Geo-replication

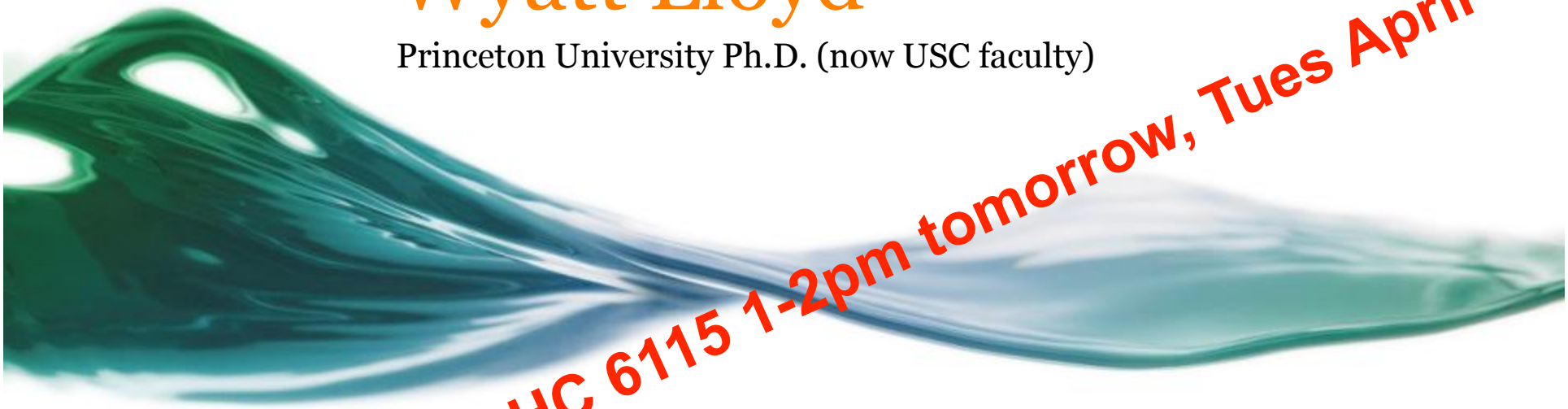
15-719
Advanced Cloud Computing

Garth Gibson
Greg Ganger
Majd Sakr

Many slides borrowed from the excellent defense talk slides of

Wyatt Lloyd

Princeton University Ph.D. (now USC faculty)



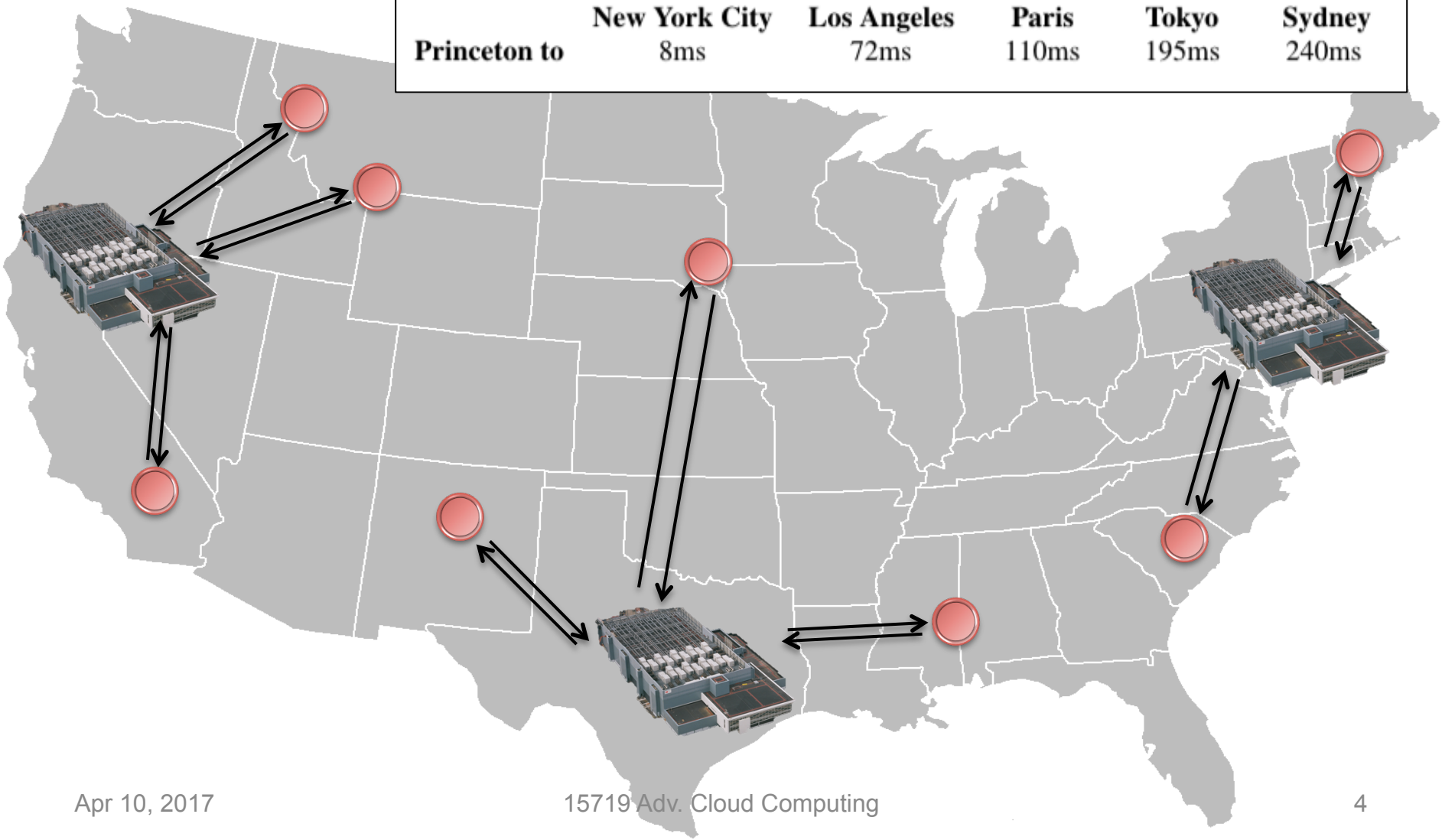
Wyatt speaks in GHC 6115 1-2pm tomorrow, Tues April 11

Why geo-replicate?

- One reason: disaster survival
  - if an entire region “fails”, others can continue
- Another reason: politics
  - some countries won’t let certain info in (censor) or out (privacy)
- Biggest reason: latency
  - lower round-trip times

Closer data centers can serve requests quicker

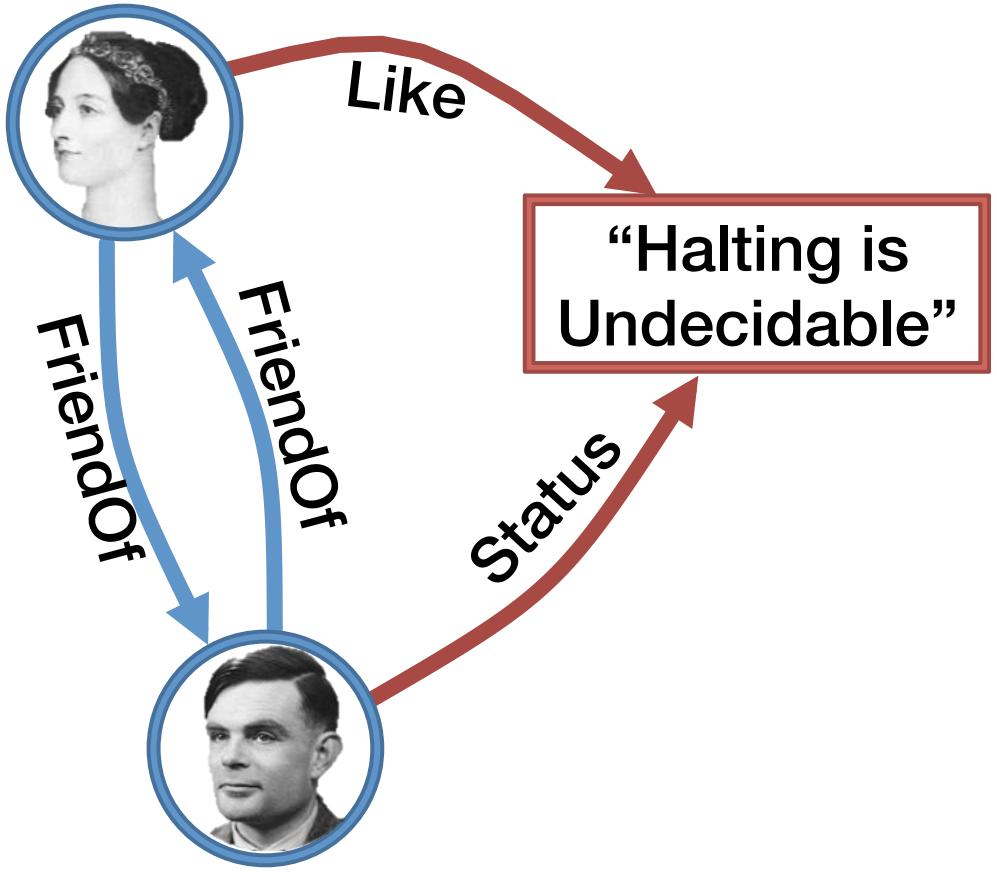
	New York City	Los Angeles	Paris	Tokyo	Sydney
Princeton to	8ms	72ms	110ms	195ms	240ms



