

# Distributed Data Processing & Distributed Databases

# Agenda



- Map Reduce Revisited
- Hive
- High Availability and Partitioning Strategies
- Scaling NoSQL Databases
  - Hbase
  - Cassandra
  - MongoDB
- Scaling Relational Databases

# Map Reduce

# What is map reduce?

-  A distributed method of processing data over several nodes
-  A programming model for data processing
-  Supports multiple languages
-  Built into Hadoop (So, Hadoop consists of HDFS and Map Reduce)

# Logical Data Flow

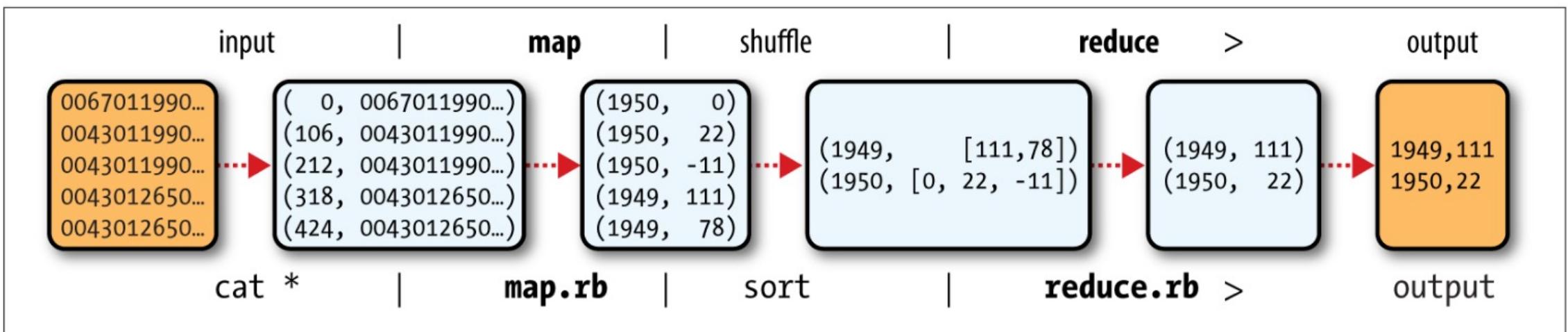


Figure 2-1. MapReduce logical data flow

- Data comes in **(key, value)** pairs,
- ...all loaded from files on the DFS.

Key, value: arbitrary, serializable types.

Each should fit in memory on one machine!

Since cluster components fail,

Parallelize computation into small **tasks**; if one fails to deliver results, restart that task.

# Map tasks

1. Read **(key, value)** pairs in input, from the DFS.
2. One **Map** task per pair  
(will be scheduled on/near the machine where the input is).
3. Computes a number of **(key, value)** pairs, decided by you.
4. Outputs to the (local!) disk in a **buffer region**.

```
map(key, value) → list(key, value)
```

# The word-count problem

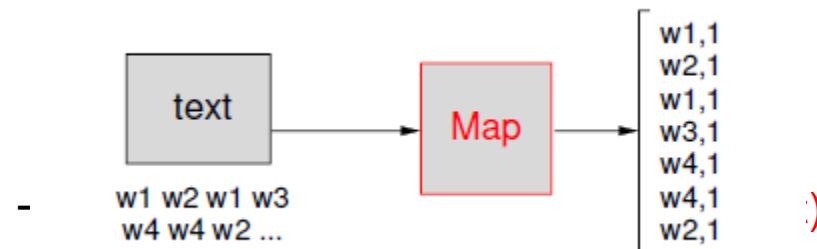
Given  $n$  files, compute the global count of each word.

File  $f_8: w_1 \ w_2 \ w_1 \ w_3 \ w_1 \dots$

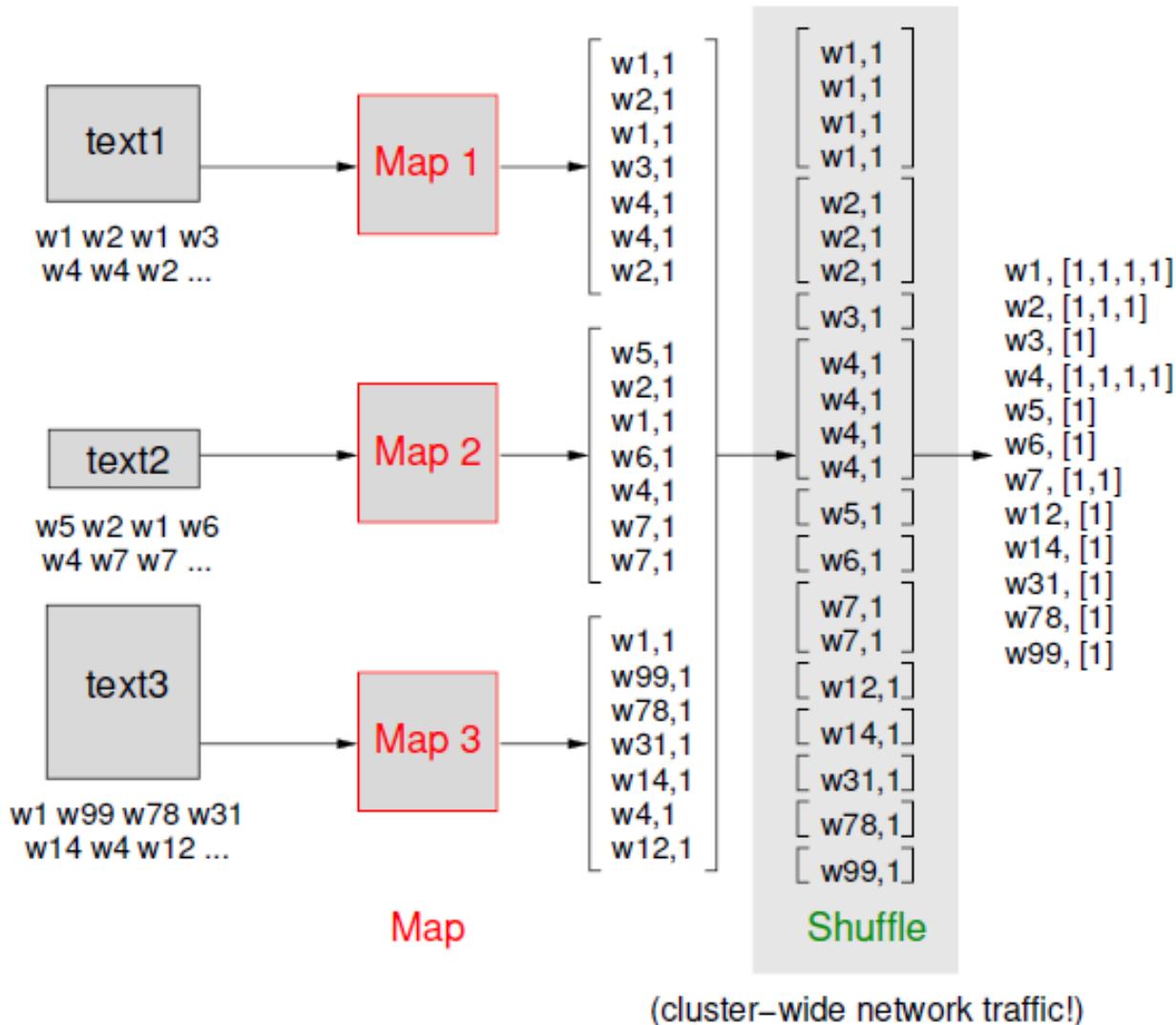
File  $f_9: w_1 \ w_3 \ w_1 \dots$

Output:  $w_1 : 5, w_3 : 3 \dots$

→ Map input: (filename, text)



# Multiple Map tasks



# Master controller (Shuffle):

→ Deals with the intermediate data:

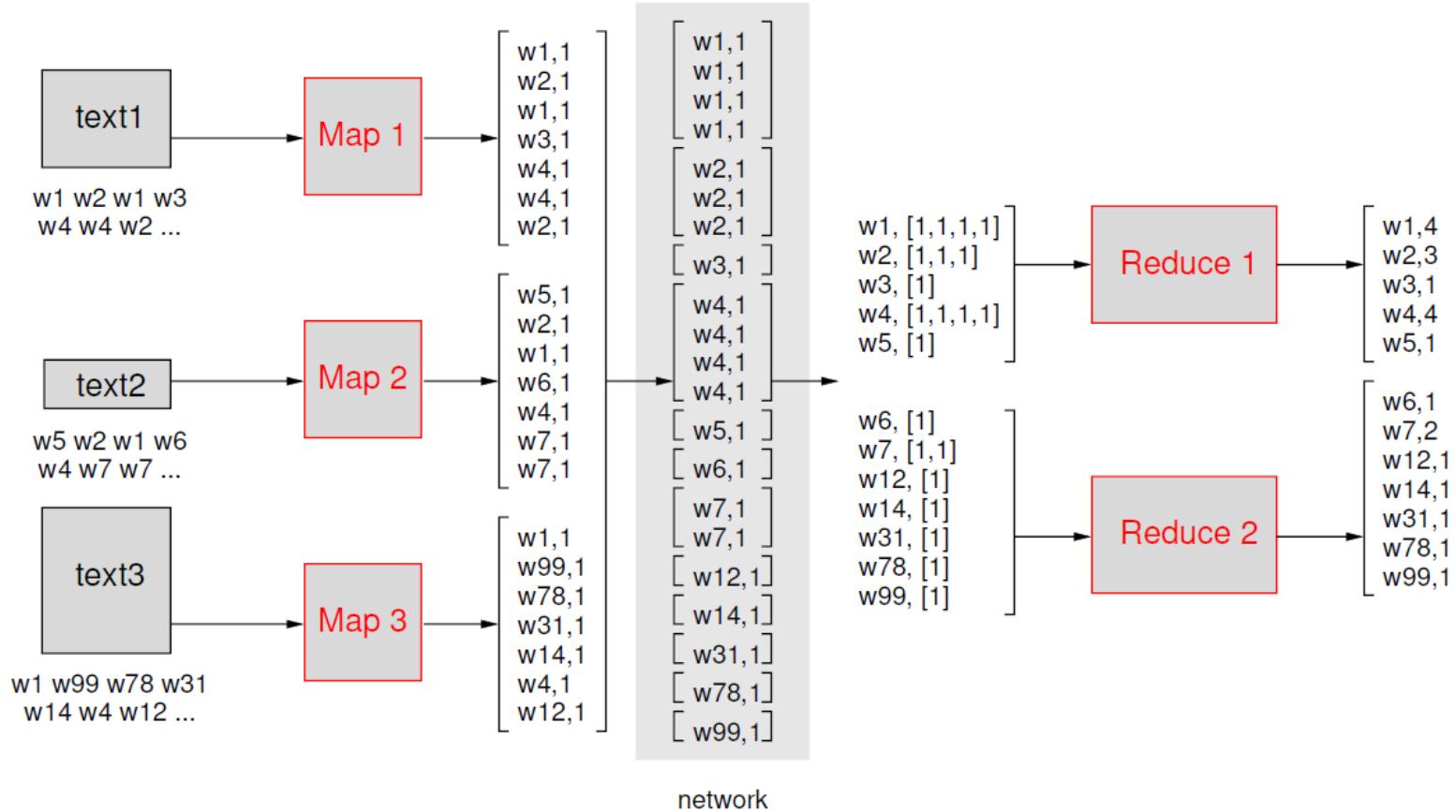
1. Keeps track of the (key, value) pairs in output of all Map tasks.
2. Does a distributed group by key operation.
3. A key and its list of values will be read by a single Reduce task.

# Reduce

1. One Reduce works on one key at a time.
2. Computes a combined value per key, decided by you.
3. The output is saved to DFS files (one per Reduce task).

```
reduce(key, list(value)) → list(value)
```

# Reduce input: (word, list(count))



For old MapReduce code (on Hadoop, in Java)  
SDU

# Map Reduce Overview

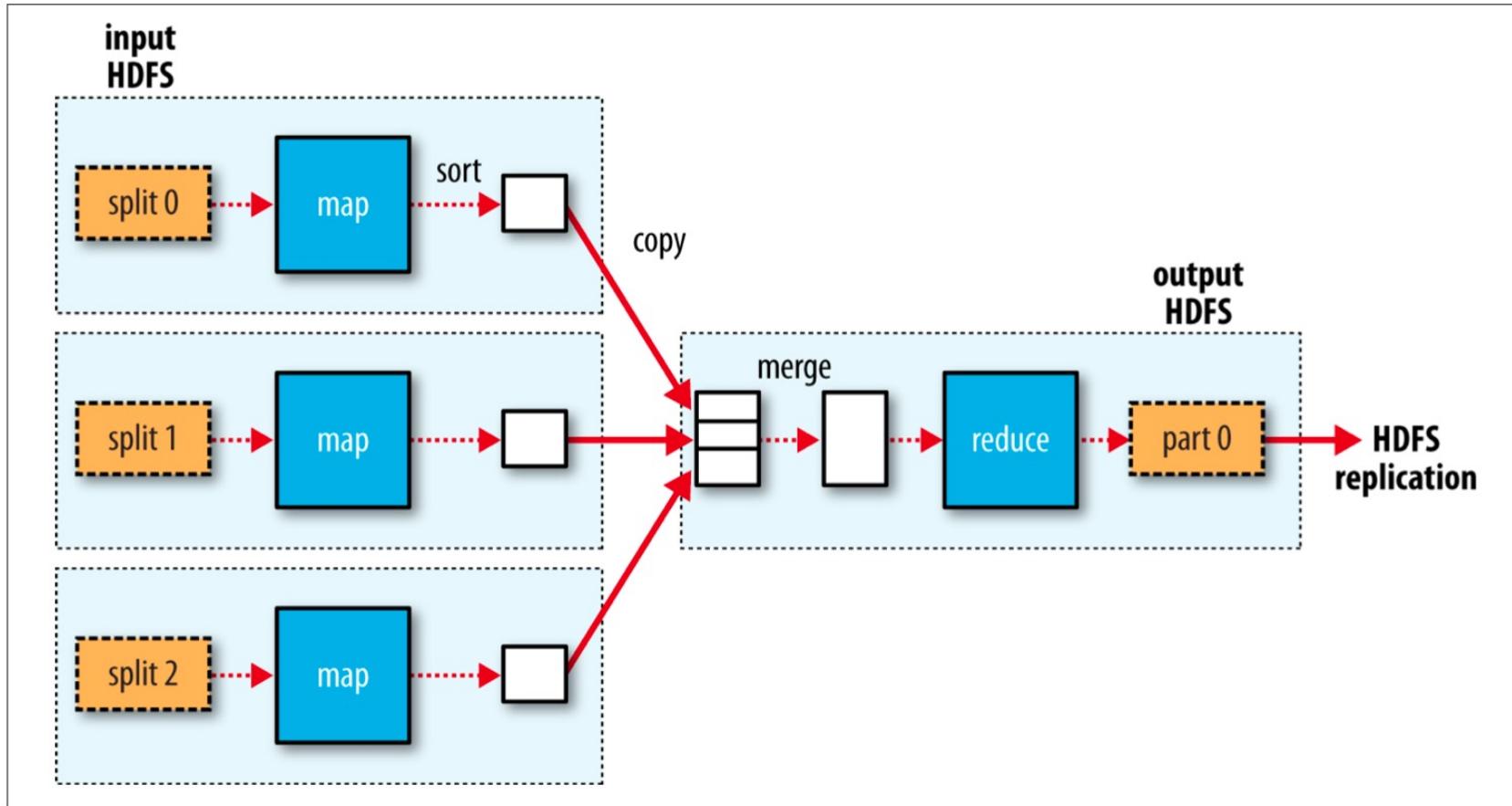
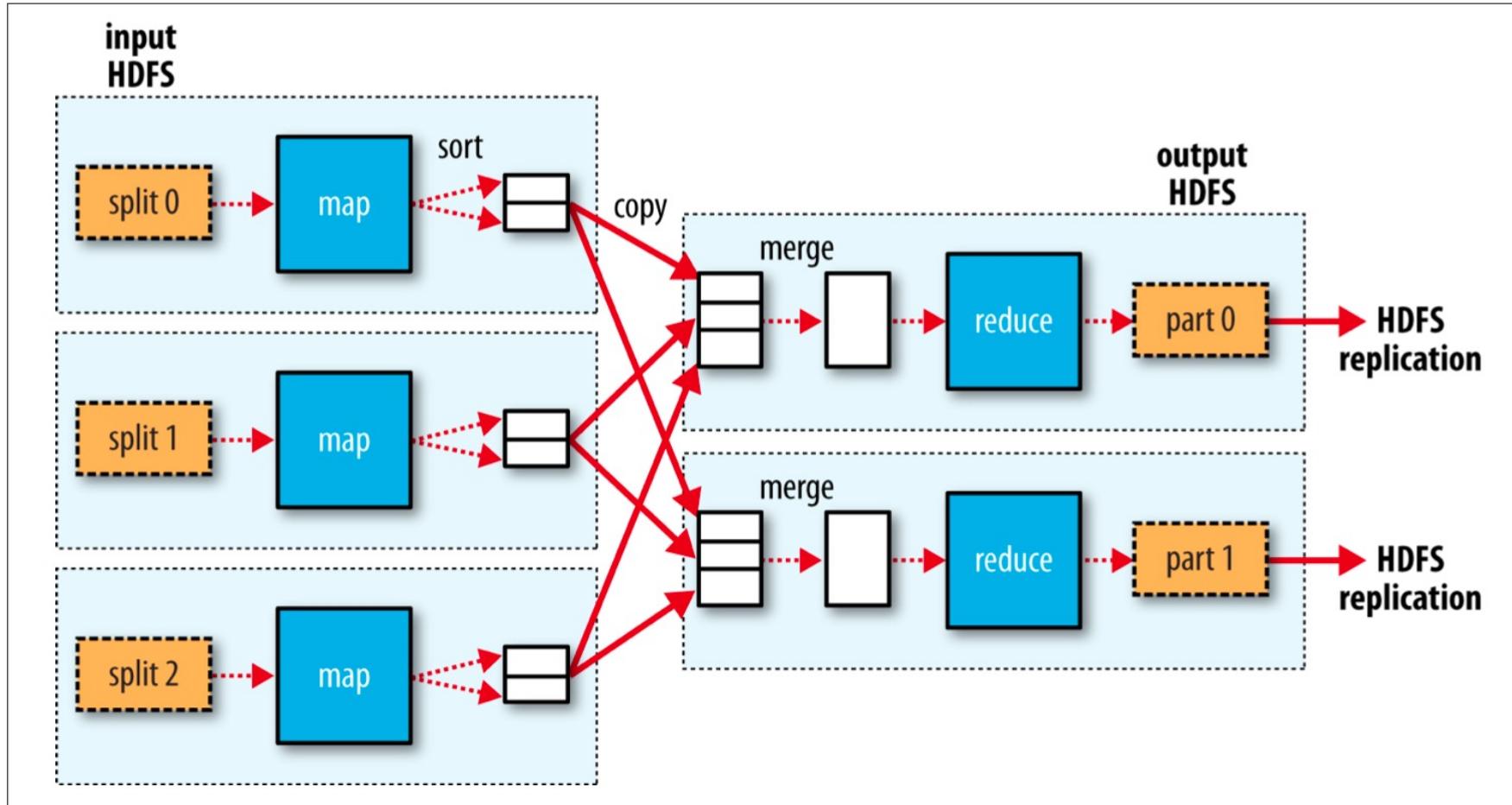
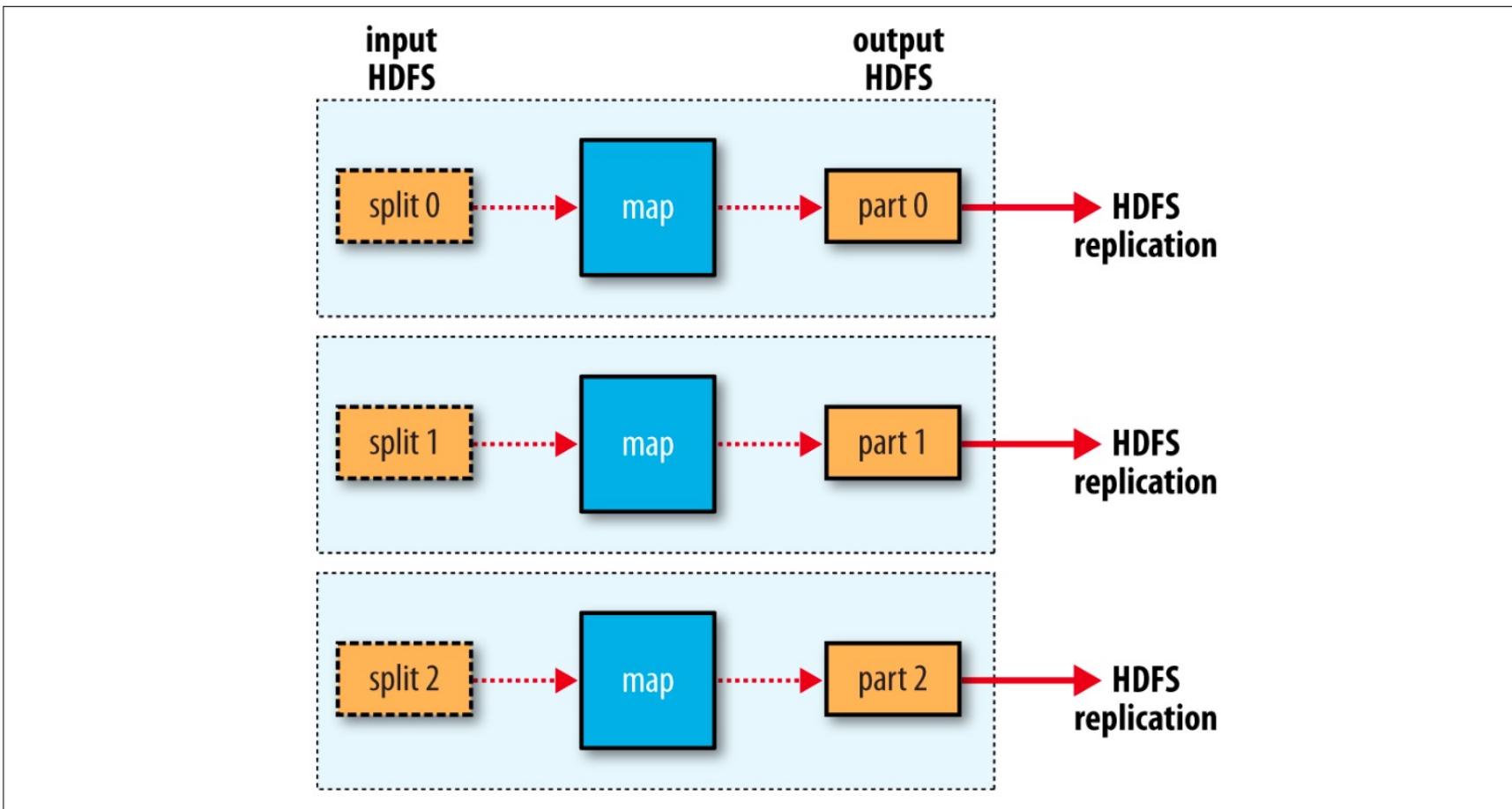


