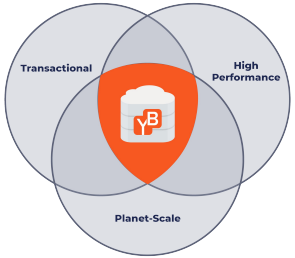


Distributed SQL Databases Deconstructed

Understanding Amazon Aurora, Google Spanner & the Spanner Derivatives

Karthik Ranganathan
Sid Choudhury
April 18, 2019

YugaByte

Introduction

Karthik Ranganathan

Co-Founder & CTO, YugaByte
Nutanix ♦ Facebook ♦ Microsoft
IIT Madras, University of Texas-Austin

@karthikr

Sid Choudhury

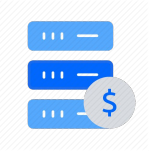
VP Product, YugaByte
AppDynamics ♦ Salesforce ♦ Oracle
IIT Kharagpur, University of Texas-Austin

@SidChoudhury

Types of Data Stores

This Talk's Focus

OLAP

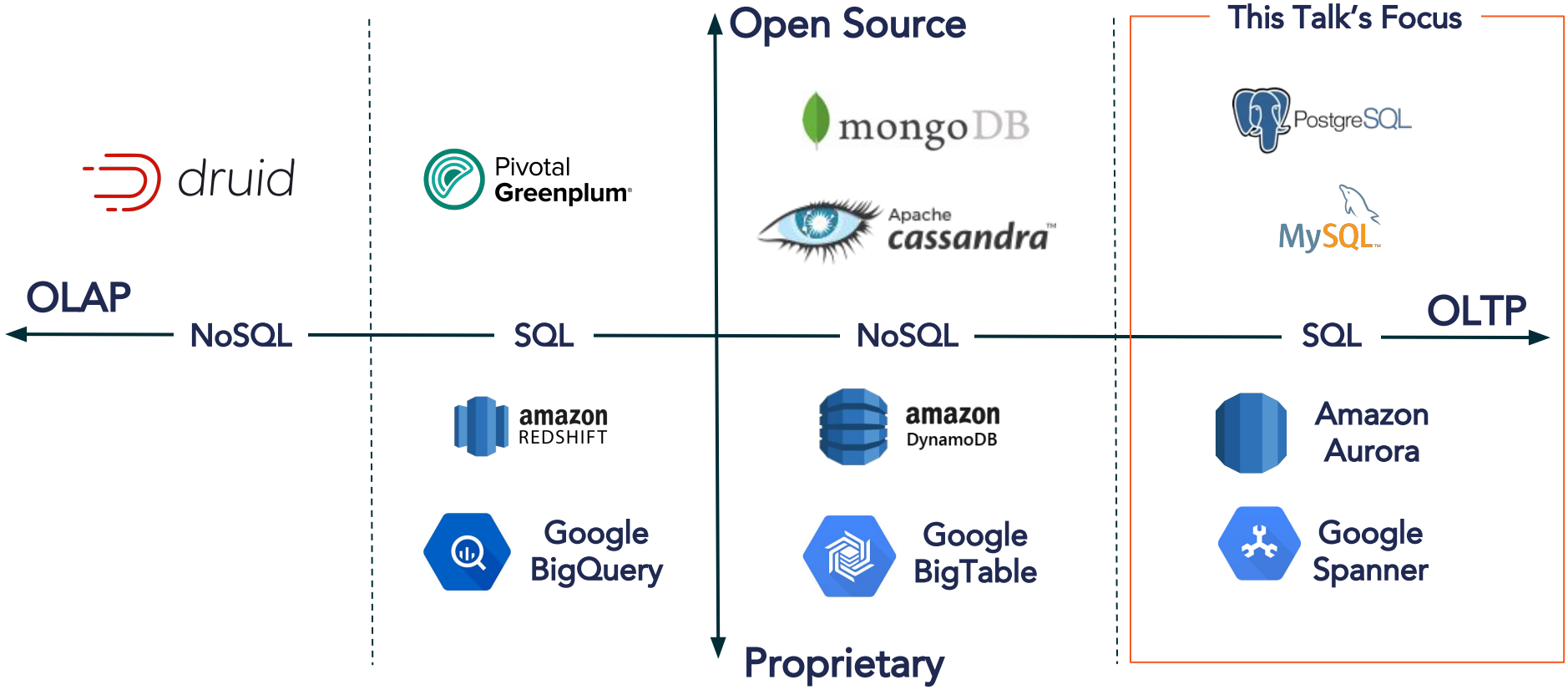


OLTP

Write once, Read many
Few concurrent sessions
Long running, ad-hoc queries
Large table scans
Petabyte-scale data storage

Mixed reads & writes
Many concurrent sessions
Single-digit ms query latency
Point reads & short-range scans
Terabyte-scale data storage

Examples



Devs ❤ SQL

1. Query Flexibility 💪
   - Model data once, change queries as business changes
   - Balance modeling richness with performance needs

2. Rich Ecosystem 🔌
   - Data modeling & query examples
   - Developer IDEs & data visualization tools
   - Easy to reuse & build integrations

3. Universal Standard for Data Access □
   - Learn once, use forever

Devs SQL

1. Large Dataset?
   - No horizontal write scalability
   - Use manually sharded SQL or non-transactional NoSQL

2. Infrastructure Failures?
   - No native failover & repair
   - Use complex replication schemes

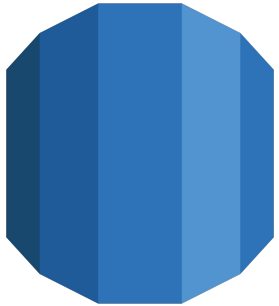
3. Multi-Region/Geo-Distributed App?
   - Multi-master deployment is the only option
   - Data loss w/ Last Writer Wins (LWW) conflict resolution

Distributed SQL = Keep  & Remove 

1. SQL Features
   - ACID, JOINs, foreign keys, serializable isolation
2. Horizontal Write Scalability
   - Scale write throughput by adding/removing nodes
3. Fault Tolerance With High Availability
   - Native failover & repair
4. Globally Consistent Writes
   - Lower end user latency and tolerate region failures
5. Low Read Latency
   - Strongly consistent (aka correct) reads

Distributed SQL Architectures - Aurora vs Spanner

Shared Storage



Amazon Aurora

"A highly available MySQL and PostgreSQL-compatible relational database service"

Available on AWS since 2015

Shared Nothing

Google Spanner

"The first horizontally scalable, strongly consistent, relational database service"

Available on Google Cloud since 2017

#1 SQL Features

Depth of SQL Support

Amazon Aurora

✓ MySQL and PostgreSQL-compatible

MySQL and PostgreSQL Compatible

The Amazon Aurora database engine is fully compatible with existing MySQL and PostgreSQL open source databases, and

Google Spanner

⚠ Subset of MySQL/PostgreSQL features

Foreign keys and referential integrity

Cloud Spanner doesn't have foreign key constraints or triggers. If you rely on these features, you must move this functionality to your application.

