



# Universidad **Ricardo Palma**

RECTORADO  
PROGRAMA DE ESPECIALIZACIÓN EN CIENCIA DE DATOS

*Formamos seres humanos para una cultura de paz*

## BIG DATA APLICADO


SESIÓN 04

**Expositores:**

**David Narváez**

**Eder Pineda**

**[bigdataplicado@gmail.com](mailto:bigdataplicado@gmail.com)**



***Big data** is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation (**Gartner**).*

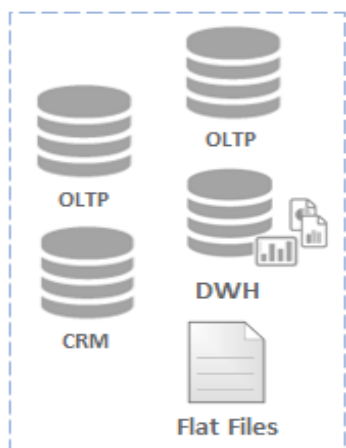






# Arquitectura Lambda

## DATA SOURCES

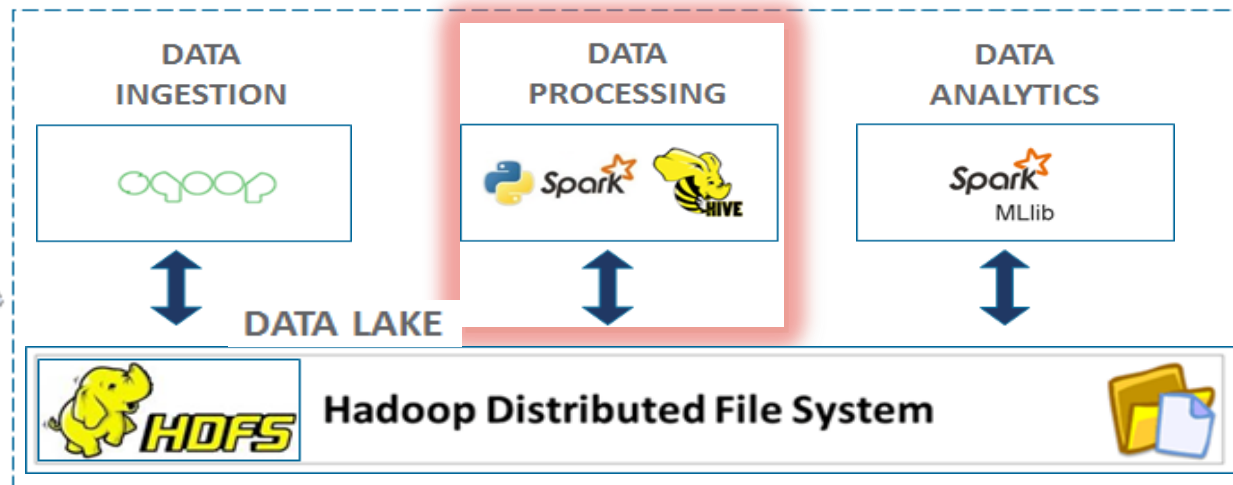


Batch

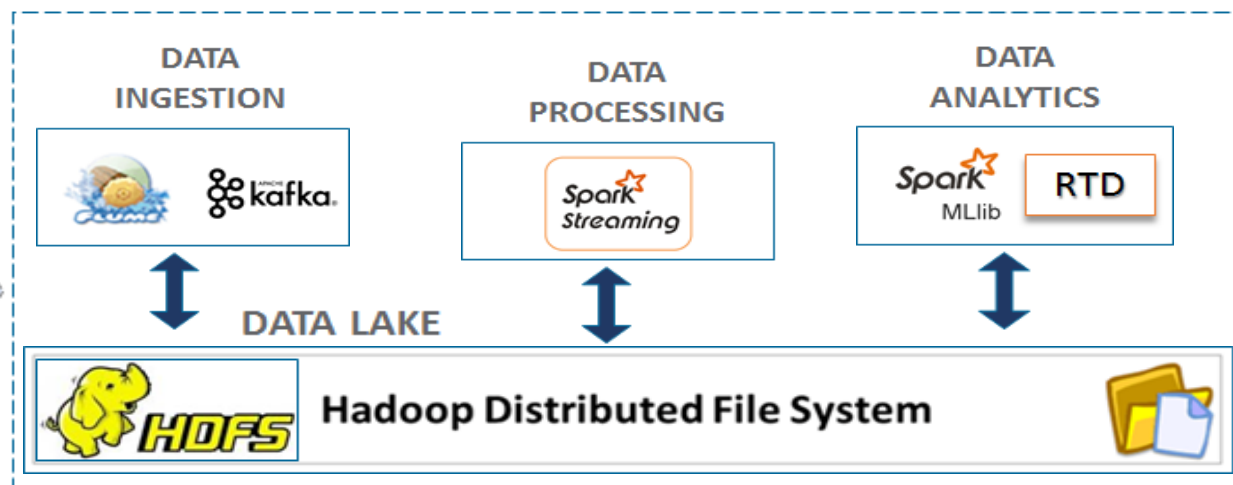
Real Time



## BATCH LAYER



## SPEED LAYER



## SERVING LAYER

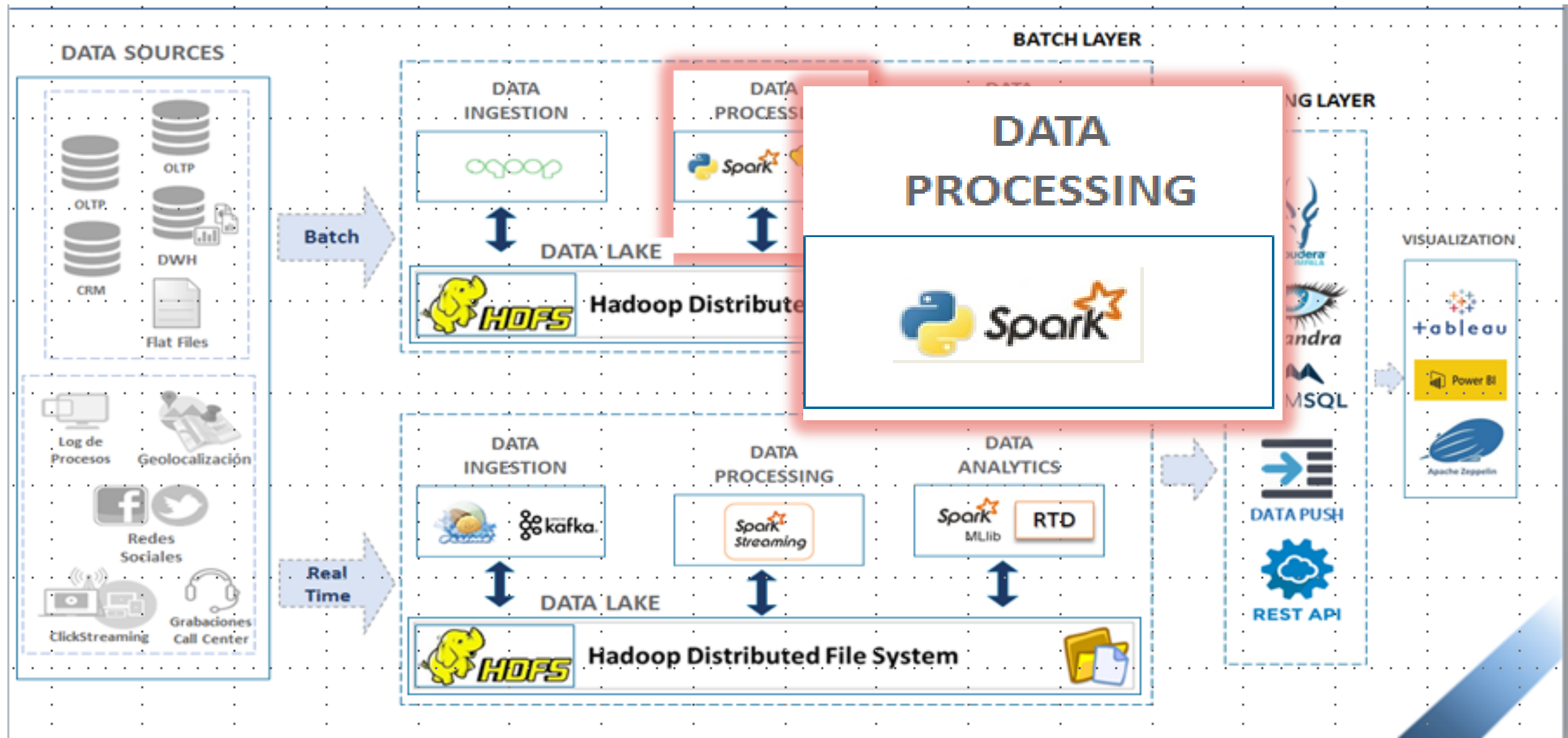


## VISUALIZATION





# Arquitectura Lambda



# AGENDA

## Introducción:

1. Introducción a Spark
2. RDD, Dataframe y Datasets
3. Formas de computación en Spark
4. Manejo de archivos y conexiones
5. Spark SQL
6. Operaciones con dataframes

