

# Spark





# Table of Contents

- 0: Welcome to Class:
  - 1: Introduction to Apache Spark:
  - 2: Create Datasets:
  - 3: Apply Operations on Datasets:
  - 4: Build a Simple Apache Spark Application:
  - 5: Monitor Apache Spark Applications:
  - 6: Create an Apache Spark Streaming Application:
  - 7: Use Apache Spark Graph Frames:
  - 8: Use Apache Spark MLLib, the machine learning library:
- 

# Lesson 0: Welcome to Class



# Welcome!

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

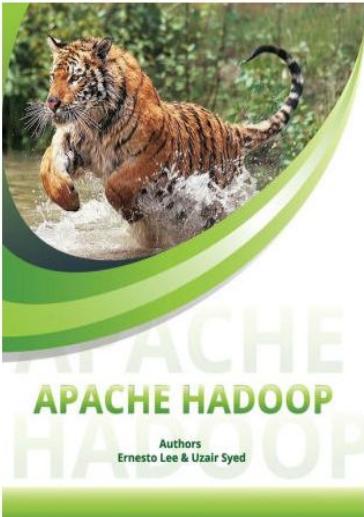
barnesandnoble.com/w/apache-hadoop-ernesto-lee/1128144600?ean=9781942864004

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

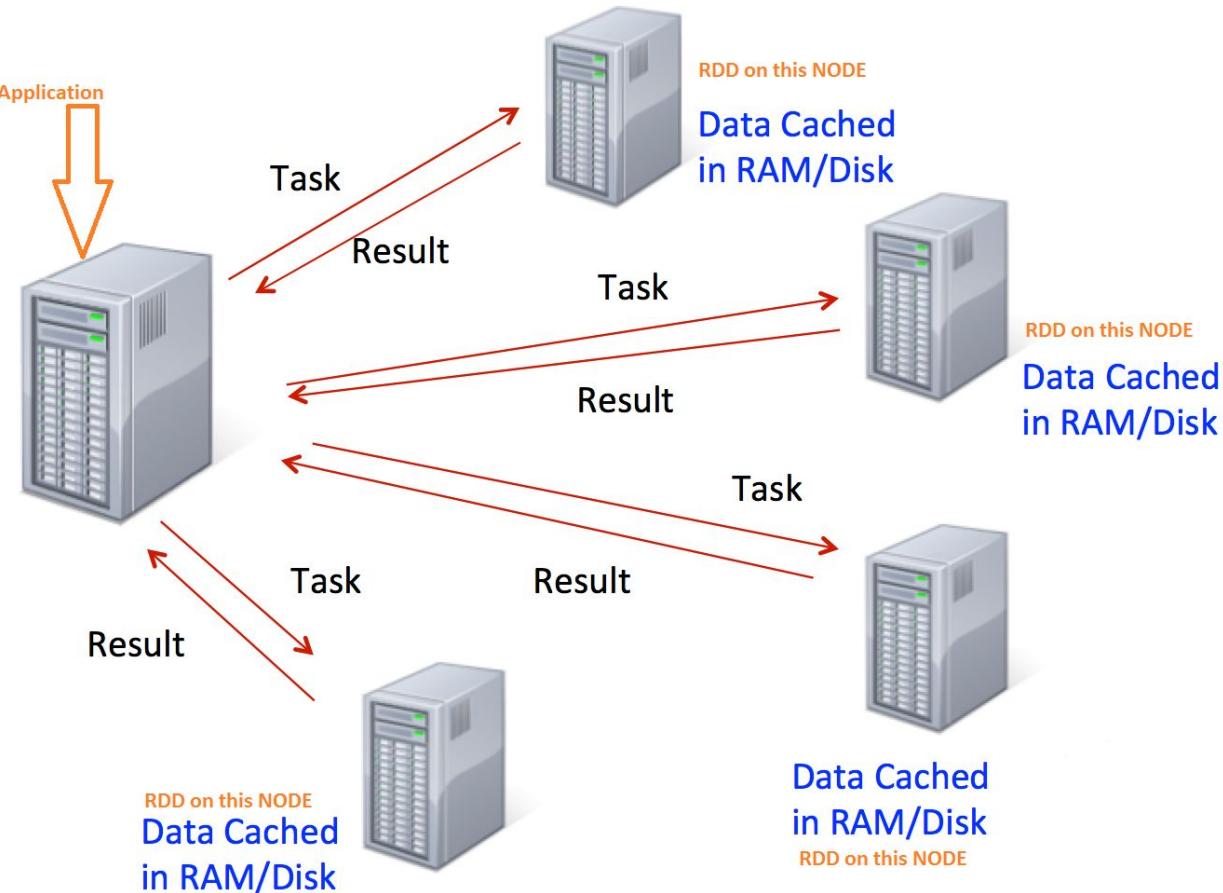
- Everyone will have their own lab environment.
- <http://LLLLL##.eastus.azurecontainer.io>
- Replace LLLLL with the word provided by your instructor.
- ## will be the number assigned to your specific container
- These containers are ephemeral!!!

# Lesson 1: Introduction to Apache Spark



# 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



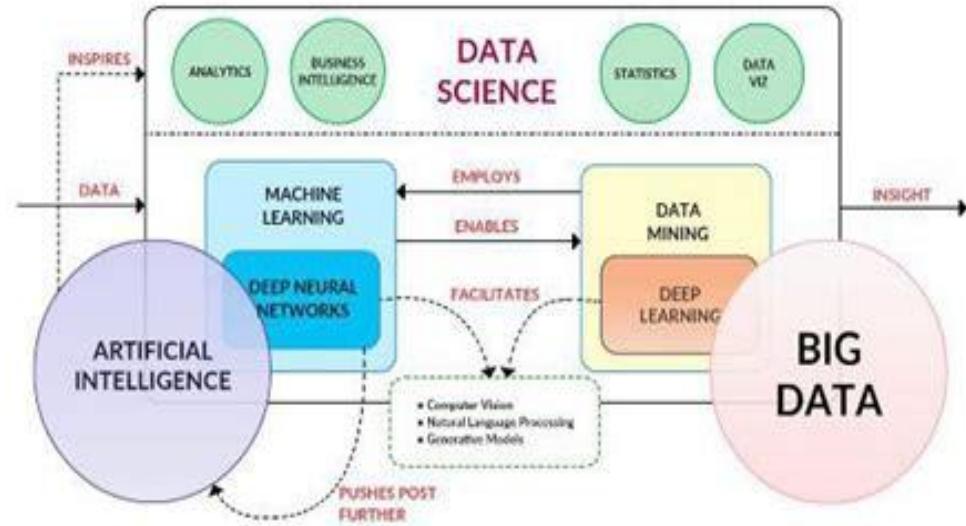
# Why Use Apache Spark?

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



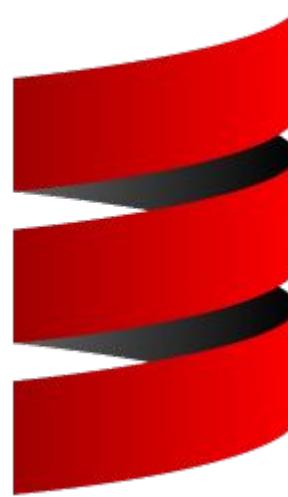
# Why Use Apache Spark?

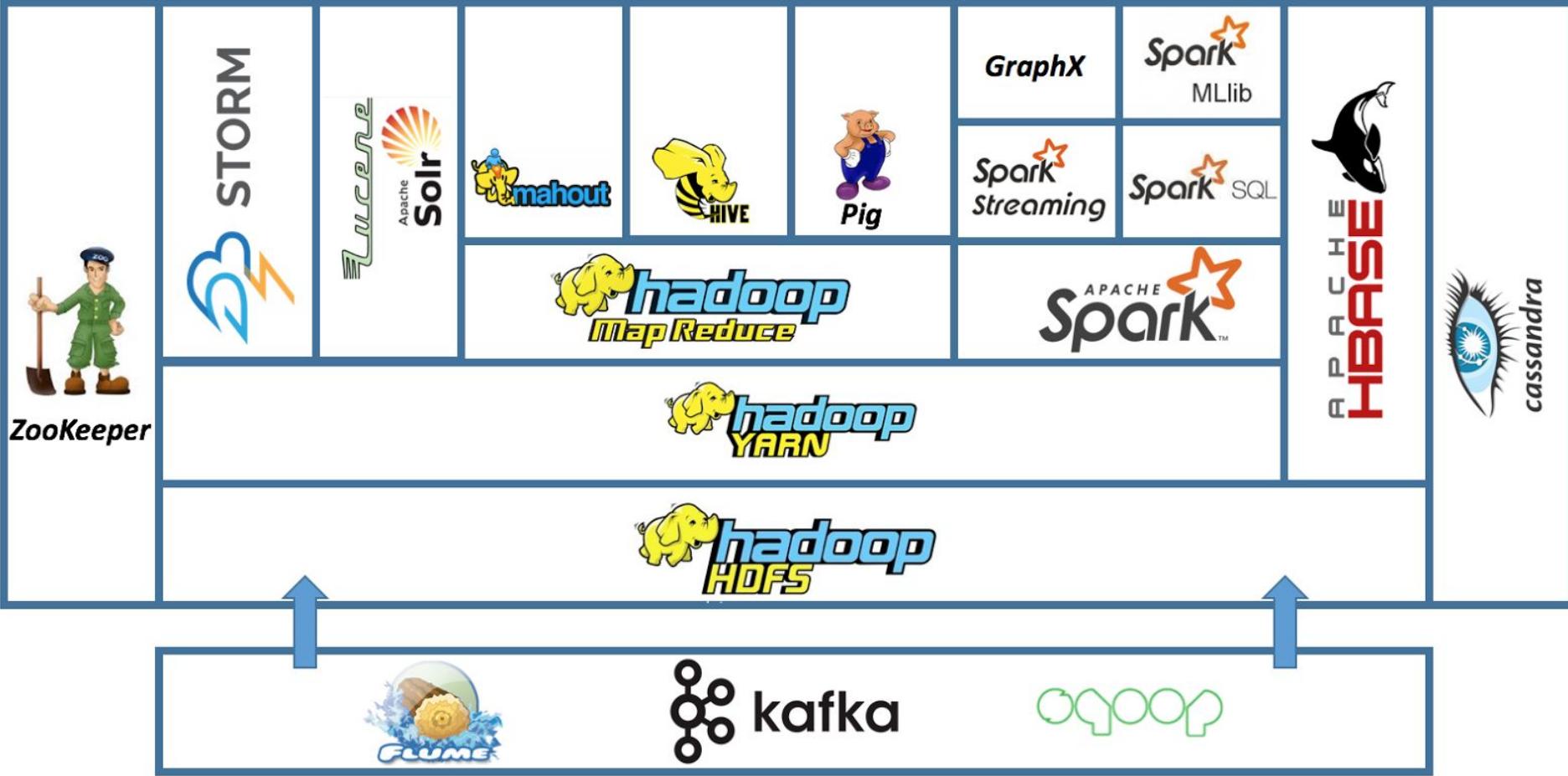
- Performance
- Development
- Deployment
- Cohesive Stack
  - Multiple architectural options
  - Batching
  - AI
  - Stream Use Cases
- Language Independence



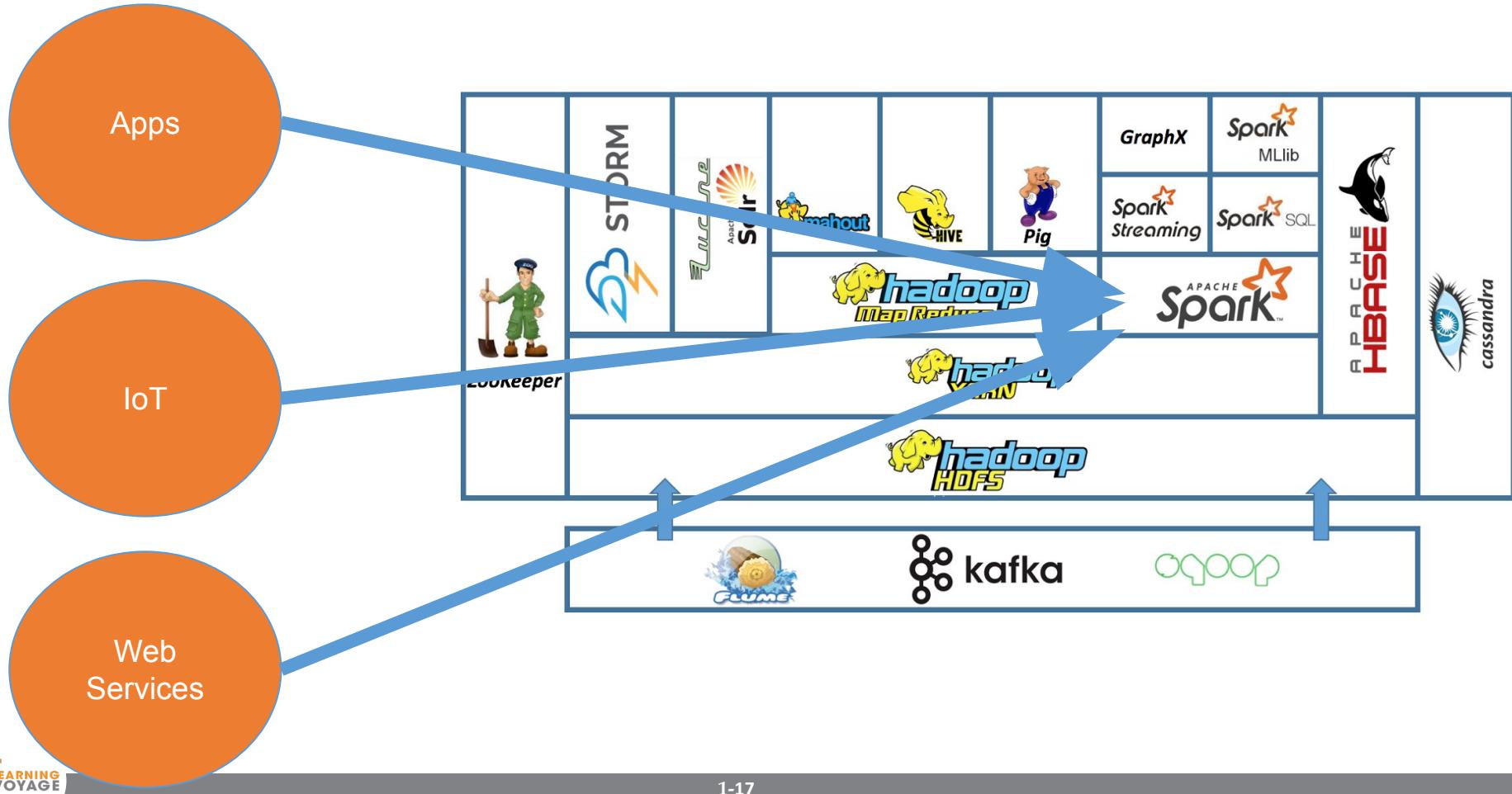
# Why Use Apache Spark?

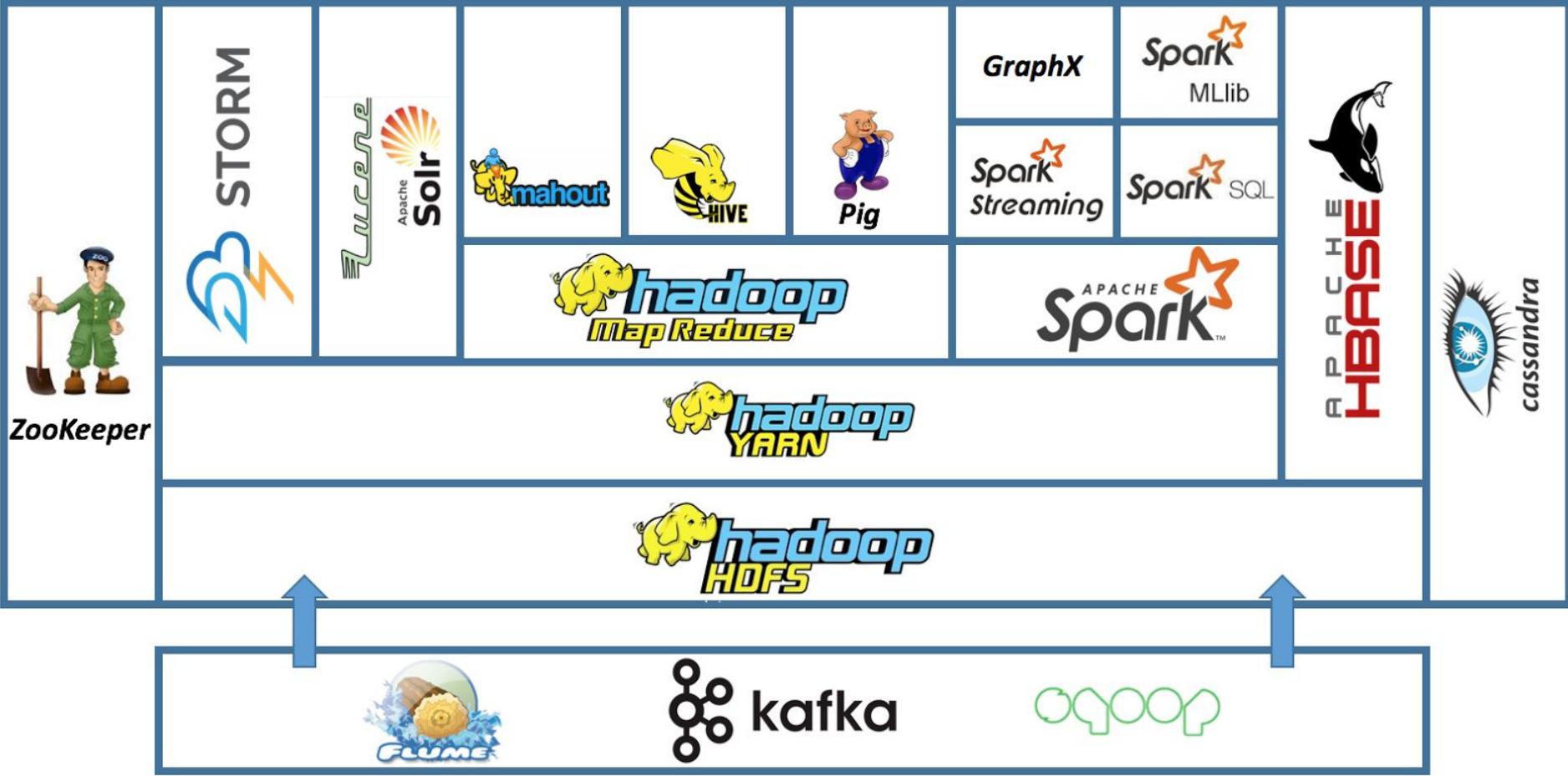
- Performance
- Development
- Deployment
- Cohesive Stack
- Language Independence
  - Java
  - Scala
  - Python (PySpark)



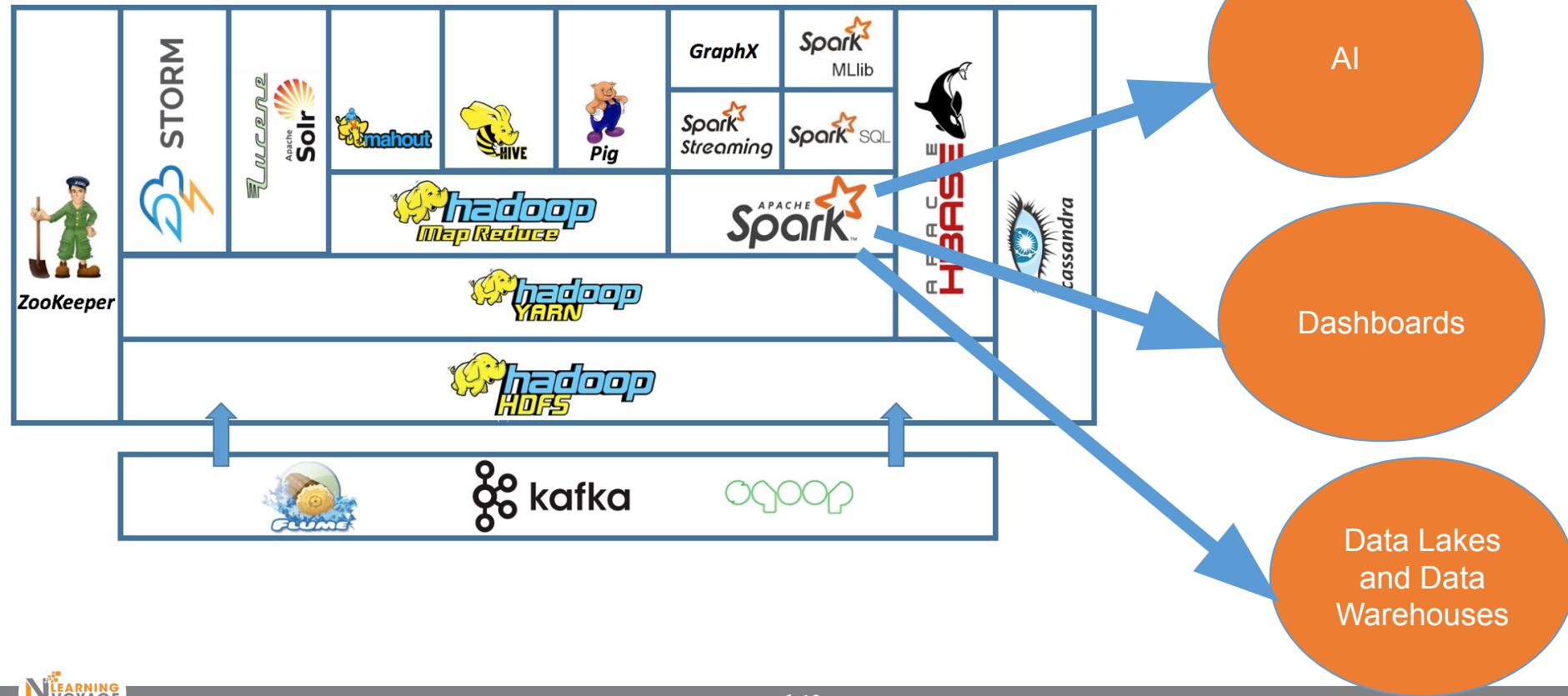


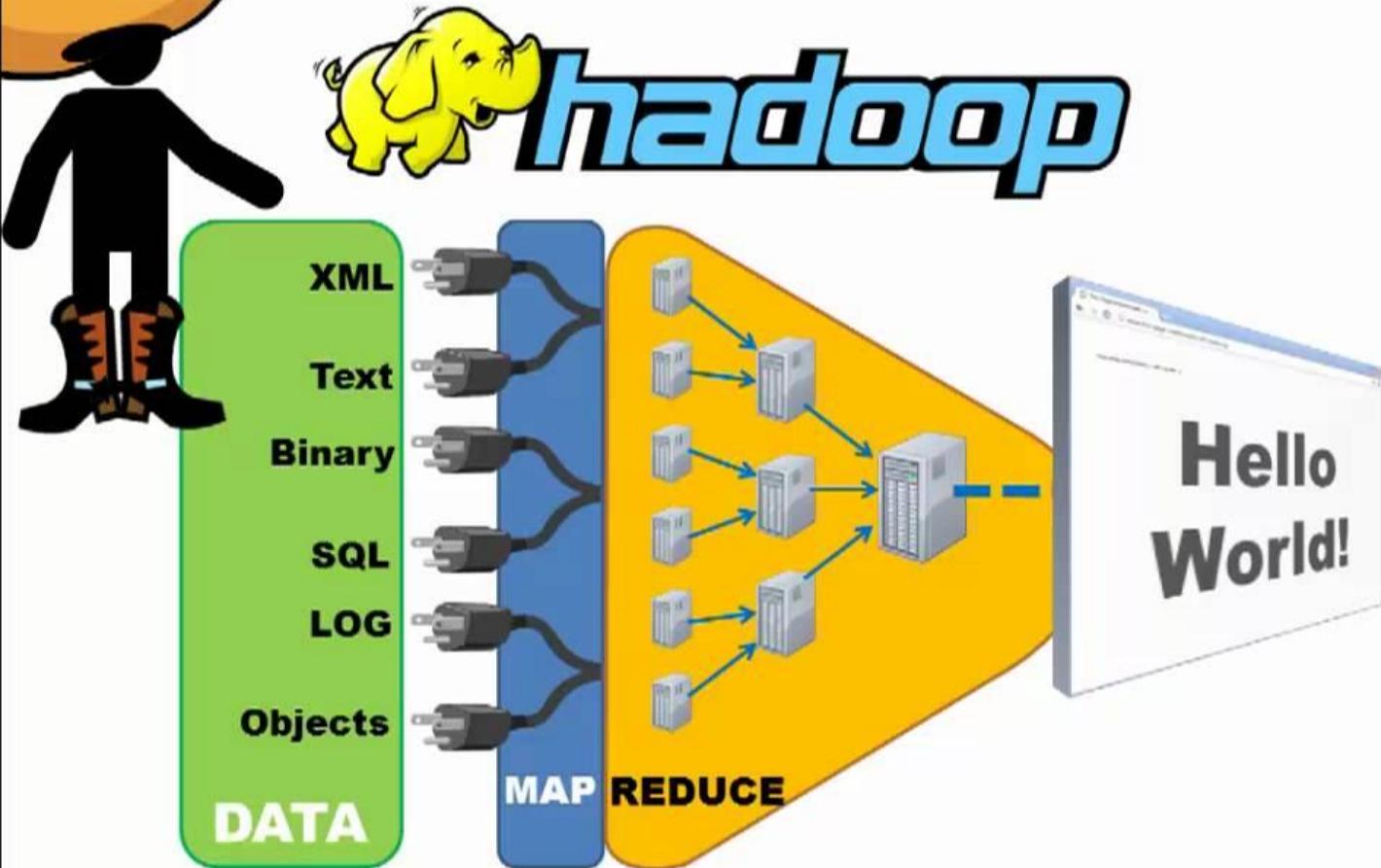
# INPUTS





# OUTPUTS



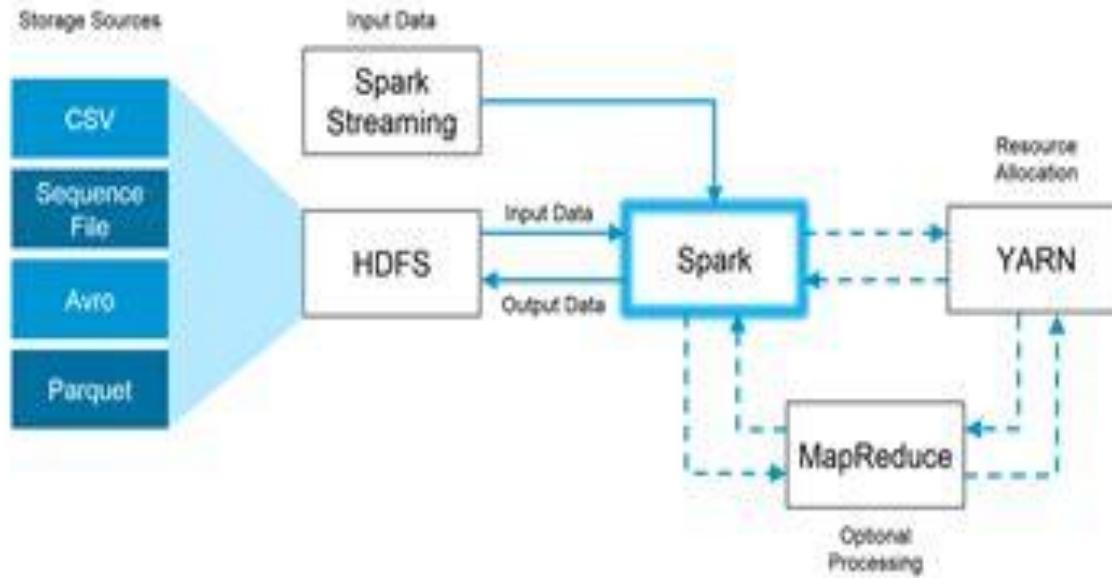


# Spark

---

- 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

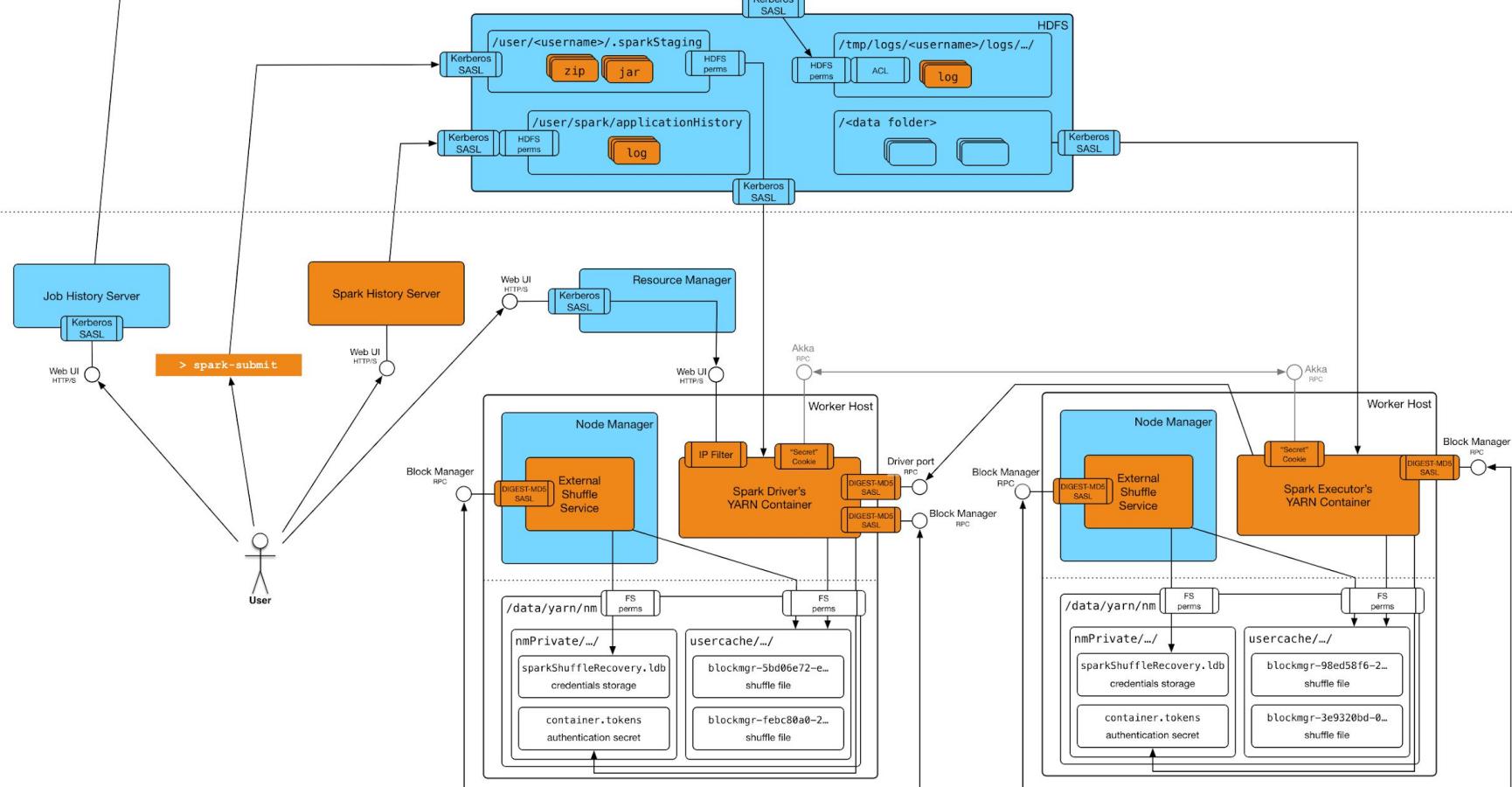
---



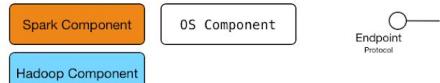
What are the advantages of using Apache Spark?

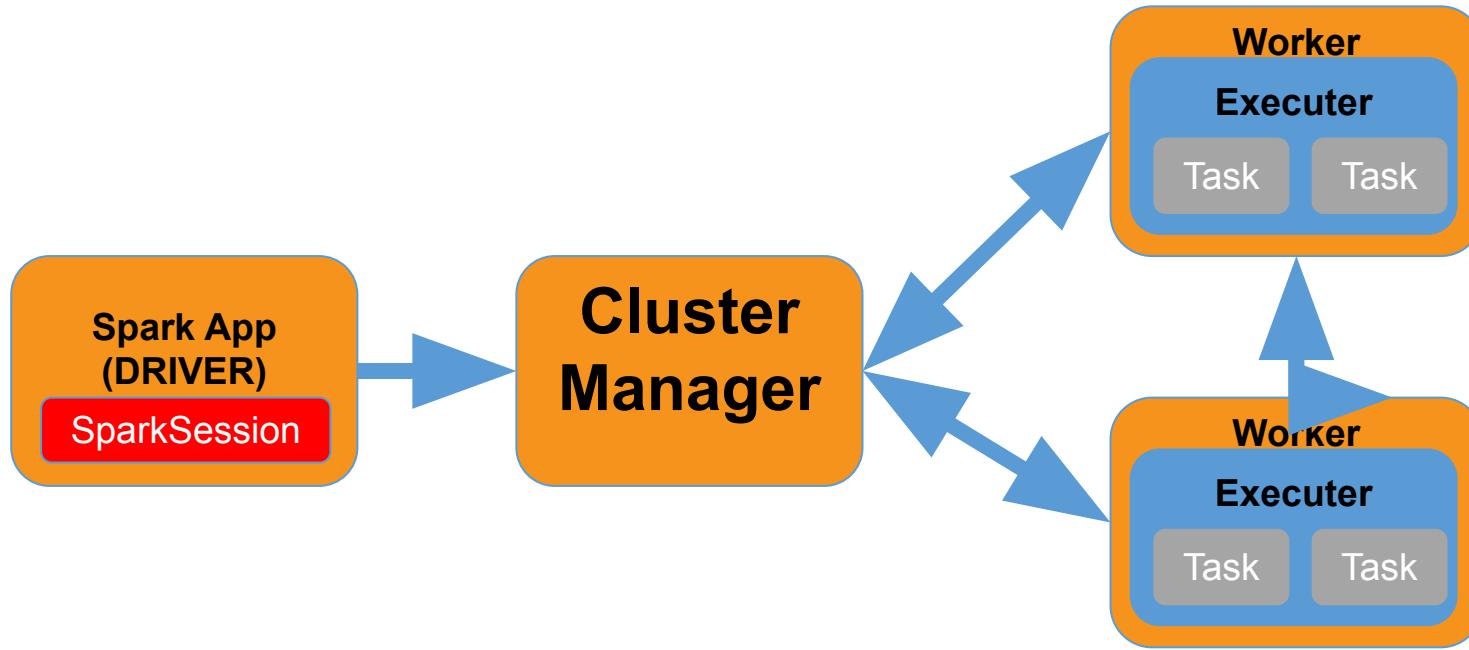
- A. Compatible with Hadoop
- B. Ease of development
- C. Fast
- D. Multiple language support
- E. Unified stack
- F. All of the above

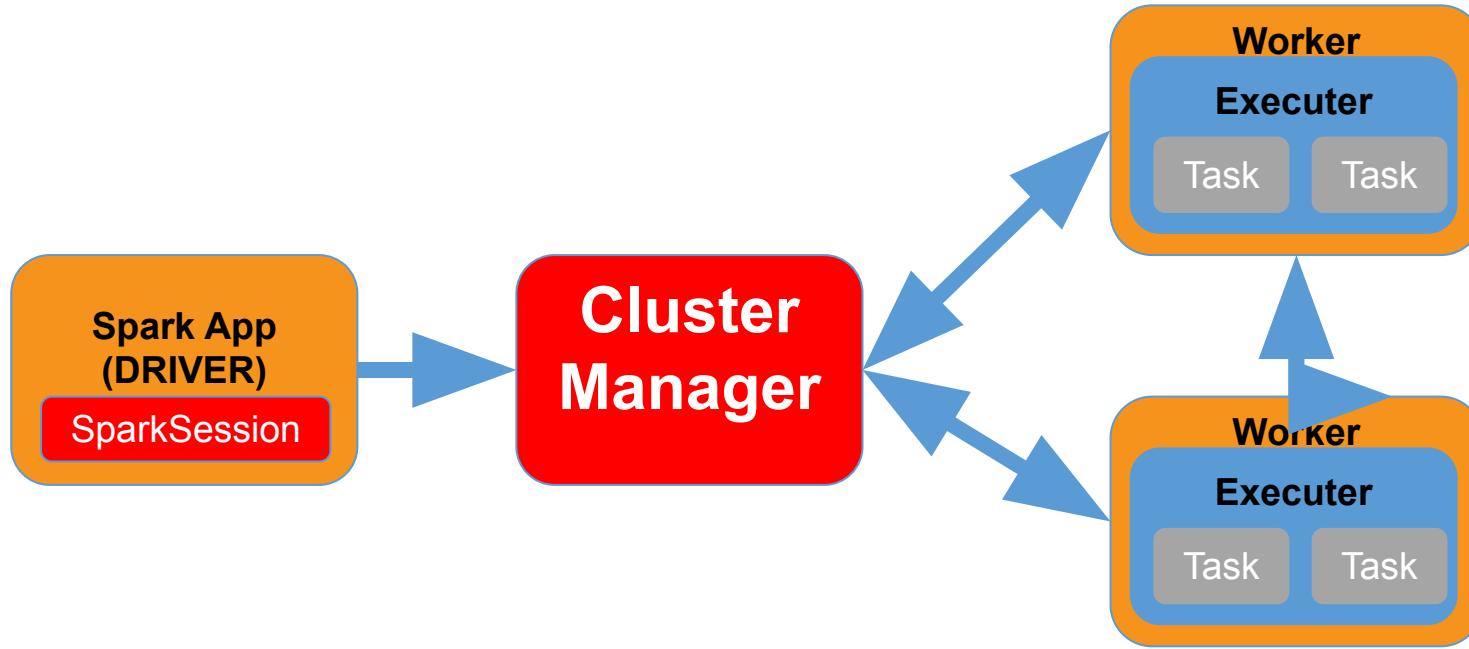
Service Level

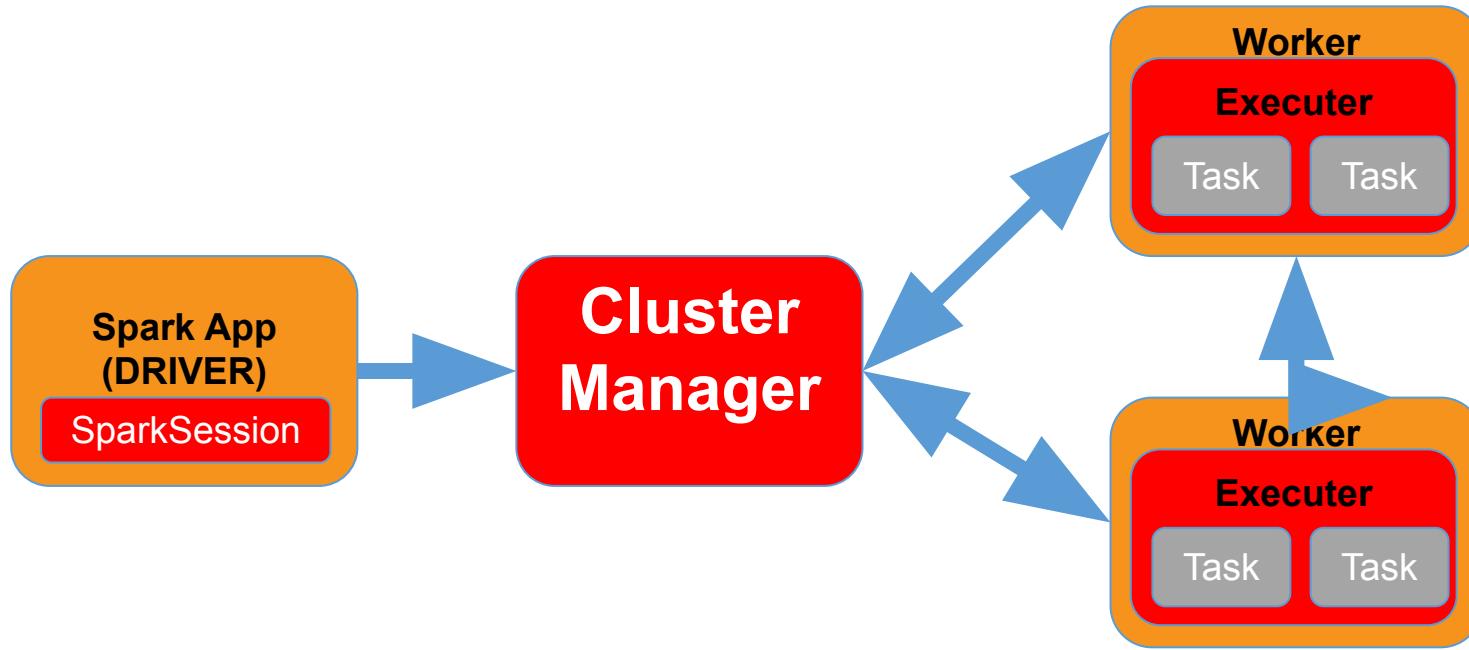


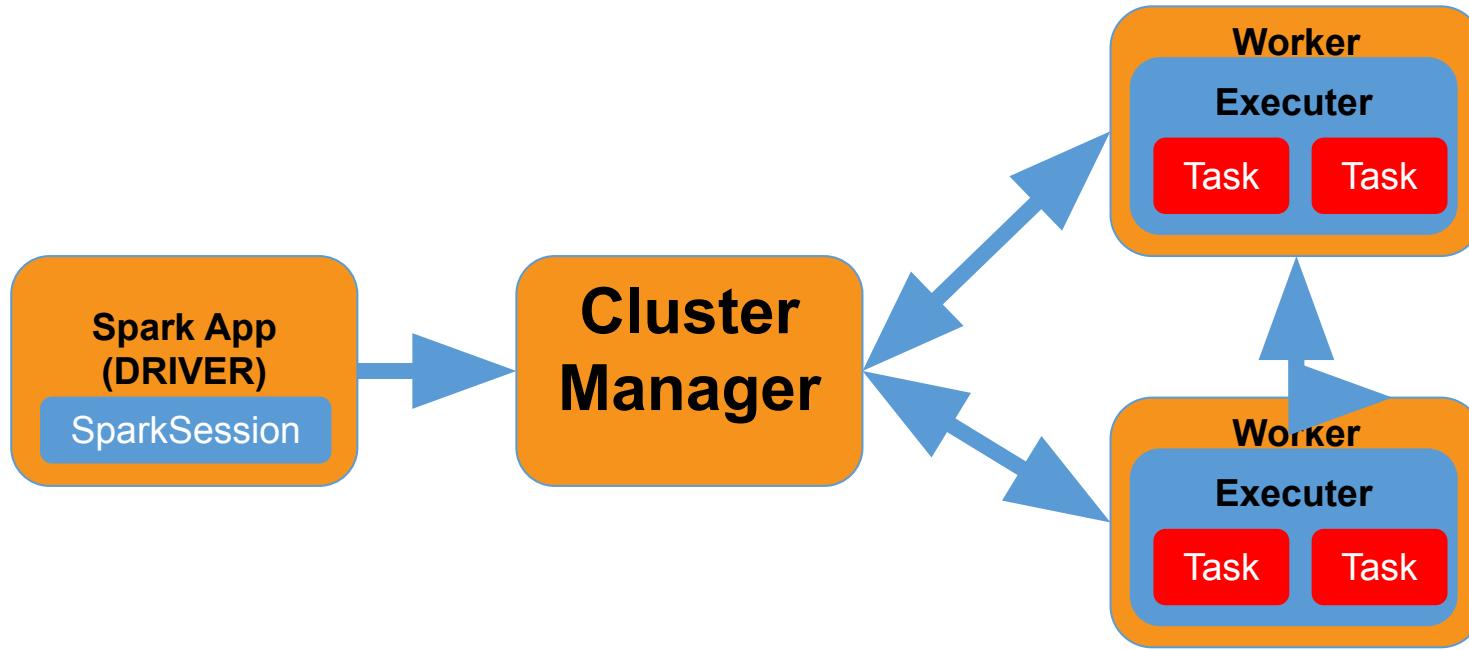
Legend









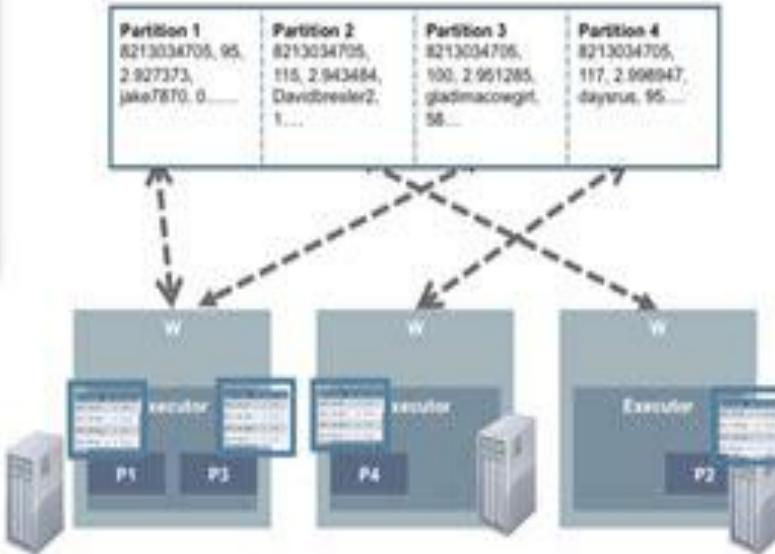


- Read only collection of **typed objects** `Dataset[T]`
- **Partitioned** across a cluster
- Operated on in **parallel**
- in memory can be **Cached**

**partitioned**

Partition	Executor	Data
P1	E1	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
P2	E1	11, 12, 13, 14, 15, 16, 17, 18, 19, 20
P3	E2	21, 22, 23, 24, 25, 26, 27, 28, 29, 30
P4	E2	31, 32, 33, 34, 35, 36, 37, 38, 39, 40

## Dataset



## Transformations

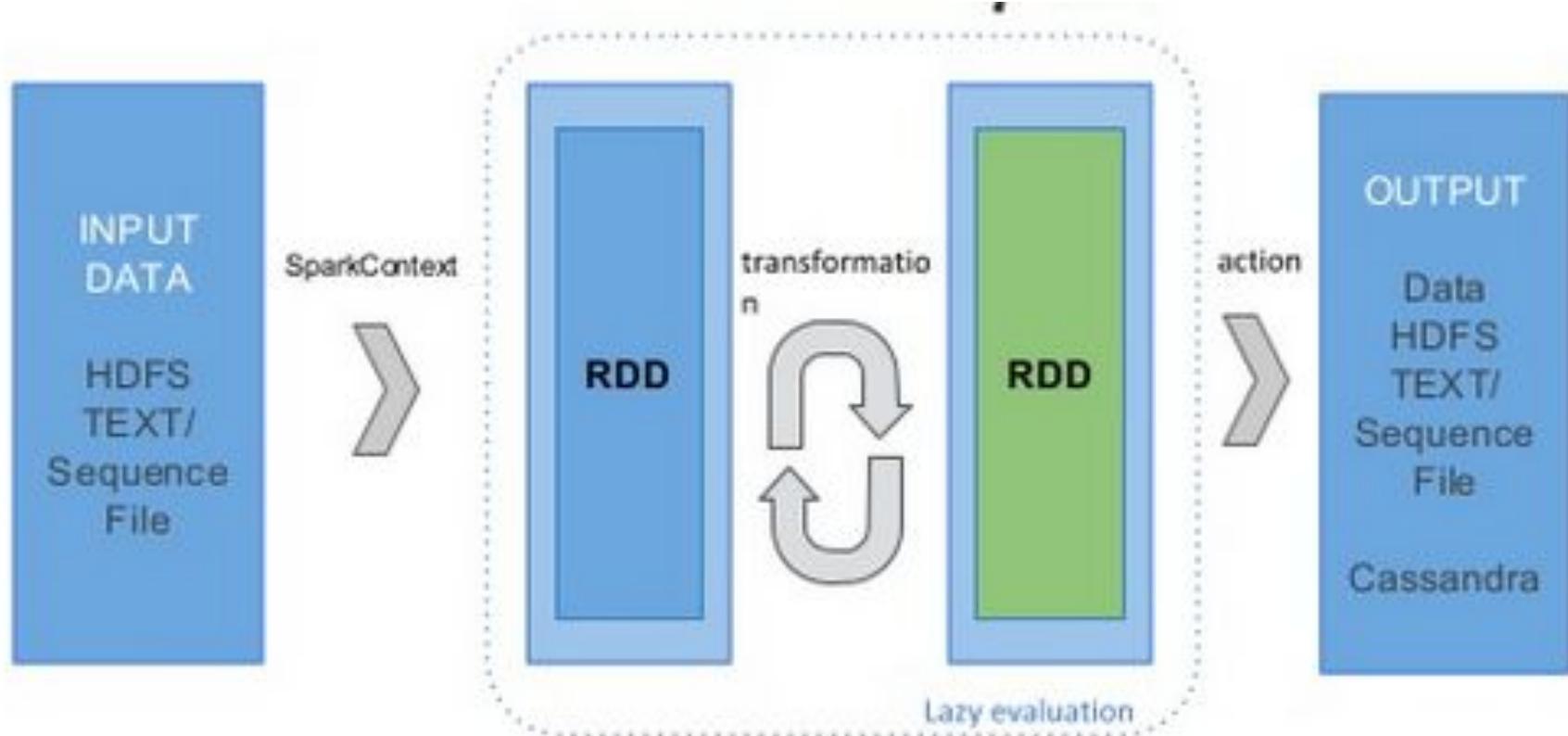
`map(func)`  
`flatMap(func)`  
`filter(func)`  
`groupByKey()`  
`reduceByKey(func)`  
`mapValues(func)`

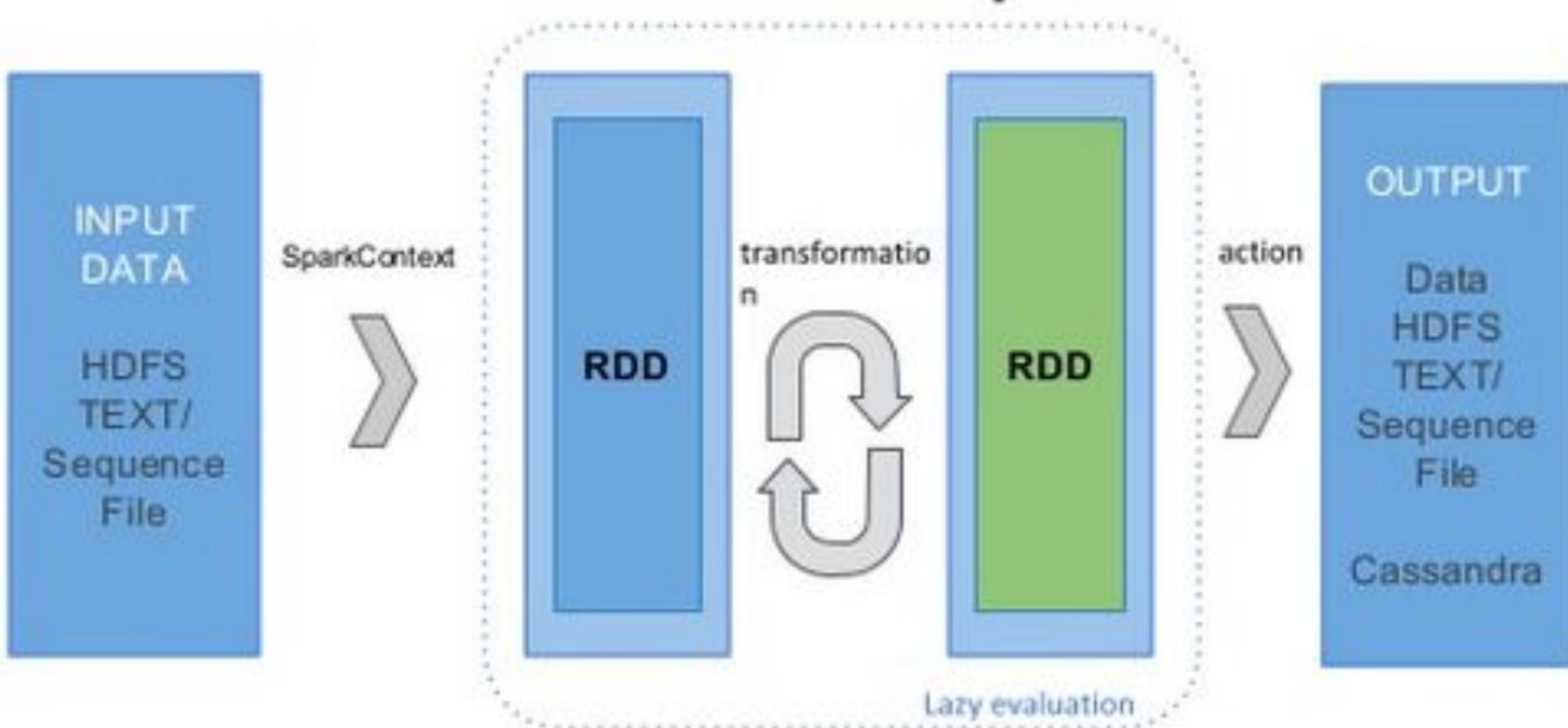
...