Figure 2-3. MapReduce data flow with a single reduce task

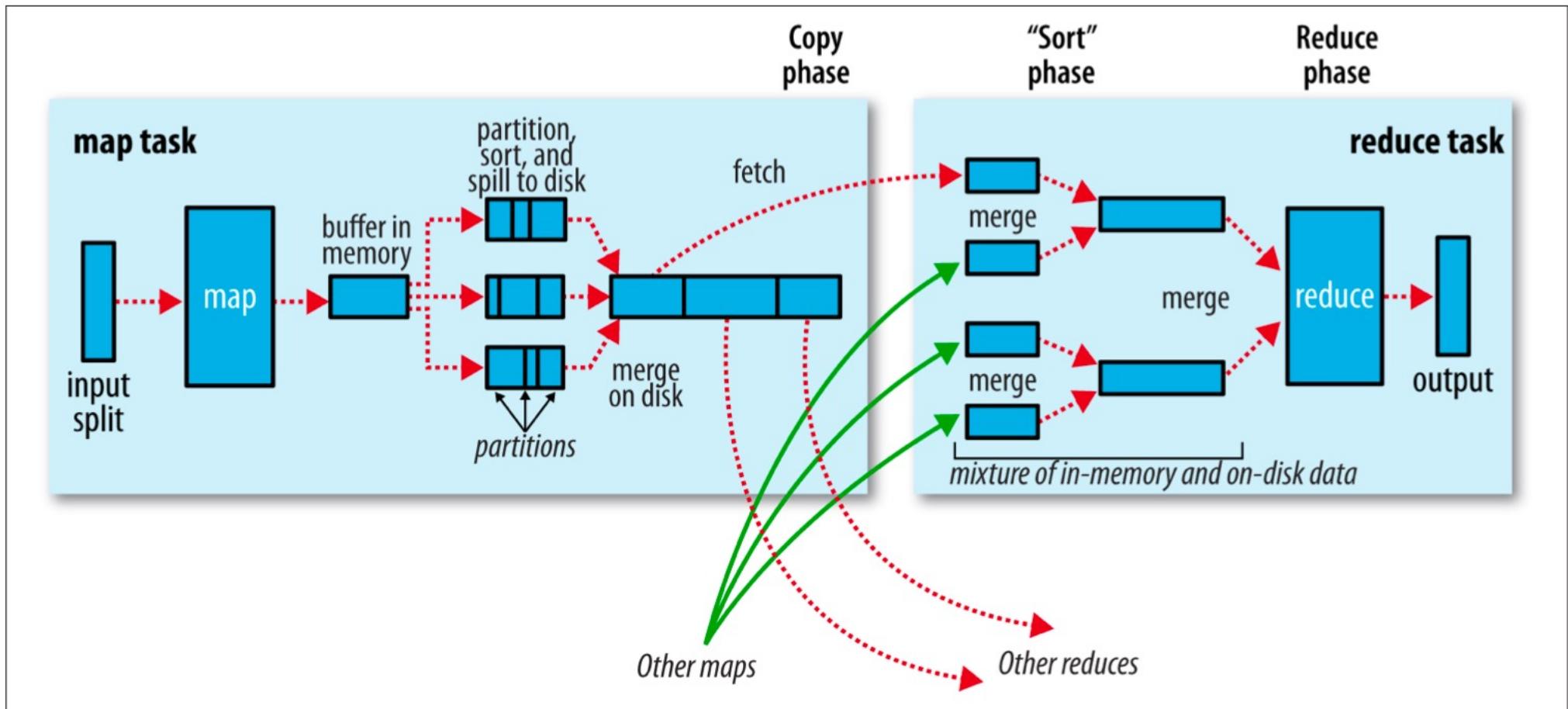
# Map Reduce Overview – Multiple Reduce



# Map Reduce Overview – Map Only



# The Complicated Version



# Data Acquisition from HDFS

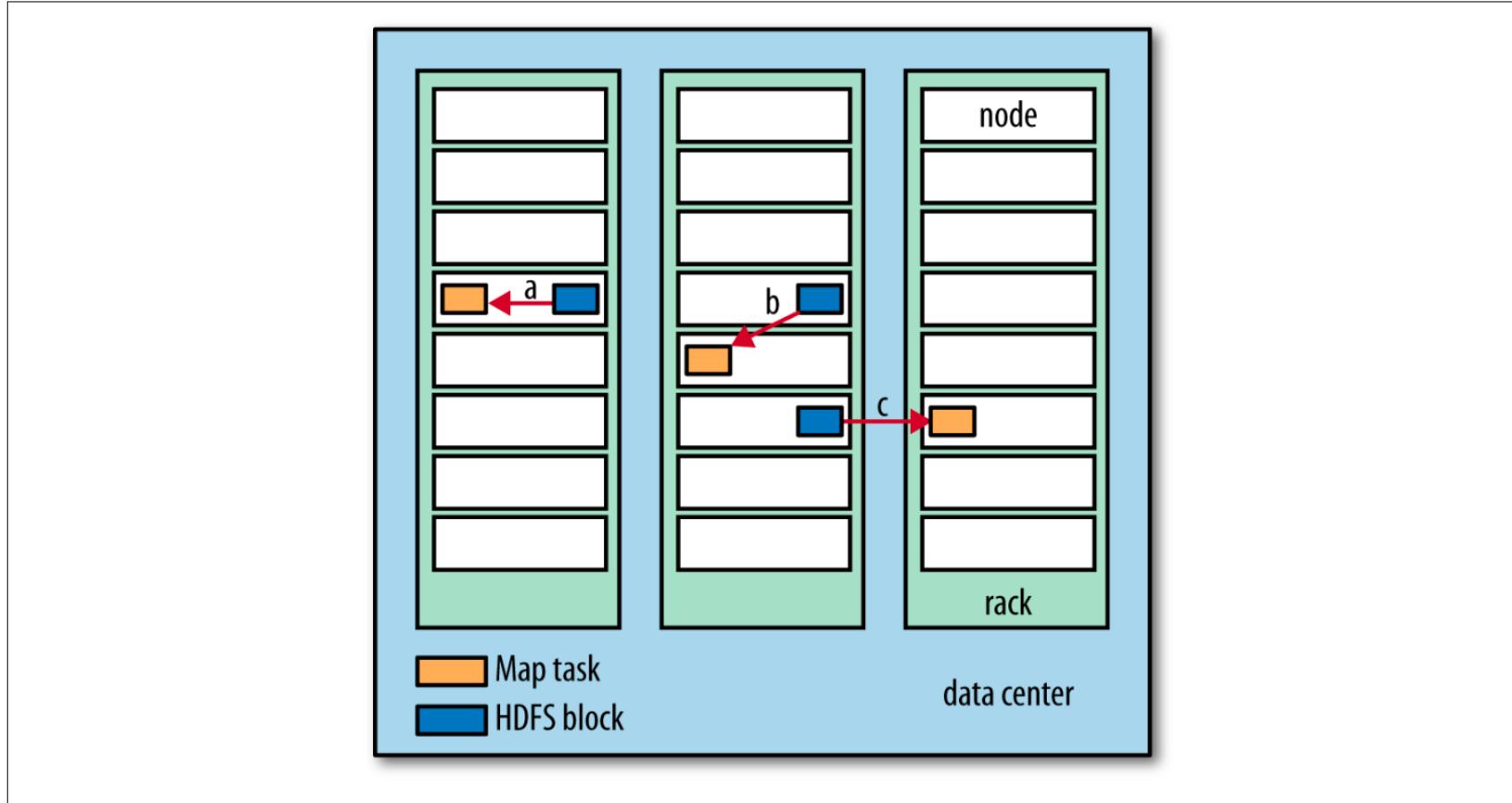


Figure 2-2. Data-local (a), rack-local (b), and off-rack (c) map tasks

# Web Overview

The screenshot shows the Hadoop Resource Manager web interface. At the top left is the Hadoop logo. To its right, the title "All Applications" is displayed. In the top right corner, it says "Logged in as: dr.who". On the left side, there's a sidebar with a tree view under "Cluster" and sections for "About", "Nodes", "Applications" (with sub-sections: NEW, NEW SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED), "Scheduler", and "Tools". The main content area has two tables: "Cluster Metrics" and "User Metrics for dr.who". Below these are two data tables showing application details. The first table has columns: Apps Submitted, Apps Pending, Apps Running, Apps Completed, Containers Running, Memory Used, Memory Total, Memory Reserved, Active Nodes, Decommissioned Nodes, Lost Nodes, Unhealthy Nodes, and Rebooted Nodes. The second table has columns: Apps Submitted, Apps Pending, Apps Running, Apps Completed, Containers Running, Containers Pending, Containers Reserved, Memory Used, Memory Pending, and Memory Reserved. Both tables show zero values. Below these tables is a search bar with "Search:" and a dropdown for "Show 20 entries". The main data table lists three applications:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1410450250506_0003	ec2-user	Max temperature	MAPREDUCE	root.ec2-user	Fri, 12 Sep 2014 10:38:11 GMT	N/A	RUNNING	UNDEFINED		<a href="#">ApplicationMaster</a>
application_1410450250506_0002	ec2-user	Max temperature	MAPREDUCE	root.ec2-user	Fri, 12 Sep 2014 10:27:23 GMT	Fri, 12 Sep 2014 10:34:36 GMT	FINISHED	SUCCEEDED		<a href="#">History</a>
application_1410450250506_0001	ec2-user	distcp	MAPREDUCE	root.ec2-user	Fri, 12 Sep 2014 08:47:09 GMT	Fri, 12 Sep 2014 08:52:56 GMT	FINISHED	SUCCEEDED		<a href="#">History</a>

At the bottom, it says "Showing 1 to 3 of 3 entries" and has navigation links: First, Previous, 1, Next, Last.

