# Detailed Design of Spanner

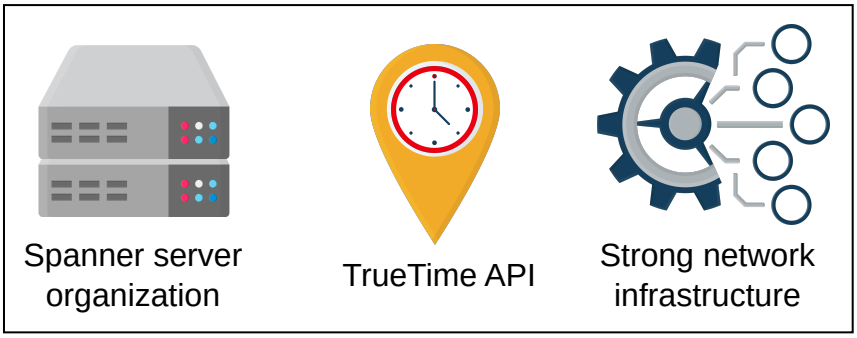
Learn about different components of the Spanner system.

**We'll cover the following**

- Components of the Spanner server organization
- The spanserver software stack
  - Using Paxos
  - Concurrency control via lock table
  - Transaction manager

**Universe** is the term for a Spanner deployment. Since Spanner handles data on a global scale, only a select few universes will be active at any given time. The components of Spanner are as follows:

1. **Optimized Spanner server organization:** This performs automatic resharding based on the data size and load and facilitates the client's request of read and write.
2. **TrueTime API:** This is an API that provides the time within well-defined error bounds. It can be used to ensure strong external consistency and global serialization.
3. **Strong network infrastructure:** We should have a redundant and highly available network that provides global connectivity to make high performance for Spanner possible.



First, we will understand how the components work together in Spanner's deployment. In the next lessons, we will learn about the TrueTime API, which makes Spanner unique. Moreover, we will also learn how a strong network helps Spanner to be a strongly consistent database.

## Components of the Spanner server organization

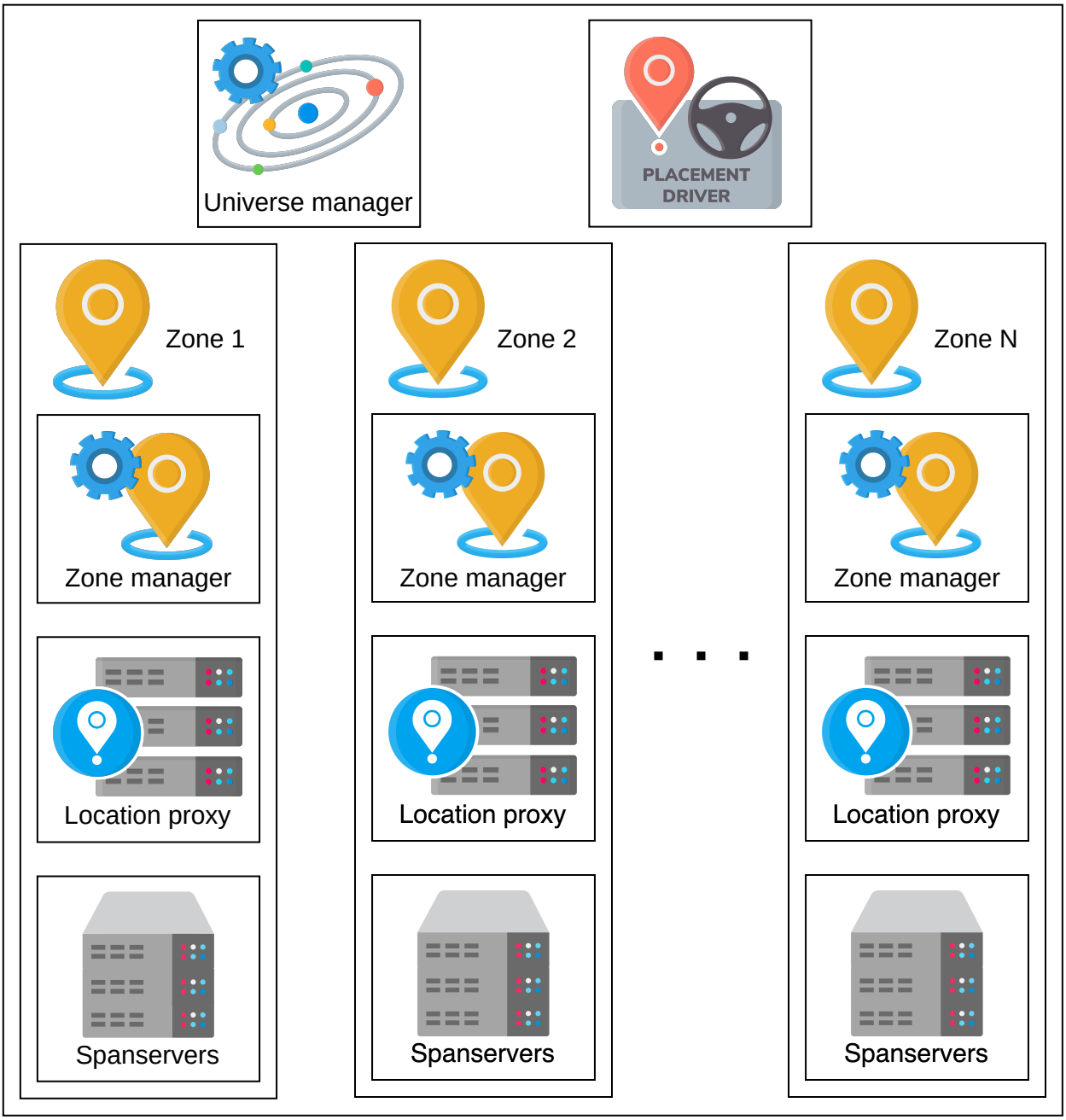
The components involved in the server organization of Spanner are as follows:

- **Zone:** A zone is a unit for administrative deployment. Both sets—the possible replication sites and the zones—are the same. As new data centers are brought online and older ones are shut down, zones can be added to or withdrawn from an active system. A zone is also a unit of physical isolation. One or more zones may exist within a data center, for instance, if data from various applications needs to be segregated across several groups of servers within the same data center. A zone consists of a single zone manager and is between a 100 to a few thousand servers.
- **Zone manager:** It manages a single zone and serves data to the clients.
- **Location proxy:** The clients use the per-zone location proxies to find the spanservers designated to serve their data. There is more than one location proxy per zone to avoid the bottleneck of requests directed to only one proxy.
- **Universe manager:** This is to facilitate interactive debugging. The manager functions as a console that shows the current state of all zones.
- **Placement driver:** It is responsible for the automated, minute-by-minute data transfer between zones. From time to time, it will communicate with the spanservers to determine if any data needs to be relocated to accommodate changed replication constraints or to distribute the workload better.
- **Spanserver:** It is responsible for serving data to the client. It consists of tablets. A Spanner tablet is a data structure like Bigtable’s tablet. Each server has between a 100 and 1000 instances of tablets.

> **Note:** Though Google calls the zone manager a "zone master" and the universe manager a "universe master" in their research paper on Spanner, we will use the terms "zone manager" and "universe manager" to refer to the same things.

The implementation of the universe manager and placement driver are both singletons. Only one instance will be actively handling the responsibilities of both. A shadow replica will take over if the universe manager or placement driver fails. Moreover, the failure of these components is not a single point of failure. It can slow down the transfer of data between zones.

The following illustration shows the placement of servers in the Spanner universe:




The Spanner server organization

## Points to ponder

Question 2

The implementation of the universe master and placement driver are both singletons. What happens if they fail?

Hide Answer

The singleton service will export only one instance of itself. However, that does not mean that failover will not happen in case of a failure.

On the other hand, the jobs of both the universe master and placement driver are such that a short outage, of a few seconds, for example, will not adversely impact the operations of the rest of the system.



## The spanserver software stack

We'll learn about the spanserver design to show how replication and distributed transactions are associated with Bigtable's design.

Each zone will function like a cluster of Bigtable servers. Each spanserver maintains several tablet instances ranging from one hundred to one thousand. Like the tablet abstraction of Bigtable implements a bag of mappings, so does a Spanner's tablet, as shown below:

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

Spanner and Bigtable offer internal routing and sharding as part of their fully managed service. Spanner is an extensible service for managing relational databases, including transactional consistency, SQL query support, and secondary indexes. While Bigtable is a NoSQL database that can scale horizontally. Spanner resembles a multi-version database more than a key-value store, and we want our data to be consistent and available. Since time is relative and the time of the servers cannot be trusted because of the Network Time Protocol (NTP), we will use the TrueTime API to provide us with a globally consistent timestamp. Moreover, we also need to store our write-ahead log. We will utilize a distributed file system called Colossus to store our data.

## Using Paxos

The Paxos consensus can ensure consistency. Each server implements a tablet with a single Paxos state machine, allowing for replication to take place. The metadata and logs of every state machine are stored in its tablet. The implementation of Paxos in Spanner supports time-based leader leases that enable long-lived leaders. The default lease time is 10 seconds. Spanner stores two copies of every Paxos write, one in the corresponding tablet and the other in Paxos logs. While Paxos applies writes in order of their receipt, Spanner's implementation of Paxos is pipelined to increase the database's throughput when WAN latencies are present.

