Antidote

A scalable and consistent transactional datastore

Annette Bieniusa
University of Kaiserslautern

SEVENTH FRAMEWORK PROGRAMME

SyncFree

Inria
INVENTEURS DU MONDE NUMÉRIQUE

Technische Universität Kaiserslautern

Koç Üniversitesi

Universidade Nova de Lisboa
Omnis civitas contra se divisa non stabit

Universitas Catholica Lovaniensis
Sedes Sapientiae
1425

basho

TRIFORK.

Erlang Solutions

ROVIO

The SyncFree vision

Geo-scale
edge/fog/cloud

Building scalable highly-available correct systems

Partition-tolerant
Fault-tolerant
Resilient

Application specification must be implemented

SyncFree Research agenda

- Use cases and specification
- Verification tools
- Programming languages, libraries, frameworks
- Deployment and maintenance of systems
- Security
- Protocols for information propagation
- Data representation

The Role of the Datastore

- Datastore API and semantics are essential
- Provide guarantees on which app developers need to rely
- Usability of app might even depend on it
- Consistency needs to be balanced with availability and other requirements

Antidote in a nutshell

Cloud-scale research database

- Performance: sharded, parallel DC
- Widely geo-replicated:
  - ➡ Many DCs, large or small, core or edge
- Available: Reads and updates do not block*
  - ➡ Enables fast response
- Sweet spot: performance vs. usability

Research platform

- Different protocols sharing same infrastructure
- Fair comparison

Why Erlang?

- First choice for industry partners
- Initially issues in hiring people and / or getting them up to speed with Erlang
- Started with 4 PhDs and 2 PostDocs who didn't know anything about Erlang
- Allows fast prototyping
- Code quality has improved a lot (dialyzer, tests)

Enabling technologies

1. CRDTs
   - Industry-mature

2. Highly available, causally-consistent transactions
   - Ready for transfer

I. CRDTs

Conflict-free replicated data types

Abstract data type

- Encapsulates state
- Well-defined interface

Replicated

- At multiple nodes (even clients!)

Conflict-free

- Update replica without coordination
- Convergence guaranteed by design (formal properties)
- Decentralized
- **No lost updates**

CRDTs in Antidote

<table>
<tr><td>Counter</td><td><code>{increment, integer()}</code>
<code>{decrement, integer()}</code></td></tr>
<tr><td>ORSet</td><td><code>{add, term()}</code>
<code>{remove, term()}</code>
<code>{add_all, [term()]}</code>
<code>{remove_all, [term()]}</code></td></tr>
<tr><td>GSet*</td><td><code>{add, {term(), actor()}}</code>
<code>{add_all, {[term()], actor()}}</code></td></tr>
<tr><td>LWW Register*</td><td><code>{assign, {term(), non_neg_integer()}}</code>
<code>{assign, term()}.</code></td></tr>
<tr><td>MV Register</td><td><code>{assign, {term(), non_neg_integer()}}</code>
<code>{assign, term()}</code></td></tr>
<tr><td>Map*</td><td><code>{update, {[map_field_update() | map_field_remove()], actorordot()}}.</code>

<code>-type actorordot() :: riak_dt:actor() | riak_dt:dot().</code>
<code>-type map_field_remove() :: {remove, field()}.</code>
<code>-type map_field_update() :: {update, field(), crdt_op()}.</code>
<code>-type crdt_op() :: term(). %% Valid riak_dt updates</code>
<code>-type field() :: term()</code></td></tr>
<tr><td>RGA
(replicated growable array)</td><td><code>{addRight, {any(), non_neg_integer()}}</code>
<code>{remove, non_neg_integer()}</code></td></tr>
</table>

* wrappers for CRDTs in riak_dt

Consistency at different levels

Single object

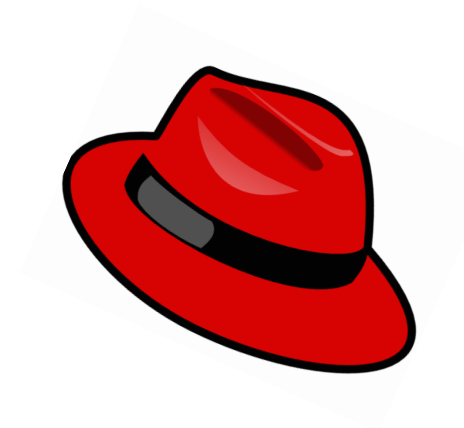
- Safe: updates, state satisfy specification, internal invariants
- Replicas converge to same state

Multiple objects

- Relations between objects
- Cross-object invariants
- Different invariants $\Longrightarrow$ different mechanisms

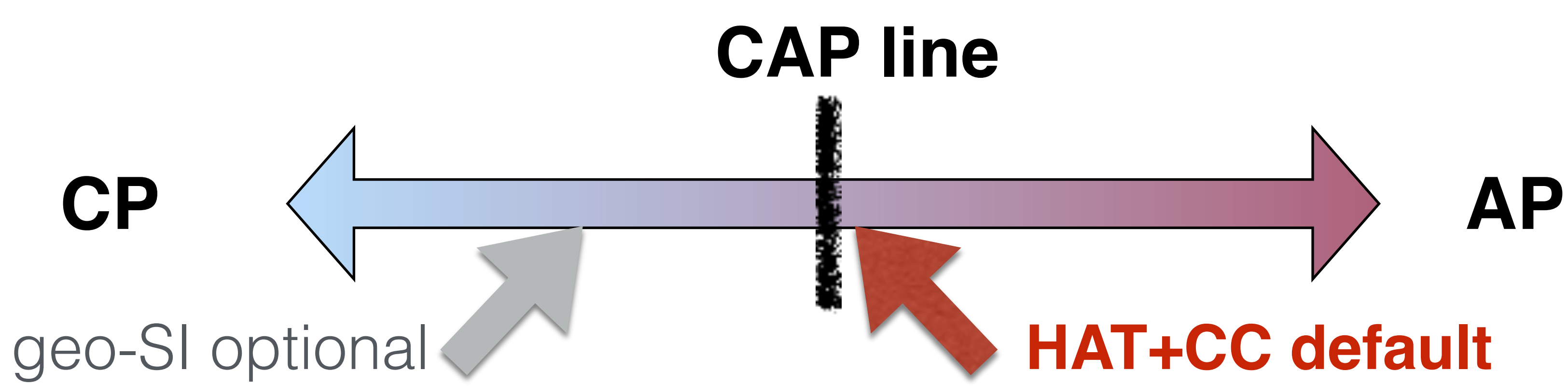
II. Transactions

Highly Available Transactions (HATs)



Transaction with weaker isolation properties [Bailis et al. VLDB' 14]

- Monotonic reads
- Monotonic writes
- Writes-follow-reads
- All-or-nothing writes

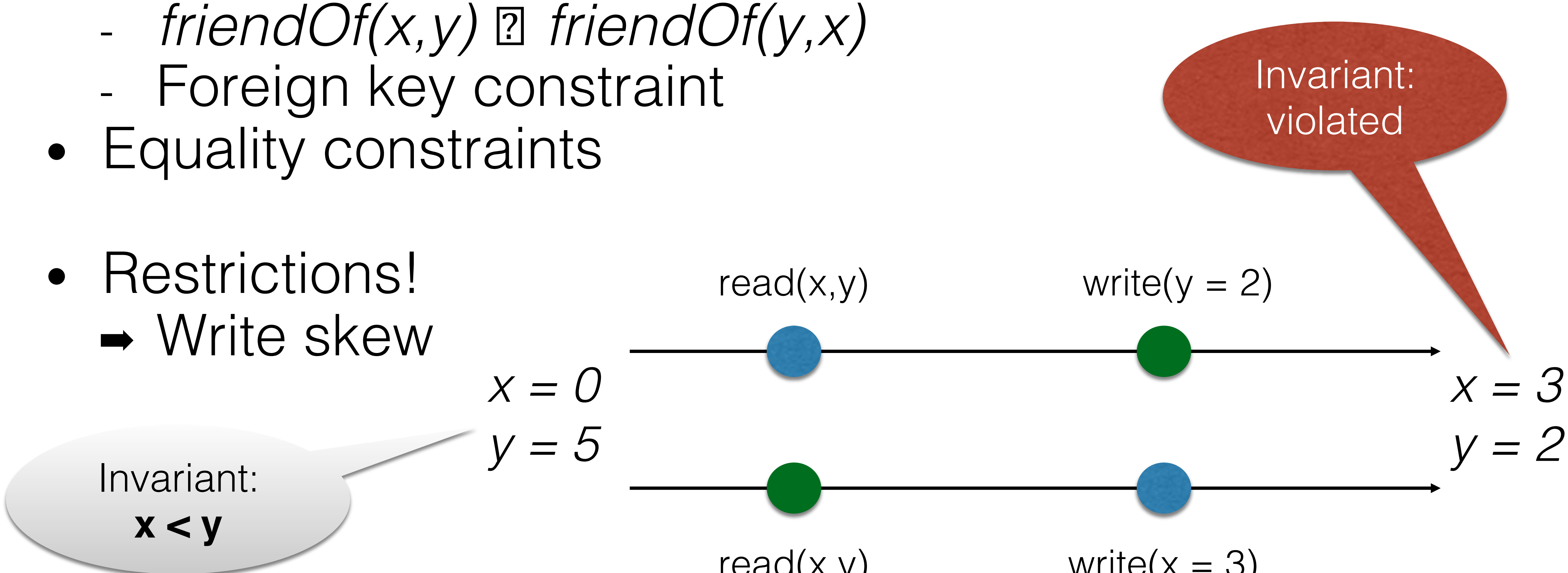


Guarantees

- Weak invariant preservation

- Example: Maintaining friend lists
  - *friendOf(x,y)* ⇔ *friendOf(y,x)*
  - Foreign key constraint
- Equality constraints