## Actions

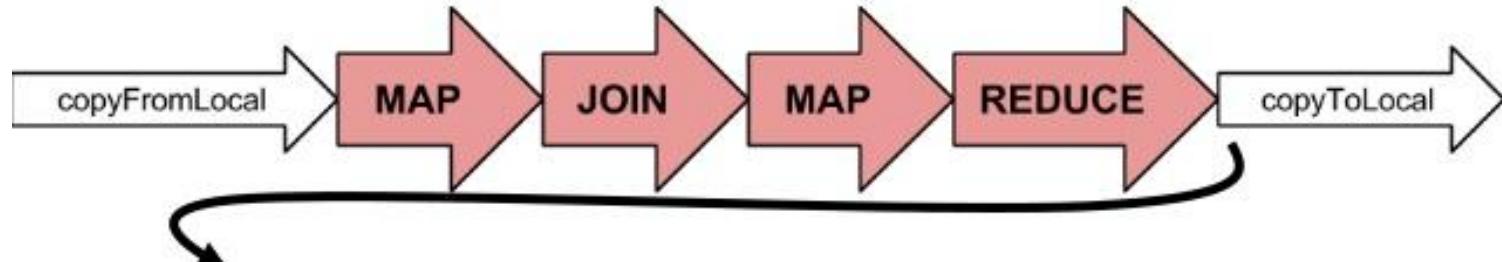
`take(N)`  
`count()`  
`collect()`  
`reduce(func)`  
`takeOrdered(N)`  
`top(N)`

...





# 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'>
```

# Spark and Big Data

Data Sources	Big Data Application Stack	Output
IoT	Batch/ETL Processing: <b>Spark, MapReduce, Hive, Pig</b>	Dashboard
	Stream Processing: <b>Spark Streaming, MapR-ES, Storm</b>	
Apps	Machine Learning: <b>Spark MLLib, Mahout</b>	Query/ Advanced Analytics
	Queries: <b>Spark SQL, Drill, Hive</b>	
Web Services	Graph Processing: <b>Spark GraphFrames, Giraph, Graphlab</b>	Enterprise Data Warehouse

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

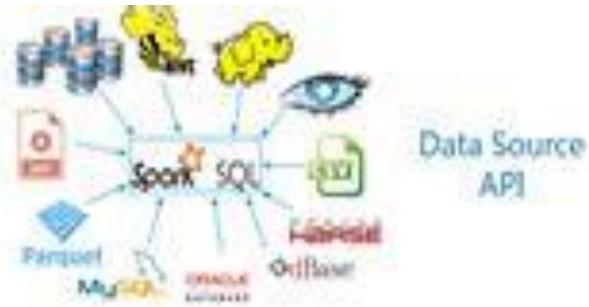
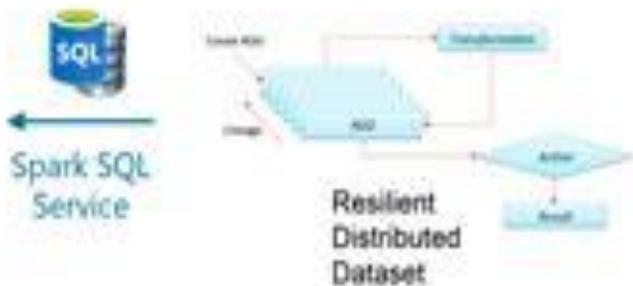
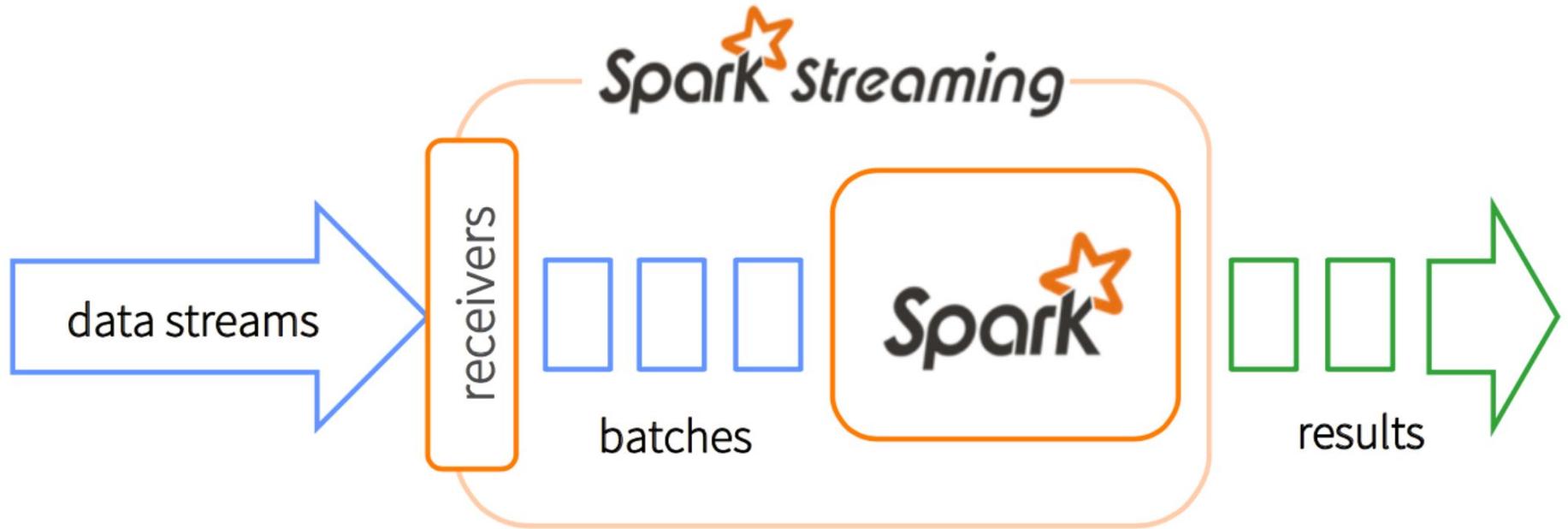
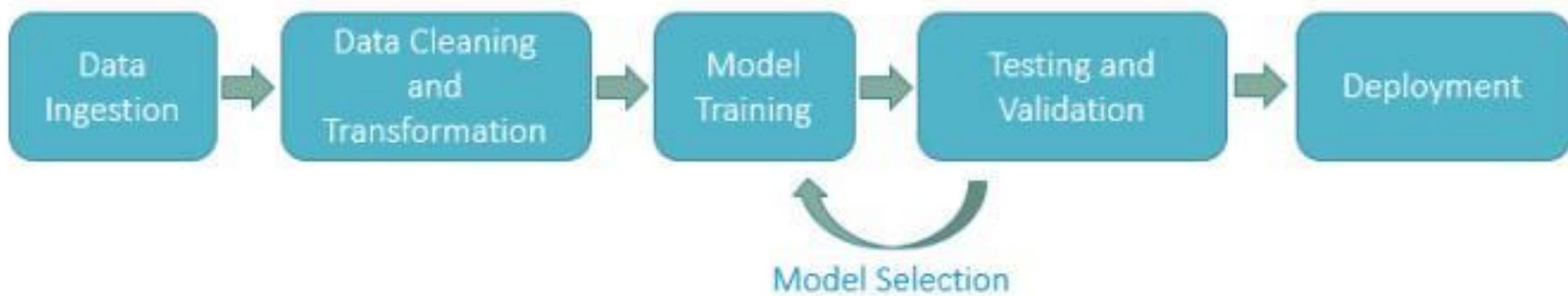
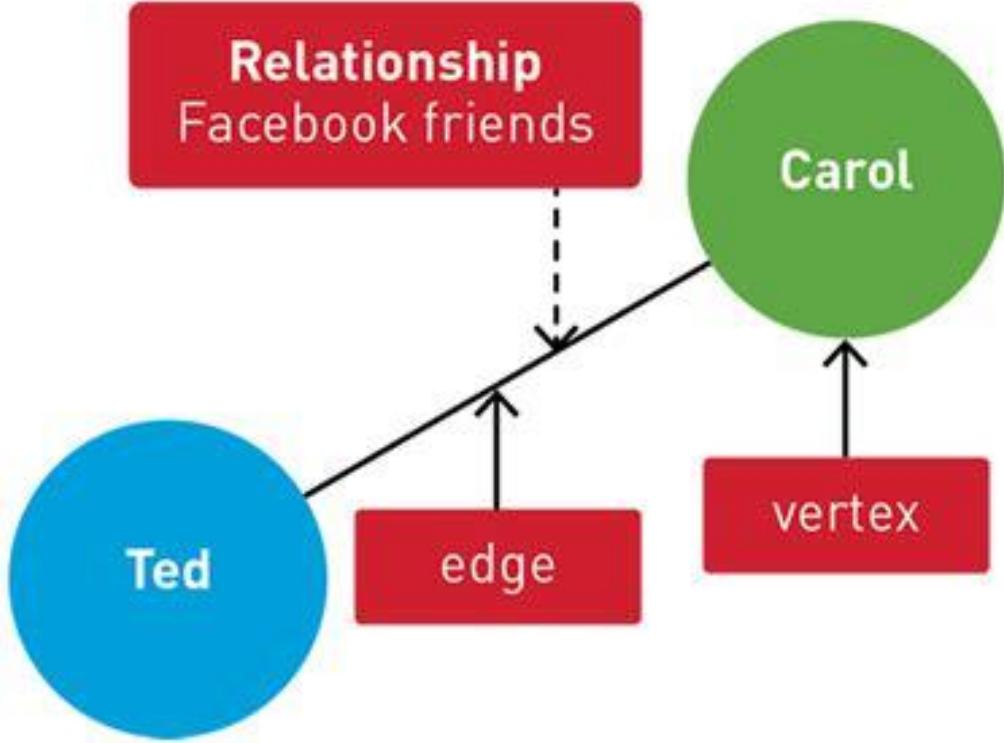


Figure: The flow diagram represents a Spark SQL process using all the four libraries in sequence.



# APACHE Spark™ ML





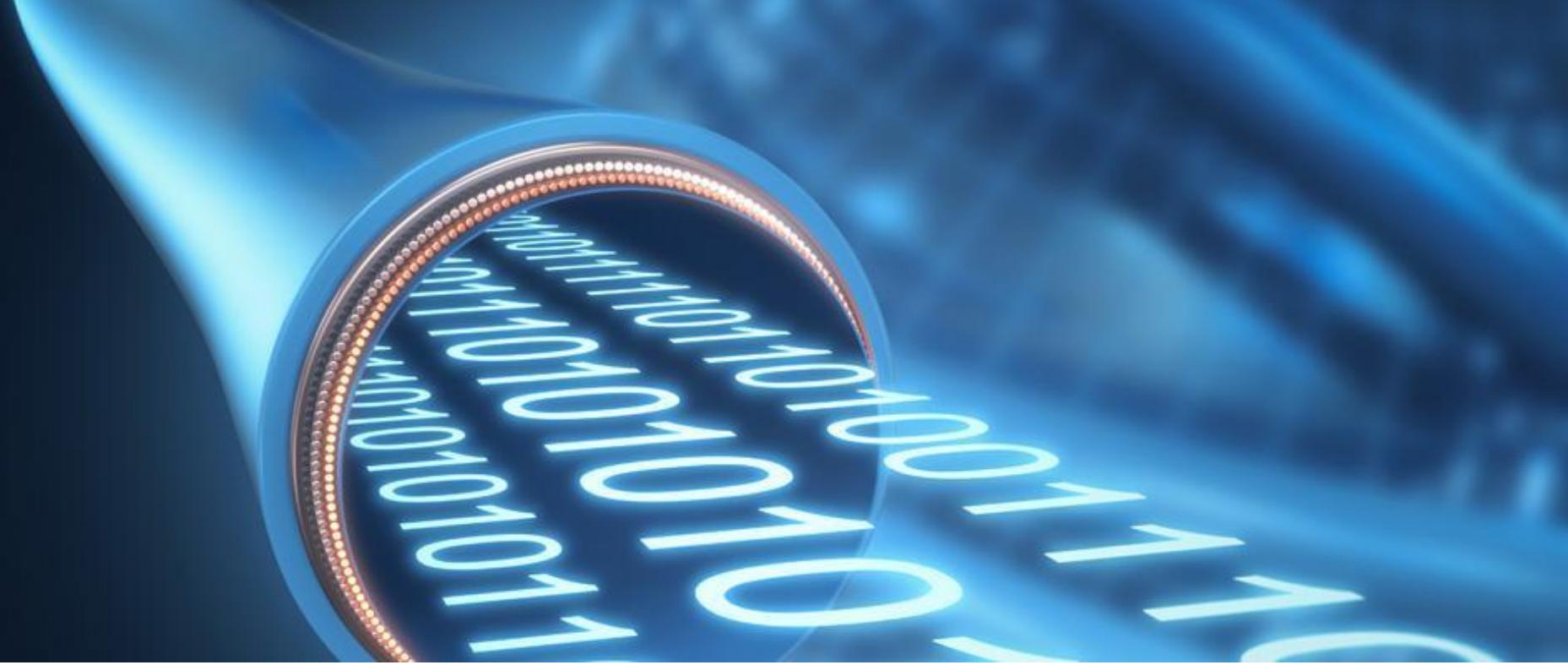
# Knowledge Check



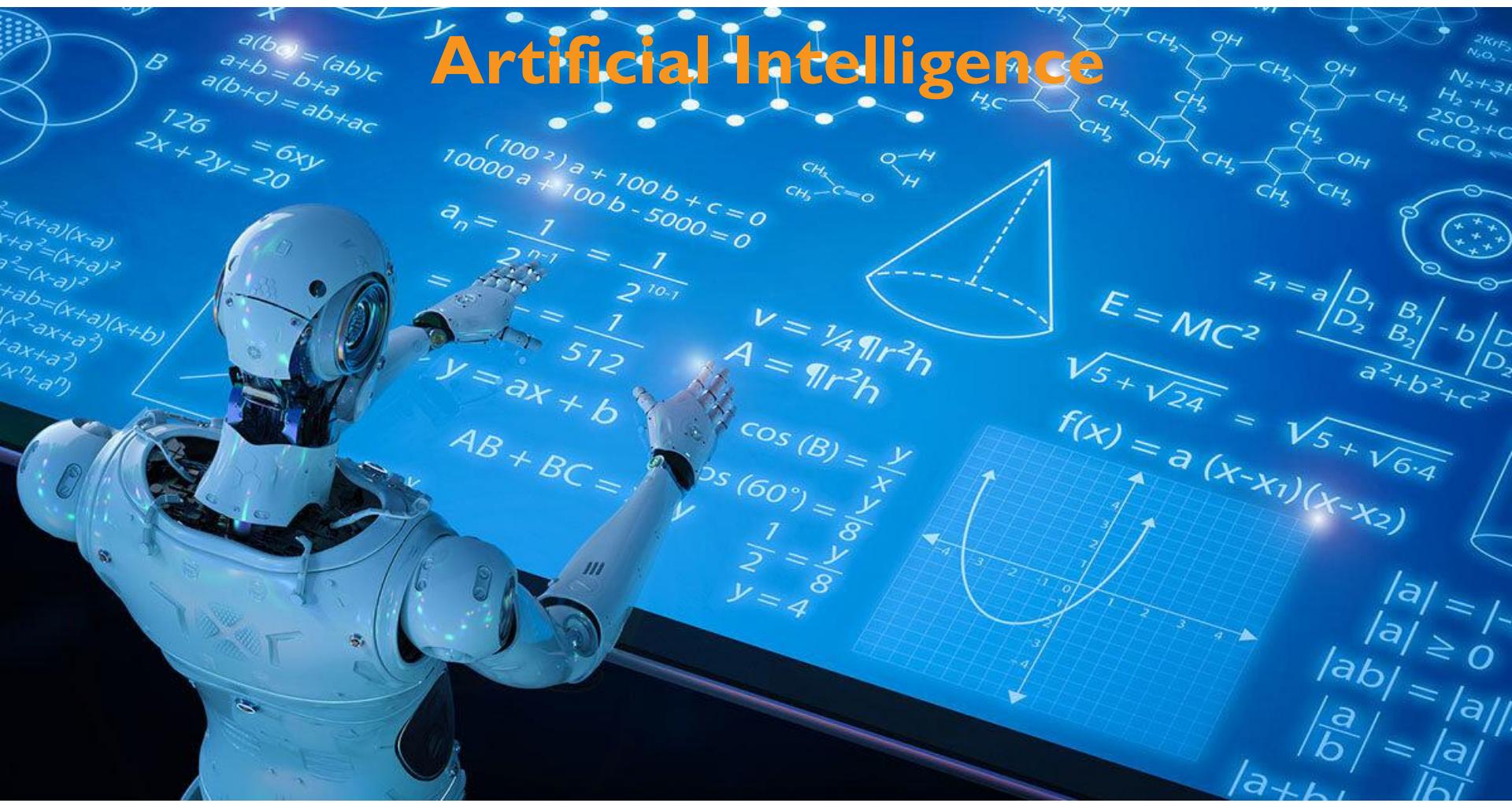
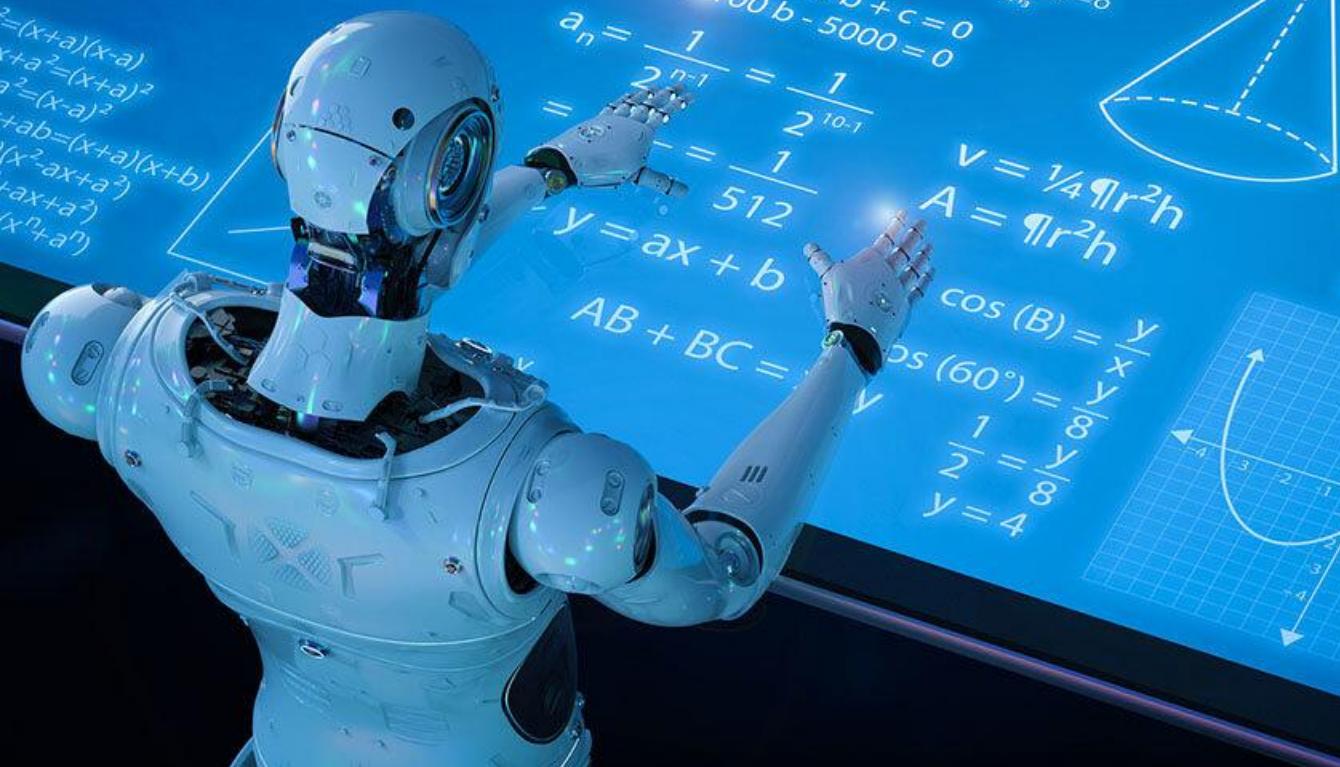
Match the following:

- 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

Thousands

Millions

Billions





## 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



elasticsearch.

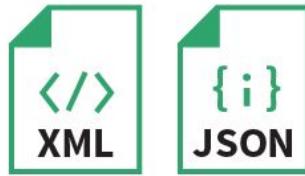


and more ..

## UNSTRUCTURED



## SEMI-STRUCTURED



## STRUCTURED

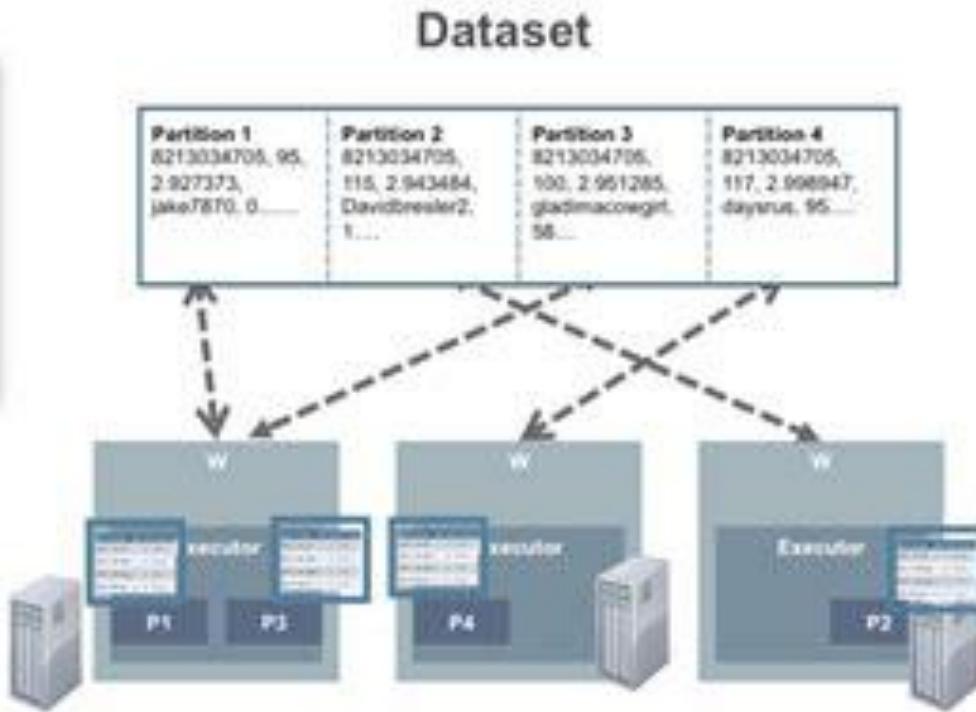
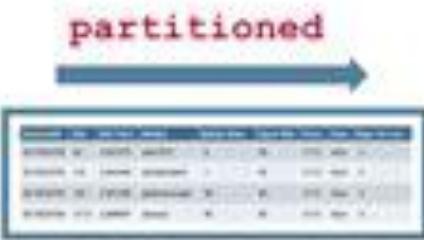


More flexible

More efficient storage and performance

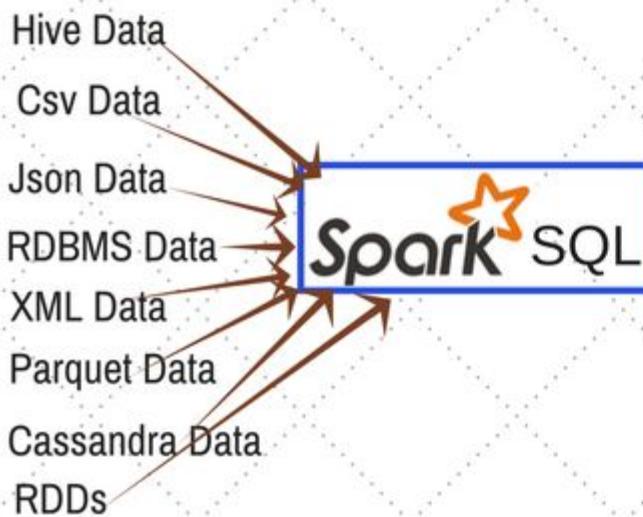


- Read only collection of **typed objects** **Dataset[T]**
- **Partitioned** across a cluster
- Operated on in **parallel**
- in memory can be **Cached**



- 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

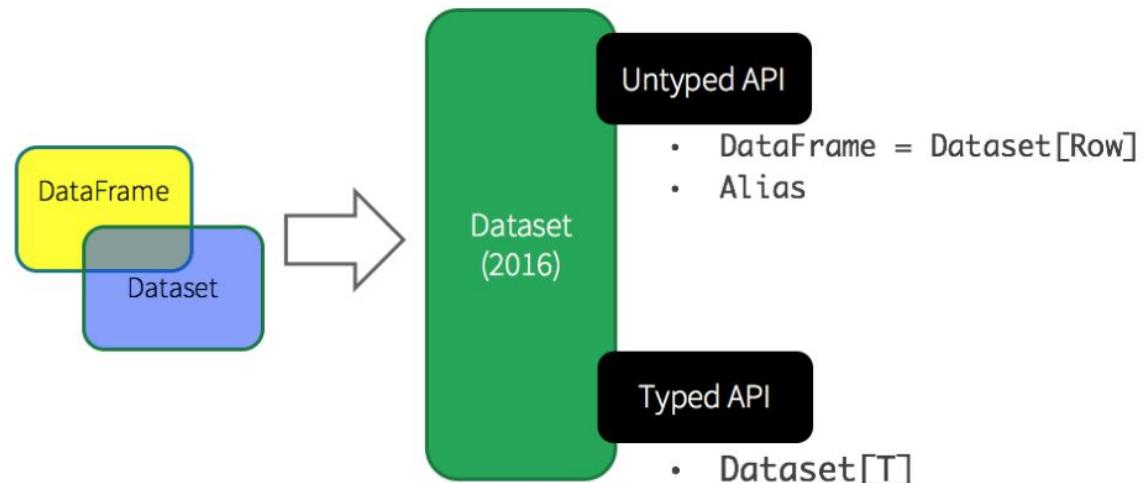
- **Dataframe**

- Unknown Schema
- Can be assigned programmatically

- **Dataset**

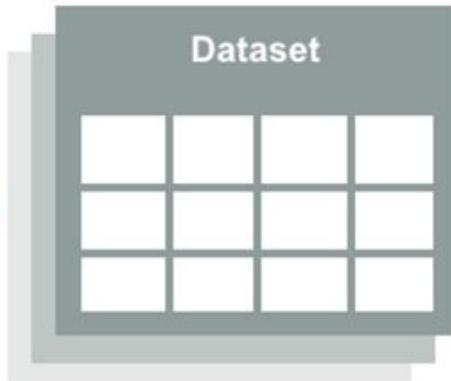
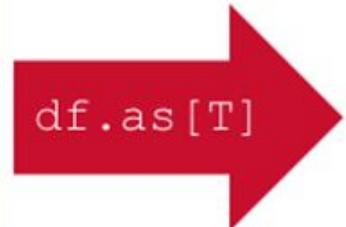
- Known schema

Unified Apache Spark 2.0 API



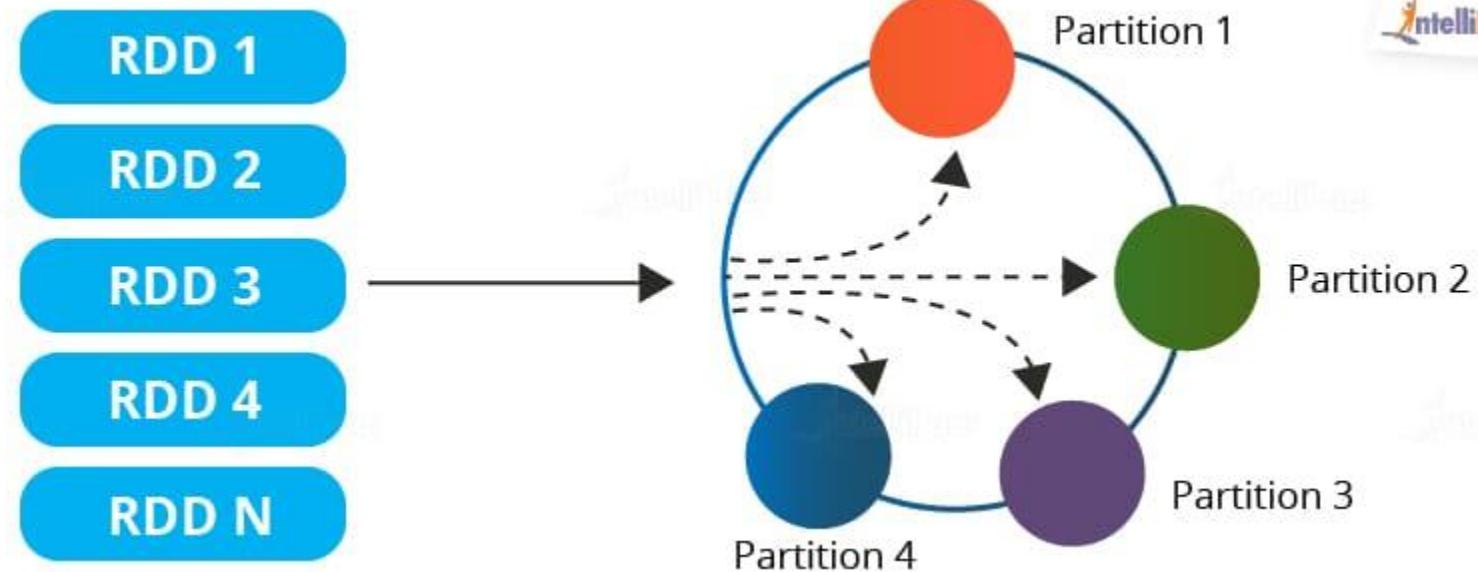
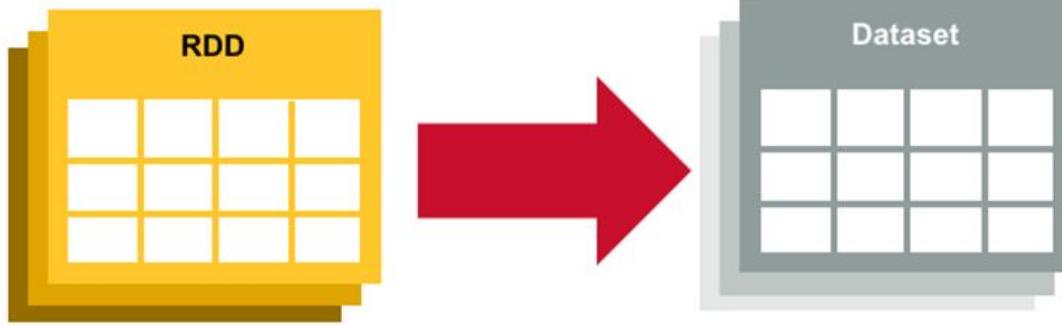


**Unknown Data Type**  
Dataset [Row]



**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 A A Wrap Text General \$ % , 0.00

B I U Merge & Center Conditional Formatting

Font Alignment Number Styles

Clipboard

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Arrest	Age	Sex	Race	ArrestDate	ArrestTim	ArrestLoc	IncidentO	IncidentLc	Charge	ChargeDe	District	Post	Neighbor	Location
2	16160529	54	M	B	11/12/2016	22:35	3500 PELH	4ECOMMC	3500 PELH	1 1415	COMMON	Northeast	432	Belair-Edi	(39.320868)
3	16160490	22	M	B	11/12/2016	21:49	300 S LOU	Unknown	300 S LOU	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-5 TO DECREE				
13	16160424	41	F	B	11/12/2016	14:20		Unknown Offense		1 0077	FAILURE TO APPEAR				
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 root@wk-c X 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	IncidentL	Charge	ChargeDe	District	Post	Neighb	Location 1		
2	16160529	54	M	B	11/12/2016	22:35	3500 PELHAM	4E COMM C	5500 PELH	1 1415	COMMON	Northeast	432	Belair-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 LAKEW	4E COMM C	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 C	5-00 MAY	1 1415	COMMON	Northeast	440	Markland	(39.30055367, -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 C	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	6CLARCE	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	Unknown Offense		Unknown	Southeast	224	Ellwood D	(39.292277410, -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

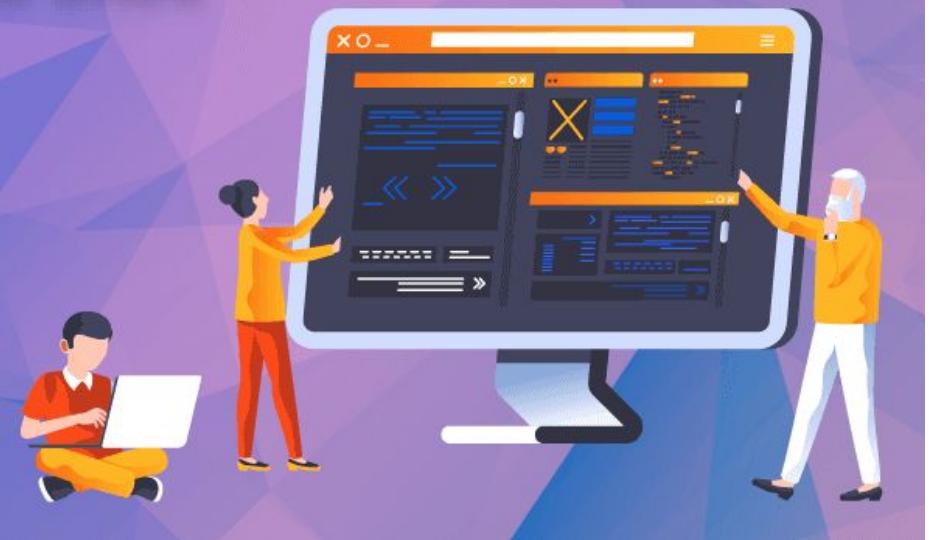
- Parquet
- JSON
- Hive Tables
- Relational Databases

## Spark SQL - Schema RDD

Unified Interface For Structured Data



# REPL Environment for Apache Spark Shell



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

## **Spark Interactive Shell**

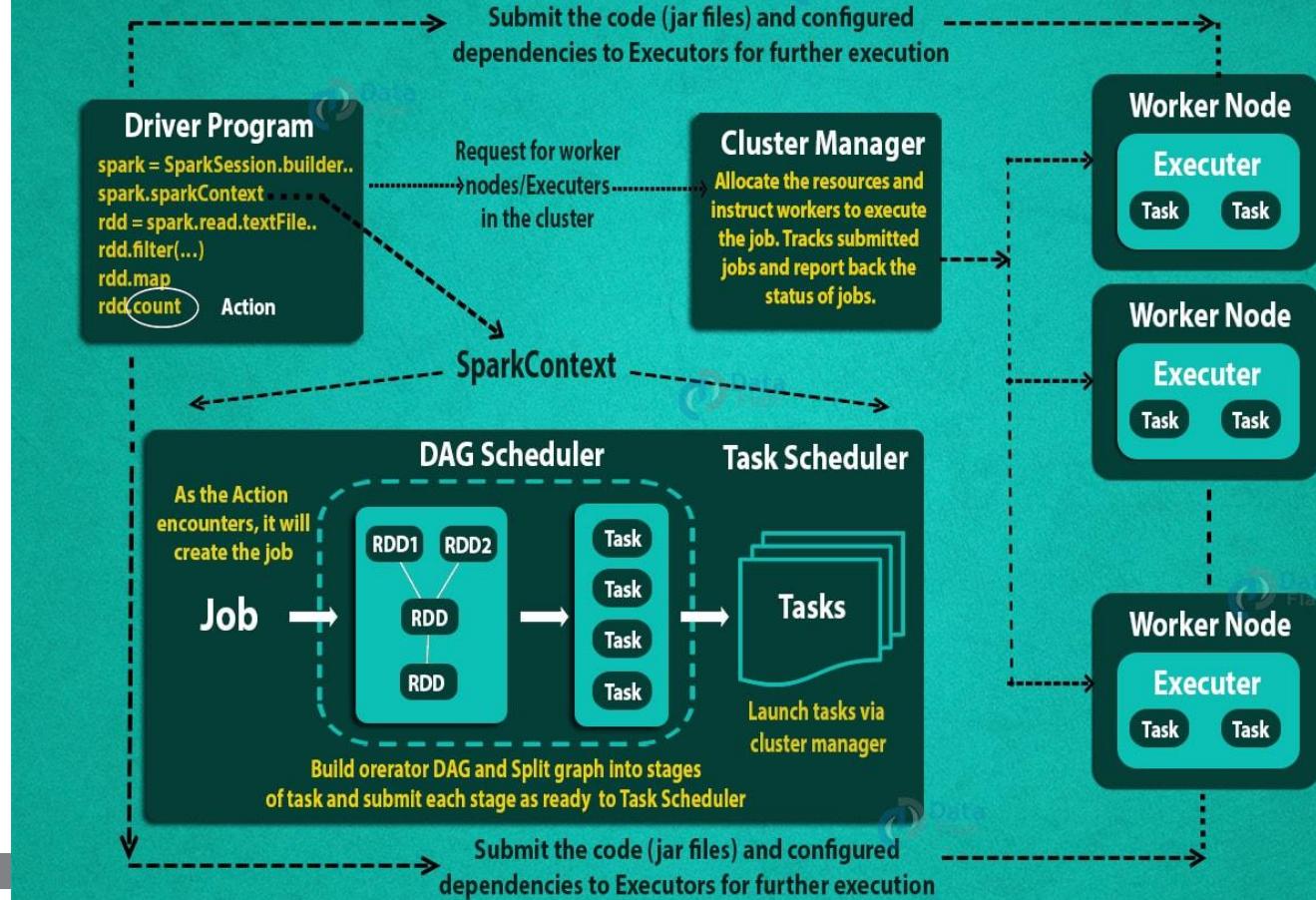
### **Scala REPL**

- Interactive Development**
- Feedback is instant**

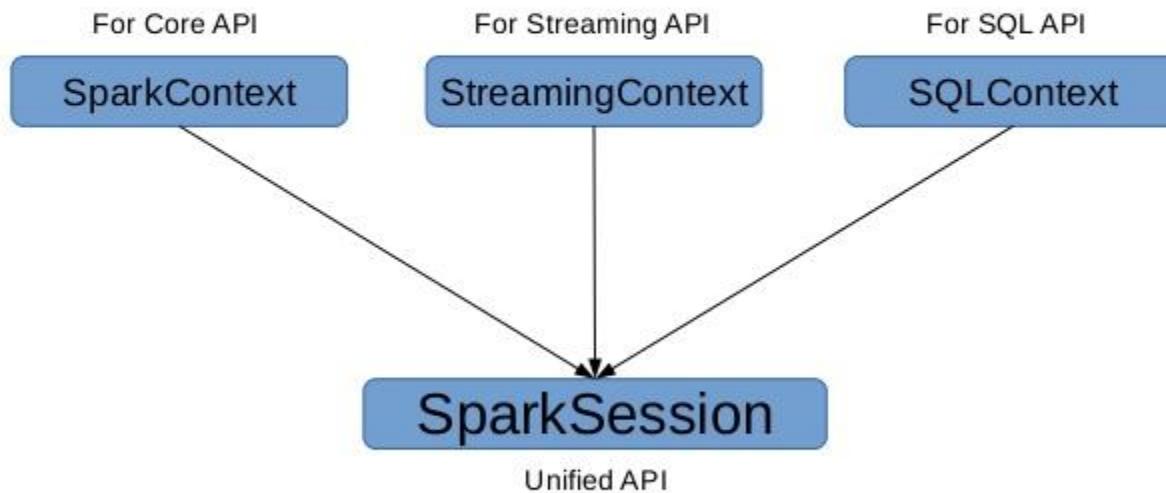
**\*\*VERY IMPORTANT - A  
Spark Session is CREATED  
when the REPL is started!**



# 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  
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  
Spark context available as 'sc' (master = local[*], app id = local-1597556449821).  
Spark session available as 'spark'.  
Welcome to  
+---+  
| / \ |  
| \ V / |  
| - \ / - |  
| / \ / \ |  
| . / \ , / / \ / \ |  
+---+  
version 2.1.2  
Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_242)  
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 a dark background. At the top, there are two tabs: "readme.md" and "root@wk-caas-d2813d52c4:~#". The main area of the terminal shows a green command prompt: "root@wk-caas-d2813d52c4:~#".

