Write Fast, Read in the Past:

Scalable Consistency all the way to the client

Marc Shapiro

Joint work with:

Marek Zawirski, Inria & LIP6
Annette Bieniusa, U. Kaiserslautern
Valter Balegas, U. Nova Lisboa
Sérgio Duarte, U. Nova Lisboa
Carlos Baquero, U. Minho
Nuno Preguiça, U. Nova Lisboa

informatics mathematics

Acknowledgments

Association for
Computing Machinery

Distinguished Speaker Program

SyncFree

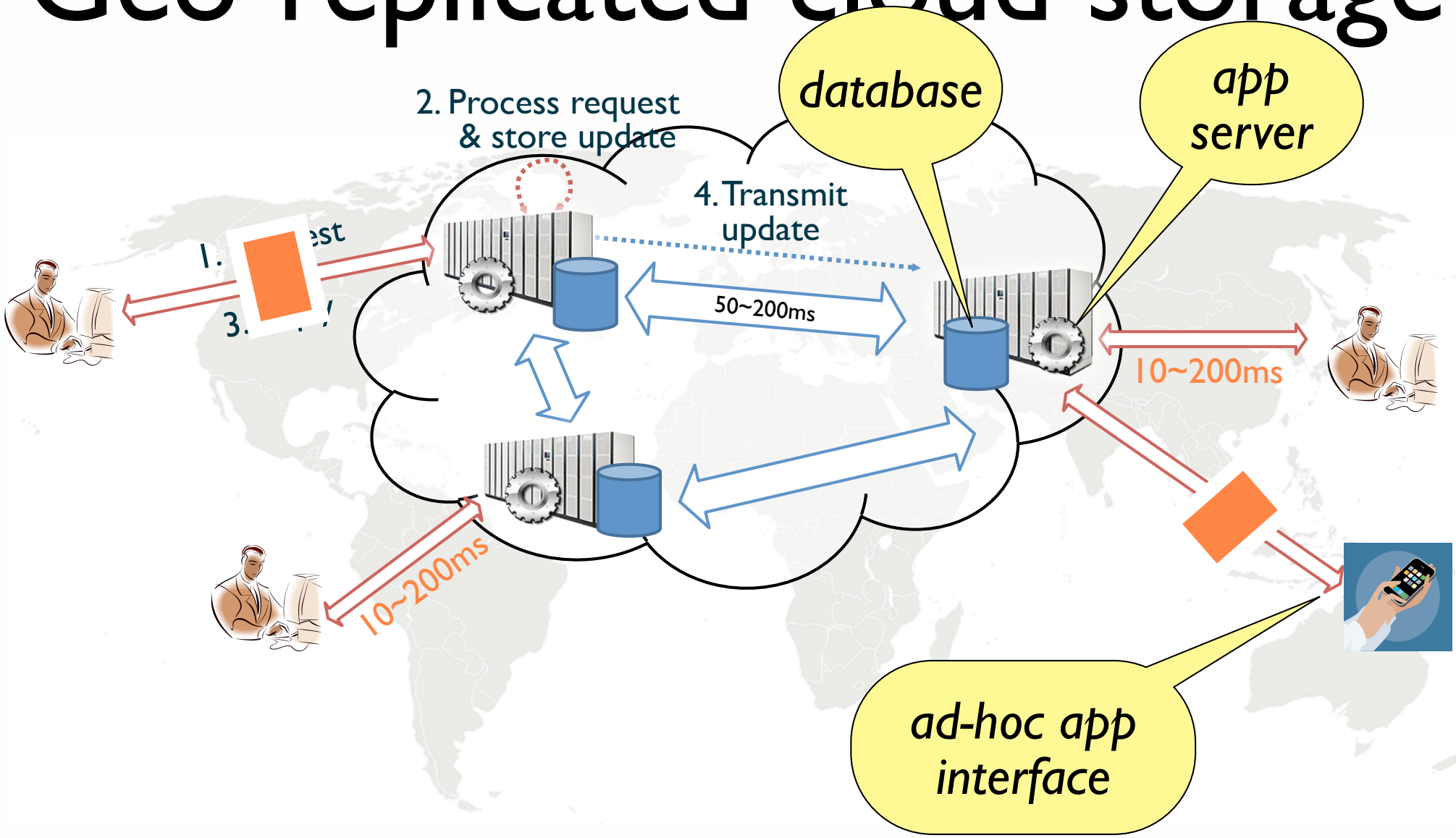
Google

ConcoRDanT

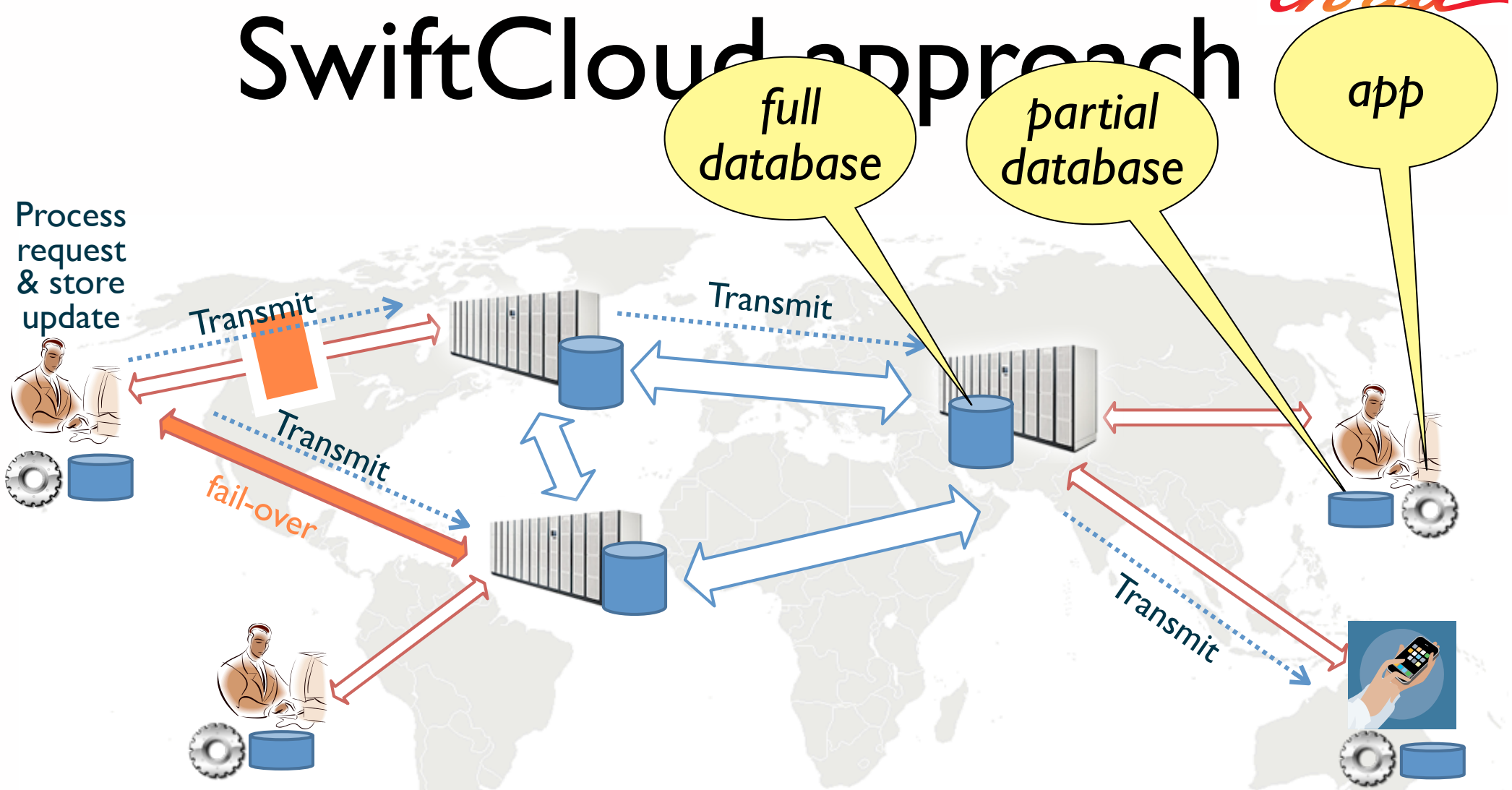
PROJET FINANCÉ PAR L'ANR - PROJECT FUNDED BY THE ANR

ANR

Geo-replicated cloud storage



SwiftCloud approach



App + database at/near client
Update shared store locally
Availability + consistency

