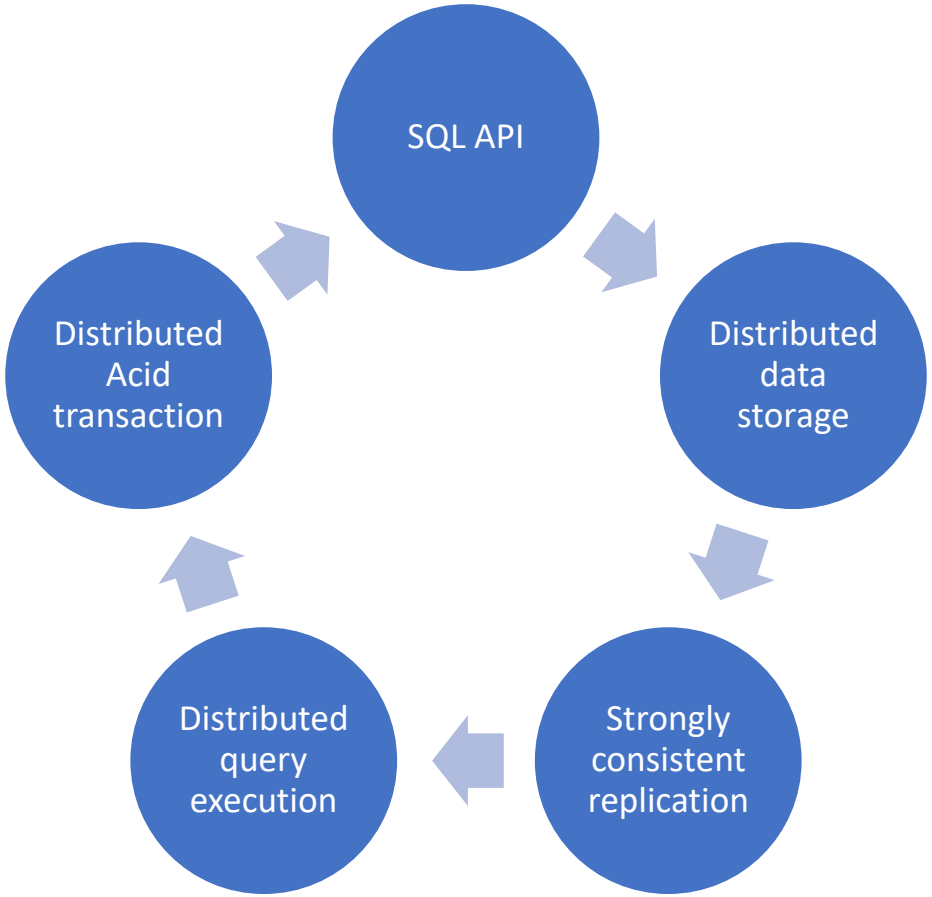
Distributed SQL : Introduction

What is distributed SQL?	A Distributed SQL database automatically distributes and strongly replicates data so that SQL can be executed against it, in a distributed and ACID compliant manner.



Distributed SQL vs NoSQL

Characteristic	Distributed SQL	NoSQL
SQL API	Fully Relational	Semi or Non-Relational
Distributed Data Storage	Automatic	Automatic or Manual
Replication	Strong Consistency	Eventually Consistent
Distributed Query Execution	Yes	Yes
Distributed ACID Transactions	Yes	No

Distributed SQL : Compare across Distributed SQL

Comparing Distributed SQL Databases

	Amazon Aurora	Google Cloud Spanner	PingCap's TiDB	CockroachDB	YugabyteDB
SQL & Transactions Compatibility	PostgreSQL & MySQL (Full Compatibility)	Proprietary SQL (No Foreign Keys)	MySQL (No Foreign Keys & Serializable Isolation)	PostgreSQL (No Partial Indexes, Stored Procedures & Triggers)	PostgreSQL (Full Compatibility)
Native Failover/Repair	✓	✓	✓	✓	✓
Horizontal Write Scalability	✓ (Multi-Master)	✓ (Auto-Sharded)	✓ (Auto-Sharded)	✓ (Auto-Sharded)	✓ (Auto-Sharded)
Geographic Data Distribution	✓ (Only a Single Region Can Take Writes)	✓ (Global Clock Sync for Consistency)	✓ (Single Region Timestamp Generator Leads to High Latency)	✓ (Global Clock Sync for Consistency)	✓ (Global Clock Sync for Consistency)
High Performance	✗ (Read Replicas Cannot Process Writes)	✗ (Not Optimized for High Volume Ingest)	✓ (High Latency for Multi-Region Clusters)	✗ (Not Optimized for High Volume Ingest)	✓ (Optimized for High Volume Ingest)
Cloud Neutral w/ Kubernetes Native	✗ (Proprietary to AWS)	✗ (Proprietary to Google Cloud)	✓	✓	✓
Open Source	✗	✗	✓ (Apache 2.0)	✗ (BSL 1.0 is not Open Source)	✓ (Apache 2.0)

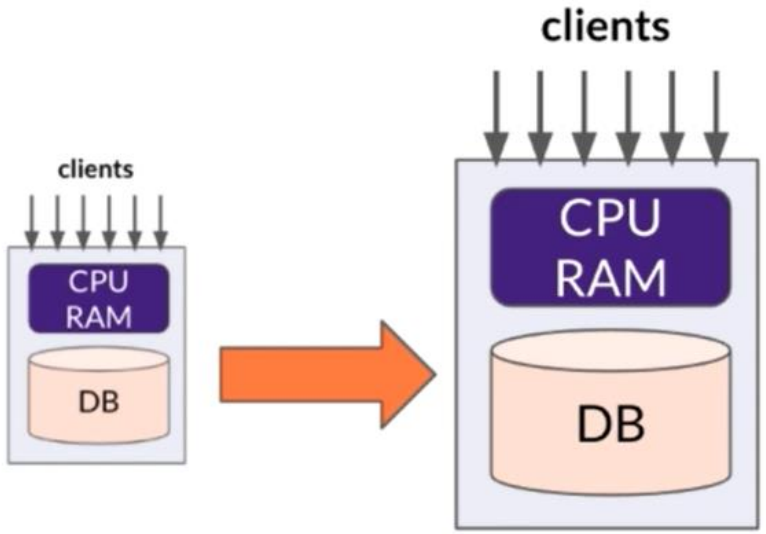
Scaling: Monolithic vs Distributed Databases

