yugabyteDB

YugabyteDB: a distributed PostgreSQL database

Bryn Llewellyn

Developer Advocate, Yugabyte

Who am I?

~

Who do I think you are?

COMMUNITY NEWS

MAY 9, 2019 BY BRYN LLEWELLYN

Why I Moved from Oracle to YugaByte

I'm thrilled at the prospect of what lies ahead of me in my new job at YugaByte. I've just started in the role of Developer Advocate for YugaByte DB. This is an open source, cloud native,

Bryn Llewellyn
Developer Advocate, Yugabyte

- **You know PostgreSQL very well**
- **Not a week goes by without you typing SQL at the *psql* prompt**
- **I hope that you know PL/pgSQL and use stored procedures**
- **You don't need me to tell you about the reasons to use SQL**
- **You don't mind that Codd and Date laid the foundations as long ago as the nineteen-sixties**

History recap:

In pursuit of scalability

- **Monolithic SQL databases: the only survivor of the pre-SQL era**
- **Sharding in application code among many monolithic SQL databases**
- **NoSQL: in with “shared nothing”; out with SQL**
- **Google develops Spanner for internal use: “shared nothing” *and* SQL**
- **Google offers Spanner as proprietary DBaaS & publishes the algorithms**
- **Open source distributed SQL databases arrive**
- **At all stages, various hybrids are born and live on**

History recap:

In pursuit of fault tolerance / HA

- Companies had their own computers on their own premises. Weekend shutdown. Full backup. Tapes stored off site.
- Shutdowns less and less frequent. Incremental backup.
- Databases back Internet-facing apps. Primary/Standby arrives.
- NoSQL: in with “shared nothing” *and* low-level automatically replicated sharding; out with SQL deluxe.
- Distributed SQL: having your cake and eating it, especially with
  - *the* Postgres SQL processing code
  - on a Spanner-inspired storage layer

What it means to have

the Postgres SQL processing code

on a Spanner-inspired storage layer

meta-meta-meta demo: SQL Feature Depth

- **The YugabyteDB documentation has code examples that you can copy-and-paste into *psql* and our own *ysqlsh***

- **My posts on *blog.yugabyte.com/author/bryn/* have code examples that you can copy-and-paste into *psql* and *ysqlsh***

- **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 ...

Background reading — blog.yugabyte.com

- **High-level "What" and "Why"**
  - What is Distributed SQL?
  - Distributed SQL vs. NewSQL
  - Why We Built YugabyteDB by Reusing the PostgreSQL Query Layer
  - Spanning the Globe without Google Spanner
- **Distributed PostgreSQL on a Google Spanner Architecture**
  - Storage layer
  - Query layer
- **PostgreSQL compatibility**
  - Google Search: "bryn Llewellyn" site:blog.yugabyte.com
  - Eight technical SQL and PL/pISQL posts with code examples
  - Why I Moved from Oracle to YugaByte

What is YugabyteDB?
~
Why might you be interested?

YugabyteDB

Intrinsically fault tolerant

Out-of-the-box maximum availability architecture

Arbitrarily scalable

Intrinsic auto-sharding. Add nodes on demand. Low Latency Queries. Millions of IOPS in Throughput. TBs per Node.

No cloud vendor lock-in

Cloud Native. Multi-Cloud & Kubernetes Ready. 100% Open Source (Apache 2.0)

Distributed SQL

Fully PostgreSQL Compatible

And now for something completely different...

~

The substance

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

Tablets, tablet peers, nodes, and replication

- **Each SQL table is sharded into ~10 so-called *tablets***
  - Each SQL table is mapped to a DocDB table
  - DocDB manages tablets as a set of RF *tablet peer* replicas
  - Each tablet peer (for some table) is on its own node.
  - RF is the so-called replication factor.
    - Minimum useful value is **3**
    - Typical choice is 3
    - Bigger values, **5, 7**, and so on, bring more fault tolerance
    - Can survive (with no safety net) on **two** (or **one**, supporting no writes)

Tablets, tablet peers, nodes, and replication

- **Each tablet peer for a given table is on a different node**
  - One of these (by dynamic election) is the current *leader*
  - The other peers in the same tablet are currently *followers*

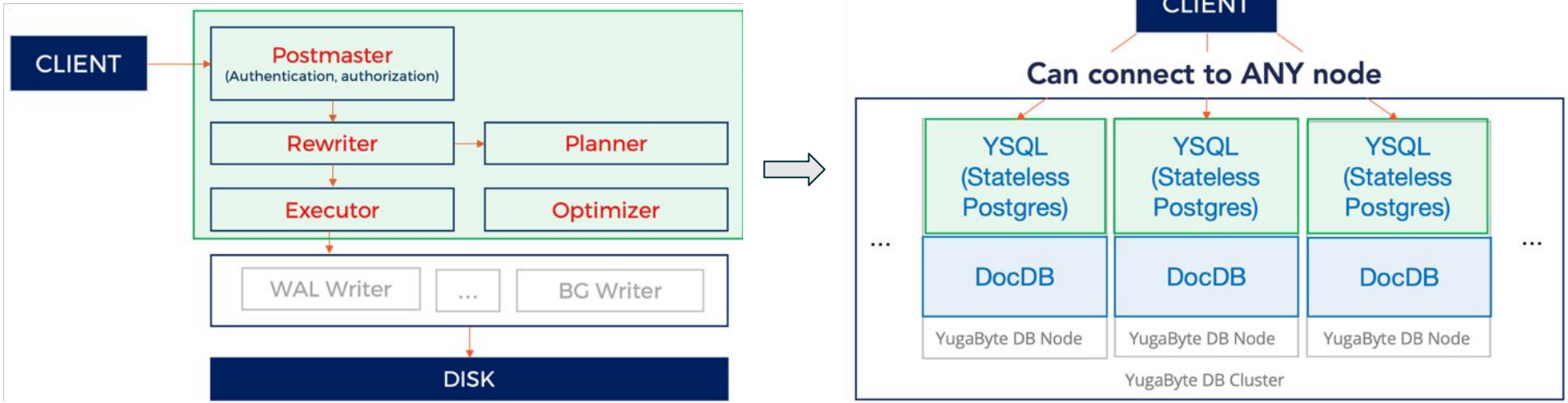
Tablets, tablet peers, nodes, and replication