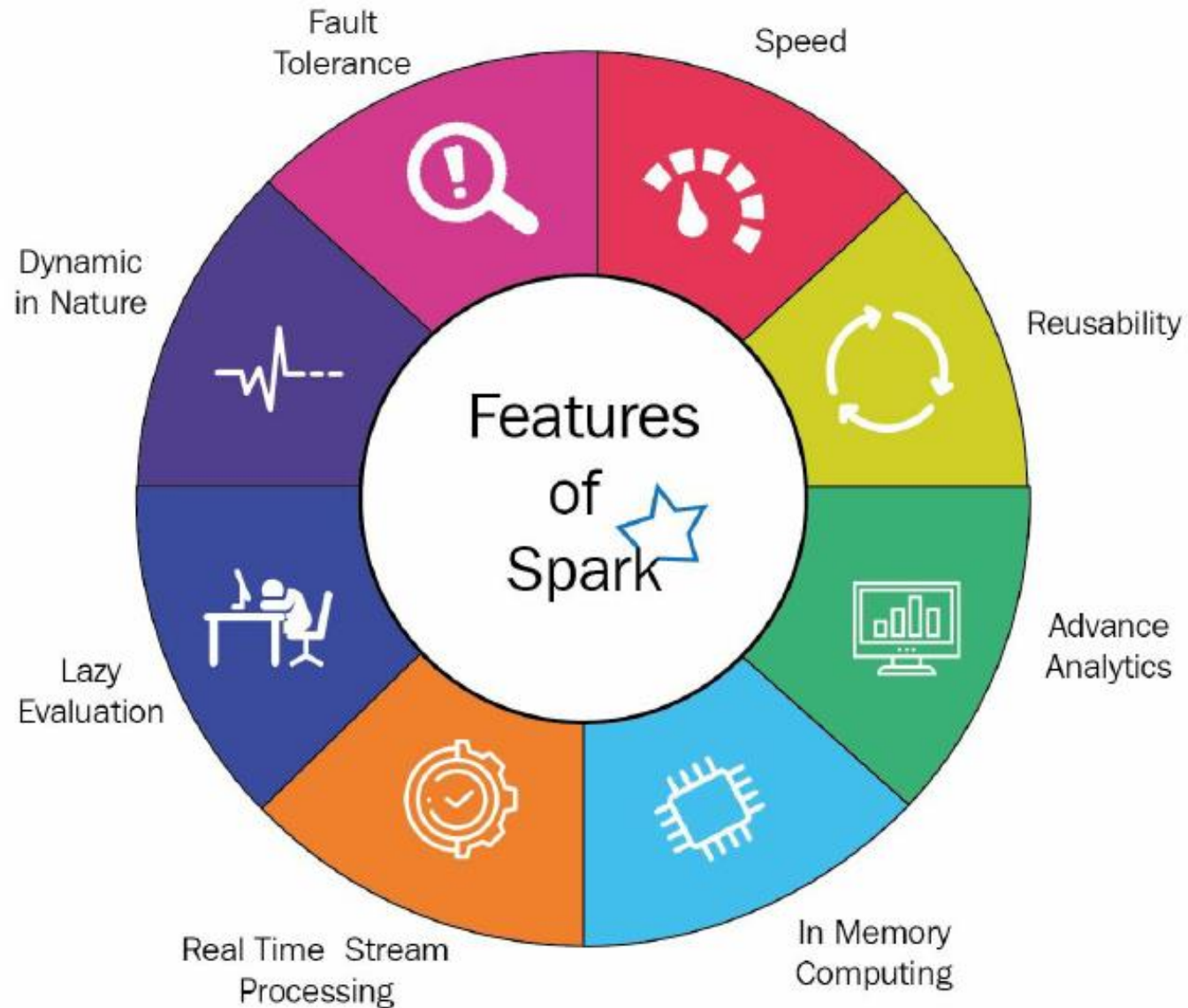


# 1. Introducción a SPARK

---

- Sistema de procesamiento que permite trabajar con grandes volúmenes de forma simple.
- Nace del propósito de orquestar un set de nodos, los cuales trabajen de manera organizada para ayudar en el procesamiento de datos.
- Se ejecuta en Apache YARN, Standalone, o sobre Apache Mesos.

# 1. Introducción a SPARK



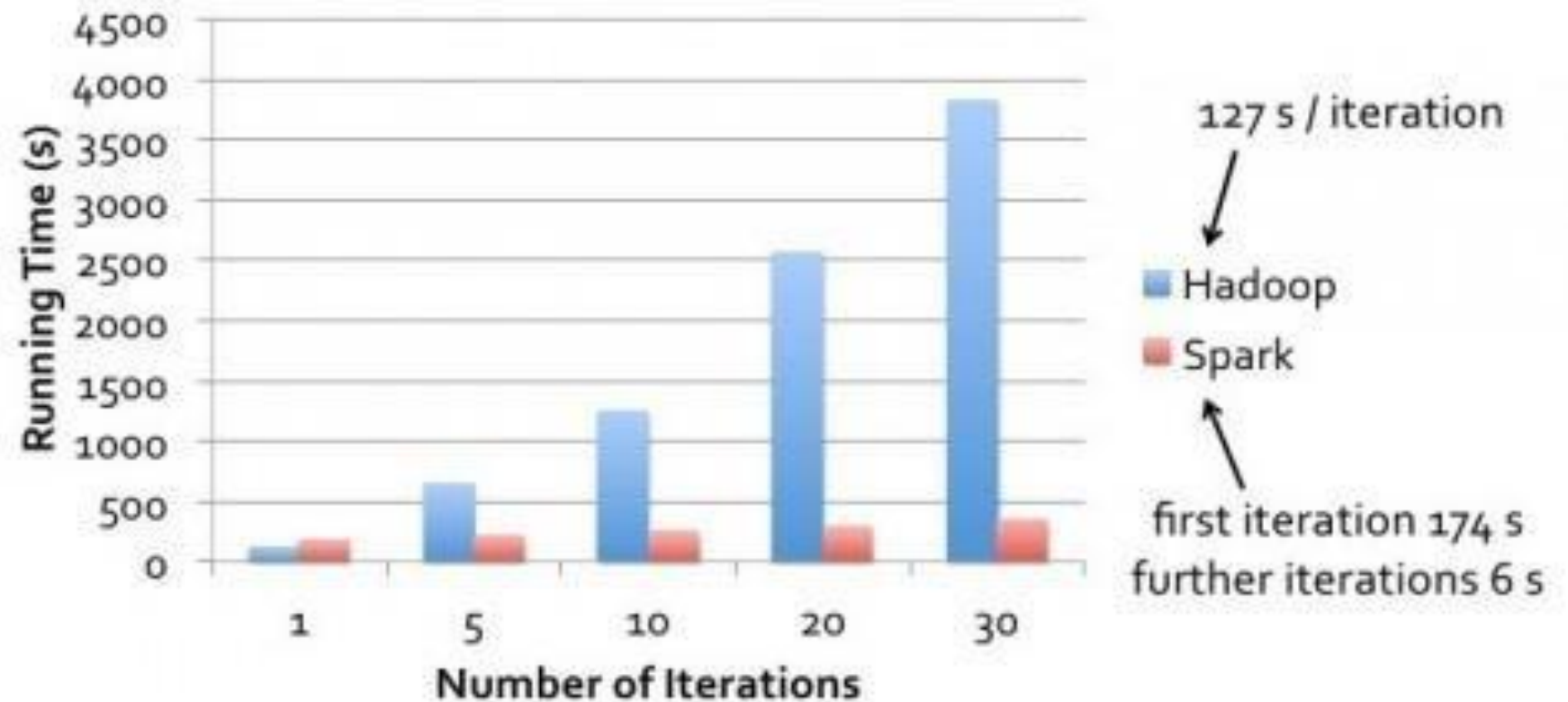
Ventajas



# 1. Introducción a SPARK

## Rendimiento

## Logistic Regression Performance



# 1. Introducción a SPARK

## Rendimiento

	<b>Hadoop World Record</b>	<b>Spark 100 TB</b>	<b>Spark 1 PB</b>
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400	6592	6080
# Reducers	10,000	29,000	250,000
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min
Sort Benchmark Daytona Rules	Yes	Yes	No
Environment	dedicated data center	EC2 (i2.8xlarge)	EC2 (i2.8xlarge)



# 1. Introducción a SPark

## COMPARACIÓN CÓDIGO

```
val file = spark.textFile("hdfs://...")

val counts = file.flatMap(line =>
line.split(" ")).map(word => (word,
1)).reduceByKey(_ + _)

counts.saveAsTextFile("hdfs://...")
```



```
1 package com.primacy.hadoop;
2
3 import org.apache.hadoop.fs.Path;
4
5
6
7
8
9
10
11 public class WordCountDriver {
12
13     public static void main(String[] args) {
14         JobClient client = new JobClient();
15         JobConf conf = new JobConf(WordCountDriver.class);
16
17         // specify output types
18         conf.setOutputKeyClass(Text.class);
19         conf.setOutputValueClass(IntWritable.class);
20
21         // specify input and output dirs
22         FileInputFormat.addInputPath(conf, new Path("input"));
23         FileOutputFormat.setOutputPath(conf, new Path("output"));
24
25         // specify a mapper
26         conf.setMapperClass(WordCountMapper.class);
27
28         // specify a reducer
29         conf.setReducerClass(WordCountReducer.class);
30         conf.setCombinerClass(WordCountReducer.class);
31
32         client.setConf(conf);
33         try {
34             JobClient.runJob(conf);
35         } catch (Exception e) {
36             e.printStackTrace();
37         }
38     }
39 }
```

# 1. Introducción a SPARK

El núcleo de spark (Spark core) esta basado en 2 componentes principales.

