THE SILENT ORCHESTRA:

The hidden choreography of distributed databases

Ahmad Alhour

October, 2025

Xpand Conference

ABOUT ME!

- Born and raised in Amman
- A proud PSUT alumnus ✨
- Living in Munich
- Father of 2!
- Life-long Martial Artist

- ~14 YOE in tech
- Staff Eng @ Grafana Labs
- Previously @ Shopify & HubSpot

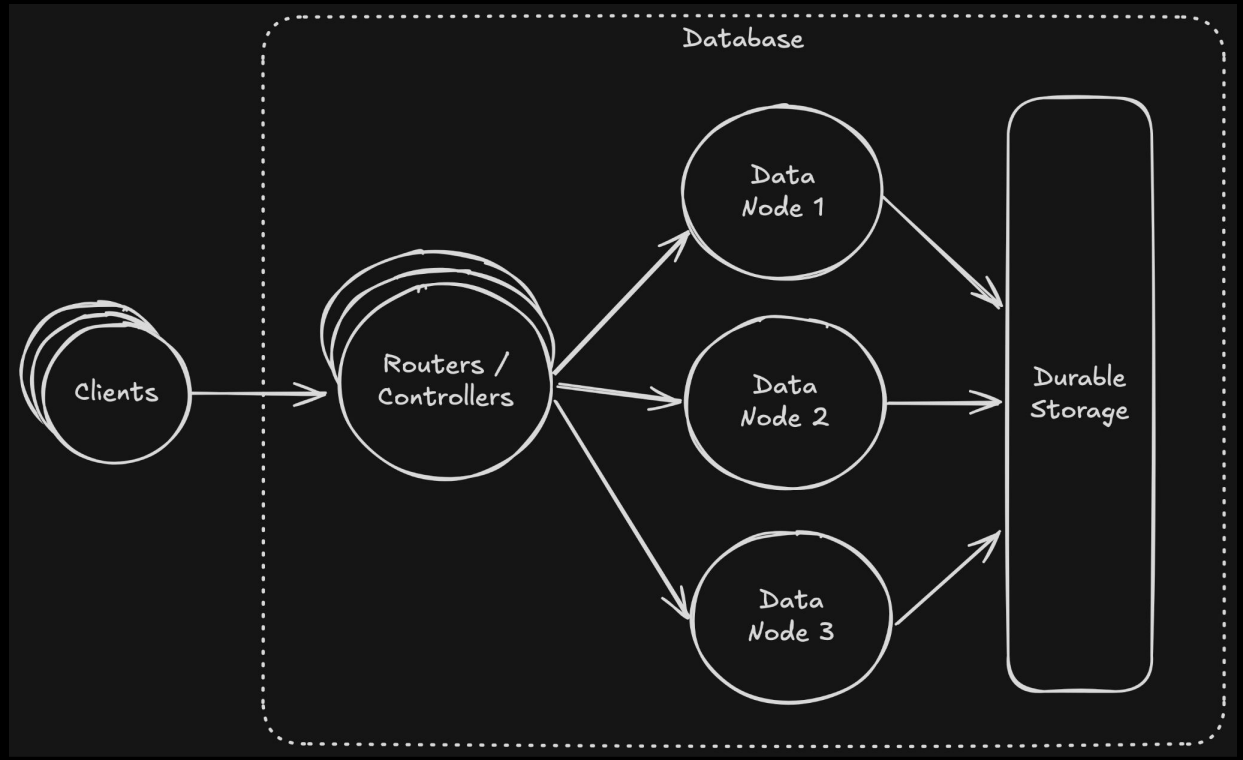
WHAT'S ON THE MENU TODAY?

- Example-driven
- 30 mins teaser
- POV of a Product Eng

PEEK BEHIND THE VEIL!

WHAT ARE DISTRIBUTED DATABASES?

A system of...
multiple machines...
working together...
and appearing as...
a single node...
to clients



USE CASES

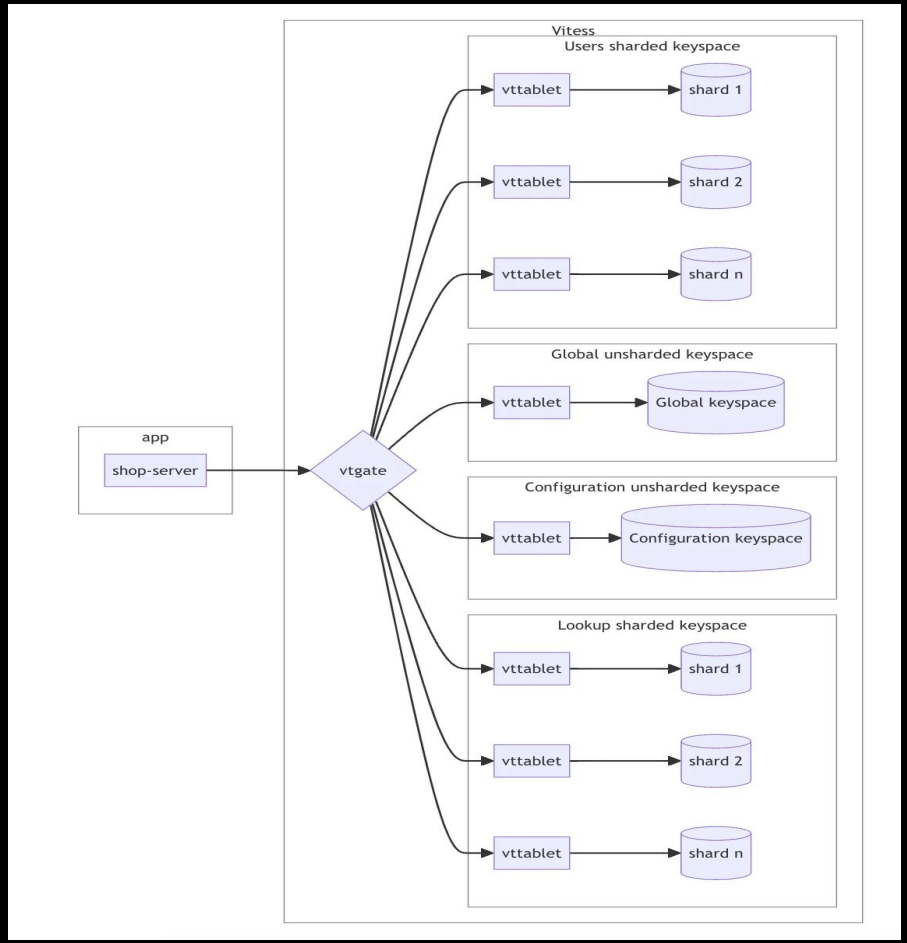
Use Case 1 - Shopify + Vitess

Setup:

- Vitess

Goals:

- Horizontally shard Shop App backend
- Minimize query latency
- Shard data per Store/User

