CS 6400 A

Database Systems Concepts and Design

Lecture 19

11/04/24

Announcements

Course project updates

- Dec 2: Project Presentation (video submission)
- Dec 6: Project Demo
  - 15min per group over Zoom
  - Our designated final exam slot: 6:00 PM - 8:50 PM
- Dec 9: Code and documentation due

Paper presentation starts this Wednesday

- Please email your slides to the staff (cs6400-staff@groups.gatech.edu) by 2 p.m. on the day of the presentation.

Recap: SQL history and motivation

Initially developed in the early 1970's

By 1986, ANSI and ISO standard groups standardize SQL

- New versions of standard published in 1989, 1992, and more up to 2016

Dark times in 2000s

- NoSQL for Web 2.0
- Google's BigTable, Amazon's Dynamo
- Are relational databases dead?

NewSQL systems in 2010s

- SQL → No SQL → Not only SQL → NewSQL
- SQL withstands the test of time and continues to evolve

The rise of NewSQL

Online transaction processing (OLTP)
- Short-lived, read/write transactions
- Touch a small subset of data using indexes
- Repetitive

Online analytical processing (OLAP)
- Introduced in the 2000's as Data Warehouses for analyzing large data
- Complex read-only queries (aggregations, multi-way joins)

At some point, OLTP was not fast enough, which led to NoSQL systems

Now we have NewSQL: NoSQL performance for OLTP + ACID
- Sacrificing ACID for better performance is no longer worth the effort

Spanner: Google's Globally-Distributed Database

Case study: Google Spanner

Google Spanner

Main features:

- Distributed, multi-version database
- General-purpose transactions (ACID)
- SQL query language
- Semi-relational data model
- Scales to millions of machines across hundreds of data centers and trillions of database rows

Used by Google Ads (has the most valuable database in Google) among others

Cloud Spanner 101: Google's mission-critical relational database (Google Cloud Next ‘17)

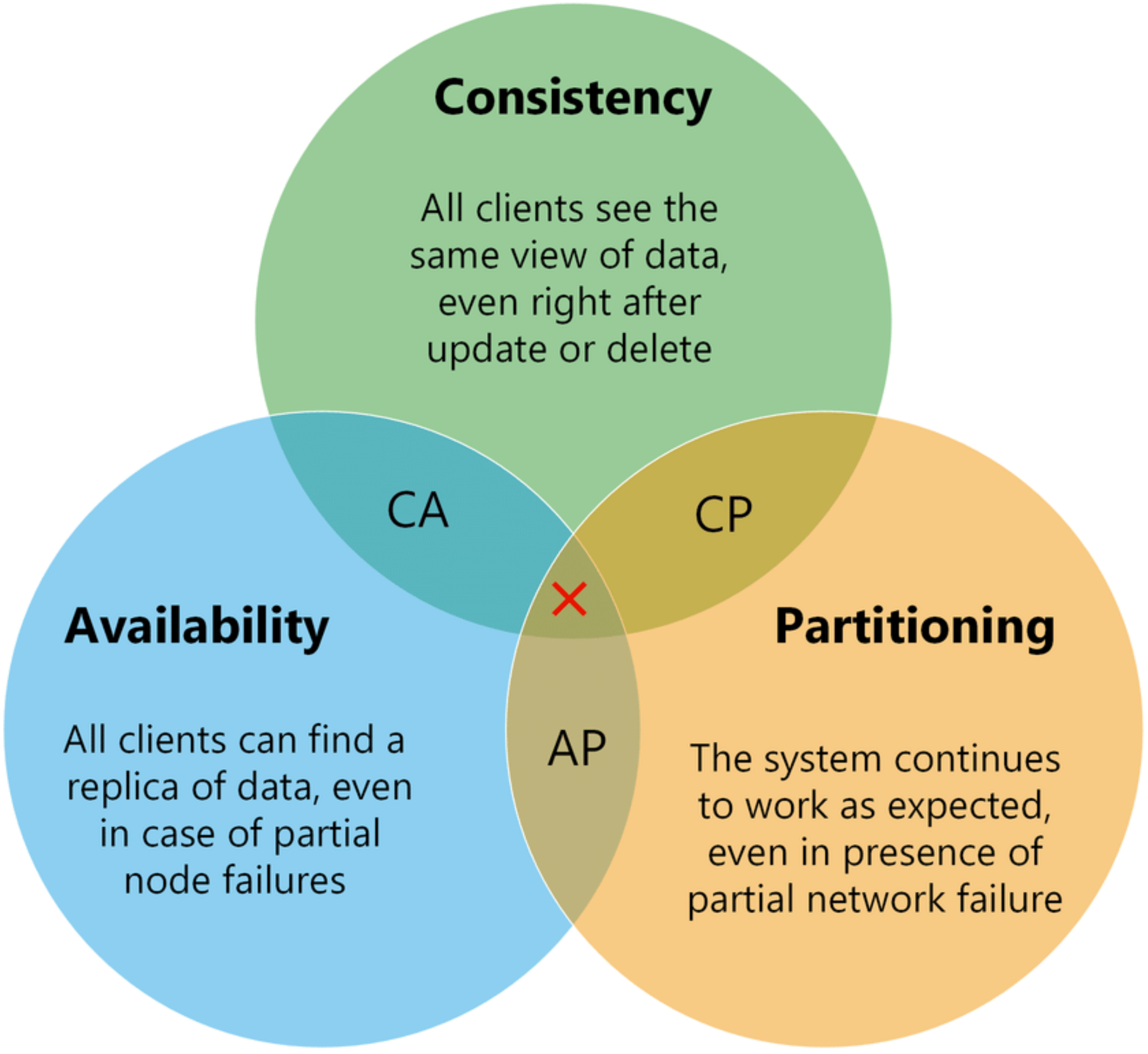
Summary: History of Spanner

- Previously, Google used sharded MySQL for their Ads database
- At some point, resharding took multiple years
  - Remember: cannot afford to shutdown Ads system, so need to do this carefully
- Could not use existing NoSQL databases (BigTable, Megastore) because they either did not fully support ACID transactions or were too slow
- Took 5 years to develop Spanner, and 5 more years to make it available on Cloud
  - These systems are not easy to implement!

CAP Theorem

Any distributed data store can provide only two of the following three guarantees: **C**onsistency, **A**vailability, and **P**artition-tolerance.

AP: eventual consistency
CP: strong consistency



Q: Which properties in the CAP theorem do Spanner provide?