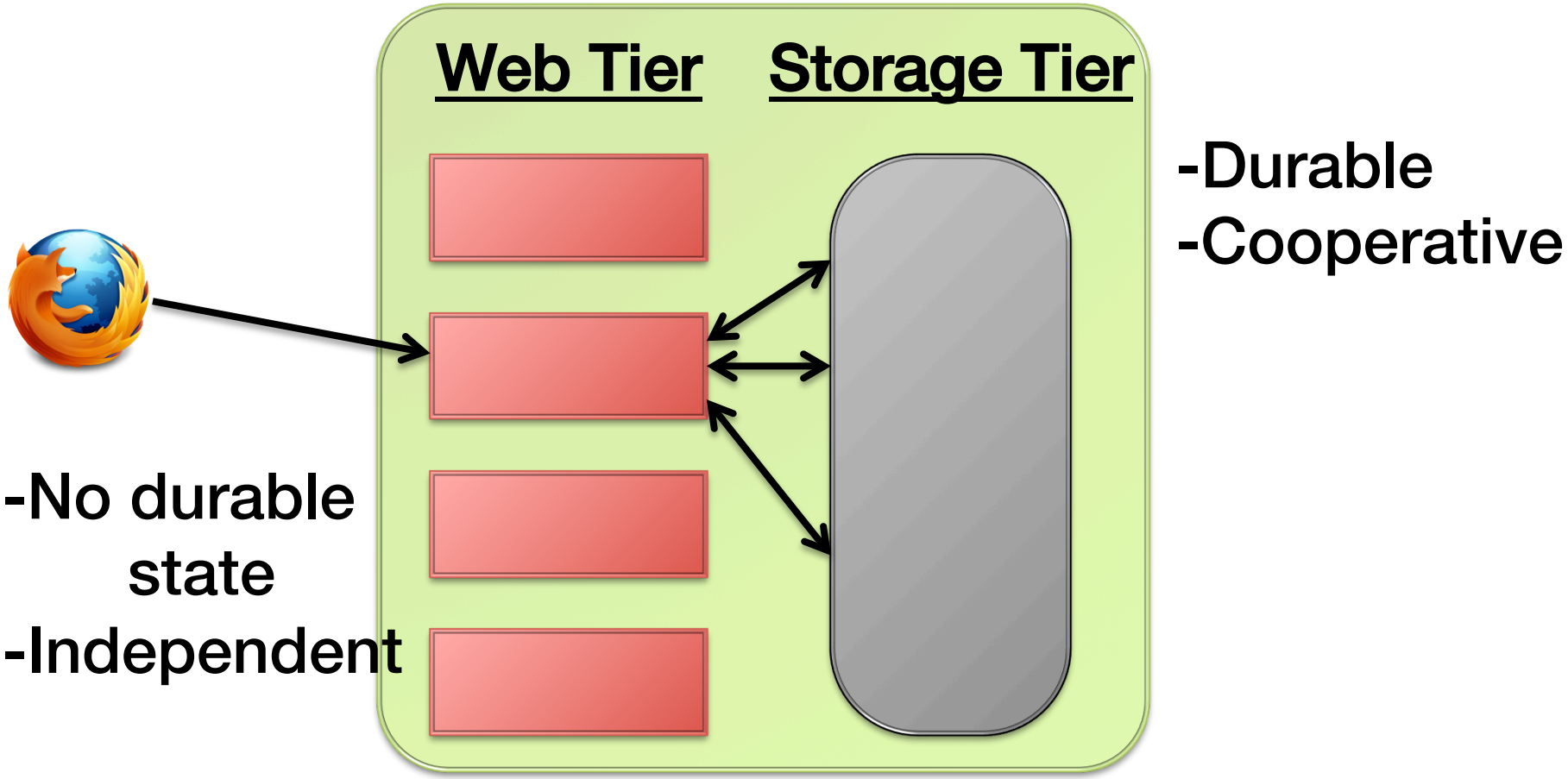
Geo-Replicated Storage

is the backend of massive websites

amazon



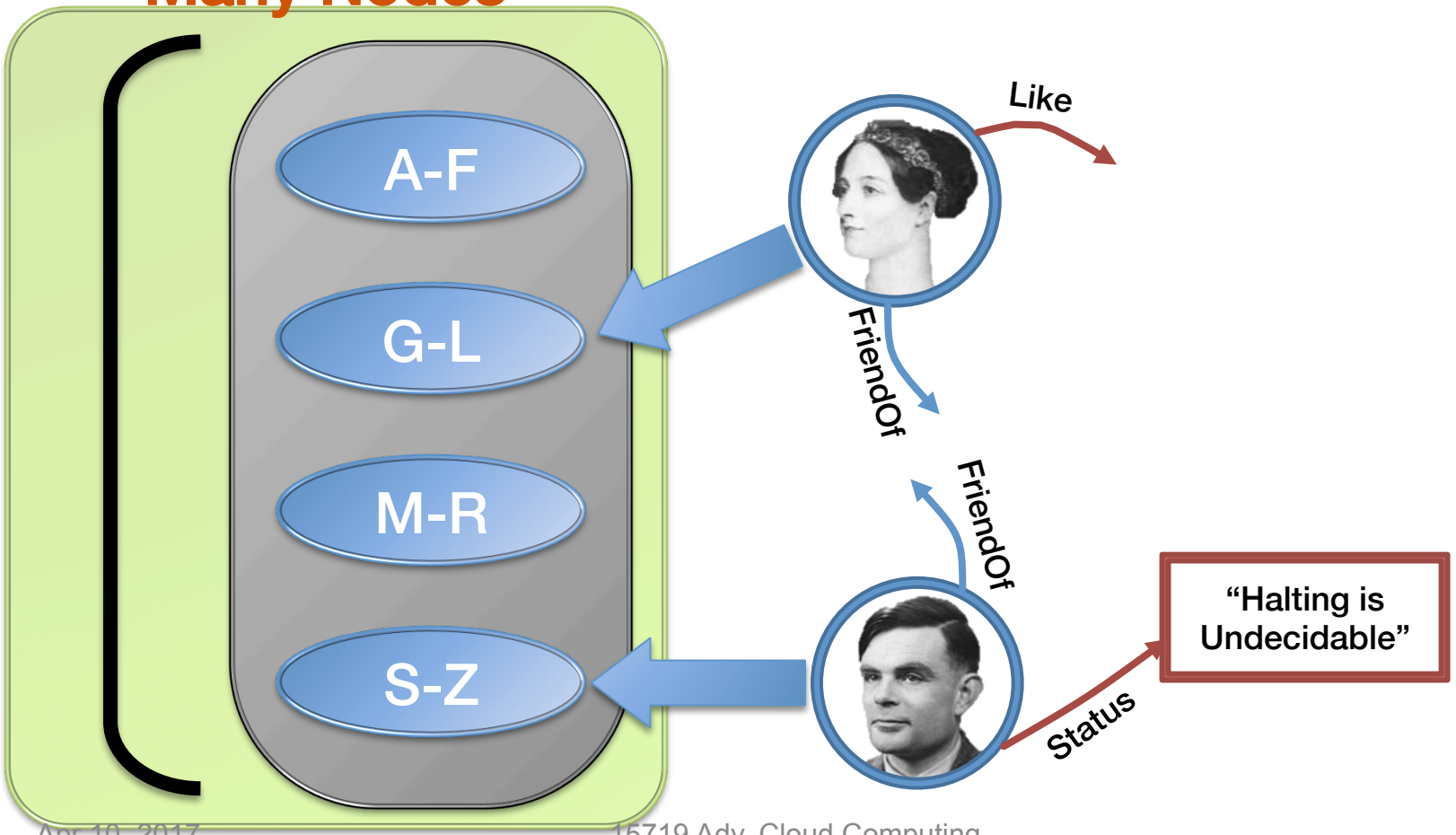
Tiers inside each Datacenter



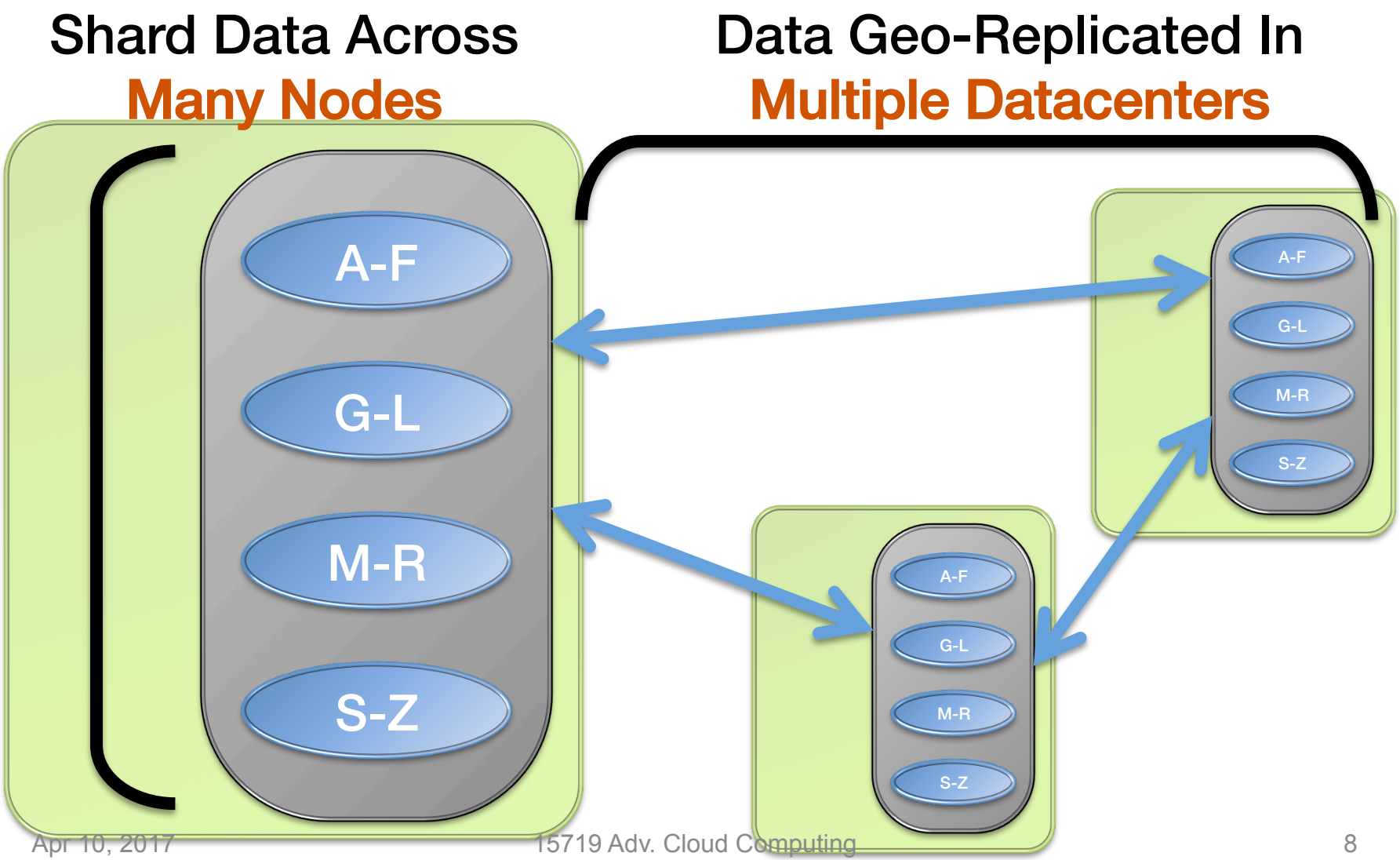
Storage Tier Dimensions

Shard Data Across

Many Nodes



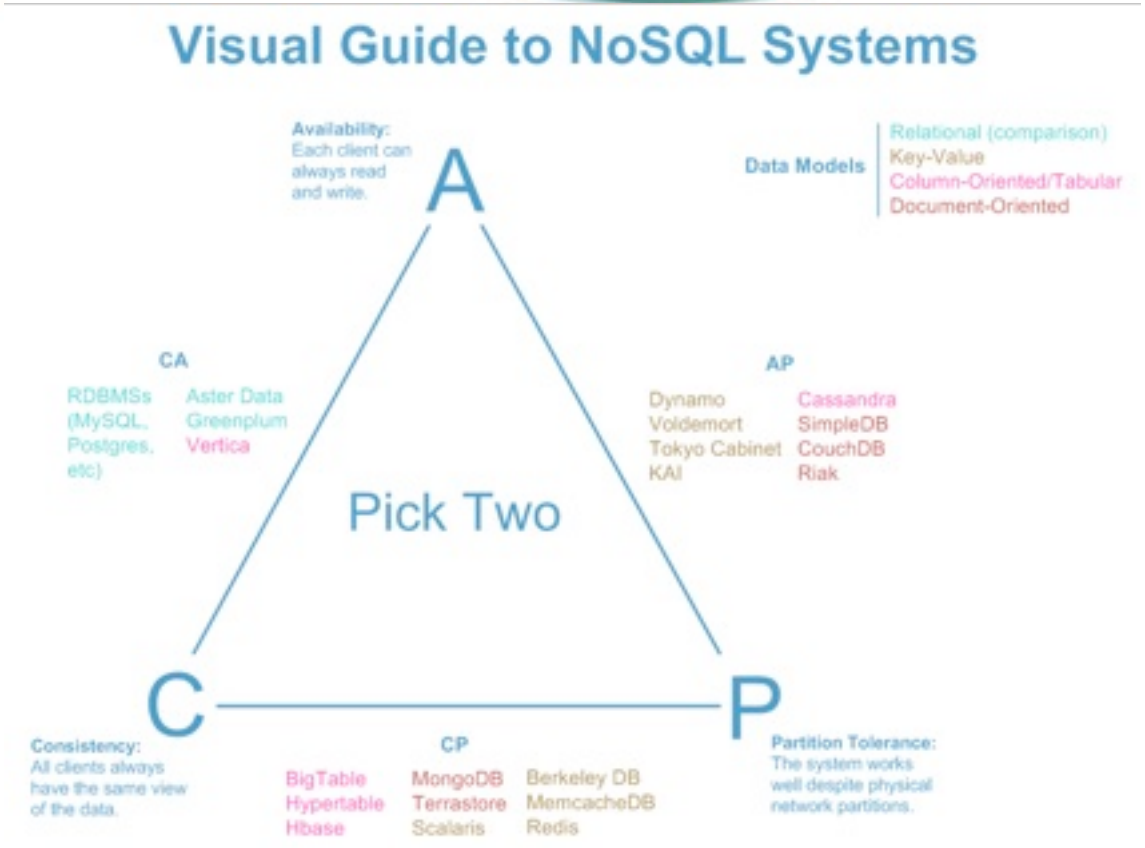
Storage Tier Dimensions



Common Geo-Replicated Storage Goals

- Serve client requests quickly
- Scale out nodes/datacenters
- Interact with data coherently

