CS 5450

Spanner

Slides from Cristina Nita-Rotaru, Mike Freedman, and others

Spanner

- Scalable, multi-version, globally distributed, synchronously replicated database
  - Externally consistent distributed transactions
  - Lock-free read-only transactions
  - Atomic schema changes
- Scale up to
  - millions of machines
  - hundreds of datacenters
  - trillions of database rows

Linearizability

A guarantee for a single operation on a single object

Informally…

- Writes should appear instantaneously within a system
- All later (by wall clock) reads reflect a value written at this or later time

Serializability (Isolation)

A guarantee for transactions consisting of one or more operations on one or more objects

- A set of transactions should execute as though each transaction ran in some serial order
- No deterministic order (no wall-clock constraints)
- This is isolation in ACID properties

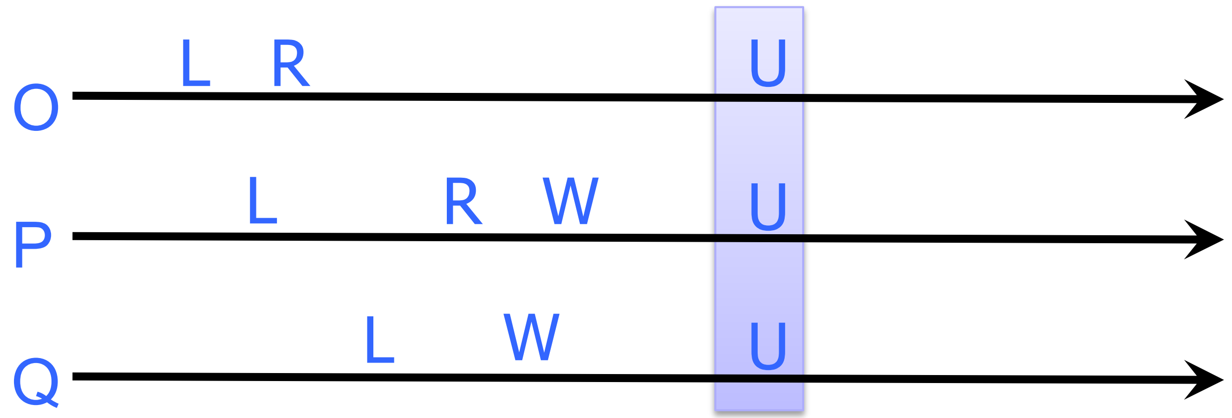
Strict Serializability

- Linearizability + serializability
- Transactions have some serial behavior corresponding to wall-clock time
- Straightforward for non-overlapping transactions but what about concurrent transactions?

Concurrency Controls

- Global lock: simple, slow
- Per-object lock: doesn’t guarantee serializability (isolation)

Partitioned Data Over Servers



- Why not just use 2-phase locking?
  - Grab locks over entire read and write set
  - Perform writes
  - Release locks (at commit time)

Concurrency Controls

- Global lock: simple, slow
- Per-object lock: doesn't guarantee serializability (isolation)
- 2-phase locking: serializability, but can deadlock
- Optimistic concurrency control: good if few conflicts, bad performance if many conflicts
- Multi-version concurrency control: snapshot isolation (weaker than serializability)

DISRUPTIVE IDEA

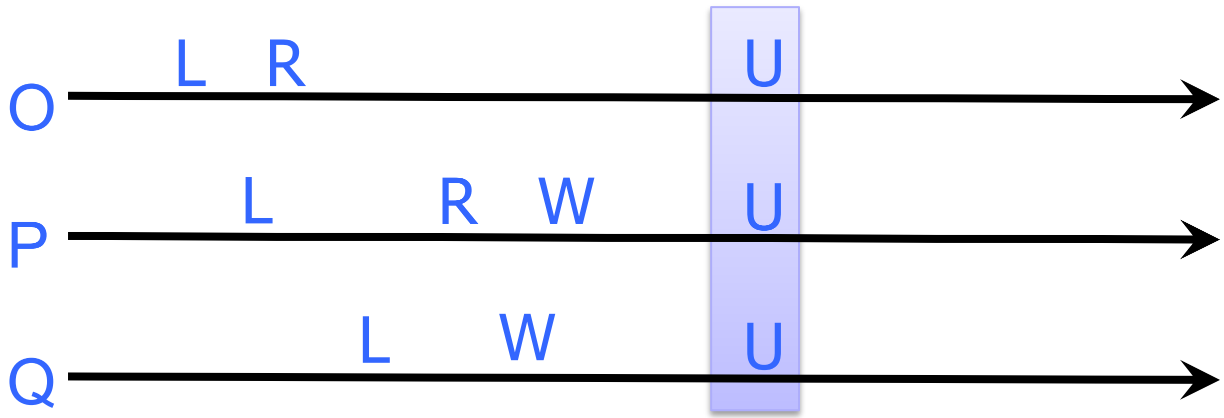
Do clocks **really** need to be arbitrarily unsynchronized?