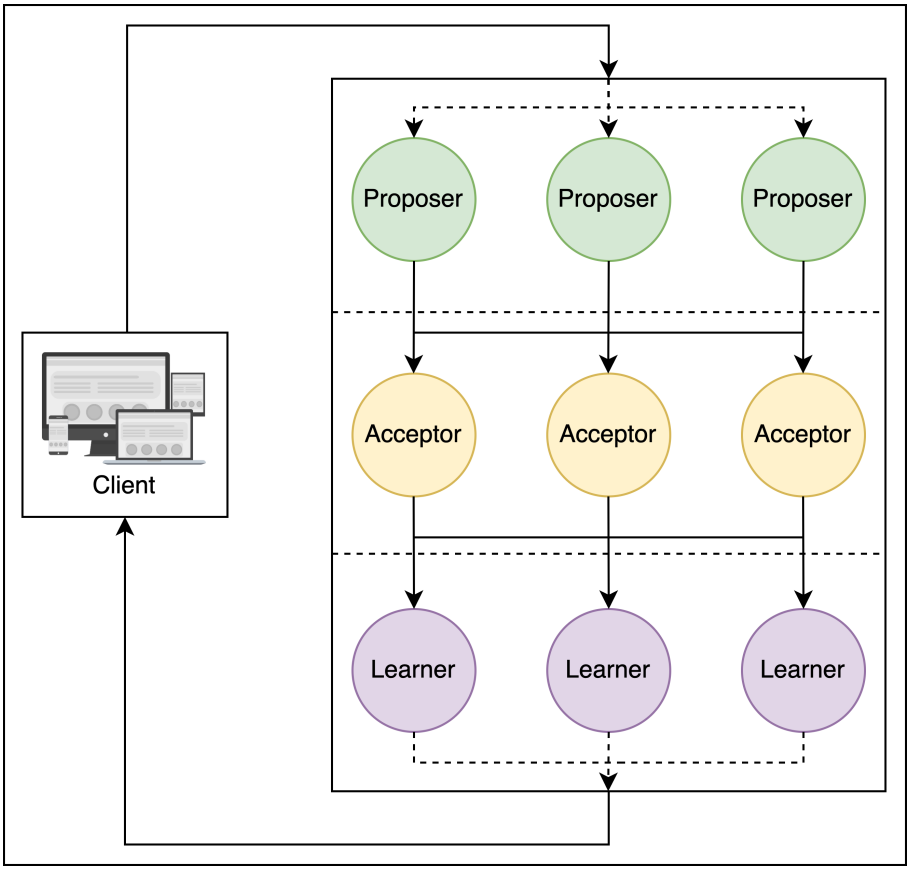
Hide me

### Paxos

The Paxos algorithm defines three different roles:

- **Proposers:** To reach a consensus, it is the proposer’s job to present values to the acceptors, which may have been obtained from client requests and try to persuade them to adopt the proposer’s value.
- **Acceptors:** The responsibility is on an acceptor to review the proposer’s suggestions and provide feedback on whether or not the proposed value is acceptable.
- **Learners:** The learners are accountable for acquiring knowledge about the consensus’s conclusion, storing it (in a replicated manner), and maybe acting on it by informing clients of the decision or taking some sort of action.

Each server in the system can serve multiple roles.




## The basic idea behind Paxos

The quorum is a crucial idea in the Paxos protocol. In particular, the majority of quorums are used in the Paxos protocol. In a system with $2k$ nodes, a majority quorum would be at least $k + 1$ nodes. Proposers need a majority quorum to move on with a proposal.

## Paxos in real life

In its most basic form, the Paxos protocol specifies how a network of computers in a distributed system can reach a consensus on a single value. However, the practical implications of selecting a single value are restricted as the overhead increases dramatically if each command requires its own copy of the basic Paxos protocol. Therefore, we need to run the Paxos protocol numerous times, leading to a different value being decided upon everytime. These instances need to be numbered, but they can run independently and in parallel.

We can apply numerous rules depending on the required functionality, like skipping the return of an instance to the client unless all the previous

instances have been entertained.

## Paxos supporting the read/write operation

The system users learn the chosen values to track the state on their side. However, there will always be use cases where newly implemented clients require access to previously selected values.

Paxos handles both the write and read operations, where the former initiates the new instances of the protocol and the latter returns the decisions of already finished instances.

These reading operations must be transmitted to the system's current leader, the node that implemented the previous proposal successfully. This leader node is unable to communicate with the client using the local copy. This is because the reply may not accurately reflect the system's current state if another node has proposed and has become the new leader in the meanwhile.

Therefore, the consensus procedures of reading and writing would not be linearizable. Proposals are examples of operations that fall into the "single-object operations" category while discussing consensus. Since this is the case, isolation is not needed to be guaranteed.

To view any prospective new proposal from another node, that node must read from the majority of nodes. The performance may suffer as the reads will be executed in two phases.