Requirements & challenges

Large-scale replication

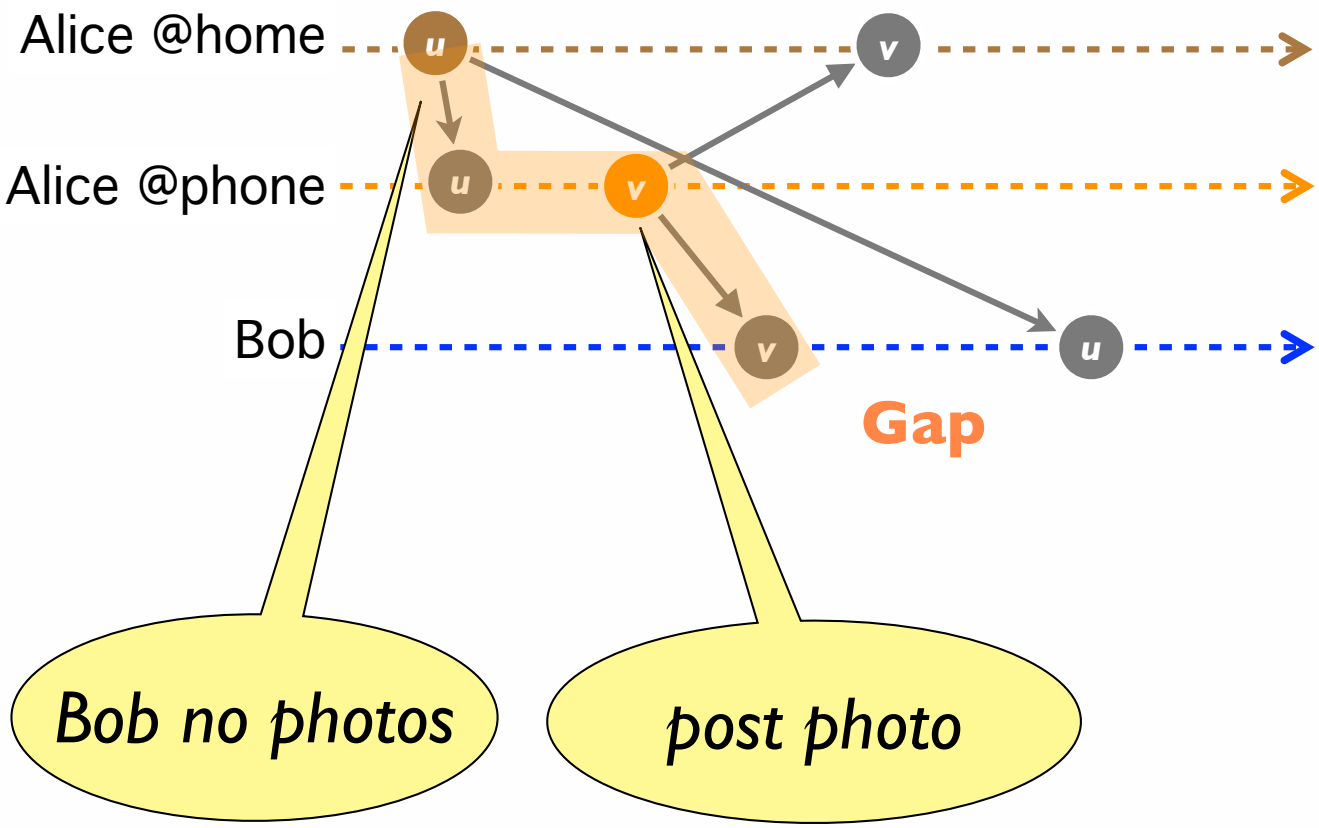
- Partial replication of data
- … and of metadata
- Small, bounded metadata

High availability □ causal consistency

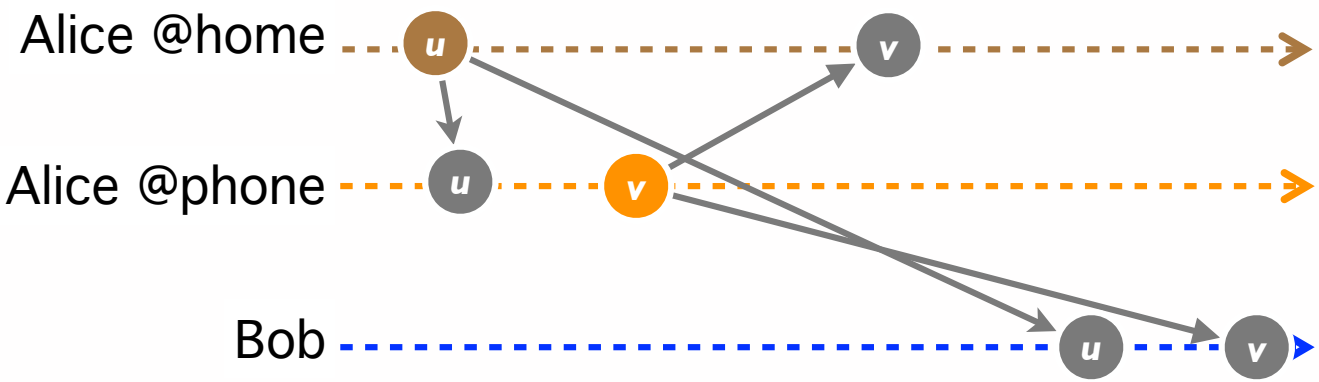
- Eventual exactly-once delivery
- Causal order versions
- … despite partial replication
- … despite transient failures
- … despite permanent failures

Convergence

Causal gap


Alice @home
Alice @phone
Bob
u
v
u
v
v
u
Gap
Bob no photos
post photo

Causal Consistency



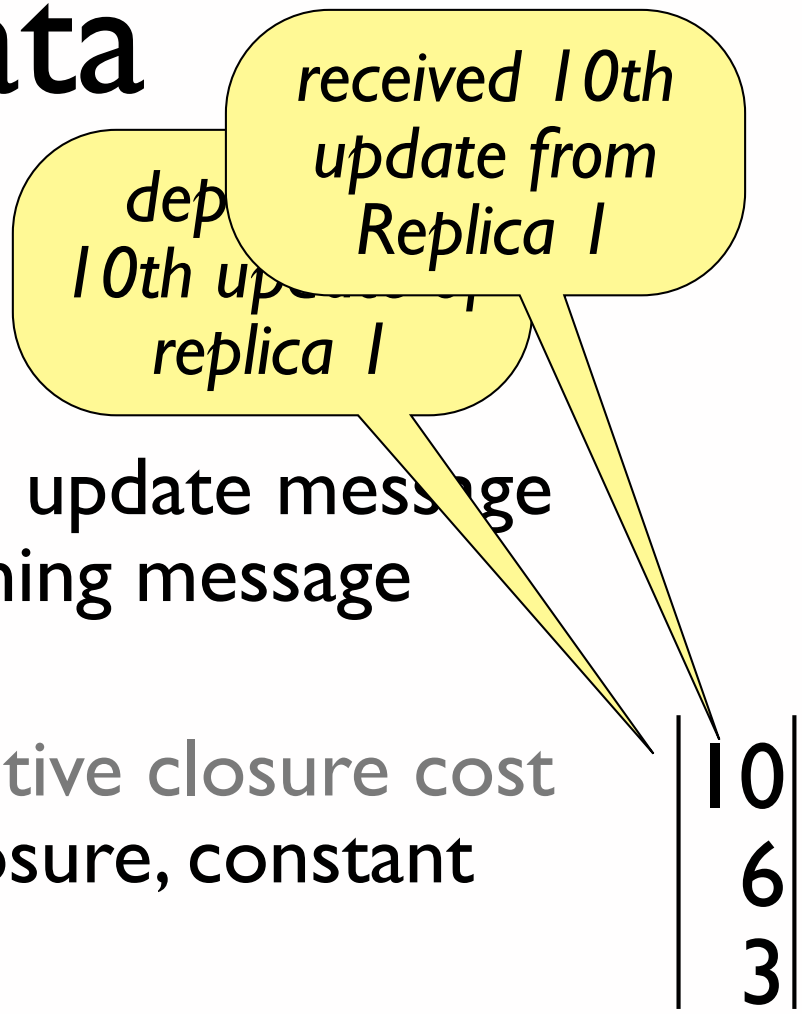
v observed effects of *u* at source
□ *v* to be delivered after *u*