```
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-dev3600/DEV360/LESSON_02/BPD_Arres
ts.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:

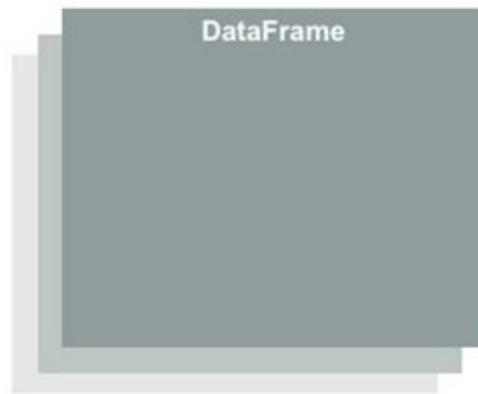
- A. Parquet Tables
- B. JSON
- C. Streaming Data
- D. JDBC

# 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")
```

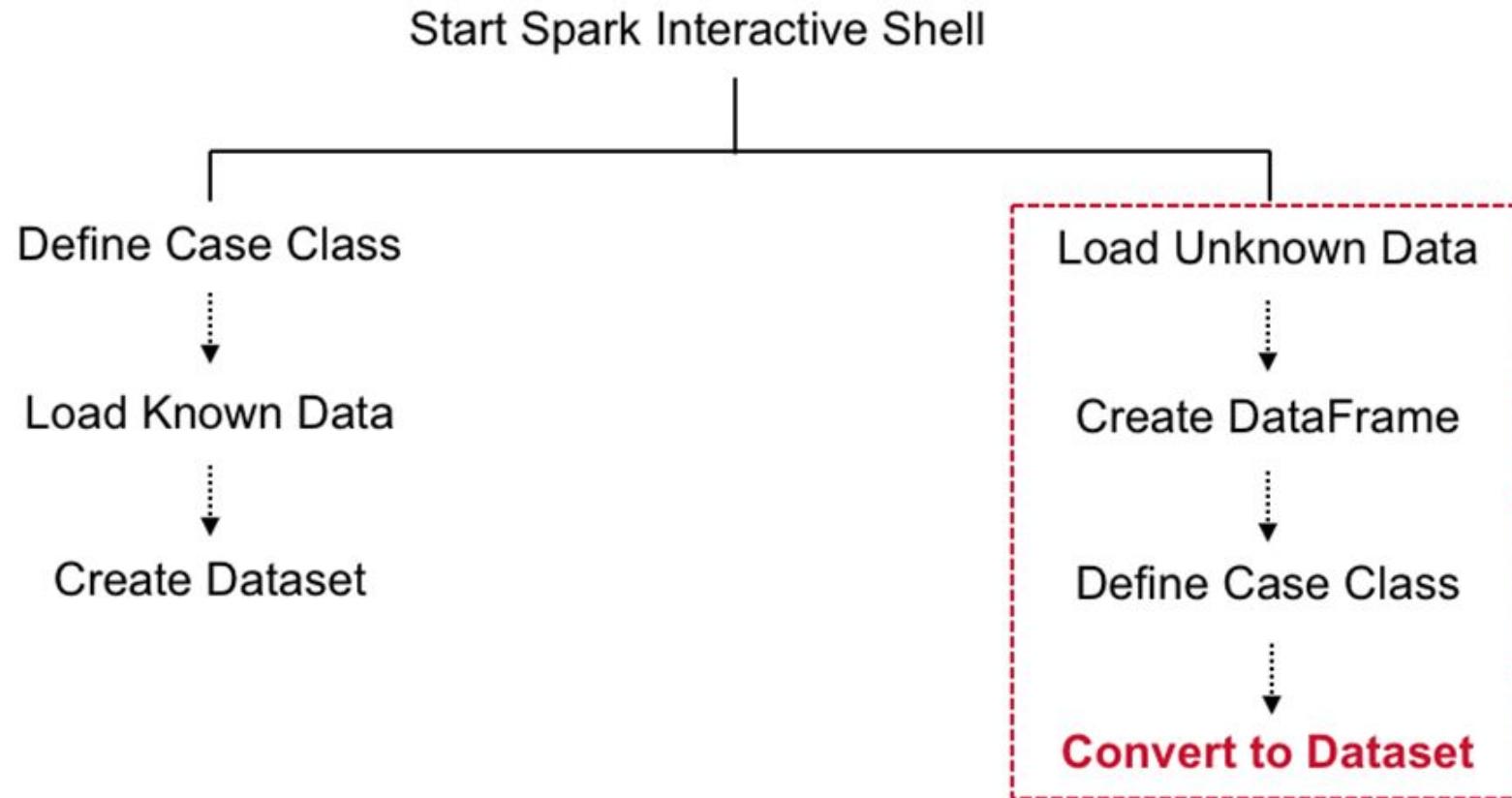
## Step 4: Register the DataFrame as a Table (optional)

```
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

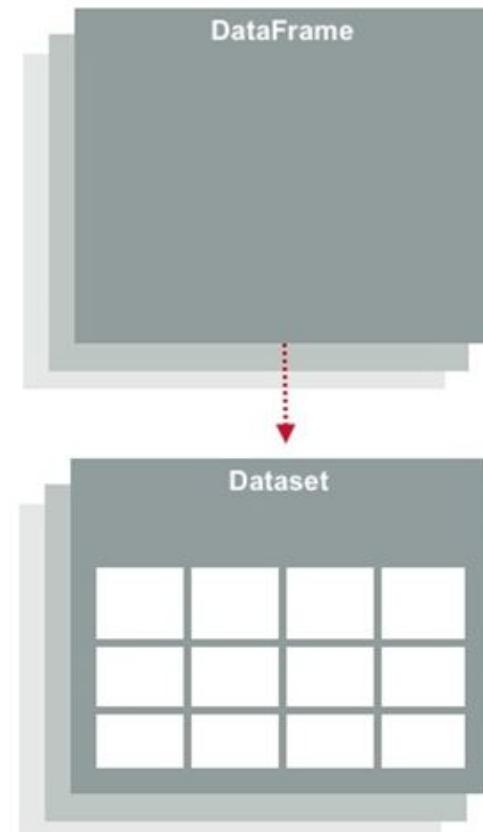
Convert to Dataset

# 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

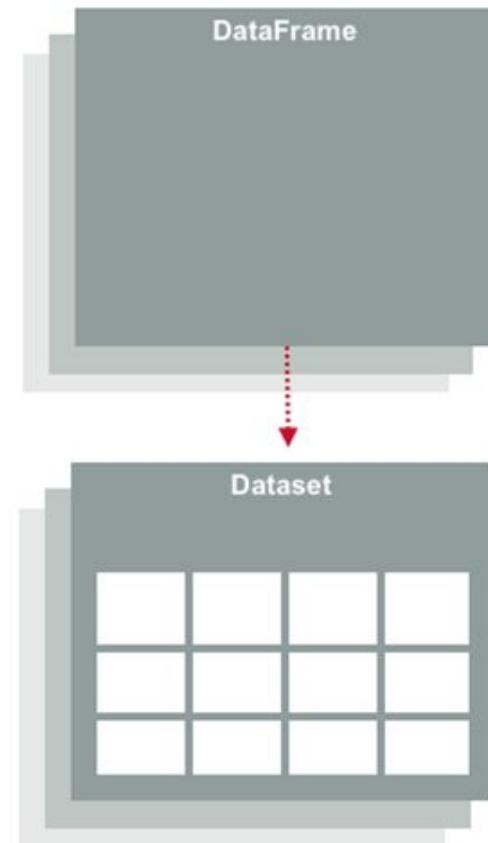


# Infer Schema by Reflection: Case Class

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

# Review



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.)

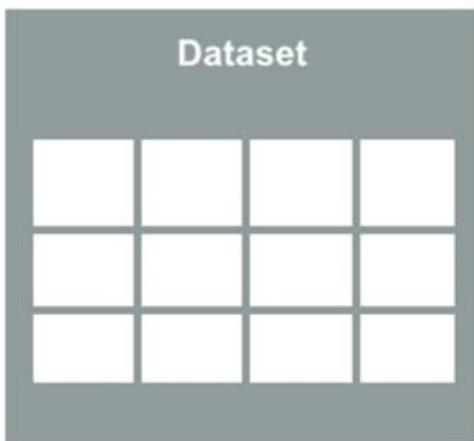
Transformation	Definition
<code>cache()</code>	Cache this Dataset
<code>persist()</code>	Persist this DataFrame
<code>createTempView (viewName)</code>	Registers this Dataset as a temporary view using the given name
<code>describe()</code>	Calculates count, mean, sttdev, min, and max for numeric columns

# Language Integrated Queries

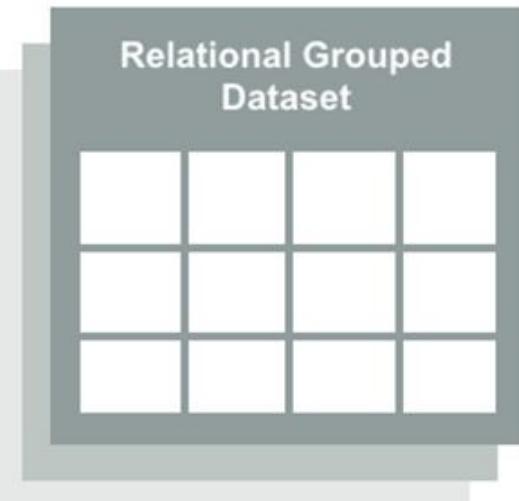
Transformation	Definition
<code>agg(expr, exprs)</code>	Aggregates on the entire Dataset without groups
<code>filter(condition Expr)</code>	Filters based on given SQL expression
<code>groupBy(col1, cols)</code>	Groups Dataset using the specified columns so we can run aggregation on them
<code>select(cols)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

# Relational Grouped Datasets

## TRANSFORMATIONS



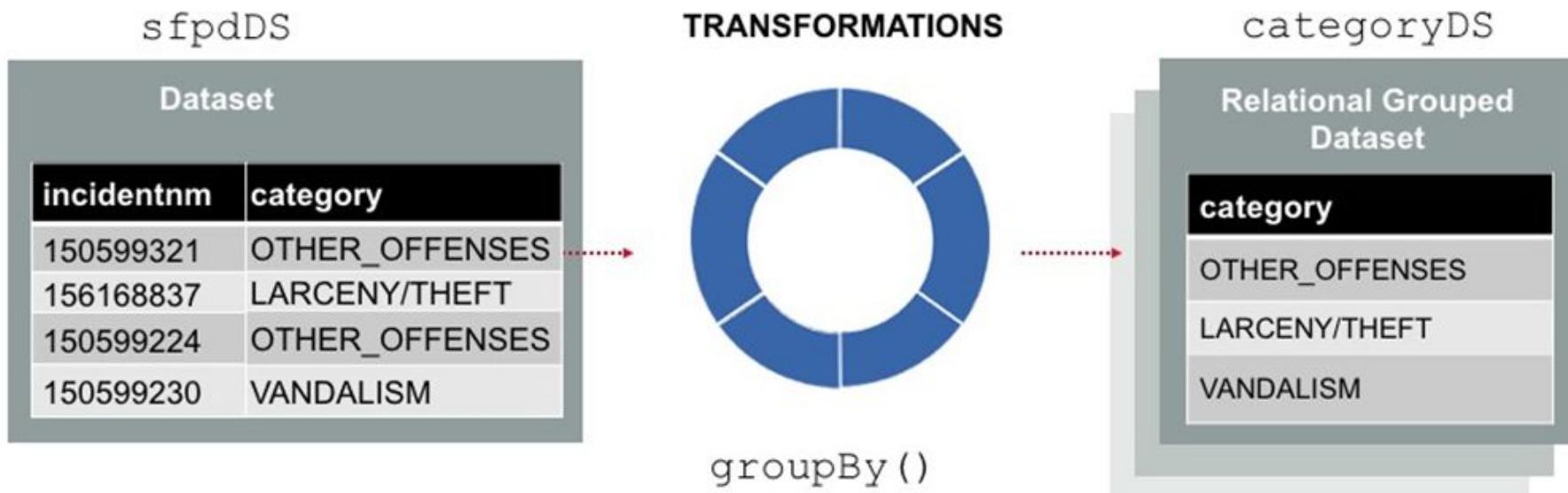
groupBy()



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_	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**

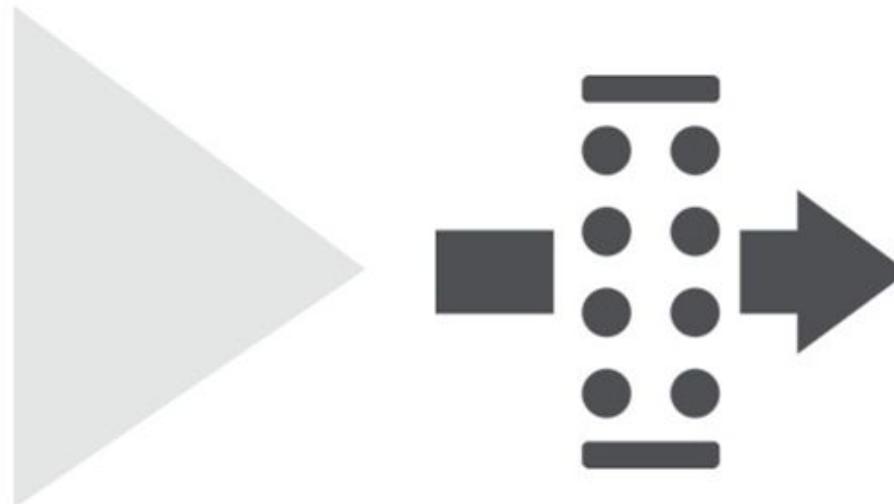
SOUTHERN

MISSION

NORTHERN

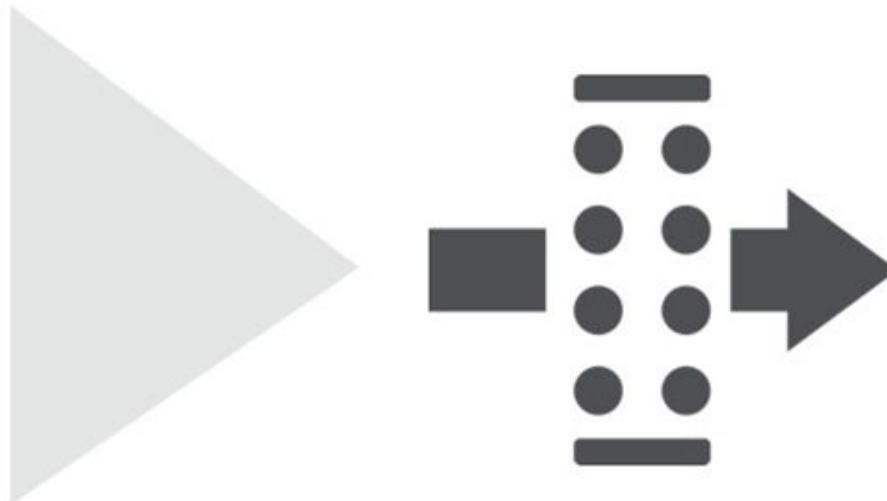
SOUTHERN

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**

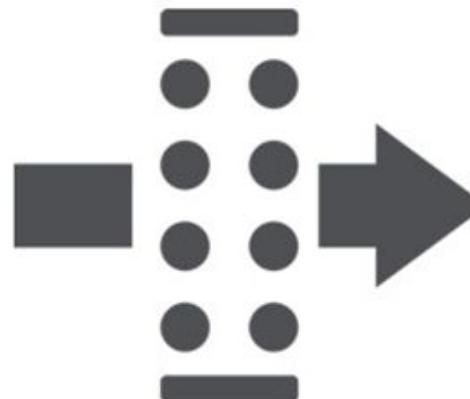
SOUTHERN

MISSION

NORTHERN

SOUTHERN

CENTRAL



**=>**

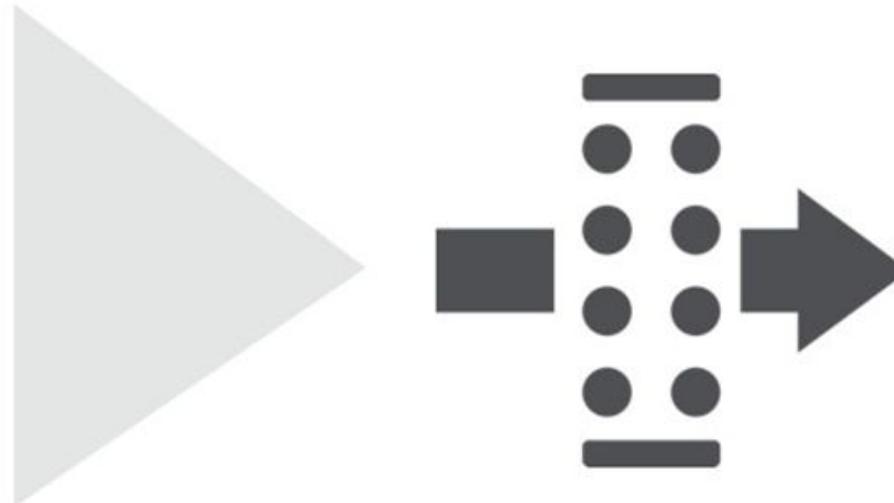
Anonymous Syntax



# Transformation: filter()

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

<b>districtDS</b>
SOUTHERN
MISSION
NORTHERN
<b>SOUTHERN</b>
CENTRAL



# Transformation: filter()

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

**districtDS**

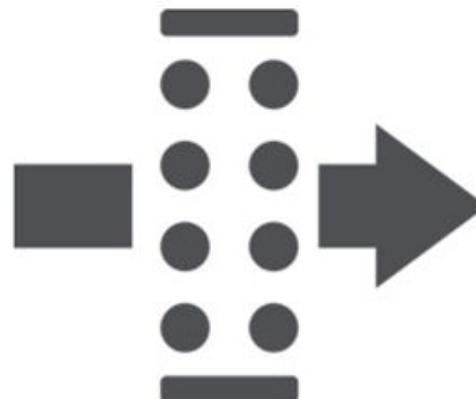
SOUTHERN

MISSION

NORTHERN

SOUTHERN

CENTRAL



**southernDS**

SOUTHERN

SOUTHERN

# Review



Load data



`spark.read.text`

Transform data

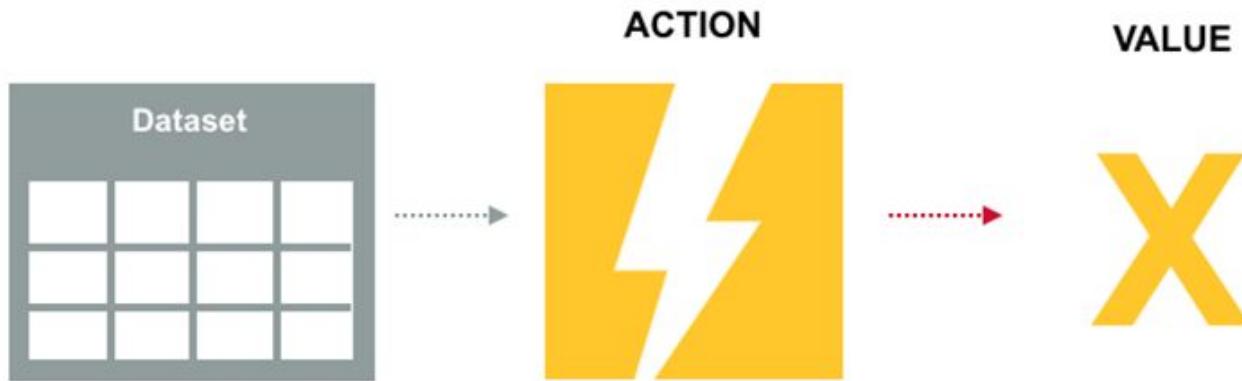


Want to see only  
SOUTHERN incidents  
`filter()`



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

# Review



# Commonly Used Actions

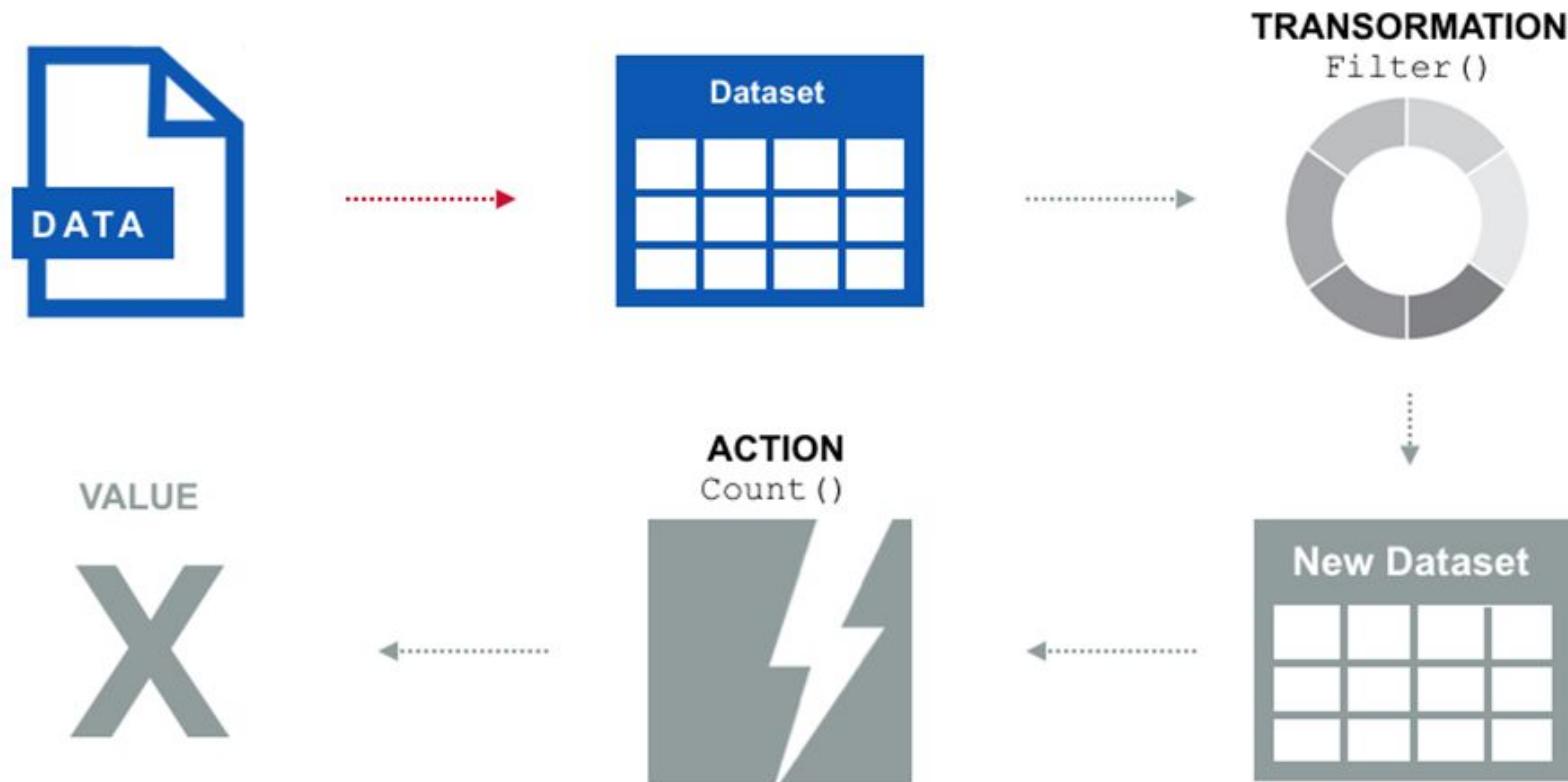
Action	Definition
<code>count()</code>	Returns the number of elements in the Dataset
<code>reduce(func)</code>	Aggregate elements of Dataset using function <code>func</code>
<code>collect()</code>	Returns all elements of Dataset as an array to driver program
<code>take(n)</code>	Returns first $n$ elements 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

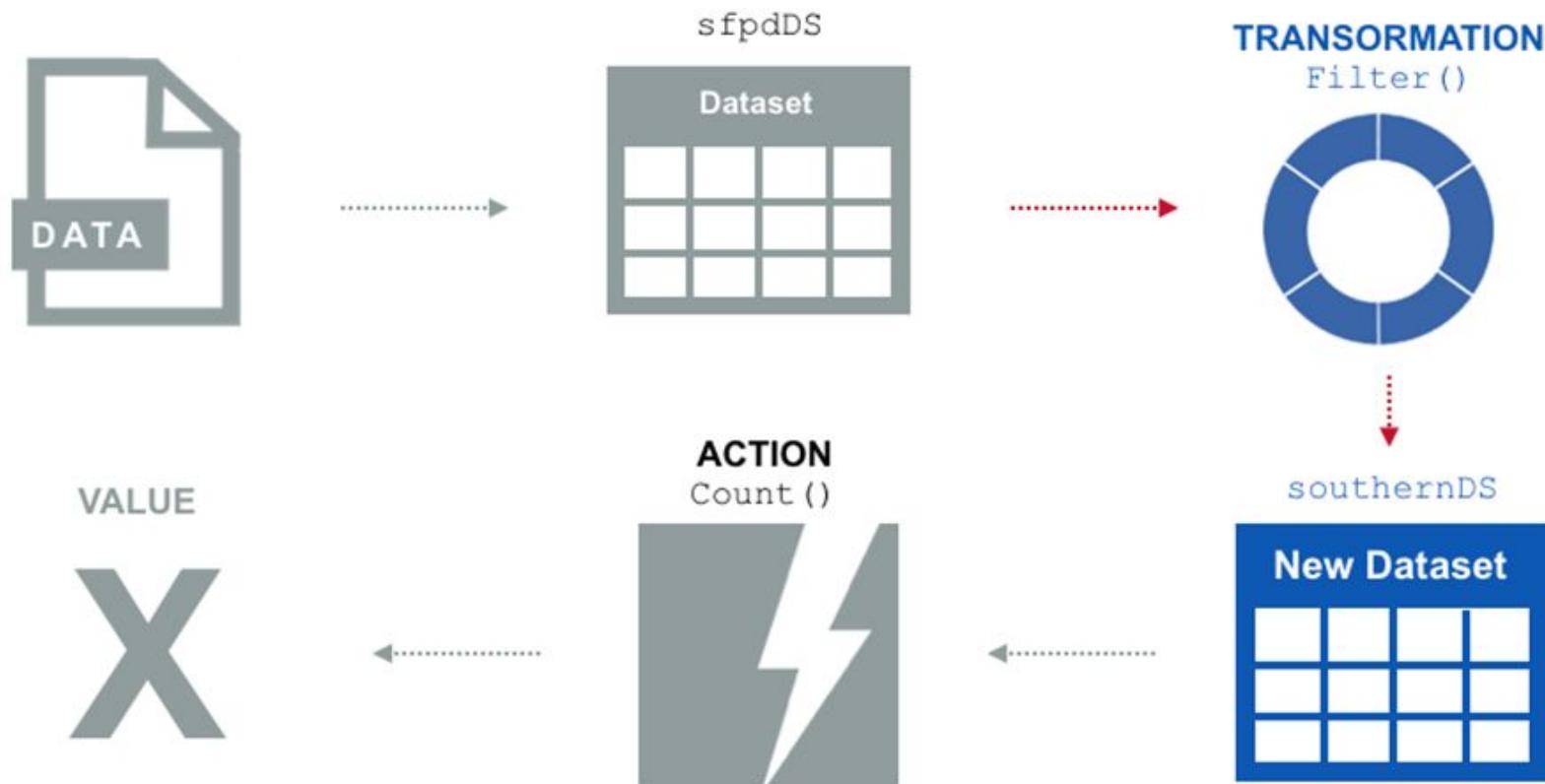
---

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

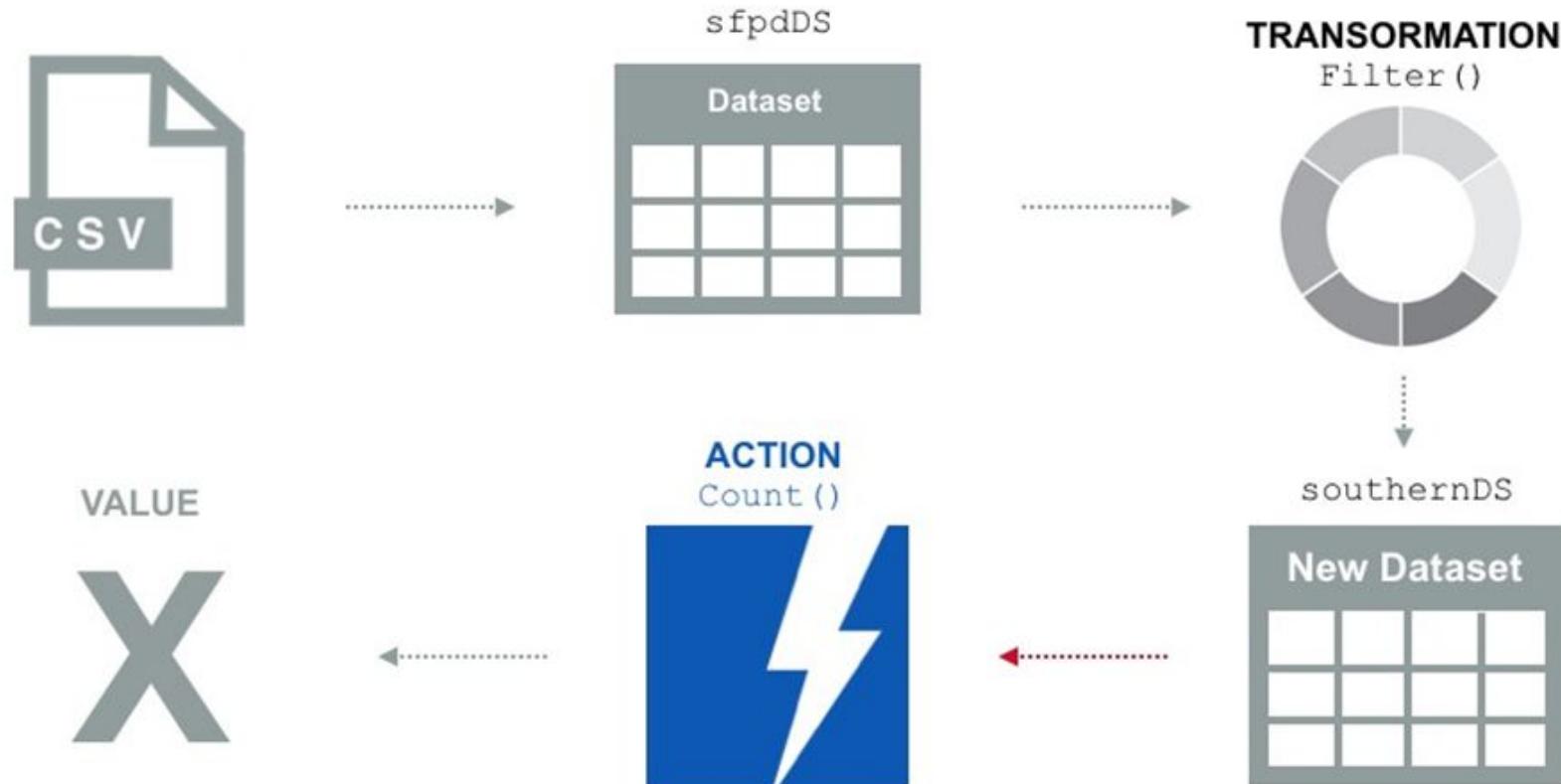
# Step 1: Define Dataset



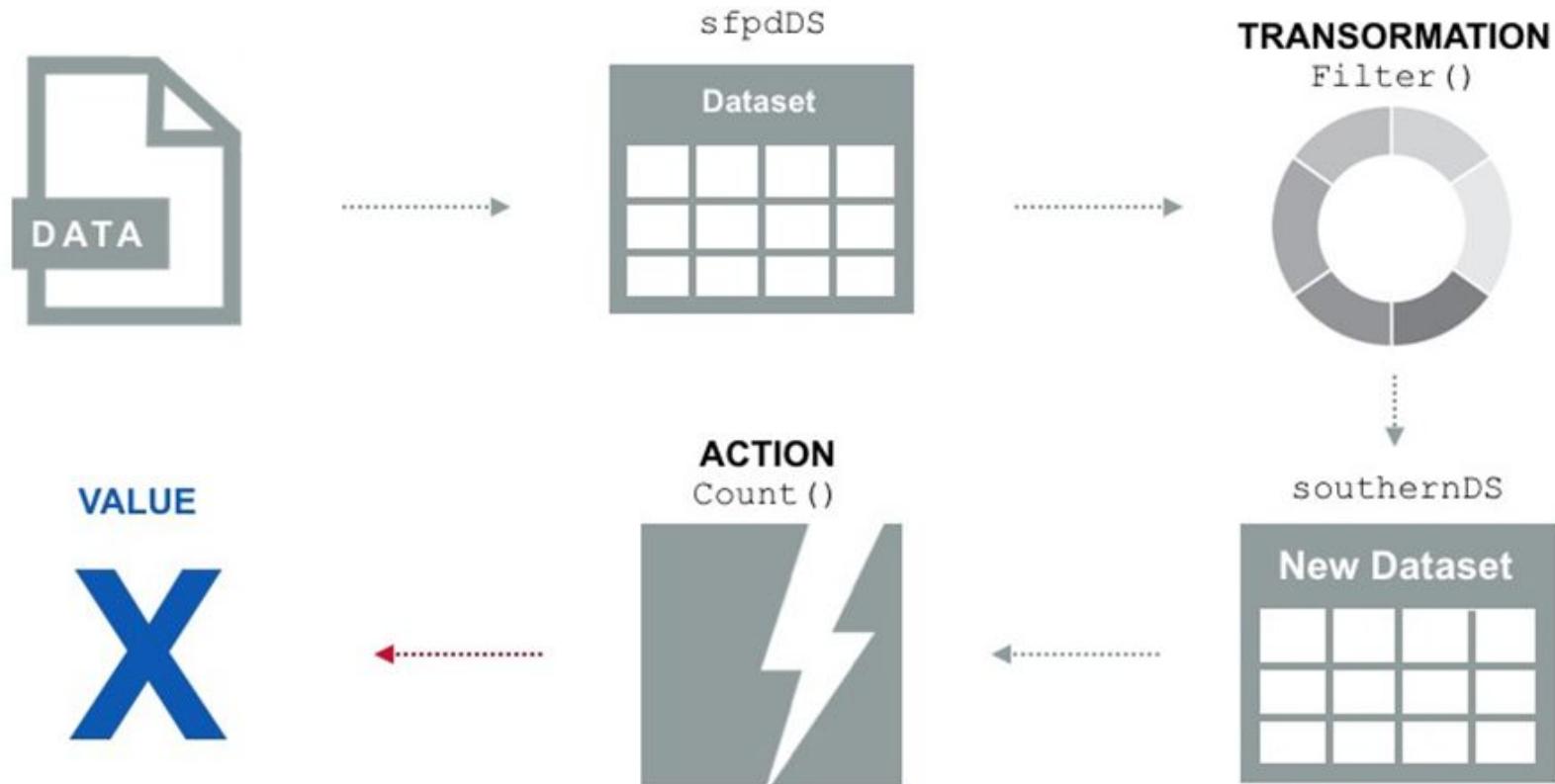
## Step 2: Define Transformation



# 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

A. Returns a Dataset

B. Returns a value

C. Computed lazily

D. Examples include count, take

E. Examples include filter, map

OR



Transformation

# 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
save (source, mode, options)	Saves contents of Dataset based on given data sources, savemode, and set of options
jdbc (url, name, overwrite)	Saves contents of Dataset to JDBC at URL under table name table
parquet (path)	Saves contents of Dataset as parquet file
saveAsTable (tablename, source, mode, options)	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?

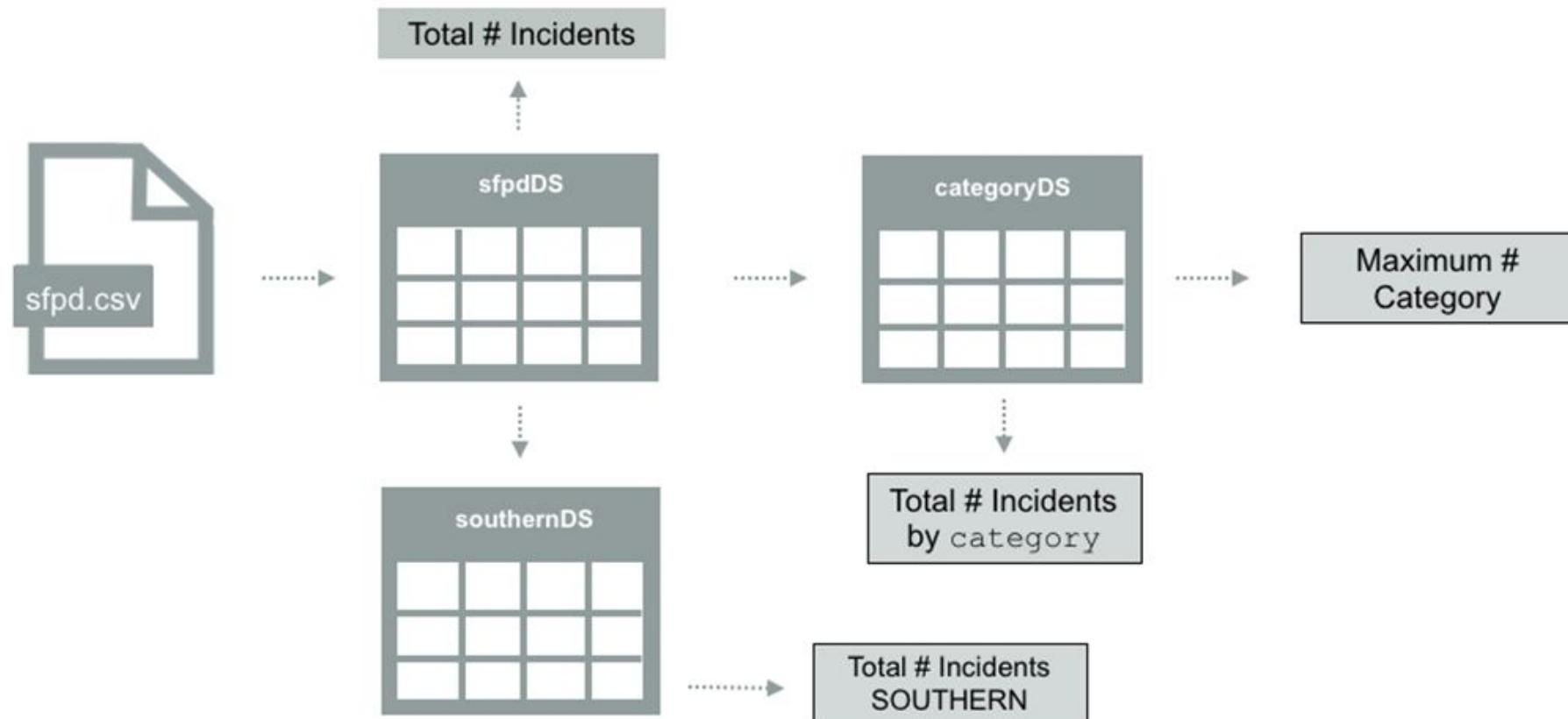
- A. spark.sql("SELECT resolution, count(incidentnum) AS inccount FROM sfpd GROUP BY resolution ORDER BY inccount DESC LIMIT 10")
- B. sfpdDS.select("resolution").count.sort(\$"count".desc).show(10)
- C. sfpdDS.groupBy("resolution").count.sort(\$"count".desc).show(10)

# Lab 3.1: Explore and Save SFPD Data

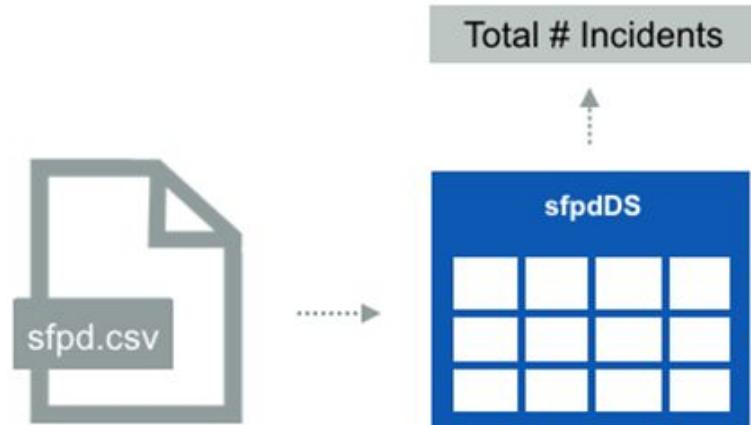


- Estimated time to complete: **20 minutes**
- In this lab, you will use SQL and Dataset operations to explore the SFPD data that you loaded in Lesson 2.

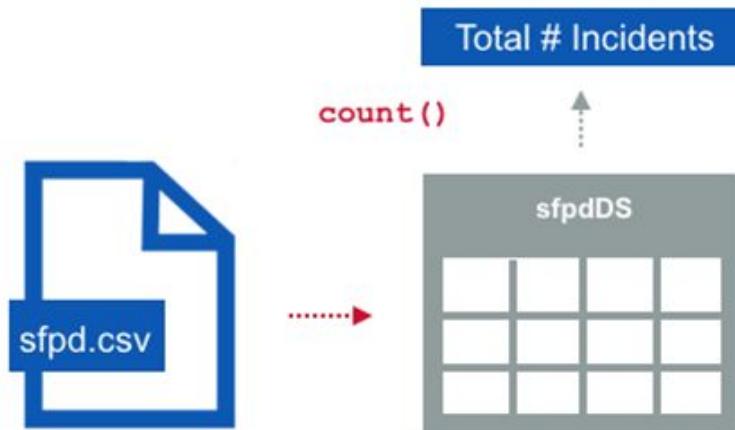
# Lineage Graph



# Lineage Graph: count()



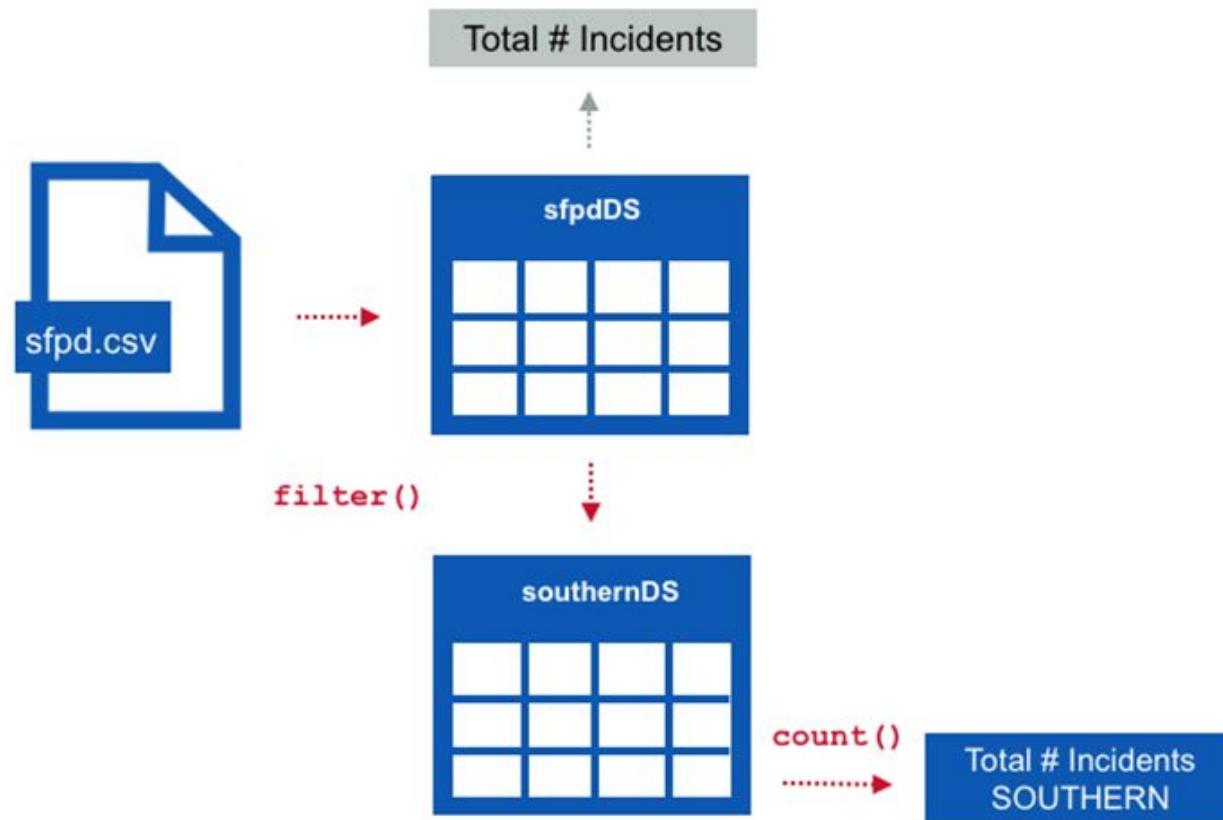
# Lineage Graph: count()



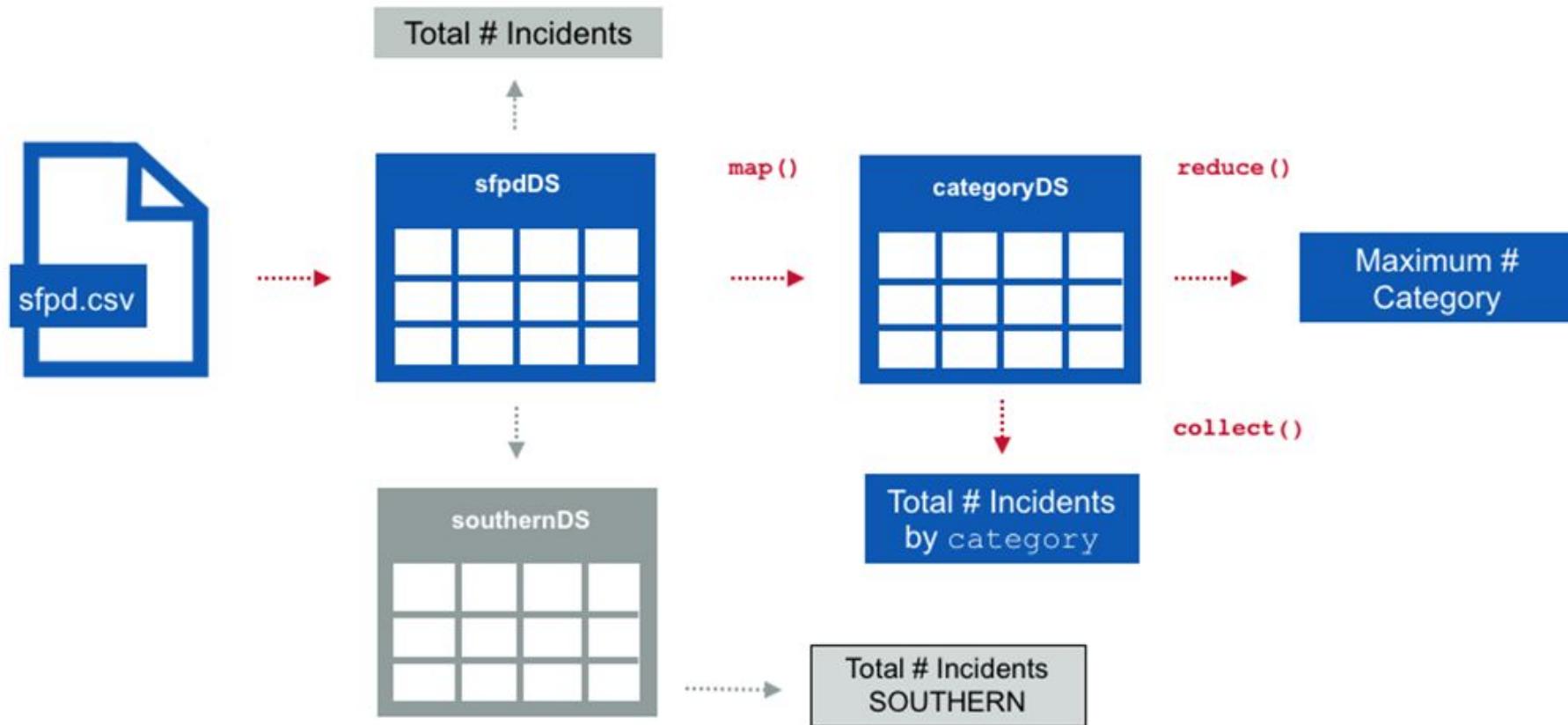
# Lineage Graph: count()



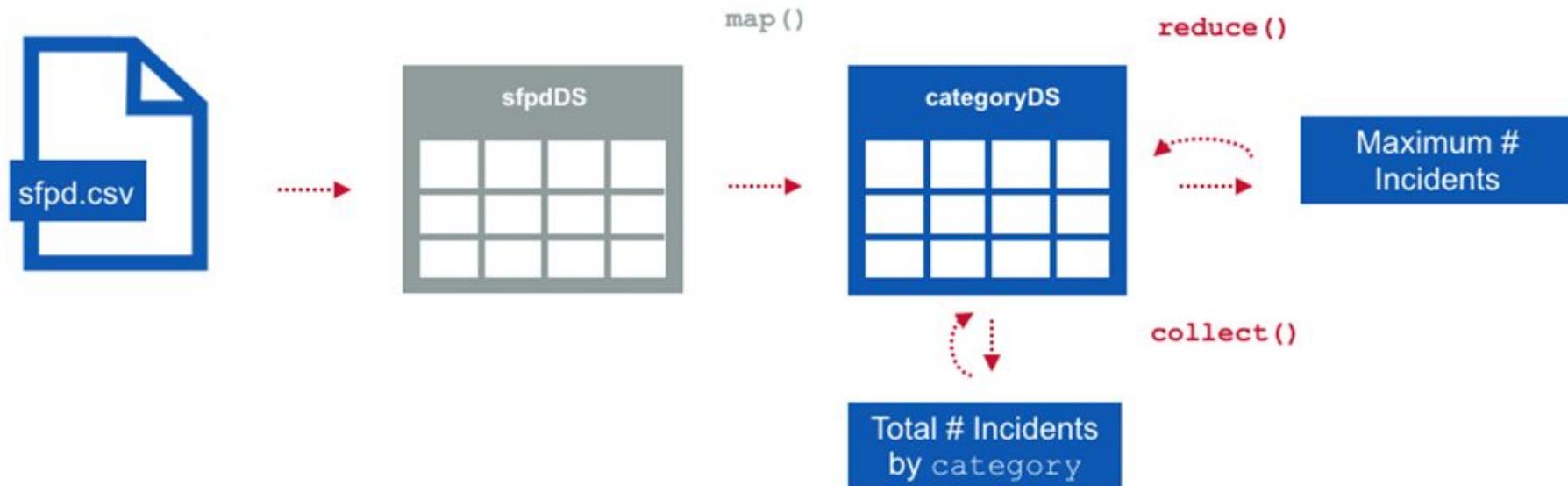
# Lineage Graph: count()



# 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

A large, bold, white "SQL" text centered on a dark grey rectangular background.

