

Next Level Big Data Ecosystem (with Advanced Topics)





Table of Contents

- 0: Welcome to Class:
 - 1: Map Reduce and Hadoop:
 - 2: Hadoop Ecosystem and NoSQL:
 - 3: Learning from Big Data:
 - 4: NN for Big Data:
 - 5: Deep Big Data Analytics:
 - 6: NLP:
 - 7: More Data Science:
 - 8: Data Visualization:
- 

Lesson 0: Welcome to Class



Welcome!

- Who am I?
- What is the class schedule?
- What will you learn?

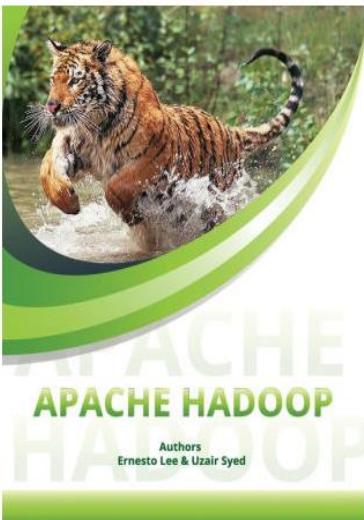
barnesandnoble.com/w/apache-hadoop-ernesto-lee/1128144600?ean=9781942864004

Super simple time ... PDF to Excel Conve... Schema Designer |... QB Online - for Acc... JupyterLab How to integrate M... XML_RPC_Open_Ne...

Books for All Ages: Buy 1, Get 1 50% Off Free Curbside Pickup Our Monthly Picks 50% Off Thousands of Items in Stores Read Midnight >

Books eBooks NOOK Textbooks Newsstand Teens Kids Toys Games & Collectibles Gift, Home & Office Movies & TV Music Sale

Shop > Books



Apache Hadoop: Invent the future
by Ernesto Lee, Uzair Syed

★★★★★ (0)

Paperback
\$79.99

Ship This Item – Qualifies for Free Shipping i

Buy Online, Pick up in Store i
Check Availability at Nearby Stores

ADD TO CART Sign in to Purchase Instantly

Members save with free shipping everyday!
See details

Who are you?

- Name and where you are from/at
- Experience with the tool
- Expectations from the course

What we will cover

- Introduction to Apache Spark
- Dataset Creation
- Operations on Datasets
- Build Spark Apps
- Spark UI
- Streaming
- Graphs
- Artificial Intelligence



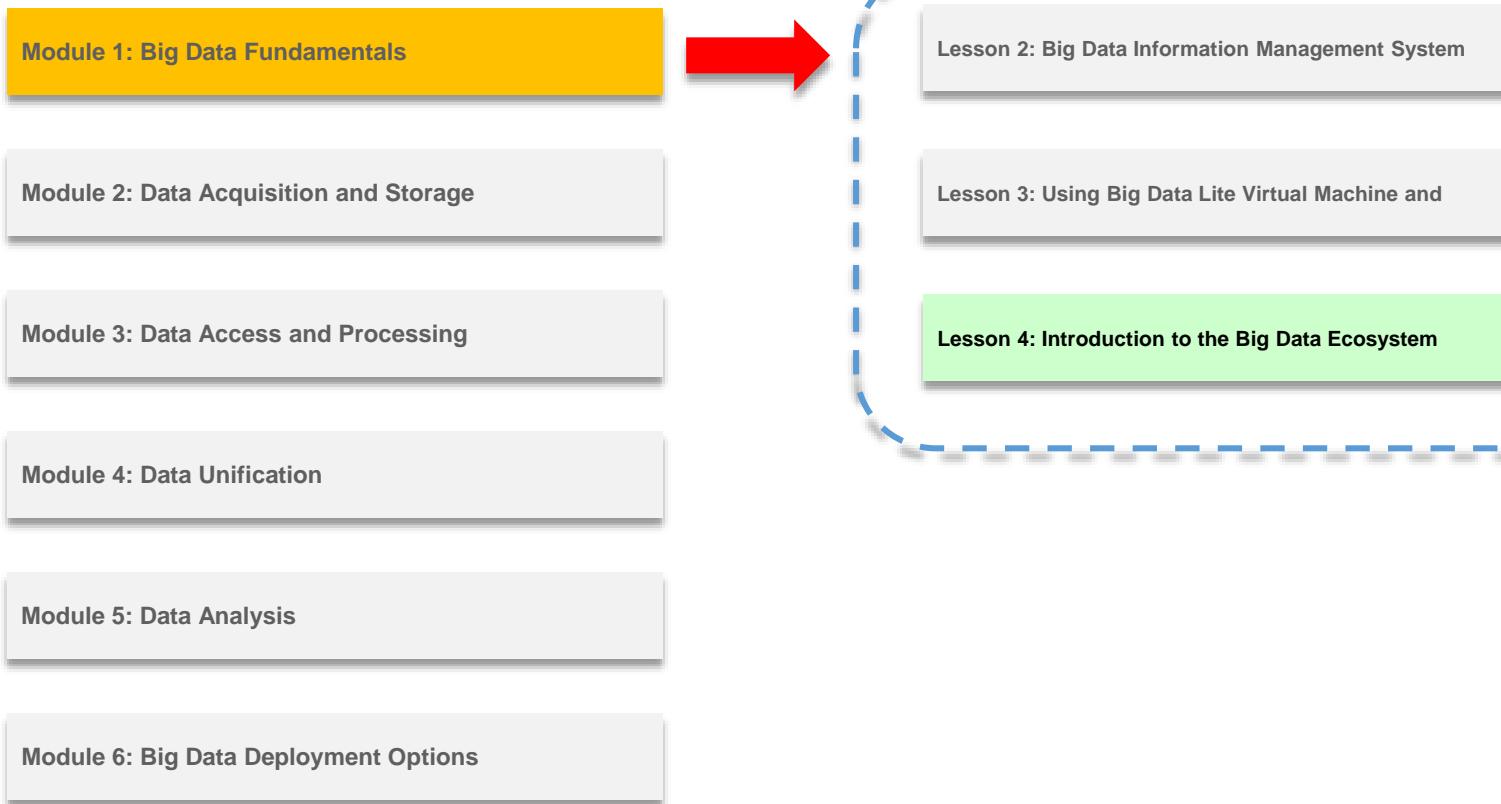
Lab Environment

- Everyone will have their own lab environment.
- <http://LLLLL##.eastus.azurecontainer.io>
- Replace LLLLL with the word provided by your instructor.
- ## will be the number assigned to your specific container
- These containers are ephemeral!!!

Introduction to the Big Data Ecosystem (Hadoop and Spark) for ML



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

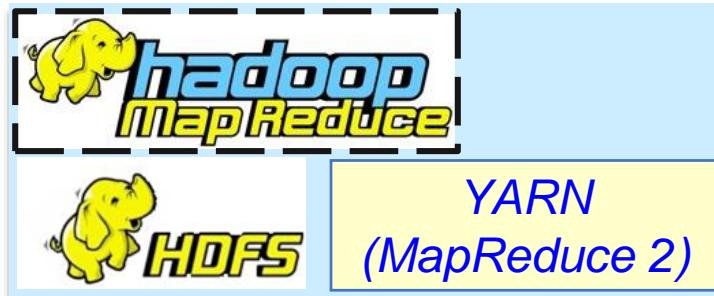
- Financial transactions
- Sensors data
- Server logs
- Analytics
- Email and text messages
- Social media



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)

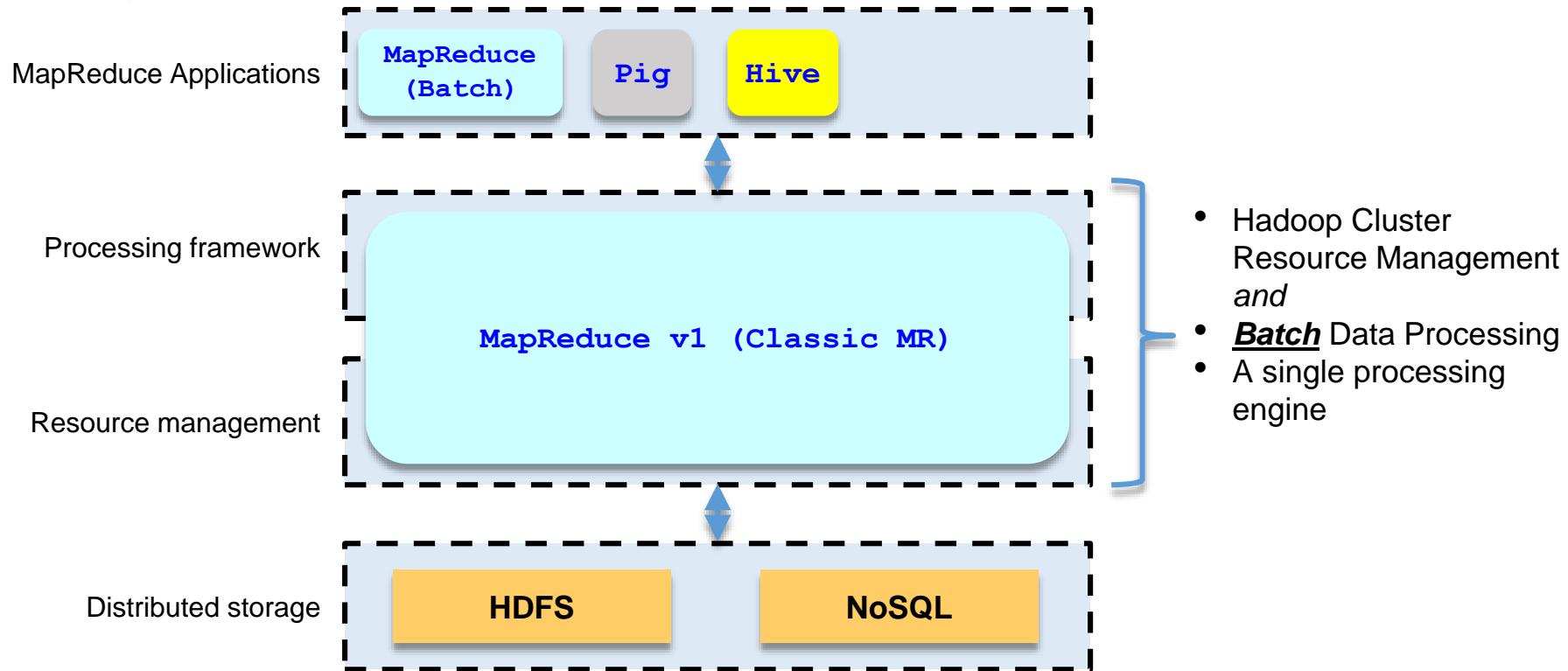


Apache Hadoop Core Components: MapReduce Framework (MRv1)

- 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)

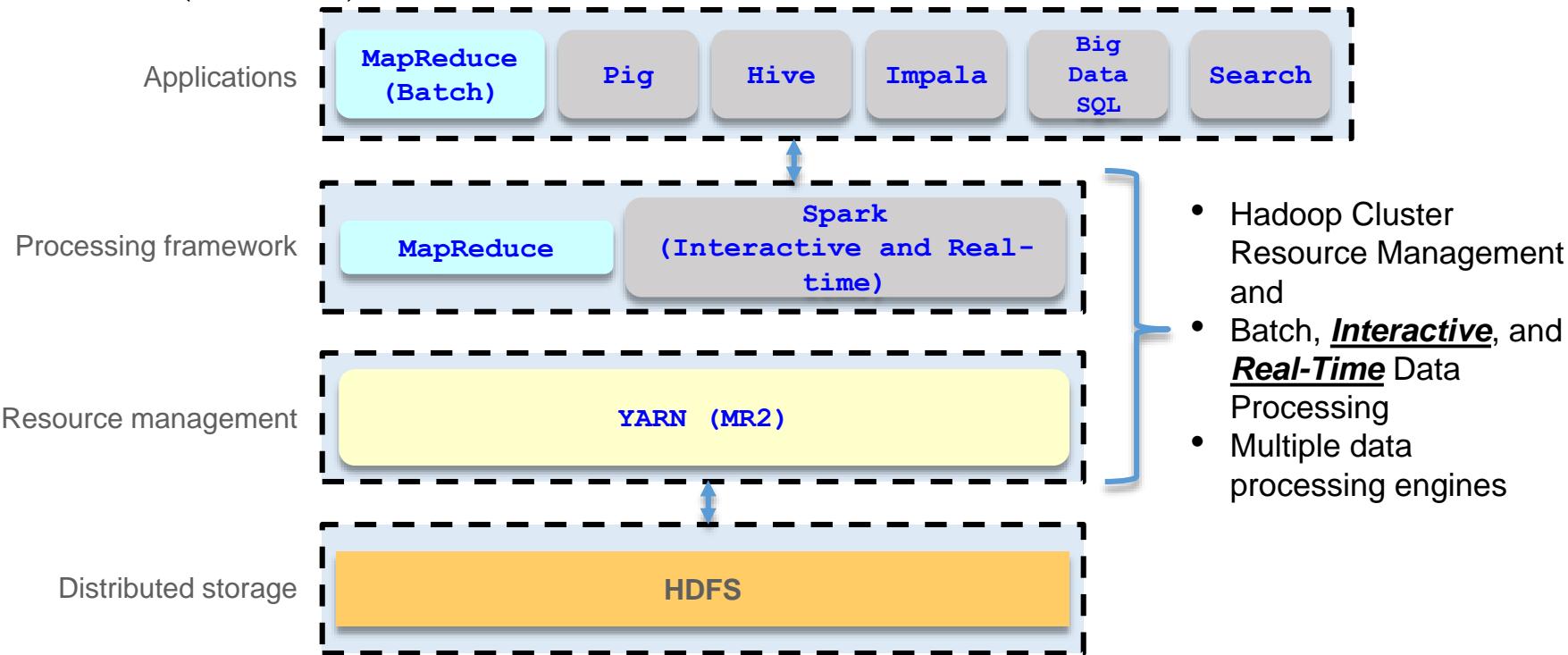


Apache Hadoop Core Components: YARN (MR2)

- 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)

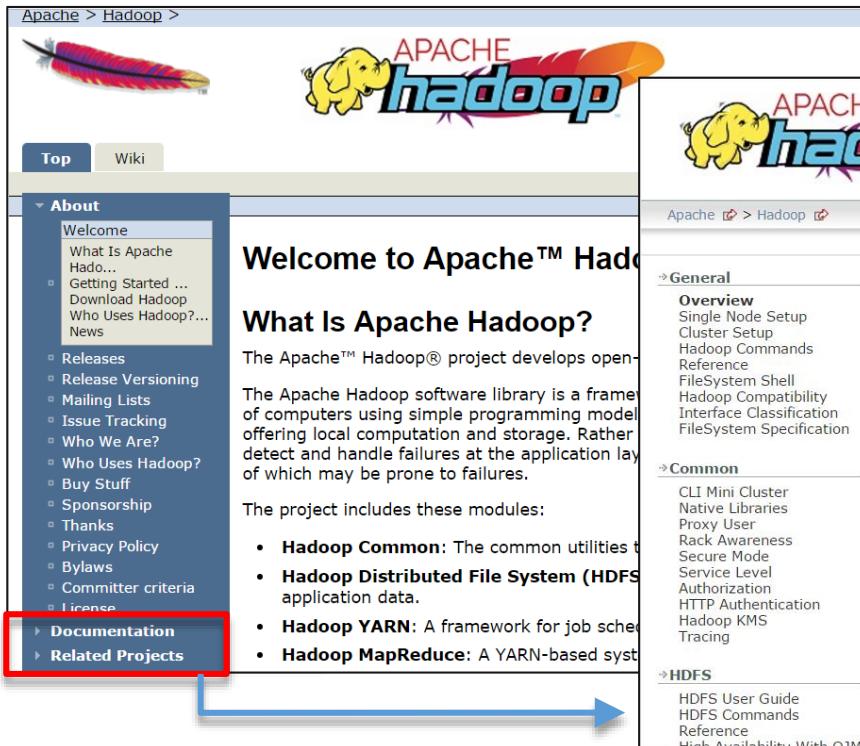


Hadoop Ecosystem:

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

Additional Resources: Apache Hadoop

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



The screenshot shows the Apache Hadoop website. At the top left is the Apache logo (feather icon) and at the top center is the Apache Hadoop logo (elephant icon). Below the logo is a navigation bar with 'Top' and 'Wiki' buttons. A sidebar on the left contains a 'About' section with links like 'Welcome', 'What Is Apache Hadoop...', 'Getting Started ...', 'Download Hadoop', 'Who Uses Hadoop?', 'News', 'Releases', 'Release Versioning', 'Mailing Lists', 'Issue Tracking', 'Who Are?', 'Who Uses Hadoop?', 'Buy Stuff', 'Sponsorship', 'Thanks', 'Privacy Policy', 'Bylaws', 'Committer criteria', and 'License'. Below this is a 'Documentation' link and a 'Related Projects' link, both of which are highlighted with a red box and connected by a blue arrow pointing to the right.

Welcome to Apache™ Hadoop

What Is Apache Hadoop?

The Apache™ Hadoop® project develops open-source software for distributed processing of large datasets across clusters of commodity servers.

The Apache Hadoop software library is a framework of computers using simple programming model offering local computation and storage. Rather than detect and handle failures at the application layer, it detects them at the system level and recovers from them automatically.

The project includes these modules:

- Hadoop Common:** The common utilities that support the system.
- Hadoop Distributed File System (HDFS):** A distributed file system designed for reliable storage of large amounts of data across clusters of commodity servers.
- Hadoop YARN:** A framework for job scheduling and resource management in clusters.
- Hadoop MapReduce:** A YARN-based system for distributed processing of large data sets.

General

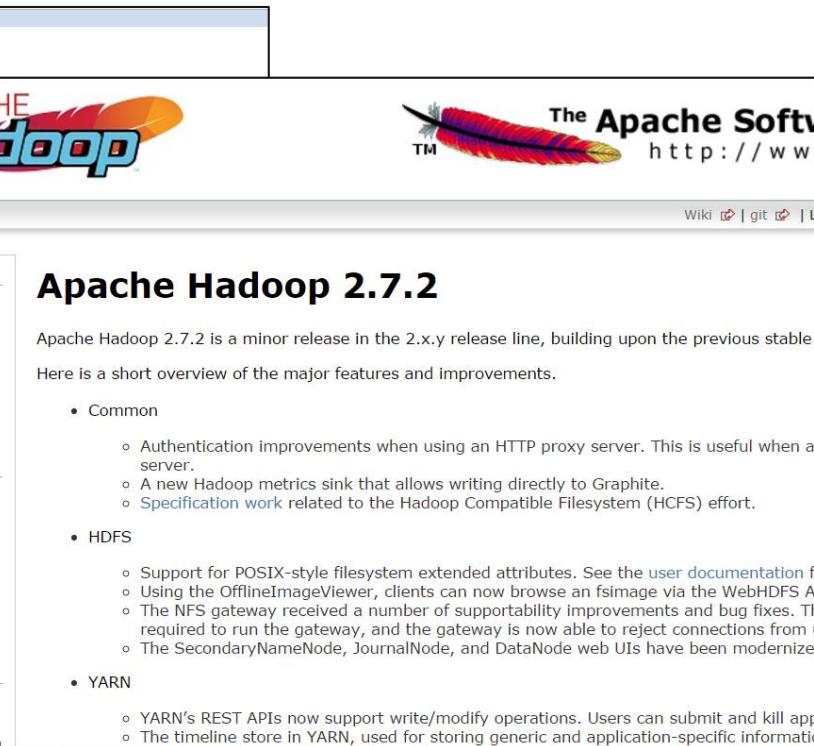
- Overview
- Single Node Setup
- Cluster Setup
- Hadoop Commands Reference
- FileSystem Shell
- Hadoop Compatibility
- Interface Classification
- FileSystem Specification

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 Ozone



The screenshot shows the Apache Hadoop 2.7.2 release page. At the top center is the Apache Hadoop logo. Below it is a banner with the text 'The Apache Software Foundation' and the URL 'http://www.apache.org/'. The main content area has a heading 'Apache Hadoop 2.7.2'. Below it is a paragraph about the release being a minor release in the 2.x.y release line. A bulleted list follows, detailing the major features and improvements.

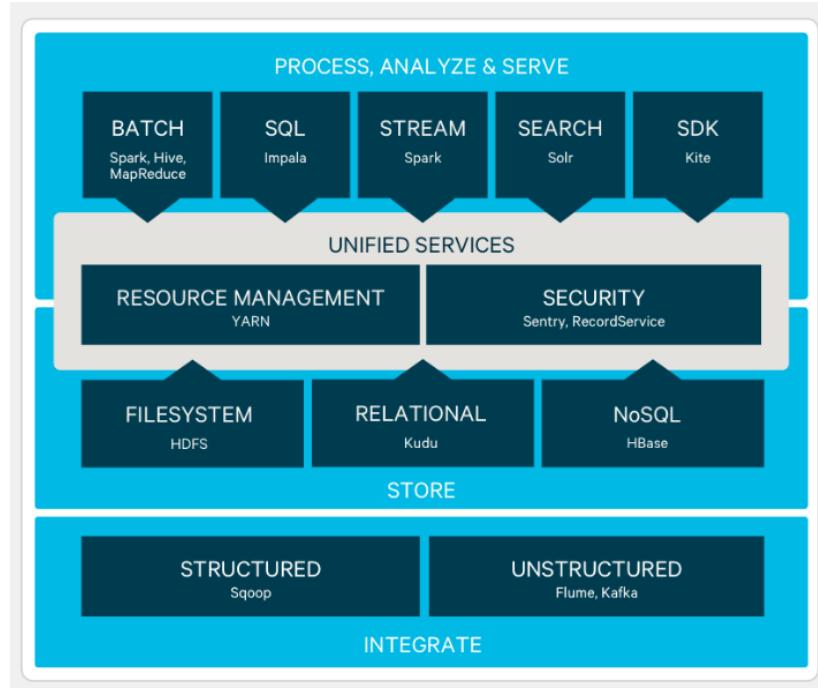
Apache Hadoop 2.7.2

Apache Hadoop 2.7.2 is a minor release in the 2.x.y release line, building upon the previous stable release.

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 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 NFS gateway is no longer 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.

Most BD Architectures



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. |

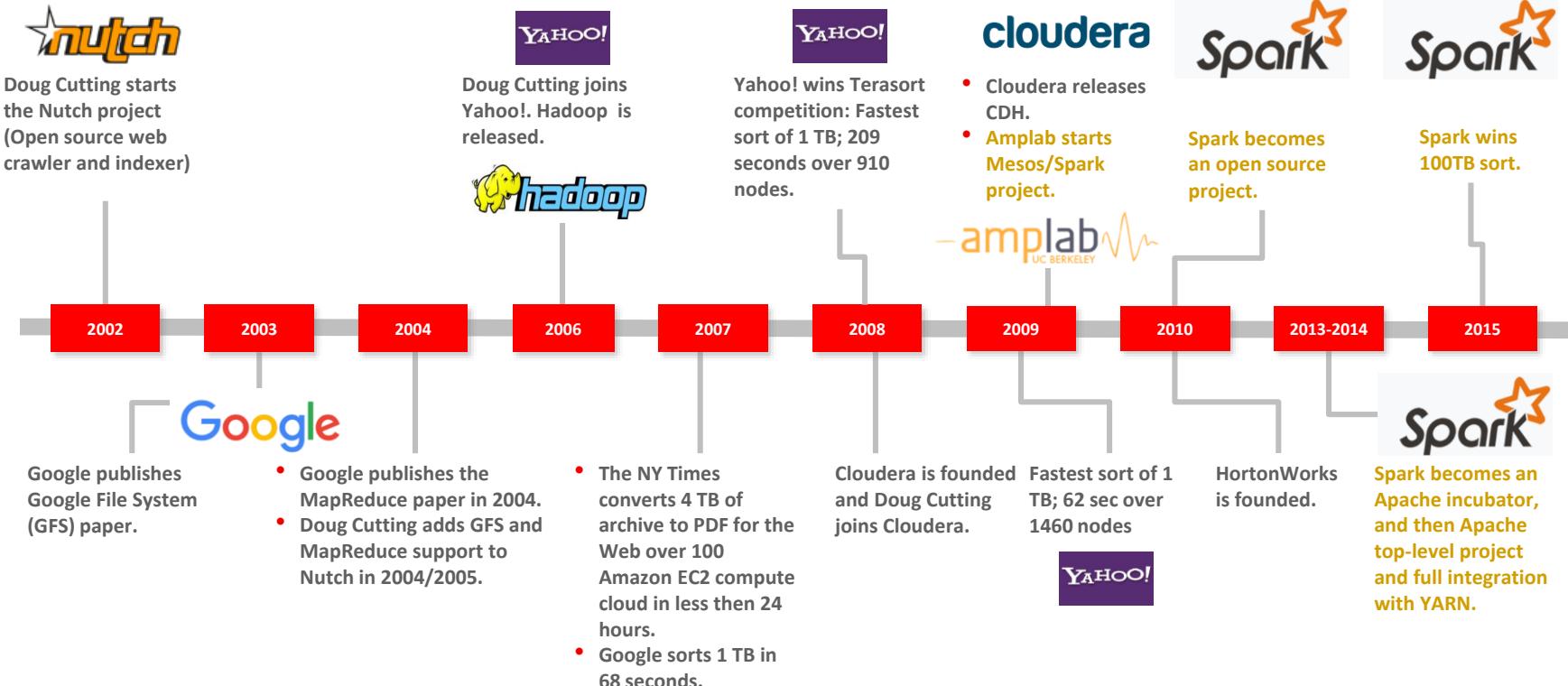
CDH Architecture

| 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, Oracle 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



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/ |

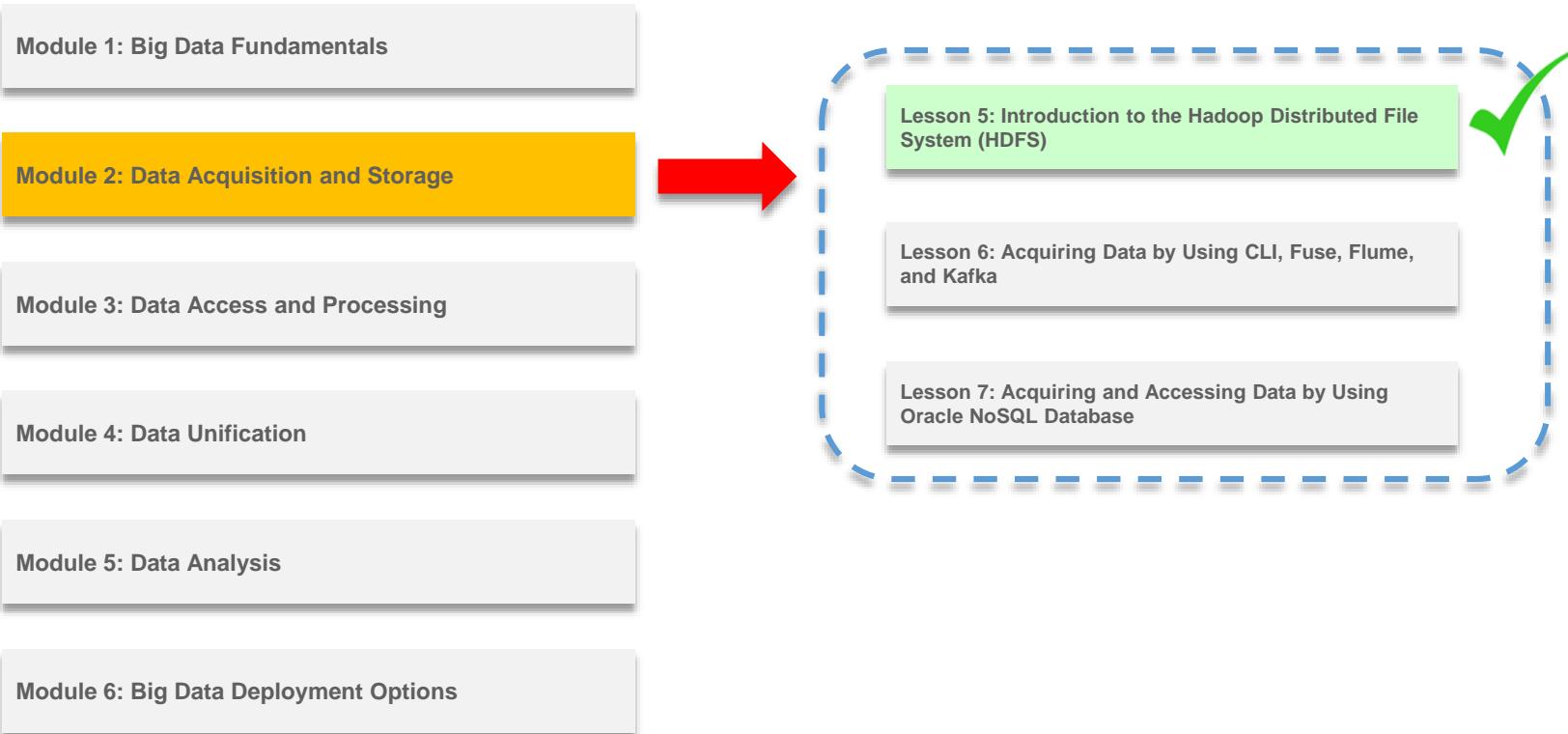
Summary

- 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

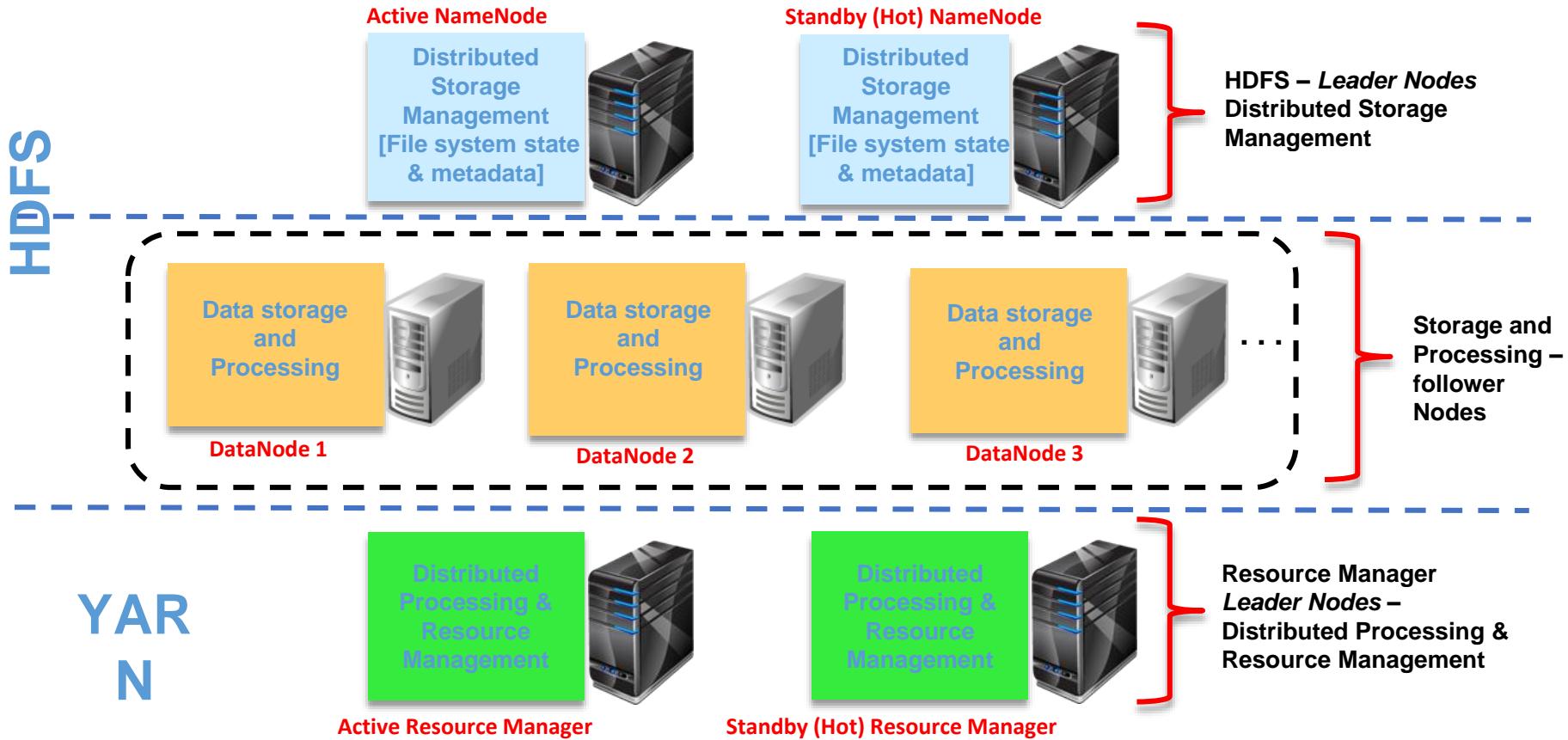
HDFS Key Definitions

| 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 (<i>does not contain actual data</i>). |
| 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. |

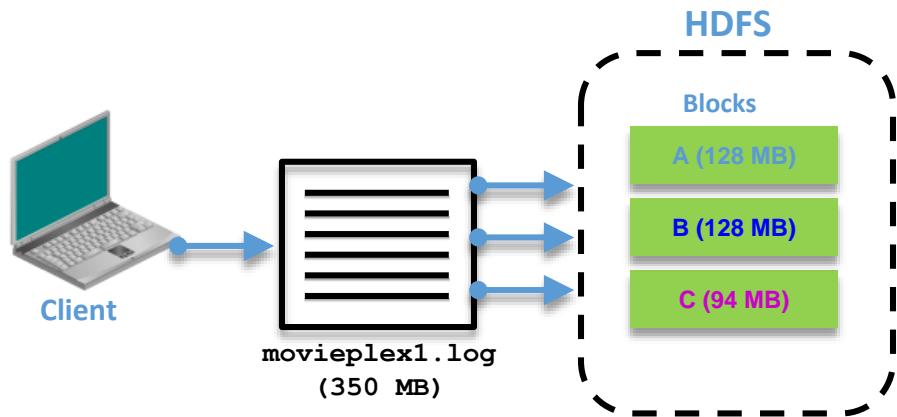
HDFS Deployments: High Availability (HA) and Non-HA

- 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

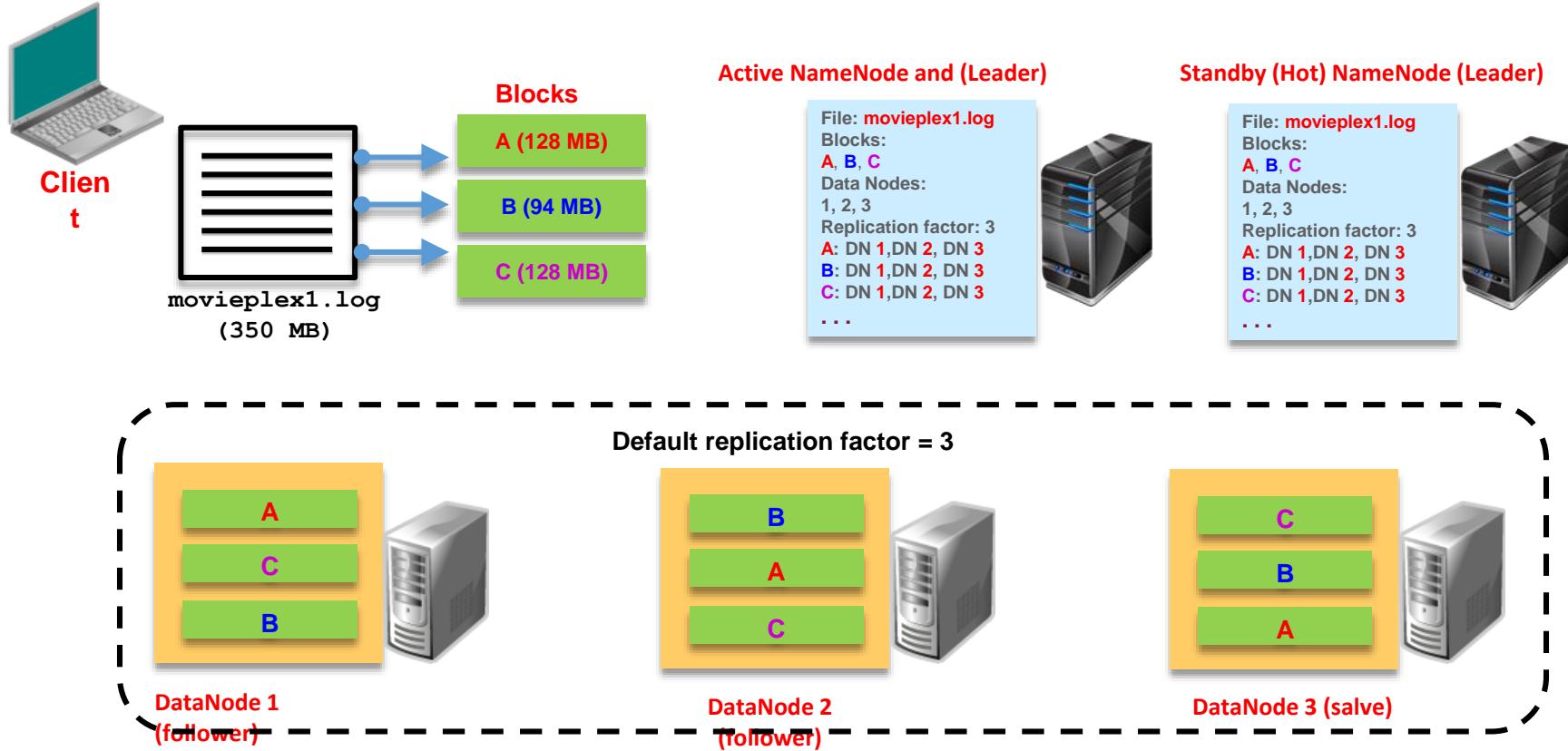


- Files in HDFS consist of blocks.
- HDFS blocks default to 128 MB in size (configurable).
- *Files are “chunked” into blocks as they are ingested (using Flume or Kafka) into HDFS.*

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 and Standby NameNodes Daemons

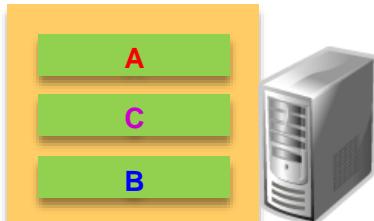


NameNode stores file system metadata such as:

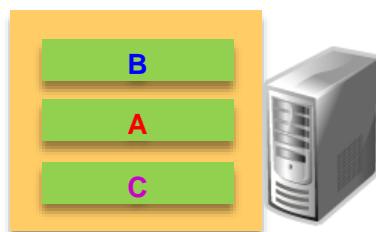
- File information (name, updates, replication factor, etc.)
- File blocks information and locations
- Access rights to the file
- Number of files in the cluster
- Number of DataNodes in the cluster

**Active NameNode and
Standby (Hot) NameNode (Leaders)**

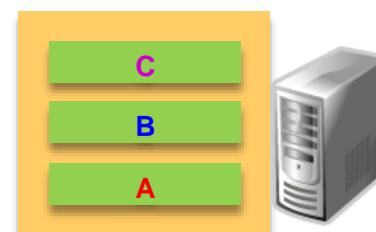
File: movieplex1.log
B1: File: movieplex1.log
A: Blocks:
Data: A, B, C
1, Data Nodes:
RF: 1, 2, 3
A: Replication Factor: 3
B: A: DN 1, DN 2, DN 3
C: B: DN 1, DN 2, DN 3
C: DN 1, DN 2, DN 3
...



DataNode 1
(follower)



DataNode 2
(follower)



DataNode 3 (salve)

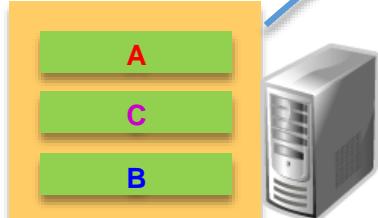
DataNodes Daemons



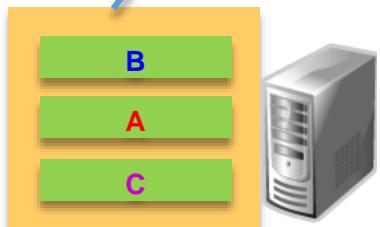
DataNodes

- 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”)

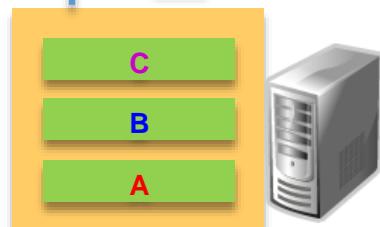
Heartbeat (every 3 seconds) &
Blockreport (every 6 hours)



DataNode 1
(follower)



DataNode 2
(follower)



DataNode 3 (salve)

Active NameNode and Standby (Hot) NameNode (Leaders)

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

File: movieplex1.log

Bl: File: movieplex1.log

A, B, C: Blocks:

Data Nodes:

1, 2, 3: RF: 3

A: RF: 3

B: A: DN 1, DN 2, DN 3

C: B: DN 1, DN 2, DN 3

C: C: DN 1, DN 2, DN 3

...

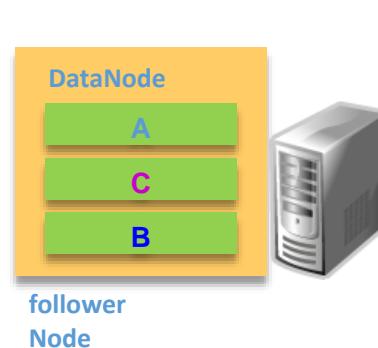


Functions of the NameNode

- 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

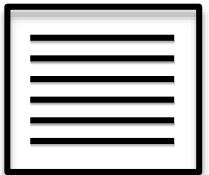


- Blockreport (what data I am storing!) every 6 hours to NN

Writing a File to HDFS: Example

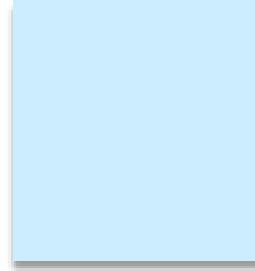


Client



movieplex1.log
(350 MB)

Active NameNode and (Leader)



Standby (Hot) NameNode (Leader)



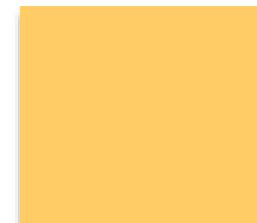
Default replication factor = 3



DataNode 1
(follower)

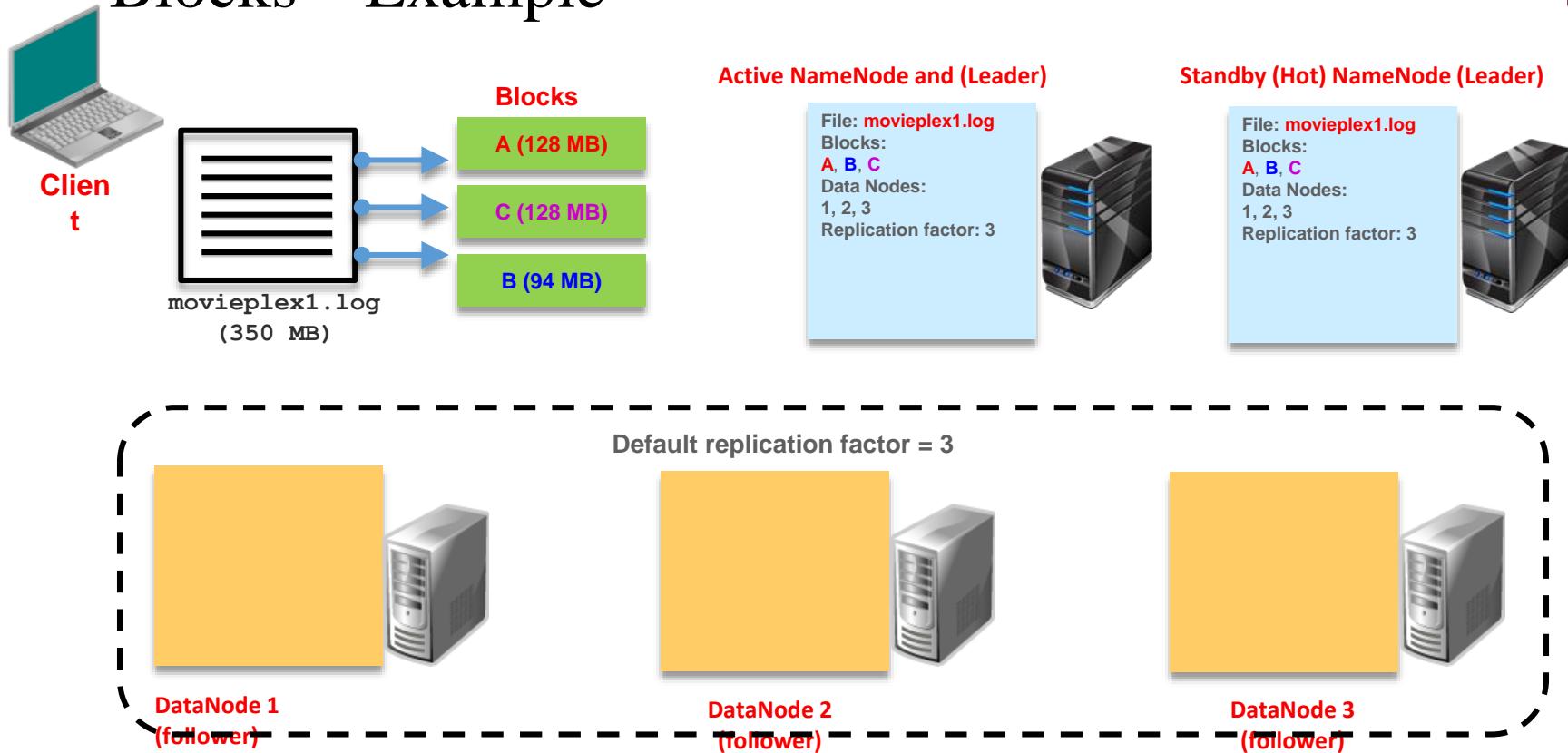


DataNode 2
(follower)

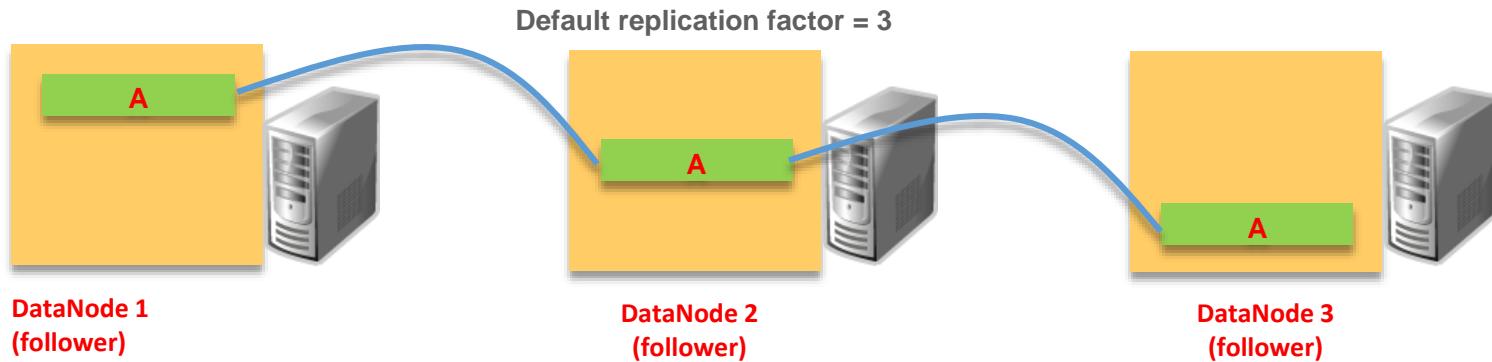
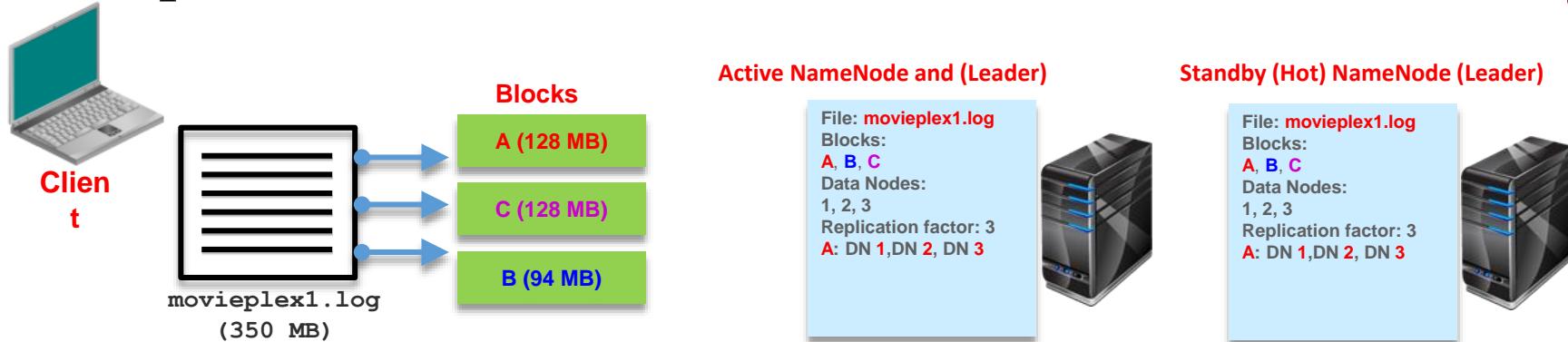


DataNode 3
(follower)

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

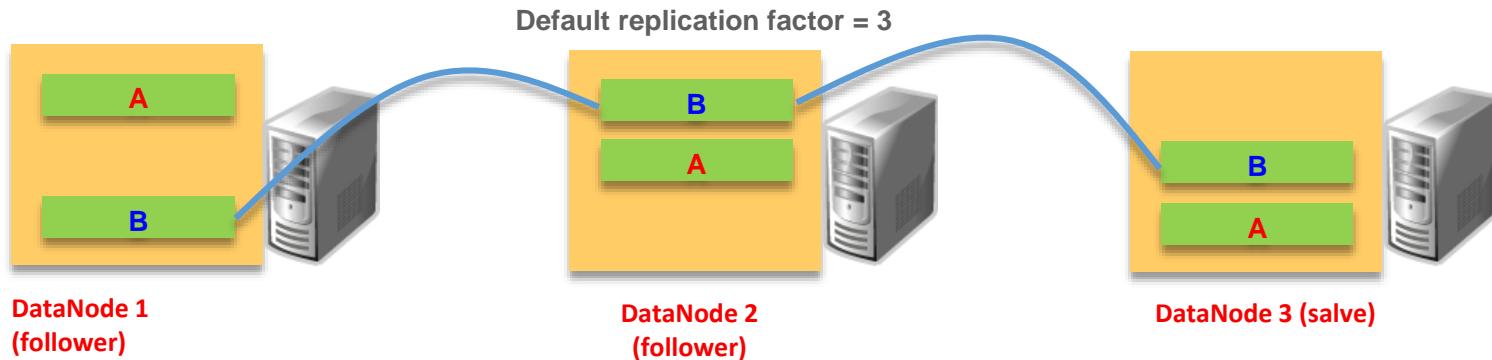
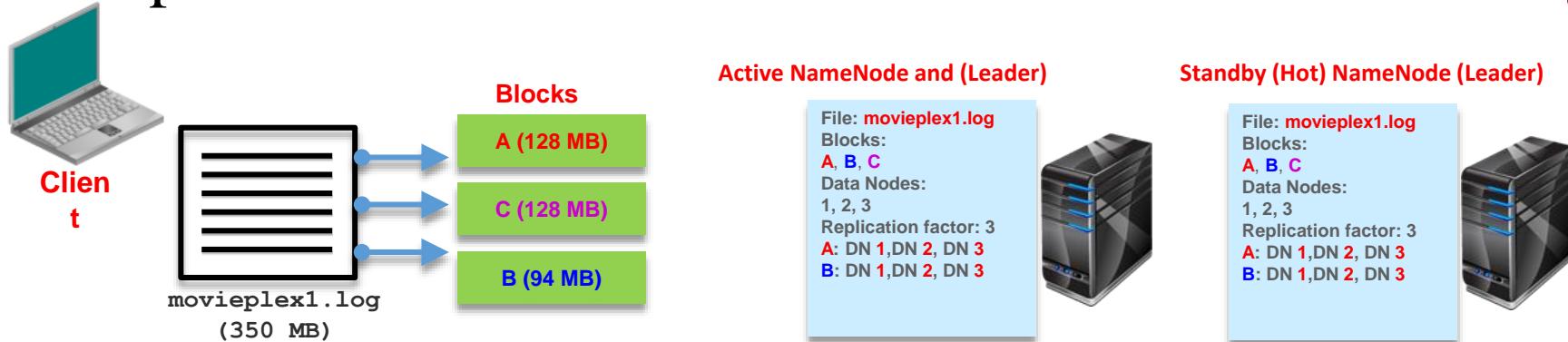


Writing a File to HDFS: Pipeline Created, Block A – Example





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

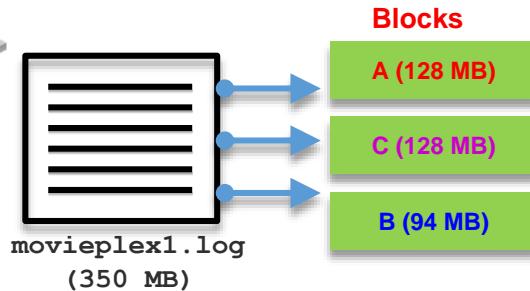




Writing a File to HDFS: Pipeline Created, Block C – Example



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
...



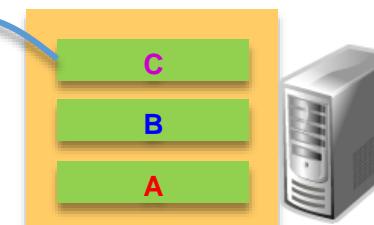
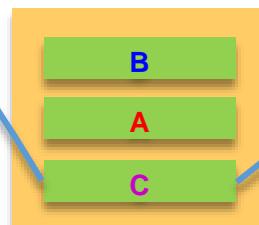
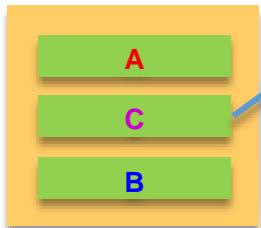
Standby (Hot) NameNode (Leader)

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)

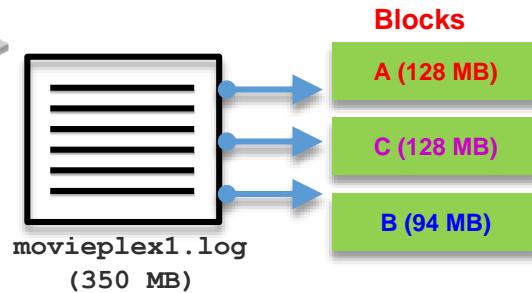
Default replication factor = 3



Writing a File to HDFS: Example



Client



Blocks

A (128 MB)

C (128 MB)

B (94 MB)

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
...



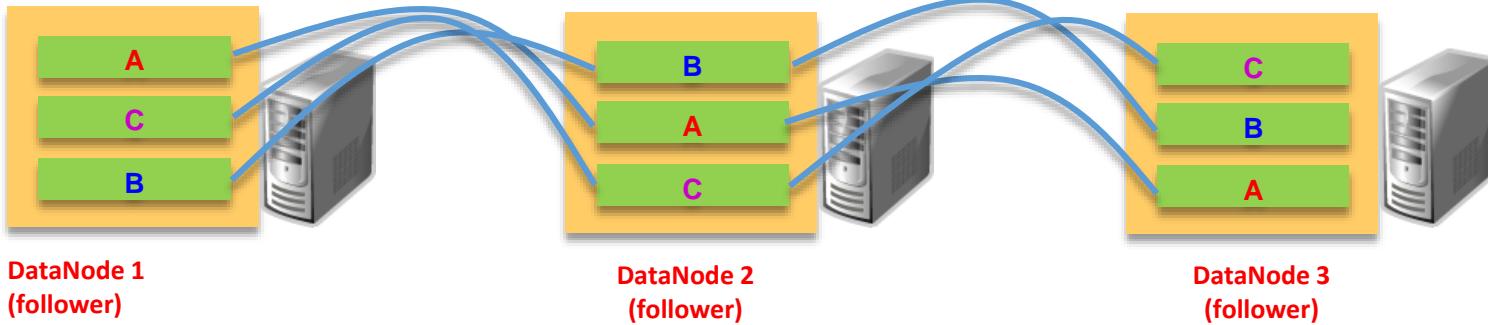
Standby (Hot) NameNode (Leader)

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)

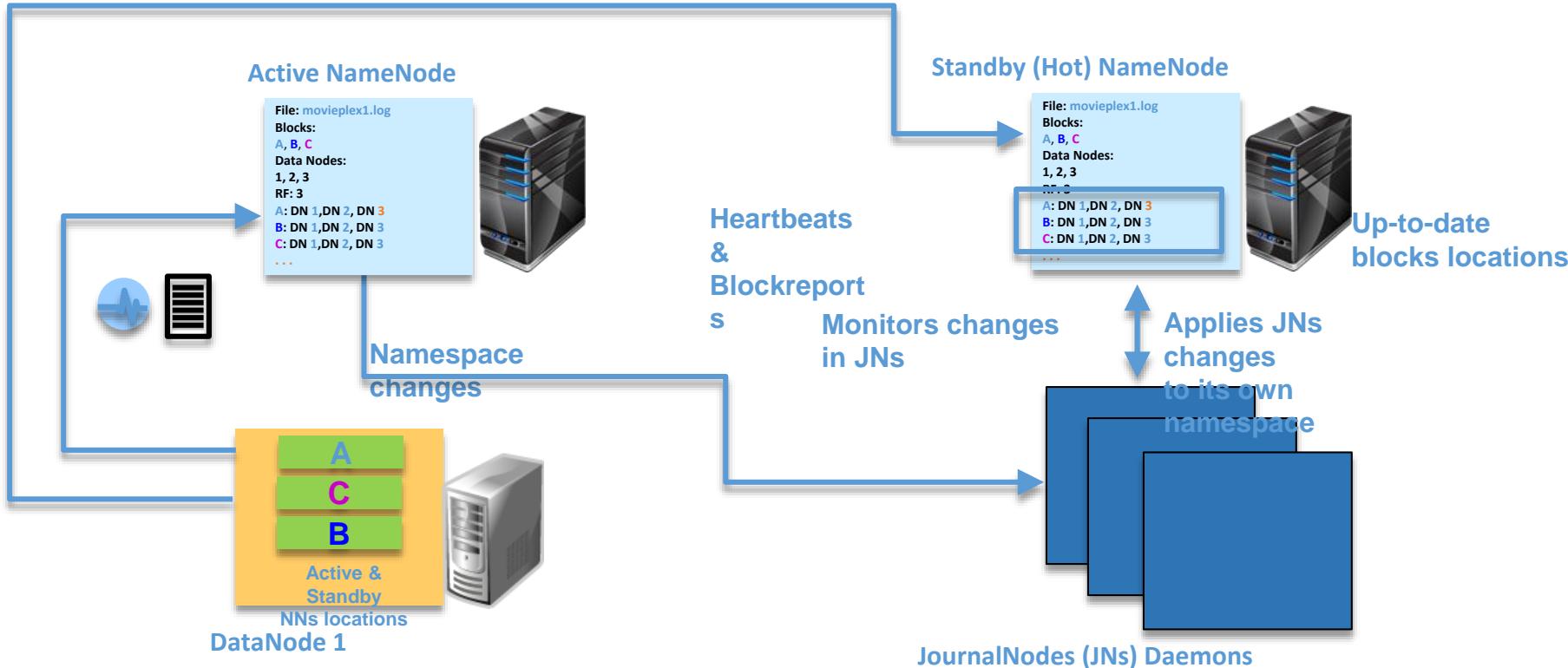
Default replication factor = 3



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

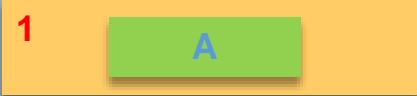
- Using Cloudera Manager:
 - Enable HA and Automatic Failover
- Using the command-line interface to configure automatic failover. Automatic failover adds the following components to an HDFS deployment:
 - A ZooKeeper quorum, which provides:
 - Failure detection
 - Active NameNode election
 - ZKFailoverController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

Data Replication Rack-Awareness in HDFS

Block A :

A

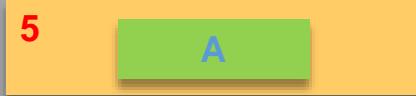
Rack 1



Block B :

B

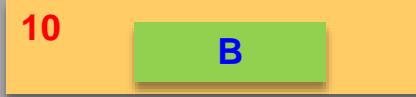
Rack 2



Block C :

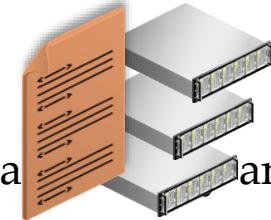
C

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 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

Mozilla Firefox

Connecting... Search or enter address

Most Visited Hue MoviePlex Hadoop Spatial and Graph BDD

NameNode bigdatalite.localdomain:8020

TASK Applications JobHistory Open All in Tabs

Hadoop NameNode bigd... - Mozilla Firefox

localhost:50070/dfshealth.jsp

Most Visited Hue MoviePlex Hadoop Spatial and Graph BDD Apex ORDS Lab Solr Admin SQL Pat

NameNode 'bigdatalite.localdomain:8020' (active)

| | |
|----------------|---|
| Started: | Sun Jan 29 20:17:54 EST 2017 |
| Version: | 2.6.0-cdh5.8.0, 042da8b868a212c843bcf3594519dd26e816e79 |
| Compiled: | 2016-07-12T23:02Z by jenkins from Unknown |
| Cluster ID: | cluster7 |
| Block Pool ID: | BP-314833115-127.0.0.1-1475613842090 |

[Browse the filesystem](#)
[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

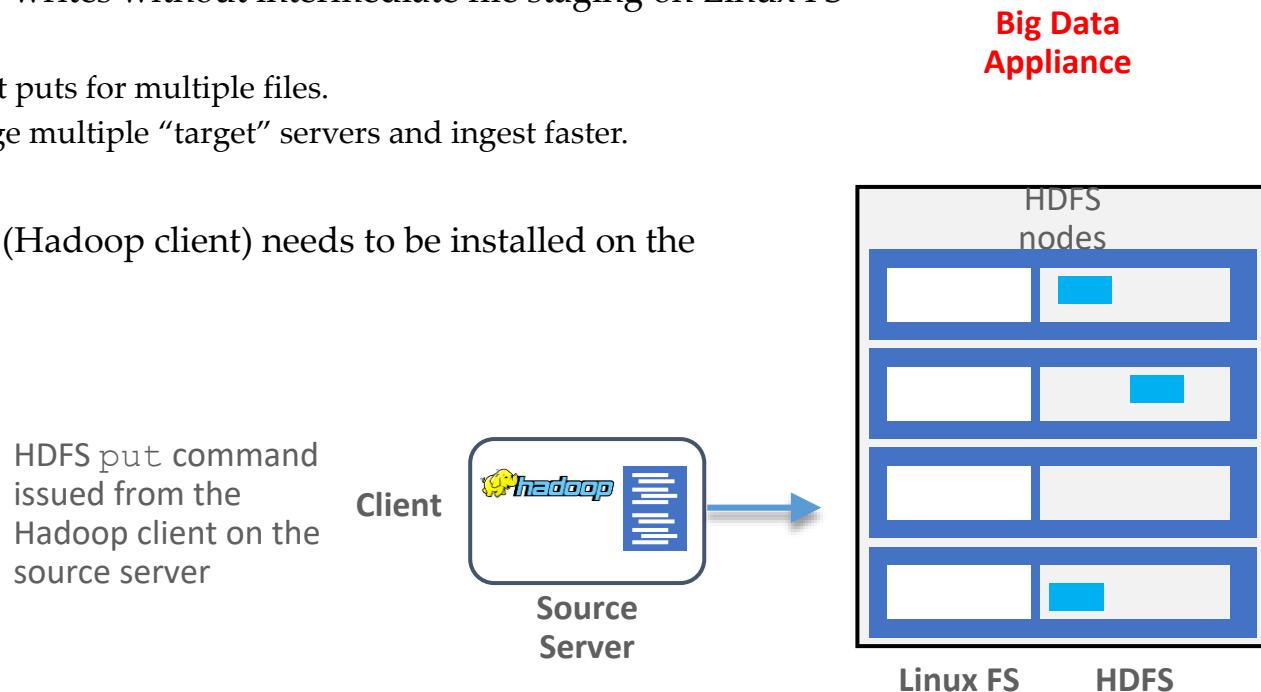
http://bda1node03.example.com:8888

The screenshot shows the Cloudera Hue interface running in Mozilla Firefox. The browser address bar displays the URL `http://bda1node03.example.com:8888`. The main window is titled "Hue - File Browser - Mozilla Firefox". The navigation bar includes links for "bigdatalite:8888/filebrowser/", "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: "File", "Machine", "View", "Input", "Devices", "Help", "Query Editors", "Data Browsers", "Workflows", "Search", and "Security". A blue arrow points to the "File Browser" link in the header. The main content area is titled "File Browser" and shows a list of files in the "/user/oracle" directory. The columns are labeled "Name", "Size", "User", "Group", "Permissions", and "Date". The table data is as follows:

| Name | Size | User | Group | Permissions | Date |
|---------------|------|--------|------------|-------------|------------------------|
| hdfs | | 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 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

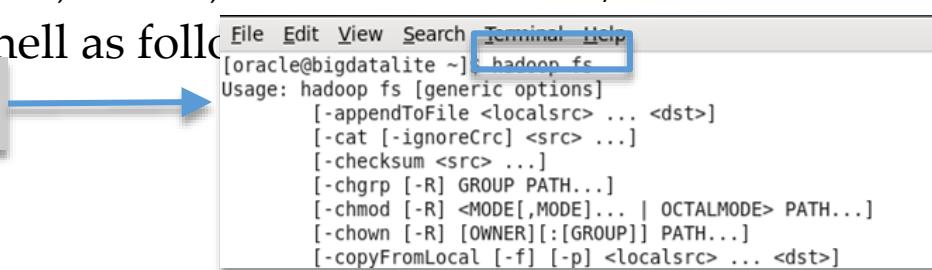
- Hadoop Commands Guide
 - Overview
 - Generic Options
 - Hadoop Common Commands
 - User Commands
 - archive
 - checknative
 - classpath
 - credential
 - distcp
 - fs
 - jar
 - key
 - trace
 - version
 - CLASSNAME
 - Administration Commands
 - daemonlog



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>
```



A screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". Below the menu is a command line prompt: "[oracle@bigdatalite ~] hadoop fs". A blue arrow points from the "hadoop fs <args>" text above to this command line. The terminal displays the usage information for the "hadoop fs" command, which includes options like "-appendToFile", "-cat", "-checksum", "-chgrp", "-chmod", "-chown", and "-copyFromLocal".

```
File Edit View Search Terminal Help  
[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|f|l|k1|f|m1|x] <src> <localSnapshot>]
```

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
- Edits 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
- rmmr

```
[oracle@bigdatalite ~]$ ls -l
total 8316
drwxr-xr-x. 2 oracle oinstall 4096 Jan 23 13:15 bigdatalog-hol
-rw-r--r--. 1 oracle oinstall 5545 Jan 23 13:17 bigdatalog-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
drwxr-----. - 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 -r 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
```

```
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 ~]$
```

- 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.SSLFileBasedKeyStoresFactory: The property 'ssl.client.truststore.location' has not 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 at Thu Mar 26 01:49:09 EDT 2015
/usr/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

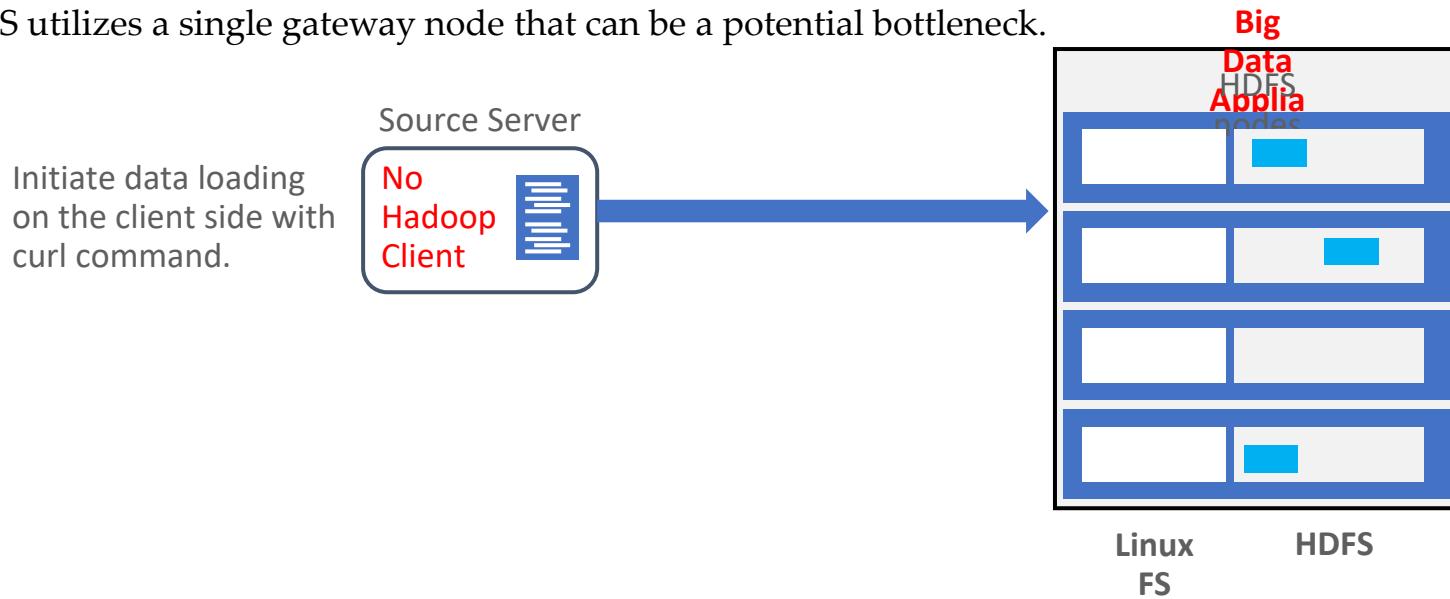
The filesystem under path '/user/oracle/wordcount/output/part-r-00000' is HEALTHY
[oracle@bigdatalite ~]$ █
```

Agenda

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

Loading Data with WebHDFS or HttpFS

- Advantages:
 - WebHDFS performance comparable with the Hadoop client
 - No additional software required on the client side
- Disadvantages:
 - Complex syntax (comparable with the Hadoop client)
 - HttpFS utilizes a single gateway node that can be a potential bottleneck.



hadoop fs -ls and LISTSTATUS

```
hadoop fs -ls
```



```
[oracle@bigdatalite ~]$ hadoop fs -ls
Found 8 items
drwxr-xr-x - oracle oracle          0 2016-05-23 20:42 .Trash
drwxr-xr-x - oracle oracle          0 2016-05-23 20:39 .sparkStaging
drwx----- - oracle oracle          0 2016-06-01 18:37 .staging
drwxr-xr-x - oracle oracle          0 2016-06-01 18:59 mediademo
drwxr-xr-x - oracle oracle          0 2016-05-15 12:02 moviedemo
drwxr-xr-x - oracle oracle          0 2016-05-15 12:03 moviework
drwxr-xr-x - oracle oracle          0 2016-05-15 12:03 oggdemo
drwxr-xr-x - oracle oracle          0 2016-05-15 12:03 oozie-oozi
[oracle@bigdatalite ~]$
```

```
curl -i
```

```
"http://bigdatalite.localdomain:50070/webhdfs/v1
/user/oracle?op=LISTSTATUS"
```



LISTSTATUS displays the same content of the
hadoop fs -ls command but in JSON format.

```
{"FileStatuses": {"FileStatus": [
    {"accessTime": 0, "blockSize": 0, "childrenNum": 1, "fileId": 25974, "group": "oracle", "length": 0, "modificationTime": 1464050554815, "owner": "oracle", "pathSuffix": ".Trash", "permission": "755", "replication": 0, "storagePolicy": 0, "type": "DIRECTORY"}, {
        {"accessTime": 0, "blockSize": 0, "childrenNum": 0, "fileId": 24648, "group": "oracle", "length": 0, "modificationTime": 1464050368869, "owner": "oracle", "pathSuffix": ".sparkStaging", "permission": "755", "replication": 0, "storagePolicy": 0, "type": "DIRECTORY"}], "accessTime": 0, "blockSize": 0, "childrenNum": 0, "fileId": 24580, "group": "oracle", "length": 0, "modificationTime": 1464820624005, "owner": "oracle", "pathSuffix": ".staging", "permission": "700", "replication": 0, "storagePolicy": 0, "type": "DIRECTORY"}, {
        {"accessTime": 0, "blockSize": 0, "childrenNum": 2, "fileId": 68152, "group": "oracle", "length": 0, "modificationTime": 1464821985653, "owner": "oracle", "pathSuffix": "mediademo", "permission": "755", "replication": 0, "storagePolicy": 0, "type": "DIRECTORY"}, {
            {"accessTime": 0, "blockSize": 0, "childrenNum": 1, "fileId": 17564, "group": "oracle", "length": 0, "modificationTime": 1463328175652, "owner": "oracle", "pathSuffix": "moviedemo", "permission": "755", "replication": 0, "storagePolicy": 0, "type": "DIRECTORY"}, {
                {"accessTime": 0, "blockSize": 0, "childrenNum": 9, "fileId": 17572, "group": "oracle", "length": 0, "modificationTime": 1463328181497, "owner": "oracle", "pathSuffix": "moviework", "permission": "755", "replication": 0, "storagePolicy": 0, "type": "DIRECTORY"}, {
                    {"accessTime": 0, "blockSize": 0, "childrenNum": 1, "fileId": 17611, "group": "oracle", "length": 0, "modificationTime": 1463328181552, "owner": "oracle", "pathSuffix": "oggdemo", "permission": "755", "replication": 0, "storagePolicy": 0, "type": "DIRECTORY"}, {
                        {"accessTime": 0, "blockSize": 0, "childrenNum": 0, "fileId": 17615, "group": "oracle", "length": 0, "modificationTime": 1463328181651, "owner": "oracle", "pathSuffix": "oozie-oozi", "permission": "755", "replication": 0, "storagePolicy": 0, "type": "DIRECTORY"}]}]}
```

Uploading a Local File to an HDFS Directory with hadoop fs

Create an HDFS directory named `test11` using hadoop fs CLI:

```
[oracle@bigdatalite ~]$ hadoop fs -mkdir test11
[oracle@bigdatalite ~]$ hadoop fs -ls
Found 10 items
drwxr-xr-x  - oracle oracle      0 2016-05-23 20:42 .Trash
drwxr-xr-x  - oracle oracle      0 2016-05-23 20:39 .sparkStaging
drwx-----  - oracle oracle      0 2016-06-01 18:37 .staging
drwxr-xr-x  - oracle oracle      0 2016-07-08 13:40 lauran
drwxr-xr-x  - oracle oracle      0 2016-06-01 18:59 mediademo
drwxr-xr-x  - oracle oracle      0 2016-05-15 12:02 moviedemo
drwxr-xr-x  - oracle oracle      0 2016-05-15 12:03 moviework
drwxr-xr-x  - oracle oracle      0 2016-05-15 12:03 oggdemo
drwxr-xr-x  - oracle oracle      0 2016-05-15 12:03 oozie-oozi
drwxr-xr-x  - oracle oracle      0 2016-07-08 13:52 test11
[oracle@bigdatalite ~]$
```

Copying the local `test1.txt` file to HDFS directory `test11` using hadoop fs CLI:

```
hadoop fs -put test1.txt
hdfs://bigdatalite.localdomain:8020/user/oracle/test11
```

```
[oracle@bigdatalite ~]$ hadoop fs -put test1.txt hdfs://bigdatalite.localdomain:8020/user/oracle/test11
[oracle@bigdatalite ~]$ hadoop fs -ls test11
Found 1 items
-rw-r--r--  1 oracle oracle      16 2016-07-08 14:03 test11/test1.txt
[oracle@bigdatalite ~]$ hadoop fs -cat test11/test1.txt
This is test1.
[oracle@bigdatalite ~]$
```

} Confirm file upload and view its content

Creating an HDFS Directory with WebHDFS

Creating an HDFS directory named `test21` by using WebHDFS:

```
curl -i -X PUT -L -H 'Content-Type:application/octet-stream'  
"http://bigdatalite.localdomain:50070/webhdfs/v1/user/oracle/test21?op=  
MKDIRS&user.name=oracle";
```

```
[oracle@bigdatalite ~]$ curl -i -X PUT -L -H 'Content-Type:application/octet-stream' "http://bigdatalite.localdomain:50070/webhdfs/v1/user/oracle/test21?op=MKDIRS&user.name=oracle";  
HTTP/1.1 200 OK  
Cache-Control: no-cache  
Expires: Fri, 08 Jul 2016 18:16:16 GMT  
Date: Fri, 08 Jul 2016 18:16:16 GMT  
Pragma: no-cache  
Expires: Fri, 08 Jul 2016 18:16:16 GMT  
Date: Fri, 08 Jul 2016 18:16:16 GMT  
Pragma: no-cache  
Content-Type: application/json  
Set-Cookie: hadoop.auth="u=oracle&p=oracle&t=simple&e=1468037776103&s=3/Lz7/Bx0FYL5SrugnxwayQFk5I="; Path=/; HttpOnly  
Transfer-Encoding: chunked  
Server: Jetty(6.1.26.cloudera.4)  
  
{"boolean":true}[oracle@hadoop fs -ls  
Found 11 items  
drwxr-xr-x - oracle oracle 0 2016-05-23 20:42 .Trash  
drwxr-xr-x - oracle oracle 0 2016-05-23 20:39 .sparkStaging  
drwx----- - oracle oracle 0 2016-06-01 18:37 .staging  

```

Uploading a Local File to HDFS with WebHDFS

Creating an HDFS directory named `test21` by using WebHDFS:

```
curl -i -X PUT -L -H 'Content-Type:application/octet-stream'  
"http://bigdatalite.localdomain:50070/webhdfs/v1/user/oracle/test21/tes  
t1.txt?op=CREATE&user.name=oracle" -T test1.txt;
```

```
[oracle@bigdatalite ~]$ curl -i -X PUT -L -H 'Content-Type:application/octet-stream'  
"http://bigdatalite.localdomain:50070/webhdfs/v1/user/oracle/test21/test1.t  
xt?op=CREATE&user.name=oracle" -T test1.txt;  
HTTP/1.1 100 Continue  
  
HTTP/1.1 307 TEMPORARY_REDIRECT  
Cache-Control: no-cache  
Expires: Fri, 08 Jul 2016 18:32:56 GMT  
Date: Fri, 08 Jul 2016 18:32:56 GMT  
Pragma: no-cache  
Expires: Fri, 08 Jul 2016 18:32:56 GMT  
Date: Fri, 08 Jul 2016 18:32:56 GMT  
Pragma: no-cache  
Set-Cookie: hadoop.auth="u=oracle&p=oracle&t=simple&e=1468038776897&s=L3iMT04D59  
QuXKKU7UtgdVVnx44="; Path=/; HttpOnly  
Location: http://bigdatalite.localdomain:50075/webhdfs/v1/user/oracle/test21/tes  
t1.txt?op=CREATE&user.name=oracle&namenoderpcaddress=bigdatalite.localdomain:802  
0&overwrite=false  
Content-Type: application/octet-stream  
Content-Length: 0  
Server: Jetty(6.1.26.cloudera.4)  
  
HTTP/1.1 100 Continue  
  
HTTP/1.1 201 Created  
Location: hdfs://bigdatalite.localdomain:8020/user/oracle/test21/test1.txt  
Content-Length: 0  
Connection: close  
  
[oracle@bigdatalite ~]$ hadoop fs -ls test21  
Found 1 items  
-rwxr-xr-x 1 oracle oracle 16 2016-07-08 14:32 test21/test1.txt  
[oracle@bigdatalite ~]$
```

Creating an HDFS Directory and Loading Data by Using HttpFS

Creating an HDFS directory named `test31` by using HttpFS and uploading `test1.txt` to `/test31` HDFS directory

```
curl -i -X PUT -L -H 'Content-Type:application/octet-stream'  
"http://bigdatalite.localdomain:14000/webhdfs/v1/user/oracle/test31/tes  
t1.txt?op=CREATE&user.name=oracle" -T test1.txt;
```

```
[oracle@bigdatalite ~]$ curl -i -X PUT -L -H 'Content-Type:application/octet-str  
eam' "http://bigdatalite.localdomain:14000/webhdfs/v1/user/oracle/test31/test1.t  
xt?op=CREATE&user.name=oracle" -T test1.txt;  
HTTP/1.1 100 Continue
```

```
HTTP/1.1 307 Temporary Redirect  
Server: Apache-Coyote/1.1  
Set-Cookie: hadoop.auth="u=oracle&p=oracle&t=simple-dt&e=1468040113065&s=rqBP4kl  
EJMUYMa68Y71BLSbMHvc="; Path=/; HttpOnly  
Location: http://bigdatalite.localdomain:14000/webhdfs/v1/user/oracle/test31/tes  
t1.txt?op=CREATE&data=true&user.name=oracle  
Content-Type: application/json  
Content-Length: 0  
Date: Fri, 08 Jul 2016 18:55:13 GMT
```

```
HTTP/1.1 100 Continue
```

```
HTTP/1.1 201 Created  
Server: Apache-Coyote/1.1  
Set-Cookie: hadoop.auth="u=oracle&p=oracle&t=simple-dt&e=1468040113091&s=HVP9fk8  
EZEPYkoYLn6nK2i2qIME="; Path=/; HttpOnly  
Content-Type: application/json  
Content-Length: 0  
Date: Fri, 08 Jul 2016 18:55:13 GMT
```

```
[oracle@bigdatalite ~]$ hadoop fs -ls test31  
Found 1 items  
-rwxr-xr-x 1 oracle oracle 16 2016-07-08 14:55 test31/test1.txt  
(reverse-i-search) :
```

HttpFS uses default port 14000

Summary

- In this lesson, you should have learned how to:
 - Describe the architectural components of HDFS
 - Use the **FS shell** command-line interface (CLI) to interact with data stored in HDFS

Acquiring Data by Using Kafka



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 5: Introduction to the Hadoop Distributed File System (HDFS)

Lesson 6: Acquiring Data by Using CLI, Fuse DFS, Flume, and Kafka

Lesson 07: Acquiring and Accessing Data by Using Oracle NoSQL Database



Objectives

- After completing this lesson, you should be able to describe:
 - Uses of the command-line interface (CLI)
 - Benefits of Fuse DFS
 - Architecture and features of Flume
 - Features and uses of 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.

