

# Internals of Google Cloud Spanner



Bhuvanesh

Follow

Feb 4, 2020 · 9 min read



## Cloud Spanner

Image Credit: Google Cloud

We have learned a lot more internal things about Google Cloud Spanner from past two days. We read some of the portions of the Spanner white paper and the deep internal things from the Google Cloud Next event videos from Youtube. We shared the video links here, but we want to summarize all the learnings in one place. A special thanks to Deepti Srivastava(Product Manager for Spanner) who presented the Spanner Deep Dive sessions in the Google Cloud Next Event.

## The Pain with MySQL:

In 2005, 2006 Google was using the MySQL at massive scale. Google Adwords is one the biggest platform where 90+ MySQL Shards are used to store the data. Due to some maintenance, they re-sharded the MySQL Clusters. This process took 2 years to complete. Google understood that they are growing very fast and these kinds of databases will be a pain in future. That is how Spanner was born.

## BigTable and Spanner:

Once they decided to build something new with distributed, the Big Table team was the one who started working for the Spanner process. Because BigTable uses distributed process, storage and highly available(or maybe some other reasons as well).

## Colossus:

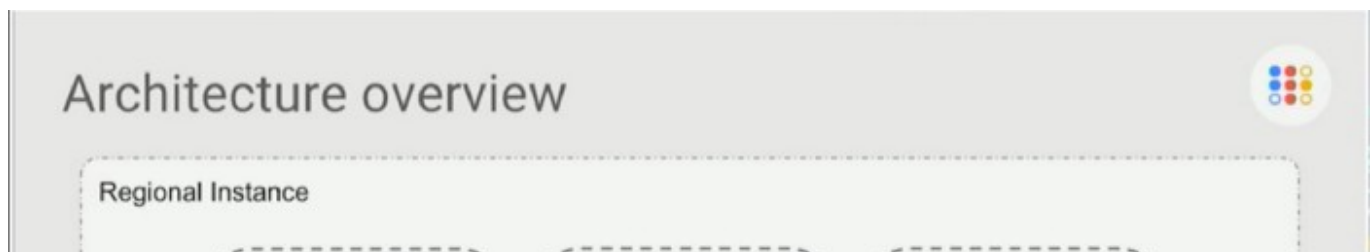
Colossus is the distributed file system which is derived from the GFS. A high performance file system is needed for a super database. This project started by BigTable team and the BigTable is powered by Colossus. So Spanner also got the colossus as a filesystem.

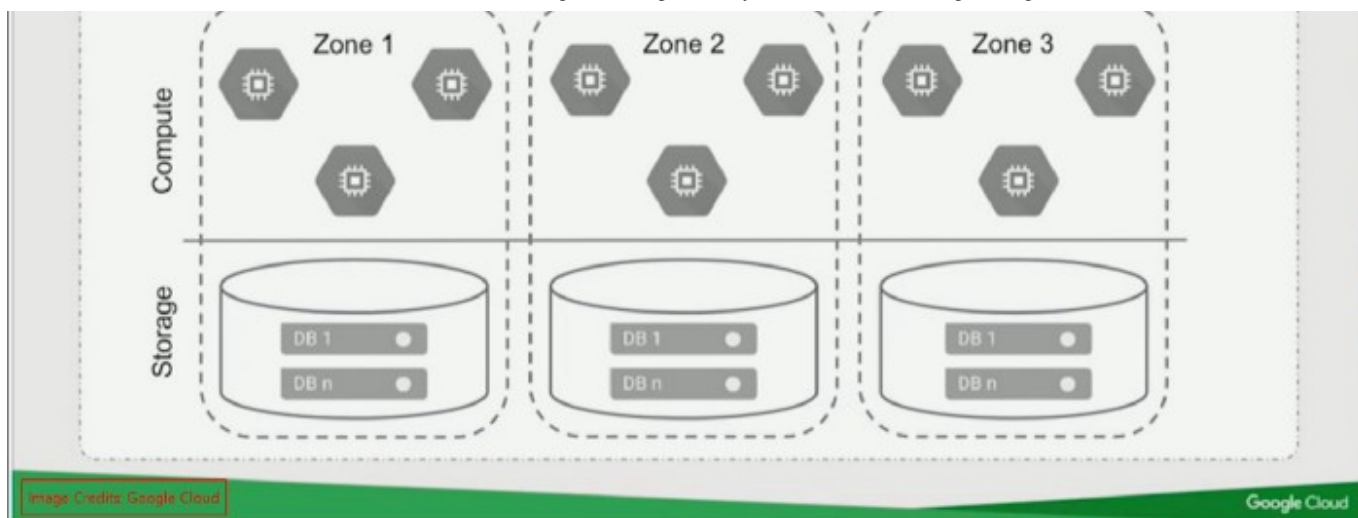
## Why Spanner?

