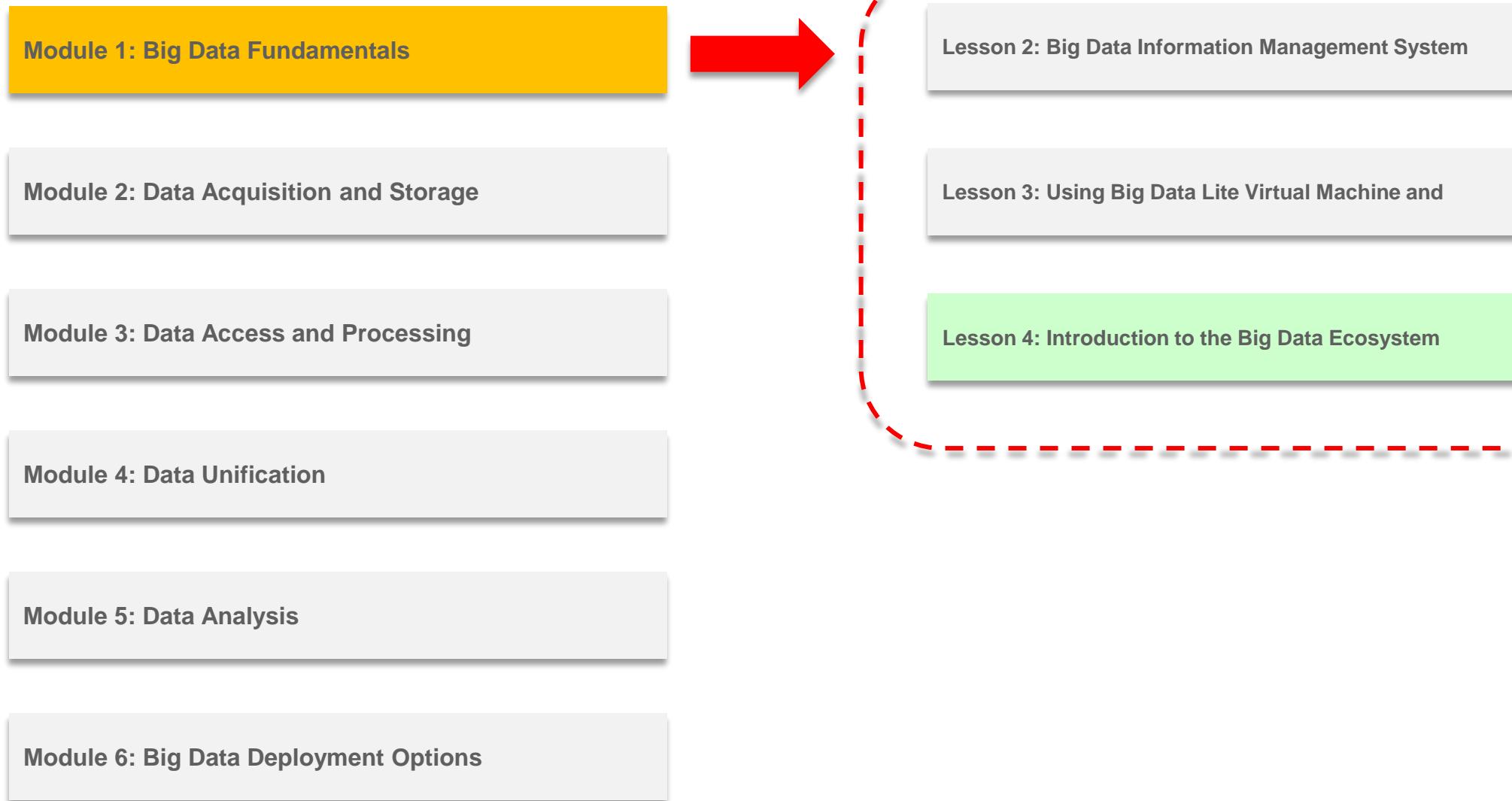


Introduction to the Big Data Ecosystem (and Data Science)

Course Road Map



Objectives

- After completing this lesson, you should be able to:
 - Define Hadoop and the *Hadoop Ecosystem*
 - List the Hadoop core components
 - Choose a Hadoop Distribution
 - List some of the other related projects in the Hadoop Ecosystem



Computer Clusters

- A computer rack (commonly called a rack) is a metal frame used to hold various hardware devices such as servers, hard disk drives, and other electronic equipment.
- A computer cluster is a single logical unit consisting of multiple computers (or racks) that are linked through a fast local area network (LAN).
- The components of a cluster, nodes (computers used as a servers), run their own instance of an operating system.
- A node typically includes CPU, memory, and disk(s) storage.



Distributed Computing

- Distributed computing is a technique that allows individual computers to be networked together.
- A distributed file system is a client/server application that allows clients to access and process data stored on the server as if it were stored on their own computer.
- File systems that manage the storage across a network of machines are called distributed file systems.



Apache Hadoop

- Apache Hadoop:
 - Is an open-source software framework for **Distributed Storage** and **Distributed Processing** of big data on clusters of commodity hardware
 - Is a batch and interactive data-processing system for enormous amounts of data
- Open source available:
 - From the Apache Hadoop Foundation
 - As distributions, such as:
 - Cloudera's Distribution Including Apache Hadoop (CDH)
 - Hortonworks (HDP)
 - MapR



Types of Analyses That Use Hadoop

- *Market analysis*
- *Product recommendations*
- Demand forecasting
- *Fraud detection*
- Text mining
- *Index building*
- Graph creation and analysis
- Pattern recognition
- Collaborative filtering
- Prediction models
- Sentiment analysis
- Risk assessment



Types of Data Generated

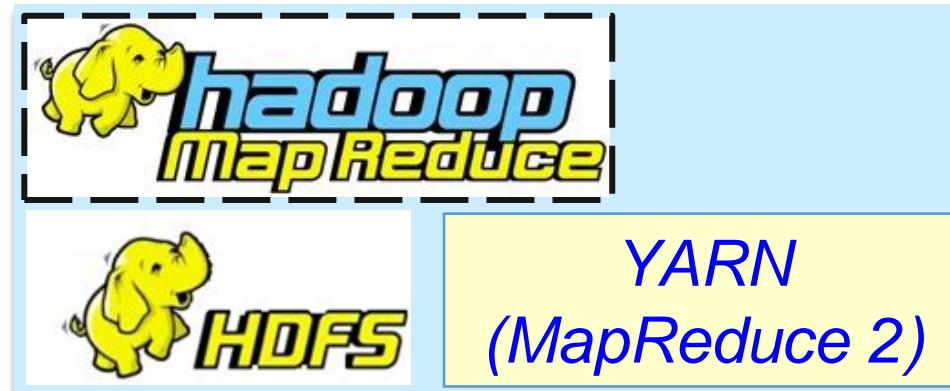
- Financial transactions
- Sensors data
- Server logs
- Analytics
- Email and text messages
- Social media



Apache Hadoop Core Components

A Hadoop cluster has:

- Distributed data using Apache Hadoop Distributed File System (HDFS)
- Distributed processing using one of the following:
 - Yet Another Resource Negotiator (YARN) MR2, an extensible framework job scheduling and cluster resource management
 - MapReduce Framework (MR1)



Apache Hadoop Core Components: HDFS

- Leader-follower architecture
- Based on Google's File System (GFS) paper
- Stores and distributes data across the nodes in the cluster as data is loaded
- Redundant (reliability)
- Fault tolerant (high availability)
- Scalable (out instead of up)



Apache Hadoop Core Components: MapReduce Framework (MRv1)

- Is a programming model or framework for distributed computing
- Schedules and monitors tasks, and re-executes failed tasks
- Uses Leader-follower architecture
- Integrates with HDFS to provide the exact same benefits for distributed parallel data processing on the cluster
- ***Sends computations where the data is stored on local disks (data locality)***
- Hides complex "housekeeping" and distributed computing complexity tasks from the developer
- Supports only MapReduce applications

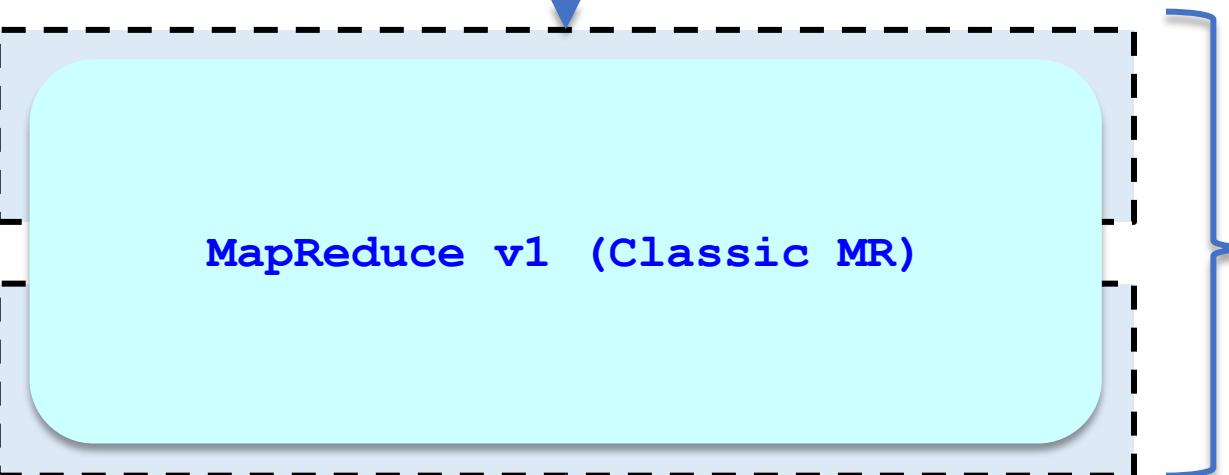


Running Applications Before Hadoop 2.x with MapReduce 1 (MR 1)

MapReduce Applications



Processing framework



Resource management

- Hadoop Cluster Resource Management and
- **Batch** Data Processing
- A single processing engine

Distributed storage

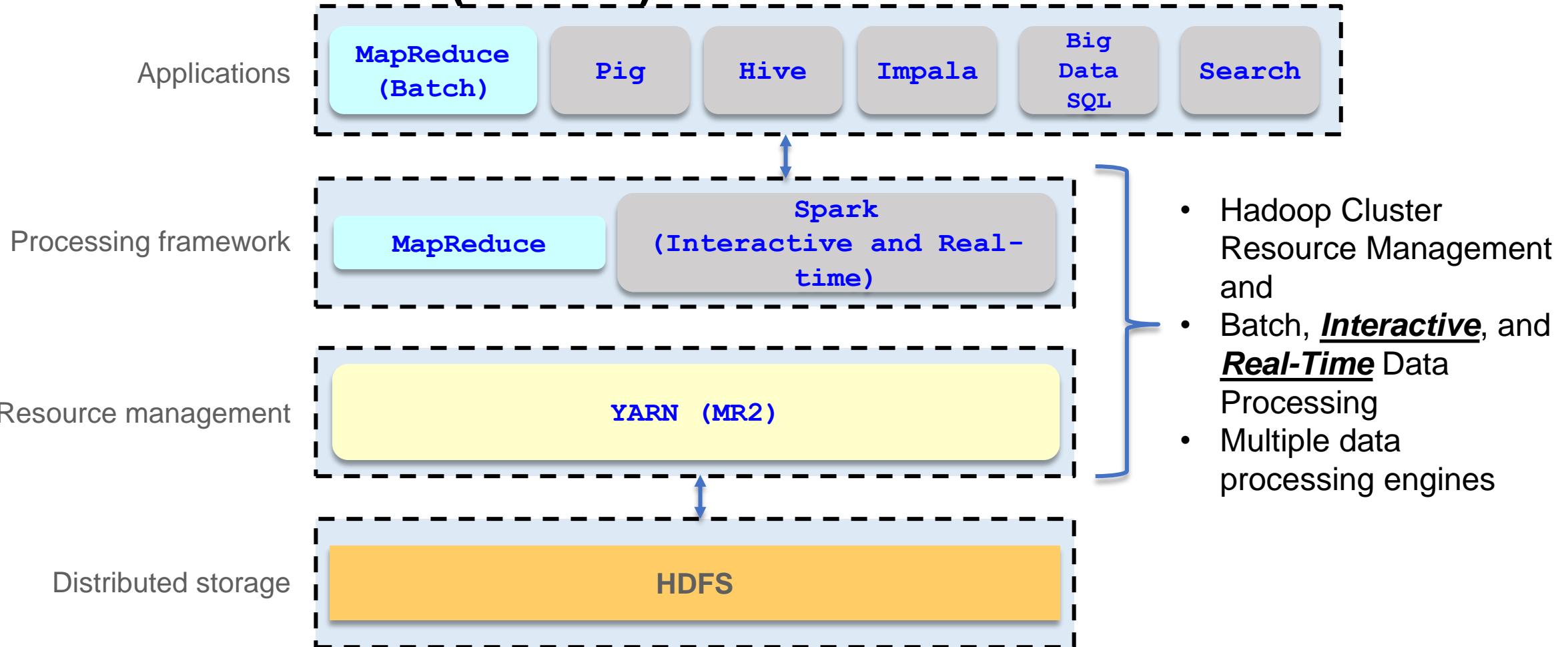


Apache Hadoop Core Components: YARN (MR2)

- Is a subproject of Hadoop that separates resource management and processing components
- Is a resource-management framework for Hadoop that is independent of execution engines
- Provides a more efficient and flexible workload scheduling as well as a resource management facility, both of which ultimately enable Hadoop to run more than just MapReduce jobs such as Impala, Spark, and so on

YARN

Running Applications Starting with Hadoop 2.x With YARN (MR 2)



Apache Hadoop Ecosystem



- Hadoop Core Components:**
- HDFS (Storage)
 - YARN (Processing and Resource Management, MR2)
 - MapReduce (Distributed processing, MR1)



Hadoop Ecosystem:
A partial list of
related projects (extend core
Hadoop or make it easier to
use)

Additional Resources: Cloudera Distribution

Cloudera Enterprise

CDH, Cloudera Manager, Search, Cloudera Navigator, Impala, and Spark documentation for 5.2 and higher releases.

VERSION

Cloudera Director

Instruction on installation, configuration, and use of Cloudera Director.

VERSION

Apache Kafka

Instruction on the installation, administration, and use of Cloudera distribution of Apache Kafka.

VERSION

Previous 5.1.x and 5.0.x Versions



CDH

These guides describe how to install, configure, and secure CDH.

VERSION

CDH 5.1.x Documentation 



Cloudera Manager

These guides describe how to install, configure, secure, and use Cloudera Manager.

VERSION

http://www.cloudera.com/documentation/cdh/5-1-x.html



CDH

These guides describe how to install, configure, and secure CDH.

VERSION

CDH 5.1.x Documentation 

Community Cloud

- <https://community.cloud.databricks.com/>



Additional Resources: Apache Hadoop

<http://hadoop.apache.org/>

The image shows two screenshots of the Apache Hadoop website. The left screenshot displays the main homepage with sections for 'About', 'Documentation' (which is highlighted with a red box), and 'Related Projects'. A blue arrow points from the 'Documentation' link to the right screenshot, which shows a detailed view of the 'Apache Hadoop 2.7.2' release notes. The right screenshot highlights the 'General' and 'Common' sections under the 'Overview' heading.

Welcome to Apache™ Hadoop

What Is Apache Hadoop?

The Apache™ Hadoop® project develops open-

The Apache Hadoop software library is a framework of computers using simple programming model offering local computation and storage. Rather detect and handle failures at the application layer of which may be prone to failures.

The project includes these modules:

- **Hadoop Common:** The common utilities to support the system.
- **Hadoop Distributed File System (HDFS):** A distributed file system designed for storing large amounts of application data.
- **Hadoop YARN:** A framework for job scheduling and resource management.
- **Hadoop MapReduce:** A YARN-based system for distributed processing of large data sets.

General

- Overview
- Single Node Setup
- Cluster Setup
- Hadoop Commands Reference
- FileSystem Shell
- Hadoop Compatibility
- Interface Classification
- FileSystem Specification

Common

- CLI Mini Cluster
- Native Libraries
- Proxy User
- Rack Awareness
- Secure Mode
- Service Level
- Authorization
- HTTP Authentication
- Hadoop KMS
- Tracing

HDFS

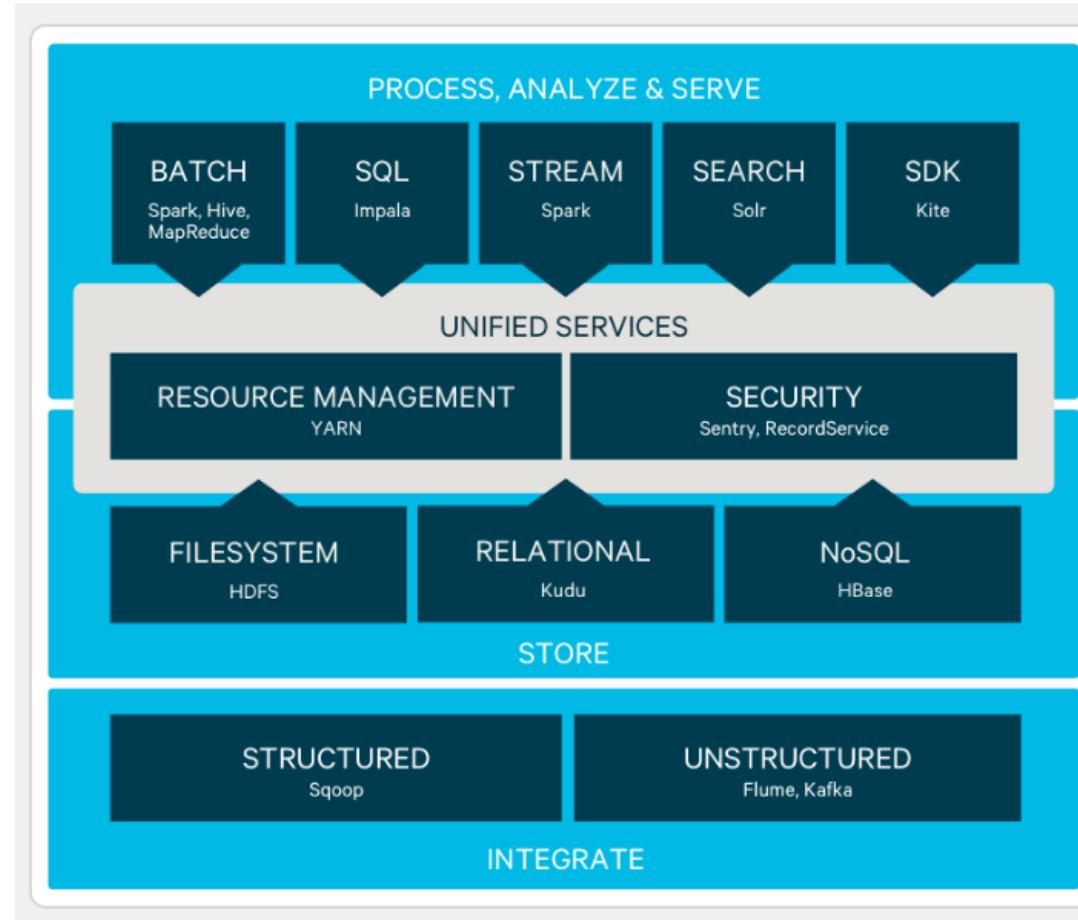
- HDFS User Guide
- HDFS Commands Reference
- High Availability With QJM

Apache Hadoop 2.7.2

Apache Hadoop 2.7.2 is a minor release in the 2.x.y release line, building upon the previous stable version 2.7.1. Here is a short overview of the major features and improvements.

- Common
 - Authentication improvements when using an HTTP proxy server. This is useful when a proxy server is between the client and the Hadoop cluster.
 - A new Hadoop metrics sink that allows writing directly to Graphite.
 - Specification work related to the Hadoop Compatible Filesystem (HCFS) effort.
- HDFS
 - Support for POSIX-style filesystem extended attributes. See the [user documentation for more details](#).
 - Using the OfflineImageViewer, clients can now browse an fsimage via the WebHDFS API.
 - The NFS gateway received a number of supportability improvements and bug fixes. The configuration required to run the gateway, and the gateway is now able to reject connections from unauthorized hosts.
 - The SecondaryNameNode, JournalNode, and DataNode web UIs have been modernized.
- YARN
 - YARN's REST APIs now support write/modify operations. Users can submit and kill applications via the REST API.
 - The timeline store in YARN, used for storing generic and application-specific information.

Overall Big Data Reference Architecture



Source: <http://www.cloudera.com/products/apache-hadoop.html>

CDH Components

Component	Description
Apache Hadoop	<ul style="list-style-type: none">• A framework for executing applications on a large cluster of servers. It is built for massively parallel processing across a large number of nodes (servers).• Consists of the following core components: Hadoop Distributed File System (HDFS) and MapReduce
Hue (Hadoop User Experience)	<ul style="list-style-type: none">• Is an open-source tool• Easy to use web front end for viewing files, running queries, performing searches, scheduling jobs, and more• Contains several applications to access a Hadoop cluster through a web front end
Apache Oozie	<ul style="list-style-type: none">• Enables developers to create, edit, and submit workflows by using the Oozie dashboard• After considering the dependencies between jobs, the Oozie server submits those jobs to the server in the proper sequence.
Apache Spark	<ul style="list-style-type: none">• It is an open source parallel data processing framework.• It complements Apache Hadoop.• It makes it easy to develop fast, unified Big Data applications combining batch, streaming, and interactive analytics on all your data.

CDH Architecture

Component	Description
Apache Solr (Cloudera Search)	<ul style="list-style-type: none">Cloudera Search is one of Cloudera's near-real-time access products and is powered by Solr. It enables nontechnical users to search and explore data stored in or ingested into Hadoop, fenago NoSQL Database, and HBase.Users do not need SQL or programming skills to use Cloudera Search because it provides a simple, full-text interface for searching.
Apache Hive	<ul style="list-style-type: none">Hive Metastore provides a metadata layer that describes the data stored in HDFS.It provides a SQL layer to data on HDFS. It can run SQL queries on HDFS data.It uses Map/Reduce for execution and HDFS for storage.
Apache Pig	<ul style="list-style-type: none">It is an analysis platform that provides a data flow language called Pig Latin.It is an alternative abstraction on top of MapReduce.
Cloudera Impala	<ul style="list-style-type: none">The Impala server is a distributed, massively parallel processing (MPP) database engine.It consists of different daemon processes that run on specific hosts within your CDH cluster.The core Impala component is a daemon process that runs on each node of the cluster.

CDH Components

Component	Description
Apache Flume	<ul style="list-style-type: none">• A distributed, reliable, available service for efficiently moving large amounts of data as it is generated• Ideal for collecting logs from diverse systems and inserting them in HDFS
Apache Sqoop	<ul style="list-style-type: none">• Imports tables from an RDBMS into HDFS• Imports data from RDBMS into HDFS as delimited text files or sequence files• Generates a class file that can encapsulate a row of the imported data
Apache Hbase	<ul style="list-style-type: none">• Is a NoSQL data store• Provides scalable inserts, efficient handling of sparse data, and a constrained data access model
Apache ZooKeeper	<ul style="list-style-type: none">• ZooKeeper is a centralized service for maintaining configuration information, naming, distributed synchronization, and group services.• HBase cannot be active without ZooKeeper.
Apache Mahout	<ul style="list-style-type: none">• Scalable machine-learning and data-mining algorithms
Apache Whirr	<ul style="list-style-type: none">• Apache Whirr is a set of libraries for running cloud services. It provides:<ul style="list-style-type: none">– A cloud-neutral way to run services– A common service API Smart defaults for services

Hadoop Major Timelines at a Glance



Doug Cutting starts the Nutch project (Open source web crawler and indexer)



Doug Cutting joins Yahoo!. Hadoop is released.



2002

2003

2004

2006

2007

2008

2009

2010

2013-2014

2015



Google publishes Google File System (GFS) paper.

- Google publishes the MapReduce paper in 2004.
- Doug Cutting adds GFS and MapReduce support to Nutch in 2004/2005.



Yahoo! wins Terasort competition: Fastest sort of 1 TB; 209 seconds over 910 nodes.



2009

2010



- Cloudera releases CDH.
- Amplab starts Mesos/Spark project.



Spark becomes an open source project.



Spark wins 100TB sort.



Spark becomes an Apache incubator, and then Apache top-level project and full integration with YARN.