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

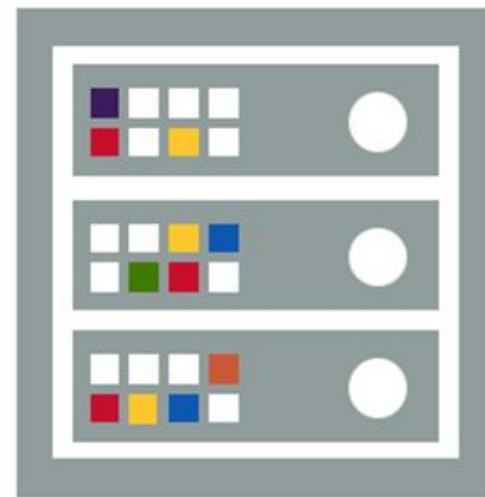
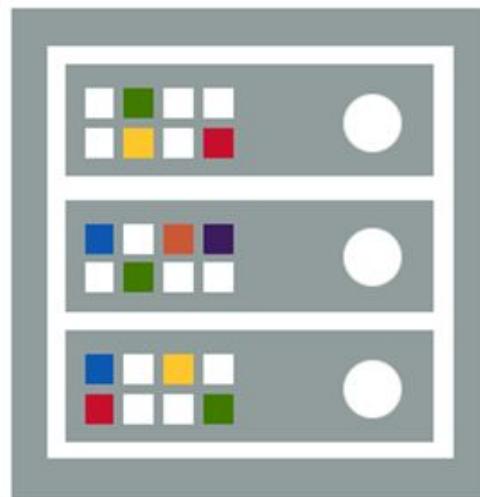
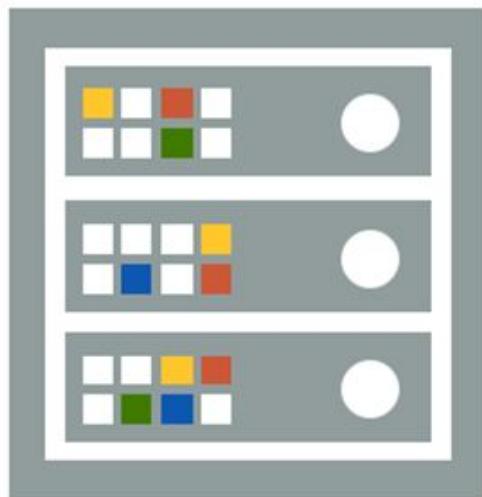
3.2 Cache Datasets

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

---

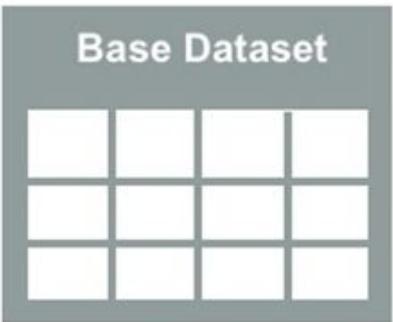
- 4.1 Define the Spark Program Lifecycle
- 4.2 Define the Function of SparkSession
- 4.3 Describe Ways to Launch Spark Applications
- 4.4 Launch a Spark Application

# Review



1

Create input  
Dataset in your  
driver program



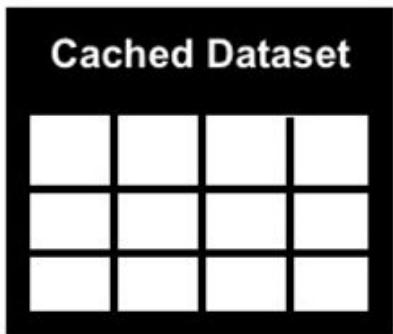
2

Use lazy  
transformations  
to define new  
Datasets



3

Cache any  
Datasets that  
are reused

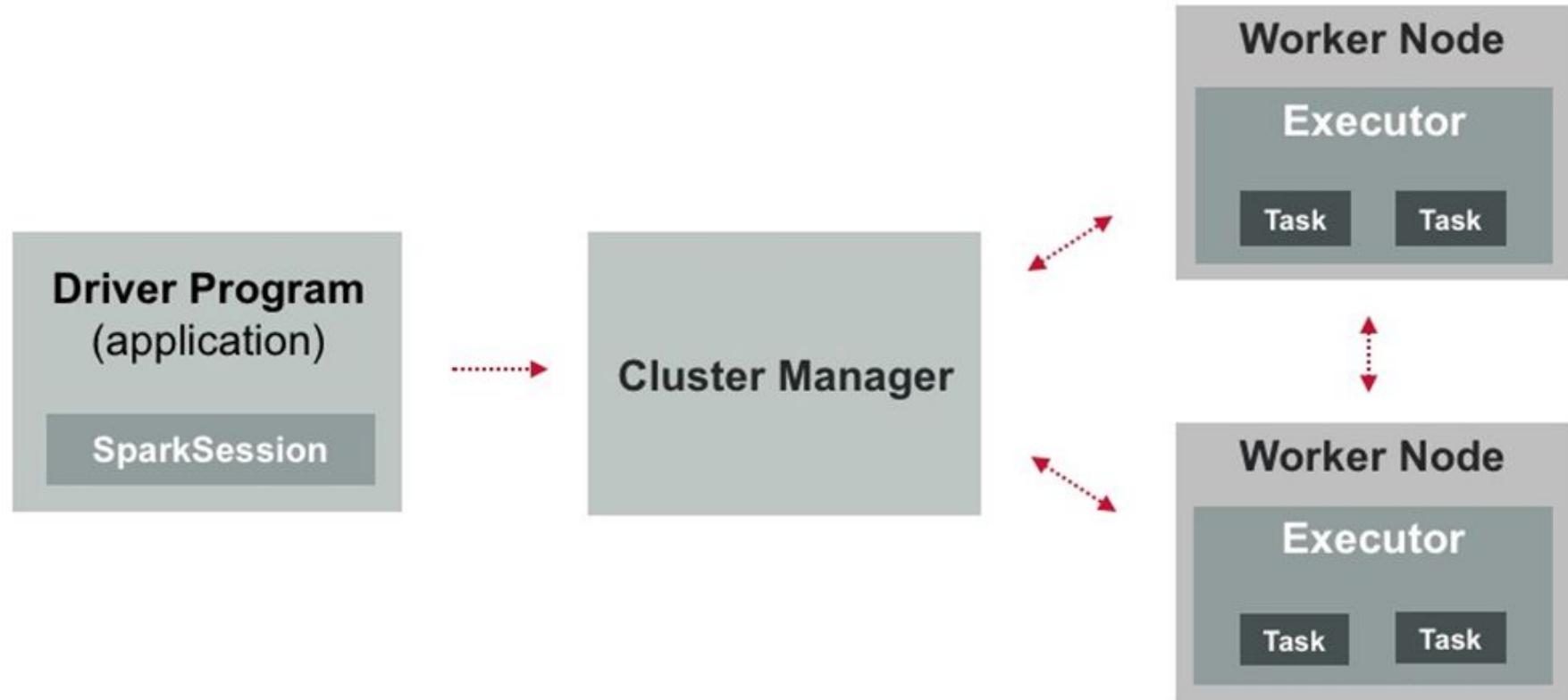


4

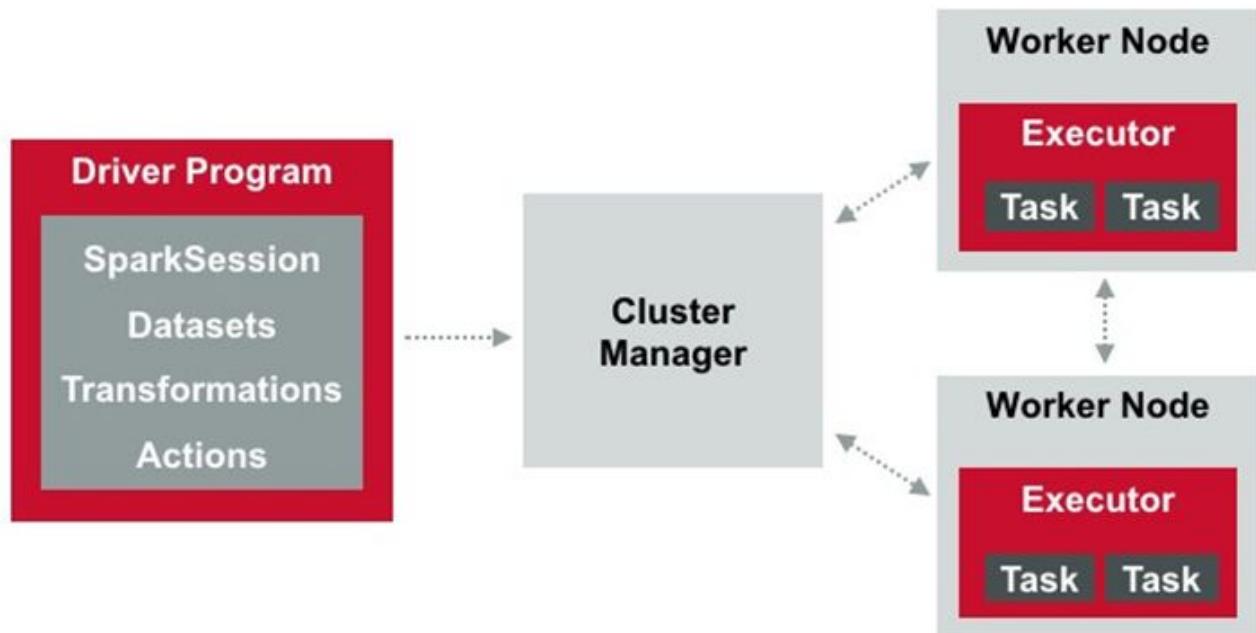
Kick off  
computations  
using actions



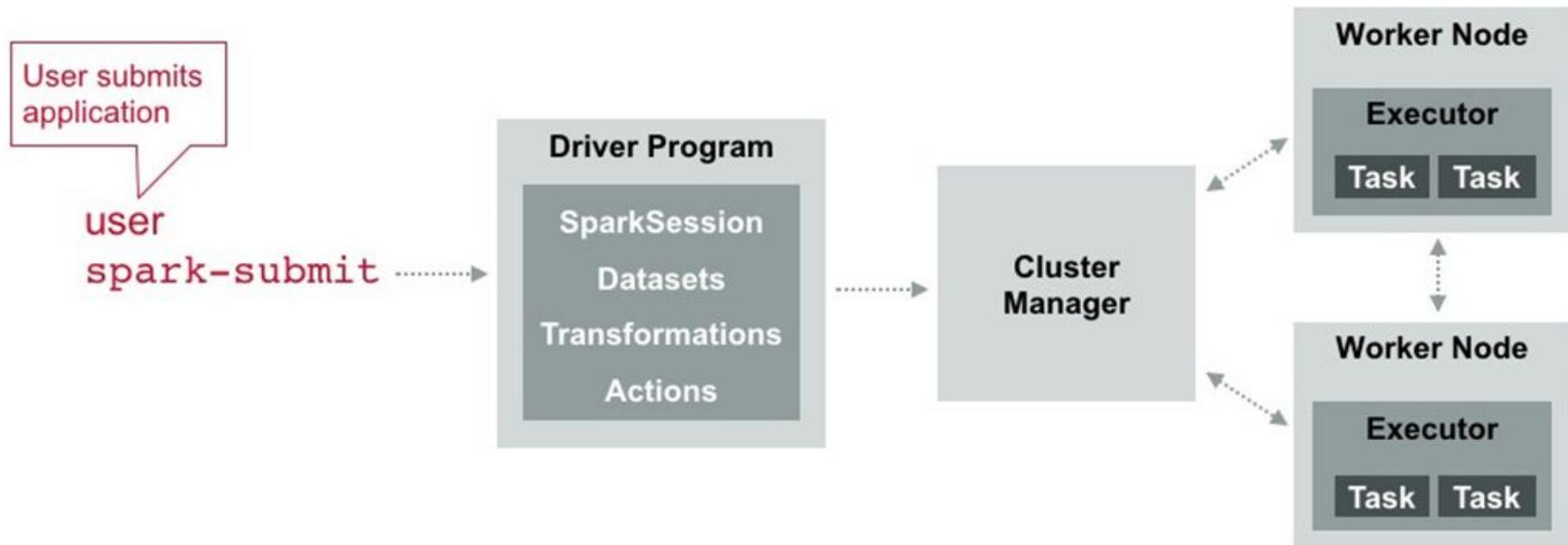
# Review



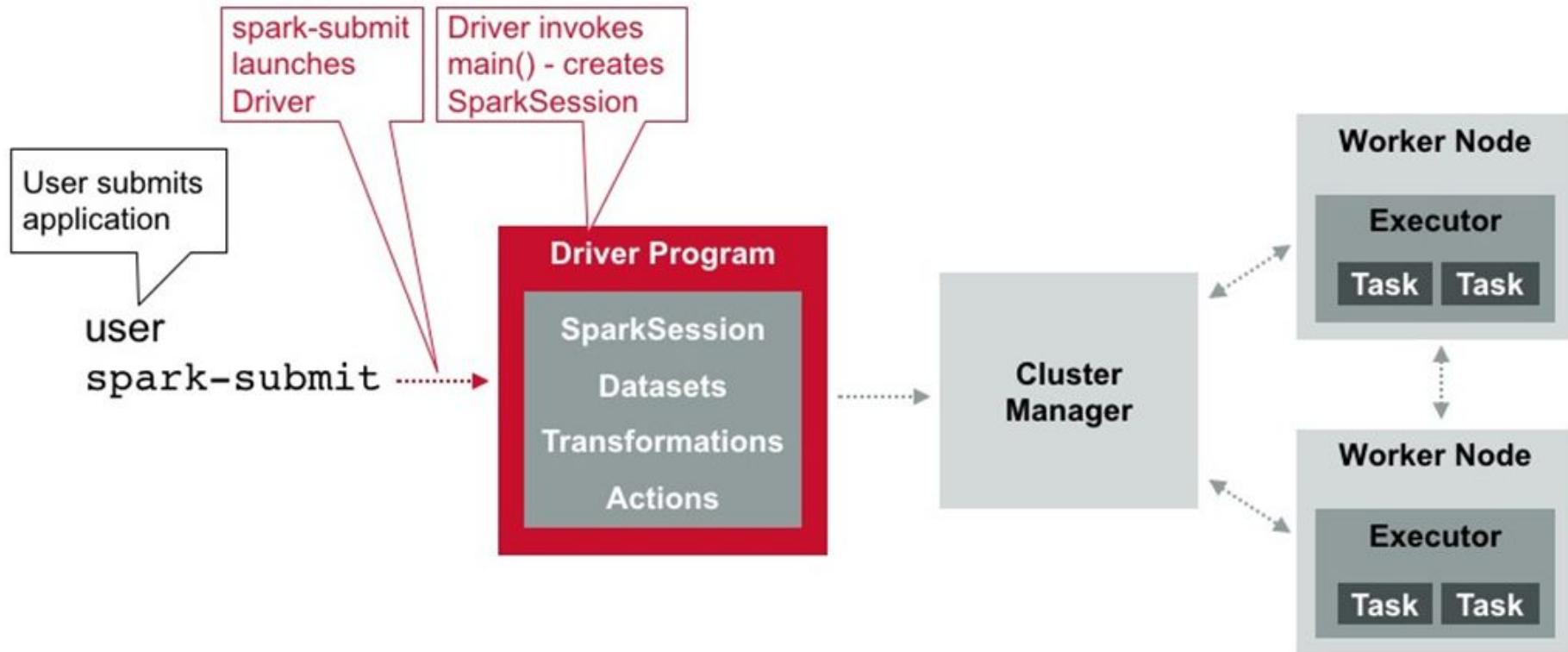
# Components of Distributed Spark Application



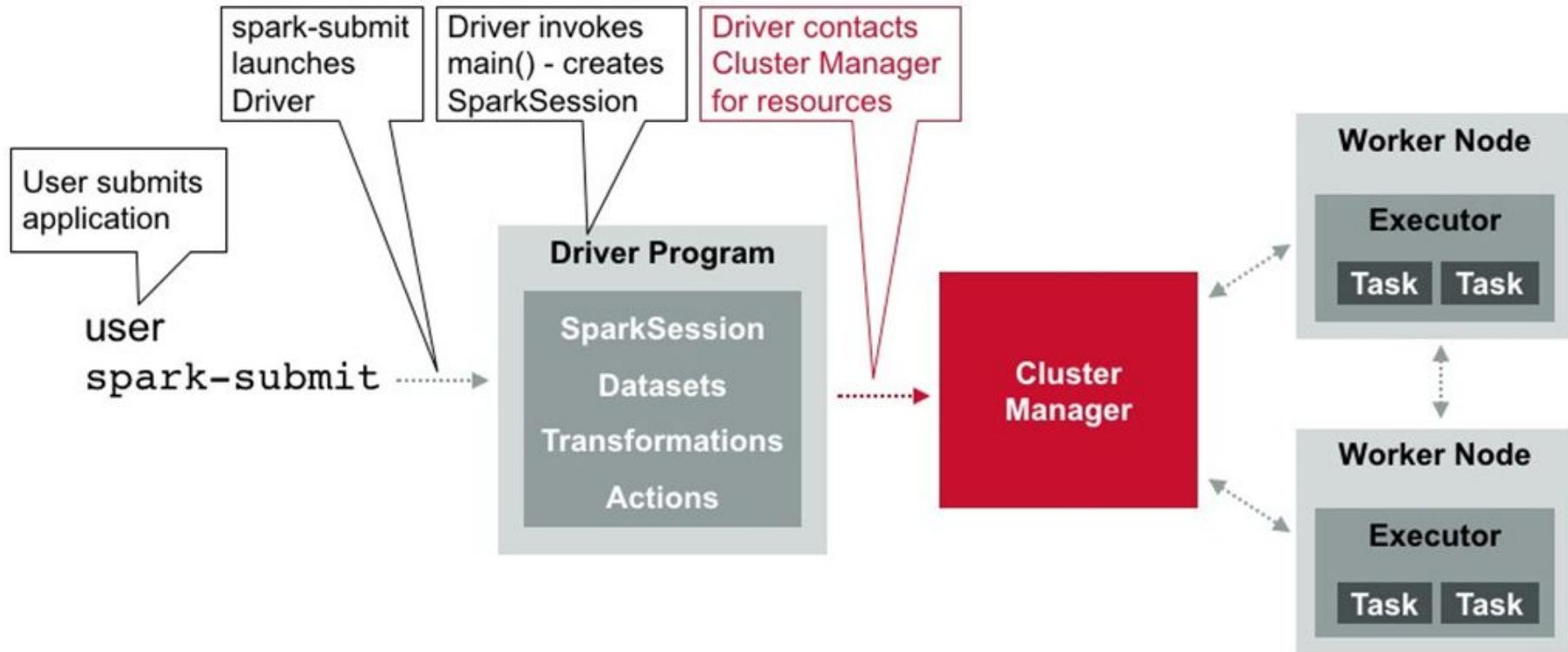
# Components of Distributed Spark Application



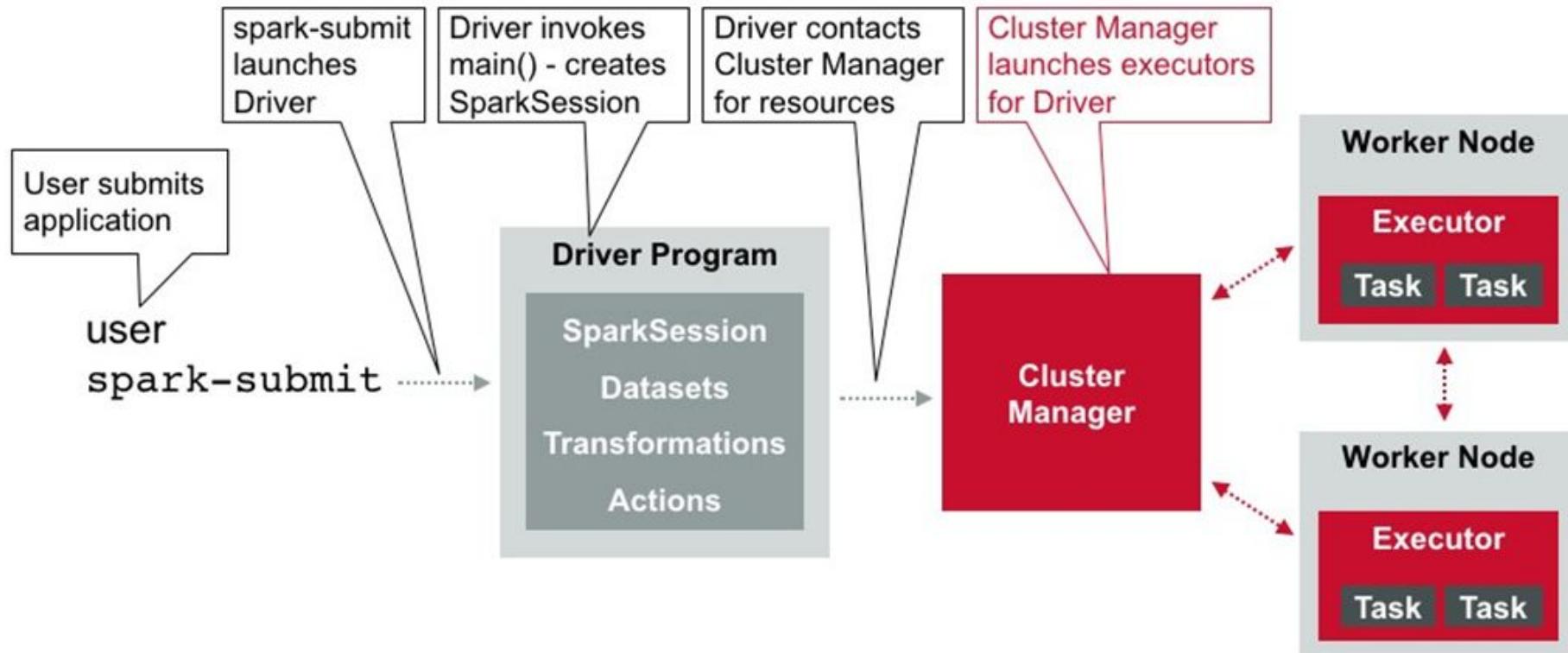
# Components of Distributed Spark Application



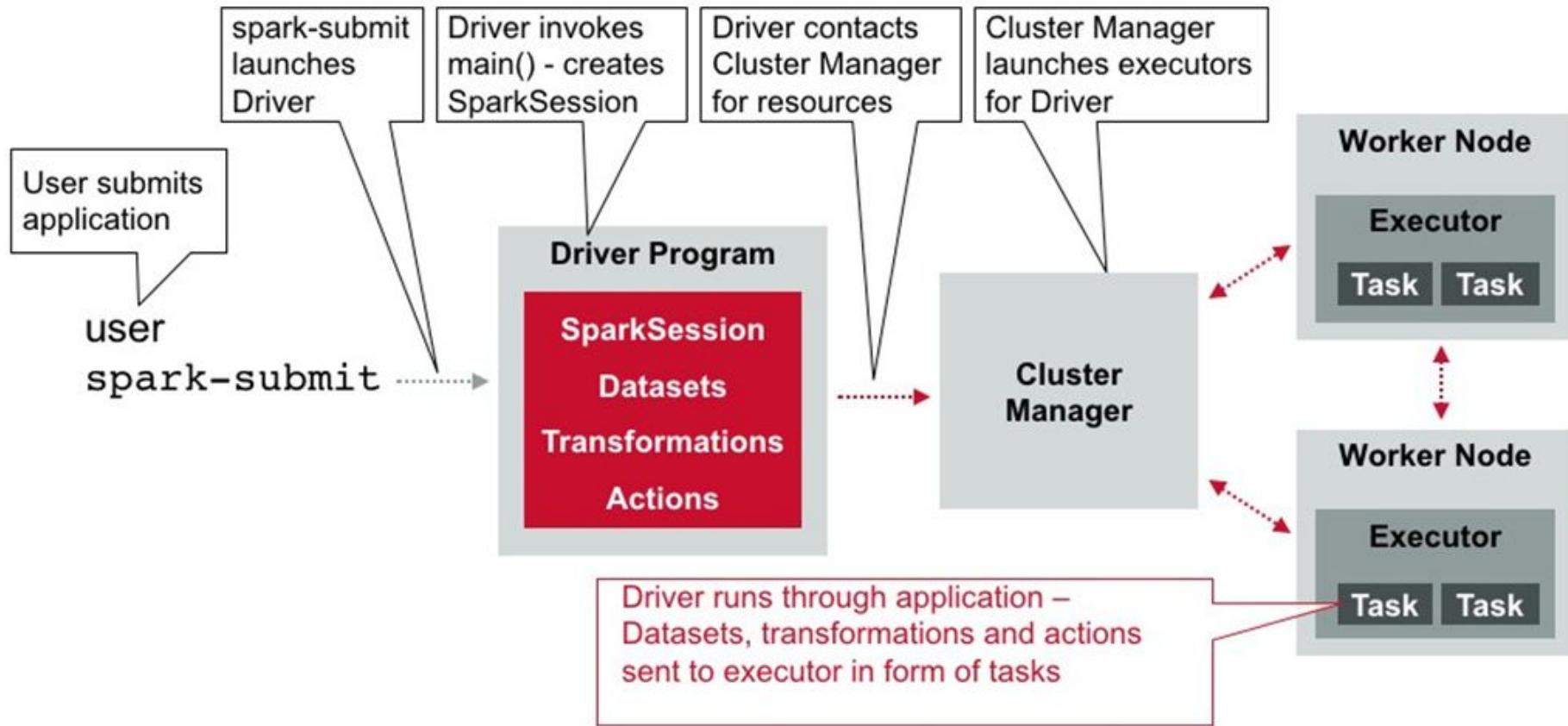
# Components of Distributed Spark Application



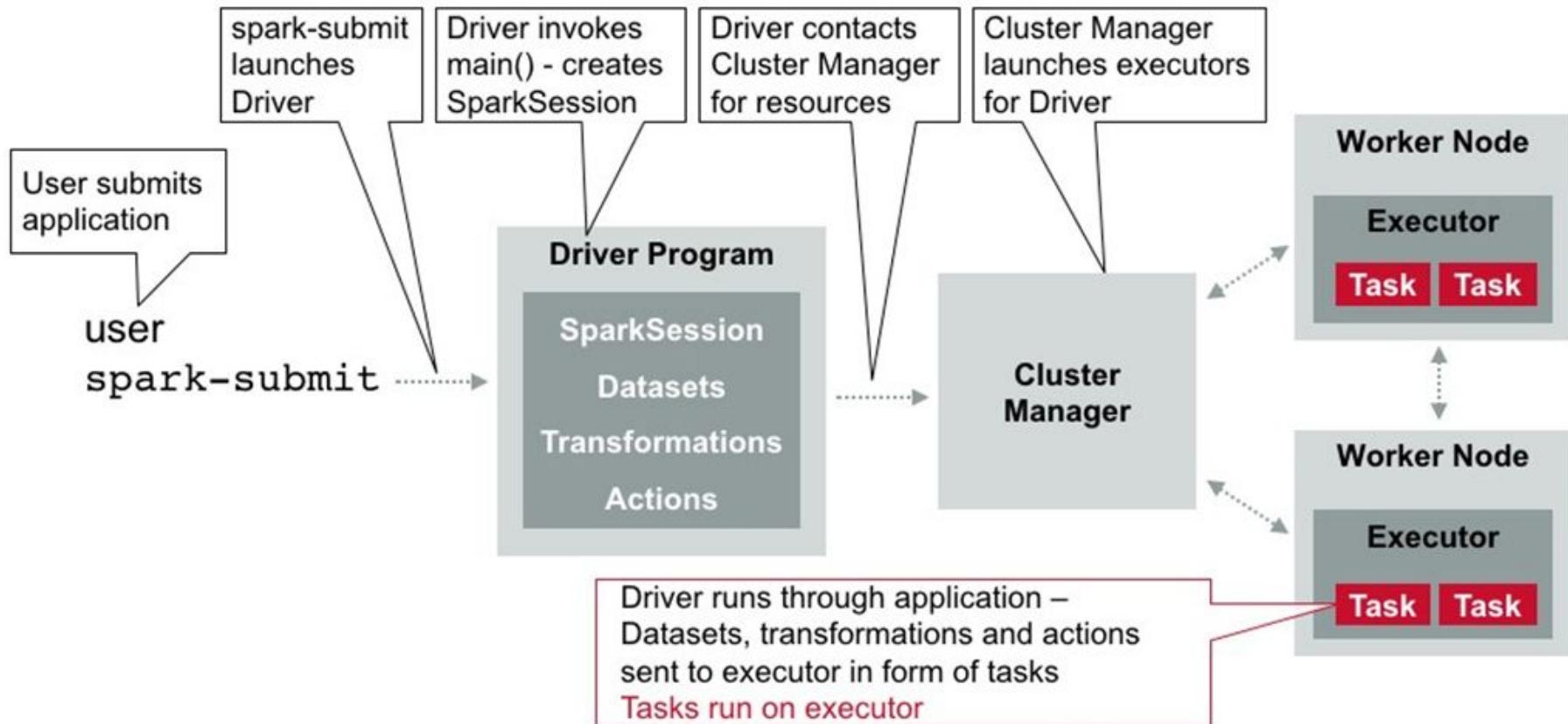
# Components of Distributed Spark Application



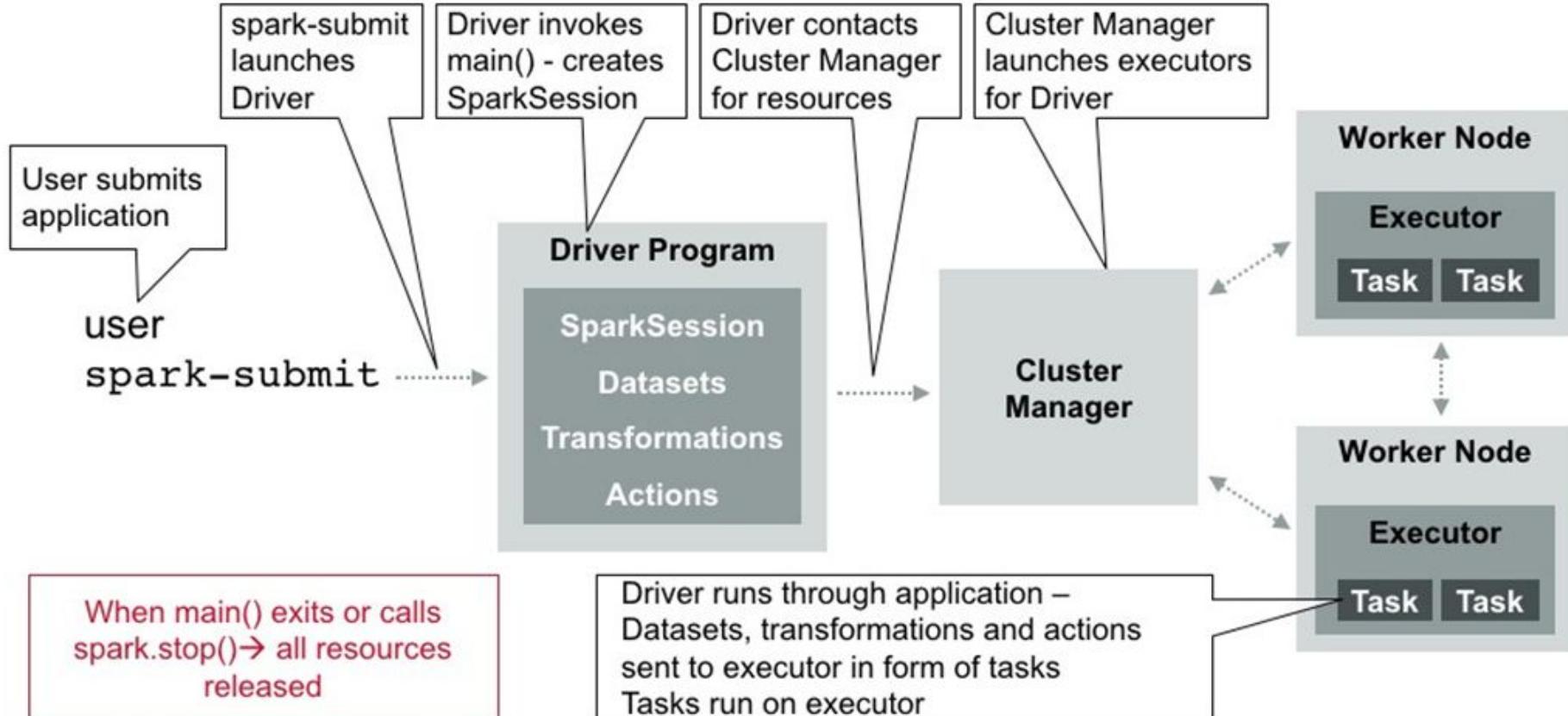
# Components of Distributed Spark Application



# Components of Distributed Spark Application



# Components of Distributed Spark Application



# Knowledge Check



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

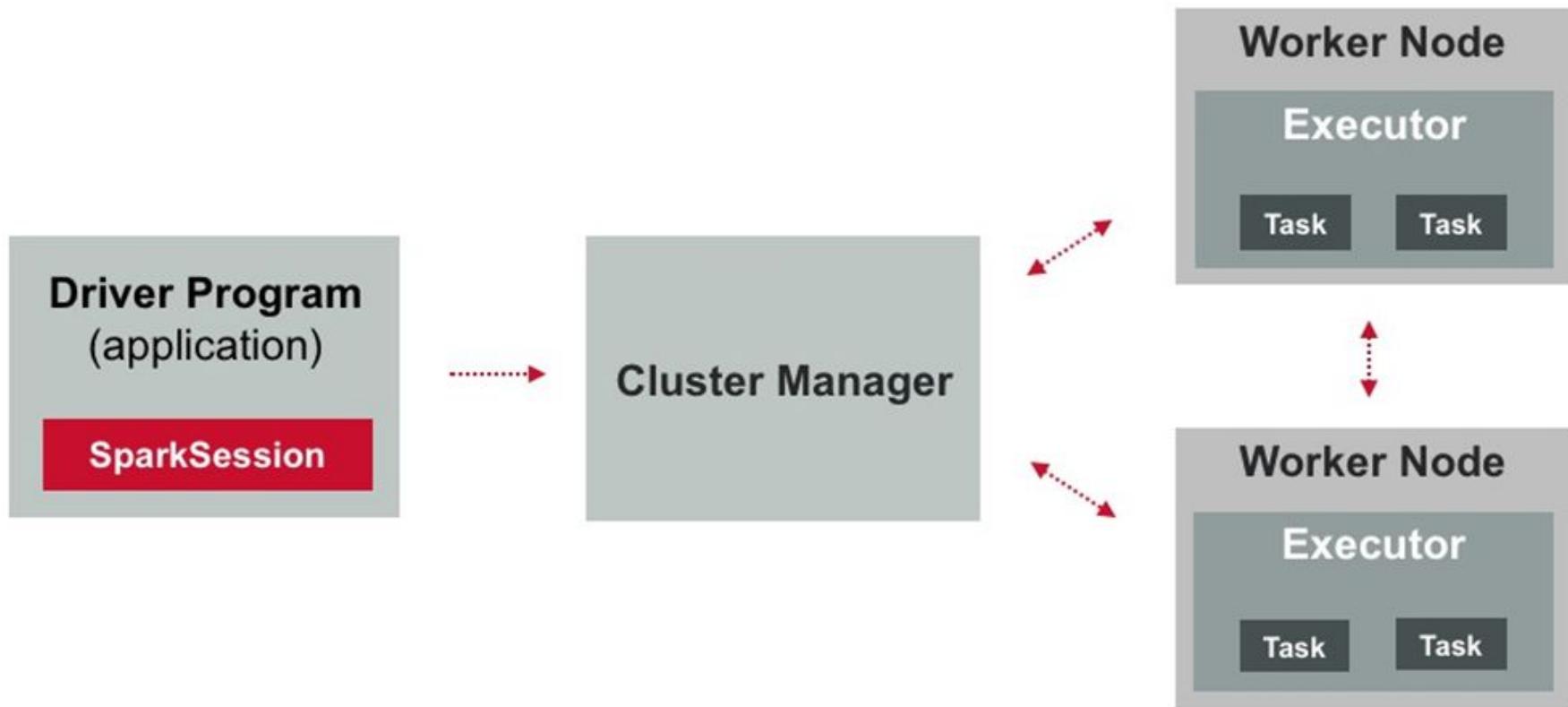
- A. val  
burglaries=sfpdDS.filter(line=>line.contains("BURGLARIES"))
- B. val  
sfpdDS=spark.read.textFile("/user/user01/data/sfpd.csv")
- C. val totburglaries=burglaries.count()
- D. burglaries.cache()

