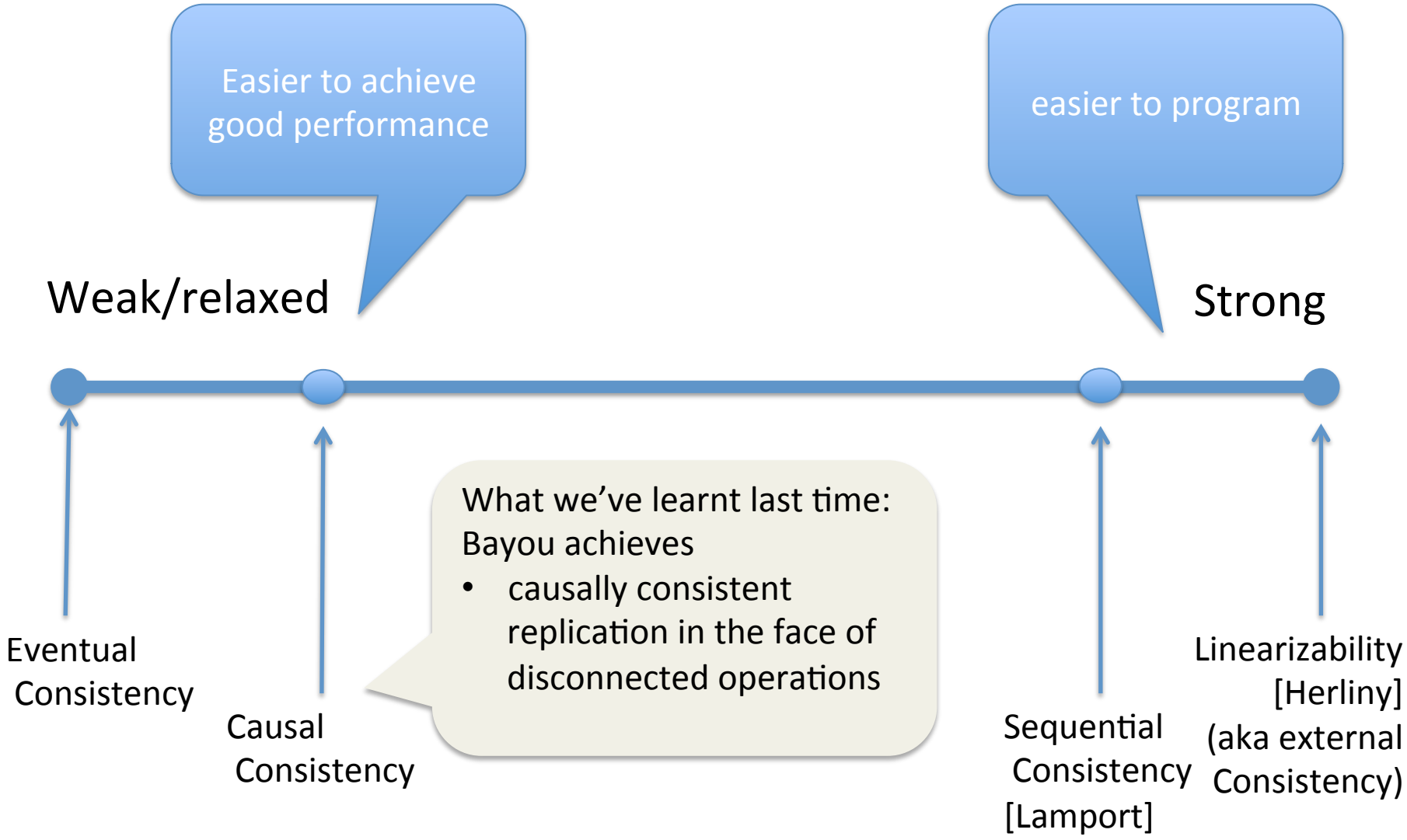
Relaxed consistency, continued

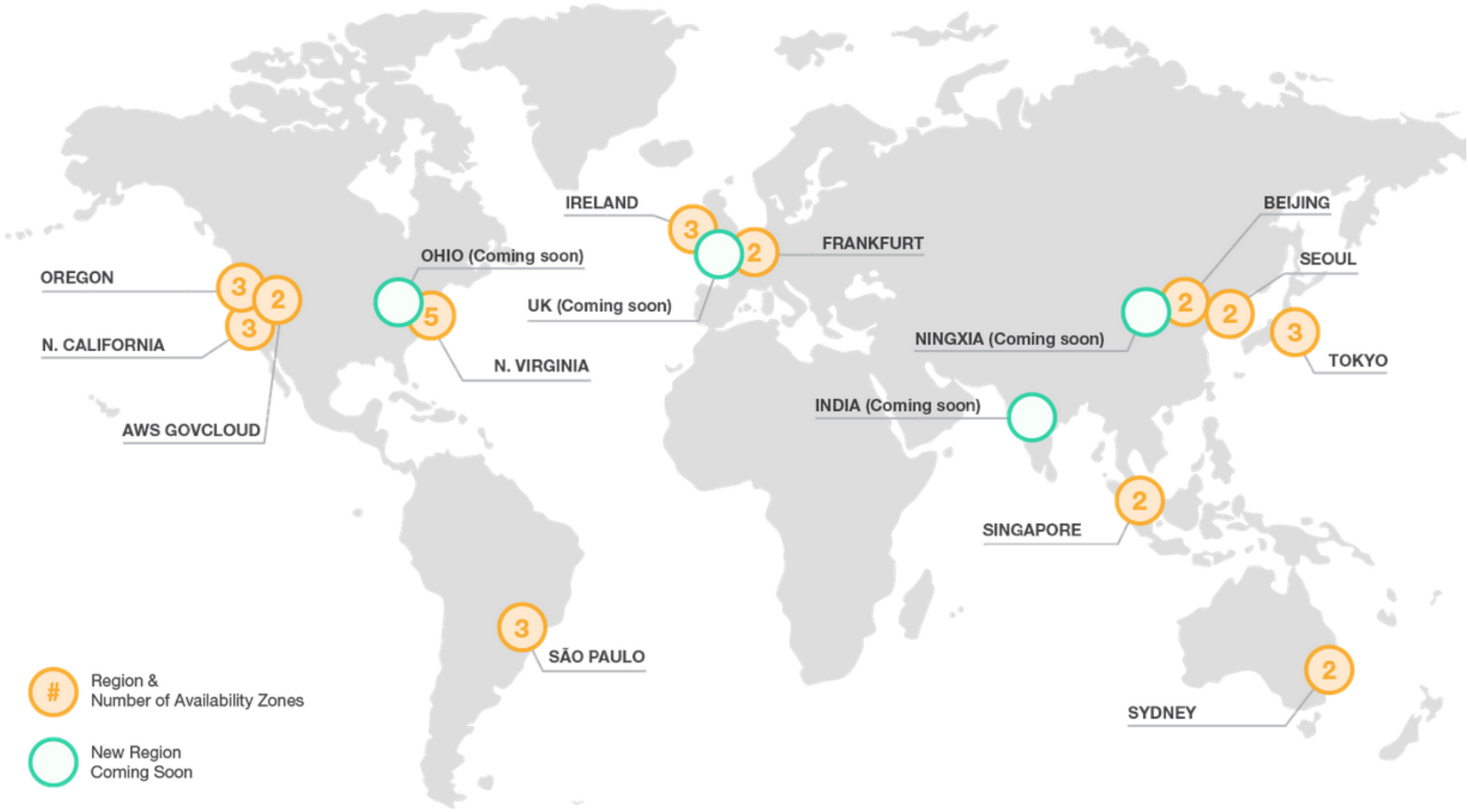
Jinyang Li

some slides are from Haonan Lu's talk on existential consistency

Spectrum of Consistency Models



Today's topic: relaxed consistency for geo-replication



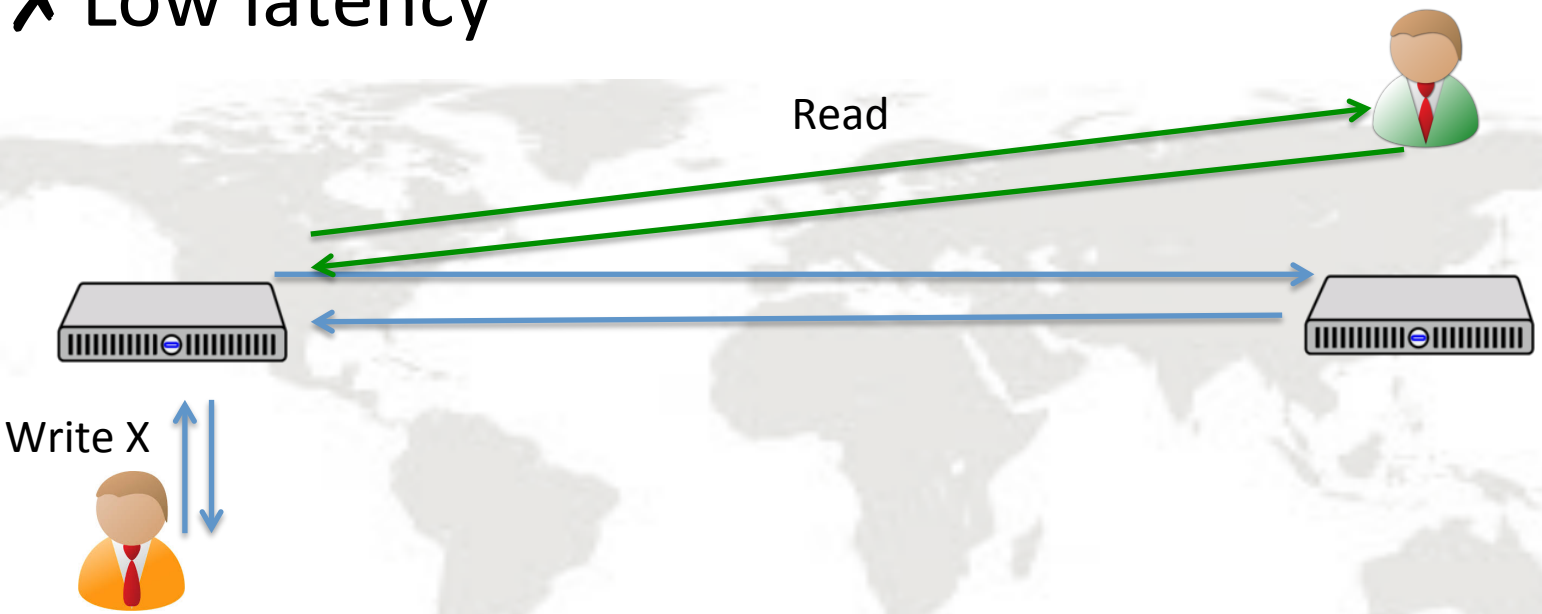
Why geo-replication?

- Why replicate across geographic regions?
- Low latency (esp. for reads)
  - users read from the closest data center
- Disaster tolerance
  - Operator mistakes:
    - AWS us-east-1 region outage in Feb 2017
  - Natural disaster: Hurricanes, earthquakes

Why not supporting linearizability

✔Scalability

✗ Low latency



Why not causal consistency?

- Scalability?
  - Bayou is not scalable (requires full replication at every node)
  - Recent protocols allow sharded implementation
    - COPS [SOSP'11] Occult [NSDI'17]
    - Track lots of dependencies or large vector timestamps
- Low latency?

  ✔Writes

  ☑Reads (not always in COPS and Occult).
    - If write X→ write Y, client reads the new value of Y, then it needs to wait for X's write as well.

Today's reading #1: PNUTS

PNUTS: Yahoo!'s Hosted Data Serving Platform

Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein,
Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver and Ramana Yerneni
Yahoo! Research

BSTRACT

describe PNUTS, a massively parallel and geographi-
y distributed database system for Yahoo!'s web applica-
ıs. PNUTS provides data storage organized as hashed
ordered tables, low latency for large numbers of con-
rent requests including updates and queries, and novel
record consistency guarantees. It is a hosted, centrally

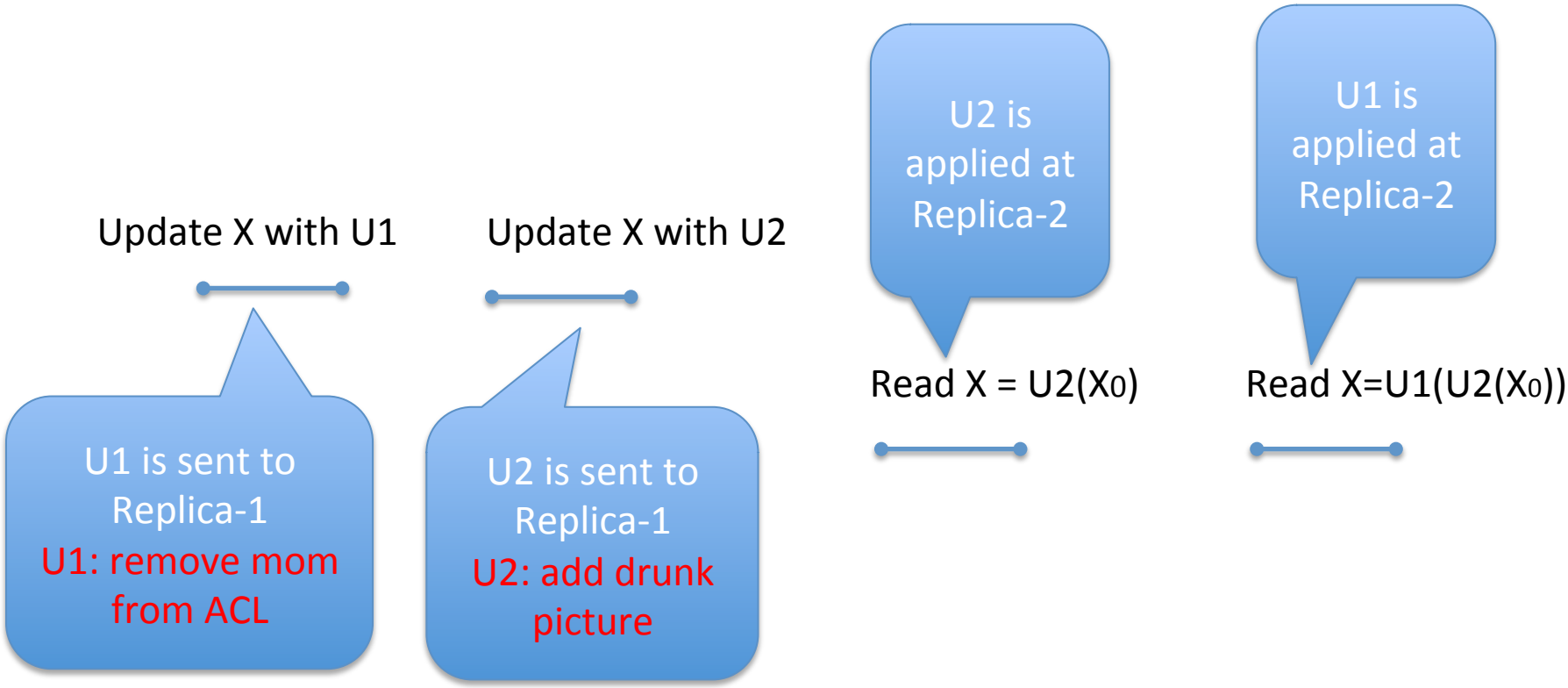
Yahoo!'s internal SLAs for page load time, placing stri
response time requirements on the data management
form. Given that web users are scattered across the g
it is critical to have data replicas on multiple continen
low-latency access. Consider social network applicatic
alumni of a university in India may reside in North Am
and Europe as well as Asia, and a particular user's data

VLDB 2008

State-of-art industry solutions before PNUTS

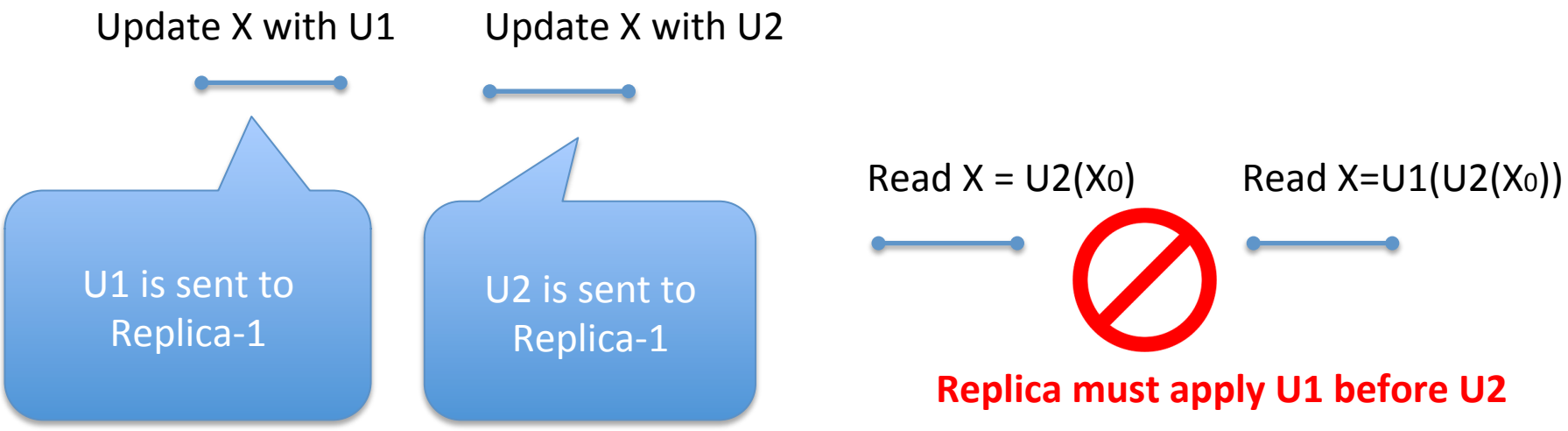
- Amazon's Dynamo key-value store
  - Eventual (but not causal) consistency
  - Ensures replica convergence
- Good performance
  - Writes is stored at any replica which returns immediately
  - Reads can also be processed by any replica
- Published at SOSP 2007

Why not eventual consistency?

