Citus MX

Write-scalable, distributed PostgreSQL tables

Marco Slot
marco@citusdata.com

Overview

This talk:

- Why scale out?
- What is Citus?
- When to use Citus?
- Citus replication models
- Citus MX: Scaling out writes
- Next steps

PostgreSQL growing pains

At a certain scale, many things start going wrong:

1. Working set and indexes **no longer fit into memory**
2. **CPU is at 100%** for part of the day
3. Disk **latency starts to spike**
4. Replication/archival cannot keep up
5. **Table bloat** grows out of control
6. **Ingestion gets bottlenecked** on index maintenance, disk throughput
7. **Things fail**
8. …

Why scale out?

When you run into scaling problems do you:

1. Rearchitect your application every week?
2. Buy bigger hardware every week?
3. Start from scratch and use NoSQL?
4. Scale out PostgreSQL?

Scaling out allows you to make performance problems go away by simply adding more servers, so

You can focus on adding features and growing your business.

What is Citus?

Citus is an **extension** that adds **distributed tables** to PostgreSQL.

Distributed tables are transparently **sharded** across other PostgreSQL servers to **horizontally scale out** memory, storage and CPU.

Available as open source software:
https://github.com/citusdata/citus

Can get started in minutes using Citus Cloud:
https://www.citusdata.com/product/cloud

CREATE EXTENSION citus;

Add nodes:

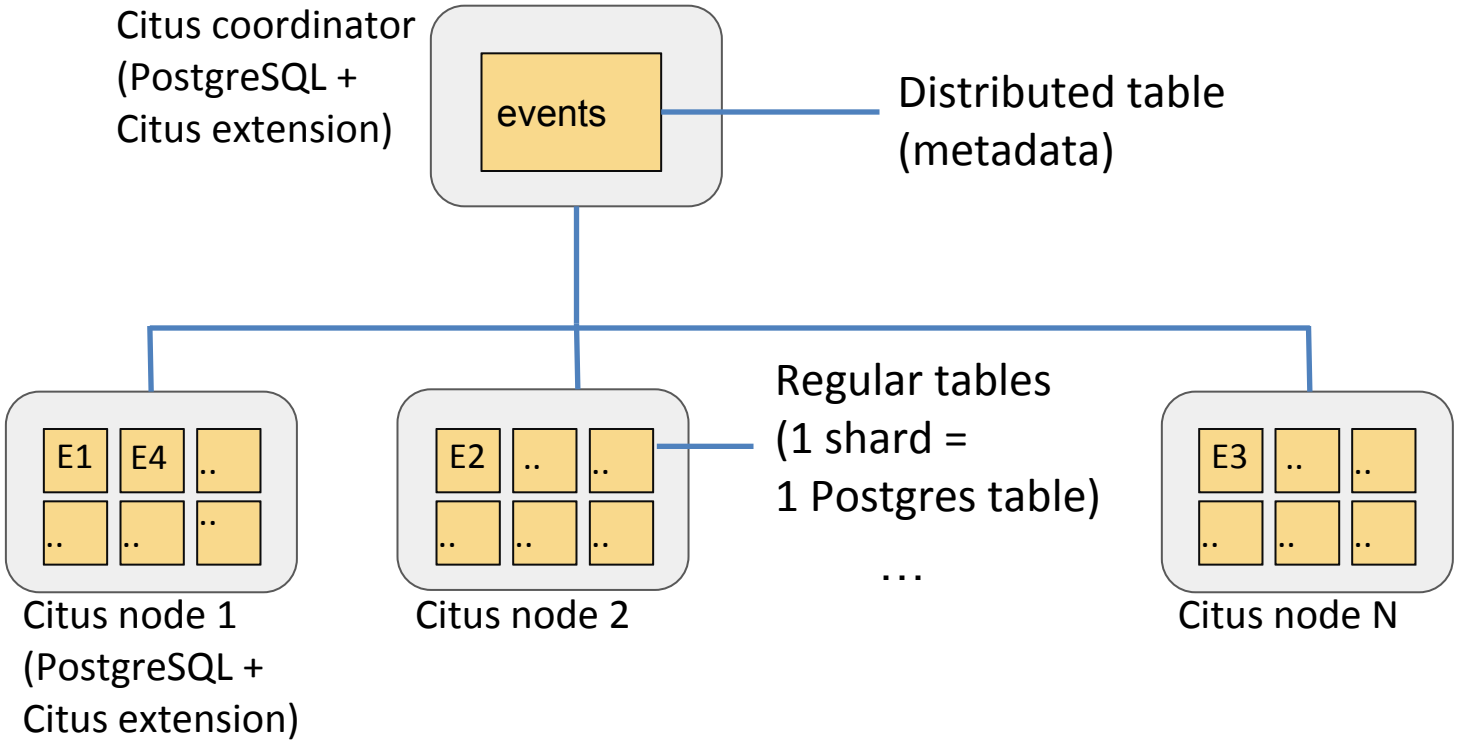
```
SELECT master_add_node('10.0.53', 5432);
SELECT master_add_node('10.1.54', 5432);
```

Create distributed table:

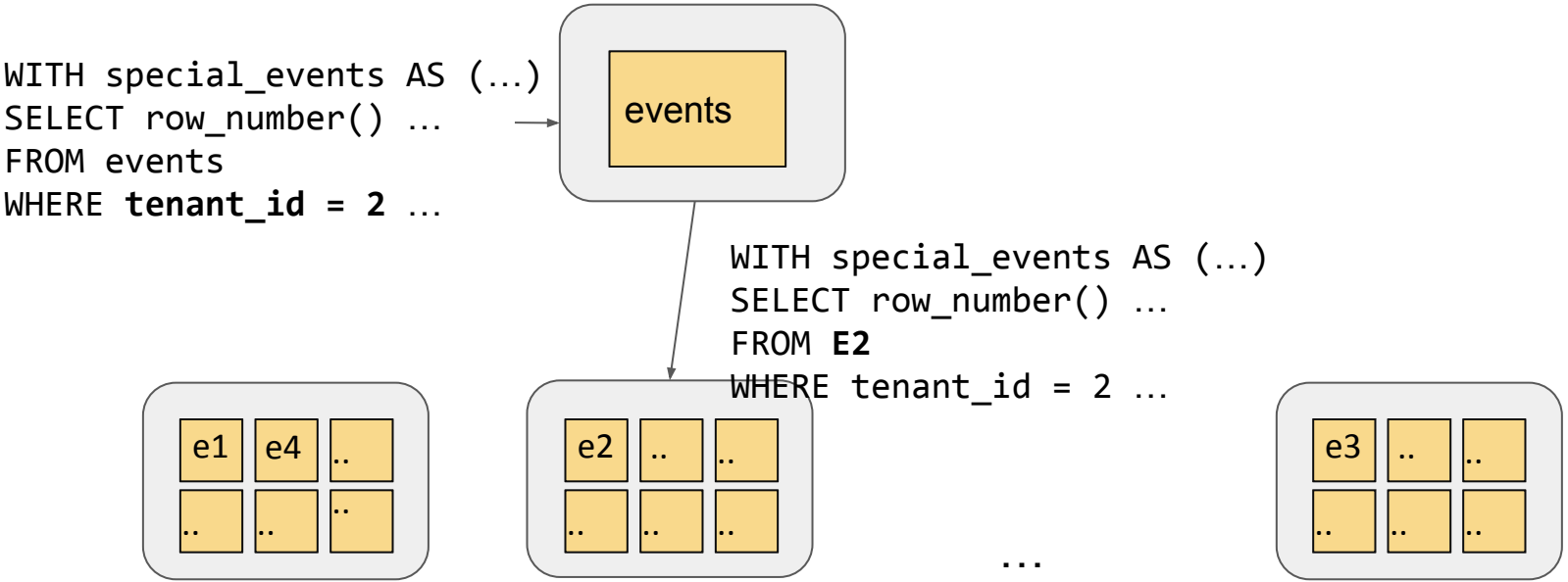
```
CREATE TABLE events (tenant_id int, ...);
SELECT create_distributed_table('events', 'tenant_id');
```