No gaps

Transitive closure

Strongest always-available consistency

Metadata



Tracking causality

- Dependence: piggy-backed on update message
- Delivery: compare with incoming message

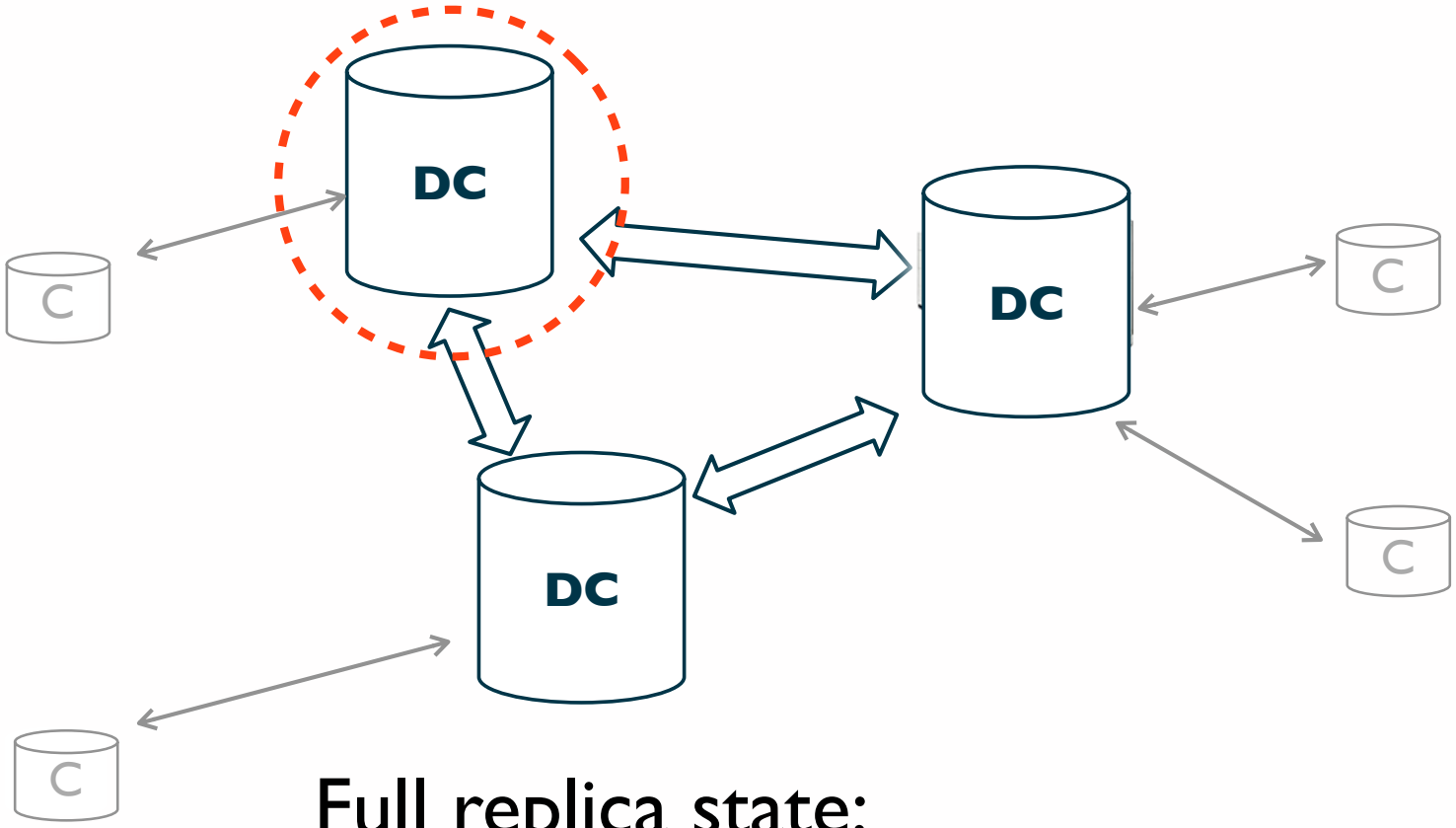
Representations:

- Graph: worst-case cost, transitive closure cost
- Vector: compact transitive closure, constant cost + optimisations

Issue: size(vector) = O (#masters)

Log: pruning not contingent on clients

Causal consistency at DC



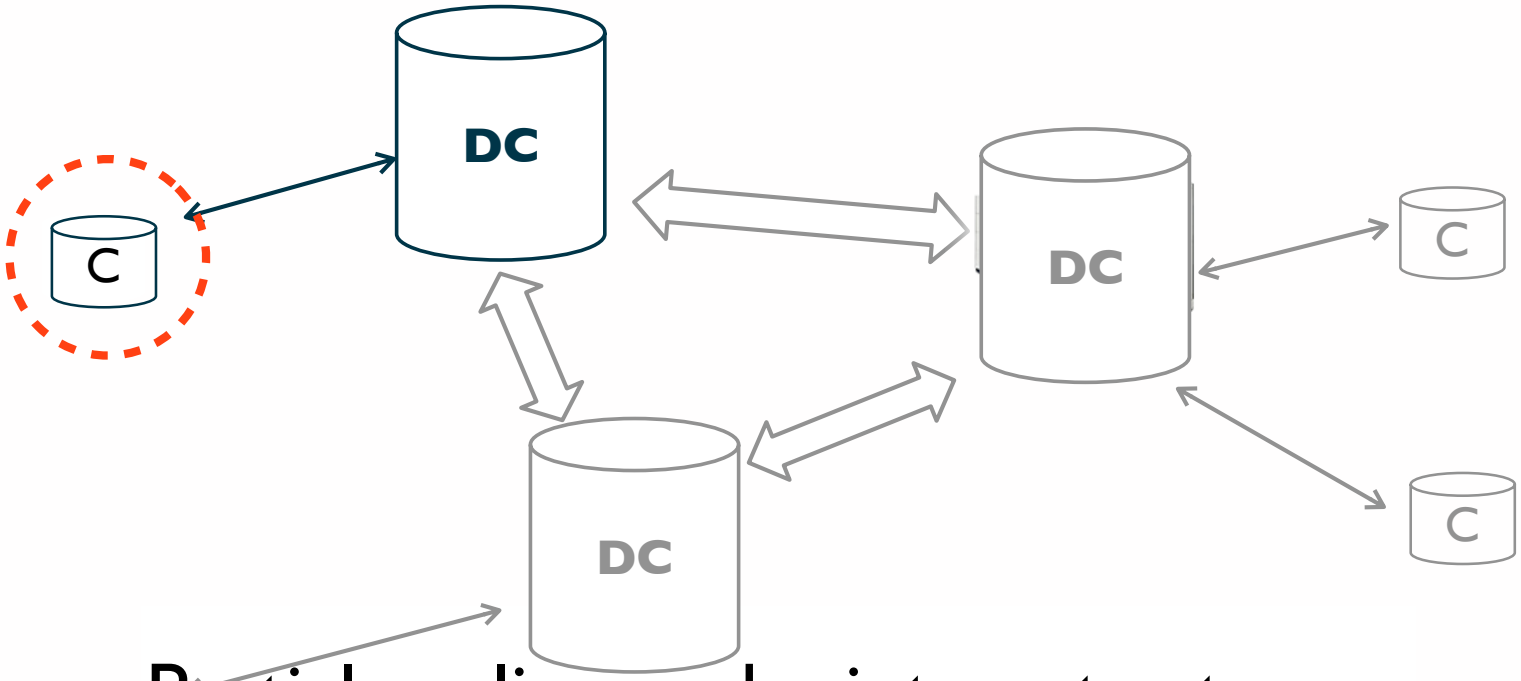
Full replica state:

- Causally consistent
- Transitively closed

Vector clock

- size = O(#DCs)

Causal consistency at client



Partial replica: cache interest set

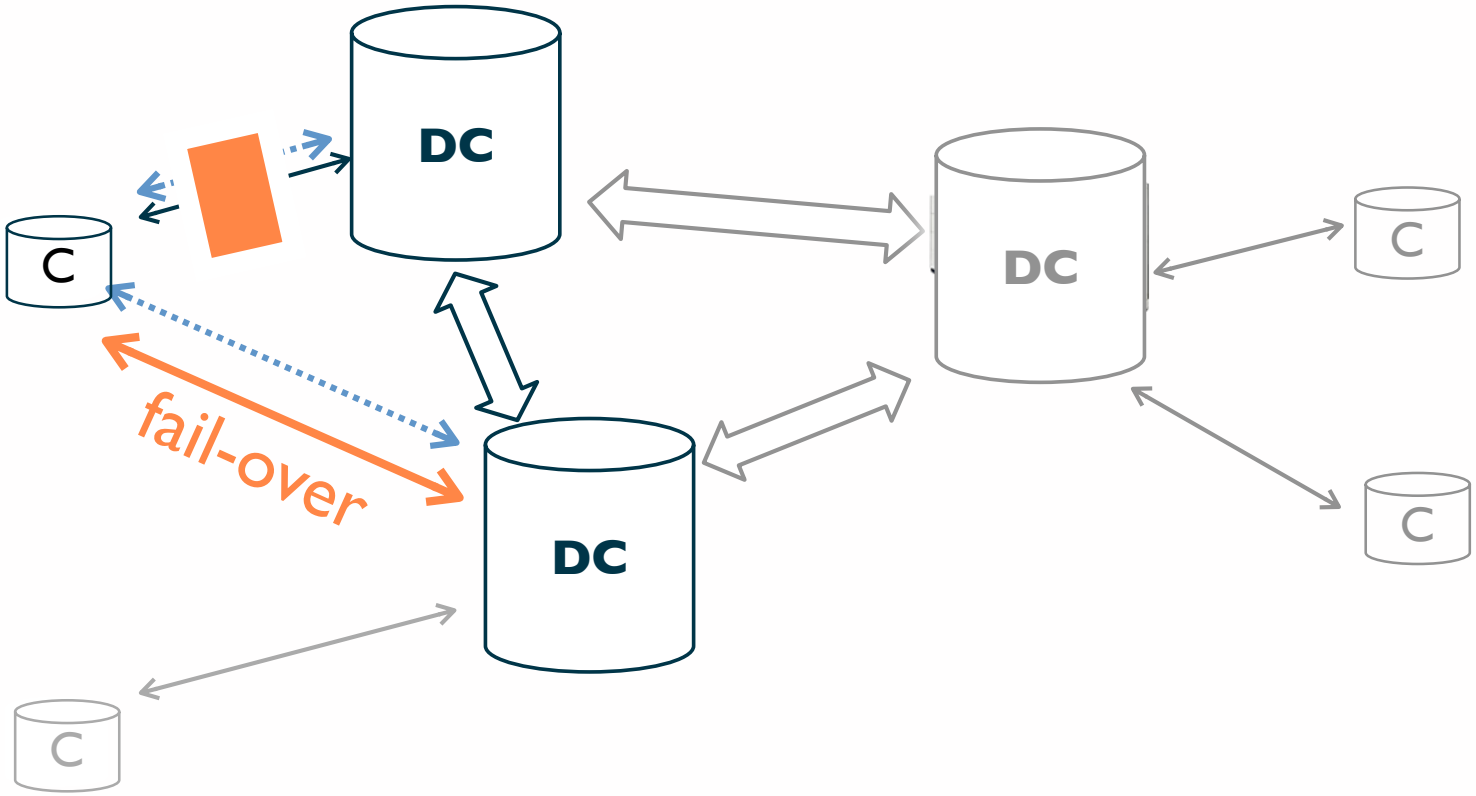
Communicates with DC only

State = DC state $|_{\text{interest set}}$

$\cup$ client updates

Size(vector) = O(#DCs)!

