CS 4440 A

Emerging Database Technologies

Lecture 15

03/25/24

Announcements

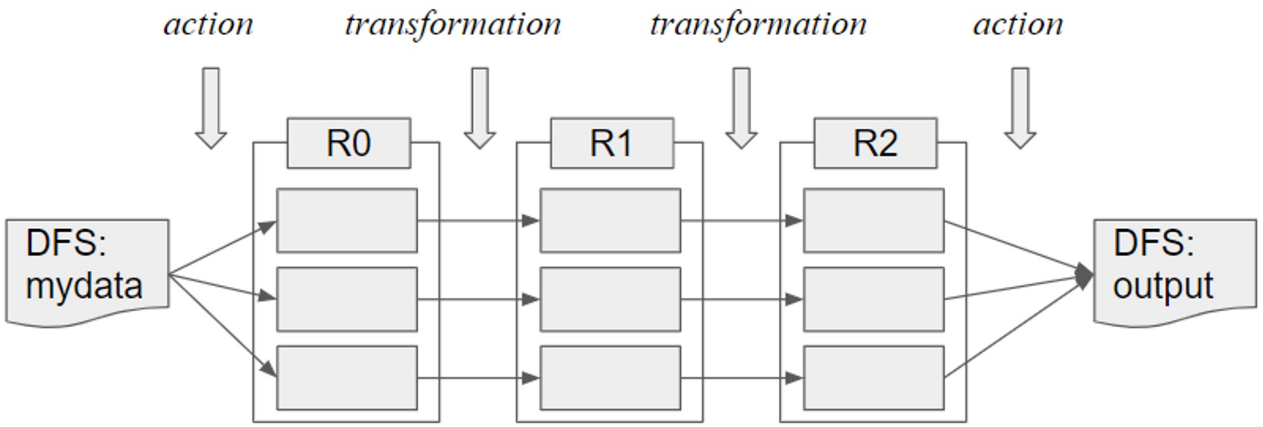
- Midterm grade distribution:
    - median: 85.5, mean: 84.67, std: 7.3, max: 96
    - Solution available on Canvas under Files
- Assignments 4 grades released
- Assignments 3, 5 released will be posted this week

Announcements

- Assignments 6, 7 released
    - Research paper presentation
    - Research paper critique

- Paper assignments: https://tinyurl.com/4xjk7duu
    - Paper presentation group = project group
    - 20min per group, plus 5min questions
    - First paper presentation: April 3 (next Wed)

Recap



- Distributed File System
- MapReduce
    - Map tasks ⇒ group by key ⇒ reduce tasks
- Spark
    - A workflow system
    - Resilient distributed dataset (RDD)
    - Transformations and actions
        - Map, flatmap, filter, reduce, join, groupByKey
    - Lazy evaluation and lineage

Review: 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

MySQL®

hadoop

Spark SQL

Google Spanner

The rise of NewSQL

- Online transaction processing (OLTP)
    - Read/write transactions are short-lived
    - Touch a small subset of data using indexes
    - Are 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

Case study: Google Spanner

Google Spanner

- State-of-the-art NewSQL database
  - Distributed multiversion 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
- Available to public through Google Cloud Spanner from 2017

History of Spanner

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

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

History of Spanner

- Most of Google’s revenue comes from selling ads
- 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!

How does it work

- We are going to start by reading the Spanner paper
- Spanner: Google's Globally-Distributed Database
  - Best Paper award at OSDI 2012

References:

- Spanner talks 1 and 2
- Cloud Spanner documentation
- Eric Brewer's paper
- Acknowledgements: some slides inspired by above material

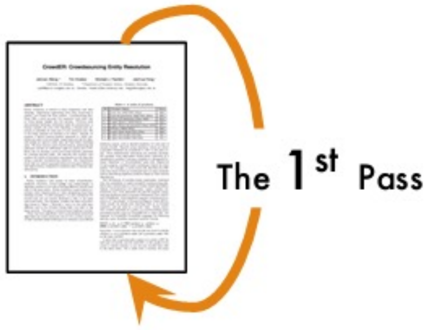
How to read a paper in depth

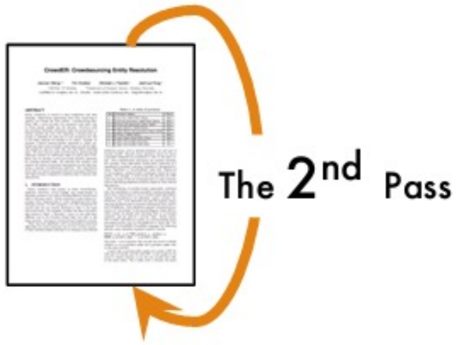
The "three-pass" approach [1]

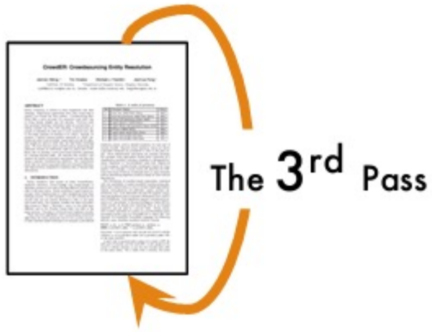
first pass: a quick scan

second pass: with greater care, but ignore the details

third pass: re-implementing the paper







---

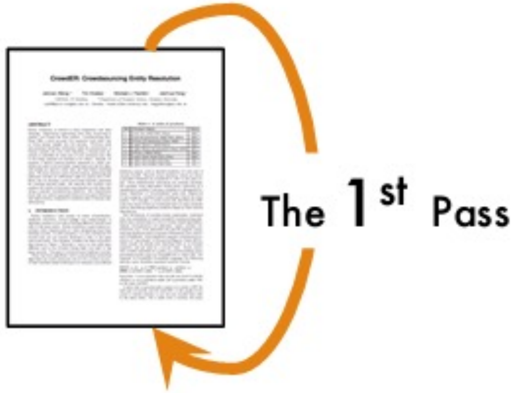
[1] S. Keshav. How to read a paper? http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/07/paper-reading.pdf

The first pass: a quick scan

Goal: get bird's-eye view of the paper (5~10 min)

What to read:
  - Title, abstract, introduction and conclusion
  - Section and sub-section headings
  - Main figures
  - Scan of bibliography

You should be able to answer:
  - What type of paper is this?
  - What are the main contributions?



The second pass: grasp the content

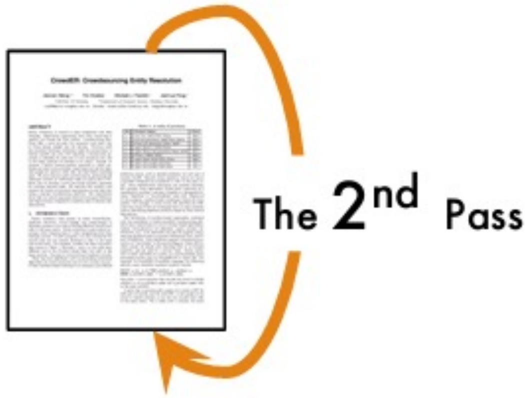
Goal: get a good understanding of the ”meat” of the paper

How to read:

- Look carefully at figures, diagrams and examples
- Take notes of questions, unread references etc.
- Ignore proofs, appendix, extensions etc.

You should be able to:

- Summarize main thrusts of the paper, with supporting evidence, to someone else



The third pass: all about the details

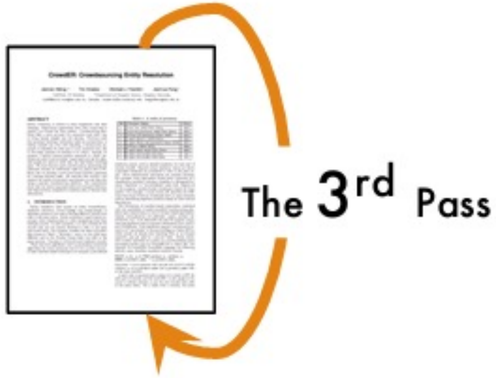
Goal: think about what you would have done if you were to re-implement such an idea

How to read:

- Challenge every assumption
- Compare your version with the actual paper
  - Often leads to questions like: why not do it this way?

You should be able to:

- Identify hidden assumptions/potential design flaws
- Get ideas for future work



Let’s try the first pass!

