Distributed Data Processing & Distributed Databases

SDU

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

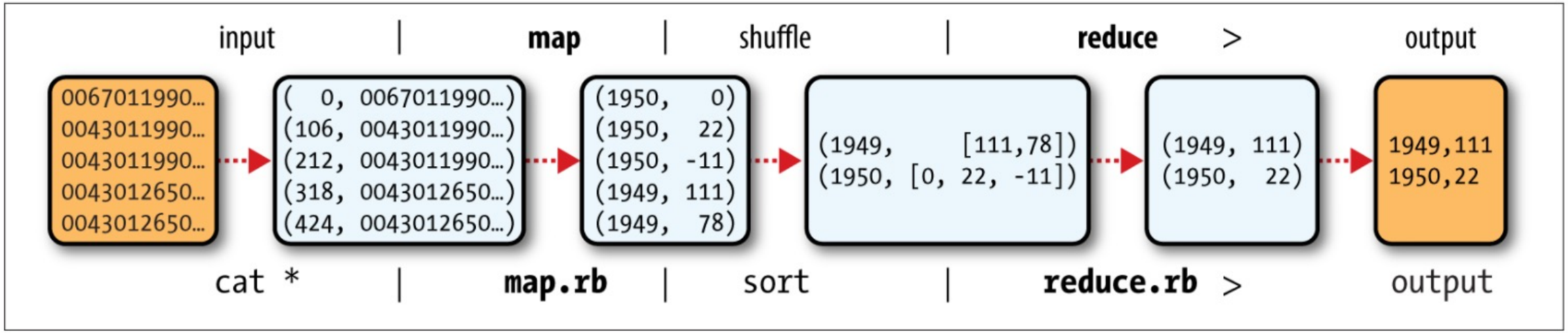


Figure 2-1. MapReduce logical data flow

→ 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**

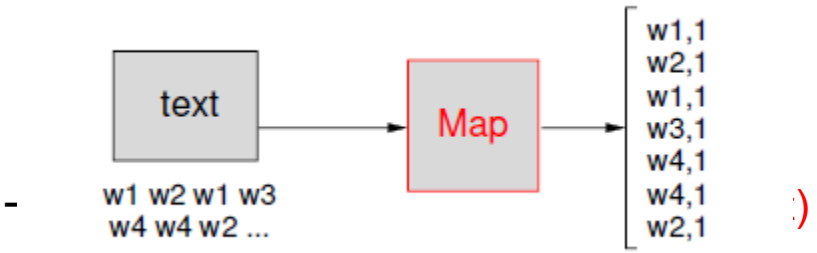
Given n files, compute the global count of each word.

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

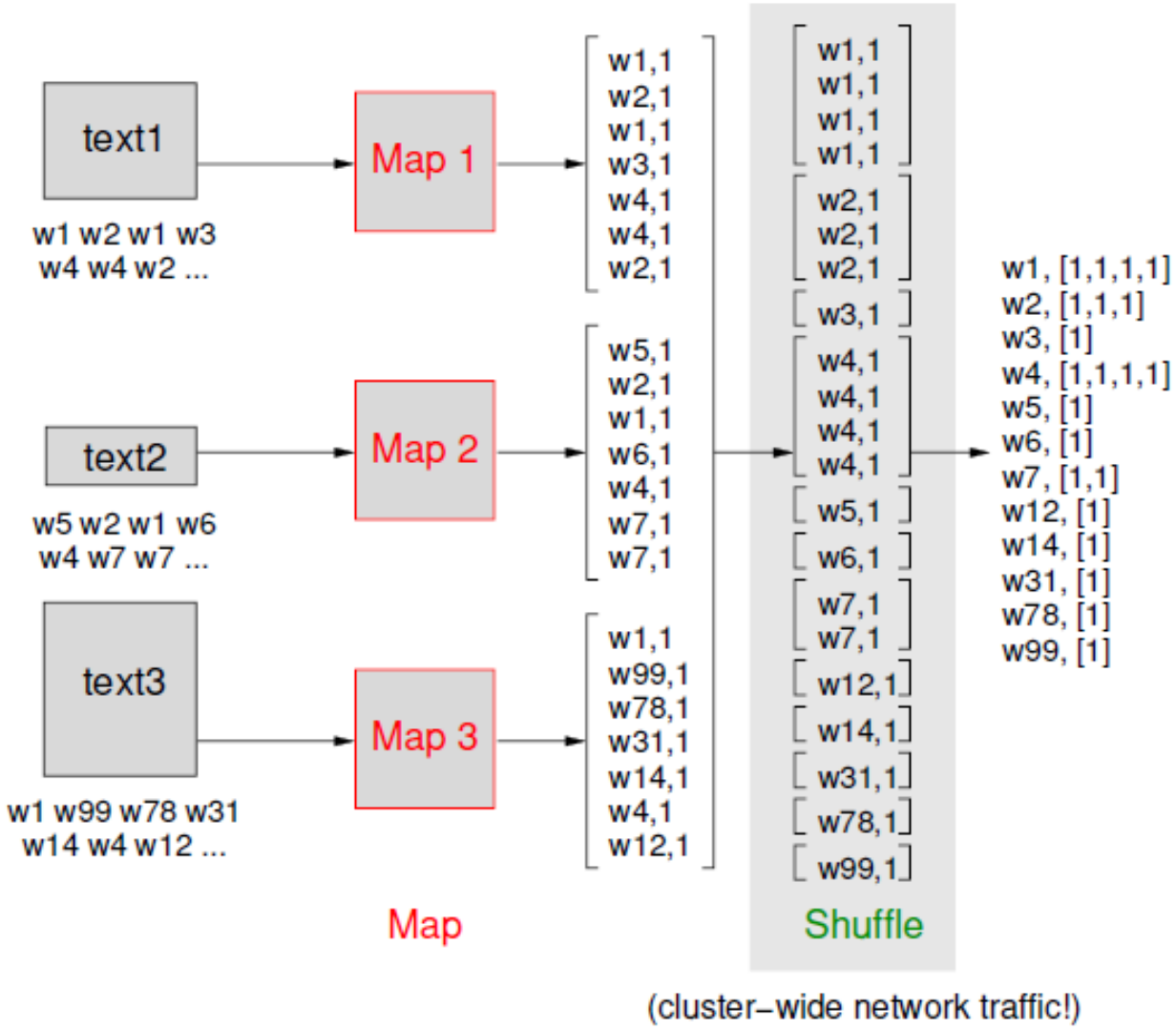
File f_9: $w_1\ w_3\ w_1$ ...

Output: $w_1 : 5, w_3 : 3$ ...

→ Map input: (filename, text)



Multiple Map tasks



Master controller (Shuffle):

→ Deals with the intermediate data:

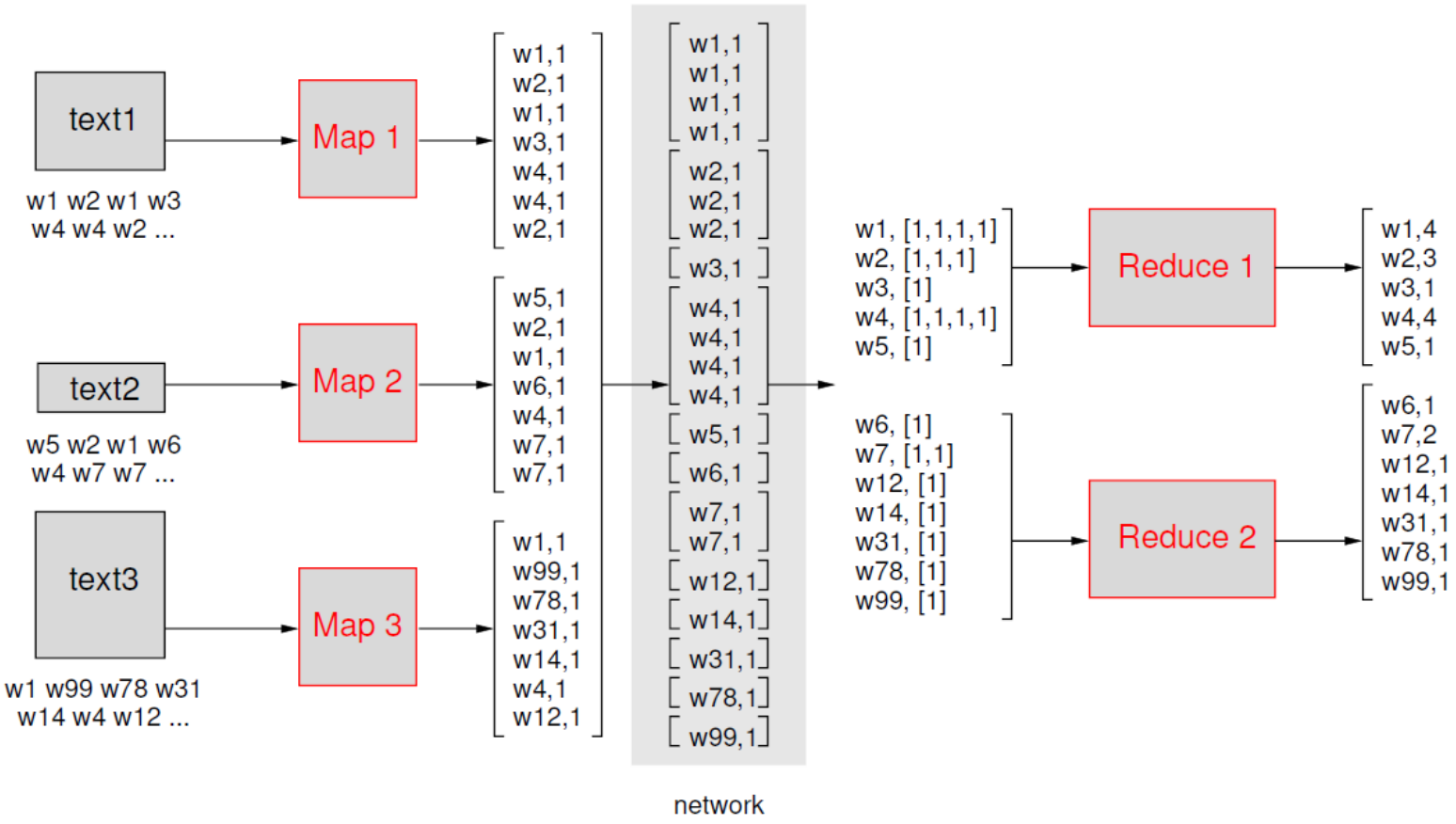
1. Keeps track of the (key, value) pairs in output of all Map tasks.
2. Does a distributed group by key operation.
3. A key and its list of values will be read by a single Reduce task.

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)

