Spanner: Google's Globally-Distributed Database

James C. Corbett, **Jeffrey Dean**, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman,Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh,Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura,David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak,Christopher Taylor, Ruth Wang, Dale Woodford

OSDI 2012

Courtesy

Md. Jahidul islam

Md. Nurul Alam

What is Spanner?

Spanner is Google's scalable, multi-version, globally-distributed, and synchronously-replicated database.

It is the first system to distribute data at global scale and support externally-consistent distributed transactions.

Running in the production:

- Storage for Google's Ad data
- Replaced a sharded MySQL database

Google

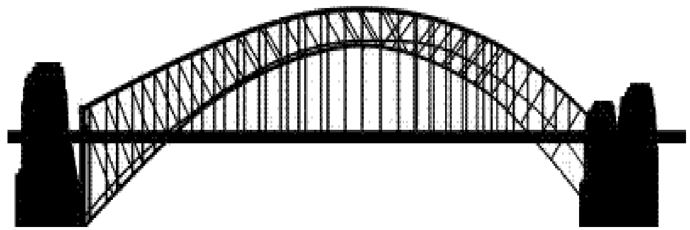
- Distributed multi-version database
  - General-purpose transactions
  - SQL query language
  - Schematized tables
  - Semi-relational data model

- Focus: managing cross-datacenter replication
  - Replication can be controlled by clients for load balancing and failure responses.

Why Spanner ?

- **Bigtable (2008):**
  - Difficult to use for complex, evolving schemas
  - Can’t give strong consistency guarantees for geo-replicated sites
- **NoSQL(2010):**
  - Similar problem as Bigtable.
- **Megastore (2011):**
  - It supports semi-relational data model.
  - synchronous replication
  - But poor write throughput

Google
Megastore



Google Bigtable

Solution: Google Spanner

- Bridging the gap between Megastore and Bigtable.
- SQL transactions + high throughput
- lower latency over higher availability

Google

Spanner Overview

- Bigtable-like versioned key-value store into a temporal multi-version database.
- Data is stored in schematized semi-relational tables.
- data is versioned,
- each version is automatically timestamped with its commit time;
- old versions of data are subject to configurable garbage-collection policies;
- applications can read data at old timestamps.
- supports general-purpose transactions,
- provides a SQL-based query language.

Google

Features of Spanner

- The replication configurations for data can be dynamically controlled at a fine grain by applications.
- It provides externally consistent reads and writes .
- It provides globally-consistent reads across the database at a timestamp.
  - These features enable Spanner to support consistent backups, consistent MapReduce executions [12], and atomic schema updates, all at global scale, and even in the presence of ongoing transactions.

