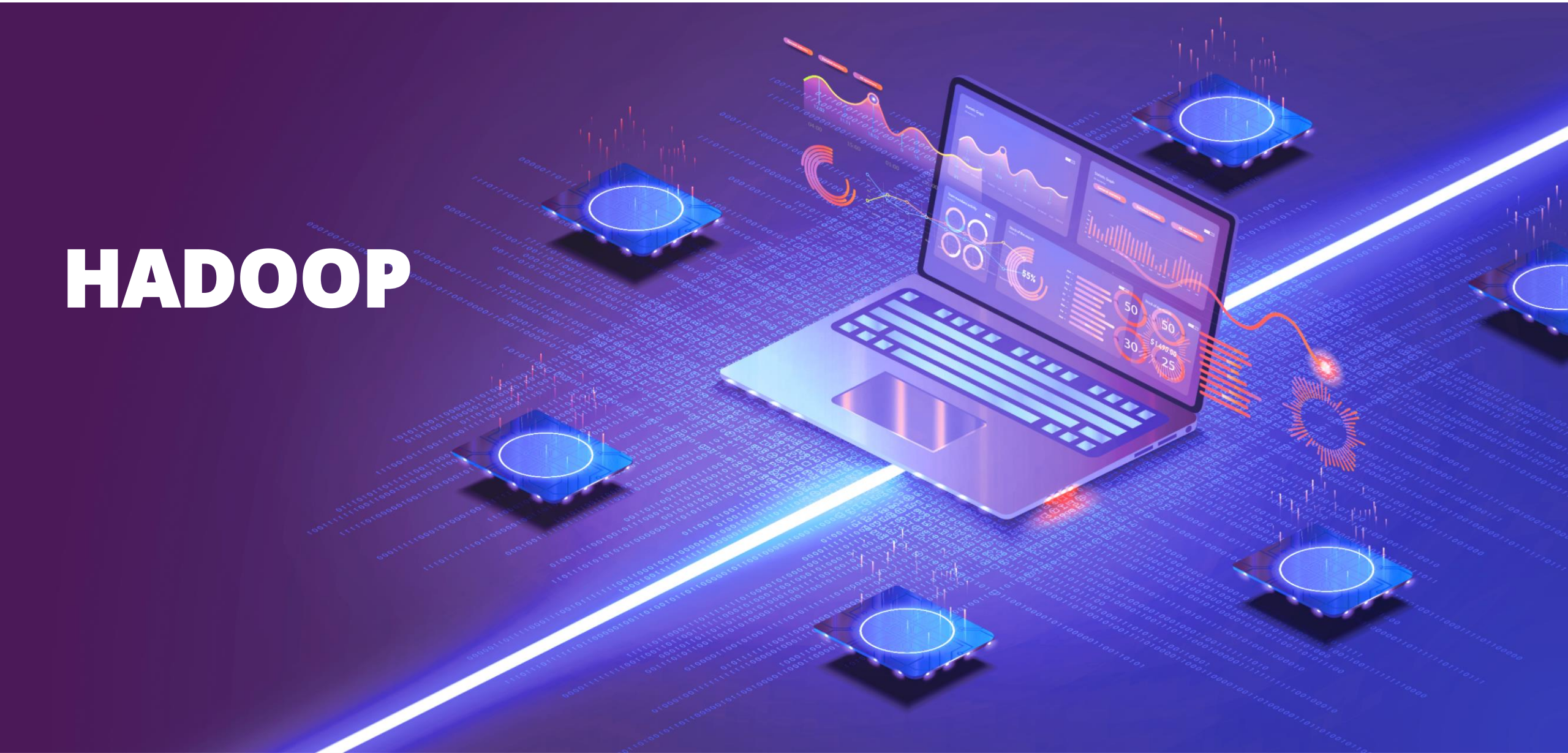
WILEY

ENABLING DISCOVERY | POWERING EDUCATION | SHAPING WORKFORCES

HADOOP



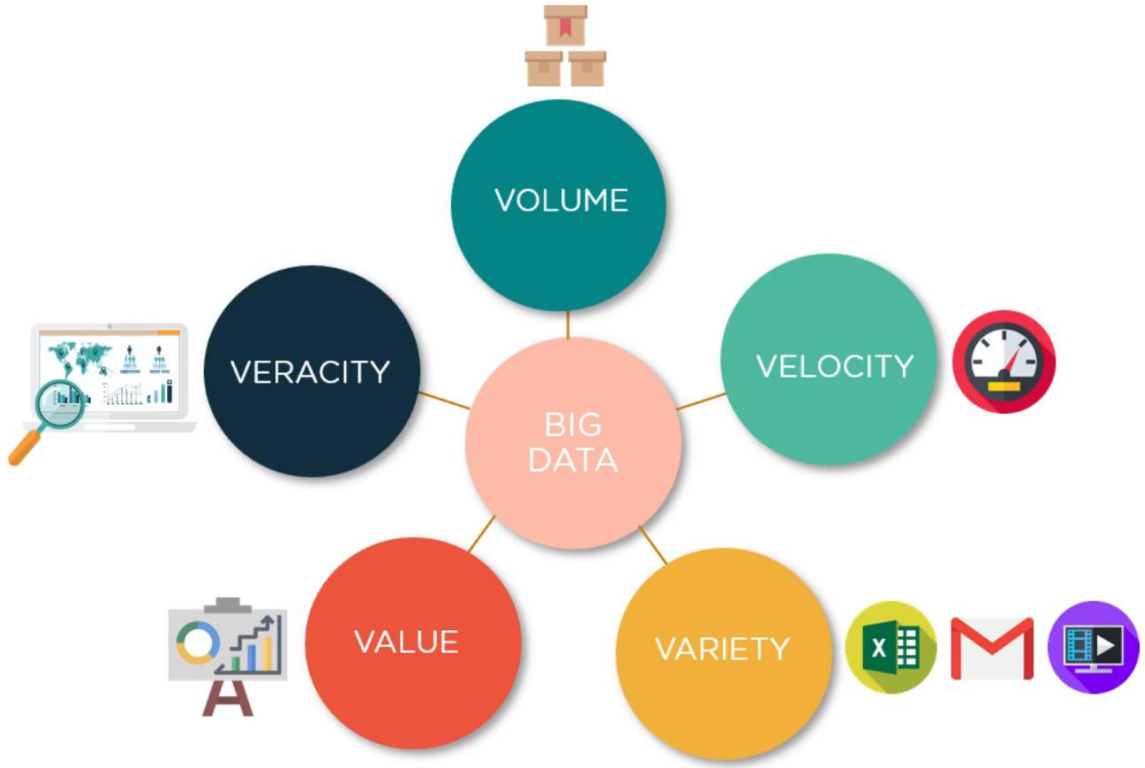

APACHE
hadoop
TM

AGENDA:

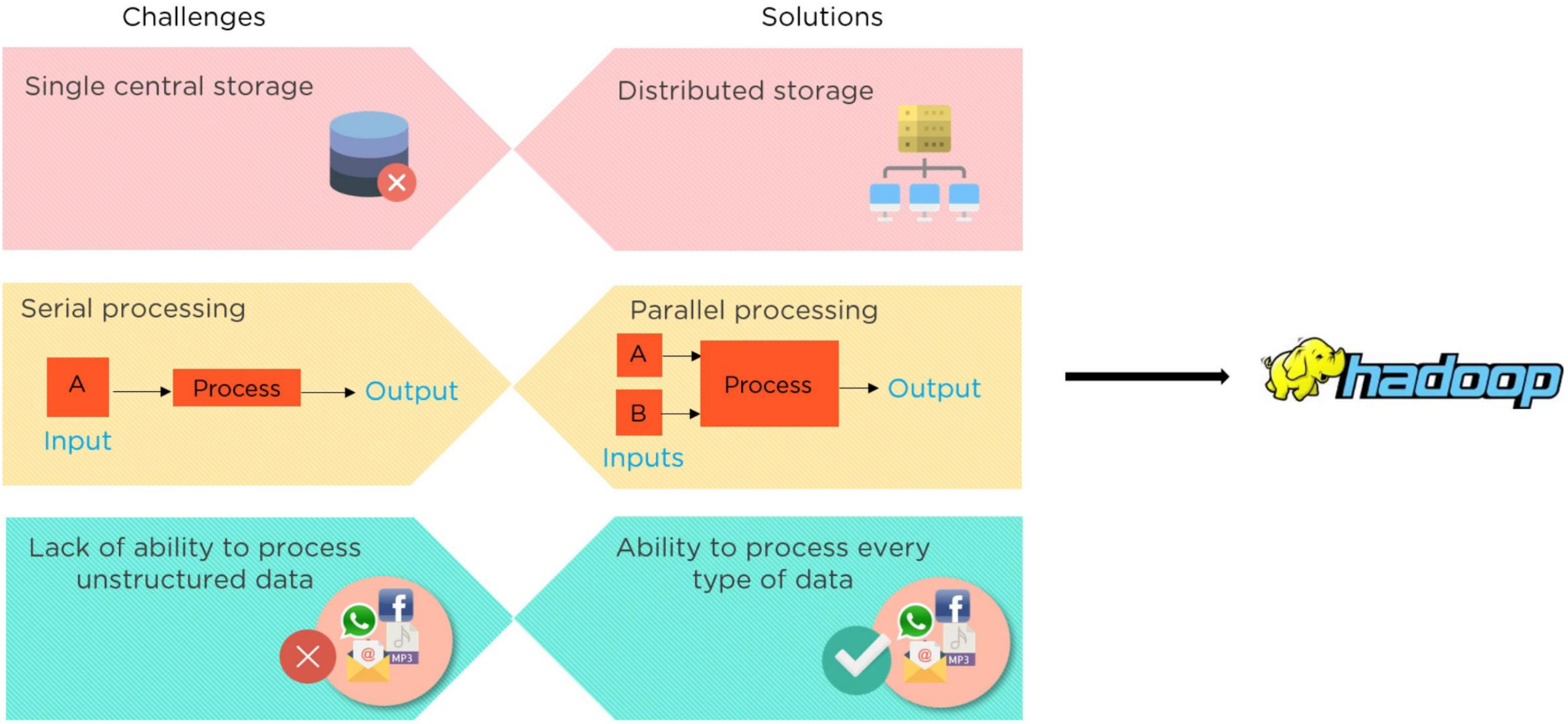
- Hadoop Architecture
- MapReduce
- Hadoop Distributed File System
- How Does Hadoop Work?
- Advantages of Hadoop
- Features of HDFS
- HDFS Architecture
- Namenode, Datanode, Block, Goals of HDFS
- What is MapReduce?
- The Algorithm

What is Big Data

Massive amount of data which cannot be stored, processed and analyzed using the traditional ways

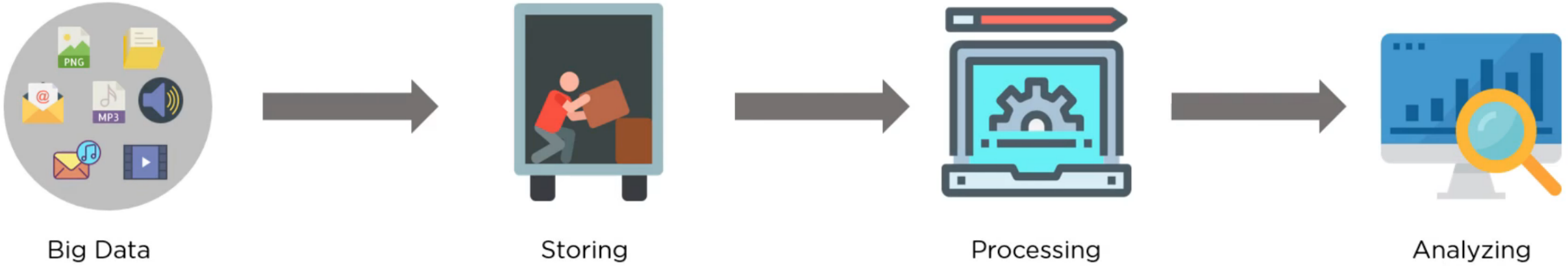


HADOOP IS THE SOLUTION

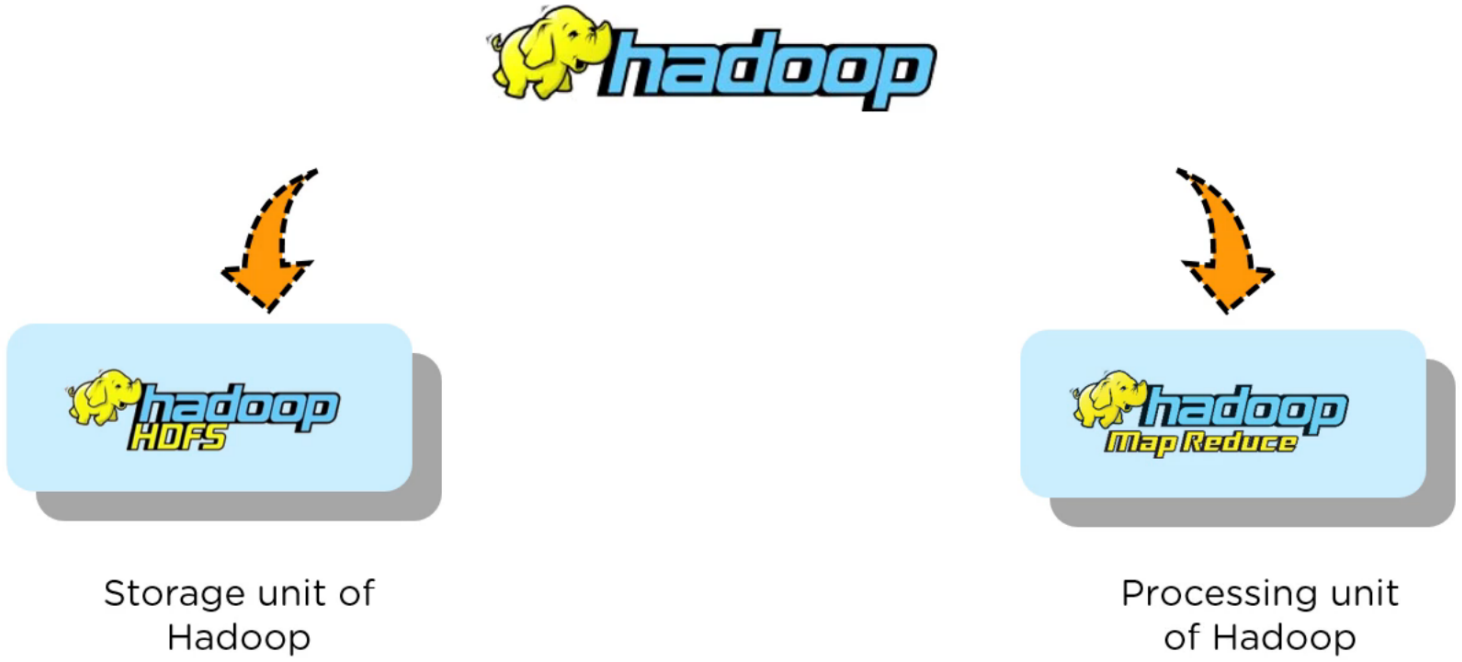


What is HADOOP?

Hadoop is a framework that manages big data storage in a distributed way and processes it parallelly



Components of Hadoop



Storage unit of Hadoop

Processing unit of Hadoop

Need for Hadoop

Big data

Requirements

- Scalability
- Fault tolerance
- Recoverability

Hadoop

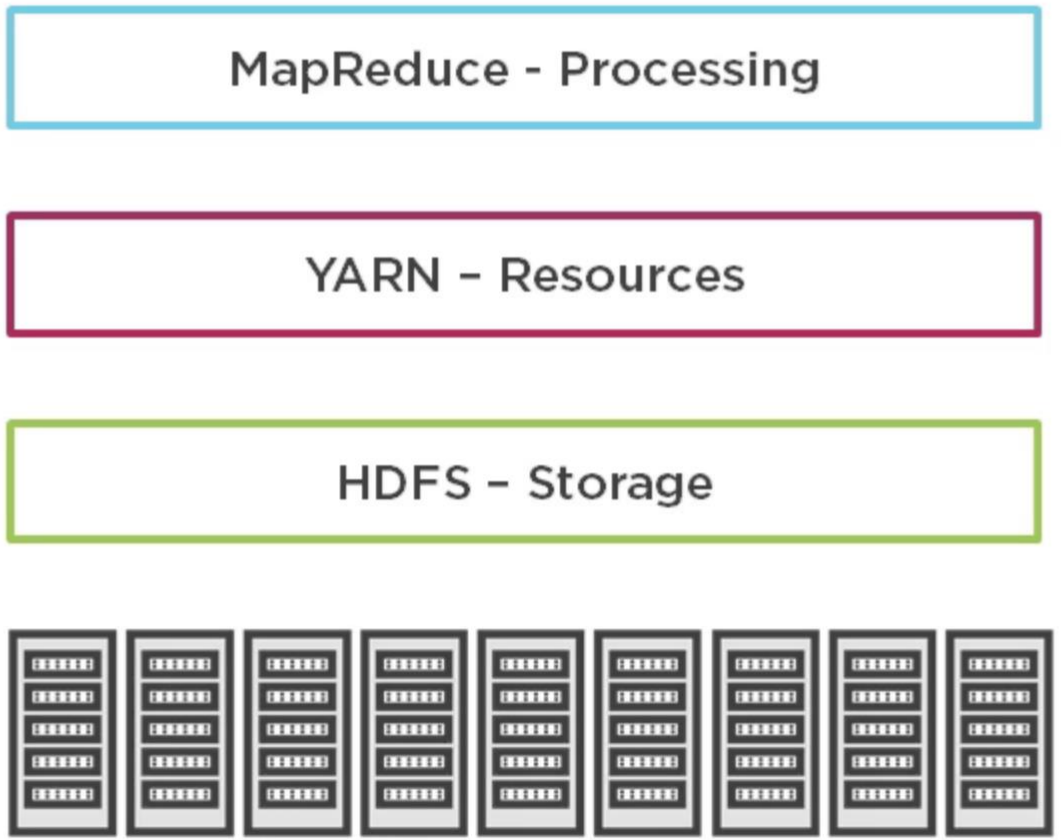
Based on ideas from Google

First released in 2006

Technology for big data

Key Hadoop Components

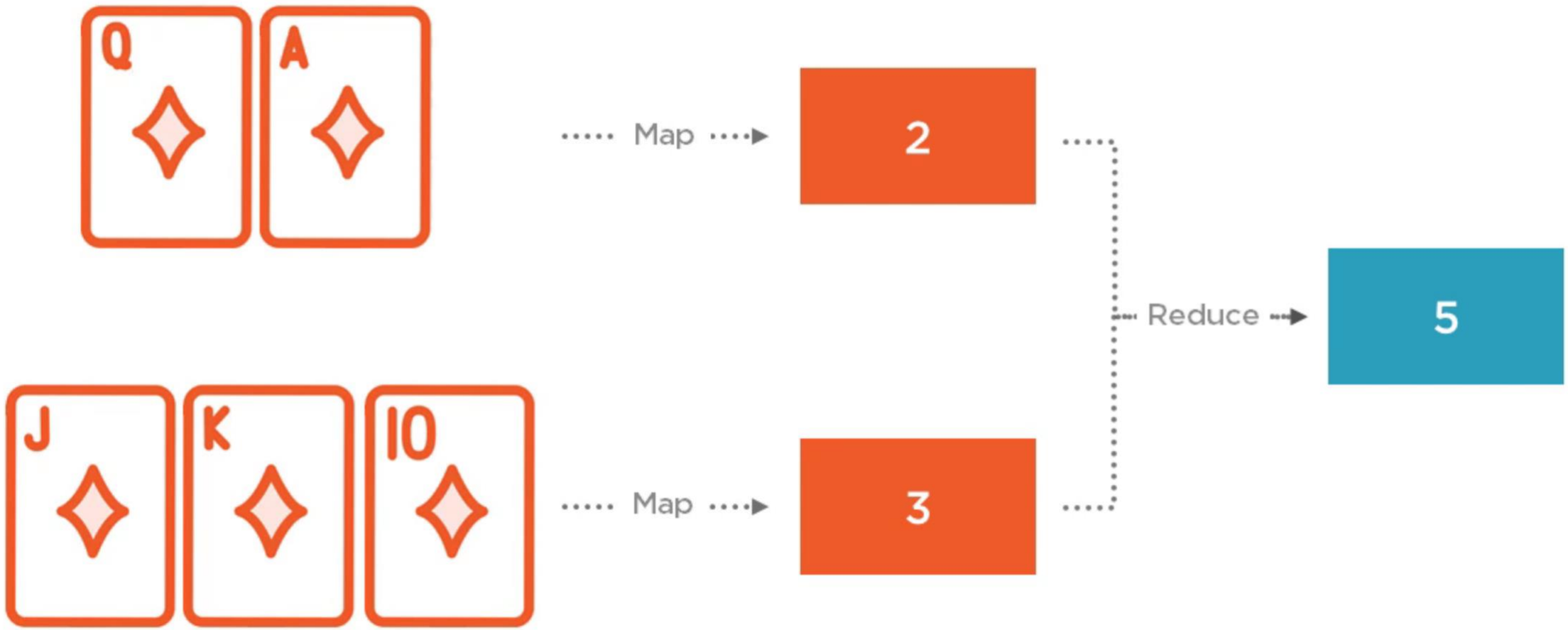
MapReduce is a programming model used for efficient processing in parallel over large data-sets in a distributed manner. The data is first split and then combined to produce the final result.



YARN is responsible for allocating system resources to the various applications running in a Hadoop cluster and scheduling tasks to be executed on different cluster nodes.

HDFS stands for Hadoop Distributed File System. HDFS operates as a distributed file system designed to run on commodity hardware. HDFS is fault-tolerant and designed to be deployed on low-cost, commodity hardware.

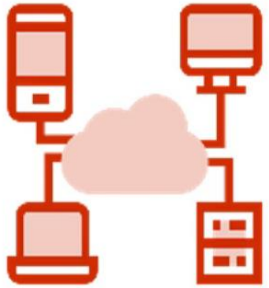
MapReduce Example



What is YARN?

