Spanner
Google's Distributed Database

Prof. Pramod Bhatotia
https://dse.in.tum.de/bhatotia/

Bird’s-eye view of Spanner

- Globally-distributed scalable multi-version database
- Synchronous replication (using Paxos)
- Externally-consistent (Linearizable) distributed transactions
- It’s just tip of iceberg (supports many more features…)

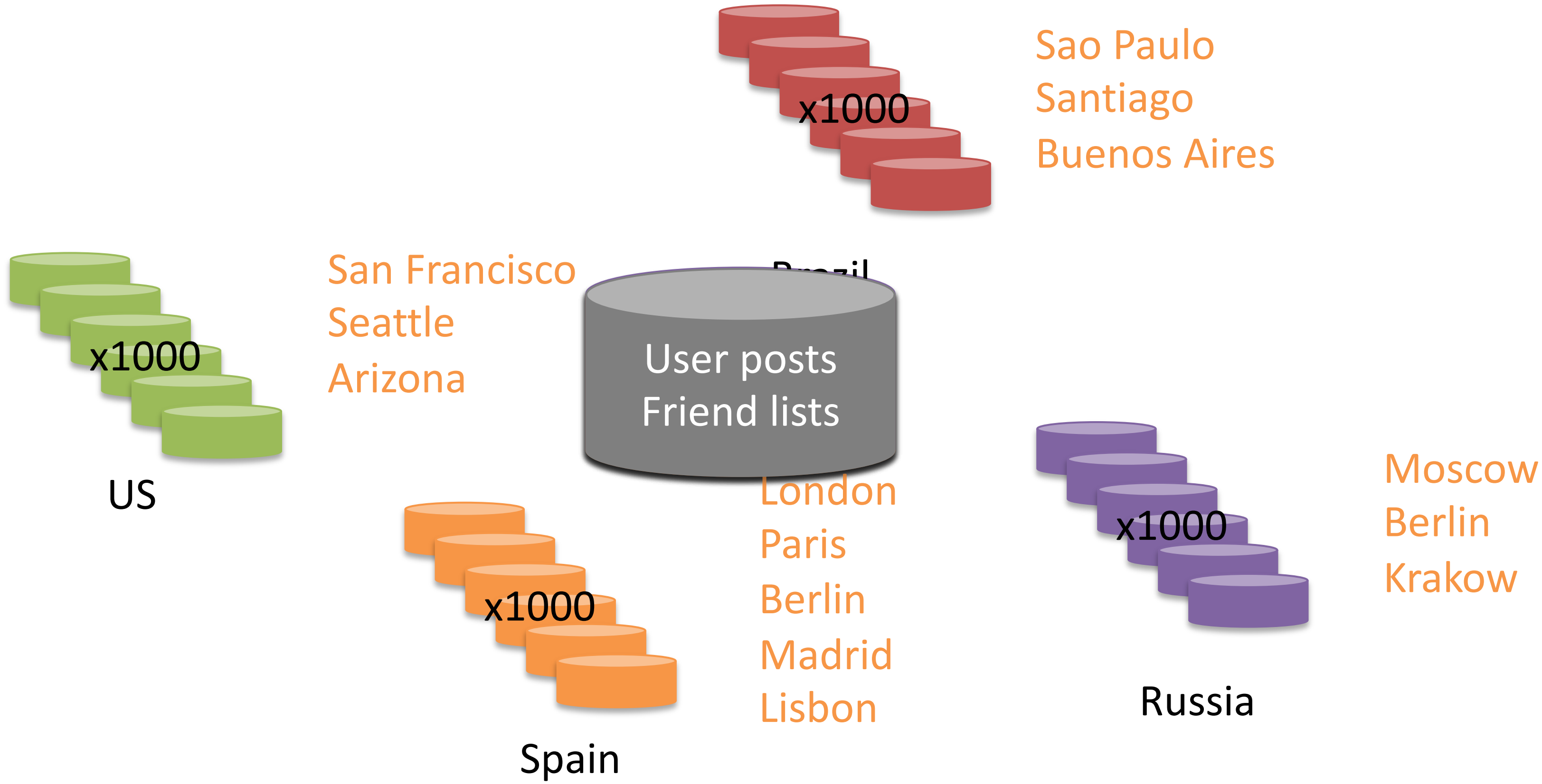
What is the problem being solved?

Building a global scale distributed database with transactional support to help application programmers

Why is it challenging?

Distributed transactions with strong semantics at global scale is difficult

Example: Social network data



Wasn't BigTable designed to support large-scale data management?

*BigTable [OSDI '06]

Complex app requirements!

- An example operation for social network application:
  - Remove untrustworthy person X as friend
  - Post P: “My government is repressive…”

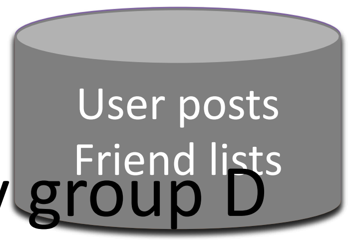
- Consistency matters!
  - Generate a page of friends’ recent posts
  - Consistent view of friend list and their posts

- Require transactions
  - BigTable doesn’t support multi-row transactions!

MegaStore [CIDR’2011]

