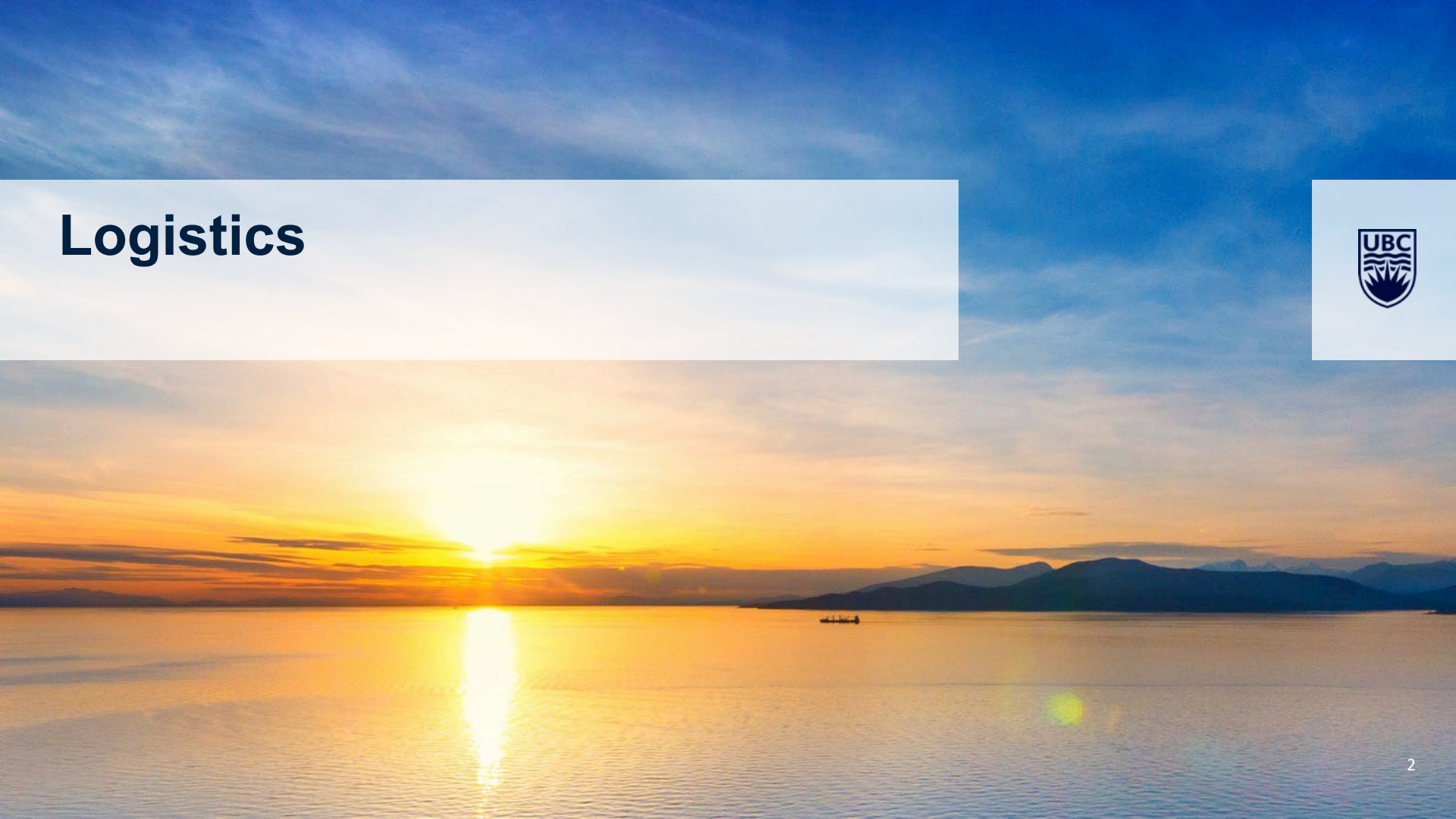
CPSC 416 Distributed Systems

Winter 2022 Term 2 (February 2, 2023)

Tony Mason (fsgeek@cs.ubc.ca), Lecturer

Logistics



Deadlines

Project 2 Report: Pending Review

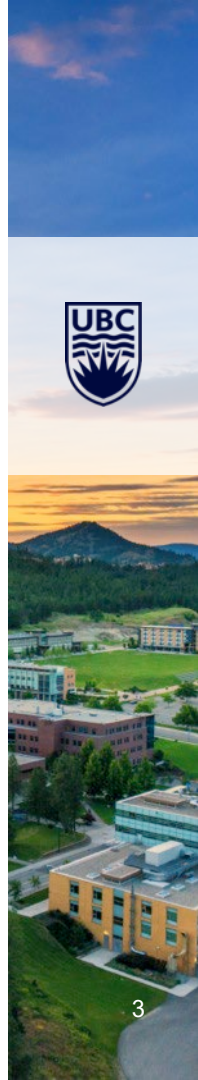
Project 3 Released. Initially Due: February 13, 2023.

- Note repo was updated February 1. (Minor README updates)

Project 4 Released. Initially Due: March 13, 2023

Project 5 Released Due: April 13, 2023

All project work is due April 13, 2023. Late projects have a 75% score cap.



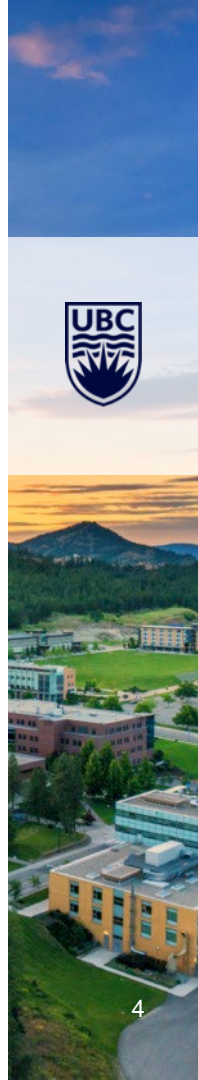
Deadlines

Alternate Path 1 & 2: Proposal was due January 30, 2023.