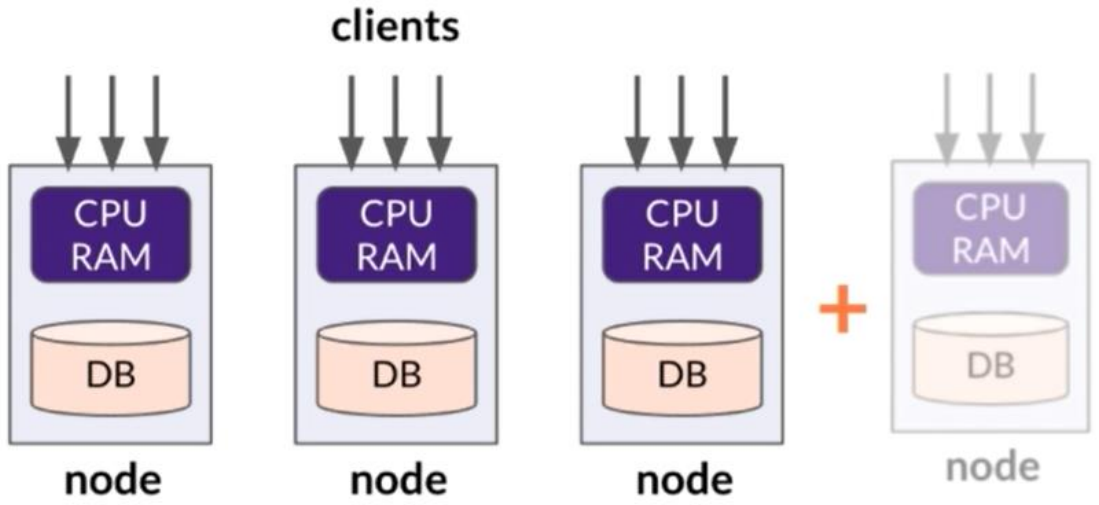
Monolithic Databases Scale Up

- Serve writes from a single node
- Scale up vertically
- Cannot tolerate failures to the primary node

Distributed Databases Scale Out

- Serve writes on every node
- Scale out horizontally
- Tolerate failures to any node





CAP Theorem

"In the face of a network Partition, it's possible to solve for either Consistency or Availability, but not both."

- **Consistency -** Every read receives the most recent write or an error
- **Availability -** Every request receives a (non-error) response, without the guarantee that it contains the most recent write
- **Partition Tolerance -** Continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes
- Like Google Spanner, YugabyteDB is a CP database

Distributed SQL : Raft consensus algorithm for YugabyteDB

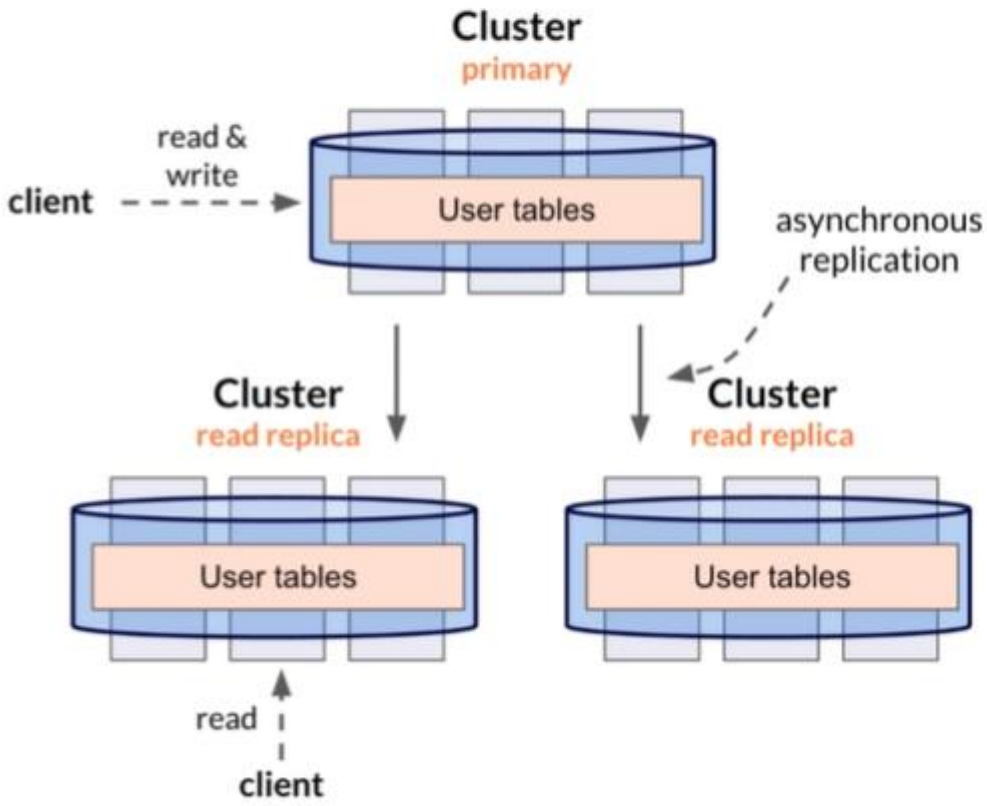
Raft_Consensus_algo

Raft Consensus Algorithm

- Raft is a consensus algorithm that is designed to be easy to understand
- It's equivalent to Paxos in fault-tolerance and performance.
- Unlike Paxos, it's decomposed into relatively independent subproblems that can be solved more cleanly
- YugabyteDB uses Raft to achieve consensus amongst leaders and followers in a "tablet peer group"