CAP Theorem

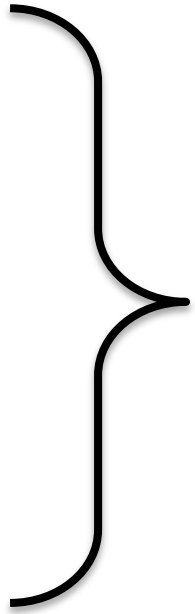


Nathan Hurst, 2010

- Eric Brewer, 1998
- You cannot always have Consistency, Availability, and Partition tolerance
- Lynch/Gilbert 2002 proved the extreme case
- Reality is that partition is rare, but during partition you have to pick between consistency (stop & wait) or availability (access stale data)

Many systems today provide "ALPS Properties"

- **A**vailability
- **L**ow Latency
  = O(Local RTT)
- **P**artition Tolerance
- **S**calability



"Always On"

In ALPS-oriented systems, each replica "independent"

- Any request can be serviced by any data center
  - read or write
  - no coordination with other data centers
- Updates propagated to other data centers in the background
  - essentially, updates are logged and streamed to other sites
    - may be done update-by-update or as atomic batches
  - "eventual consistency" means no guarantees on when

So, ALPS-oriented Geo-Replicated Storage Achieves

√ Serve client requests quickly

√ Scale out nodes/datacenter

- But, often users would like to interact with data coherently
  - Stronger consistency
  - Stronger semantics

Consistency

- Guarantees on the shared view across the system
  - For example, which writes is a reader guaranteed to see?
- Restricts order/timing of operations
- Stronger consistency…
  - Makes programming easier
  - Makes user experience better

Strong Consistency: Linearizability

- [Herlihy Wing '90]
- Ensures a total order of operations
- The order agrees with “real time”
- West coast reads see east coast writes

Consistency with ALPS

Linearizability	Impossible [Brewer '00, Gilbert Lynch '02]	
Serializability	Impossible [Lipton Sandberg '88, Attiya Welch '94]	
Sequential		
Causal	**Wyatt Lloyd's work (and others')**	
"Eventual"	Amazon Dynamo	Facebook/Apache Cassandra

ALPS versus Strongest Consistency

- The choice is fundamental

- Amazon's Dynamo [DeCandia et al. SOSP '07]:

> **Dynamo [provides] an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency...."**

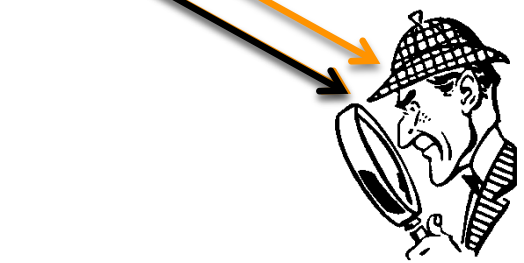
- Wyatt argues: Don't settle for eventual consistency

Causality By Example

Remove boss from friends group

Post to friends: "Time for a new job!"

Friend reads post



Causality (⟶)

Thread-of-Execution

Reads-From

Transitivity

Users Love Causality

Because sites work as expected

↓ Then ↓

New Job!

Employment retained

↓ Then ↓

Purchase retained

↓ Then ↓

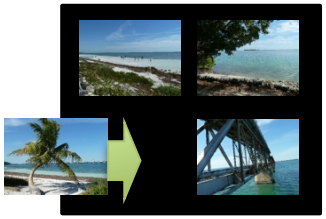
Error
404 – File not found

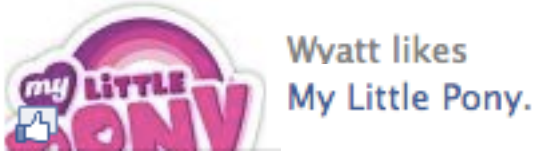
Deletion retained

Programmers Love Causality

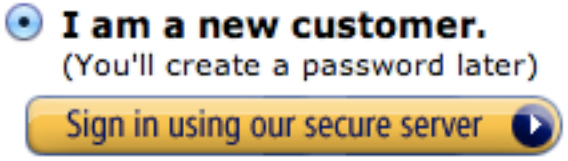
Because it simplifies programming

↓ Then ↓





↓ Then ↓

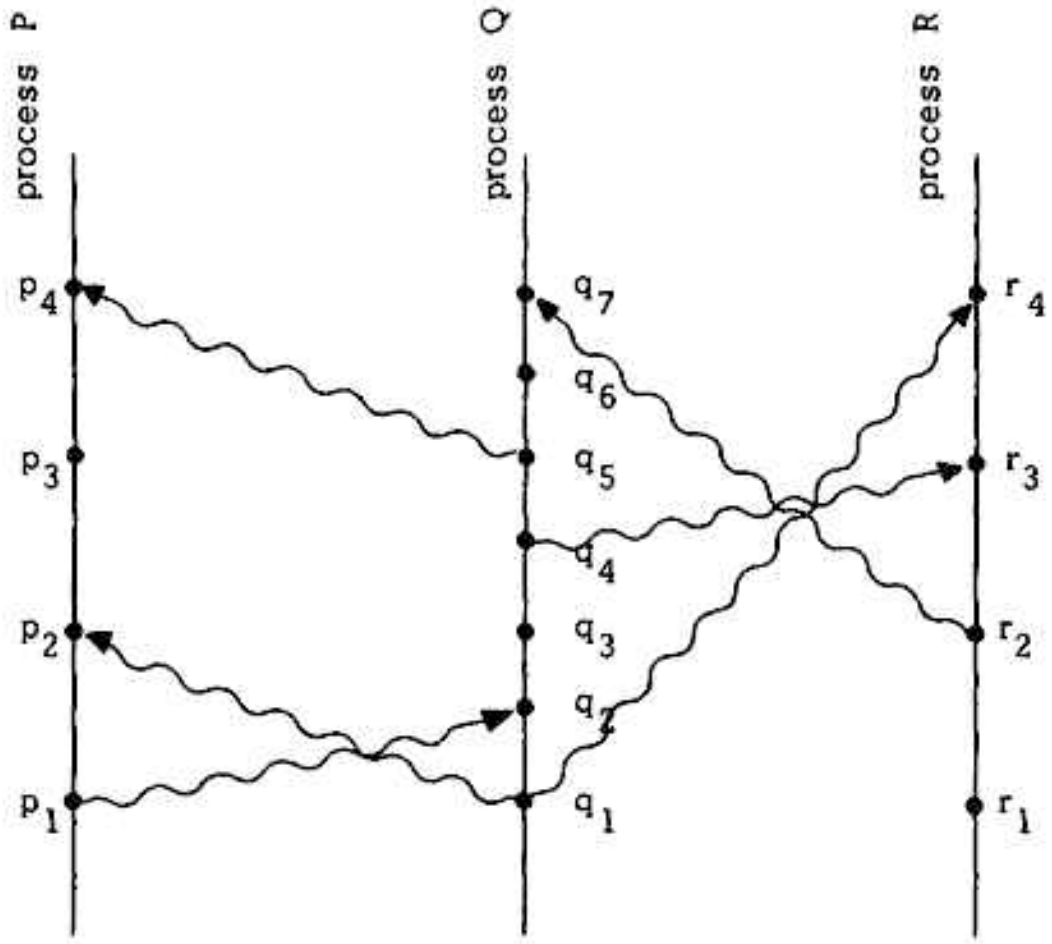


↓ Then ↓

No reasoning about out-of-order operations

Remember "Happens Before"

- Two events are not concurrent if one "happens before" the other
- Eg. P1 happens before R3 but P2 and R4 may be concurrent
- Rename happens before as *potential causality*



Causal Consistency vs Eventual Consistency

- Causal Consistency requires all values returned by reads to be consistent with all potential causality relationships (partial ordering)
  - Note that no potential causality means logically concurrent
  - Conflicts are logically concurrent operations where ordering matters
    - Writes to replicas without a message path between them
  - Conflict resolution mustbe deterministic (later observer sees same result)
    - E.g. “last writer wins”, or fenced for user resolution (Coda)
- Eventual consistency does not strive to maximize potential causality

Achieving Good Consistency given ALPS

- Assign versions (logical clock or physical clock) so
  - version ordering is consistent with potential causality, and
  - resolves conflicts deterministically
- Replicas log an order
  - Record version info
- Replicas converge
  - Exchange logs, select a deterministic ordering, and apply it

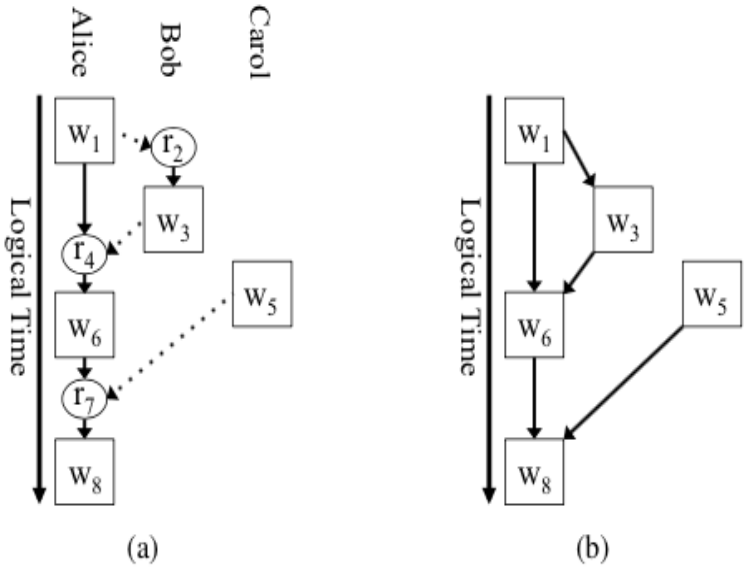
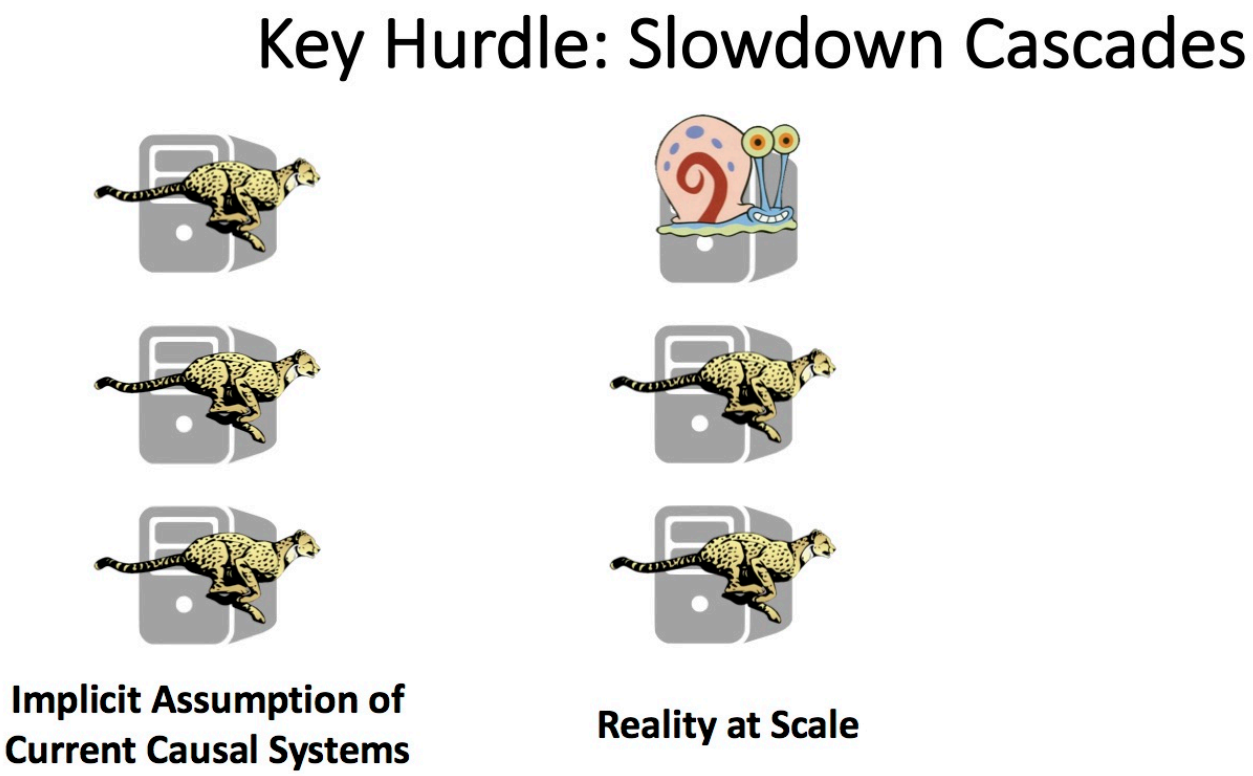


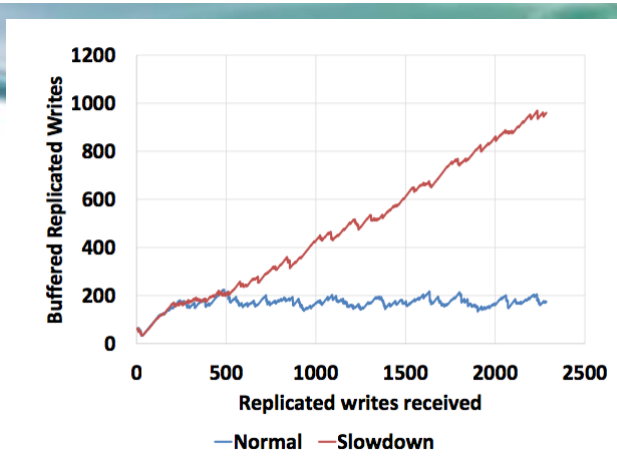
Figure 3.3: A graph of causality is shown in (a) and the corresponding dependency graph is shown in (b).

