



Big Data Infrastructure

CS 489/698 Big Data Infrastructure (Winter 2017)

Week 10: Mutable State (2/2)

March 16, 2017

Jimmy Lin

David R. Cheriton School of Computer Science
University of Waterloo

These slides are available at <http://lintool.github.io/bigdata-2017w/>



This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States
See <http://creativecommons.org/licenses/by-nc-sa/3.0/us/> for details

The Fundamental Problem

We want to keep track of *mutable* state in a *scalable* manner

Assumptions:

State organized in terms of logical records

State unlikely to fit on single machine, must be distributed

MapReduce won't do!

Motivating Scenarios

Money shouldn't be created or destroyed:

Alice transfers \$100 to Bob and \$50 to Carol

The total amount of money after the transfer should be the same

Phantom shopping cart:

Bob removes an item from his shopping cart...

Item still remains in the shopping cart

Bob refreshes the page a couple of times... item finally gone

Motivating Scenarios

People you don't want seeing your pictures:

Alice removes mom from list of people who can view photos

Alice posts embarrassing pictures from Spring Break

Can mom see Alice's photo?

Why am I still getting messages?

Bob unsubscribes from mailing list and receives confirmation

Message sent to mailing list right after unsubscribe

Does Bob receive the message?

Three Core Ideas

Why do these scenarios happen?

Partitioning (sharding)

To increase scalability and to decrease latency

Need distributed transactions!

Replication

To increase robustness (availability) and to increase throughput

Need replica coherence protocol!

Caching

To reduce latency

Need cache coherence protocol!



To: All Graduate Students

Due to a recent incident, we would like to remind all Grad Students that refreshments provided in communal areas during an event are for attendees of that event only.

Please vacate the communal area and do not consume the refreshments unless you have been specifically invited to participate.

To avoid any misunderstanding, you are only invited if you received a specific invitation by e-mail or if it was arranged by your supervisor for you to attend.

Thank you for your cooperation,

The Department Administrator

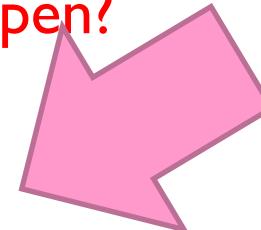


WWW.PHDCOMICS.COM

Morale of the story: there's no free lunch!
(Everything is a tradeoff)

Three Core Ideas

Why do these scenarios happen?



Partitioning (sharding)

To increase scalability and to decrease latency

Need distributed transactions!

Replication

To increase robustness (availability) and to increase throughput

Need replica coherence protocol!

Caching

To reduce latency

Need cache coherence protocol!

Relational Databases

... to the rescue!

How do RDBMSes do it?

Transactions on a single machine: (relatively) easy!

