ROCKSET

# Rockset's Aggregator-Leaf-Tailer Architecture for SQL on semi structured data

High Performance Transaction Systems HPTS 2019

# About Me

- CTO and Co-Founder of Rockset
- RocksDB @ Facebook
- Hadoop File System @ Yahoo
- HBase & Hive




# Overview of the Talk

I. Changing Landscape of Data Analytics
II. Overview of Rockset's Aggregator Leaf Tailer (ALT) Architecture
III. Smart Schemas for SQL
IV. Converged Indexing powers SQL
V. Cloud Scaling Architecture

# Where has data processing been?


2006-2012
Batch Processing
Optimized for efficiency


2012-2018
Real-time Processing
Optimized for data latency


2018-
Operational Analytics
Optimized for: data latency, query latency & QPS

# 1. Where is data analytics going? Live decisions on data

- Complex queries
  - no more key-values please
- Low latency queries
  - milli-sec and seconds
- High QPS queries
  - thousands of queries per second
- Low data latency
  - less than a few seconds
  - mixed types

**Analytics on the fly**

Operational Analytics

# What is Rockset

**Serverless search and analytics service** (SaaS)

for building real-time apps and dashboards

without ETL or pipelines.

# The Aggregator Leaf Tailer (ALT) Architecture


https://rockset.com/blog/aggregator-leaf-tailer-an-architecture-for-live-analytics-on-event-streams/

# Key Benefits of ALT

1. **Makes queries fast**: Indexing all data at time of ingest
2. **Runs complex queries on the fly:** Distributed aggregator tier that scales compute and memory resources independently
3. **It's cost-effective:** Tailer, Leaf, and Aggregator can be independently scaled up and down
4. **Optimizes read & writes in isolation:** CQRS ensures that database writes do not impact queries

# Key Components of ALT

1. Converged Indexing
2. Smart Schemas
3. Cloud Native Architecture

# Converged Indexing

# What and Why converged indexing?

**What?** *Rockset stores every column of every document in a row-based index, column-based index, AND an inverted index.*

**Why?**

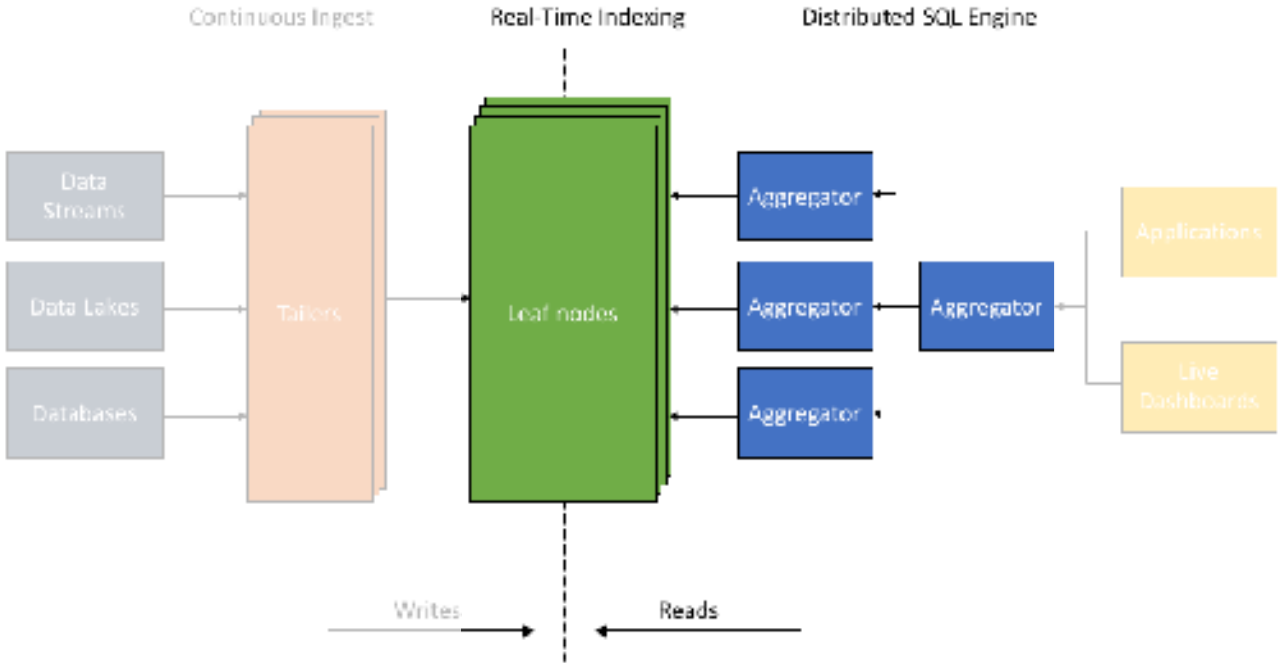
- No need to configure and maintain indexes
- No slow queries because of missing indexes

INDEX ALL

# How does converged indexing fit into ALT?

**The leaf** houses Rockset's converged indexes- column, row, AND inverted indexes

The optimizer can pick the index for the fastest query, enhancing the performance of **aggregators**.



