yugabyteDB

YugabyteDB: a distributed PostgreSQL database

Bryn Llewellyn

Developer Advocate, Yugabyte, Inc

YugaByte DB

Distributed SQL

PostgreSQL Compatible, 100% Open Source (Apache 2.0)

Massive Scale

Millions of IOPS in Throughput, TBs per Node

High Performance

Low Latency Queries

Cloud Native

Fault Tolerant, Multi-Cloud & Kubernetes Ready

Design Goals

- **PostgreSQL compatible**
  - Re-uses PostgreSQL query layer
  - New changes do not break existing PostgreSQL functionality

- **Enable migrating to newer PostgreSQL versions**
  - New features are implemented in a modular fashion
  - Integrate with new PostgreSQL features as they are available
  - E.g. Moved from PostgreSQL 10.4 → 11.2 in 2 weeks!

- **Cloud native architecture**
  - Fully decentralized to enable scaling to 1000s of nodes
  - Tolerate rack/zone and datacenter/region failures automatically
  - Run natively in containers and Kubernetes
  - Zero-downtime rolling software upgrades and machine reconfig

Functional Architecture

YugaByte SQL (YSQL)

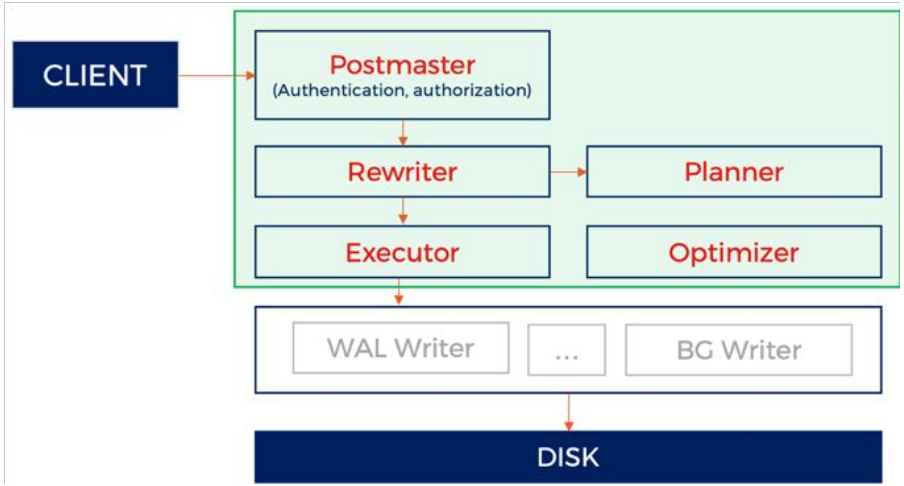
PostgreSQL-Compatible Distributed SQL API

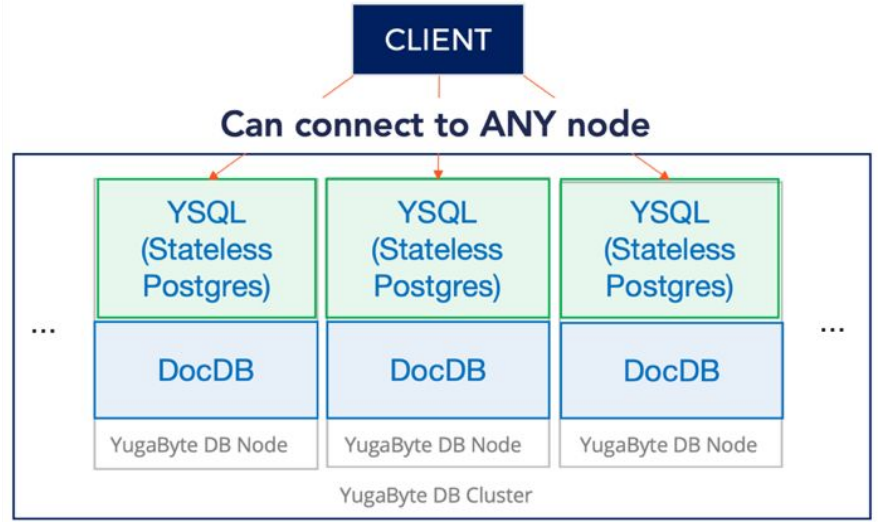
DOCDB

Spanner-Inspired Distributed Document Store

Cloud Neutral: No Specialized Hardware Needed

PostgreSQL Transformed into Distributed SQL





SQL Feature Depth

- **Traditional SQL**
  - Data types
  - Relational integrity (Foreign keys)
  - Built-in functions
  - Expressions
  - JSON column type
  - Secondary indexes
  - JOINs
  - Transactions
  - Views
- **Advanced SQL**
  - Partial indexes
  - Stored procedures
  - Triggers
  - Extensions

  And more ...

Create Table & Insert Data

YSQL Tables

- **Tables**
  - Each table maps to one DocDB table
  - Each DocDB table is sharded into multiple tablets
- **System tables**
  - PostgreSQL system catalog tables map to special DocDB tables
  - All such special DocDB tables use a single tablet
- **(Internal) DocDB tables**
  - Have same key → document format
  - Schema enforcement using the table schema metadata