Entity group B

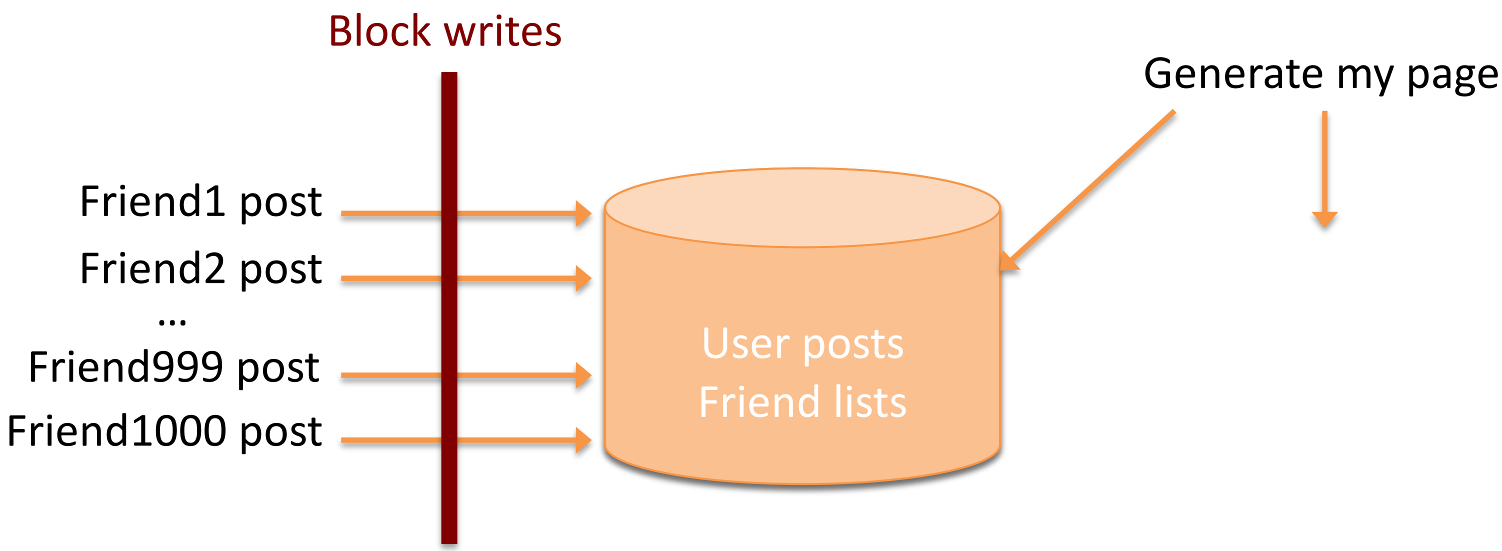
Entity group A



Entity group C

Entity group D

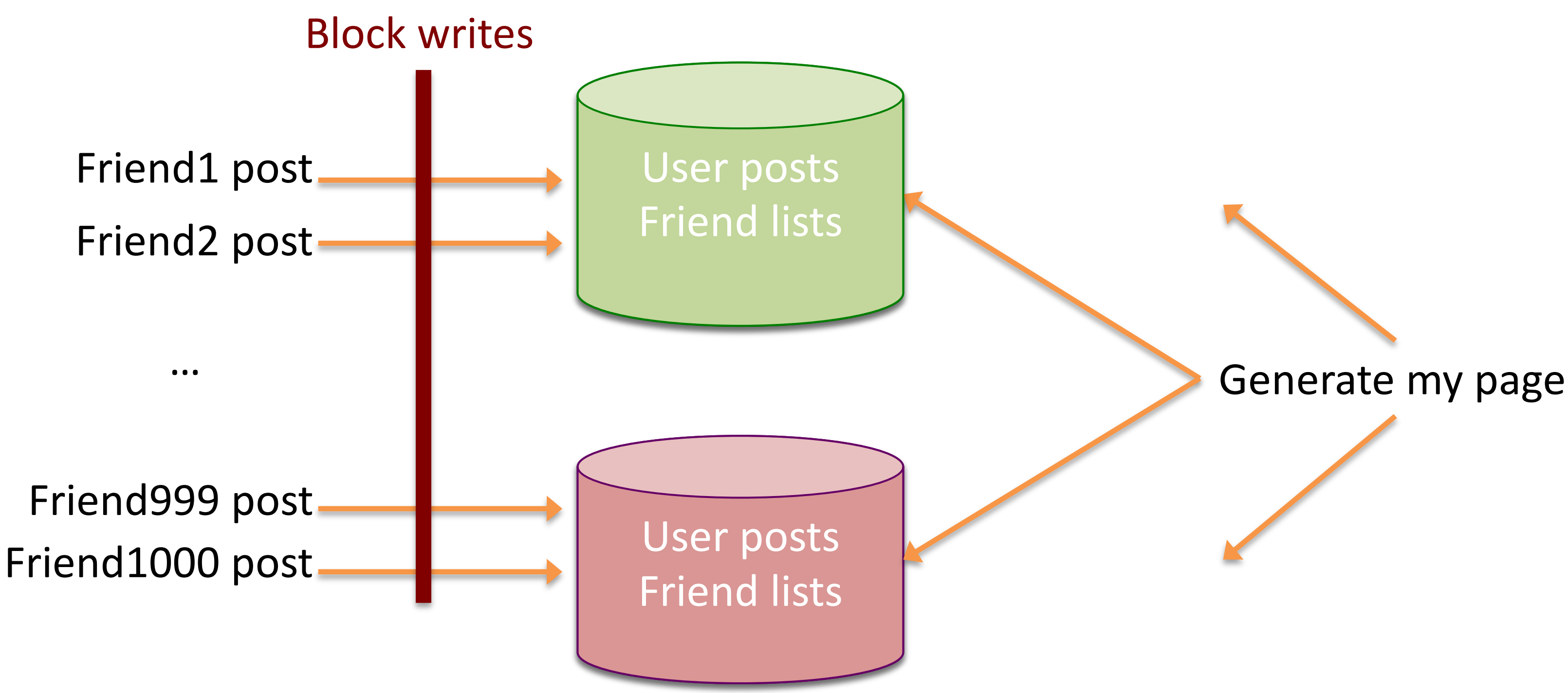
Transaction support in Megastore



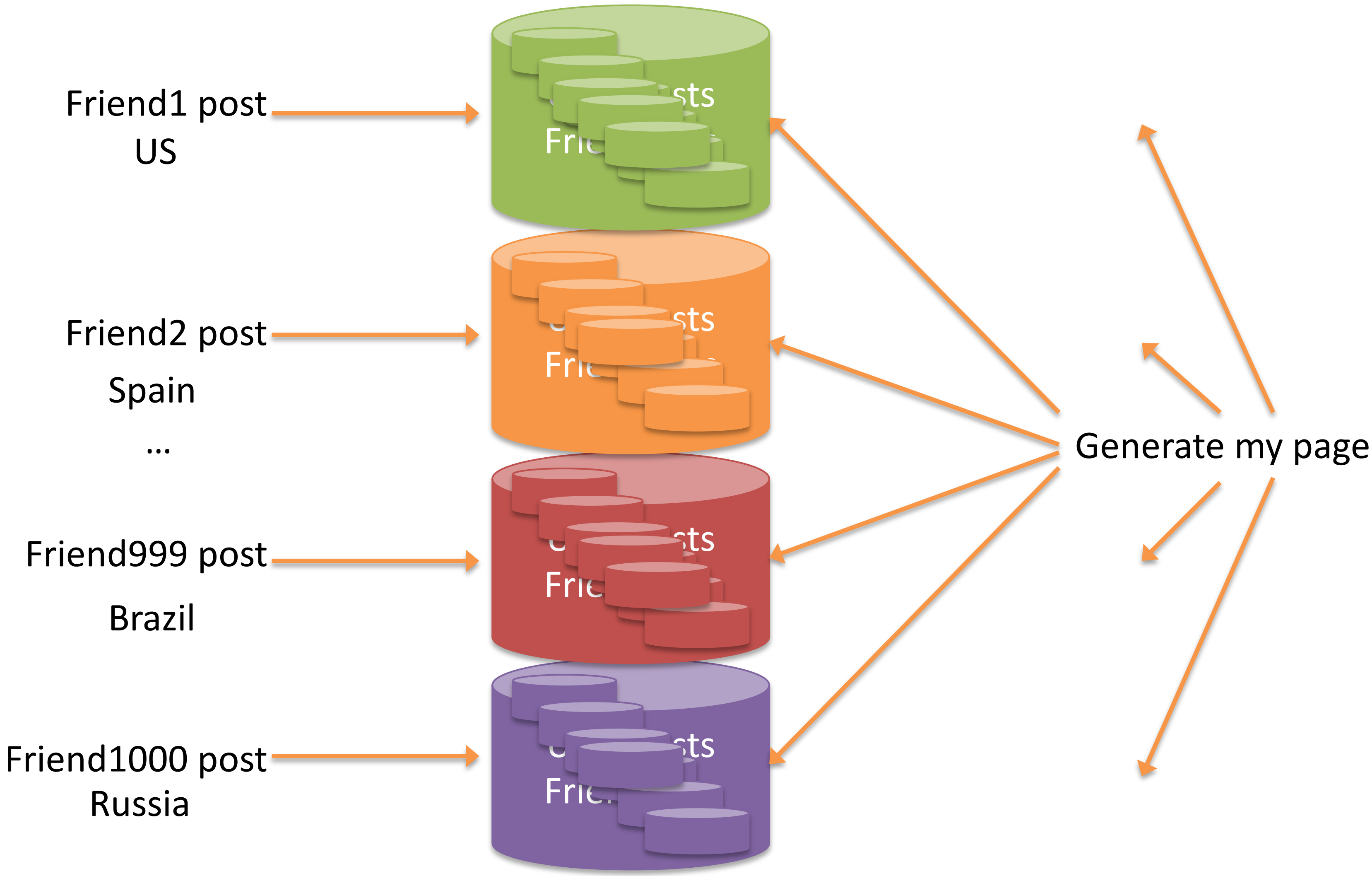
- Supports transactions within entity groups
- Between entity groups require 2PC (using async queue)

Low write throughput!

Multiple entity groups



Multiple data-centers



Let's take a step back!

What property Spanner is trying to achieve?

Externally consistent transactions
(Linearlizability)

Externally consistent transactions

- Transactions that write use strict 2PL
  - Each transaction *T* is assigned a timestamp *s*
  - Data written by *T* is timestamped with *s*