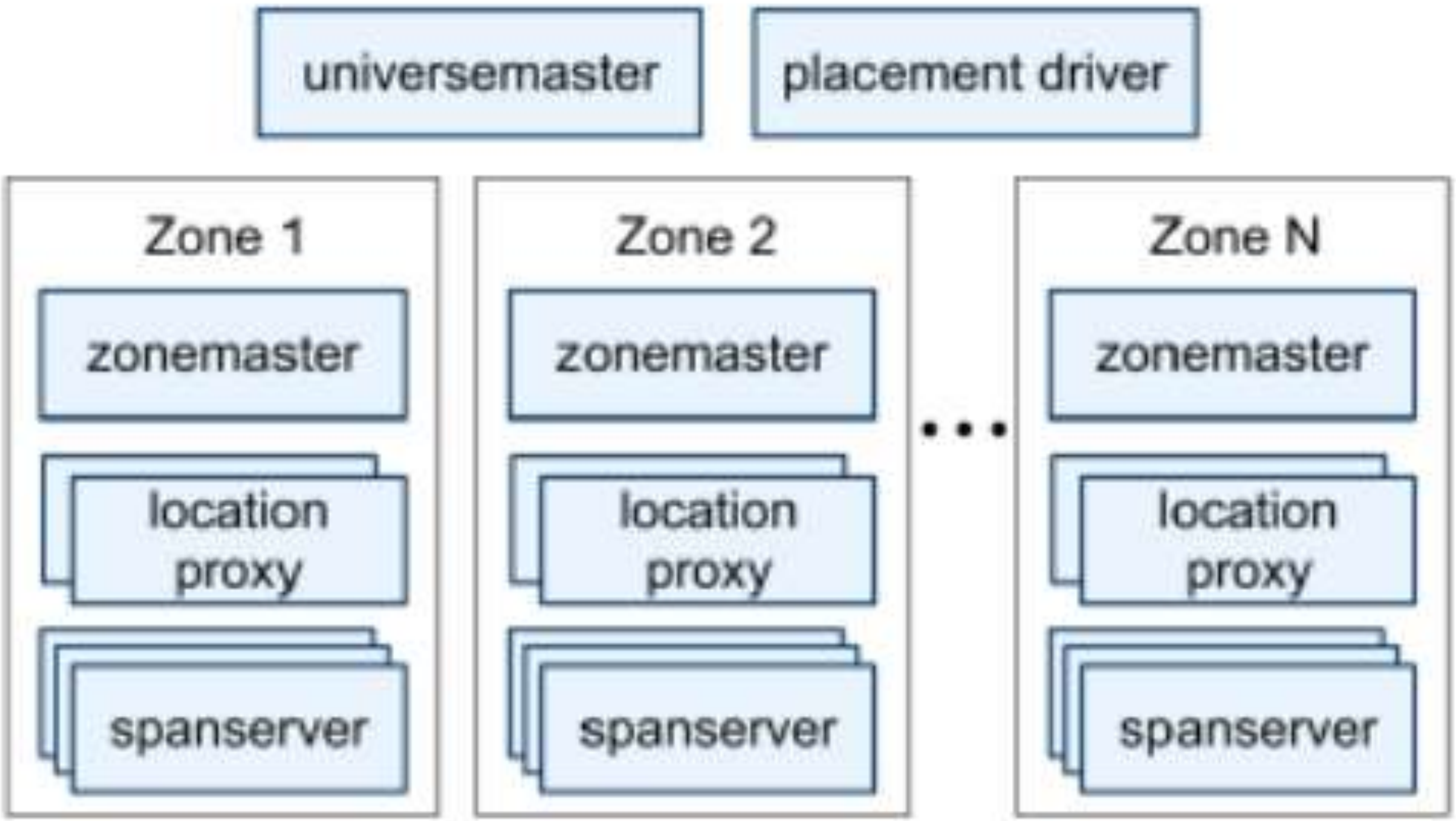
Replication configurations

- dynamically controlled at a fine grain by applications
- control which datacenters contain which data,
- how far data is from its users (to control read latency),
- how far replicas are from each other (to control write latency), and
- how many replicas are maintained (to control durability, availability, and read performance).

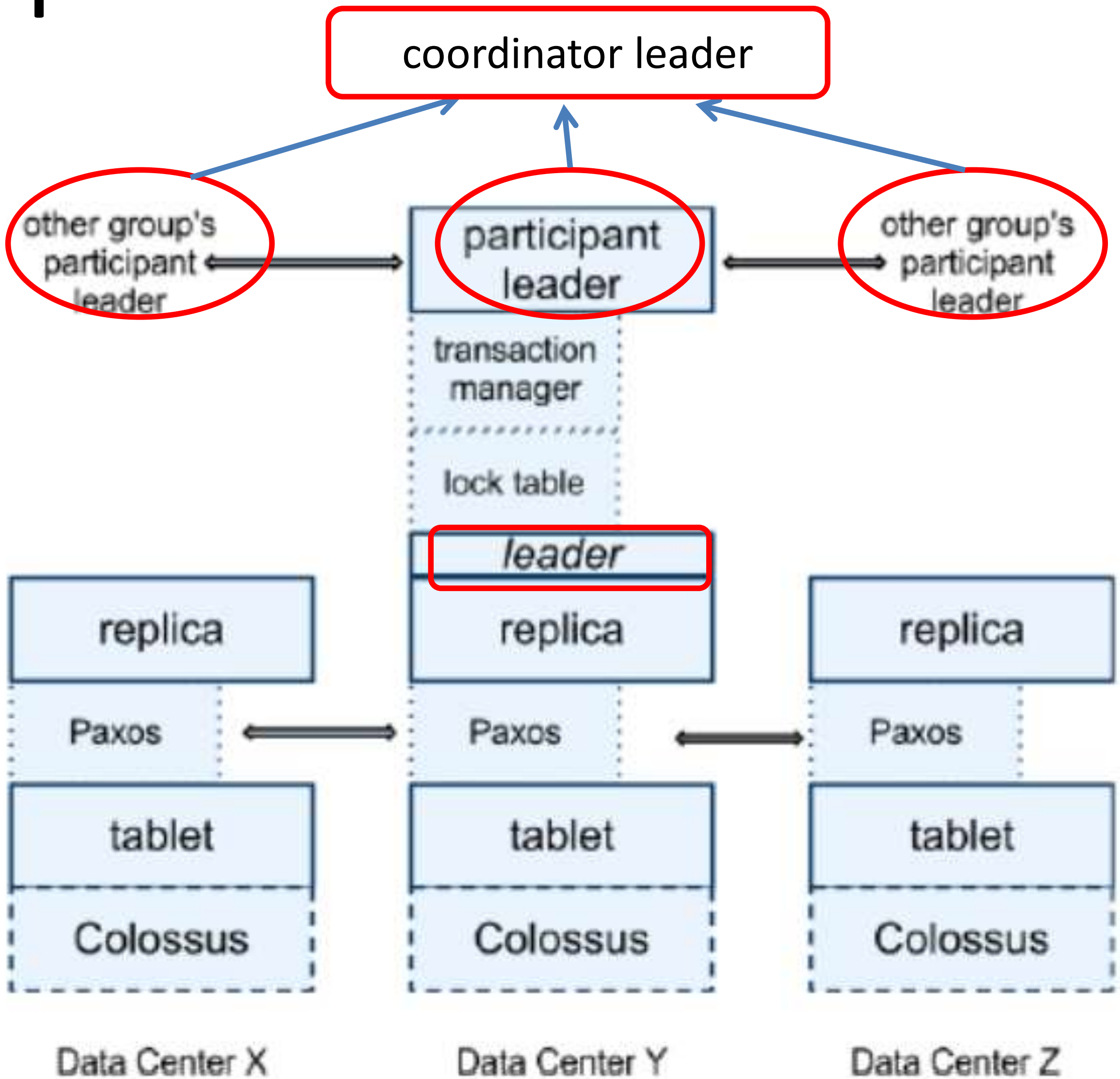
Replication configurations

- Data can also be dynamically and transparently moved between datacenters by the system to balance resource usage across datacenters.

Spanner server organization



Spanserver software stack



Directories and Placement

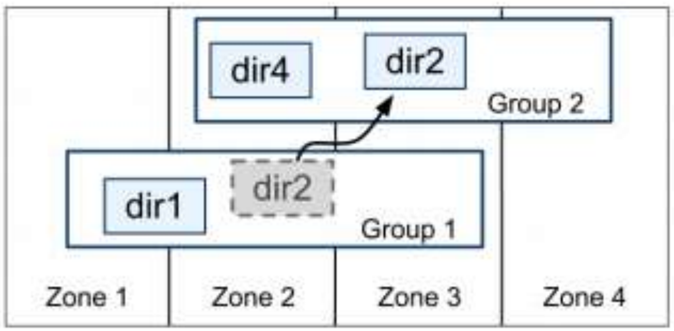


Figure 3: Directories are the unit of data movement between Paxos groups.