- Review in progress
- Piazza private threads created
  - Review
  - Respond to outstanding questions
- **Proceed according to your plan.**

Instructor Office Hours:

- Zoom Office Hours (Tuesday) @ 13:00-14:00
- Discord (Casual) Office Hours (Thursday) @ 14:00-15:00

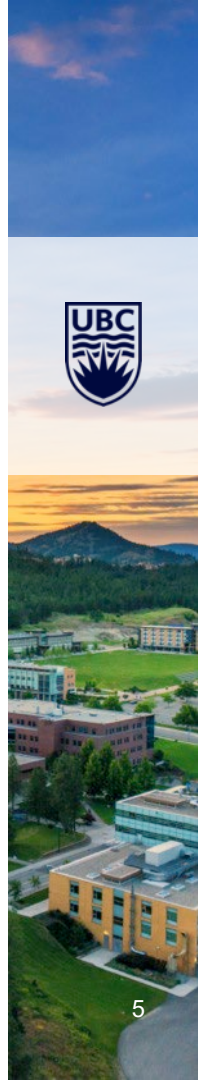


Readings

Required:

Recommended:

- Spanner: Google's globally distributed database
- A low-cost atomic commit protocol
- BigTable: A distributed storage system for structured data
- Megastore: Providing scalable, highly available storage for interactive services
- The part-time parliament
- Viewstamped replication: A new primary copy method to support highly-available distributed systems.
- From Viewstamped Replication to Byzantine Fault Tolerance

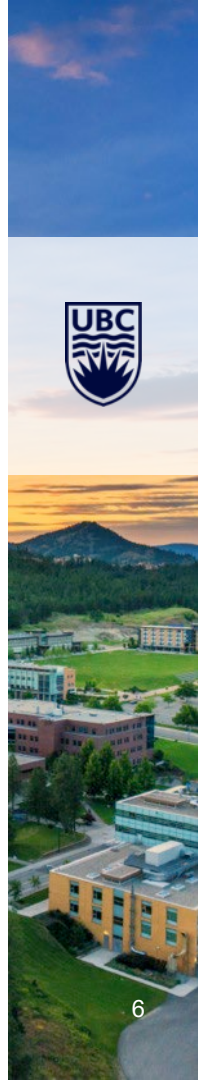


Questions?

Questions about the class?

Questions about the previous lecture?

Funny stories to share?



Today’s Failure

Amazon Web Services (S3)

February 28, 2017 09:37 PT

AWS S3 is an “object store” service

- Provides a key/value web interface
  - PUT
  - GET
  - DELETE

Used for *many* services including:

- Netflix
- Reddit
- Pinterist

AWS S3 Outage

“The Amazon Simple Storage Service (S3) team was debugging an issue causing the S3 billing system to progress more slowly than expected. At 9:37AM PST, an authorized S3 team member using an established playbook executed a command which was intended to remove a small number of servers for one of the S3 subsystems that is used by the S3 billing process. Unfortunately, one of the inputs to the command was entered incorrectly and a larger set of servers was removed than intended.”

Oops…

AWS S3 Outage

“The servers that were inadvertently removed supported two other S3 subsystems. One of these subsystems, the index subsystem, manages the metadata and location information of all S3 objects in the region. This subsystem is necessary to serve all GET, LIST, PUT, and DELETE requests. The second subsystem, the placement subsystem, manages allocation of new storage and requires the index subsystem to be functioning properly to correctly operate. The placement subsystem is used during PUT requests to allocate storage for new objects. Removing a significant portion of the capacity caused each of these systems to require a full restart. While these subsystems were being restarted, S3 was unable to service requests. Other AWS services in the US-EAST-1 Region that rely on S3 for storage, including the S3 console, Amazon Elastic Compute Cloud (EC2) new instance launches, Amazon Elastic Block Store (EBS) volumes (when data was needed from a S3 snapshot), and AWS Lambda were also impacted while the S3 APIs were unavailable. “

Big impact

AWS S3 Outage

Service fully restored 13:54 PT.

Total disruption: 4:14!

Well, sort of… (“some … services had accumulated a backlog of work during the S3 disruption and required additional time to fully recover.”)

“From the beginning of this event until 11:37AM PST, we were unable to update the individual services’ status on the AWS Service Health Dashboard (SHD) because of a dependency the SHD administration console has on Amazon S3. Instead, we used the AWS Twitter feed (@AWSCloud) and SHD banner text to communicate status until we were able to update the individual services’ status on the SHD. We understand that the SHD provides important visibility to our customers during operational events and we have changed the SHD administration console to run across multiple AWS regions.”

Their own notification service was unavailable!

AWS S3 Outage

Lessons:

- Manual interventions are *high risk*
- Things do not work as expected
- Failure of a system can lead to unintended secondary failures
- It is *essential* to do root-cause analysis, identify problems, *fix the problems*
  - Note there is a **social** aspect here (“failure is expected, learn from it, fix it, grow”)
  - **Create a culture of review and improvement, not blame**

Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region

Lesson Goals

Distributed Transactions

Consistency

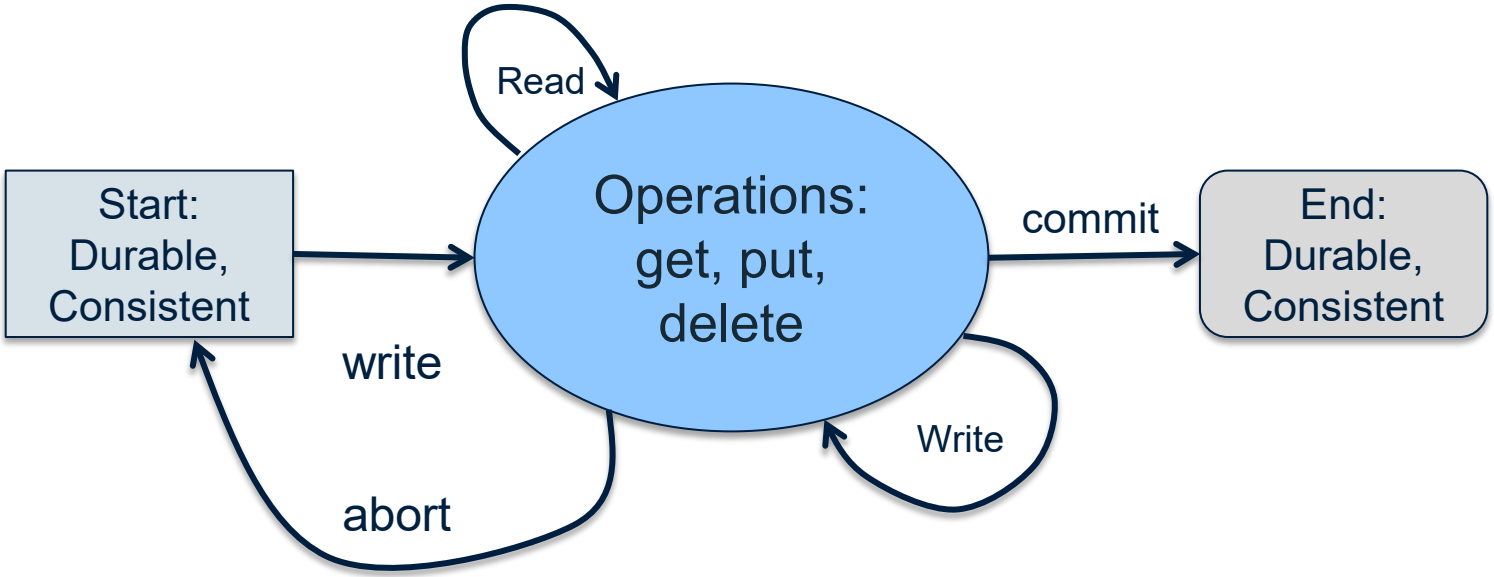
Distributed Transaction Implementation Approaches