1. **Category**: What type of paper is this? A measurement paper? An analysis of an existing system? A description of a research prototype?
2. **Context**: Which other papers is it related to?
3. **Correctness**: Do the assumptions appear to be valid?
4. **Contributions**: What are the paper’s main contributions?
5. **Clarity**: Is the paper well written?

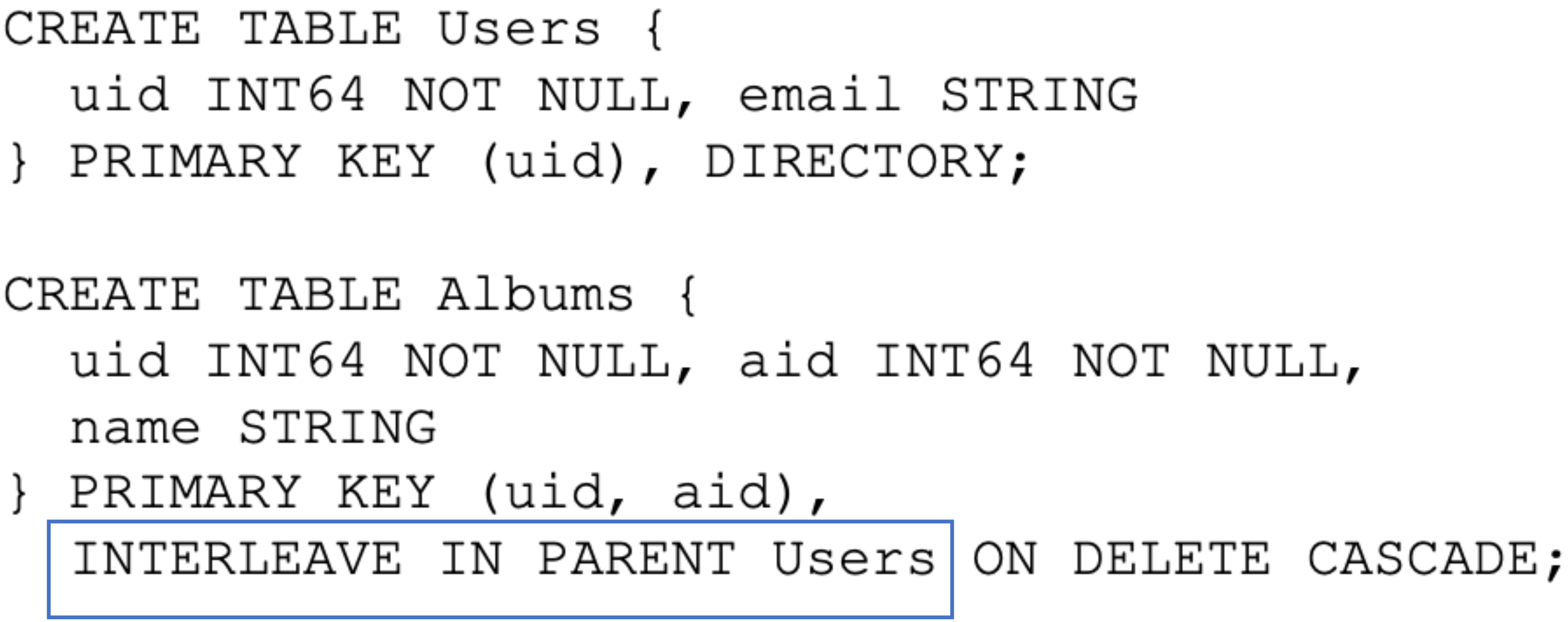
For research paper presentations

- Always start with the first pass to get a general impression
  - You should be able to give high-level answers to questions like “what problem the paper is trying to solve”, “why does it matter”, and “why is the problem challenging” after this pass
- Do a second pass to understand the main technical contributions
  - We have prepared a detailed reading guide for each paper that tells you which sections to focus on versus which sections to skip
- No need to do a third pass

