Google

Building Spanner

Better clocks → stronger semantics

Alex Lloyd

Senior Staff Software Engineer

How to build a planet-scale serializable database

Google

Build clocks with bounded absolute error,
and integrate them with timestamp assignment:

- Ensure timestamp total order respects transaction partial order
- Offer efficient serializable queries over everything

Spanner

Google

- Descendant of Bigtable, successor to Megastore
- Scalable, global, Paxos-replicated SQL database
- Geographic partitioning
  - Fluid: online data moves
  - Hidden: no effect on semantics

Spanner: why?

Google

Goal: building rich apps easy at Google scale

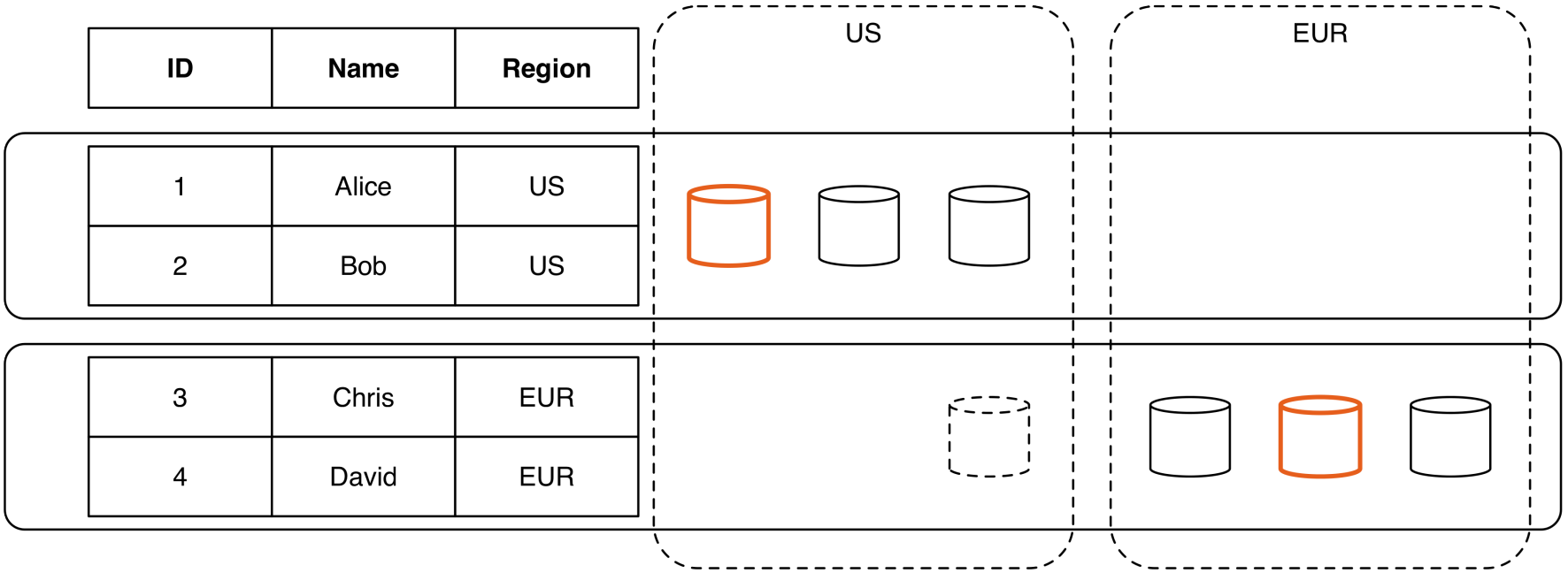
Megastore experience

- Replicated ACID transactions
- Performance, lack of query language, rigid partitioning

Bigtable experience

- Scalability, throughput
- Eventual consistency difficult with cross-entity invariants

Spanner: data model (simplified)



Spanner: physical representation

Google

Customer.ID.1.Name@11 → Alice
Customer.ID.1.Name@10 → Alize
Customer.ID.1.Region@10 → US
Customer.ID.1.Order.ID.100.Product@20 → Camera
Customer.ID.2.Name@5 → Bob

Spanner: concurrency

Google

Default: serializability

- Strict two-phase locking for read-modify-write transactions
  - Big performance hit (two-phase commit) if spans partitions
- Snapshot isolation (no locks) for read-only transactions
  - Small performance hit (timestamp negotiation) if spans partitions

