ROCKSET: The design and implementation of a data system for low-latency queries for search and analytics

Dhruba Borthakur, Igor Canadi

Rockset

[ROCKSET]

Speakers

Dhruba

- CTO and Co-Founder at Rockset
- RocksDB at Facebook
- Hadoop File System
- HBase

Igor

- Software Engineer at Rockset
- RocksDB at Facebook
- GraphQL

Overview

1. Converged indexing
2. Query execution over a distributed index
3. High-throughput index updates
4. Scaling in the cloud

Motivation

Ease of use

- Minimize configuration
- Can connect to any data source
- Real time writes

High performance

- Low latency queries
- High throughput writes

Converged indexing

Columnar storage

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

Columnar storage

- Store each column separately
- Great compression
- Only fetch columns 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

Columnar storage

Advantages

- Cost effective
- Narrow queries, wide tables
- Scan queries
- Analytical queries

Disadvantages

- High write latency
- High minimum read latency
- Not suitable for online applications

Search indexing

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

Search indexing

- 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

Search indexing

Advantages

- High selectivity queries
- Low latency queries
- Suitable for online applications

Disadvantages

- Slower analytical queries

Converged indexing

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

Converged indexing

- Columnar and search 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		**Search index**
S.name.Igor.0		

Converged indexing - queries

- Fast analytical queries + fast search queries
- Optimizer picks between columnar store or search index

Converged indexing - queries

- Fast analytical queries + fast search queries
- Optimizer picks between columnar store or search index

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

Search index

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

Columnar store

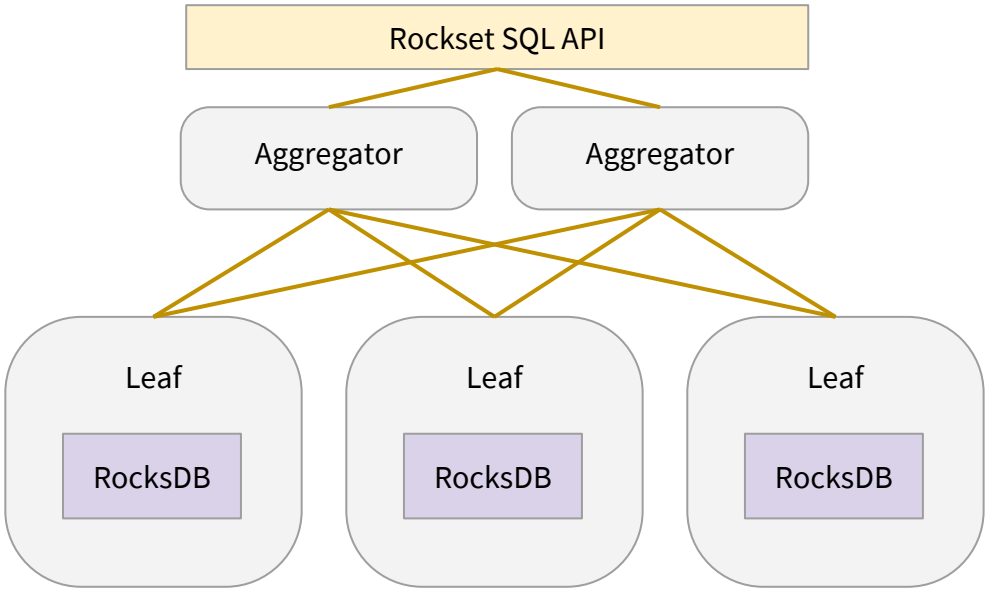
Rockset - data system built on converged indexing

- All fields indexed
- SQL
- Document model, schemaless
- Real-time writes, updates and deletes
- Cloud service

