ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

HY-559
Infrastructure Technologies for Large-Scale Service-Oriented Systems

Kostas Magoutis
magoutis@csd.uoc.gr
http://www.csd.uoc.gr/~hy559

Geo-replicated storage

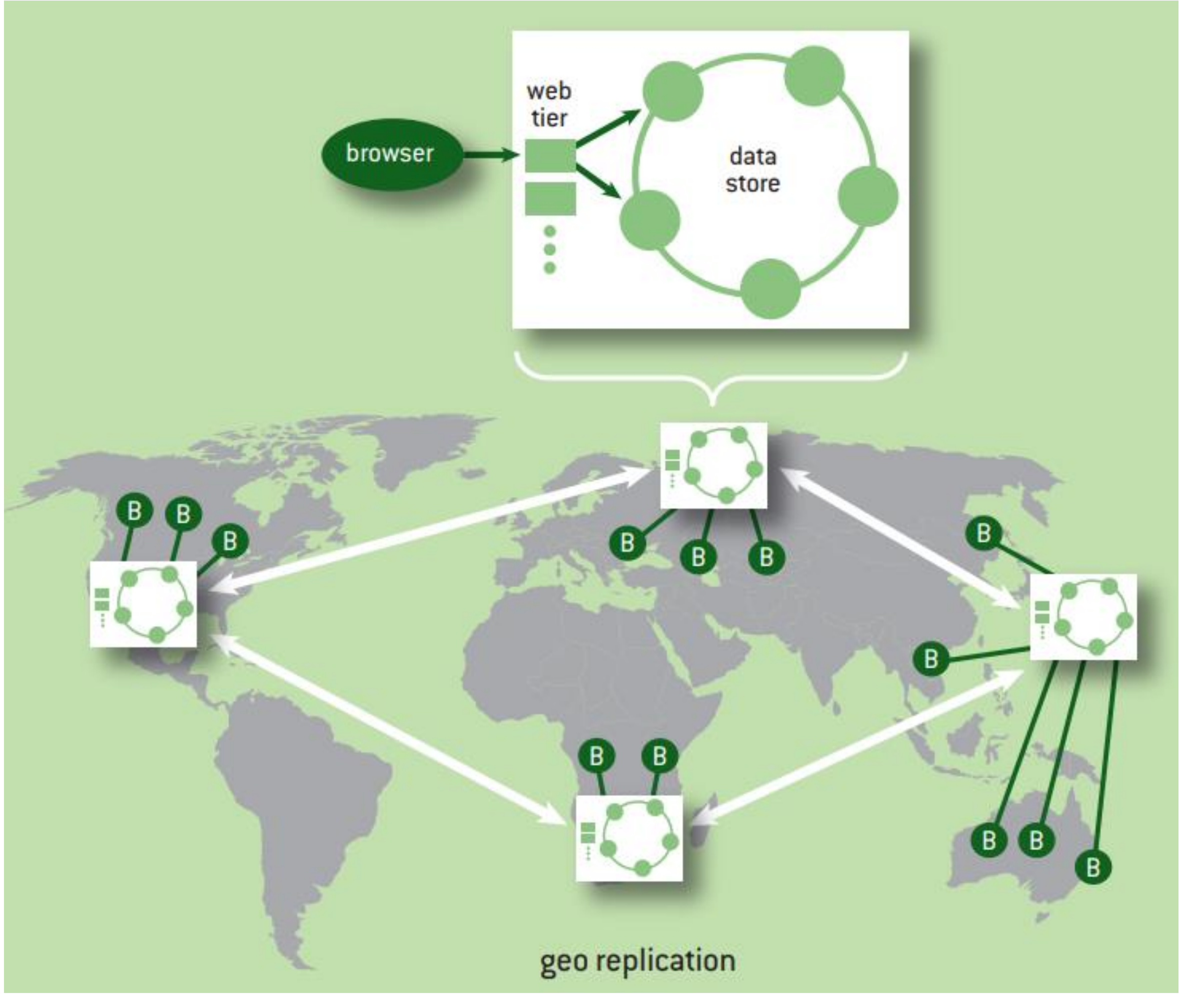


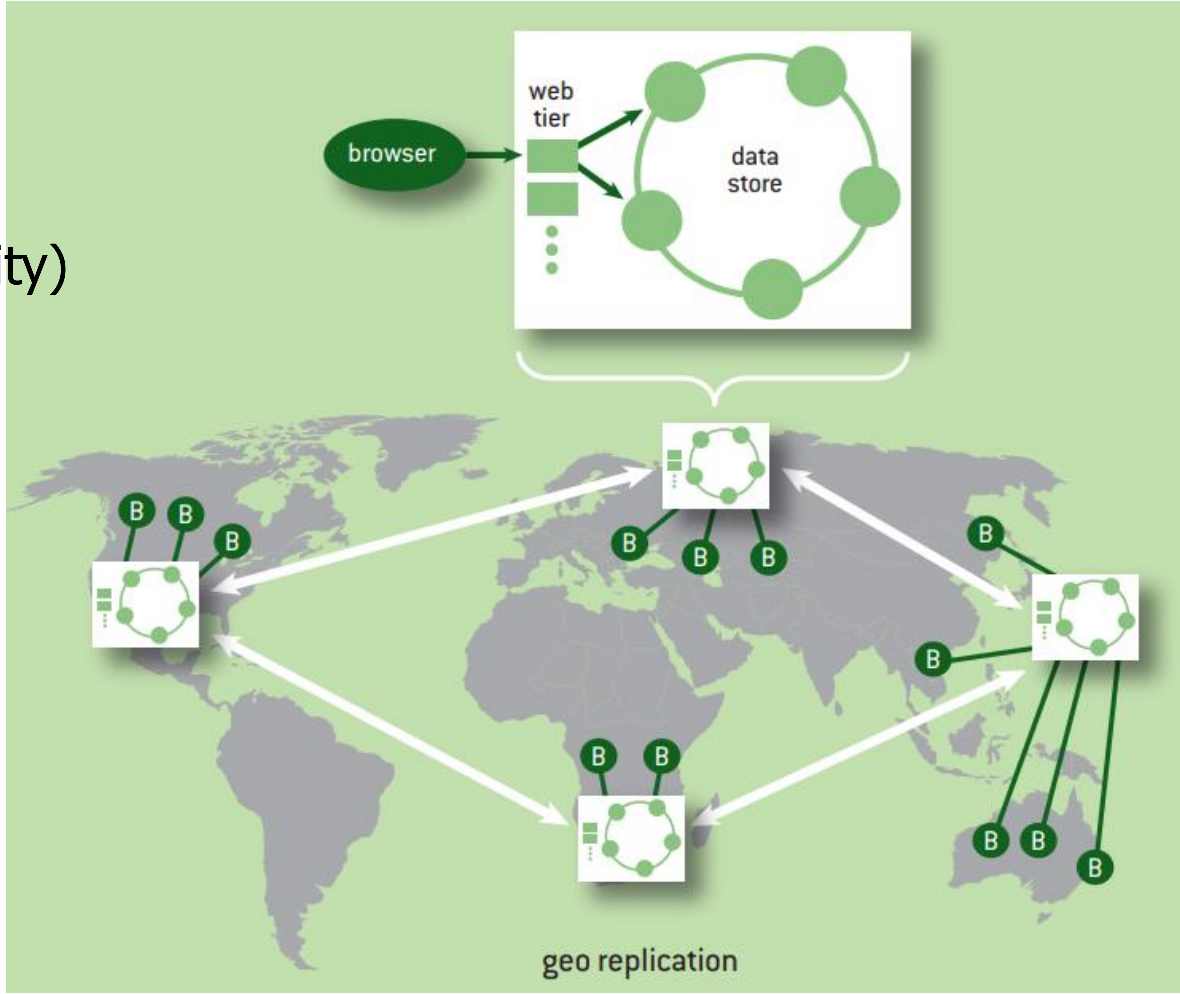
Image courtesy of: L. Wyatt et al. Don't Settle for Eventual Consistency, CACM'14