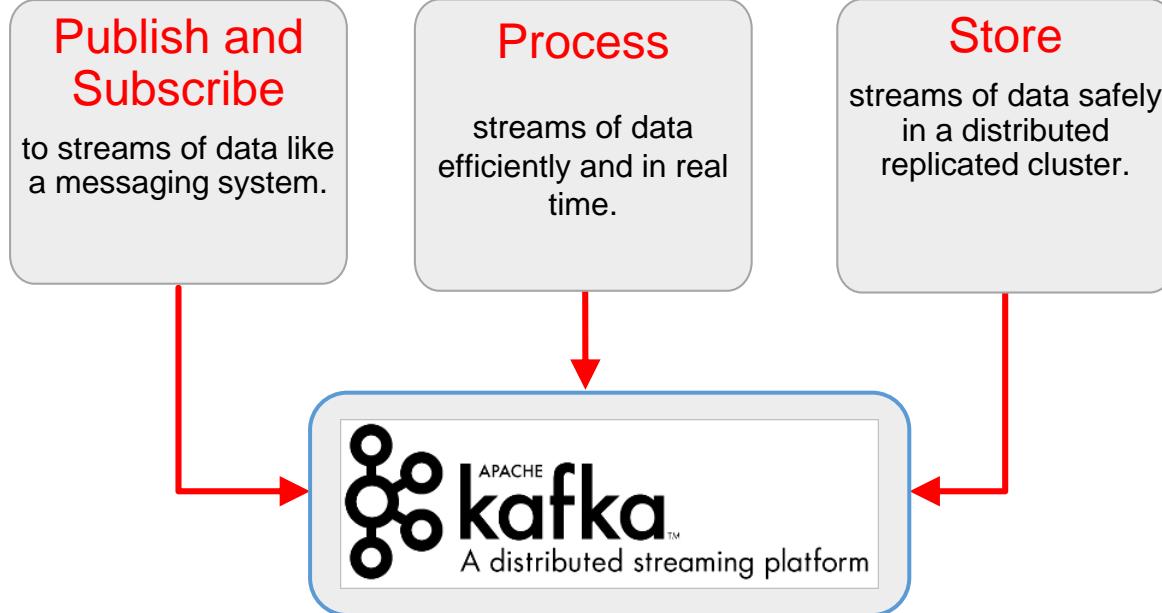


Activity Log Processing in Real Time

- 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.



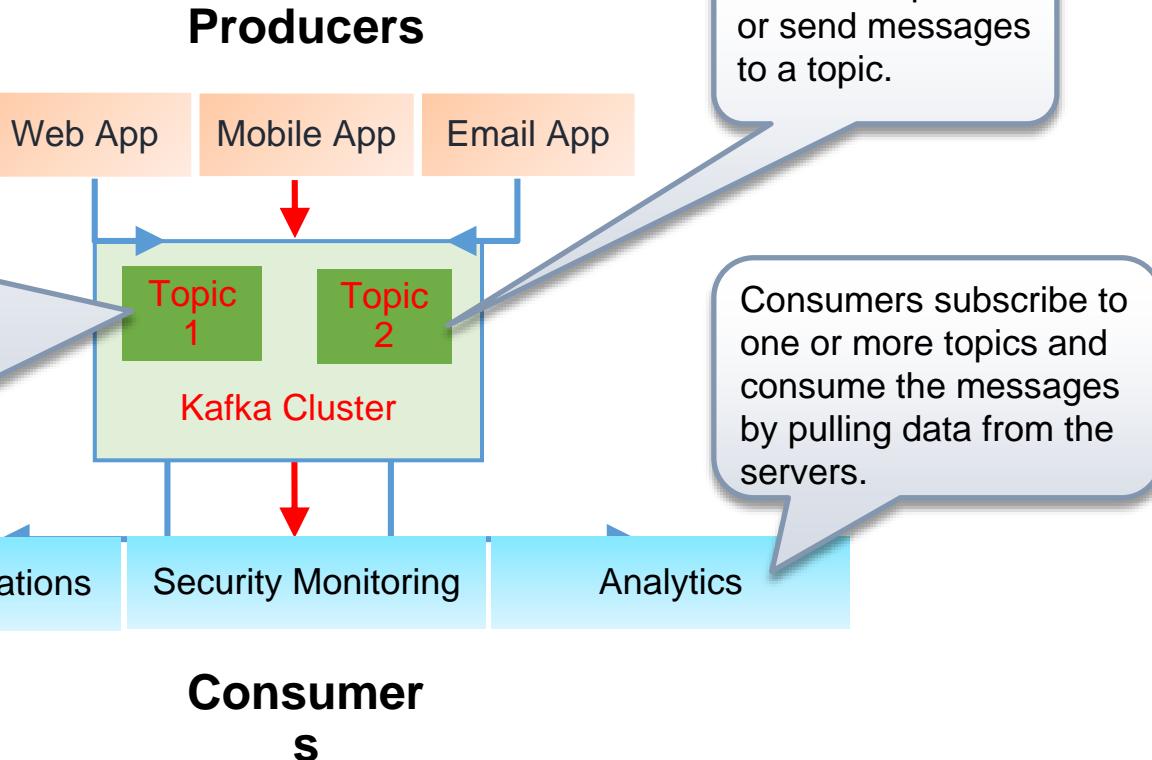
Apache Kafka: Features



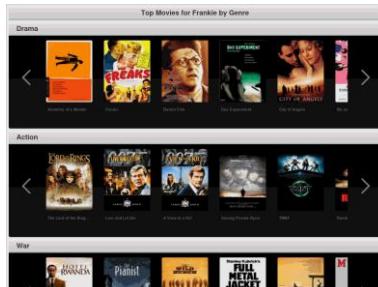
Apache Kafka: Basic Concepts

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.



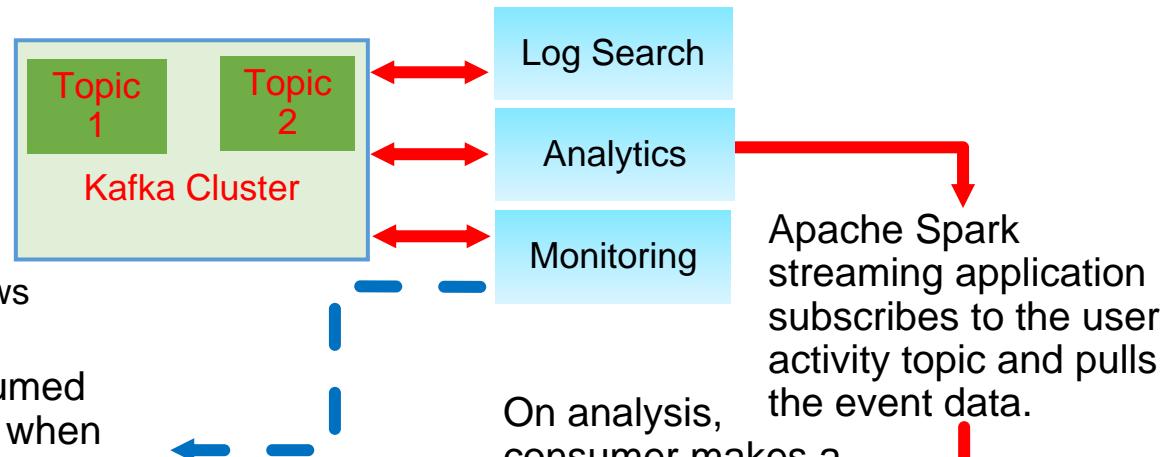
Using Apache Kafka: Oracle MoviePlex Examples



Stream of
Movie Views



Example 1: User guest1 watched *The Incredibles*.



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!**

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

Additional Resources

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

Summary

- In this lesson, you should have learned to describe:
 - Uses of the command-line interface (CLI)
 - Benefits of Fuse DFS
 - Architecture and features of Flume
 - 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 Oracle 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”).



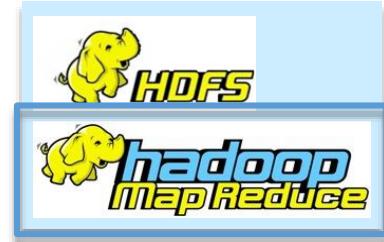
MapReduce Framework: Features

- 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

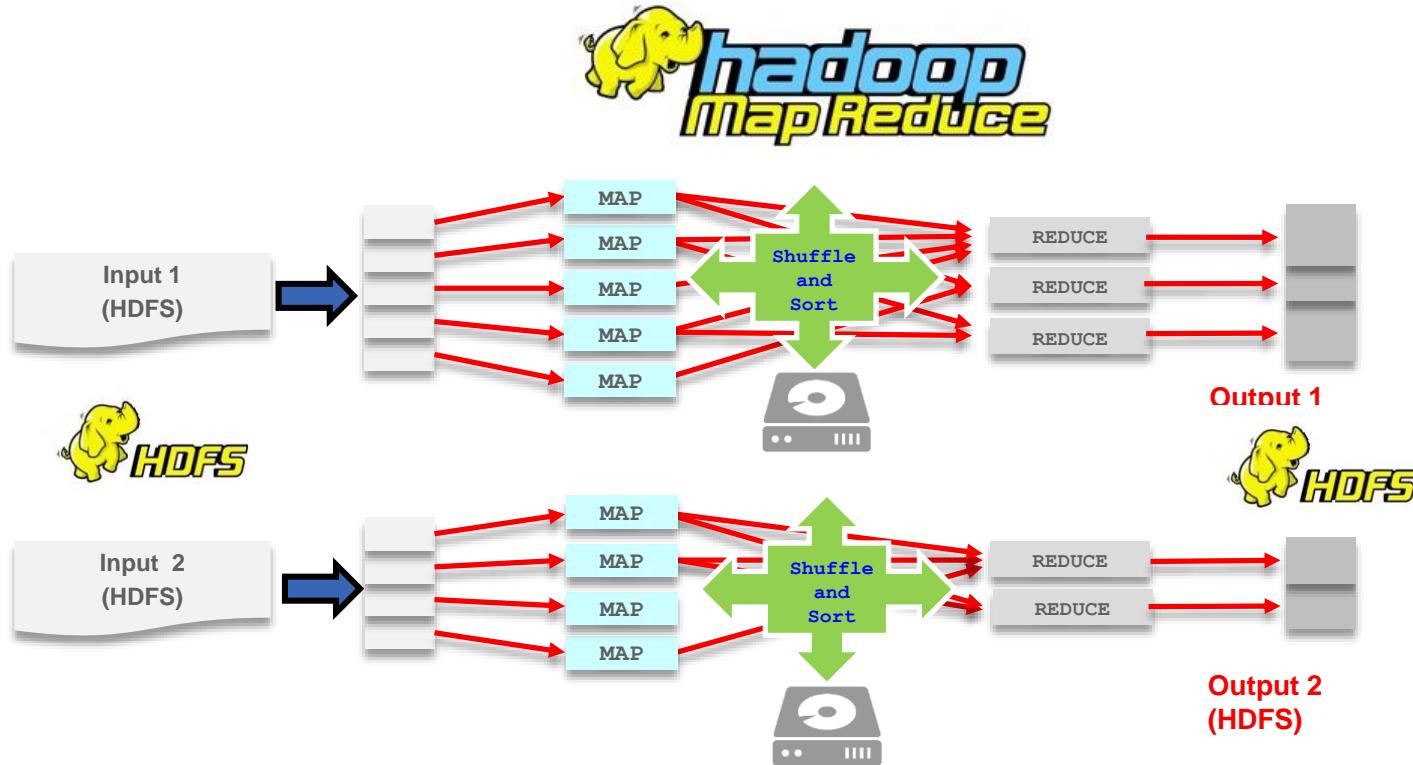


Benefits of MapReduce

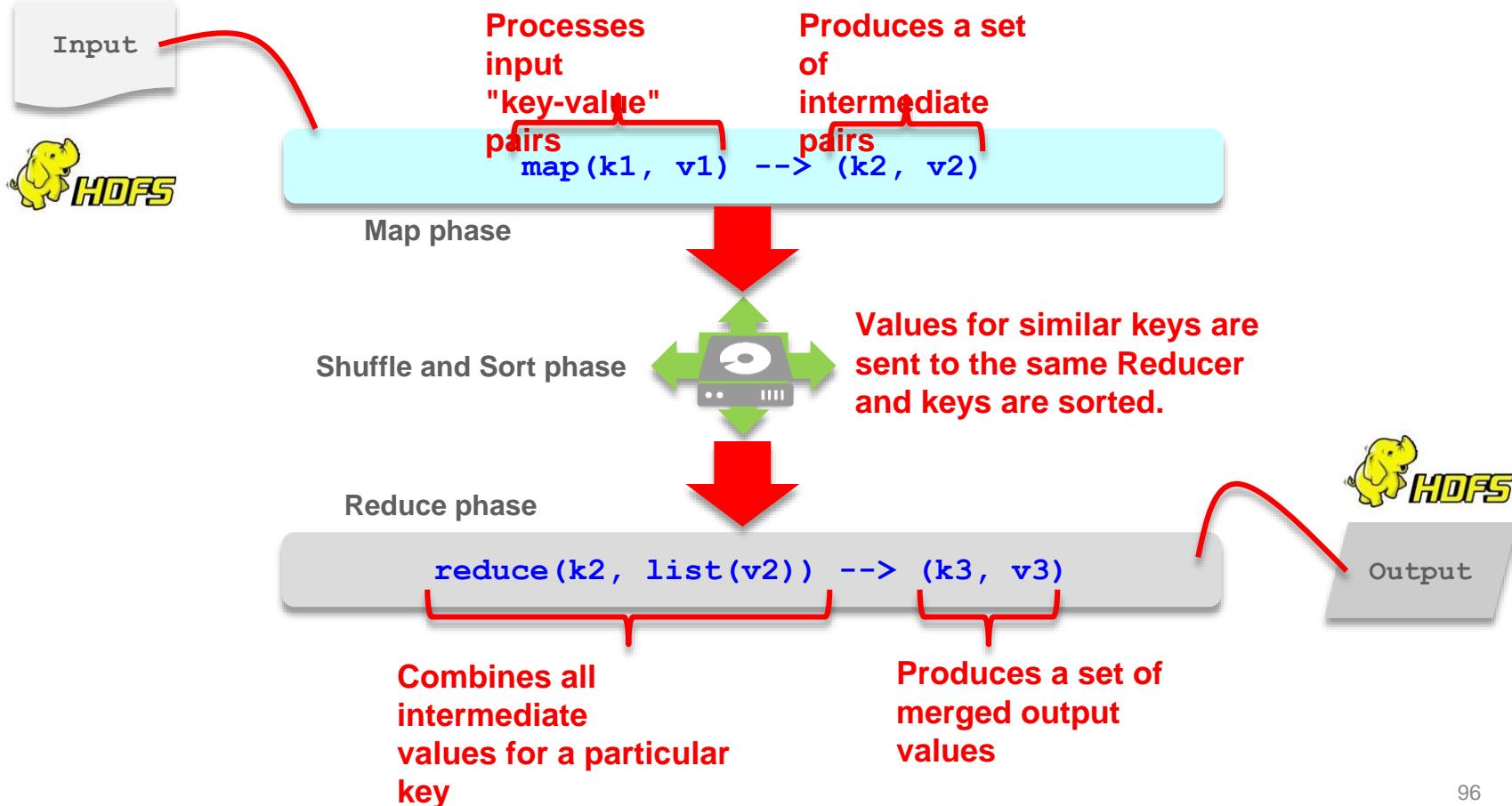
- MapReduce provides:
 - Automatic parallelization and distribution of large data sets that are stored and distributed on Hadoop cluster follower nodes
 - Fault-tolerance
 - I/O scheduling
 - Status and monitoring



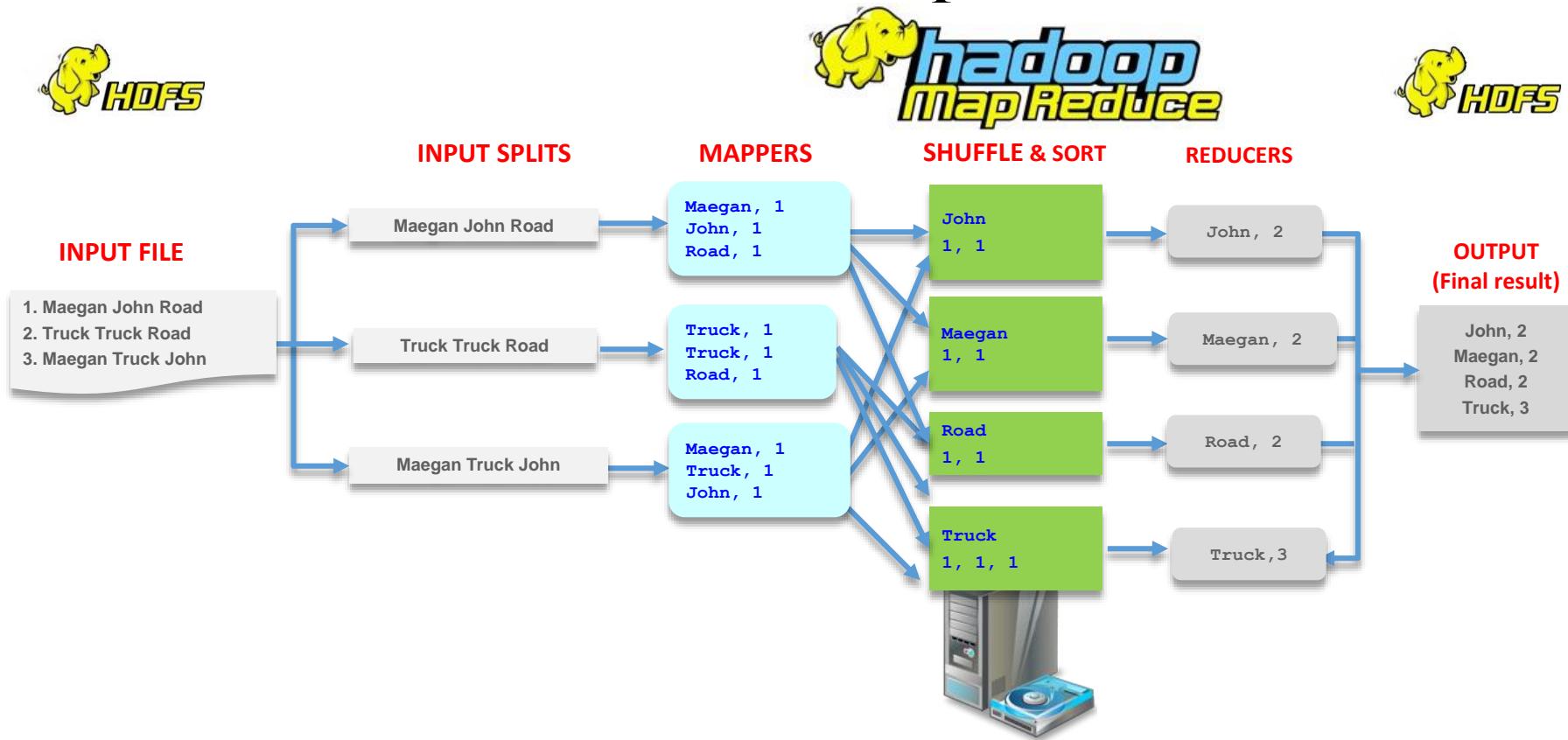
MapReduce Jobs



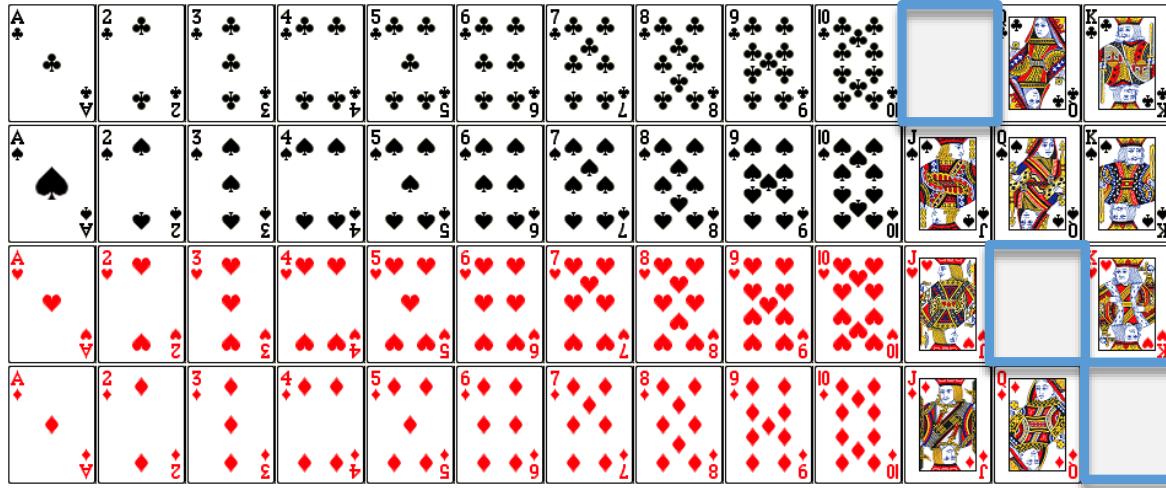
Parallel Processing with MapReduce



Word Count Process: Example 1



MapReduce Mechanics: Deck of Cards Example

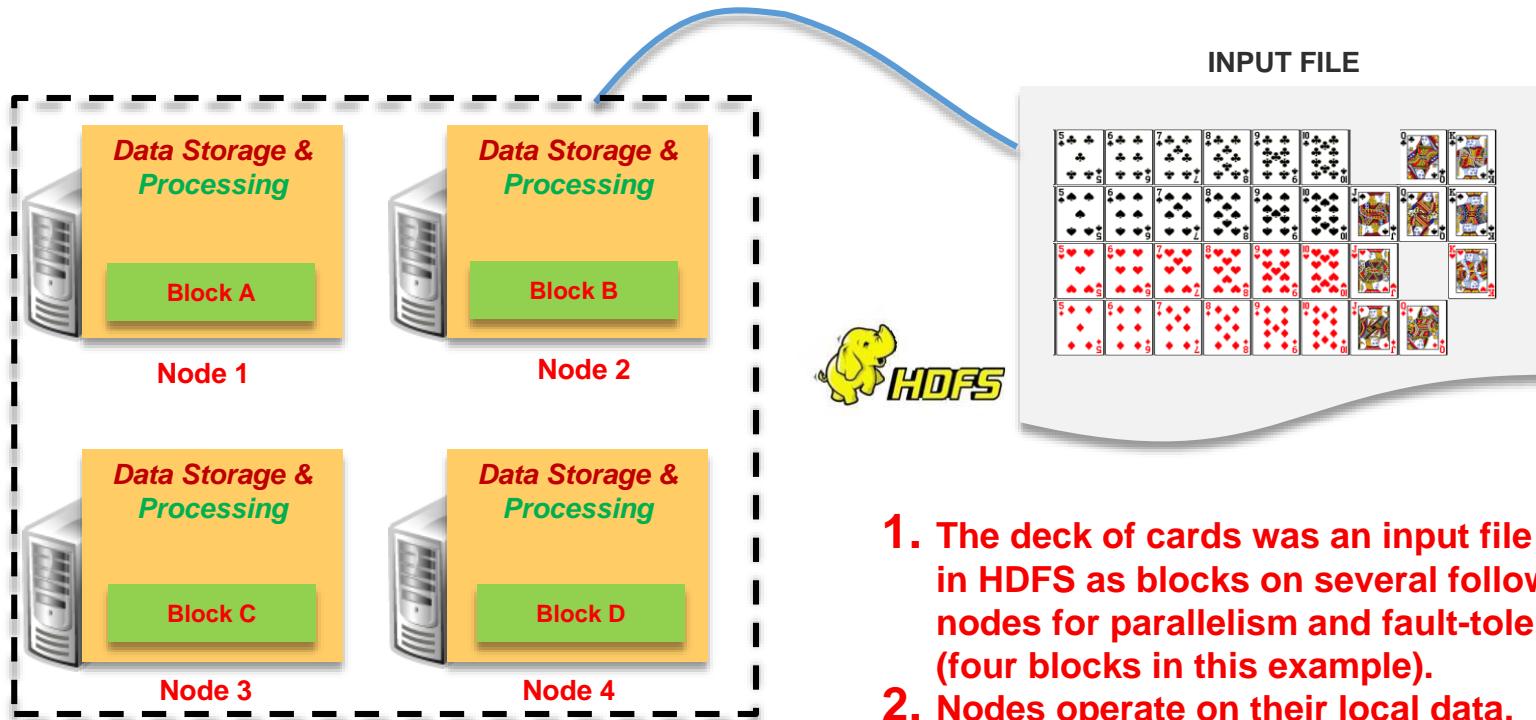


Suppose three face cards are removed from the deck of playing cards.

How do you find the suits whose face cards are short by using MapReduce?

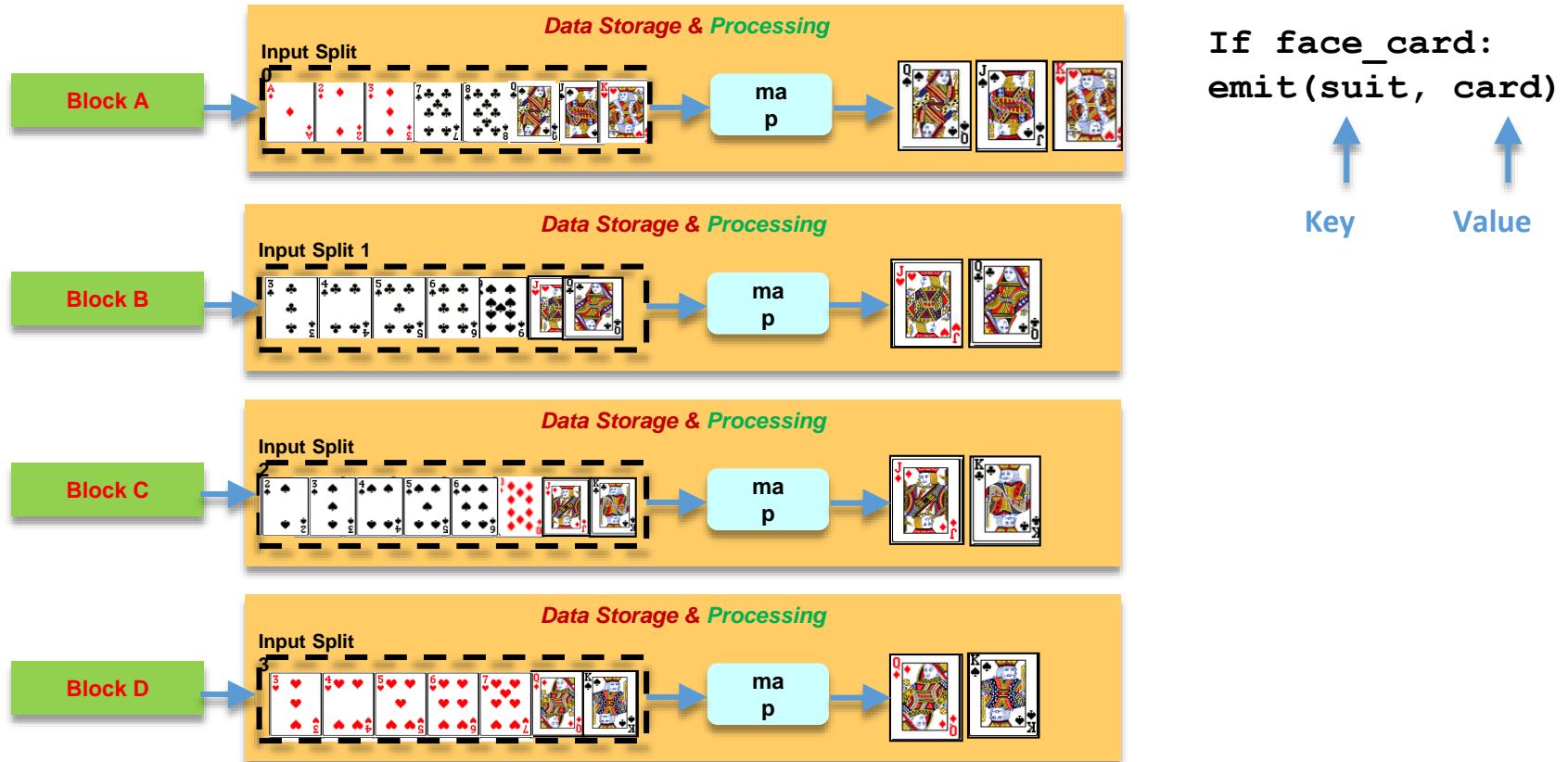


MapReduce Mechanics Example: Assumptions



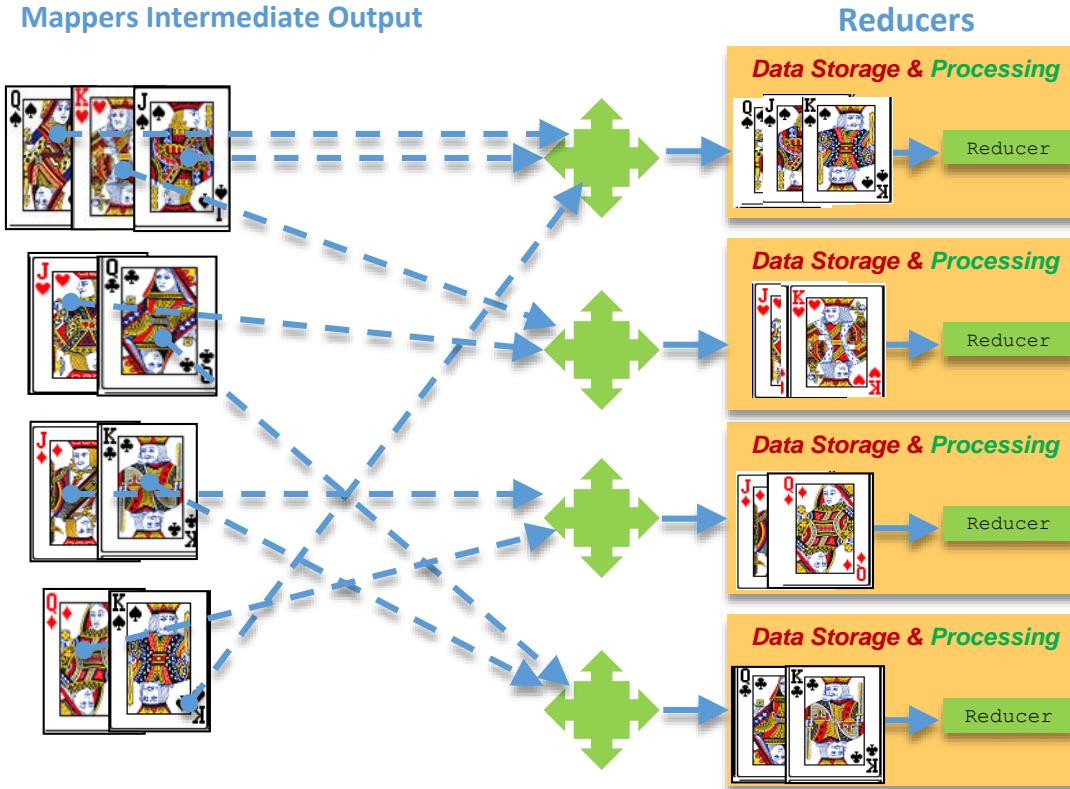
1. The deck of cards was an input file stored in HDFS as blocks on several follower nodes for parallelism and fault-tolerance (four blocks in this example).
2. Nodes operate on their local data.

MapReduce Mechanics: Map Phase

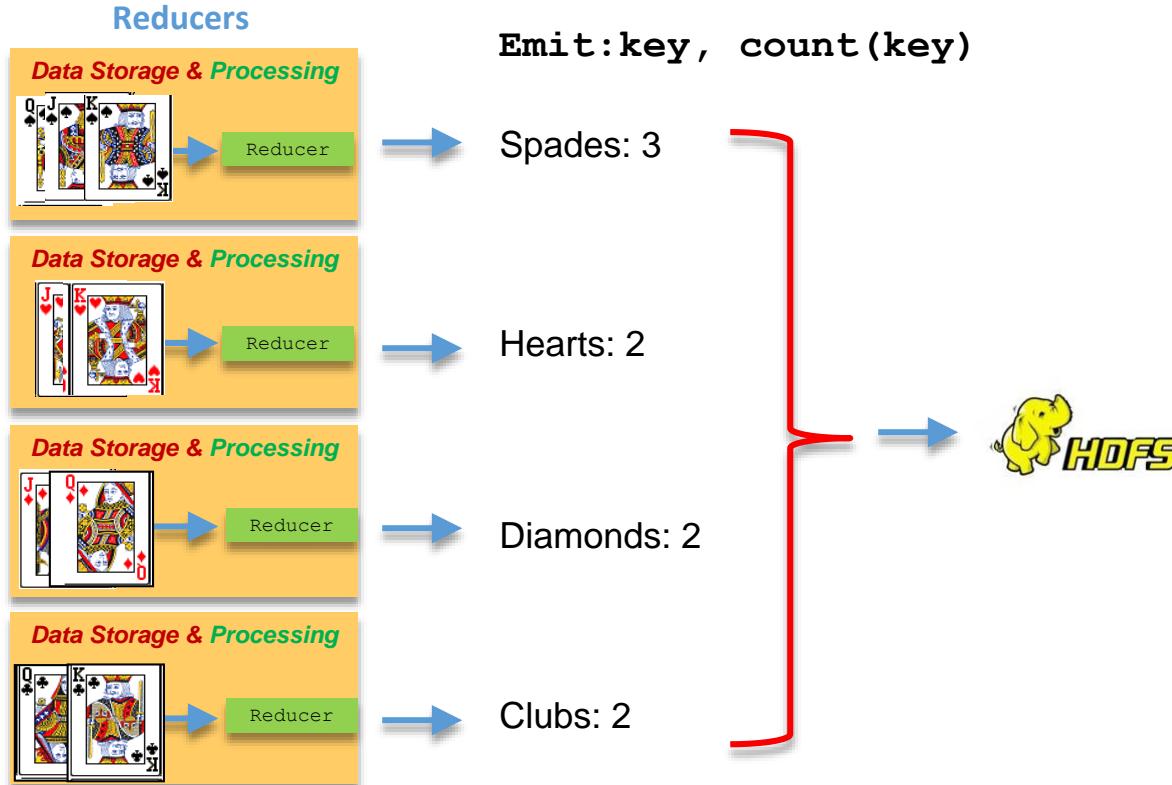


MapReduce Mechanics: Shuffle and Sort Phase

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



MapReduce Mechanics: Reduce Phase (Result)

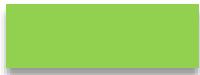


Interacting with MapReduce

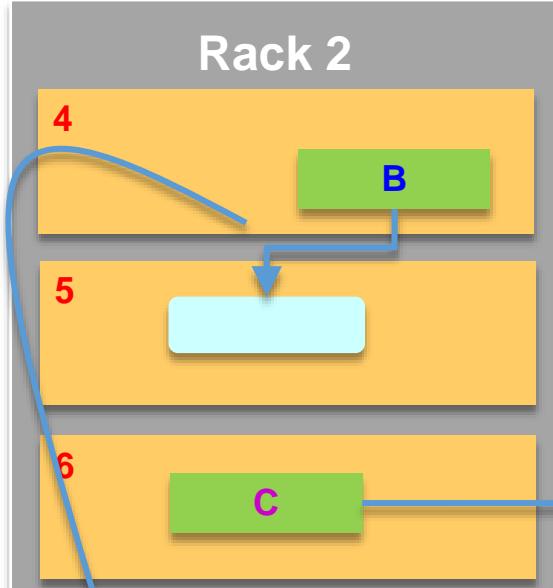
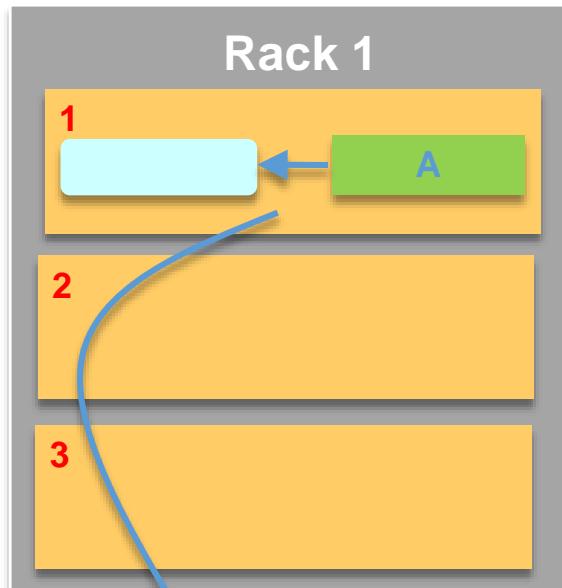
- MapReduce code can be written in Java, C, and scripting languages.
- Higher-level abstractions (Hive, Pig) enable easy interaction.
 - Optimizers construct MapReduce jobs.
- Code is:
 - Submitted to the JobTracker daemons on the Leader node and executed by the TaskTrackers on the follower nodes with 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

HDFS blocks:



Map tasks:



Node-local map tasks

Rack-local map tasks

Off-rack map tasks

Submitting a MapReduce Job

```
hadoop jar WordCount.jar WordCount  
/user/oracle/wordcount/input  
/user/oracle/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/oracle/wordcount/input</code> | The input directory |
| <code>/user/oracle/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/oracle/wordcount/input \
/usr/oracle/wordcount/output
```



```
oracle@bigdatalite:~/exercises/wordCount
File Edit View Search Terminal Help
input/file02
[oracle@bigdatalite wordCount]$ hadoop jar WordCount.jar WordCount /user/oracle/wordcount/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_1472055509950_0001
16/08/24 12:23:39 INFO impl.YarnClientImpl: Submitted application application_1472055509950_0001
16/08/24 12:23:40 INFO mapreduce.Job: The url to track the job: http://bigdatalite.localdomain: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 successfully
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

All Applications

localhost:8088/cluster

Hadoop Spatial and Graph BDD Apex ORDS Lab Solr Admin Pattern Matching Cloudera Manager

NameNode bigdatalite.localdomain:8020

YARN Applications

JobHistory

All Applications

Cluster Metrics

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Memory Used | Memory Total | Memory Reserved | Vcores Used | Vcores Total | Vcores Reserved | Active Nodes |
|----------------|--------------|--------------|----------------|--------------------|-------------|--------------|-----------------|-------------|--------------|-----------------|--------------|
| 1 | 0 | 0 | 1 | 0 | 0 B | 8 GB | 0 B | 0 | 8 | 0 | 1 |

User Metrics for dr.who

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Containers Pending | Containers Reserved | Memory Used | Memory Pending |
|----------------|--------------|--------------|----------------|--------------------|--------------------|---------------------|-------------|----------------|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 B | 0 B |

Show 20 entries

| ID | User | Name | Application Type | Queue | StartTime | FinishTime | State | FinalStatus | Run Cont |
|-------------------------------|--------|-----------|------------------|-------------|--------------------------------|--------------------------------|----------|-------------|----------|
| application_147205509950_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 |

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 Job History 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 Hadoop JobHistory Web UI. The URL in the address bar is `http://bigdatalite.localdomain:19888/jobhistory`. The page title is "JobHistory". On the left, there's a sidebar with a "hadoop" logo and links for "Application" (About Jobs, Tools), "Retired Jobs" (Show 20 entries), and a search bar. The main content area displays a table of a single job entry:

| 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 12:23:52 EDT 2016 | 2016.08.24 12:24:19 | job_1472055509950_0001 | WordCount | oracle | root.oracle | SUCCEEDED | 2 | 2 | 1 |

Below the table is a footer with links for "Submit Time", "Start Time", "Finish Time", "Job ID", "Name", "User", "Queue", "State", "Maps", "Maps Compl", and "Reduces". The footer also shows "Showing 1 to 1 of 1 entries".

Viewing the WordCount.java Program Output

The image displays two terminal windows side-by-side, both titled "oracle@bigdatalite:~".

The left terminal window shows the command `hadoop fs -ls /user/oracle/wordcount/output` being run. The output lists two items:

```
[oracle@bigdatalite ~]$ hadoop fs -ls /user/oracle/wordcount/output
Found 2 items
-rw-r--r-- 1 oracle oracle          0 2016-08-24 12:24 /user/oracle/wordcount/
output/_SUCCESS
-rw-r--r-- 1 oracle oracle     208 2016-08-24 12:24 /user/oracle/wordcount/
output/part-r-00000
[oracle@bigdatalite ~]$
```

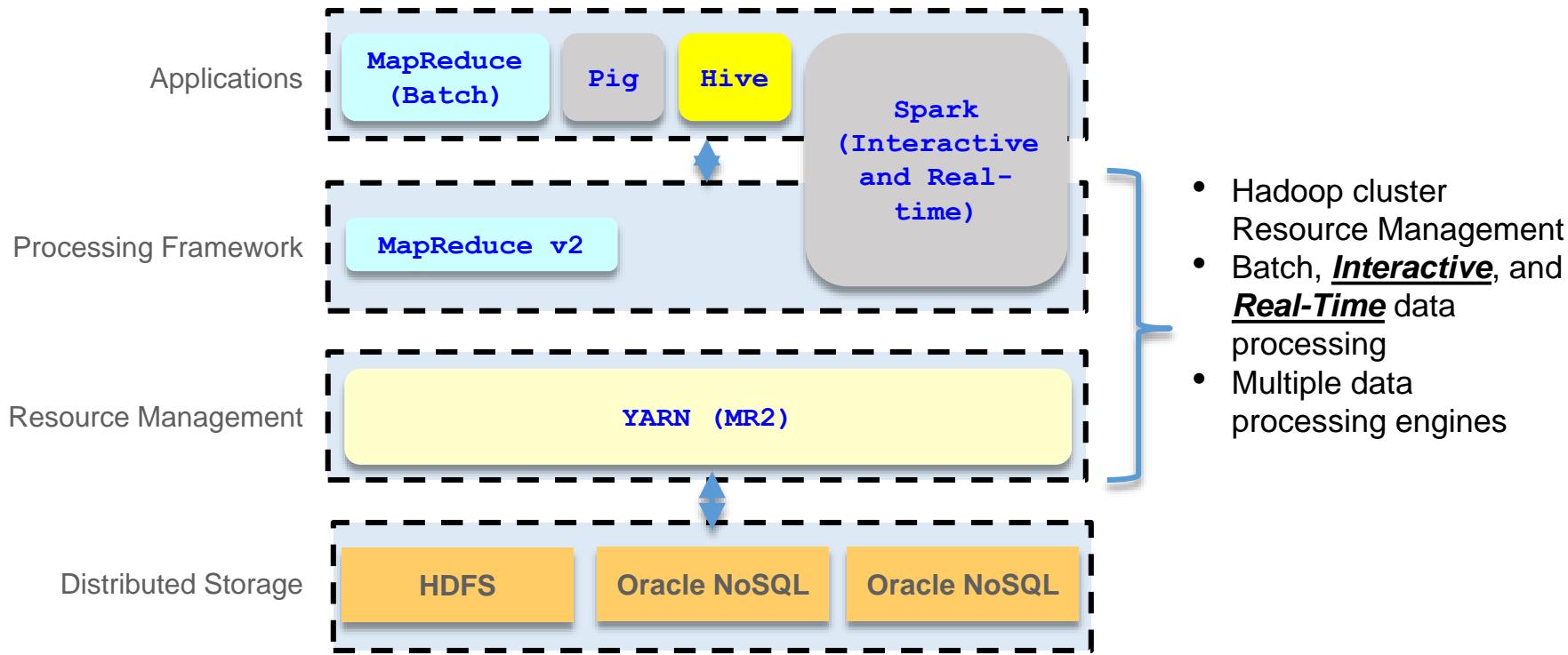
The right terminal window shows the command `hadoop fs -cat /user/oracle/wordcount/output/part-r-00000` being run. The output is a list of words and their counts:

```
[oracle@bigdatalite ~]$ hadoop fs -cat /user/oracle/wordcount/output/part-r-00000
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
[oracle@bigdatalite ~]$
```

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.

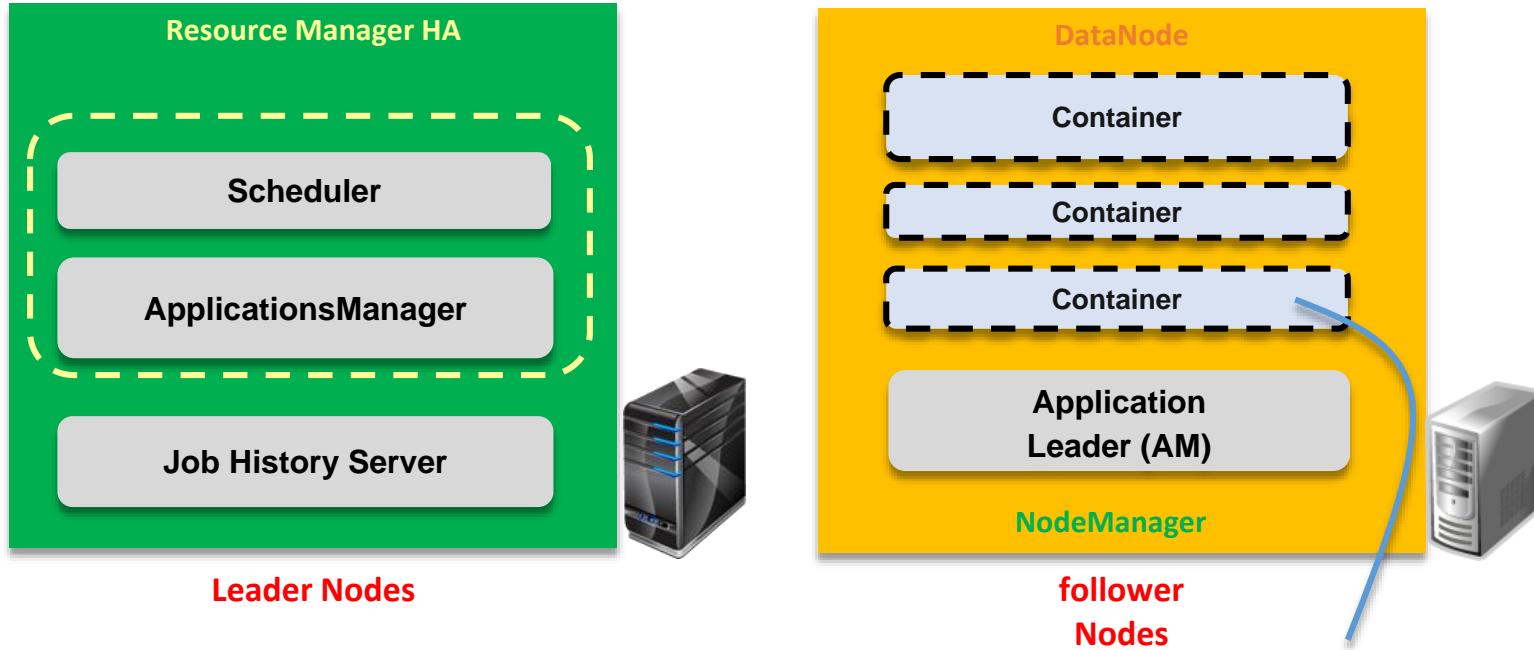
YARN: Features

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

YARN Daemons

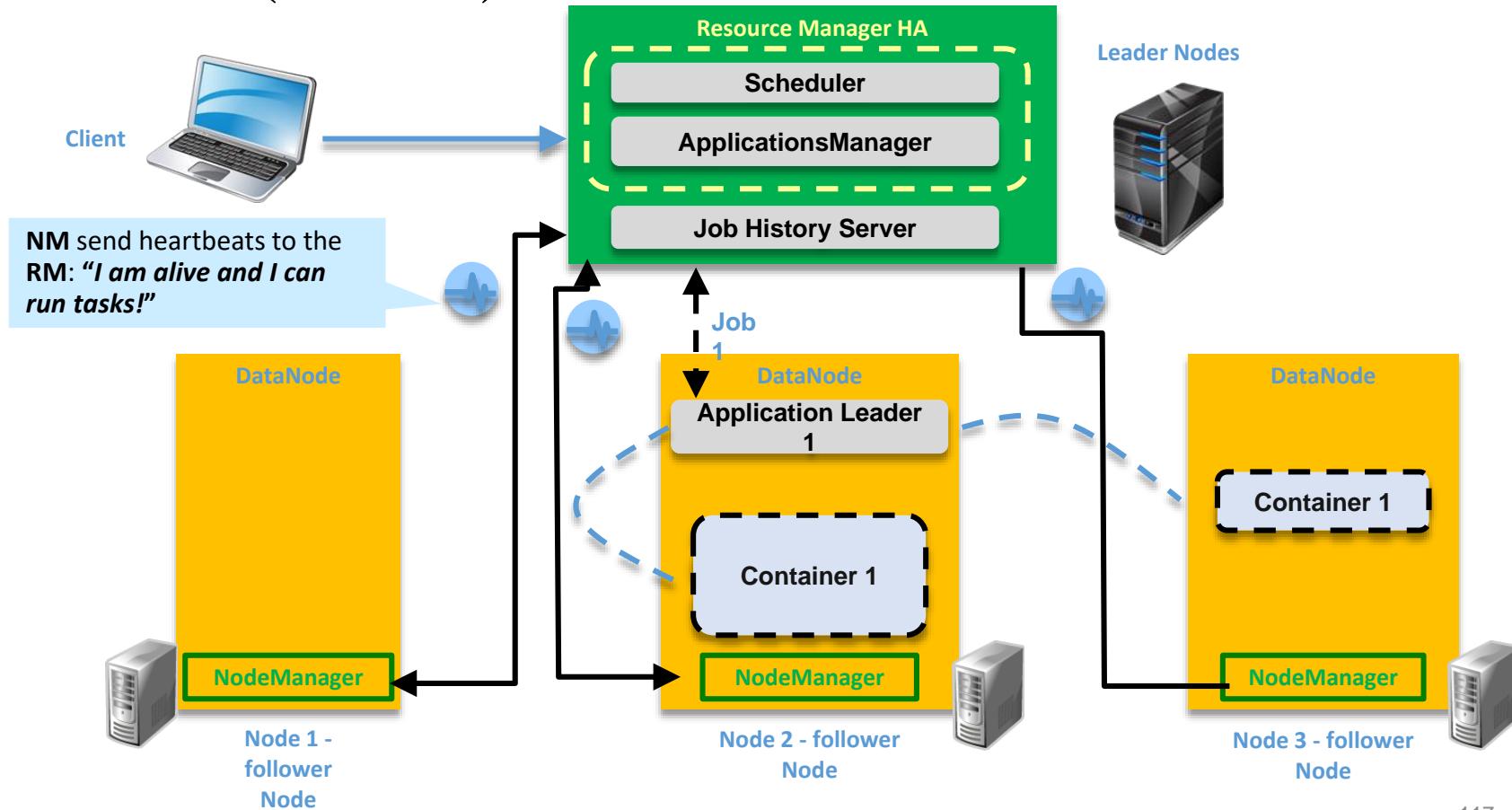
| Component | Description |
|-------------------------------|--|
| ResourceManager (RM) | <ul style="list-style-type: none">• A dedicated scheduler that allocates resources to the requesting applications• The RM has two main components: Scheduler and ApplicationsManager• It is a critical component of 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

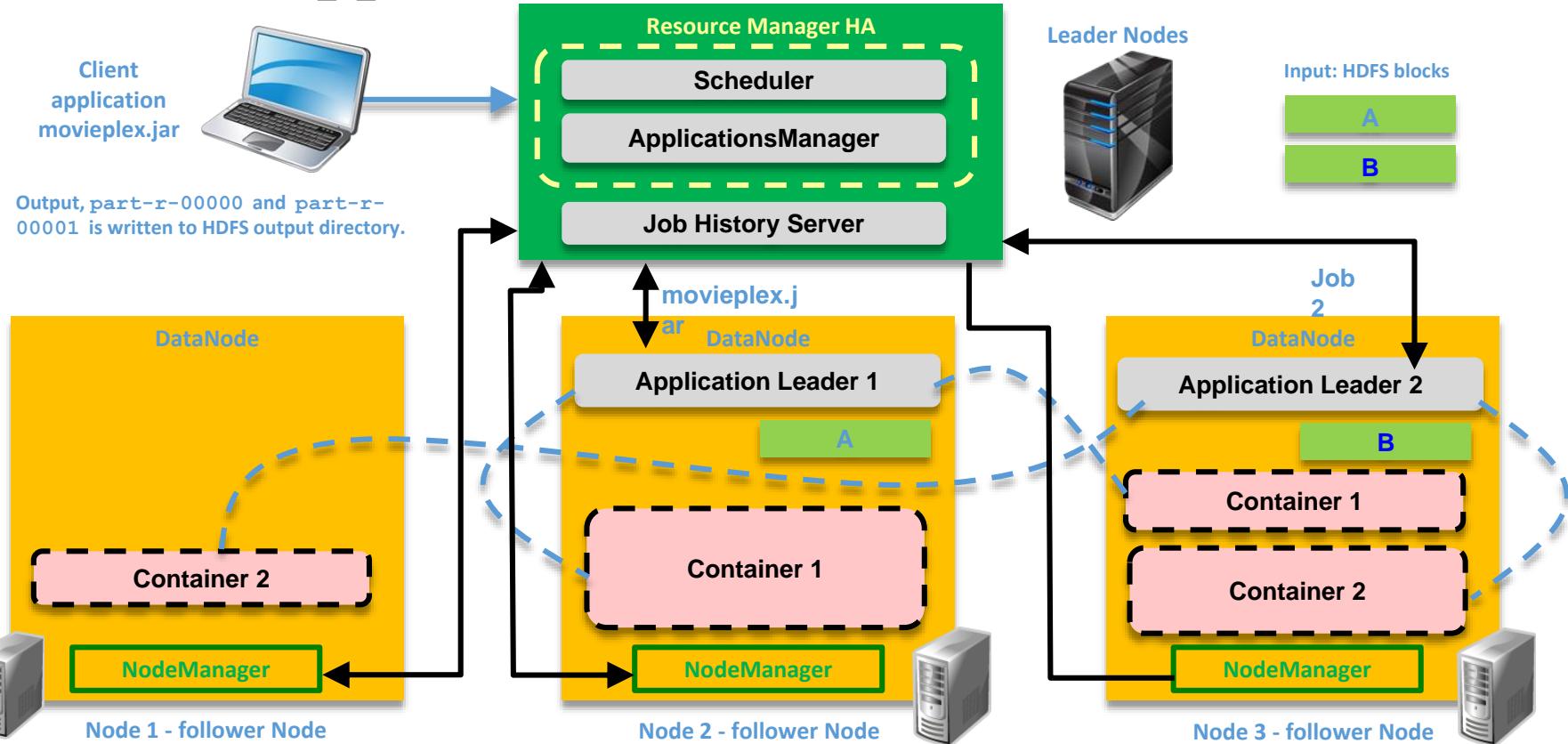


Application Leader Containers specifications use the physical world approach such as CPU and cores, memory, disk space, and network Bandwidth (instead Of the Map and Reduce slots in MapReduce V1)

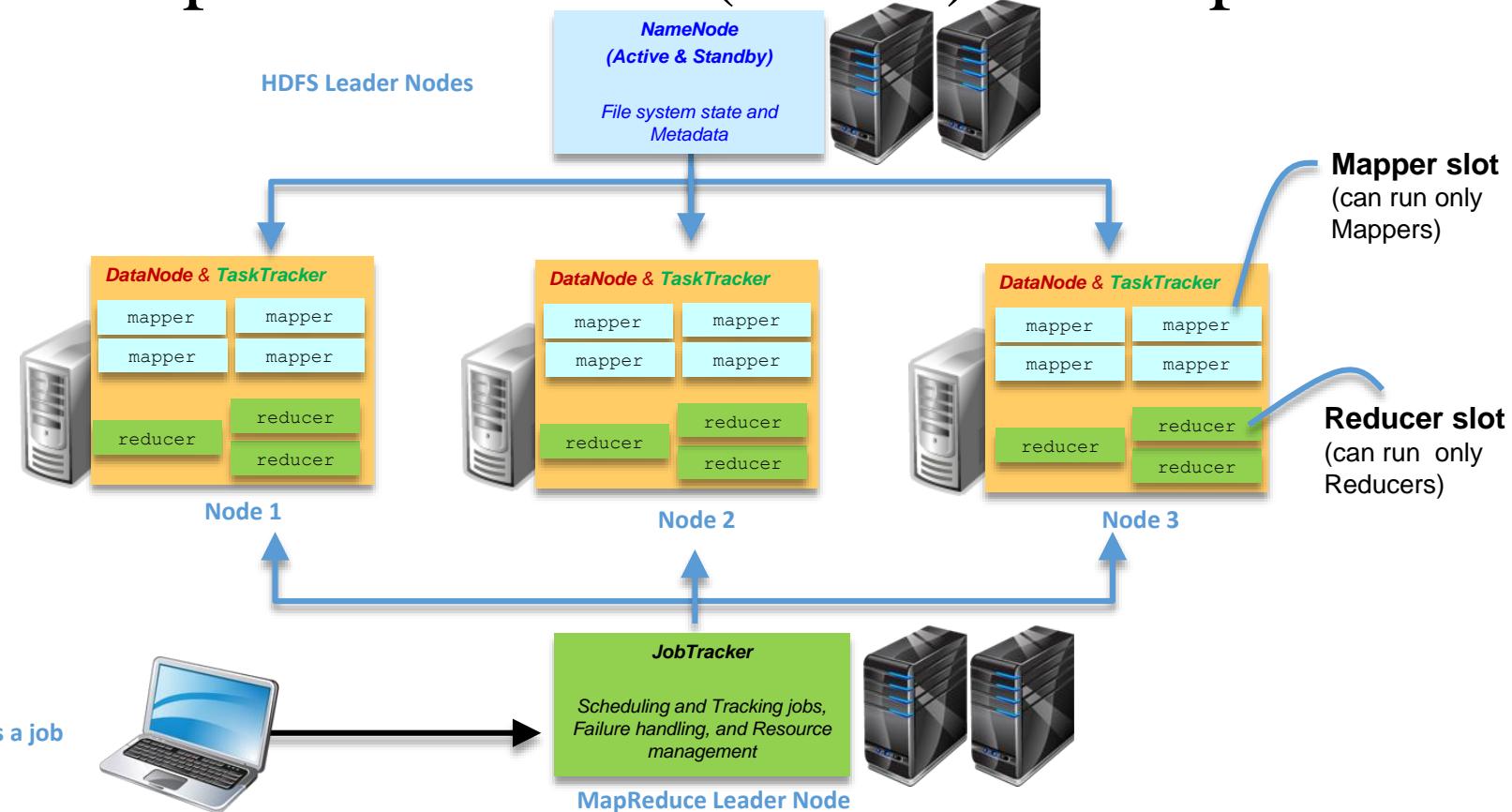
YARN (MRv2) Architecture



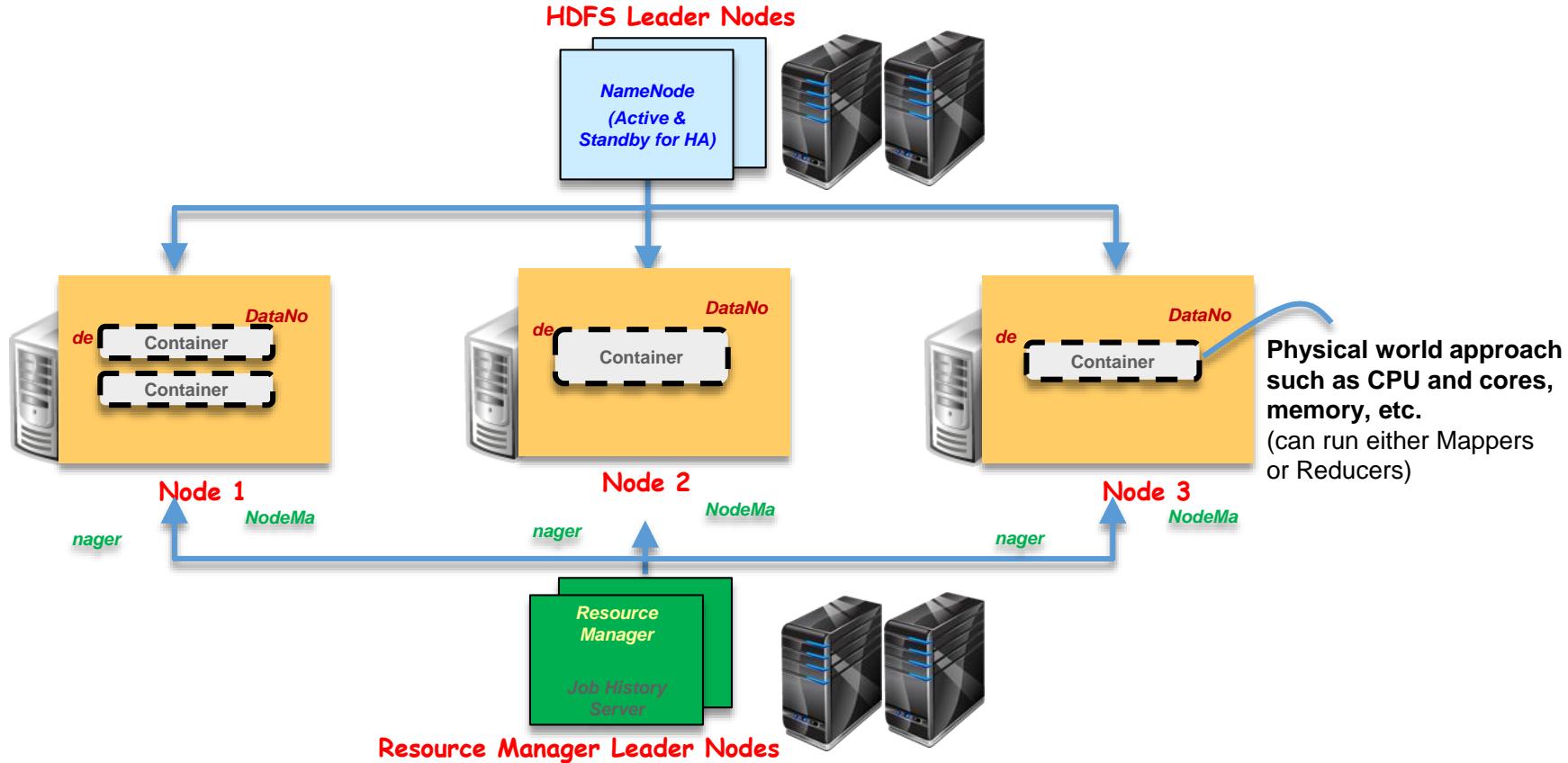
YARN Application Workflow



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

Resource Management Using YARN



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

Lesson 9: Resource Management Using YARN

Lesson 10: Apache Spark

Lesson 11: Overview of Apache Hive

Lesson 12: Overview of Cloudera Impala

Lesson 13: Using Oracle XQuery for Hadoop

Lesson 14: Overview of Solr



Objectives

- After completing this lesson, you should be able to do the following:
 - Use YARN to manage resources efficiently
 - Use the YARN application command

Agenda

- Job Scheduling in YARN
- YARN application command

Job Scheduling in YARN

- YARN provides a pluggable model to schedule policies. The scheduler is responsible for deciding where and when to run tasks. YARN supports the following pluggable schedulers:
 - First In, First Out (FIFO)
 - Allocates resources based on arrival time
 - Capacity Scheduler
 - Allocates resources to pools, with FIFO scheduling within each pool
 - Default in Hadoop
 - Fair Scheduler
 - Allows YARN applications to share resources in large clusters fairly
 - This course focuses on the Fair Scheduler
 - Default in CDH5 (used in Oracle BDA)

YARN Fair Scheduler

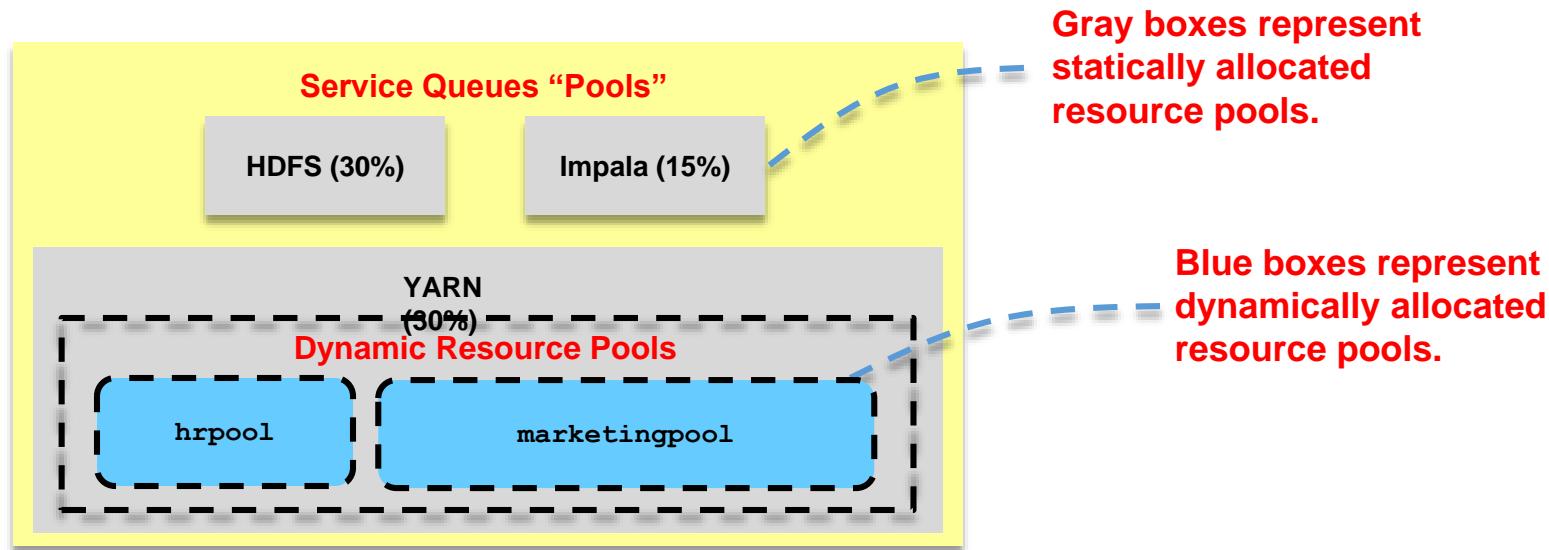
- It is a pluggable scheduler that allows YARN applications to share resources in large clusters fairly.
- In the YARN Fair Scheduler, the terms “queue” and “pool” are used interchangeably.
- It assigns resources to applications so that all applications get an equal share of allocated resources over time.
- It bases scheduling fairness decisions only on memory but you can configure it to be based on both memory and CPU.
- It organizes applications into “queues” (pools), and shares resources fairly between these queues. Every application belongs to a queue (“default” queue is the default).
- It grants YARN containers to the queue with the least amount of allocated resources.

YARN Fair Scheduler

- It works also with application priorities. The priorities are used as weights to determine the fraction of total resources that each application must get.
- If an application specifically lists a queue in a container resource request, the request is submitted to that queue.
- The Fair Scheduler supports moving a running application to a different queue, for example:
 - Moving an important application to a higher priority queue
 - Moving an unimportant application to a lower priority queue
- The Fair Scheduler also enables you to assign guaranteed minimum shares to queue.

Cloudera Manager Resource Management: Features

- Cloudera Manager provides the following features to assist you with allocating cluster resources to services:
 - Static allocation (percentage of cluster resources)
 - Dynamic allocation



Static Service Pools

The image illustrates the steps to set up Static Service Pools in Cloudera Manager:

- Step 1: Clusters Menu** (Numbered 1)
The 'Clusters' dropdown menu is highlighted. A sub-menu is open, showing 'Dynamic Resource Pools' and 'Static Service Pools'. The 'Static Service Pools' option is selected.
- Step 2: Status Tab** (Numbered 2)
The 'Status' tab of the 'Static Service Pools' page is selected. It displays overall CPU and memory usage across all hosts.
- Step 3: Basic Allocation Setup** (Numbered 3)
The 'Basic' allocation setup step is shown. It lists services (HBase, Impala, bigdatasql, hdfs, yarn) with their current allocation percentages. The total allocation is 0%. A help panel explains the purpose of static service pools and provides instructions for starting the setup.
- Step 4: Summary Step** (Numbered 4)
The final step of the wizard shows a summary of the configuration. It includes a navigation bar with steps 1-4, a 'Back' button, and a 'Continue' button.
- Step 5: Configuration Tab** (Numbered 5)
The 'Configuration' tab of the 'Static Service Pools' page is selected, showing a table of service resource usage.

| Service | CPU % | Memory |
|---------|-------|-----------|
| HBase | 0.51 | 158.7 GiB |
| Impala | 0.00 | 6.1 GiB |
| hdfs | 0.02 | 6.8 GiB |
| yarn | 0.01 | 1.9 GiB |

Working with the Fair Scheduler

- Creating Dynamic Resource Pools
- Assigning Priorities to Dynamic Resource Pools
- Monitoring Jobs

Cloudera Manager Dynamic Resource Management: Example

```
# Linux commands for creating the groups and
# users in the new groups which are used in the
# on the next few slides.
groupadd marketing
groupadd hr
groupadd development
useradd -r -g marketing bob
useradd -r -g hr lucy
. . .
```

Cloudera Manager Dynamic Resource Management: Example

The screenshot shows the Cloudera Manager web interface. The top navigation bar includes Home, Clusters (which is highlighted with a blue box), Hosts, Diagnostics, Audits, Charts, Backup, and Administration. The main content area displays the 'scaj51cdh (CDH 5.3.0)' cluster. On the left, there's a sidebar with 'Status' and 'Config' sections, and a 'Services' list containing HBase, Impala, KMS (File), bigdatasql, hdfs, hive, hue, oozie, sentry, yarn, and zookeeper. Below this is a 'Categories' section. The right side features a 'General' section with links for Hosts, Reports, Activities, YARN Applications, and Impala Queries. Under 'Resource Management', two options are listed: 'Dynamic Resource Pools' (which is highlighted with a blue box and has a cursor icon over it) and 'Static Service Pools'. A large button labeled 'Save Changes' is visible on the right. A message box on the right says 'Switch to the new layout' and 'Save Changes'.

Cloudera Manager Dynamic Resource Management: Example

cloudera manager

Home Clusters Hosts Diagnostics Audits Charts Backup Administration

scaj51cdh »

Dynamic Resource Pools Status Configuration

Resource Pools Scheduling Rules Placement Rules User Limits Other Settings

Applications can run in a pool based on the user, the group of the submitting user, as well as specific pools and the default pool.

Allocate resources across pools using weights, minimum, and maximum limits. Configuration sets allow switching on different weight and limit settings activated by user-defined schedules.

Pools can be nested, each level of which can support a different scheduler, such as FIFO or fair scheduler. Each pool can be configured to allow only a certain set of users and groups to access the pool.

| Name | Weight % | Virtual Cores Min / Max | Memory Min / Max | Max Running Apps | Scheduling Policy | Configuration Sets |
|---------------|-----------|----------------------------|---------------------|------------------|-------------------|-------------------------------------|
| root | 1 100.0% | - / - | - / - | - | DRF | <input type="button" value="Edit"/> |
| default | 1 0.9% | - / - | - / - | - | DRF | <input type="button" value="Edit"/> |
| hrpool | 10 9.0% | 5 / 25 | 100MB / 10000MB | 5 | DRF | <input type="button" value="Edit"/> |
| marketingpool | 100 90.1% | 25 / 50 | 500MB / 5000MB | 10 | DRF | <input type="button" value="Edit"/> |

Cloudera Manager Dynamic Resource Management: Example

| Name | YARN | | | | | | |
|---------------|-----------|----------------------------|---------------------|------------------|-------------------|--|--|
| | Weight % | Virtual Cores Min / Max | Memory Min / Max | Max Running Apps | Scheduling Policy | | |
| root | 1 100.0% | - / - | - / - | - | DRF | <input checked="" type="checkbox"/> Edit ▾ | |
| default | 1 0.9% | - / - | - / - | - | DRF | <input checked="" type="checkbox"/> Edit ▾ | |
| hrpool | 10 9.0% | 5 / 25 | 100MB / 10000MB | 5 | DRF | <input checked="" type="checkbox"/> Edit ▾ | |
| marketingpool | 100 90.1% | 25 / 50 | 500MB / 5000MB | 10 | DRF | <input checked="" type="checkbox"/> Edit ▾ | |

Edit Resource Pool: hrpool

General [YARN](#) [Submission Access Control](#) [Administration Access Control](#)

Resource Pool Name Alphanumeric characters only.

Scheduling Policy DRF: Dominant Resource Fairness. Schedules resources fairly based on both CPU and memory. (Recommended)
 FAIR: Schedules resources fairly based only on memory.
 FIFO: First in, first out.

Cloudera Manager Dynamic Resource Management: Example

Edit Resource Pool: hrpool

General **YARN** Submission Access Control Administration Access Control

Multiple configuration sets allow you to specify different settings based on your schedule.

| default | Weight | Share of resources relative to other pools. |
|---------|---------------------------|--|
| | 10 | |
| | Virtual Cores (Min / Max) | 5 / 25 |
| | | The minimum and the maximum number of virtual cores available to the pool. These override weight settings. (optional) |
| | Memory (Min / Max) | 100 / 10000 MB |
| | | The minimum and the maximum amount of aggregate memory available to the pool. These override weight settings. (optional) |
| | Max Running Apps | 5 |
| | | A limit on the number of applications simultaneously running in a pool. |

Cloudera Manager Dynamic Resource Management: Example

Edit Resource Pool: hrpool

General YARN Submission Access Control Administration Access Control

This feature is relevant only if **Enable ResourceManager ACLs** is set to true and **Admin ACL** is NOT set to * (See Other)

Fair Scheduler Access Control Lists control who can submit applications to pools. For subpools, users who have permissions automatically inherit the same ability for the child.

Allow anyone to submit to this pool
 Allow these users and groups to submit to this pool

Users Comma separated list of users. Space characters are not allowed.
Inherited from parent pools: oracle,root

Groups hr

Edit Resource Pool: hrpool

General YARN Submission Access Control Administration Access Control

This feature is relevant only if **Enable ResourceManager ACLs** is set to true and **Admin ACL** is NOT set to * (See Other)

Fair Scheduler Access Control Lists control who can administer pools. For subpools, users who have permissions automatically inherit the same ability for the child.

Allow anyone to administer this pool
 Allow these users and groups to administer this pool

Users Comma separated list of users. Space characters are not allowed.
Inherited from parent pools: oracle,root

Groups hr

Submitting a Job to hrpool by the lucy User from the hr Group

```
[oracle@scaj51bda12 jars]$ id lucy  
uid=478(lucy) gid=1005(hr) groups=1005(hr)  
[oracle@scaj51bda12 jars]$ kinit lucy  
Password for lucy@DEV.ORACLE.COM:
```

lucy belongs to the hr group; therefore, she can submit the job to hrpool.

```
$ hadoop jar hadoop-mapreduce-examples-2.5.0-cdh5.3.0.jar teragen -  
Dmapred.map.tasks=5 -D mapreducqueuename=hrpool 1000000  
/user/hr/d2
```



```
15/02/26 10:08:59 INFO mapreduce.Job: map 0% reduce 0%  
15/02/26 10:09:09 INFO mapreduce.Job: map 40% reduce 0%  
15/02/26 10:10:06 INFO mapreduce.Job: map 60% reduce 0%  
15/02/26 10:10:34 INFO mapreduce.Job: map 100% reduce 0%  
15/02/26 10:10:34 INFO mapreduce.Job: Job job_1424929332007_0001 completed successfully  
15/02/26 10:10:34 INFO mapreduce.Job: Counters: 31  
File System Counters  
FILE: Number of bytes read=0  
FILE: Number of bytes written=576800  
FILE: Number of read operations=0
```

Monitoring the Status of the Submitted MapReduce Job

output directory

```
[oracle@scaj51bda12 jars]$ hadoop fs -ls /user/hr/d2
Found 6 items
-rw-r--r-- 3 lucy hive 0 2015-02-26 10:10 /user/hr/d2/_SUCCESS
-rw-r--r-- 3 lucy hive 20000000 2015-02-26 10:10 /user/hr/d2/part-m-00000
-rw-r--r-- 3 lucy hive 20000000 2015-02-26 10:10 /user/hr/d2/part-m-00001
-rw-r--r-- 3 lucy hive 20000000 2015-02-26 10:09 /user/hr/d2/part-m-00002
-rw-r--r-- 3 lucy hive 20000000 2015-02-26 10:09 /user/hr/d2/part-m-00003
-rw-r--r-- 3 lucy hive 20000000 2015-02-26 10:10 /user/hr/d2/part-m-00004
[oracle@scaj51bda12 jars]$
```

All Applications

Cluster Metrics

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Memory Used | Memory Total | Memory Reserved | Vcores Used | Vcores Total | Vcores Reserved | Active Nodes | Decommissioned Nodes | Lost Nodes | Unhealthy Nodes | Rebooted Nodes |
|----------------|--------------|--------------|----------------|--------------------|-------------|--------------|-----------------|-------------|--------------|-----------------|--------------|----------------------|------------|-----------------|----------------|
| 1 | 0 | 0 | 1 | 0 | 0 B | 159 GB | 0 B | 0 | 248 | 0 | 4 | 0 | 0 | 0 | 0 |

User Metrics for dr.who

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Containers Pending | Containers Reserved | Memory Used | Memory Pending | Memory Reserved | Vcores Used | Vcores Pending | Vcores Reserved |
|----------------|--------------|--------------|----------------|--------------------|--------------------|---------------------|-------------|----------------|-----------------|-------------|----------------|-----------------|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 B | 0 B | 0 B | 0 | 0 | 0 |

Show 20 entries Search:

| ID | User | Name | Application Type | Queue | StartTime | FinishTime | State | FinalStatus | Progress | Tracking UI |
|--------------------------------|------|---------|------------------|-------------|--------------------------------|--------------------------------|----------|-------------|----------|-------------------------|
| application_1424929332007_0001 | lucy | TeraGen | MAPREDUCE | root.hrpool | Thu Feb 26 13:07:21 -0500 2015 | Thu Feb 26 13:10:32 -0500 2015 | FINISHED | SUCCEEDED | | History |

The job was submitted to the
root.hrpool.

Examining marketingpool

| Name | Weight % | Virtual Cores Min / Max | Memory Min / Max | Max Running Apps | Scheduling Policy | |
|---------------|-----------|----------------------------|---------------------|------------------|-------------------|--|
| root | 1 100.0% | - / - | - / - | - | DRF | <input checked="" type="checkbox"/> Edit ▾ |
| default | 1 0.9% | - / - | - / - | - | DRF | <input checked="" type="checkbox"/> Edit ▾ |
| hrpool | 10 9.0% | 5 / 25 | 100MB / 10000MB | 5 | DRF | <input checked="" type="checkbox"/> Edit ▾ |
| marketingpool | 100 90.1% | 25 / 50 | 500MB / 5000MB | 10 | DRF | <input checked="" type="checkbox"/> Edit ▾ |

Edit Resource Pool: marketingpool

General YARN Submission Access Control Administration Access Control

This feature is relevant only if **Enable ResourceManager ACLs** is set to true and **Admin ACL** is NOT set to * (See Other Settings).

Fair Scheduler Access Control Lists control who can submit applications to pools. For subpools, users who have permission to submit a parent pool automatically inherit the same ability for the child.

- Allow anyone to submit to this pool
 Allow these users and groups to submit to this pool

Users

Comma separated list of users. Space characters are not allowed.

Inherited from parent pools: oracle,root

Groups

marketing

Submitting a Job to marketingpool by the lucy User from the hr Group

```
[oracle@scaj51bda12 jars]$ id lucy  
uid=478(lucy) gid=1005(hr) groups=1005(hr)  
[oracle@scaj51bda12 jars]$ kinit lucy  
Password for lucy@DEV.ORACLE.COM:
```

lucy belongs to the hr group; therefore, she CANNOT submit the job to marketingpool.

```
$ hadoop fs -rm -r /user/hr/d2  
$ hadoop jar hadoop-mapreduce-examples-2.5.0-cdh5.3.0.jar teragen  
-D mapred.map.tasks=5 -D mapreduce.job.queuename=marketingpool  
1000000 /user/hr/d2
```

```
15/02/26 10:43:21 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1424929332007_0003  
15/02/26 10:43:21 WARN token.Token: Cannot find class for token kind kms-dt  
15/02/26 10:43:21 WARN token.Token: Cannot find class for token kind kms-dt  
Kind: kms-dt, Service: 192.168.42.121:16000, Ident: 00 04 6c 75 63 79 04 79 61 72 6e 00 8a 01 4b c7 33 46 ab 8a 01 4b  
eb 3f ca ab 3c 1d  
15/02/26 10:43:21 INFO mapreduce.JobSubmitter: Kind: HDFS_DELEGATION_TOKEN, Service: ha-hdfs:scaj51cdh-ns, Ident: (HD  
FS_DELEGATION_TOKEN token 202 for lucy)  
15/02/26 10:43:22 INFO impl.YarnClientImpl: Submitted application application_1424929332007_0003  
15/02/26 10:43:22 INFO mapreduce.JobSubmitter: Cleaning up the staging area /user/lucy/.staging/job_1424929332007_000  
15/02/26 10:43:22 WARN security.UserGroupInformation: PrivilegedActionException as:lucy@DEV.ORACLE.COM (auth:KERBERO  
S) cause:java.io.IOException: Failed to run job : User lucy cannot submit applications to queue root.marketingpool  
java.io.IOException: Failed to run job : User lucy cannot submit applications to queue root.marketingpool  
at org.apache.hadoop.mapreduce.YARNRunner.submitJob(YARNRunner.java:200)
```



Monitoring the Status of the Submitted MapReduce Job

Logged in as: dr.who

All Applications

Cluster Metrics

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Memory Used | Memory Total | Memory Reserved | Vcores Used | Vcores Total | Vcores Reserved | Active Nodes | Decommissioned Nodes | Lost Nodes | Unhealthy Nodes | Rebooted Nodes |
|----------------|--------------|--------------|----------------|--------------------|-------------|--------------|-----------------|-------------|--------------|-----------------|--------------|----------------------|------------|-----------------|----------------|
| 1 | 0 | 0 | 1 | 0 | 0 B | 159 GB | 0 B | 0 | 248 | 0 | 4 | 0 | 0 | 0 | 0 |

User Metrics for dr.who

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Containers Pending | Containers Reserved | Memory Used | Memory Pending | Memory Reserved | Vcores Used | Vcores Pending | Vcores Reserved |
|----------------|--------------|--------------|----------------|--------------------|--------------------|---------------------|-------------|----------------|-----------------|-------------|----------------|-----------------|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 B | 0 B | 0 B | 0 | 0 | 0 |

Show 20 entries Search:

| ID | User | Name | Application Type | Queue | StartTime | FinishTime | State | FinalStatus | Progress | Tracking UI |
|--------------------------------|------|---------|------------------|--------------------|--------------------------------------|--------------------------------------|----------|-------------|----------|-------------|
| application_1424929332007_0003 | lucy | TeraGen | MAPREDUCE | root.marketingpool | Thu Feb 26 13:43:22 -0500 2015 | Thu Feb 26 13:43:22 -0500 2015 | FAILED | FAILED | | UNASSIGNED |
| application_1424929332007_0002 | lucy | TeraGen | MAPREDUCE | nrmarketingpool | Thu Feb 26 13:41:48 -0500 2015 | Thu Feb 26 13:41:48 -0500 2015 | FAILED | FAILED | | UNASSIGNED |
| application_1424929332007_0001 | lucy | TeraGen | MAPREDUCE | root.hrpool | Thu Feb 26 13:07:21 -0500 2015 | Thu Feb 26 13:10:32 -0500 2015 | FINISHED | SUCCEEDED | History | |



Submitting a Job to marketingpool by the bob User from the marketing Group

```
[oracle@scaj51bda12 jars]$ id bob  
uid=477(bob) gid=1007(marketing) groups=1007(marketing)  
[oracle@scaj51bda12 jars]$ kinit bob  
Password for bob@DEV.ORACLE.COM:  
[oracle@scaj51bda12 jars]$ █
```

bob belongs to the **marketing** group; therefore, he can submit the job to **marketingpool**.