# Under the Hood: Converged Indexing

- Columnar and inverted indexes in the same system
- Built on top of key-value store abstraction
- Each document maps to many key-value pairs

```
<doc 0>
{
  "name": "Igor"
}

<doc 1>
{
  "name": "Dhruba"
}
```

| Key | Value | |
|---|---|---|
| R.0.name | Igor | Row Store |
| R.1.name | Dhruba | |
| C.name.0 | Igor | Column Store |
| C.name.1 | Dhruba | |
| S.name.Dhruba.1 | | Inverted index |
| S.name.Igor.0 | | |

# Inverted Indexing for point lookups

- For each value, store documents containing that value (posting list)
- Quickly retrieve a list of document IDs that match a predicate

```
<doc 0>
{
  "name": "Igor",
  "interests": ["databases", "snowboarding"],
  "last_active": 2019/3/15
}

<doc 1>
{
  "name": "Dhruba",
  "interests": ["cars", "databases"],
  "last_active": 2019/3/22
}
```

"name"

| | |
|---|---|
| Dhruba | 1 |
| Igor | 0 |

"interests"

| | |
|---|---|
| databases | 0.0; 1.1 |
| cars | 1.0 |
| snowboarding | 0.1 |

"last_active"

| | |
|---|---|
| 2019/3/15 | 0 |
| 2019/3/22 | 1 |

# Columnar Indexing for aggregations

- Store each column separately
- Great compression
- Only fetch columns the query needs

```
<doc 0>
{
  "name": "Igor",
  "interests": ["databases", "snowboarding"],
  "last_active": 2019/3/15
}

<doc 1>
{
  "name": "Dhruba",
  "interests": ["cars", "databases"],
  "last_active": 2019/3/22
}
```

"name"

| | |
|---|---|
| 0 | Igor |
| 1 | Dhruba |

"interests"

| | |
|---|---|
| 0.0 | databases |
| 0.1 | snowboarding |
| 1.0 | cars |
| 1.1 | databases |

"last_active"

| | |
|---|---|
| 0 | 2019/3/15 |
| 1 | 2019/3/22 |

# The Power of the Query Optimizer

- Low latency for both highly selective queries and large scans
- Optimizer picks between columnar store or inverted index

```
SELECT *
FROM search_logs
WHERE keyword = 'hpts'
AND locale = 'en'
```

**Inverted index**
**(for highly selective queries)**

```
SELECT keyword, count(*)
FROM search_logs
GROUP BY keyword
ORDER BY count(*) DESC
```

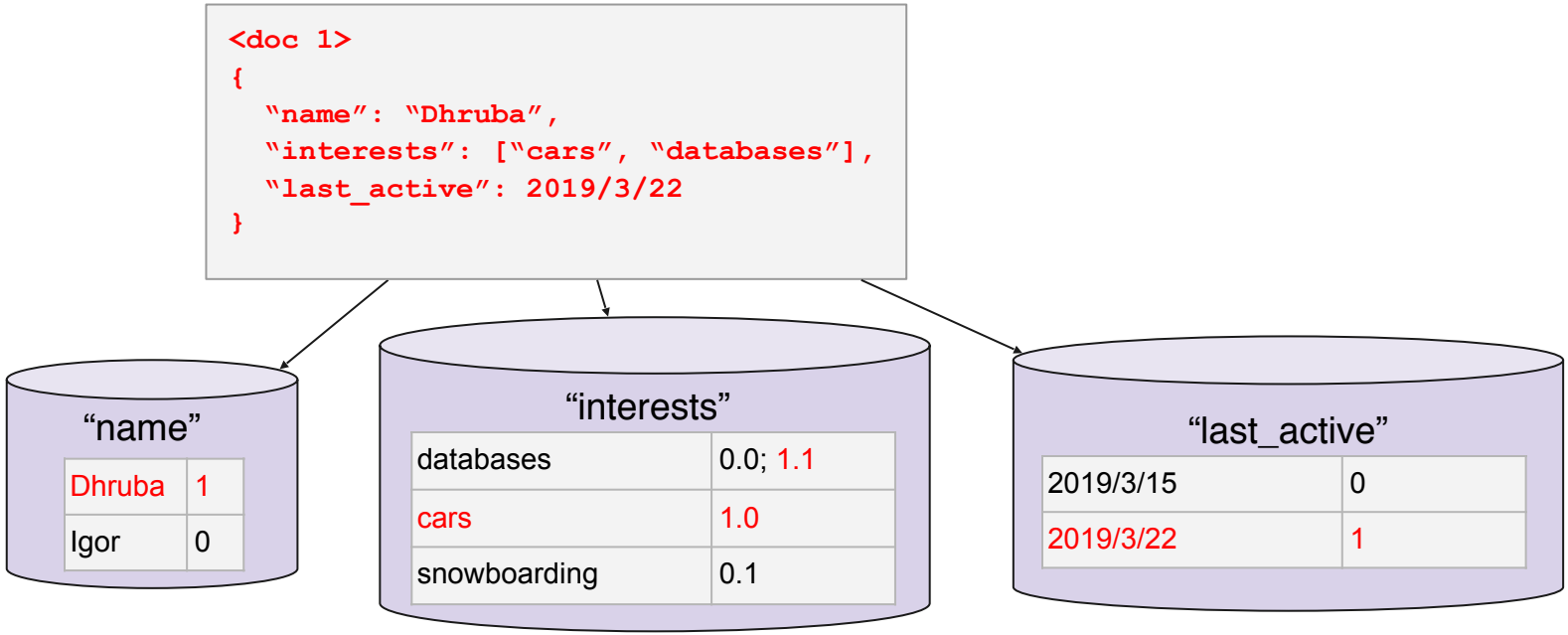
**Columnar store**
**(for large scans)**