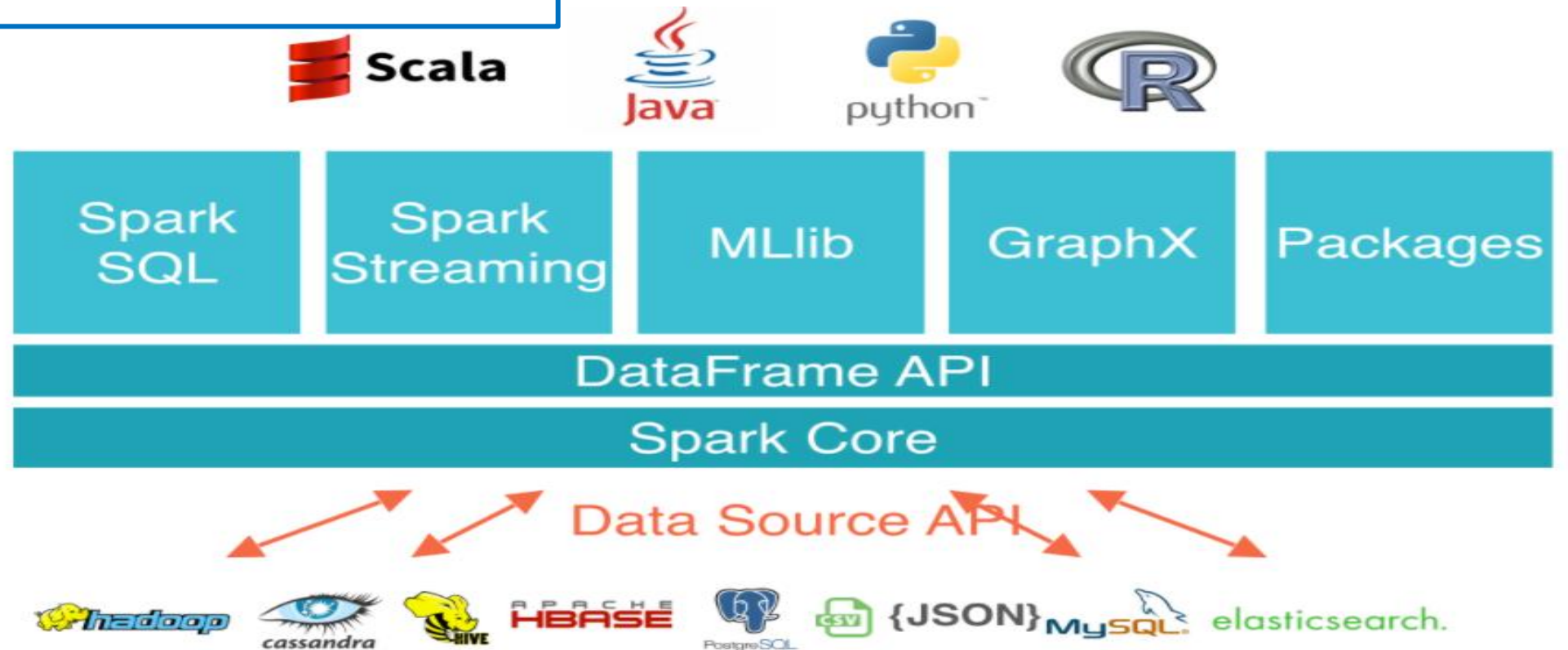
- Spark Context
- Spark config





# 1. Introducción a SPARK

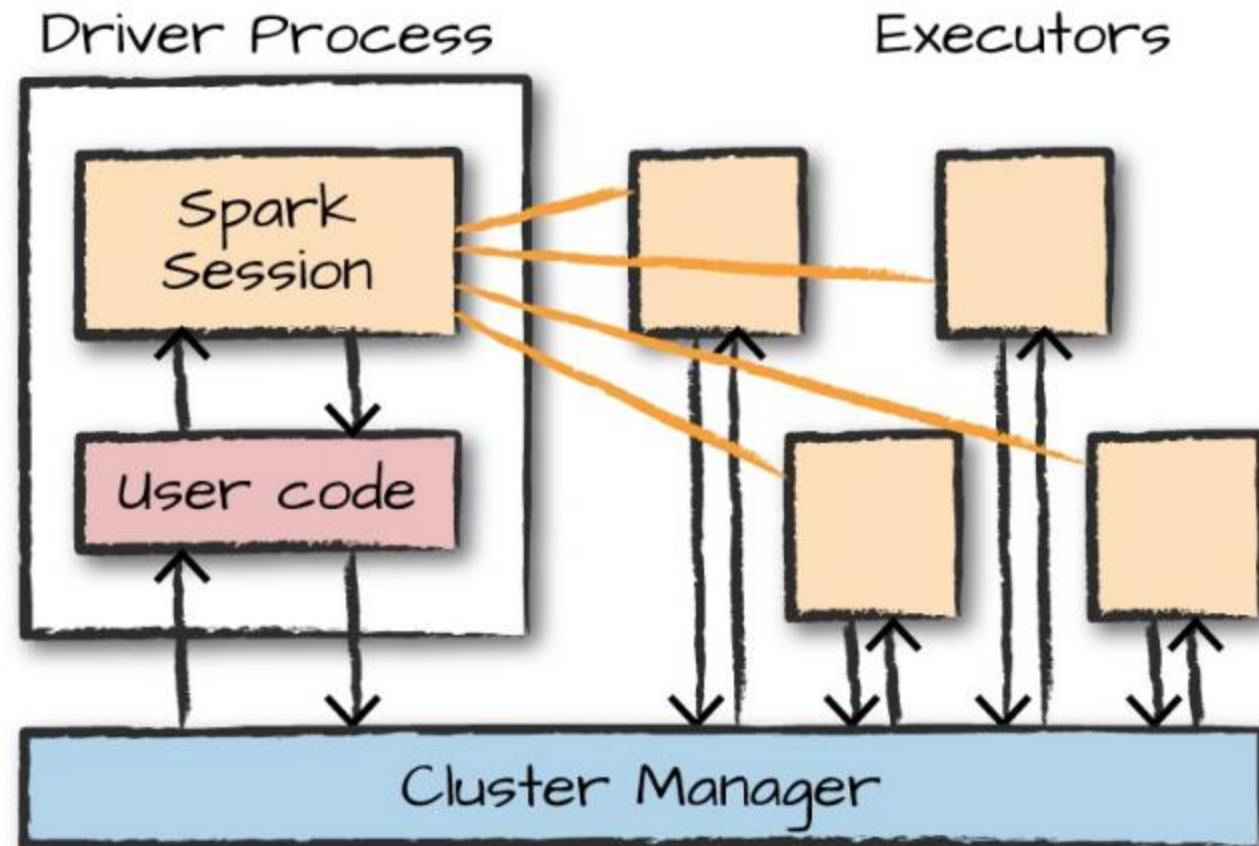
Adicionalmente SPARK tiene una gran variedad de herramientas, librería y lenguaje tales como: MLIB, GRAPHX Y SPARKR



# 1. Introducción a SPARK

En orden de entender como Spark trabaja vamos a entender los principios básicos de la arquitectura de SPARK, el cual consta de 3 componentes principales:

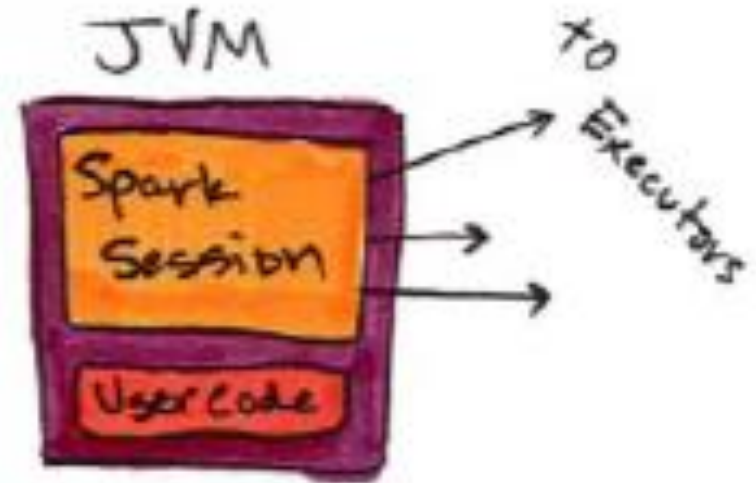
- Driver Process
- Executors
- Cluster Manager



# 1. Introducción a SPARK

## Driver Process:

- Mantiene la información acerca de la aplicación.
- Responde el programa del usuario.
- Analiza, distribuye y programa el trabajo a través de los executors.
- Es encargado de instanciar el Spark Session.



# 1. Introducción a SPARK

## Driver Process:

- Ejemplo:

Entramos al jupyter y ingresamos la siguiente instrucción, en la sección `.appName("XXXX")`, reemplazar por su nombre

```
In [ ]: import findspark
        findspark.init()
        import pyspark
        from pyspark.sql import SparkSession
        spark = SparkSession.builder.appName('SESSION_NAME').getOrCreate()
```

```
: findspark.find()
```

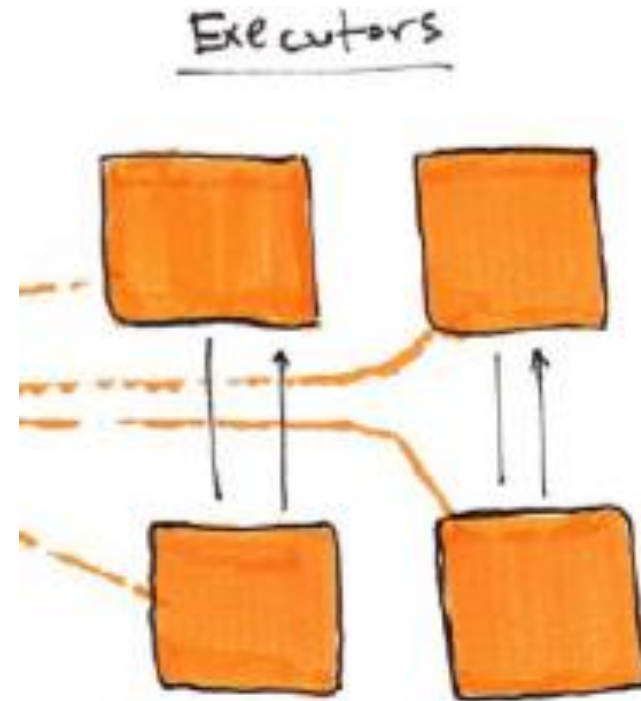
```
: '/opt/cloudera/parcels/SPARK2/lib/spark2'
```



# 1. Introducción a SPARK

## Executors Process:

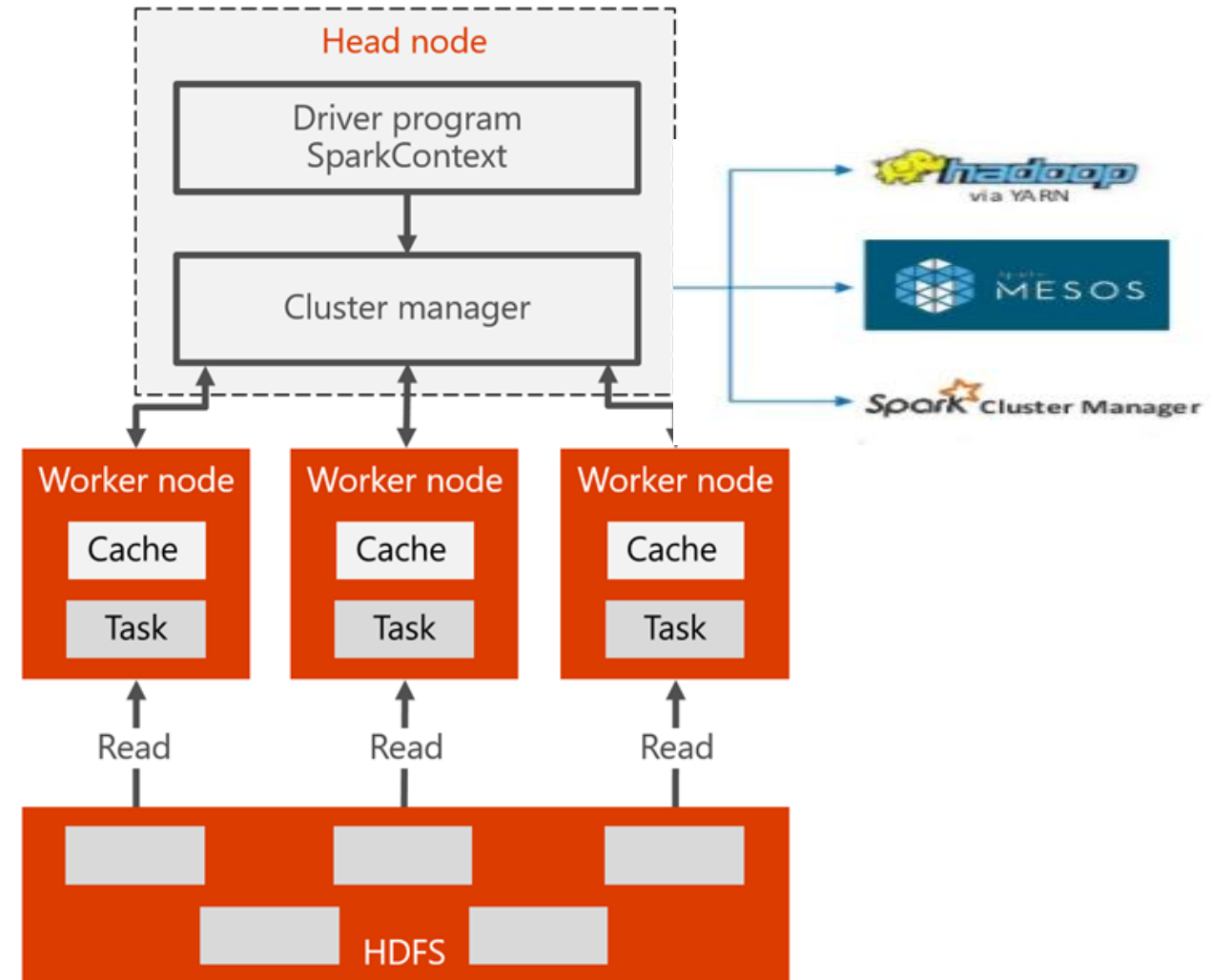
- Ejecutor de código asignado para driver.
- Reporte al estado de procesamiento



