

# Apache Spark

## Introducción

**Taller de Big Data**

# Agenda

- Apache Spark
- Estructura de datos: Spark RDD
- Operaciones: Transformaciones y Acciones
- Estructura de datos: Spark DataFrames

# Apache Spark

# ¿Qué es Apache Spark?

- Un motor general para procesamiento a gran escala de datos.
- Ejecuta en Apache YARN, *standalone*, en Amazon EC2 o sobre Apache Mesos.
  - Aquí sólo veremos el funcionamiento *standalone*
- Soporta varios lenguajes de programación
  - Python, Scala, Java y recientemente R.

# ¿Por qué Apache Spark?

- Soporta Algoritmos Iterativos
  - Como algoritmos de grafos
- Exploración Iterativa
- Puede ejecutar en *batch*.
- Soporta *streaming*.

# Historia

- Todo empezó con el proyecto **Mesos**, un *framework* distribuido de ejecución, realizado para una clase en UC Berkeley en 2009.
- **Spark** fué creado para probar **Mesos**.
- Fué tal su éxito que se abrió el código en 2010.

# Funcionamiento

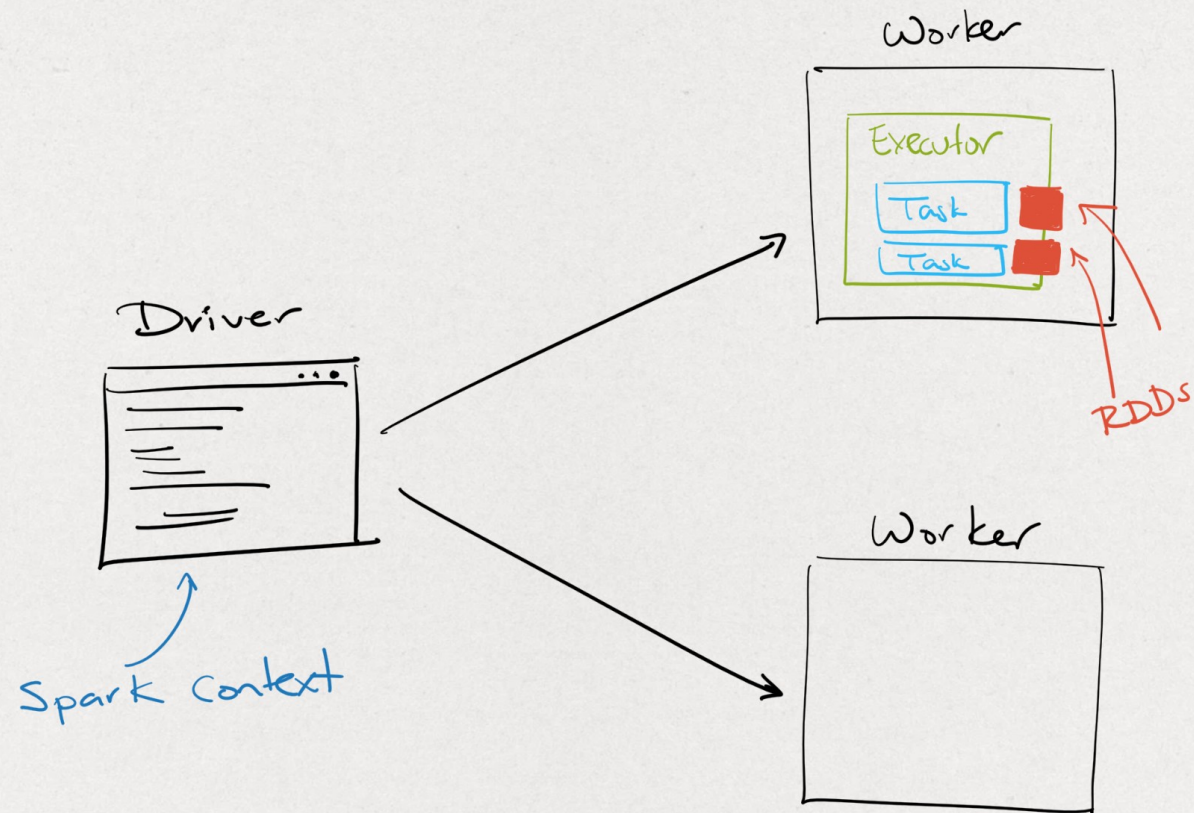
```
jovyan@211aca58f850:~$ /usr/local/spark/bin/pyspark