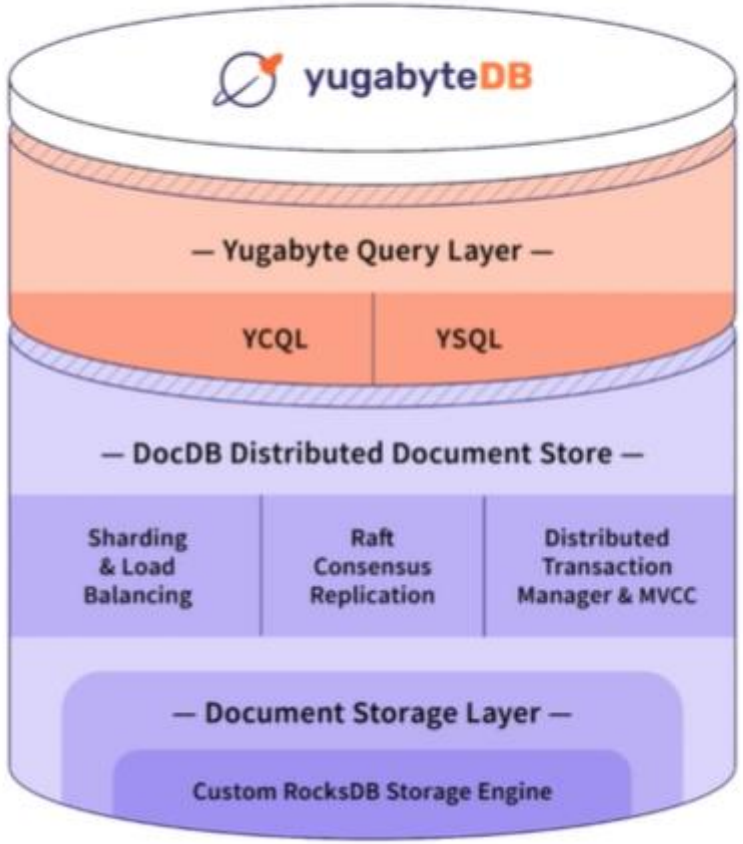
YugabyteDB Components

- Nodes are machines, VMs, or containers
- A universe/cluster is a group of nodes that collectively function as a clustered database
- A cluster is composed of 3+ nodes
- There can be primary cluster and optional read replicas
- Replication is synchronous within a primary cluster
- Replication is asynchronous from primary to read replicas



Distributed SQL : Architecture overview for YugabyteDB

Architecture Overview

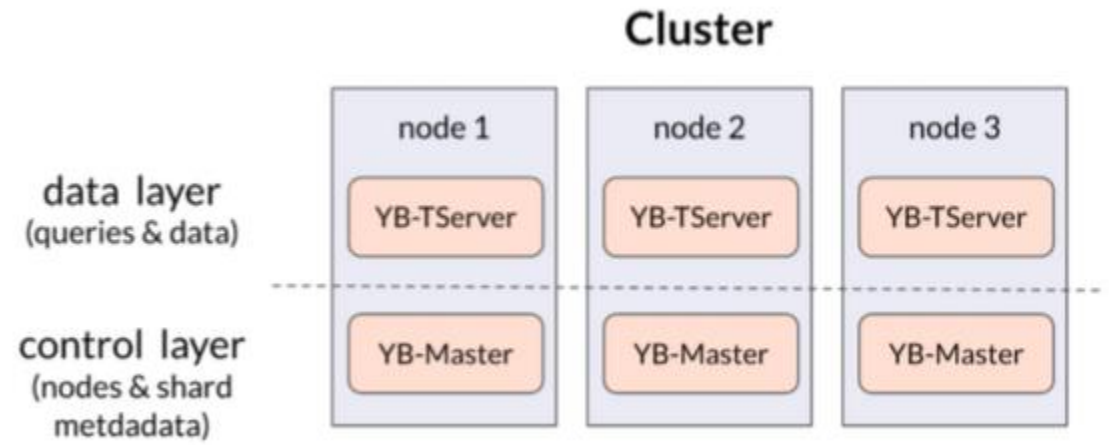


YugabyteDB Services

A cluster relies on two processes:

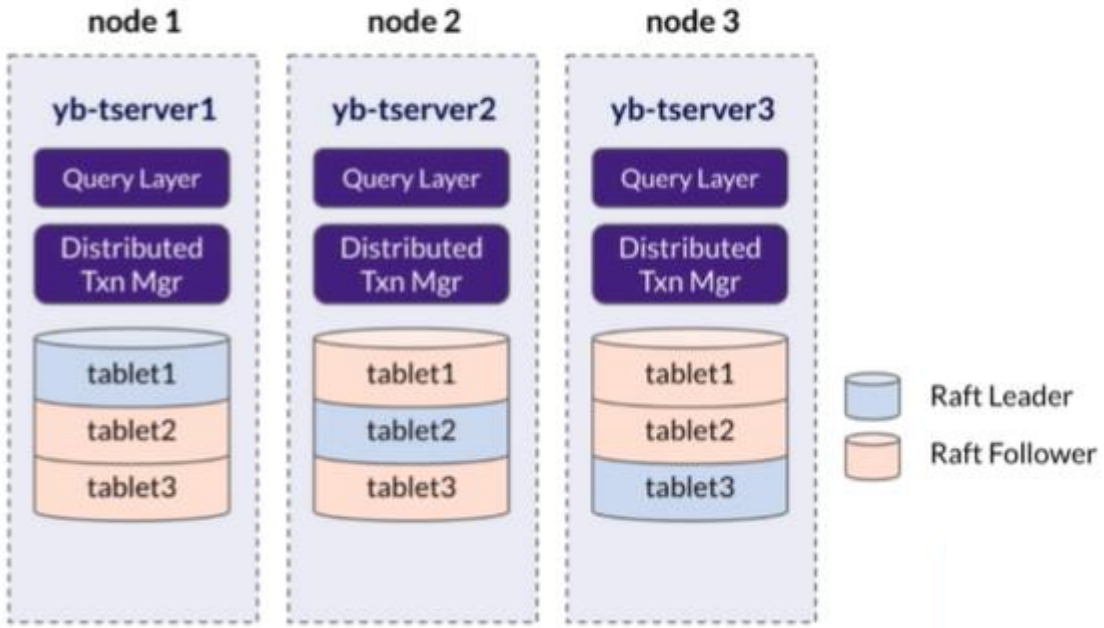
- YB-TServer - hosts and serves data
- YB-Master - stores metadata for the system

Process Name	Memory	Threads	Ports	PID
yb-tserver	202.3 MB	122	263	23833
yb-tserver	207.5 MB	91	199	24098
yb-tserver	206.7 MB	90	197	24101
yb-master	139.5 MB	43	86	23824
yb-master	154.0 MB	38	63	24094
yb-master	155.0 MB	38	65	24091