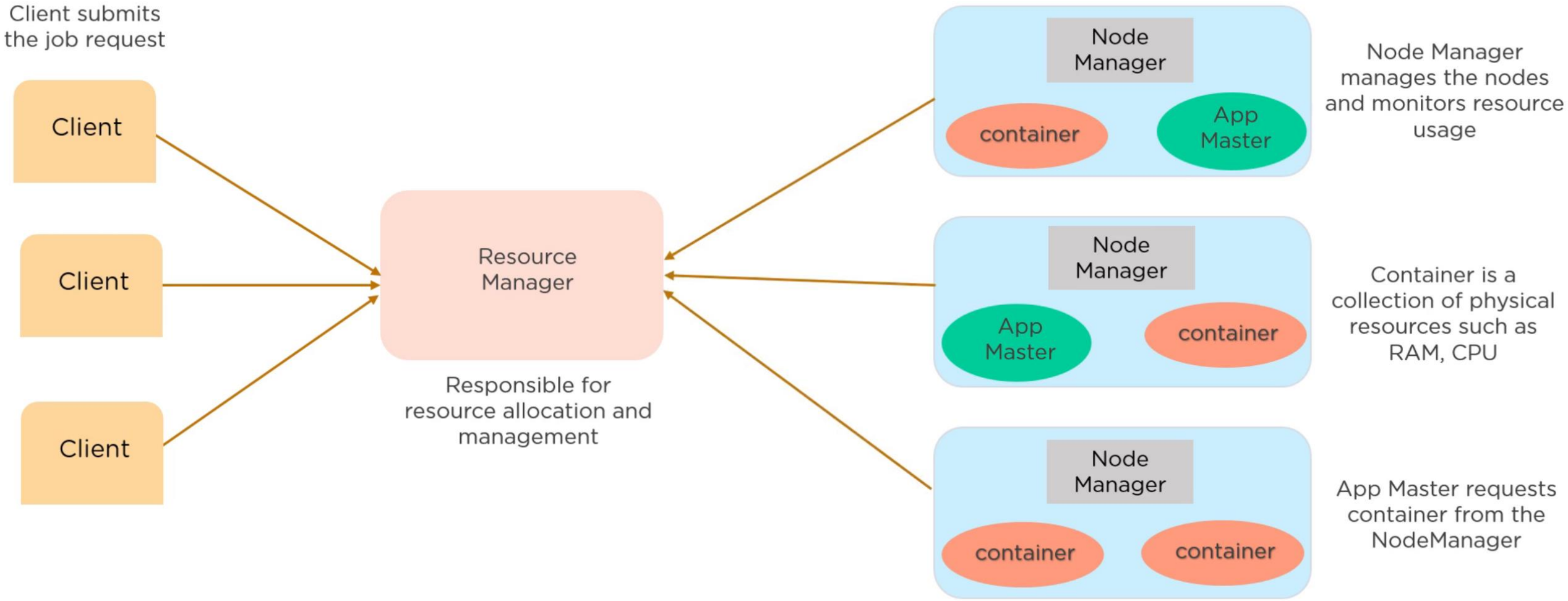
YARN – Yet Another Resource Negotiator

Acts like an OS
to Hadoop 2

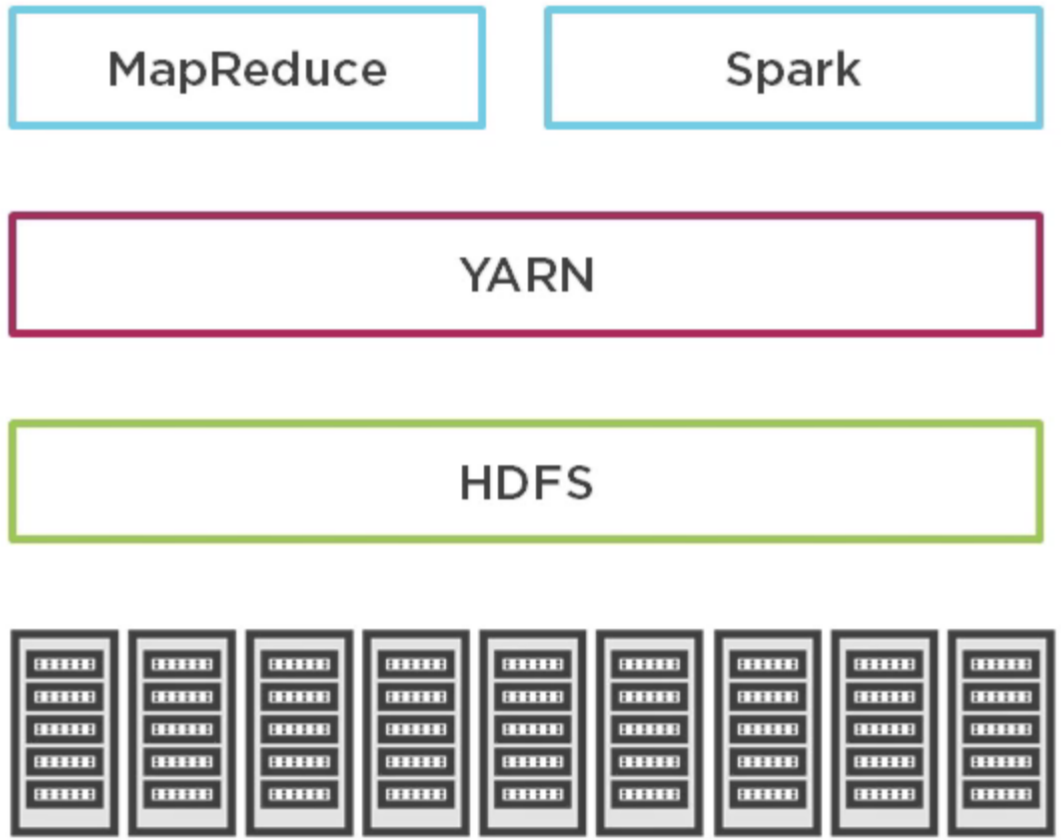


Responsible for managing
cluster resources

Does job scheduling


Client submits the job request
Client
Client
Client
Resource Manager
Responsible for resource allocation and management
Node Manager
container
App Master
Node Manager manages the nodes and monitors resource usage
Node Manager
App Master
container
Container is a collection of physical resources such as RAM, CPU
Node Manager
container
container
App Master requests container from the NodeManager

Machine Learning Frameworks



Processing on Hadoop

MapReduce	Spark
Uses HDFS for intermediate data	In-memory processing
Slower performance	Faster performance
Cheaper	More expensive
Great for batch processing	Great for iterative algorithms

Machine Learning Frameworks

MLlib

Included in Spark
Classification, recommenders
Clustering

Mahout

Runs on Spark
Classification, recommenders
Clustering

Deep Learning Frameworks

MXNet

Preferred by Amazon
Requires GPUs
Thriving ecosystem

TensorFlow

Created by Google
Requires GPUs
Thriving ecosystem

Notebooks

Notes

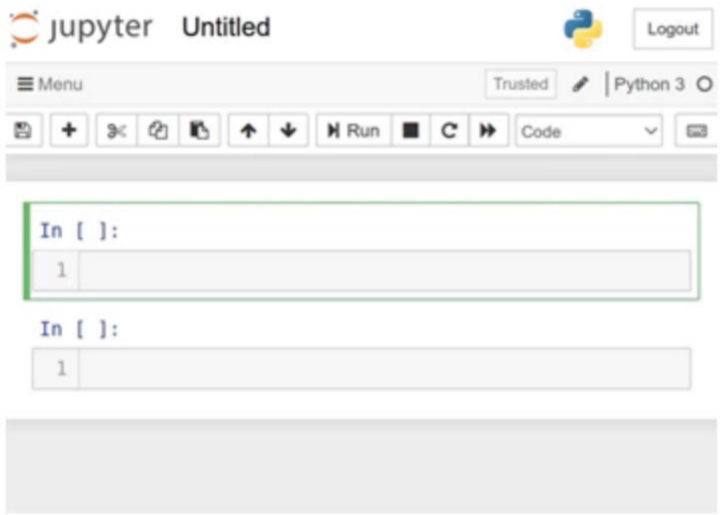
Code

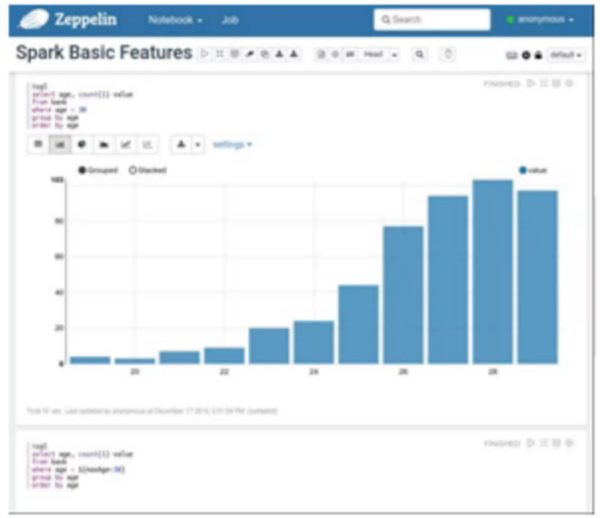
Visualizations

Popular Notebooks

Jupyter	Zeppelin
Most popular notebook	Less popular
Use JupyterHub for multiple users	Supports multiple users out of the box

Popular Notebooks





Jupyter

More established

Zeppelin

Newer project

Hue

Hadoop user experience

Web-based interface for end users

Execute SQL queries

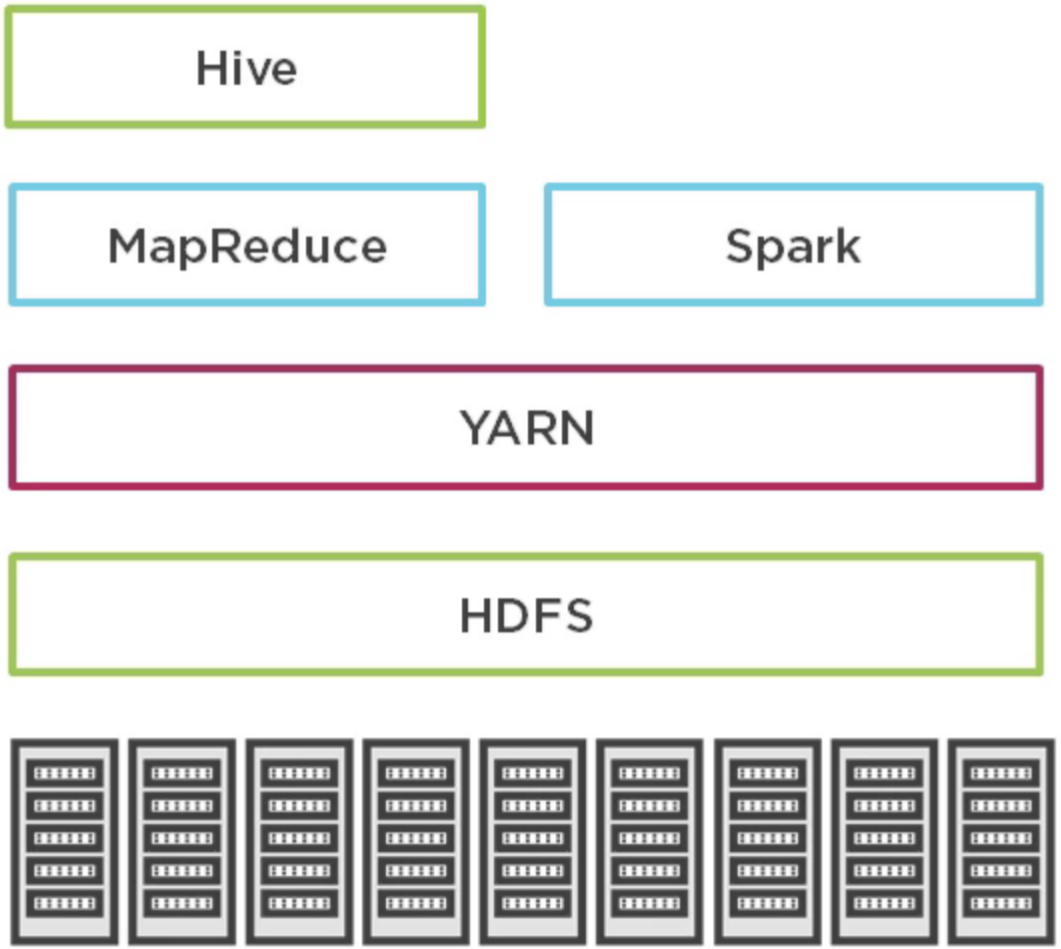
Manage files in HDFS

Querying Big Data

Spark SQL	Presto
Spark module	Complicated setup – use Athena
Avro, Orc, Parquet, Json, JDBC	Similar to Spark SQL
Not a relational database replacement	Not a relational database replacement

Hadoop Tools - Hive

Apache **Hive** is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis. Hive gives an SQL-like interface to query data stored in various databases and file systems that integrate with Hadoop



Hive Components

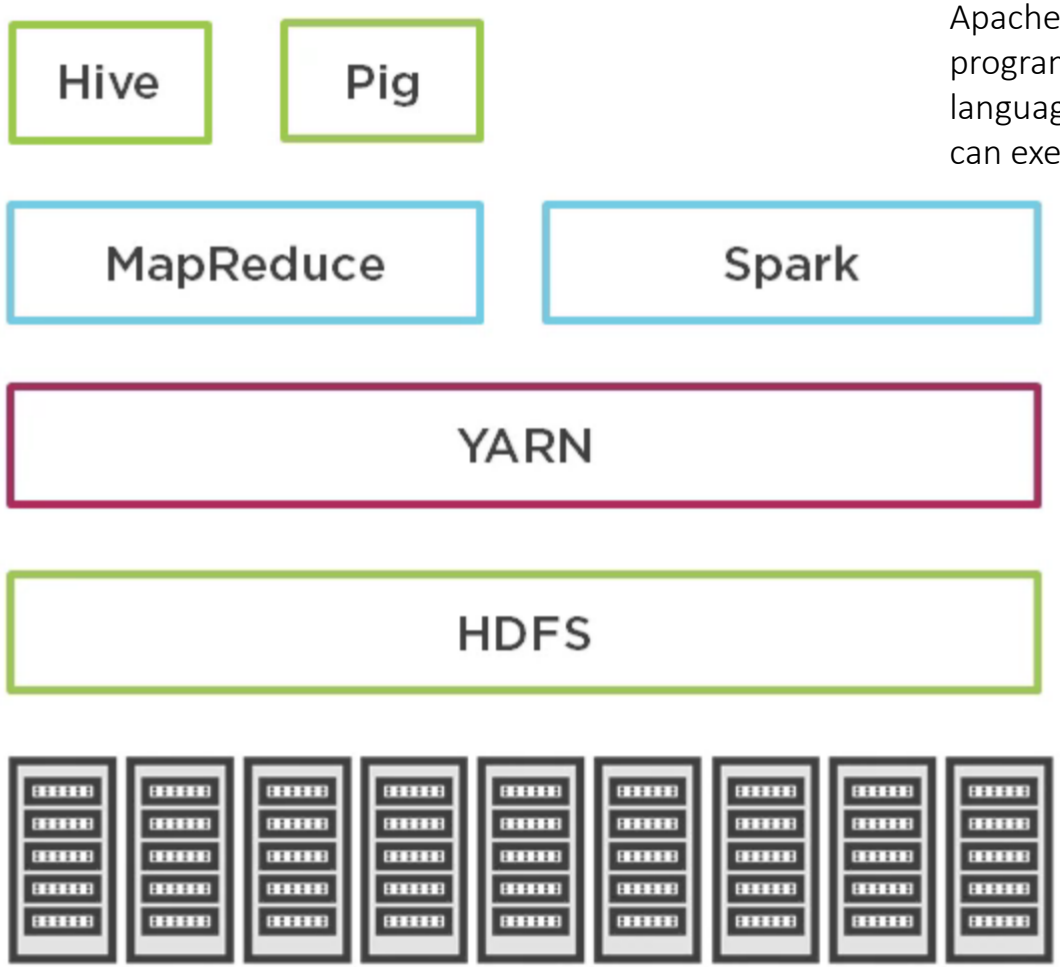
Hive metastore

- Single source of truth on schemas
- AWS Glue Data Catalog

HCatalog

- Helps connect to the Hive metastore

Hadoop Tools - Pig

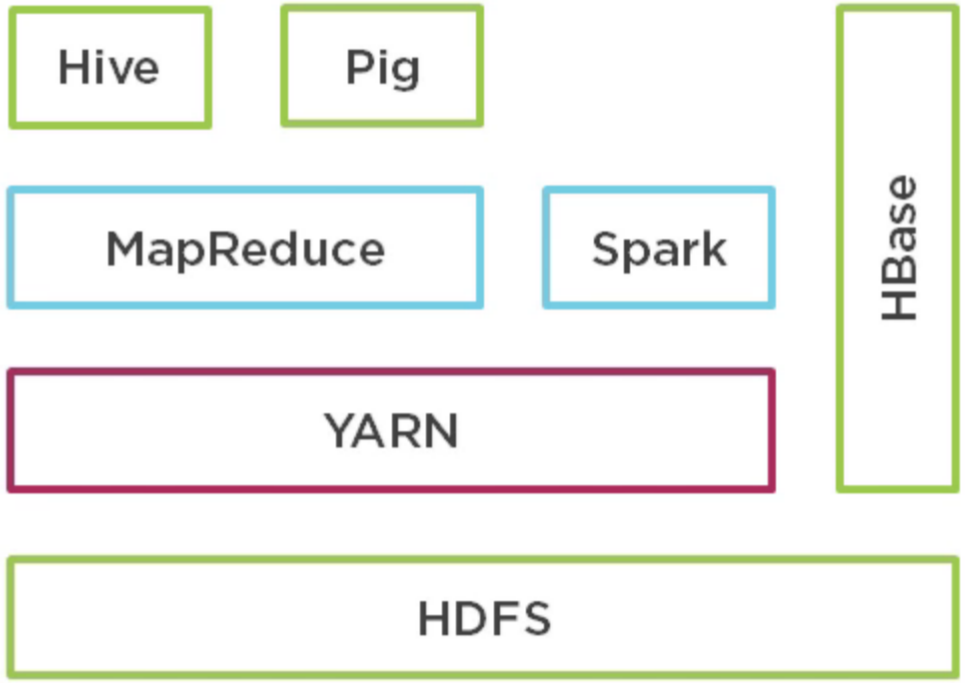


Apache Pig is a high-level platform for creating programs that run on Apache Hadoop. The language for this platform is called Pig Latin. Pig can execute its Hadoop jobs in MapReduce

Hive vs Pig

Hive	Pig
Declarative language (HQL)	Procedural language (Pig Latin)
Used by data scientists, analysts	Used by researchers, programmers
Better suited for structured data	Better suited for semi-structured data

Hadoop Tools - HBase



HBase is a column-oriented, non-relational database. This means that data is stored in individual columns, and indexed by a unique row key. This architecture allows for rapid retrieval of individual rows and columns and efficient scans over individual columns within a table.

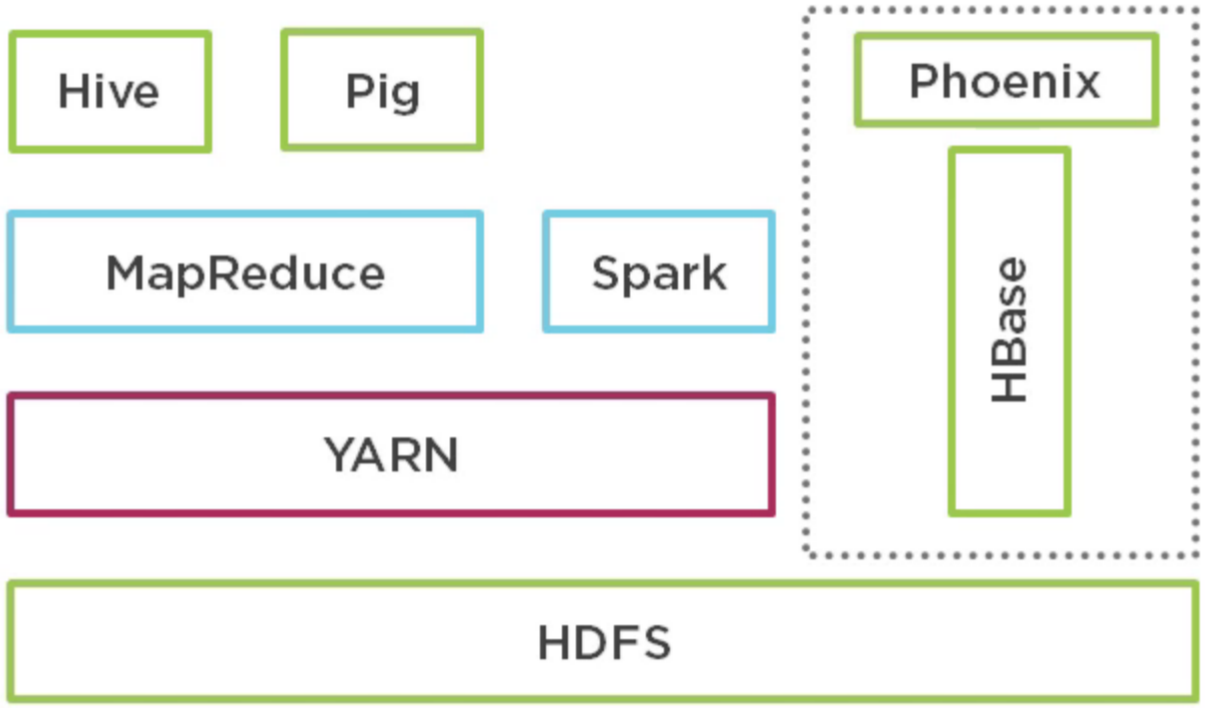
HBase

Key-value store

Used for variable schemas

Not a replacement for relational databases

Hadoop Tools - Phoenix



Apache Phoenix is an open source, massively parallel, relational database engine supporting OLTP for Hadoop using Apache HBase as its backing store.

