CockroachDB

Scalable, survivable, strongly consistent, SQL

presented by Daniel Harrison / Member of Technical Staff

Cockroach LABS

About Me

- Previously Google and Foursquare
- At Cockroach Labs for two and a half years
- We have several distributed consensus experts but I am not one of them

Agenda

- Motivation
- High-level architecture
- Some CockroachDB Features
- Q & A
- Interruptions are encouraged!

Motivation

Limitations of Existing Databases

Relational

Hard to scale horizontally

- Scalability: manual sharding results in high operational complexity and application rewrites
- Replication: wasted resources (stand-by servers) or lost consistency (asynchronous replication)

OR

NoSQL

Scalability with strings attached

- Limited transactions: developer burden due to complex data modeling
- Limited indexes: lost flexibility with querying and analytics
- Eventual consistency: correctness issues and higher risk of data corruption

CockroachDB: The Best of Both Worlds

- Single binary/symmetric nodes
- Applications see one logical DB, including cross-datacenter, global
- Self-healing/self-balancing
- Scale out is as simple as adding nodes
- SQL

High-Level Architecture

Abstraction Stack

SQL

Transactional KV

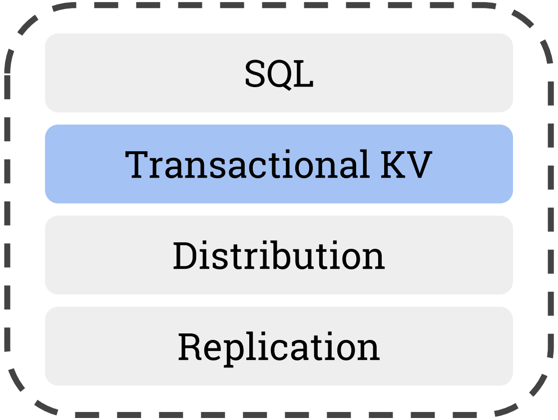
Distribution

Replication

Storage

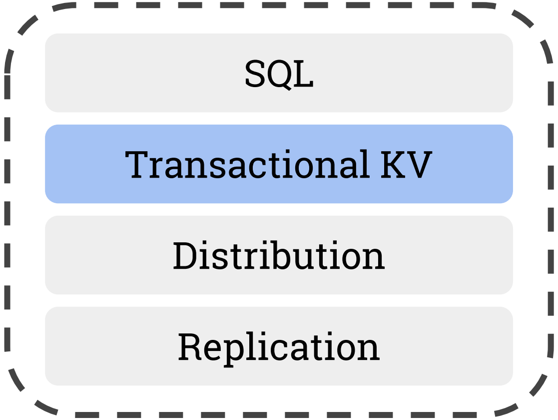
Transactional KV

- Monolithic sorted key-value map
- Automatically replicated and distributed
- Consistent
- Self-healing



Transactional KV: ACID

- **Atomicity.** All operations or no operations.
- **Consistency.** No violating constraints.
- **Isolation.** Exclusive database access.
- **Durability.** Committed data survives crashes.



SQL: Structured Data Model

Inventory

- Tables

SQL: Structured Data Model

- Tables
- Rows

Inventory

SQL: Structured Data Model

- Tables
- Rows
- Columns

Inventory

ID	Name	Quantity
1	Glove	1
2	Ball	4
3	Shirt	2
4	Shorts	12
5	Bat	0
6	Shoes	4

SQL: Structured Data Model

- Tables
- Rows
- Columns
- Indexes

Name_Idx

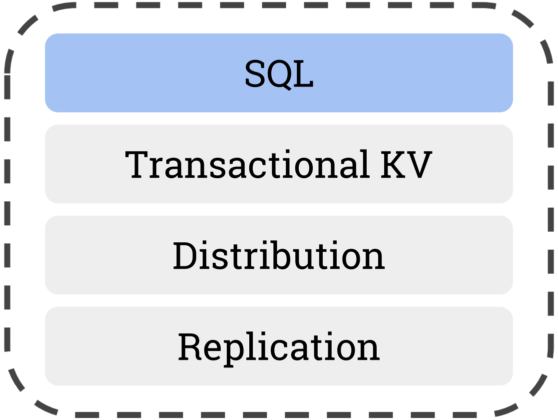
Name
Ball
Bat
Glove
Shirt
Shoes
Shorts

Inventory

ID	Name	Quantity
1	Glove	1
2	Ball	4
3	Shirt	2
4	Shorts	12
5	Bat	0
6	Shoes	4

SQL

```
CREATE TABLE inventory (
    id          INTEGER PRIMARY KEY,
    name        VARCHAR,
    quantity    INTEGER,
   INDEX name_index (name)
);
```



SQL: Key anatomy

```
INSERT INTO inventory VALUES (1, 'Apple', 12);
INSERT INTO inventory VALUES (2, 'Orange', 15);
```