The Google Adwords is MySQL based stack and the new system required to have essentials of a relational database like ACID compliance without the limitations of scale. The pain with MySQL is resharding. So they wanted the sharding features like the traditional NoSQL sharding that will take care of resharding and rebalancing. Plus more availability, Horizontal Scale and globally distributed.

## Spanner Architecture:

Spanner is a global database system, per region we'll get a minimum of 3 shards. Each shard will be in each zone. In Spanner terms, a shard is called Split. If your provision 1 Node Spanner cluster, you'll get 2 more Nodes on the different zone which are invisible to you. And the Compute and Storage layers are de-coupled. Paxos algorithm is used to maintain one leader at a time and the rest of the nodes will be the followers.





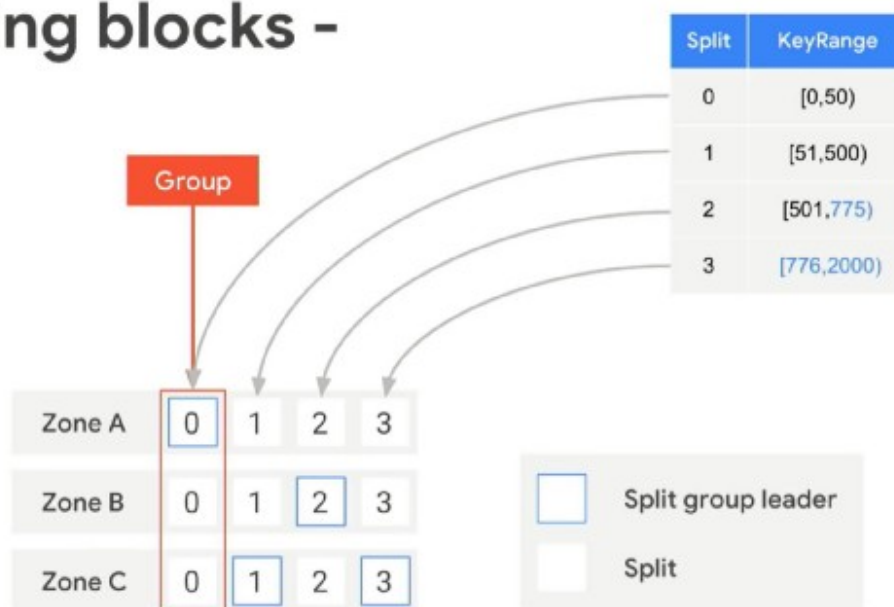
Based on the partitions, we'll have more Splits(shards) in the storage layer. Each shard will be replicated to the other Zones. For eg: if you have a shard called S1 on Zone A, it'll be replicated to Zone B and C. The replication works based on Leader follower method. So the Paxos will help to maintain the quorum and will help to select a new Leader during the failure. If you are writing something on this Split, the Spanner APIs are aware of the Leaders. So the write directly goes to the Zone where it has the Leader Split. Each Split has its own leader zone.