Geo-replicated storage

CAP theorem, cannot simultaneously achieve
- Consistency
- Availability
- Partition tolerance

Choose any two
(e.g., CA/CP -> Linearizability)

Real-world services opt for
- Availability
- Partition tolerance

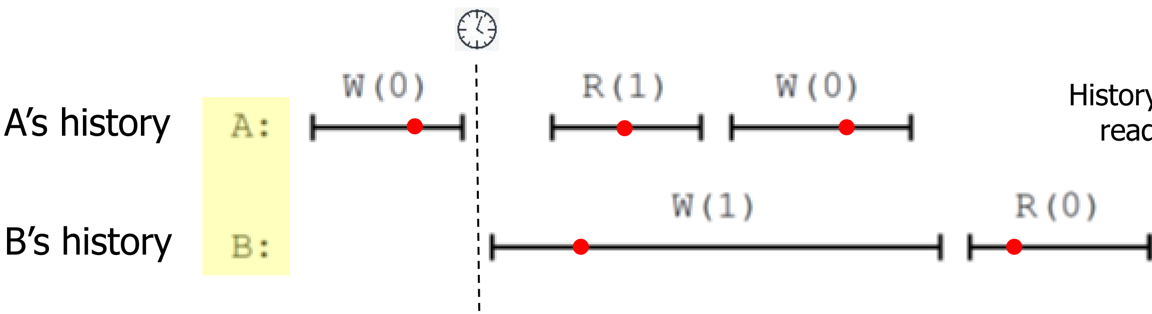


Desirable properties in geo-replicated services

- ALPS
  - Availability
  - Low latency
  - Partition tolerance
  - Scalability
- Linearizability (strong consistency) is not partition-tolerant
- One way to achieve ALPS: Eventual consistency
  - Writes to one data center (DC) eventually appear at other DCs
  - If all DCs receive the same set of writes, they will have the same values for all data
- This can lead to problems

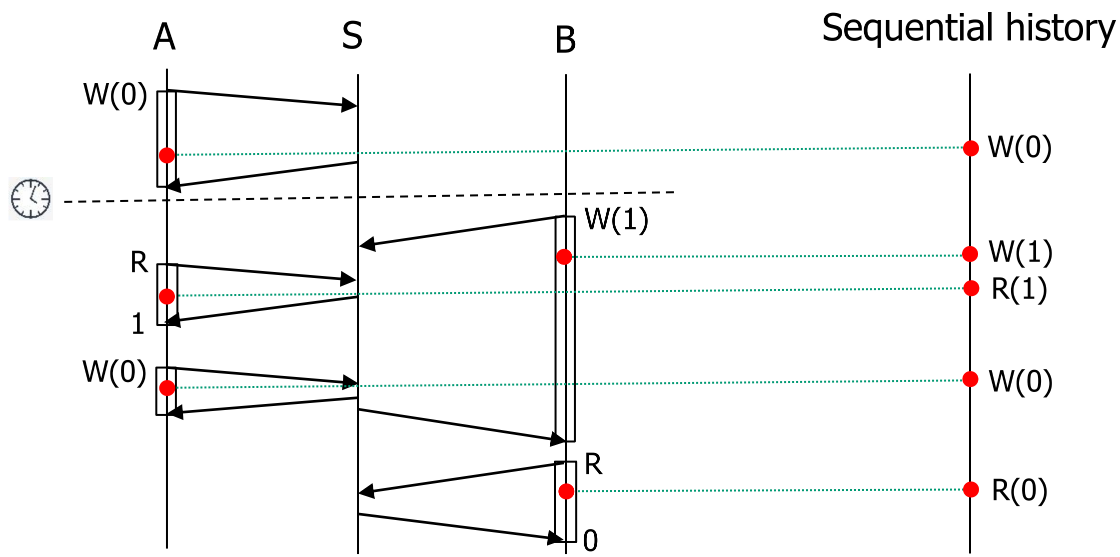
Linearizability

A collection of operations is linearizable if each operation appears to occur instantaneously and exactly once at some point-in-time between its invocation and its completion



History of concurrent clients performing reads and writes on a single object

Another view:



Satisfies sequential specification

Sources: C. Lee et al, Implementing Linearizability at Large Scale and Low Latency, SOSP'15,
M. Herlihy, J. Wing, Linearizability: A Correctness Condition for Concurrent Objects. ACM Trans. Program. Lang. Syst. 12 (July 1990), 463–492.