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

If transaction T1 commits before another transaction T2 starts, then T1's commit timestamp is smaller than T2's timestamp.

What are the key insights?

Spanner's recipe

Spanner builds on many standard techniques

- State machine replication (Paxos) within a shard
- Two-phase locking (2PL) for serialization
- Two-phase commit (2PC) for distributed transactions or cross-shard transactions
- Multi-version database systems
- Snapshot isolation

Key idea: Synchronize snapshots

Global wall-clock time

==

External Consistency:

Commit order respects global wall-time order

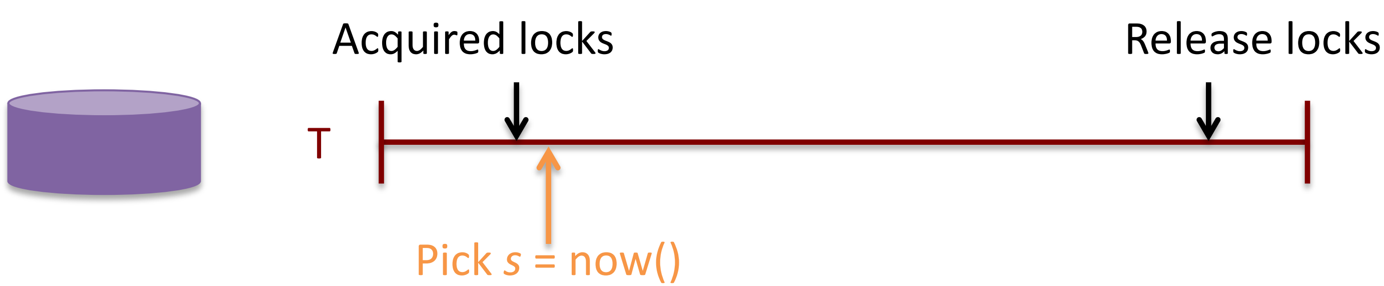
Timestamp order respects global wall-time order