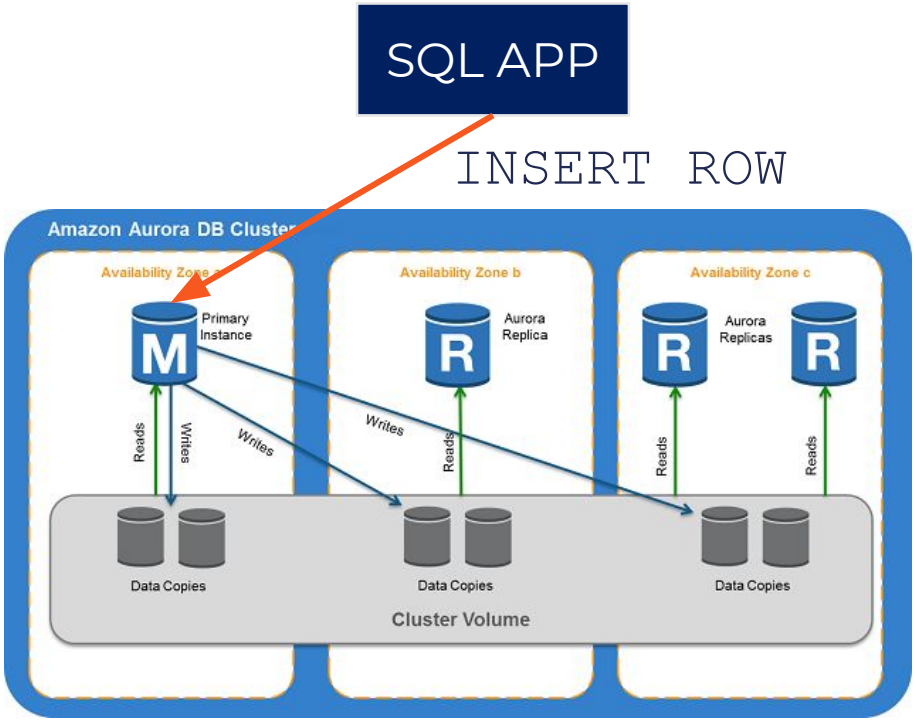
Aurora vs Spanner

Feature	Amazon Aurora	Google Spanner
SQL Features	✓	
Horizontal Write Scalability		
Fault Tolerance with HA		
Globally Consistent Writes		
Low Read Latency		

#2 Horizontal Write Scalability

Amazon Aurora

Single Node SQL on Multi-Zone Distributed Storage

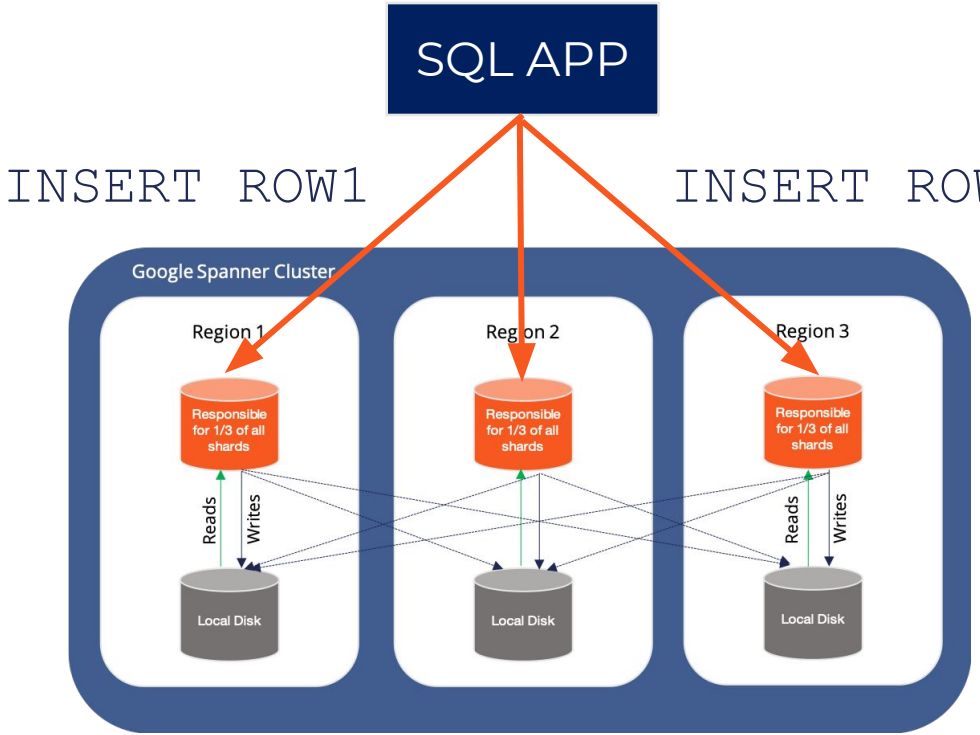


× Add Primary Instances for Write Scaling

✓ Add Read Replicas for Read Scaling

Google Spanner

Multi-Node SQL on Multi-Region Distributed Storage



- ✓ Add Primary Instances for Write Scaling
- ✓ Add Read Replicas for Read Scaling