## Primary building blocks - Data splits

Key Space is partitioned into splits

Each split is replicated across zones

Each split has an elected leader

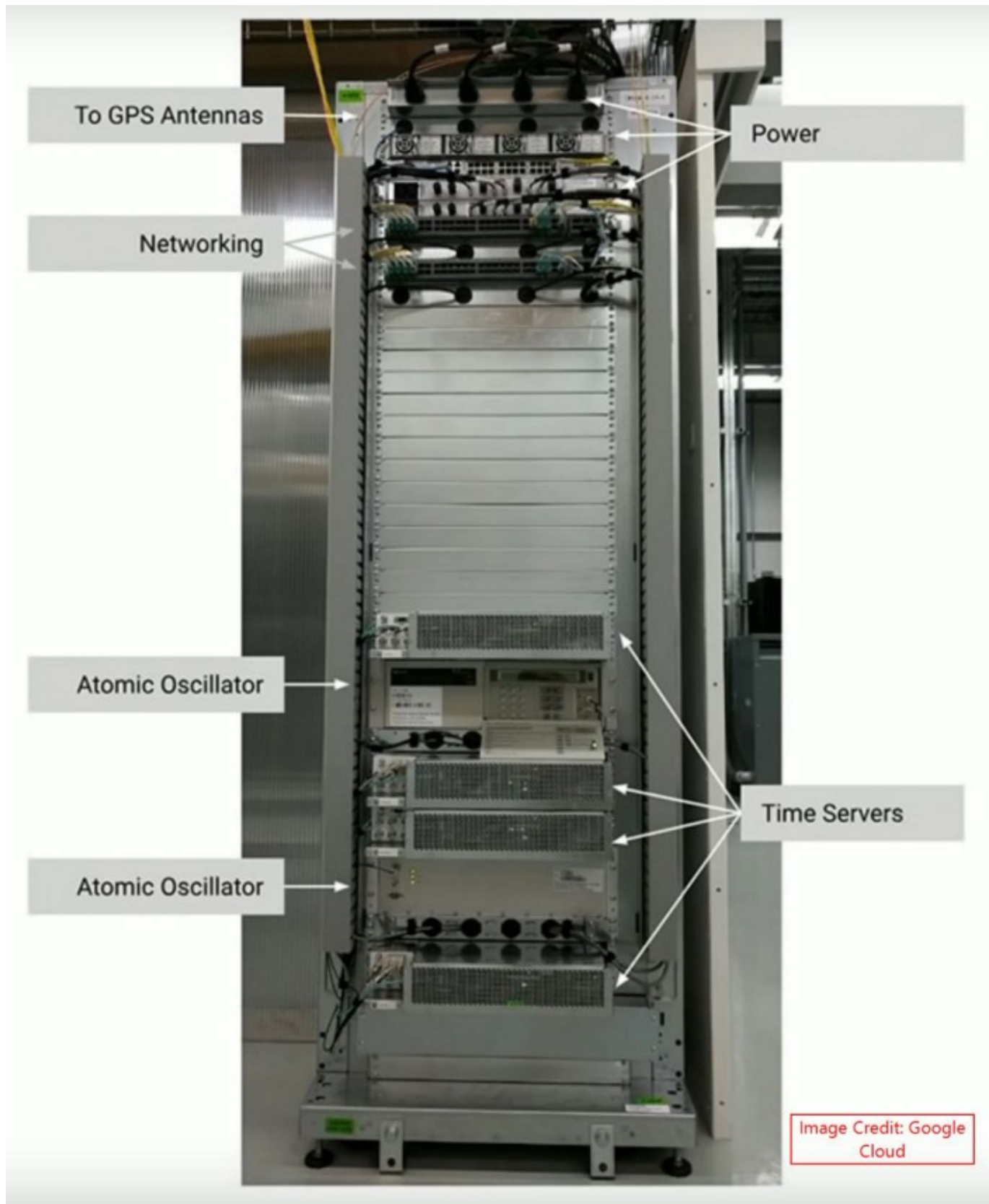


## Global strong Consistency:

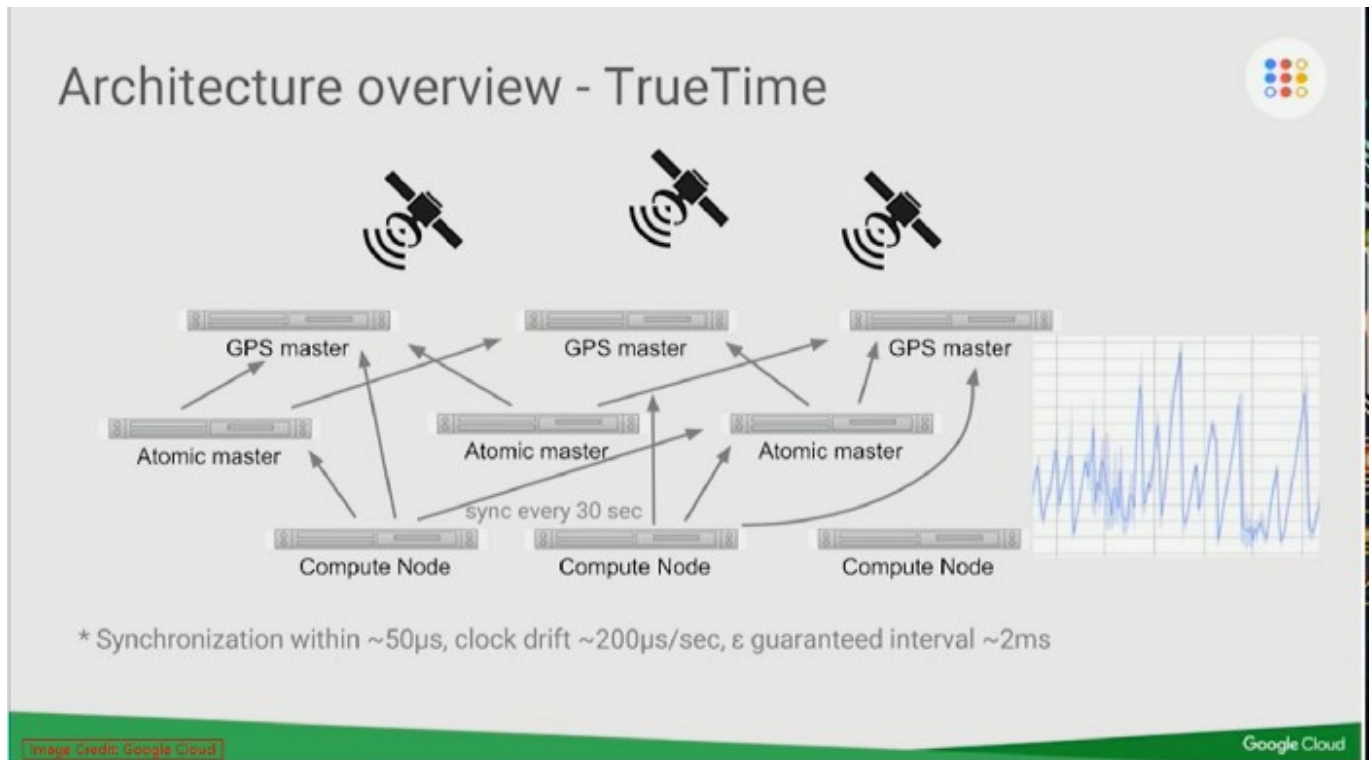
when I was watching the deep dive video of Spanner, they were discussing the strong consistency. Spanner supports strong consistency across all the nodes(Globally). If you

write something in US region, you can read that same data from the Asia region or any other region. How they implemented this logic? It's called TrueTime.

## TrueTime:



Spanner synchronizes and maintains the same time across all the nodes globally spread across multiple data centers. The hardware is built with Atomic Clocks to maintain the time. If you take a look at the Server Hardware Rack, the Server is having 4 time servers. 2 Servers are connected with GPS and the remaining 2 are connect with Atomic Oscillators. There are 2 different brands of Oscillators for better failover processing. The GPS time servers will sync with Oscillators to synchronize the time across the global data centers with every 30sec interval.



