

# Introducing Yugabyte



# About Us

---



Bill Cook  
CEO



Kannan Muthukkaruppan  
President/Co-founder



Karthik Ranganathan  
CTO/Co-founder



Mikhail Bautin  
Software Architect/Co-founder

Creators/Committers of:

**Cassandra, HBase, YugabyteDB**

Ran Facebook's public-cloud scale DBaaS:

**+1 Trillion ops/day, +100 Petabytes**

Scaled data platforms at Facebook:

**30 Million to 1.4 Billion users**

Founded Feb 2016

Funded by leading cloud-infra VCs



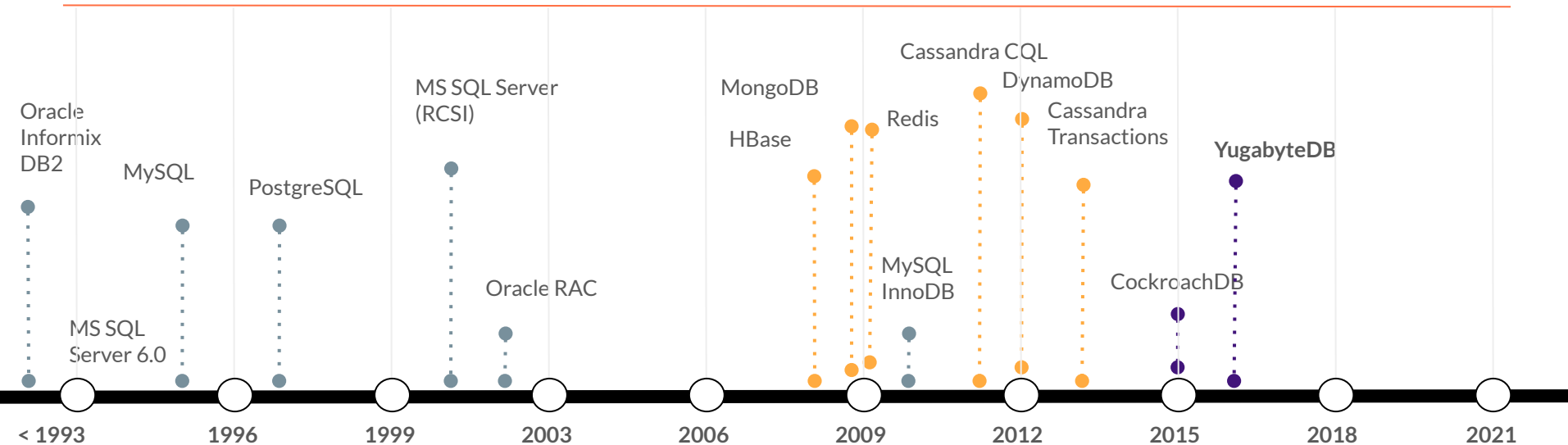







**Transactional, distributed SQL database designed for resilience and scale**






*100% open source, PostgreSQL compatible, enterprise-grade RDBMS  
.....built to run across all your cloud environments*



# Database Evolution



-  No architectural resilience
-  No horizontal scaling
-  No geo-distribution
-  ACID transactions
-  SQL + RDBMS features

-  Architectural resilience
-  Horizontal scaling
-  Geo-distribution
-  No ACID transactions
-  No SQL + RDBMS features

-  Architectural resilience
-  Horizontal scaling
-  Geo-distribution
-  ACID transactions
-  SQL + RDBMS features

# How Do Cloud Native Workloads Evolve?

## Phase 1

Start with an app framework + DB that I know well and has a strong feature set

SQL



App getting built, gains traction

## Phase 2

Need scale, along with transactions and relational integrity

NoSQL + SQL



App moderately scalable but complex

## Phase 3

Need app to be fast for my global users

NoSQL + SQL +  
Geo-Replication + Cache



App now very complex, slowing down development significantly

# How Do Cloud Native Workloads Evolve?

## Phase 1

Start with an app framework + DB that I know well and has a strong feature set

SQL



App getting built, gains traction

## Phase 2

Need scale, along with transactions and relational integrity

NoSQL + SQL



App moderately scalable but complex

## Phase 3

Need app to be fast for my global users

NoSQL + SQL +  
Geo-Replication + Cache



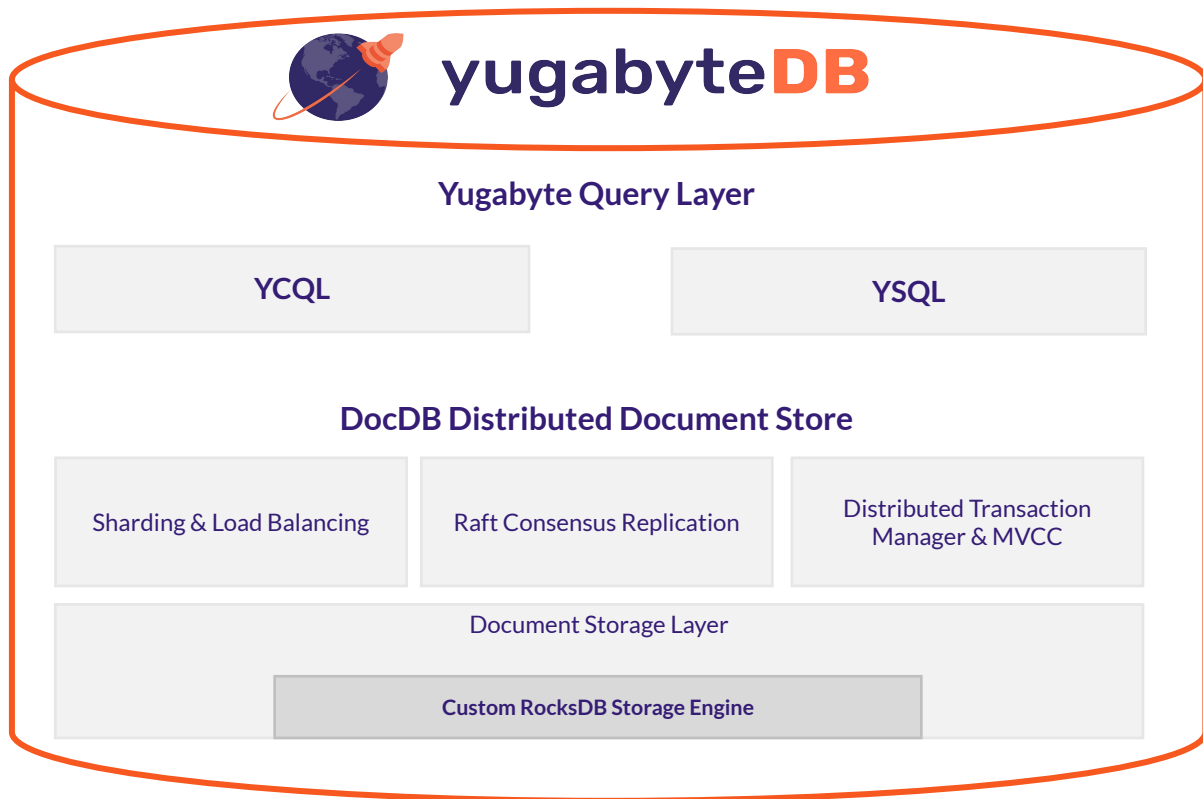
App now very complex, slowing down development significantly