Where to Go for More Information

Component	Website
Cloudera Manager	http://www.cloudera.com/content/cloudera/en/products-and-services/cloudera-enterprise/cloudera-manager.html
Apache Hadoop	http://hadoop.apache.org/
Apache Hadoop – Cloudera	http://www.cloudera.com/products/apache-hadoop.html
fuse-dfs	http://fuse.sourceforge.net/
Cloudera Hue	http://www.cloudera.com/content/cloudera-content/cloudera-docs/CDH4/4.2.0/Hue-2-User-Guide/hue2.html
Apache Oozie	http://oozie.apache.org/
Apache Hive	https://hive.apache.org/
Apache Pig	http://pig.apache.org
Apache Flume	http://flume.apache.org/
Apache Sqoop	http://sqoop.apache.org/
Apache HBase	http://hbase.apache.org/
Apache ZooKeeper	http://zookeeper.apache.org
Apache Mahout	http://mahout.apache.org
Apache Whirr	https://whirr.apache.org/

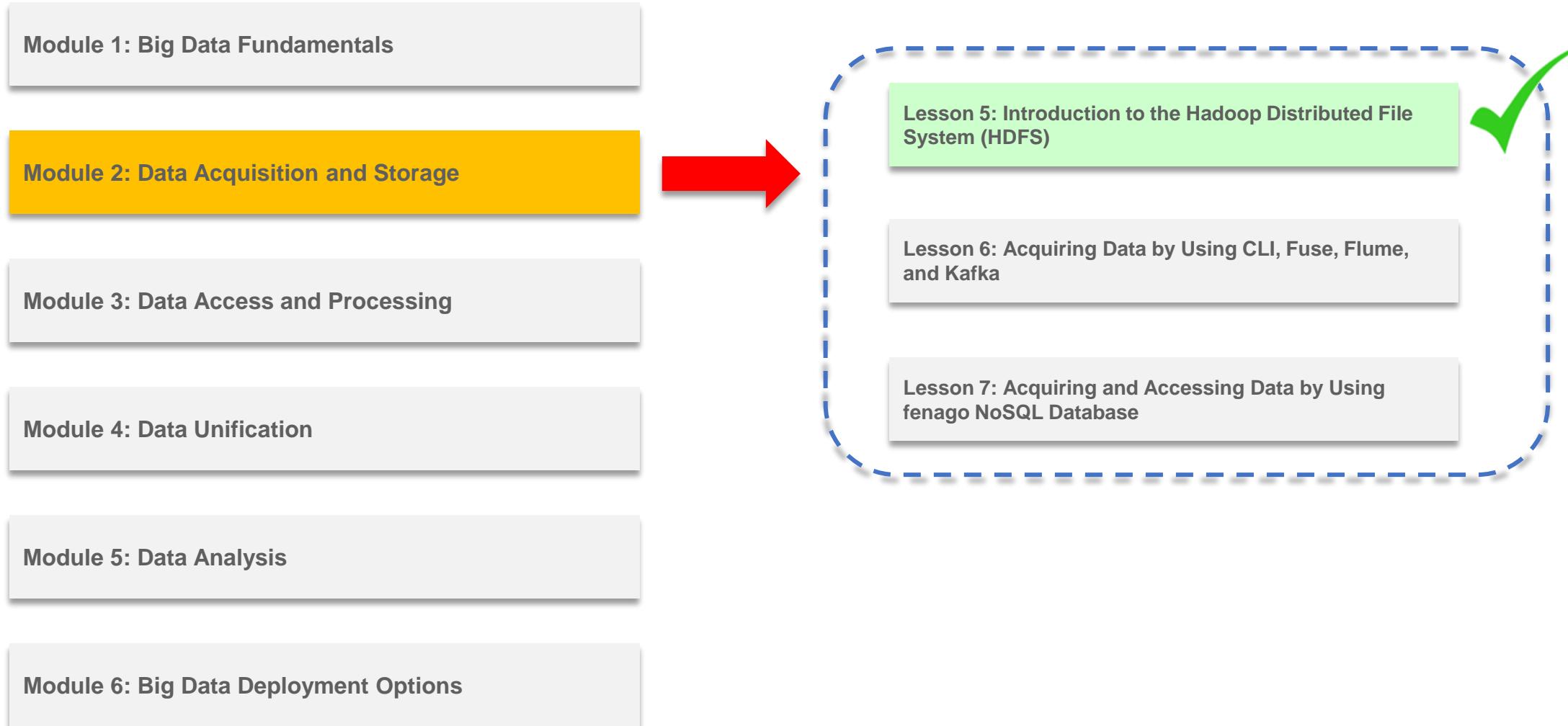
Summary

- In this lesson, you should have learned how to:
 - Define Hadoop and the *Hadoop Ecosystem*
 - Describe the Hadoop core components
 - Choose a Hadoop Distribution
 - List some of the other related projects in the Hadoop Ecosystem



Introduction to the Hadoop Distributed File System (HDFS)

Course Road Map



Objectives

- After completing this lesson, you should be able to:
 - Describe the architectural components of HDFS
 - Interact with data stored in HDFS by using various methods



Agenda

- Understand the architectural components of HDFS
- Interact with data stored in HDFS
 - Hue
 - Hadoop client file system shell command-line interface (CLI)
 - WebHDFS
 - HttpFS

HDFS Design Principles and Characteristics



Leader-follower architecture

Fault-tolerant (HA)

Redundant

Supports MapReduce & Spark

Scalable

Commodity Hardware

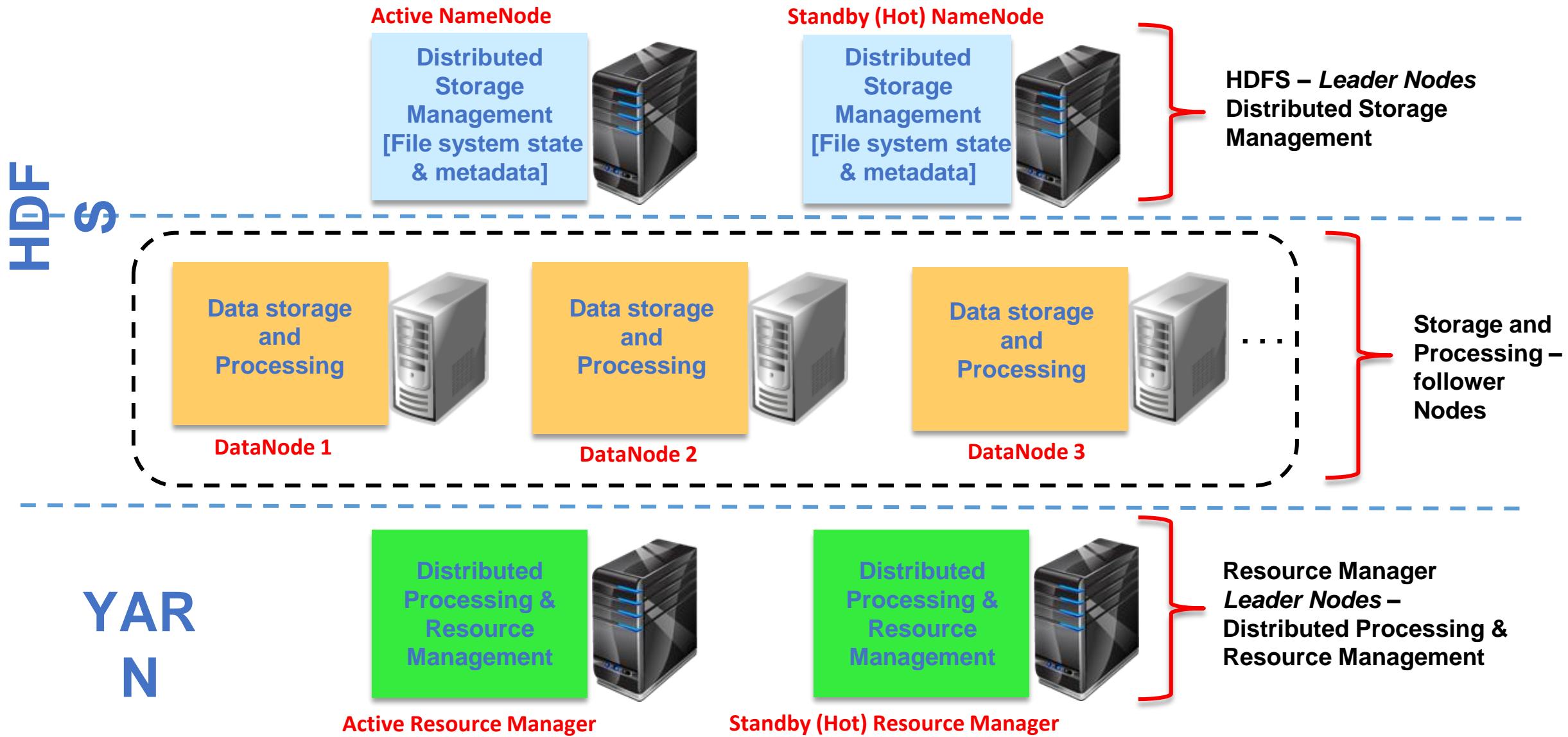
HDFS Key Definitions

Term	Description
Hadoop	A batch (MapReduce), interactive, or real-time (Spark and MR2) processing infrastructure that stores and distributes files and distributes work across a group of servers (nodes)
Hadoop Cluster	A collection of racks containing Leader and follower nodes
Blocks	HDFS breaks down a data file into blocks or “chunks” and stores the data blocks on different follower DataNodes in the Hadoop cluster.
Replication Factor	HDFS makes three copies of data blocks and stores them on different data nodes/racks in the Hadoop cluster.
NameNode (NN)	A service (daemon) that maintains a directory of all files in HDFS and tracks where data is stored in the HDFS cluster. It basically manages the file system’s metadata (<u>does not contain actual data</u>).
DataNode (DN)	This is where the data is stored (HDFS) and processed (MapReduce). This is a follower node. HDFS stores the blocks or “chunks of data for a set of files on the data nodes.”

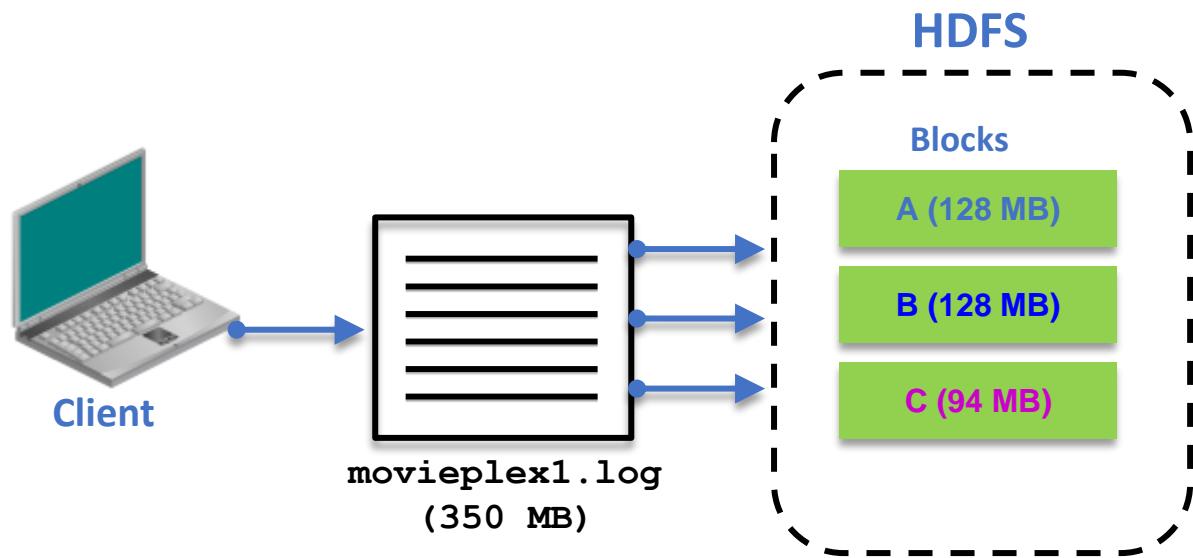
HDFS Deployments: High Availability (HA) and Non-HA

- Non-HA Deployment (Prior to Hadoop 2.0):
 - Uses the NameNode/Secondary NameNode architecture
 - The Secondary NameNode is not a failover for the NameNode.
 - The NameNode was the **Single Point of Failure (SPOF)** of the cluster prior to Hadoop 2.0 and CDH 4.0.
- HA Deployment (Hadoop 2.0 and later):
 - HDFS HA addresses the SPOF by running two redundant NameNodes in the same cluster to provide a **Fast Failover** if needed:
 - **Active NameNode**: Responsible for all client operations in the cluster
 - **Standby NameNode**: Acts as a “hot” backup to the Active NameNode, maintaining enough system information to provide a fast failover if necessary
 - **HA allows a fast failover** to a new NameNode in case a machine crashes.
- In this course, you will focus on the HDFS HA deployment option only.

Sample Hadoop High Availability (HA) Cluster



HDFS Files and Blocks

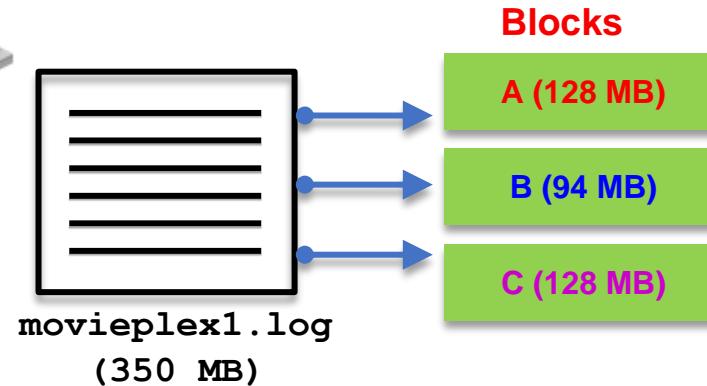


- Files in HDFS consist of blocks.
- HDFS blocks default to 128 MB in size (configurable).
- *Files are “chunked” into blocks as they are ingested (using Flume or Kafka) into HDFS.*

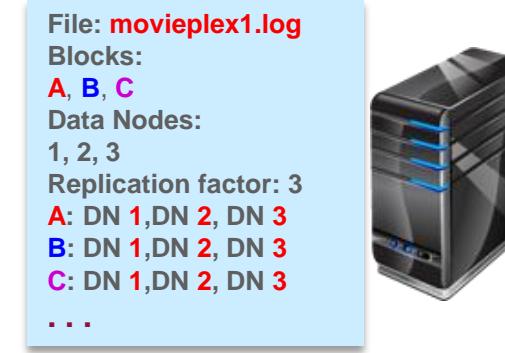
Assuming a default block size of 128 MB, HDFS ingests the `movieplex1.log` file into (3) blocks:

- A (128 MB)
- B (128 MB)
- C (94 MB)

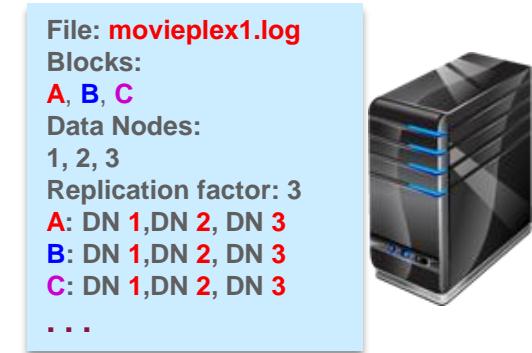
Blocks are Replicated in the Cluster Upon Ingestion into HDFS



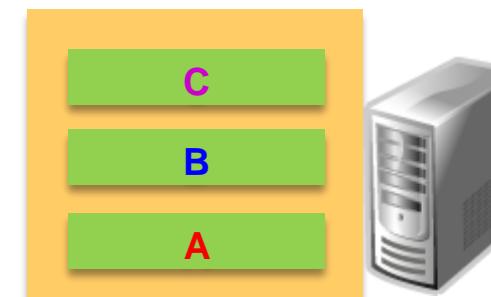
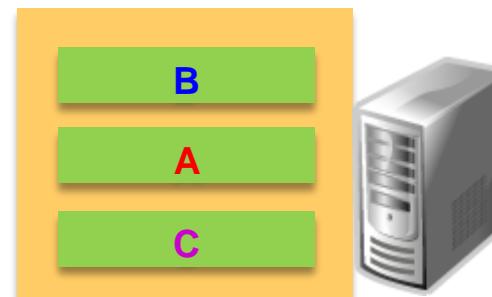
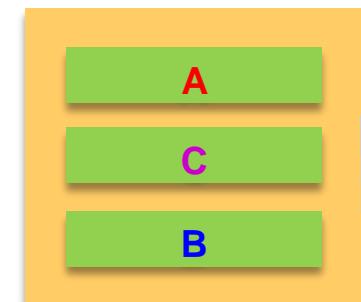
Active NameNode and (Leader)



Standby (Hot) NameNode (Leader)



Default replication factor = 3



Active and Standby NameNodes Daemons

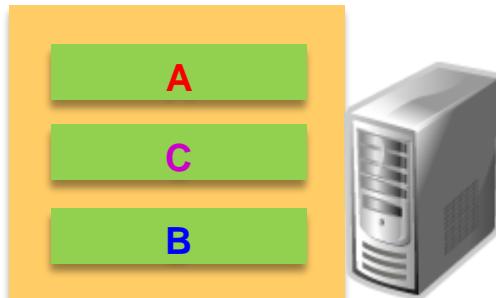


NameNode stores file system metadata such as:

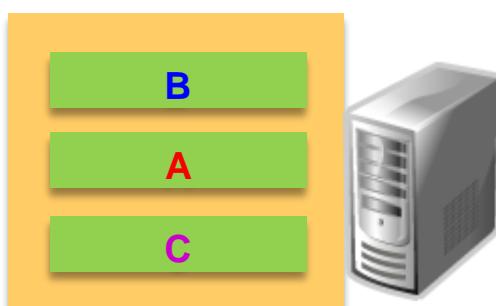
- File information (name, updates, replication factor, etc.)
- File blocks information and locations
- Access rights to the file
- Number of files in the cluster
- Number of DataNodes in the cluster

Active NameNode and
Standby (Hot) NameNode (Leaders)

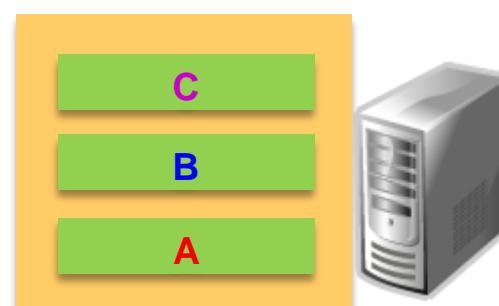
File: movieplex1.log
File: movieplex1.log
Blocks:
Data Nodes:
Replication Factor:
A: DN 1, DN 2, DN 3
B: DN 1, DN 2, DN 3
C: DN 1, DN 2, DN 3
...



DataNode 1
(follower)



DataNode 2
(follower)



DataNode 3 (salve)

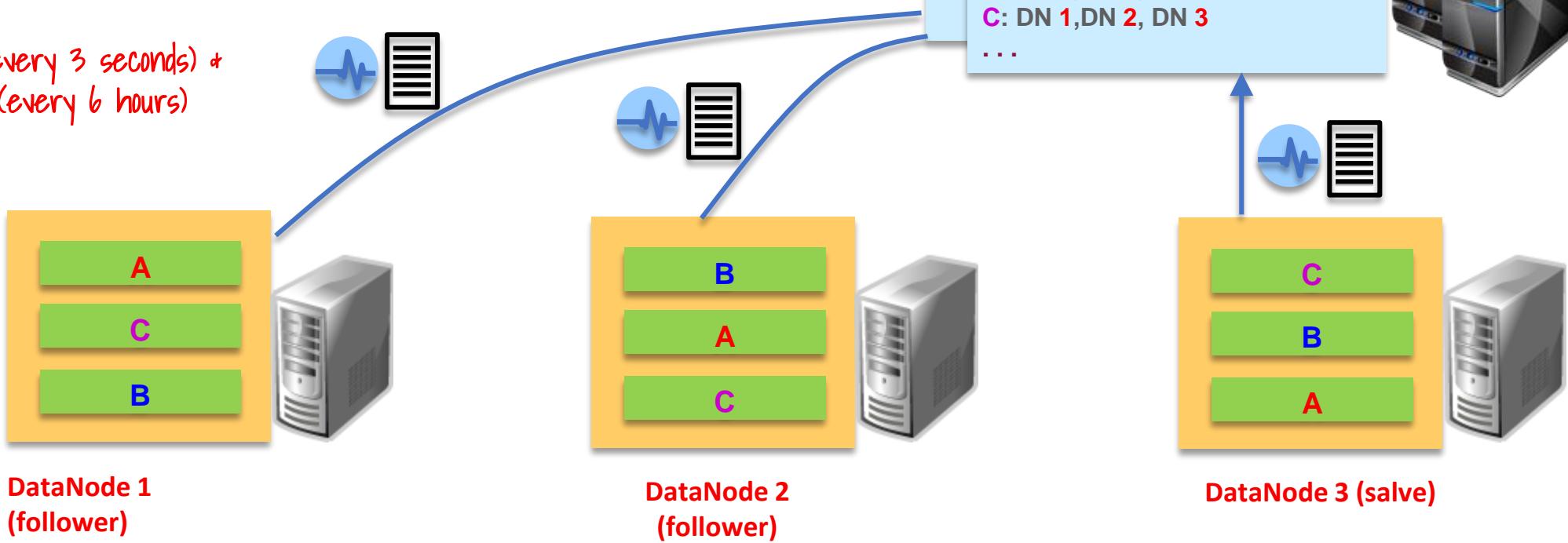
DataNodes Daemons



DataNodes

- Serve read and write requests from clients
- Perform block creation, deletion, and replication based on instructions from the NameNode
- Provide simultaneous send/receive operations to DataNodes during replication ("replication pipelining")

Heartbeat (every 3 seconds) +
Blockreport (every 6 hours)

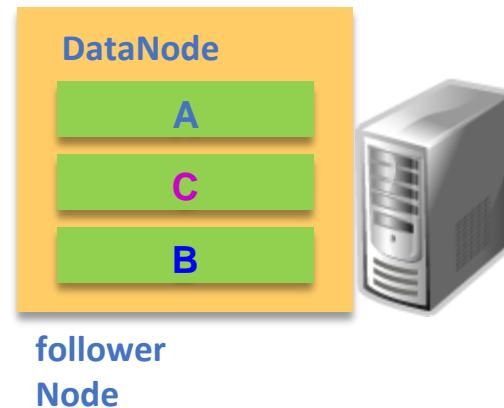


Functions of the NameNode

- Acts as the repository for all HDFS metadata
- Maintains the file system namespace
- Executes the directives for opening, closing, and renaming files and directories
- Stores the HDFS state in an image file (`fsimage`)
- Stores file system modifications in an edit log file (`edits`)
- On startup, merges the `fsimage` and `edits` files, and then empties `edits`
- Places replicas of blocks on multiple racks for fault tolerance
- Records the number of replicas (replication factor) of a file specified by an application

Functions of DataNodes

- DataNodes perform the following functions:
 - Serving read and write requests from the file system clients
 - Performing block creation, deletion, and replication based on instructions from the NameNode
 - Providing simultaneous send/receive operations to DataNodes during replication (“replication pipelining”)



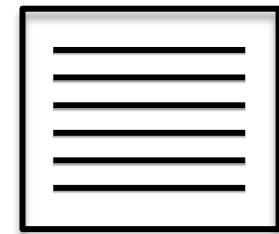
- Heartbeat (I am alive!) every 3 seconds
- Blockreport (what data I am storing!) every 6 hours to NN



Writing a File to HDFS: Example

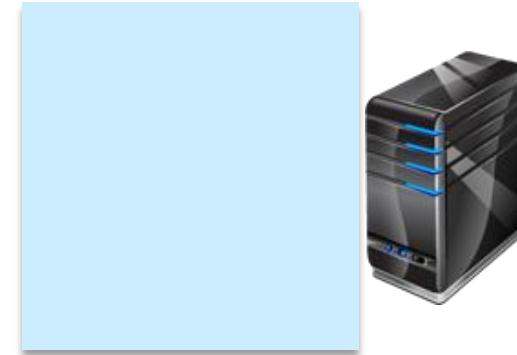


Client

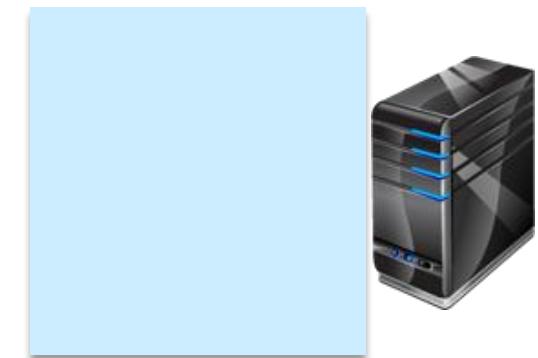


movieplex1.log
(350 MB)

Active NameNode and (Leader)



Standby (Hot) NameNode (Leader)



Default replication factor = 3



DataNode 1
(follower)



DataNode 2
(follower)

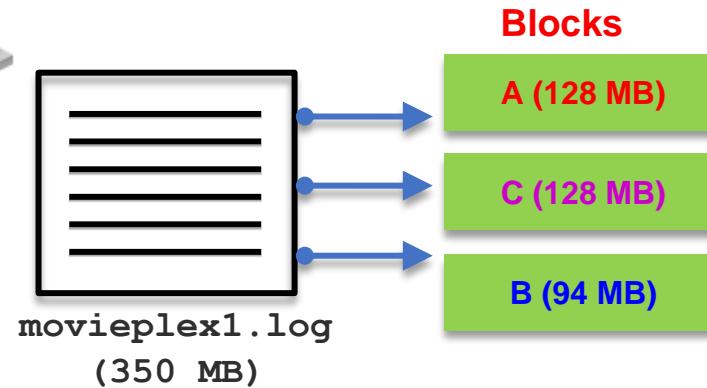


DataNode 3
(follower)

Writing a File to HDFS: File is “Chunked” into Blocks – Example

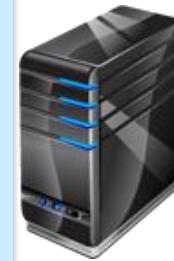


Client



Active NameNode and (Leader)

File: **movieplex1.log**
Blocks:
A, B, C
Data Nodes:
1, 2, 3
Replication factor: 3



Standby (Hot) NameNode (Leader)

File: **movieplex1.log**
Blocks:
A, B, C
Data Nodes:
1, 2, 3
Replication factor: 3



Default replication factor = 3



DataNode 1
(follower)

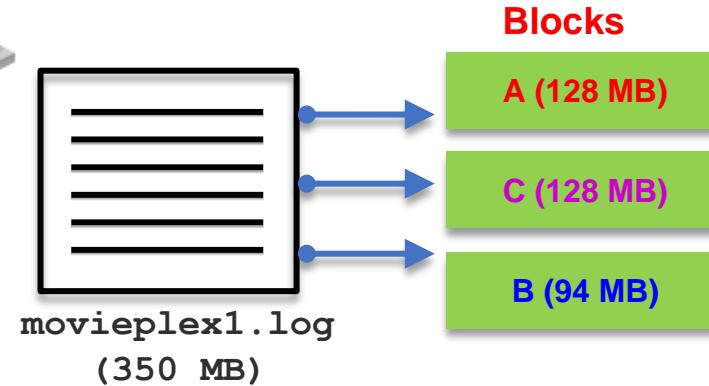
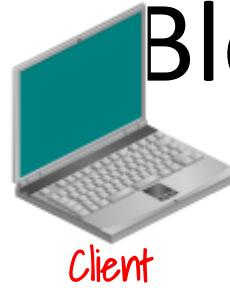


DataNode 2
(follower)



DataNode 3
(follower)

Writing a File to HDFS: Pipeline Created, Block A – Example



Active NameNode and (Leader)

File: **movieplex1.log**
Blocks:
A, B, C
Data Nodes:
1, 2, 3
Replication factor: 3
A: DN 1, DN 2, DN 3

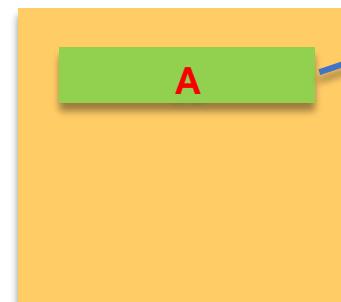


Standby (Hot) NameNode (Leader)

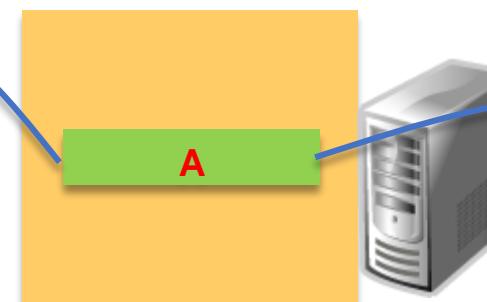
File: **movieplex1.log**
Blocks:
A, B, C
Data Nodes:
1, 2, 3
Replication factor: 3
A: DN 1, DN 2, DN 3



Default replication factor = 3



DataNode 1
(follower)



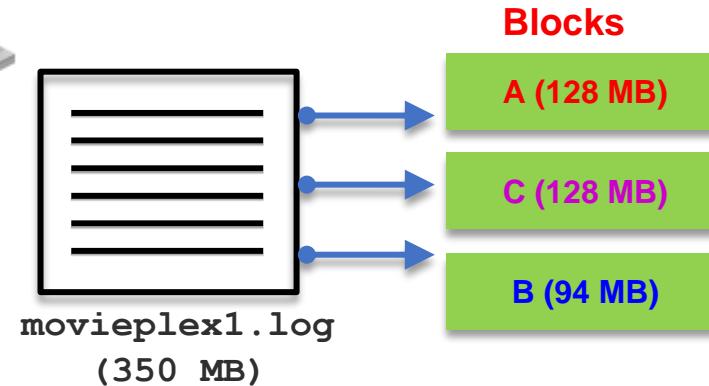
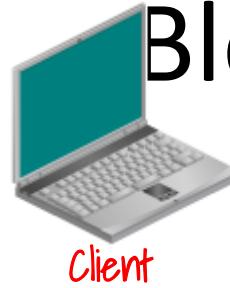
DataNode 2
(follower)



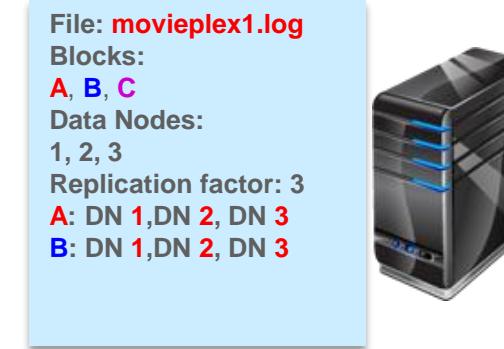
DataNode 3
(follower)



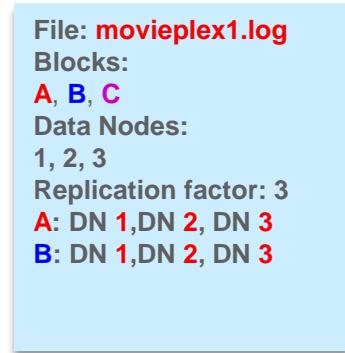
Writing a File to HDFS: Pipeline Created, Block B – Example



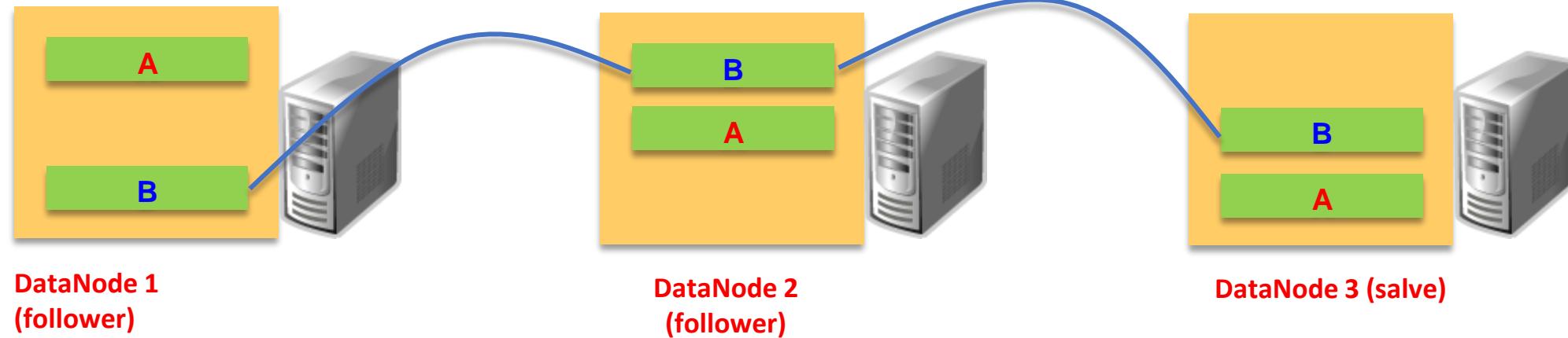
Active NameNode and (Leader)