Causal consistency & fail-over

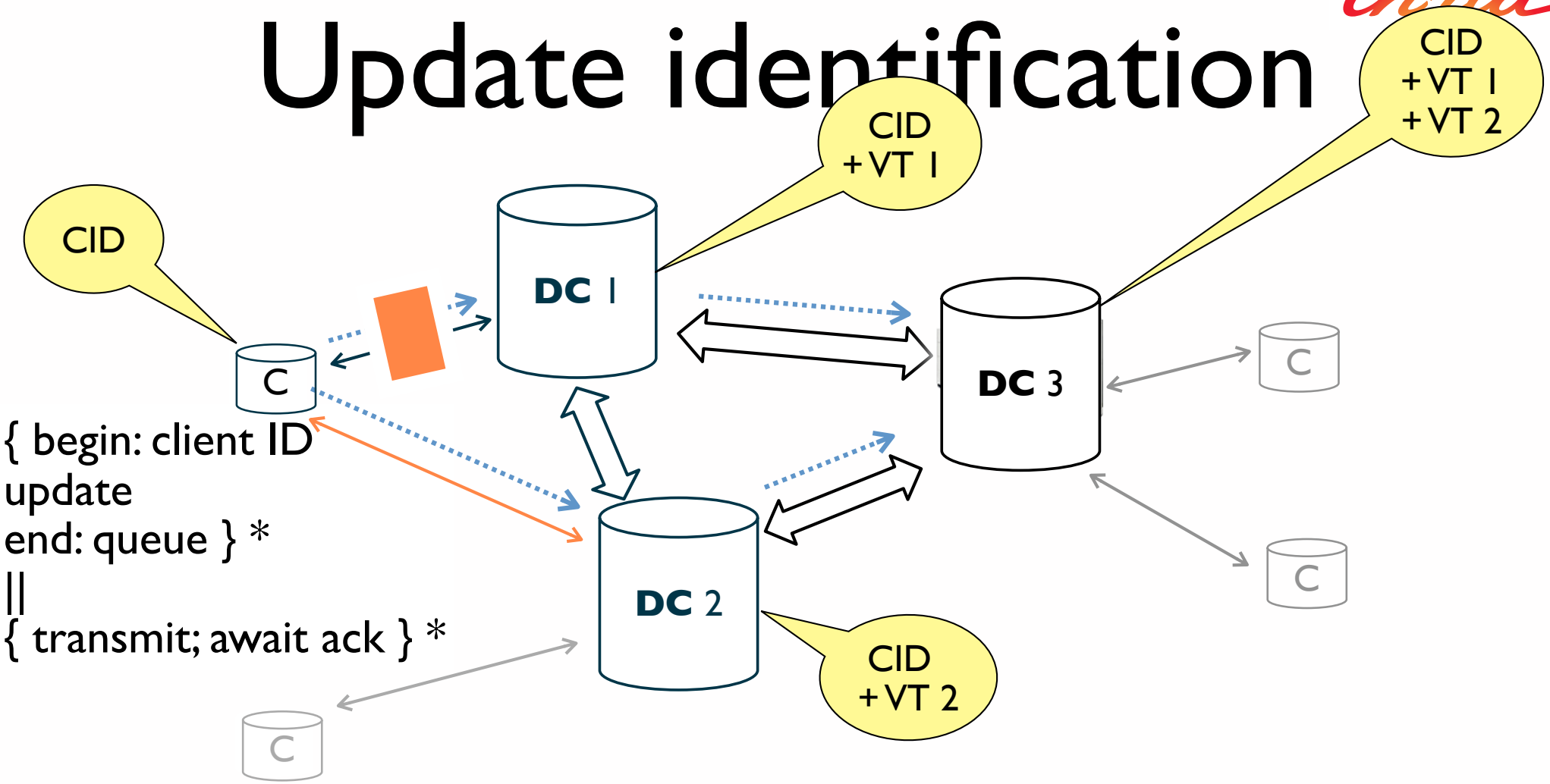


At least once: stubborn

At most once: filter out duplicates

Consistency with new DC

Update identification



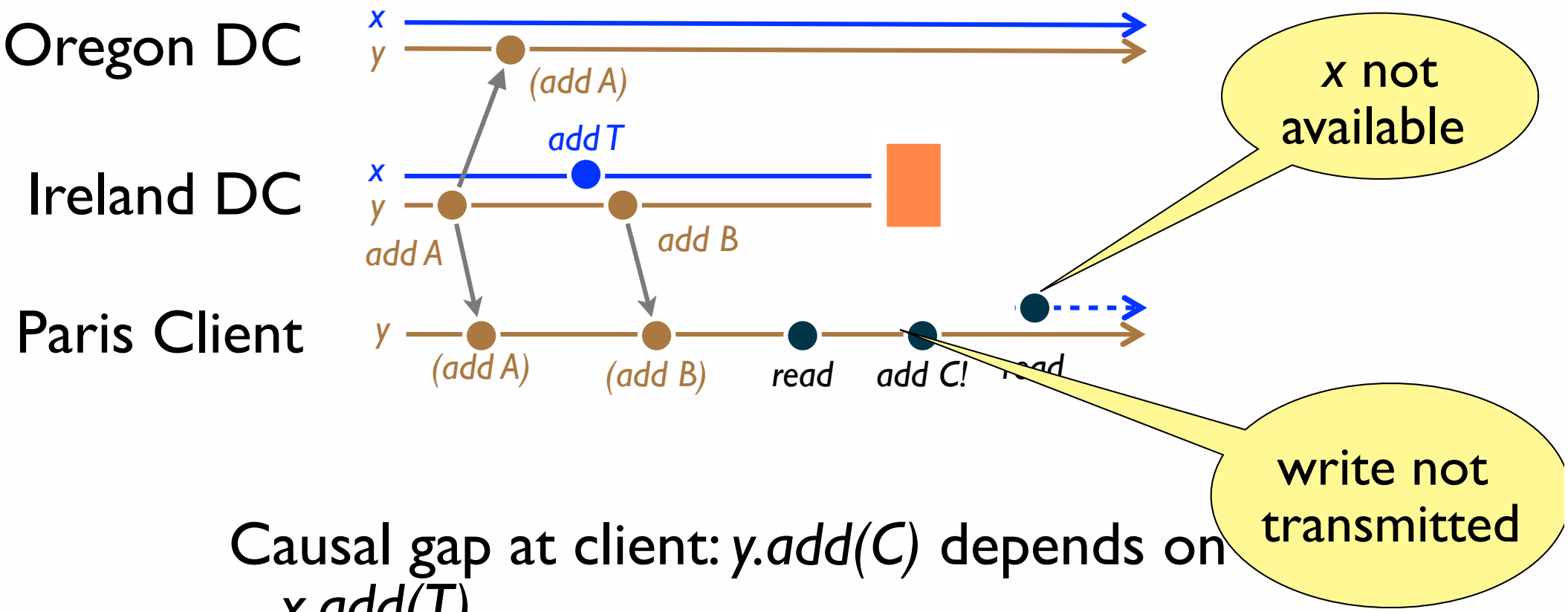
{ begin: client ID
update
end: queue } *
||
{ transmit; await ack } *

Causal order: Vector timestamp VT

No duplicates: client-assigned unique ID CID

Union of vector timestamps

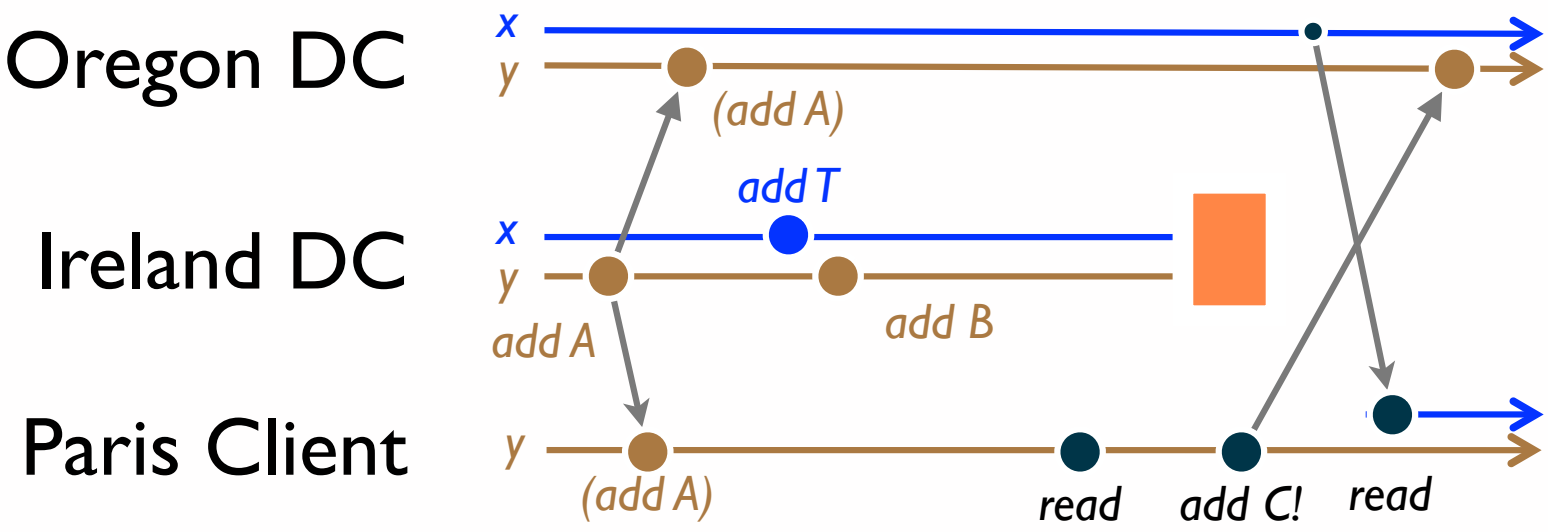
Inconsistency with new DC



Causal gap at client: *y.add(C)* depends on *x.add(T)*

Oregon cannot satisfy: fail-over impossible; client blocks until Ireland recovers