Image source: https://medium.com/nerd-for-tech/understand-cap-theorem-751f0672890e

Data model

- Not purely relation but pretty similar
- Create tables using SQL DDL

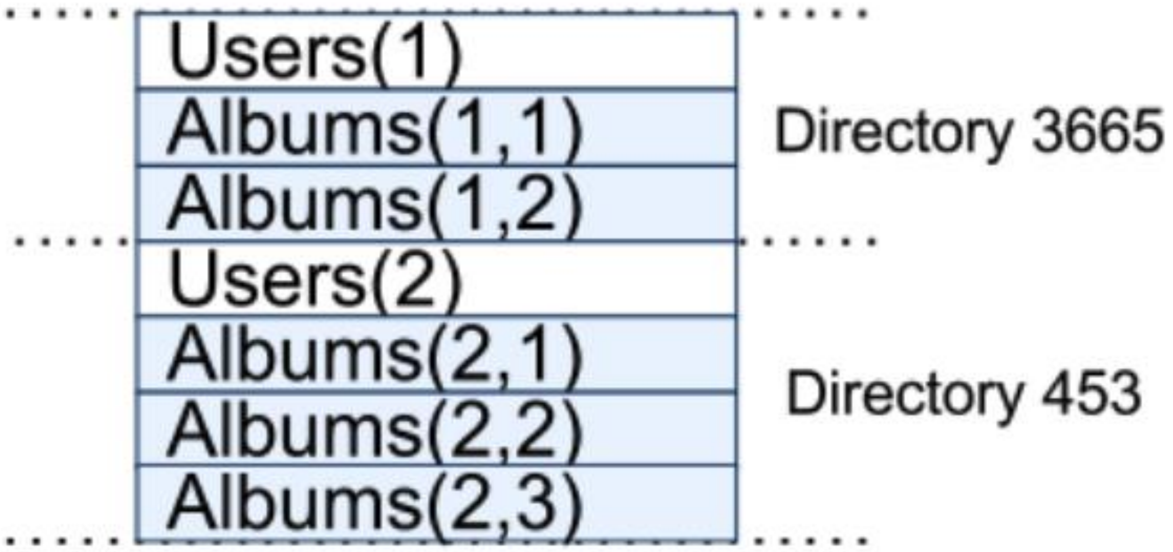
```
CREATE TABLE Users {
  uid INT64 NOT NULL, email STRING
} PRIMARY KEY (uid), DIRECTORY;

CREATE TABLE Albums {
  uid INT64 NOT NULL, aid INT64 NOT NULL,
  name STRING
} PRIMARY KEY (uid, aid),
  INTERLEAVE IN PARENT Users ON DELETE CASCADE;
```

Hierarchies

Data model

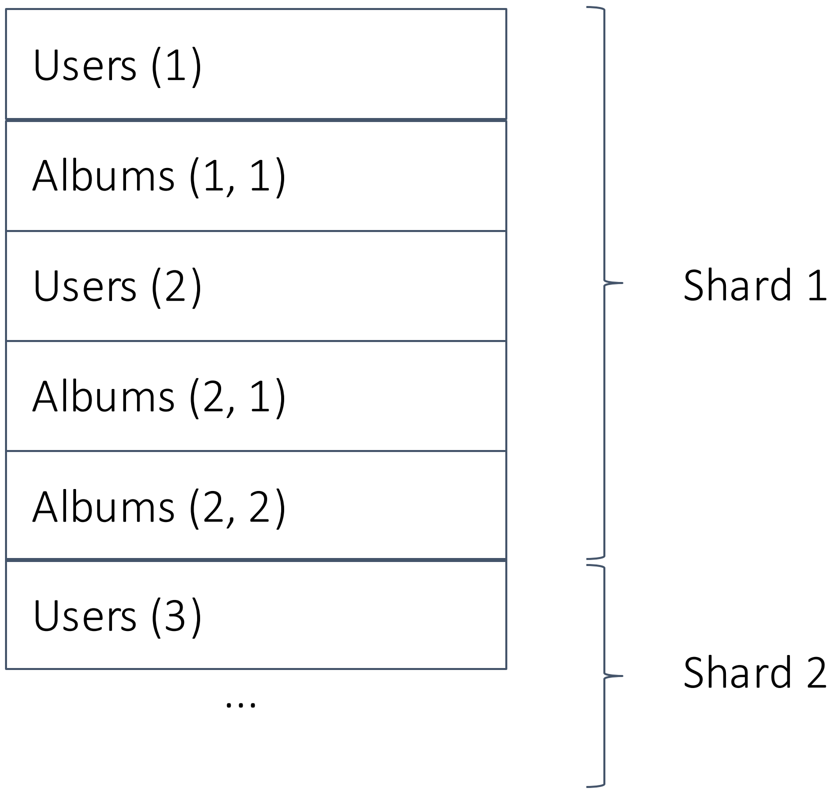
- Users(uid, email)
- Albums(uid, aid, name)

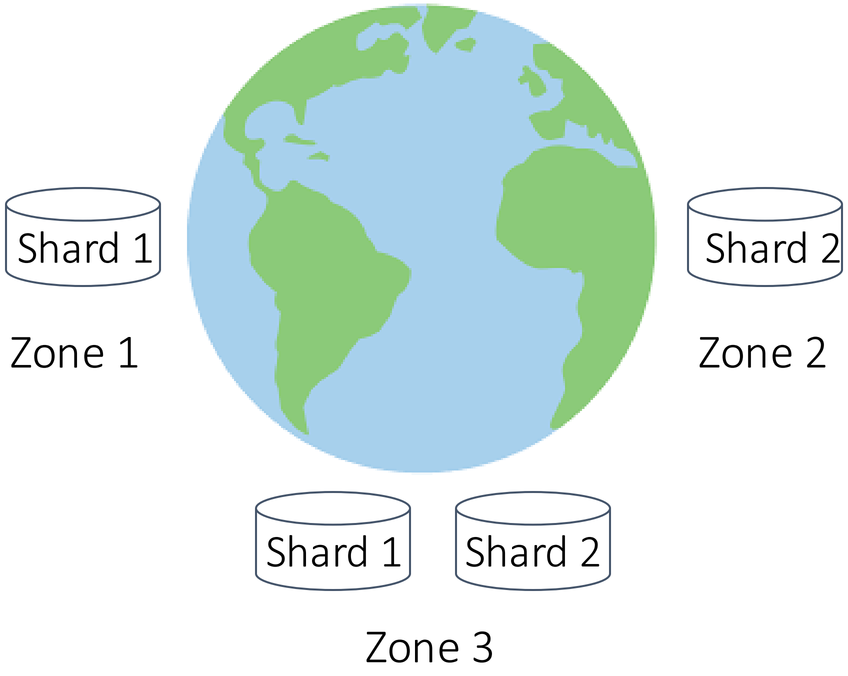
Tables can be interleaved for better locality



