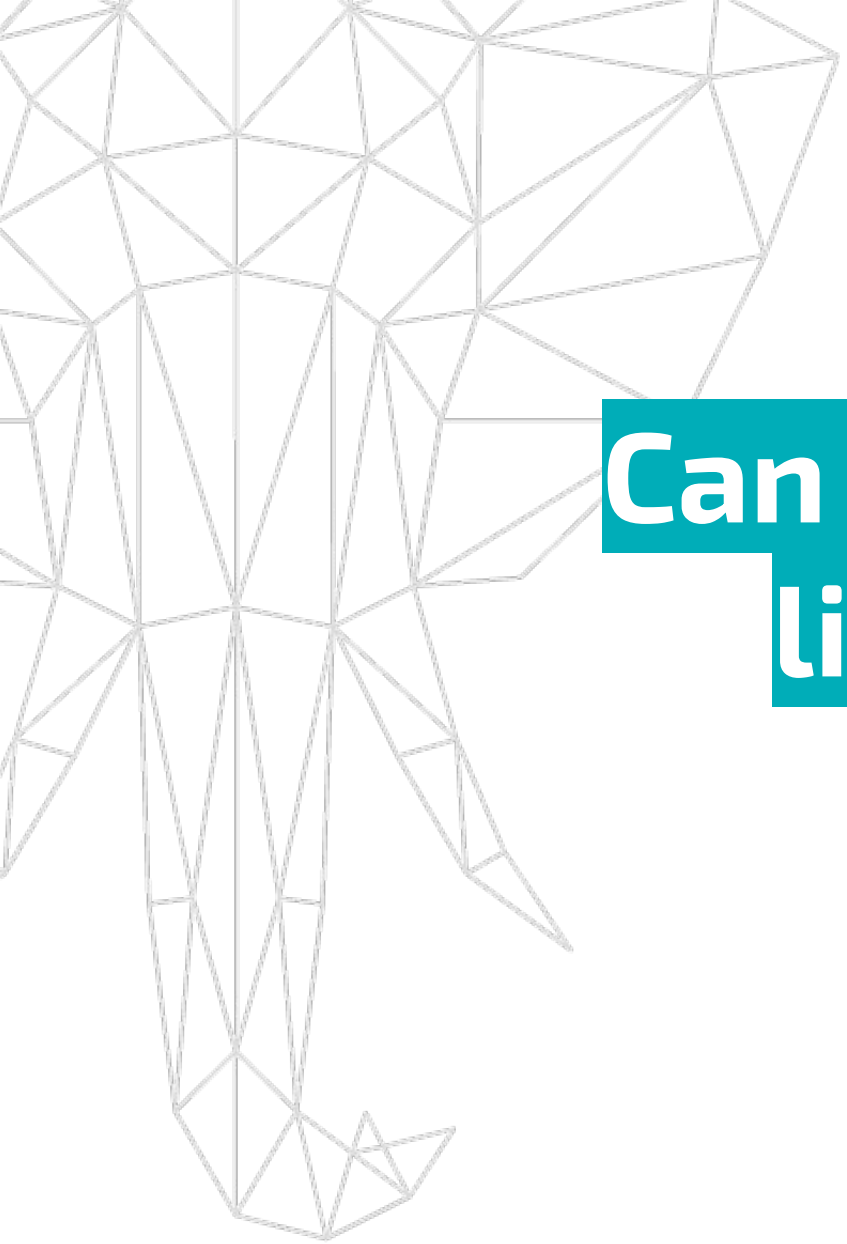


Can Postgres Scale like DynamoDB?

ONGRES

`whoami `

- Founder & CEO, OnGres
- 20+ years Postgres user and DBA
- Mostly doing R&D to create new, innovative software on Postgres
- Frequent speaker at Postgres, database conferences
- Principal Architect of StackGres, ToroDB
- Founder and President of the NPO *Fundación PostgreSQL*
- AWS Data Hero

Álvaro Hernández

aht.es

@ahachete

ONGRES

A little bit about DynamoDB

Is DynamoDB good?