An example with replicated data

Data type: 4-location byte-valued read/write snapshot register

A multi-location read-write memory has

- a set of locations (or addresses)
- operations such as
  - read(*a*)
  - write(*a*, *w*)
  - snapshot()
- snapshot() returns a set of values, one for each location

location	value
1	0
2	0
3	0
4	0

Linearizable execution

Note:
- Writes indeed applied in the same order on all replicas in this example
- Mechanism for achieving order between writes is not shown here

Implementation rules:
- each read or snapshot is done on one replica
- each write is done on both replicas
- different writes are done in the same order at the replicas
- <u>a write doesn't return to the client until acked</u>

a legal history

(write(1, 5), "OK")

(read(1), 5)

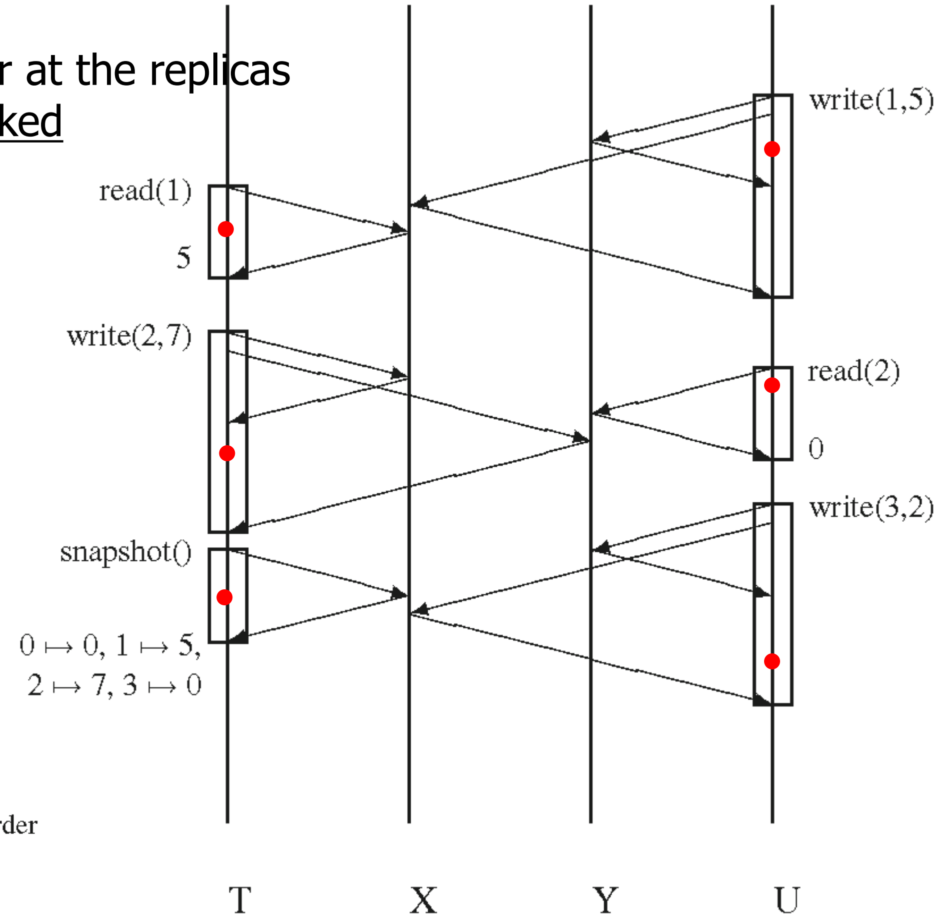
(read(2), 0)

(write(2, 7), "OK")

(snapshot(), $(0 \mapsto 0, 1 \mapsto 5, 2 \mapsto 7, 3 \mapsto 0)$)

(write(3, 2), "OK")

the order of operations as they occur in the sequence must not contradict any order information visible to an observer of the system execution.



From "Consistency Models for Replicated Data", A. D. Fekete, Krithi Mamamritham

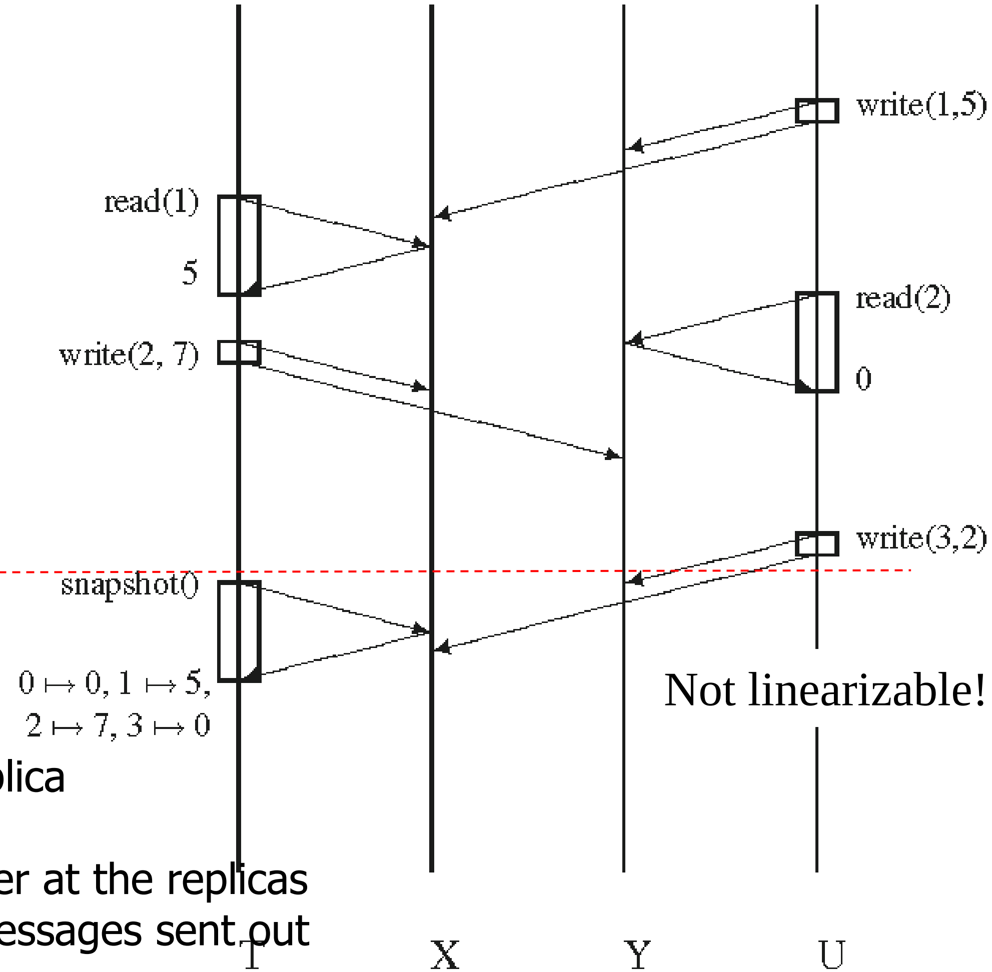
B. Charron-Bost, F. Pedone, and A. Schiper (Eds.): Replication, LNCS 5959, pp. 1–17, 2010.
© Springer-Verlag Berlin Heidelberg 2010

