Spanner: Google's Globally-Distributed Database

Tuochao Chen, Tongyan Wang

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** (provides clients with the strictest concurrency-control guarantees for transactions) distributed transactions.

Why Spanner?

Pros & Cons of Bigtable & Megastore:

Bigtable:
Pros: It supports high read and write throughput at low latency
Cons: asynchronous when performing cross-data center replication, thus only achieve eventual consistency; more like a key-value storage thus can be difficult to use.

Megastore:
Pros: support schemas and provides a SQL-based query language.
Cons: suffers from relatively poor write throughput.

Advantages of Spanner

1. Supports general-purpose transactions, and provides a SQL-based query language; Data is stored in schematized semi-relational tables.
2. Each transaction is automatically timestamped with its commit time. Provides externally consistent reads and writes, and globally-consistent reads across the database at a time-stamp.

Data Model of Spanner

“Each database can contain an unlimited number of schematized tables…”
“Spanner’ s data model is not purely relational. More precisely, every table is required to have an ordered set of one or more primary-key Columns..”

Goolge Spanner

```
| key  |  name    |  age  |
|------|----------|-------|
| goog | Google   |  20   |
| fb   | Facebook |  14   |
...
```

Google Bigtable

```
"goog": {name: "Google" }
"fb"  : {age: 14, location: "Menlo Park, CA"}
"amzn": {url: "www.amazon.com"}
...
```

Bigtable (schemaless) vs Spanner

Organization

- A Spanner deployment is called a *universe*. Each Spanner is separate for different *universe.*

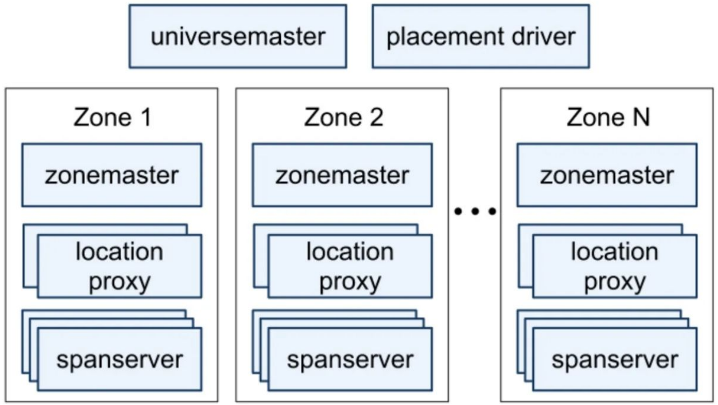


Figure 1: Spanner server organization

Organization

- Spanner is organized as a set of *zones*, which are the unit of administrative deployment. The set of zones is also the set of locations across which data can be replicated.

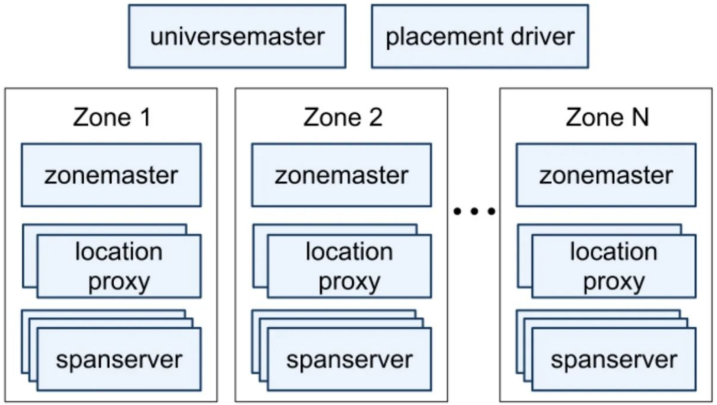


Figure 1: Spanner server organization

Organization

- A zone has one *zonemaster* and between one hundred and several thousand *spanservers*.

  Zonemaster assigns data to spanservers while *spanserver* serve data to clients.

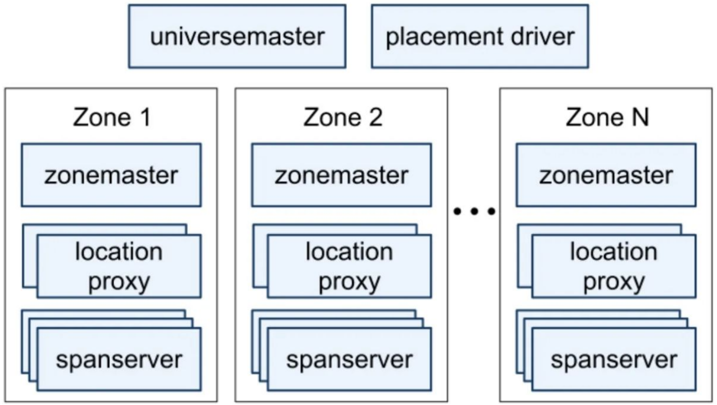


Figure 1: Spanner server organization

Organization

- The per-zone *location proxies* are used by clients to locate the spanservers assigned to serve their data.

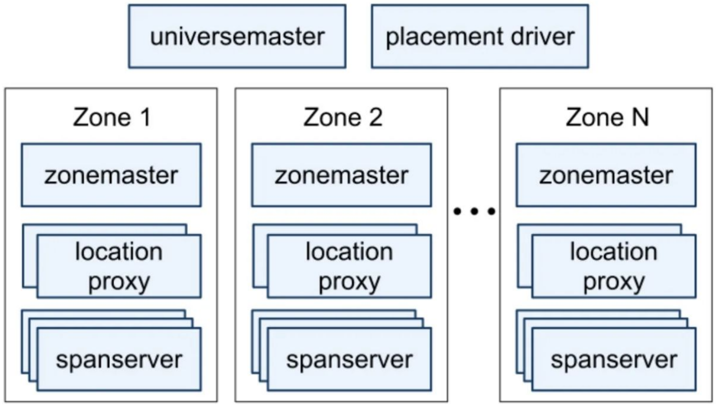


Figure 1: Spanner server organization

Organization

- The *universe master* is primarily a console that displays status information about all the zones for interactive debugging.
- The placement driver handles auto-mated movement of data across zones on the timescale of minutes.