Data model

- Each directory/shard is a unit of data movement (e.g., place shard 1 in Zones 1 and 3)





Motivating example: banking

Start with $50 in account (consists of checkings and savings accounts)

- T1: deposit $150 on savings account
- T2: debit $200 from checkings account

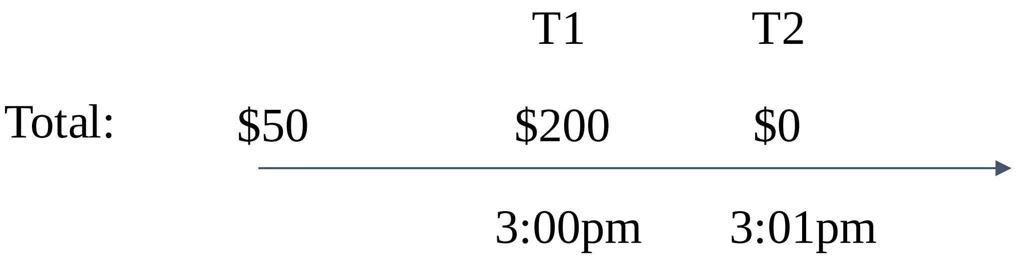
Say client (i.e., you) issues T1 and then T2

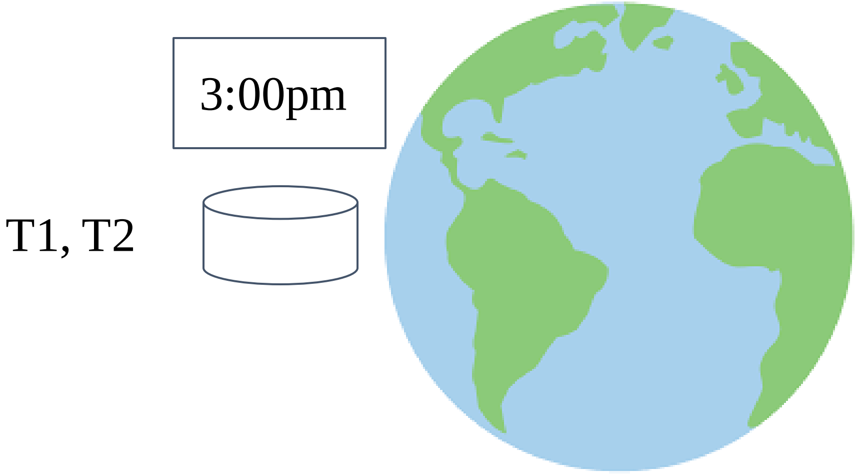
Suppose **total balance must not be negative at any point**

- That is, Spanner must never run T2 and then T1

Easy on single-machine database

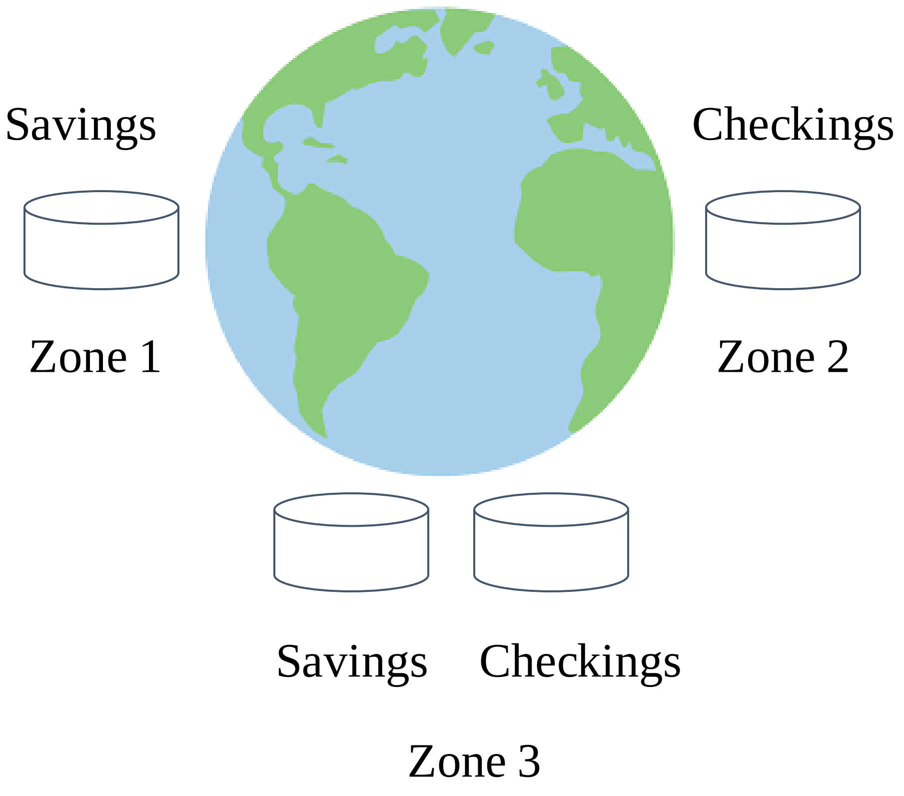
- Give monotonically-increasing timestamps to T1 and then T2
- If another transaction reads the database, use snapshot with most recent timestamp
  - Total balance is never negative