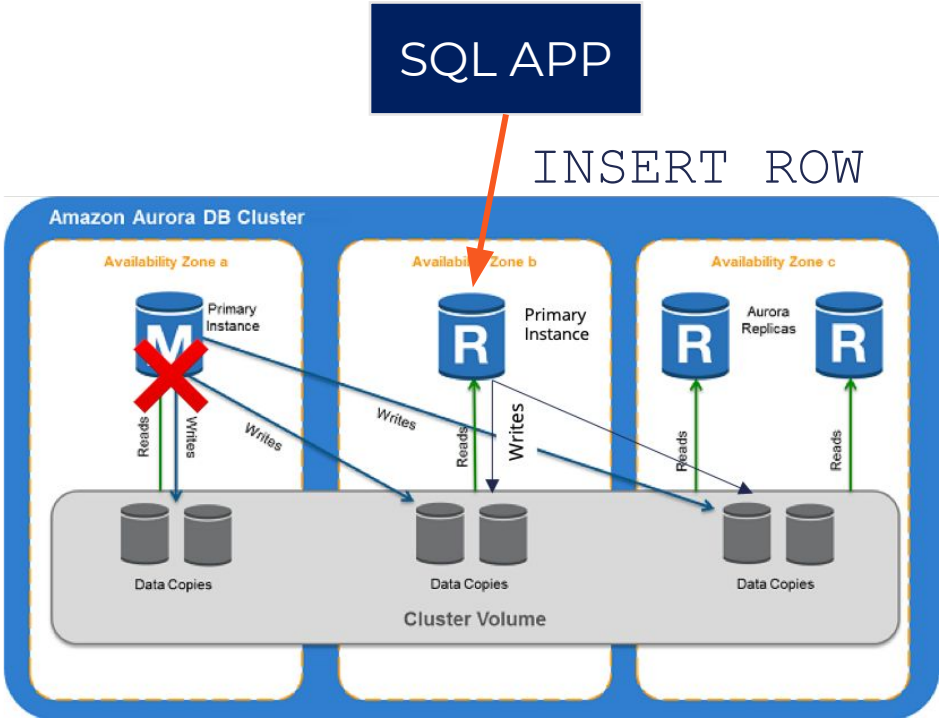
Aurora vs Spanner

Feature	Amazon Aurora	Google Spanner
SQL Features	✓	
Horizontal Write Scalability	✗	✓
Fault Tolerance with HA		
Globally Consistent Writes		
Low Read Latency		

#3 Fault Tolerance with HA

Amazon Aurora

Native Failover & Repair Through Primary Auto Election

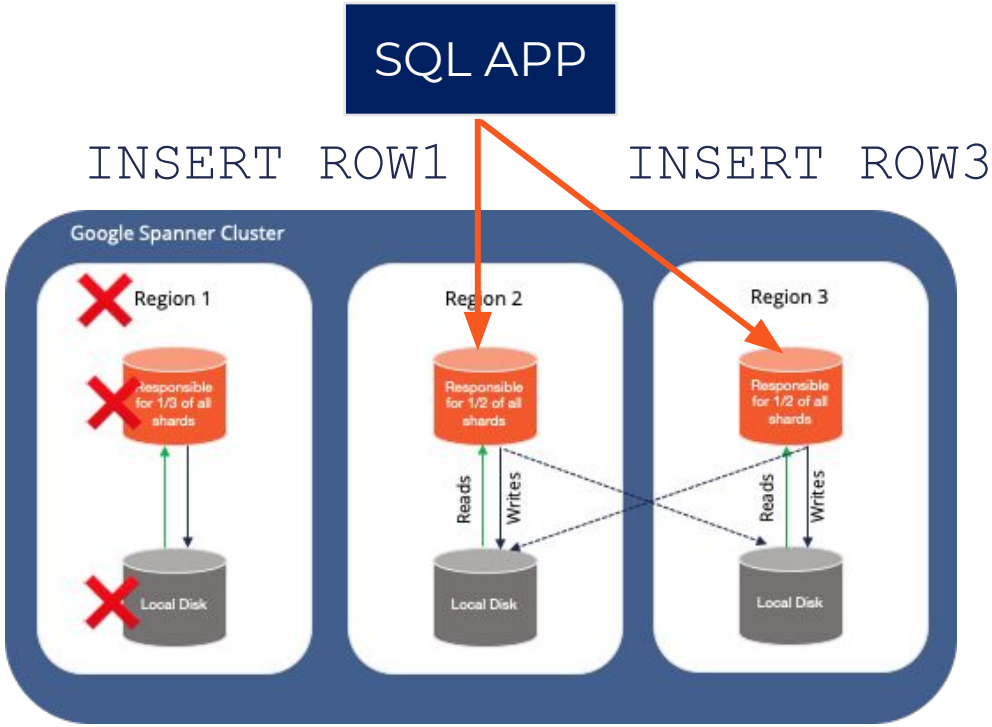


✓ HA When Primary Instance Fails

✓ HA When Read Replica Fails

Google Spanner

Native Failover & Repair Through Shard Leader Auto Election



✓ HA When Any Primary Node Fails

✓ HA When Read Replica Fails

Aurora vs Spanner

Feature	Amazon Aurora	Google Spanner
SQL Features	✓	
Horizontal Write Scalability		✓
Fault Tolerance with HA	✓	✓
Globally Consistent Writes		
Low Read Latency		

#4 Globally Consistent Writes

Amazon Aurora

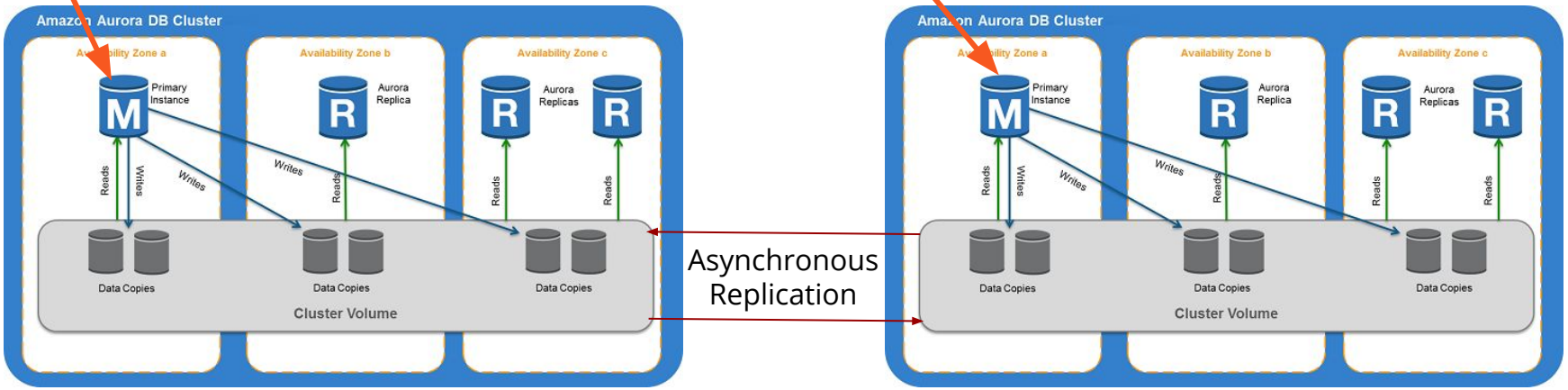
Multi-Master Last Writer Wins Conflict Resolution Leads to Inconsistencies