Query execution over a distributed index

Rockset query architecture

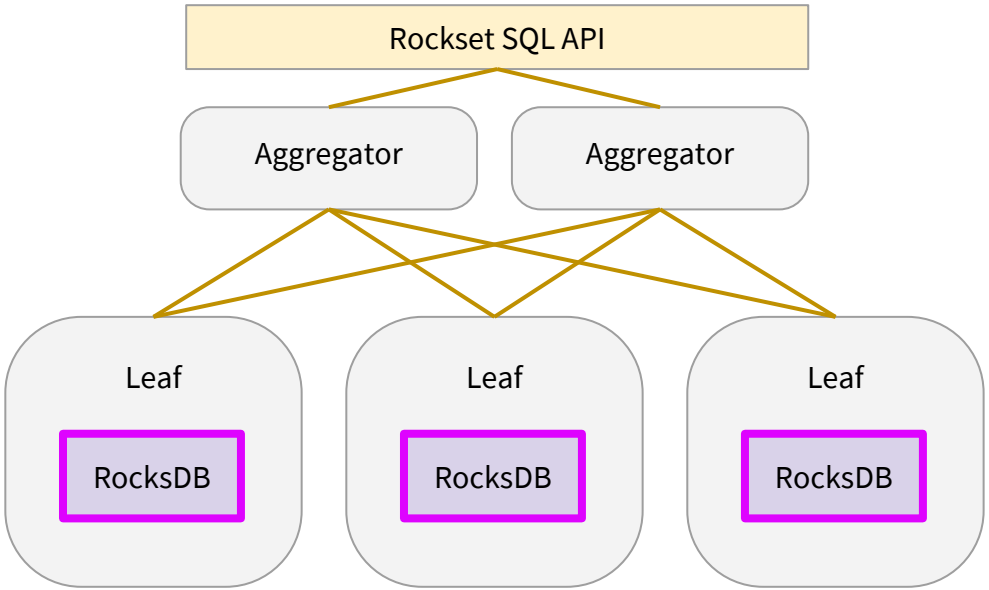
- API layer
- Aggregators
- Leaves



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

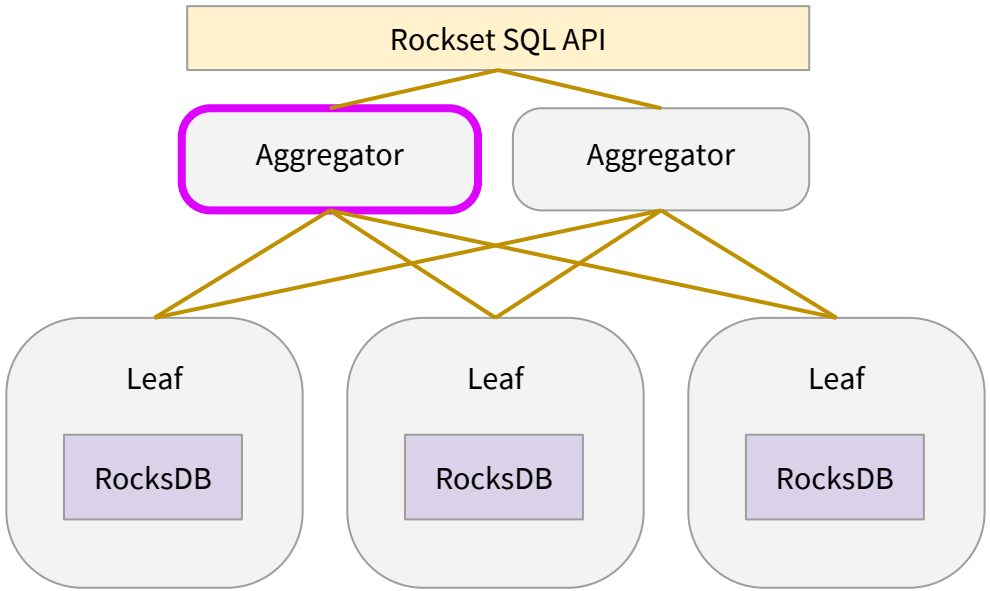
Rockset query architecture

- Converged index sharded across RocksDB instances
- Document-based sharding
- Query hits all shards



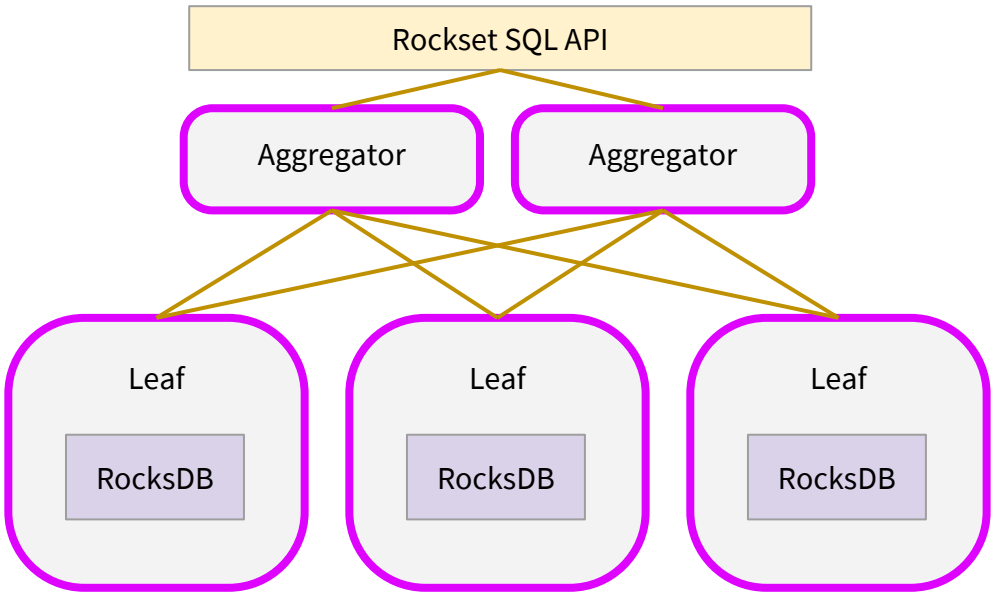
Rockset query architecture

- Aggregator receives the query
- Parsing, compilation, optimization
- Produces operator DAG



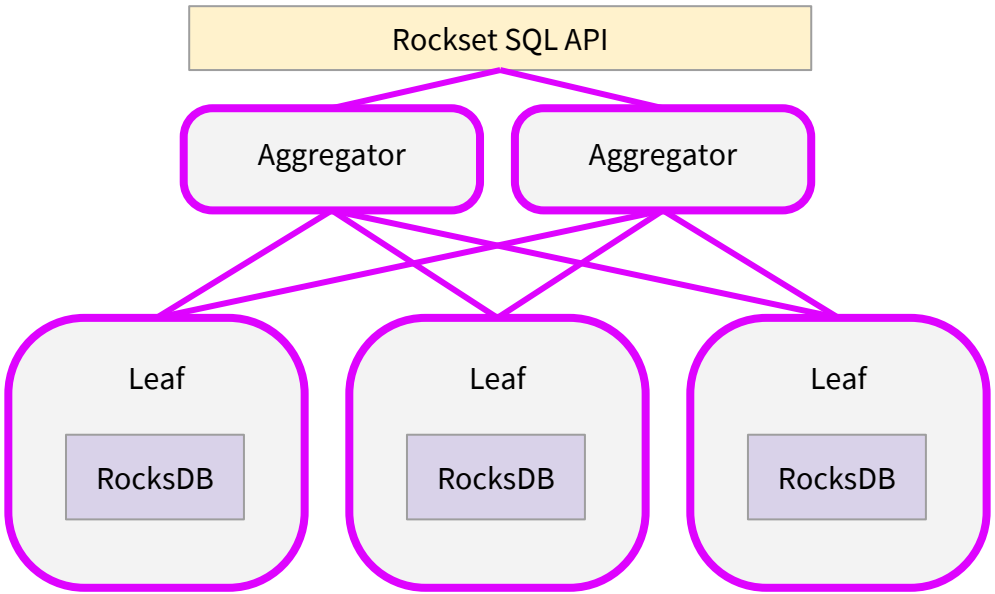
Rockset query architecture

- Set of instructions for each node:
  - Predecessors
  - Operators
  - Successors



Rockset query architecture

- Set of instructions for each node:
  - Predecessors
  - Operators
  - Successors
- On each node:
  - Wait for predecessors
  - Connect to successors
  - Data starts flowing bottom-up