- 2-phase commit
- 2-phase locking
- Truetime
- Relaxed consistency (“eventual”)

Examples

- Google Spanner
- AWS Aurora
- CockroachDB

Transaction

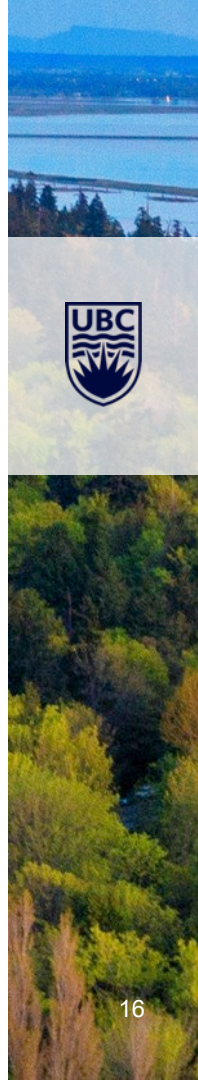


Transactions

Transactions provide:

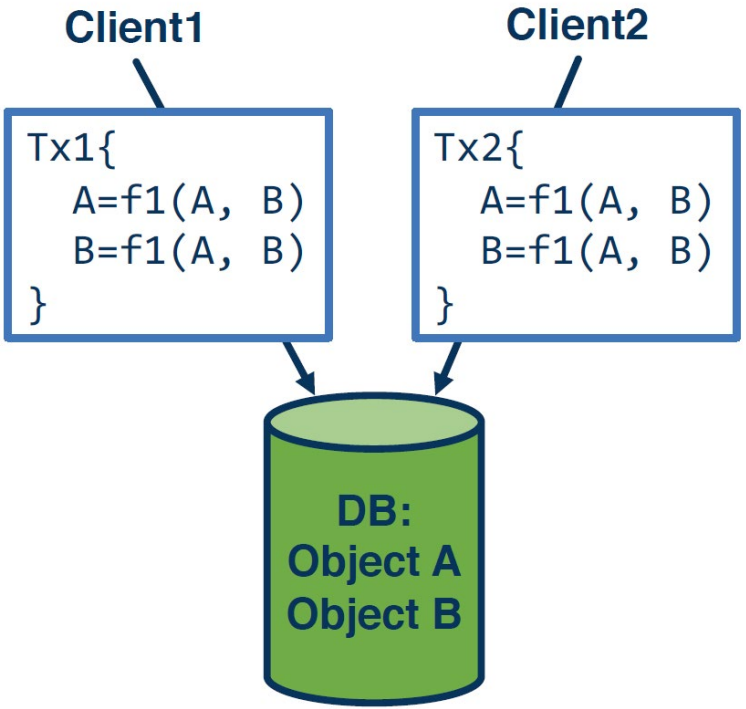
- Atomicity of *multiple* distinct operations
- Consistent state (beginning/end)
- Isolation (intermediate states *are not visible*)
- Durable (outcome is preserved)