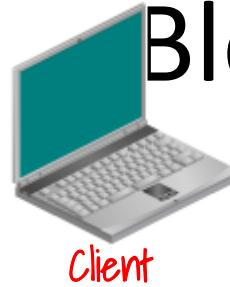
Standby (Hot) NameNode (Leader)



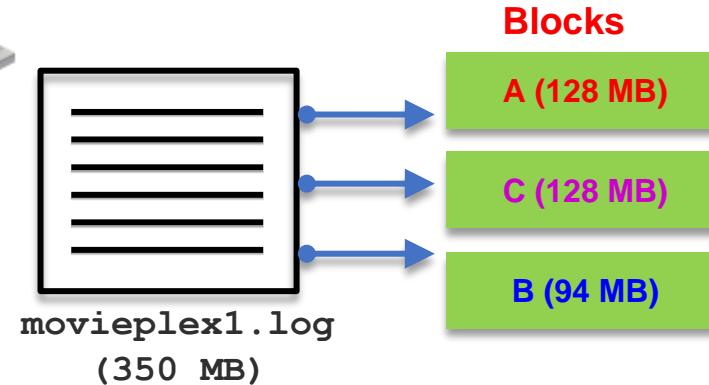
Default replication factor = 3



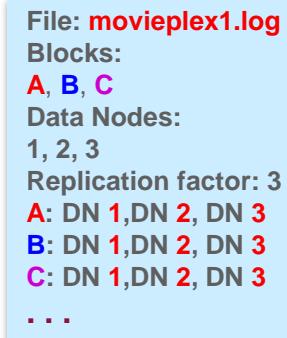
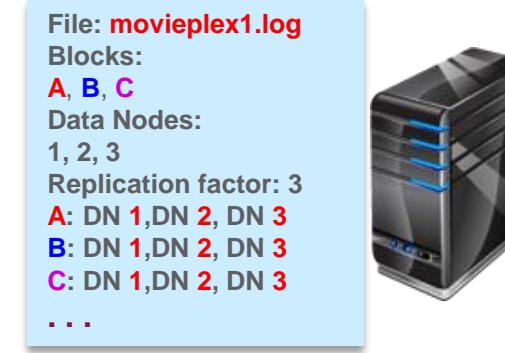
Writing a File to HDFS: Pipeline Created, Block C – Example



Client

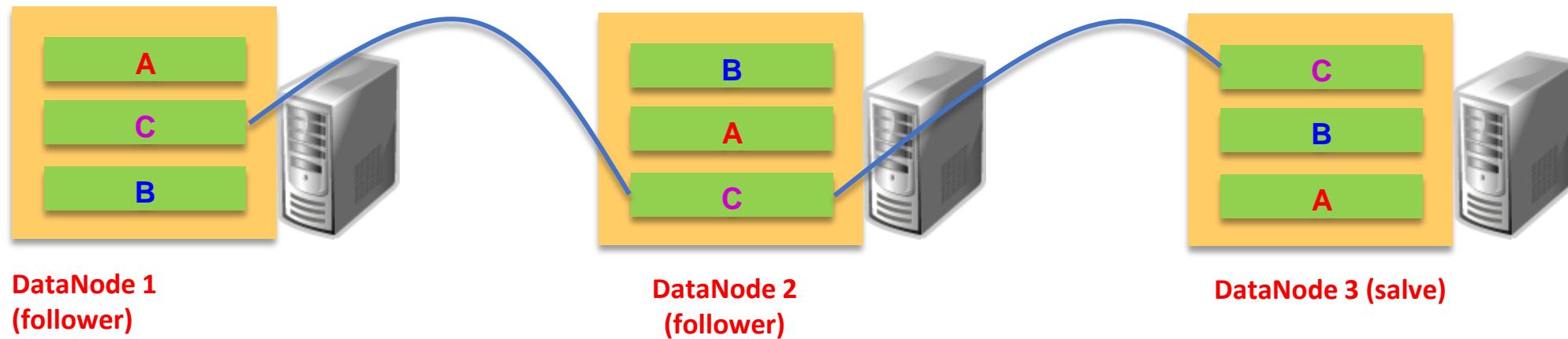


Active NameNode and (Leader)



✓ Ack messages from the pipeline are sent back to the client (blocks are copied)

Default replication factor = 3

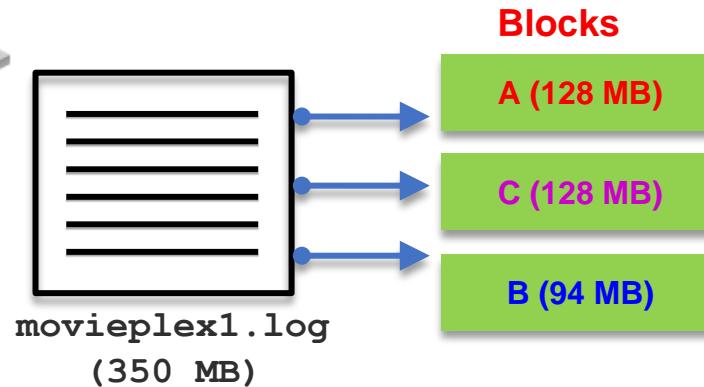




Writing a File to HDFS: Example



Client



Active NameNode and (Leader)

File: movieplex1.log
Blocks:
A, B, C
Data Nodes:
1, 2, 3
Replication factor: 3
A: DN 1, DN 2, DN 3
B: DN 1, DN 2, DN 3
C: DN 1, DN 2, DN 3
...



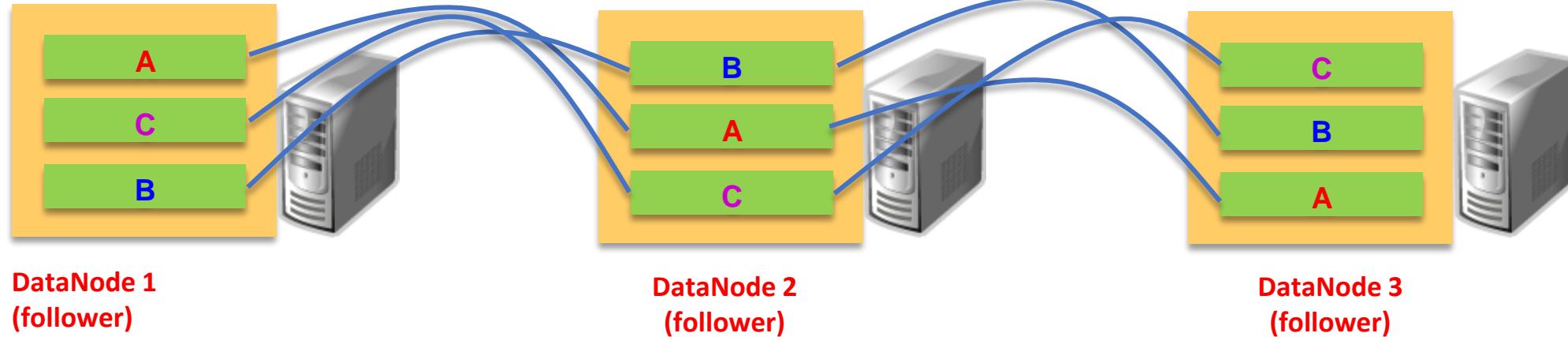
Standby (Hot) NameNode (Leader)

File: movieplex1.log
Blocks:
A, B, C
Data Nodes:
1, 2, 3
Replication factor: 3
A: DN 1, DN 2, DN 3
B: DN 1, DN 2, DN 3
C: DN 1, DN 2, DN 3
...



✓ Ack messages from the pipeline are sent back to the client (blocks are copied)

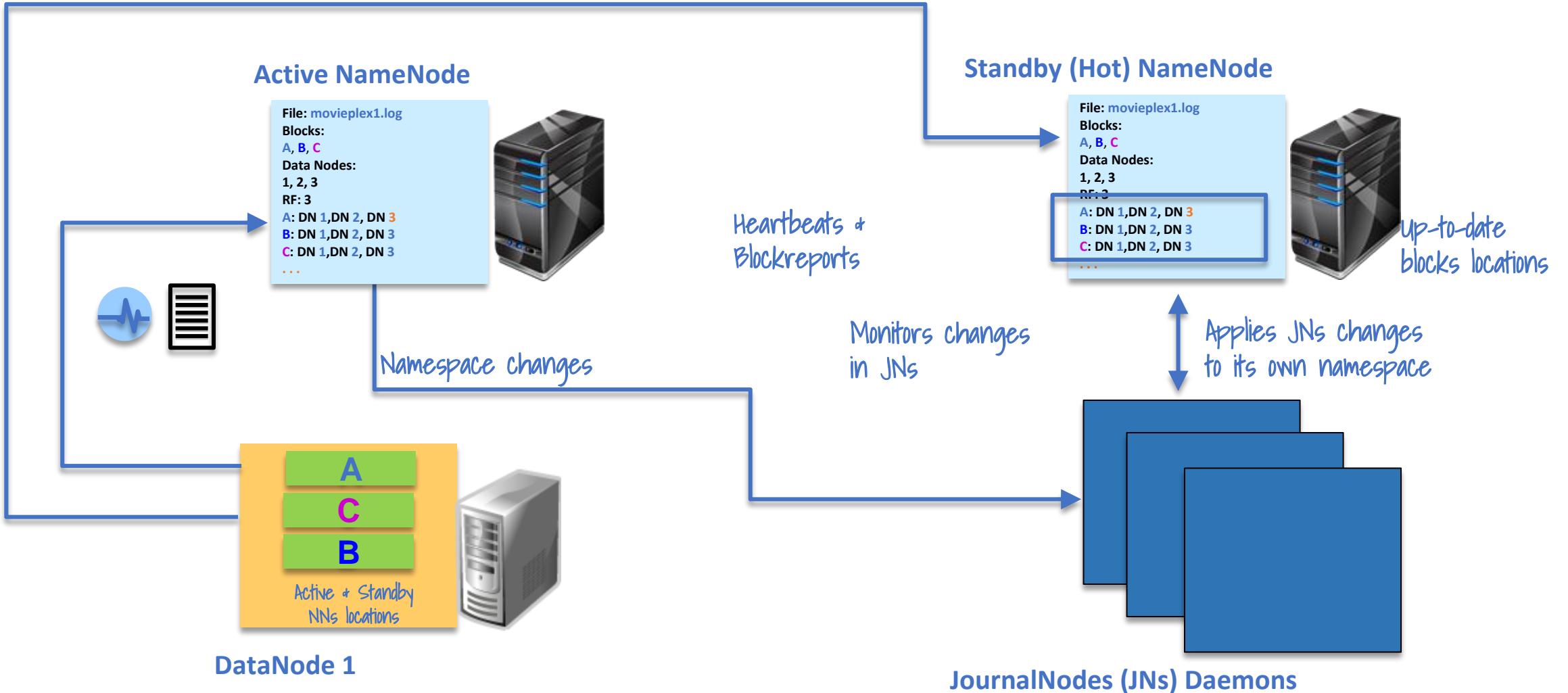
Default replication factor = 3



HDFS High Availability (HA) Using the Quorum Journal Manager (QJM)

- Prior to Hadoop 2.0.0, the NameNode was a single point of failure (SPOF) in an HDFS cluster.
- Each cluster had a single NameNode.
- The cluster is unavailable when the NameNode machine crashes or during software and hardware maintenance.
- HDFS HA addresses this problem by:
 - Running two redundant NameNodes in the same cluster:
An **Active** NameNode and a **Hot Standby** NameNode
- HA provides fast failover to a new NameNode when the NameNode machine crashes or during regular software and hardware maintenance.
- Big Data Appliance (BDA) uses the HA implementation.

HDFS High Availability (HA) Using the Quorum Journal Manager (QJM) Feature



Enabling HDFS HA

- Using Cloudera Manager:
 - Enable HA and Automatic Failover
- Using the command-line interface to configure automatic failover. Automatic failover adds the following components to an HDFS deployment:
 - A ZooKeeper quorum, which provides:
 - Failure detection
 - Active NameNode election
 - ZKFailoverController process (ZKFC), which provides:
 - Health monitoring
 - ZooKeeper session management
 - ZooKeeper-based election

Data Replication Rack-Awareness in HDFS

Block A :



Block B :



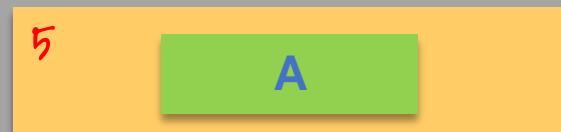
Block C :



Rack 1



Rack 2

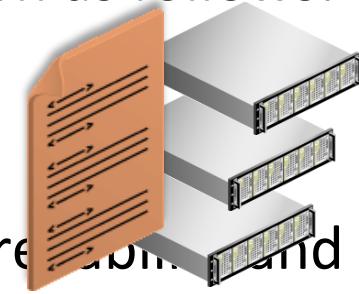


Rack 3



Data Replication Process

- The number of file replicas that will be maintained by HDFS (the “replication factor”) is stored in the NameNode.
 - If the factor is (3), the ***HDFS Placement Policy*** directs replication as follows:
 - One copy on one node in a local rack
 - One copy on a different remote rack
 - One copy on a different node in the same remote rack
 - This policy improves the write performance and ensures data redundancy and availability.
 - *If the reader process requires data, HDFS makes sure that it pulls the nearest replica for the task, thereby reducing the read latency (data locality).*



Accessing HDFS

The screenshot illustrates the process of accessing the HDFS NameNode interface via a web browser. On the left, a Mozilla Firefox window is shown with the address bar containing "Connecting...". Below the address bar, the "Hadoop" menu item is highlighted with a blue box. A red arrow points from the "NameNode bigdatalite.localdomain:8020" link in this menu to the right-hand browser window. The right-hand window displays the "Hadoop NameNode bigdatalite.localdomain:8020 - Mozilla Firefox" title bar. The address bar in this window shows "localhost:50070/dfshealth.jsp". The main content area of the right window is titled "NameNode 'bigdatalite.localdomain:8020' (active)". It contains a table with the following information:

Started:	Sun Jan 29 20:17:54 EST 2017
Version:	2.6.0-cdh5.8.0, 042da8b868a212c843bcbf3594519dd26e816e79
Compiled:	2016-07-12T23:02Z by jenkins from Unknown
Cluster ID:	cluster7
Block Pool ID:	BP-314833115-127.0.0.1-1475613842090

Below the table, there are two links: "Browse the filesystem" and "NameNode Logs".

Cluster Summary

Security is OFF

4366 files and directories, 1248 blocks = 5614 total.

Heap Memory used 23.62 MB is 18% of Committed Heap Memory 125.50 MB. Max Heap Memory is 889

Non Heap Memory used 56.55 MB is 97% of Committed Non Heap Memory 57.77 MB. Max Non Heap M

Configured Capacity	:	196.73 GB
DFS Used	:	2.18 GB
Non DFS Used	:	10.82 GB
DFS Remaining	:	183.74 GB
DFS Used%	:	1 11%

Agenda

- Understand the architectural components of HDFS
- Interact with data stored in HDFS
 - Hue
 - Hadoop client
 - WebHDFS
 - HttpFS

Using Cloudera Hue to Interact with HDFS

<http://bdalnode03.example.com:8888>

The screenshot shows the Cloudera Hue File Browser interface running in Mozilla Firefox. The browser title bar reads "Hue - File Browser - Mozilla Firefox". The address bar shows the URL "bigdatalite:8888/filebrowser/". The main navigation bar includes links for "Hue", "MoviePlex", "Hadoop", "Spatial and Graph", "BDD", "Apex", "ORDS Lab", "Solr Admin", and "SQL Pattern Matching". Below the navigation bar is a blue header bar with the "HUE" logo and various menu items: "Query Editors", "Data Browsers", "Workflows", "Search", and "Security". A blue arrow points from the text "Manage HDFS" to a blue button labeled "Manage HDFS" located in the top right corner of the blue header bar. The main content area is titled "File Browser" and shows a file listing for the directory "/user/oracle". The table has columns: Name, Size, User, Group, Permissions, and Date. The data in the table is as follows:

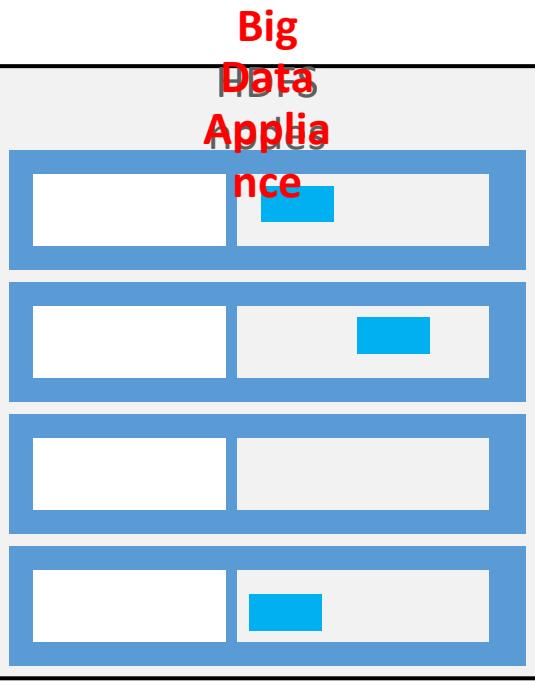
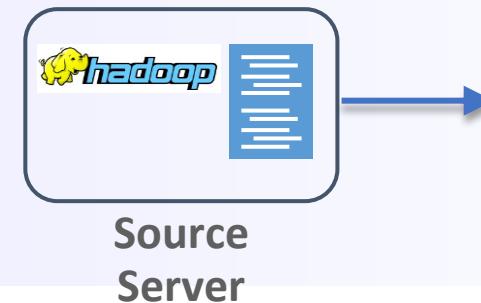
Name	Size	User	Group	Permissions	Date
..		hdfs	supergroup	drwxr-xr-x	June 01, 2016 02:14 PM
.		oracle	oracle	drwxr-xr-x	June 01, 2016 03:59 PM
.Trash		oracle	oracle	drwxr-xr-x	May 23, 2016 05:42 PM
.sparkStaging		oracle	oracle	drwxr-xr-x	May 23, 2016 05:39 PM

Using Hadoop Client to Batch Load Data

- Advantages:
 - Enables direct HDFS writes without intermediate file staging on Linux FS
 - Easy to scale:
 - Initiate concurrent puts for multiple files.
 - HDFS will leverage multiple “target” servers and ingest faster.
- Disadvantages:
 - Additional software (Hadoop client) needs to be installed on the source server.

HDFS put command issued from the Hadoop client on the source server

Client



HDFS Commands



Apache  > Hadoop  > Apache Hadoop Project Dist POM > Apache Hadoop 2.7.2 Wiki  | git

General

- Overview
- Single Node Setup
- Cluster Setup
- Hadoop Commands Reference**
- FileSystem Shell
- Hadoop Compatibility
- Interface Classification
- FileSystem Specification

Common

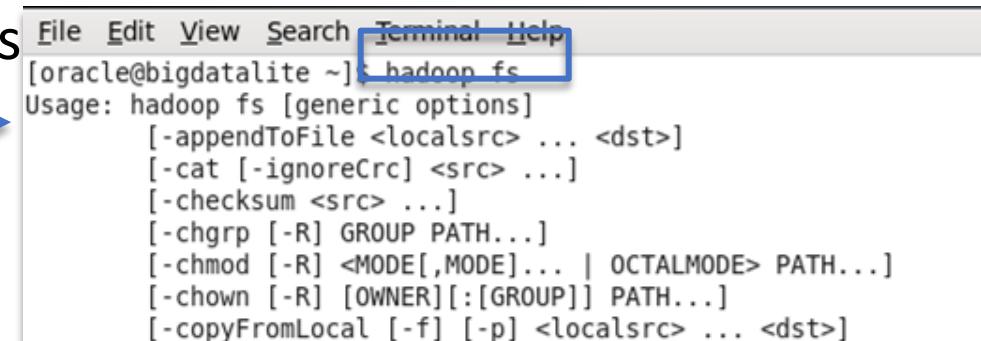
- CLI Mini Cluster
- Native Libraries
- Proxy User
- Rack Awareness
- Secure Mode
- Service Level
- Authorization
- HTTP Authentication
- Hadoop KMS

- Hadoop Commands Guide
 - Overview
 - Generic Options
 - **Hadoop Common Commands**
 - **User Commands**
 - archive
 - checknative
 - classpath
 - credential
 - distcp
 - fs
 - jar
 - key
 - trace
 - version
 - CLASSNAME
 - **Administration Commands**
 - daemonlog

HDFS File System (FS) Shell Interface

- HDFS supports a traditional hierarchical file organization.
- You can use the **FS shell** command-line interface to interact with the data in HDFS.
- The syntax of this command set is similar to that of other shells.
 - You can create, remove, rename, and move directories/files.
- You can invoke FS shell as follows

```
hadoop fs <args>
```



```
[oracle@bigdatalite ~]$ hadoop fs
Usage: hadoop fs [generic options]
      [-appendToFile <localsrc> ... <dst>]
      [-cat [-ignoreCrc] <src> ...]
      [-checksum <src> ...]
      [-chgrp [-R] GROUP PATH...]
      [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
      [-chown [-R] [OWNER][:[GROUP]] PATH...]
      [-copyFromLocal [-f] [-p] <localsrc> ... <dst>]
```

- The general command-line syntax is as follows:

```
hadoop command [genericOptions] [commandOptions]
```

HDFS FS (File System) Shell Interface

```
hadoop fs -help
```

```
[oracle@bigdatalite ~]$ hadoop fs -help
Usage: hadoop fs [generic options]
      [-appendToFile <localsrc> ... <dst>]
      [-cat [-ignoreCrc] <src> ...]
      [-checksum <src> ...]
      [-chgrp [-R] GROUP PATH...]
      [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
      [-chown [-R] [OWNER][:[GROUP]] PATH...]
      [-copyFromLocal [-f] [-p] <localsrc> ... <dst>]
      [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
      [-count [-q] <path> ...]
      [-cp [-f] [-p] <src> ... <dst>]
      [-createSnapshot <snapshotDir> [<snapshotName>]]
      [-deleteSnapshot <snapshotDir> <snapshotName>]
      [-df [-h] [<path> ...]]
      [-du [-s] [-h] <path> ...]
      [-expunge]
      [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
      [-getfacl [-R] <path>]
      [-getmerge [-nl] <src> <localdst>]
      [-help [cmd ...]]
      [-ls [-d] [-h] [-R] [<path> ...]]
      [-mkdir [-p] <path> ...]
      [-moveFromLocal <localsrc> ... <dst>]
      [-moveToLocal <src> <localdst>]
      [-mv <src> ... <dst>]
      [-put [-f] [-p] <localsrc> ... <dst>]
      [-renameSnapshot <snapshotDir> <oldName> <newName>]
      [-rm [-f] [-r|-R] [-skipTrash] <src> ...]
      [-rmdir [--ignore-fail-on-non-empty] <dir> ...]
      [-setfacl [-R] [-b|l|k] [-m|-x] <aclSpec> <path> ...]
```

FS Shell Commands

Apache > Hadoop > Apache Hadoop Project Dist POM > Apache Hadoop 2.7.2

- General
 - Overview
 - Single Node Setup
 - Cluster Setup
 - Hadoop Commands Reference
 - FileSystem Shell
 - Hadoop Compatibility
 - Interface Classification
 - FileSystem Specification
- Common
 - CLI Mini Cluster
 - Native Libraries
 - Proxy User
 - Rack Awareness
 - Secure Mode
 - Service Level
 - Authorization
 - HTTP Authentication
 - Hadoop KMS
 - Tracing
- HDFS
 - HDFS User Guide
 - HDFS Commands Reference
 - High Availability With QJM
 - High Availability With NFS
 - Federation
 - ViewFs Guide
 - HDFS Snapshots
 - HDFS Architecture
 - Edit Viewer

- Overview
 - appendToFile
 - cat
 - checksum
 - chgrp
 - chmod
 - chown
 - copyFromLocal
 - copyToLocal
 - count
 - cp
 - createSnapshot
 - deleteSnapshot
 - df
 - du
 - dus
 - expunge
 - find
 - get
 - getfacl
 - getfattr
 - getmerge
 - help
 - ls
 - lsr
 - mkdir
 - moveFromLocal
 - moveToLocal
 - mv
 - put
 - renameSnapshot
 - rm
 - rmdir
 - rmr

oracle@bigdatalite:~

```
[oracle@bigdatalite ~]$ ls -l
total 8316
drwxr-xr-x. 2 oracle oinstall 4096 Jan 23 13:16 bigdatasql-hol
-rw-r--r--. 1 oracle oinstall 5545 Jan 23 13:07 bigdatasql-hol.zip
drwxr-xr-x. 3 oracle oracle 4096 Jan 23 13:42 Desktop
drwxr-xr-x. 2 oracle oracle 4096 Oct 2 2015 Documents
drwxr-xr-x. 2 oracle oracle 4096 Oct 24 20:08 Downloads
drwxr-xr-x. 16 oracle oinstall 4096 Jan 23 13:15 exercises
-rw-r--r--. 1 oracle oinstall 4892012 Jan 23 13:07 exercises.zip
lrwxrwxrwx. 1 oracle oracle 31 Oct 4 15:21 GettingStarted -> /home/oracle/
drwxr-xr-x. 4 oracle oinstall 4096 Oct 24 16:21 movie
drwxr-xr-x. 2 oracle oracle 4096 Oct 2 2015 Music
drwxr-xr-x. 4 oracle oinstall 4096 Jan 26 2015 orabalancerdemo-2.3.0-h2
-rw-r--r--. 1 oracle oinstall 3536258 Jan 23 13:07 orabalancerdemo-2.3.0-h2.zip
drwxr-xr-x. 2 oracle oracle 4096 Jan 15 2015 Pictures
drwxr-xr-x. 2 oracle oinstall 4096 Jan 23 13:16 practice_commands
-rw-r--r--. 1 oracle oinstall 11219 Jan 23 13:07 practice_commands.zip
drwxr-xr-x. 2 oracle oracle 4096 Oct 2 2015 Public
drwxr-x---. 4 oracle oracle 4096 Oct 18 18:23 scripts
drwxr-xr-x. 9 oracle oracle 4096 Oct 24 16:21 src
drwxr-xr-x. 2 oracle oracle 4096 Oct 2 2015 Templates
drwxr-xr-x. 2 oracle oracle 4096 Oct 2 2015 Videos
[oracle@bigdatalite ~]$ hadoop fs -ls
Found 9 items
drwxr-xr-x - oracle oracle 0 2016-10-21 15:52 .sparkStaging
drwx----- - oracle oracle 0 2016-10-24 18:24 .staging
drwxr-xr-x - hdfs oracle 0 2016-10-24 16:16 indexMetadata
drwxr-xr-x - hdfs oracle 0 2016-10-24 16:14 jobRegistry
drwxr-xr-x - oracle oracle 0 2016-10-04 19:29 mediademo
drwxr-xr-x - oracle oracle 0 2016-10-04 19:30 moviedemo
drwxr-xr-x - oracle oracle 0 2016-10-04 19:30 moviework
drwxr-xr-x - oracle oracle 0 2016-10-04 19:30 oggdemo
drwxr-xr-x - oracle oracle 0 2016-10-04 19:30 oozie-oozi
[oracle@bigdatalite ~]$
```