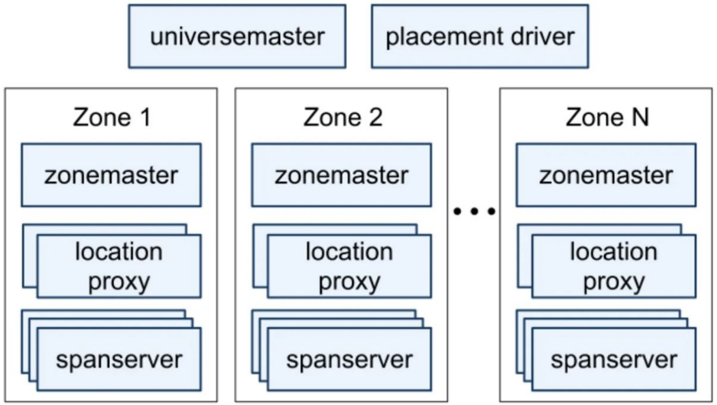


Figure 1: Spanner server organization

Spanserver Software Stack

The main components of distributed storage system are familiar:

- 2 Phase Commit
- Paxos and Replica

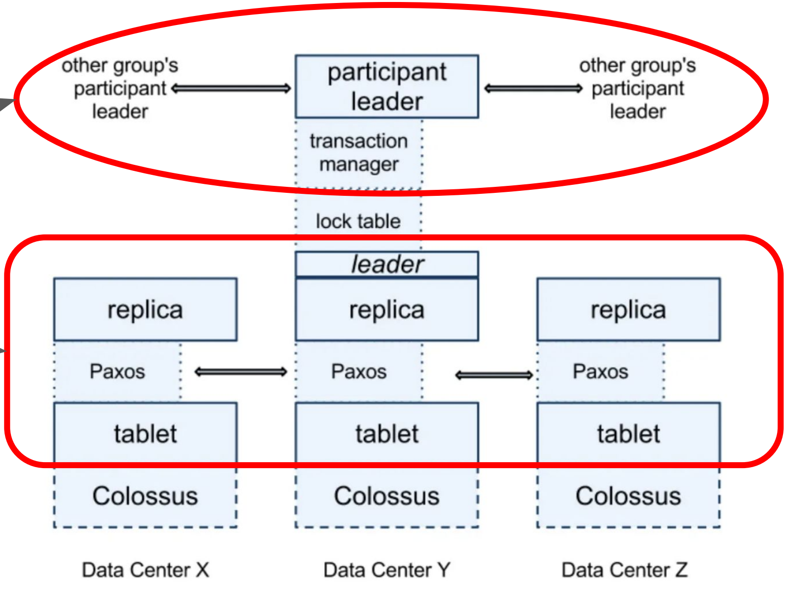


Figure 2: Spanserver software stack

Spanserver Software Stack - Bottom Part

- Each Replica is responsible for one tablet and Colossus, which is used for data storage
- Leader: efficiently execute Paxos protocol.
- The set of part that are responsible for the same dataset is called a paxos group.

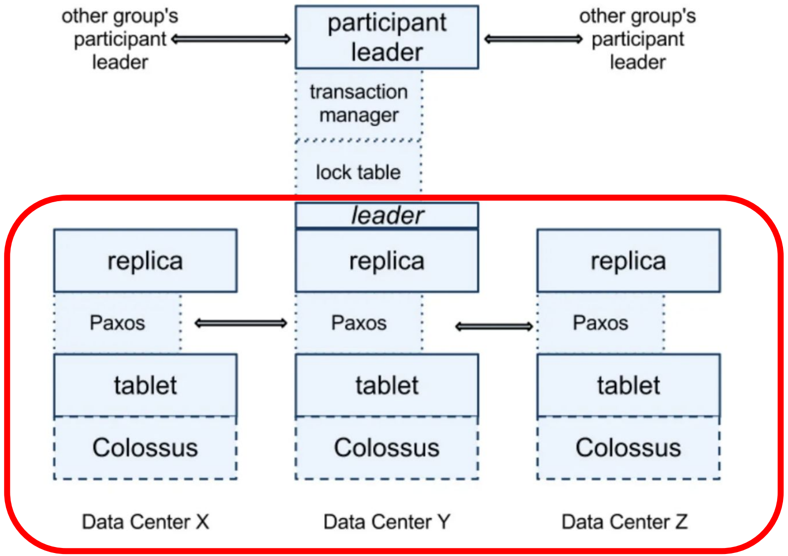


Figure 2: Spanserver software stack

Spanserver Software Stack - Top Part

- Lock table:
  Control read-write access

- Transition manager:
  Responsible for the data exchange between groups

- Participant Leader:
  In charge of 2 phase commit and ensure consistency of transaction.

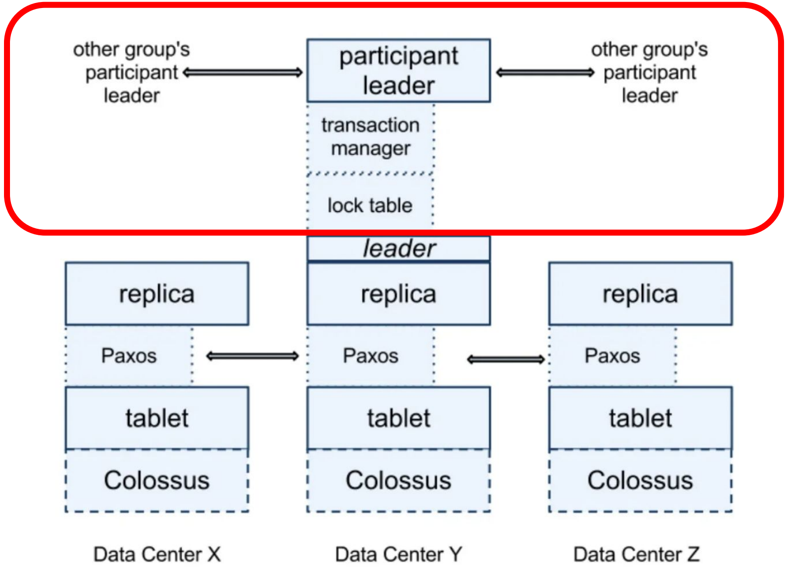


Figure 2: Spanserver software stack

Directories and Placement

A directory is the unit of data placement. All data in a directory has the same replication configuration. When data is moved between Paxos groups, it is moved directory by directory.

Pros and cons of directory movement:

1. Balance load between different paxos group
2. Lower latency of data read & write by moving directory to a paxos group that are closer to the client.