Note: in the “real world” we often explore different ways of realizing these



Distributed Transactions

Transactions implemented *across* systems



Distributed Transaction

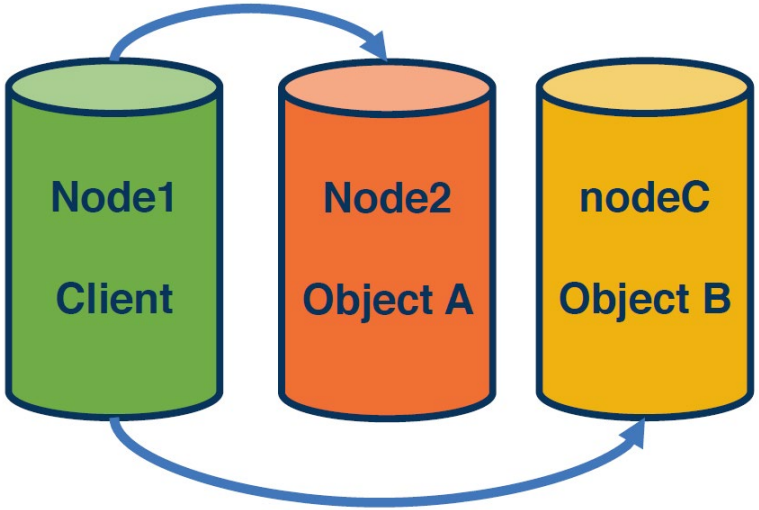
Coordinator + Multiple nodes

Nodes – store *state*

- Typically mutable
- Participates in some transactions

Coordinator

- Leader
- Determines outcome
- Involved in one or more transaction



Spanner

Google’s relational database service

- Internal Google Services
- External Cloud DB services
- Applications

Spanner

Global data store

- Geographically distributed
- Sharded
- Replicated

Google's relational database service

- Internal services
- External Cloud DB
- Applications

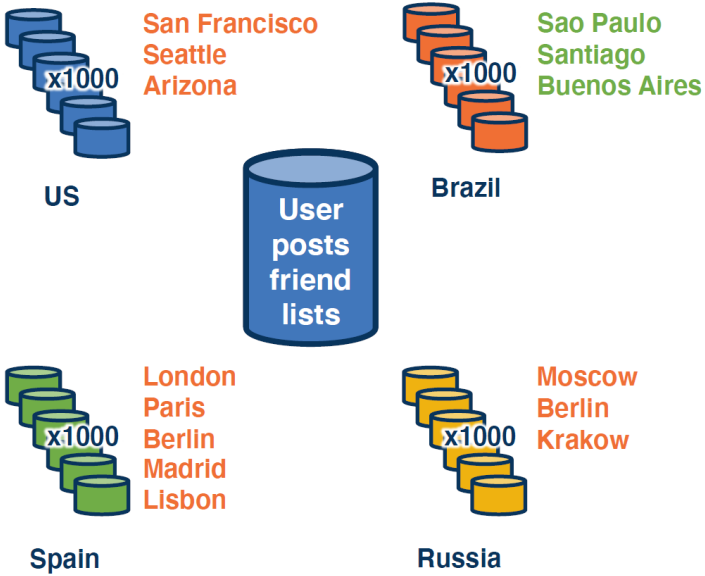


Figure Adapted from Spanner presentation at OSDI'12

Spanner: Technology Stack

Distributed persistent storage

- Google Filesystem
- Colossus

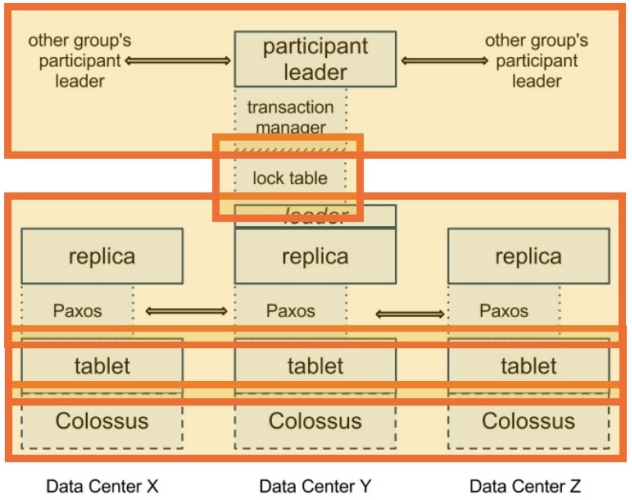
Data model

- Versioned key-value store (BigTable)
- Tablet, directory (related objects)
- Relational database (Megastore)

Replicated state machine (per tablet) using Paxos

2-phase locking (concurrency control)

2-phase commit (cross-replica set transactions)