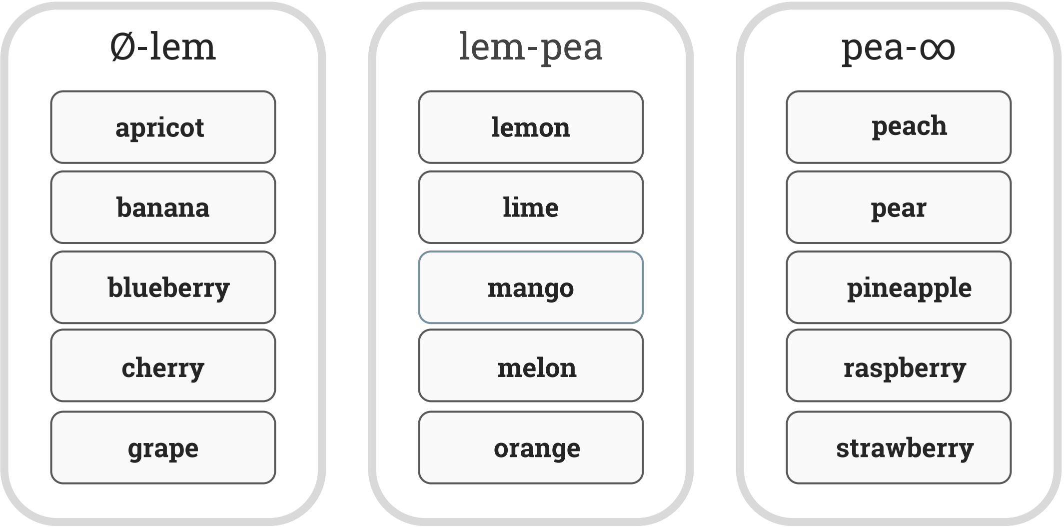
id	name	quantity
1	Apple	12
2	Orange	15

=

key /<table>/<index>/<key>/<column>	Value
/inventory/primary/1/name	Apple
/inventory/primary/1/quantity	12
/inventory/primary/2/name	Orange
/inventory/primary/2/quantity	15

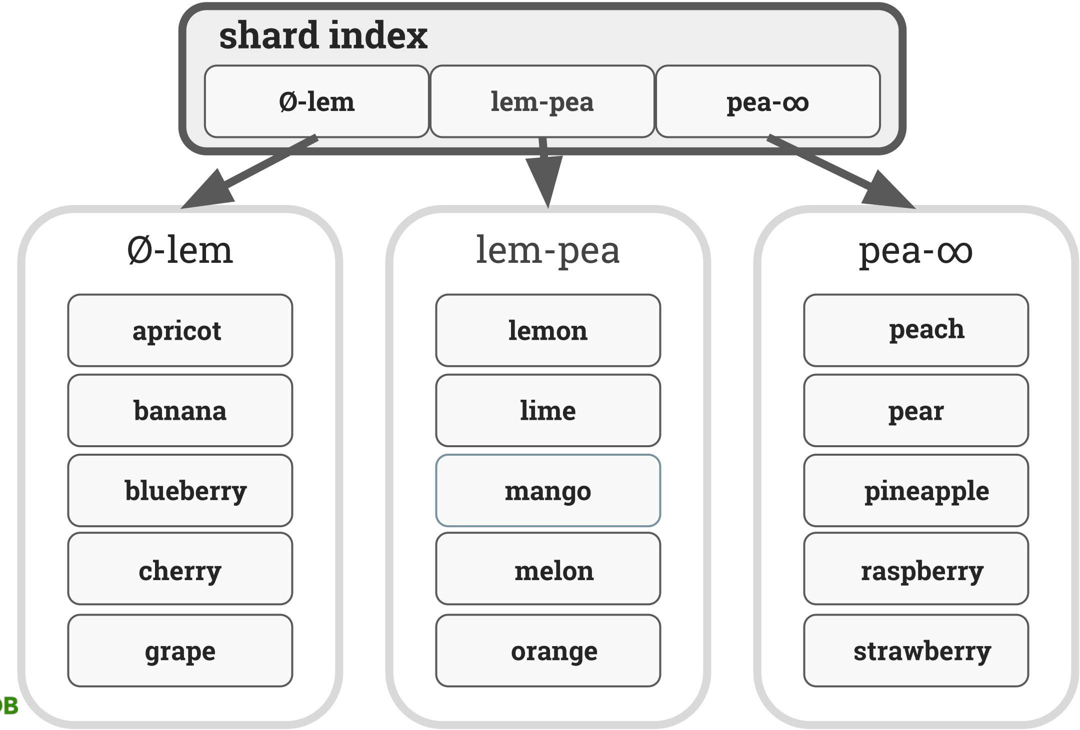
Distribution: Sharding

The data is split into ~64MB **ranges**. Each holds a contiguous range of the key space.