Can you engineer some max divergence?

Key Idea Behind Spanner

- Attach global commit timestamps to transactions, even though transactions may be distributed
  - Timestamps represent serialization order: if transaction T1 commits before another transaction T2 starts, then T1's commit timestamp is smaller than T2's
- How to get the global timestamps: TrueTime
- Use existing algorithms such as Paxos and 2PC

Partitioned Data Over Servers



- How do you get serializability?
  - Single machine: single COMMIT op in write-ahead log
  - Distributed setting: assign global timestamp to txn (at some time after lock acquisition and before commit)
    - Centralized transaction manager
    - Distributed consensus on timestamps (not all ops)

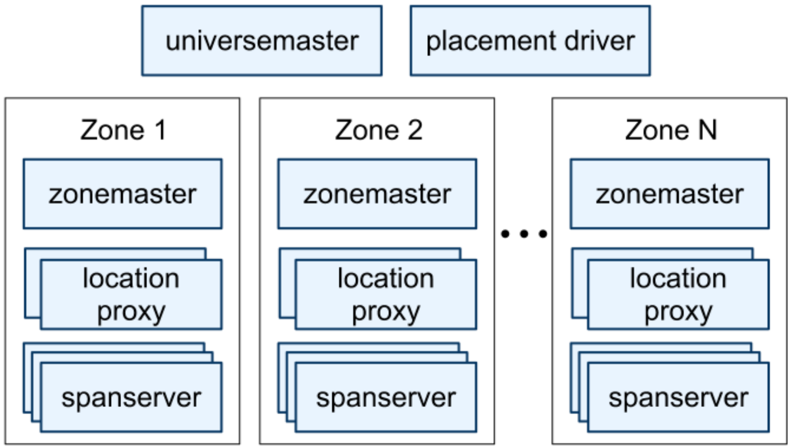
Google's Setting

- Dozens of zones (datacenters)
- Per zone, 100-1000s of servers
- Per server, 100-1000 partitions (tablets)
- Every tablet replicated for fault-tolerance (e.g., 5x)

Spanner Features

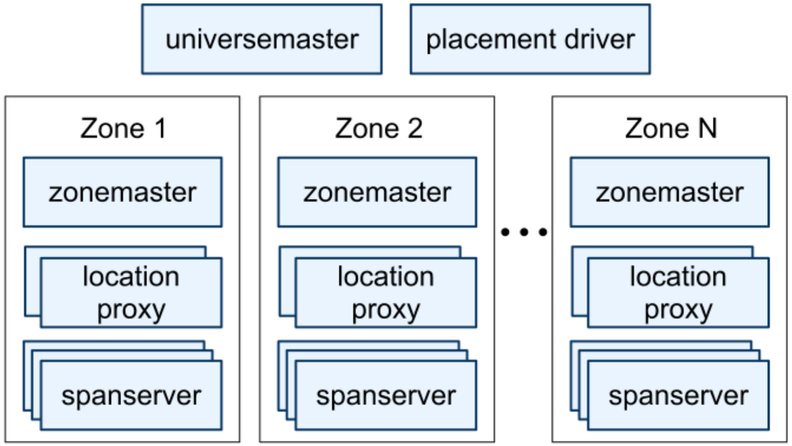
- Applications can control replication configurations for data, specify constraints
  - which datacenters contain which data, how far data is from its users (to control read latency)
  - how far replicas are from each other (to control write latency)
  - how many replicas are maintained (to control durability, availability, and read performance)
- Data can also be dynamically and transparently moved between datacenters by the system to balance resource usage across datacenters

Architecture



- Universe master
- Placement driver
  - Handles automated movement of data across zones on the timescale of minutes
  - Periodically communicates with the spanservers to find data that needs to be moved, either to meet updated replication constraints or to balance load.
- Universe consists of zones
  - Denotes physical isolation
  - Several zones can be in a datacenter

Zones



- Zonemaster
  - Assigns the data to spanservers
- Spanservers
  - Hundreds to thousands
  - Store data
  - Responsible for 100-1000 instances of a data structure called a tablet (different from the BigTable tablet)
- Location proxies
  - Used by clients to locate the spanservers assigned to serve their data