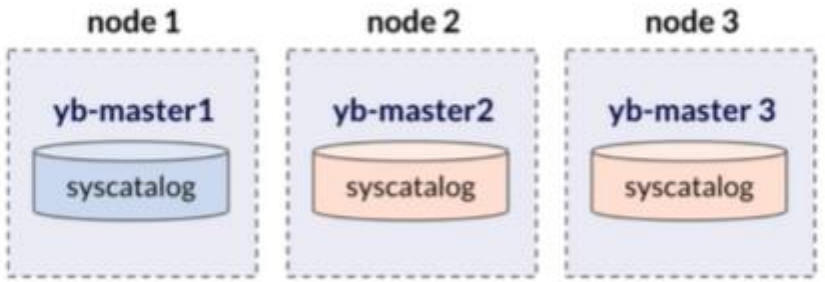
YB-TServer Service

- Hosts and serves user data
- Tables are split into tablets (shards)
- Tablets are synchronously replicated
- Replicated tablets form a Raft group with a leader and followers



YB-Master Service

- Manages system metadata including tables, tablets, nodes, clusters
- Coordinates system-wide operations such as create/alter/drop tables
- Initiates maintenance operations

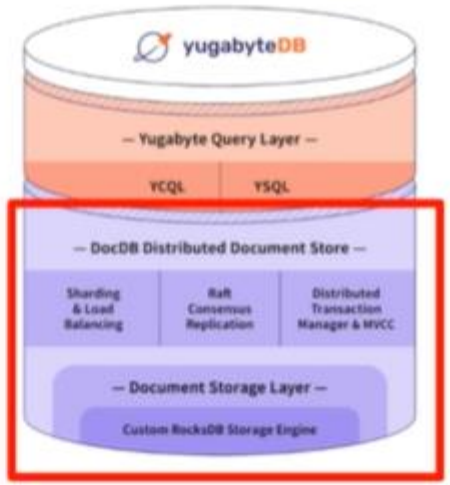


DocDB Document Store Overview

DocDB is YugabyteDB's distributed document store responsible for transactions, sharding, replication, and persistence with roots in RocksDB.

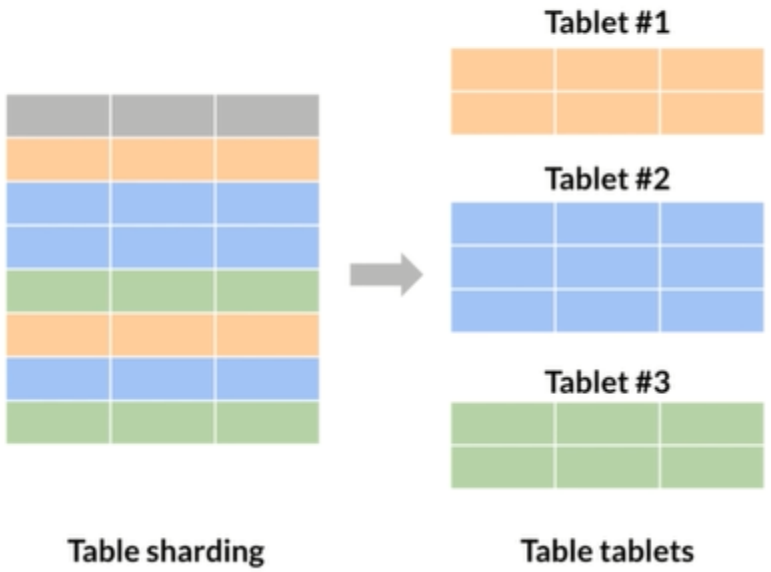
Features

- Strong write consistency
- Extreme resilience to failures
- Automatic sharding and load balancing
- Zone/region/cloud aware data placement policies
- Tunable read consistency



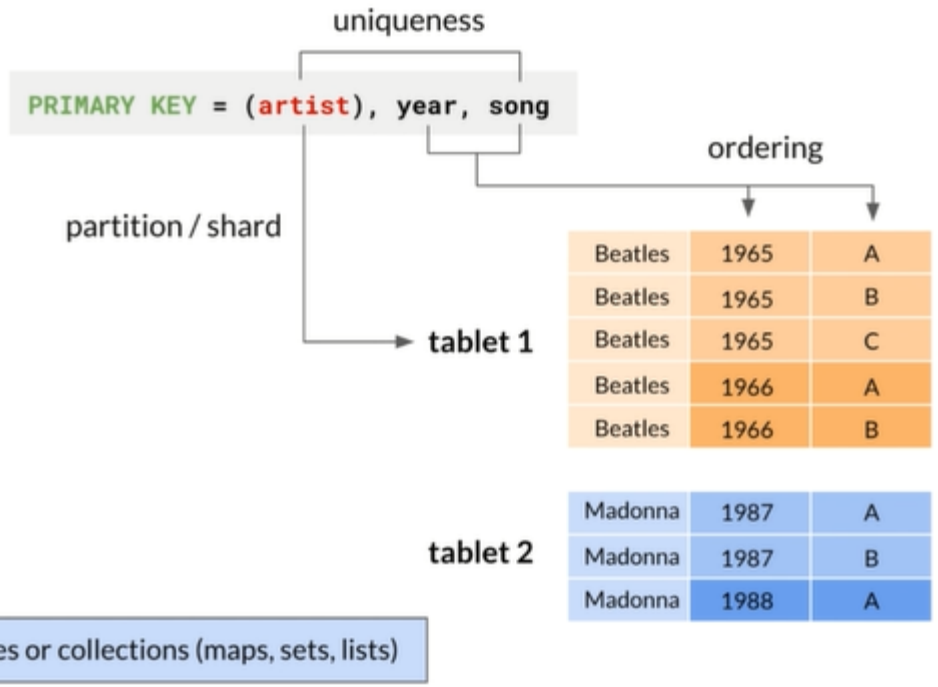
Sharding Data

- Tables are sharded into tablets by rows
- A row's primary key determines the tablet that owns it and the node it resides in.
- Multiple sharding strategies are supported for primary keys including hash and range.