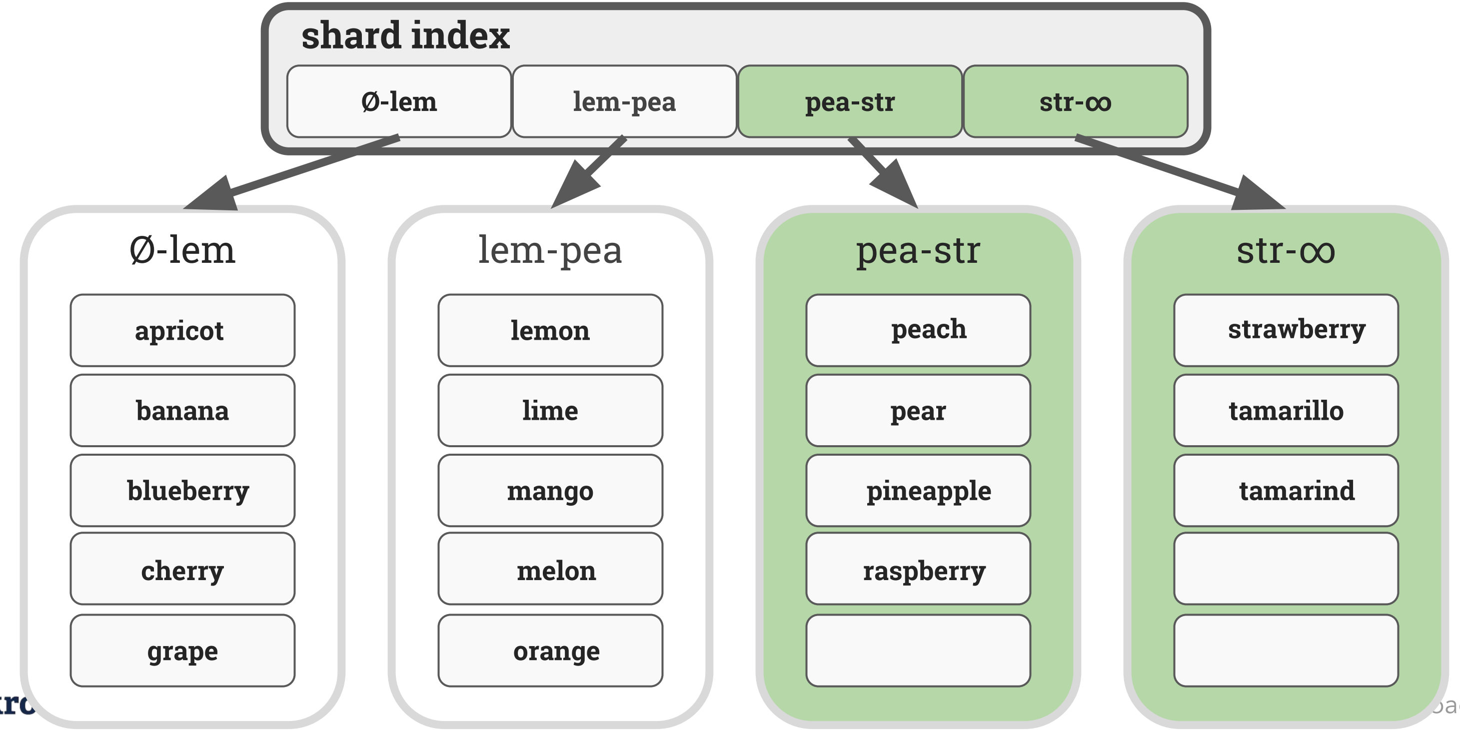
Distribution: Index

An index maps from key to range ID



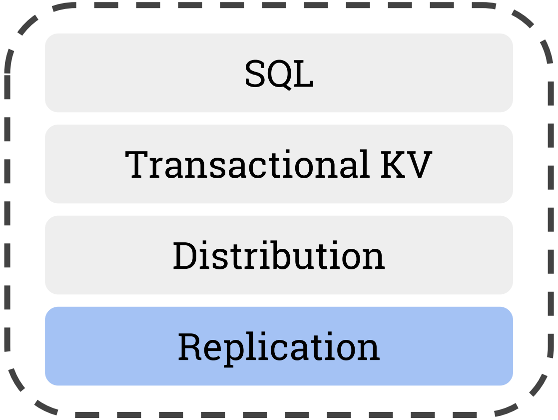
Distribution: Split

Split when a range is too large (or too hot, or…)

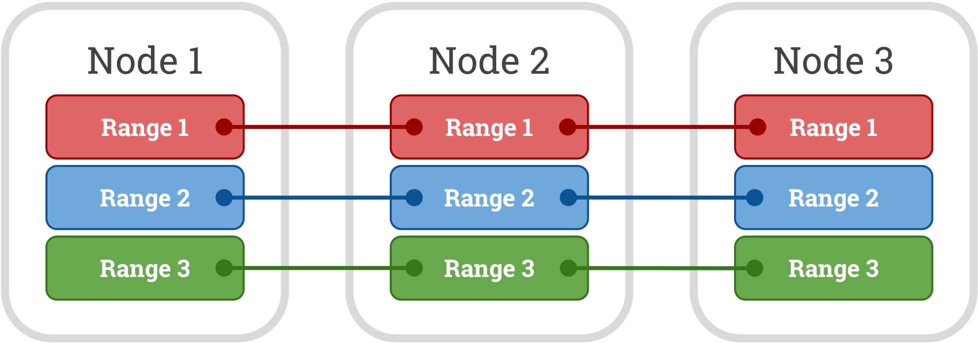


Replication: Survivability

- Each range is replicated to three or more nodes
- Consensus via Raft
- "Leaseholder" optimization to allow reads to be served without consensus
- Multi-Version Concurrency Control