Sequential consistency

Two replicas at sites X and Y, clients located at T and U

a legal history

(write(1, 5), "OK")
(read(1), 5)
(read(2), 0)
(write(2, 7), "OK")
(snapshot(), $(0 \mapsto 0, 1 \mapsto 5, 2 \mapsto 7, 3 \mapsto 0)$)
(write(3, 2), "OK")



Implementation rules:

- each read or snapshot is done on one replica
- each write is done on both replicas
- different writes are done in the same order at the replicas
- a write returns to the client as soon as messages sent out

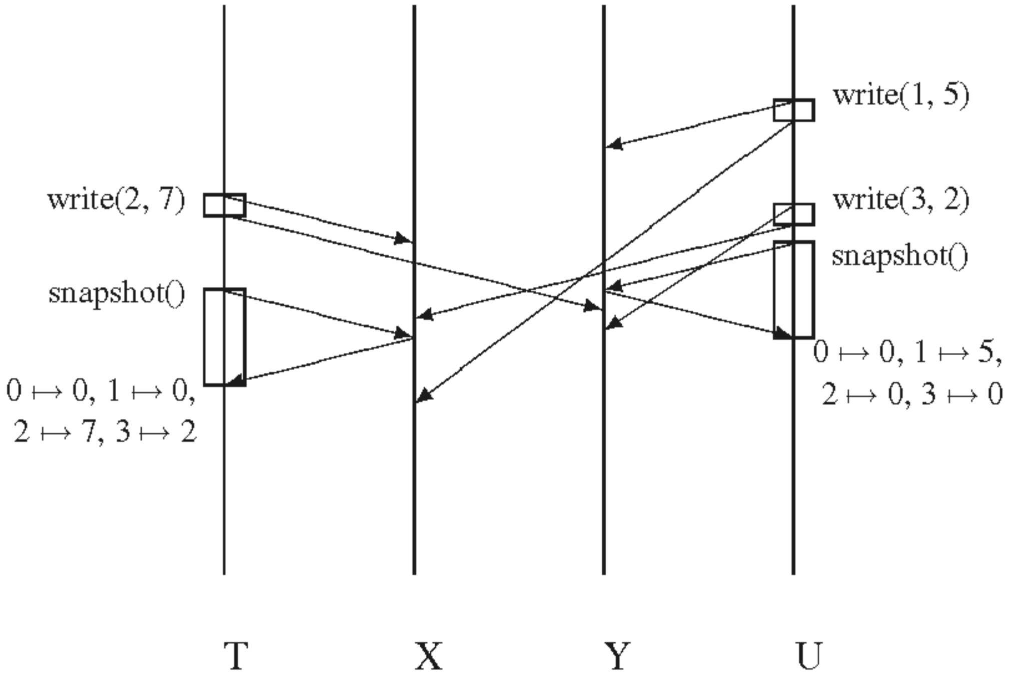
From "Consistency Models for Replicated Data", A. D. Fekete, Krithi Mamamritham

B. Charron-Bost, F. Pedone, and A. Schiper (Eds.): Replication, LNCS 5959, pp. 1–17, 2010.
© Springer-Verlag Berlin Heidelberg 2010

Weak consistency

Implementation rules:

- each read or snapshot is done on one replica
- each write is done on both replicas
- ~~different writes are done in the same order at the replicas~~
- a write returns to the client as soon as messages sent out