Spanserver software stack

Single Machine Consistency

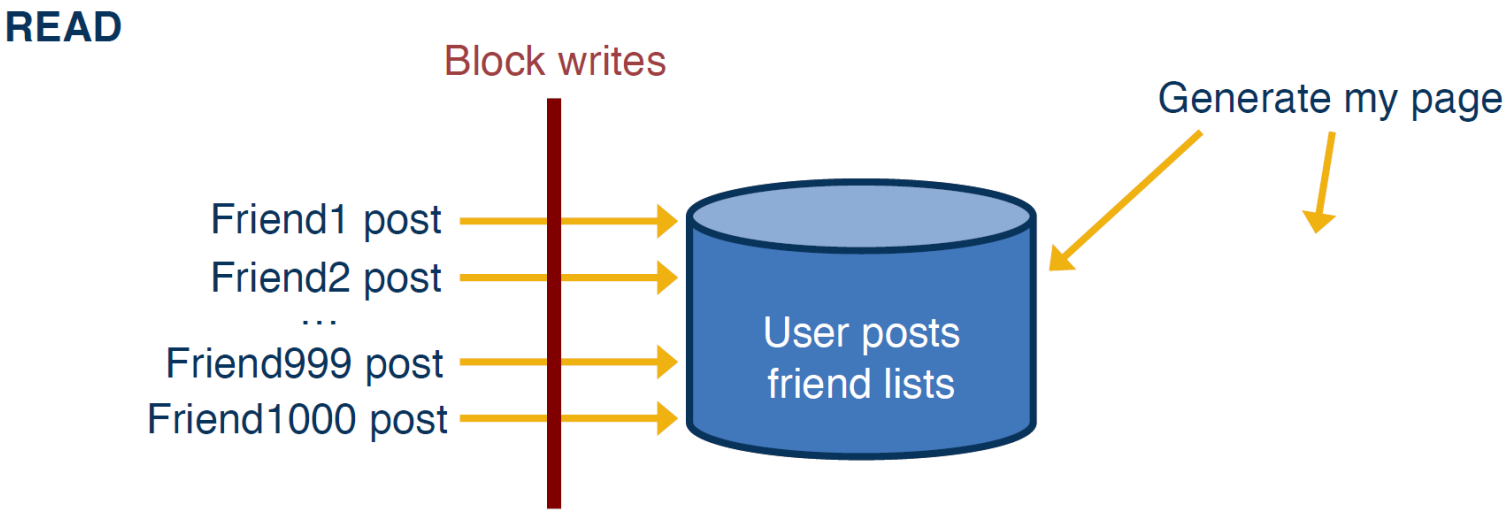


Figure Adapted from Spanner presentation at OSDI'12

Multi-machine Multi-datacenter Consistency

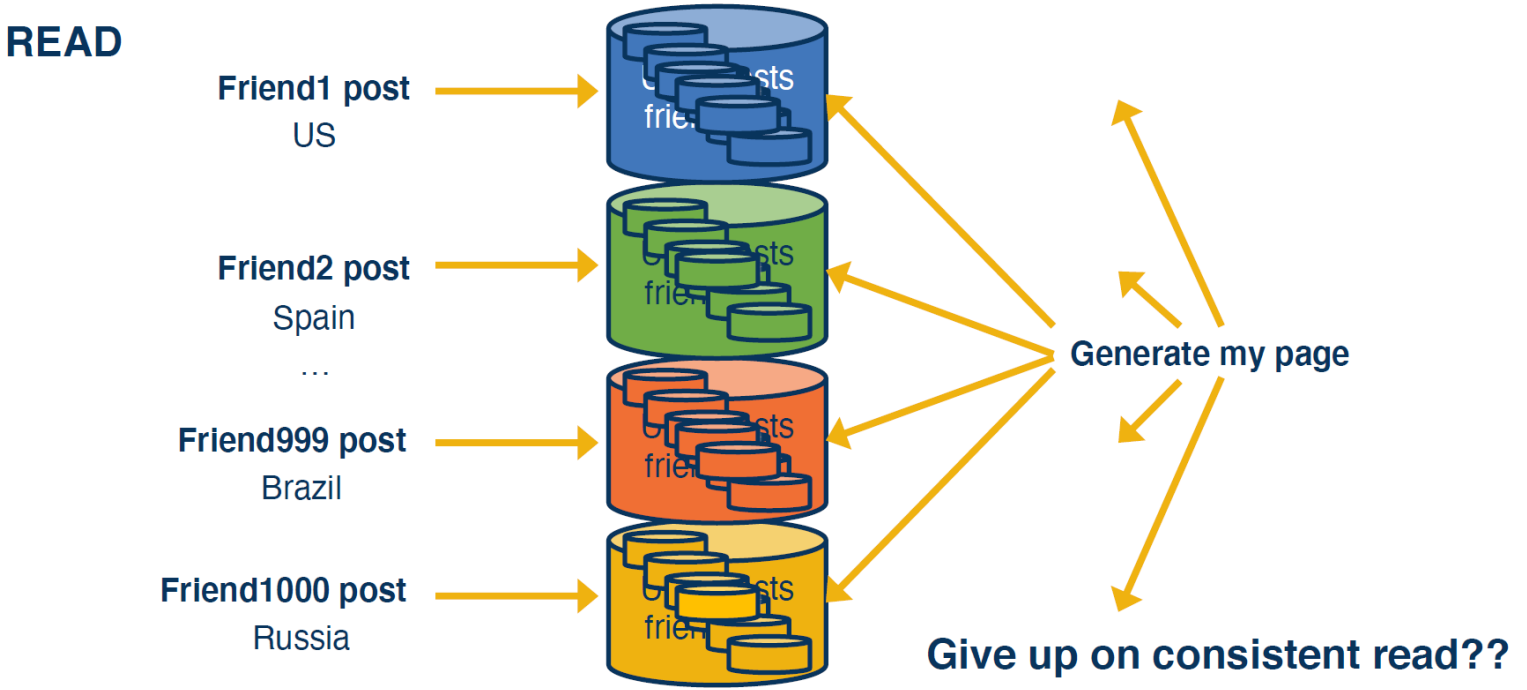


Figure Adapted from Spanner presentation at OSDI'12

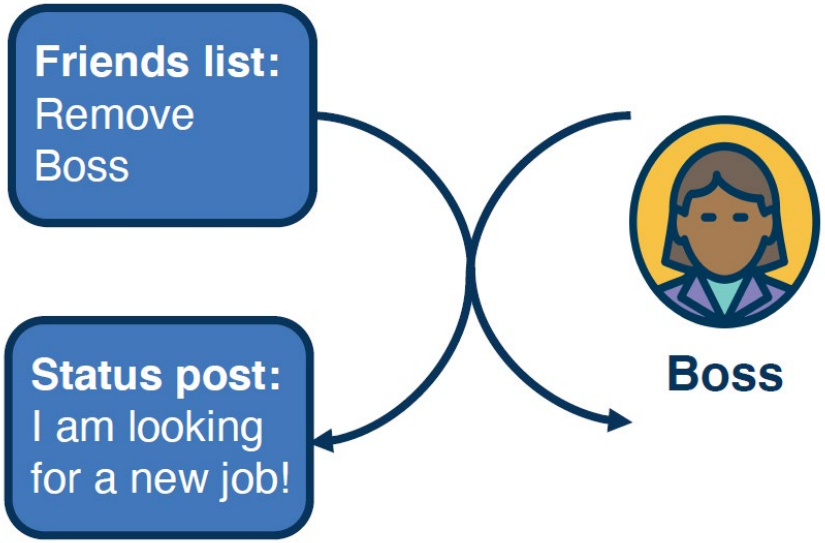
Consistency Matters!

External consistency matters

- System ordering of events = “real world” ordering of events
- Tx1 commits *before* Tx2 then Tx2 must see Tx1’s writes

This is **serializability**

How to achieve?



Google TrueTime

TrueTime != absolute “real” time

TrueTime is a *bounded* uncertainty around real time

Probe master clocks periodically

- Master clocks are “close”
- Master clocks are highly accurate
  - Use GPS
  - Use atomic clocks
- Bound is 2ϵ

Method	Returns
TT.now()	*TTinterval*: [*earliest*, *latest*]
TT.after(*t*)	true if *t* has definitely passed
TT.before(*t*)	true if *t* has definitely not arrived