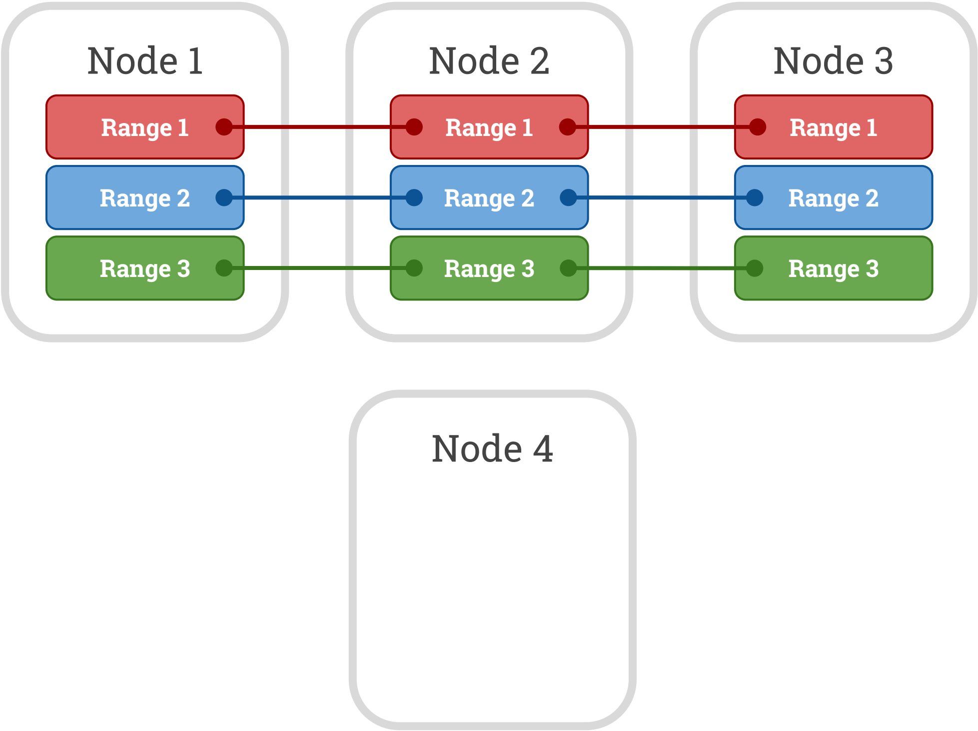


Data Distribution: Placement



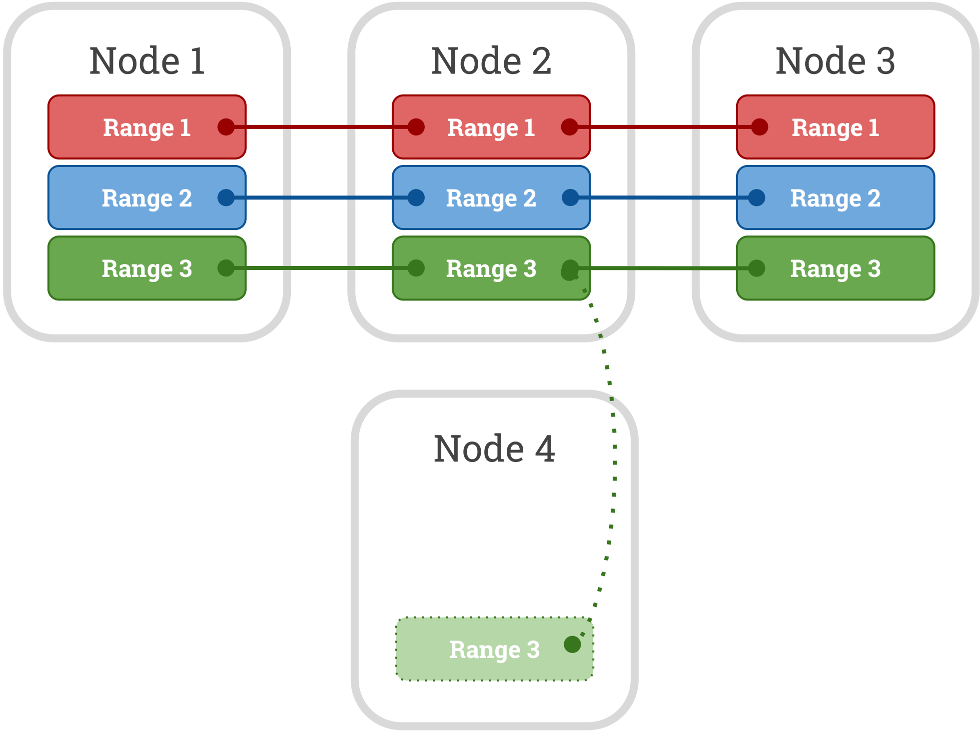
Each range is replicated to three or more nodes