- A directory is the unit of data placement.

- Spanner might move a directory to shed load from a Paxos group;
- Put directories that are frequently accessed together into the same group;
- Move a directory into a group that is closer to its accessors;

Data Model

- An application creates one or more *databases* in a universe.
- Each database can contain an unlimited number of schematized *tables*.
- Tables look like relational-database tables, with rows, columns, and versioned values.

Data Model

- Schematized semi-relational tables

BigTable ： (row:string, column:string, time:int64)→string

Spanner ： (key:string, timestamp:int64) → string

- **query language like sql**

```
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;
```

TrueTime API

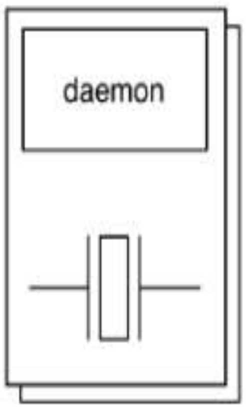
Method	Returns
$TT.now()$	$TTinterval$: $[earliest, latest]$
$TT.after(t)$	true if t has definitely passed
$TT.before(t)$	true if t has definitely not arrived

Table 1: TrueTime API. The argument t is of type *TTstamp*.

- Google’s cluster-management software provides an implementation of the TrueTime API.
- This implementation keeps uncertainty small (generally less than 10ms) by using multiple modern clock references (GPS and atomic clocks).

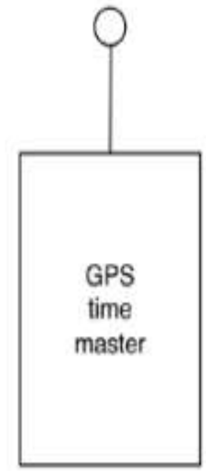
How TrueTime Is Implemented? (1/2)

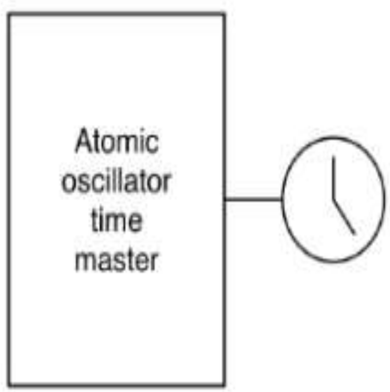
timeslave daemon per **machine**



set of time **master machines per datacenter**

majority of masters have
GPS receivers
with dedicated antennas





The remaining masters (which we refer to as **Armageddon masters**) are equipped with **atomic clocks.**

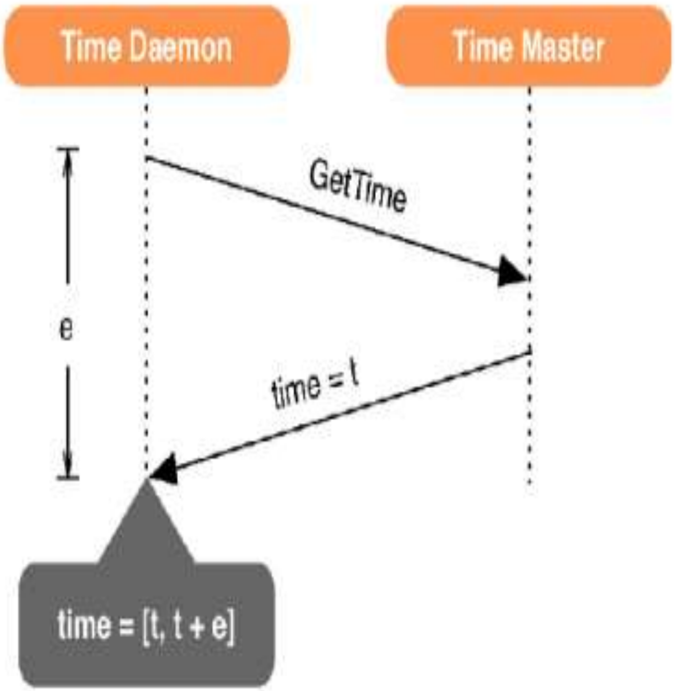
How TrueTime Is Implemented? (2/2)

➢ Daemon polls variety of masters:

• Chosen from nearby datacenters

• From further datacenters

• Armageddon masters



➢ Daemon polls variety of masters and reaches a consensus about correct timestamp.

Google

Synchronizing Snapshots

Global wall-clock time

==

External Consistency:

Commit order respects global wall-time order

==

Timestamp order respects global wall-time order

given

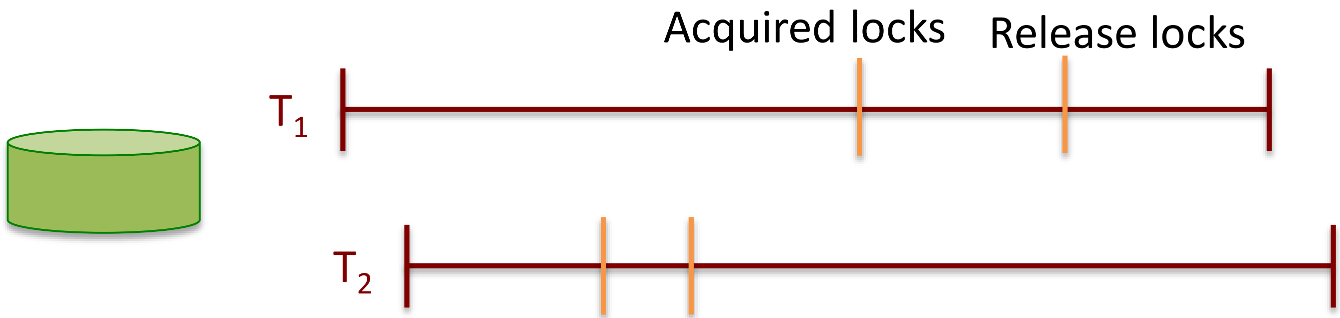
timestamp order == commit order

Timestamps, Global Clock

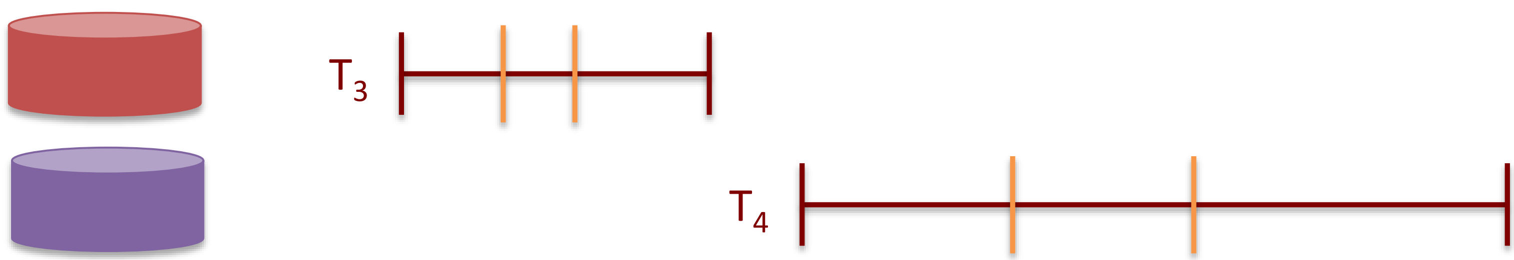
- Strict **two-phase locking** for write transactions
- Assign timestamp while locks are held

Timestamp Invariants

- Timestamp order == commit order



- Timestamp order respects global wall-time order



Types of Reads in Spanner

Operation	Timestamp Discussion	Concurrency Control	Replica Required
Read-Write Transaction	§ 4.1.2	pessimistic	leader
Read-Only Transaction	§ 4.1.4	lock-free	leader for timestamp; any for read, subject to § 4.1.3
Snapshot Read, client-provided timestamp	—	lock-free	any, subject to § 4.1.3
Snapshot Read, client-provided bound	§ 4.1.3	lock-free	any, subject to § 4.1.3

Table 2: Types of reads and writes in Spanner, and how they compare.

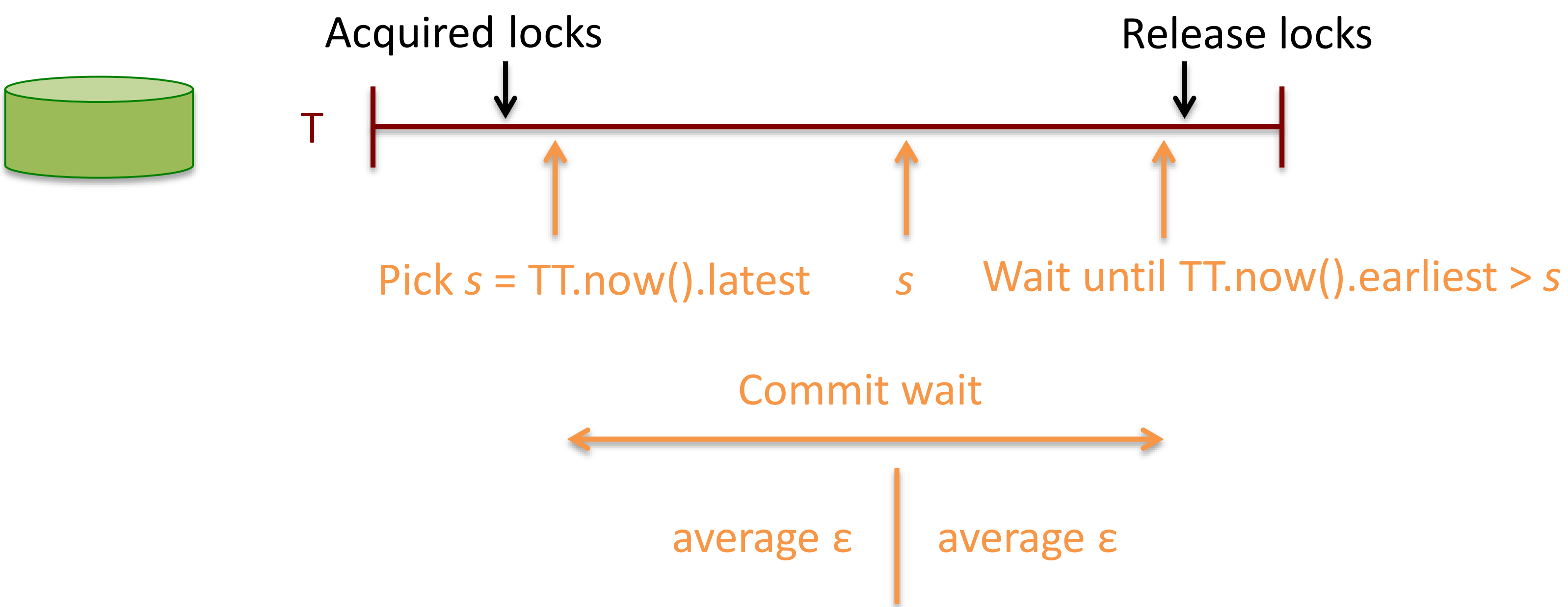
Google

Timestamps and TrueTime

- **Two rules:**

1. **Start:** s_i for T_i > TT.now.latest() computed after e_i^{server} (arrival event at leader)

2. **Commit wait:** Clients should not see data committed by T_i until TT.after(s_i) is true
   $s_i < t_{abs}(e_i^{commit})$



Snapshot reads

– Read in past without locking

– Client can specify timestamp for read or an upper bound of timestamp's staleness

– Each replica tracks a value called safe time t_{safe} which is the maximum timestamp at which a replica is up-to-date.