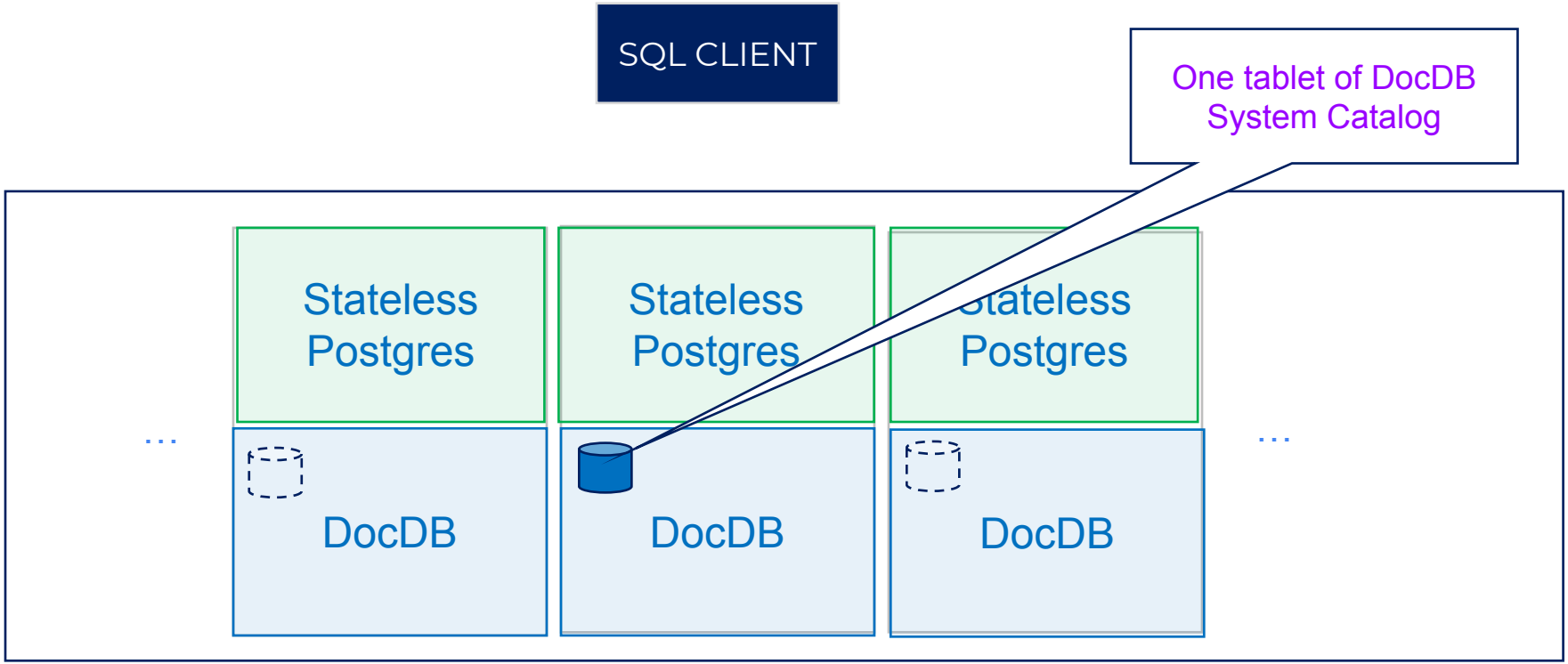
Three important numbers

- **Replication factor (RF) — an odd number: 3, 5, ...**
  - Each (conceptual) *tablet* is stored as RF identical *tablet peers* each on its of node
  - Fixed at cluster creation time

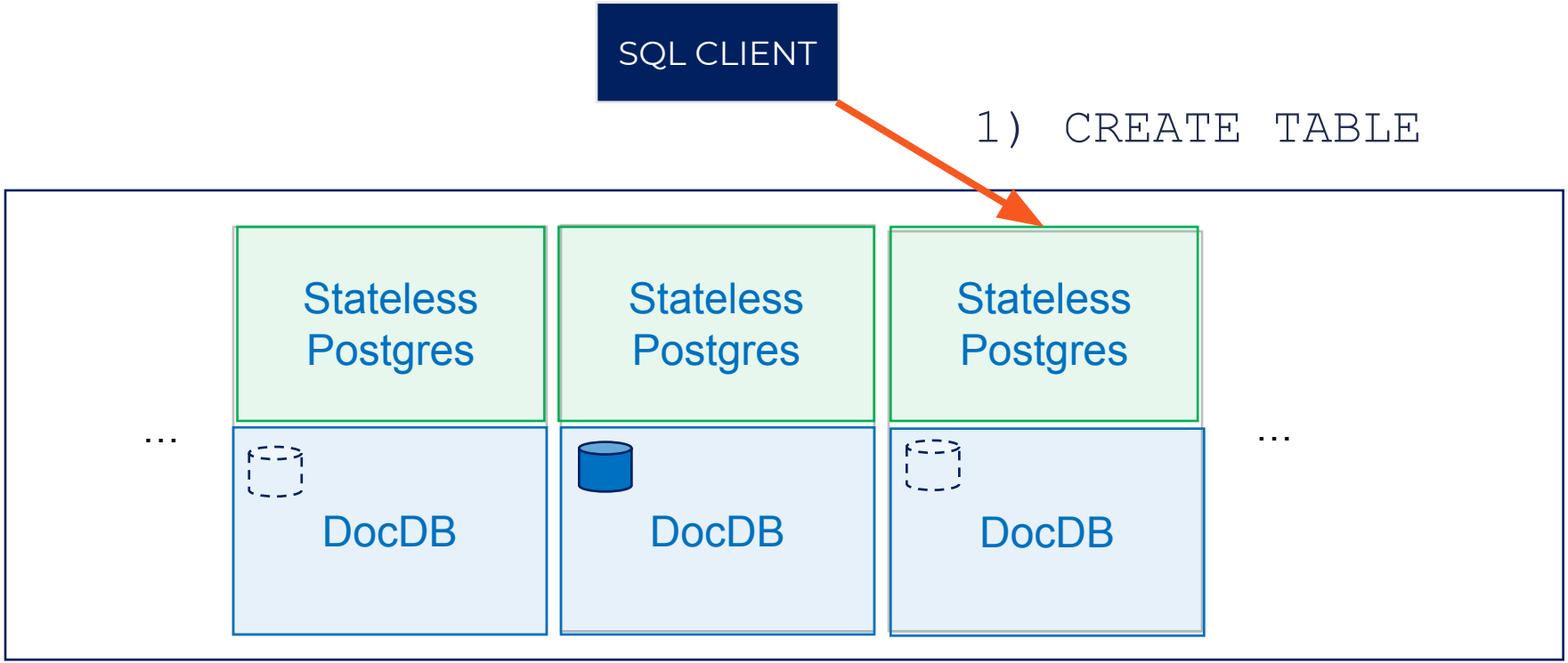
- **Number of nodes**
  - At least equal to RF — but as big as you need.
  - Can increase — or decrease — on demand

- **Number of tablets per table**
  - Currently: fixed by RF and initial number of nodes. Planned: user-specifiable per table.