Spanner: Google's Globally-Distributed Database

Data model

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



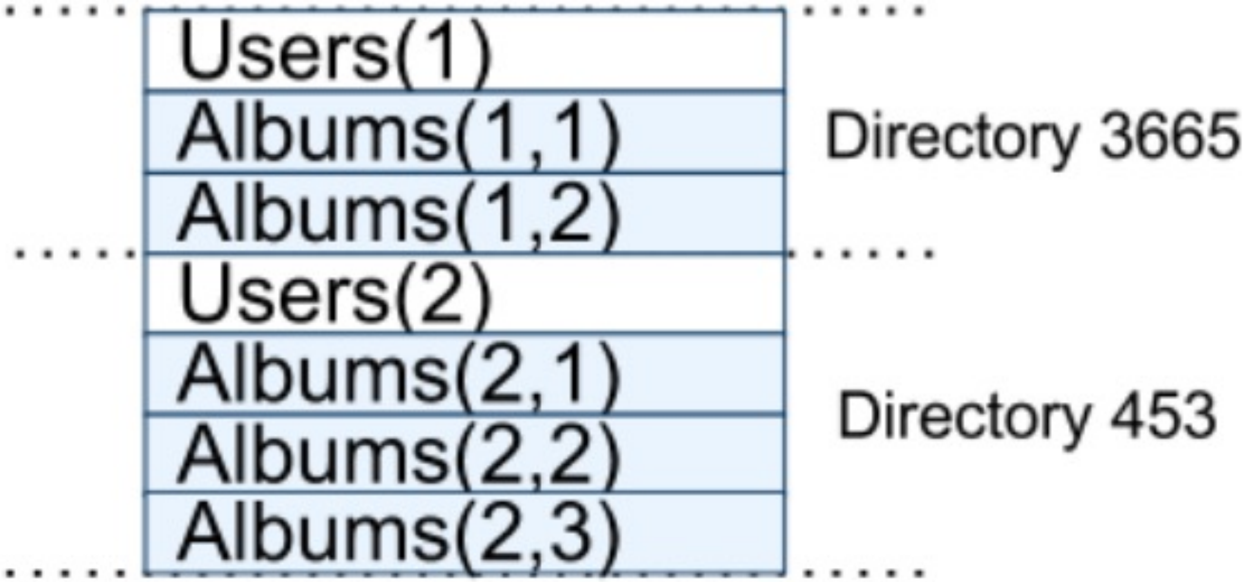
```
CREATE TABLE Users {
  uid INT64 NOT NULL, email STRING
} PRIMARY KEY (uid), DIRECTORY;

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)

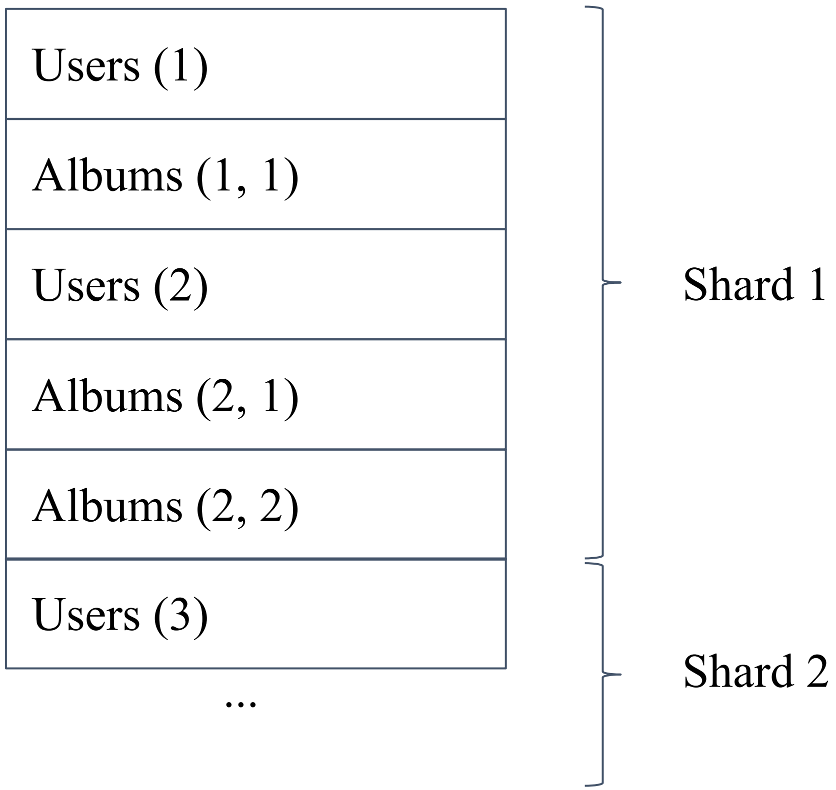
Tables can be interleaved for better locality

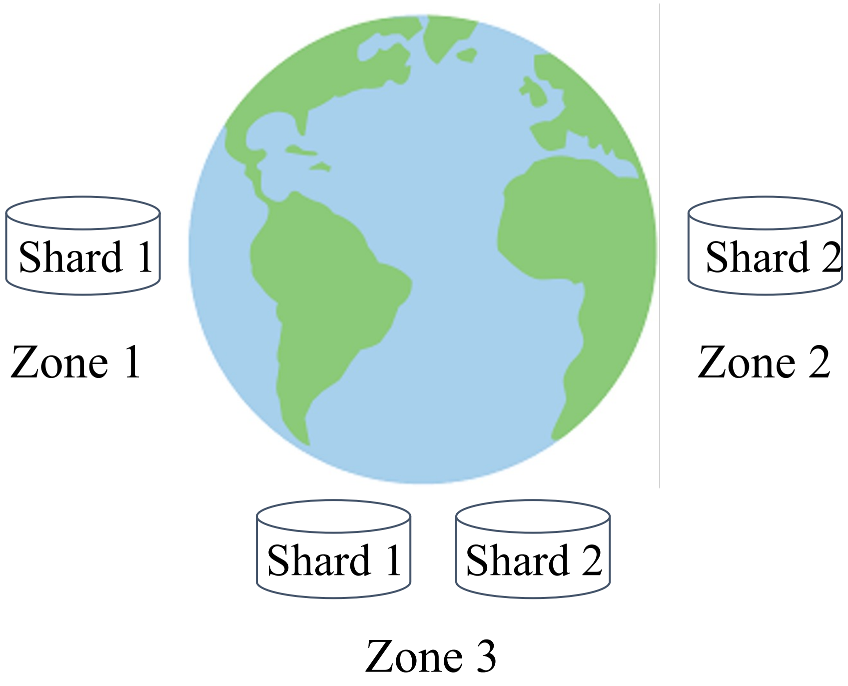


Users(1)
Albums(1,1)
Albums(1,2)
Users(2)
Albums(2,1)
Albums(2,2)
Albums(2,3)

Data model

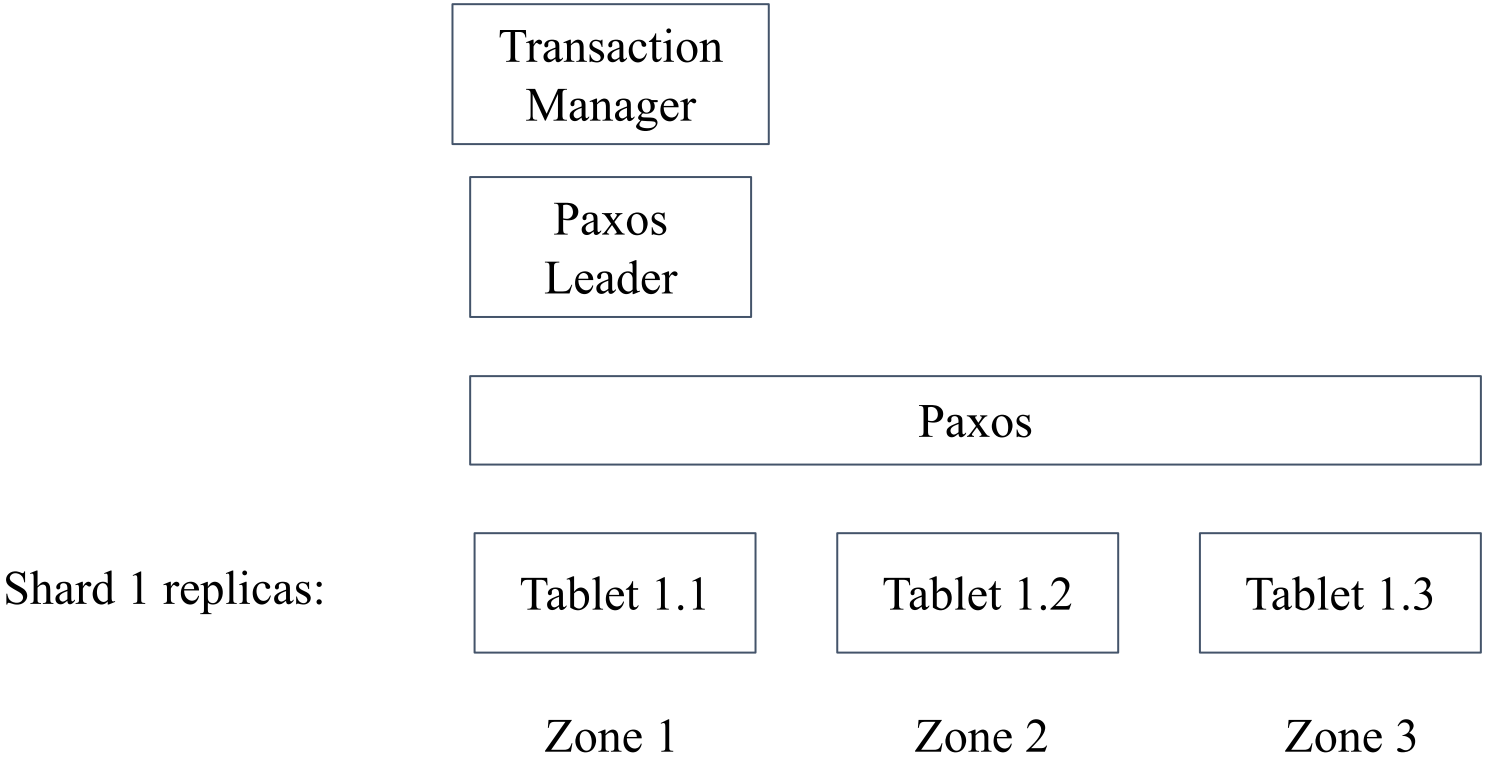
- Each directory/shard is a unit of data movement (e.g., place shard 1 in Zones 1 and 3)


Users (1)
Albums (1, 1)
Users (2)
Albums (2, 1)
Albums (2, 2)
Users (3)
...
Shard 1
Shard 2



Handling replication

- At any time, a Paxos leader runs transactions including locking
- Synchronous replication as long as the majority of replicas is up



Transaction Manager

Paxos Leader

Paxos

Shard 1 replicas:

Tablet 1.1

Tablet 1.2

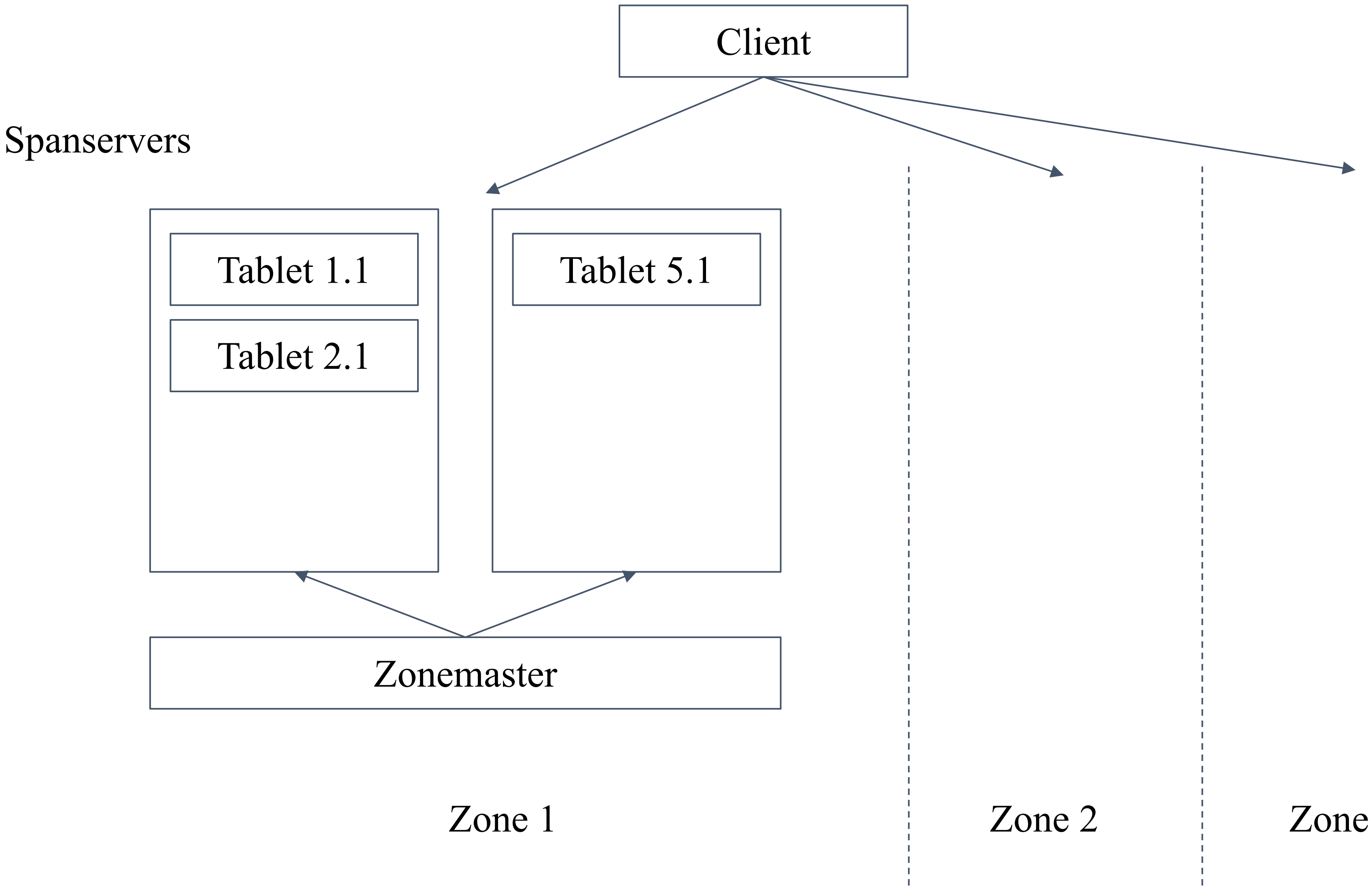
Tablet 1.3

Zone 1

Zone 2

Zone 3

Serving structure



Client

Spanservers

Tablet 1.1