Solution: reading in the past



Client reads only updates that are *K-stable* □ no gaps

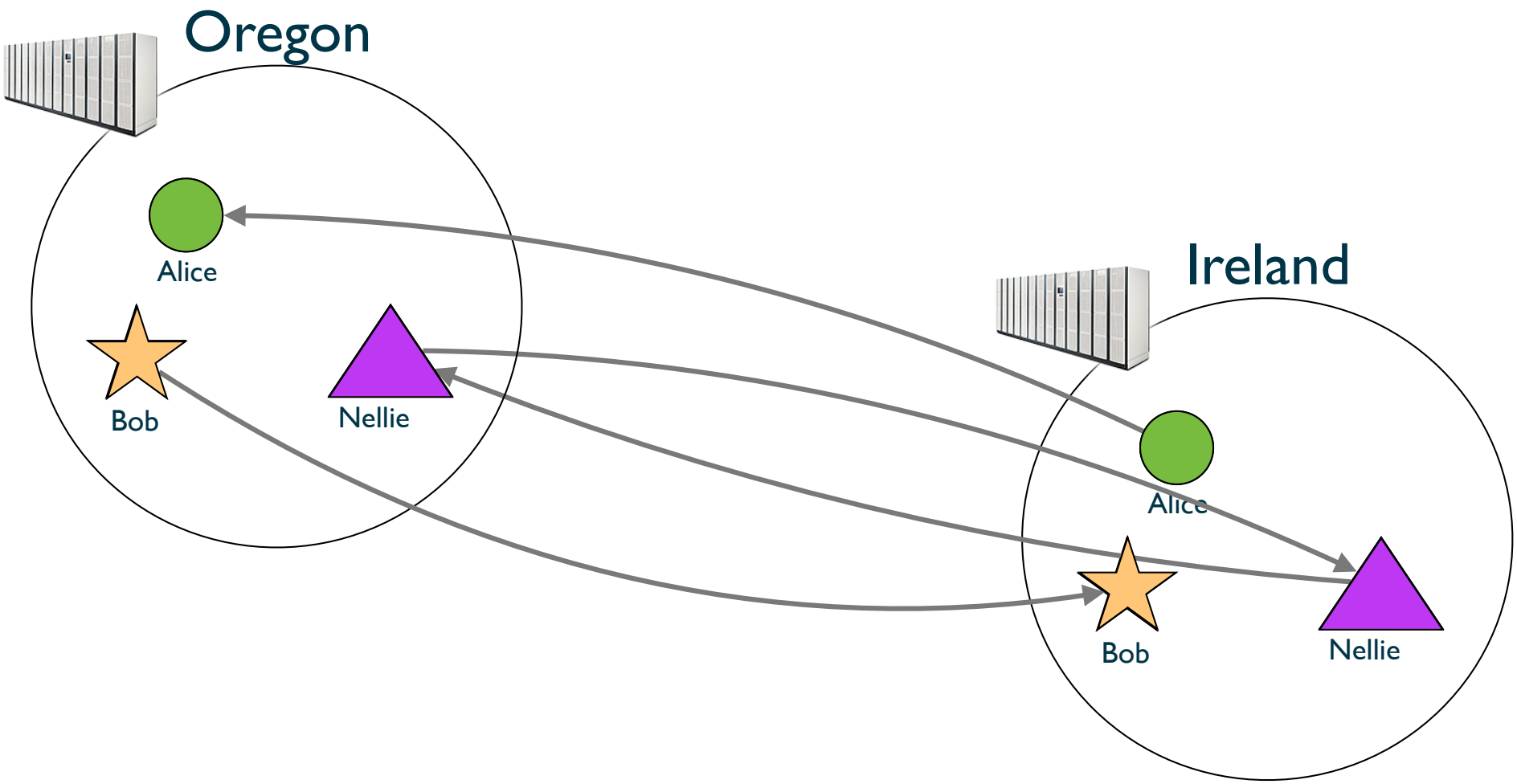
- *K=1*: always fresh, fail-over problematic
- *K=N*: fail-over guaranteed, staleness
  - updates possibly buffered indefinitely

Default: K=2, covers common failures

Also improves response time

[Write Fast, Read in the Past: Scalable Consistency all the way to the client]

Converging concurrent updates



Convergence by construction

Merging concurrent updates:

- Deterministic
- Dependent only on delivery poset
- Not on delivery order, local info

Sufficient conditions:

- Delivered updates commute
- (or) Monotonic semi-lattice

CRDTs

Conflict-free Replicated Data Types (CRDTs)

Encapsulate replication
& resolution semantics

Converge by construction

Register
- Last-Writer Wins
- Multi-Value

Set
- Grow-Only
- 2P
- Observed-Remove

Map

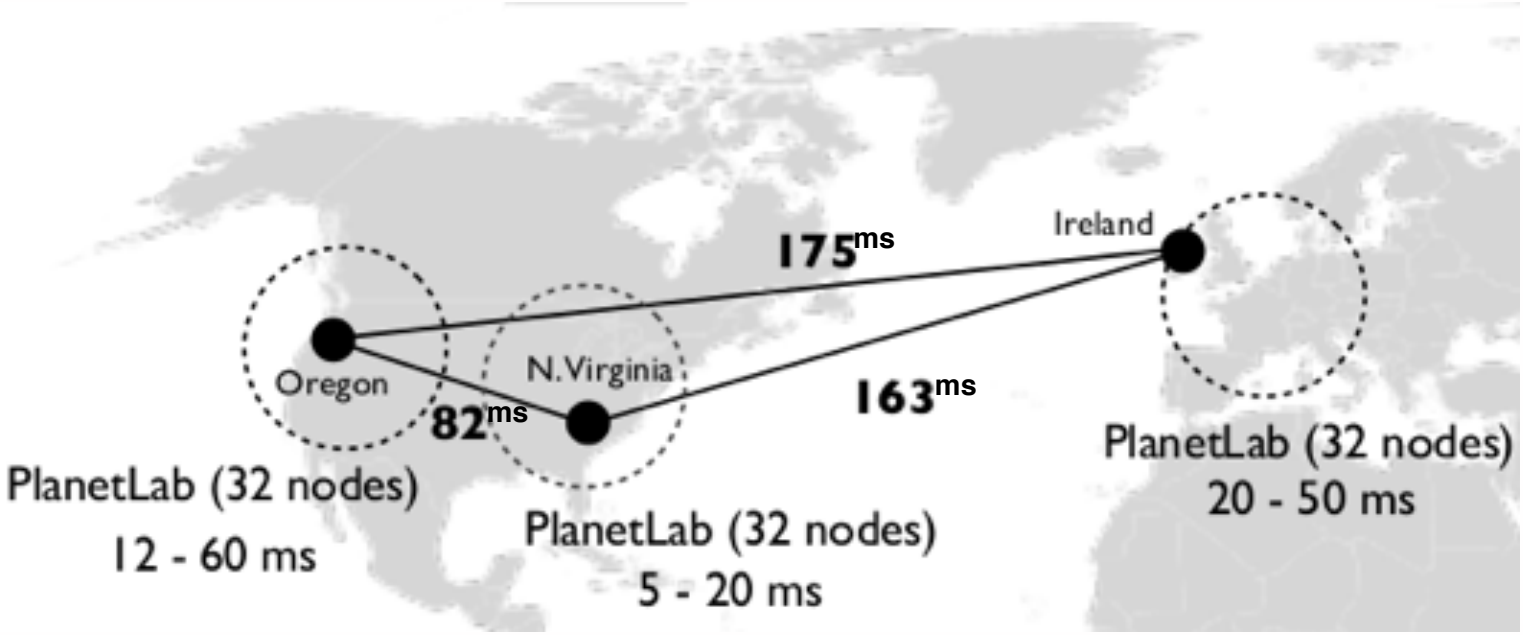
Counter
- Unlimited
- Restricted non-negative

Graph
- Directed
- Monotonic DAG
- Edit graph

Sequence

[Write Fast, Read in the Past: Scalable Consistency all the way to the client]