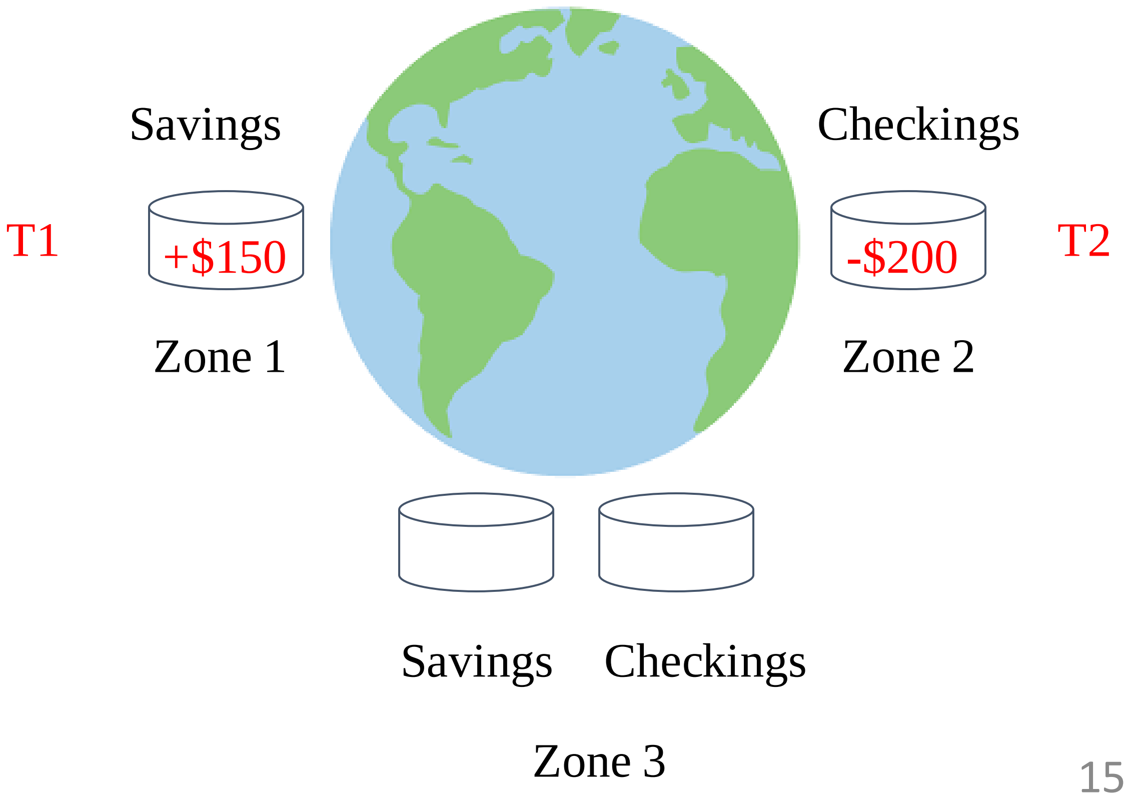
Not easy if database is distributed

- Suppose database is sharded and replicated in three different data centers



Challenge 1: consistency

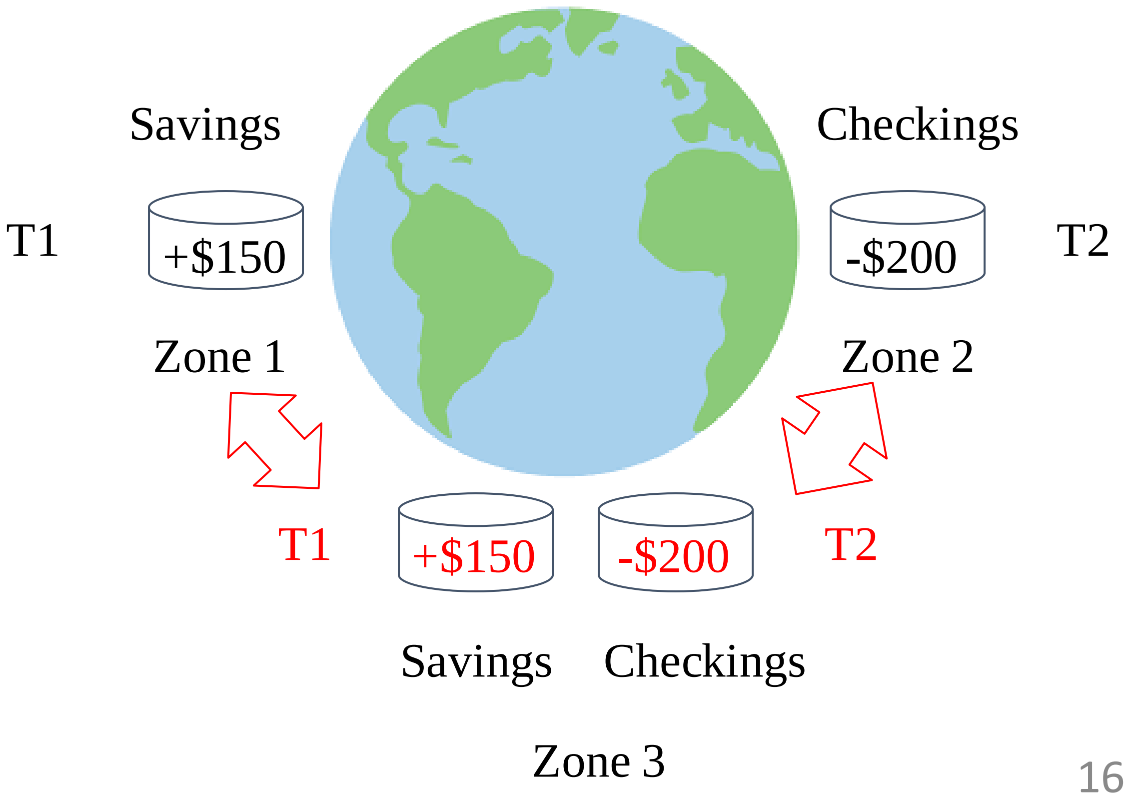
- Need to write on replicas as if there was a single transaction running



Challenge 1: consistency

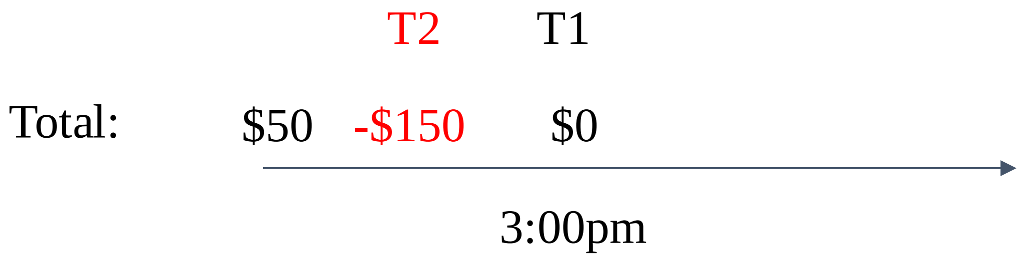
- Need to write on replicas as if there was a single transaction running