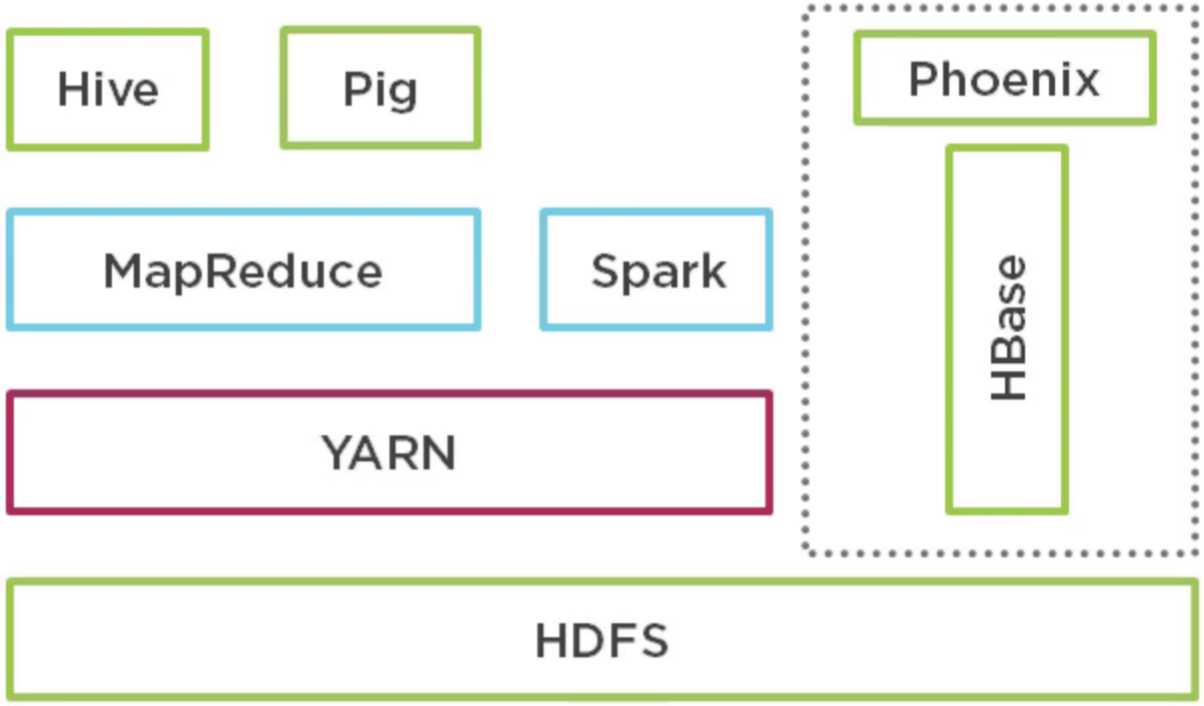
Phoenix

OLTP in Hadoop

JDBC driver

Integrates with Hive, Pig, Spark, MapReduce

Hadoop Tools - Sqoop



Sqoop is a command-line interface application for transferring data between relational databases and Hadoop.

Oozie

Workflow scheduler

Hadoop jobs: Pig, Hive, Sqoop, Spark, shell

Directed acyclic graph

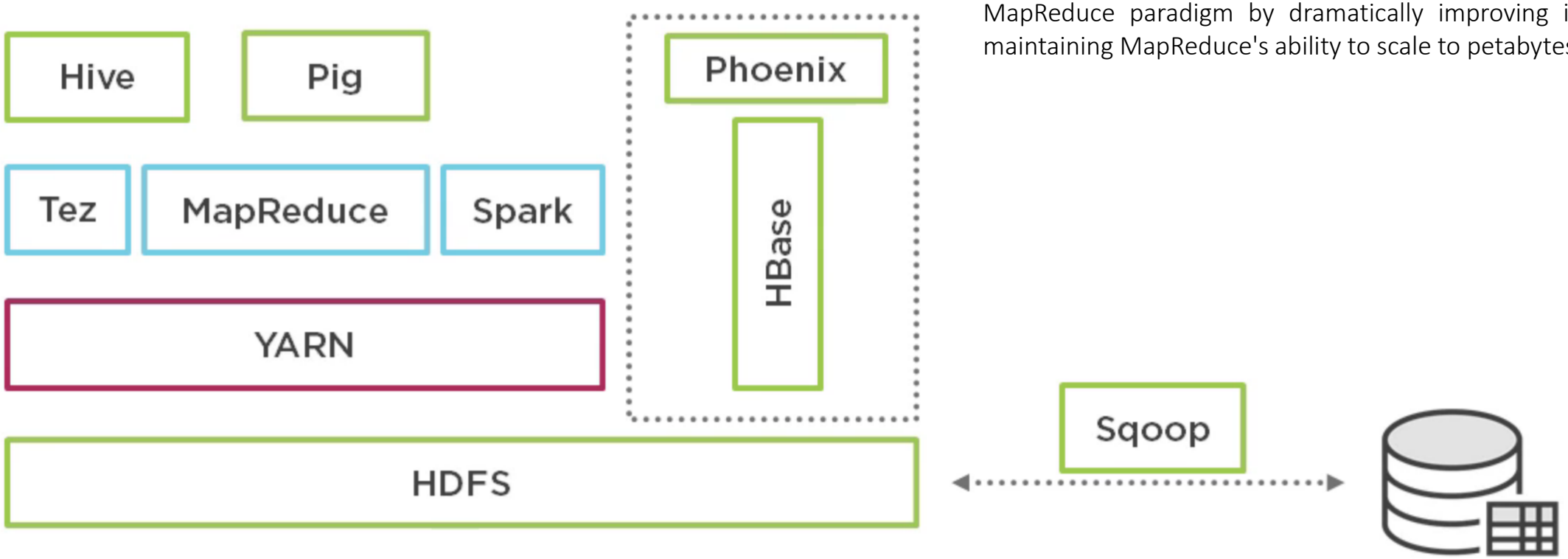
Directed Acyclic Graph

Directed Acyclic Graph

Directed Acyclic Graph

Hadoop Tools - Tez

Apache™ **Tez** is an extensible framework for building high performance batch and interactive data processing applications, coordinated by YARN in Apache Hadoop. Tez improves the MapReduce paradigm by dramatically improving its speed, while maintaining MapReduce's ability to scale to petabytes of data.



Spark

Supports Scala, Java, Python

Batch processing

Use Livy to submit Spark jobs with REST api

Data Processing

Batch	Stream
Process large volumes of data	Process very small volumes of data
Data is collected over a time interval	Data is collected continuously
Optimized for processing large data	Optimized for instant results

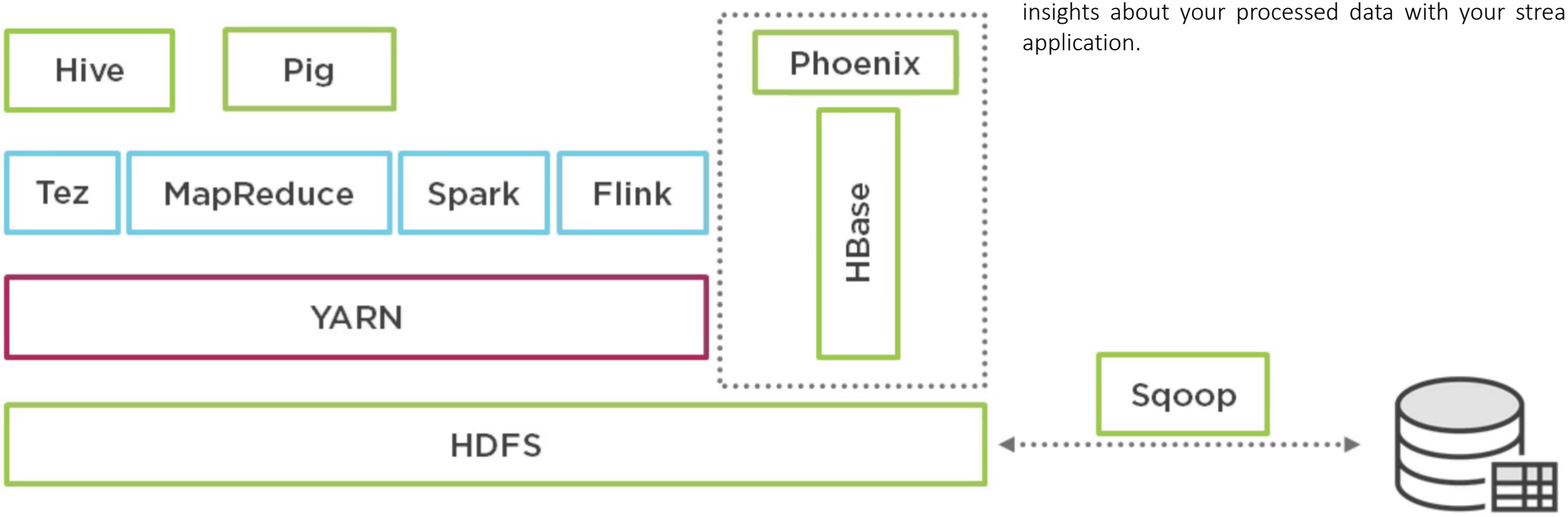
Spark Streaming

Spark component

Spark integration

Hadoop Tools - Flink

Flink is a distributed processing engine and a scalable data analytics framework. You can use Flink to process data streams at a large scale and to deliver real-time analytical insights about your processed data with your streaming application.

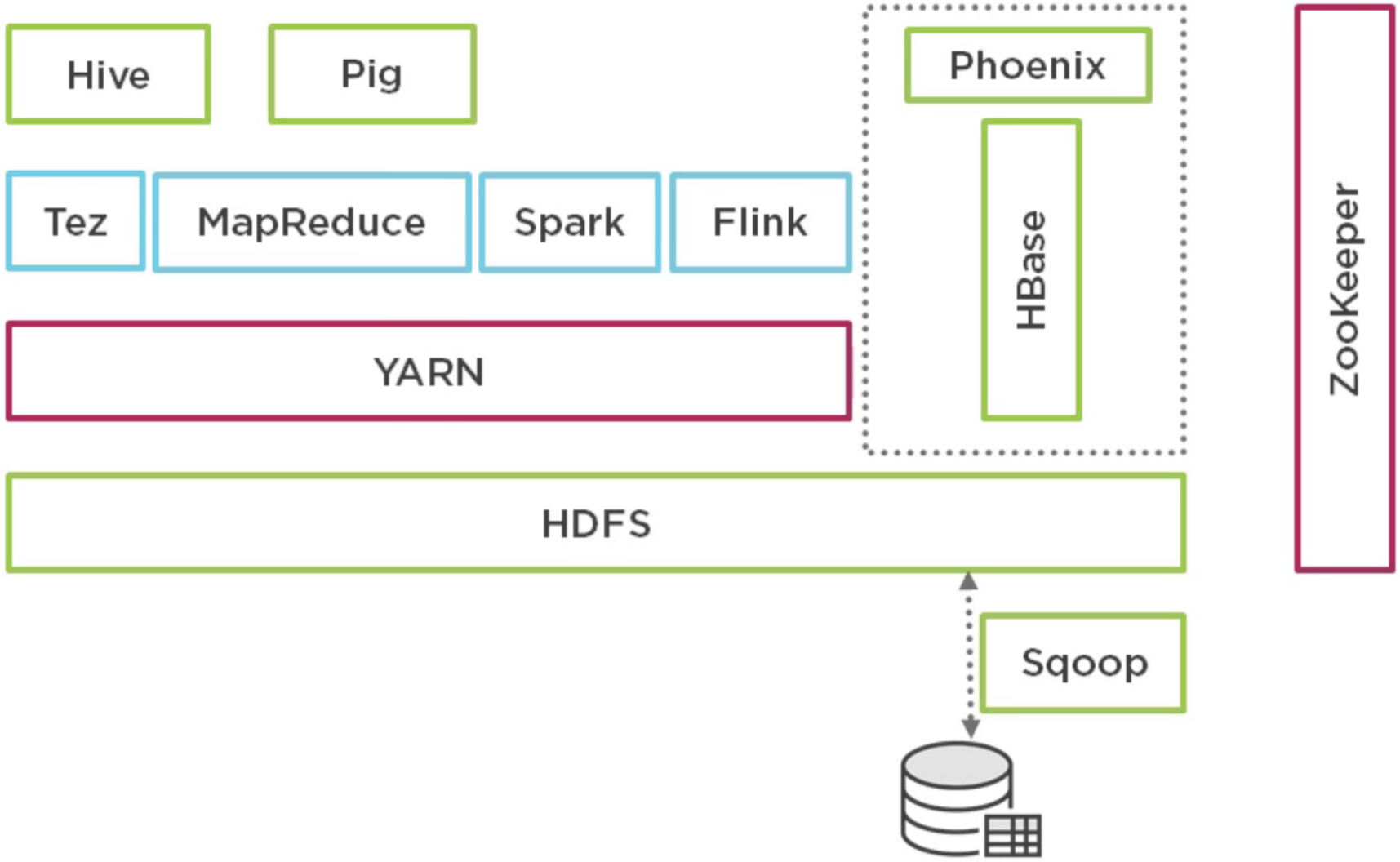


Flink

Faster than Spark Streaming

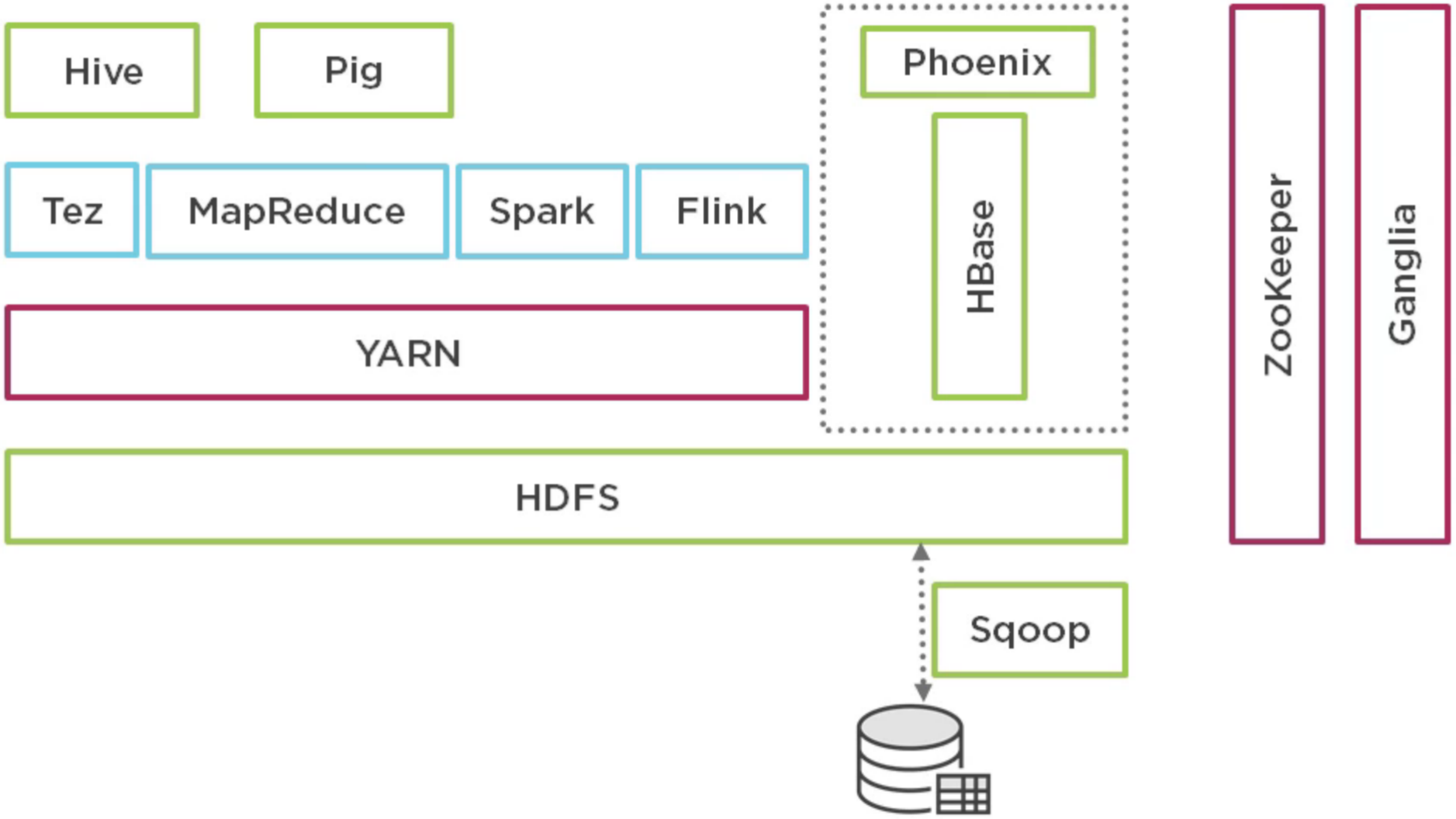
Requires a lot of memory

Hadoop Tools - Zookeeper



Apache **ZooKeeper** provides operational services for a Hadoop cluster. ZooKeeper provides a distributed configuration service, a synchronization service and a naming registry for distributed systems. Distributed applications use Zookeeper to store and mediate updates to important configuration information.

Hadoop Tools - Ganglia



Ganglia is cluster monitoring tool to monitor the health of distributed cluster of spark and hadoop.

Hadoop Distributed File System

File Browser

hortonwork.cloudapp.net:8000/filebrowser/view/user/hue

Search

hue

/user/hue/appl_daily.csv upload succeeded

File Browser

Search for file name | Rename | Move | Copy | Change Permissions | Download | Delete | New | Upload

Home / user / hue

Trash

Type	Name	Size	User	Group	Permissions	Date
	.		hue	hue	drwxr-xr-x	August 07, 2015 08:52 AM
	..		hdfs	hdfs	drwxr-xr-x	May 16, 2015 04:31 PM
	.Trash		hue	hue	drwxr-xr-x	August 07, 2015 08:37 AM
	.staging		hue	hue	drwx------	July 29, 2015 08:10 AM
	Pig_Examples		hue	hue	drwxr-xr-x	July 29, 2015 08:01 AM
	appl_daily.csv	580.9 KB	hue	hue	-rwxr-xr-x	August 07, 2015 08:52 AM
	jobsub		hue	hue	drwxrwxrwx	December 29, 2014 07:05 AM
	oozie		hue	hue	drwxrwxrwx	December 29, 2014 07:05 AM

Show 45 items per page. Showing 1 to 6 of 6 items, page 1 of 1.

appl_daily.csv - File Viewer

hortonwork.cloudapp.net:8000/filebrowser/view//user/hue/appl_daily.csv

Search

hue

Home / user / hue / **appl_daily.csv**

ACTIONS

View As Binary

Download

View File Location

Refresh

INFO

Last Modified
Aug. 7, 2015 8:52 a.m.