given

timestamp order == commit order

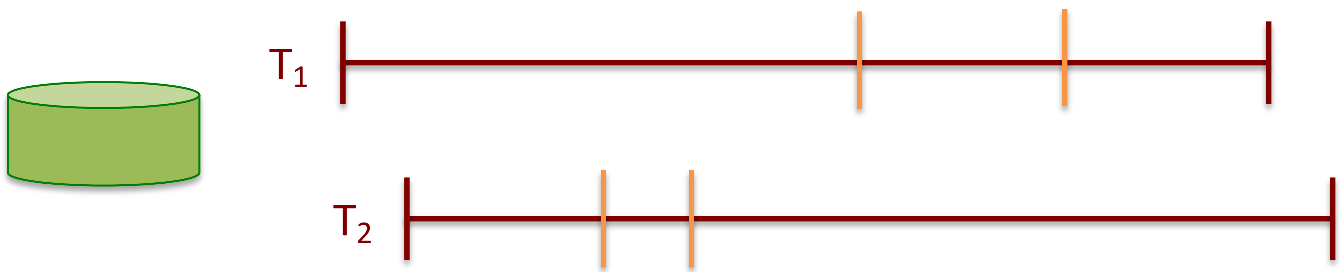
Single machine transaction

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

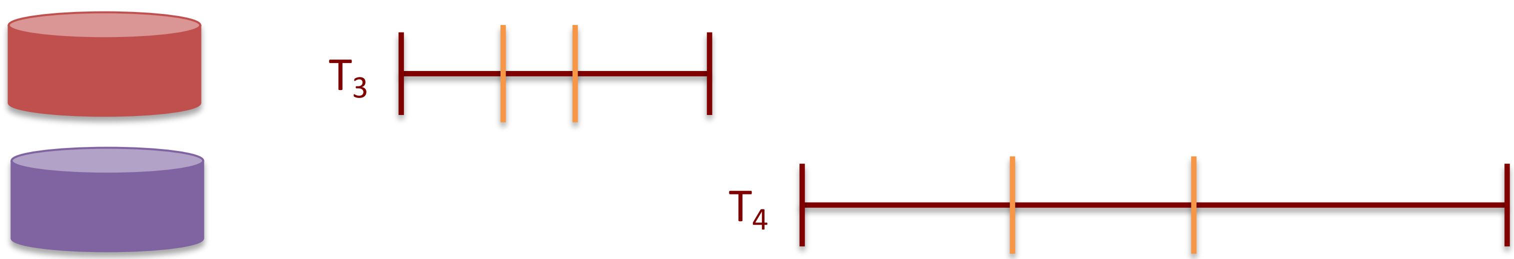


Timestamp invariants

- Timestamp order == commit order

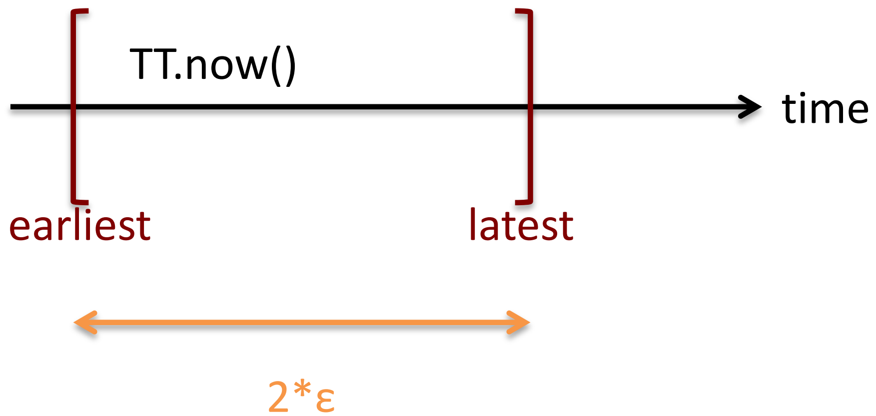


- Timestamp order respects global wall-time order



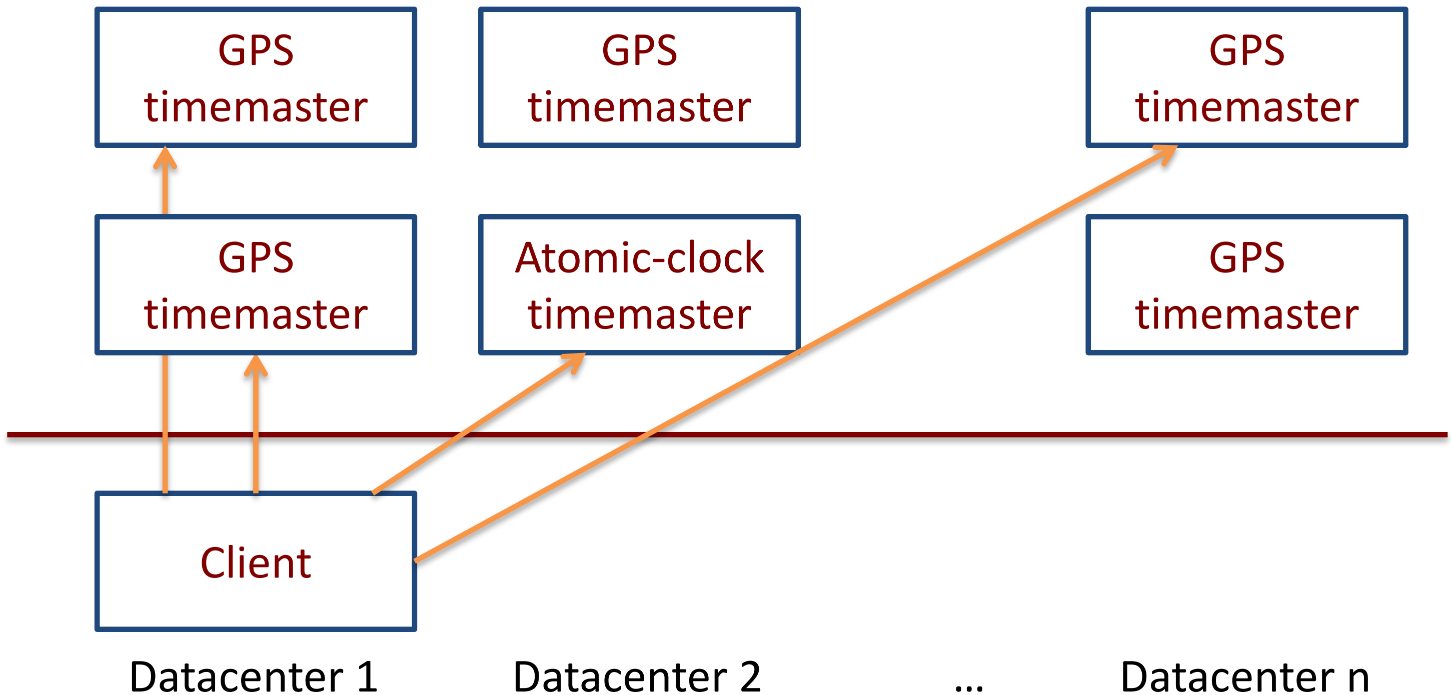
TrueTime

- A novel time API that exposes clock uncertainty



“Global wall-clock time”