- Restrictions!
  ➡ Write skew



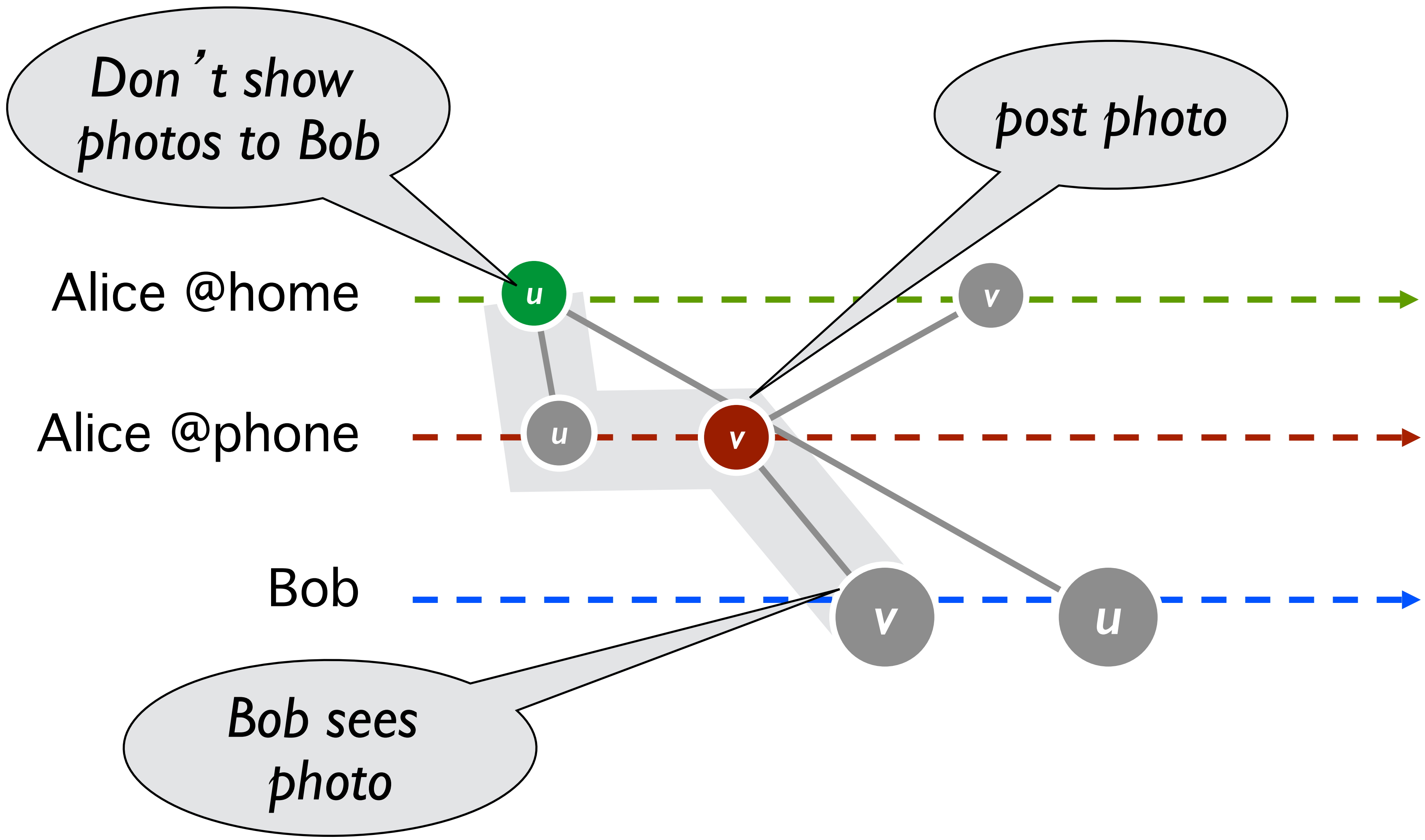
Interactive Transaction API

```
type bound_object() = {key(), crdt_type(), bucket()}.
type snapshot_time() = vectorclock() | ignore.

start_transation(snapshot_time(), properties()) ->
         {ok, txid()} | {error, term()}.

update_objects([{bound_object(),operation(),op_param()}],txid()) ->
         ok | {error, term()}.

read_objects([bound_object()], txid()) -> {ok, [term()]}.

commit_transaction(txid()) ->
         {ok, vectorclock()} | {error, term()}.
```

Static Transaction API

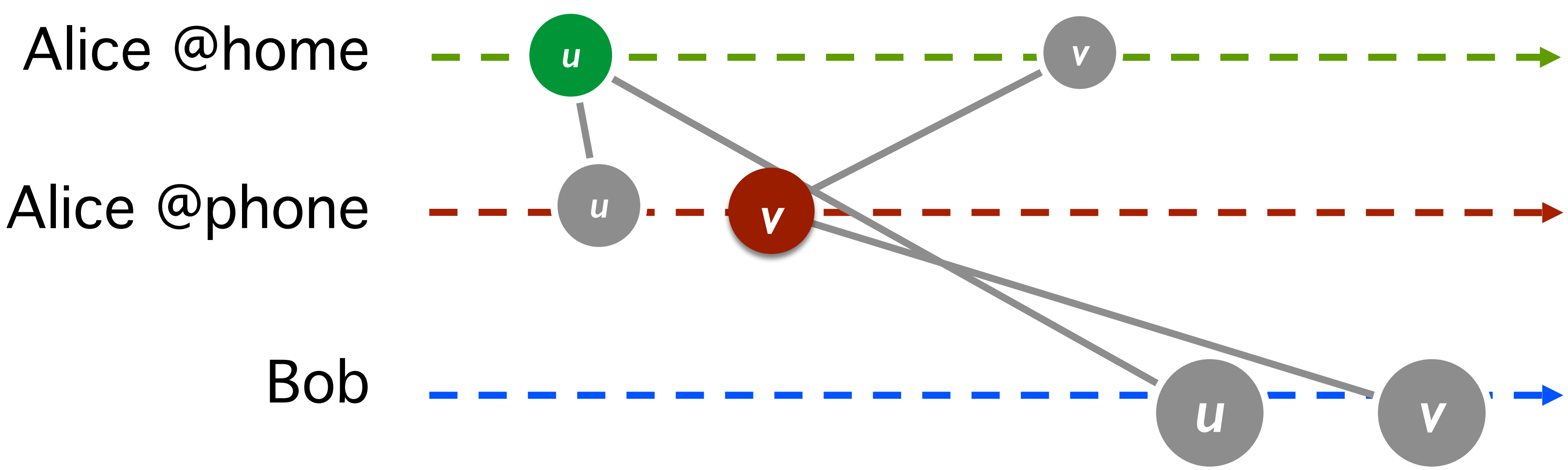
```
type bound_object() = {key(), crdt_type(), bucket()}.
type snapshot_time() = vectorclock() | ignore.

update_objects(snapshot_time(), properties(),
    [{bound_object(), operation(), op_param()}]) ->
         {ok, vectorclock()} | {error, reason()}.

read_objects(snapshot_time(), properties(),
     [bound_object()]) ->
          {ok, [term()], vectorclock()}.
```

Not causal



access (Bob, photo) $\Longrightarrow$ *ACL (Bob, photo)*

Causal consistency



v observed effects of *u*

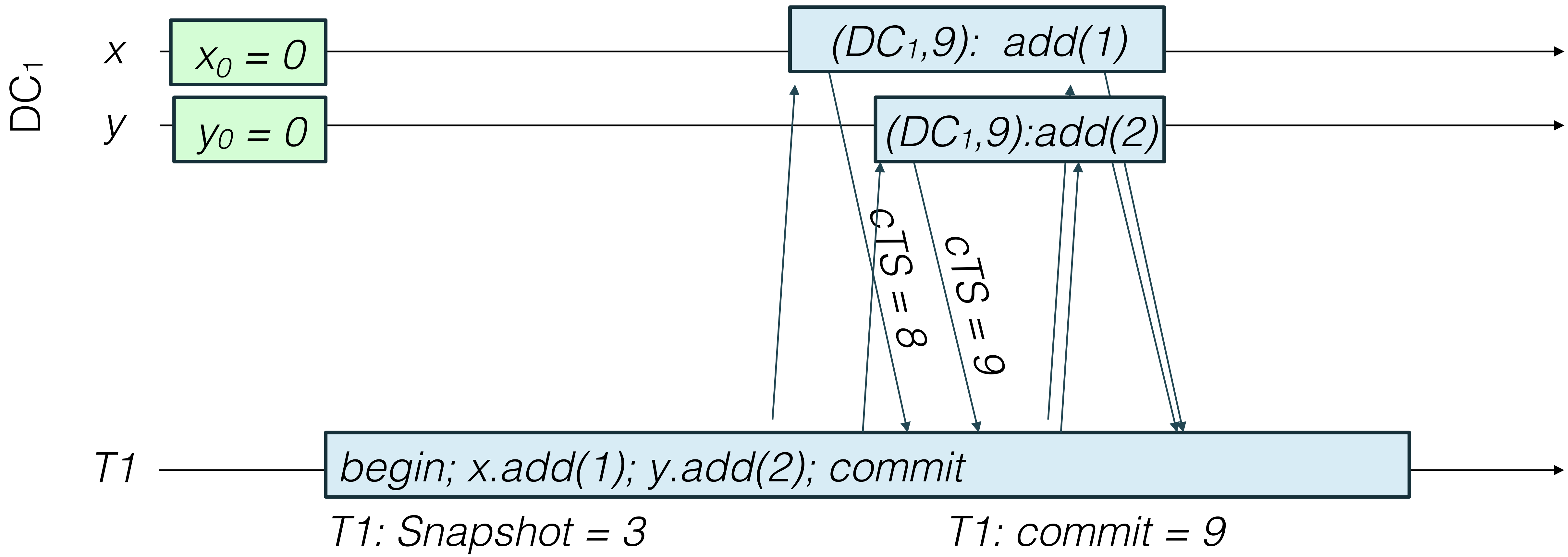
$\Longrightarrow$ *v* should be delivered after *u*

- Guarantees additionally read-your-own-writes
- **Strongest** partition tolerant and available consistency model
- Doesn’t slow down the sender

Clock-SI [Du et al., SRDS 2013]

- Loosely-synchronized physical clocks, per shard
- Data objects versioned by timestamp
- Transaction
  - Read timestamp: Coordinator's current clock
    ⟹ snapshot includes all earlier txns
  - Commit timestamp: 2PC
    ⟹ max(shards' clocks)
  - Snapshot: consistent
  - Writes: all-or-nothing, total order (per DC)
- Read-only txns, single-node txns: no coordination

Clock-SI protocol



Local total order: useful, acceptable

- 2PC disjoint-access parallelism
- Intra-DC communication is fast
- 1 scalar value per DC

Cure

„Geo-replicated Clock-SI“: updates originating in a DC are totally ordered, all-or-nothing