User
hue

Group
hue

Size
580.9 KB

Mode
100755

First Block | Previous Block | Next Block | Last Block

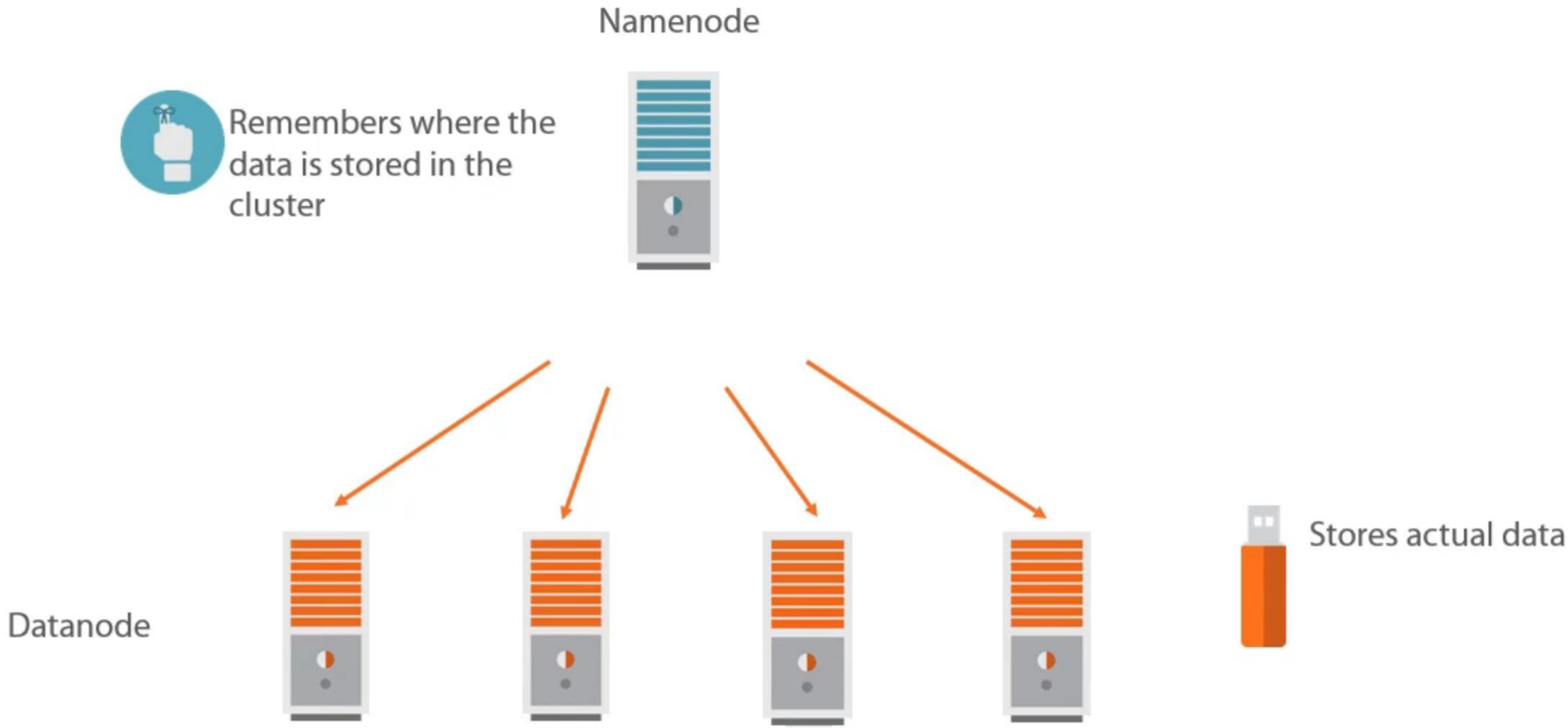
Viewing Bytes: 1 - 4096 of 594881 (4096 B block size)

```
Date,Open,High,Low,Close,Volume,Adj Close
2015-07-09,123.849998,124.059998,119.220001,120.07,77821600,120.07
2015-07-08,124.480003,124.639999,122.540001,122.57,60490200,122.57
2015-07-07,125.889999,126.150002,123.769997,125.690002,46716100,125.690002
2015-07-06,124.940002,126.230003,124.849998,126.00,27900200,126.00
2015-07-02,126.43,126.690002,125.769997,126.440002,27122500,126.440002
2015-07-01,126.900002,126.940002,125.989998,126.599998,30128600,126.599998
2015-06-30,125.57,126.120003,124.860001,125.43,43849800,125.43
2015-06-29,125.459999,126.470001,124.480003,124.529999,48911400,124.529999
2015-06-26,127.669998,127.989998,126.510002,126.75,42111000,126.75
2015-06-25,128.860001,129.199997,127.50,127.50,31816700,127.50
2015-06-24,127.209999,129.800003,127.120003,128.110001,54964900,128.110001
2015-06-23,127.480003,127.610001,126.879997,127.029999,30137100,127.029999
2015-06-22,127.489998,128.059998,127.080002,127.610001,33833500,127.610001
2015-06-19,127.709999,127.82,126.400002,126.599998,54181300,126.599998
2015-06-18,127.230003,128.309998,127.220001,127.879997,35241100,127.879997
2015-06-17,127.720001,127.879997,126.739998,127.300003,32768500,127.300003
2015-06-16,127.029999,127.849998,126.370003,127.599998,31404000,127.599998
2015-06-15,126.099998,127.239998,125.709999,126.919998,39842600,126.919998
2015-06-12,128.190002,128.330002,127.110001,127.169998,36754200,127.169998
```

HDFS

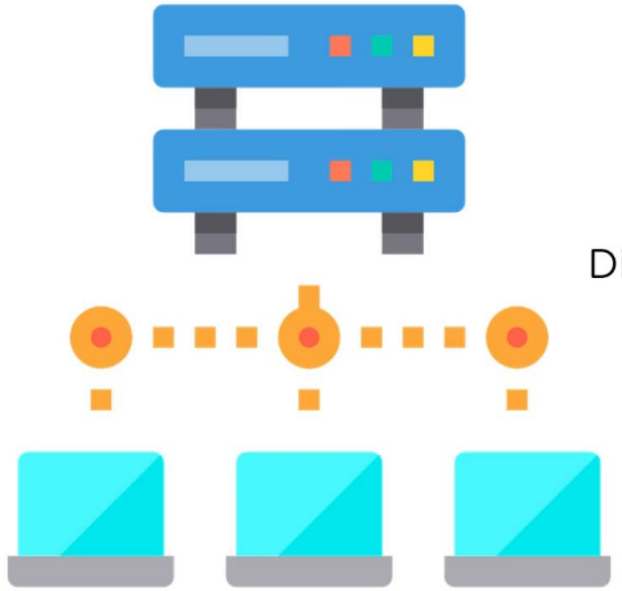
Distributed file system | Batch processing | Written in Java

HDFS consists of a NameNode
and DataNode


Namenode
Remembers where the data is stored in the cluster
Datanode
Stores actual data

What is HDFS?

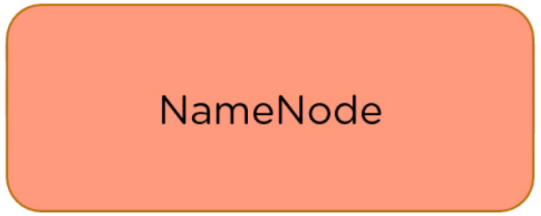
Hadoop Distributed File System (HDFS) is specially designed for storing huge datasets in commodity hardware



Distributed storage

What is HDFS?

Hadoop Distributed File System (HDFS) has two core components NameNode and DataNode



There is only one NameNode

What is HDFS?

Hadoop Distributed File System (HDFS) has two core components NameNode and DataNode

NameNode

There is only one NameNode

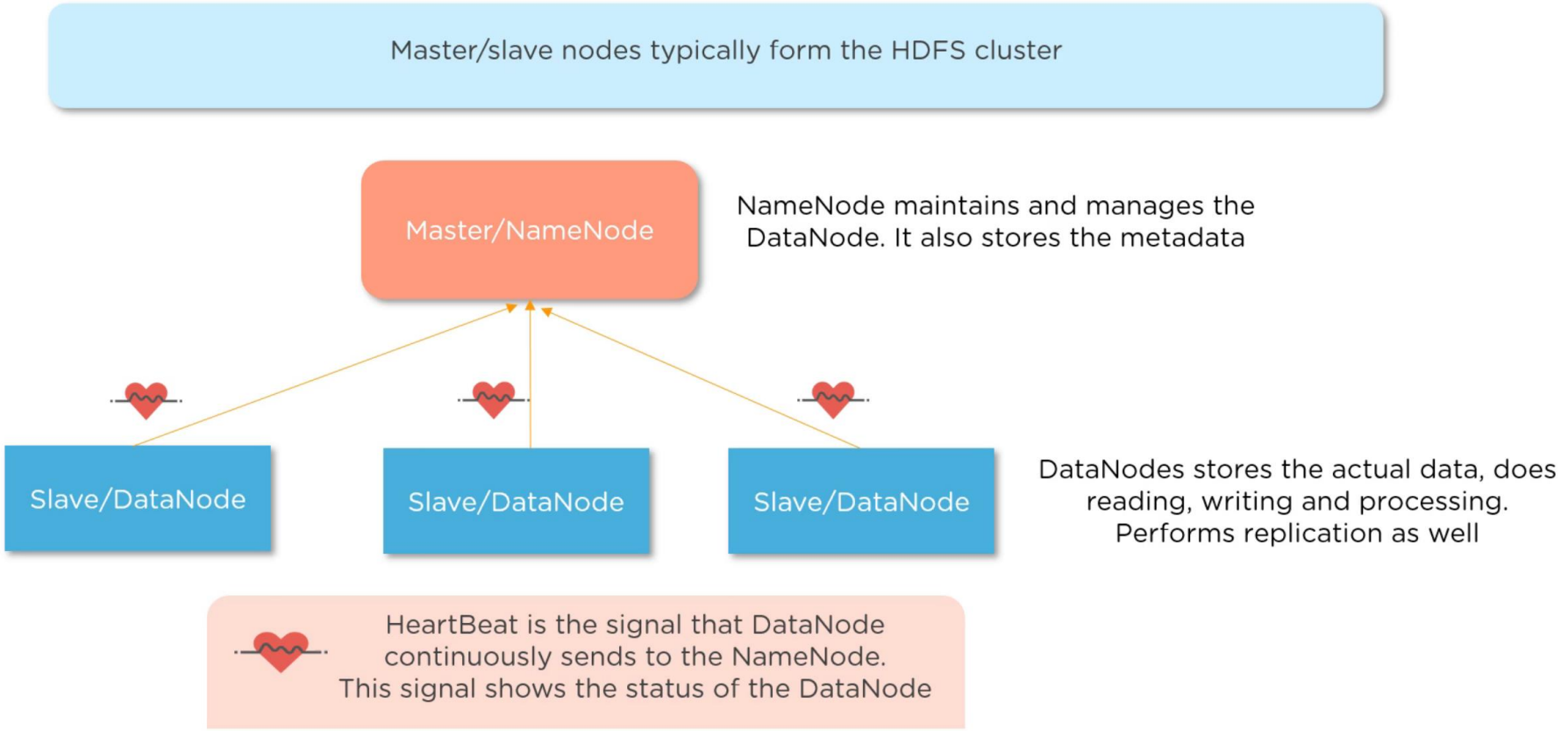
There can be multiple DataNodes

DataNode

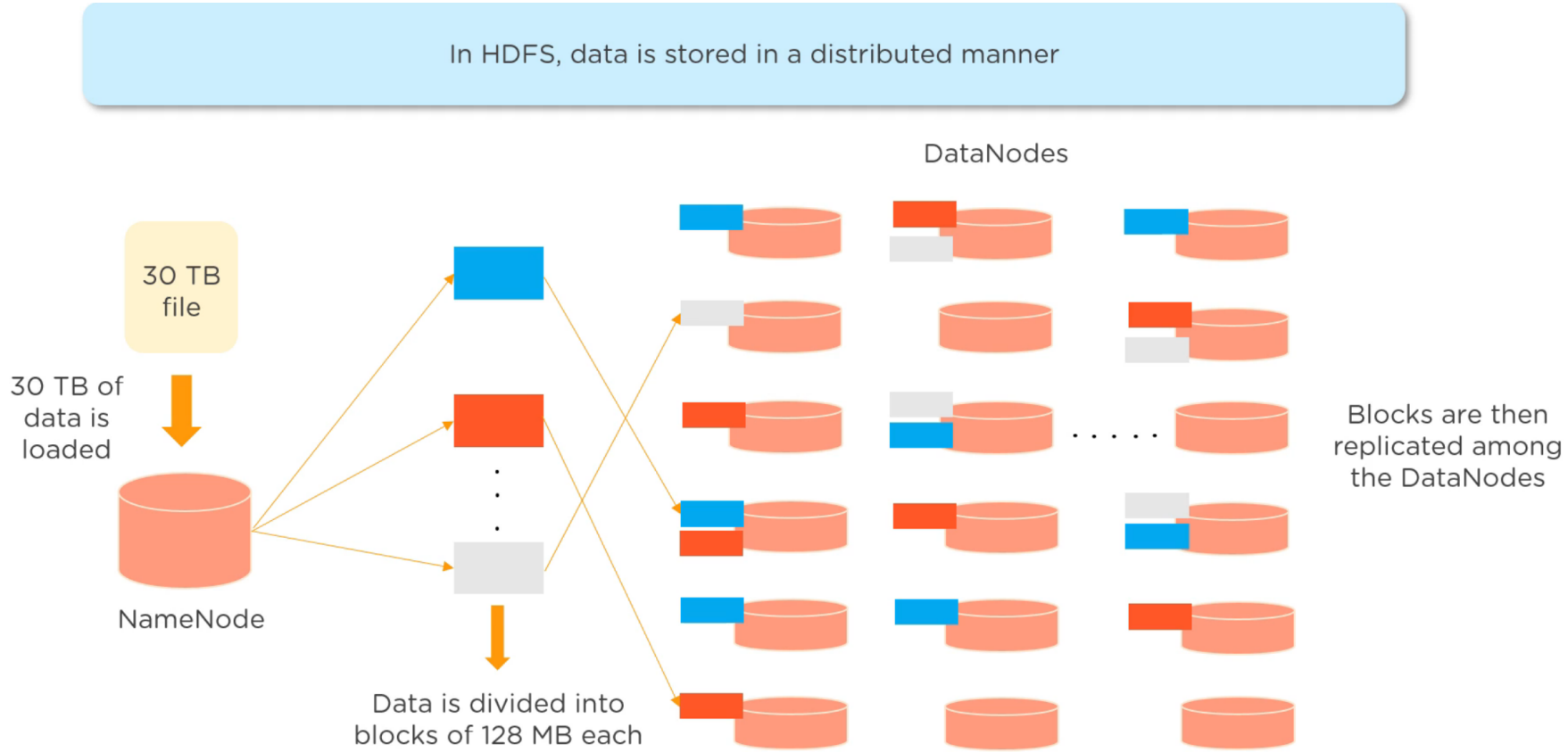
DataNode

DataNode

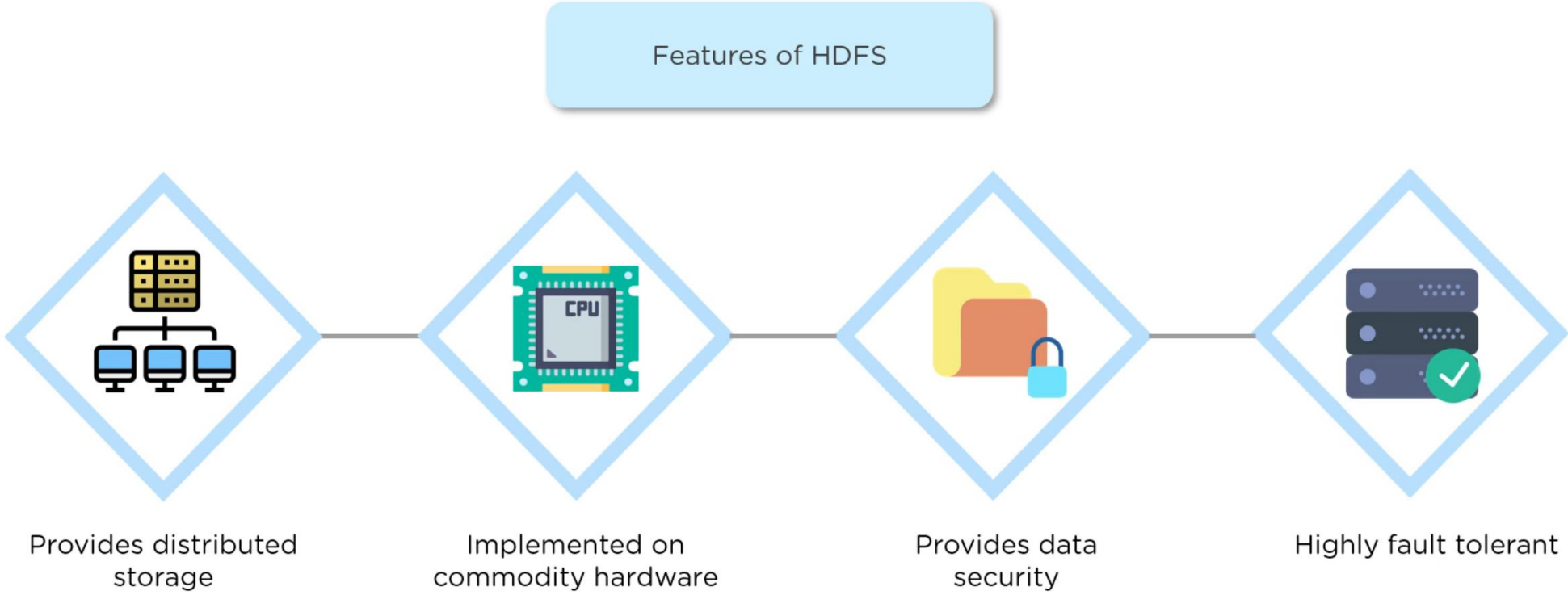
What is HDFS?


Master/slave nodes typically form the HDFS cluster
Master/NameNode
NameNode maintains and manages the DataNode. It also stores the metadata
Slave/DataNode
Slave/DataNode
Slave/DataNode
DataNodes stores the actual data, does reading, writing and processing. Performs replication as well
HeartBeat is the signal that DataNode continuously sends to the NameNode. This signal shows the status of the DataNode

What is HDFS?


In HDFS, data is stored in a distributed manner
DataNodes
30 TB file
30 TB of data is loaded
NameNode
Data is divided into blocks of 128 MB each
Blocks are then replicated among the DataNodes

Features of HDFS



HDFS

Built on commodity hardware

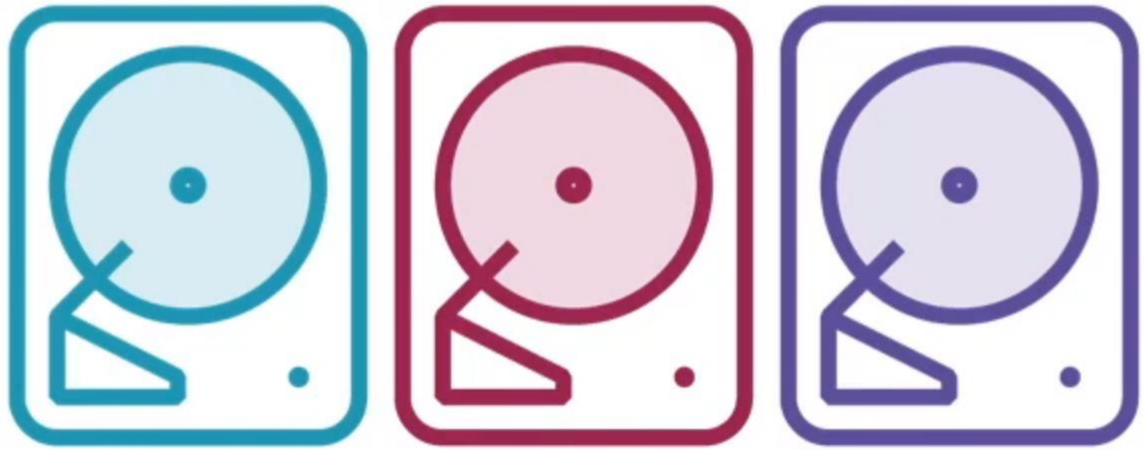
Highly fault tolerant, hardware failure is the norm

Suited to batch processing - data access has high throughput rather than low latency