Partition tables to keep transactions on a single machine

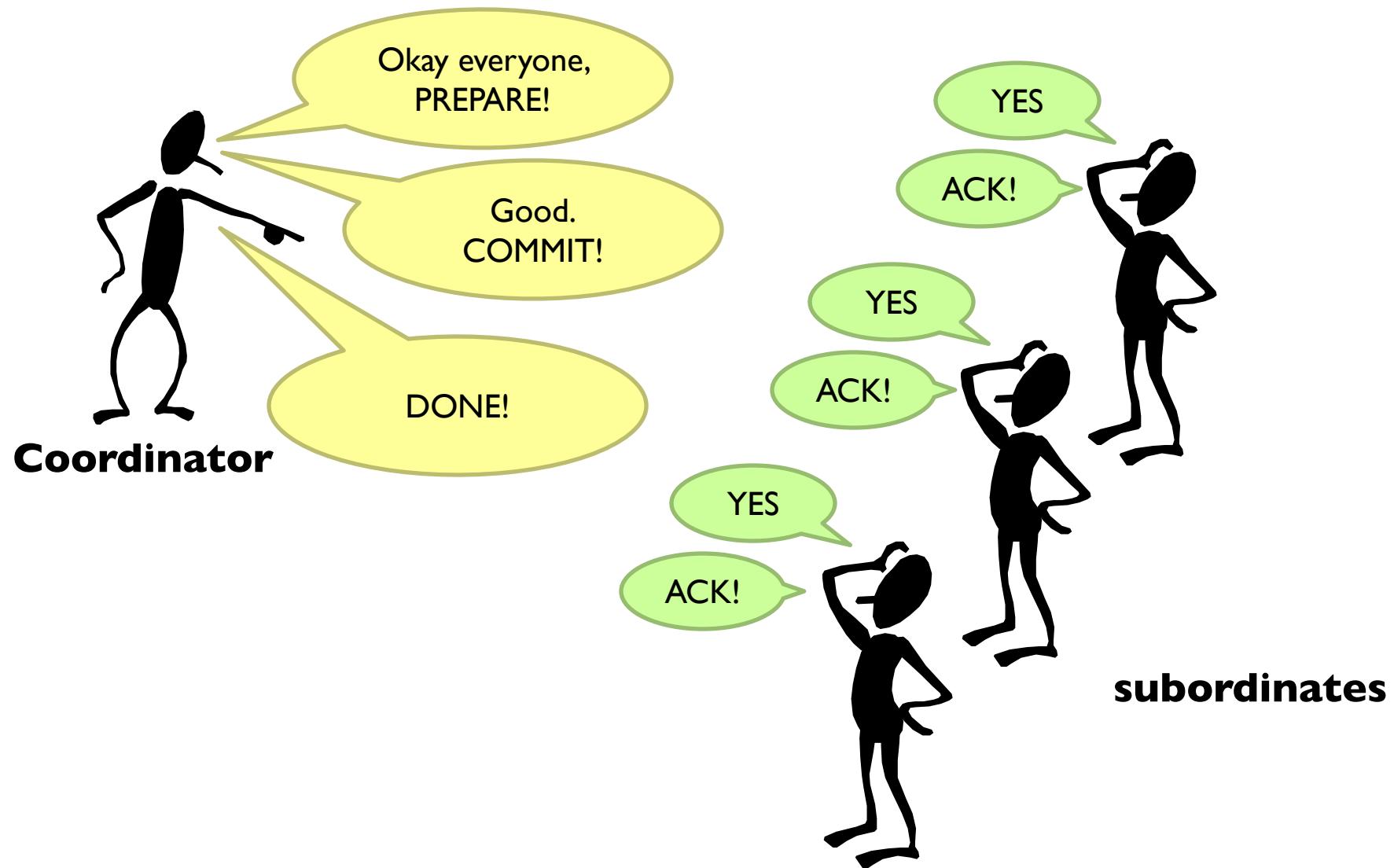
Example: partition by user

What about transactions that require multiple machines?

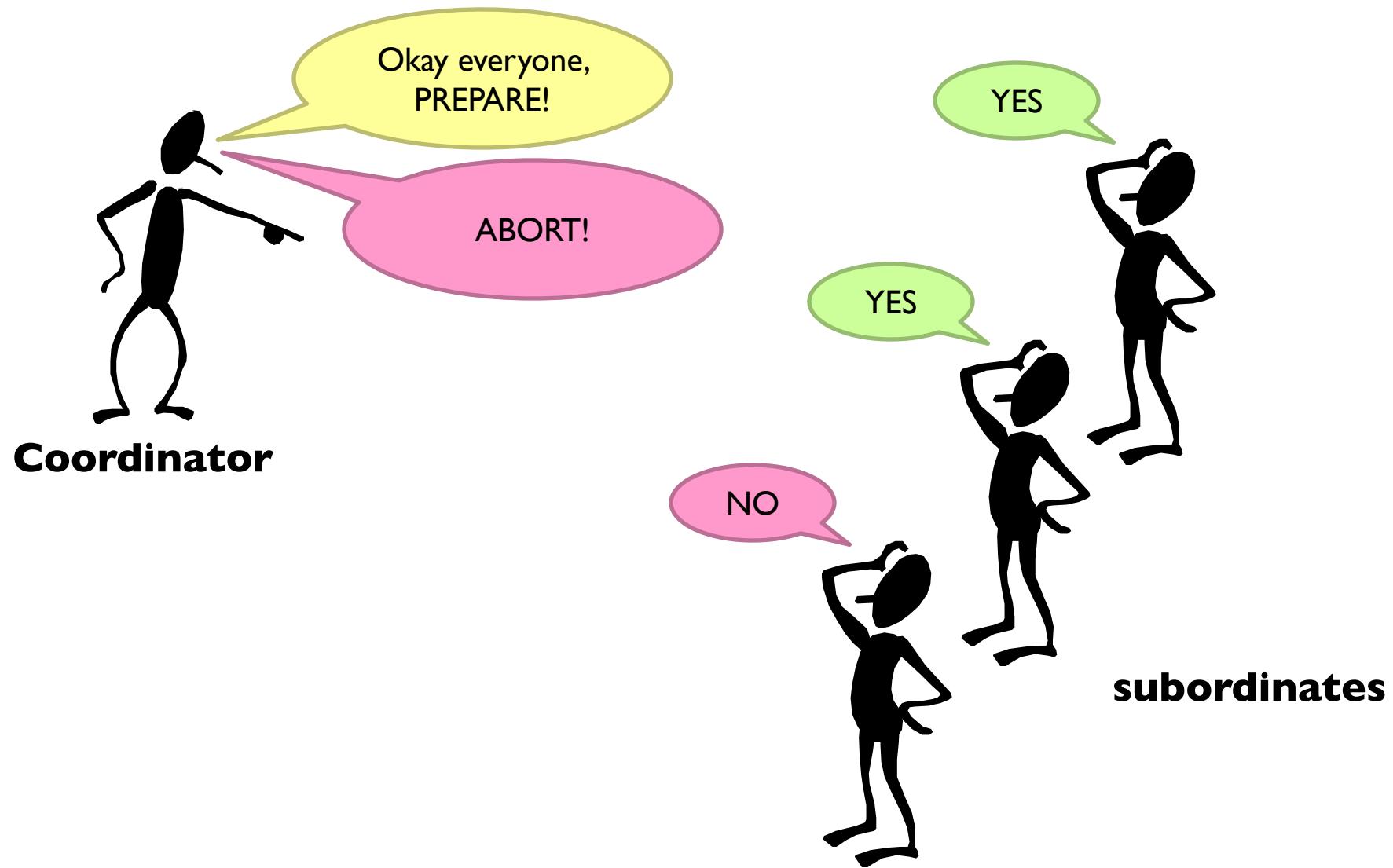
Example: transactions involving multiple users