Events is now distributed across shards on the nodes.

Citus Architecture

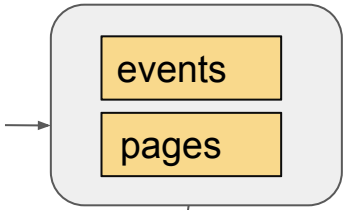


Single-node queries: Full SQL push-down

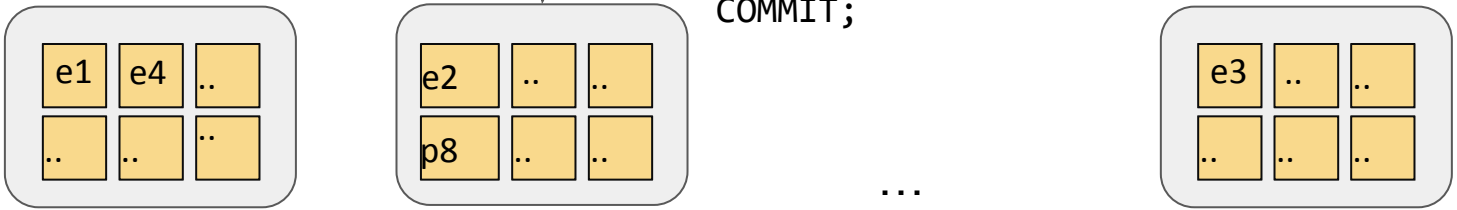


Single-node writes: Transaction push-down

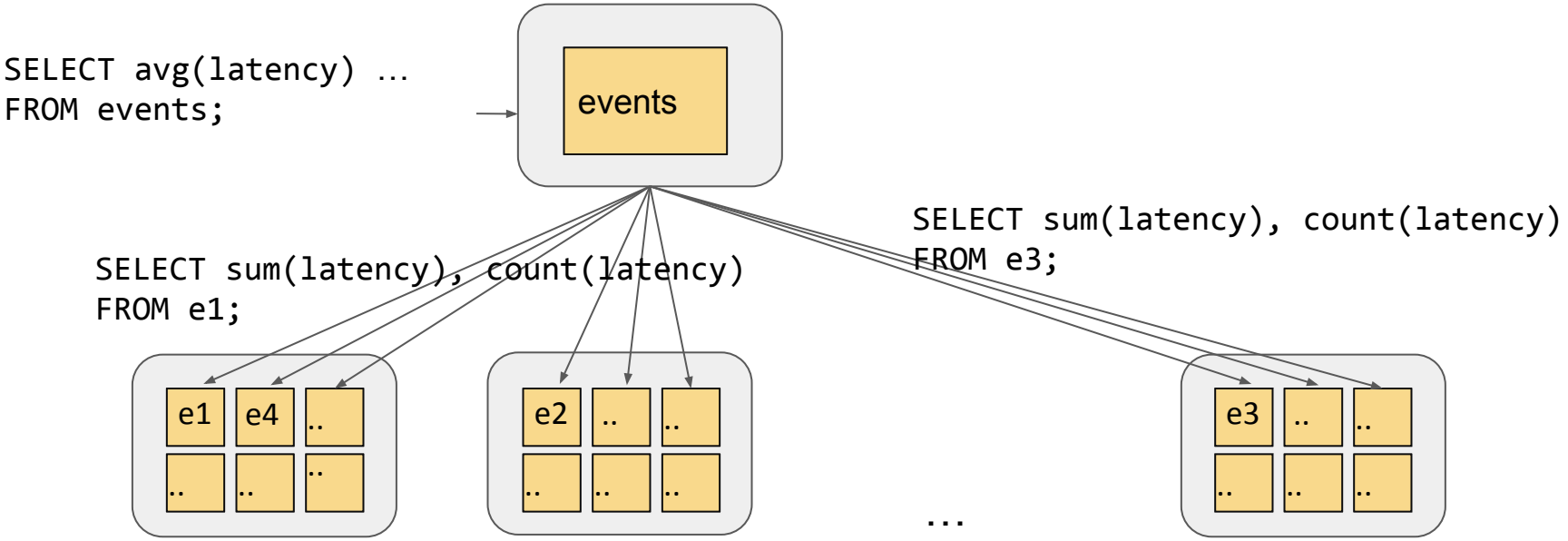
```
BEGIN;
INSERT INTO events ...
UPDATE pages ...
COMMIT;
```