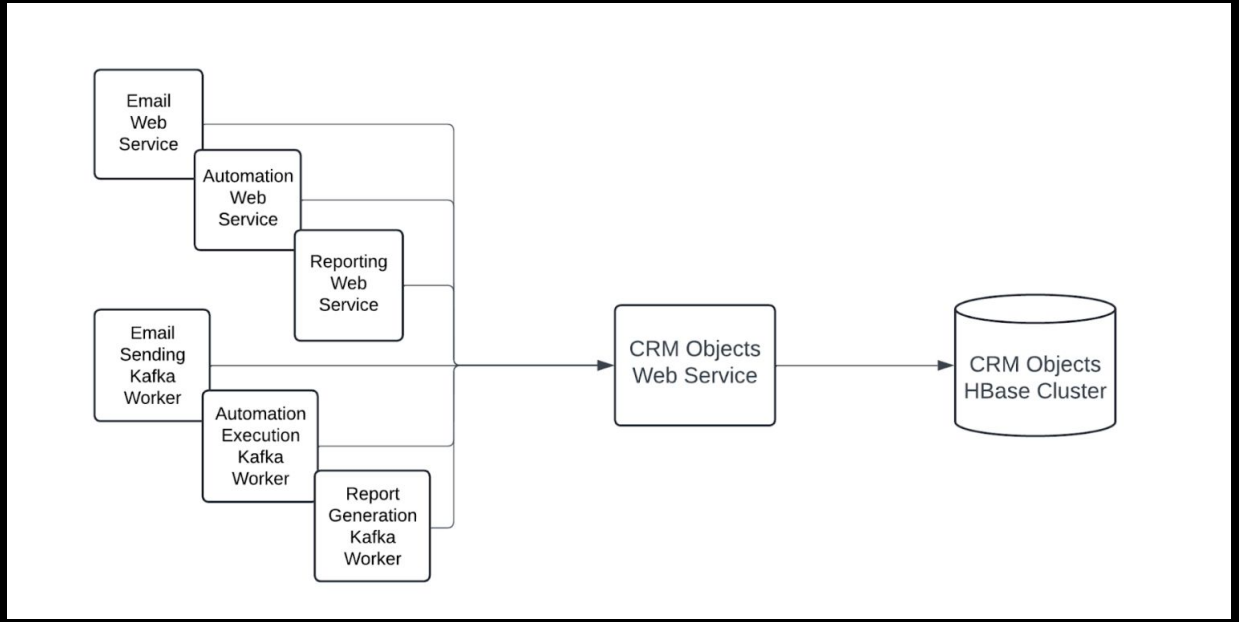


Source: https://shopify.engineering/horizontally-scaling-the-rails-backend-of-shop-app-with-vitess#

Use Case 2 - HubSpot + HBase

Setup:
- Apache HBase (OSS)
- Across 5 datacenters

Goals:
- Low-latency queries
- High throughput
- Terabytes of data
- Scales >25M req/sec



Source: https://product.hubspot.com/blog/hbase-share-resources

A CASE STUDY

Apache HBase

NoSQL distributed database

Based on Google's Bigtable paper

Written in Java ☕

Uses HDFS for storage

Used for real-time read/write access to Bigdata

Scales to

- millions of columns
- billions of rows
- petabytes of data
- millions of req/sec

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com
Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

1 Introduction

Over the last two and a half years we have designed, implemented, and deployed a distributed storage system for managing structured data at Google called Bigtable. Bigtable is designed to reliably scale to petabytes of data and thousands of machines. Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability. Bigtable is used by more than sixty Google products and projects, including Google Analytics, Google Finance, Orkut, Personalized Search, Writely, and Google Earth. These products use Bigtable for a variety of demanding workloads, which range from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users. The Bigtable clusters used by these products span a wide range of configurations, from a handful to thousands of servers, and store up to several hundred terabytes of data.

In many ways, Bigtable resembles a database: it shares many implementation strategies with databases. Parallel databases [14] and main-memory databases [13] have achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. Finally, Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

Section 2 describes the data model in more detail, and Section 3 provides an overview of the client API. Section 4 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 5 describes the fundamentals of the Bigtable implementation, and Section 6 describes some of the refinements that we made to improve Bigtable's performance. Section 7 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 8, and discuss some lessons we learned in designing and supporting Bigtable in Section 9. Finally, Section 10 describes related work, and Section 11 presents our conclusions.