### Solution

Simplify cloud-native apps with distributed SQL  
YugabyteDB is PostgreSQL-compatible, scalable and global

# Achieving Scalability without Compromise

# YugabyteDB Architecture Overview





# Goal:

## Support RDBMS functionality

# PostgreSQL feature set is huge!

- **Basics**

- Data types, queries, built-in functions
- Various types of joins, recursive queries

- **Advanced features**

- Partial indexes, table functions, CTE, etc
- Triggers, stored procedures, etc
- User-defined types
- Security features (row + column level security, etc)

- **Extensibility features**

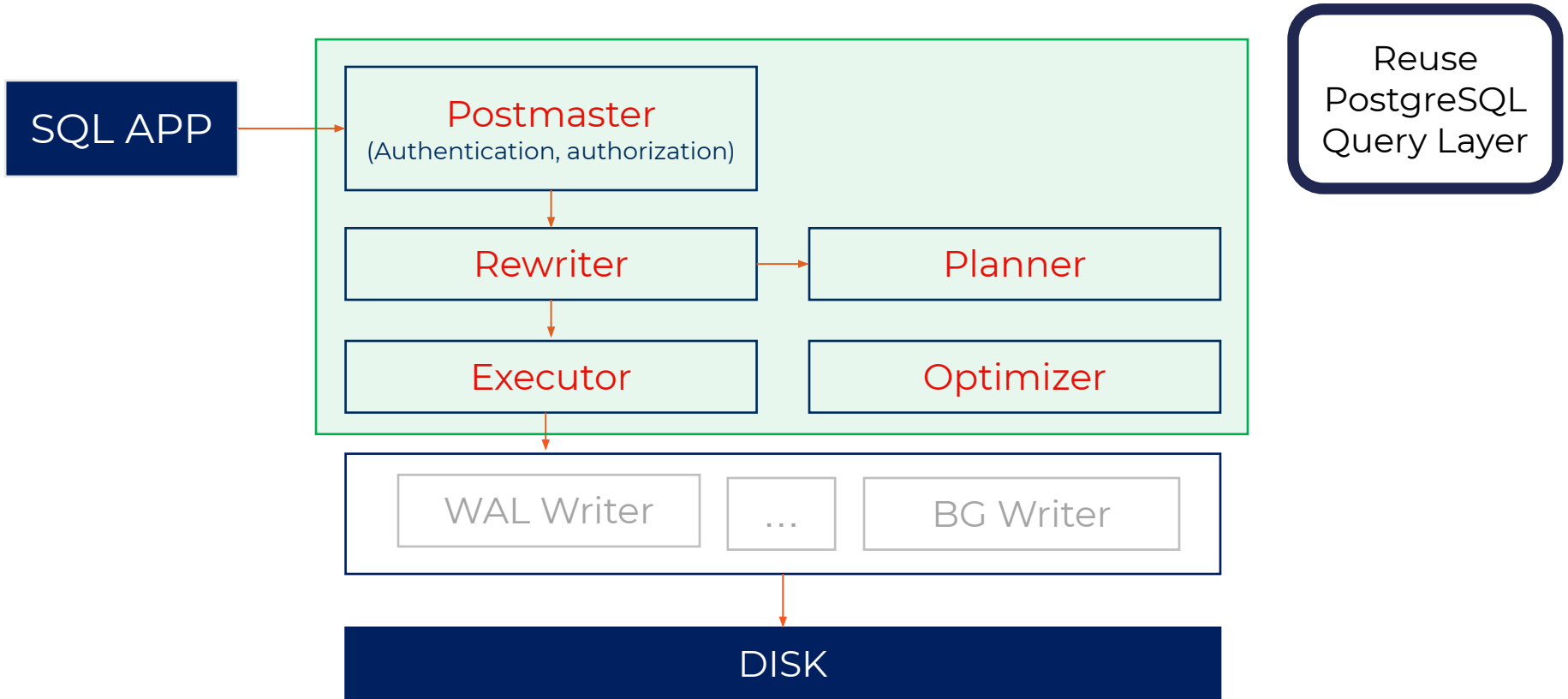
- Extensions support
- Foreign data wrappers

PostgreSQL Feature
Most operators, expressions, and built-in functions
Expression-based indexes
Partial indexes
Table functions
Stored procedures (SQL, pl-pgsql)
Triggers
User-defined types
Temporary tables
Row level security
Column level privileges
PostgreSQL extensions
Foreign data wrappers

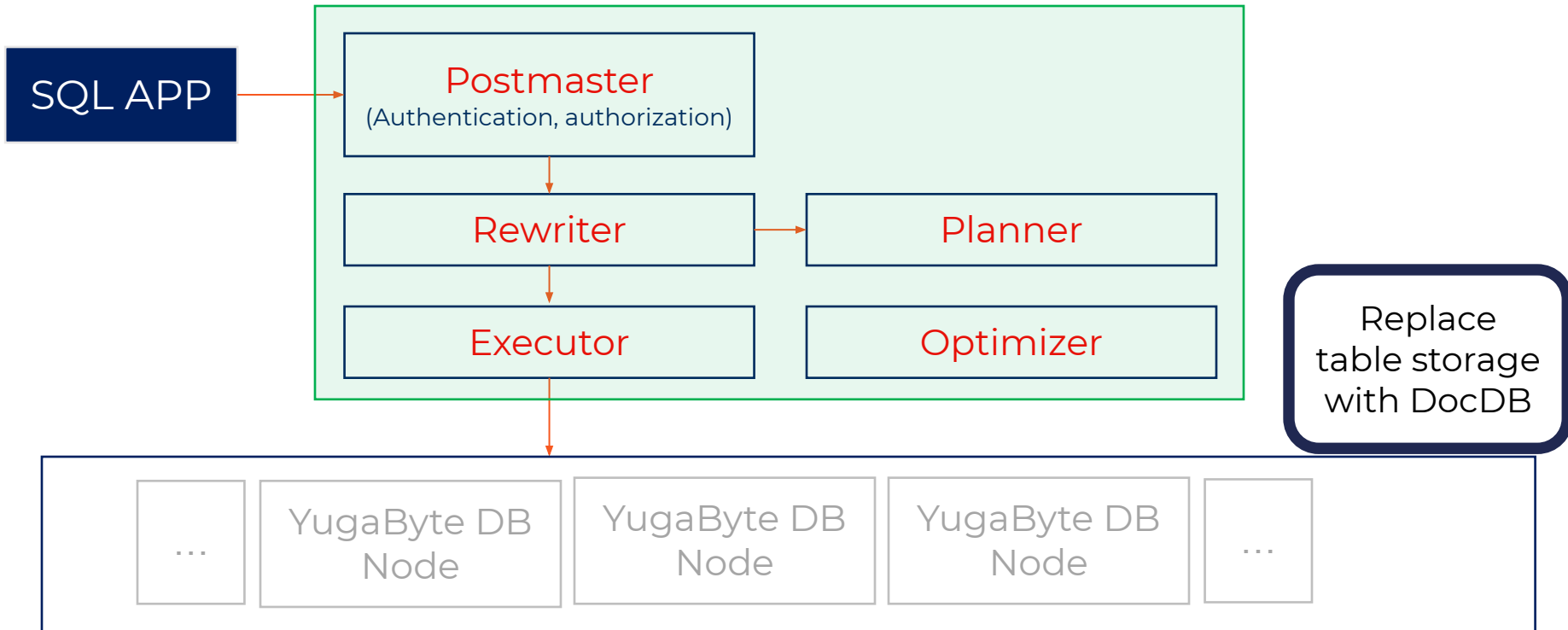
# Conclusion: impossible without reusing PostgreSQL code!