# 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()
```



# 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



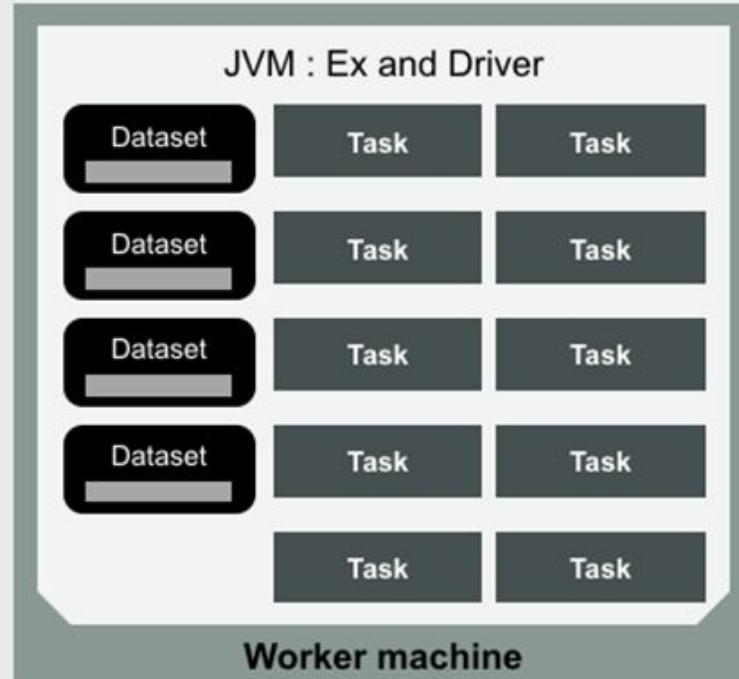
- Estimated completion time: **25 minutes**
- In this lab, you will download application files and import them into the IntelliJ IDE for use in a subsequent lab.

# 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

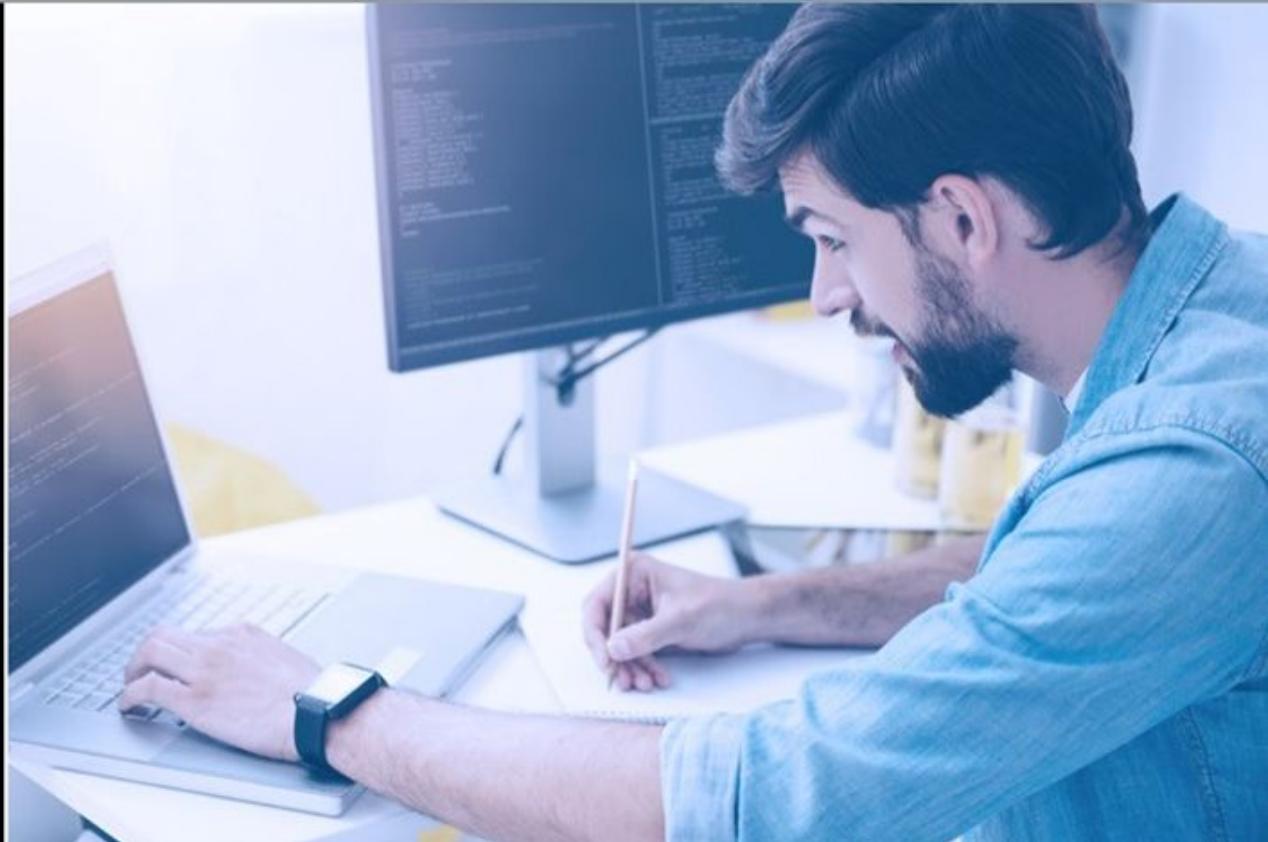
# Mode 1: Local

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



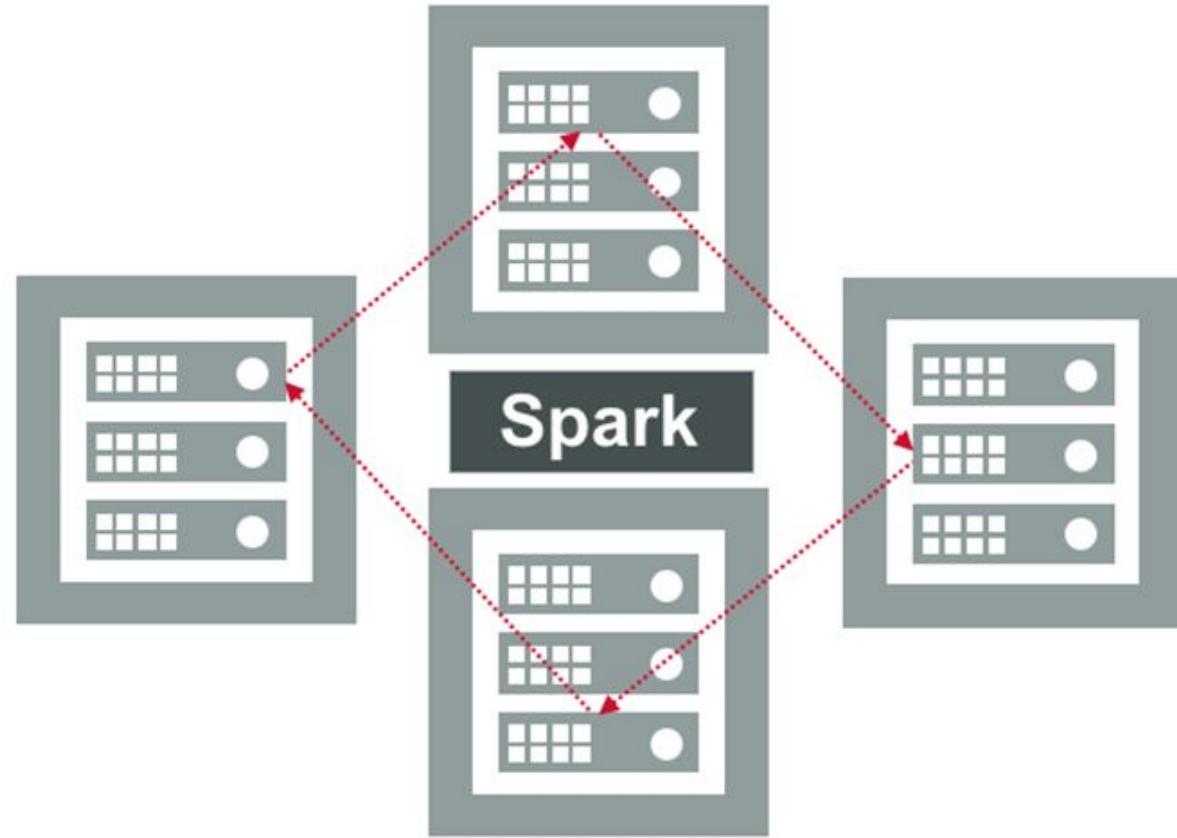
# Mode 1: Local

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

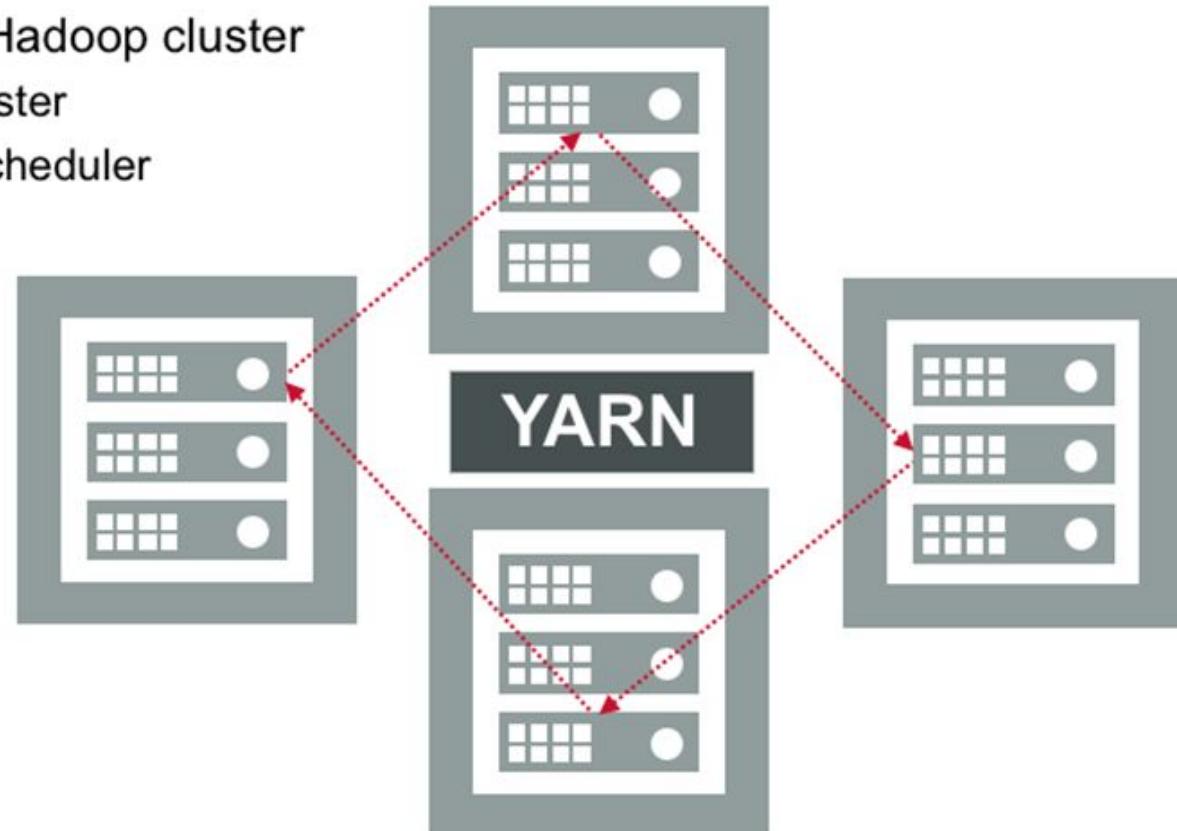
# Standalone: Deployment Modes

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 a table of application logs. The table has columns: App ID, App Name, Started, Completed, Duration, Spark User, Last Updated, and Event Log. One entry is listed:

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	<a href="#">Download</a>

Below the table, there are 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.





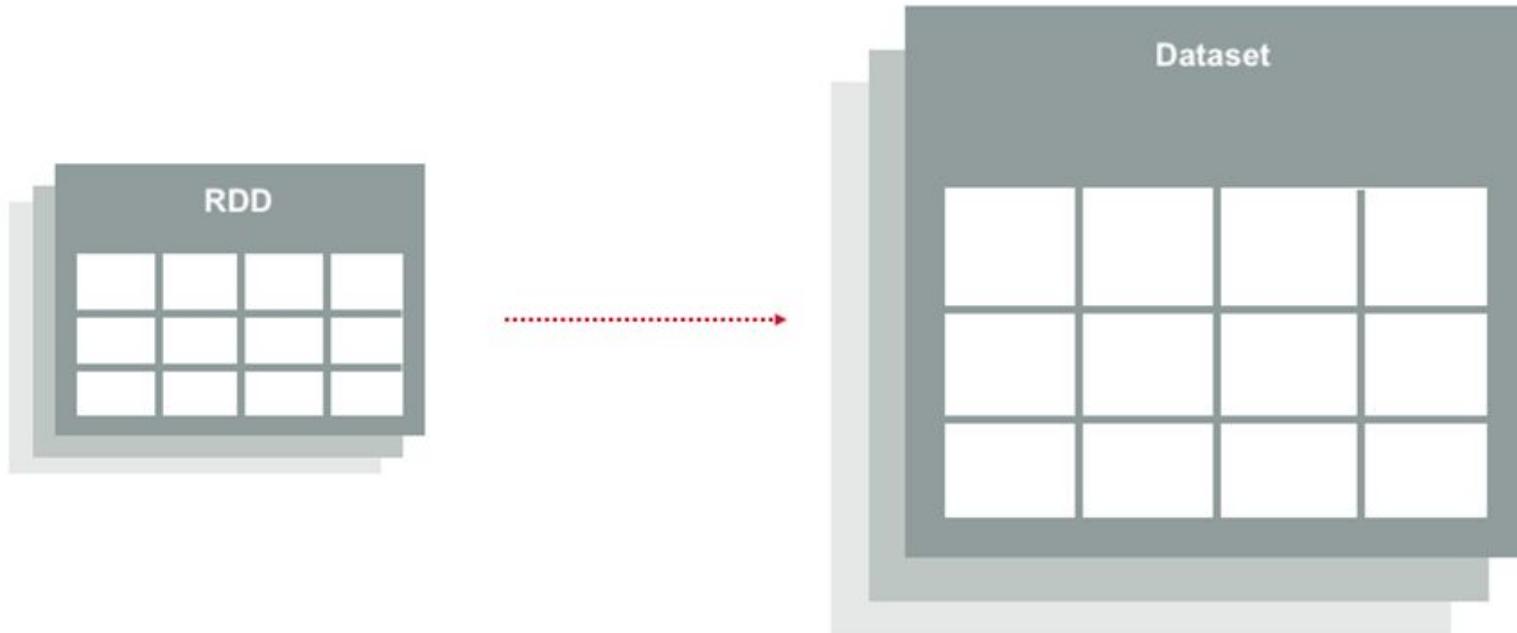
# Learning Goals

- 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

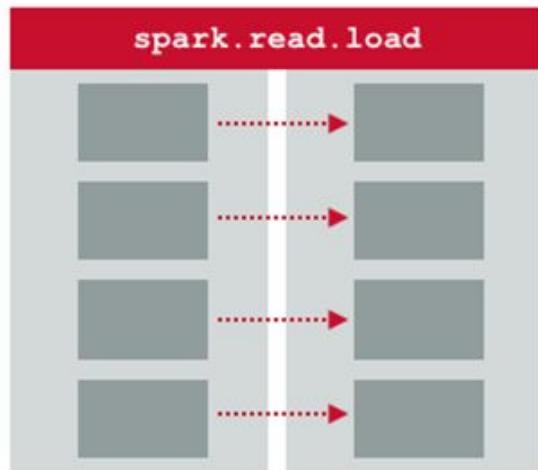
Tasks scheduled and  
executed on cluster

# 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()

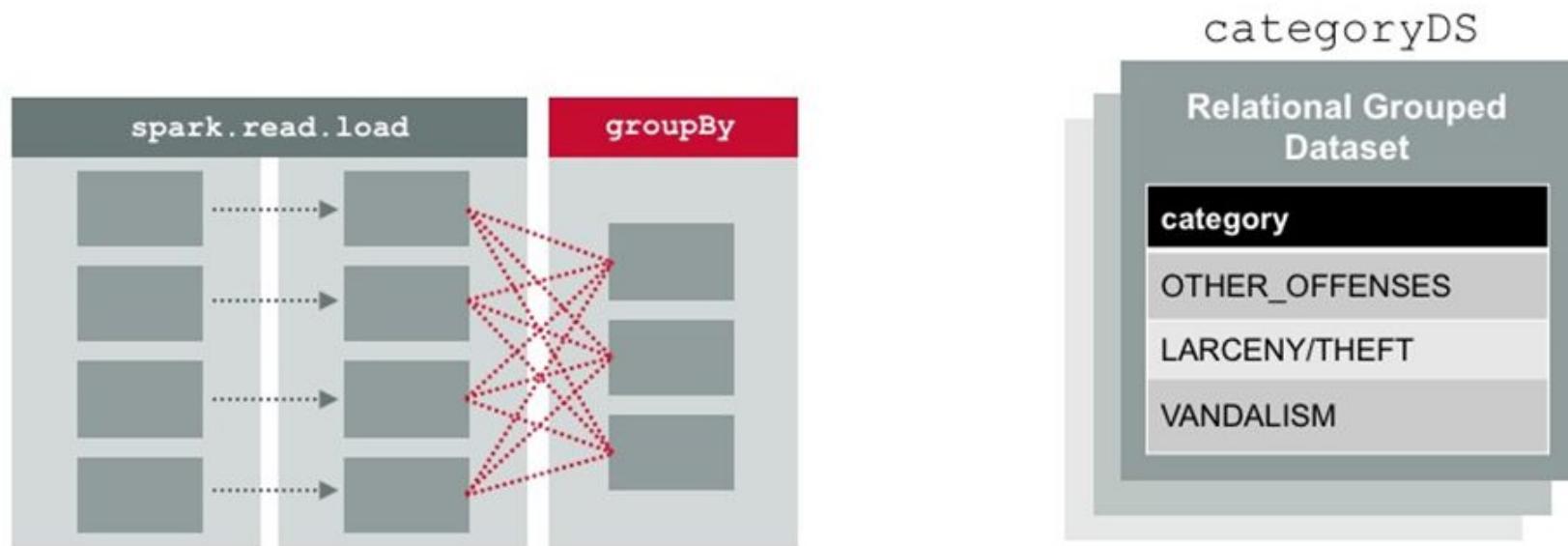
# Logical Plan Step 1: Import Data