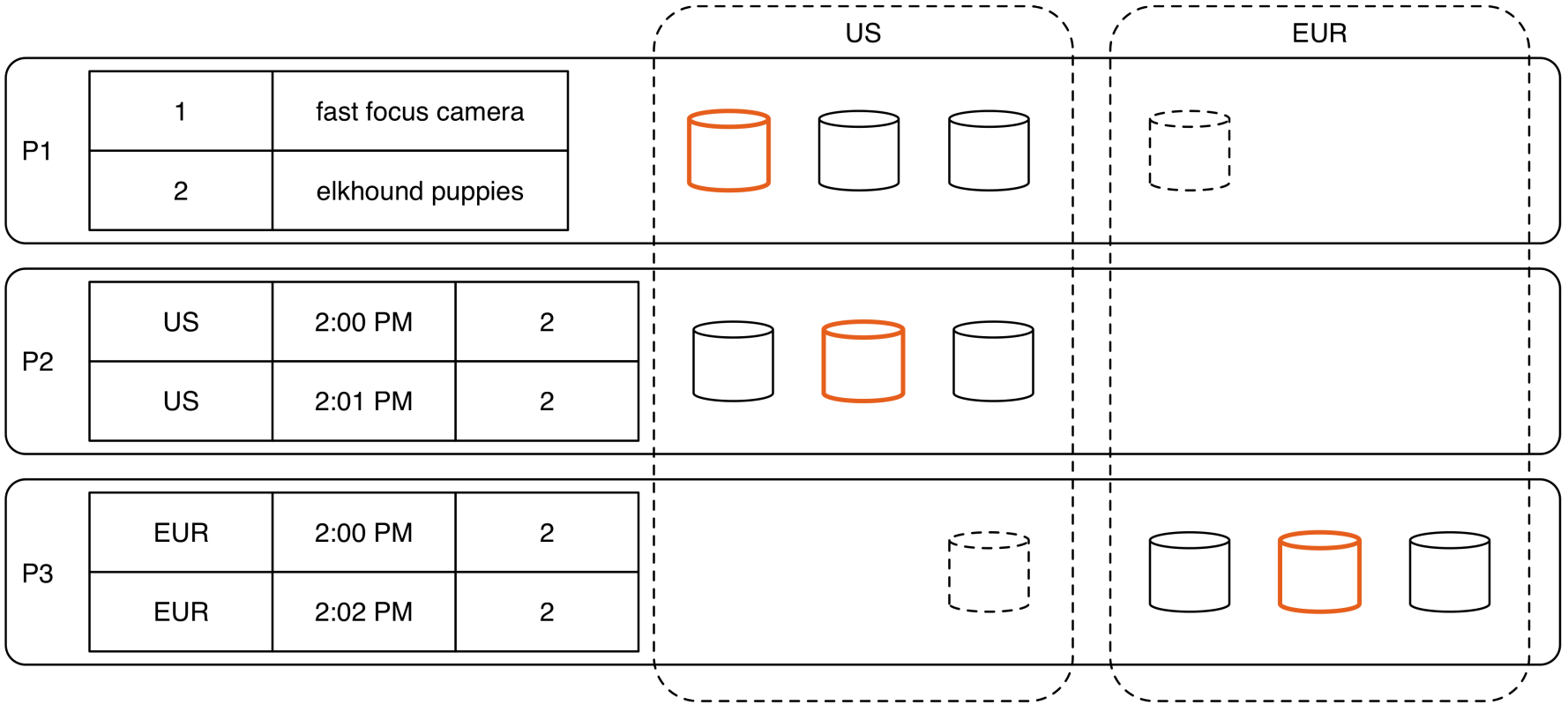
Opt-in: serialize read in the past

- Consistent MapReduce over all data
- Boundedly-stale reads (useful at lagging replicas)

What guarantees do we want?