High-throughput index updates

Challenges with updating multiple indexes

- Maintaining multiple indexes adversely impacts write throughput

Challenges with updating multiple indexes

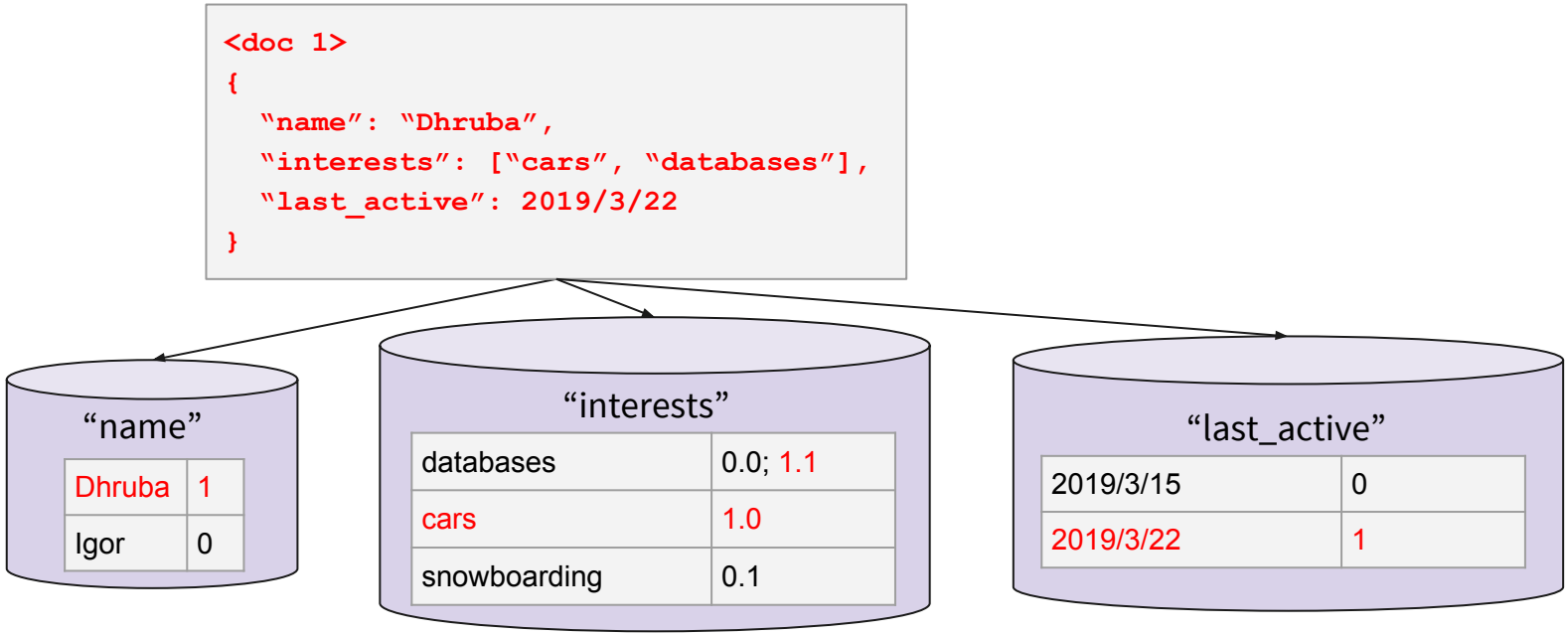
- Maintaining multiple indexes adversely impacts write throughput
- Challenge 1: one new record = multiple servers updates
    - Requires consensus coordination between servers

Challenges with updating multiple indexes

- 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

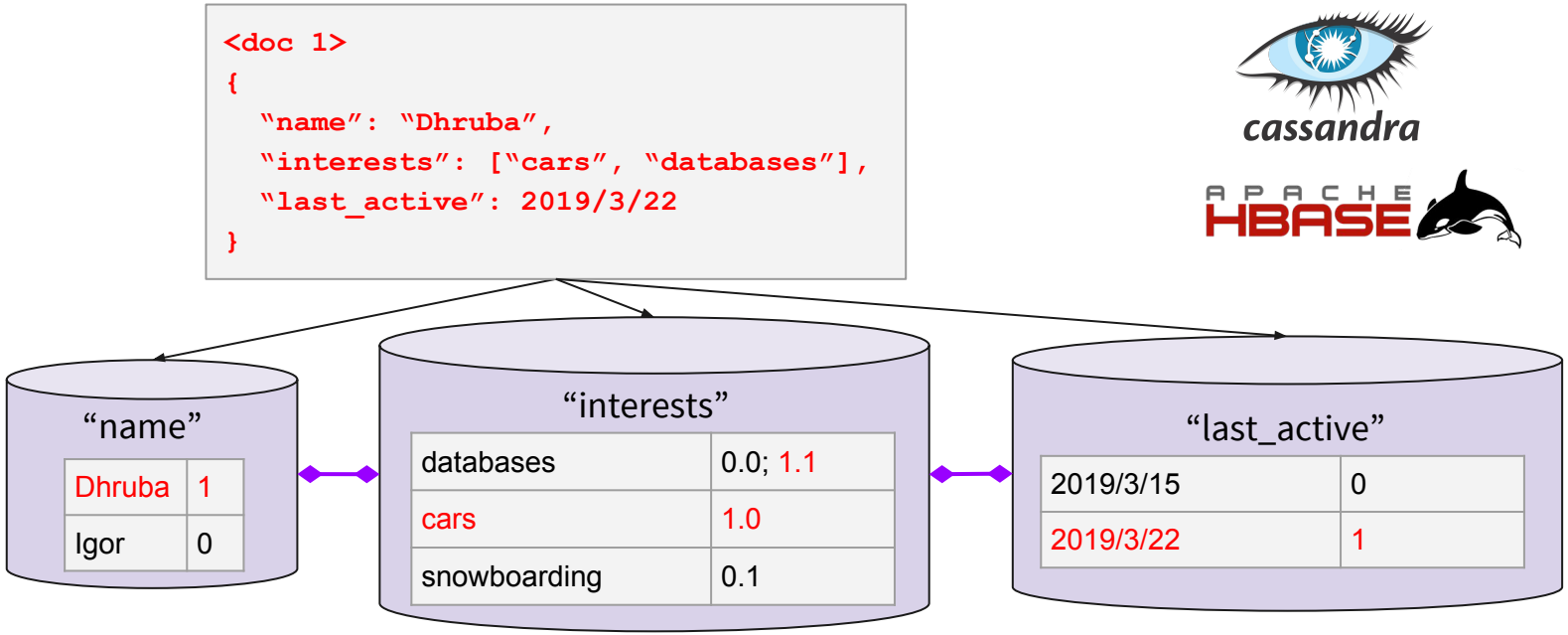
Challenge 1: one new record = multiple servers updates

- In a traditional database with term sharding and **n** indexes, one write incurs updates to **n** different indexes on n servers