```
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")
```



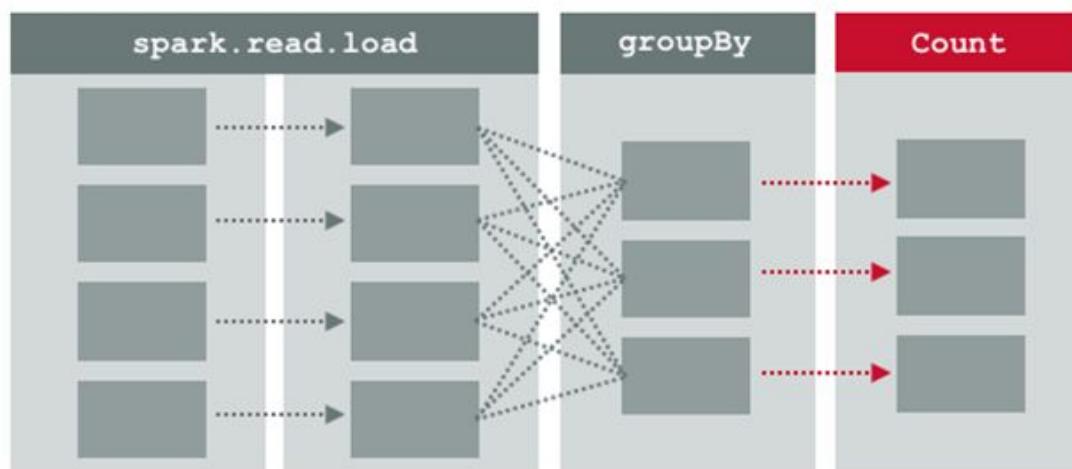
# Logical Plan Step 2: Apply groupBy Transformation

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



# Logical Plan Step 3: Apply count Transformation

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

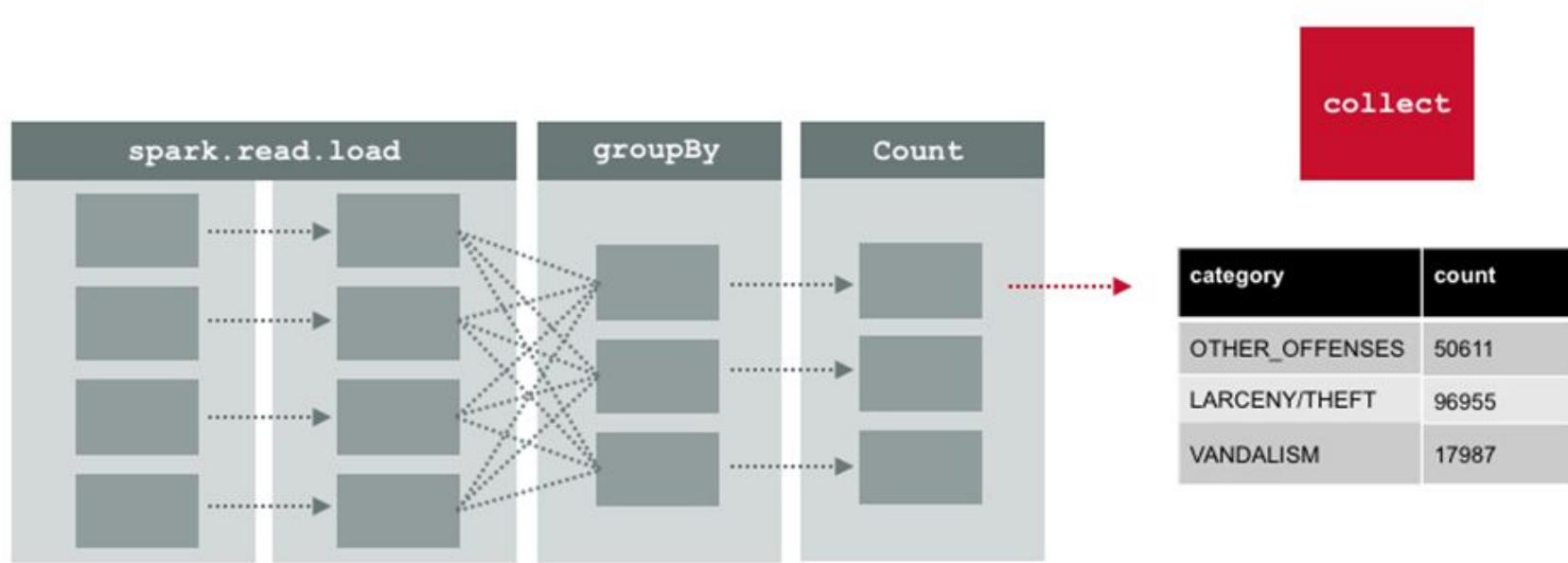


categoryCount

Dataset	
category	count
OTHER_OFFENSES	
LARCENY/THEFT	
VANDALISM	

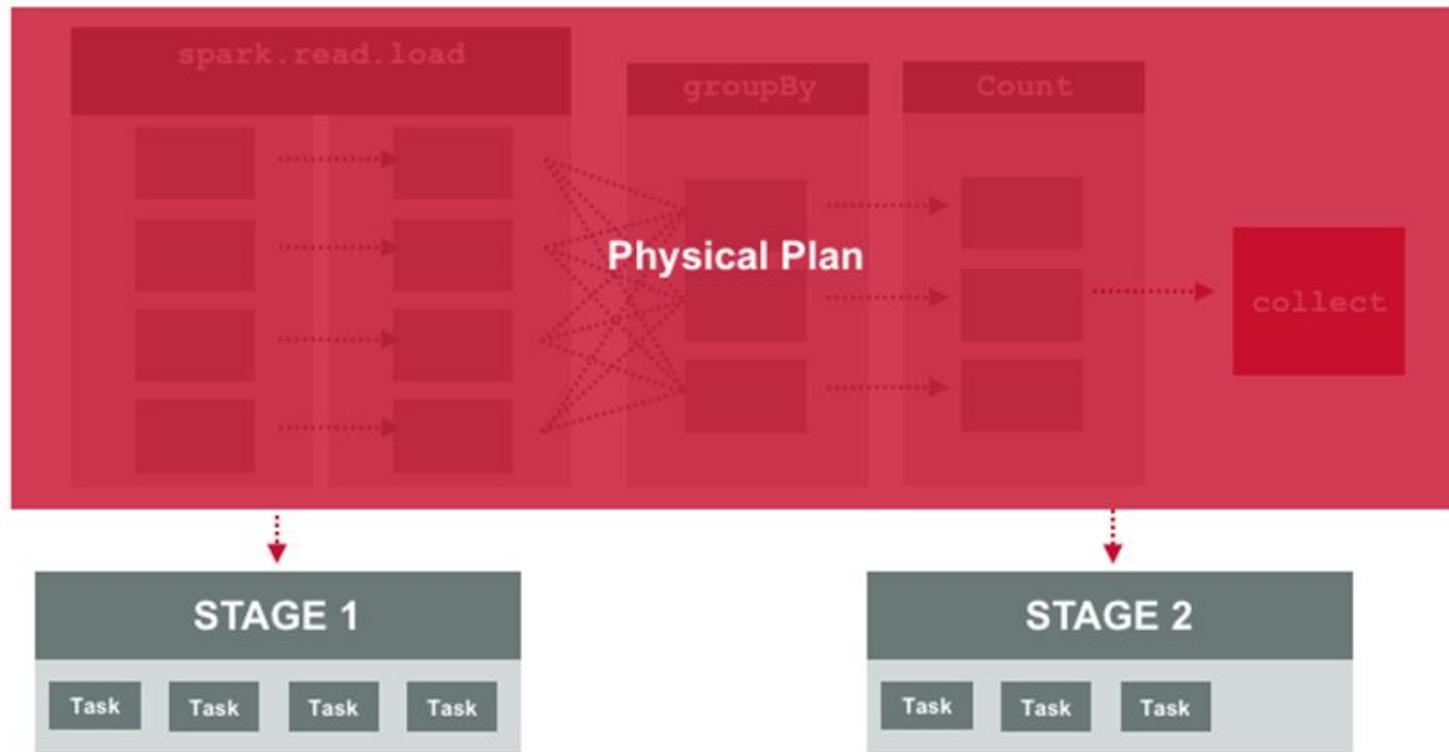
# Logical Plan Step 4: Apply collect Action

4. categoryCount.collect()



# Physical Plan

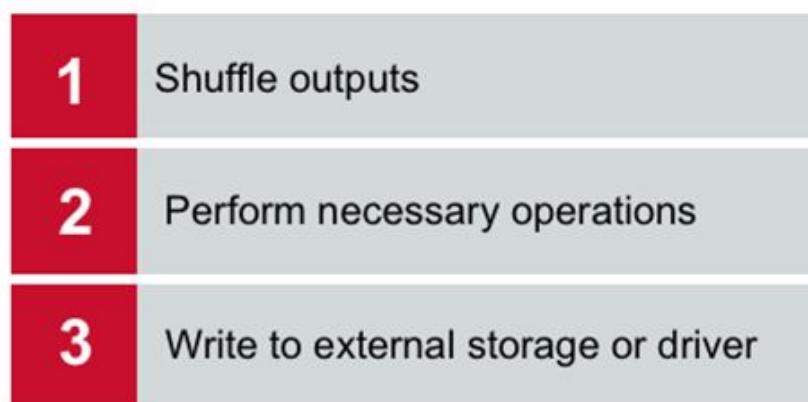
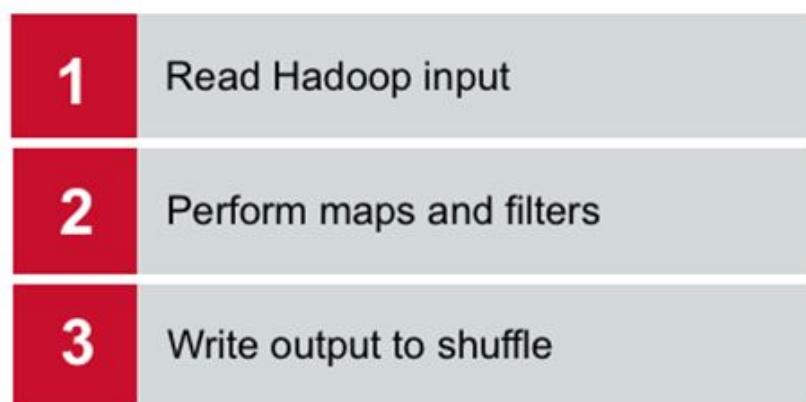
When an action is encountered, the DAG is translated into a physical plan.



# 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

Available on machine where driver is running

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)

Page: <a href="#">1</a> <a href="#">2</a> > 2 Pages, Jump to <input type="text"/> Show <a href="#">100</a> items in a page. Go										
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	

# 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&lt;_c0:int&gt;</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-1787 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

APACHE 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

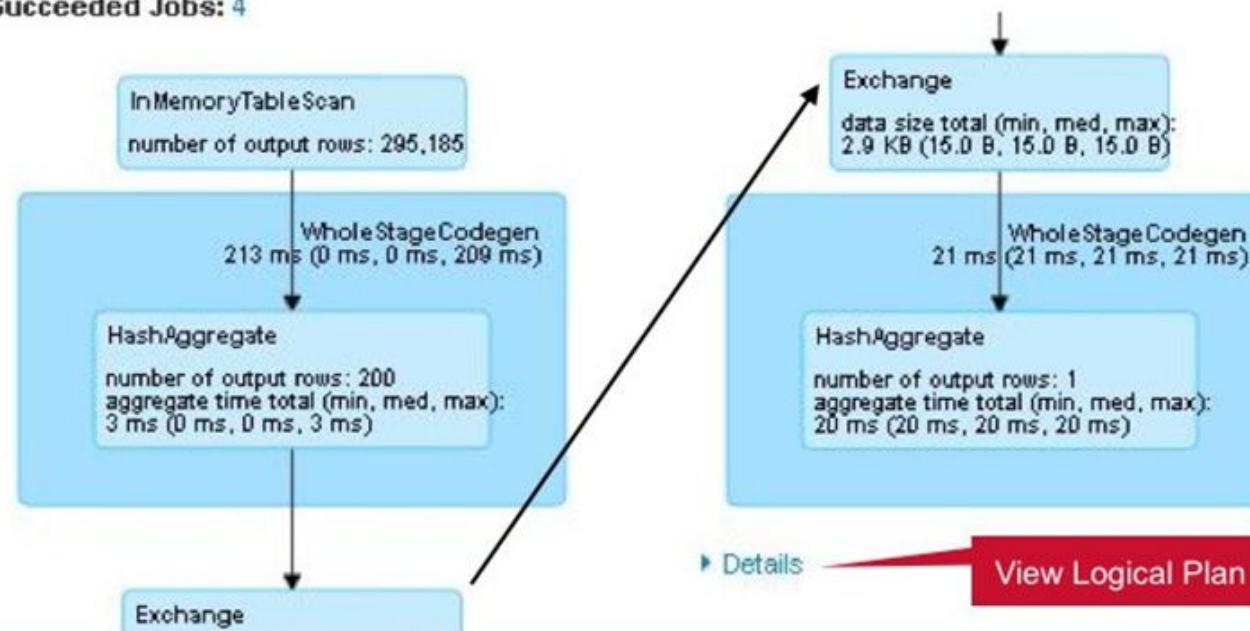
# Spark Web UI: SQL Page

## 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.



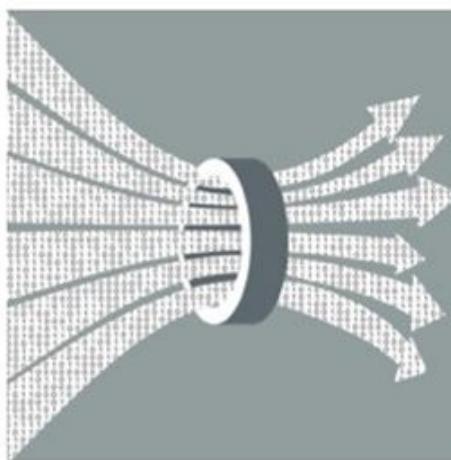
# Learning Goals

- 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

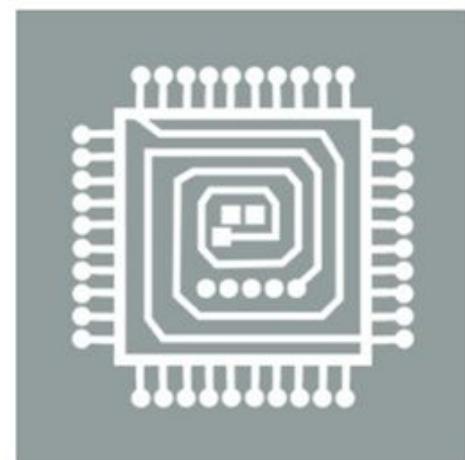
# Common Slow Performance Issues



**Level of  
Parallelism**

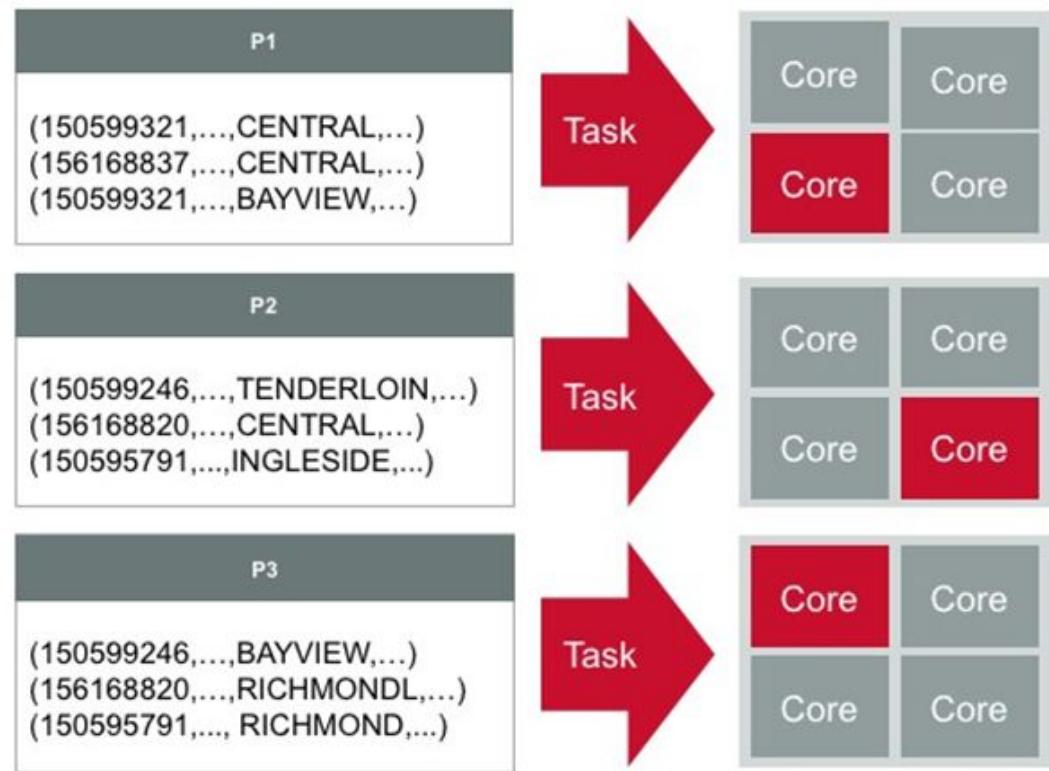


**Serialization  
Format**



**Memory  
Management**

# Review: Partitioning



# Common Performance Issues: Level of Parallelism



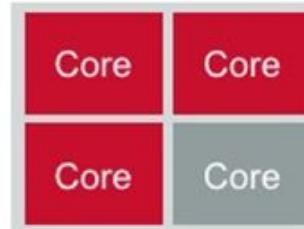
**Level of  
Parallelism**



# Common Performance Issues: Level of Parallelism



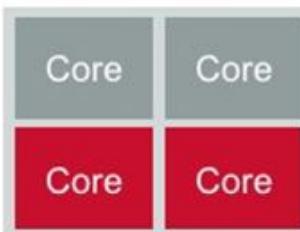
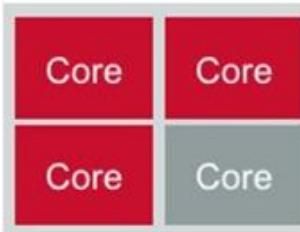
**Level of  
Parallelism**



# 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

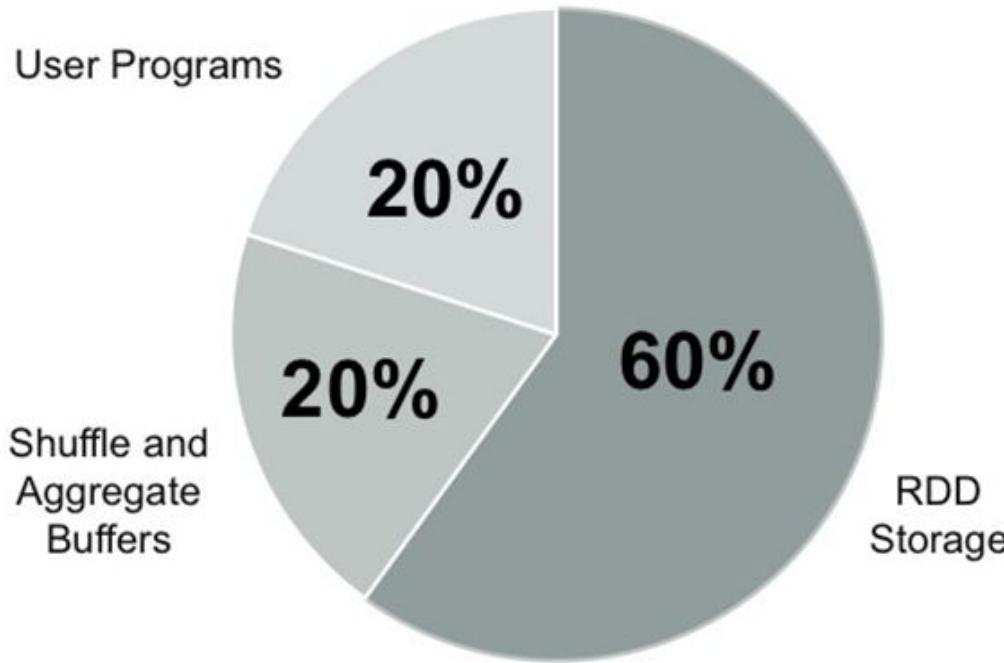
**Serialization  
Format**

# Common Performance Issues: Memory Management

Memory used in different ways in Spark



**Memory  
Management**



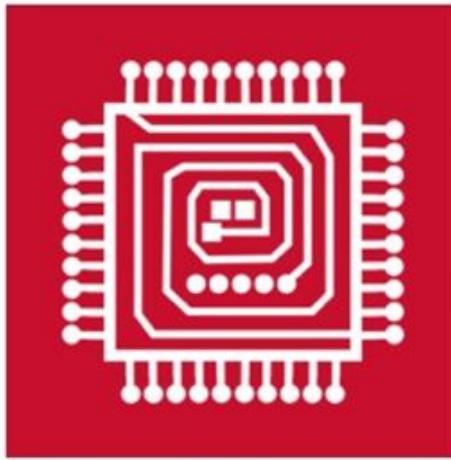
# Tuning Memory Usage



## 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

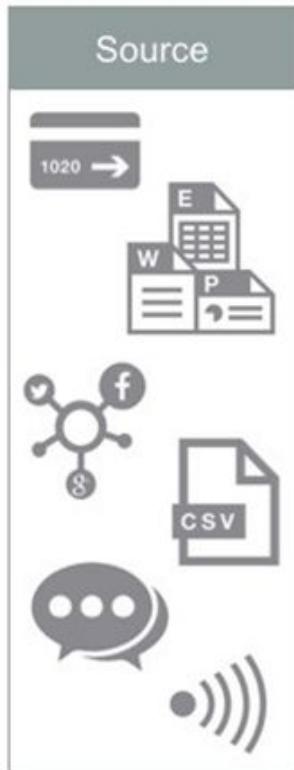
- 6.1 Describe Spark Streaming Architecture
- 6.2 Create a Spark Structured Streaming Application
  - Define Use Case
  - Basic Steps and Save Data to Parquet Tables
- 6.3 Apply Operations on Streaming DataFrames
- 6.4 Define Windowed Operations
- 6.5 Describe How Streaming Applications are Fault Tolerant



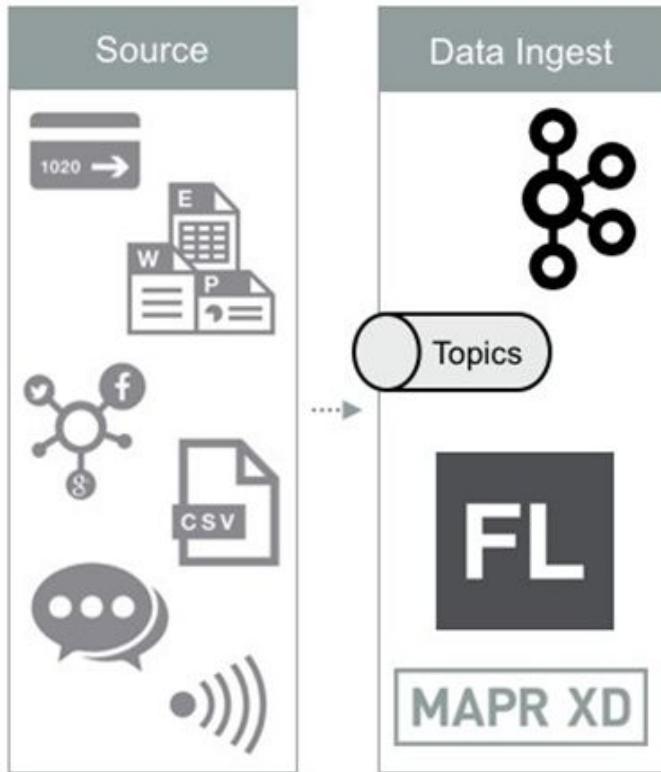
# Learning Goals

- 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

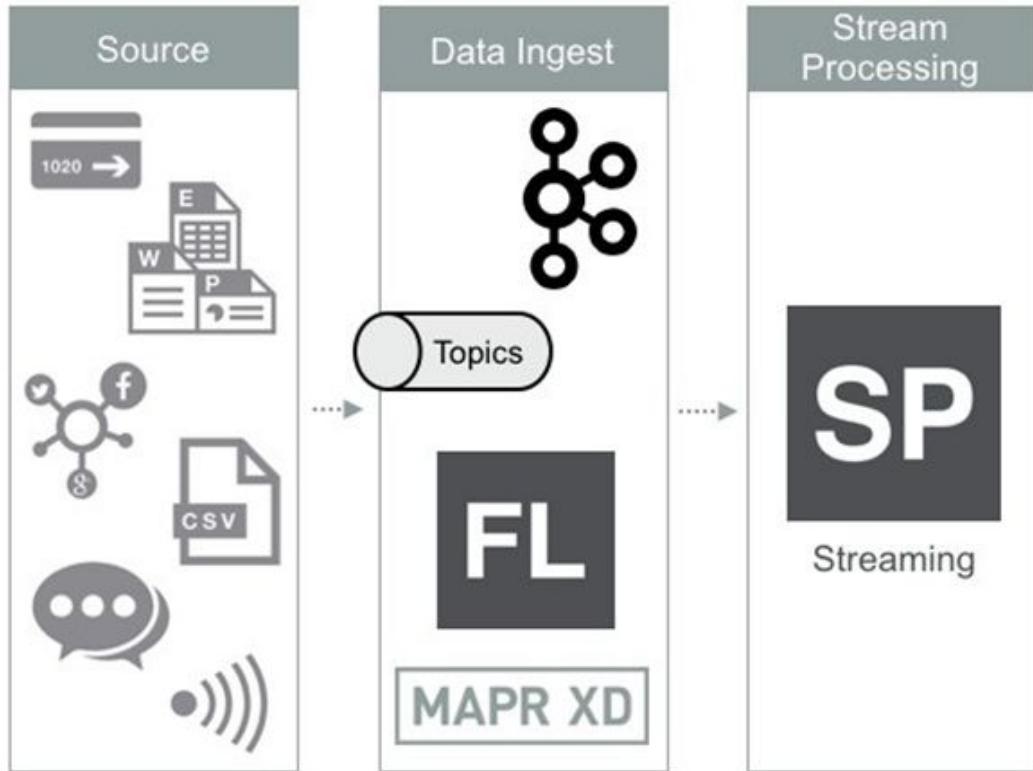
# Stream Processing Architecture: Streaming Data Source



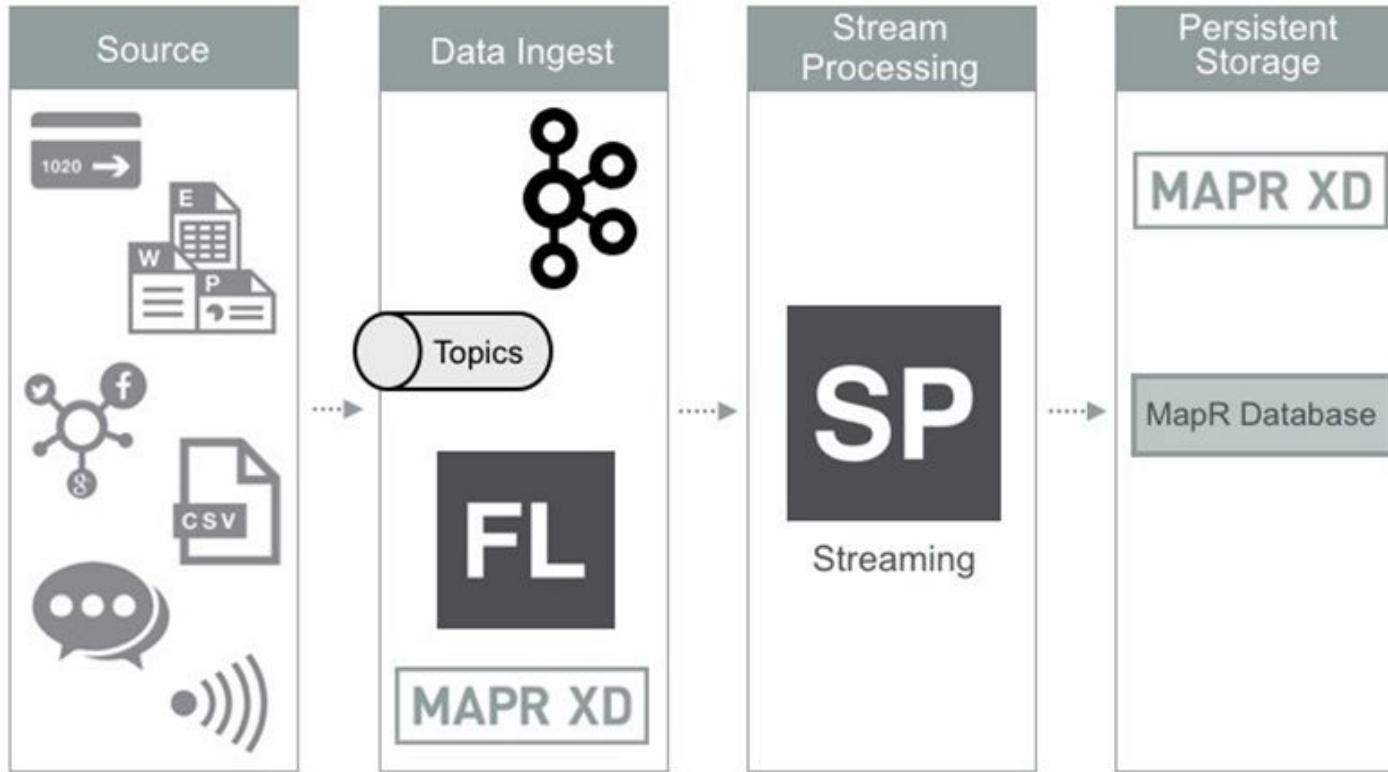
# Stream Processing Architecture: Ingest Data



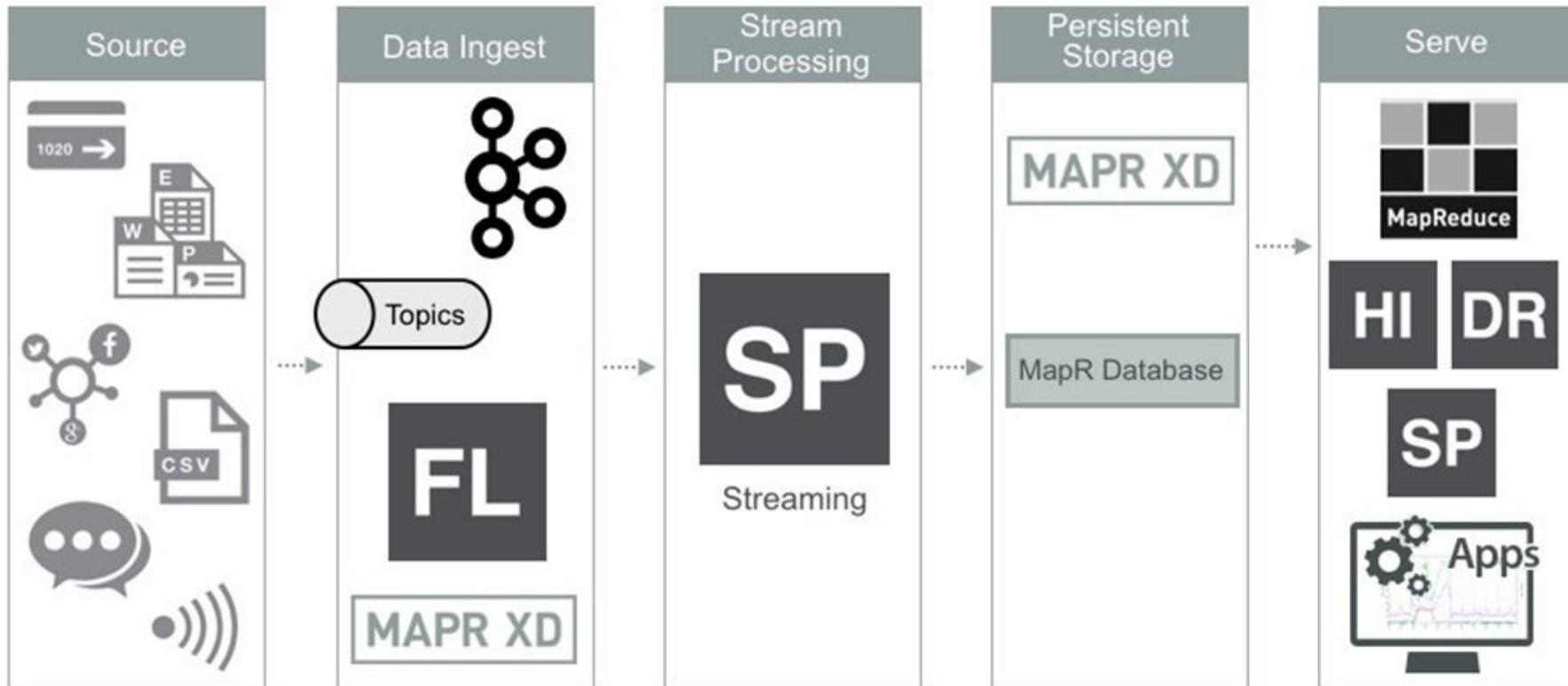
# Stream Processing Architecture: Process Data



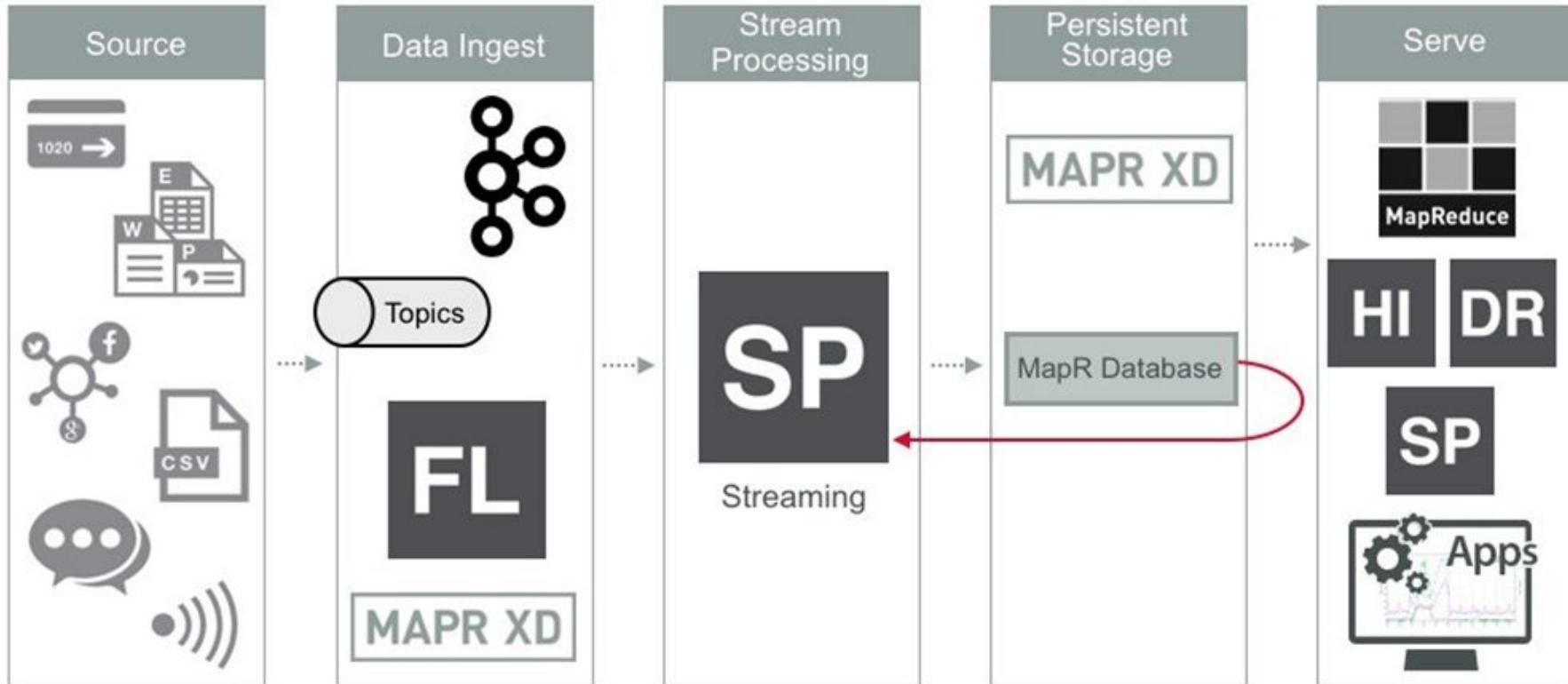
# Stream Processing Architecture: Store Processed Data



# Stream Processing Architecture: Visualize Processed Data



# Stream Processing Architecture: Store Output



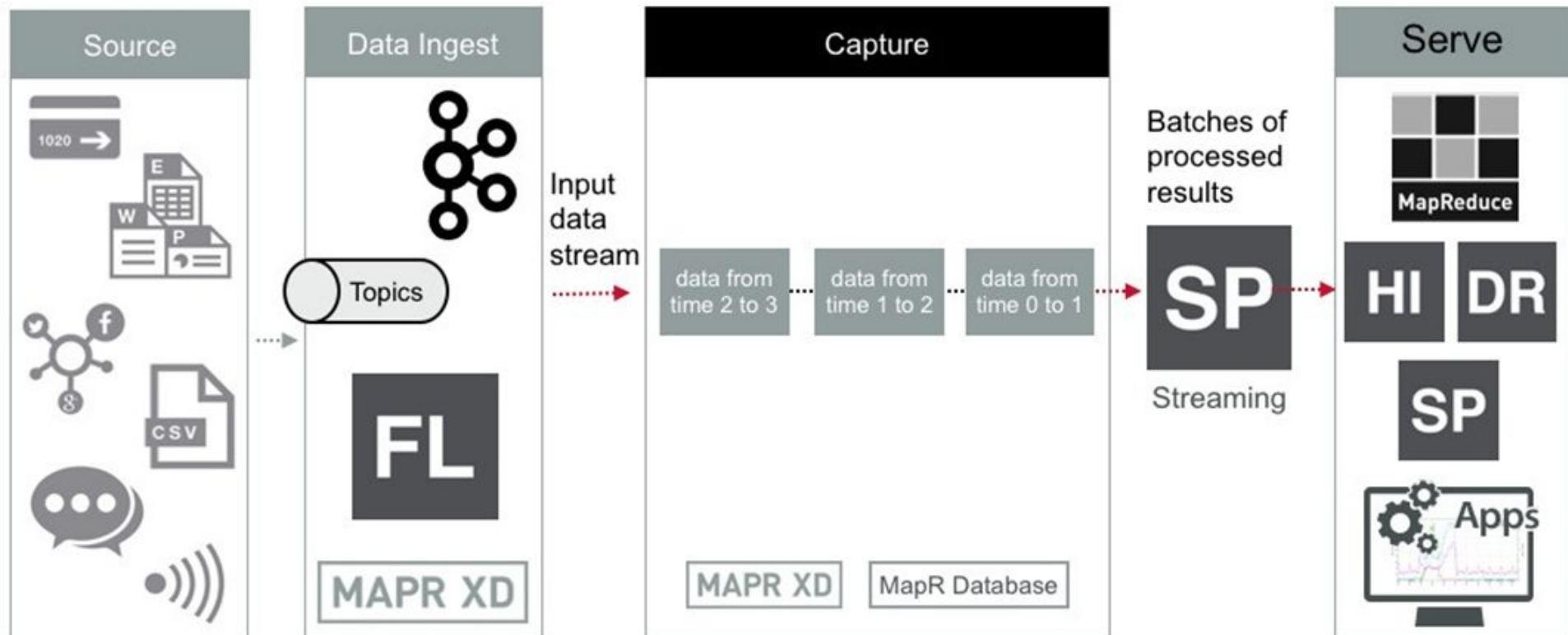
## Knowledge Check



**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					
	<b>id</b>	<b>first</b>	<b>last</b>	<b>amt</b>	<b>city</b>	...
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					
	<b>id</b>	<b>first</b>	<b>last</b>	<b>amt</b>	<b>city</b>	...
	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

<b>id</b>	<b>first</b>	<b>last</b>	<b>amt</b>	<b>city</b>	<b>...</b>
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	

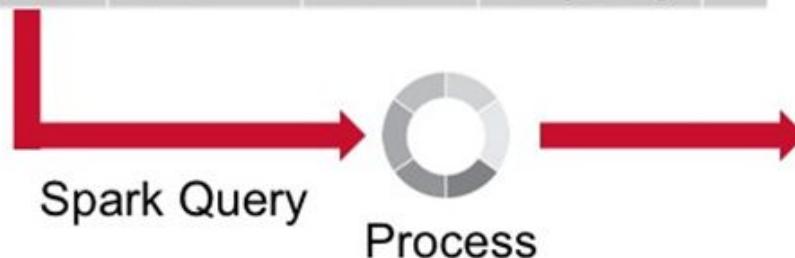
Spark  
Query



# 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

<b>id</b>	<b>first</b>	<b>last</b>	<b>amt</b>	<b>city</b>	<b>...</b>
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

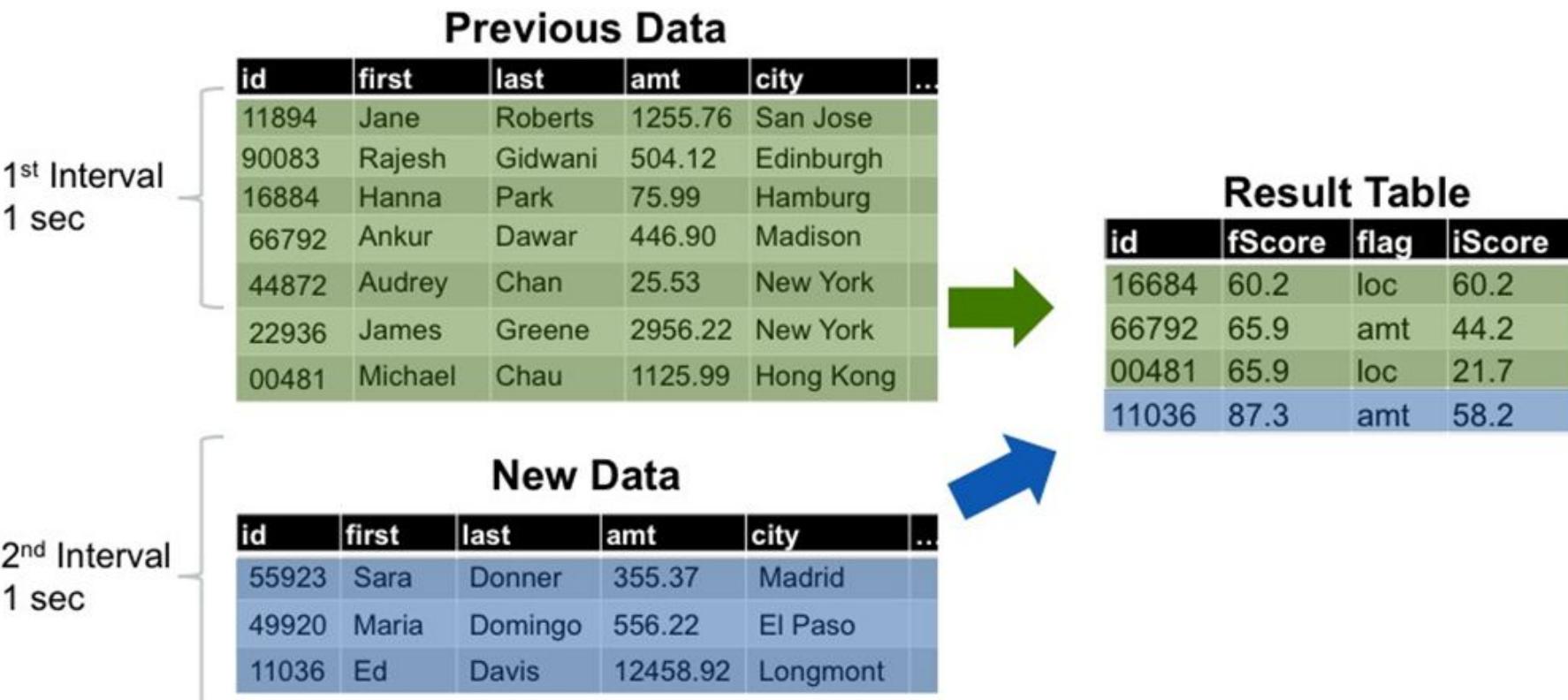
Previous Data

New Data

<b>id</b>	<b>first</b>	<b>last</b>	<b>amt</b>	<b>city</b>	<b>...</b>
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



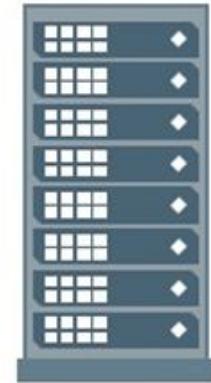
# Save Output to Storage

Save output as:

- Complete
- Append
- Update

**Result Table**

<b>id</b>	<b>fScore</b>	<b>flag</b>	<b>iScore</b>
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

<b>id</b>	<b>amount</b>	<b>city</b>
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison

<b>id</b>	<b>amount</b>	<b>city</b>
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison
00481	1125.99	Hong Kong

new row

Result Table

<b>id</b>	<b>fScore</b>	<b>flag</b>	<b>iScore</b>
16684	60.2	loc	60.2
66792	446.90	amt	44.2

<b>id</b>	<b>fScore</b>	<b>flag</b>	<b>iScore</b>
16684	70.1	loc	60.2
66792	65.9	amt	44.2
00481	65.9	loc	21.7

updated row

Output Saved

<b>id</b>	<b>fScore</b>	<b>flag</b>	<b>iScore</b>
16684	60.2	loc	60.2
66792	446.90	amt	55.2

<b>id</b>	<b>fScore</b>	<b>flag</b>	<b>iScore</b>
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

<b>id</b>	<b>amount</b>	<b>city</b>
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison

Time Interval 2

Time Interval 3

<b>id</b>	<b>amount</b>	<b>city</b>
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison
00481	1125.99	Hong Kong

new row

Result Table

<b>id</b>	<b>fScore</b>	<b>flag</b>	<b>iScore</b>
16684	60.2	loc	60.2
66792	446.90	amt	44.2

<b>id</b>	<b>fScore</b>	<b>flag</b>	<b>iScore</b>
16684	70.1	loc	60.2
66792	65.9	amt	44.2
00481	65.9	loc	21.7

updated row

Output Saved

<b>id</b>	<b>fScore</b>	<b>flag</b>	<b>iScore</b>
16684	60.2	loc	60.2
66792	446.90	amt	55.2

<b>id</b>	<b>fScore</b>	<b>flag</b>	<b>iScore</b>
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

<b>id</b>	<b>amount</b>	<b>city</b>
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison

Time Interval 2

Time Interval 3

<b>id</b>	<b>amount</b>	<b>city</b>
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison
00481	1125.99	Hong Kong

new row

Result Table

<b>id</b>	<b>fScore</b>	<b>flag</b>	<b>iScore</b>
16684	60.2	loc	60.2
66792	446.90	amt	44.2

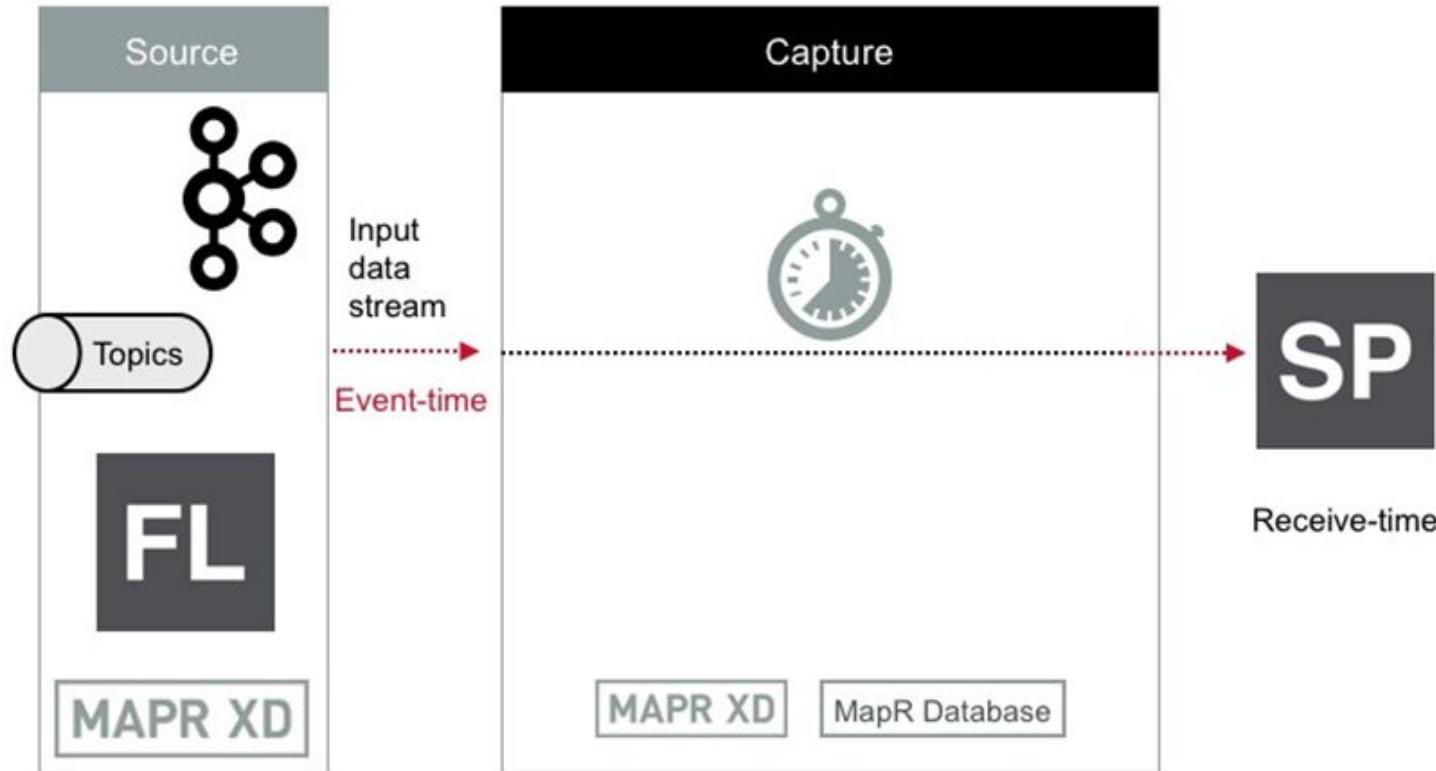
<b>id</b>	<b>fScore</b>	<b>flag</b>	<b>iScore</b>
16684	70.1	loc	60.2
66792	446.90	amt	44.2
00481	65.9	loc	21.7

Output Saved

<b>id</b>	<b>fScore</b>	<b>flag</b>	<b>iScore</b>
16684	60.2	loc	60.2
66792	446.90	amt	55.2

<b>id</b>	<b>fScore</b>	<b>flag</b>	<b>iScore</b>
16684	70.1	loc	60.2
00481	65.9	loc	21.7

# Handling Event-time



# Handling Late Data

<b>id</b>	<b>first</b>	<b>last</b>	<b>amt</b>	<b>city</b>	<b>...</b>
11894	Jane	Roberts	1255.76	San Jose	
90083	Rajesh	Gidwani	504.12	Edinburgh	
16884	Hanna	Park	75.99	Hamburg	

<b>id</b>	<b>first</b>	<b>last</b>	<b>amt</b>	<b>city</b>	<b>...</b>
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?

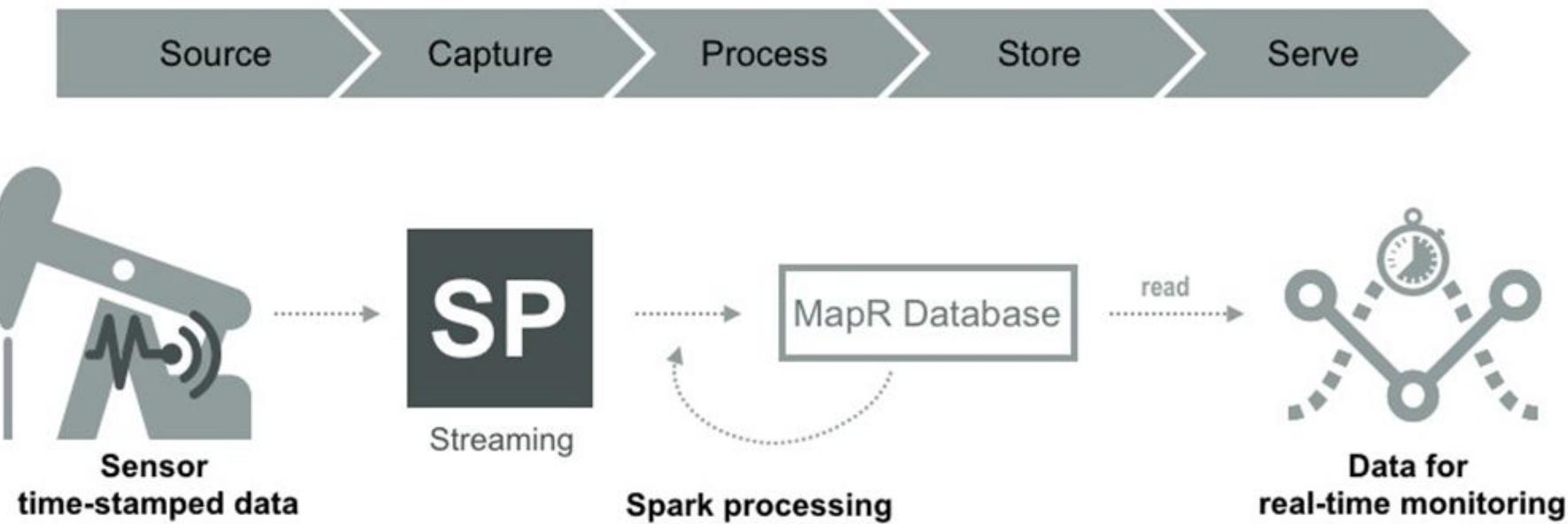
- A. Append
- B. Complete
- C. Overwrite
- D. Update



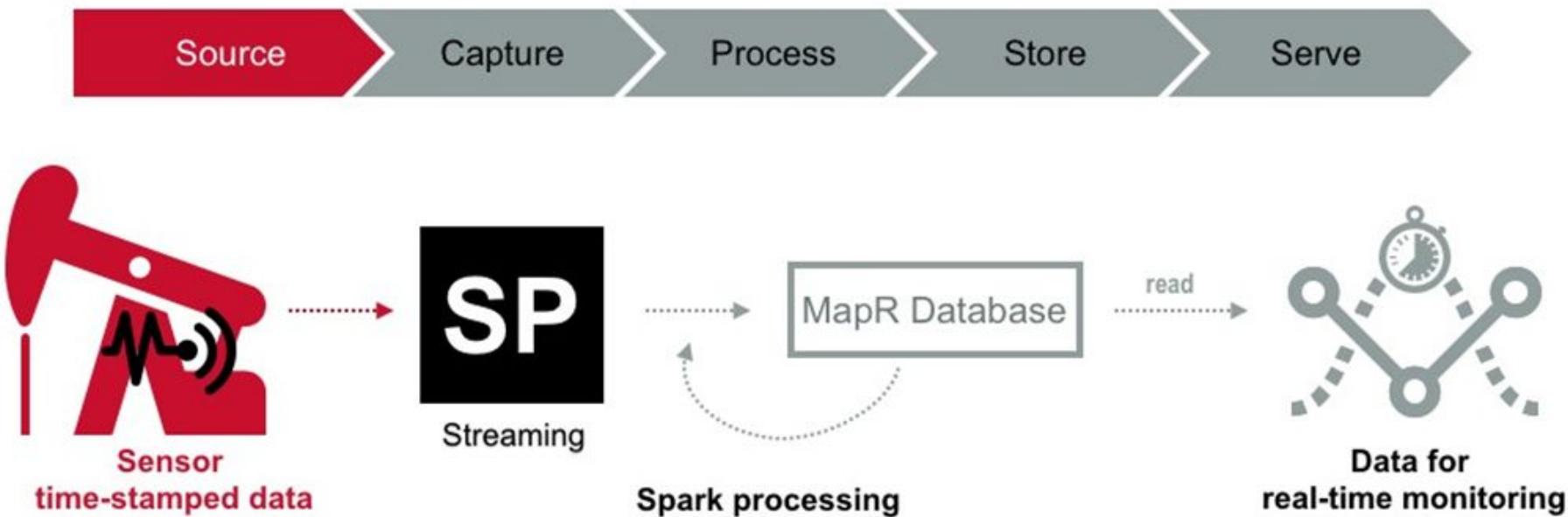
# Learning Goals

- 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

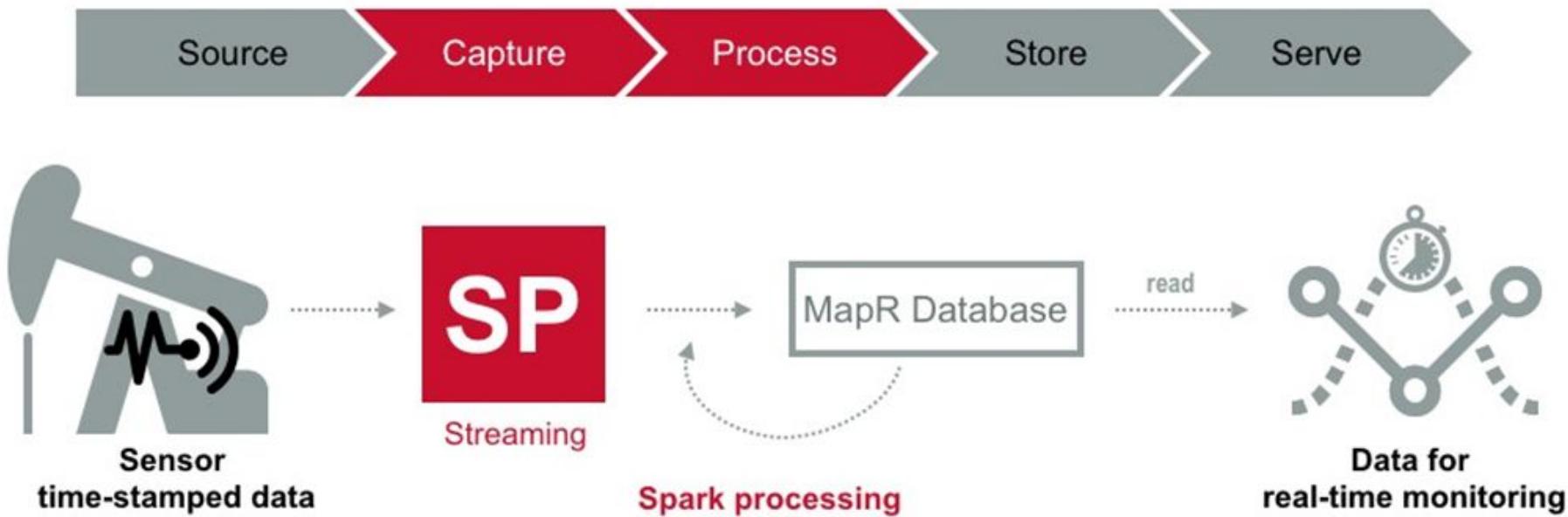
# Use Case: Time Series Data



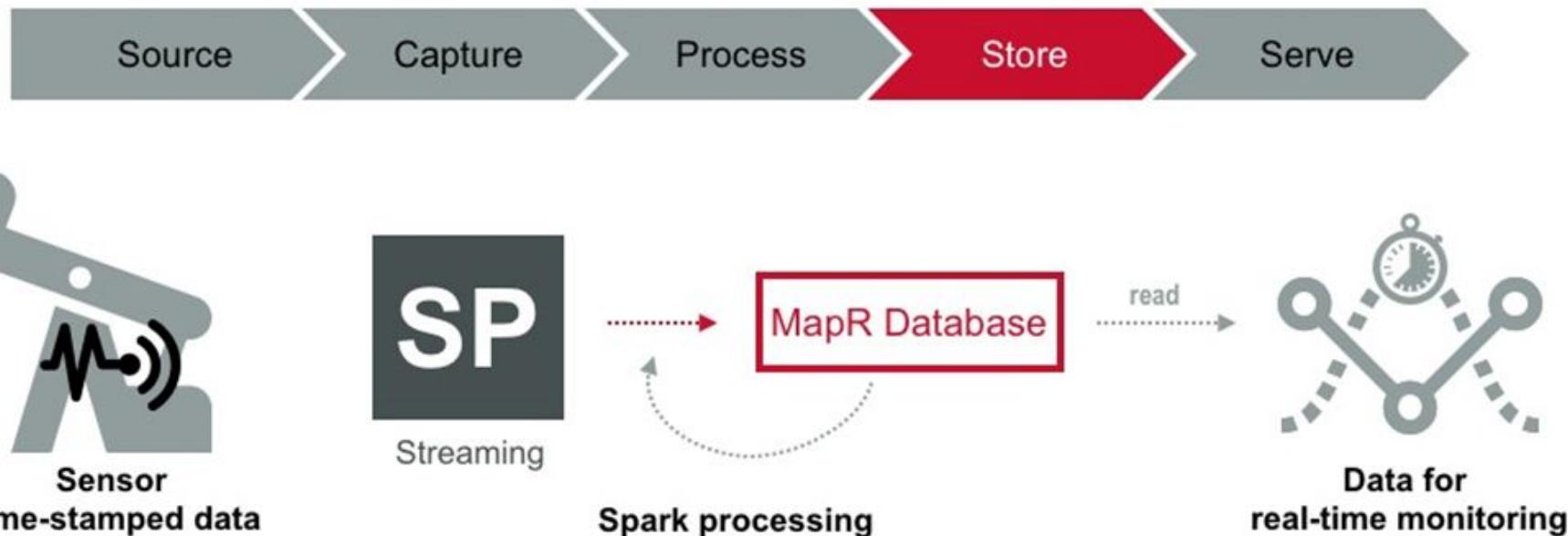
# Use Case: Time Series Data



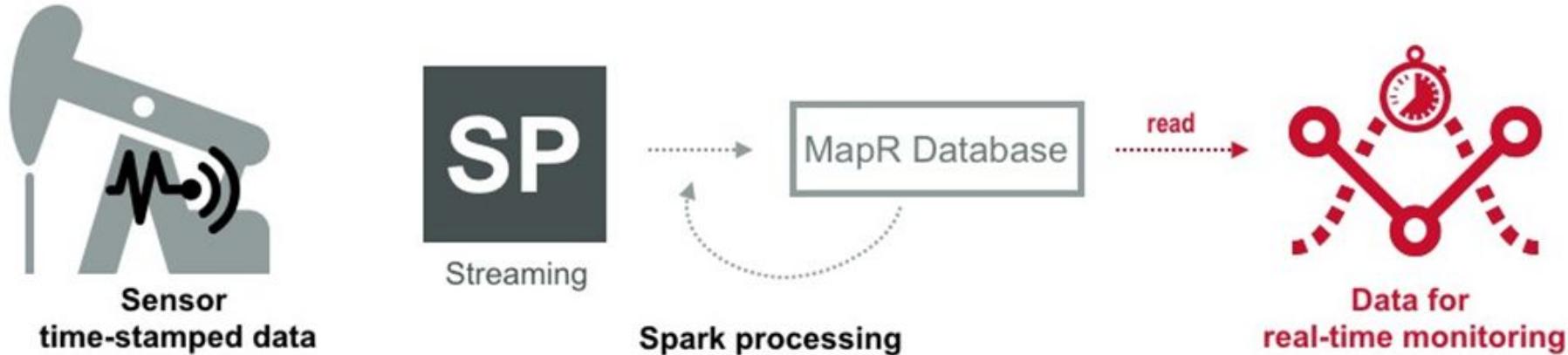
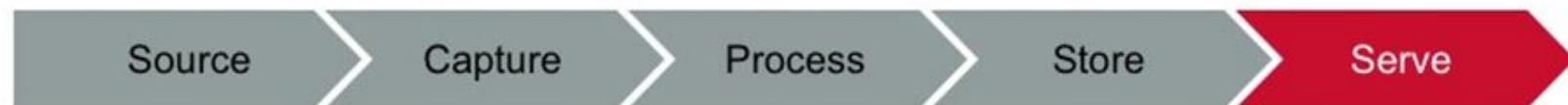
# Use Case: Time Series Data



# 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      ×

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



# Learning Goals

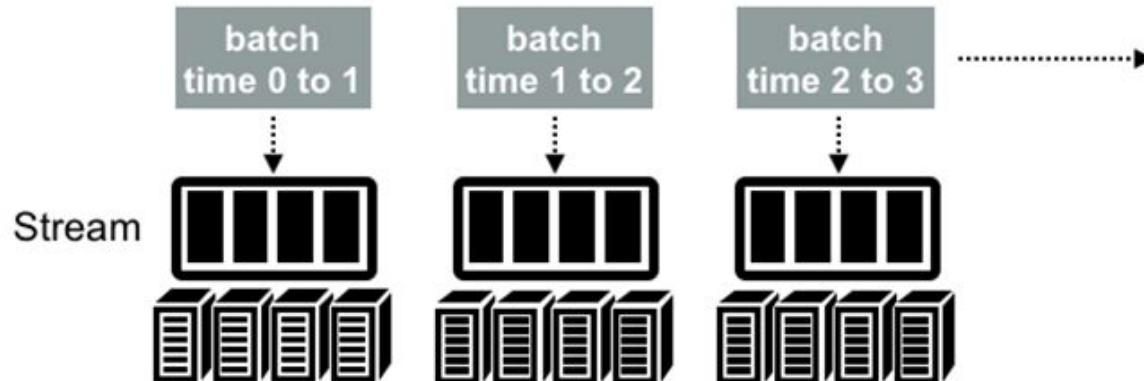
- 6.1 Describe Spark Streaming Architecture
- 6.2 Create a Spark Structured Streaming Application
  - Define Use Case
  - **Basic Steps and Save Data to Parquet Tables**
- 6.3 Apply Operations on Streaming DataFrames
- 6.4 Define Windowed Operations
- 6.5 Describe How Streaming Applications are Fault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

The screenshot shows a terminal window with a light gray background and a dark gray border. In the top-left corner are three small colored icons: red, yellow, and green. In the top-right corner is a user icon followed by the text "cmcDonald -". The main area of the window contains a table of data with the following columns: resid, date, psi, pumpType, purchaseDate, serviceDate, vendor, eventDate, technician, and description. The data consists of 20 identical rows, each representing a maintenance event for a pump at location LAGNAPPE. The rows are as follows:

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

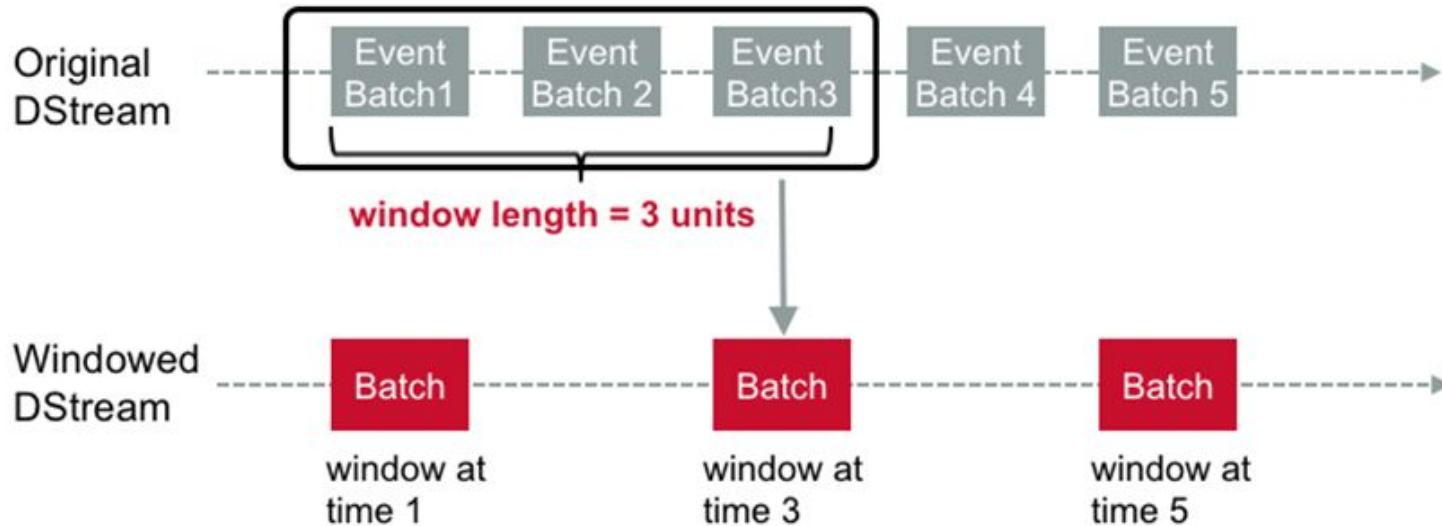
# Sliding Windows

---

- 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

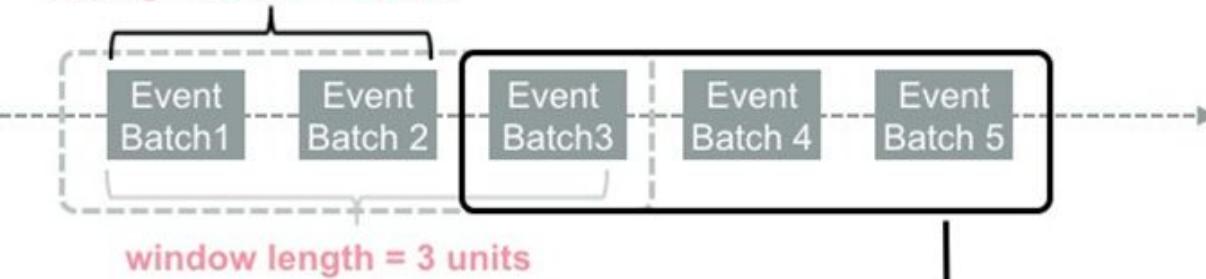


# Windowed Computations

batch interval = 1 second

sliding interval = 2 units

Original  
DStream



window length = 3 units

Windowed  
DStream

Batch

Batch

Batch

window at  
time 1

window at  
time 3

window at  
time 5

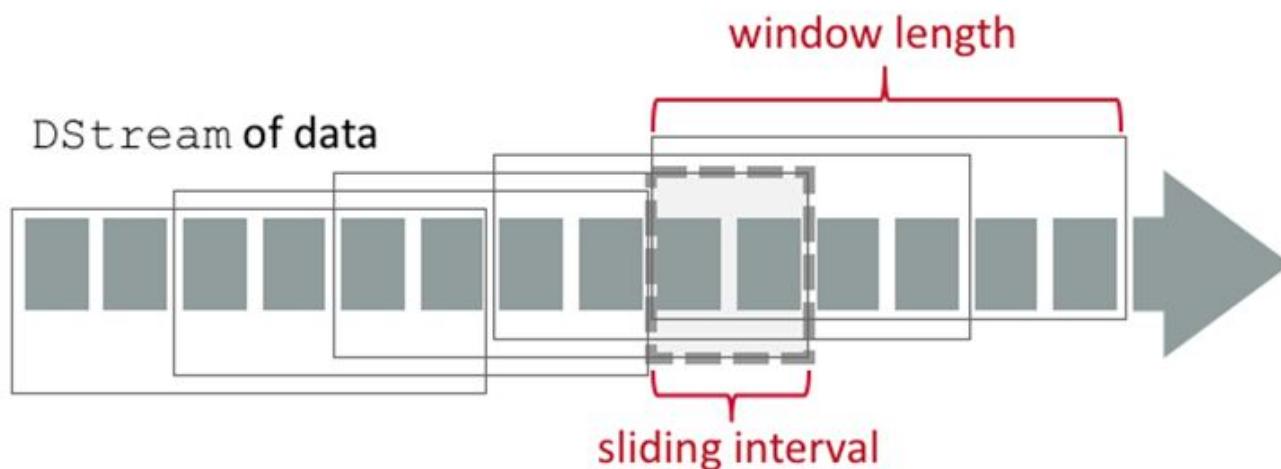
# Window-based Transformations

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

sliding window operation

window length

sliding interval



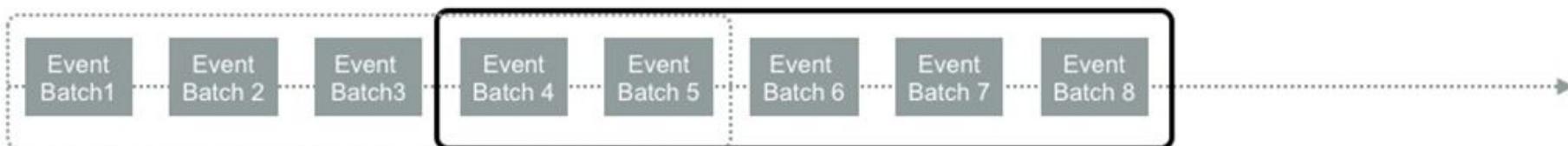
# Window Operations

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.

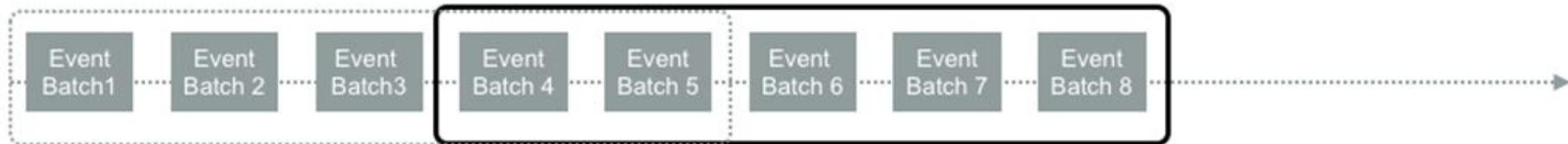
# Window Operations on DStreams

With a windowed stream:

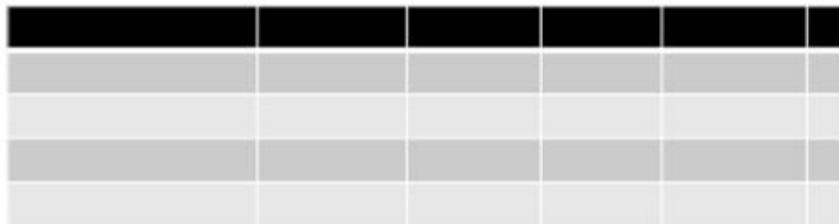
- What is the count of sensor events by pump ID?
- What is the Max, Min, and Average for PSI?



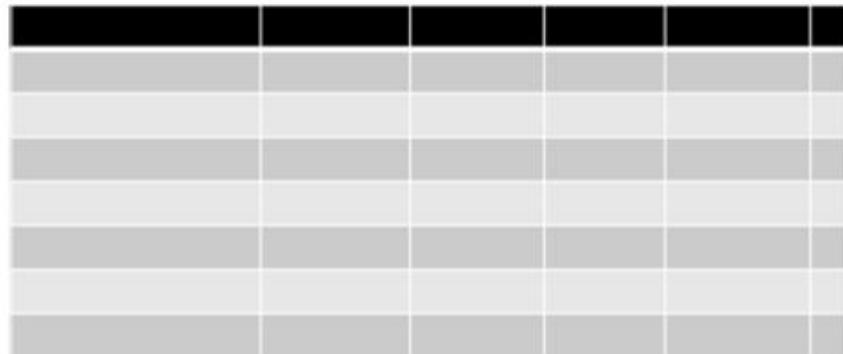
# Handling Event-time



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

```
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



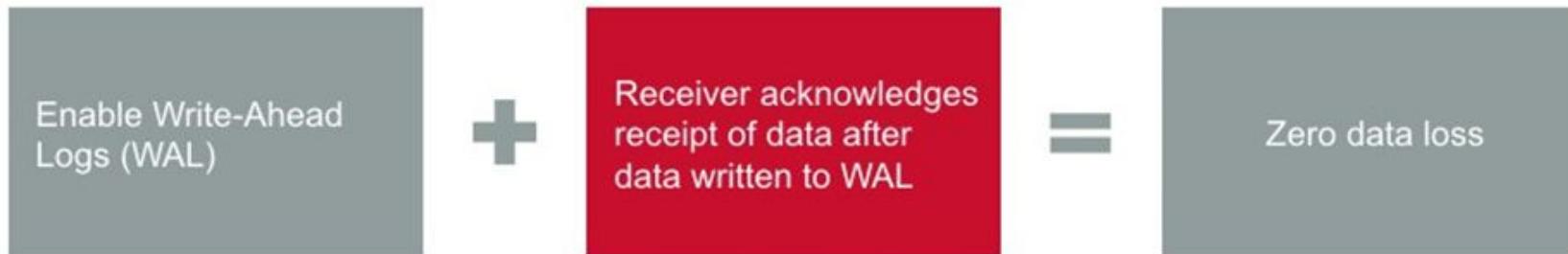
# Fault Tolerance in Write-Ahead Logs

- 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

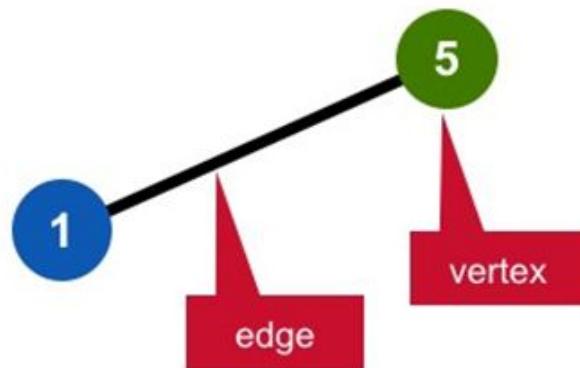
# Learning Goals



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

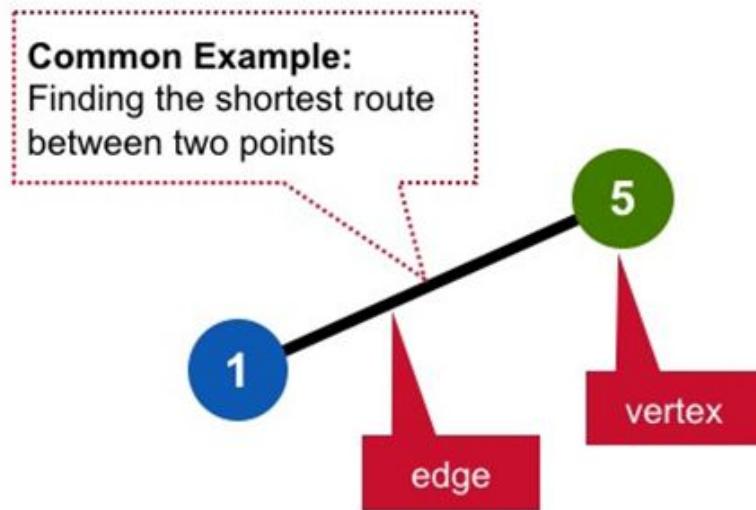
# What is a Graph?