# 1. Introducción a SPARK

## Cluster Manager:

- Es el encargado de administrar las máquinas físicas y allocate (reservar) los recursos para las aplicaciones SPARK, entre los clúster managers mas utilizados se encuentran:
  - Spark standalone (local mode)
  - Yarn
  - mesos



# 1. Introducción a SPARK

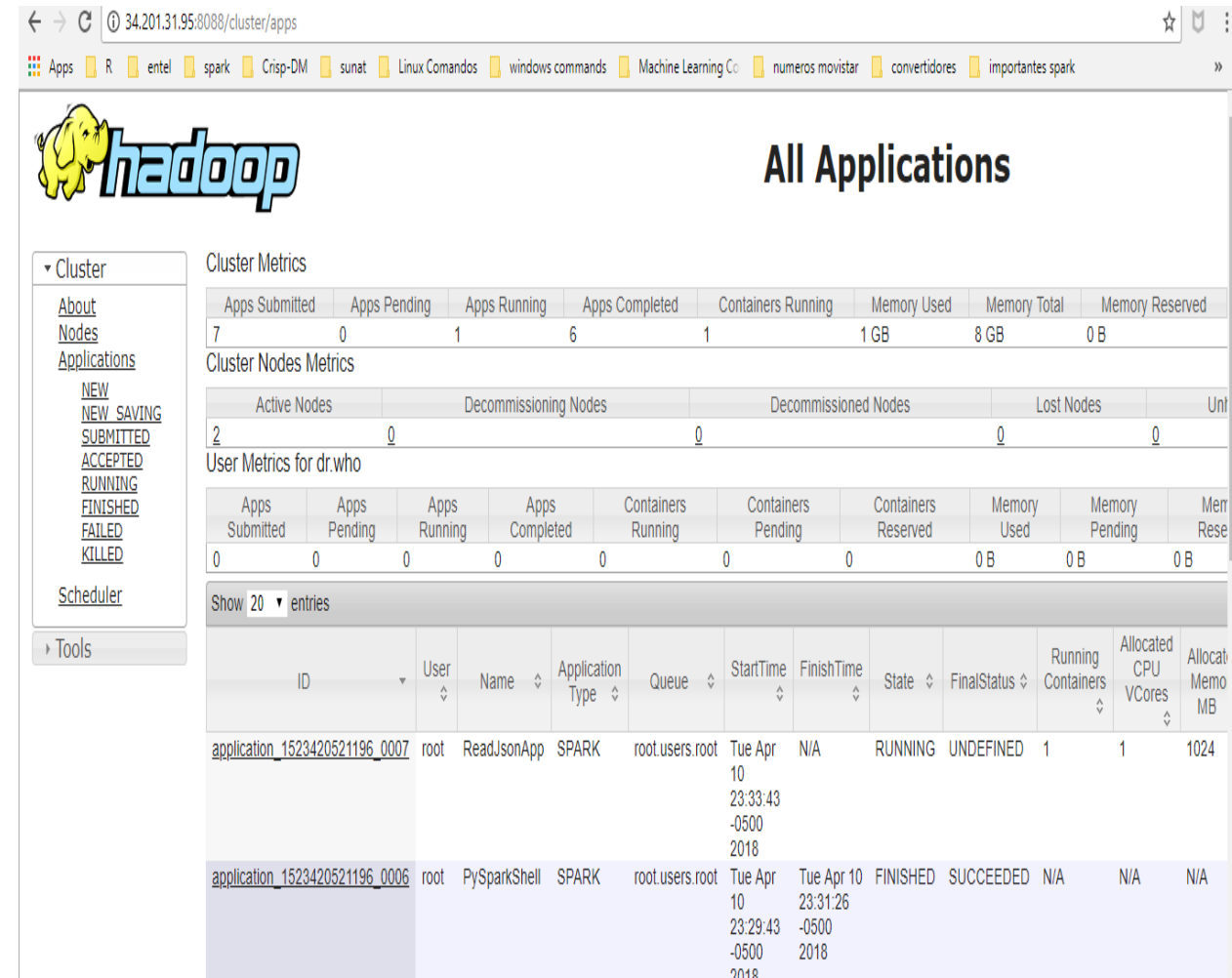
## Cluster Manager:

Ejemplo:

Para ir a la interfaz del cluster manager, o cluster UI nos dirigimos al siguiente enlace:

<http://52.204.240.67:8088/cluster/>

Donde la ip ingresada es la ip publica de tu instancia master que has registrado



The screenshot displays the Hadoop Cluster Manager interface. At the top, the Hadoop logo is visible. The main heading is "All Applications". Below this, there are several sections:

- Cluster Metrics:** A table showing overall cluster statistics.
- Cluster Nodes Metrics:** A table showing metrics for active and decommissioning nodes.
- User Metrics for dr.who:** A table showing metrics for the user 'dr.who'.
- Applications List:** A table listing individual applications with columns for ID, User, Name, Application Type, Queue, Start Time, Finish Time, State, Final Status, Running Containers, Allocated CPU V-Cores, and Allocated Memory MB.

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved
7	0	1	6	1	1 GB	8 GB	0 B

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unlabeled Nodes
2	0	0	0	0

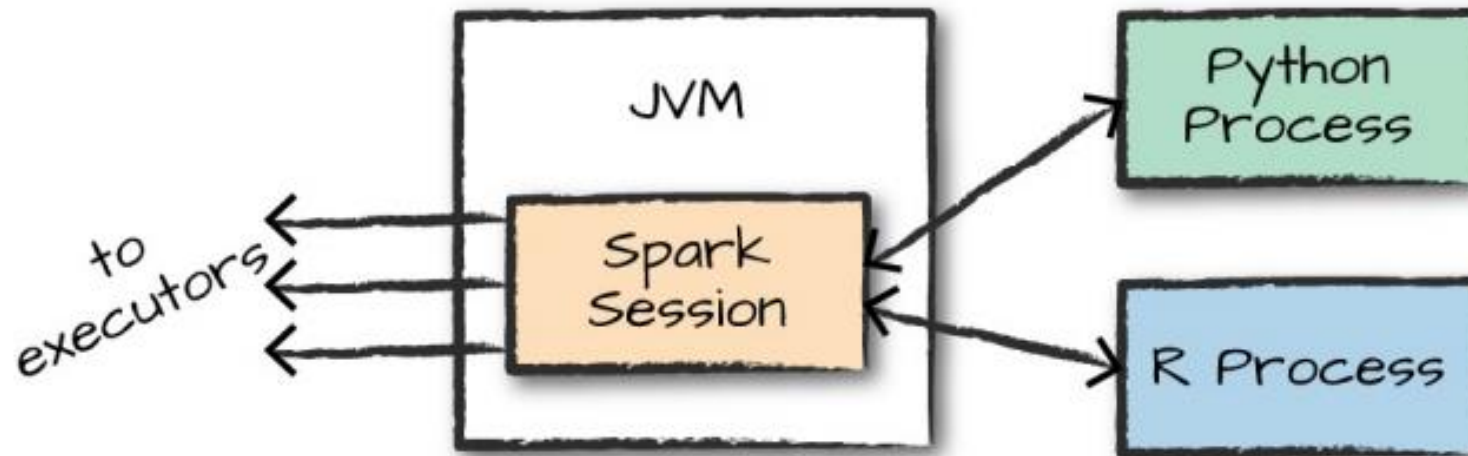
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved
0	0	0	0	0	0	0	0 B	0 B	0 B

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU V-Cores	Allocated Memory MB
application_1523420521196_0007	root	ReadJsonApp	SPARK	root.users.root	Tue Apr 10 23:33:43 -0500 2018	N/A	RUNNING	UNDEFINED	1	1	1024
application_1523420521196_0006	root	PySparkShell	SPARK	root.users.root	Tue Apr 10 23:29:43 -0500 2018	Tue Apr 10 23:31:26 -0500 2018	FINISHED	SUCCEEDED	N/A	N/A	N/A

# 1. Introducción a SPARK

API Lenguaje SPARK permite correr códigos SPARK en otros lenguajes:

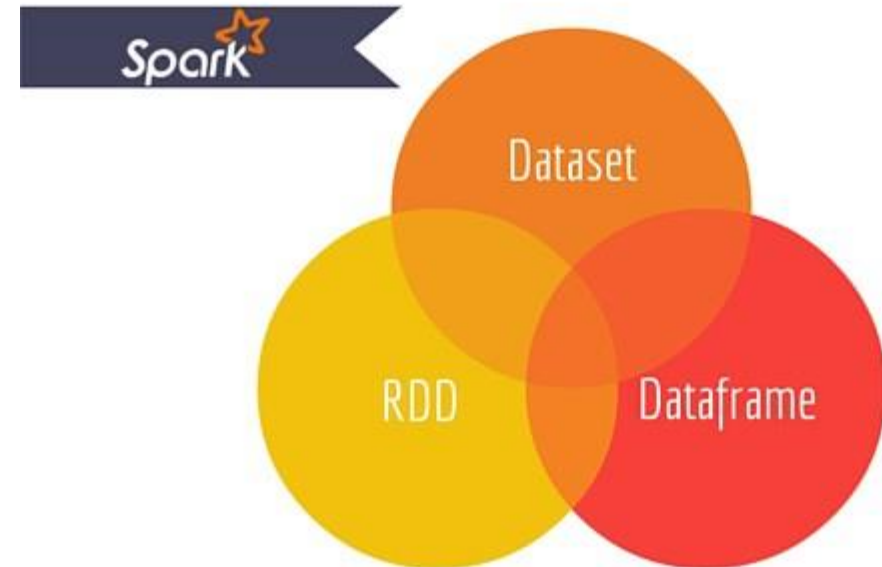
- SCALA : Es el lenguaje nativo de SPARK, haciendo el lenguaje por defecto
- PYTHON : Soporta casi todo lo que SCALA puede hacer.
- JAVA: Aunque SPARK esta basado en SCALA los autores se han asegurado que se pueda escribir código SPARK en JAVA.
- SQL: SPARK soporta código de usuario escrito en ANSI 2003 COMPLIANT.
- R : SPARK soporta la ejecución del código R a través del proyecto SPARK R.