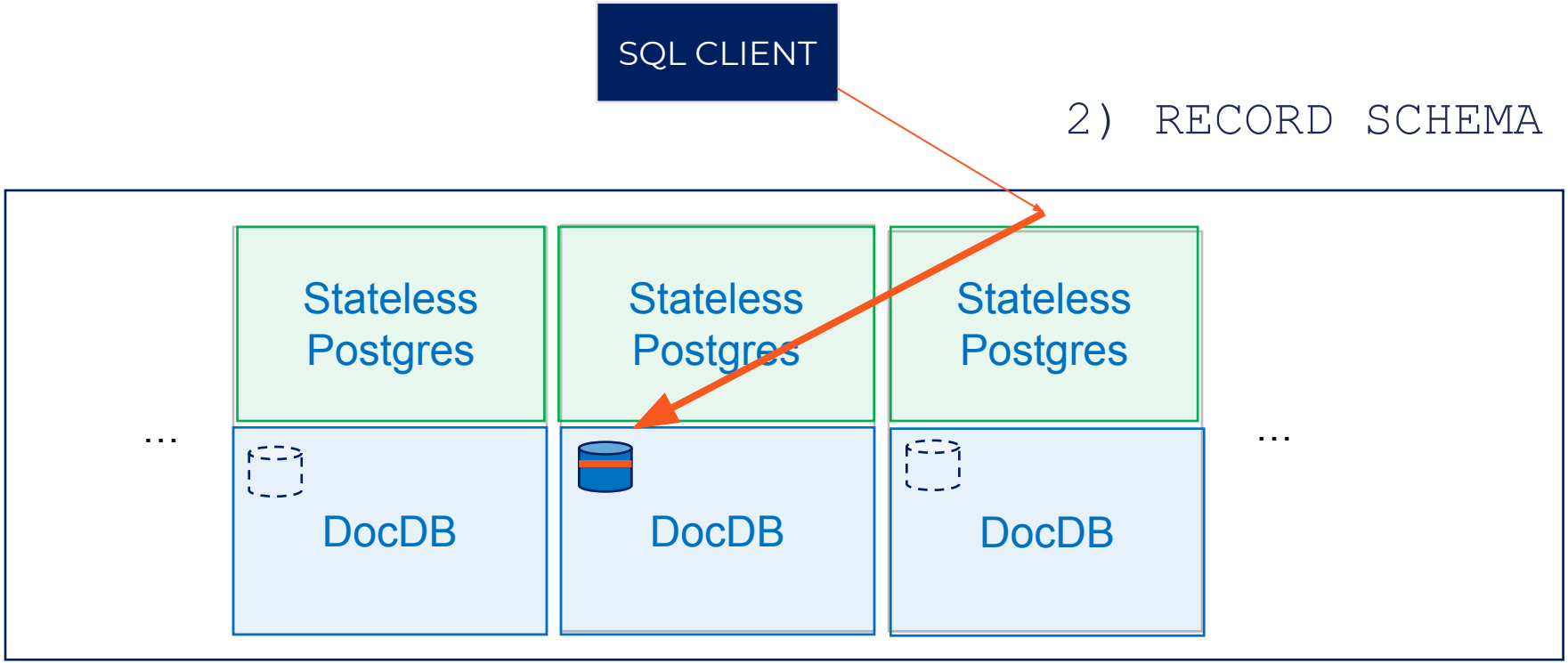
System Catalog Tables are Special Tables