Python 2.7.9 (default, Mar 1 2015, 12:57:24)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
15/11/09 03:09:44 INFO SparkContext: Running Spark version 1.5.1
15/11/09 03:09:44 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
15/11/09 03:09:44 INFO SecurityManager: Changing view acls to: jovyan
15/11/09 03:09:44 INFO SecurityManager: Changing modify acls to: jovyan
15/11/09 03:09:44 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(jovyan); users with modify permissions: Set(jovyan)
15/11/09 03:09:45 INFO Slf4jLogger: Slf4jLogger started
15/11/09 03:09:45 INFO Remoting: Starting remoting
15/11/09 03:09:45 INFO Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriver@172.17.0.2:35263]
15/11/09 03:09:45 INFO Utils: Successfully started service 'sparkDriver' on port 35263.
15/11/09 03:09:45 INFO SparkEnv: Registering MapOutputTracker
15/11/09 03:09:45 INFO SparkEnv: Registering BlockManagerMaster
15/11/09 03:09:45 INFO DiskBlockManager: Created local directory at /tmp/blockmgr-373733e5-e6ae-42e2-a291-57e9e5135d90
15/11/09 03:09:45 INFO MemoryStore: MemoryStore started with capacity 530.3 MB
15/11/09 03:09:45 INFO HttpFileServer: HTTP File server directory is /tmp/spark-f605c1cc-148f-46f0-ba83-3817fc9e4fe5/httpd-05b227b1-017f-47e8-8c88-c671dc8c294b
15/11/09 03:09:45 INFO HttpServer: Starting HTTP Server
15/11/09 03:09:45 INFO Utils: Successfully started service 'HTTP file server' on port 44824.
15/11/09 03:09:45 INFO SparkEnv: Registering OutputCommitCoordinator
15/11/09 03:09:46 INFO Utils: Successfully started service 'SparkUI' on port 4040.
15/11/09 03:09:46 INFO SparkUI: Started SparkUI at http://172.17.0.2:4040
15/11/09 03:09:46 WARN MetricsSystem: Using default name DAGScheduler for source because spark.app.id is not set.
15/11/09 03:09:46 INFO Executor: Starting executor ID driver on host localhost
15/11/09 03:09:46 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 36024.
15/11/09 03:09:46 INFO NettyBlockTransferService: Server created on 36024
15/11/09 03:09:46 INFO BlockManagerMaster: Trying to register BlockManager