Scale-out vs. Fault Tolerance

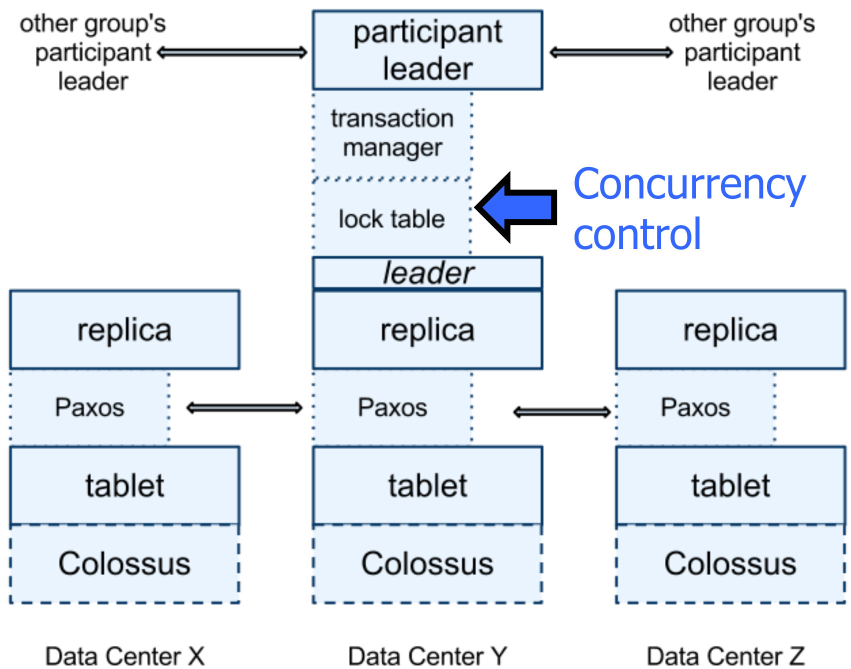
- Every tablet replicated via Paxos (with leader election)
- So every "operation" within transactions across tablets is actually a replicated operation within Paxos RSM
- Paxos groups can stretch across datacenters!

Tablet and Directory

- **Tablet** implements a bag of mappings (key: string, timestamp: int64) → string
  - Spanner assigns timestamps to data… unlike Bigtable, Spanner is more like a multi-version database than a key-value store
- **Directory** is a set of contiguous keys that share a common prefix
  - Smallest unit of data placement
  - Smallest unit to define replication properties

Replication

- Each tablet replicated using Paxos
  - Stores metadata and logs of the tablet
- Leader among replicas in a Paxos group is chosen and all write requests for replicas in that group start at the leader



Paxos Leader Leases

- Paxos uses timed leases to make leadership long-lived (10 seconds by default)
  - Potential leader sends requests for timed lease votes
  - Upon receiving a quorum of lease votes the leader knows it has a lease
  - A replica extends its lease vote implicitly on a successful write, and the leader requests lease-vote extensions if they are near expiration
- Spanner depends on (and enforces) invariant: for each Paxos group, each Paxos leader's lease interval is disjoint from every other leader's

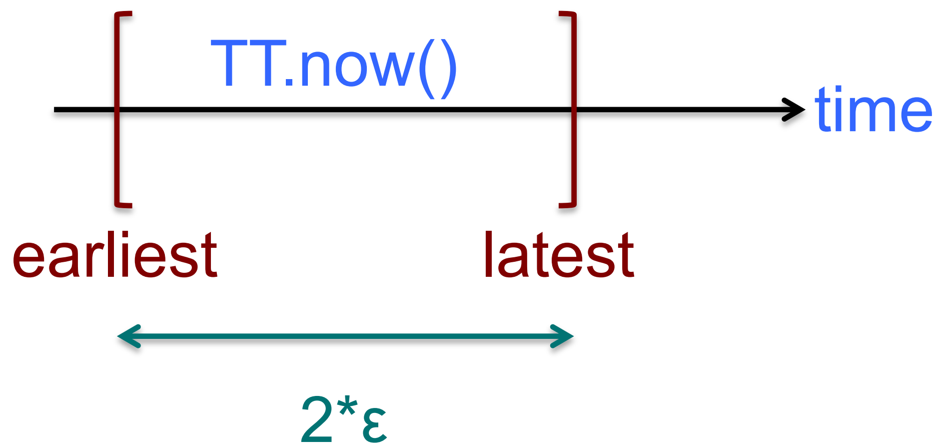
Paxos Leaders and Time

- A Paxos leader can abdicate by releasing replicas from their lease votes. To preserve the disjointness invariant, Spanner constrains when abdication is permissible.
  - S_{max} is the maximum timestamp used by a leader
  - Leader uses after(S_{max}) to check if S_{max} is passed so it can abdicate and release its replicas
- Paxos leaders cannot assign timestamp S_i greater than S_{max} for transaction T_i and clients cannot see the data committed by transaction T_i till after(S_i) is true
- Replicas maintain a timestamp t_{safe} which is the maximum timestamp at which that replica is up to date

All of this depends on global timestamps

TrueTime

- "Global wall-clock time" with bounded uncertainty
  - Timestamps become intervals, not single values



Consider event e_{now} which invoked tt = TT.new():