Supports very large data sets

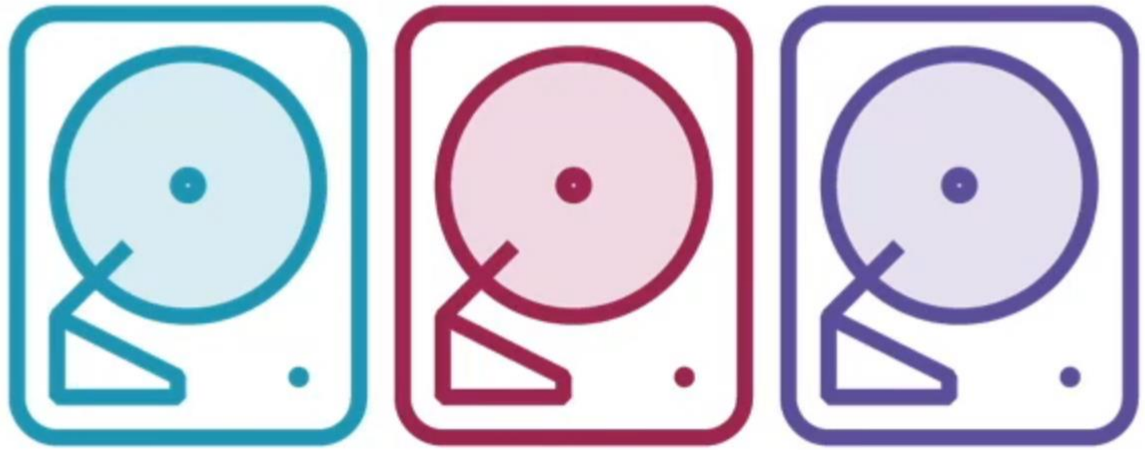
HDFS

Manage file storage across multiple disks



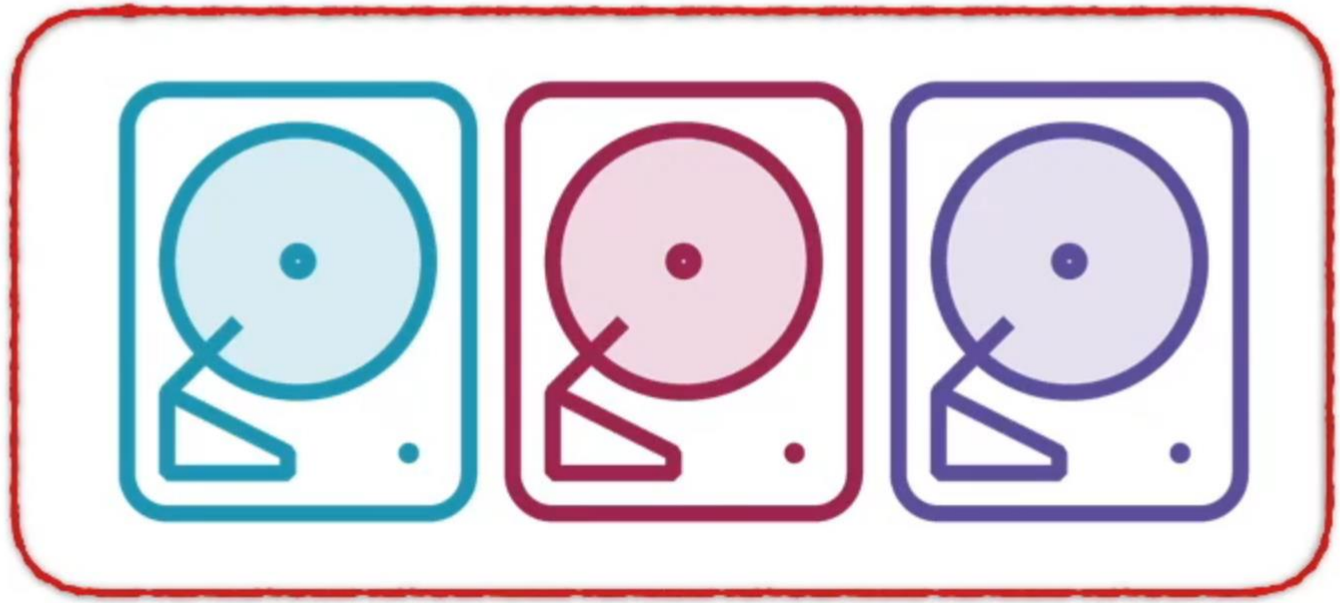
HDFS

Each disk on a different machine in a cluster



HDFS

A cluster of machines

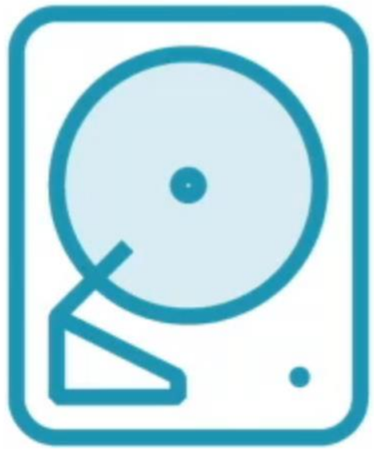


HDFS

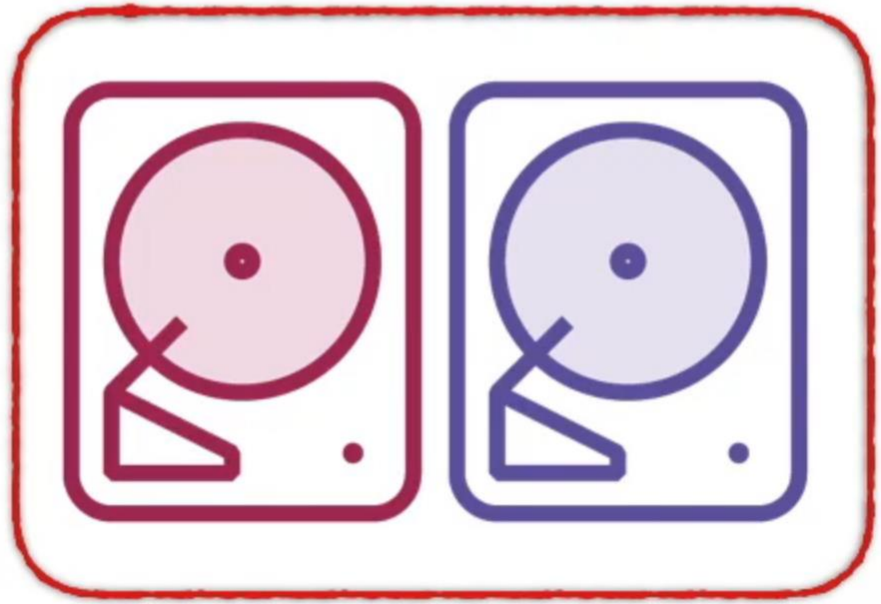
1 node is the master node

HDFS

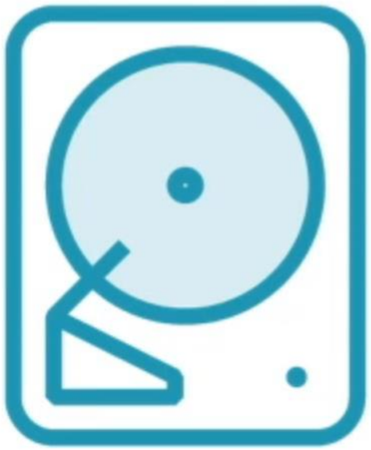
Name node



Data nodes

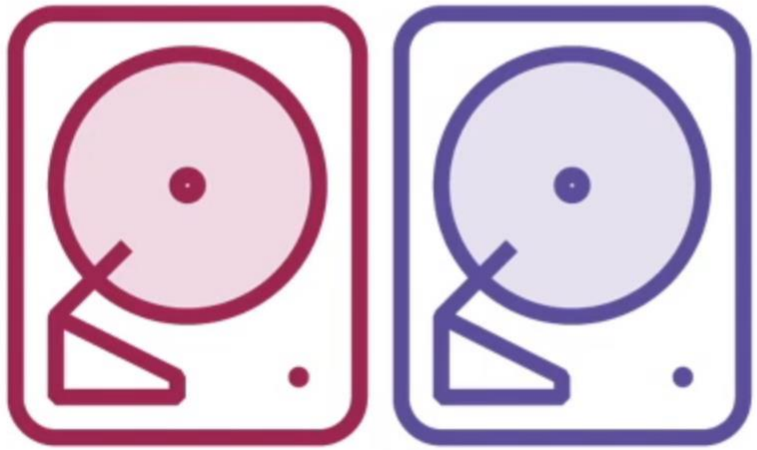


Name node



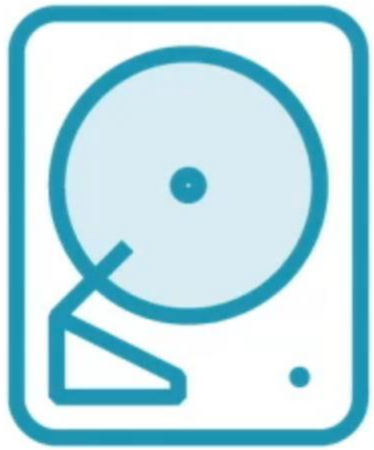
The name node is the table of contents

Data nodes



The data nodes hold the actual text in each page

Name node

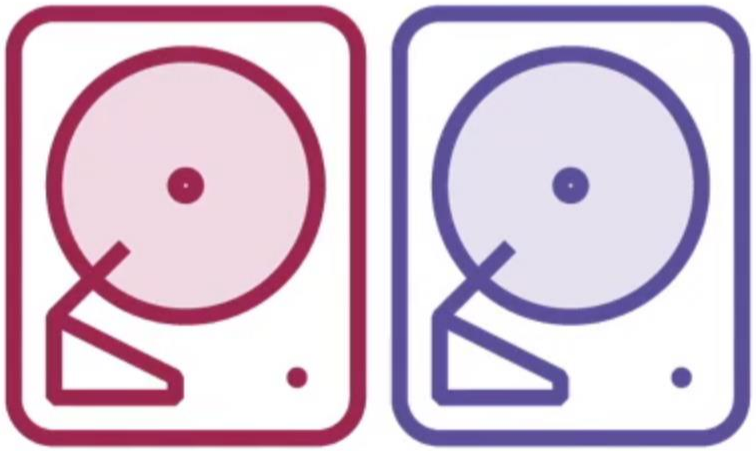


Manages the overall file system

Stores

- The directory structure
- Metadata of the files

Data nodes



Physically stores the data in the files

Storing a File in HDFS

Distributed indexing

Collections are often so large that we cannot perform index construction efficiently on a single machine. This is particularly true of the World Wide Web for which we need large computer clusters [*]to construct any reasonably sized web index. Web search engines, therefore, use distributed indexing algorithms for index construction. The result of the construction process is a distributed index that is partitioned across several machines - either according to term or according to document. In this section, we describe distributed indexing for a
term-partitioned index . Most large search engines prefer a document-partitioned index (which can be easily generated from a term-partitioned index). We discuss this topic further in Section 20.3 (page [*]).

The distributed index construction method we describe in this section is an application of MapReduce , a general architecture for distributed computing. MapReduce is designed for large computer clusters. The point of a cluster is to solve large computing problems on cheap commodity machines or nodes that are built from standard parts (processor, memory, disk) as opposed to on a supercomputer with specialized hardware. Although hundreds or thousands of machines are available in such clusters, individual machines can fail at any time. One requirement for robust distributed indexing is, therefore, that we divide the work up into chunks that we can easily assign and - in case of failure - reassign. A master node directs the process of assigning and reassigning tasks to individual worker nodes.

The map and reduce phases of MapReduce split up the computing job into chunks that standard machines can process in a short time. The various steps of MapReduce are shown in Figure 4.5 and an example on a collection consisting of two documents is shown in Figure 4.6 . First, the input data, in our case a collection of web pages, are split into n splits where the size of the split is chosen to ensure that the work can be distributed evenly (chunks should not be too large) and efficiently (the total number of chunks we need to manage should not be too large); 16 or 64 MB are good sizes in distributed indexing. Splits are not preassigned to machines, but are instead assigned by the master node on an ongoing basis: As a machine finishes processing one split, it is assigned the next one. If a machine dies or becomes a laggard due to hardware problems, the split it is working on is simply reassigned to another machine.

Figure 4.5: An example of distributed indexing with MapReduce. Adapted from Dean and Ghemawat (2004).
\includegraphics[width=11.5cm]{art/mapreduce2.eps}
In general, MapReduce breaks a large computing problem into smaller parts by recasting it in terms of manipulation of key-value pairs . For indexing, a key-value pair has the form (termID,docID). In distributed indexing, the mapping from terms to termIDs is also distributed and therefore more complex than in single-machine indexing. A simple solution is to maintain a (perhaps precomputed) mapping for frequent terms that is copied to all nodes and to use terms directly (instead of termIDs) for infrequent terms. We do not address this problem here and assume that all nodes share a consistent term $\rightarrow$ termID mapping.

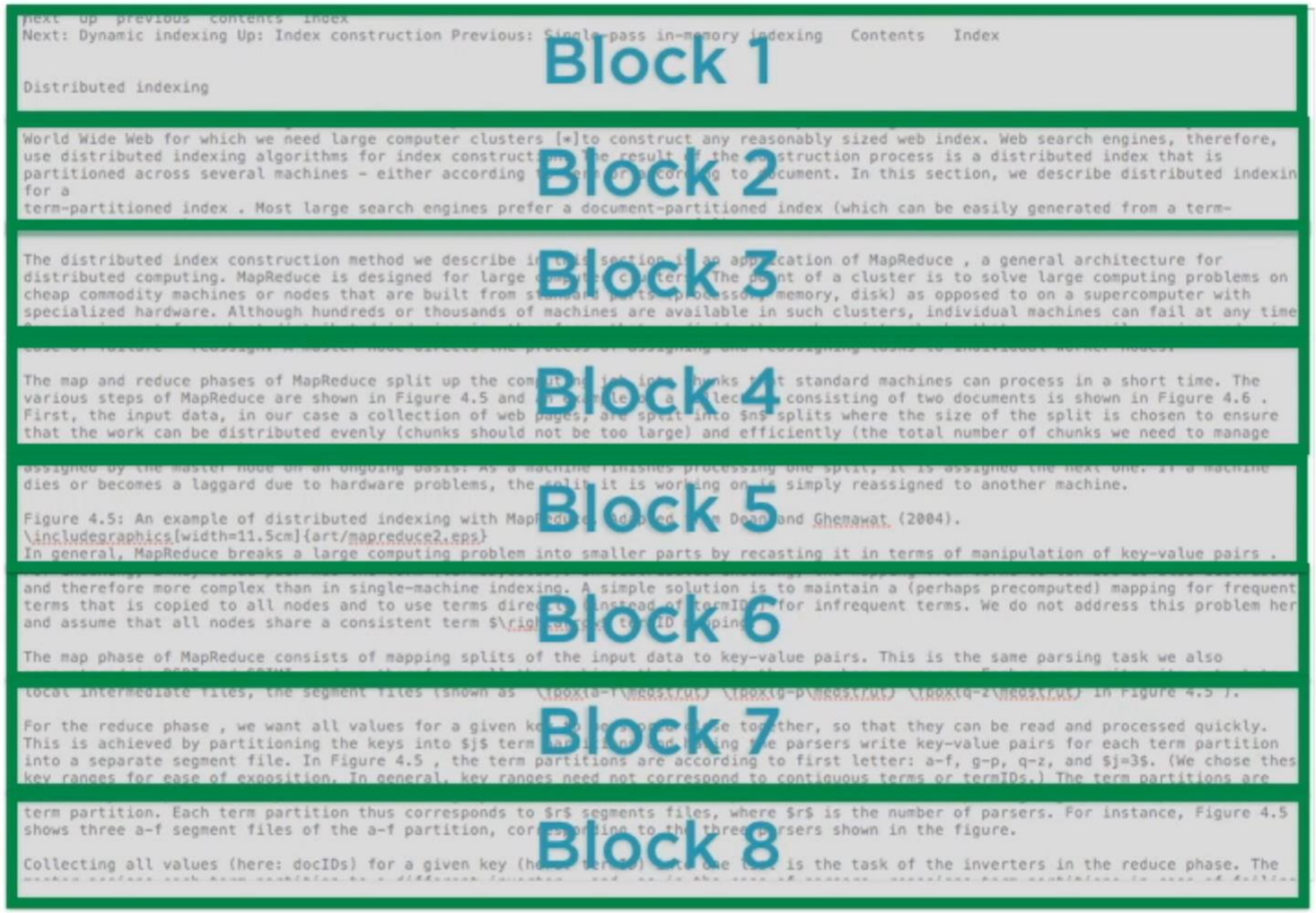
The map phase of MapReduce consists of mapping splits of the input data to key-value pairs. This is the same parsing task we also encountered in BSBI and SPIMI, and we therefore call the machines that execute the map phase parsers . Each parser writes its output to local intermediate files, the segment files (shown as \fbox{a-f\medstrut} \fbox{g-p\medstrut} \fbox{q-z\medstrut} in Figure 4.5).

For the reduce phase , we want all values for a given key to be stored close together, so that they can be read and processed quickly. This is achieved by partitioning the keys into j term partitions and having the parsers write key-value pairs for each term partition into a separate segment file. In Figure 4.5 , the term partitions are according to first letter: a-f, g-p, q-z, and $j=3$. (We chose these key ranges for ease of exposition. In general, key ranges need not correspond to contiguous terms or termIDs.) The term partitions are defined by the person who operates the indexing system (Exercise 4.6). The parsers then write corresponding segment files, one for each term partition. Each term partition thus corresponds to r segments files, where r is the number of parsers. For instance, Figure 4.5 shows three a-f segment files of the a-f partition, corresponding to the three parsers shown in the figure.

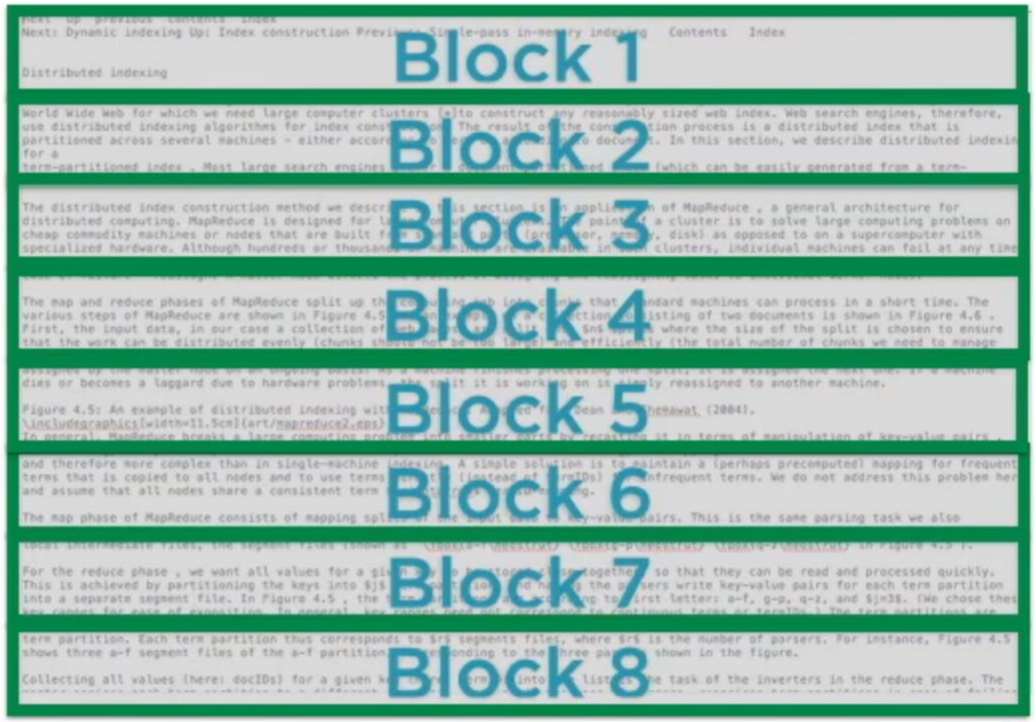
Collecting all values (here: docIDs) for a given key (here: termID) into one list is the task of the inverters in the reduce phase. The

A large text file

Storing a File in HDFS



Storing a File in HDFS



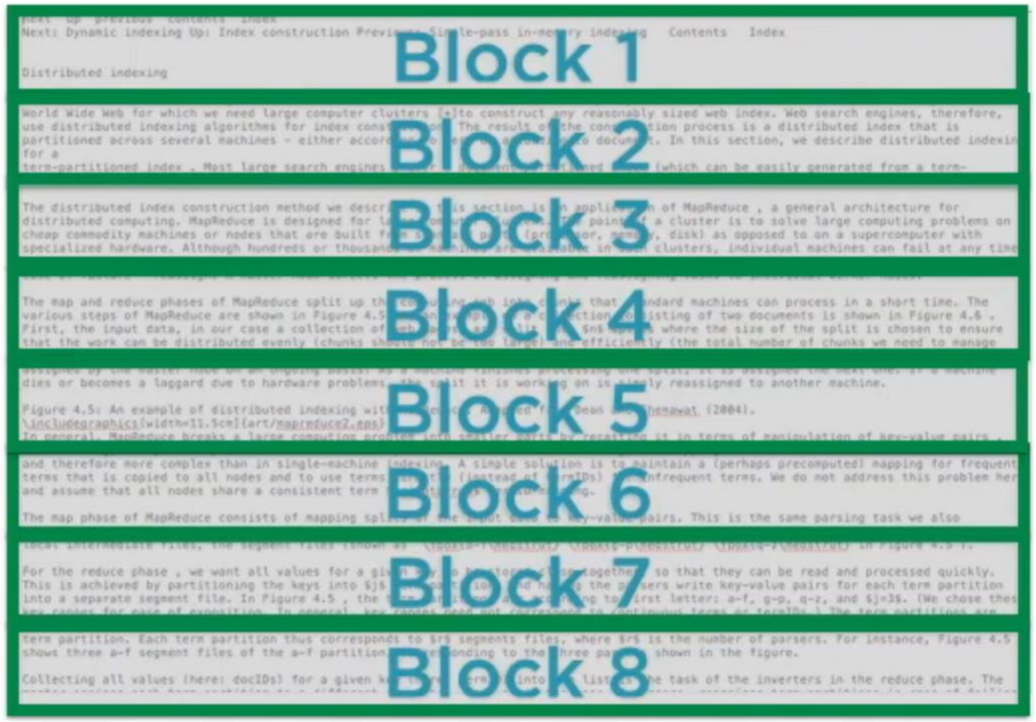
Break the data into blocks

Different length files are treated the same way

Storage is simplified

Unit for replication and fault tolerance

Storing a File in HDFS



The blocks are of size 128 MB

Storing a File in HDFS

size 128 MB

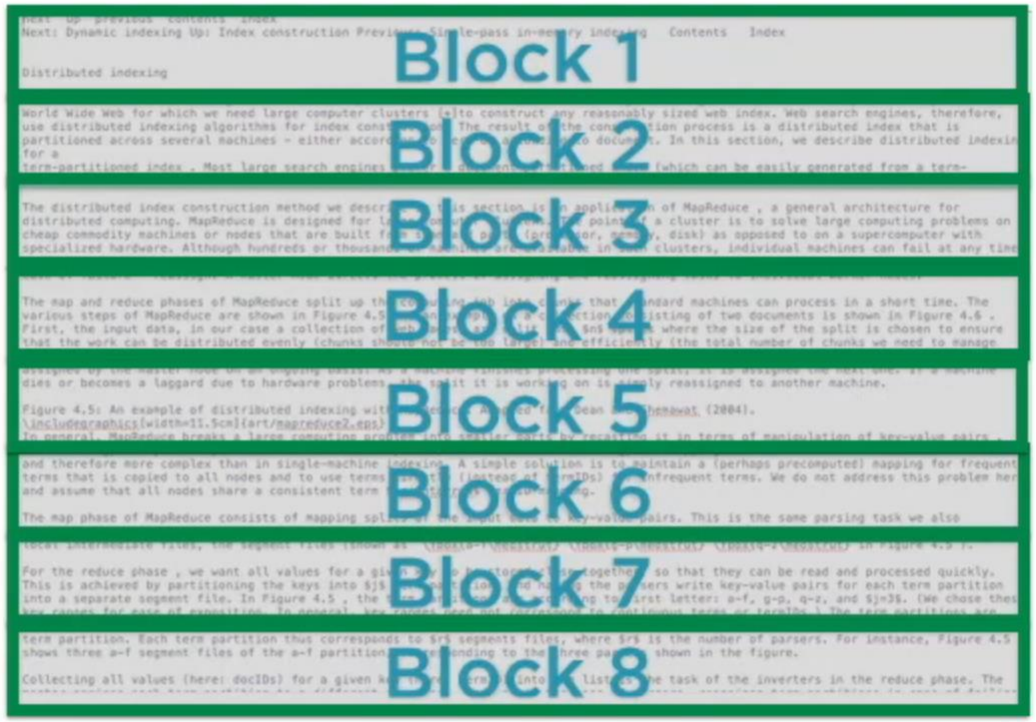
Block size is a trade off

Storing a File in HDFS

size 128 MB

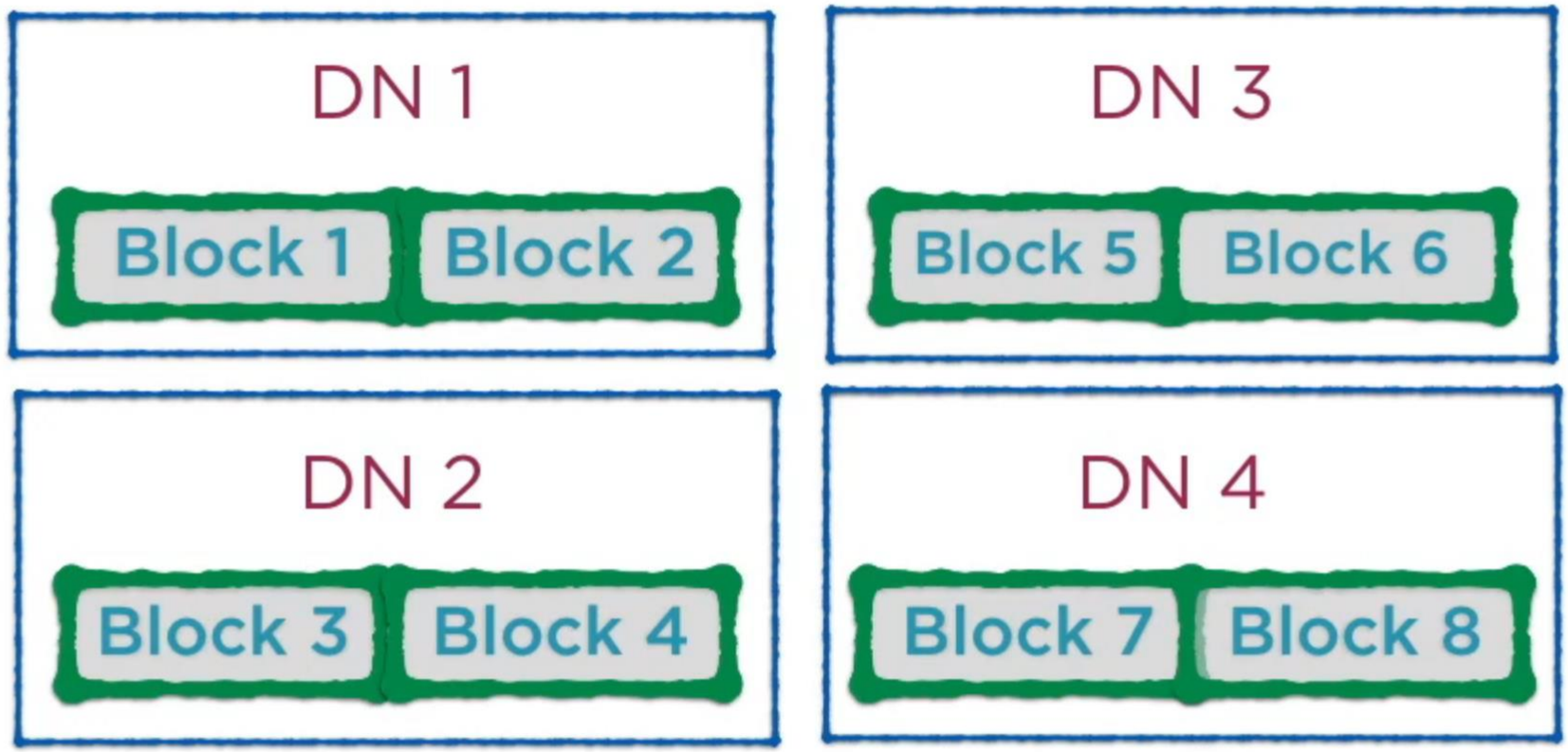
This size helps minimize the time taken to seek to the block on the disk