A little later, bumps in the road…

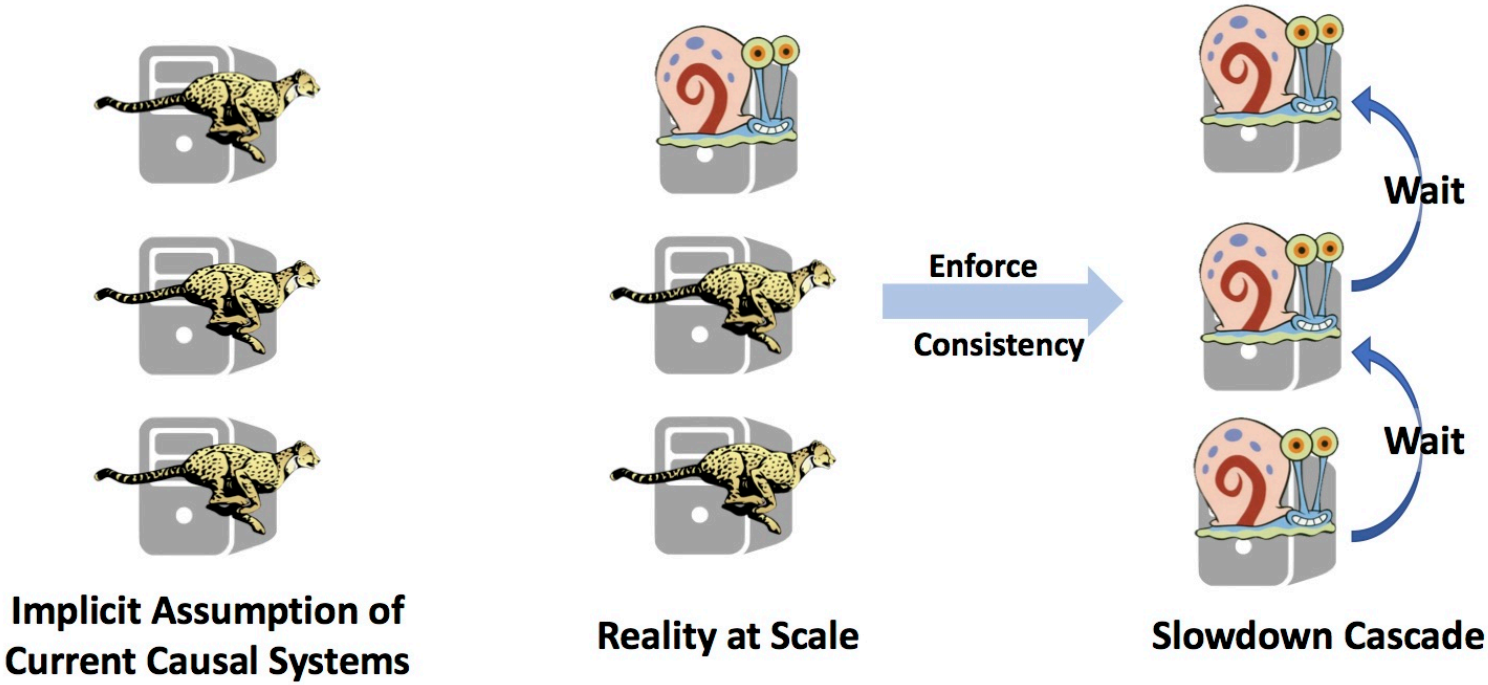


[Mehdi-NSDI17]

A little later, bumps in the road…

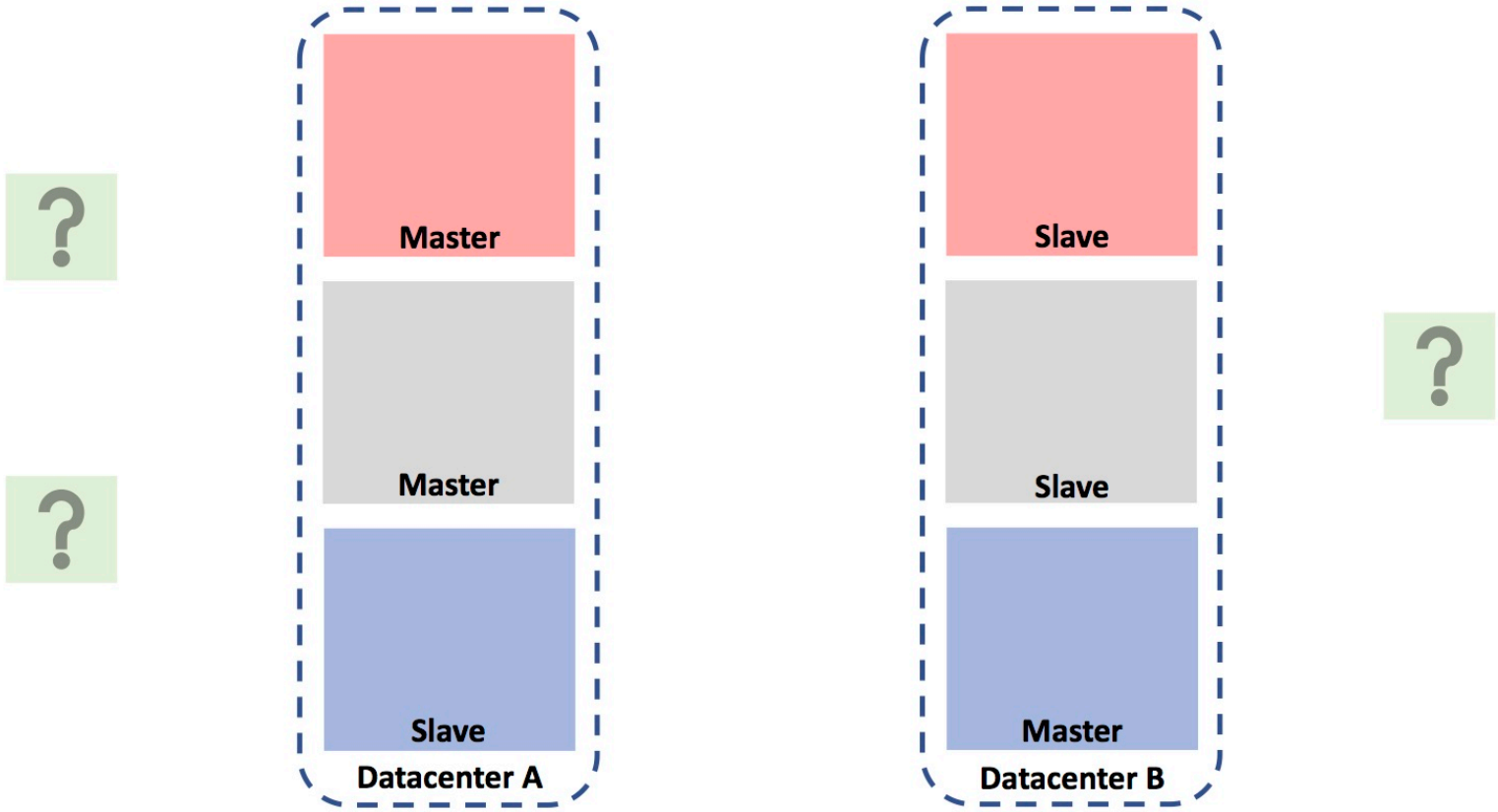


Key Hurdle: Slowdown Cascades



[Mehdi-NSDI17]

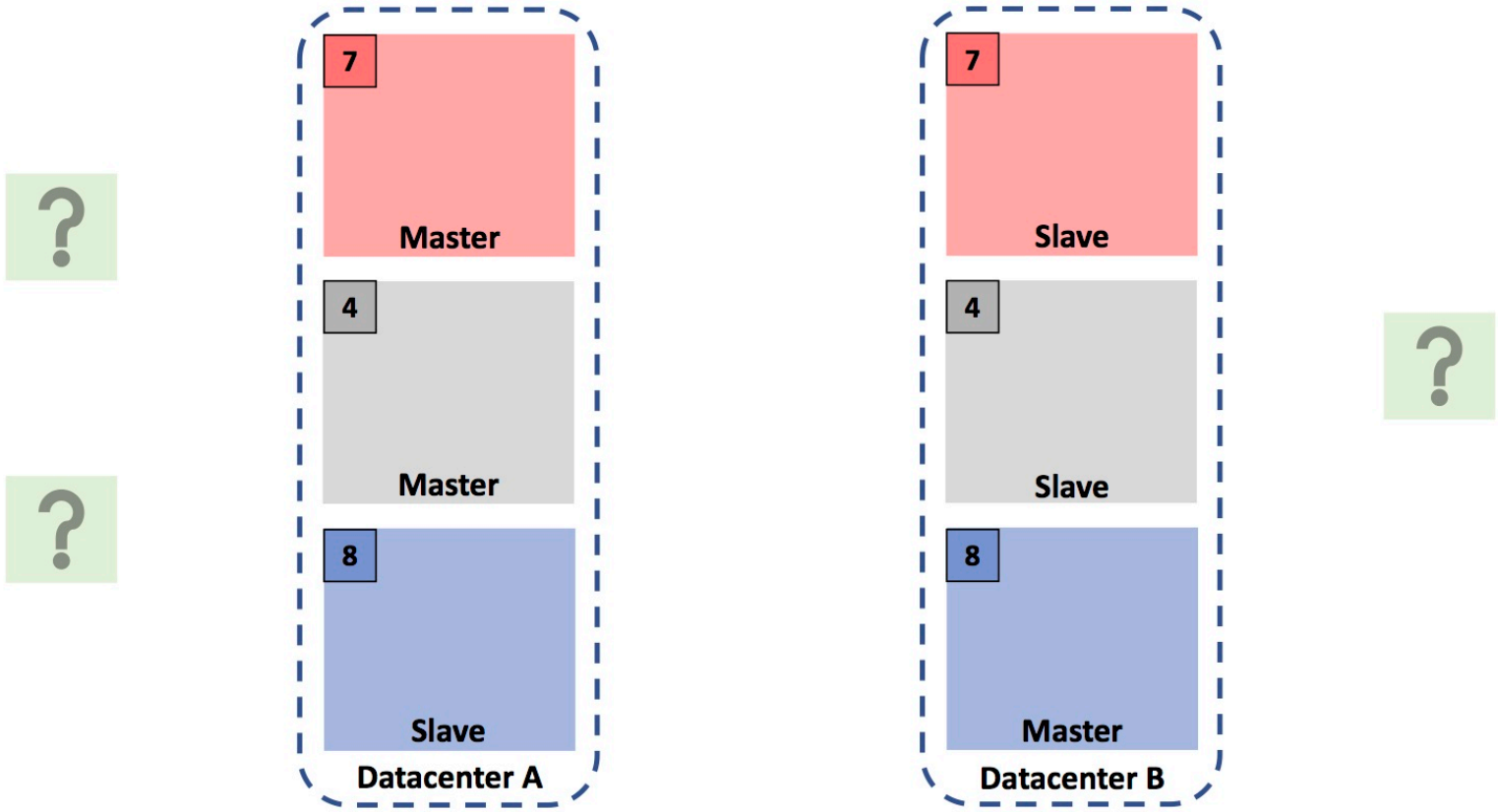
Push waiting out of store to client



Writes accepted only by master shards and then replicated asynchronously in-order to slaves