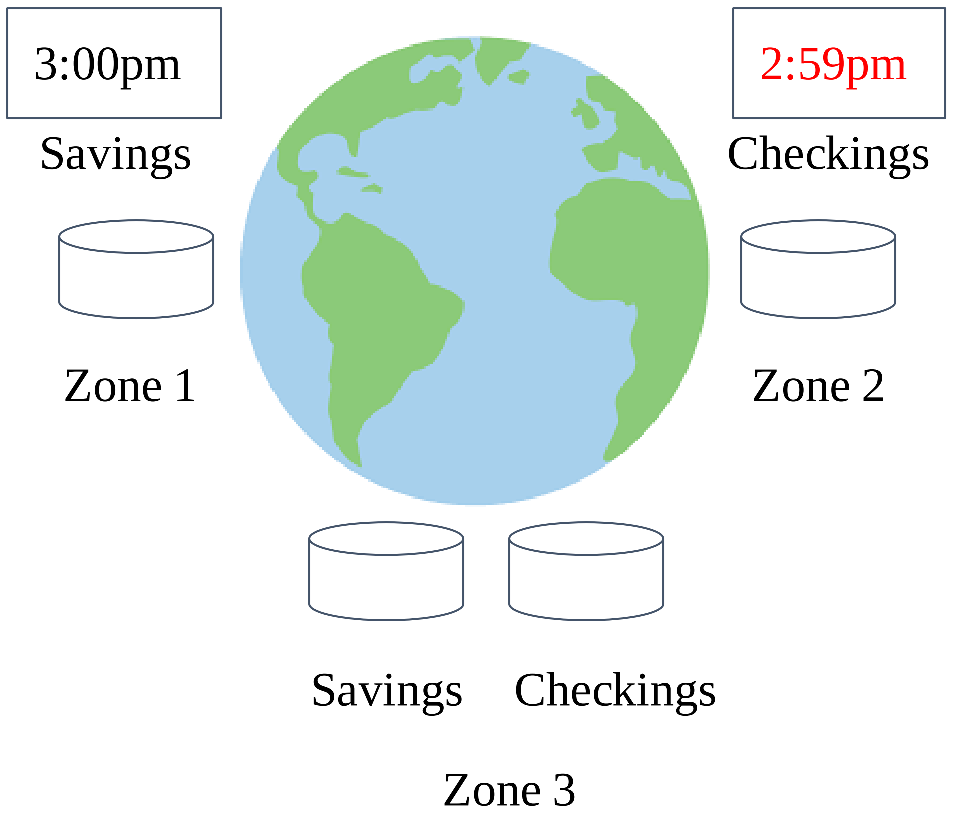
- Use existing distributed database techniques
  - Use Paxos algorithm for synchronizing writes
  - Will not go into details



Challenge 2: clock uncertainty

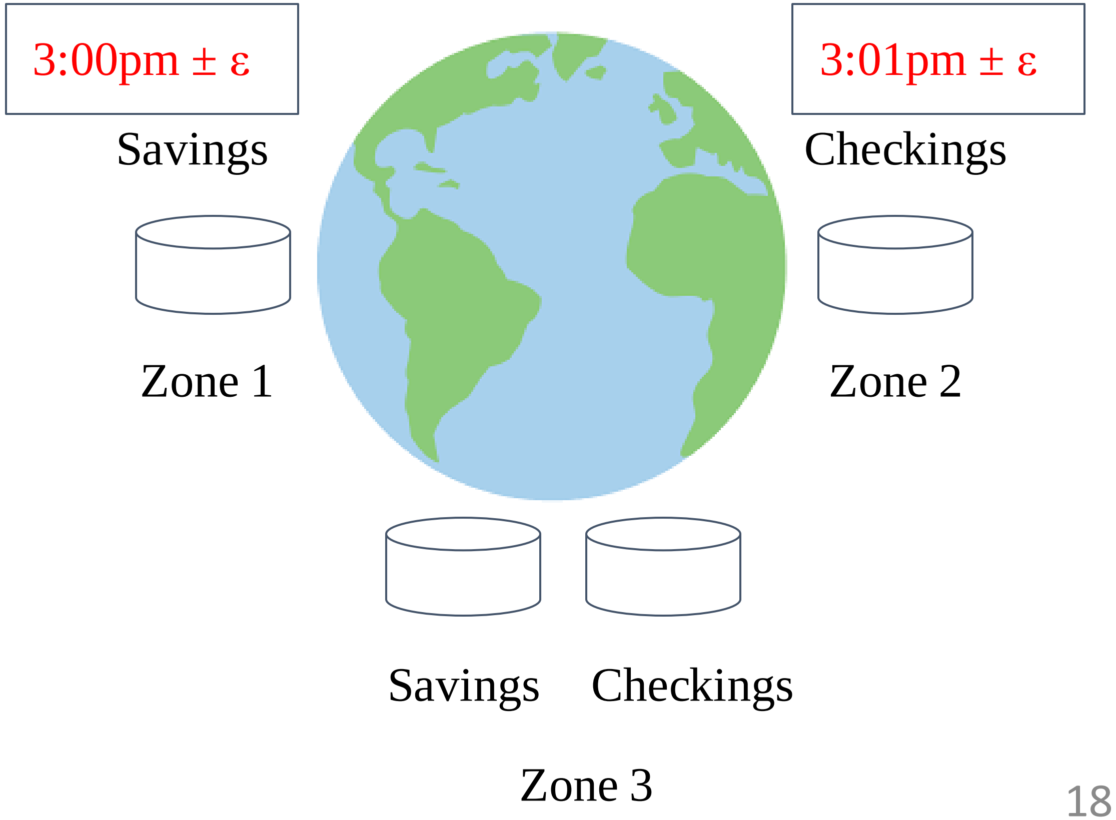
- If clock in Zone 1 is slower than Zone 2, then T2 may have a smaller timestamp than T1
- A transaction that reads after T2 sees a negative total balance!