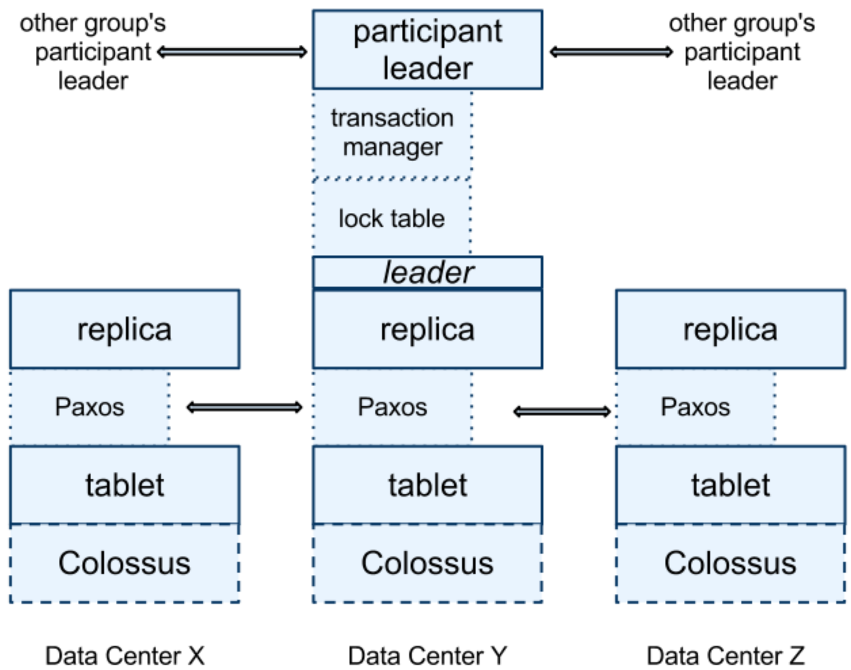
Guarantee: tt.earliest <= $t_{abs}(e_{now})$ <= tt.latest

TrueTime

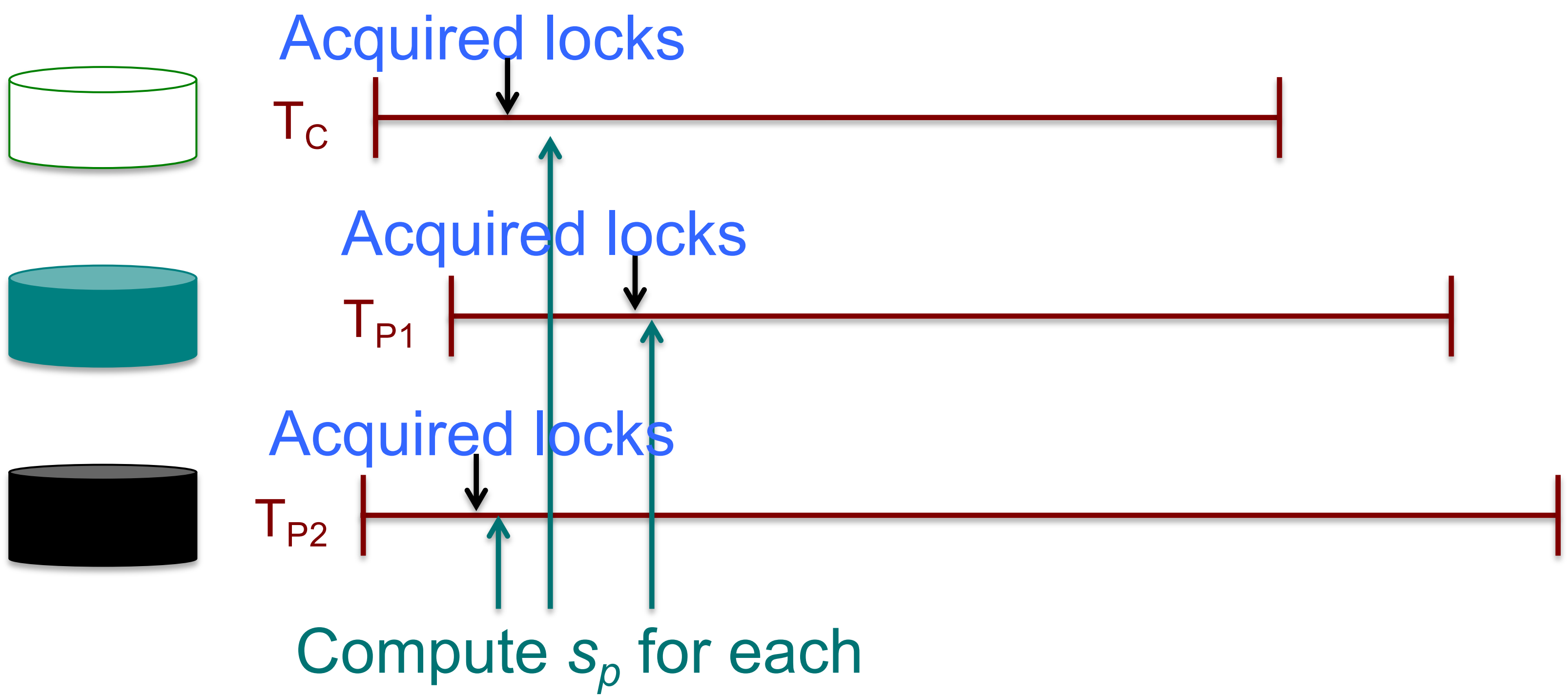
- Daemon polls variety of timemasters and reaches consensus about correct timestamp
  - Majority of time masters are GPS-fitted, a few ”Armageddon masters” are atomic clock-fitted
  - Different failure rates and scenarios
- TrueTime API
  - Key method is **now()** which not only returns current system time but also the maximum uncertainty ε (less than 10ms) in the time returned
  - **After(t)** – returns TRUE if t is definitely passed
  - **Before(t)** – returns TRUE if t is definitely not arrived