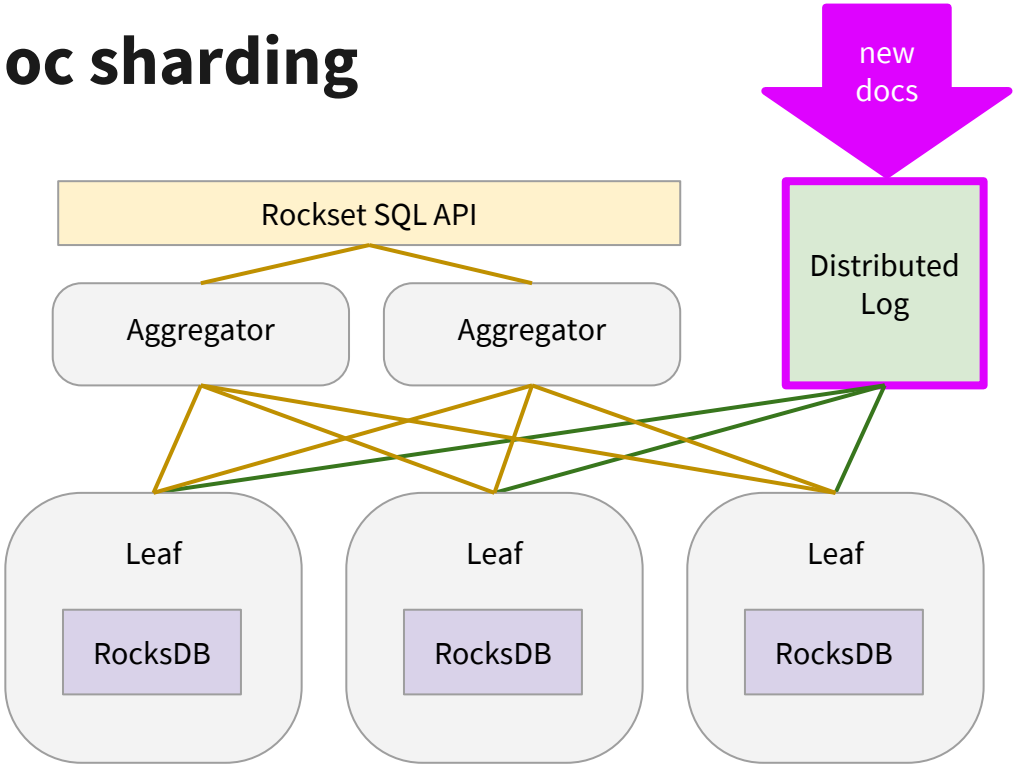
Challenge 1: one new record = multiple servers updates

- 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



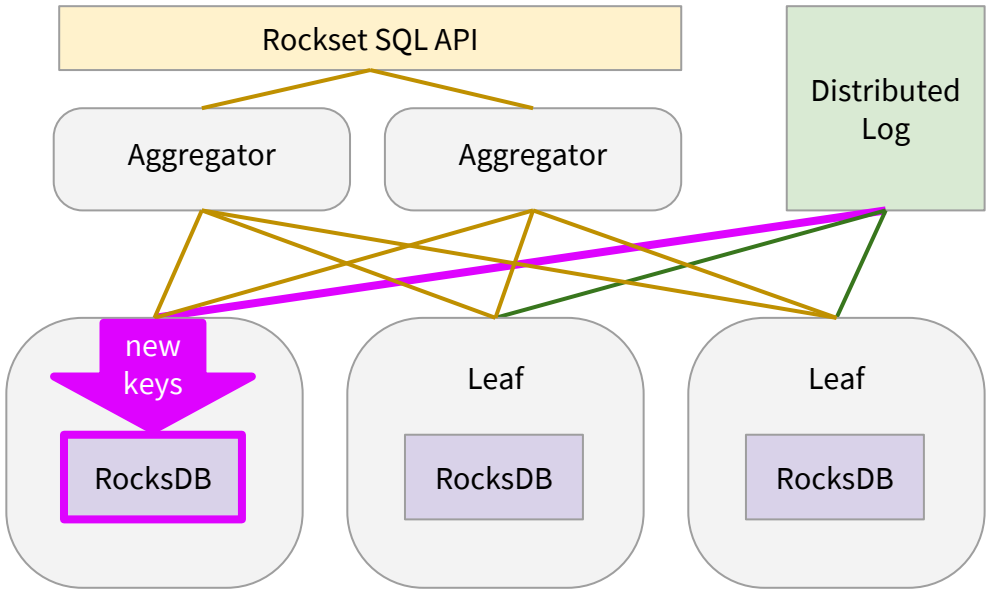
Addressing challenge 1: doc sharding

- Updates are durably-buffered to a distributed log



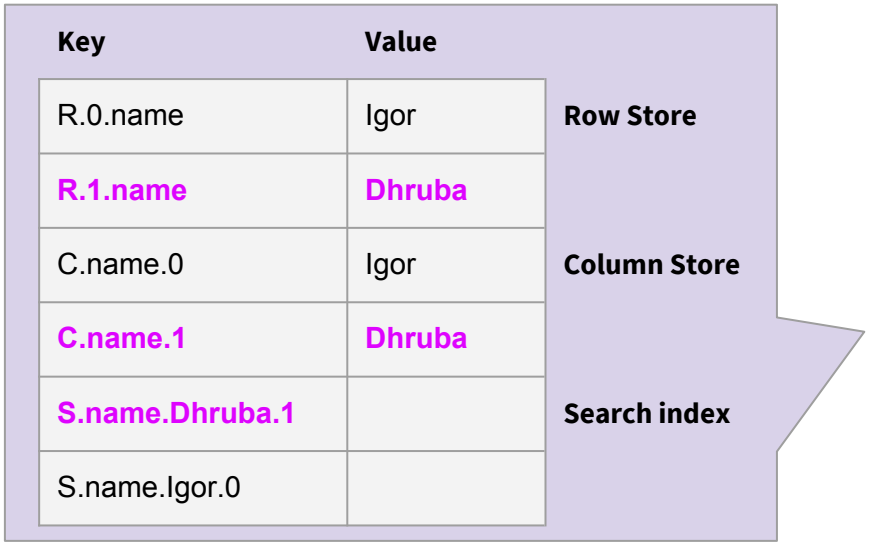
Addressing challenge 1: doc sharding

- 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



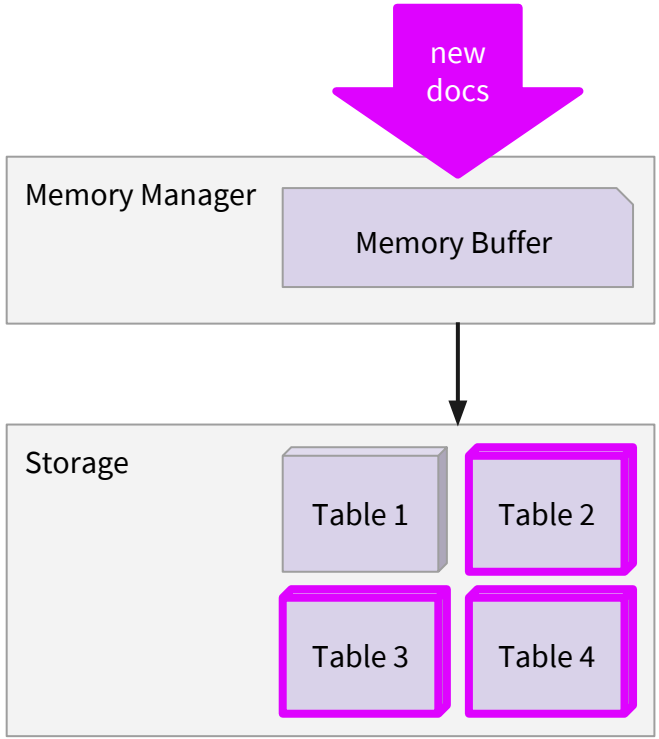
Challenge 2: one new doc = multiple random writes