## Manager/leader leases

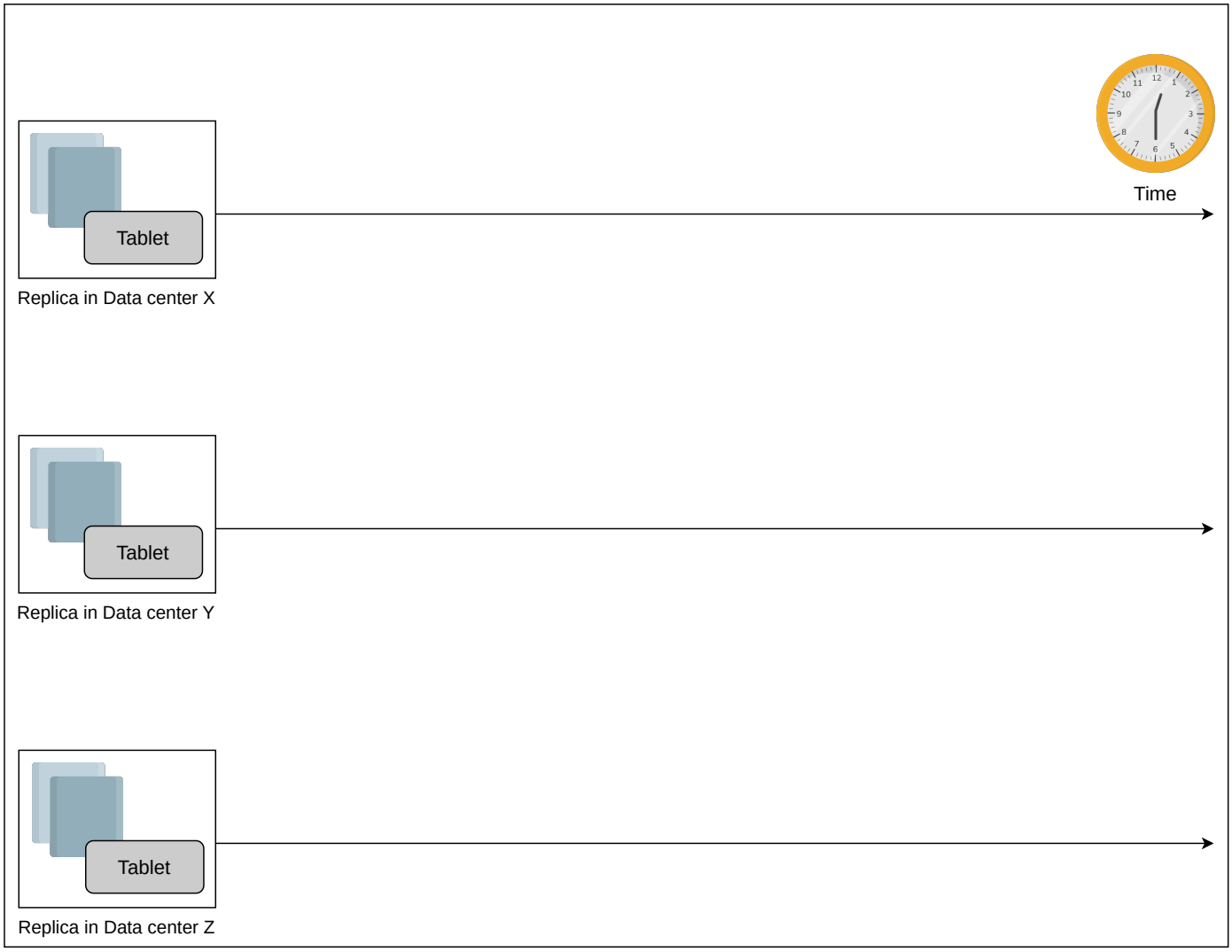
Lampson says that "An alternative option that works as optimization is to make use of the so-called manager/leader leases."

Through this method, a node can guarantee its position as a leader for a predetermined amount of time by running a Paxos instance and signing a lease until that time has elapsed. As a result, this node can now perform read operations locally. However, this can affect availability if the leader dies and

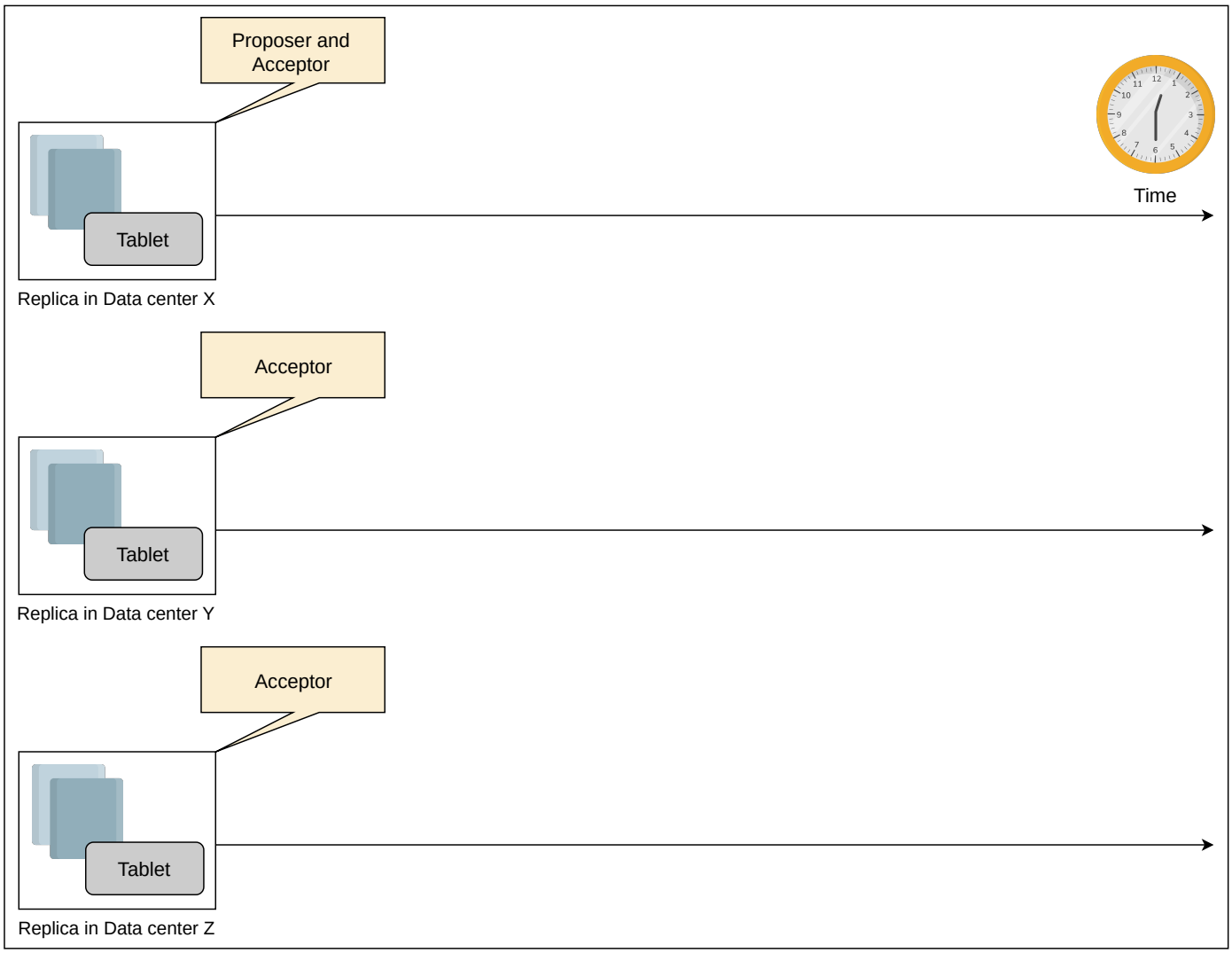
all the participants have to wait for the lease to end. However, we can use services like Chubby to cater to this.