Tablet 2.1

Tablet 5.1

Zonemaster

Zone 1

Zone 2

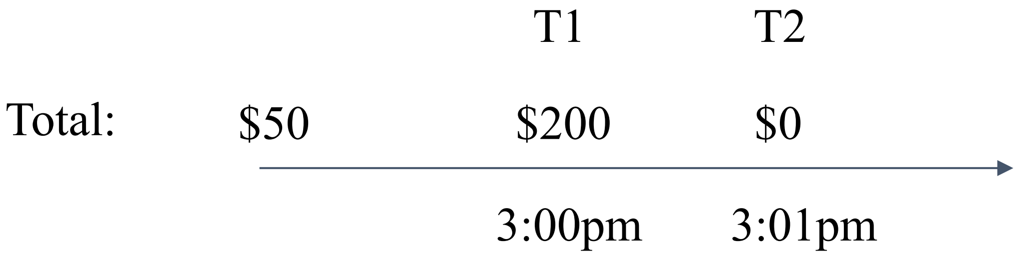
Zone 3

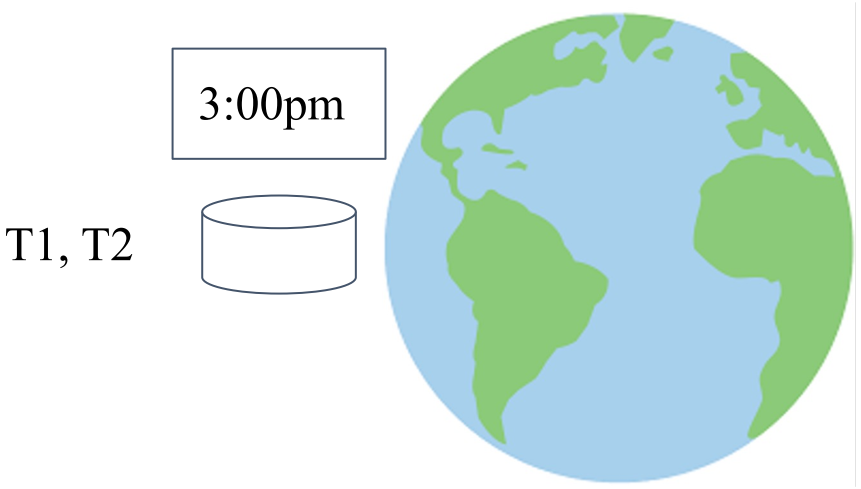
Motivating scenario: banking

- 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
- At the end of the day, any negative balance in one account is covered by the other
- 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

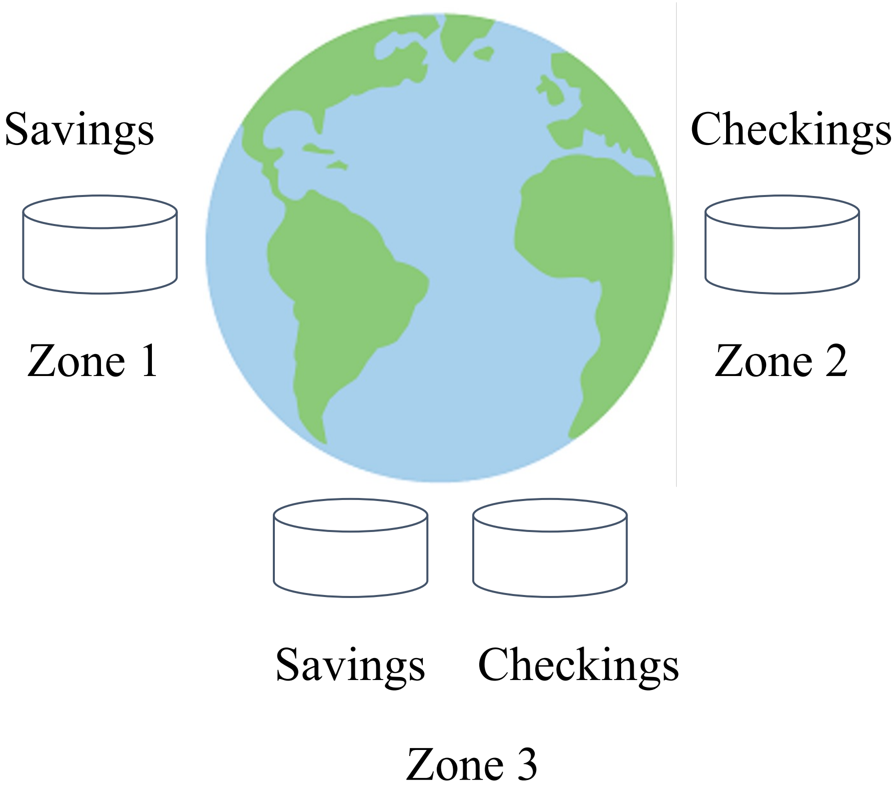
- Give monotonically-increasing timestamps to T1 and then T2
- If another transaction reads the database, use snapshot with most recent timestamp
    - Total balance is never negative





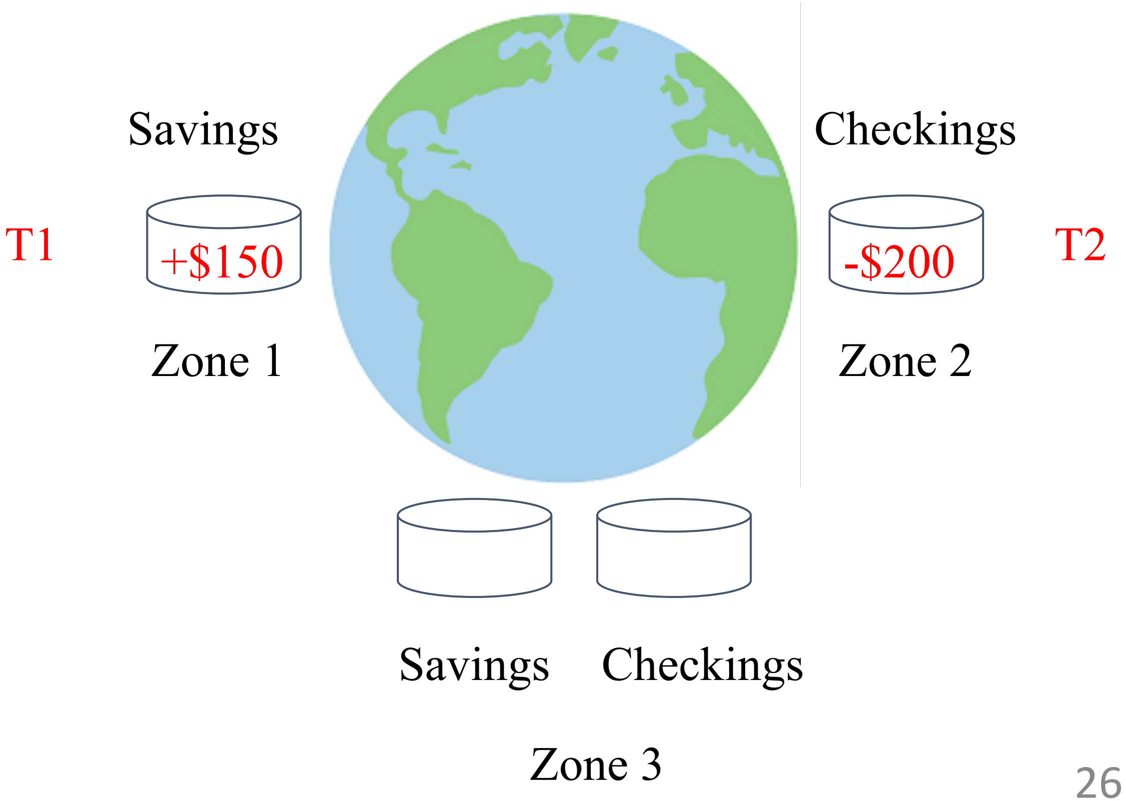
Not easy if database is distributed

- 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



T1

Savings

+$150

Zone 1

Checkings

-$200

Zone 2

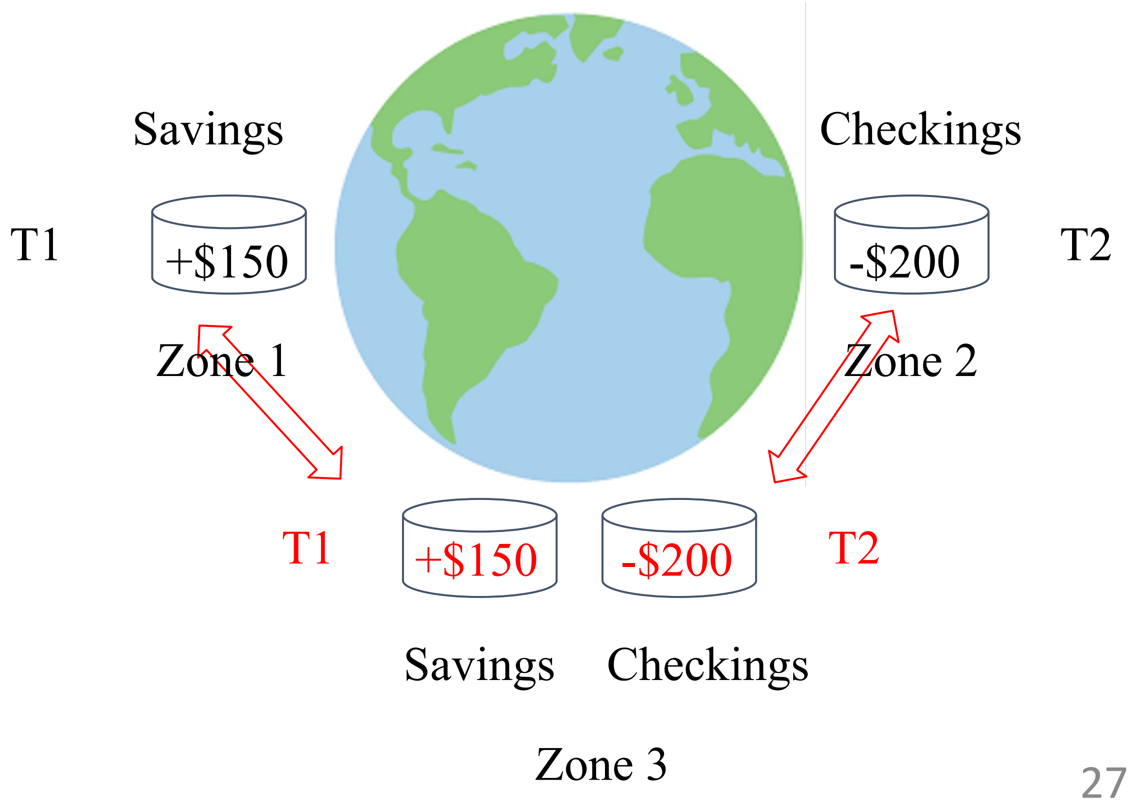
T2

Savings Checkings

Zone 3

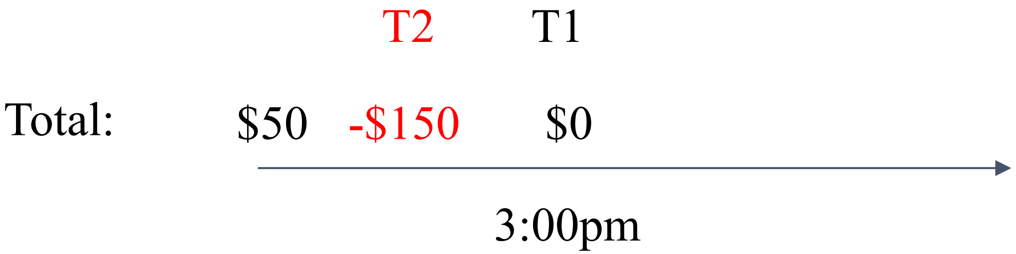
Challenge 1: consistency

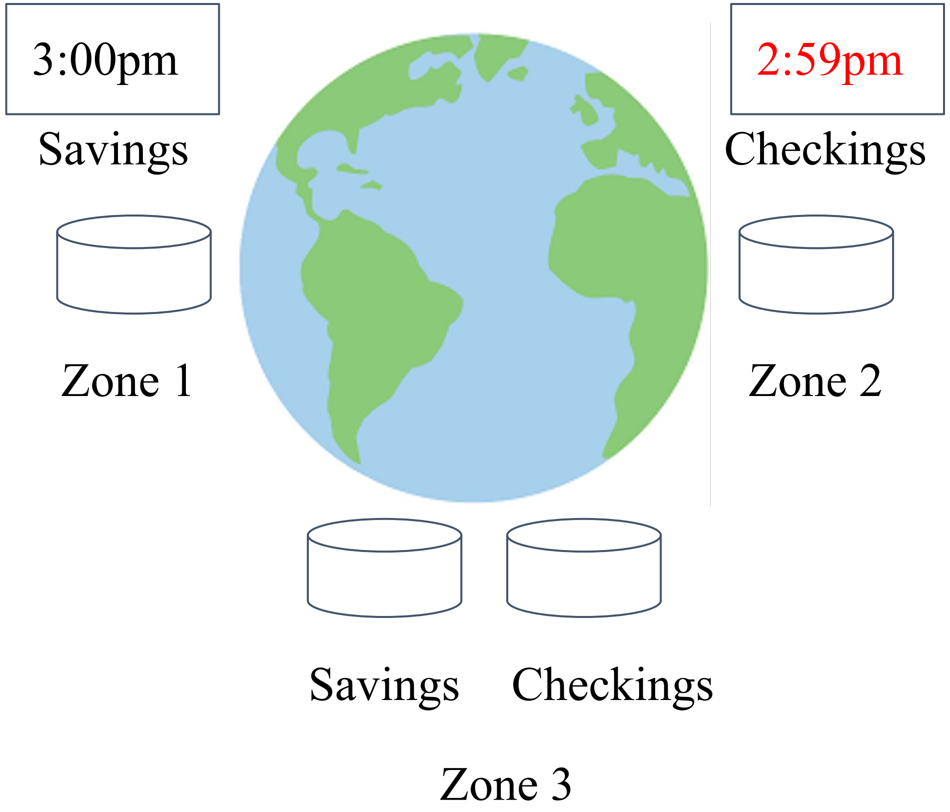
- Need to write on replicas as if there was a single transaction running
- Use existing distributed database techniques
  - Use Paxos algorithm for synchronizing writes