```
$ hadoop fs -rm -r /user/marketing/d1  
$ hadoop jar hadoop-mapreduce-examples-2.5.0-cdh5.3.0.jar teragen  
-Dmapred.map.tasks=5 -D mapreduce.job.queuename=marketingpool  
1000000 /user/marketing/d1
```

```
15/02/26 16:07:57 INFO mapreduce.Job: The url to track the job: http://scaj51bda12.us.oracle.com:8088/proxy/application_1424929332007_0004/  
15/02/26 16:07:57 INFO mapreduce.Job: Running job: job_1424929332007_0004  
15/02/26 16:08:09 INFO mapreduce.Job: Job job_1424929332007_0004 running in uber mode : false  
15/02/26 16:08:09 INFO mapreduce.Job: map 0% reduce 0%  
15/02/26 16:08:19 INFO mapreduce.Job: map 80% reduce 0%  
15/02/26 16:08:23 INFO mapreduce.Job: map 100% reduce 0%  
15/02/26 16:08:23 INFO mapreduce.Job: Job job_1424929332007_0004 completed successfully
```



Monitoring the Status of the Submitted MapReduce Job

output directory

```
[oracle@scaj51bda12 jars]$ hadoop fs -ls /user/marketing/d1
Found 6 items
-rw-r--r-- 3 bob marketing          0 2015-02-26 16:08 /user/marketing/d1/_SUCCESS
-rw-r--r-- 3 bob marketing 20000000 2015-02-26 16:08 /user/marketing/d1/part-m-00000
-rw-r--r-- 3 bob marketing 20000000 2015-02-26 16:08 /user/marketing/d1/part-m-00001
-rw-r--r-- 3 bob marketing 20000000 2015-02-26 16:08 /user/marketing/d1/part-m-00002
-rw-r--r-- 3 bob marketing 20000000 2015-02-26 16:08 /user/marketing/d1/part-m-00003
-rw-r--r-- 3 bob marketing 20000000 2015-02-26 16:08 /user/marketing/d1/part-m-00004
[oracle@scaj51bda12 jars]$
```

All Applications

Cluster Metrics

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Memory Used | Memory Total | Memory Reserved | VCores Used | VCores Total | VCores Reserved | Active Nodes | Decommissioned Nodes | Lost Nodes | Unhealthy Nodes | Rebooted Nodes |
|----------------|--------------|--------------|----------------|--------------------|-------------|--------------|-----------------|-------------|--------------|-----------------|--------------|----------------------|------------|-----------------|----------------|
| 2 | 0 | 0 | 2 | 0 | 0 B | 159 GB | 0 B | 0 | 248 | 0 | 4 | 0 | 0 | 0 | 0 |

User Metrics for dr.who

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Containers Pending | Containers Reserved | Memory Used | Memory Pending | Memory Reserved | VCores Used | VCores Pending | VCores Reserved |
|----------------|--------------|--------------|----------------|--------------------|--------------------|---------------------|-------------|----------------|-----------------|-------------|----------------|-----------------|
| 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 B | 0 B | 0 B | 0 | 0 | 0 |

Show 20 entries Search:

| ID | User | Name | Application | Queue | StartTime | FinishTime | State | FinalStatus | Progress | Tracking UI |
|--------------------------------|------|---------|-------------|--------------------|--------------------------------|--------------------------------|----------|-------------|----------|-------------|
| application_1424929332007_0004 | bob | TeraGen | MAPREDUCE | root.marketingpool | Thu Feb 26 19:07:56 -0500 2015 | Thu Feb 26 19:08:22 -0500 2015 | FINISHED | SUCCEEDED | | History |
| application_1424929332007_0003 | lucy | TeraGen | MAPREDUCE | root.marketingpool | Thu Feb 26 | Thu Feb 26 13:43:22 | FAILED | FAILED | | UNASSIGNED |

Delay Scheduling

- YARN schedulers attempt to honor locality container requests.
- If the requested node is not available, delaying the granting of an alternative resource for a few seconds can increase the chances of getting a container on the requested node.
- Delay scheduling is supported by both Capacity and Fair Schedulers.

Agenda

- Job Scheduling in YARN
- YARN application command

YARN application Command

- You can use the `yarn application` command to list and kill applications:

```
$ yarn application <options>
```

| <options> | Description |
|------------------------------------|--|
| <code>-list</code> | Lists applications from the Resource Manager |
| <code>-appStates</code> | Works with <code>-list</code> to filter applications based on input comma-separated list of application states. The options are: ALL, NEW, NEW_SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED. |
| <code>-status ApplicationID</code> | Prints the status of the application |
| <code>-kill ApplicationID</code> | Kills the application |

YARN application Command: Example

```
$ yarn application -list
```

```
[oracle@bigdatalite ~]$ yarn application -list
15/02/12 09:00:00 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
Total number of applications (application-types: [] and states: [SUBMITTED, ACCEPTED, RUNNING]):0
      Application-Id      Application-Name      Application-Type
      User        Queue        State        Final-State
      Progress      Tracking-URL
[oracle@bigdatalite ~]$
```

1

2. Insert into staging table Inserts filtered and transformed log data into staging table

```
1 INSERT OVERWRITE TABLE moviework.movieapp_log_stage
2 SELECT * FROM (
3   SELECT custid,
4     CASE WHEN m1.genreid = 0 THEN m1.genreid ELSE 1 END genreid,
5     m1.time,
6     CAST((CASE m1.recommended WHEN 'Y' THEN 1 ELSE 0 END) AS INT) recommended,
7     ...
8
9 Execute Save Save as... Explain or create a New query
```

2

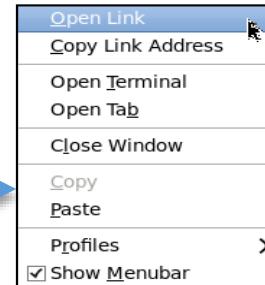
```
Progress          TRACKING URL
[oracle@bigdatalite ~]$ yarn application -list
15/02/12 09:57:57 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
Total number of applications (application-types: [] and states: [SUBMITTED, ACCEPTED, RUNNING]):1
      Application-Id      Application-Name      Application-Type
      Queue        State        Final-State        Progress
      Tracking-URL
application_1423174990155_0022  INSERT OVERWRITE TABLE moview...union_result(Stage-1)
MAPREDUCE          oracle        root.oracle        RUNNING
DEFINED           5% http://bigdatalite.localdomain:54444
```

3

YARN application Command: Example

```
$ yarn application -status <job_id>
```

```
[oracle@bigdatalite ~]$ yarn application -status application_1423174990155_0022
15/02/12 10:14:39 INFO client.RMProxy: Connecting to ResourceManager at localhost/127
:8032
Application Report :
  Application-Id : application_1423174990155_0022
  Application-Name : INSERT OVERWRITE TABLE moview...union_result(Stage-1)
  Application-Type : MAPREDUCE
  User : oracle
  Queue : root.oracle
  Start-Time : 1423753071390
  Finish-Time : 1423753095903
  Progress : 100%
  State : FINISHED
  Final-State : SUCCEEDED
  Tracking-URL : http://bigdatalite.localdomain:19888/jobhistory/job/job_142317
5_0022
  RPC Port : 14374
  AM Host : bigdatalite.localdomain
  Diagnostics :
[oracle@bigdatalite ~]$
```



The screenshot shows the Hadoop MapReduce Job interface. At the top, there's a logo and the text "MapReduce Job job_1423174990155_0022". On the left, a sidebar has "Application" expanded, showing "Job" with "Overview", "Counters", "Configuration", "Map tasks", and "Reduce tasks". Below that is a "Tools" section. The main area displays job details:

| | |
|-------------------|---|
| Job Name: | INSERT OVERWRITE TABLE moview...union_result(Stage-1) |
| User Name: | oracle |
| Queue: | root.oracle |
| State: | SUCCEEDED |
| Uberized: | false |
| Submitted: | Thu Feb 12 09:57:51 EST 2015 |
| Started: | Thu Feb 12 09:57:57 EST 2015 |
| Finished: | Thu Feb 12 09:58:15 EST 2015 |

Monitoring an Application by Using the ResourceManager Web UI

The screenshot illustrates the monitoring of an application using the Hadoop ResourceManager Web UI. It consists of two main parts:

- Top Panel:** Shows the "YARN Applications" section of the "Hadoop" tab. A red box highlights the "YARN Applications" link. The page displays a "Reduce Job" with ID **23174990155_0022**.
- Bottom Panel:** Shows the "All Applications" page. A blue arrow points from the "Queue or 'pool'" label to the "name" of the application listed in the table.

Cluster Metrics:

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Memory Used | Memory Total | Memory Reserved | Active Nodes | Decommissioned Nodes | Lost Nodes |
|----------------|--------------|--------------|----------------|--------------------|-------------|--------------|-----------------|--------------|----------------------|------------|
| 25 | 0 | 0 | 25 | 0 | 0 B | 4 GB | 0 B | 1 | 0 | 0 |

User Metrics for dr.who:

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Containers Pending | Containers Reserved | Memory Used | Mem Pending |
|----------------|--------------|--------------|----------------|--------------------|--------------------|---------------------|-------------|-------------|
| 0 | 0 | 0 | 25 | 0 | 0 | 0 | 0 B | 0 B |

All Applications:

| ID | User | Name | Application Type | Queue | StartTime | FinishTime | State | FinalStatus |
|--------------------------------|--------|--|------------------|-------------|-------------------------------|-------------------------------|----------|-------------|
| application_1423174990155_0025 | oracle | INSERT OVERWRITE TABLE moview...union_result(Stage-4) | MAPREDUCE | root.oracle | Thu, 12 Feb 2015 14:59:01 GMT | Thu, 12 Feb 2015 14:59:14 GMT | FINISHED | SUCCESS |
| application_1423174990155_0024 | oracle | INSERT OVERWRITE TABLE moview...union_result(Stage-3) | MAPREDUCE | root.oracle | Thu, 12 Feb 2015 14:58:39 | Thu, 12 Feb 2015 14:58:59 | FINISHED | SUCCESS |

Scheduler: BDA Example

The screenshot shows the Apache Ambari UI interface for managing a cluster. On the left, there's a sidebar with navigation links: Cluster, About Nodes, Applications (with sub-options NEW, NEW_SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED), Scheduler, and Tools.

Cluster Metrics

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Memory Used | Memory Total | Memory Reserved | Active Nodes | Decommissioned Nodes | Lost Nodes | Unhealt Nodes |
|----------------|--------------|--------------|----------------|--------------------|-------------|--------------|-----------------|--------------|----------------------|------------|---------------|
| 25 | 0 | 0 | 25 | 0 | 0 B | 4 GB | 0 B | 1 | 0 | 0 | 0 |

User Metrics for dr.who

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Containers Pending | Containers Reserved | Memory Used | Memory Pending |
|----------------|--------------|--------------|----------------|--------------------|--------------------|---------------------|-------------|----------------|
| 0 | 0 | 0 | 25 | 0 | 0 | 0 | 0 B | 0 B |

Application Queues

Legend: Fair Share (grey), Used (green), Used (over fair share) (orange), Max Capacity (light grey)

| Queue | Fair Share | Used | Used (over fair share) | Max Capacity |
|----------------|------------|-----------|------------------------|--------------|
| - root | 0.0% used | 0.0% used | 0.0% used | 0.0% used |
| + root.default | 0.0% used | 0.0% used | 0.0% used | 0.0% used |
| + root.oracle | 0.0% used | 0.0% used | 0.0% used | 0.0% used |
| + root.pool1 | 0.0% used | 0.0% used | 0.0% used | 0.0% used |

Show 20 entries Search:

| ID | User | Name | Application Type | Queue | Fair Share | StartTime | FinishTime | State | FinalStatus | Progress |
|----|------|------|------------------|-------|------------|-----------|------------|-------|-------------|----------|
|----|------|------|------------------|-------|------------|-----------|------------|-------|-------------|----------|

Summary

- In this lesson, you should have learned how to:
 - Use YARN to manage resources efficiently
 - Use the YARN application command

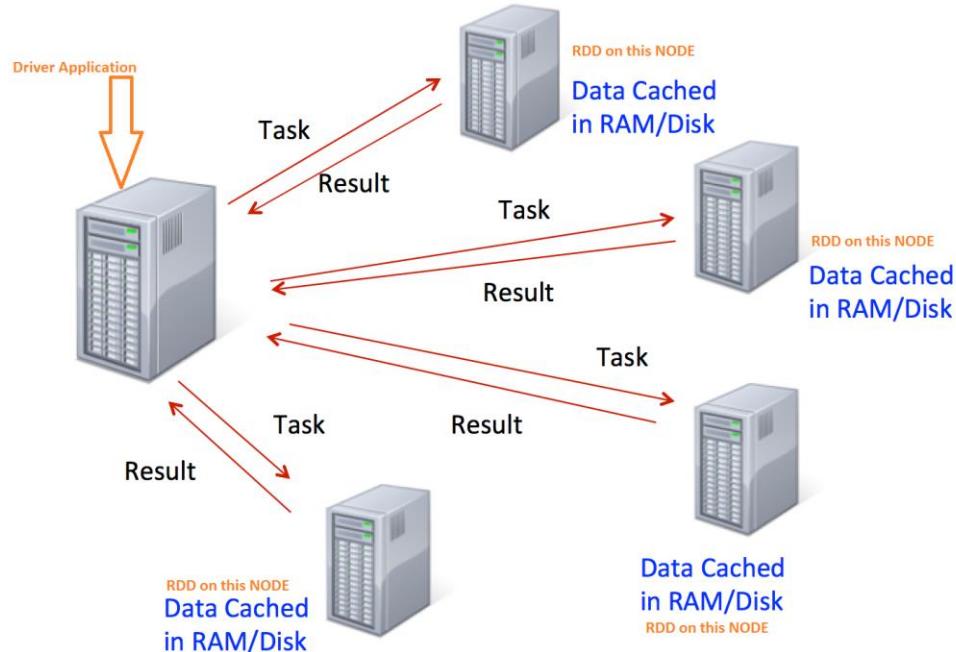
Lesson 1: Introduction to Apache Spark



What you will learn

How does Spark execute a job

1. Apache Spark Features
2. Spark Components
3. Use Cases



What is Spark?

- It is a cluster platform on top of a storage layer
- It is an extension of MapReduce that also supports: Streaming and AI
- Most importantly, it runs in MEMORY



Why Use Apache Spark?

- Performance
 - Significantly faster on DISK
 - 100X+ performance increase in MEMORY
- Development
- Deployment
- Cohesive Stack
- Language Independence



Why Use Apache Spark?

- Performance
- Development
 - Easily and quickly develop apps
 - Compressed code (much smaller than Hadoop Code)
 - REPL
- Deployment
- Cohesive Stack
- Language Independence



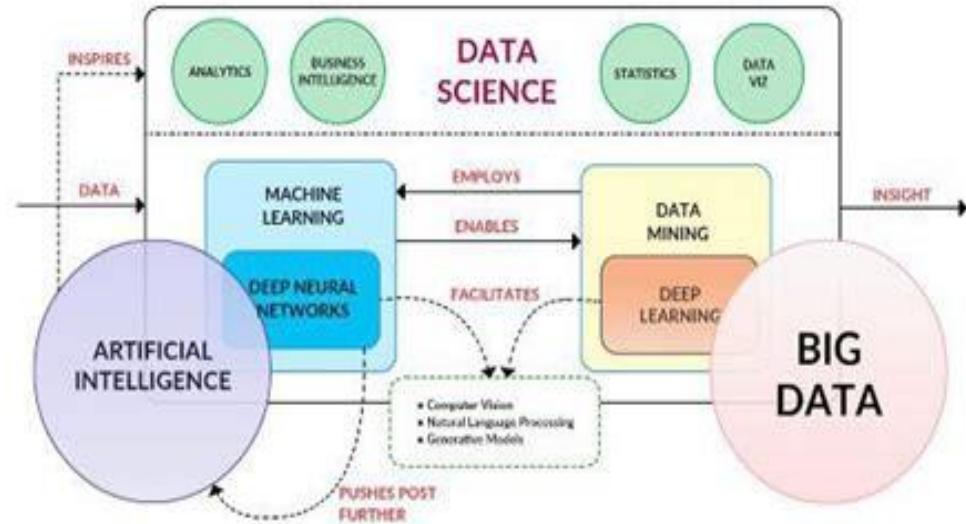
Why Use Apache Spark?

- Performance
- Development
- Deployment
 - Standalone
 - YARN
 - MESOS
- Cohesive Stack
- Language Independence



Why Use Apache Spark?

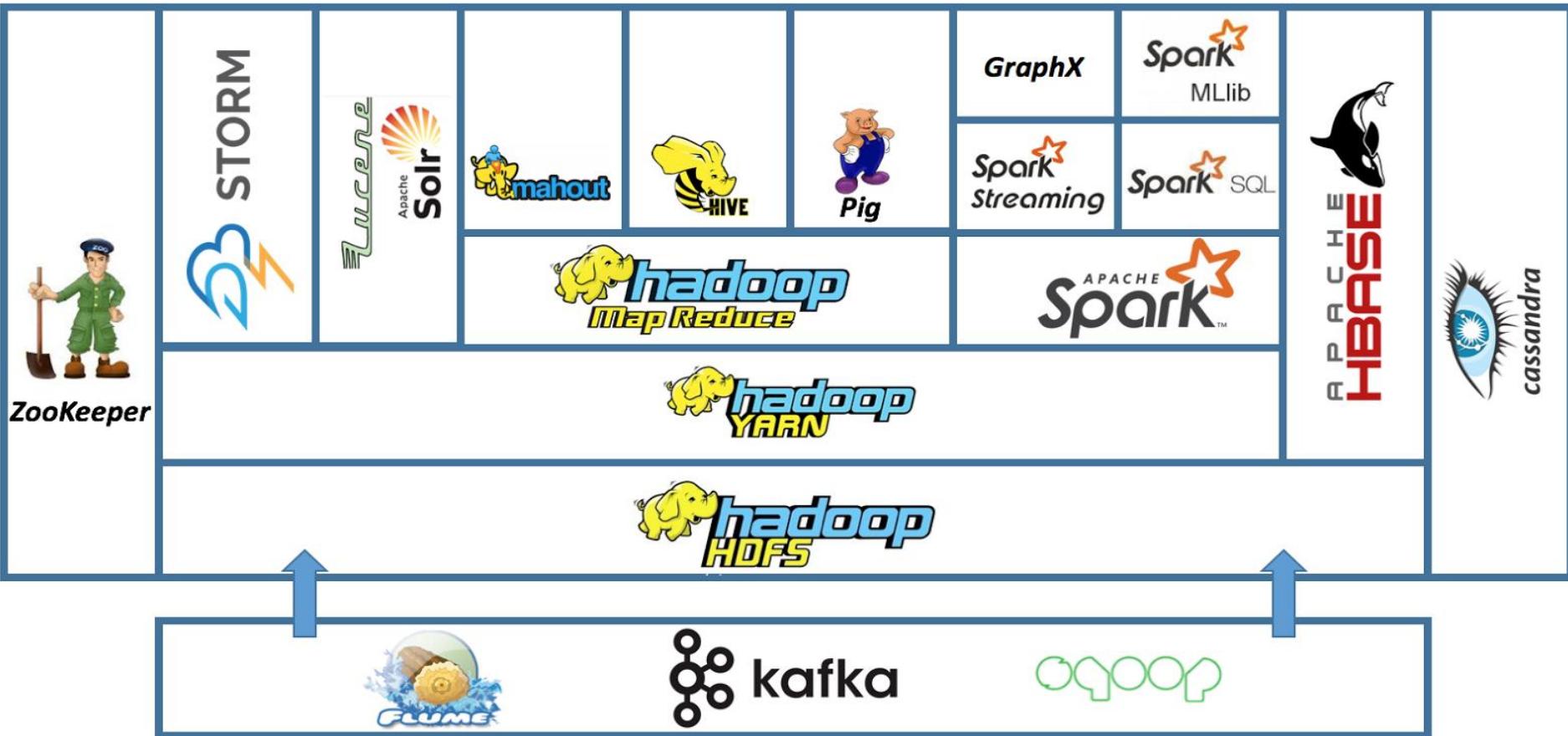
- Performance
- Development
- Deployment
- Cohesive Stack
 - Multiple architectural options
 - Batching
 - AI
 - Stream Use Cases
- Language Independence



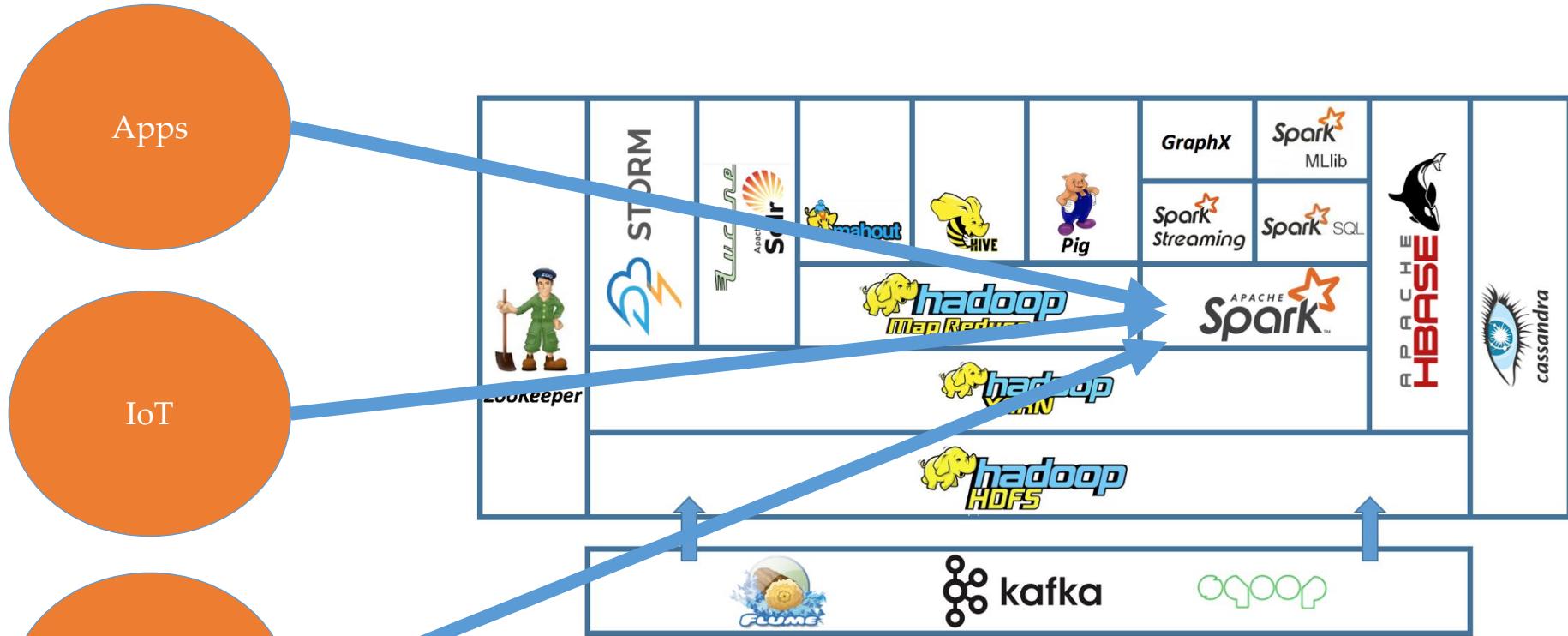
Why Use Apache Spark?

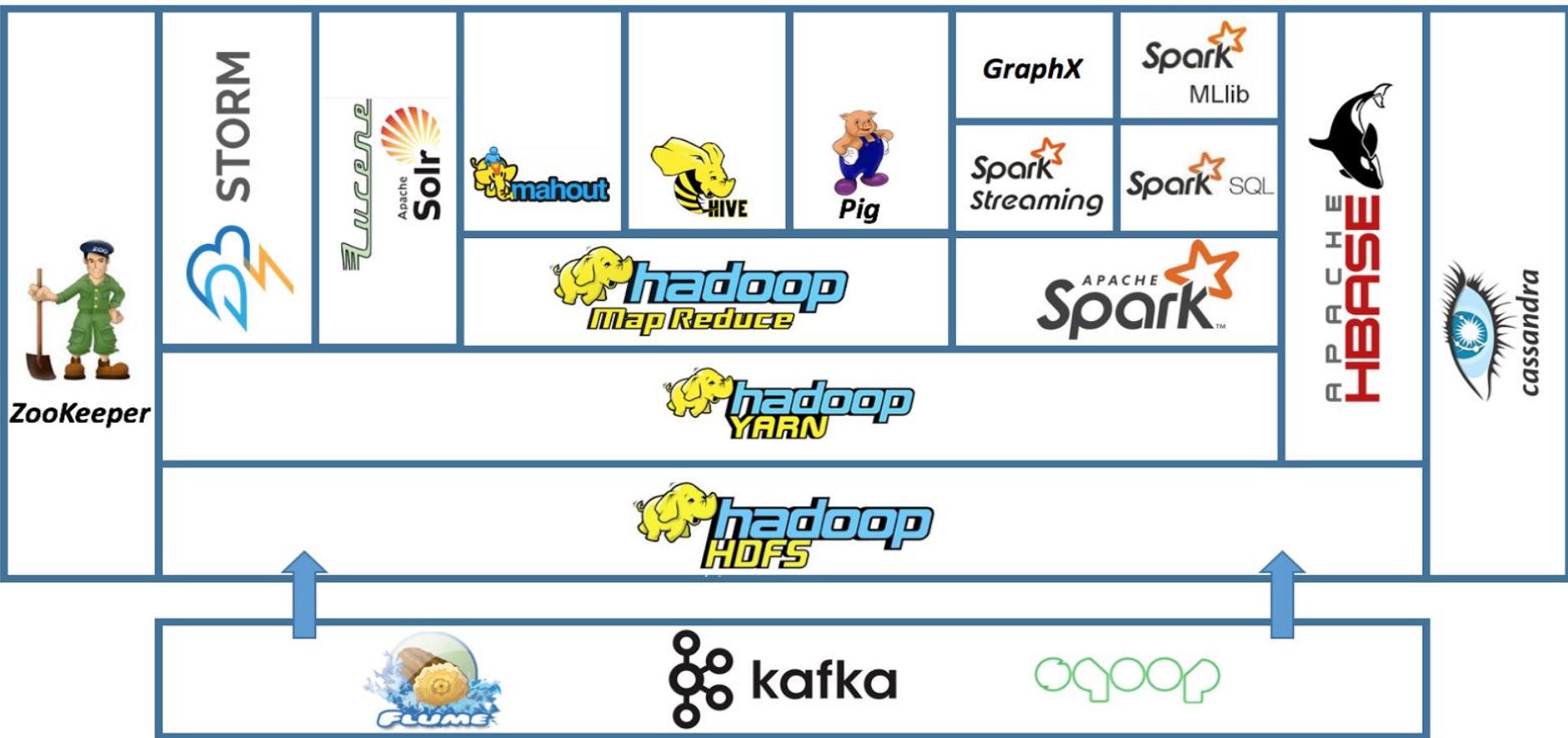
- Performance
- Development
- Deployment
- Cohesive Stack
- Language Independence
 - Java
 - Scala
 - Python (PySpark)



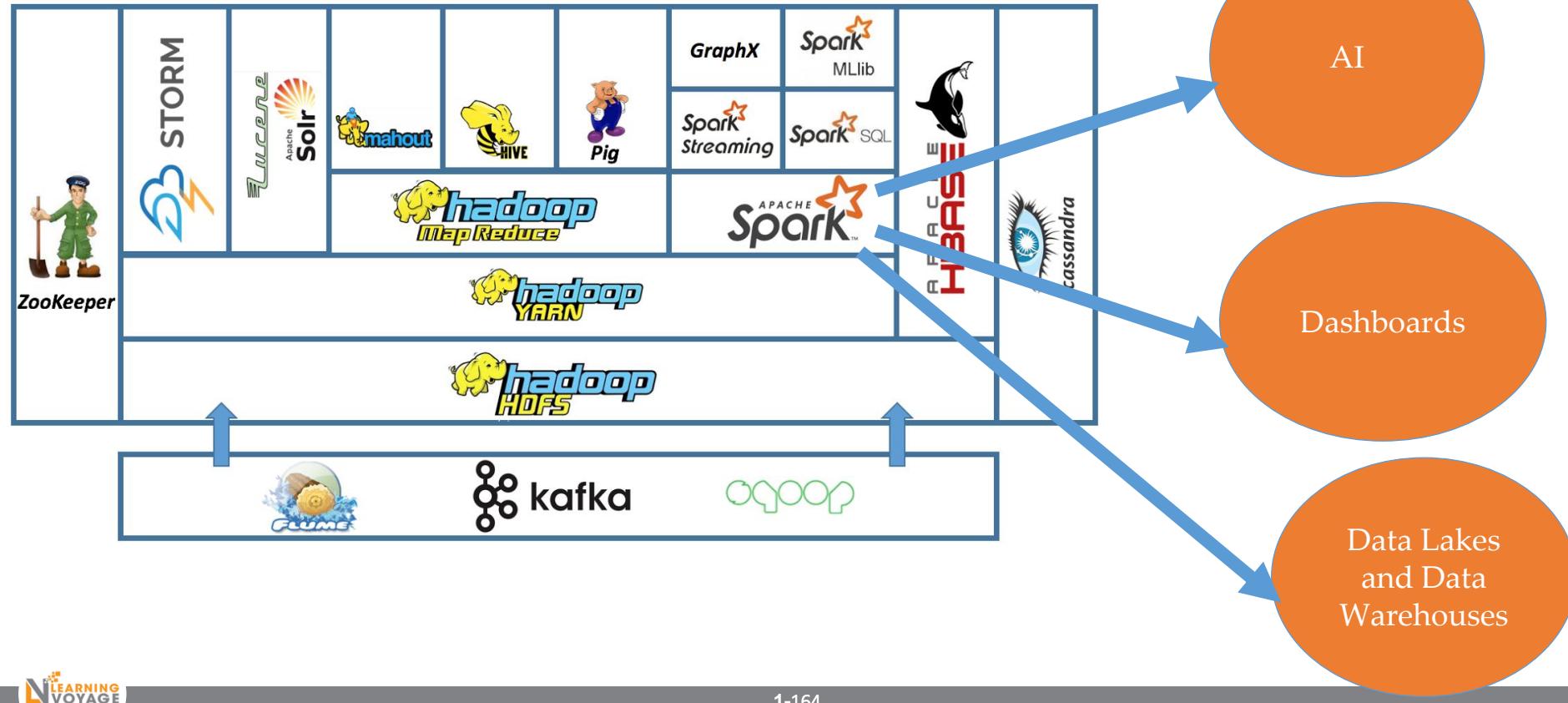


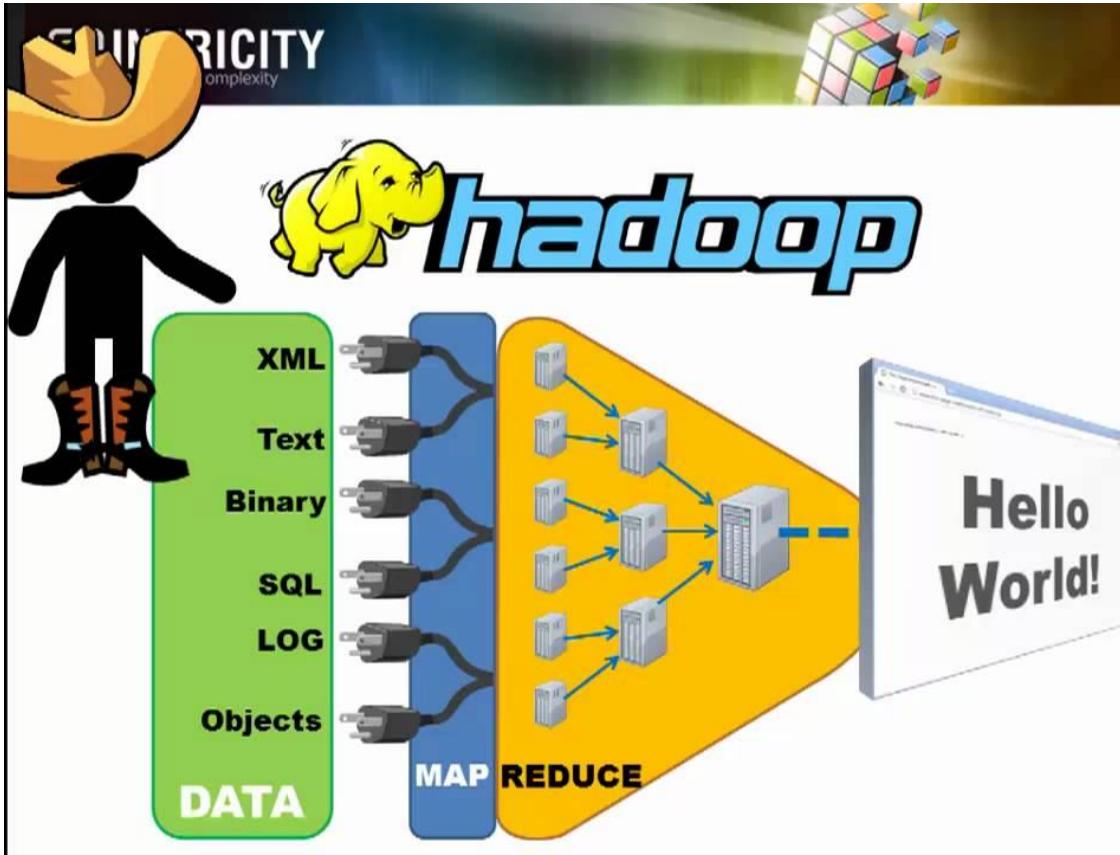
INPUTS





OUTPUTS

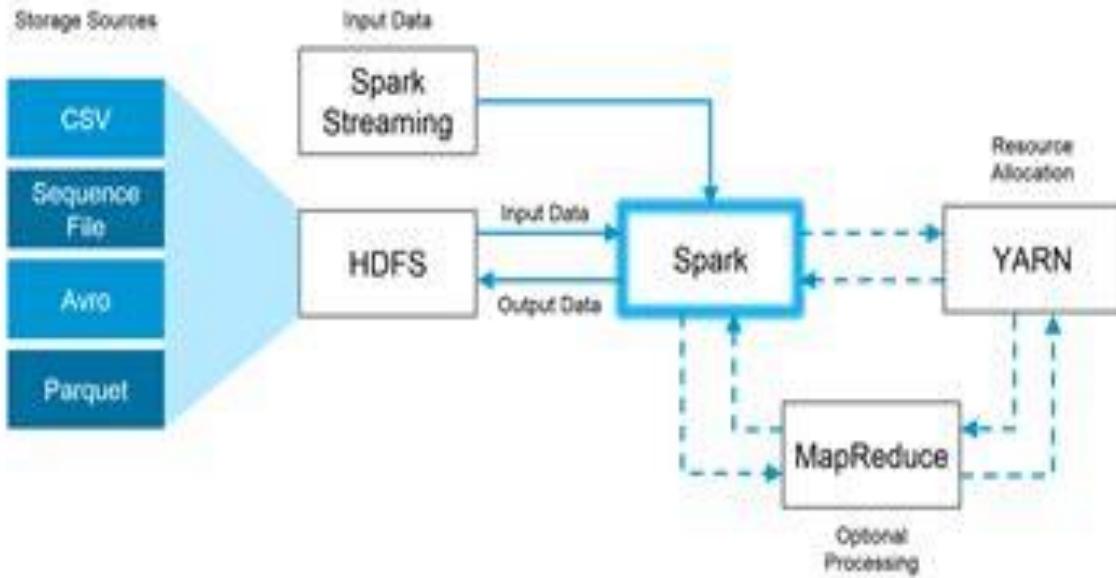




Spark

- Parallel in-memory processing across platform
- Multiple data sources, applications, users





Spark vs 🐘 Hadoop MapReduce

Factors

Speed

Written In

Data Processing

Ease of Use

Caching

Spark

100x times than MapReduce

Scala

Batch / real-time / iterative /
interactive /graph

Compact & easier than Hadoop

Caches the data in-memory &
enhances the system performance

Hadoop MapReduce

Faster than traditional system

Java

Batch processing

Complex & lengthy

Doesn't support caching of data

≡ LAB_2_1.sc X | ≡ LAB_2_1.py X | ≡ LAB_2_2.sc X | ≡ LAB_2_2.py X | root@wk-c X | root@wk-c X

```
root@wk-caas-d2813d52c4c44e5cb42c5907b9404989-f85d3c16adde1b40c633d4:~/work/spark-dev3600/DEV360
ON_02# # This is data from the Baltimore Police Department[]
```



Spark vs. MapReduce Example: Word Count

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

    public static class ReduceClass extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text word, Iterable<IntWritable> values, Context con) throws IOException, InterruptedException {
            int sum = 0;
            for(IntWritable value : values) {
                sum += value.get();
            }
            con.write(word, new IntWritable(sum));
        }
    }

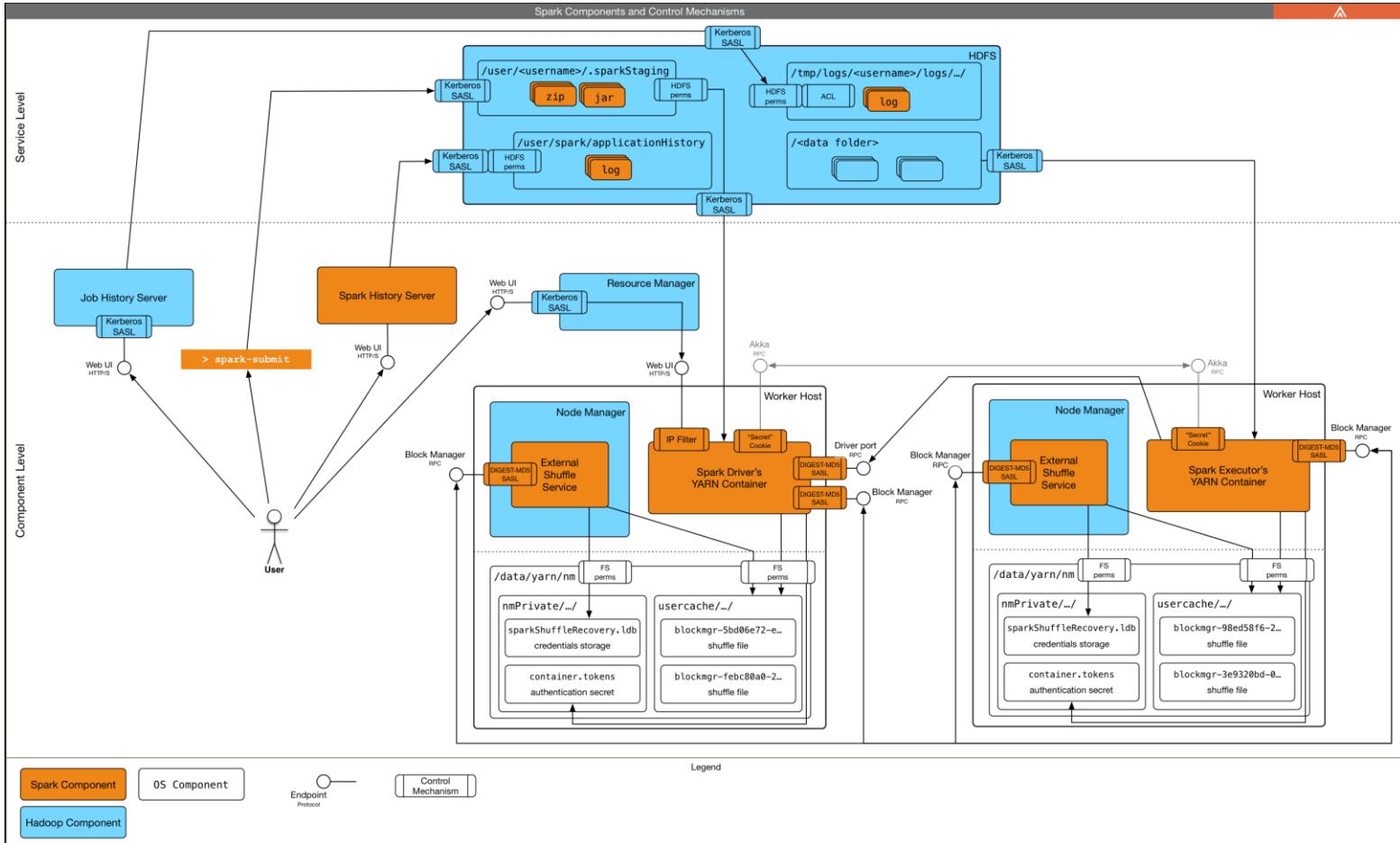
    public static void main(String [] args) throws Exception {
        Configuration conf=new Configuration();
        String[] files=new GenericOptionsParser(conf,args).getRemainingArgs();
        Path input=new Path(files[0]);
        Path output=new Path(files[1]);
        Job job=new Job(conf,"wordcount");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(MapClass.class);
        job.setReducerClass(ReduceClass.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, input);
        FileOutputFormat.setOutputPath(job, output);
        System.exit(job.waitForCompletion(true)?0:1);
    }
}
```

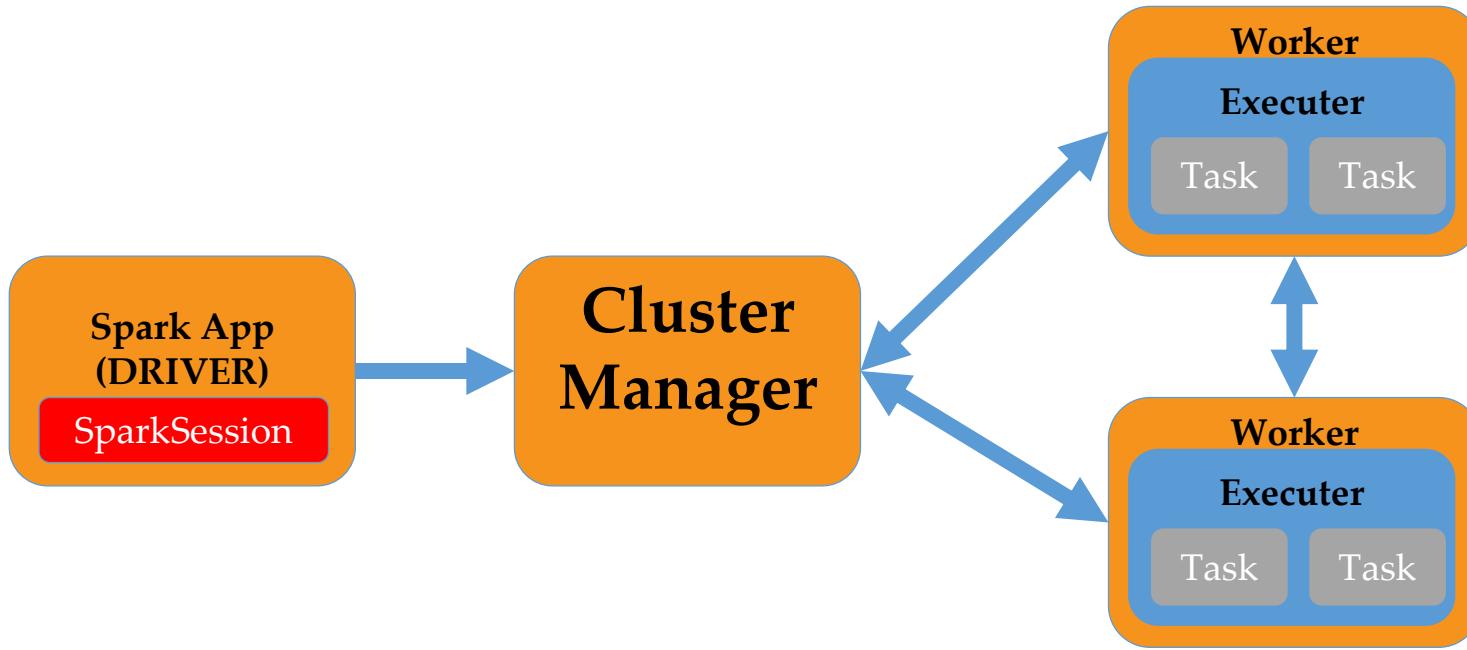
Knowledge Check

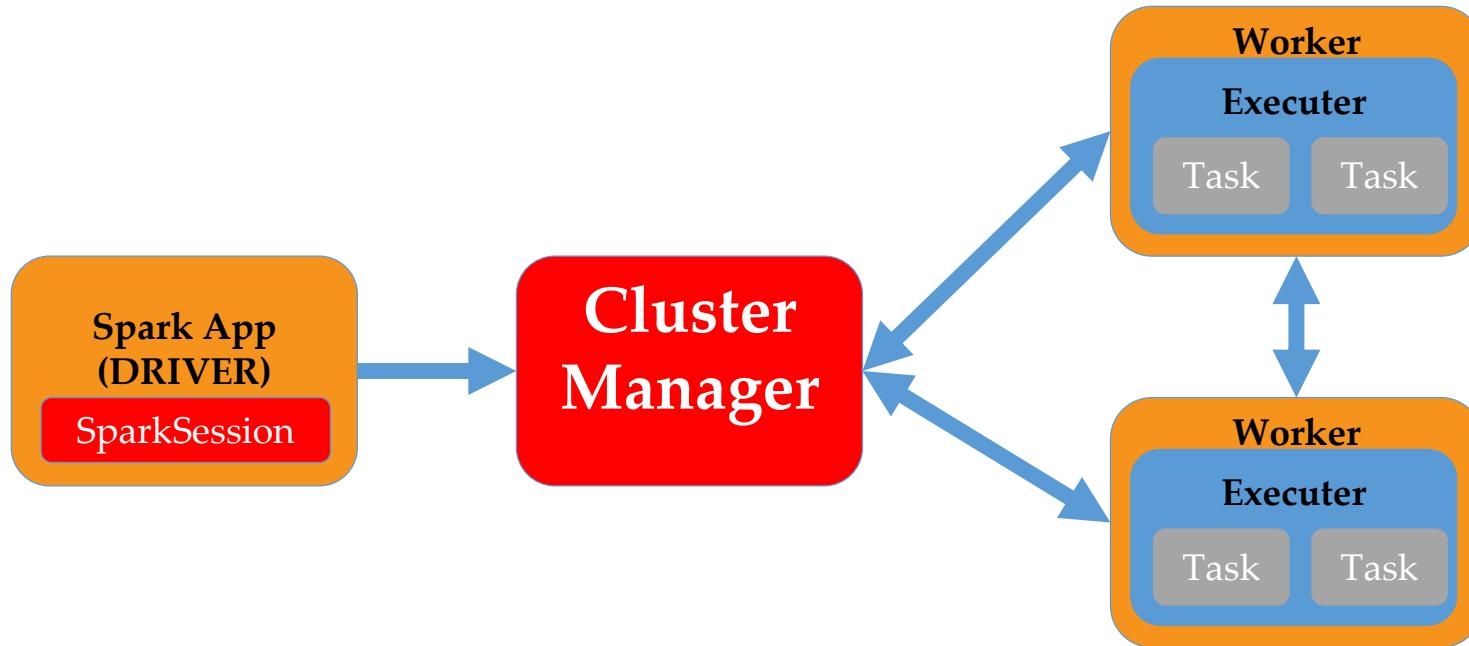


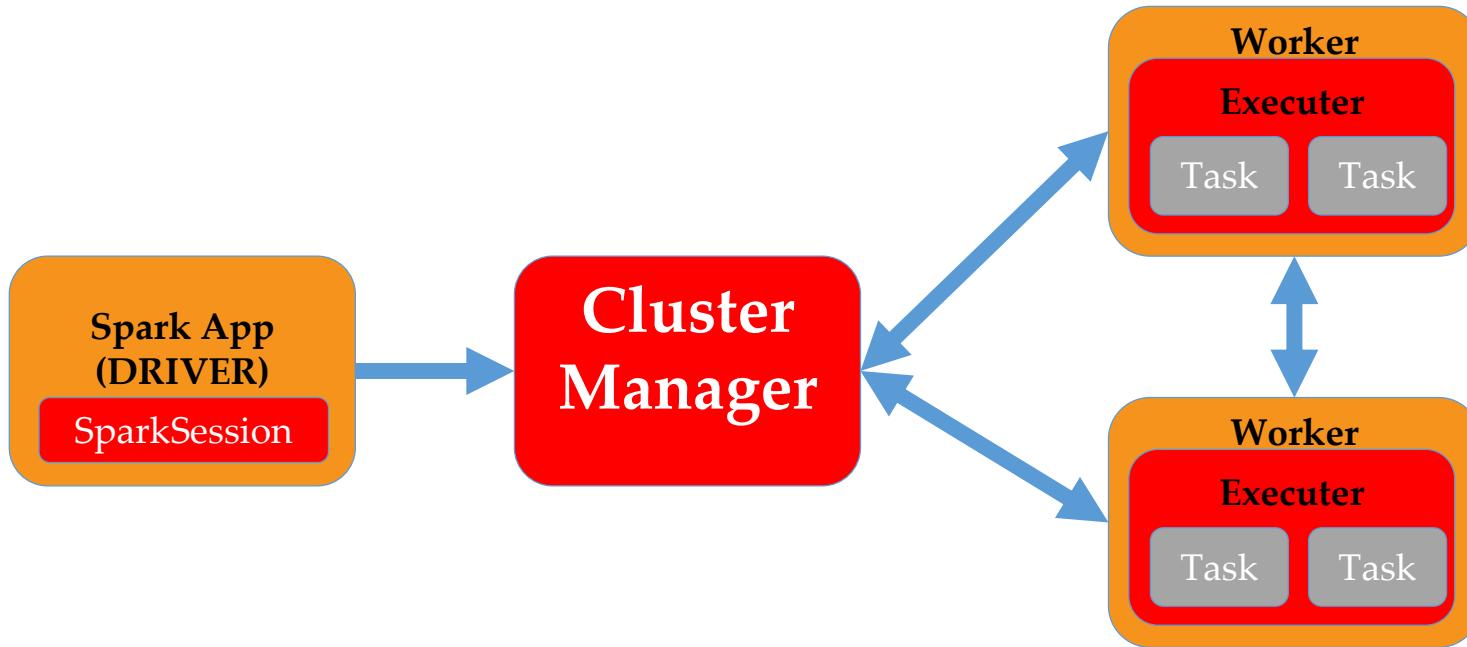
What are the advantages of using Apache Spark?

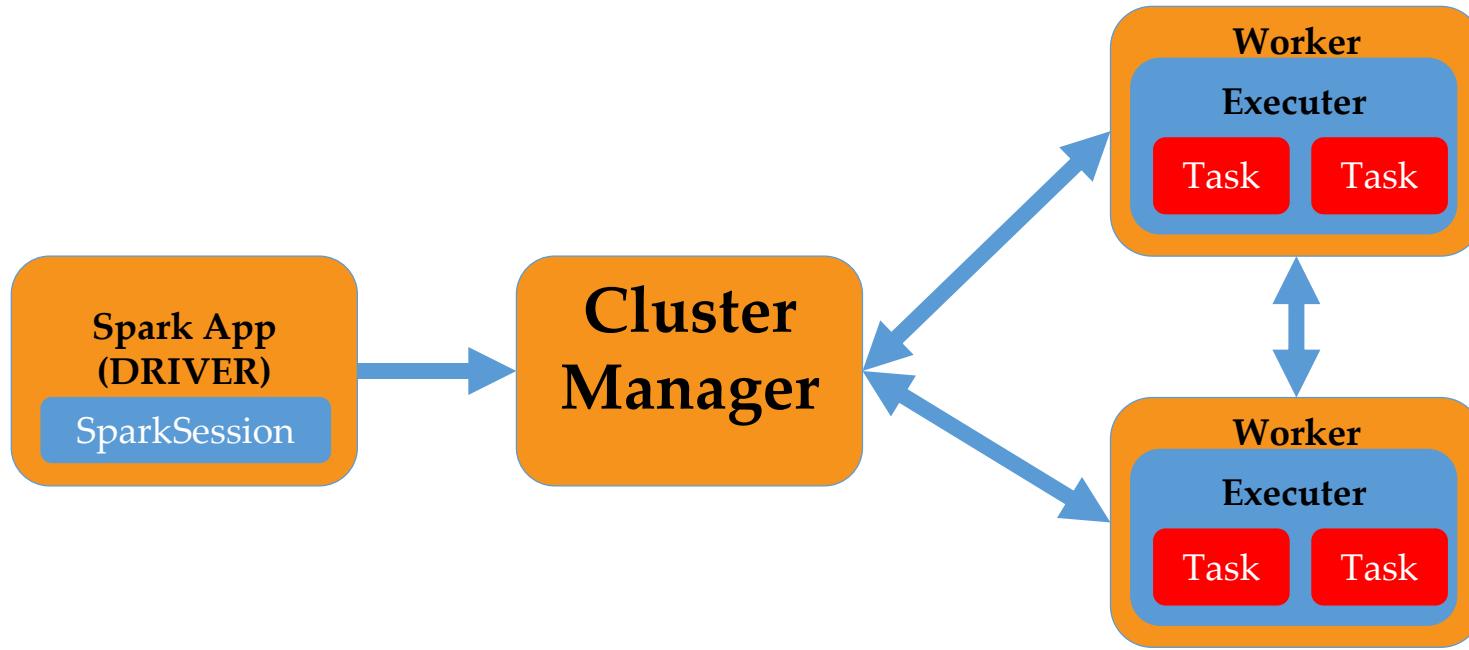
- A. Compatible with Hadoop
- B. Ease of development
- C. Fast
- D. Multiple language support
- E. Unified stack
- F. All of the above









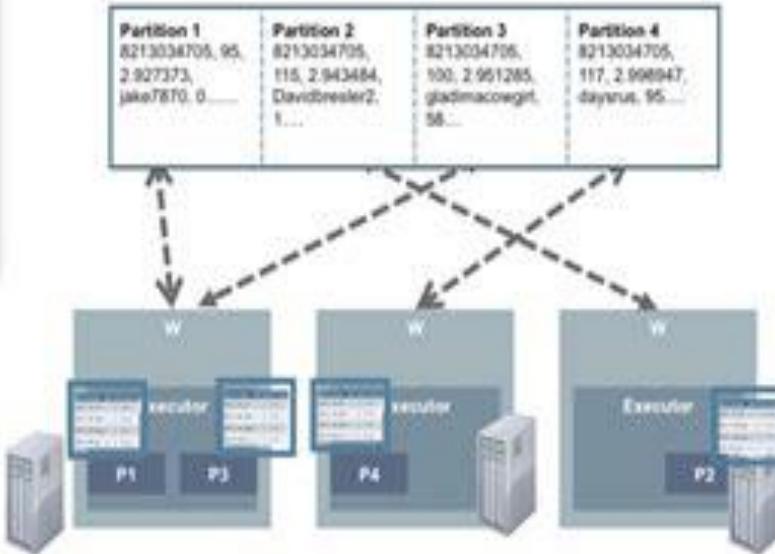


- Read only collection of **typed objects** `Dataset[T]`
- **Partitioned** across a cluster
- Operated on in **parallel**
- in memory can be **Cached**

partitioned

| Partition | Executor | Data |
|-----------|----------|--|
| P1 | W1 | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 |
| P2 | W1 | 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 |
| P3 | W2 | 21, 22, 23, 24, 25, 26, 27, 28, 29, 30 |
| P4 | W2 | 31, 32, 33, 34, 35, 36, 37, 38, 39, 40 |

Dataset



Transformations

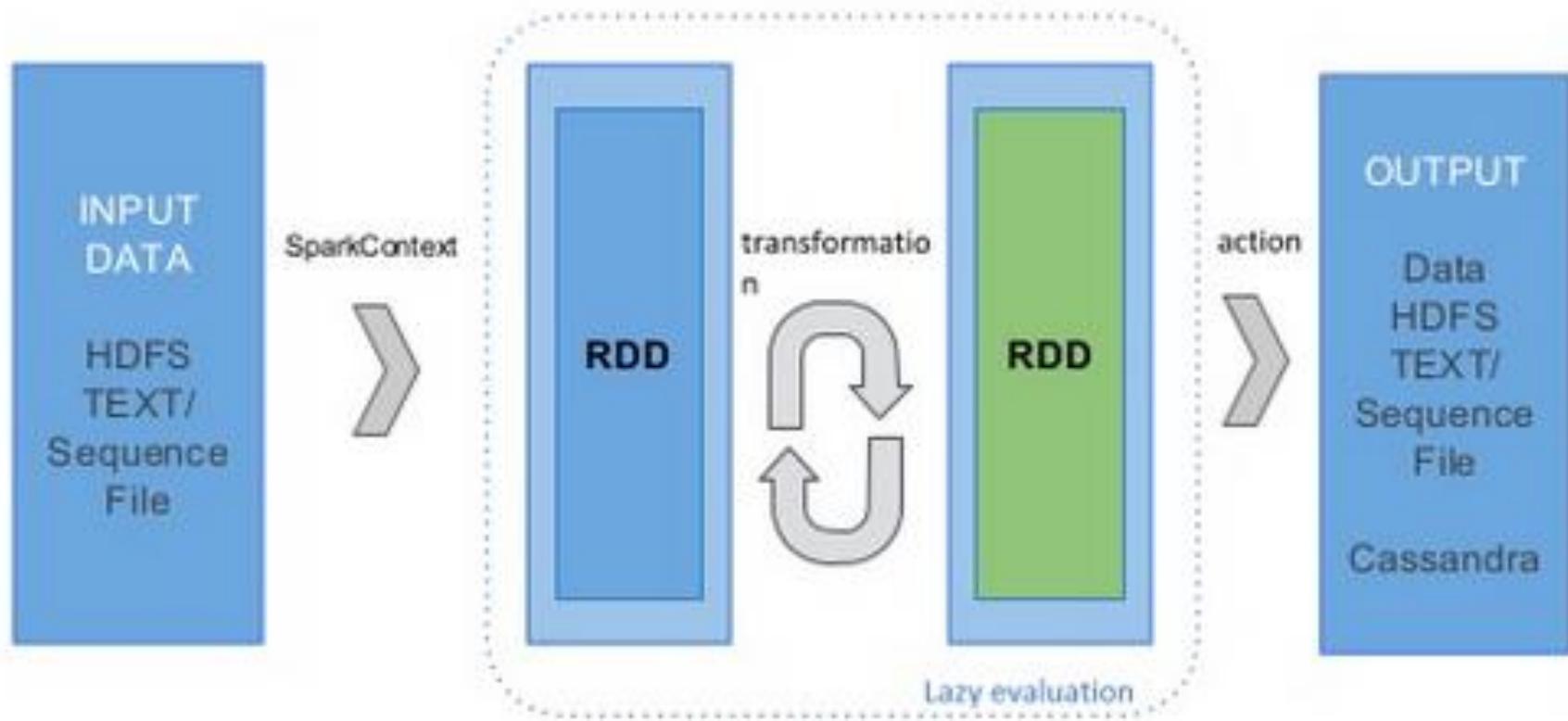
`map(func)`
`flatMap(func)`
`filter(func)`
`groupByKey()`
`reduceByKey(func)`
`mapValues(func)`

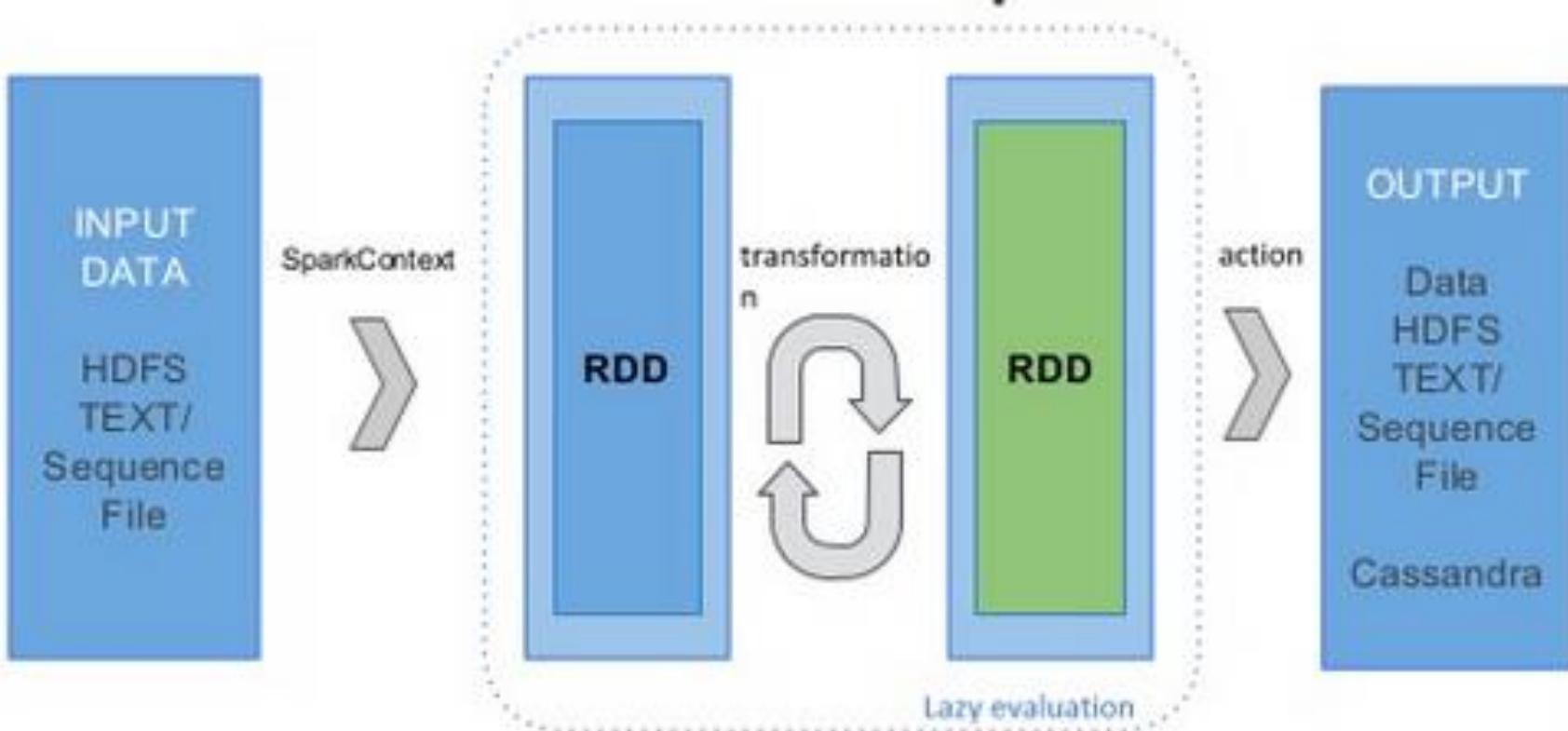
...

Actions

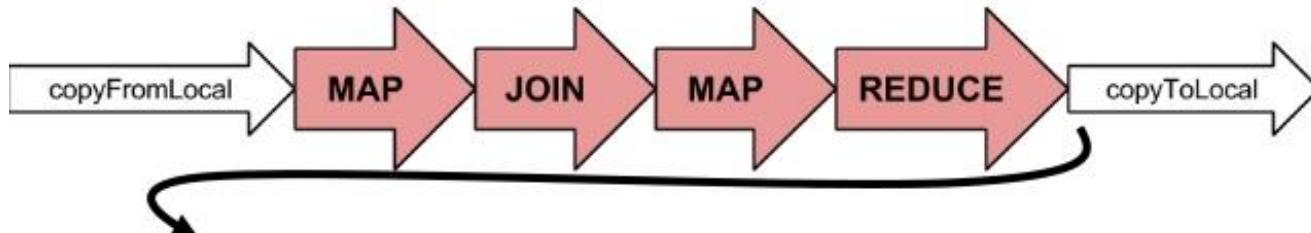
`take(N)`
`count()`
`collect()`
`reduce(func)`
`takeOrdered(N)`
`top(N)`

...





RDD Transformation vs. Action



- Must insert an *action* here to get pipeline to execute.
- Actions create files or objects:

```
# The saveAsTextFile action dumps the contents of an RDD to disk
>>> rdd.saveAsTextFile('hdfs://master.ibnet0/user/glock/output.txt')

# The count action returns the number of elements in an RDD
>>> num_elements = rdd.count();
      num_elements;
      type(num_elements)
215136
<type 'int'>
```

Fast engine for Hive
Interactive Queries

Real time streaming
data analysis

Machine learning
algorithms

Graph Processing
Algorithms

Spark SQL

Spark Streaming

MLLib

GraphX

Apache Spark Core Engine

Spark SQL

Spark SQL
RESULTS

Spark SQL Service



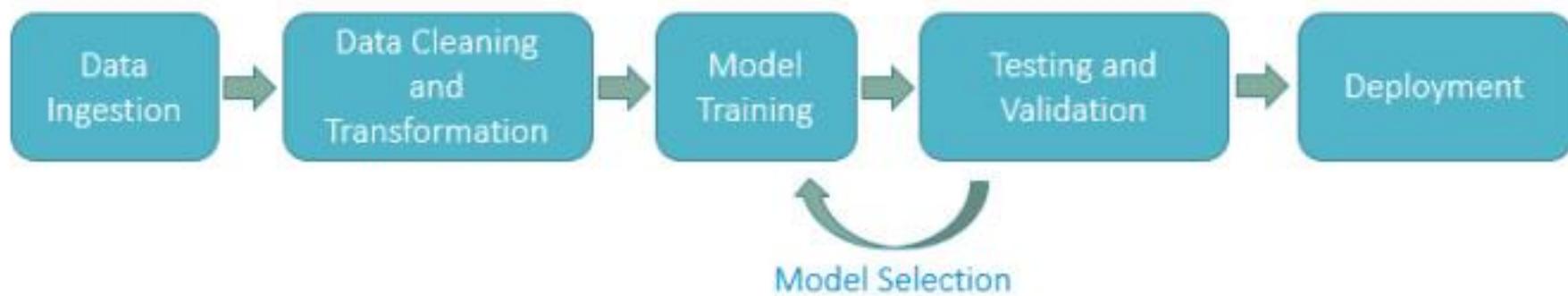
Interpreter &
Optimizer

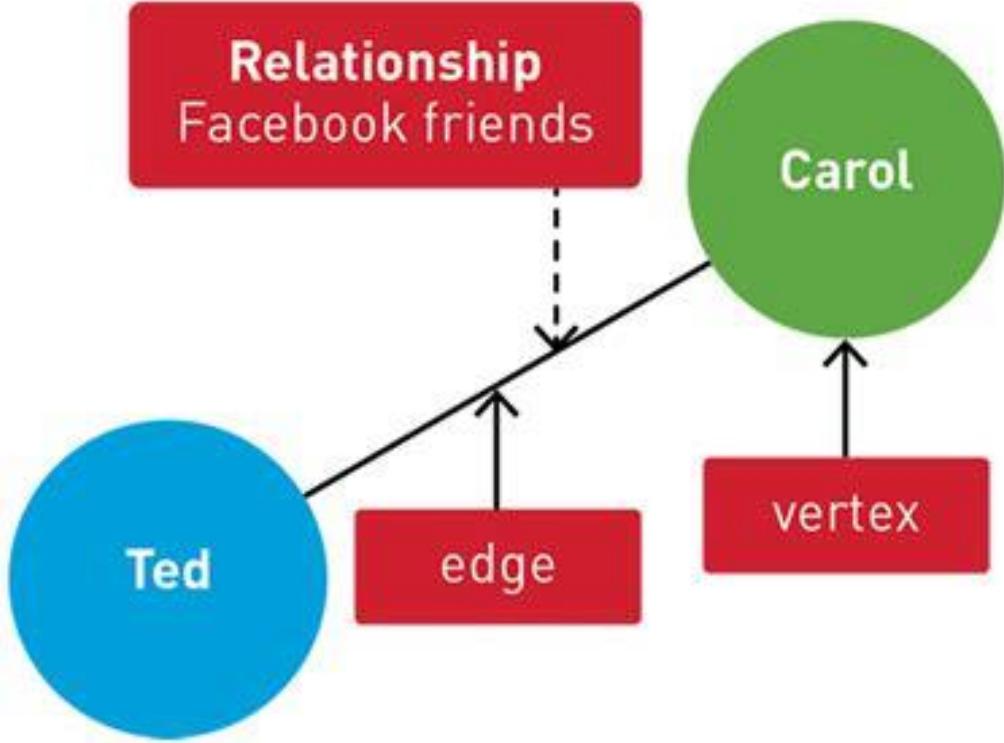


Figure: The flow diagram represents a Spark SQL process using all the four libraries in sequence.



APACHE Spark™ ML





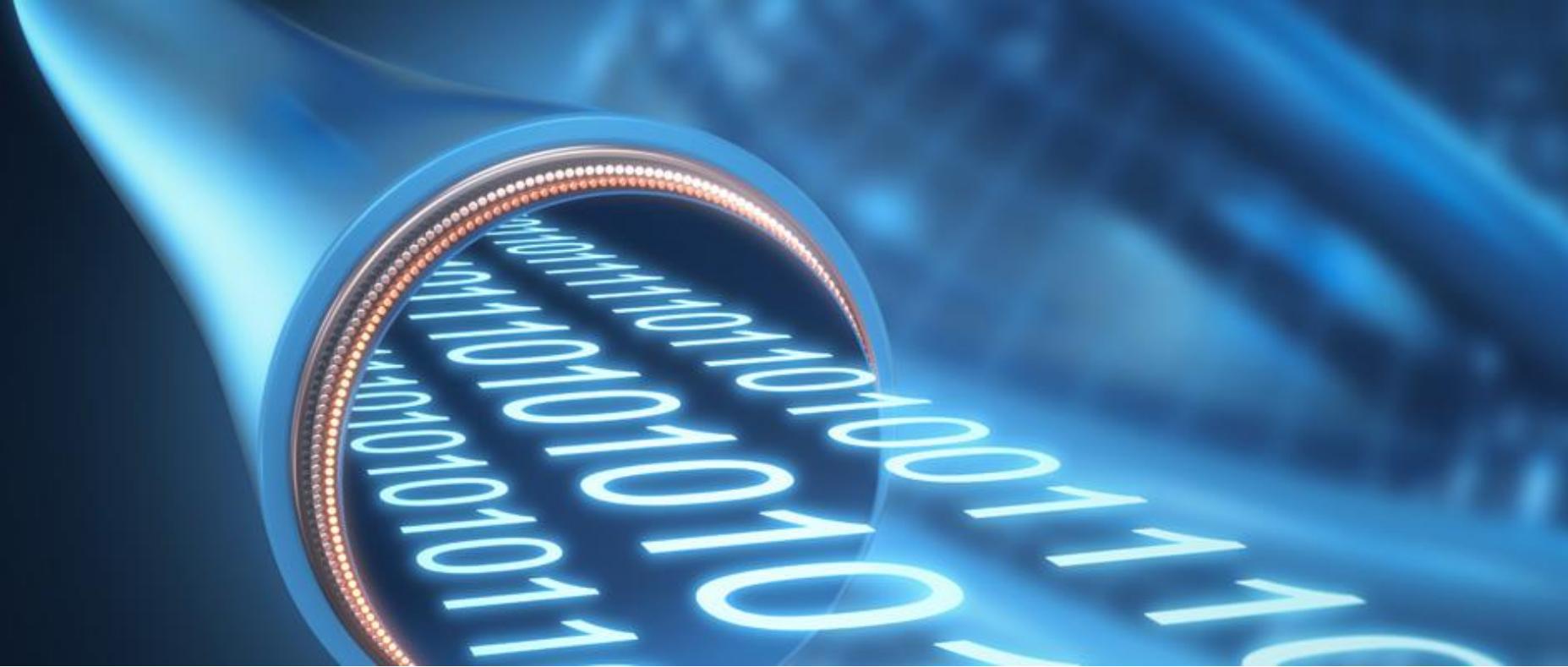
Knowledge Check



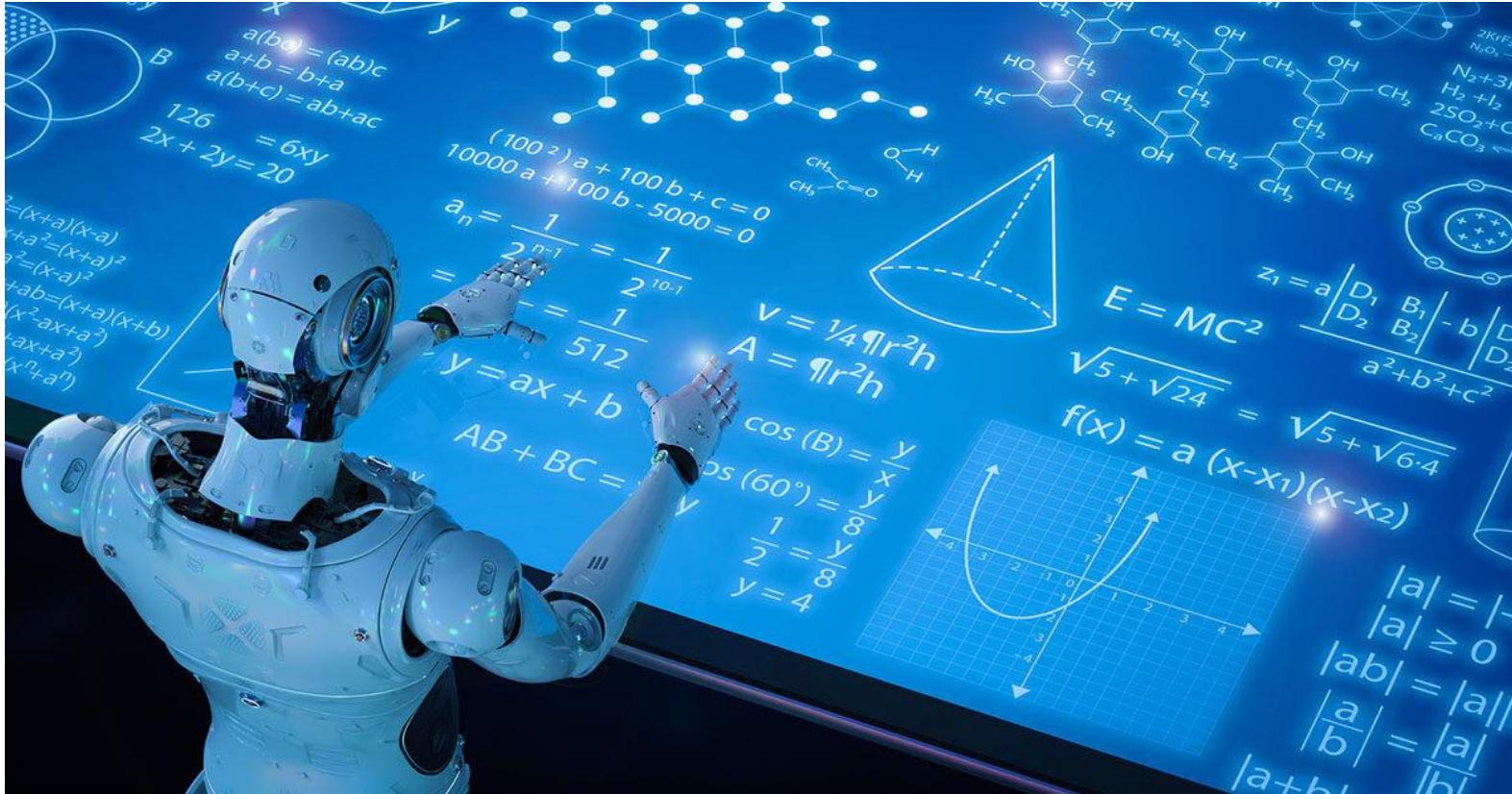
Match the following:

- Responsible for task scheduling, memory management A. SparkSession
- Tells Spark how and where to access a cluster B. Datasets
- Collection of objects distributed across many nodes in a cluster C. Spark Core

Streaming Data



Artificial Intelligence



Analytics

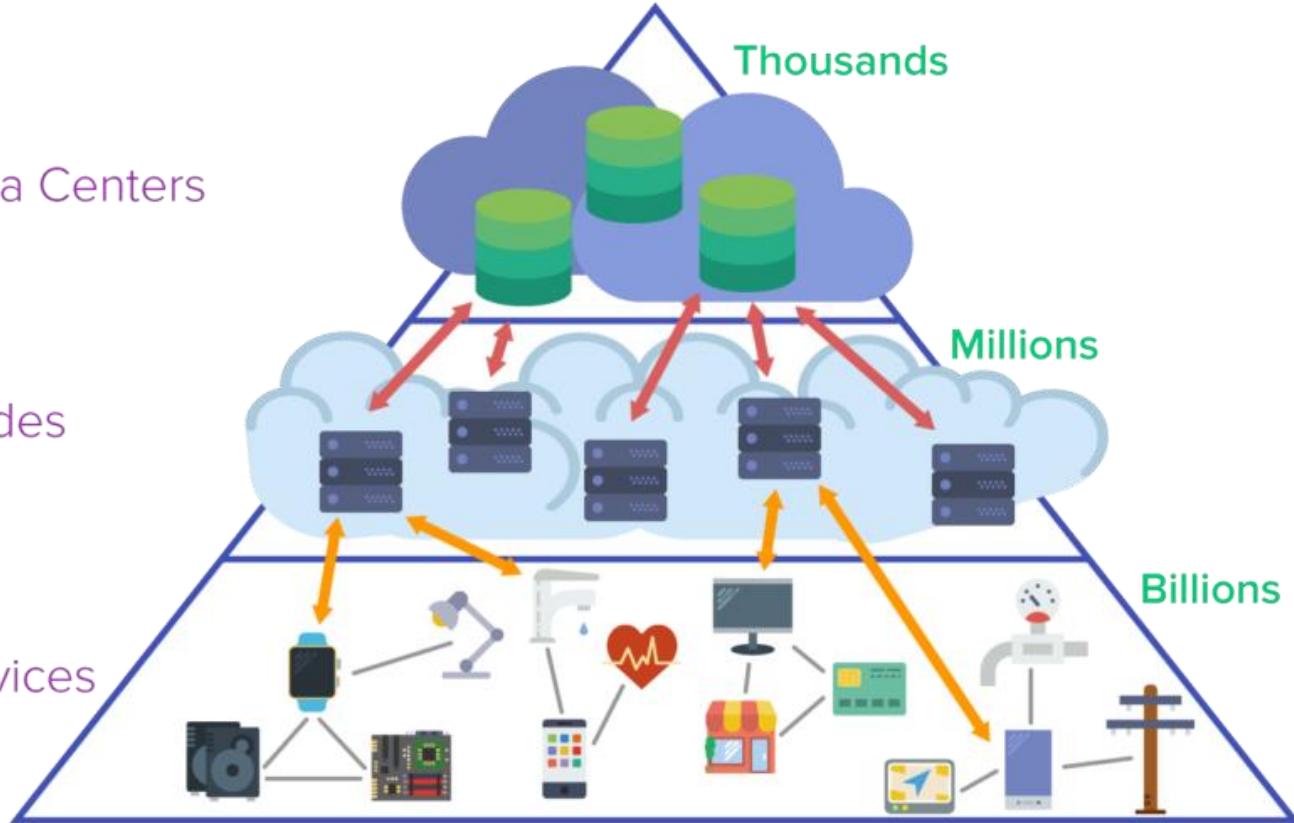


Fog Computing

CLOUD | Data Centers

FOG | Nodes

EDGE | Devices





Knowledge Check



We have a real-time Twitter feed. We need to build an application that is near real-time and classifies the Twitter feeds based on relevant and not relevant, where “relevant” means that it contains the words “FIFA”, “Women’s”, and “World Cup”. Which of the following Apache Spark libraries could we use in the application?

- A. Spark SQL
- B. Spark Streaming
- C. Spark MLlib
- D. Spark GraphFrames

Lesson 2: Create Datasets.



What you will learn

- Data Sources
- DataFrames and DataSets
- Casting

Data Sources supported by DataFrames

built-in



{ JSON }



JDBC



PostgreSQL



external



dBase



elasticsearch.



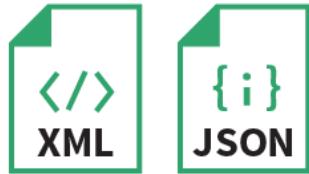
Amazon Redshift

and more ..