Other distributed SQL databases do not support most of these features.  
Building these features from ground-up is **hard** and **takes a lot of time**.

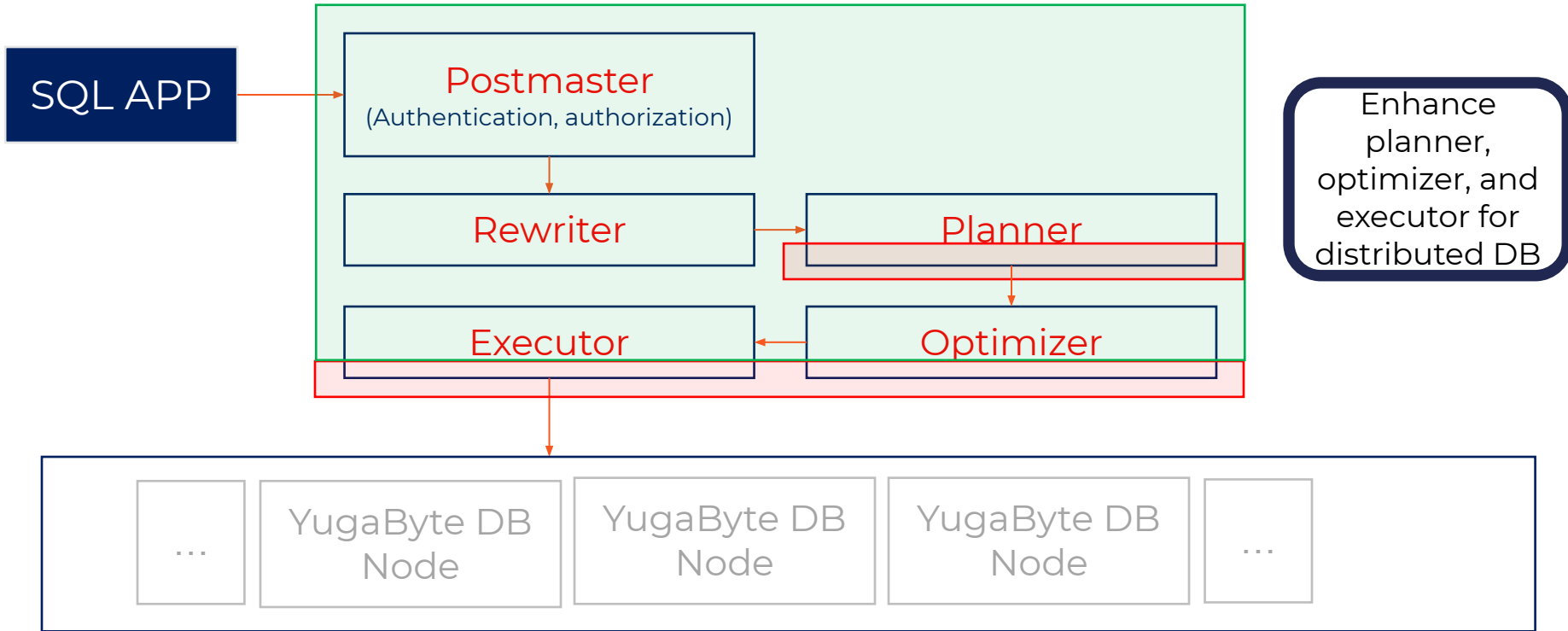
# Existing PostgreSQL Architecture



# DocDB as Storage Engine



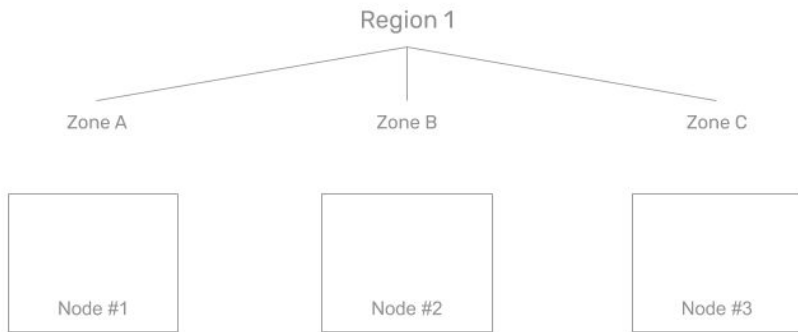
# Make PostgreSQL Run on Distributed Store



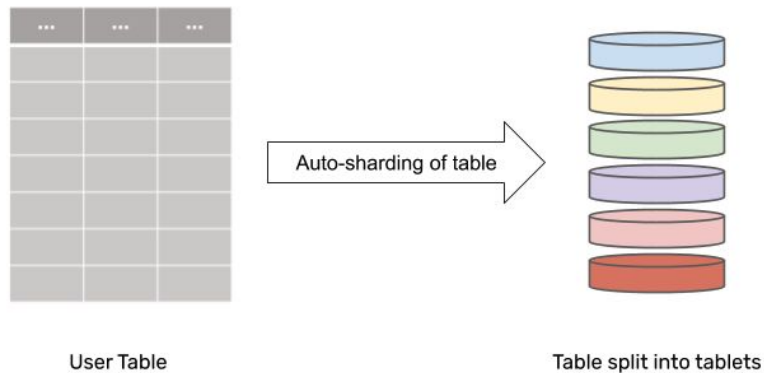
# Goal:

## Achieve horizontal scalability

# Distributing Data For Horizontal Scalability



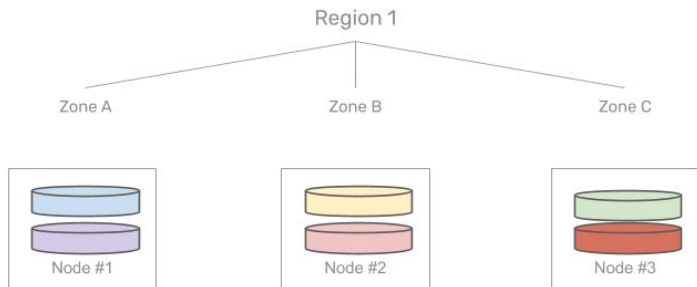
- Assume 3-nodes across zones
- How to distribute data across nodes?