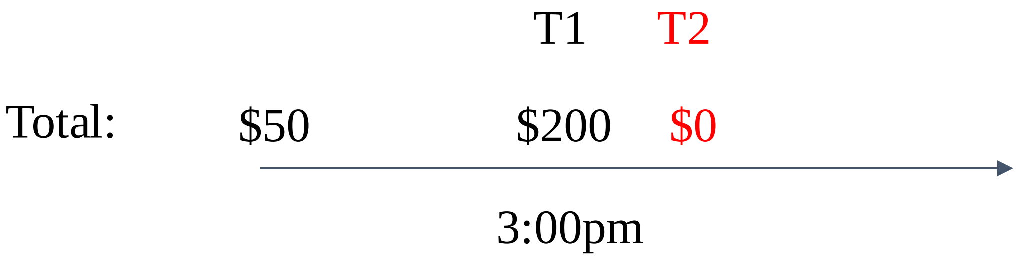




Solution: TrueTime

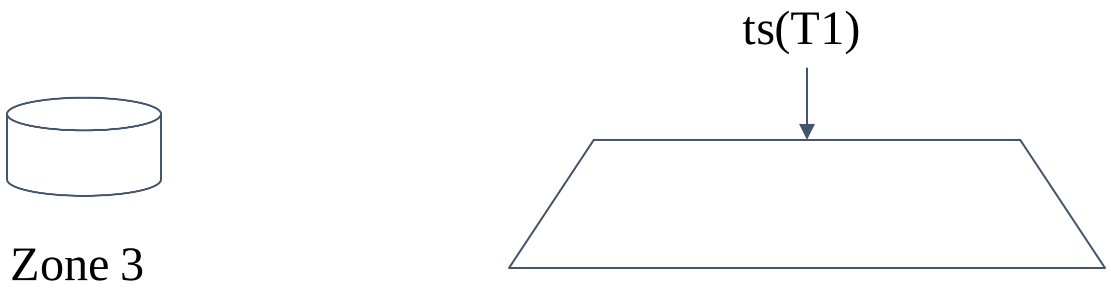
- Global time with bounded uncertainty
- Guarantees that if T1 commits before T2 starts, then ts(T1) < ts(T2)
- Spanner “waits out” any uncertainty





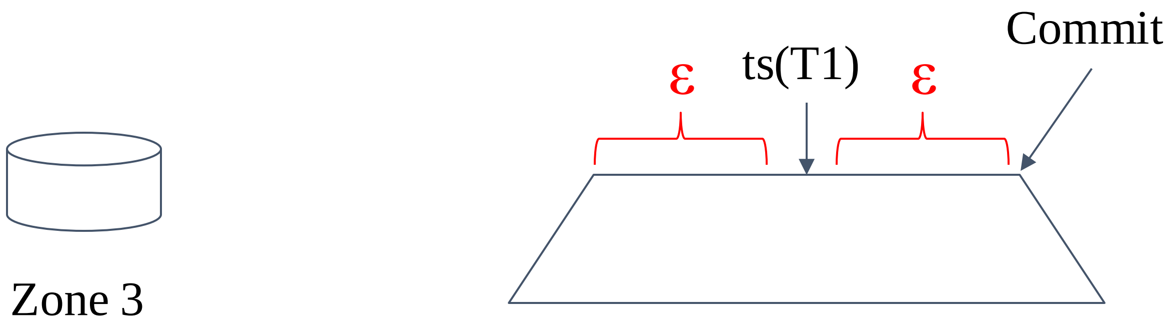
Running T1 and T2

- Use strict 2PL (strict = keep locks until commit or abort)
- Timestamp is some time between when locks are acquired and released
- In addition, need to take care of:



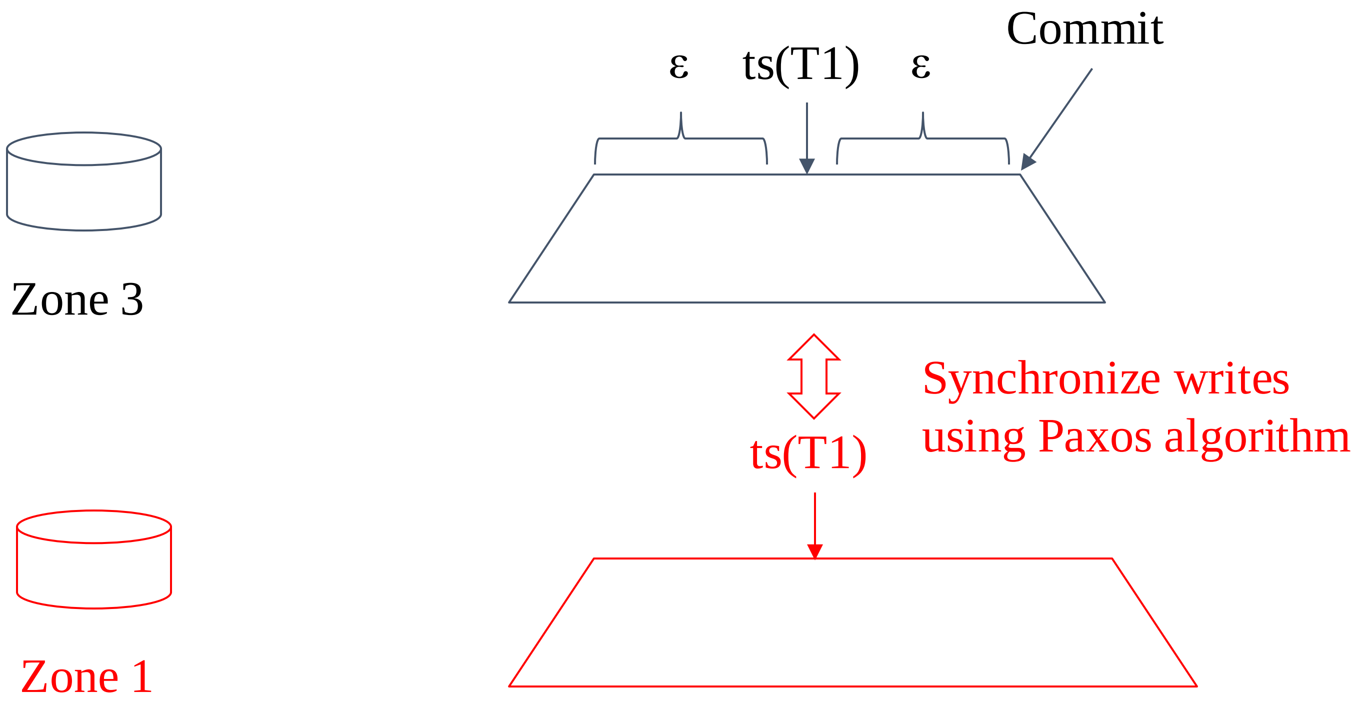
Running T1 and T2

- Use strict 2PL (strict = keep locks until commit or abort)
- Timestamp is some time between when locks are acquired and released
- In addition, need to take care of: time uncertainty and