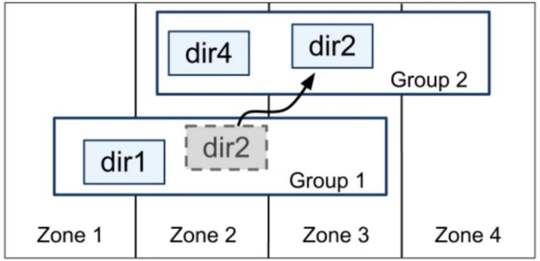
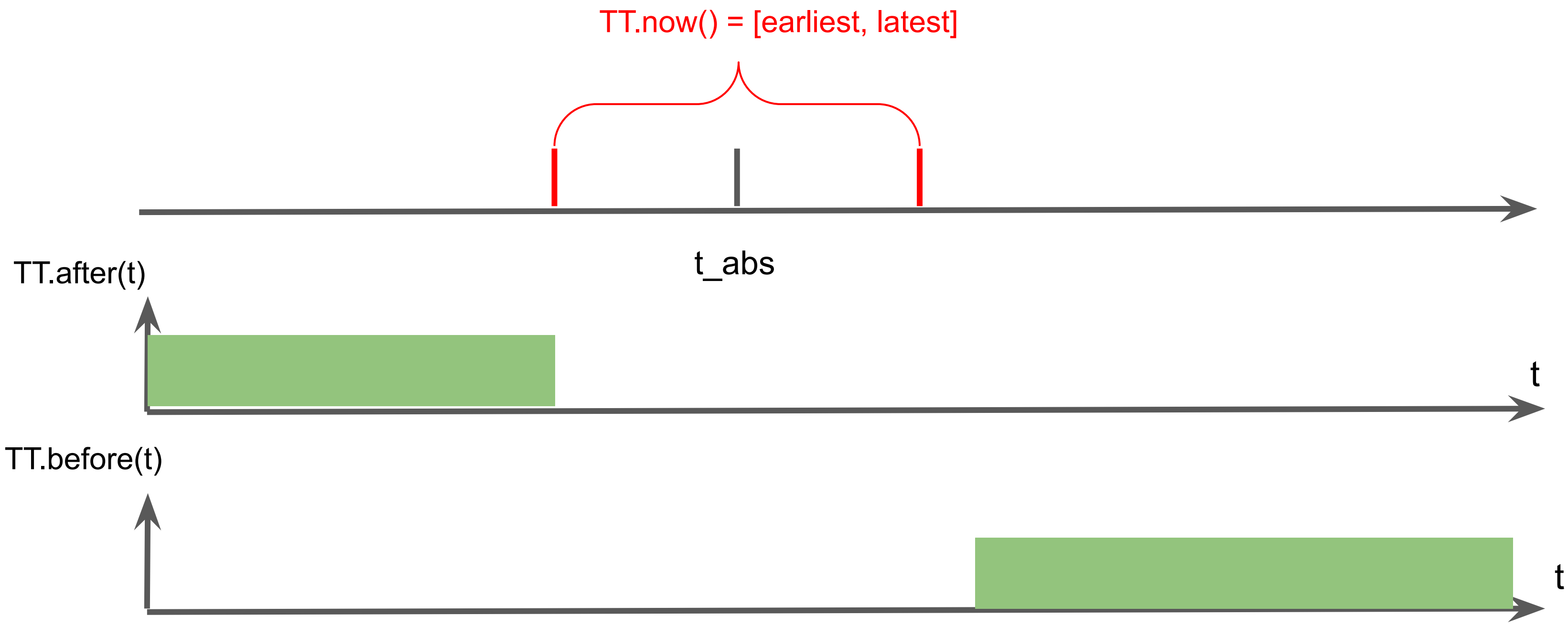


Figure 3: Directories are the unit of data movement between Paxos groups.