  - Will not go into details



Challenge 2: clock uncertainty

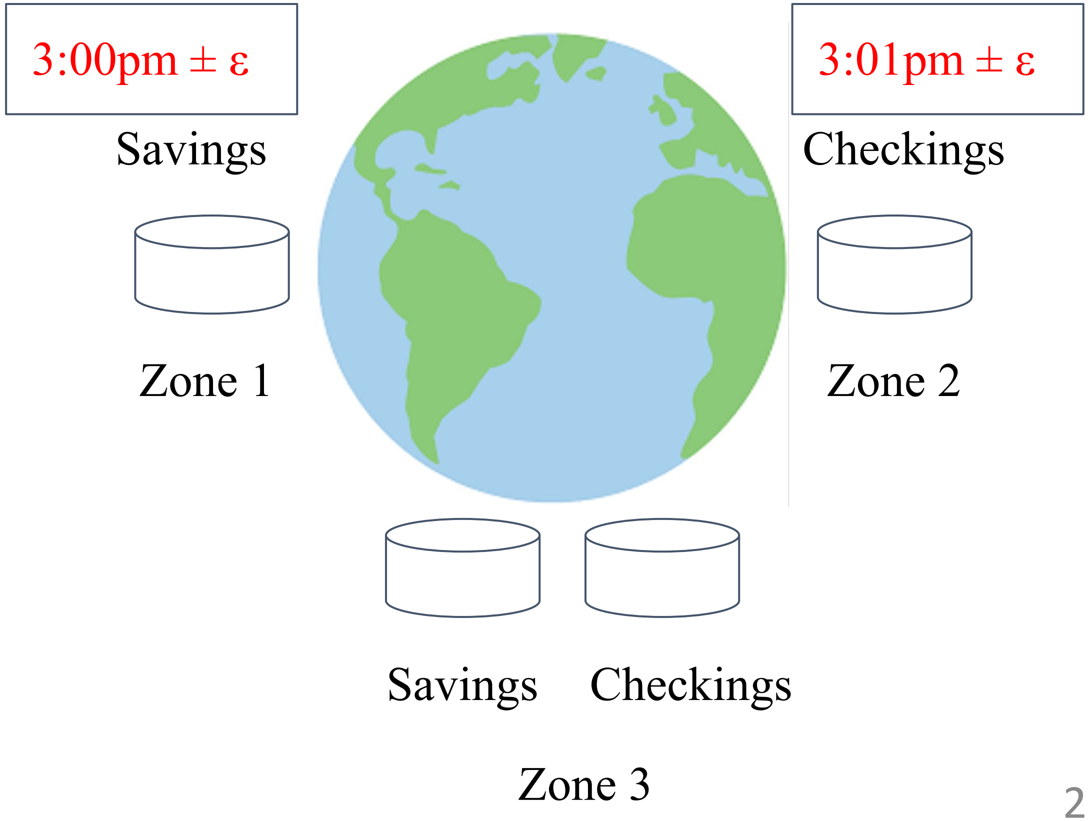
- If clock on the right is slower, then T2 may have a smaller timestamp than T1
- A transaction that reads after T2 sees a negative total balance!

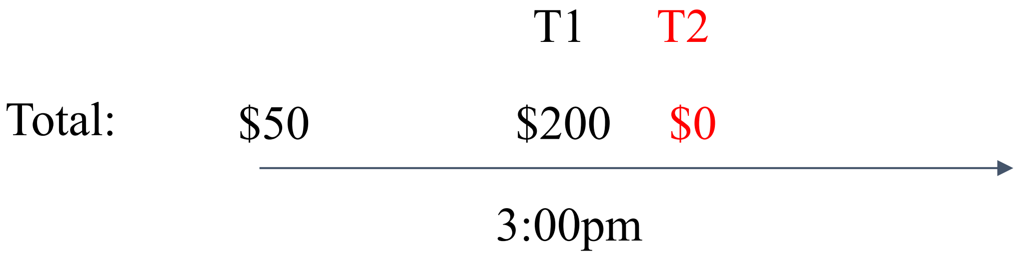




Solution: TrueTime

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





Transactions & Concurrency

- Spanner is designed for *long-lived* transactions
    - E.g., report generation might take a few minutes
- Therefore optimistic concurrency control performs poorly

- Protocol used: strict 2PL

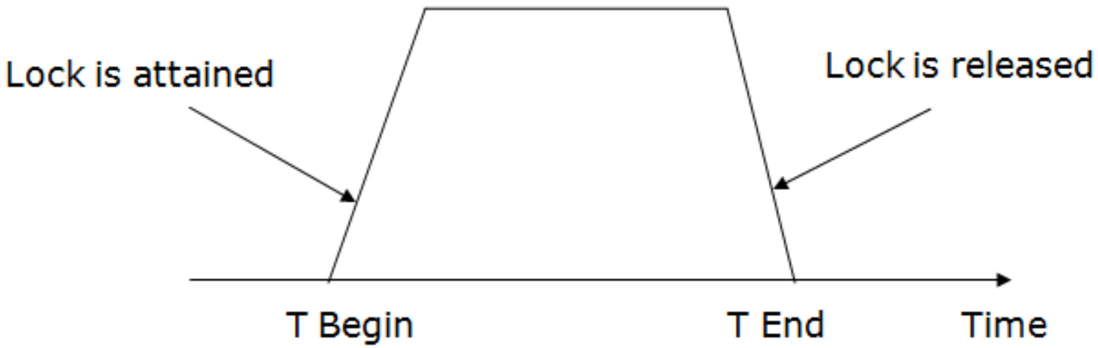
Strict Two-Phase Locking

- Problem of 2PL: Can not avoid cascading aborts
- Example: rollback of T1 requires rollback of T2

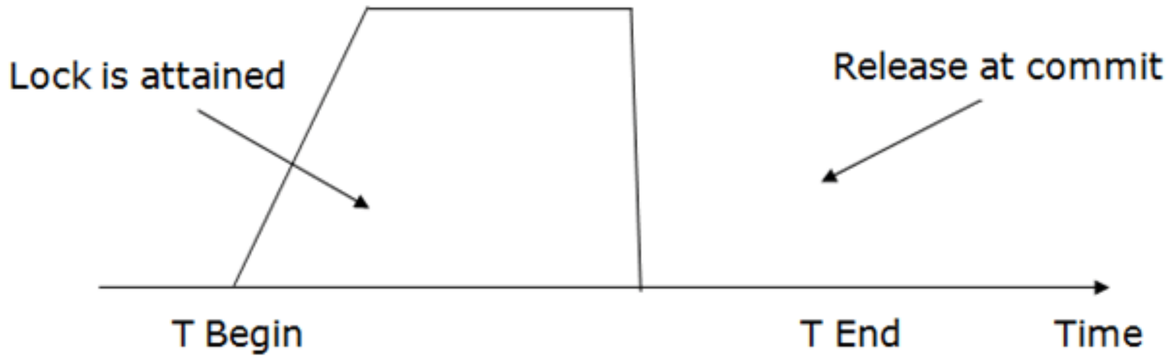
T1:	$r_1(A)$, $w_1(A)$		Abort
T2:		$r_2(A)$, $w_2(A)$	

Strict Two-Phase Locking

- Same as 2PL, except all locks released together when transaction completes:
  - Transaction has committed (all writes durable), OR
  - Transaction has aborted (all writes have been undone)



2PL

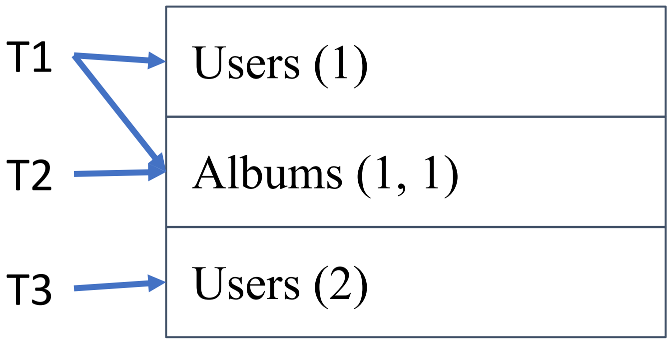