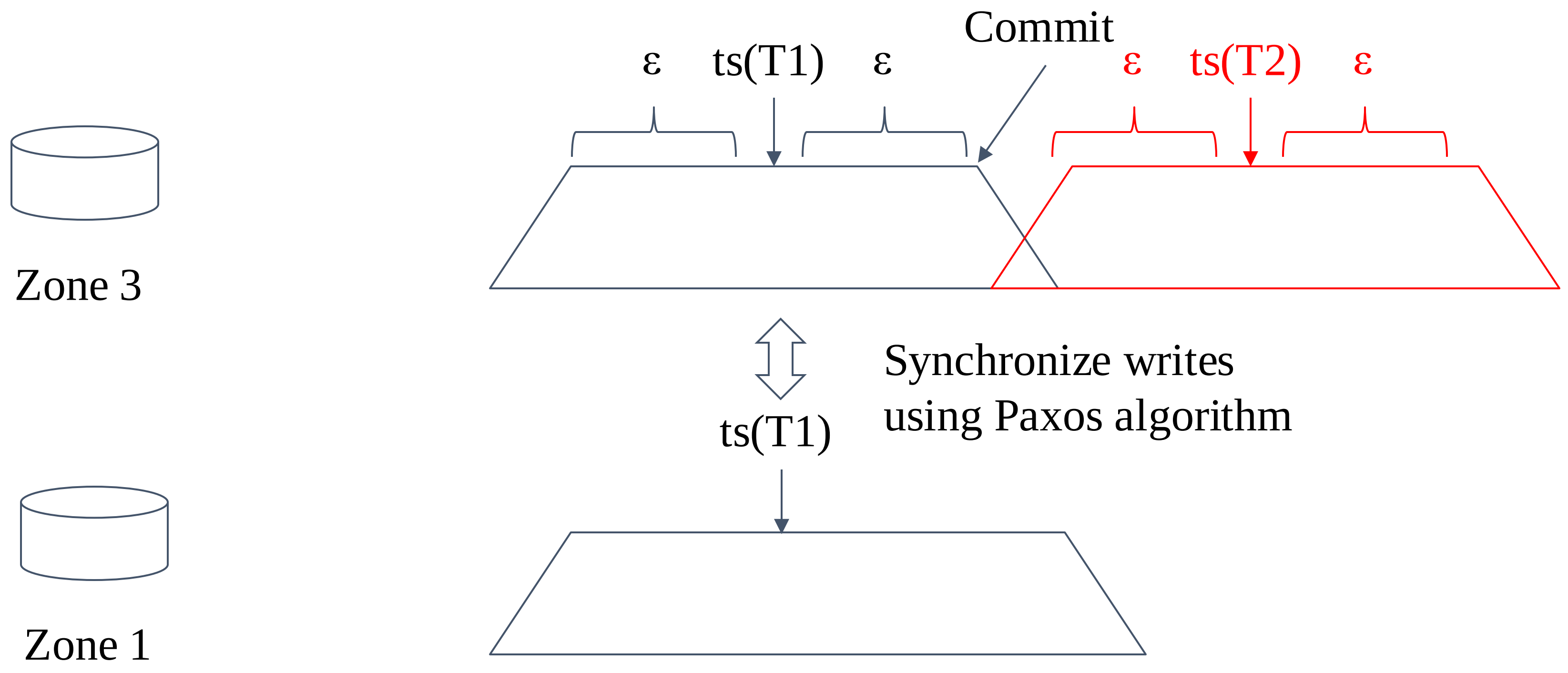
Running T1 and T2

- Use strict 2PL (strict = keep locks until commit or abort)
- Timestamp is some time between when locks are acquired and released
- In addition, need to take care of: time uncertainty and consensus with replicas



Running T1 and T2

- Use strict 2PL (strict = keep locks until commit or abort)
- Timestamp is some time between when locks are acquired and released
- In addition, need to take care of: time uncertainty and consensus with replicas

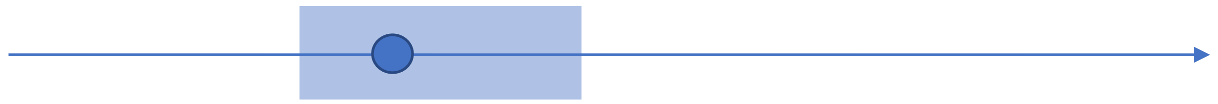


True Time

Idea: There is a global “true” time t

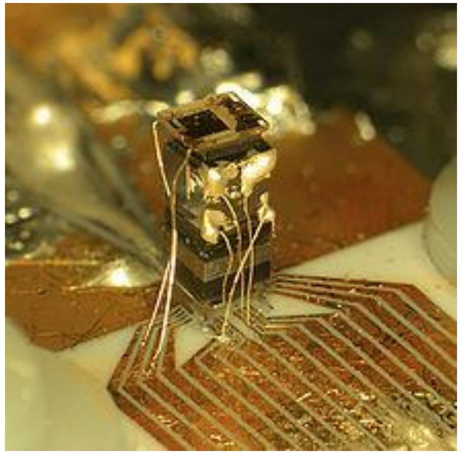
TT.now() = t ∈ [earliest, latest]

- TT.now().earliest: definitely in the past
- TT.now().latest: definitely in the future