Transaction Manager

- If a transaction involves more than one Paxos group, those groups' leaders coordinate to perform two-phase commit
- One of the tablet groups is chosen as the coordinator
- The state of each transaction manager is stored in the underlying Paxos group (and therefore is replicated)

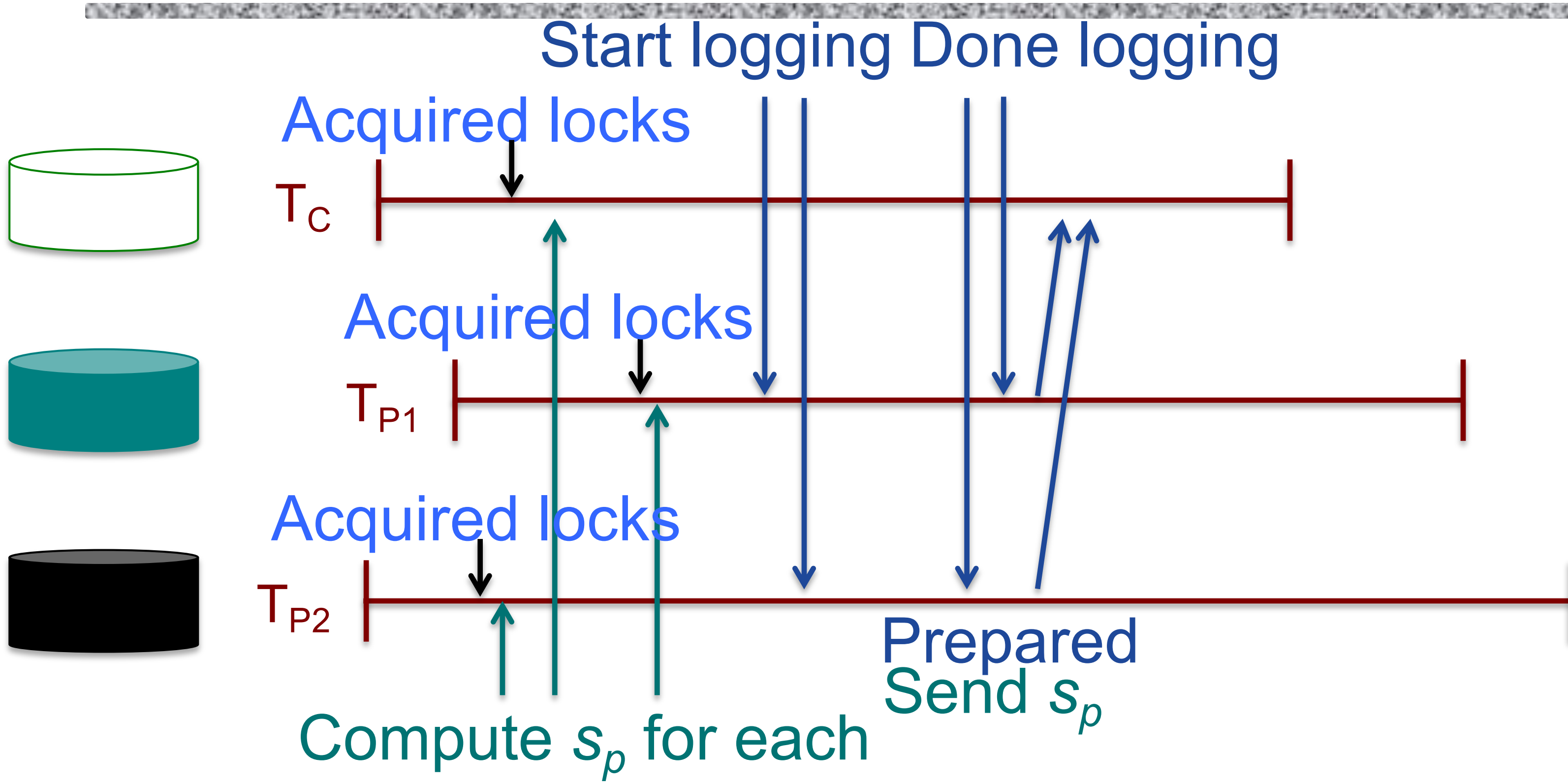