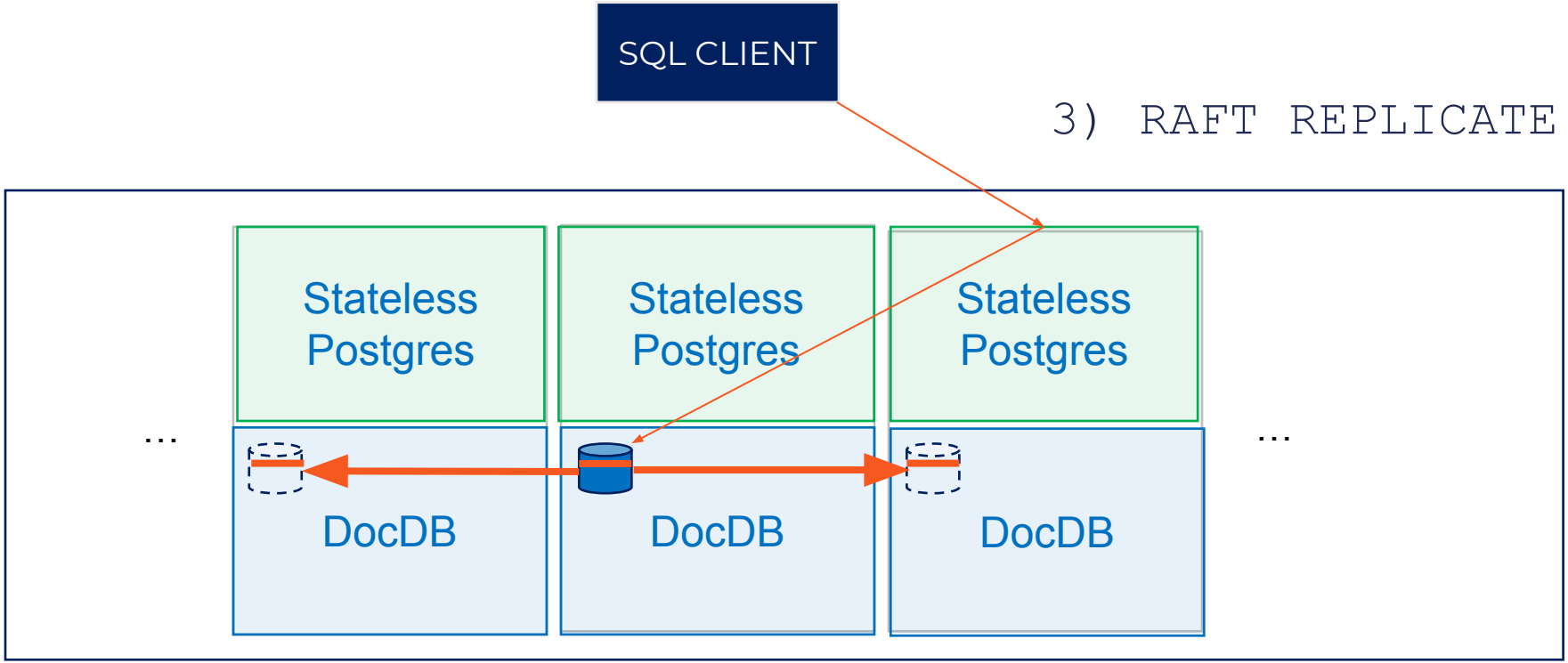
Create a Table



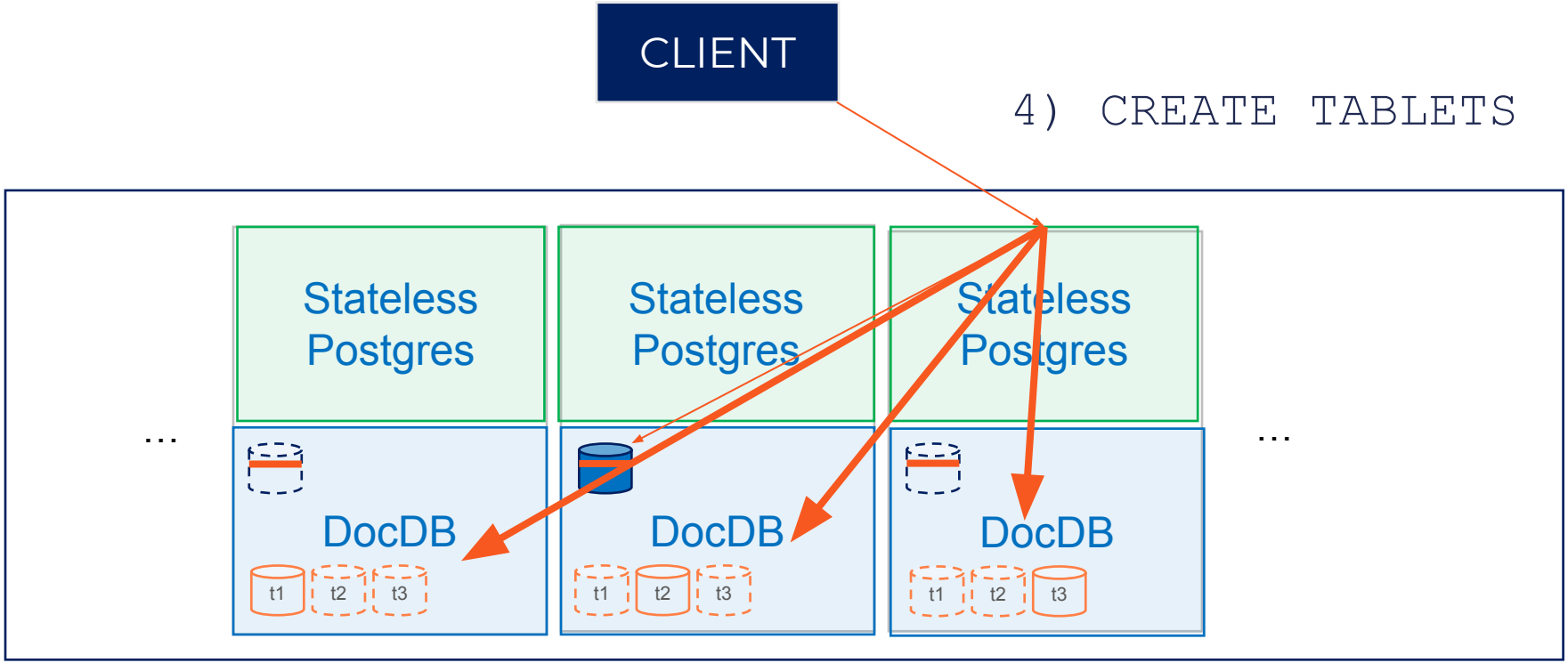
Create a Table



Create a Table



Create a Table



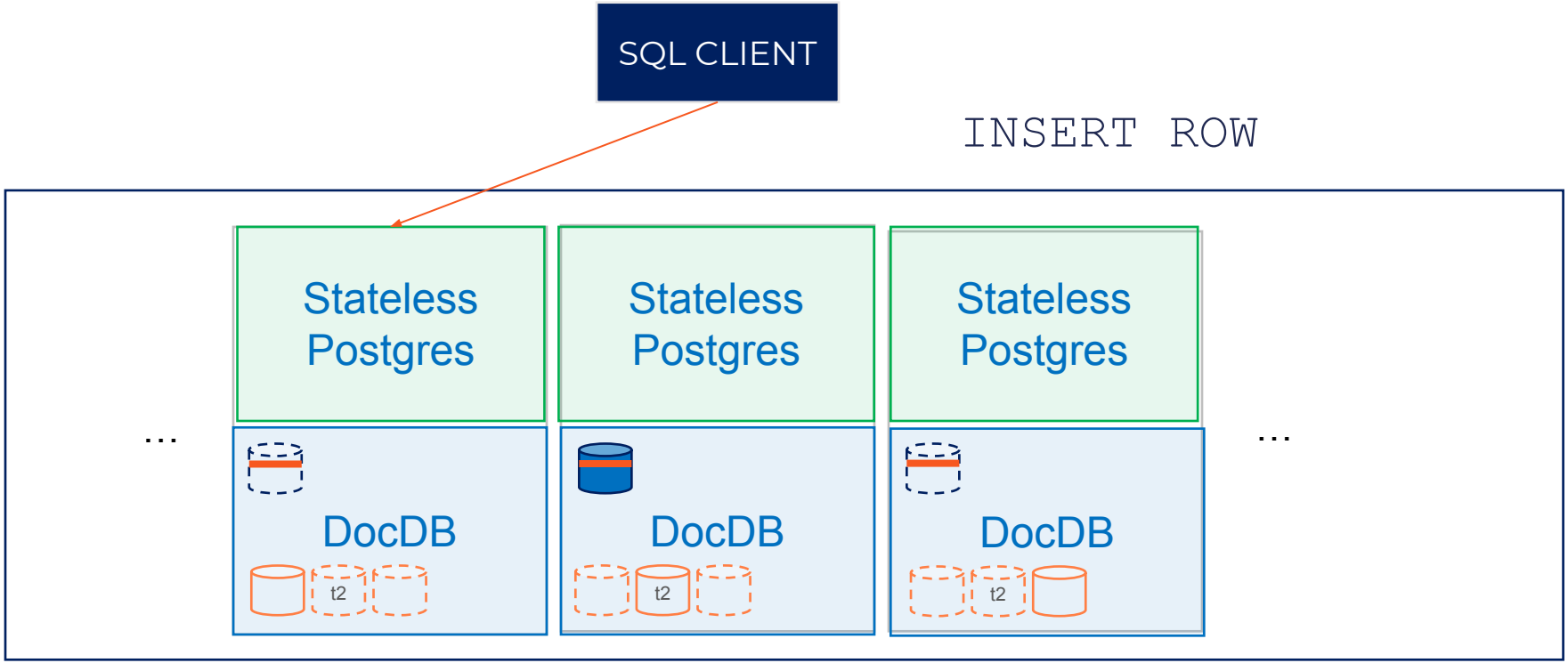
Insert Data into Tables

- **Primary keys**
  - The primary key column(s) map to a single document key
  - Each row maps to one document in DocDB
  - Tables without primary key use an internal ID (logically a row-id)