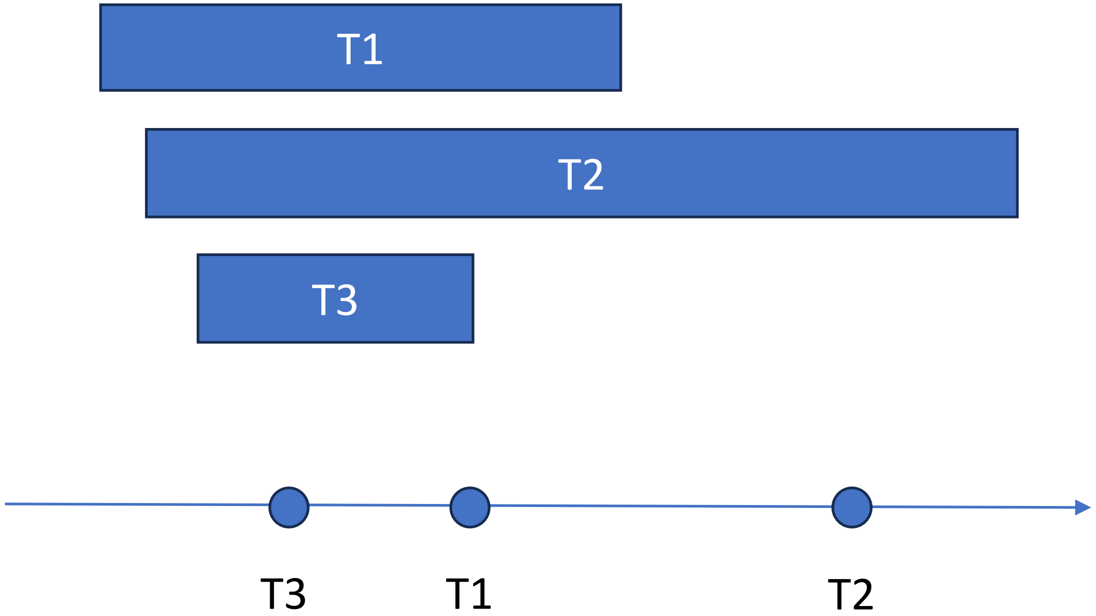
Strict 2PL

Image source: https://www.javatpoint.com/dbms-lock-based-protocol

Strict 2PL Transaction protocol

1. Acquire locks
2. Execute reads
3. Pick commit timestamp
4. Replicate writes using Paxos
5. Ack Commit
6. Apply writes
7. Release locks

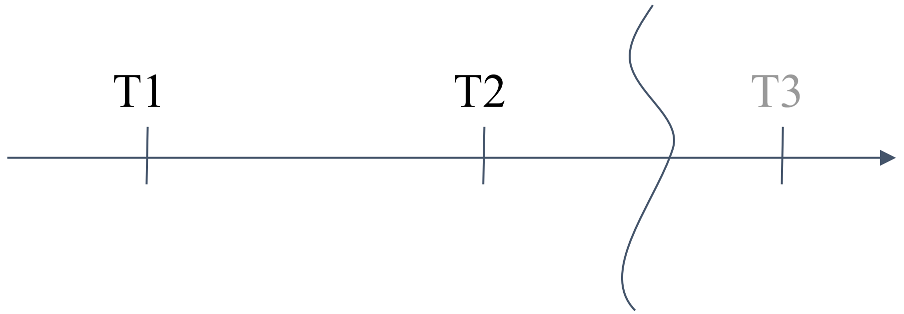




Source: https://www.infoq.com/presentations/spanner-distributed-google/

Multi-version concurrency control

- Reading the most current data will block writes on that data, which is slow
- To read without blocking writes, a classic technique is to use snapshot reads
- A snapshot should contain a prefix of the commit history to make it consistent



A *prefix* of commit history

Challenge: pick commit timestamps

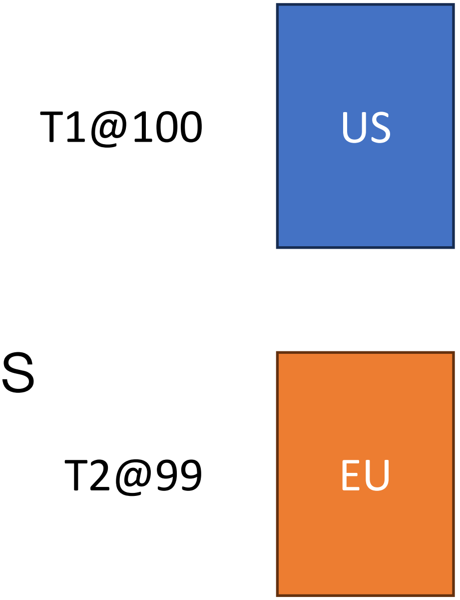
Attempt #1: Assign from local (monotonic) clock

1. Acquire locks
2. Execute reads
3. **Pick commit timestamp = now()**
4. Replicate writes using Paxos
5. Ack Commit
6. Apply writes
7. Release locks

Source: https://www.infoq.com/presentations/spanner-distributed-google/

Example: What goes wrong

- T1 creates a new ad in the *campaign table* on US servers