Commit Wait and 2-Phase Commit



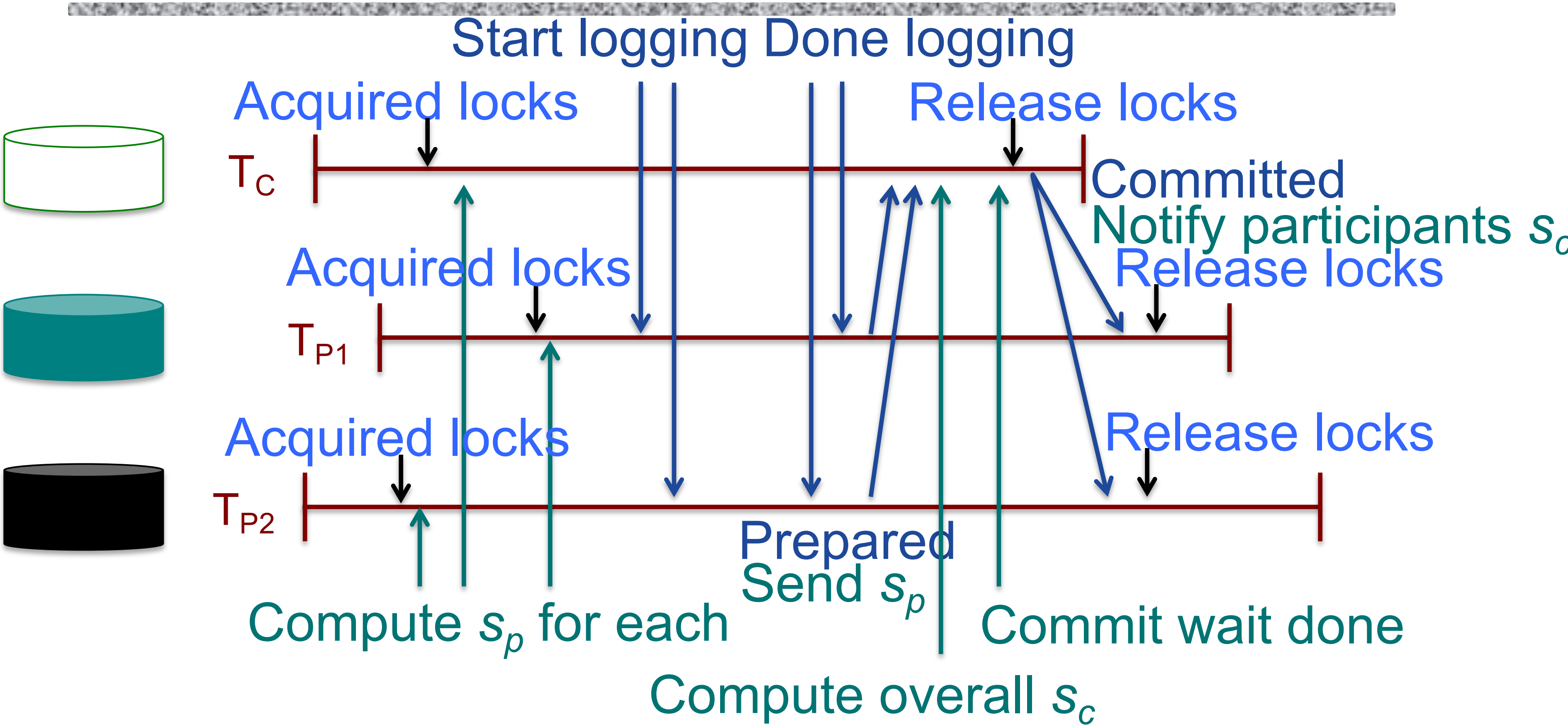
1. Client issues reads to leader of each tablet group, leader acquires read locks and returns most recent data

Commit Wait and 2-Phase Commit



2. Client locally performs (buffers) writes
3. … chooses coordinator from leaders, initiates commit
4. … sends commit msg to each leader (+ coord. identity)