15/11/09 03:09:46 INFO BlockManagerMasterEndpoint: Registering block manager localhost:36024 with 530.3 MB RAM, BlockManagerId(driver, localhost, 36024)
15/11/09 03:09:46 INFO BlockManagerMaster: Registered BlockManager
Welcome to

  ____      _
 / ___|    / \
| |  | |  / _ \
| |  | | / ___ \
| |  | |/_/   \_\
| |  | |
|_|  |_|

 version 1.5.1

Using Python version 2.7.9 (default, Mar 1 2015 12:57:24)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

# Modelo de Ejecución





# Ciclo de vida

- Crear un RDD a partir de datos externos
- Transformarlo *lazily* en nuevos RDDs usando **transformaciones**.
- Usa `cache()` en los RDDs que vayas a reutilizar
- Utiliza **acciones** para iniciar la ejecución en paralelo

# Spark RDD

# Spark RDD

- *Resilient Distributed Dataset*
- Abstracción que representa una colección de objetos de sólo lectura particionada a lo largo de varias máquinas.

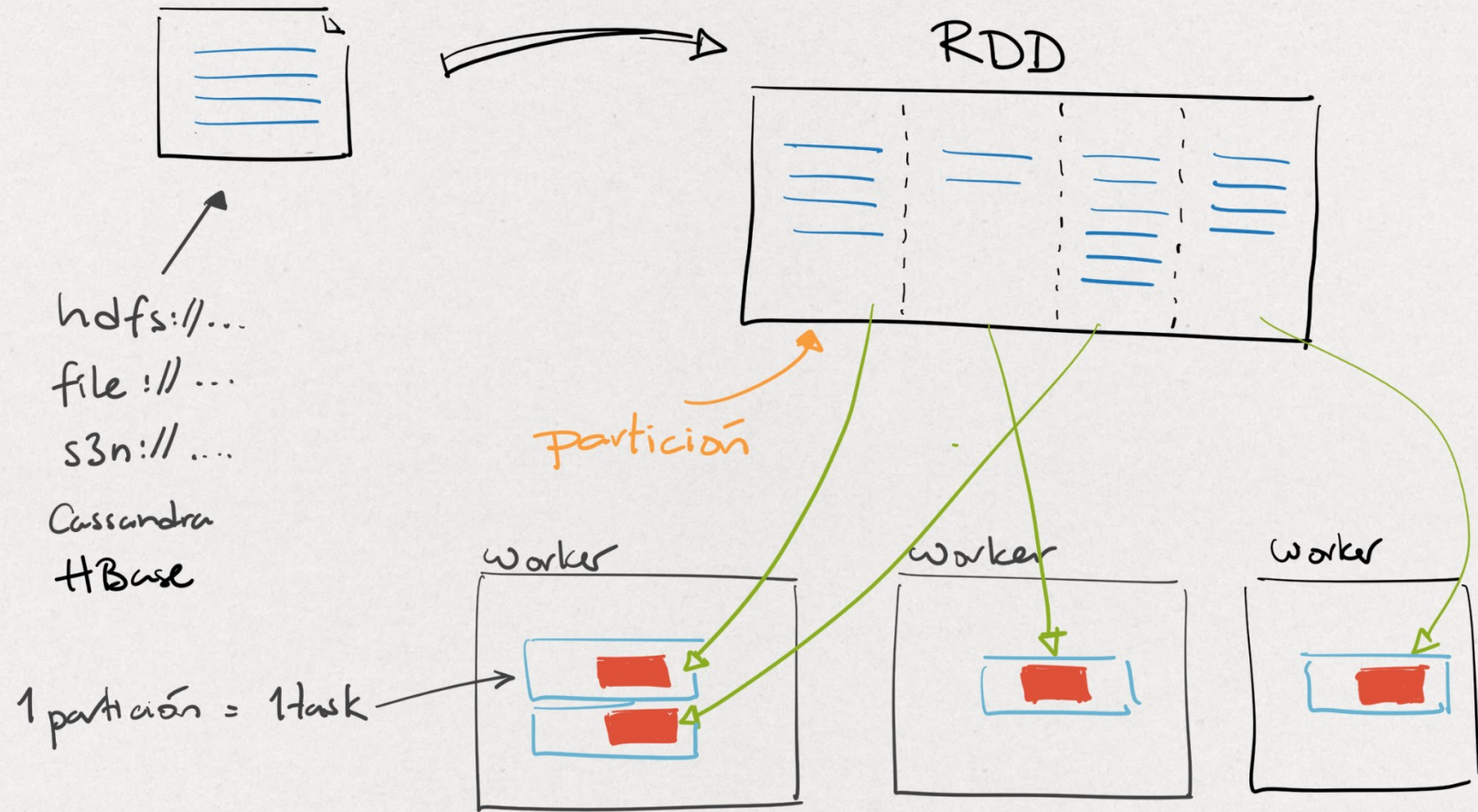
# Spark RDD: Ventajas

- Pueden ser reconstruidas gracias a su *linage*.