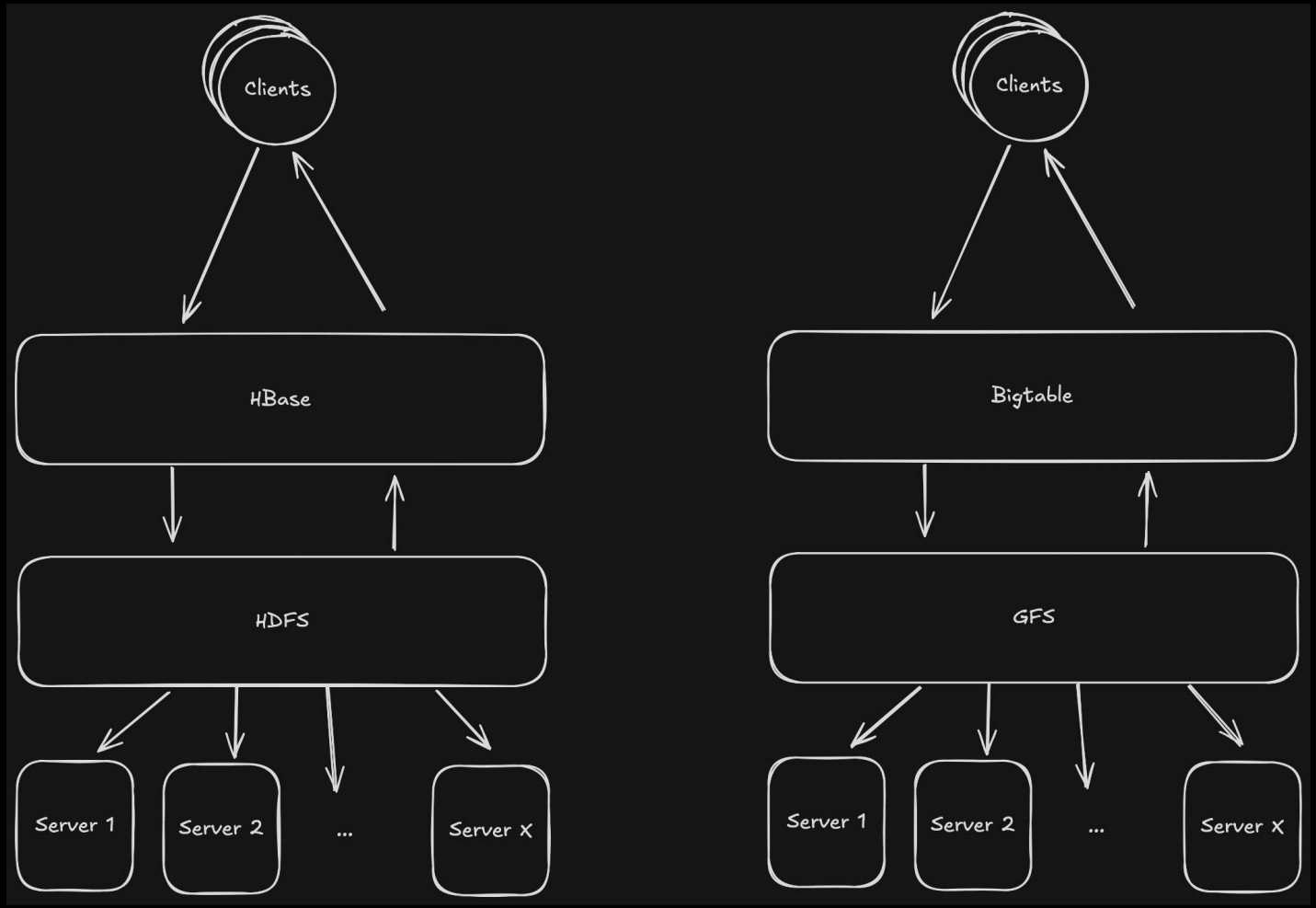
2 Data Model

A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

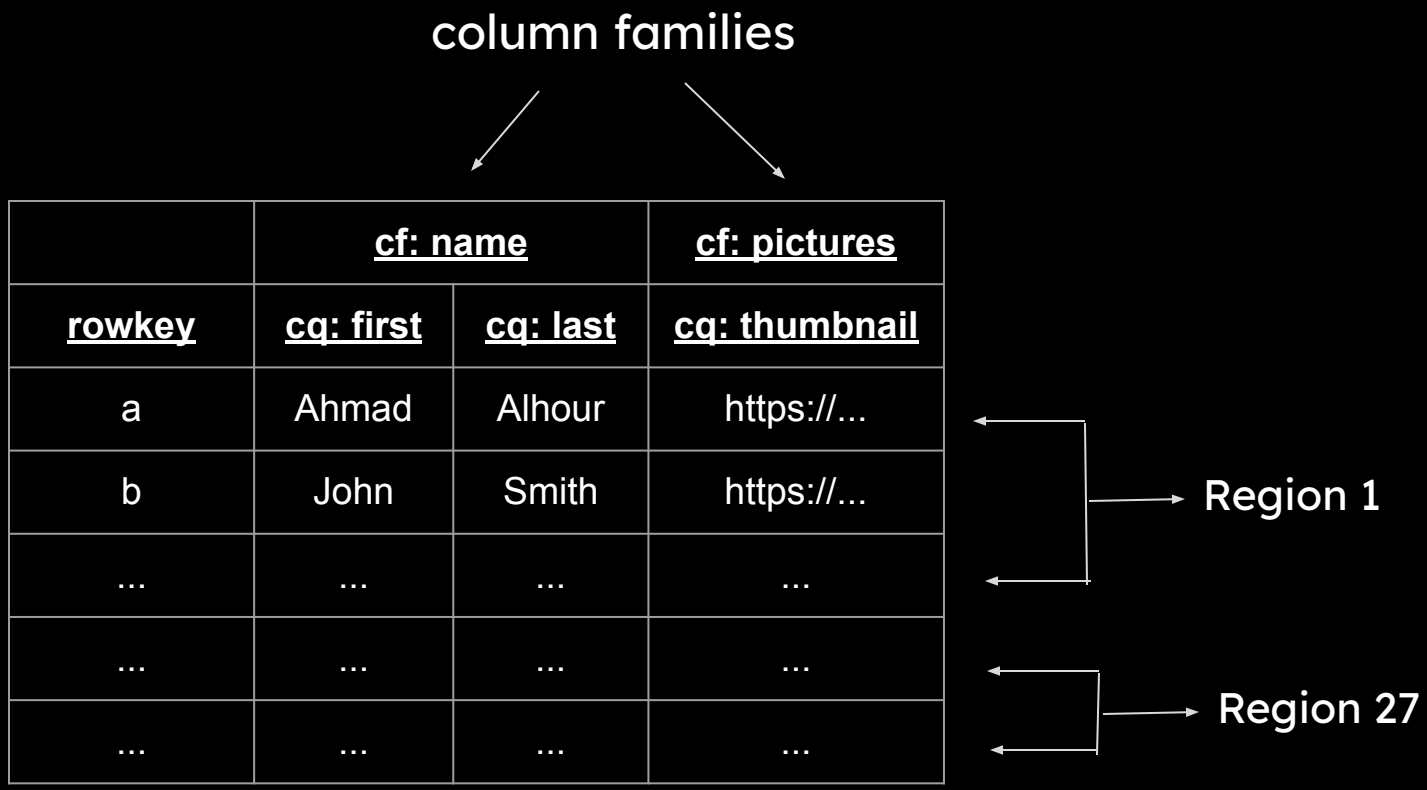
(row:string, column:string, time:int64) → string

