UNOBTRUSIVE DEFERRED UPDATE STABILIZATION FOR EFFICIENT GEO-REPLICATION

Chathuri Gunawardhana, Manuel Bravo, **Luís Rodrigues**

inesc id lisboa

TÉCNICO LISBOA

UCL Université catholique de Louvain

Problem

In a geo-replicated system,
how to apply **efficiently,**
remote **updates,**
in **causal** order?

Why causal consistency?

Limitations of Highly-Available Eventually-Consistent Data Stores

Hagit Attiya
Computer Science Department
Technion

Faith Ellen
Department of Computer Science
University of Toronto

Adam Morrison
Computer Science Department
Technion

ABSTRACT

Modern *replicated data stores* aim to provide high availability, by immediately responding to client requests, often by implementing objects that expose concurrency. Such objects, for example, multi-valued registers (MVRs), do not have sequential specifications. This paper explores a recent model for replicated data stores that can be used to precisely specify *causal consistency* for such objects, and liveness properties like *eventual consistency*, without revealing details of the underlying implementation. The model is used to prove the following results:

- An eventually consistent data store implementing MVRs cannot satisfy a consistency model strictly stronger than *observable causal consistency* ($\mathcal{OCC}$). $\mathcal{OCC}$ is a model somewhat stronger than causal consistency, which captures executions in which client observations can use causality to infer concurrency of operations. This result holds under certain assumptions about the data store.
- Under the same assumptions, an eventually consistent and causally consistent replicated data store must send messages of unbounded size: If s objects are supported by n replicas, then, for every $k > 1$, there is an execution in which an $\Omega(\min\{n, s\}k)$-bit message is sent.

of data (i.e. accesses to data return without delay), and its *consistency*, while tolerating message delays. The *CAP theorem* [8,18] demonstrates the difficulty of achieving this balance, showing that strong *C*onsistency (i.e. *atomicity*) cannot be satisfied together with high *A*vailability and *P*artition tolerance.

One aspect of data consistency is a safety property restricting the possible values observed by clients accessing different replicas. The set of possible executions is called a *consistency model*. For example, *causal consistency* [2] ensures that the causes of an operation are visible at a replica no later than the operation itself. (A precise definition appears in Section 3.) A smaller set of possible executions means that there is less uncertainty about the data. So, one consistency model is *strictly stronger* than another if its executions are a proper subset of the executions of the other.

Some weak consistency models can be trivially satisfied by never updating the data. Therefore, another aspect of data consistency is a liveness property, ensuring that updates are applied at all replicas. The designers of many systems, e.g., Dynamo [13] and Cassandra [1], opt for a very weak liveness property called, somewhat confusingly, *eventual consistency* [5,9,10,29]. Eventual consistency only ensures that each replica eventually observes all updates to the object, a property also referred to as *update propagation* [11].

Strongest without compromising availability

Why causal consistency?

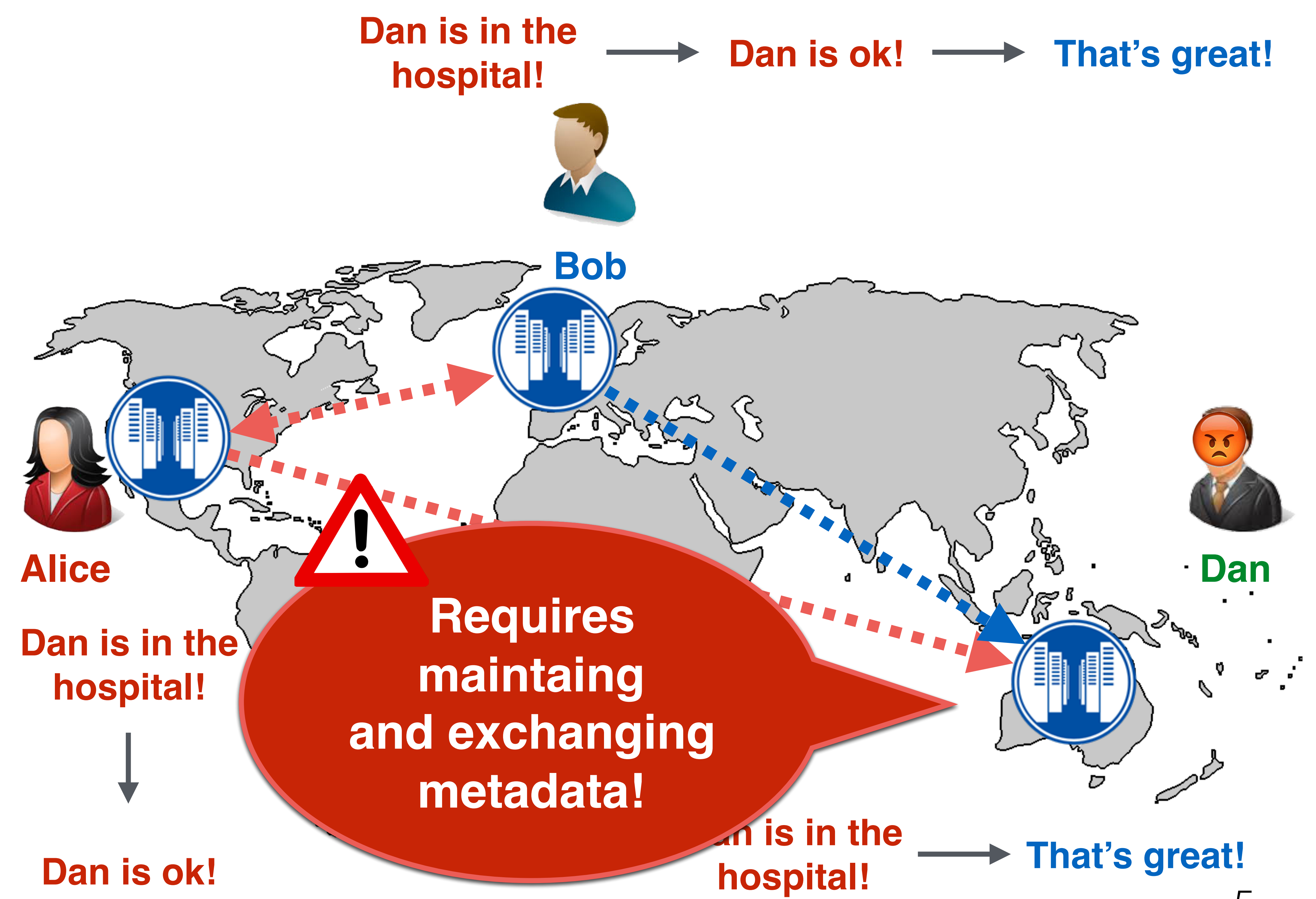
Key ingredient
of several
consistency
criteria

Parallel Snapshot Isolation
[SOSP'11]

RedBlue Consistency
[OSDI'12]

Explicit Consistency
[EuroSys'15]

Session guarantees
[SOSP'97]


Dan is in the hospital!
Dan is ok!
That's great!
Bob
Alice
Dan
Dan is in the hospital!
Dan is ok!
Requires maintaing and exchanging metadata!
an is in the hospital!
That's great!

Metadata

Needs a way to compress metadata!

more metadata | less metadata

precise | **false positives**

expensive | **cheap**

Metadata compression

Two main ways to compress and manage metadata

Global stabilization procedures

Serializers

Metadata compression

Global stabilization procedures

Updates are propagated concurrently
and later ordered at remote datacenters

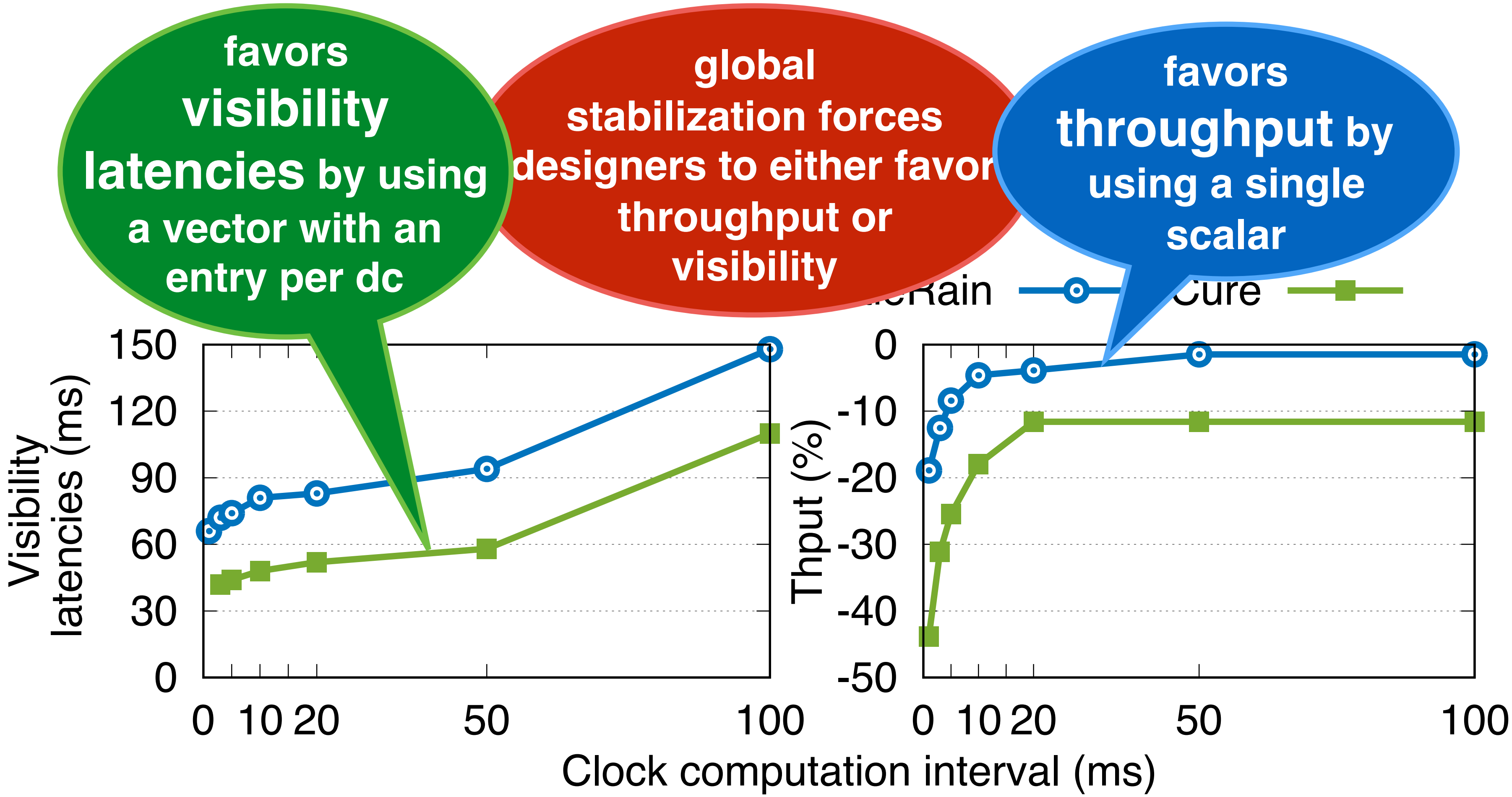
Metadata compression

Serializers

Updates are ordered before being applied
at the origin datacenter

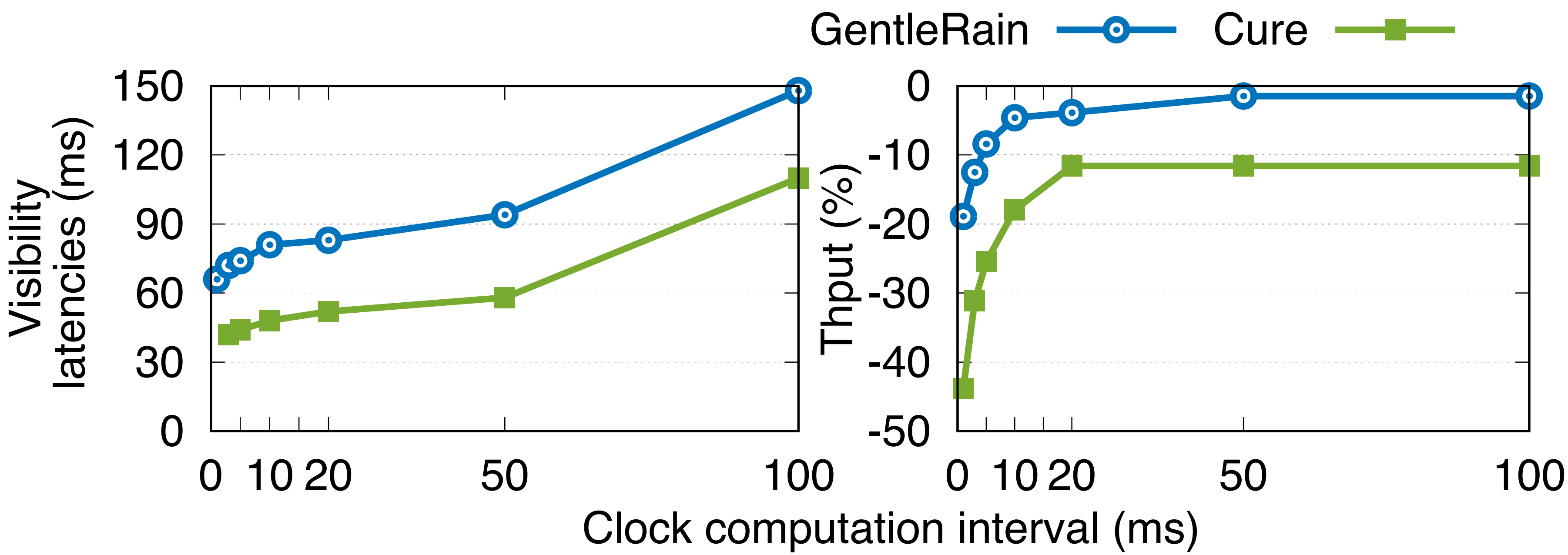
Metadata compression

GentleRain and **Cure** use **global stabilization**!