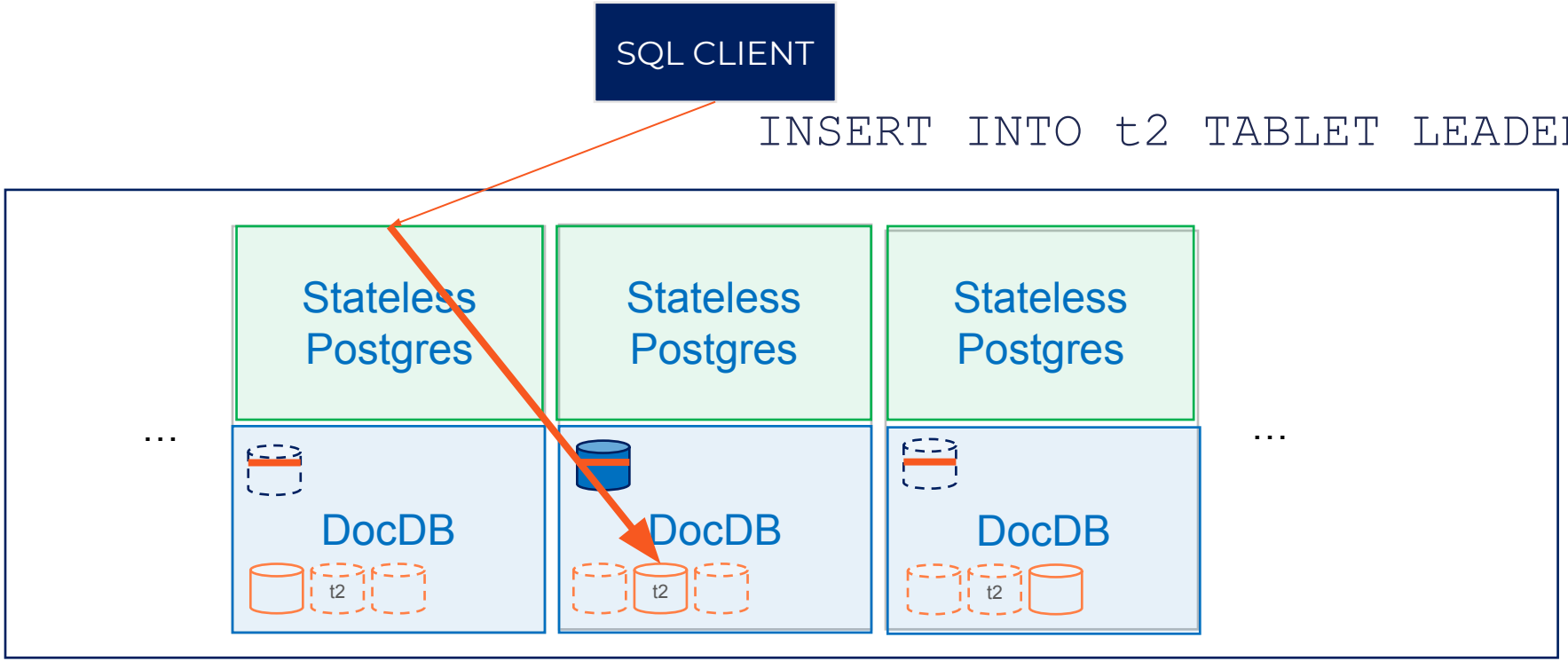
- **Secondary indexes**
  - Each index maps to a separate distributed DocDB table
  - DML implemented using **DocDB distributed transactions**
  - E.g: insert into table with one index will perform the following:

```
BEGIN DOCDB DISTRIBUTED TRANSACTION
    insert into index values (…)
    insert into table values (…)
COMMIT
```