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

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

## 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

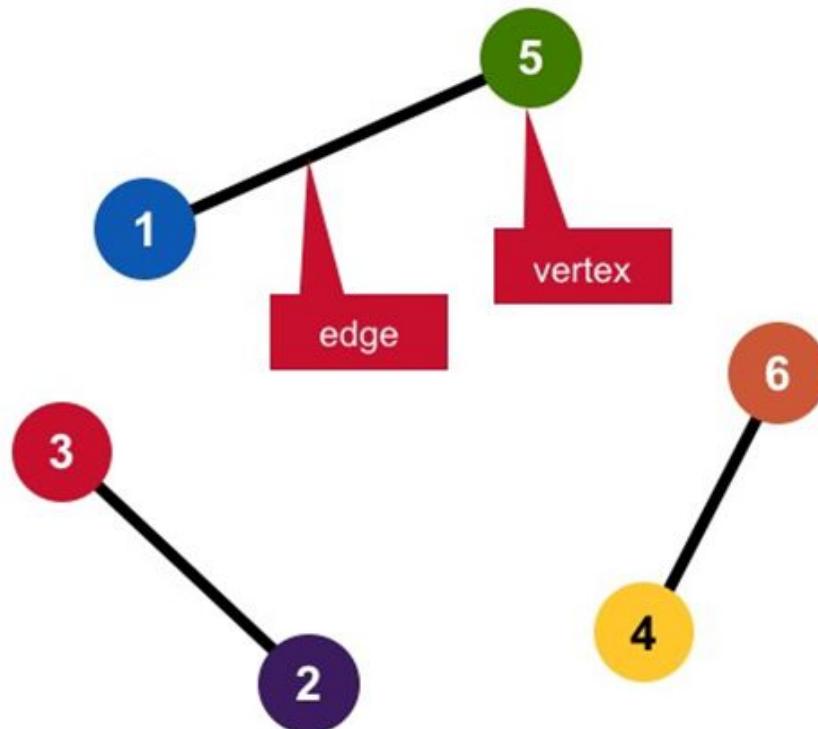


# Learning Goals

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

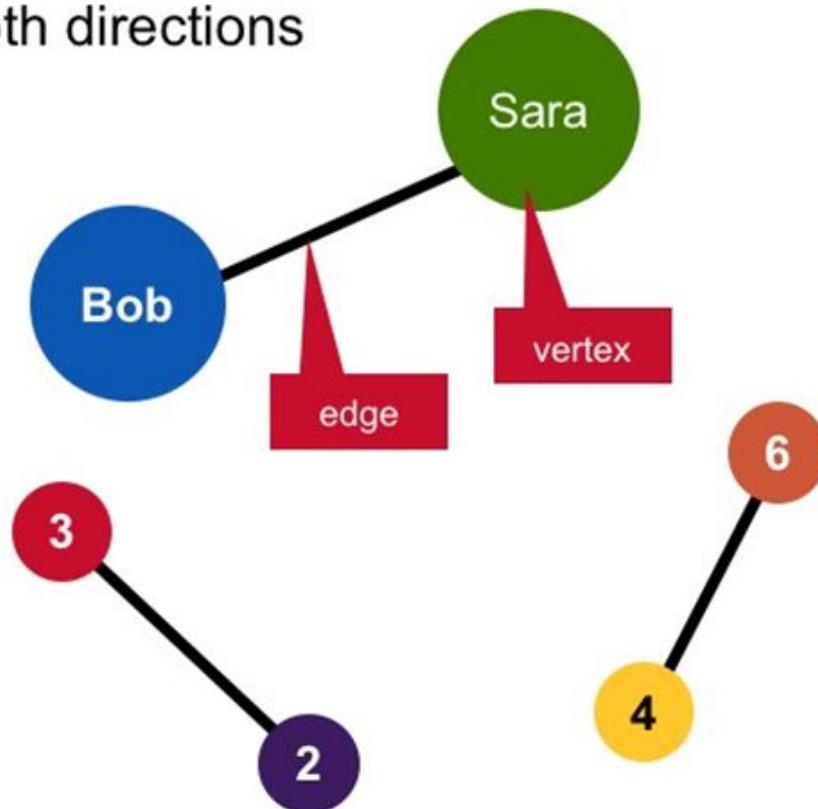
# Regular Graphs

Each vertex has the same number of edges



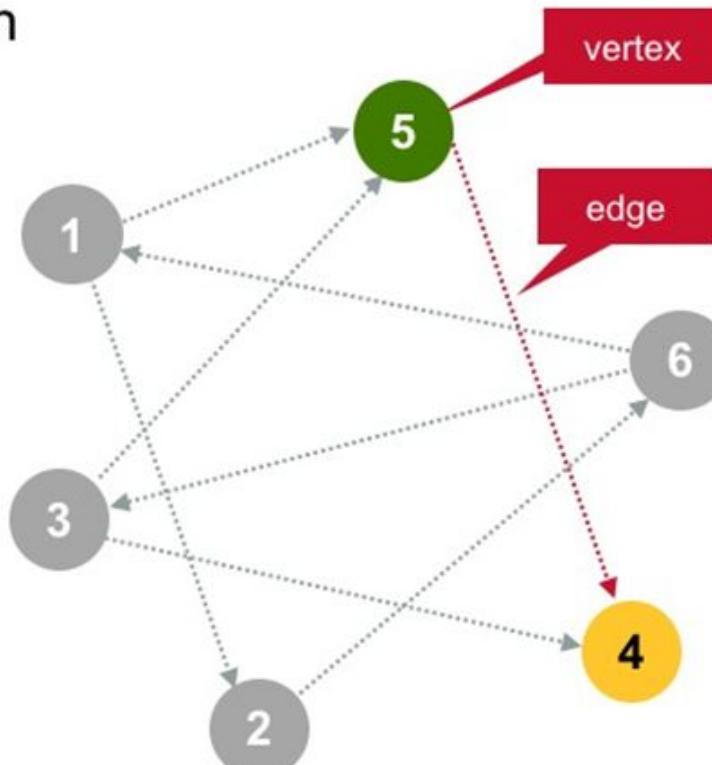
# Regular Graphs: Example

Edges run in both directions



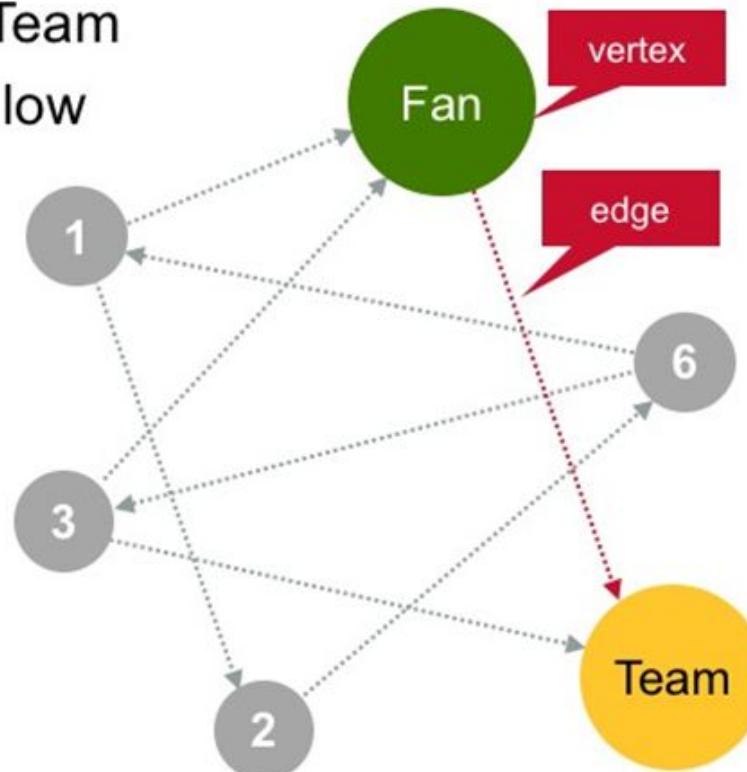
# Directed Graphs

Edges run in one direction



# Directed Graphs: Social Media Example

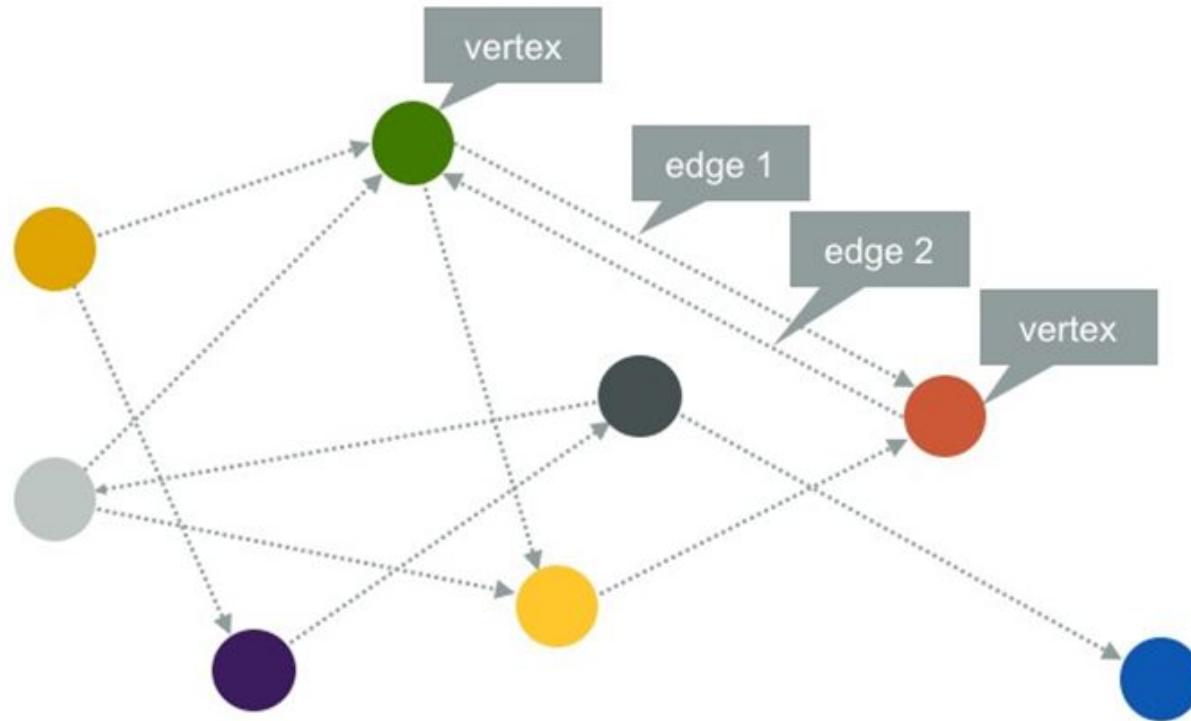
- User Fan follows user Team
- User Team does not follow user Fan