Metadata compression

Serializer (typically one per dc): SwiftCloud [Middleware'15], ChainReaction [EuroSys'13], …



Metadata compression

Serializer (typica… …Cloud [Middleware'15],

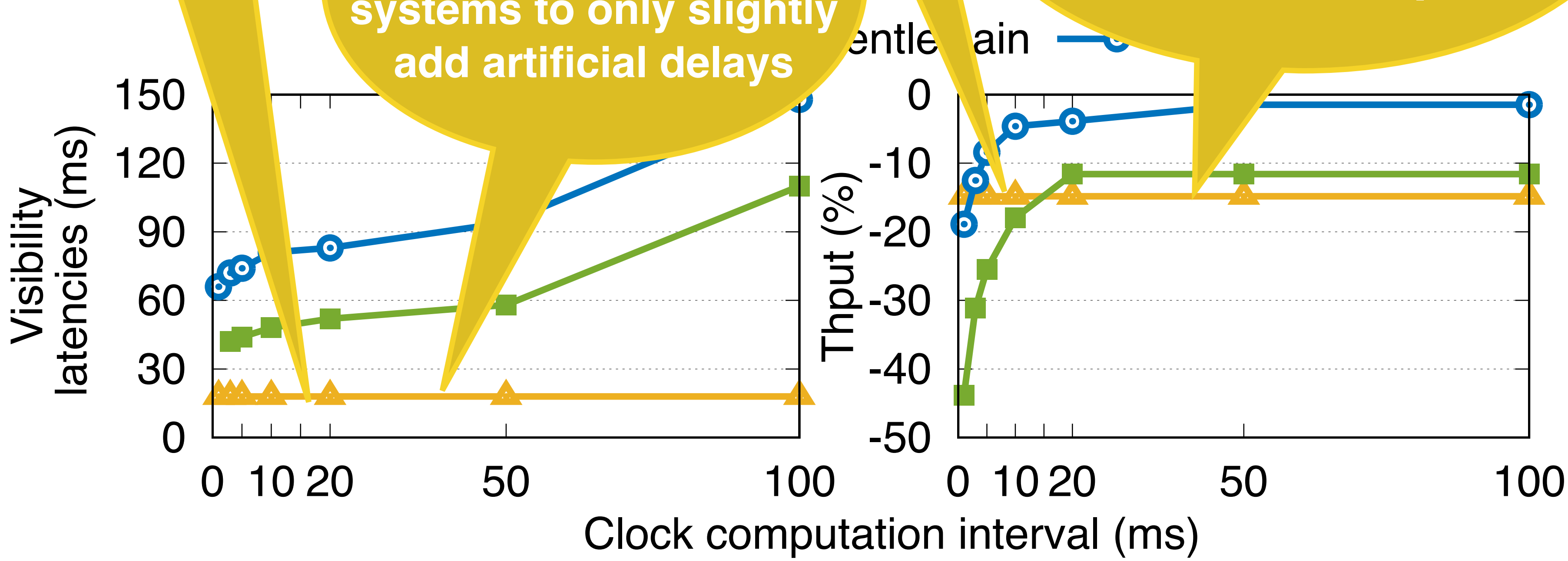
Ch… …ion [EuroSy…

abstracts the complexity of partitioned datacenters, which enables trivial dependency checking

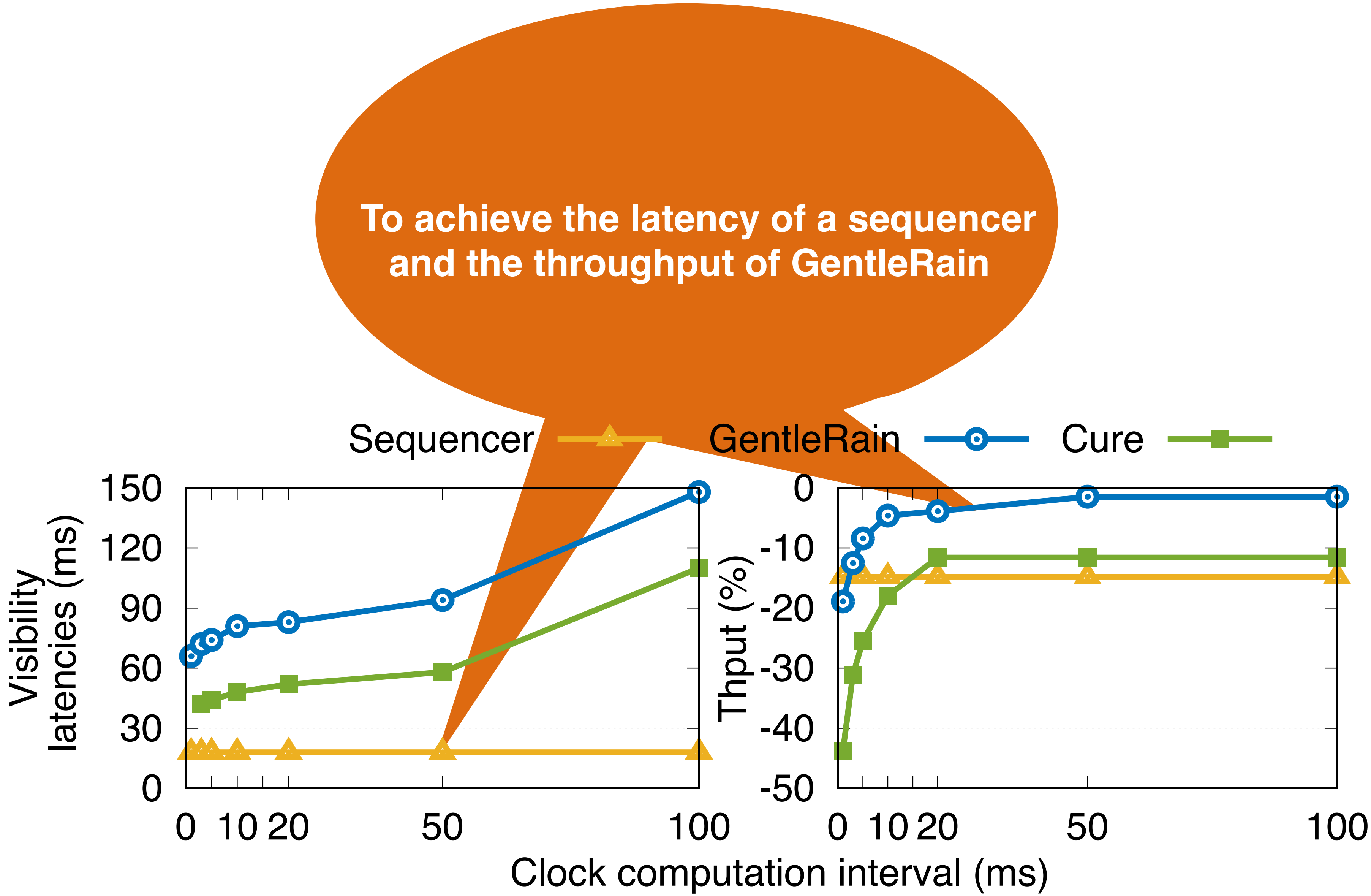
unfortunately, coordination is on the client's critical path, which significantly penalises throughput (16%)

thus, allowing sequencer-based systems to only slightly add artificial delays

an sequencers can easily be overloaded for medium-size clusters. In our experiments, a **max of 48 kops/s**



Our goal


To achieve the latency of a sequencer
and the throughput of GentleRain
Sequencer
GentleRain
Cure
Visibility latencies (ms)
150
120
90
60
30
0
0
10
20
50
100
Thput (%)
0
-10
-20
-30
-40
-50
0
10
20
50
100
Clock computation interval (ms)

Eunomia

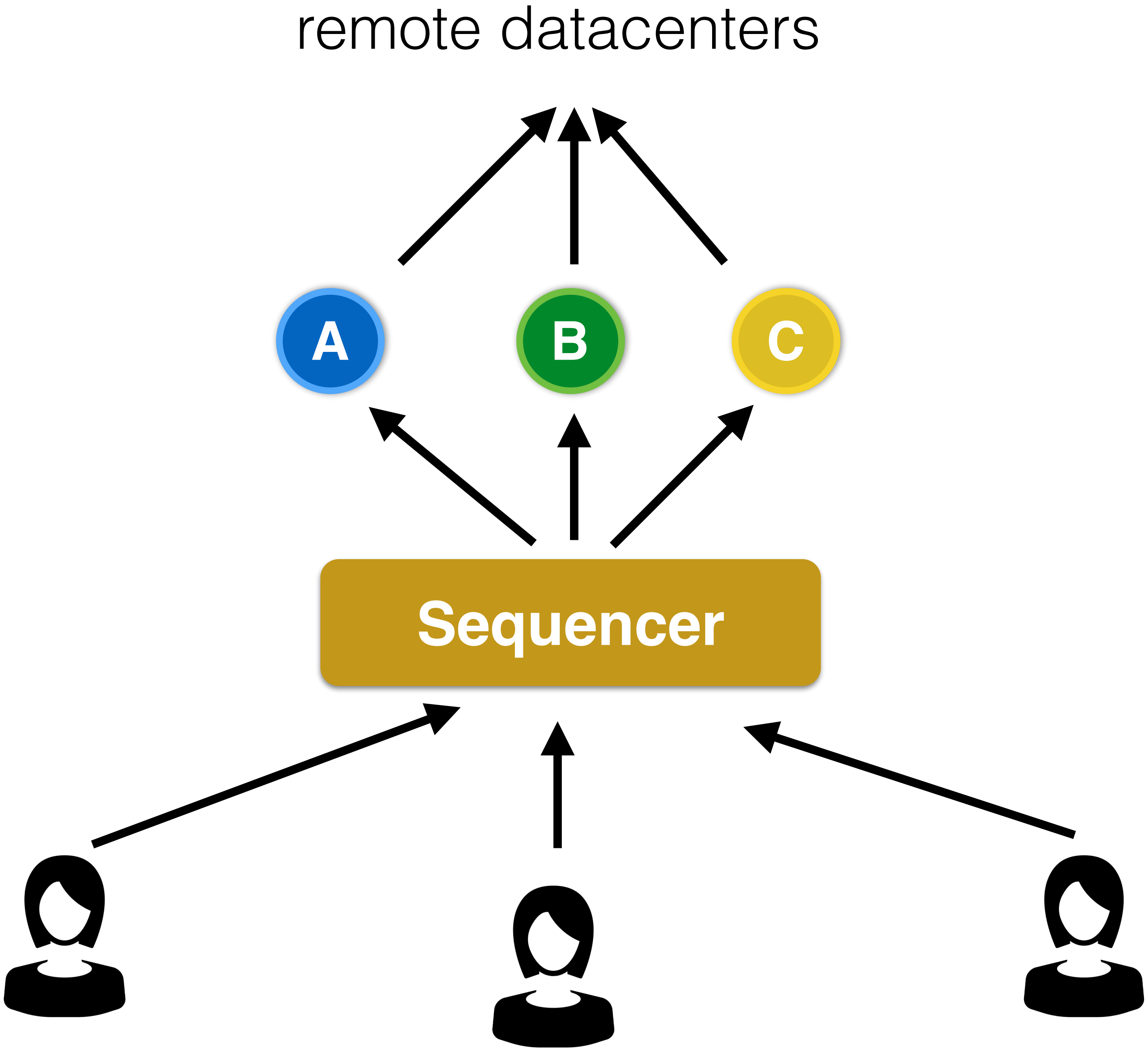
aims at finding a **novel** way of compressing metadata that allows to pick a **better spot** in the throughput-visibility tradeoff