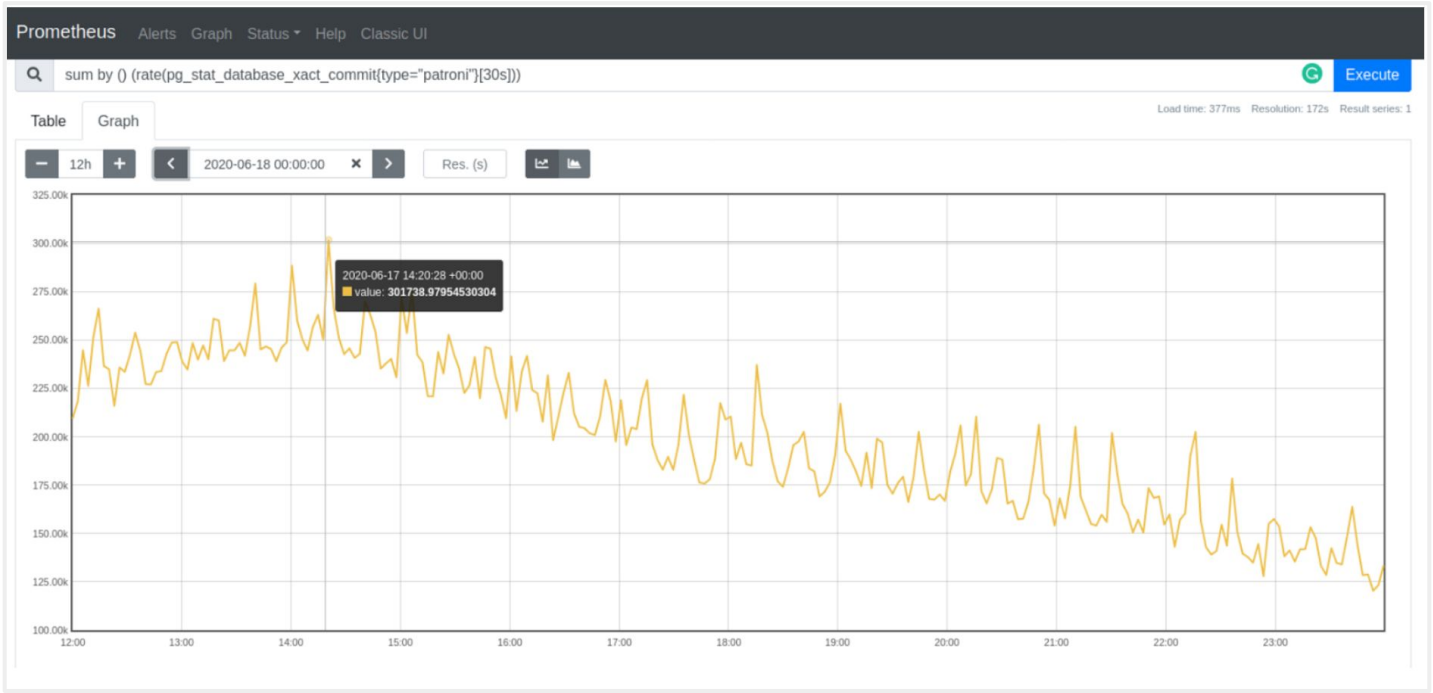
Storage for Prime Day

Amazon DynamoDB powers multiple high-traffic Amazon properties and systems including Alexa, the Amazon.com sites, and all Amazon fulfillment centers. Over the course of the 66-hour Prime Day, these sources made 16.4 trillion calls to the DynamoDB API, peaking at 80.1 million requests per second.

https://aws.amazon.com/blogs/aws/amazon-prime-day-2020-powered-by-aws/

ONGRES

A high-traffic Postgres example



GitLab.com spikes to >300K Postgres tx/s on a single cluster:

https://about.gitlab.com/blog/2020/09/11/gitlab-pg-upgrade/

ONGRES

DynamoDB is a building block, too

Impact on Other Services

There are several other AWS services that use DynamoDB that experienced problems during the event. Rather than list them all, which had similar explanations for their status, we'll list a few that customers most asked us about or where the actions are more independent from DynamoDB's Correction of Errors ("COE").

Simple Queue Service (SQS)

In the early stages of the DynamoDB event, the Amazon Simple Queue Service was delivering slightly elevated errors and latencies. Amazon SQS uses an internal table stored in DynamoDB to store information describing its queues. While the queue information is cached within SQS, and is not in the direct path for "send-message" and "receive-message" APIs, the caches are refreshed frequently to accommodate creation, deletion, and reassignment across infrastructure. When DynamoDB finished disabling traffic at 5:45am PDT (to enable the metadata service to recover), the Simple Queue Service was unable to read this data to refresh caches, resulting in significantly elevated error rates. Once DynamoDB began re-enabling customer traffic at 7:10am PDT, the Simple Queue Service recovered. No data in queues, or information describing queues was lost as a result of the event.

In addition to the actions being taken by the DynamoDB service, we will be adjusting our SQS metadata caching to ensure that send and receive operations continue even without prolonged access to the metadata table.

EC2 Auto Scaling

Between 2:15am PDT and 7:10am PDT, the EC2 Auto Scaling Service delivered significantly increased API faults. From 7:10am PDT to 10:52am PDT, the Auto Scaling service was substantially delayed in bringing new instances into service, or terminating existing unhealthy instances. Existing instances continued to operate properly throughout the event.

Auto Scaling stores information about its groups and launch configurations in an internal table in DynamoDB. When DynamoDB began to experience elevated error rates starting at 2:19am PDT, Auto Scaling could not update this internal table when APIs were called. Once DynamoDB began recovery at 7:10am PDT, the Auto Scaling APIs recovered. Recovery was incomplete at this time, as a significant backlog of scaling activities had built up throughout the event. The Auto Scaling service executes its launch and termination activities in a background scheduling service. Throughout the event, a very large amount of pending activities built up in this job scheduler and it took until 10:52am PDT to complete all of these tasks.

In addition to the actions taken by the DynamoDB team, to ensure we can recover quickly when a large backlog of scaling activities accumulate, we will adjust the way we partition work on the fleet of Auto Scaling servers to allow for more parallelism in processing these jobs, integrate mechanisms to prune older scaling activities that have been superseded, and increase the capacity available to process scaling activities.

CloudWatch

Starting at 2:35am PDT, the Amazon CloudWatch Metrics Service began experiencing delayed and missing EC2 Metrics along with slightly elevated errors. CloudWatch uses an internal table stored in DynamoDB to add information regarding Auto Scaling group membership to incoming EC2 metrics. From 2:35am PDT to 5:45am PDT, the elevated DynamoDB failure rates caused intermittent availability of EC2 metrics in CloudWatch. CloudWatch also observed an abnormally low rate of metrics publication from other services that were experiencing issues over this time period, further contributing to missing or delayed metrics.

Then, from approximately 5:51am PDT to 7:10am PDT CloudWatch delivered significantly elevated error rates for PutMetricData calls affecting all AWS Service metrics and custom metrics. The impact was due to the significantly elevated error rates in DynamoDB for the group membership additions mentioned above. The CloudWatch Metrics Service was fully recovered at 7:29am PDT.