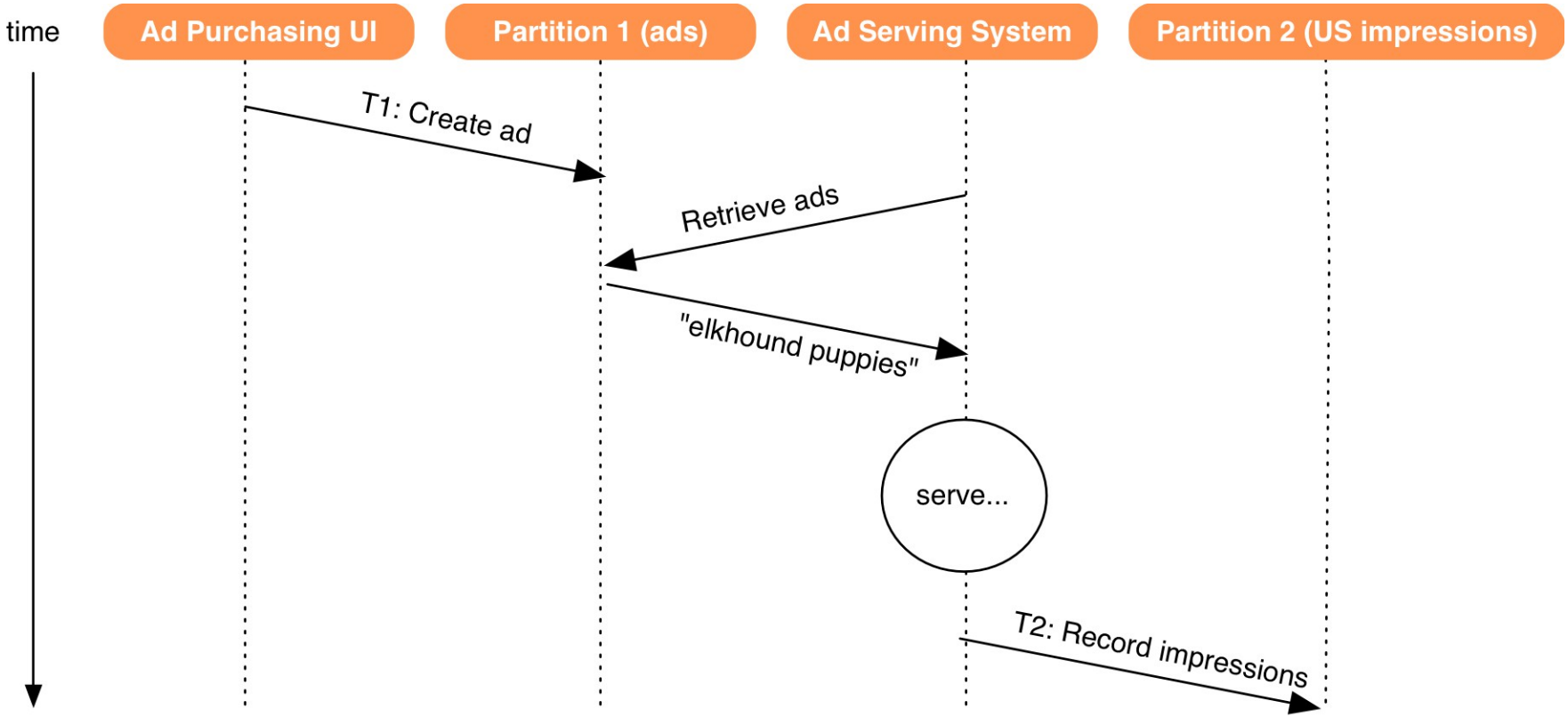
… coming up: how we get them at reasonable cost.

Preserving commit order: example schema



Preserving commit order

Google

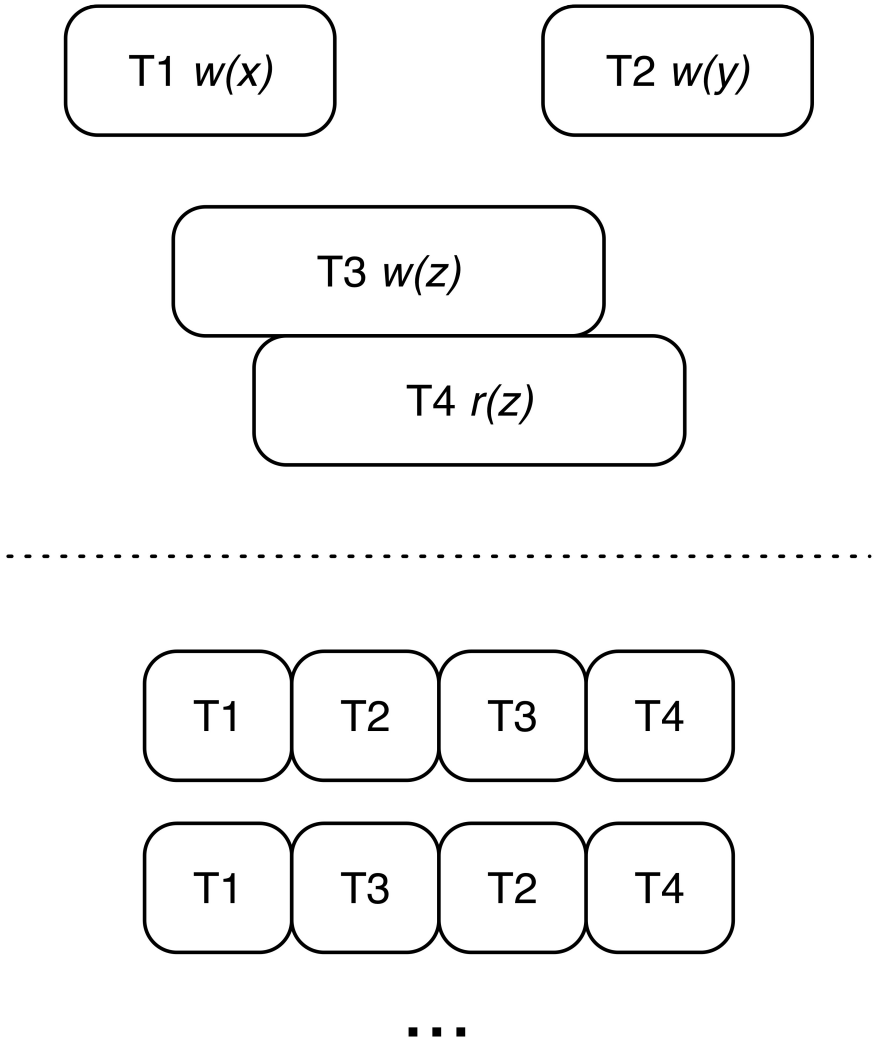


Snapshot MapReduce and queries

Google

Initial state	
T1@ts1	INSERT INTO ads VALUES (2, “elkhound puppies”)
T2@ts2	INSERT INTO impressions VALUES (US, 2PM, 2)