[Mehdi-NSDI17]

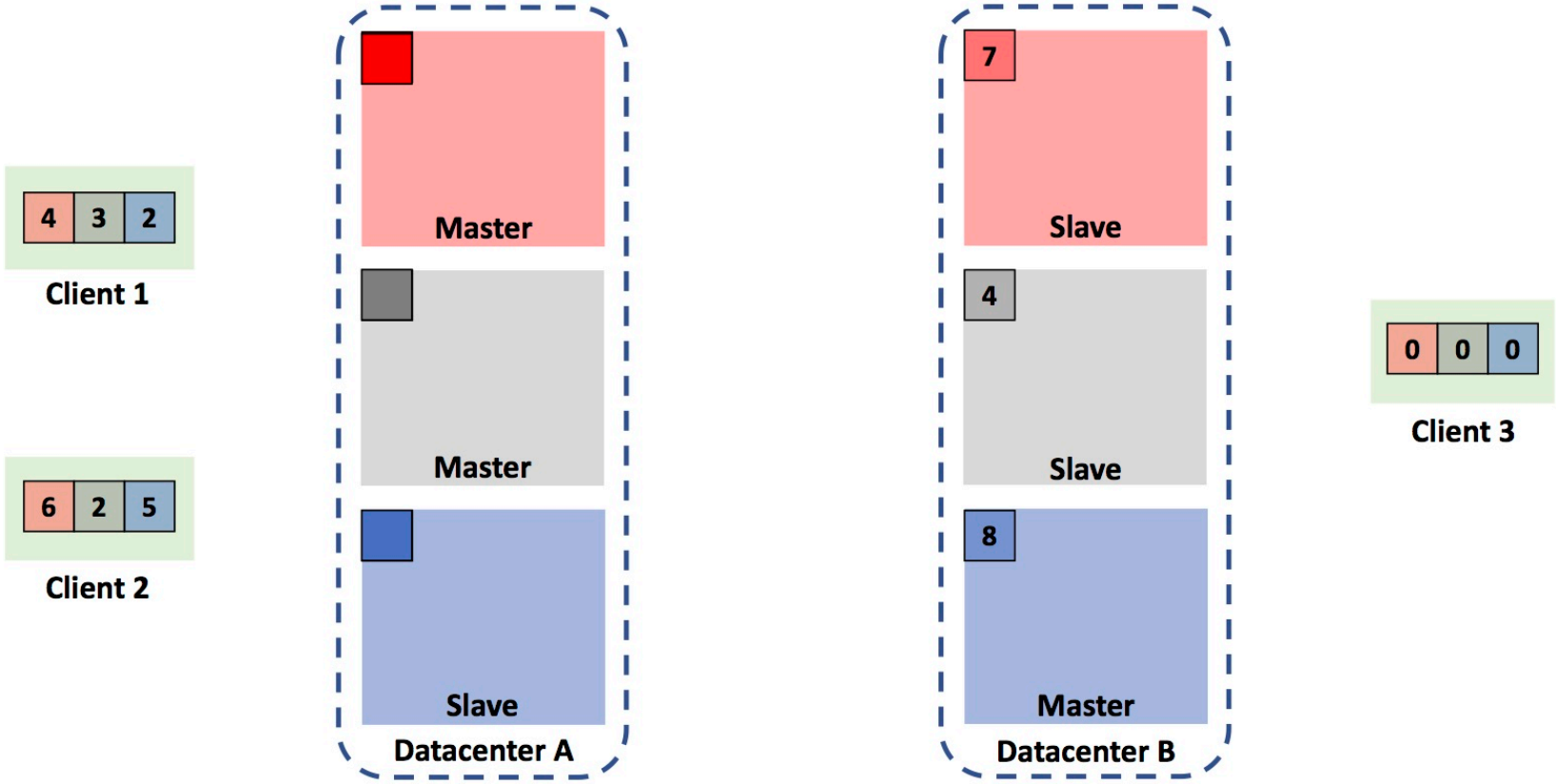
Push waiting out of store to client



Each shard keeps track of a **shardstamp** which counts the writes it has applied

[Mehdi-NSDI17]

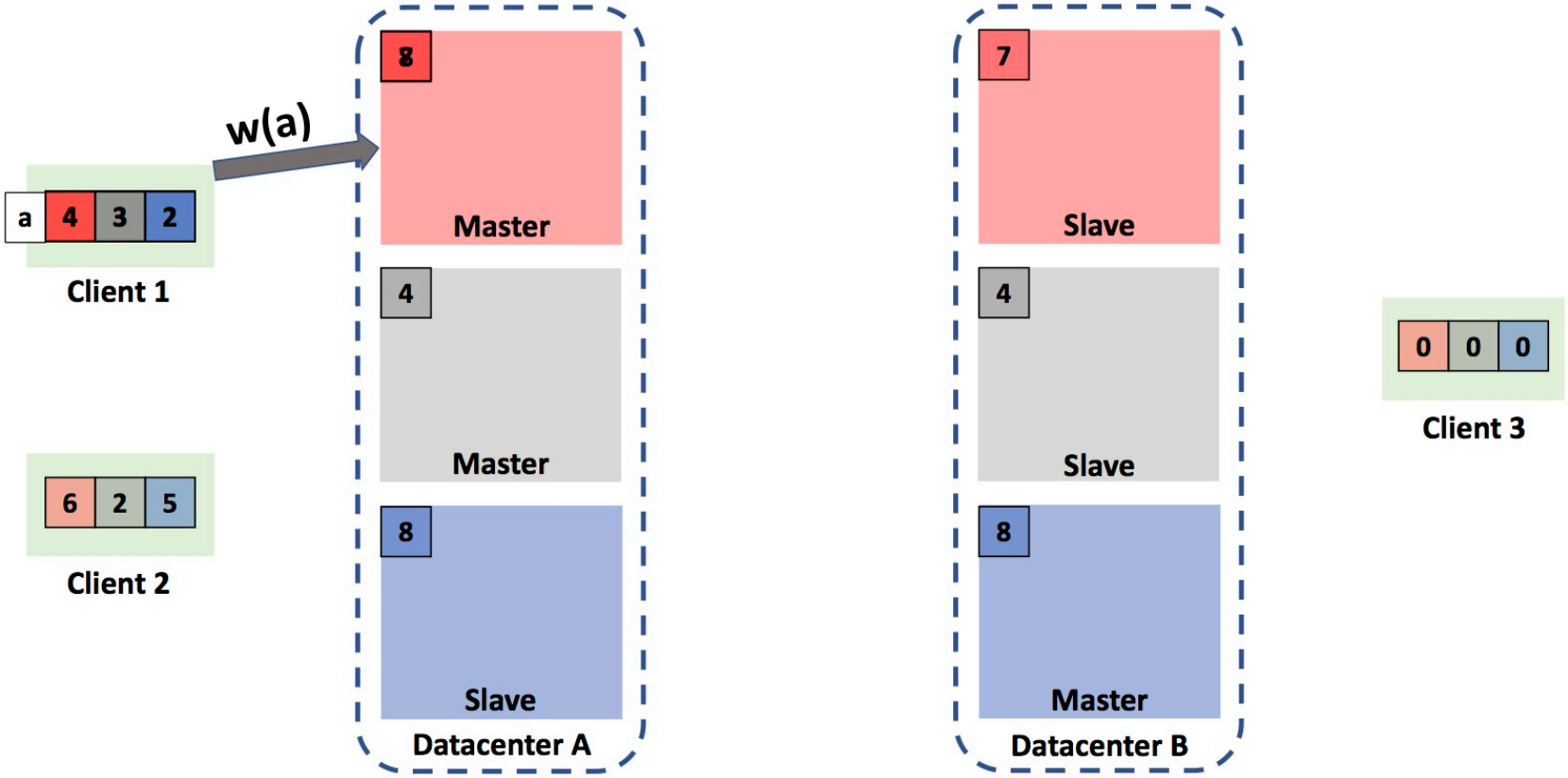
Push waiting out of store to client



Causal Timestamp: Vector of shardstamps which identifies a global state across all shards

[Mehdi-NSDI17]

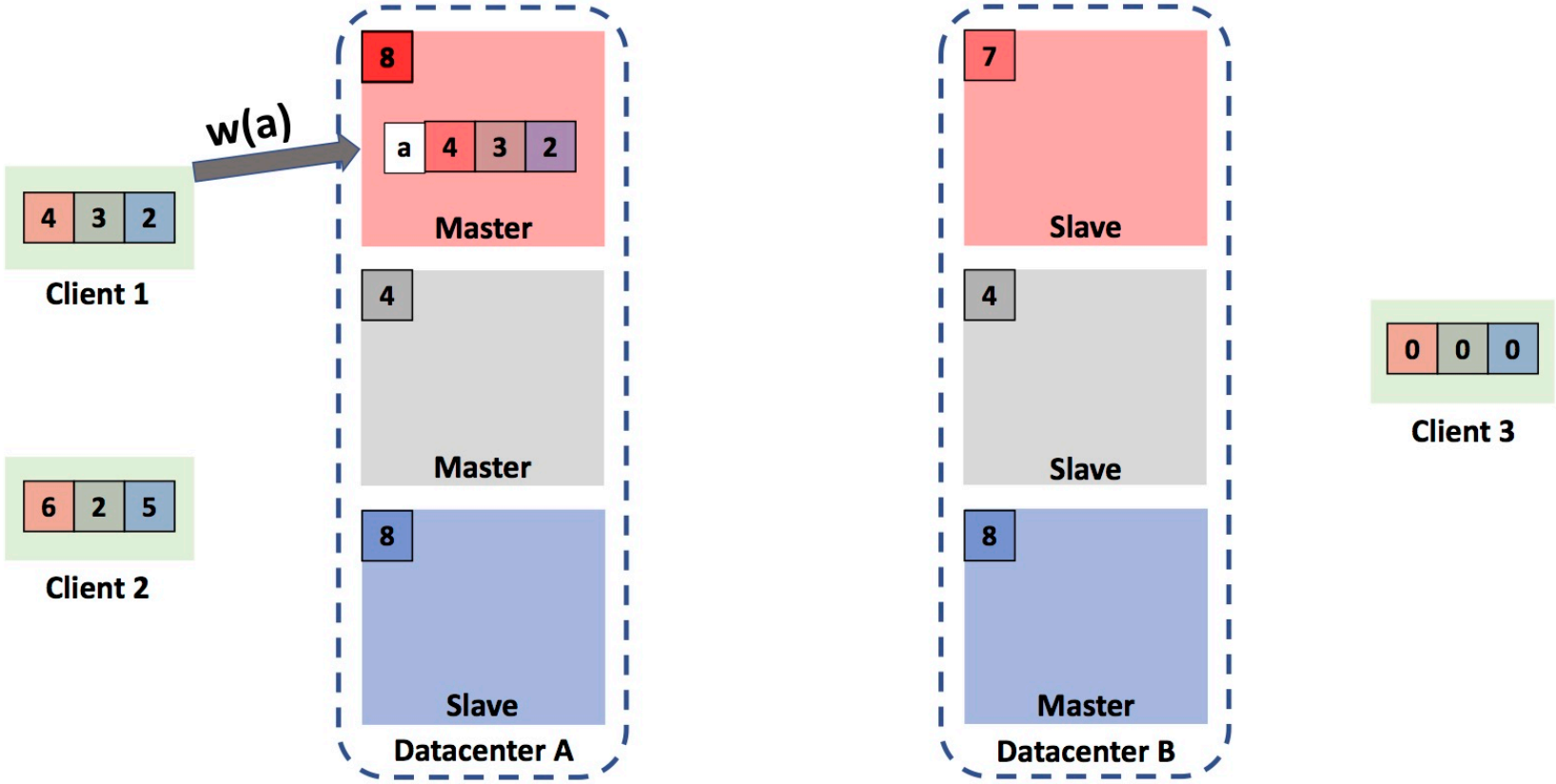
Push waiting out of store to client



Write Protocol: Causal timestamps stored with objects to propagate dependencies

[Mehdi-NSDI17]