We understand how important metrics are, especially during an event. To further increase the resilience of CloudWatch, we will adjust our caching strategy for the DynamoDB group membership data and only require refresh for the smallest possible set of metrics. We also have been developing faster metrics delivery through write-through caching. This cache will provide the ability to present metrics directly before persisting them and will, as a side benefit, provide additional protection during an event.

Console

https://aws.amazon.com/message/5467D2/

ONGRES

What is DynamoDB

- A scale-out, NoSQL database
- Key-Value:
  - Key: a simple or composite PK
  - Value: a JSON blob
- Consistent performance at any scale: single-digit ms queries
- Severless
- Pay-per-use
  - WCUs, RCUs
  - Storage, data transfer

ONGRES

What makes DynamoDB so successful

- Yeah, that it's serverless.
- Yeah, that it scales without limits.
- But in reality, what makes DynamoDB unique is:

  Consistent and low latency at any scale. Below 10ms

ONGRES

What makes DynamoDB so special

- Yeah, that it's serverless.
- Yeah, that it scales without limits.
- But in reality, what makes DynamoDB unique is:

  Consistent and low latency at any scale. Below 10ms

- What, 10ms???? My Postgres answers queries in less than 1ms!

ONGRES

What makes DynamoDB so special

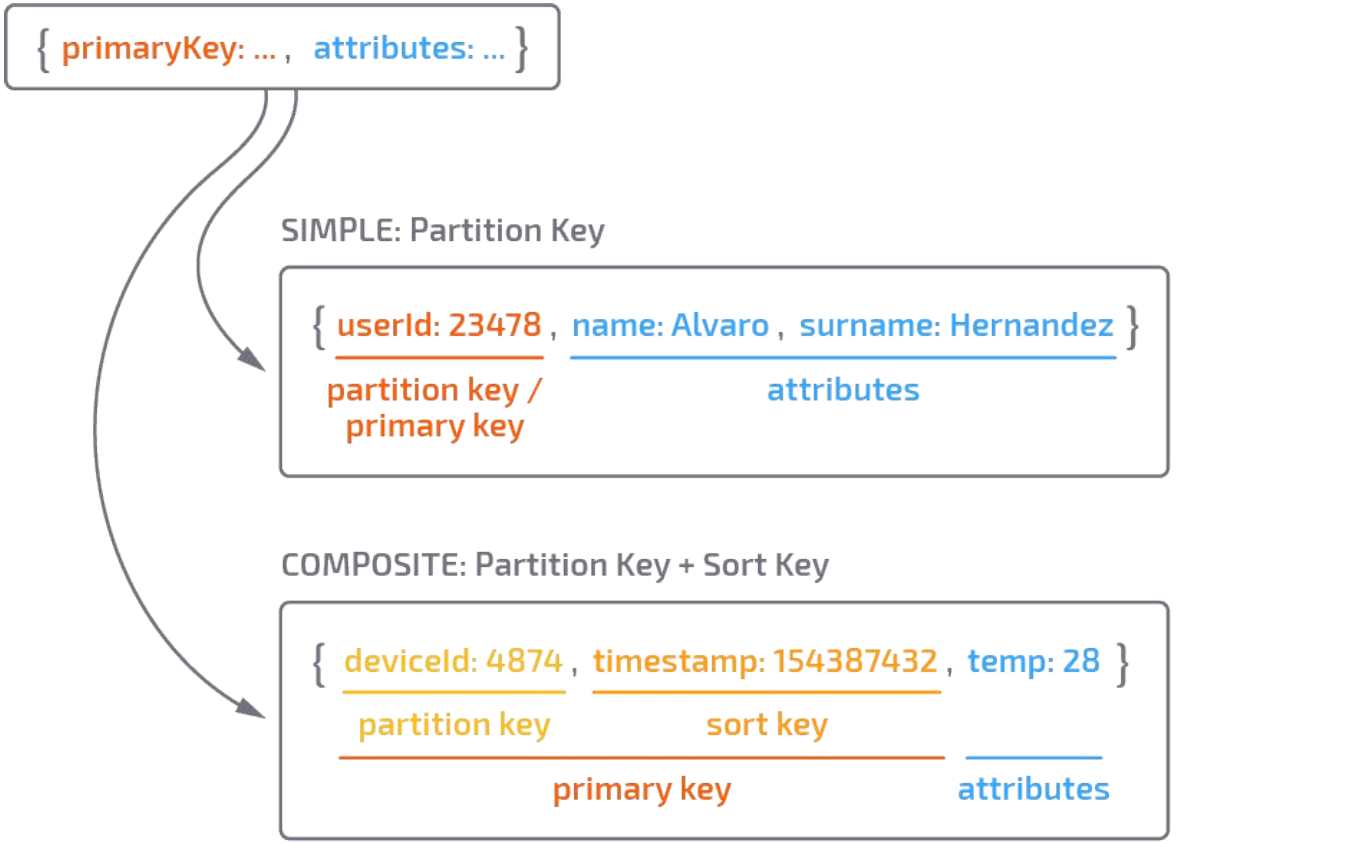
- Yeah, that it's serverless.
- Yeah, that it scales without limits.
- But in reality, what makes DynamoDB unique is:

  Consistent and low latency at any scale. Below 10ms

- What, 10ms???? My Postgres answers queries in less than 1ms!
- At any scale?
- **Consistently? What are your p99 response times?**