Storage Layer

- Every row is a document with a primary key
- A primary key consists of one or more hash components and zero or more ordering components
- Keys determine row uniqueness and shard location

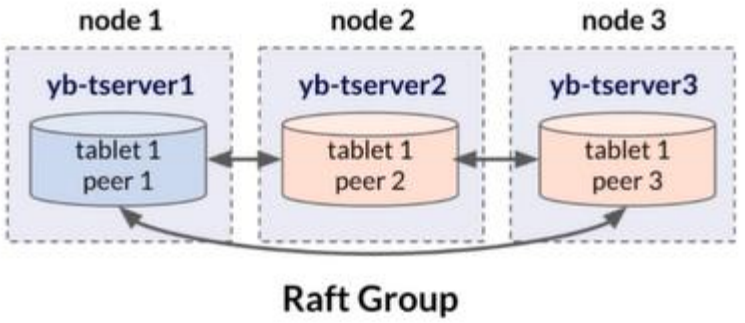


Values can be primitives or collections (maps, sets, lists)

Distributed SQL : Replication approach for YugabyteDB

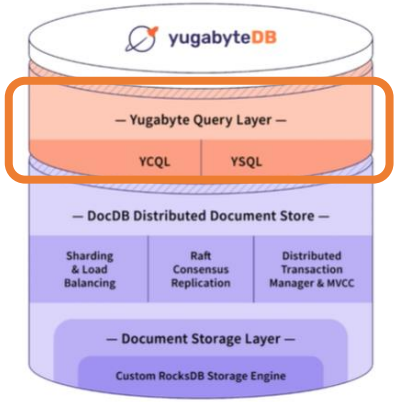
Replicating Data

- YugabyteDB's replication design is inspired by Google Spanner
- Tablet peers form a Raft group
- Peers replicate synchronously
- They can span nodes, zones, racks, regions, or providers
- Writes are logged to a majority of peers, then applied to storage



Query Layer Overview

- Query/command compilation
- Run-time (data type representations, built-in operations, etc)
- A “statement cache” for prepared statements
- A command parser and execution layer
- YSQL - PostgreSQL-compatible API for relational workloads
- YCQL - Cassandra inspired API for semi-relational workloads
- Clients connect with drivers (Java, Python, Node, Go, Ruby, C#, etc)



YSQL Overview

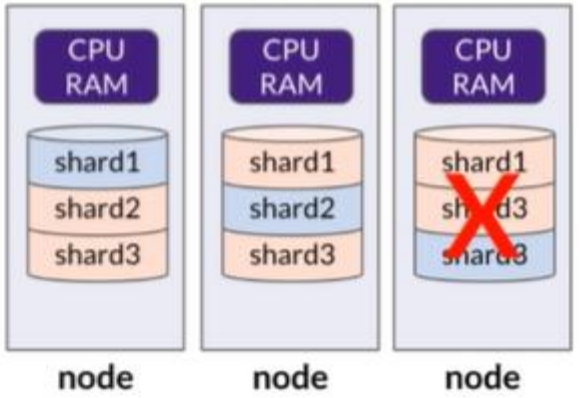
- YSQL is a PostgreSQL-compatible API for relational workloads
- Best for scale-out RDBMS applications needing ultra resilience, massive write scalability and geographic data distribution
- Supports FKs, JOINs, distributed transactions, partial indexes, triggers, stored procedures and some PostgreSQL extensions
- Compatible with PostgreSQL 11.2

YCQL Overview

- YCQL is a Cassandra inspired API for semi-relational workloads
- Best for OLTP and HTAP applications needing massive data ingestion and blazing-fast queries
- Supports strongly consistent secondary indexes, a native JSON type and distributed transactions

Node Count, Replication Factor & Fault Tolerance

- Fault tolerance (FT) is the max number of node failures a cluster can survive while still preserving the correctness of data.
- YugabyteDB replicates data across nodes in order to tolerate faults.
- Nodes hold copies of data, *but no single node holds all the data*
- The replication factor (RF) is the number of copies of data in the cluster.
- To achieve a FT of k nodes, the cluster has to be configured with a RF = (2k + 1).
- **Example:** To survive the failure of 1 node, you need an RF of 3.

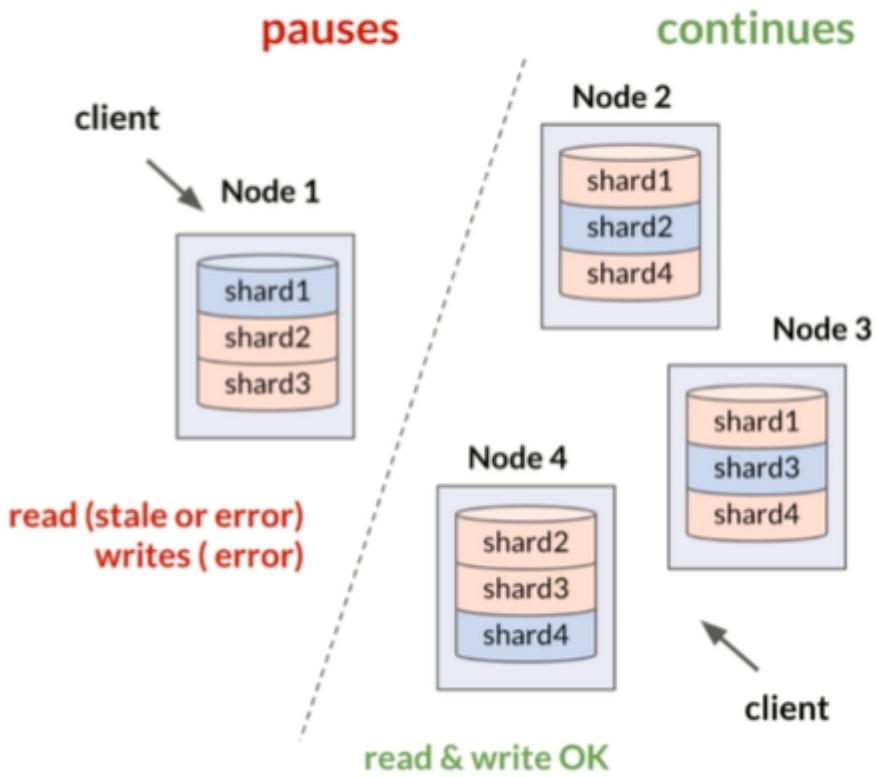


Cluster Deployment Configurations

- **Recall:** The data in a cluster is replicated synchronously between nodes
- All nodes can be in a single zone
- Nodes can be in multiple zones
- Nodes can be in multiple regions that are geographically replicated
- Nodes can be in multiple clouds (both public and private clouds)
- Additional clusters can be deployed that are being asynchronously replicated to