Local filesystem

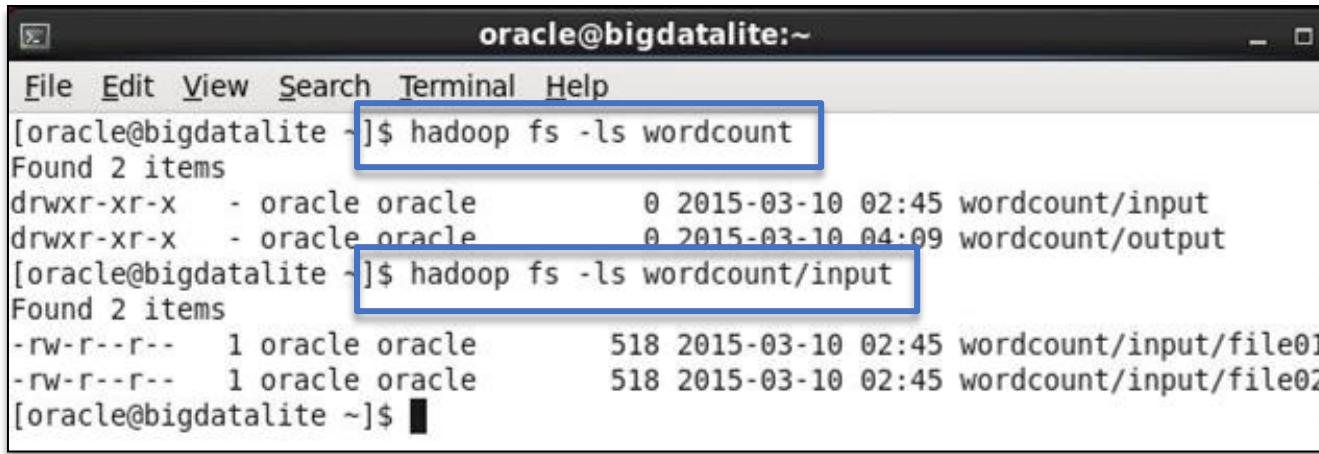
Hadoop namespace

Sample FS Shell Commands

Command	Description
ls	Lists attributes of files and directories
cat	Copies source paths to stdout
cp	Copy files from source to destination in HDFS
mv	Moves files from source to destination. Moving files across file systems is not permitted.
rm	Deletes files specified. The <code>-r</code> option deletes the directory and its contents.
put	Copies files from the local file system to HDFS
get	Copies files from HDFS to the local file system
mkdir	Creates one or more HDFS directories
rmdir	Deletes a directory
jar	Runs a jar file. Users can bundle their MapReduce code in a JAR file and execute it using this command.
version	Prints the Hadoop version
help	Return usage output (available commands to use)

ls Command

```
hadoop fs -ls
```



The screenshot shows a terminal window titled "oracle@bigdatalite:~". The user runs the command "hadoop fs -ls wordcount". The output shows two items: "wordcount/input" and "wordcount/output", both listed as directories. Then, the user runs "hadoop fs -ls wordcount/input", which lists two files: "file01" and "file02", each with size 518 and modified on 2015-03-10 02:45.

```
oracle@bigdatalite:~$ hadoop fs -ls wordcount
Found 2 items
drwxr-xr-x  - oracle oracle          0 2015-03-10 02:45 wordcount/input
drwxr-xr-x  - oracle oracle          0 2015-03-10 04:09 wordcount/output
[oracle@bigdatalite ~]$ hadoop fs -ls wordcount/input
Found 2 items
-rw-r--r--  1 oracle oracle      518 2015-03-10 02:45 wordcount/input/file01
-rw-r--r--  1 oracle oracle      518 2015-03-10 02:45 wordcount/input/file02
[oracle@bigdatalite ~]$
```

directories

files

- For a file, it returns stat on the file with the following format:
 - permissions number_of_replicas userid groupid filesize modification_date modification_time filename
- For a directory, it returns a list of its direct children as in UNIX. A directory is listed as:
 - permissions userid groupid modification_date modification_time dirname

mkdir and copyFromLocal Commands

Create an HDFS directory named `curriculum` by using the `mkdir` command:

```
[oracle@bigdatalite ~]$ hadoop fs -mkdir curriculum
[oracle@bigdatalite ~]$ hadoop fs -ls
Found 9 items
drwx-----  - oracle oracle      0 2014-08-25 05:55 .Trash
drwx-----  - oracle oracle      0 2015-03-10 04:09 staging
drwxr-xr-x  - oracle oracle      0 2015-03-24 09:38 curriculum
drwxr-xr-x  - oracle oracle      0 2014-01-12 18:15 moviedemo
drwxr-xr-x  - oracle oracle      0 2014-09-24 09:38 moviework
drwxr-xr-x  - oracle oracle      0 2014-09-08 15:50 oggdemo
drwxr-xr-x  - oracle oracle      0 2014-09-20 13:59 oozie-oozi
drwxr-xr-x  - oracle oracle      0 2015-03-24 00:57 test
drwxr-xr-x  - oracle oracle      0 2015-03-10 04:09 wordcount
[oracle@bigdatalite ~]$
```

Copy `lab_05_01.txt` from the local file system to the `curriculum` HDFS directory by using the `copyFromLocal` command:

```
[oracle@bigdatalite ~]$ cd Practice_Commands
[oracle@bigdatalite Practice_Commands]$ ls
lab_05_01.txt lab_09_01.txt lab_13_01.txt lab_15_01.txt lab_19_02.txt lab_21_02.txt
lab_07_01.txt lab_11_01.txt lab_13_02.txt lab_18_01.txt lab_19_03.txt lab_23_01.txt
lab_07_02.txt lab_11_02.txt lab_13_03.txt lab_18_02.txt lab_20_01.txt lab_27_01.txt
lab_07_04.txt lab_11_03.txt lab_14_01.txt lab_19_01.txt lab_21_01.txt
[oracle@bigdatalite Practice_Commands]$ hadoop fs -copyFromLocal lab_05_01.txt curriculum/lab_05_01.txt
[oracle@bigdatalite Practice_Commands]$ hadoop fs -ls curriculum
Found 1 items
-rw-r--r--  1 oracle oracle      524 2015-03-24 10:14 curriculum/lab_05_01.txt
[oracle@bigdatalite Practice_Commands]$
```

rm and cat Commands

Delete the curriculum HDFS directory by using the rm command. Use the -r option to delete the directory and any content under it recursively:

```
[oracle@bigdatalite Practice_Commands $ hadoop fs -rm -r curriculum  
15/03/24 10:31:01 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval  
= 0 minutes.  
Deleted curriculum
```

Display the contents of the part-r-00000 HDFS file by using the cat

```
[oracle@bigdatalite ~ $ hadoop fs -cat /user/oracle/wordcount/output/part-r-00000  
and    12  
awful   2  
bank    2  
company 4  
cover   2  
customer 6  
disappointed 6  
expensive 12  
insurance 18  
is      2  
professional 2  
protocols 2  
service 12  
staff    2  
terrible  4  
the     2  
unreliable 6  
very    6  
with    4  
worst   16  
worthless 4  
[oracle@bigdatalite ~]$
```

Using the hdfs fsck Command: Example

Use the hdfs fsck file system checking utility to perform health checks on the file system.

```
[oracle@bigdatalite ~]$ hdfs fsck /user/oracle/wordcount/output/part-r-00000 -files -blocks
15/03/26 01:49:08 WARN ssl.FileBasedKeyStoresFactory: The property 'ssl.client.truststore.location' has no
t been set, no TrustStore will be loaded
Connecting to namenode via http://bigdatalite.localdomain:50070
FSCK started by oracle (auth:SIMPLE) from /127.0.0.1 for path /user/oracle/wordcount/output/part-r-00000 a
t Thu Mar 26 01:49:09 EDT 2015
/user/oracle/wordcount/output/part-r-00000 208 bytes, 1 block(s): OK
0. BP-703742109-127.0.0.1-1398459391664:blk_1073754500_13678 len=208 repl=1

Status: HEALTHY
Total size:    208 B
Total dirs:    0
Total files:   1
Total symlinks:      0
Total blocks (validated): 1 (avg. block size 208 B)
Minimally replicated blocks: 1 (100.0 %)
Over-replicated blocks: 0 (0.0 %)
Under-replicated blocks: 0 (0.0 %)
Mis-replicated blocks: 0 (0.0 %)
Default replication factor: 1
Average block replication: 1.0
Corrupt blocks: 0
Missing replicas: 0 (0.0 %)
Number of data-nodes: 1
Number of racks: 1
FSCK ended at Thu Mar 26 01:49:09 EDT 2015 in 0 milliseconds

The filesystem under path '/user/oracle/wordcount/output/part-r-00000' is HEALTHY
[oracle@bigdatalite ~]$ █
```

Acquiring Data with Kafka

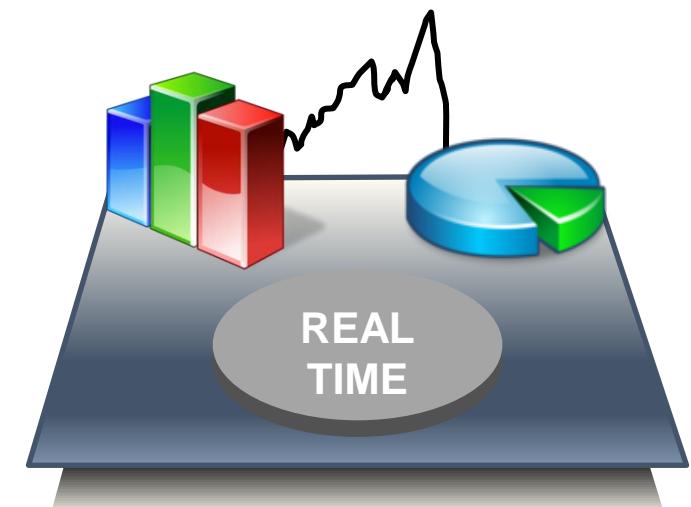
What Is Apache Kafka?

- It is a messaging system that collects and delivers high volumes of activity log data with low latency.
- Developed at LinkedIn, Kafka can handle over 500 billion events per day, spread over a number of data centers.
- In contrast to Flume, Kafka is about the online consumption of log data.
- Unlike traditional offline log processing systems, Kafka is designed to process log data in real-time.
- It enables data transfer between disparate data systems (such as relational, Hadoop, or NoSQL) that are geographically distributed.



Activity Log Processing in Real Time

- For Internet-based applications, log data is now required in real time.
- It enables user-oriented features such as:
 - Search
 - Ad targeting and reporting
 - Recommendations
 - Security against spam or unauthorized usage
 - Fraud detection
 - Newsfeed updates
- Log data is important for smooth running of the website, not just for offline log analysis.



Apache Kafka: Features

Publish and Subscribe

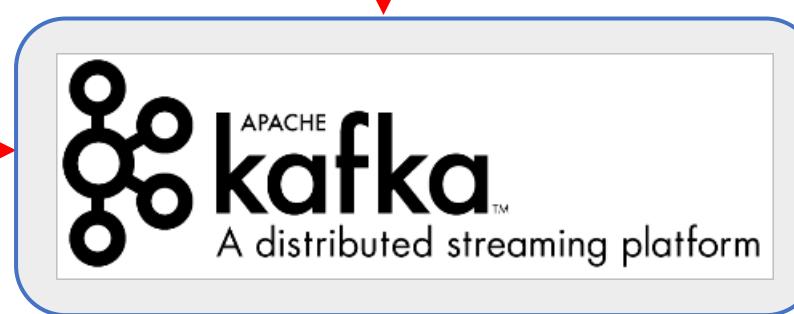
to streams of data like a messaging system.

Process

streams of data efficiently and in real time.

Store

streams of data safely in a distributed replicated cluster.



Apache Kafka: Basic Concepts

Kafka is run as a cluster on one or more servers. The Kafka cluster stores streams of records in categories called *topics*.

Each record consists of a key, a value, and a time stamp.

Producers

Web App Mobile App Email App

Topic
1

Topic
2

Kafka Cluster

Producers publish or send messages to a topic.

Consumers subscribe to one or more topics and consume the messages by pulling data from the servers.

Recommendations

Security Monitoring

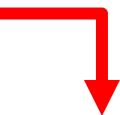
Analytics

Consumer s

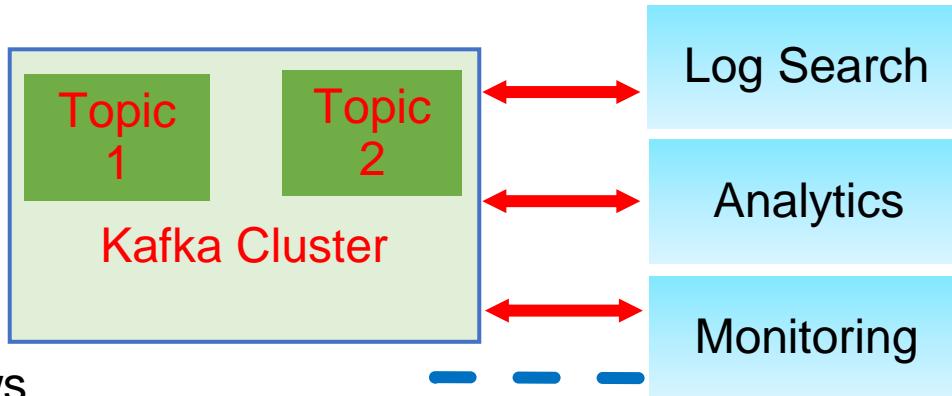
Using Apache Kafka: fenago MoviePlex Examples



Stream of
Movie Views



Example 1: User guest1 watched The Incredibles.



Apache Spark streaming application subscribes to the user activity topic and pulls the event data.

Example 2: Events consumed from a web-metrics topic, when analyzed, point out multiple users using the same sign-in. Check for fraud!

On analysis, consumer makes a special offer:
You get a Superman lunch box!



Additional Resources

- <http://flume.apache.org/index.html>
- <https://kafka.apache.org/>

Summary

- In this lesson, you should have learned to describe:
 - Features and uses of Kafka



Introduction to MapReduce and YARN Processing Frameworks

Course Road Map

Module 1: Big Data Fundamentals

Module 2: Data Acquisition and Storage

Module 3: Data Access and Processing

Module 4: Data Unification

Module 5: Data Analysis

Module 6: Big Data Deployment Options



Lesson 8: Introduction to MapReduce and YARN Processing Frameworks

Lesson 9: Resource Management Using YARN

Lesson 10: Overview of Spark

Lesson 11: Overview of Apache Hive

Lesson 12: Overview of Cloudera Impala

Lesson 13: Using fenago XQuery for Hadoop

Lesson 14: Overview of Solr



Agenda

- MapReduce process
- YARN architecture



Objectives

- After completing this lesson, you should be able to describe the following:
 - Identify the benefits of MapReduce, run a MapReduce job, and monitor the job
 - Review YARN architecture
 - Use YARN to monitor jobs and to manage resources in your Hadoop cluster



Apache Hadoop Core Components

- Apache Hadoop is a system for large-scale and distributed data processing. It has two core components:
 - Distributed storage with HDFS (covered in the lesson titled “Introduction to the HDFS”)
 - Distributed and parallel processing with MapReduce or Spark framework
- MapReduce is a batch-oriented software framework that enables you to write applications that will process large amounts of data in parallel, on large clusters of commodity hardware, and in a reliable and fault-tolerant manner.
- The Apache Spark framework is a cluster-computing platform designed to be fast and general-purpose. It improves the MapReduce functionality (covered later).
- Both MapReduce and Spark are managed by YARN (the acronym for “Yet Another Resource Negotiator”)



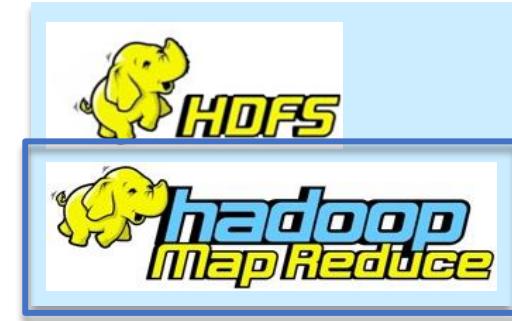
MapReduce Framework: Features

- Integrates with HDFS and provides the same benefits for parallel data processing
- Parallelizes and distributes computations to where the data is stored (data locality)
- The framework:
 - Schedules and monitors tasks, and re-executes failed tasks
 - Hides complex distributed computing tasks from the developer
 - Enables developers to focus on writing the Map and Reduce functions



MapReduce Job

- A MapReduce job is a unit of work requested by a client.
- The job consists of:
 - Input data (usually stored in HDFS)
 - A MapReduce program (user-defined)
- Hadoop runs the job by dividing it into tasks:
 - Map tasks (user-defined)
 - Shuffle and sort tasks (MapReduce)
 - Reduce tasks (user-defined)
- Hadoop divides the input data into fixed-size pieces called *input splits*.
 - Hadoop creates one map task for each input split
 - Each input split runs the user-defined map function for each *record* in the input split

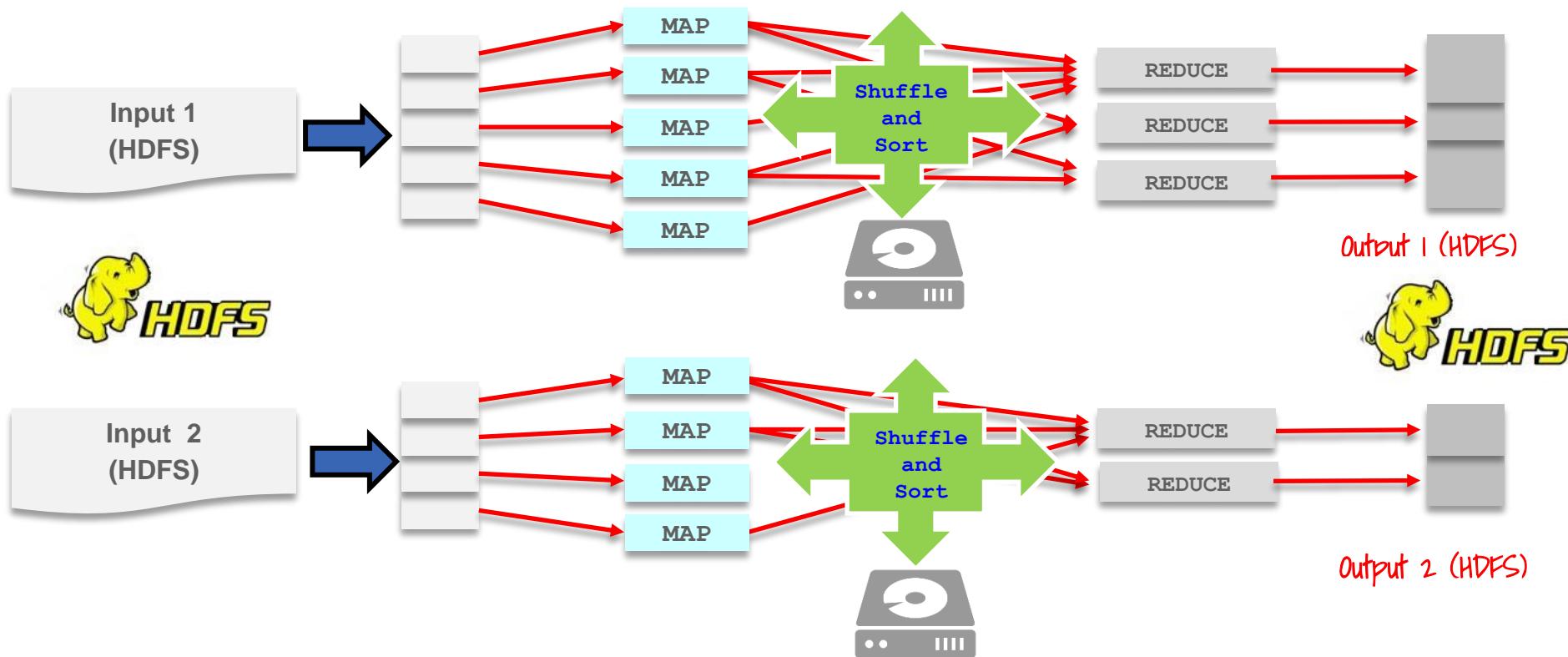


Benefits of MapReduce

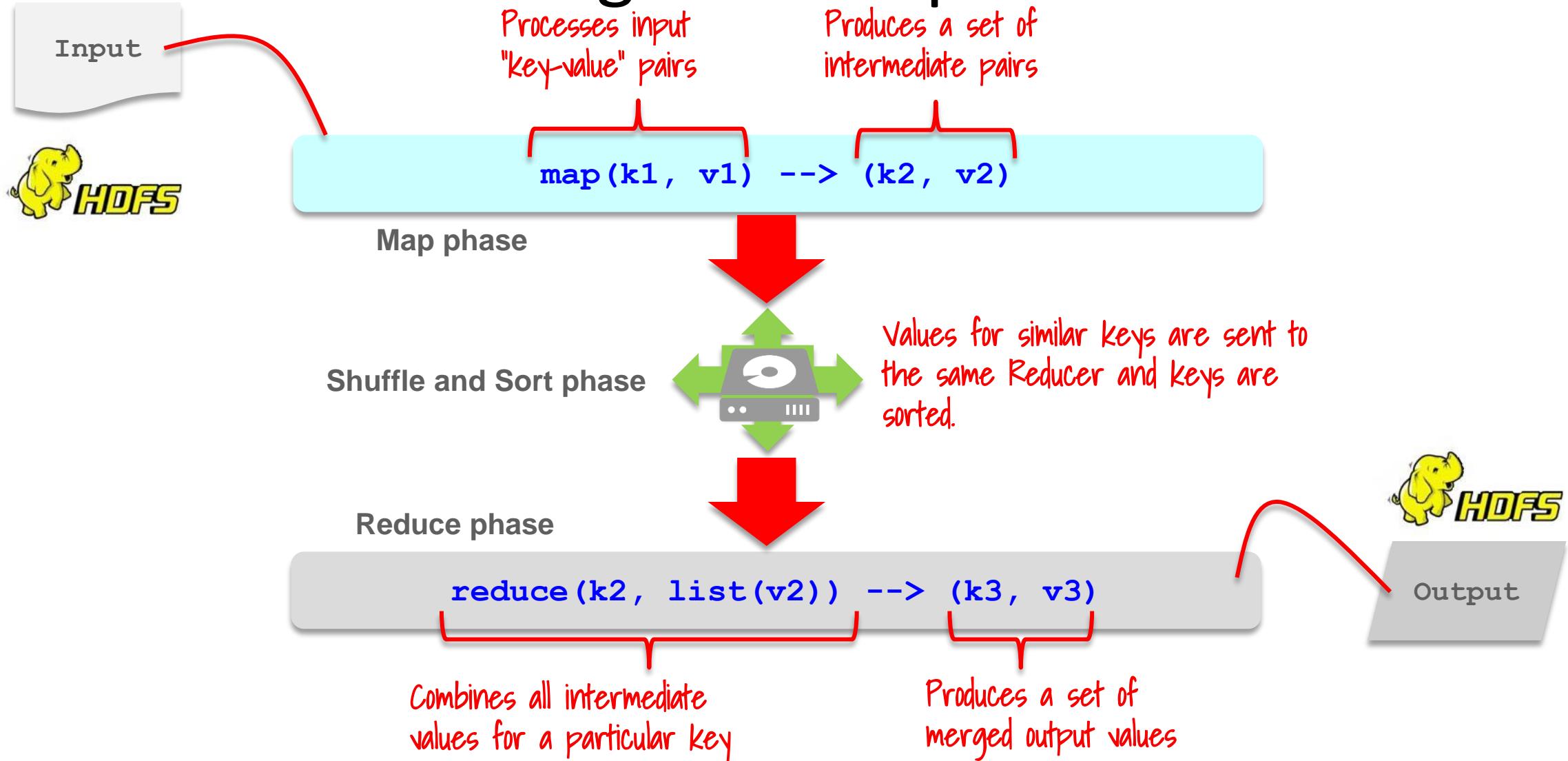
- MapReduce provides:
 - Automatic parallelization and distribution of large data sets that are stored and distributed on Hadoop cluster follower nodes
 - Fault-tolerance
 - I/O scheduling
 - Status and monitoring



