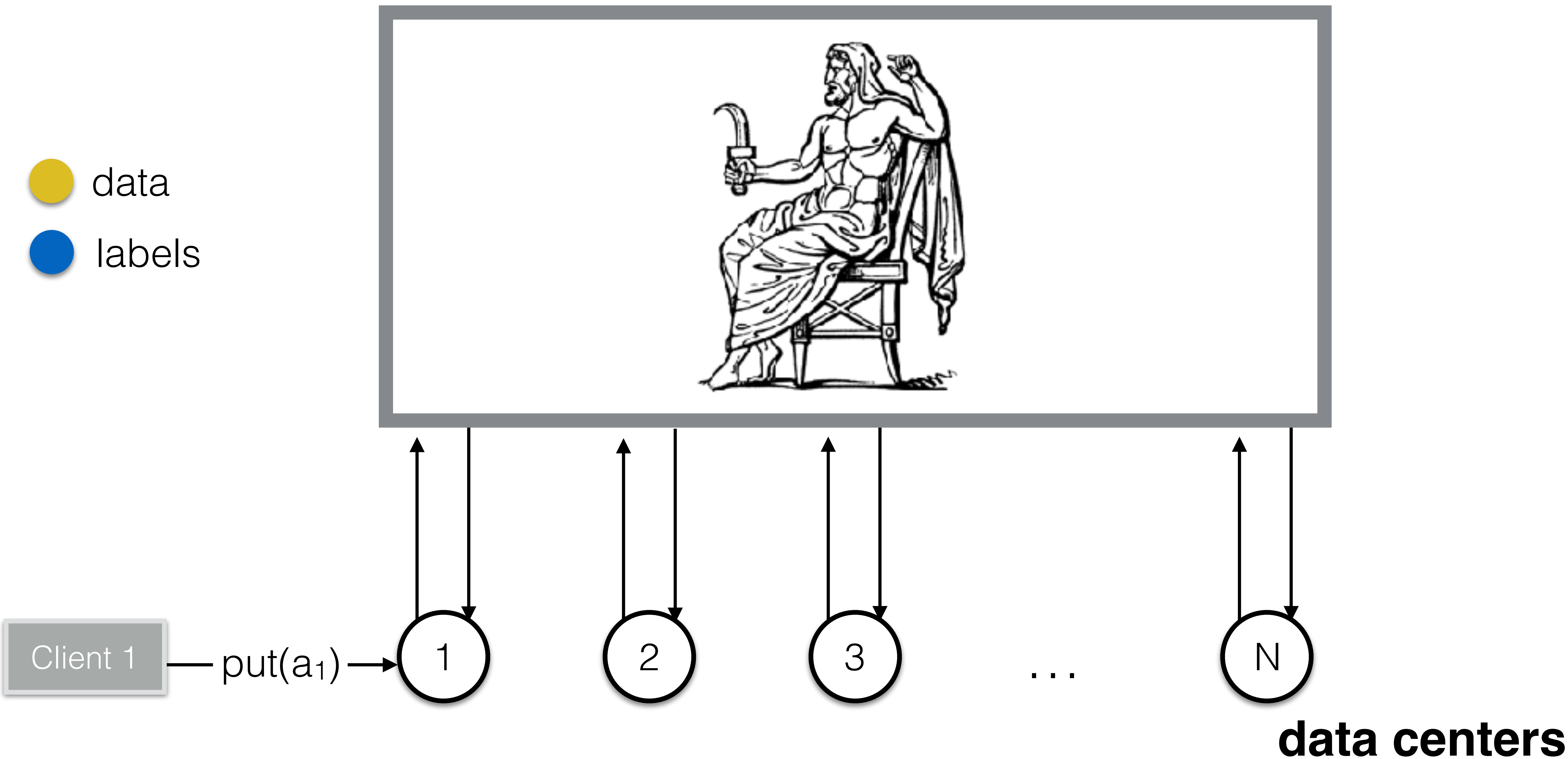
Example: write request



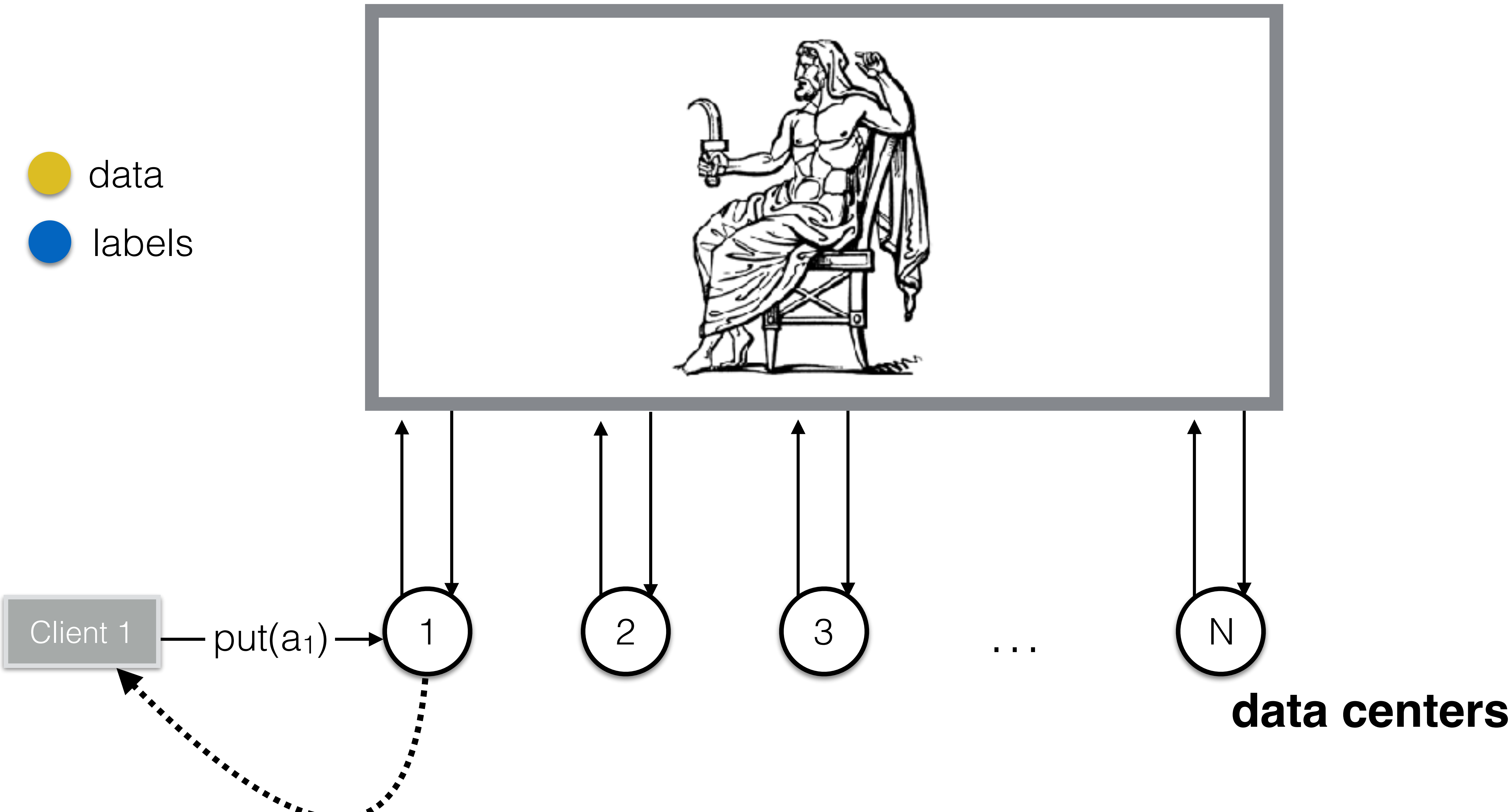
Example: write request



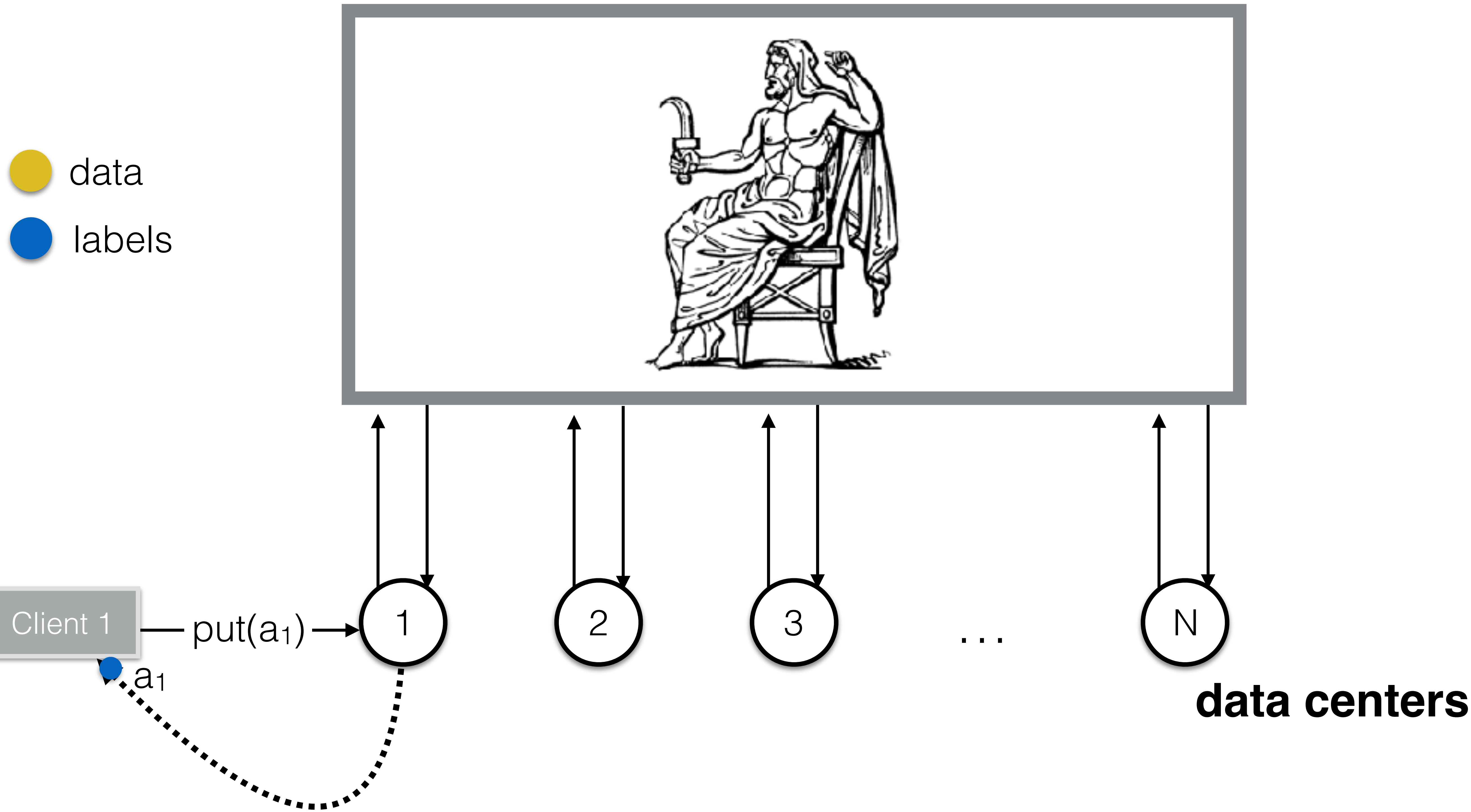
Example: write request



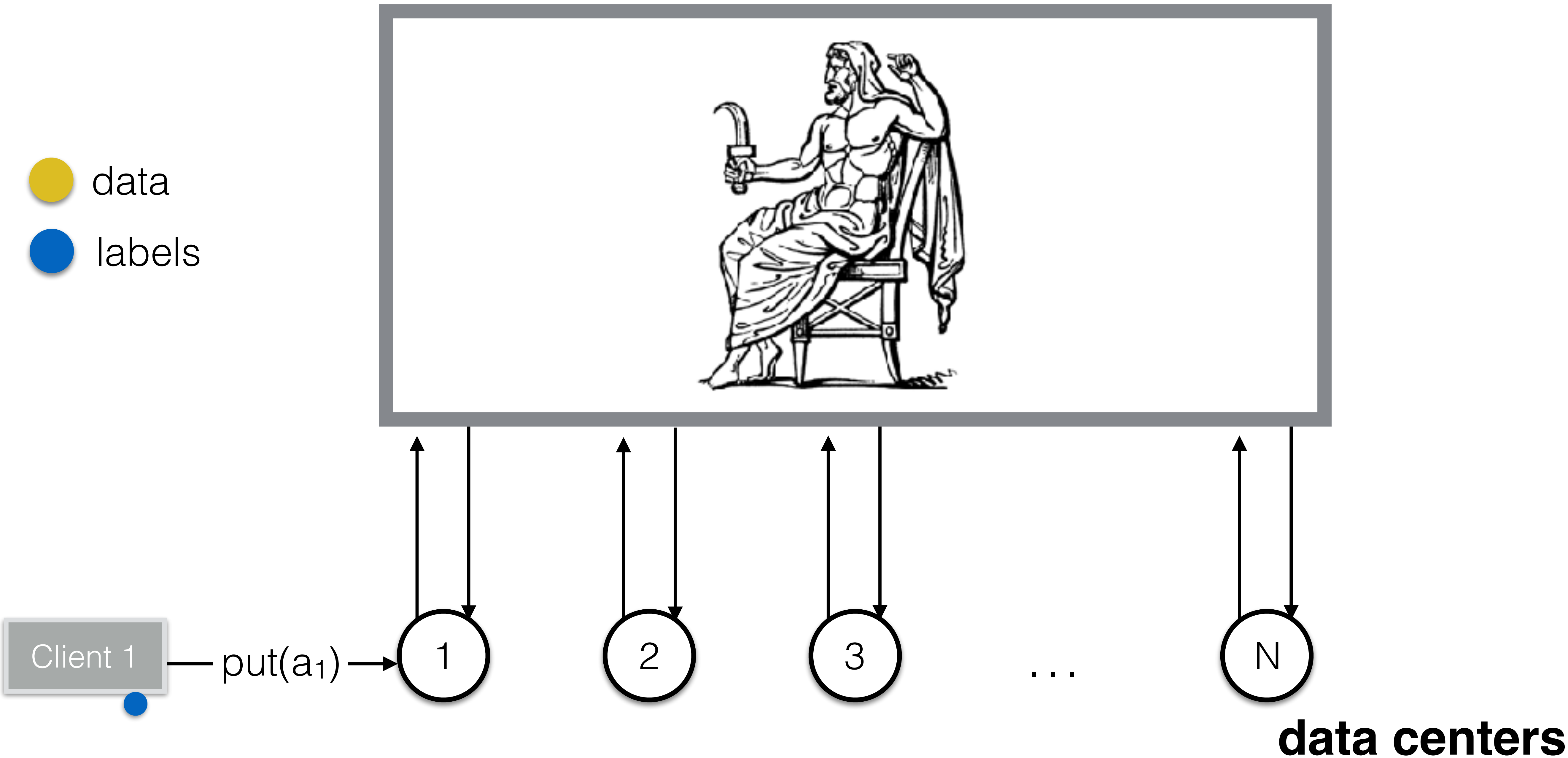
Example: write request



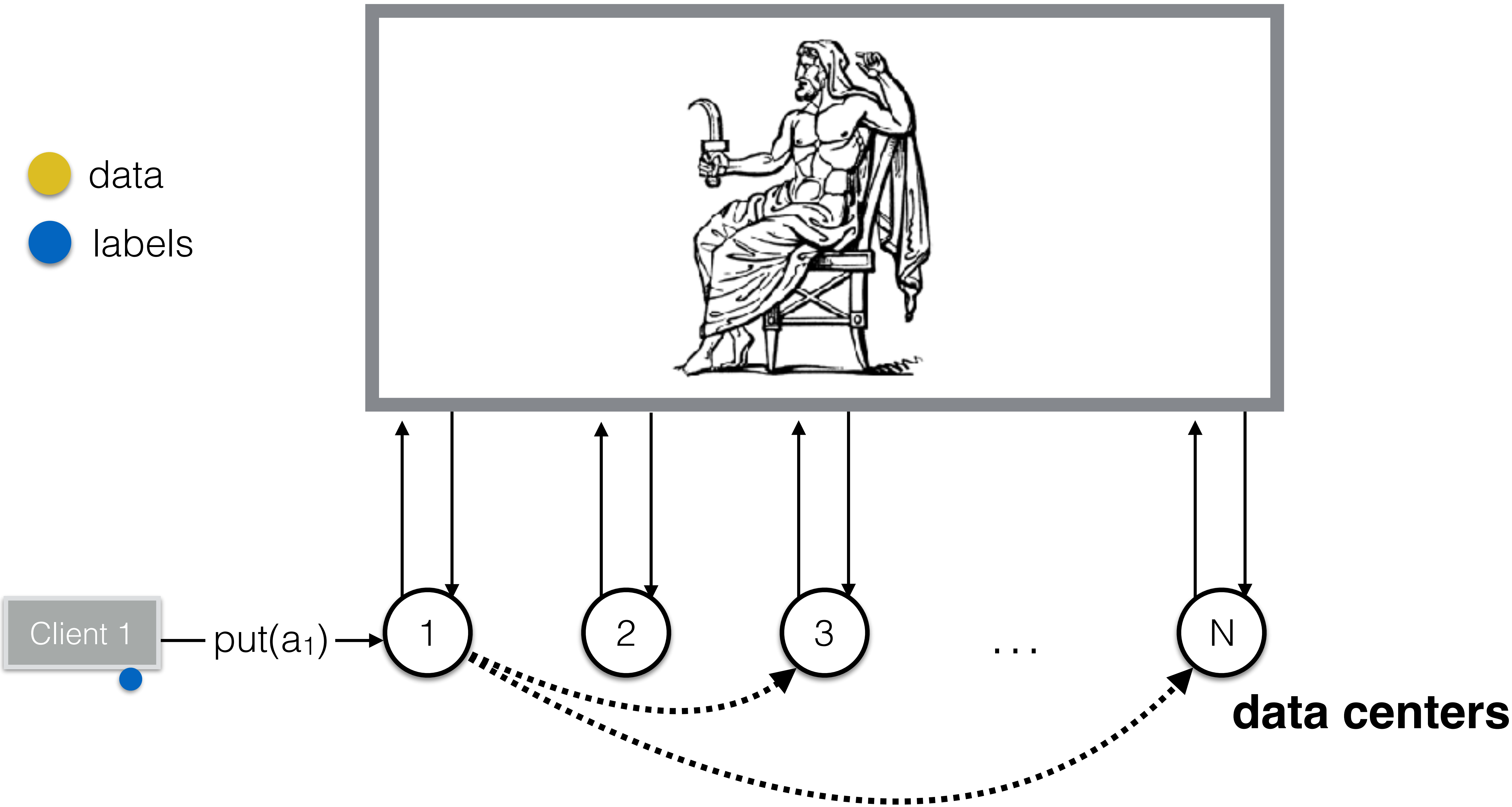
Example: write request



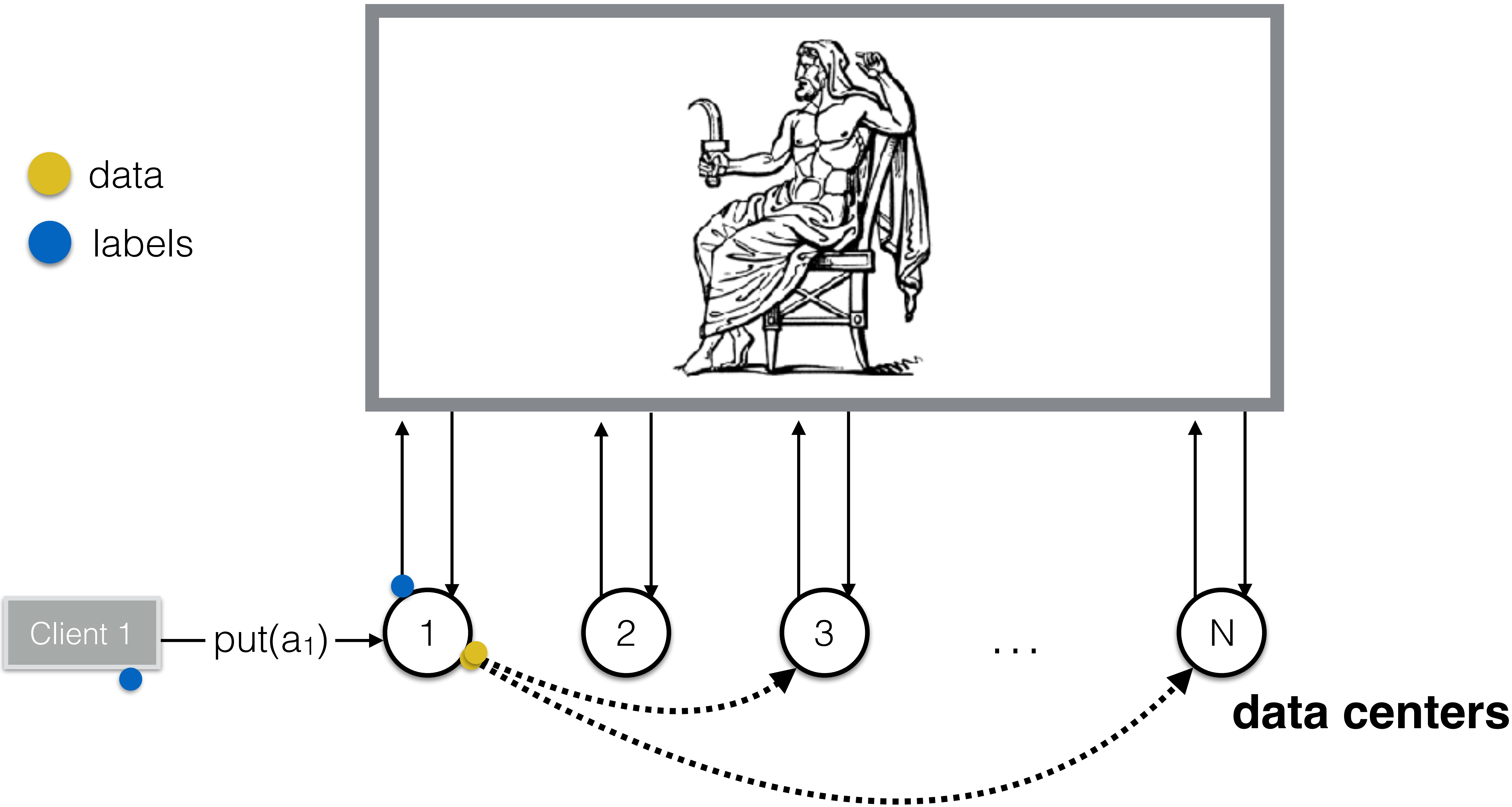
Example: write request



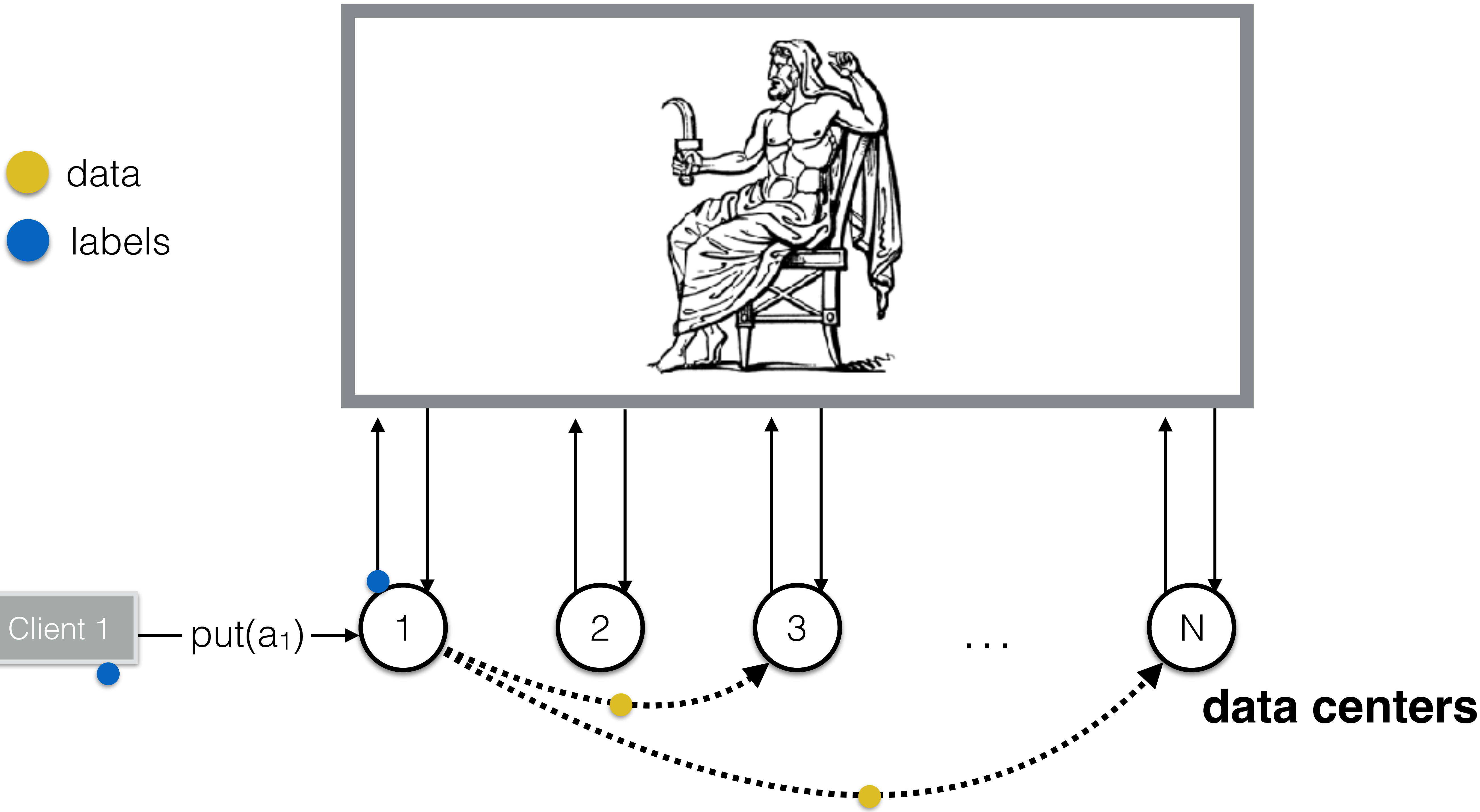
Example: write request



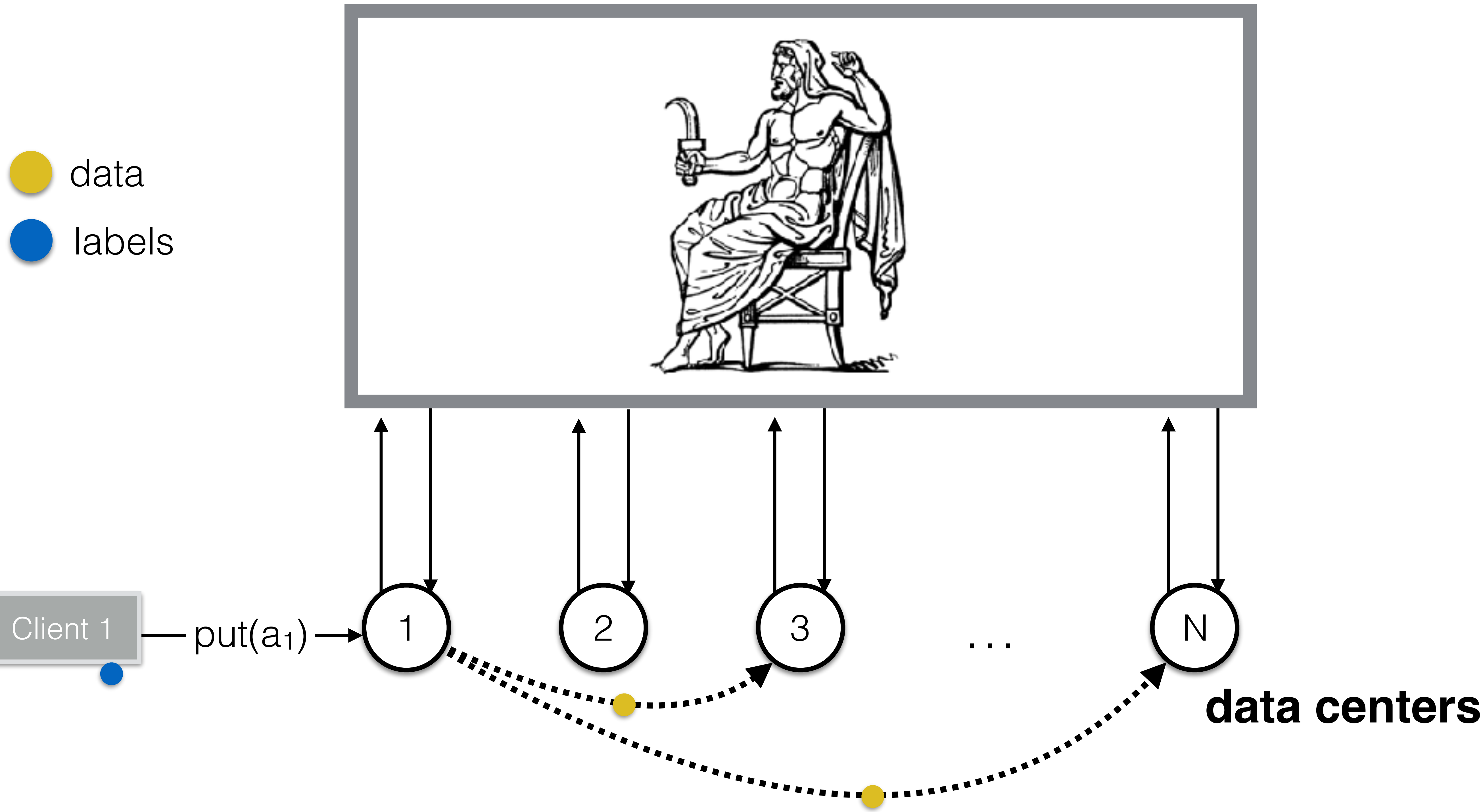
Example: write request



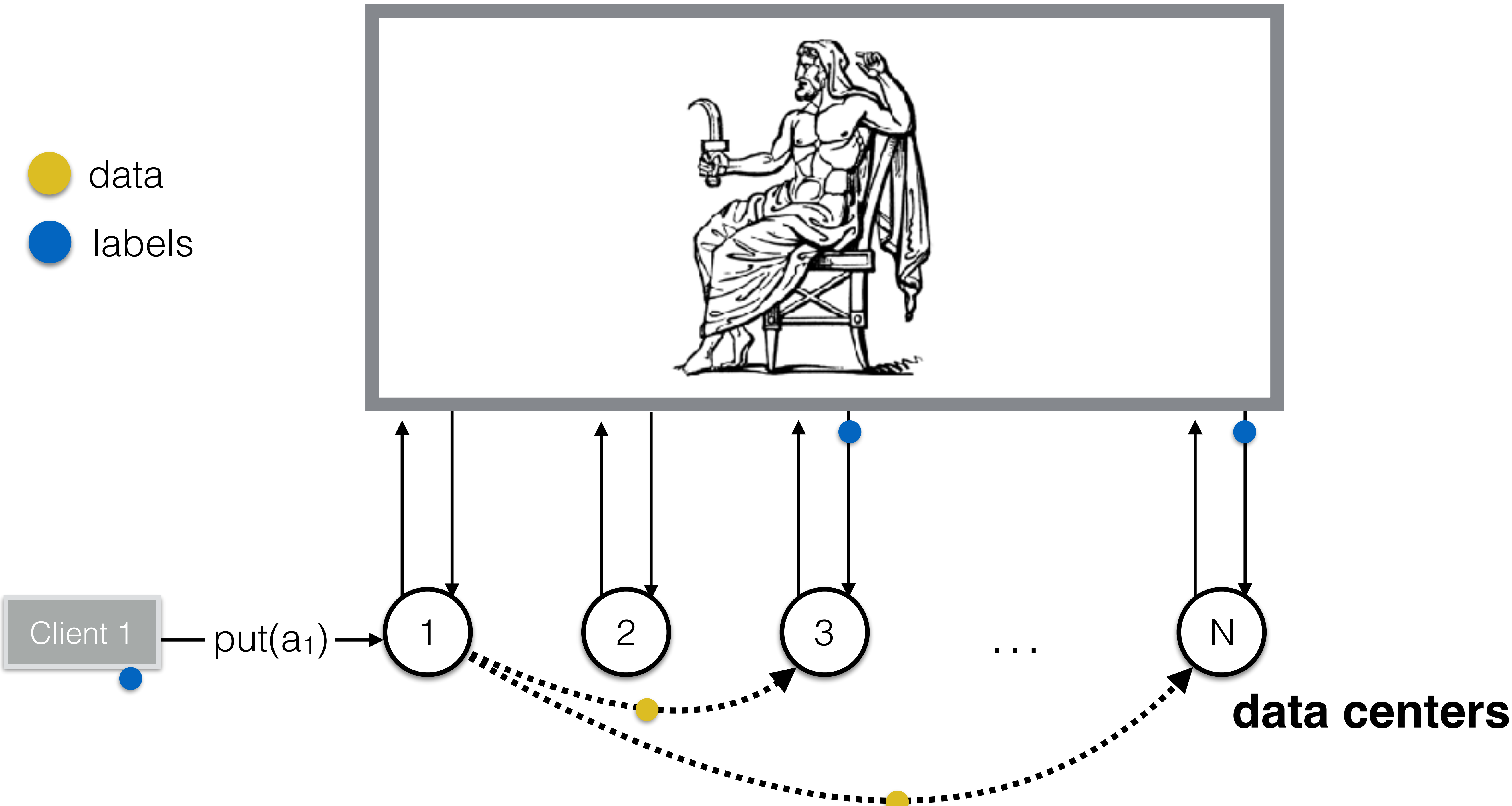
Example: write request



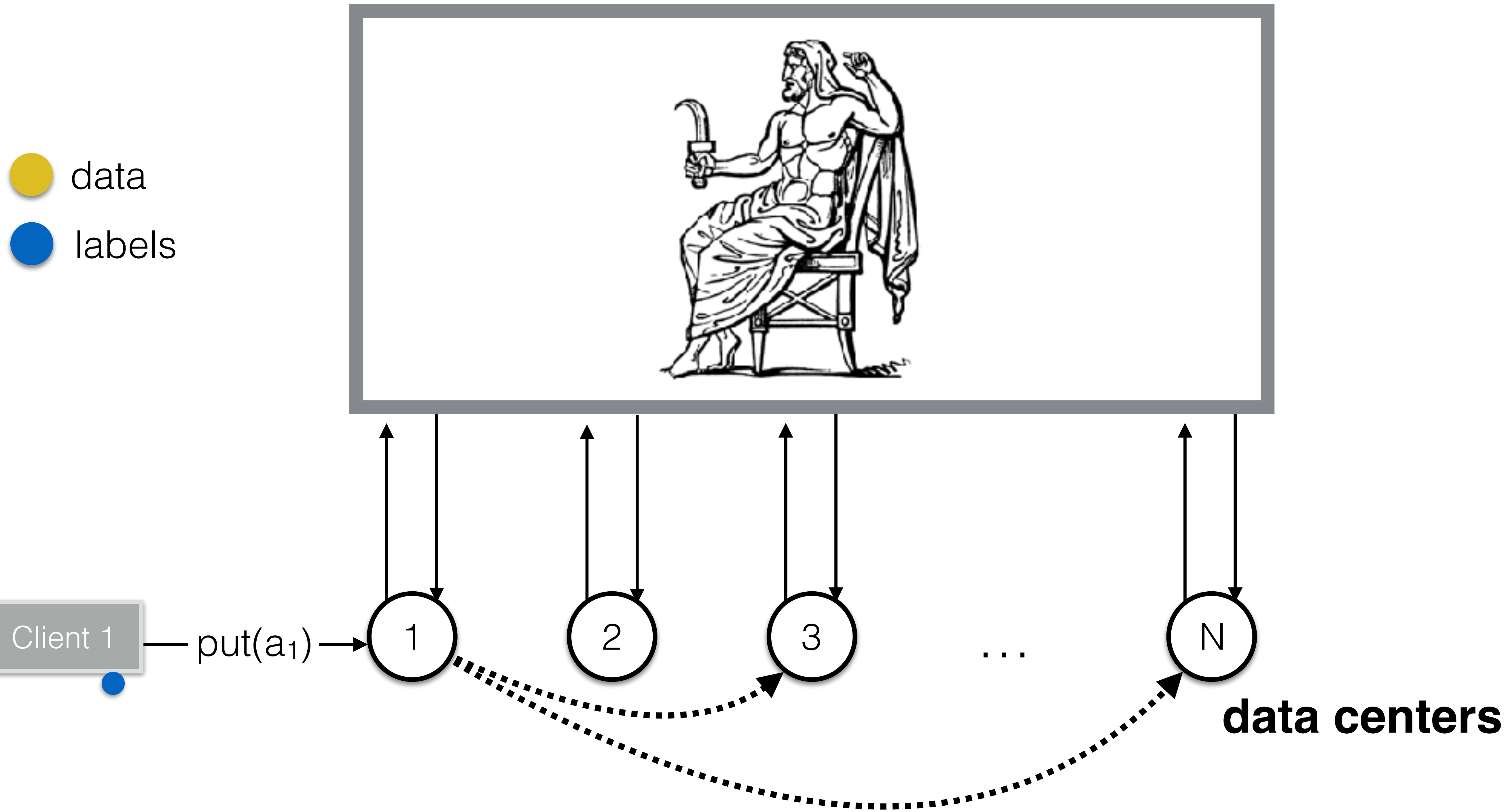
Example: write request



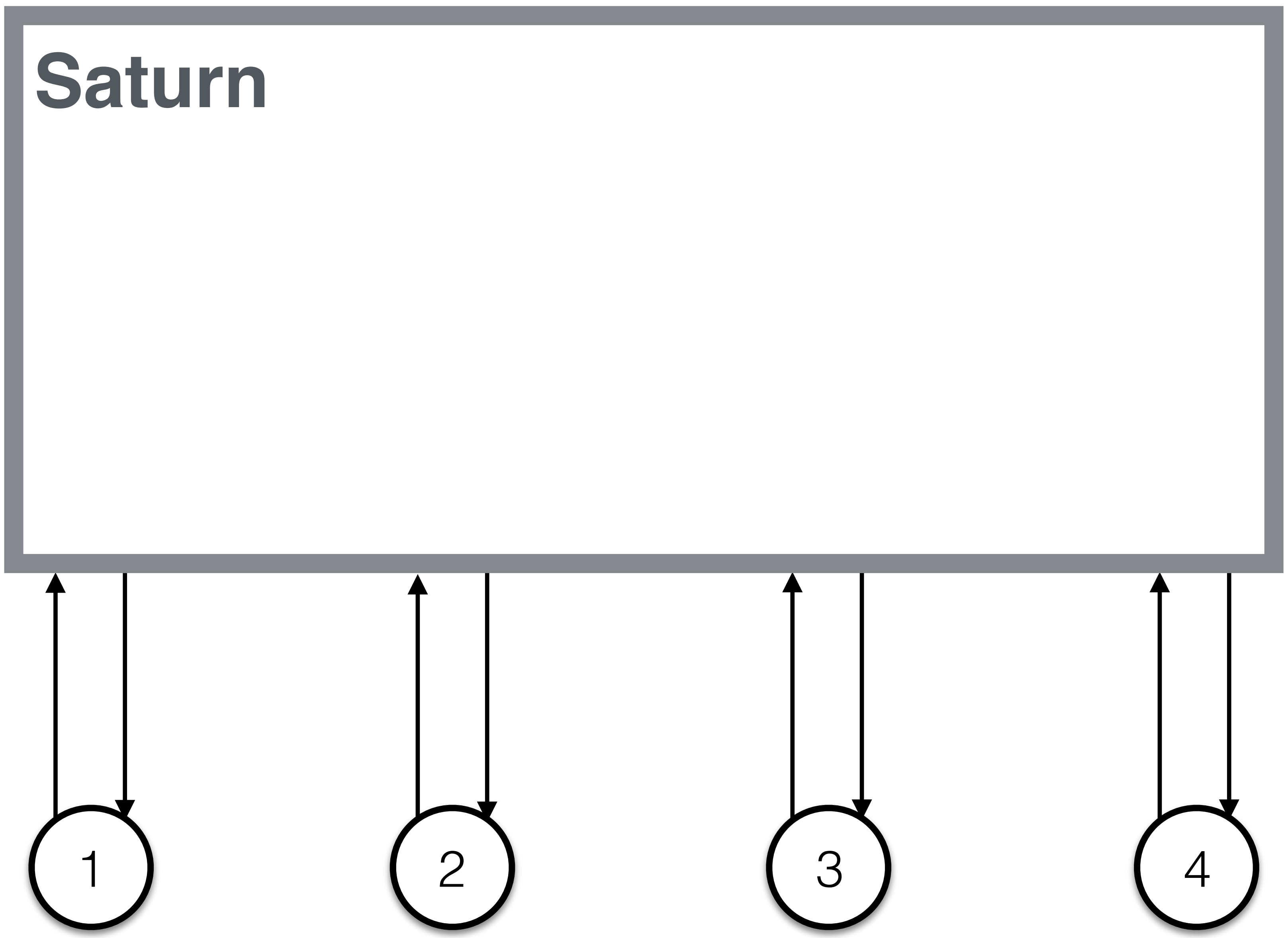