- **Each node has tablet peers from many different tables**
  - You can have many more than RF nodes , and the node count needn't be odd
  - If a node "vanishes", all the tablet peers that were leaders there are then lead on surviving nodes
  - The surviving nodes, for each tablet, that used to host only followers, automatically elect one among themselves to be the new leader for that tablet
  - This is the clue to YugabyteDB's intrinsic, automatic fault tolerance

Tablets, tablet peers, nodes, and replication

- **You can add a node to the cluster, or decommission one at any time**
  - This is the clue to demand-based scalability
  - The new node automatically takes over tablet peers over a period of several minutes

PostgreSQL Transformed into Distributed SQL

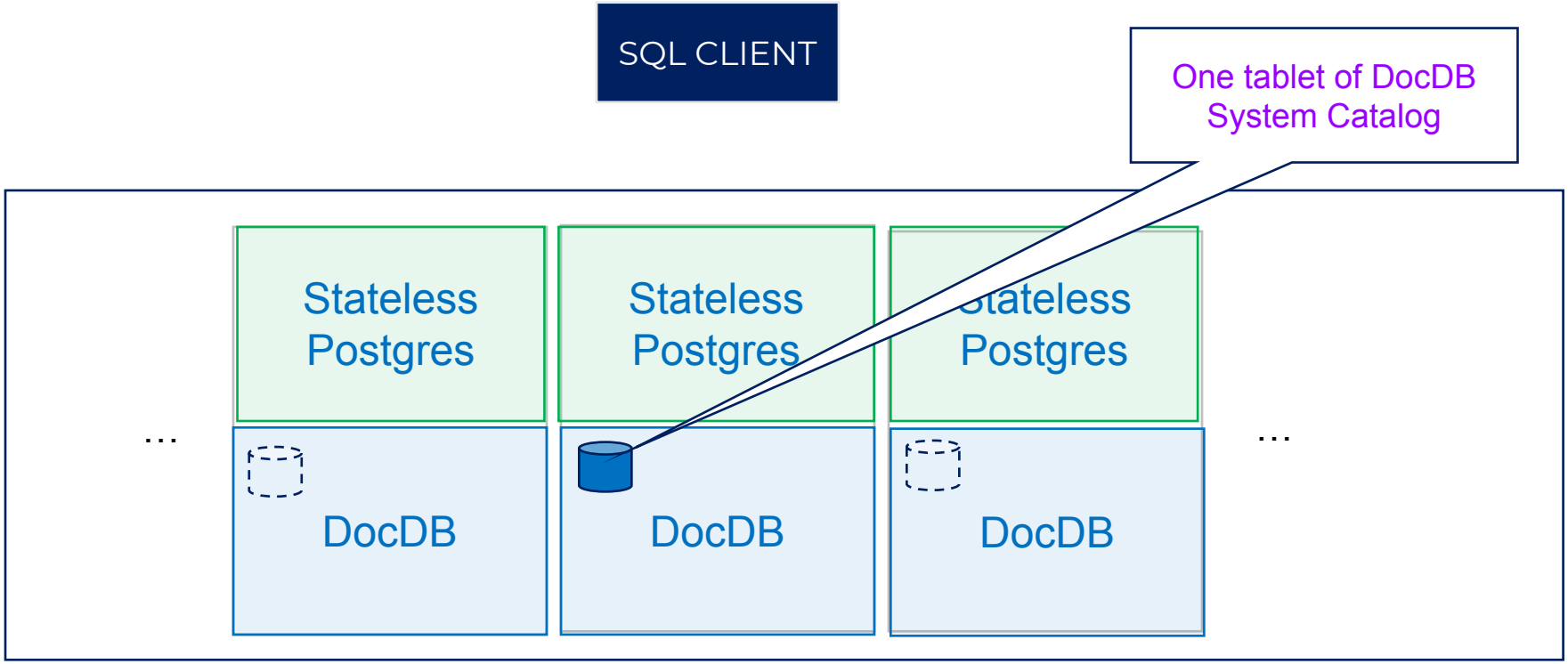


Create Table & Insert Data

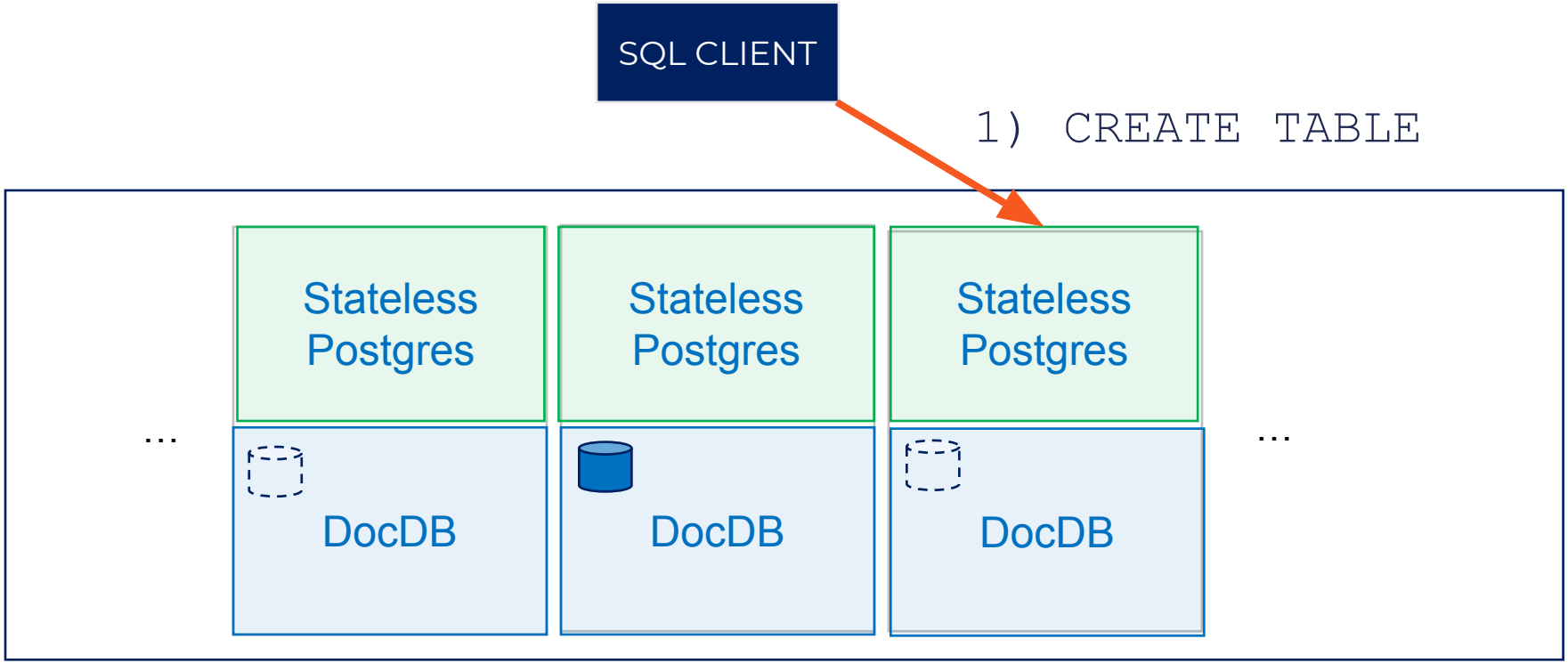
YSQL Tables

- **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

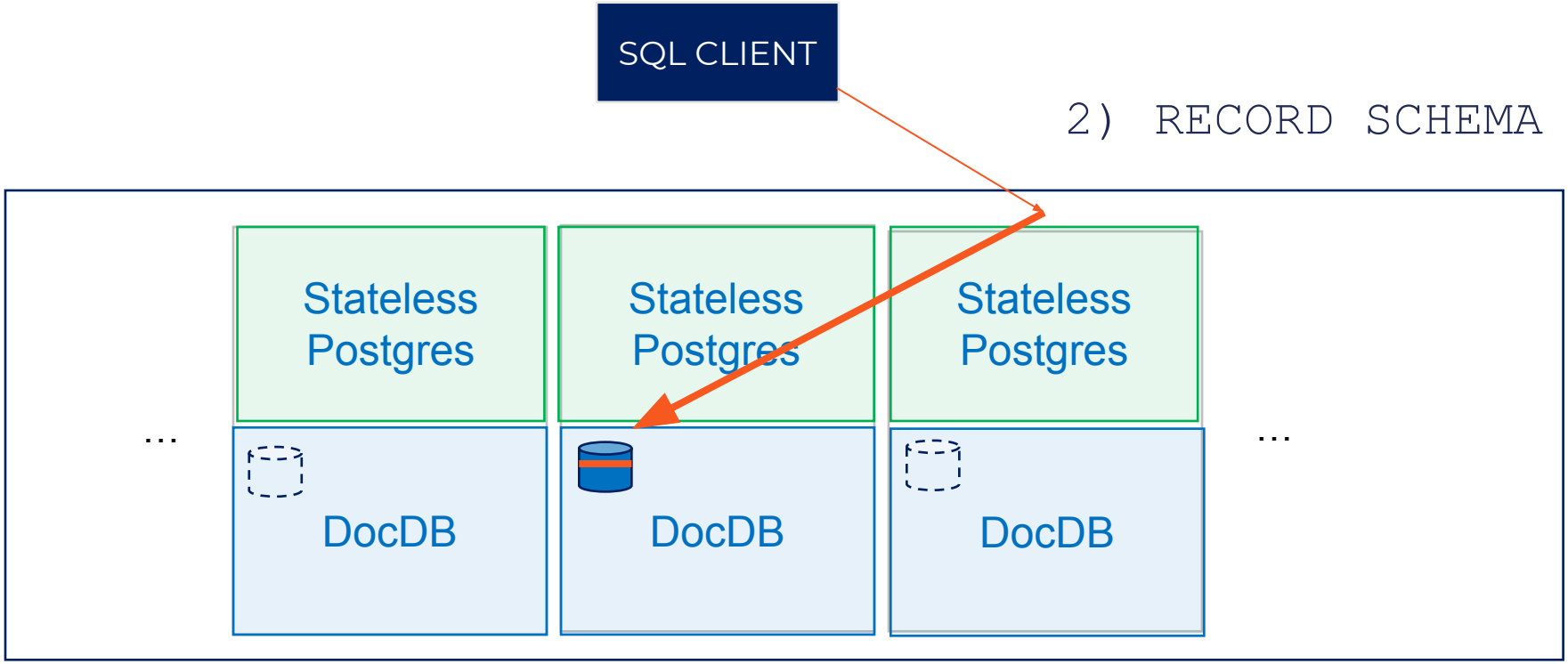
System Catalog Tables are Special Tables