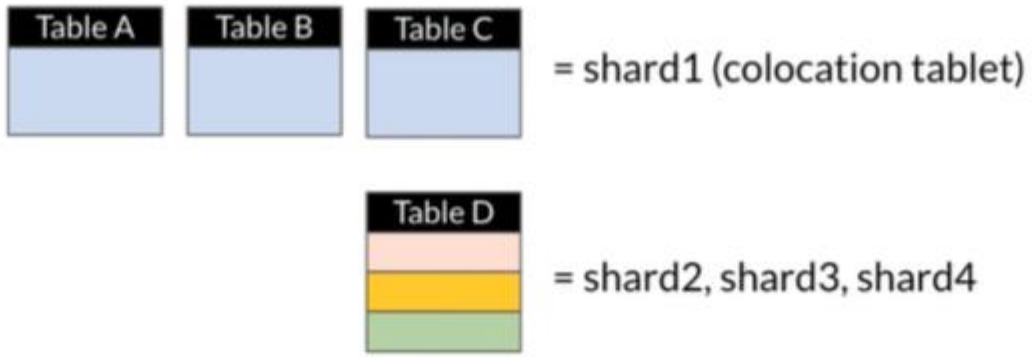
Handling Network Partitions in YugabyteDB

- One side pauses and the other continues
- Each side can determine if it has more or less nodes talking than the other
- The majority writes and gets latest reads
- The minority stops writing and gets no reads or stale data
- This architectural design is similar to Google Cloud Spanner, which is also a CP system.



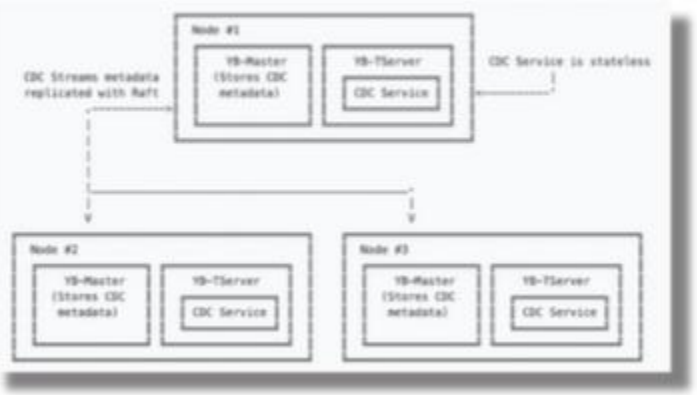
Colocated Tables

- Can dramatically increase the number of relations (tables, indexes, etc.) that can be supported per node while keeping the number of tablets per node low.
- All of their data is in a single tablet, but, the tablet is replicated on multiple nodes

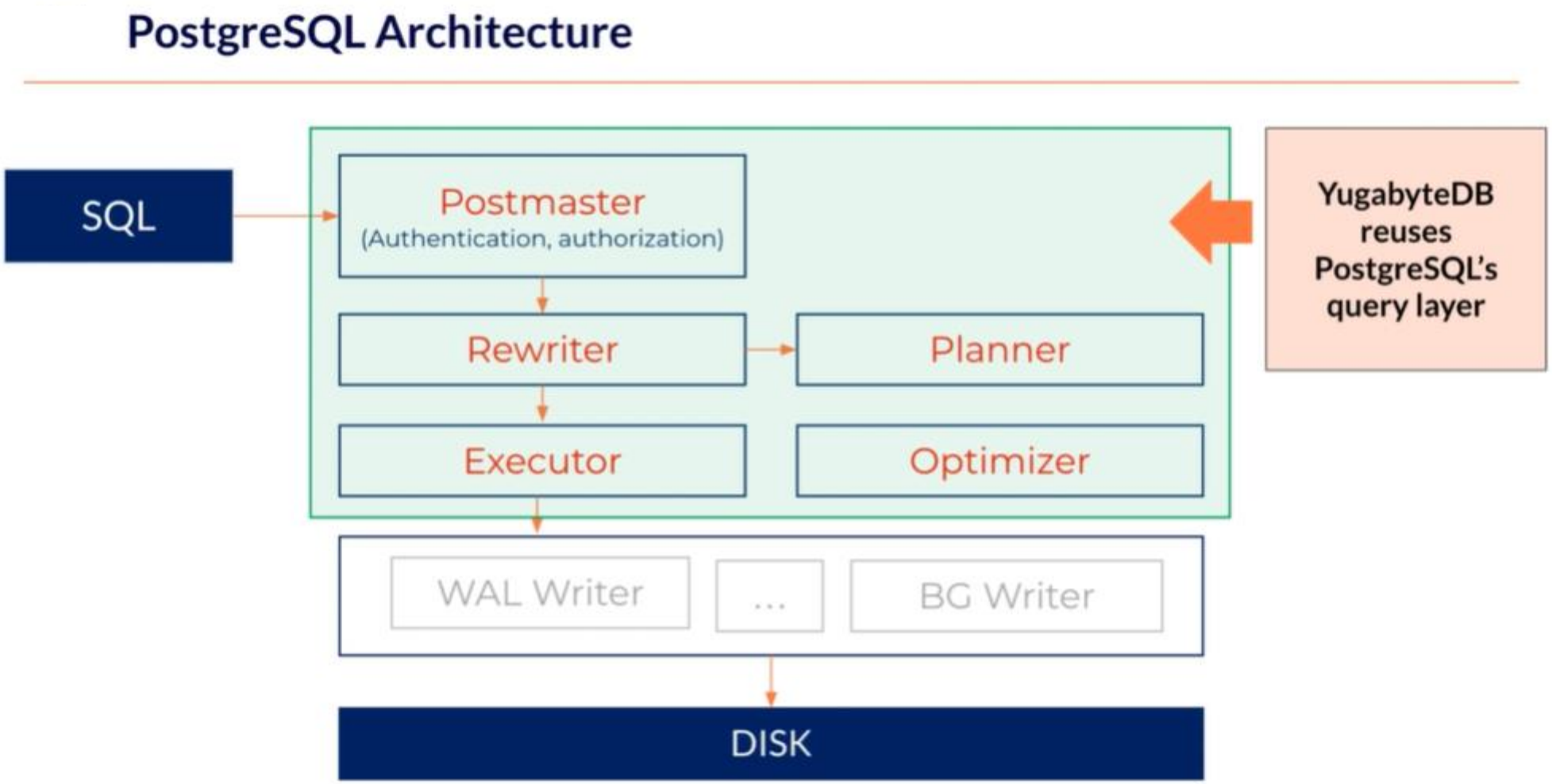


Change Data Capture (CDC)