Example: write request



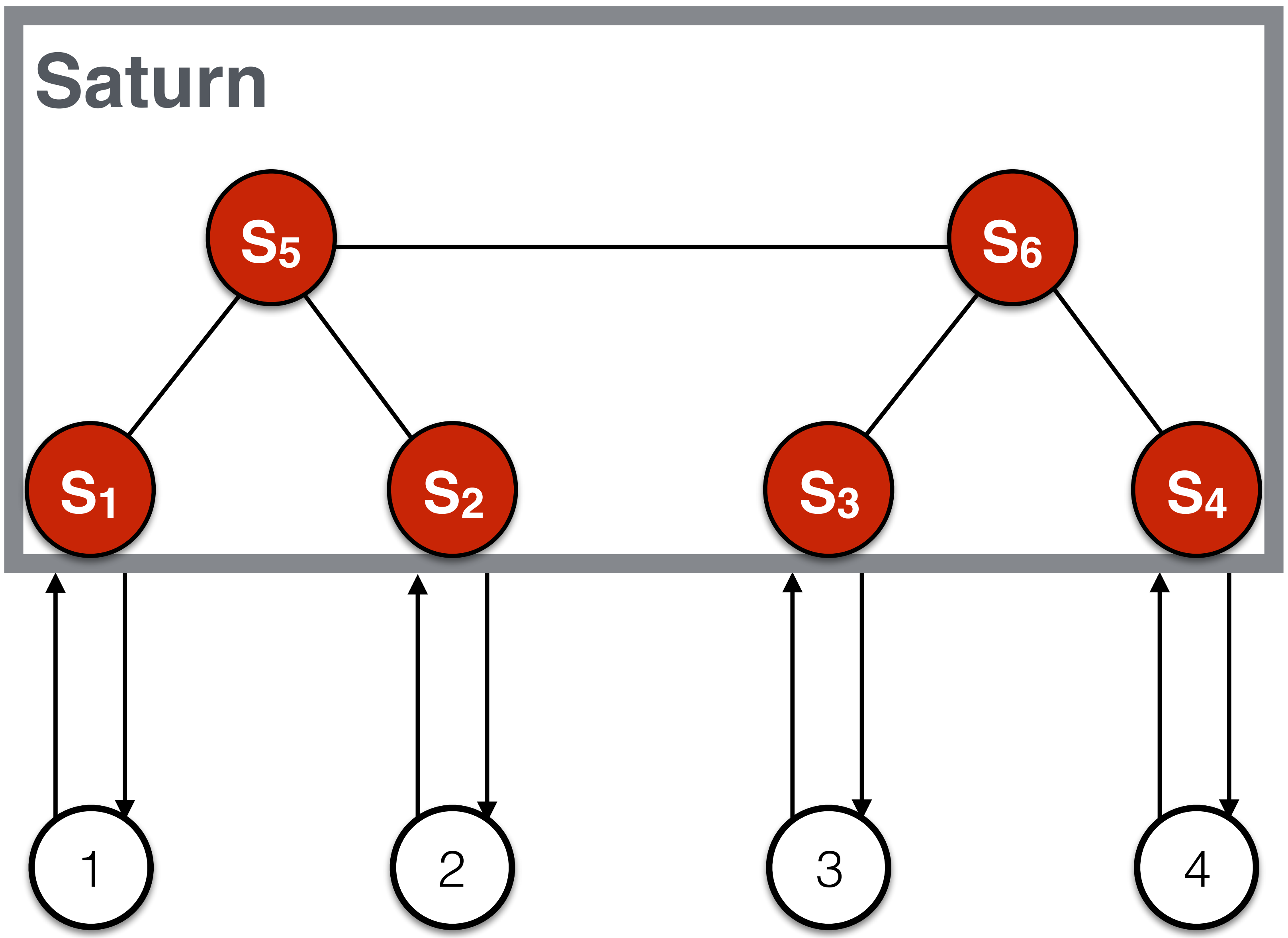
Example: write request



Metadata dissemination graph



Metadata dissemination graph



Optimal dissemination graph

The goal is to build the tree such that metadata-paths latencies (through the tree) **match** data-paths

Weighted Minimal Mismatch

$$mismatch_{i,j} = |\Delta^{M}(i,j) - \Delta(i,j)|$$

$$min \sum_{\forall i,j \in V} c_{i,j} \cdot mismatch_{i,j}$$

Optimal dissemination graph

The goal is to build the tree such that metadata-paths latencies (through th data-paths

absolute difference between label-paths and data paths

Weighted Minimal Mismat

$$mismatch_{i,j} = |\Delta^M(i,j) - \Delta(i,j)|$$