– Replica can satisfy read at any $t <= t_{safe}$

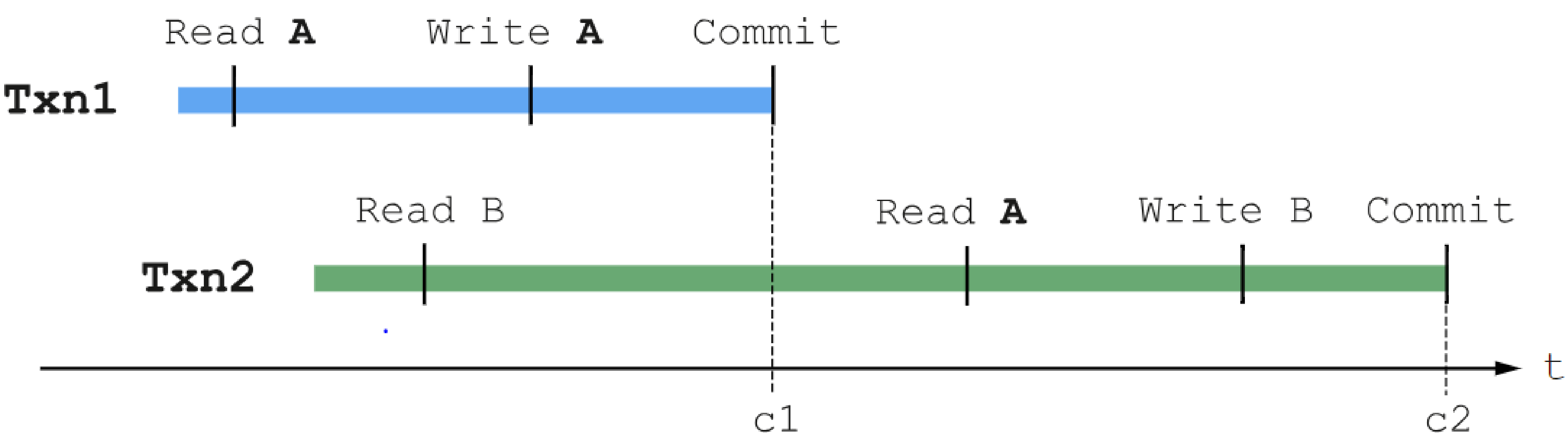
Read-only transaction

- Assign timestamp s_{read} and executes read at s_{read}
- $s_{read} = TT.now().latest()$ guarantees external consistency
- For read at single paxos group:
  - Let LastTS() = timestamp of the last committed write at the Paxos group.
  - If there are no prepared transactions, the assignment s_{read} = LastTS() trivially satisfies external consistency: the transaction will see the result of the last write
- For read at multiple paxos group:
  - $s_{read} = TT.now().latest()$ [may wait for safe time to advance]

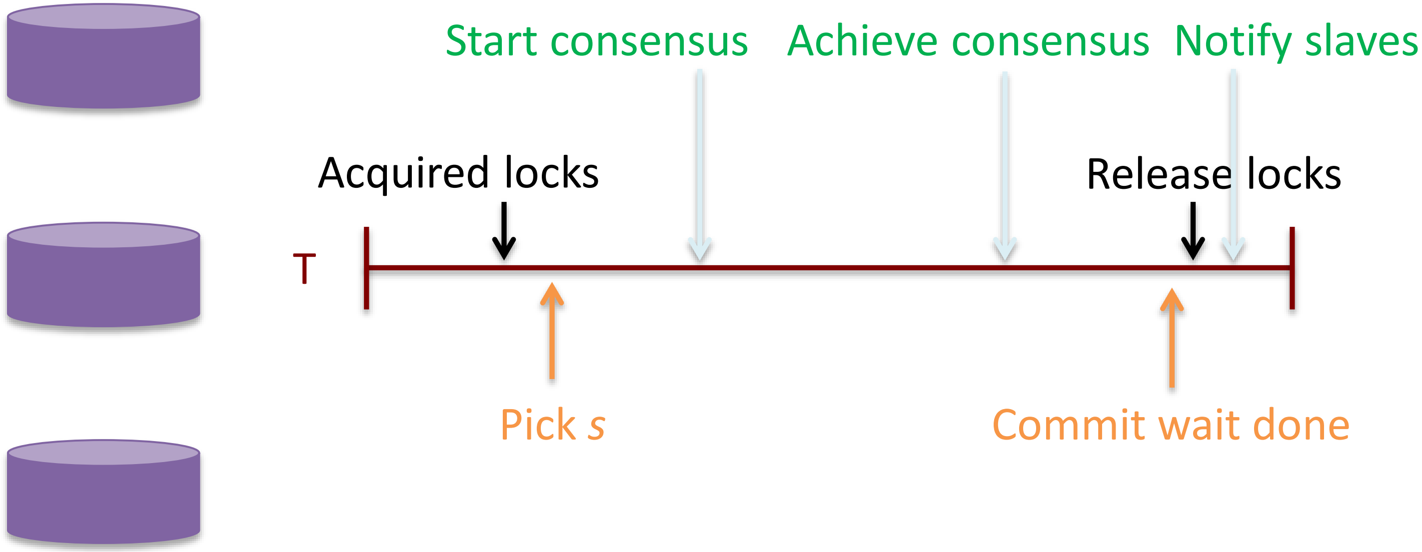
Google

Read Write Transactions

- Leader must only assign timestamps within the interval of its leader lease.
- Timestamps must be assigned in monotonically increasing order.
- Wound-wait protocol to avoid deadlocks