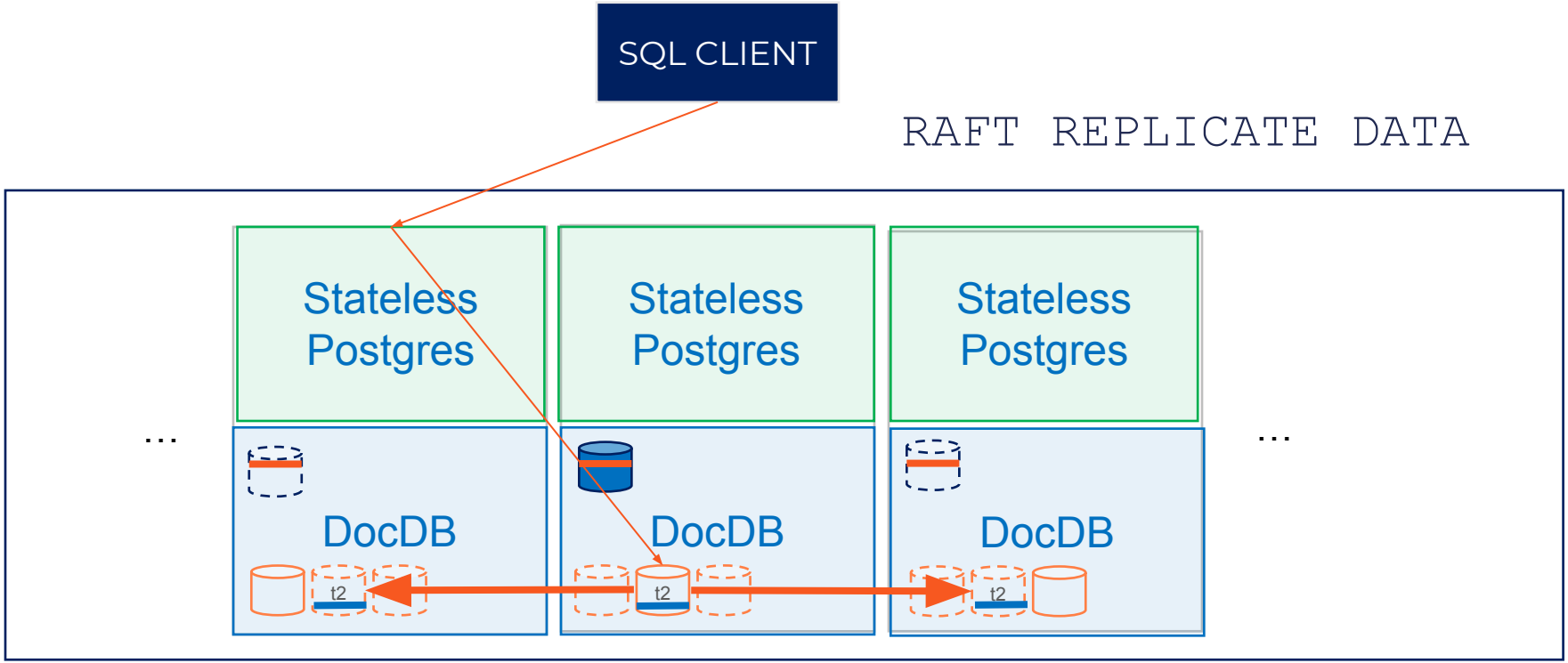
Insert Data



Insert Data



Insert Data



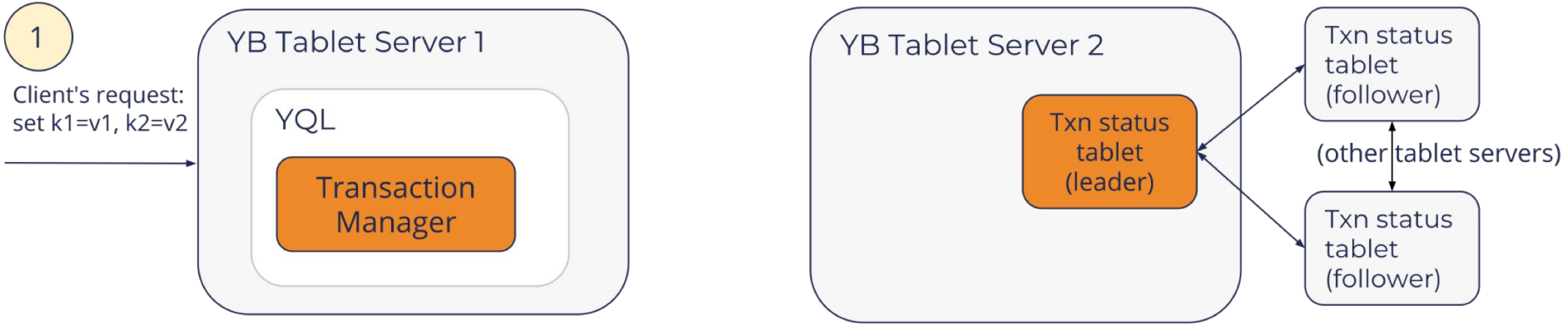
Distributed Transactions

Fully Decentralized Architecture

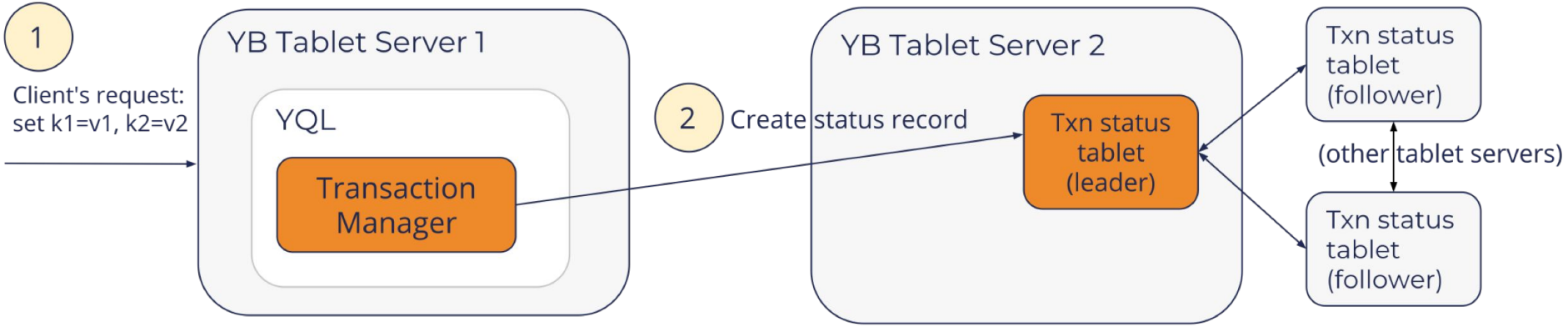
- **No single point of failure or bottleneck**
  - Any node can act as a Transaction Manager
- **Transaction status table distributed across multiple nodes**
  - Tracks state of active transactions
- **Transactions have 3 states**
  - Pending
  - Committed
  - Aborted
- **Reads served only for Committed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