$$min \sum_{\forall i,j \in V} c_{i,j} \cdot mismatch_{i,j}$$

Optimal dissemination graph

The goal is to build the tree such that metadata-paths latencies (through the tree) **match** data-paths

Weighted Minimal Mismatch

$$mismatch_{i,j} = |\Delta^M(i,j) -$$

minimize mismatch of busiest paths

$$min \sum_{\forall i,j \in V} c_{i,j} \cdot mismatch_{i,j}$$

Metadata propagation: building the tree

Finding the optimal tree is modelled as a **constraint optimization** problem

Input

- Data-paths average latencies
- Candidate locations for serializers (an latencies among them)
- Access-patterns: to minimize the impact of mismatches

Reading

Reading/writing from the “local” datacenter is non-blocking: dependencies do not need to be checked at every operation

Reading

Reading/writing from the “local” datacenter is non-blocking: dependencies do not need to be checked at every operation

Due to partial replication not all data is replicated locally: client needs to “migrate” to perform remote reads

Reading

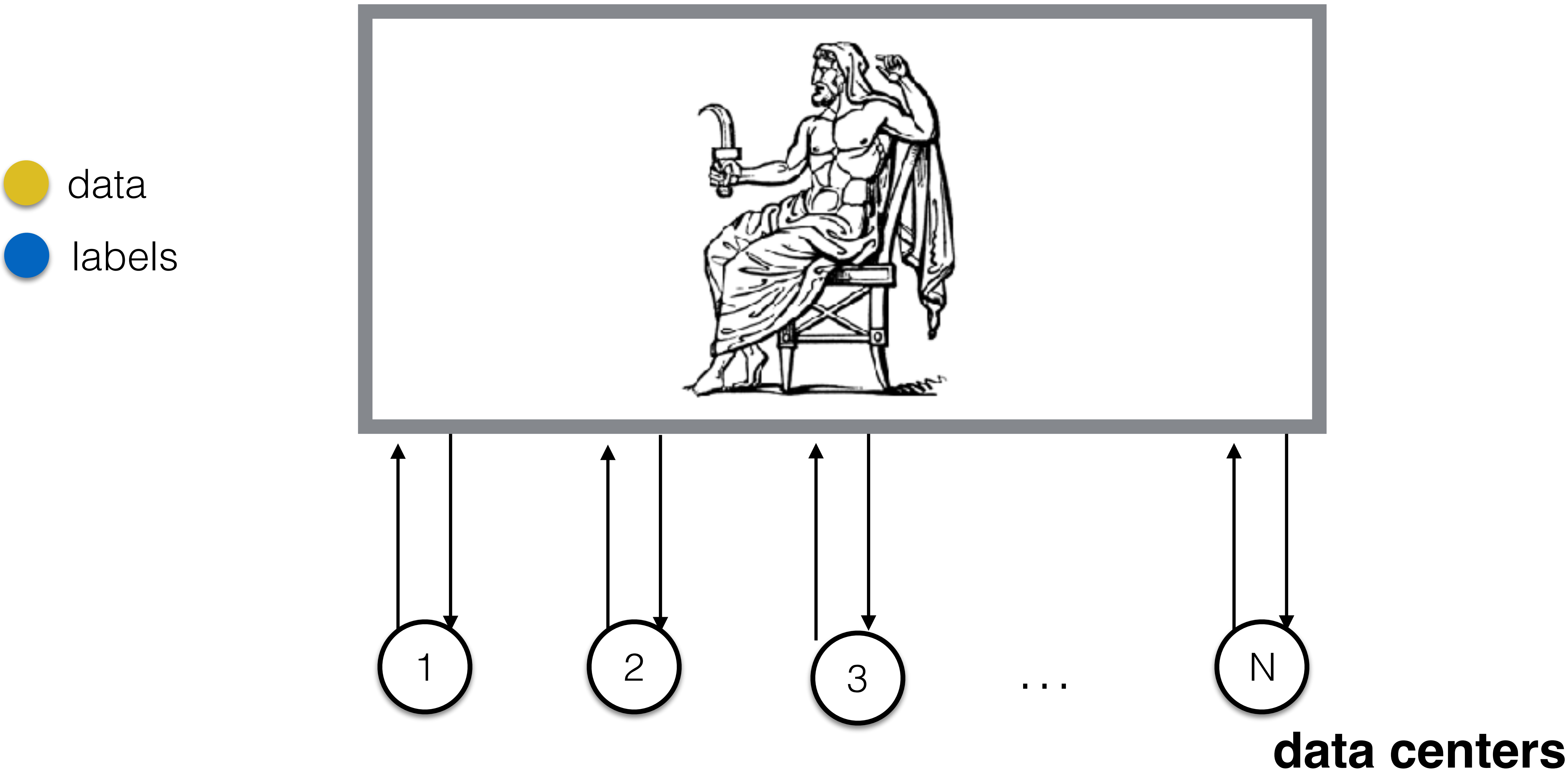
Reading/writing from the “local” datacenter is non-blocking: dependencies do not need to be checked at every operation

Partial replication: not all data is replicated locally: client needs to “migrate” to perform remote reads

When migrating the client may need to block: waiting for remote datacenter to be “in sync” with its causal past