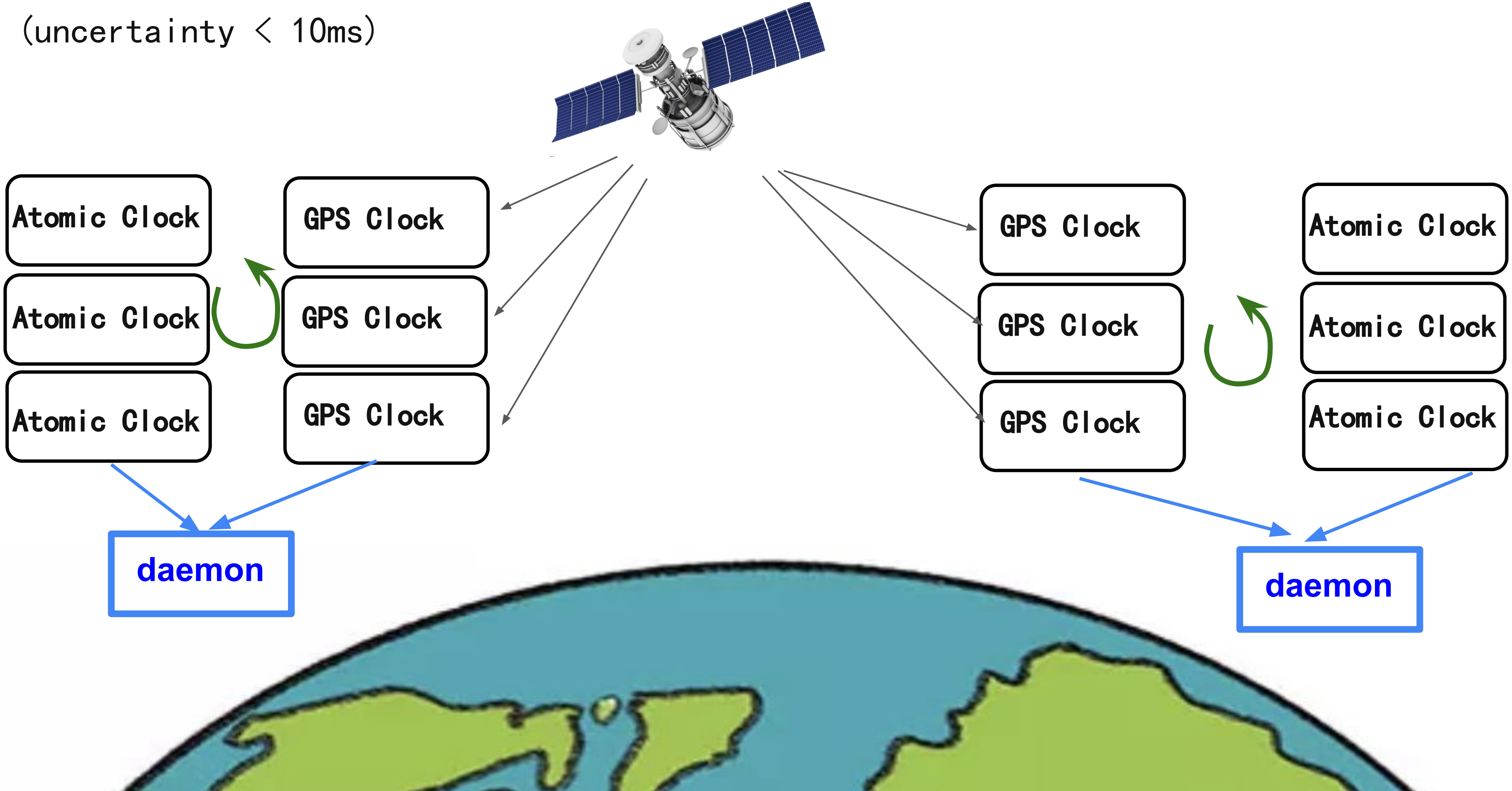
TrueTime API



TrueTime API

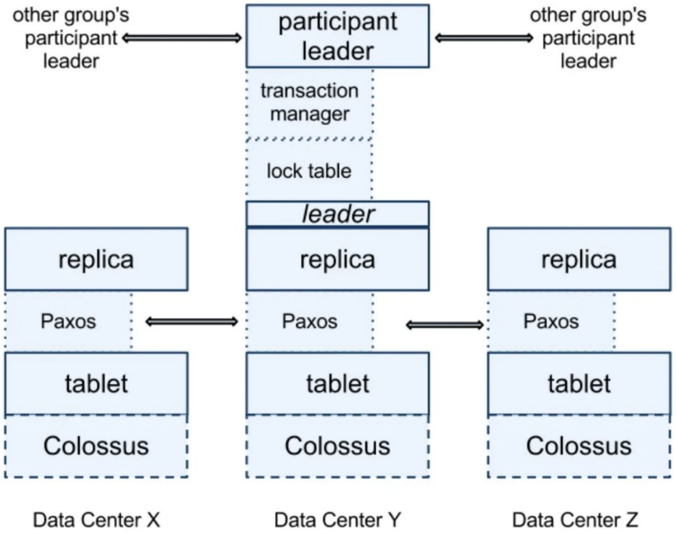
(uncertainty < 10ms)

Achieved by GPS and atomic clock



Assign Timestamp

Given transaction, Spanner assigns it the timestamp that Paxos assigns to the Paxos write that represents the transaction commit.



External Consistency Requirement

If a transaction *e*1 commits before another transaction *e*2 starts, then *e1*'s commit timestamp is smaller than *e2*'s:

s1 → e1

s2 → e2

t_abs(e1_commit) < t_abs(e2_start) → s1 < s2

External Consistency Rules

$t\_abs(e1\_commit) < t\_abs(e2\_start) \longrightarrow s1 < s2$

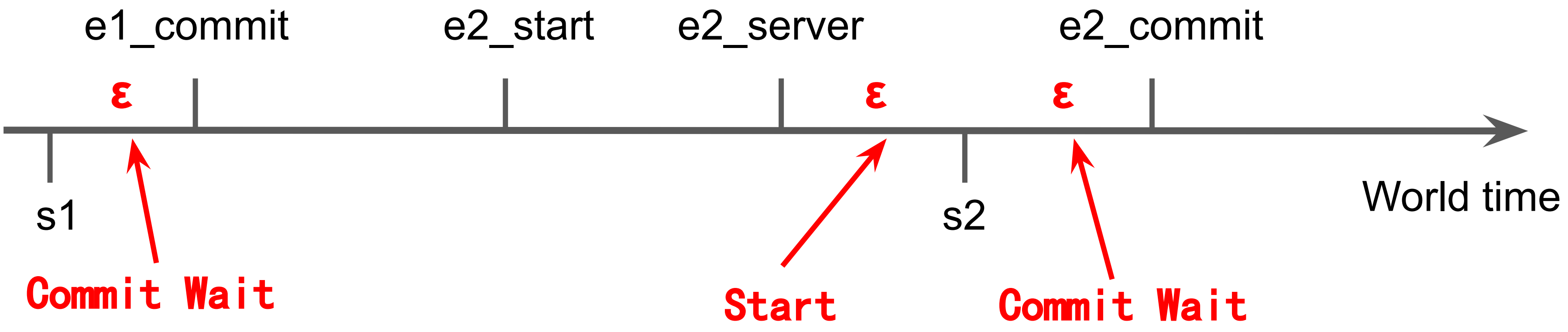
Start: When coordinate leader recv commit request (ei_server), assign si to ei with si > TT.now().latest

Commit Wait: After si is assigned, commit the transaction (ei_commit) when TT.after(si) comes true.

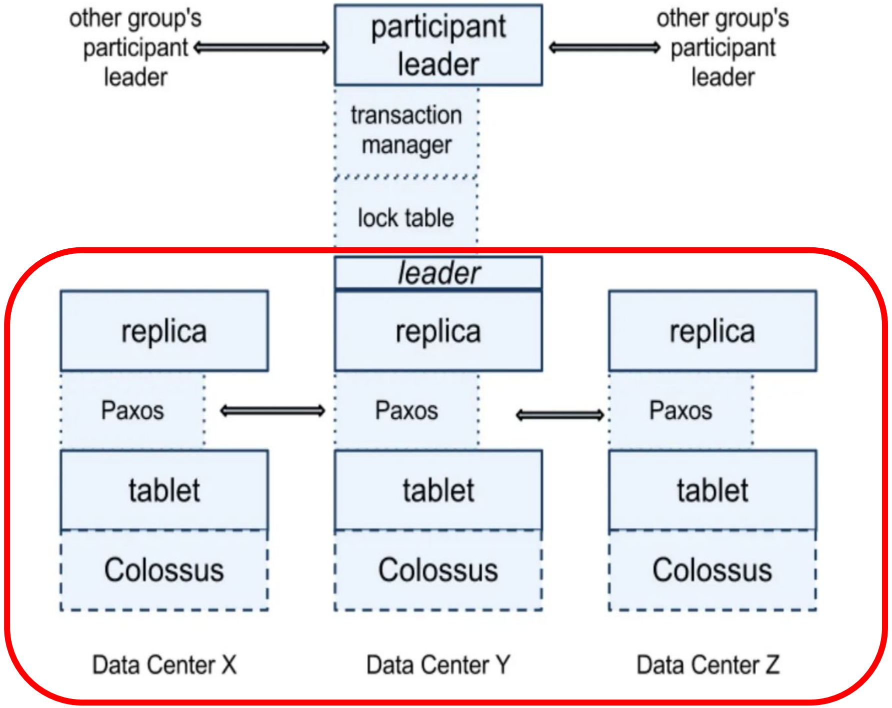
External Consistency Achievement

t_abs(e1_commit) < t_abs(e2_start) ⟶ s1 < s2

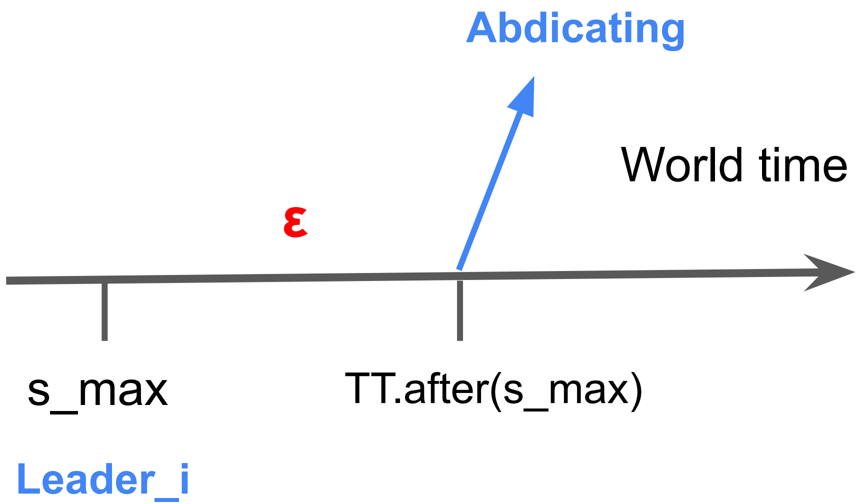
Coordinate leader:



Monotonicity Invariant



A leader must only assign timestamps within the interval of its leader lease.