Version vector

- 1 entry / DC
- Private to partition

Other DC's updates visible once all partitions sync'd

Heartbeat msg guarantees progress

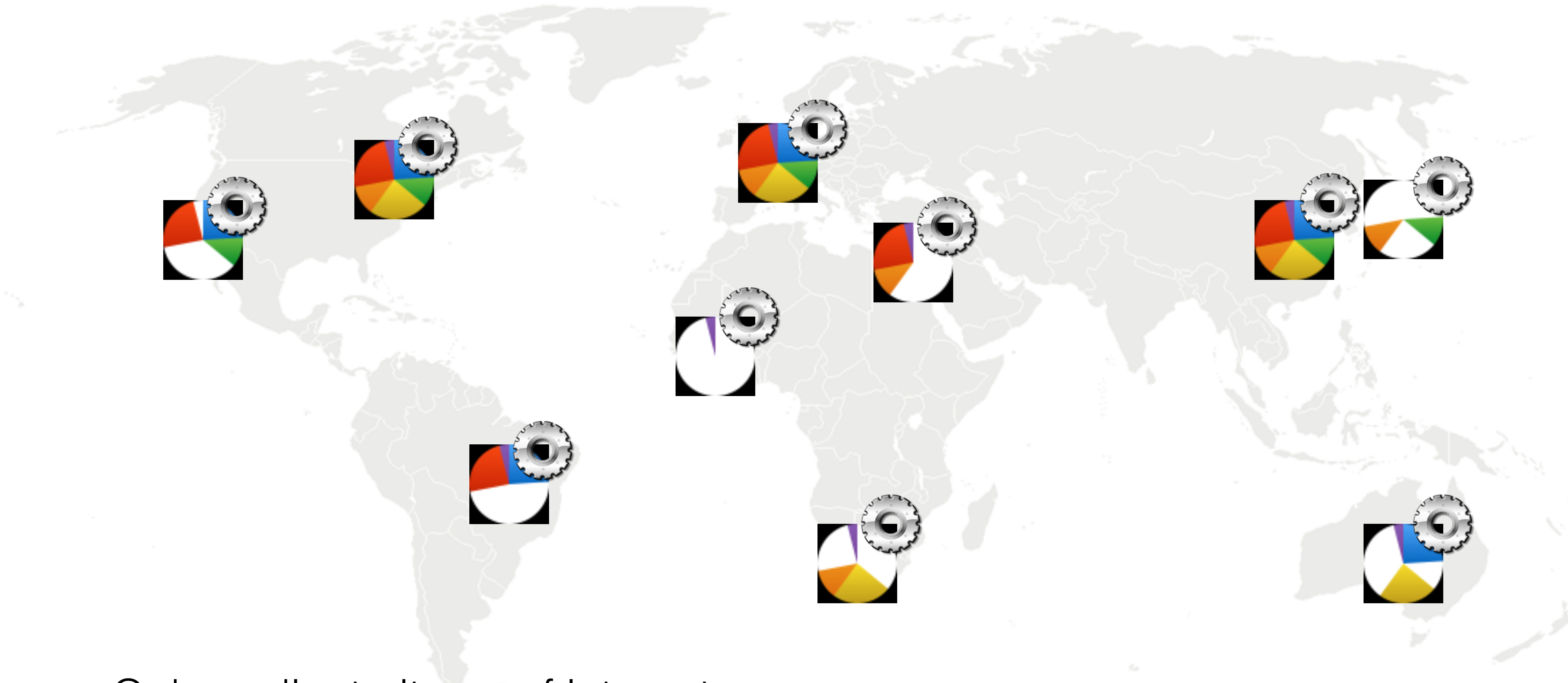
Causal+ Consistency Protocols

	Resolution Logic	Transactions
COPS [SOSP11]	LWW	static read-only
Eiger [NSDI13]	LWW	static read-only static write-only
GentleRain [SoCC14]	LWW	static read-only
Cure [ICDCS16]	CRDTs	read-write interactive

Related Work: implementation

	Update visibility protocol	Update visibility latency	Metadata Size
COPS [SOSP11]	check messages	small	O(objects)
Eiger [NSDI13]	check messages	small (1/2 RTT each DC)	O(objects)
GentleRain [SoCC14]	stabilization	high (1/2 RTT furthest DC)	O(1)
Cure [ICDCS16]	stabilization	small	O(DCs)

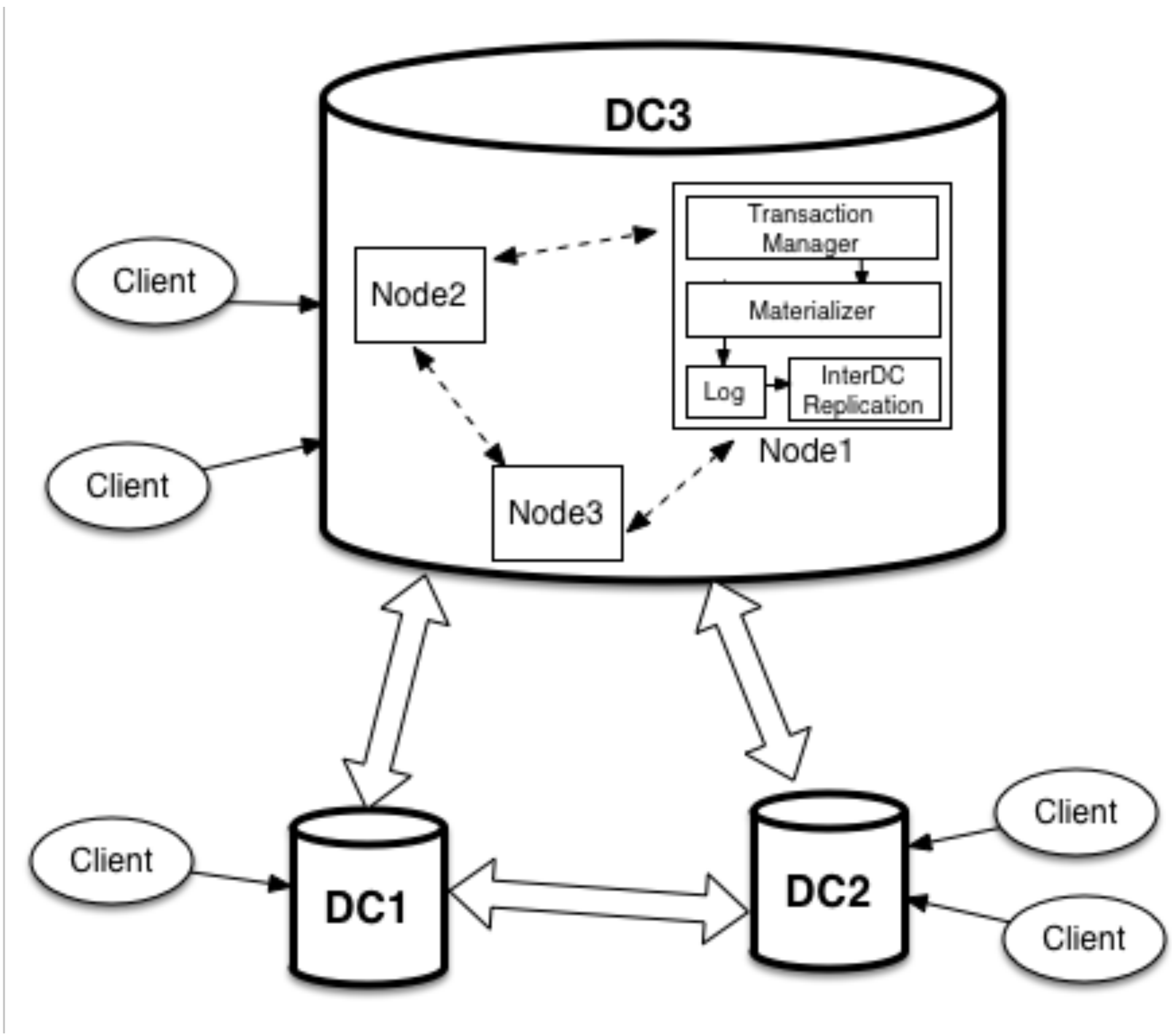
Outlook: Partial replication



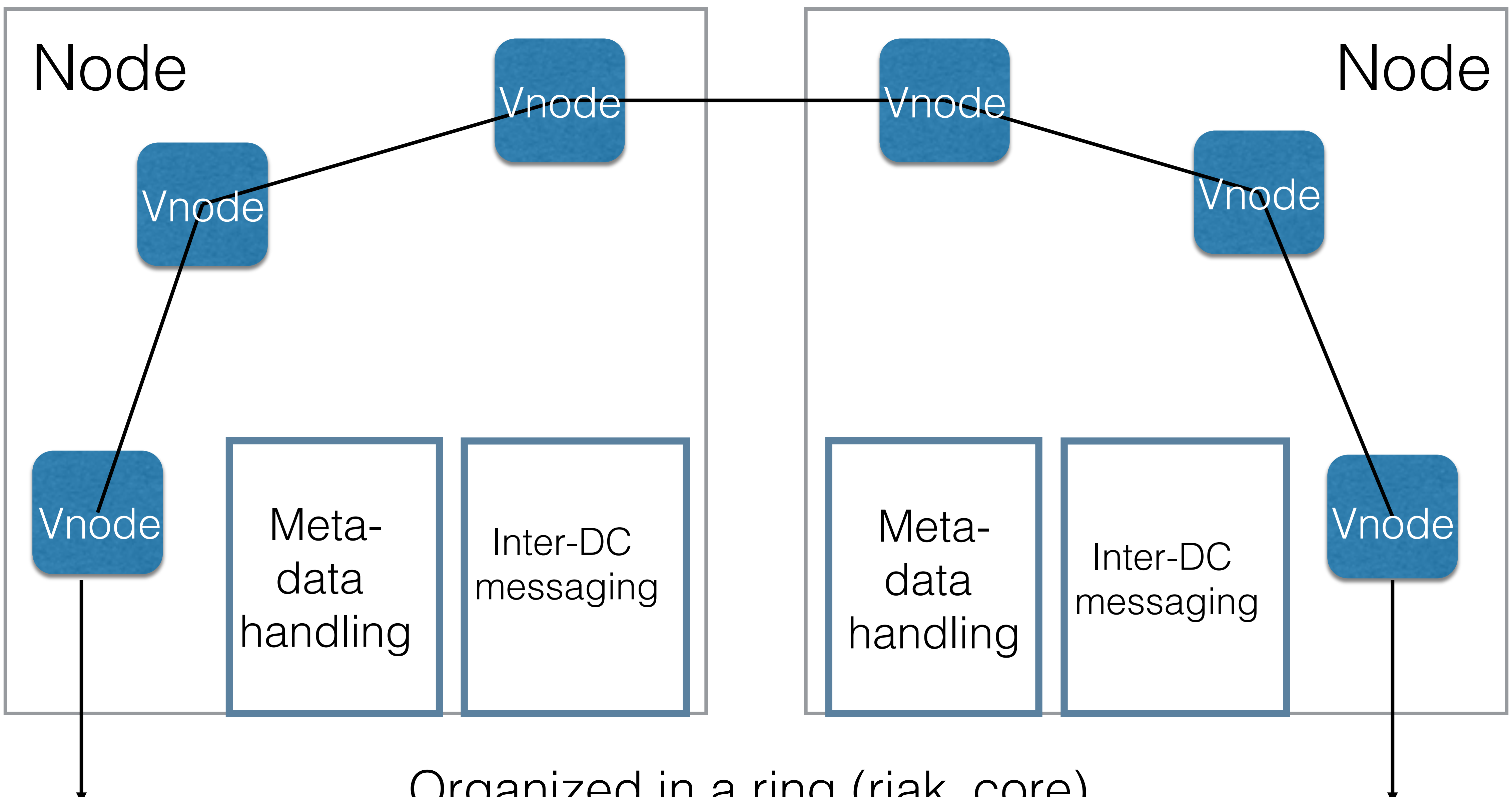
- Only replicate items of interest
- Performance: assumes locality
- ***Genuine***: only receive updates of interest
- Missing information complicates consistency