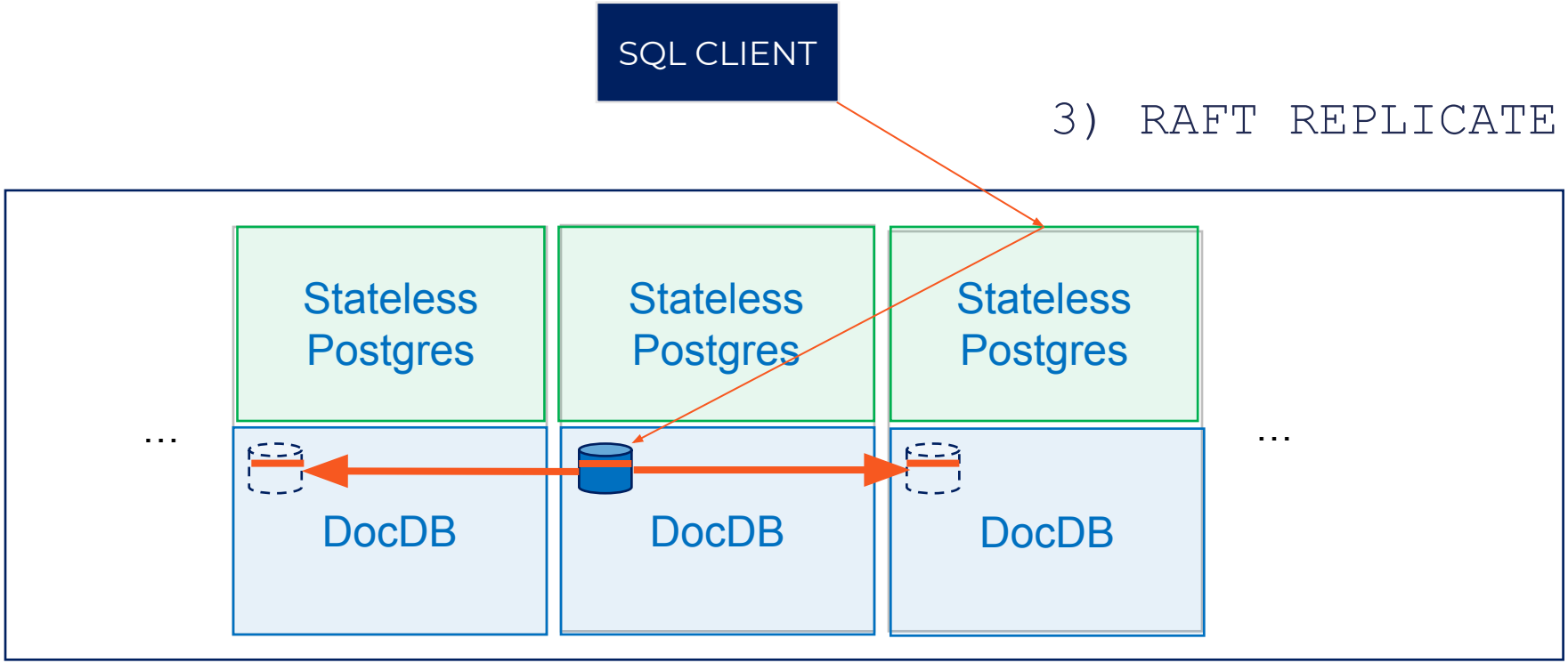
Create a Table



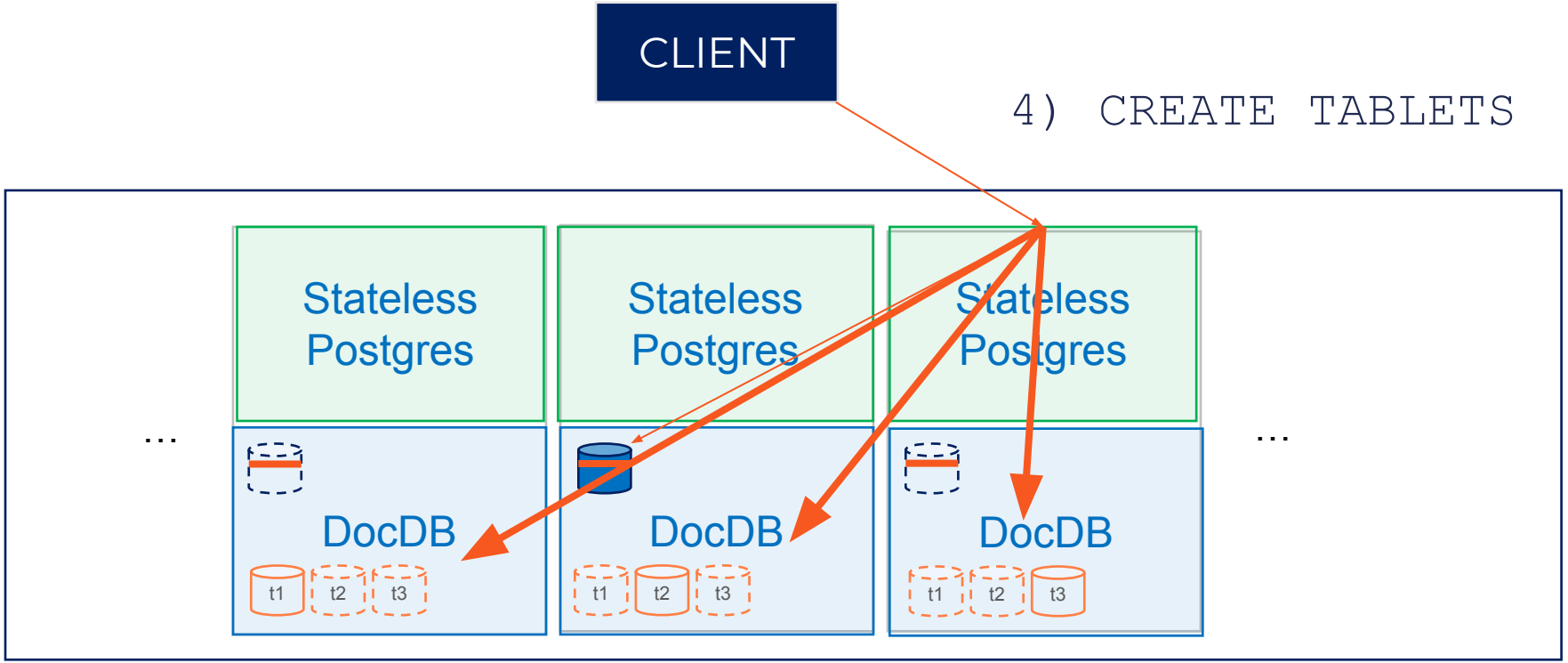
Create a Table



Create a Table



Create a Table



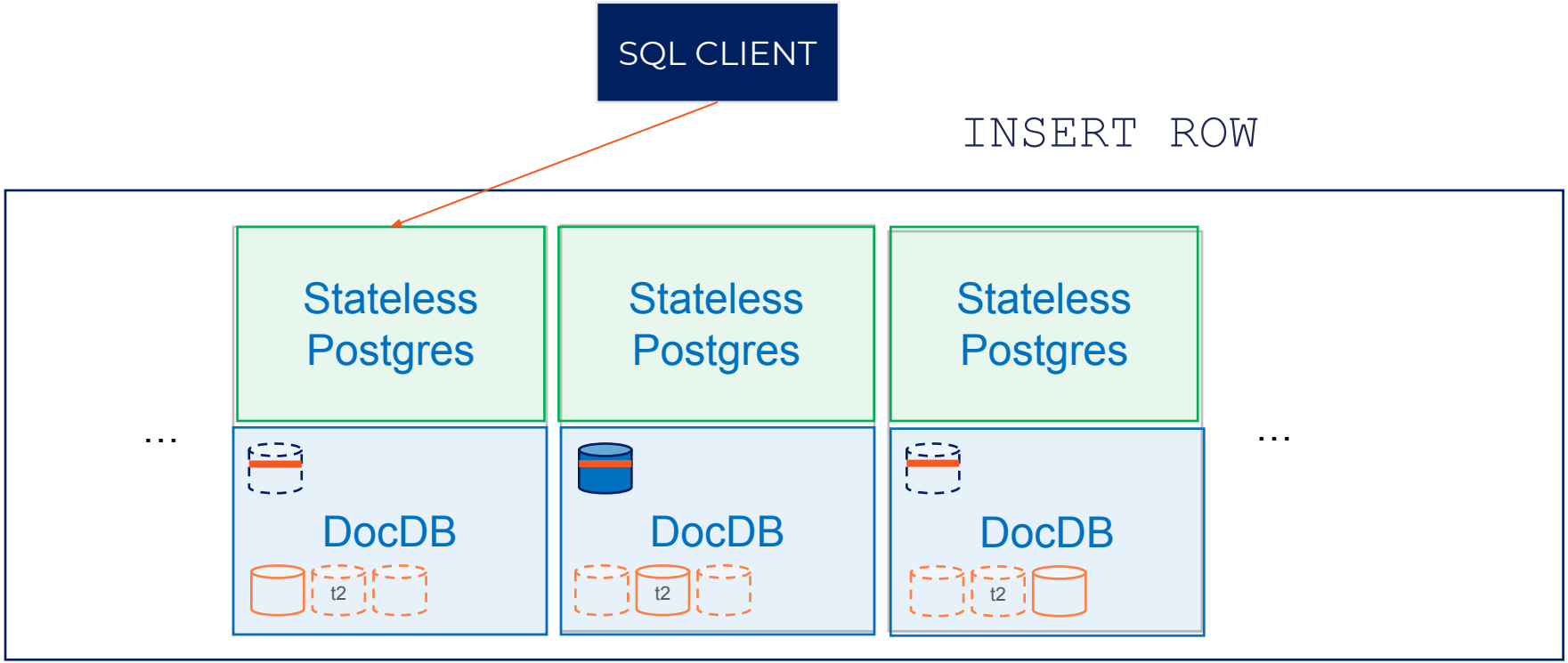
Insert Data into Tables

- **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)