- User tables sharded into tablets
- Tablet = group of rows
- Sharding is transparent to user

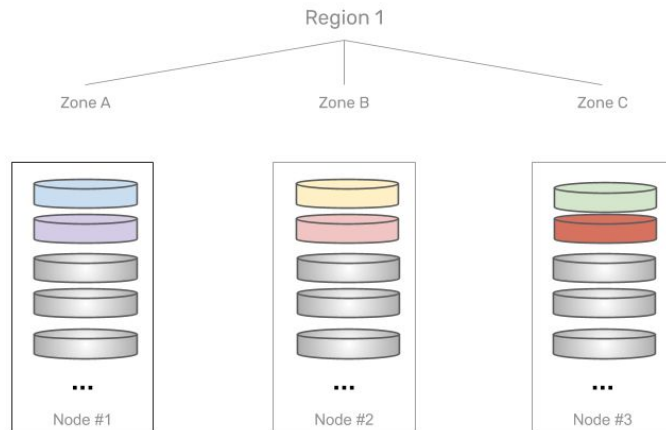


# Distributing Data Across Nodes, Zones, Regions

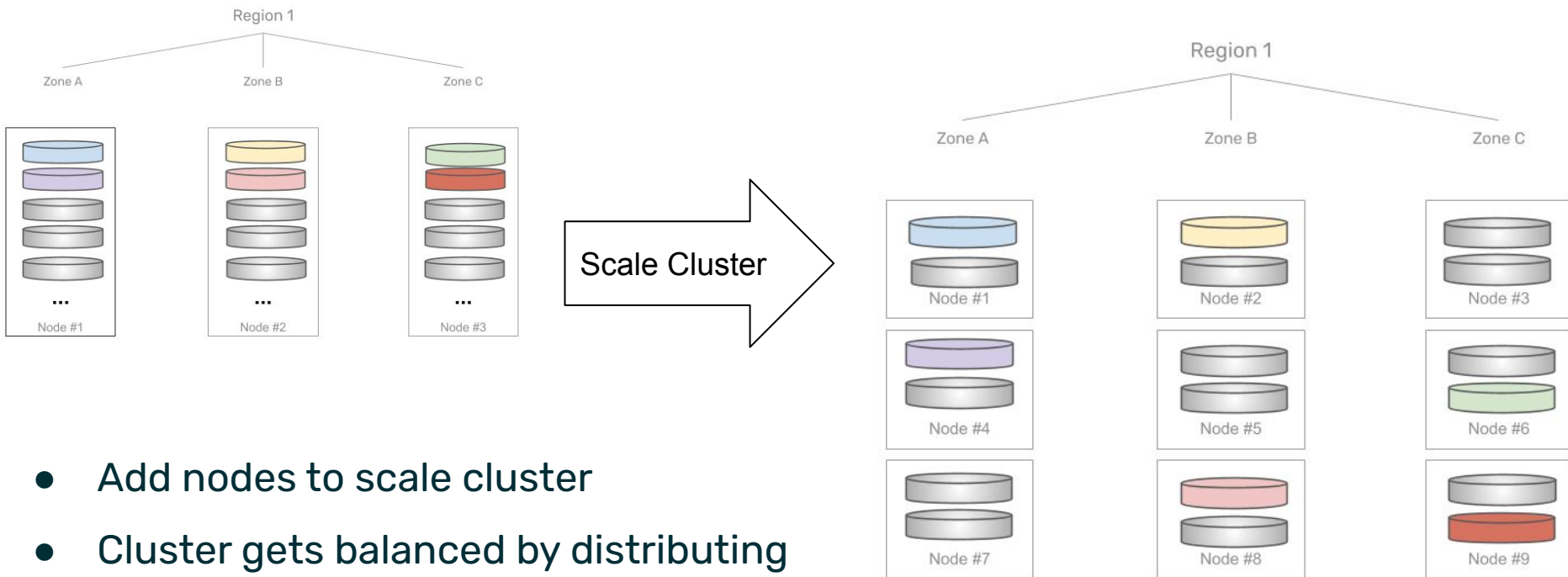


Tablets (per-table, across tables)  
evenly distributed across nodes

In real deployments,  
many tablets per node



# How Horizontal Scalability Works



- Add nodes to scale cluster
- Cluster gets balanced by distributing existing tablets to new nodes

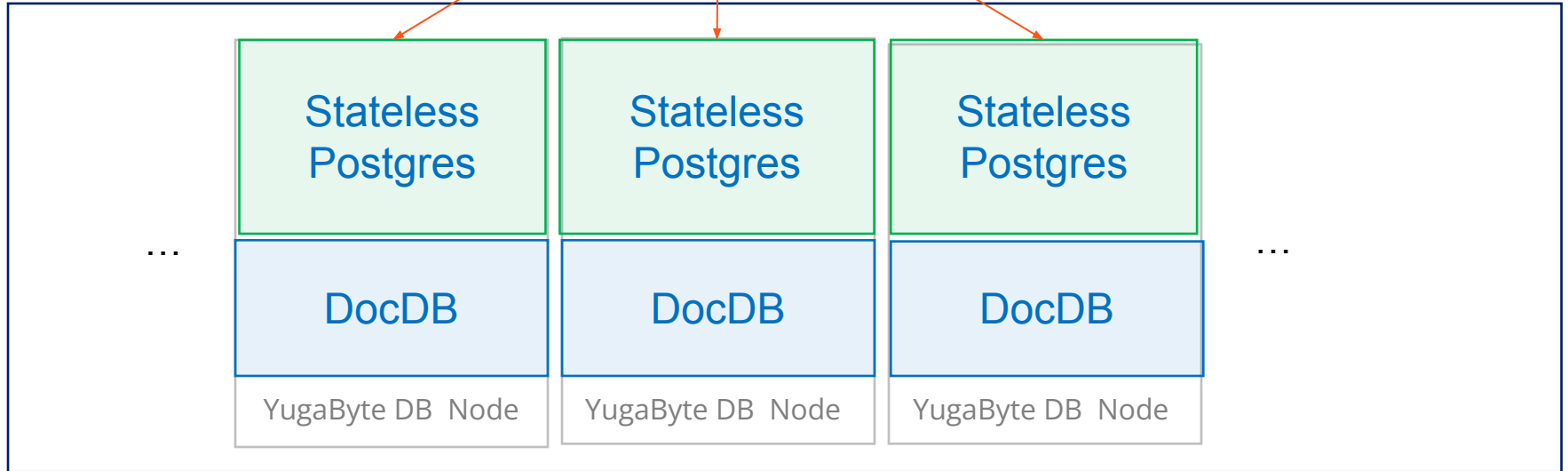
# Goal:

Build a highly-available, fault-tolerant database

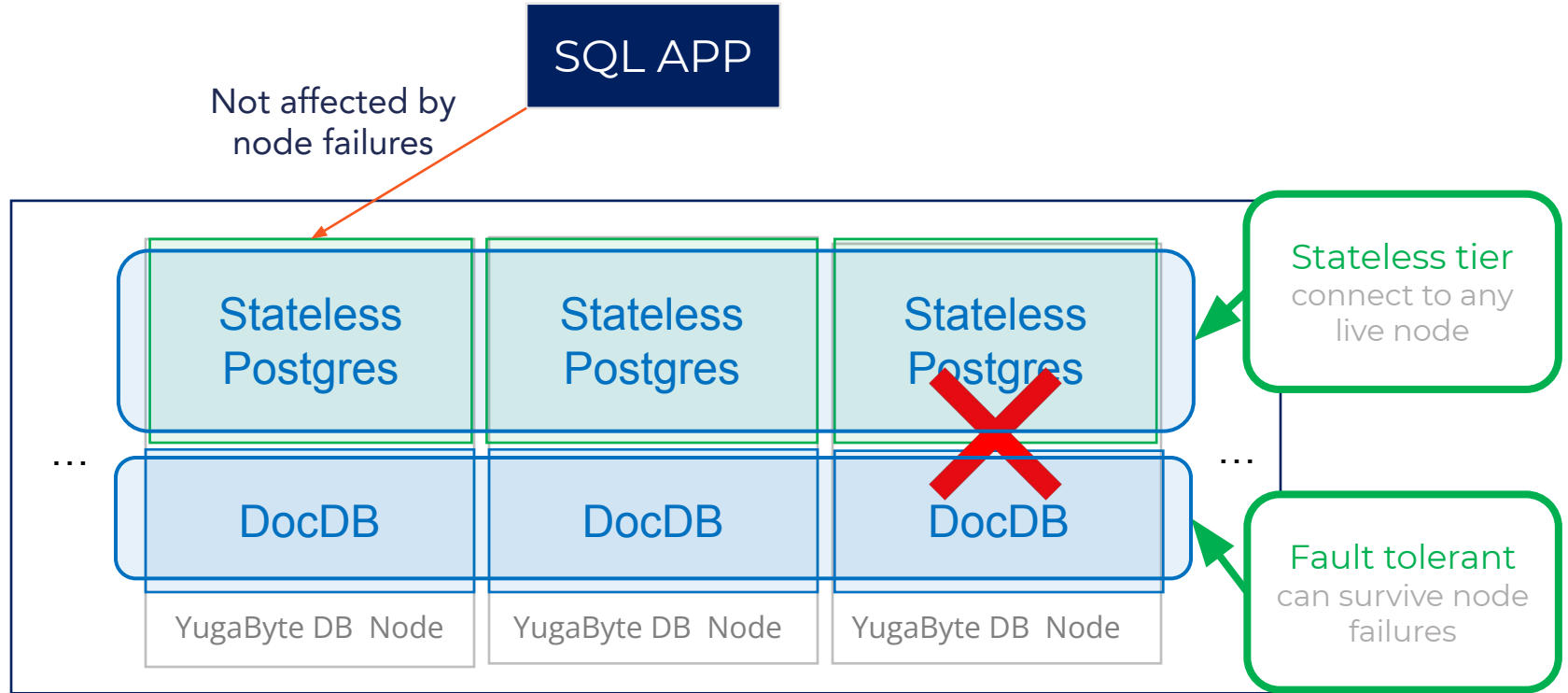
# All Nodes are Identical

SQL APP

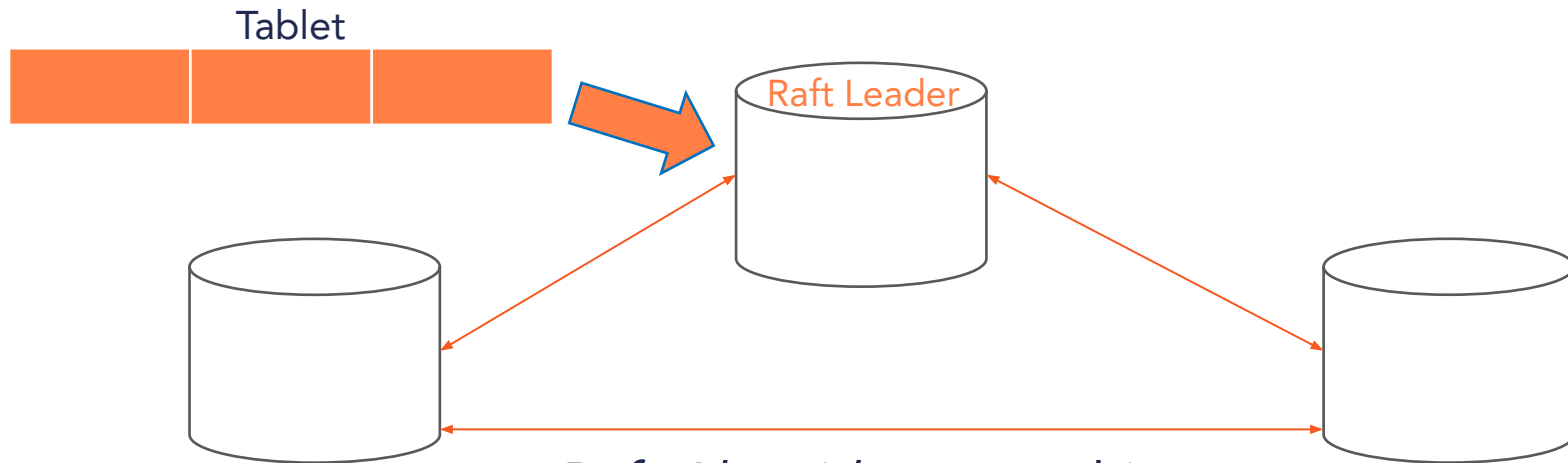
Can connect to ANY node  
Add/remove nodes anytime



# Resilience to Failures



# Replication uses Raft Consensus algorithm



*Raft Algorithm* can achieve per-row consistency across nodes

On failure, HA achieved because new leader elected quickly

# Replication in a 3 node cluster

- Assume  $rf = 3$
- Survives 1 node or zone failure
- Tablets replicated across 3 nodes
- Follower (replica) tablets balanced across nodes in cluster