Figure 6-1. Screenshot of the resource manager page

# Web Overview – Single Task

The screenshot shows the Hadoop Web UI interface. At the top left is the Hadoop logo. On the right, it says "Logged in as: dr.who". The main title is "MapReduce Job job\_1410450250506\_0003". The left sidebar has a tree structure: Cluster, Application, Job (with sub-options: Overview, Counters, Configuration, Map tasks, Reduce tasks, AM Logs), and Tools. The "Job" section is expanded. The "Job Overview" section contains the following details:

Job Name:	Max temperature
State:	RUNNING
Uberized:	false
Started:	Fri Sep 12 06:38:24 EDT 2014
Elapsed:	6mins, 25sec

The "ApplicationMaster" section shows one entry:

Attempt Number	Start Time	Node	Logs
1	Fri Sep 12 06:38:19 EDT 2014	ip-10-1-1-172.ec2.internal:8042	logs

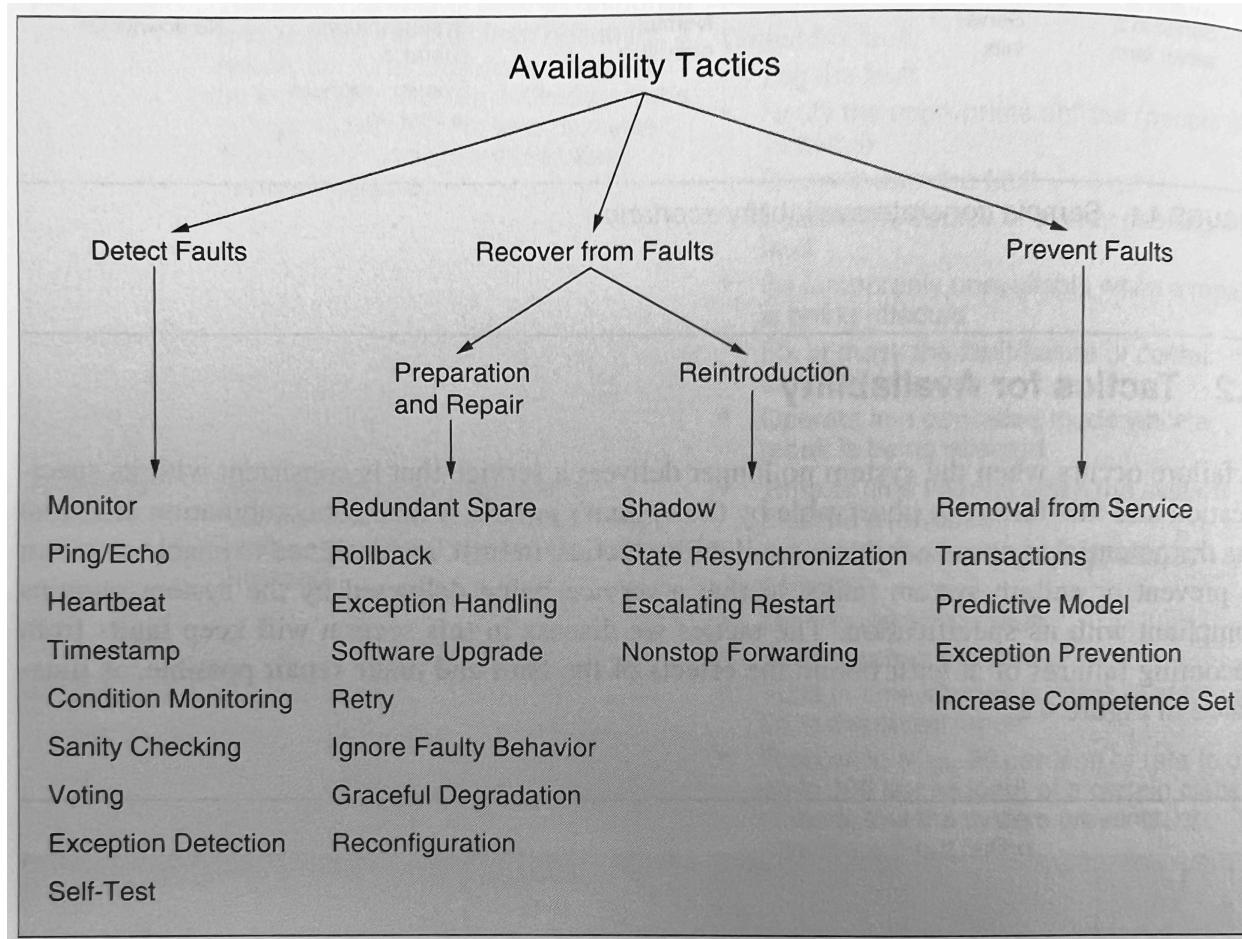
The "Task Type" section provides a summary of task counts:

Task Type	Progress	Total	Pending	Running	Complete
Map	<div style="width: 40%;"></div>	101	25	14	62
Reduce	<div style="width: 3%;"></div>	8	8	0	0

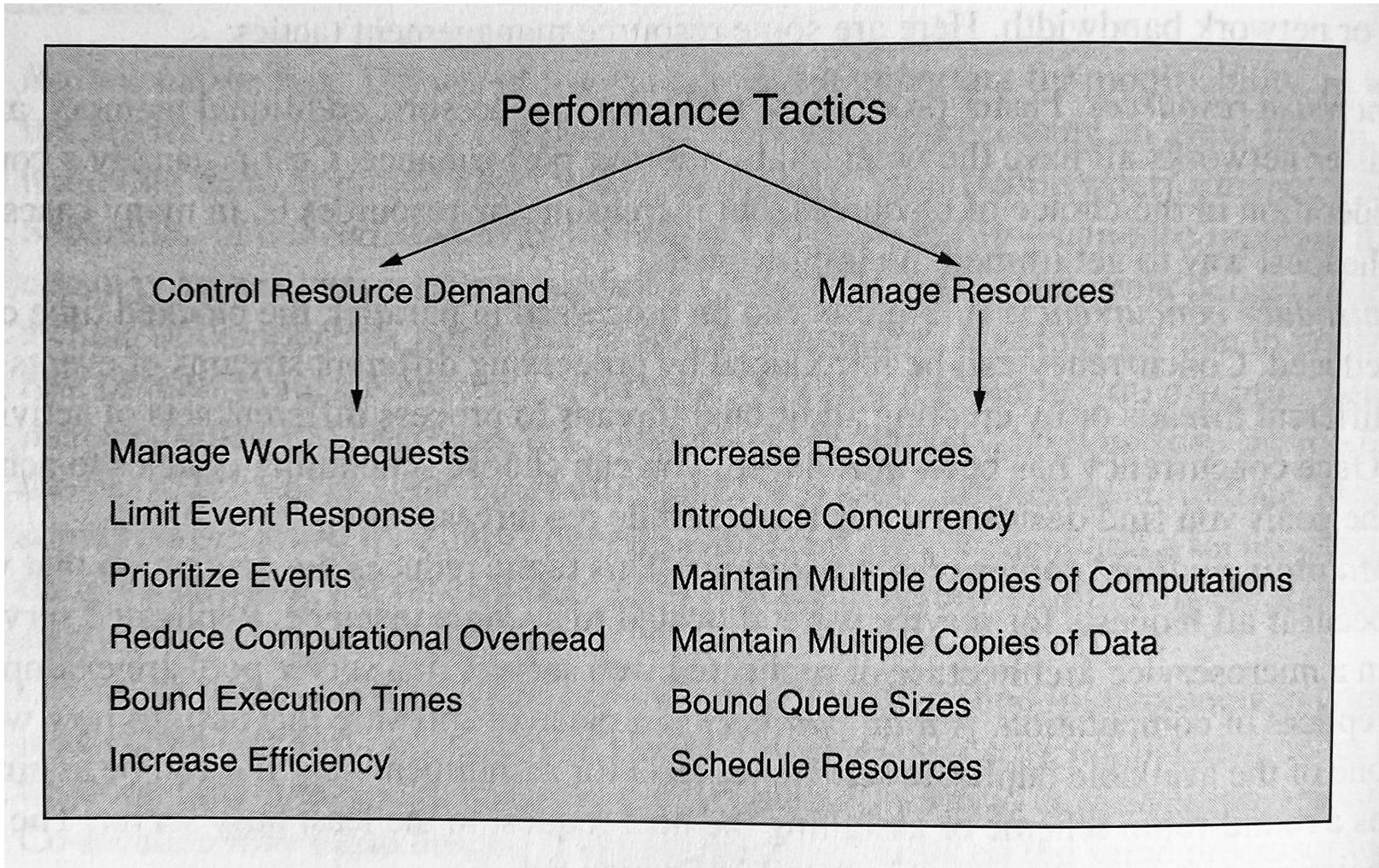
The "Attempt Type" section details the status of individual map and reduce attempts:

Attempt Type	New	Running	Failed	Killed	Successful
Maps	25	14	0	0	62
Reduces	8	0	0	0	0

# Availability Tactics



# Performance



# Yarn

# What is Yarn?

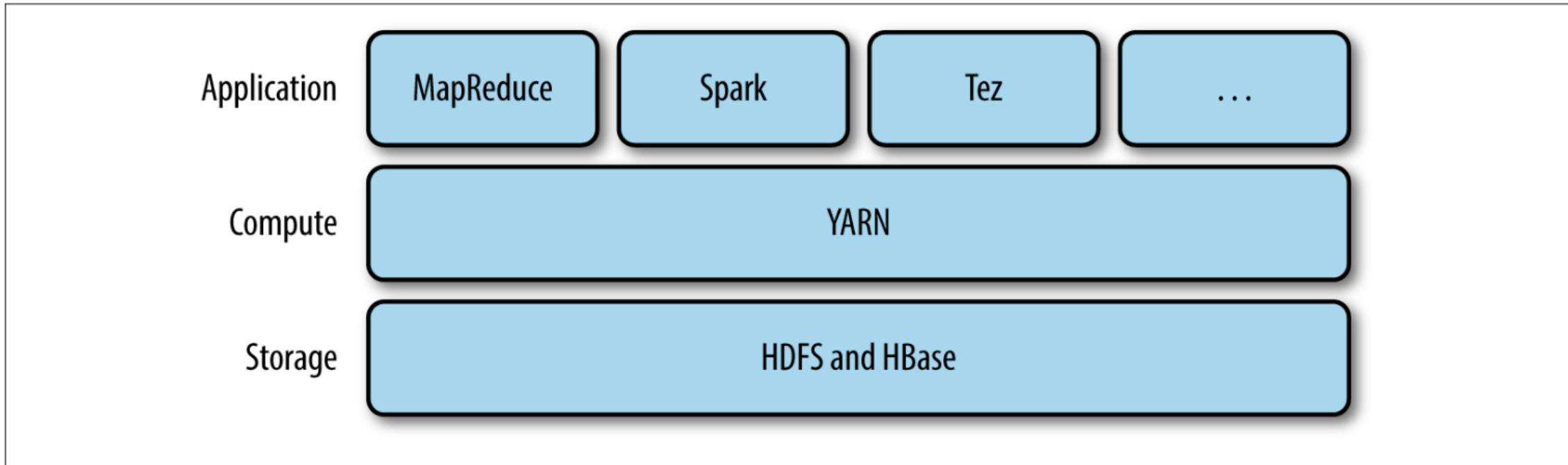
MapReduce 1

YARN

Jobtracker	Resource manager, application master, timeline server
Tasktracker	Node manager
Slot	Container

- Acronym for Yet Another Resource Negotiator
- The newer Hadoop cluster resource manager
- Deploys and distributes tasks to:
  - Map Reduce
  - Spark
  - ... (and more)
- Automatically distributes tasks to multiple hosts inside of containerized environments

# Yarn Stack



*Figure 4-1. YARN applications*

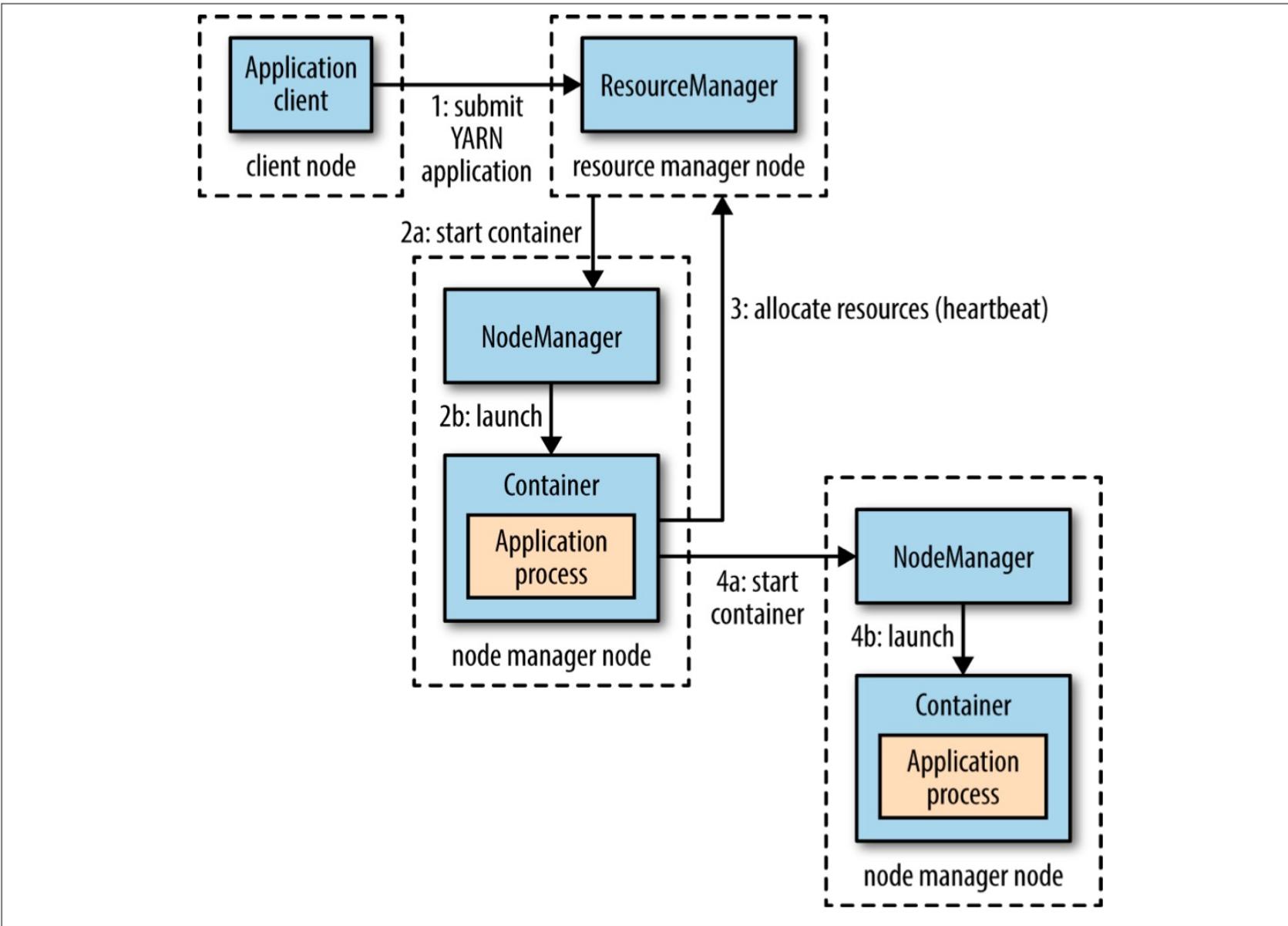


Figure 4-2. How YARN runs an application

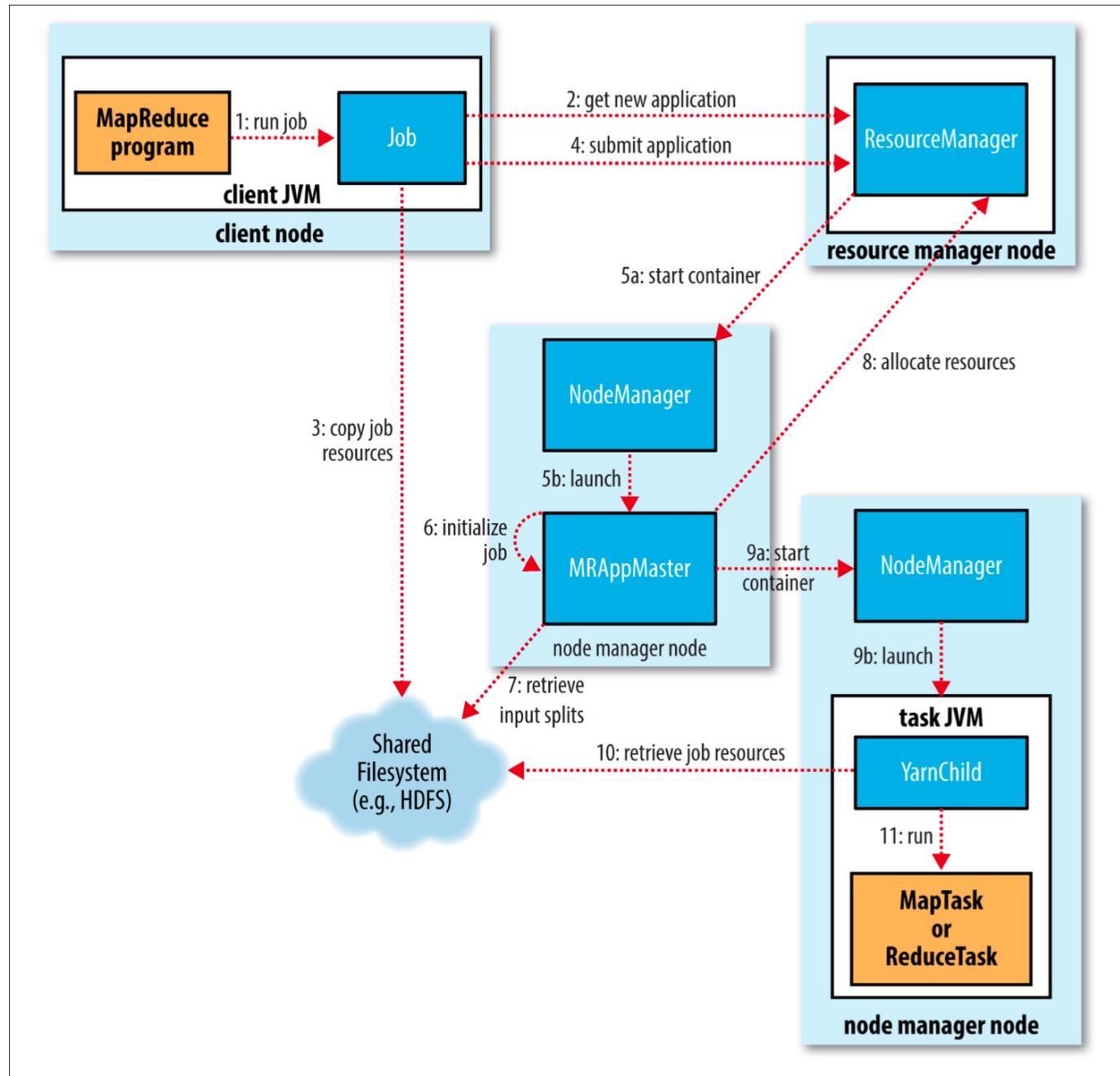


Figure 7-1. How Hadoop runs a MapReduce job

# Apache Hive

# What is Hive?

- A system for managing and querying structured data built on top of Hadoop (HDFS)
- Uses MapReduce for execution
- Allows for a typical SQL interface to a distributed file structure
- Even works with CSV files
- Performs well