ONGRES

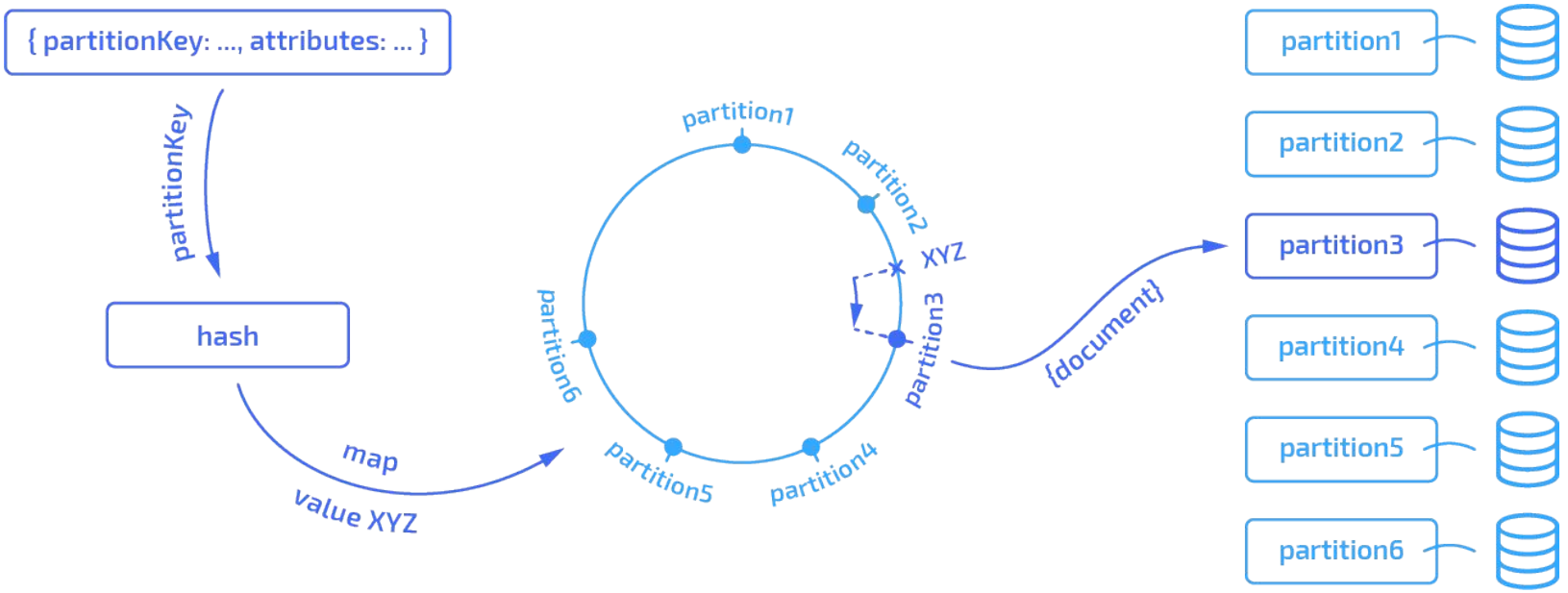
DynamoDB Data Model



- Primary key has schema: fixed datatype (number, string, binary) and NOT NULL, UNIQUE.
- Attributes are optional and schema-less.