To appear in OSDI 2006

Logical Model

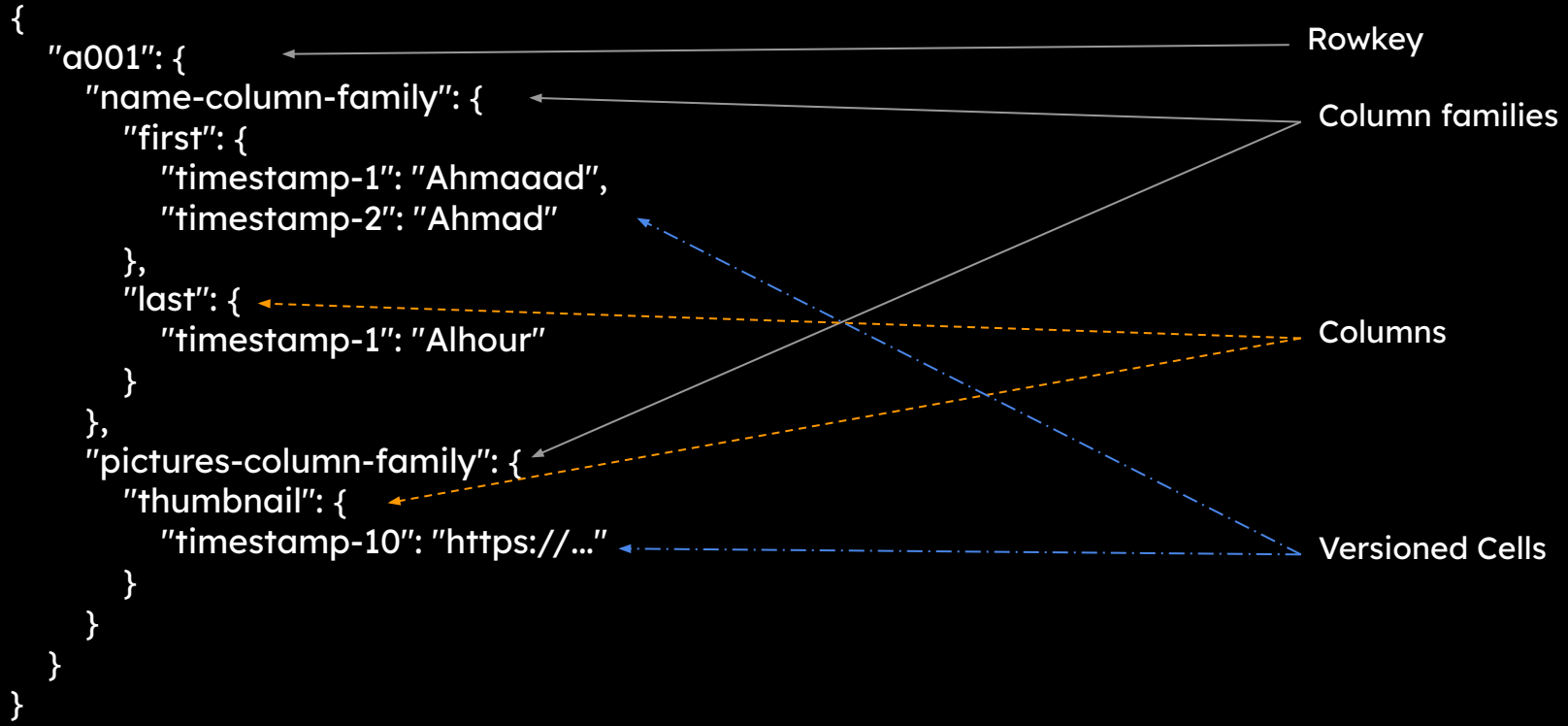


Storage Model

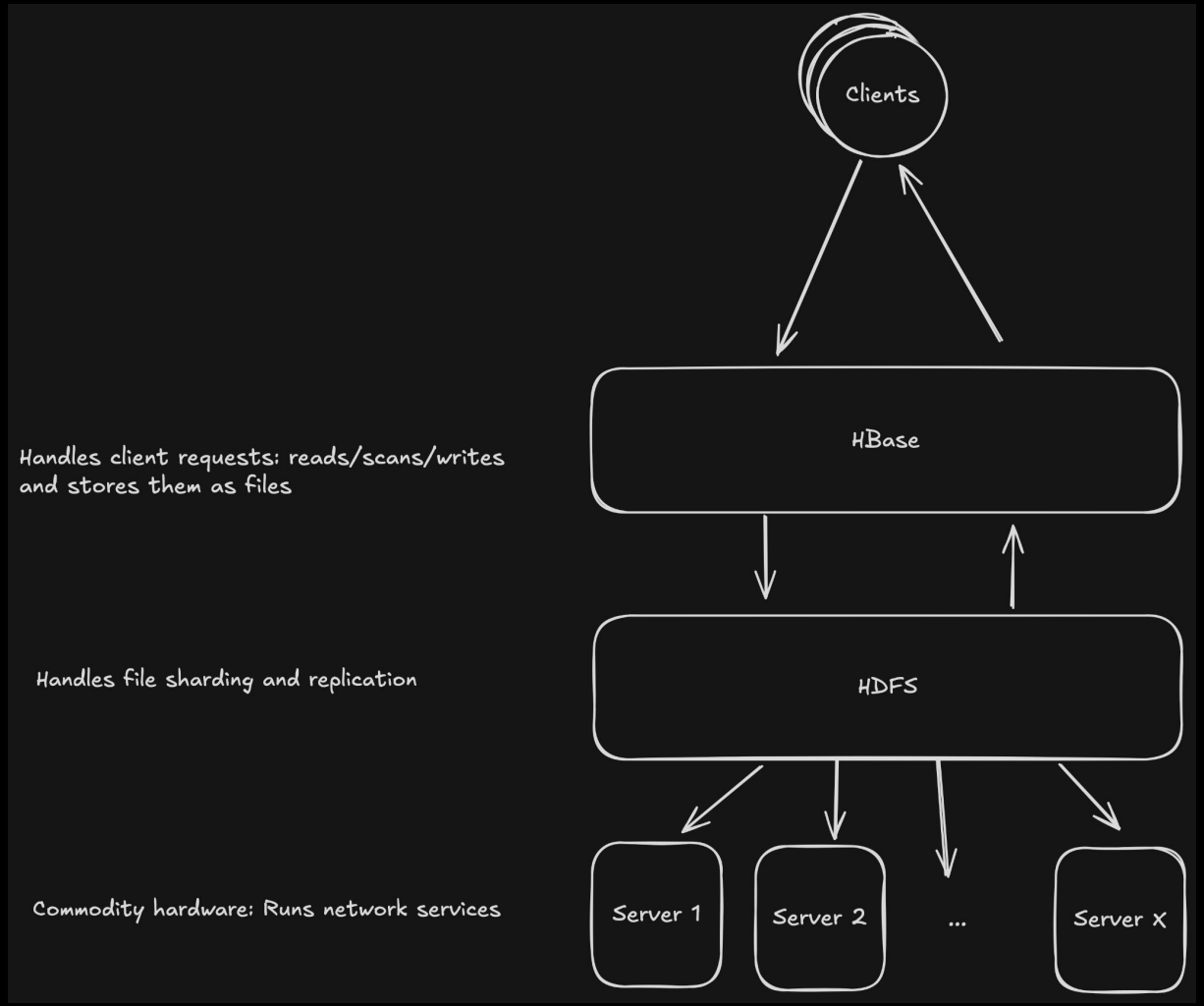


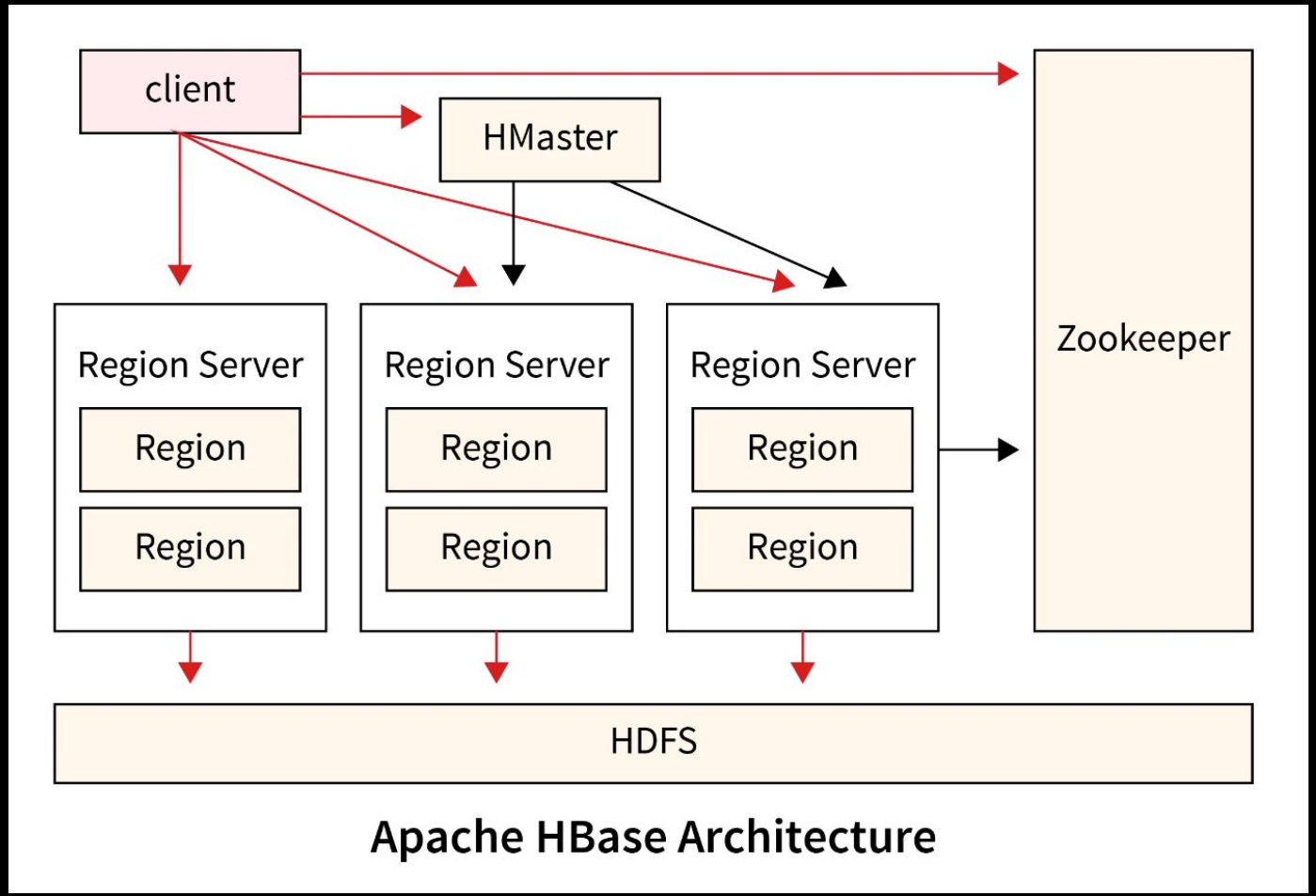
Regions as Parts of the Rowkey Space

The Row as a Map Representation