Data Distribution: Rebalancing



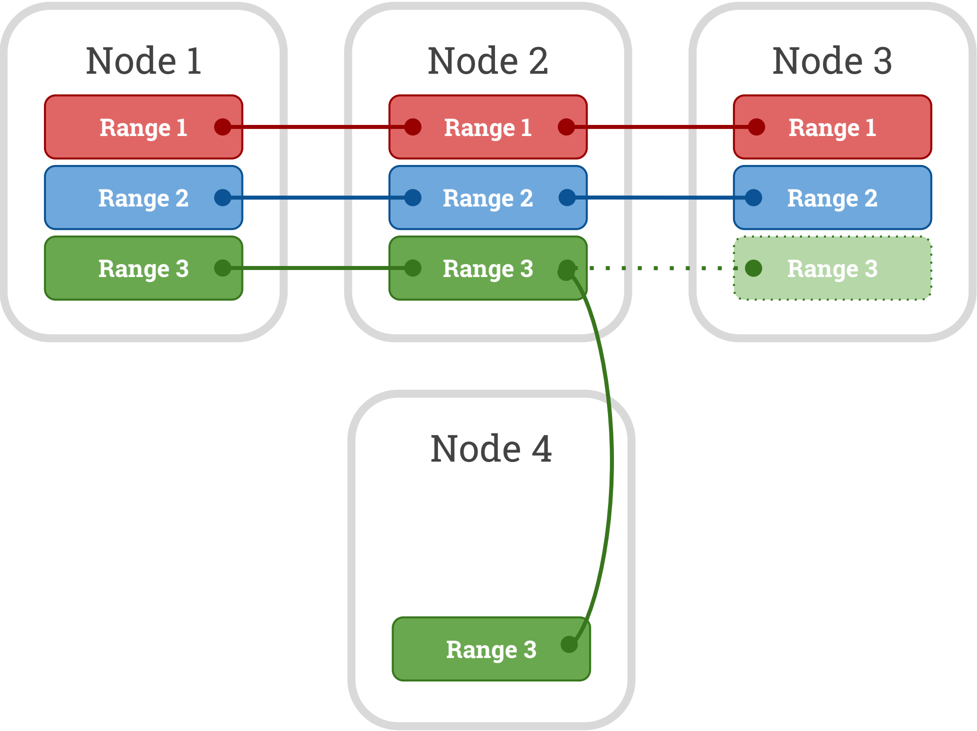
Adding a new (empty) node

Data Distribution: Rebalancing



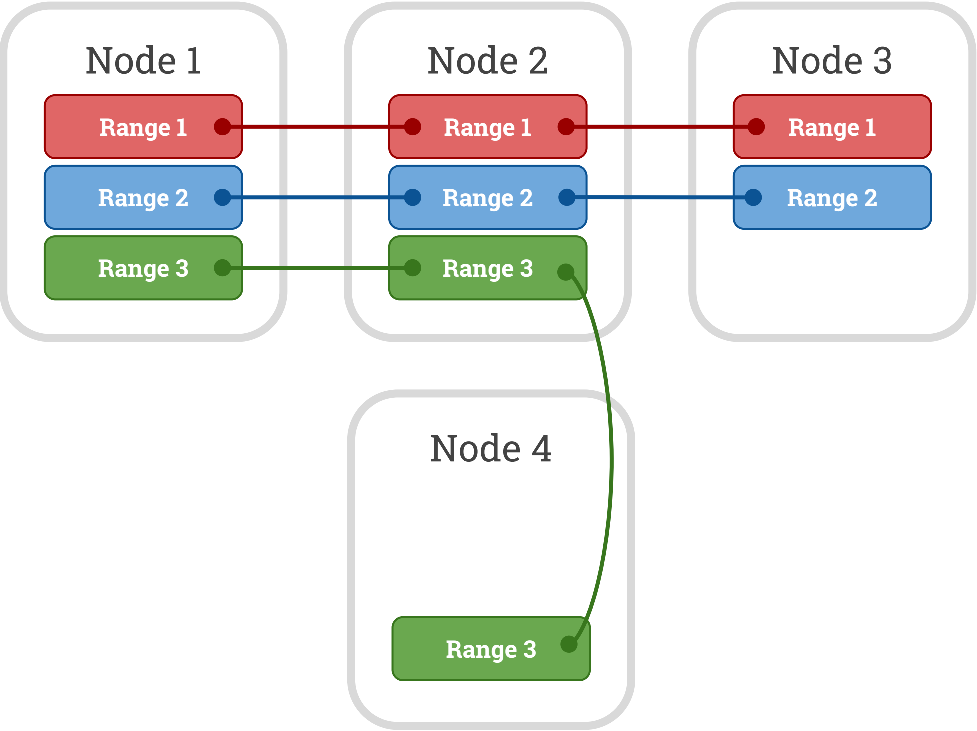
A new replica is allocated, data is copied.

Data Distribution: Rebalancing



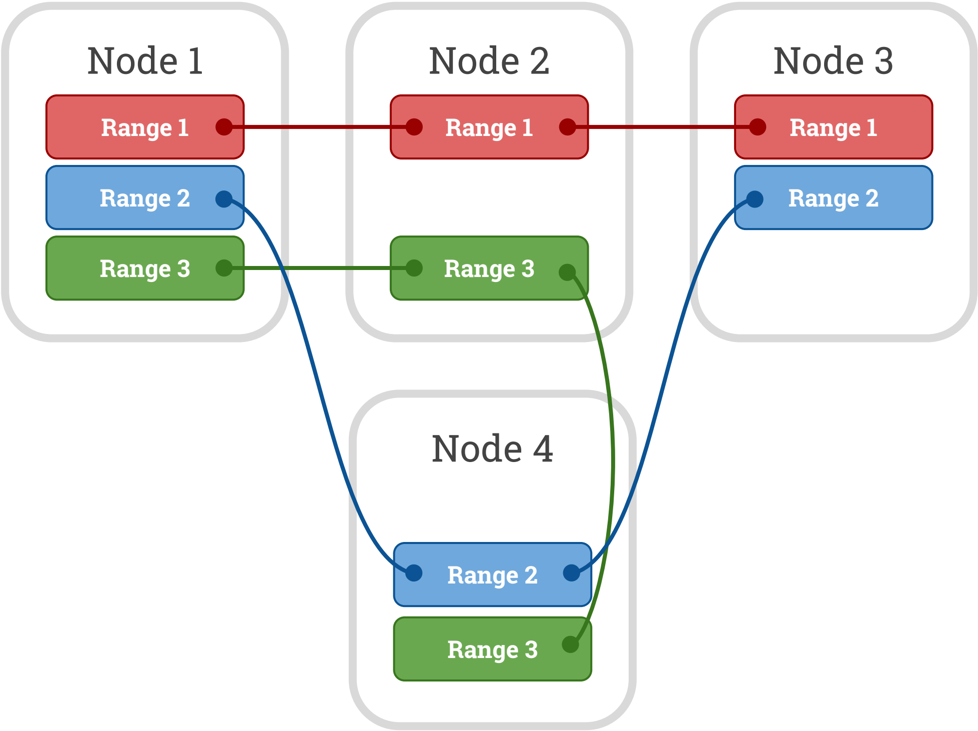
The new replica is made live, replacing another.