It is important to remember that this method will only be secure if the clock skew is constrained to a certain upper bound when putting it into action.

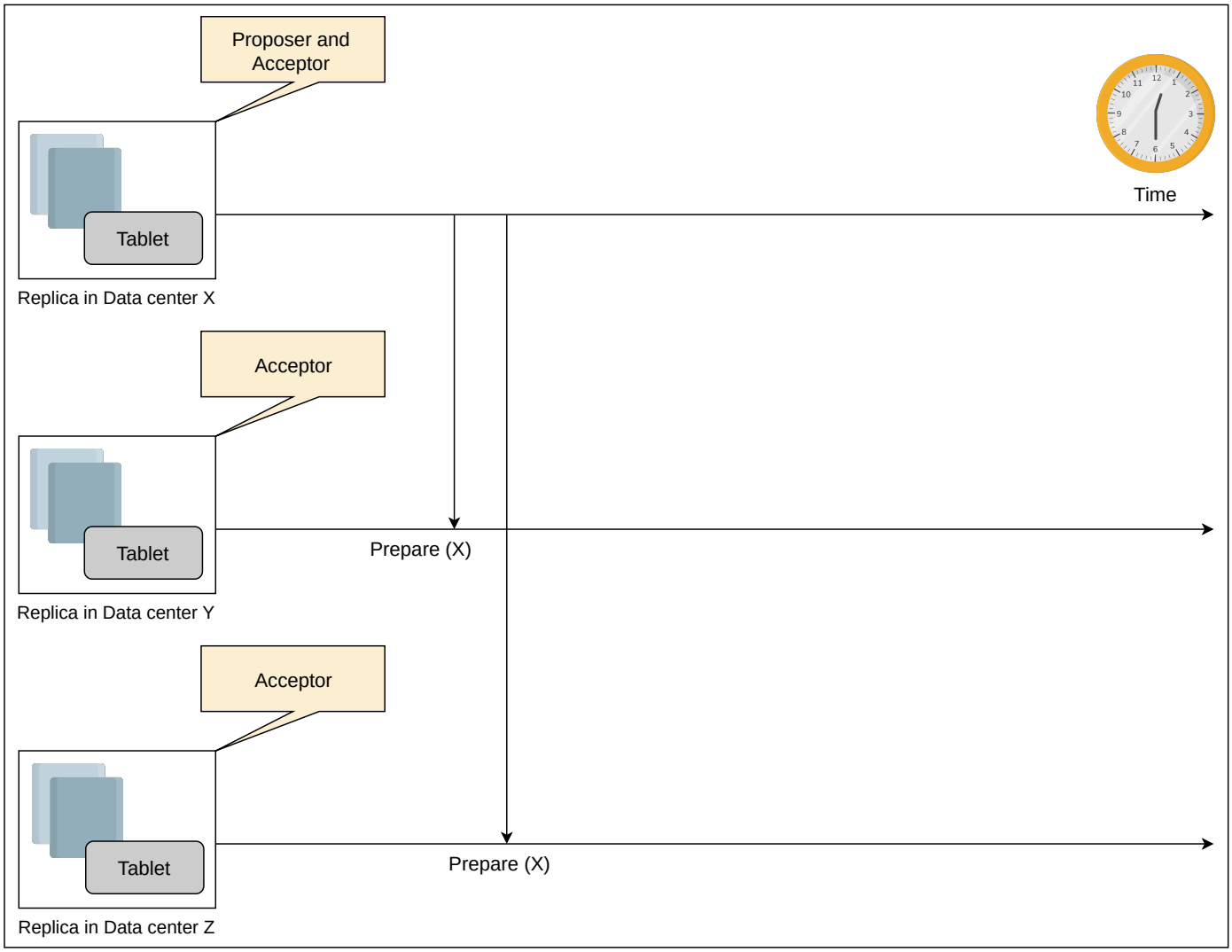
The following slides illustrate how Spanner uses Paxos:



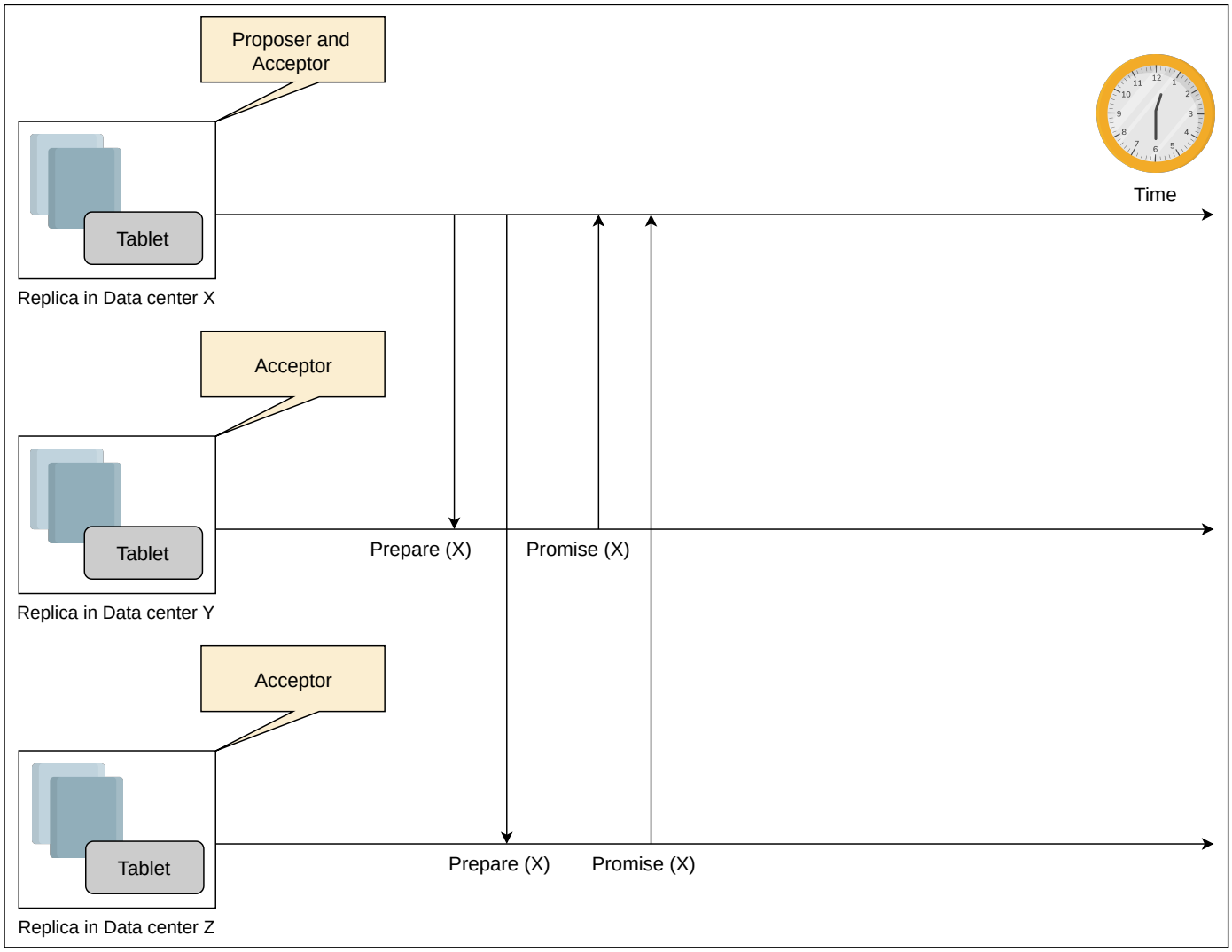

Three replicas in the Spanner universe



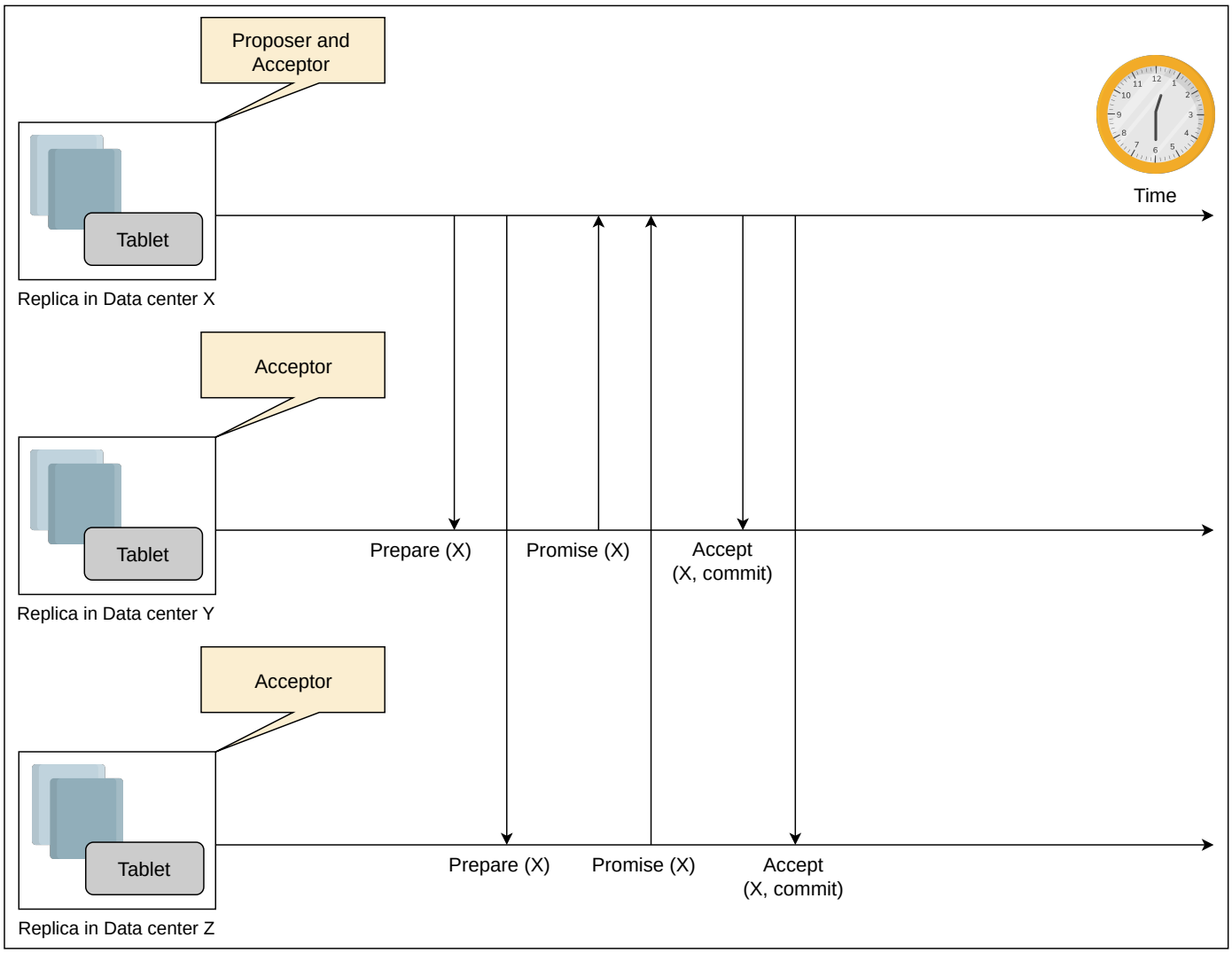

Tablet in the replica of data center X plays two roles (the Proposer and the Acceptor) while the other two are acting as Acceptors