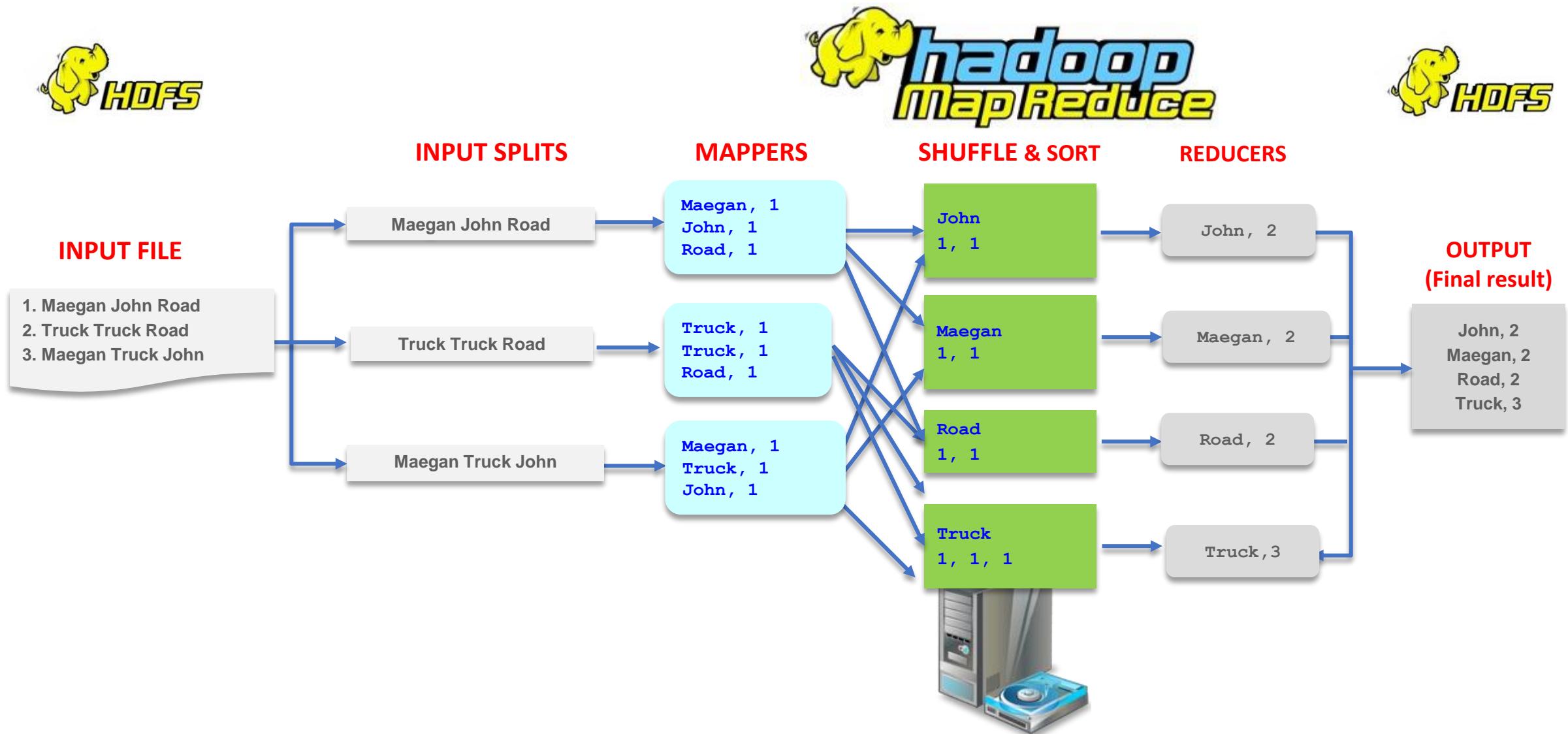
MapReduce Jobs



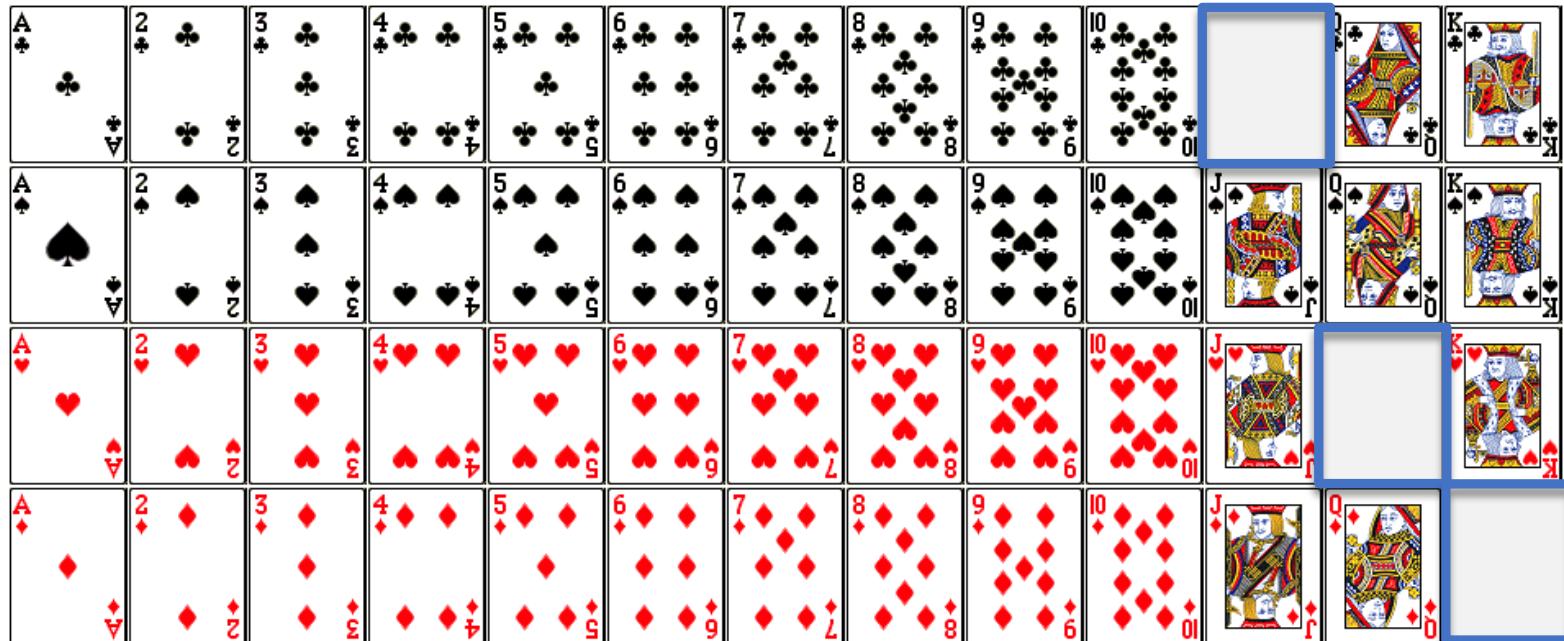
Parallel Processing with MapReduce



Word Count Process: Example 1



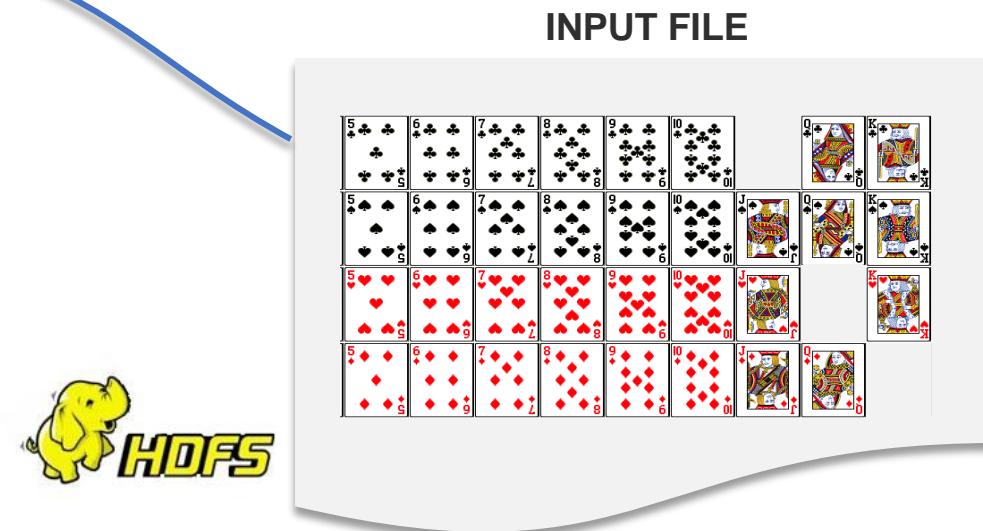
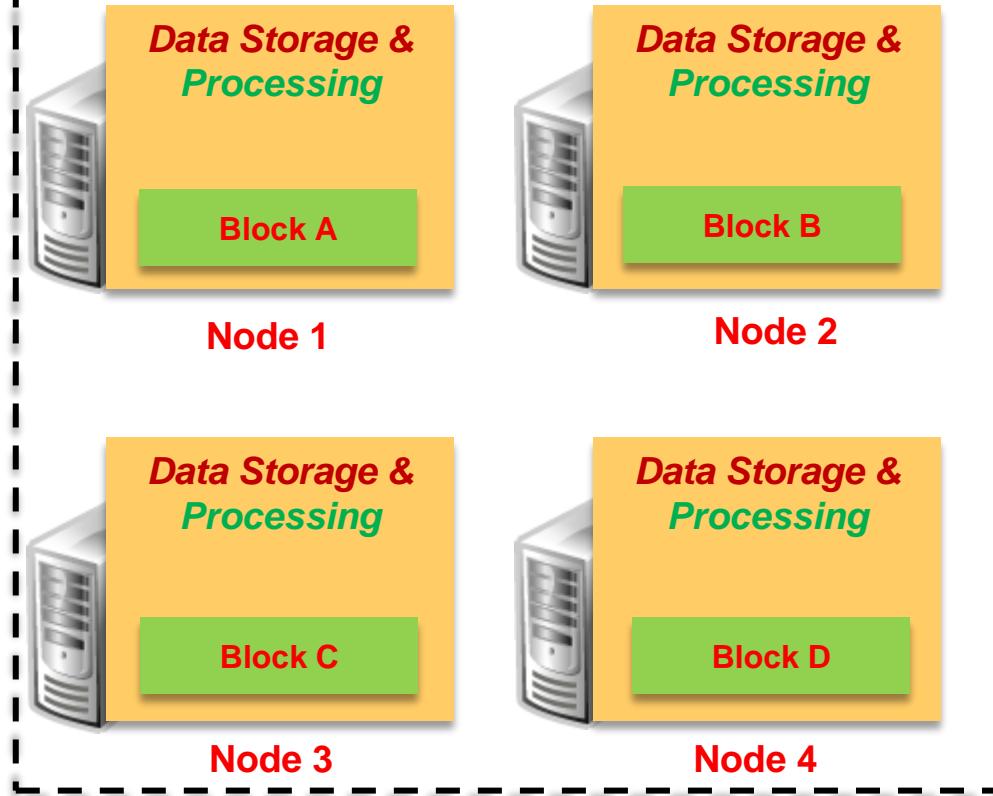
MapReduce Mechanics: Deck of Cards Example



Suppose three face cards are removed from the deck of playing cards.
How do you find the suits whose face cards are short by using MapReduce?

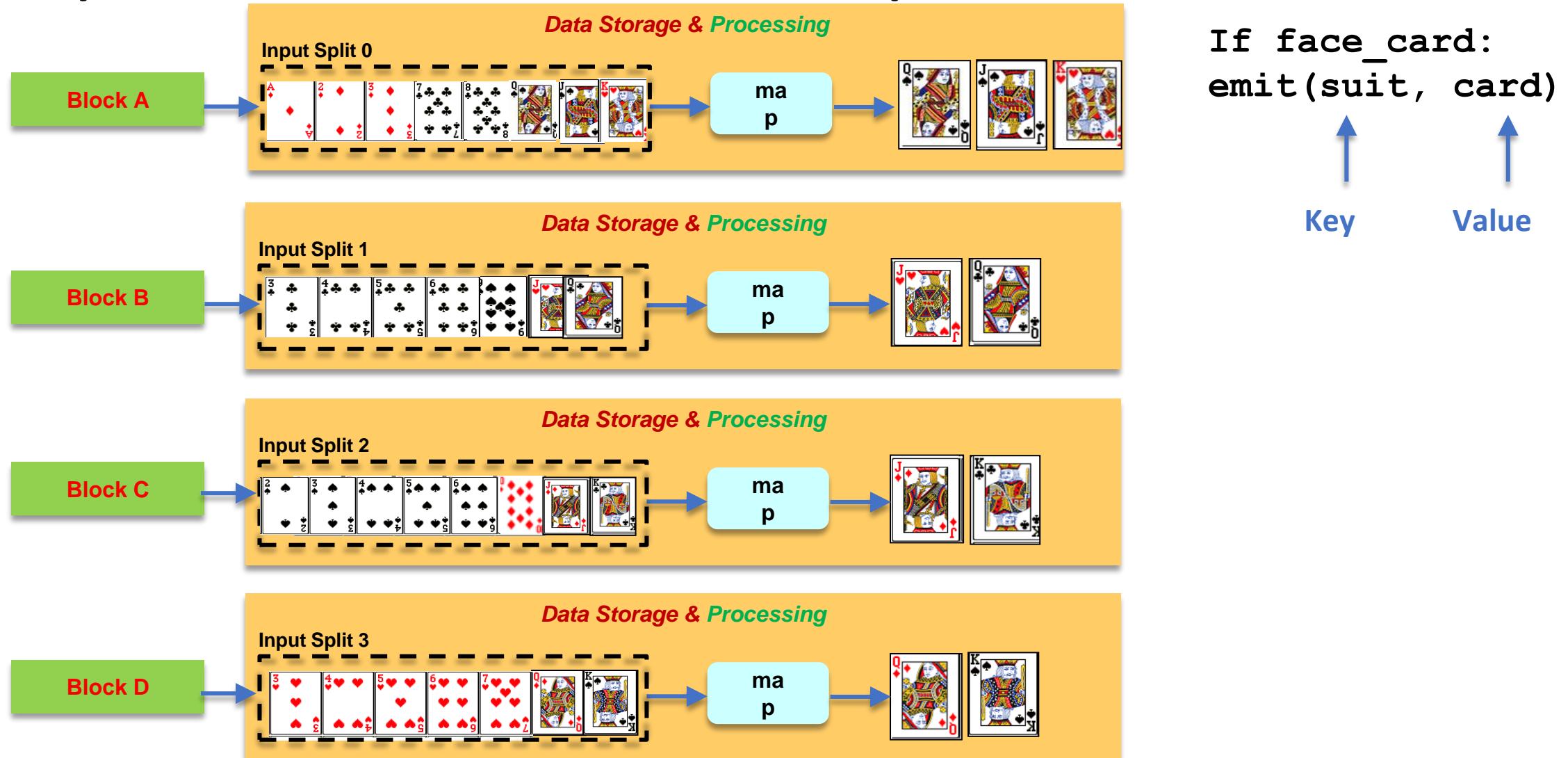


MapReduce Mechanics Example: Assumptions



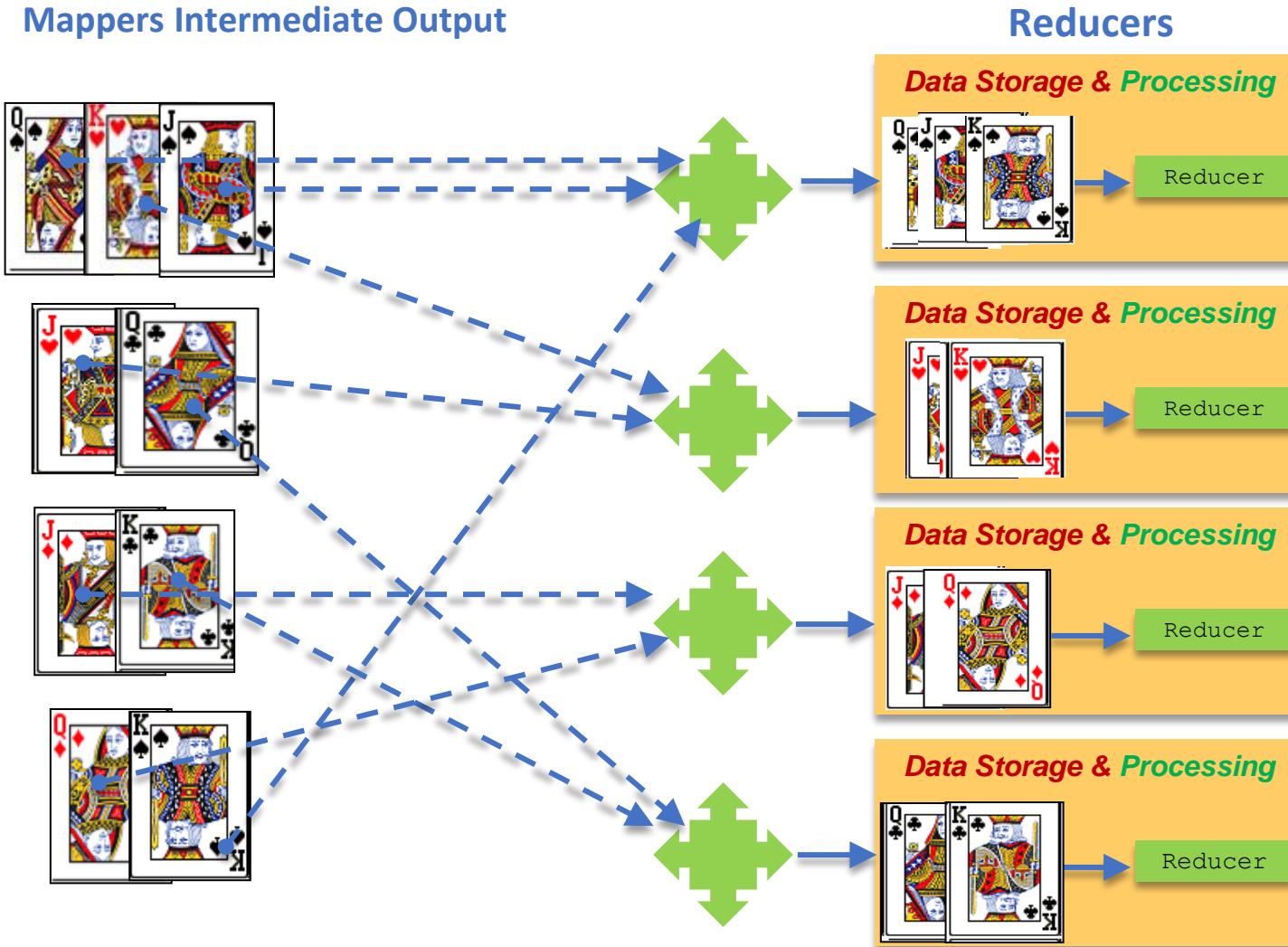
1. The deck of cards was an input file stored in HDFS as blocks on several follower nodes for parallelism and fault-tolerance (four blocks in this example).
2. Nodes operate on their local data.

MapReduce Mechanics: Map Phase

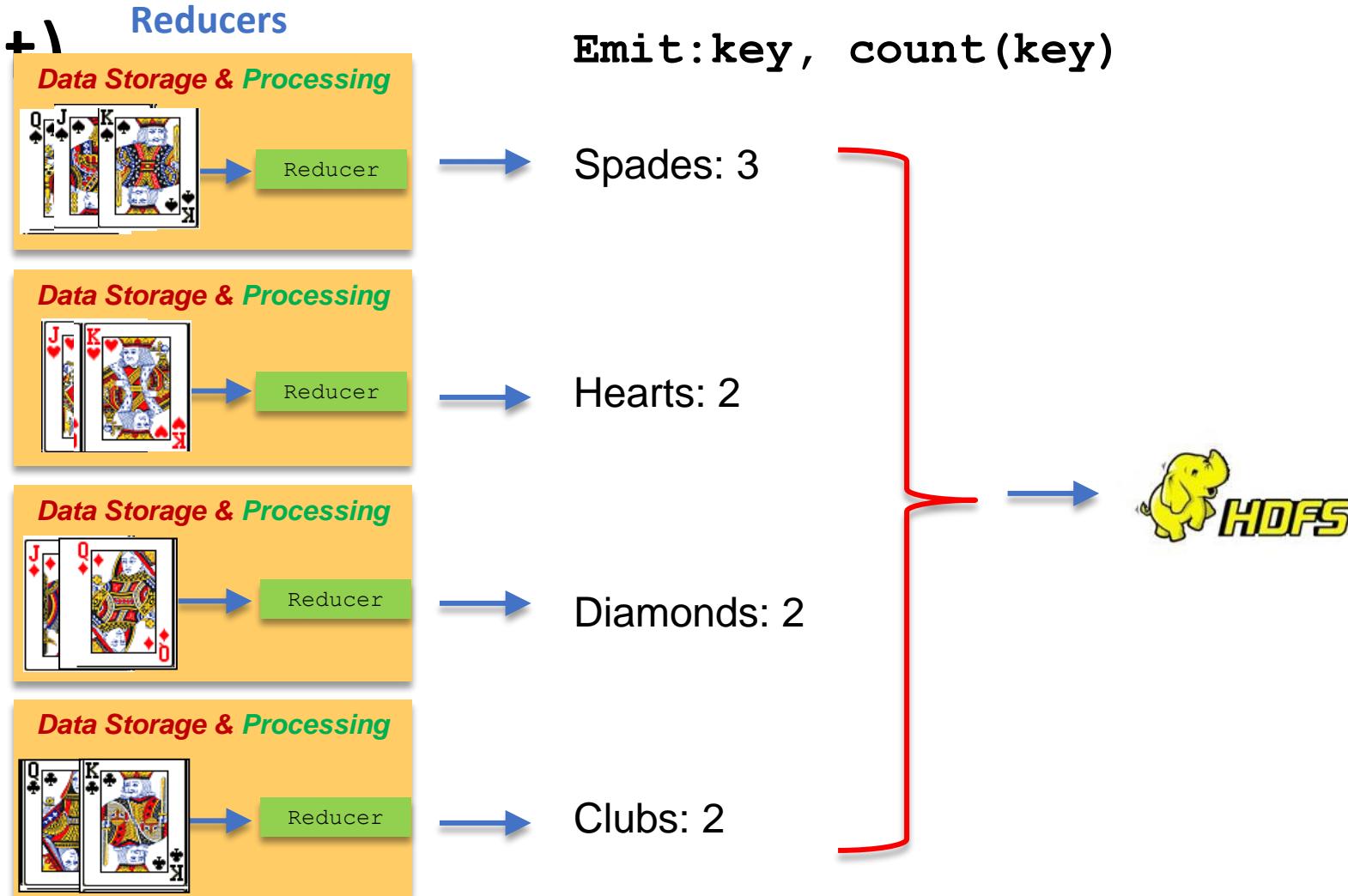


MapReduce Mechanics: Shuffle and Sort Phase

Intermediate data is shuffled and sorted for delivery (stored on local disk) to the reduce tasks.



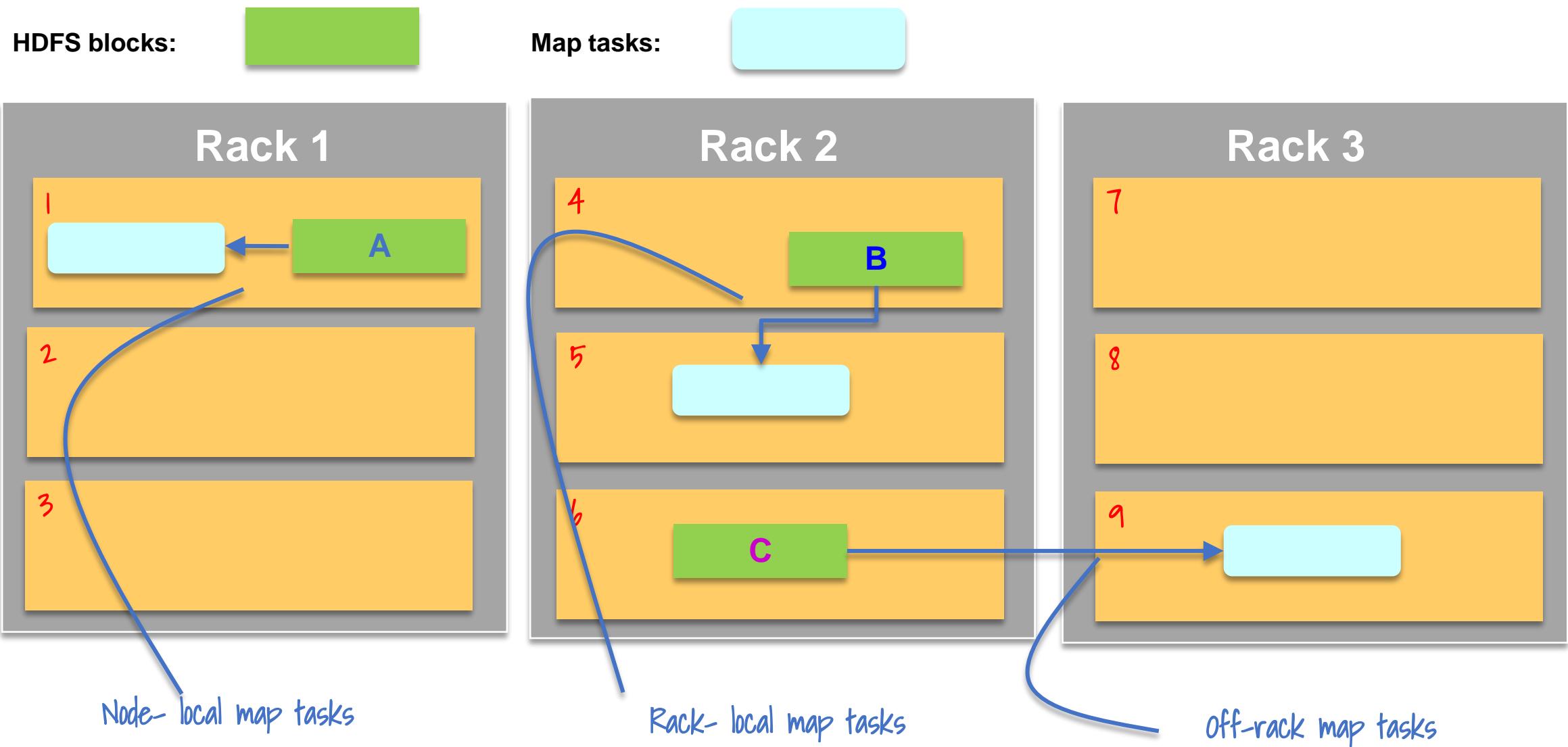
MapReduce Mechanics: Reduce Phase (Result)



Interacting with MapReduce

- MapReduce code can be written in Java, C, and scripting languages.
- Higher-level abstractions (Hive, Pig) enable easy interaction.
 - Optimizers construct MapReduce jobs.
- Code is:
 - Submitted to the JobTracker daemons on the Leader node and executed by the TaskTrackers on the follower nodes with the older MR1.
 - Runs only MapReduce applications
 - MRv1 was the key processing framework prior to YARN.
 - Submitted to the ResourceManager daemon on the Leader node and executed by the ApplicationLeaders on the follower nodes with the YARN (MR2).
 - Runs MapReduce applications, Impala, and Spark
 - Adds the advantage of YARN where ResourceManager is now a scheduler and a resource manager
 - YARN introduced both flexibility and scalability to the processing framework.
 - Most Hadoop implementations use YARN.

Data Locality Optimization in Hadoop



Submitting a MapReduce Job

```
hadoop jar WordCount.jar WordCount  
/user/fenago/wordcount/input  
/user/fenago/wordcount/output
```

Code	Description
<code>hadoop jar</code>	Tells the client to submit job to the JobTracker
<code>WordCount.jar</code>	The JAR file that contains the Map and Reduce code
<code>WordCount</code>	The name of the class that contains the main method where processing starts
<code>/user/fenago/wordcount/input</code>	The input directory
<code>/user/fenago/wordcount/output</code>	A single HDFS output path. All final output will be written to this directory.

Submitting a WordCount MapReduce Job: Reviewing the Input Data Files

```
cat file01 file02
```

```
[oracle@bigdatalite wordCount]$ cat file01 file02
very disappointed and very expensive
expensive and unreliable insurance
worthless insurance and expensive
worst customer service
worst insurance company
worst professional staff and unreliable insurance company
insurance is very expensive
worst insurance cover
terrible service
disappointed with the expensive insurance service
worthless and expensive
awful customer service
terrible worst service
worst bank and worst customer service
worst insurance
disappointed with protocols
unreliable insurance
best service I recommend it
good professionals and efficient insurance
I will recommend it
good customer service
best insurance I found I recommend it[oracle@bigdatalite wordCount]$
```

Submitting the WordCount MapReduce Job

```
hadoop jar WordCount.jar WordCount \
/usr/fenago/wordcount/input \
/usr/fenago/wordcount/output
```



```
oracle@bigdatalite:~/exercises/wordCount
File Edit View Search Terminal Help
input/file02
[oracle@bigdatalite wordCount]$ hadoop jar WordCount.jar WordCount /user/oracle/wordco
unt/input /user/oracle/wordcount/output
16/08/24 12:23:33 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/08/24 12:23:36 INFO input.FileInputFormat: Total input paths to process : 2
16/08/24 12:23:37 INFO mapreduce.JobSubmitter: number of splits:2
16/08/24 12:23:37 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_14720555
09950_0001
16/08/24 12:23:39 INFO impl.YarnClientImpl: Submitted application application_14720555
09950_0001
16/08/24 12:23:40 INFO mapreduce.Job: The url to track the job: http://bigdatalite.loc
aldomain:8088/proxy/application_1472055509950_0001
16/08/24 12:23:40 INFO mapreduce.Job: Running job: job_1472055509950_0001
16/08/24 12:23:55 INFO mapreduce.Job: Job job_1472055509950_0001 running in uber mode
: false
16/08/24 12:23:55 INFO mapreduce.Job: map 0% reduce 0%
16/08/24 12:24:07 INFO mapreduce.Job: map 100% reduce 0%
16/08/24 12:24:20 INFO mapreduce.Job: map 100% reduce 100%
16/08/24 12:24:21 INFO mapreduce.Job: Job job_1472055509950_0001 completed successfull
y
16/08/24 12:24:22 INFO mapreduce.Job: Counters: 49
    File System Counters
        FILE: Number of bytes read=1764
        FILE: Number of bytes written=347480
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=1306
        HDFS: Number of bytes written=208
        HDFS: Number of read operations=9
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
    Job Counters
        Launched map tasks=2
        Launched reduce tasks=1
        Data-local map tasks=2
        Total time spent by all maps in occupied slots (ms)=18420
        Total time spent by all reduces in occupied slots (ms)=10267
        Total time spent by all map tasks (ms)=18420
```

Monitoring MapReduce Jobs by Using the YARN Resource Manager Web UI

<http://localhost:8088/cluster>

The screenshot shows the 'All Applications' page of the Hadoop YARN Resource Manager Web UI. At the top, there's a navigation bar with tabs like 'Most Visited', 'Hue', 'MoviePie', 'Hadoop', 'Spatial and Graph', 'BDD', 'Apex', 'ORDS Lab', 'Solr Admin', 'Pattern Matching', and 'Cloudera Manager'. A red box highlights the 'Hadoop' tab, which is currently active. Below the navigation bar, there's a logo for 'hadoop' and a dropdown menu showing 'NameNode bigdatalite.localdomain:8020', 'YARN Applications' (which is also highlighted with a yellow box), and 'JobHistory'. The main content area is titled 'All Applications' and contains several sections: 'Cluster Metrics' (with tables for Apps Submitted, Apps Pending, Apps Running, Apps Completed, Containers Running, Memory Used, Memory Total, Vcores Used, Vcores Total, Vcores Reserved, Active Nodes, and Duration); 'User Metrics for dr.who' (with tables for Apps Submitted, Apps Pending, Apps Running, Apps Completed, Containers Running, Containers Pending, Containers Reserved, Memory Used, Memory Pending, and Memory Pending); and a large table listing applications. This table has columns for ID (application_1472055509950_0001), User (oracle), Name (WordCount), Application Type (MAPREDUCE), Queue (root.oracle), Start Time (Wed Aug 24 12:23:39 -0400 2016), Finish Time (Wed Aug 24 12:24:19 -0400 2016), State (FINISHED), Final Status (SUCCEEDED), and Run Cont (N/A). A red box also highlights this application row in the table. At the bottom of the table, it says 'Showing 1 to 1 of 1 entries'.

YARN (covered later in this lesson) is used to manage resources on a Hadoop cluster. In this example, YARN is managing a MapReduce job, but can also be used to monitor Spark jobs.

Monitoring MapReduce Jobs by Using the JobHistory Server Web UI

<http://bigdatalite.localdomain:19888/jobhistory>

MapReduce JobHistoryServer archives jobs' metrics and can be accessed through the JobHistory Web UI or Hue.

The screenshot shows the JobHistory Web UI interface. At the top, there's a navigation bar with links like 'Most Visited', 'Hue', 'MoviePie', 'Hadoop' (which is highlighted in green), 'Spatial and Graph', 'BDD D', 'Apex', 'ORDS Lab', 'Solr Admin', 'SQL Pattern Matching', and 'Cloudera Manager'. Below the navigation bar, there's a logo for 'hadoop' and a sidebar with 'Application' dropdowns for 'About Jobs' and 'Tools'. The main content area is titled 'JobHistory' and displays a table titled 'Retired Jobs'. The table has columns for 'Submit Time', 'Start Time', 'Finish Time', 'Job ID', 'Name', 'User', 'Queue', 'State', 'Maps Total', 'Maps Completed', and 'Reduces Total'. A single row is shown, corresponding to the job ID 'job_1472055509950_0001'. At the bottom of the table, there are search and filter fields, and at the very bottom, a message 'Showing 1 to 1 of 1 entries'.

Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total
2016.08.24 12:23:39 EDT	Wed Aug 24 EDT	2016.08.24 12:24:19 EDT	job_1472055509950_0001	WordCount	oracle	root.oracle	SUCCEEDED	2	2	1

Viewing the WordCount.java Program

The image displays two terminal windows from a Linux system named 'bigdatalite'. The top window shows the command `hadoop fs -ls /user/oracle/wordcount/output` being run, which lists two files: `SUCCESS` and `part-r-00000`. The bottom window shows the command `hadoop fs -cat /user/oracle/wordcount/output/part-r-0000` being run, which displays the word count output. The output is as follows:

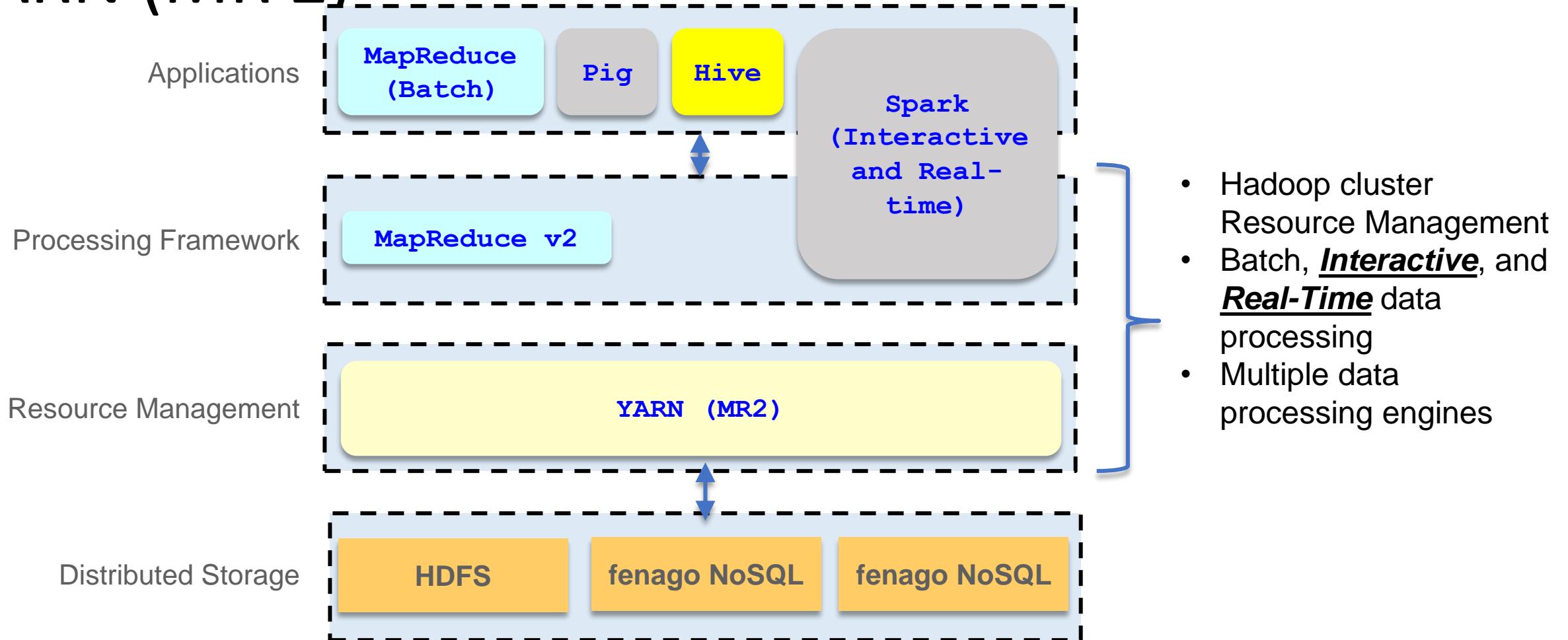
```
0
and    12
awful   2
bank    2
company 4
cover   2
customer 6
disappointed 6
expensive 12
insurance 18
is      2
professional 2
protocols 2
service 12
staff   2
terrible  4
the     2
unreliable 6
very    6
with    4
worst   16
worthless 4
```

Agenda

- MapReduce process
- YARN architecture



Running Applications Starting with Hadoop 2.x with YARN (MR 2)



YARN Architecture

- YARN splits up the two major functionalities of the JobTracker, resource management and job scheduling/monitoring, into separate new daemons.
 - A global ResourceManager (RM)
 - A per-application ApplicationLeader (AM)
- The single RM and NodeManager (NM) on each follower node, form the data-computation framework
- The RM is the ultimate authority that arbitrates resources among all the applications in the system.
- The per-application AM negotiates resources from the RM and works with the NodeManager(s) to execute and monitor the tasks.

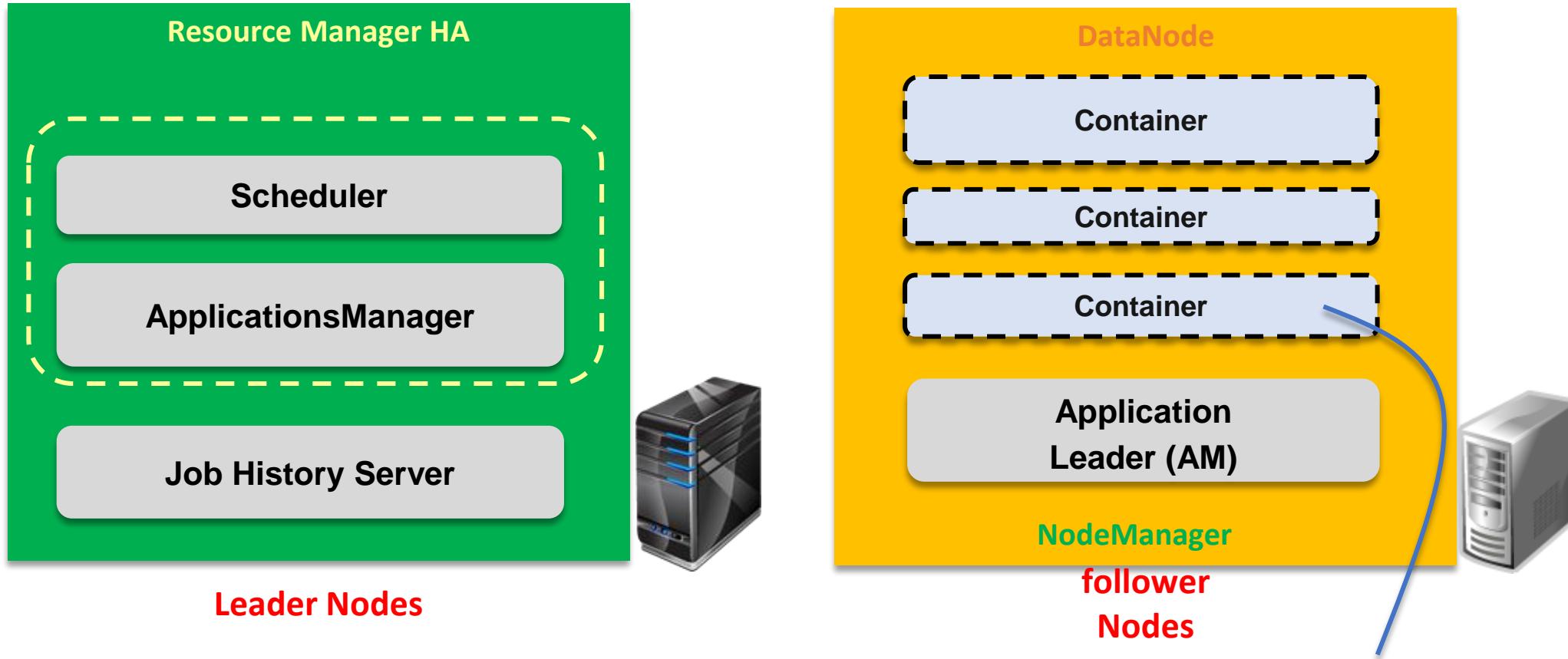
YARN: Features

- Scalability
- Compatibility with MapReduce
- Improved cluster utilization
- Support for workloads other than MapReduce

YARN Daemons

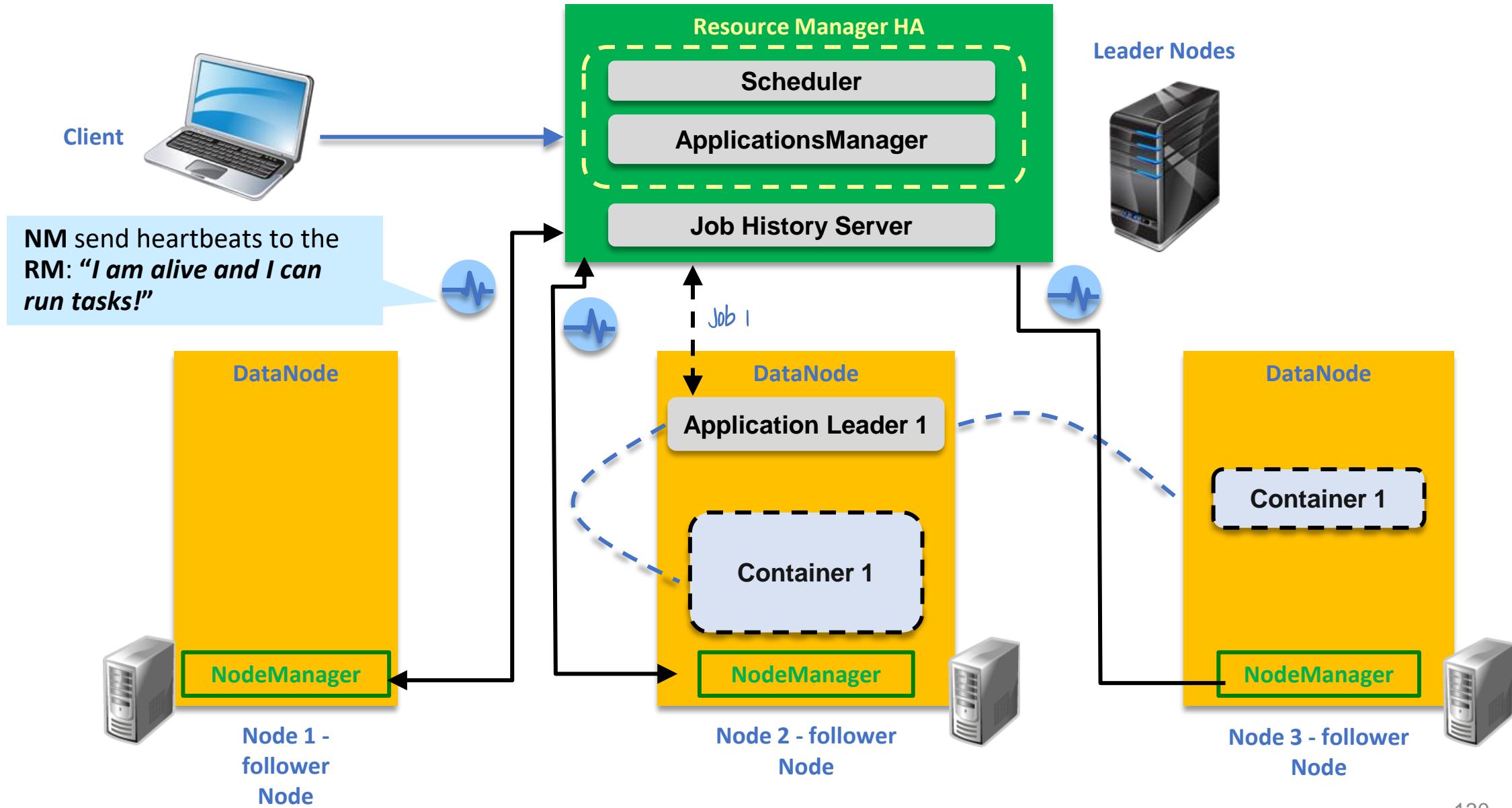
Component	Description
ResourceManager (RM)	<ul style="list-style-type: none">• A dedicated scheduler that allocates resources to the requesting applications• The RM has two main components: Scheduler and ApplicationsManager• It is a critical component of the cluster and runs on a dedicated Leader node.
NodeManager (NM)	<ul style="list-style-type: none">• Each follower node in the cluster has an NM daemon, which acts as a follower for the RM.• Each NM tracks the available data processing resources and usage (cpu, memory, disk, network) on its follower node and sends regular reports to the RM.
ApplicationLeader (AM)	<ul style="list-style-type: none">• The per-application AM is responsible for negotiating resources from the RM and working with the NM(s) to execute and monitor the tasks.• Runs on a follower node
Container	<ul style="list-style-type: none">• A container is a collection of all the resources necessary to run an application: CPU, memory, network bandwidth, and disk space.• Runs on a follower node in a cluster.
Job History Server	<ul style="list-style-type: none">• Archives jobs and metadata

YARN Architecture

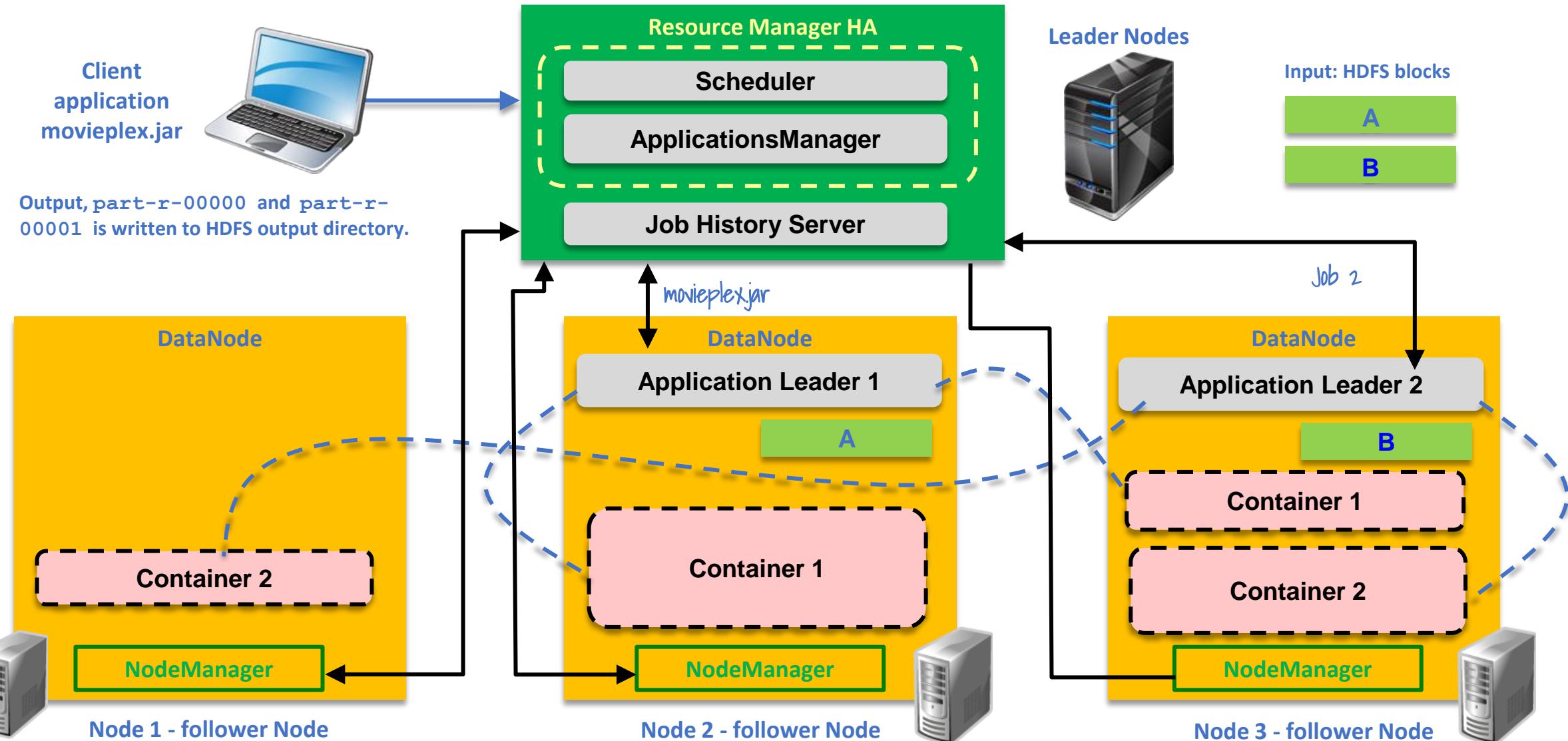


Application Leader Containers specifications use the physical world approach such as CPU and cores, memory, disk space, and network Bandwidth (instead Of the Map and Reduce slots in MapReduce V1)

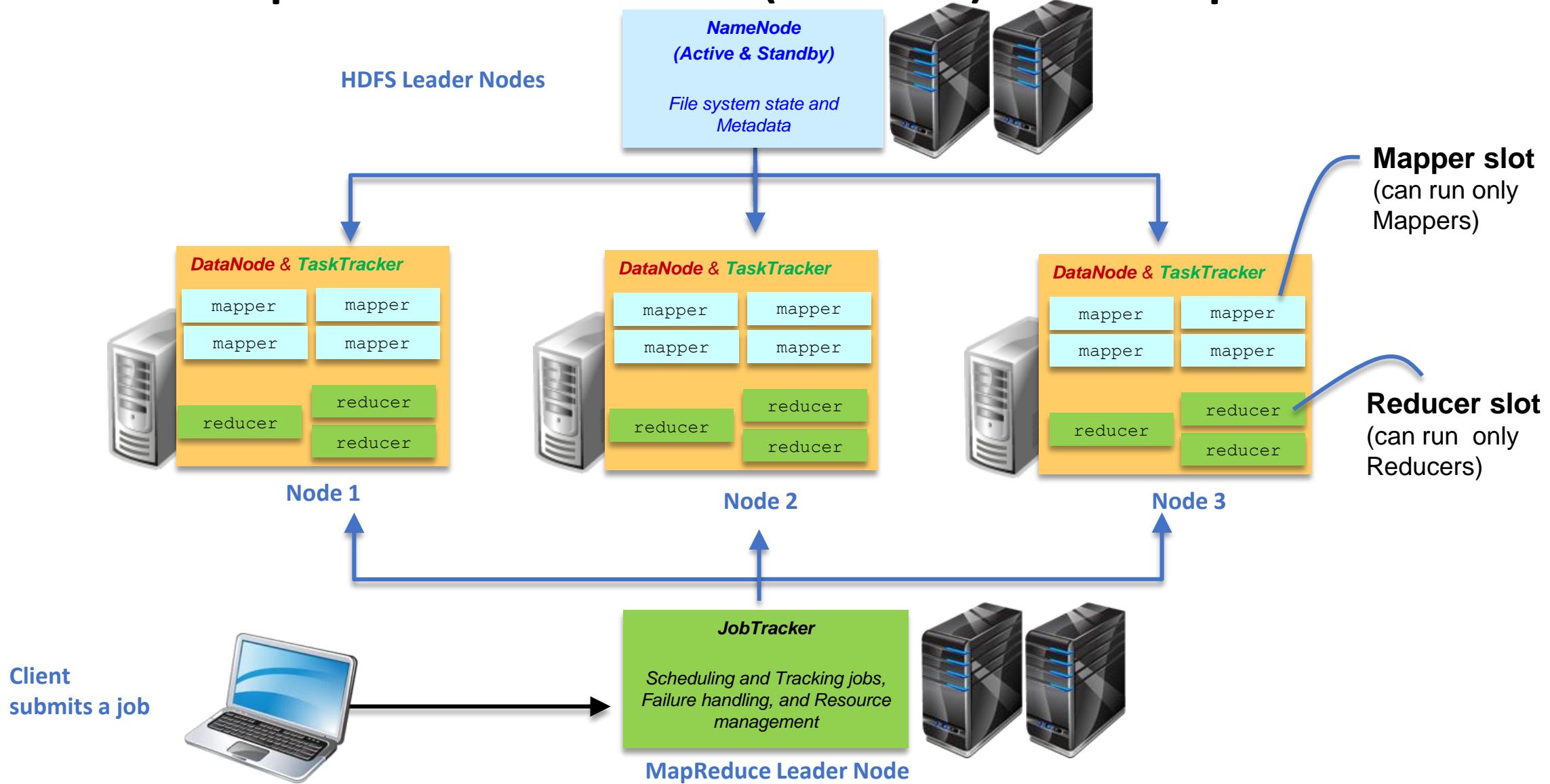
YARN (MRv2) Architecture



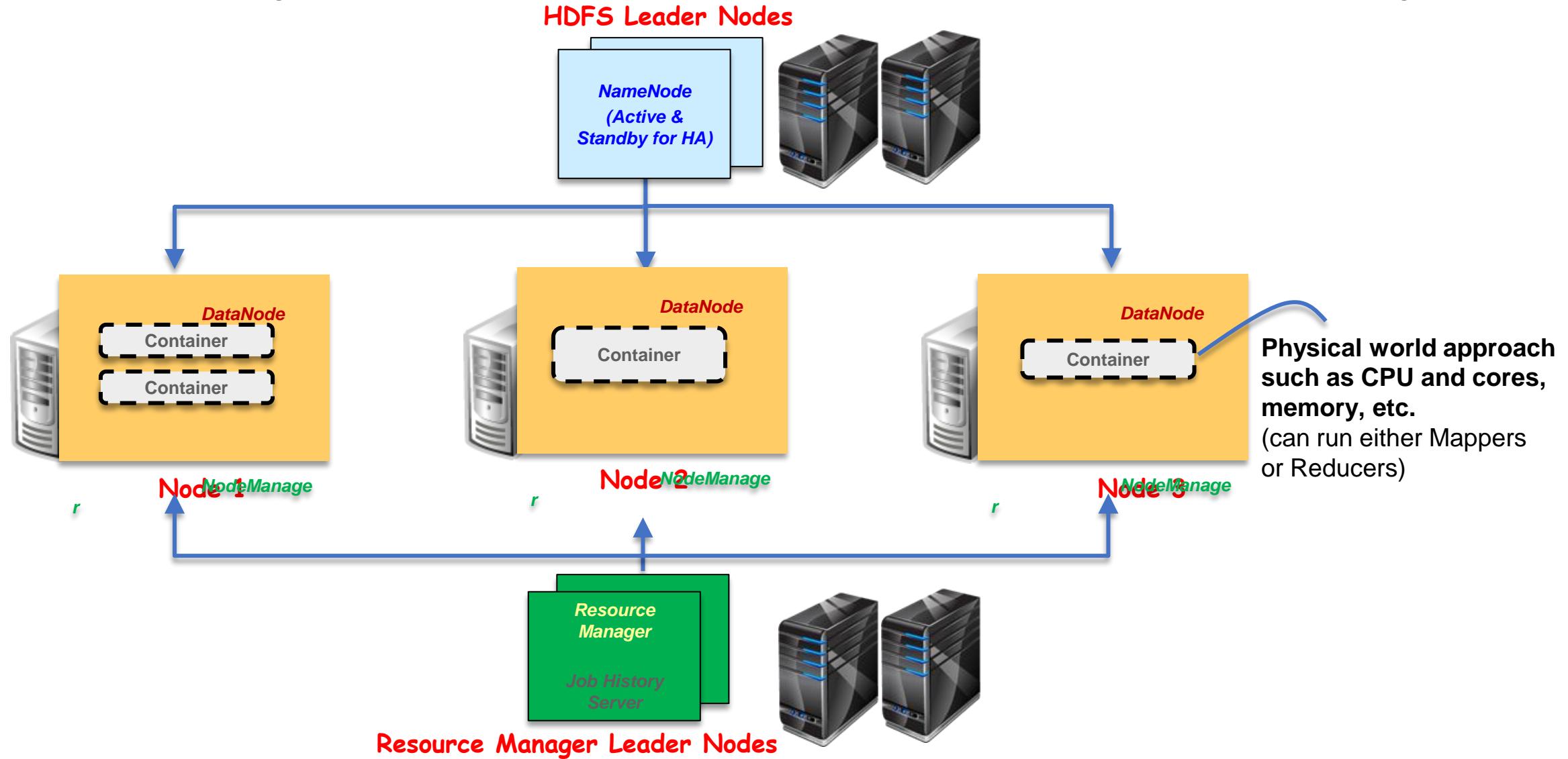
YARN Application Workflow



Hadoop Basic Cluster (MRv1): Example



Hadoop Basic Cluster YARN (MRv2): Example



Summary

- In this lesson, you should have learned how to:
 - Identify the benefits of MapReduce, run a MapReduce job, and monitor the job
 - Identify the MapReduce version 1 and YARN (MR2) architectures
 - Use YARN to monitor jobs and to manage resources in your Hadoop cluster



Practice 8: Overview

- In these practices, you:
 - Compile a WordCount.java program, which runs on a Hadoop Cluster
 - Upload the files on which you run the WordCount into the Hadoop Distributed File System (HDFS)
 - Run the WordCount.java program and view the results
 - Monitor a job with the Resource Manager and JobHistory Web UIs

Overview of Spark

Course Road Map

Module 1: Big Data Fundamentals

Module 2: Data Acquisition and Storage

Module 3: Data Access and Processing

Module 4: Data Unification

Module 5: Data Analysis

Module 6: Big Data Deployment Options



Lesson 8: Introduction to MapReduce and YARN Processing Frameworks

Lesson 9: Resource Management Using YARN

Lesson 10: Overview of Spark

Lesson 11: Overview of Apache Hive

Lesson 12: Overview of Cloudera Impala

Lesson 13: Using fenago XQuery for Hadoop

Lesson 14: Overview of Solr



Objectives

- After completing this lesson, you should be able to:
 - Describe Spark and identify its benefits
 - Identify Spark's components and architecture
 - Interact with Spark
 - Run a Spark job and monitor the job by using YARN's Resource Manager Web UI



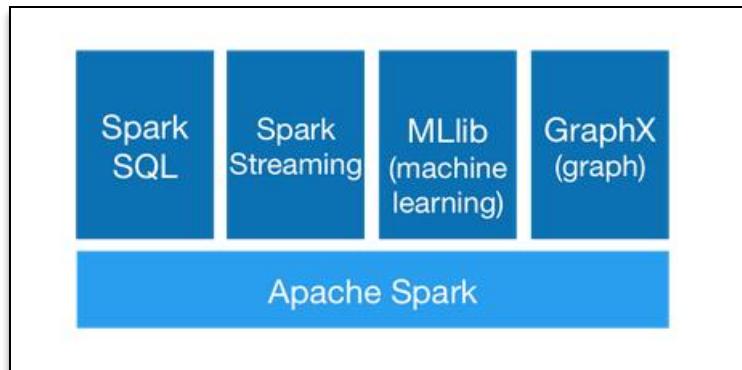
What Is Apache Spark?

- A fast and general engine for large-scale (and multi-stage processing routines) data processing
- An open-source, parallel data-processing framework with a proven scalability
- Makes it easy to develop fast, unified big data applications combining batch, streaming, and interactive analytics on all your data
- Hides complex distributed computing tasks from the developer so that they can focus on writing the Map and Reduce functions (abstraction layer)
- An alternative to using traditional MapReduce on Hadoop, which does not work well for interactive queries or real-time, low-latency applications
 - A major disadvantage of Hadoop's MapReduce implementation was its persistence of intermediate data to disk between the Map and Reduce processing phases.