Architecture

Architecture



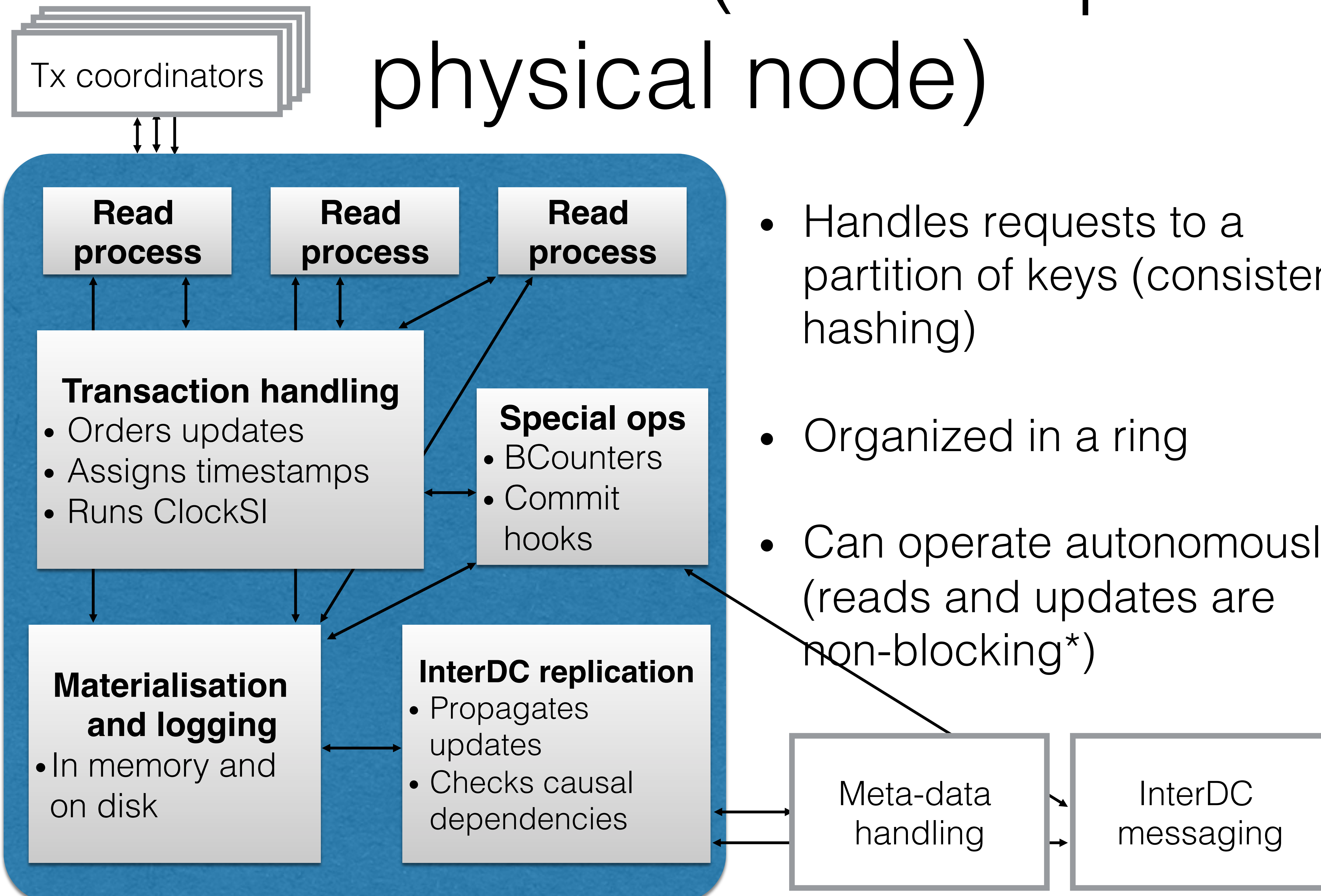
(Physical) Nodes



Organized in a ring (riak_core)

Partitioned by consistent hashing

Virtual Nodes (Several per physical node)



- Handles requests to a partition of keys (consistent hashing)
- Organized in a ring
- Can operate autonomously (reads and updates are non-blocking*)

Evaluation

Preliminary benchmarks

Single item read or read + update transactions

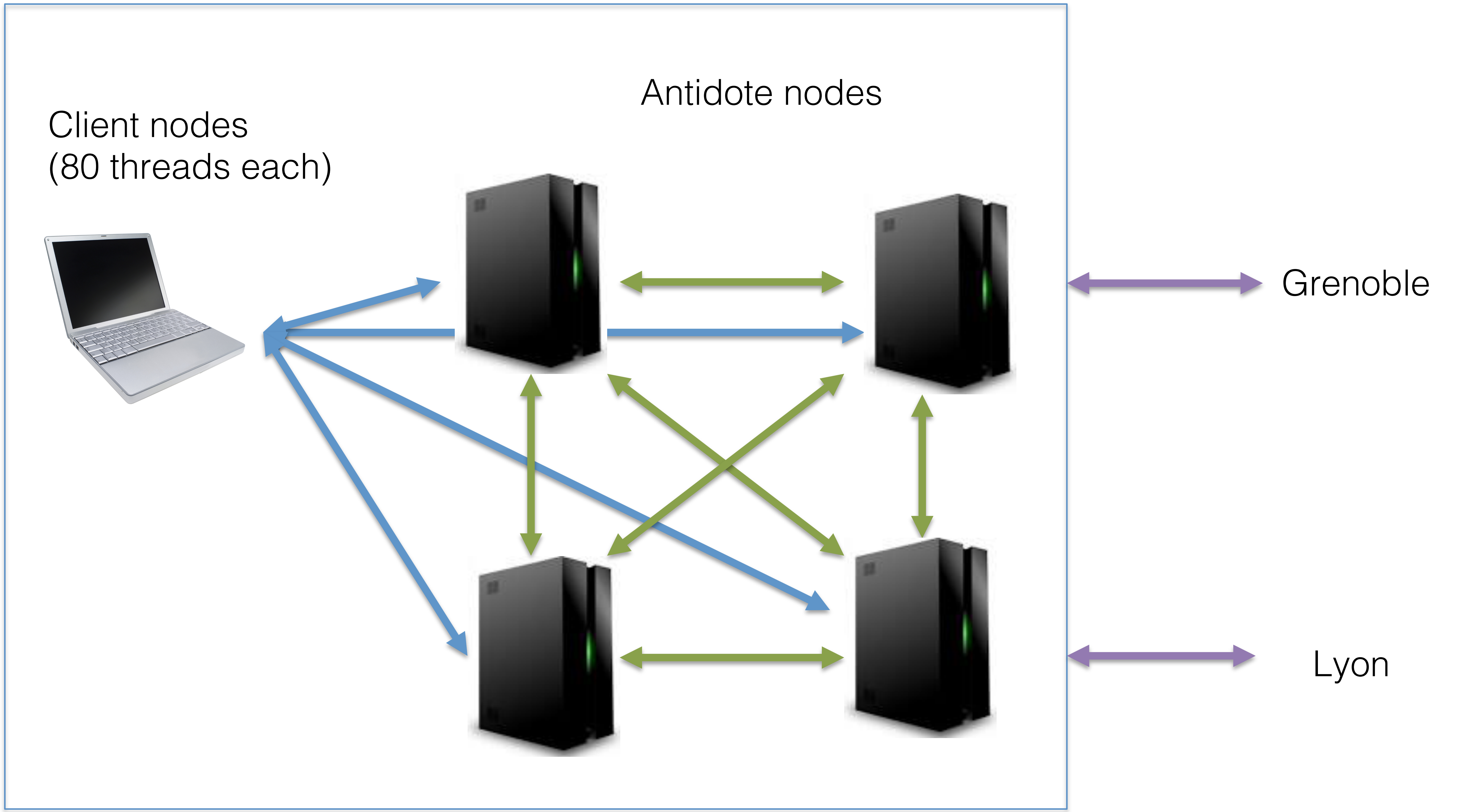
LWW registers, 100 byte objects, 50 000 objects per node

Clients run on different machines, ≈1 client node / 4 Antidote nodes

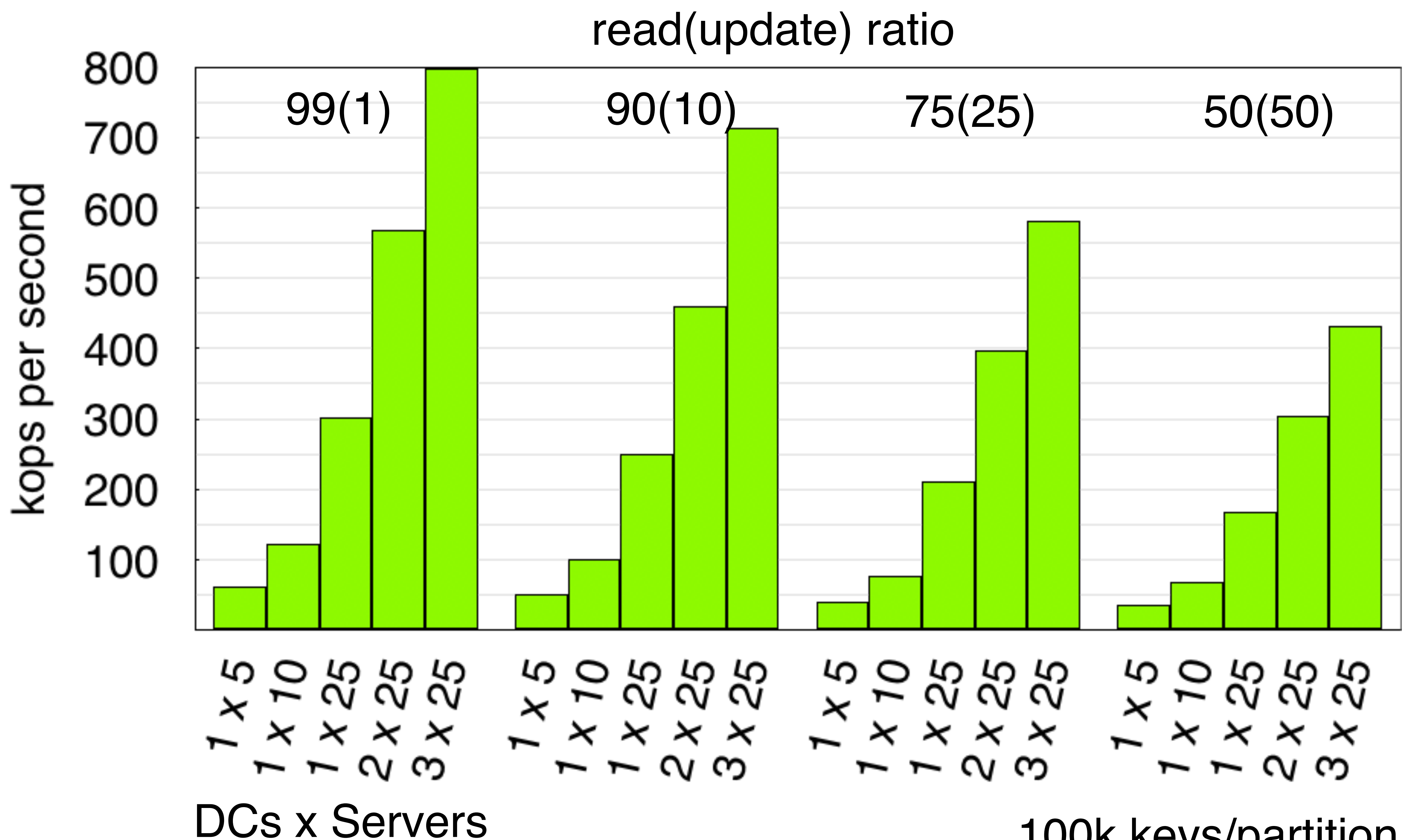
Grid'5000: 130 nodes

Nodes: 2 CPUs Intel Xeon E5-2660, 8 cores/CPU, 64GB RAM, 1863GB HDD, 10Gbps ethernet