# Challenges with Converged Indexing

- Maintaining multiple indexes adversely impacts write throughput
- Challenge 1: one new record = multiple servers updates
  - Requires consensus coordination between servers
- Challenge 2: one new field = multiple random writes
  - Requires increased disk I/O

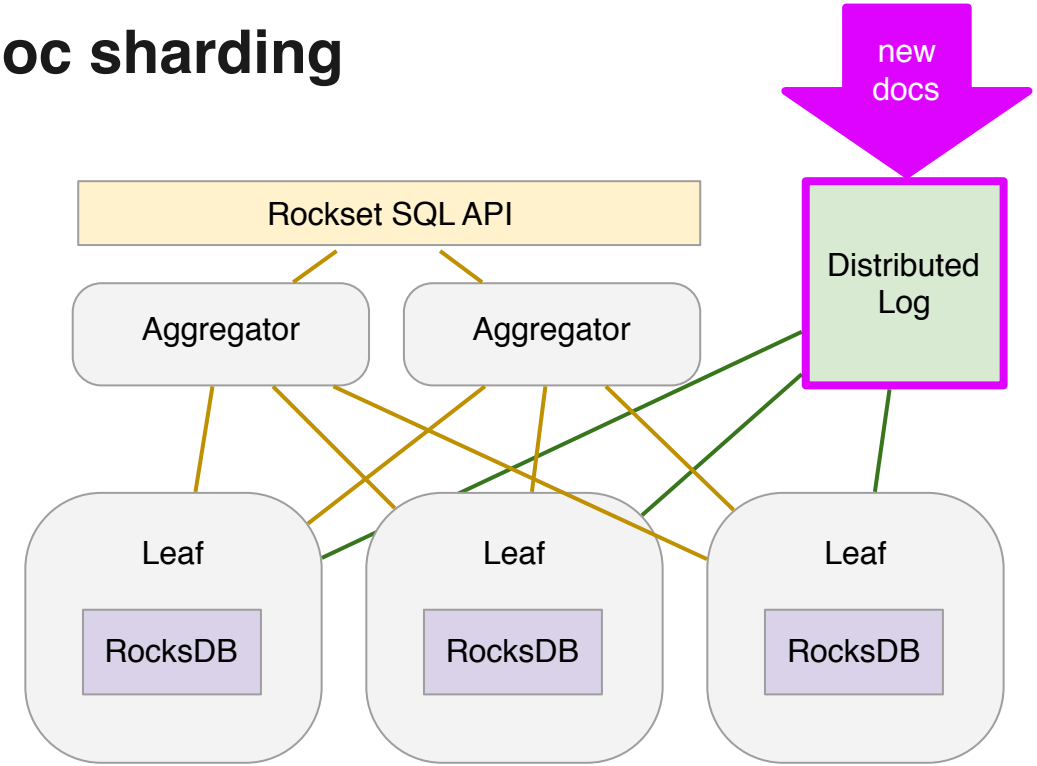
# Challenge 1: one new record = multiple servers updates

- In a traditional database with term sharding and **n** indexes, one write incurs updates to **n** different indexes on n servers
- Requires a distributed transaction (paxos, raft) between **n** servers