Benefits of Using Spark

- Speed (much faster than MapReduce)
- Ease of use (higher level of abstraction than MapReduce)
- Sophisticated analytics
- Runs on various cluster managers: YARN, stand-alone (built-in) scheduler, and Mesos
- A cross-platform application capable of running on Windows and Linux
- Native integration with programming interfaces such as Java, Python, Scala, SQL, and R



Source: Apache Spark page on <http://spark.apache.org/>

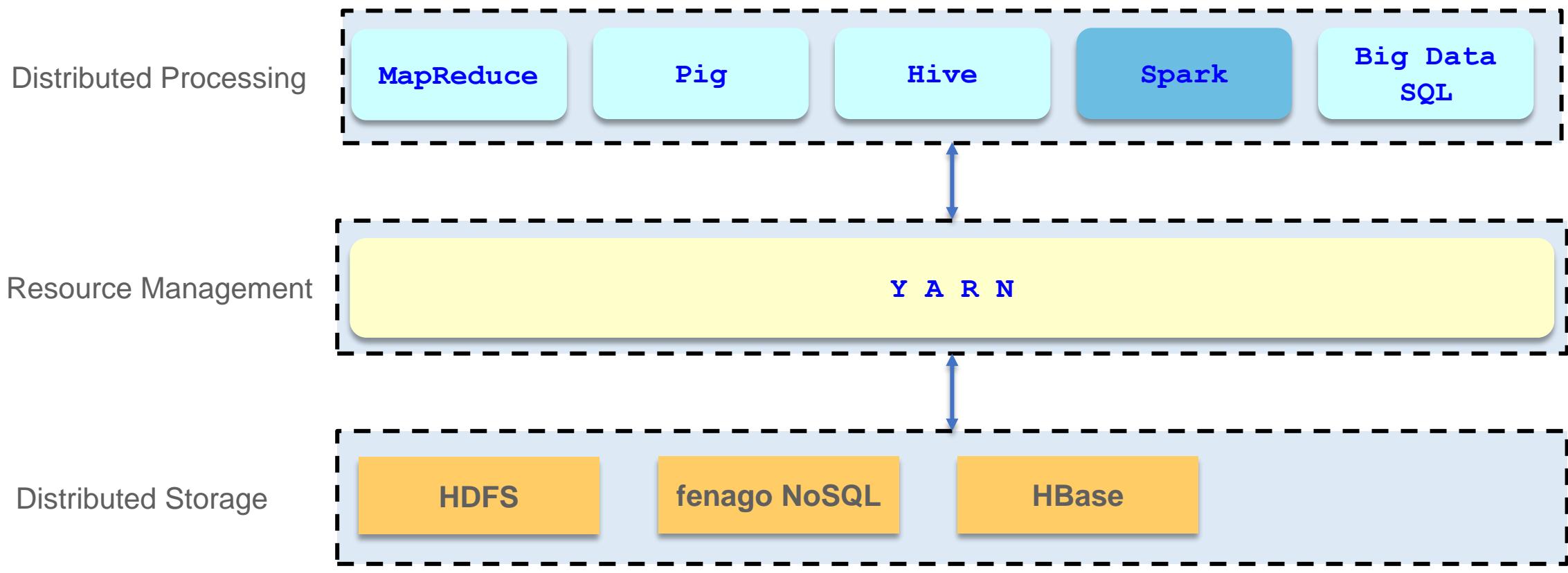
Spark Versus MapReduce

Apache Spark	Hadoop MapReduce
Spark stores data in memory.	MapReduce stores data on disk.
Spark can be used to perform streaming, batch processing, and machine learning all in the same cluster.	MapReduce is used for generating reports that help find answers to historical queries. It performs only batch processing.
Spark is written in Scala.	MapReduce is written in Java.
Spark executes jobs 10 to 100 times faster than Hadoop MapReduce.	Hadoop MapReduce doesn't leverage the memory of the Hadoop cluster to the maximum.
Spark is easy to program.	MapReduce is difficult to program.

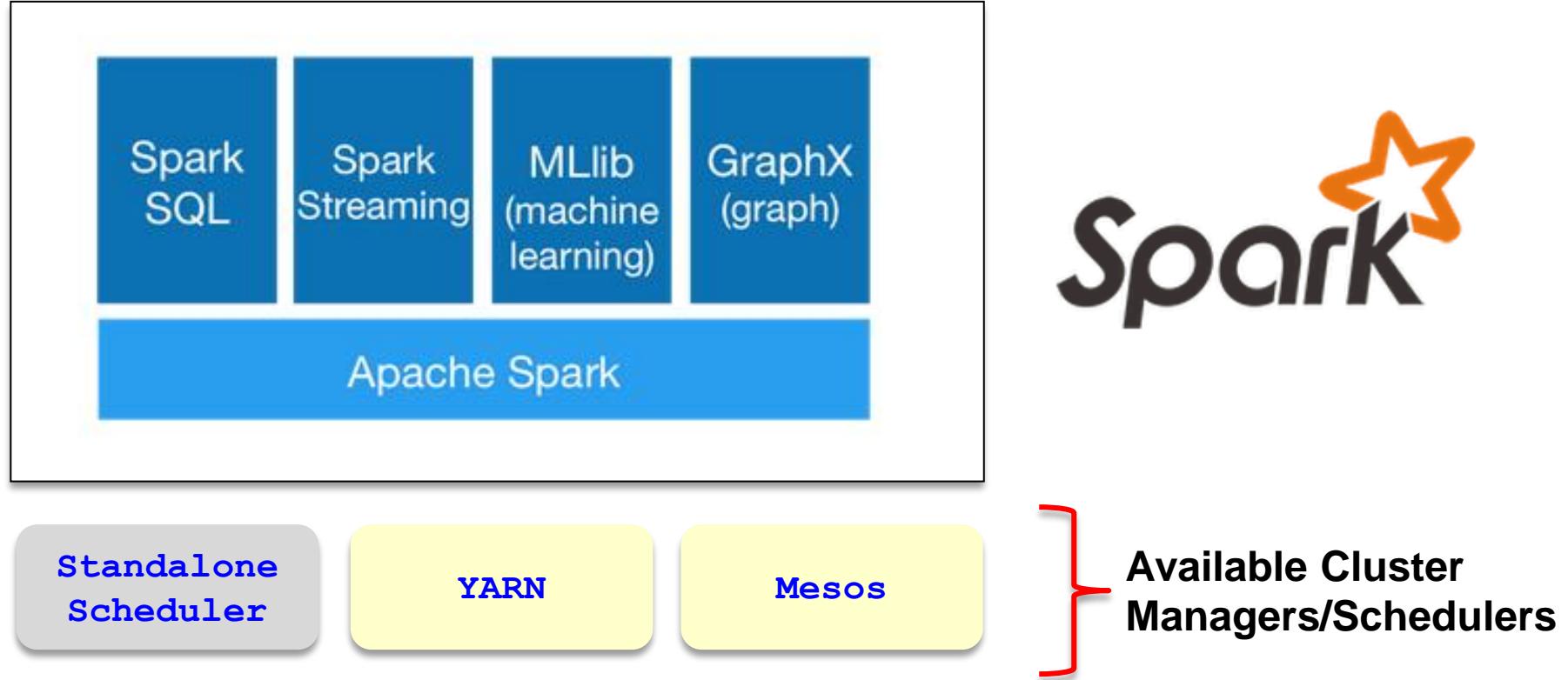
How is Spark Used?

- Real-time data analysis
 - Fraud detection
 - Website monitoring
 - Network monitoring
 - Web clicks and advertising
- Customer segmentation for marketing purposes, and sentiment analysis
- Machine learning and advanced analytics

Spark Architecture

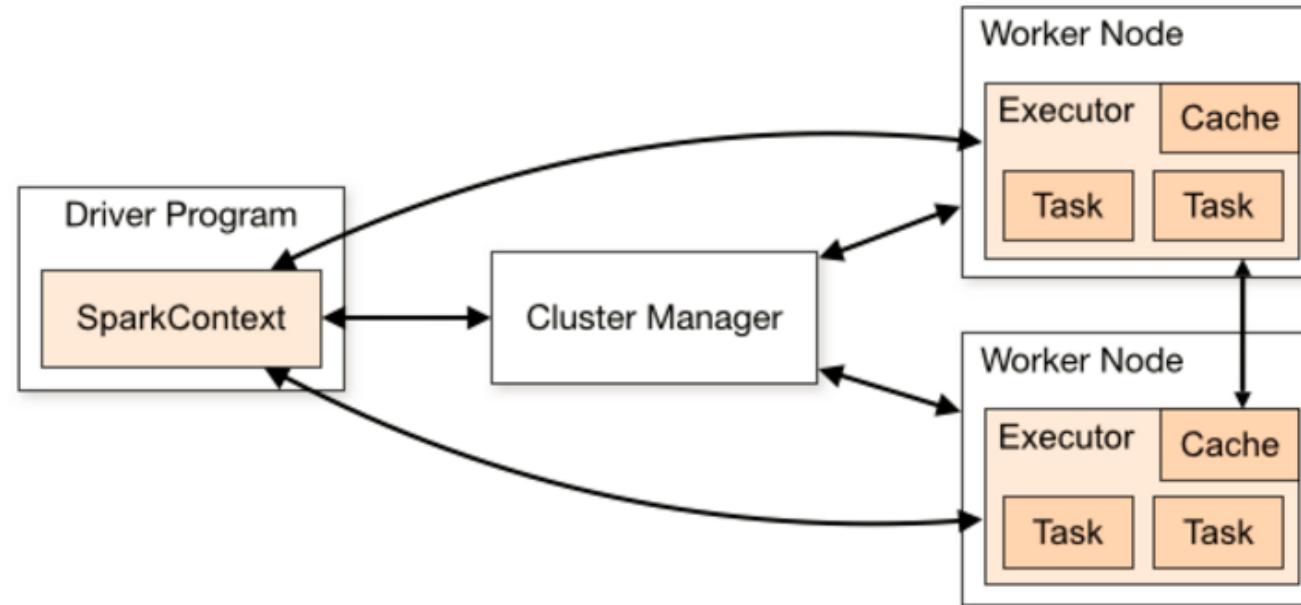


Spark Architecture



Spark Application Components

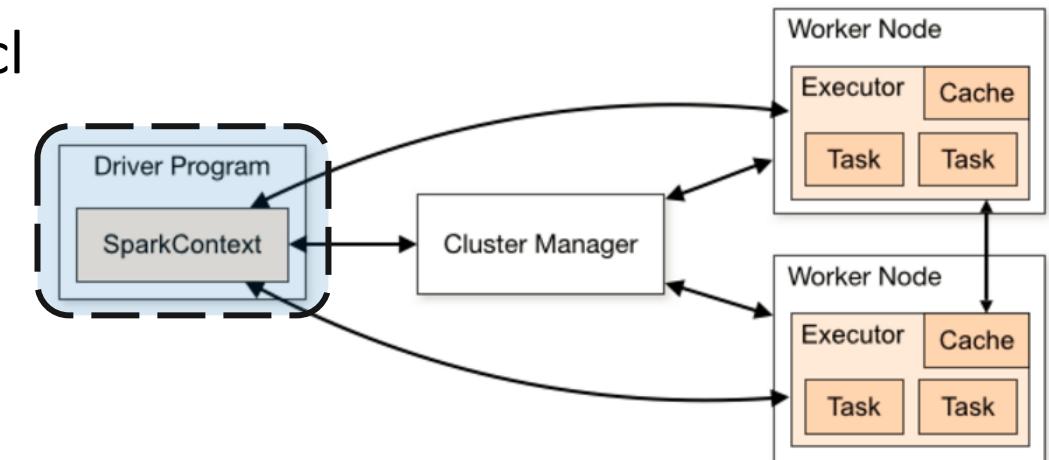
- Spark Driver
- Spark Leader
- Cluster Manager
- Executors



Source: Apache Spark page on <http://spark.apache.org/docs/latest/cluster-overview.html>

Spark Driver

- Runs in the Application Leader container (if you are using YARN)
- Creates the `SparkContext`, which is the application instance representing the connection to the Spark Leader and Spark worker or follower nodes in the cluster
- Plans, schedules, and monitors the Spark application
- Hadoop injects the Spark code into the driver (similar to what happens in MapReduce)
- Controls the job or the interactive query ~~that you are running~~
- Returns the application's results to the cl



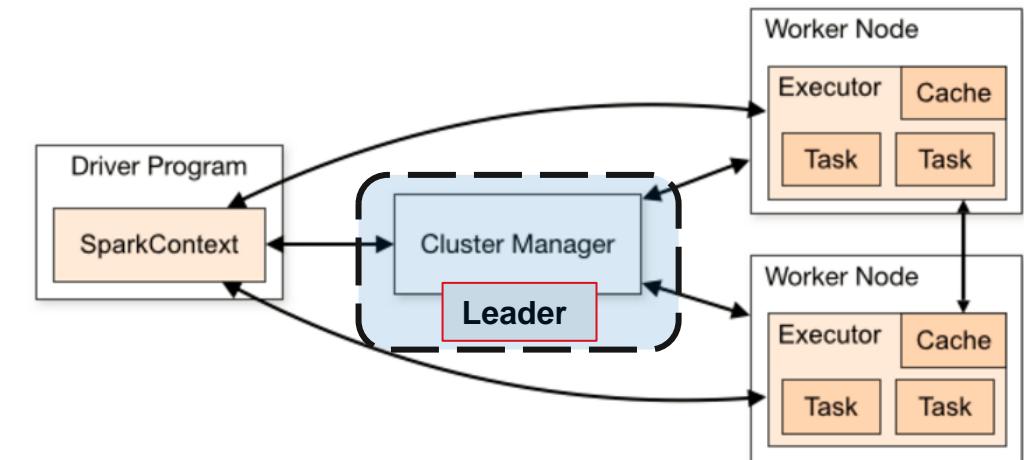
Spark Driver

The driver:

- Takes the input and plans the execution of the Spark program
- Uses all of the requested transformations (data manipulation operations) and actions (requests for output or a prompt to execute the program) and creates a directed acyclic graph (DAG).
 - The DAG consists of *tasks* and *stages*.
 - Tasks are the smallest unit of schedulable work in a Spark program.
 - Stages are sets of tasks that can be run together. Stages are dependent on one another.
 - Coordinates the running of stages and tasks defined in the DAG
-

Spark Leader

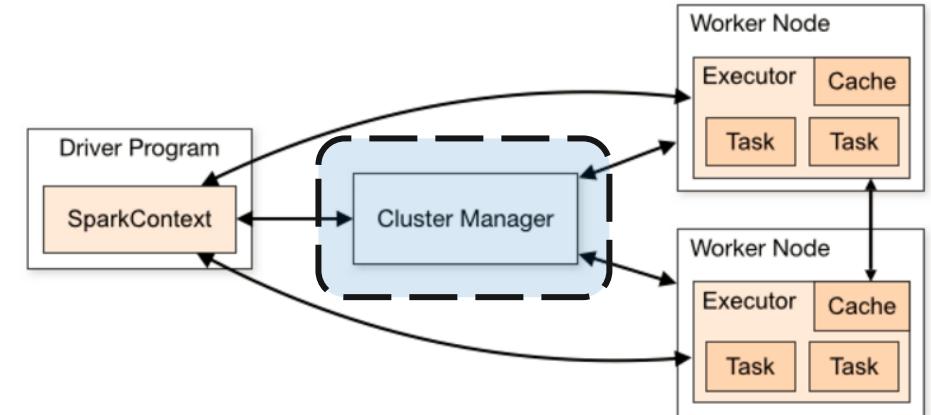
- The Spark Leader and the cluster manager are the central processes that monitor, reserve, and allocate the distributed cluster resources (containers in the case of YARN) on which the executors run.
- The Spark Leader and the cluster manager can be separate processes or they can be combined into one process, as is the case when running Spark in Standalone mode.
- The Spark Leader is the process that requests resources in the cluster and makes these available to the Spark driver.
- In either deployment mode, the Leader negotiates resources or containers with worker nodes and tracks their status and monitors their progress.



Cluster Manager

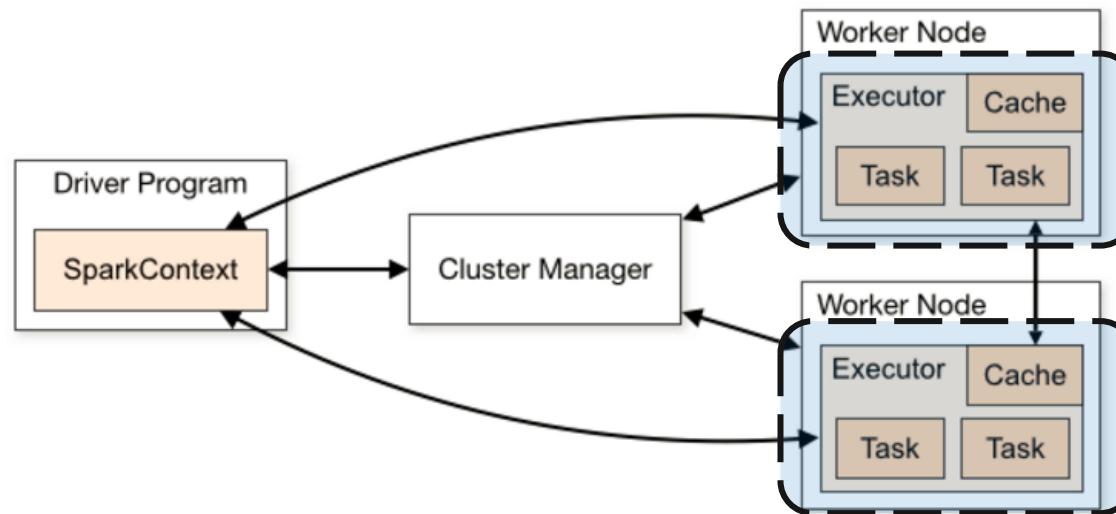
- Spark currently supports three cluster managers:
 - Standalone
 - The Leader process is also a simple cluster manager.
 - YARN (ResourceManager)
 - You can submit Spark applications in YARN by using two submission modes:
 - yarn-client
 - yarn-cluster
 - Apache Mesos
 - A general cluster manager that can also run Hadoop MapReduce and service applications

Note: Apache Mesos is not covered in this course

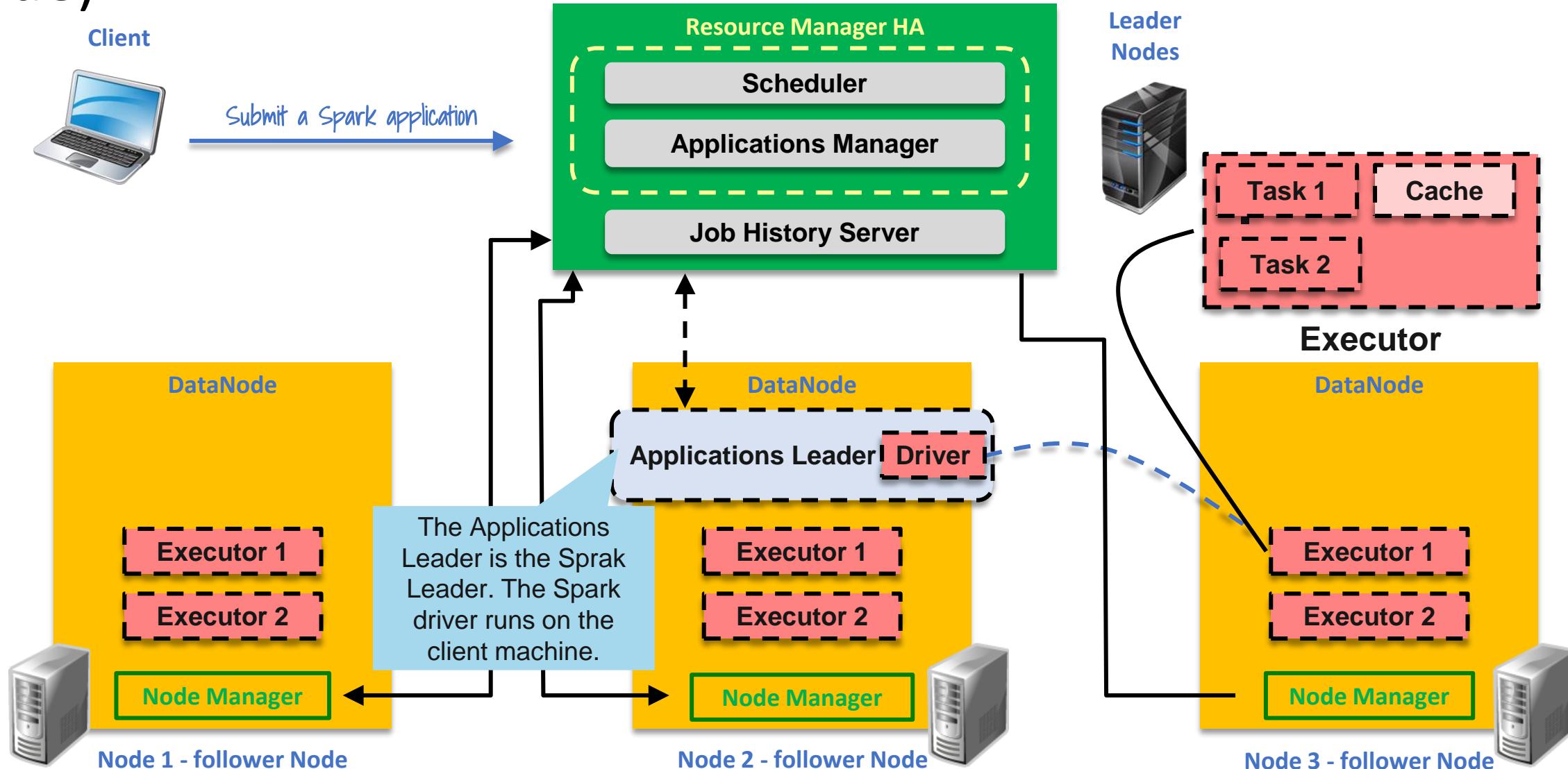


Executors

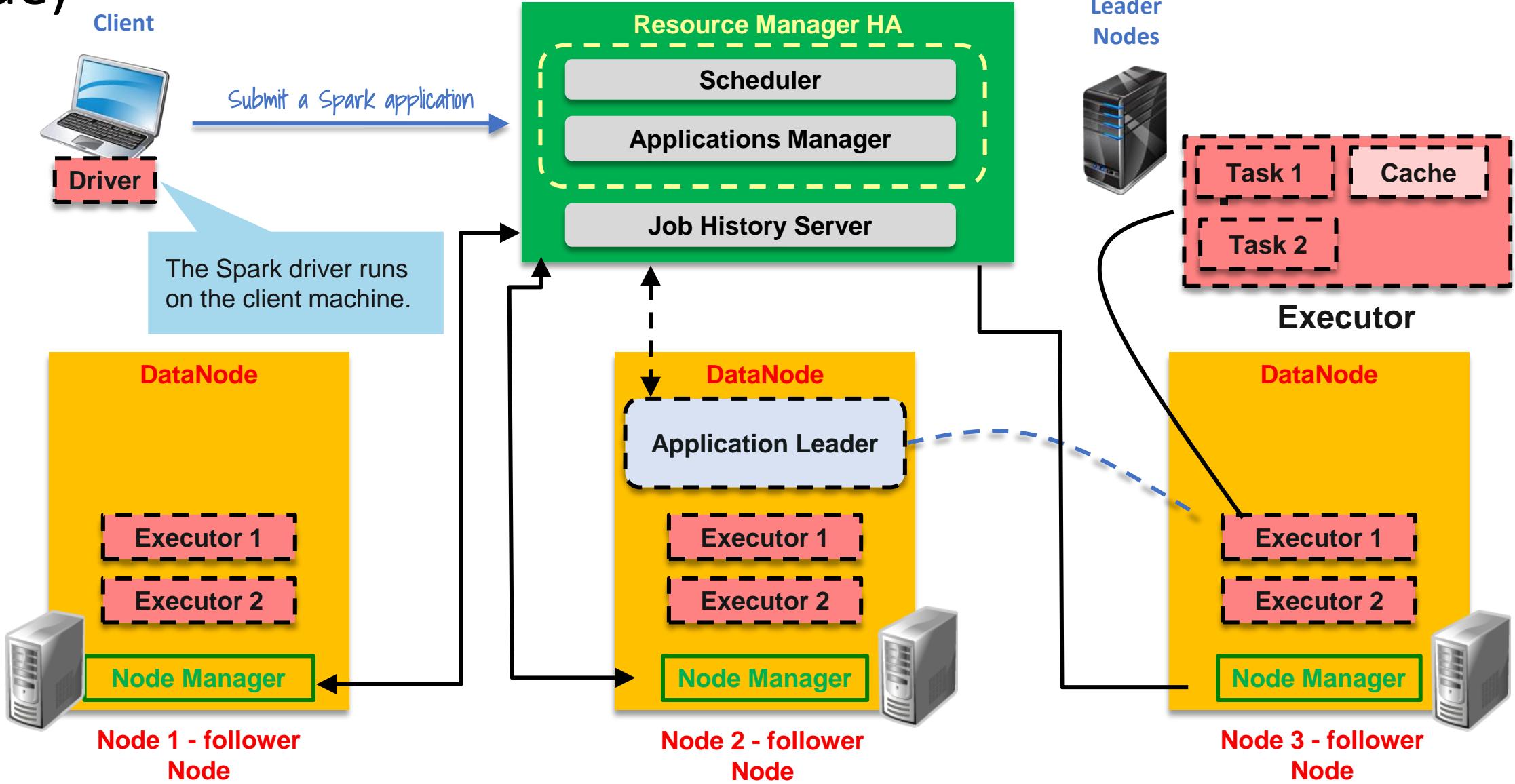
- Spark executors are the host processes on which tasks from a Spark DAG are run.
- Executors reserve CPU and memory resources on follower nodes or workers in a Spark cluster.
- Executors are dedicated to a specific Spark application and terminated when the application completes.



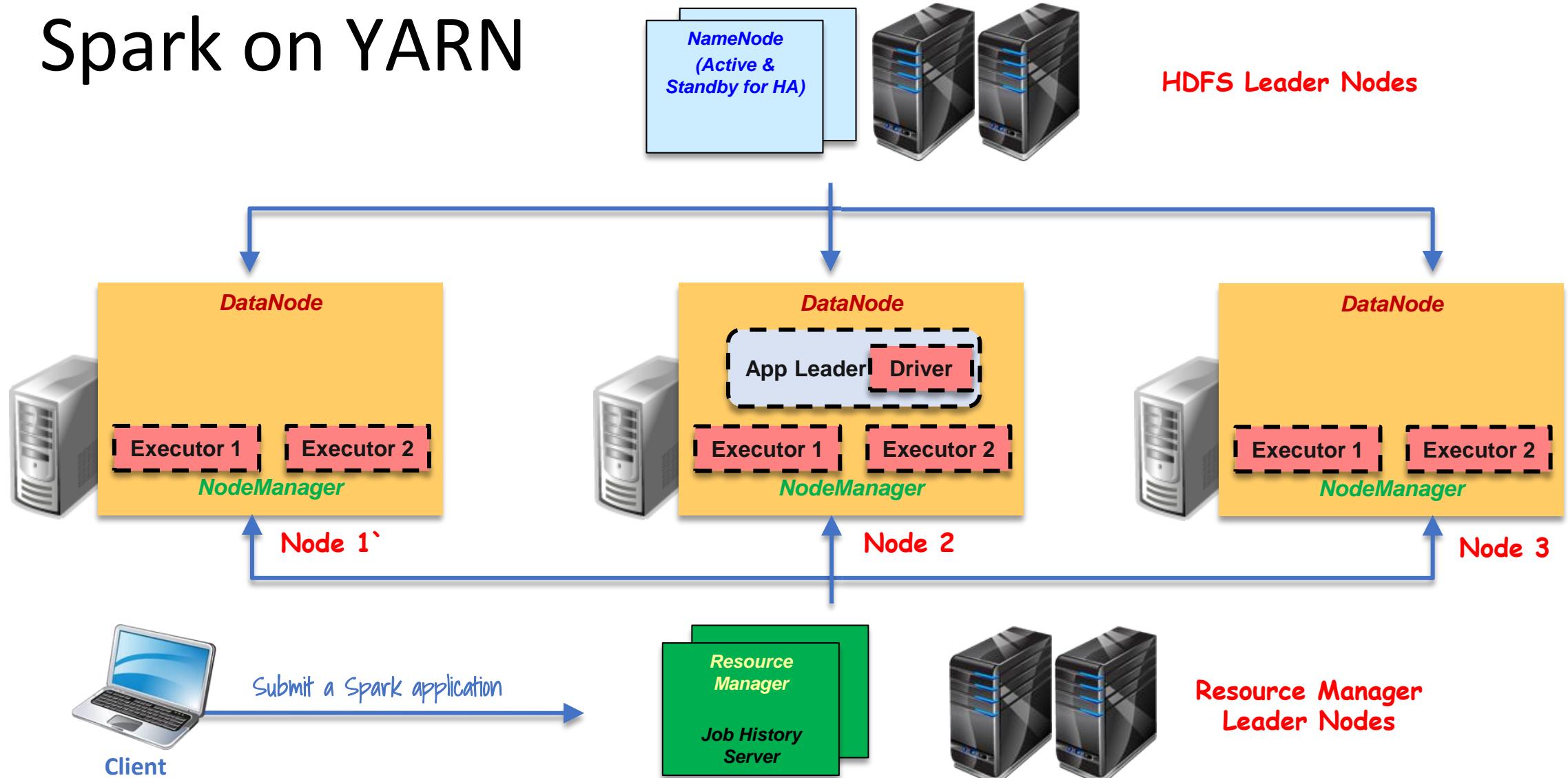
Running a Spark Application on YARN (yarn-cluster Mode)



Running a Spark Application on YARN (yarn-client Mode)



Spark on YARN



Resilient Distributed Dataset (RDD)

- Spark implements a distributed, fault tolerant, *in-memory* structure called a Resilient Distributed Dataset (RDD). RDD is the basic level of abstraction in Spark.
- Spark maximizes the use of memory across multiple machines, improving overall performance. Spark's reuse of these in-memory structures makes it well suited for iterative, machine learning operations, and interactive queries.
- RDD represents an immutable, partitioned collection of elements that can be operated on in parallel.
- RDD is input for your Spark jobs
- Spark provides two ways to create RDDs:
 - Loading an external data set (such as files, SQL or NoSQL datastores), or programmatically
 - Parallelizing a collection in your driver program

Resilient Distributed Dataset (RDD)

- Resilient:
 - If a node that is performing an application in Spark is lost, the data set can be reconstructed using the sequence of steps required to create the RDD.
- Distributed:
 - Data in RDDs is divided into one or many partitions and distributed as in-memory collections of objects across worker nodes in the cluster.
- Data set:
 - RDDs are data sets that consist of records.
 - Records are uniquely identifiable data collections within a data set.
 - Records could be a collection of fields similar to a row in a table in a relational database, a line of text in a file, or some other formats.