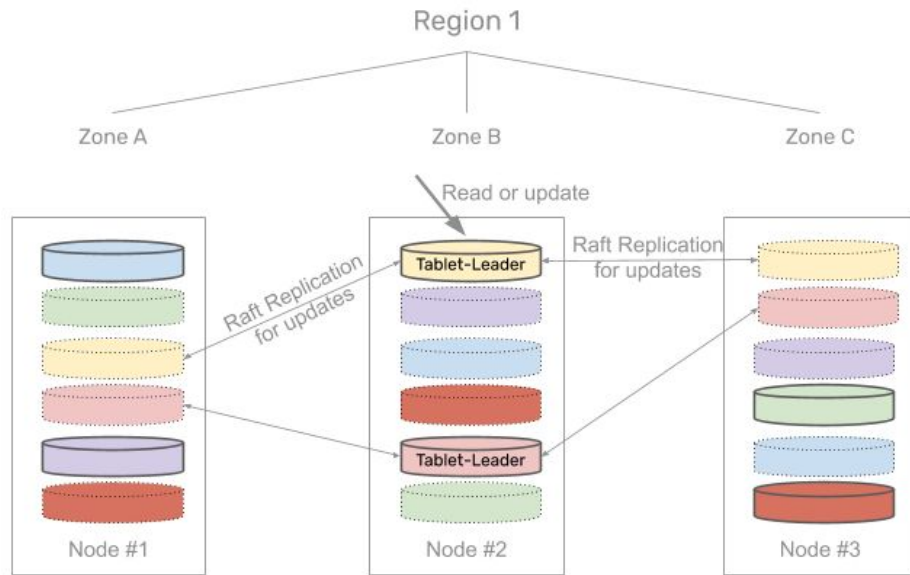
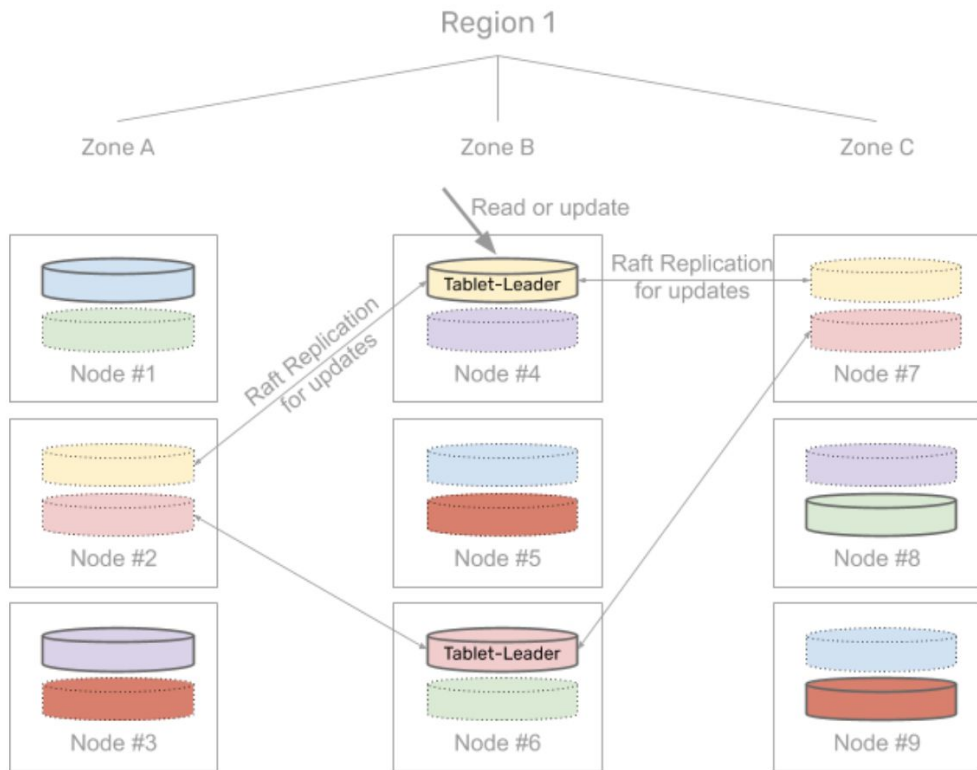


Diagram with replication factor = 3

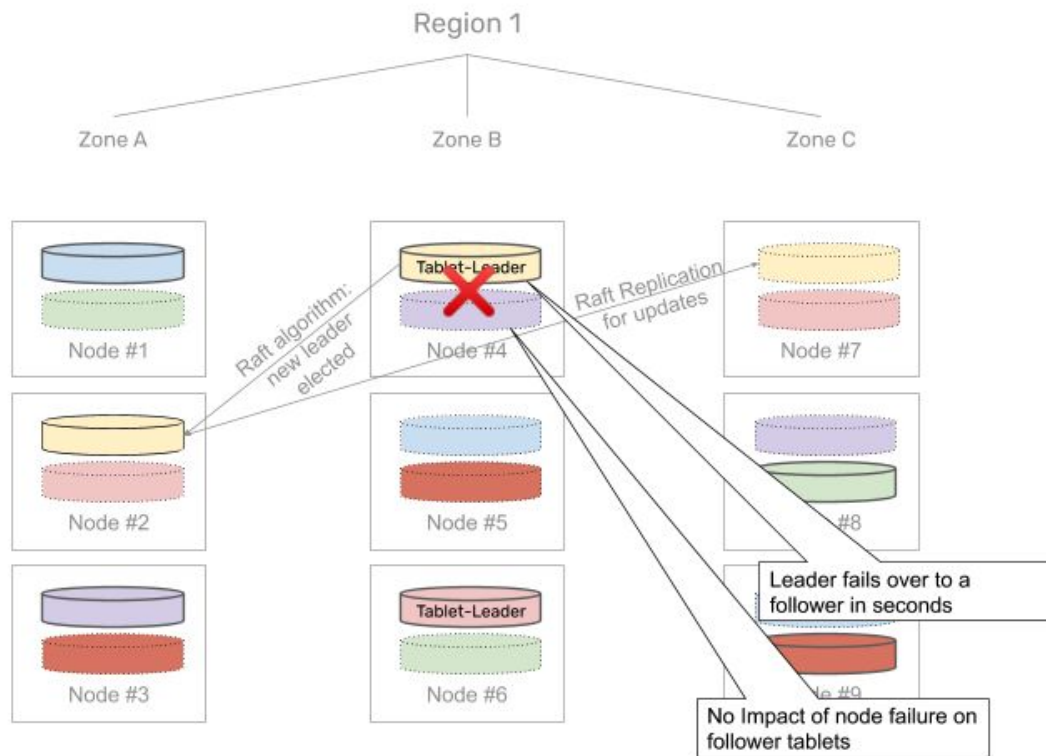
# Scaling to 9 nodes



- Multi-region is similar
- 6 tablets in table
- Replication = 3
- 1 replica per zone