Data Distribution: Rebalancing



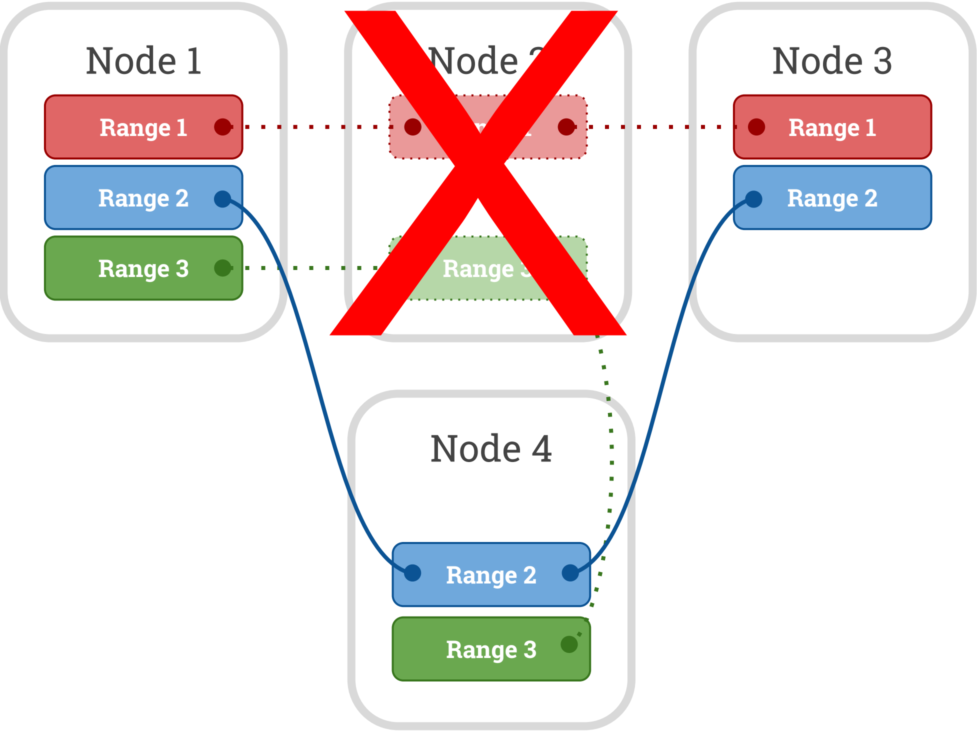
The old (inactive) replica is deleted.

Data Distribution: Rebalancing



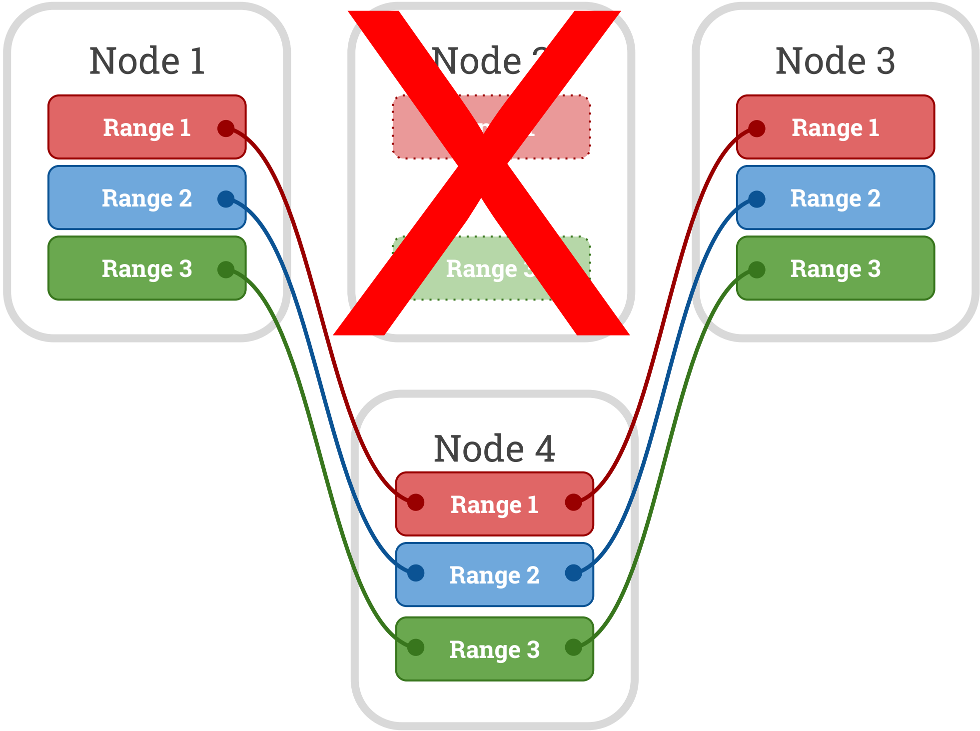
Process continues until nodes are balanced.