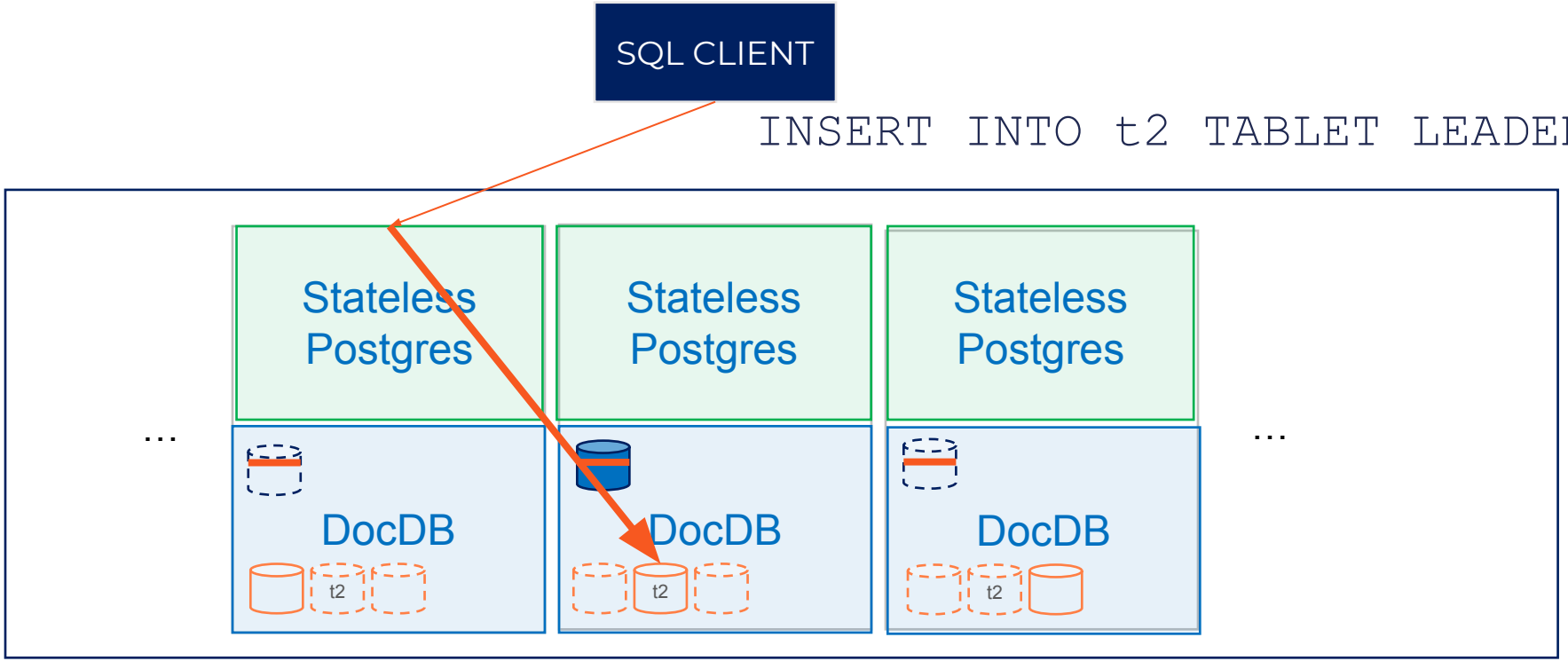
- **Secondary indexes**
  - Each index maps to a separate distributed DocDB table
  - DML implemented using **DocDB distributed transactions**
  - E.g: insert into table with one index will perform the following:

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

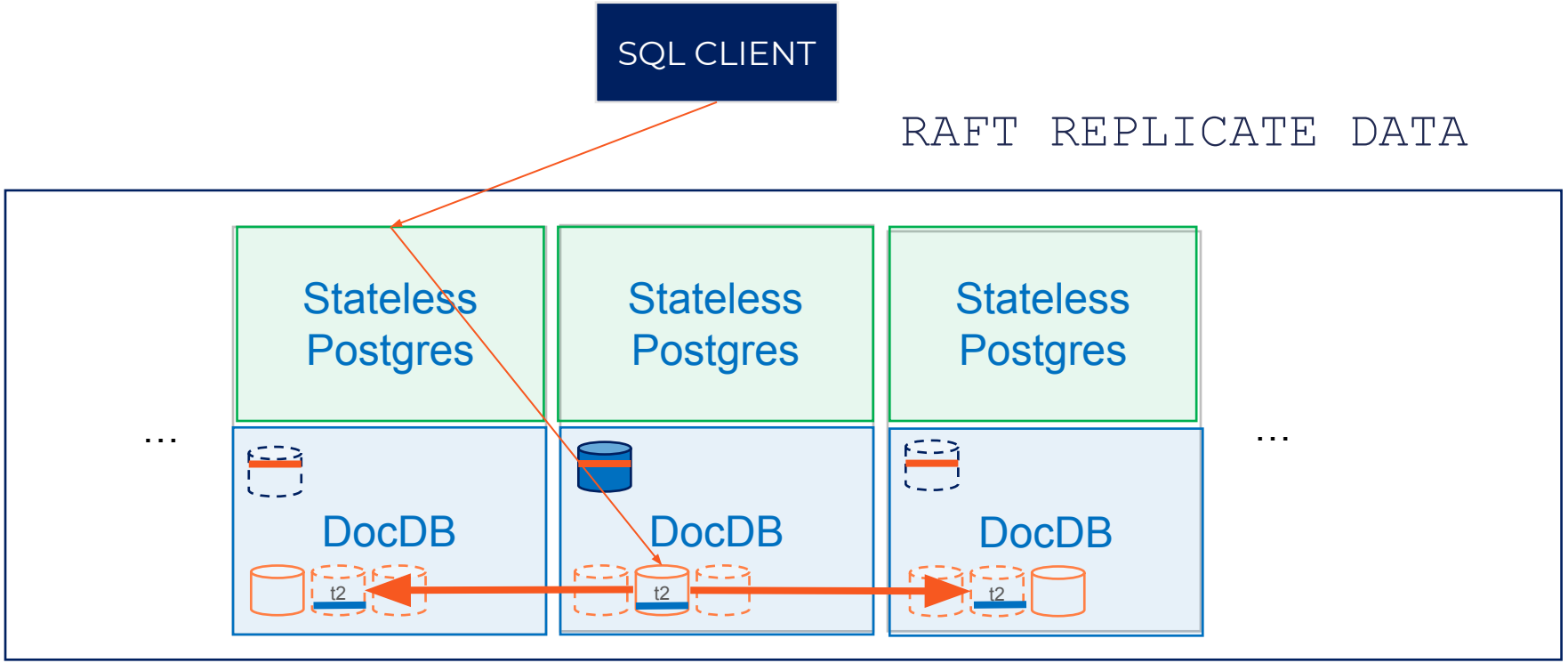
Insert Data



Insert Data



Insert Data



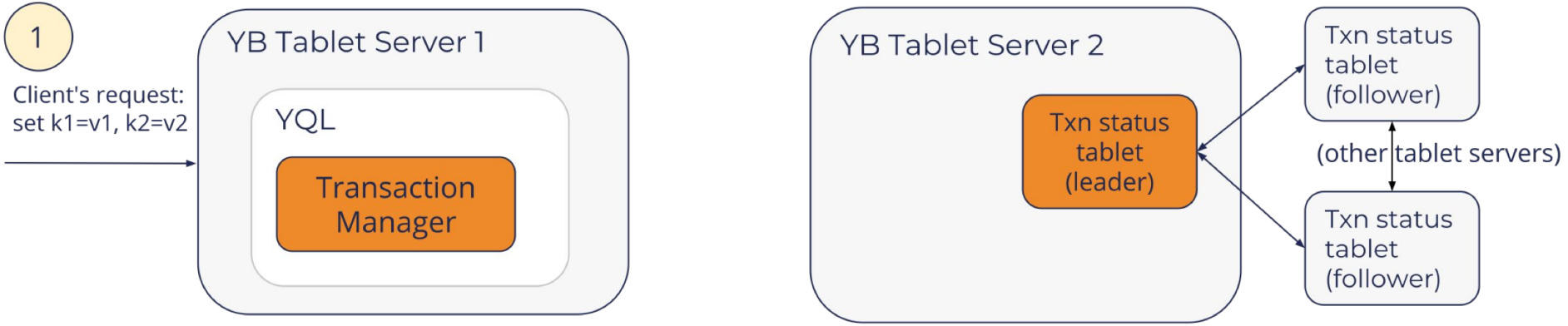
Distributed Transactions

Fully Decentralized Architecture

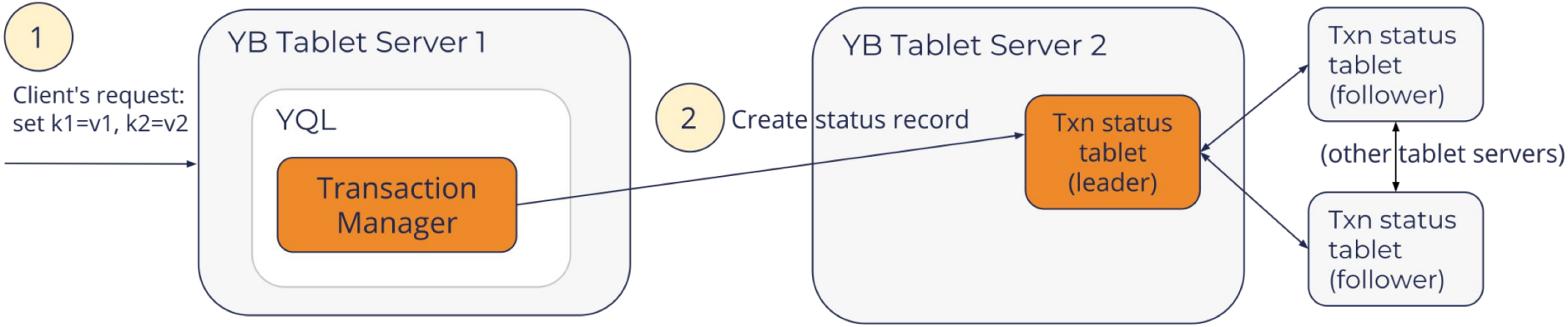
- **No single point of failure or bottleneck**
  - Any node can act as a Transaction Manager
- **Transaction status table distributed across multiple nodes**
  - Tracks state of active transactions
- **Transactions have 3 states**
  - Pending
  - Committed
  - Aborted
- **Reads served only for Committed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