Sophia



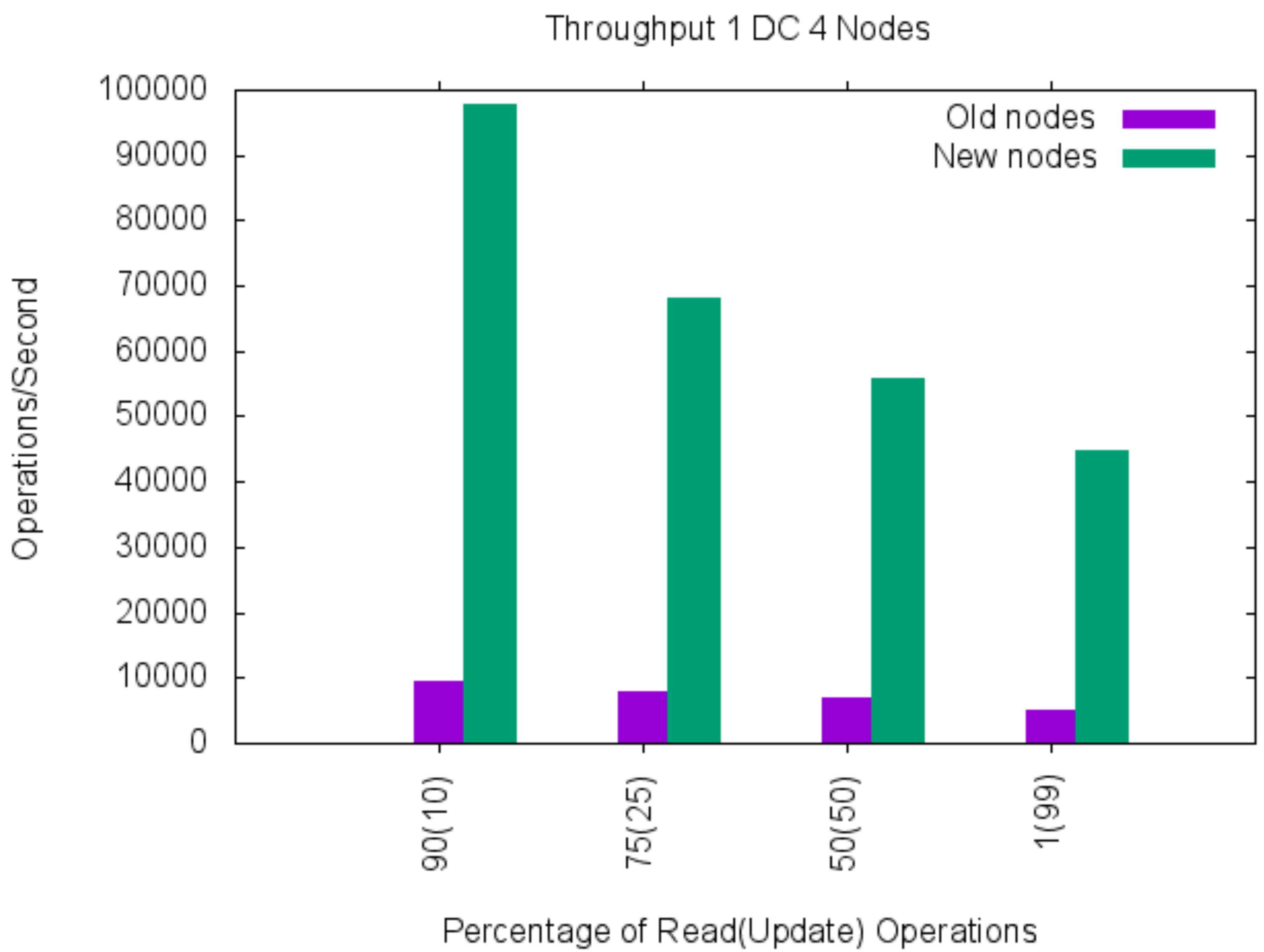
Evaluation: Scalability



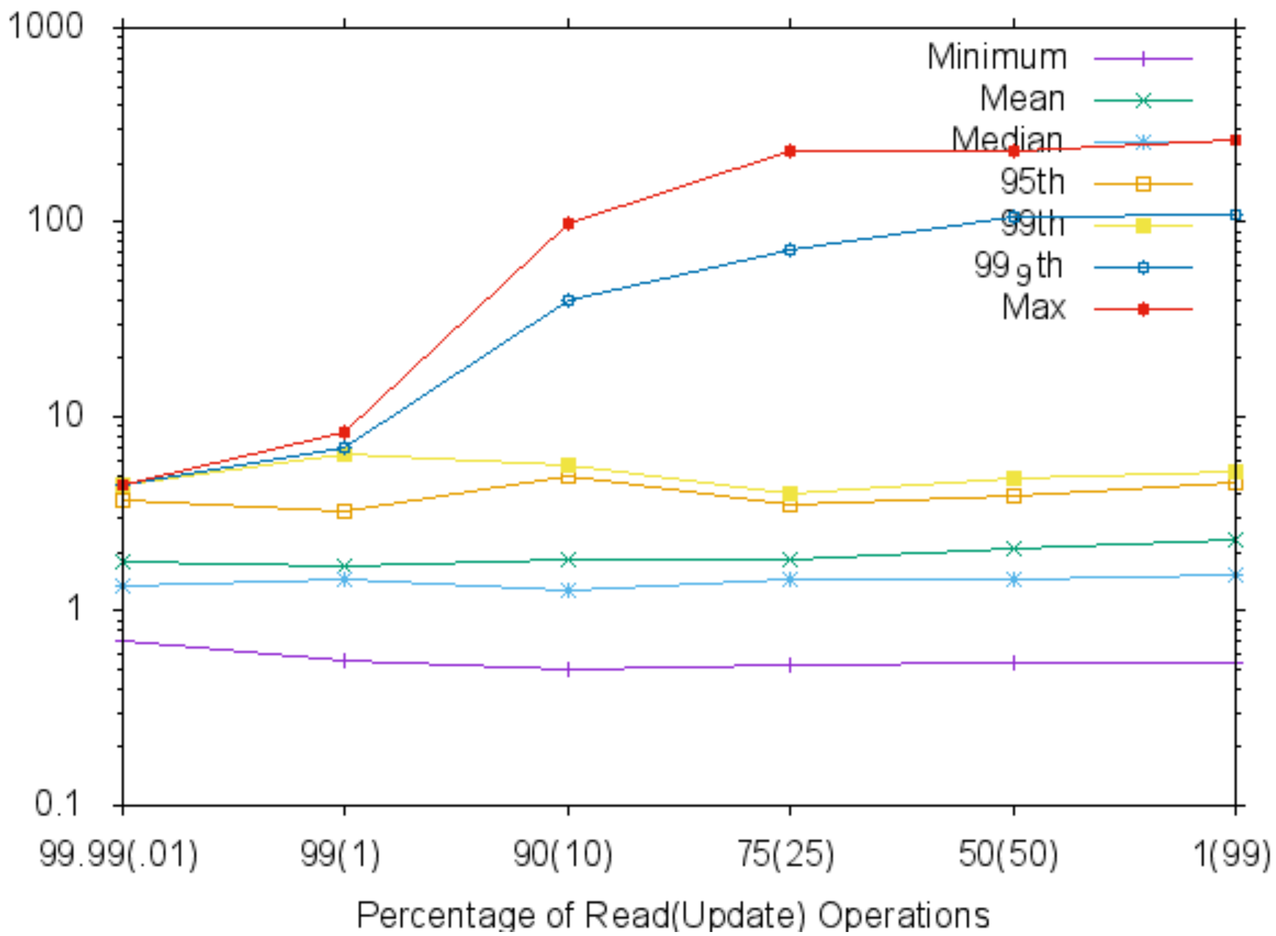
LWW registers

100k keys/partition
power law distribution

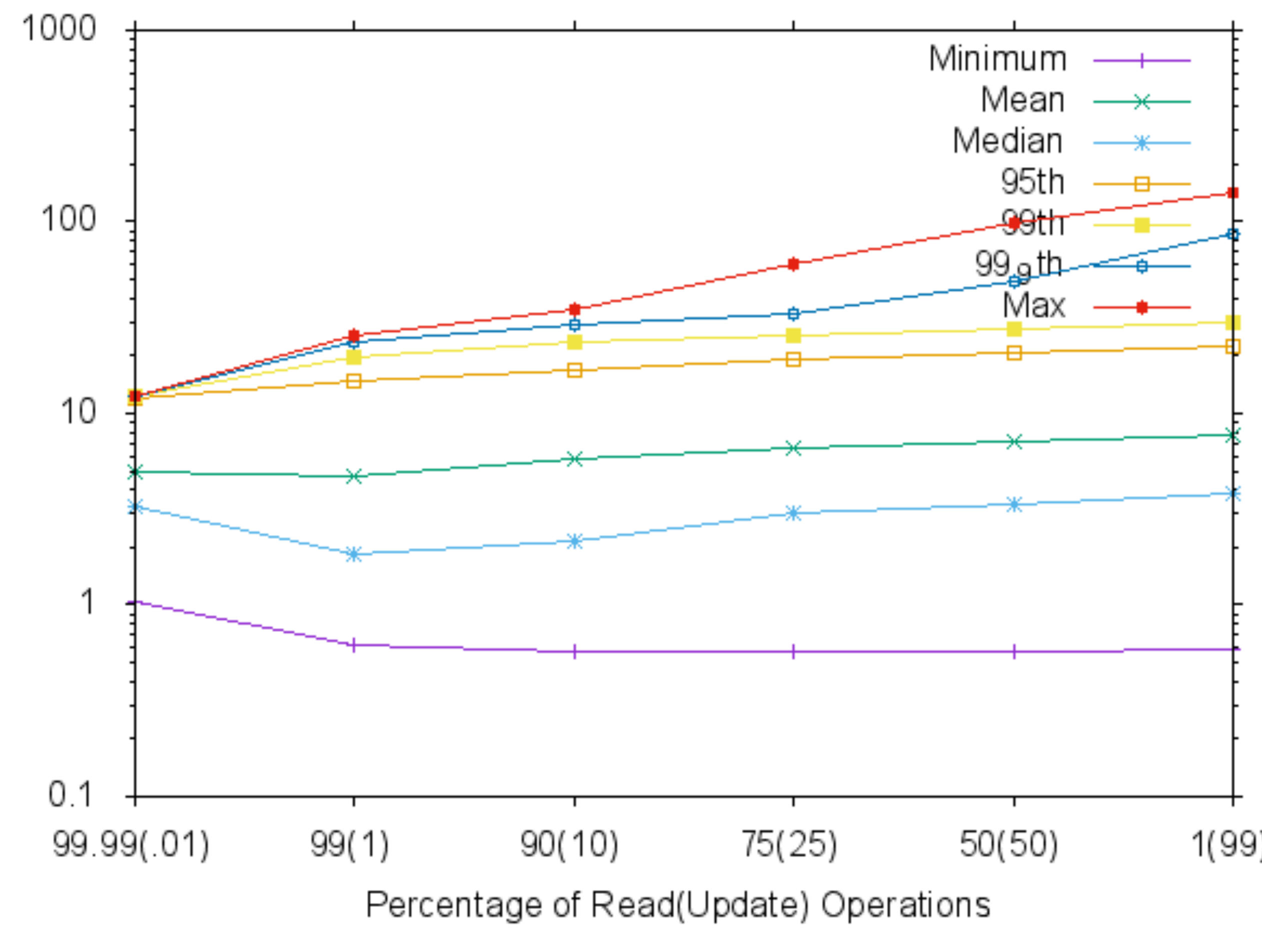
Scale to new hardware - throughput



Update latencies (ms)