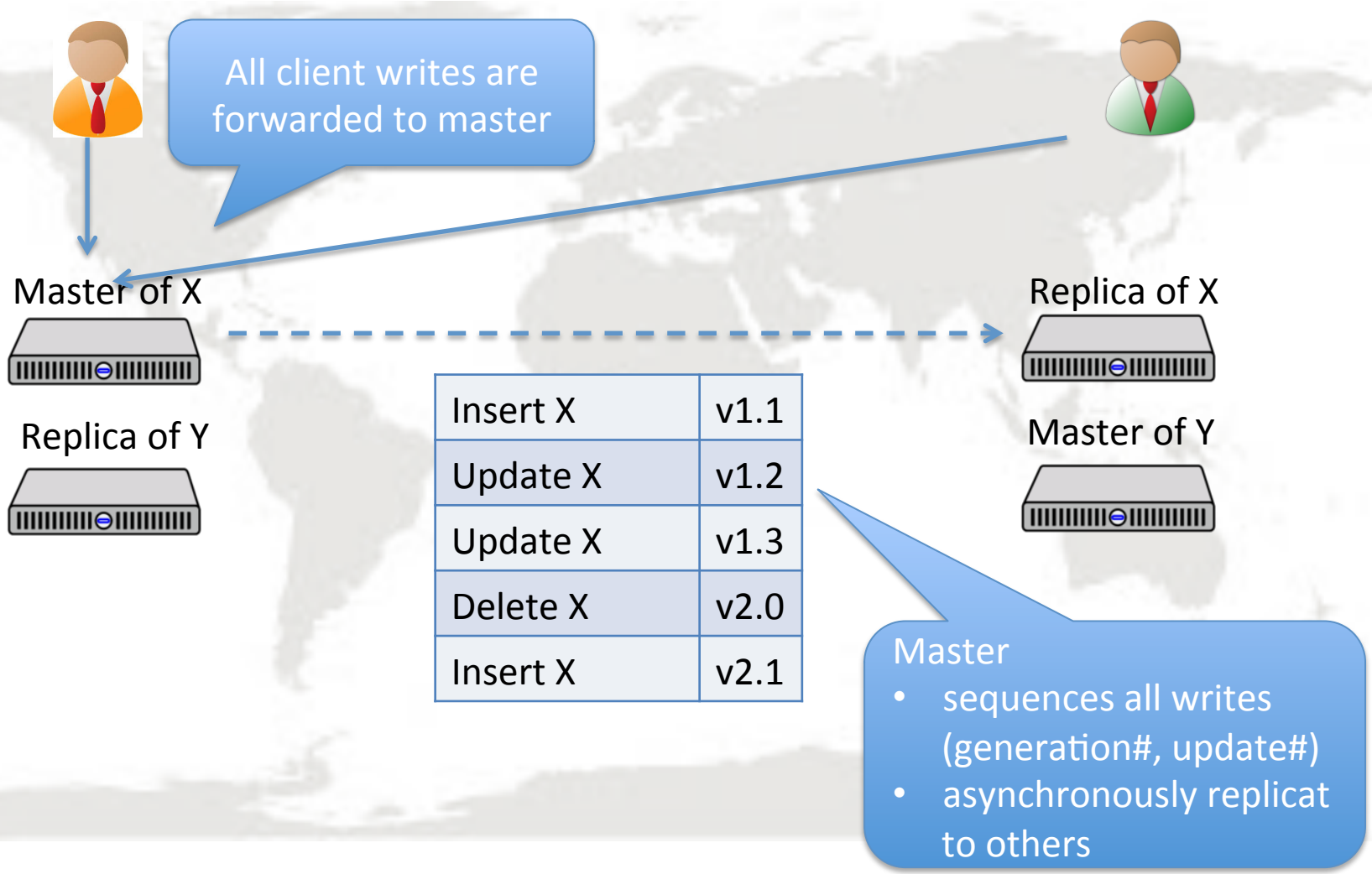


PNUTS' consistency model

- Per-record timeline consistency
- All updates to a given record/object are applied in the same order at all replicas



How PNUTS' consistency is implemented



PNUTS' APIs

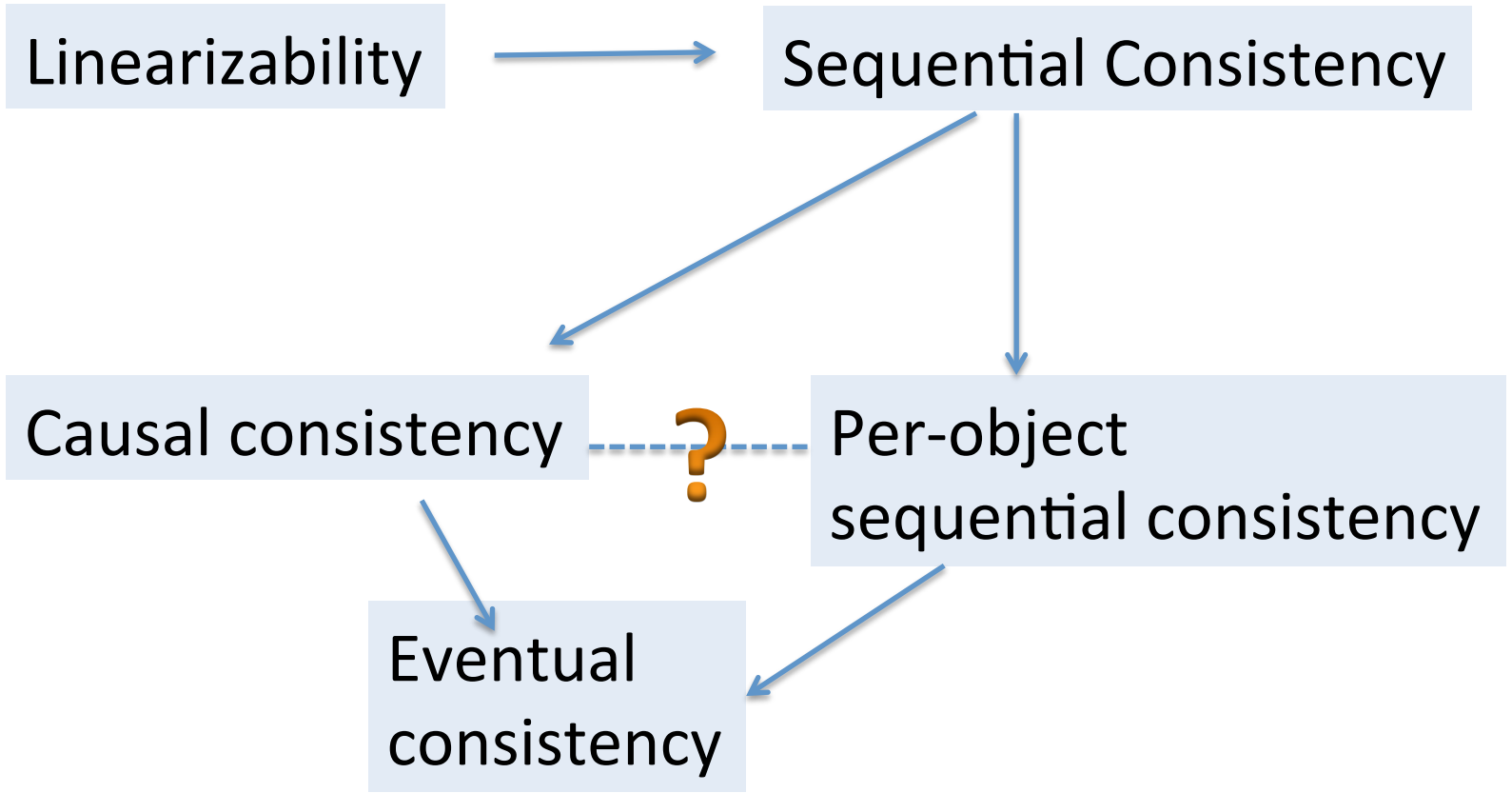
- Read-any
  - read from any replica (possibly a stale value)
- Read-critical(required_version)
  - must read a version that's >= required version
- Read-latest
  - read the most recent copy of data
- Write
- Test-and-set-write(required_version)
  - Perform write at master iff present version of record is same as requred_version

Per-record timeline consistency vs. sequential consistency

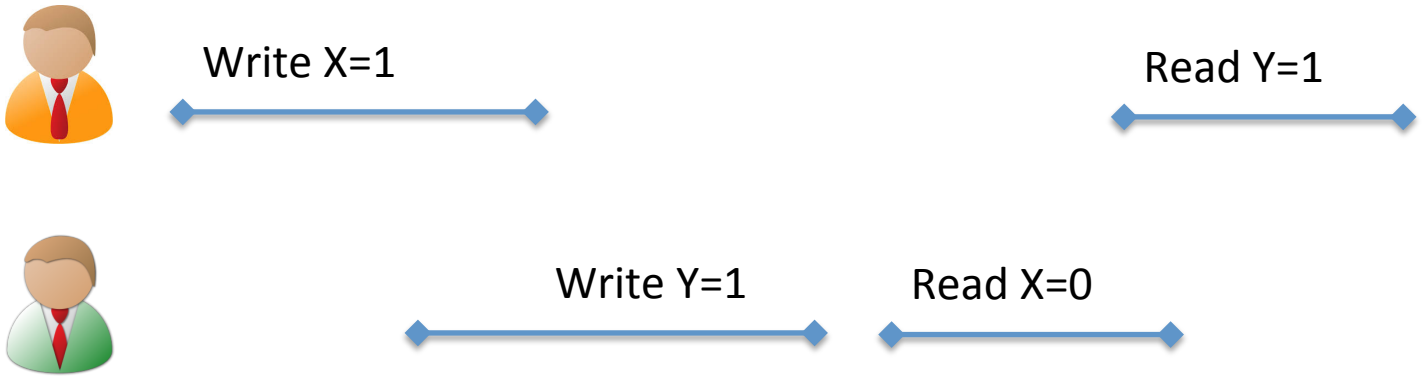
- Per-object Sequential consistency
  - all operations to a single object can be serialized w.r.t. each other
  - the issue order of operations to a single object for a given client is preserved.
- Per-object sequential consistency can be achieved in PNUTS by
  - Client tracks latest-version written/read. Use read-critical API for reads