- One doc becomes multiple keys in the index



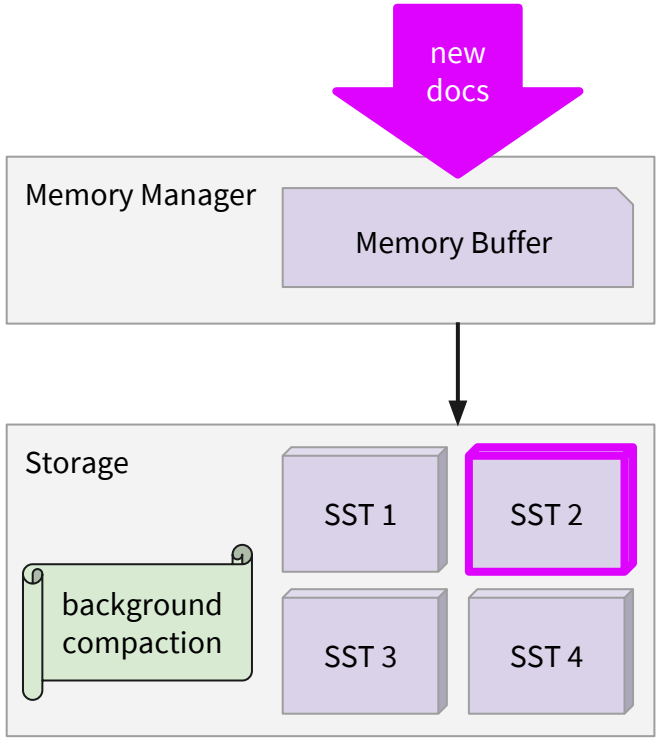
Challenge 2: one new doc = multiple random writes

- 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



Scaling in the cloud

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

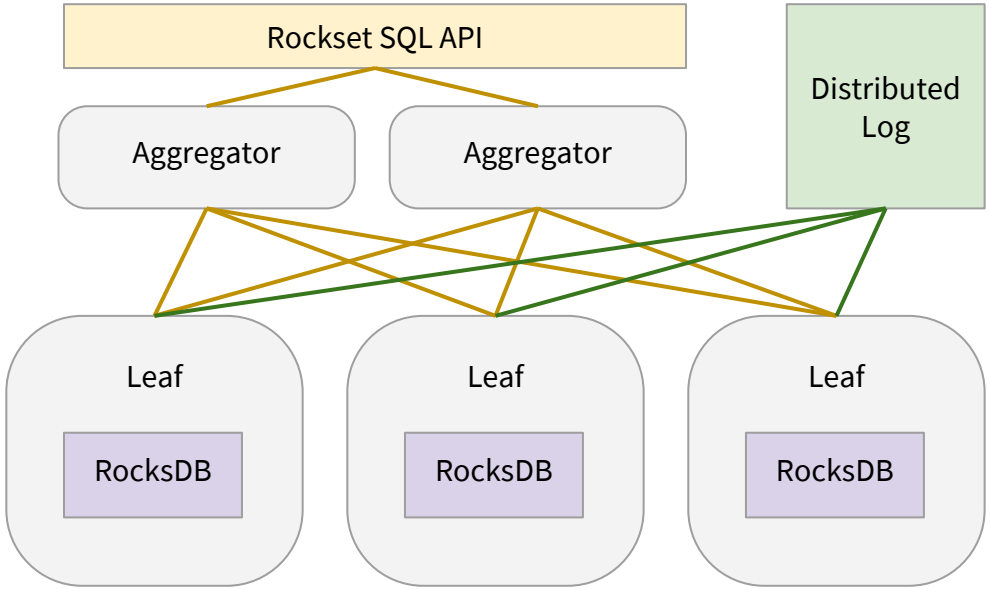
- Cost of 1 cpu for 100 minutes == Cost of 100 cpu for 1 minute!!
  - Without cloud: statically provision for peak demand
  - With cloud: dynamically provision for current demand
- Goal: scale up and down storage as needed to achieve desired performance

Scheduling compute

- Kubernetes Horizontal Pod Autoscaler (HPA) to schedule compute pods across pool of nodes (AWS EC2 machines)
  - Using cpu and memory thresholds
- Custom scheduler to manage underlying nodes
  - Manage both demand (of workload) and supply (of hardware)
  - Sum CPU usage of all pods to spin up new nodes when needed
  - Aggressively shed nodes when not needed