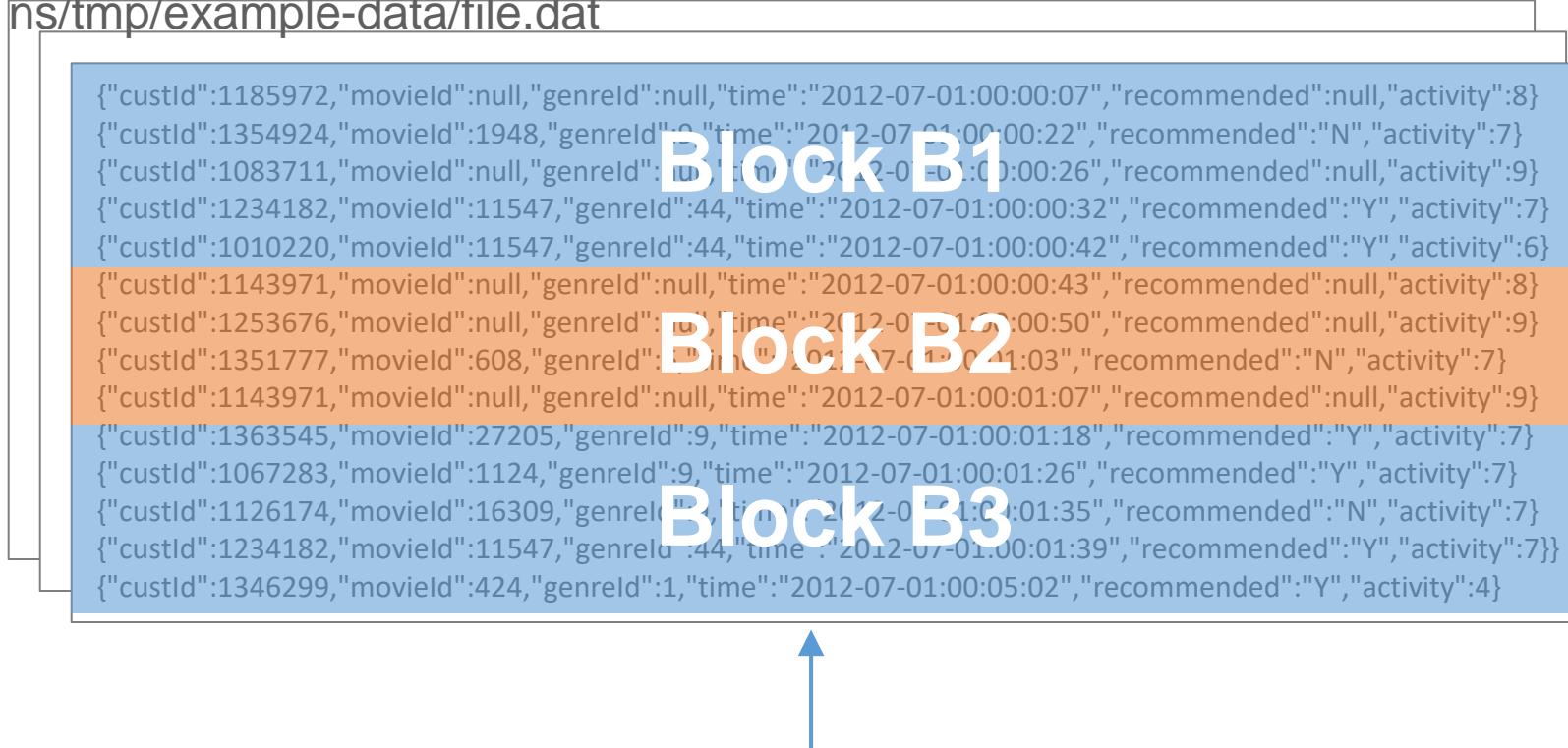
RDDs

- An RDD is the most fundamental data object used in Spark programming.
- RDDs are data sets within a Spark application. This includes:
 - The initial data set(s) loaded
 - Any intermediate data sets
 - The final resultant data set(s)
- Most Spark programming consists of creating new RDDs by performing operations on existing RDDs.
- RDDs are typically stored in memory and not on disk, a big advantage over MapReduce where intermediate data is stored on disk.

RDD

Terminology

Example: 1TB File: hdfs://namespace-ns/tmp/example-data/file.dat



RDD Operations

Data manipulation operations

Transformations

- Map
- Filter
- Sample
- Union
- Groupbykey
- Reducebykey
- Join & cache

Actions: Requests for output or a prompt to execute the program

Parallel Operations

- Reduce
- Collect
- Count
- Save
- lookupkey

Commonly Used Transformations

Transformations	Description
map ()	Returns a new RDD by applying a function to each element of the source
Filter ()	Creates a new RDD by passing in the supplied function used to filter the results
reduceByKey ()	Returns data set (K, V) pairs where the value for each key is aggregated using the given reduce function
flatMap ()	Returns a new RDD by applying a function to each element of the source. The function returns a sequence rather than a single item.

Sample Actions

Actions	Description
count ()	Returns the number of elements in the data set
reduce ()	Aggregates the elements of the data set using a function func (which takes two arguments and returns one)
collect ()	Returns all the elements of the data set as an array at the driver program
take (n)	Returns an array with the first n elements of the data set
first ()	Returns the first element of the data set (similar to take(1))

RDD Example

```
val files = sc.textFile("input.txt")
```

File: input.txt

Hello welcome to Hadoop World
Hello welcome to Spark
This is a Scala program



RDD: files

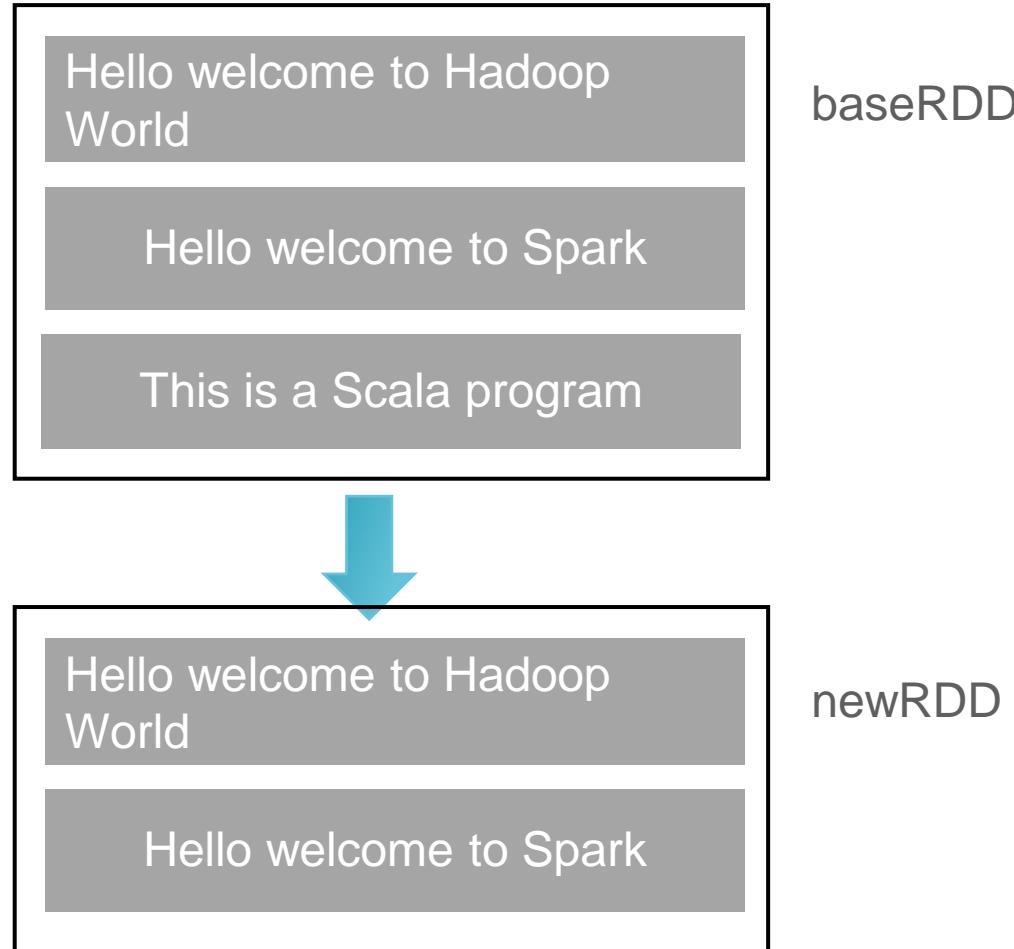
Hello welcome to Hadoop World

Hello welcome to Spark

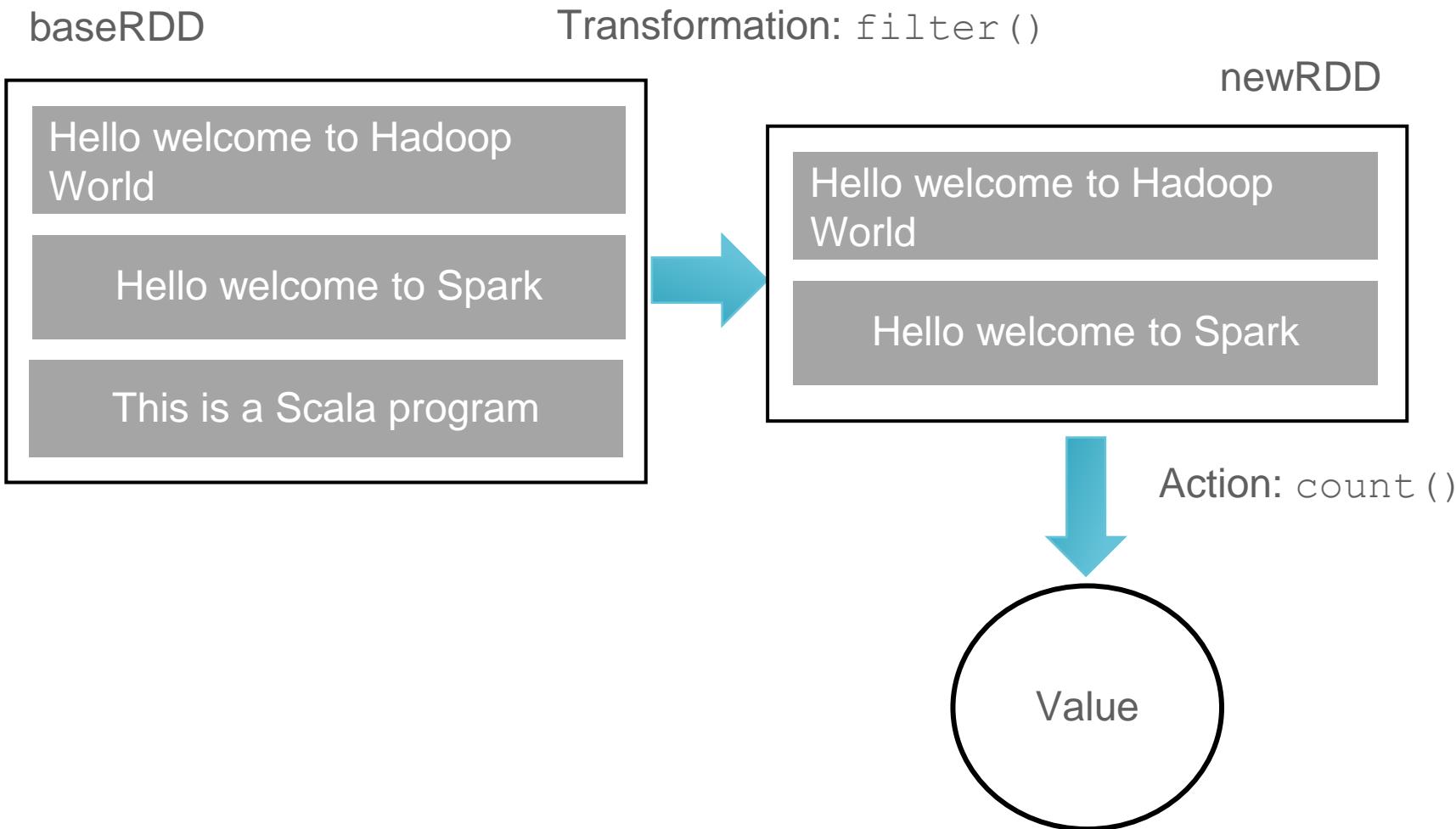
This is a Scala program

RDD Transformation: Filter() example

**Filter lines that start
with "H"**



RDD Action: Count () Example



Spark Interactive Shells

- There are two interactive Spark shells available to execute the Spark programs.

spark-shell is used for Scala.

pyspark is used for Python.

```
oracle@bigdatalite:~$ spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
Welcome to

    / \   / \
   / \ \ / \ \
  / \ \ / \ / \
 / \ \ / \ / \ / \
version 1.6.0

Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_101)
Type in expressions to have them evaluated.
Type :help for more information.
17/03/02 16:54:03 WARN util.Utils: Your hostname, bigdatalite.localdomain resolves to a loopback address: 127.0.0.1; using 10.247.45.235 instead (on interface eth0)
17/03/02 16:54:03 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Spark context available as sc (master = yarn-client, app id = application_1485739108130_0004).
SQL context available as sqlContext.

scala> |
```

Scala Language: Overview

- Scala programming language can be used for implementing Spark.
 - The acronym for “Scalable Language”
 - A pure-bred object-oriented language
 - Runs on the JVM
 - Reliable for large mission critical systems



Starting Interactive Spark-Shell for Scala

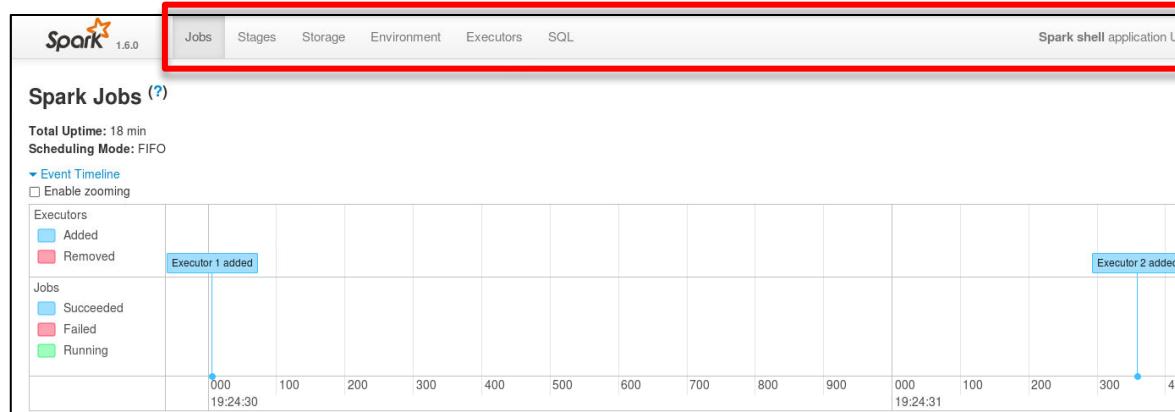
```
[oracle@bigdatalite ~]$ cd /home/oracle/exercises/spark  
[oracle@bigdatalite spark]$ pwd  
/home/oracle/exercises/spark  
[oracle@bigdatalite spark]$ spark-shell  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel).  
Welcome to  
 version 1.6.0  
Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_101)  
Type in expressions to have them evaluated.  
Type :help for more information.  
17/03/09 19:24:18 WARN util.Utils: Your hostname, bigdatalite.localdomain resolves to a loopback address: 127.0.0.1; using 10.247.45.235 instead (on interface eth0)  
17/03/09 19:24:18 WARN util.Utils: Set SPARK LOCAL IP if you need to bind to another address  
Spark context available as sc (master = yarn-client, app id = application_1485739108130_0016).  
SQL context available as sqlContext.  
scala>
```



ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Progress	Tracking UI
application_1485739108130_0016	oracle	Spark shell	SPARK	root.oracle	Thu Mar 9 19:24:21 -0500 2017	N/A	RUNNING	UNDEFINED	3	3	4096	<div style="width: 100%;"></div>	ApplicationMaster
application_1485739108130_0015	oracle	Spark shell	SPARK	root.oracle	Thu Mar 9 19:23:15 -0500 2017	Thu Mar 9 19:23:26 -0500 2017	FINISHED	SUCCEEDED	N/A	N/A	N/A	<div style="width: 100%;"></div>	History

Using the ResourceManager UI to Monitor a Spark Application

FinishTime	State	FinalStatus	Running Containers	Allocated CPU Vcores	Allocated Memory MB	Progress	Tracking UI
N/A	RUNNING	UNDEFINED	3	3	4096	<div style="width: 100%; height: 10px; background-color: #ccc;"></div>	ApplicationMaster



Spark 1.6.0

Jobs Stages Storage Environment Executors SQL

Spark shell application UI

Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Shuffle Write
Active(3)	0	0.0 B / 1590.0 MB	0.0 B	2	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0.0 B
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0.0 B
Total(3)	0	0.0 B / 1590.0 MB	0.0 B	2	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0.0 B

Executors

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
1	slc09fci.us.oracle.com:28089	Active	0	0.0 B / 530.0 MB	0.0 B	1	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
2	slc09fci.us.oracle.com:51625	Active	0	0.0 B / 530.0 MB	0.0 B	1	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
driver	10.247.45.235:53570	Active	0	0.0 B / 530.0 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump

This screenshot shows the Executors page. It displays a summary table with three active executors and a total of three. Below the summary is a detailed table listing each executor's ID, address, status, RDD blocks, storage memory, disk used, cores, active tasks, failed tasks, complete tasks, total tasks, task time, input, shuffle read, shuffle write, logs, and thread dump. A red box highlights the top navigation bar, and another red box highlights the 'ApplicationMaster' link in the Tracking UI dropdown.

Word Count Example by Using Interactive Scala

```
// Open the two input files and get their file handles.  
val files =  
  sc.textFile("hdfs://bigdatalite:8020/user/fenago/wordcount/input")  
  
// Set the execution context for the data.  
val counts = files.flatMap(line => line.split(" ")).map(word => (word,  
  1)).reduceByKey(_ + _)  
  
// Execute the counts and save the output to sp-out directory.  
counts.saveAsTextFile("hdfs://bigdatalite:8020/user/fenago/wordcount/sp-  
  out")  
  
// print out the counts to the console.  
counts.collect().foreach(println)
```

Creates a new RDD
from HDFS

Executes the word
Count transformation

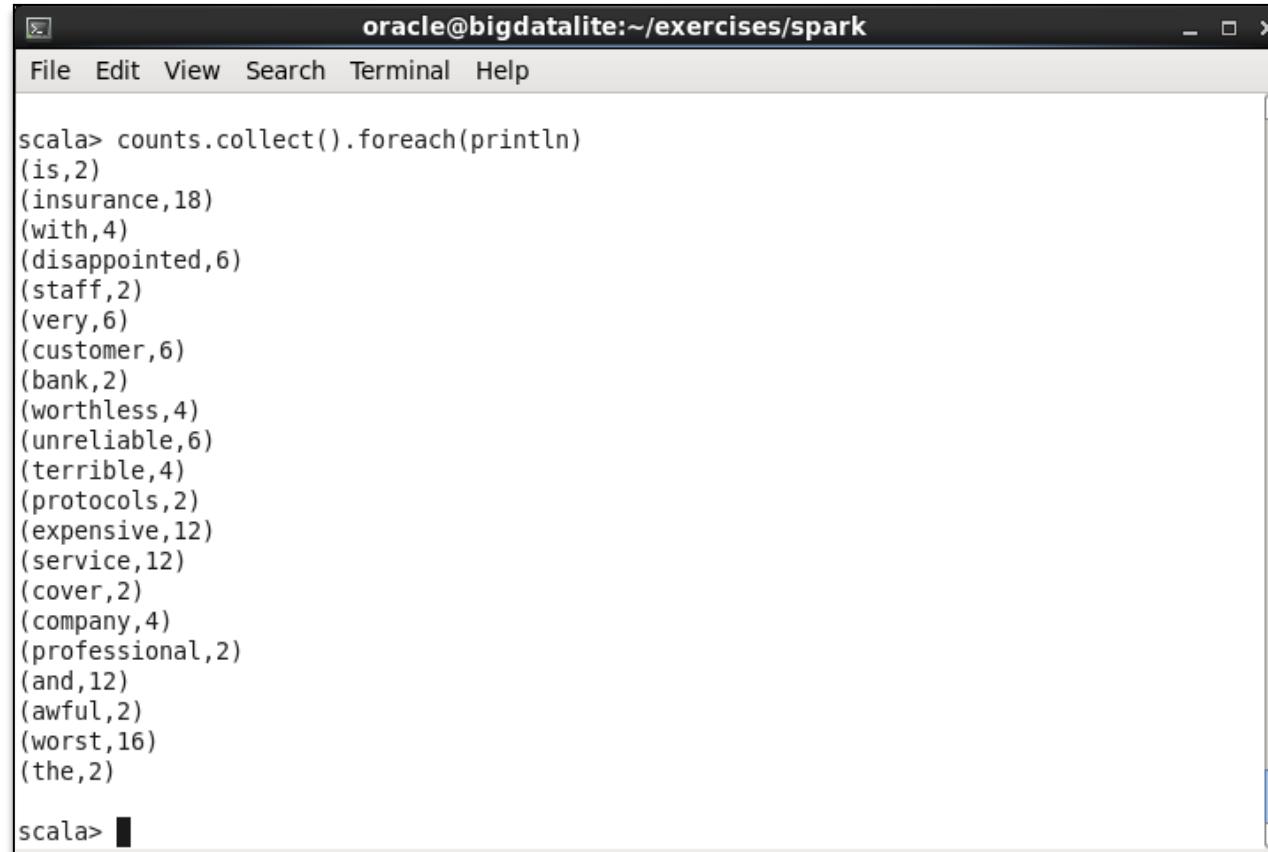
Applies the action

Print the output

```
[oracle@bigdatalite ~]$ hadoop fs -ls /user/oracle/wordcount/input  
Found 2 items  
-rw-r--r-- 1 oracle oracle 518 2016-08-24 12:07 /user/oracle/wordcount/input/file01  
-rw-r--r-- 1 oracle oracle 518 2016-08-24 12:08 /user/oracle/wordcount/input/file02  
[oracle@bigdatalite ~]$ hadoop fs -ls /user/oracle/wordcount/sp-out  
Found 3 items  
-rw-r--r-- 1 oracle oracle 0 2016-08-24 17:21 /user/oracle/wordcount/sp-out/_SUCCESS  
-rw-r--r-- 1 oracle oracle 89 2016-08-24 17:21 /user/oracle/wordcount/sp-out/part-00000  
-rw-r--r-- 1 oracle oracle 161 2016-08-24 17:21 /user/oracle/wordcount/sp-out/part-00001  
[oracle@bigdatalite ~]$
```

Check the output directory.

Word Count Example by Using Interactive Scala: Output



A screenshot of a terminal window titled "oracle@bigdatalite:~/exercises/spark". The window has a standard Linux-style title bar with icons for close, minimize, and maximize. Below the title bar is a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main area of the terminal shows the following Scala code and its output:

```
scala> counts.collect().foreach(println)
(is,2)
(insurance,18)
(with,4)
(disappointed,6)
(staff,2)
(very,6)
(customer,6)
(bank,2)
(worthless,4)
(unreliable,6)
(terrible,4)
(protocols,2)
(expensive,12)
(service,12)
(cover,2)
(company,4)
(professional,2)
(and,12)
(awful,2)
(worst,16)
(the,2)

scala>
```

Monitoring Spark Jobs by Using YARN's ResourceManager Web UI

The screenshot shows the Hadoop ResourceManager Web UI interface. At the top, there is a navigation bar with tabs: 'Hadoop' (selected), 'Spatial and Graph', and 'BDD'. Below the tabs, a dropdown menu shows 'NameNode bigdatalite.localdomain:8020' and a list of applications: 'YARN Applications' (selected) and 'JobHistory'. A blue arrow points from the 'YARN Applications' item in the dropdown to the corresponding section on the main page.

The main page title is 'All Applications'. It features a logo on the left with the word 'hadoop' in blue and yellow. On the left side, there is a sidebar with a 'Cluster Metrics' section and a 'User Metrics for dr.who' section. The 'Cluster Metrics' section contains tables for 'About', 'Nodes', and 'Applications' with various status counts like NEW, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, and KILLED. The 'User Metrics' section shows metrics for user 'dr.who'.

The central part of the page is a table titled 'All Applications' listing three completed Spark shell jobs:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated Capacity	VCore
application_1472055509950_0003	oracle	Spark shell	SPARK	root.oracle	Wed Aug 24 17:18:16 -0400 2016	Wed Aug 24 17:32:26 -0400 2016	FINISHED	SUCCEEDED	N/A	N/A	
application_1472055509950_0002	oracle	Spark shell	SPARK	root.oracle	Wed Aug 24 14:23:08 -0400 2016	Wed Aug 24 17:14:32 -0400 2016	FINISHED	SUCCEEDED	N/A	N/A	
application_1472055509950_0001	oracle	WordCount	MAPREDUCE	root.oracle	Wed Aug 24 12:23:39 -0400 2016	Wed Aug 24 12:24:19 -0400 2016	FINISHED	SUCCEEDED	N/A	N/A	

Scala Program: Word Count Example

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SparkWordCount {
    def main(args: Array[String]) {
        val sc = new SparkContext(new SparkConf().setAppName("Spark Count"))
        val threshold = args(1).toInt
        // split each document into words
        val tokenized = sc.textFile(args(0)).flatMap(_.split(" "))
        // count the occurrence of each word
        val wordCounts = tokenized.map((_, 1)).reduceByKey(_ + _)
        // filter out words with less than threshold occurrences
        val filtered = wordCounts.filter(_.value >= threshold)
        // count characters
        val charCounts = filtered.flatMap(_.value.toCharArray).map((_, 1)).reduceByKey(_ + _)
        System.out.println(charCounts.collect().mkString(", "))
    }
}
```

Definitions

Term	Description
Application	User program built on Spark. Consists of a <u>driver</u> program and <u>executors</u> on the cluster.
Application JAR	A JAR containing the user's Spark application. The user's JAR should never include Hadoop or Spark libraries; however, these are added at run time.
Driver program	The process running the <code>main()</code> function of the application and creating the <code>SparkContext</code>
Cluster Manager	An external service for acquiring resources on the cluster (e.g. stand-alone manager, Mesos, YARN)
Deploy mode	Distinguishes where the driver process runs. In "cluster" mode, the framework launches the driver inside of the cluster. In "client" mode, the submitter launches the driver outside of the cluster.
Worker node	Any follower node that can run application code in the cluster
Executor	A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.
Task	A unit of work that will be sent to one executor.
Job	A parallel computation consisting of multiple tasks that get spawned in response to a Spark action (e.g. <code>save</code> , <code>collect</code>); you'll see this term used in the driver's logs.
Stage	Each job gets divided into smaller sets of tasks called stages that depend on each other (similar to the map and reduce stages in MapReduce); you'll see this term used in the driver's logs.

Launching Spark Applications Using the spark-submit Command

```
./bin/spark-submit \
  --class <main-class> \
  --Leader <Leader-url> \
  --deploy-mode <deploy-mode> \
  --conf <key>=<value> \
  ... # other options
<application-jar> \
  [application-arguments]
```

```
# Run application locally on 8 cores
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --Leader local[8] \
  /path/to/examples.jar \
  100
```

Source: Apache Spark page on <http://spark.apache.org/docs/latest/submitting-applications.html>

Summary

- In this lesson, you should have learned how to:
 - Describe Spark and identify its benefits
 - Identify Spark's components and architecture
 - Interact with Spark
 - Run a Spark job and monitor the job using YARN's Resource Manager Web UI