- Ad serving system notified
- Ad server in Europe
- User clicks on ad
- T2 logs clicks in the impressions table on EU servers



Any snapshot that contains T2 should also contain T1

Source: https://www.infoq.com/presentations/spanner-distributed-google/

Desired property: external consistency

Definition: 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.

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

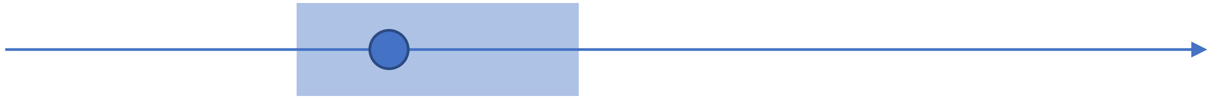
- ○ Same as a traditional database using strict two-phase locking
- ○ System behaves as if all (conflicting) transactions were executed sequentially in one machine

True Time

Idea: There is a global “true” time t

TT.now() = $t \in$ [earliest, latest]

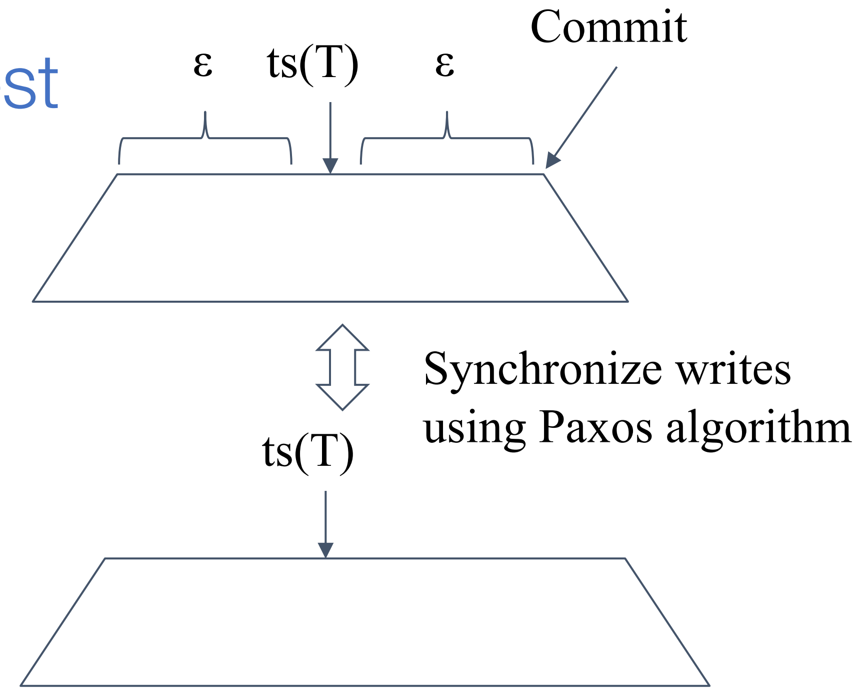
- TT.now().earliest: definitely in the past
- TT.now().latest: definitely in the future



Slide Source: https://www.infoq.com/presentations/spanner-distributed-google/

Transaction protocol

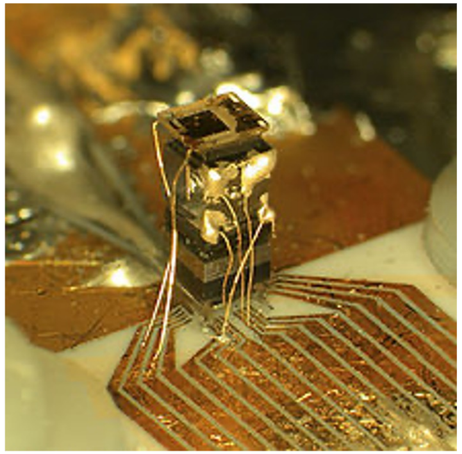
1. Acquire locks
2. Execute reads
3. Pick commit timestamp T = TT.now().latest
4. Replicate writes using Paxos