UNSTRUCTURED



SEMI-STRUCTURED



STRUCTURED



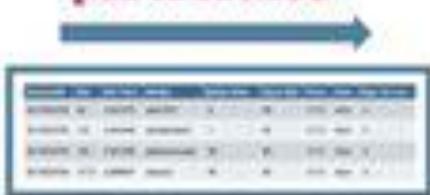
More flexible

More efficient storage and performance

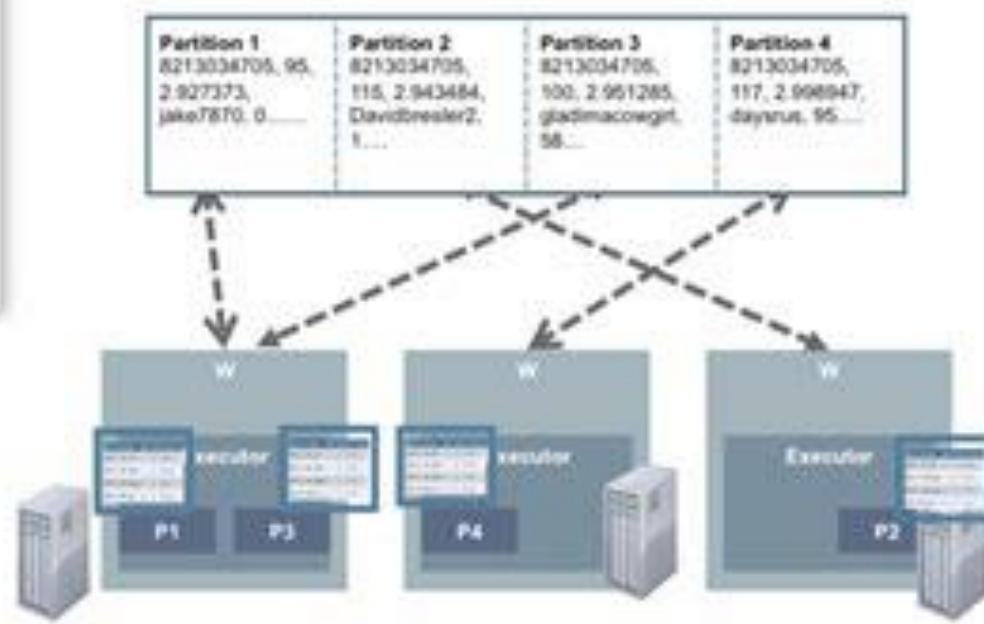


- Read only collection of **typed objects** **Dataset[T]**
- **Partitioned** across a cluster
- Operated on in **parallel**
- in memory can be **Cached**

partitioned

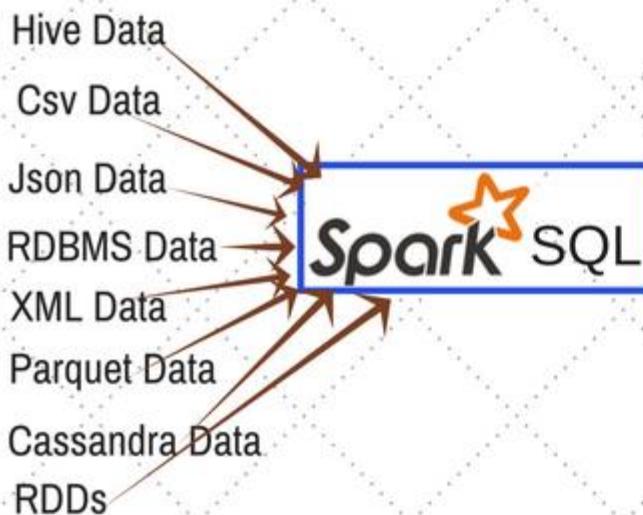


Dataset



- Primary abstraction in Spark
- Collection of objects distributed across a cluster
- Immutable once created
- Fault-tolerant
- Persist or cache in memory or on disk
- Data stored in tabular format

Ways to Create DataFrame in Spark



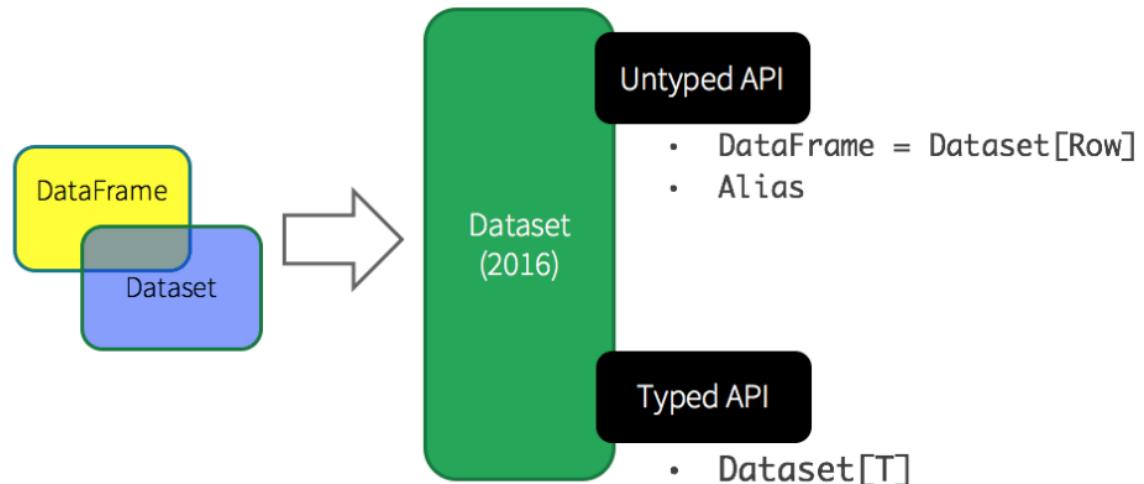
DataFrame

| | Col1 | Col2 | Col3 | |
|-------|------|------|------|------|
| Row 1 | | | | |
| Row 2 | | | | |
| Row 3 | | | | |
| : | | | | |

Dataframe vs Dataset

- **Dataframe**
 - Unknown Schema
 - Can be assigned programmatically
- **Dataset**
 - Known schema

Unified Apache Spark 2.0 API



DataFrame

`df.as[T]`

Dataset

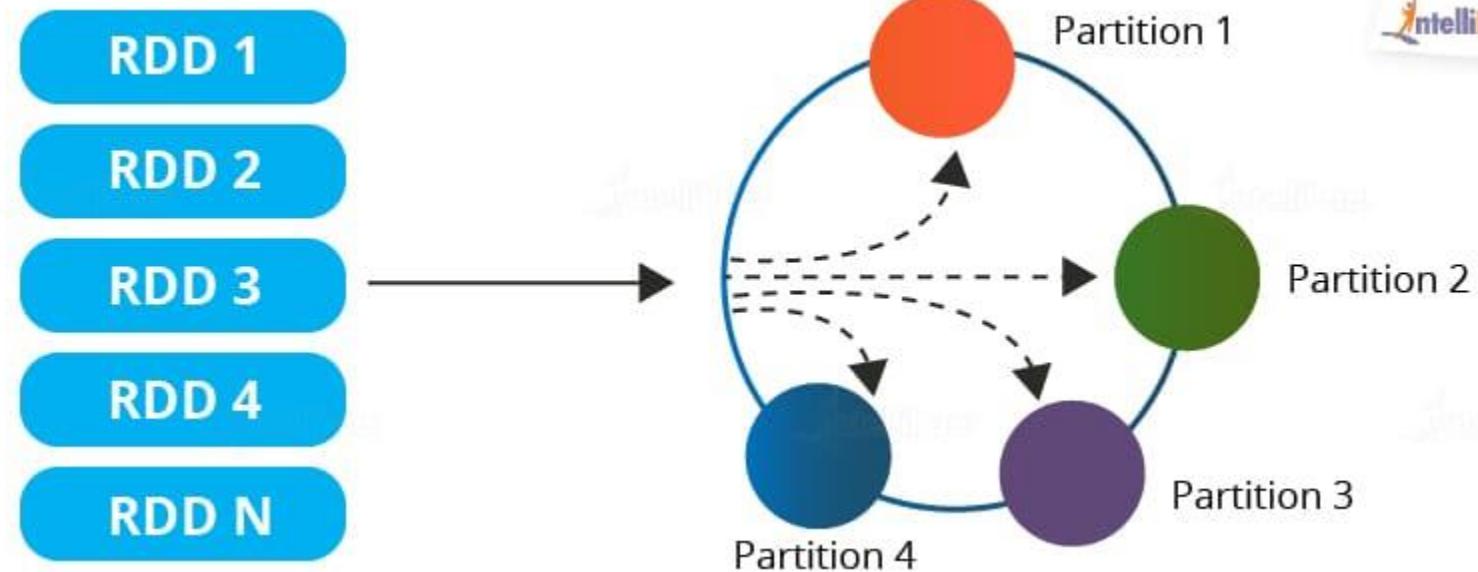
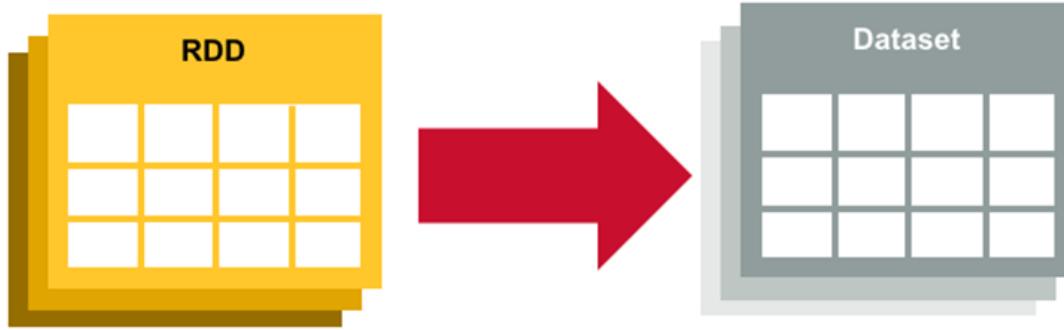
Unknown Data Type

Dataset [Row]

Known Data Type

Dataset [T]

```
case class MyCase(id: Int, name: String)  
  
val encoder = org.apache.spark.sql.Encoders.product[MyCase]  
  
val dataframe = ...  
  
val dataset = dataframe.as(encoder)
```



File Home Insert Page Layout Formulas Data Review View Help ACROBAT Tell me what you want to do

Cut Copy Format Painter

Calibri 11 Wrap Text General \$ % , .00 .00

B I U Font Alignment Number Conditional Formatting Table Cell Styles

Clipboard

| A1 | Arrest | Age | Sex | Race | ArrestDate | ArrestTim | ArrestLoc | IncidentO | IncidentLc | Charge | ChargeDe | District | Post | Neighbor | Location | |
|----|----------|----------|-----|------|------------|------------|------------|-----------------|------------|-----------|----------|---------------------------|-----------|-----------|-------------------------|------------------------|
| 1 | Arrest | 16160529 | 54 | M | B | 11/12/2016 | 22:35 | 3500 PELH | 4ECOMMC | 3500 PELH | 1 | 1415 | COMMON | Northeast | 432 | Belair-Edi (39.320868) |
| 2 | 16160490 | 22 | M | B | 11/12/2016 | 21:49 | 300 S LOUI | Unknown | 300 S LOUI | 4 | 3550 | POSSESS(| Southwes | 833 | Irvington (39.281148) | |
| 4 | 16160487 | 31 | M | B | 11/12/2016 | 21:40 | | Unknown Offense | | 1 | 0077 | FAILURE TO APPEAR | | | | |
| 5 | 16160485 | 31 | M | B | 11/12/2016 | 20:30 | | Unknown Offense | | 1 | 0077 | FAILURE TO APPEAR | | | | |
| 6 | 16160481 | 33 | M | B | 11/12/2016 | 19:45 | | Unknown Offense | | 2 | 0480 | MOTOR VEH/UNLAWFUL TAKING | | | | |
| 7 | 16160461 | 39 | F | W | 11/12/2016 | 17:50 | | Unknown Offense | | 1 | 0077 | FAILURE TO APPEAR | | | | |
| 8 | 16160451 | 54 | M | B | 11/12/2016 | 16:30 | | Unknown Offense | | 1 | 0077 | FAILURE TO APPEAR | | | | |
| 9 | 16160447 | 39 | M | W | 11/12/2016 | 16:30 | | Unknown Offense | | 1 | 0077 | FAILURE TO APPEAR | | | | |
| 10 | 16160449 | 46 | M | A | 11/12/2016 | 15:37 | 1500 RUSS | Unknown | 1500 RUSS | 2 | 2220 | TRESPASS | Southern | 941 | Carroll - C (39.274378) | |
| 11 | 16160438 | 20 | M | U | 11/12/2016 | 15:30 | 500 N LAK | 4ECOMMC | 500 N LAK | 1 | 1415 | COMMON | Southeast | 221 | McElderry (39.297000) | |
| 12 | 16160445 | 22 | F | B | 11/12/2016 | 15:30 | | Unknown Offense | | 1 | 1415 | AS AUL-S-C DECRE | | | | |
| 13 | 16160424 | 41 | F | B | 11/12/2016 | 14:20 | | Unknown Offense | | 1 | 0077 | FAILURE TO APPEA | | | | |
| 14 | 16160419 | 22 | M | B | 11/12/2016 | 13:55 | 5400 MAY | 4ECOMMC | 5400 MAY | 1 | 1415 | COMMON | Northeast | 444 | Frankford (39.333005) | |
| 15 | 16160406 | 35 | M | W | 11/12/2016 | 13:50 | | Unknown Offense | | 1 | 0621 | THEFT: LESS \$1,000 VALUE | | | | |
| 16 | 16160456 | 39 | M | B | 11/12/2016 | 13:34 | 2800 PATA | Unknown | 2800 W PA | 1 | 0521 | DESTRUCT | Southern | 923 | Lakeland (39.255302) | |

ERNESTO.NET

≡ LAB_2_1.sc X | ≡ LAB_2_1.py X | ≡ LAB_2_2.sc X | ≡ LAB_2_2.py X | s root@wk-c X | s root@wk-c X

```
scala> dataDS.groupBy("race").count().show()
[Stage 3:>
```

```
[Stage 10:=====
```

| race | count |
|------|--------|
| B | 106652 |
| U | 2660 |
| A | 343 |
| W | 20683 |
| I | 375 |

```
scala> d
```



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q |
|----|----------|-----|-----|------|------------|-----------|--------------|-----------------|------------|--------|---------------------------|-----------|------|-------------|---------------------------------|---|---|
| 1 | Arrest | Age | Sex | Race | ArrestDate | ArrestTim | ArrestLocati | IncidentO | IncidentLo | Charge | ChargeDe | District | Post | Neighbor | Location 1 | | |
| 2 | 16160523 | 54 | M | B | 11/12/2016 | 22:35 | 3500 PELHAM | 4E COMM | 5500 PELH | 1 1415 | COMMON | Northeast | 432 | Pelair-Edi | (39.3208685519, -76.5652449141) | | |
| 3 | 16160490 | 22 | M | B | 11/12/2016 | 21:49 | 300 S LOUDC | Unknown | 300 S LOU | 4 3550 | POSSESSI | Southwes | 833 | Irvington | (39.2811486601, -76.6821278085) | | |
| 4 | 16160487 | 31 | M | B | 11/12/2016 | 21:40 | | Unknown Offense | | 1 0077 | FAILURE TO APPEAR | | | | | | |
| 5 | 16160485 | 31 | M | B | 11/12/2016 | 20:30 | | Unknown Offense | | 1 0077 | FAILURE TO APPEAR | | | | | | |
| 6 | 16160481 | 33 | M | B | 11/12/2016 | 19:45 | | Unknown Offense | | 2 0480 | MOTOR VEH/UNLAWFUL TAKING | | | | | | |
| 7 | 16160461 | 39 | F | W | 11/12/2016 | 17:50 | | Unknown Offense | | 1 0077 | FAILURE TO APPEAR | | | | | | |
| 8 | 16160451 | 54 | M | B | 11/12/2016 | 16:30 | | Unknown Offense | | 1 0077 | FAILURE TO APPEAR | | | | | | |
| 9 | 16160447 | 39 | M | W | 11/12/2016 | 16:30 | | Unknown Offense | | 1 0077 | FAILURE TO APPEAR | | | | | | |
| 10 | 16160449 | 46 | M | A | 11/12/2016 | 15:37 | 1500 RUSSEL | Unknown | 1500 RUSS | 2 2220 | TRESPASS | Southern | 941 | Carroll - C | (39.2743782847, -76.6276296924) | | |
| 11 | 16160438 | 20 | M | U | 11/12/2016 | 15:30 | 500 N LAKERW | 4E COMM | 500 N LAK | 1 1415 | COMMON | Southeast | 221 | McElberry | (39.2970007586, -76.5793864662) | | |
| 12 | 16160445 | 22 | F | B | 11/12/2016 | 15:30 | | Unknown Offense | | 1 1415 | ASSAULT-SEC DEGREE | | | | | | |
| 13 | 16160424 | 41 | F | B | 11/12/2016 | 14:2 | | Unknown Offense | | 1 0077 | FAILURE TO APPEAR | | | | | | |
| 14 | 16160419 | 22 | M | B | 11/12/2016 | 13:53 | 3400 MAY | 4E COMM | 5-00 MAY | 1 1415 | COMMON | North Las | 444 | Lakeland | (39.30055387, -76.5459717682) | | |
| 15 | 16160406 | 35 | M | W | 11/12/2016 | 13:50 | | Unknown Offense | | 1 0621 | THEFT: LESS \$1,000 VALUE | | | | | | |
| 16 | 16160456 | 39 | M | B | 11/12/2016 | 13:34 | 2800 PATAPS | Unknown | 2800 W PA | 1 0521 | DESTRUCT | Southern | 923 | Lakeland | (39.2553025715, -76.6581366084) | | |
| 17 | 16160436 | 31 | M | B | 11/12/2016 | 13:00 | | Unknown Offense | | 1 0077 | FAILURE TO APPEAR | | | | | | |
| 18 | 16160431 | 31 | F | B | 11/12/2016 | 12:45 | | Unknown Offense | | 1 0088 | VIOLATION OF PROBATION | | | | | | |
| 19 | 16160403 | 21 | F | B | 11/12/2016 | 12:17 | | 4E COMM | 300 E MAD | 1 1415 | ASSAULT-SEC DEGREE | | | | | | |
| 20 | 16160378 | 19 | M | B | 11/12/2016 | 10:50 | | Unknown Offense | | 1 1420 | ASSAULT-FIRST DEGREE | | | | | | |
| 21 | 16160400 | 27 | M | W | 11/12/2016 | 10:49 | 3500 ANNAP | 6CLARCEN | 3300 ANN | 1 0521 | LARCENY | Southe | 922 | Cherry Hil | (39.2472331057, -76.6402786910) | | |
| 22 | | 20 | M | B | 11/12/2016 | 10:37 | 500 N DOTON | Unknown | Offense | | Unknown | Southeast | 224 | Ellwood D | (39.282877410, -76.5752206540) | | |

SCHEMA

*All data in a
tabular format*

How to Declare a Schema in Spark

- Use the Case Class
 - Create DS when the types and columns are known in advance of runtime (scala Case has a 22 field restriction)
- Programmatically
 - Create a DS or DF when types and columns are ONLY known at runtime
 - There is no field limit so you can use in that case also

How to Declare a Schema in Spark

- Use the Case Class
 - Create DS when the types and columns are known in advance of runtime (scala Case has a 22 field) restriction)
- Programmatically
 - Create a DS or DF when types and columns are ONLY known at runtime
 - There is no field limit so you can use in that case also

How to Declare a Schema in Spark

- Use the Case Class
 - Create DS when the types and columns are known in advance of runtime (scala Case has a 22 field) restriction)
- Programmatically
 - Create a DS or DF when types and columns are ONLY known at runtime
 - There is no field limit so you can use in that case also

Create a Dataset

Spark Shell



Create a RAW Dataframe

Spark Shell



Create a Dataframe with a Programmatically Defined Schema

Spark Shell



Cast Dataframe to a Dataset

Spark Shell



Create a Dataset

Spark Shell



Operate on variety of data sources through Dataset interface

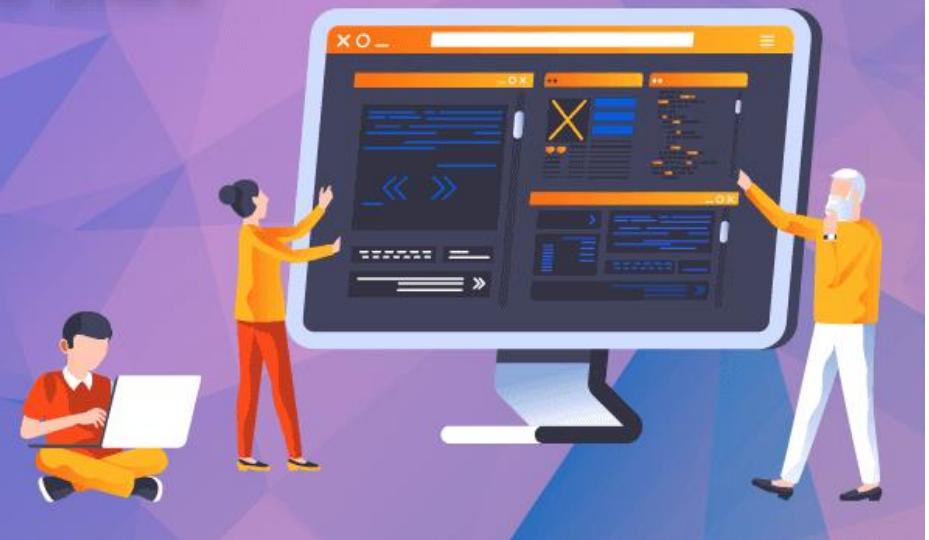
- Parquet
- JSON
- Hive Tables
- Relational Databases

Spark SQL - Schema RDD

Unified Interface For Structured Data



REPL Environment for Apache Spark Shell



```
root@wk-caas-d2813d52c4c44e5cb42c5907b9404989-f85d3c16adde1b40c633d4:~# 
```

Spark Interactive Shell

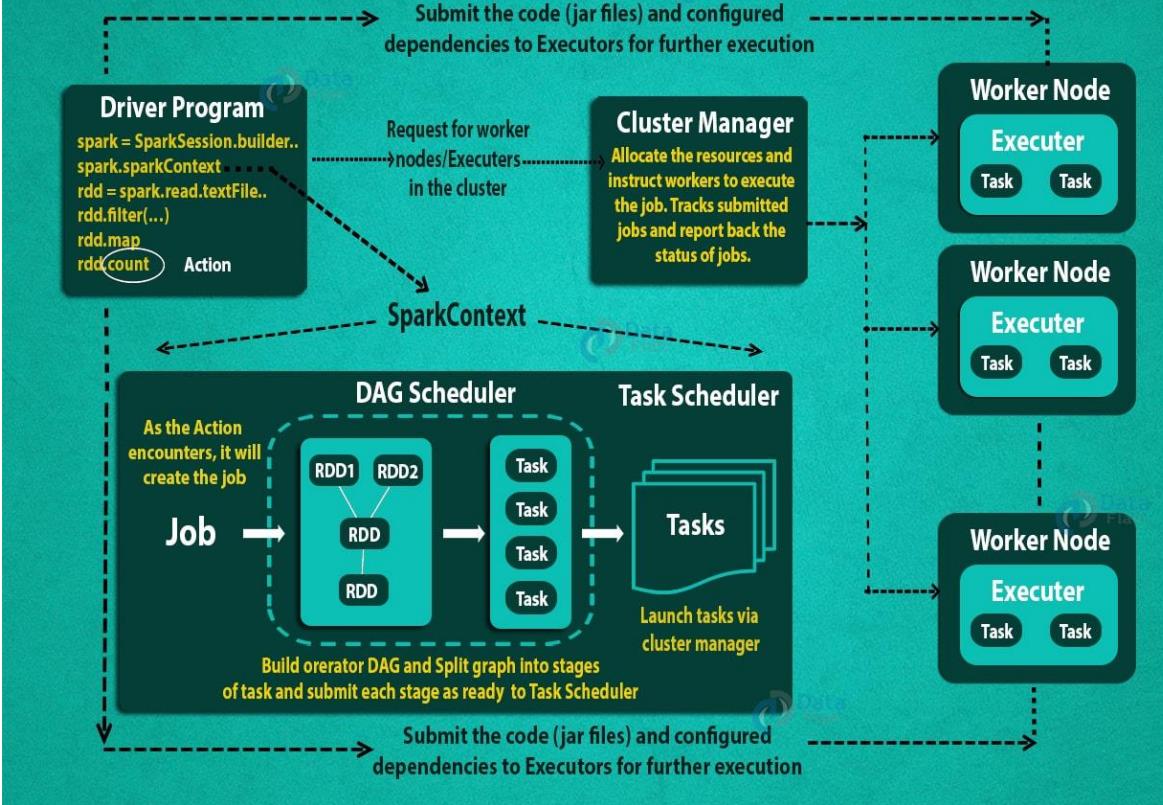
Scala REPL

- Interactive Development**
- Feedback is instant**

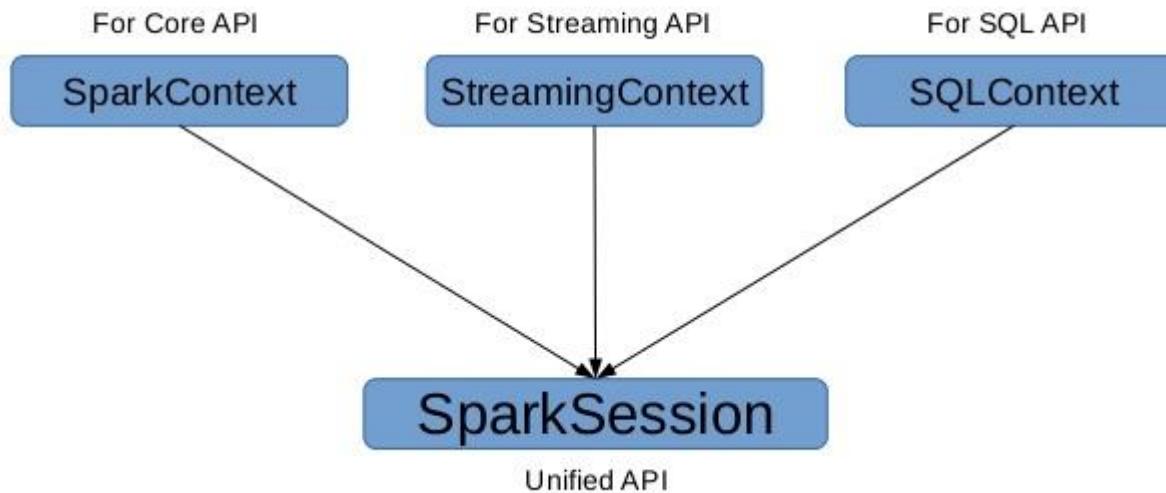
****VERY IMPORTANT - A
Spark Session is CREATED
when the REPL is started!**



Internals of Job Execution In Spark



What is SparkSession ?



Knowledge Check



Which of the following is true of the Spark Interactive Shell?

- A. Initializes SparkSession and makes it available
- B. Available in Java
- C. Provides instant feedback as code is entered
- D. Allows you to write programs interactively

Baltimore Police Department Use Case

- Based on the publicly available data from BPD:
 - What are the top 20 addresses with BPD calls?
 - What are the top 20 charges?
 - Districts?
Neighborhoods?



```
root@wk-caas-d2813d52c4c44e5cb42c5907b9404989-f85d3c16adde1b40c633d4:~# spark-shell ①
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
20/08/16 05:40:49 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Spark context Web UI available at http://10.244.22.5:4040 ②
Spark context available as 'sc' (master = local[*], app id = local-1597556449821).
Spark session available as 'spark'. ③
Welcome to ④

    / \   / \ / \ / \ / \
   / \ / \ / \ / \ / \ / \
  / \ . / \ , / \ / \ / \
 / \ / \ / \ / \ / \ / \
version 2.1.2 ⑤

⑥
Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_242) ⑦
Type in expressions to have them evaluated.
Type :help for more information.
```

Create a Dataset

1. Import Classes
2. Define Case Class
3. Load Data
4. Register Dataset as View (optional)

```
import spark.implicits._
```

```
case class Police(Arrest:Integer, Age:Integer, Sex:String, Race:String, ArrestDate:String,  
ArrestTime:String, ArrestLocation:String, IncidentOffense:String, IncidentLocation:String,  
Charge:String, ChargeDescription:String, District:String, Post:Integer, Neighborhood:String,  
Location:String)
```

```
val bpdDS = spark.read.option("header", true).csv("/home/jovyan/work/spark-  
dev3600/DEV360/LESSON_02/BPD_Arrests.csv").as[Police]
```

```
bpdDS.createTempView("bpd")
```



A screenshot of a terminal window with two tabs open. The left tab is titled "readme.md" and the right tab is titled "root@wk-caas-d2813d52c4:~#". The command prompt shows the user is root on a host named "wk-caas-d2813d52c4" with a session ID "f85d3c16adde1b40c633d4". The terminal is mostly empty, with a few small icons visible at the bottom.

```
import spark.implicits._
```

```
case class Police(Arrest:Integer, Age:Integer, Sex:String, Race:String, ArrestDate:String,  
ArrestTime:String, ArrestLocation:String, IncidentOffense:String, IncidentLocation:String,  
Charge:String, ChargeDescription:String, District:String, Post:Integer, Neighborhood:String,  
Location:String)
```

```
val bpdDS = spark.read.option("header", true).csv("/home/jovyan/work/spark-  
dev3600/DEV360/LESSON_02/BPD_Arrests.csv").as[Police]
```

```
bpdDS.createTempView("bpd")
```

```
import spark.implicits._

case class Police(Arrest:Integer, Age:Integer, Sex:String, Race:String,
ArrestDate:String, ArrestTime:String, ArrestLocation:String,
IncidentOffense:String, IncidentLocation:String, Charge:String,
ChargeDescription:String, District:String, Post:Integer, Neighborhood:String,
Location:String)

val bpdDS = spark.read.option("header", true).csv("/home/jovyan/work/spark-
dev360/DEV360/LESSON_02/BPD_Arrests.csv").as[Police]

bpdDS.createTempView("bpd")
```

```
import spark.implicits._
```

```
case class Police(Arrest:Integer, Age:Integer, Sex:String, Race:String, ArrestDate:String,  
ArrestTime:String, ArrestLocation:String, IncidentOffense:String, IncidentLocation:String,  
Charge:String, ChargeDescription:String, District:String, Post:Integer, Neighborhood:String,  
Location:String)
```

```
val bpdDS = spark.read.option("header", true).csv("/home/jovyan/work/spark-  
dev3600/DEV360/LESSON_02/BPD_Arrests.csv").as[Police]
```

```
bpdDS.createTempView("bpd")
```

```
// returns Dataset[String]
val ds:Dataset[String] = spark.read.textFile("src/main/resources/csv/text01.txt")
ds.printSchema()
ds.show(false)
```

```
//returns DataFrame  
val df:DataFrame = spark.read.text("src/main/resources/csv/text01.txt")  
df.printSchema()  
df.show(false)
```

Details: Load Data

- `spark.read.csv(path_to_CSV_file)`
 - Data type: CSV
 - Loads CSV data in path. Similar to `spark.read.load(path).format("csv")`
- `spark.read.json(path_to_JSON_file)`
 - Data type: JSON
 - Loads JSON data in path. Similar to `spark.read.load(path).format("json")`
- `spark.read.parquet(path_to_parquet_file)`
 - Data type: Parquet
 - Loads parquet data in path. Similar to
`spark.read.load(path).format("parquet")`

Knowledge Check



You can operate on the following data sources using the Dataset:

- A. Parquet Tables
- B. JSON
- C. Streaming Data
- D. JDBC

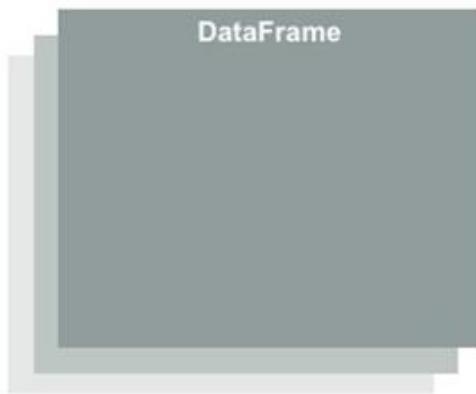
Spark Shell



!

Construct Schema Programmatically

- **Defines table schema**
 - Schema defined at runtime
- **Can be used**
 - When dynamic, based on certain conditions
 - When your Dataset includes more than 22 fields



Create DataFrame and Construct Schema Programmatically

1. Import Classes
2. Create Schema Programmatically
3. Create DataFrame by Loading Data
4. Register the DataFrame as a Table

Step 1: Import Classes

```
import spark.implicits._

import org.apache.spark.sql.types._

val sfpdSchema = StructType(Array(StructField("incidentnum",
StringType,true), StructField("category",StringType,true),
StructField("description",StringType,true),
StructField("dayofweek",StringType,true),
StructField("date",StringType,true), StructField("time",StringType,true),
StructField("pddistrict",StringType,true),
StructField("resolution",StringType,true),
StructField("address",StringType,true), StructField("X",DoubleType,true),
StructField("Y",DoubleType,true), StructField("pdid",StringType,true)))
...
...
```

Step 2: Create Schema Programmatically

```
import spark.implicits._

import org.apache.spark.sql.types._

val sfpdSchema = StructType(Array(StructField("incidentnum",
StringType,true), StructField("category",StringType,true),
StructField("description",StringType,true),
StructField("dayofweek",StringType,true),
StructField("date",StringType,true), StructField("time",StringType,true),
StructField("pddistrict",StringType,true),
StructField("resolution",StringType,true),
StructField("address",StringType,true), StructField("X",DoubleType,true),
StructField("Y",DoubleType,true), StructField("pdid",StringType,true))))
```

Step 3: Create DataFrame

```
import spark.implicits._

import org.apache.spark.sql.types._

val sfpdSchema = StructType(Array(StructField("incidentnum",
...
val sfpdDF =
spark.read.format("csv").schema(sfpdSchema).load("/spark/lab4/sfpd.csv").toDF("incidentnum", "category", "description", "dayofweek", "date", "time", "pddistrict", "resolution", "address", "X", "Y", "pdid")
sfpdDF.createTempView("sfpd")
```

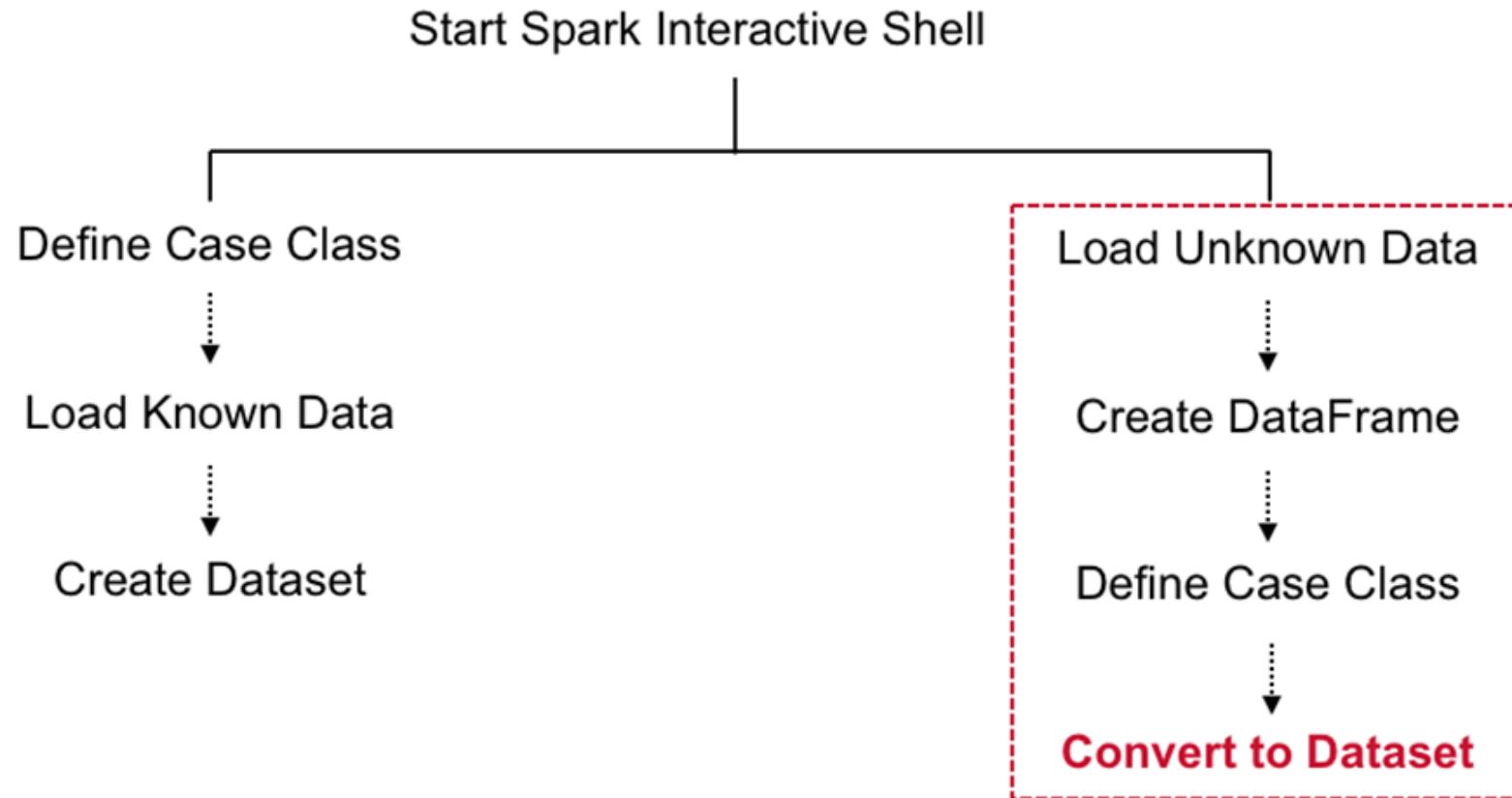
Step 4: Register the DataFrame as a Table (optional)

```
import spark.implicits._

import org.apache.spark.sql.types._

val sfpdSchema = StructType(Array(StructField("incidentnum",
...
val sfpdDF =
spark.read.format("csv").schema(sfpdSchema).load("/spark/lab4/sfpd.csv").t
oDF("incidentnum", "category", "description", "dayofweek", "date", "time",
"pddistrict", "resolution", "address", "X", "Y", "pdid")
sfpdDF.createTempView("sfpd")
```

Datasets vs. DataFrames



Infer Schema by Reflection

Infer schema by reflection

- Using Case Classes
- Use when schema is known

Load Unknown Data

Create DataFrame

Define Case Class

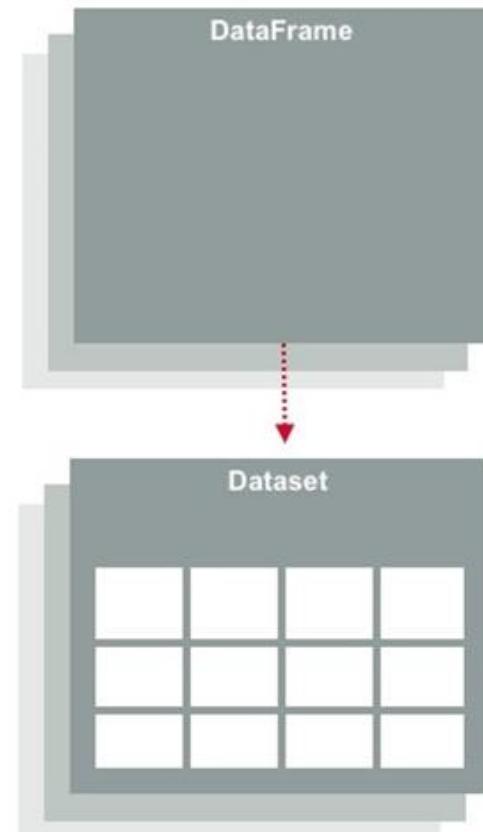
Convert to Dataset

Infer Schema by Reflection: Case Class

Defines table schema

- Column names of DF are matched with names of Case Class using reflection
- Names become name of column

| Incident Num | Category | Descript | DayOf Week |
|--------------|----------------|--------------------------------------|------------|
| 150599321 | OTHER_OFFENSES | POSSESSION_OF_BURGLARY_TOOLS | Thurs |
| 156168837 | LARCENY/THEFT | PETTY_THEFT_OF_PROPERTY | Thurs |
| 150599224 | OTHER_OFFENSES | DRIVERS_LICENSE/SUSPENDED_OR_REVOKED | Thurs |
| 150599230 | VANDALISM | MALICIOUS_MISCHIEF/BREAKING_WINDOWS | Thurs |

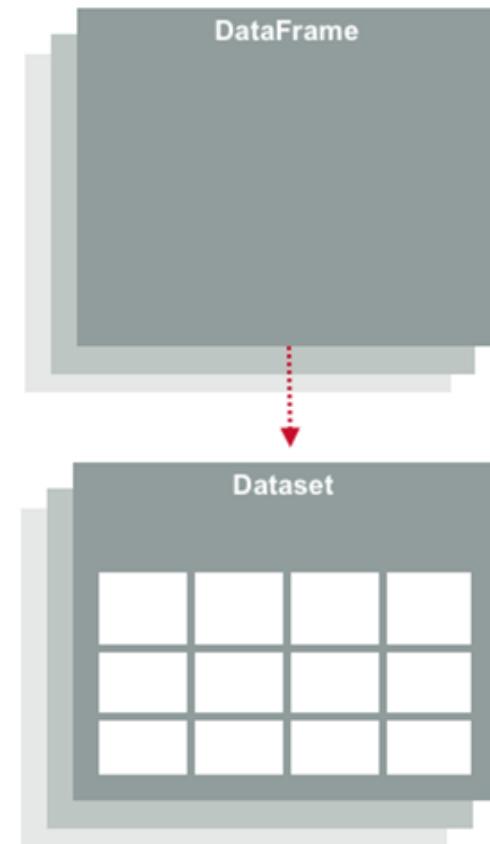


Infer Schema by Reflection: Case Class

Can be

- Nested
- Contain complex data (Sequences or Arrays)

```
case class Address (pddistrict:String,  
address:String, pdid:String)  
  
case class Detail(address:Address,  
incidentnum:String, category:String,  
description:String, resolution:String, X:double,  
Y:double)
```



Infer Schema by Reflection

1. Import Classes
2. Create DataFrame by Loading Data
3. Define Case Class
4. Convert DataFrame (Dataset[Rows]) into Dataset (Dataset[T]) using Case Class
5. Register Dataset as view (optional)

Step 1: Import Classes

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum",
"category", "desc", "date", "pddistrict", "resolution")

case class Incidents(incidentnum:String, category:String,
desc:String, date:String,pddistrict:String, resolution:String)

sfpdDS = sfpdDF.as[Incidents]
sfpdDS.createTempView("sfpd")
```

Step 2: Create DataFrame by Loading Data

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum",
"category", "desc", "date", "pddistrict", "resolution")

case class Incidents(incidentnum:String, category:String,
desc:String, date:String,pddistrict:String, resolution:String)

sfpdDS = sfpdDF.as[Incidents]
sfpdDS.createTempView("sfpd")
```

Step 3: Define Case Class

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum",
"category", "desc", "date", "pddistrict", "resolution")

case class Incidents(incidentnum:String, category:String,
desc:String, date:String,pddistrict:String, resolution:String)

sfpdDS = sfpdDF.as[Incidents]
sfpdDS.createTempView("sfpd")
```

Step 4: Convert DataFrame to Dataset Using Case Class

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum",
"category", "desc", "date", "pddistrict", "resolution")

case class Incidents(incidentnum:String, category:String,
desc:String, date:String,pddistrict:String, resolution:String)

sfpdDS = sfpdDF.as[Incidents]

sfpdDS.createTempView("sfpd")
```

Step 5: Register Dataset as View (optional)

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum",
"category", "desc", "date", "pddistrict", "resolution")

case class Incidents(incidentnum:String, category:String,
desc:String, date:String,pddistrict:String, resolution:String)

sfpdDS = sfpdDF.as[Incidents]
sfpdDS.createTempView("sfpd")
```

Knowledge Check



Which of the following statements are true when converting a DataFrame to a Dataset?

- A. Must infer the schema by reflection
- B. Must infer the schema programmatically
- C. Must know the schema
- D. Must use the Case Class
- E. Can be over 22 fields

Lazy Evaluation

DEFINE DATASET

```
val sfpdDS = spark.read.option("inferSchema",  
true).csv("/spark/data/sfpd.csv").toDF("incidentnum",  
"category", "description", "dayofweek", "date", "time",  
"pddistrict", "resolution", "address", "X", "Y",  
"pdid").as[Incidents]
```

- Location of data to load
- Defines schema

DEFINE TRANSFORMATIONS

RUN ACTION



Labs 2.3a and 2.3b

- Estimated time to complete: **50 minutes**
- This lab consists of two parts:
 - In Lab 2.3a (20 minutes) you will load data using the Scala shell, and create Datasets using reflection.
 - In Lab 2.3b (30 minutes) you will implement Word Count using Datasets. Start on this lab after completing 2.3a. If you do not finish Lab 2.3b, you can complete it on your own outside of class.

Lesson 3: Apply Operations on Datasets.





Learning Goals

3.1 Apply Operations on Datasets

3.2 Cache Datasets

3.3 Create User Defined Functions

3.4 Repartition Datasets



Two types of data operations can be performed on a Dataset:

TRANSFORMATION



ACTION



Exploring the Data

- What are the top five addresses with most incidents?
- What are the top five districts with most incidents?
- What are the top 10 resolutions?
- What are the top 10 categories of incidents?

Dataset Operations: Transformations



Commonly Used Transformations

| Transformation | Definition |
|-------------------------|---|
| <code>map()</code> | Returns new Dataset by applying func to each element of source |
| <code>filter()</code> | Returns new Dataset consisting of elements from source on which function is true |
| <code>groupBy()</code> | Returns Dataset (K, Iterable<V>) where the data is grouped by the given key func. |
| <code>reduce()</code> | Reduces the elements of this Dataset using the specified binary function. |
| <code>flatMap()</code> | Similar to map(), but function should return a sequence rather than a single item |
| <code>distinct()</code> | Returns new Dataset containing distinct elements of source |

Commonly Used Transformations (cont.)

| Transformation | Definition |
|--|--|
| <code>cache()</code> | Cache this Dataset |
| <code>persist()</code> | Persist this DataFrame |
| <code>createTempView (viewName)</code> | Registers this Dataset as a temporary view using the given name |
| <code>describe()</code> | Calculates count, mean, sttdev, min, and max for numeric columns |

Language Integrated Queries

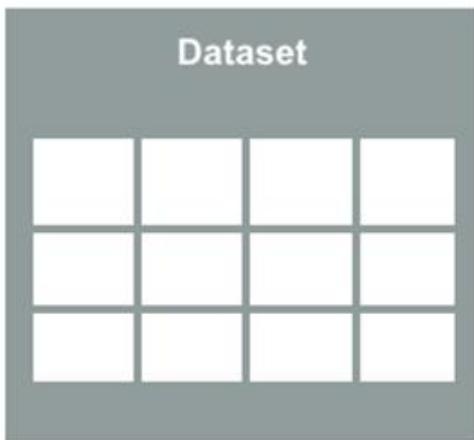
| Transformation | Definition |
|-------------------------------------|--|
| <code>agg(expr, exprs)</code> | Aggregates on the entire Dataset without groups |
| <code>filter(condition Expr)</code> | Filters based on given SQL expression |
| <code>groupBy(col1, cols)</code> | Groups Dataset using the specified columns so we can run aggregation on them |
| <code>select(cols)</code> | Selects a set of columns based on expressions |

Relational Grouped Datasets

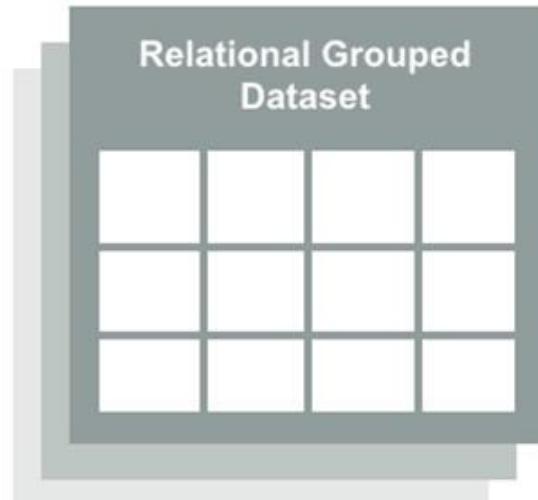
TRANSFORMATIONS



groupBy()

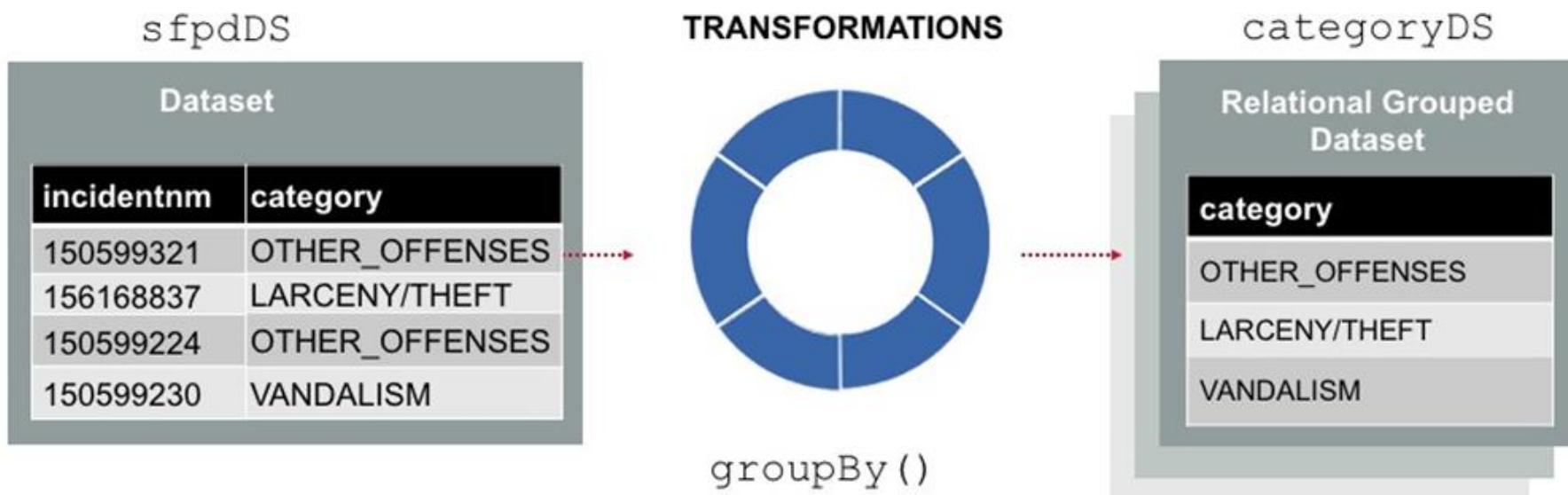


Object:
org.apache.spark.sql.RelationalGroupedDataset



Dataset Operations: Transformations

```
sfpdDS.groupBy("category")
```



Dataset Operations: Transformations vs. Actions

categoryDS

| Relational Grouped Dataset | |
|----------------------------|--|
| category | |
| OTHER_OFFENSES | |
| LARCENY/THEFT | |
| VANDALISM | |

TRANSFORMATIONS



count ()

countDS

| Dataset | |
|----------------|-------|
| category | count |
| OTHER_OFFENSES | 50611 |
| LARCENY/THEFT | 96955 |
| VANDALISM | 17987 |



Class Discussion

- Scenario: A sample of data in `sfpd.csv` is shown here. Each line represents a particular incident in San Francisco. We only want to look at incidents in the Southern district.

Q. What transformation could we use to get incidents only in the Southern district?

| | | | | | | |
|-----------|------------|-------------|----------|--------|-------|----------|
| 150598652 | WARRANTS | WARRANT_A | Thursday | 7/9/15 | 19:32 | SOUTHERN |
| 150598652 | ASSAULT | THREATS_AC | Thursday | 7/9/15 | 19:32 | SOUTHERN |
| 150598652 | WEAPON_LA | EXHIBITING_ | Thursday | 7/9/15 | 19:32 | SOUTHERN |
| 150599343 | LARCENY/TH | PETTY_THEF | Thursday | 7/9/15 | 19:30 | SOUTHERN |
| 150598583 | LARCENY/TH | GRAND_THE | Thursday | 7/9/15 | 19:30 | MISSION |
| 150598834 | ASSAULT | THREATS_AC | Thursday | 7/9/15 | 19:30 | NORTHERN |
| 150598834 | OTHER_OFFE | VIOLATION_I | Thursday | 7/9/15 | 19:30 | NORTHERN |
| 150599014 | LARCENY/TH | PETTY_THEF | Thursday | 7/9/15 | 19:15 | CENTRAL |
| 156168398 | NON-CRIMIN | LOST_PROPE | Thursday | 7/9/15 | 19:15 | CENTRAL |

Transformation: filter()

```
val districtDS=sfpdDS.filter(line=>line.contains("SOUTHERN"))
```

districtDS

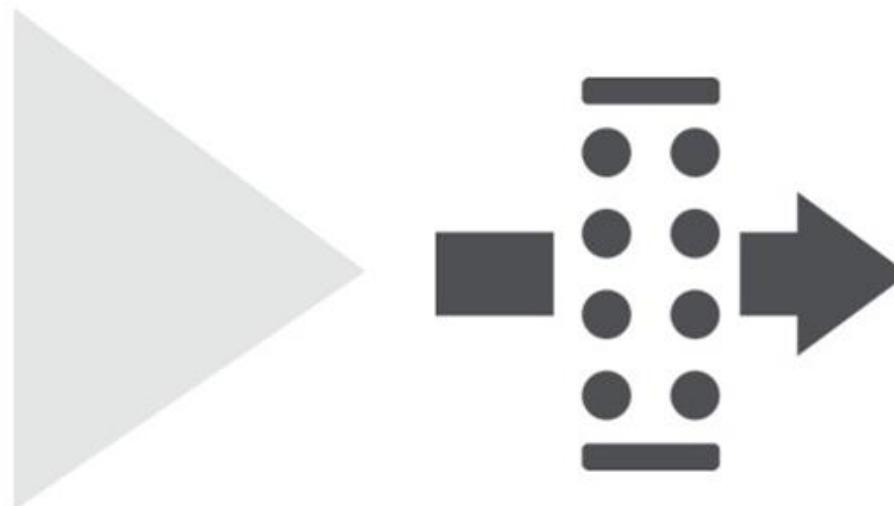
SOUTHERN

MISSION

NORTHERN

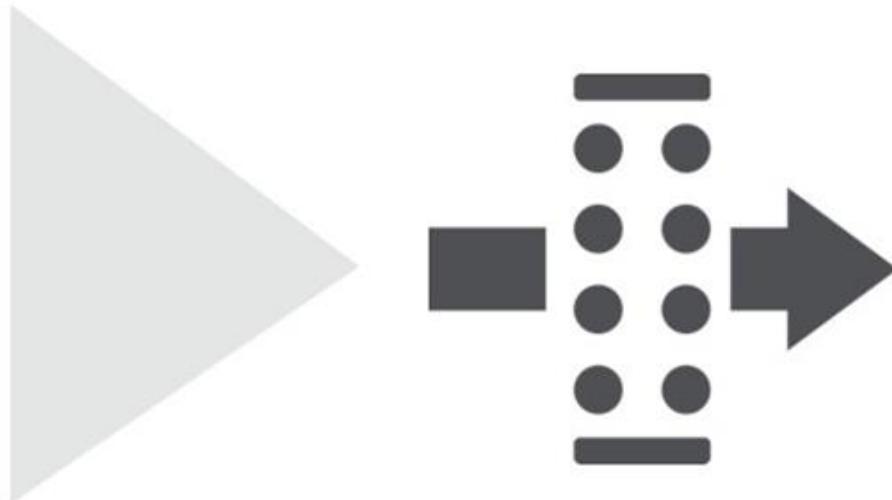
SOUTHERN

CENTRAL



Transformation: filter()

```
val districtDS=sfpdDS.filter(line=>line.contains("SOUTHERN"))
```



Transformation: filter() with Anonymous Function

```
val districtDS=sfpdDS.filter(line=>line.contains("SOUTHERN"))
```

districtDS

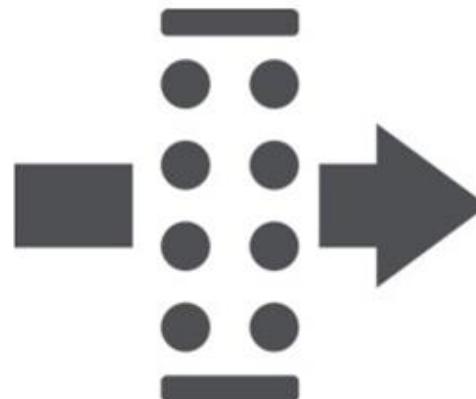
SOUTHERN

MISSION

NORTHERN

SOUTHERN

CENTRAL



=>

Anonymous Syntax



Transformation: filter()

```
val districtDS=sfpdDS.filter(line=>line.contains("SOUTHERN"))
```

districtDS

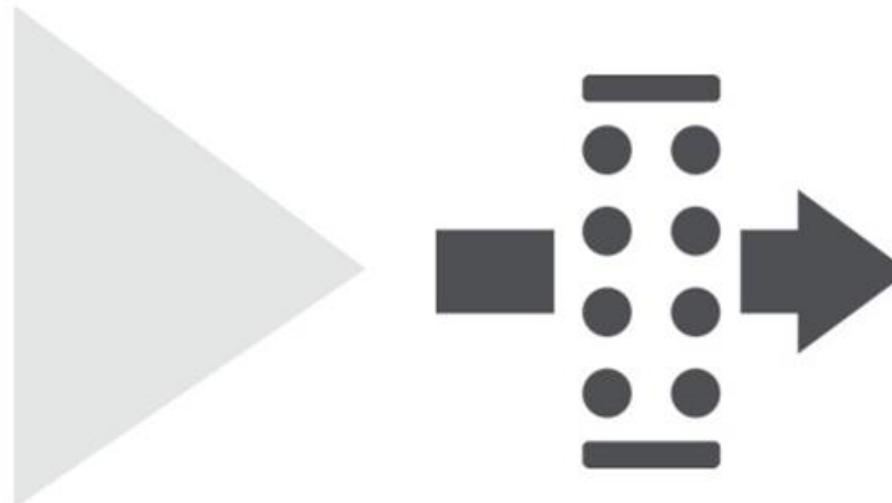
SOUTHERN

MISSION

NORTHERN

SOUTHERN

CENTRAL



Transformation: filter()

```
val southernDS=sfpdDS.filter(line=>line.contains("SOUTHERN"))
```

districtDS

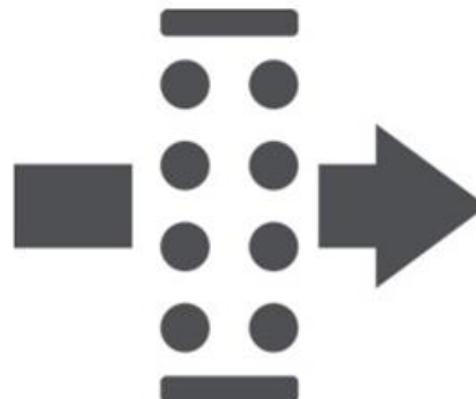
SOUTHERN

MISSION

NORTHERN

SOUTHERN

CENTRAL



southernDS

SOUTHERN

SOUTHERN

Review



Load data



`spark.read.text`

`districtDS`

Transform data



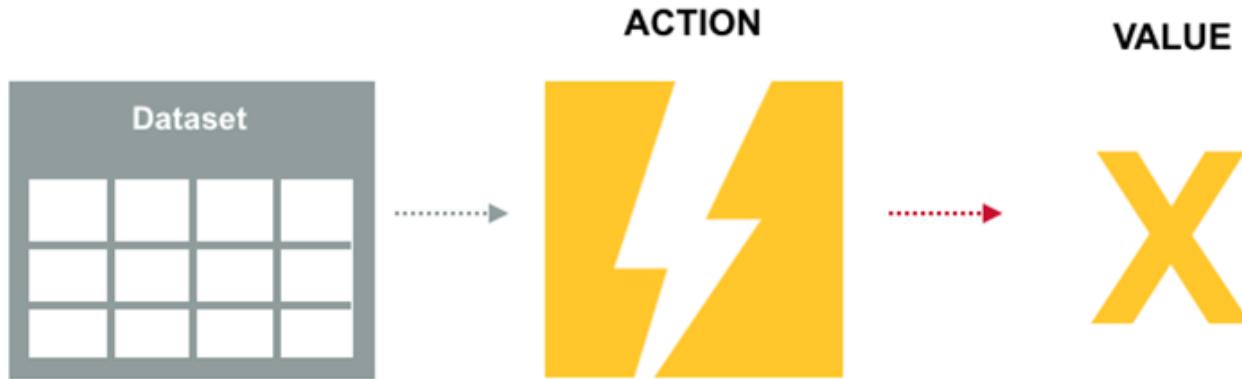
Want to see only
SOUTHERN incidents
`filter()`

`southernDS`



Lazy evaluation: No transformations are executed until an action is called

Review



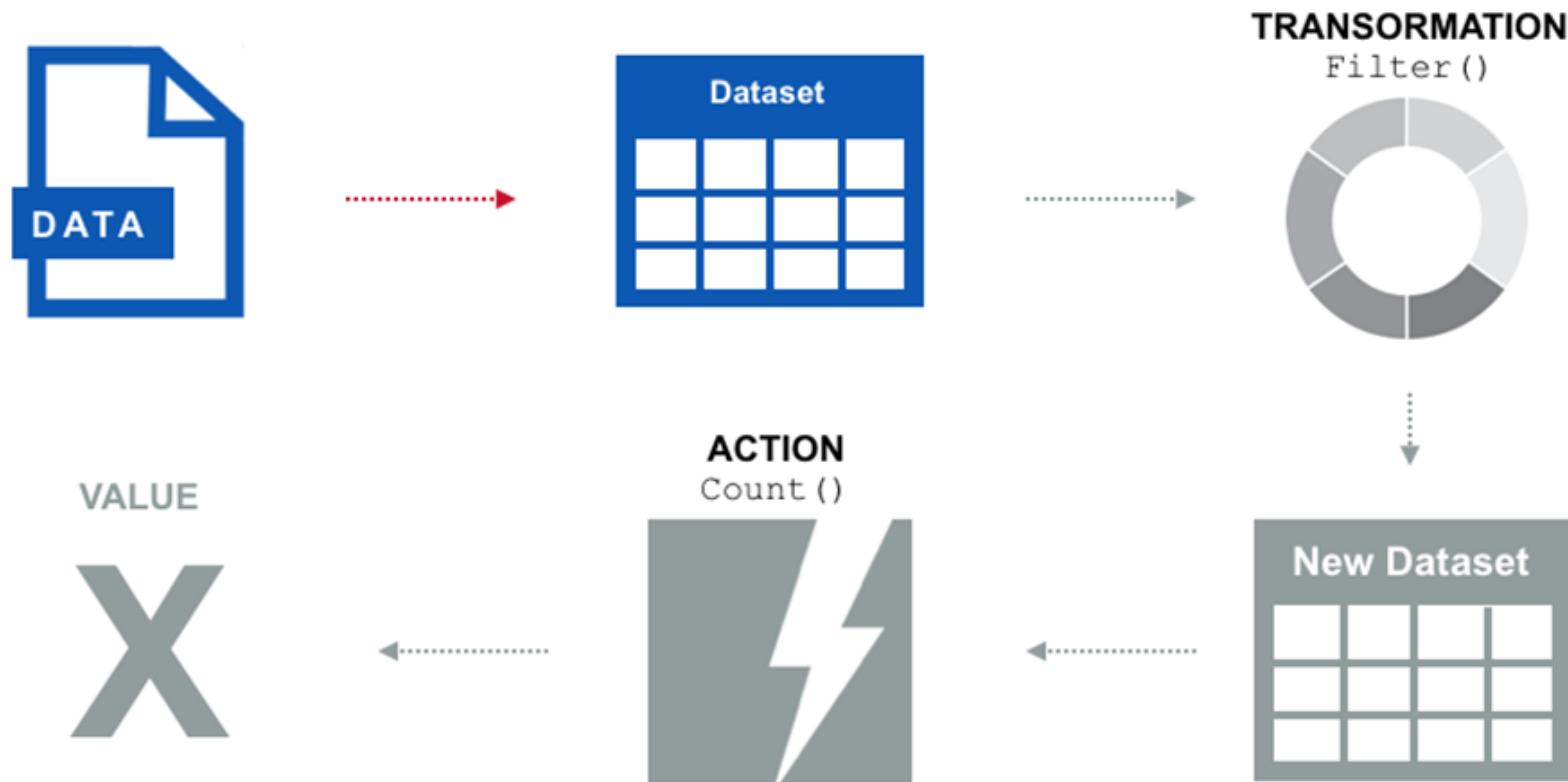
Commonly Used Actions

| Action | Definition |
|------------------------------|--|
| <code>count()</code> | Returns the number of elements in the Dataset |
| <code>reduce(func)</code> | Aggregate elements of Dataset using function <code>func</code> |
| <code>collect()</code> | Returns all elements of Dataset as an array to driver program |
| <code>take(n)</code> | Returns first n elements of Dataset |
| <code>show()</code> | Displays the first 20 rows of DataFrame in tabular form |
| <code>first(); head()</code> | Returns the first row of the Dataset |
| <code>takeAsList(n)</code> | Return first n elements of Dataset as list |

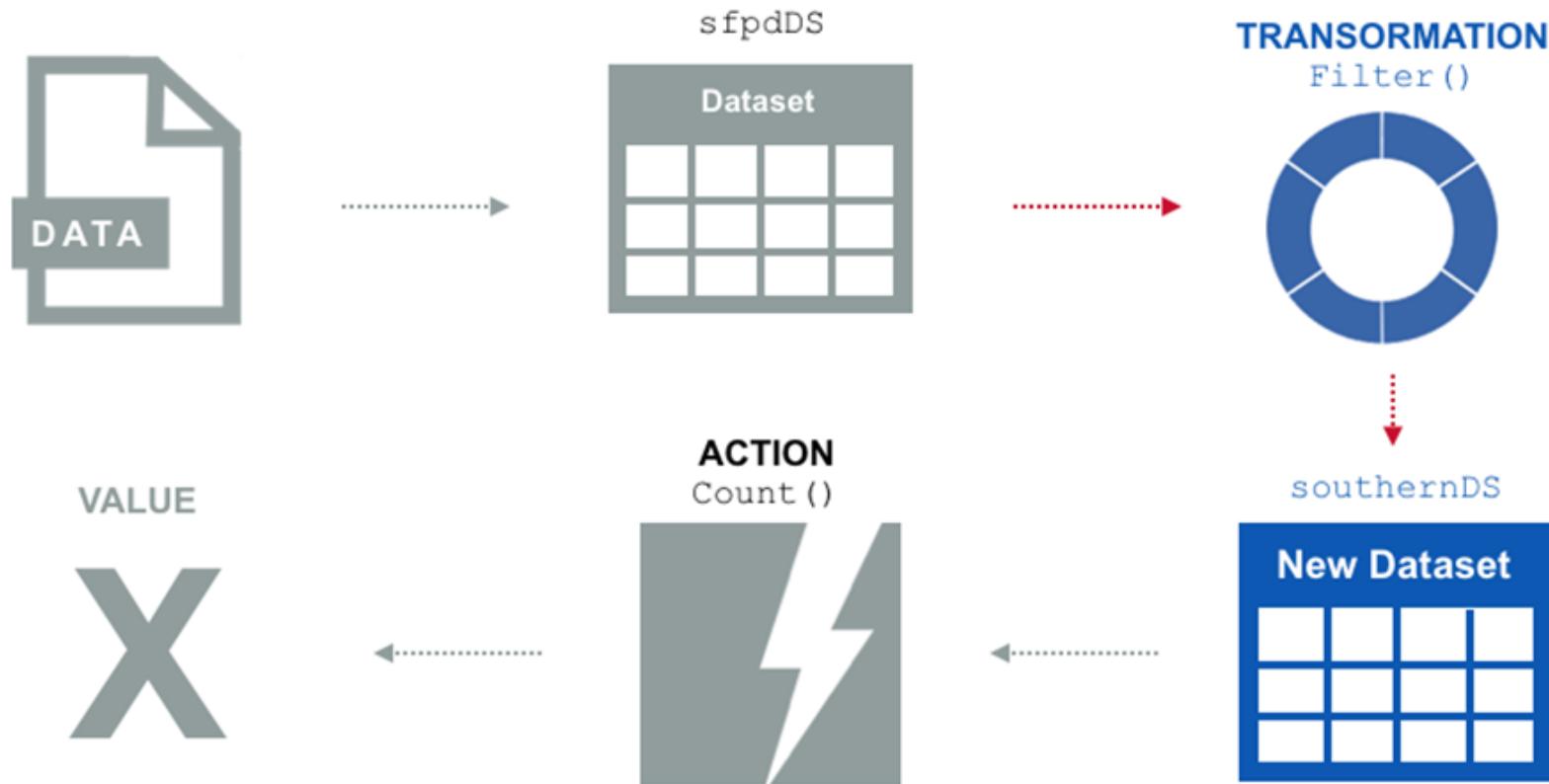
Applying Transformations and Actions

1. Define Dataset
2. Define Transformation
3. Apply an Action
4. Return Value

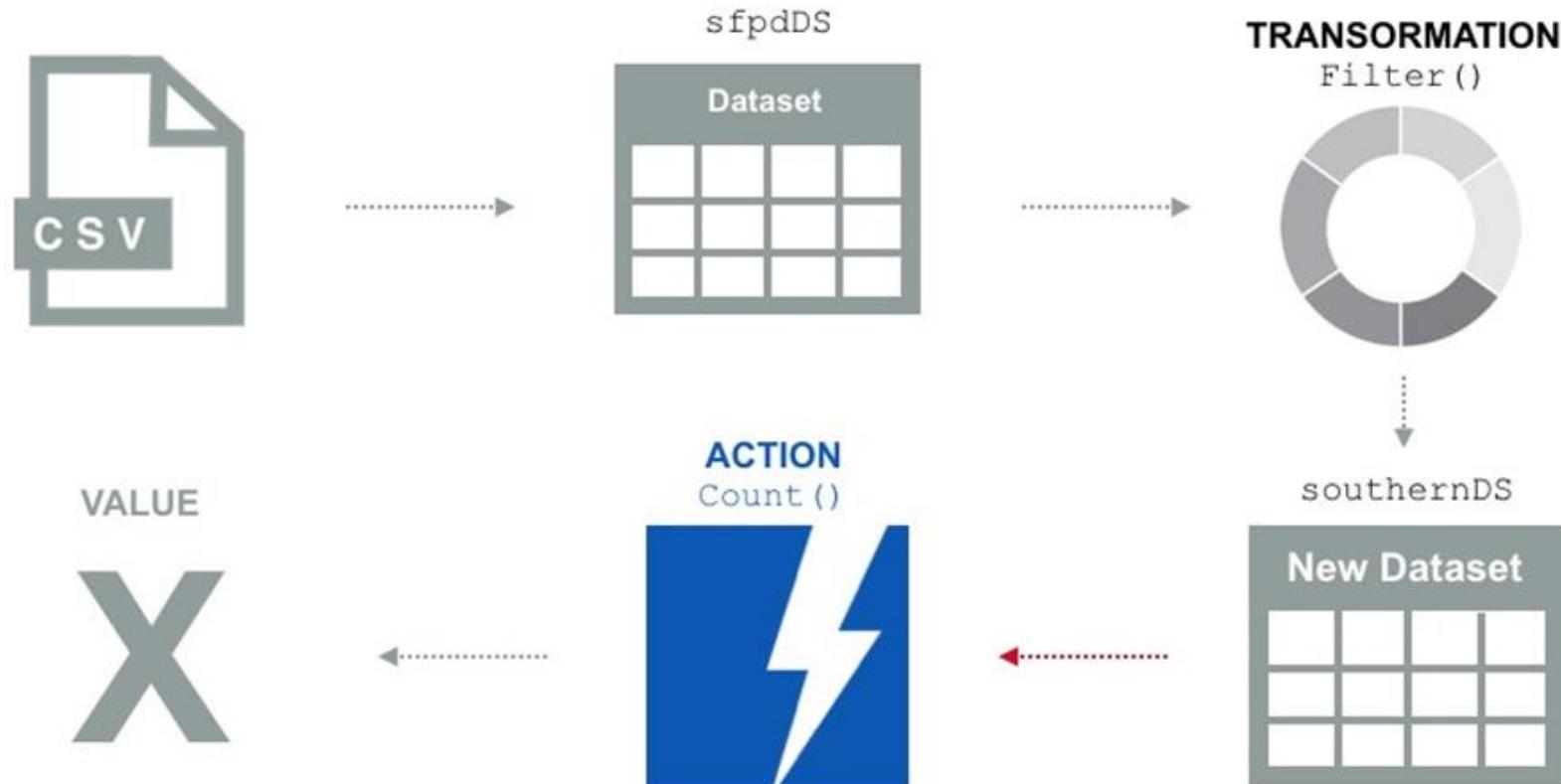
Step 1: Define Dataset



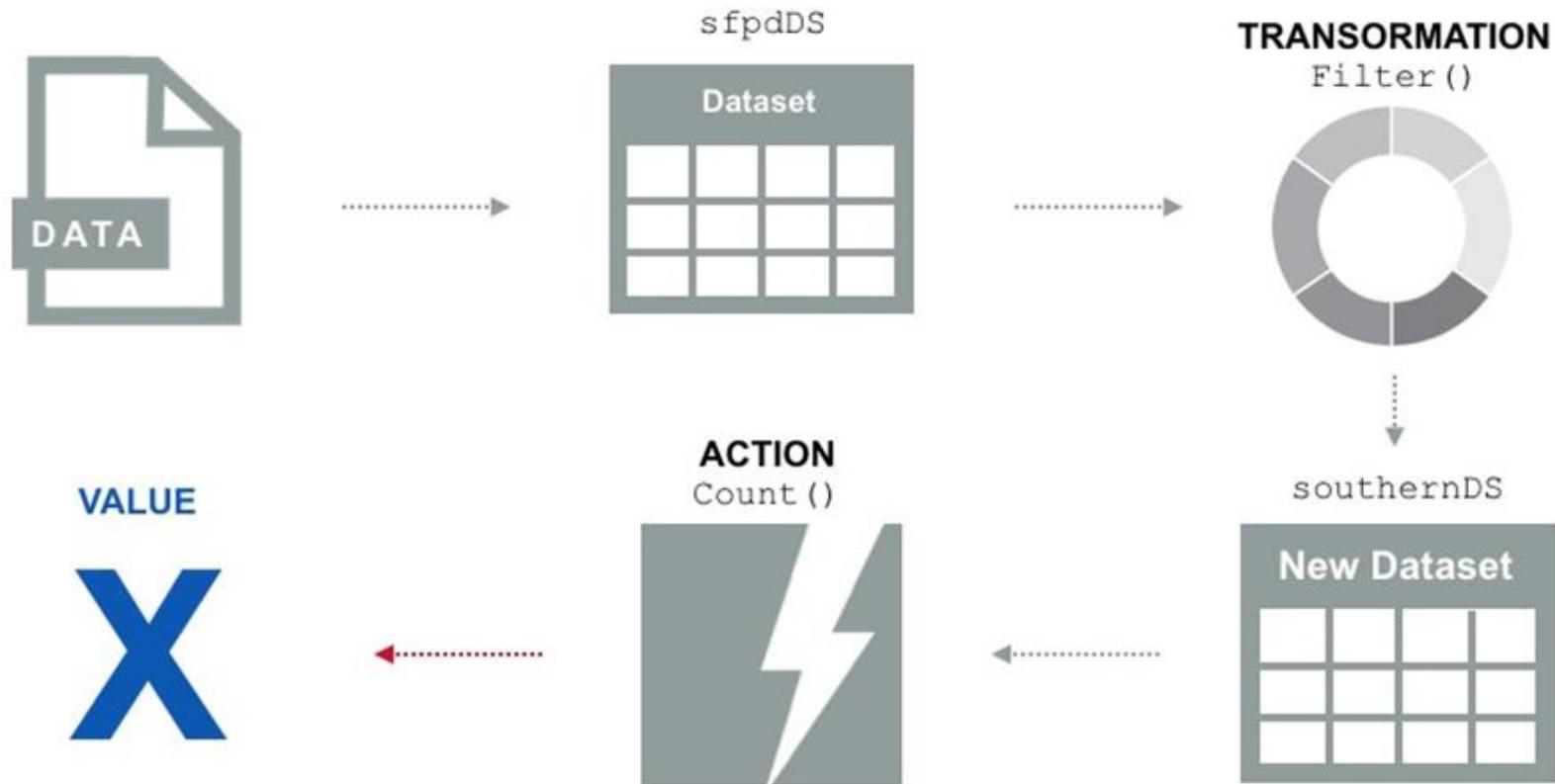
Step 2: Define Transformation



Step 3: Apply an Action



Step 4: Resulting Value





DEFINE DATASET

```
val sfpdDS = spark.read.option("inferSchema",  
true).csv("/spark/data/sfpd.csv").toDF("incidentnum",  
"category", "description", "dayofweek", "date", "time",  
"pddistrict", "resolution", "address", "X", "Y",  
"pdid").as[Incidents]
```

DEFINE TRANSFORMATIONS

```
val districtDS = sfpdDS.filter("pddistrict = 'SOUTHERN'")
```

RUN ACTION

```
val southernDS = districtDS.count()
```

Actions on Dataset

Once the action has run and the value returned, the data is no longer in memory.



Knowledge Check



Match the Dataset operation to the result:

Result

- A. Number of distinct categories
- B. Number of incidents per district
- C. First 20 rows in Dataset

Dataset Operation

1. `incidentsDS.groupBy("pdDistrict").count()`
2. `sfpdDS.show()`
3. `sfpdDS.select("category").distinct().count()`

Top Five Addresses with Most Incidents: Scala

1. Create a Dataset: Group incidents by address
2. Count the number of incidents for each address
3. Sort the result of the previous step in descending order
4. Show the first five which is the top five addresses with the most incidents

Top Five Addresses with Most Incidents: Scala

```
val incByAddDS=sfpdDS.groupBy("address")
val numAddDS=incByAdd.count
val numAddDesc=numAdd.sort($"count".desc)
val top5Add=numAddDesc.show(5)
```

Top Five Addresses with Most Incidents: Scala

```
val incByAdd = sfpdDS.groupBy("address")
    .count
    .sort($"count".desc)
    .show(5)
```

Top Five Addresses with Most Incidents: SQL

```
val top5Addresses = spark.sql("SELECT address, count(incidentnum) AS  
inccount FROM sfpd GROUP BY address ORDER BY inccount DESC LIMIT 5") .show
```

Top Five Addresses with Most Incidents: Results

Scala

| address | count |
|----------------------|-------|
| 800_Block_of_BRYA... | 10852 |
| 800_Block_of_MARK... | 3671 |
| 1000_Block_of_POT... | 2027 |
| 2000_Block_of_MIS... | 1585 |
| 16TH_ST/MISSION_ST | 1512 |

SQL

| address | inccount |
|----------------------|----------|
| 800_Block_of_BRYA... | 10852 |
| 800_Block_of_MARK... | 3671 |
| 1000_Block_of_POT... | 2027 |
| 2000_Block_of_MIS... | 1585 |
| 16TH_ST/MISSION_ST | 1512 |

Knowledge Check



Identify whether each statement describes a Transformation, or an Action:



Action

OR



Transformation

- A. Returns a Dataset
- B. Returns a value
- C. Computed lazily
- D. Examples include count, take
- E. Examples include filter, map

Class Discussion



```
val incByAddDS = sfpdDF.groupBy("address").count  
.sort($"count".desc).show(5)
```

- What if you want the top 10?
- What if you want category information?

Output Operations: `save()`

| Operation | Description |
|---|---|
| <code>save (source, mode, options)</code> | Saves contents of Dataset based on given data sources, savemode, and set of options |
| <code>jdbc (url, name, overwrite)</code> | Saves contents of Dataset to JDBC at URL under table name table |
| <code>parquet (path)</code> | Saves contents of Dataset as parquet file |
| <code>saveAsTable (tablename, source, mode, options)</code> | Creates a table from contents of Dataset using data source, options, and mode |

Output Operations

To save contents of top5Addresses Dataset:

```
top5Addresses.write.format("json").mode("overwrite").save("/user/  
user01/test")
```

Knowledge Check



Which of the following will give the top 10 resolutions to the console, assuming that sfpdDS is the Dataset registered as a view named sfpd?

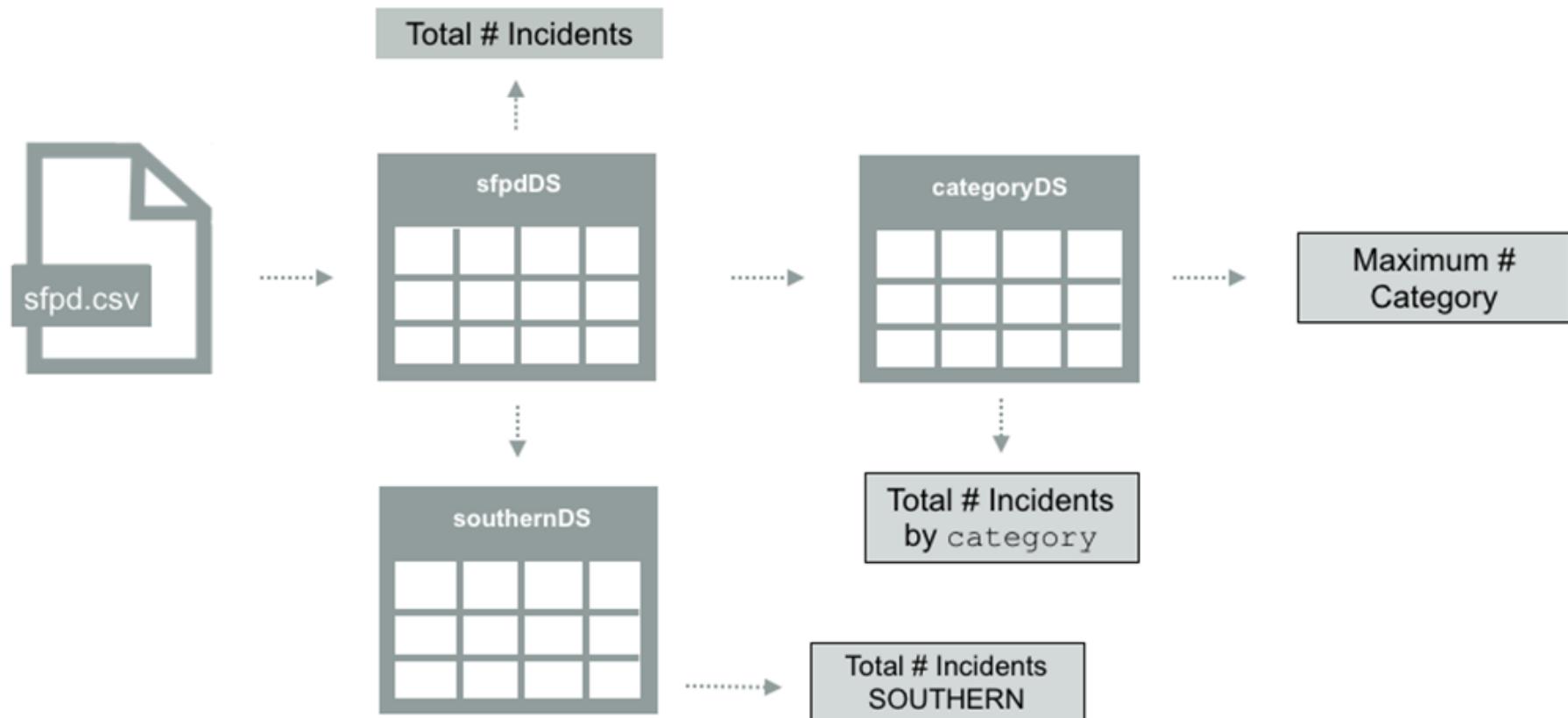
- A. `spark.sql("SELECT resolution, count(incidentnum) AS inccount FROM sfpd GROUP BY resolution ORDER BY inccount DESC LIMIT 10")`
- B. `sfpdDS.select("resolution").count.sort($"count".desc).show(10)`
- C. `sfpdDS.groupBy("resolution").count.sort($"count".desc).show(10)`

Lab 3.1: Explore and Save SFPD Data

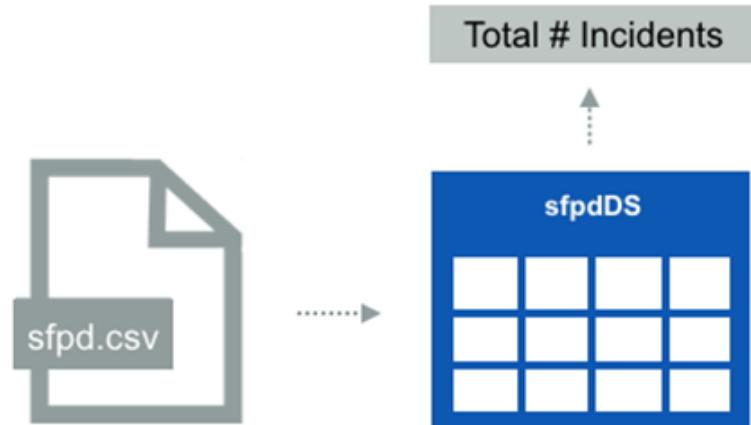


- Estimated time to complete: **20 minutes**
- In this lab, you will use SQL and Dataset operations to explore the SFPD data that you loaded in Lesson 2.

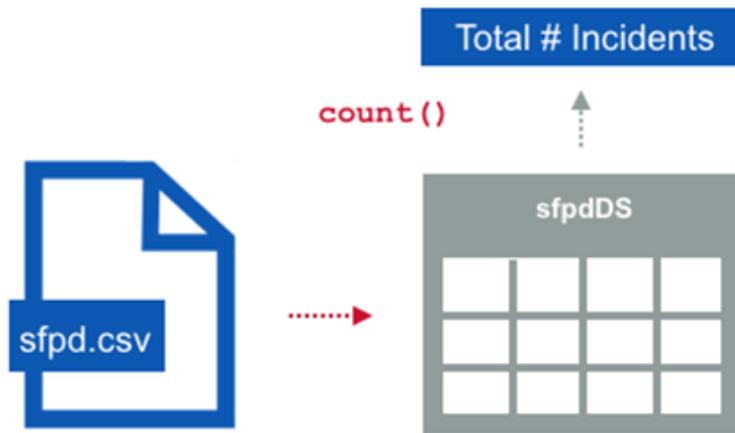
Lineage Graph



Lineage Graph: count()



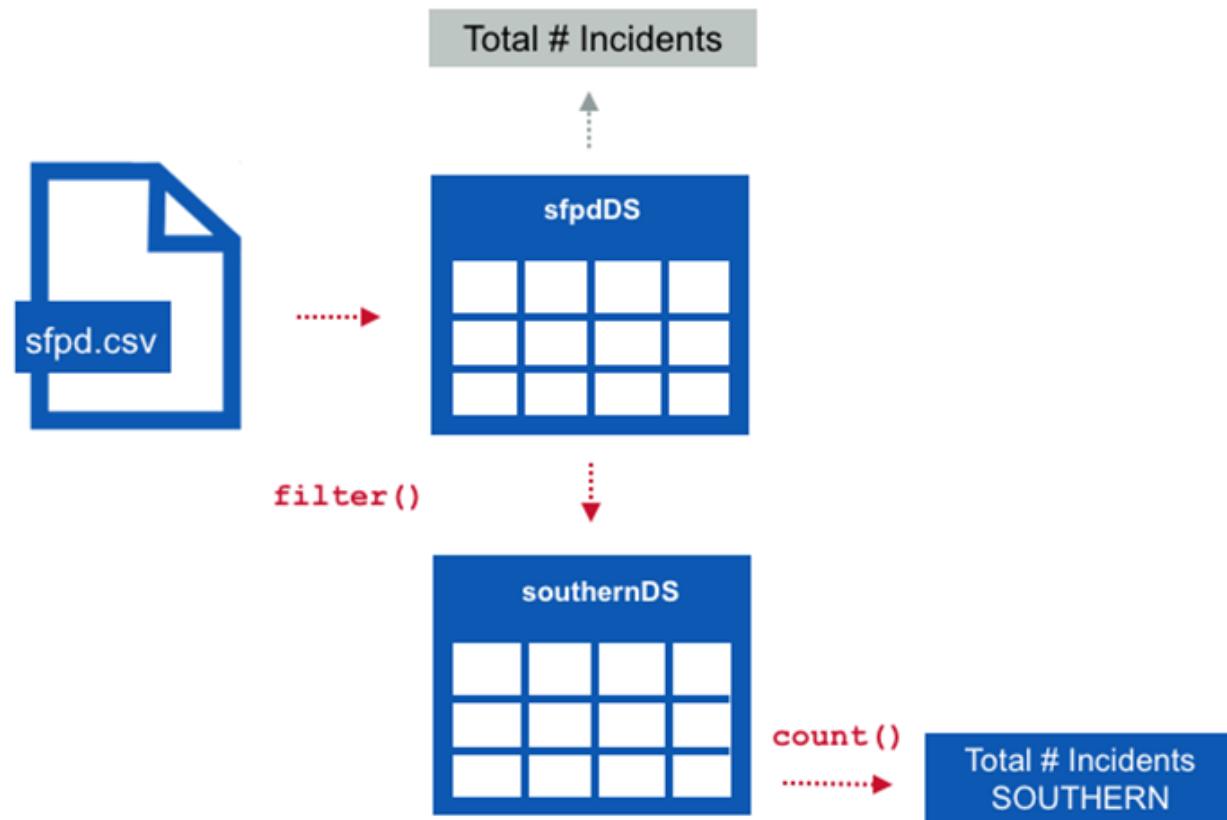
Lineage Graph: count()



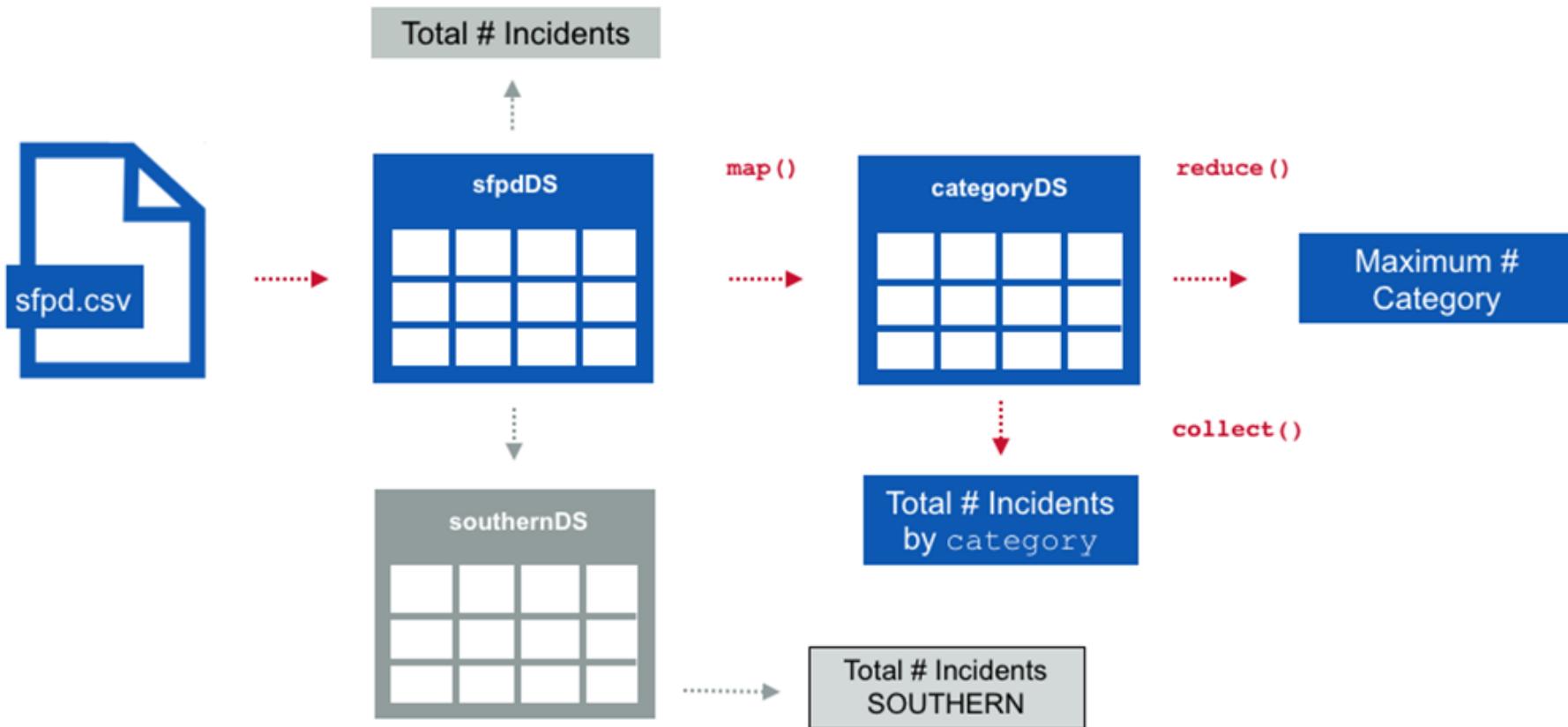
Lineage Graph: count()



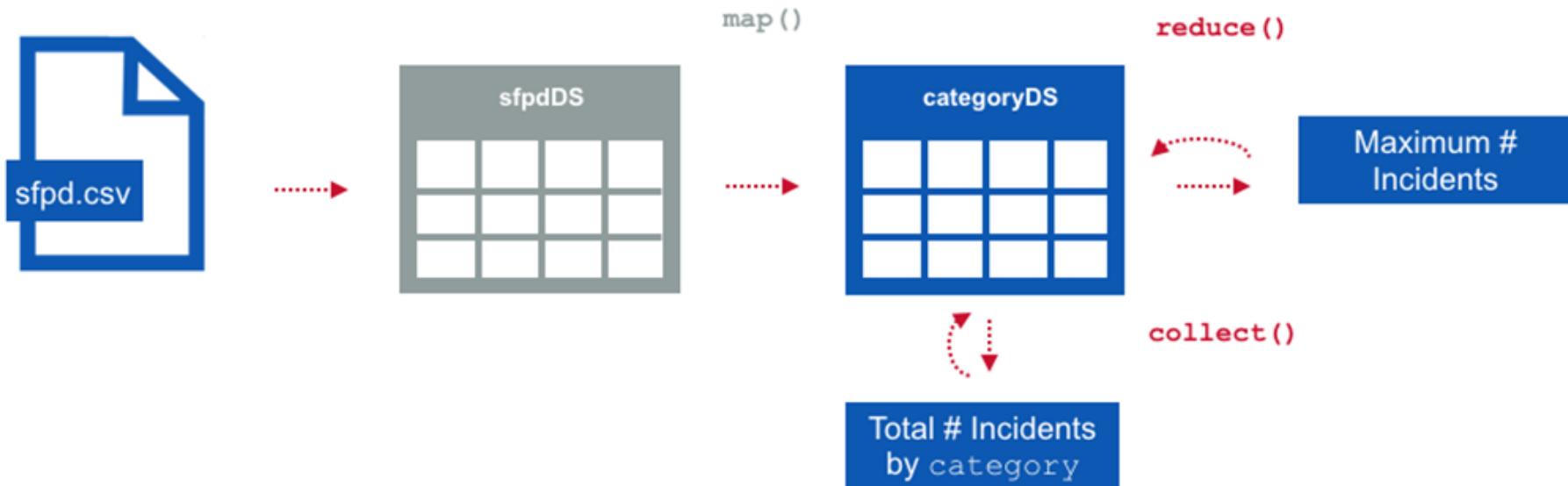
Lineage Graph: count()



Lineage Graph: `collect()` and `reduce()`



Why Cache Datasets



Caching a Dataset

1. Import Classes
2. Define DataFrame
3. Define Transformation
4. Cache Dataset
5. Apply Action
6. Subsequent Actions use Cached Data

Step 1: Import Classes

Import the necessary classes

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum", "category",
"description", "dayofweek", "date", "time", "pddistrict", "resolution",
"address", "X", "Y", "pdid")

val categoryDS = sfpdDF.groupBy("category")

categoryDS.cache

categoryDS.count.collect
categoryDS.count.collect
```

Step 2: Define DataFrame

Define instructions to create DataFrame

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum", "category",
"description", "dayofweek", "date", "time", "pddistrict", "resolution",
"address", "X", "Y", "pdid")

val categoryDS = sfpdDF.groupBy("category")

categoryDS.cache

categoryDS.count.collect
categoryDS.count.collect
```

Step 3: Define Transformation

Apply a transformation to create categoryDS

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum", "category",
"description", "dayofweek", "date", "time", "pddistrict", "resolution",
"address", "X", "Y", "pdid")

val categoryDS = sfpdDF.groupBy("category")

categoryDS.cache

categoryDS.count.collect
categoryDS.count.collect
```

Step 4: Cache Dataset

Cache the contents of categoryDS

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum", "category",
"description", "dayofweek", "date", "time", "pddistrict", "resolution",
"address", "X", "Y", "pdid")

val categoryDS = sfpdDF.groupBy("category")

categoryDS.cache

categoryDS.count.collect
categoryDS.count.collect
```

Step 5: Apply Action

Apply action that causes the categoryDS to be cached

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum", "category",
"description", "dayofweek", "date", "time", "pddistrict", "resolution",
"address", "X", "Y", "pdid")

val categoryDS = sfpdDF.groupBy("category")

categoryDS.cache

categoryDS.count.collect
categoryDS.count.collect
```

Step 6: Subsequent Actions use Cached Data

The next action on categoryDS will use the cached data

```
import spark.implicits._

val sfpdDF = spark.read.format("csv").option("inferSchema",
true).load("/spark/lab4/sfpd.csv").toDF("incidentnum", "category",
"description", "dayofweek", "date", "time", "pddistrict", "resolution",
"address", "X", "Y", "pdid")

val categoryDS = sfpdDF.groupBy("category")

categoryDS.cache

categoryDS.count.collect
categoryDS.count.collect
```

Best Practices

Release unneeded cached Datasets:

```
unpersist()
```

If not enough memory is available, application may crash

Knowledge Check



When considering the caching of Datasets, which of the following are true?

- A. When there is branching in lineage, it is advisable to cache the Dataset
- B. Use `dataset.cache()` to cache the Dataset
- C. Cache behavior depends on available memory. If not enough memory, then action will reload from file instead of from cache
- D. `dataset.persist(MEMORY_ONLY)` is the same as `dataset.cache()`
- E. All of the above



Learning Goals

3.1 Apply Operations on Datasets

3.2 Cache Datasets

3.3 Create User Defined Functions

3.4 Repartition Datasets

User Defined Functions (UDFs)

- Write your own functions
- In Spark, can define UDFs inline
- No complicated registration or packaging process

UDF Types

- Two types of UDFs:
 - Used with Scala (Dataset operations)
 - Used with SQL

Scala

SQL

Scenario: Find Incidents by Year

- Date in the format: “dd/mm/yy”
- Need to extract string after last slash
- Can then compute incidents by year

| Incident Num | Category | Descript | Day OfWeek | Date | Time |
|--------------|----------------|--------------------------------------|------------|--------|-------|
| 150599321 | OTHER_OFFENSES | POSSESSION_OF_BURGLARY_TOOLS | Thursday | 7/9/15 | 23:45 |
| 156168837 | LARCENY/THEFT | PETTY_THEFT_OF_PROPERTY | Thursday | 7/9/15 | 23:45 |
| 150599224 | OTHER_OFFENSES | DRIVERS_LICENSE/SUSPENDED_OR_REVOKED | Thursday | 7/9/15 | 23:36 |
| 150599230 | VANDALISM | MALICIOUS_MISCHIEF/BREAKING_WINDOW | Thursday | 7/9/15 | 23:20 |

Scala: Define UDF

- Inline creation
 - Use `udf()`
- Used with Dataset operations

```
val getStr = udf( arguments )=>{ function definition }
```

Scala

Scala: Define Function

```
val getStr = udf((s:String)=>{
  val lastS = s.substring(s.lastIndexOf('/')+1)
  lastS
})
val yy = sfpdDS.groupBy(getStr(sfpdDS("date")))
  .count
  .show
```

Scala: Use UDF

```
val getStr = udf((s:String)=>{
  val lastS = s.substring(s.lastIndexOf('/')+1)
  lastS
})
val yy = sfpdDS.groupBy(getStr(sfpdDS("date")))
    .count
    .show
```

Scala Results: Find Incidents by Year

```
scalaUDF(date) count
13           152830
14           150185
15           80760
```

SQL: Define and Register UDF

- To register and create inline, use:

```
spark.udf.register("funcname", func definition)
```

- Use in SQL statements



SQL

SQL: Register UDF

```
spark.udf.register("getStr", (s:String)=>{
    val strAfter=s.substring(s.lastIndexOf('/')+1)
    strAfter
})
```

SQL: Define Function

```
spark.udf.register("getStr", (s:String)=>{
    val strAfter=s.substring(s.lastIndexOf('/')+1)
    strAfter
})
```

SQL: Use in SQL Statements

```
val numIncByYear = spark.sql("SELECT getStr(date),  
count(incidentnum) AS countbyyear  
FROM sfpd GROUP BY getStr(date)  
ORDER BY countbyyear DESC  
LIMIT 5")
```

SQL Results: Find Incidents by Year

```
numIncByYear.show
```

```
[13,152830]
```

```
[14,150185]
```

```
[15,80760]
```

Knowledge Check



To use a UDF in a SQL query, the function in Scala:

- A. Must be defined inline using `udf (<function definition>)`
- B. Must be registered using `spark.udf.register`
- C. Only needs to be defined as a function
- D. Only needs a definition

Knowledge Check



To use a UDF in a SQL query, the function in Scala:

- A. Must be defined inline using `udf(<function definition>)`
- B. Must be registered using `spark.udf.register`**
- C. Only needs to be defined as a function
- D. Only needs a definition

Lab 3.3: Create and Use User-Defined Functions (UDFs)



- Estimated time to complete: **20 minutes**
- In this lab, you will create a UDF that extracts the year from the data field in the SFPD data. You will then register and use the UDF to query SFPD data by year.



Learning Goals

3.1 Apply Operations on Datasets

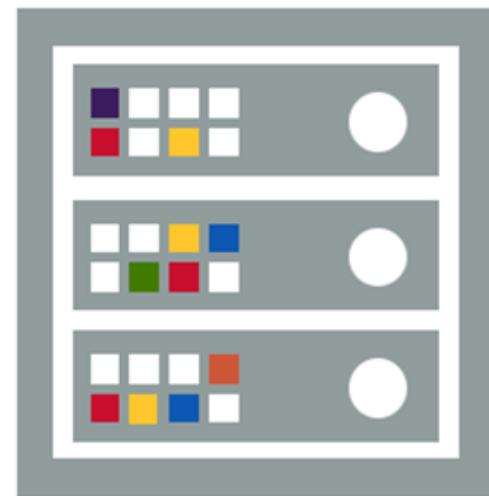
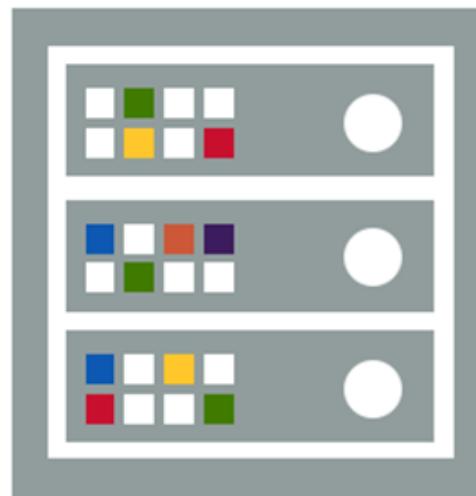
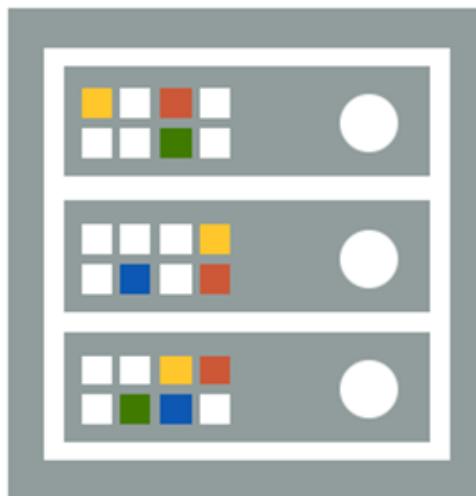
3.2 Cache Datasets

3.3 Create User Defined Functions

3.4 Repartition Datasets

Why Repartition?

- Spark partitions are basic units of parallelism, which help distribute data across the cluster
- Spark automatically partitions data, but this setting can be manually controlled when needed



Partition Datasets

- Sets number of partitions in Datasets after shuffle:

```
spark.sql.shuffle.partitions
```

- Default value set to 200

- Can change parameter using:

```
spark.conf.set
```

- Internally SparkSQL partitions data for joins and aggregations
- If applying other operations on result of Dataset operations, can manually control partitioning

Partition Dataset

Example – SFPD dataset with four partitions

| P1 | | | P2 | | | P3 | | | P4 | | |
|------------------|-------------------|---------------------|------------------|-------------------|---------------------|------------------|-------------------|---------------------|------------------|-------------------|---------------------|
| Incnum (Str) | Category (Str) | PdDistrict (Str) |
| 150598981 | ASSAULT | CENTRAL | 150599183 | ASSAULT | SOUTHERN | 150597701 | ASSAULT | MISSION | 150597400 | ROBBERY | TARAVAL |
| 150599161 | BURGLARY | PARK | 150599246 | ASSAULT | CENTRAL | 150597701 | ROBBERY | INGLESIDE | 150596468 | FRAUD | SOUTHERN |
| 150599127 | SUSPICIOUS | SOUTHERN | 150599246 | WARRANTS | CENTRAL | 150597701 | ASSAULT | SOUTHERN | 150597234 | BURGLARY | SOUTHERN |
| 150603455 | VANDALISM | NORTHERN | 150599246 | WARRANTS | CENTRAL | 150597591 | ROBBERY | SOUTHERN | 150596468 | FRAUD | TARAVAL |

Partitioning

- To determine current number of partitions:

```
ds.rdd.partitions.size()
```

- To repartition, use

```
ds.repartition(numPartitions)
```

Knowledge Check



In Scala, to find the number of partitions in the Dataset, use:

- A. `ds.numPartitions()`
- B. `ds.rdd.partitions.size()`
- C. `df.partitions.size()`
- D. `df.partitionnumber()`

Lab 3.4: Analyze Data Using UDFs and Queries



- Estimated time to complete: **30 minutes**
- In this lab, you will build a standalone application that uses what you have learned so far. You will create a Dataset, create a UDF to extract the year 2015, query some SFPD data for that year, and save the results to a JSON file.

Lesson 4: Build a Simple Apache Spark Application.





Learning Goals

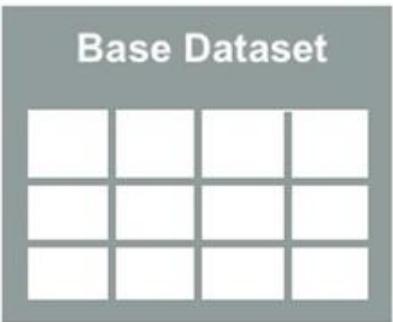
- 4.1 Define the Spark Program Lifecycle
- 4.2 Define the Function of SparkSession
- 4.3 Describe Ways to Launch Spark Applications
- 4.4 Launch a Spark Application

Review



1

Create input
Dataset in your
driver program



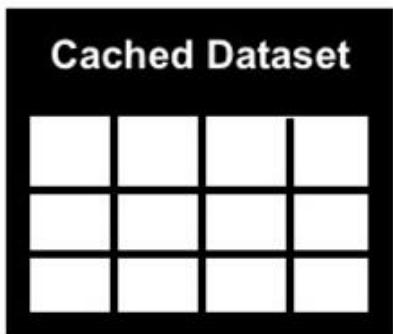
2

Use lazy
transformations
to define new
Datasets



3

Cache any
Datasets that
are reused

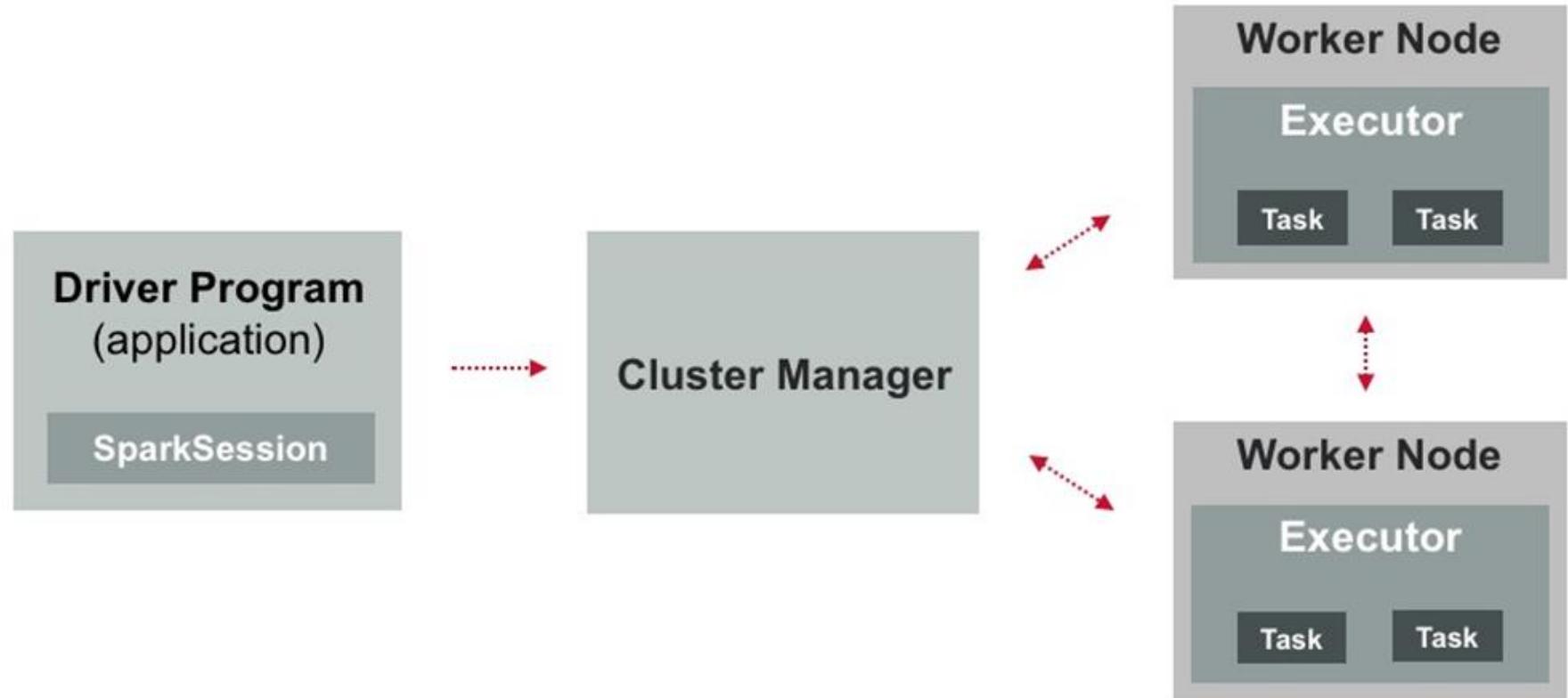


4

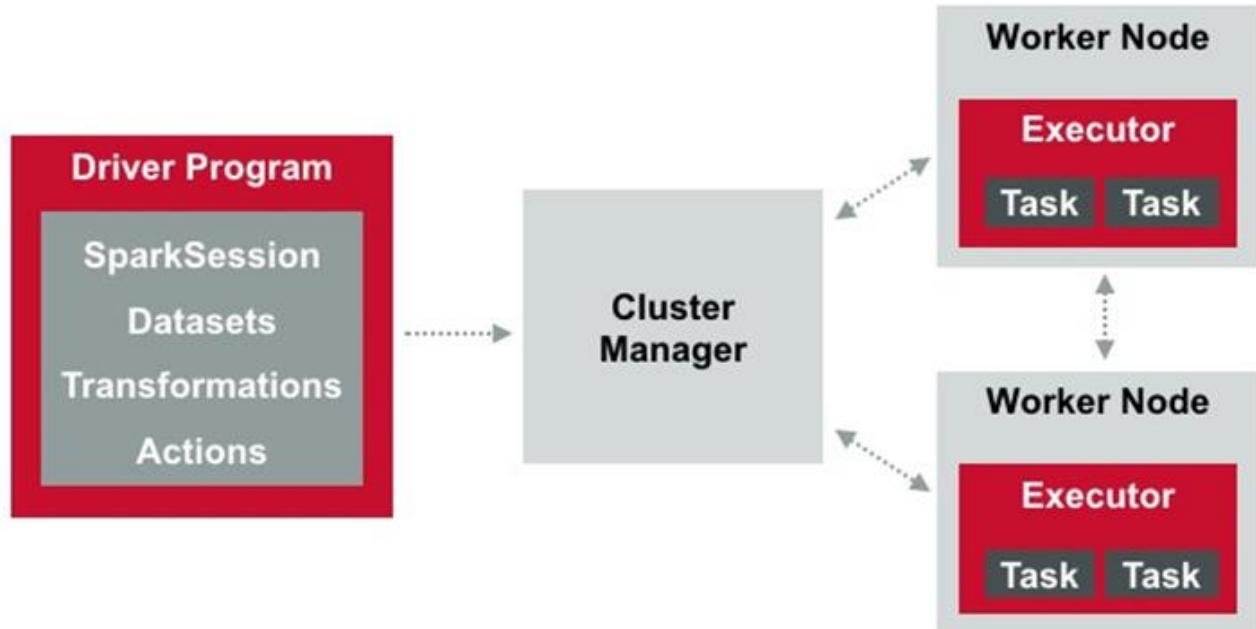
Kick off
computations
using actions



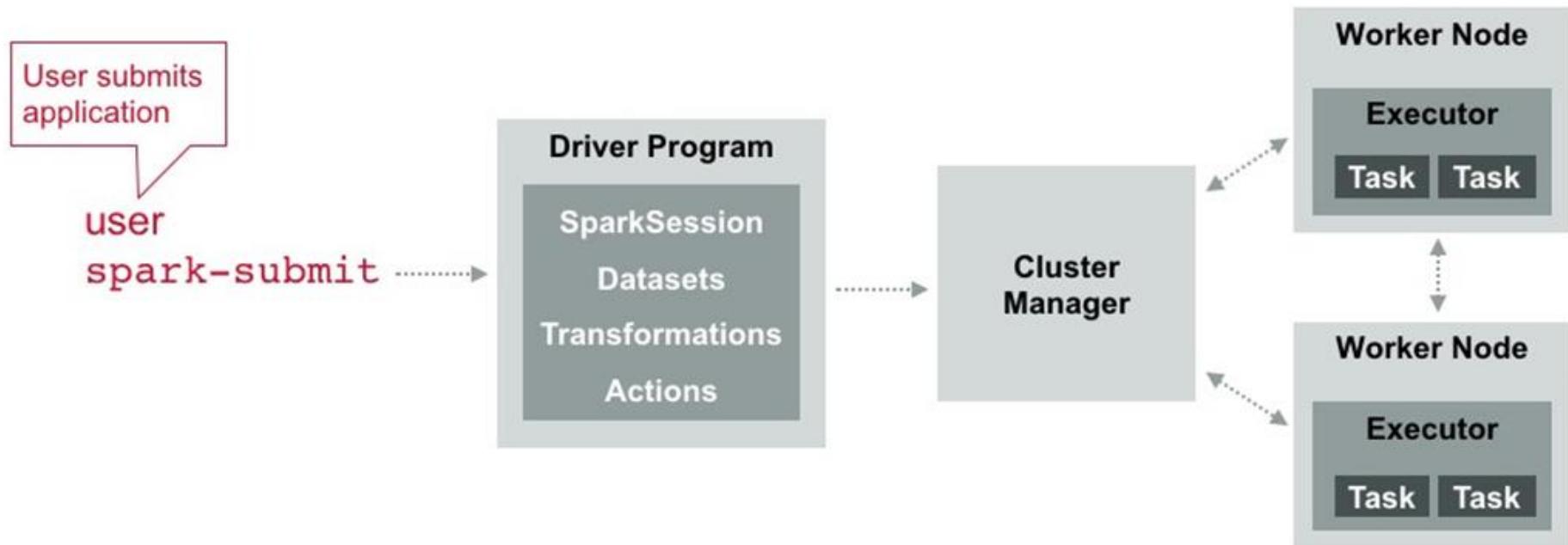
Review



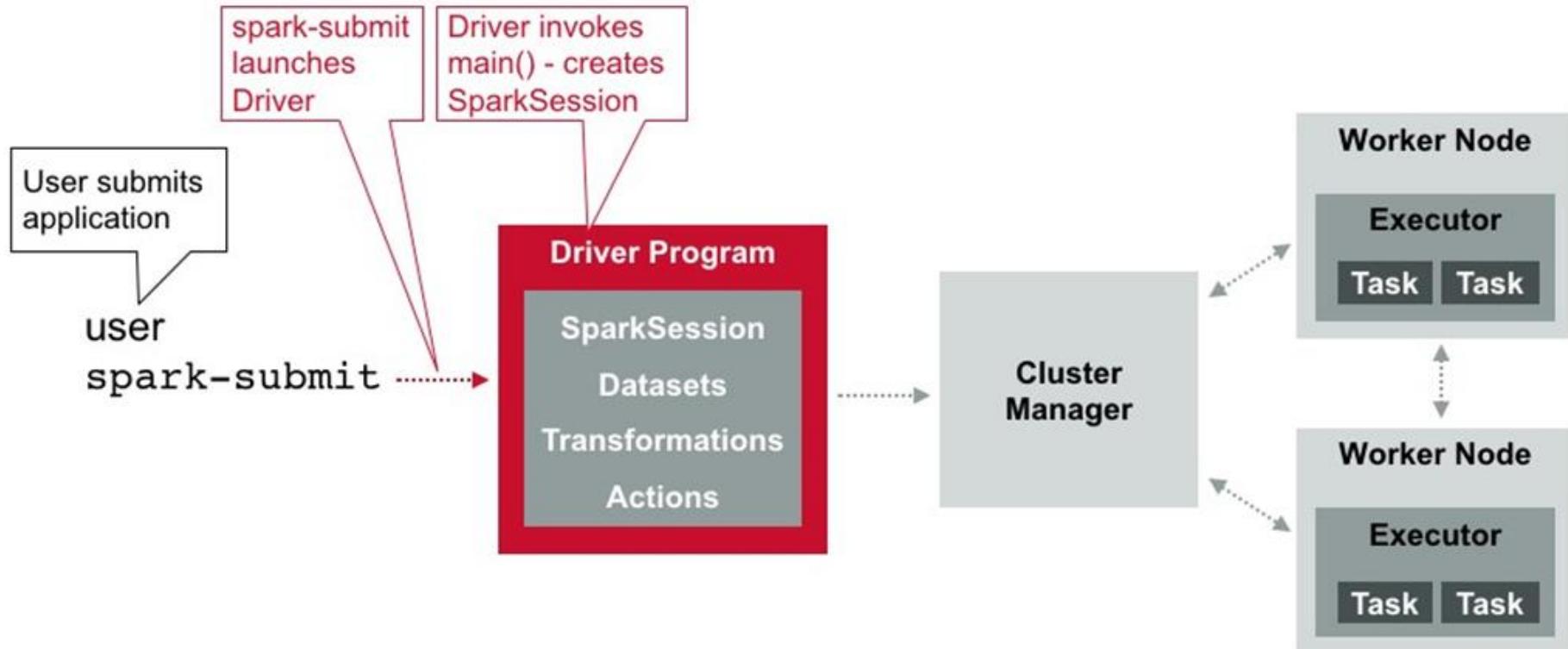
Components of Distributed Spark Application



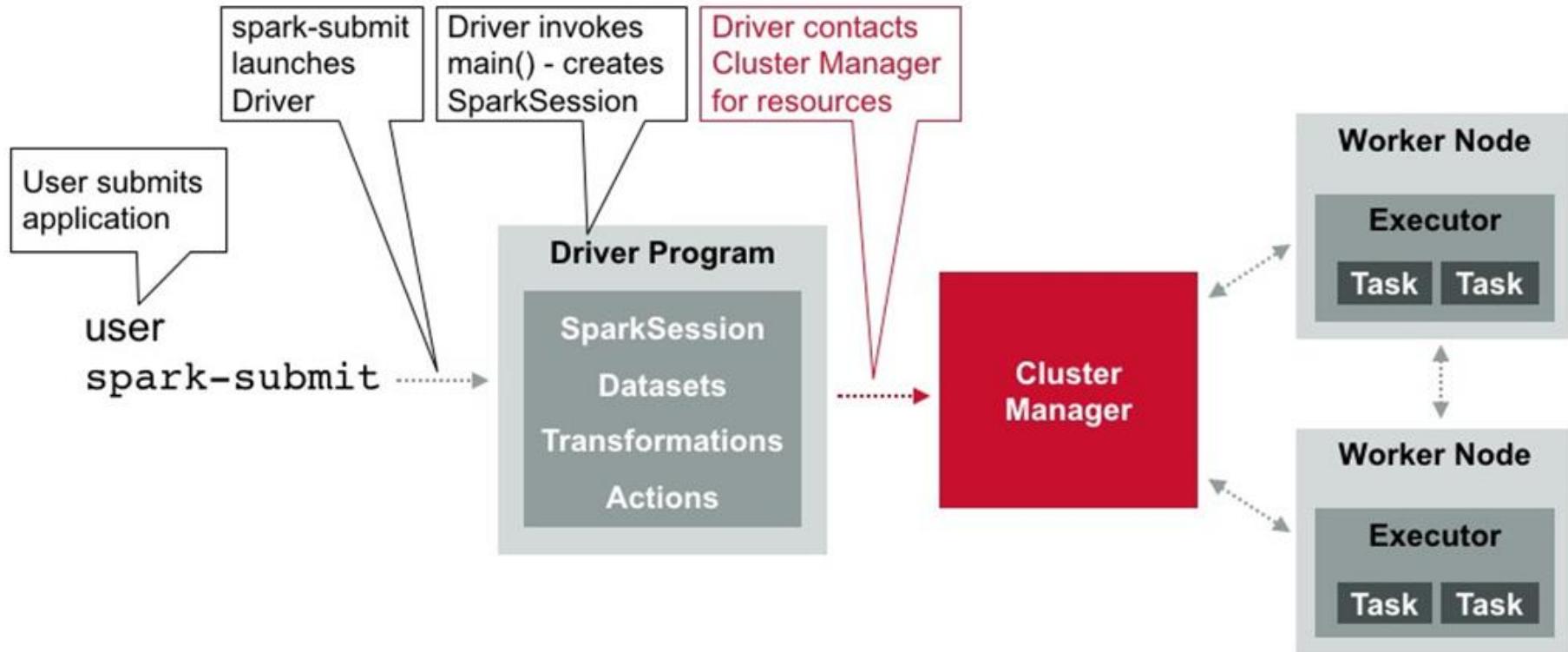
Components of Distributed Spark Application



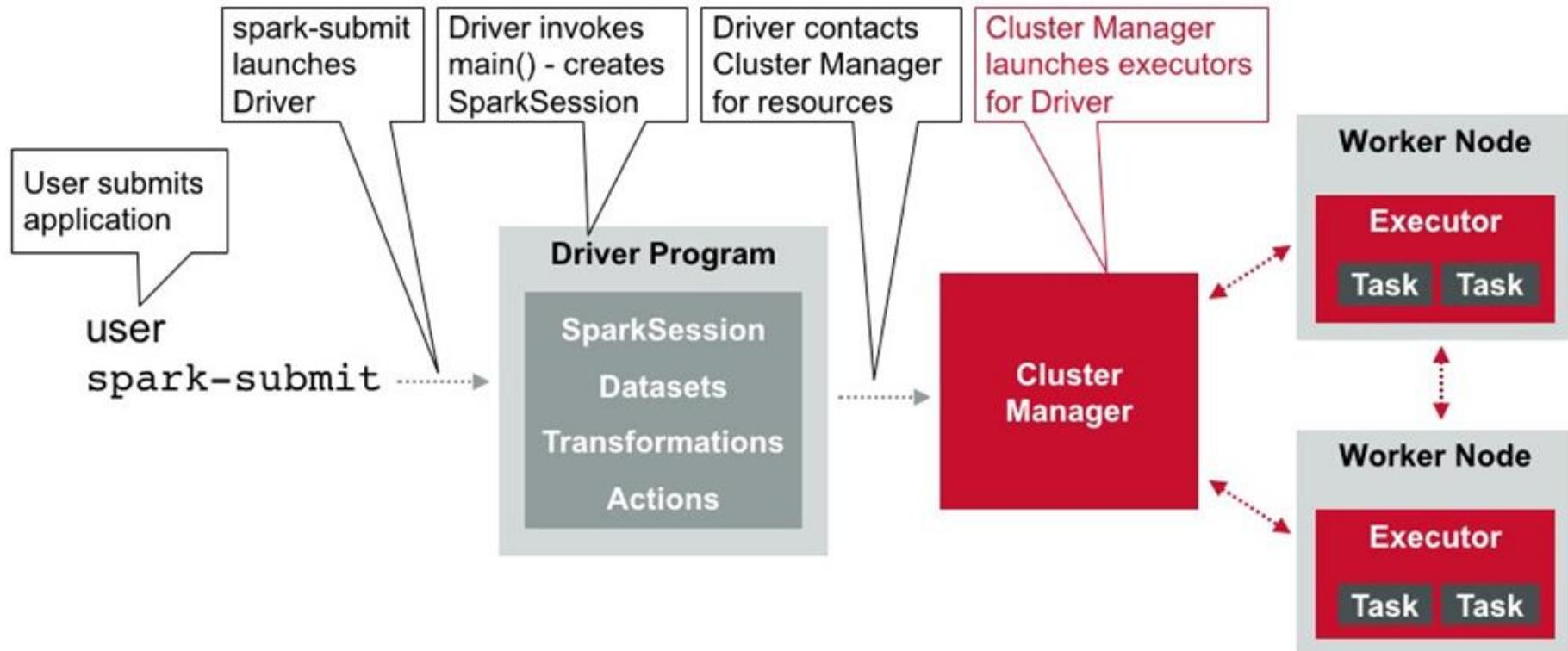
Components of Distributed Spark Application



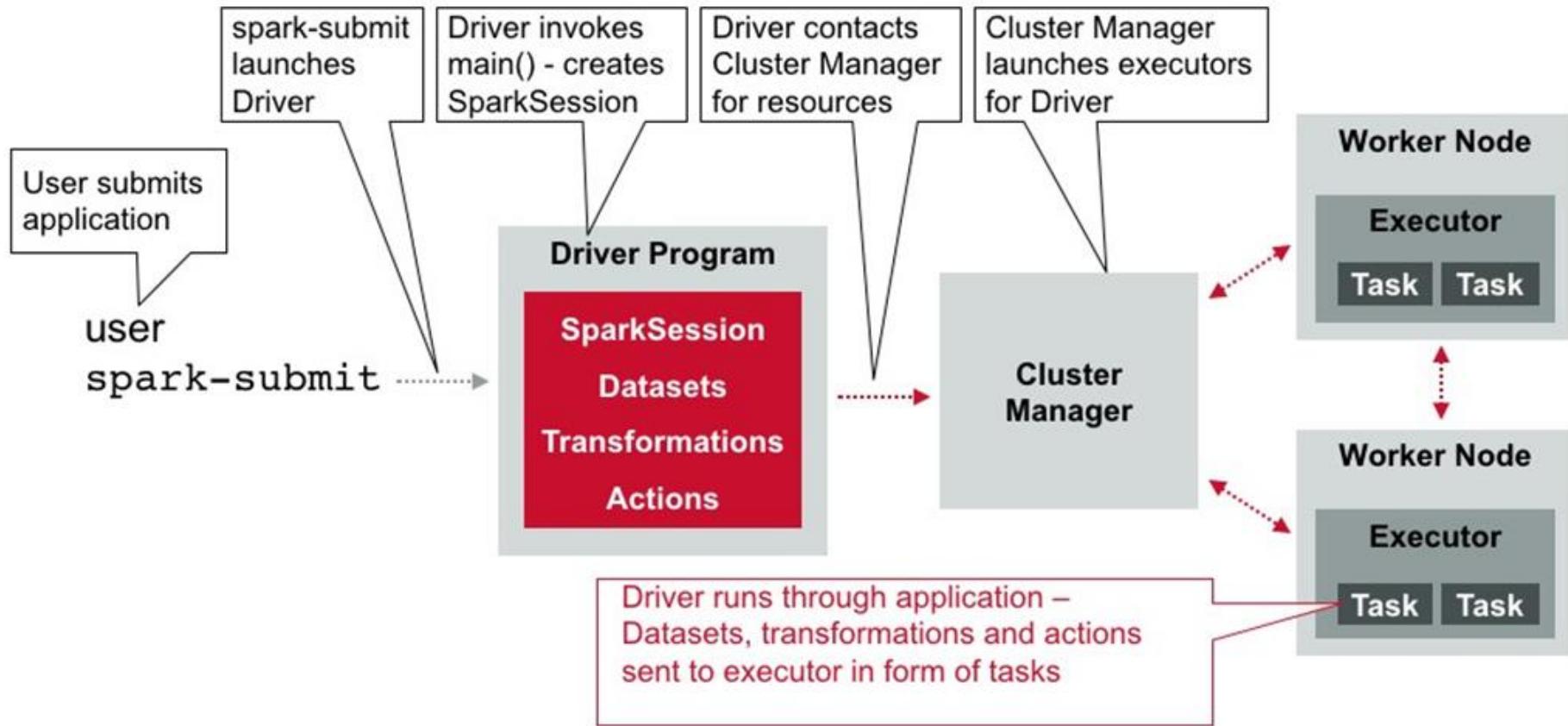
Components of Distributed Spark Application



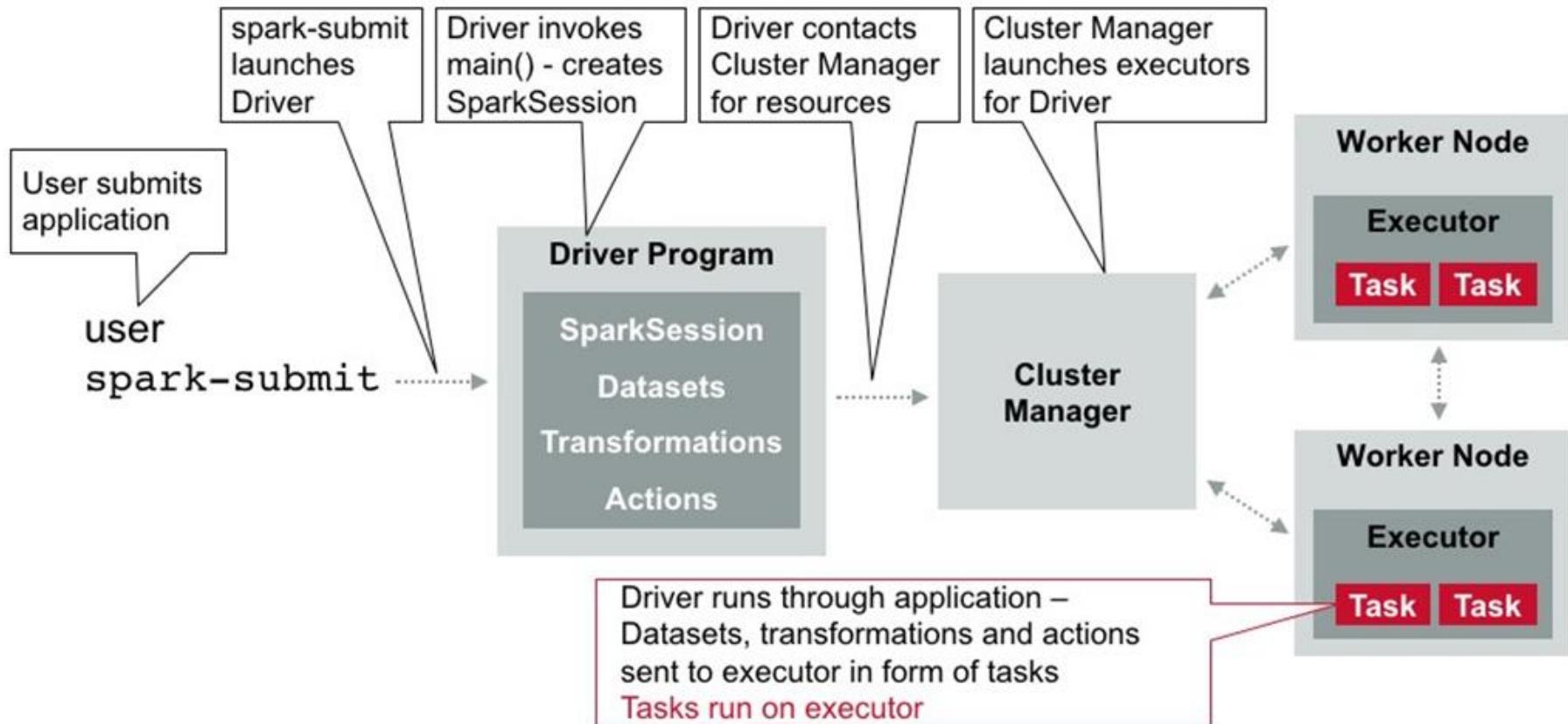
Components of Distributed Spark Application



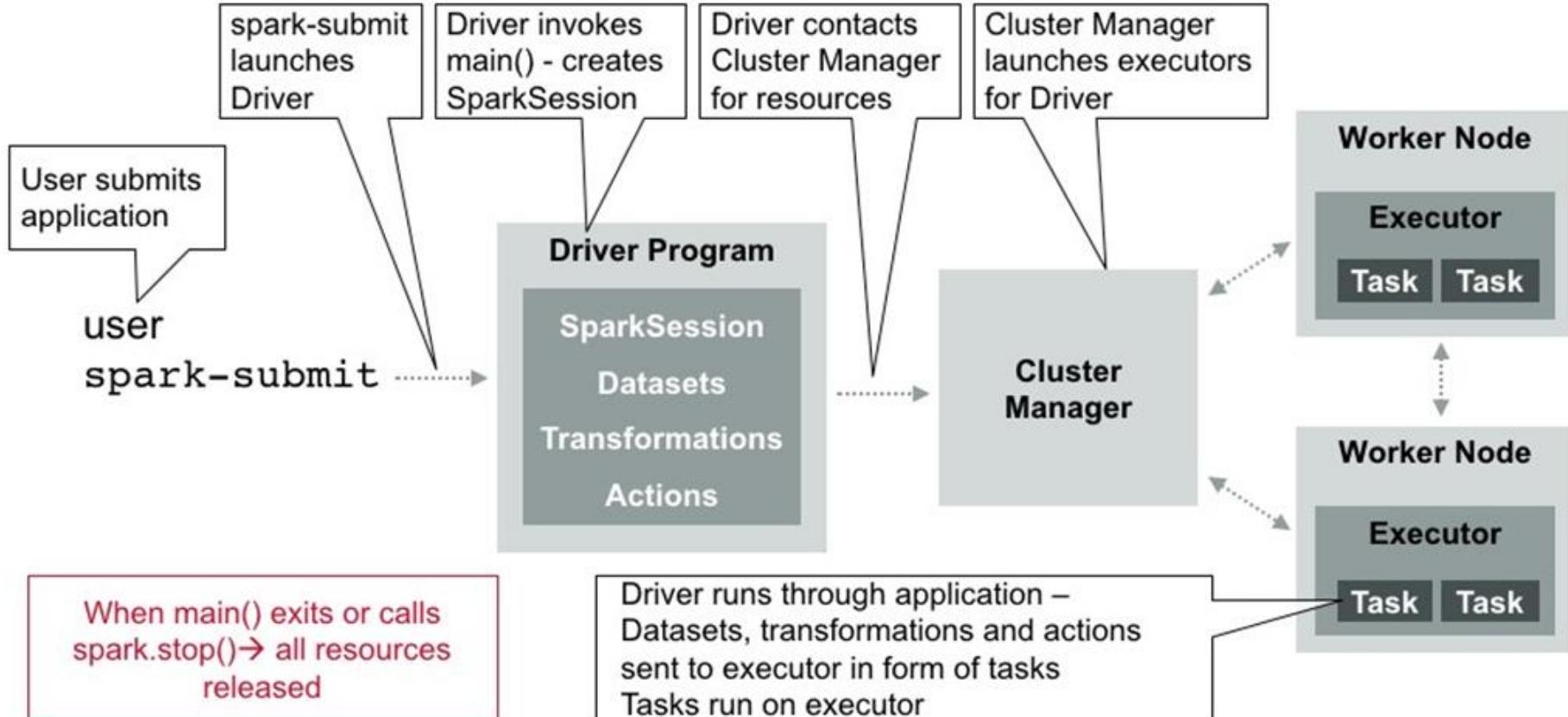
Components of Distributed Spark Application



Components of Distributed Spark Application



Components of Distributed Spark Application



Knowledge Check



Place the steps of a Spark application lifecycle in the right order

- A. val
burglaries=sfpdDS.filter(line=>line.contains("BURGLARIES"))
- B. val
sfpdDS=spark.read.textFile("/user/user01/data/sfpd.csv")
- C. val totburglaries=burglaries.count()
- D. burglaries.cache()

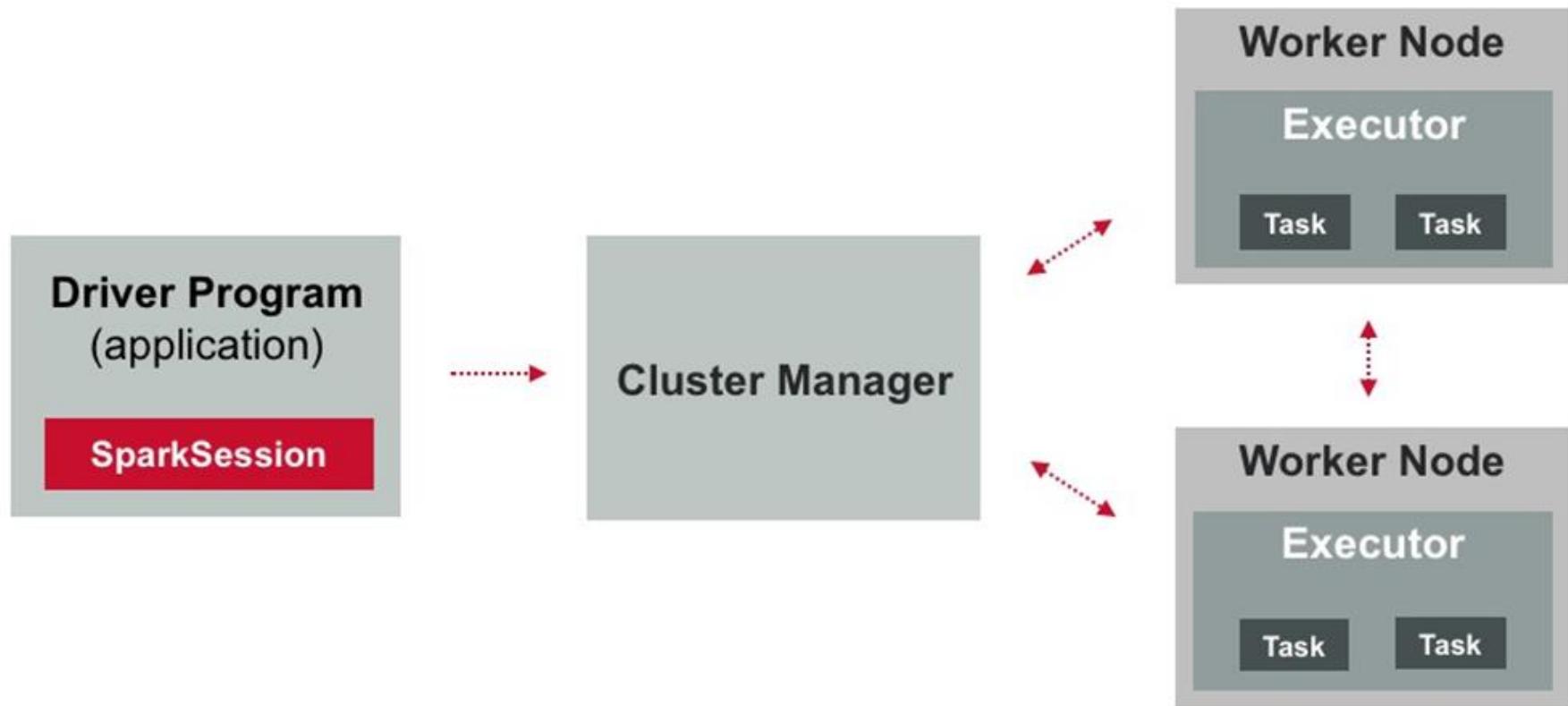
Knowledge Check



Place the steps of a Spark application lifecycle in the right order:

```
val  
sfpdDS=spark.read.textFile("/user/user01/data/sfpd.csv")  
  
val  
burglaries=sfpdDS.filter(line=>line.contains("BURGLARIES"  
))  
  
burglaries.cache()  
  
val totburglaries=burglaries.count()
```

Review



Build a Standalone Application

1. Import Class
2. Define Class
3. Create `SparkSession` Object
4. Import `spark.implicits` Using `SparkSession`
5. Declare New Variable and Assign Path to Data File
6. Create Dataset (using `read` method)

Import Class

```
/* sfpdApp.scala - Simple App to inspect sfpd data */
/* The following import statement imports SparkSession*/
import org.apache.spark.sql.SparkSession

object sfpdApp{
    def main(args:Array[String]){
        val spark =
SparkSession.builder.master("local").appName("sfpdApp").getOrCreate()
        import spark.implicits._

        /* Add location of input file */
        val sfpdFile ="/user/user01/data/sfpd.csv"
        //build the input Dataset
        val sfpdDF = spark.read.format("csv").option("inferSchema",true).
load(sfpdFile).toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid").cache()

        ....
    }
}
```

Define Class

```
/* sfpdApp.scala - Simple App to inspect sfpd data */
/* The following import statement imports SparkSession*/
import org.apache.spark.sql.SparkSession

object sfpdApp{
    def main(args:Array[String]){
        val spark =
SparkSession.builder.master("local").appName("sfpdApp").getOrCreate()
        import spark.implicits._

        /* Add location of input file */
        val sfpdFile ="/user/user01/data/sfpd.csv"
        //build the input Dataset
        val sfpdDF = spark.read.format("csv").option("inferSchema",true).
load(sfpdFile).toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid").cache()

        ....
    }
}
```

Creating SparkSession Object

```
/* sfpdApp.scala - Simple App to inspect sfpd data */
/* The following import statement imports SparkSession*/
import org.apache.spark.sql.SparkSession

object sfpdApp{
    def main(args:Array[String]){
        val spark =
SparkSession.builder.master("local").appName("sfpdApp").getOrCreate()
        import spark.implicits._

        /* Add location of input file */
        val sfpdFile ="/user/user01/data/sfpd.csv"
        //build the input Dataset
        val sfpdDF = spark.read.format("csv").option("inferSchema",true).
load(sfpdFile).toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid").cache()
        ....
    }
}
```

Import spark.implicits Using SparkSession

```
/* sfpdApp.scala - Simple App to inspect sfpd data */
/* The following import statement imports SparkSession*/
import org.apache.spark.sql.SparkSession

object sfpdApp{
    def main(args:Array[String]){
        val spark =
SparkSession.builder.master("local").appName("sfpdApp").getOrCreate()
        import spark.implicits._

        /* Add location of input file */
        val sfpdFile ="/user/user01/data/sfpd.csv"
        //build the input Dataset
        val sfpdDF = spark.read.format("csv").option("inferSchema",true).
load(sfpdFile).toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid").cache()

        ....
    }
}
```

Declare New Variable and Assign Path

```
/* sfpdApp.scala - Simple App to inspect sfpd data */
/* The following import statement imports SparkSession*/
import org.apache.spark.sql.SparkSession

object sfpdApp{
    def main(args:Array[String]){
        val spark =
SparkSession.builder.master("local").appName("sfpdApp").getOrCreate()
        import spark.implicits._

        /* Add location of input file */
        val sfpdFile ="/user/user01/data/sfpd.csv"
        //build the input Dataset
        val sfpdDF = spark.read.format("csv").option("inferSchema",true).
load(sfpdFile).toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid").cache()
        ....
    }
}
```

Create Dataset

```
/* sfpdApp.scala - Simple App to inspect sfpd data */
/* The following import statement imports SparkSession*/
import org.apache.spark.sql.SparkSession

object sfpdApp{
    def main(args:Array[String]){
        val spark =
SparkSession.builder.master("local").appName("sfpdApp").getOrCreate()
        import spark.implicits._

        /* Add location of input file */
        val sfpdFile ="/user/user01/data/sfpd.csv"
        //build the input Dataset
        val sfpdDF = spark.read.format("csv").option("inferSchema",true).
load(sfpdFile).toDF("incidentnum", "category", "description", "dayofweek", "date",
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid").cache()

        ....
    }
}
```

Knowledge Check



Which of the following statements about SparkSession are TRUE?

- A. SparkSession is the starting point of any Spark program
- B. In a standalone Spark application, you don't need to initialize SparkSession
- C. Interactive Shell (Scala and Python) initializes SparkSession

Lab 4.2: Import and Configure Application Files



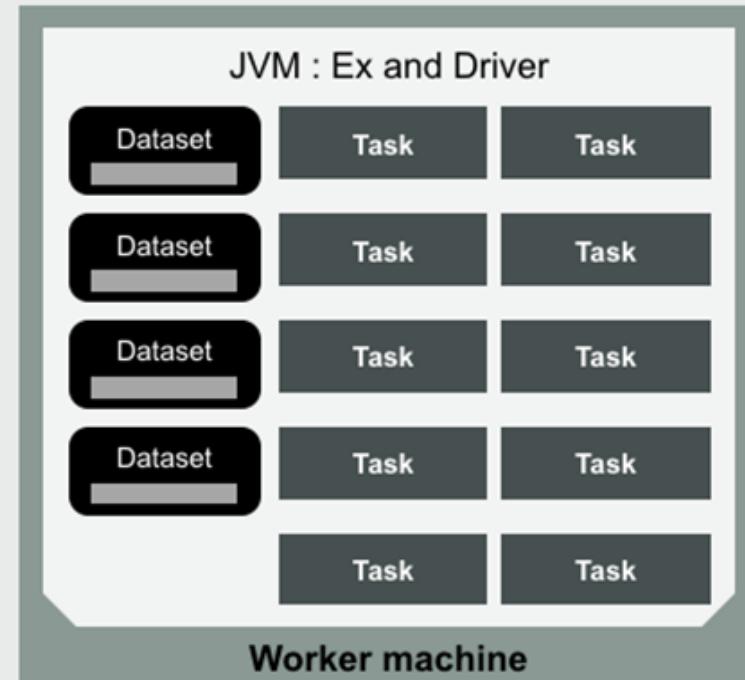
- Estimated completion time: **25 minutes**
- In this lab, you will download application files and import them into the IntelliJ IDE for use in a subsequent lab.

Ways to Launch a Spark Application

| Launch Mode | Runtime Environment |
|-----------------|------------------------------|
| 1. Local | Runs in the same JVM |
| 2. Standalone | Simple cluster manager |
| 3. Hadoop YARN | Resource manager in Hadoop 2 |
| 4. Apache Mesos | General cluster manager |

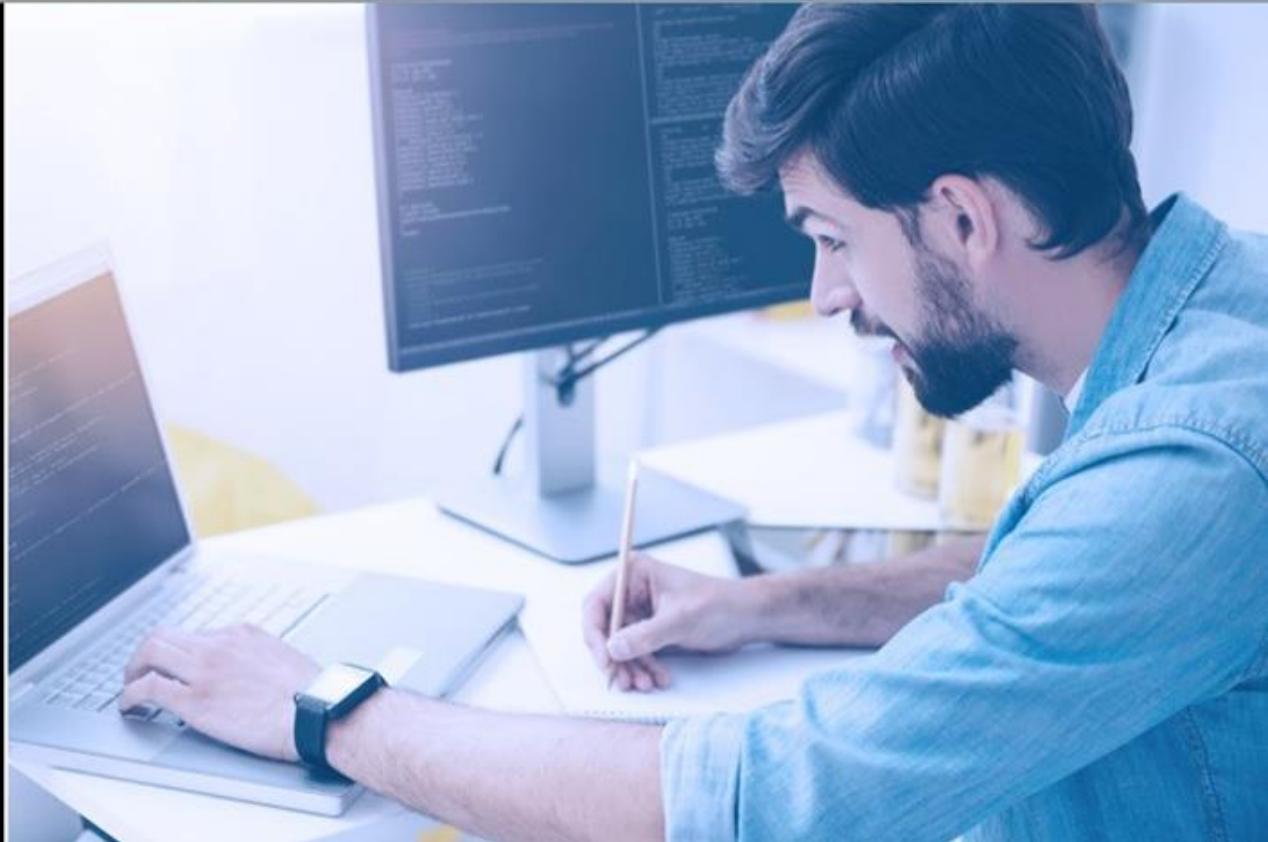
Mode 1: Local

- Driver program and workers in same JVM
- Datasets and variables in same memory space
- No central master
- Execution started by user



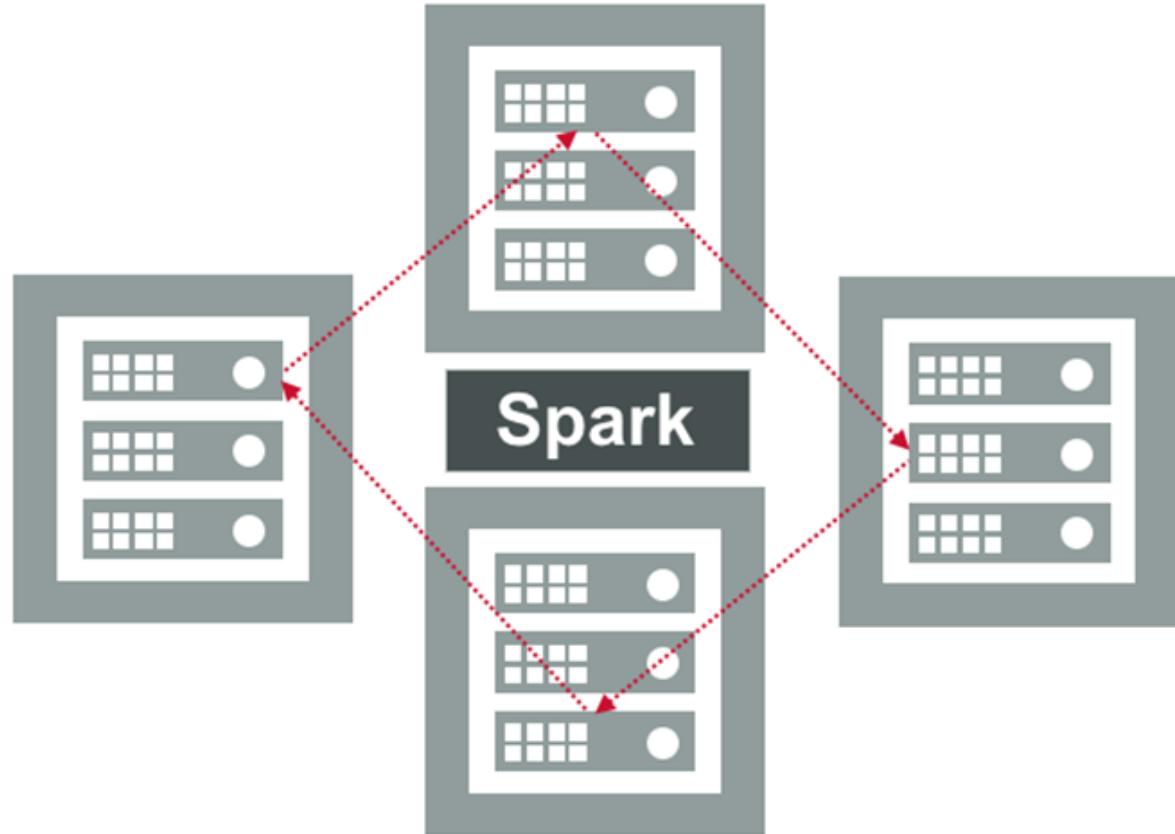
Mode 1: Local

Local mode is useful for prototyping, developing, debugging, and testing.



Mode 2: Standalone

- Simple standalone deploy mode
- Install by placing compiled version of Spark on each cluster node
- Can be used on multiple nodes



Mode 2: Standalone

```
import org.apache.spark.sql.SparkSession  
  
val spark =  
SparkSession.builder.master("spark://<host_IP>:7077").appName(  
"sfpdApp").getOrCreate()
```

Standalone: Deployment Modes

| Cluster Mode | Client Mode |
|--|--|
| Driver launched from worker process inside cluster | Driver launches in the client process that submitted the job |
| Asynchronous: can quit without waiting for result | Synchronous: must wait for result |

If using `spark-submit` → application automatically distributed to all worker nodes

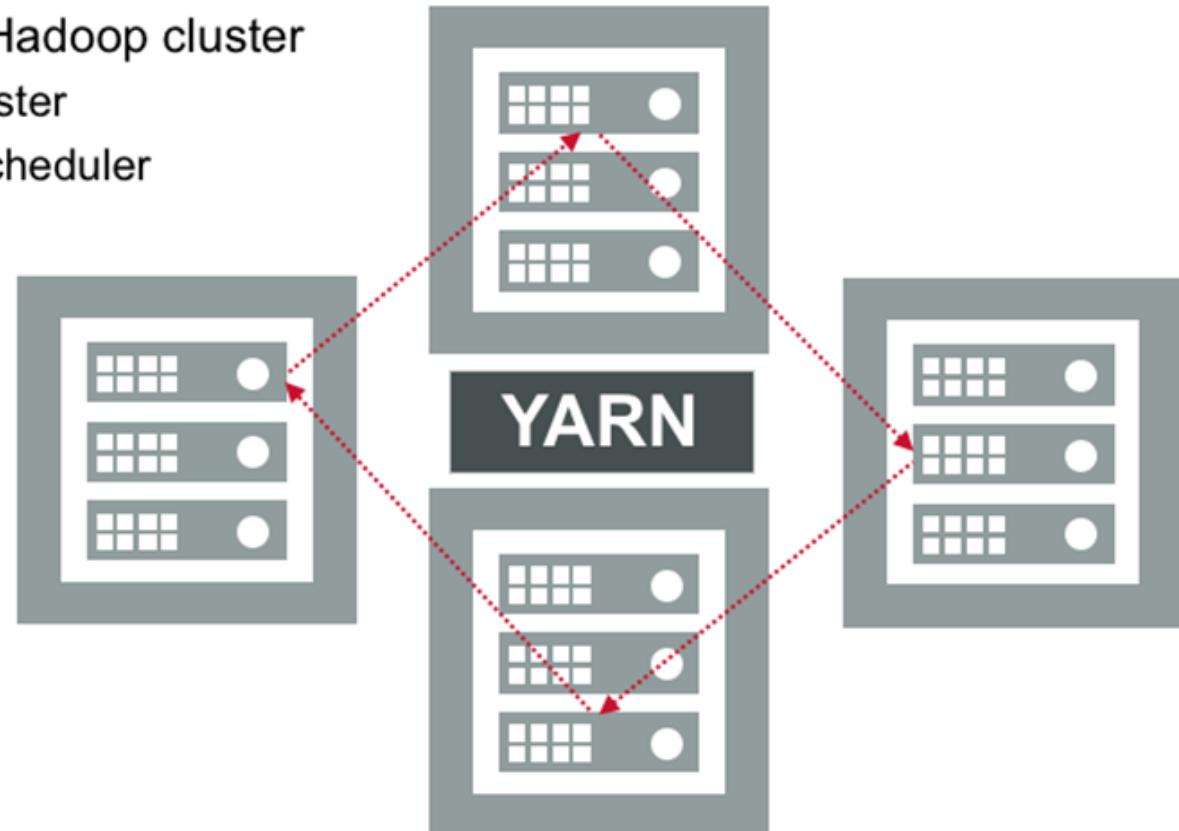
Standalone: Deployment Modes

| Cluster Mode | Client Mode |
|--|--|
| Driver launched from worker process inside cluster | Driver launches in the client process that submitted the job |
| Asynchronous: can quit without waiting for result | Synchronous: must wait for result |

If using `spark-submit` → application automatically distributed to all worker nodes

Mode 3: Hadoop YARN

- Run on YARN if you have Hadoop cluster
 - Uses existing Hadoop cluster
 - Uses features of YARN scheduler



Hadoop YARN: Deployment Modes

| Cluster Mode | Client Mode |
|--|--|
| Driver launched in Application Master in cluster or worker | Driver launched in the client process that submitted the job |
| Can quit without waiting for job results (async) | Need to wait for result when job finishes (sync) |
| Suitable for production deployments | Useful for Spark interactive shell or debugging |

Hadoop YARN: Deployment Modes

| Cluster Mode | Client Mode |
|--|--|
| Driver launched in Application Master in cluster or worker | Driver launched in the client process that submitted the job |
| Can quit without waiting for job results (async) | Need to wait for result when job finishes (sync) |
| Suitable for production deployments | Useful for Spark interactive shell or debugging |

Mode 4: Apache Mesos Cluster Manager

- Mesos Master replaces Spark Master as Cluster Manager
- Driver creates job and Mesos determines what machines handle what tasks
- Multiple frameworks can coexist on same cluster



Apache Mesos: Deployment Modes

Coarse-grained Mode (Default)

Launches one task on each Mesos machine

No sharing between users

Much lower startup overhead

Apache Mesos: Deployment Modes

Coarse-grained Mode (Default)

Launches one task on each Mesos machine

No sharing between users

Much lower startup overhead

Dynamic Resource Allocation

Each Spark task runs as a separate Mesos task

Benefits when enabled:

- Resources allocated dynamically for Spark tasks
- Scales up/down automatically based on running tasks

Class Discussion



Based on what you've learned about the different launch modes available, which mode would you use and why? What are some of the advantages offered by these different modes?

Launch a Program: Package Application

Package application and dependencies
into a .jar file (SBT or Maven)



Launch a Program: Use spark-submit

Use `./bin/spark-submit` to launch application

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

Mode 1: Local

Run local mode on n cores:

```
./bin/spark-submit --class <classpath> \
--master local[n] \
/path/to/application-jar
```

Mode 2: Standalone

Run Standalone client mode:

```
./bin/spark-submit --class <classpath>\  
-- master spark:<master url> \  
/path/to/application-jar
```

Mode 3: Hadoop YARN

Run on YARN cluster mode:

```
./bin/spark-submit --class <classpath> \
--master yarn-cluster \
/path/to/application-jar
```

Run on YARN client mode:

```
./bin/spark-submit --class <classpath> \
--master yarn-client \
/path/to/application-jar
```

Mode 4: Apache Mesos

Run with Apache Mesos:

```
$ /opt/mapr/spark/spark-2.1.0/bin/spark-submit \
--class "AuctionsApp" \
--master mesos://<host_IP>:5050 \
/path/to/auctions-project_2.11-1.0.jar
```

Monitor the Spark Job

The screenshot shows a web browser window with the URL `maprdemo:18080`. The page title is "History Server" with a "Spark" logo and version "2.1.0-mapr-1707". The main content area displays an application log entry:

| App ID | App Name | Started | Completed | Duration | Spark User | Last Updated | Event Log |
|---------------------|-------------|---------------------|---------------------|----------|------------|---------------------|--------------------------|
| local-1513940088331 | Spark shell | 2017-12-22 10:54:44 | 2017-12-22 11:10:01 | 15 min | mapr | 2017-12-22 11:10:01 | Download |

Below the table, there are two links: "Showing 1 to 1 of 1 entries" and "Show incomplete applications".

SparkHistoryServer Direct Port Access

- `http://<ip address>:18080`

Monitor Spark Applications in Real-Time

- `http://<ip address>:4040`

MapR Control System (MCS)

- `http://<ip address>:8443`
- `Spark History Server`

Knowledge Check



Select the most appropriate command to launch your Scala application, “IncidentsApp” in a YARN cluster mode, where the application with its dependencies is packaged to “/path/to/file/incidentsapp.jar” :

- A. ./bin/spark-submit --class IncidentsApp --master cluster /path/to/file/incidentsapp.jar
- B. ./bin/spark-submit --class IncidentsApp spark://100.10.60.120:7077 /path/to/file/incidentsapp.jar
- C. ./bin/spark-submit --class IncidentsApp --master yarn-cluster /path/to/file/incidentsapp.jar

Lab 4.4: Complete, Package, and Launch the Application



- Estimated time to complete: **20 minutes**
- In this lab, you still start with a partially completed project. The code is marked with `TODO` labels; locate them and follow the instructions to complete the code. Your code will load SFPD data into a Dataset, cache it, and print several values to the console.

The solutions directory contains completed code.

Lesson 5 - Monitor Apache Spark Applications.