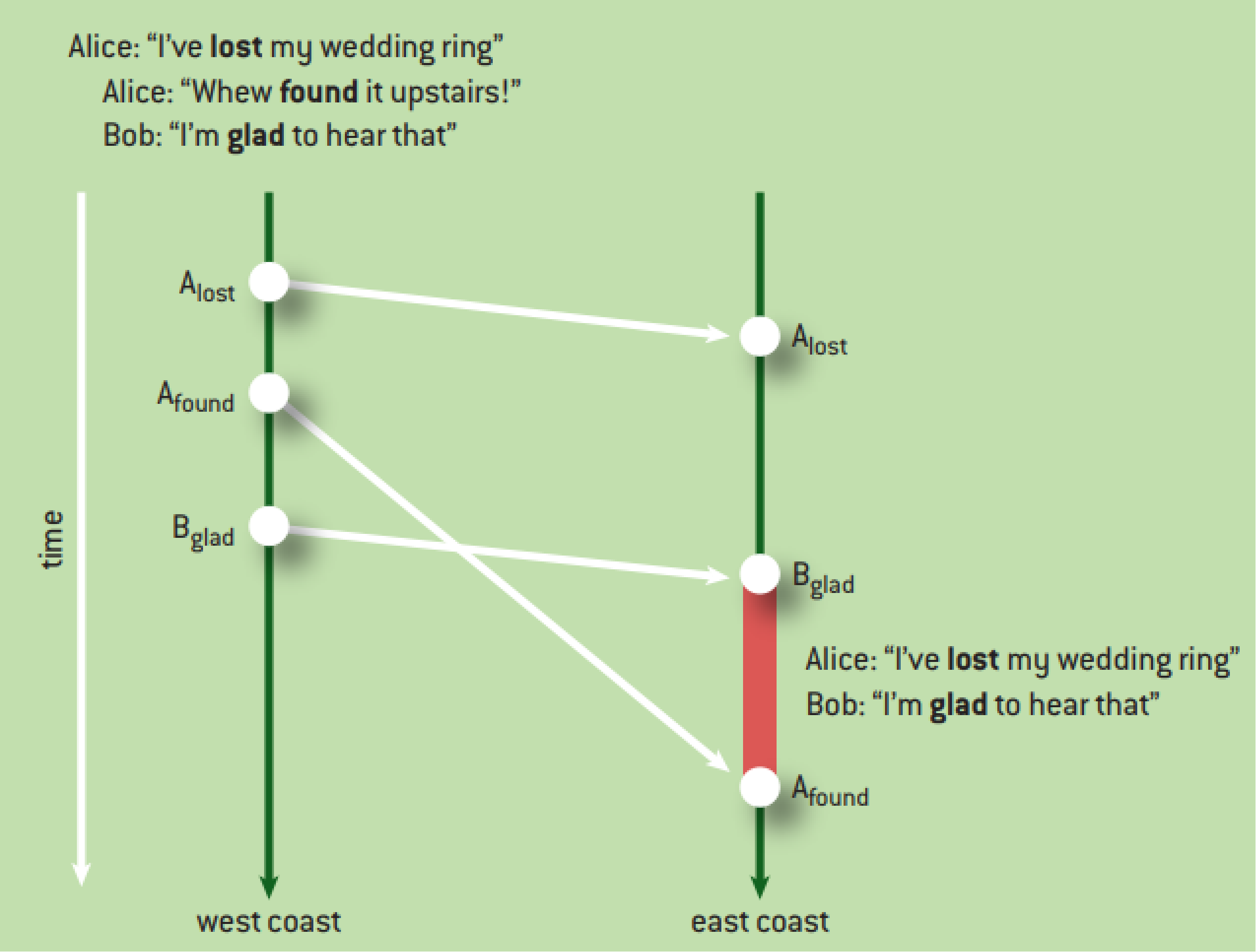


<u>Cannot find a legal history that would satisfy either linearizability or SC conditions</u>

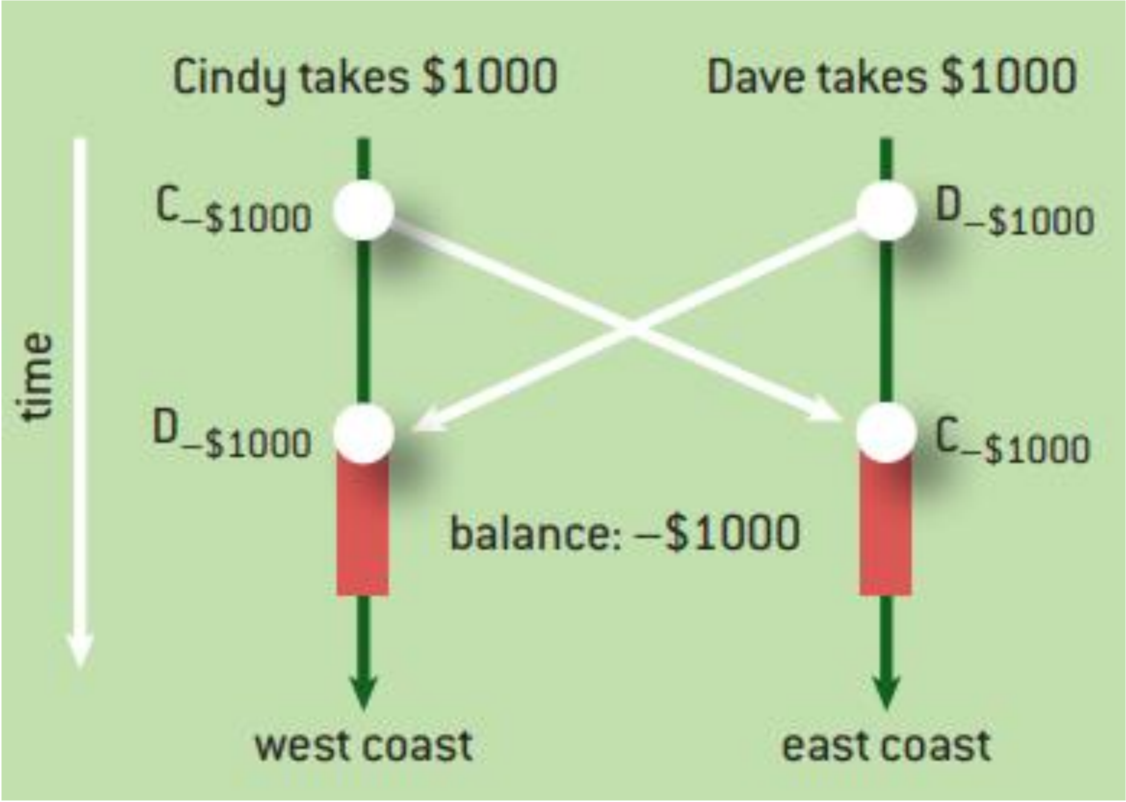
Desirable properties in geo-replicated services

- ALPS
  - Availability
  - Low latency
  - Partition tolerance
  - Scalability
- Linearizability (strong consistency) is not partition-tolerant
- One way to achieve ALPS: Eventual consistency
  - Writes to one data center (DC) eventually appear at other DCs
  - If all DCs receive the same set of writes, they will have the same values for all data
- This can lead to problems