Data Distribution: Recovery



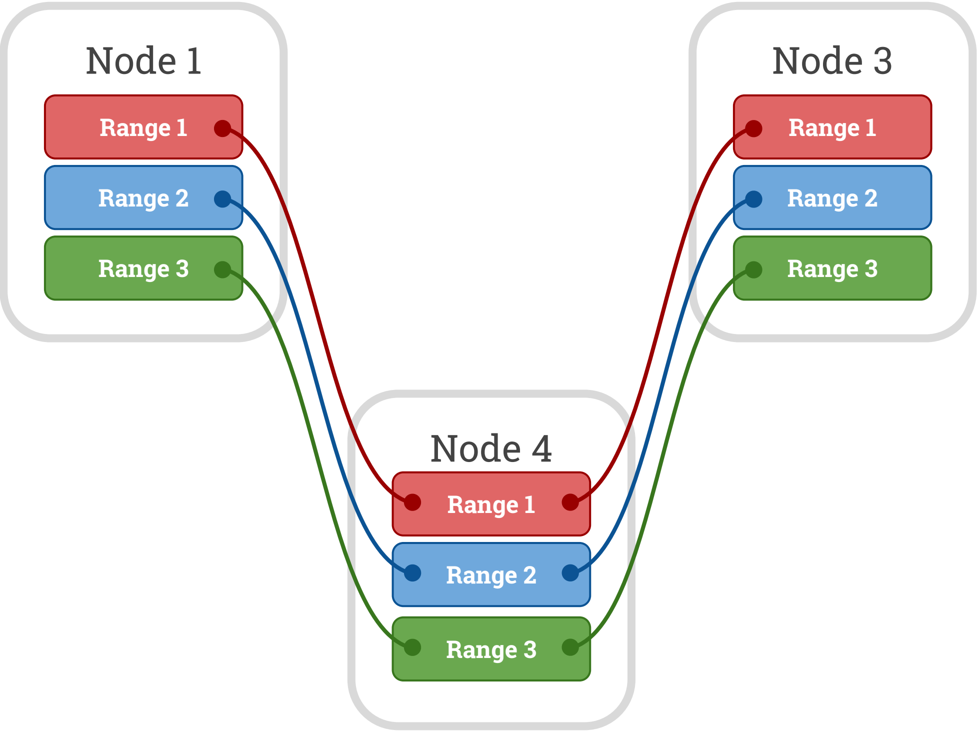
Losing a node causes recovery of its replicas.

Data Distribution: Recovery



A new replica gets created on an existing node.

Data Distribution: Recovery

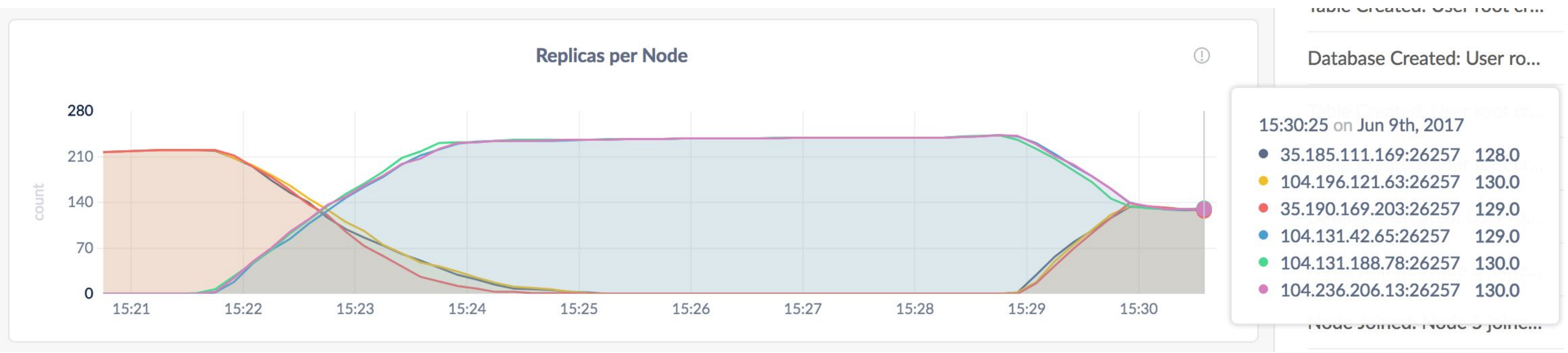


Once at full replication, the old replicas are forgotten.