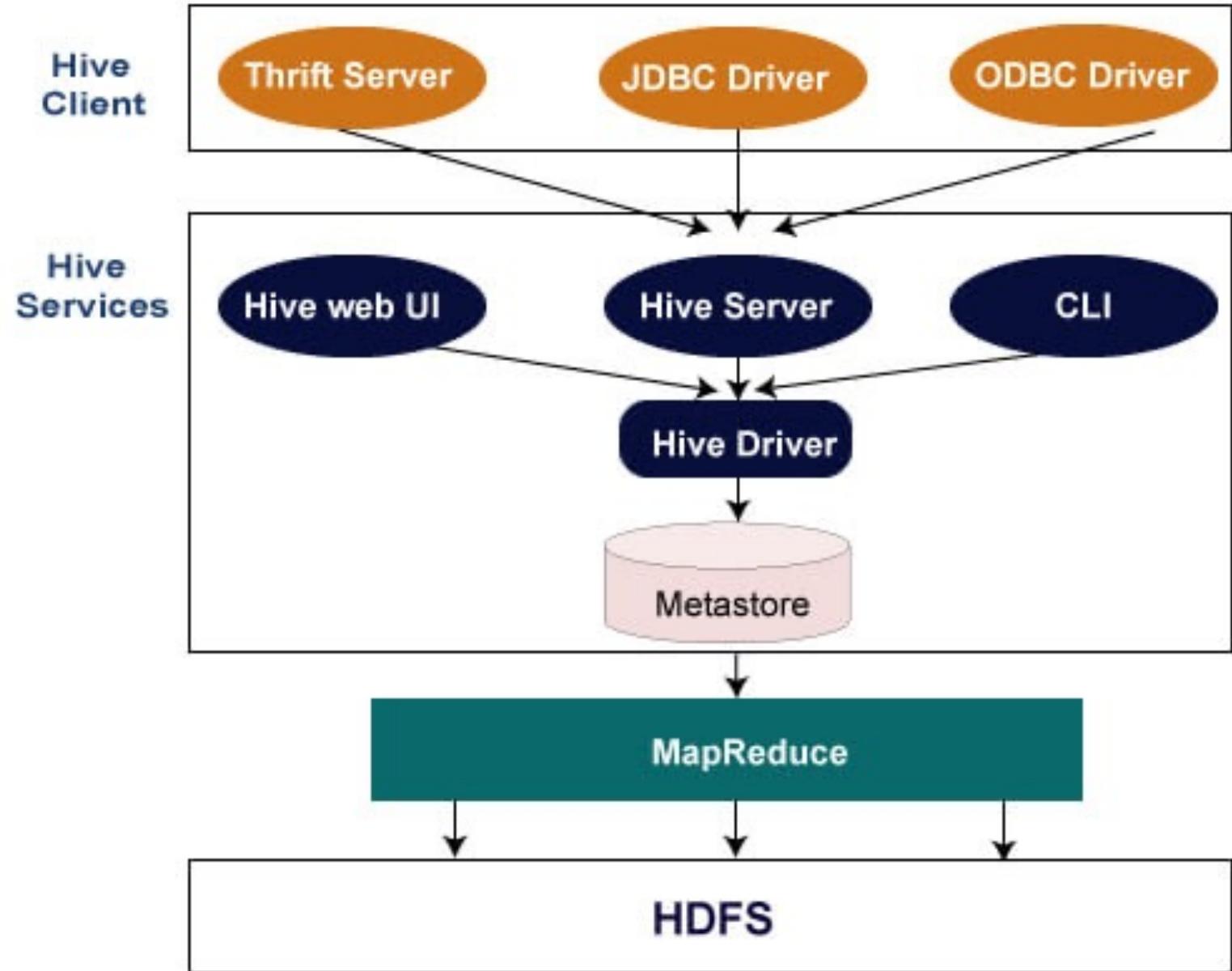
# Define Schema – Then query!

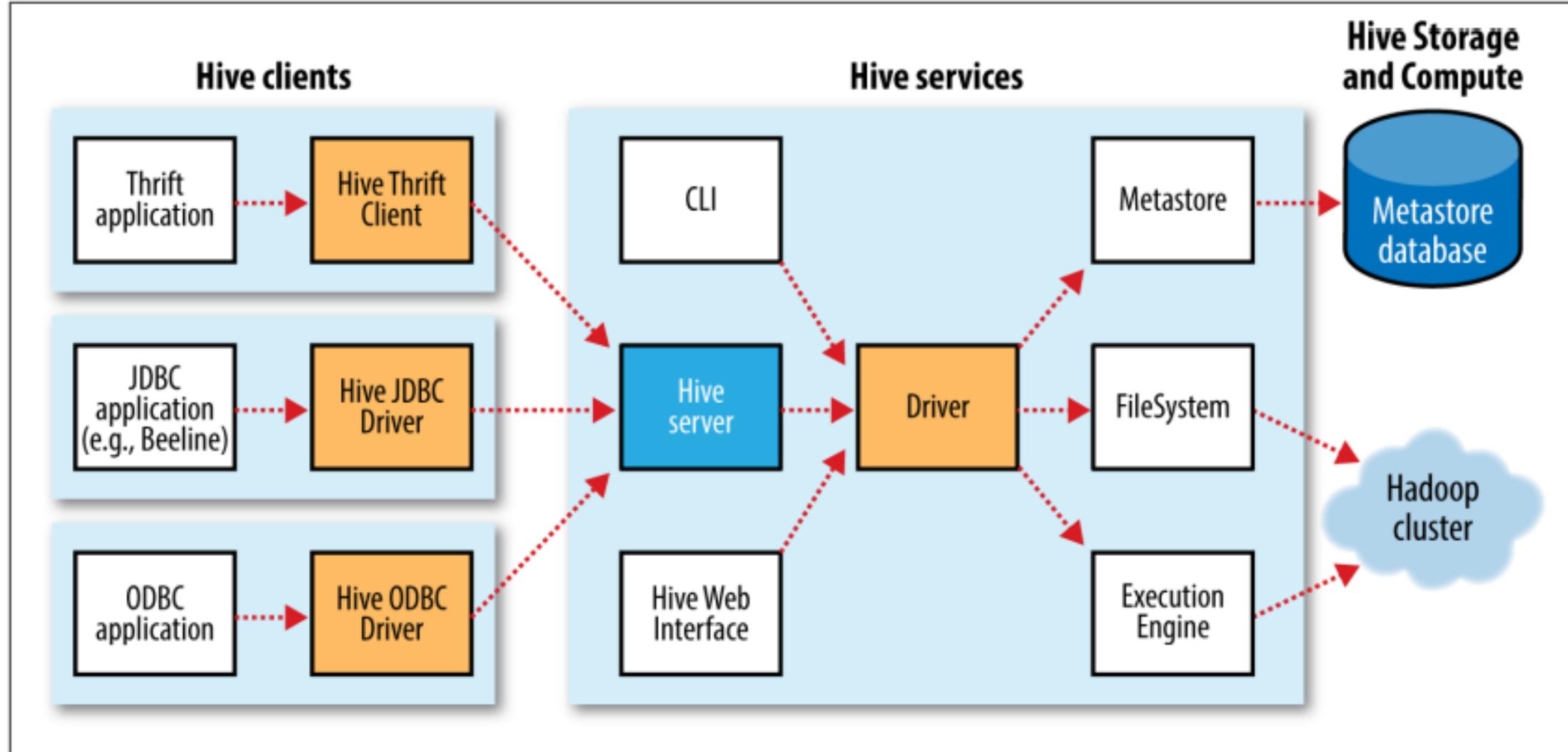
```
CREATE TABLE records (year STRING, temperature INT, quality INT)
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '\t';
```



```
hive> SELECT year, MAX(temperature)
    > FROM records
    > WHERE temperature != 9999 AND quality IN (0, 1, 4, 5, 9)
    > GROUP BY year;
1949      111
1950      22
```

# Apache Hive



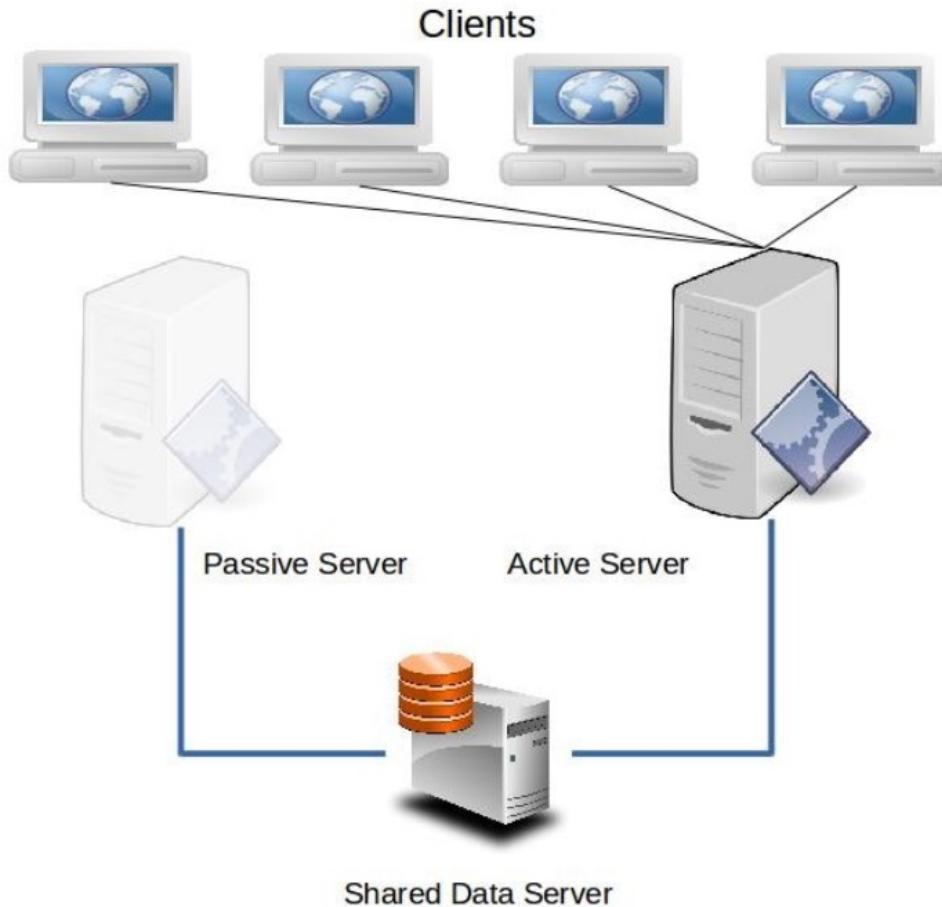


*Figure 17-1. Hive architecture*

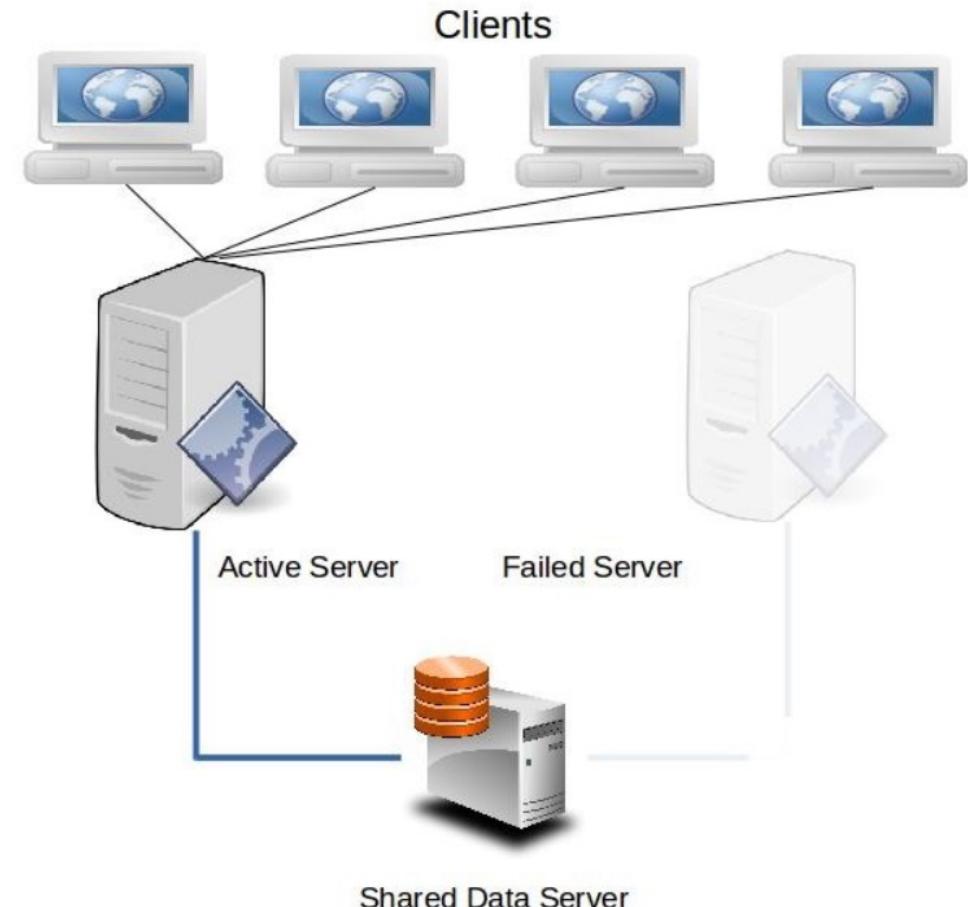
# High Availability and Partitioning Strategies