Example: migration

Example: migration

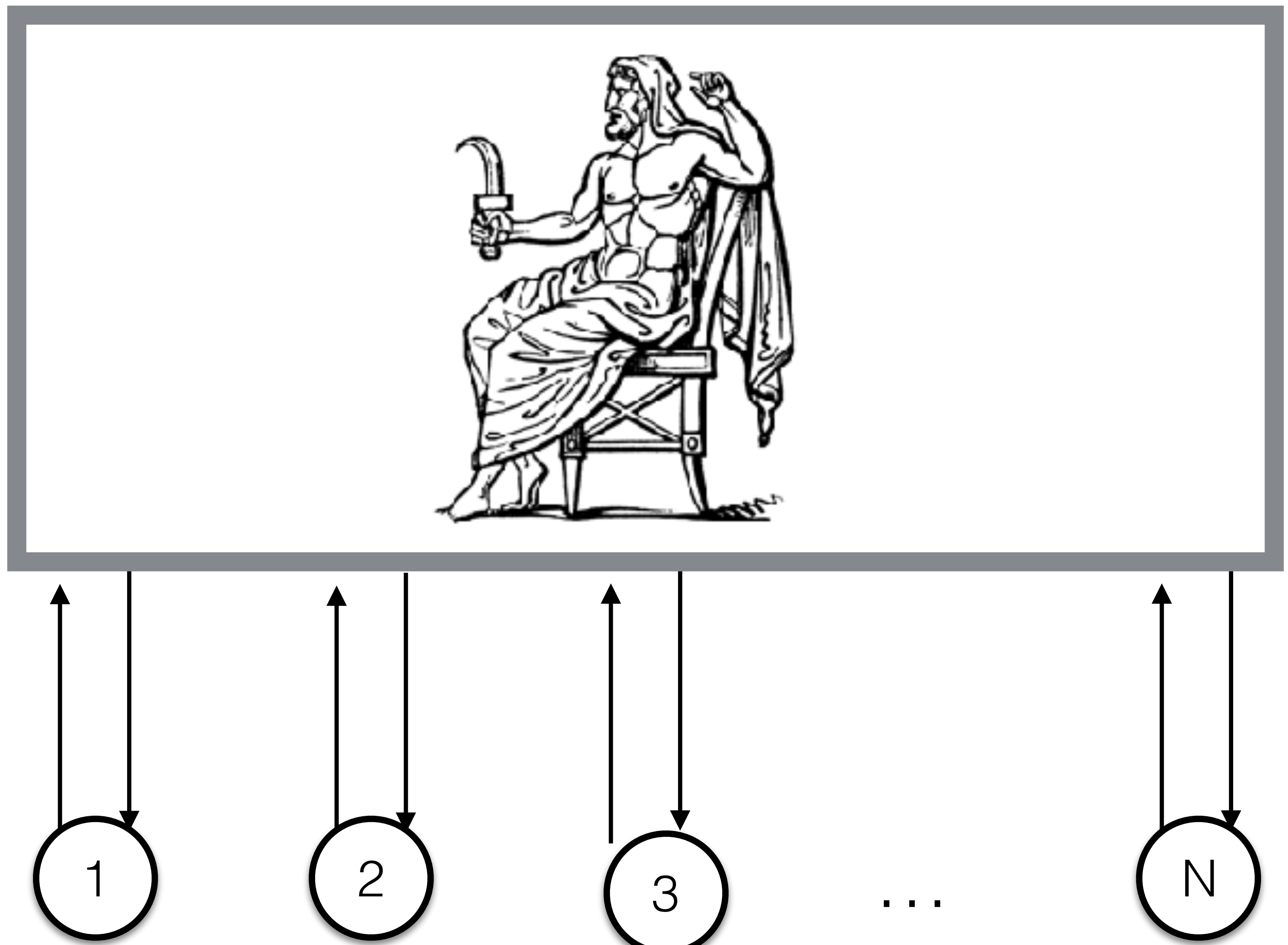


Example: migration

data

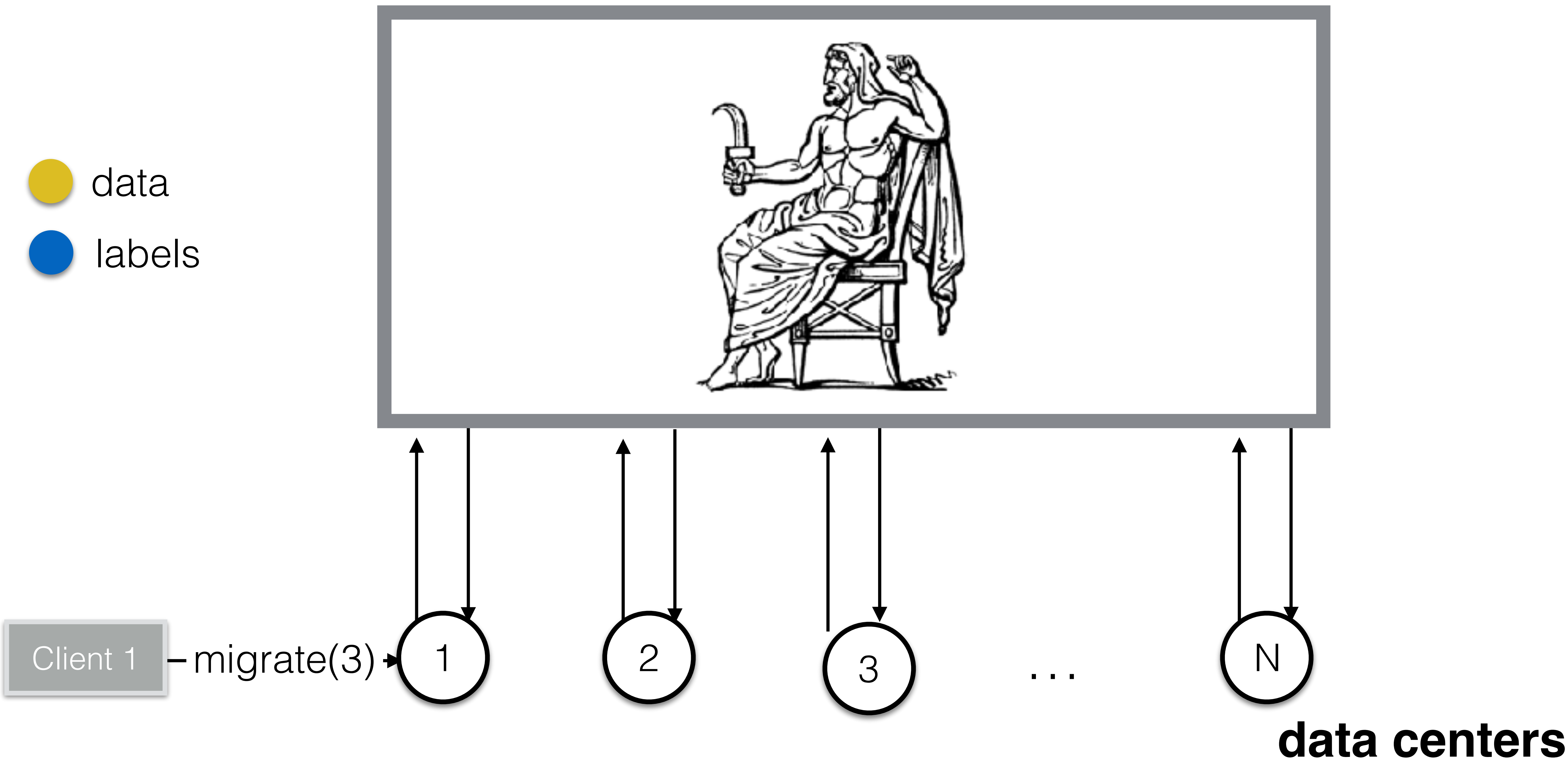
labels

Client 1

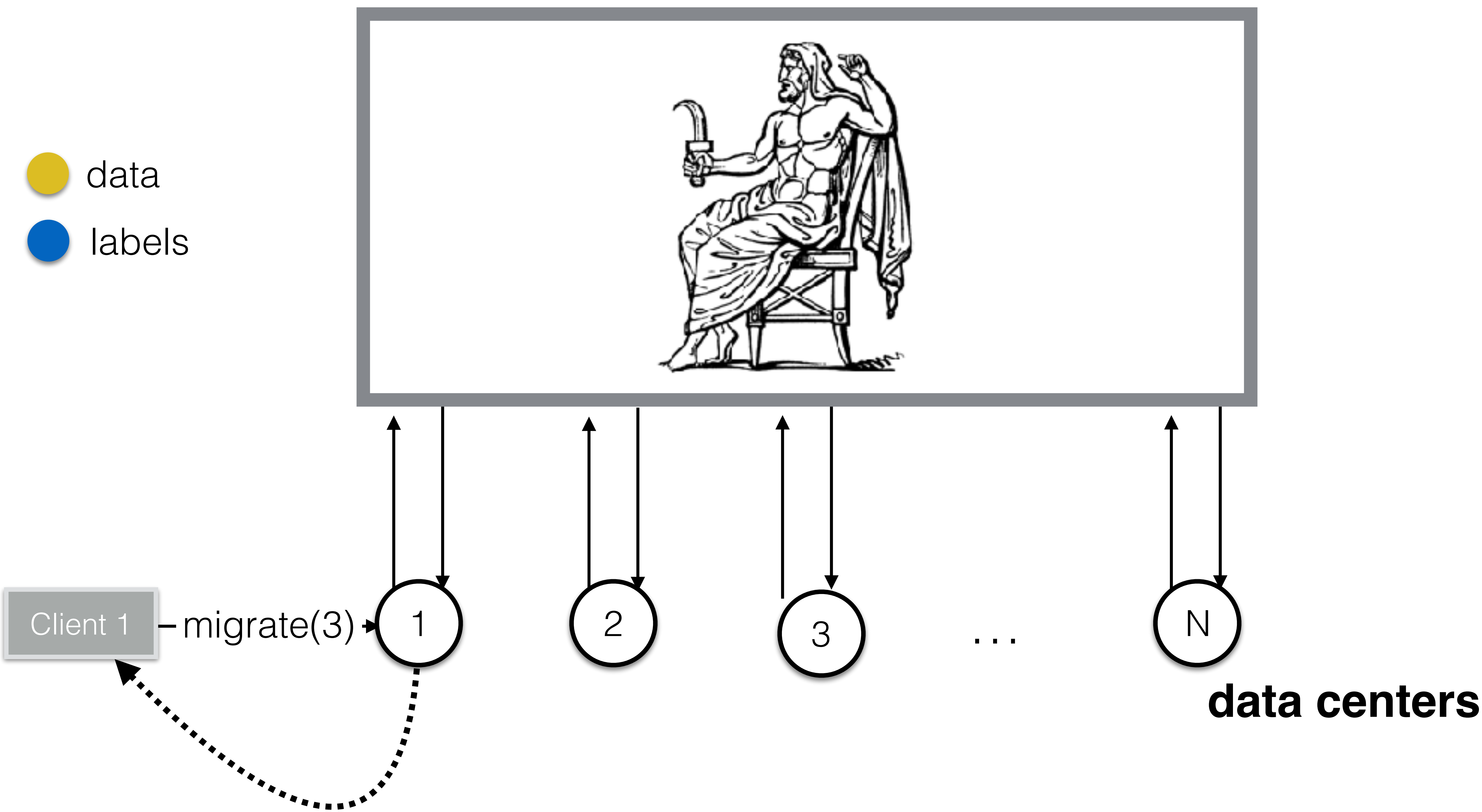


data centers

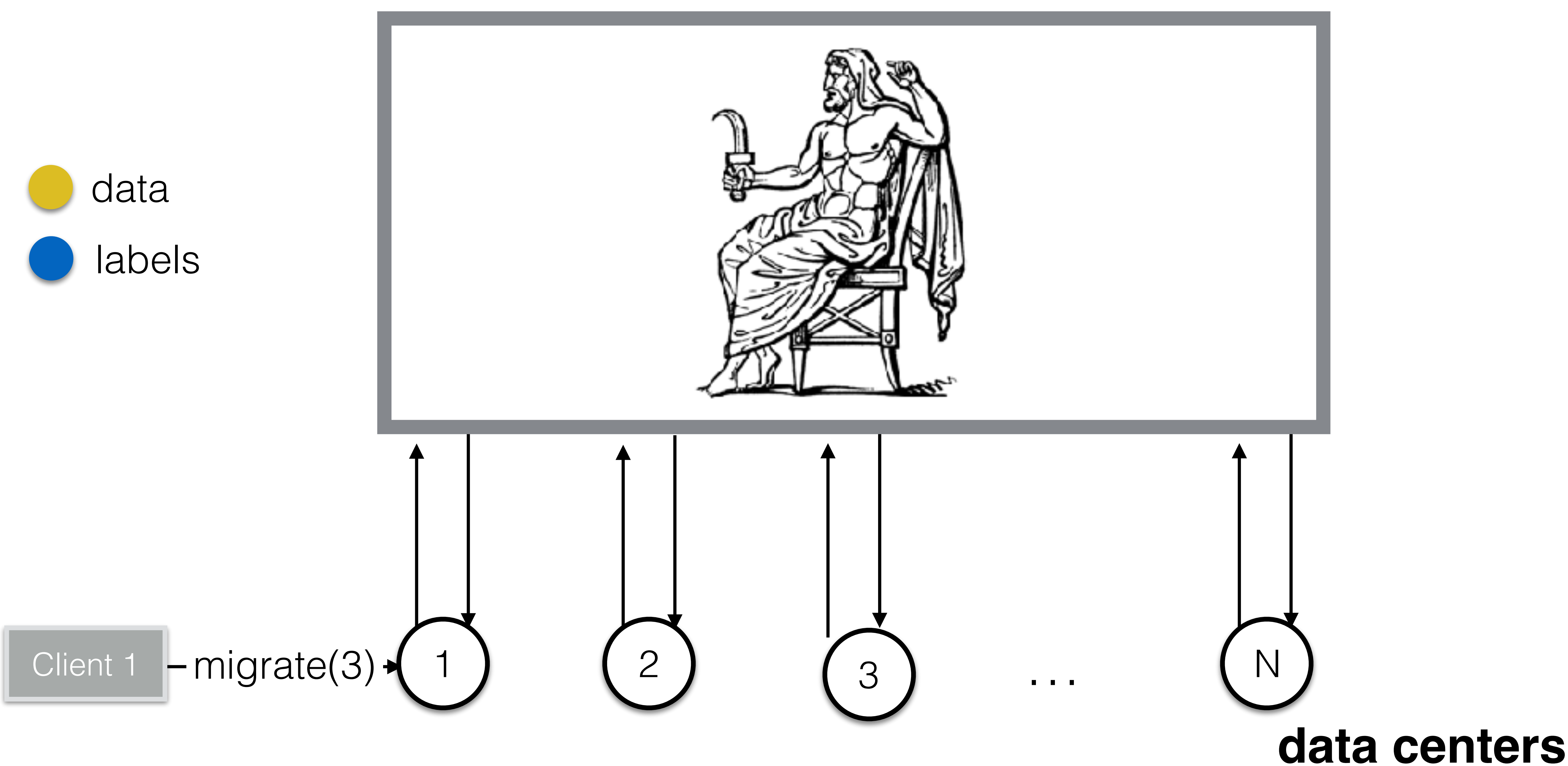
Example: migration



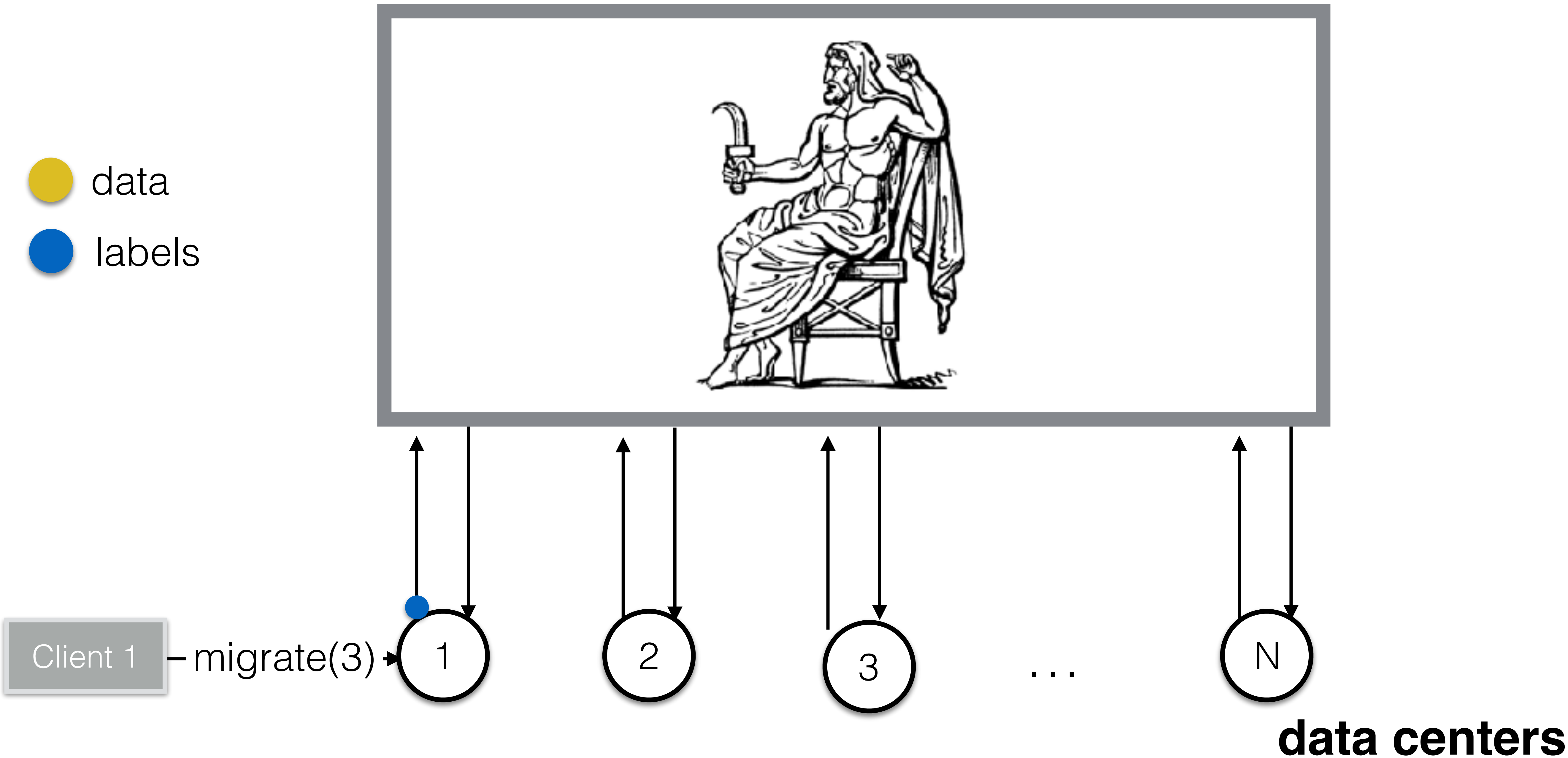
Example: migration



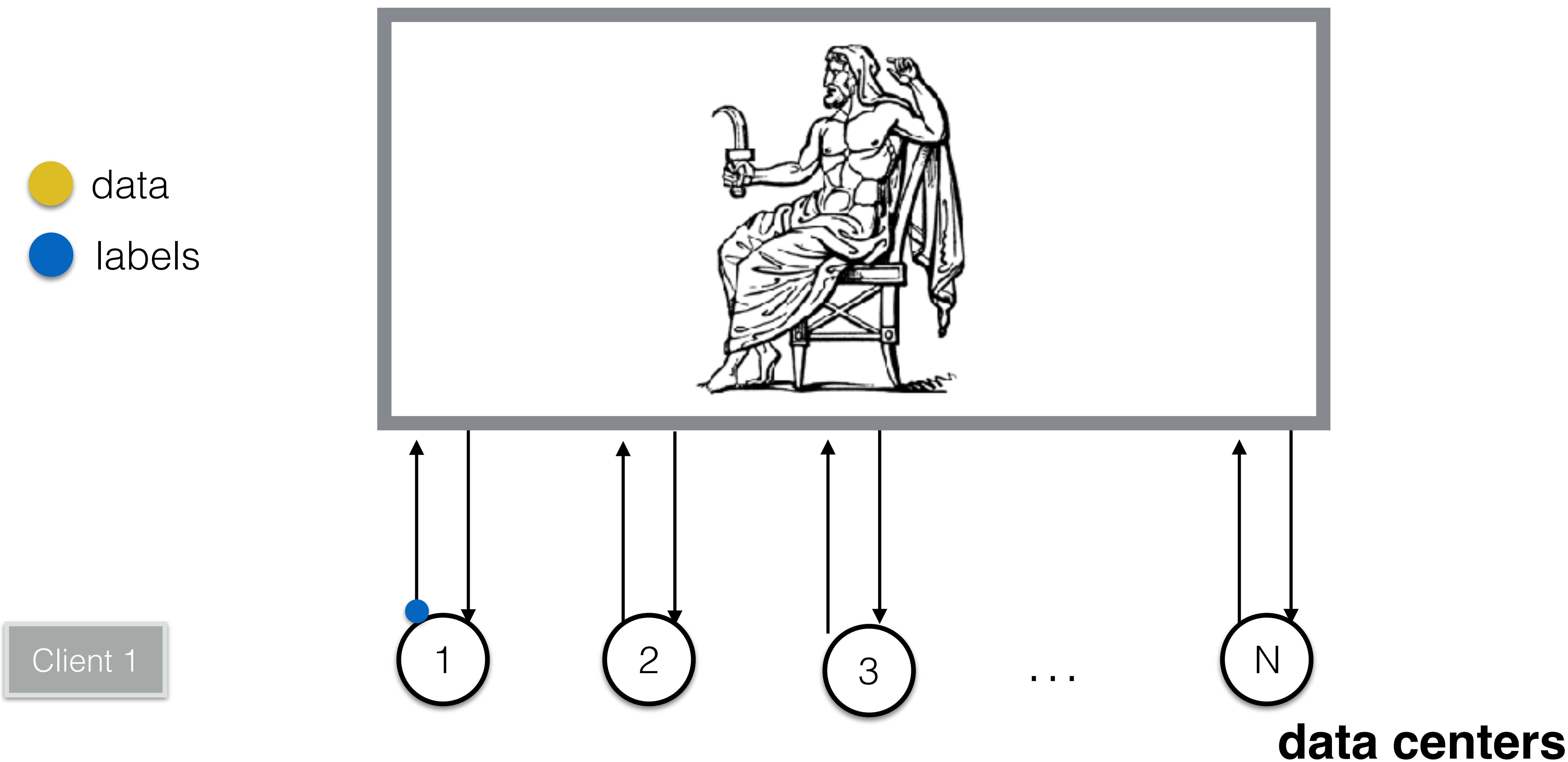