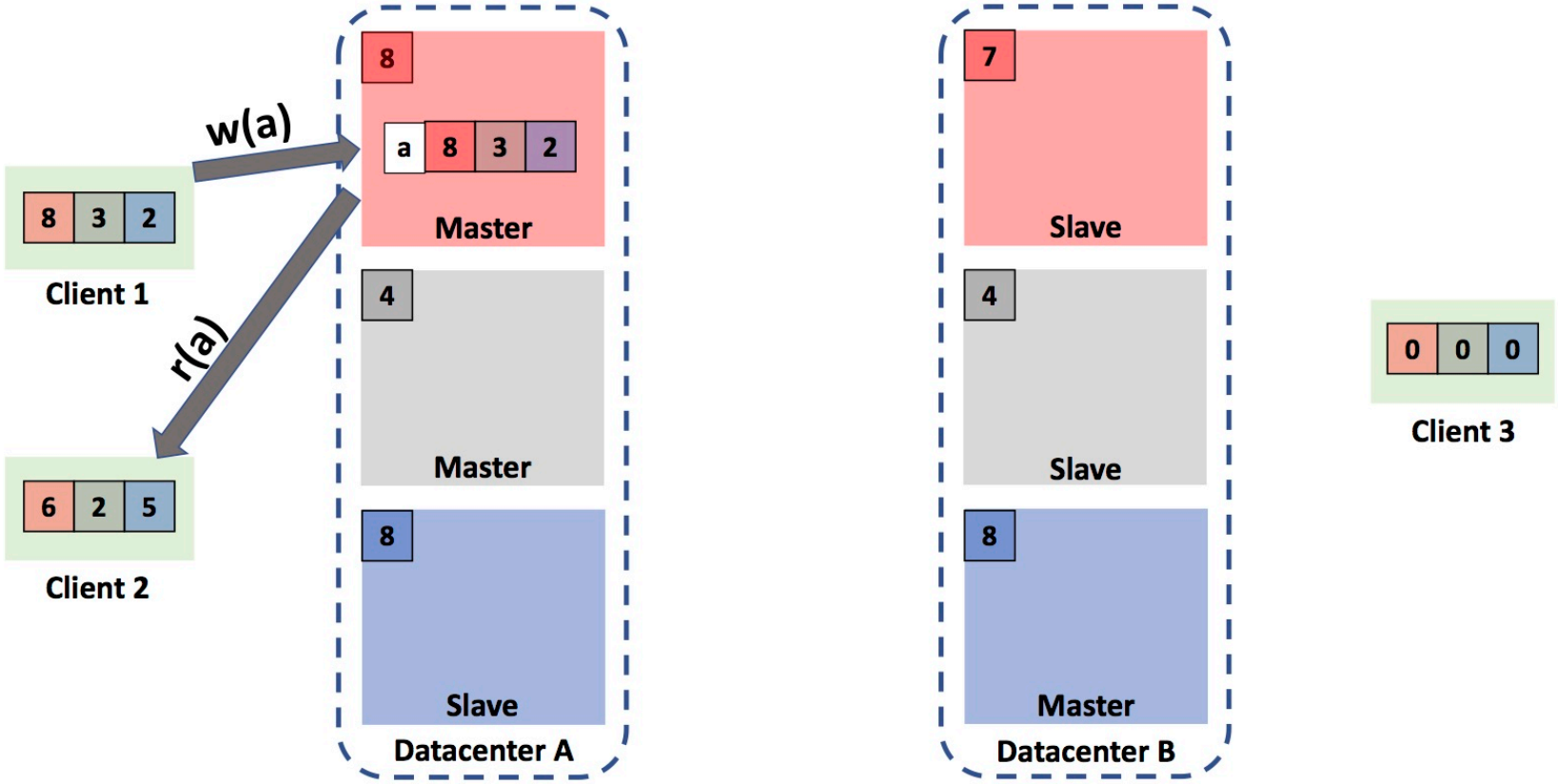
Push waiting out of store to client



Write Protocol: Server shardstamp is incremented and merged into causal timestamps

[Mehdi-NSDI17]

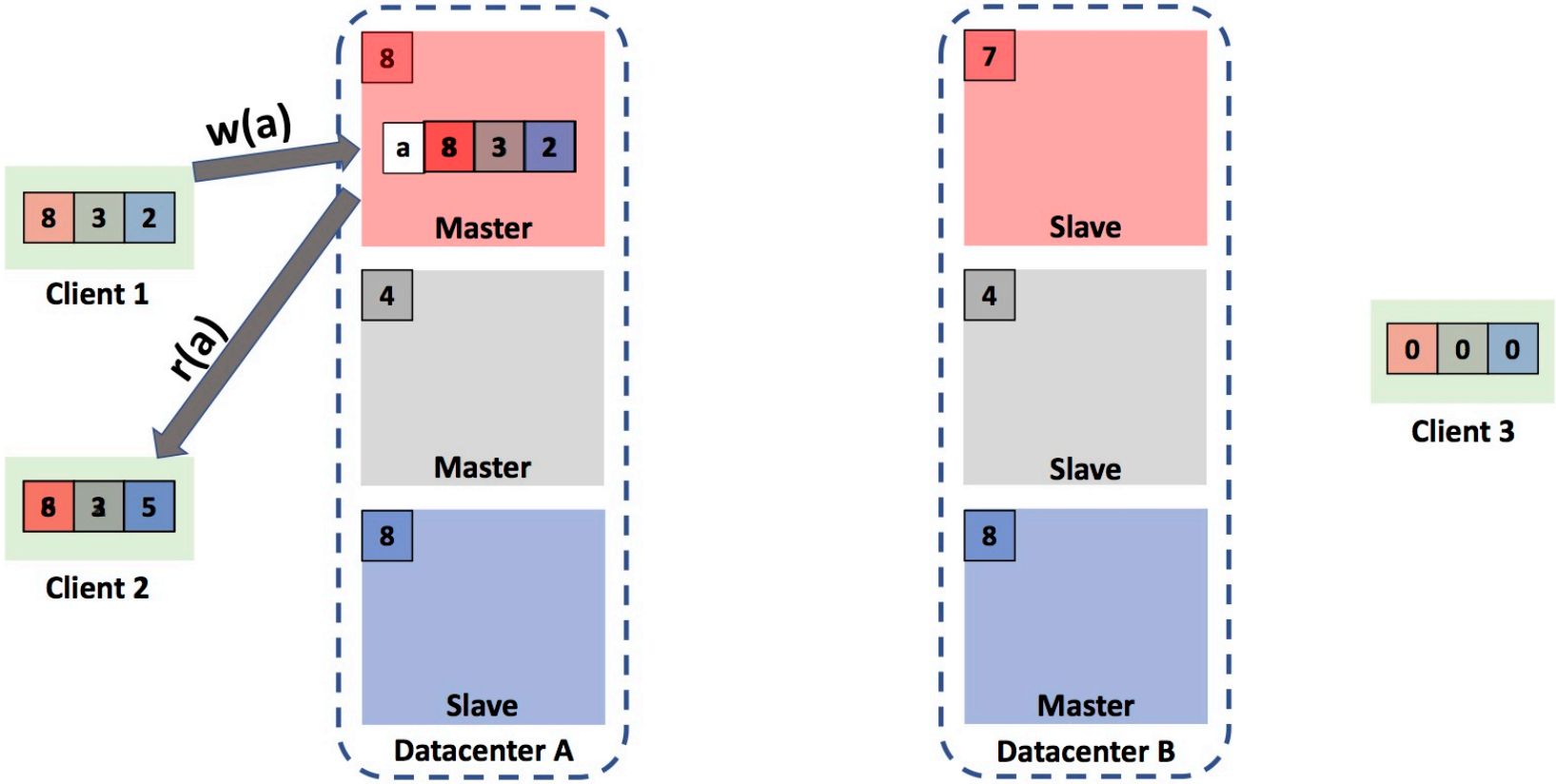
Push waiting out of store to client



Read Protocol: Always safe to read from master

[Mehdi-NSDI17]

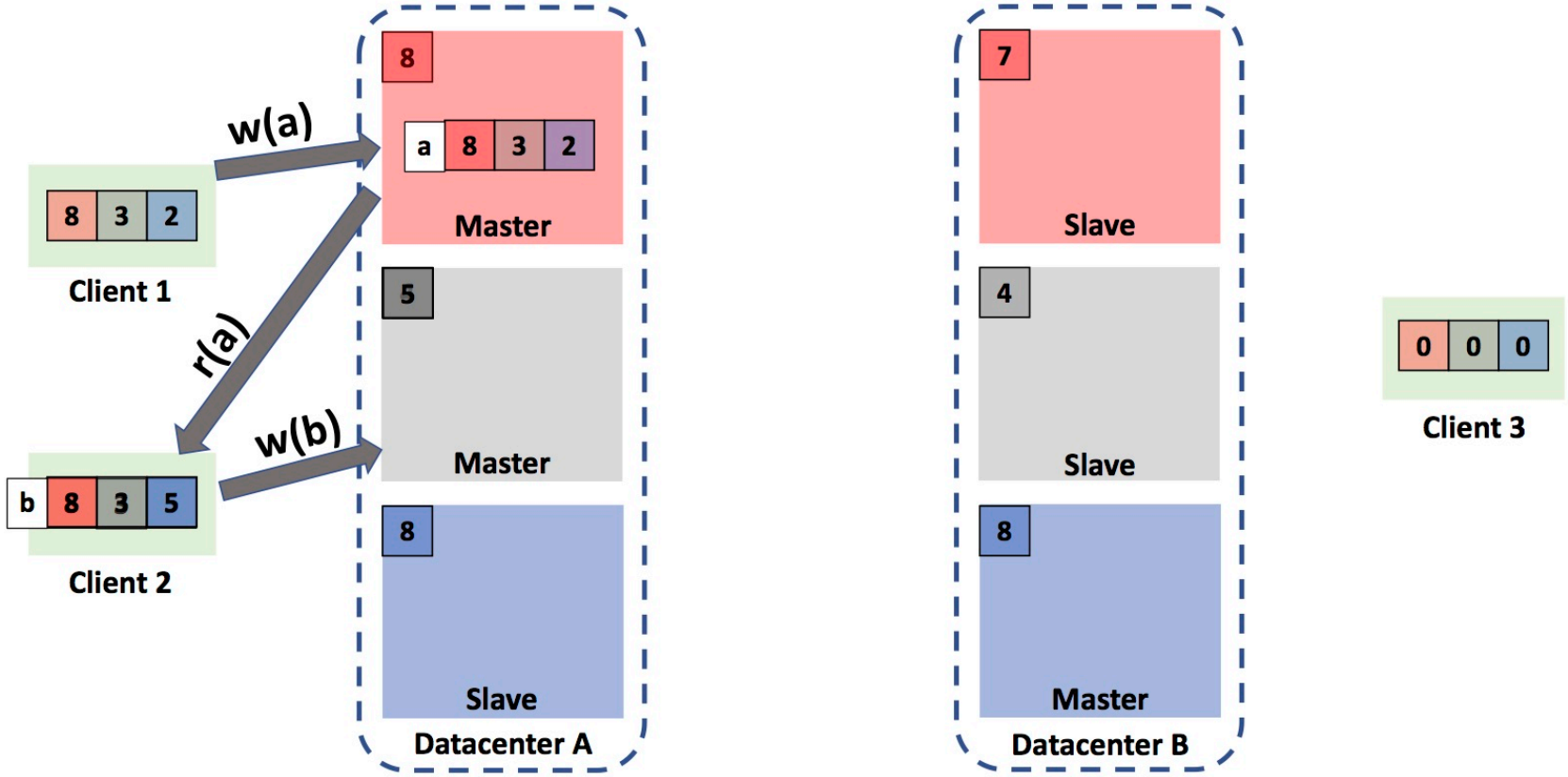
Push waiting out of store to client



Read Protocol: Object's causal timestamp merged into client's causal timestamp

[Mehdi-NSDI17]