# High Availability 1/3

## Active/Passive Cluster

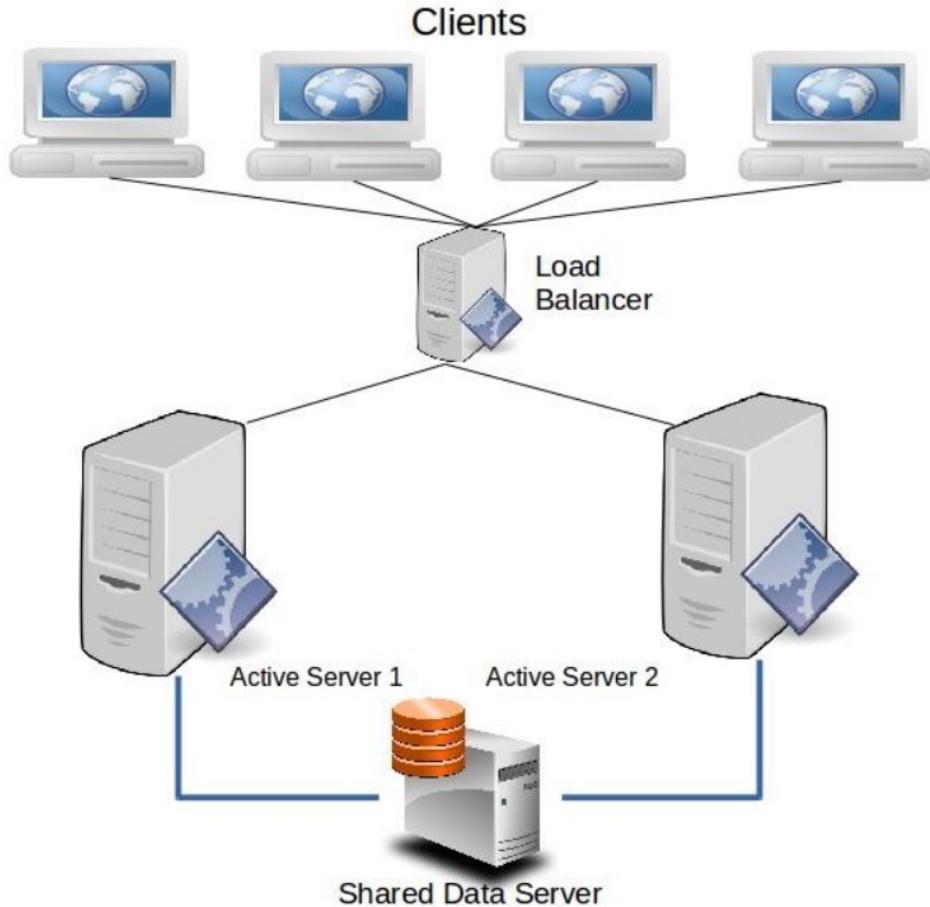


## Active/Passive Cluster - Failover

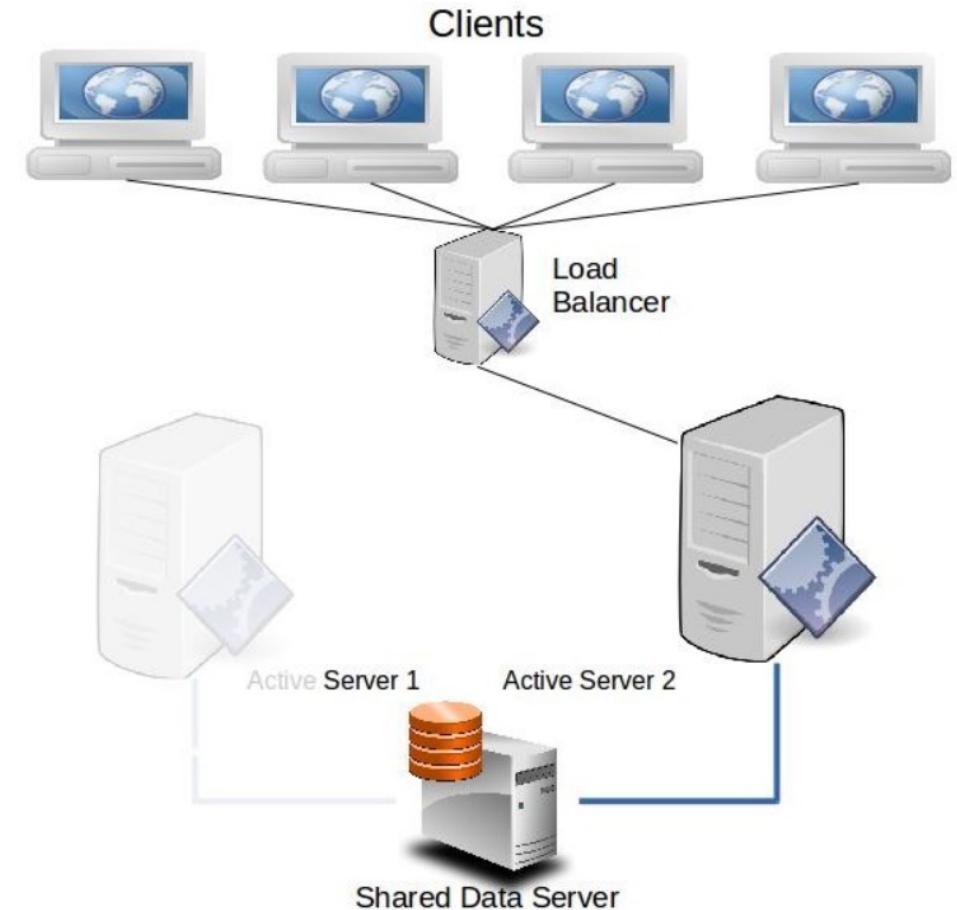


# High Availability 2/3

## Active/Active Cluster

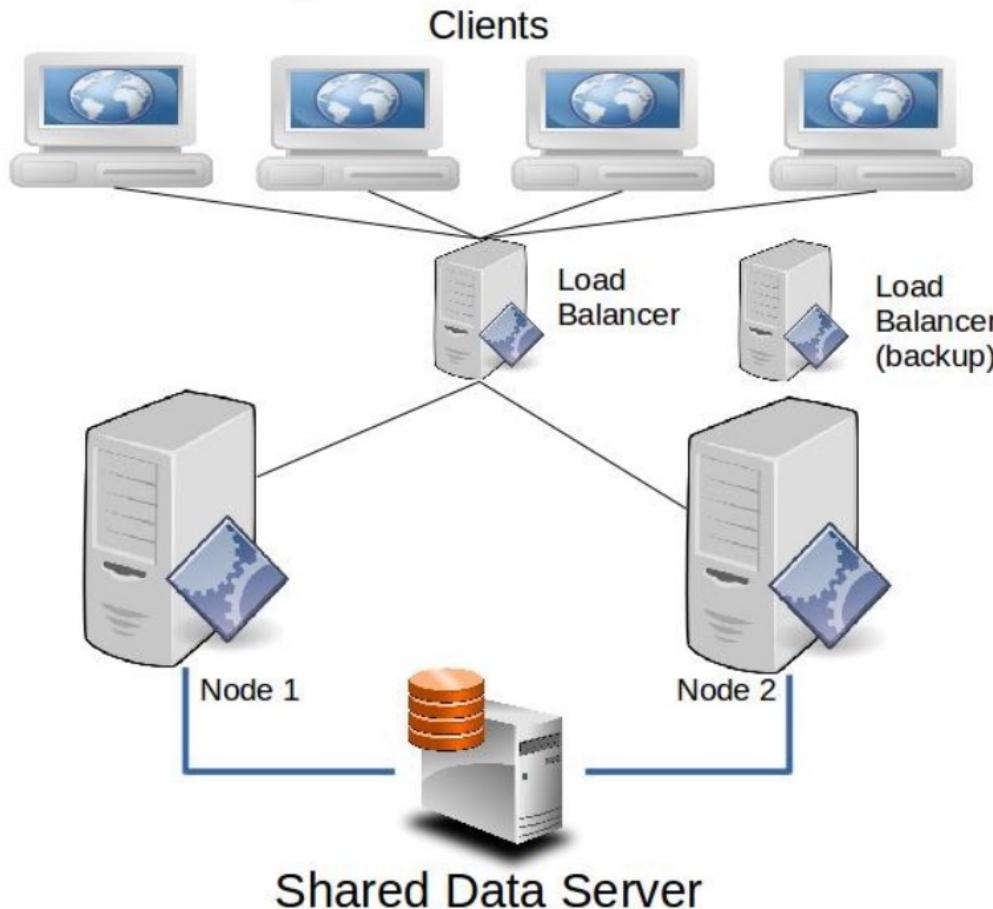


## Active/Active Cluster - Failure

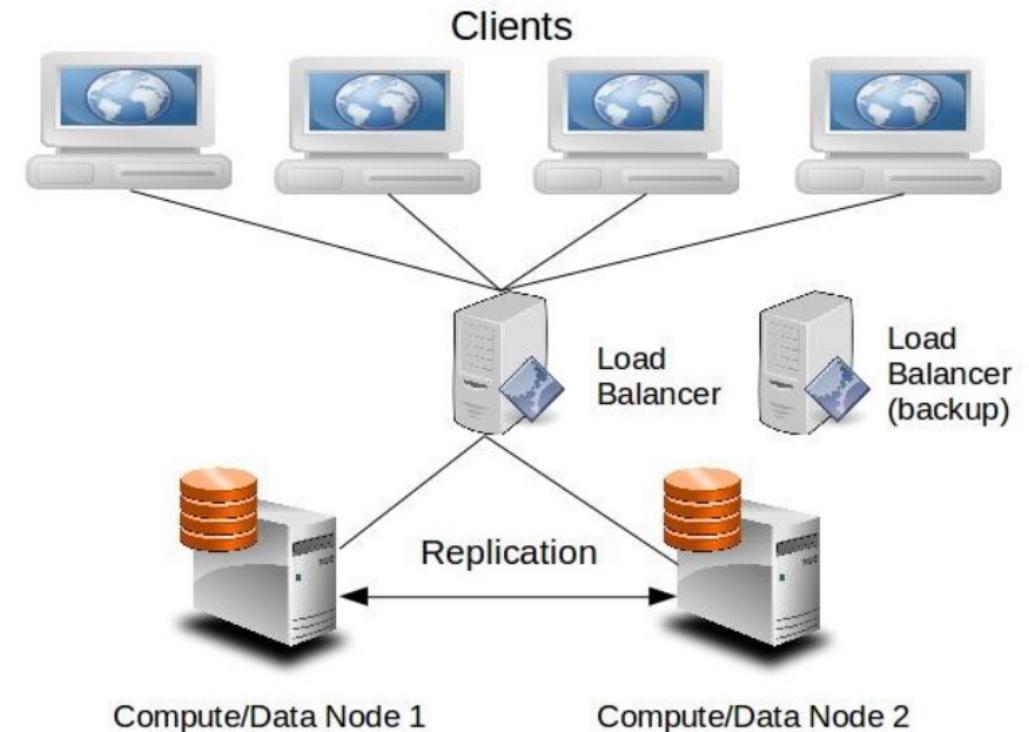


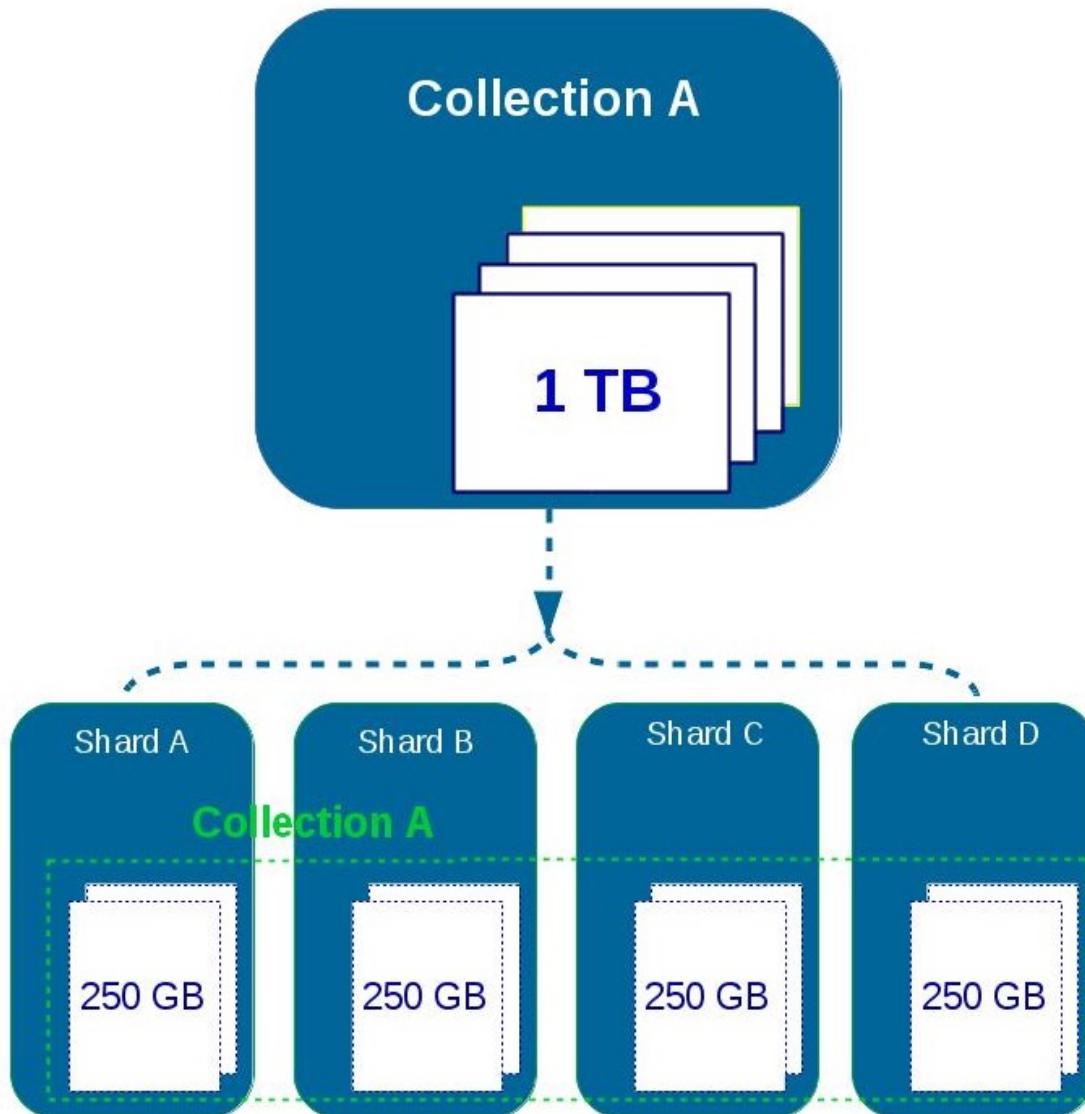
# High Availability 3/3

Single Point of Failure



No Single Point of Failure

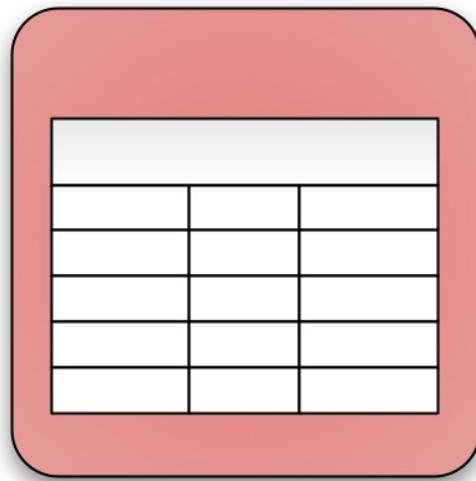




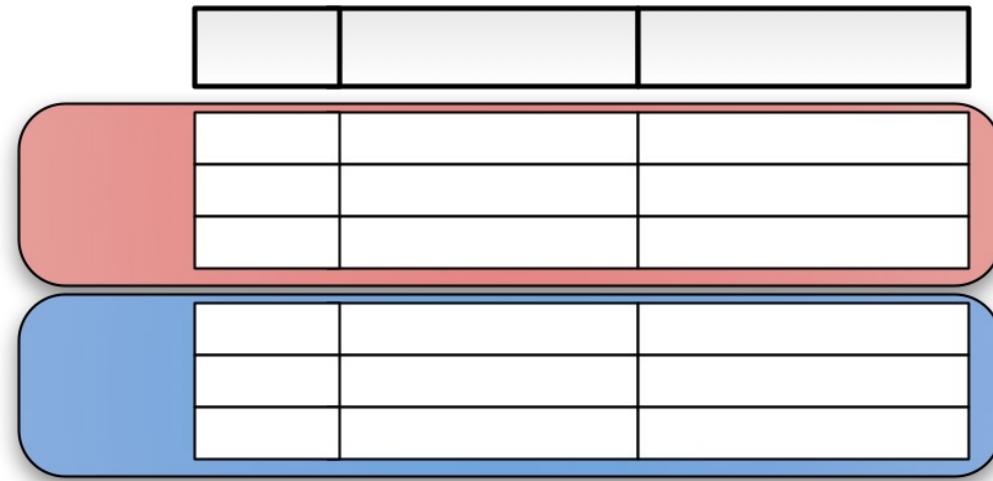
# Partitioning (Sharding)

- Splitting up the data between multiple servers
- Together they represent the entire collection
- Allows for better throughput as calculations and search are shared
- Makes sense for Big Data Applications

# Partitioning (Sharding): Vertical vs Horizontal

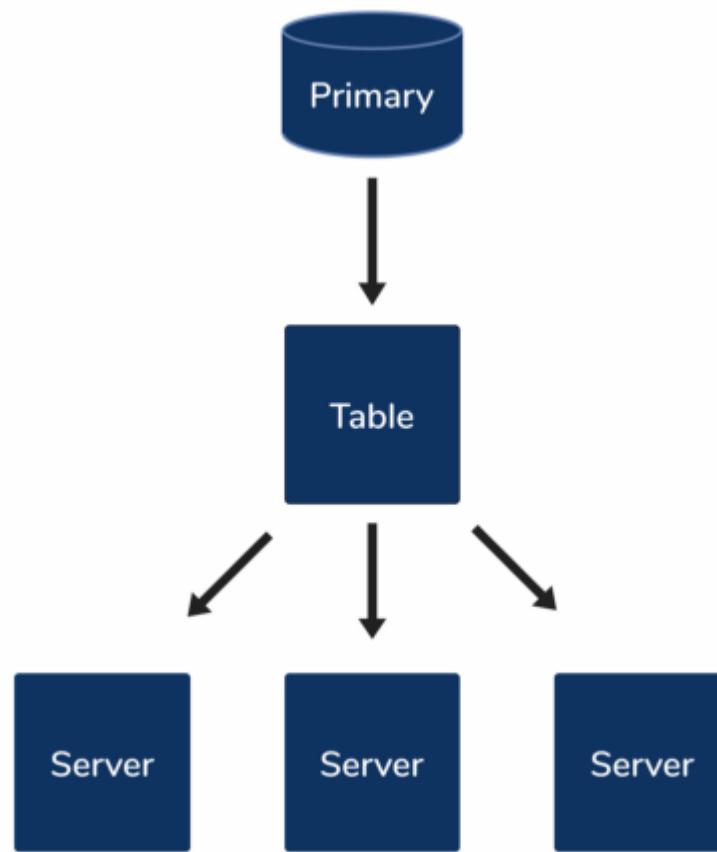
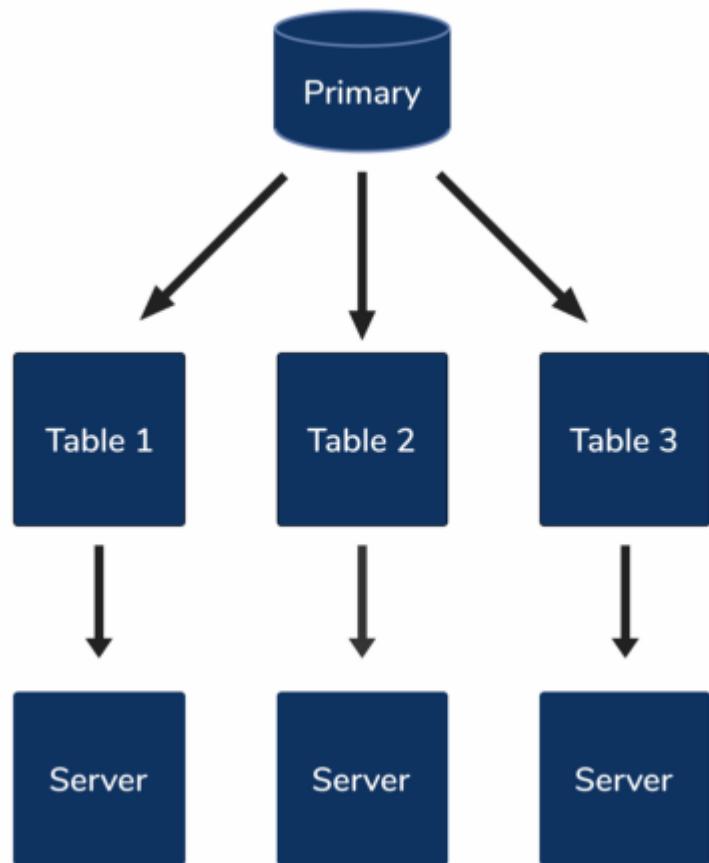


Vertical



Horizontal

# Partitioning (Sharding): Vertical vs Horizontal



# Scaling NoSQL Databases:

## Case 1 - Hbase



# HBase

Based on Google Big Table

- HBase is a column-oriented non-relational database management system that runs on top of Hadoop Distributed File System (HDFS).
- An HBase system is designed to scale linearly.
- It comprises a set of standard tables with rows and columns, much like a traditional database.
- HBase works well with Hive, a query engine for batch processing of big data, to enable fault-tolerant big data applications.