Experiments

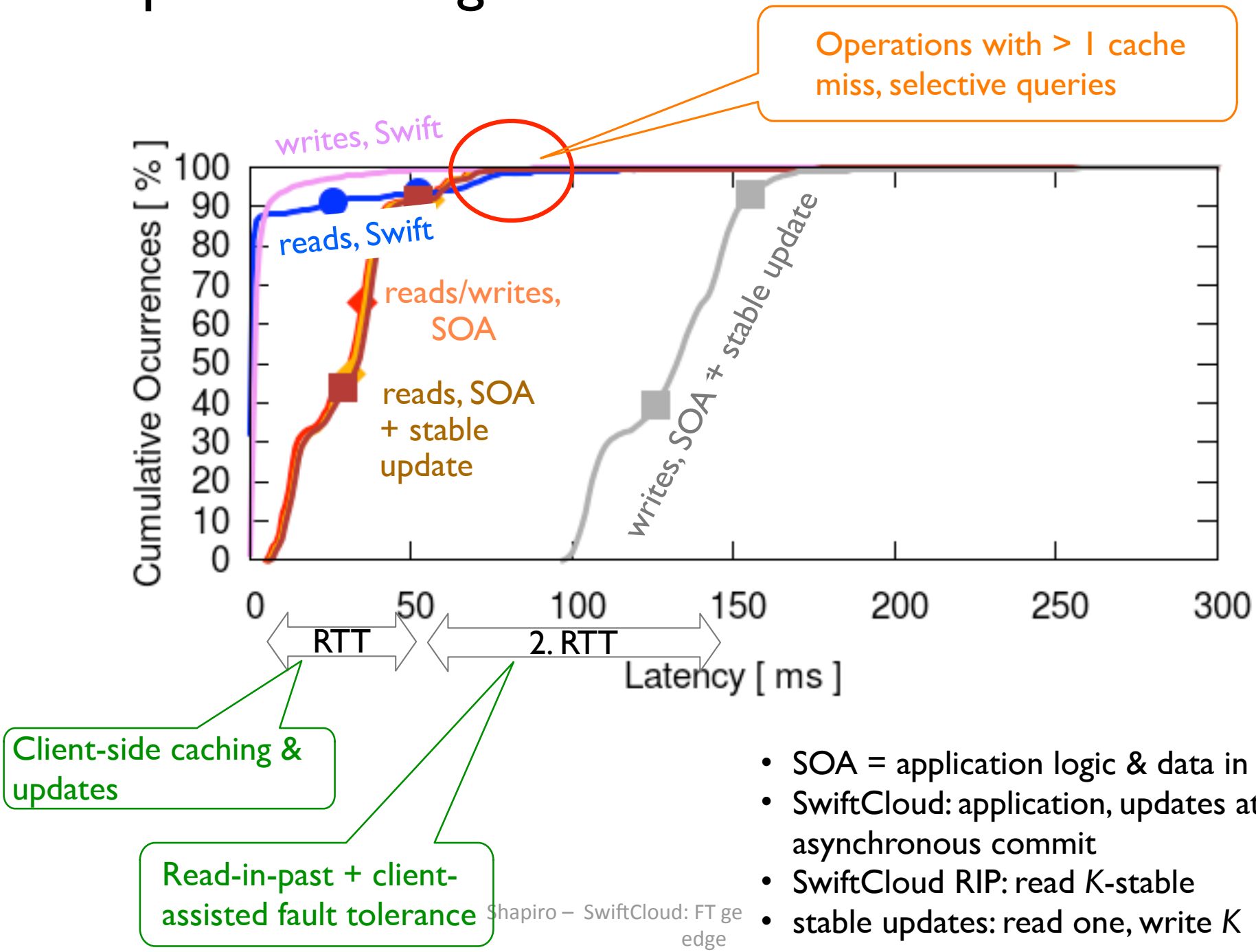


Social networking application
- 90% cache hits

Evaluate
- Availability, responsiveness
- Size of metadata
- Latency, throughput

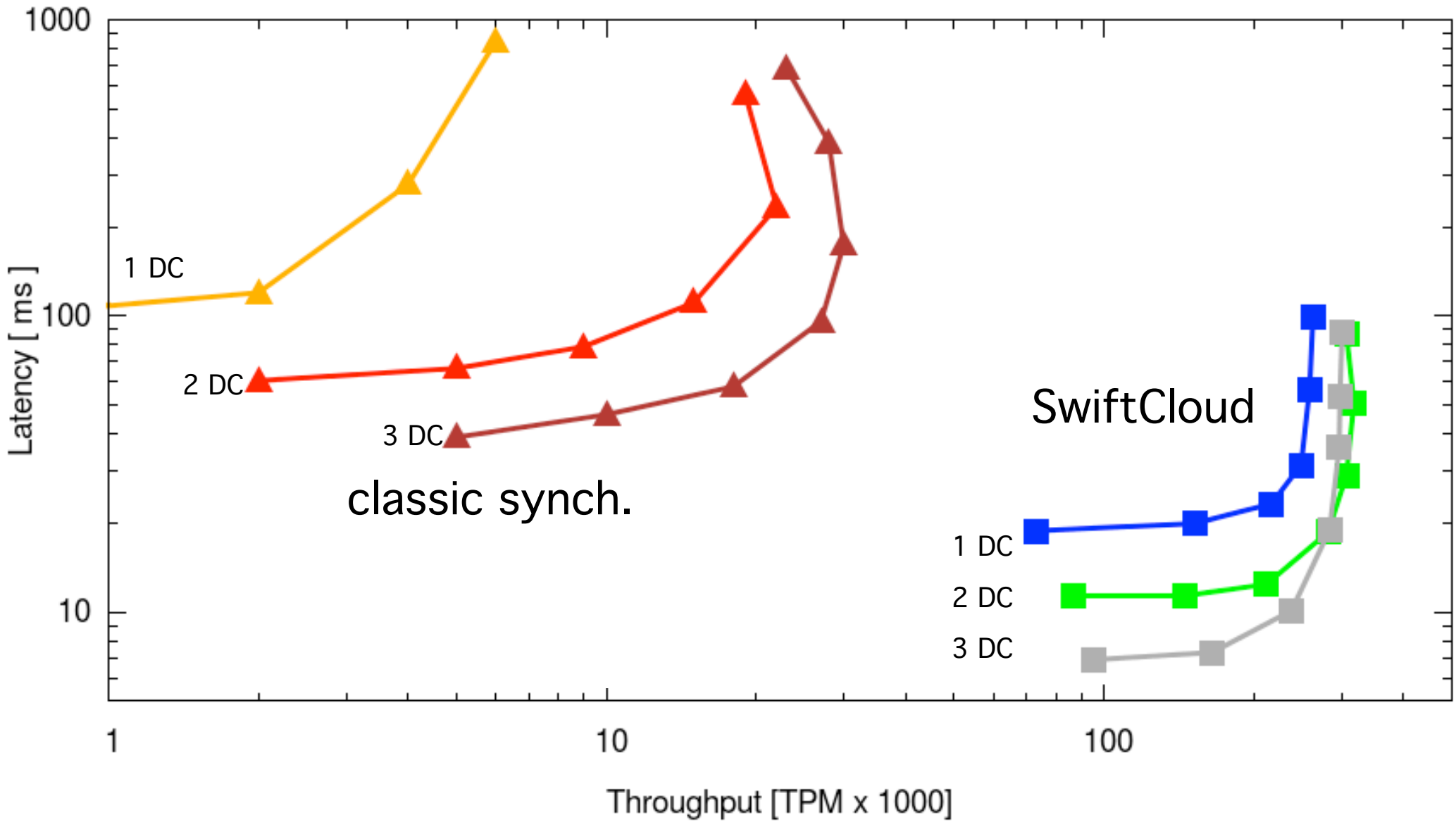
3 DCs in Amazon EC2
1 DC = 1 EC2 medium instance
100 client nodes in PlanetLab
Multiple sessions per node
Cache size: 512 objects