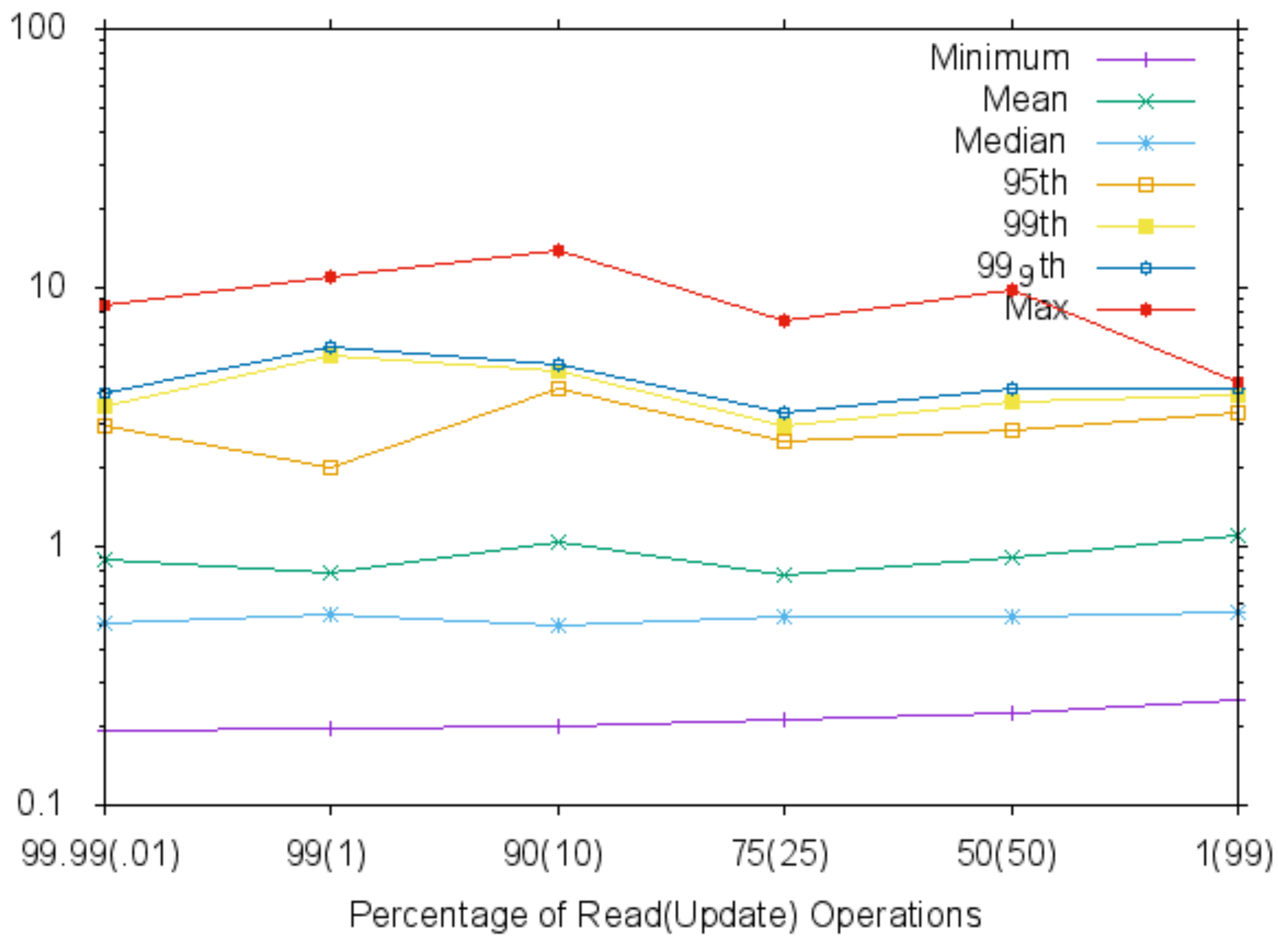


New hardware

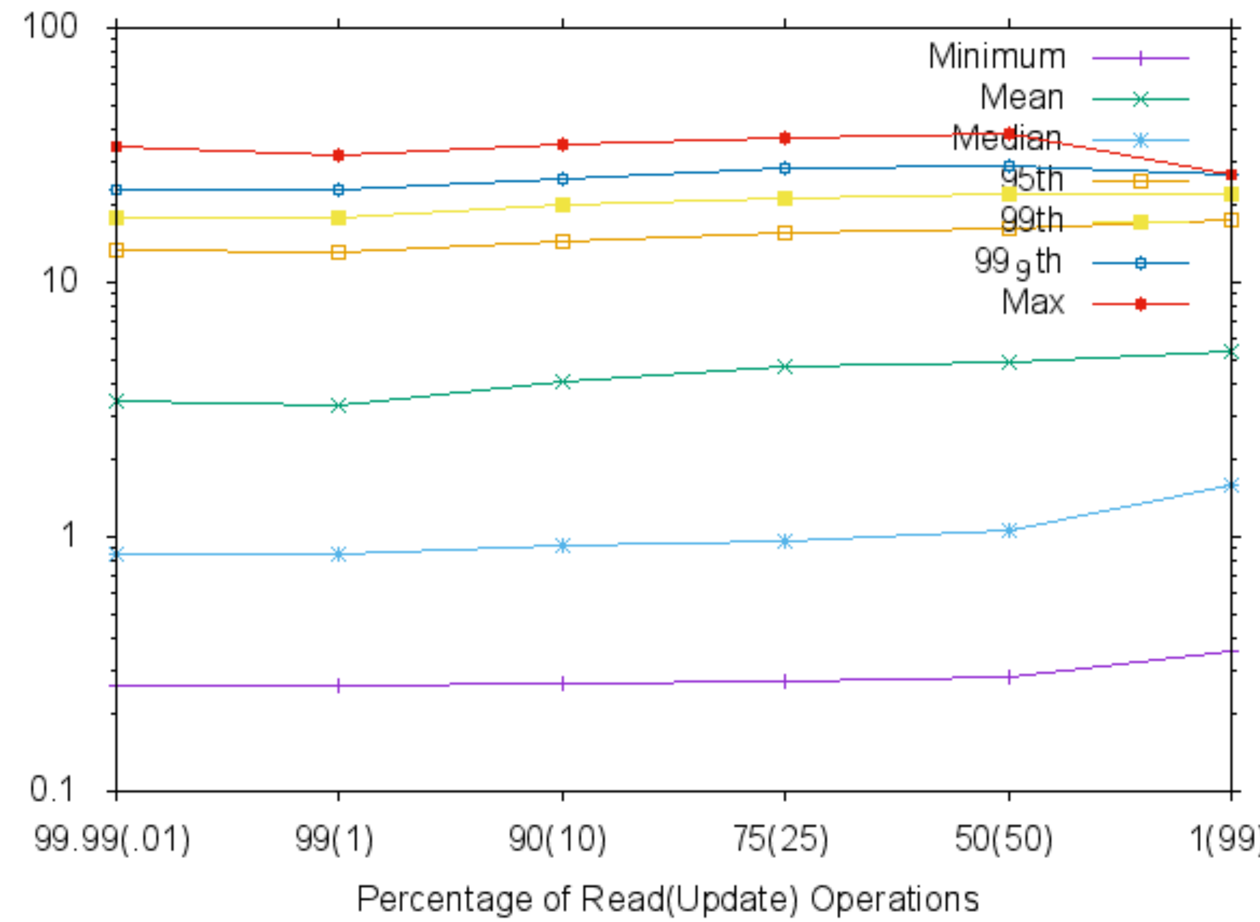


Old hardware

Read latencies (ms)