# HBase Data Model

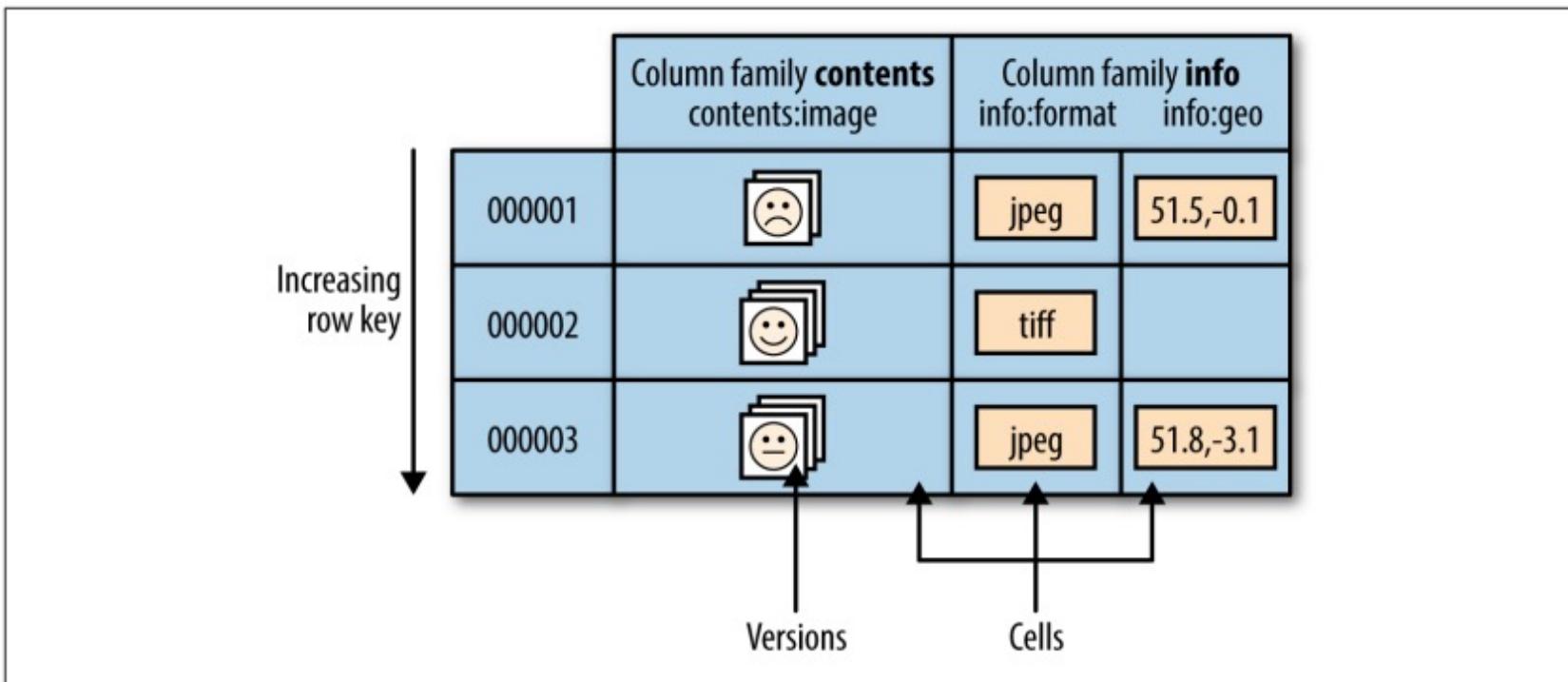


Figure 20-1. The HBase data model, illustrated for a table storing photos

# HBase Cluster

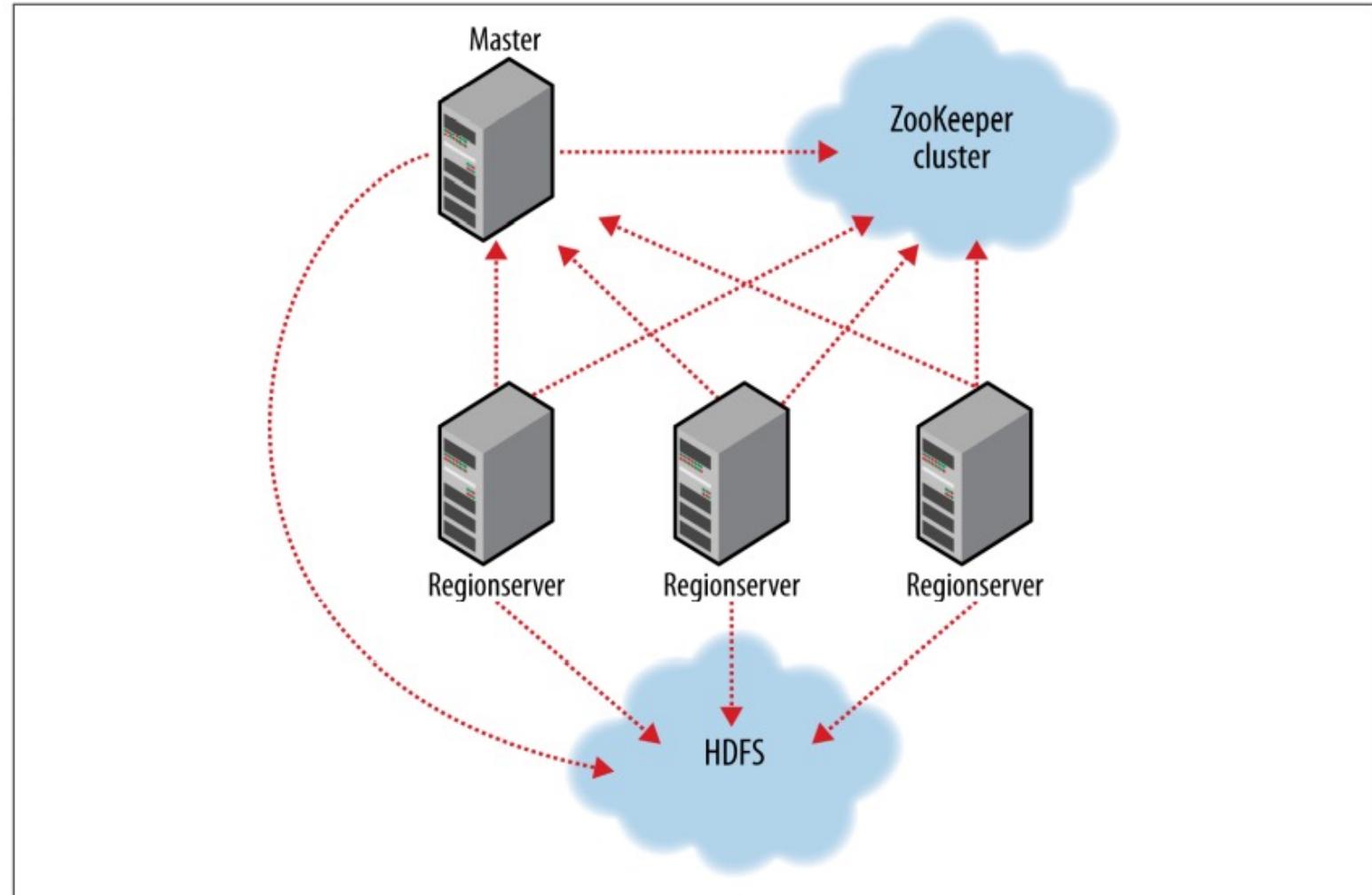
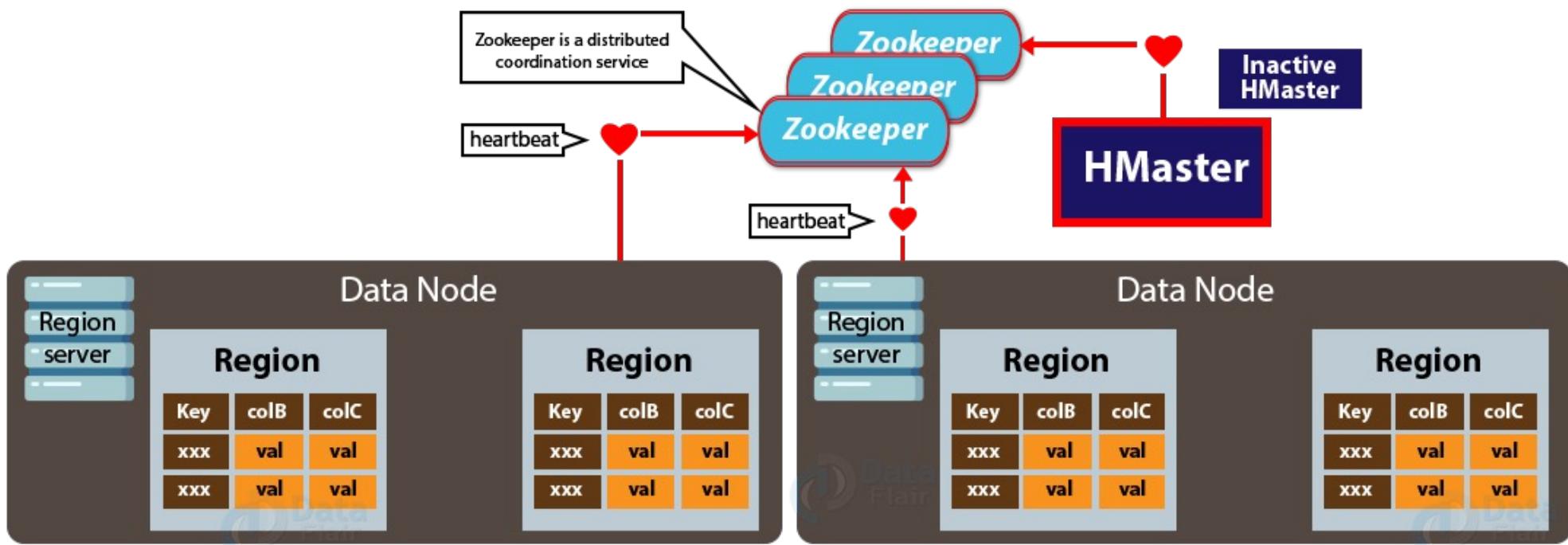


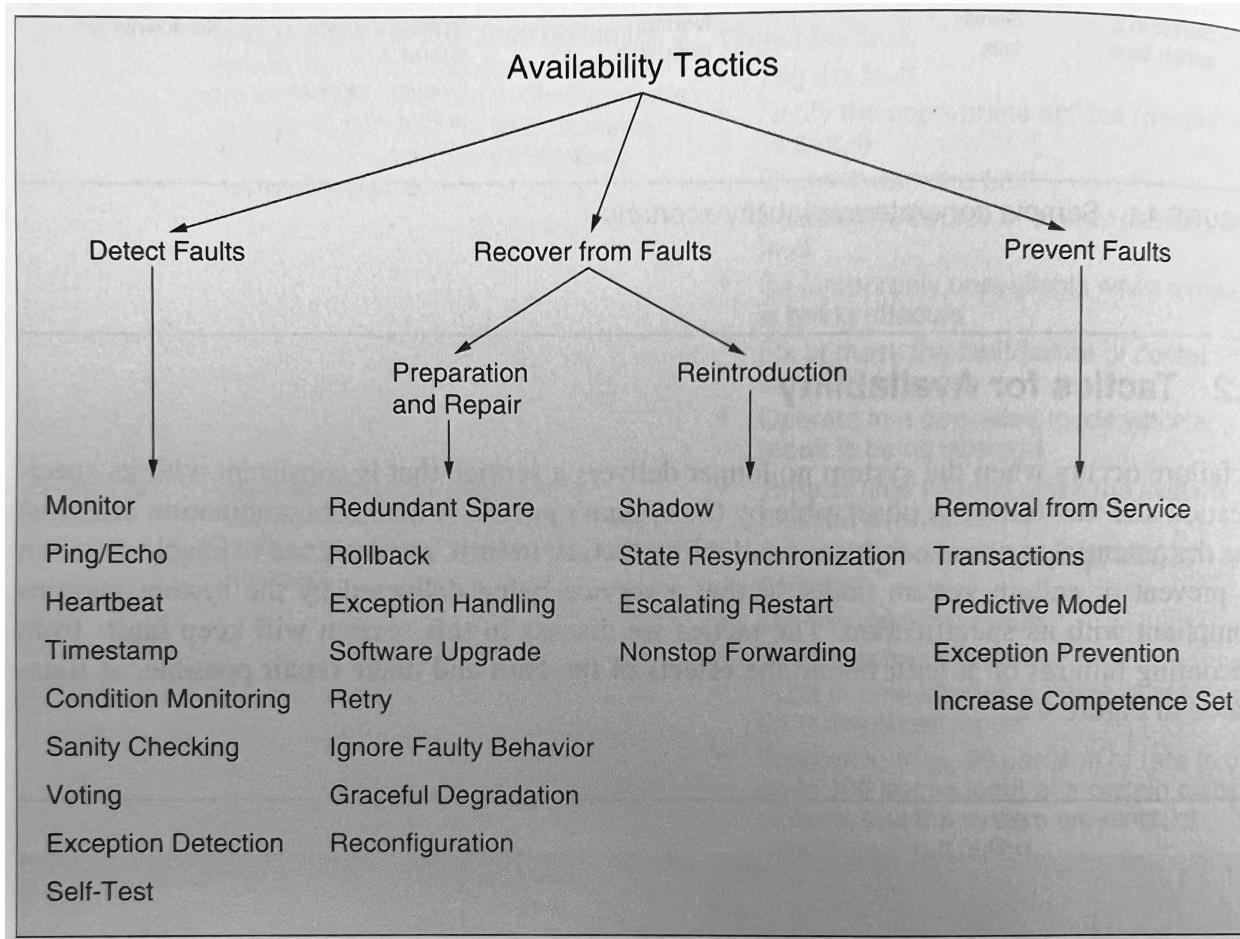
Figure 20-2. HBase cluster members

# HBase Architecture

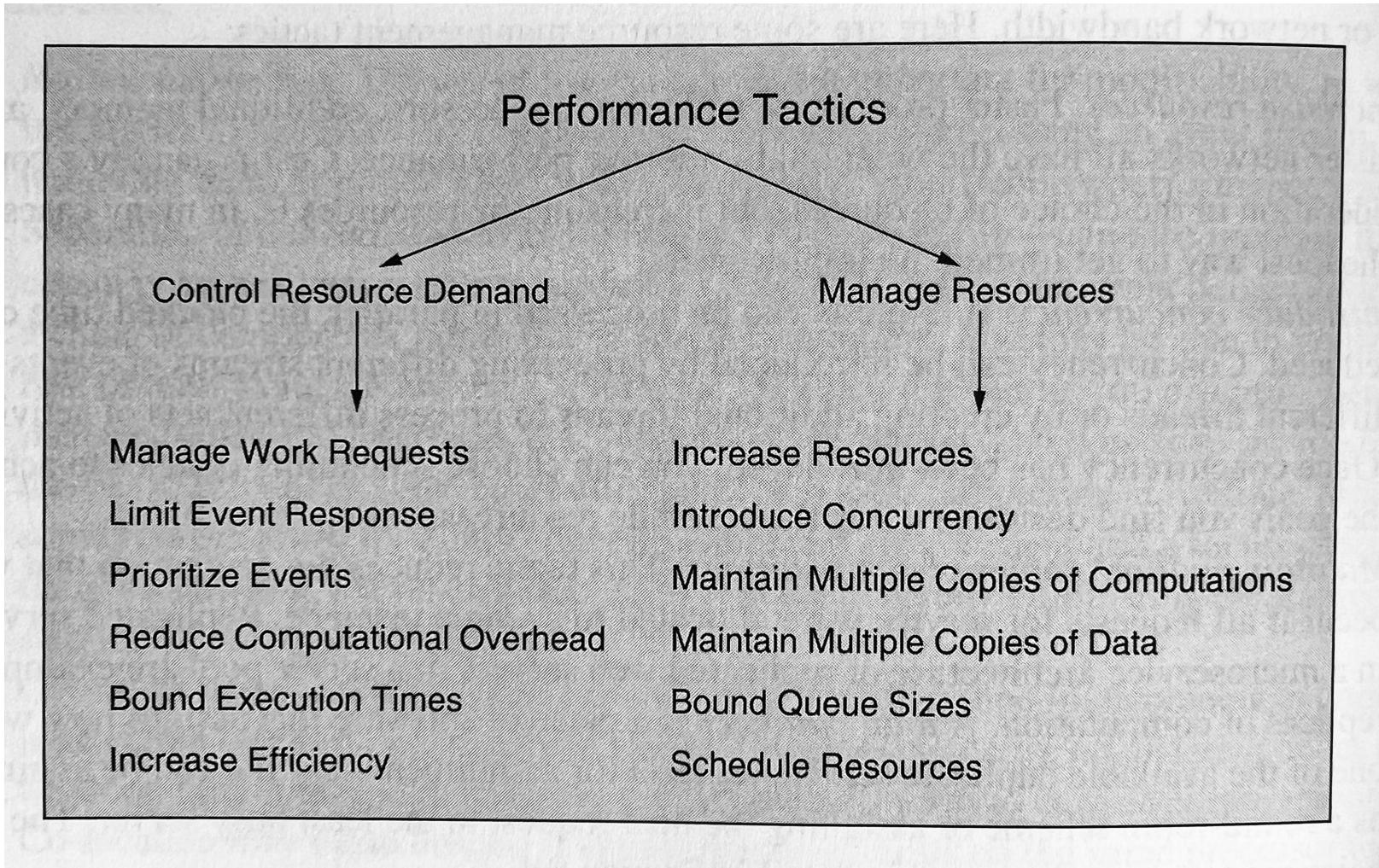


Source: <https://data-flair.training/blogs/hbase-architecture/>  
Go here to get a great overview of the Hbase architecture!!!