Solution: Two-Phase Commit

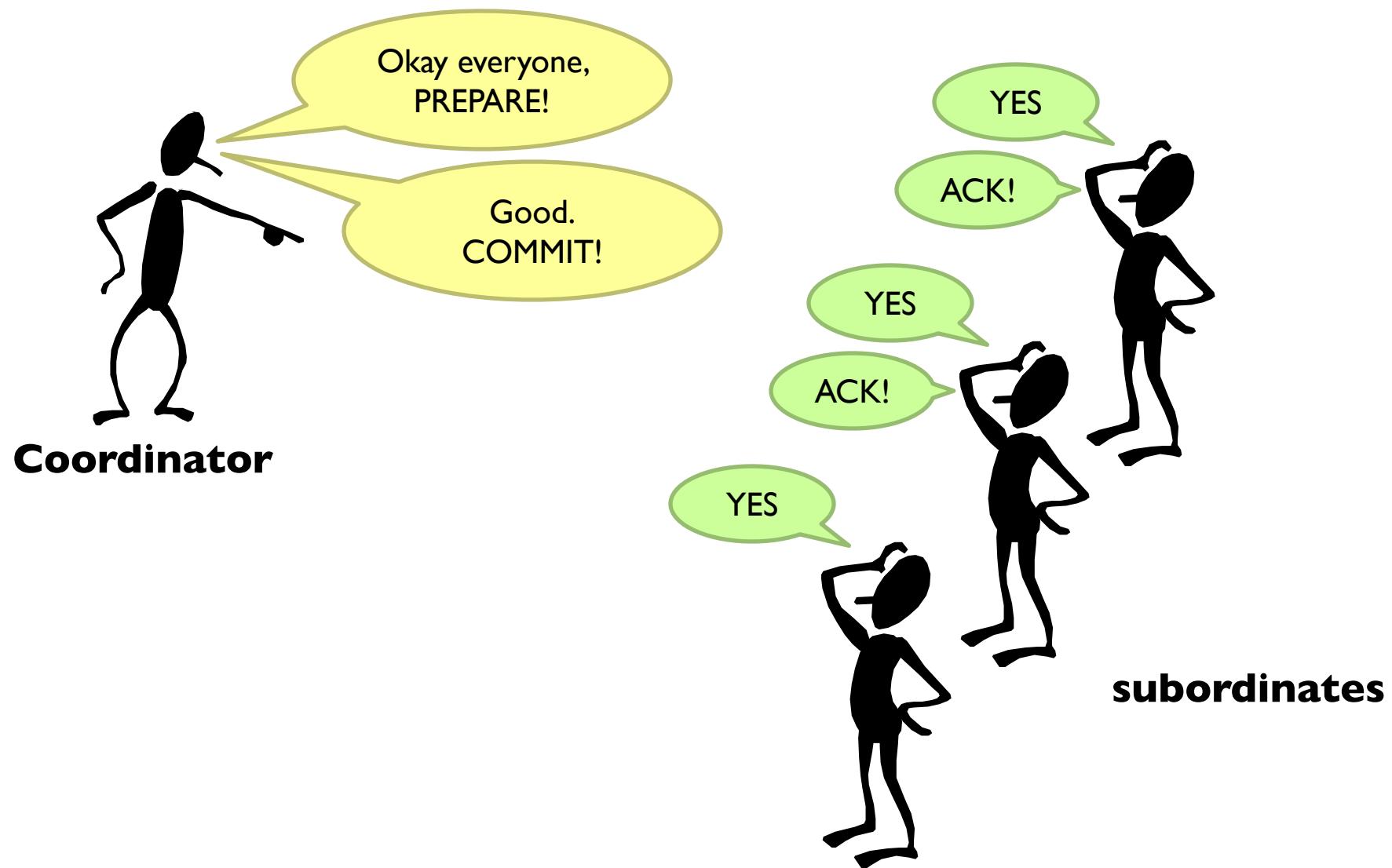
2PC: Sketch



2PC: Sketch



2PC: Sketch



2PC: Assumptions and Limitations

Assumptions:

Persistent storage and write-ahead log at every node
WAL is never permanently lost

Limitations:

It's blocking and slow
What if the coordinator dies?

Beyond 2PC: Paxos!
(details beyond scope of this course)

“Unit of Consistency”

Single record transactions:

Relatively straightforward

Complex application logic to handle multi-record transactions

Arbitrary transactions:

Requires 2PC or Paxos

Middle ground: entity groups

Groups of entities that share affinity

Co-locate entity groups

Provide transaction support within entity groups

Example: user + user's photos + user's posts etc.

Where have we learned this trick before?

Three Core Ideas

Why do these scenarios happen?

Partitioning (sharding)

To increase scalability and to decrease latency

Need distributed transactions!

Replication

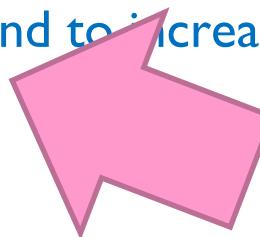
To increase robustness (availability) and to increase throughput

Need replica coherence protocol!

Caching

To reduce latency

Need cache coherence protocol!



CAP “Theorem”

(Brewer, 2000)

Consistency

Availability

Partition tolerance

... pick two

CAP Tradeoffs

CA = consistency + availability

E.g., parallel databases that use 2PC

AP = availability + tolerance to partitions

E.g., DNS, web caching

Is this helpful?