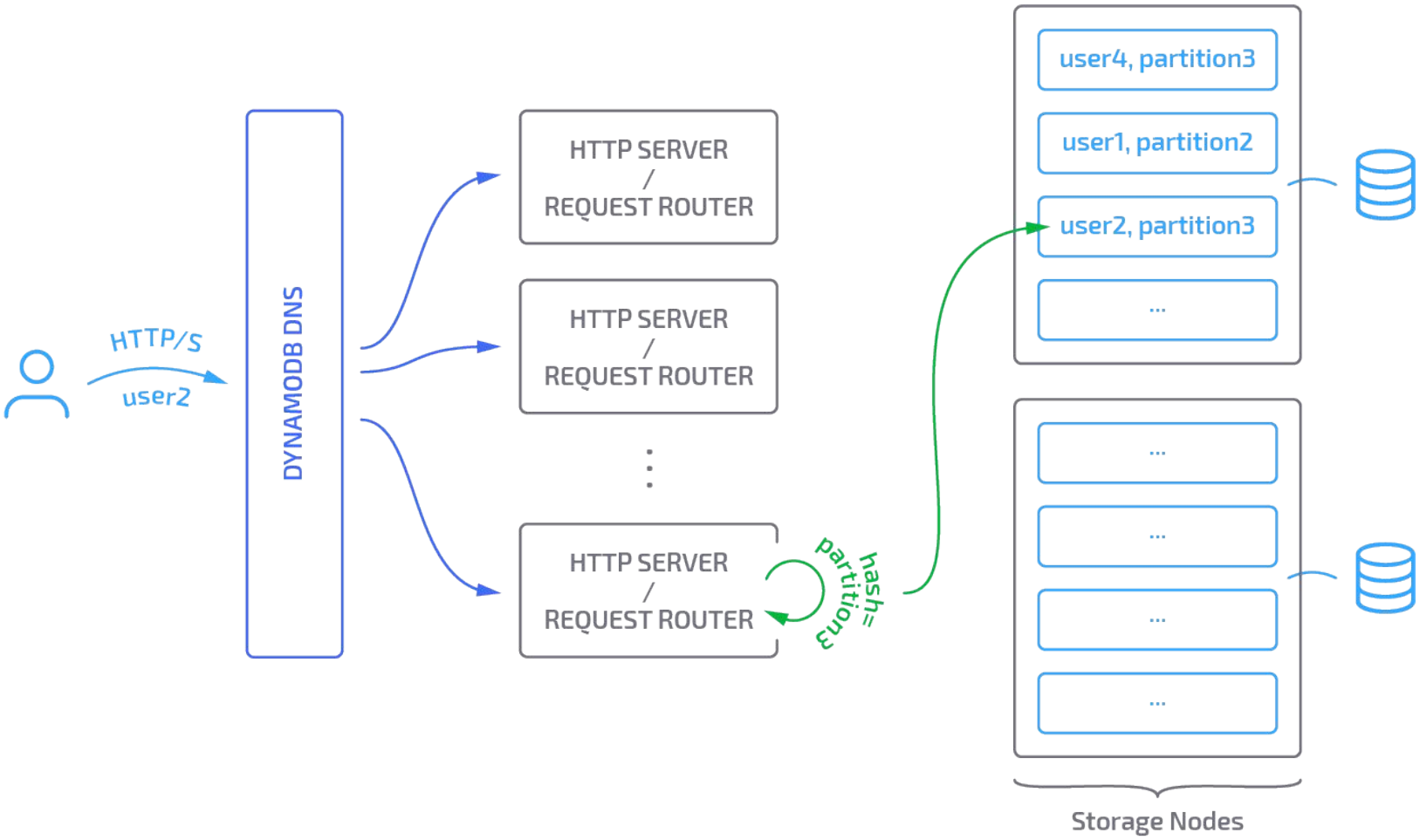
ONGRES

DynamoDB Sharding Logic



ONGRES

DynamoDB (simplified) Request Routing



ONGRES

DynamoDB (relevant) Operations

- **Single-value, single-partition operations**:
  - PutItem, DeleteItem, GetItem, UpdateItem
  - Compute hash of partition key, go to shard, operate on value
- **Multiple-value, single-partition operations**:
  - Query. Reads values with the same hash, sorted by sort key
- **Multiple-value, multiple-partition operations**:
  - Scan
  - Supports (server assisted) parallelism
- Multiple-value operations: max 1MB results, provides pagination mechanisms, filtering (still consumes RCUs!)

ONGRES

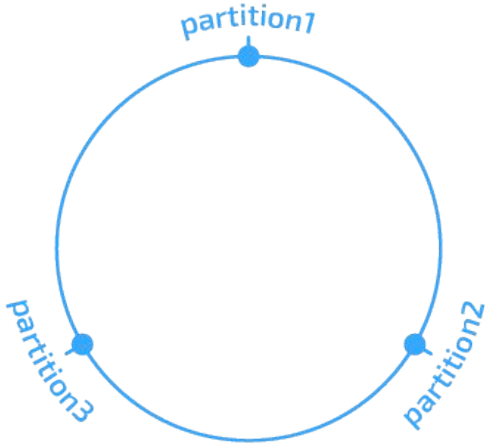
DynamoDB (missing?) Operations

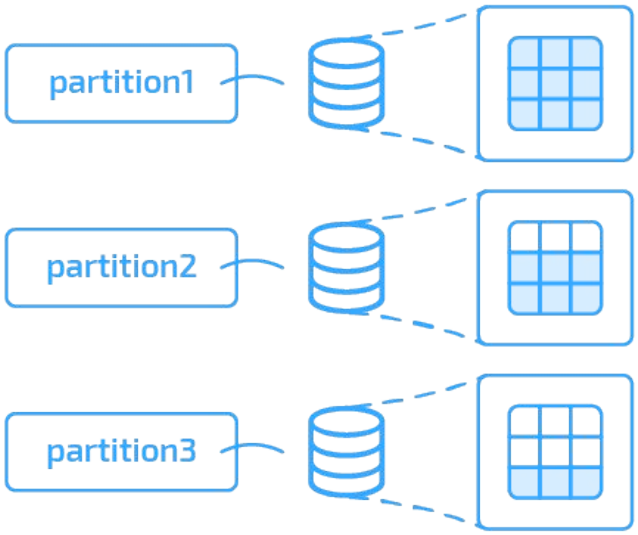
- No joins
- No aggregations
- No advanced queries (windows, subqueries…)

Why??

By design. To keep latency single-digit ms.