Map Reduce Overview

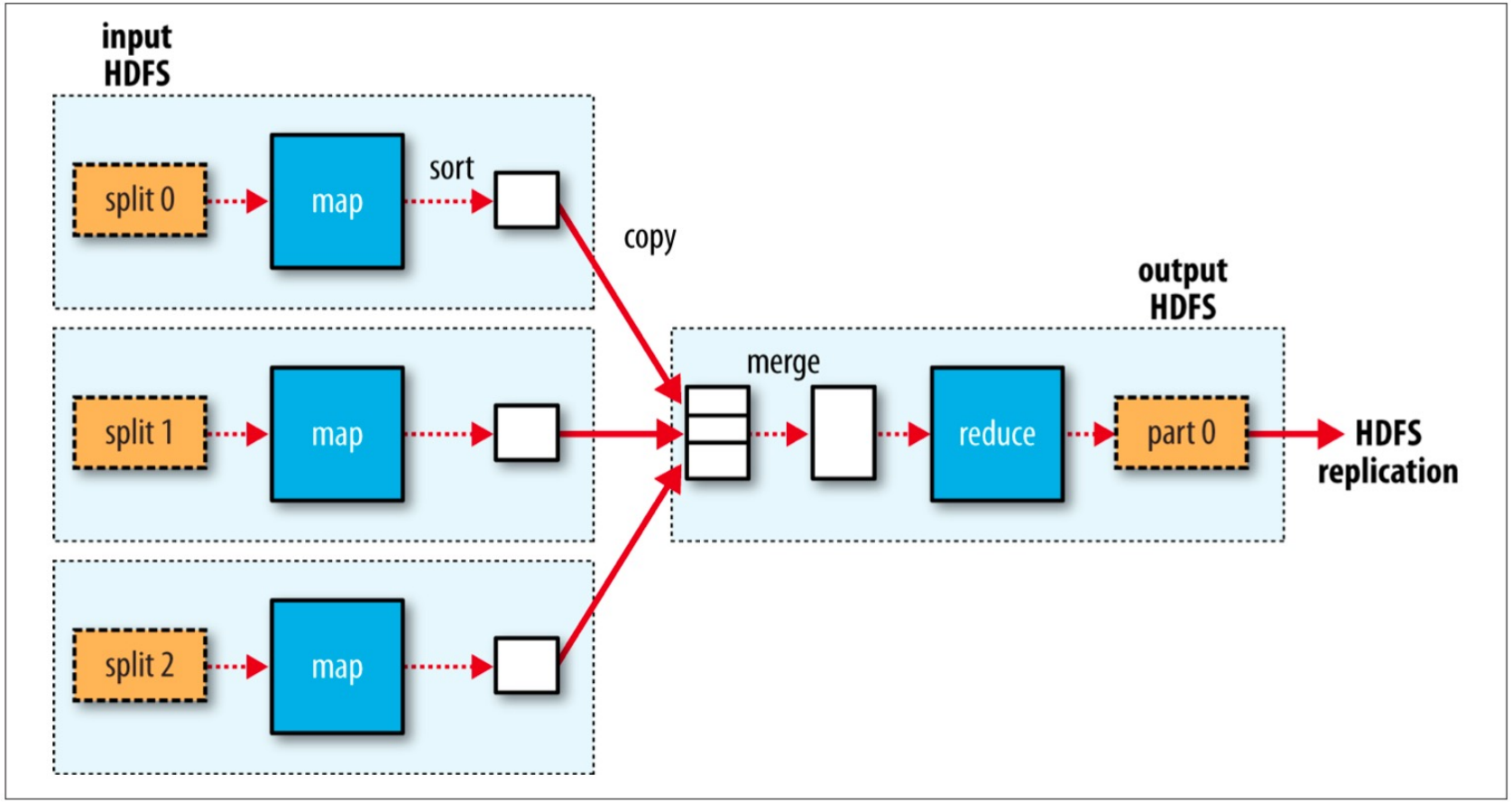


Figure 2-3. MapReduce data flow with a single reduce task

Map Reduce Overview – Multiple Reduce

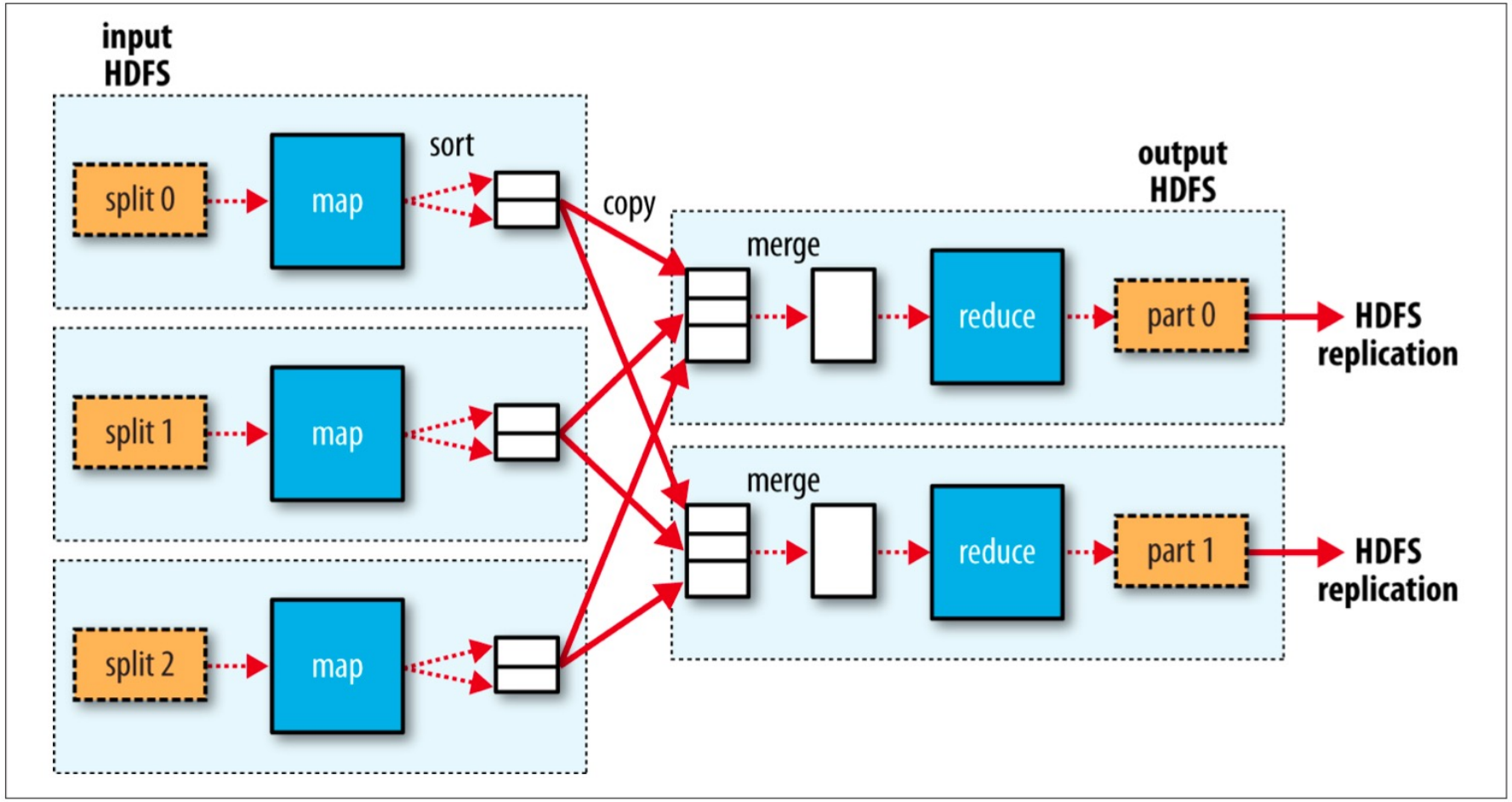


Figure 2-4. MapReduce data flow with multiple reduce tasks

Map Reduce Overview – Map Only

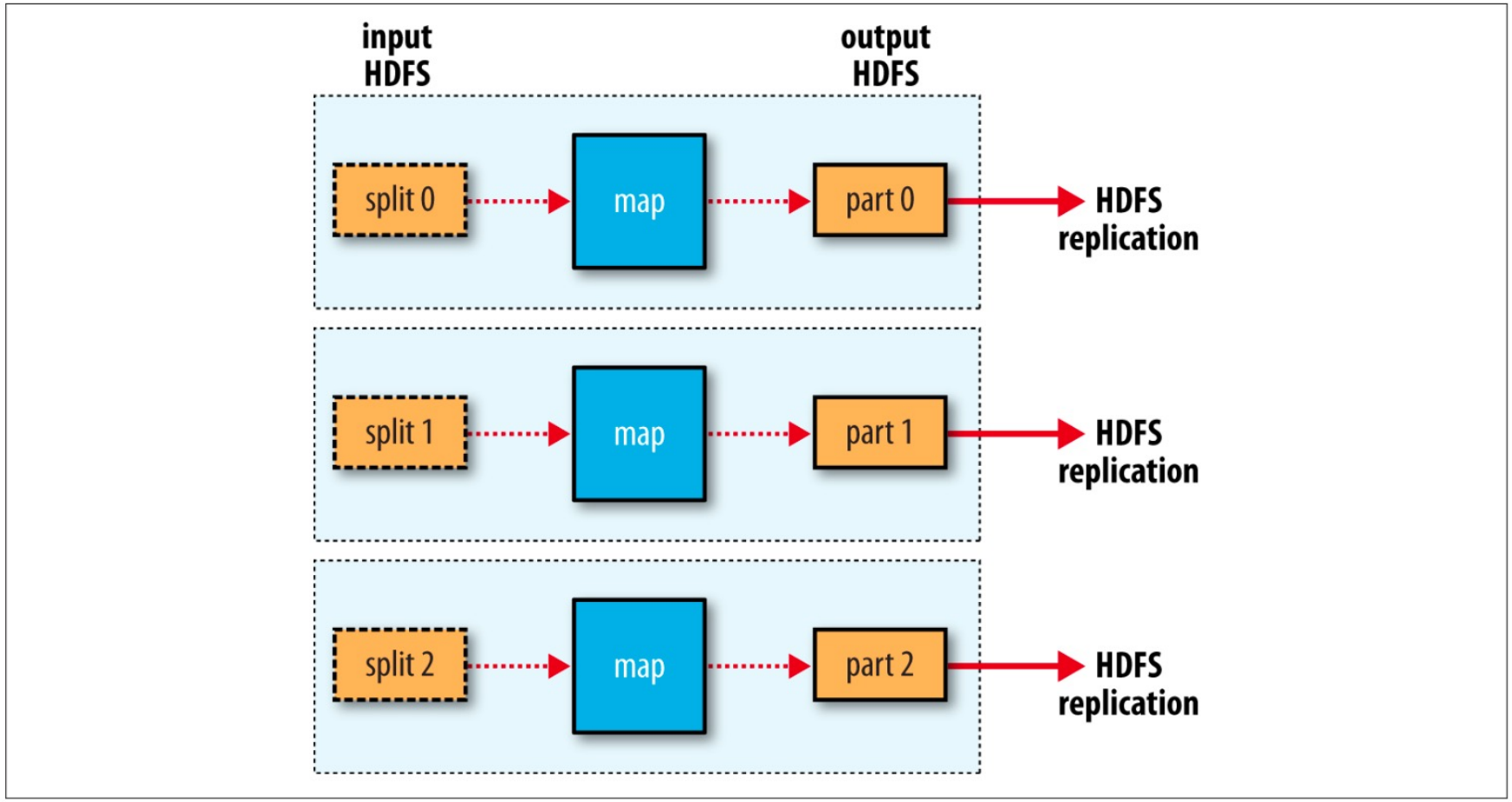


Figure 2-5. MapReduce data flow with no reduce tasks

The Complicated Version

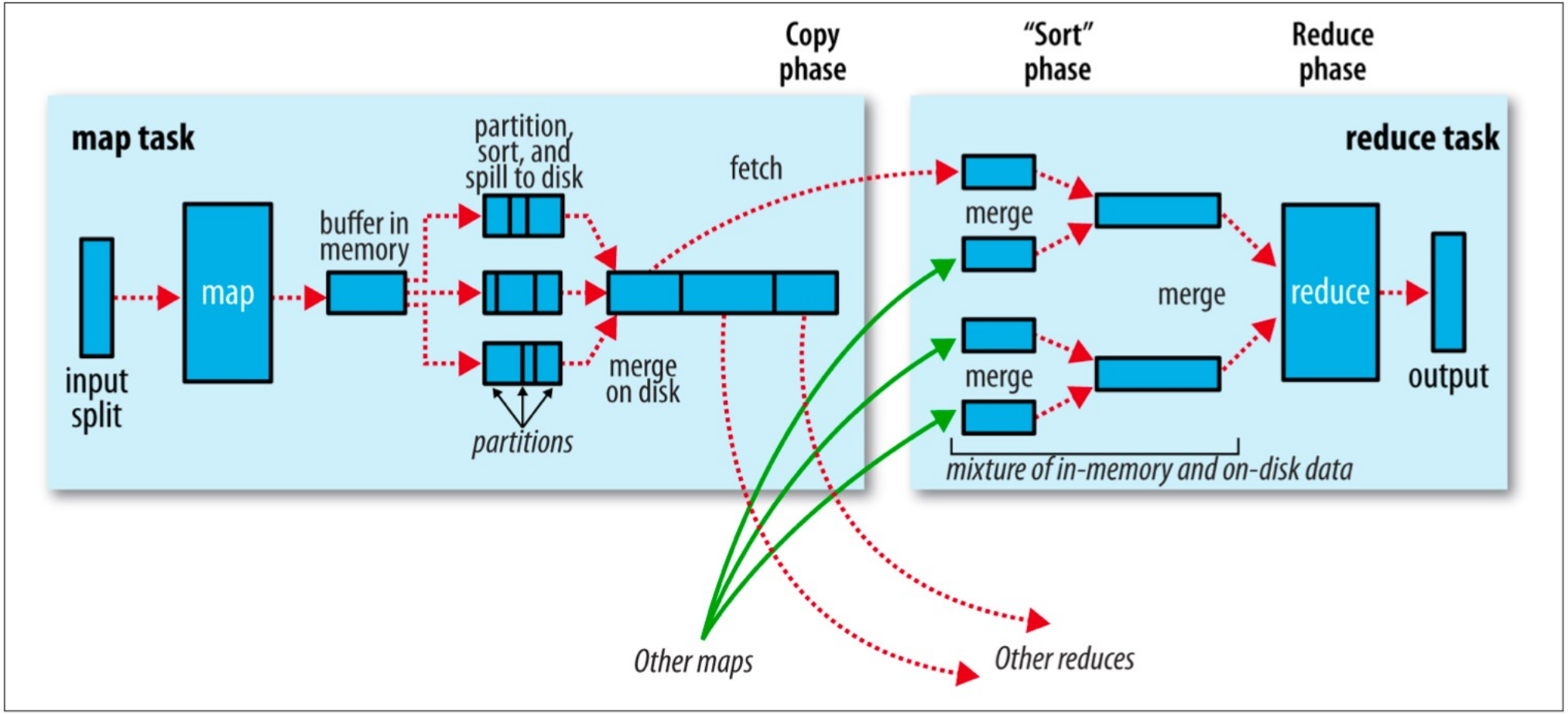


Figure 7-4. Shuffle and sort in MapReduce

Data Acquisition from HDFS

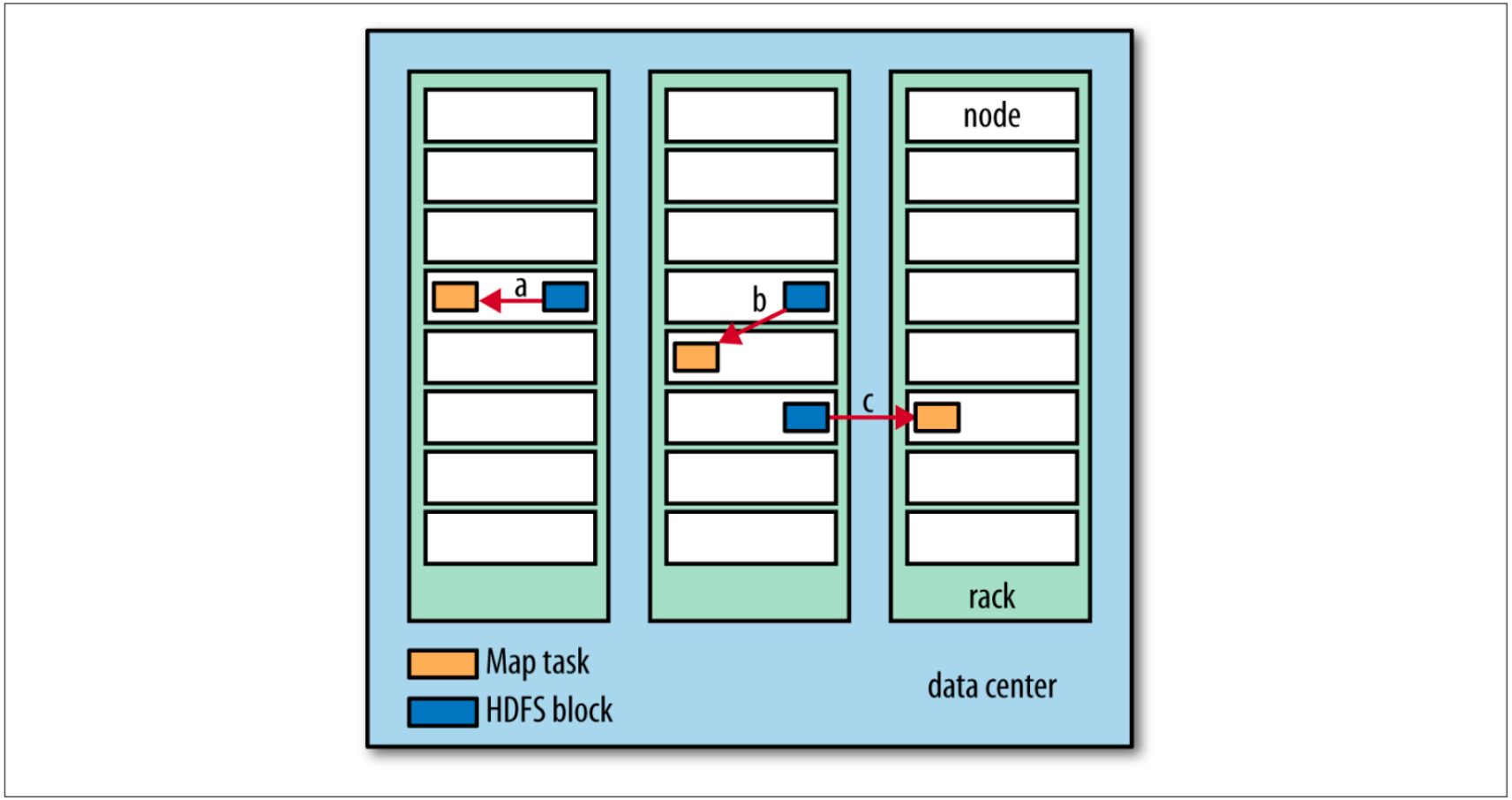


Figure 2-2. Data-local (a), rack-local (b), and off-rack (c) map tasks