## 2. RDD, Dataset y Dataframe

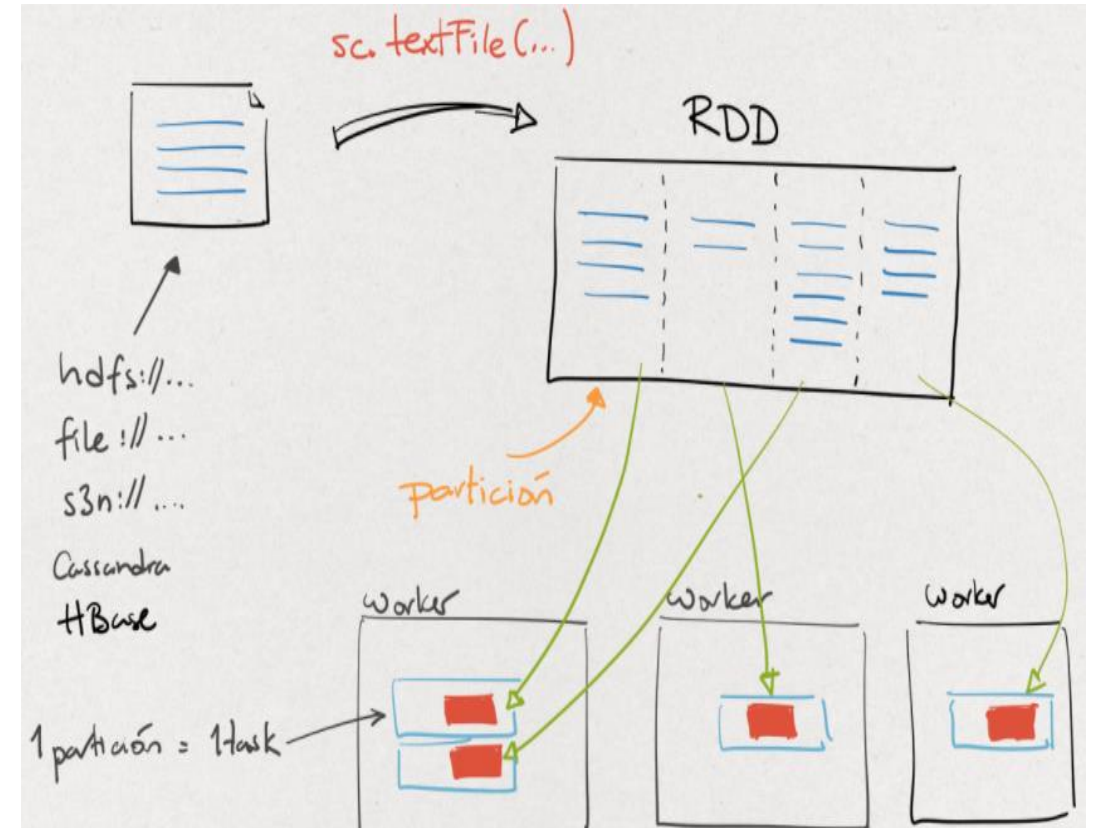
- Existen 3 API's principales para el manejo de los datos, estas abstracciones son:



## 2. RDD, Dataset y Dataframe

### RDD (Resilient Distributed Dataset)

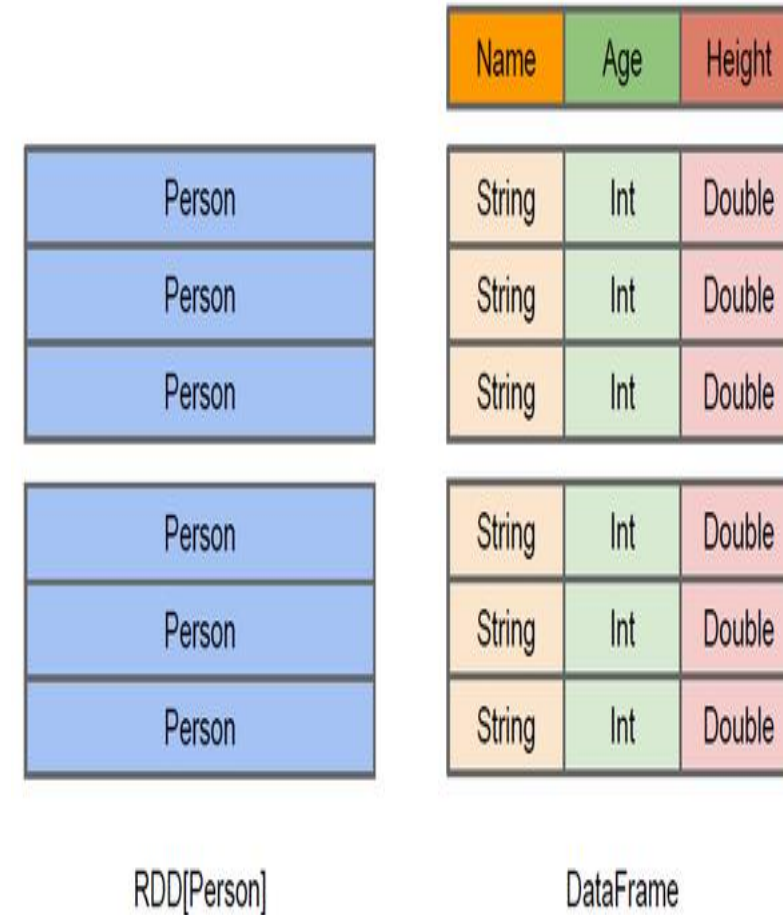
- La abstracción base es el RDD, la cual representa a una colección de objetos de solo lectura particionada a lo largo de varias maquinas.
- La razón de poner los datos en mas de una PCS es intuitiva, si los datos son tan grandes para ponerlos en una sola maquina tomaría mucho tiempo para realizarlo en una sola PCS.
- Hay ocasiones donde lo más sencillo sigue siendo usar RDD, por ejemplo:
  - Si tus datos están sin estructurar
  - Si quieres manipular los datos funcionalmente



## 2. RDD, Dataset y Dataframe

### Dataframe:

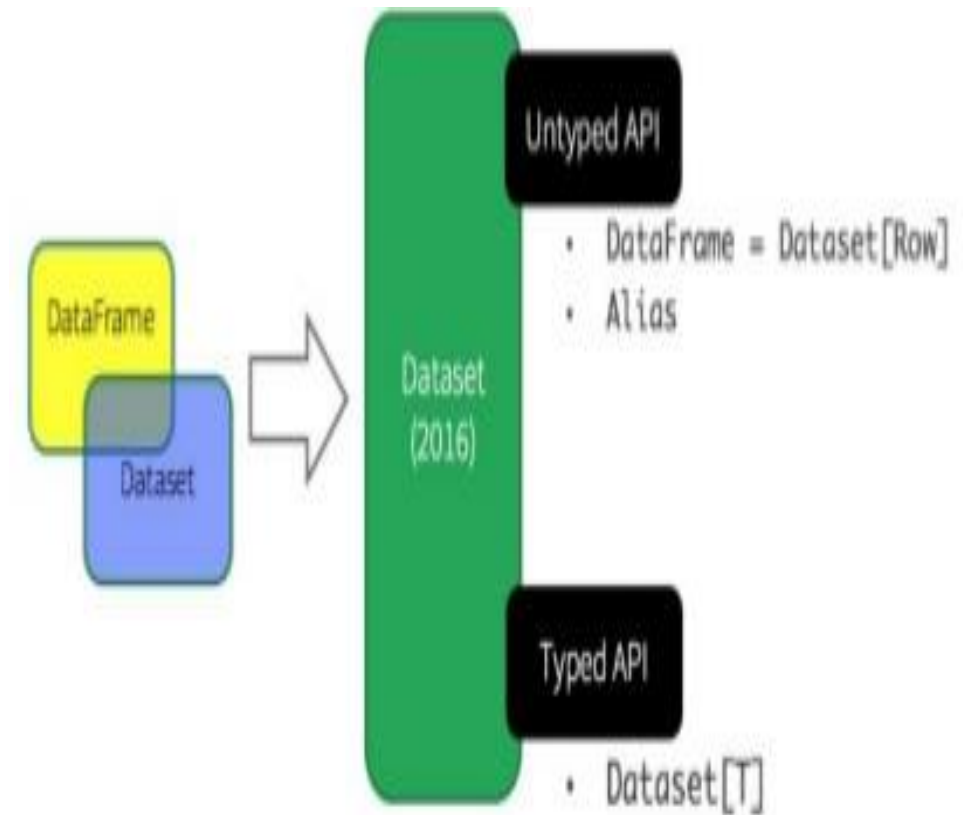
- Es una tabla de datos en filas y columnas, es nombrado por una lista de columnas y sus tipos de datos. Esta idea se ha llevado más lejos, y a partir de Saprk 2.0, Dataset es la nueva abstracción de Spark.
- Una analogía simple es una hoja de calculo Excel, la diferencia principal es que una hoja de cálculos se encuentra en una computadora, mientras un DATAFRAME puede estar alojado en cientos de PCS .
- Básicamente un DataFrame es un RDD[Row] donde Row es una tupla (o un Array[Any] en scala).
- Es preferible usar DataFrames cuando:
  - Quieres una abstracción rica con mucha semántica
  - Quieres utilizar los beneficios de Catalyst
  - Eres usuario de R o python



## 2. RDD, Dataset y Dataframe

### Datasets:

- Solo funciona con Scala y Java.
- Posee todas las funcionalidades optimizadas del dataframe.
- Provee Type-safety el cual no esta disponible en dataframe, nos ayuda a poder realizar transformaciones sobre los dataset sin especificar el tipo de dato (lambda functions).
- Es orientado a objetos POO.