Legal transaction orderings



Linearizability (multiprocessing term)

Google

Equivalent to some serial order

Can't commute commit order: system preserves *happens-before* relationship among transactions

- even when there's no detectable dependency
- even across machines

Options for Scaling

Google

Lots of WAN communication

- Include all partitions in every transaction
- Centralized timestamp oracle

No extra communication

- Propagate timestamp through every external system & protocol (Lamport clocks)
- Distributed timestamp oracle

Options for Scaling

Google

Lots of WAN communication

- Include all partitions in every transaction
- Centralized timestamp oracle

No extra communication

- Propagate timestamp through every external system & protocol (Lamport clocks)
- Distributed timestamp oracle
  - TrueTime: now() = {time, epsilon} derived from GPS, backed up by atomic oscillators

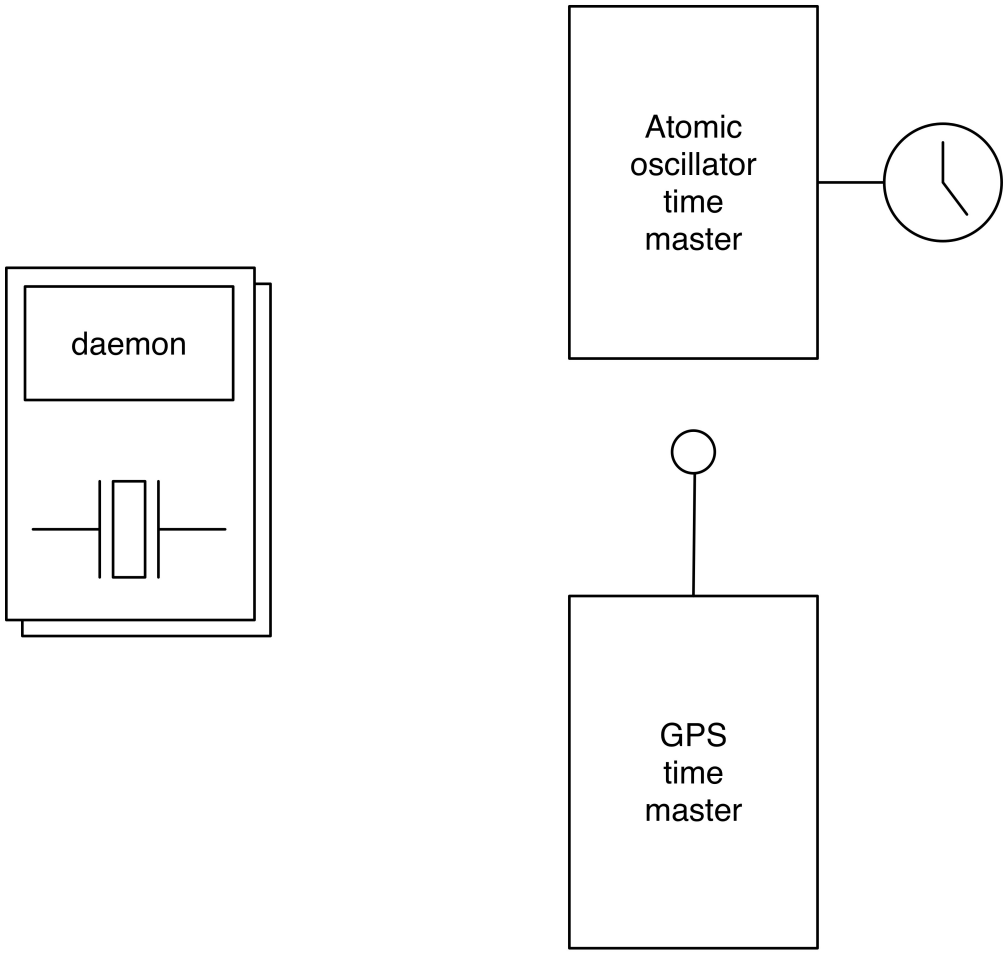
What guarantees do we want,

… and how we get them.