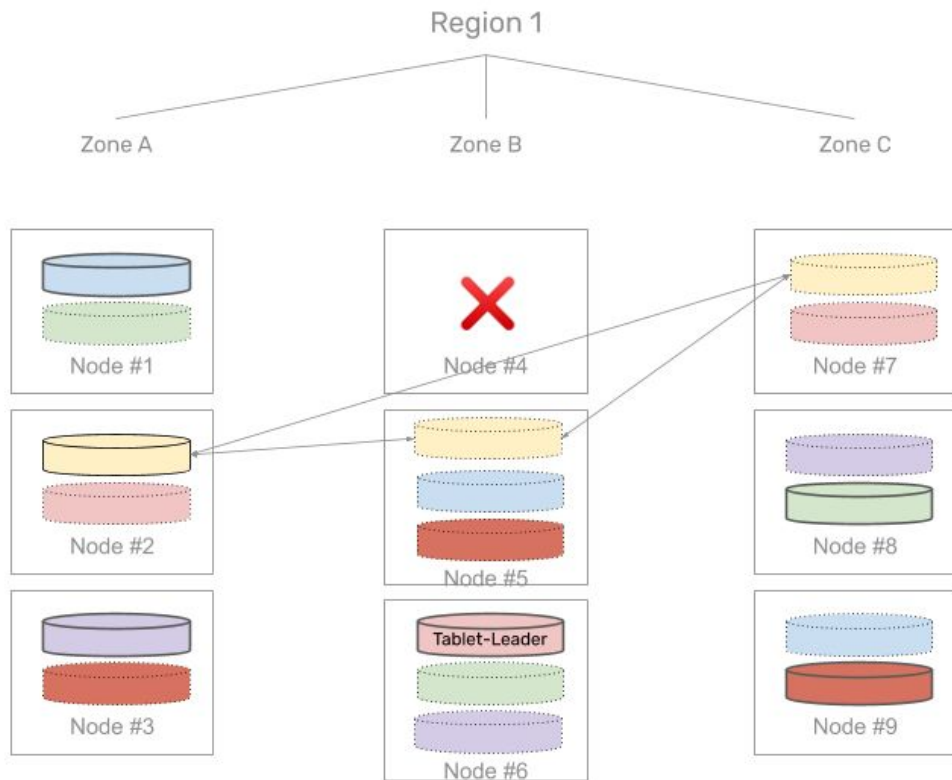


# Tolerating Node Outage



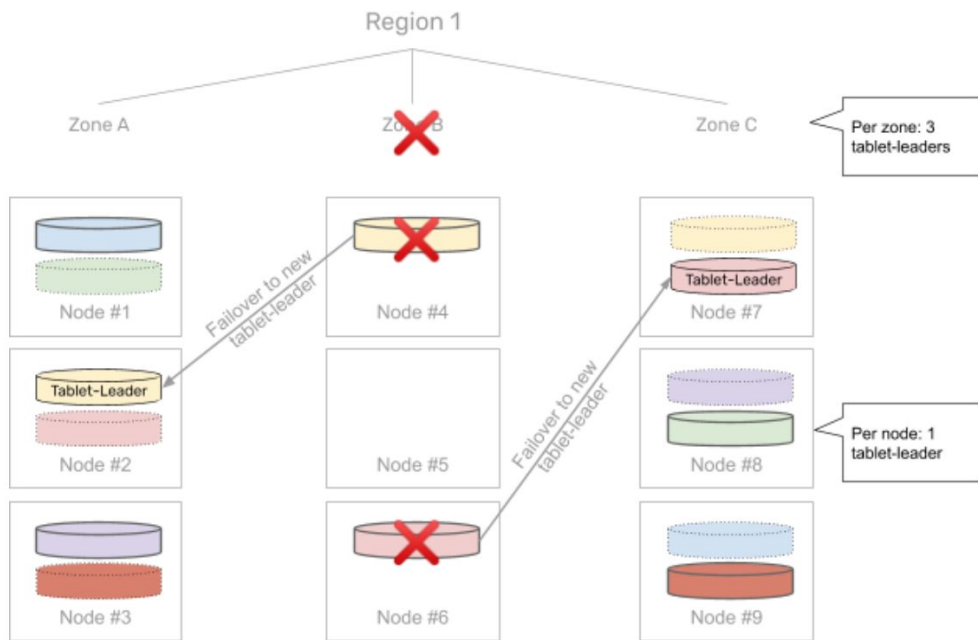
- New tablet leaders re-elected (~3 sec)
- No impact on tablet follower outage
- DB unavailable during re-election window

# Automatic Resilience



- After 15 mins, data is re-replicated (if possible)
- On failed node recovery, automatically catch up
- Tablet leaders auto-rebalanced

# Automatic rebalancing



- New leaders evenly rebalanced
- On failed node recovery, automatically catch up