SQL APP

SQL APP

```
SET BALANCE = BALANCE - 10
```

```
SET BALANCE = BALANCE - 100
```

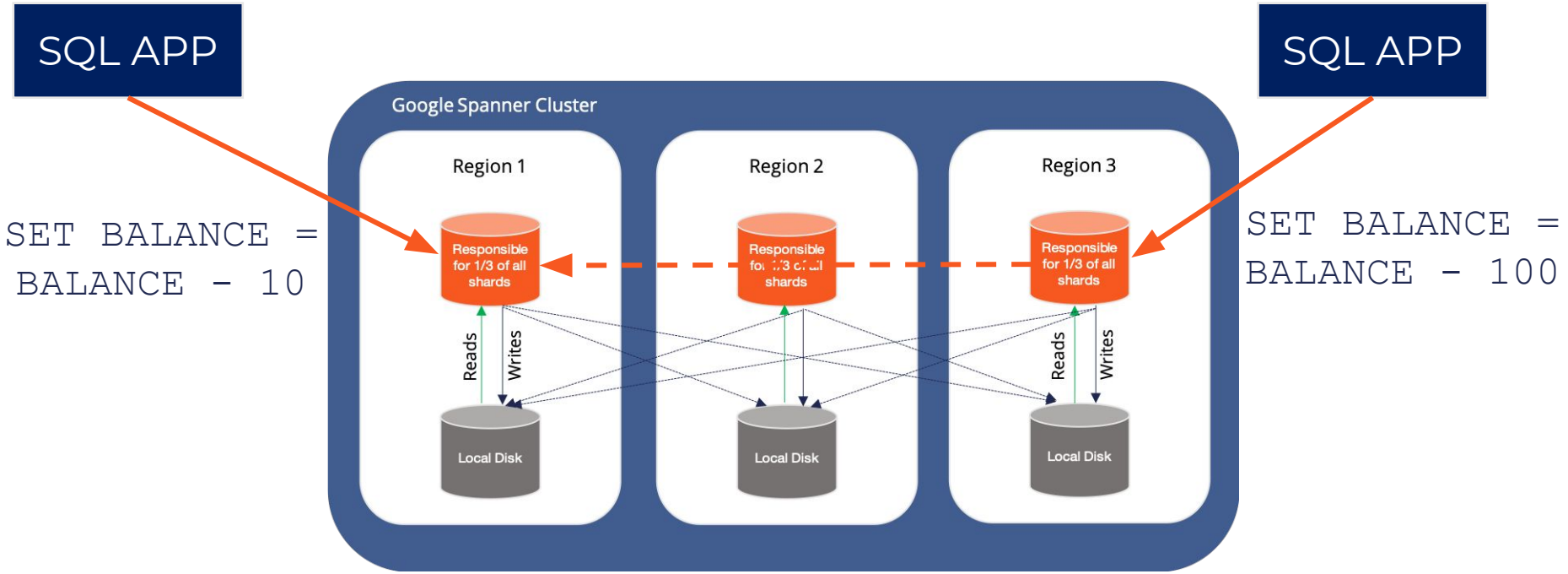


Region 1

Region 2

Google Spanner

Purpose-Built for Globally Consistent Writes



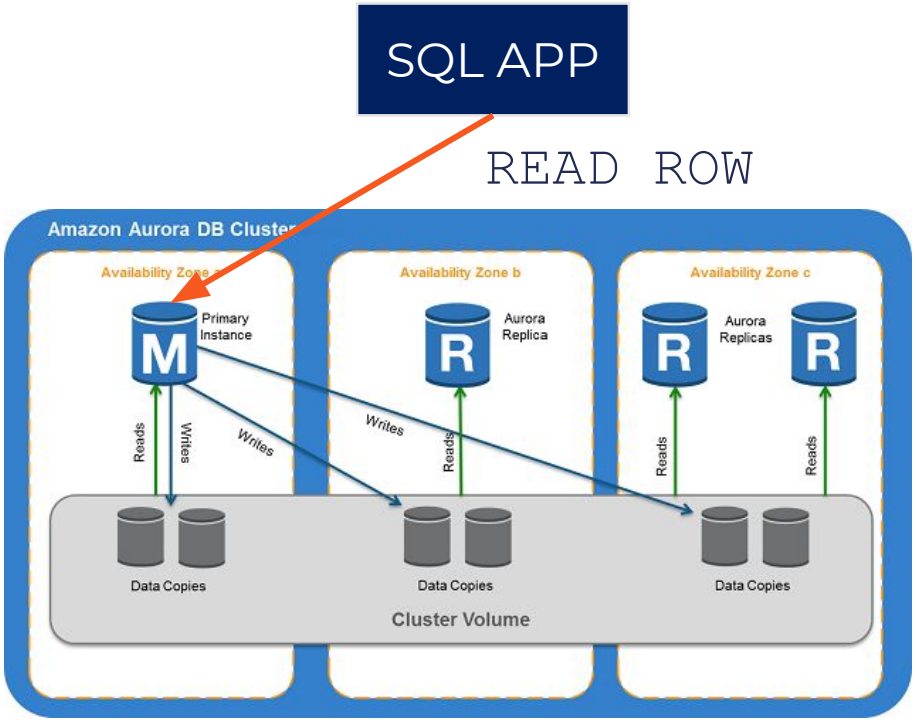
Aurora vs Spanner

Feature	Amazon Aurora	Google Spanner
SQL Features	✓	
Horizontal Write Scalability	✗	✓
Fault Tolerance with HA	✓	✓
Globally Consistent Writes	✗	✓
Low Read Latency		