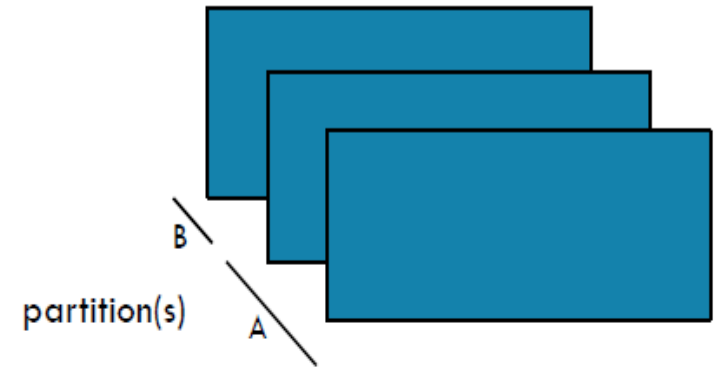




## 2. RDD, Dataset y Dataframe

### Particiones:

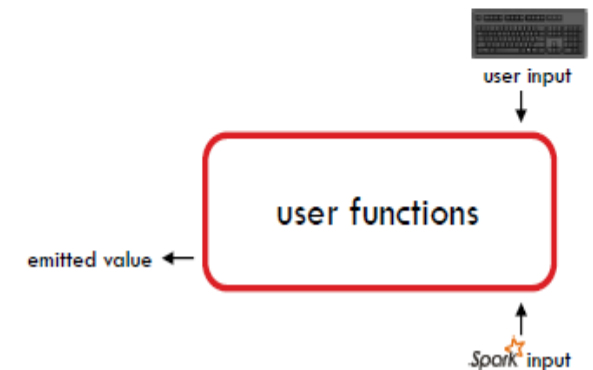
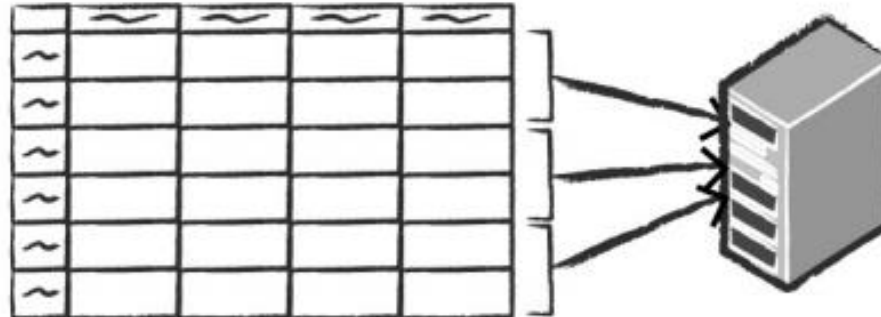
- En orden de aprovechar los mayores recursos en un CLUSTER, SPARK distribuye los datos en bloques llamados particiones.
- Una partición es un conjunto de filas que se sitúa en una maquina física en un clúster. Un dataframe puede tener 0 o mas particiones .
- Cuando se ejecuta alguna computación (acciones o transformaciones), Spark opera en cada partición en paralelo a menos que una operación requiera un shuffle, donde múltiples particiones necesitan compartir datos.



Spreadsheet on  
a single machine




Table or Data Frame  
partitioned across servers  
in a data center



### 3. Formas de computación en Spark

- Spark organiza su computación en dos categorías en:
  - Transformaciones
  - Acciones

**Spark**  Operations =

  
**TRANSFORMATIONS**

+



**ACTIONS**

# 3. Formas de computación en Spark

- Transformaciones:
  - En spark la estructura CORE son inmutables eso significa que no puede cambiarse una vez creada, puede parecer extraño al comienzo, si no cambia nada como lo usare ?.
  - A fin de cambiar un objeto DATAFRAME tu debes instruir a SPARK como deseas modificarlo, estas instrucciones se llaman transformaciones, las cuales son la forma como usted (como usuario) especifica como desea transformar el DATAFRAME (agruparlo, filtrarlo, ordenarlo, etc).
  - Las transformaciones son ejecutadas en Lazy Evaluation.

### 3. Formas de computación en Spark

- Transformaciones:
  - Ejemplo:

```
In [81]: myRange = spark.range(1000).toDF("number")
        divisBy2 = myRange.where("number % 2 = 0")
        divisBy2
```

```
Out[81]: DataFrame[number: bigint]
```

```
In [82]: divisBy2.explain()

== Physical Plan ==
*Project [id#271L AS number#274L]
+- *Filter ((id#271L % 2) = 0)
   +- *Range (0, 1000, step=1, splits=2)
```

# 3. Formas de computación en Spark

---

- **Acciones:**

- Para ejecutar (TRIGGER) la computación se ejecuta una acción.
- Una acción instruye a SPARK a computar un resultado de una serie de transformaciones, la acción mas simples es COUNT, la cual devuelve el total de registros en un DATAFRAME.

### 3. Formas de computación en Spark

- Acciones :
  - Ejemplo:

```
In [92]: sorteddivisBy2=divisBy2.sort("number")
```

```
In [93]: sorteddivisBy2.explain()
```

```
== Physical Plan ==
*Sort [number#299L ASC NULLS FIRST], true, 0
+- Exchange rangepartitioning(number#299L ASC NULLS FIRST, 200)
   +- *Project [id#296L AS number#299L]
      +- *Filter ((id#296L % 2) = 0)
         +- *Range (0, 1000, step=1, splits=2)
```

```
In [96]: sorteddivisBy2.show()
```

```
+-----+
|number|
+-----+
|      0|
|      2|
|      4|
|      6|
|      8|
|     10|
|     12|
```



# 3. Formas de computación en Spark

- Transformaciones y acciones :
  - Soportan dos tipos de operaciones: – Transformaciones – Acciones
  - Las transformaciones construyen un nuevo RDD o Dataframe a partir del anterior.
    - El cual queda guardado en el lineage graph (DAG)
  - Las acciones calculan el resultado basado en el RDD o Dataframe.
  - La diferencia es que las transformaciones son computadas de manera lazy y sólo son ejecutadas hasta la acción.

### 3. Formas de computación en Spark

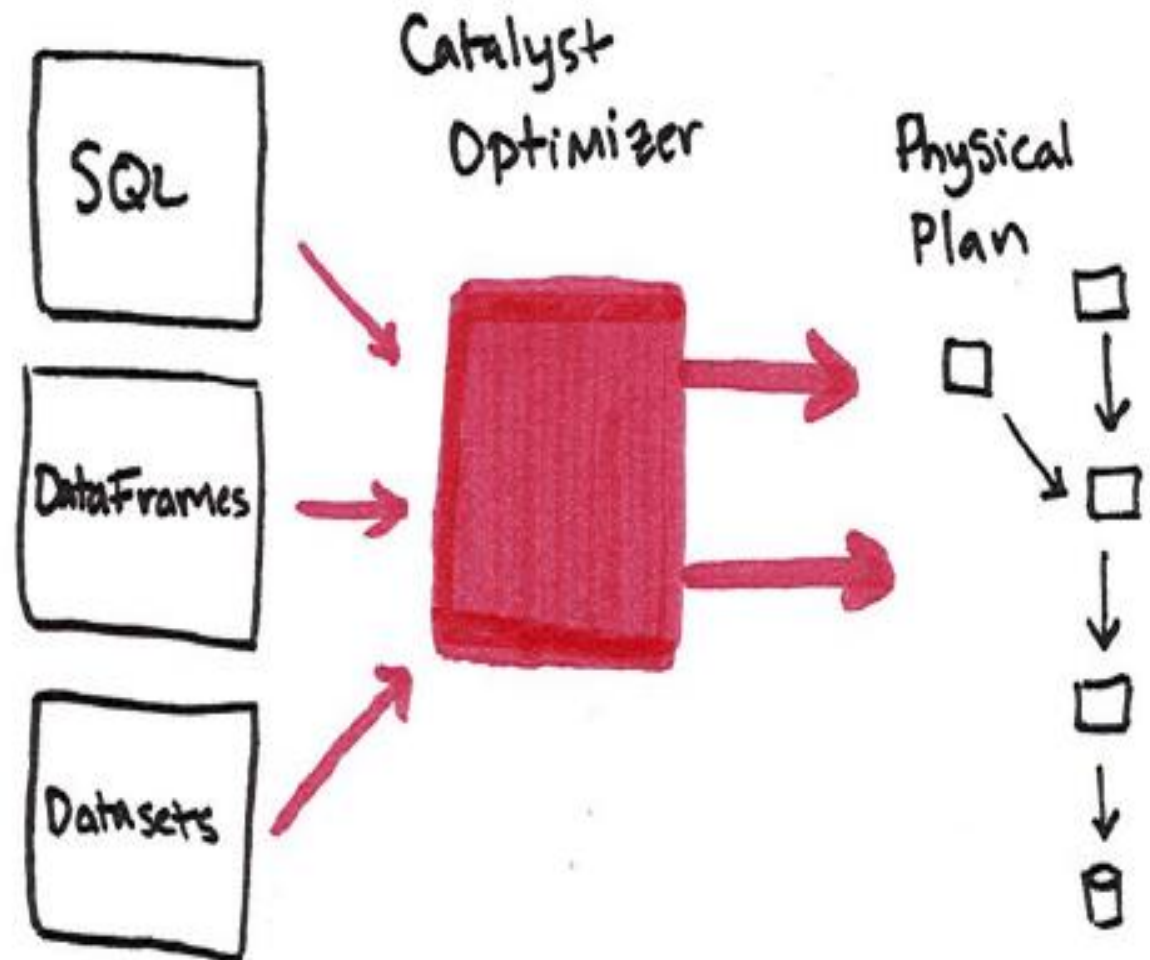
- Transformaciones y acciones :

- Plan de ejecución:

Como hemos revisado en los códigos anteriores, aparece un termino cuando ejecutamos la función `explain()`, al parecer se refiere a un plan de ejecución, existen dos planes de ejecución:

- Logical Planning
    - Physical Planning

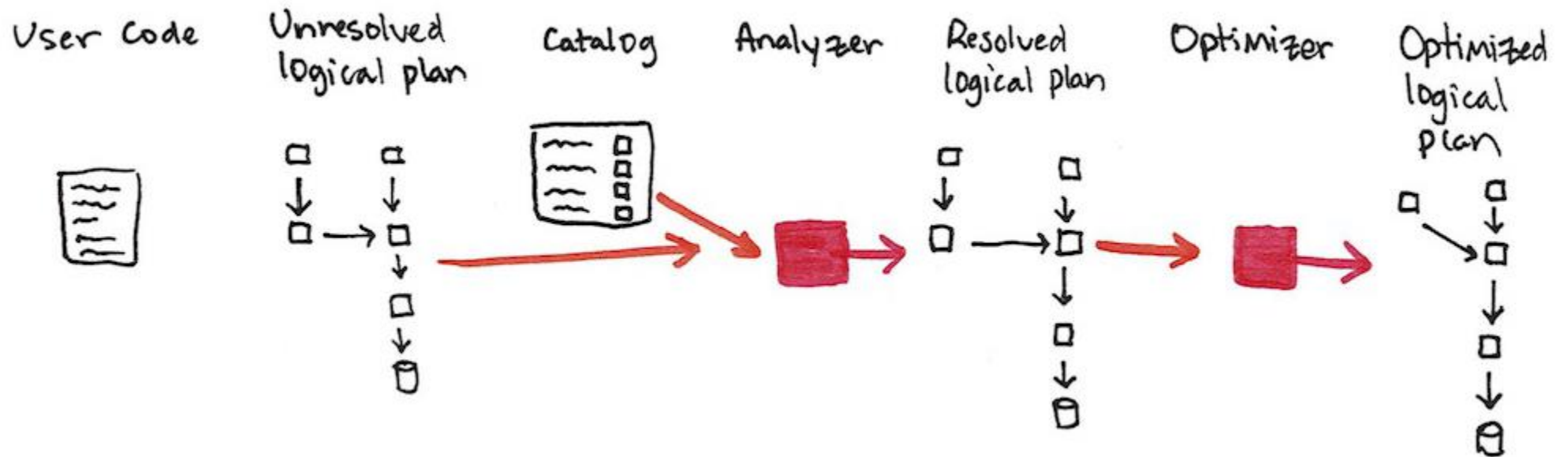
Estos tipos de planes son utilizados para optimizar la ejecución de tu script spark sobre el Cluster,