Storing a File in HDFS



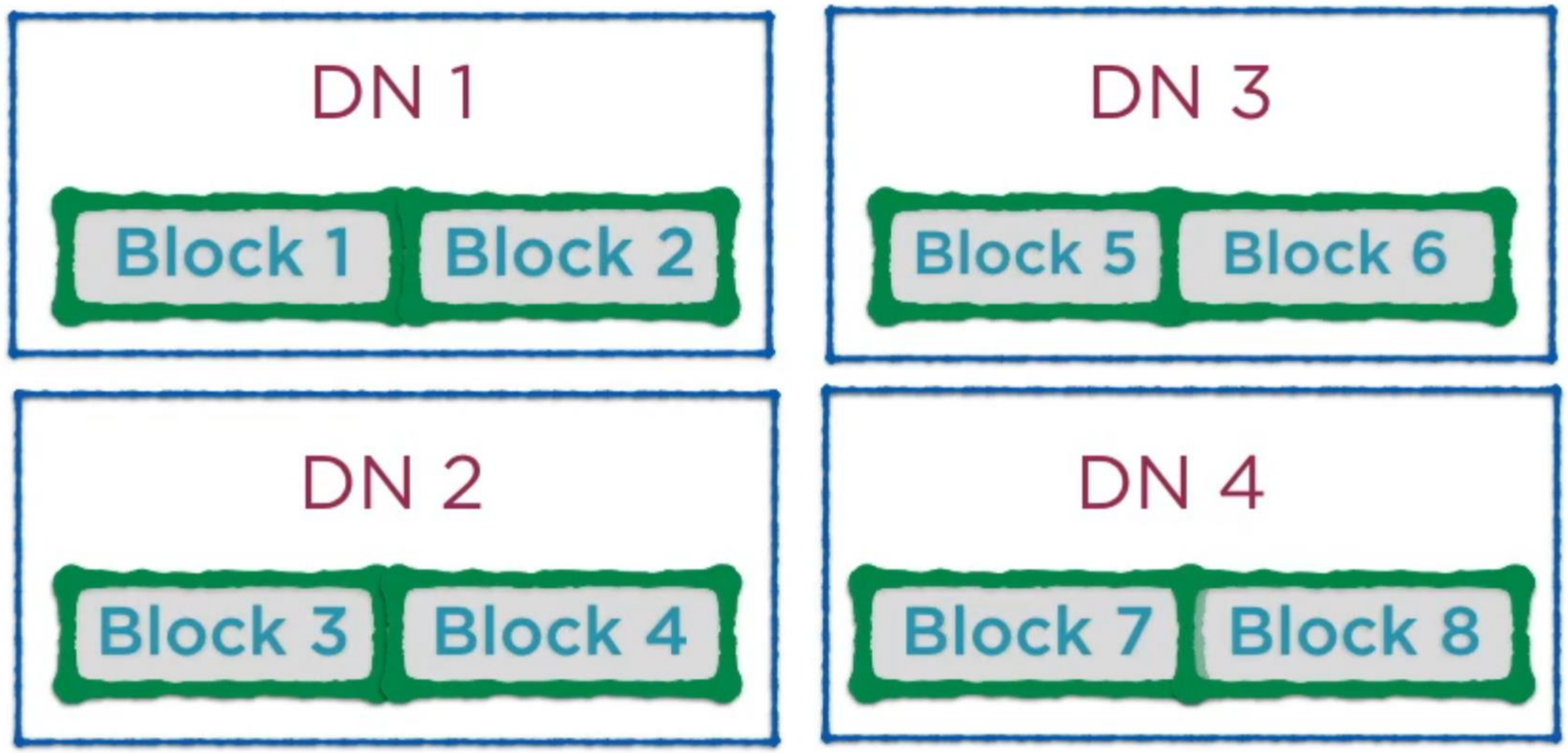
Store the blocks across the data nodes

Storing a File in HDFS



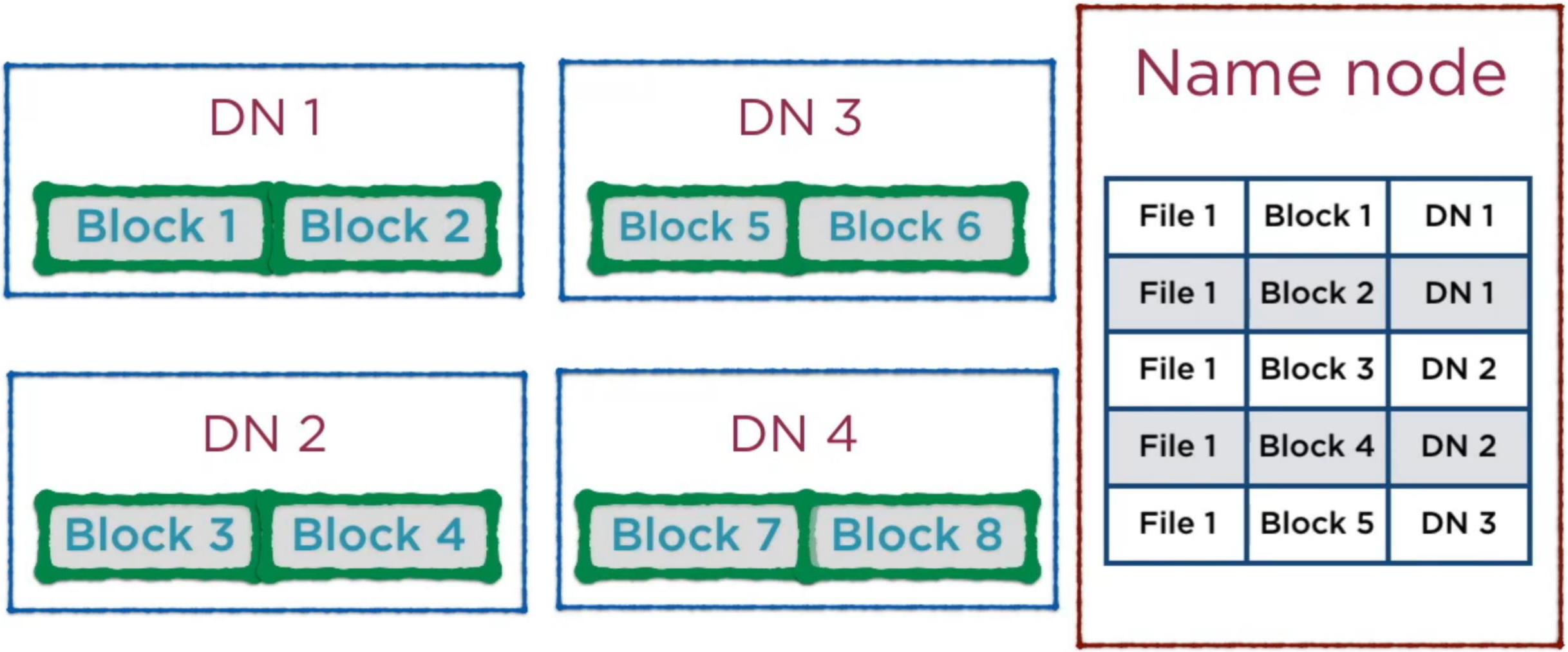
Each node contains a partition or a split of data

Storing a File in HDFS



How do we know where the splits of a particular file are?

Storing a File in HDFS

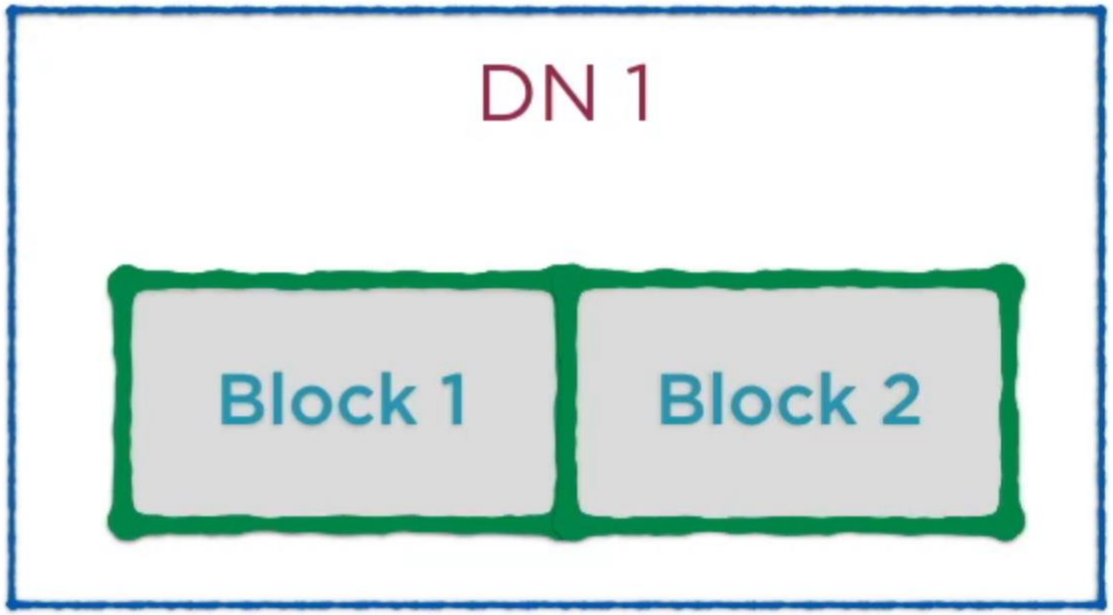


Reading a File in HDFS

1. Use metadata in the name node to look up block locations
2. Read the blocks from respective locations

Reading a File in HDFS

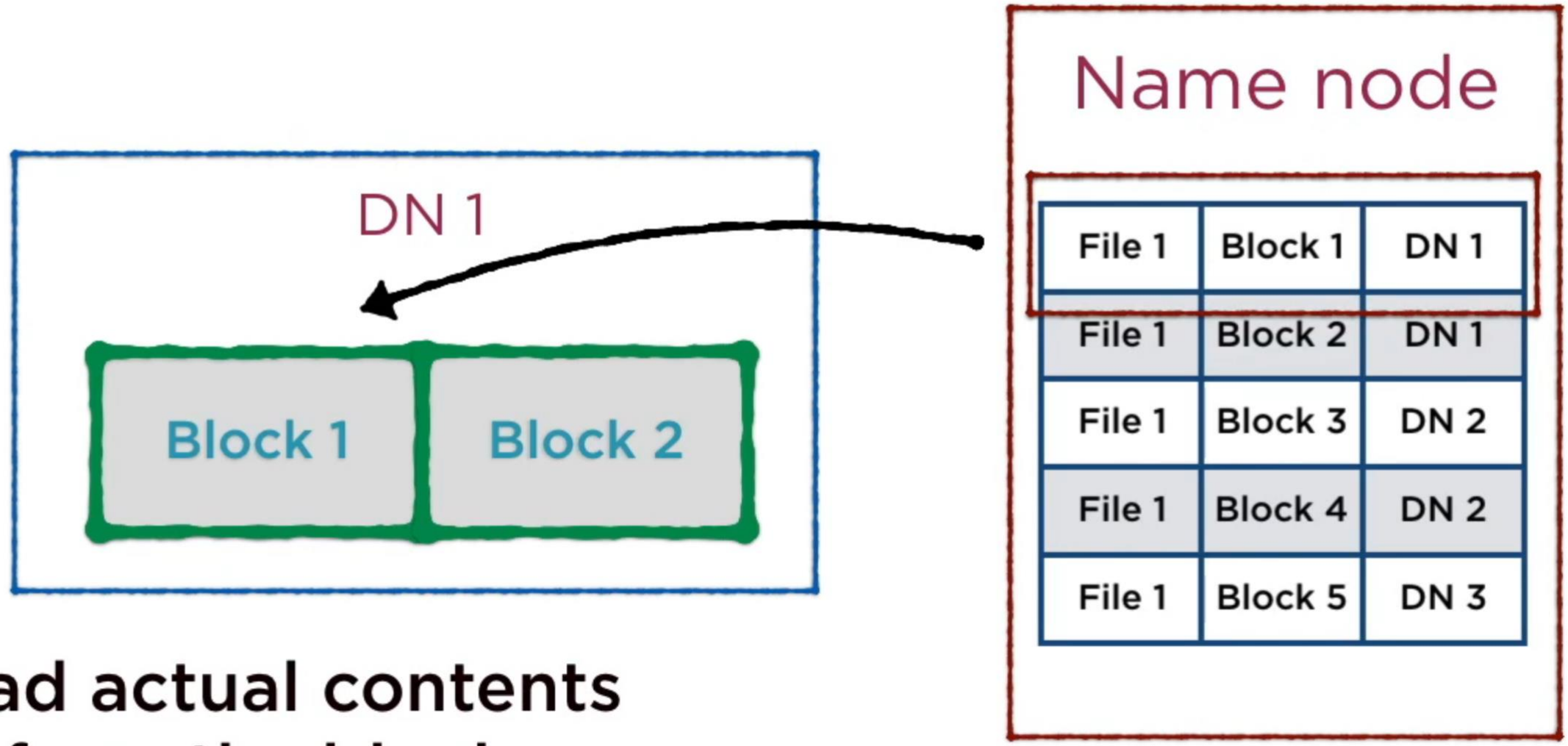
Request to the name node



Name node

File 1	Block 1	DN 1
File 1	Block 2	DN 1
File 1	Block 3	DN 2
File 1	Block 4	DN 2
File 1	Block 5	DN 3

Reading a File in HDFS

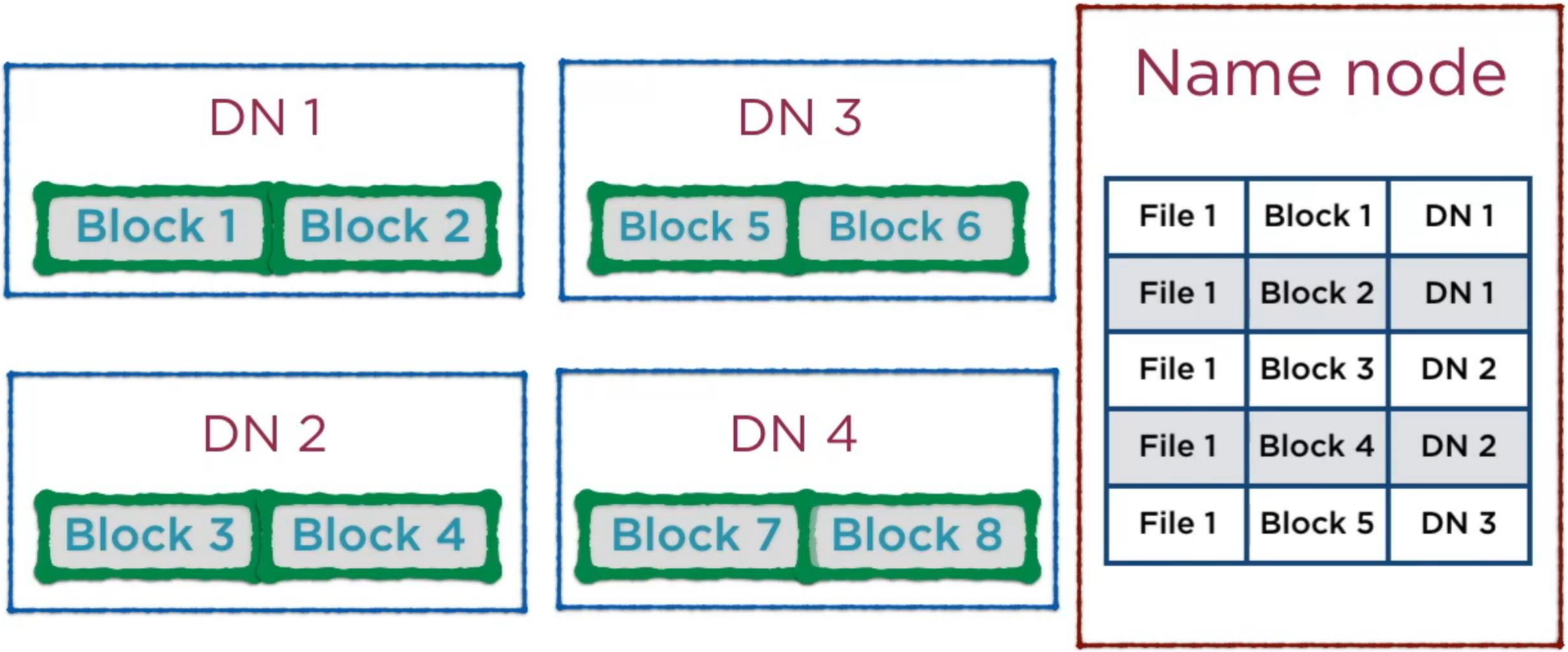


Read actual contents from the block

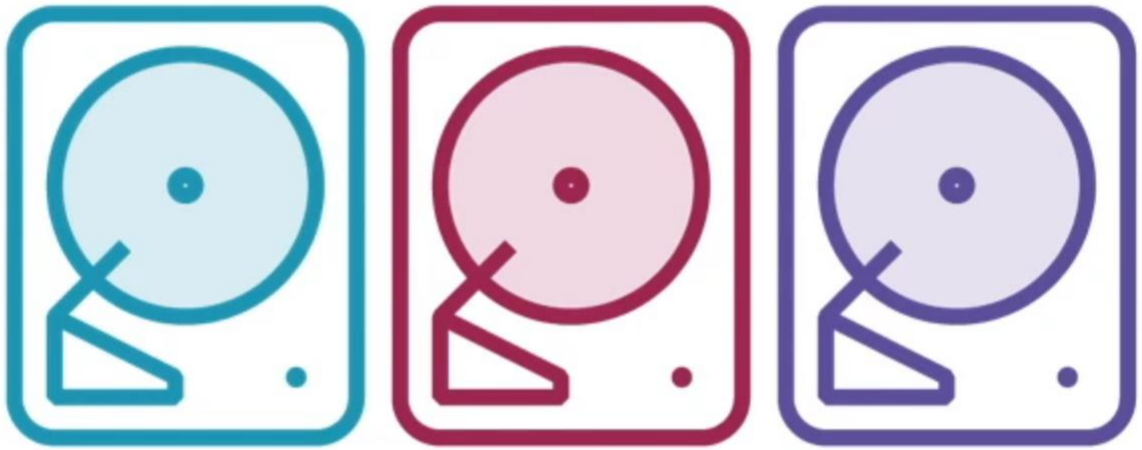
```
~/hadoop-install/hadoop-2.7.3/bin>
>ls -l
total 640
-rwxr-xr-x@ 1 jananiravi  staff  108813 Aug 18 07:19 container-executor
-rwxr-xr-x@ 1 jananiravi  staff    6488 Aug 18 07:19 hadoop
-rwxr-xr-x@ 1 jananiravi  staff    8514 Aug 18 07:19 hadoop.cmd
-rwxr-xr-x@ 1 jananiravi  staff   12223 Aug 18 07:19 hdfs
-rwxr-xr-x@ 1 jananiravi  staff    7238 Aug 18 07:19 hdfs.cmd
-rwxr-xr-x@ 1 jananiravi  staff    5953 Aug 18 07:19 mapred
-rwxr-xr-x@ 1 jananiravi  staff    6094 Aug 18 07:19 mapred.cmd
-rwxr-xr-x@ 1 jananiravi  staff    1776 Aug 18 07:19 rcc
-rwxr-xr-x@ 1 jananiravi  staff  125269 Aug 18 07:19 test-container-executor
-rwxr-xr-x@ 1 jananiravi  staff   13352 Aug 18 07:19 yarn
-rwxr-xr-x@ 1 jananiravi  staff   11054 Aug 18 07:19 yarn.cmd
~/hadoop-install/hadoop-2.7.3/bin>
>
```

A whole bunch of Hadoop related commands in this directory

Storing a File in HDFS



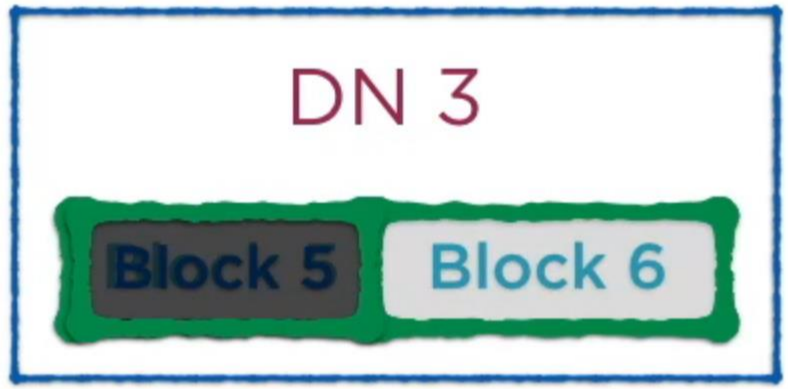
Challenges of Distributed Storage



Failure management in the data nodes

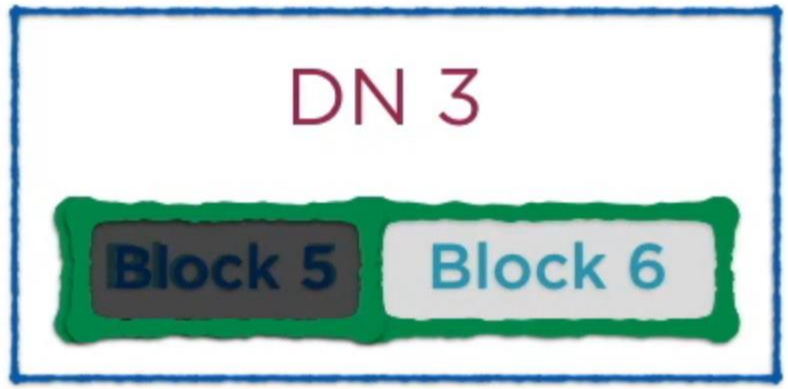
Failure management for the name node

A File Stored in HDFS



What if one of the blocks gets corrupted?