# Property Graphs

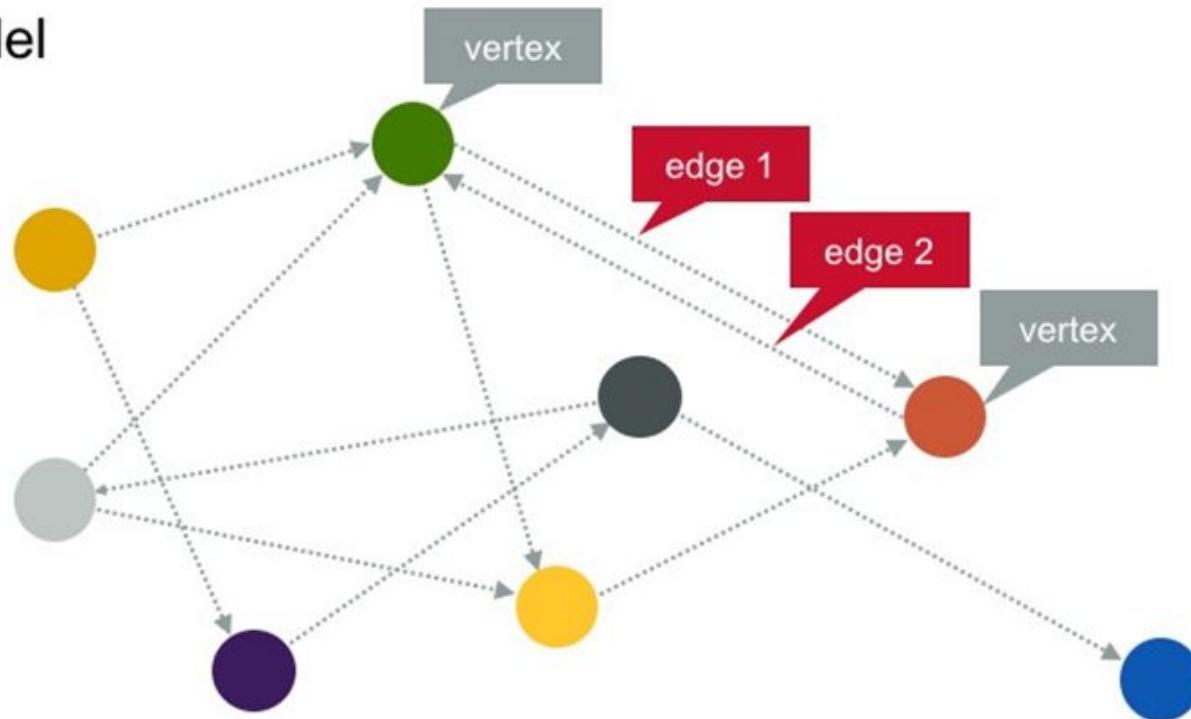
Primary abstraction of Spark GraphFrame

- Immutable
- Distributed
- Fault-tolerant



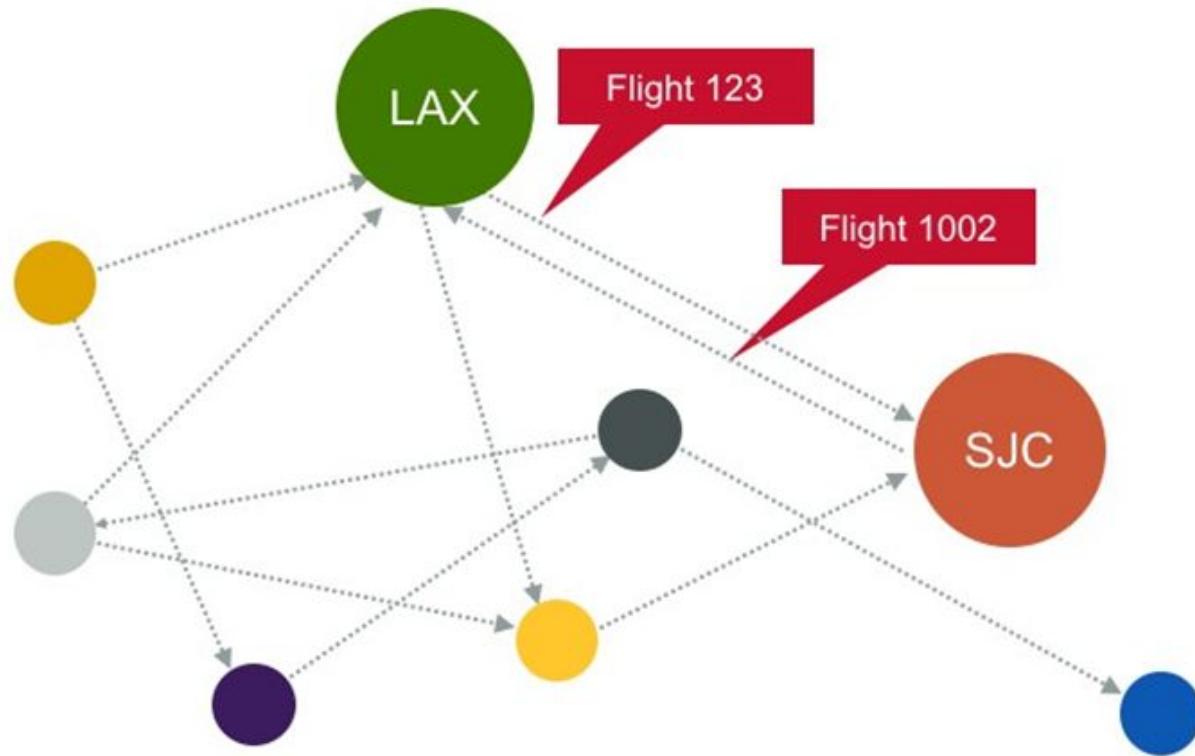
# Property Graphs

- 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"))
```

<b>id</b>	<b>name (Property)</b>
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

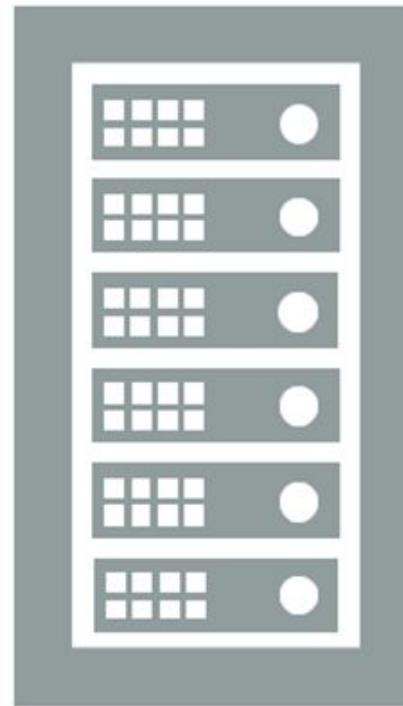


# Learning Goals

- 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

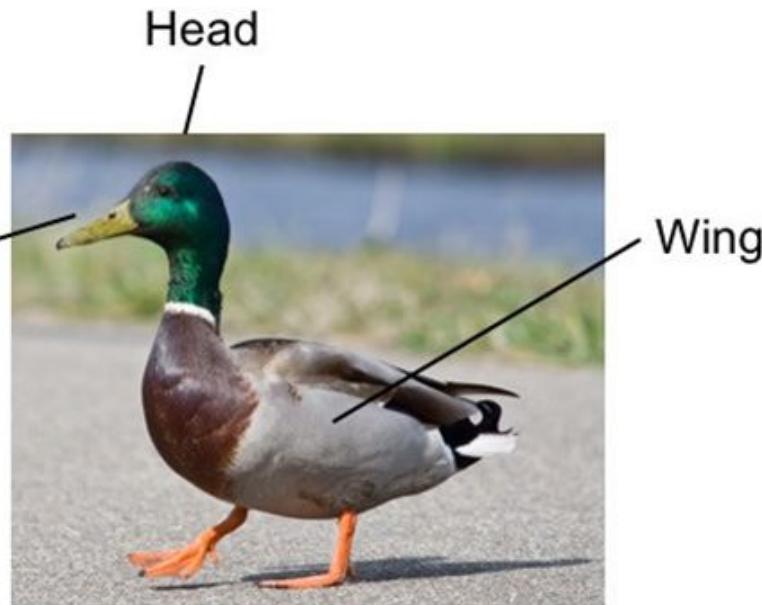
## Unsupervised

- Clustering
- Dimensionality Reduction

# Examples of ML Algorithms: Supervised

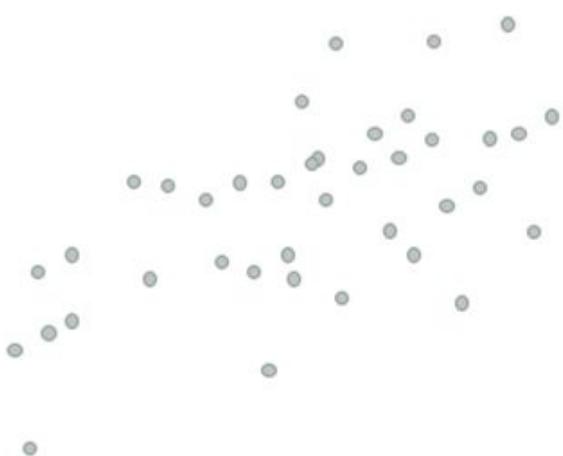
## Supervised

- 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



# Classification: Example

Classification

Identifies category for item

The conv

Google in:spam

Gmail ▾

COMPOSE

Inbox (2,960)

Important

Sent Mail

Drafts (21)

Circles

[Gmail]Drafts

judithouedrago HELP ME DONATE THIS

z.loftus (no subject) - DO YOU

Timothy Diehl, Board Pre. Leadership change at E

David Foster Standards all of us sh

Sofia Kipkalya Dearest One, - Dearest

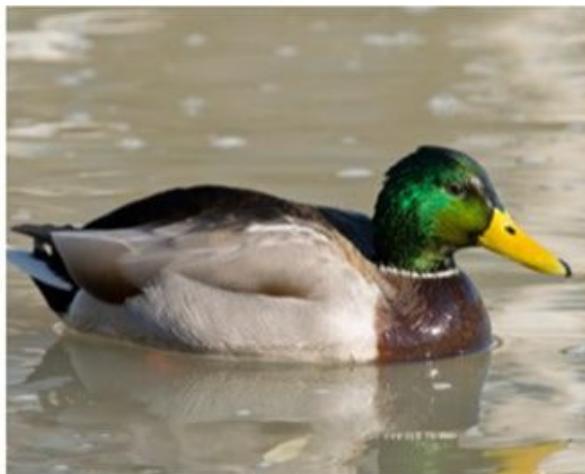
# If it Walks/Swims/Quacks Like a Duck... Then It Must Be a Duck

## Features of ML:

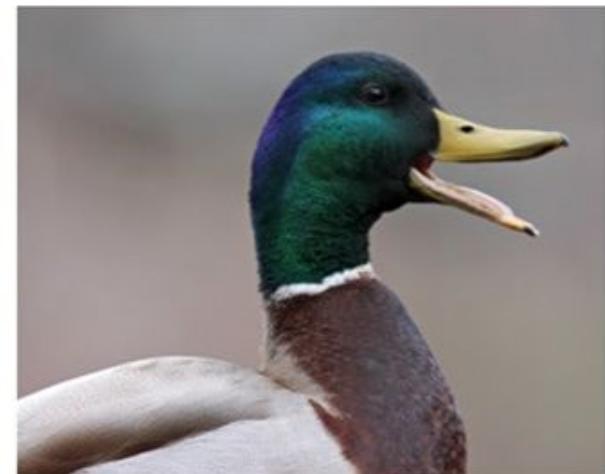
- Classify something based on pre-determined features
- Features are the “if questions” that you ask



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!

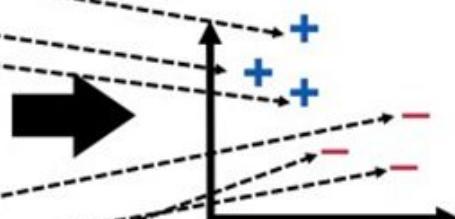
free savings \$\$\$

**Non-spam:**

how are you?

that Spark job

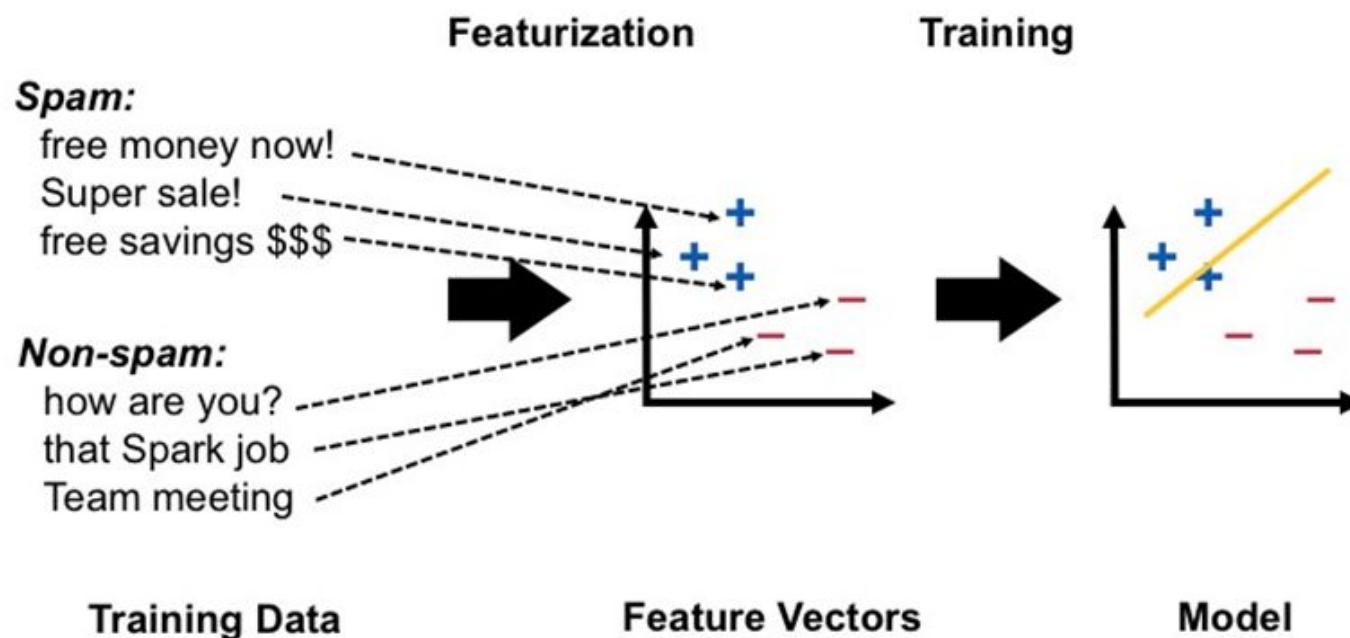
Team meeting



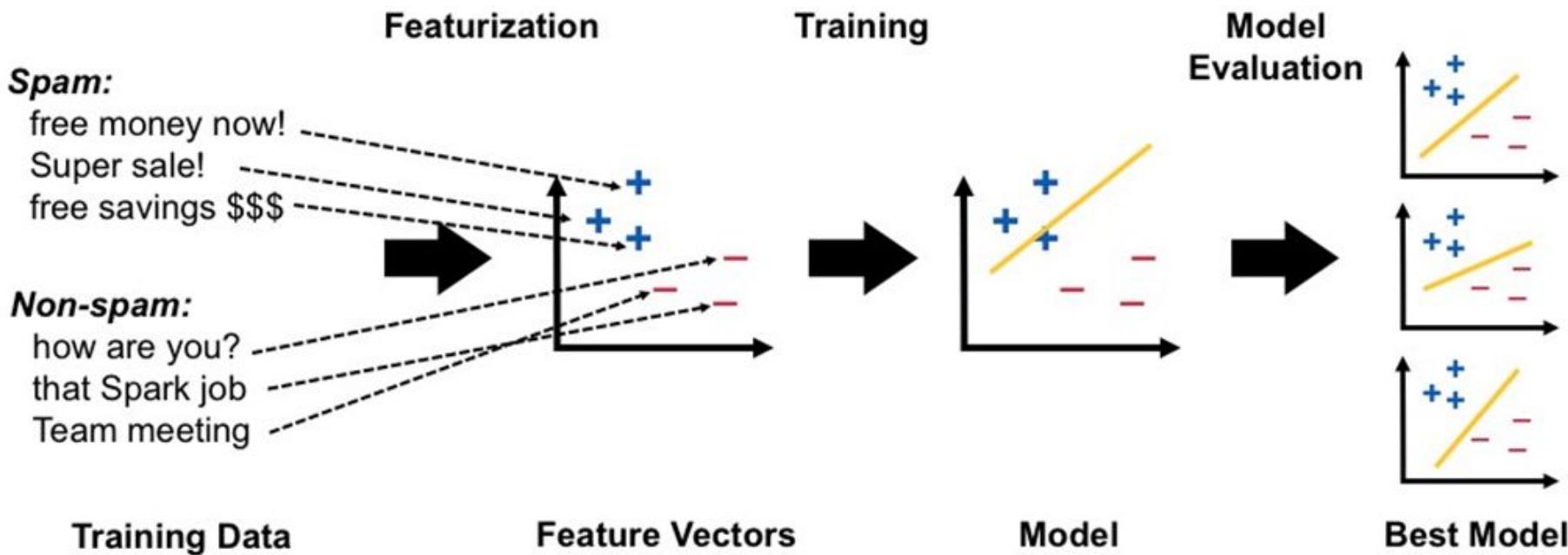
Training Data

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

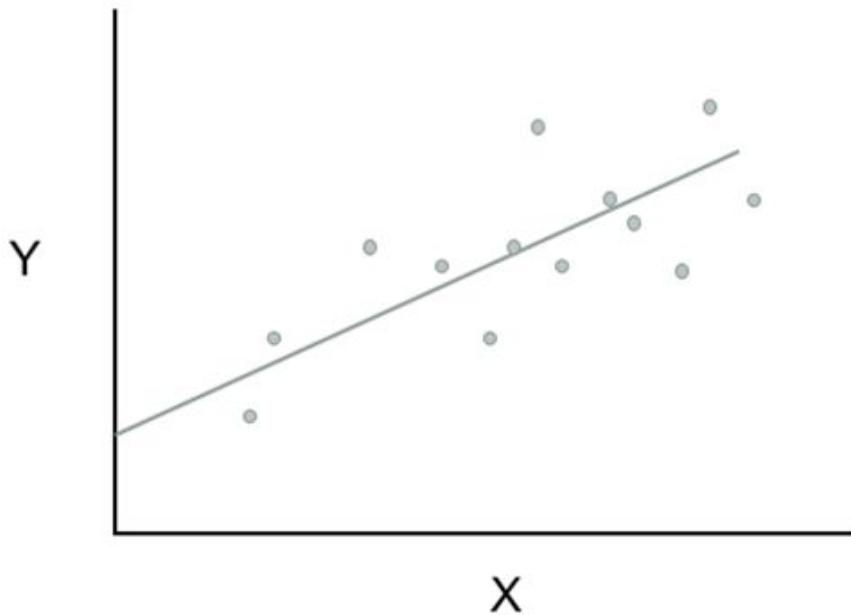
# Knowledge Check



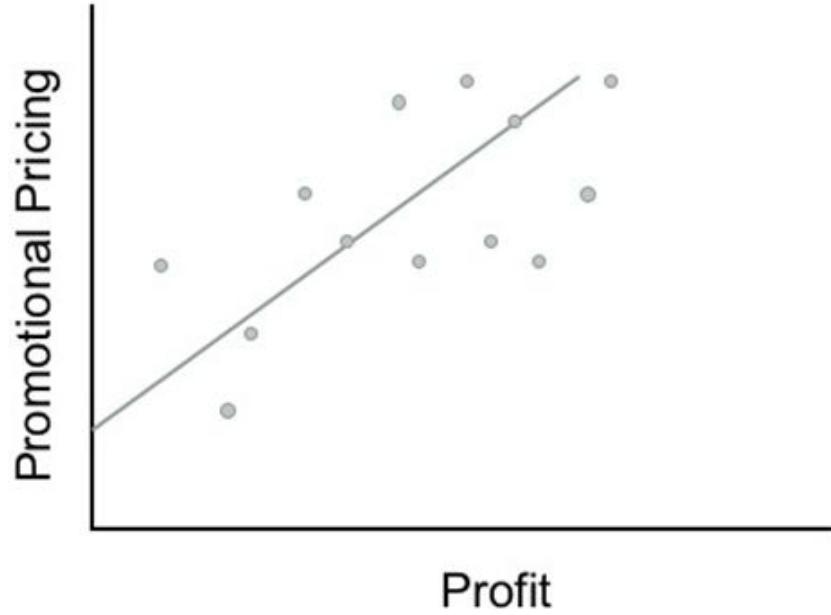
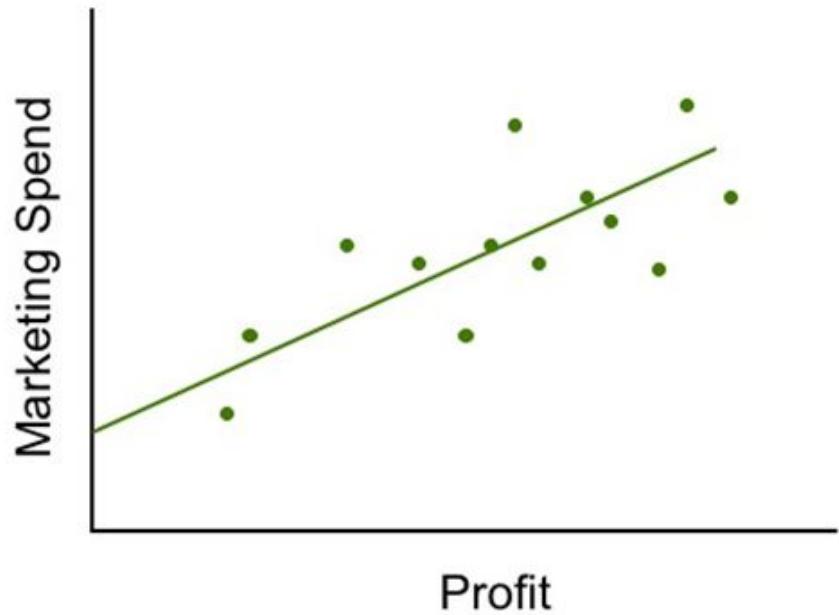
Put the steps for creating and training a Classification model into the correct order:

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

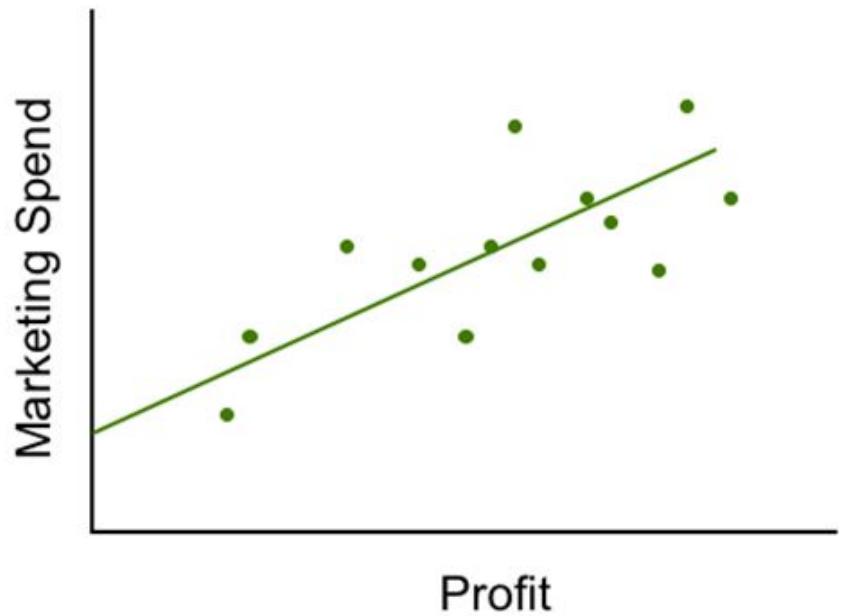
# Regression Analysis



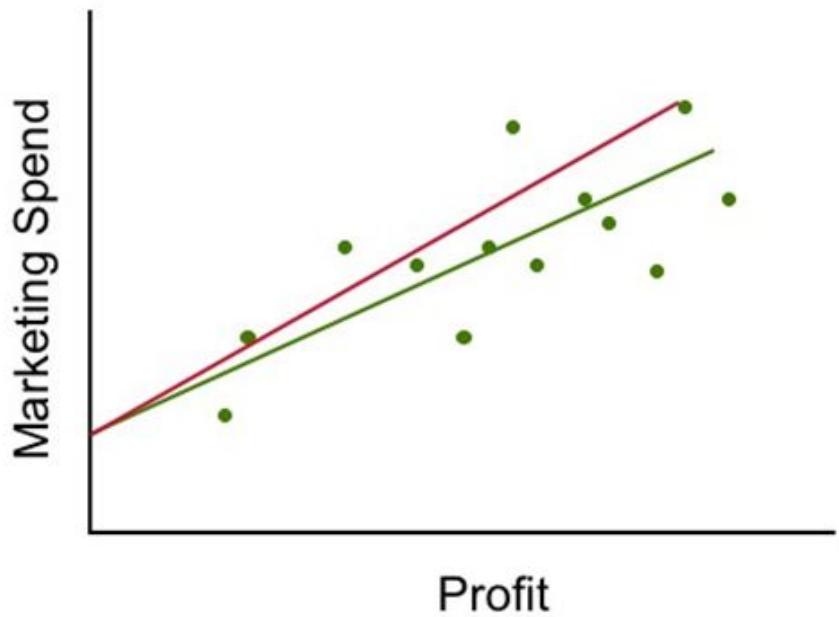
# Regression Analysis



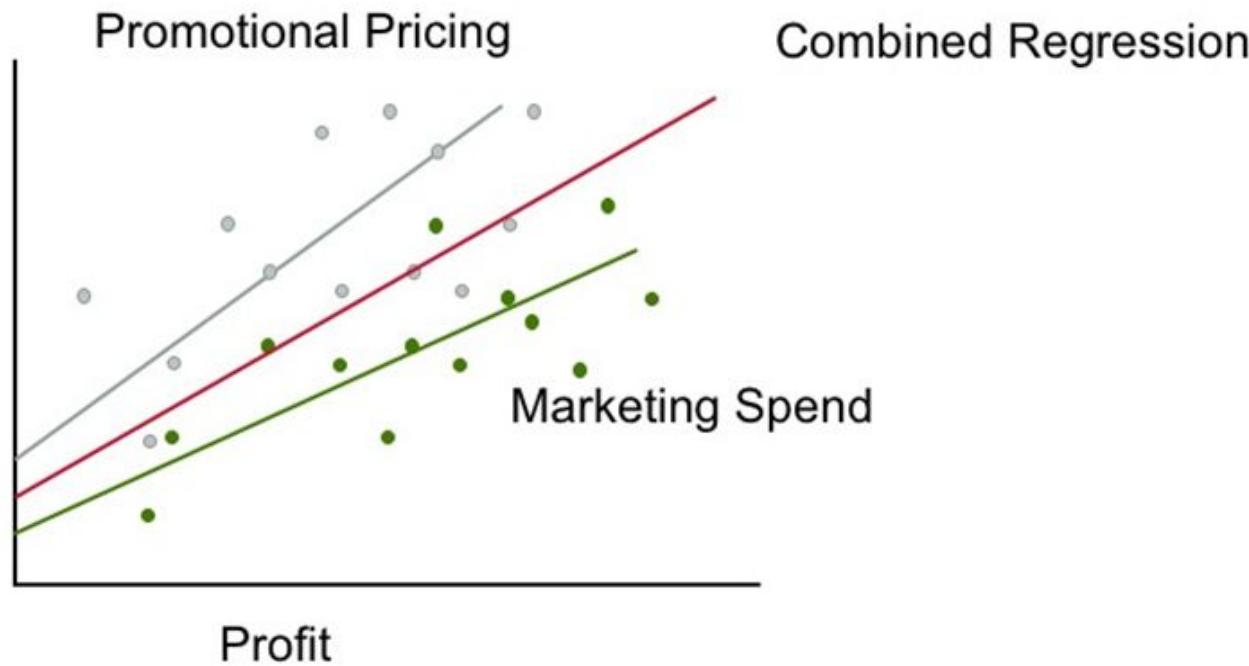
# Regression Analysis



# Regression Analysis

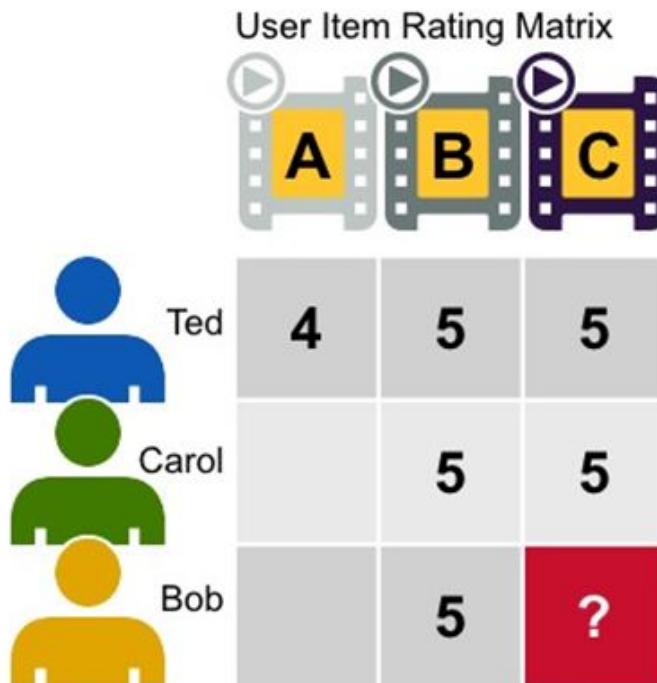


# Regression Analysis



# Machine Learning: Collaborative Filtering

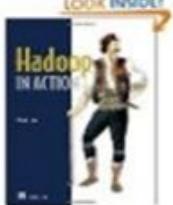
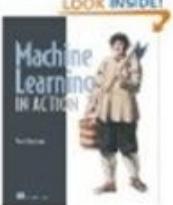
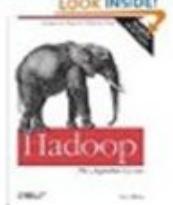
- Based on user preferences data
  - Collaborative
- Recommend items
  - Filtering



# Machine Learning: Collaborative Filtering

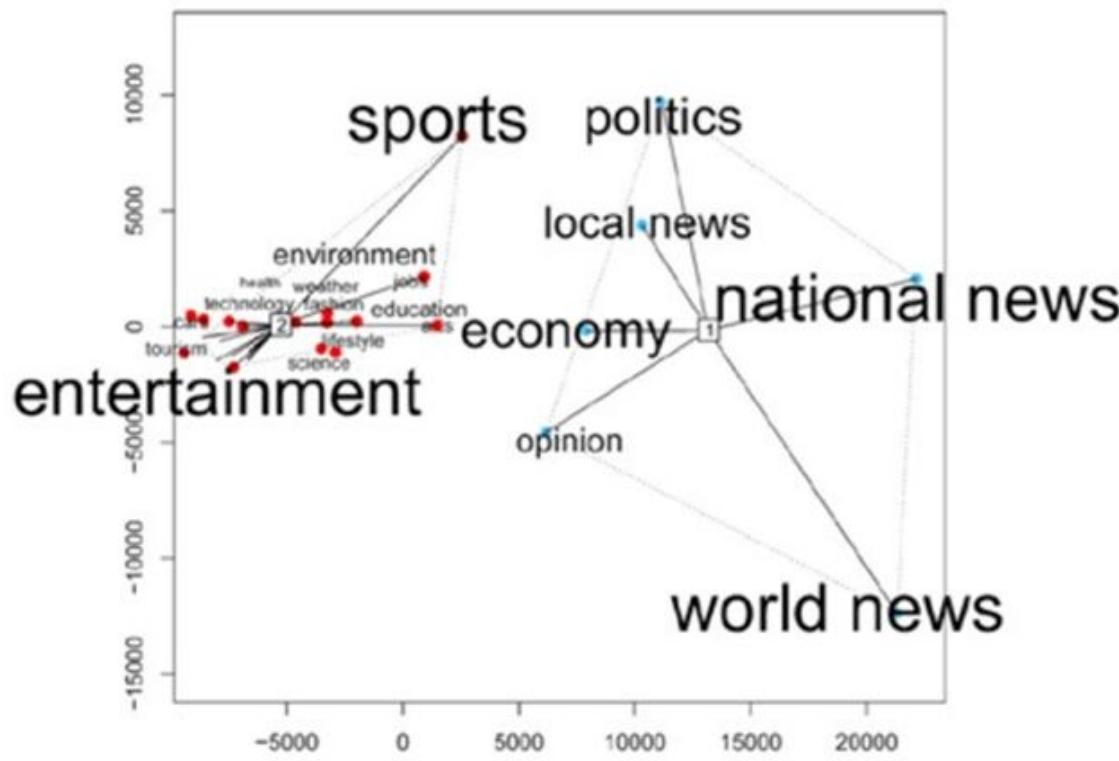
Collaborative Filtering  
(Recommendation)

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

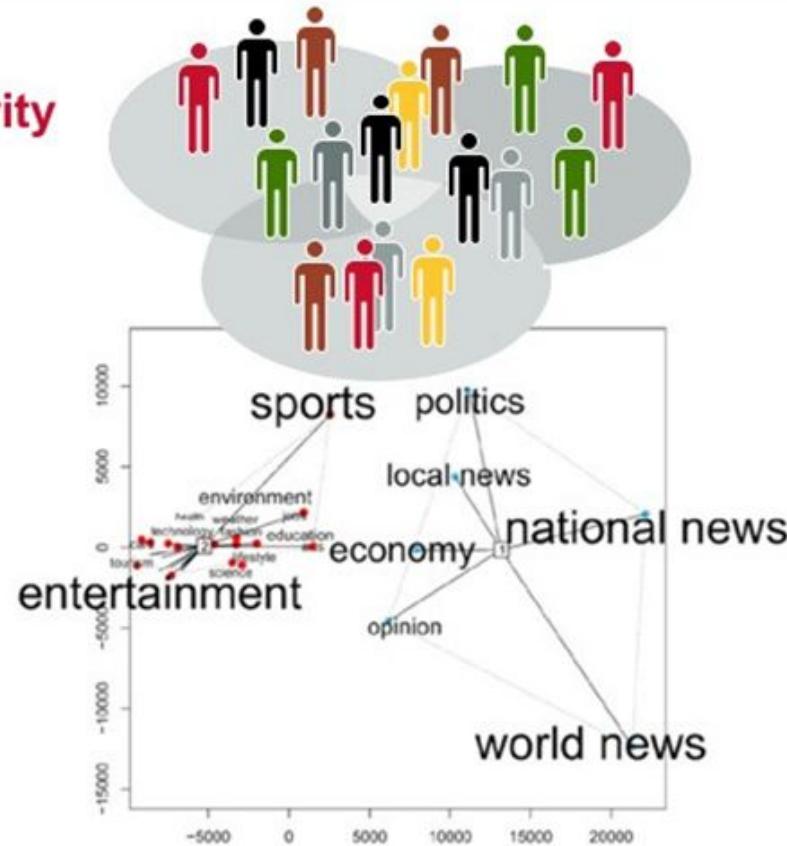
- Classifies inputs into categories by analyzing similarities between input examples



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

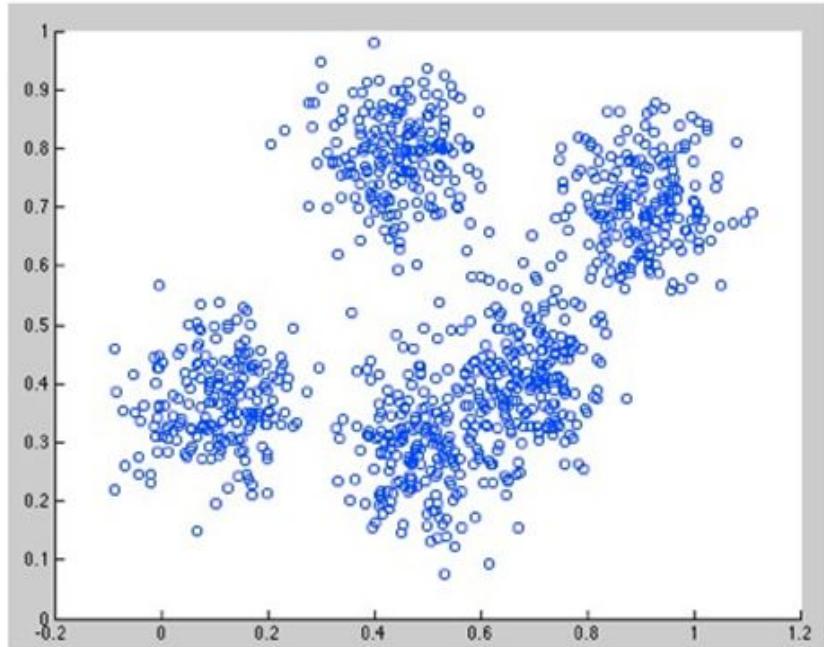
[See realtime coverage »](#)

### Vaccine protects against malaria in early test



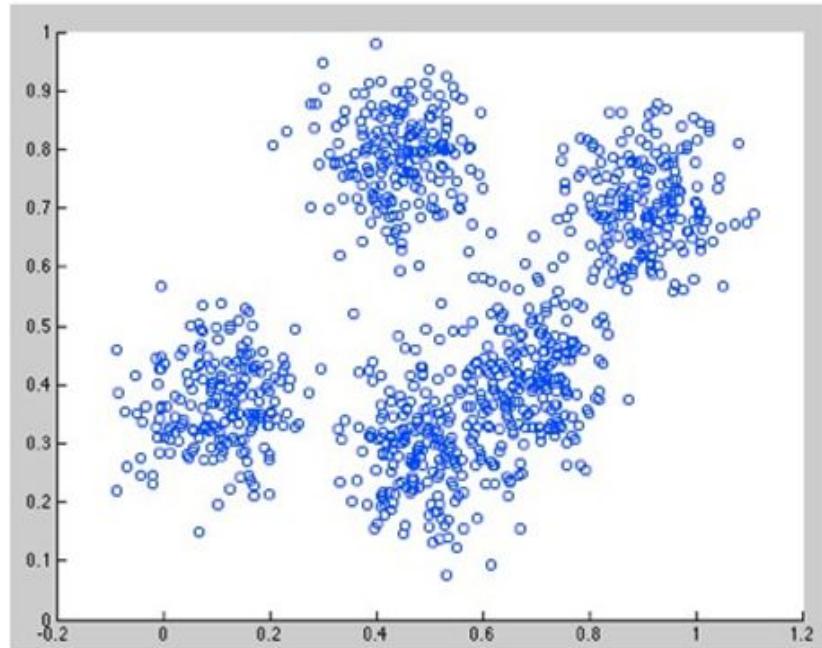
# Clustering: Example

- Group similar objects



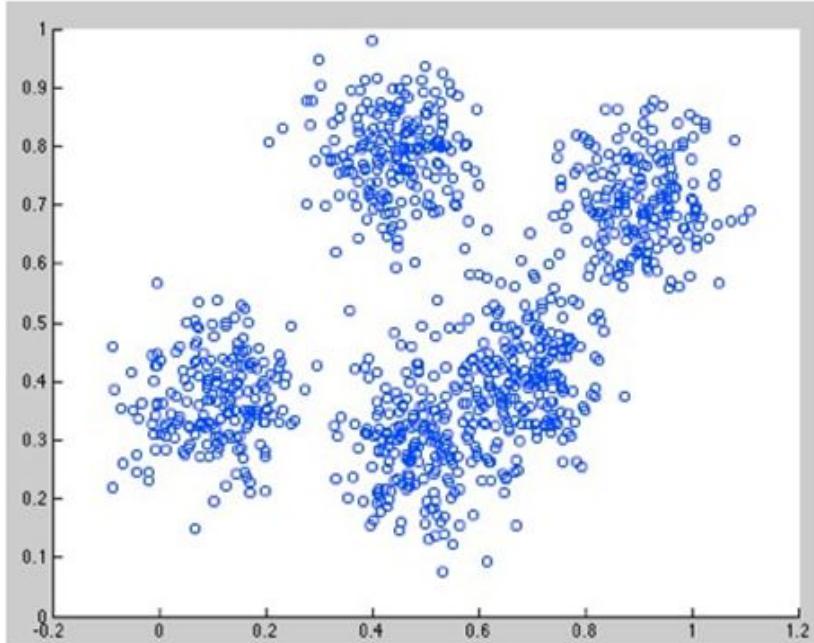
# Clustering: Example

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



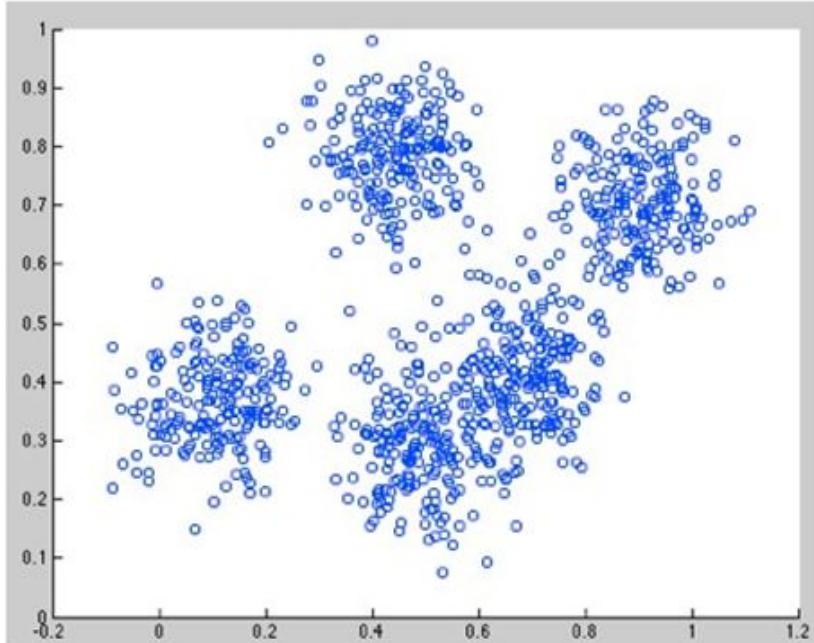
# Clustering: Example

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



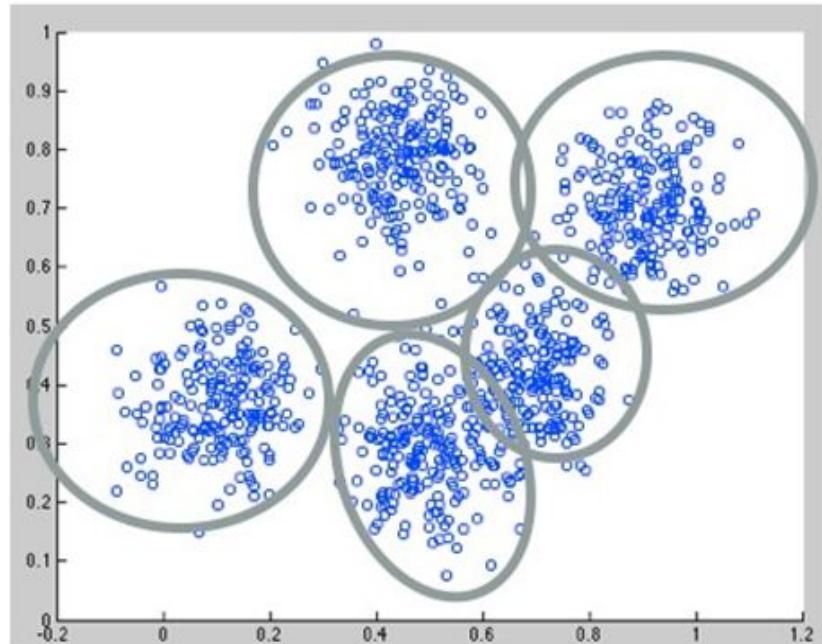
# Clustering: Example

- 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

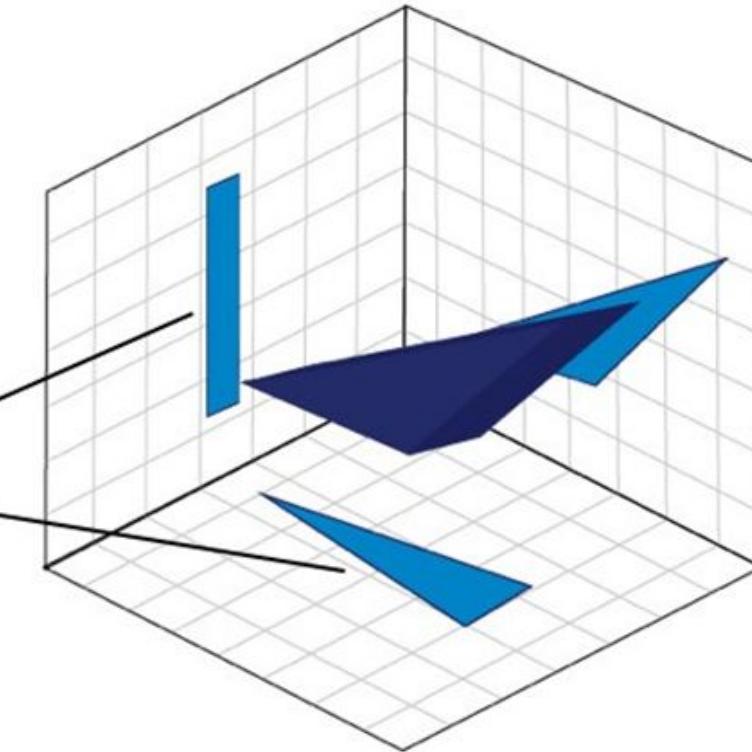
- Group similar objects
  - Use MLlib K-means algorithm
1. Initialize coordinates to center of clusters (centroid)
  2. Assign all points to nearest centroid
  3. Update centroids to center of points
  4. Repeat until conditions met



# Dimensionality Reduction

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

Reduced Two-Dimensional Views

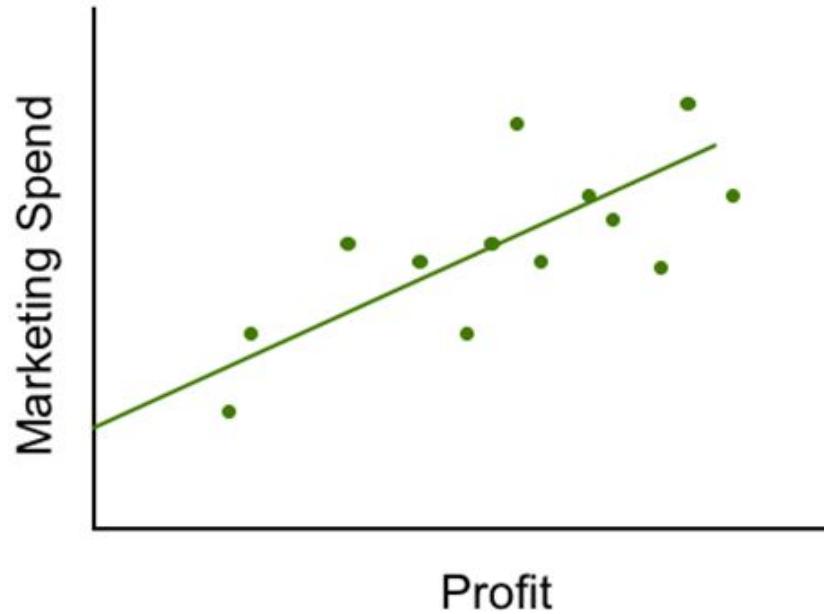


Three-Dimensional Data Set

# Dimensionality Reduction: Feature Selection



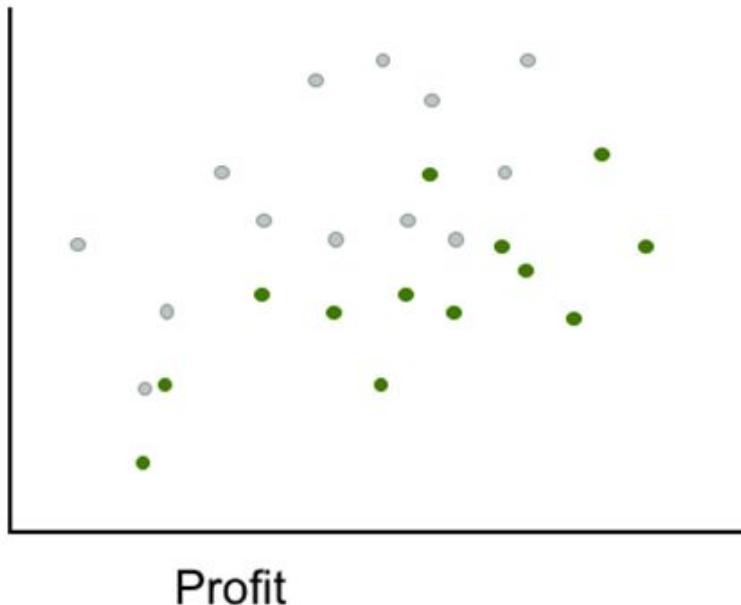
# Dimensionality Reduction: Feature Selection



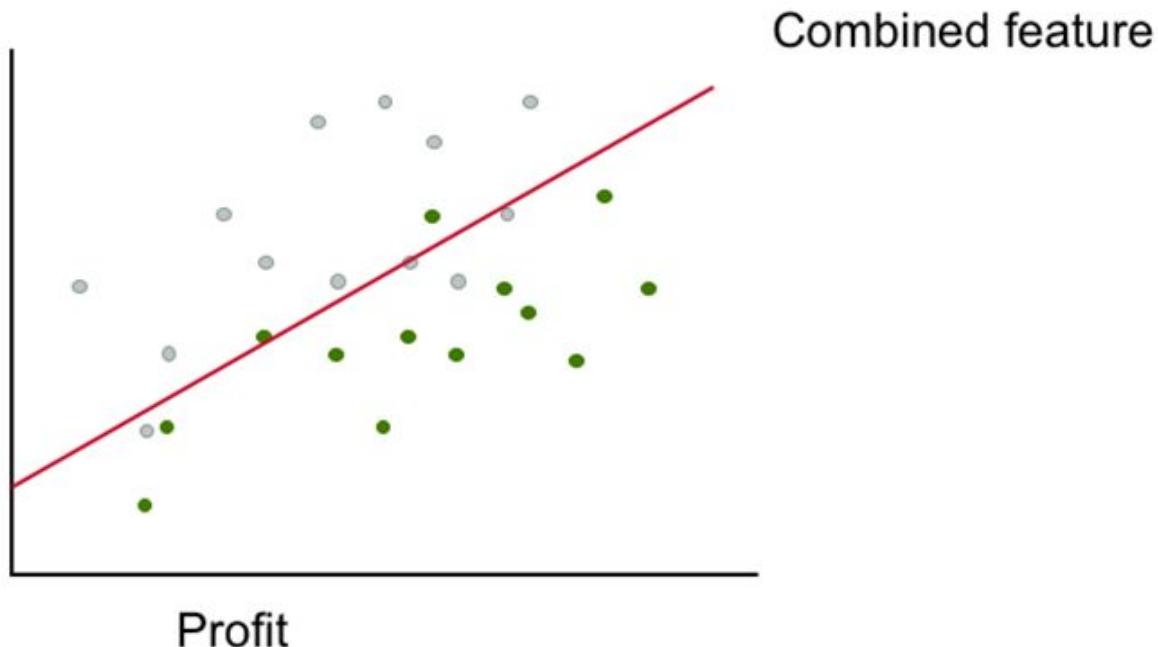
# Dimensionality Reduction: Feature Selection

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



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

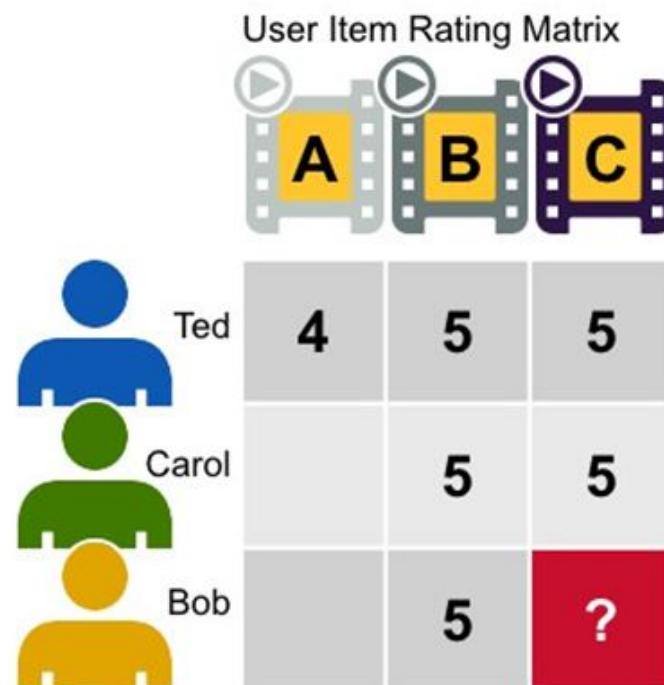
Ted and Carol like movies B and C



Bob likes movie B, what might he like?



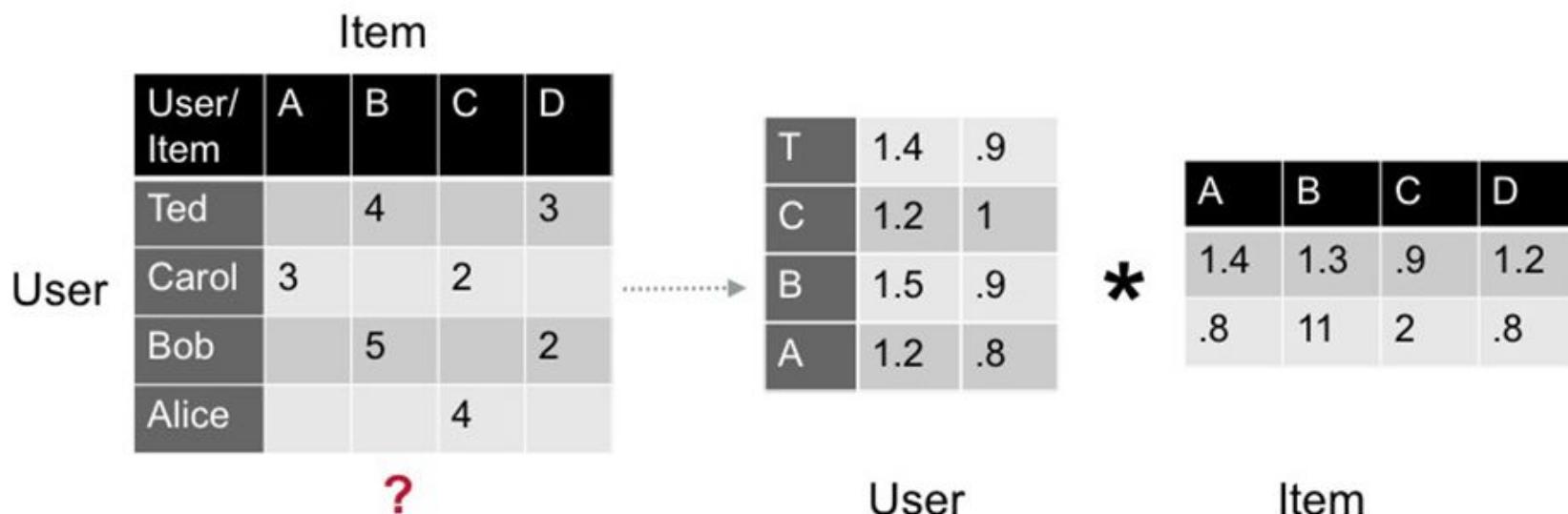
Bob likes movie B, **predict C**



# Alternating Least Squares

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

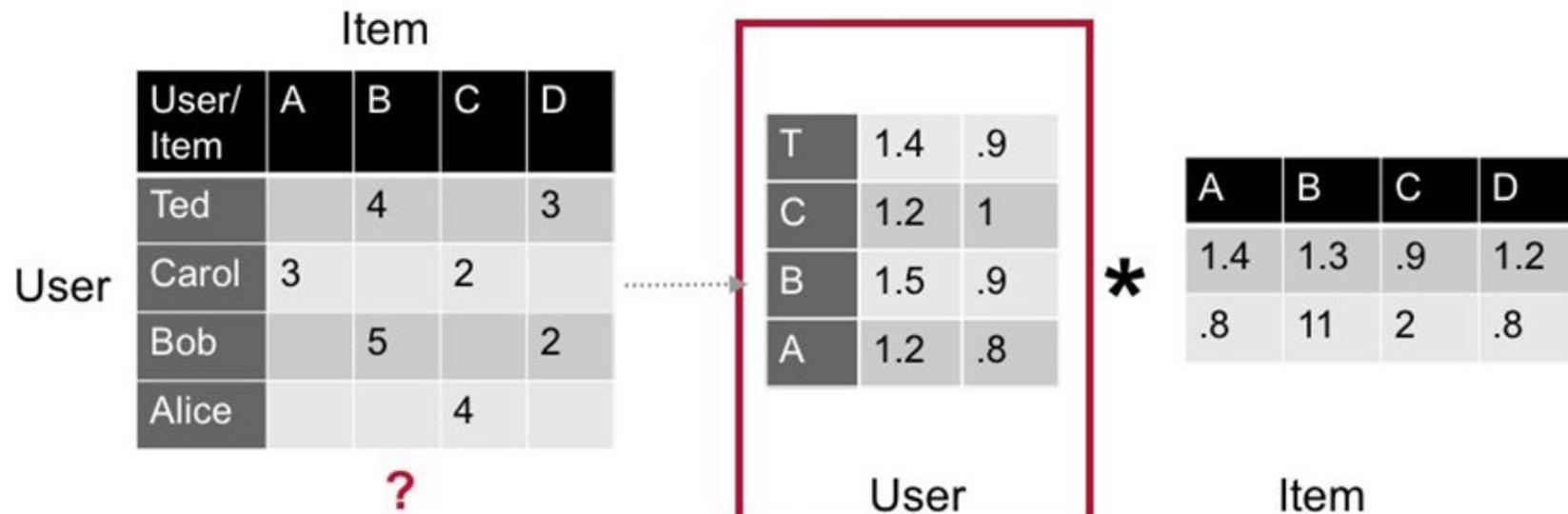
- User and Item factor matrices



# Alternating Least Squares

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

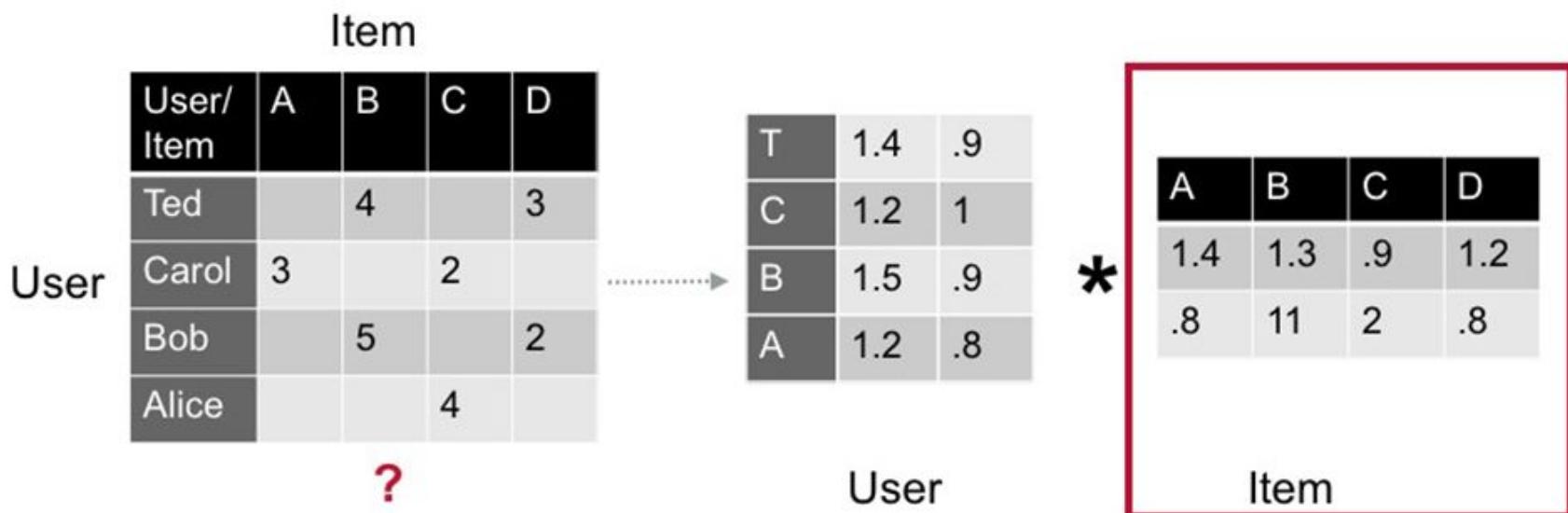
- User and Item factor matrices
- Tries to learn the hidden features of each user and item



# Alternating Least Squares

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

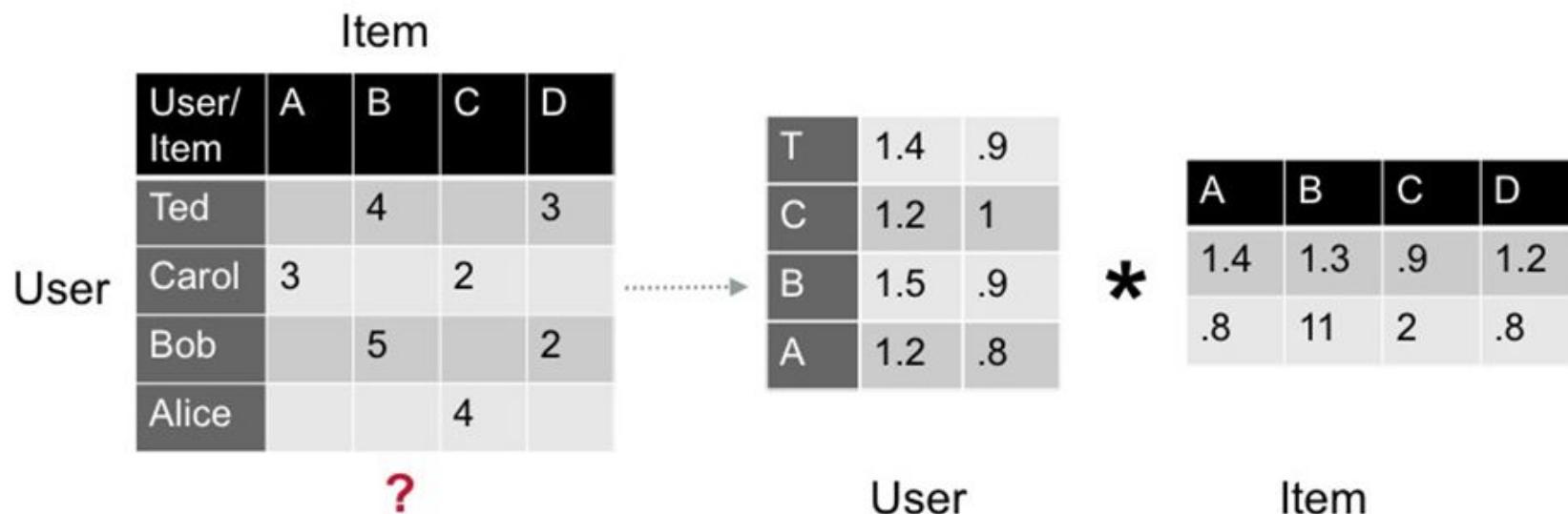
- User and Item factor matrices
- Tries to learn the hidden features of each user and item



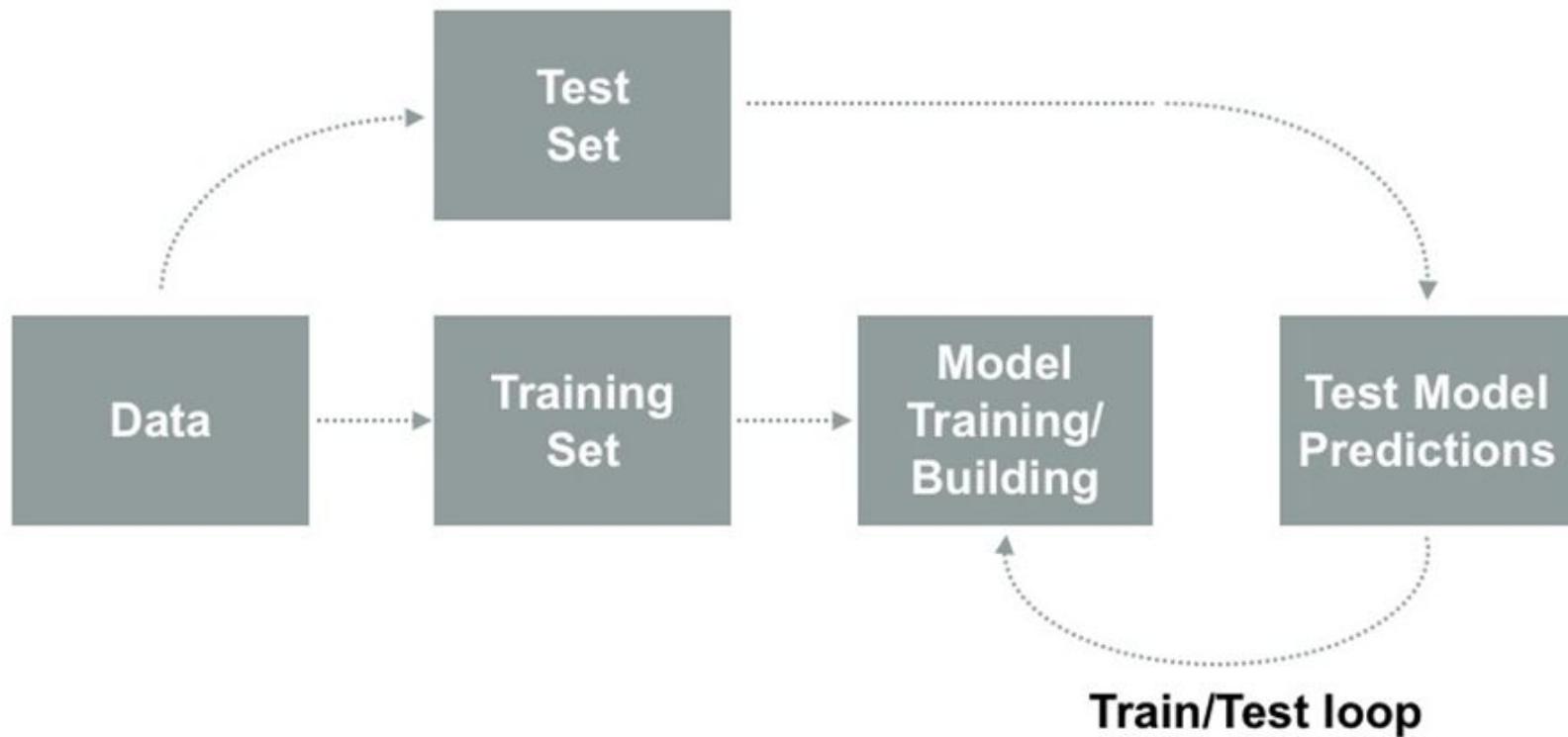
# Alternating Least Squares

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

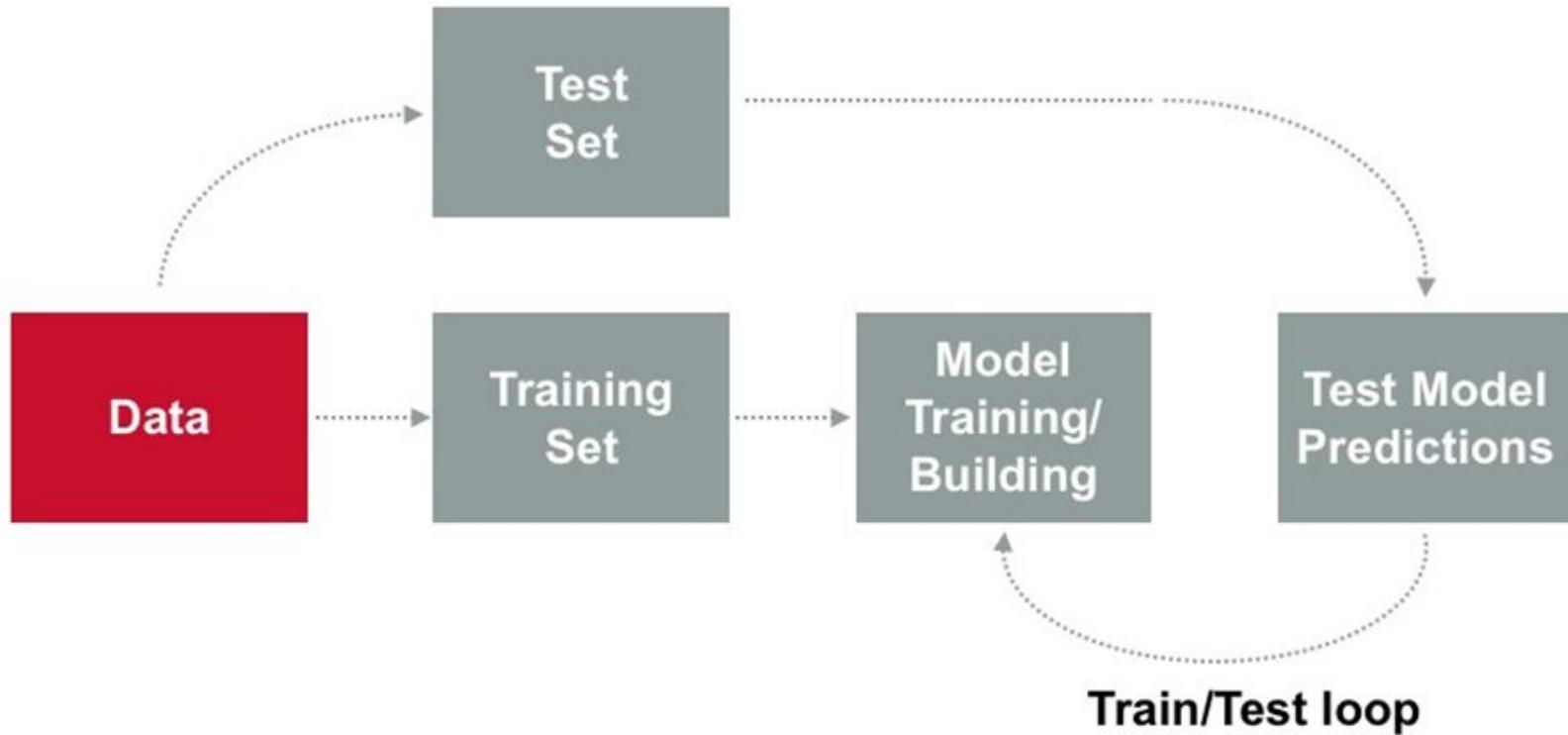
- User and Item factor matrices
- Tries to learn the hidden features of each user and item
- Algorithm alternatively fixes one factor matrix and solves for the other



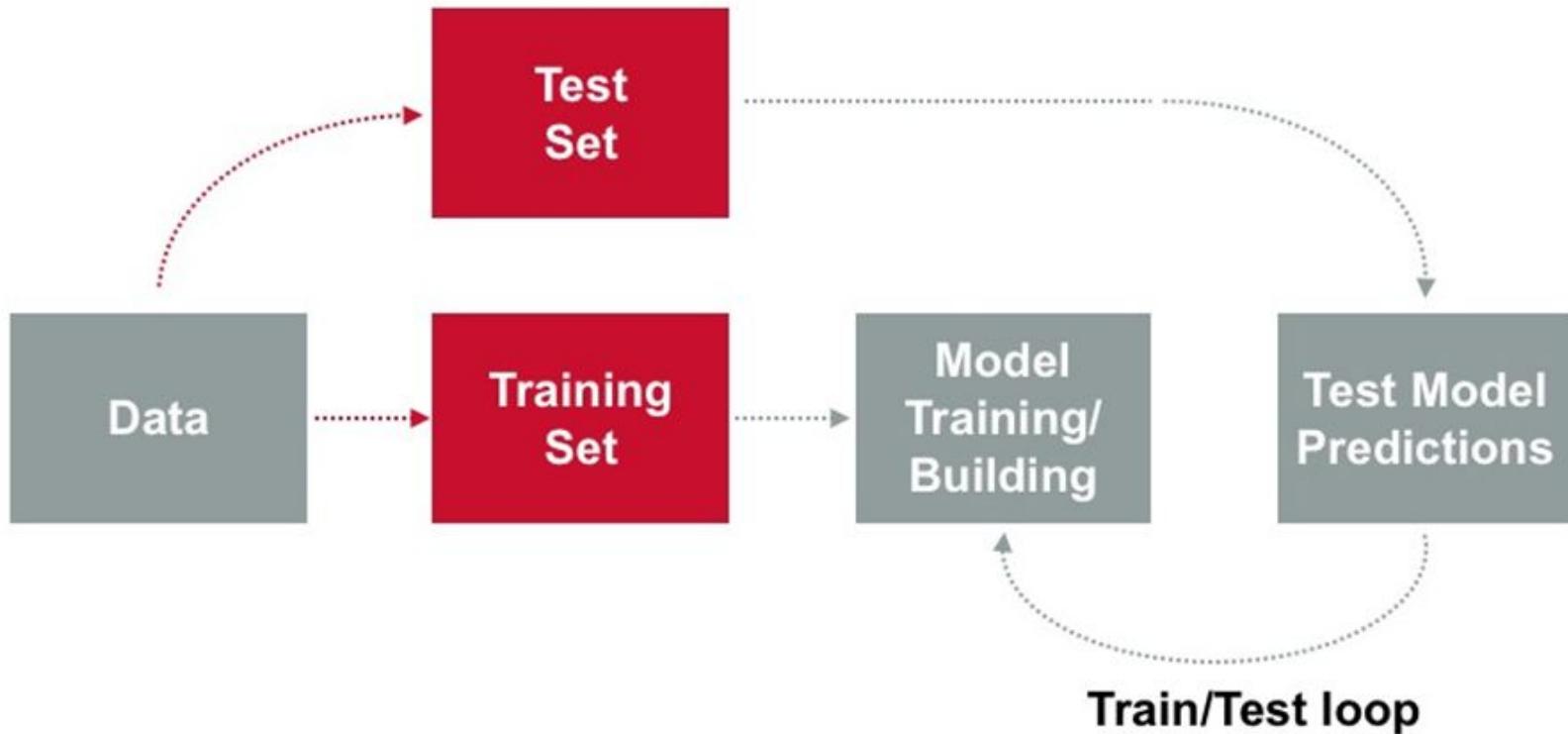
# ML Cross-Validation Process



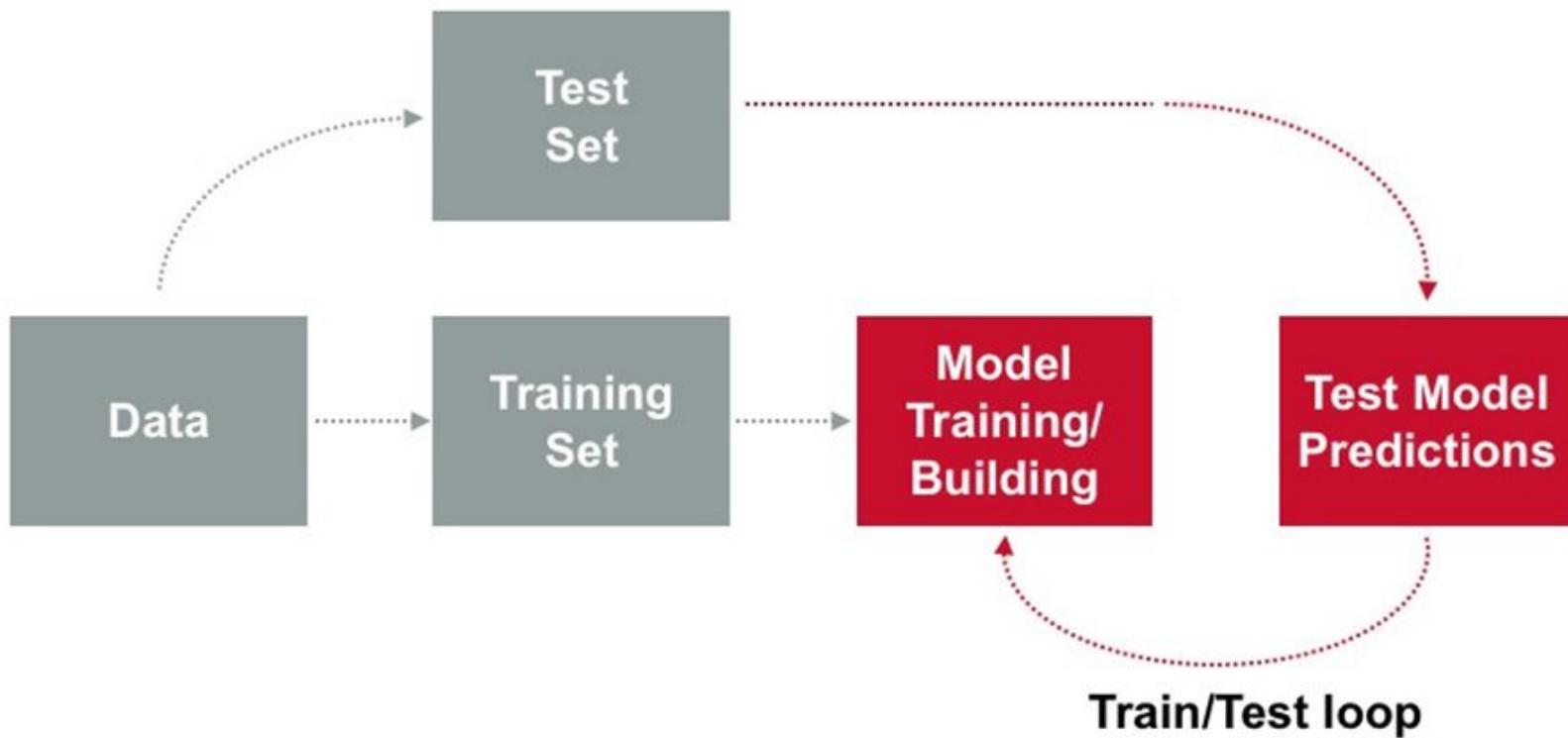
# ML Cross-Validation Process



# 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)
}
```

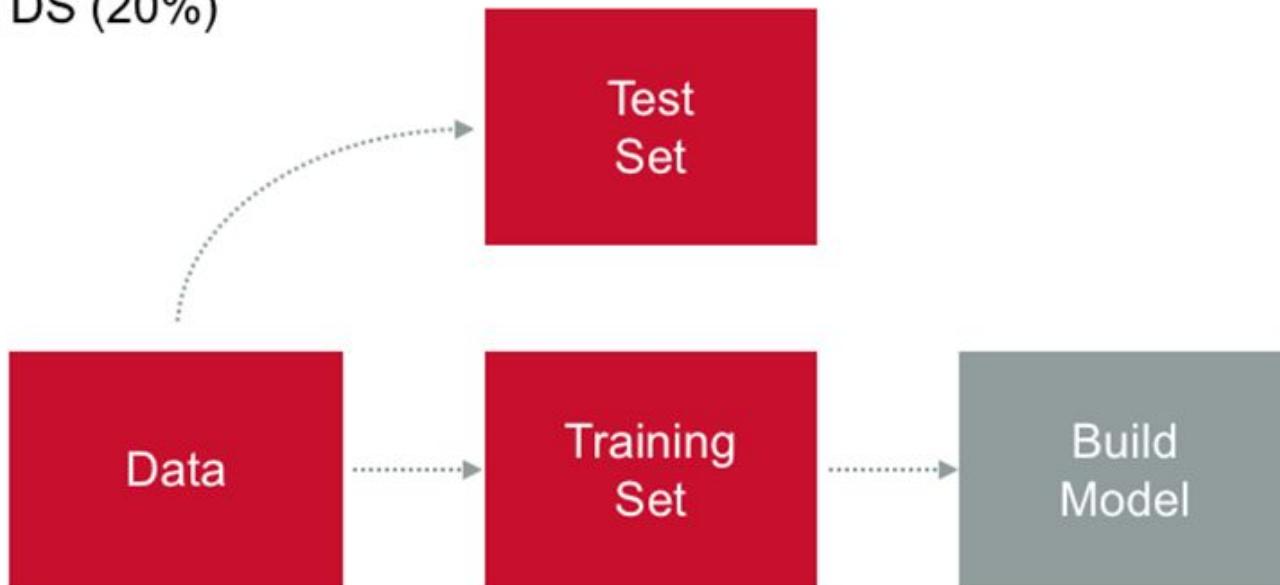
# Parse Input

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

# Build Model

Split ratingsDS into:

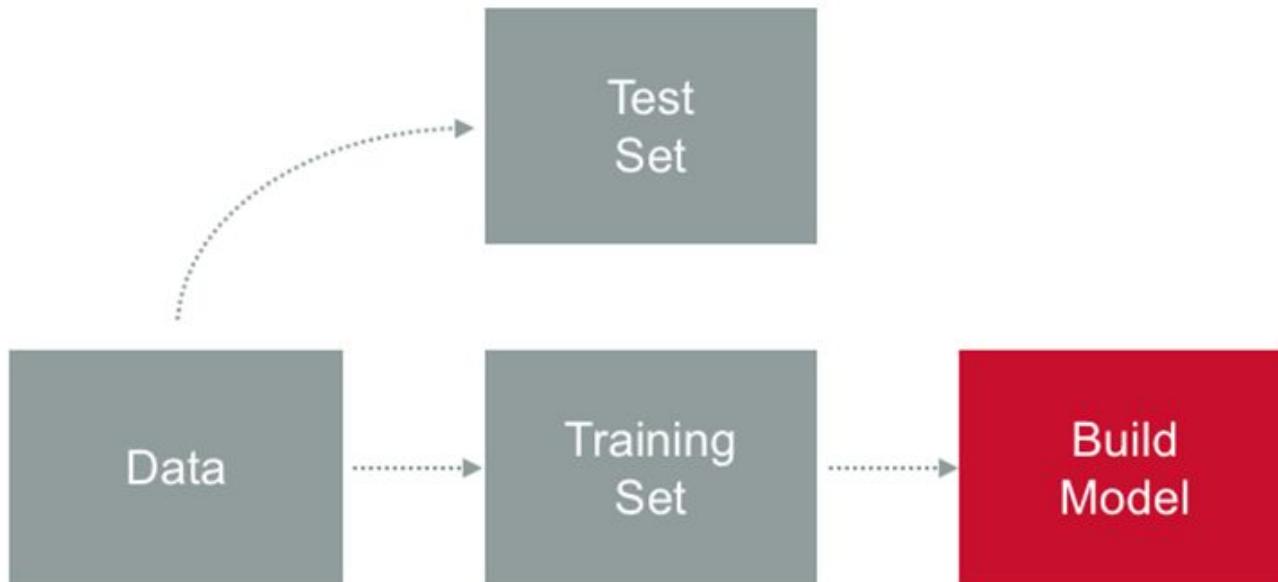
- Training data DS (80%)
- Test data DS (20%)



## 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



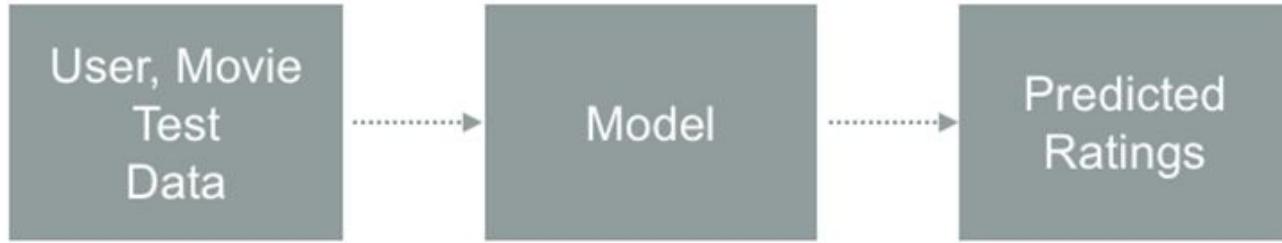
Build a user product matrix model

# 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

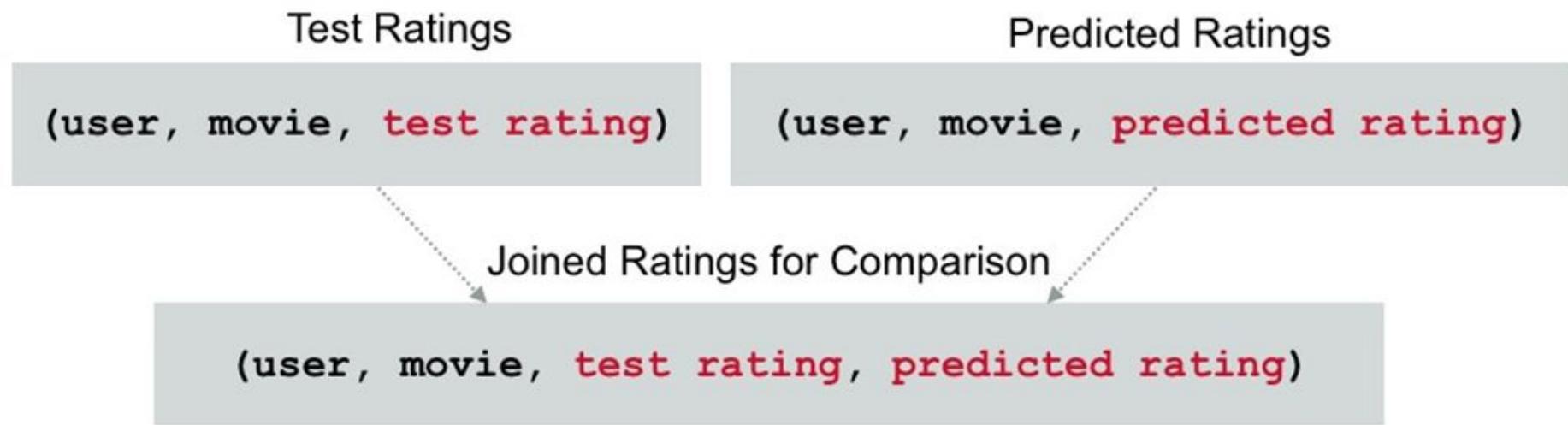
---



## 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.