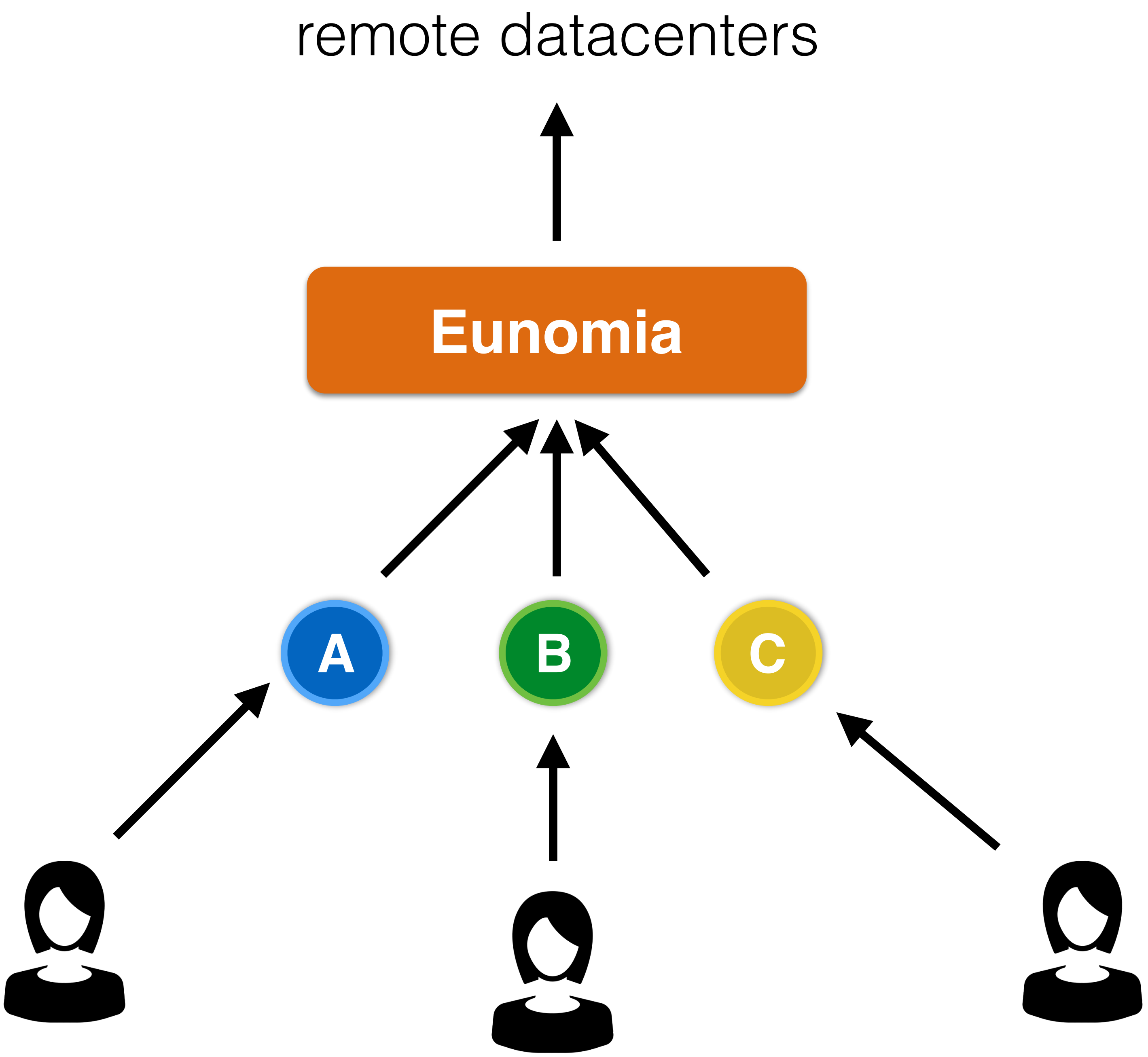
Eunomia

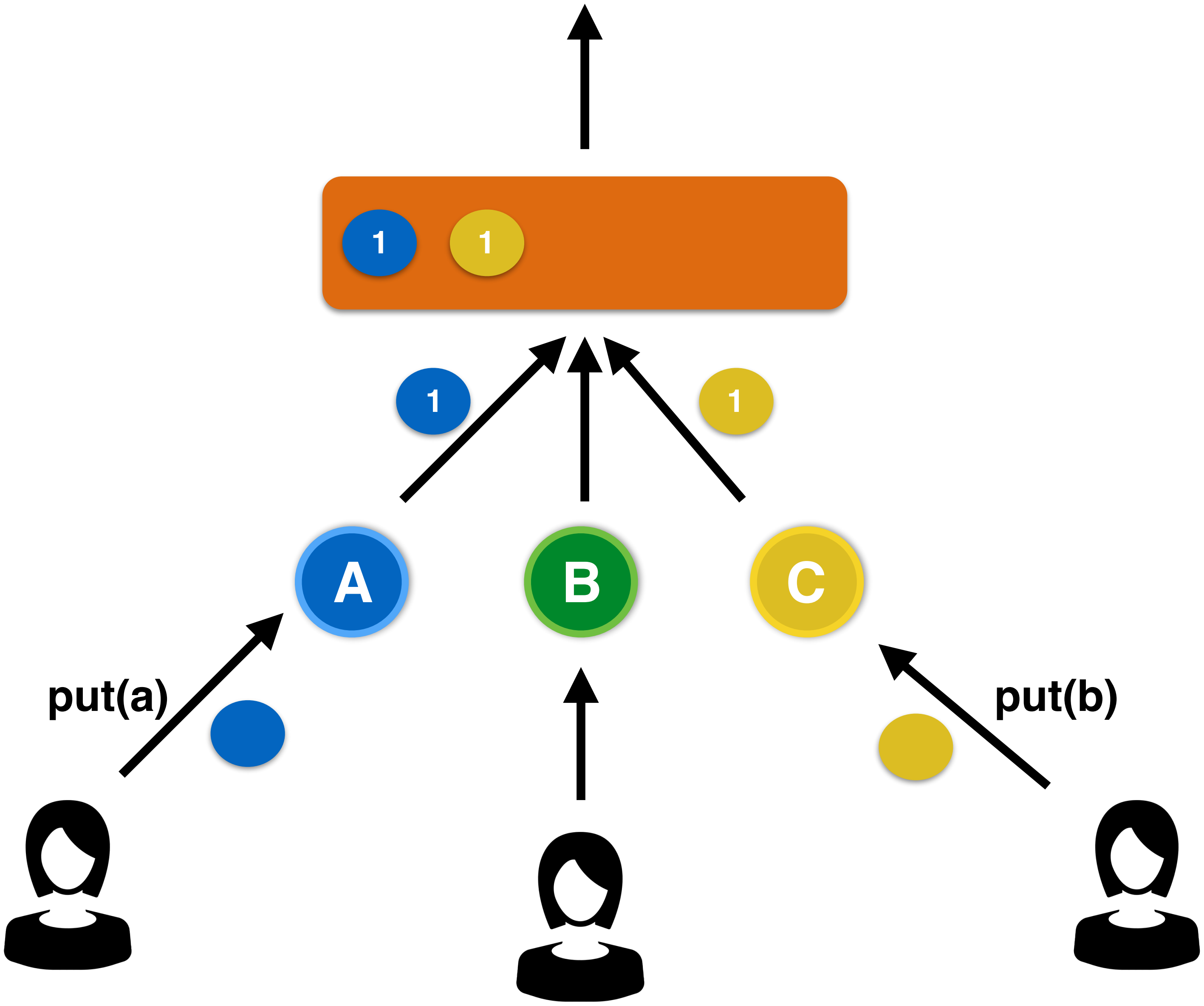
conceived to **replace sequencers** in geo-replicated storage systems

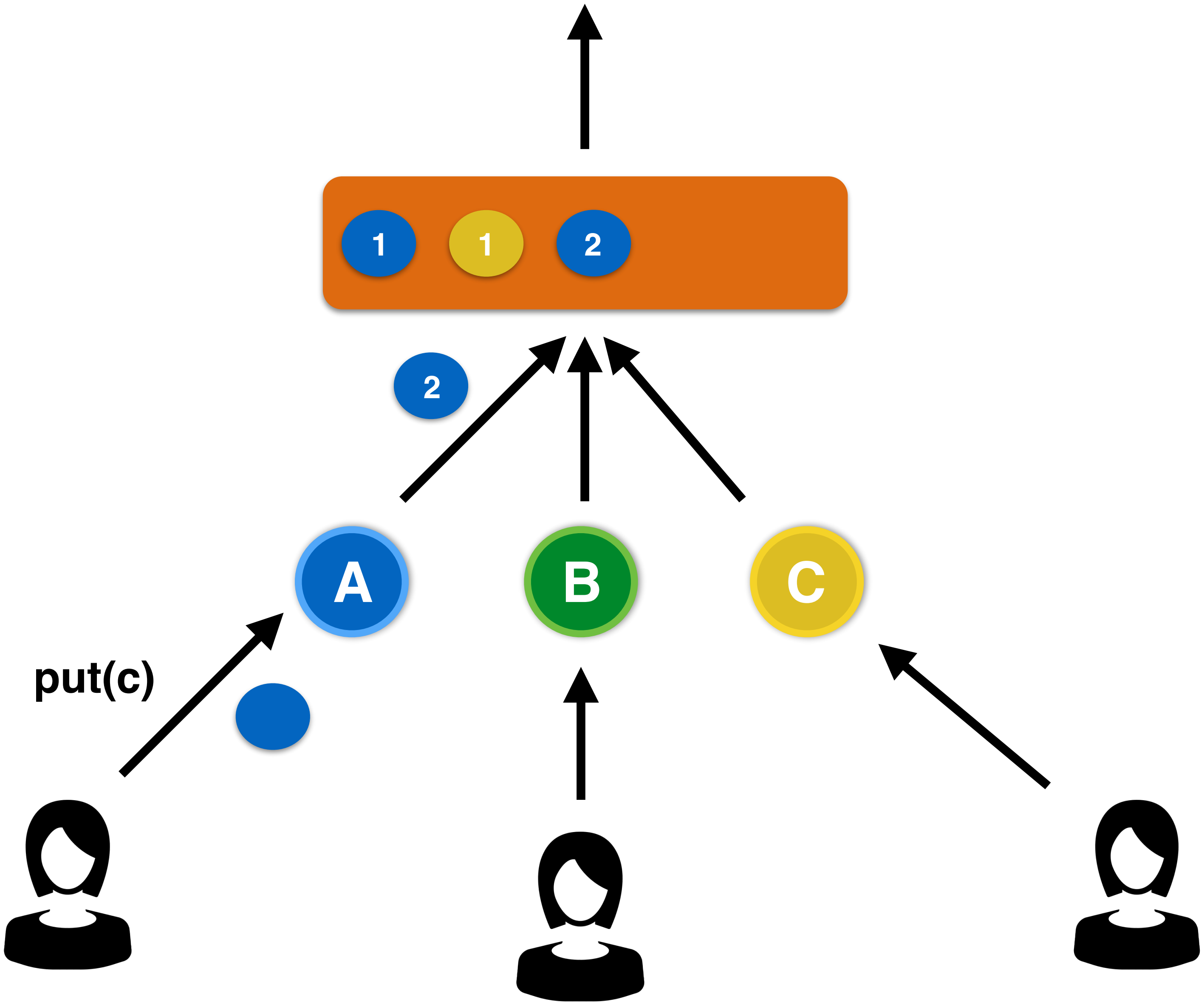
totally orders—consistently with causality—local updates, before shipping them to other dcs