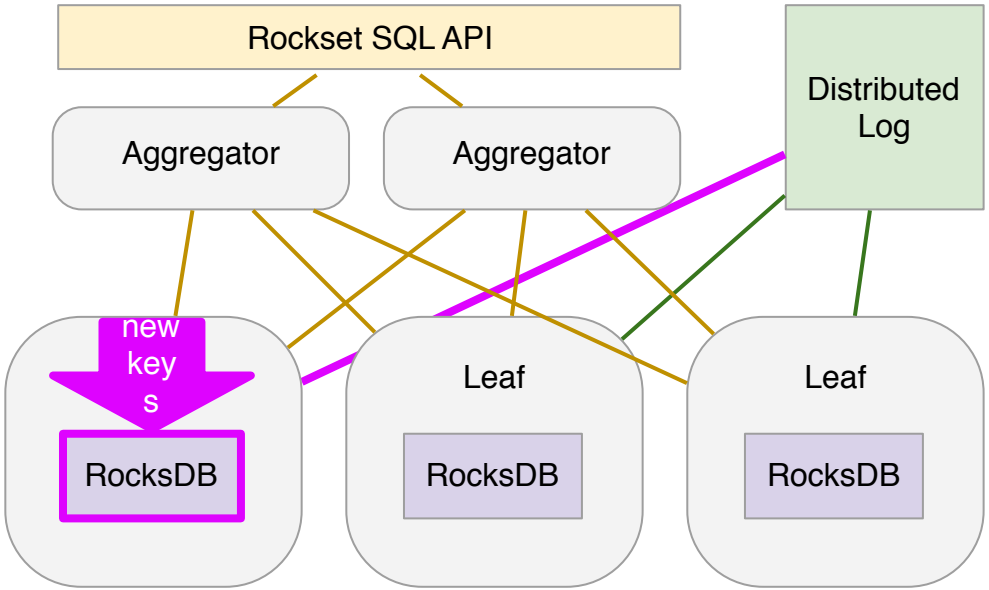
# Addressing challenge 1: doc sharding

- Updates are durably-buffered to a distributed log



# Addressing challenge 1: doc sharding

- Updates are durably buffered to a distributed log
- Leafs tail only documents in the shards they are responsible for
- Doc sharding means all new keys will only affect a single shard/leaf



# What is document sharding?

Let's say you were running a search on restaurant reviews in the area….

Traditional distributed databases

Restaurant Data  Review Data

Optimized for query throughput

Rockset

Restaurant & Review Data

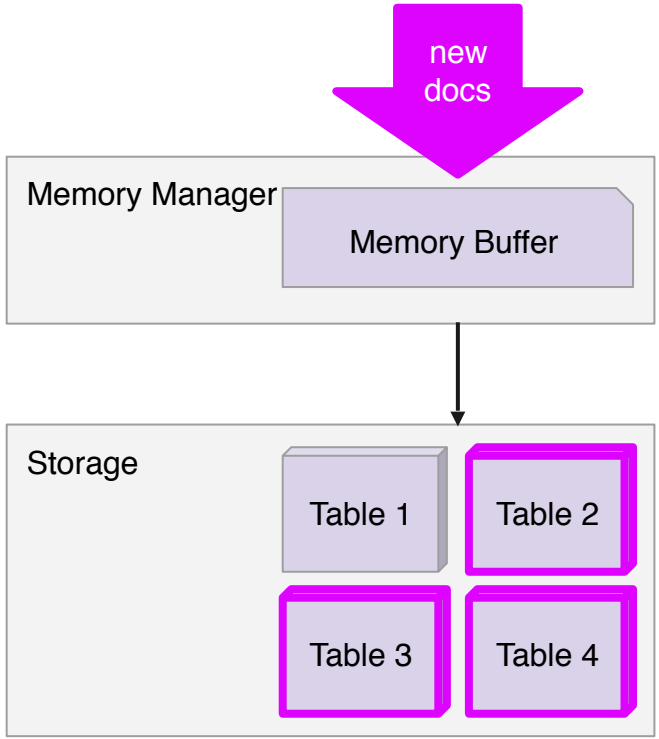
Optimized for query latency

Big data analytics requires you to run complex queries at interactive speed

First, optimize for latency and then optimize for throughput

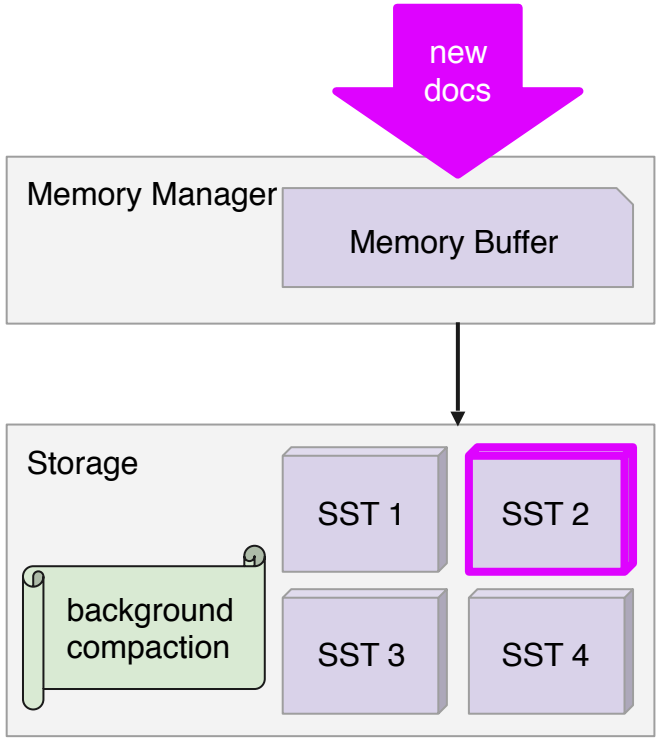
# Challenge 2: one new doc = multiple random writes

- Traditional systems use B-tree storage structure
- Keys are sorted across tables
- A single record update would incur writes to multiple different tables



# Addressing challenge 2: RocksDB LSM

- RocksDB uses log-structured merge-tree (LSM)
- Multiple record updates accumulate in memory and written into a single SST file
- Keys are sorted between SST files via compaction in a background process
- Multiple index updates from multiple docs result in one write to storage