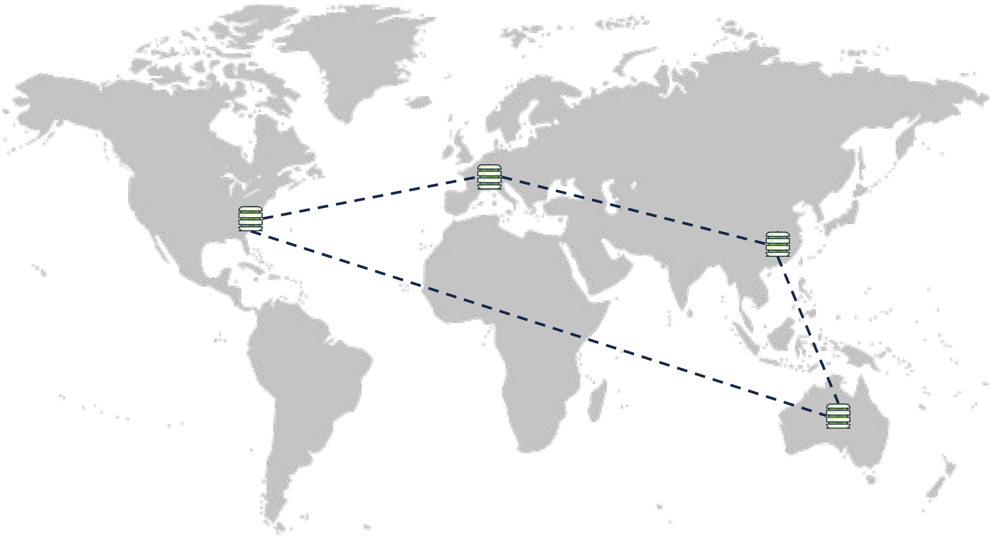
Some CockroachDB Features

Geographic Zone Configurations

- Control where your data is
- Nodes are tagged with attributes and hierarchical localities
- Rules target these
- Zero downtime data migrations



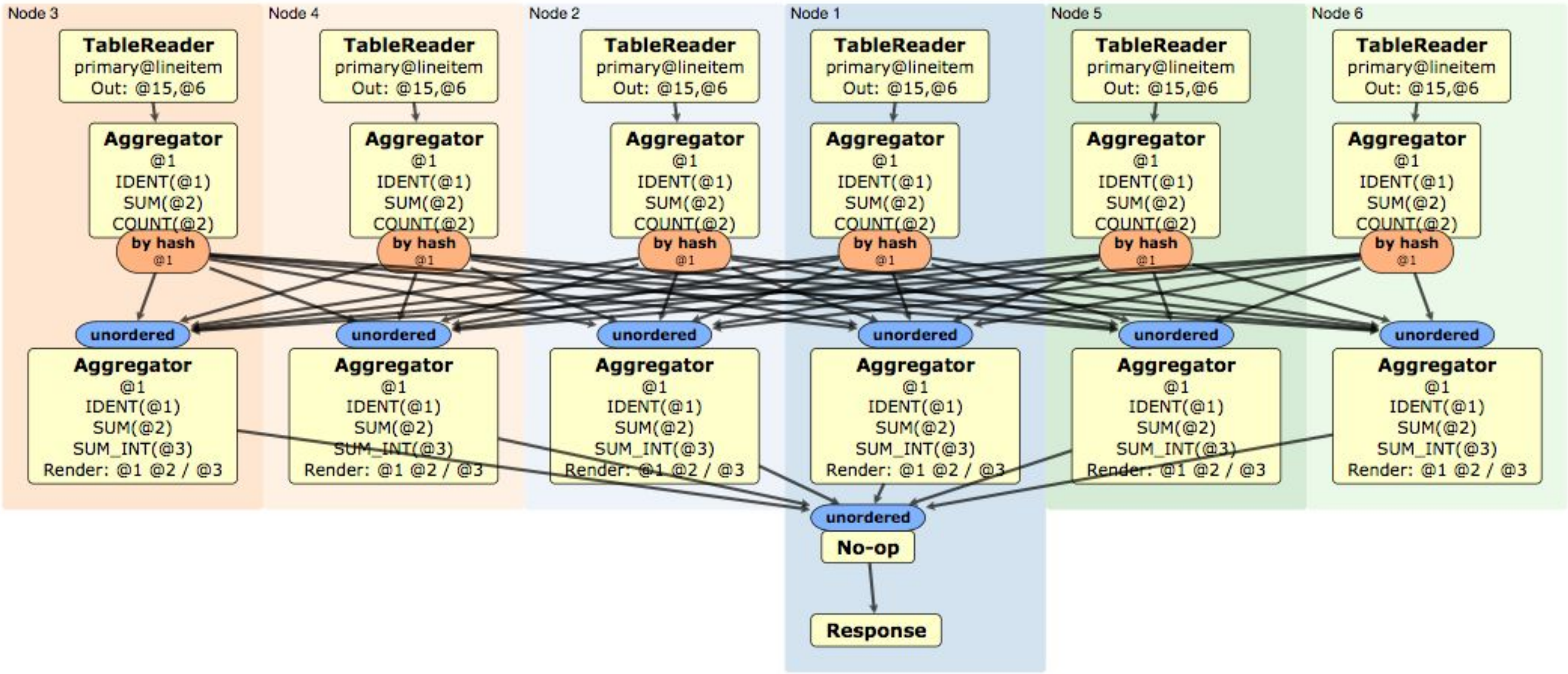
Geo-Partitioning



- Domicile data according to customer
  - Meet regulatory constraints
  - Low-latency reads / writes
- One *logical* database
  - Simplified app development

Distributed SQL

```
SELECT l_shipmode, AVG(l_extendedprice) FROM lineitem GROUP BY l_shipmode;
```



Online Schema Changes

- Based on Google's F1 Paper
- State machine, possibly with backfill
- Appears instantaneous to the client
- Zero downtime

Backup/Restore

- Distributed
- Consistent to a point in time
- Incremental

Other Topics

- (New) Query optimizer
- Graphical Admin UI
- Distributed Import
- (New) Change Data Capture

Questions?

jobs@cockroachlabs.com

github.com/cockroachdb

www.cockroachlabs.com

Cockroach LABS