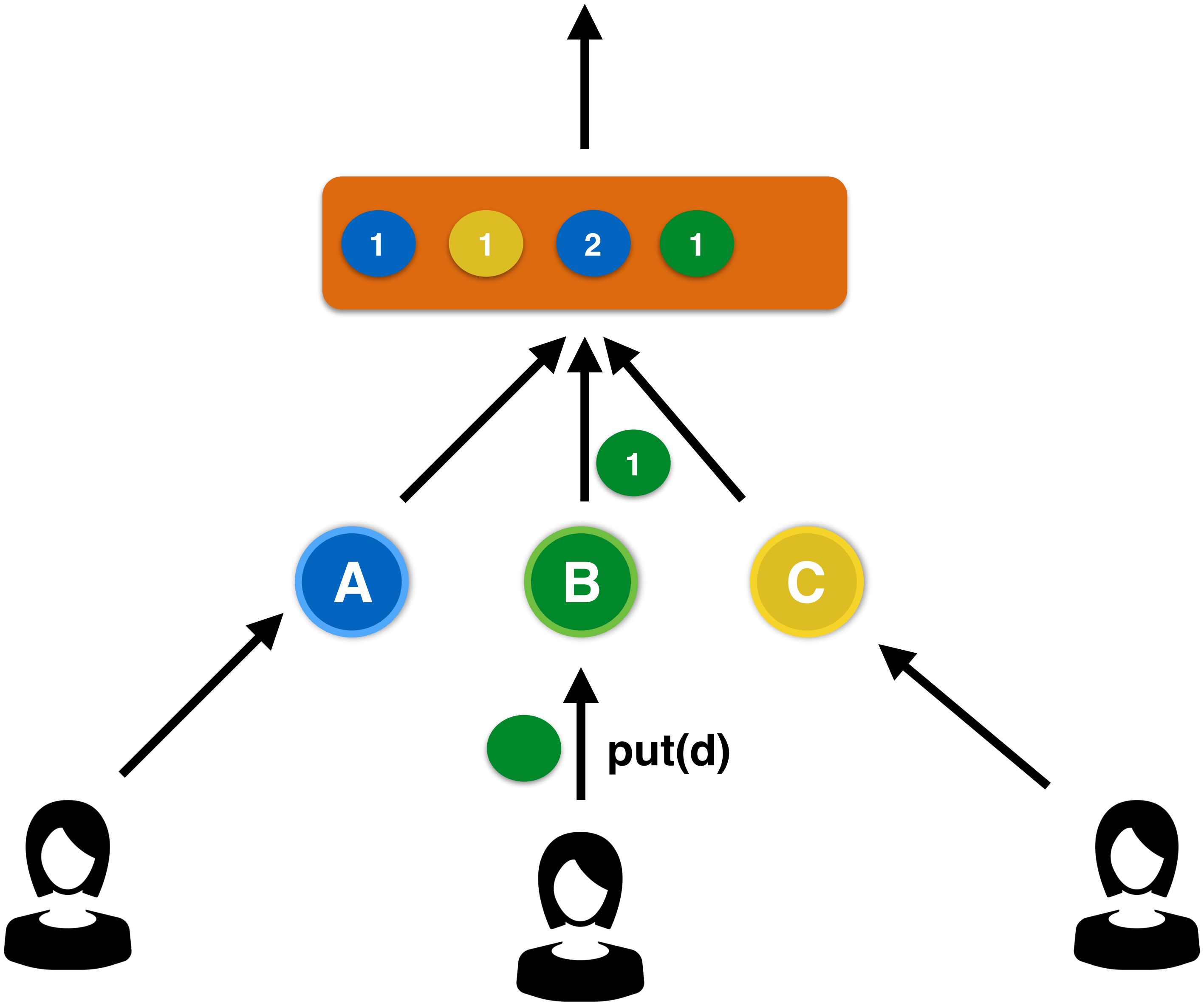
the ordering is done **in the background**, out of client's critical path

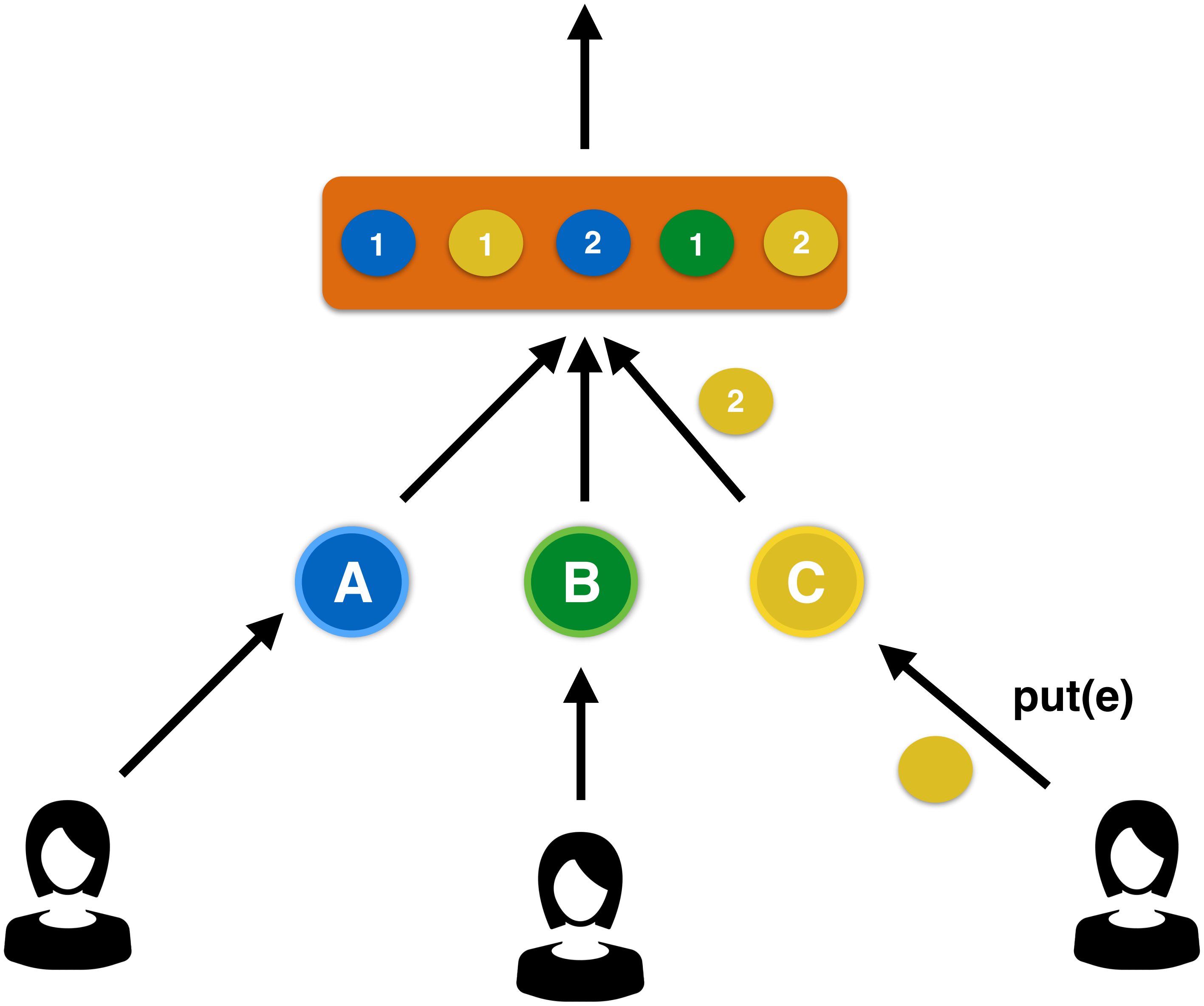

remote datacenters
A
B
C
Sequencer

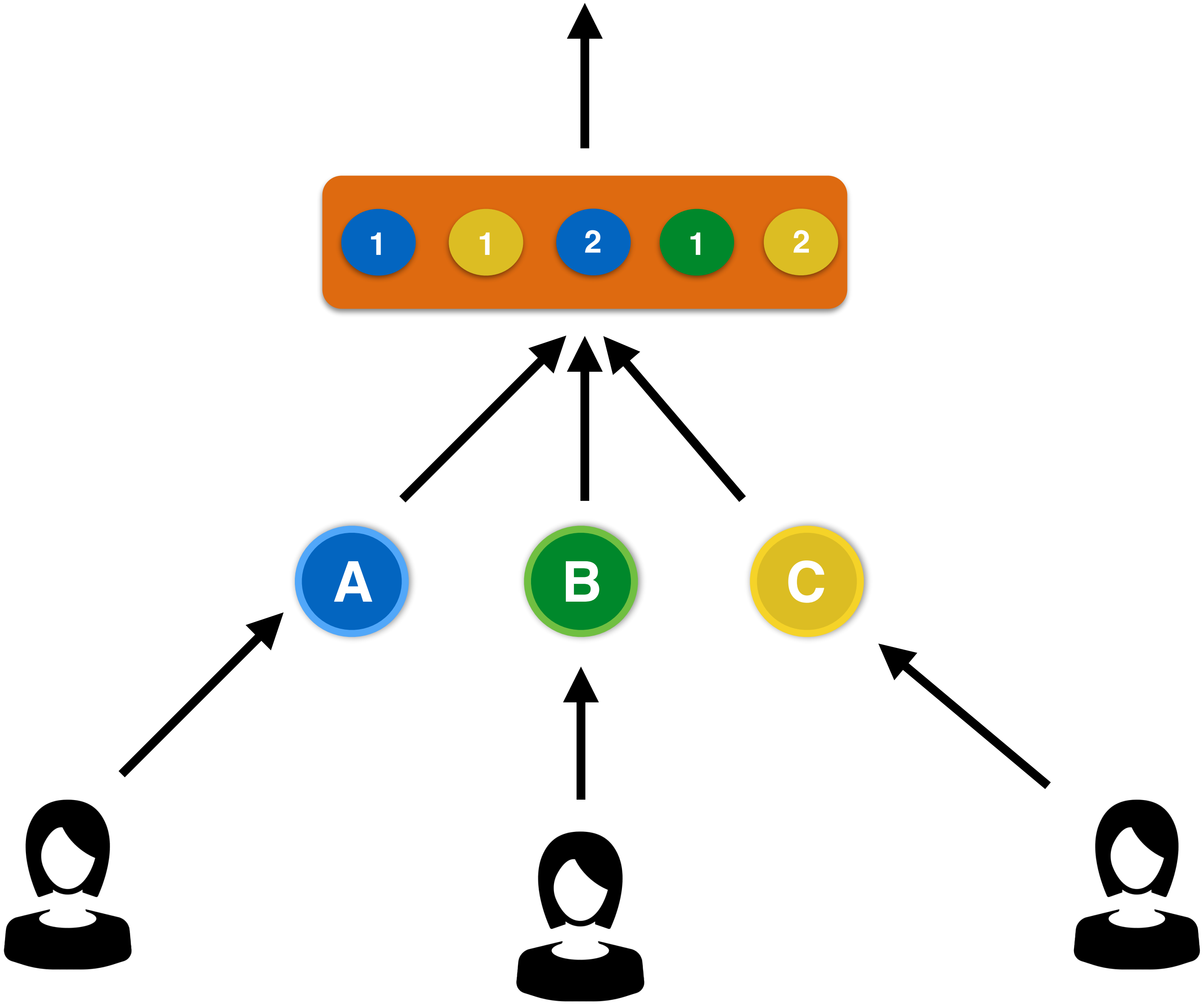

remote datacenters
Eunomia
A
B
C

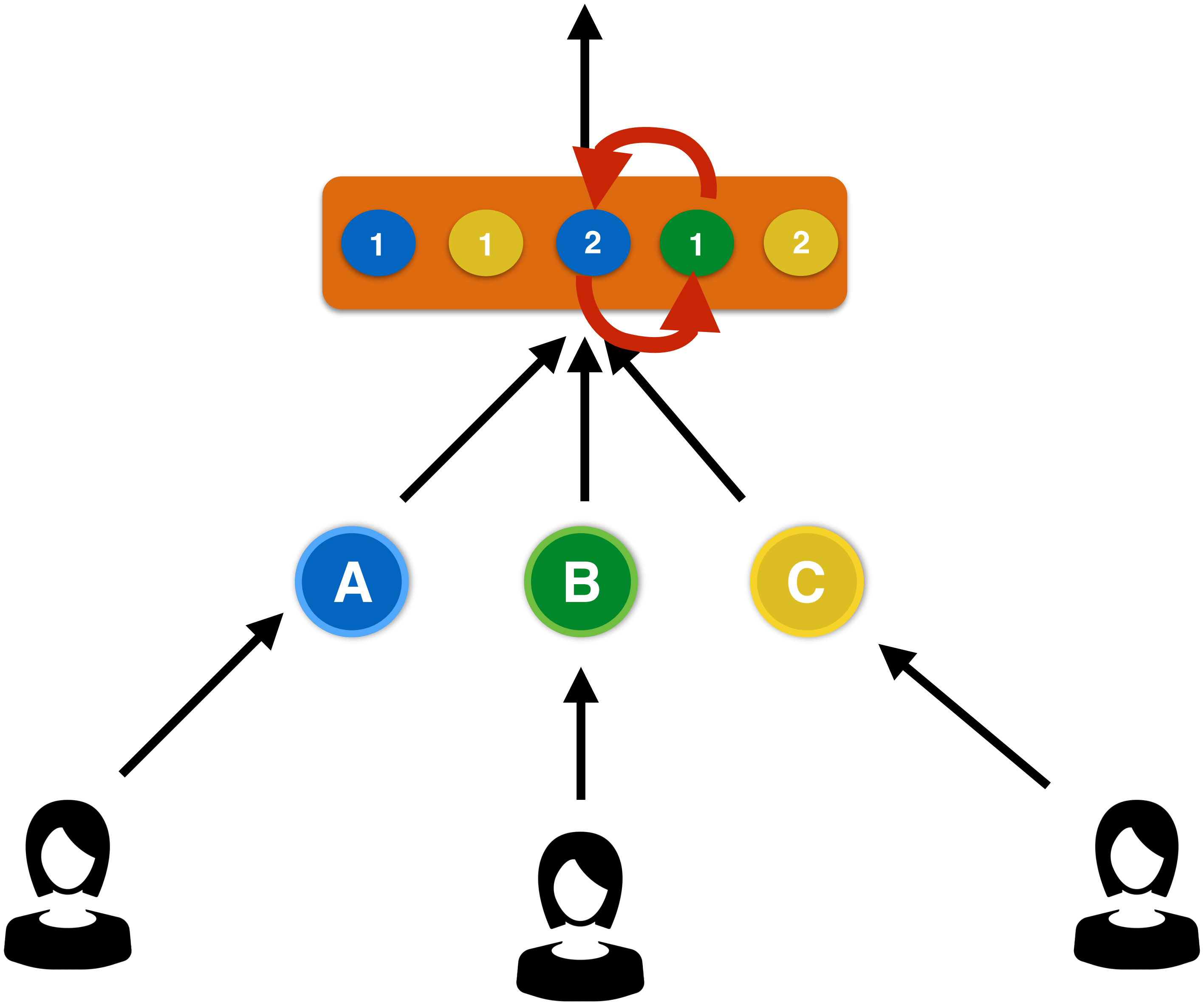

1
1
1
1
A
B
C
put(a)
put(b)