#5 Low Read Latency

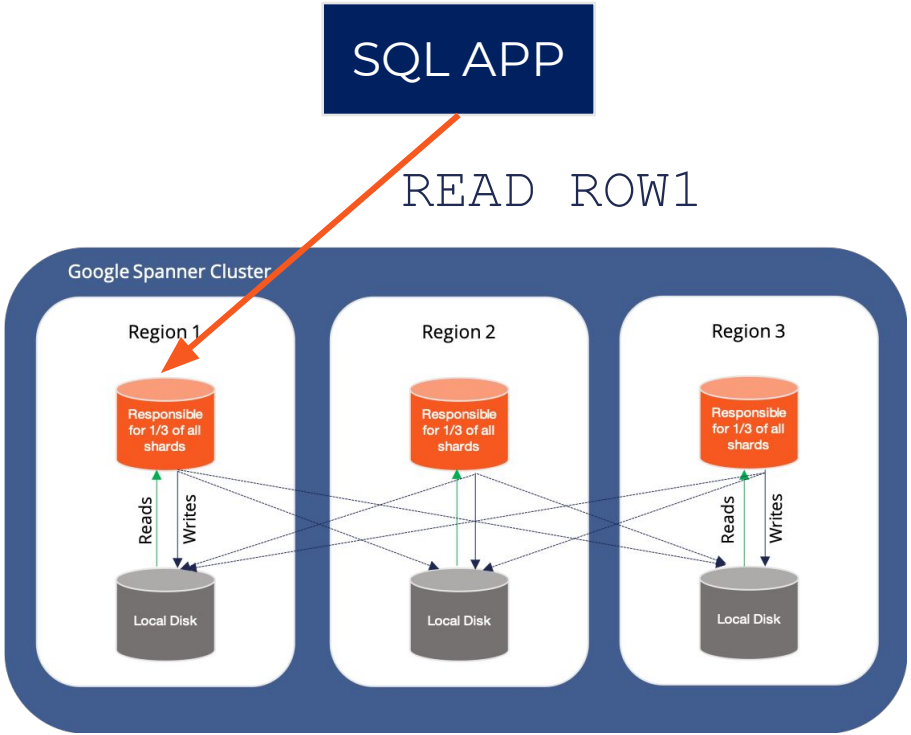
Amazon Aurora

Strongly Consistent Reads Served By Primary Instance



Google Spanner

Strongly Consistent Reads Served By Shard Leaders w/o Read Quorum



Aurora vs Spanner

Feature	Amazon Aurora	Google Spanner
SQL Features	✓	
Horizontal Write Scalability	✗	✓
Fault Tolerance with HA	✓	✓
Globally Consistent Writes	✗	✓
Low Read Latency	✓	✓

Battle of Architectures - Spanner Beats Aurora

No Performance & Availability Bottlenecks

Scale to Large Clusters while Remaining Highly Available

Built for Geo-Distributed Apps

Future Proofs Data Tier at Global Businesses

Complex to Engineer

Needs Clock Skew Tracking Across Instances

Analyzing Open Source Spanner Derivatives

Spanner Brought to Life in Open Source

YugaByte DB

Cockroach DB

TiDB

Design Principles

- **CP in CAP Theorem**
  - Consistent
  - Partition Tolerant
  - HA on failures
    (new leader elected in seconds)
- **ACID Transactions**
  - Single-row linearizability
  - Multi-row ACID
    - Serializable
    - Snapshot
- **High Performance**
  - All layers in C++ to ensure high perf
  - Run on large memory machines
  - Optimized for SSDs
- **Run Anywhere**
  - No external dependencies
  - No atomic clocks
  - Bare metal, VM and Kubernetes

Functional Architecture

YSQL

PostgreSQL-Compatible Distributed SQL API

DOCDB

Spanner-Inspired Distributed Document Store

CLOUD NEUTRAL

No Specialized Hardware Needed

Distributed SQL = Keep  & Remove 

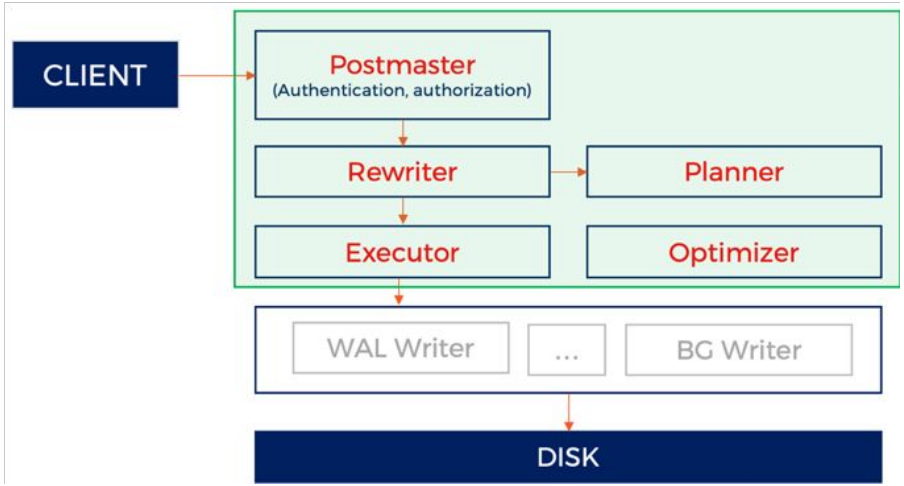
1. SQL Features
2. Replication Protocol
3. Clock Skew Tracking
4. Transactions Manager

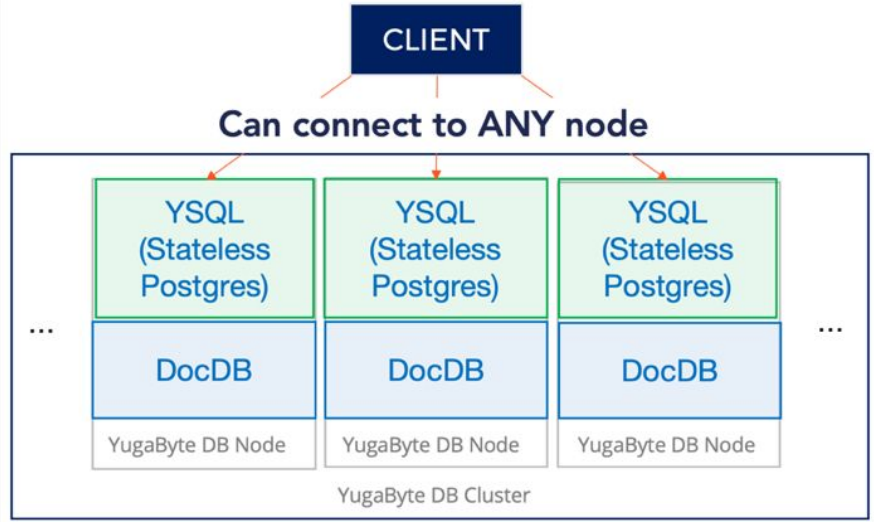
Spanner vs. its Open Source Derivatives

Feature	Google Spanner	YugaByte DB	CockroachDB	TiDB
Cost	Expensive	Free	Free	Free
SQL API Compatibility				
Replication Protocol				
Clock Skew Tracking				
Transaction Manager				
Tunable Read Latency				
Official Jepsen Tests				