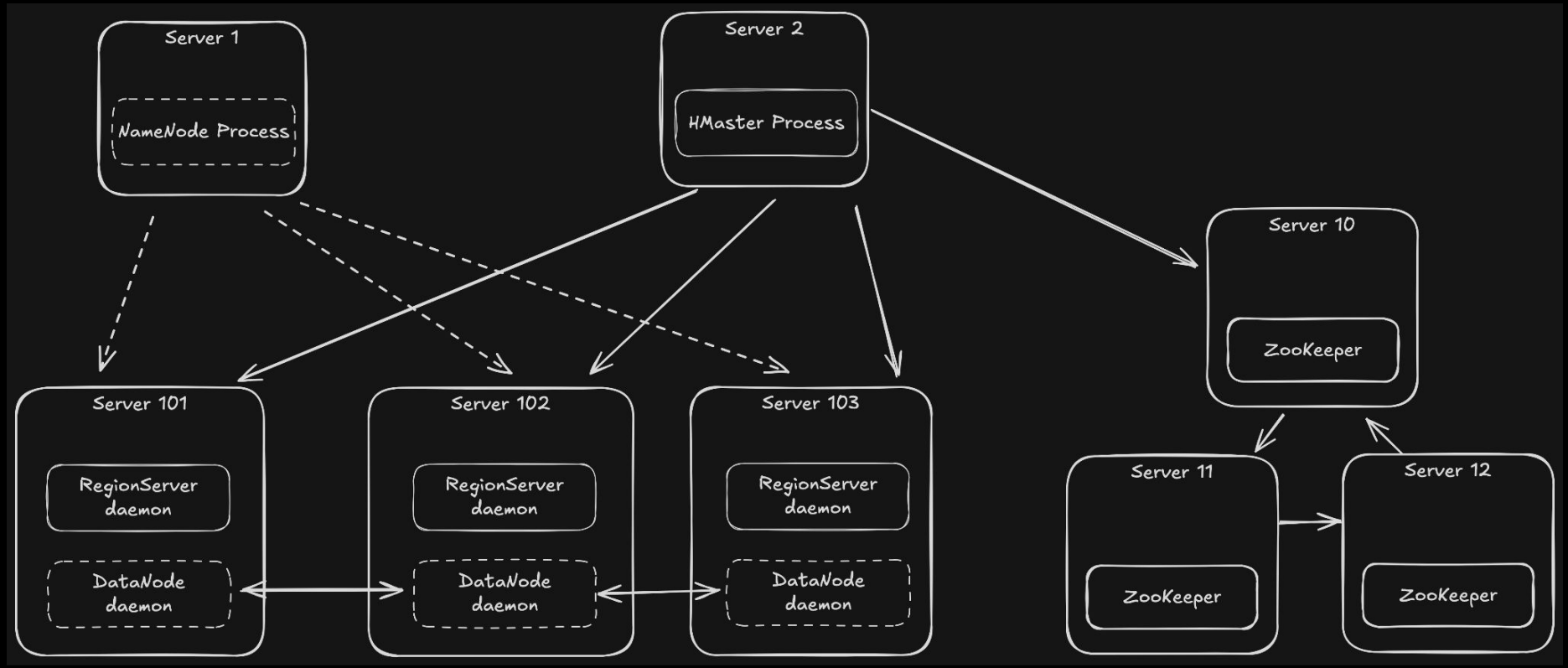
ARCHITECTURE


Clients
HBase
Handles client requests: reads/scans/writes and stores them as files
HDFS
Handles file sharding and replication
Server 1
Server 2
...
Server X
Commodity hardware: Runs network services