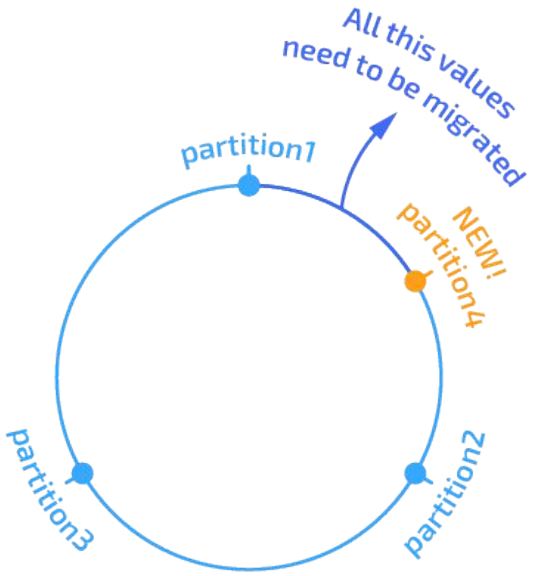
DynamoDB Scaling

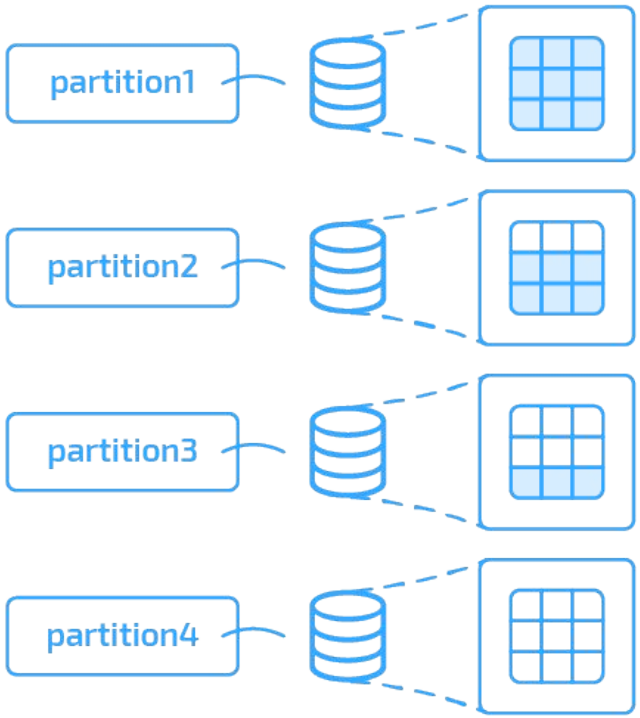




ONGRES

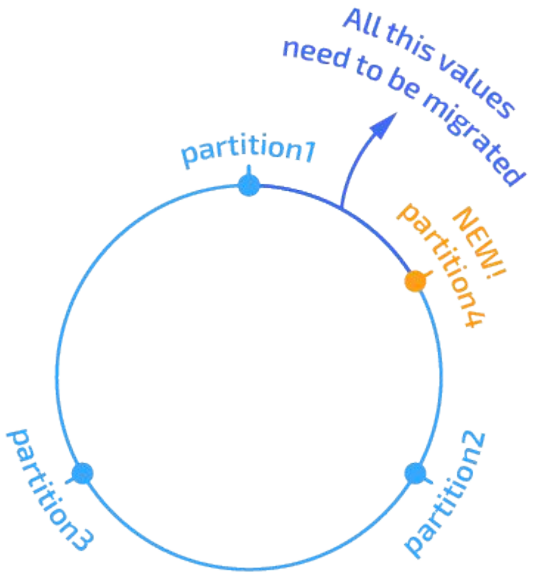
DynamoDB Scaling

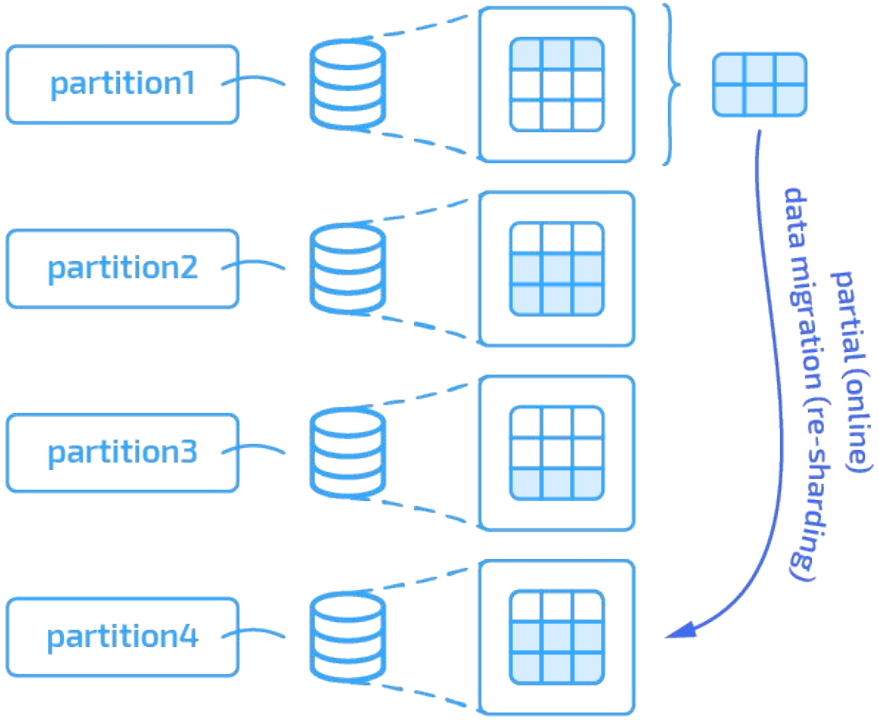




ONGRES

DynamoDB Scaling



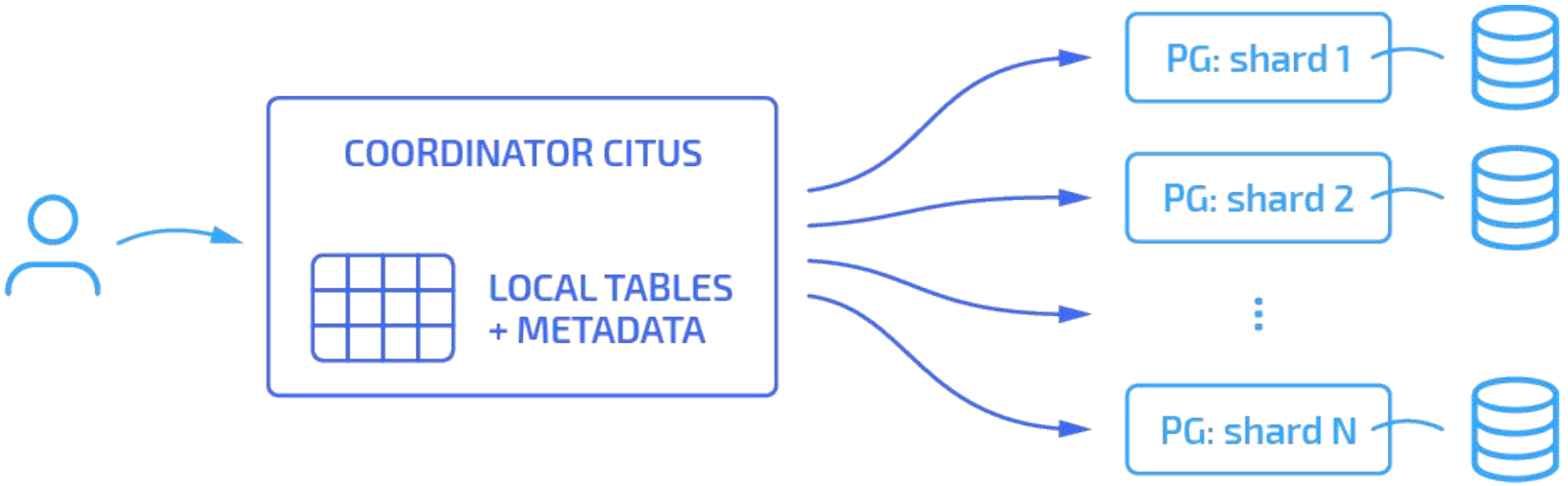


ONGRES

Can Postgres scale like DynamoDB?

ONGRES

Option #1. Coordinator model: Citus



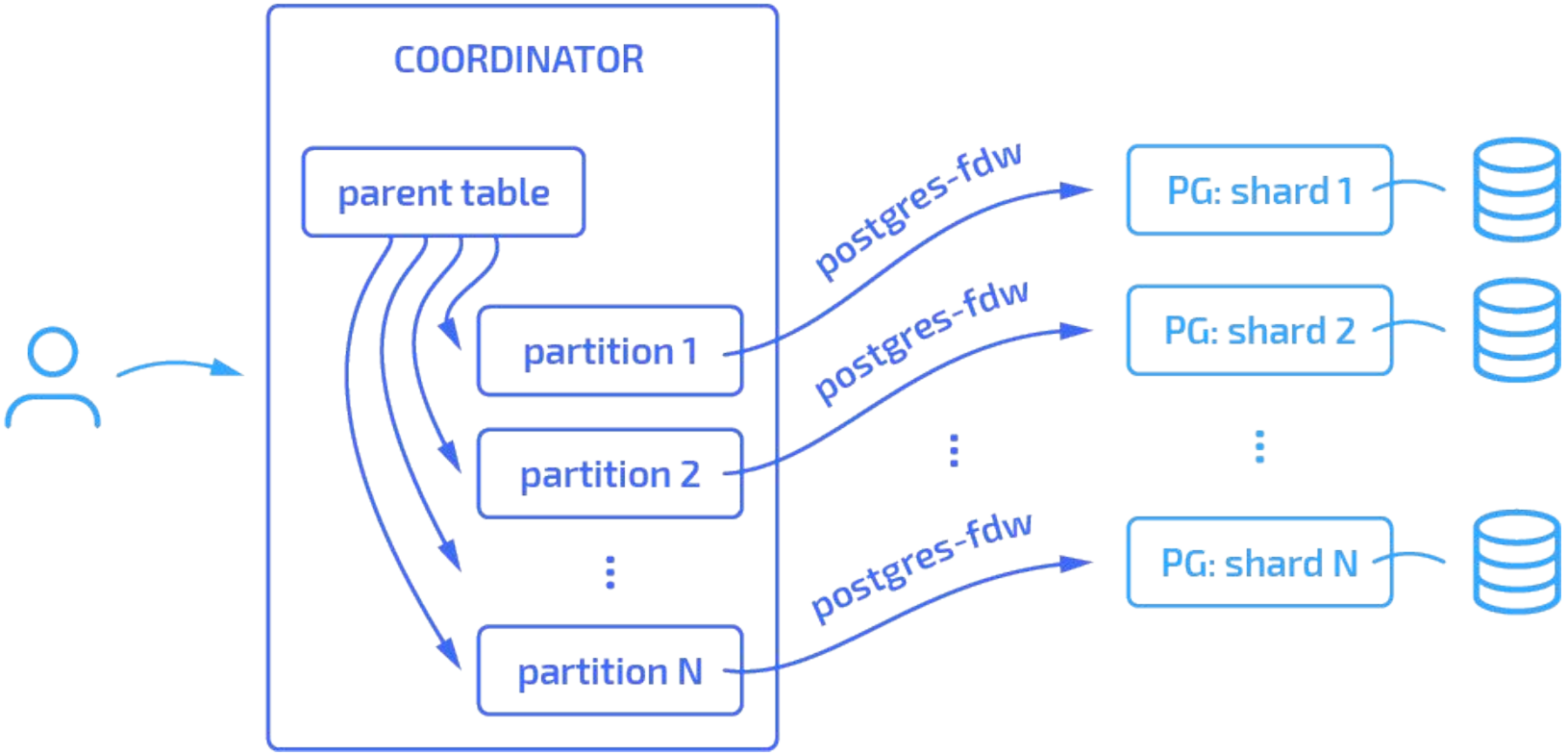
ONGRES

Citus limitations for DynamoDB scale

- Single controller
  - Controller has a bit of state (metadata + local tables)
  - It's possible to have multiple (with replication among them), but is not mainstream
  - Don't use local tables

- **Main reason: processing time in the controller is not guaranteed to scale like DynamoDB. Complex queries and scatter-gather communication with shards are an anti-pattern in DynamoDB model**.