The hierarchy of consistency

- Relative order of consistency models

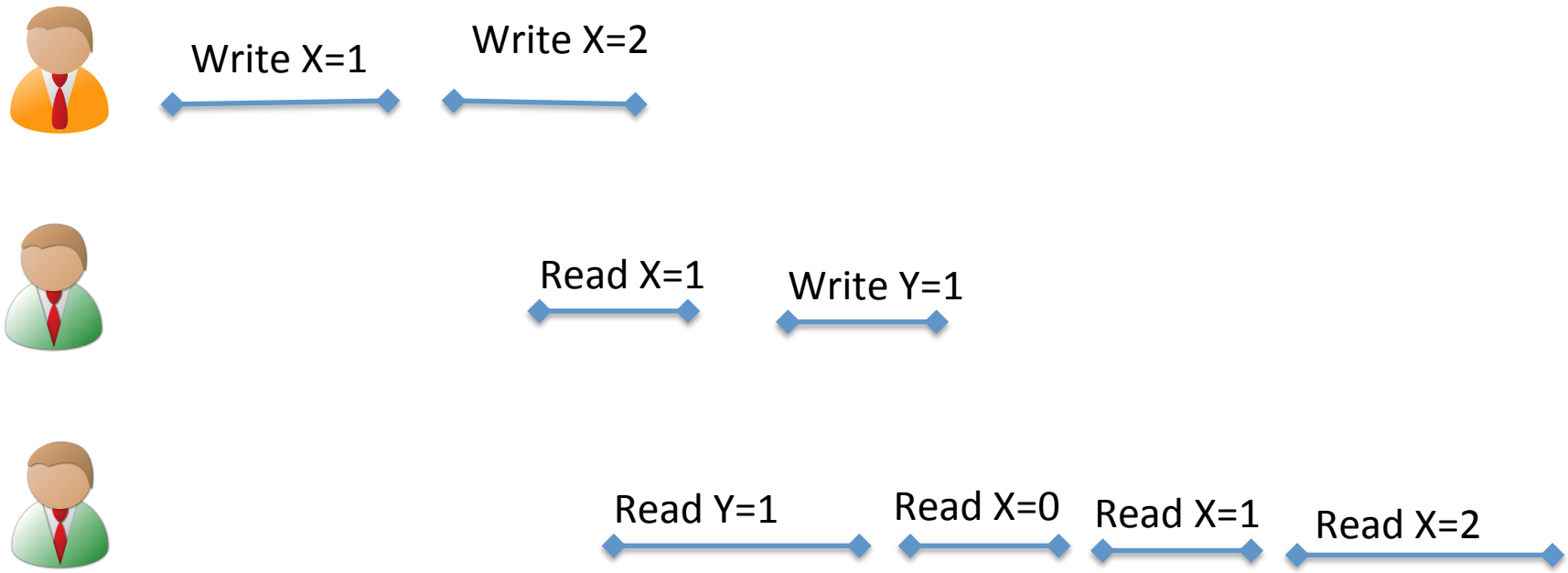


Consistency example #1



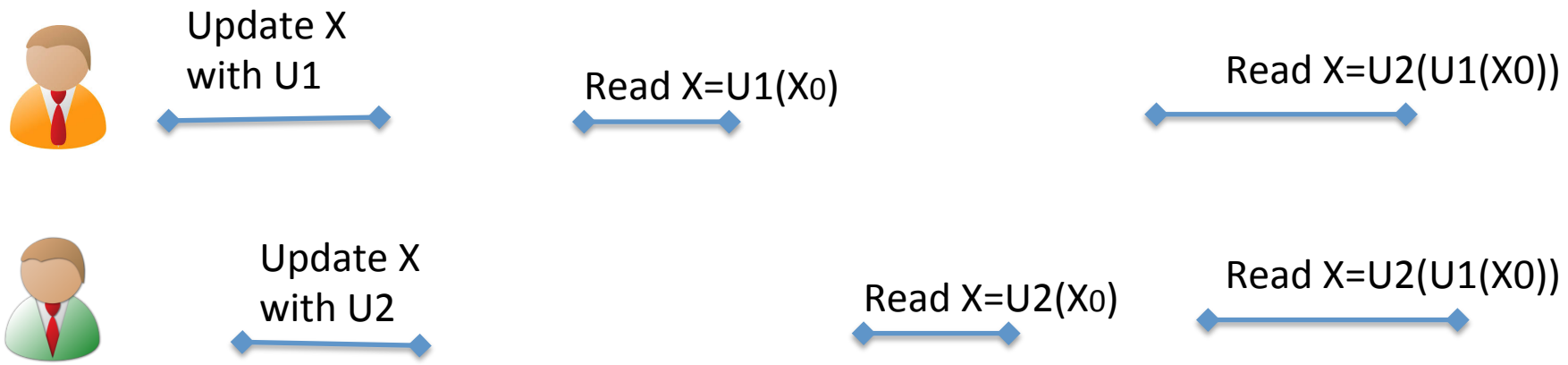
- ✔ Sequential consistency
- ✗ Linearizability

Consistency example #2



- ✔Per-object sequential consistency
- ✗ Sequential consistency
- ✗ Causal consistency