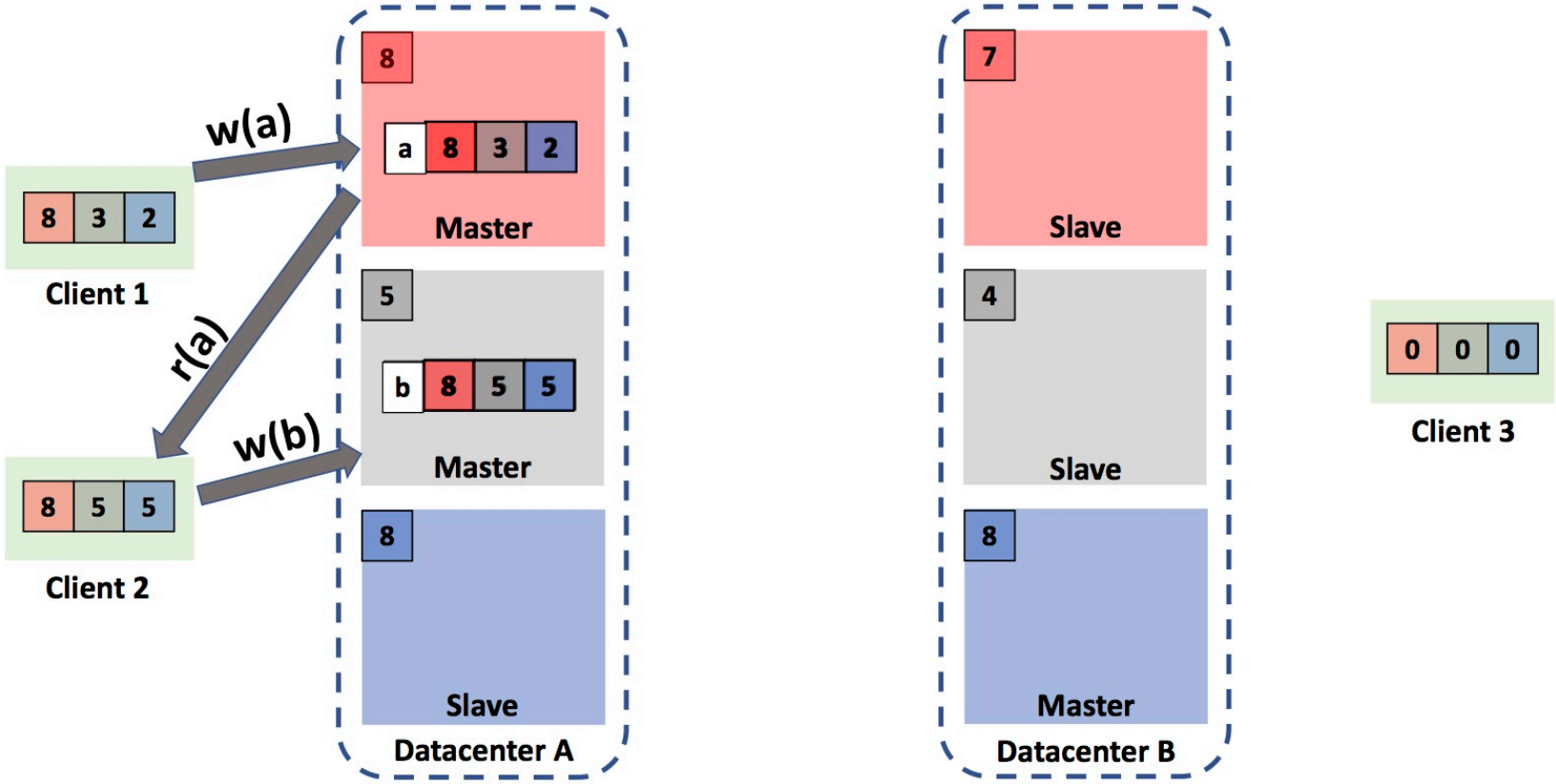
Push waiting out of store to client



Read Protocol: Causal timestamp merging tracks causal ordering for writes following reads

[Mehdi-NSDI17]

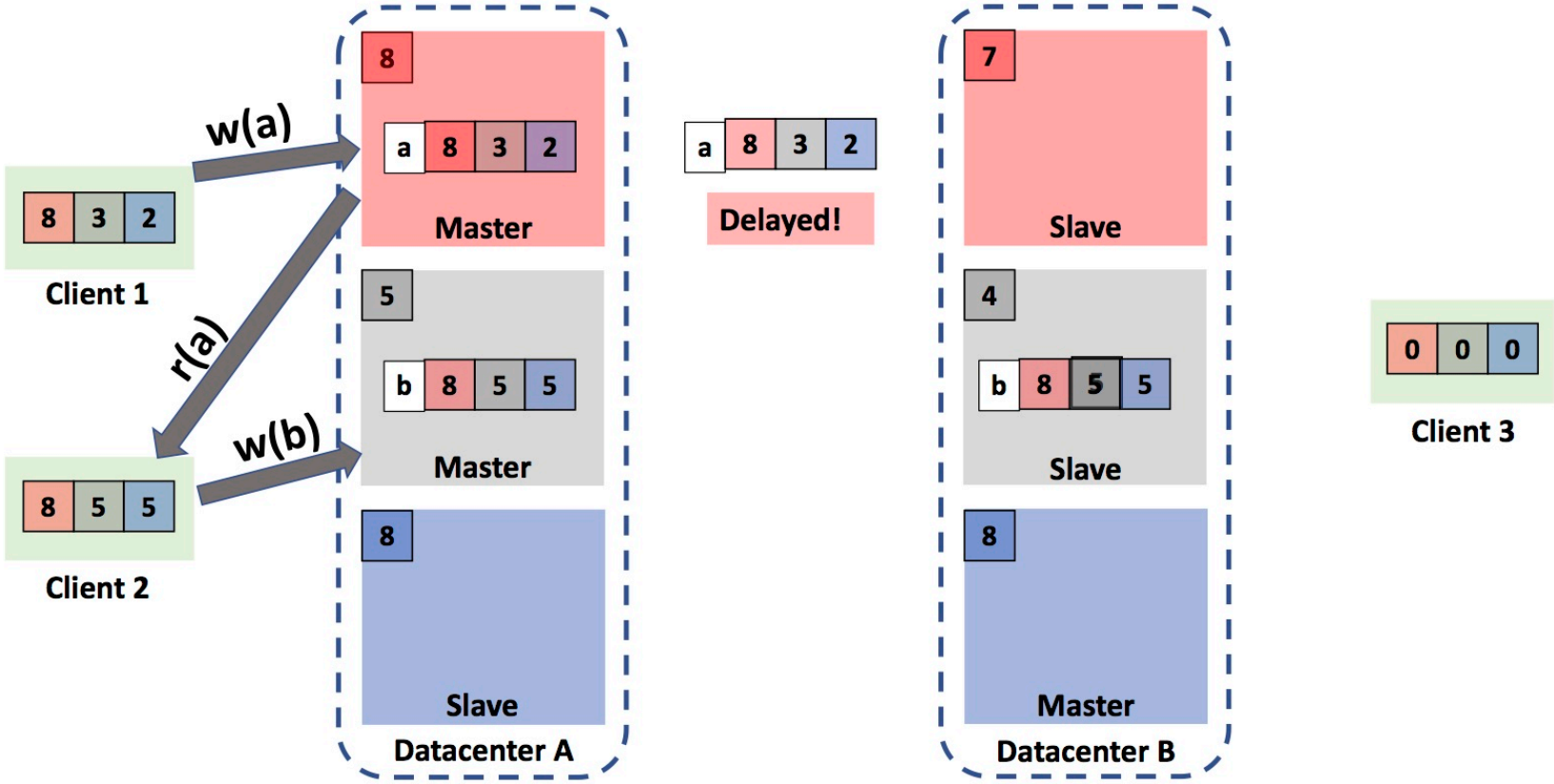
Push waiting out of store to client



Replication: Like eventual consistency; asynchronous, unordered, writes applied immediately

[Mehdi-NSDI17]

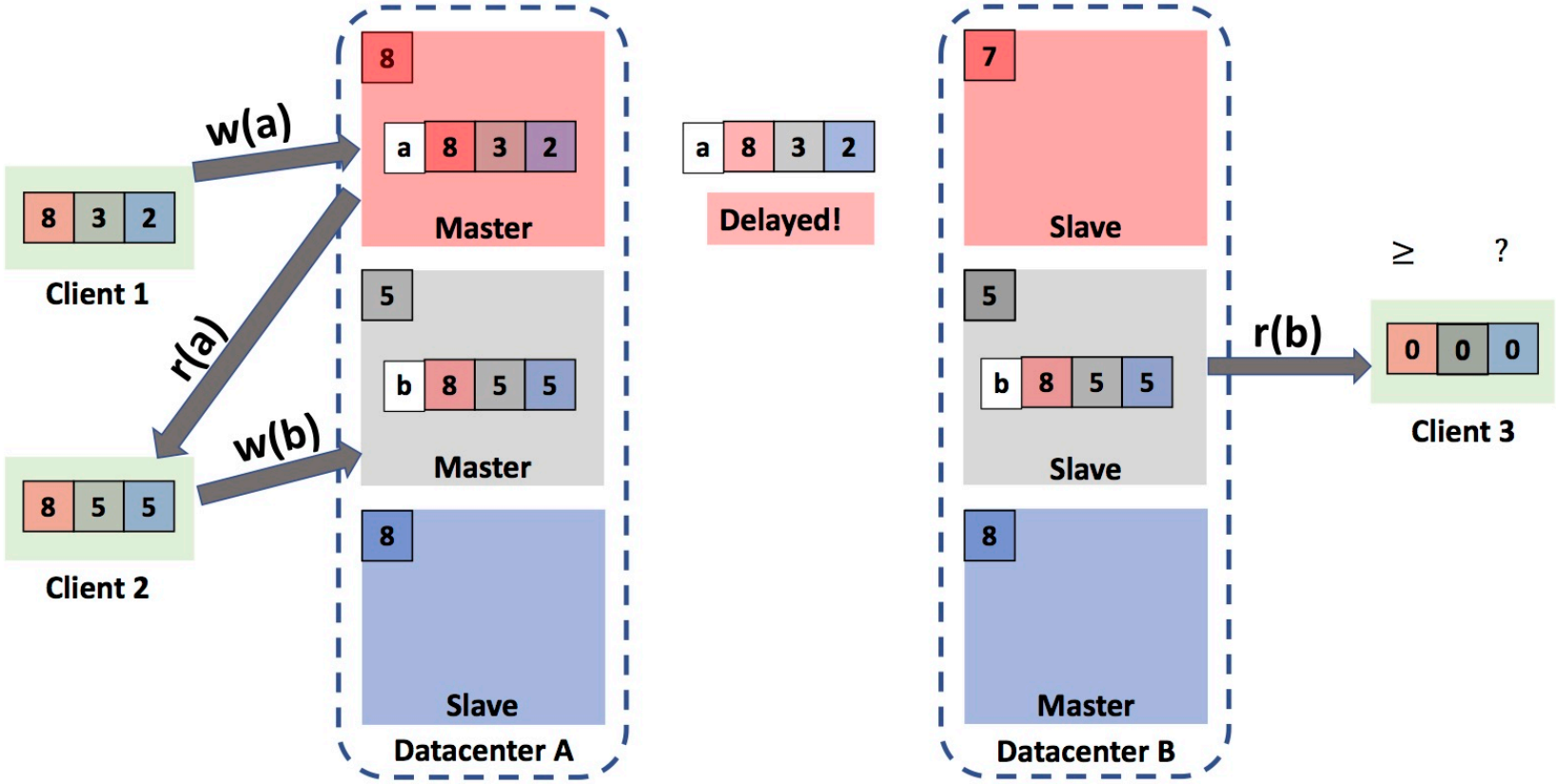
Push waiting out of store to client



Replication: Slaves increment their shardstamps using causal timestamp of a replicated write

[Mehdi-NSDI17]