TrueTime Architecture



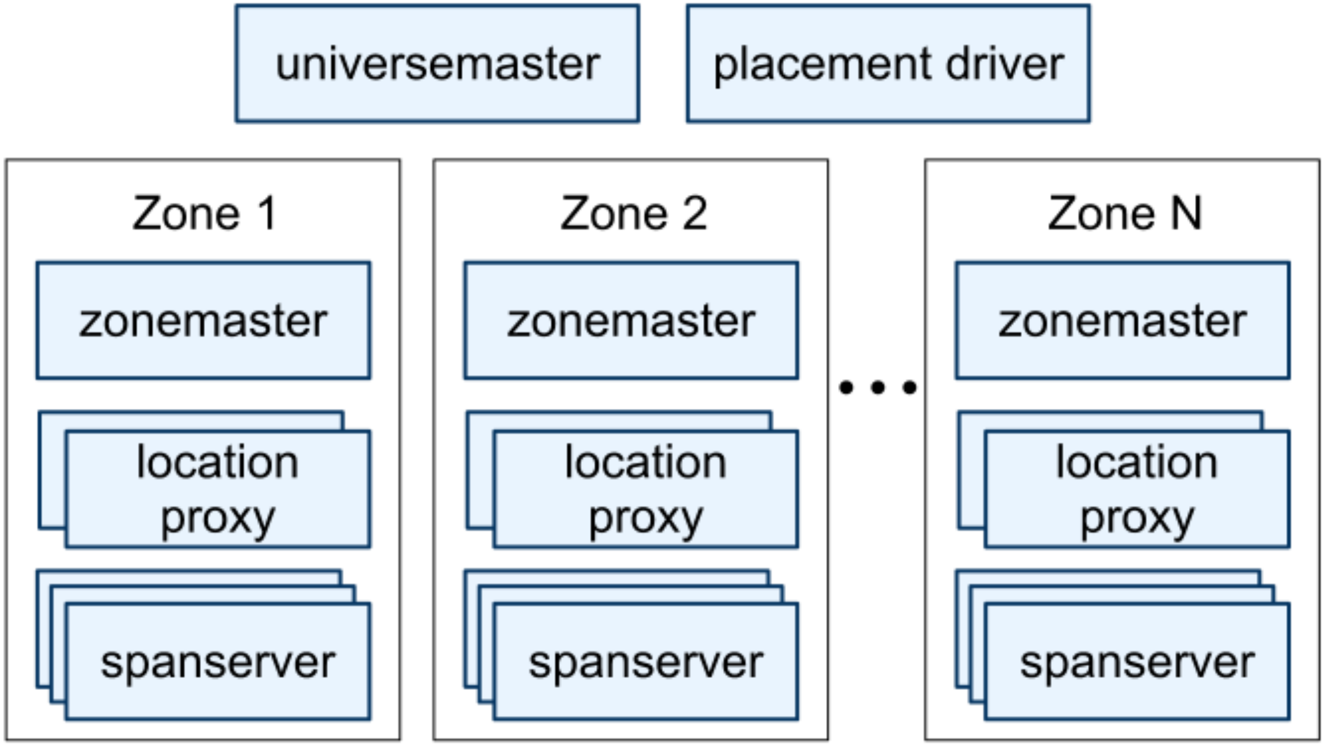
Compute reference [earliest, latest] = now ± ε

How does it works?

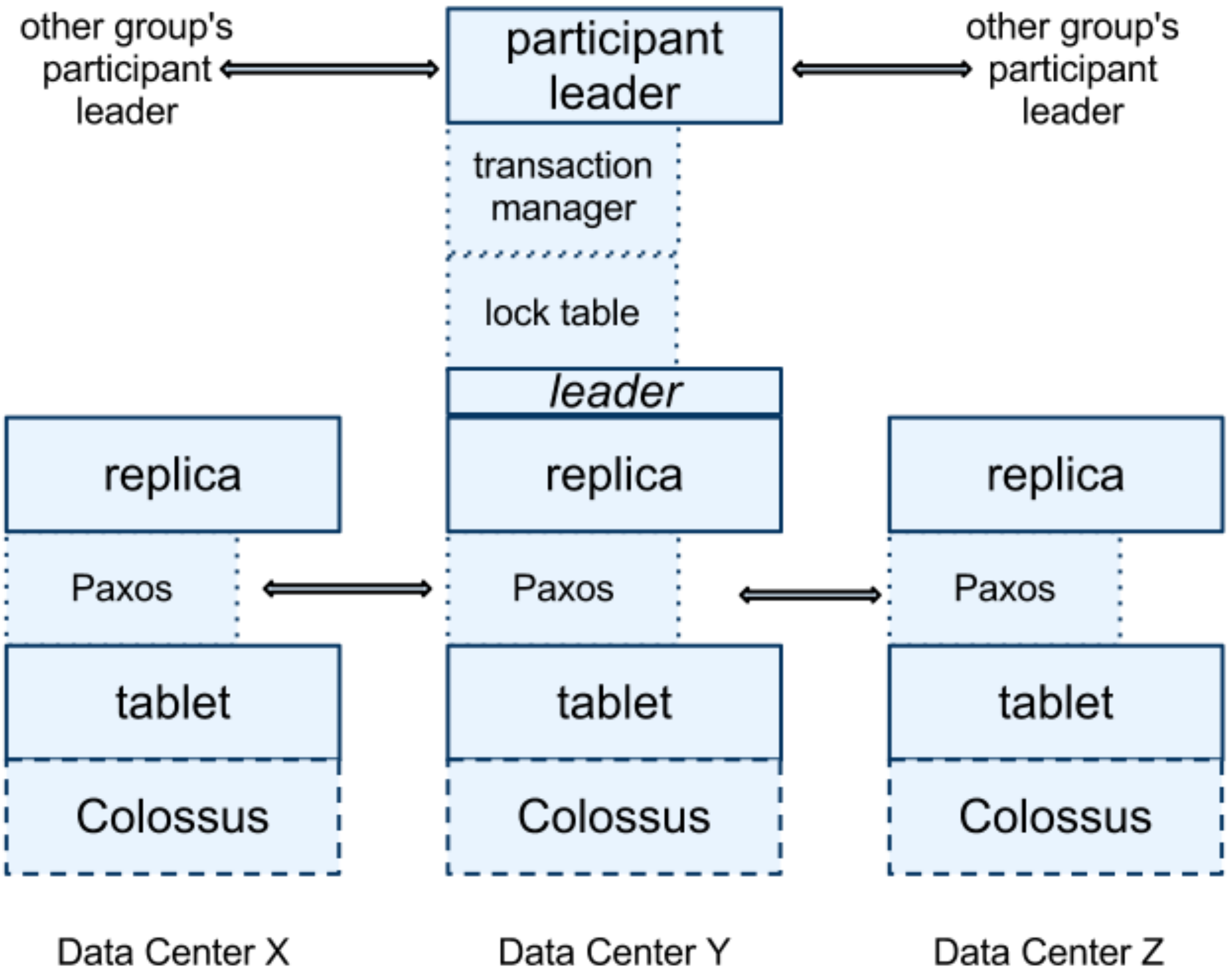
One line answer:

Running 2-PC over Paxos

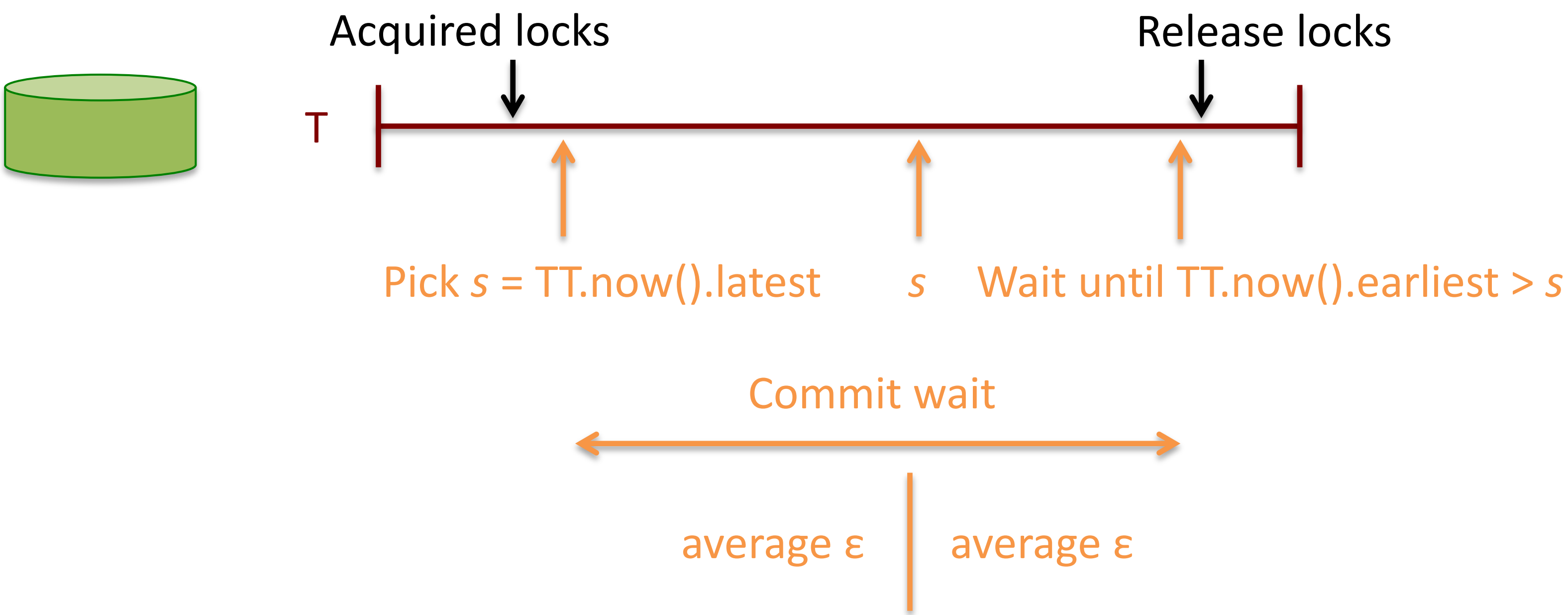
Spanner server organization



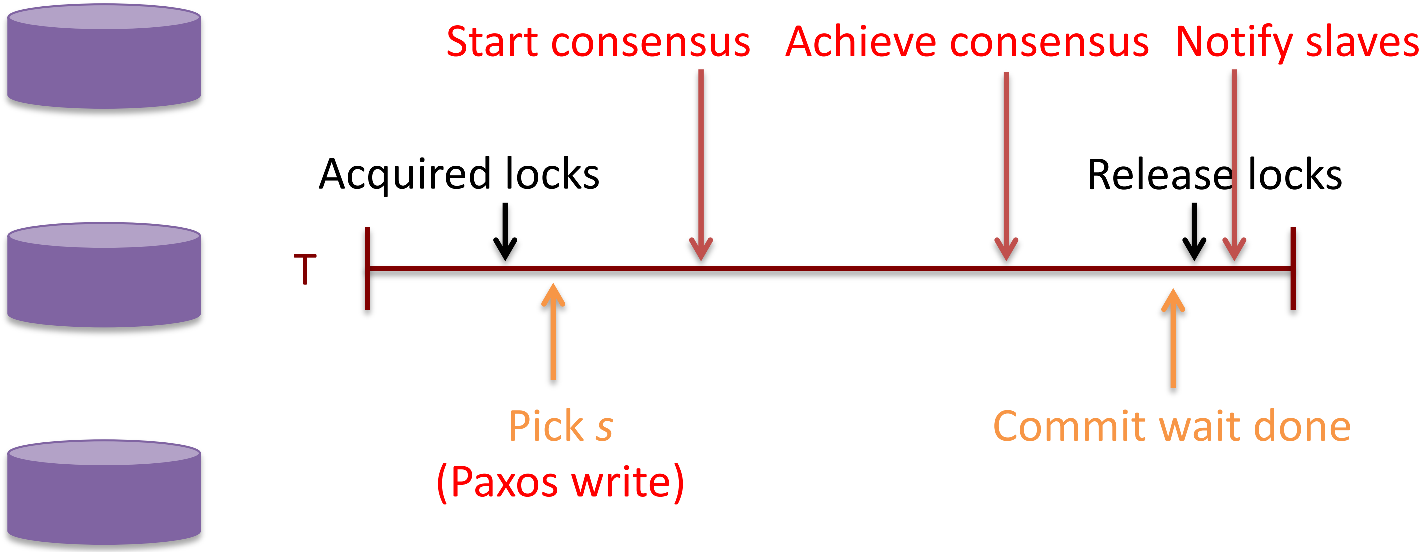
SpanServer software stack



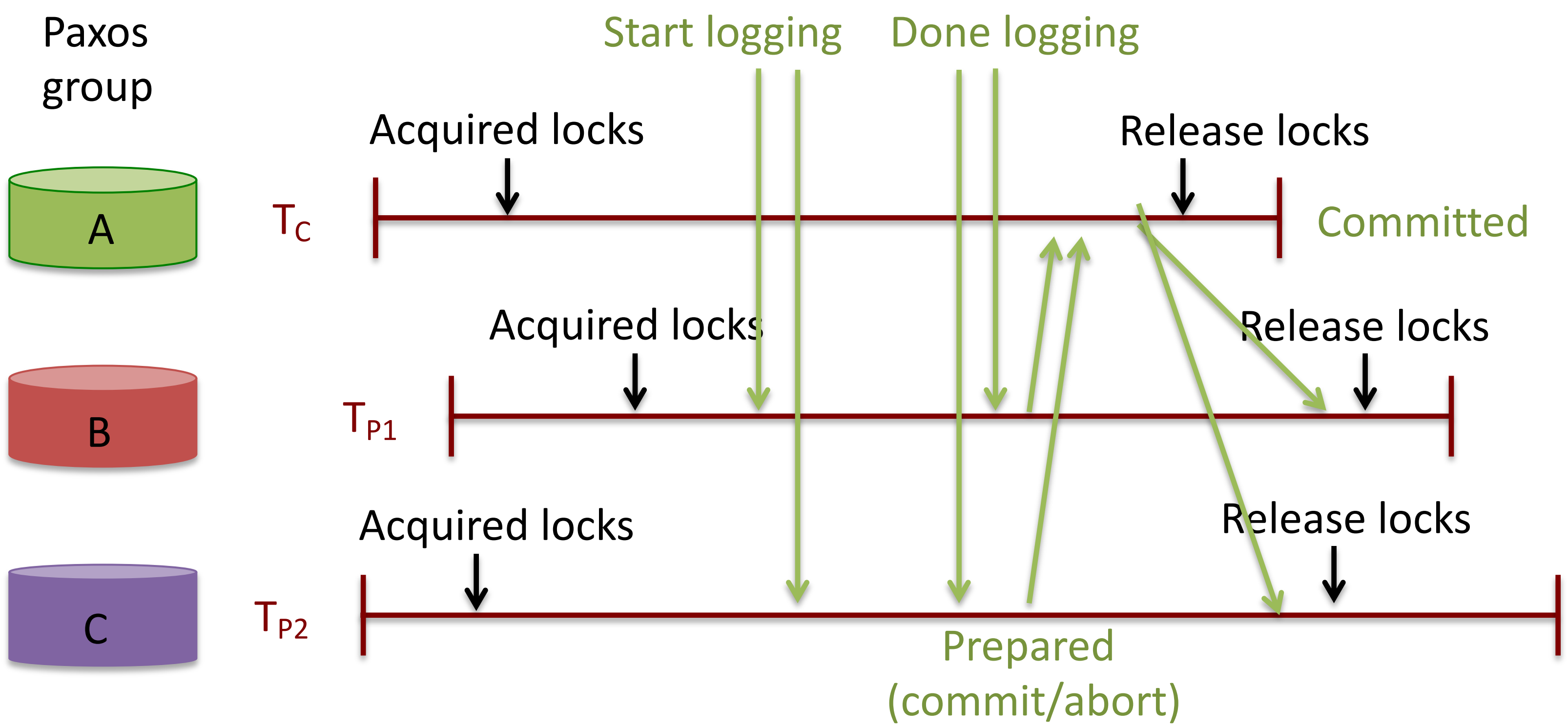
Timestamp assignment w/ TrueTime



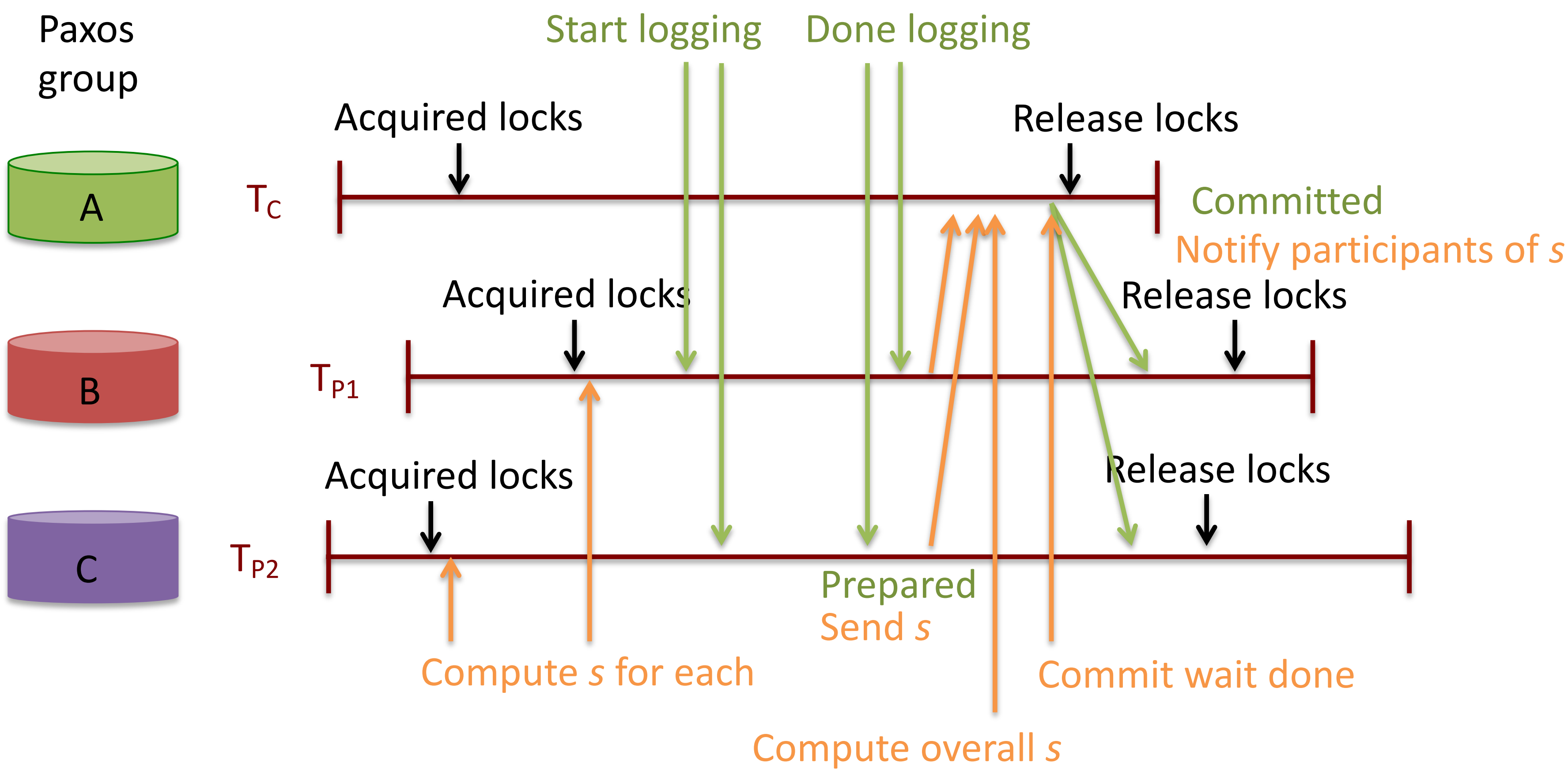
Commit wait and replication



Distributed TXs w/ Paxos group