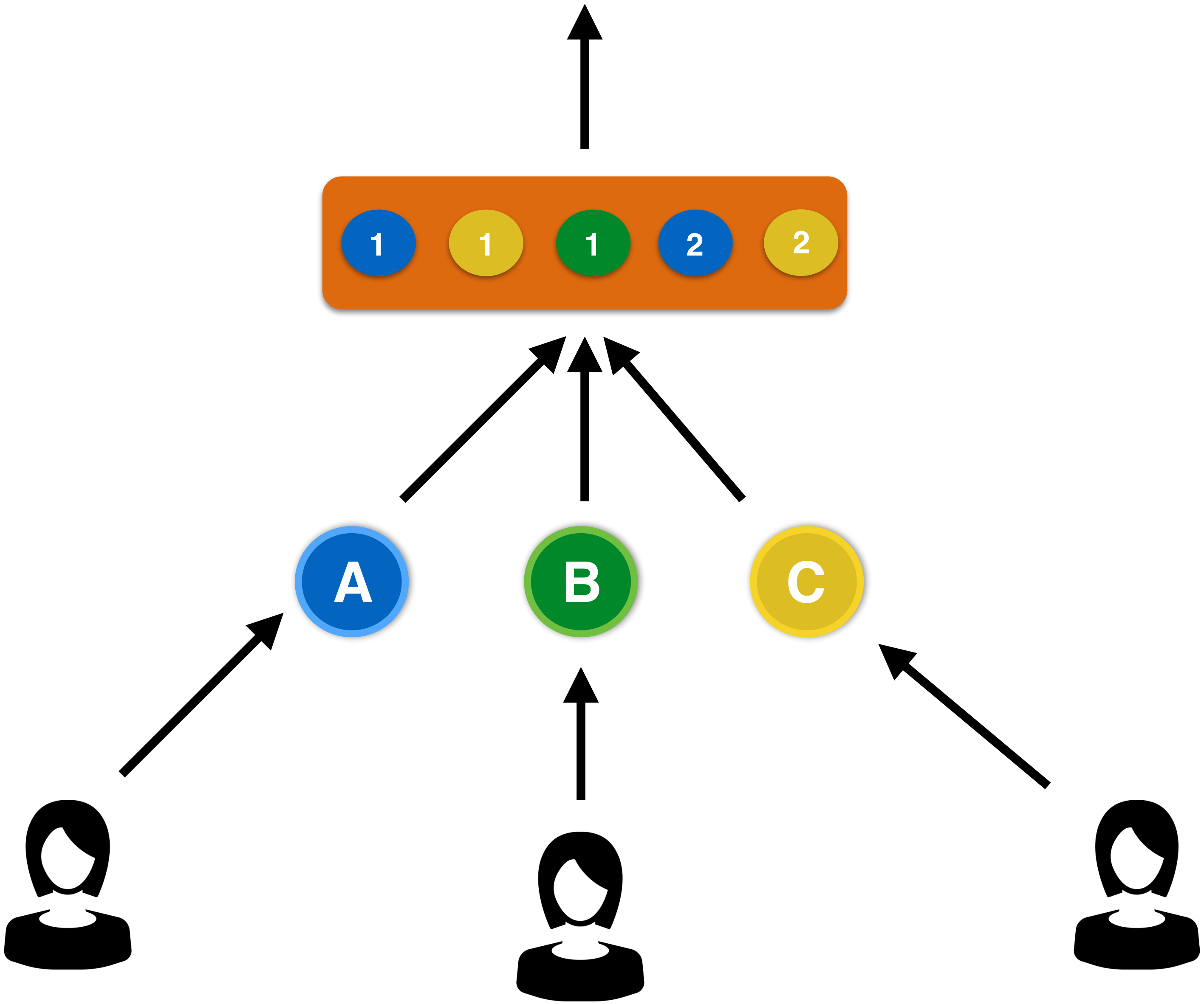

1
1
2
2
A
B
C
put(c)

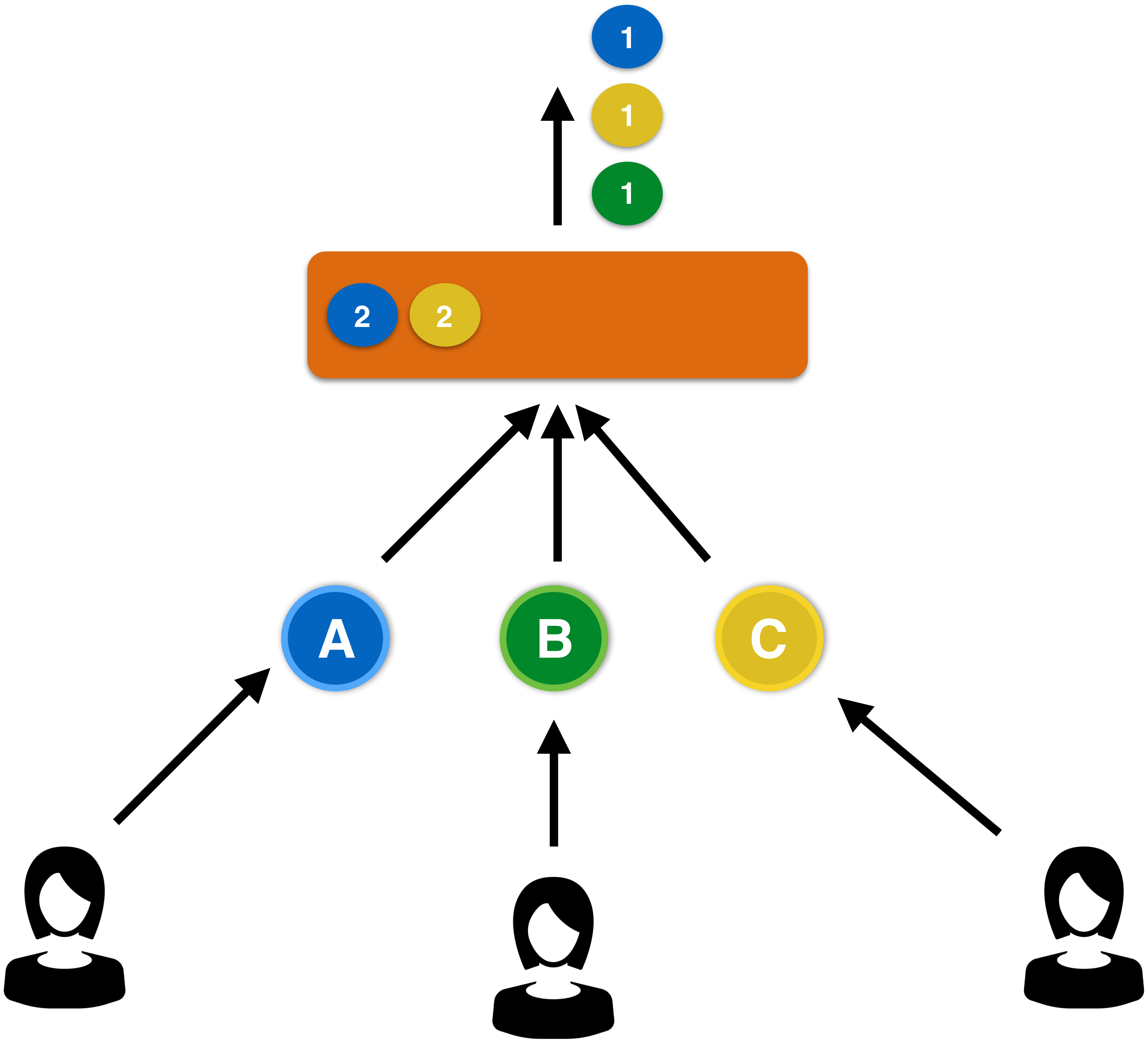

1
1
2
1
1
A
B
C
put(d)

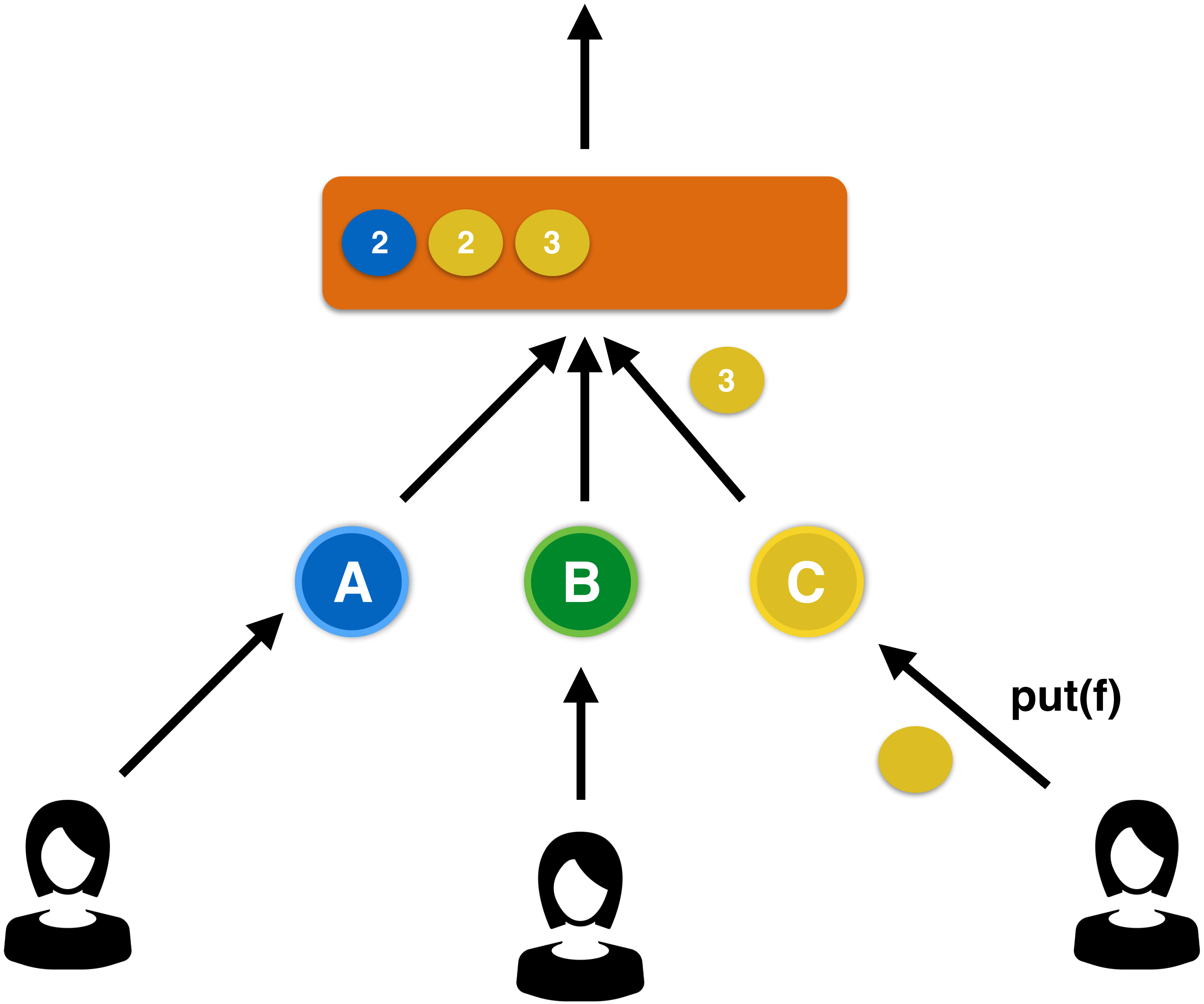

1
1
2
1
2
2
A
B
C
put(e)