Let's now try to understand how this TrueTime helps Spanner stay consistent.

## Consistency with TrueTime

To understand the relationship between consistency and TrueTime, we have to understand how a write operation works in Spanner. During every write operation Spanner picks up the current TrueTime value and this TrueTime timestamp will create an order for the write operations. So every commit has been shipped with a timestamp.

**For Eg:** If you are writing a data on Node 1, it'll commit the data with the TrueTime timestamp and replicate the data and timestamp to the other nodes. This timestamp is the same on all the nodes. Lets say we committed this data on Node 1, if you are reading the same data from the Node B, then the Spanner API will ask the leader of the Split for

last committed data's timestamp, if the timestamp is matching from the Node A's timestamp then the data will be returned from Node B, else it'll wait until the Node A sync the data to Node B and then it'll return the data.

## Lifecycle of a single row Write operation:

Here is the lifecycle of a single write operation. We are writing a row that will go to Split 2. Now the Spanner API will understand who is the leader node for Split 2, then the request will go to Zone B node (Blue indication refers to the leader). Then it'll acquire the lock write it on the split. Once this write has been done, it'll send the requests to Zone A and C Nodes to write the same. It'll wait for the acknowledgement from the majority of the nodes. Once the leader split got the majority of the acknowledgement, then it'll send the success response to the client.



## Multi Row write operation:

If you are writing the data in a single transaction, but the data resides on different splits, then the spanner will handle it in a different way. For eg: we have to update 2 rows.

- Row 1 is in Split1 — Zone C is the Leader Split
- Row 2 is in Split2 — Zone B is the Leader Split

When we initiate the transaction, the Spanner API will understand that the rows are in the different split. And they will randomly pick a Co-ordinator zone. In our example, the API has chosen Zone C is the coordinator zone. The following steps will be performed for the multiple row operations.




1. Select coordinator zone. (Zone C)
2. Acquire the locks on the data on both leaders splits at the same time.
3. Add the new data on both Leader splits. Leader Splits will replicate the new data to the follower splits. And then Get the acknowledgement from the follower splits (Both splits will wait to get the acknowledgement).
4. Then zone B split will send a message to the Coordinator zone's split that its done with the update and it's ready to commit.
5. Then the Split1 in zone C will tell to the Split2, go ahead and commit the data. Same time, Split 1 also will commit.
6. The commit request will go to all the splits(both Leader and follower) and commit the data permanently.
7. And then the success response will go the client.



## Life of a Read operation:

While reading the data from Spanner, data will be fetched from the nearest follower split. Let's explain this with an example. Refer the below image.