Distributed Transactions - Write Path

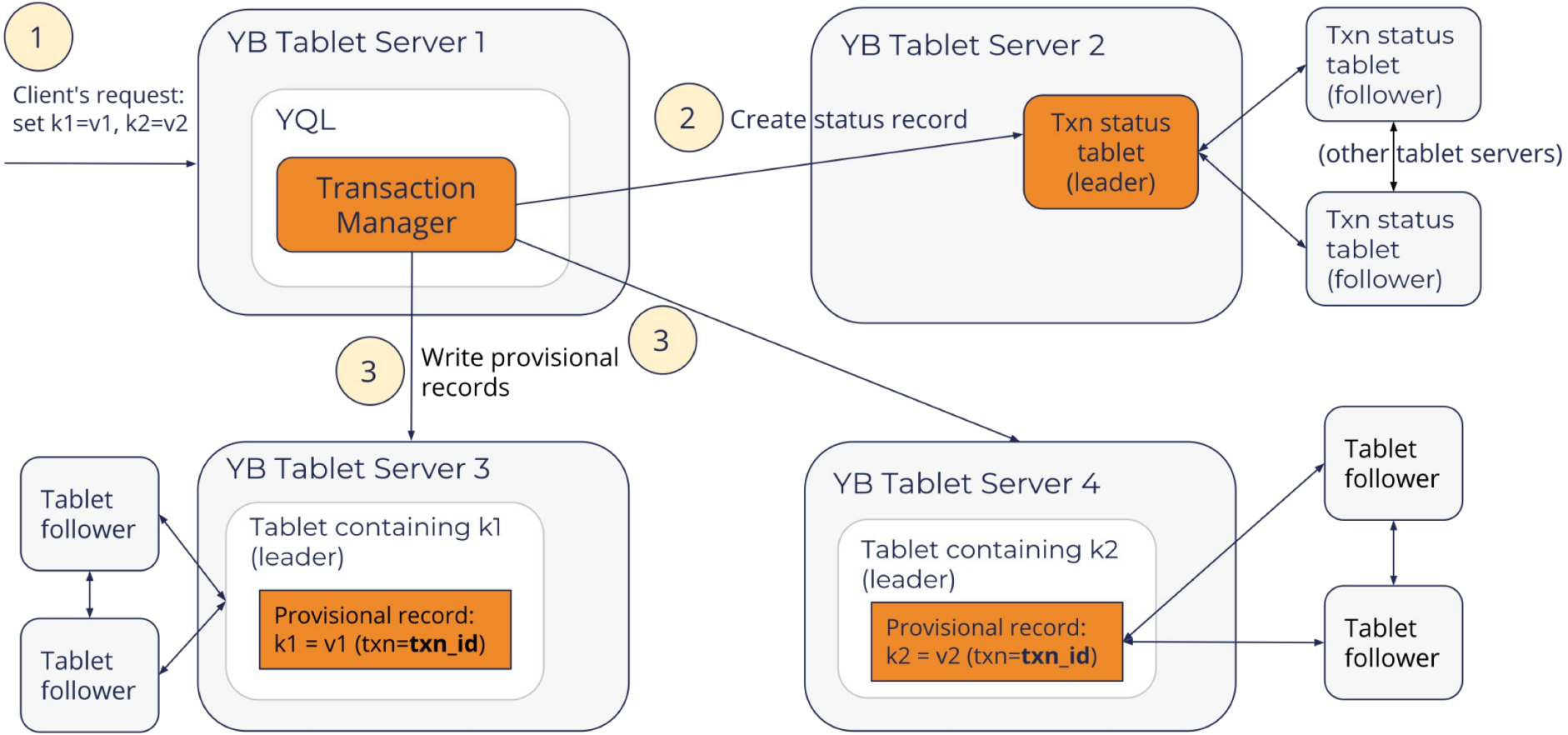
Distributed Transactions - Write Path



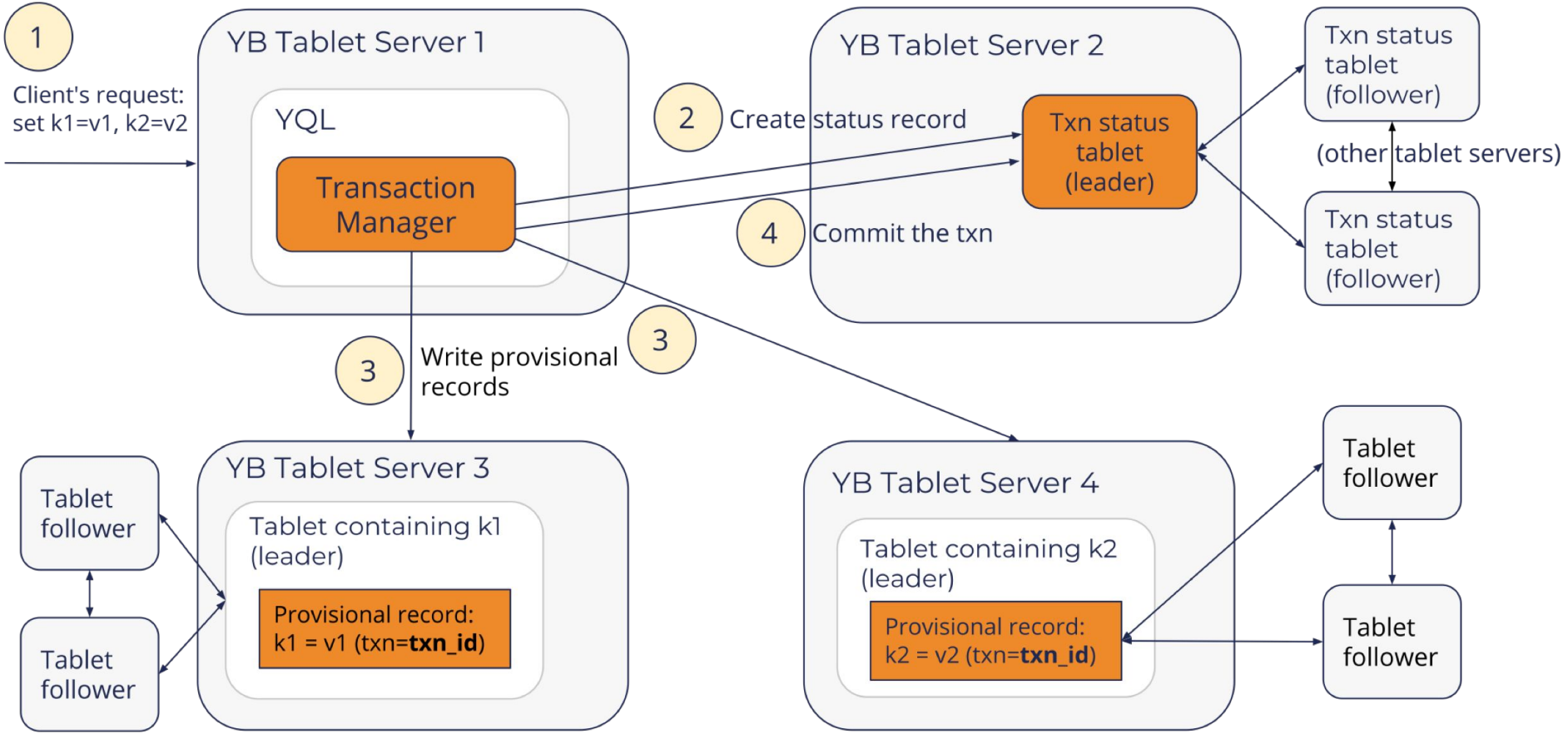
Distributed Transactions - Write Path



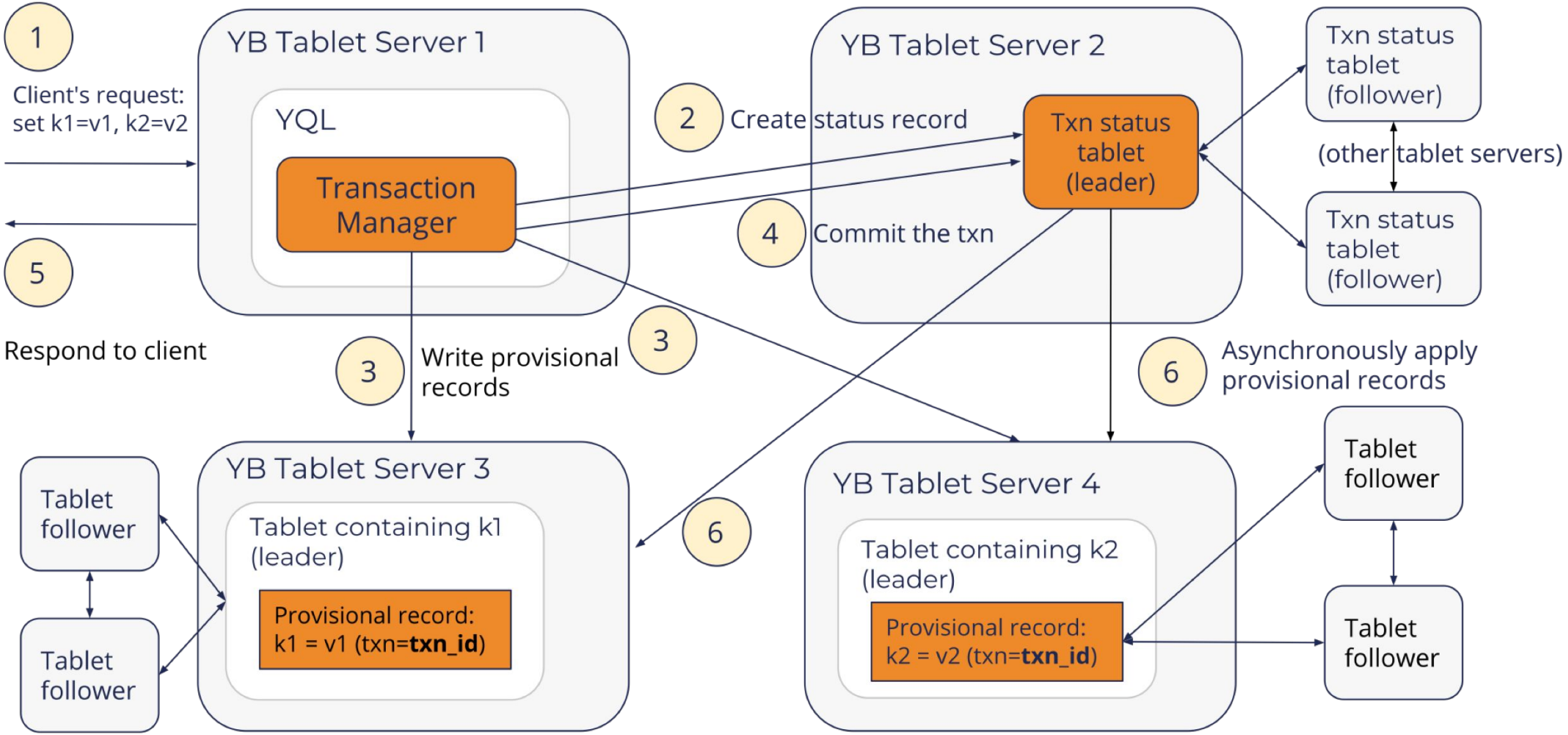
Distributed Transactions - Write Path



Distributed Transactions - Write Path



Distributed Transactions - Write Path



Isolation Levels

- **Serializable Isolation**
  - Read-write conflicts get auto-detected
  - Both reads and writes in read-write txns need provisional records
  - Maps to SERIALIZABLE in PostgreSQL

- **Snapshot Isolation**
  - 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

- **Read-only Transactions**
  - Lock free

Summary

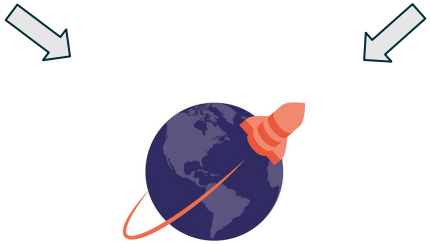
Most Advanced Open Source Distributed SQL

Query Layer

World's Most Advanced
Open Source SQL Engine

Storage Layer

World's Most Advanced
Distributed OLTP Architecture



yugabyteDB

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?

Download
download.yugabyte.com

Join Slack Discussions
yugabyte.com/slack

Star on GitHub
github.com/YugaByte/yugabyte-db