1
1
2
1
2
A
B
C

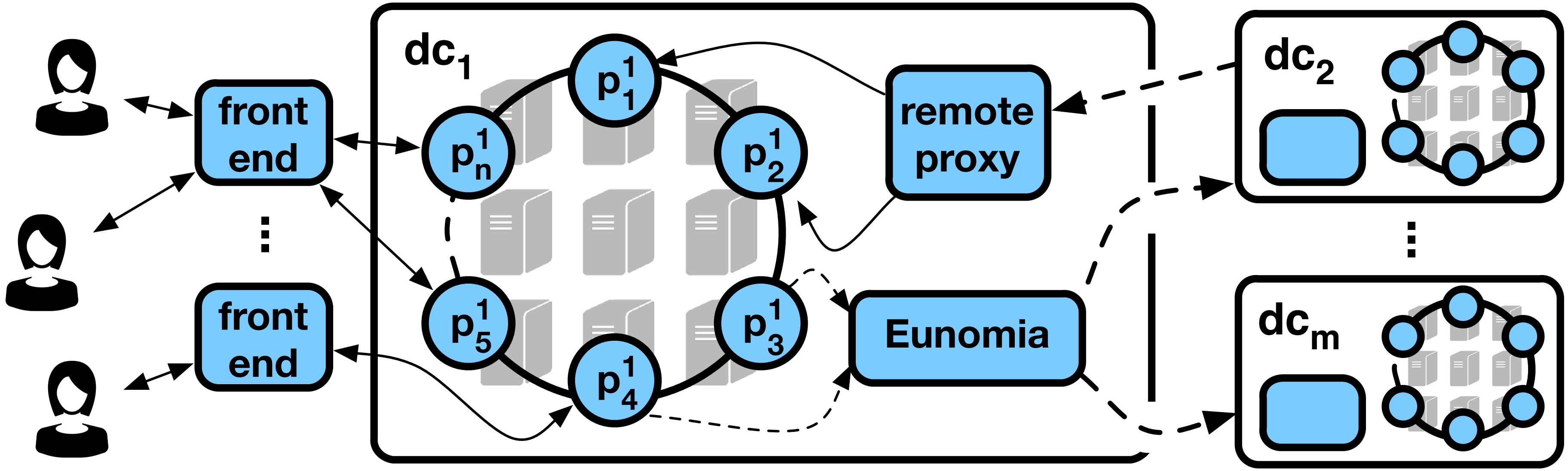

1
1
2
1
2
A
B
C


1
1
1
2
2
A
B
C

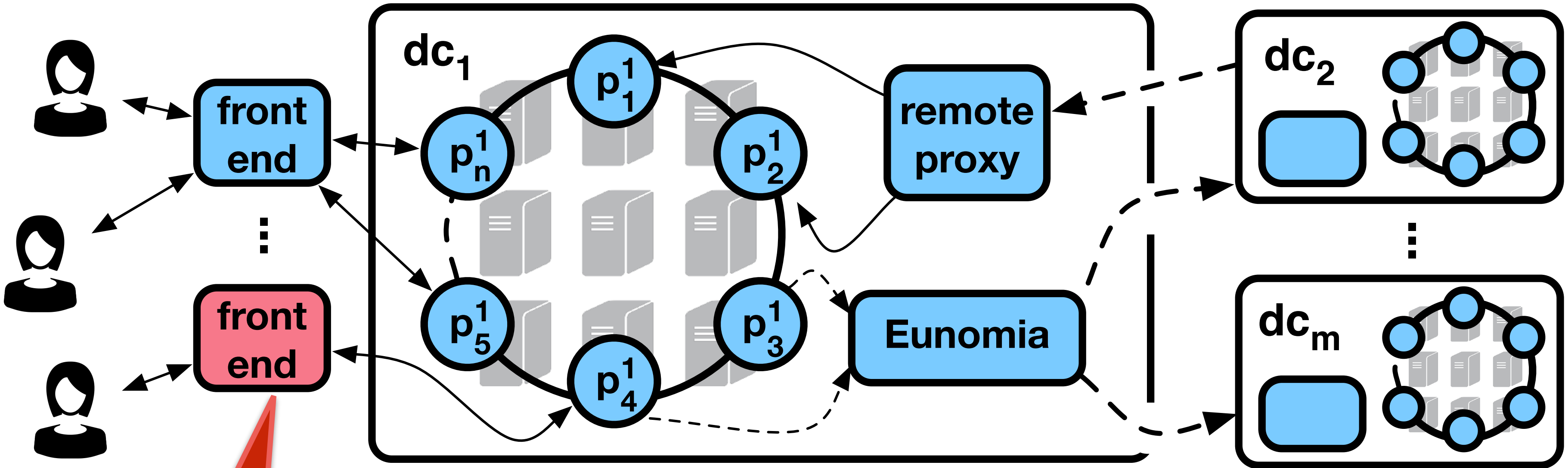

1
1
1
2
2
A
B
C



2
2
3
3
A
B
C
put(f)