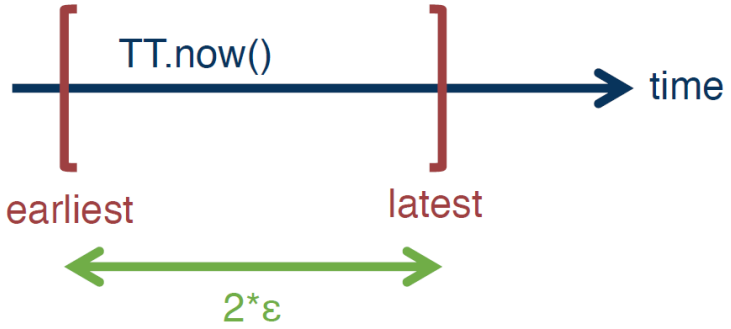


Figure Adapted from Spanner presentation at OSDI’12

Timestamps and TrueTime

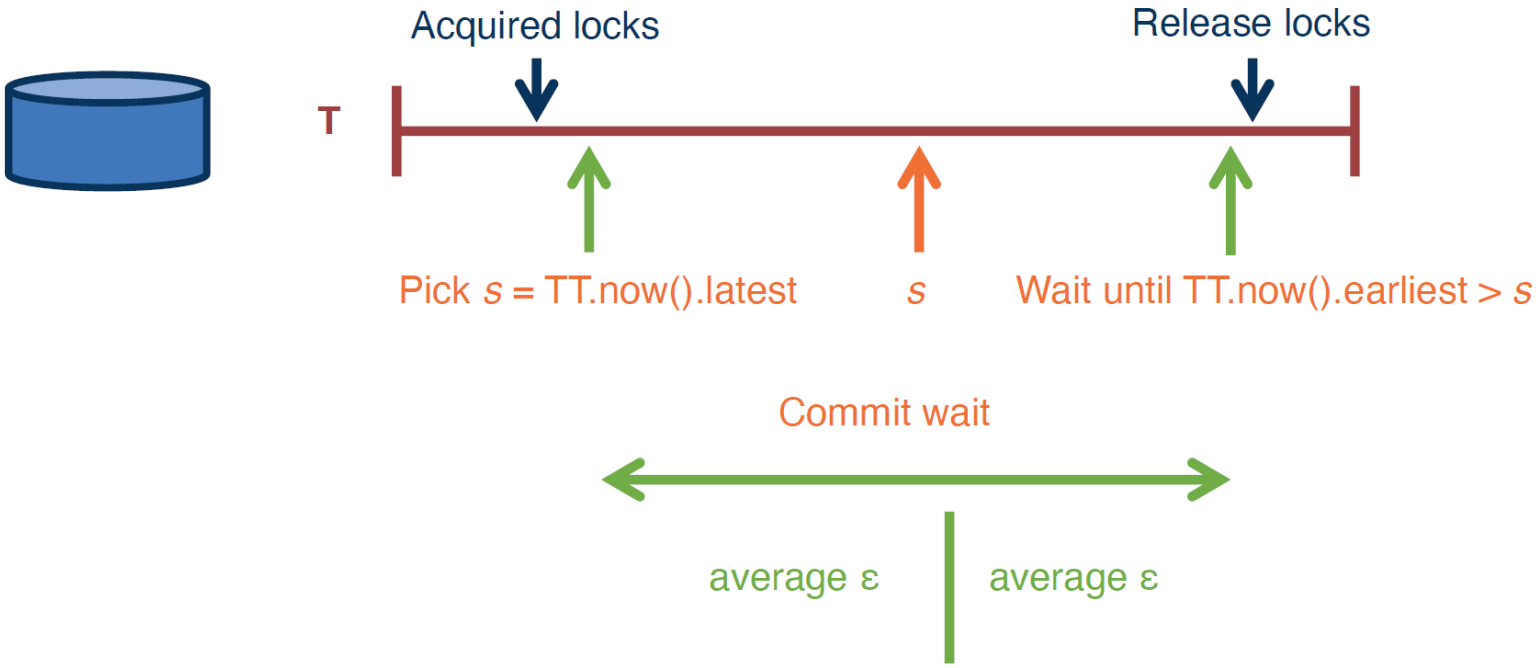
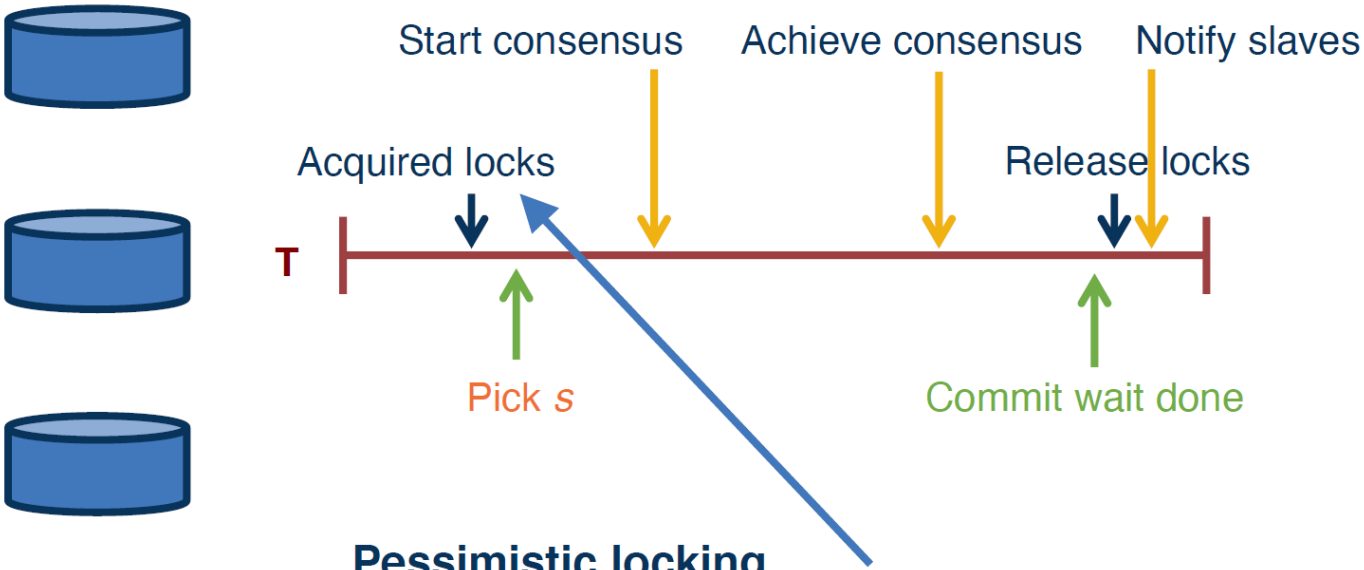


Figure Adapted from Spanner presentation at OSDI'12

Commit and Wait Replication



Pessimistic locking

- Acquire all locks up front, prevents later conflicts
- In **optimistic locking** aborts may be necessary, and more likely with long transactions