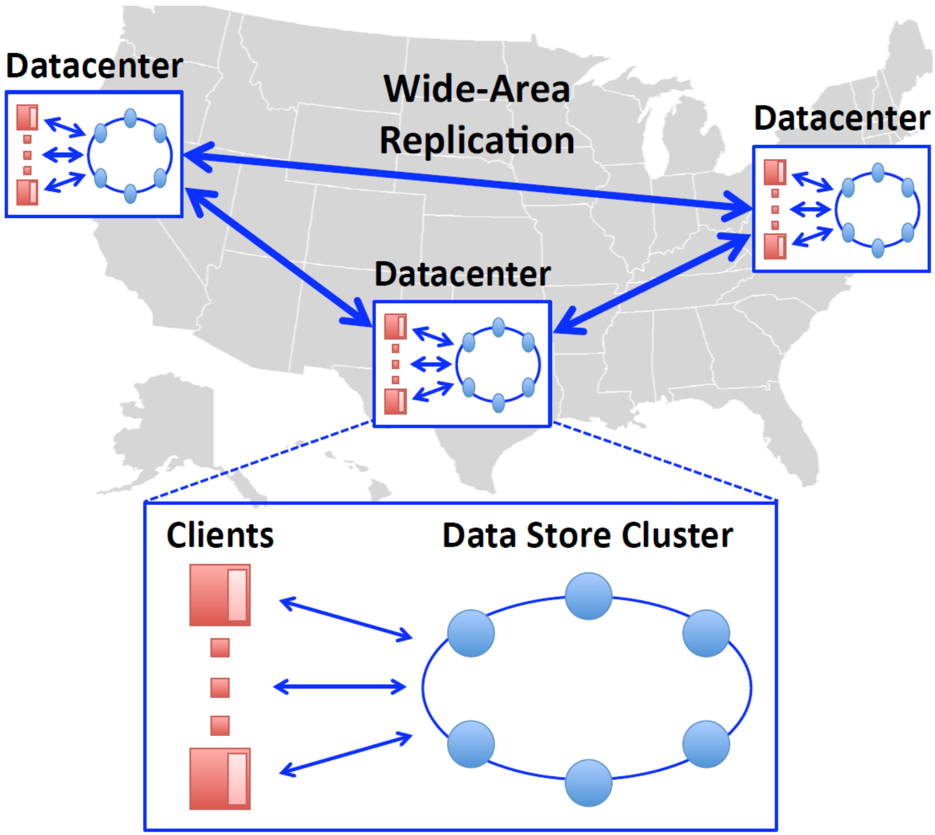
Problem 1: Comment reordering



Problem 2: Double money withdrawal



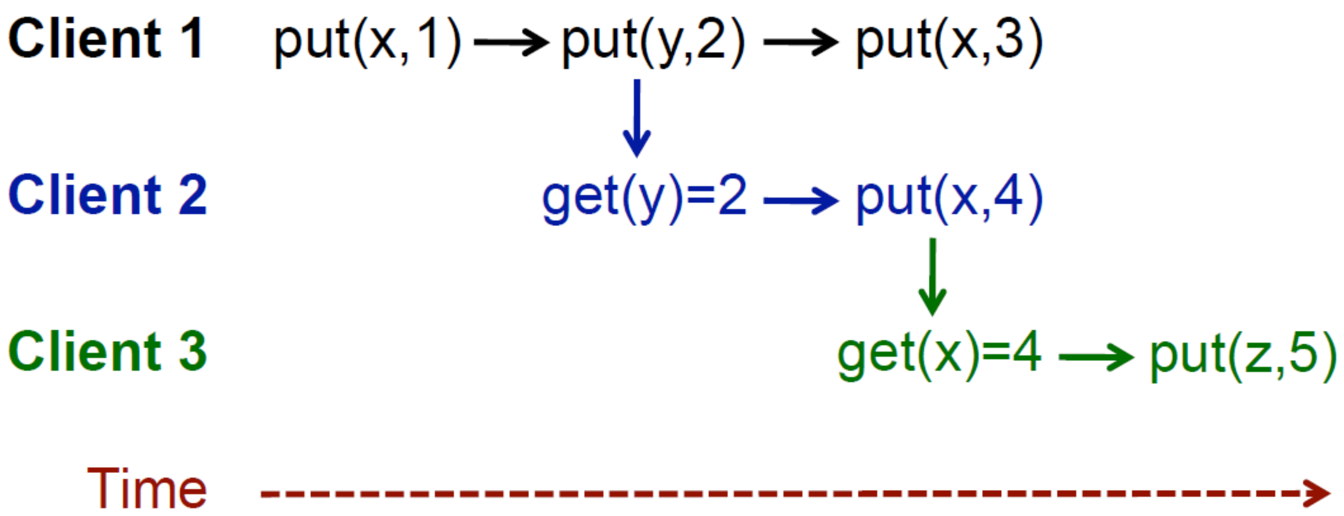
General architecture of modern web services



Images courtesy of: L. Wyatt et al. Don’t Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS, SOSP’11