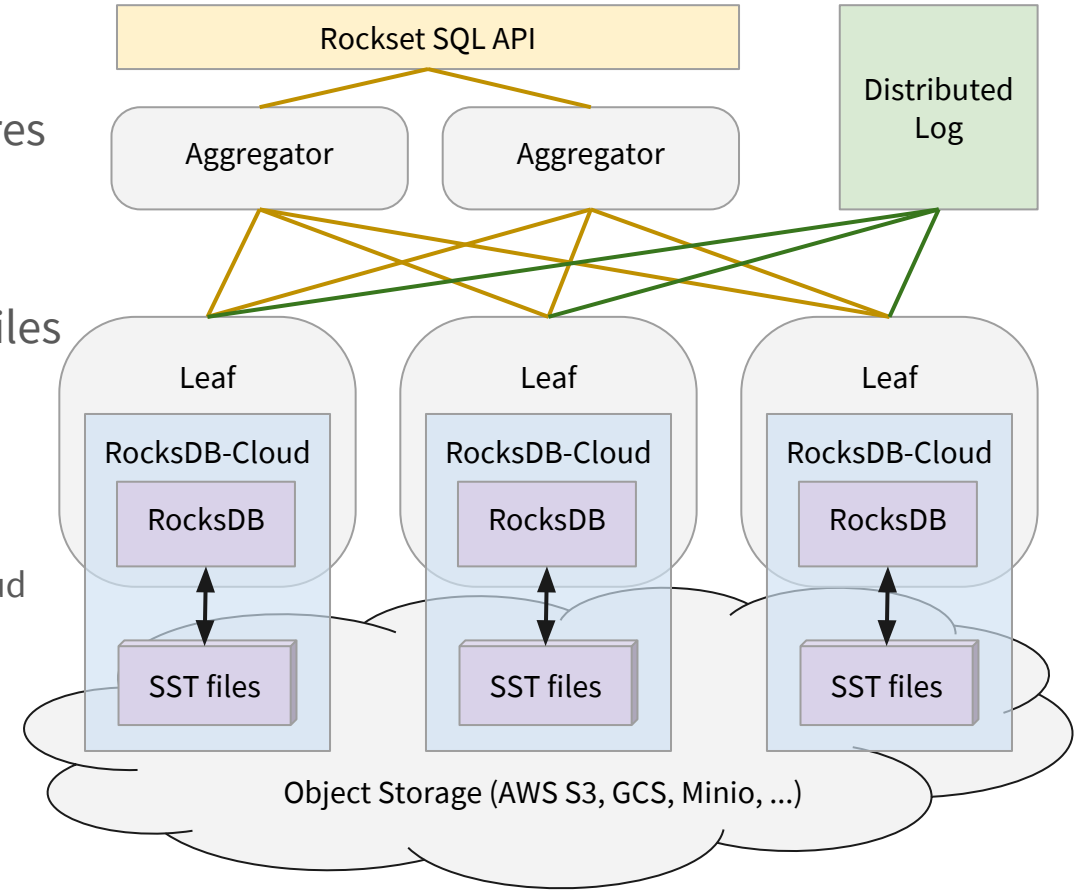
Scheduling storage

- Each leaf running RocksDB stores indices



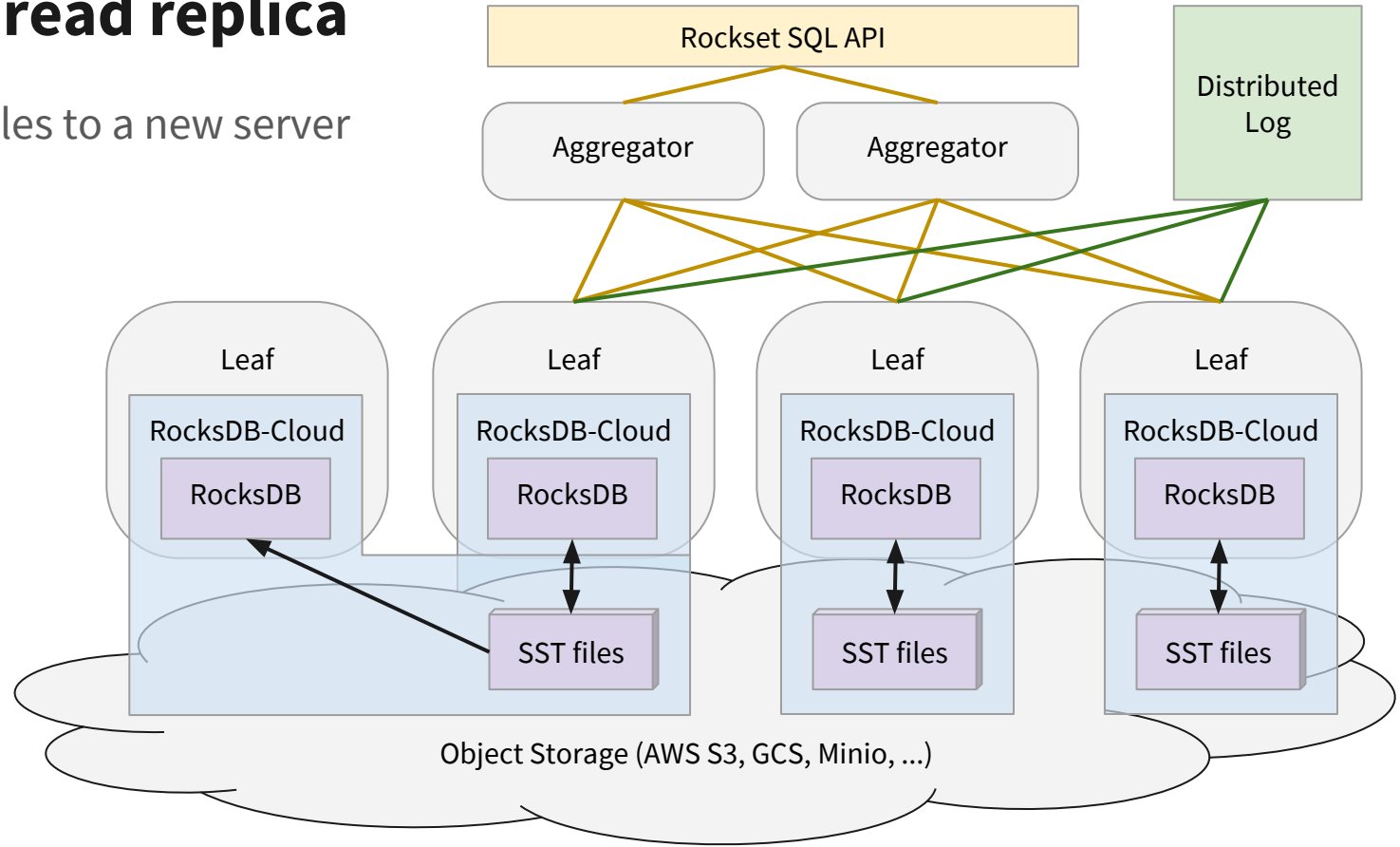
Scheduling storage

- Each leaf running RocksDB stores indices
- RocksDB-Cloud (open source) extends RocksDB to flush SST files to durable cloud storage
- RocksDB-Cloud = serverless storage
- http://github.com/rockset/rocksdb-cloud