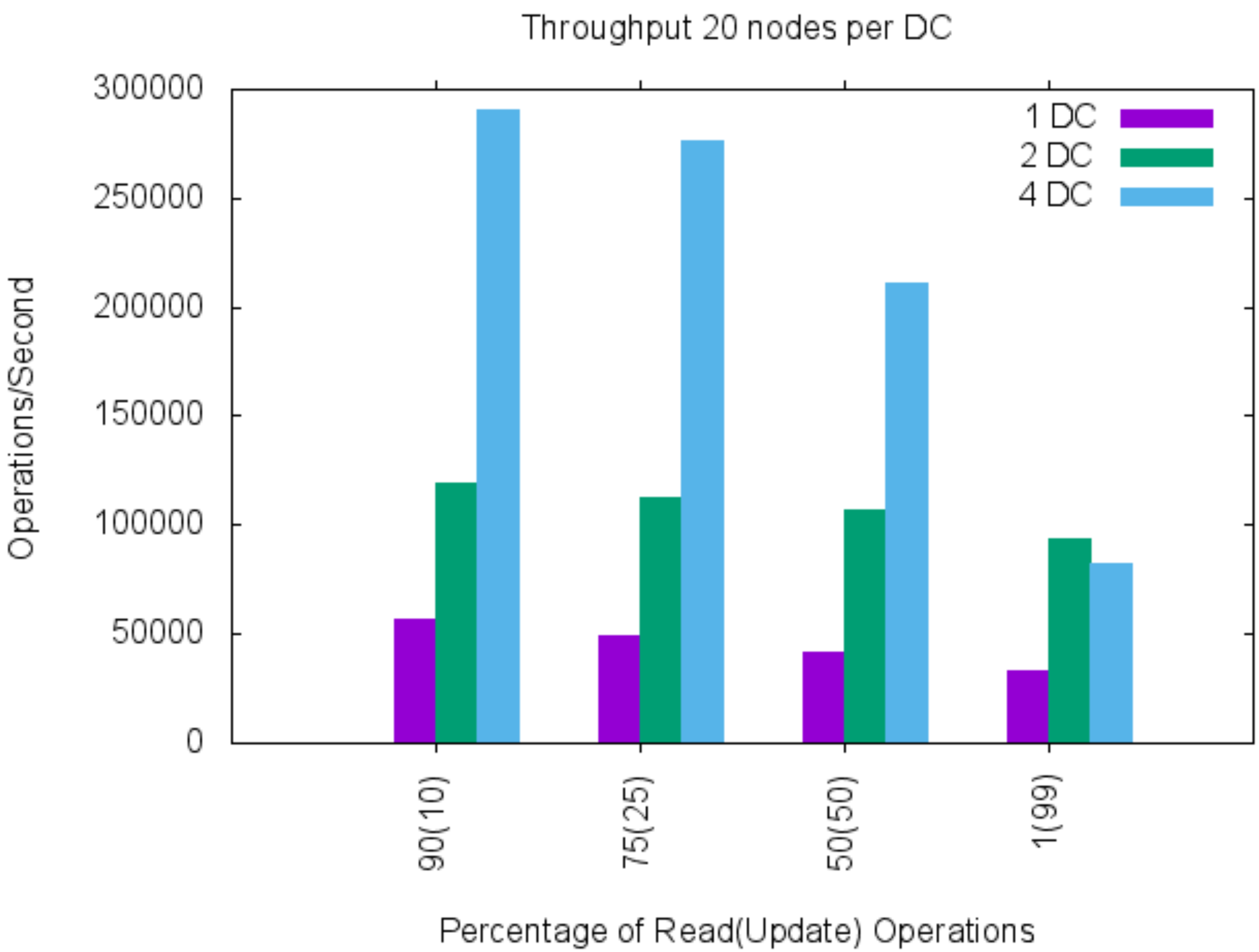


New hardware



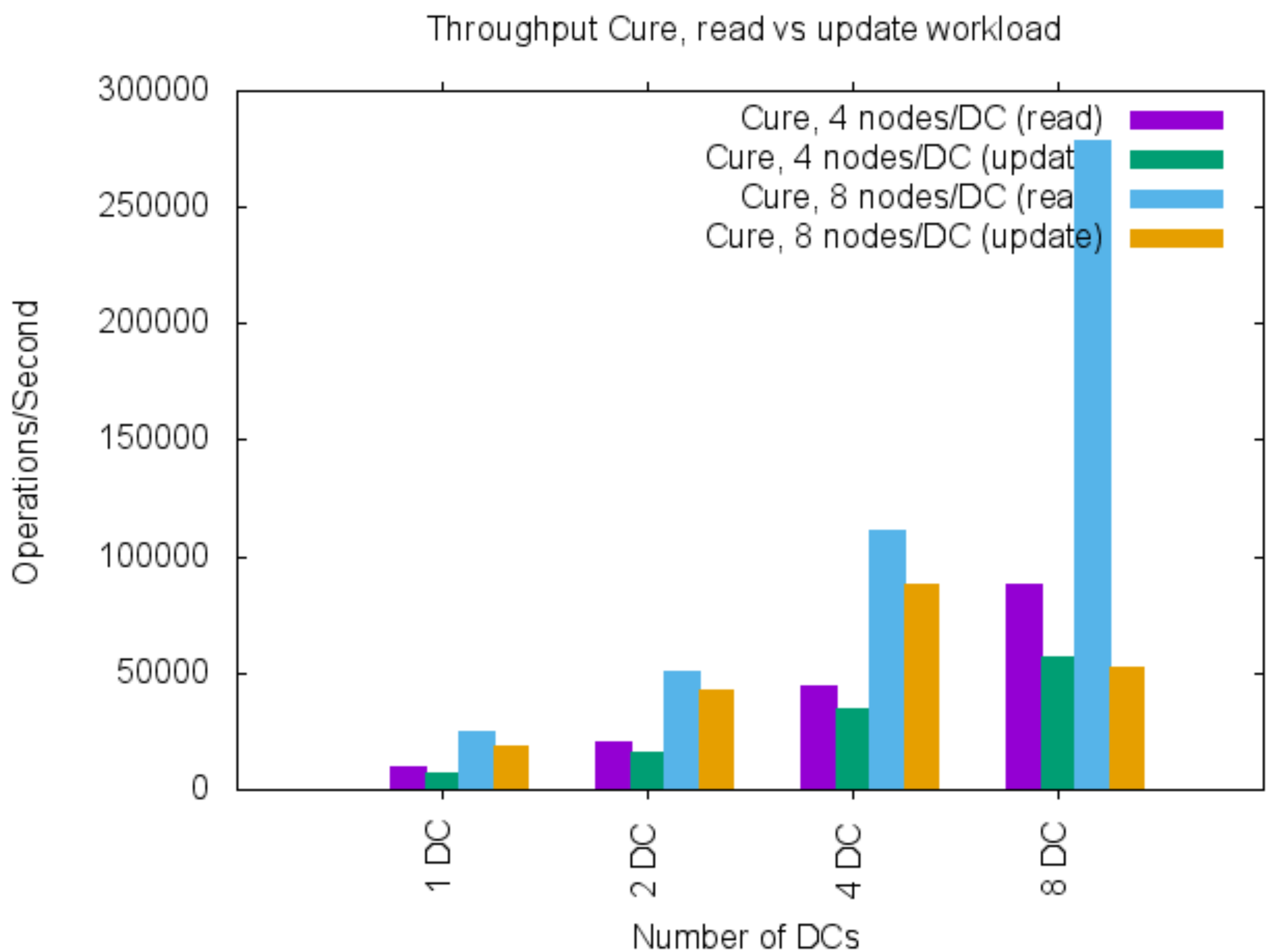
Old hardware

20 Nodes/DC



Throughput 20 nodes per DC
300000
250000
200000
150000
100000
50000
0
Operations/Second
1 DC
2 DC
4 DC
90(10)
75(25)
50(50)
1(99)
Percentage of Read(Update) Operations

4 or 8 Nodes/DC



More features

Transaction protocols

- Cure protocol to define snapshots and causal dependencies, geo-replicated version of ClockSI protocol [Akkoorath et al., ICDCS' 16]
- GentleRain uses global stable time mechanism [Du et al., SoCC' 14]
- Eiger protocol with explicit dependency checks, write-only txns [Lloyd, NSID' 13]
- Eventual consistency

Protocol buffer interface

- Uses PB encoding for efficient message transfer
- Connection via protocol buffer socket instead of RPC calls
- Supports working with local obj proxy at client side

Commit hooks

- Hooks are functions that are executed when updating an object

```
fun (update_object()) -> {ok, update_object()} | {error, Reason}.
 type update_object() :: { {key(), bucket()}, crdt_type(), update_op() }
 type update_op() :: {atom(), term()}
```

- Pre- / Post-commit hooks can be registered per bucket
- Before an object in the bucket is updated, pre-hook might modify the update operation
- Post-hook gets the (potentially modified) operation and executes before returning to client

```
register_post_hook(bucket(), module_name(), function_name())
      -> ok | {error, function_not_exported}.

register_pre_hook(bucket(), module_name(), function_name())
      -> ok | {error, function_not_exported}.

unregister_hook(pre_commit | post_commit, bucket()) -> ok.
```

Things on our agenda

- Upgrade to Erlang 19
- Flexible data storage backend
- Security: Access control
- Support for Just-Right consistency

Feedback welcome!

Sources

- Code repository

  https://github.com/SyncFree/antidote

- Documentation

  http://syncfree.github.io/antidote

- EU-Project Syncfree

  https://syncfree.lip6.fr

The Antidote Team

Deepthi Devaki Akkoorath, Alejandro Tomsiç, Manuel Bravo, Zhongmiao Li, Tyler Crain, Annette Bieniusa, Nuno Preguiça, Marc Shapiro, Christopher Meiklejohn, Michał Jabczynski, Santiago Alvarez Colombo, Mathias Weber, Peter Zeller, Ruma Paul, …

Interested in collaboration?

- Tired of fixing inconsistencies in your data?
- You want to adapt your system to use Highly Available Transactions?
- Need cutting edge research on consistency, availability, edge/fog storage,…?

Contact us!

Open positions

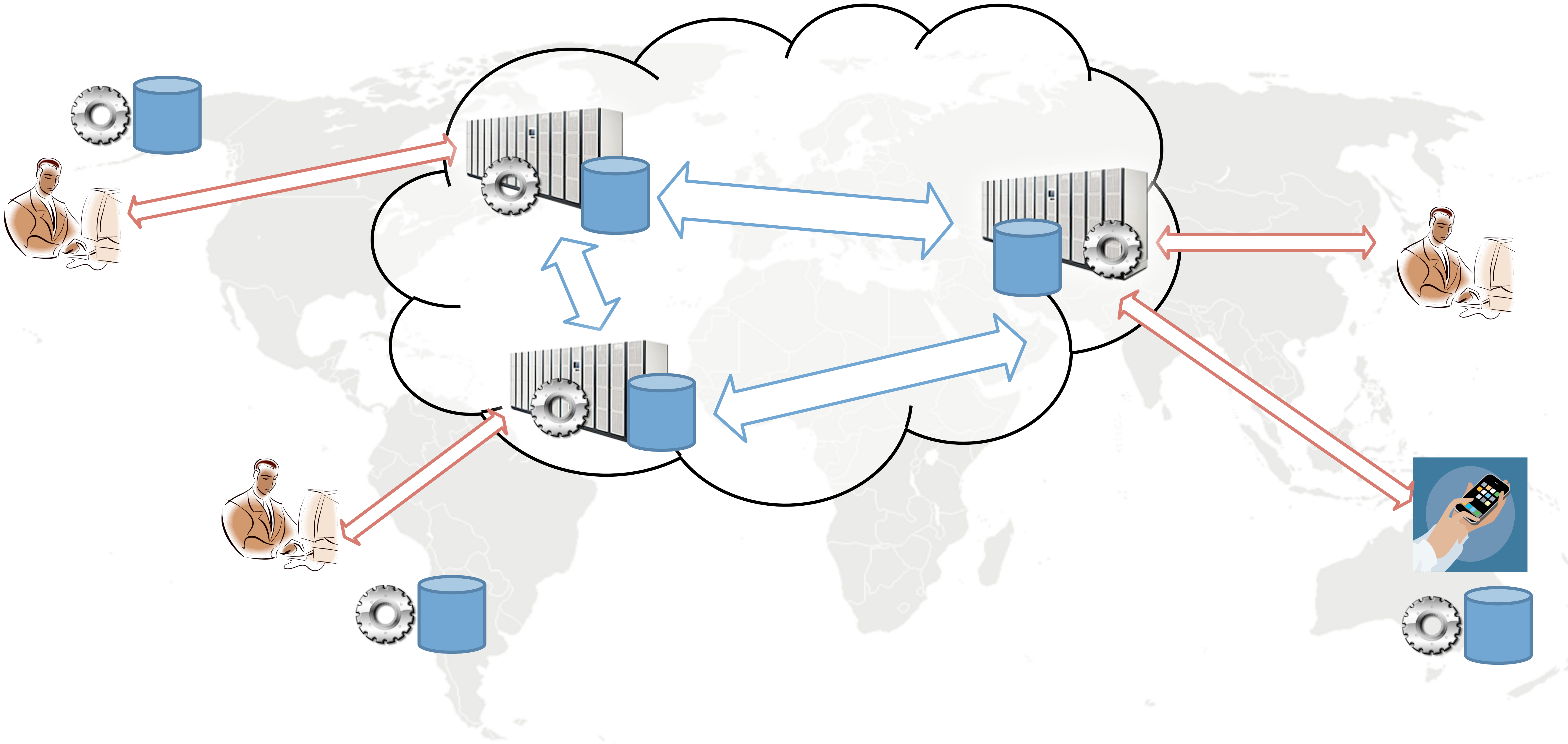
Just-Right Consistency database of the future