Figure Adapted from Spanner presentation at OSDI'12

Combine 2PC with Commit Wait

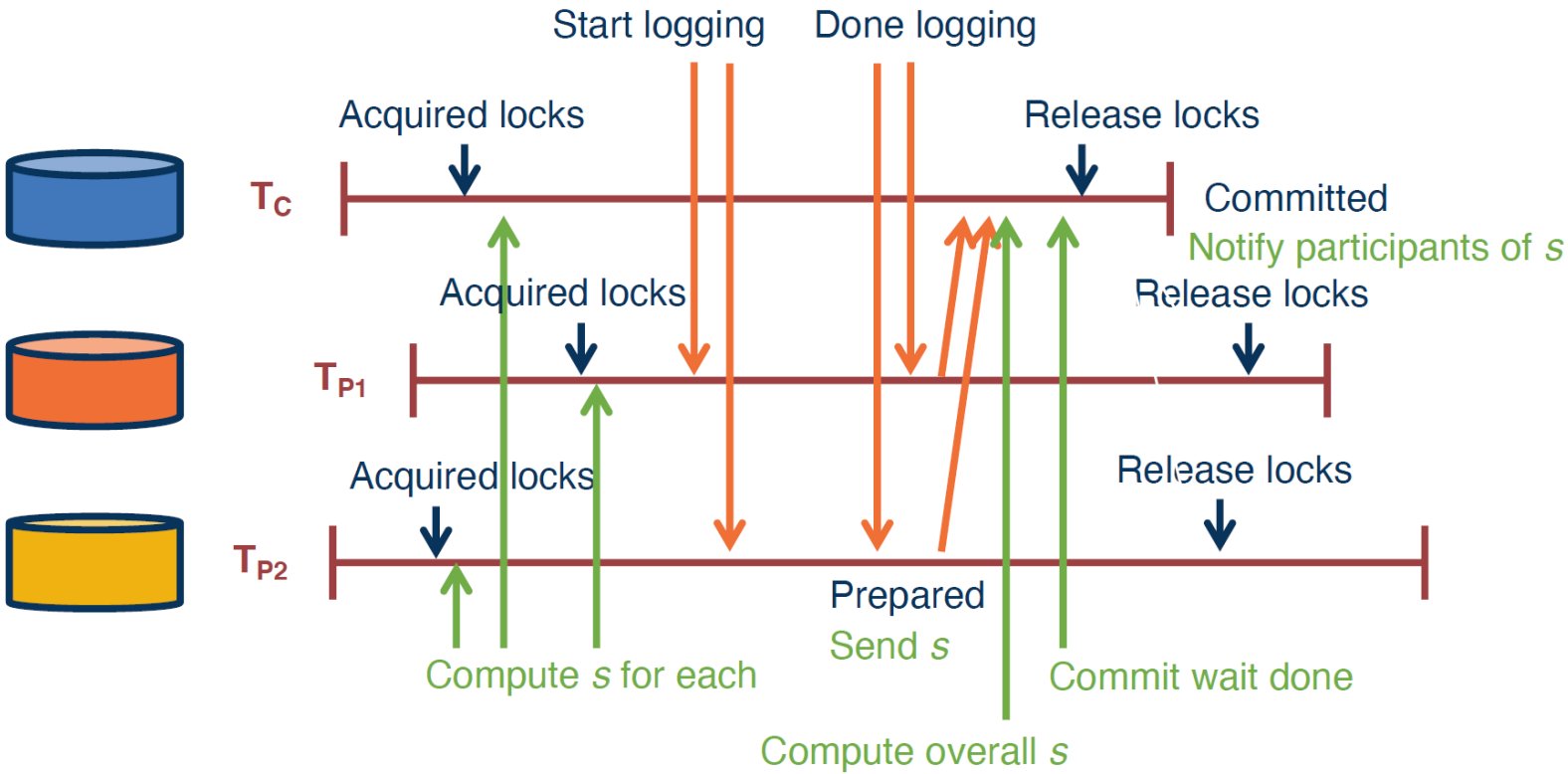


Figure Adapted from Spanner presentation at OSDI'12

Example ("Risky Post")

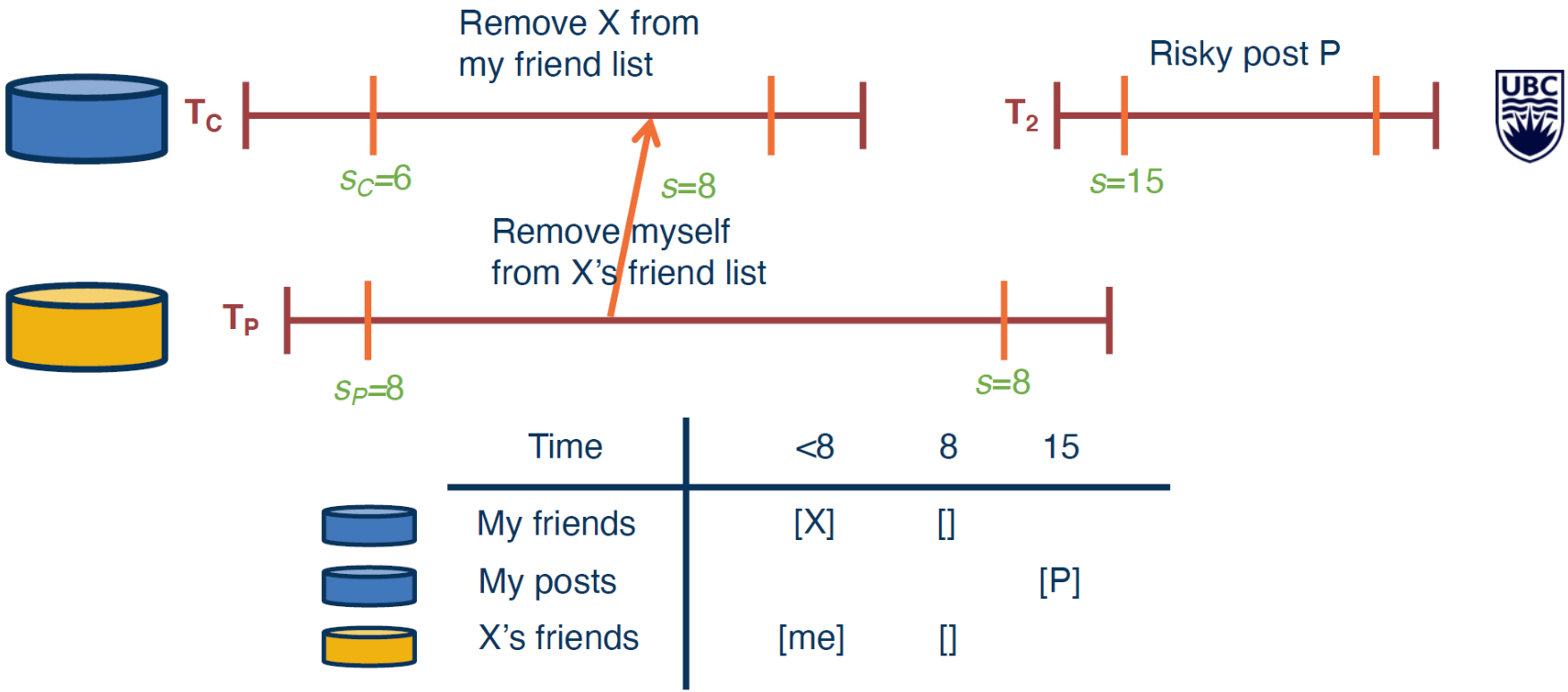


Figure Adapted from Spanner presentation at OSDI'12

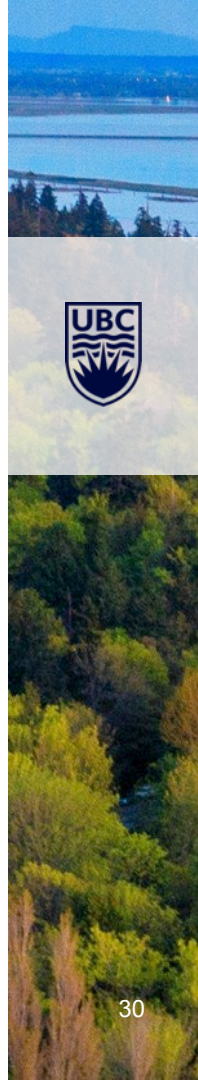
Read Transactions

Use timestamp for reading

- Timestamp is a *version*

No distributed cut required

- Use timestamp to obtain correct version



TrueTime is not *required*

GPS/Atomic clocks

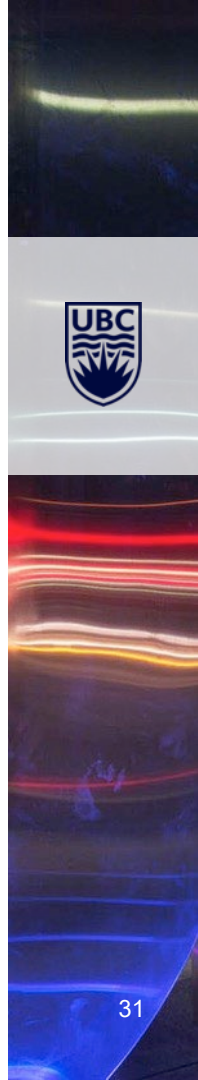
- Short windows (5-7 milliseconds)
- Delay response

NTP

- Longer window: ~100 milliseconds
- Wait?
- Sacrifice external consistency?
- Allow external consistency *when needed* (wait!)