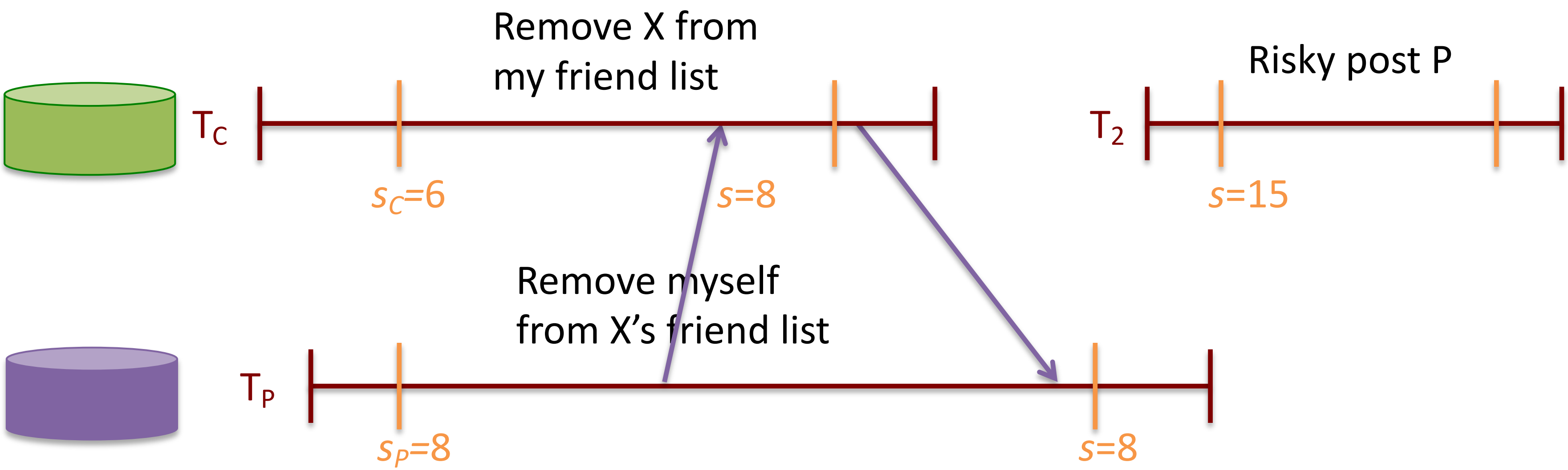
Commit Wait and 2-Phase Commit



Commit time stamp should be greater or equal than

- *TT.now().latest*
- *Any time stamps the co-ordinate assigned to TXs*
- *Prepared time stamps from participants*

Example



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

Conclusions

- Reify clock uncertainty in time APIs
  - Known unknowns are better than unknown unknowns
  - Rethink algorithms to make use of uncertainty

- Stronger semantics are achievable
  - Greater scale != weaker semantics

- References: Spanner: Google's Globally-Distributed Database [OSDI'12]
  - https://www.usenix.org/system/files/conference/osdi12/osdi12-final-16.pdf