Software Engineer position
Post-doc position
and several PhD Positions

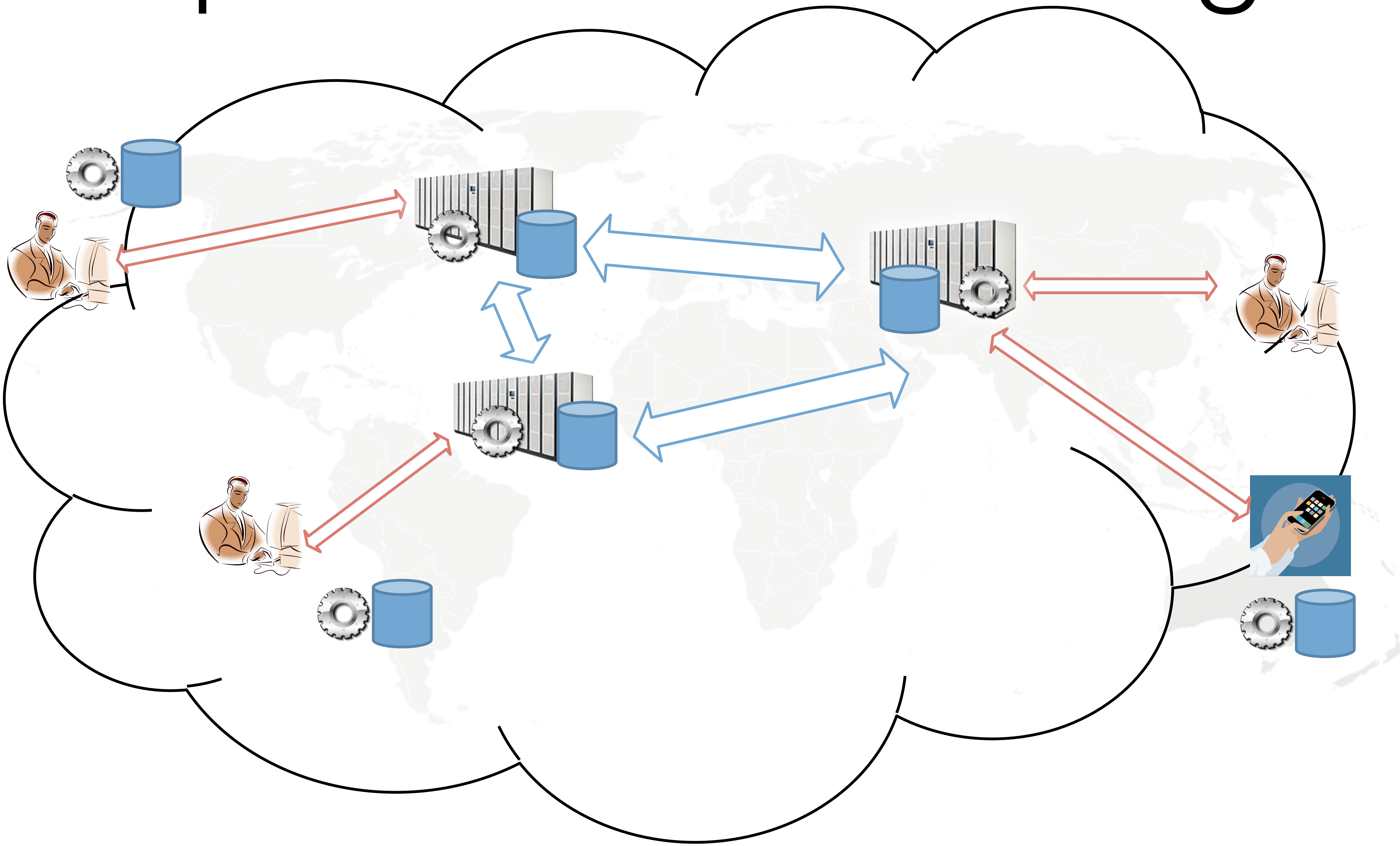
https://team.inria.fr/regal/job-offers/

Inria
INVENTEURS DU MONDE NUMÉRIQUE

Geo-replication



Replication to the edge



Antidote protocol and platform

- **Available** - A transaction can execute as long a copy of the objects accessed
- **Low latency** - Replication, transactions execute in local DC
- **Scalability**
  - Distributed - Transactions only touch the servers that replicate the objects they access locally
  - Meta-data size of O(DCs) (ClockSI used to total order DC)
    - Per DC background stabilisation mechanism (to ensure causal dependencies) (inspiration from GentleRain)

In-DC Total Order

In Data Centre

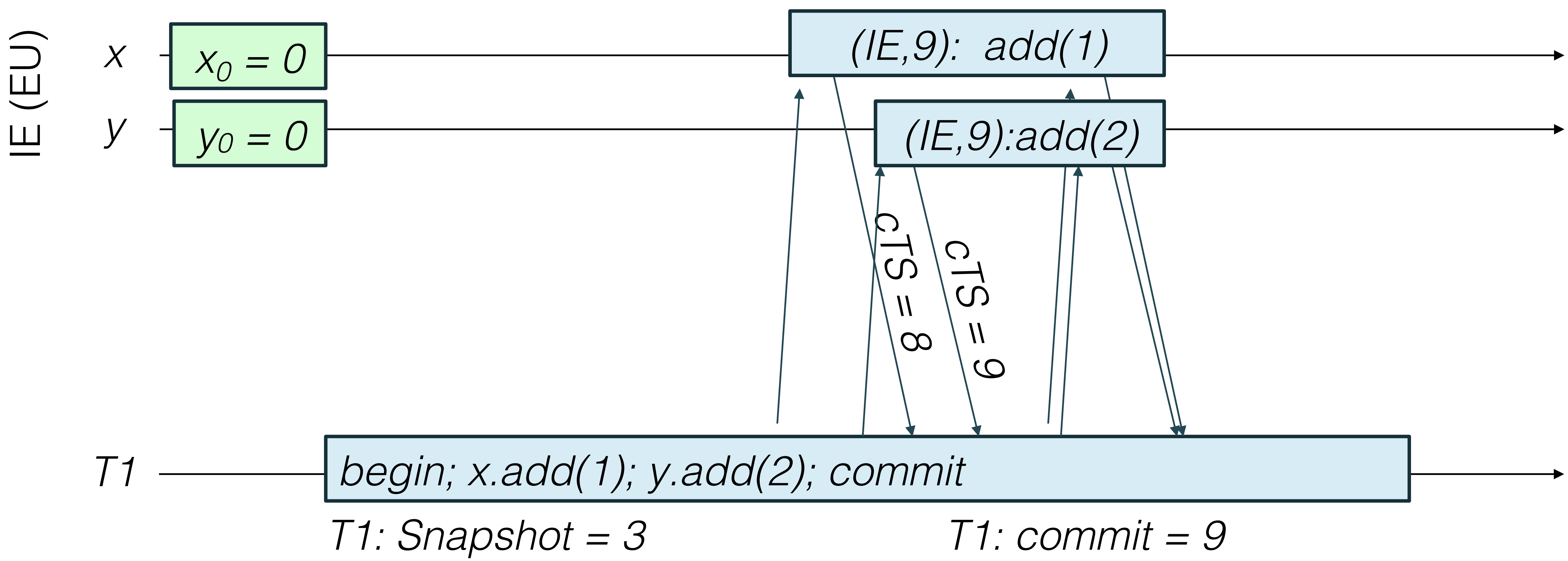
- Internal parallelism: sharding, disjoint transactions
- Abstract view: sequential
- Low cost, footprint
- Snapshot Isolation (SI)
  - All-or-Nothing Transactions
  - Writes are totally ordered
  - Reads are consistent, decoupled from writes
  - Read-only transactions are free

Clock-SI

[Du, SRDS 2013]

- Loosely-synchronized clocks, 1 / shard
- Data items versioned by timestamp
- Transaction
  - Read timestamp: coordinator's current clock ⟹ snapshot includes all earlier txns
  - Commit timestamp: 2PC ⟹ *max* (shards' clocks)
  - Snapshot: consistent
  - Writes: all-or-nothing, total order (per DC)
- Disjoint-access parallel (GPR)
- Read-only txns, single-server txns: free

Clock-SI protocol



Local total order: useful, acceptable

- 2PC disjoint-access parallelism
- in DC communication is fast
- 1 scalar / DC

Geo-replicated Clock-SI

Clock-SI: updates originating in a DC are totally ordered, all-or-nothing

Version vector

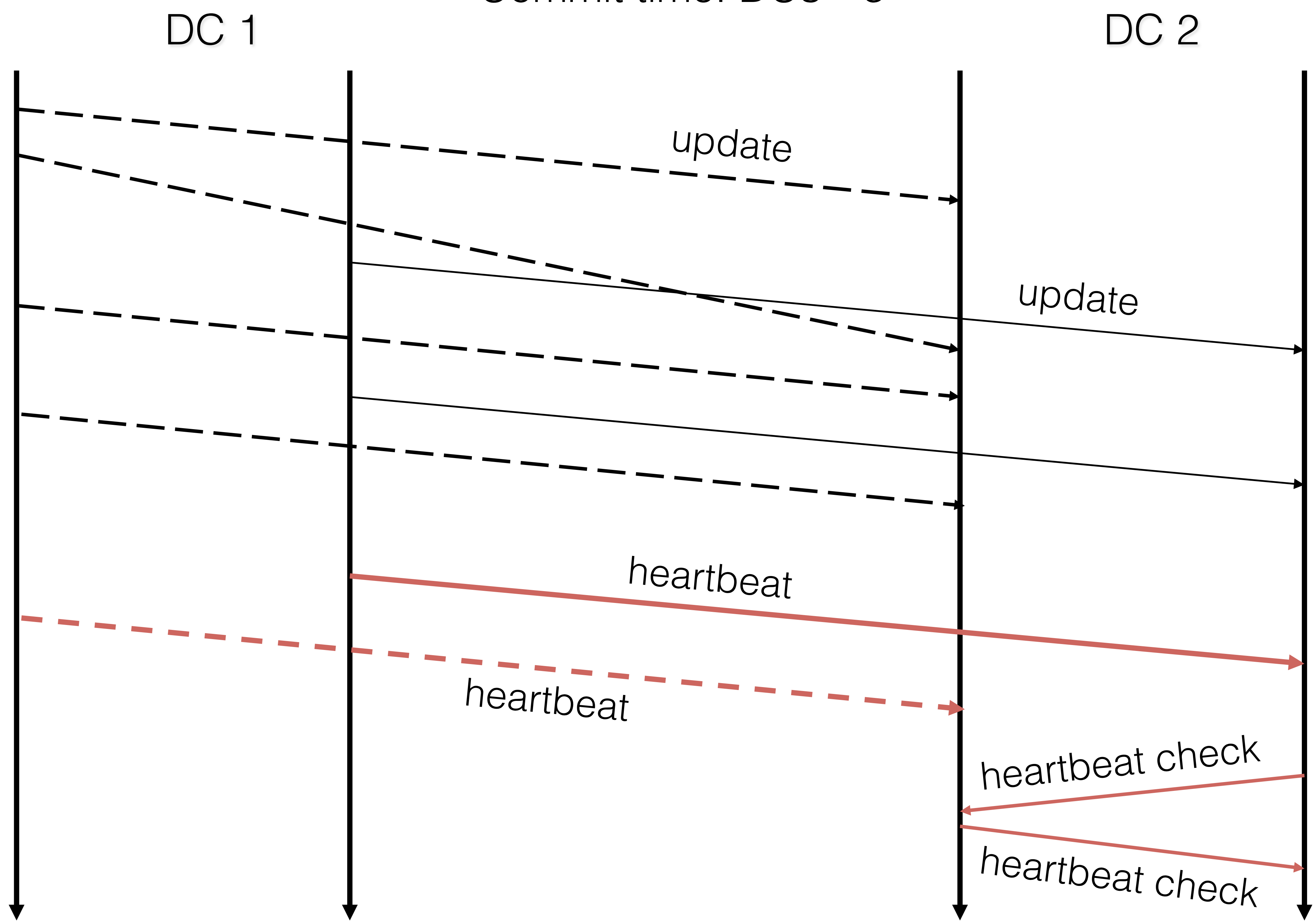
- 1 entry / DC
- Private to partition

Other DC's updates visible once all partitions sync'd

Heartbeat msg guarantees progress

DC1	DC2	DC3	DC4	DC5
2	3	7	5	3

Commit time: DC3= 8



1. Send heartbeat from each server with its local clock
2. Calculate minimum
   - All updates up to that value have been received from that DC