# Smart Schema SQL

# What and Why smart schemas?

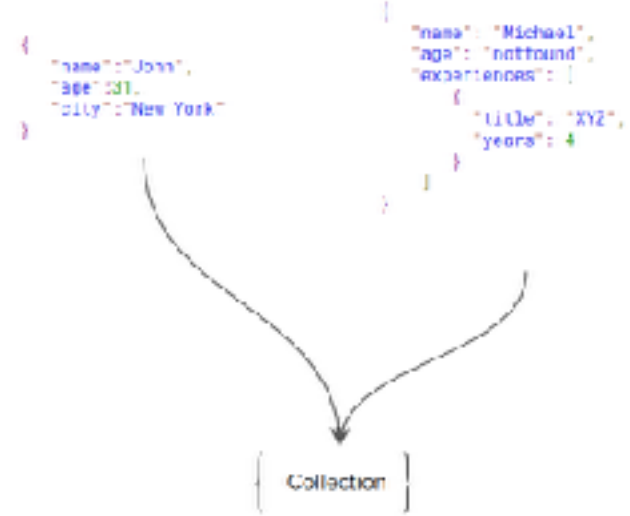
**What?** *Automatic generation of a schema based on the exact fields and types present at the time of ingest. There is no need to ingest the data with a predefined schema (ie: schemaless ingestion).*

**Why?** *Avoid data pipelines that cause data latency*

- Semi-structured data is complex and messy
- Ingest any semi-structured data (similar to NoSQL)
- Make it easy to read the data (using standard SQL)

# Under the Hood: Smart Schemas

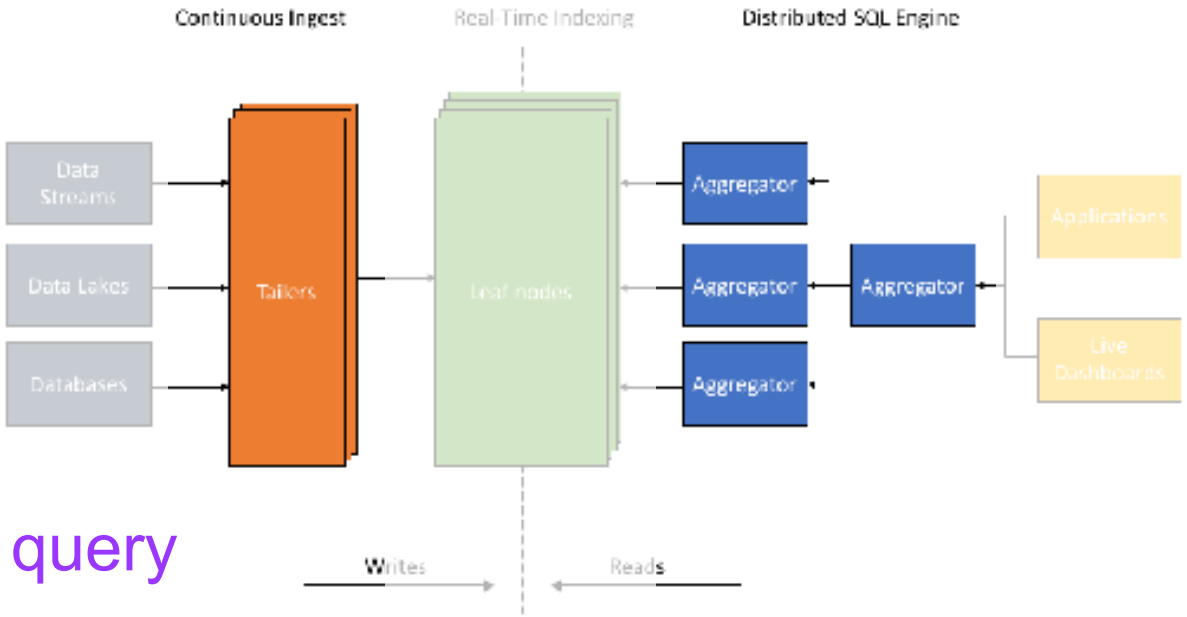
- Type information stored with values, not “columns”
- Strongly types queries on dynamically typed fields
- Designed for nested semi-structured data