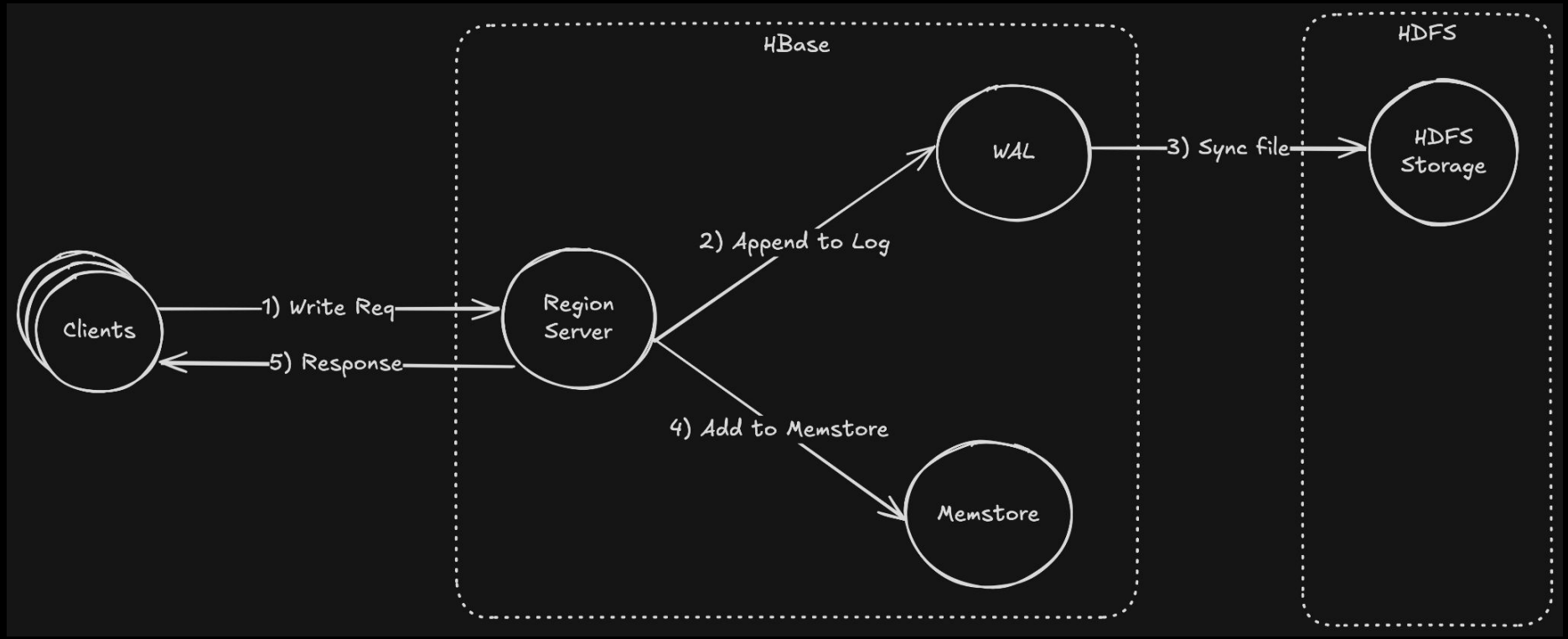
Source: https://www.scaler.com/topics/hadoop/hbase-architecture/