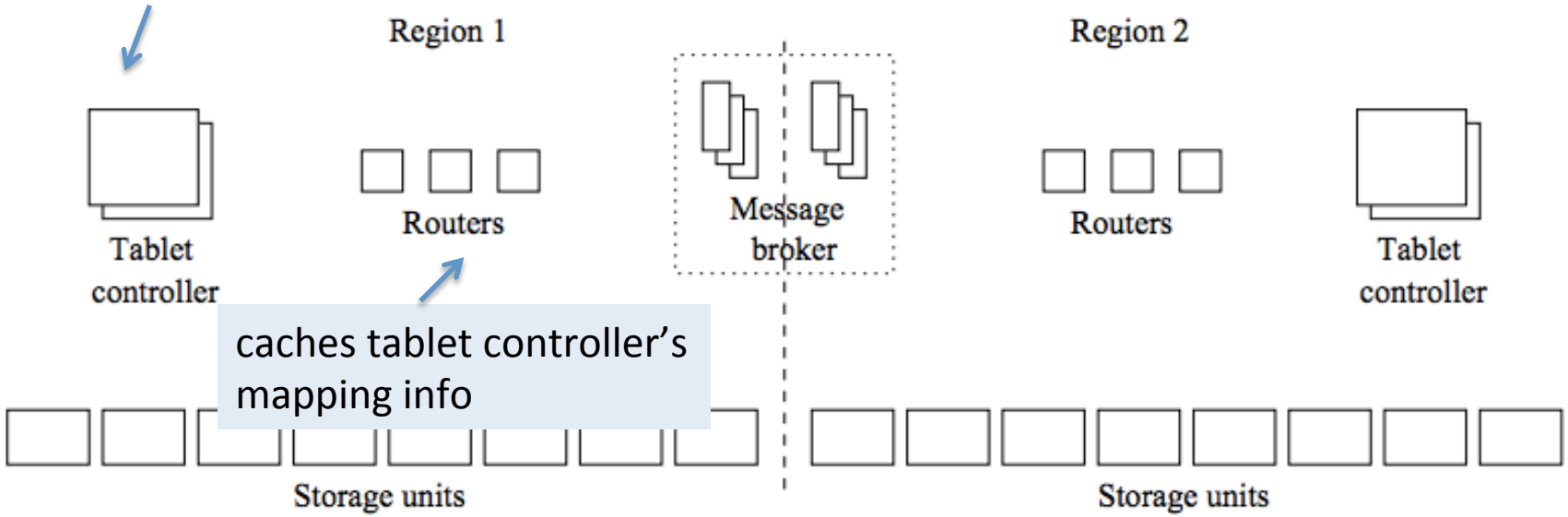
Consistency example #3



- ✔Causal consistency
- ✗ Per-object sequential consistency

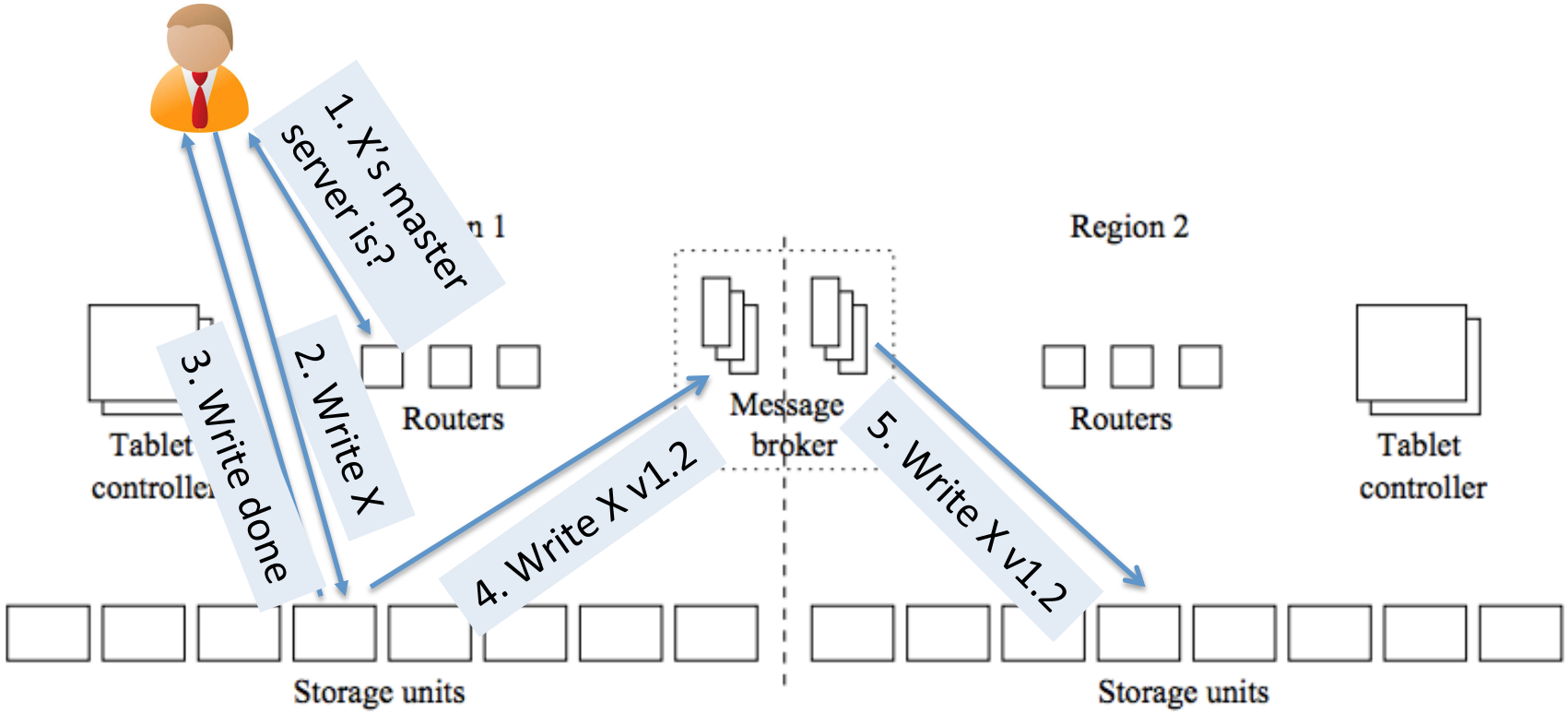
PNUTS' system architecture

Determines 1) record to tablet mapping
2) tablet to server mapping



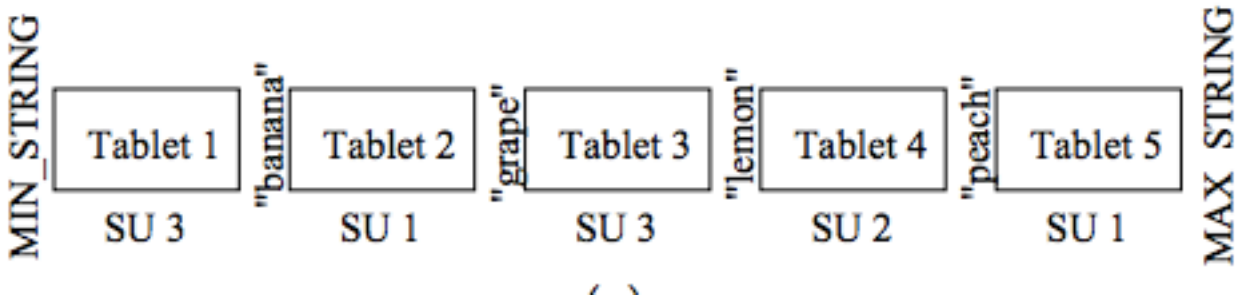
Data is sharded across many servers

PNUTS' system architecture



PNUTS' sharding

- Un-ordered table
  - Compute n-bit hash of key: hash(key) → [0...2^n)
  - tablet# = hash % total_number_of_tablets
    - Not used by PNUTS
- Ordered table
  - Controller maintains key intervals of all tablets
  - Controller splits a tablet if it becomes too large



PNUTS' failure tolerance

- Master replica failure
  - Re-generate a new replica somewhere else
  - writes-in-transit are not lost because they are stored in YMB (internally replicated across multiple machines)
- Tablet controller failure
  - it is consistently replicated to a backup controller
- Region failure
  - Writes-in-transit are not available (if failure is temporary) or lost (if failure is permanent)

Reading #2:Measuring consistency violation

Existential Consistency: Measuring and Understanding Consistency at Facebook

Haonan Lu*†, Kaushik Veeraraghavan†, Philippe Ajoux†, Jim Hunt†,
Yee Jiun Song†, Wendy Tobagus†, Sanjeev Kumar†, Wyatt Lloyd*†

*University of Southern California, †Facebook, Inc.

Abstract

Replicated storage for large Web services faces a trade-off between stronger forms of consistency and higher performance properties. Stronger consistency prevents anomalies, i.e., unexpected behavior visible to users, and reduces programming complexity. There is much recent work on improving the performance properties of systems with stronger consistency, yet the flip-side of this trade-off remains elu-

1. Introduction

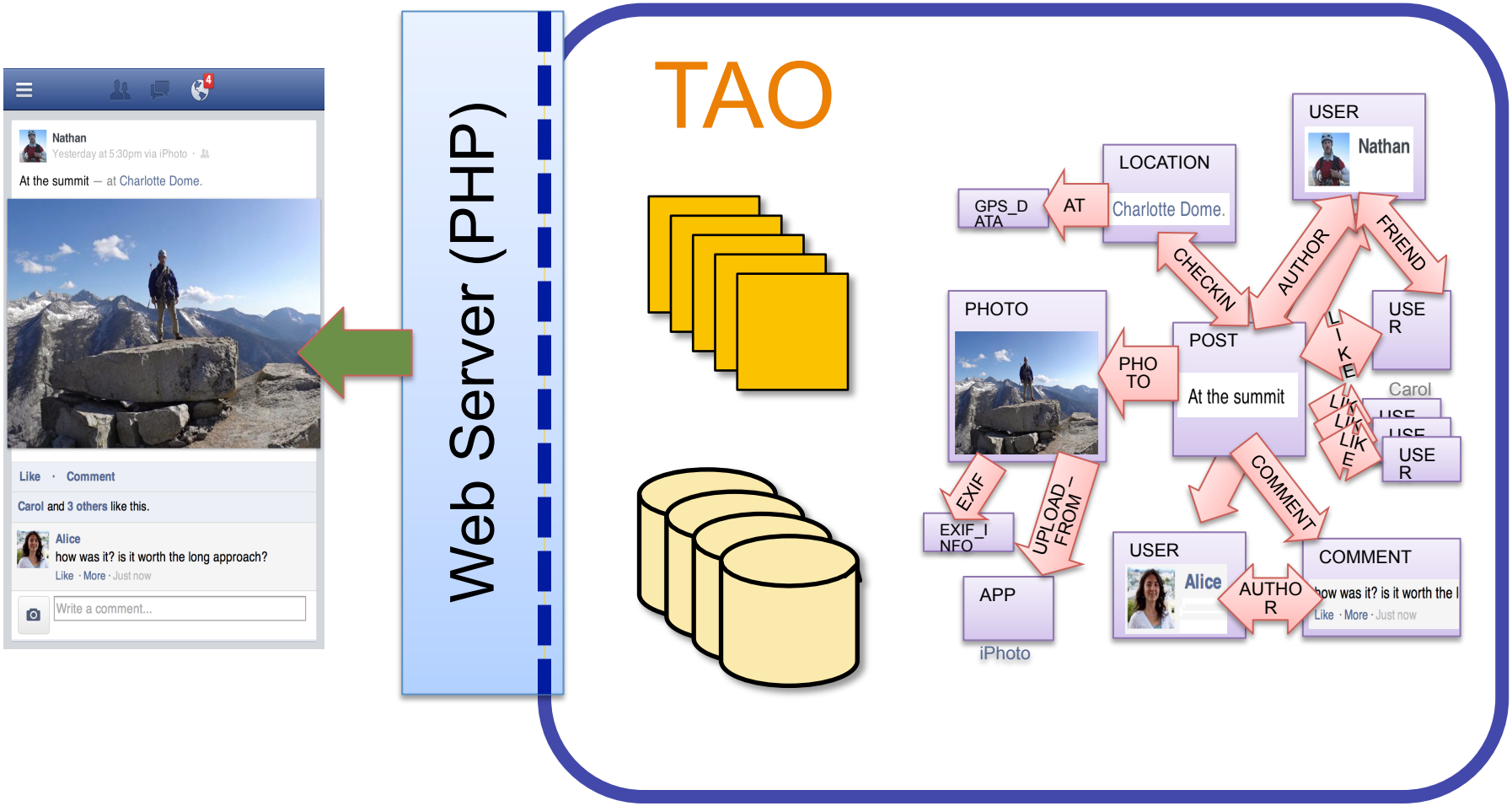
Replicated storage is an important component of large Web services and the consistency model it provides determines the guarantees for operations upon it. The guarantees range from eventual consistency, which ensures replicas eventually agree on the value of data items after receiving the same set of updates to strict serializability [12] that ensures transactional isolation and external consistency [25]. Stronger

SOSP 2015

Why measuring consistency violations?