# How do smart schemas fit into ALT?

**Tailers** ingest data without predefined schemas (ie: schemaless ingest)

**Aggregators** use the schema to make queries fast

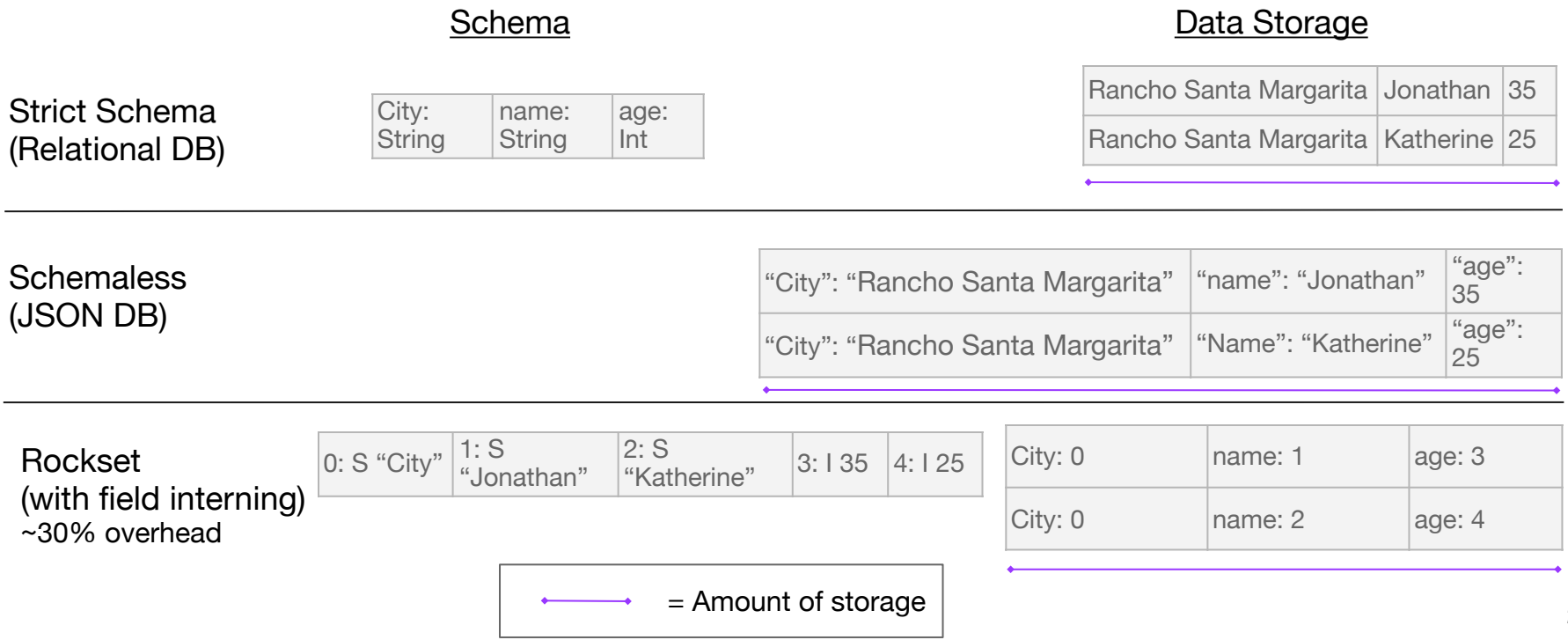



schema binding at query time

# Challenges with Smart Schemas

- Challenge 1: Additional CPU usage for processing queries
- Challenge 2: Requires increased disk space to store types

# We use field interning to reduce the space required to store schemas



# We use type hoisting to reduce CPU required to run queries


Rows

Strict schema

| 1 | 10 | 7 | 4 | 5 |
|---|---|---|---|---|
| a | b | c | d | e |

Schemaless

| I 1 | I 10 | I 7 | I 4 | I 5 |
|---|---|---|---|---|
| S a | S b | S c | S d | S e |

Rockset (with type hoisting)

| I | 1 | 10 | 7 | 4 | 5 |
|---|---|---|---|---|---|
| S | a | b | c | d | e |

**Rockset query performance is almost on par with strict schema systems**

# Cloud Scaling Architecture

# Key insight into economics of cloud

- Cost of 1 cpu for 100 minutes == Cost of 100 cpu for 1 minute!!

# Key insight into economics of cloud

- Cost of 1 cpu for 100 minutes == Cost of 100 cpu for 1 minute!!
  - Without cloud: statically provision for peak demand
  - With cloud: dynamically provision for current demand

# Key insight into economics of cloud

- Cost of 1 cpu for 100 minutes == Cost of 100 cpu for 1 minute!!
  - Without cloud: statically provision for peak demand
  - With cloud: dynamically provision for current demand
- Goal: scale up and down storage as needed to achieve desired performance

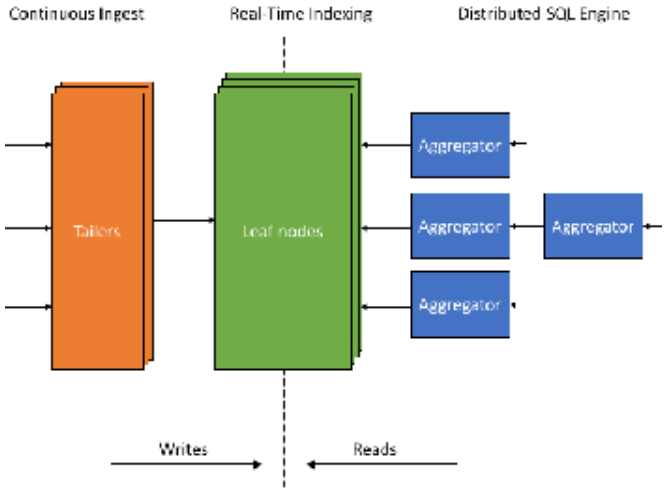
If your query is slow, the challenge is in the software you are using.