Physical Architecture

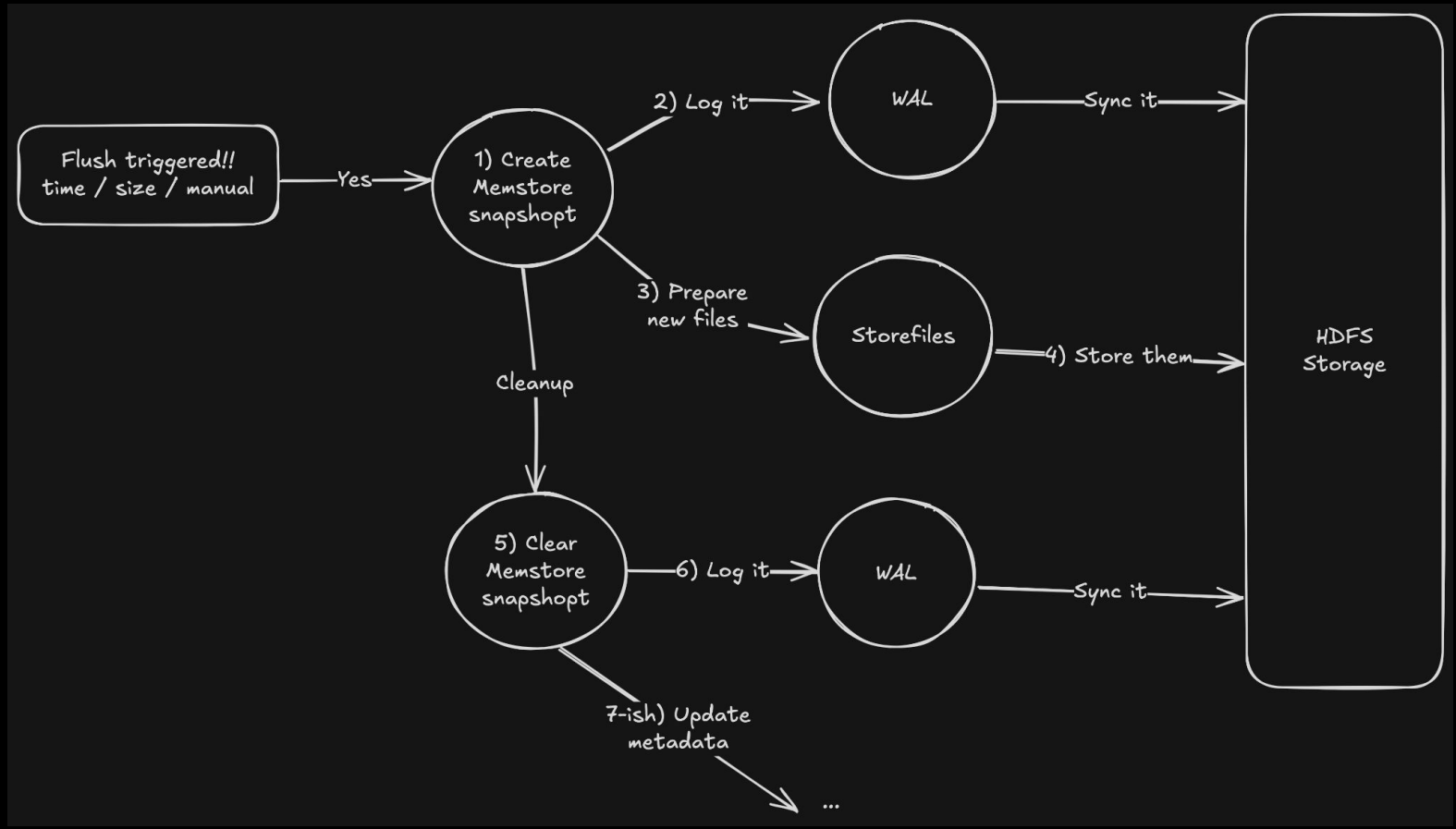


READ & WRITE PATHS

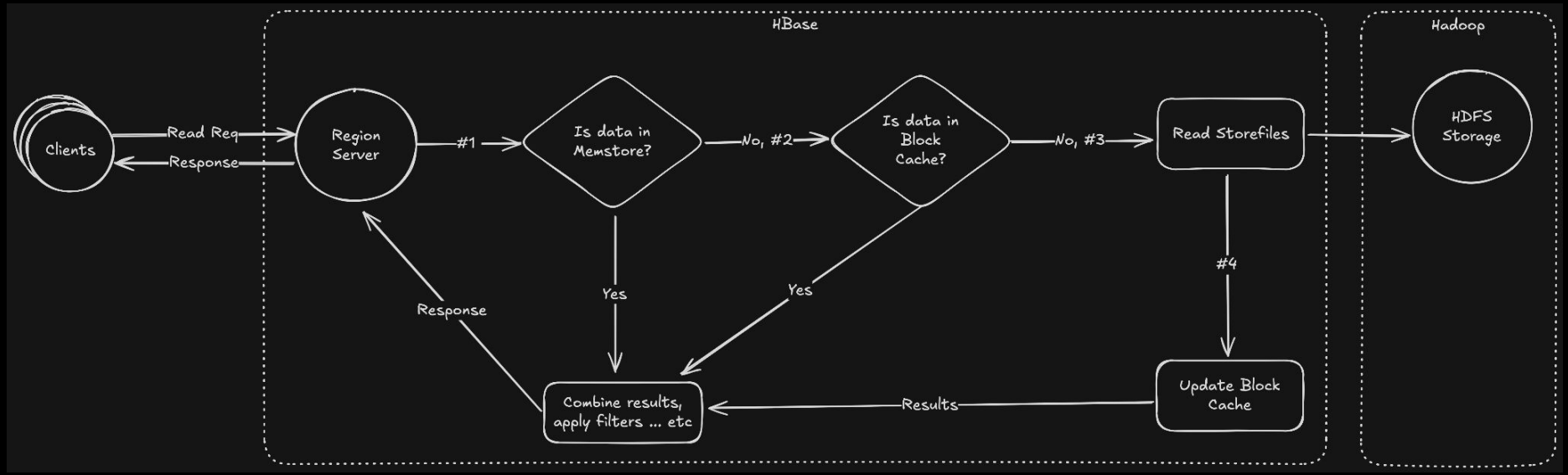
Write Path


HBase
HDFS
WAL
3) Sync file
HDFS Storage
2) Append to Log
Clients
1) Write Req
Region Server
5) Response
4) Add to Memstore
Memstore

Flush Process

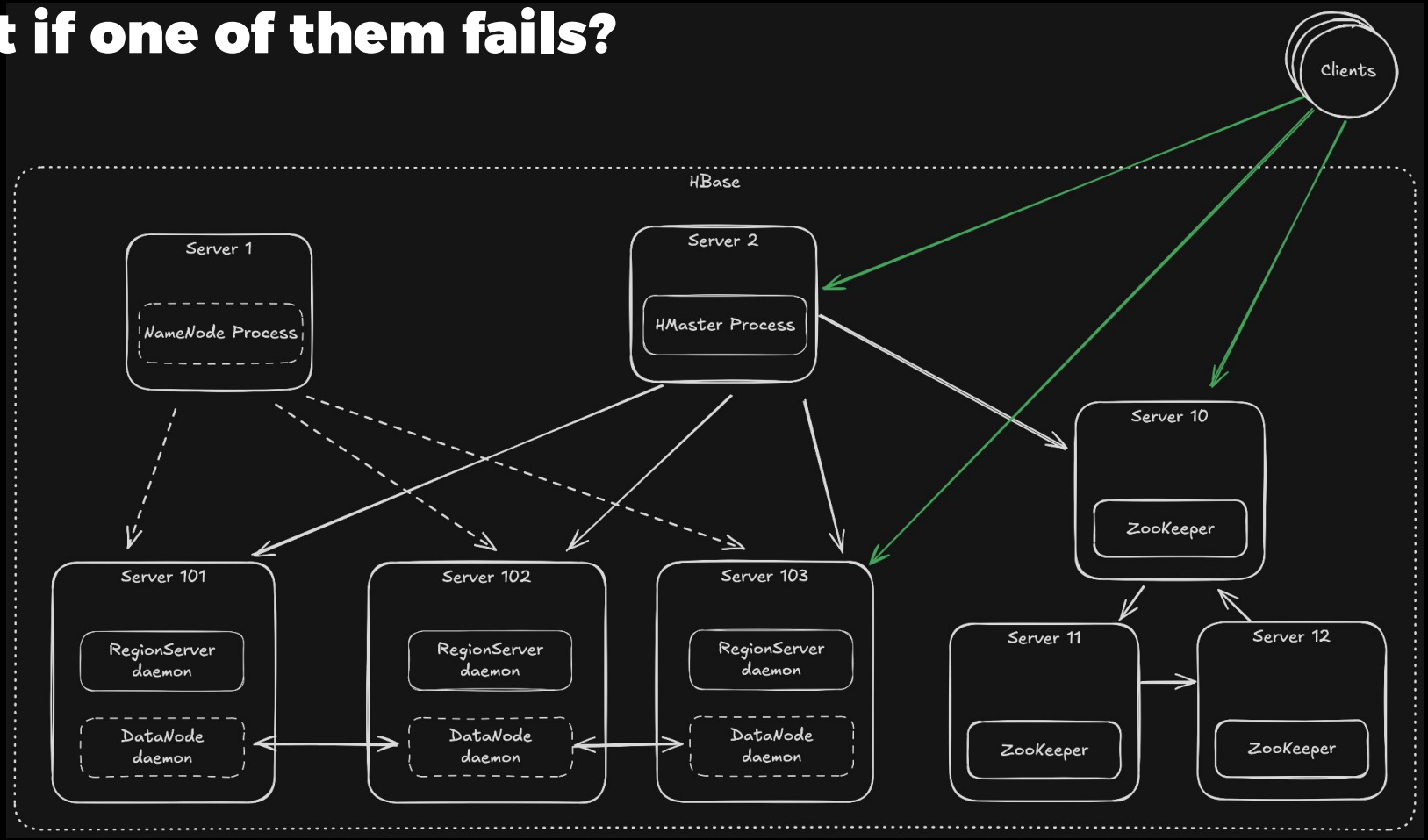


Read Path


HBase
Hadoop
Clients
Read Req
Response
Region Server
#1
Is data in Memstore?
No, #2
Is data in Block Cache?
No, #3
Read Storefiles
HDFS Storage
#4
Yes
Yes
Response
Combine results, apply filters ... etc
Results
Update Block Cache

FAILURE SCENARIOS

What if one of them fails?



OUTRO

What I cannot create, I do not understand.

— Richard Feynman

THANK YOU!