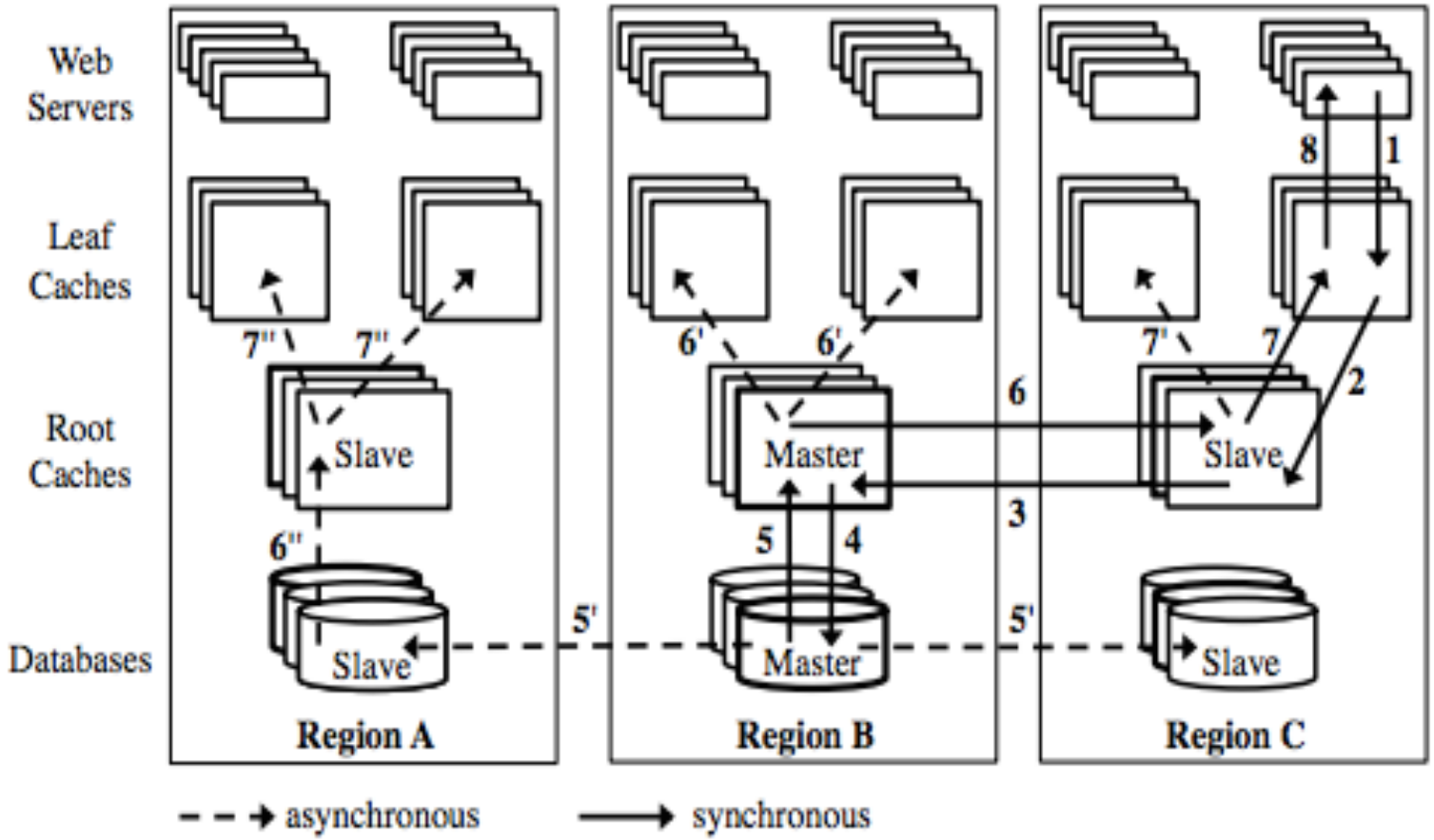
- Once the system relaxes consistency, it exposes certain anomalies to applications.
- Are anomalies prevalent or rare in deployment?
  - How frequently anomalies occur, depends on
    - how fast asynchronous replication finishes
    - how frequently writes occur
  - Types of anomalies?
  - Answered empirically through measurements.

Facebook's distributed storage: TAO graph database



slide from Facebook's TAO talk

TAO internals



1. Per-object sequential consistency for writes
2. Read-after-writes within a cache
3. Eventual consistent reads across cache

Measurement setup

- Collect traces on web servers
  - No modification to TAO
- Measurement can be very expensive
  - TAO handles 1 billion requests/sec
- Solution: Sample on objects:
  - Object: vertex in social graph
  - Log all requests to sampled objects
  - Sufficient to detect violations of local consistency models

Local Property Enables Sampling

- "… the system as a whole satisfies P whenever each individual object satisfies P."[1]

Local consistency models can be checked on a per object basis

- Local
  - Linearizability
  - Per-Object Sequential consistency
  - Read-After-Write
- Non-local:
  - Sequential consistency
  - Causal consistency

[1] M. P. Herlihy and J. M. Wing "Linearizability: A Correctness Condition for Concurrent Objects." ACM TOPLAS, 1990

Logging Details

- Logged information:
  - Start time
  - Finish time
  - Read or write
  - Value: match read with write

- Sampling rate: 1 out of 1 million objects

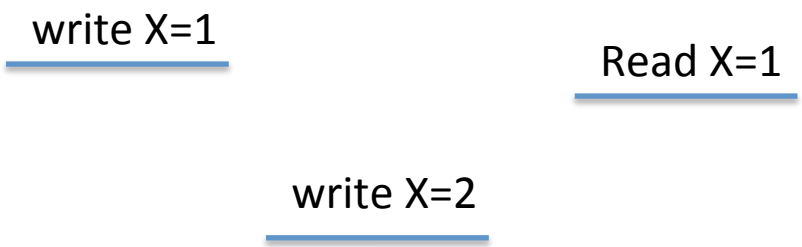
  ~100% of requests to sampled objects

What about clock skew?

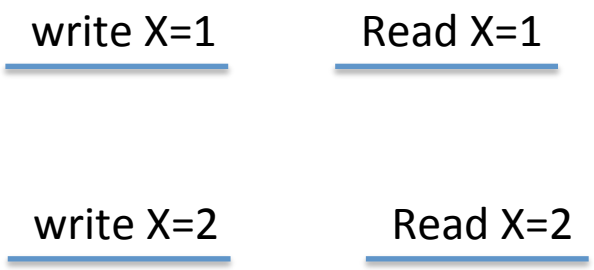
- Clock skew across web servers
  - 99.9 percentile for 1 week: 35ms
- Add slack time to request's duration
  - Subtract 35ms to invocation time, add 35ms to response time
  - Result in more overlapped requests
  - Anomalies detected represent a lower bound of true anomalies

Checking for linearizability violation

- Violations of linearizability boils down two types of anomalies
- Stale read anomaly:



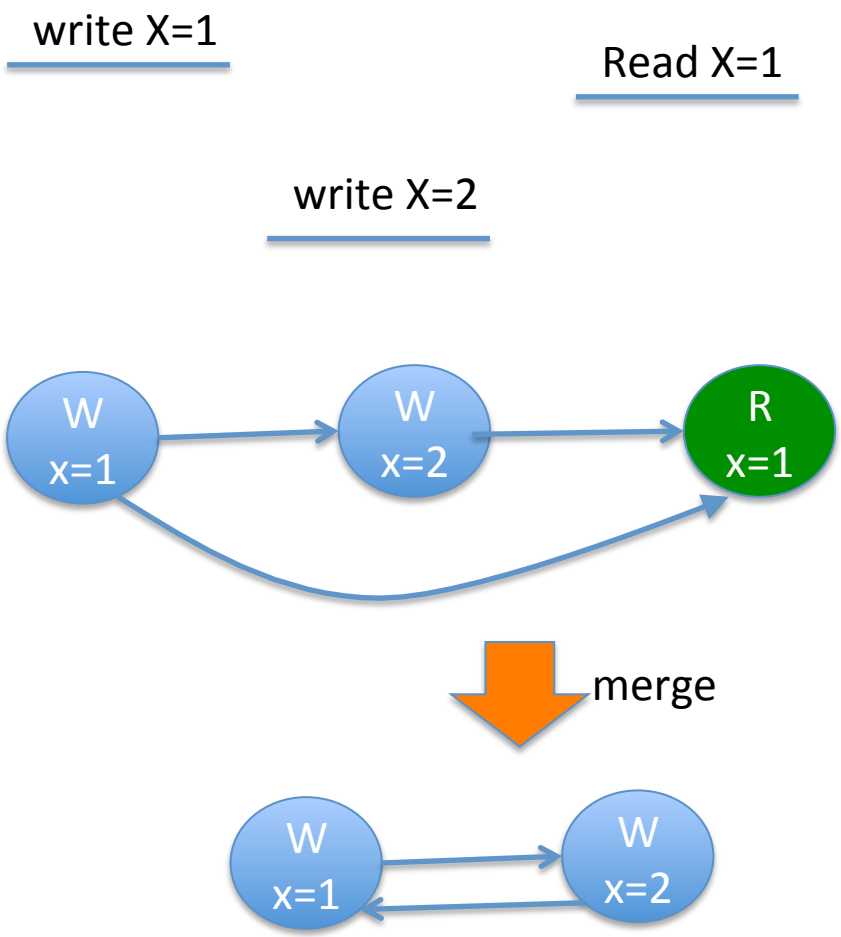
- Total order anomaly:



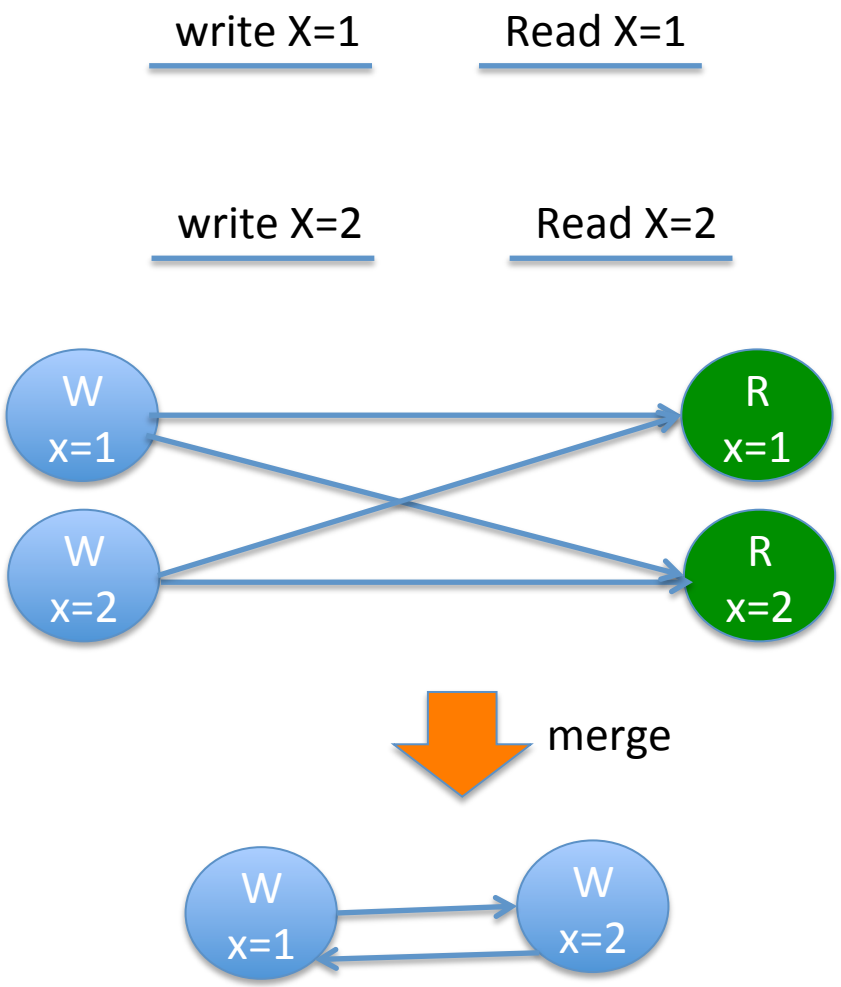
Linearizability Checker

- Graph captures state transitions
  - Vertex: write operations
  - Edge: real-time order
- Merge read with its write
  - Captures state transitions seen by users
- Anomaly if merge causes a cycle
  - Cycle indicates user's view ≠ system view

Examples: detecting read-after-write violation



Examples: detecting total order violation



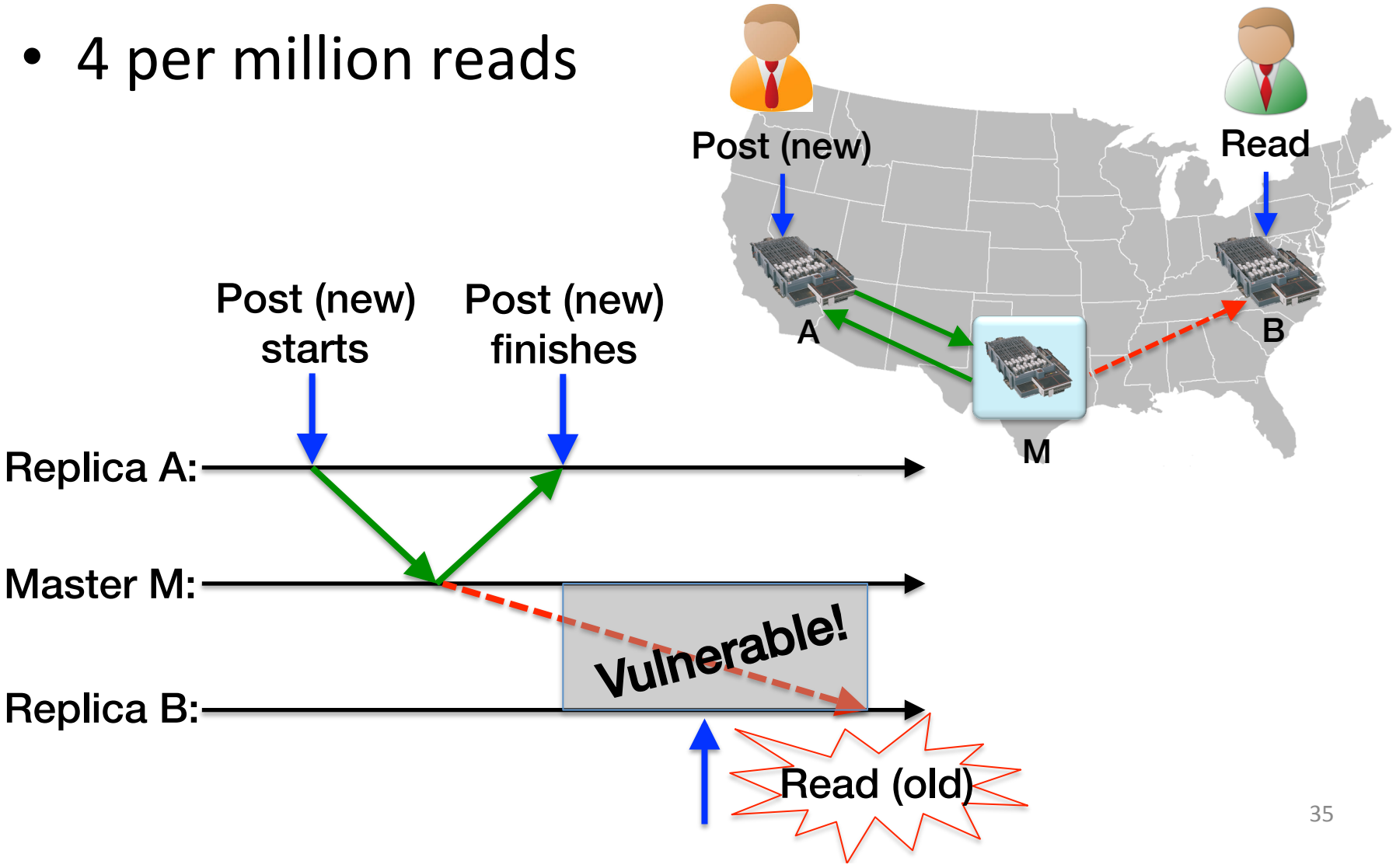
Linearizability Results

- Trace statistics (12 days, 17 million objects, 3 billion requests)
- 5 anomalies per million reads
  - Would have been prevented by a Paxos-based implementation at the cost of higher latency
- Lower bound
  - Because of clock skew adjustment