Web Overview

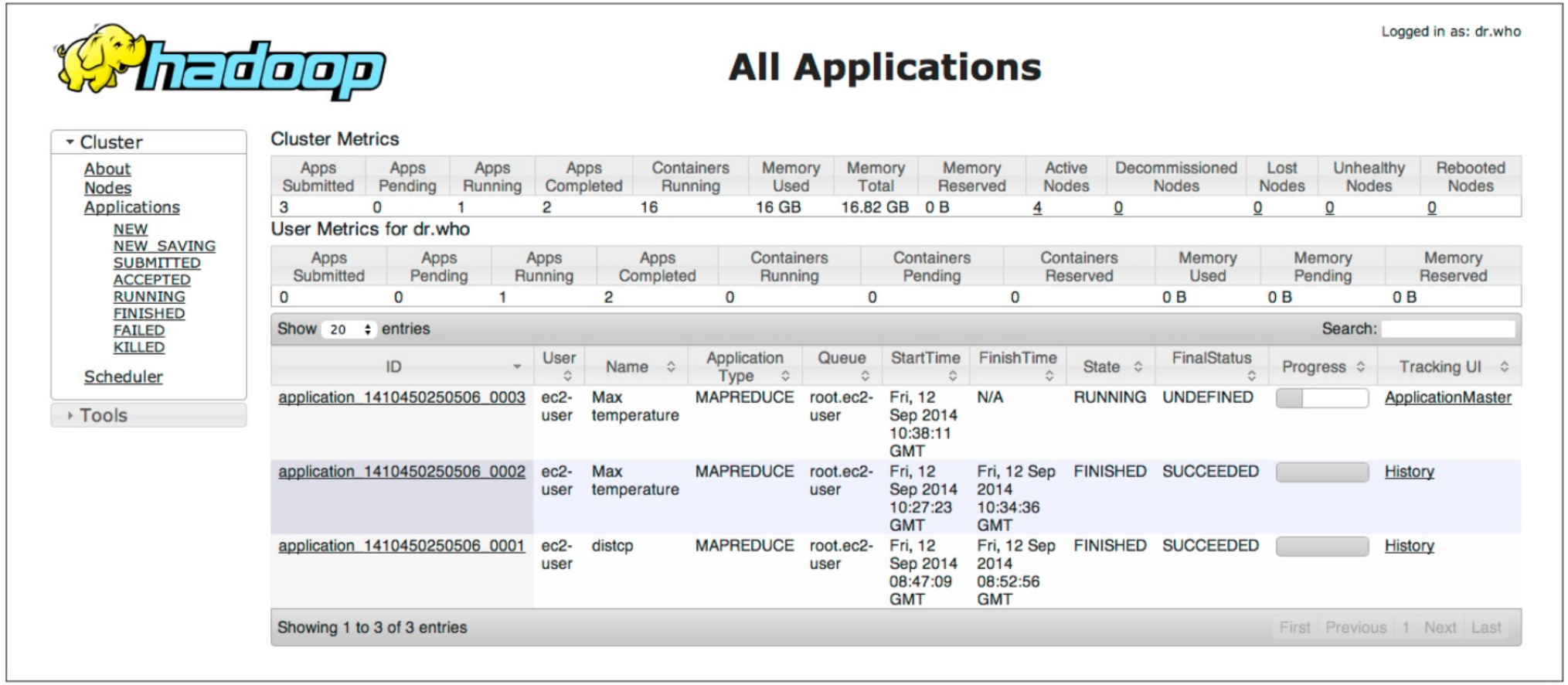


Figure 6-1. Screenshot of the resource manager page

SDU

Web Overview – Single Task

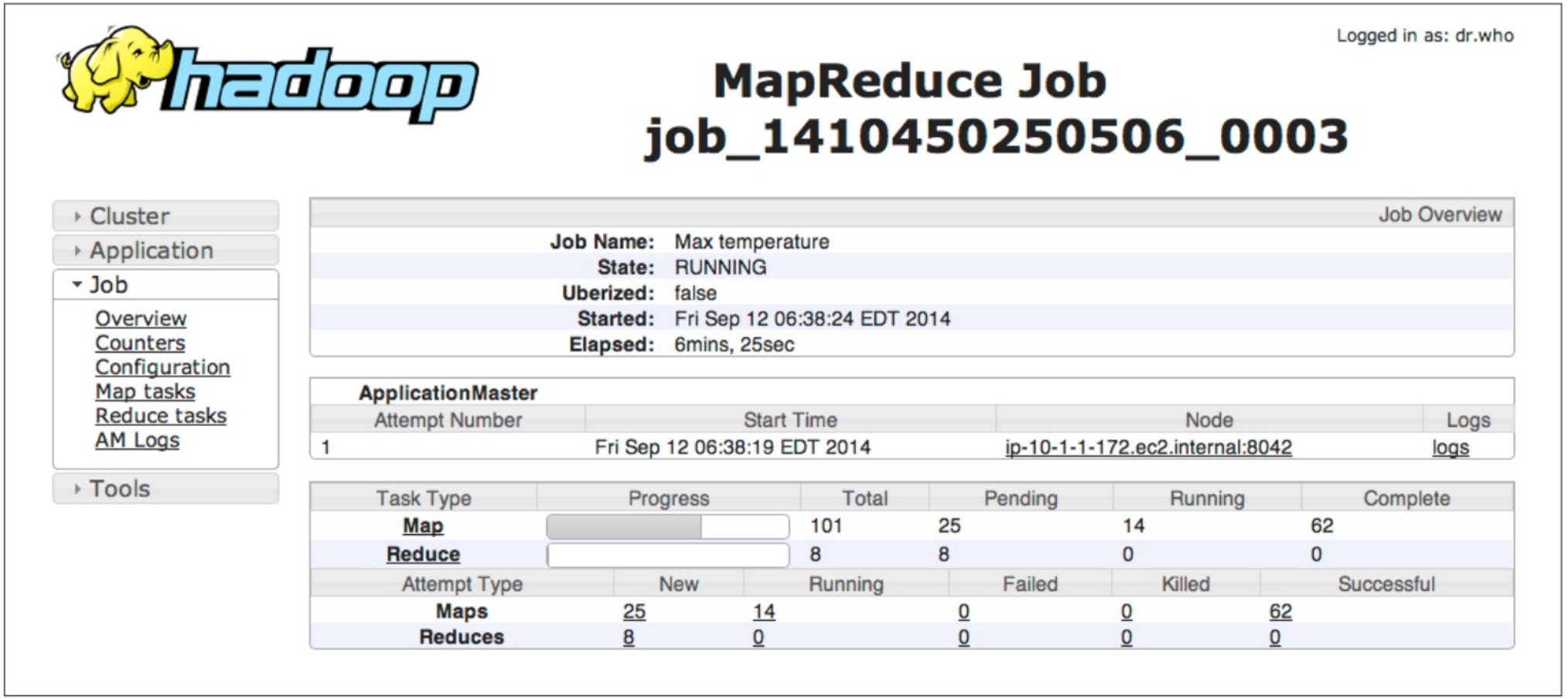
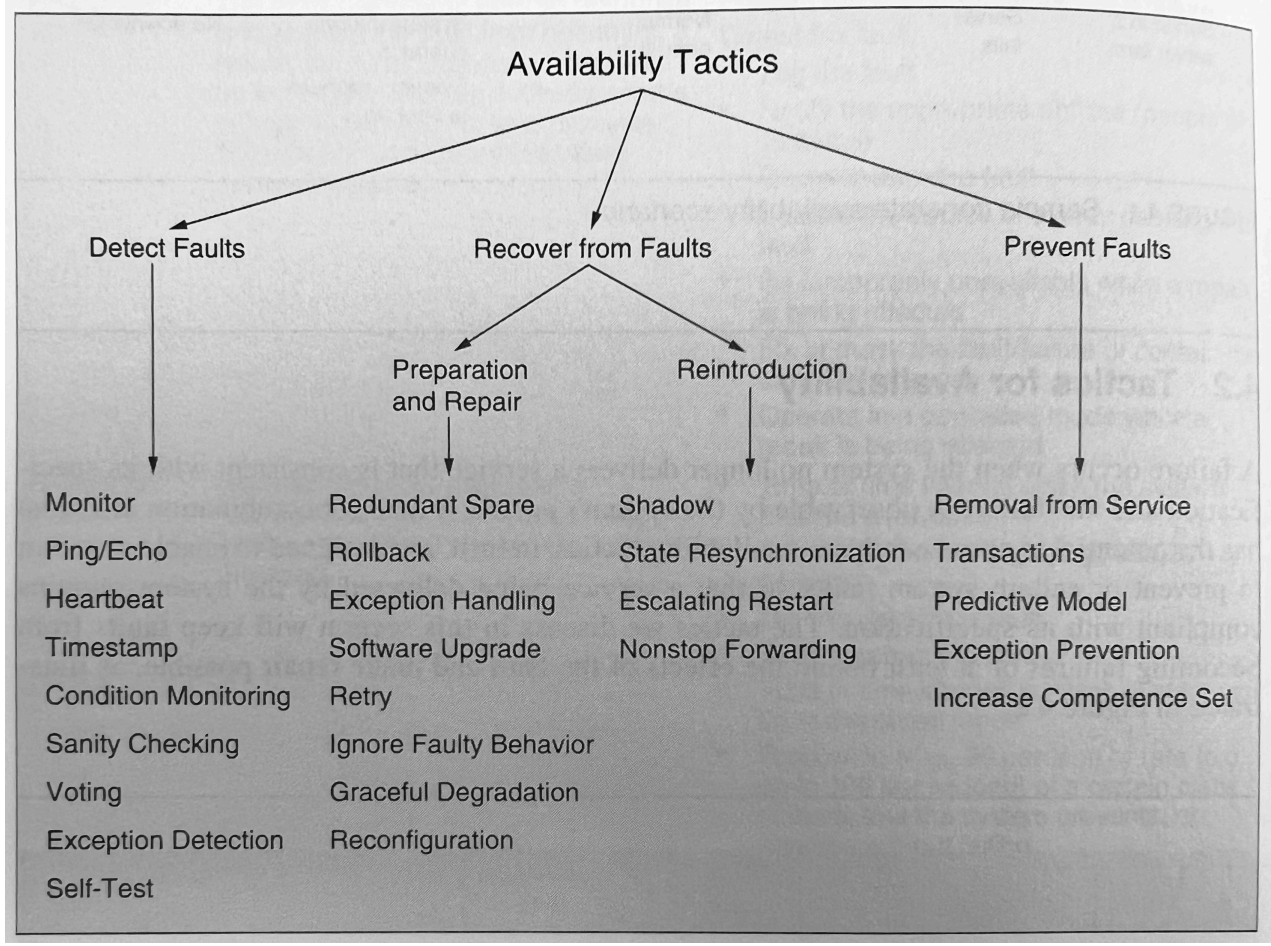
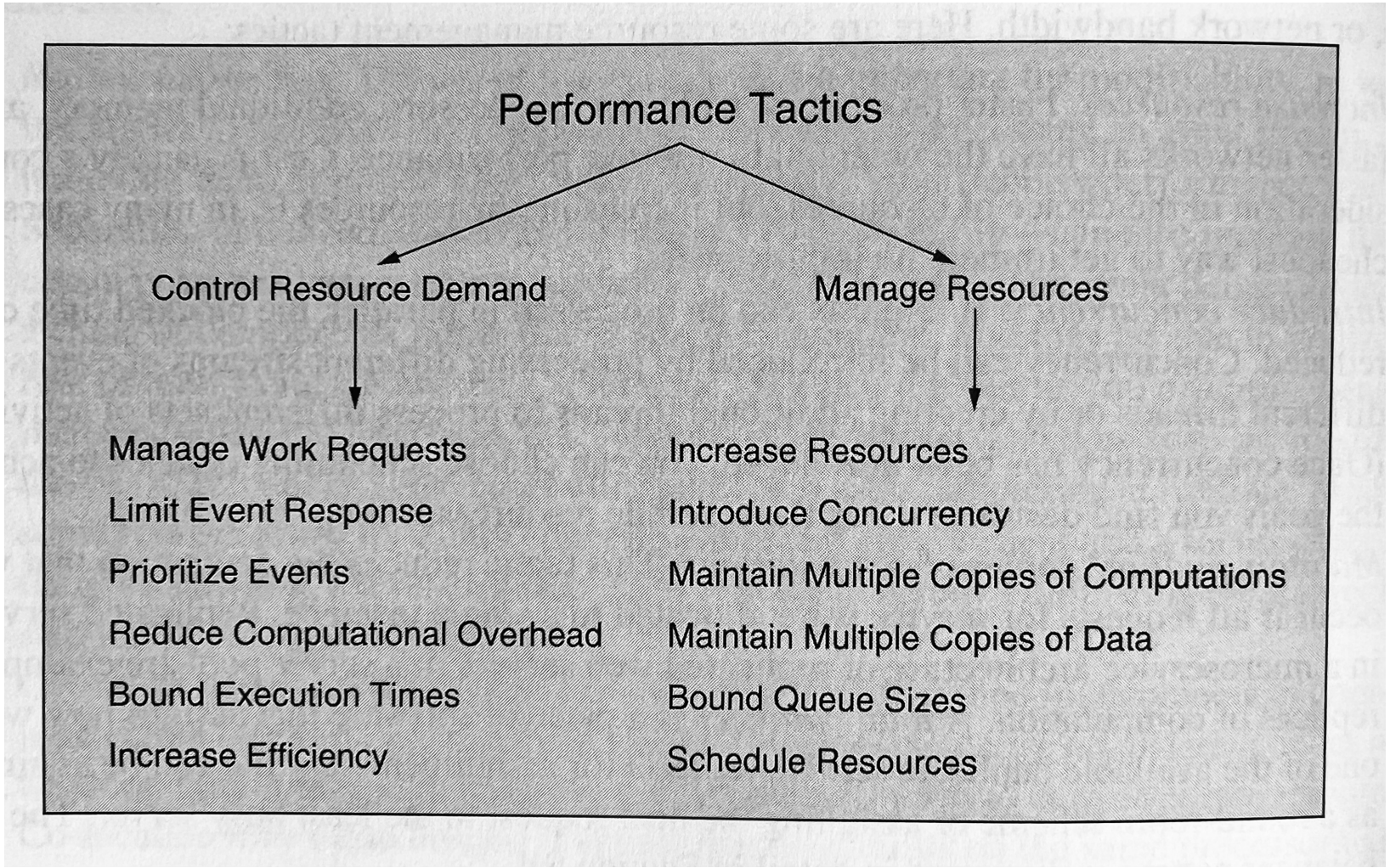


Figure 6-2. Screenshot of the job page

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

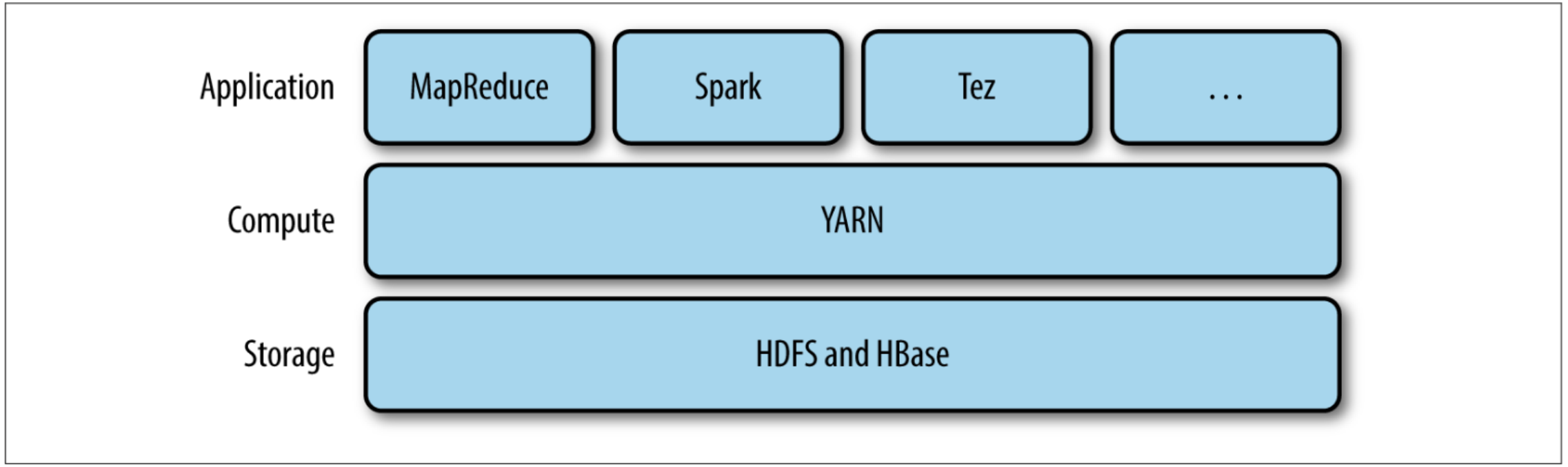


Figure 4-1. YARN applications

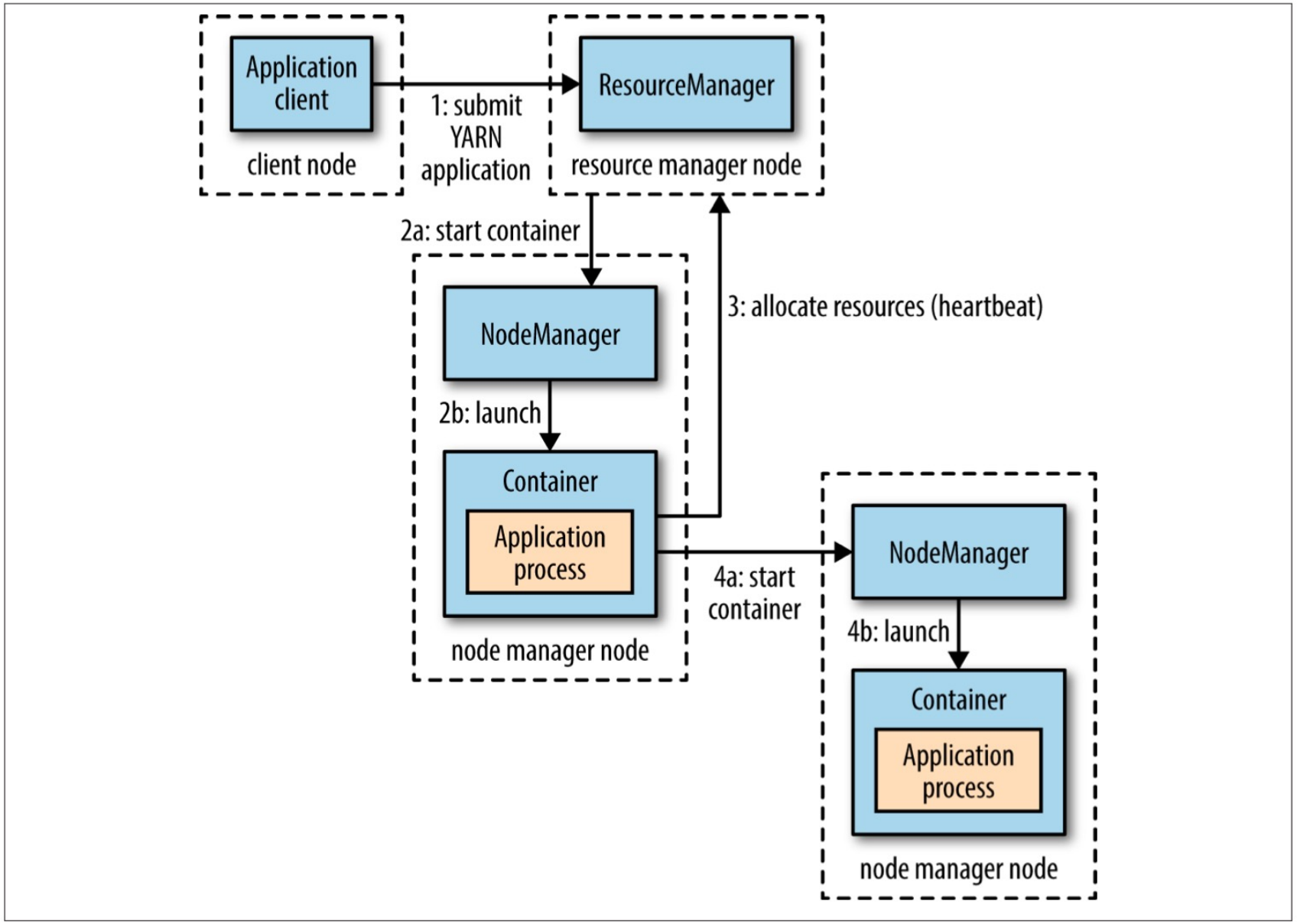


Figure 4-2. How YARN runs an application

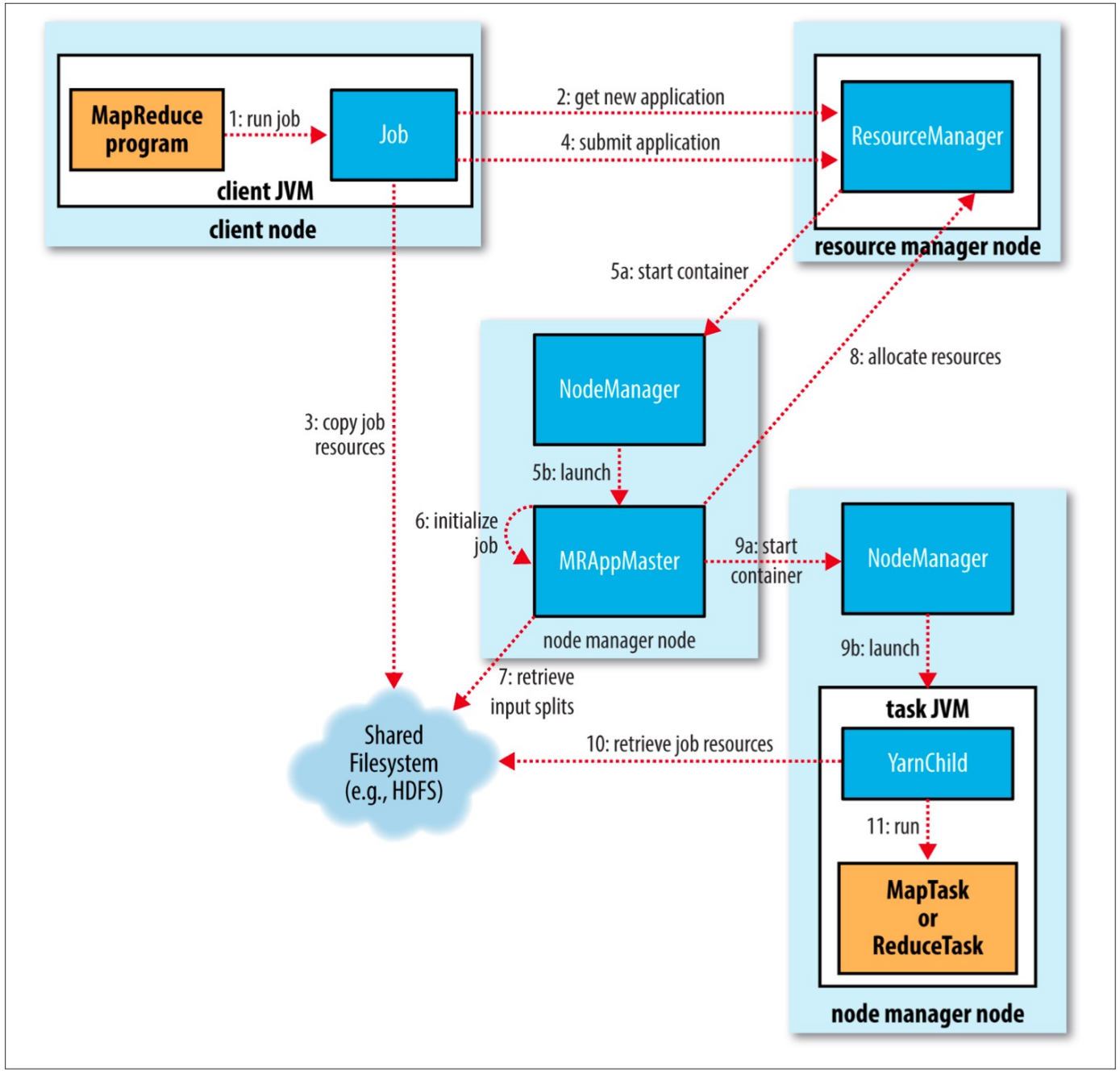


Figure 7-1. How Hadoop runs a MapReduce job

Apache Hive

What is Hive?