Update caching + Read-In-Past minimize latency

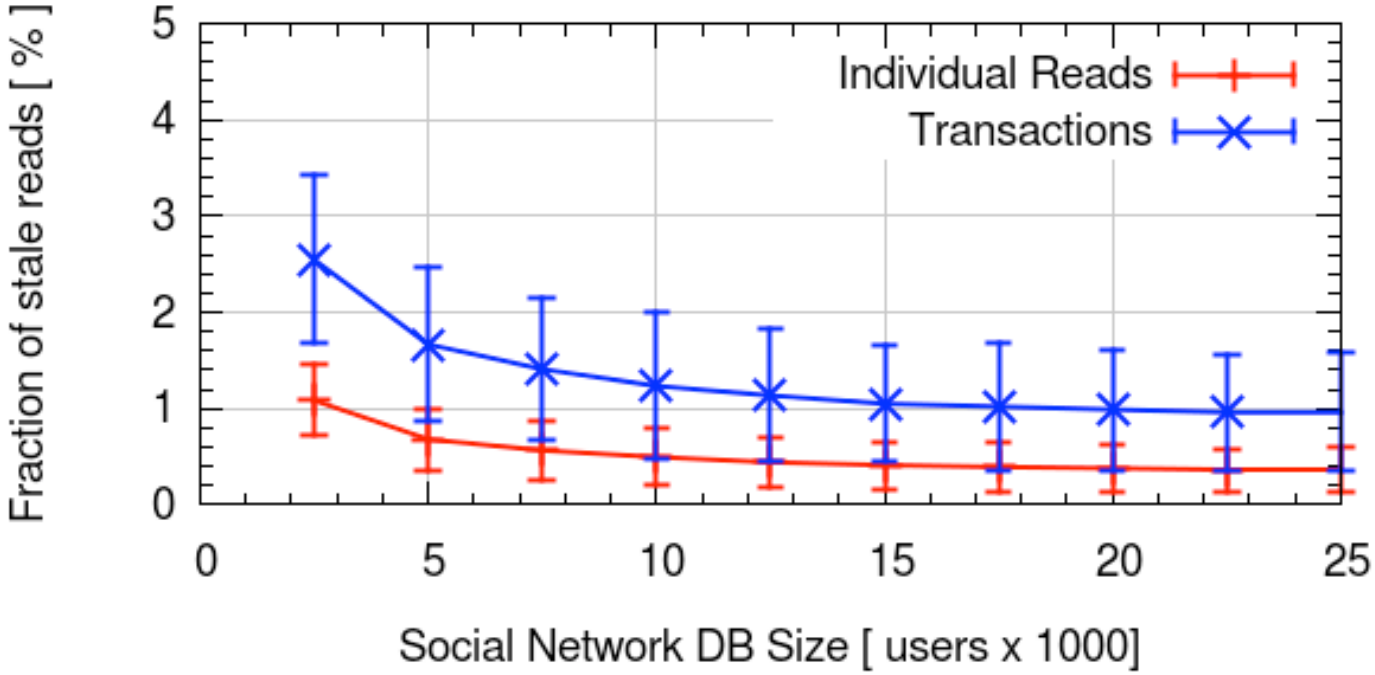


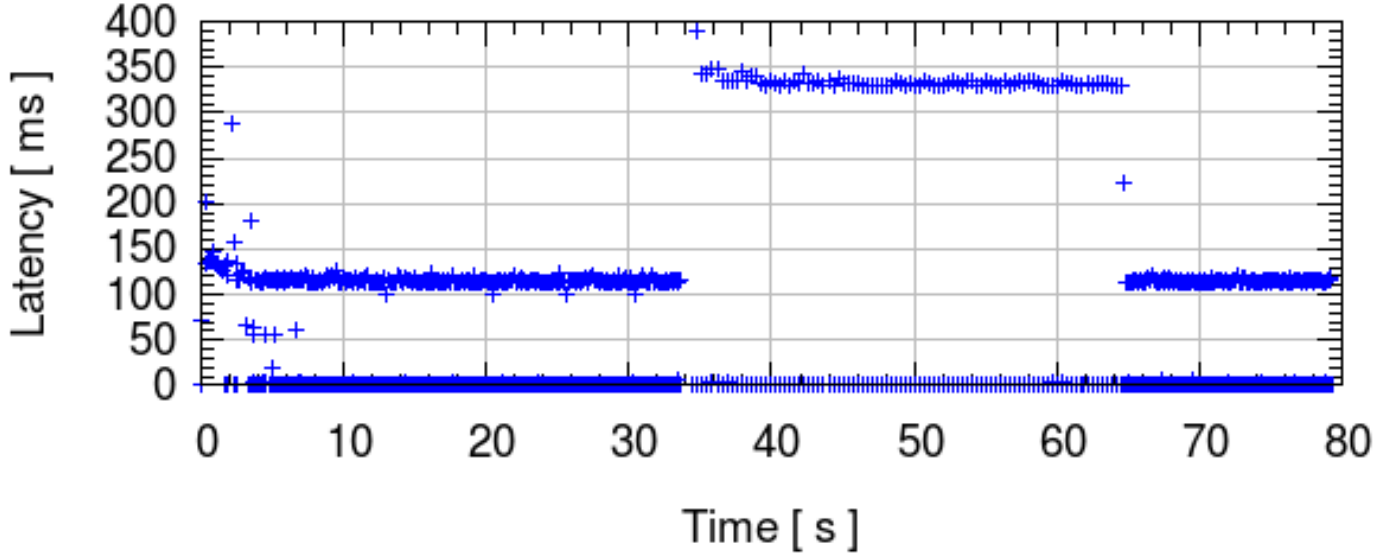
- SOA = application logic & data in DC
- SwiftCloud: application, updates at client, asynchronous commit
- SwiftCloud RIP: read *K*-stable
- stable updates: read one, write *K*

Latency vs. throughput

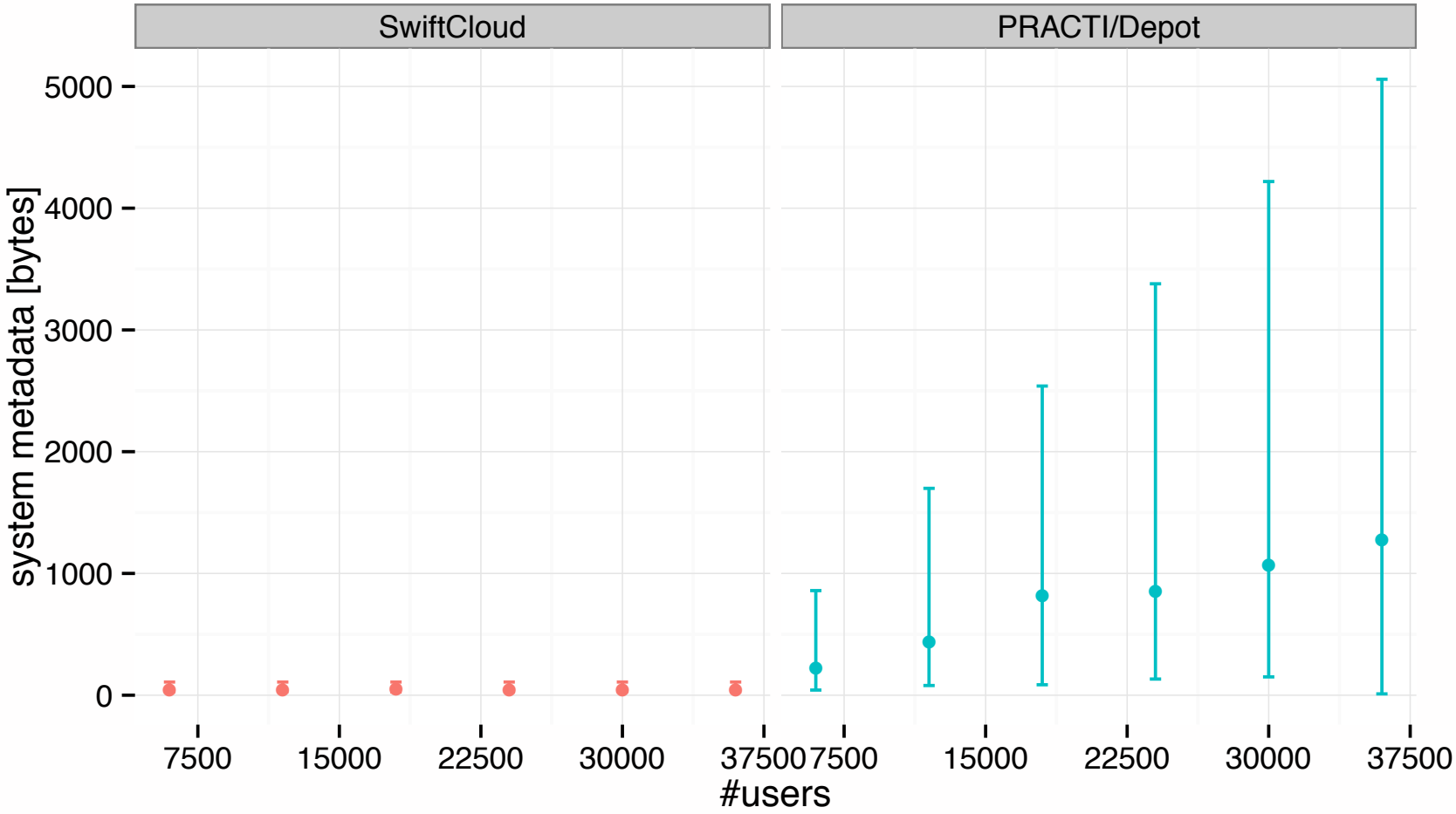


Staleness for fault tolerance





Metadata overhead



Summary

Fast response, high availability

- Shared store replicated at client
- Full consistency guarantees

Availability: Causal consistency is strongest possible

(+ transactions + red-blue)

Challenges & solutions:

- Large database: partial replicate data, metadata
- Small, bounded metadata: DC-based, pruning
- Fail-over
  - At-most-once: Client ID + merge clocks
  - Avoid gaps: read in the past (*K*-stable)
- Convergence: CRDTs