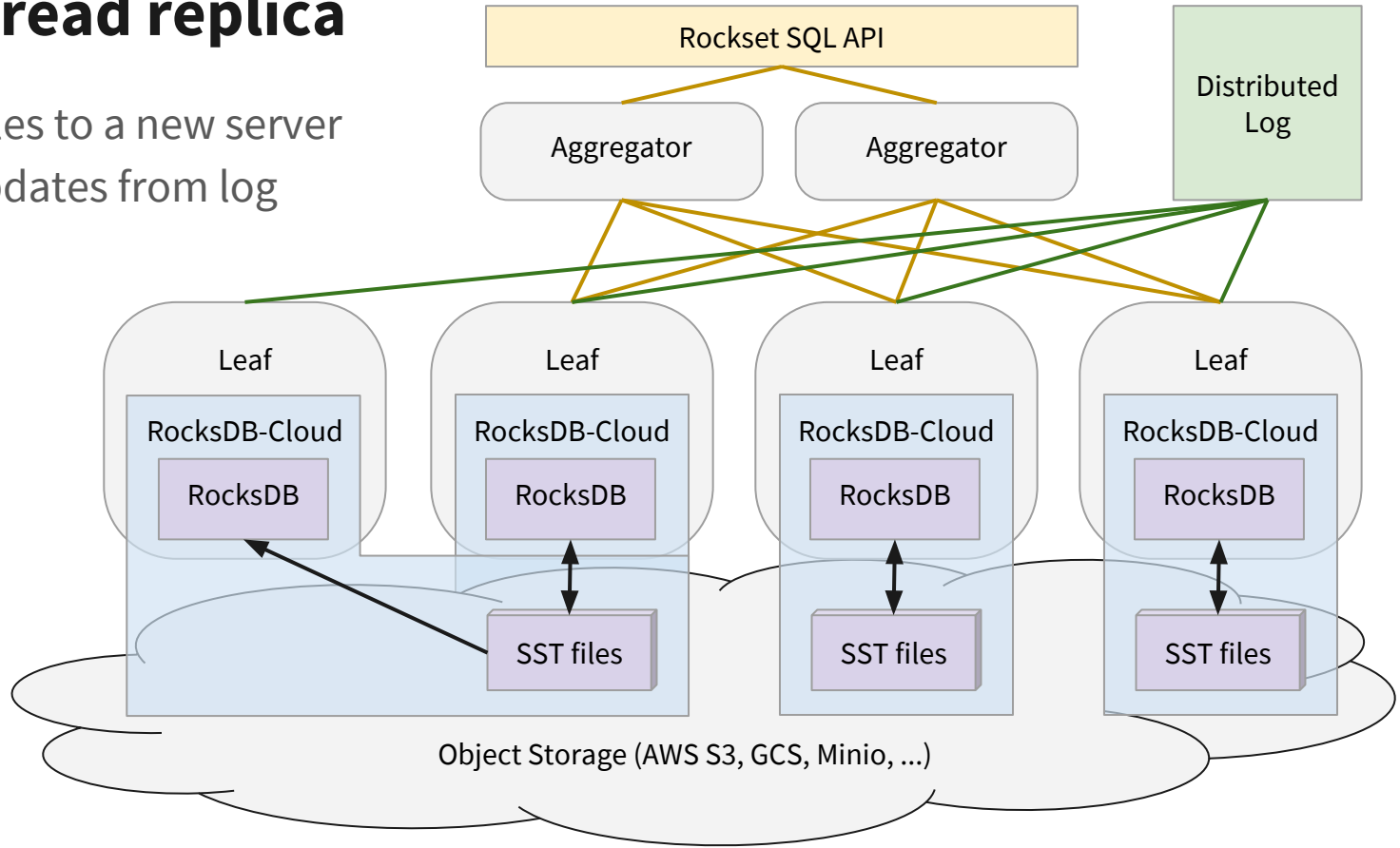
Zero-copy read replica

- Copy SST files to a new server



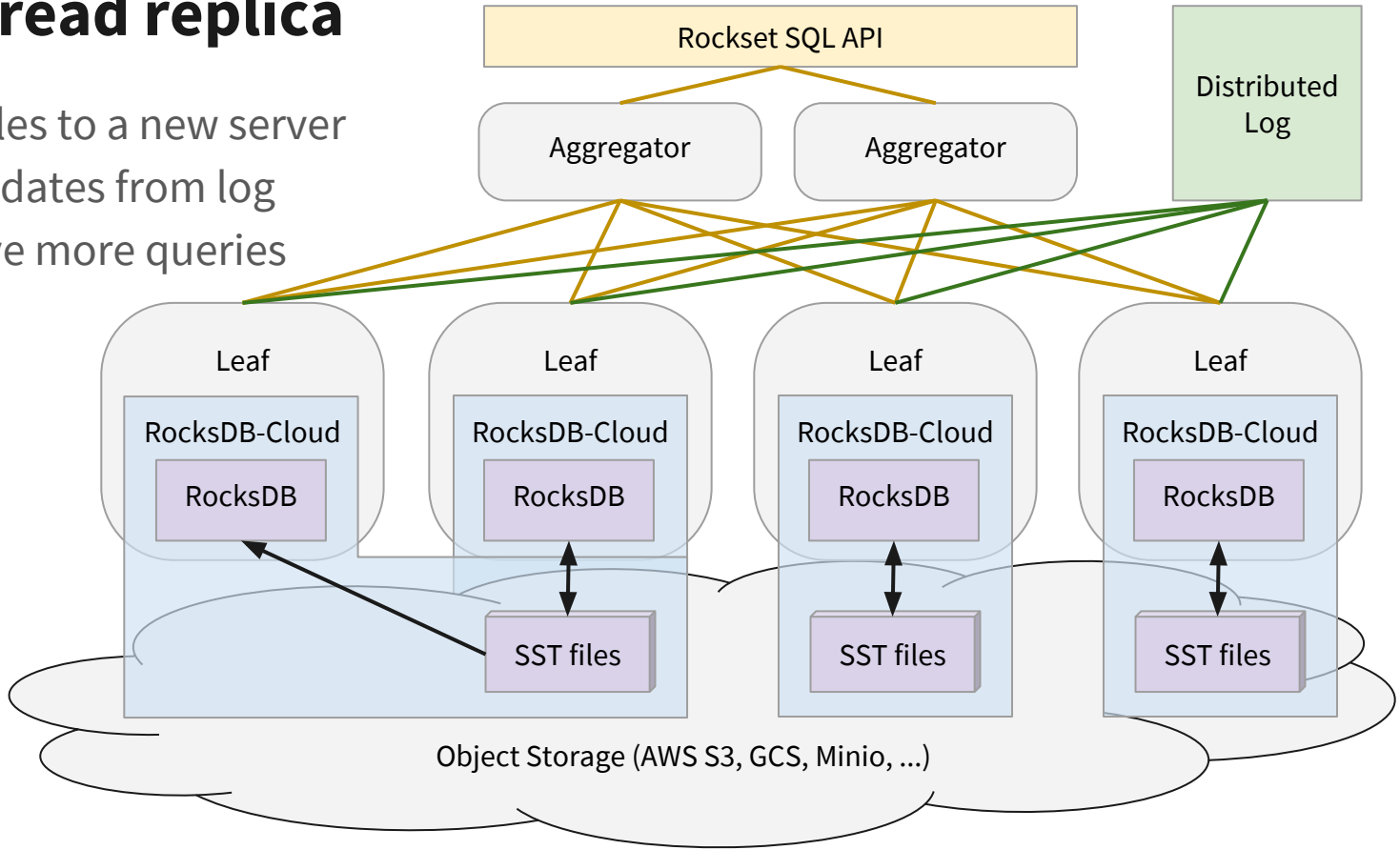
Zero-copy read replica

- Copt SST files to a new server
- Tail new updates from log

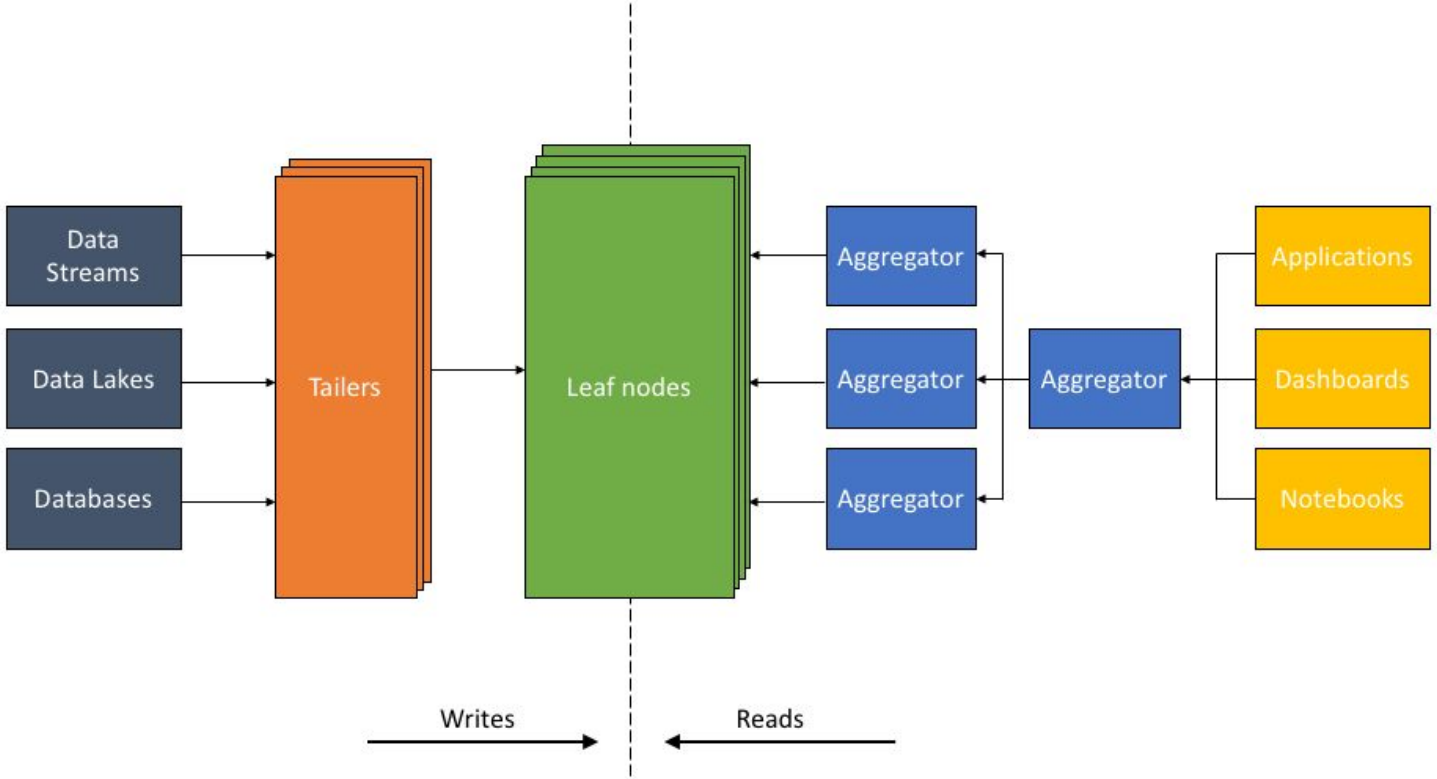


Zero-copy read replica

- Copy SST files to a new server
- Tail new updates from log
- Able to serve more queries