Distributed Transactions - Write Path

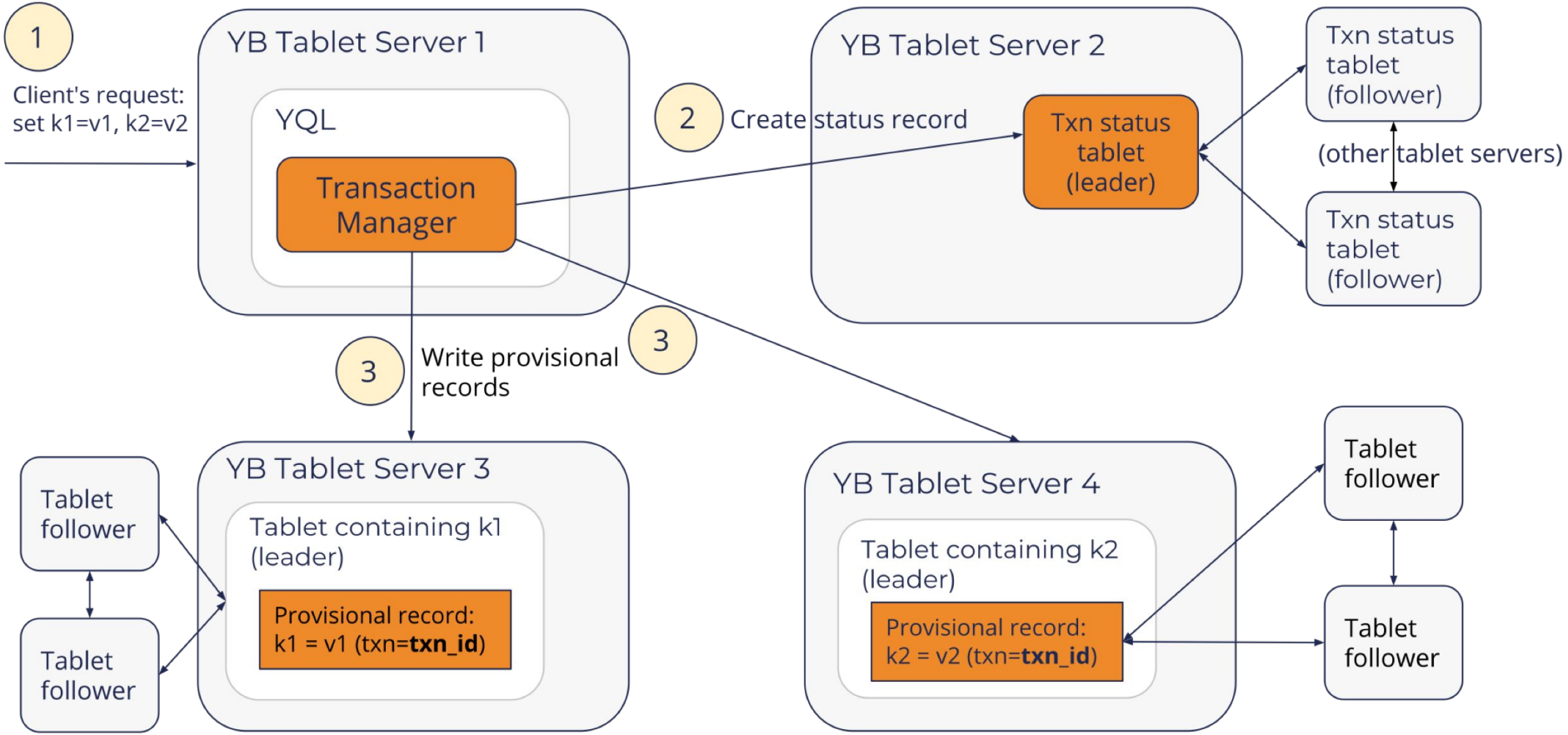
Distributed Transactions - Write Path



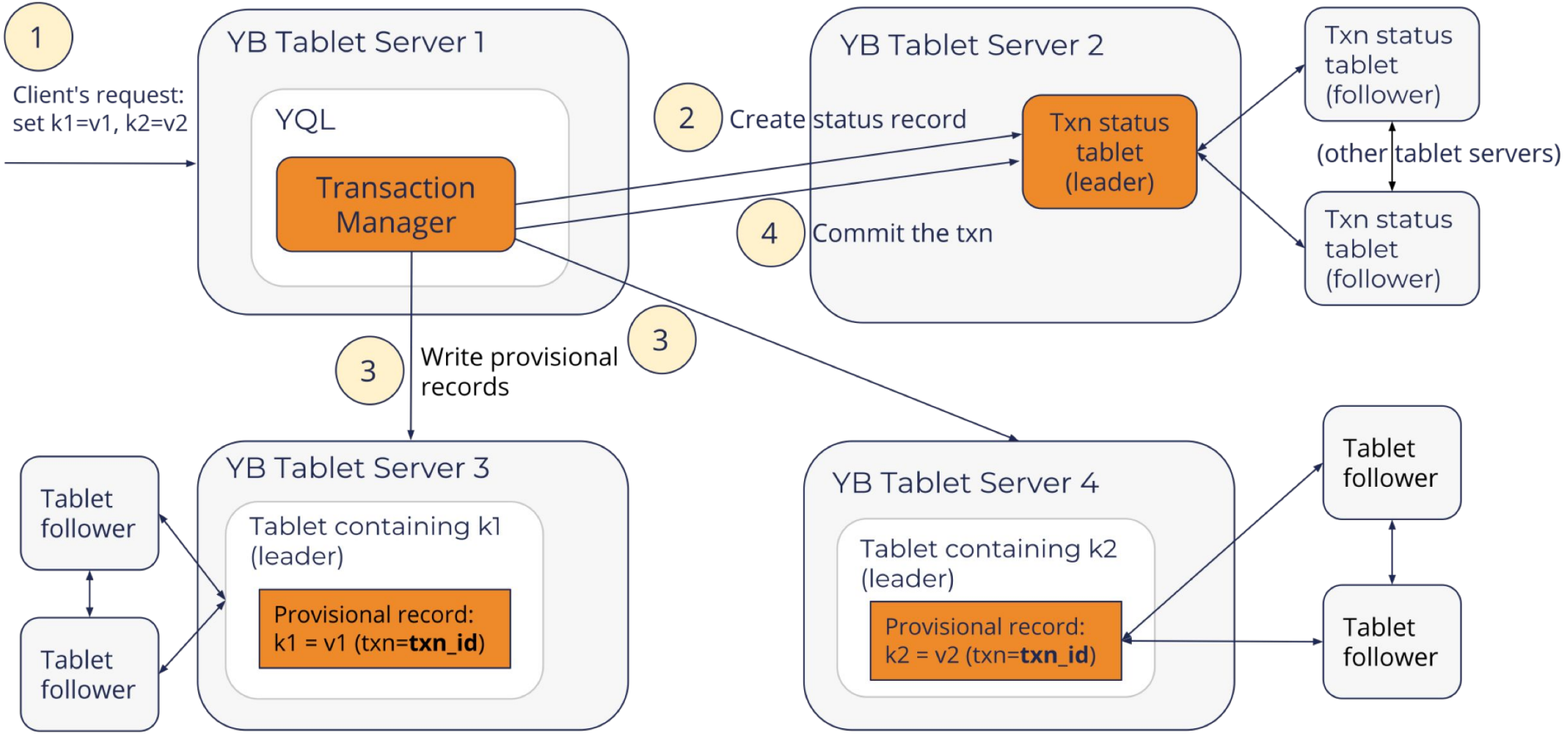
Distributed Transactions - Write Path



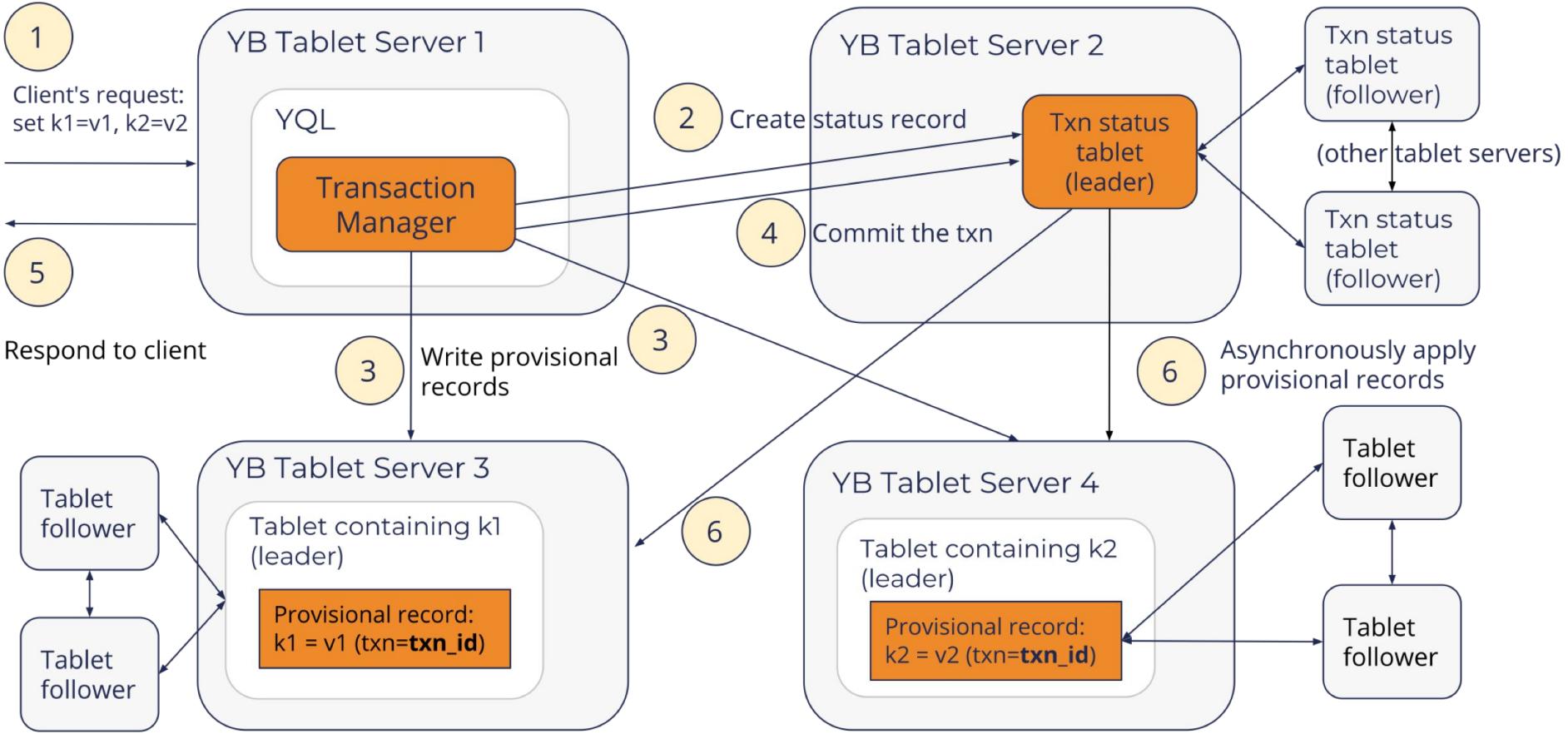
Distributed Transactions - Write Path



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

Reuse

Inspiration

yugabyteDB

Slight return — blog.yugabyte.com

- **High-level "What" and "Why"**
  - What is Distributed SQL?
  - Distributed SQL vs. NewSQL
  - Why We Built YugabyteDB by Reusing the PostgreSQL Query Layer
  - Spanning the Globe without Google Spanner
- **Distributed PostgreSQL on a Google Spanner Architecture**
  - Storage layer
  - Query layer
- **PostgreSQL compatibility**
  - Google Search: "bryn Llewellyn" site:blog.yugabyte.com
  - Eight technical SQL and PL/pISQL posts with code examples
  - Why I Moved from Oracle to YugaByte

Questions?

Download
download.yugabyte.com

Join Slack Discussions
yugabyte.com/slack

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