CAP not really even a “theorem” because vague definitions
More precise formulation came a few years later



Abadi Says...

CP makes no sense!

CAP says, in the presence of P, choose A or C

But you'd want to make this tradeoff even when there is no P

Fundamental tradeoff is between consistency and latency

Not available = (very) long latency

Replication possibilities

Update sent to all replicas at the same time
To guarantee consistency you need something like Paxos

Update sent to a master
Replication is synchronous
Replication is asynchronous
Combination of both

Okay, but if the master fails?

Update sent to an arbitrary replica
Okay, now what?

All these possibilities involve tradeoffs!
“eventual consistency”

Move over, CAP

PACELC (“pass-elk”)

PAC

If there's a partition, do we choose A or C?

ELC

Otherwise, do we choose Latency or Consistency?

Eventual Consistency

Sounds reasonable in theory...
What about in practice?

It really depends on the application!

To: All Graduate Students

Due to a recent incident, we would like to remind all Grad Students that refreshments provided in communal areas during an event are for attendees of that event only.

Please vacate the communal area and do not consume the refreshments unless you have been specifically invited to participate.

To avoid any misunderstanding, you are only invited if you received a specific invitation by e-mail or if it was arranged by your supervisor for you to attend.

Thank you for your cooperation,

The Department Administrator



WWW.PHDCOMICS.COM

Morale of the story: there's no free lunch!
(Everything is a tradeoff)

Three Core Ideas

Why do these scenarios happen?

Partitioning (sharding)

To increase scalability and to decrease latency

Need distributed transactions!

Replication

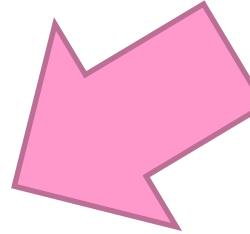
To increase robustness (availability) and to increase throughput

Need replica coherence protocol!

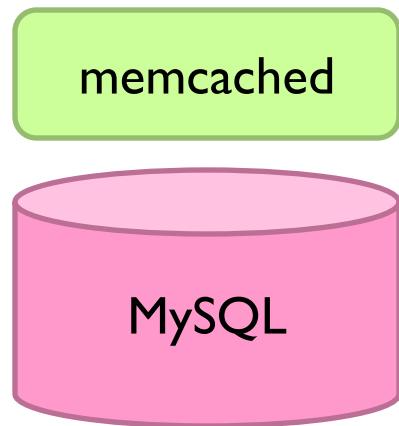
Caching

To reduce latency

Need cache coherence protocol!



Facebook Architecture



Read path:

- Look in memcached
- Look in MySQL
- Populate in memcached

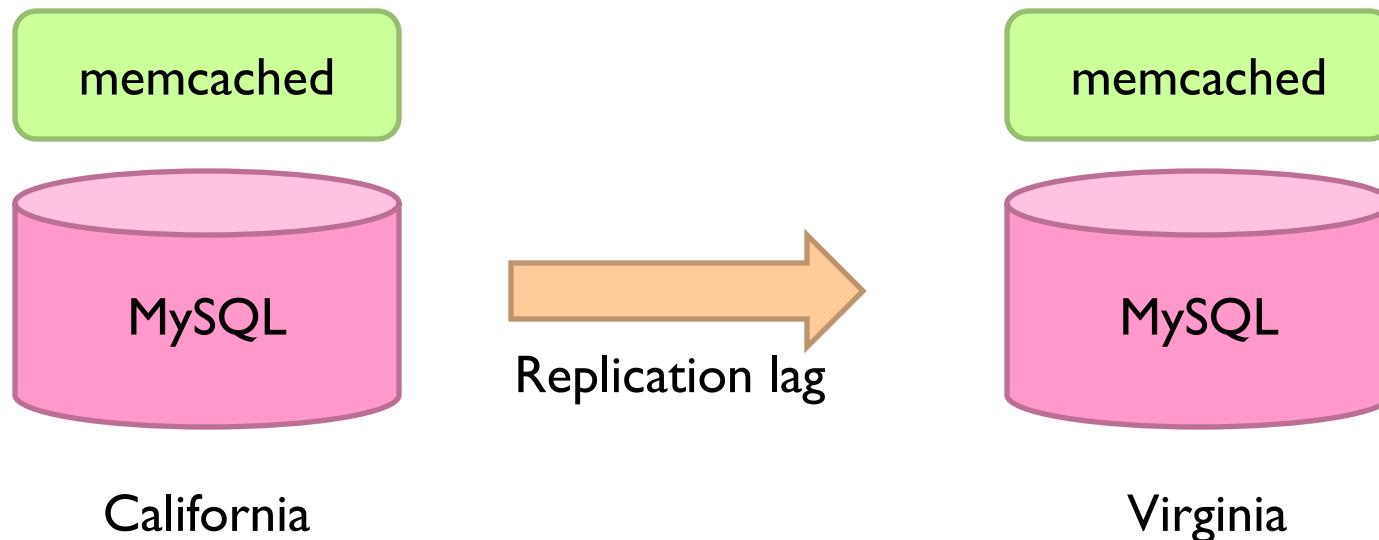
Write path:

- Write in MySQL
- Remove in memcached

Subsequent read:

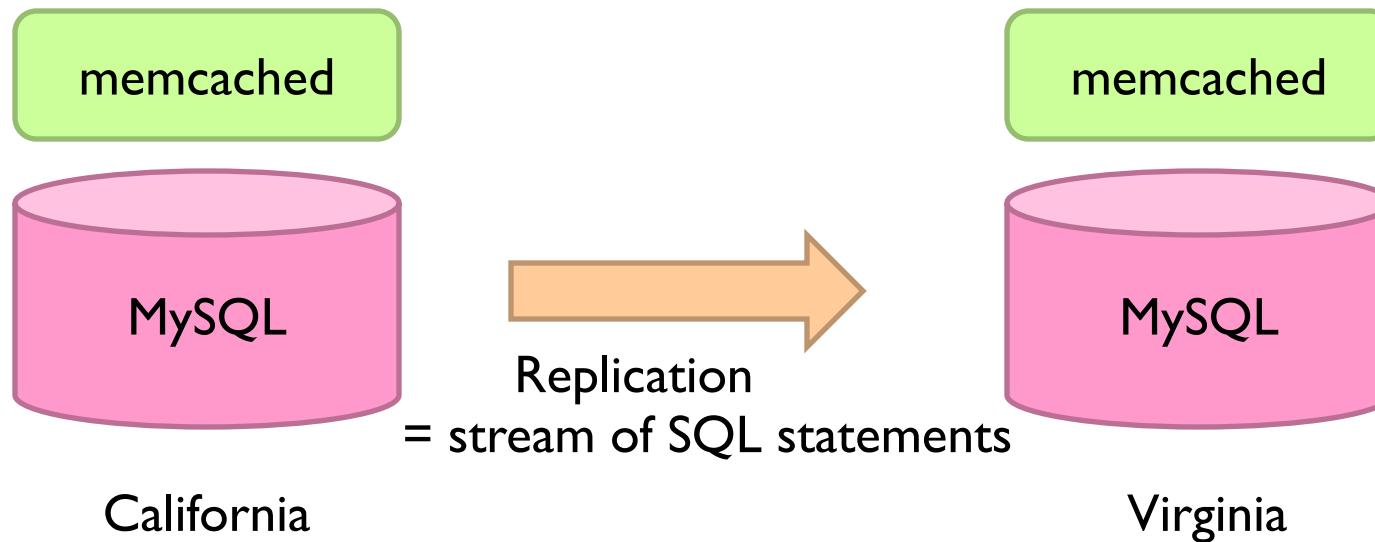
- Look in MySQL
- Populate in memcached

Facebook Architecture: Multi-DC



1. User updates first name from “Jason” to “Monkey”.
2. Write “Monkey” in master DB in CA, delete memcached entry in CA and VA.
3. Someone goes to profile in Virginia, read VA replica DB, get “Jason”.
4. Update VA memcache with first name as “Jason”.
5. Replication catches up. “Jason” stuck in memcached until another write!

Facebook Architecture: Multi-DC



Solution: Piggyback on replication stream, tweak SQL

```
REPLACE INTO profile (`first_name`) VALUES ('Monkey')
WHERE `user_id`='jsobel' MEMCACHE_DIRTY 'jsobel:first_name'
```

Three Core Ideas

Why do these scenarios happen?

Partitioning (sharding)

To increase scalability and to decrease latency

Need distributed transactions!

Replication

To increase robustness (availability) and to increase throughput

Need replica coherence protocol!

Caching

To reduce latency

Need cache coherence protocol!

An aerial photograph of a large datacenter facility during sunset. The sky is a warm orange and yellow. In the foreground, there's a complex of several buildings, including a large white building with a flat roof and a smaller green building with a green roof. There are also several white cylindrical storage tanks in a row. A road or highway runs through the middle ground, with some traffic visible. The background shows a vast, green, agricultural landscape stretching to the horizon under a hazy sky.

Now imagine multiple datacenters...
What's different?

Yahoo's PNUTS

Yahoo's globally distributed/replicated key-value store

Provides *per-record* timeline consistency

Guarantees that all replicas provide all updates in same order

Different classes of reads:

Read-any: may time travel!