# Goal:

## Support cross-shard transactions

# Distributed 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**

# Isolation Levels

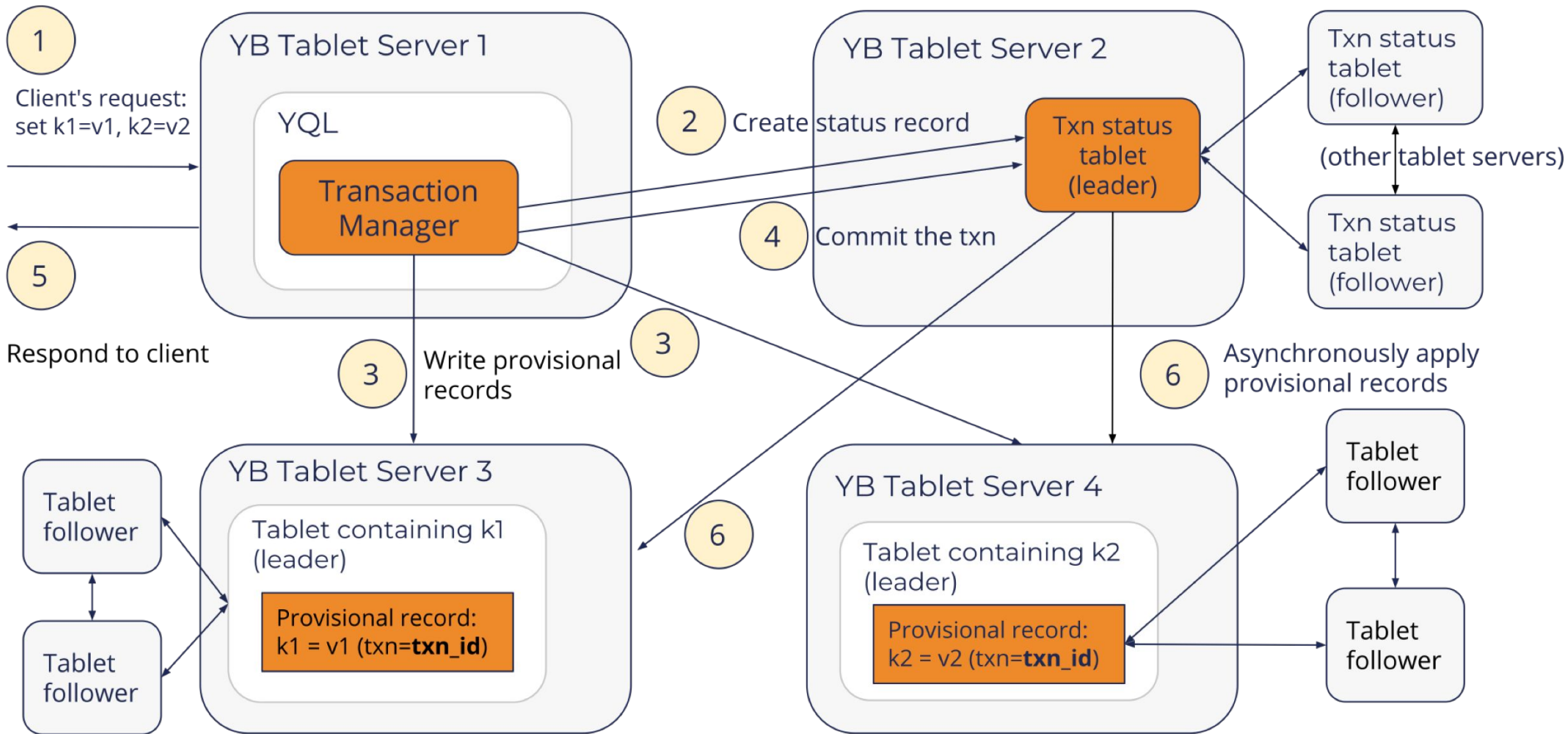
- **Snapshot Isolation**

- All operations to the database happen in a “consistent snapshot”
- Write-write conflicts get auto-detected
- Only writes in read-write txns need provisional records
- Maps to REPEATABLE READ, READ COMMITTED & READ UNCOMMITTED in PostgreSQL

- **Serializable Isolation**

- Snapshot PLUS all transactions happen in some serialized order
- Read-write conflicts get auto-detected
- Both reads and writes in read-write txns need provisional records
- Maps to SERIALIZABLE in PostgreSQL

# Distributed Transactions - Write Path



# Fully Decentralized Architecture

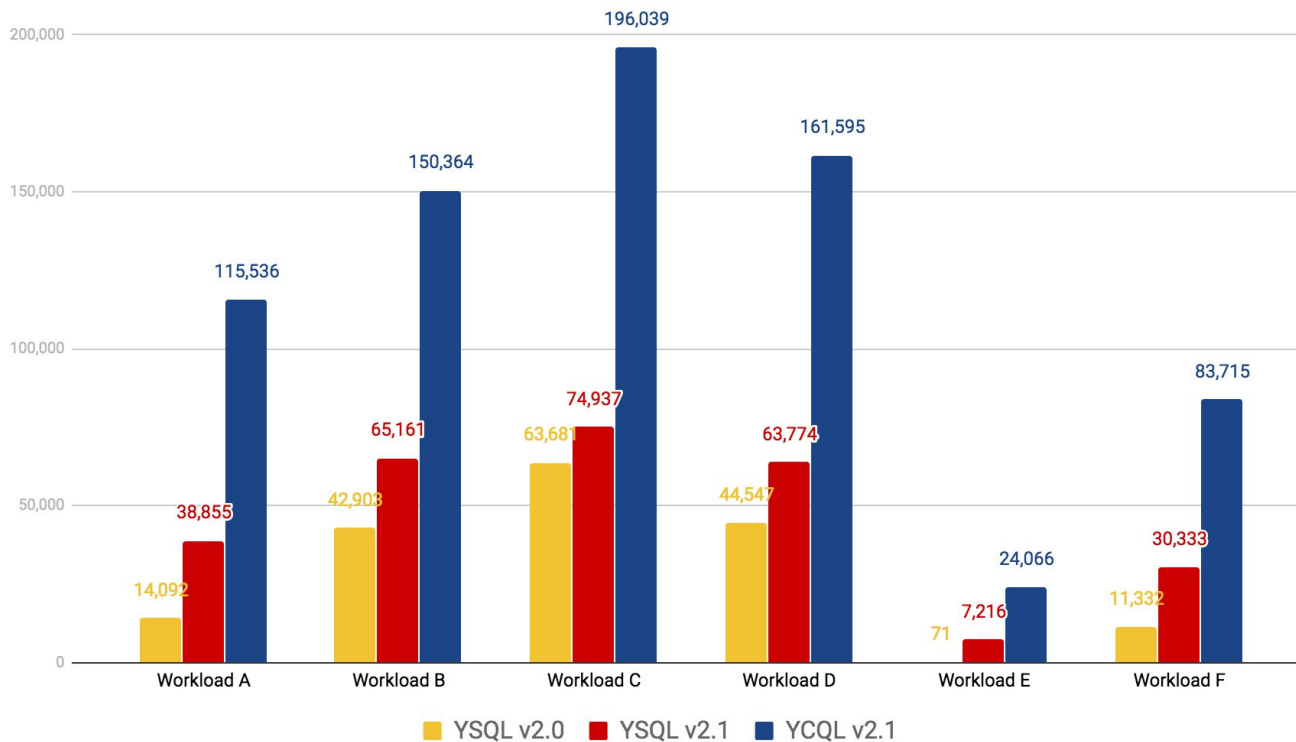
- **No single point of failure or bottleneck**
  - Any node can act as a Transaction Manager
- **Transaction status table distributed across multiple nodes**
  - Tracks state of active transactions
- **Transactions have 3 states**
  - Pending
  - Committed
  - Aborted
- **Reads served only for Committed Transactions**
  - Clients never see inconsistent data



# Results

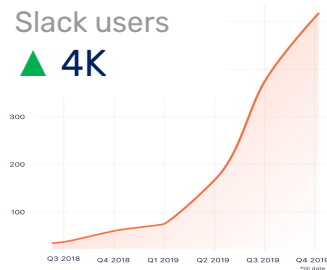
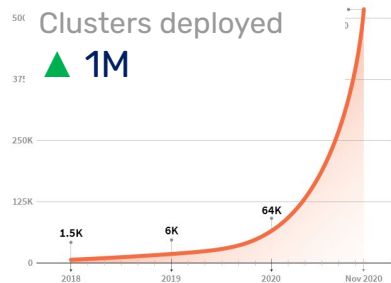
# YCSB Performance

YCSB benchmark



# INSERT INTO Yugabyte (We're hiring)

# Fast growing Distributed SQL Project



Powers business-critical apps at scale.



**27B+**

Ops/Day

**xignite**

**10B+**

Ops/Day

**CIIPHERTRACE**

**3B+**

Ops/Day

**nanvan**

**1B+**

Ops/Day

Join our community:

[yugabyte.com/slack](https://yugabyte.com/slack)

[github.com/YugaByte/yugabyte-db](https://github.com/YugaByte/yugabyte-db)



# We're HIRING !!

---

## Full Time Positions [Sunnyvale, US; Toronto, Canada]:

- Software Engineer, Distributed Storage & Transactions (DST)
- Software Engineer, Languages & Relational Technologies (LRT)
- Backend Engineer, Platform & Cloud Infrastructure
- Frontend Engineer, Platform & Cloud Infrastructure

## Internships [Sunnyvale, US; Toronto, Canada]:

- Software Engineering Intern, Core Database
- Software Engineering Intern - Backend, Platform & Cloud Infrastructure
- Software Engineering Intern - Frontend, Platform & Cloud Infrastructure

See <https://www.yugabyte.com/careers/> for the complete list of openings

Reach out to our recruiter Sean Nadir (snadir@yugabyte.com)

# Thanks!