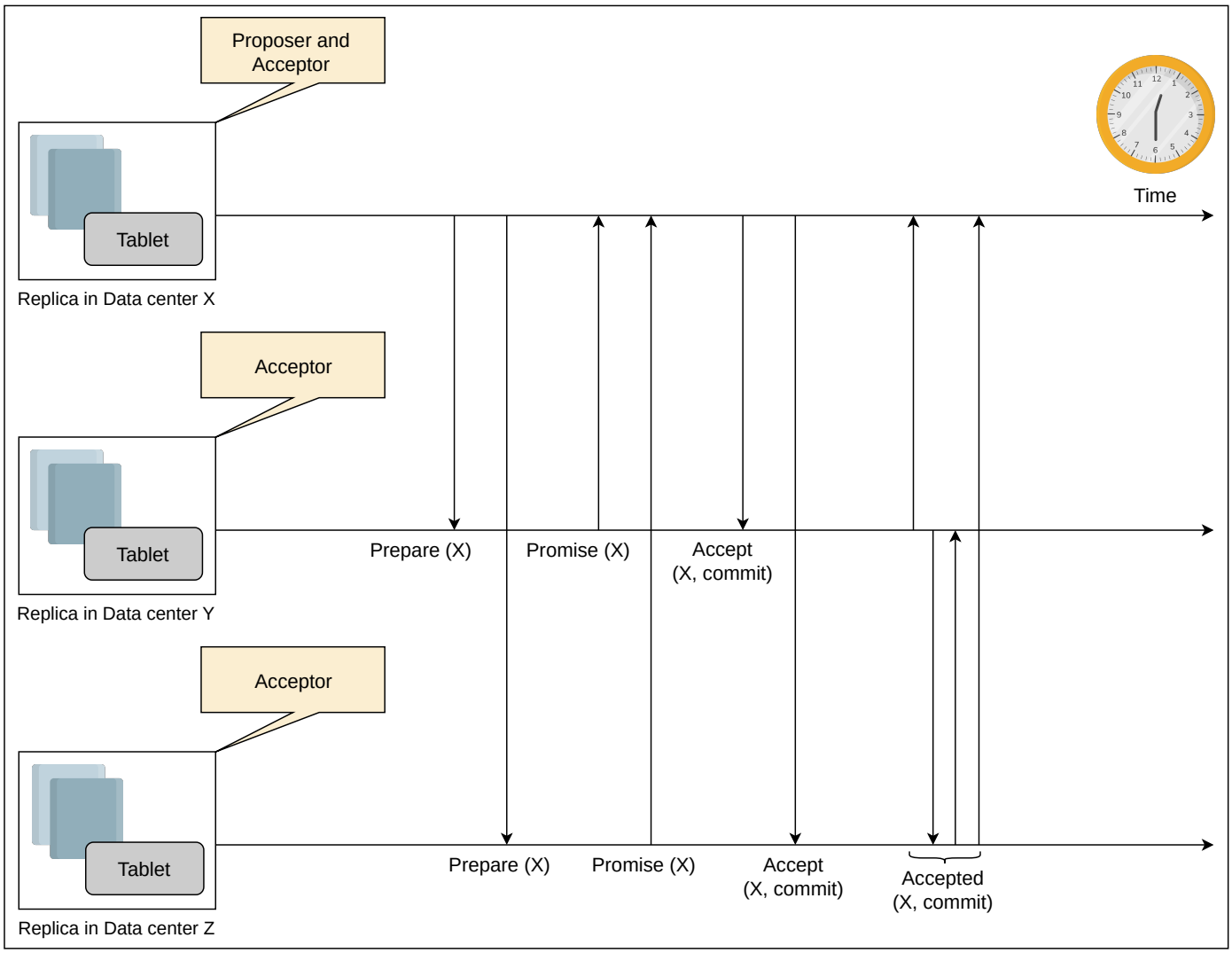

Proposer sends a Prepare (X) request to at least a majority of the Acceptors



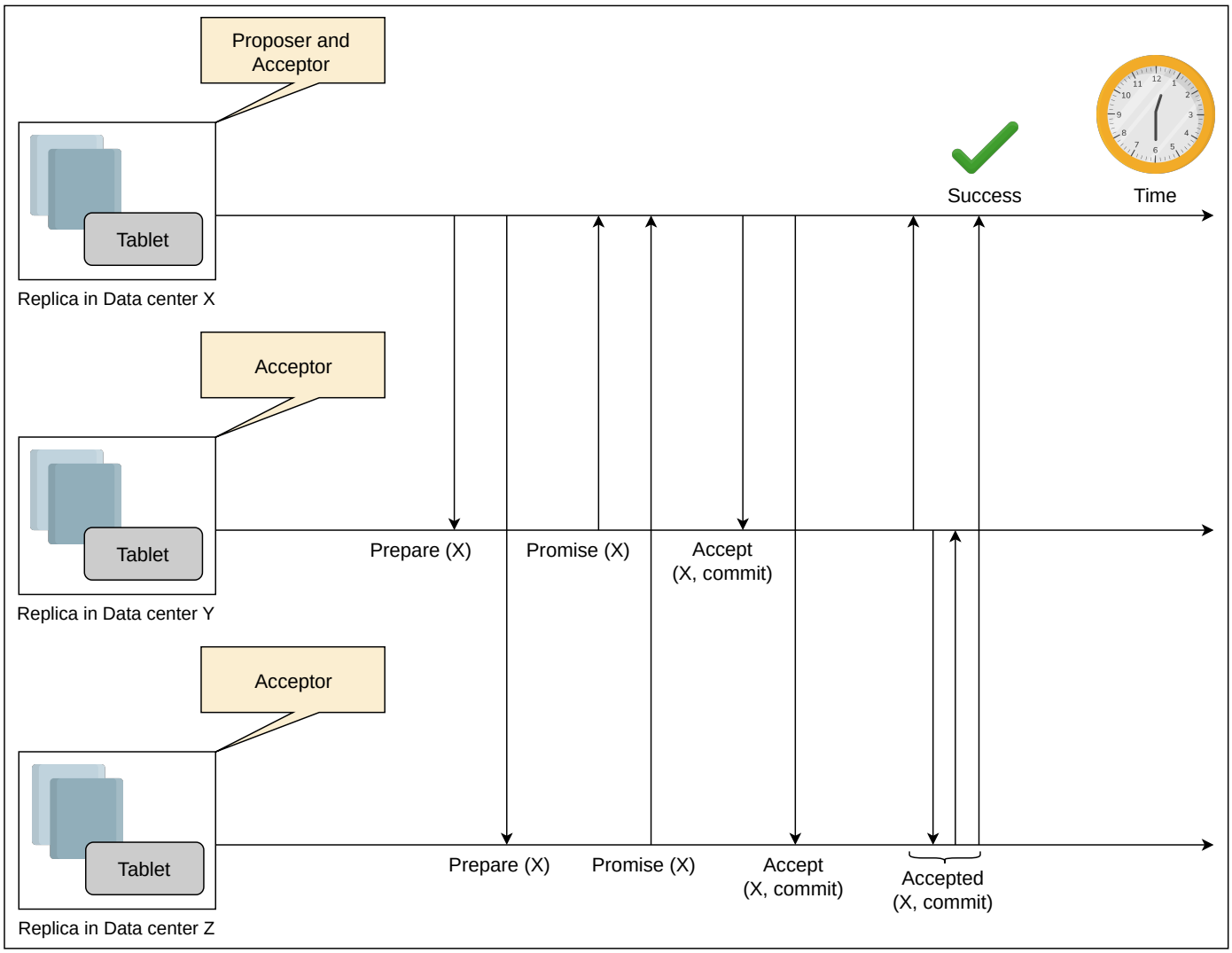

Acceptors respond to the Prepare request




The Proposer starts the commit phase and sends the Accept request to all the Acceptors that promised on value X




Acceptors respond to the Proposer either to commit or not and also communicate about its decision to the other tablets involved


Proposer and Acceptor
Tablet
Replica in Data center X
Success
Time
Acceptor
Tablet
Replica in Data center Y
Prepare (X)
Promise (X)
Accept (X, commit)
Acceptor
Tablet
Replica in Data center Z
Prepare (X)
Promise (X)
Accept (X, commit)
Accepted (X, commit)

Points to ponder

Question 1

What would happen if two zones are in the same data center and that data center is taken down due to a natural disaster?

Hide Answer

To avoid data loss during an outage of the data center, the users should replicate the data in multiple regions of different data centers. Spanner's regional configurations allow replicating the data between multiple regional zones. However, with multiple regions, the data is saved at physically different places. Moreover, the availability of the data in the chosen region also improves.



Points to ponder

Question 2

The implementation of the universe master and placement driver are both singletons. What happens if they fail?

Hide Answer

The singleton service will export only one instance of itself. However, that does not mean that failover will not happen in case of a failure.

On the other hand, the jobs of both the universe master and placement driver are such that a short outage, of a few seconds, for example, will not adversely impact the operations of the rest of the system.