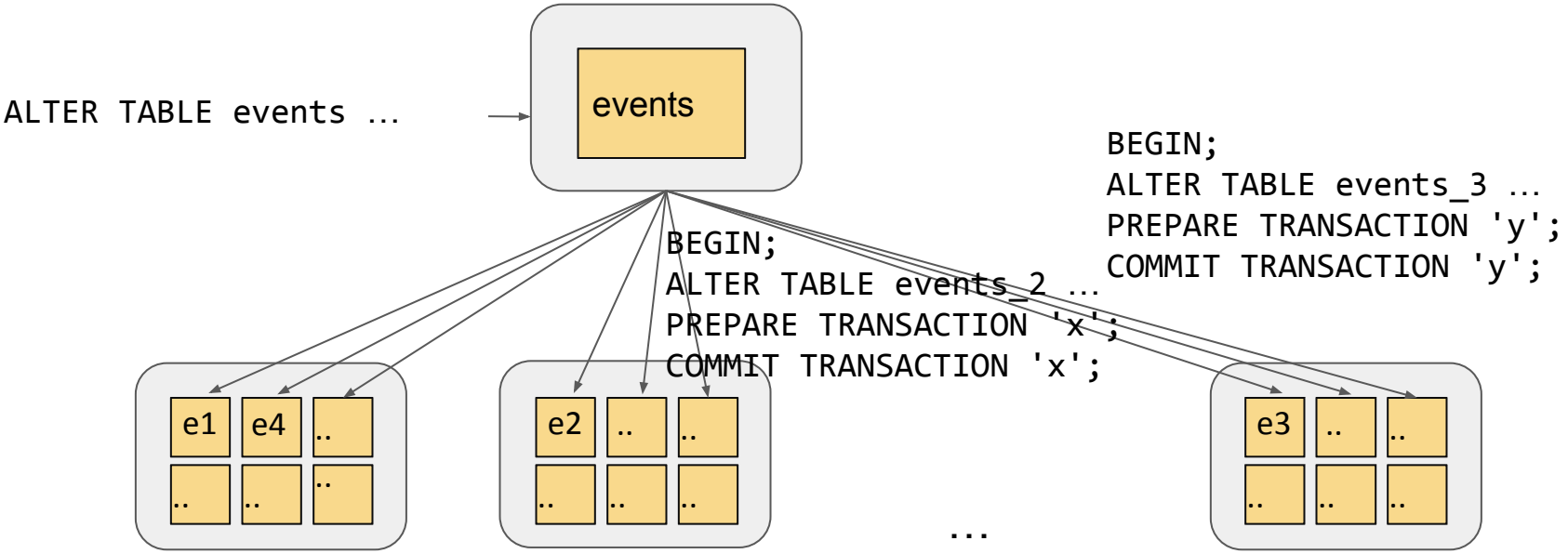

```
BEGIN;
INSERT INTO events_2 ...
UPDATE pages_8 ...
COMMIT;
```



Multi-node queries: Parallel SQL subset



Multi-node writes: Parallel DDL, COPY, … in 2PC



When to use Citus for scaling out?

Citus is suitable for scaling out several broad use-cases:

- **Multi-tenant (SaaS) applications - Shard by tenant**
  Citus co-locates data, routes queries, offers full SQL, ACID transactions
- **Real-time analytics applications - Shard by entity**
  Citus parallelises analytical queries, COPY, INSERT..SELECT
- **Key-value storage - Shard by key**
  Citus routes queries, parallelises secondary index queries

Multi-tenant applications - Shard by tenant

In a SaaS, almost all queries and transaction concern only one tenant.

Multi-tenancy using Citus:

- Add a `tenant_id` column to all tables
- Distribute all tables by tenant_id - Citus ensures **data co-location**
- Include `WHERE tenant_id = $1` in queries
- Include `tenant_id` in joins and foreign keys

Citus offers **full SQL** for selects, **ACID** transactions, **parallel DDL**, foreign keys.

Can easily convert existing applications (e.g. using activerecord-multi-tenant).

Real-time analytics applications - Shard by entity

High volume **event stream** with **real-time analytical dashboard** using Citus:

- Distribute event table(s) by entity_id (e.g. pages)
- Create reference tables for common attributes (e.g. users)
- **Bulk load** event data using parallel COPY
- Create **roll-ups** using parallel INSERT..SELECT
- Run **parallel SELECT** on the roll-ups/raw data

Citus offers tools for parallelising and scaling out the whole data pipeline.

Mainly suitable for new applications.

Key-value storage - Shard by key

Key-value storage using Citus (NoSQL++):

- Distribute tables by `key`
- Include `WHERE key = $1` in queries
- Use JSONB for unstructured data

Citus offers NoSQL functionality + parallel “secondary” index queries.

Could replace NoSQL (ORM) databases.

When not to use Citus?

Citus is not suitable for these use-cases:

- **Ad-hoc reporting queries (Data warehouse)**
  Not all SQL queries across shards are supported
- **Normalized data model**
  ACID transactions across shards are not supported

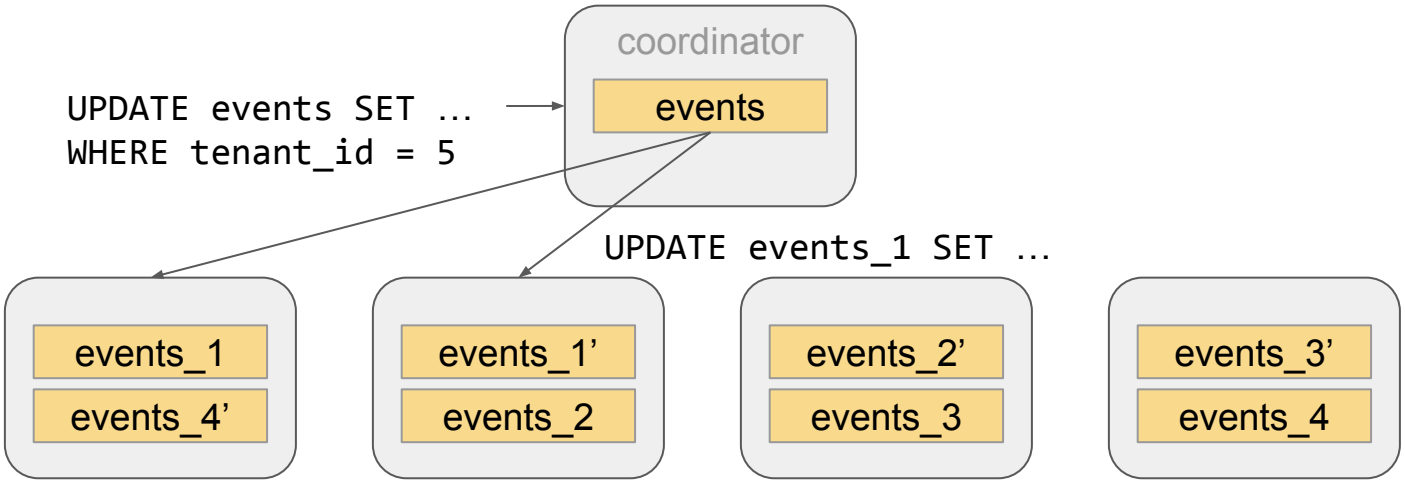
Citus Replication

Citus supports two replication models:

1. Replicate shards through **statement-based replication**
2. Replicate nodes through **streaming replication**

Statement-based replication in Citus

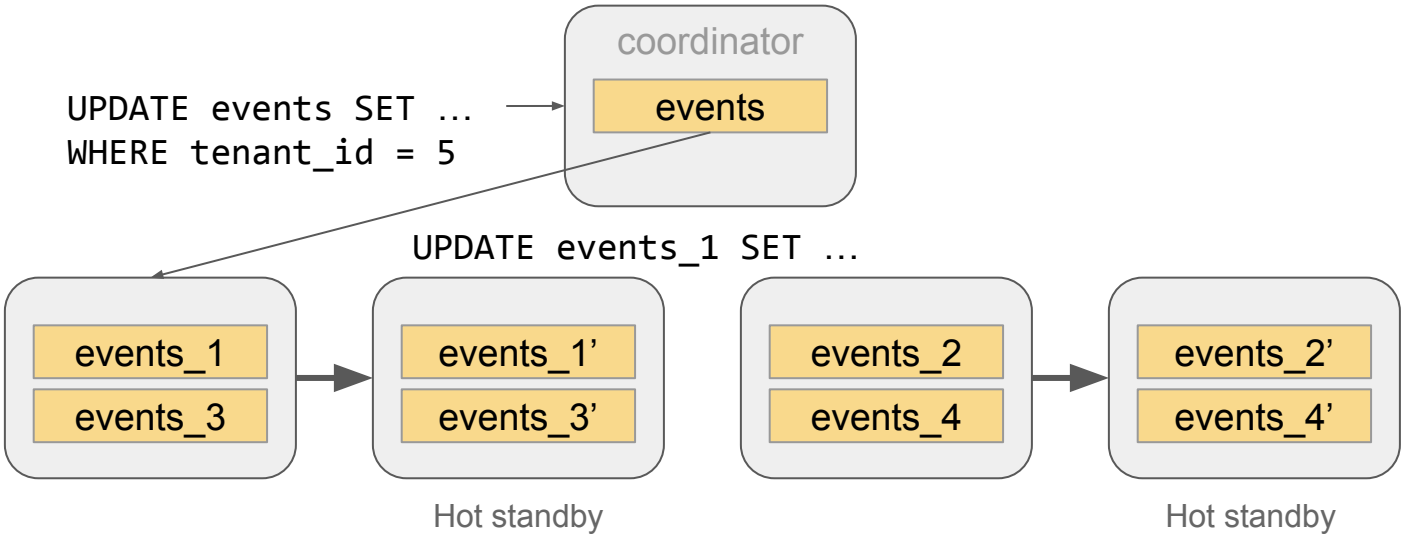
Replicate shards by sending DML to each node.



Single coordinator necessary for locking and tracking shard health.

Streaming replication in Citus (Cloud)

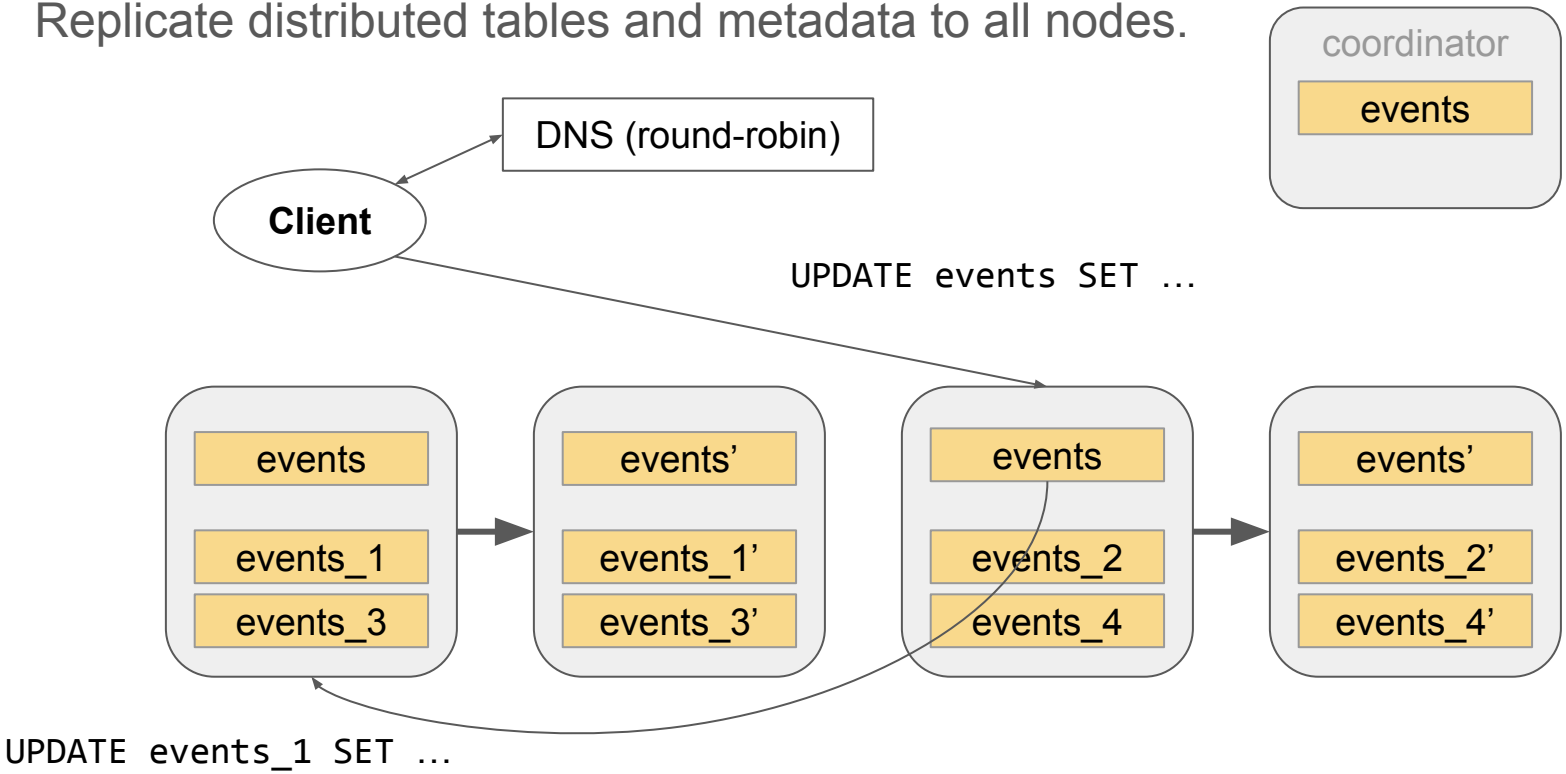
Replicate nodes using streaming replication and auto failover.