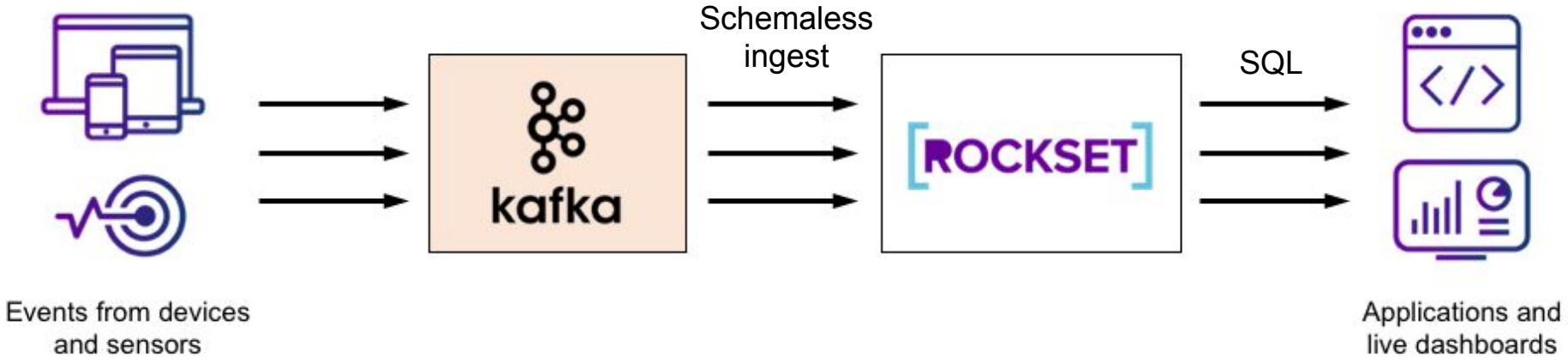
The Aggregator Leaf Tailer Architecture



ROCKSET

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

Event Analytics Using Rockset



- All fields and values indexed
- Low-latency queries
- Serving layer for online applications and live dashboards

Check it out: rockset.com

dhruba@rockset.com
igor@rockset.com

Thank you.

Rate today's session