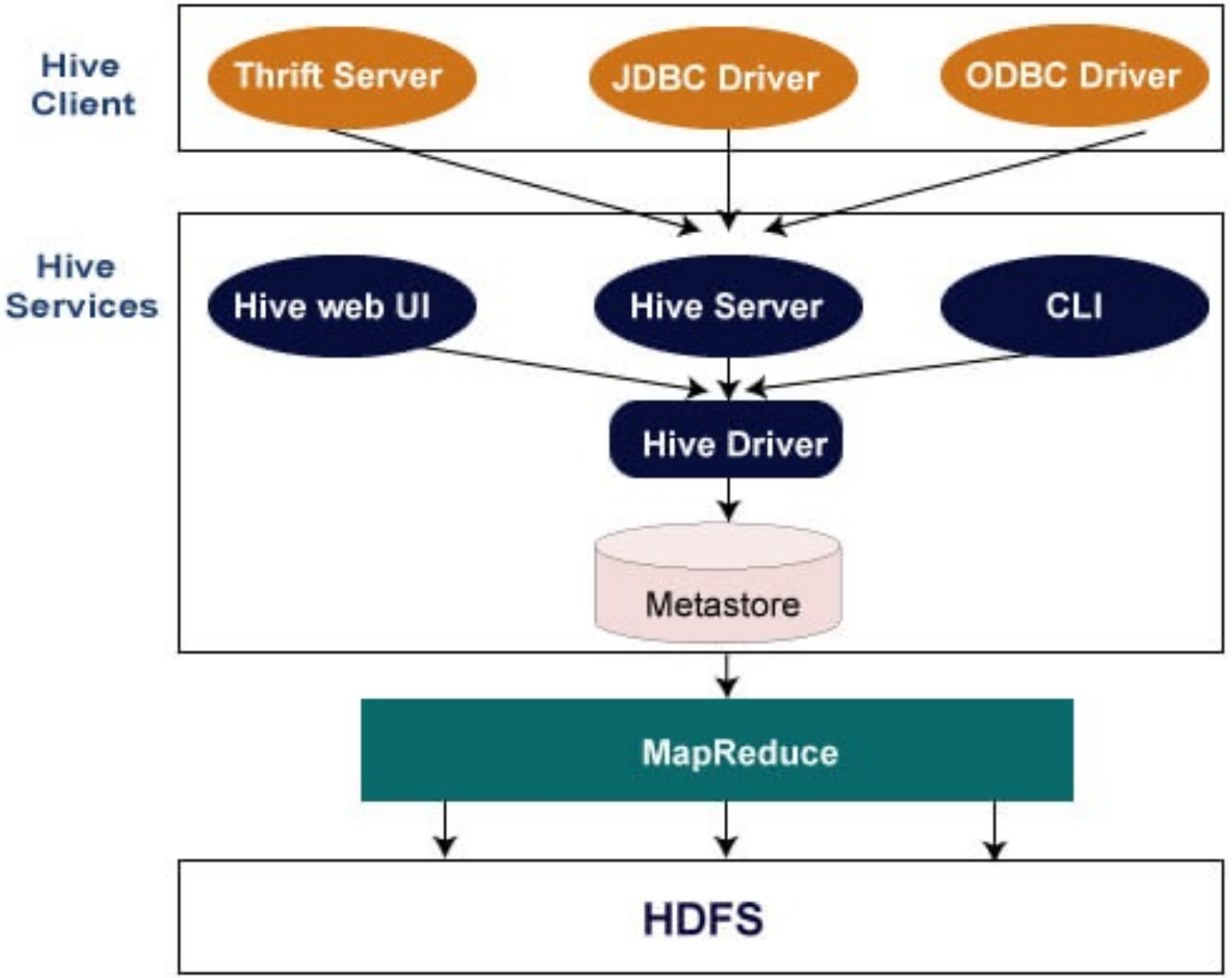
→A system for managing and querying structed data built on top of Hadoop (HDFS)
→Uses MapReduce for execution
→Allows for a typical SQL interface to a distributed file structure
→Even works with CSV files
→Performs well

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


Hive Client
Thrift Server
JDBC Driver
ODBC Driver
Hive Services
Hive web UI
Hive Server
CLI
Hive Driver
Metastore
MapReduce
HDFS

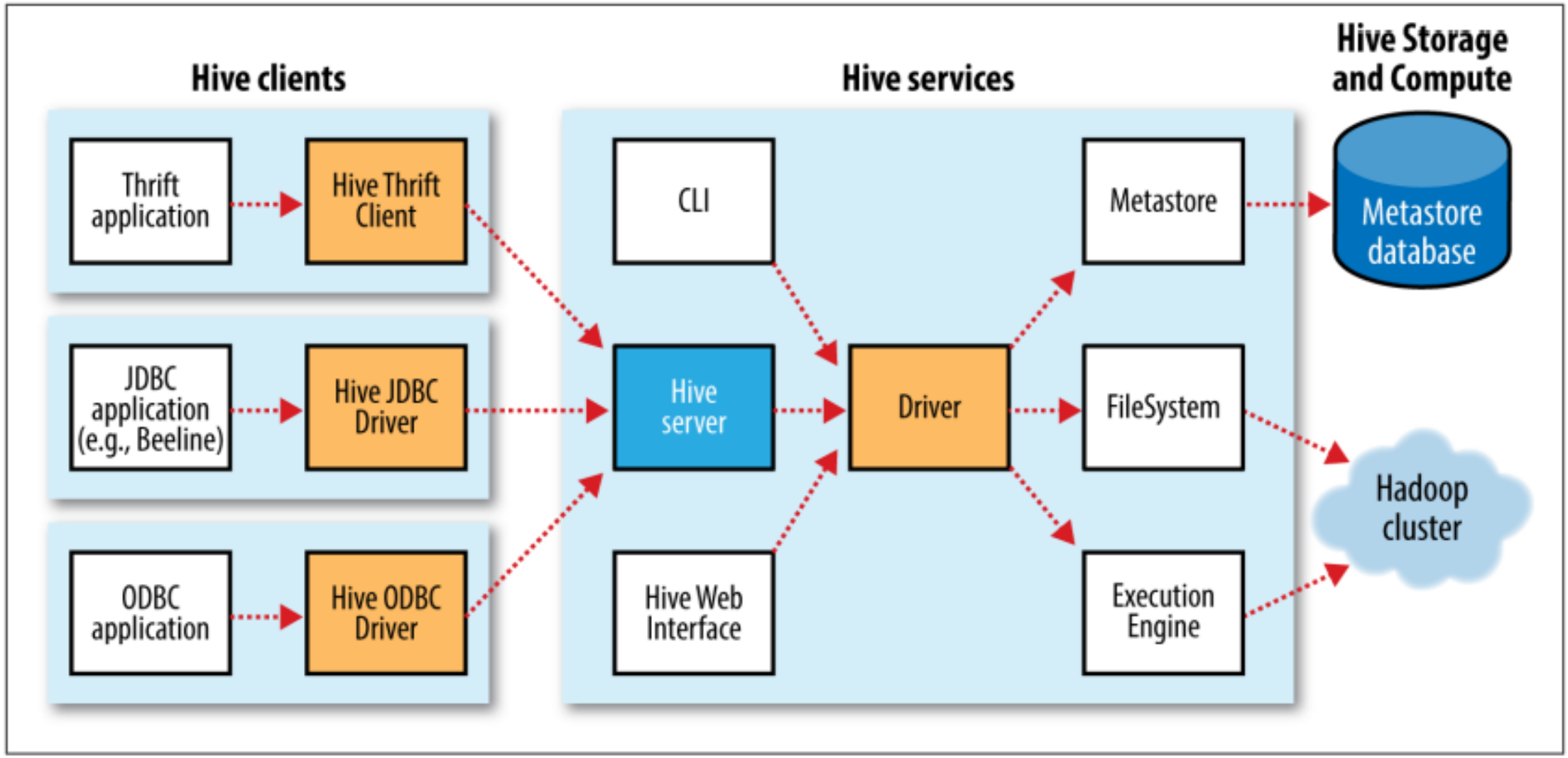
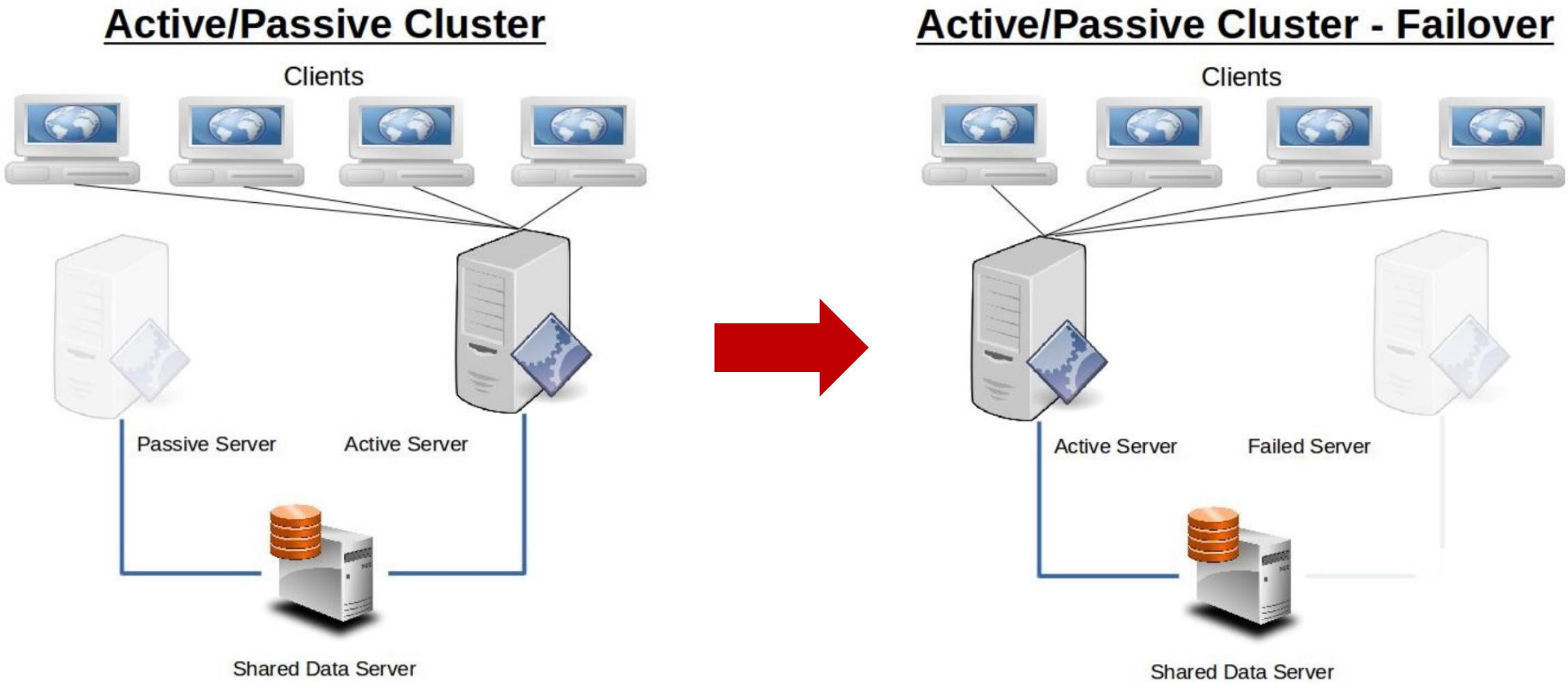