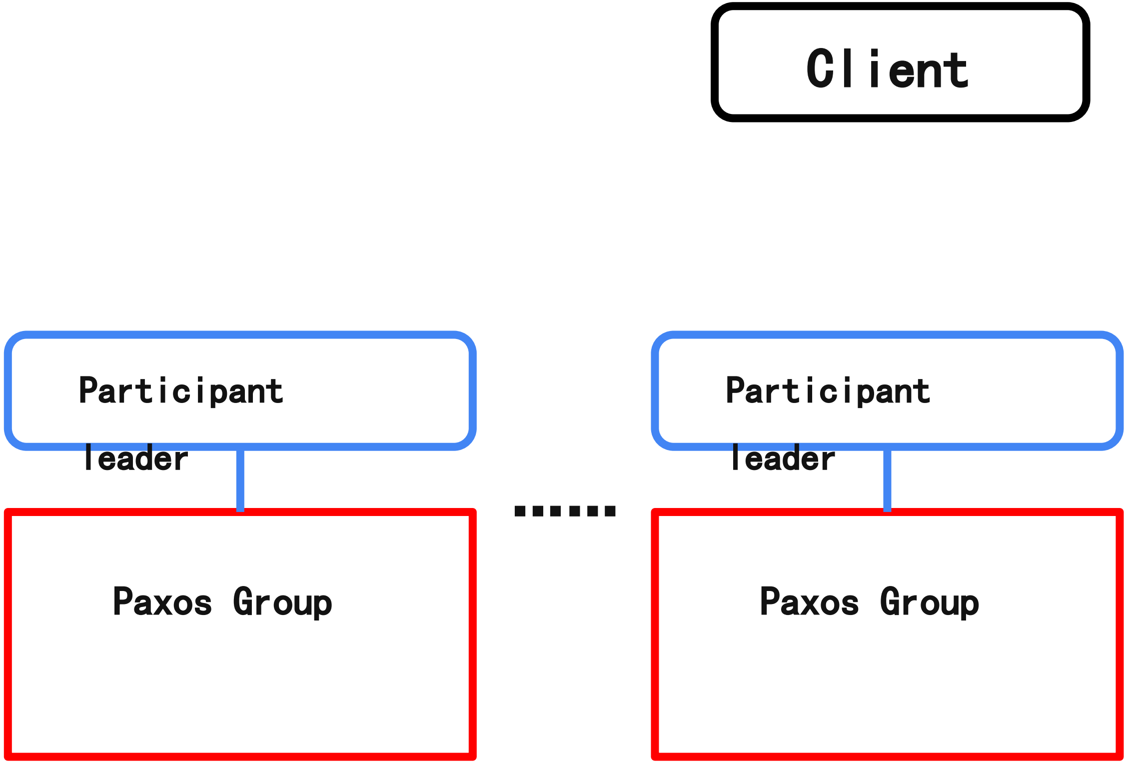


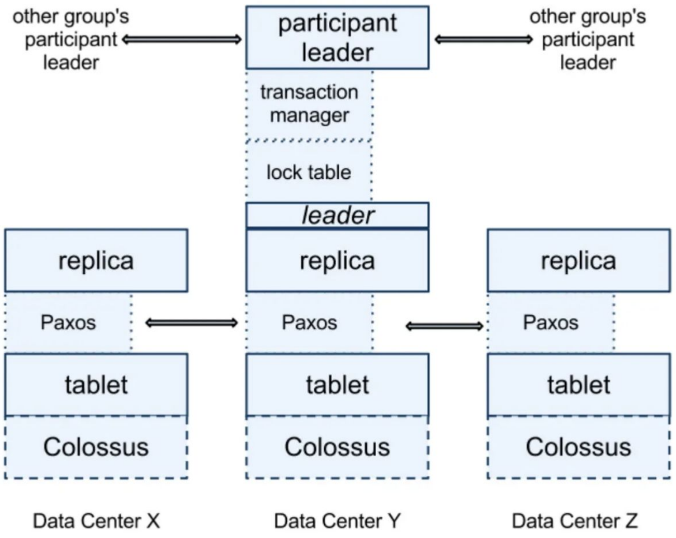
Concurrency control

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

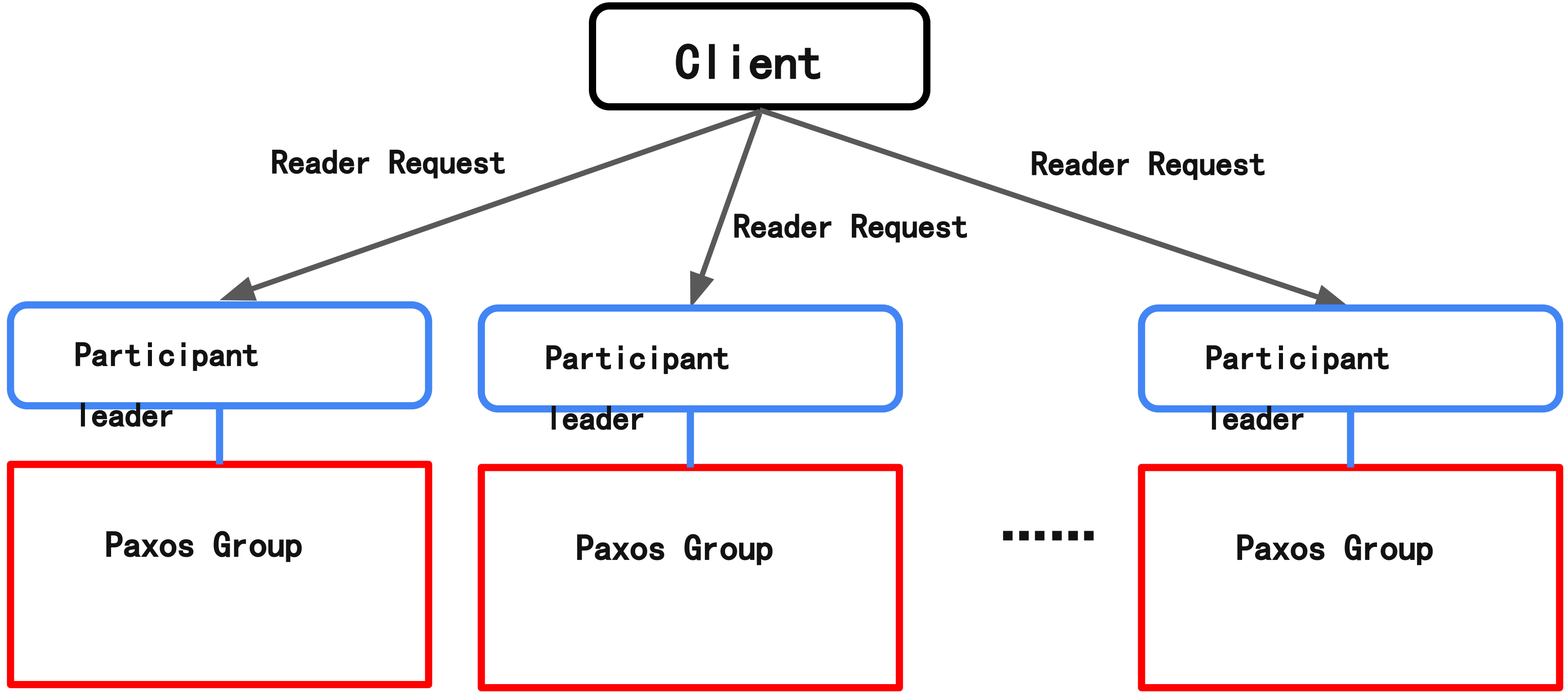
Operation	Timestamp Discussion	Concurrency Control	Replica Required
Read-Write Transaction	§ 4.1.2	pessimistic	leader
Snapshot Transaction	§ 4.1.4	lock-free	leader for timestamp; any for read, subject to § 4.1.3
Snapshot Read, client-chosen timestamp	—	lock-free	any, subject to § 4.1.3
Snapshot Read,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