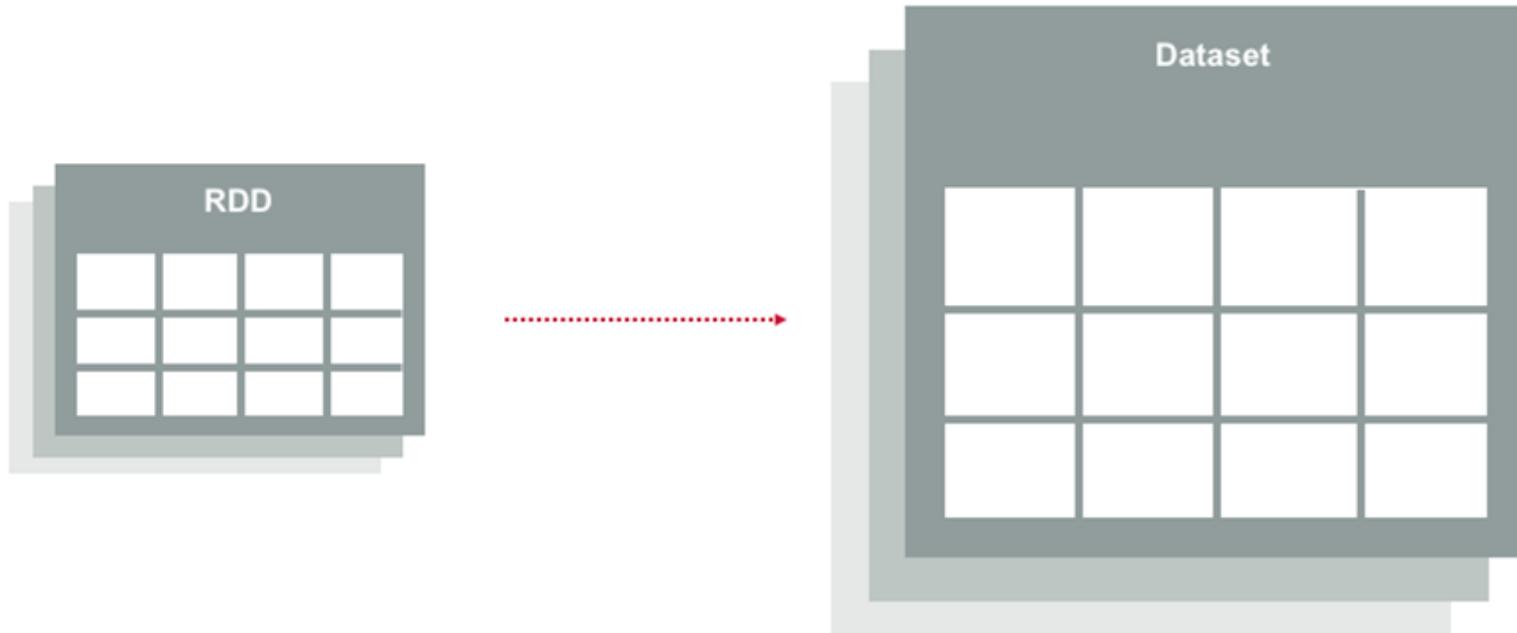
Learning Goals

- 5.1 Describe Logical and Physical Plans of Spark Execution
- 5.2 Use Spark Web UI to Monitor Spark Applications
- 5.3 Debug and Tune Spark Applications

Review



RDDs are the basic building blocks of Spark



Dataset Operations: Transformations

```
sfpdDS.groupBy("category")
```

sfpdDS

| Dataset | |
|------------|----------------|
| incidentnm | category |
| 150599321 | OTHER_OFFENSES |
| 156168837 | LARCENY/THEFT |
| 150599224 | OTHER_OFFENSES |
| 150599230 | VANDALISM |

TRANSFORMATIONS



groupBy()

categoryDS

| Relational Grouped Dataset | |
|----------------------------|--|
| category | |
| OTHER_OFFENSES | |
| LARCENY/THEFT | |
| VANDALISM | |

Components of Spark Execution

| Component | Description |
|-------------------------------------|--|
| Tasks | Unit of work within a stage corresponds to one RDD partition |
| Stages | Group of tasks which perform the same computation in parallel |
| Shuffle | Transfer of data between stages |
| Jobs | Work required to compute RDD; has one or more stages |
| Pipelining | Collapsing of RDDs into a single stage when RDD transformations can be computed without data movement |
| Directed Acyclic Graph (DAG) | Logical graph of Dataset operations |
| Resilient Distributed Dataset (RDD) | Parallel Dataset with partitions |
| Dataset/DataFrame | Distributed collection of data and primary abstraction in Spark. Data is organized as named columns similar to a table in relational database. |

Phases During Spark Execution

PHASE 1

Create the Logical Plan:
User code defines the
DAG

PHASE 2

Actions responsible for
translating DAG into
physical execution plan

PHASE 3

Tasks scheduled and
executed on cluster

Phases During Spark Execution

PHASE 1

Create the Logical Plan:
User code defines the
DAG

PHASE 2

Actions responsible for
translating DAG into
physical execution plan

PHASE 3

Tasks scheduled and
executed on cluster

Phases During Spark Execution

PHASE 1

Create the Logical Plan:
User code defines the
DAG

PHASE 2

Actions responsible for
translating DAG into
physical execution plan

PHASE 3

Tasks scheduled and
executed on cluster

Phases During Spark Execution

PHASE 1

Create the Logical Plan:
User code defines the
DAG

PHASE 2

Actions responsible for
translating DAG into
physical execution plan

PHASE 3

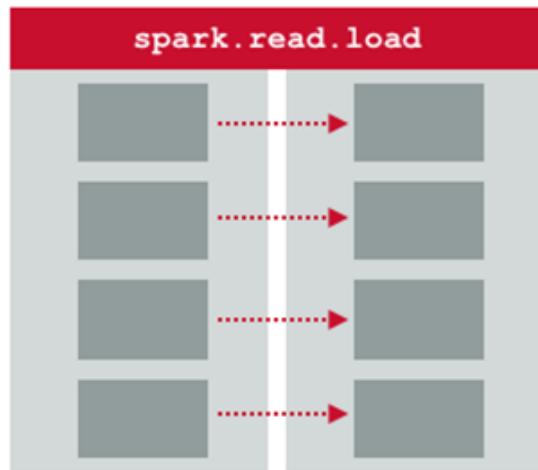
Tasks scheduled and
executed on cluster

Spark Execution Model: Logical Plan

1. val sfpdDF = spark.read.format("csv").option("inferSchema", true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category", "description", "dayofweek", "date", "time", "pddistrict", "resolution", "address", "X", "Y", "pdid")
2. val categoryDS = sfpdDF.groupBy("category")
3. val categoryCountDS = categoryDS.count()
4. categoryCountDS.collect()

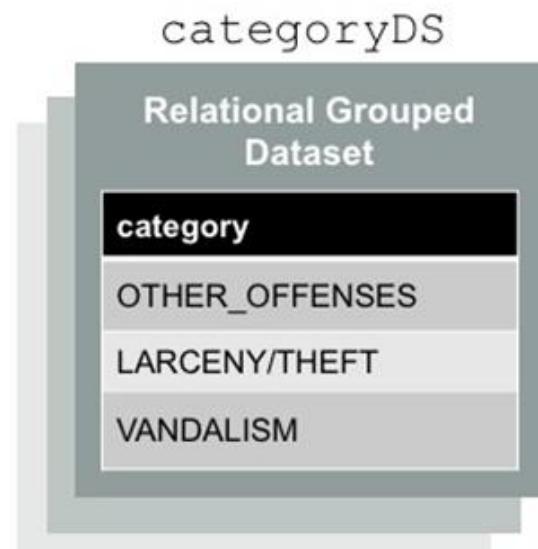
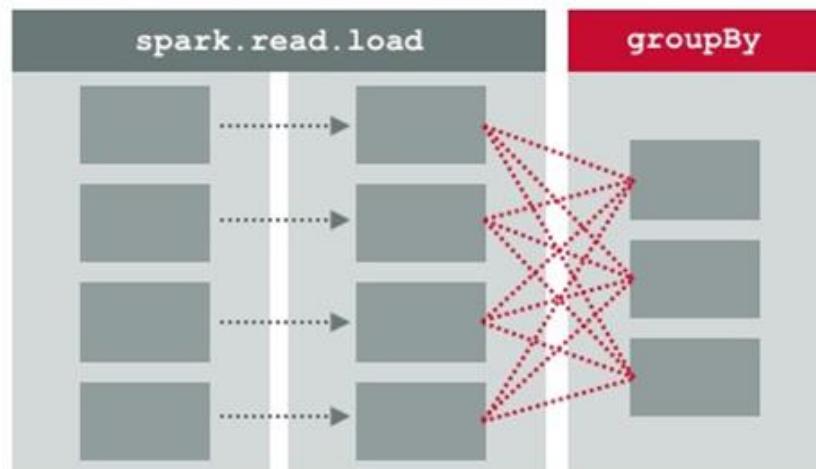
Logical Plan Step 1: Import Data

```
1. val sfpdDF = spark.read.format("csv").option("inferSchema",  
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category",  
"description", "dayofweek", "date", "time", "pddistrict",  
"resolution", "address", "X", "Y", "pdid")
```



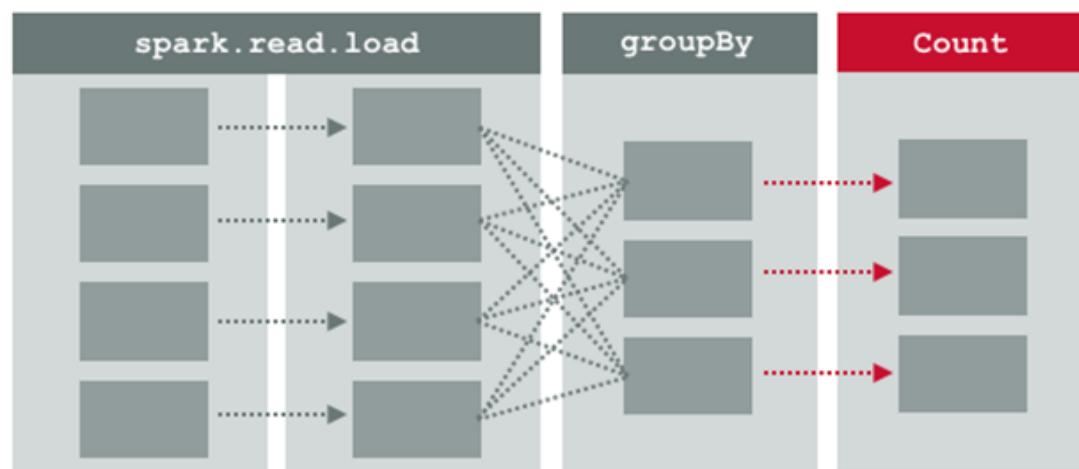
Logical Plan Step 2: Apply groupBy Transformation

2. val categoryDS = sfpdDF.groupBy("category")



Logical Plan Step 3: Apply count Transformation

3. val **categoryCount** = categoryDS.**count()**

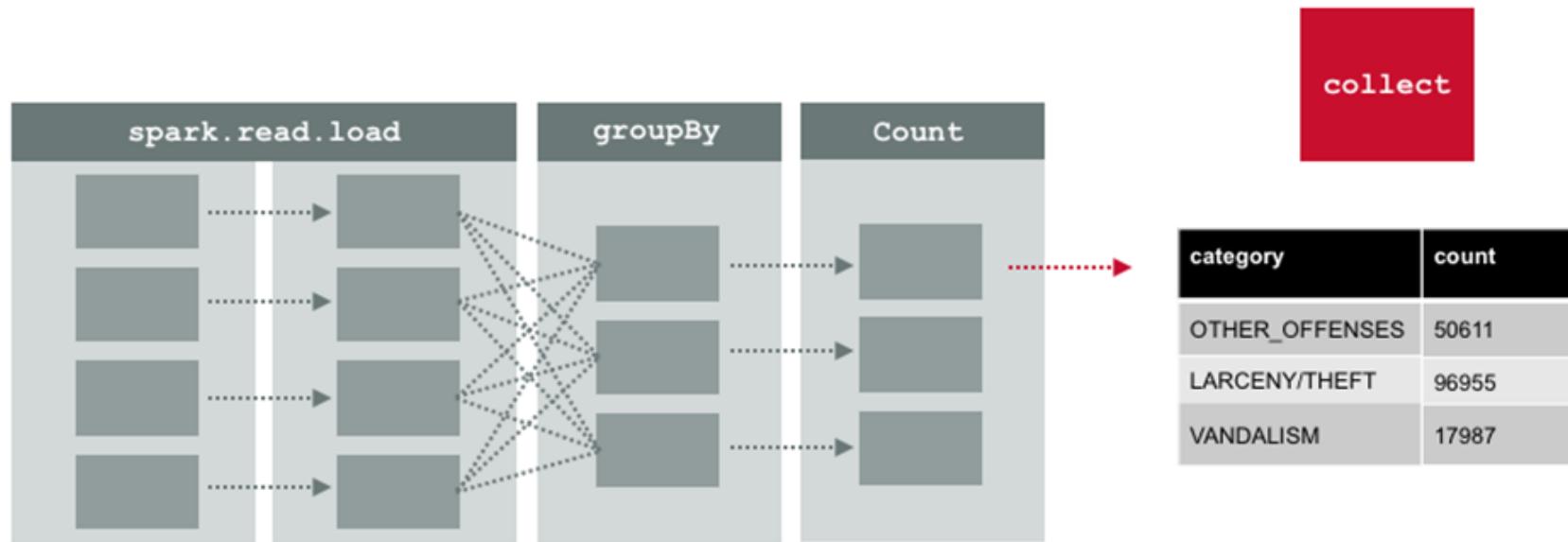


categoryCount

| Dataset | |
|----------------|-------|
| category | count |
| OTHER_OFFENSES | |
| LARCENY/THEFT | |
| VANDALISM | |

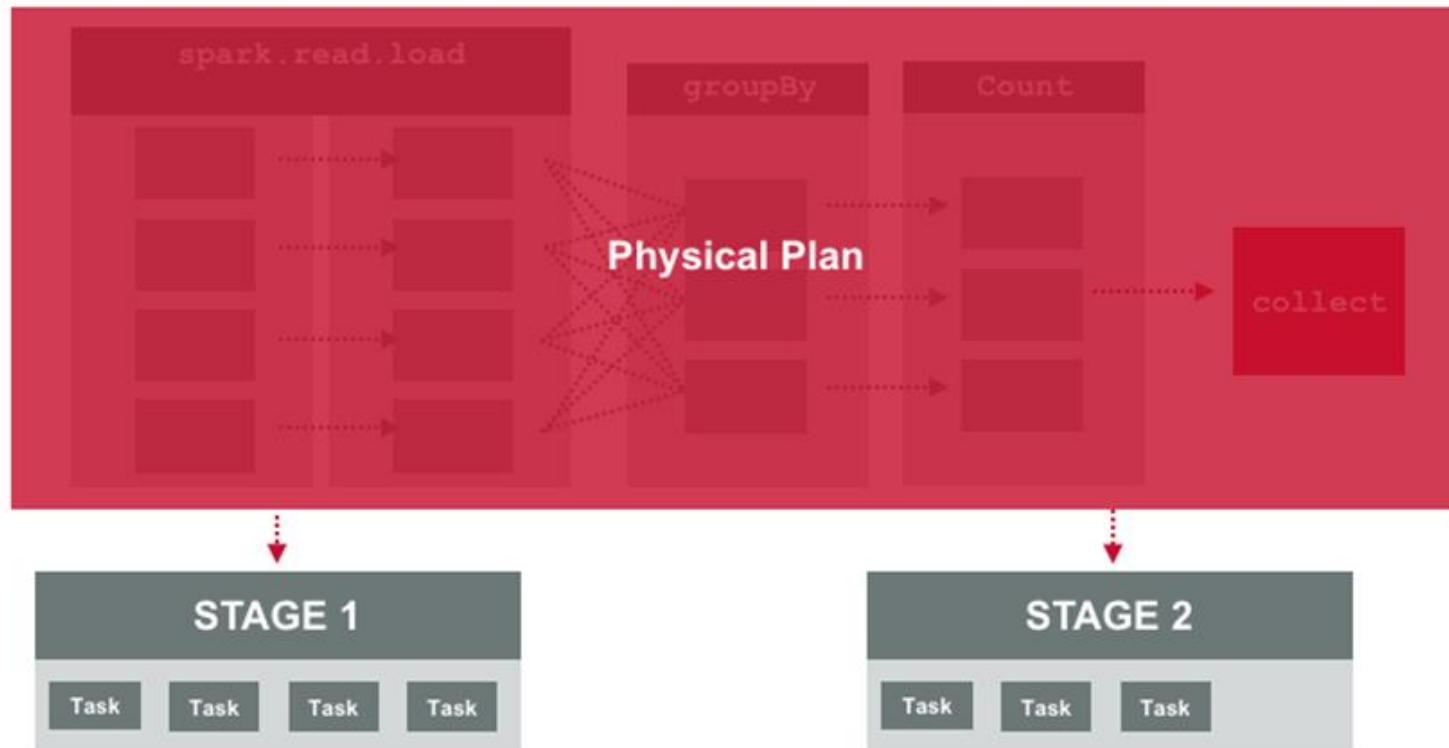
Logical Plan Step 4: Apply collect Action

4. categoryCount.collect()



Physical Plan

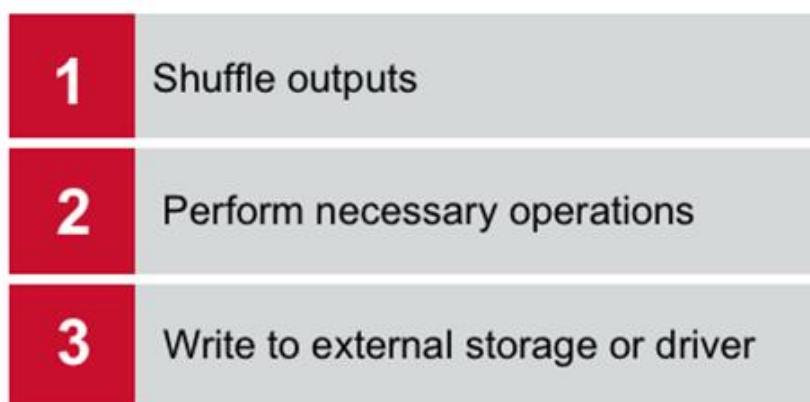
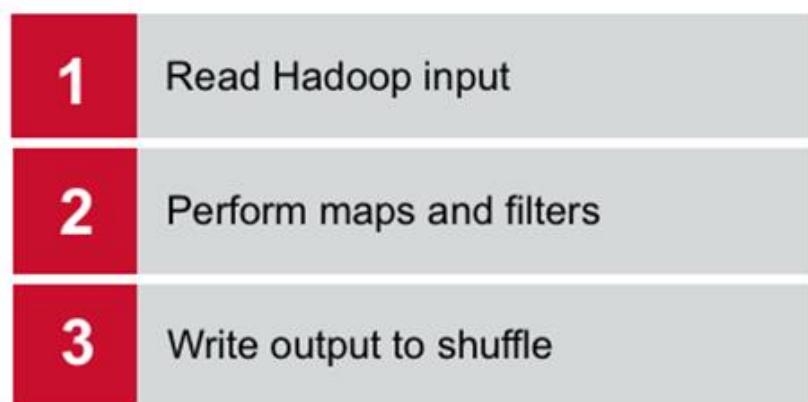
When an action is encountered, the DAG is translated into a physical plan.



Physical Plan: Stages and Tasks

| STAGE | | |
|-------|---|--|
| | Task 1 | Task 2 |
| 1 | Fetch input from data storage or existing Dataset, or shuffle outputs | |
| 2 | | Actions responsible for translating DAG into physical execution plan |
| 3 | | Write output to shuffle, external storage, or back to the driver |

Physical Plan: Stages and Tasks



Knowledge Check



The number of stages in a job is usually equal to the number of Datasets underlying RDDs in the DAG. However, the scheduler can truncate the lineage when:

- A. There is no movement of data from the parent Dataset
- B. There is a shuffle
- C. The Dataset is cached or persisted
- D. The Dataset was materialized due to an earlier shuffle

Spark Web UI

Spark 2.1.0-mapr-1707 Jobs Stages Storage Environment Executors SQL Spark shell application UI

Spark Jobs (?)

User: mapr
Total Uptime: 91.0 h
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 25

Event Timeline

Active Jobs (1)

Detailed progress information and performance metrics

| Job Id | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|--------|--------------------------------|---------------------|----------|-------------------------|---|
| 25 | collect at <console>:24 (kill) | 2017/10/22 06:29:34 | 6 s | 0/1 | 0/1 |

Completed Jobs (25)

| Job Id | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|--------|-------------------------|---------------------|----------|-------------------------|---|
| 24 | collect at <console>:26 | 2017/10/22 06:27:14 | 2 s | 1/1 | 1/1 |
| 23 | collect at <console>:26 | 2017/10/22 05:39:08 | 2 s | 1/1 | 1/1 |
| 22 | collect at <console>:26 | 2017/10/22 05:38:23 | 2 s | 1/1 | 1/1 |
| 21 | collect at <console>:26 | 2017/10/22 05:37:53 | 4 s | 1/1 | 1/1 |

<http://<driver-node>:4040>

Spark Web UI: Jobs Page

```
val sfpdDF = spark.read.format("csv").option("inferSchema",  
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category", "description", "dayofweek", "date",  
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid")  
val incidentCountDF = sfpdDF.groupBy("incidentnum").count()
```

Spark Jobs (?)

User: mapr

Total Uptime: 46 min

Scheduling Mode: FIFO

Completed Jobs: 5

Event Timeline

Completed Jobs (5)

| Job Id | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|--------|-------------------------|---------------------|----------|-------------------------|---|
| 4 | count at <console>:28 | 2017/10/22 10:52:05 | 2 s | 2/2 (1 skipped) | 201/201 (1 skipped) |
| 3 | collect at <console>:28 | 2017/10/22 10:51:41 | 2 s | 1/1 (1 skipped) | 200/200 (1 skipped) |
| 2 | collect at <console>:28 | 2017/10/22 10:45:18 | 12 s | 2/2 | 201/201 |
| 1 | load at <console>:23 | 2017/10/22 10:43:46 | 3 s | 1/1 | 1/1 |
| 0 | load at <console>:23 | 2017/10/22 10:43:45 | 0.5 s | 1/1 | 1/1 |

Spark Web UI: Jobs Page

```
val sfpdDF = spark.read.format("csv").option("inferSchema",  
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category", "description", "dayofweek", "date",  
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid")  
val incidentCountDF = sfpdDF.groupBy("incidentnum").count()  
incidentCountDF.cache()  
incidentCountDF.collect
```

Spark Jobs (?)

User: mapr

Total Uptime: 46 min

Scheduling Mode: FIFO

Completed Jobs: 5

▶ Event Timeline

Completed Jobs (5)

| Job Id | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|--------|-------------------------|---------------------|----------|-------------------------|---|
| 4 | count at <console>:28 | 2017/10/22 10:52:05 | 2 s | 2/2 (1 skipped) | 201/201 (1 skipped) |
| 3 | collect at <console>:28 | 2017/10/22 10:51:41 | 2 s | 1/1 (1 skipped) | 200/200 (1 skipped) |
| 2 | collect at <console>:28 | 2017/10/22 10:45:18 | 12 s | 2/2 | 201/201 |
| 1 | load at <console>:23 | 2017/10/22 10:43:46 | 3 s | 1/1 | 1/1 |
| 0 | load at <console>:23 | 2017/10/22 10:43:45 | 0.5 s | 1/1 | 1/1 |

Spark Web UI: Jobs Page

```
val sfpdDF = spark.read.format("csv").option("inferSchema",  
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category", "description", "dayofweek", "date",  
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid")  
val incidentCountDF = sfpdDF.groupBy("incidentnum").count()  
incidentCountDF.cache()  
incidentCountDF.collect  
incidentCountDF.collect
```

Spark Jobs [\(?\)](#)

User: mapr

Total Uptime: 46 min

Scheduling Mode: FIFO

Completed Jobs: 5

[Event Timeline](#)

Completed Jobs (5)

| Job Id | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|--------|-------------------------|---------------------|----------|-------------------------|---|
| 4 | count at <console>:28 | 2017/10/22 10:52:05 | 2 s | 2/2 (1 skipped) | 201/201 (1 skipped) |
| 3 | collect at <console>:28 | 2017/10/22 10:51:41 | 2 s | 1/1 (1 skipped) | 200/200 (1 skipped) |
| 2 | collect at <console>:28 | 2017/10/22 10:45:18 | 12 s | 2/2 | 201/201 |
| 1 | load at <console>:23 | 2017/10/22 10:43:46 | 3 s | 1/1 | 1/1 |
| 0 | load at <console>:23 | 2017/10/22 10:43:45 | 0.5 s | 1/1 | 1/1 |

Spark Web UI: Jobs Page

```
val sfpdDF = spark.read.format("csv").option("inferSchema",  
true).load("/spark/lab5/sfpd.csv").toDF("incidentnum", "category", "description", "dayofweek", "date",  
"time", "pddistrict", "resolution", "address", "X", "Y", "pdid")  
val incidentCountDF = sfpdDF.groupBy("incidentnum").count()  
incidentCountDF.cache()  
incidentCountDF.collect  
incidentCountDF.collect  
incidentCountDF.count
```

Spark Jobs (?)

User: mapr

Total Uptime: 46 min

Scheduling Mode: FIFO

Completed Jobs: 5

▶ Event Timeline

Completed Jobs (5)

| Job Id | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|--------|-------------------------|---------------------|----------|-------------------------|---|
| 4 | count at <console>:28 | 2017/10/22 10:52:05 | 2 s | 2/2 (1 skipped) | 201/201 (1 skipped) |
| 3 | collect at <console>:28 | 2017/10/22 10:51:41 | 2 s | 1/1 (1 skipped) | 200/200 (1 skipped) |
| 2 | collect at <console>:28 | 2017/10/22 10:45:18 | 12 s | 2/2 | 201/201 |
| 1 | load at <console>:23 | 2017/10/22 10:43:46 | 3 s | 1/1 | 1/1 |
| 0 | load at <console>:23 | 2017/10/22 10:43:45 | 0.5 s | 1/1 | 1/1 |

Class Discussion



The second Collect and Count jobs had skipped one stage. Why was one stage “skipped” and why is the duration lesser than job 2?

Spark Jobs (?)

User: mapr

Total Uptime: 46 min

Scheduling Mode: FIFO

Completed Jobs: 5

» Event Timeline

Completed Jobs (5)

| Job Id | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|--------|-------------------------|---------------------|----------|-------------------------|---|
| 4 | count at <console>:28 | 2017/10/22 10:52:05 | 2 s | 2/2 (1 skipped) | 201/201 (1 skipped) |
| 3 | collect at <console>:28 | 2017/10/22 10:51:41 | 2 s | 1/1 (1 skipped) | 200/200 (1 skipped) |
| 2 | collect at <console>:28 | 2017/10/22 10:45:18 | 12 s | 2/2 | 201/201 |
| 1 | load at <console>:23 | 2017/10/22 10:43:46 | 3 s | 1/1 | 1/1 |
| 0 | load at <console>:23 | 2017/10/22 10:43:45 | 0.5 s | 1/1 | 1/1 |

Spark Web UI: Job Details

Details for Job 2

Status: SUCCEEDED

Completed Stages: 2

- ▶ Event Timeline
- ▶ DAG Visualization

Completed Stages (2)

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write |
|----------|-------------------------------------|------------------------|----------|---------------------------|---------|--------|--------------|---------------|
| 3 | collect at <console>:28 +details | 2017/10/22 10:45:24 | 6 s | 200/200 | | | 2.0 MB | |
| 2 | cache at <console>:28 +details | 2017/10/22 10:45:18 | 6 s | 1/1 | 55.1 MB | | | 2.0 MB |

Spark Web UI: Job Details

Details for Job 3

Status: SUCCEEDED

Completed Stages: 1

Skipped Stages: 1

▶ Event Timeline

▶ DAG Visualization

Completed Stages (1)

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write |
|----------|-------------------------------------|------------------------|----------|------------------------|-----------|--------|--------------|---------------|
| 5 | collect at <console>:28 +details | 2017/10/22 10:51:41 | 2 s | 200/200 | 1515.9 KB | | | |

Skipped Stages (1)

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write |
|----------|-----------------------------------|-----------|----------|------------------------|-------|--------|--------------|---------------|
| 4 | cache at <console>:28 +details | Unknown | Unknown | 0/1 | | | | |

Spark Web UI: Stage Details for Job 1

Details for Stage 3 (Attempt 0)

Total Time Across All Tasks: 3 s

Locality Level Summary: Any: 200

Shuffle Read: 2.0 MB / 295185

[DAG Visualization](#)

[Show Additional Metrics](#)

[Event Timeline](#)

Summary Metrics for 200 Completed Tasks

| Metric | Min | 25th percentile | Median | 75th percentile | Max |
|-----------------------------|---------------|-----------------|----------------|-----------------|----------------|
| Duration | 5 ms | 6 ms | 8 ms | 12 ms | 0.6 s |
| GC Time | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms |
| Shuffle Read Size / Records | 9.3 KB / 1340 | 10.1 KB / 1449 | 10.3 KB / 1476 | 10.5 KB / 1502 | 11.0 KB / 1571 |

Aggregated Metrics by Executor

| Executor ID | Address | Task Time | Total Tasks | Failed Tasks | Killed Tasks | Succeeded Tasks | Shuffle Read Size / Records |
|-------------|-----------------------|-----------|-------------|--------------|--------------|-----------------|-----------------------------|
| driver | 192.168.100.128:37279 | 6 s | 200 | 0 | 0 | 200 | 2.0 MB / 295185 |

Tasks (200)

| Index | ID | Attempt | Status | Locality Level | Executor ID / Host | Launch Time | Duration | GC Time | Shuffle Read Size / Records | Errors |
|-------|----|---------|---------|----------------|--------------------|---------------------|----------|---------|-----------------------------|--------|
| 0 | 3 | 0 | SUCCESS | ANY | driver / localhost | 2017/10/22 10:45:24 | 0.6 s | | 10.1 KB / 1449 | |
| 1 | 4 | 0 | SUCCESS | ANY | driver / localhost | 2017/10/22 10:45:24 | 0.1 s | | 10.1 KB / 1436 | |

Page: [1](#) [2](#) >

2 Pages, Jump to , Show items in a page. [Go](#)

Spark Web UI: Storage Page



2.1.0-mapr-1707

| Jobs | Stages | Storage | Environment | Executors | SQL | Spark shell application UI |
|---|-----------------------------------|-------------------|-----------------|----------------|--------------|----------------------------|
| Storage | | | | | | |
| RDDs | | | | | | |
| RDD Name | Storage Level | Cached Partitions | Fraction Cached | Size in Memory | Size on Disk | |
| <pre>*HashAggregate(keys=[incidentnum#25], functions=[count(1)], output=[incidentnum#25, count#64L]) + Exchange hashpartitioning(incidentnum#25, 200) + *HashAggregate(keys=[incidentnum#25], functions=[partial_count(1)], output=[incidentnum#25, count#69L]) + *Project [_c0#0 AS incidentnum#25] + *FileScan csv [_c0#0] Batched: false, Format: CSV, Location: InMemoryFileIndex[maprfs://spark/lab5/sfpd.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<_c0:int></pre> | Memory Deserialized 1x Replicated | 200 | 100% | 1515.9 KB | 0.0 B | |

Spark Web UI: Storage Page

RDD Storage Info for *HashAggregate(keys=[incidentnum#25], functions=[count(1)], output=[incidentnum#25, count#64L]) +- Exchange hashpartitioning(incidentnum#25, 200) +- *HashAggregate(keys=[incidentnum#25], functions=[partial_count(1)], output=[incidentnum#25, count#69L]) +- *Project [_c0#0 AS incidentnum#25] +- *FileScan csv [_c0#0] Batched: false, Format: CSV, Location: InMemoryFileIndex[maprfs:///spark/lab5/sfpd.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<_c0:int>

Storage Level: Memory Deserialized 1x Replicated

Cached Partitions: 200

Total Partitions: 200

Memory Size: 1515.9 KB

Disk Size: 0.0 B

Data Distribution on 1 Executors

| Host | Memory Usage | Disk Usage |
|-----------------------|--------------------------------|------------|
| 192.168.100.128:37279 | 1515.9 KB (364.8 MB Remaining) | 0.0 B |

200 Partitions

| Page: 1 2 > | 2 Pages. Jump to 1 | | Show 100 items in a page. | Go |
|-------------|-----------------------------------|----------------|---------------------------|-----------------------|
| Block Name | Storage Level | Size in Memory | Size on Disk | Executors |
| rdd_11_0 | Memory Deserialized 1x Replicated | 7.5 KB | 0.0 B | 192.168.100.128:37279 |
| rdd_11_1 | Memory Deserialized 1x Replicated | 7.4 KB | 0.0 B | 192.168.100.128:37279 |

Spark Web UI: Environment Page

Spark 2.1.0-mapr-1707 Jobs Stages Storage Environment Executors SQL Spark shell application UI

Environment

Runtime Information

| Name | Value |
|---------------|---|
| Java Home | /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.79.x86_64/jre |
| Java Version | 1.7.0_79 (Oracle Corporation) |
| Scala Version | version 2.11.8 |

Spark Properties

| Name | Value |
|------------------------|---------------------|
| spark.app.id | local-1508691998953 |
| spark.app.name | Spark shell |
| spark.driver.host | 192.168.100.128 |
| spark.driver.port | 41576 |
| spark.eventLog.dir | maprfs://apps/spark |
| spark.eventLog.enabled | true |

Spark Web UI: Executors Page

Spark 2.1.0-mapr-1707 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 |
|-----------|------------|-------------------|-----------|-------|--------------|--------------|----------------|-------------|---------------------|----------|--------------|---------------|
| Active(1) | 201 | 1.6 MB / 384.1 MB | 0.0 B | 1 | 0 | 0 | 604 | 604 | 19 s (2 s) | 118.7 MB | 0.0 B | 2.1 MB |
| 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 |
| Total(1) | 201 | 1.6 MB / 384.1 MB | 0.0 B | 1 | 0 | 0 | 604 | 604 | 19 s (2 s) | 118.7 MB | 0.0 B | 2.1 MB |

Executors

Show 20 entries Search:

| 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 | Thread Dump |
|-------------|-----------------------|--------|------------|-------------------|-----------|-------|--------------|--------------|----------------|-------------|---------------------|----------|--------------|---------------|-------------|
| driver | 192.168.100.128:37279 | Active | 201 | 1.6 MB / 384.1 MB | 0.0 B | 1 | 0 | 0 | 604 | 604 | 19 s (2 s) | 118.7 MB | 0.0 B | 2.1 MB | Thread Dump |

Showing 1 to 1 of 1 entries [Previous](#) [Next](#)

Access executors stack trace

Spark Web UI: SQL Page

AUTHOR: SPARK 2.1.0-mapr-1707

Jobs Stages Storage Environment Executors SQL Spark shell application UI

SQL

Completed Queries

| ID | Description | Submitted | Duration | Jobs |
|----|-------------------------|------------------------------|----------|------|
| 2 | count at <console>:28 | +details 2017/10/22 10:52:05 | 2 s | 4 |
| 1 | collect at <console>:28 | +details 2017/10/22 10:51:41 | 2 s | 3 |
| 0 | collect at <console>:28 | +details 2017/10/22 10:45:18 | 12 s | 2 |

Access Query
Plan Details

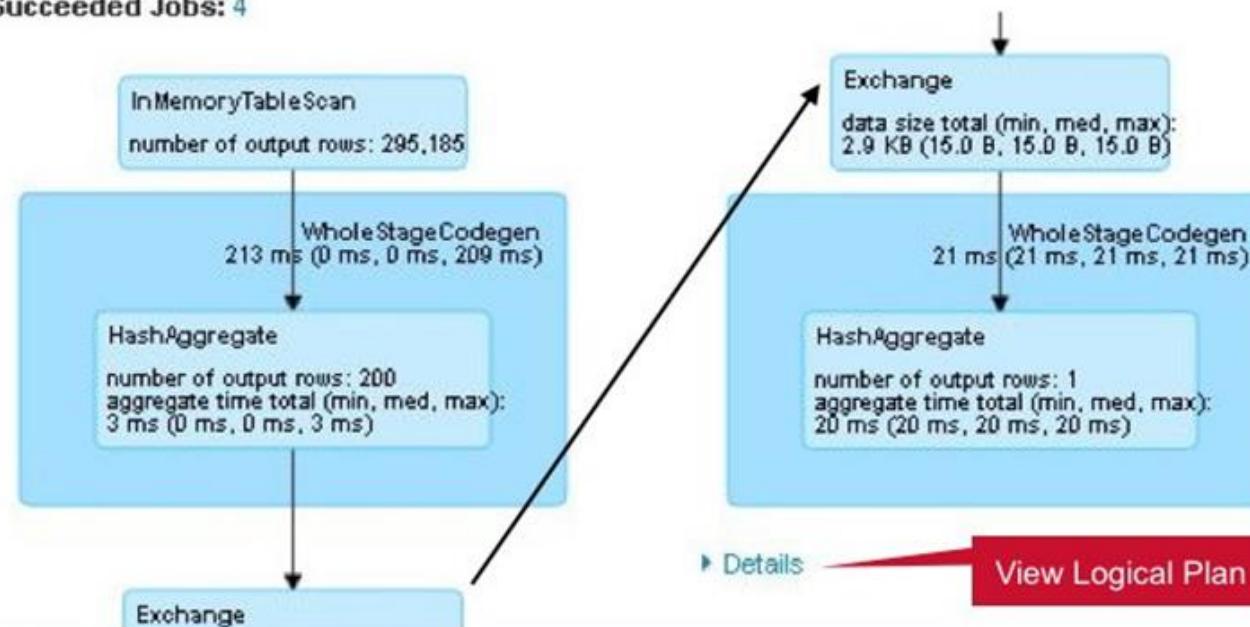
Spark Web UI: SQL Page

Details for Query 2

Submitted Time: 2017/10/22 10:52:05

Duration: 2 s

Succeeded Jobs: 4



Knowledge Check



What are some of the things you can monitor in the Spark Web UI?

- A. Which stages are running slow
- B. Your application has the resources as expected
- C. If the datasets are fitting into memory
- D. All of the above



Labs 5.2a and 5.2b

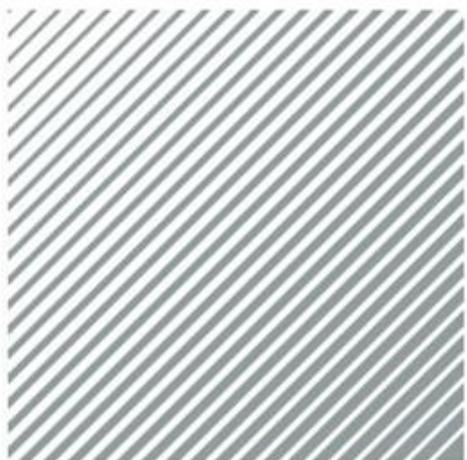
- Estimated time to complete: **25 minutes**
- In these two short labs you will explore different Spark interfaces and resources:
 - In Lab 5.2a (15 minutes), you will use the Spark interactive shell to load and transform data. You will then view information in the Spark History Server page and the Spark Web UI.
 - In Lab 5.2b (10 minutes), you will perform a more in-depth exploration of the Spark Web UI.



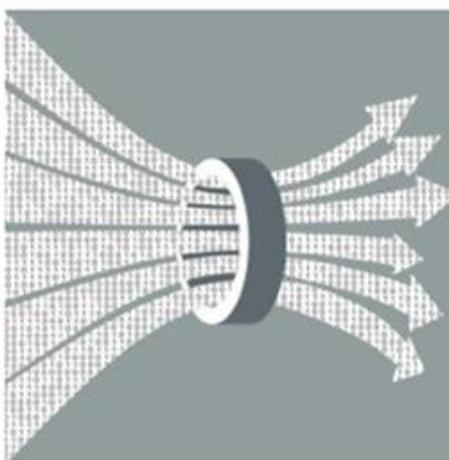
Learning Goals

- 5.1 Describe Logical and Physical Plans of Spark Execution
- 5.2 Use Spark Web UI to Monitor Spark Applications
- 5.3 Debug and Tune Spark Applications

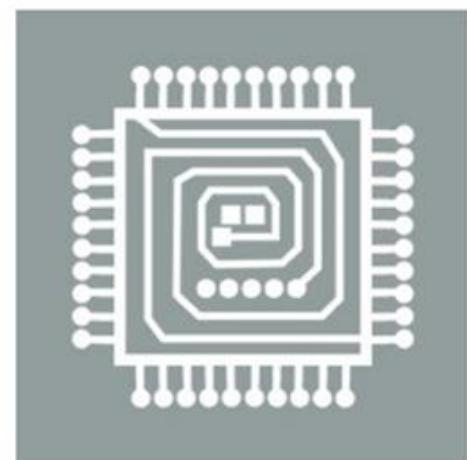
Common Slow Performance Issues



**Level of
Parallelism**

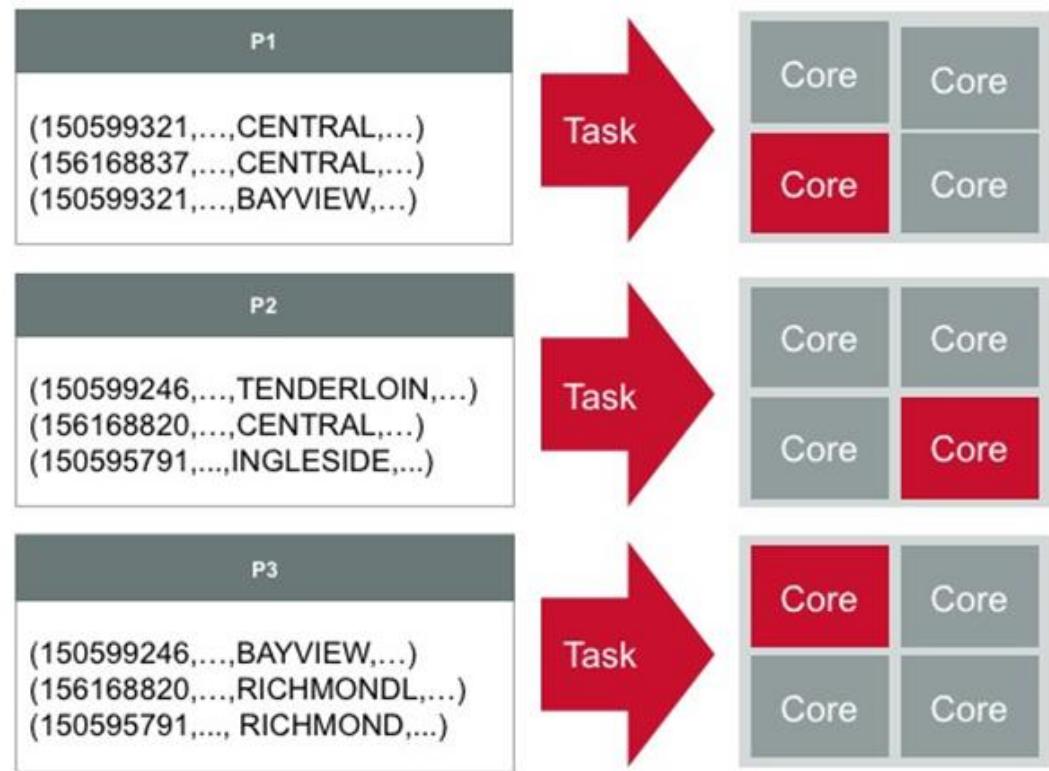


**Serialization
Format**



**Memory
Management**

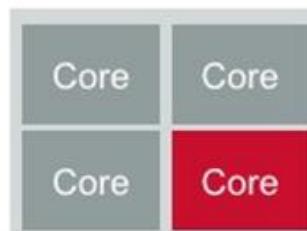
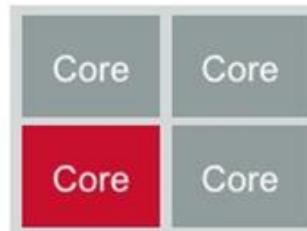
Review: Partitioning



Common Performance Issues: Level of Parallelism



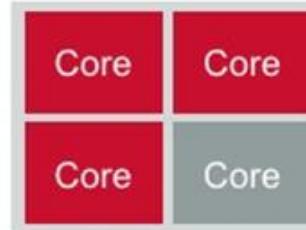
**Level of
Parallelism**



Common Performance Issues: Level of Parallelism



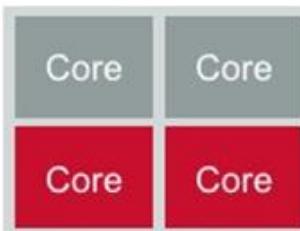
**Level of
Parallelism**



Common Performance Issues: Level of Parallelism



**Level of
Parallelism**



Tuning Level of Parallelism



**Level of
Parallelism**

1

Check the number of partitions:

`ds.rdd.partitions.size()`

or

`ds.rdd.getNumPartitions()`

Tuning Level of Parallelism



**Level of
Parallelism**

2

Tune level of parallelism:

Change the number of partitions

`coalesce()`

`repartition()`

Common Performance Issues: Serialization Format



**Serialization
Format**

- During data transfer, Spark serializes data
- Happens during shuffle operations
- Datasets use a specialized encoder for serialization

Tuning Options: Serialization Format



- Increase network bandwidth
- Decrease data movement between nodes

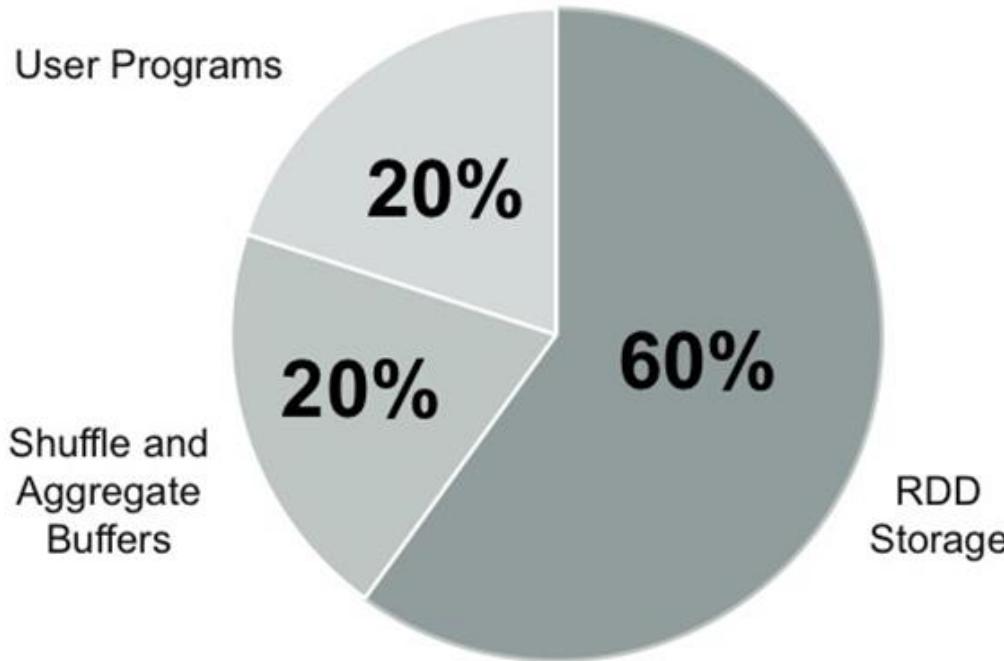
**Serialization
Format**

Common Performance Issues: Memory Management

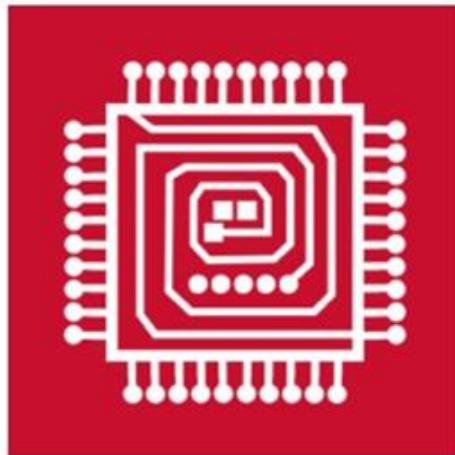
Memory used in different ways in Spark



**Memory
Management**



Tuning Memory Usage



Memory Management

```
cache()  
persist()  
persist(MEMORY_ONLY)  
persist(MEMORY_ONLY_SER)
```

Tuning Memory Usage



+
`persist(MEMORY_AND_DISK)`

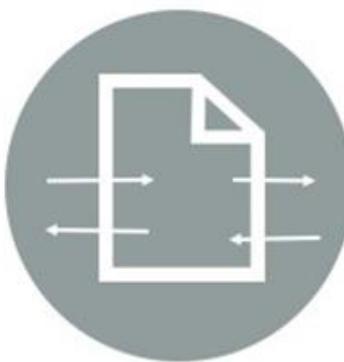
**Memory
Management**

Detect Performance Problems

Spark Web UI: Jobs, Stages, and Stage Details



Slow Tasks



Read/Write Issues



Skew

Detect Performance Problems



Slow Tasks

- Are there any tasks that are significantly slow?
- Are tasks running on certain nodes slow?



Read/Write Issues

- Do some tasks read or write much more data than others?
- How much time tasks spend on each phase: read, compute, write?



Skew

- Is issue because of skew?

Detect Performance Problems



Slow Tasks

- Are there any tasks that are significantly slow?
- Are tasks running on certain nodes slow?



Read/Write Issues

- **Do some tasks read or write much more data than others?**
- **How much time tasks spend on each phase: read, compute, write?**



Skew

- Is issue because of skew?

Detect Performance Problems



Slow Tasks

- Are there any tasks that are significantly slow?
- Are tasks running on certain nodes slow?



Read/Write Issues

- Do some tasks read or write much more data than others?
- How much time tasks spend on each phase: read, compute, write?



Skew

- Is issue because of skew?

Log Files

Spark logging subsystem based on log4j

| Deployment Mode | Location |
|------------------|---|
| Spark Standalone | work/ directory of distribution on each worker node |
| Apache Mesos | work/ directory of Mesos slave node |
| Hadoop YARN | Use YARN log collection tool |

Log Files

| Deployment Mode | Location |
|------------------|---|
| Spark Standalone | work/ directory of distribution on each worker node |
| Apache Mesos | work/ directory of Mesos slave node |
| Hadoop YARN | Use YARN log collection tool |

Log Files

| Deployment Mode | Location |
|------------------|---|
| Spark Standalone | work/ directory of distribution on each worker node |
| Apache Mesos | work/ directory of Mesos slave node |
| Hadoop YARN | Use YARN log collection tool |

Log Files

| Deployment Mode | Location |
|------------------|---|
| Spark Standalone | work/ directory of distribution on each worker node |
| Apache Mesos | work/ directory of Mesos slave node |
| Hadoop YARN | Use YARN log collection tool |

Knowledge Check



Some ways to improve performance of your Spark application include:

- A. Tune the degree of parallelism
- B. Avoid shuffling large amounts of data
- C. Use `ds.rdd.partitions.size()` or `ds.rdd.getNumPartitions()`
- D. Check your log files

Lesson 6: Create an Apache Spark Streaming Application.





Learning Goals

- 6.1 Describe Spark Streaming Architecture
- 6.2 Create a Spark Structured Streaming Application
 - Define Use Case
 - Basic Steps and Save Data to Parquet Tables
- 6.3 Apply Operations on Streaming DataFrames
- 6.4 Define Windowed Operations
- 6.5 Describe How Streaming Applications are Fault Tolerant



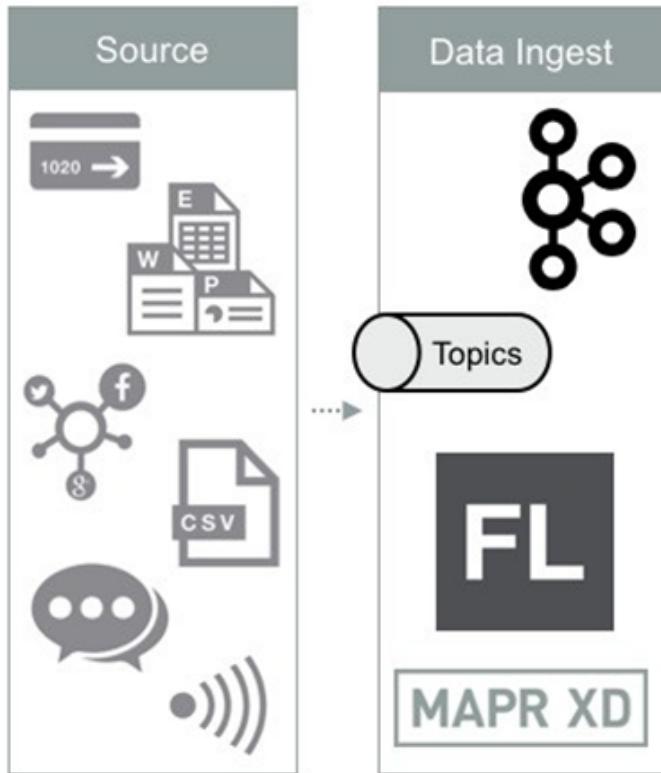
Learning Goals

- 6.1 **Describe Spark Streaming Architecture**
- 6.2 Create a Spark Structured Streaming Application
 - Define Use Case
 - Basic Steps and Save Data to Parquet Tables
- 6.3 Apply Operations on Streaming DataFrames
- 6.4 Define Windowed Operations
- 6.5 Describe How Streaming Applications are Fault Tolerant

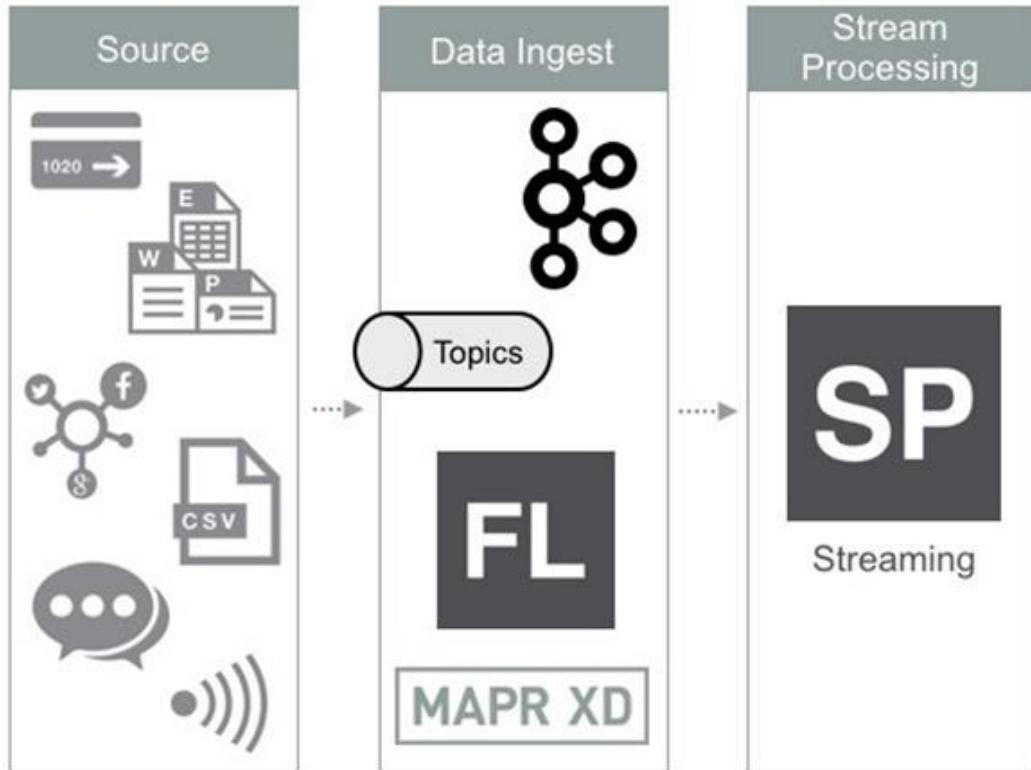
Stream Processing Architecture: Streaming Data Source



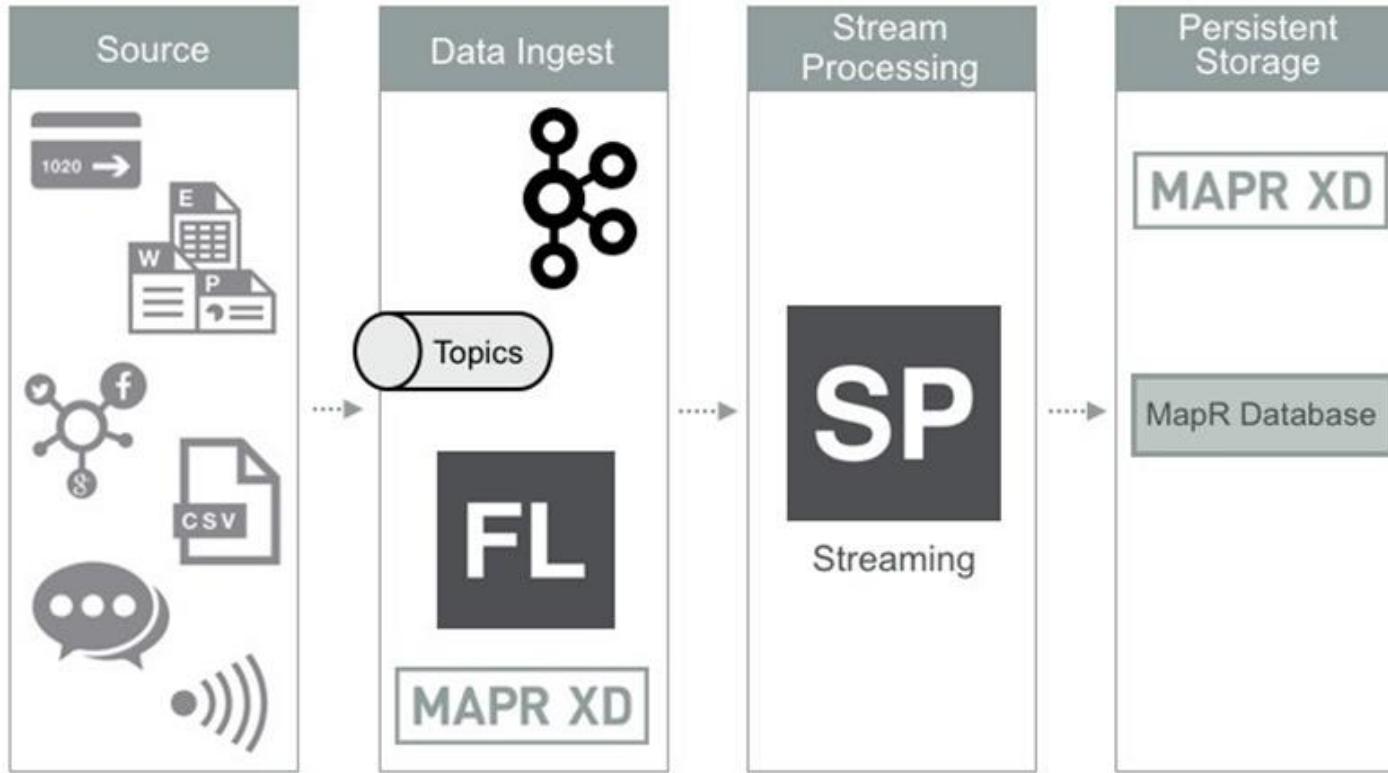
Stream Processing Architecture: Ingest Data



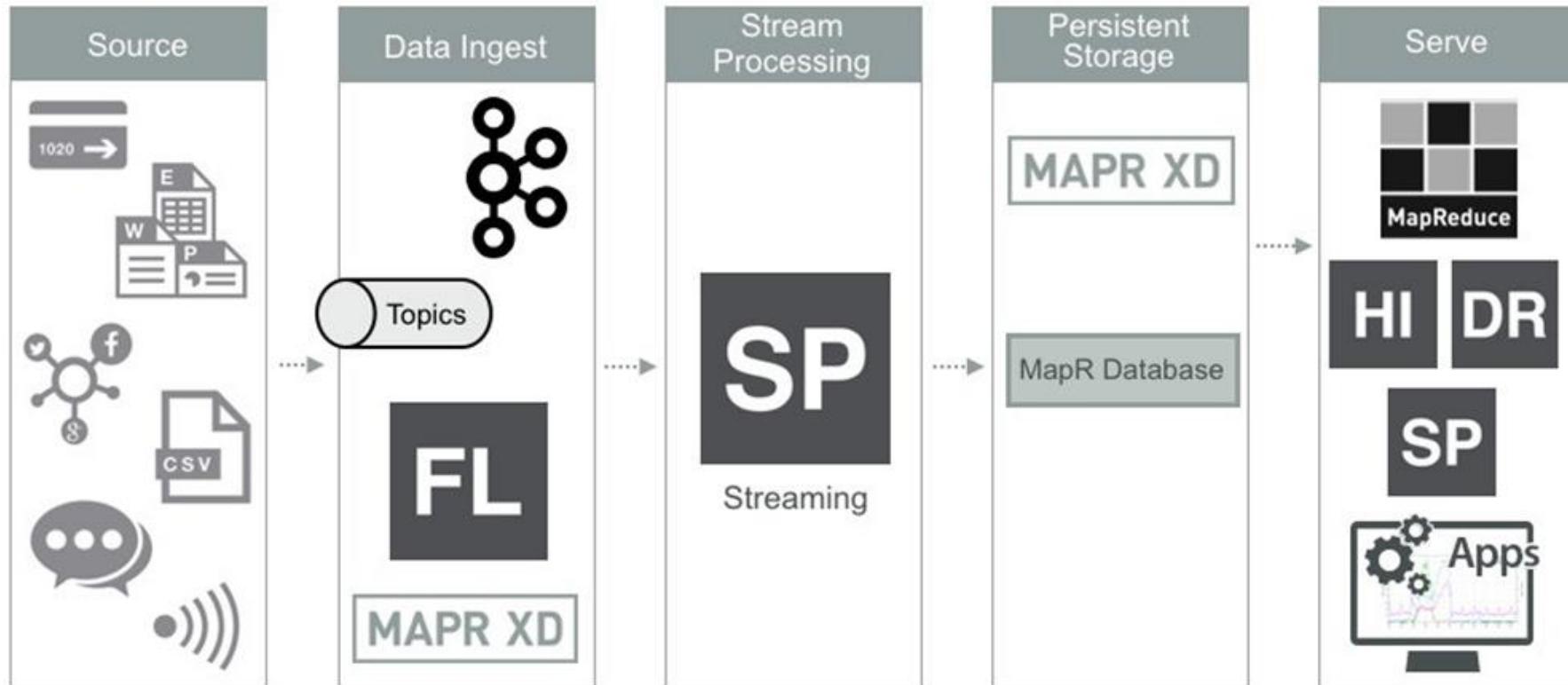
Stream Processing Architecture: Process Data



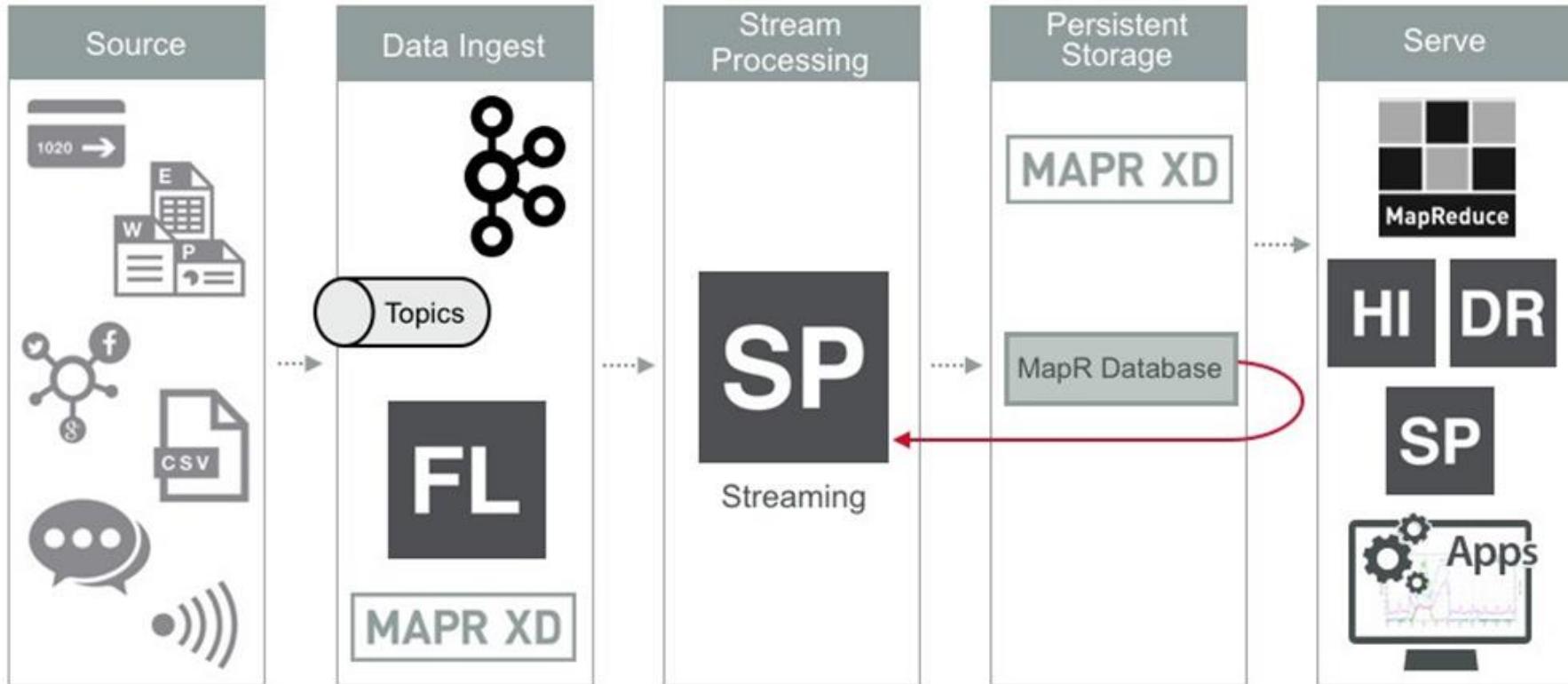
Stream Processing Architecture: Store Processed Data



Stream Processing Architecture: Visualize Processed Data



Stream Processing Architecture: Store Output



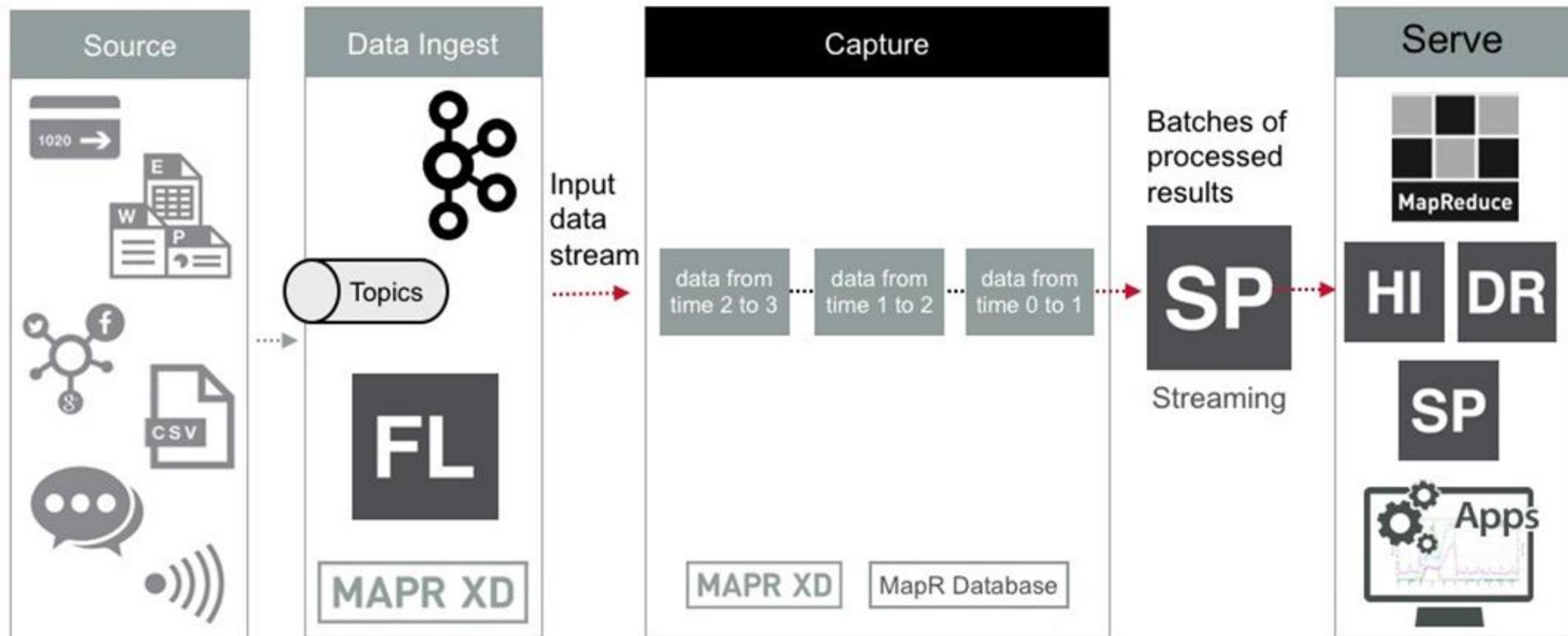
Knowledge Check



Indicate whether the following statements are TRUE or FALSE.

- Ingesting and processing streaming data involves a structured flow, with flexibility and options at each step.
- The output of processing must go to a data visualization application.
- Streaming data is data that comes from a remote, cloud-based source.

Spark Streaming Architecture



Structured Streaming

Time ↓

| Transactions | | | | | | |
|--------------|--------|---------|---------|-----------|-----|--|
| id | first | last | amt | city | ... | |
| 11894 | Jane | Roberts | 1255.76 | San Jose | | |
| 90083 | Rajesh | Gidwani | 504.12 | Edinburgh | | |
| 16884 | Hanna | Park | 75.99 | Hamburg | | |
| 66792 | Ankur | Dawar | 446.90 | Madison | | |

Structured Streaming

Time

Transactions



| id | first | last | amt | city | ... |
|-----------|--------------|-------------|------------|-------------|------------|
| 11894 | Jane | Roberts | 1255.76 | San Jose | |
| 90083 | Rajesh | Gidwani | 504.12 | Edinburgh | |
| 16884 | Hanna | Park | 75.99 | Hamburg | |
| 66792 | Ankur | Dawar | 446.90 | Madison | |

44872 Audrey Chan 25.53 New York



22936 James Greene 2956.22 New York



00481 Michael Chau 1125.99 Hong Kong



Understanding Structured Streaming

Current Data

| id | first | last | amt | city | ... |
|-----------|--------------|-------------|------------|-------------|-----|
| 11894 | Jane | Roberts | 1255.76 | San Jose | |
| 90083 | Rajesh | Gidwani | 504.12 | Edinburgh | |
| 16884 | Hanna | Park | 75.99 | Hamburg | |
| 66792 | Ankur | Dawar | 446.90 | Madison | |
| 44872 | Audrey | Chan | 25.53 | New York | |
| 22936 | James | Greene | 2956.22 | New York | |
| 00481 | Michael | Chau | 1125.99 | Hong Kong | |

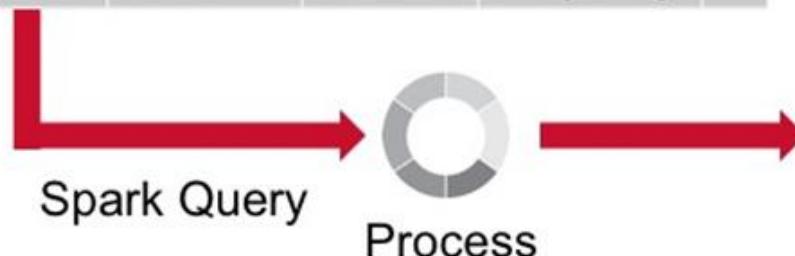
Spark
Query



Understanding Structured Streaming

Current Data

| id | first | last | amt | city | ... |
|-------|---------|---------|---------|-----------|-----|
| 11894 | Jane | Roberts | 1255.76 | San Jose | |
| 90083 | Rajesh | Gidwani | 504.12 | Edinburgh | |
| 16884 | Hanna | Park | 75.99 | Hamburg | |
| 66792 | Ankur | Dawar | 446.90 | Madison | |
| 44872 | Audrey | Chan | 25.53 | New York | |
| 22936 | James | Greene | 2956.22 | New York | |
| 00481 | Michael | Chau | 1125.99 | Hong Kong | |



Result Table

| id | fScore | flag | iScore |
|-------|--------|------|--------|
| 16684 | 60.2 | loc | 60.2 |
| 66792 | 65.9 | amt | 44.2 |
| 00481 | 65.9 | loc | 21.7 |

Understanding Structured Streaming

| id | first | last | amt | city | ... |
|-----------|--------------|-------------|------------|-------------|-----|
| 11894 | Jane | Roberts | 1255.76 | San Jose | |
| 90083 | Rajesh | Gidwani | 504.12 | Edinburgh | |
| 16884 | Hanna | Park | 75.99 | Hamburg | |
| 66792 | Ankur | Dawar | 446.90 | Madison | |
| 44872 | Audrey | Chan | 25.53 | New York | |
| 22936 | James | Greene | 2956.22 | New York | |
| 00481 | Michael | Chau | 1125.99 | Hong Kong | |

44207 Sara Donner 355.37 Madrid



30927 Maria Domingo 556.22 El Paso



Understanding Structured Streaming

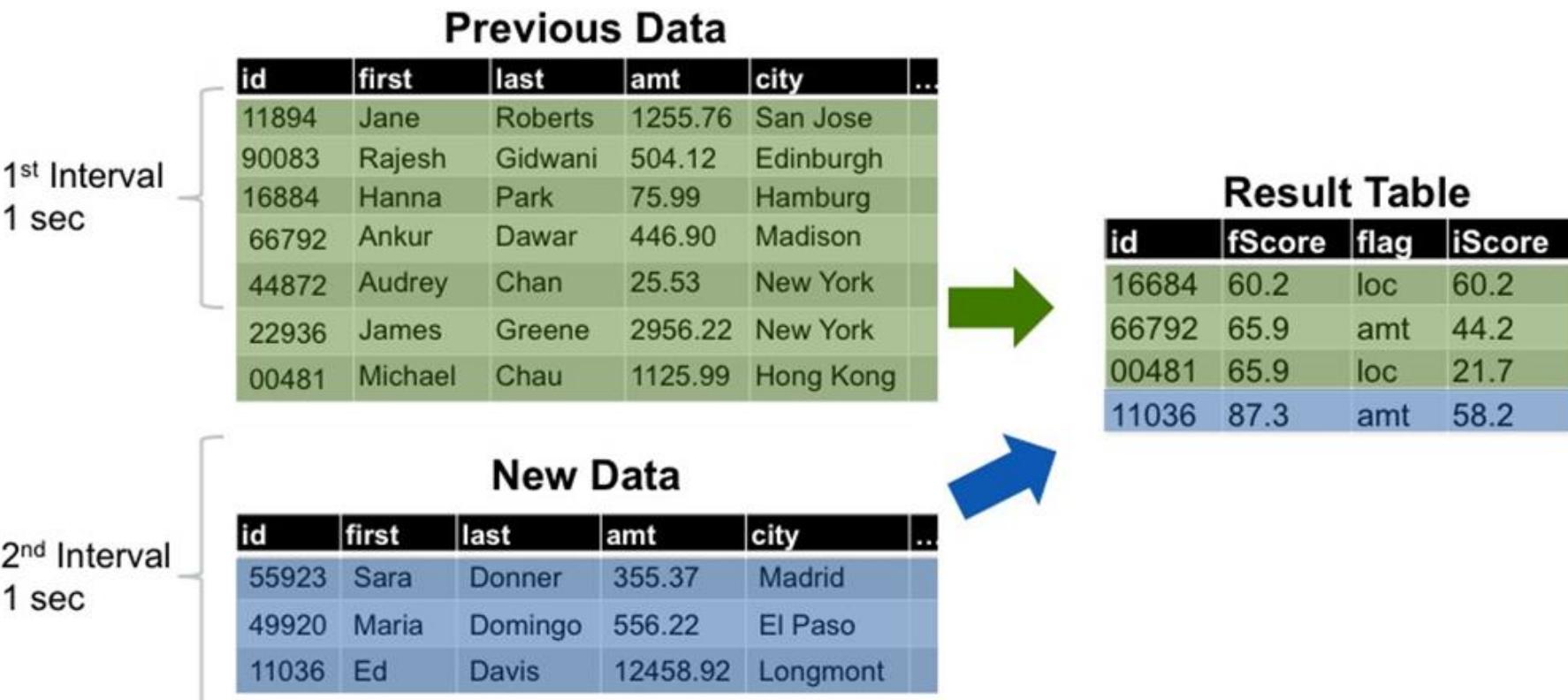
Previous Data

New Data

| id | first | last | amt | city | ... |
|-----------|--------------|-------------|------------|-------------|------------|
| 11894 | Jane | Roberts | 1255.76 | San Jose | |
| 90083 | Rajesh | Gidwani | 504.12 | Edinburgh | |
| 16884 | Hanna | Park | 75.99 | Hamburg | |
| 66792 | Ankur | Dawar | 446.90 | Madison | |
| 44872 | Audrey | Chan | 25.53 | New York | |
| 22936 | James | Greene | 2956.22 | New York | |
| 00481 | Michael | Chau | 1125.99 | Hong Kong | |
| 55923 | Sara | Donner | 355.37 | Madrid | |
| 49920 | Maria | Domingo | 556.22 | El Paso | |
| 11036 | Ed | Davis | 12458.92 | Longmont | |

Incremental Query

Understanding Structured Streaming



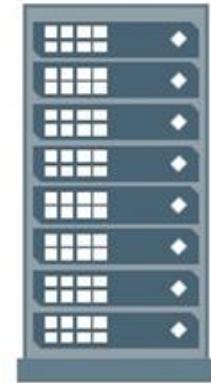
Save Output to Storage

Save output as:

- Complete
- Append
- Update

Result Table

| id | fScore | flag | iScore |
|-----------|---------------|-------------|---------------|
| 16684 | 60.2 | loc | 60.2 |
| 66792 | 65.9 | amt | 44.2 |
| 00481 | 65.9 | loc | 21.7 |
| 11036 | 87.3 | amt | 58.2 |



Output: Complete Mode

The entire Result Table
is written to external
storage each interval

(No fraud detected at Time Interval 1)

Time Interval 2 Time Interval 3

Input Table

| id | amount | city |
|-----------|---------------|-------------|
| 90083 | 504.12 | Edinburgh |
| 16884 | 75.99 | Hamburg |
| 66792 | 446.90 | Madison |

| id | amount | city |
|-----------|---------------|-------------|
| 90083 | 504.12 | Edinburgh |
| 16884 | 75.99 | Hamburg |
| 66792 | 446.90 | Madison |
| 00481 | 1125.99 | Hong Kong |

new row

Result Table

| id | fScore | flag | iScore |
|-----------|---------------|-------------|---------------|
| 16684 | 60.2 | loc | 60.2 |
| 66792 | 446.90 | amt | 44.2 |

| id | fScore | flag | iScore |
|-----------|---------------|-------------|---------------|
| 16684 | 70.1 | loc | 60.2 |
| 66792 | 65.9 | amt | 44.2 |
| 00481 | 65.9 | loc | 21.7 |

updated row

Output Saved

| id | fScore | flag | iScore |
|-----------|---------------|-------------|---------------|
| 16684 | 60.2 | loc | 60.2 |
| 66792 | 446.90 | amt | 55.2 |

| id | fScore | flag | iScore |
|-----------|---------------|-------------|---------------|
| 16684 | 70.1 | loc | 60.2 |
| 66792 | 65.9 | amt | 44.2 |
| 00481 | 65.9 | loc | 21.7 |

Output: Append Mode

Only new rows in the result table are written each interval

(No fraud detected at Time Interval 1)

Input Table

| id | amount | city |
|-----------|---------------|-------------|
| 90083 | 504.12 | Edinburgh |
| 16884 | 75.99 | Hamburg |
| 66792 | 446.90 | Madison |

Time Interval 2

Time Interval 3

| id | amount | city |
|-----------|---------------|-------------|
| 90083 | 504.12 | Edinburgh |
| 16884 | 75.99 | Hamburg |
| 66792 | 446.90 | Madison |
| 00481 | 1125.99 | Hong Kong |

new row

Result Table

| id | fScore | flag | iScore |
|-----------|---------------|-------------|---------------|
| 16684 | 60.2 | loc | 60.2 |
| 66792 | 446.90 | amt | 44.2 |

| id | fScore | flag | iScore |
|-----------|---------------|-------------|---------------|
| 16684 | 70.1 | loc | 60.2 |
| 66792 | 65.9 | amt | 44.2 |
| 00481 | 65.9 | loc | 21.7 |

updated row

Output Saved

| id | fScore | flag | iScore |
|-----------|---------------|-------------|---------------|
| 16684 | 60.2 | loc | 60.2 |
| 66792 | 446.90 | amt | 55.2 |

| id | fScore | flag | iScore |
|-----------|---------------|-------------|---------------|
| 00481 | 65.9 | loc | 21.7 |

Output: Update Mode

Updated rows in the result table are written each interval (new rows are considered updated)

(No fraud detected at Time Interval 1)

Input Table

| id | amount | city |
|-----------|---------------|-------------|
| 90083 | 504.12 | Edinburgh |
| 16884 | 75.99 | Hamburg |
| 66792 | 446.90 | Madison |

Time Interval 2

Time Interval 3

| id | amount | city |
|-----------|---------------|-------------|
| 90083 | 504.12 | Edinburgh |
| 16884 | 75.99 | Hamburg |
| 66792 | 446.90 | Madison |
| 00481 | 1125.99 | Hong Kong |

new row

Result Table

| id | fScore | flag | iScore |
|-----------|---------------|-------------|---------------|
| 16684 | 60.2 | loc | 60.2 |
| 66792 | 446.90 | amt | 44.2 |

| id | fScore | flag | iScore |
|-----------|---------------|-------------|---------------|
| 16684 | 70.1 | loc | 60.2 |
| 66792 | 446.90 | amt | 44.2 |
| 00481 | 65.9 | loc | 21.7 |

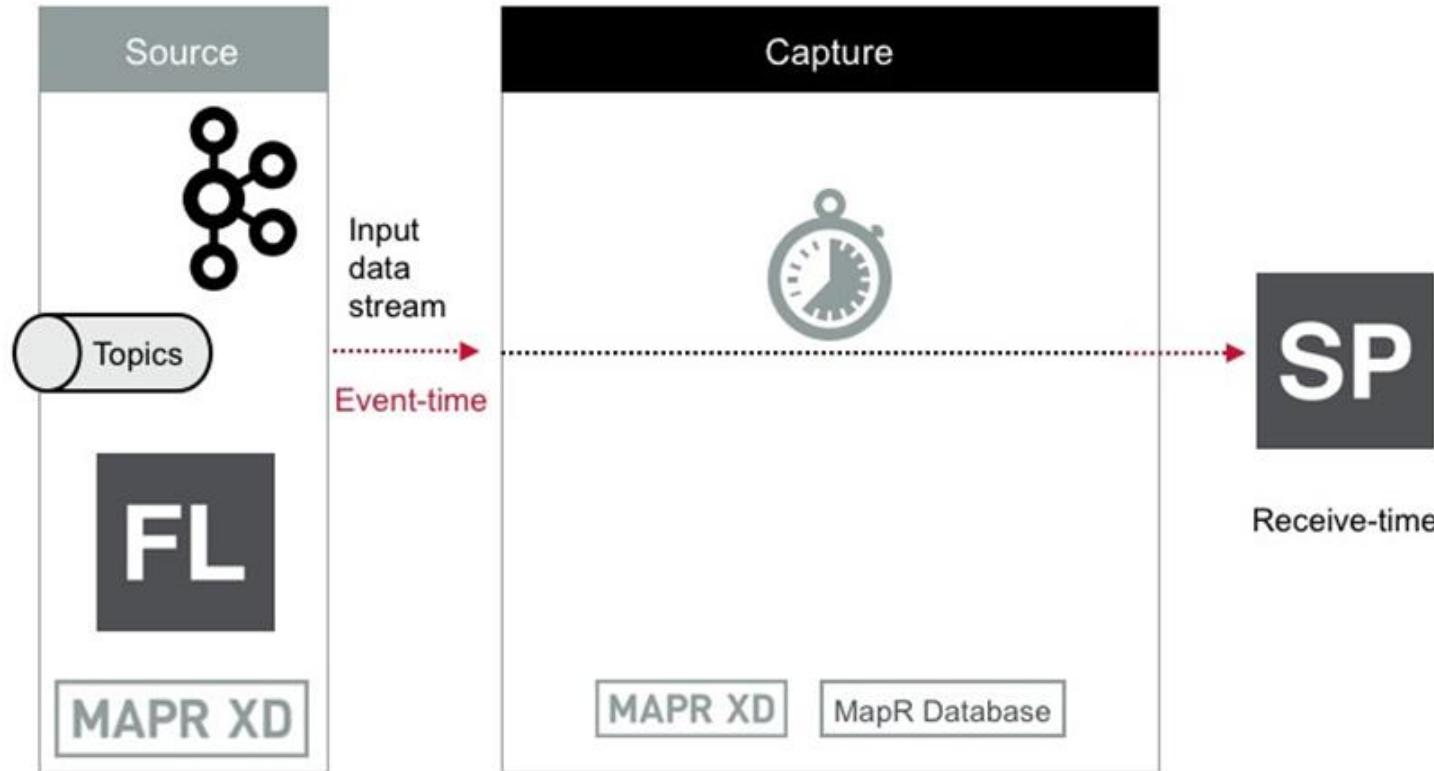
updated row

Output Saved

| id | fScore | flag | iScore |
|-----------|---------------|-------------|---------------|
| 16684 | 60.2 | loc | 60.2 |
| 66792 | 446.90 | amt | 55.2 |

| id | fScore | flag | iScore |
|-----------|---------------|-------------|---------------|
| 16684 | 70.1 | loc | 60.2 |
| 00481 | 65.9 | loc | 21.7 |

Handling Event-time



Handling Late Data

| id | first | last | amt | city | ... |
|-----------|--------------|-------------|------------|-------------|------------|
| 11894 | Jane | Roberts | 1255.76 | San Jose | |
| 90083 | Rajesh | Gidwani | 504.12 | Edinburgh | |
| 16884 | Hanna | Park | 75.99 | Hamburg | |

| id | first | last | amt | city | ... |
|-----------|--------------|-------------|------------|-------------|------------|
| 11894 | Jane | Roberts | 1255.76 | San Jose | |
| 90083 | Rajesh | Gidwani | 526.40 | Edinburgh | |
| 16884 | Hanna | Park | 75.99 | Hamburg | |
| 66792 | Ankur | Dawar | 446.90 | Madison | |
| 44872 | Audrey | Chan | 25.53 | New York | |
| 22936 | James | Greene | 2956.22 | New York | |
| 00481 | Michael | Chau | 1125.99 | Hong Kong | |

Streaming DataFrames and Streaming Datasets

```
val spark =  
SparkSession.builder.appName("SensorData").getOrCreate()  
  
val sensorCsvDF = spark.readStream ("sep", ",")  
.schema(userSchema) // Specify schema of the csv files  
.csv("/path/to/directory") // Equivalent to format("csv")
```

Knowledge Check



Which of these is not a mode of defining the output?