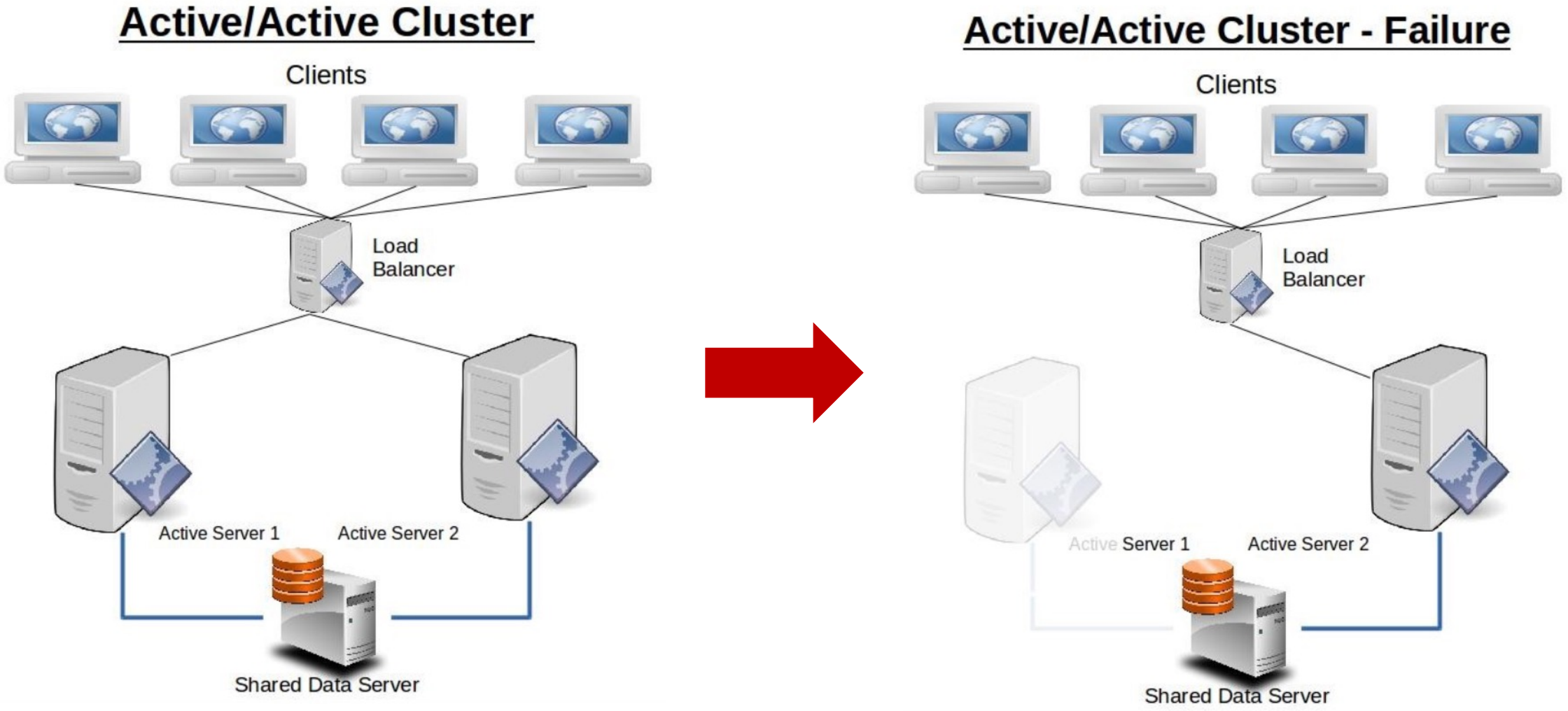
Figure 17-1. Hive architecture

High Availability and Partitioning Strategies

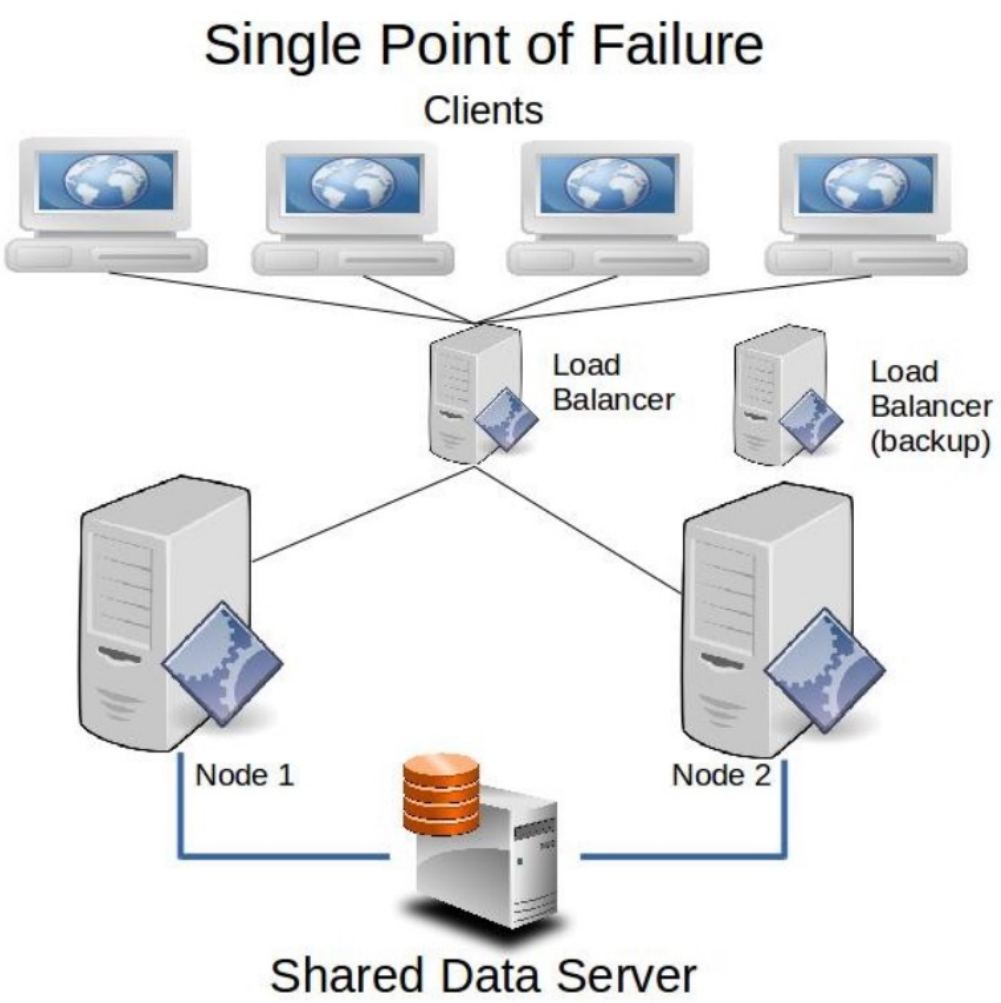
High Availability 1/3

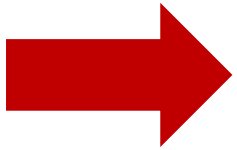


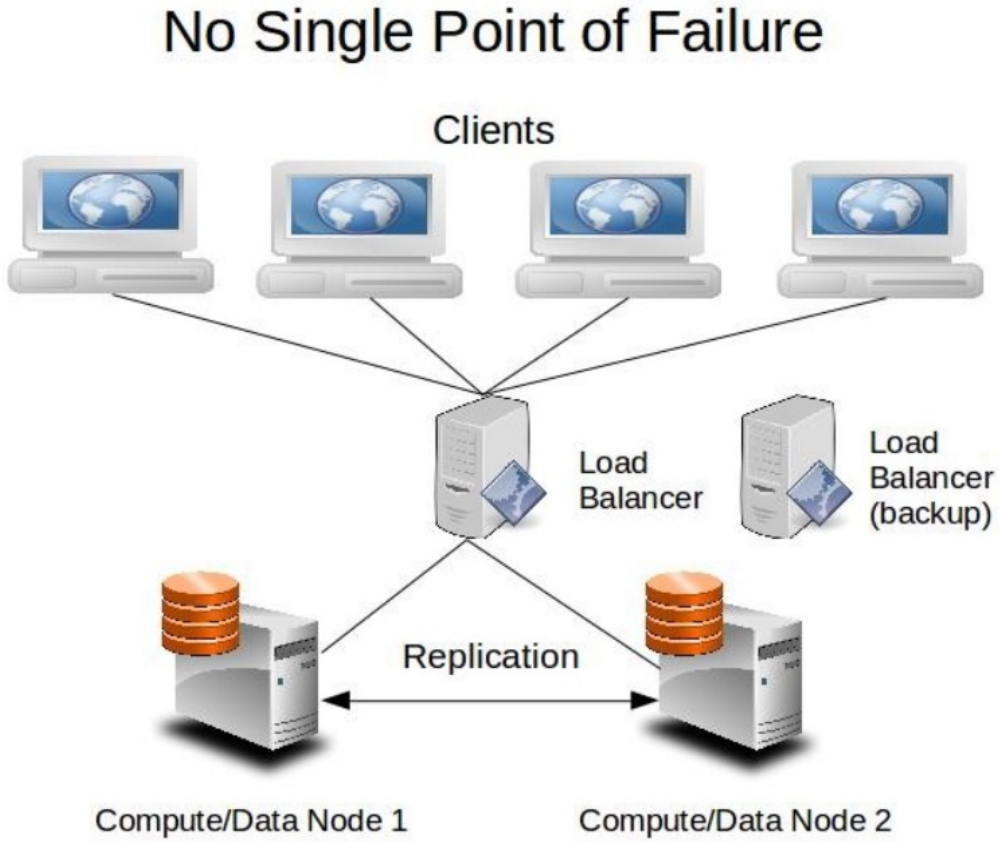
High Availability 2/3

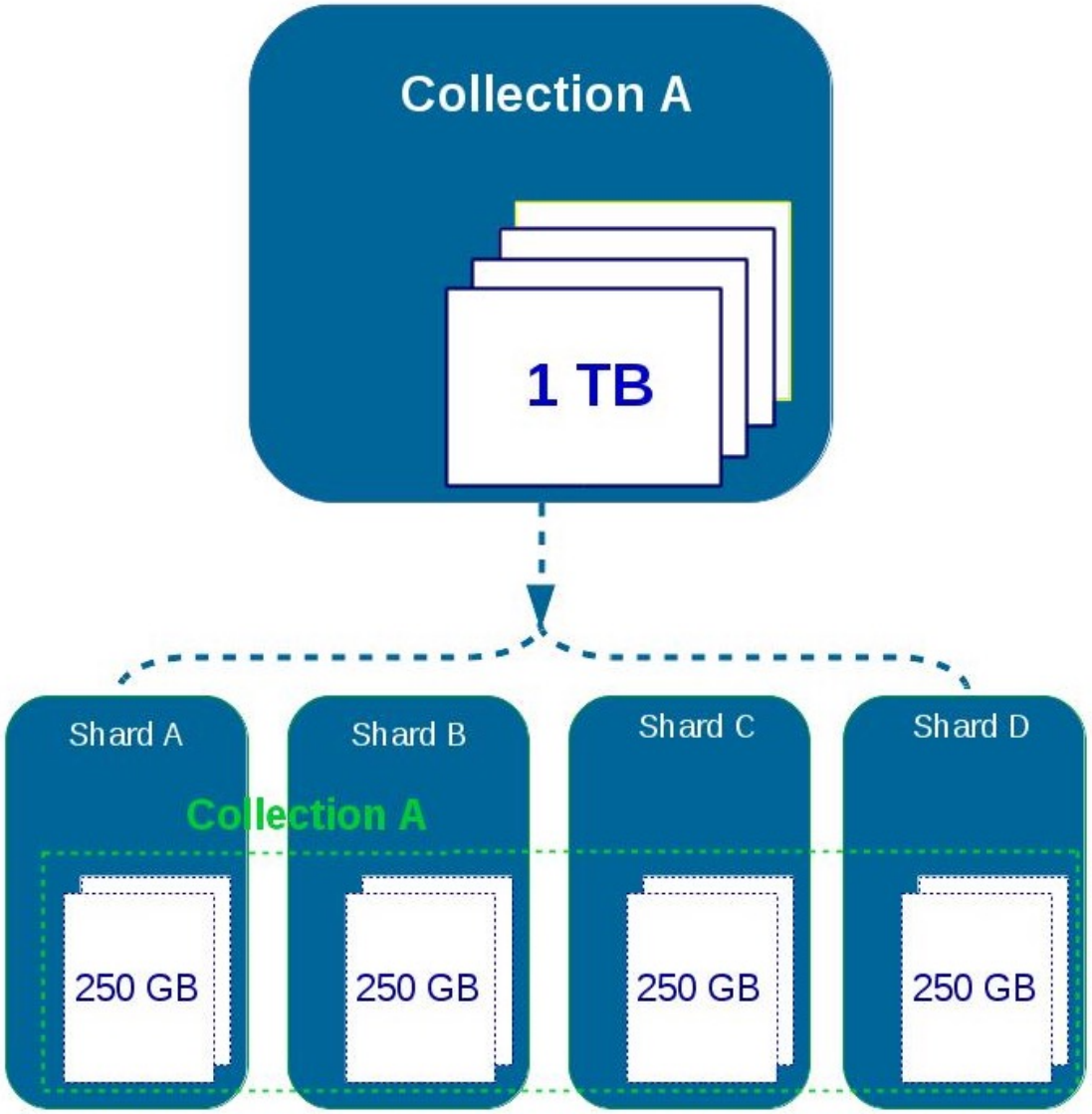


High Availability 3/3









Partitioning (Sharding)

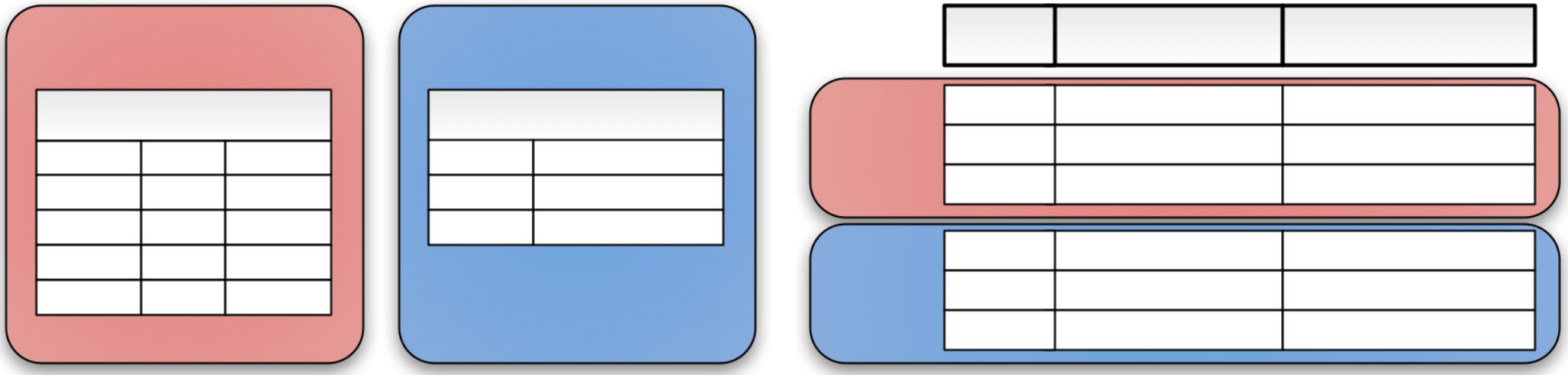
→ Splitting up the data between multiple servers

→ Together they represent the entire collection

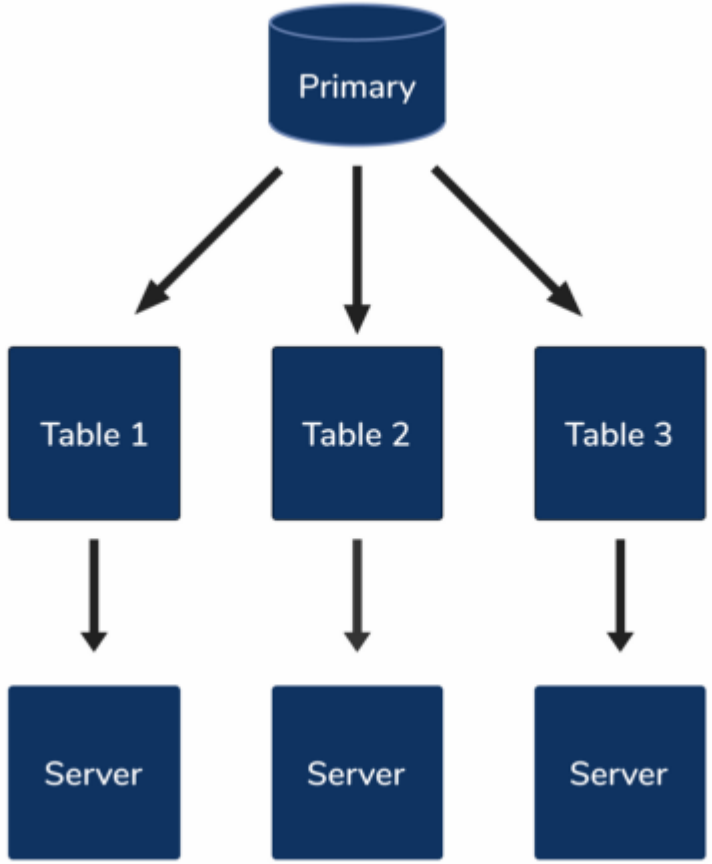
→ Allows for better throughput as calculations and search are shared

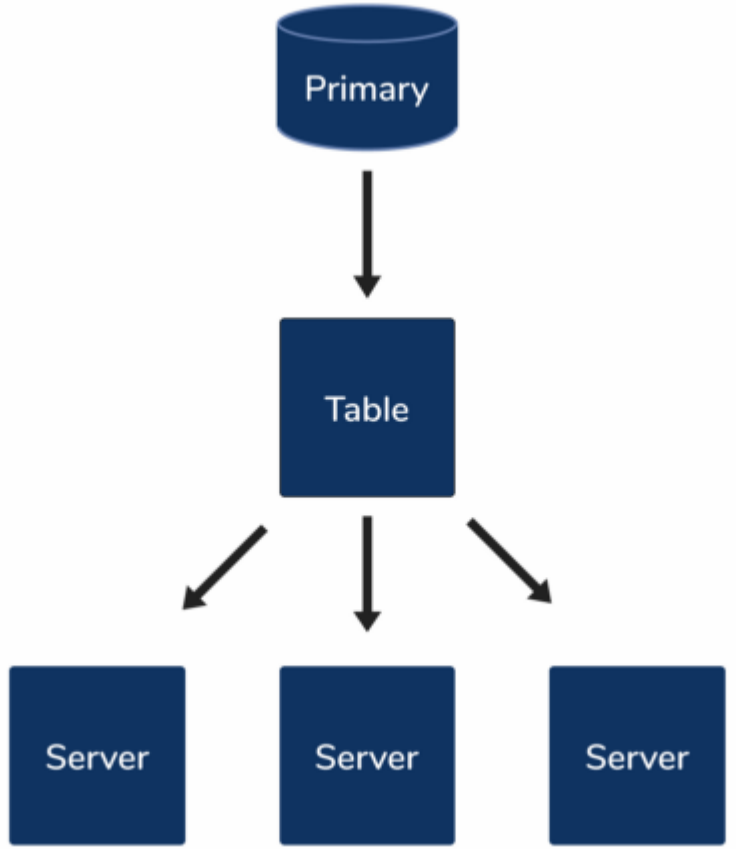
→ Makes sense for Big Data Applications

Partitioning (Sharding): Vertical vs Horizontal



Partitioning (Sharding): Vertical vs Horizontal





Scaling NoSQL Databases:

Case 1 - Hbase

SDU

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

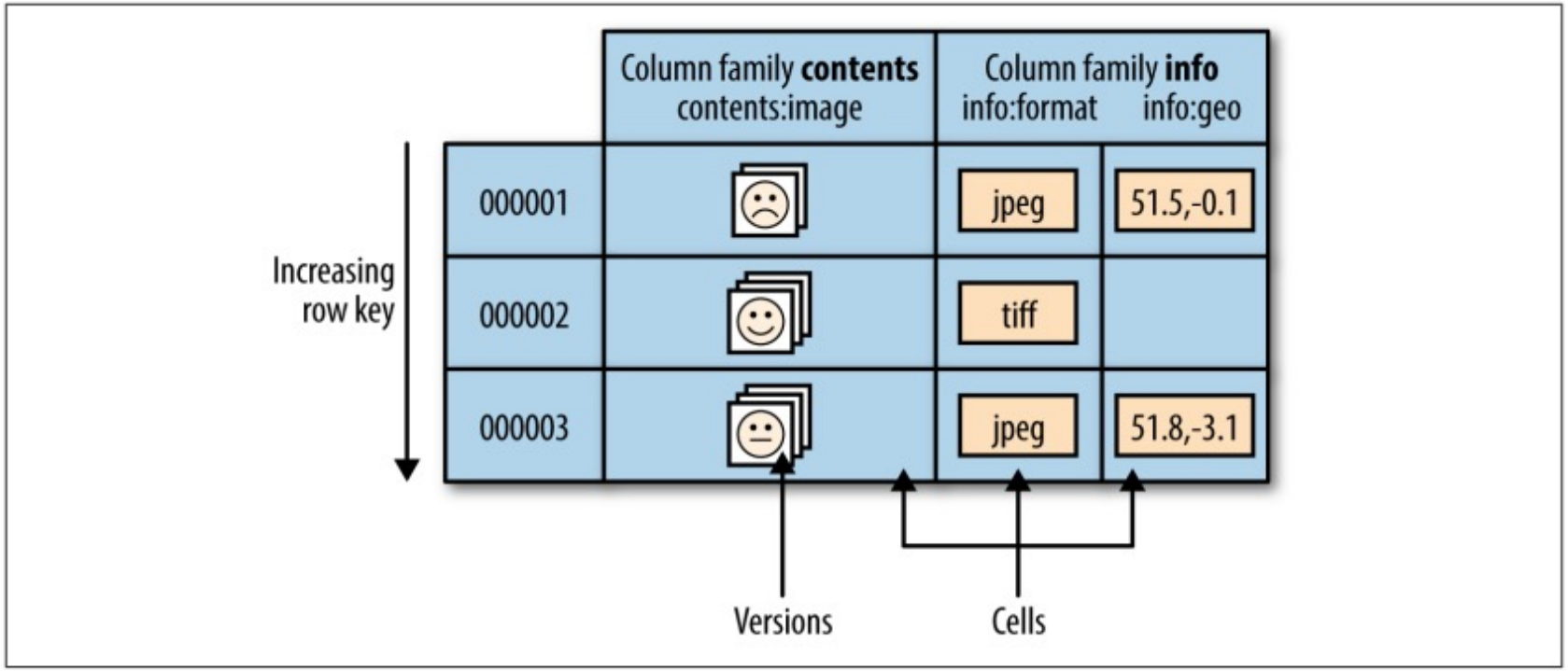


Figure 20-1. The HBase data model, illustrated for a table storing photos

HBase Cluster

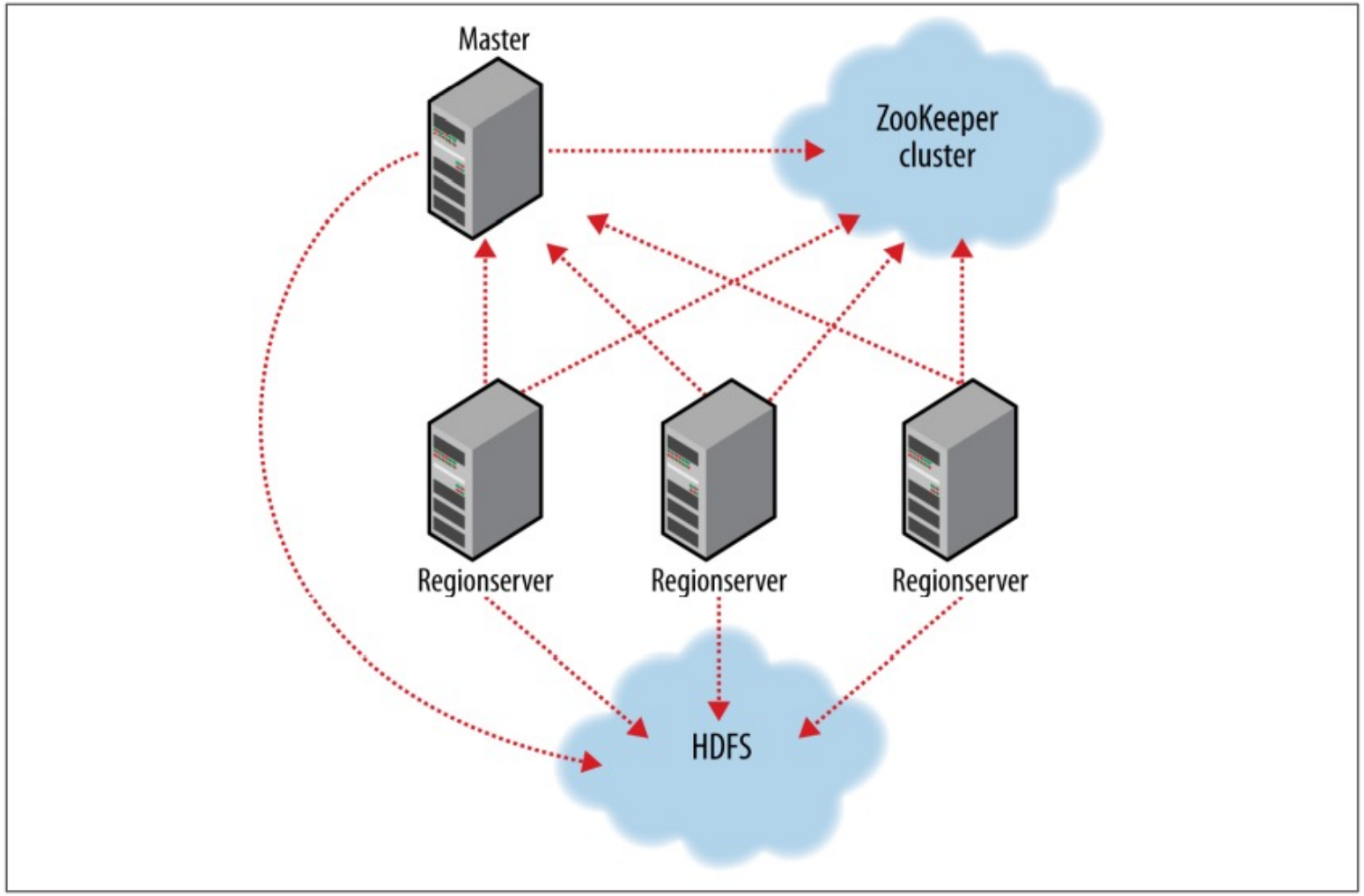
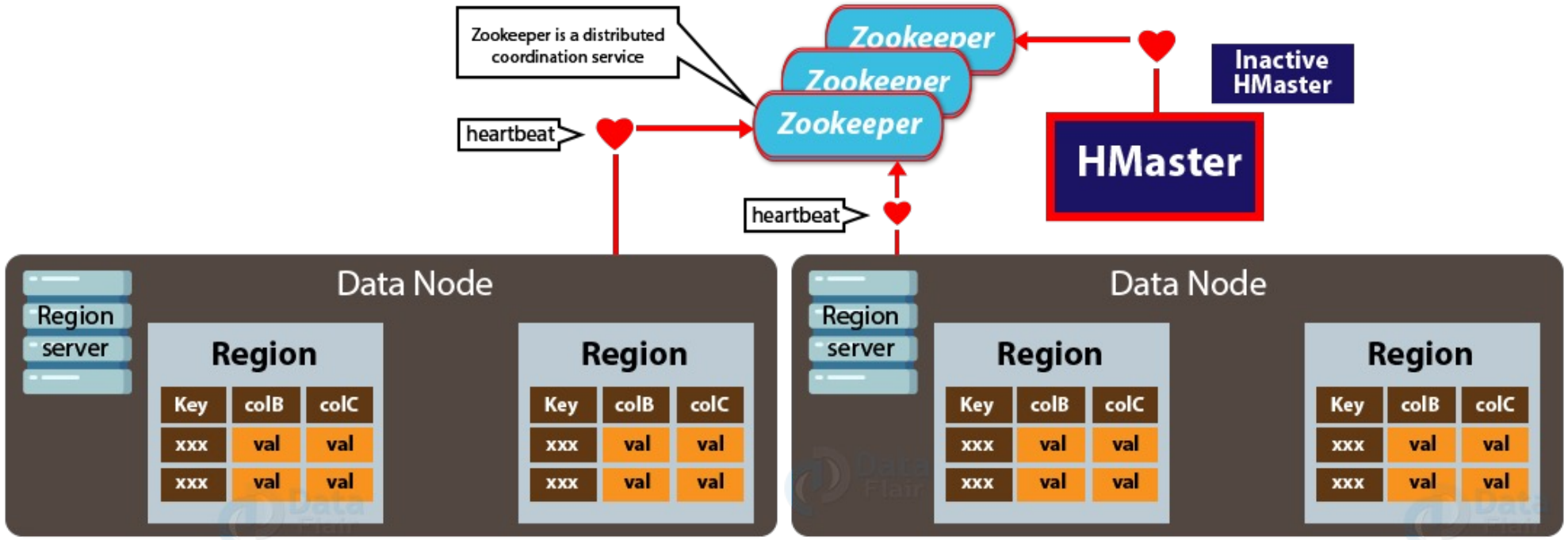


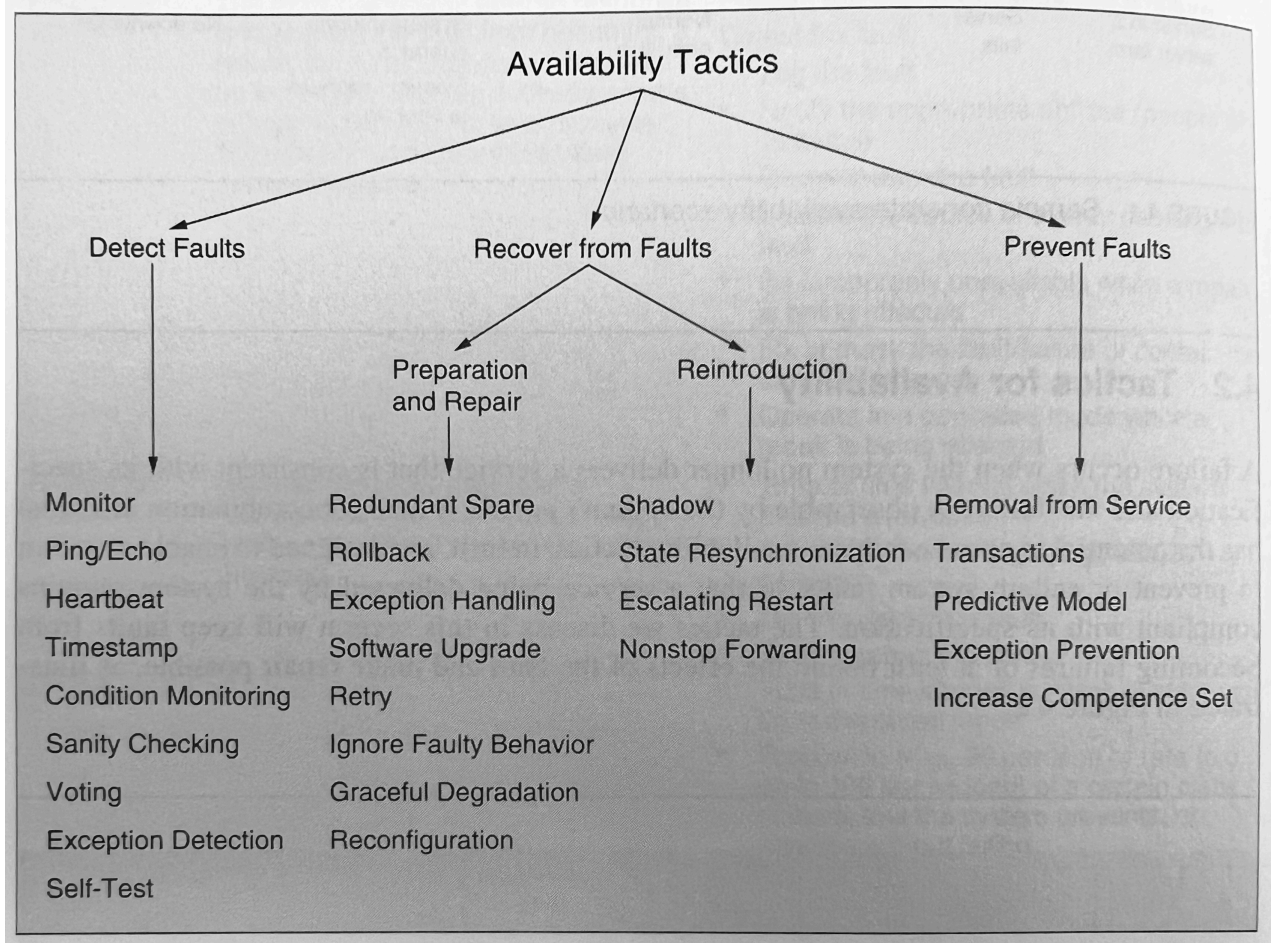
Figure 20-2. HBase cluster members

HBase Architecture

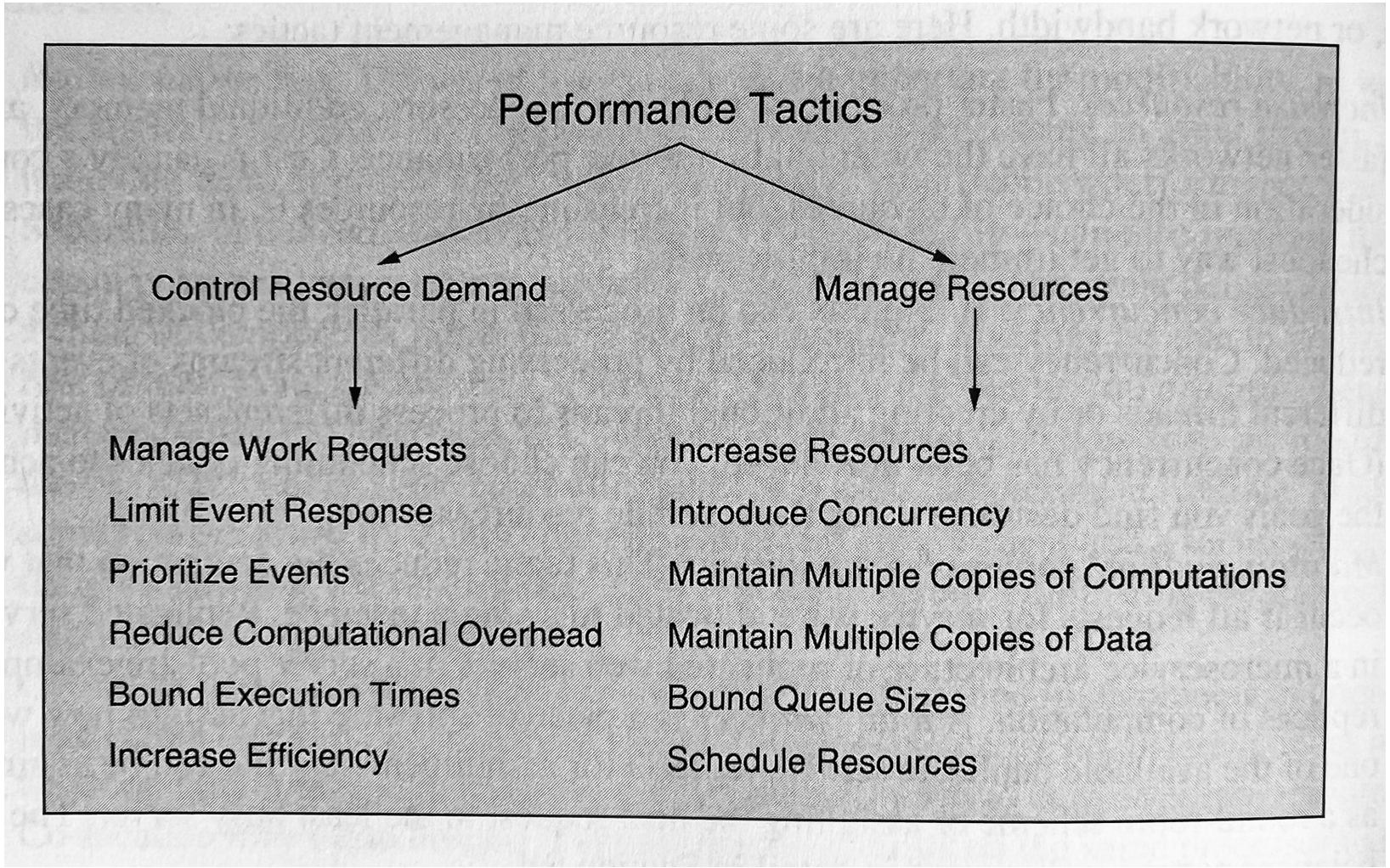


Source: https://data-flair.training/blogs/hbase-architecture/
Go here to get a great overview of the Hbase architecture!!!