Celestial navigation

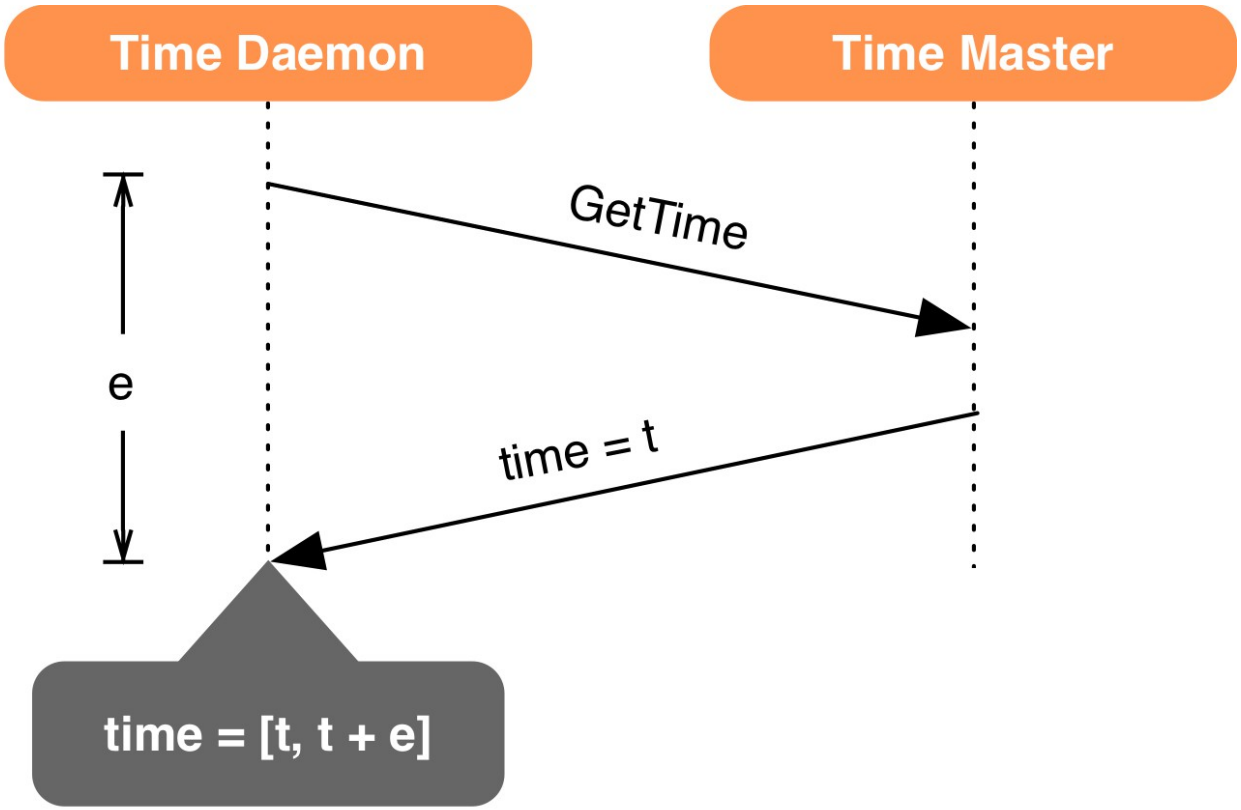
TrueTime

Google



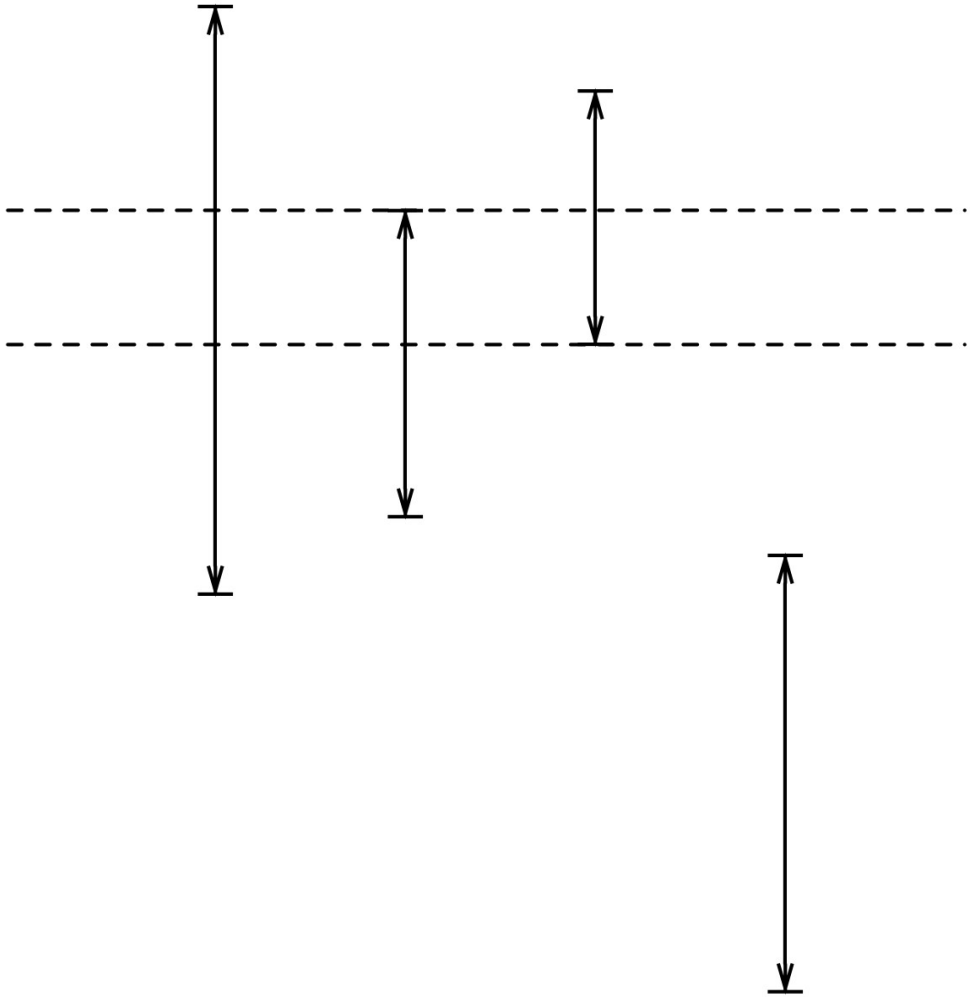
TrueTime

Google



TrueTime: Marzullo's algorithm (also used in NTP)