A File Stored in HDFS

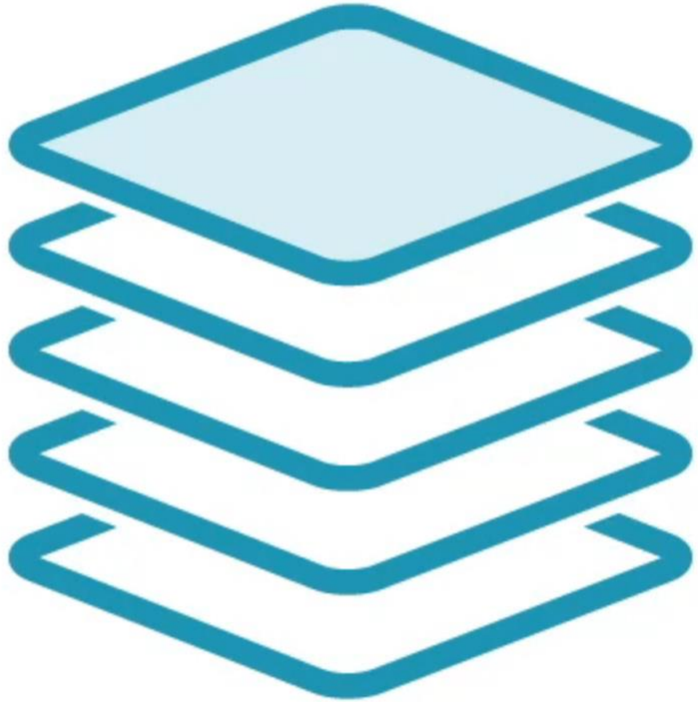


What if a data node containing some blocks crashes?

A File Stored in HDFS

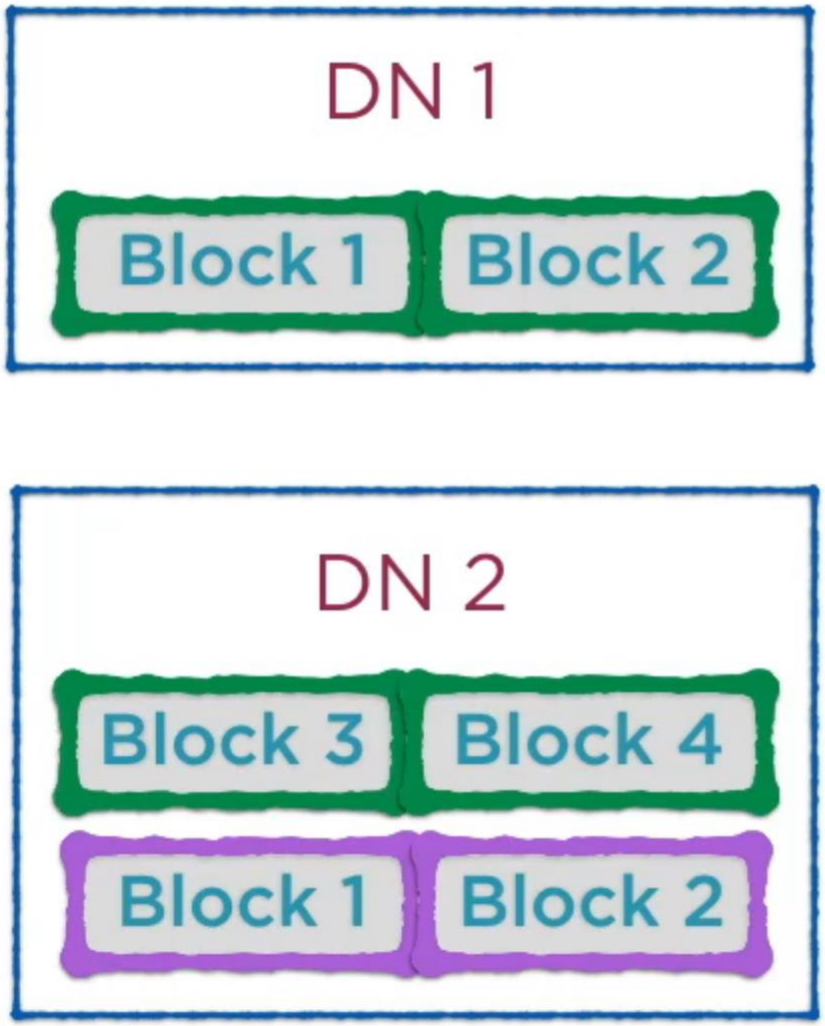
What if a data node containing some blocks crashes?

Managing Failures in Data Nodes



Define a replication factor

Replication



1. Replicate blocks based on the replication factor
2. Store replicas in different locations

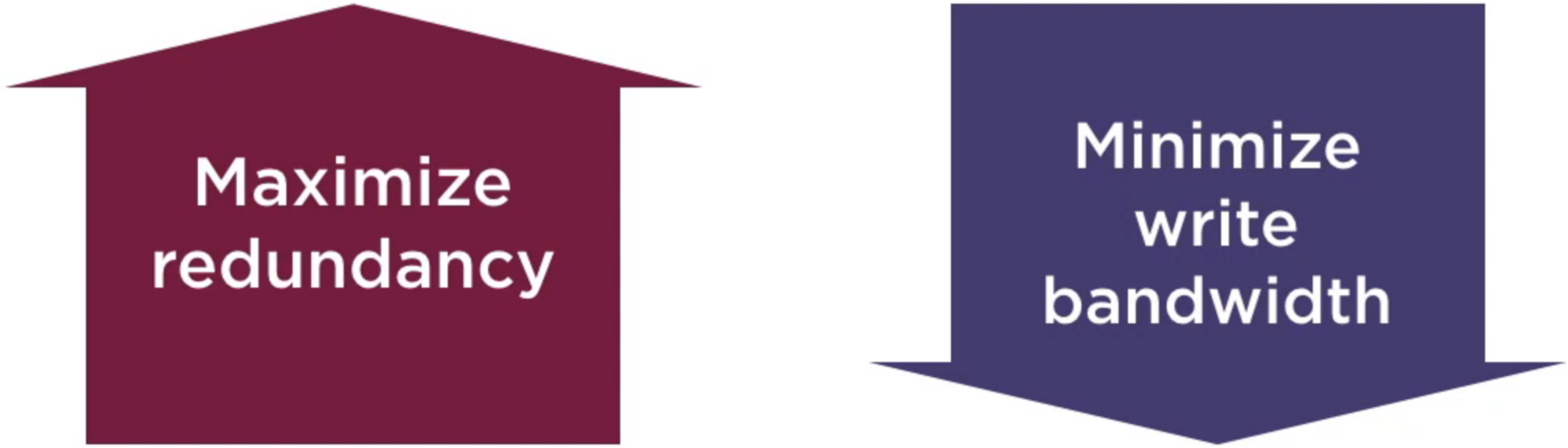
Replication

The replica locations are also stored in the name node

Name node

File 1	Block 1	DN 1
File 1	Block 2	DN 1
File 1	Block 3	DN 2
File 1	Block 4	DN 2
File 1	Block 5	DN 3
File 1	Block 1	DN 2
File 1	Block 2	DN 2

Choosing Replica Locations



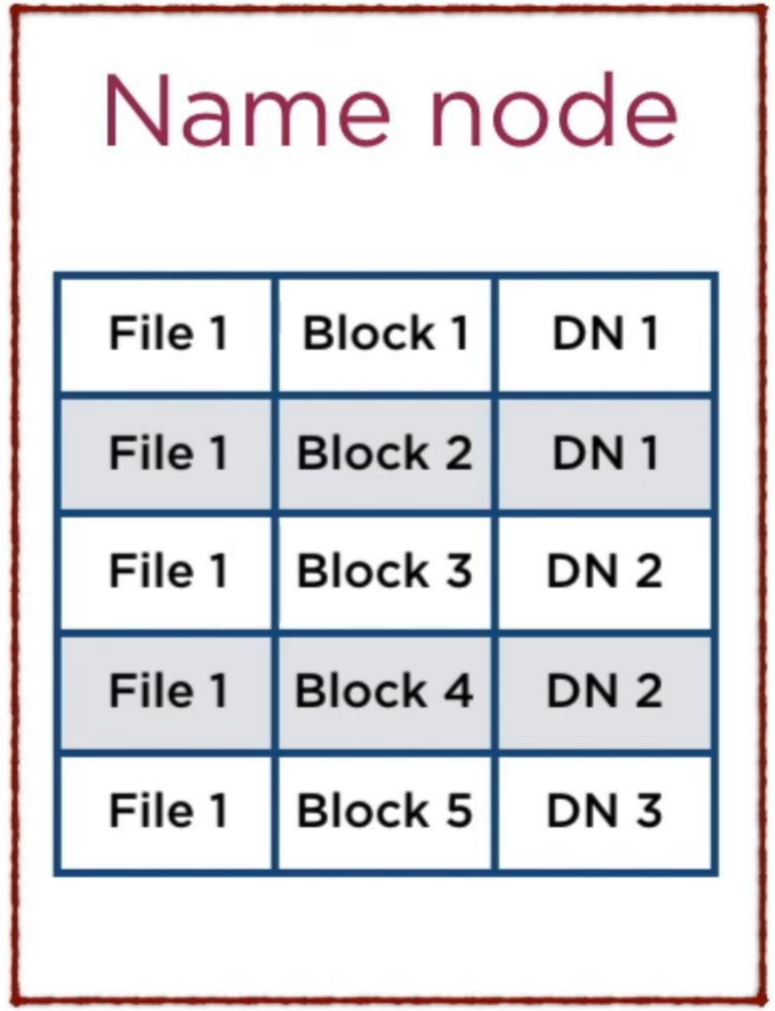
Setting the Replication Factor

dfs.replication

3

Name Node Failures

The name node is the heart of HDFS

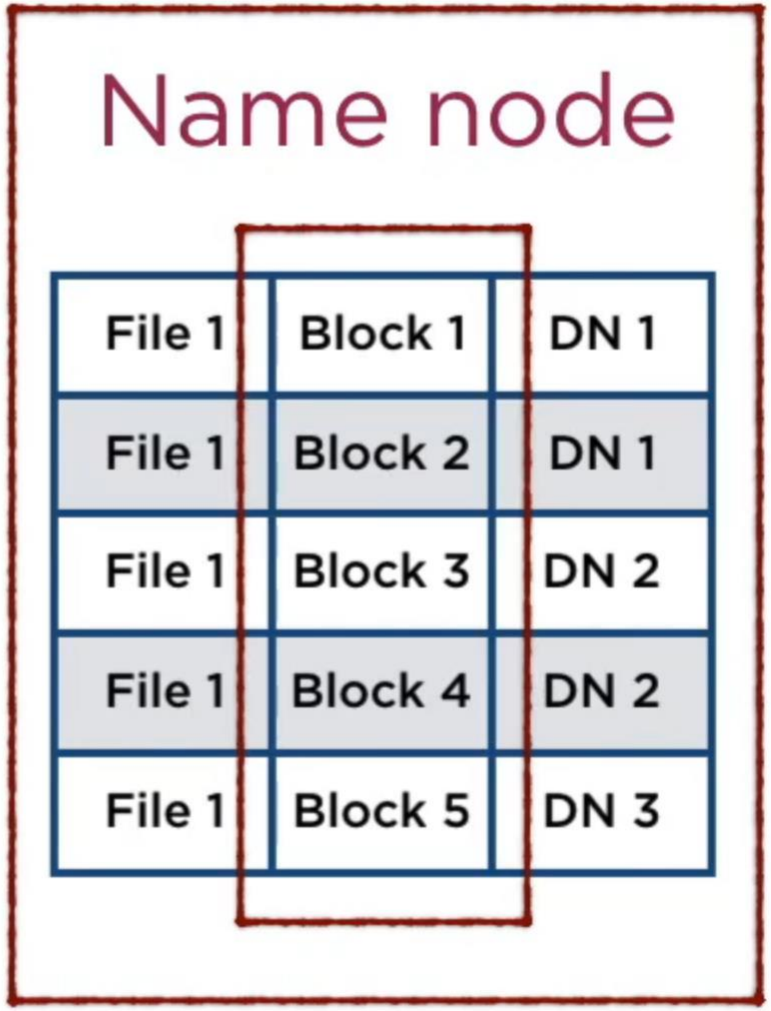


Name Node Failures

Block locations are not persistent

i.e. they are stored in memory

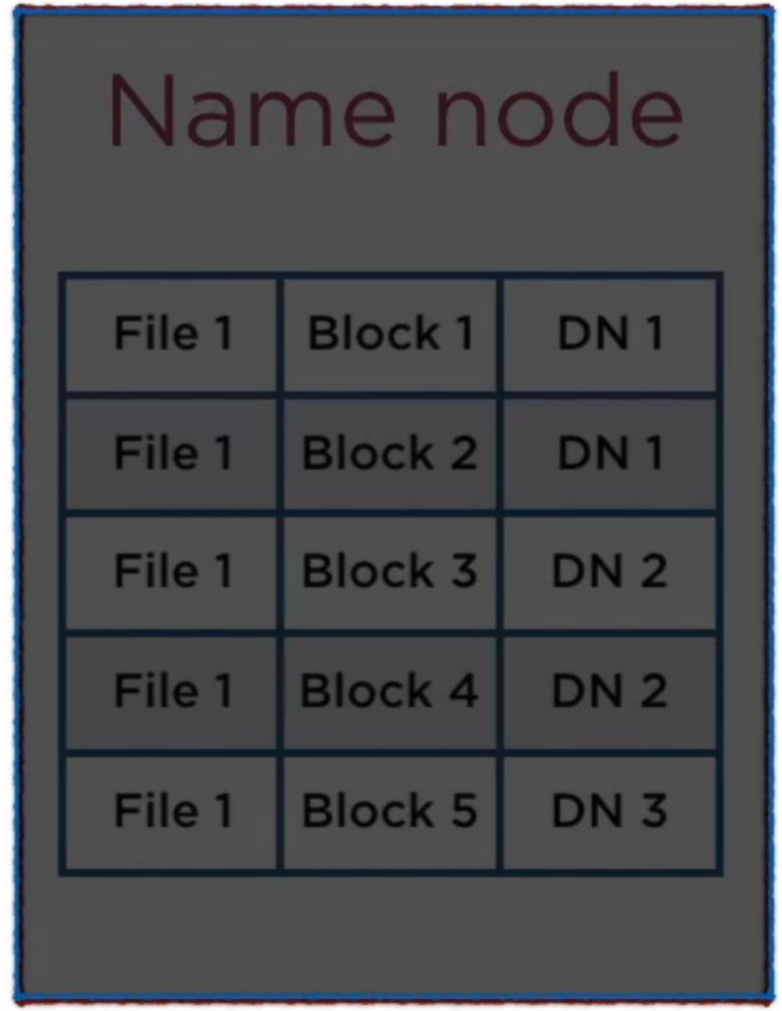
block caching



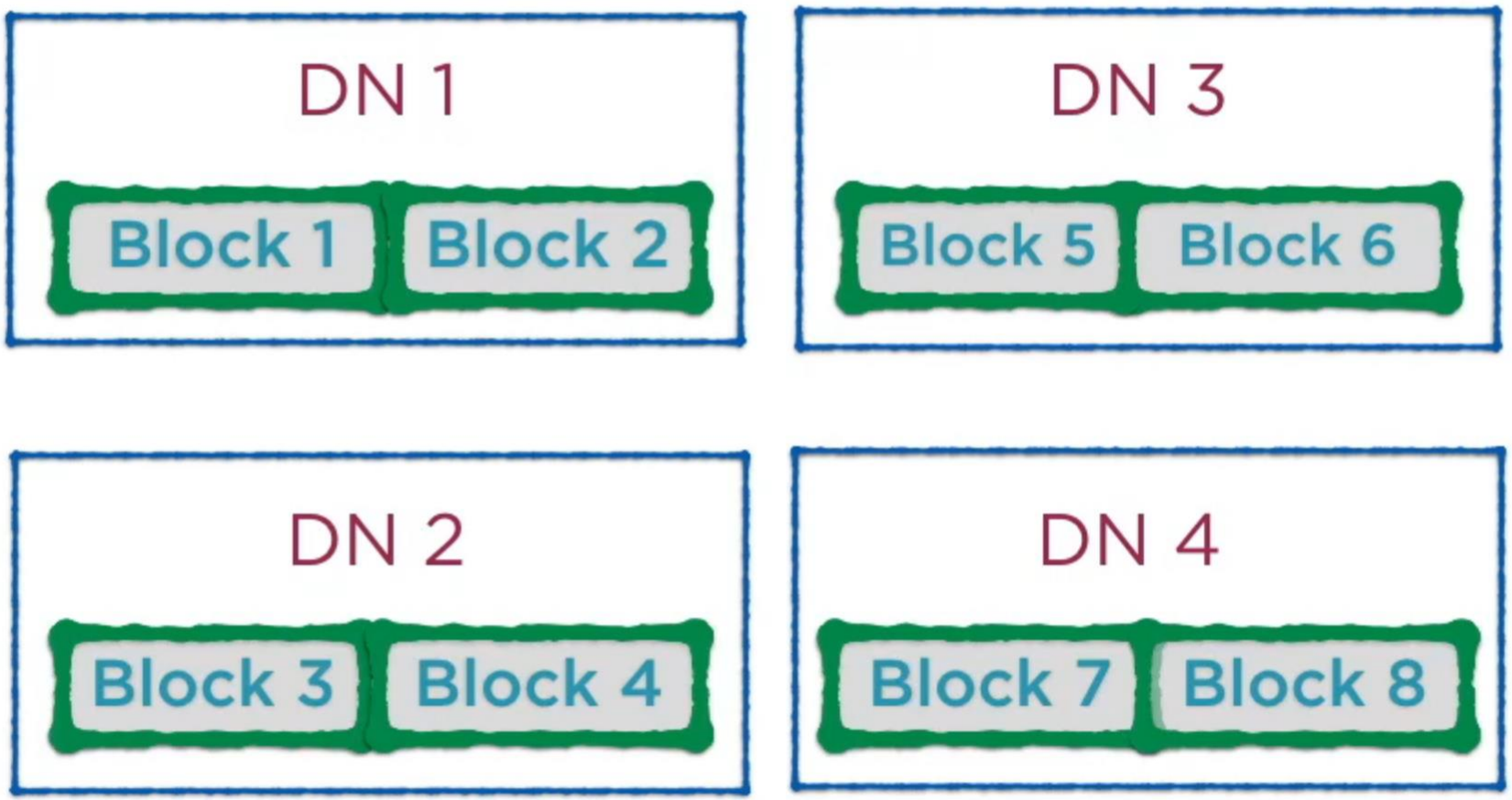
Name Node Failures

If the name node fails

File-Block Location mapping is lost!