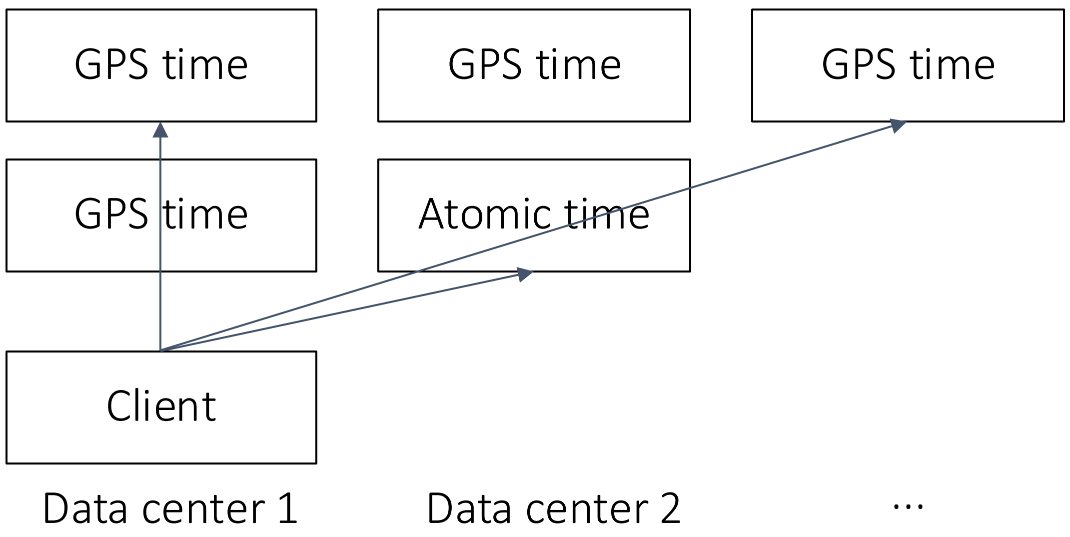
TrueTime implementation

- Use time master machines that have GPS or atomic clocks
  - GPS is precise, but may have connection problems
  - Atomic clocks do not have connections, but may drift
  - The two types complement each other and are not expensive

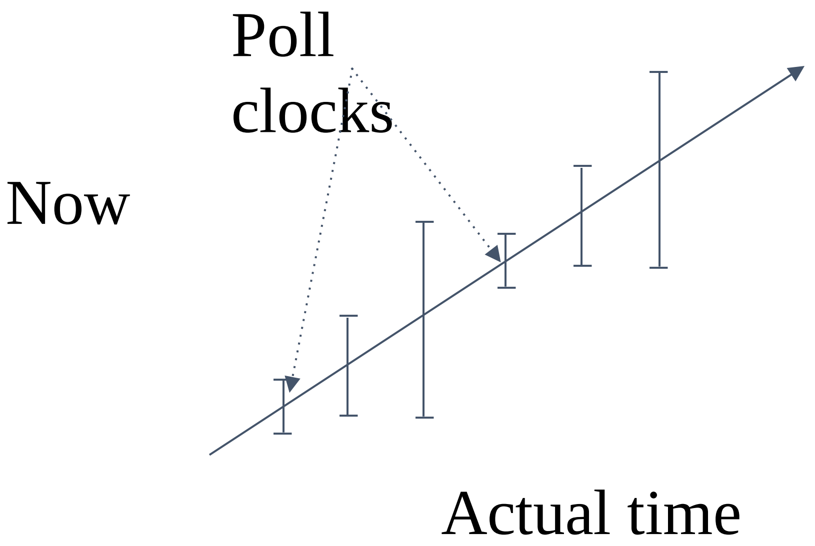


TrueTime implementation

- Step 1: periodically poll [earliest, latest] of selected GPS and atomic clock times
- Initially, [earliest, latest] = now ± ε



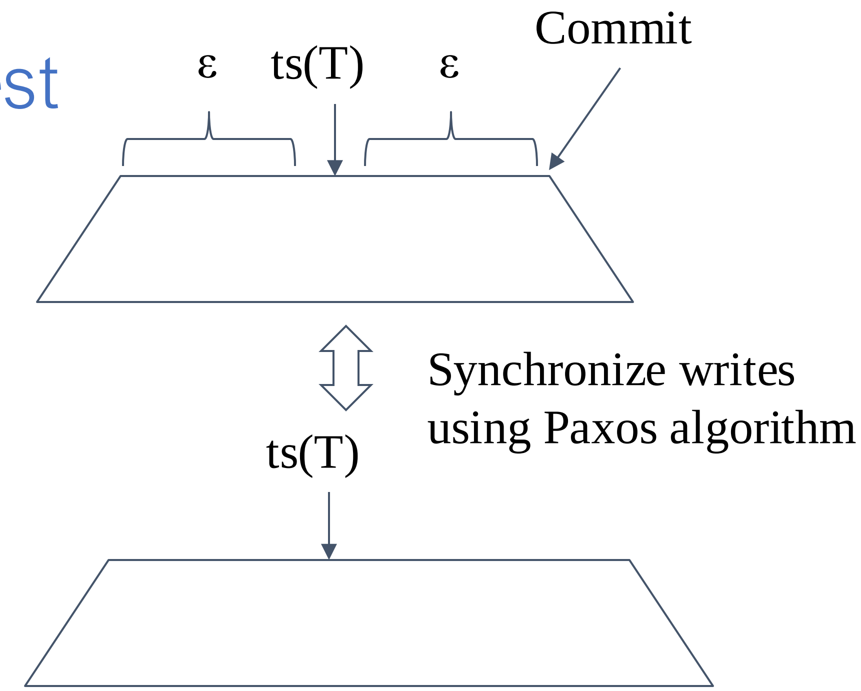
TrueTime implementation



- Step 2: reflect local clock drift between polls
- Recall we start from [earliest, latest] = now ± ε
- If X seconds passed,
  - now += X seconds
  - ε += X * 200μs (200μs per second is an upper bound of clock drift)
- Basically clock becomes more and more uncertain until we poll again

Transaction protocol

1. Acquire locks
2. Execute reads
3. Pick commit timestamp T = TT.now().latest
4. Replicate writes using Paxos
5. Wait until TT.now().earliest > T
6. Commit
7. Apply write
8. Release locks



MVCC for read-only
2PL for read-write

Current read: T = TT.now().latest

Guarantee: external consistency

In Spanner, commit order (= timestamp order) respects global wall-time order

- System behaves as if all (conflicting) transactions were executed sequentially in one machine

External Consistency: If T1 commits before T2 starts, T1 should be serialized before T2. In other words, T2's commit timestamp should be greater than T1's commit timestamp.

NewSQL techniques

Main memory storage
- Entire database can be stored in memory

Partitioning/sharding
- Not a new idea, but now feasible to implement high performance distributed DBMS

Concurrency control
- Use variants of time-stamping ordering concurrency control

Secondary indexes
- Challenge is to implement these on a distributed system

Replication
- Most support strongly consistent replication

Crash recovery
- Need to perform in a distributed DBMS

Source: https://db.cs.cmu.edu/papers/2016/pavlo-newsql-sigmodrec2016.pdf

NewSQL summary

Some applications need SQL, ACID transactions, and scalability at the same time

NewSQL systems require significant engineering effort, but are now commercialized

- The individual techniques are not new, but incorporating them into a single platform is