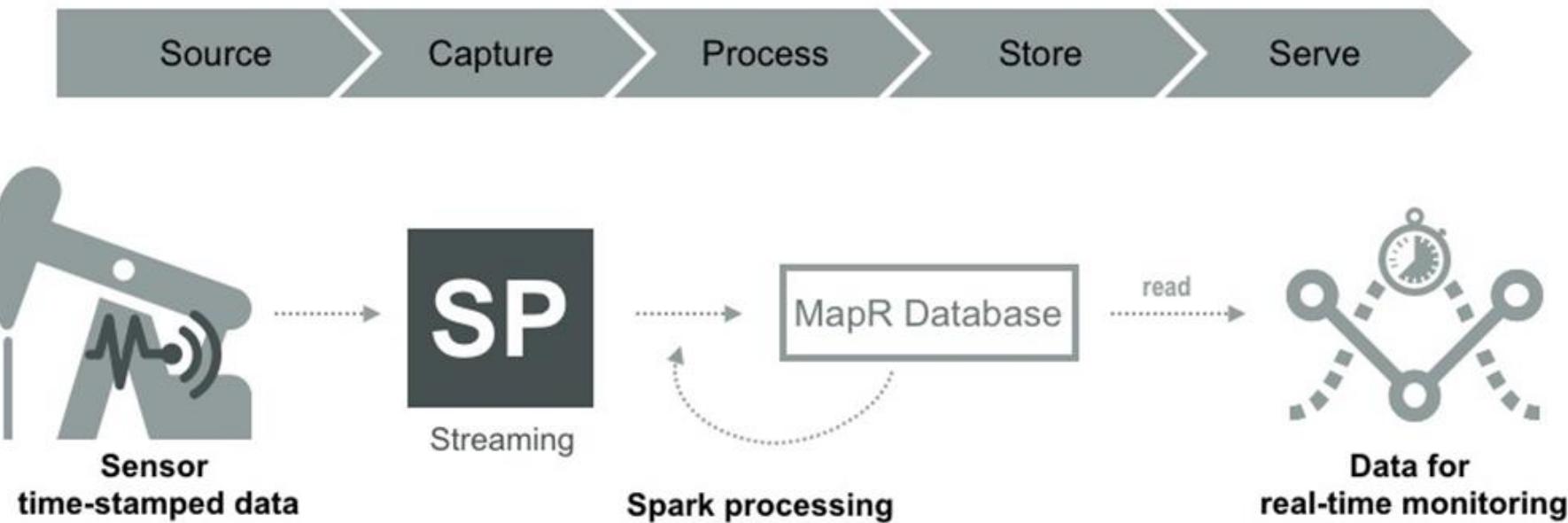
- A. Append
- B. Complete
- C. Overwrite
- D. Update



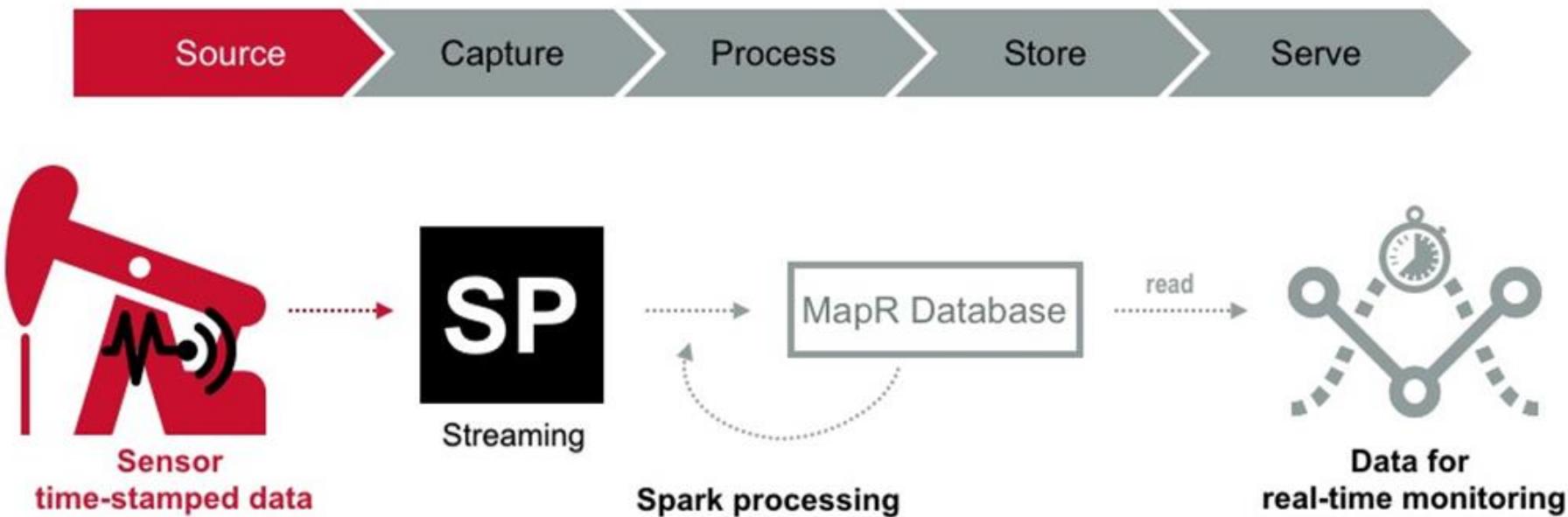
Learning Goals

- 6.1 Describe Spark Streaming Architecture
- 6.2 **Create a Spark Structured Streaming Application**
 - Define Use Case
 - Basic Steps and Save Data to Parquet Tables
- 6.3 Apply Operations on Streaming DataFrames
- 6.4 Define Windowed Operations
- 6.5 Describe How Streaming Applications are Fault Tolerant

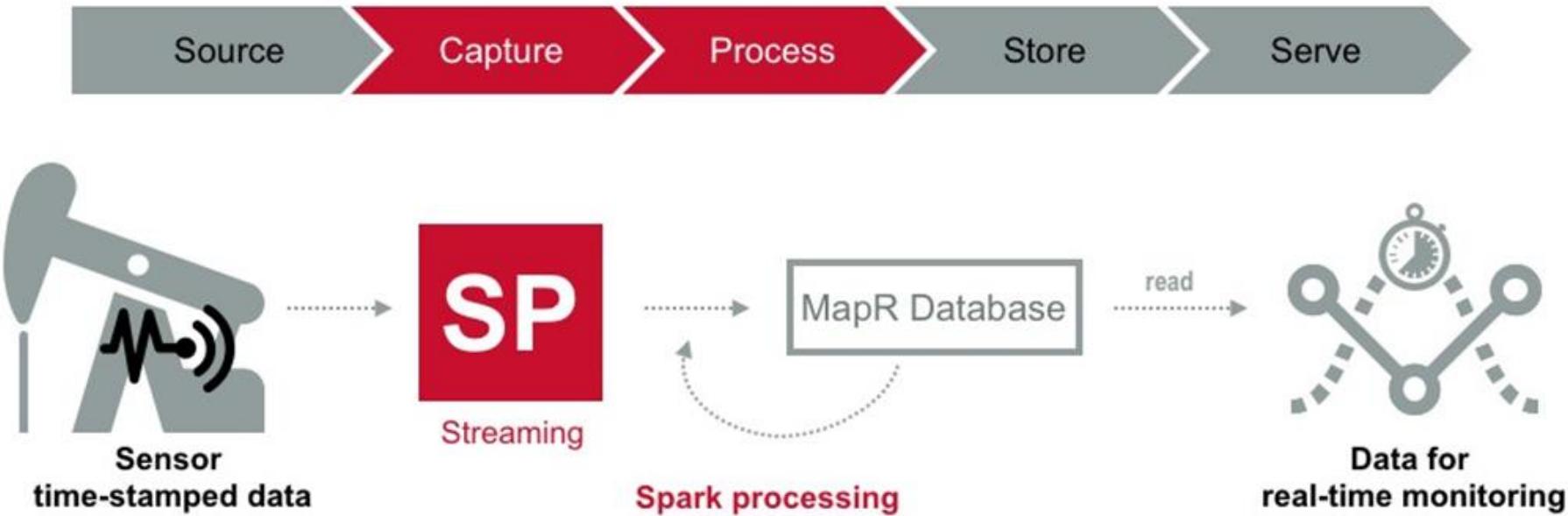
Use Case: Time Series Data



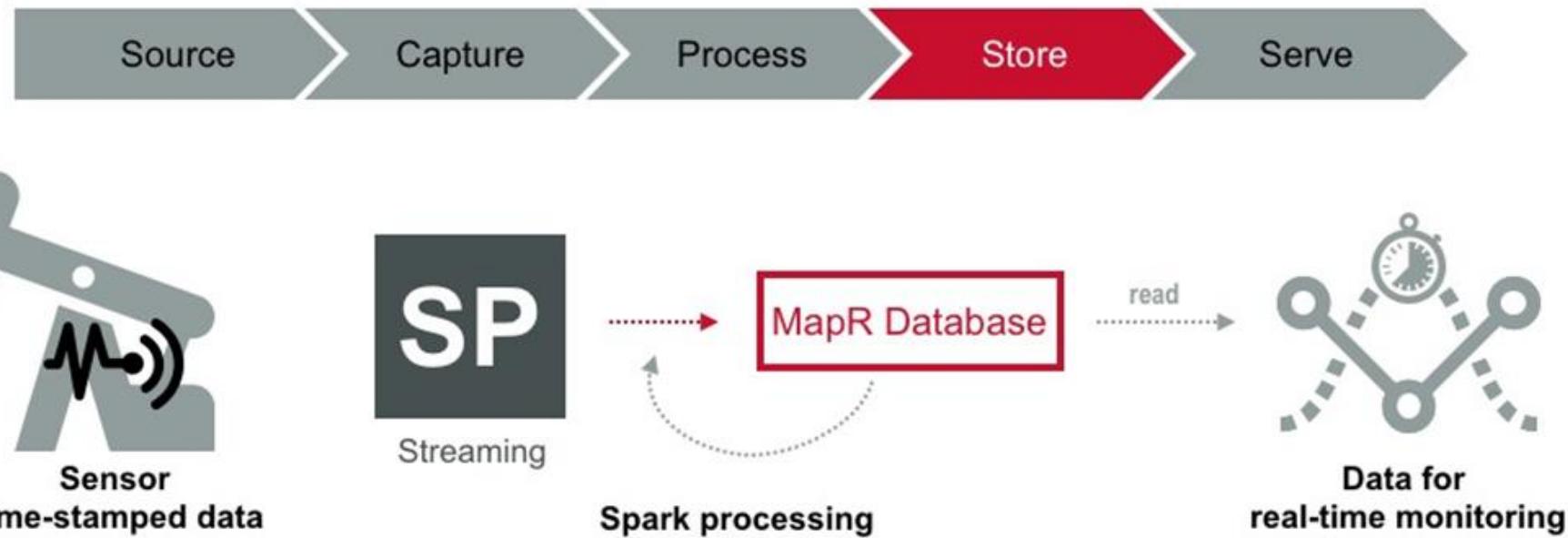
Use Case: Time Series Data



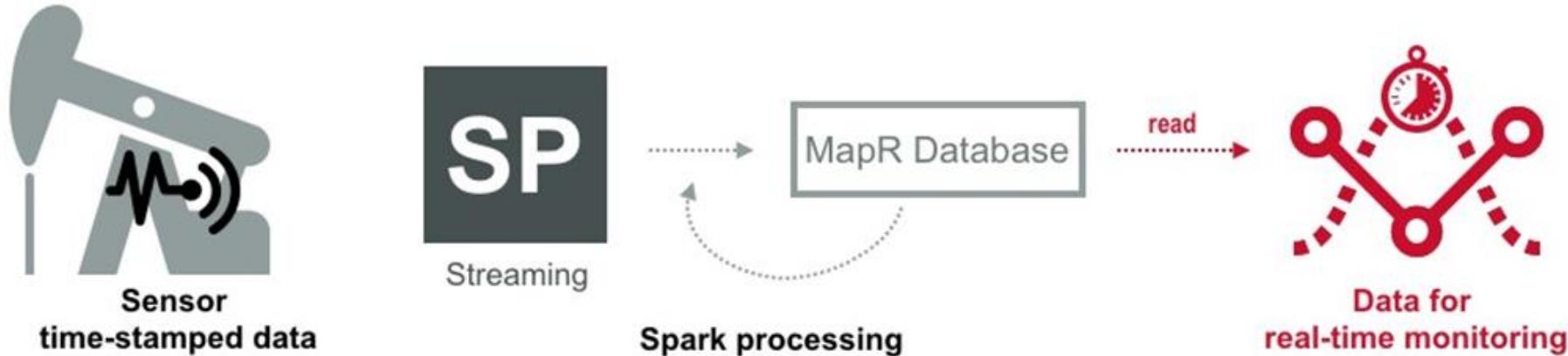
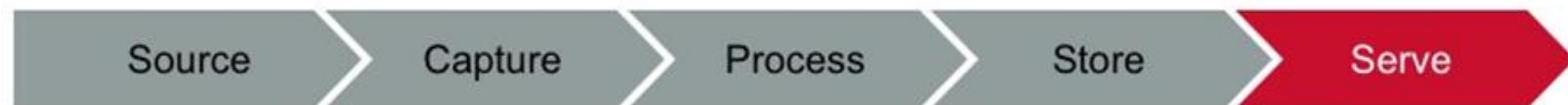
Use Case: Time Series Data



Use Case: Time Series Data



Use Case: Time Series Data



Convert Line of CSV Data to Sensor Object

```
val userSchema = new StructType().add("resid", "string").add("date",  
"string").add("time", "string").add("hz", "double").add("disp",  
"double").add("flow", "double").add("sendPPM", "double").add("psi",  
"double").add("chlppm", "double")
```

sensordata.csv ×

| | ResourceID | Date | Time | HZ | Displace | Flow | SedimentPPM | PressureLbs | ChlorinePPM |
|---|------------|---------|------|-------|----------|------|-------------|-------------|-------------|
| 1 | COHUTTA | 3/10/14 | 1:01 | 10.27 | 1.73 | 881 | 1.56 | 85 | 1.94 |
| 2 | COHUTTA | 3/10/14 | 1:02 | 9.67 | 1.731 | 882 | 0.52 | 87 | 1.79 |
| 3 | COHUTTA | 3/10/14 | 1:03 | 10.47 | 1.732 | 882 | 1.7 | 92 | 0.66 |



Learning Goals

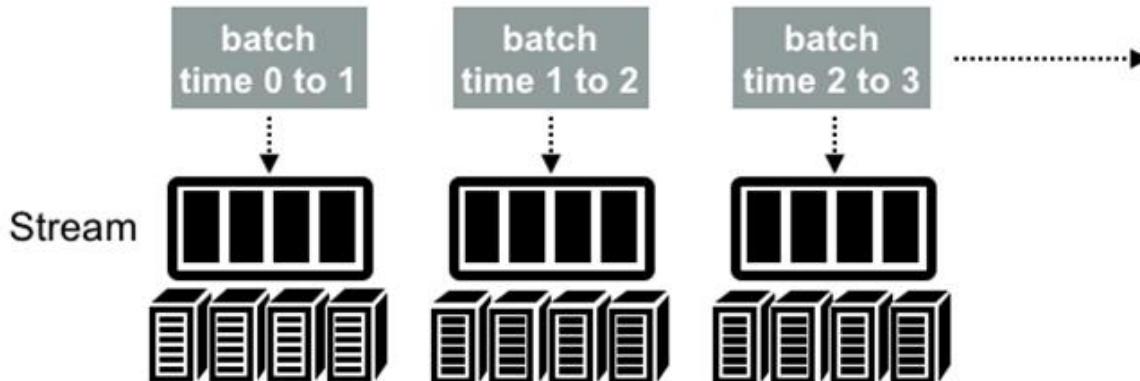
- 6.1 Describe Spark Streaming Architecture
- 6.2 Create a Spark Structured Streaming Application
 - Define Use Case
 - **Basic Steps and Save Data to Parquet Tables**
- 6.3 Apply Operations on Streaming DataFrames
- 6.4 Define Windowed Operations
- 6.5 Describe How Streaming Applications are Fault Tolerant

Basic Steps for Spark Streaming Code

1. Initialize a Spark StreamingContext object
2. Using context, create a DStream
 - Represents streaming data from a source
3. Apply transformations and/or output operations to DStreams
4. Receive and process data
 - Use `streamingContext.start()`
5. Wait for the processing to be stopped
 - Use `streamingContext.awaitTermination()`

Create a Streaming DataFrame

```
val spark = SparkSession.builder.appName("SensorData").getOrCreate()
```



Create a Streaming DataFrame

```
val spark = SparkSession.builder.appName("SensorData").getOrCreate()

val userSchema = new StructType().add("resid", "string").add("date",
"string").add("time", "string").add("hz", "double").add("disp",
"double").add("flow", "double").add("sendPPM", "double").add("psi",
"double").add("chlppm", "double")
```

Create a Streaming DataFrame

```
val spark = SparkSession.builder.appName("SensorData").getOrCreate()

val userSchema = new StructType().add("resid", "string").add("date",
"string").add("time", "string").add("hz", "double").add("disp",
"double").add("flow", "double").add("sendPPM", "double").add("psi",
"double").add("chlppm", "double")

val sensorCsvDF = spark.readStream.option("sep", ",")
.schema(userSchema) // Specify schema of the csv files
.csv("/path/to/directory") // Equivalent to format("csv")
```

Process Streaming DataFrame

```
val spark = SparkSession.builder.appName("SensorData").getOrCreate()

val userSchema = new StructType().add("resid", "string").add("date",
"string").add("time", "string").add("hz", "double").add("disp",
"double").add("flow", "double").add("sendPPM", "double").add("psi",
"double").add("chlppm", "double")

val sensorCsvDF = spark.readStream.option("sep", ",")
.schema(userSchema) // Specify schema of the csv files
.csv("/path/to/directory") // Equivalent to format("csv")

// filter sensor data for low psi
sensorCsvDF.filter(_.psi < 5.0)
```

Wait for Termination

```
val spark = SparkSession.builder.appName("SensorData").getOrCreate()

val userSchema = new StructType().add("resid", "string").add("date",
"string").add("time", "string").add("hz", "double").add("disp",
"double").add("flow", "double").add("sendPPM", "double").add("psi",
"double").add("chlppm", "double")

val sensorCsvDF = spark.readStream.option("sep", ",")
.schema(userSchema) // Specify schema of the csv files
.csv("/path/to/directory") // Equivalent to format("csv")

// filter sensor data for low psi
sensorCsvDF.filter(_.psi < 5.0)

// Wait for the computation to terminate
ssc.awaitTermination()
```

Streaming Application Output

```
-----
Time: 1452886488000 ms

-----
COHUTTA,3/10/14,1:01,10.27,1.73,881,1.56,85,1.94
COHUTTA,3/10/14,1:02,9.67,1.731,882,0.52,87,1.79
COHUTTA,3/10/14,1:03,10.47,1.732,882,1.7,92,0.66
COHUTTA,3/10/14,1:05,9.56,1.734,883,1.35,99,0.68
COHUTTA,3/10/14,1:06,9.74,1.736,884,1.27,92,0.73
COHUTTA,3/10/14,1:08,10.44,1.737,885,1.34,93,1.54
COHUTTA,3/10/14,1:09,9.83,1.738,885,0.06,76,1.44
COHUTTA,3/10/14,1:11,10.49,1.739,886,1.51,81,1.83
COHUTTA,3/10/14,1:12,9.79,1.739,886,1.74,82,1.91
COHUTTA,3/10/14,1:13,10.02,1.739,886,1.24,86,1.79
...
low pressure alert
Sensor(NANTAHALLA,3/13/14,2:05,0.0,0.0,0.0,1.73,0.0,1.51)
Sensor(NANTAHALLA,3/13/14,2:07,0.0,0.0,0.0,1.21,0.0,1.51)

-----
Time: 1452886490000 ms
-----
```

Save Data to Parquet

```
noAggDF.writeStream  
.format("parquet")  
.option("checkpointLocation", "/path/to/checkpoint/dir")  
.option("path", "/path/to/destination/dir")  
.start()
```

Knowledge Check



Referring to the Spark Streaming code shown here:

```
val sensorCsvDF = spark.readStream.option("sep", ",")  
  .schema(userSchema) // Specify schema of the csv files  
  .csv("maprfs:///tmp/sensorData") // Equivalent to format("csv")
```

- What is the streaming DataFrame object?
- What method is used to create a streaming DataFrame?
- Where is the streaming data being read from?

Knowledge Check



Referring to the Spark Streaming code shown here:

```
val sensorCsvDF = spark.readStream.option("sep", ",")  
  .schema(userSchema) // Specify schema of the csv files  
  .csv("maprfs:///tmp/sensorData") // Equivalent to format("csv")
```

- What is the streaming DataFrame object? **sensorCsvDF**
- What method is used to create a streaming DataFrame? **spark.readStream()**
- Where is the streaming data being read from? **maprfs:///tmp/sensorData**



Learning Goals

- 6.1 Describe Spark Streaming Architecture
- 6.2 Create a Spark Structured Streaming Application
 - Define Use Case
 - Basic Steps and Save Data to Parquet Tables
- 6.3 **Apply Operations on Streaming DataFrames**
- 6.4 Define Windowed Operations
- 6.5 Describe How Streaming Applications are Fault Tolerant

Operations on Streaming DataFrames

- What is the maximum, minimum, and average for sensor attributes?
- What is the pump vendor and maintenance information for sensors with low pressure alerts?



DataFrame and SQL Operations

```
sensorCsvDF.createOrReplaceTempView("sensor")
val res = spark.sql( "SELECT resid, date,
    max(hz) as maxhz, min(hz) as minhz, avg(hz) as avghz,
    max(disp) as maxdisp, min(disp) as mindisp, avg(disp) as avgdisp,
    max(flo) as maxflo, min(flo) as minflo, avg(flo) as avgflo,
    max(psi) as maxpsi, min(psi) as minpsi, avg(psi) as avgpsi
    FROM sensor GROUP BY resid,date")
res.show()
```

Streaming Application Output

| sensor max, min, averages | | | | | | | | | | | | |
|---------------------------|---------|-------|-------|--------------------|---------|---------|--------------------|--------|--------|--------------------|--|--|
| resid | date | maxhz | minhz | avghz | maxdisp | mindisp | avgdisp | maxflo | minflo | avgflo | | |
| LAGNAPPE | 3/12/14 | 10.5 | 9.5 | 9.988799582463459 | 3.235 | 1.623 | 2.4714206680584563 | 1570.0 | 788.0 | 1199.1022964509395 | | |
| CHER | 3/13/14 | 10.5 | 9.5 | 10.005271398747391 | 3.552 | 1.857 | 2.732156576200417 | 1670.0 | 873.0 | 1284.660751565762 | | |
| BBKING | 3/13/14 | 10.5 | 9.5 | 9.996033402922755 | 1.58 | 0.902 | 1.26099164926931 | 1822.0 | 1041.0 | 1454.0240083507306 | | |
| MOJO | 3/12/14 | 10.5 | 9.5 | 10.006482254697294 | 3.589 | 2.143 | 2.888004175365341 | 1899.0 | 1134.0 | 1528.2724425887266 | | |
| CARGO | 3/10/14 | 10.5 | 9.5 | 9.998507306889353 | 3.752 | 1.903 | 2.8525417536534423 | 1533.0 | 778.0 | 1165.5302713987473 | | |
| LAGNAPPE | 3/11/14 | 10.5 | 9.5 | 10.009540709812102 | 3.092 | 1.454 | 2.307491649269313 | 1500.0 | 706.0 | 1119.5647181628392 | | |
| CHER | 3/12/14 | 10.5 | 9.5 | 10.008256784968692 | 3.443 | 1.728 | 2.6184937369519825 | 1619.0 | 812.0 | 1231.230688935282 | | |
| BBKING | 3/12/14 | 10.5 | 9.5 | 10.006795407098123 | 1.556 | 0.862 | 1.2138110647181624 | 1794.0 | 994.0 | 1399.608559498956 | | |
| MOJO | 3/11/14 | 10.5 | 9.5 | 10.0029331941545 | 3.513 | 1.979 | 2.8009843423799587 | 1859.0 | 1047.0 | 1482.2160751565762 | | |
| LAGNAPPE | 3/10/14 | 10.5 | 9.5 | 10.00329853862213 | 3.116 | 1.453 | 2.349840292275576 | 1512.0 | 705.0 | 1140.1022964509395 | | |
| CHER | 3/11/14 | 10.5 | 9.5 | 9.990396659707717 | 3.325 | 1.691 | 2.545035490605431 | 1564.0 | 795.0 | 1196.6837160751566 | | |
| BBKING | 3/11/14 | 10.5 | 9.5 | 9.997348643006282 | 1.49 | 0.821 | 1.1701722338204592 | 1718.0 | 947.0 | 1349.2849686847599 | | |
| MOJO | 3/10/14 | 10.5 | 9.5 | 9.999457202505194 | 3.345 | 1.828 | 2.6188089770354868 | 1770.0 | 967.0 | 1385.8131524008352 | | |
| CHER | 3/10/14 | 10.5 | 9.5 | 9.998726513569954 | 3.172 | 1.653 | 2.4233079331941516 | 1492.0 | 777.0 | 1139.4488517745303 | | |
| BBKING | 3/10/14 | 10.5 | 9.5 | 10.001409185803748 | 1.502 | 0.821 | 1.18567223382046 | 1732.0 | 947.0 | 1367.1711899791233 | | |
| THERMALITO | 3/14/14 | 10.5 | 9.5 | 9.983862212943635 | 3.784 | 2.135 | 3.0065960334029254 | 1680.0 | 948.0 | 1335.1002087682673 | | |
| THERMALITO | 3/13/14 | 10.5 | 9.5 | 9.999311064718169 | 3.896 | 2.116 | 3.069195198329852 | 1730.0 | 940.0 | 1362.910229645094 | | |
| ANDOUILLE | 3/14/14 | 10.5 | 9.5 | 10.011388308977041 | 2.146 | 1.139 | 1.6609373695198306 | 1592.0 | 845.0 | 1232.2296450939457 | | |
| THERMALITO | 3/12/14 | 10.5 | 9.5 | 10.007526096033398 | 3.823 | 1.986 | 2.9294561586638808 | 1697.0 | 882.0 | 1300.8622129436326 | | |
| ANDOUILLE | 3/13/14 | 10.5 | 9.5 | 9.99043841336118 | 2.174 | 1.104 | 1.656331941544886 | 1613.0 | 819.0 | 1228.8371607515658 | | |

DataFrame and SQL Operations

```
// put pump vendor and maintenance data in temp table
spark.sparkContext.textFile("vendor.csv").map(parsePump.toDF()).createOrReplaceTempView("pump")
spark.sparkContext("maint.csv").map(parseMaint).toDF().createOrReplaceTempView("maint")
// 
sensorCsvDF.filter(_.psi < 5.0).().createOrReplaceTempView("alert")

val alertpumpmaint = sqlContext.sql("select s.resid, s.date, s.psi,
    p.pumpType, p.vendor, m.eventDate, m.technician from alert s join
    pump p on s.resid = p.resid join maint m on p.resid=m.resid")

alertpumpmaint.show()
```

Streaming Application Output

| resid | date | psi | pumpType | purchaseDate | serviceDate | vendor | eventDate | technician | description |
|----------|---------|-----|-----------|--------------|-------------|---------|-----------|------------|----------------------|
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 10/2/09 | T. LaBou | Install |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 10/5/09 | W.Stevens | Inspect |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 11/24/09 | W.Stevens | Tighten Mounts |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 6/10/10 | T. LaBou | Inspect |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 1/7/11 | T. LaBou | Inspect |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 9/30/11 | W.Stevens | Inspect |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 10/3/11 | T. LaBou | Bearing Seal |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 11/5/11 | D.Pitre | Inspect |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 5/22/12 | D.Pitre | Inspect |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 12/15/12 | W.Stevens | Inspect |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 6/18/13 | T. LaBou | Vane clearance ad... |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 7/11/13 | W.Stevens | Inspect |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 2/5/14 | D.Pitre | Inspect |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 3/14/14 | D.Pitre | Shutdown Main Fee... |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 10/2/09 | T. LaBou | Install |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 10/5/09 | W.Stevens | Inspect |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 11/24/09 | W.Stevens | Tighten Mounts |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 6/10/10 | T. LaBou | Inspect |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 1/7/11 | T. LaBou | Inspect |
| LAGNAPPE | 3/14/14 | 0.0 | HYDROPUMP | 5/25/08 | 9/26/09 | GENPUMP | 9/30/11 | W.Stevens | Inspect |

Time: 1452887522000 ms

Knowledge Check



Indicate whether the statements below are TRUE or FALSE.

- A. Transformations on a streaming DataFrame return another streaming DataFrame
- B. Transformations trigger computations
- C. Streaming DataFrames can only be made from streaming sources



Learning Goals

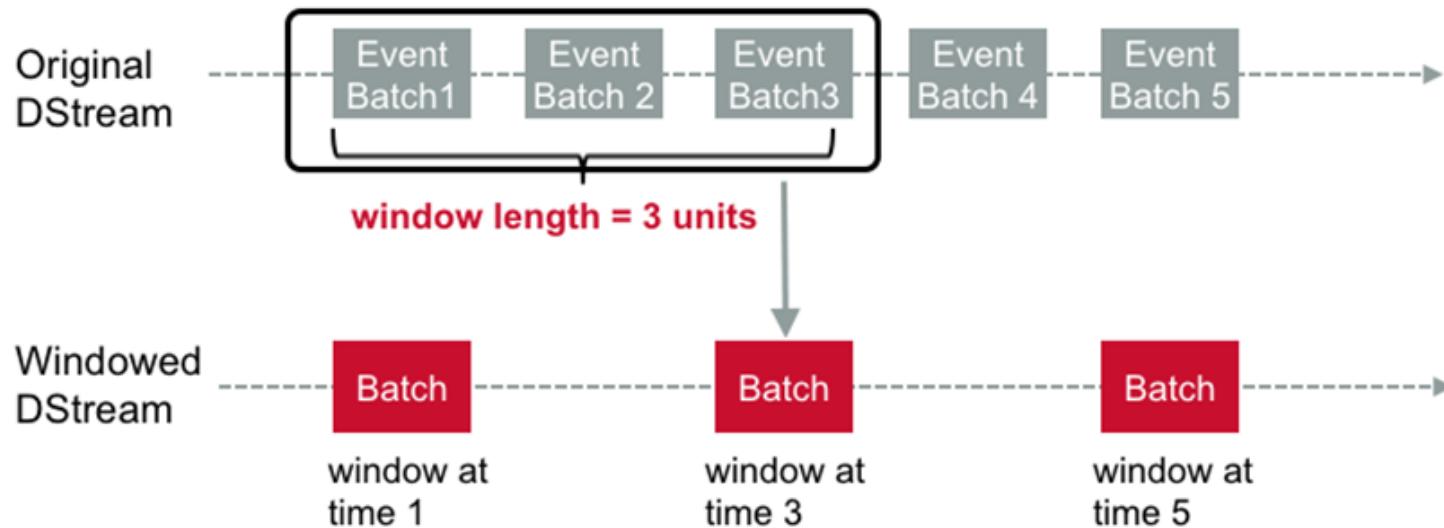
- 6.1 Describe Spark Streaming Architecture
- 6.2 Create a Spark Structured Streaming Application
 - Define Use Case
 - Basic Steps and Save Data to Parquet Tables
- 6.3 Apply Operations on Streaming DataFrames
- 6.4 Define Windowed Operations**
- 6.5 Describe How Streaming Applications are Fault Tolerant

Sliding Windows

- Can compute results across longer time period
- Example – want average oil pressure data
 - Create a sliding window of the last 60 seconds
 - Computed every 30 seconds

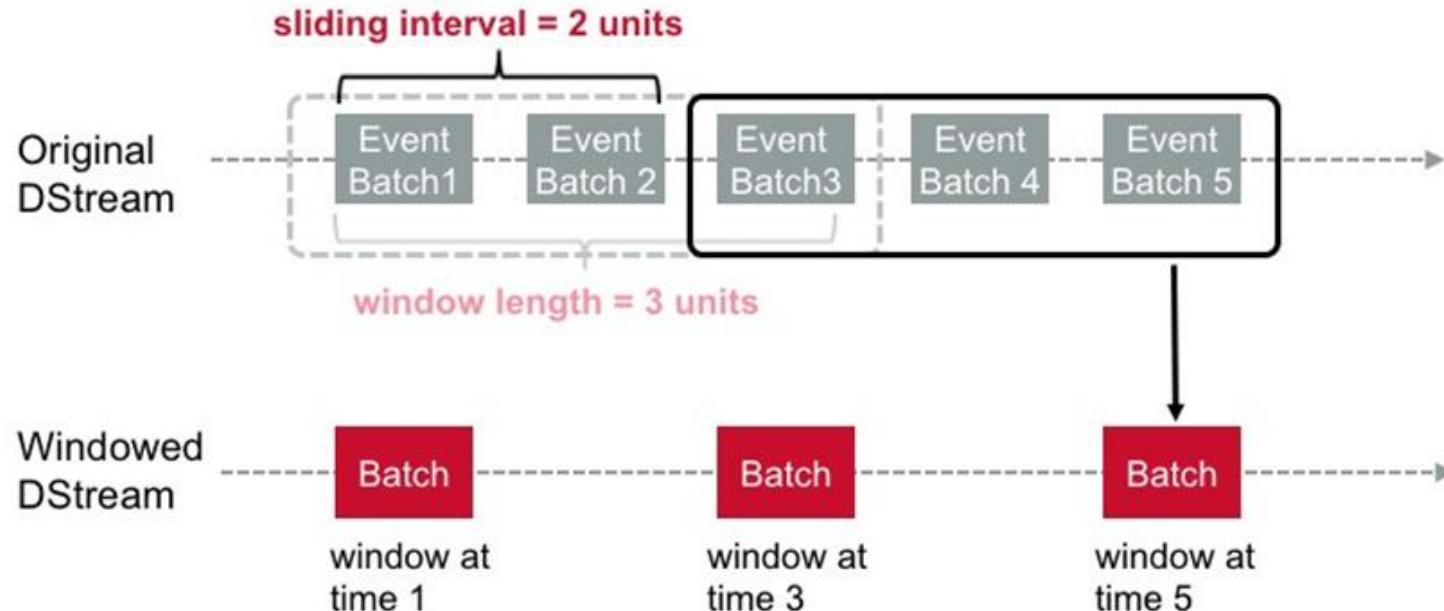
Windowed Computations

batch interval = 1 second



Windowed Computations

batch interval = 1 second



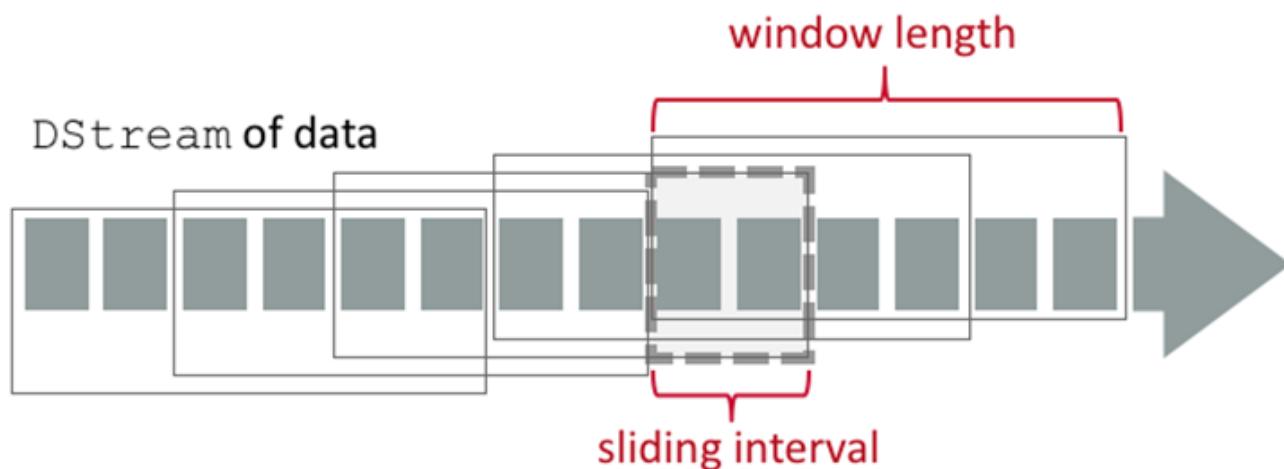
Window-based Transformations

```
val windowedWordCounts = pairs.reduceByKeyAndWindow(  
  (a:Int,b:Int) => (a + b), Seconds(12), Seconds(4))
```

sliding window operation

window length

sliding interval



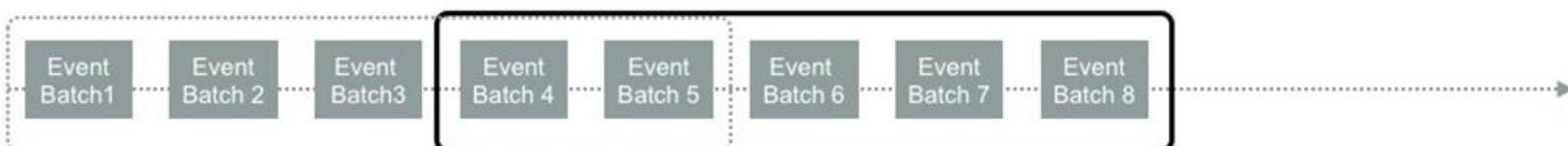
Window Operations

| Output Operation | Description |
|--|--|
| <code>window(windowLength, slideInterval)</code> | Returns new DStream computed based on windowed batches of source DStream. |
| <code>countByWindow(windowLength, slideInterval)</code> | Returns a sliding window count of elements in the stream. |
| <code>reduceByWindow(func, windowLength, slideInterval)</code> | Returns a new single-element stream created by aggregating elements over sliding interval using <code>func</code> . |
| <code>reduceByKeyAndWindow(func, windowLength, slideInterval, [numTasks])</code> | Returns a new DStream of (K,V) pairs from DStream of (K,V) pairs; aggregates using given reduce function <code>func</code> over batches of sliding window. |
| <code>countByValueAndWindow(windowLength, slideInterval, [numTasks])</code> | Returns new DStream of (K,V) pairs where value of each key is its frequency within a sliding window; it acts on DStreams of (K,V) pairs. |

Window Operations on DStreams

With a windowed stream:

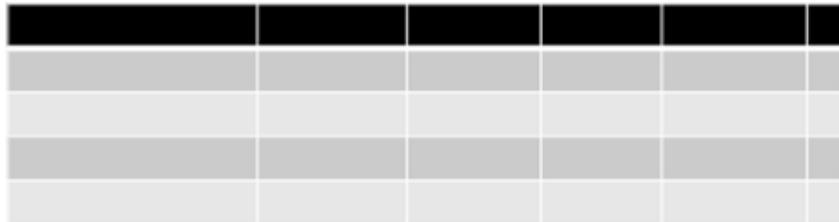
- What is the count of sensor events by pump ID?
- What is the Max, Min, and Average for PSI?



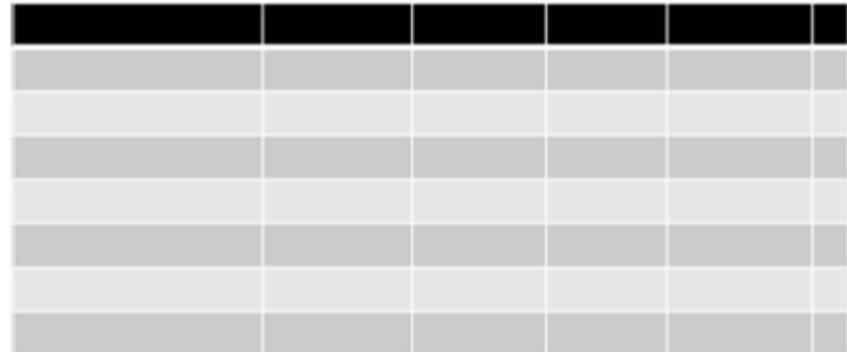
Handling Event-time



Window 1: 1-5 min



Window 2: 4-8 min



DataFrame and SQL Operations

```
//window of 5 minutes of data every 3 minutes.

val res = sensorCsvDF.groupBy(window($"timestamp", "5 minutes", "3
minutes"), $"resid", $"date").agg(count("resid").alias("total"))

res.show
```

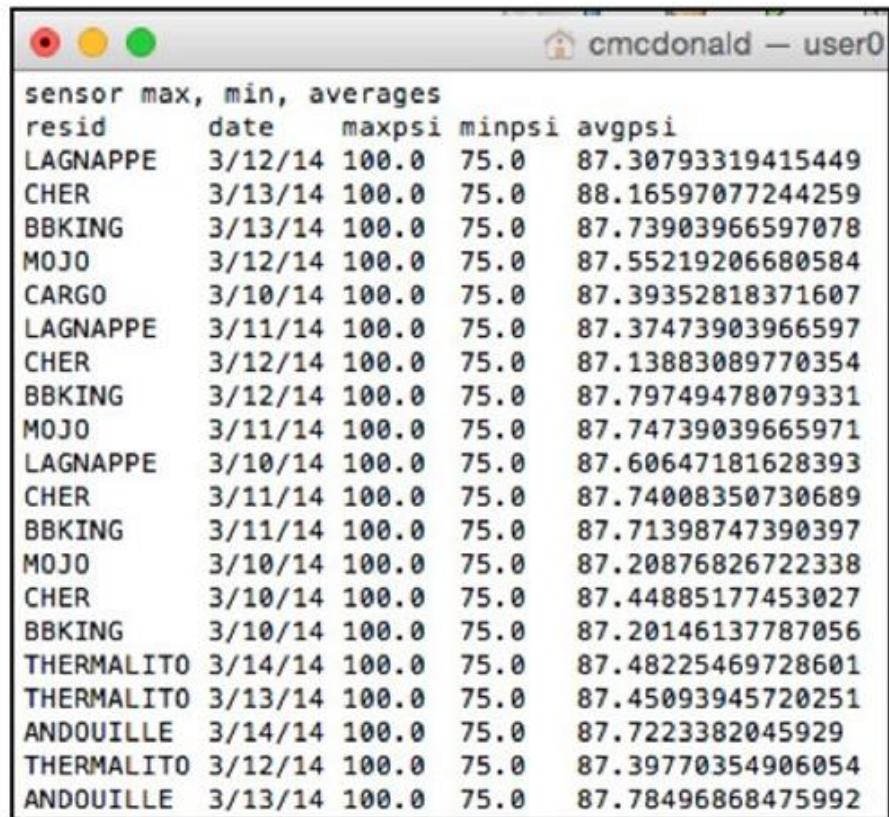
Streaming Application Output

```
sensor count
resid      date    total
LAGNAPPE   3/12/14 958
CHER       3/13/14 958
BBKING     3/13/14 958
MOJO       3/12/14 958
CARGO      3/10/14 958
LAGNAPPE   3/11/14 958
CHER       3/12/14 958
BBKING     3/12/14 958
MOJO       3/11/14 958
LAGNAPPE   3/10/14 958
CHER       3/11/14 958
BBKING     3/11/14 958
MOJO       3/10/14 958
CHER       3/10/14 958
BBKING     3/10/14 958
THERMALITO 3/14/14 958
THERMALITO 3/13/14 958
ANDOUILLE  3/14/14 958
THERMALITO 3/12/14 958
ANDOUILLE  3/13/14 958
```

DataFrame and SQL Operations

```
//window of 5 minutes of data every 3 minutes.  
val res = sensorCsvDF.groupBy(window($"timestamp", "5 minutes", "3  
minutes"), $"resid", $"date").agg(max("psi").alias("maxpsi"),  
min("psi").alias("minpsi"), avg("psi").alias("avgpsi"))  
  
res.show
```

Streaming Application Output



The screenshot shows a terminal window with a title bar 'cmcdonald — user0'. The window contains a table of sensor data with the following columns: resid, date, maxpsi, minpsi, and avgpsi. The data is sorted by resid and date.

| sensor max, min, averages | | | | |
|---------------------------|---------|--------|--------|-------------------|
| resid | date | maxpsi | minpsi | avgpsi |
| LAGNAPPE | 3/12/14 | 100.0 | 75.0 | 87.30793319415449 |
| CHER | 3/13/14 | 100.0 | 75.0 | 88.16597077244259 |
| BBKING | 3/13/14 | 100.0 | 75.0 | 87.73903966597078 |
| MOJO | 3/12/14 | 100.0 | 75.0 | 87.55219206680584 |
| CARGO | 3/10/14 | 100.0 | 75.0 | 87.39352818371607 |
| LAGNAPPE | 3/11/14 | 100.0 | 75.0 | 87.37473903966597 |
| CHER | 3/12/14 | 100.0 | 75.0 | 87.13883089770354 |
| BBKING | 3/12/14 | 100.0 | 75.0 | 87.79749478079331 |
| MOJO | 3/11/14 | 100.0 | 75.0 | 87.74739039665971 |
| LAGNAPPE | 3/10/14 | 100.0 | 75.0 | 87.60647181628393 |
| CHER | 3/11/14 | 100.0 | 75.0 | 87.74008350730689 |
| BBKING | 3/11/14 | 100.0 | 75.0 | 87.71398747390397 |
| MOJO | 3/10/14 | 100.0 | 75.0 | 87.20876826722338 |
| CHER | 3/10/14 | 100.0 | 75.0 | 87.44885177453027 |
| BBKING | 3/10/14 | 100.0 | 75.0 | 87.20146137787056 |
| THERMALITO | 3/14/14 | 100.0 | 75.0 | 87.48225469728601 |
| THERMALITO | 3/13/14 | 100.0 | 75.0 | 87.45093945720251 |
| ANDOUILLE | 3/14/14 | 100.0 | 75.0 | 87.7223382045929 |
| THERMALITO | 3/12/14 | 100.0 | 75.0 | 87.39770354906054 |
| ANDOUILLE | 3/13/14 | 100.0 | 75.0 | 87.78496868475992 |



Learning Goals

- 6.1 Describe Spark Streaming Architecture
- 6.2 Create a Spark Structured Streaming Application
 - Define Use Case
 - Basic Steps and Save Data to Parquet Tables
- 6.3 Apply Operations on Streaming DataFrames
- 6.4 Define Windowed Operations
- 6.5 **Describe How Streaming Applications are Fault Tolerant**

Write-Ahead Logs: Review

- All modifications are written to a log before being applied
- Redo and undo information is stored in the log



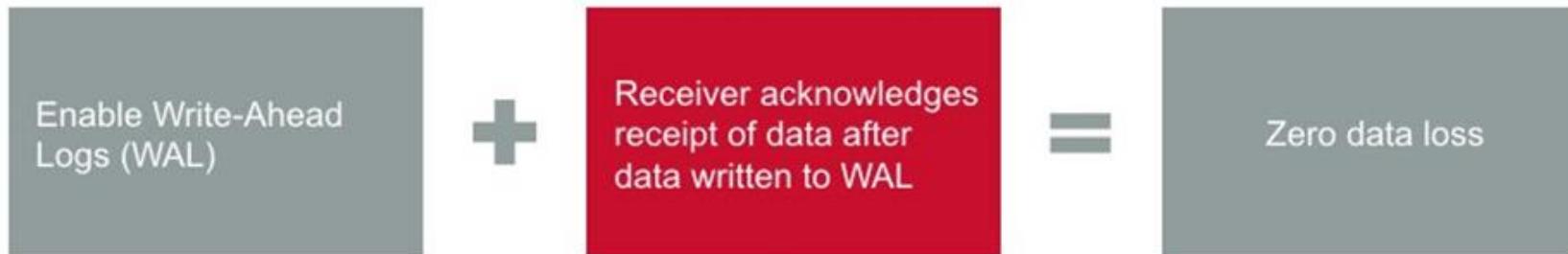
Fault Tolerance in Write-Ahead Logs

- Flume, MapR Streams, and Kafka use receivers
- Store received data in executor memory
- Driver runs tasks on executors



Fault Tolerance in Spark Write-Ahead Logs

- Flume, MapR Streams, and Kafka use receivers
- Store received data in executor memory
- Driver runs tasks on executors



Fault Tolerance in Spark Write-Ahead Logs

- Flume, MapR Streams, and Kafka use receivers
- Store received data in executor memory
- Driver runs tasks on executors



Checkpointing

- Provides fault tolerance for driver
- Periodically saves data to fault tolerant system
- Two types of checkpointing:
 - Metadata – for recovery from driver failures
 - Data checkpointing – for basic functioning if using stateful transformations
- Enable checkpointing
 - `ssc.checkpoint("hdfs://...")`

Knowledge Check



Spark Streaming uses several techniques to achieve fault tolerance on streaming data. Match the techniques listed below with the way in which they help ensure fault tolerance.

- | | |
|---------------------|--|
| A. Data Replication | <input type="checkbox"/> Writes data to a fault tolerant system, and sends acknowledgement of receipt, to protect against receiver failure |
| B. Write-Ahead Logs | <input type="checkbox"/> Save data to a fault tolerant system for recovery from a driver failure |
| C. Checkpointing | <input type="checkbox"/> Saves data on multiple nodes for recovery should a single node fail |



Lab 6.5: Build a Streaming Application

- Estimated time to complete: **1 hour, 40 minutes**
- In this multi-part lab, you will:
 - Load data using the Spark shell
 - Use Spark Streaming to process a CSV file and display the contents
 - Build a Spark Streaming application using various methods

Lesson 7. Use Apache Spark Graph Frames.





Learning Goals

- 7.1 Describe GraphFrame
- 7.2 Define Regular, Directed, and Property Graphs
- 7.3 Create a Property Graph
- 7.4 Perform Operations on Graphs

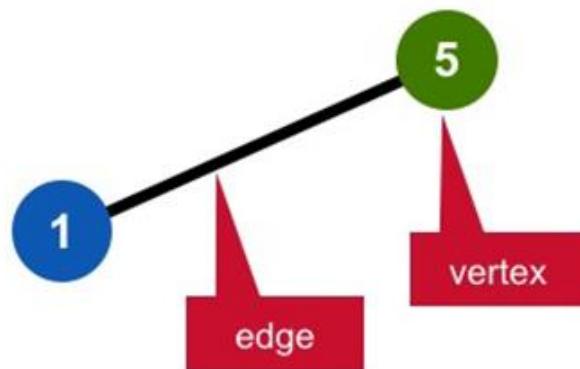


Learning Goals

- 7.1 **Describe GraphFrame**
- 7.2 Define Regular, Directed, and Property Graphs
- 7.3 Create a Property Graph
- 7.4 Perform Operations on Graphs

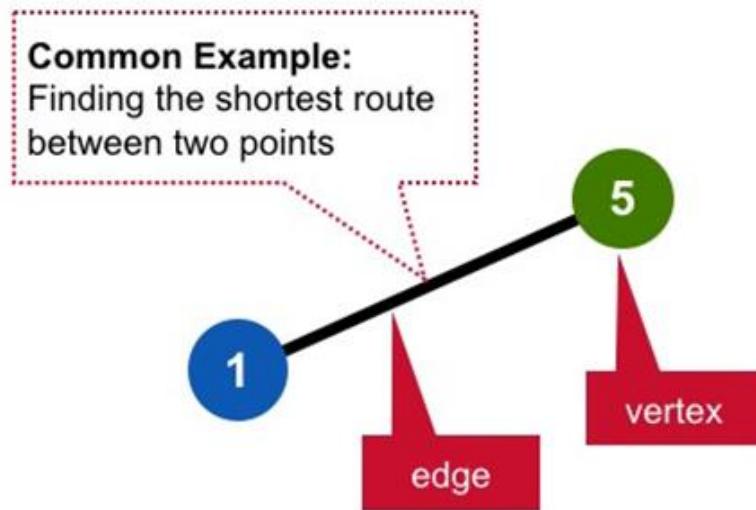
What is a Graph?

Represent a set of vertices that may be connected by edges



What is a Graph?

Represent a set of vertices that may be connected by edges



What is GraphFrame?

- The Apache Spark component for graphs and graph-parallel computations.
- A distributed graph processing framework that sits on top of Spark core

| Components | Function |
|-----------------|--|
| Spark SQL | <ul style="list-style-type: none">• Structure Data• Querying with SQL/HQL |
| Spark Streaming | <ul style="list-style-type: none">• Processing of live streams• Micro-batching |
| MLlib | <ul style="list-style-type: none">• Machine Learning• Multiple types of ML algorithms |
| GraphFrame | <ul style="list-style-type: none">• Graph processing• Graph parallel computations |
| Spark Core | <ul style="list-style-type: none">• Task scheduling• Memory management• Fault recovery• Interacting with storage system |

Apache Spark GraphFrame

- Combines data parallel and graph parallel processing in single API
- View data as graphs and as collections (DataFrames)
 - No duplication or movement of data
- Provides graph algorithms and builders

| Components | Function |
|-----------------|--|
| Spark SQL | <ul style="list-style-type: none">• Structure Data• Querying with SQL/HQL |
| Spark Streaming | <ul style="list-style-type: none">• Processing of live streams• Micro-batching |
| MLlib | <ul style="list-style-type: none">• Machine Learning• Multiple types of ML algorithms |
| GraphFrame | <ul style="list-style-type: none">• Graph processing• Graph parallel computations |
| Spark Core | <ul style="list-style-type: none">• Task scheduling• Memory management• Fault recovery• Interacting with storage system |

Knowledge Check



GraphFrame is the Apache Spark component for graphs and graph parallel computations. Which of the characteristics listed below are true of GraphFrame?

- A. GraphFrame sits on top of Spark Core
- B. GraphFrame provides distinct APIs for data parallel and graph parallel processing
- C. GraphFrame provides multiple operators
- D. When using GraphFrame, we can view data as graphs and as collections without the need for duplication or movement of data

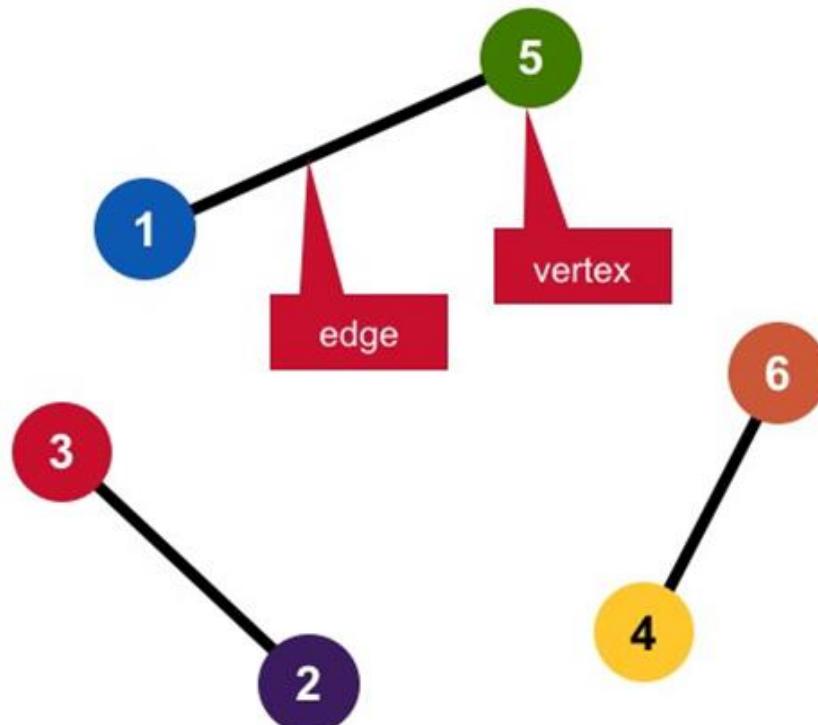


Learning Goals

- 7.1 Describe GraphFrame
- 7.2 Define Regular, Directed, and Property Graphs**
- 7.3 Create a Property Graph
- 7.4 Perform Operations on Graphs

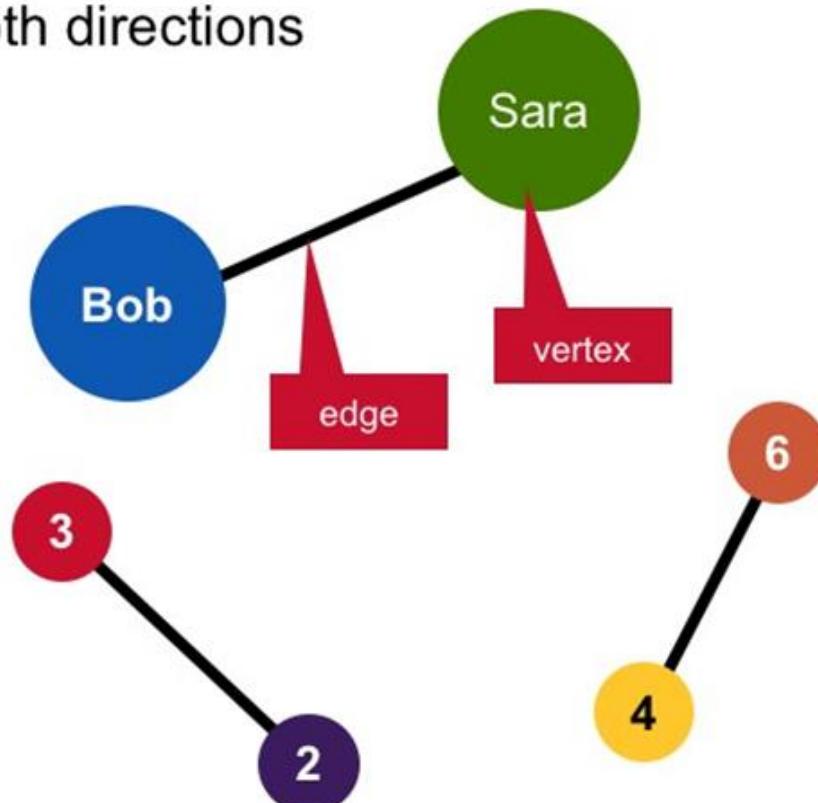
Regular Graphs

Each vertex has the same number of edges



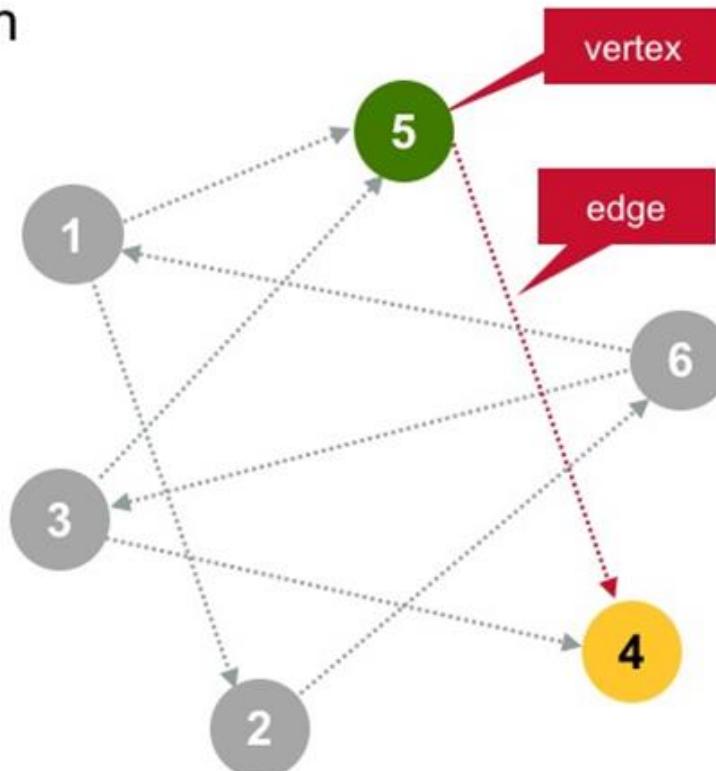
Regular Graphs: Example

Edges run in both directions



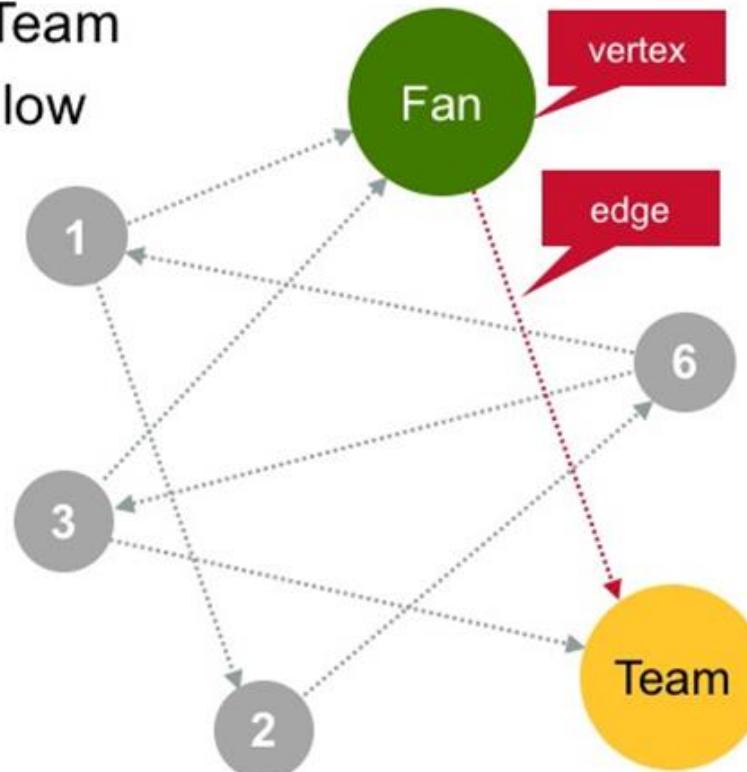
Directed Graphs

Edges run in one direction



Directed Graphs: Social Media Example

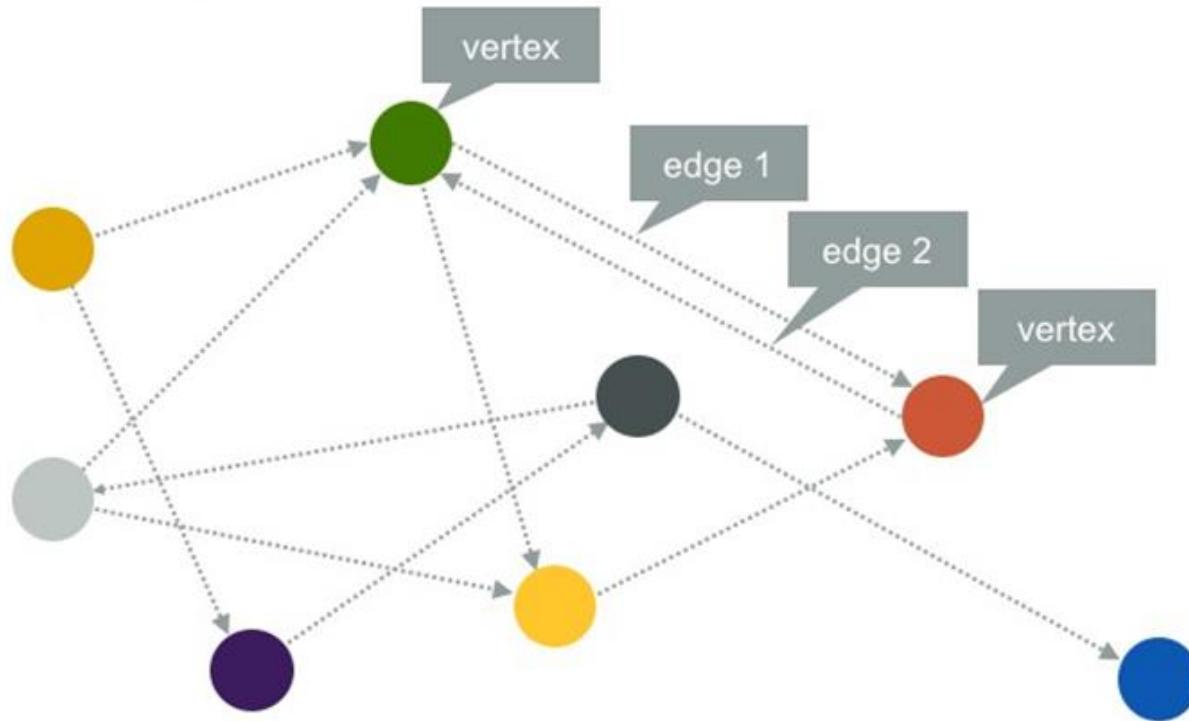
- User Fan follows user Team
- User Team does not follow user Fan



Property Graphs

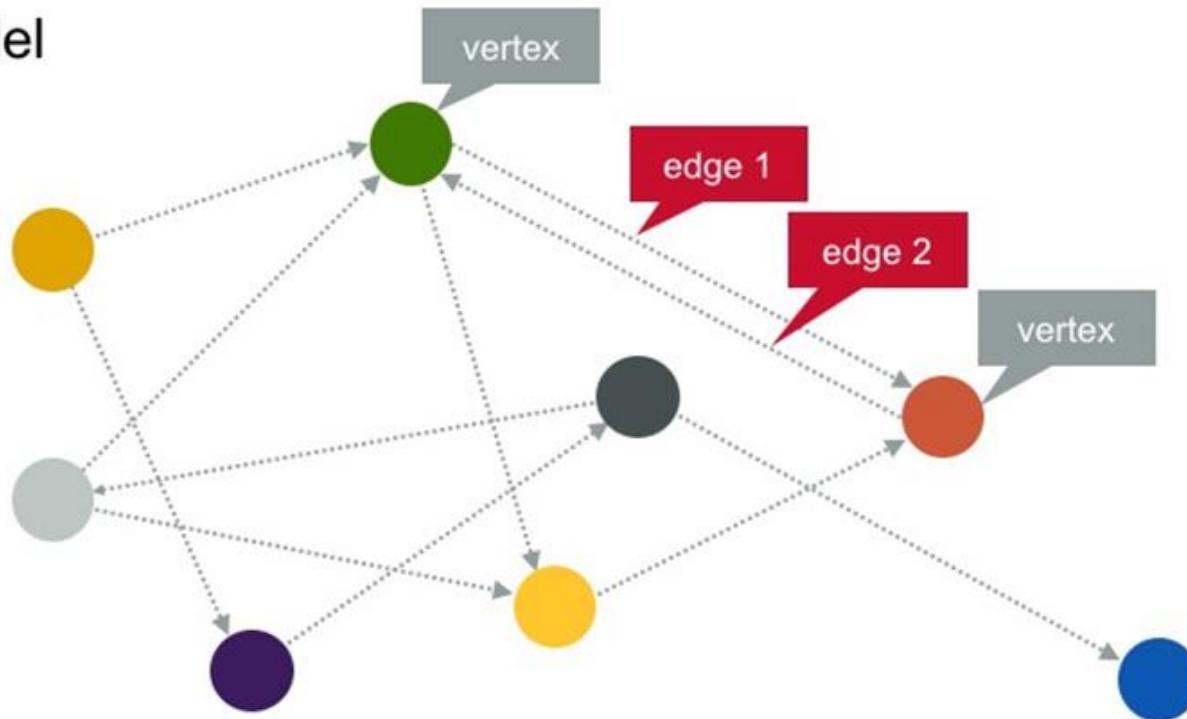
Primary abstraction of Spark GraphFrame

- Immutable
- Distributed
- Fault-tolerant



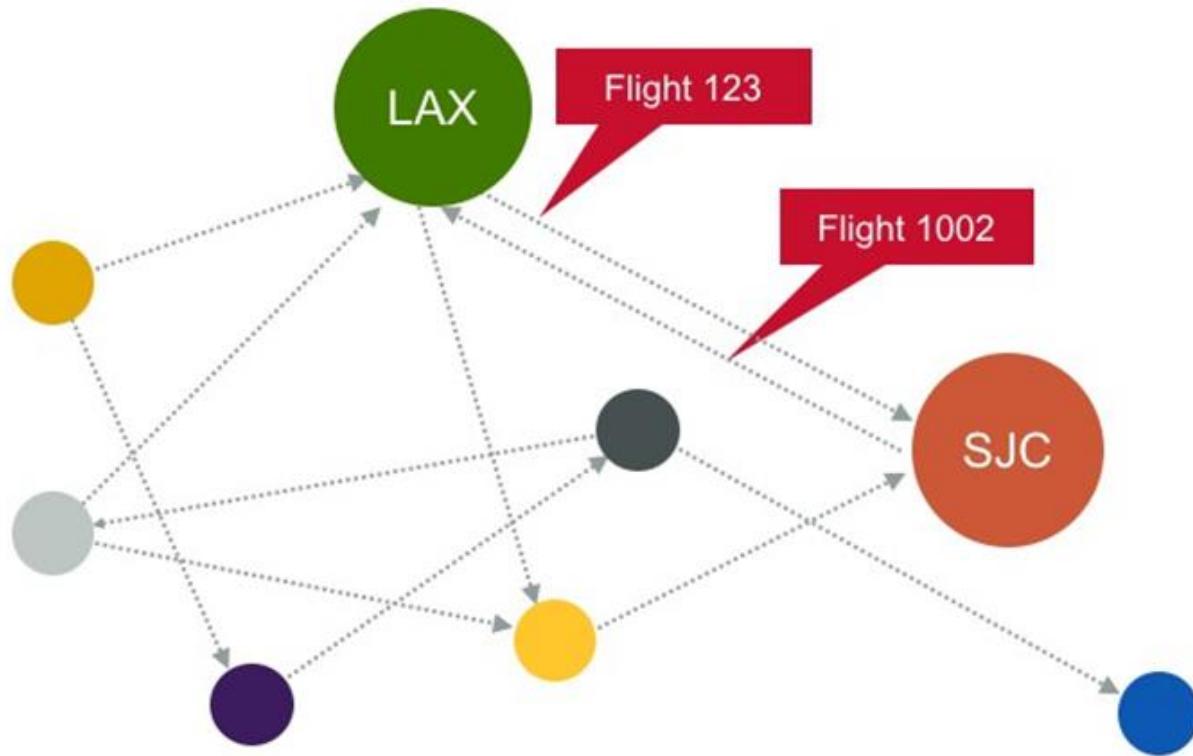
Property Graphs

- Directed multi-graph
- Multiple edges in parallel
 - Unique, user-defined properties for every edge and vertex



Property Graphs: Example

- Airport and flight map
 - Airports are vertices
 - Flights are separate, directed edges





Knowledge Check

The primary abstraction in Spark GraphFrame is the property graph. Which characteristics below are true of a property graph?

- A. Property graphs are bidirectional
- B. Edges and vertices have user-defined properties associated with them
- C. Property graphs are directional
- D. Every edge and vertex is unique
- E. Property graphs are immutable



Learning Goals

-
- 7.1 Describe GraphFrame
 - 7.2 Define Regular, Directed, and Property Graphs
 - 7.3 Create a Property Graph**
 - 7.4 Perform Operations on Graphs

Create a Property Graph: Overview

- 1 Import required classes
- 2 Create vertex DataFrame
- 3 Create edge DataFrame
- 4 Create Graph



Sample Flight Data

Airports (Vertices)

| Vertex ID | Property (V) |
|-----------|--------------|
| id | name |

Routes (Edges)

| Source ID | Dest ID | Property (E) |
|-----------|---------|--------------|
| source | dest | distance |

Step 1: Import Required Classes

```
//import org.graphframes._  
/opt/mapr/spark/spark-2.1.0/bin/spark-shell --packages  
graphframes:graphframes:0.5.0-spark2.1-s_2.11  
  
val vertices = List((1, "SFO"), (2, "ORD"), (3, "DFW"))  
  
val verticesDF = spark.createDataFrame(vertices).toDF("id", "name")  
  
val edges = List((1, 2, 1800), (2, 3, 800), (3, 1, 1400))  
  
val edgesDF =  
spark.createDataFrame(edges).toDF("source", "dest", "distance")  
  
val graph = GraphFrame(verticesDF, edgesDF)
```

Step 2: Create Vertex Data

```
//First create data for vertices (Airports)  
val vertices = List((1, "SFO"), (2, "ORD"), (3, "DFW"))
```

| id | name (Property) |
|-----------|------------------------|
| 1 | SFO |
| 2 | ORD |
| 3 | DFW |

Step 2: Create Vertex Data

```
//import org.graphframes._  
/opt/mapr/spark/spark-2.1.0/bin/spark-shell --packages  
graphframes:graphframes:0.5.0-spark2.1-s_2.11  
  
val vertices = List((1, "SFO") , (2, "ORD") , (3,"DFW"))  
  
val verticesDF = spark.createDataFrame(vertices).toDF("id","name")  
  
val edges = List((1,2,1800) , (2,3,800) , (3,1,1400))  
  
val edgesDF =  
spark.createDataFrame(edges).toDF("source","dest","distance")  
  
val graph = GraphFrame(verticesDF, edgesDF)
```

Step 3: Create Vertex DataFrame

```
//import org.graphframes._  
/opt/mapr/spark/spark-2.1.0/bin/spark-shell --packages  
graphframes:graphframes:0.5.0-spark2.1-s_2.11  
  
val vertices = List((1, "SFO"), (2, "ORD"), (3, "DFW"))  
  
val verticesDF = spark.createDataFrame(vertices).toDF("id","name")  
  
val edges = List((1,2,1800), (2,3,800), (3,1,1400))  
  
val edgesDF =  
spark.createDataFrame(edges).toDF("source","dest","distance")  
  
val graph = GraphFrame(verticesDF, edgesDF)
```

Step 4: Create Edge Data

```
//Create data for edges (Routes)  
val edges = List((1,2,1800), (2,3,800), (3,1,1400))
```



| source | dest | distance (Property) |
|--------|------|---------------------|
| 1 | 2 | 1800 |
| 2 | 3 | 800 |
| 3 | 1 | 1400 |

Step 4: Create Edge Data

```
//import org.graphframes._  
/opt/mapr/spark/spark-2.1.0/bin/spark-shell --packages  
graphframes:graphframes:0.5.0-spark2.1-s_2.11  
  
val vertices = List((1, "SFO"), (2, "ORD"), (3, "DFW"))  
  
val verticesDF = spark.createDataFrame(vertices).toDF("id", "name")  
  
val edges = List((1,2,1800), (2,3,800), (3,1,1400))  
  
val edgesDF =  
spark.createDataFrame(edges).toDF("source", "dest", "distance")  
  
val graph = GraphFrame(verticesDF, edgesDF)
```

Step 5: Create Edge DataFrame

```
//import org.graphframes._  
/opt/mapr/spark/spark-2.1.0/bin/spark-shell --packages  
graphframes:graphframes:0.5.0-spark2.1-s_2.11  
  
val vertices = List((1, "SFO"), (2, "ORD"), (3, "DFW"))  
  
val verticesDF = spark.createDataFrame(vertices).toDF("id", "name")  
  
val edges = List((1,2,1800), (2,3,800), (3,1,1400))  
  
val edgesDF =  
spark.createDataFrame(edges).toDF("source", "dest", "distance")  
  
val graph = GraphFrame(verticesDF, edgesDF)
```

Step 6: Create Property Graph

```
//import org.graphframes._  
/opt/mapr/spark/spark-2.1.0/bin/spark-shell --packages  
graphframes:graphframes:0.5.0-spark2.1-s_2.11  
  
val vertices = List((1, "SFO"), (2, "ORD"), (3, "DFW"))  
  
val verticesDF = spark.createDataFrame(vertices).toDF("id", "name")  
  
val edges = List((1, 2, 1800), (2, 3, 800), (3, 1, 1400))  
  
val edgesDF =  
spark.createDataFrame(edges).toDF("source", "dest", "distance")  
  
val graph = GraphFrame(verticesDF, edgesDF)
```

Knowledge Check



The steps to create a Property graph are listed below. List these steps in the correct order.

- Create Vertex DataFrame
- Create Graph
- Import Required Classes
- Create Edge DataFrame

Learning Goals



- 7.1 Describe GraphFrame
- 7.2 Define Regular, Directed, and Property Graphs
- 7.3 Create a Property Graph
- 7.4 Perform Operations on Graphs**

Graph Operators: Collection and Structural

| Operator | Description |
|---------------|--|
| vertices | Returns a DataFrame containing the vertices and associated attributes. |
| edges | Returns a DataFrame containing the edges and associated attributes. |
| find(pattern) | Searches the graph for structural patterns (Motif finding) and returns a DataFrame which consists of all instances of Motif. |
| triplets | Returns a DataFrame with 3 columns: source, edge and dest. Schema for columns source and dest match GraphFrame.vertices and the schema for the column edge matches GraphFrame.edges. |

Graph Operators: Example

```
//Check vertices and edges of Graph  
graph.vertices.show  
graph.edges.show
```

```
scala > graph.vertices.show  
+---+  
| id|name|  
+---+  
| 1 | SFO|  
| 2 | ORD|  
| 3 | DFW|  
+---+
```

```
scala > graph.edges.show  
+-----+---+-----+  
| source|dest|distance|  
+-----+---+-----+  
| 1     | 2   | 1800 |  
| 2     | 3   | 800  |  
| 3     | 1   | 1400 |  
+-----+---+-----+
```

Graph Operators: Graph Information

| Operator | Description |
|---------------|---|
| vertexColumns | Names of columns in vertices DataFrame. |
| edgeColumns | Names of columns in edges DataFrame. |
| inDegrees | The in-degree of each vertex. |
| outDegrees | The out-degree of each vertex. |
| Degrees | The degree of each vertex. |

Graph Operators: Vertex Degrees

```
//Check degrees of vertices of Graph  
graph.degrees.show
```

```
//Check in degree of vertices  
graph.inDegrees.show
```

```
scala > graph.degrees.show  
+---+-----+  
| id|degree|  
+---+-----+  
| 1 | 2 |  
| 2 | 2 |  
| 3 | 2 |  
+---+-----+
```

```
scala > graph.inDegrees.show  
+---+-----+  
| id|inDegree|  
+---+-----+  
| 1 | 1 |  
| 3 | 1 |  
| 2 | 1 |  
+---+-----+
```

Graph Operators: Caching Graphs

| Operator | Description |
|----------------------------------|---|
| <code>cache()</code> | Caches the vertices and edges; default level is <code>MEMORY_ONLY</code> . |
| <code>persist(newLevel)</code> | Caches the vertices and edges at specified storage level; returns a reference to this graph. |
| <code>unpersist()</code> | Uncaches both vertices and edges of this graph from memory and disk. |
| <code>unpersist(blocking)</code> | Similar to <code>unpersist()</code> but accepts a boolean value whether to block until all blocks are removed from memory and disk. |

Graph Operators: Standard Graph Algorithms

| Operator | Description |
|-----------------------------|--|
| bfs | Breadth-First-Search (BFS) algorithm for traversing a graph. |
| pageRank | Algorithm that measures the importance of vertices in a graph. |
| shortestPaths | Finds the shortest path between vertices. |
| connectedComponents | Finds the connected components of a graph. |
| stronglyConnectedComponents | Finds strongly connected components of a directed graph. |



Class Discussion

1. How many airports are there?
 - In our graph, what represents airports?
 - Which operator could you use to find the number of airports?

2. How many routes are there?
 - In our graph, what represents routes?
 - Which operator could you use to find the number of routes?

How Many Airports are There?

How many airports are there?

- In our graph, what represents airports?

Vertices

- Which operator could you use to find the number of airports?

`graph.vertices.count`

How Many Routes are There?

How many routes are there?

- In our graph, what represents routes?
Edges
- Which operator could you use to find the number of routes?
graph.edges.count

Use Cases

- Monitor air traffic at airports
- Monitor delays
- Analyze airport and routes overall
- Analyze airport and routes by airline





Lab 7.4: Analyze Data with GraphFrame

- Estimated time to complete: **40 minutes**
- In this lab, you will use GraphFrame to analyze flight data and retrieve various flight statistics.

Lesson 8: Use Apache Spark MLlib, the machine learning library.





Learning Goals

-
- 8.1 Describe Apache Spark MLlib Machine Learning Algorithms
 - 8.2 Use Collaborative Filtering to Predict User Choice

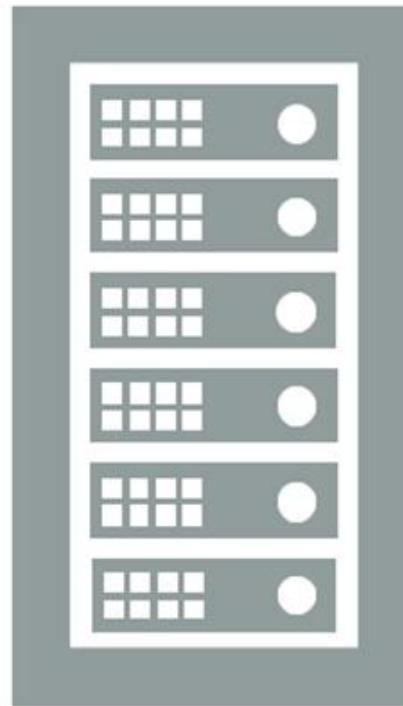


Learning Goals

- 8.1 **Describe Apache Spark MLlib Machine Learning Algorithms**
- 8.2 Use Collaborative Filtering to Predict User Choice

What is MLlib?

- Only includes machine learning algorithms designed to run on clusters
- Most suitable for running on a single large data set



Examples of ML Algorithms

Supervised

- Classification
- Regression Analysis
- Collaborative Filtering

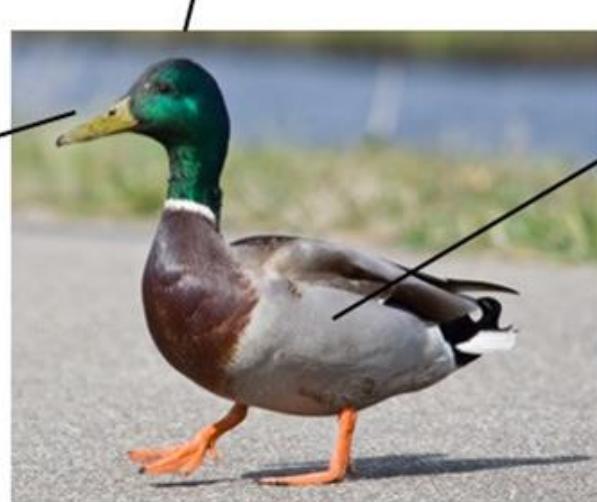
Unsupervised

- Clustering
- Dimensionality Reduction

Examples of ML Algorithms: Supervised

Supervised

- Classification
- Regression Analysis
- Collaborative Filtering



Bird

Examples of ML Algorithms: Unsupervised



Unknown Data

Unsupervised

- Clustering
- Dimensionality Reduction

Classification: Definition

Form of ML that:

- Identifies which category an item belongs to
- Uses supervised learning algorithms
 - Data is labeled



Sentiment

Classification: Example

Classification

The screenshot shows a Gmail inbox interface. At the top, there's a red callout box labeled "Classification" pointing to the search bar. Inside the search bar, the text "in:spam" is highlighted with a red box. To the right of the search bar, another red callout box contains the text "Identifies category for item". Below the search bar, the inbox lists several messages from various senders like judithouedrago, z.loftus, Timothy Diehl, David Foster, and Sofia Kipkalya. A yellow button labeled "The conv..." is partially visible on the right. On the left sidebar, there are links for "COMPOSE", "Inbox (2,960)", "Important", "Sent Mail", "Drafts (21)", "Circles", and "[Gmail]Drafts".

| | | | | |
|--------------------------|----------------------|------------------------|---------------------------|-------------------------|
| <input type="checkbox"/> | <input type="star"/> | <input type="square"/> | judithouedrago | HELP ME DONATE THI |
| <input type="checkbox"/> | <input type="star"/> | <input type="square"/> | z.loftus | (no subject) - DO YOU |
| <input type="checkbox"/> | <input type="star"/> | <input type="square"/> | Timothy Diehl, Board Pre. | Leadership change at E |
| <input type="checkbox"/> | <input type="star"/> | <input type="square"/> | David Foster | Standards all of us sho |
| <input type="checkbox"/> | <input type="star"/> | <input type="square"/> | Sofia Kipkalya | Dearest One, - Dearest |

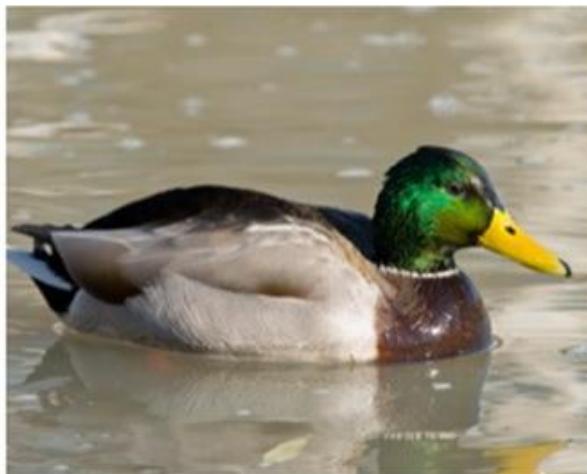
If it Walks/Swims/Quacks Like a Duck... Then It Must Be a Duck

Features of ML:

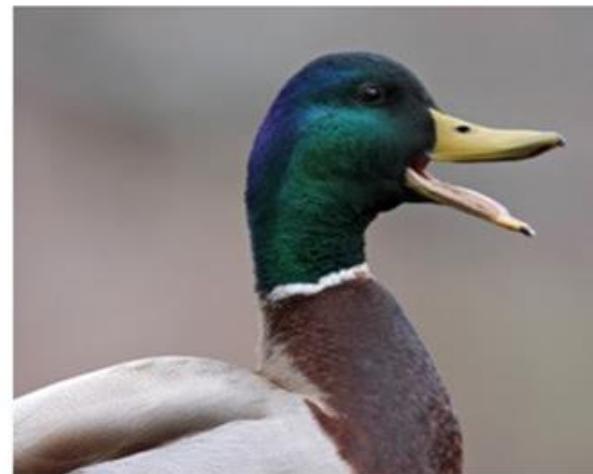
- Classify something based on pre-determined features
- Features are the “if questions” that you ask



Walks



Swims



Quacks

Building and Deploying a Classifier Model

Spam:

free money now!
Super sale!
free savings \$\$\$

Non-spam:

how are you?
that Spark job
Team meeting

Training Data

Building and Deploying a Classifier Model

Featurization

Spam:

free money now!