EunomiaKV: supporting geo-replication

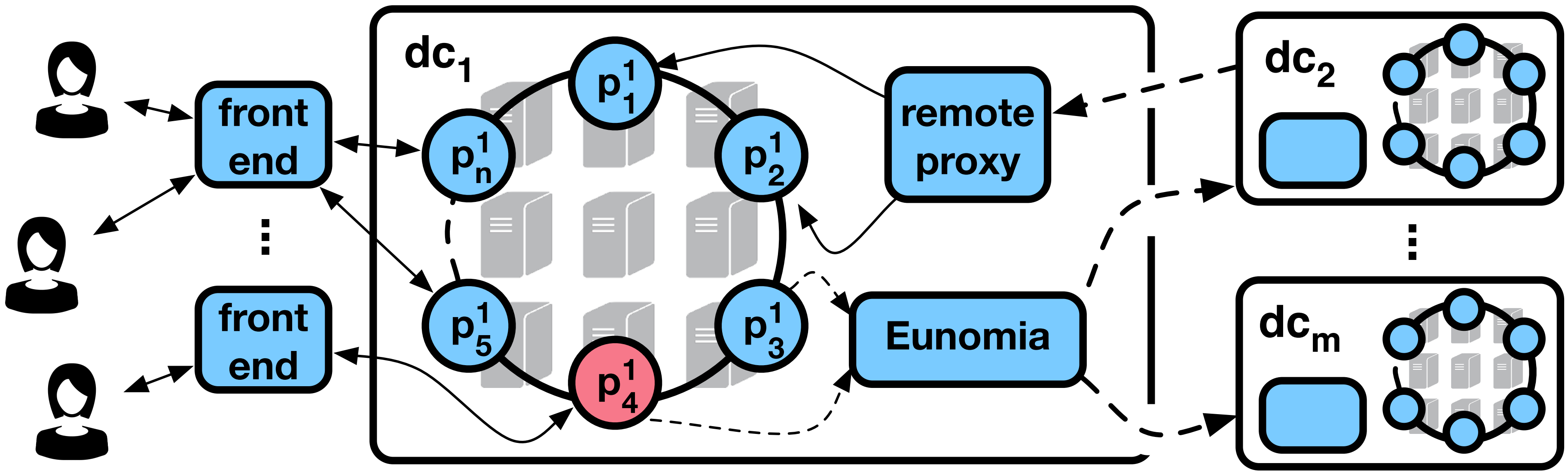


EunomiaKV: supporting geo-replication

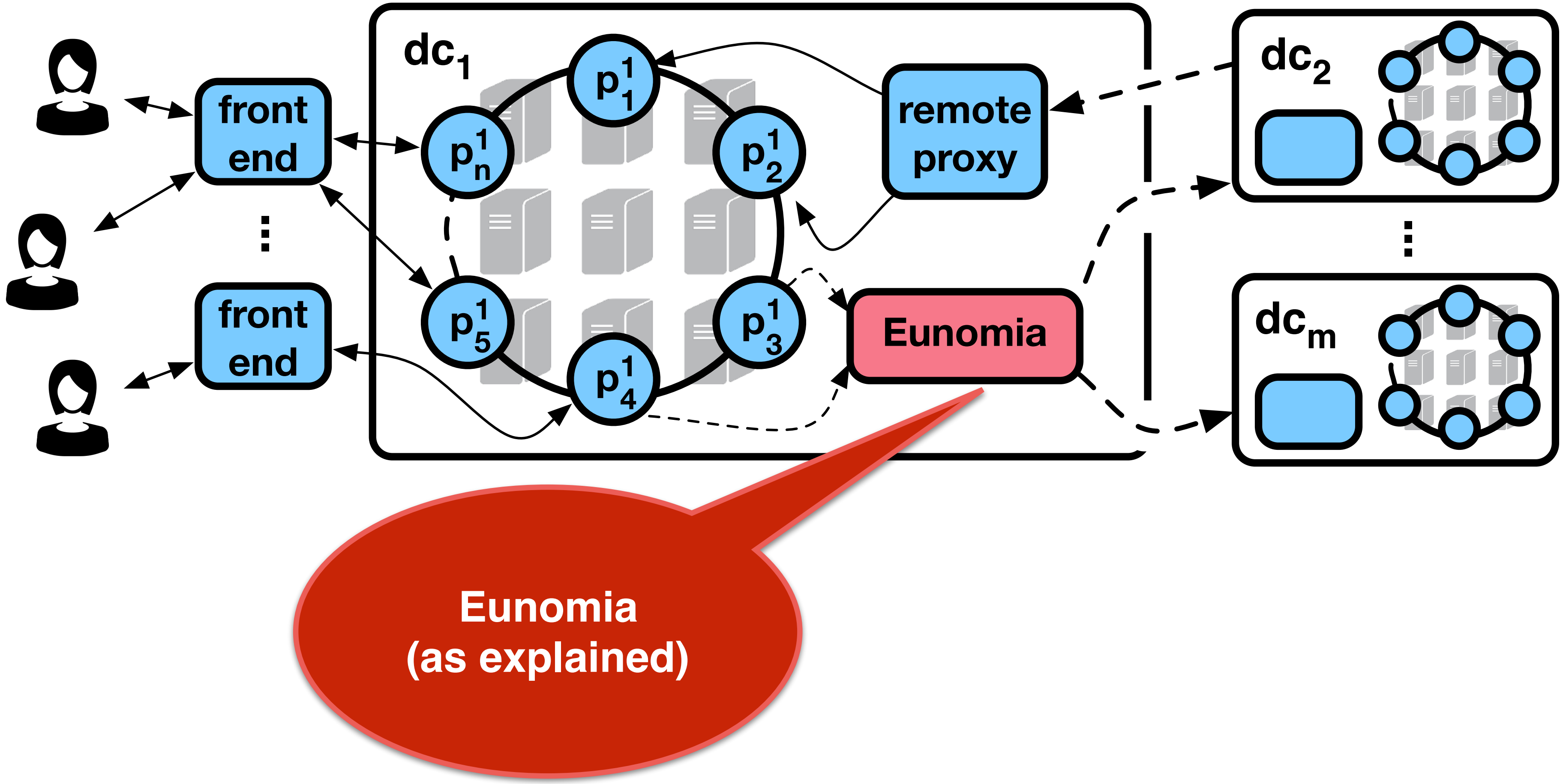


abstract partitioning and replication details from clients

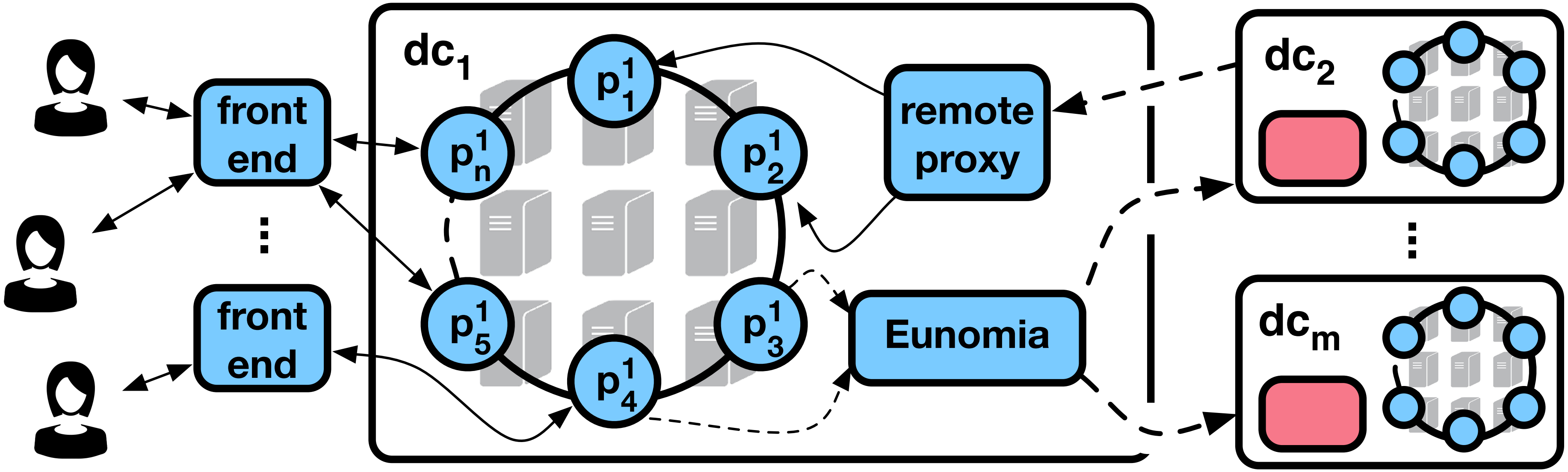
EunomiaKV: supporting geo-replication



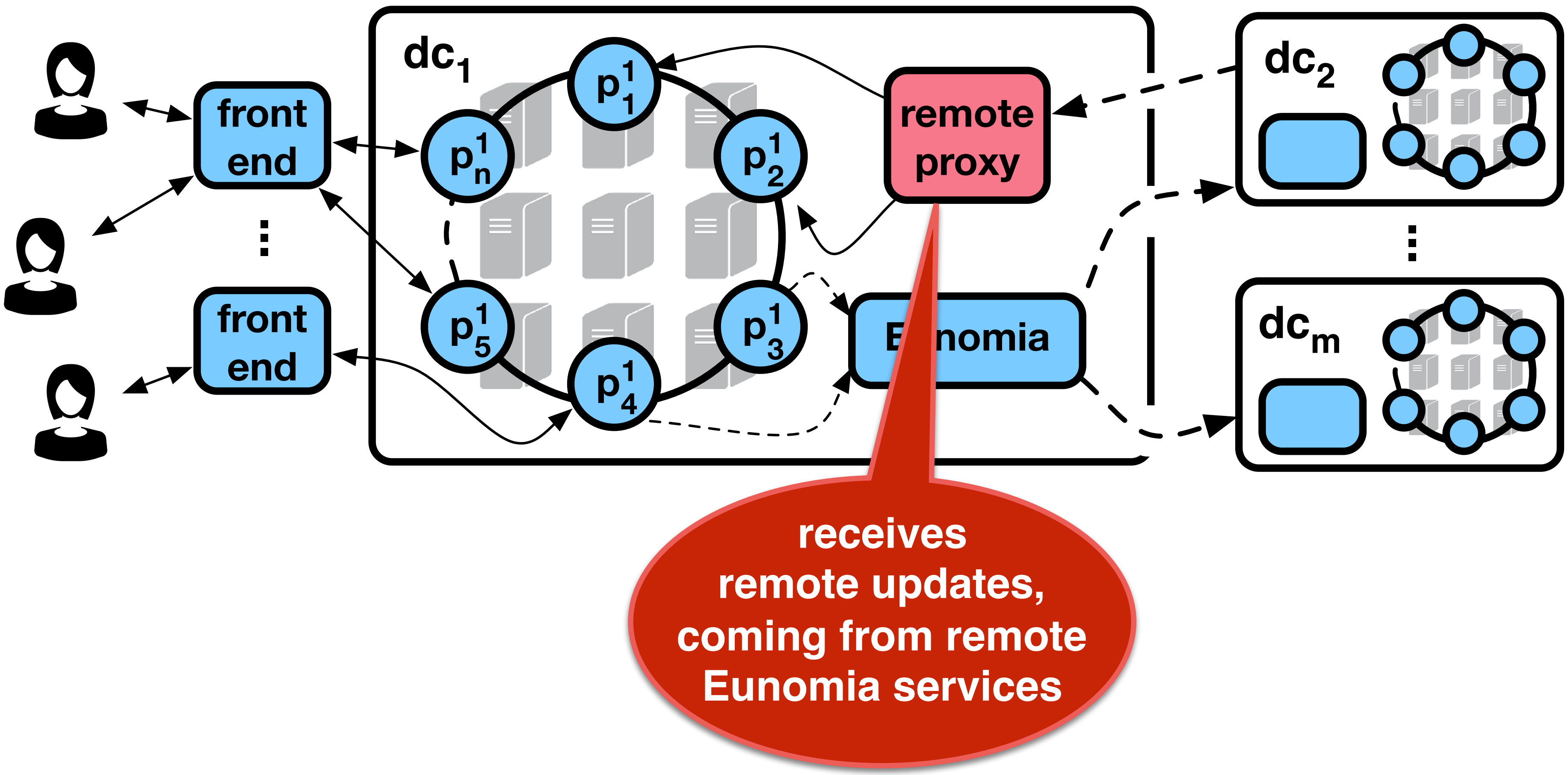
EunomiaKV: supporting geo-replication



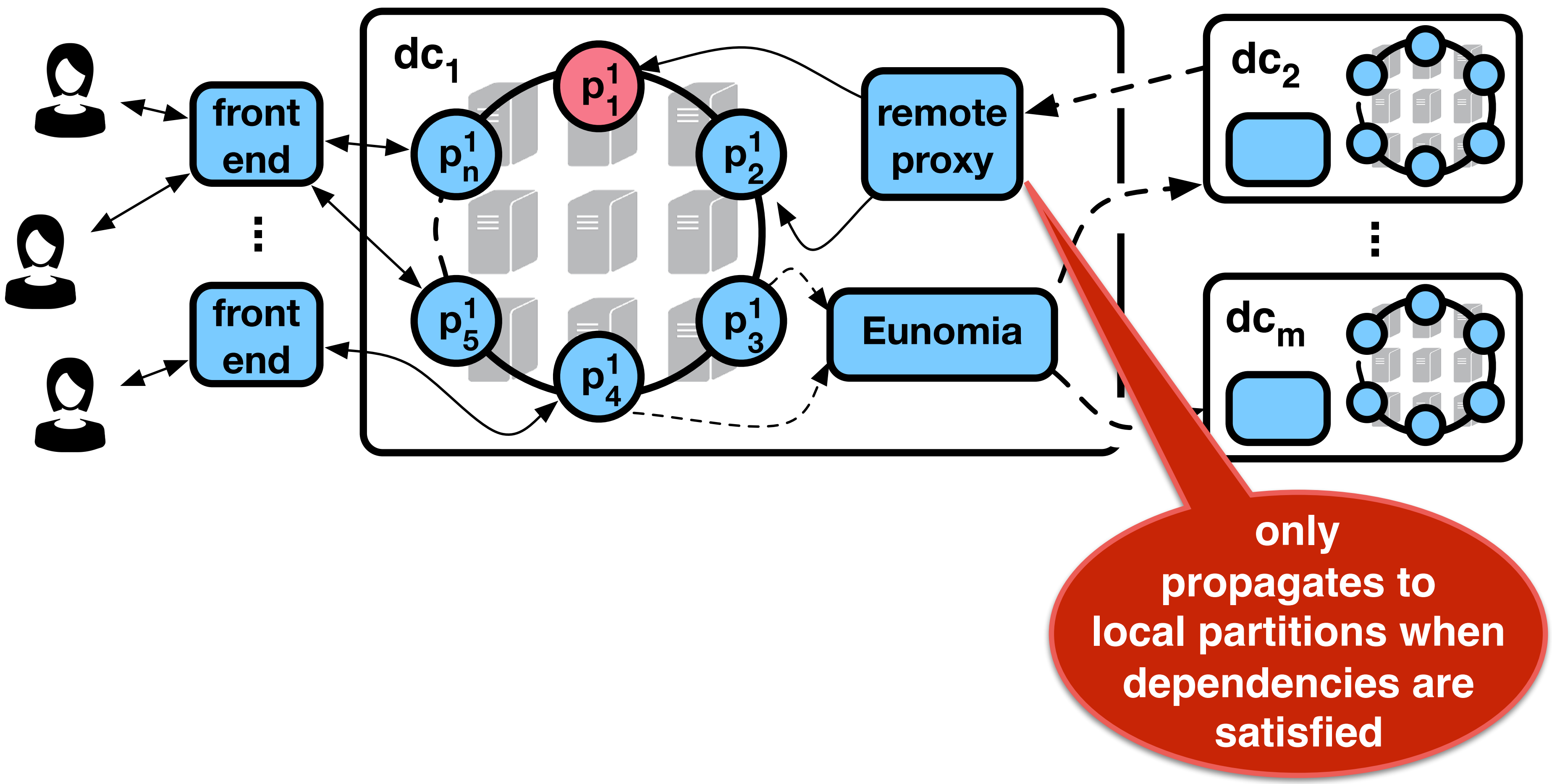
EunomiaKV: supporting geo-replication



EunomiaKV: supporting geo-replication



EunomiaKV: supporting geo-replication



Implementation

Eunomia is implemented in C++ (200 LOC).
At its core, it uses a **red-black tree**

EunomiaKV is built as a **variant of** the open source version of **Riak** (100 lines of Erlang code)

Our implementation uses **hybrid logical clocks** (HLC) becoming resilient to both clock and workload skew

Evaluation: EunomiaKV vs. state-of-the-art

Three datacenter deployment, emulating Amazon EC2 Virginia, Oregon, and Ireland regions

baseline: system that adds no overhead due to consistency management

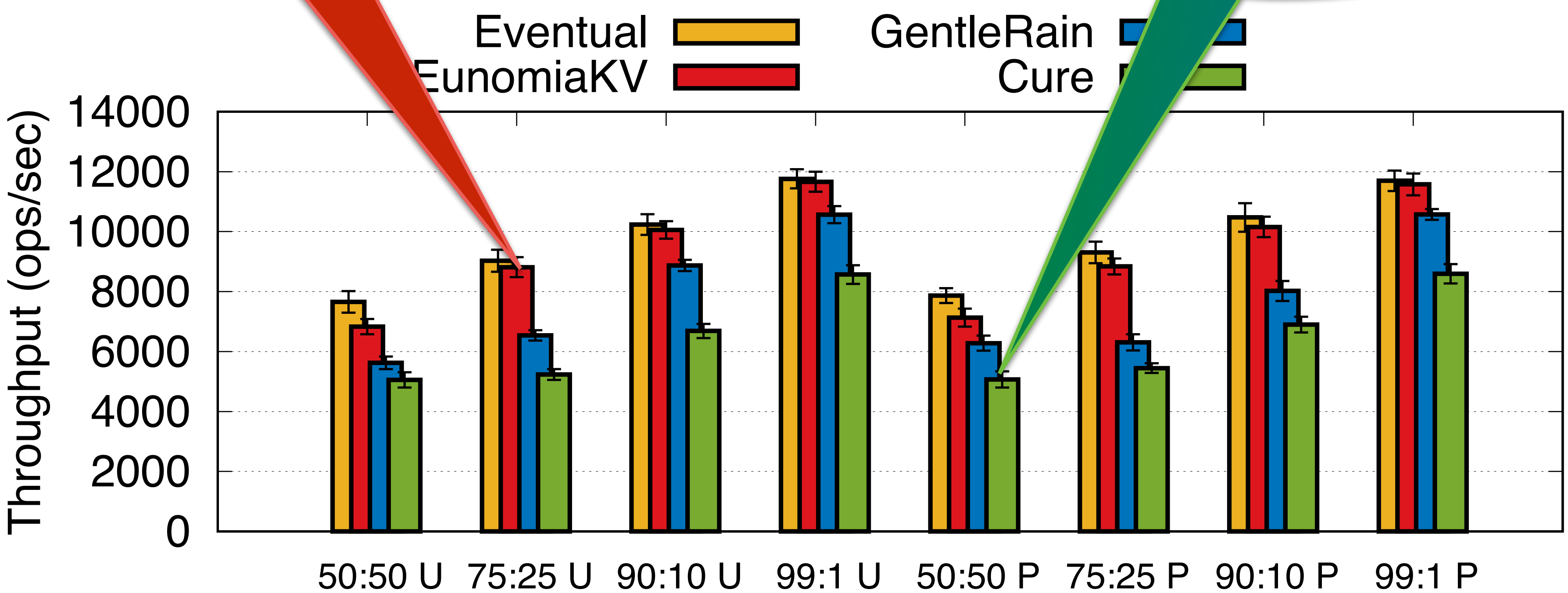
comparison to: global stabilisation solutions, **GentleRain** (single scalar) and **Cure** (vector as EunomiaKV)

Evaluation

EunomiaKV barely penalizes throughput when compared to the baseline

hroughput compari

Global stabilisation has a significant cost, specially in Cure, that uses more metadata