Transaction within paxos group

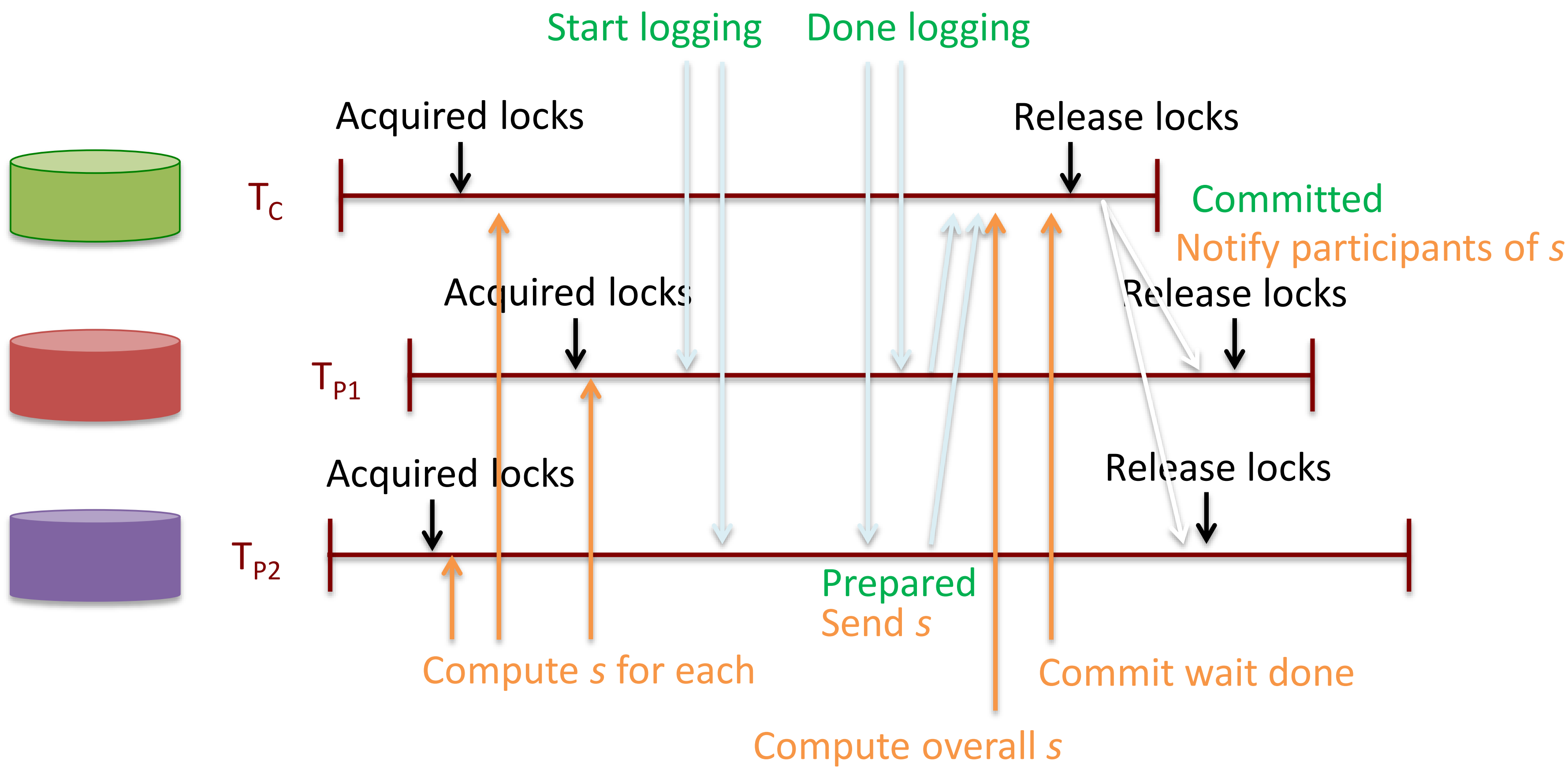


Paxos algorithm is used for consensus

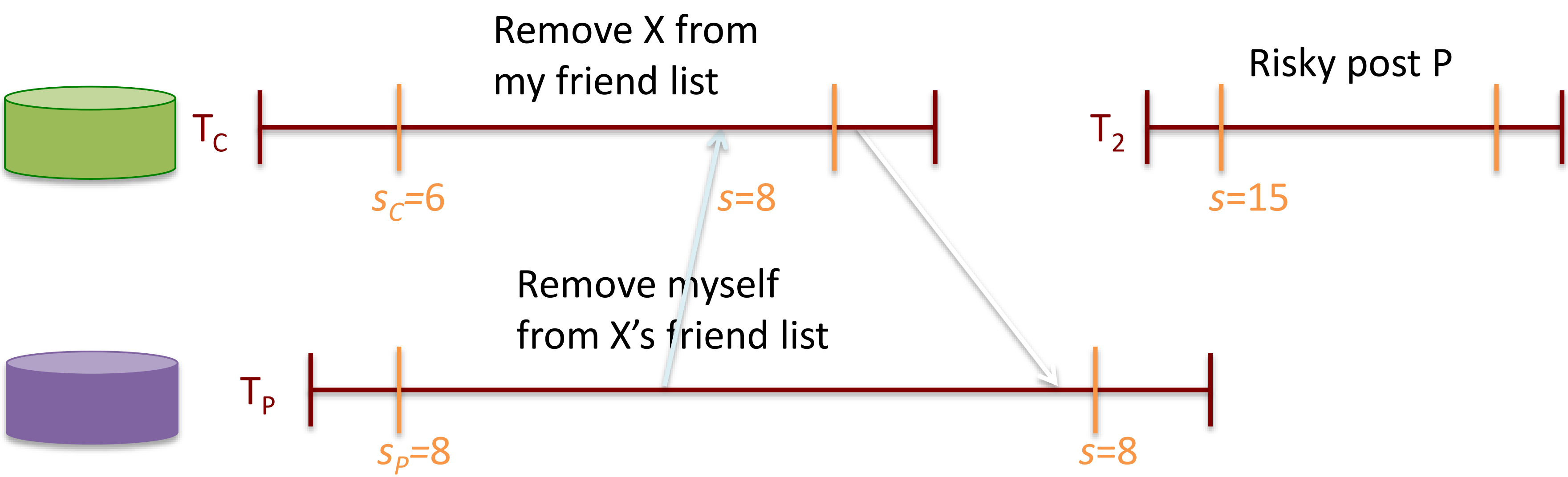
Transactions across Paxos groups

- Client buffer writes
- Client chooses a coordinating group that initiates 2PC
- A non-coordinator-participant leader chooses a prepare timestamp and logs a prepare record through paxos and notifies the coordinator.