SQL API Compatibility

PostgreSQL Transformed into Distributed SQL





Depth of SQL Support

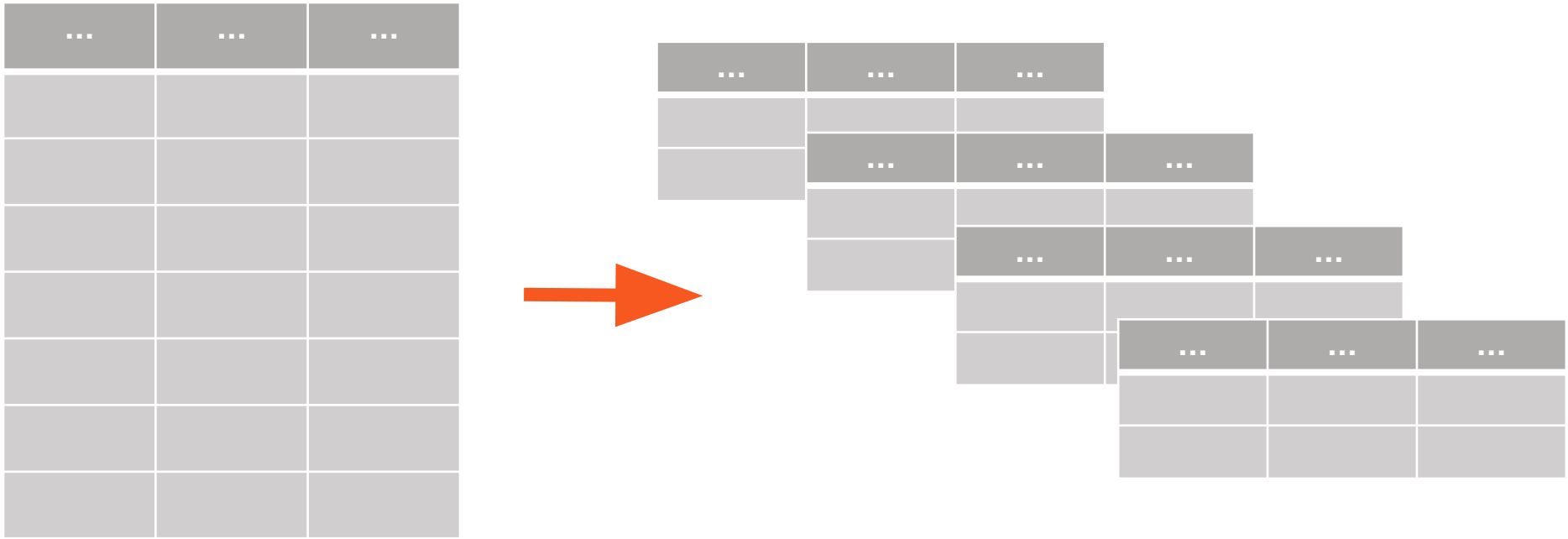
- **Current**
  - Data Types
  - Built-in Functions
  - Expressions
  - JSON Column Type
  - Secondary Indexes
  - JOINs
  - Transactions
  - Views
- **Future**
  - Relational Integrity (Foreign Keys)
  - Stored Procedures
  - Triggers
  - Foreign Data Wrappers
  - And more …

Spanner vs. its Open Source Derivatives

Feature	Google Spanner	YugaByte DB	CockroachDB	TiDB
Cost	Expensive	Free	Free	Free
SQL API Compatibility	Proprietary	PostgreSQL	PostgreSQL	MySQL
Replication Protocol				
Clock Skew Tracking				
Transaction Manager				
Tunable Read Latency				
Official Jepsen Tests				