CockroachDB:

- Default is *eventual consistency*
- Optional *external consistency*



Snapshot Isolation

Optimistic Concurrency Control (OCC)

Requires Atomicity (**A**CID)

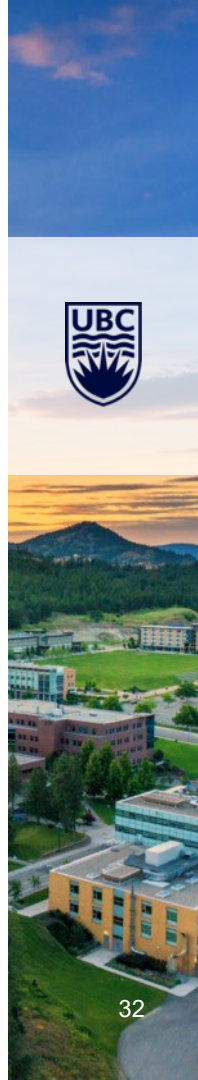
Requires Isolation (AC**I**D)

Snapshot isolation

Multi-version concurrency control

Correctness guaranteed:

- Transactions read from a version (snapshot) of distributed state
- Sequence of snapshots is sterializable (no cycle)



Amazon Web Services Aurora

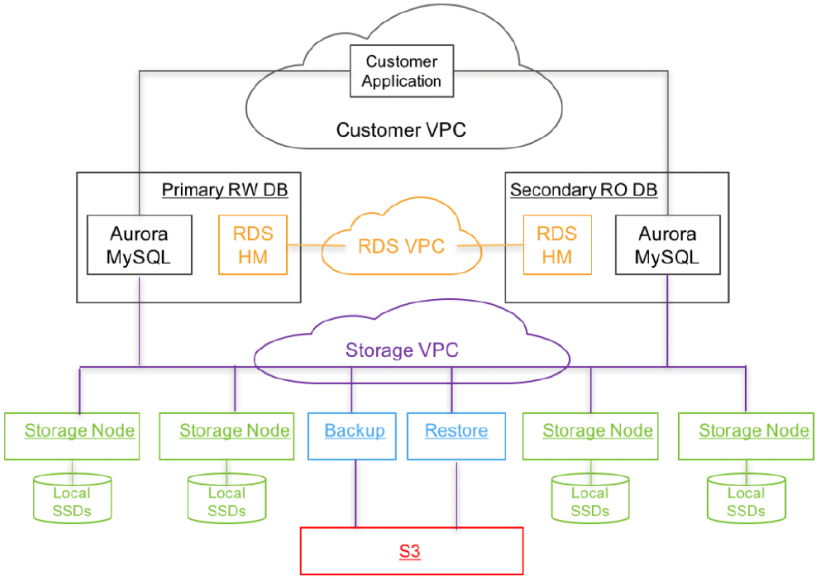
SIGMOD 2015

Primary/Backups Model

High Availability:

- Two nodes in a zone
- Three zones
- Quorum replication model

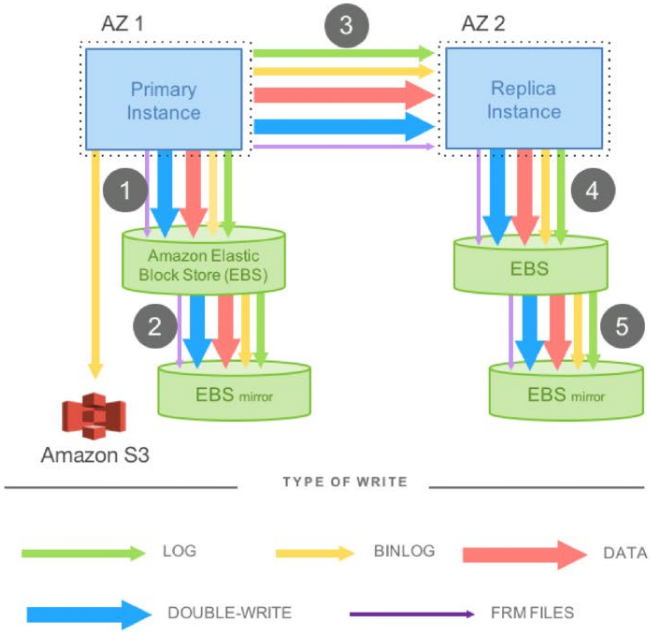
Expensive: 6 replicas ⇒ I/O amplification!



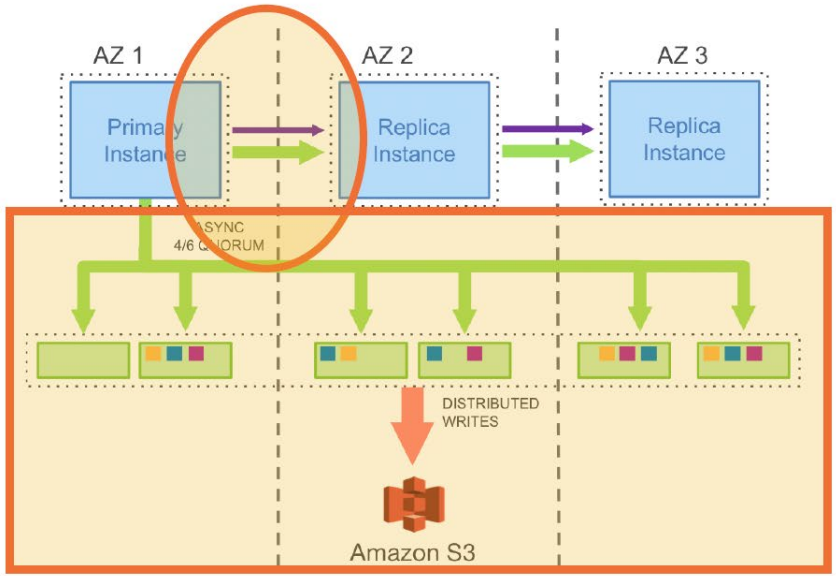
Aurora Architecture: A Bird's Eye View

Adapted from Aurora SIGMOD'17 paper

AWS Aurora (versus MySQL)



Network IO in Mirrored MySQL



Network IO in Amazon Aurora

Adapted from Aurora SIGMOD'17 paper

Lesson Summary

Distributed Transactions

Multiple techniques for Distributed Transactions

Google Spanner: using most of the tools in our toolbox

- Paxos
- 2PC
- 2PL
- TrueTime

Other Systems:

- Optimistic concurrency control
- Snapshot isolation
- Multi-version concurrency control
- Log replication

Questions?


THE UNIVERSITY OF BRITISH COLUMBIA
THE UNIVERSITY OF BRITISH