Super sale!

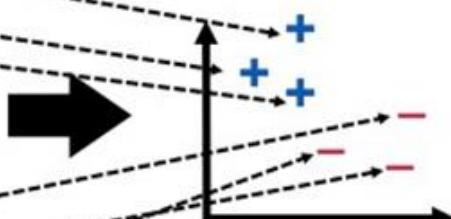
free savings \$\$\$

Non-spam:

how are you?

that Spark job

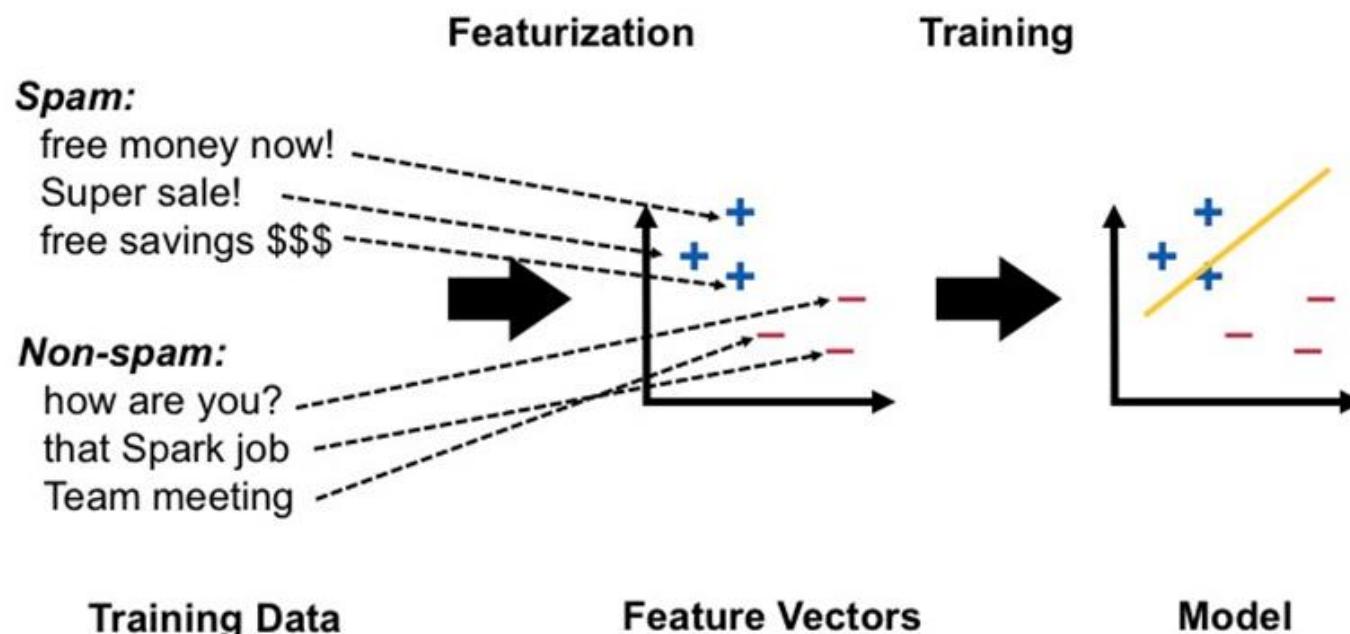
Team meeting



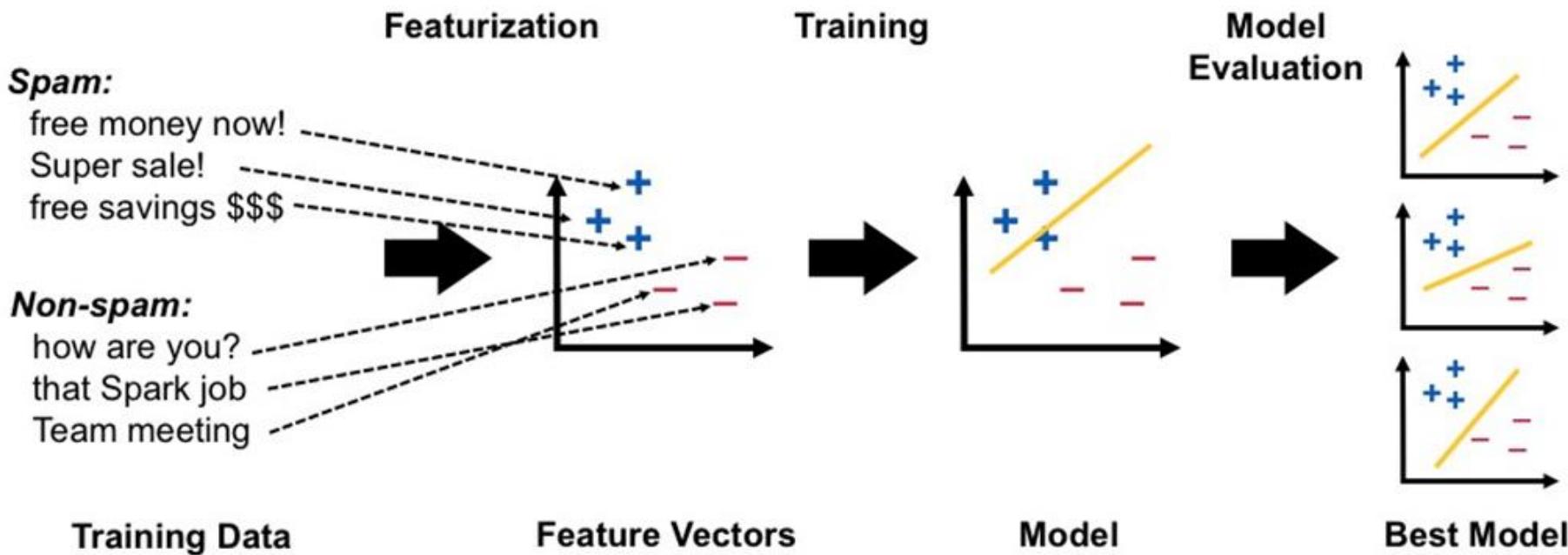
Training Data

Feature Vectors

Building and Deploying a Classifier Model



Building and Deploying a Classifier Model



Training Data

Feature Vectors

Model

Best Model

Knowledge Check



Put the steps for creating and training a Classification model into the correct order:

Define the classification features

Create feature vectors by ranking how strongly each feature classifies the data

Associate the features with the label to train the model

Extract the set of features from new data

Classify the new data with one of the labels

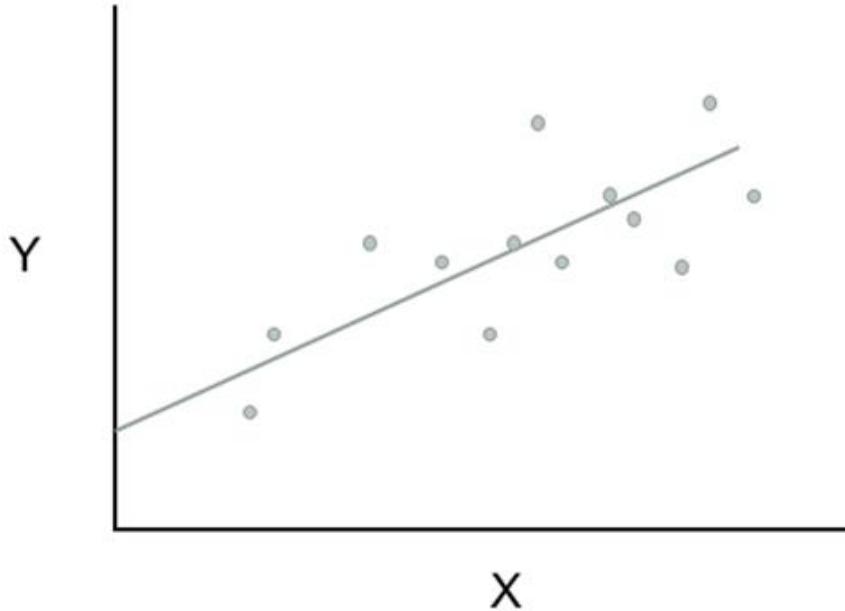
Knowledge Check



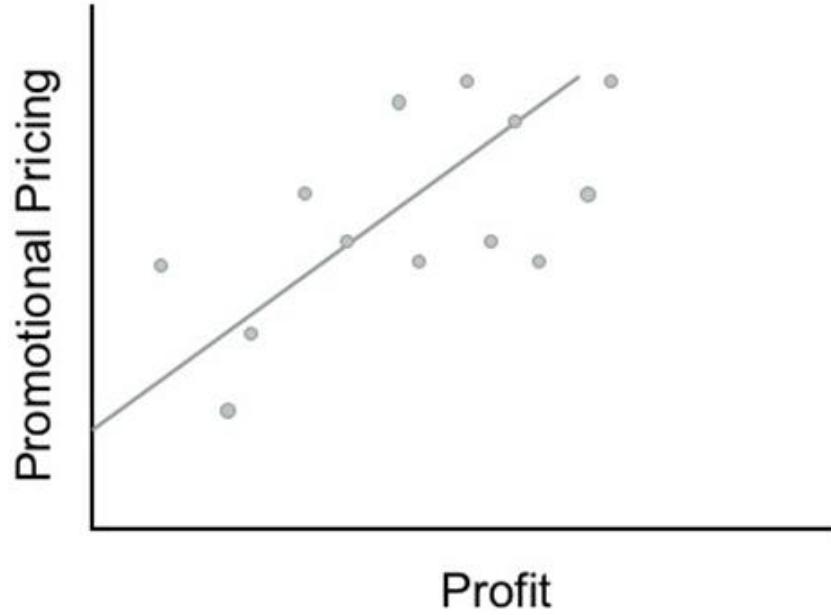
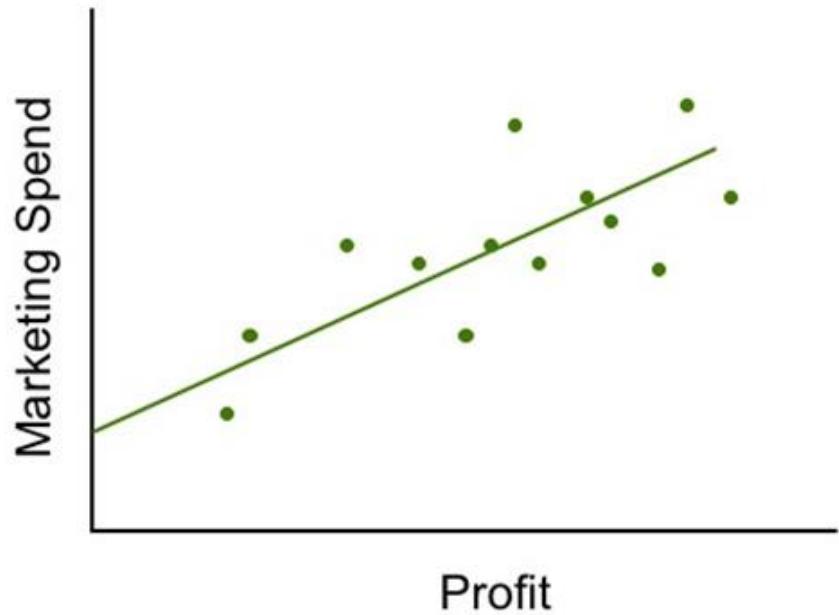
Put the steps for creating and training a Classification model into the correct order:

1. Define the classification features
2. Extract the set of features from new data
3. Create feature vectors by ranking how strongly each feature classifies the data
4. Associate the features with the label to train the model
5. Classify the new data with one of the labels

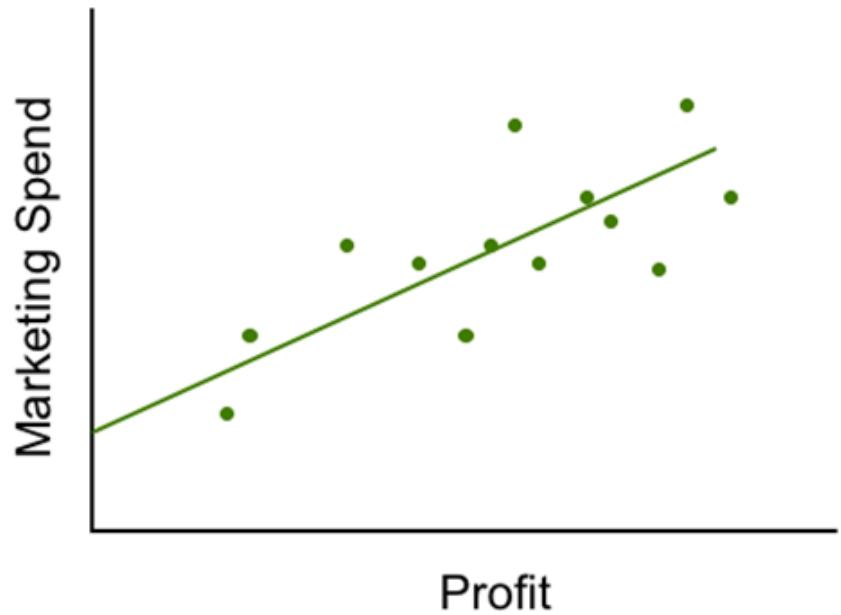
Regression Analysis



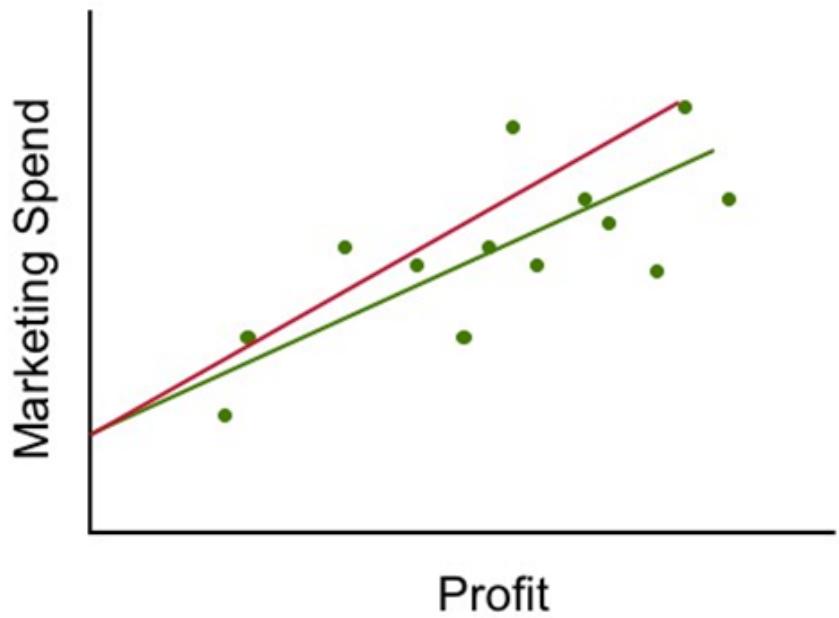
Regression Analysis



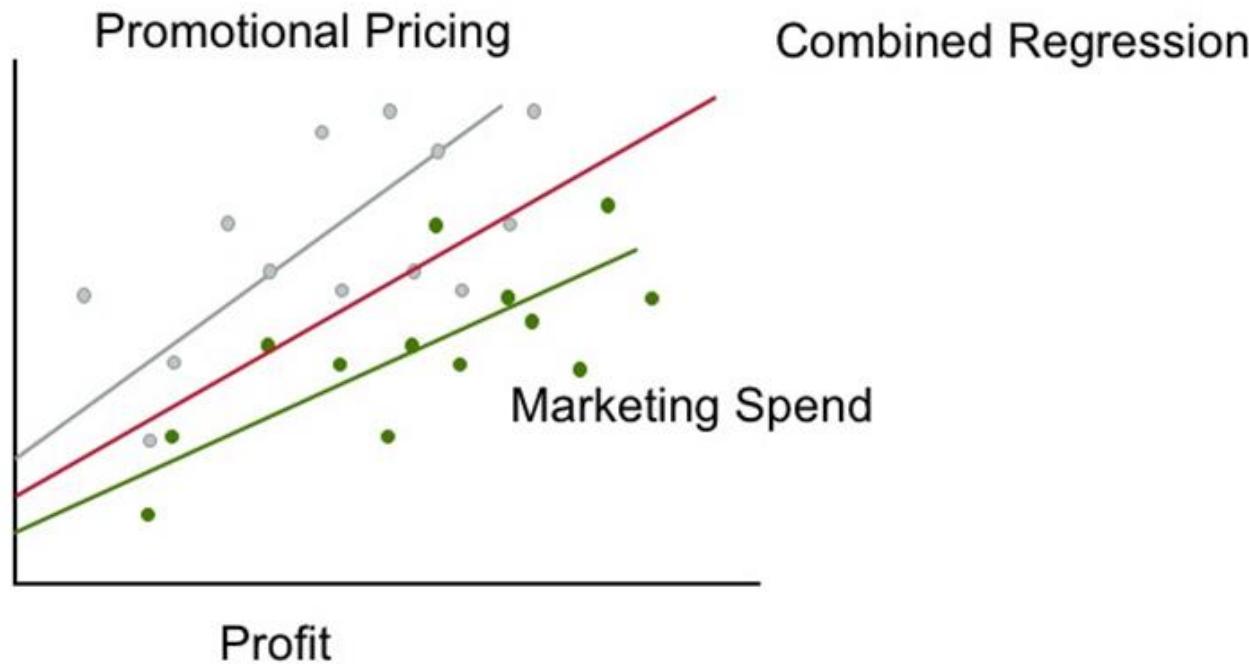
Regression Analysis



Regression Analysis

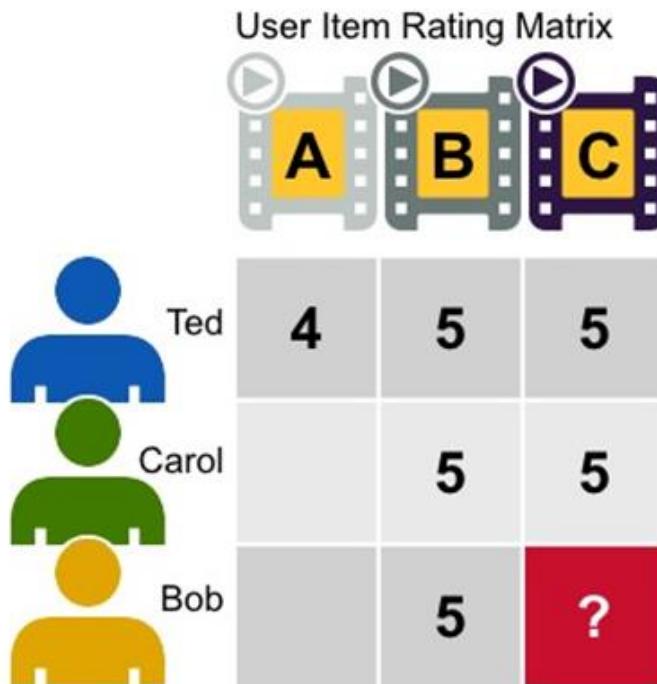


Regression Analysis



Machine Learning: Collaborative Filtering

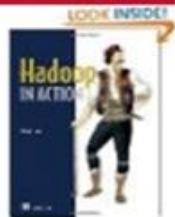
- Based on user preferences data
 - Collaborative
- Recommend items
 - Filtering



Machine Learning: Collaborative Filtering

Collaborative Filtering
(Recommendation)

Customers Who Bought This Item Also Bought



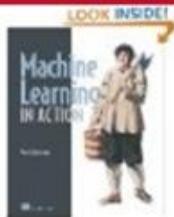
Hadoop in Action

► Chuck Lam

★★★★★ (10)

Paperback

\$27.45



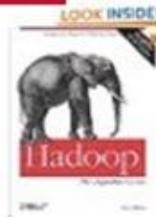
Machine Learning in Action

► Peter Harrington

★★★★★ (17)

Paperback

\$26.49



Hadoop: The Definitive Guide

Tom White

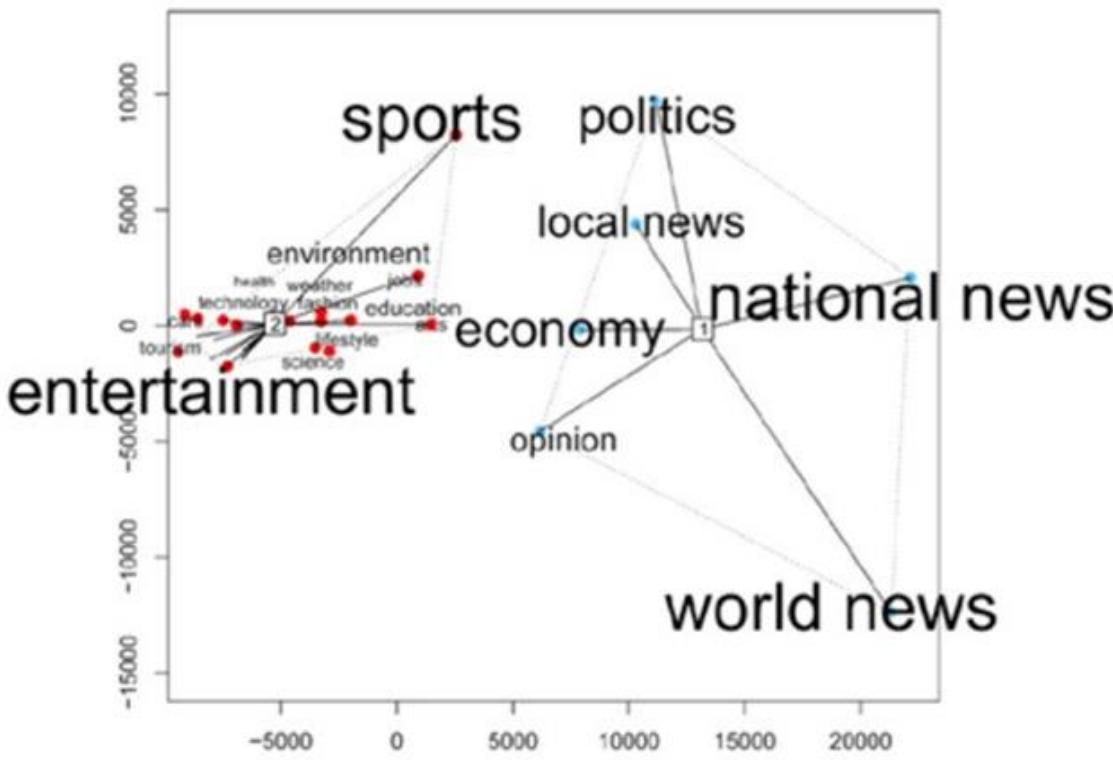
★★★★★ (32)

Paperback

\$28.65

Clustering

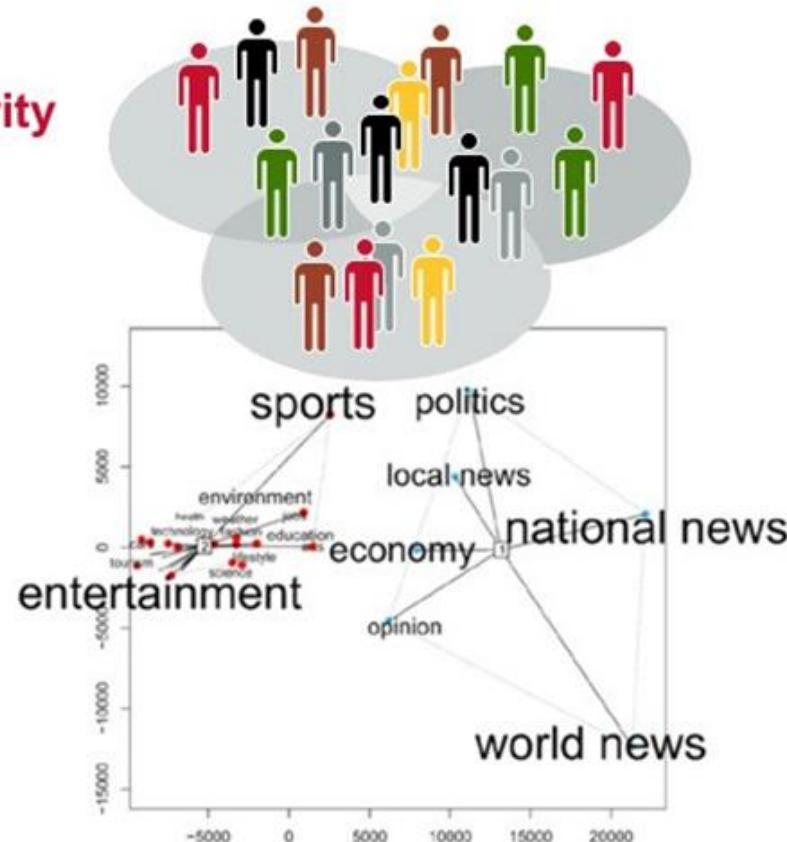
- Classifies inputs into categories by analyzing similarities between input examples



Marco Toledo Bastos. and Gabriela Zaid SAGE

Clustering: Definition

- Unsupervised learning task
- Groups objects into **clusters of high similarity**
 - **Search results** grouping
 - **Grouping of customers**
 - **Anomaly detection**
 - Text categorization



Machine Learning: Clustering

Clustering

Business
Technology
Entertainment
Health
Sports
Spotlight
Science

FDA: New voluntary recall from compounding pharmacy

USA TODAY - 1 hour ago



Fifteen Texas patients got infections after receiving calcium gluconate injections, in the latest nationwide recall associated with compounding pharmacies.



DigitalJournal.com

Texas pharmacy recalls products after infections NBCNews.com

Specialty Compounding recalls sterile medications

Houston Chronicle

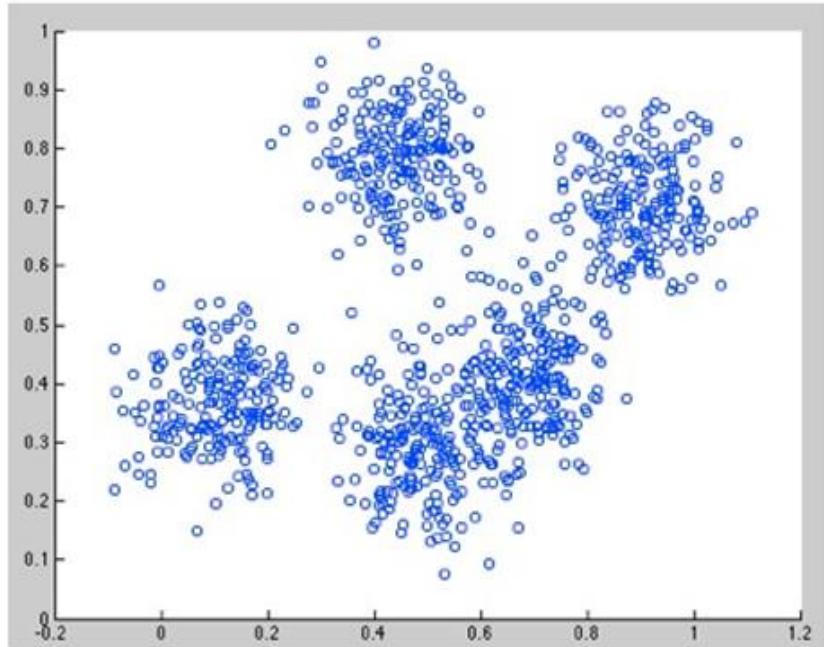
[See realtime coverage »](#)

Vaccine protects against malaria in early test



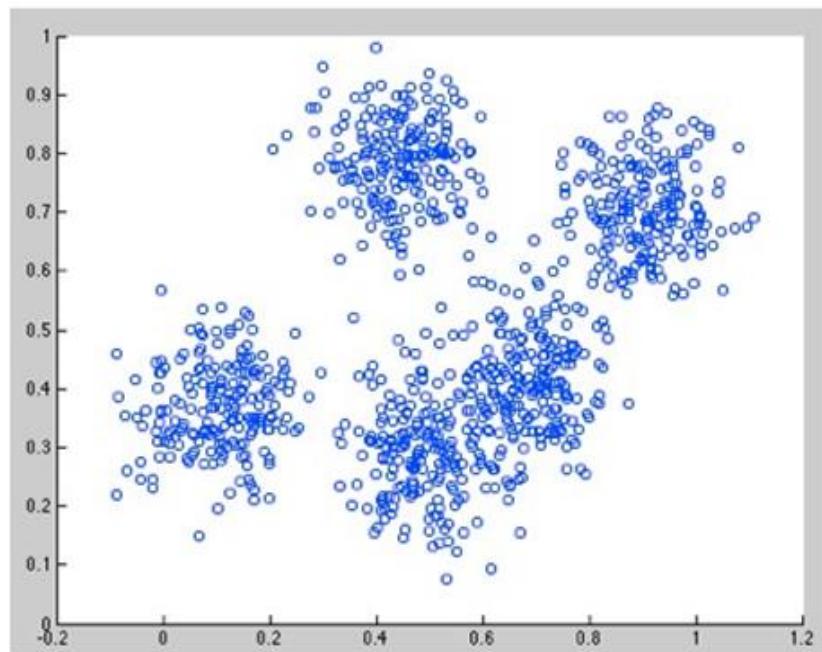
Clustering: Example

- Group similar objects



Clustering: Example

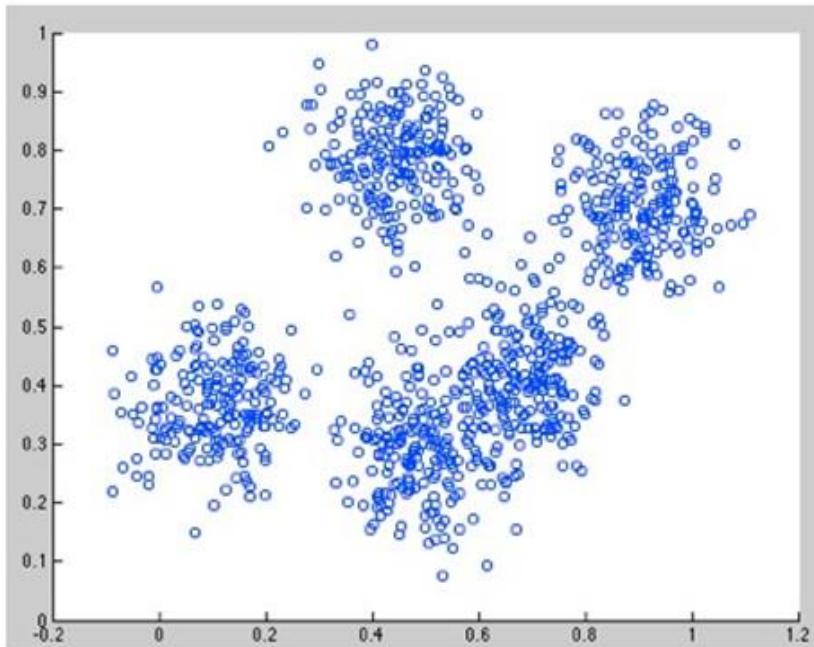
- Group similar objects
 - Use MLlib K-means algorithm
1. Initialize coordinates to center of clusters (centroid)



Clustering: Example

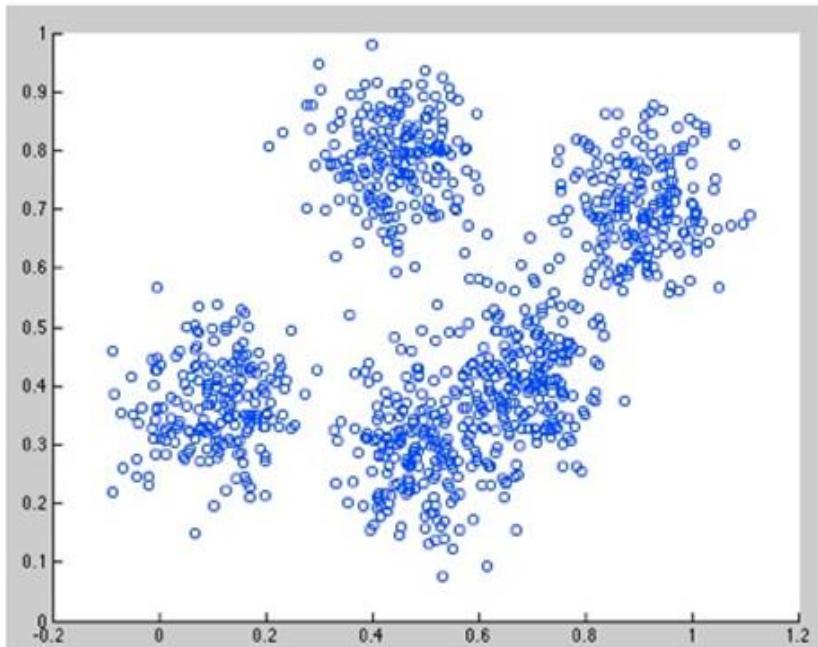
- Group similar objects
- Use MLlib K-means algorithm

1. Initialize coordinates to center of clusters (centroid)
2. Assign all points to nearest centroid



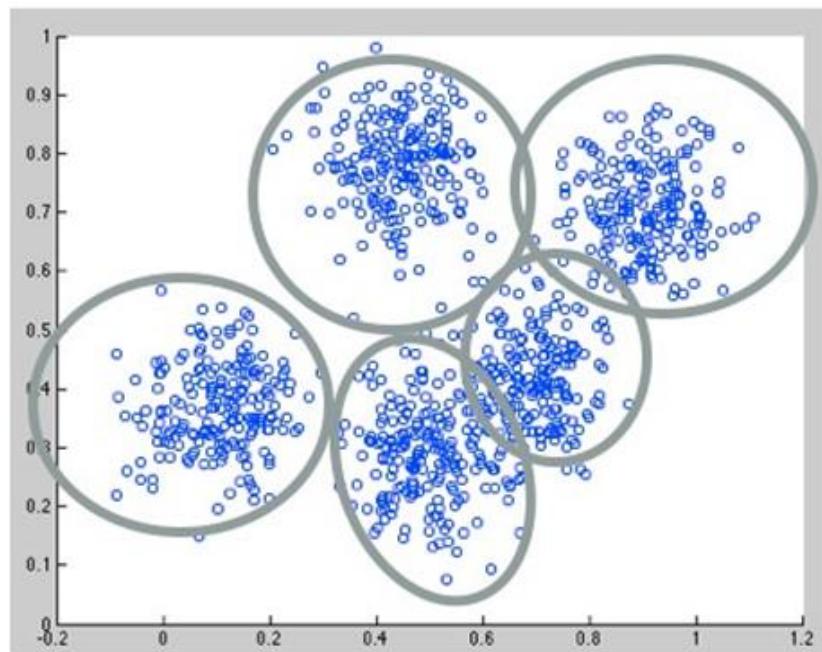
Clustering: Example

- Group similar objects
 - Use MLlib K-means algorithm
1. Initialize coordinates to center of clusters (centroid)
 2. Assign all points to nearest centroid
 3. Update centroids to center of points



Clustering: Example

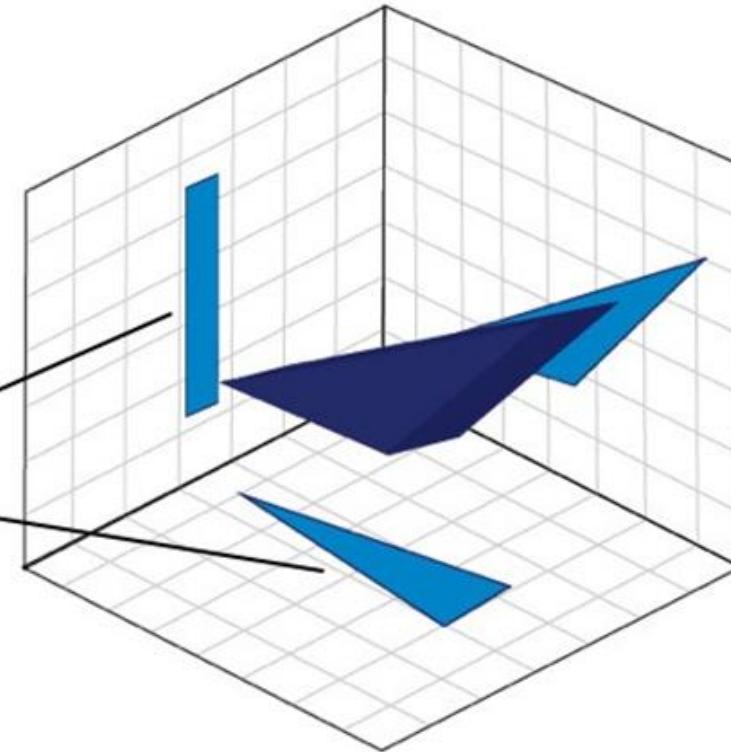
- Group similar objects
 - Use MLlib K-means algorithm
1. Initialize coordinates to center of clusters (centroid)
 2. Assign all points to nearest centroid
 3. Update centroids to center of points
 4. Repeat until conditions met



Dimensionality Reduction

- Reduce dimensions in large dataset
- Process only vital data
- More accurate and efficient

Reduced Two-Dimensional Views

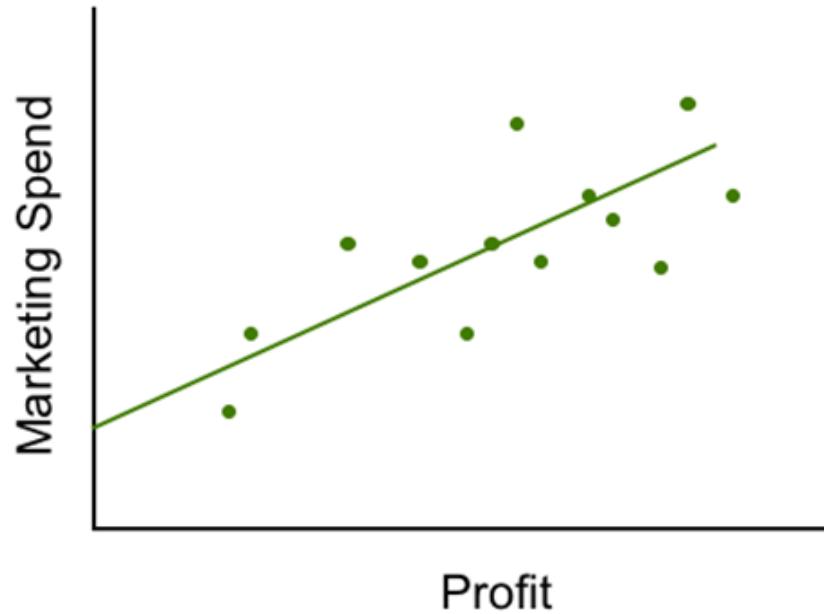


Three-Dimensional Data Set

Dimensionality Reduction: Feature Selection



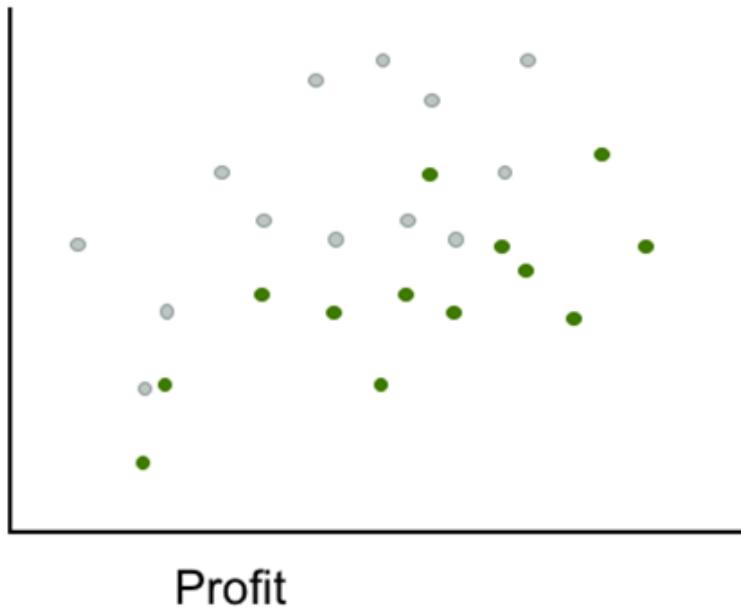
Dimensionality Reduction: Feature Selection



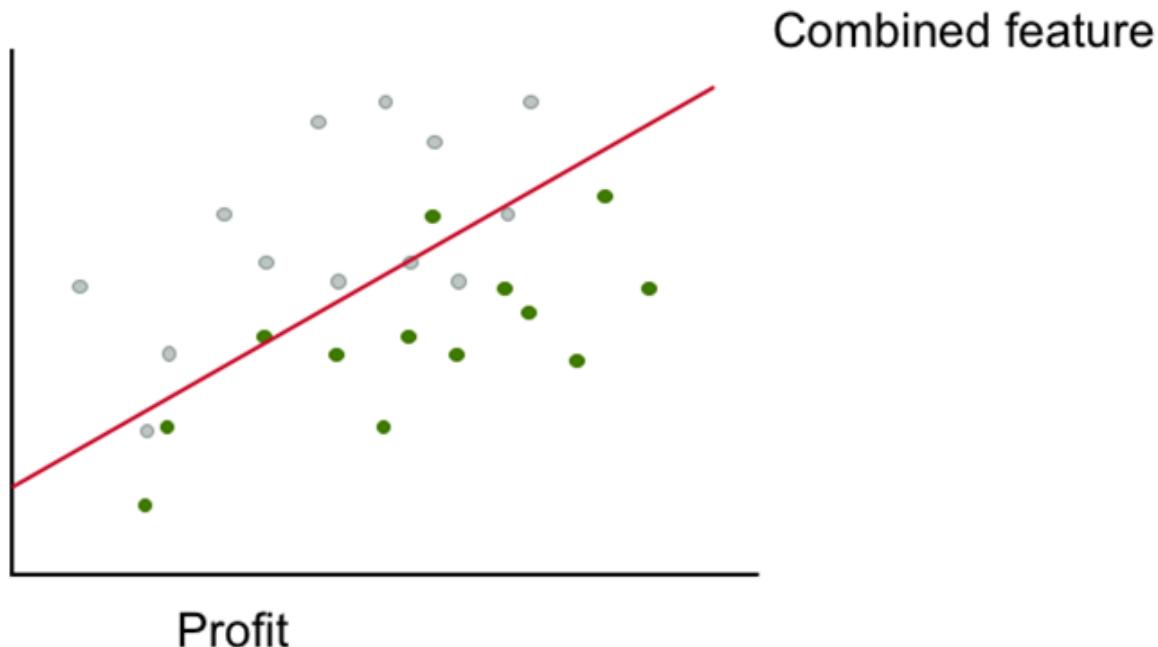
Dimensionality Reduction: Feature Selection

| ID | Longitude | Latitude | Elevation | Oil Pressure | Wind Speed | Fuel |
|--------------|-----------|----------|-----------|--------------|------------|------|
| 180931080809 | 121.9552W | 37.3541N | 152.64 | 31.0 | 4.2ESE | 16.2 |
| 180931080810 | 121.9563W | 37.3765N | 152.33 | 31.1 | 4.2ESE | 16.2 |
| 180931080811 | 121.9575W | 38.4561N | 151.22 | 31.0 | 4.1ESE | 16.2 |
| 180931080812 | 122.0152W | 38.9562N | 150.01 | 31.0 | 4.1ESE | 16.2 |

Dimensionality Reduction: Feature Extraction



Dimensionality Reduction: Feature Extraction



Dimensionality Reduction: Feature Extraction



Video Feed



Reduce Out Color
and Shading



Phone



Basket

Person

Lettuce



Efficient Recognition

Knowledge Check



Match the scenarios listed here with the machine learning technique that would provide the best results: Classification, Clustering, or Collaborative Filtering

- A. You want to determine what restaurant someone may like, based on restaurants they have previously liked.
- B. You want to detect fraudulent attempts to log into your website.
- C. You need to list which students have passed, and which have failed an exam.
- D. You want to organize your music collection based on genre metadata.

Class Discussion



- What is the difference between supervised and unsupervised machine learning algorithms?
- Why is classification considered supervised, while clustering is considered unsupervised?



Learning Goals

- 8.1 Describe Apache Spark MLlib Machine Learning Algorithms
- 8.2 Use Collaborative Filtering to Predict User Choice

Train a Model to Make Predictions

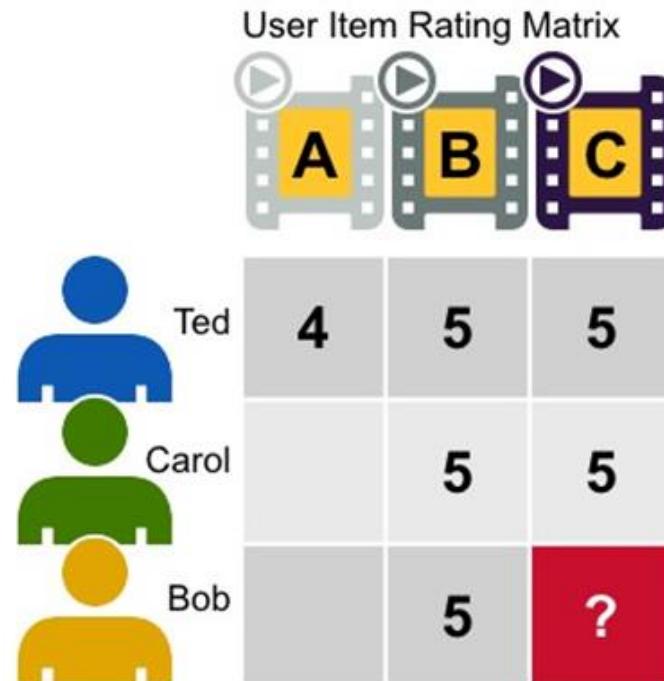
Ted and Carol like movies B and C



Bob likes movie B, what might he like?



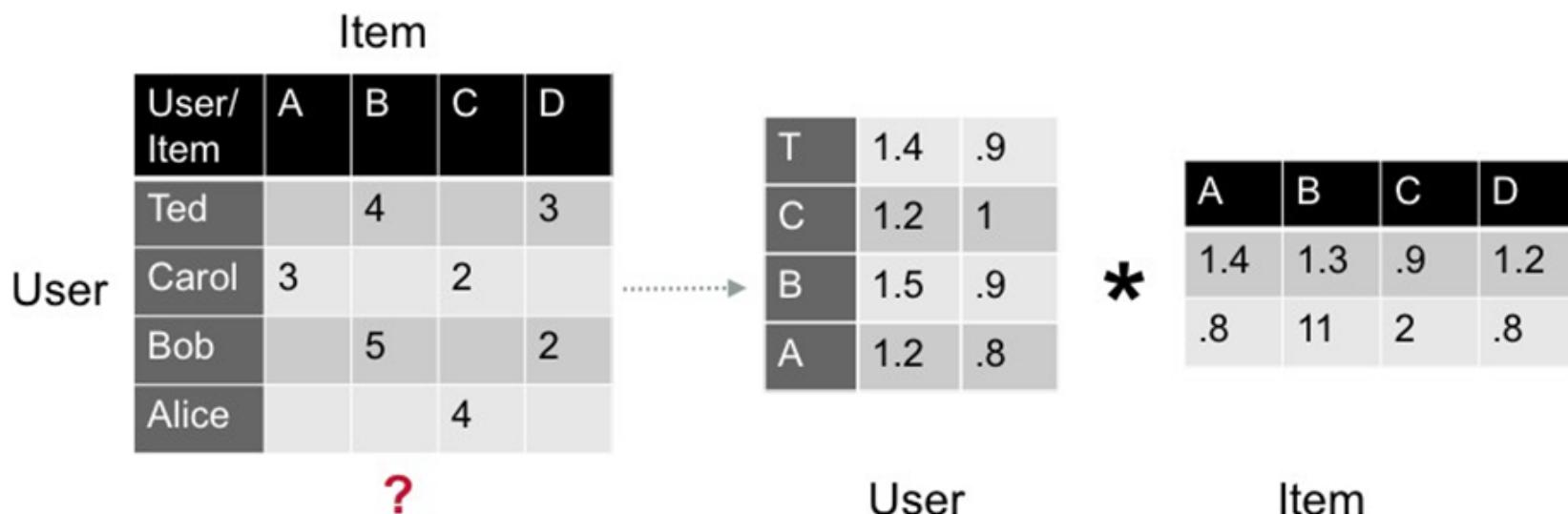
Bob likes movie B, **predict C**



Alternating Least Squares

Approximates sparse user item rating matrix as a product of two dense matrices

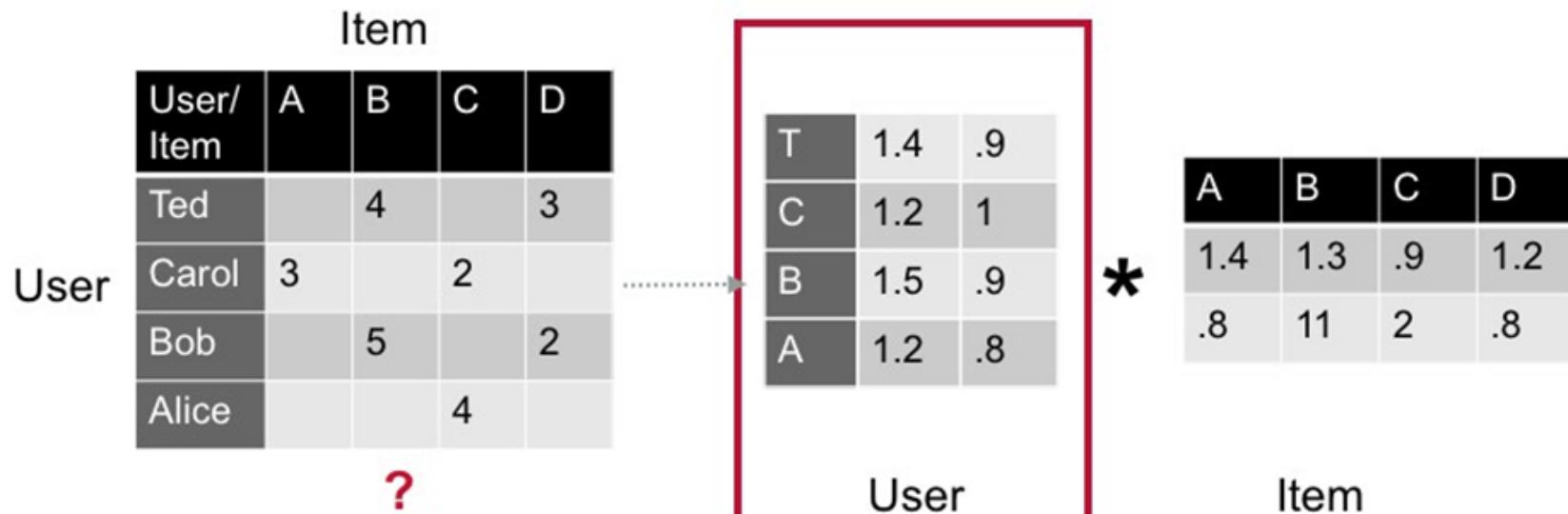
- User and Item factor matrices



Alternating Least Squares

Approximates sparse user item rating matrix as a product of two dense matrices

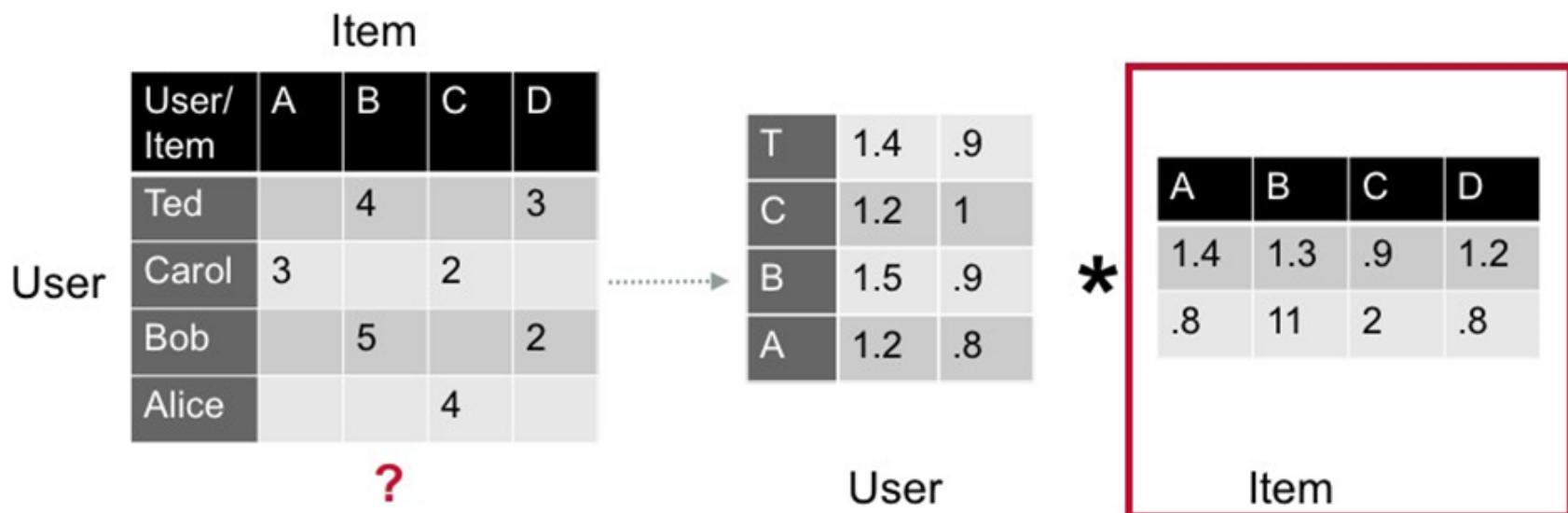
- User and Item factor matrices
- Tries to learn the hidden features of each user and item



Alternating Least Squares

Approximates sparse user item rating matrix as a product of two dense matrices

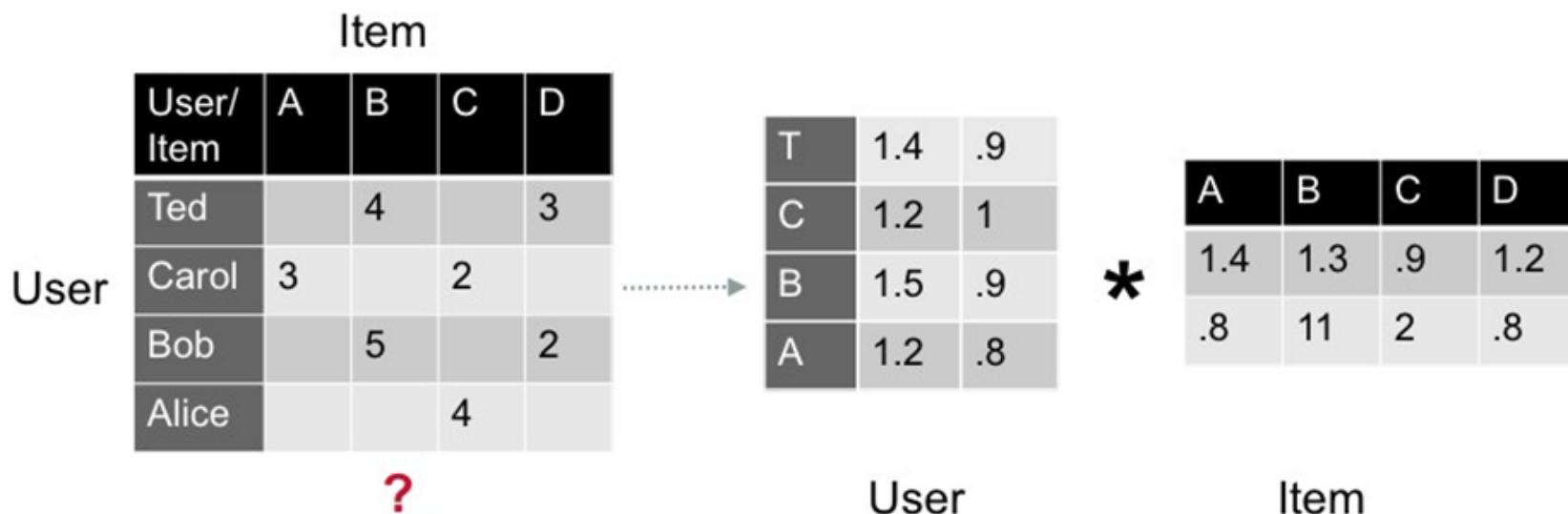
- User and Item factor matrices
- Tries to learn the hidden features of each user and item



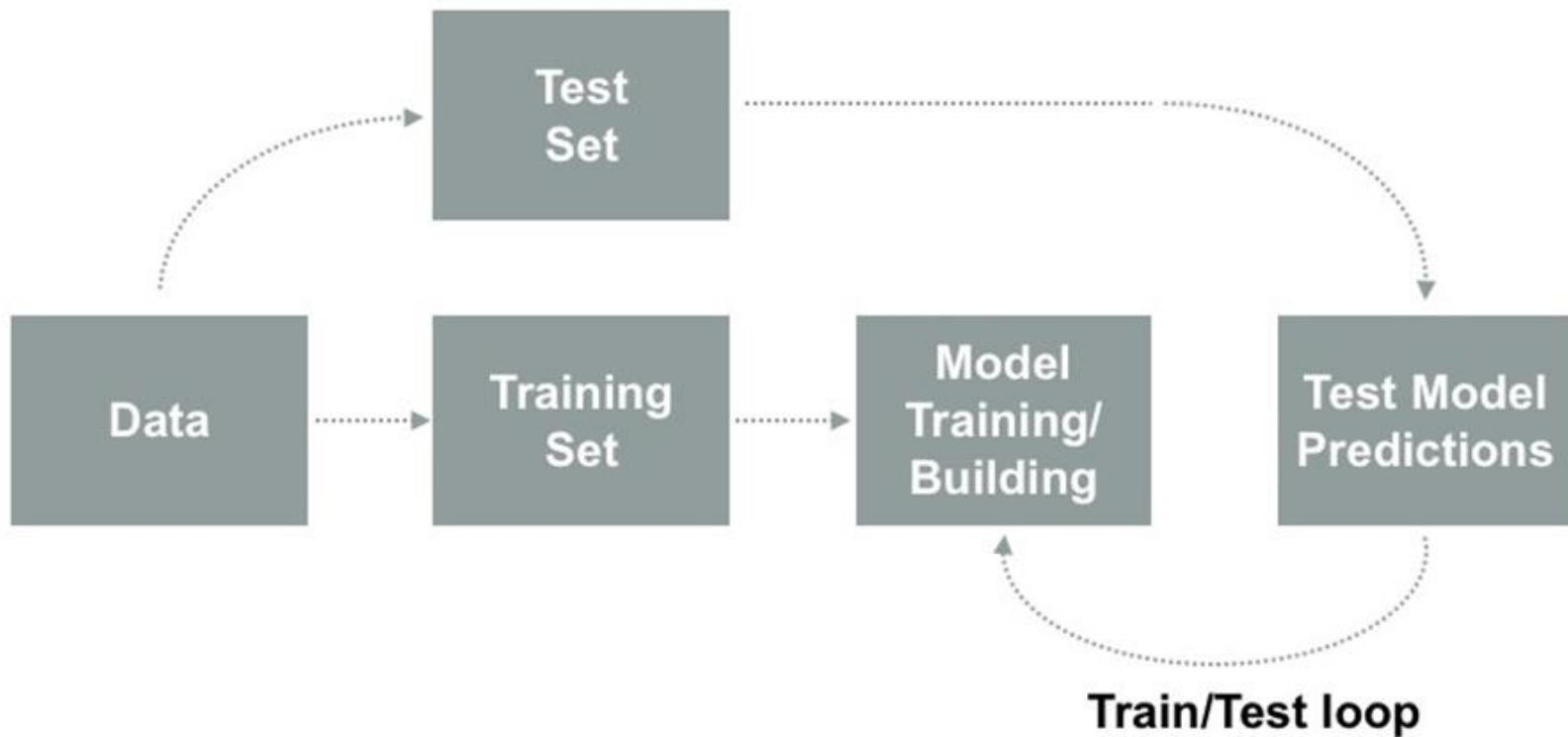
Alternating Least Squares

Approximates sparse user item rating matrix as a product of two dense matrices

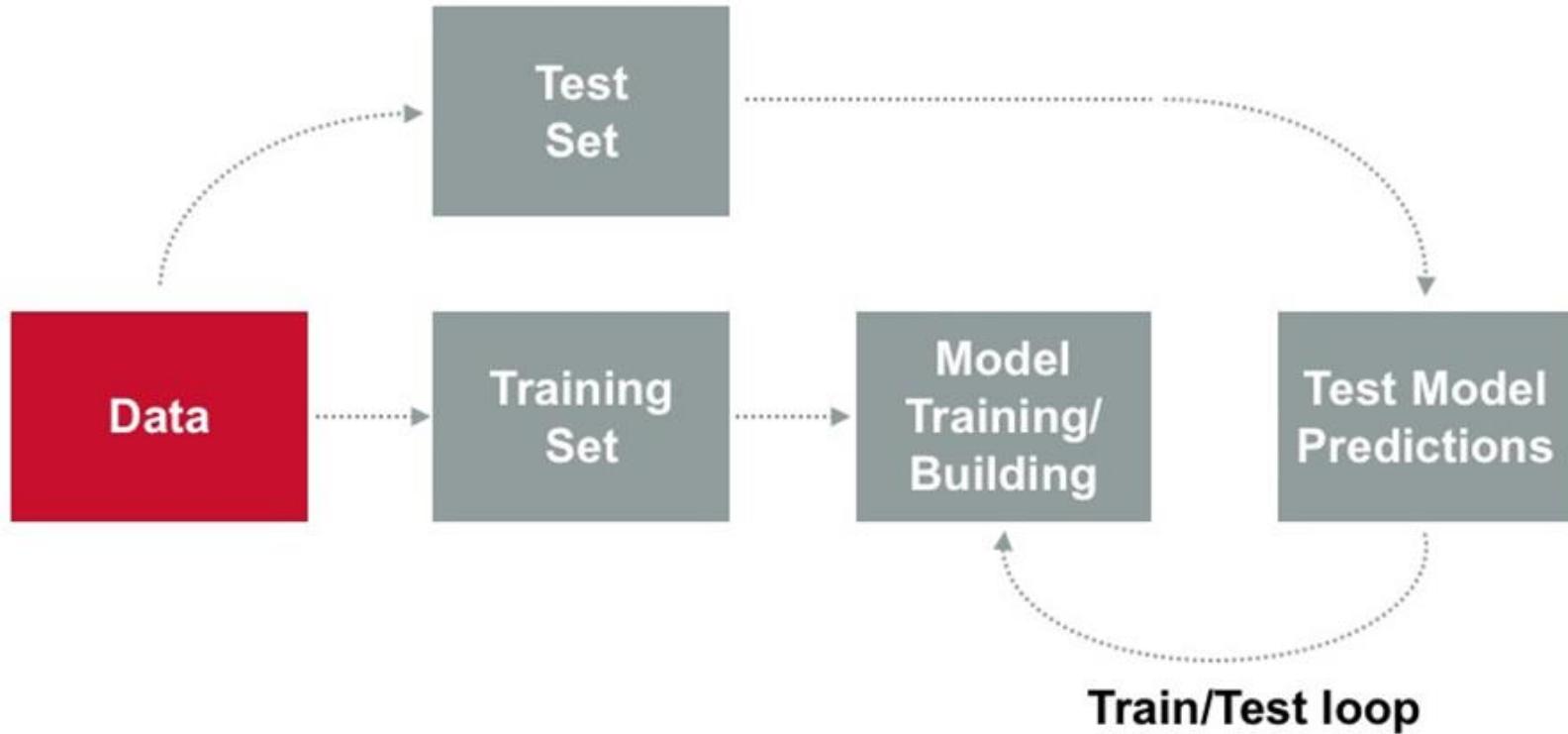
- User and Item factor matrices
- Tries to learn the hidden features of each user and item
- Algorithm alternatively fixes one factor matrix and solves for the other



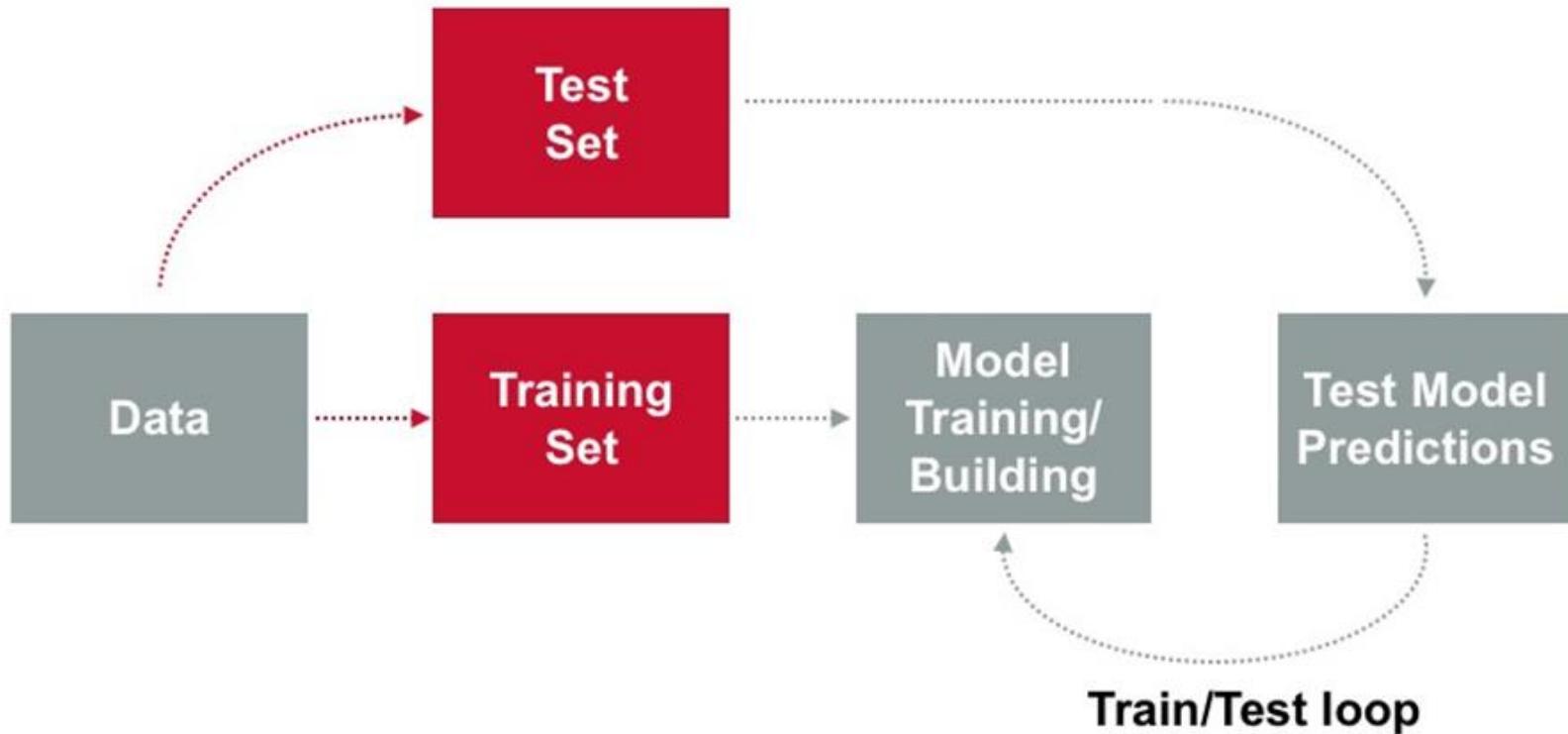
ML Cross-Validation Process



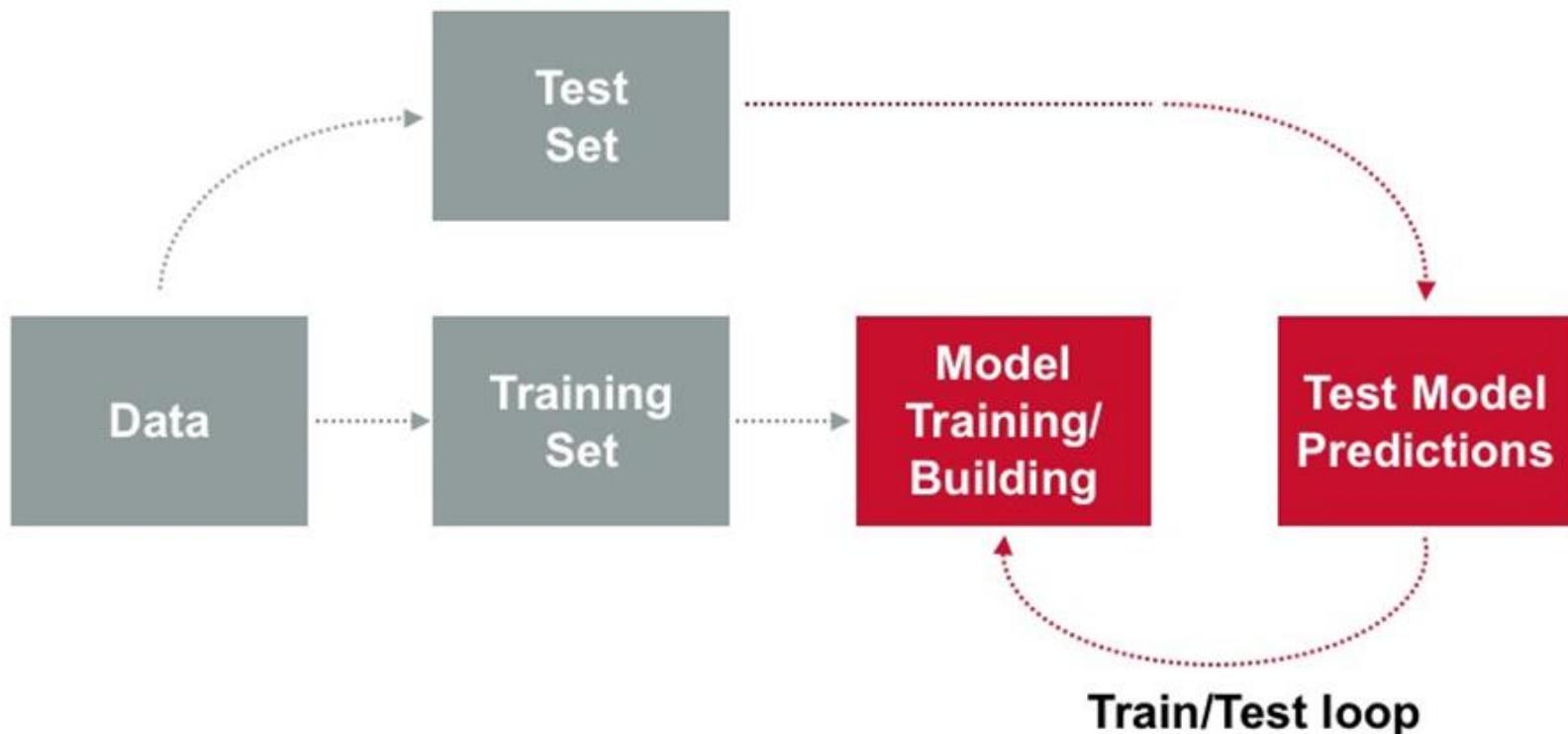
ML Cross-Validation Process



ML Cross-Validation Process



ML Cross-Validation Process



Parse Input Lines into Ratings Objects

```
// Import ml recommendation data types
import org.apache.spark.ml.evaluation.RegressionEvaluator
import org.apache.spark.ml.recommendation.ALS

//Define case class
case class Rating(userId: Int, movieId: Int, rating: Float)

// parse string: UserID::MovieID::Rating
def parseRating(str: String): Rating = {
  val fields = str.split("::")
  assert(fields.size == 4)
  Rating(fields(0).toInt, fields(1).toInt, fields(2).toFloat)
}
```

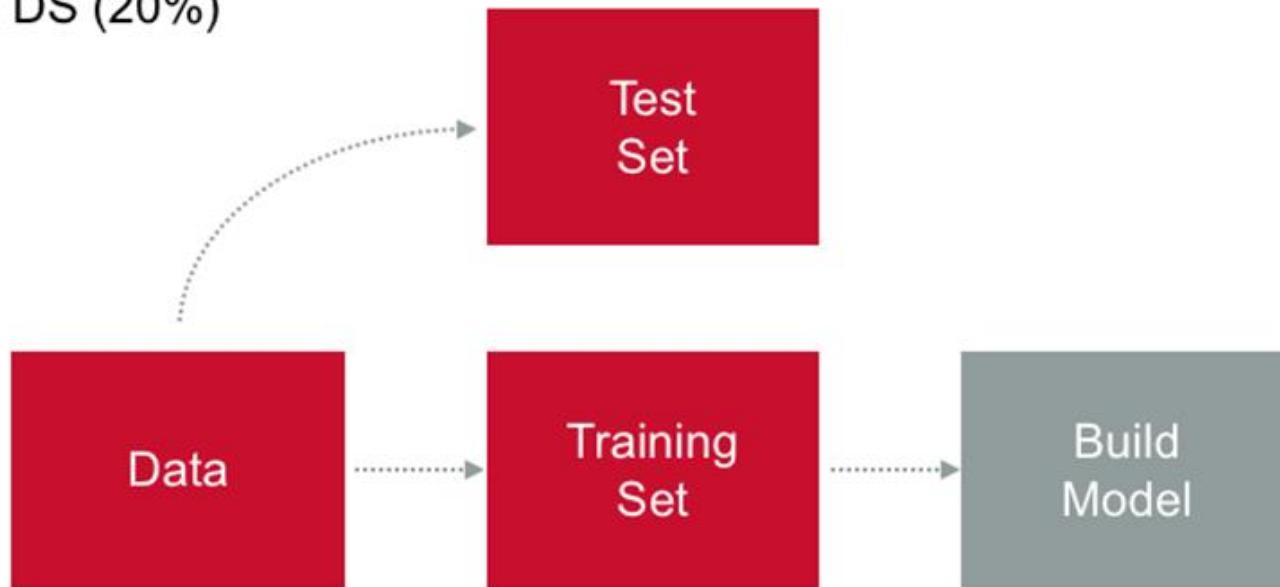
Parse Input

```
// load ratings data into a Dataset
val ratingsDS =
spark.read.textFile("/spark/lab8/ratings.dat")
.map(parseRating)
```

Build Model

Split ratingsDS into:

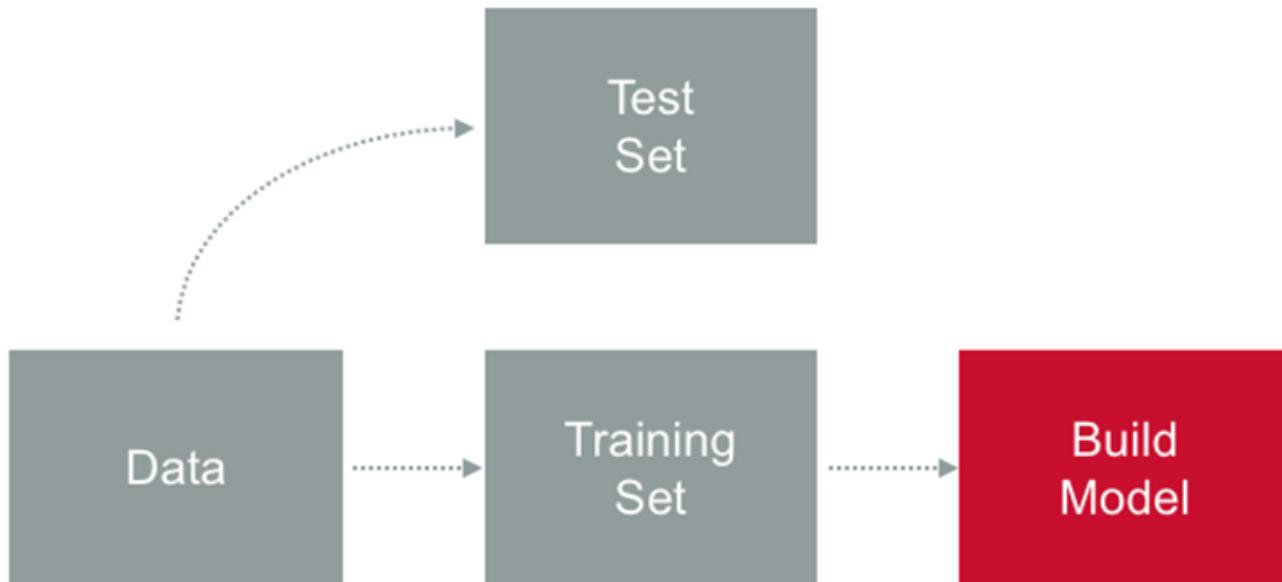
- Training data DS (80%)
- Test data DS (20%)



Build Model

```
// Randomly split ratings DS into training data DS (80%) and  
// test data DS (20%)  
val splits = ratingsDS.randomSplit(Array(0.8, 0.2), 0L)  
  
val trainingRatingsDS = splits(0).cache()  
val testRatingsDS = splits(1).cache()
```

Build Model



Build a user product matrix model

Build Model

```
// build a ALS user product matrix model with rank=20,  
iterations=10  
val model = new ALS()  
.setMaxIter(10).setRank(20).setRegParam(0.01)  
.setUserCol("userId").setItemCol("movieId")  
.setRatingCol("rating").fit(trainingRatingsDS)
```

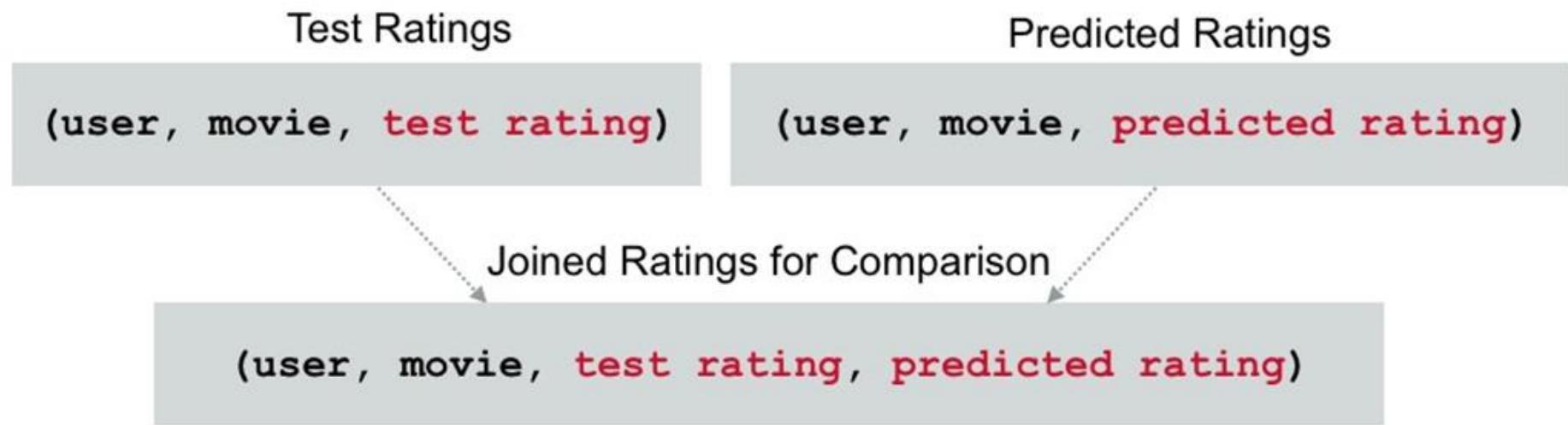
Get Predictions



Get Predictions

```
// call model.transform with test Userid, MovieId input data  
val predictionsDF = model.transform(testRatingsDS)
```

Compare Predictions to Tests



Compare Predictions to Tests

```
// Register Dataset testRatingsDS as testratings view
testRatingsDS.createTempView("testratings")

// Register DataFrame predictionsDF as predictions view
predictionsDF.createTempView("predictions")

// Join tables and compare the results using query
val compareDS = spark.sql("select t.userid, t.movieid, t.rating,
p.prediction from testratings t, predictions p where t.userid =
p.userid and t.movieid = p.movieid")
```

Compare Predictions to Tests

```
// Check results of compareDS  
compareDS.show(5)
```

```
scala> compareDS.show(5)  
+----+----+----+----+  
|userid|movieid|rating|prediction|  
+----+----+----+----+  
|    53|    148|    5.0|  4.388935|  
|  1605|    148|    2.0| 2.3322415|  
|  2507|    148|    4.0|  3.632678|  
|  3650|    463|    2.0|  2.505344|  
|  5306|    463|    2.0|  3.174282|  
+----+----+----+----+  
only showing top 5 rows
```

Compare Predictions to Tests

Find false positives where:

```
test rating <= 1 and predicted rating >= 4
```

```
compareDS
```

```
(user, movie, test rating, predicted rating)
```

```
// Register Dataset compareDS as compare view  
compareDS.createTempView("compare")
```

Test Model

```
spark.sql("select userid, movieid, rating, prediction from compare where rating<=1 and prediction>=4") .show(5)
```

| userid | movieid | rating | prediction |
|--------|---------|--------|------------|
| 1645 | 1088 | 1.0 | 5.8135347 |
| 1808 | 1088 | 1.0 | 4.23835 |
| 4011 | 1342 | 1.0 | 4.4337964 |
| 4342 | 1959 | 1.0 | 4.641789 |
| 1017 | 2659 | 1.0 | 4.3406224 |

only showing top 5 rows

Knowledge Check



Place the steps below in the order to correctly describe a supervised learning workflow.

- Use the validated model with new data
- Build, test and tune the model
- Load sample data
- Split the data into training and data sets
- Extract features

Lab 8.2 – Use Apache Spark MLlib



- Estimated time to complete: **60 minutes**
- In this lab, you will use the Spark shell to load and inspect data, make movie recommendations, and analyze a simple flight example using decision trees.