Name Node Failures



This data is worthless without the name node

Managing Name Node Failures

Metadata Files

fsimage
edits

Two files that store the filesystem metadata

fsimage

A snapshot of the complete file system at start up

Loaded into memory

edits

A log of all in-memory edits to the file system

Metadata Files

`fsimage`

`edits`

Default backup location

Name node
local file system

Metadata Files

fsimage

edits

Alternative backup location

A remote drive

Configuring the Backup Location

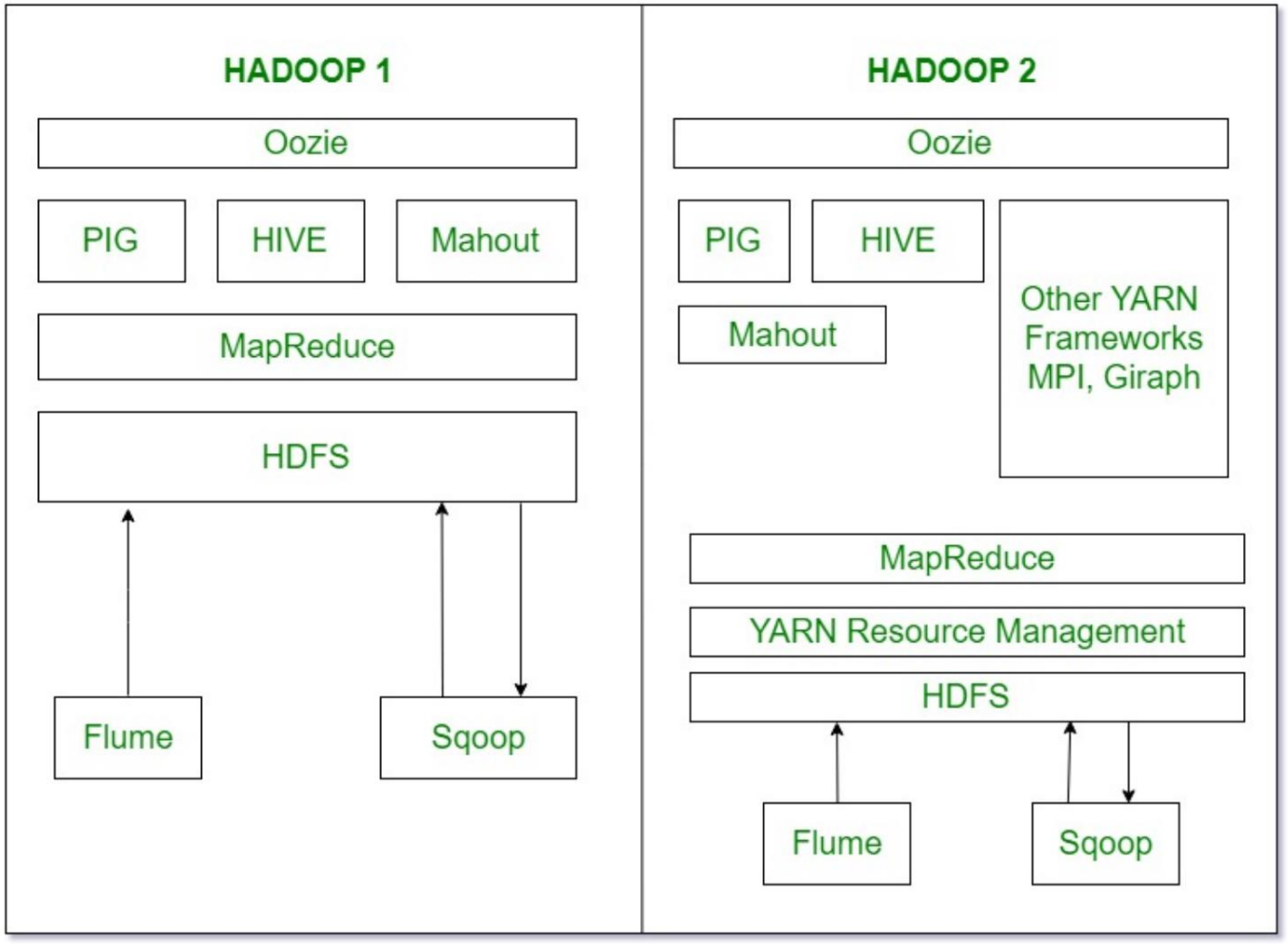
Set the property

`dfs.namenode.name.dir`

in hdfs-site.xml

Fault Tolerance

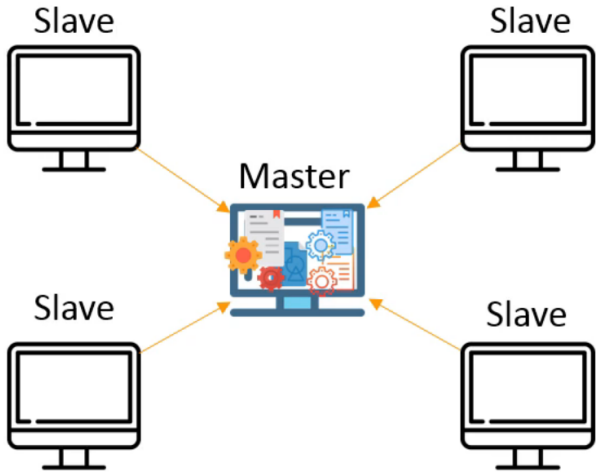
Ensures that when hardware fails, users will still have their data available. Fault tolerance is achieved through storing multiple copies throughout cluster.


HADOOP 1
Oozie
PIG
HIVE
Mahout
MapReduce
HDFS
Flume
Sqoop
HADOOP 2
Oozie
PIG
HIVE
Mahout
Other YARN
Frameworks
MPI, Giraph
MapReduce
YARN Resource Management
HDFS
Flume
Sqoop

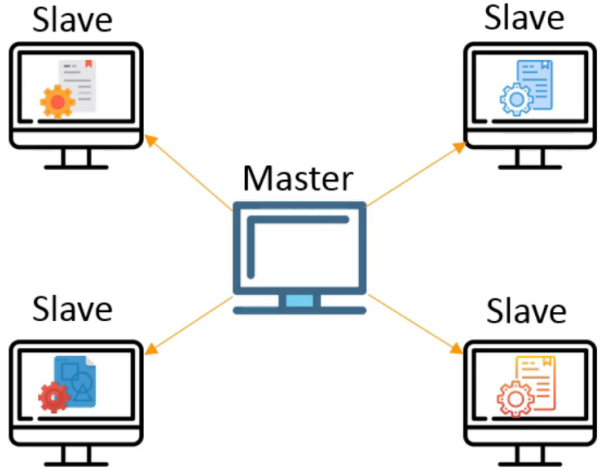
S.No.	Feature	Hadoop 2.x	Hadoop 3.x
1	License	Apache 2.0 is used for licensing which is open-source.	Apache 2.0 is used for licensing which is open-source.
2	Minimum supported Java version	JAVA 7 is the minimum compatible version.	JAVA 8 is the minimum compatible version.
3	Fault Tolerance	Replication is the only way to handle fault tolerance which is not space optimized.	Erasure coding is used for handling fault tolerance.
4	Data Balancing	HDFS balancer is used for Data Balancing.	Intra-data node balancer is used which is called via HDFS disk-balancer command-line interface.
5	Storage Scheme	3x Replication Scheme is used.	uses eraser encoding in HDFS.
6	Storage Overhead	200% of HDFS is consumed in Hadoop 2.x	50% used in Hadoop 3.x means we have more space to work.
7	YARN Timeline Service	Uses timeline service with scalability issue.	Improve the time line service along with improving scalability and reliability of this service.
8	Scalability	Limited Scalability, can have upto 10000 nodes in a cluster.	Scalability is improved, can have more then 10000 nodes in a cluster.
9	Default Port Range (32768-61000)	Linux ephemeral port range is used as default, which is failed to bind at startup time.	Ports used are out of this ephemeral port range.
10	Compatible File System.	HDFS(default), FTP, Amazon S3 and Windows Azure Storage Blobs (WASB) file system.	All file systems including Microsoft Azure Data Lake filesystem.
11	Name Node recovery	Manual intervention is needed for the namenode recovery.	No need of Manual intervention for name node recovery.

What is Map Reduce?

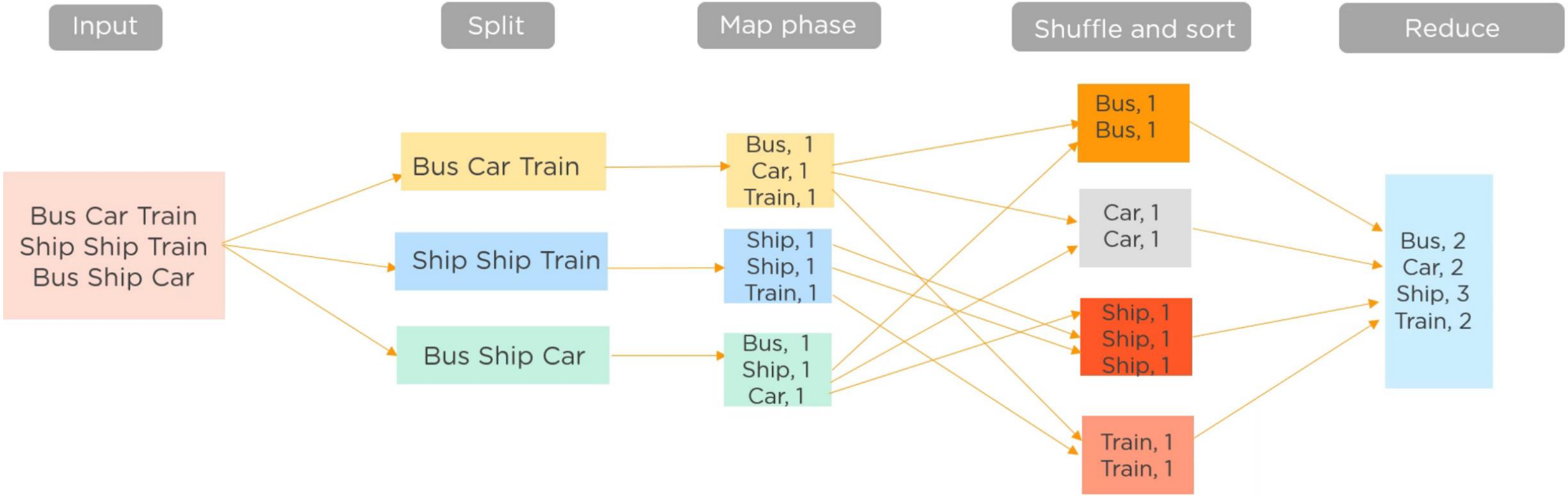
In MapReduce approach, processing is done at the slave nodes and the final result is sent to the master node



Traditional approach – Data is processed at the Master node



MapReduce approach – Data is processed at the Slave nodes


Input
Split
Map phase
Shuffle and sort
Reduce
Bus Car Train
Ship Ship Train
Bus Ship Car
Bus Car Train
Ship Ship Train
Bus Ship Car
Bus, 1
Car, 1
Train, 1
Ship, 1
Ship, 1
Train, 1
Bus, 1
Ship, 1
Car, 1
Bus, 1
Bus, 1
Car, 1
Car, 1
Ship, 1
Ship, 1
Ship, 1
Train, 1
Train, 1
Bus, 2
Car, 2
Ship, 3
Train, 2

Key Insight Behind MapReduce

Many data processing tasks can be expressed in this form

Counting Word Frequencies

Twinkle twinkle little star
How I wonder what you are
Up above the world so high
Like a diamond in the sky

For each word
in each line

Word	Count
twinkle	2
little	1
...	...
...	...
...	...
...	...

Implementing in Java

A class where the map logic is implemented

A class where the reduce logic is implemented

A driver program that sets up the job

Map Step

Map Class

<input key type,
input value type,
output key type,
output value type>

Mapper Class

This is a generic class, with 4 type parameters

Reduce Step

Reduce Class

<input key type,
input value type,
output key type,
output value type>

Reducer Class

This is also a generic class, with 4 type parameters

Matching Data Types

Map Class

output key type,
output value type>

Mapper Class

Reduce Class

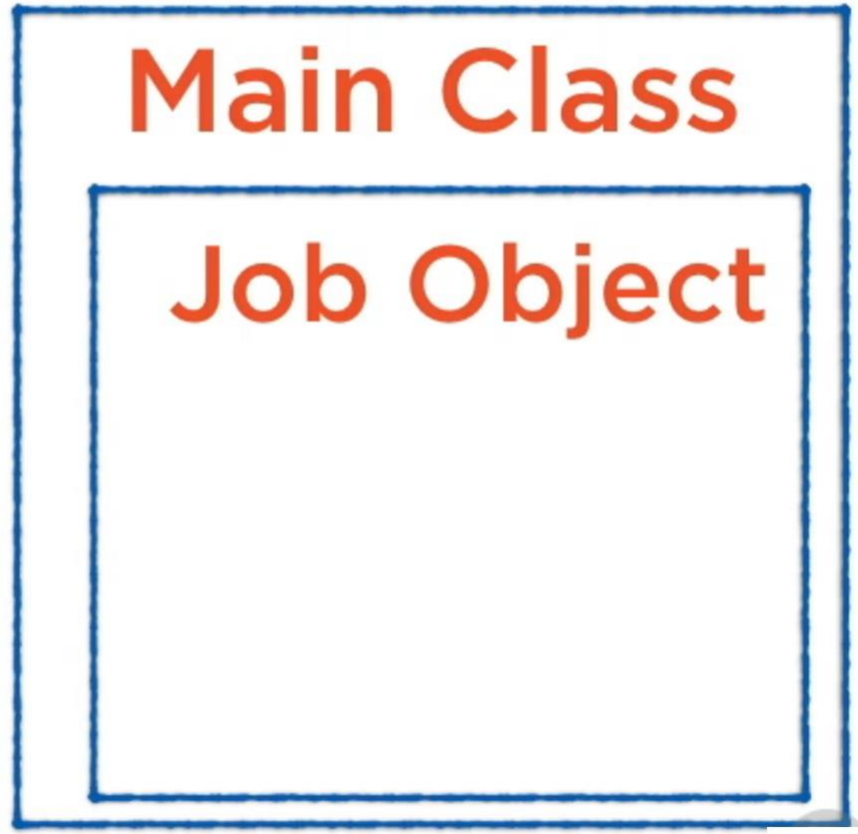
<input key type,
input value type,

Reducer Class

The output types of the Mapper should match the input types of the Reducer

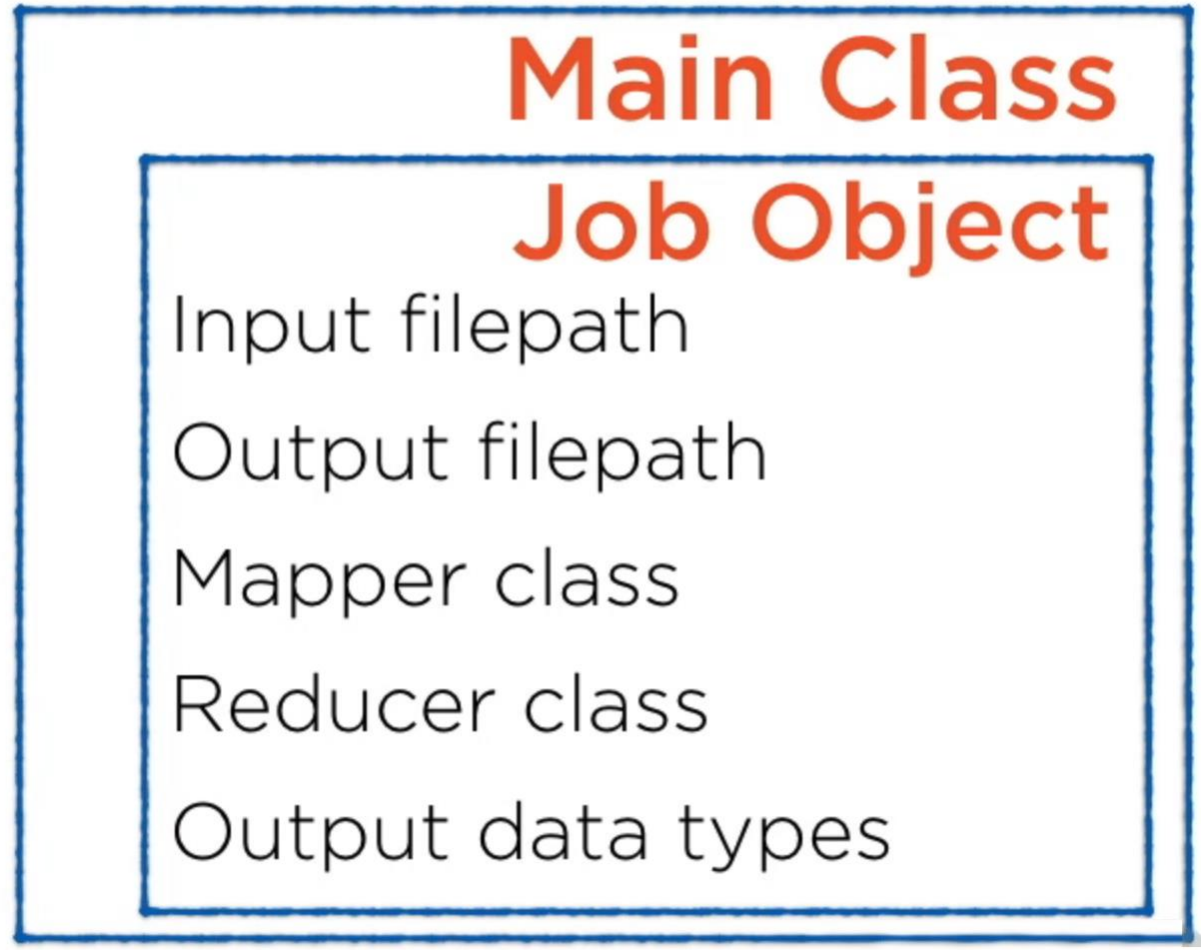
Setting up the Job

The Mapper and Reducer classes are used by a Job that is configured in the Main Class



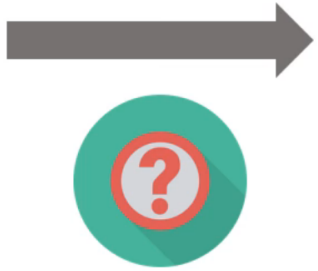
Setting up the Job

The Job has a bunch of properties that need to be configured

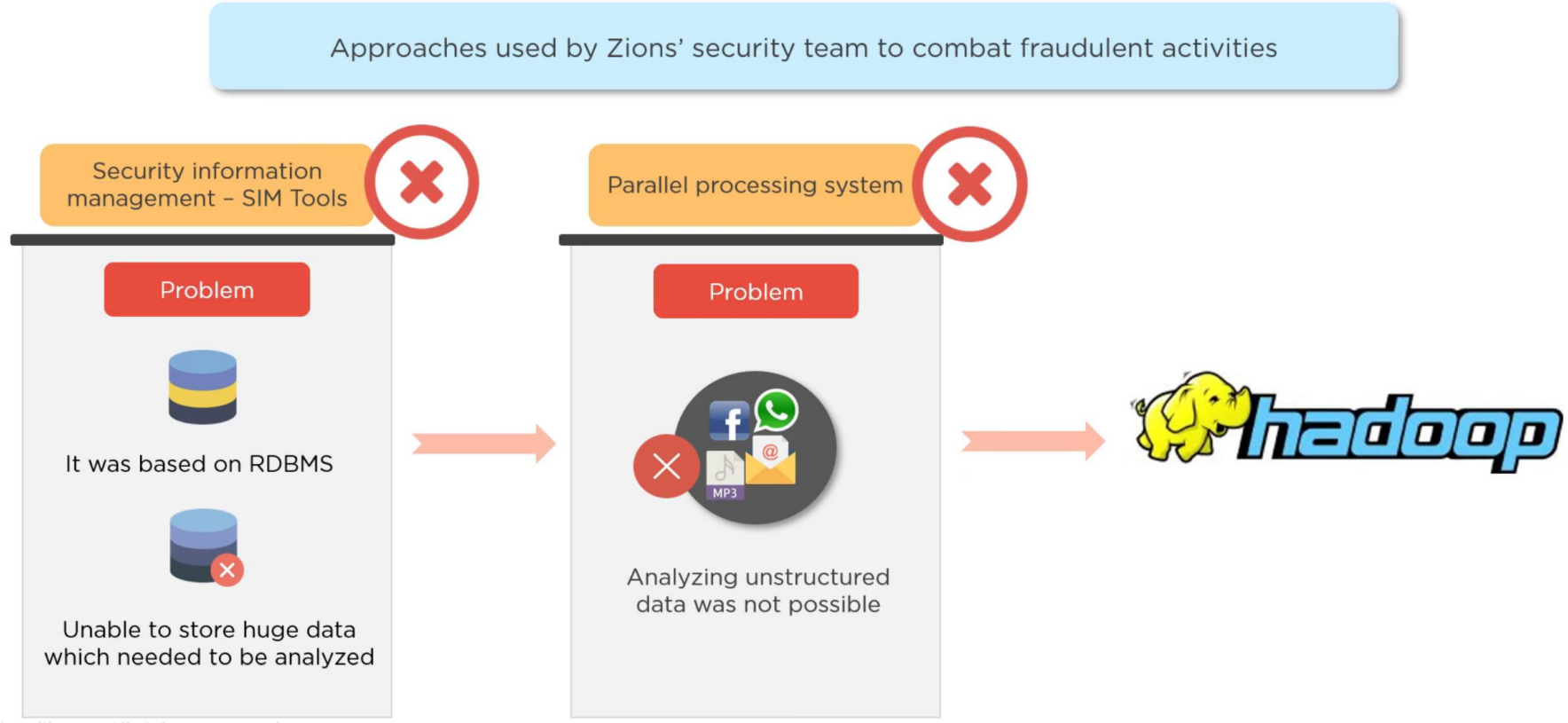


Hadoop Use Case – Combating fraudulent activities

Zions' main challenge was to combat the fraudulent activities which were taking place



Hadoop Use Case – Combating fraudulent activities



Hadoop Use Case – Combating fraudulent activities

How Hadoop solved the problems

Storing

Zions could now store massive amount of data using Hadoop

Processing

Processing of unstructured data (like server logs, customer data, customer transactions) was now possible

Analyzing

In-depth analysis of different data formats became easy and time efficient

Detecting

The team could now detect everything from malware, spear phishing attempts to account takeovers

References

https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

https://examples.javacodegeeks.com/enterprise-java/apache-hadoop/hadoop-hello-world-example/

https://learn.microsoft.com/en-us/azure/hdinsight/hadoop/apache-hadoop-develop-deploy-java-mapreduce-linux

https://www.h2kinfosys.com/blog/hadoop-mapreduce-examples/