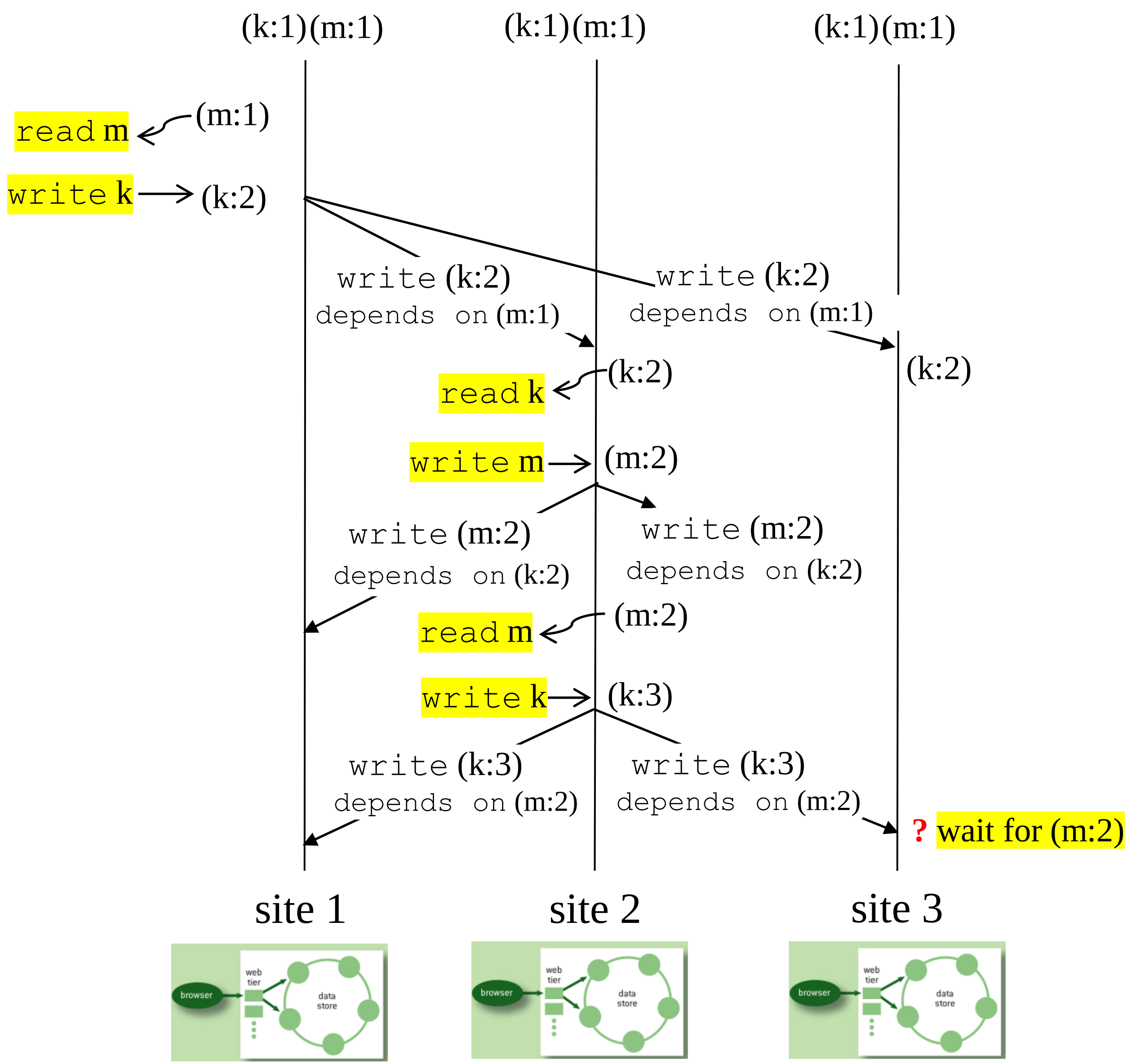
Example of causal relationships

1. **Execution Thread.** If a and b are two operations in a *single thread of execution*, then $a \rightsquigarrow b$ if operation a happens before operation b.
2. **Gets From.** If a is a `put` operation and b is a `get` operation that returns the value written by a, then $a \rightsquigarrow b$.
3. **Transitivity.** For operations a, b, and c, if $a \rightsquigarrow b$ and $b \rightsquigarrow c$, then $a \rightsquigarrow c$.

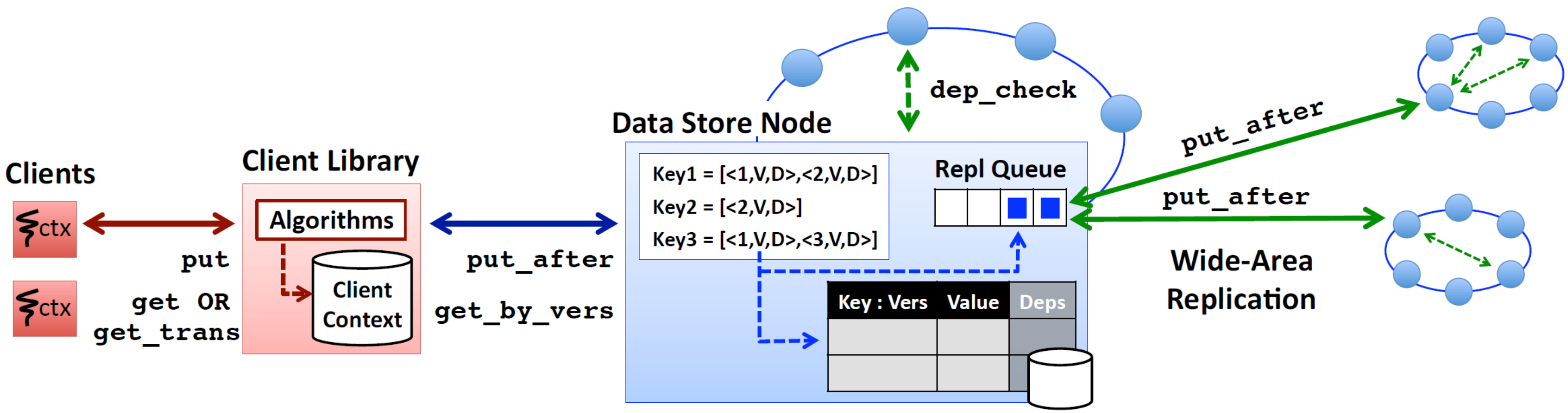


(m:1) means “key **m**, version **1**”

Example



COPS architecture



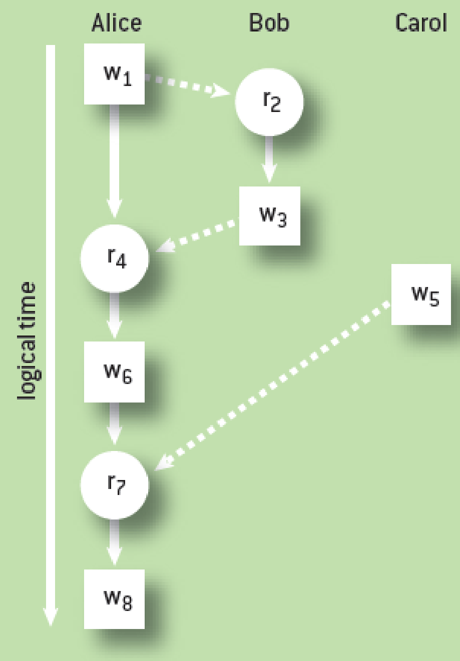

```
# Alice's Photo Upload
ctx_id = createContext()    // Alice logs in
put(Photo, "Portuguese Coast", ctx_id)
put(Album, "add &Photo", ctx_id)
deleteContext(ctx_id)       // Alice logs out

# Bob's Photo View
ctx_id = createContext()    // Bob logs in
"&Photo" ← get(Album, ctx_id)
"Portuguese Coast" ← get(Photo, ctx_id)
deleteContext(ctx_id)       // Bob logs out
```