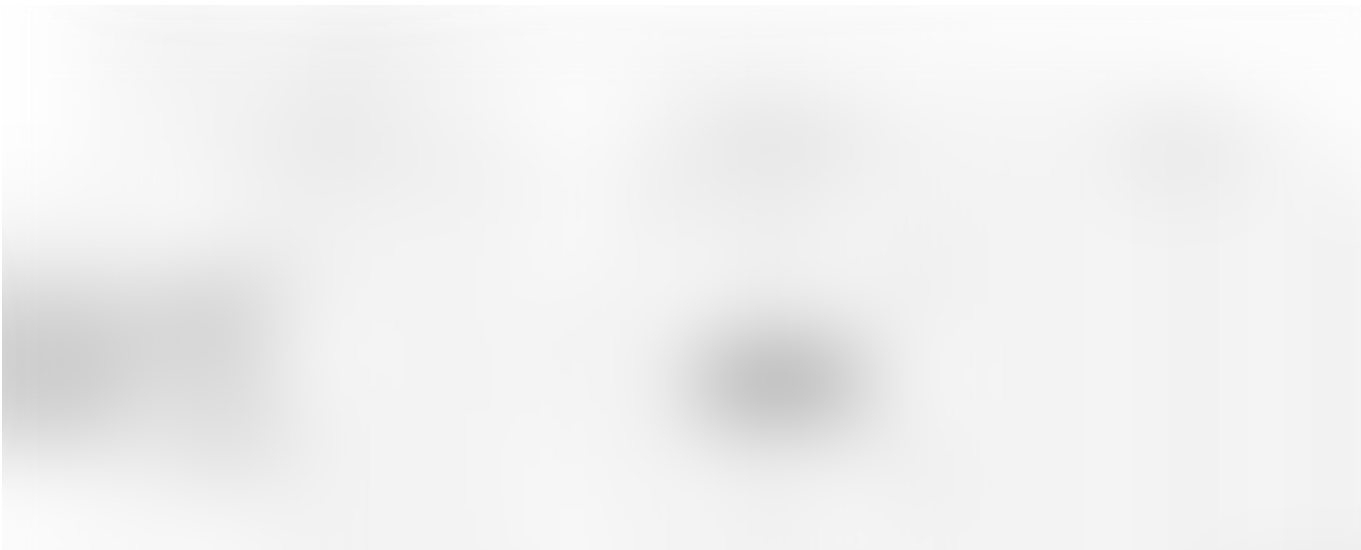




We want to read the data from MyTable, for the value 123. This value is stored in Split 2. Now once the request reached the Spanner Frontend server, then it'll understand who is the nearest follower split and forward the request to that split. In our case, Zone A is the nearest split. Once the request reached the split, then that split will ask to the Leader split to get the last committed TrueTime. And then it'll compare the Timestamp with its own timestamp. If both match then it'll serve the data to the application. If the timestamps are not matched then the leader split will ask the follower to wait until it sync the data to that Zone. And then the split will serve the data.

### **Stale/Time bounded read:**

Spanner supports MVCC. So it'll keep the old data for some period of time. If our applications are fine to get the old data (older than X seconds) then we don't need to wait for data sync from the leader split. For example, We have to tell the Split that we are fine with 15sec old data, then it'll check the committed timestamp and that is less than 15 seconds, then the old data will be served to the application.





## Spanner with Multi Region:

All scenarios explained above apply to clusters within a single region — zone level. But Spanner is built for global scale and multi-region deployments. The architecture and write/read operations will have a slight difference in the multi region setup. In the single region concept, we need a minimum of 3 zones to create the cluster. And the zones support both read and write. But in Multi region concept, One Continent will be act as a Leader and the rest of the Continent will be the followers. In Spanner terms, the Continent where we have more region will be the quorum. All the writes will go to any region in this continent. In the quorum continent, 2 regions will be hosting the data nodes, and 1 region will host the witness for failover. Other continents will have read only replica nodes.



## Consistency in Multi-region:

In a multi region cluster, the writes are always performed on the Quorum continent. Let's say, US region is the R/W continent, then if you are sending a write request from the US region, then the Spanner API will send it the nearest region, once the data has been committed then the success response will go to the client. If you are sending a write request from Asia region, then the Asia region's API servers will put the request into Google's internal network and send the request to the US region's API server. Then that

US region API server will commit the data and the success response will be send it to Asia region client.

For Reads, the process is same as single region concept, if the TrueTime matches, then the data will be served from the local region, else it'll wait until the data sync to the local region and then served to the clients.

## Conclusion:

We covered most of the internal concepts of Spanner. But still there are a lot more things to learn in Cloud Spanner. Please refer the Google Cloud Next event videos links below.

1. [Cloud Spanner 101](#)
2. [Cloud Spanner 201](#)
3. [Spanner Internals Part 1: What Makes Spanner Tick?](#)
4. [Spanner Internals Part 2: Global Meta-Data and Scalable Data Backend](#)
5. [Cloud Spanner: How It Works](#)

Gcp   Cloud Spanner   Distributed Systems

[About](#) [Help](#) [Legal](#)

Get the Medium app