No locking and inactive shards: Single coordinator is no longer necessary.

Citus MX Architecture

Replicate distributed tables and metadata to all nodes.



The Citus MX project

Citus distributes tables across many servers to scale out queries.

Citus MX replicates distributed tables across many servers to scale out writes.

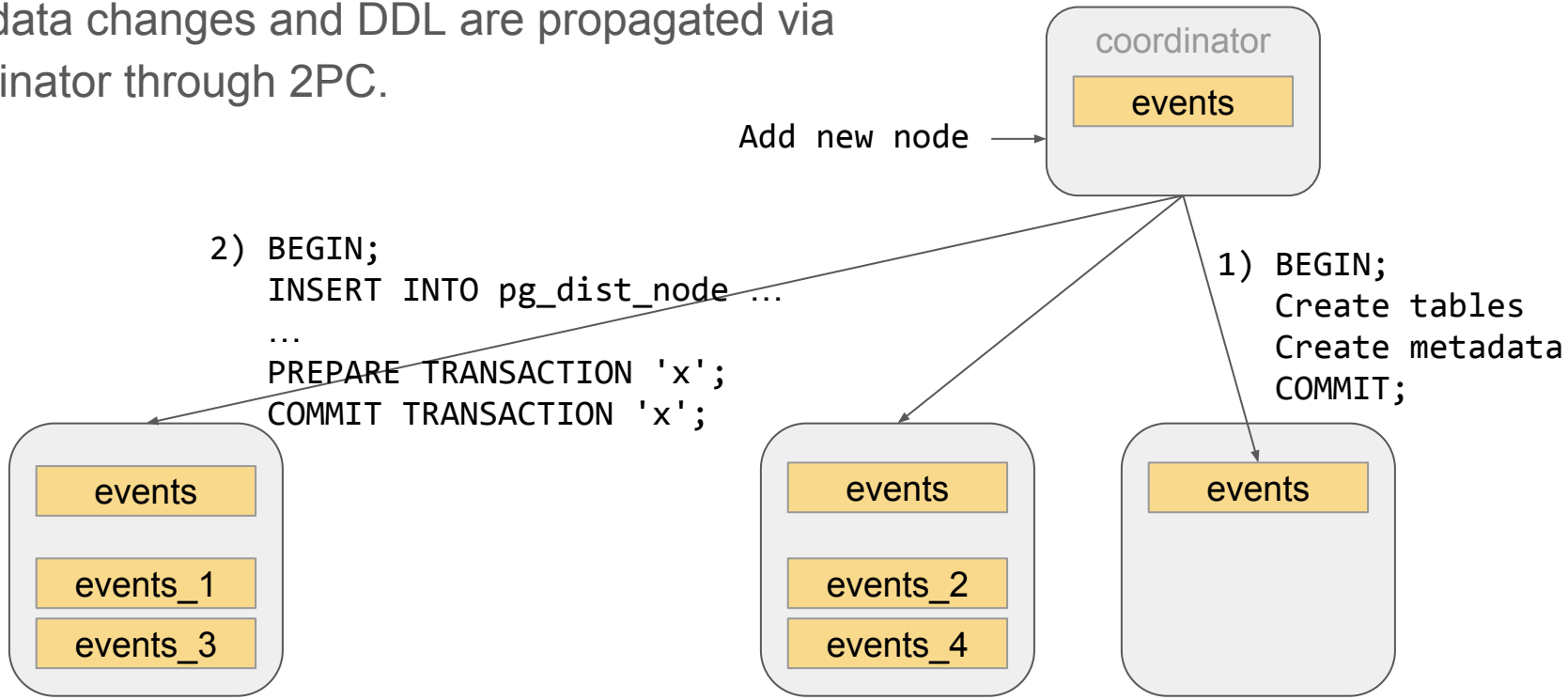
1. Automatically synchronize metadata to all nodes.
2. Mitigate inter-node connection explosion.
3. Become competitive with NoSQL on write-scalability.
4. Merge back into Citus.

In progress:

5. Streaming replication + auto failover for on-premises.

Metadata propagation (2PC)

Metadata changes and DDL are propagated via coordinator through 2PC.



2PC auto-recovery

Pre-commit on coordinator:

Write [*node ID*, *prepared transaction name*] records to pg_dist_transaction

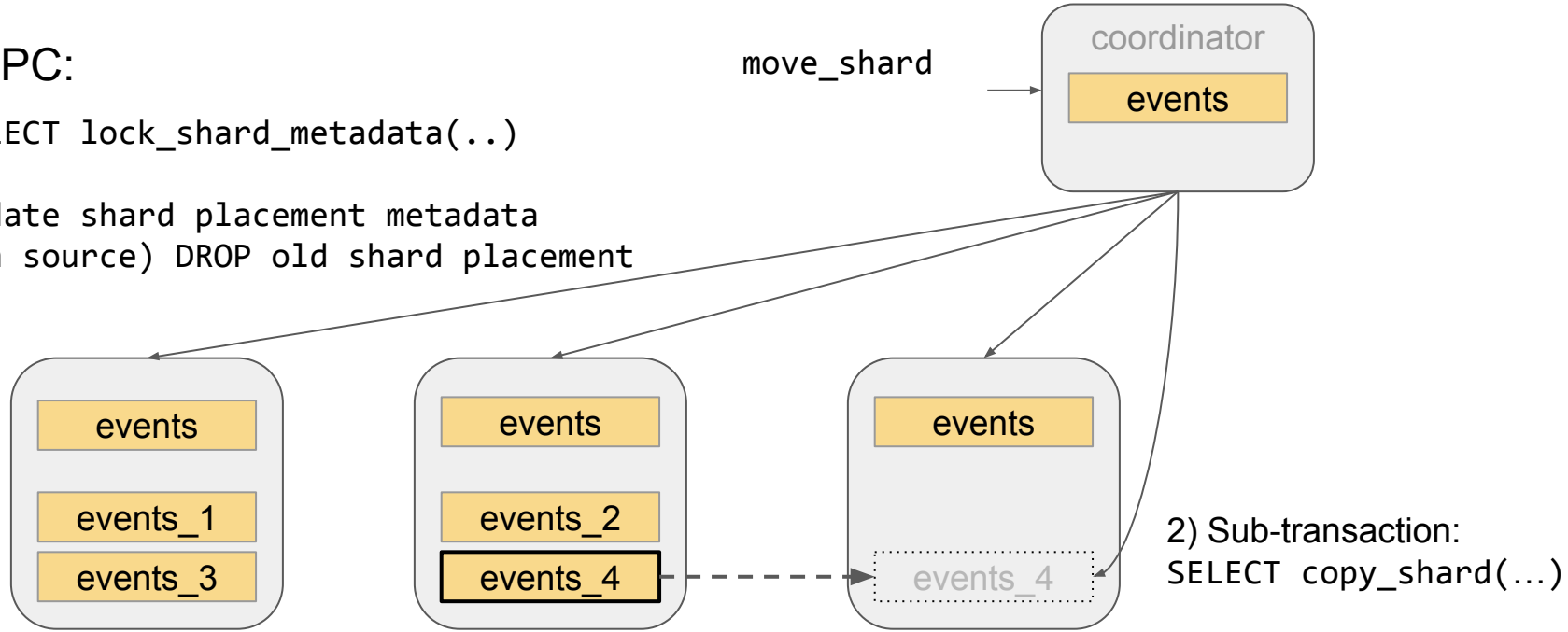
`SELECT recover_prepared_transactions()`:

Fetches prepared transactions from worker
If there is a corresponding record in pg_dist_transaction, commit
If there is no corresponding record in pg_dist_transaction, roll back

Shard copy

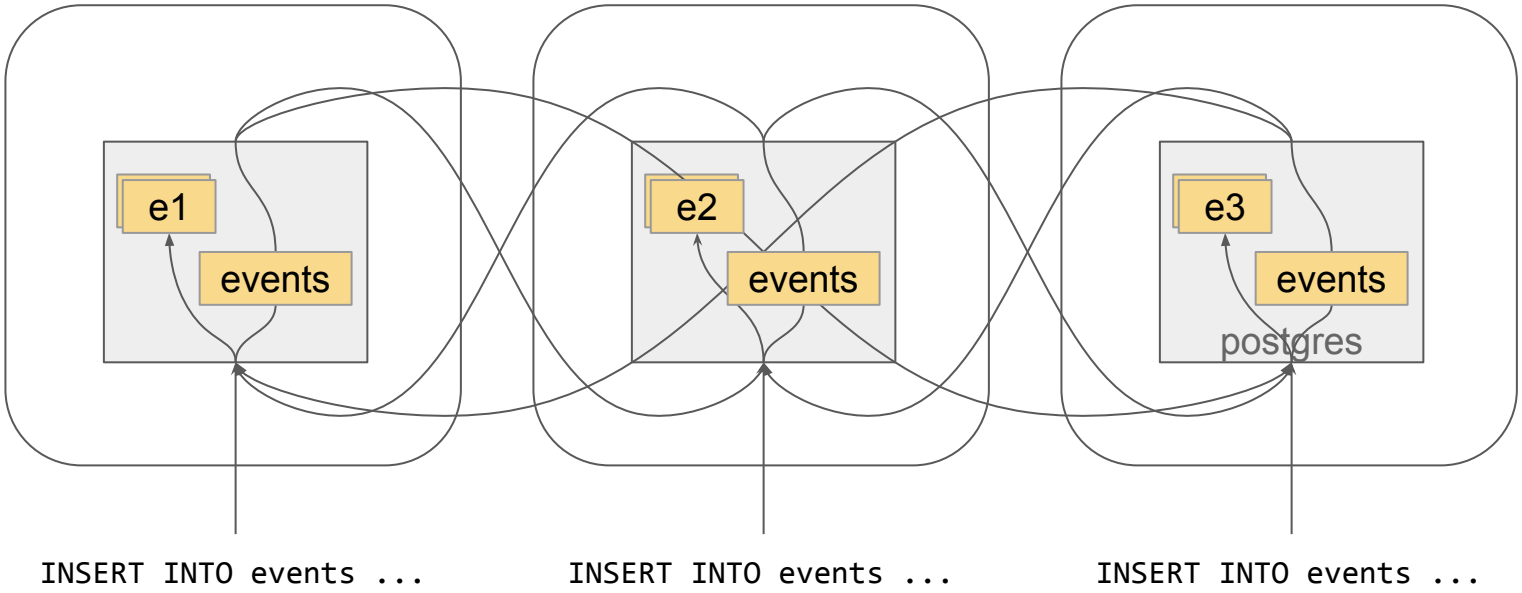
In a 2PC:

```
1) SELECT lock_shard_metadata(..)
...
3) Update shard placement metadata
4) (on source) DROP old shard placement
```



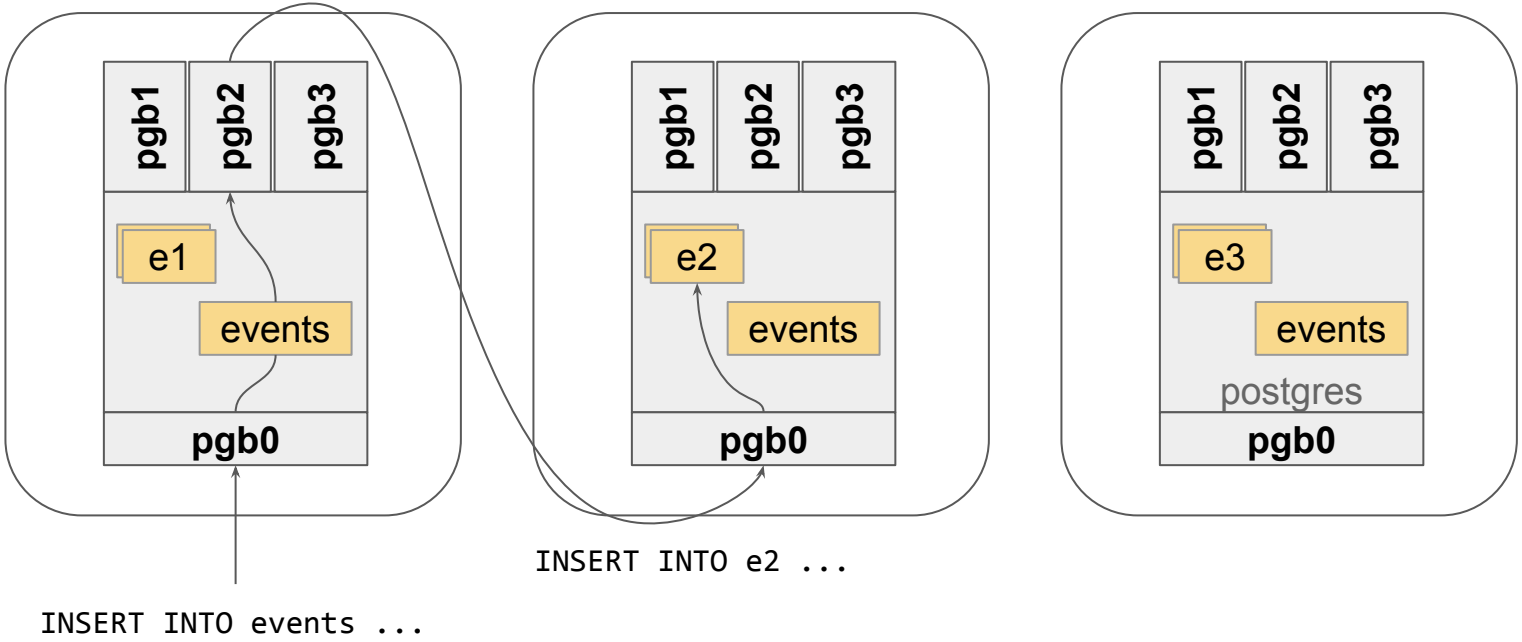
Quadratic Connection explosion

Citus requires many sessions to get high throughput under latency, but MX makes it quadratic...