Example: migration



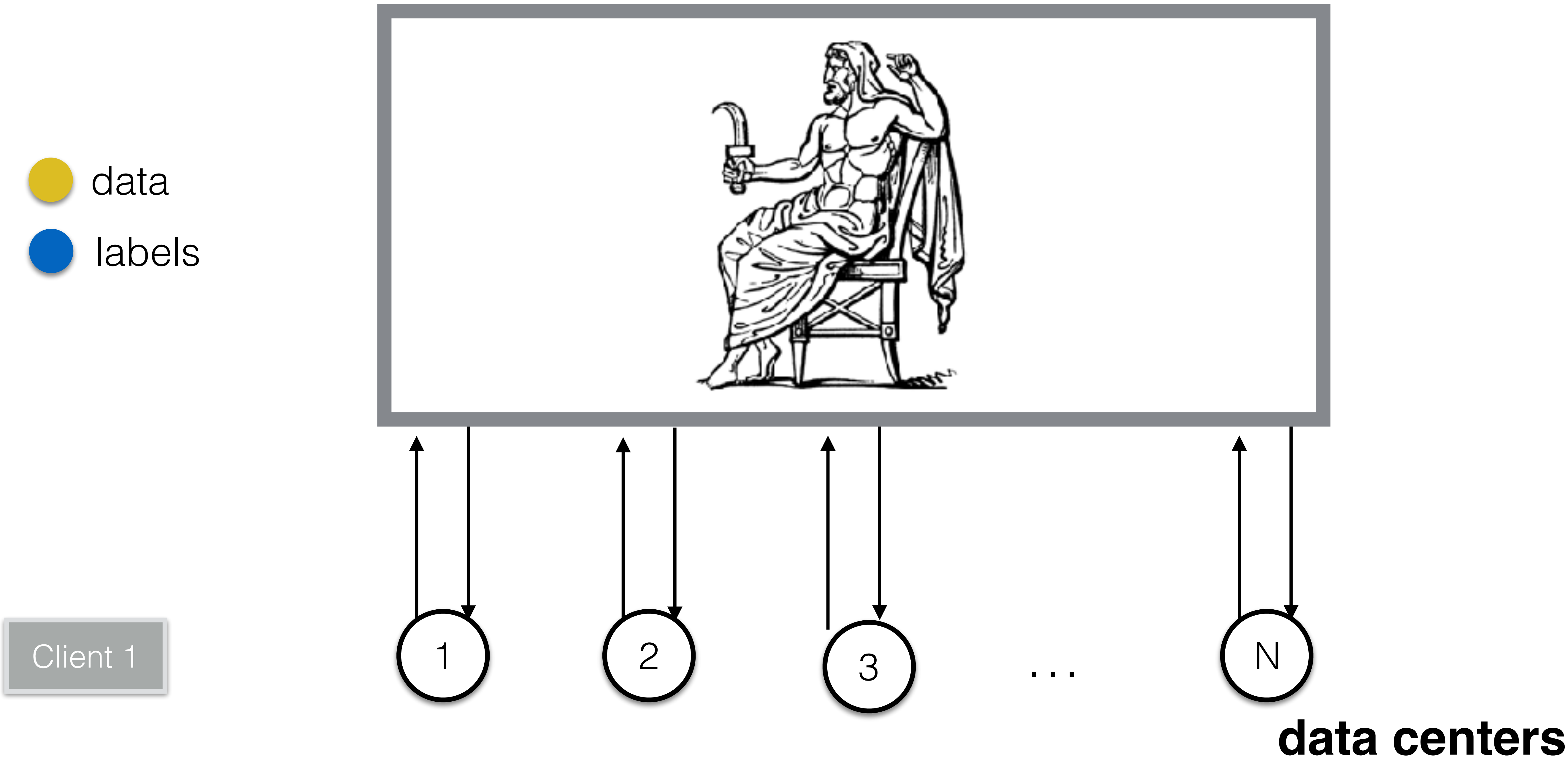
Example: migration



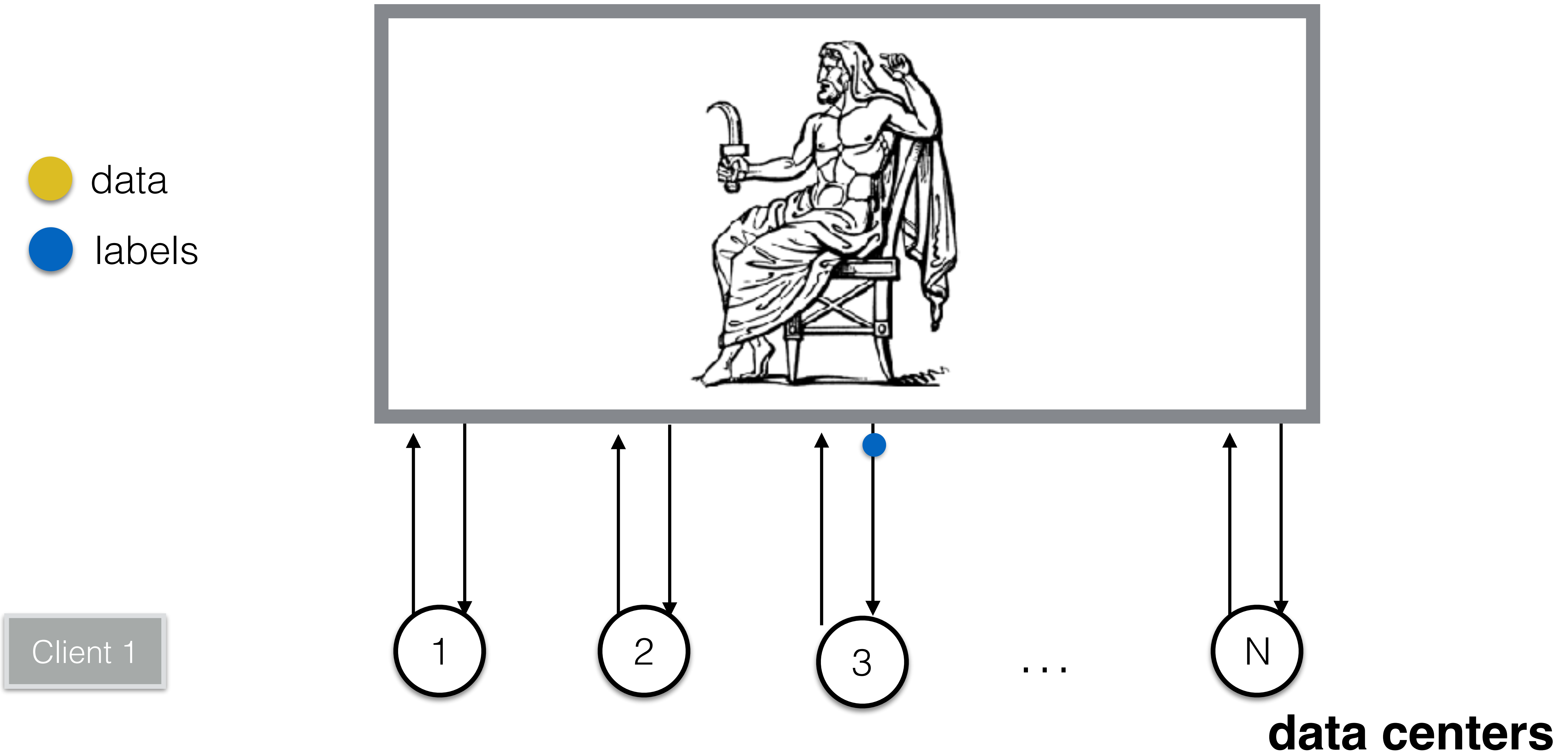
Example: migration



Example: migration



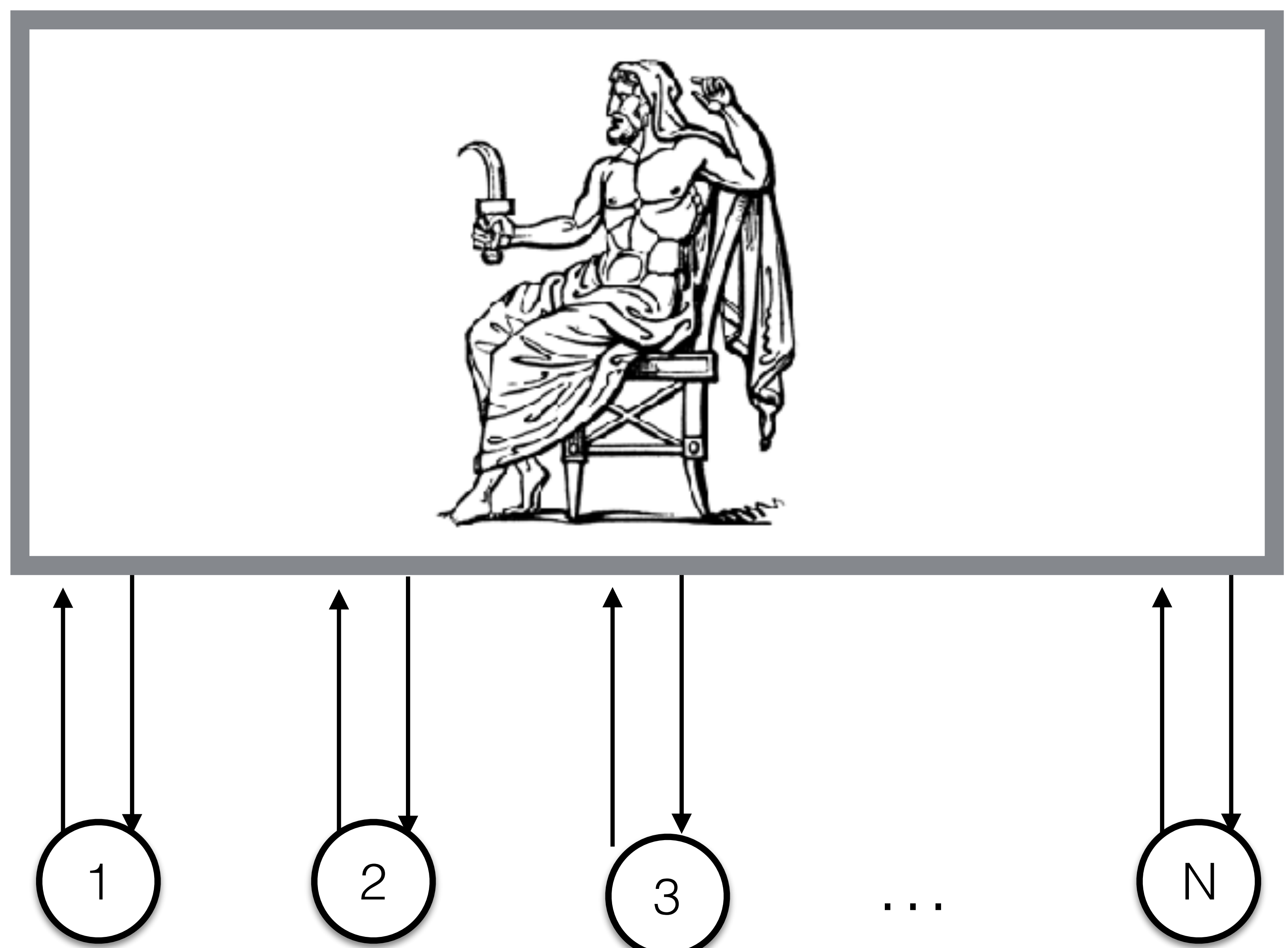
Example: migration



Example: migration

data

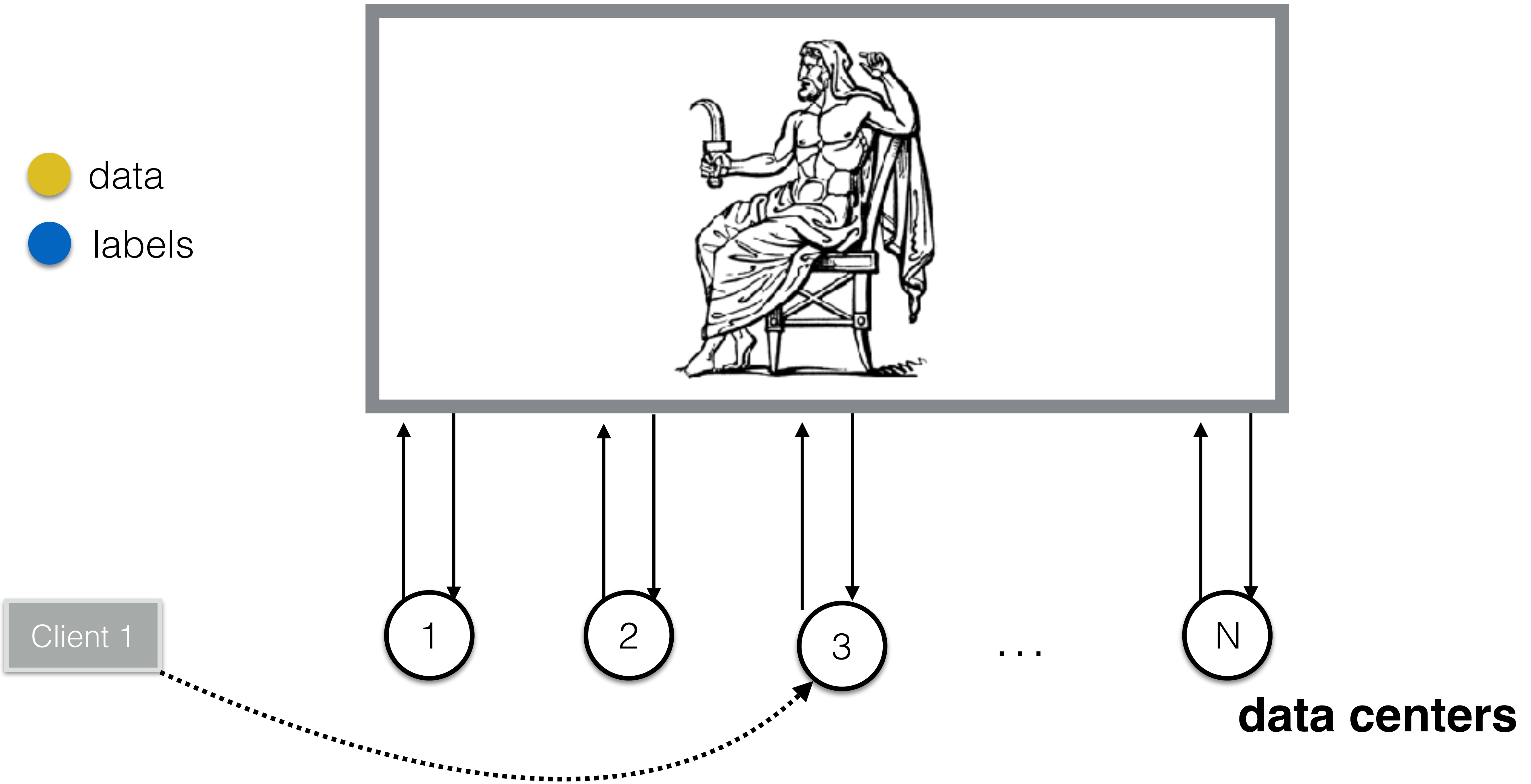
labels



Client 1

data centers

Example: migration