Read-Write Transaction:

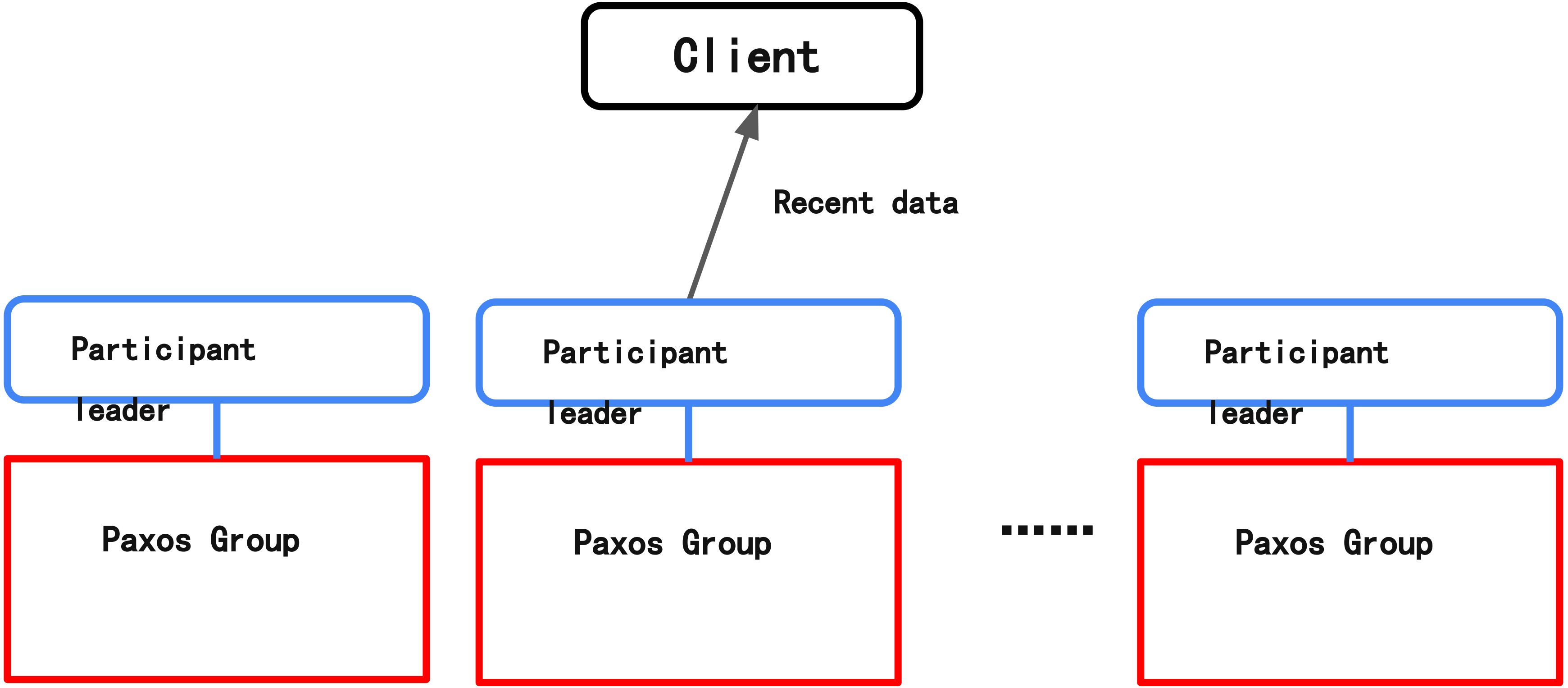


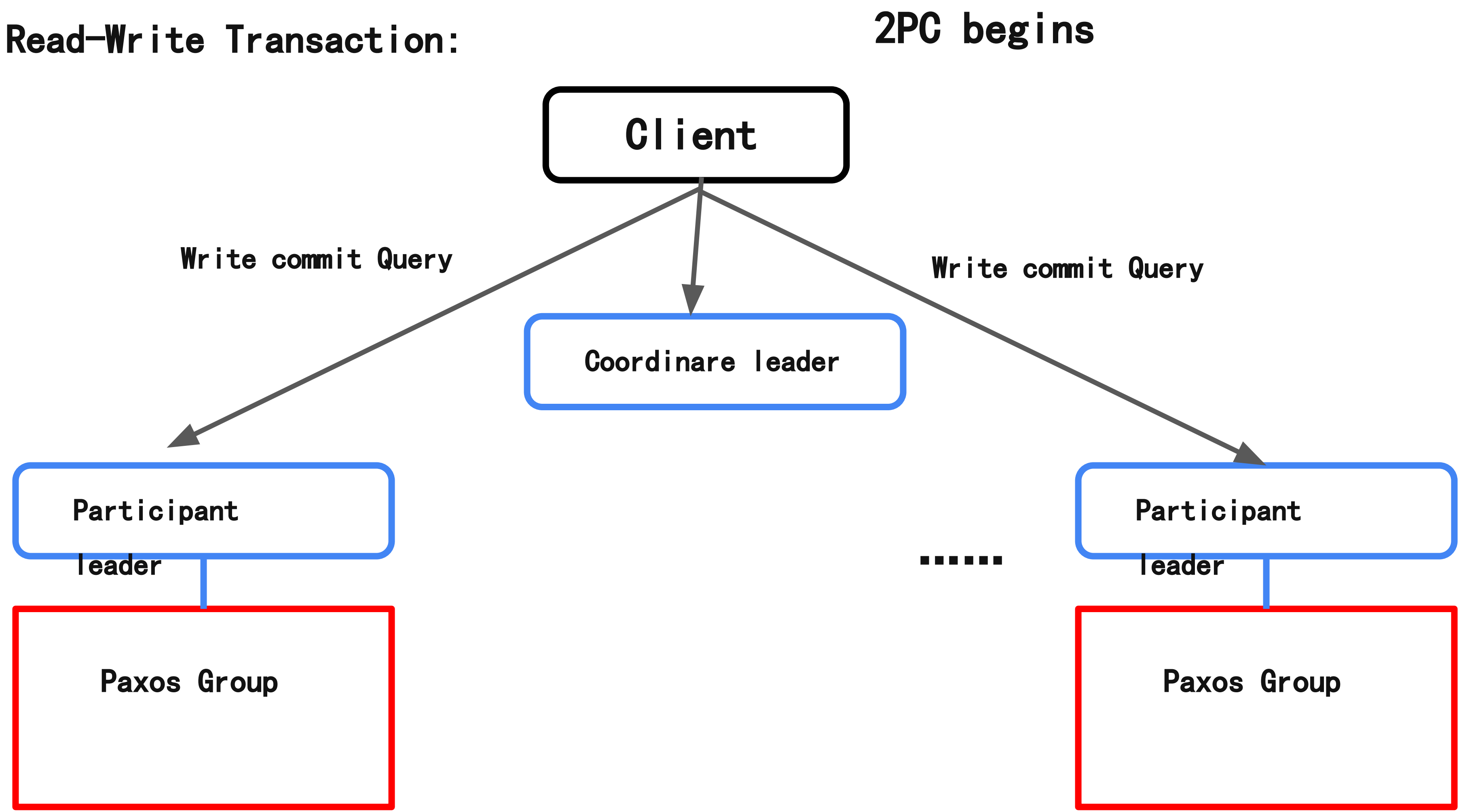


Read-Write Transaction:



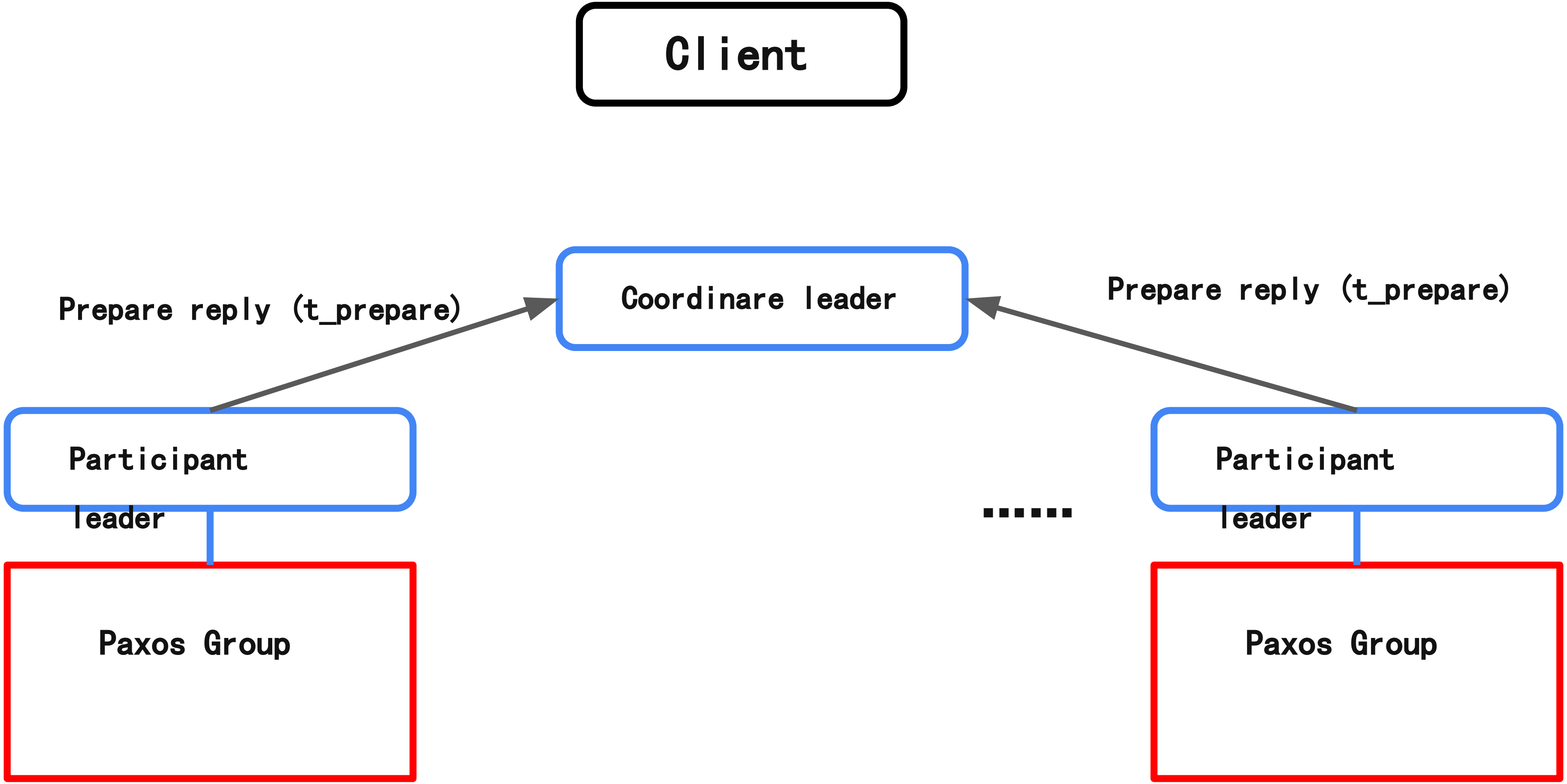
Read-Write Transaction:




Read-Write Transaction:
2PC begins
Client
Write commit Query
Write commit Query
Coordinare leader
Participant
leader
Participant
leader
......
Paxos Group
Paxos Group

Read-Write Transaction:

2PC begins



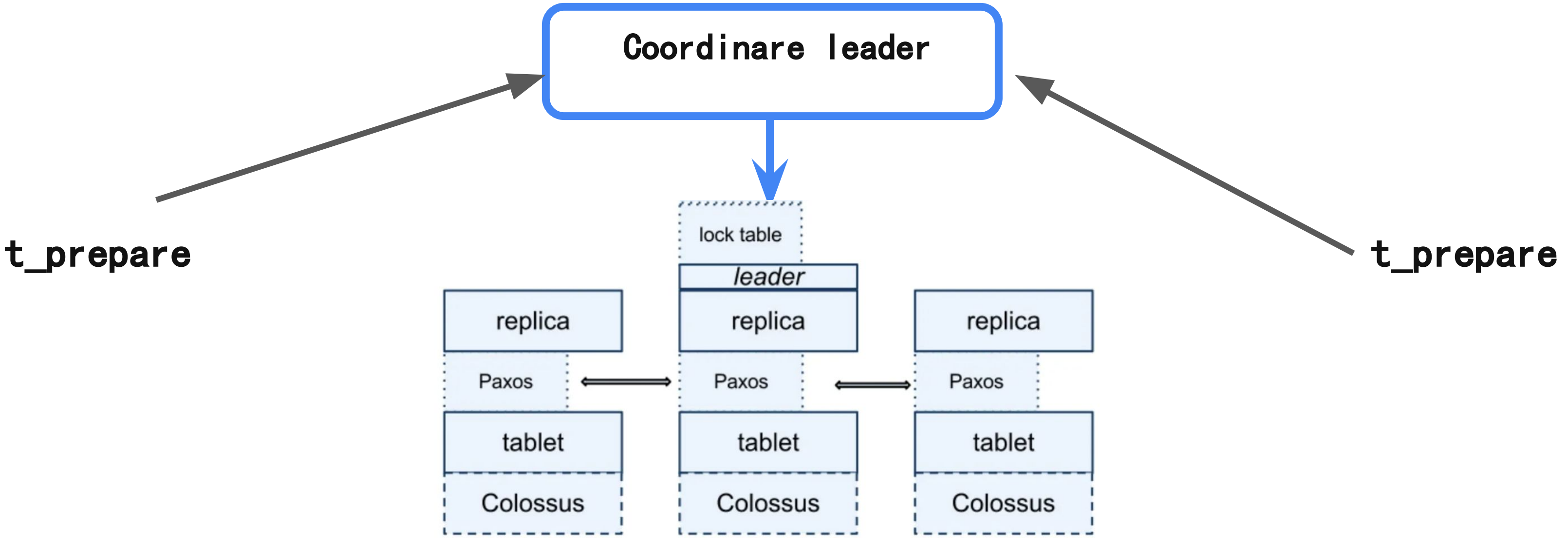
Read-Write Transaction:

Assign timestamp and commit

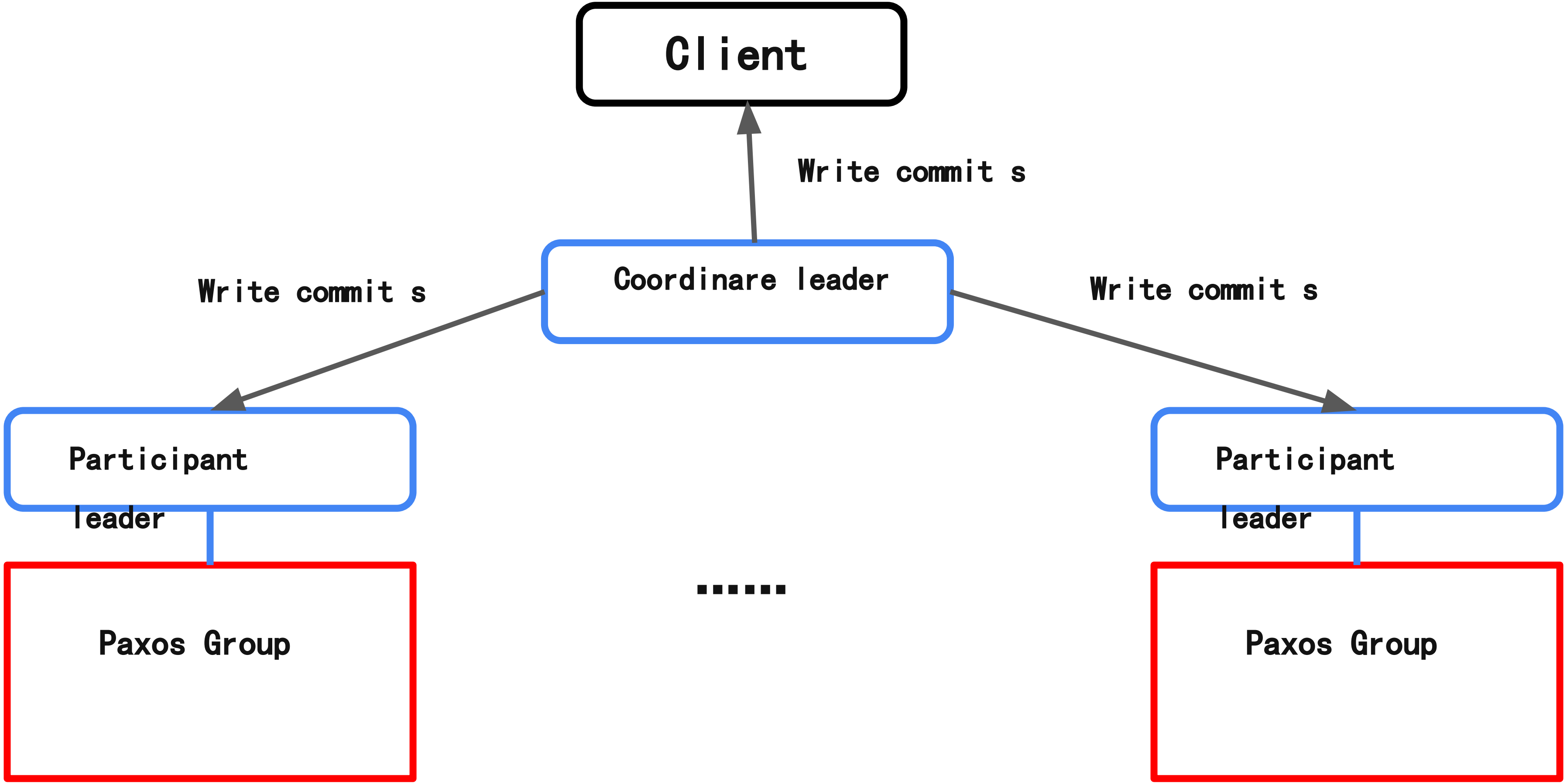
wait

Client

$$s > \max\left(\max_g\left(t^g_{\text{prepare}}\right), \text{TT.now().latest}\right)$$



Read-Write Transaction:



Thank you