Linearizability Results

Read-after-write Violations

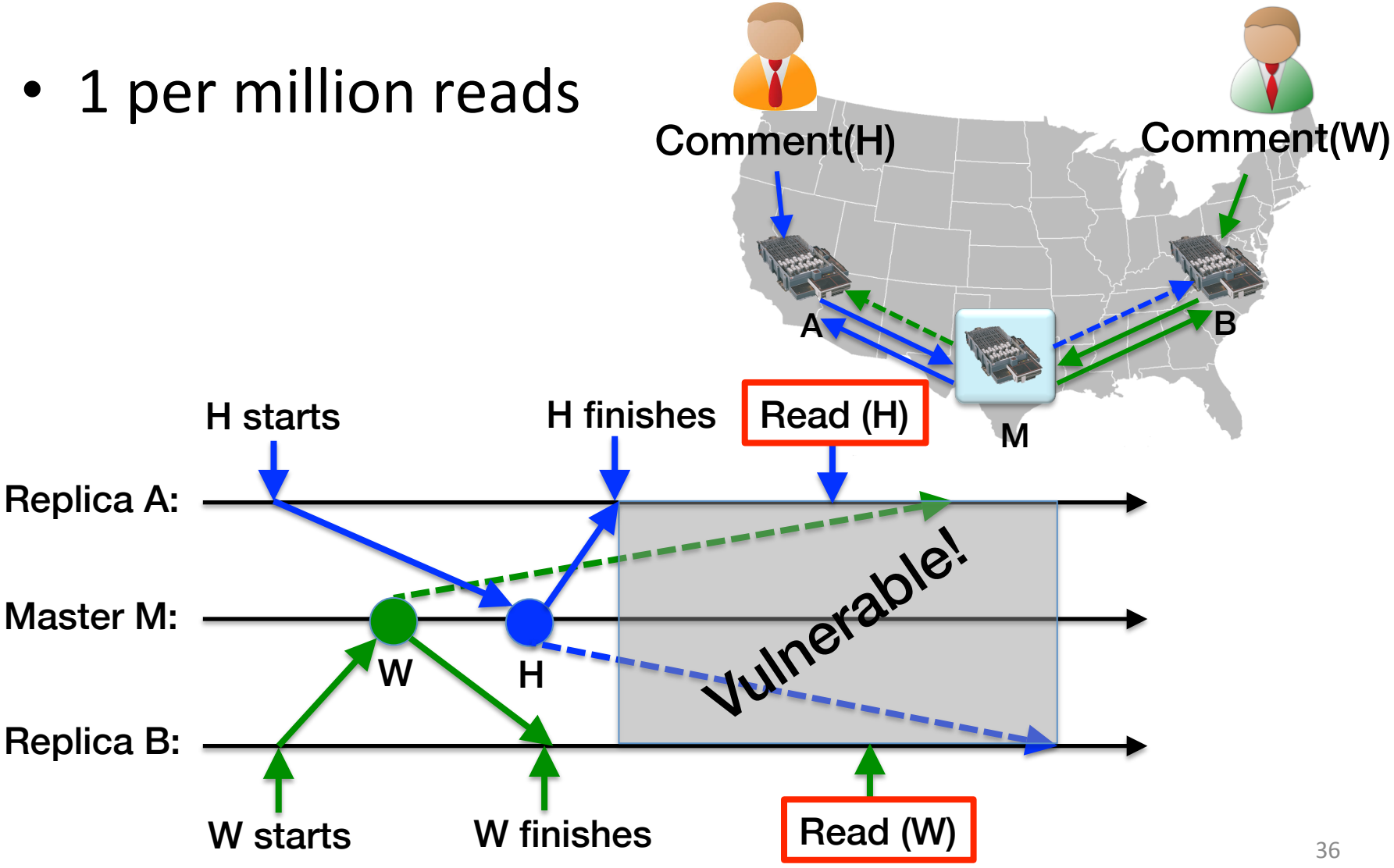
- 4 per million reads



Linearizability Results

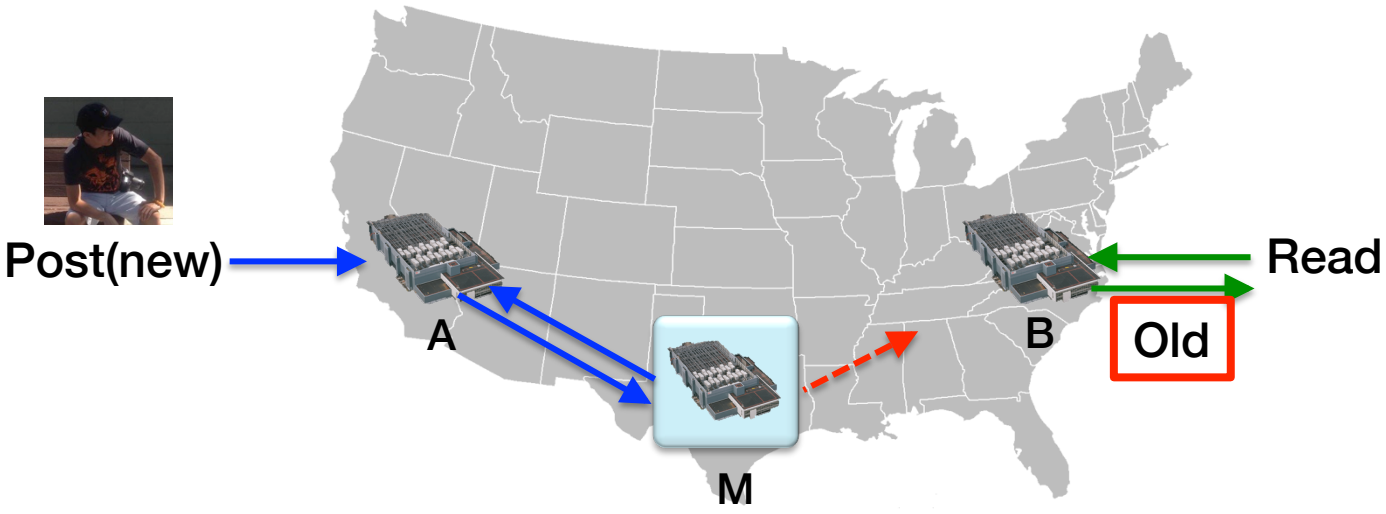
Total Order Constraint Violations

- 1 per million reads

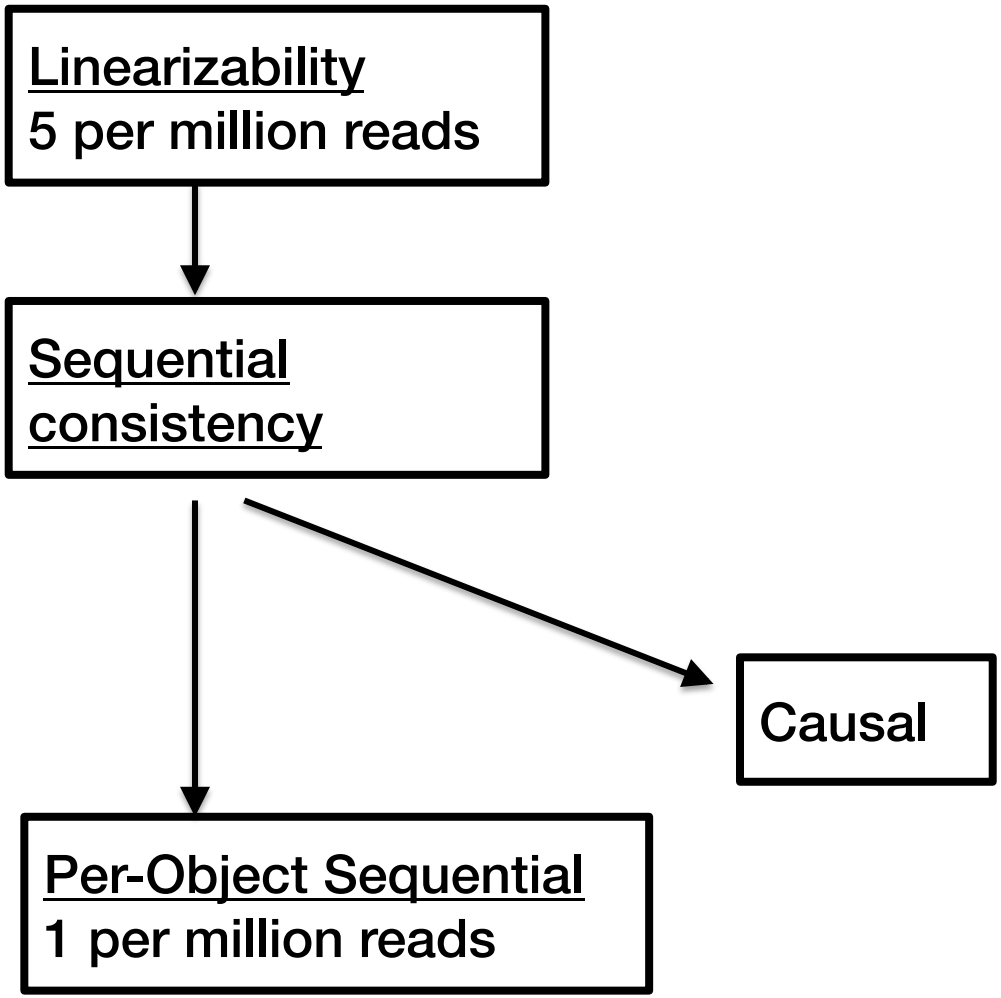


Per-Object Sequential Results

- 1 anomaly per million reads
  - User session constraint (1 per 10 million)
    - Users should see their writes



Anomaly bounds



Summary

- Per-object sequential consistency (for writes) is commonly used in industry
  - No write-write conflict (No need for conflict resolution)
  - simple/efficient to implement (compared to causal consistency)
- Certain local consistency property violations can be measured at scale
  - read-after-writes
  - total-order-violation
  - read-your-own-writes