# What and Why cloud autoscaling?

**What?** Each tailer, leaf, or aggregator can be independently scaled up and down as needed. Tailers scale when more data to ingest, leaves scale when data size grows and aggregators scale when query volume increases

**Why?**

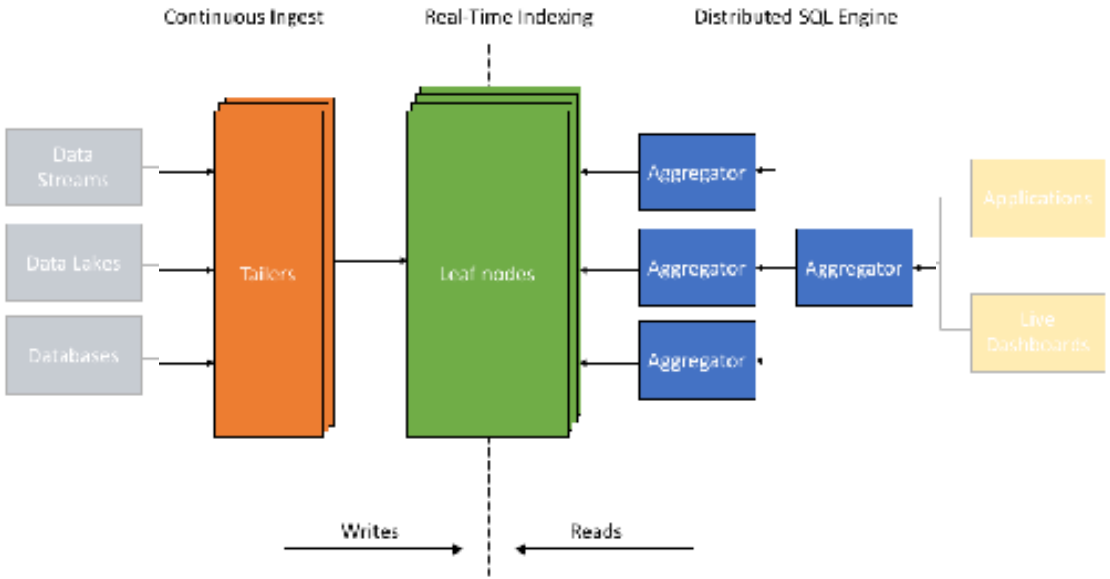
- No provisioning
- Pay for usage
- No need to provision for peak capacity



# We are going to focus on scaling out the leaf

Scaling tailers and aggregators are easy because they are stateless

Scaling leaf nodes is tricky because they are stateful



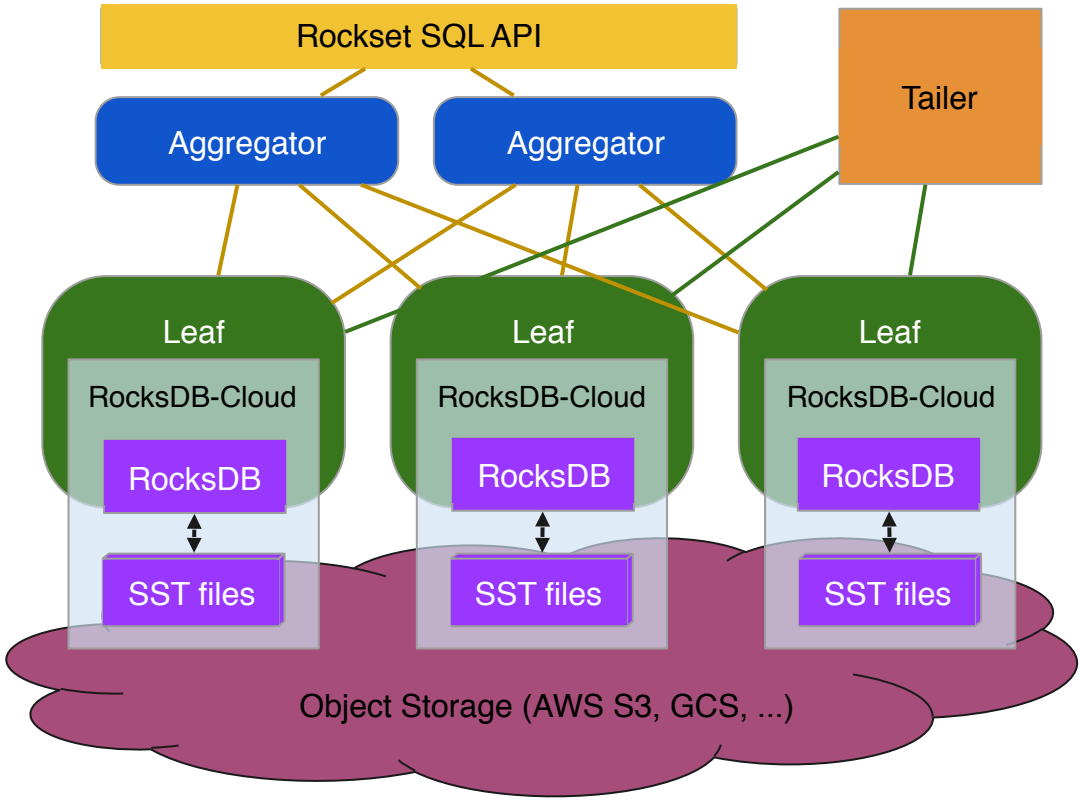
# Quotation from Dr David Dewitt

(The End of ”Shared-Nothing”)