# Availability Tactics



# Performance

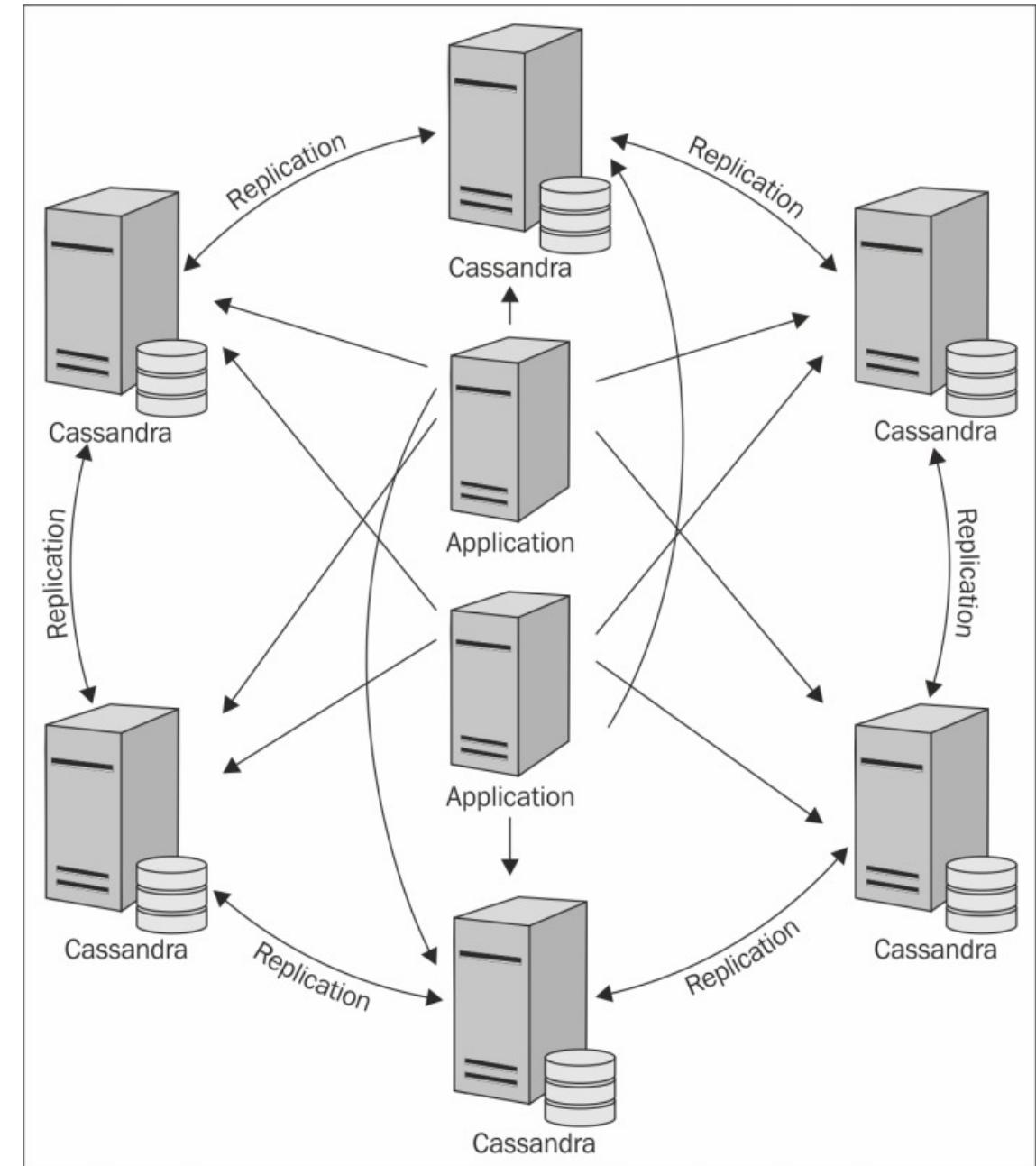


# Scaling NoSQL Databases:

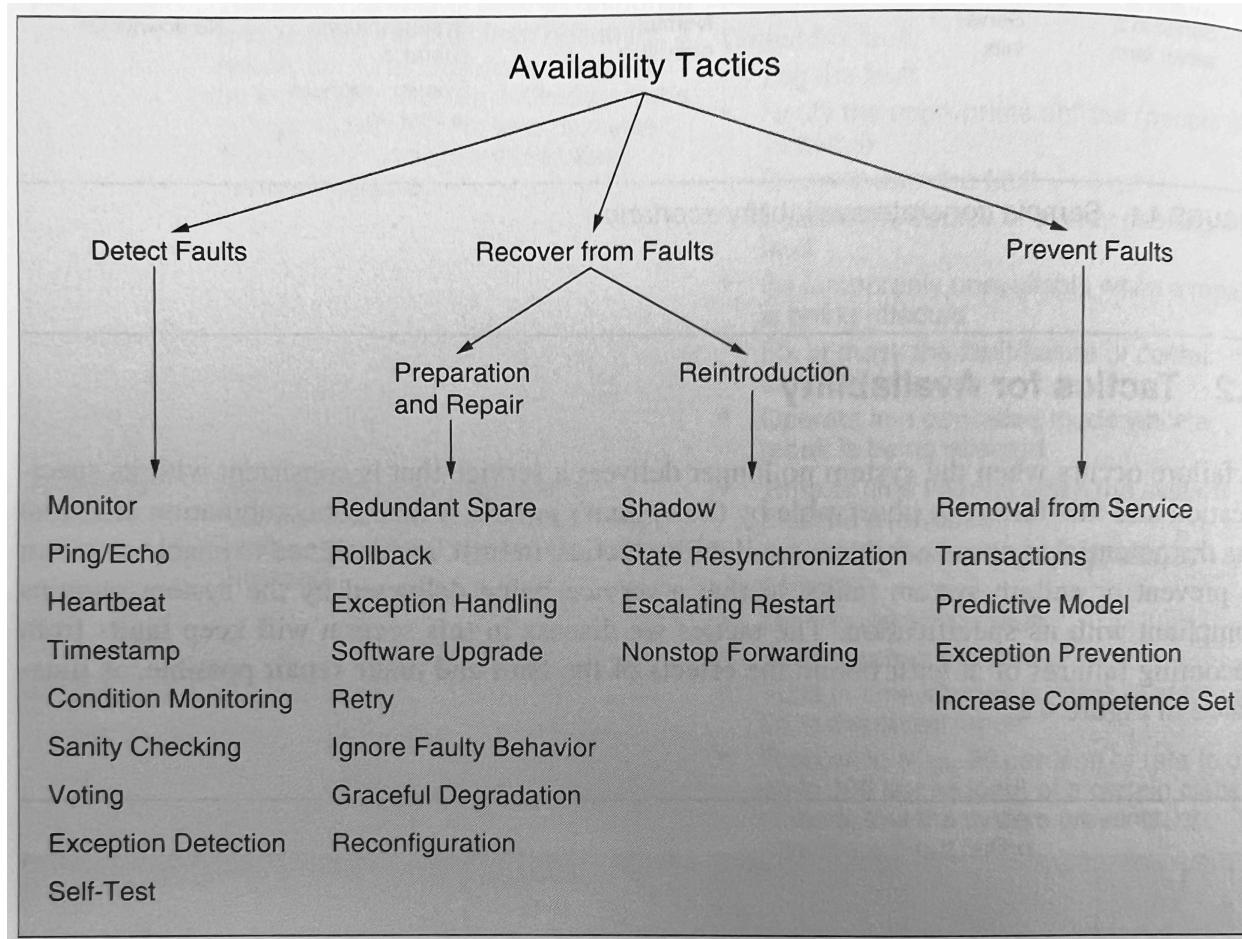
## Case 2 - Cassandra

# What is Cassandra?

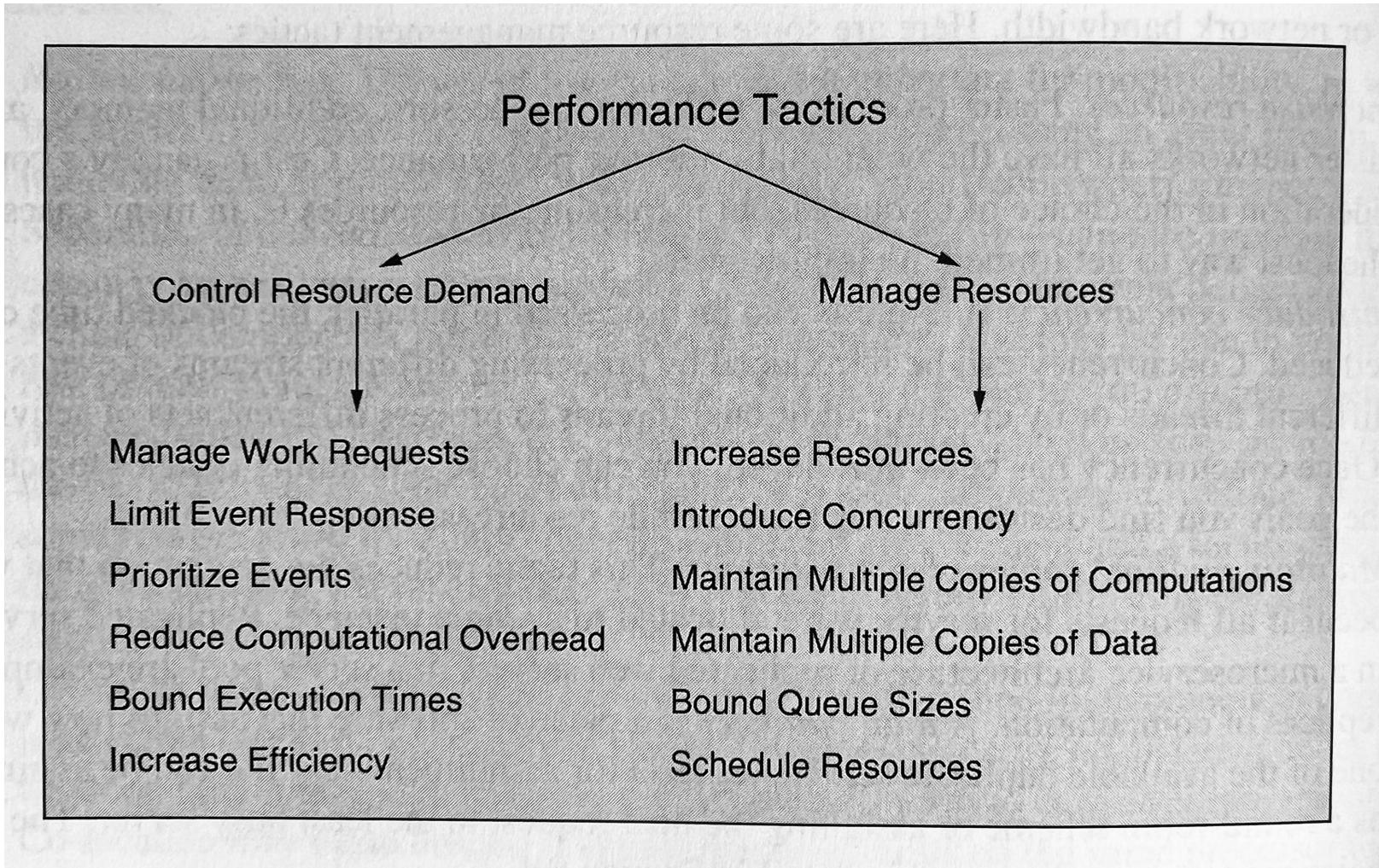
- Also based on Google Big Table
- High Performance Column based Database
- Performance reached through manual definition of tables, and how to store the data, during creation
- Can Write to any instance
- ACK sent when replication is done (based on replication factor set)
- Hbase replicates blocks using Hadoop HDFS, while Cassandra takes care of the replication factor itself using the Gossip Protocol