Option #2. Coordinator model: postgres_fdw



ONGRES

postgres_fdw limitations for DynamoDB scale

- postgres_fdw limitations
  - Doesn't push down all the clauses
  - When talking to multiple shards, it works serially
  - Requires connection pooling

- **Main reason: processing time in the controller is not guaranteed to scale like DynamoDB. Complex queries and scatter-gather communication with shards are an anti-pattern in DynamoDB model**.

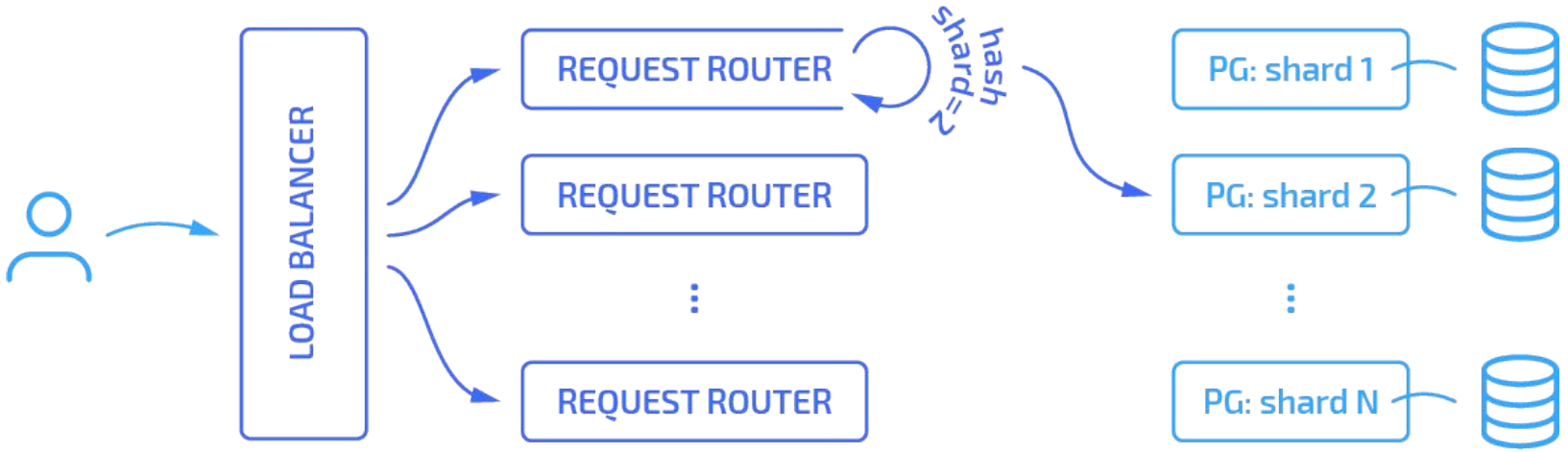
ONGRES

Application-based sharding

- Noted that the main reason for not achieving DynamoDB scale with either Citus or postgres_fdw is essentially the same?
- Processing time in the coordinator and complexity of allowed operations violate DynamoDB's main promise: single-digit ms response times.

- What's the alternative then?
- **Application-based sharding**.
- Involving the client or application in the sharding process, sending the queries directly to the appropriate shard.
- Except for scan, all operations are single-shard (single partition)

Postgres application-based sharding



ONGRES

Possible table structure

```
          Table "public.pglikedy_simple"
┌─────────┬────────┬───────────┬──────────┬─────────┐
│ Column  │  Type  │ Collation │ Nullable │ Default │
├─────────┼────────┼───────────┼──────────┼─────────┤
│ hash    │ bigint │           │ not null │         │
│ content │ jsonb  │           │ not null │         │
└─────────┴────────┴───────────┴──────────┴─────────┘
Indexes:
    "pglikedy_simple_hash_key" UNIQUE CONSTRAINT, btree (hash)
    "pglikedy_simple_pk" UNIQUE, btree ((content -> 'partitionKey'::text))


          Table "public.pglikedy_composite"
┌─────────┬────────┬───────────┬──────────┬─────────┐
│ Column  │  Type  │ Collation │ Nullable │ Default │
├─────────┼────────┼───────────┼──────────┼─────────┤
│ hash    │ bigint │           │ not null │         │
│ content │ jsonb  │           │ not null │         │
└─────────┴────────┴───────────┴──────────┴─────────┘
Indexes:
    "pglikedy_composite_pk" UNIQUE, btree ((content -> 'partitionKey'::text),
(content -> 'sortKey'::text))
```

ONGRES

Would it scale like DynamoDB?

- Scaling is essentially linear with the number of shards (partitions)
- Almost all (permitted) operations are single-partition, and the issuer knows which partition to be directed to:
  `hash(primaryKey) -> partition`
- Scan is essentially a composition of Query commands, potentially out-of-order.
- Architecture is complex, needing request routers, metadata servers for partition -> server placement, re-sharding…
- But would allow, theoretically,
  Postgres to scale like DynamoDB!

ONGRES

Because, after all...

ONGRES

DynamoDB is “just” an HTTP application backed by MySQL!

DynamoDB is this very elegant system of highly-available replication, millisecond latencies, Paxos distributed state machines, etc. Then at the very bottom of it all there's a big pile of MySQL instances. Plus some ugly JNI/C++ code allowing the Java parts to come in through a side door of the MySQL interface, bypassing most of the query optimizer (since none of the queries are complex) and hitting InnoDB almost directly.

https://news.ycombinator.com/item?id=13173927

ONGRES

Stay tuned.
Coming soon....

ONGRES

Stay tuned.
Coming soon....

Postgres scaling like DynamoDB benchmark!

Follow @ahachete

ONGRES

Questions?

ONGRES