Read-critical(required version): monotonic reads

Read-latest

PNUTS: Implementation Principles

Each record has a single master

Asynchronous replication across datacenters

Allow for synchronous replication within datacenters

All updates routed to master first, updates applied, then propagated

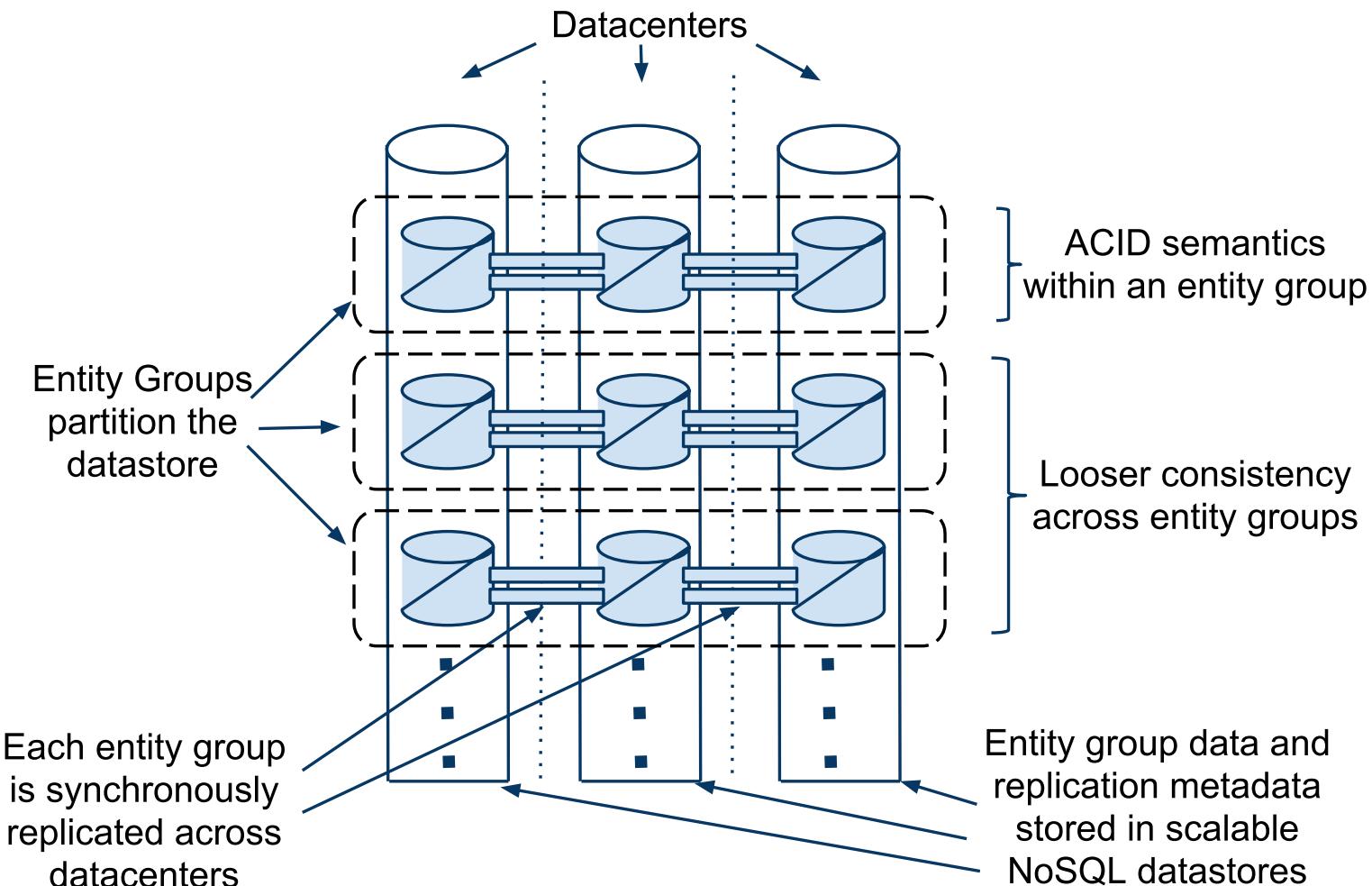
Protocols for recognizing master failure and load balancing

Tradeoffs:

Different types of reads have different latencies

Availability compromised during simultaneous master and partition failure

Google's Megastore



Google's Spanner

Features:

Full ACID translations across multiple datacenters, across continents!

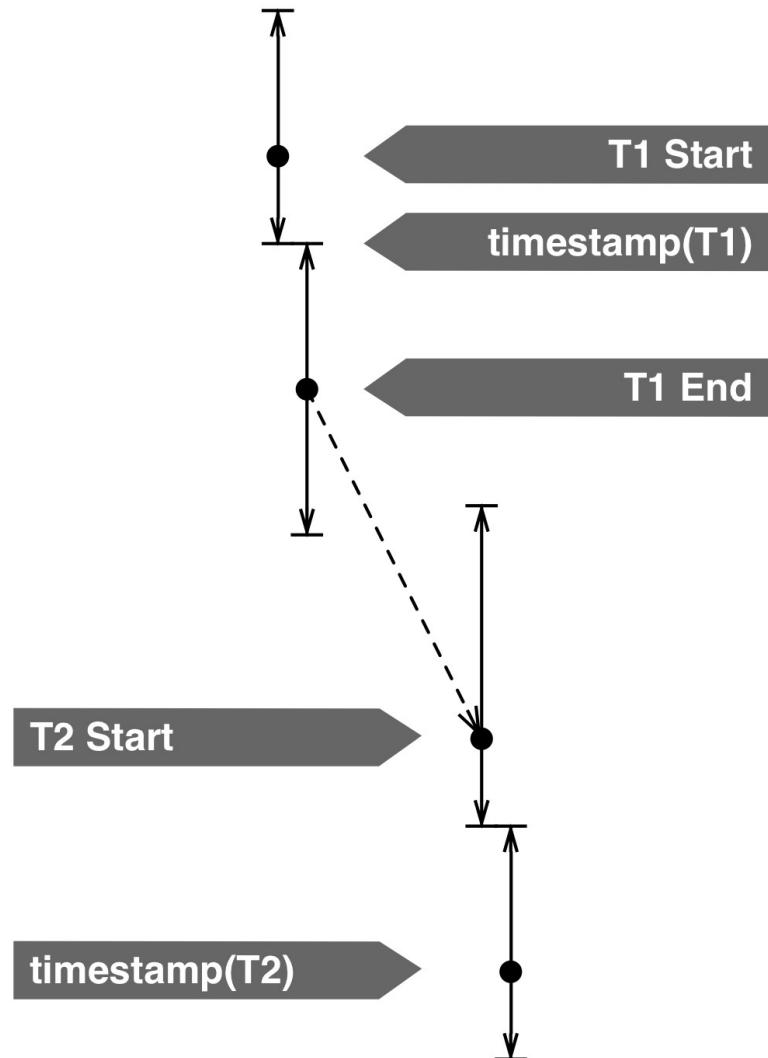
External consistency (= linearizability):