David J. DeWitt
MIT

Willis Lang
Microsoft Jim Gray Systems Lab

# Scale down leaves: Use durability of cloud storage

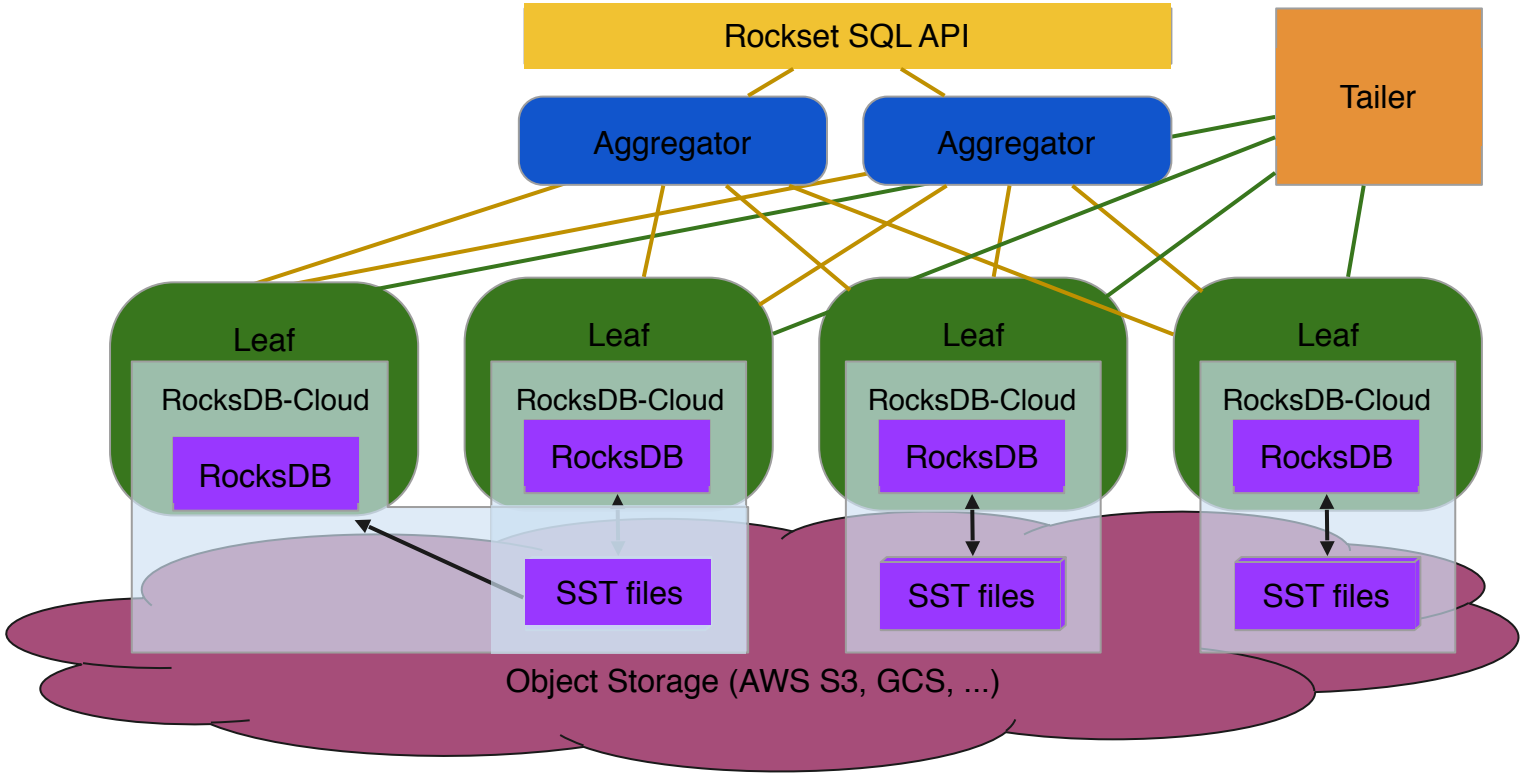



SST files are uploaded to the cloud storage

Durability is achieved by keeping data in cloud storage

No data loss even if all leaf servers crash and burn!

# Scale up new leaves: use zero-copy clones of rocksdb-cloud




No performance impact on existing leaf servers

# Recap: ALT Architecture provides SQL on json

No need to manage indexes with converged indexing

No need to pre-define schemas with smart schemas

No need to provision servers with cloud scaling architecture

# Summary: Operational Analytics

I. Index rather than partition-and-scan
II. Aggregator Leaf Tailer (ALT) Architecture rather than Lambda architecture
III. Optimized for low data latency, low query latency, and high QPS


https://rockset.com/blog/operational-analytics-what-every-software-engineer-should-know/

# Thank you

Dhruba Borthakur
dhruba@rockset.com
@dhruba_rocks

Check us out: http://rockset.com