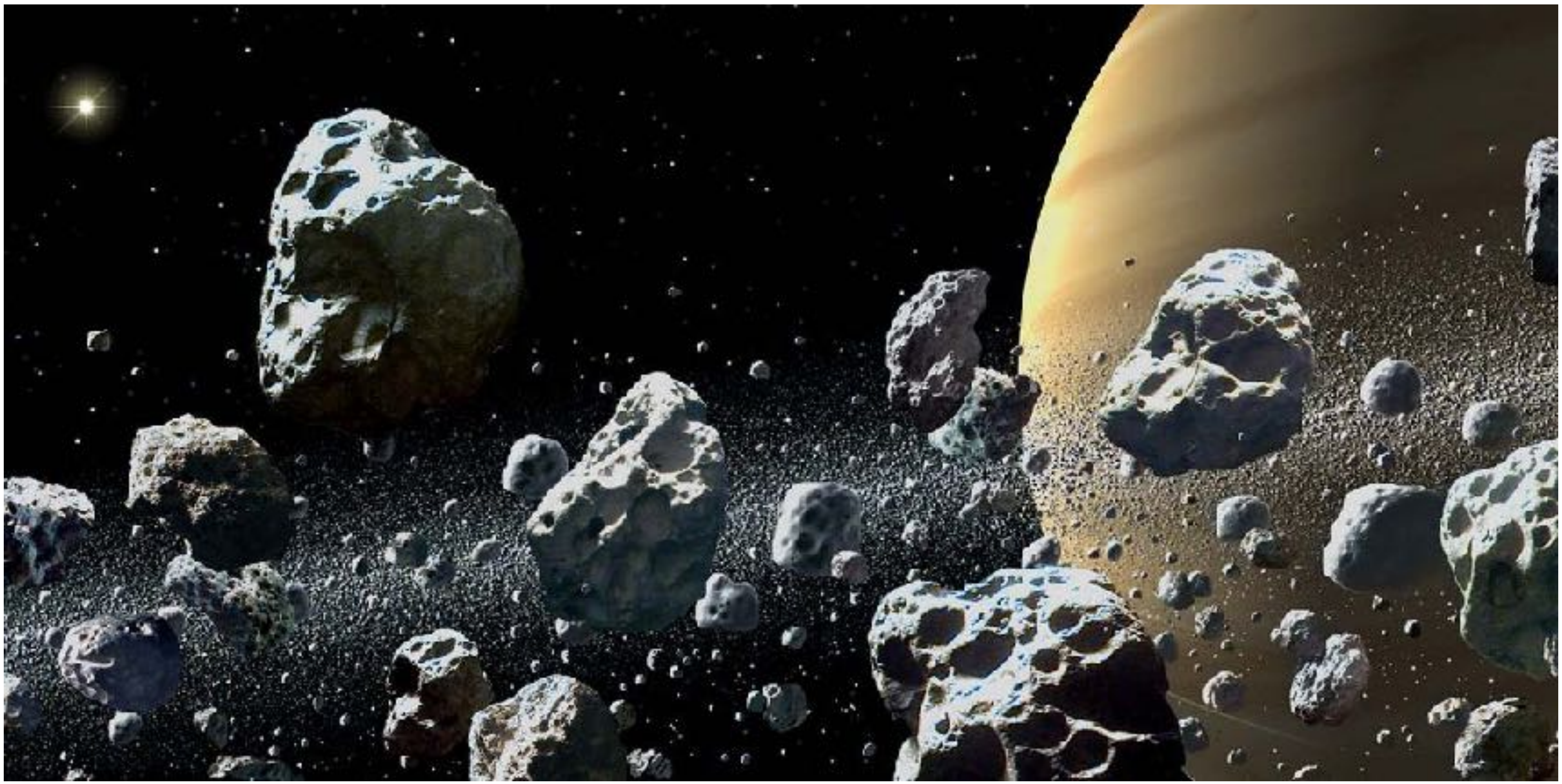


Saturn on the edge

Challenges

- Many nodes:
  - **Optimal tree may be expensive to build**
- Cloudlets are smaller than datacenters:
  - **Migration will be more frequent**

The Saturn Rings

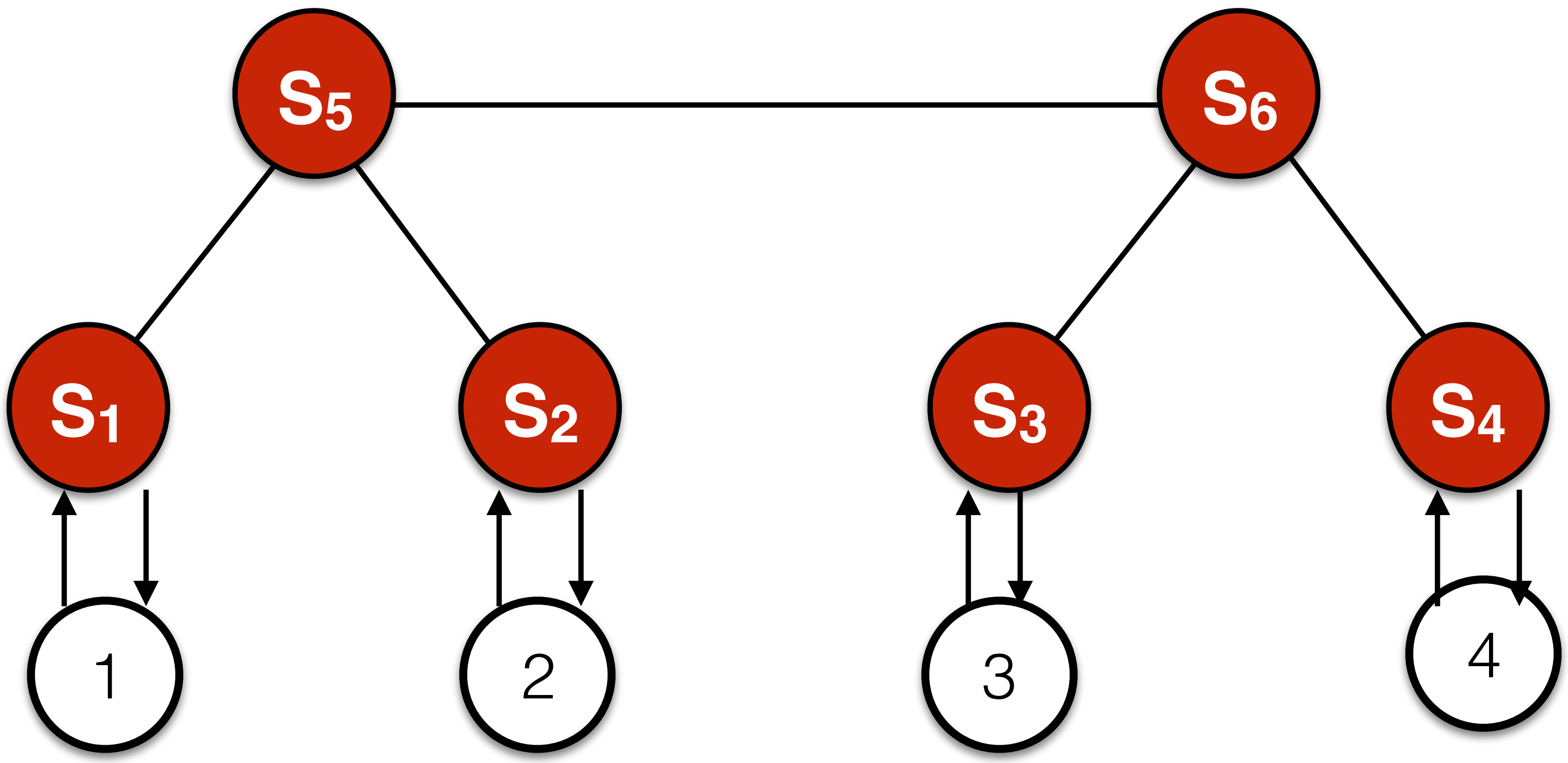


Let's assume that each cloudlet stores a subset of the data maintained by
a **single** datacenter