system preserves *happens-before* relationship among transactions

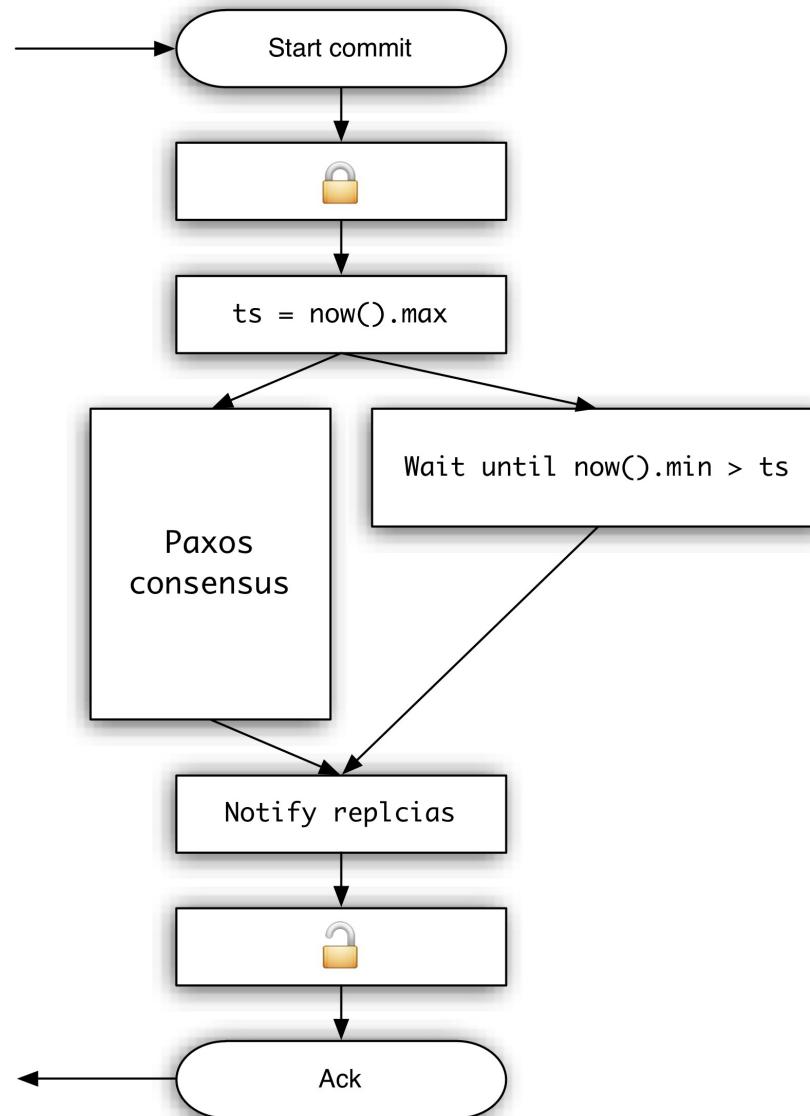
How?

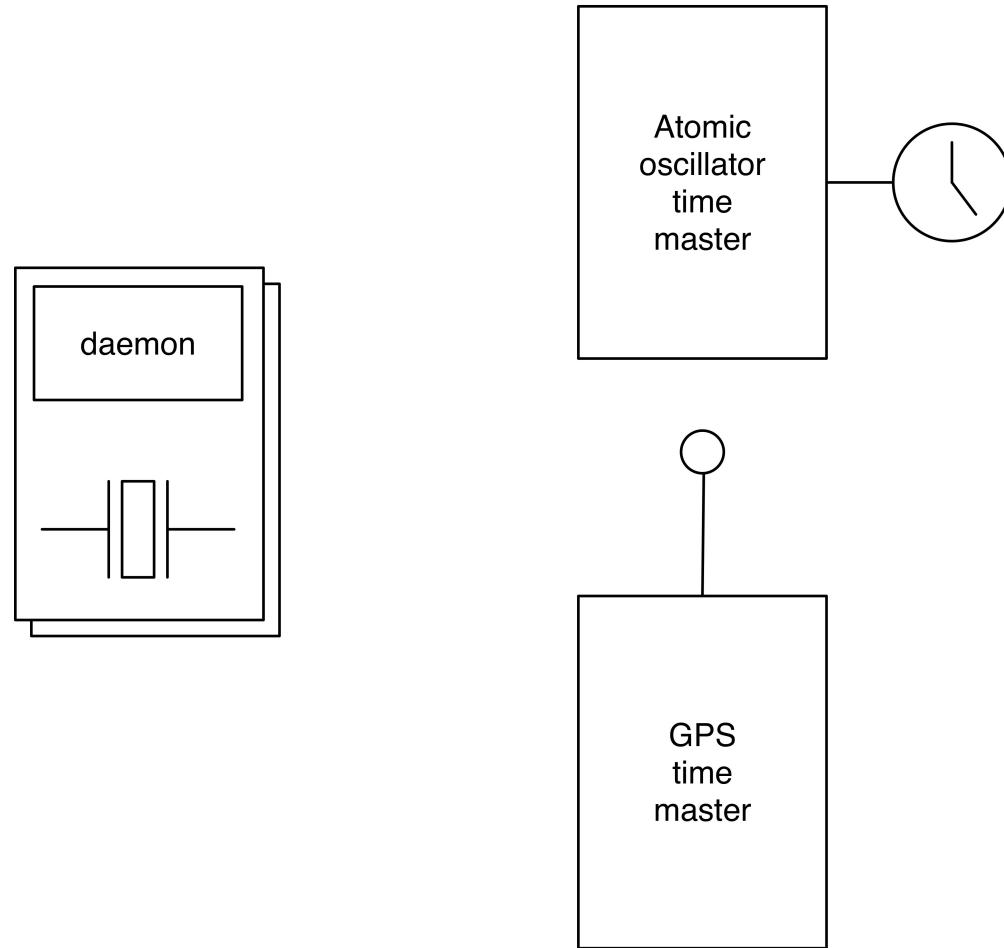
Given write transactions A and B, if A *happens-before* B, then
 $\text{timestamp}(A) < \text{timestamp}(B)$