### 3. Formas de computación en Spark

- Transformaciones y acciones :  
Plan de ejecución: / Logical Planning:

La primera fase en la ejecución de un script es realizar el plan lógico, que no es mas que convertir tu código en un plan de ejecución

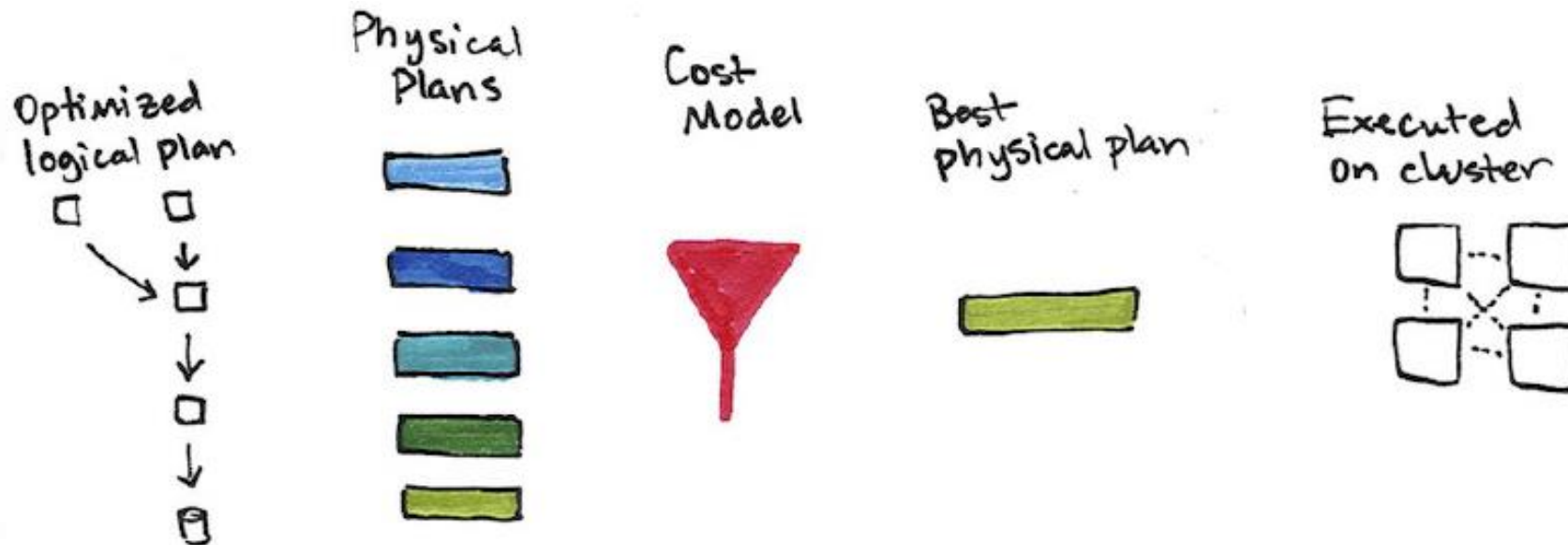


### 3. Formas de computación en Spark

- Transformaciones y acciones :

Plan de ejecución: / Physical Planning:

Después de haber creado el plan lógico, el plan físico lo que va a realizar es buscar la mejor estrategia de ejecución realizando diferentes planes físicos de ejecución mediante un modelo de costos.



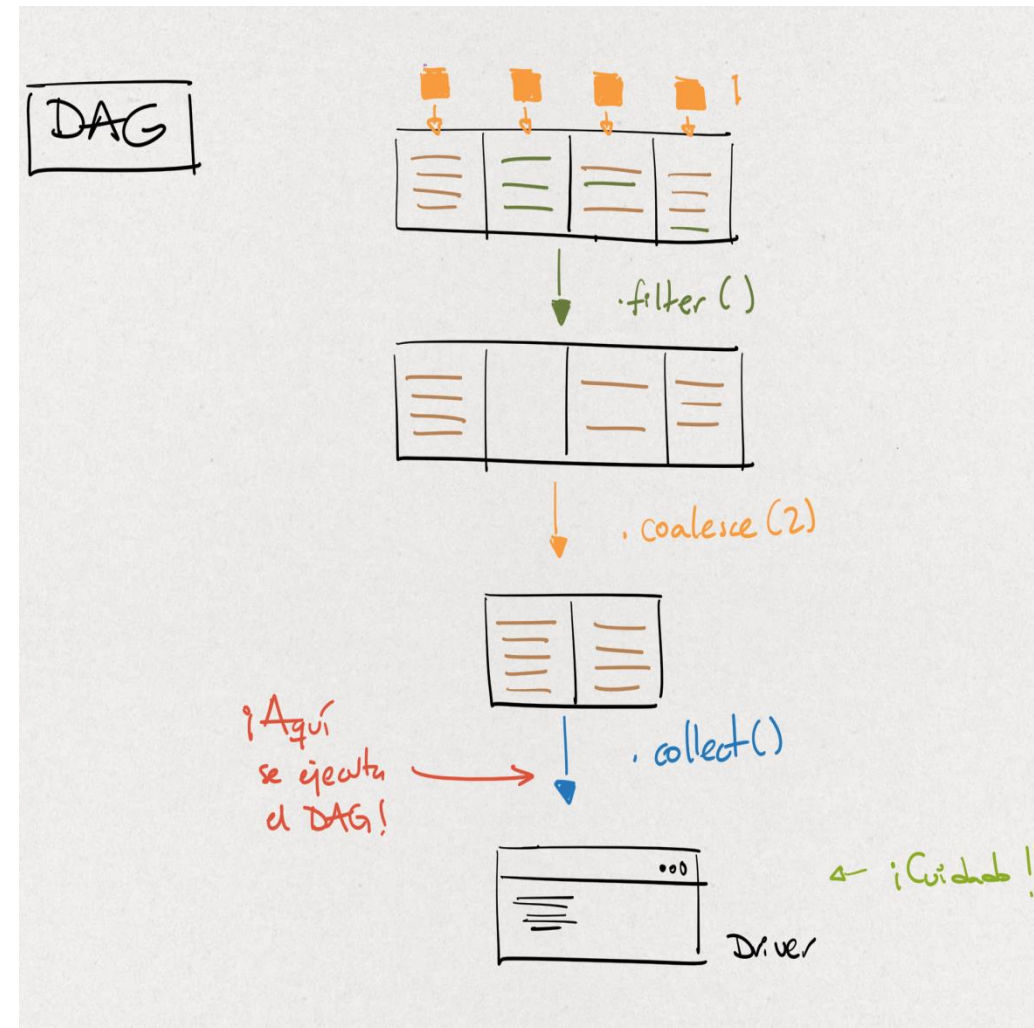
### 3. Formas de computación en Spark

- Transformaciones y acciones :

Plan de ejecución: / Physical Planning:

Al correr un código spark y luego de seleccionar el plan físico de ejecución, Spark corre todo el código sobre RDD la interfaz de programación de mas bajo nivel.

La forma gráfica de representación de este plan físico de transformaciones y acciones se le conoce como DAG (Direct Acyclic Graph), y se realiza luego de una **acción**.



## 4. Manejo de archivos y conexiones

---

- Manejo archivos:

Una de las razones de la popularidad de Spark es por la habilidad que tiene en leer y escribir en diferente variedad de data sources:

- CSV
- Json
- Parquet
- JDBC



## 4. Manejo de archivos y conexiones

- Manejo archivos:

Read and write CSV Files:

Ejemplo:

```
In [4]: import findspark
        findspark.init()
        import pyspark
        from pyspark.sql import SparkSession
        spark = SparkSession.builder.appName('ReadCSVApp').getOrCreate()
```

```
In [7]: fligthData2015CSV = spark.read.csv('hdfs:///tmp/clasespark/2015-summary.csv',inferSchema=True,header=True)
```

```
In [8]: fligthData2015CSV
```

```
Out[8]: DataFrame[DEST_COUNTRY_NAME: string, ORIGIN_COUNTRY_NAME: string, count: int]
```

## 4. Manejo de archivos y conexiones

- Manejo archivos:

Read and write Json Files:

Ejemplo:

```
[4]: import findspark
      findspark.init()
      import pyspark
      from pyspark.sql import SparkSession

      spark = SparkSession.builder.appName('ReadJsonApp').getOrCreate()

In [29]: findspark.find()
Out[29]: '/opt/cloudera/parcels/SPARK2/lib/spark2'

In [36]: fligthData2015=spark.read.json('hdfs:///tmp/clasespark/2015-summary.json')

In [37]: fligthData2015
Out[37]: DataFrame[DEST_COUNTRY_NAME: string, ORIGIN_COUNTRY_NAME: string, count: bigint]
```

```
csvFile.write.format("csv")
      .mode("overwrite")
      .option("sep", "\t")
      .save("/tmp/my-tsv-file.tsv")
```

## 4. Manejo de archivos y conexiones

- Manejo archivos:

Read and write Parquet Files:  
Ejemplo:

```
In [4]: import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('ReadParquetApp').getOrCreate()
```

```
In [13]: fligthData2015Parquet= spark.read.format("parquet").\
load('hdfs:///tmp/clasespark/2010-summary.parquet')
```

```
In [14]: fligthData2015Parquet.show(5)
```

```
+-----+-----+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|
+-----+-----+-----+
|      United States|          Romania|    1|
|      United States|          Ireland|  264|
|      United States|           India|   69|
|           Egypt|    United States|   24|
|Equatorial Guinea|    United States|    1|
+-----+-----+-----+
only showing top 5 rows
```

## 4. Manejo de archivos y conexiones

- Manejo archivos:

Read and JDBC Source:

Ejemplo:

```
In [1]: import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('ReadParquetFiles').getOrCreate()
```

```
In [7]: url = "jdbc:mysql://dbdata01.ccpetoqzkfsy.us-east-1.rds.amazonaws.com:3306/DBRIO"
connectionProperties = {
    'user' : 'admin',
    'password' : 'admin123'
}
pushdown_query = "(select * from venta_diaria ) venta"
```

```
In [8]: df_mysql = spark.read.jdbc(url=url, table=pushdown_query, properties=connectionProperties)
```

```
In [9]: type(df_mysql)
```

```
Out[9]: pyspark.sql.dataframe.DataFrame
```

```
In [13]: df_mysql.write.format("parquet").mode("overwrite").save("hdfs:///tmp/clasespark/my-parquet-file.parquet")
```

## 5. Spark SQL

Spark provee otra manera de manejar los dataframes y es mediante SQL; Spark SQL nos permite convertir cualquier dataframe en una tabla y utilizar las sentencias conocidas de query usando puro SQL. No existe diferencia en tiempo de procesamiento en ejecutar una rutina con query o dataframe sintaxis.