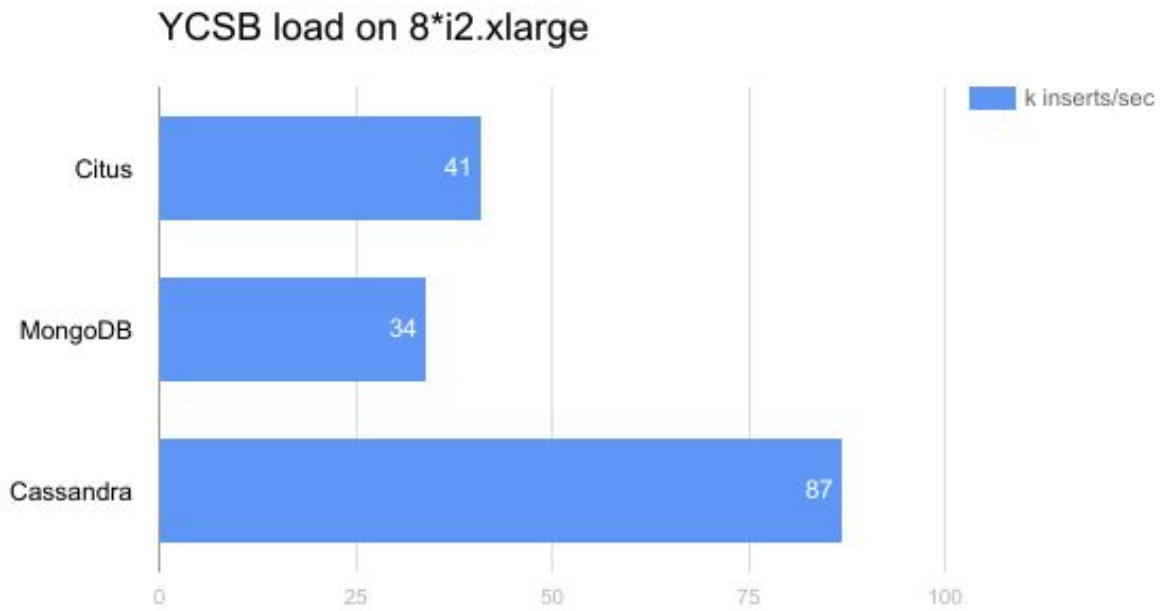
Pgbouncer pooling

Every node keeps a pool of 128/#nodes+1 connections to every other node.



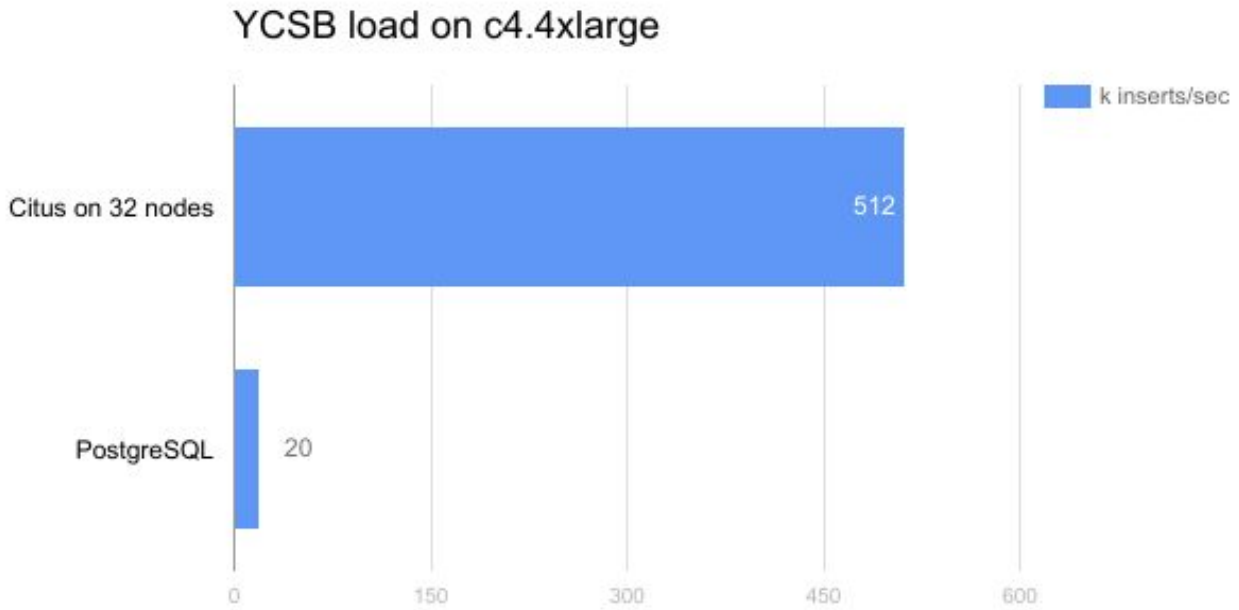
YCSB Benchmark

Compared to Datastax benchmark:



YCSB Benchmark

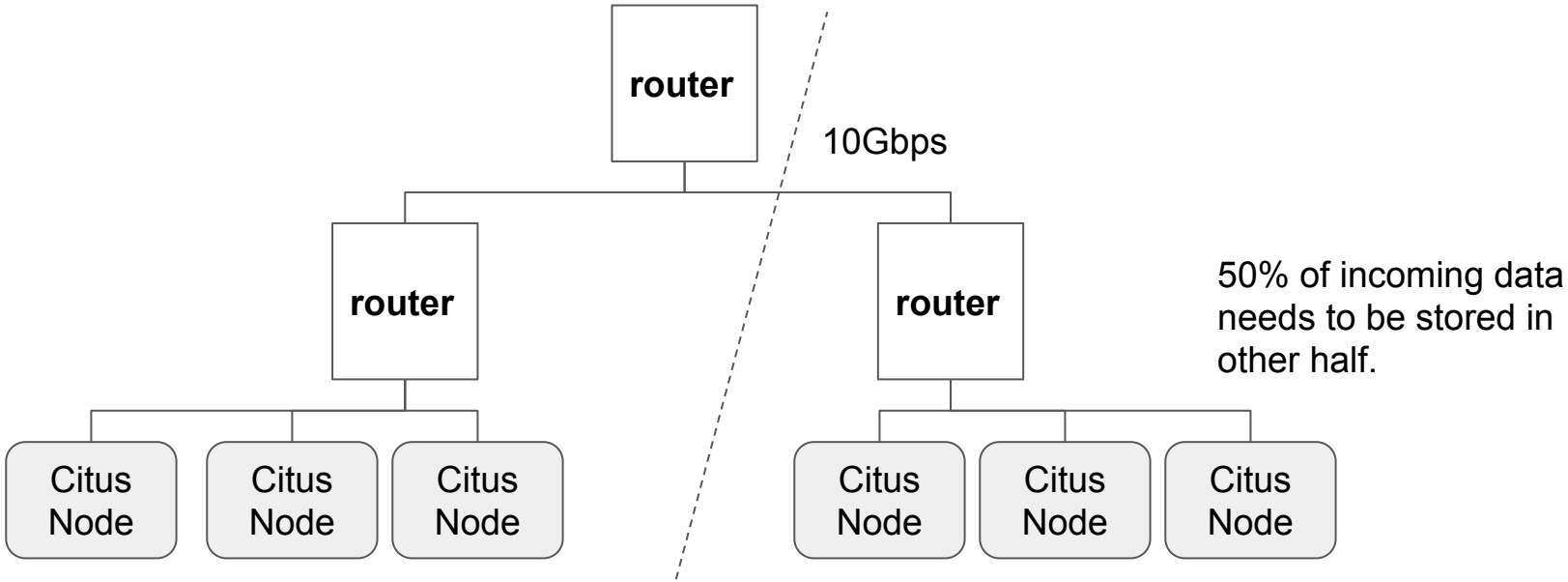
Compared to single-node PostgreSQL:



~7 million rows/sec with COPY

Network limits

Bisection bandwidth ultimately becomes a bottleneck:




citusdata

Next steps

Ongoing work:

- Citus auto failover solution for on-premises

Possible future steps:

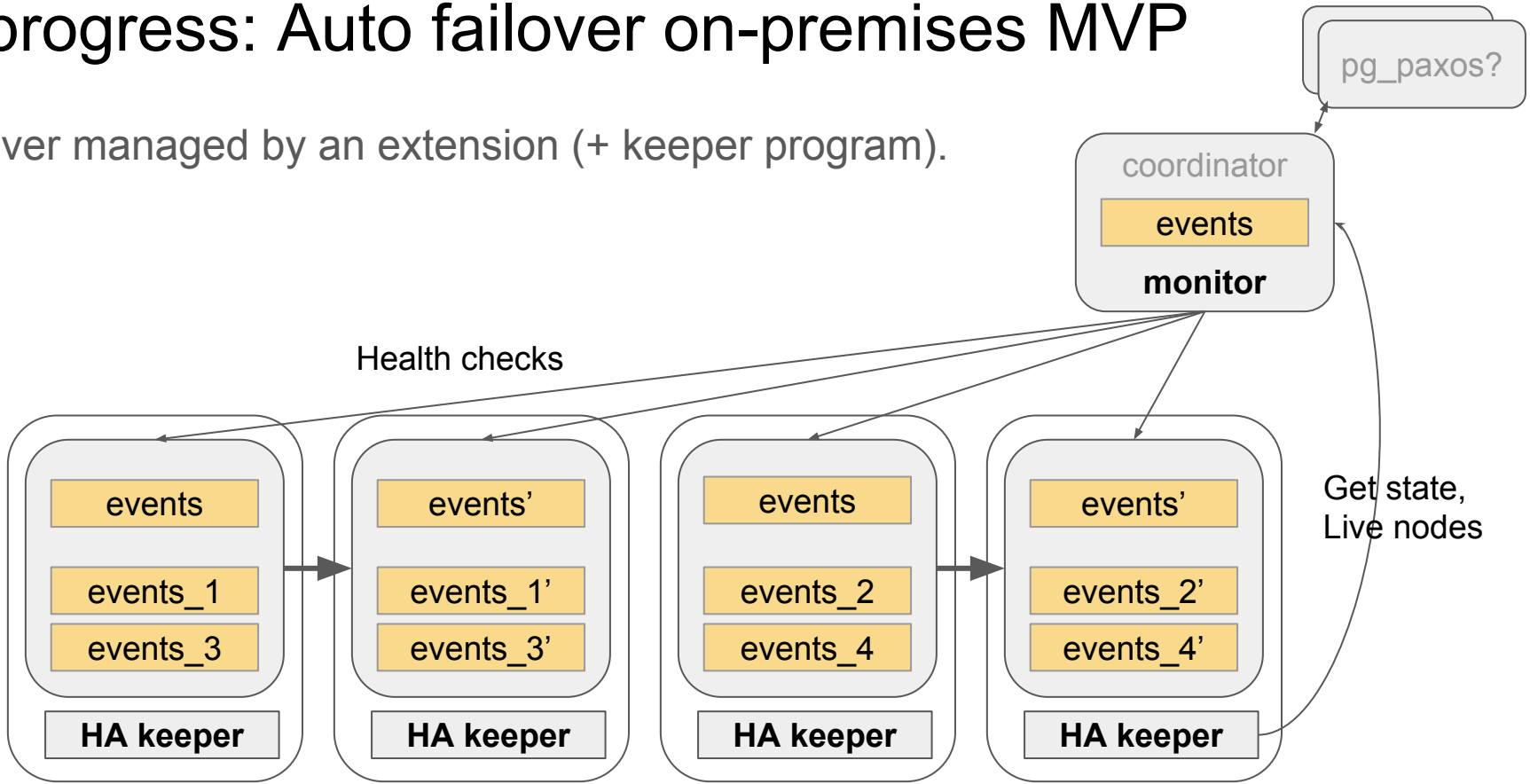
- Benchmark at 1M writes/sec
- Kubernetes?
- Integrate pgbouncer into Citus?
- pg_paxos for coordinator?

Questions?

marco@citusdata.com

In progress: Auto failover on-premises MVP

Failover managed by an extension (+ keeper program).



Distributed sequences