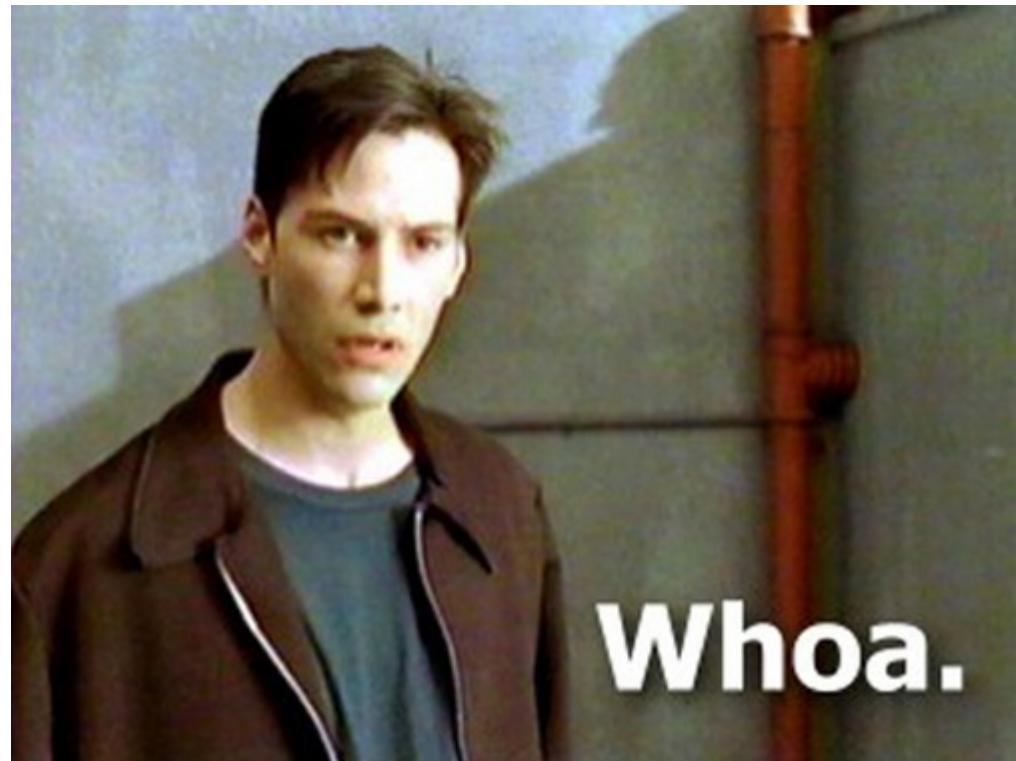
Why this works



TrueTime → write timestamps







What's the catch?

Three Core Ideas

Partitioning (sharding)

To increase scalability and to decrease latency

Need distributed transactions!

Replication

To increase robustness (availability) and to increase throughput

Need replica coherence protocol!

Caching

To reduce latency

Need cache coherence protocol!



To: All Graduate Students

Due to a recent incident, we would like to remind all Grad Students that refreshments provided in communal areas during an event are for attendees of that event only.

Please vacate the communal area and do not consume the refreshments unless you have been specifically invited to participate.

To avoid any misunderstanding, you are only invited if you received a specific invitation by e-mail or if it was arranged by your supervisor for you to attend.

Thank you for your cooperation,

The Department Administrator



WWW.PHDCOMICS.COM

Morale of the story: there's no free lunch!
(Everything is a tradeoff)

A photograph of a traditional Japanese rock garden. In the foreground, a gravel path is raked into fine, parallel lines. Several large, dark, irregular stones are scattered across the garden. A small, shallow pond is visible in the middle ground, surrounded by more stones and some low-lying green plants. In the background, there are more stones, some small trees, and the wooden buildings of a residence with tiled roofs.

Questions?