Google

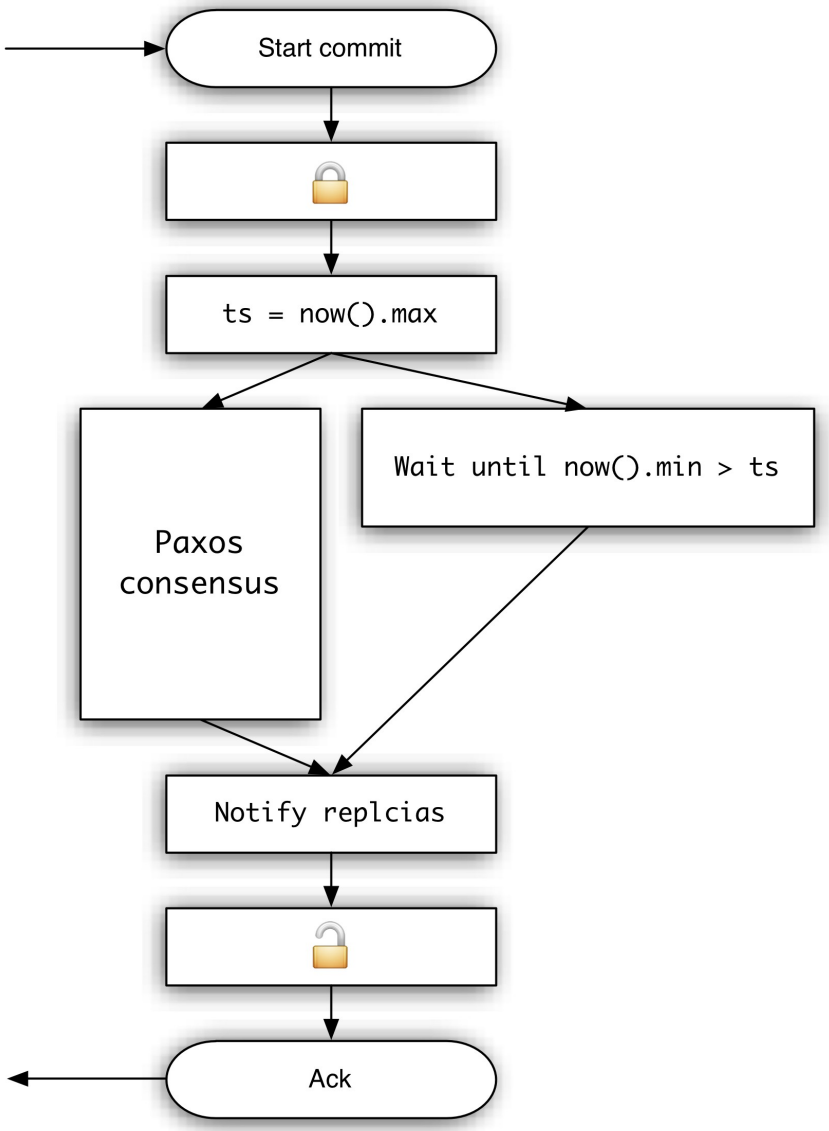


TrueTime → write timestamps

Google

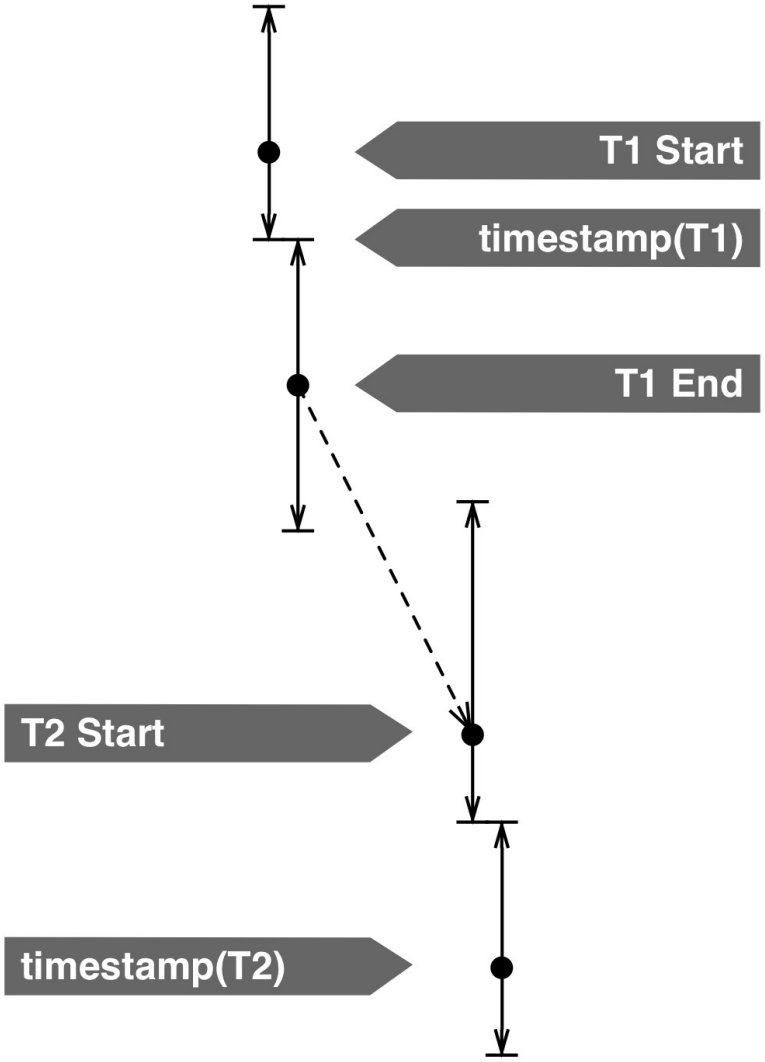
- Given write transactions A and B, if A *happens-before* B, then timestamp(A) < timestamp(B) even if A and B have no partitions in common.
- A *happens-before* B if its effects become visible before B begins, in real time.
  - Visible means acked to client, or updates applied at some replica.
  - Begins means first request arrived at Spanner server.
- Ensures serializability of future snapshot reads at arbitrary timestamps.