2-Phase Commit



Example

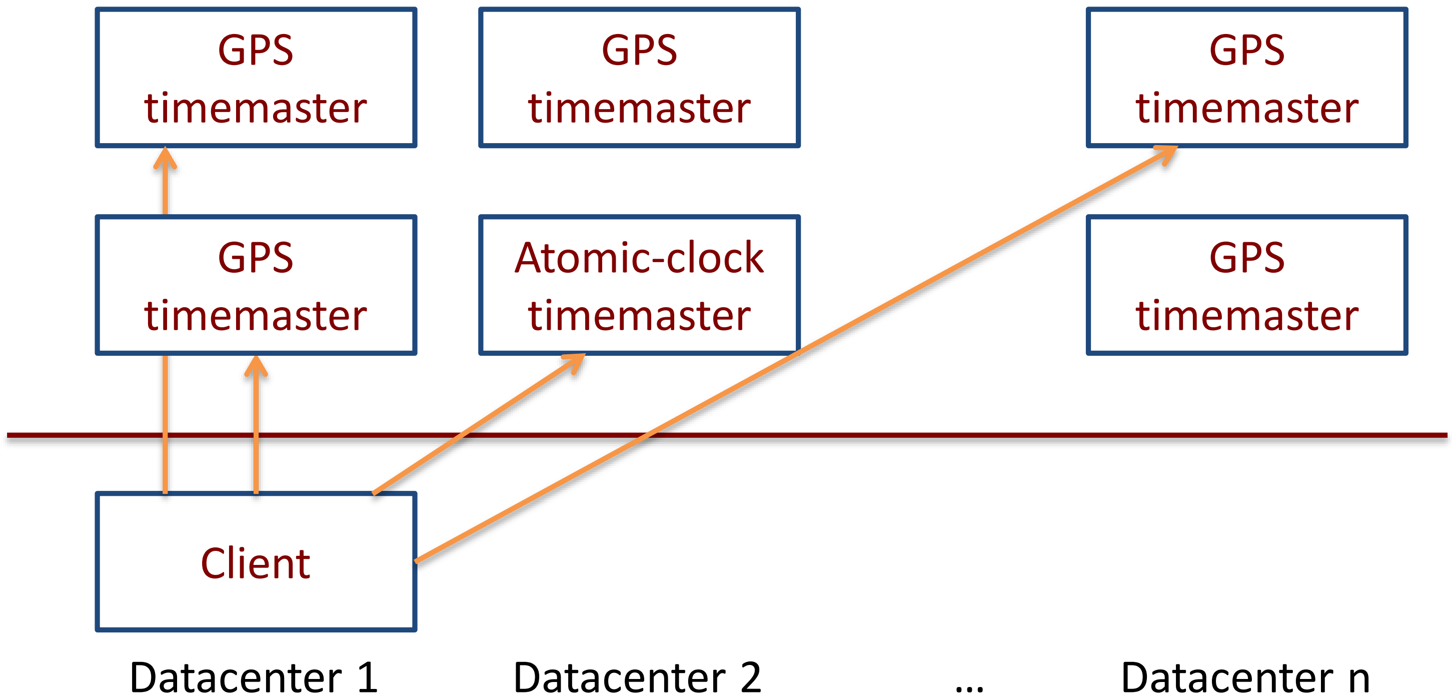


Time	<8	8	15
My friends	[X]	[]	
My posts			[P]
X's friends	[me]	[]	

Schema-change transaction

- How do you make an atomic schema change?
- BigTable only supports changes in one data center, but locks everything
- Spanner has a Non-blocking variant of standard transaction
- Assign a timestamp in the future
- Reads and Writes synchronize with any registered schema at the future timestamp, but not before