Commit Wait and 2-Phase Commit

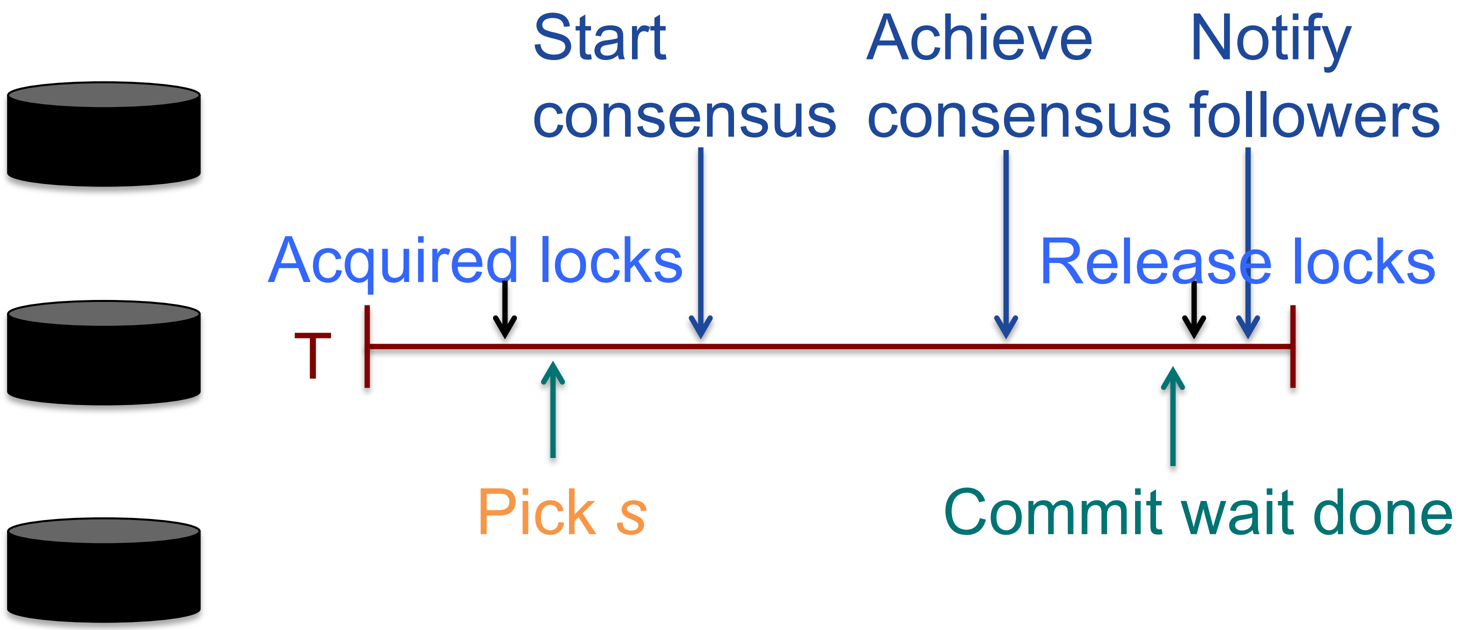


5. Client waits for commit from coordinator

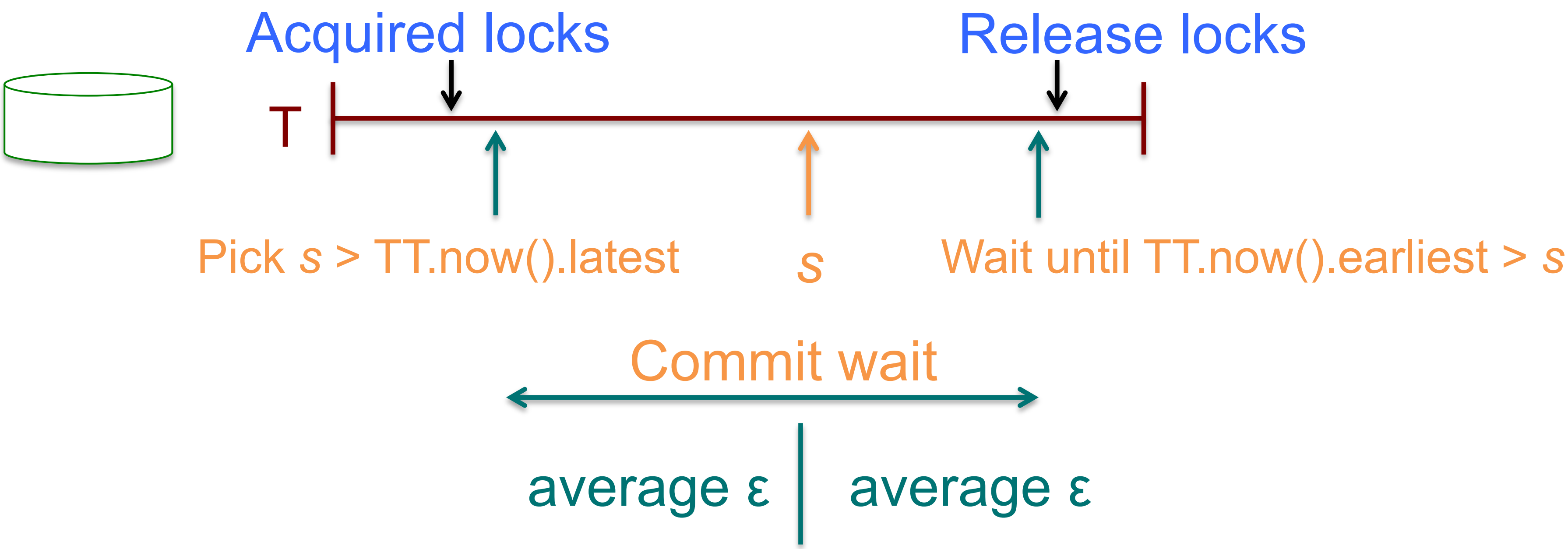
Commit Wait and 2-Phase Commit

- On commit msg from client, leaders acquire local write locks
  - If non-coordinator:
    - Choose prepare ts > previous local timestamps
    - Log prepare record through Paxos
    - Notify coordinator of prepare timestamp
  - If coordinator:
    - Wait until hear from other participants
    - Choose commit timestamp >= prepare ts > local ts
    - Logs commit record through Paxos
    - Wait for the commit-wait period
    - Sends commit timestamp to replicas, other leaders, client
- All apply at-commit timestamp and release locks