Availability Tactics



Performance



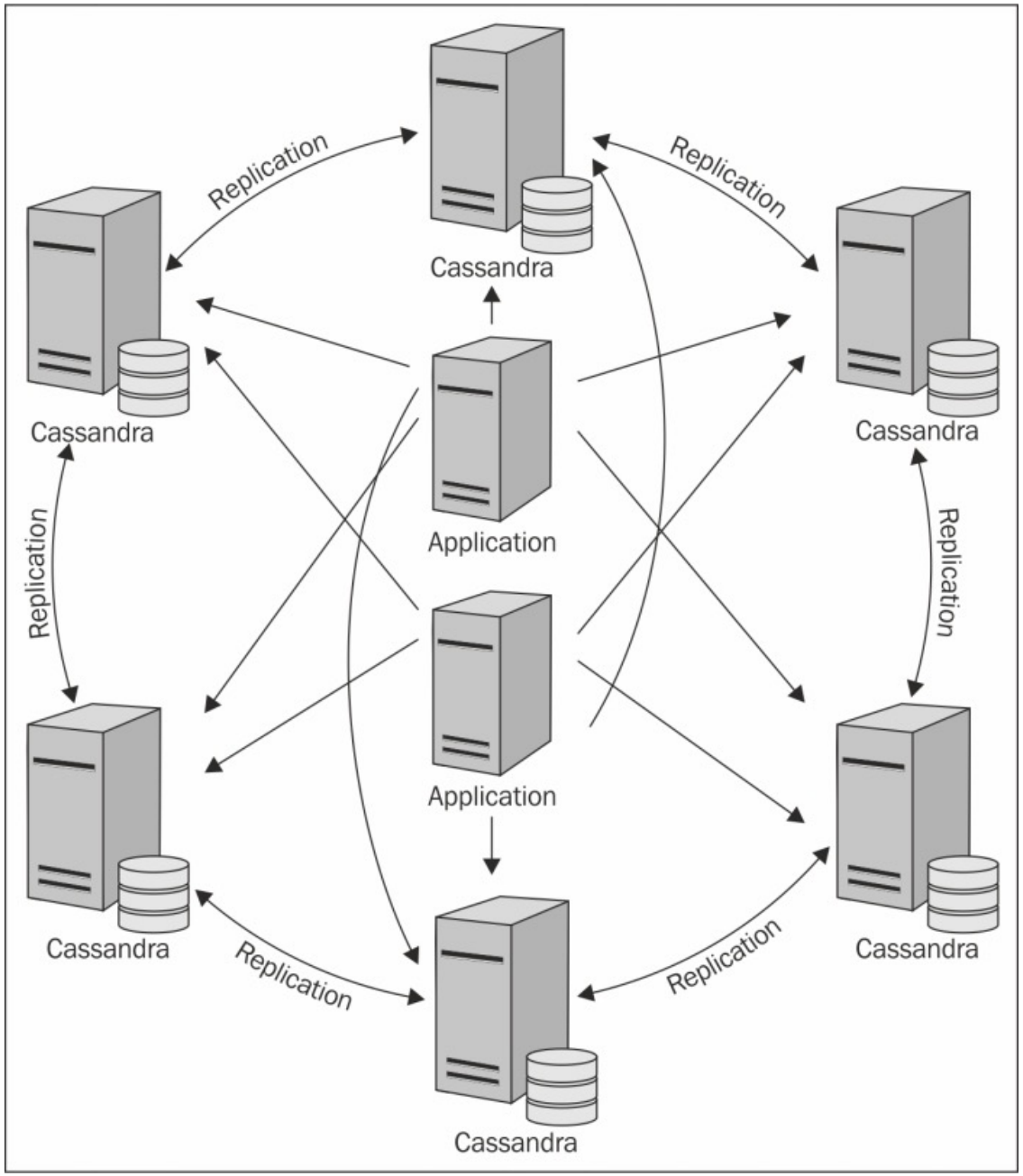
Scaling NoSQL Databases:

Case 2 - Cassandra

SDU

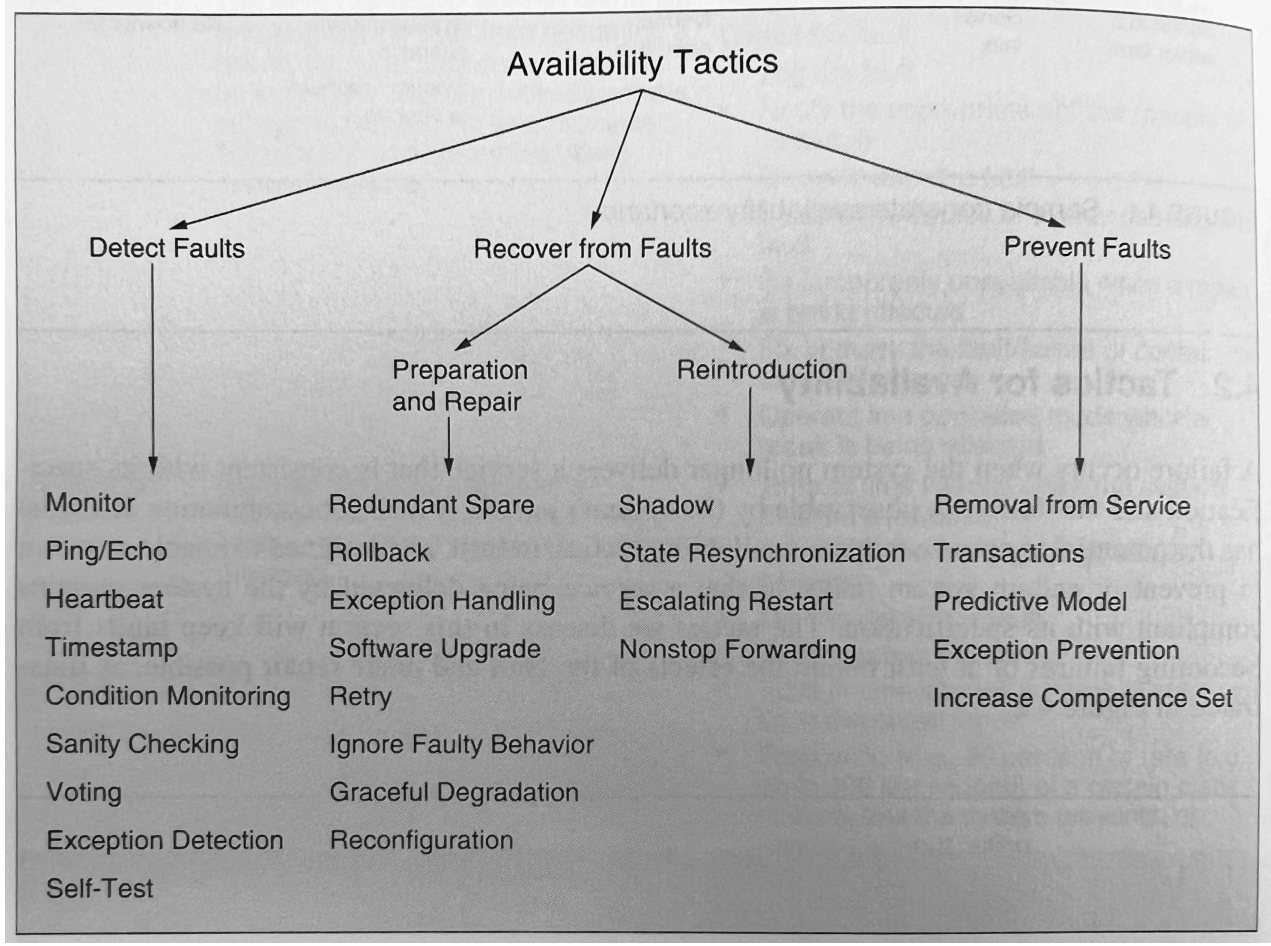
What is Cassandra?

- → Also based on Google Big Table
- → High Performance Colum 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

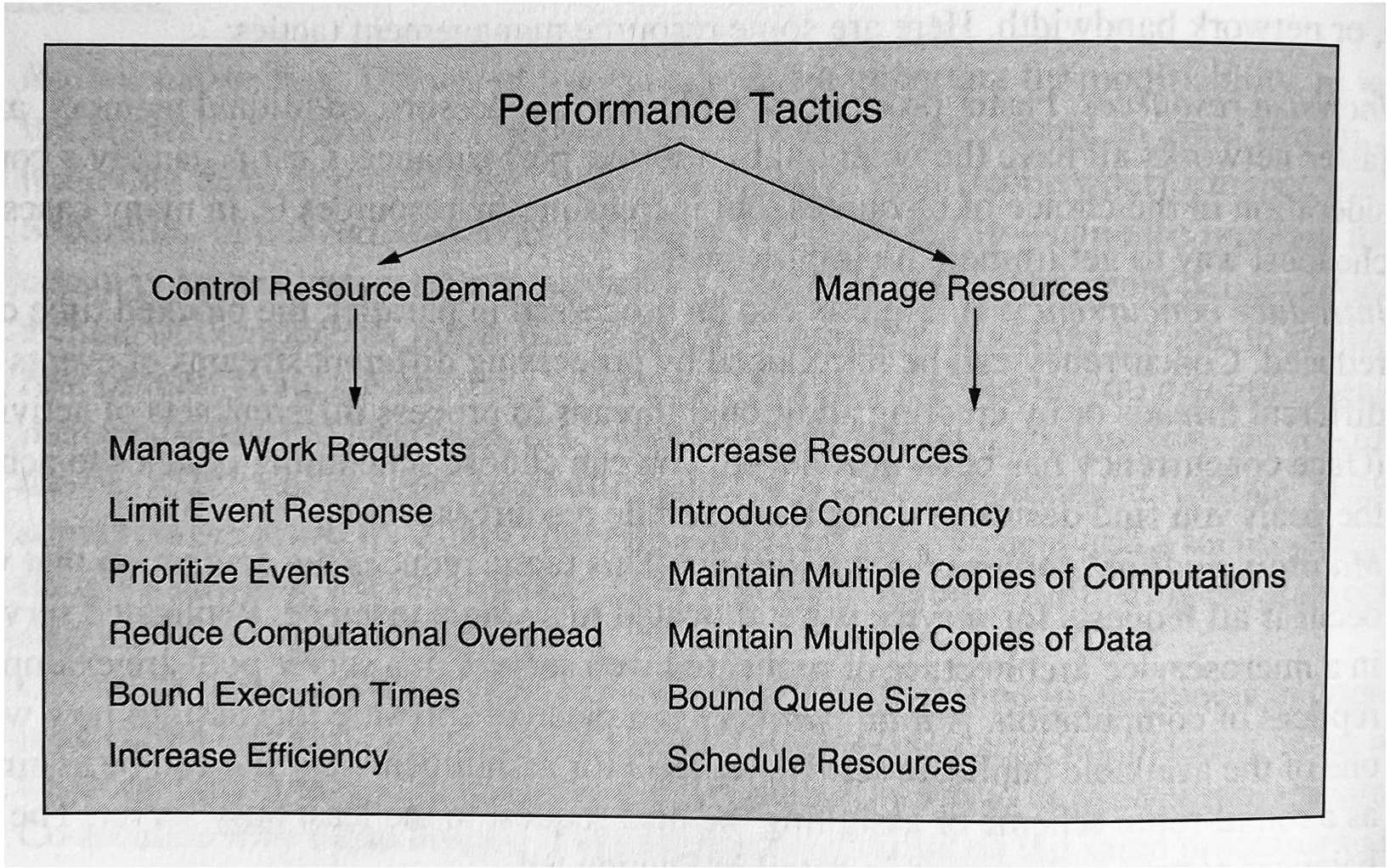


SDU

Availability Tactics



Performance



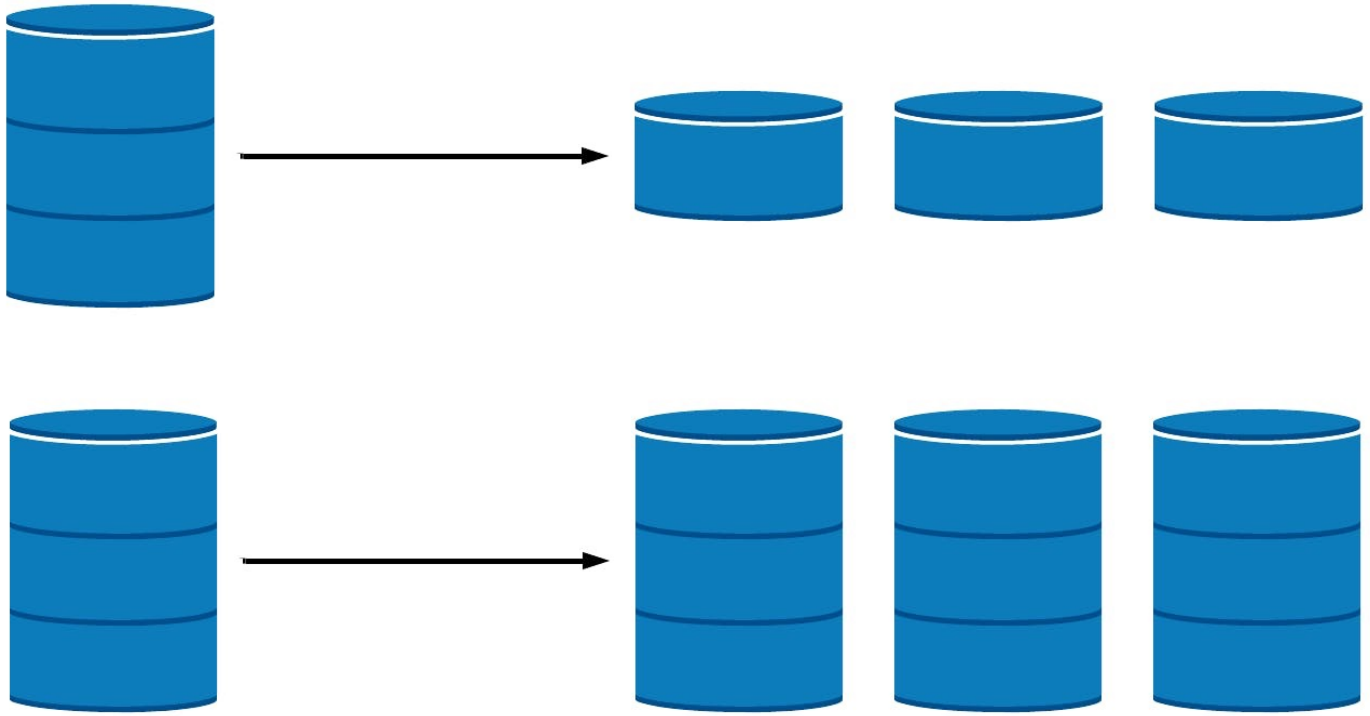
Scaling NoSQL Databases:

Case 3 - MongoDB

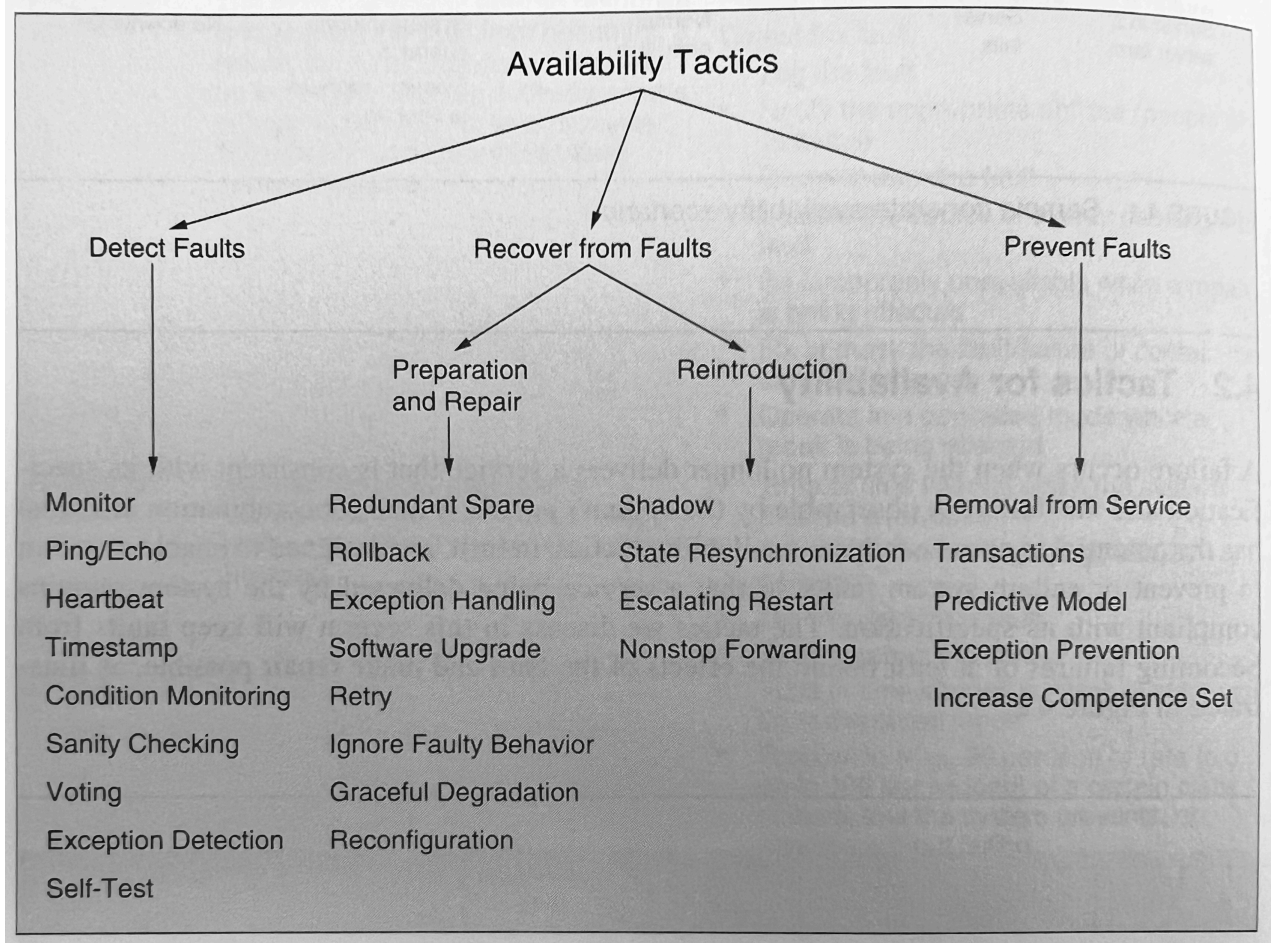
SDU

What is MongoDB?