X: @ahmad_alhour
GitHub: @aalhour
LinkedIn: @aalhour

EXTRAS

Summaries of Use Cases

- Large (+ sparse) datasets
- High-volume writes
- Low-latency random reads
- Horizontal scalability

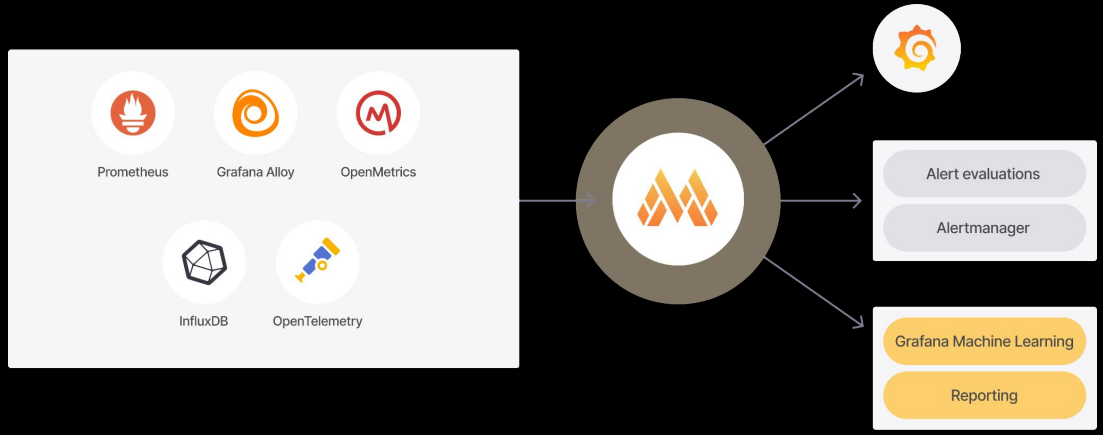
Use Case 3 - Grafana Cloud + Mimir

Setup:

- Grafana Mimir (OSS)
- In all cloud datacenters

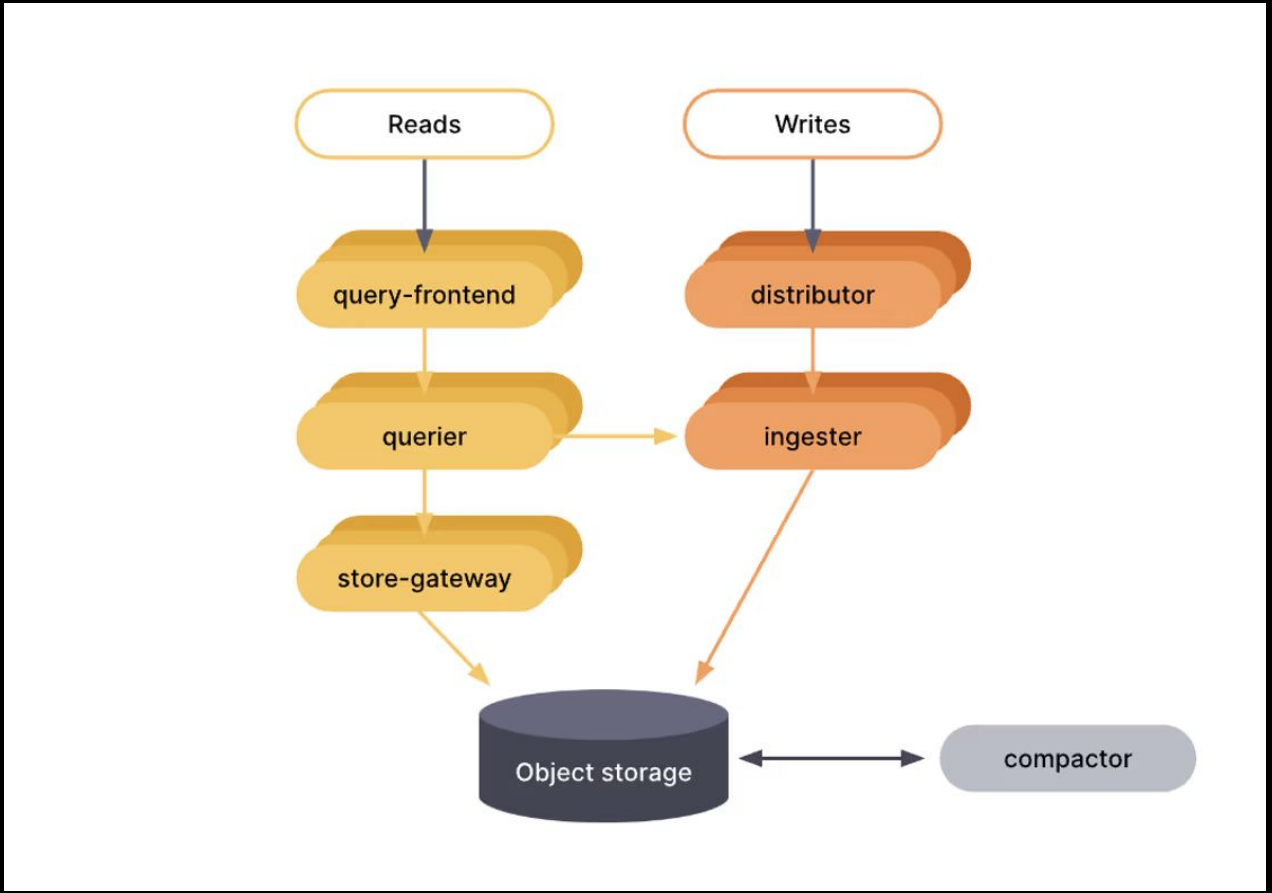
Goals:

- Scales to >1B active metrics series
- HA, multi-tenancy, durable storage
- Fast query performance over long periods of time



Source: https://grafana.com/oss/mimir/

Mimir Architecture



Example Table: Web Crawling

rowkey	timestamp	Family: contents	Family: anchor		Family: people	
		html	cnnsi	my.look.ca	author	email
"com.cnn.www"	t9		"CNN"			
"com.cnn.www"	t8			"CNN.com"		
"com.cnn.www"	t6	"<html>..."				
"com.cnn.www"	t5	"<html>..."				
"com.cnn.www"	t3	"<html>..."			John Doe	john...@...

Source: https://hbase.apache.org/book.html#datamodel

Example HBase Query in Java - Part 1/2

```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;

public class HBaseReadExample {
    public static void main(String[] args) throws Exception {
        // Create HBase configuration
        Configuration config = HBaseConfiguration.create();
        config.set("hbase.zookeeper.quorum", "localhost"); // Replace with actual ZK quorum
        config.set("hbase.zookeeper.property.clientPort", "2181");
```

Example HBase Query in Java: Part 2/2

```java
// Connect to HBase
try (Connection connection = ConnectionFactory.createConnection(config);
     Table table = connection.getTable(TableName.valueOf( name: "my_table"))) {

    // Create a Get query for a specific row key
    Get get = new Get(Bytes.toBytes( s: "row1"));

    // Optional: specify column family and column to retrieve
    get.addColumn(Bytes.toBytes( s: "cf1"), Bytes.toBytes( s: "name"));

    // Execute the get query
    Result result = table.get(get);

    // Retrieve the value
    byte[] value = result.getValue(Bytes.toBytes( s: "cf1"), Bytes.toBytes( s: "name"));
    String name = Bytes.toString(value);

    System.out.println("Name: " + name);
}
```

THANK YOU!

Encourage your audience to ask questions or provide feedback on the presentation. Invite them to continue the conversation at a later date.

Contact
name@example.com