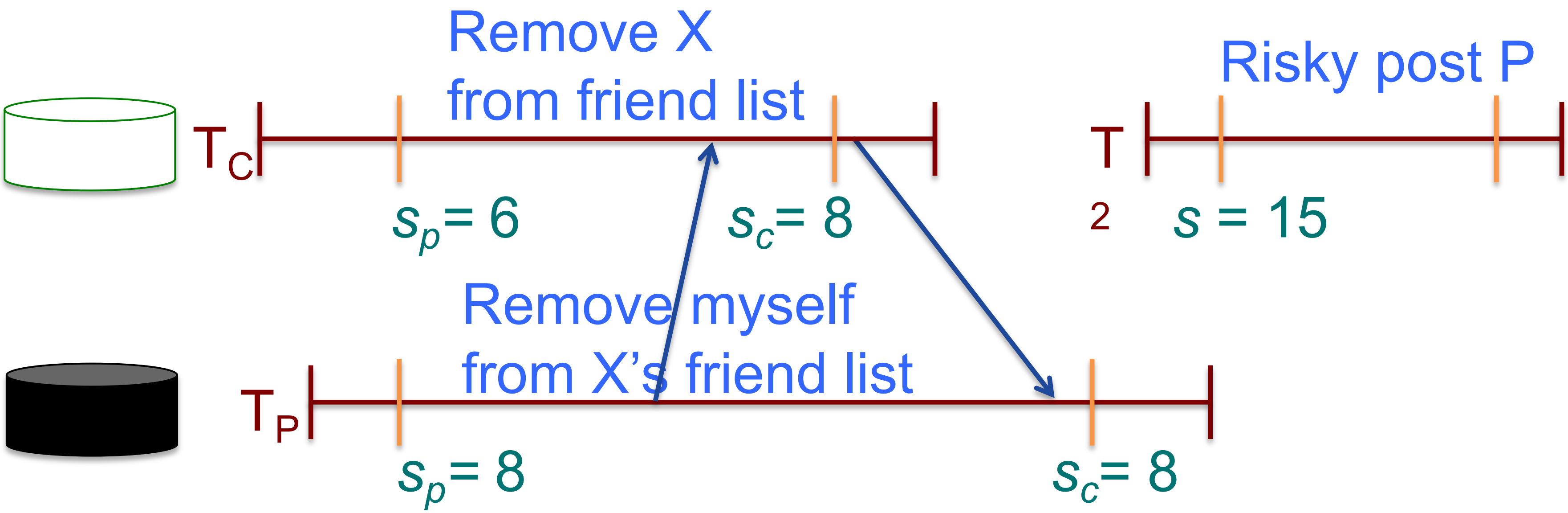
Commit Wait and Replication



Commit Wait and TrueTime



Example



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

Read-only Transactions (1)

- Assigning a timestamp requires a negotiation between all Paxos groups involved in the reads
  - Spanner requires a <u>scope</u> expression for every read-only transaction, summarizes the keys that will be read by the entire transaction
- If the scope's values are served by a single Paxos group, then the client issues the read-only transaction to that group's leader, who assigns reads and executes the transaction

Read-only Transactions (2)

If the scope's values are served by multiple Paxos groups…

- Option 1: Do a round of communication with all leaders to negotiate s_{read} based on LastTS()
- Option 2: The client avoids a negotiation round and has its reads execute at s_{read} = TT.now().latest
  - May wait for safe time to advance
  - All reads in the transaction can be sent to replicas that are sufficiently up-to-date

Schema-Change Transactions

- Schema-change transactions explicitly assigned a timestamp in the future
  - Future timestamp registered in the prepare phase
  - Atomic schema changes across thousands of servers do not disrupt concurrent activities
- Reads and writes, which implicitly depend on the schema, synchronize with any registered schema-change timestamp at time t
  - May proceed if their timestamps precede t
  - Must block behind the schema-change transaction if their timestamps are after t

Consistency Properties

- Eventual consistency
  - Client may see an arbitrary subset of updates
- Snapshot isolation
  - Client can read a consistent database snapshot at any time, incl. all updates up to snapshot timestamp
    - … from any replica, in Spanner
    - Newer updates may still be replicating
- Strong consistency
  - Client always sees the latest, consistent updates

Partition Tolerance in Spanner

- For reads?
- For writes?
- The role of TrueTime?

Remember the CAP theorem?