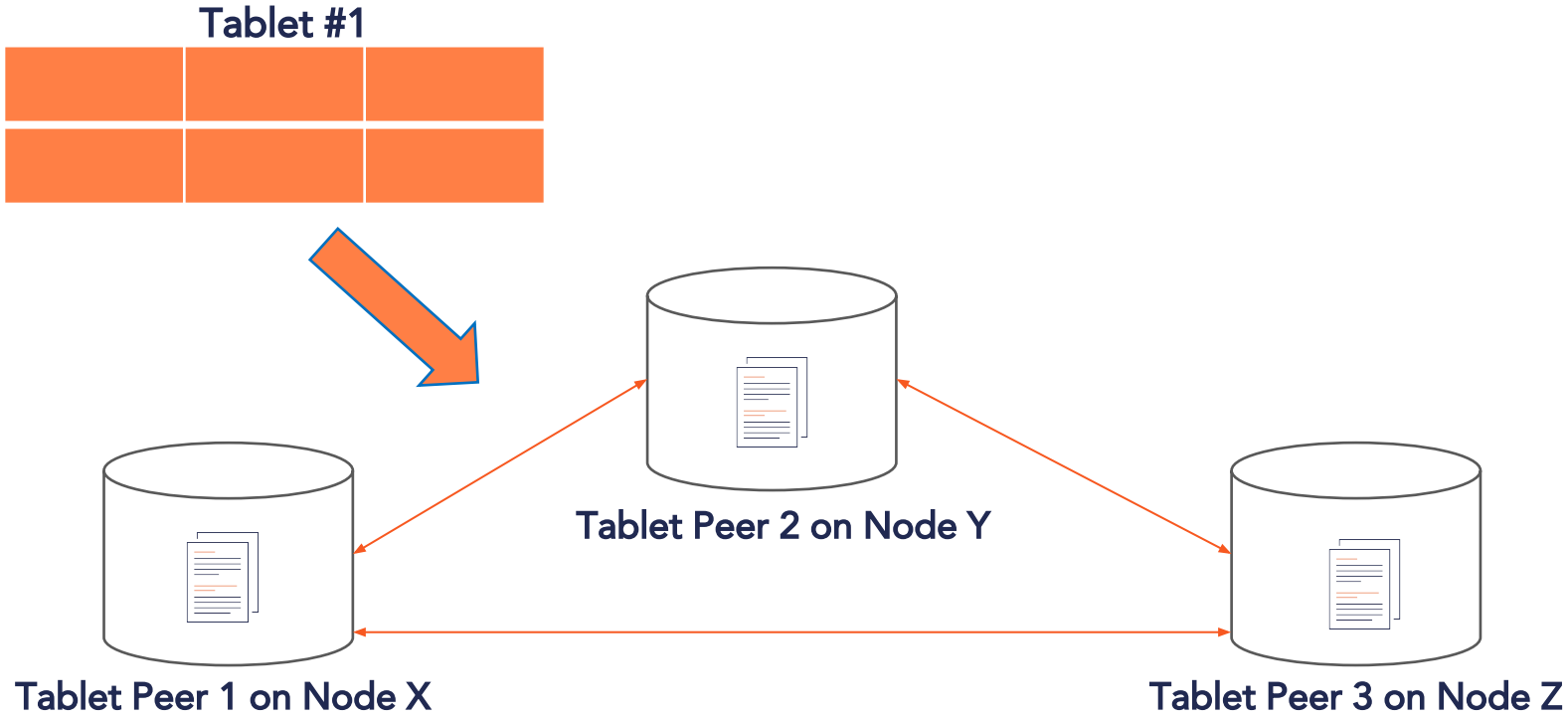
Replication Protocol

Every Table is Automatically Sharded

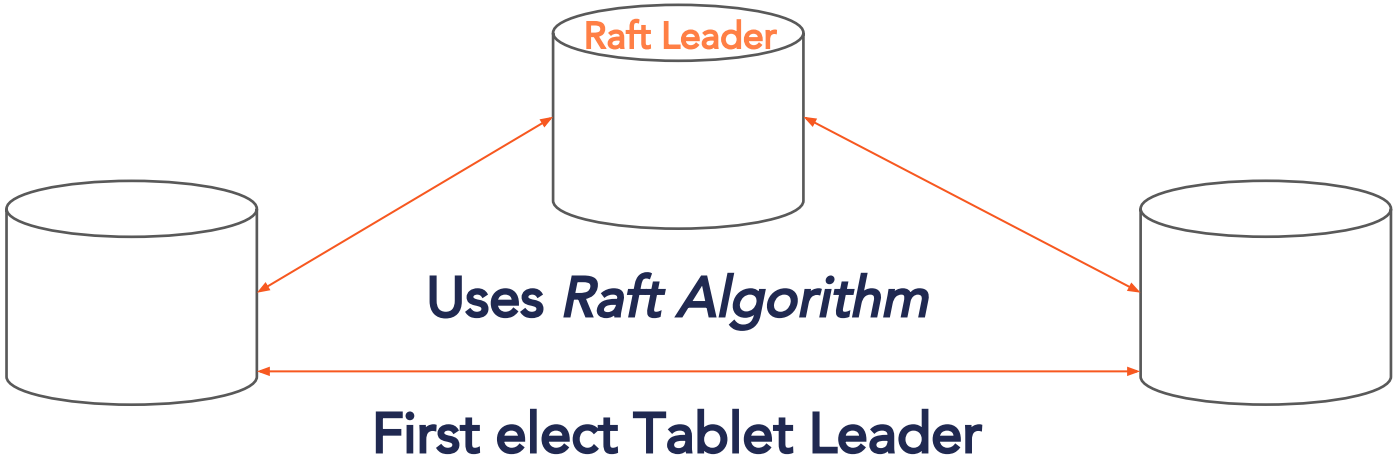


SHARDING = **AUTOMATIC PARTITIONING OF TABLES**

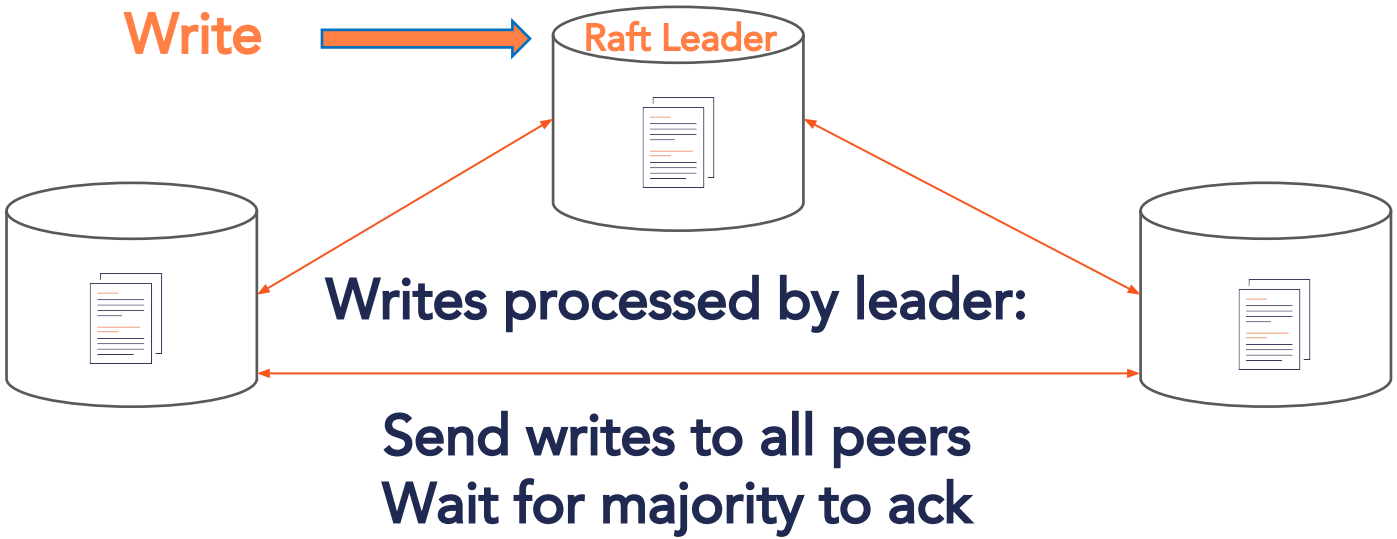
Replication Done at Shard Level



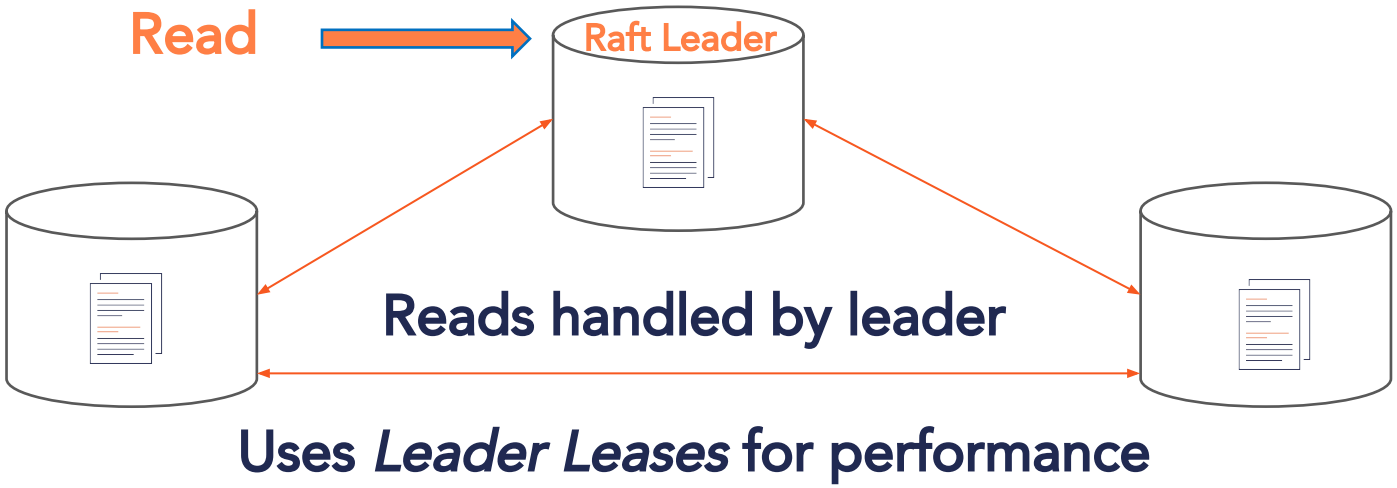
Replication uses a Consensus algorithm



Writes in Raft Consensus



Reads in Raft Consensus



Uses *Leader Leases* for performance

Spanner vs. its Open Source Derivatives

Feature	Google Spanner	YugaByte DB	CockroachDB	TiDB
Cost	Expensive	Free	Free	Free
SQL API Compatibility	Proprietary	PostgreSQL	PostgreSQL	MySQL
Replication Protocol	Paxos	Raft	Raft	Raft
Clock Skew Tracking				
Transaction Manager				
Tunable Read Latency				
Official Jepsen Tests				