5. Wait until TT.now().earliest > T
6. Commit
7. Apply write
8. Release locks



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

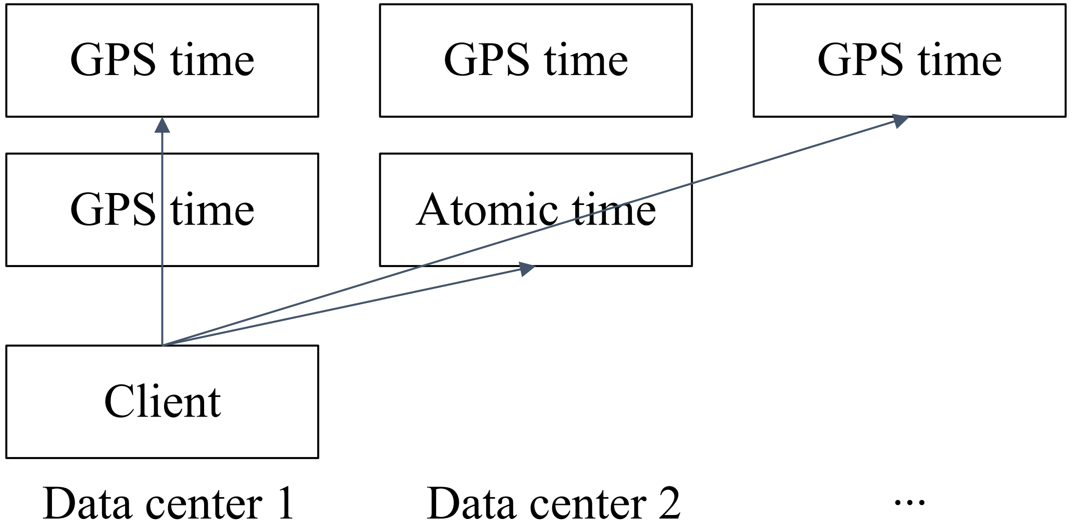
TrueTime implementation

- 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



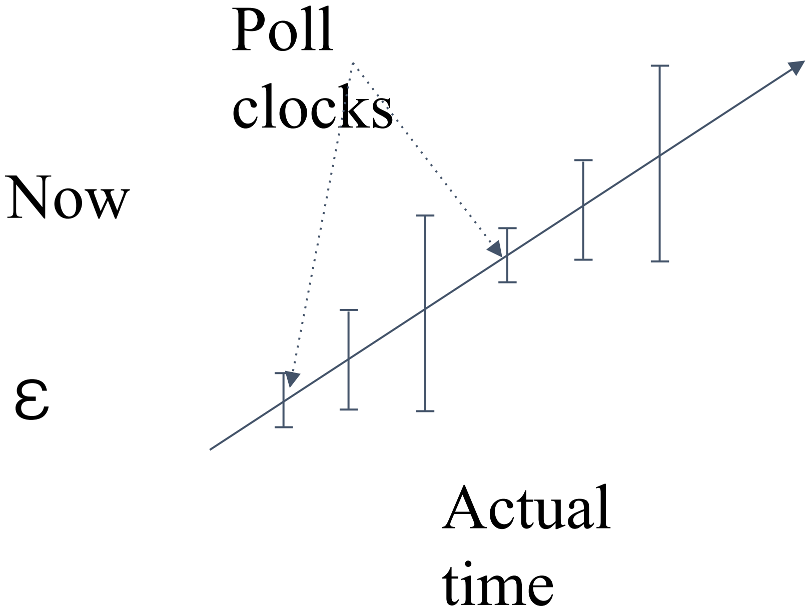
TrueTime implementation

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



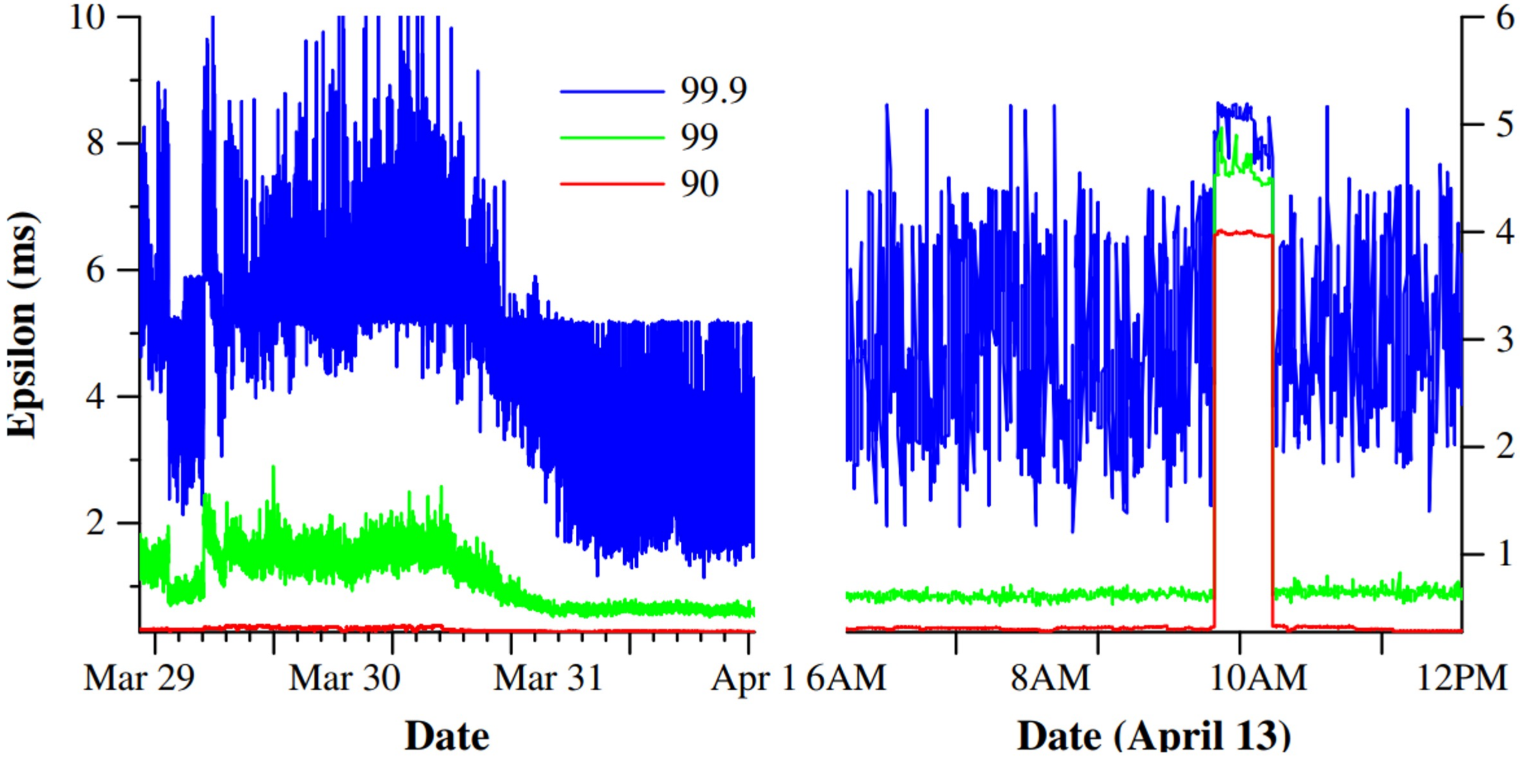
TrueTime implementation

- 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



TrueTime reliability

- 6x more reliable than CPU
- That is, if you trust your computers work, you can trust your clocks as well



Other NewSQL systems

- Novel systems built from ground up
  - Clustrix, CockroachDB, Google Spanner, H-Store, HyPer, MemSQL, NuoDB, SAP HANA, VoltDB
- Middleware that re-implements sharding infrastructure
  - AgilData Scalable Cluster, MariaDB MaxScale, ScaleArc, ScaleBase
- Database-as-a-service
  - Amazon Aurora, ClearDB

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

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
- In the future, there will be a convergence of SQL, NoSQL, and NewSQL