# Availability Tactics



# Performance

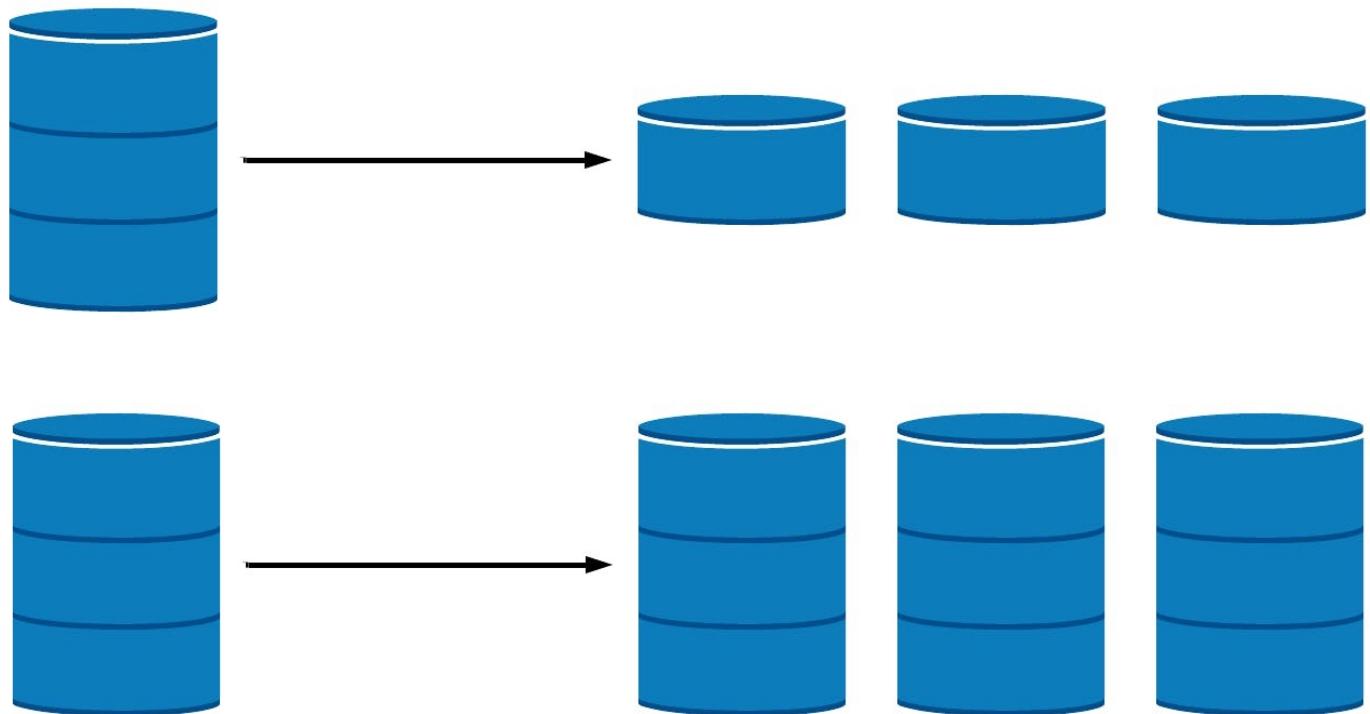


# Scaling NoSQL Databases:

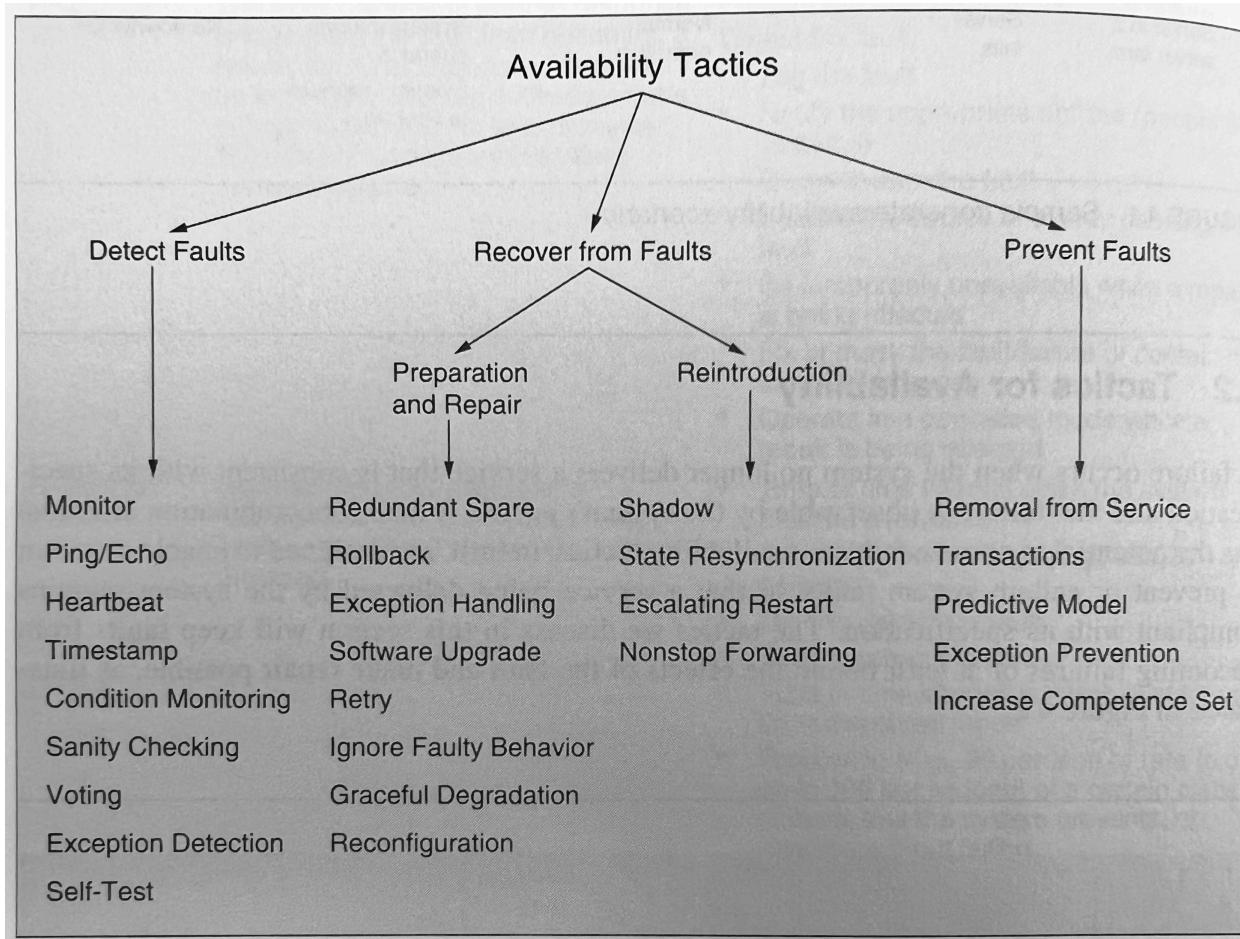
## Case 3 - MongoDB

# What is MongoDB?

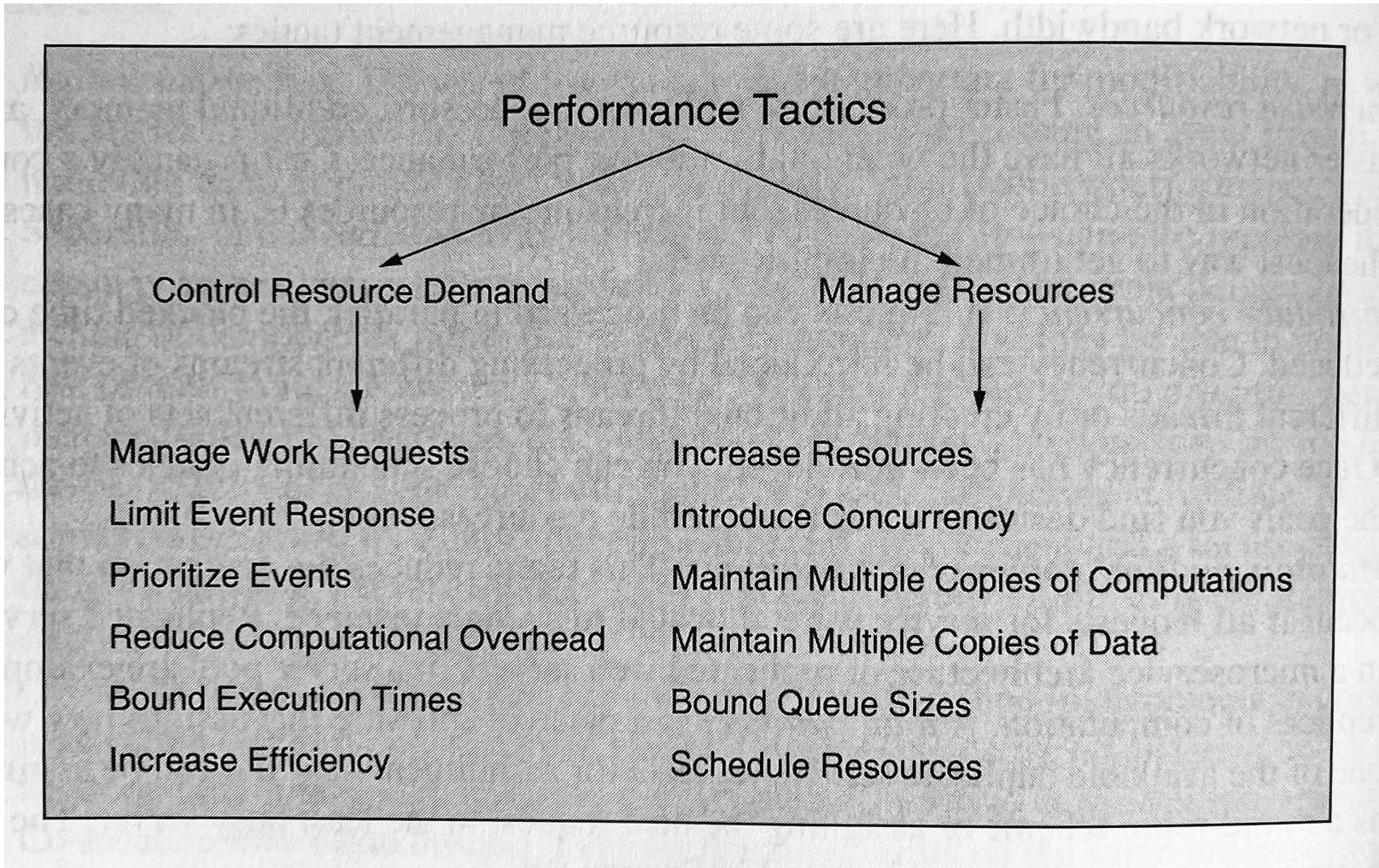
- Strategies
  - Sharding
- Replica Sets
  - HA implementation
  - Great for total read redundancy
  - Potential issues with large amounts of writes due to propagation
- Document based NoSQL Database
- Data format: JSON



# Availability Tactics



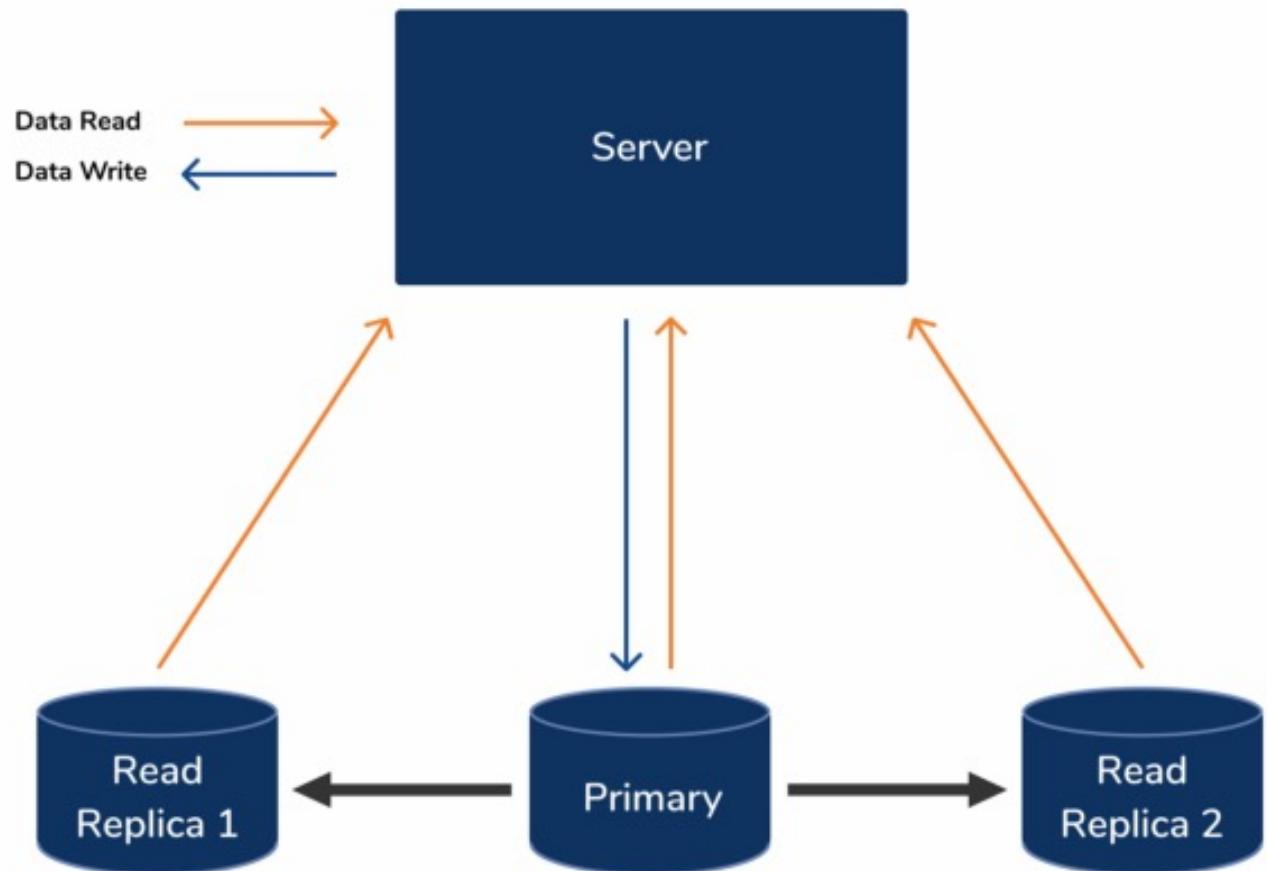
# Performance



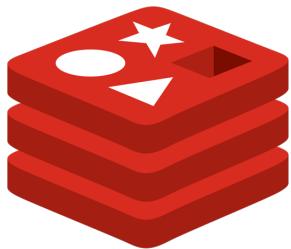
# Scaling Relational Databases

# Scaling Relational Databases

- Not easy to keep consistency
- Typical Strategy:
  - One master for writes
  - Replicas for Reads
- Backup Strategy:
  - Multiple Separate Relational Databases
  - Splitting Data between nodes, sacrificing constraints
- **Why does most Relational Databases scale badly?**
- What are the tradeoffs?

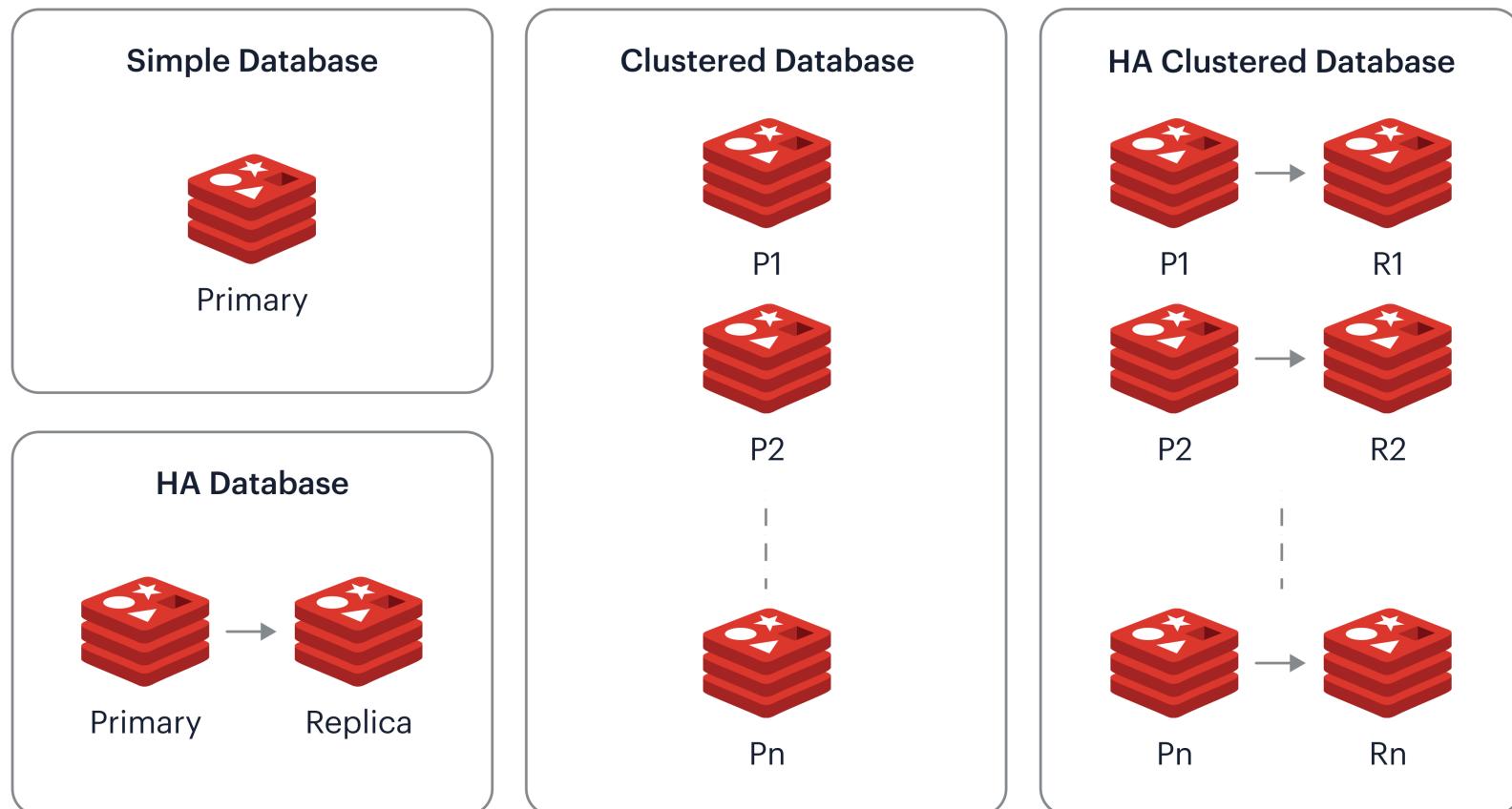


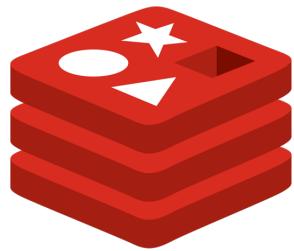
# Bonus: Redis



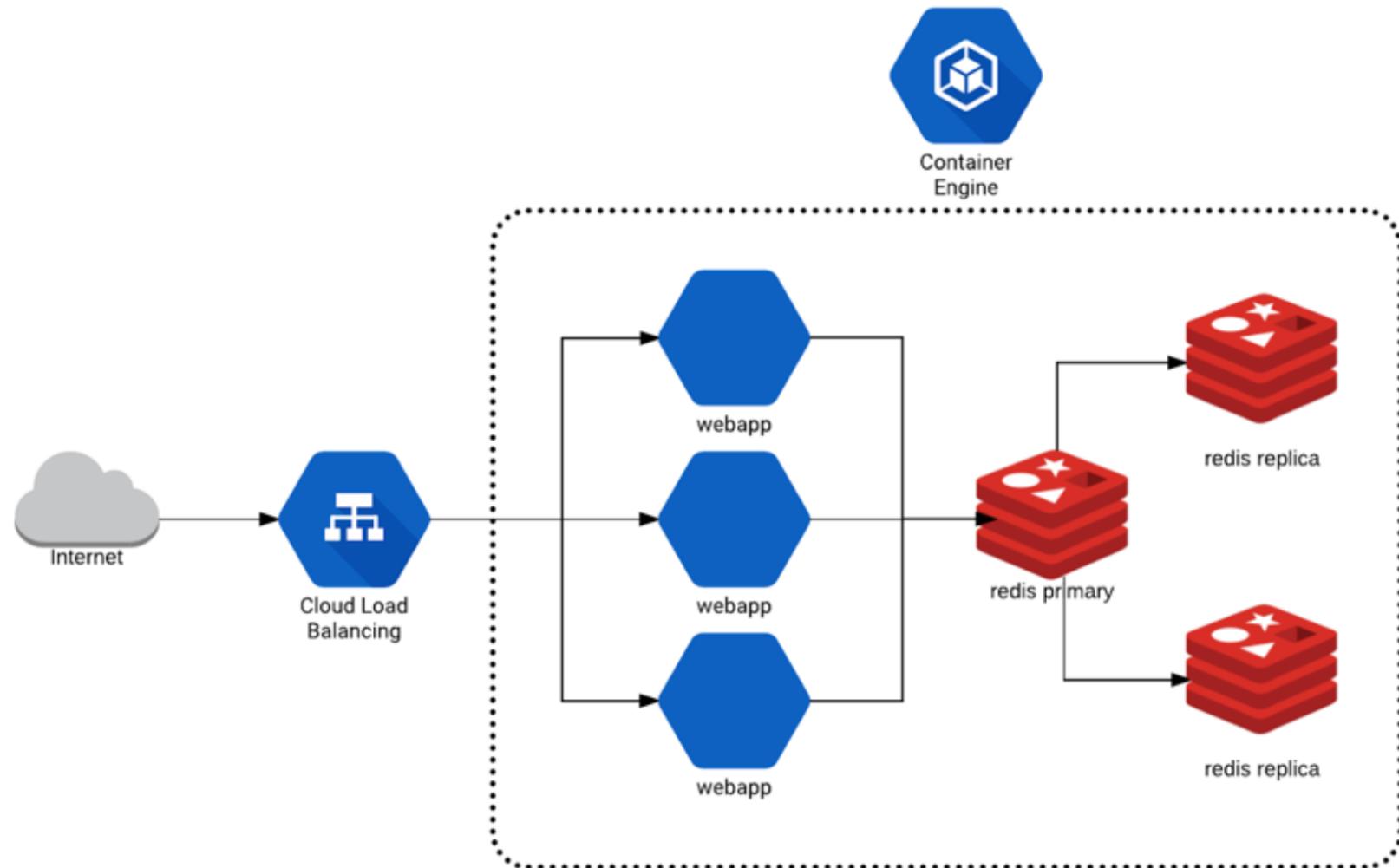
# redis

- Super fast in-memory key value-based NoSQL database
  - Transactions
  - Pub/Sub
  - Keys with a limited time-to-live
  - Automatic failover
- 
- Often used to synchronize states between Kubernetes Pods as it is fast, resilient, and configurable





# redis



# Exercises!