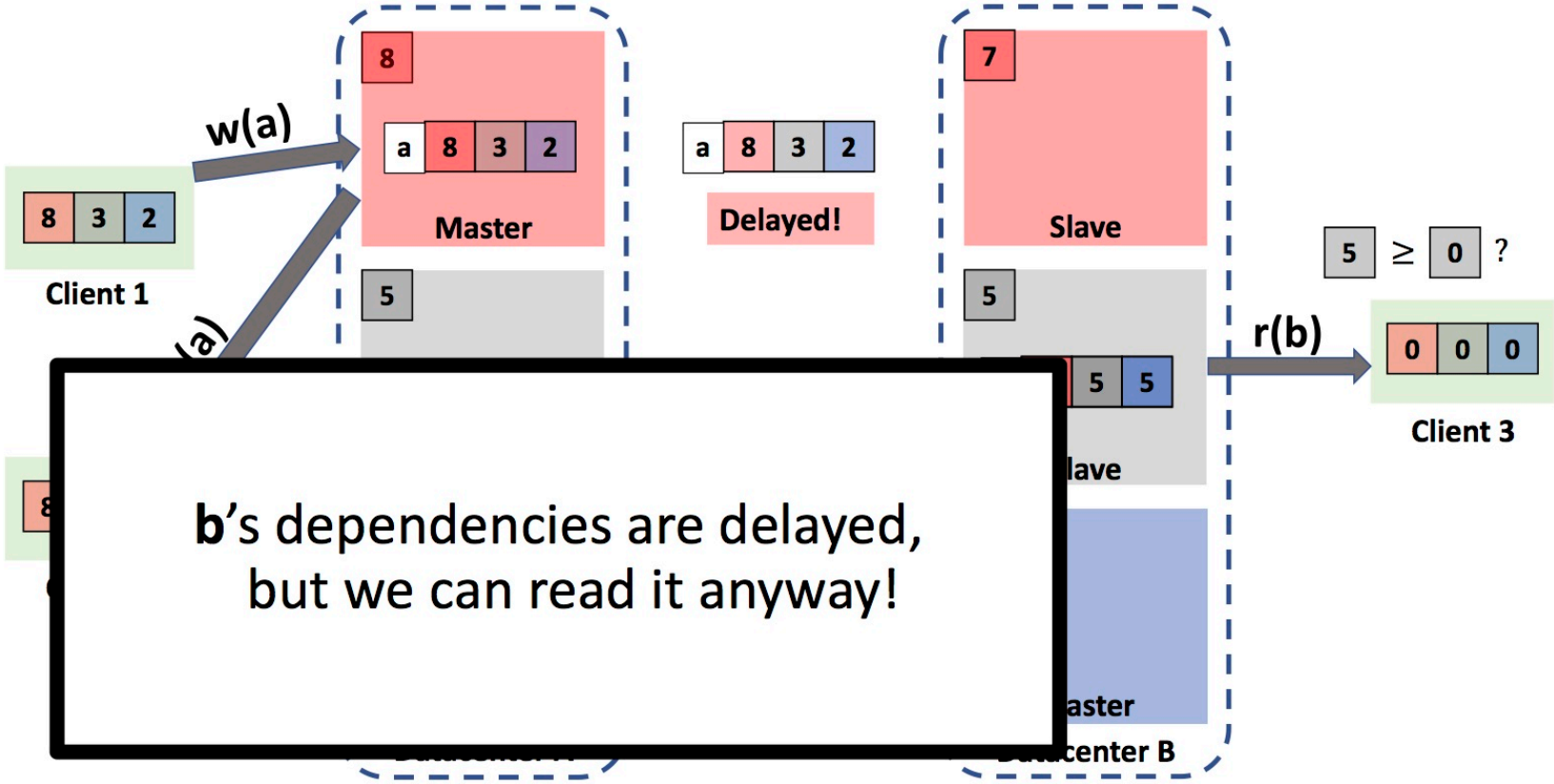
Push waiting out of store to client



Read Protocol: Clients do consistency check when reading from slaves

[Mehdi-NSDI17]

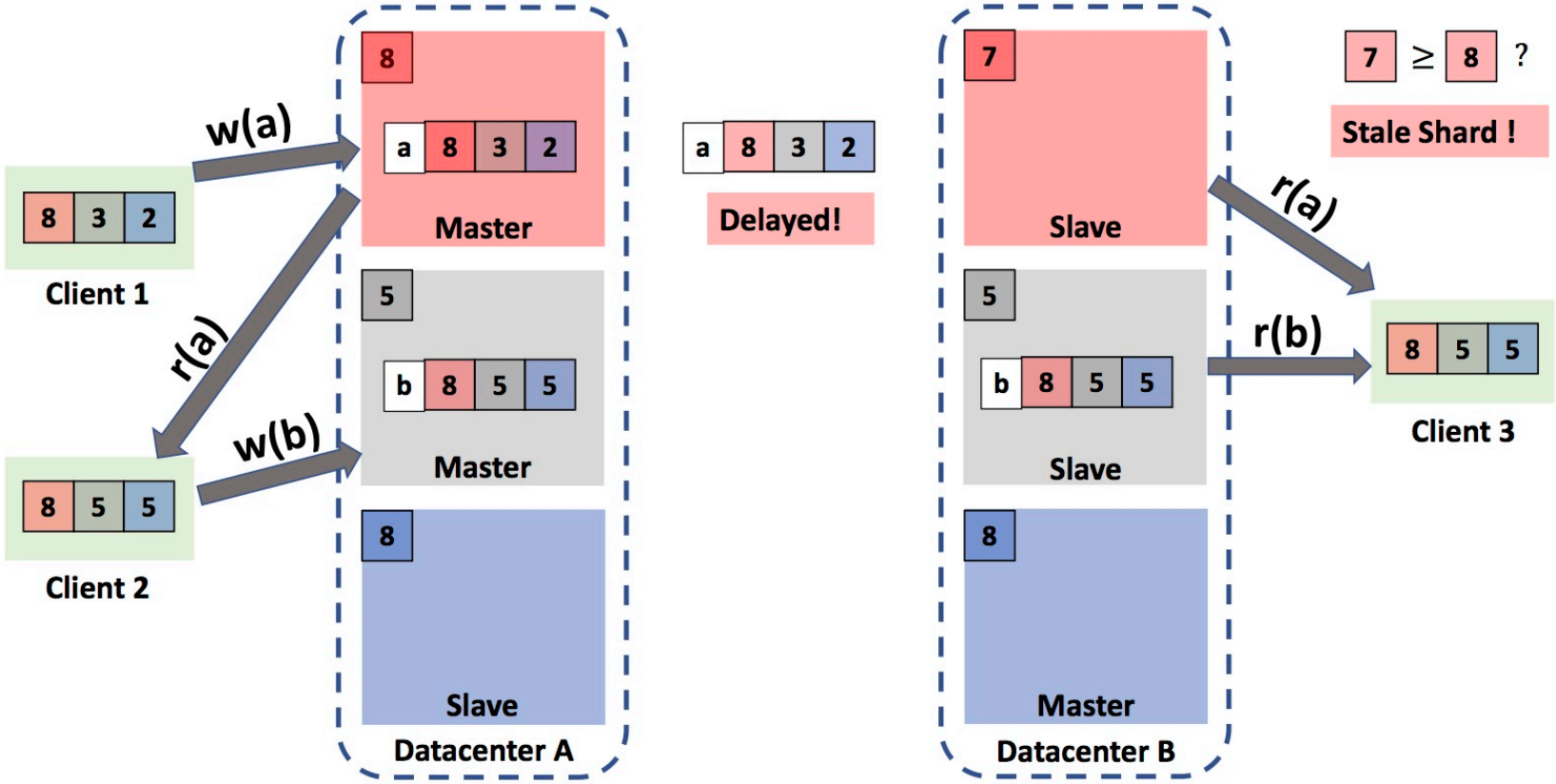
Push waiting out of store to client



Read Protocol: Clients do consistency check when reading from slaves

[Mehdi-NSDI17]

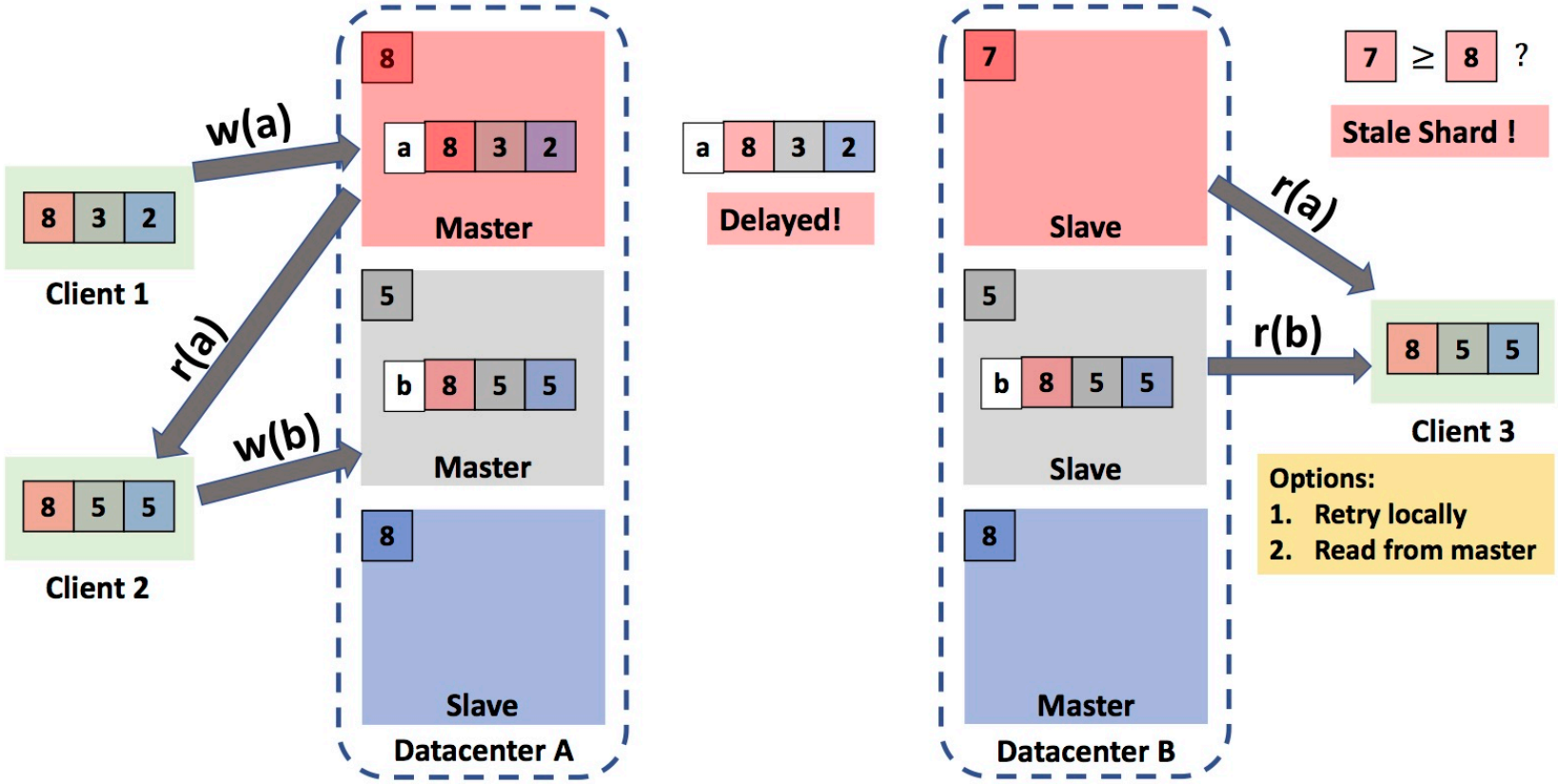
Push waiting out of store to client



Read Protocol: Clients do consistency check when reading from slaves

[Mehdi-NSDI17]

Push waiting out of store to client which might retry



Read Protocol: Resolving stale reads

[Mehdi-NSDI17]

And if Simple & Consistent outranks Availability?

- Storage abstraction offers "single system image" and you wait
- Google Spanner [OSDI'12] integrates logical clock versioning with global clock synchronization (TrueTime, an expansion on NTP)
  - So timestamp at any replica can be used to globally order all concurrency
    - Slows down decision making if clocks are poorly synch'd
  - Gives Linearizability (strongest consistency) to global transactions
    - Under partition, slow down could be arbitrary
- Google works hard to minimize partition events and duration
- TrueTime's integration of clock synch, consensus decisions, and distributed transactions provides better bounds, faster speed
  - At the cost of implementation complexity (integration of 3 complex codes)

Next days plan

- Monday: guest lecture on Cloud Adoption and Technology Trends
  - Mark Russinovich, Microsoft Azure
- Wednesday: security