Transactions and Clock Skew Tracking

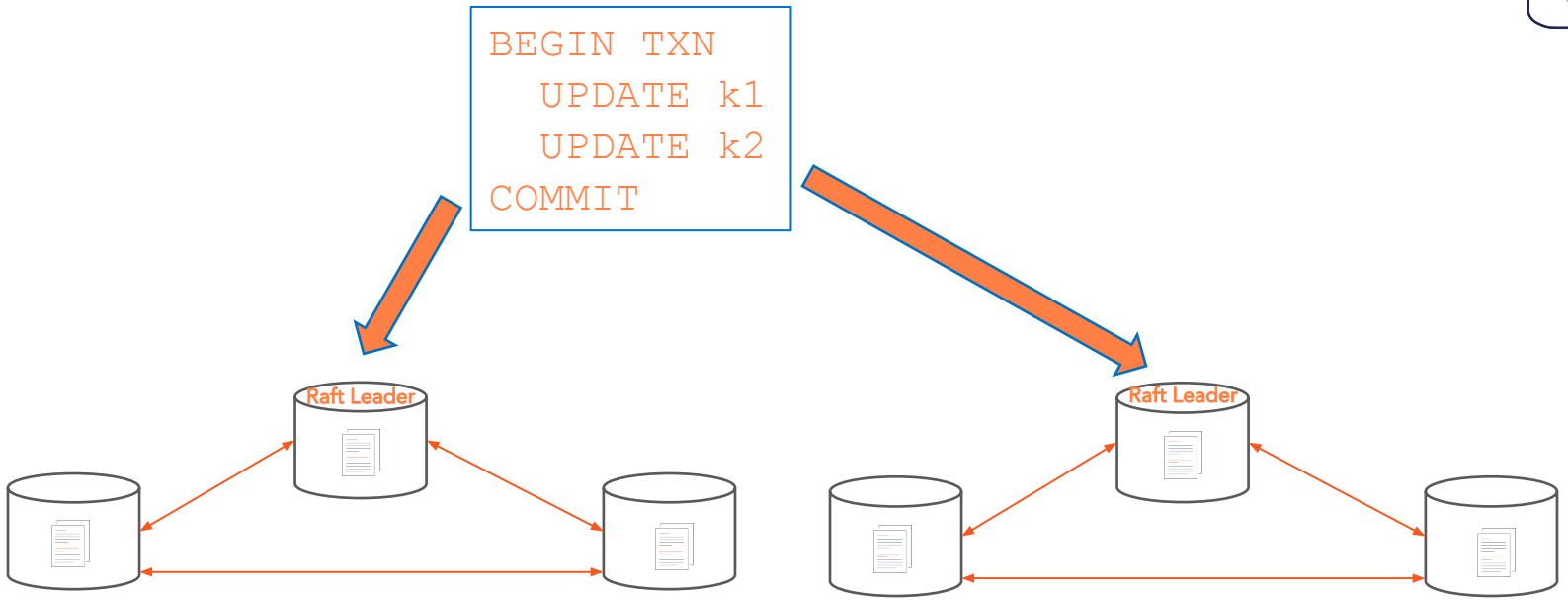
Multi-Shard Transactions

```
BEGIN TXN
  UPDATE k1
  UPDATE k2
COMMIT
```

k1 and k2 may belong to **different shards**

Belong to **different Raft groups** on completely **different nodes**

What do Distributed Transactions need?



Updates should get written at the same physical time

But how will nodes agree on time?

Use a Physical Clock

You would need an Atomic Clock or two lying around

Atomic Clocks are highly available,
globally synchronized clocks with tight error bounds

Jeez! I'm fresh out of those.

Most of my physical clocks are never synchronized

Hybrid Logical Clock or HLC

Combine coarsely-synchronized physical clocks with Lamport Clocks to track causal relationships

(physical component, logical component)

synchronized using NTP

a monotonic counter

Nodes update HLC on each Raft exchange for things like heartbeats, leader election and data replication

Spanner vs. its Open Source Derivatives

Feature	Google Spanner	YugaByte DB	CockroachDB	TiDB
Cost	Expensive	Free	Free	Free
SQL API Compatibility	Proprietary	PostgreSQL	PostgreSQL	MySQL
Replication Protocol	Paxos	Raft	Raft	Raft
Clock Skew Tracking	TrueTime Atomic Clock	Hybrid Logical Clock + Max Clock Skew	Hybrid Logical Clock + Max Clock Skew	Single Timestamp Gen ⇒ No Tracking Needed
Transaction Manager	At Every Node	At Every Node	At Every Node	Special Node for Timestamp Generation
Tunable Read Latency				
Official Jepsen Tests				

Miscellaneous

Spanner vs. its Open Source Derivatives

Feature	Google Spanner	YugaByte DB	CockroachDB	TiDB
Cost	Expensive	Free	Free	Free
SQL API Compatibility	Proprietary	PostgreSQL	PostgreSQL	MySQL
Replication Protocol	Paxos	Raft	Raft	Raft
Clock Skew Tracking	TrueTime Atomic Clock	Hybrid Logical Clock + Max Clock Skew	Hybrid Logical Clock + Max Clock Skew	Single Timestamp Gen ⇒ No Tracking Needed
Transaction Manager	At Every Node	At Every Node	At Every Node	Special Node for Timestamp Generation
Tunable Read Latency	✓	✓	✗	✗
Official Jepsen Tests	Unknown	✓	✓	✗

Read more at blog.yugabyte.com

Storage Layer

blog.yugabyte.com/distributed-postgresql-on-a-google-spanner-architecture-storage-layer

Query Layer

blog.yugabyte.com/distributed-postgresql-on-a-google-spanner-architecture-query-layer

Questions?

Try it at docs.yugabyte.com/quick-start

Check us out on GitHub
https://github.com/YugaByte/yugabyte-db