TrueTime Architecture

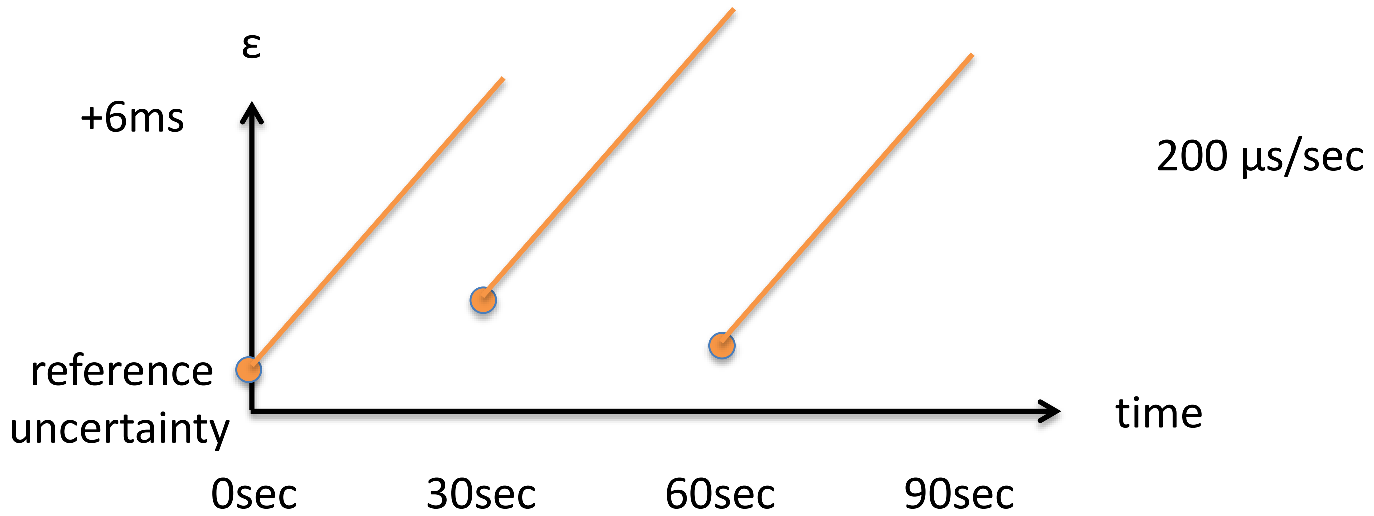


Compute reference [earliest, latest] = now ± ε

TrueTime implementation

now = reference now + local-clock offset

ε = reference ε + worst-case local-clock drift



What If a Clock Goes Rogue?

- Timestamp assignment would violate external consistency
- Empirically unlikely based on 1 year of data
  - Bad CPUs 6 times more likely than bad clocks

Evaluation

replicas	latency (ms)			throughput (Kops/sec)		
	write	read-only transaction	snapshot read	write	read-only transaction	snapshot read
1D	9.4±.6	—	—	4.0±.3	—	—
1	14.4±1.0	1.4±.1	1.3±.1	4.1±.05	10.9±.4	13.5±.1
3	13.9±.6	1.3±.1	1.2±.1	2.2±.5	13.8±3.2	38.5±.3
5	14.4±.4	1.4±.05	1.3±.04	2.8±.3	25.3±5.2	50.0±1.1

Table 3: Operation microbenchmarks. Mean and standard deviation over 10 runs. 1D means one replica with commit wait disabled.

participants	latency (ms)	
	mean	99th percentile
1	17.0 ±1.4	75.0 ±34.9
2	24.5 ±2.5	87.6 ±35.9
5	31.5 ±6.2	104.5 ±52.2
10	30.0 ±3.7	95.6 ±25.4
25	35.5 ±5.6	100.4 ±42.7
50	42.7 ±4.1	93.7 ±22.9
100	71.4 ±7.6	131.2 ±17.6
200	150.5 ±11.0	320.3 ±35.1

Table 4: Two-phase commit scalability. Mean and standard deviations over 10 runs.

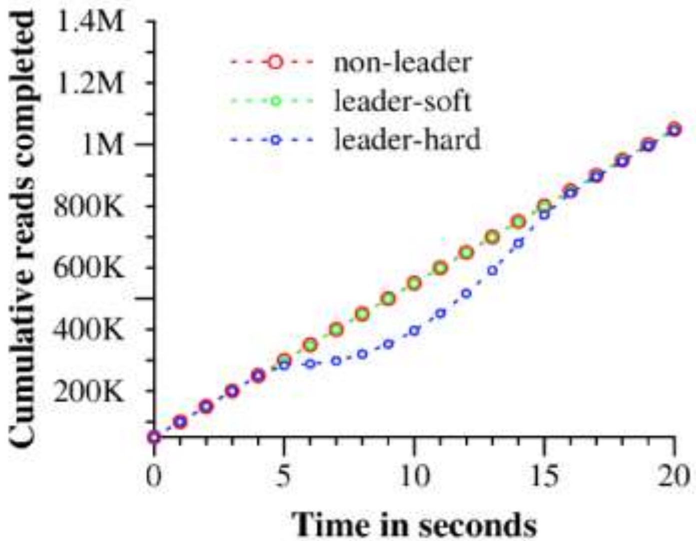


Figure 5: Effect of killing servers on throughput.

F1 Case Study

- Spanner experimentally evaluated under production workloads; rewrite of Google's advertising backend called F1.
- Replaced a MySQL database that was manually sharded.
- The uncompressed dataset is tens of terabytes.
- Spanner removes need to manually reshard
- Spanner provides synchronous replication and auto failover
- F1 requires strong transactional semantics

Future Work

- Improving TrueTime
  - Lower $\varepsilon < 1$ ms
- Building out database features
  - Finish implementing basic features
  - Efficiently support rich query patterns

Advancements

- Spanner claims to be consistent and available
  (A white paper published by Eric Brewer, Google)

- Spanner: Becoming a SQL System
  (SIGMOD'17, May 14–19, 2017, Chicago, IL, USA)
  – Distributed query execution in the presence of resharding
  – query restarts upon transient failures
  – range extraction that drives query routing and index seeks
  – OLTP data management system

Cloud Spanner

	CLOUD SPANNER	TRADITIONAL RELATIONAL	TRADITIONAL NON-RELATIONAL
Schema	✓ **Yes**	✓ Yes	✗ No
SQL	✓ **Yes**	✓ Yes	✗ No
Consistency	✓ **Strong**	✓ Strong	✗ Eventual
Availability	✓ **High**	✗ Failover	✓ High
Scalability	✓ **Horizontal**	✗ Vertical	✓ Horizontal
Replication	✓ **Automatic**	Configurable	Configurable

Google

Conclusions

- The first service to provide global externally consistent multi-version database
- Relies on novel time API (TrueTime)
  - Known unknowns are better than unknown unknowns
  - Rethink algorithms to make use of uncertainty
- Stronger semantics are achievable
  - Greater scale != weaker semantics

Thanks

- Reference:
  - Spanner: Google’s Globally-Distributed Database
  - Slides on spanner by Google in OSDI 2012 talk
  - http://research.google.com/archive/spanner.html

- Questions?