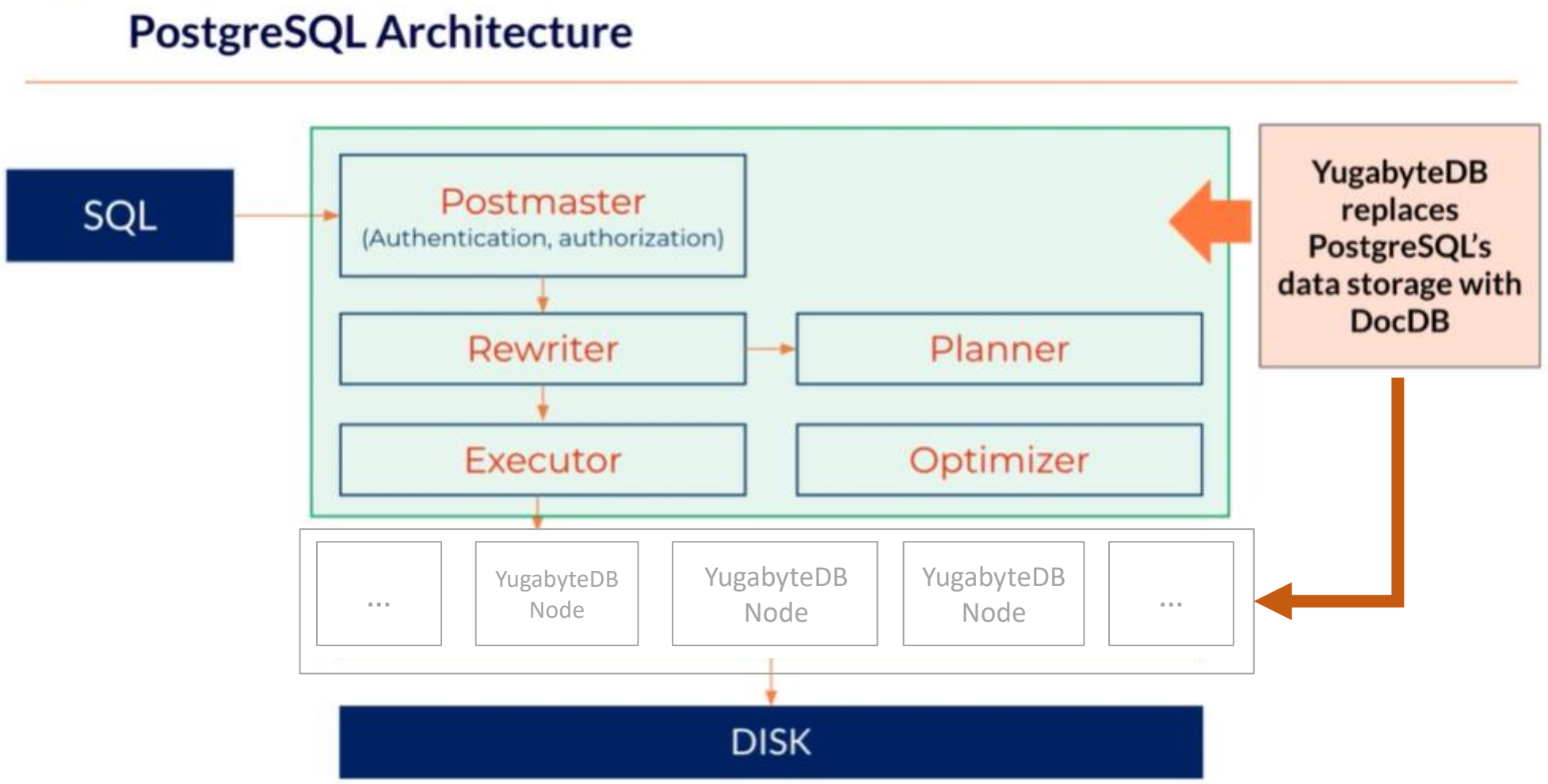
- Ensures that any changes in data are identified, captured, and automatically applied to another data repository instance or made available for consumption by applications and other tools.
- For example changes in YugabyteDB can be sent to Kafka