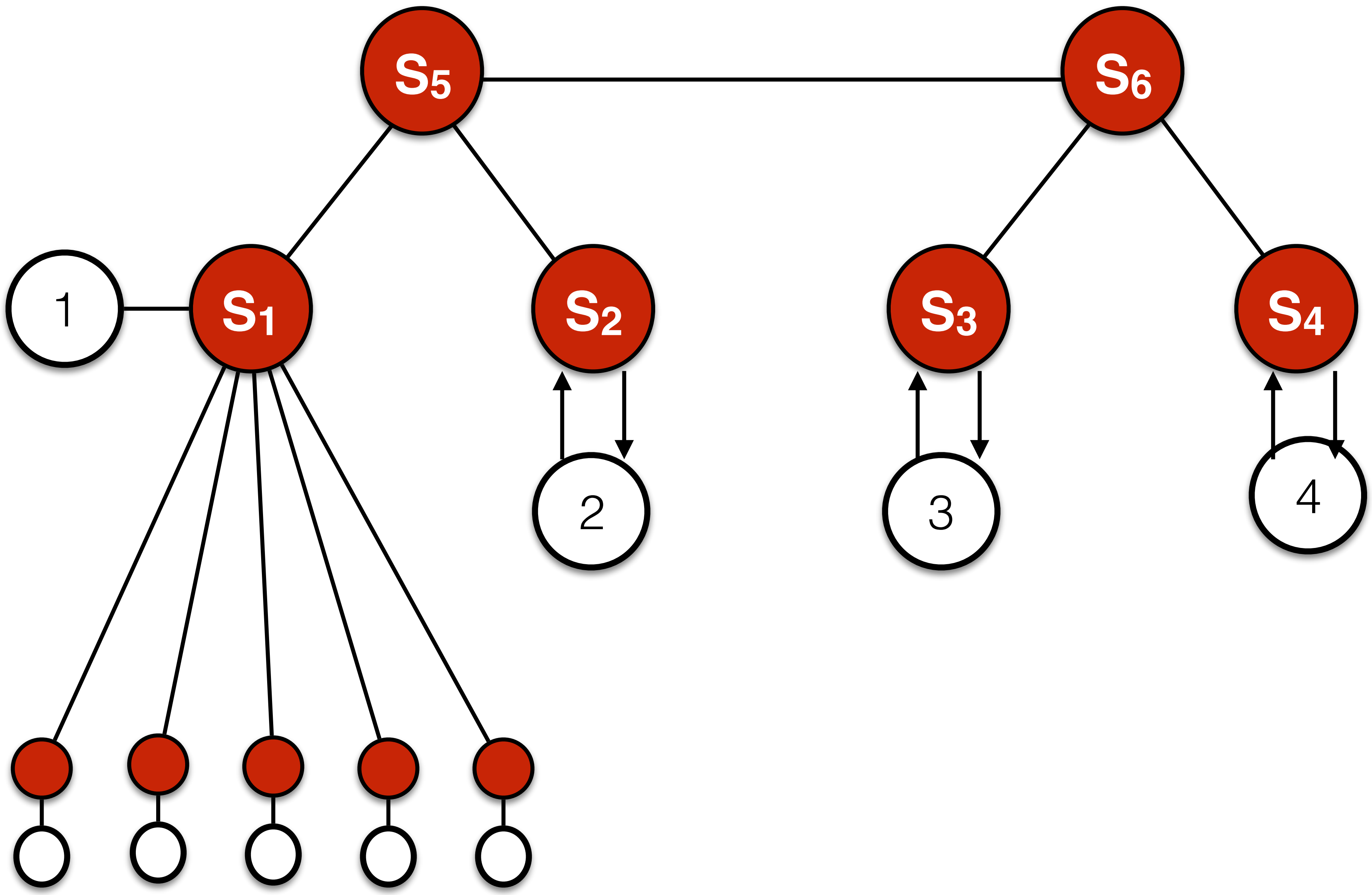
Let's assume that each cloudlet stores a subset of the data maintained by
a **single** datacenter

That datacenter is named the cloudlet's **ancestor**

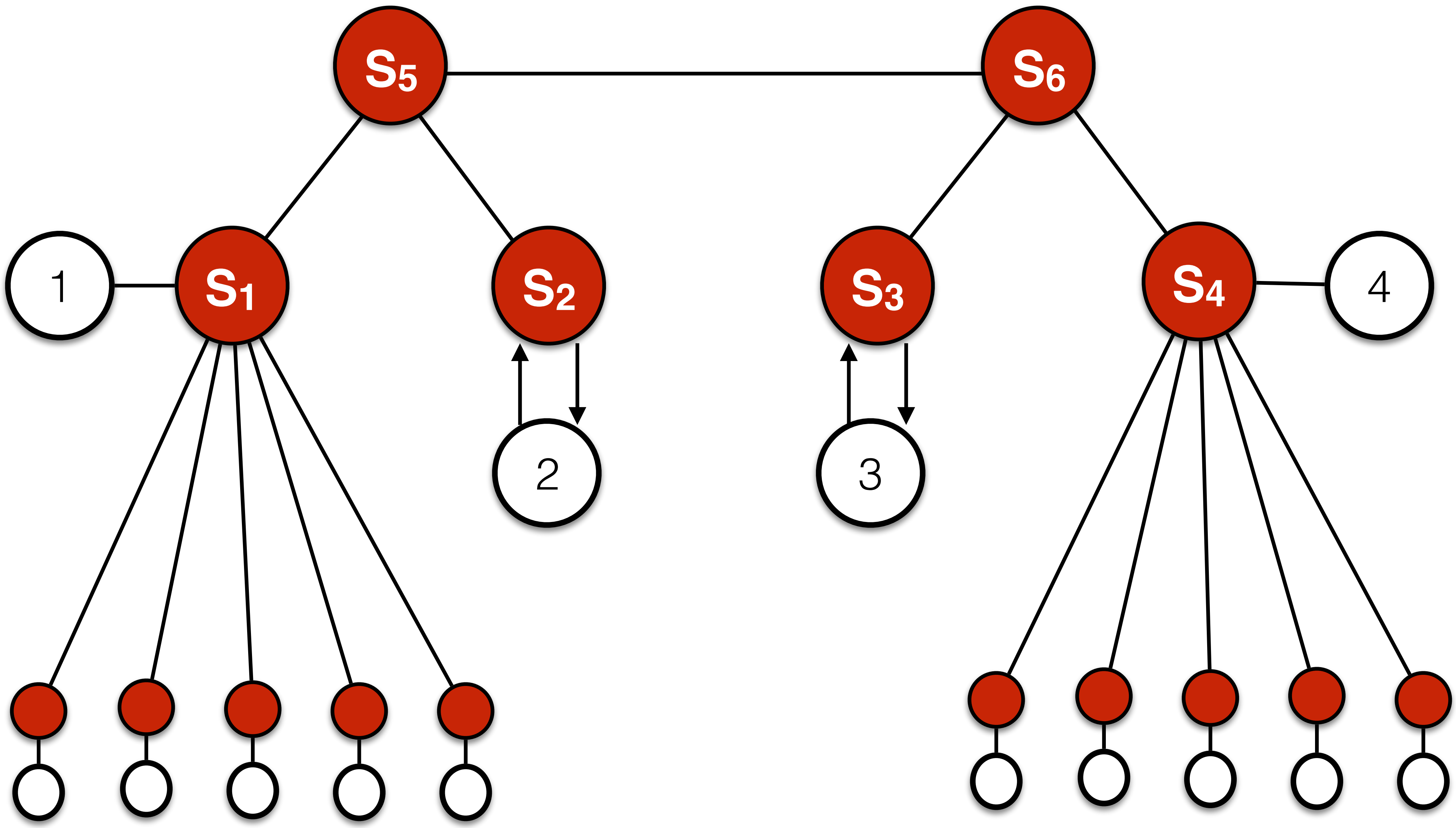
The Saturn metadata tree is extended with a star of cloudlets connected to each datacenter



S5
S6
S1
S2
S3
S4
1
2
3
4



S5
S6
1
S1
S2
S3
S4
2
3
4


S_5
S_6
1
S_1
S_2
S_3
S_4
4
2
3

This topology allows us to implement
fast migration strategies

Fast Migration

Clients connect to the nearest cloudlet and obtain labels from the cloudlet when reading/writing data:

If a request cannot be served from the cloudlet they perform a fast migration to the datacenter (**ascending fast migration**).

Clients can later do a fast migration back to their local cloudlet to continue to be served locally (**descending fast migration**)

Fast Migration

Ascending fast migration:

Descending fast migration:

Fast Migration

Ascending fast migration:

Client simply presents its label (obtained from the cloudlet) to the datacenter and blocks until the datacenter is synced with the cloudlet.

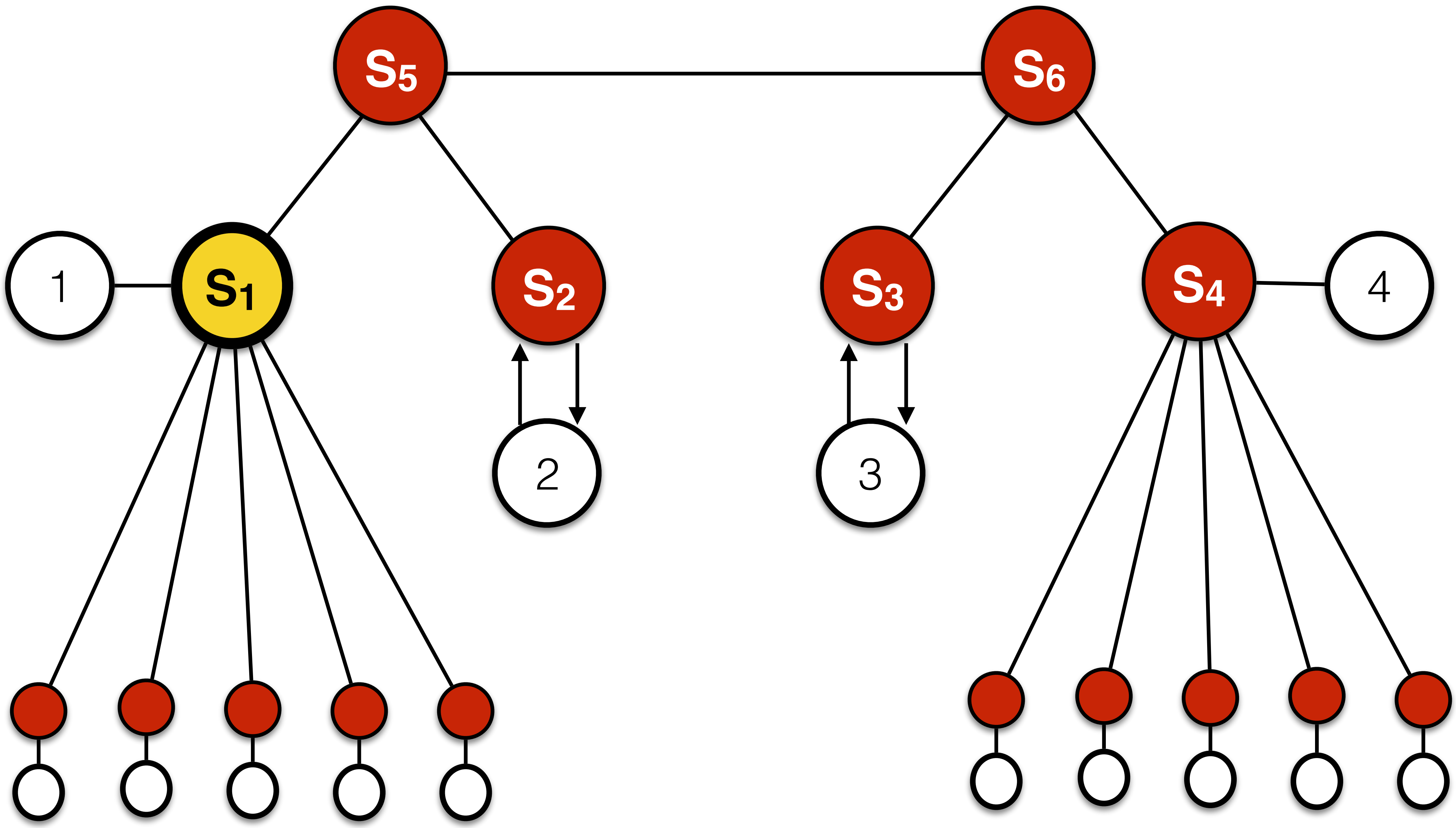
Descending fast migration:

Fast Migration

Ascending fast migration:

Descending fast migration:

Need a little help from the Saturn brokers…


S5
S6
1
S1
S2
S3
S4
4
2
3

Fast Migration

The operation of a Saturn broker is extended as follows:

When a broker ships a label to a datacenter
it immediately schedules that label for transmission
to the relevant cloudlet.

The broker keeps a vector with the
Last Dispatched Label (LDL)
schedule to be sent to each cloudlet **c**
LDL[c]

Fast Migration

Ascending fast migration:

Descending fast migration:

Client obtains the last dispatched label (LDL) from the datacenter to its own cloudlet, presents the LDL to the cloudlet and waits for the cloudlet to be synced with the datacenter.

Fast Migration

Migrations among siblings cloudlets:

Migration to remote datacenters/ cloudlets:

Fast Migration

Migrations among siblings cloudlets:

Requires a read operation on the ancestor datacenter.

Migration to remote datacenters/ cloudlets:

Fast Migration

Migrations among siblings cloudlets:

Migration to remote datacenters/ cloudlets:

Uses the default Saturn mechanism

Provides efficient metadata management to support causality on the edged (cloudlets).

In worst case, only two labels need to be maintained by clients: a data label (used for reads/writes) and a LDL label used fro fast descending migrations.