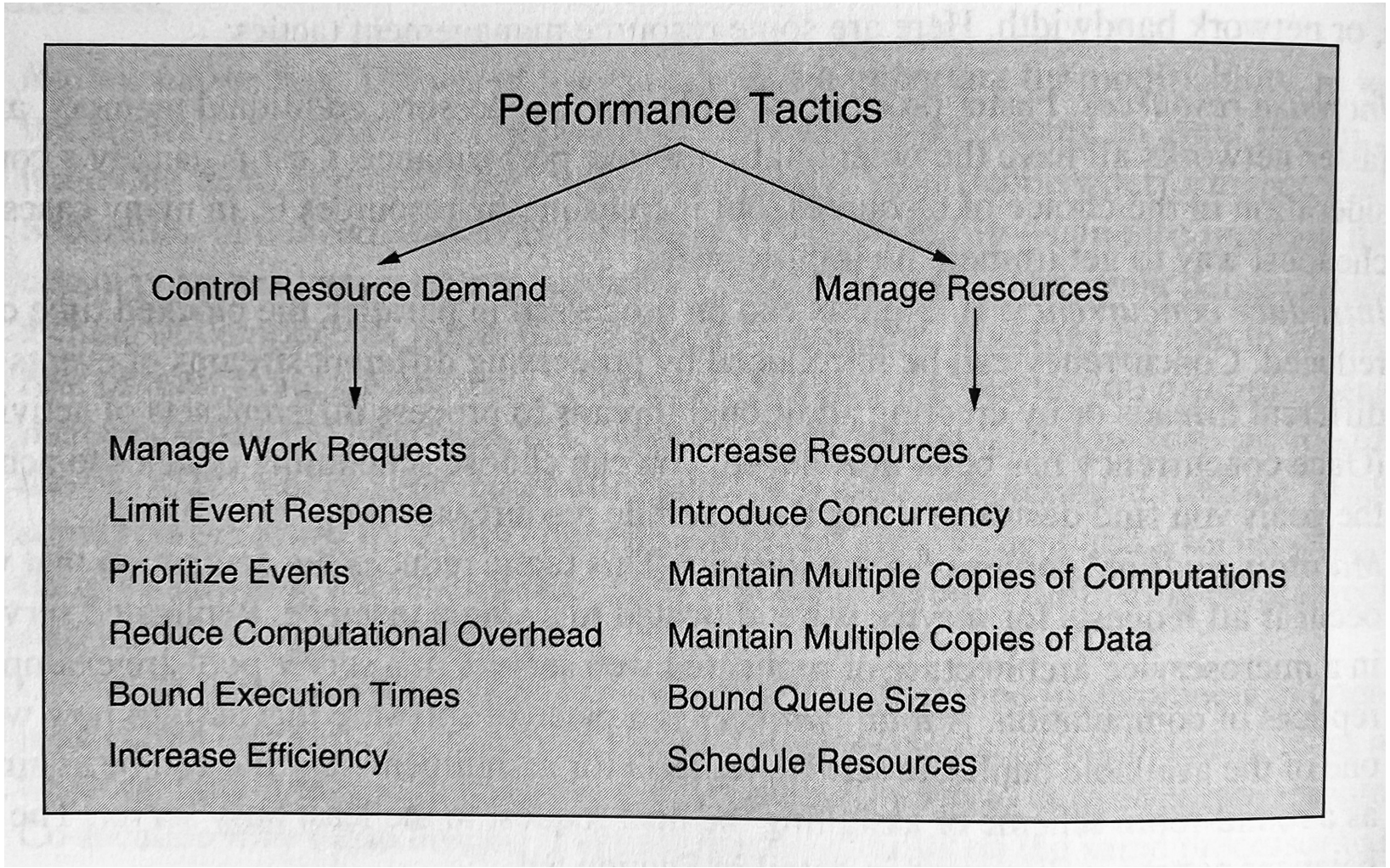
- → Strategies
  - → Sharding
  - → Replica Sets
    - → HA implementation
    - → Great for total read redundancy
    - → Potential issues with large amounts of writes due to propagation
- → Document based NoSQL Database
- → Data format: JSON



Availability Tactics



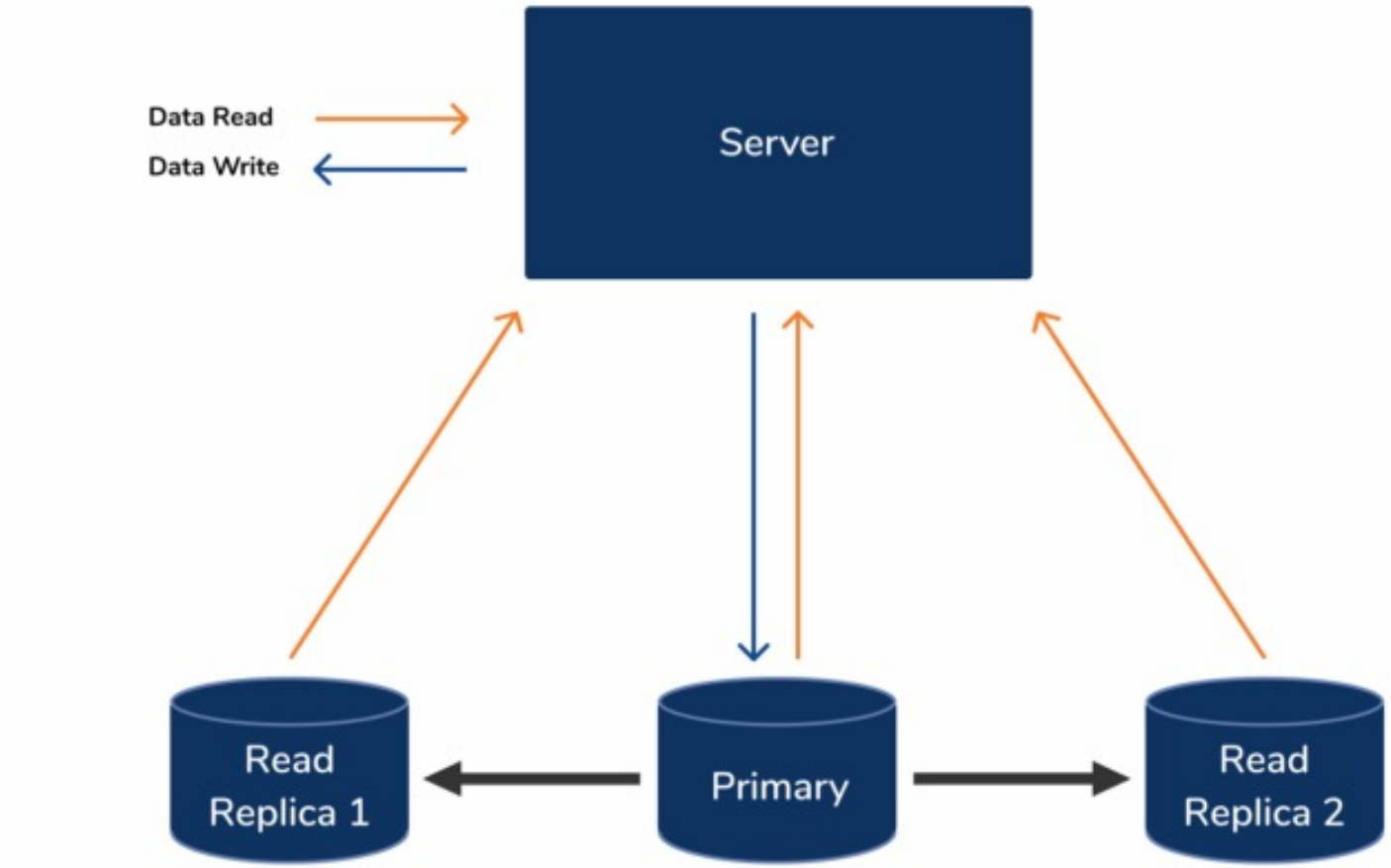
Performance



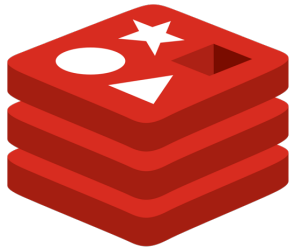
Scaling Relational Databases

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?**



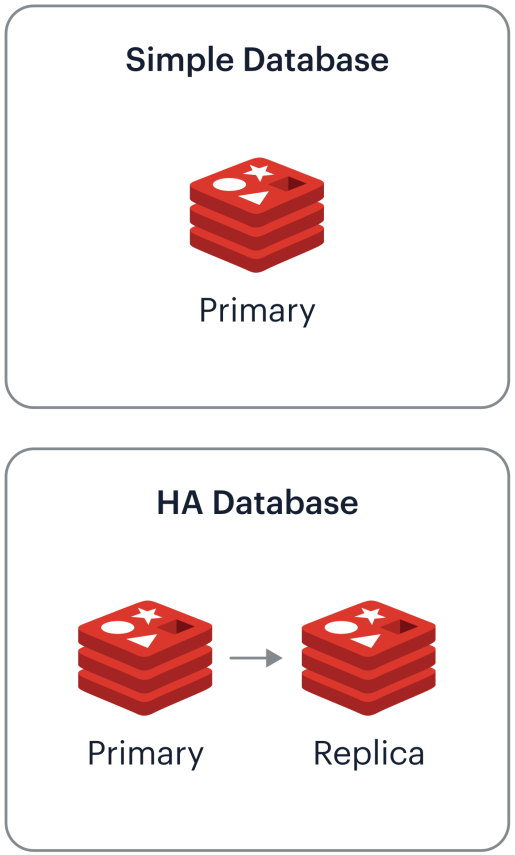
Bonus: Redis

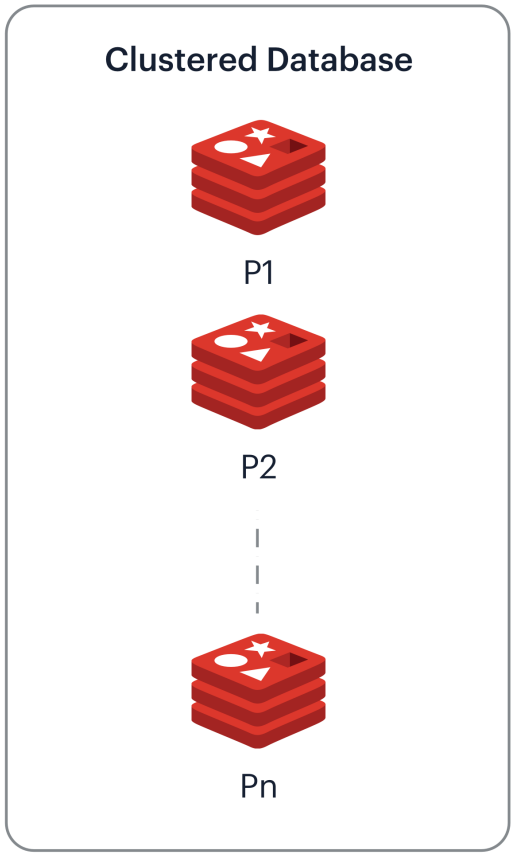


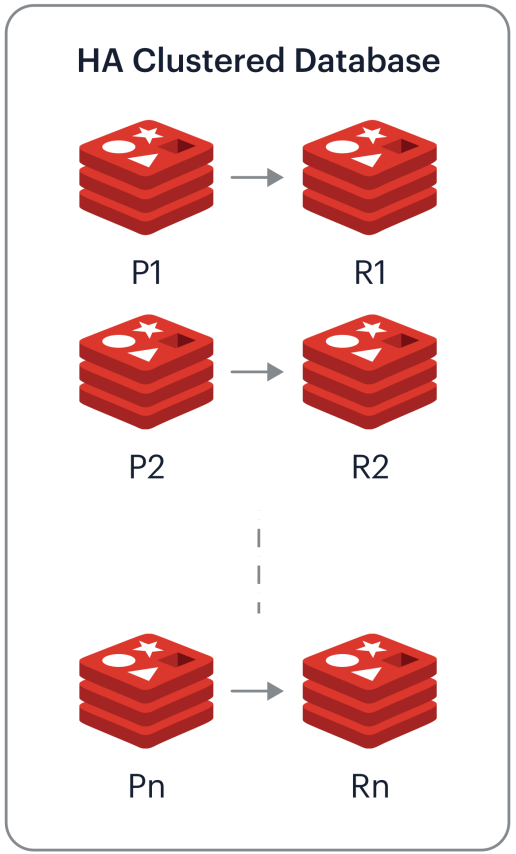
redis

- → Super fast in-memory key value-based NoSQL database
- → Transactions
- → Pub/Sub
- → Keys with a limited time-to-live
- → Automatic failover

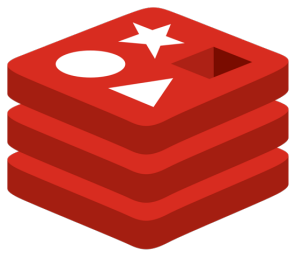
- → Often used to synchronize states between Kubernetes Pods as it is fast, resilient, and configurable



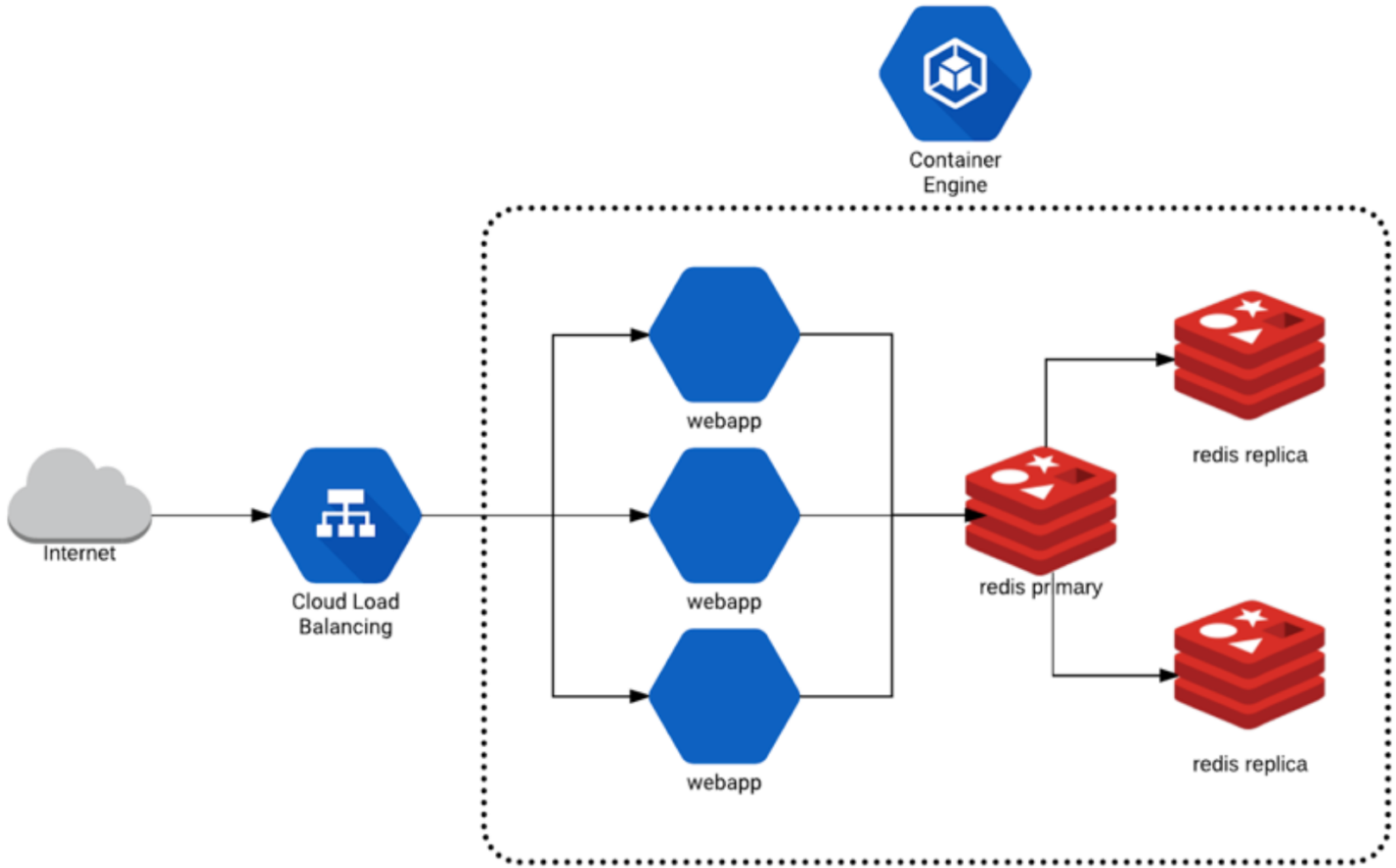




SDU



redis


Container
Engine
webapp
webapp
webapp
Internet
Cloud Load
Balancing
redis primary
redis replica
redis replica

Exercises!