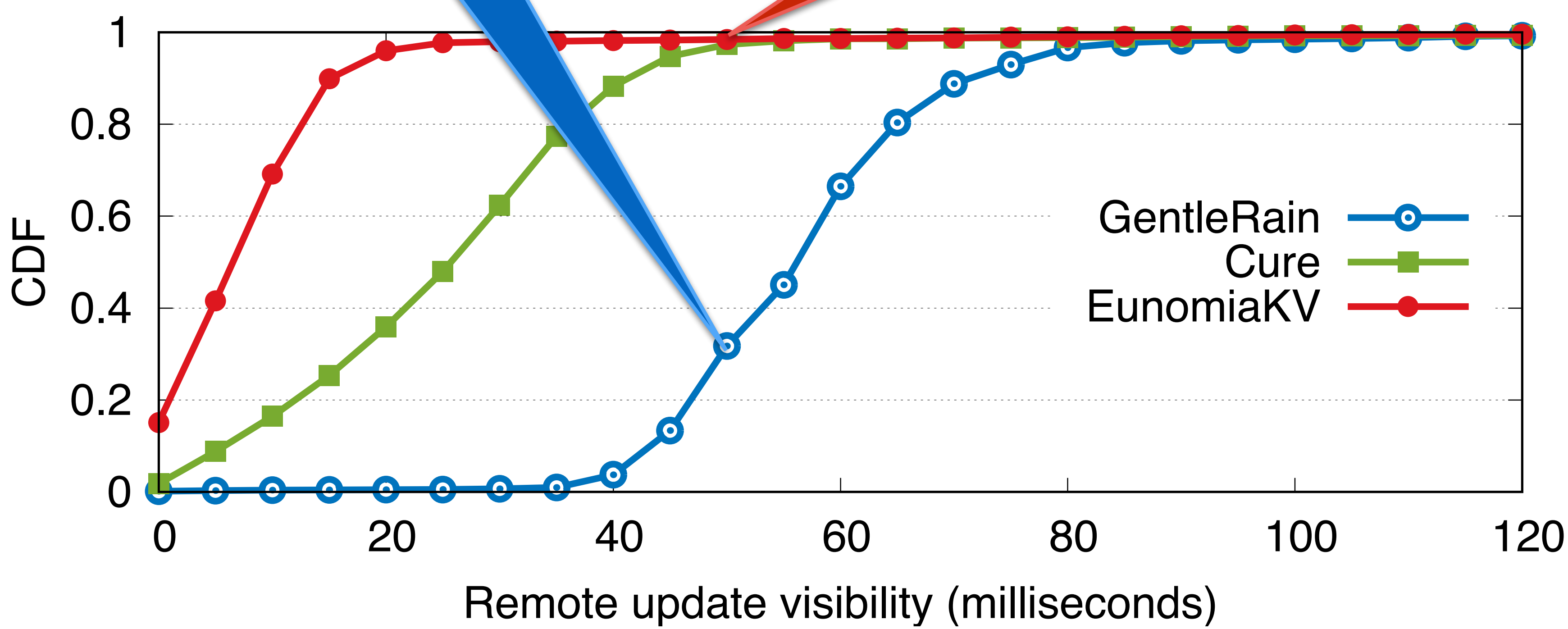


Evaluation

GentleRain significantly penalizes visibility due to the amount of false positives

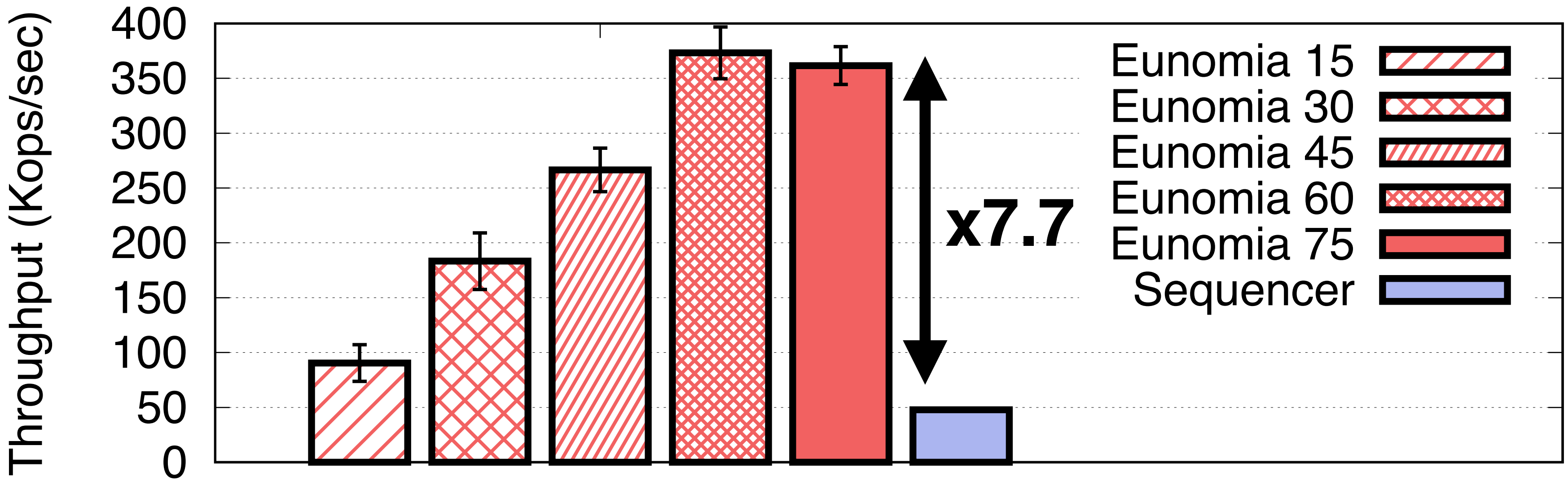
ility compari

EunomiaKV provides even better results than Cure, using the same amount of metadata

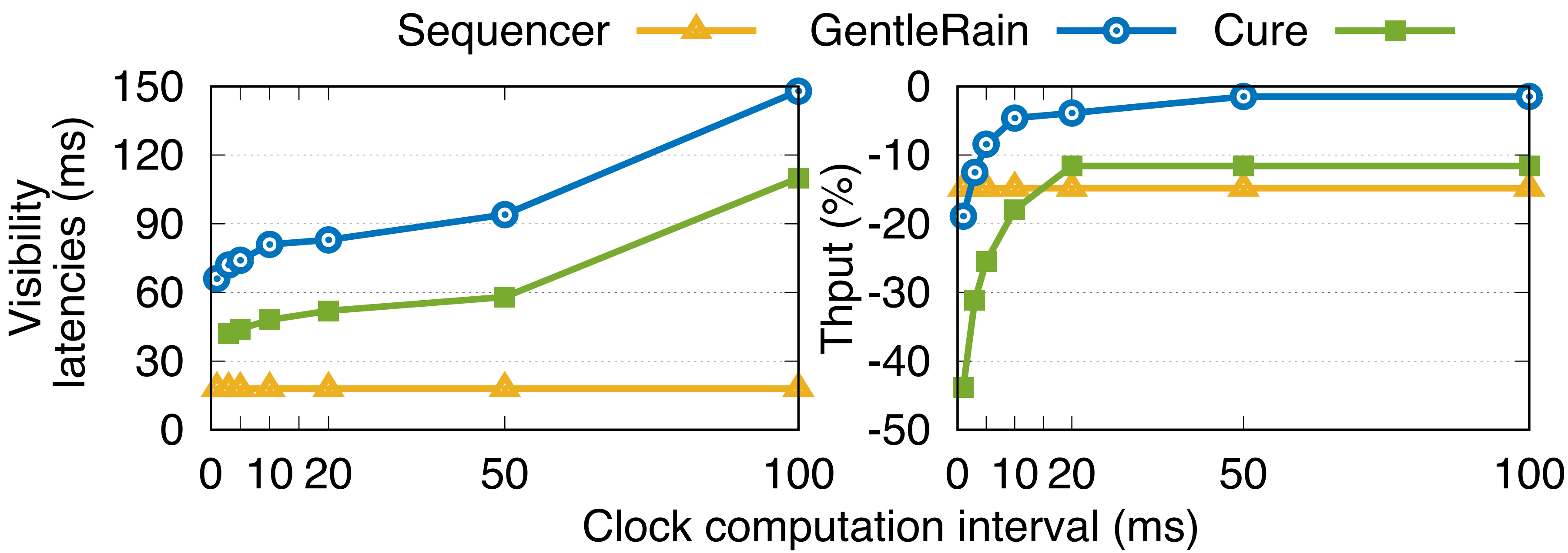


Evaluation

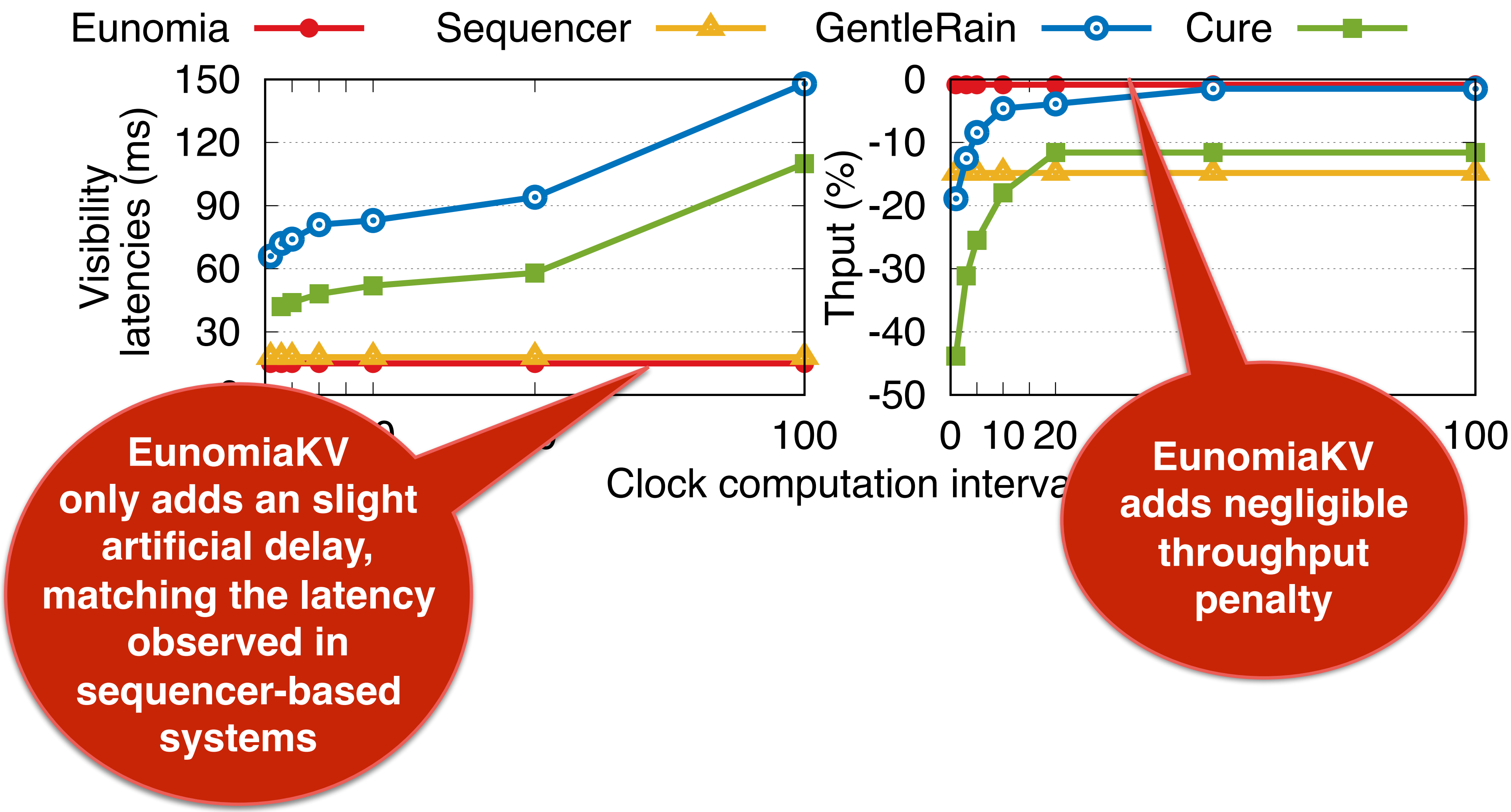
maximum throughput achievable by
Eunomia vs a classical **sequencer**



Going back to the beginning



Going back to the beginning



Thanks!

take-away message

by taking the coordination with an ordering service out the the client's critical path, one can pick a sweet-spot in the throughput vs. visibility tradeoff

check the paper!!!

fault-tolerant version of Eunomia

impact of **stragglers**

more **experiments**