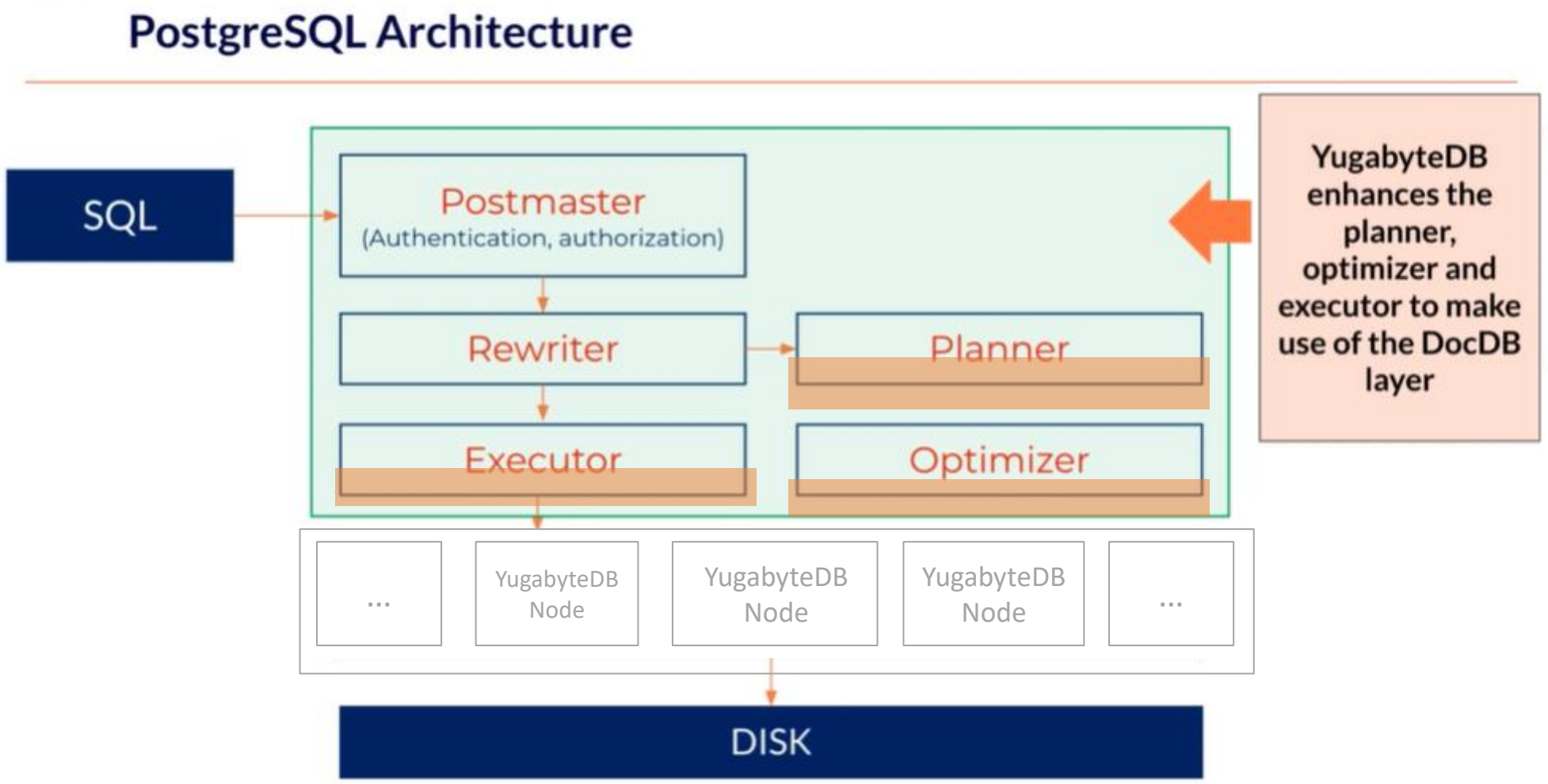
Distributed SQL : Understanding of PostgreSQL architecture for YugabyteDB



Distributed SQL : PostgreSQL data storage replaced with DocDB for YugabyteDB

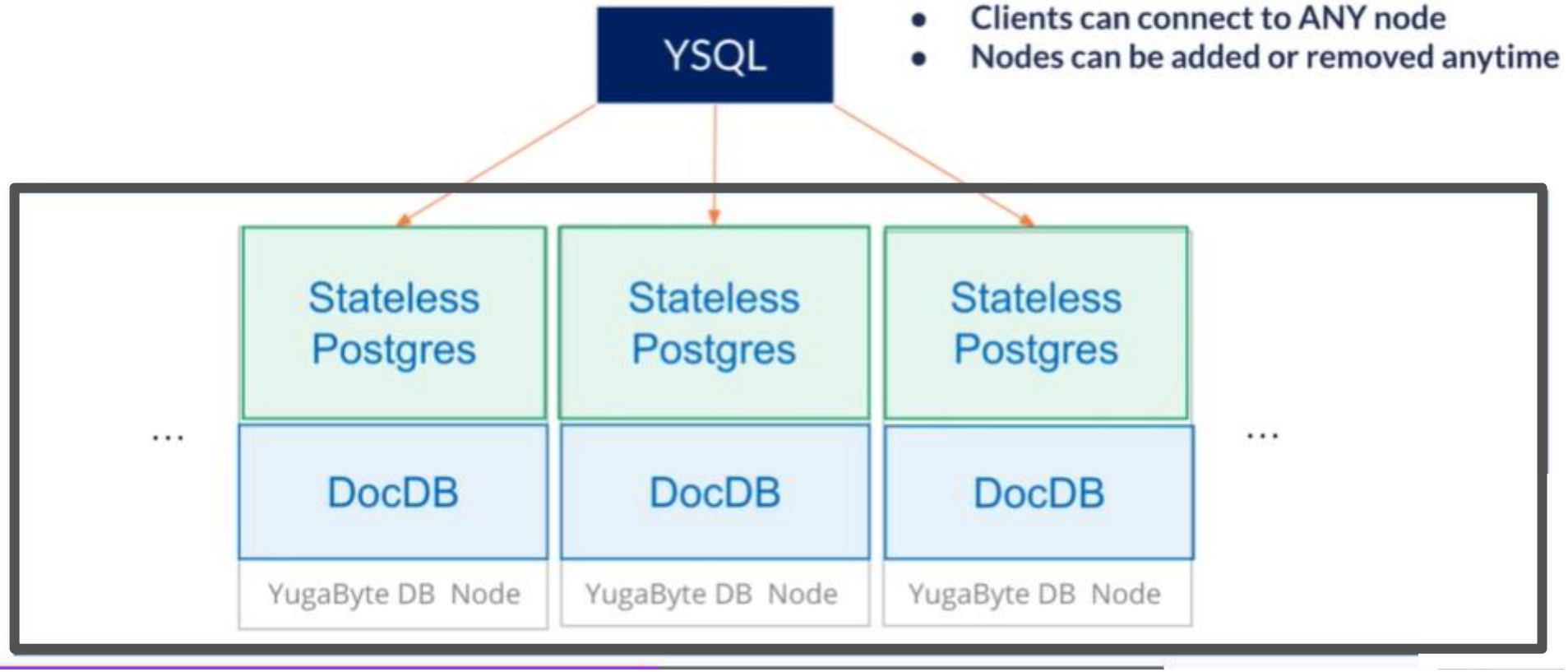


Distributed SQL : PostgreSQL architecture components with modification for YugabyteDB



Distributed SQL : Stateless PostgreSQL for YugabyteDB

All Nodes are Identical



Distributed SQL : Self healing of nodes for YugabyteDB

Self-Healing Against Failures