Ejemplo:

Utilizemos el archivo fligthData2015 para nuestro ejemplo:

```
In [58]: fligthData2015.createOrReplaceTempView("flight_data_2015")
```

```
In [59]: sqlWay = spark.sql("""
        SELECT DEST_COUNTRY_NAME, count(1)
        FROM flight_data_2015
        GROUP BY DEST_COUNTRY_NAME
        """)
```

```
In [61]: dataFrameWay = fligthData2015.groupBy("DEST_COUNTRY_NAME").count()
```

## 5. Spark SQL

Ejemplo:

```
In [62]: sqlWay.explain()
```

```
== Physical Plan ==
*HashAggregate(keys=[DEST_COUNTRY_NAME#92], functions=[count(1)])
+- Exchange hashpartitioning(DEST_COUNTRY_NAME#92, 200)
   +- *HashAggregate(keys=[DEST_COUNTRY_NAME#92], functions=[partial_count(1)])
      +- *FileScan json [DEST_COUNTRY_NAME#92] Batched: false, Format: JSON, Location: InMemoryFileIndex[hdfs://ip-10-0-0-74.ec
2.internal:8020/tmp/clasespark/2015-summary.json], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<DEST_COUNTRY_NAM
E:string>
```

```
In [63]: dataframeWay.explain()
```

```
== Physical Plan ==
*HashAggregate(keys=[DEST_COUNTRY_NAME#92], functions=[count(1)])
+- Exchange hashpartitioning(DEST_COUNTRY_NAME#92, 200)
   +- *HashAggregate(keys=[DEST_COUNTRY_NAME#92], functions=[partial_count(1)])
      +- *FileScan json [DEST_COUNTRY_NAME#92] Batched: false, Format: JSON, Location: InMemoryFileIndex[hdfs://ip-10-0-0-74.ec
2.internal:8020/tmp/clasespark/2015-summary.json], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<DEST_COUNTRY_NAM
E:string>
```

Como se puede observar los dos generan el mismo plan de ejecución



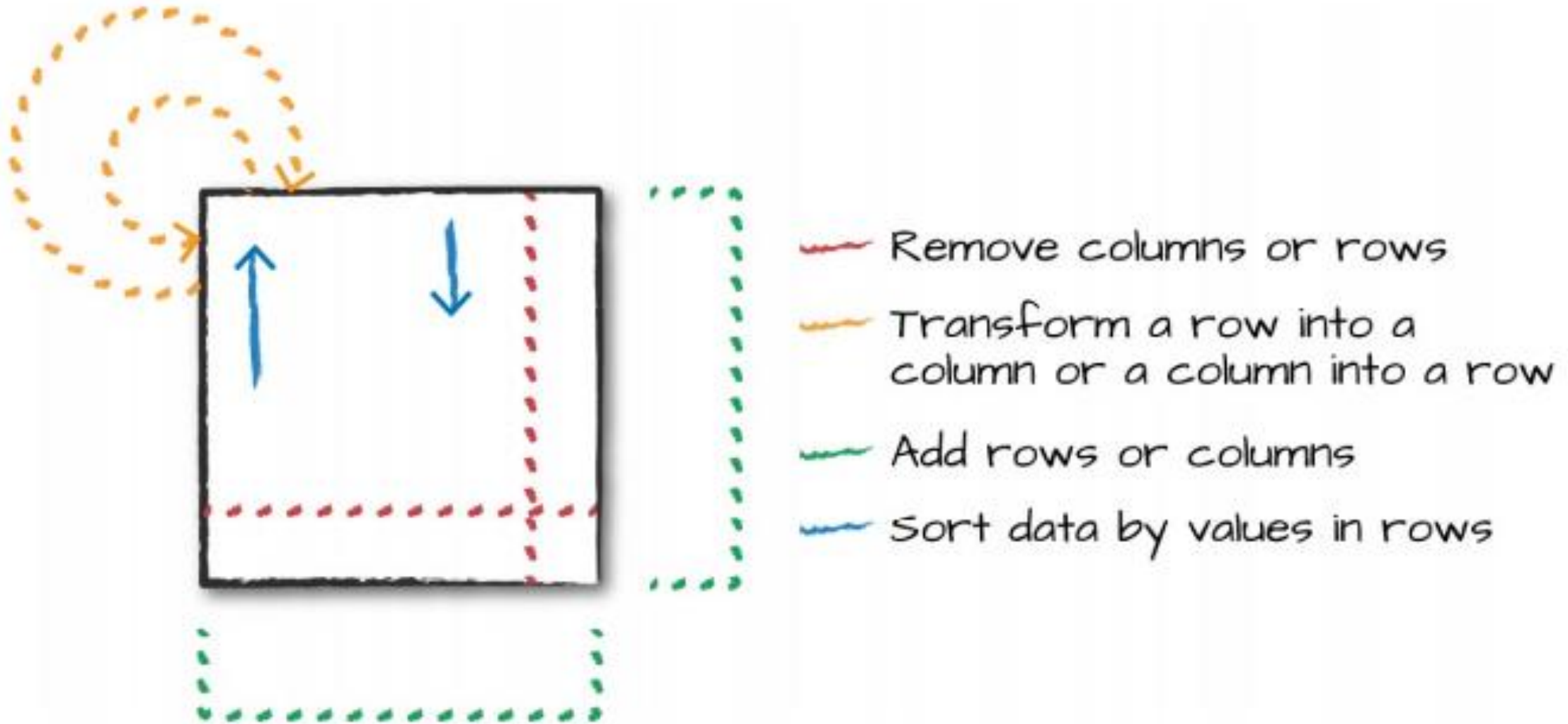
## 5. Spark SQL

---

Ejemplo:

Ahora sobre el mismo datarame `fligthData2015`, hagamos algo mas complicado obteniendo los 5 primeras ciudades destinos en el conjunto de datos, y luego guardemos , en SQL sintaxis.

## 6. Operaciones con dataframes



## 6. Operaciones con dataframes

- A continuación revisaremos las mas usadas operaciones sobre dataframes.

Muestra un resumen del contenido de la tabla

```
In [110]: fligthData2015.show()
```

+-----+-----+-----+			
DEST_COUNTRY_NAME		ORIGIN_COUNTRY_NAME	count
+-----+-----+-----+			
	United States	Romania	15
	United States	Croatia	1
	United States	Ireland	344
	Egypt	United States	15
	United States	India	62
	United States	Singapore	1
	United States	Grenada	62
	Costa Rica	United States	588
	Senegal	United States	40
	Moldova	United States	1
	United States	Sint Maarten	325
	United States	Marshall Islands	39
	Guyana	United States	64

## 6. Operaciones con dataframes

Muestra un resumen del contenido de la tabla

```
In [110]: fligthData2015.show()
```

DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
United States	Romania	15
United States	Croatia	1
United States	Ireland	344
Egypt	United States	15
United States	India	62
United States	Singapore	1
United States	Grenada	62
Costa Rica	United States	588
Senegal	United States	40
Moldova	United States	1
United States	Sint Maarten	325
United States	Marshall Islands	39
Guyana	United States	64

## 6. Operaciones con dataframes

---

```
In [114]: fligthData2015.printSchema()
```

```
root
 |-- DEST_COUNTRY_NAME: string (nullable = true)
 |-- ORIGIN_COUNTRY_NAME: string (nullable = true)
 |-- count: long (nullable = true)
```

```
In [115]: # Muestra las columnas del dataframe
```

```
In [116]: fligthData2015.columns
```

```
Out[116]: ['DEST_COUNTRY_NAME', 'ORIGIN_COUNTRY_NAME', 'count']
```

## 6. Operaciones con dataframes

```
In [119]: # Muestra un resumen de las observaciones que estan dentro del dataframe
```

```
In [120]: fligthData2015.describe().show()
```

summary	DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
count	256	256	256
mean	null	null	1770.765625
stddev	null	null	23126.516918551915
min	Algeria	Angola	1
max	Zambia	Vietnam	370002

```
In [124]: # Realizar filtros sobre el dataframe
```

```
In [125]: fligthData2015.filter((fligthData2015['count'] > 20)).show()
```

DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
United States	Ireland	344
United States	India	62
United States	Grenada	62
Costa Rica	United States	588
Senegal	United States	40



## 6. Operaciones con dataframes

```
In [126]: # Almacena en una variable el resultado
```

```
In [127]: fligthDataOriUSA=fligthData2015.filter((fligthData2015['ORIGIN_COUNTRY_NAME'] == "United States")).collect()
```

```
In [129]: type(fligthDataOriUSA)
```

```
Out[129]: list
```

## 6. Operaciones con dataframes

```
In [180]: from pyspark.sql.functions import countDistinct, avg, stddev, format_number, min, max, count, expr, sum
```

```
In [130]: # Agrupando datos
```

```
In [131]: fligthData2015.groupBy('ORIGIN_COUNTRY_NAME').mean().show()
```

ORIGIN_COUNTRY_NAME	avg(count)
Paraguay	6.0
Russia	161.0
Anguilla	38.0
Senegal	42.0
Sweden	119.0
Kiribati	35.0
Guyana	63.0
Philippines	126.0
Singapore	1.0
Malaysia	3.0
Fiji	25.0
Turkey	120.0

## 6. Operaciones con dataframes

```
In [ ]: # Transformaciones con multiples operaciones
```

```
In [183]: fligthData2015.filter((fligthData2015['count'] > 20))\
          .groupBy("DEST_COUNTRY_NAME")\
          .agg(
              count("count").alias("cantidad"),
              expr("sum(count)").alias("suma"),
              expr("avg(count)").alias("promedio")
          )\
          .orderBy(fligthData2015["DEST_COUNTRY_NAME"].desc())\
          .show()
```

DEST_COUNTRY_NAME	cantidad	suma	promedio
Venezuela	1	290	290.0
Uruguay	1	43	43.0
United States	90	411116	4567.9555555555555
United Kingdom	1	2025	2025.0
United Arab Emirates	1	320	320.0
Turks and Caicos ...	1	230	230.0
Turkey	1	138	138.0

## 6. Operaciones con dataframes

```
In [184]: # Formateando salida
```

```
In [161]: fligthData2015stdv.select(format_number('std',2)).show()
```

```
+-----+  
|format_number(std, 2)|  
+-----+  
|          23,126.52|  
+-----+
```



Thank you