TrueTime → write timestamps

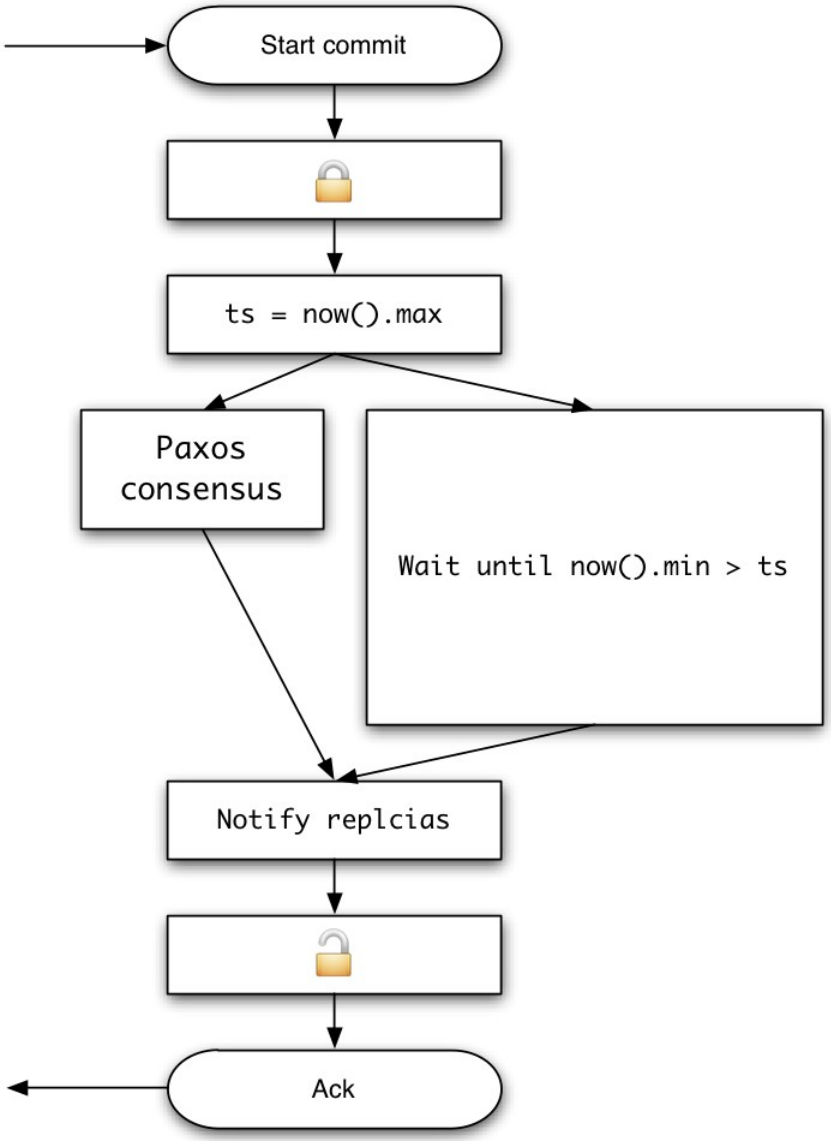


Why this works

Google



When this costs something



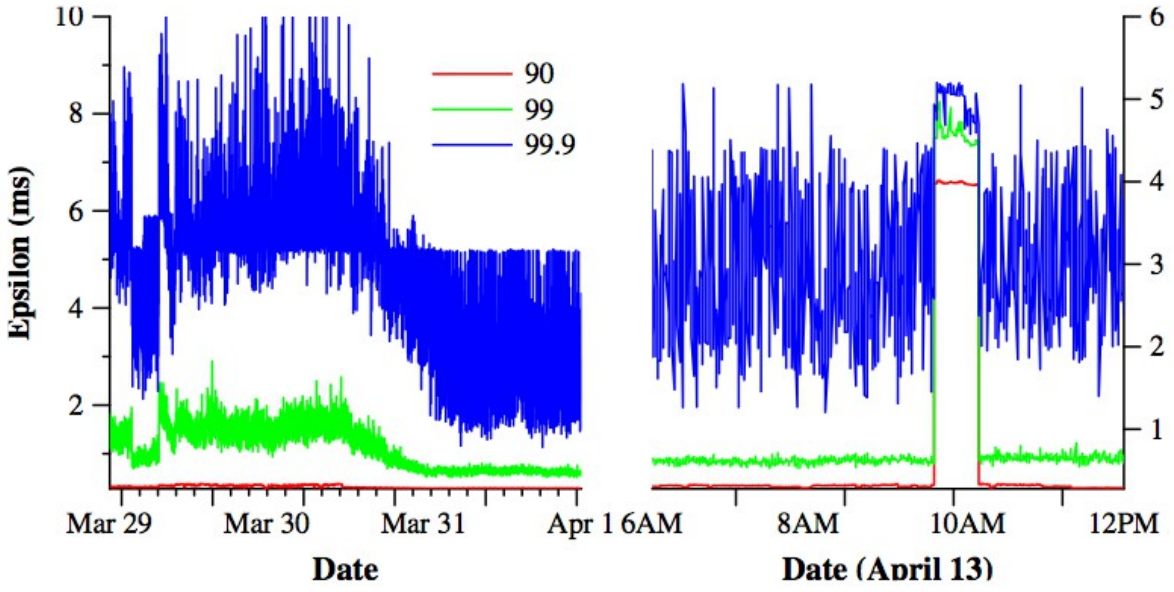
TrueTime epsilon

Google

Sawtooth function from 1-7ms in existing system

Slope: oscillator error assumptions

Minimum: latency to time masters



Reducing TrueTime epsilon

Google

Poll time masters more often (currently every 30s)

Poll at high QoS

- Must enforce even in kernel

Record timestamps in NIC driver

Buy better oscillators

… and watch out for kernel bugs!

- Spanner: distributed database
- Concurrency properties: linearizability
- TrueTime: GPS and atomic oscillators
- TrueTime intervals → write timestamps
- **So how do we read?**

Kinds of read

Google

- Within read-modify-write
  - Acquire locks in lock manager at Paxos leader(s)
- “Strong” reads
  - Spanner picks timestamp, reads at timestamp
- Boundedly-stale reads
  - Spanner picks largest committed timestamp, within staleness bounds
- MapReduce / batch read
  - Client picks timestamp

Timestamps for strong read

Google

Using TrueTime

- timestamp = now().max

Using commit history

- Remember commit timestamps from recent writes
- Must declare “scope” up front
  - trivial for stand-alone queries
  - or, “orders from user alloyd”
- Complicated by prepared distributed transactions

Principles for effective use

Google

Still design schema for data locality

- Example: try to put customer and orders in same partition; big users span partitions

Design app for correctness

Relax semantics for carefully audited high-traffic queries

First big user: F1

Google

Migrated revenue-critical sharded MySQL instance to Spanner

Substantial influence on Spanner data model

Slides from SIGMOD 2012 talk online

Evolution of data model

Google

1. Distributed filesystem metaphor; directory was unit of geographic placement
2. Added structured keys to directory and filenames
3. Made Spanner a hierarchical “store for protocol buffers”

   (Meanwhile, started work on SQL engine)
4. Watched F1 build relational schemas atop Spanner → moved to a relational data model

Examples of ongoing work

Google

Polishing SQL engine

- Restartable SQL queries across server versions (!)

Hardening

- Finer control over memory usage
- Finer-grained CPU scheduling

SI-based “strong” reads

Scaling to large numbers of replicas per Paxos group (partition)

Thanks!

Questions?