Causality and dependency

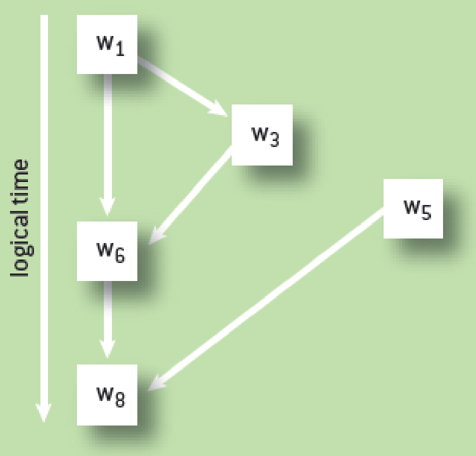
(a)

user	op ID	operation
Alice	w_1	write(Alice:town, NYC)
Bob	r_2	read(Alice:town)
Bob	w_3	write(Bob:town, LA)
Alice	r_4	read(Bob:town)
Carol	w_5	write(Carol:likes, ACM, 8/31/12)
Alice	w_6	write(Alice:likes, ACM, 9/1/12)
Alice	r_7	read(Carol:likes, ACM)
Alice	w_8	write(Alice:friends, Carol, 9/2/12)

(b)



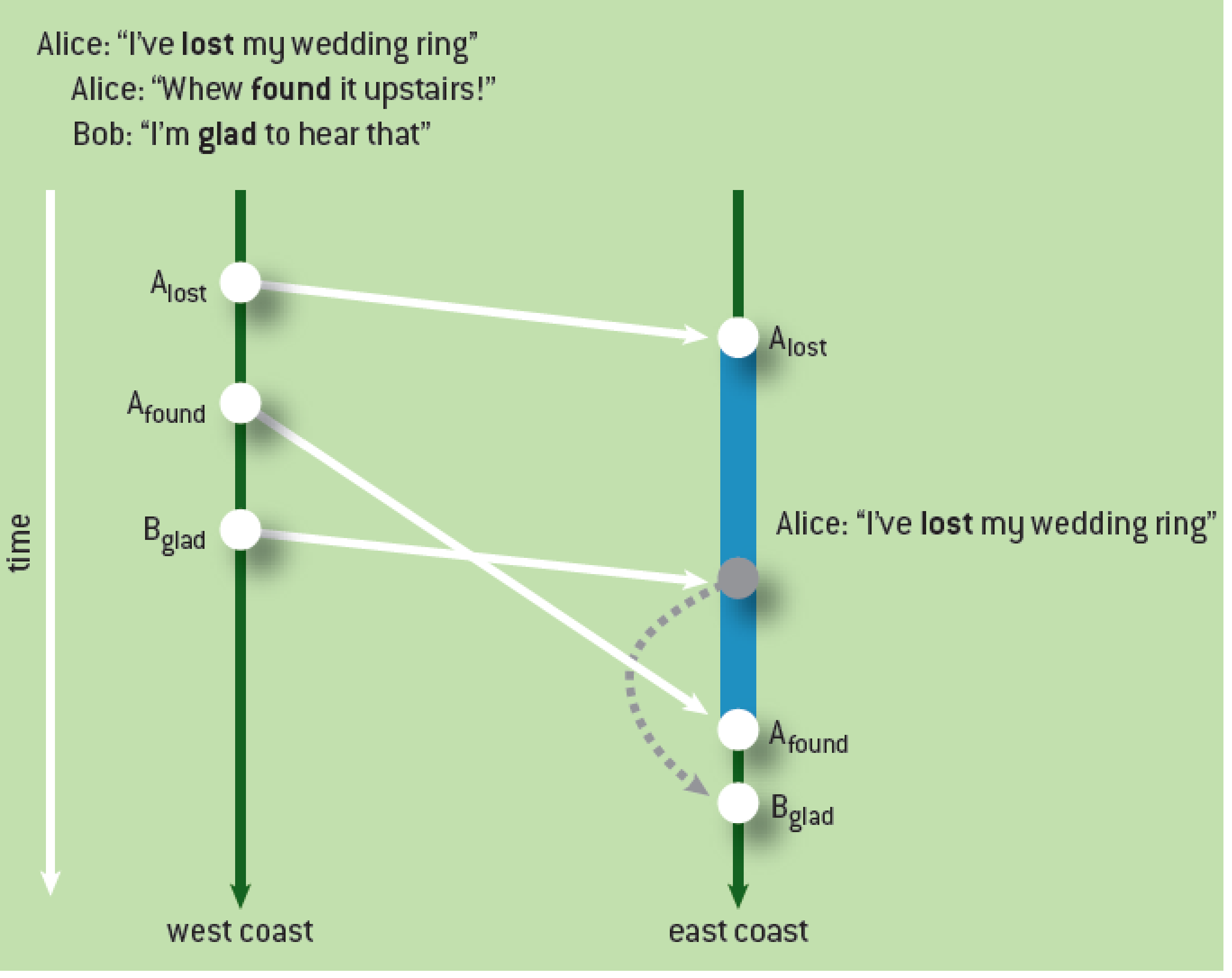
(c)



(d)

op ID	Dependencies
w_1	–
w_3	w_1
w_5	–
w_6	$w_3\ w_1$
w_8	$w_6\ w_5\ w_3\ w_1$

Problem 1 fixed



How to handle concurrent writes?

- Causally-unrelated writes require additional support
  - Hard to maintain global invariants (e.g., balance > 0)

- These are rare, and can be handled with
  - Later reconciliation
  - Last-writer-wins