- Pueden ser manipuladas en paralelo.
- Están *cached* en memoria para su uso inmediato.
- Son almacenadas de manera distribuida.
- Contienen cualquier tipo de dato, incluidos los definidos por el programador.

# Spark RDD: Operaciones

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

sc.textFile(...)



# Spark RDD: Operaciones

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

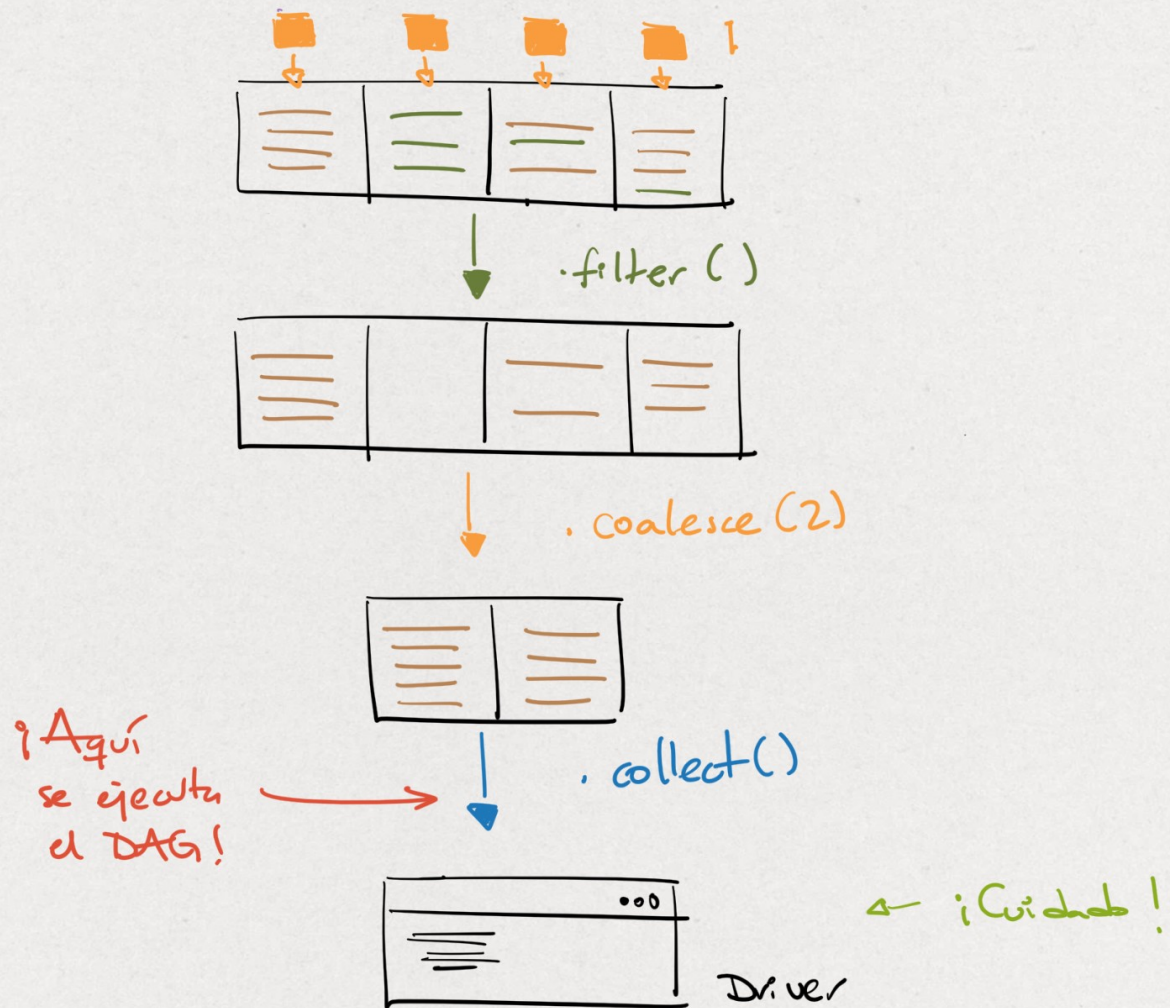
# Spark RDD

**Demo**

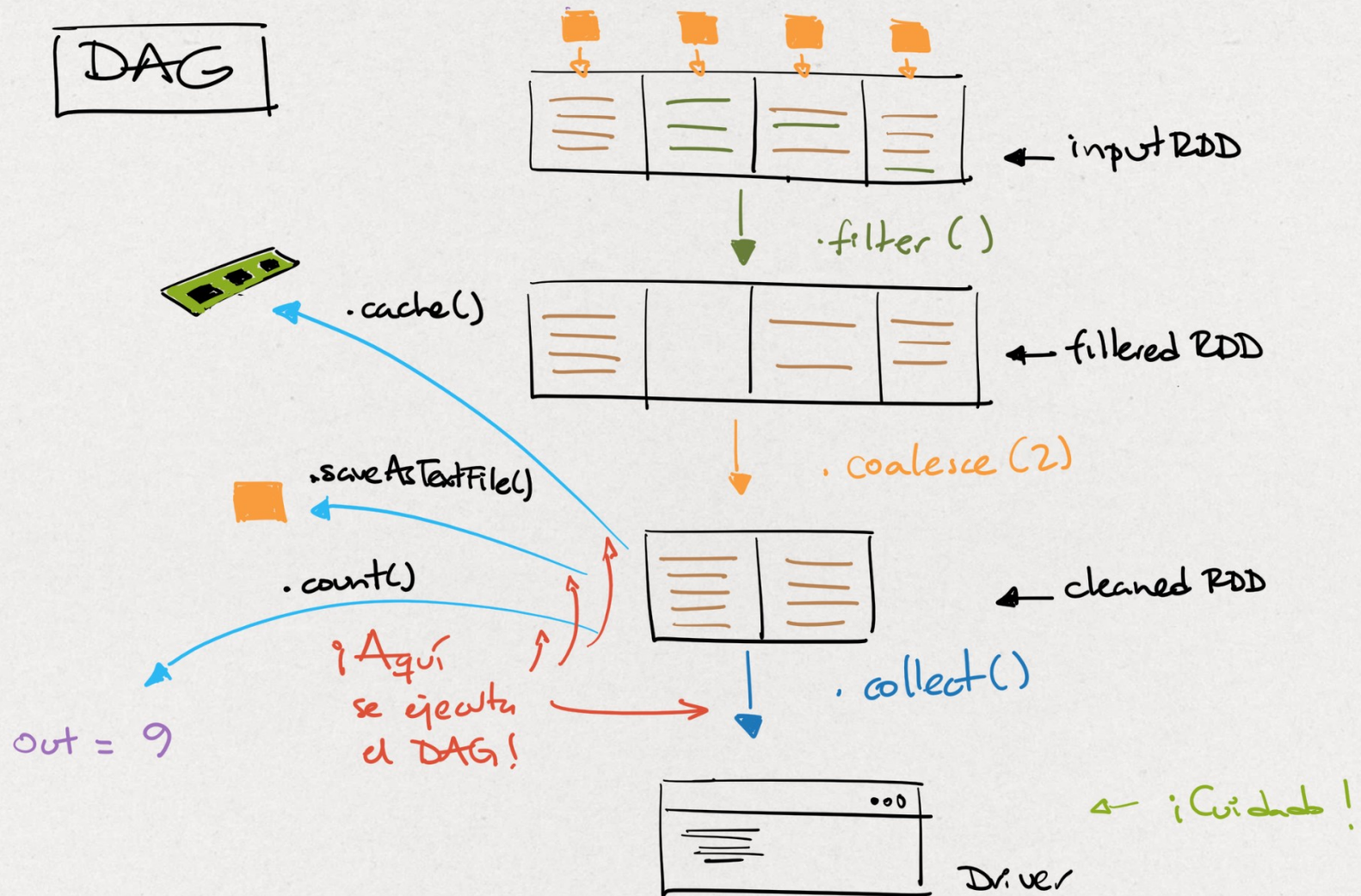


# Operaciones

DAG



DAG



# Transformaciones

**Demo**

# Acciones

## **Demo**

# Spark DataFrames

# Spark SQL

- Agregado recientemente (2015) a Apache Spark
- Ejecuta *queries* SQL o HiveQL.
- Agrega una capa de abstracción encima de los RDDs al convertirlos en **Data Frames**, análogos a los usados en R y Pandas.
- Define un `sqlContext` análogo a `sc`

# Spark SQL

- Unifica la interfaz para escribir/leer en varios formatos

- ```
df = sqlContext.read \  
    .format("json") \  
    .load("...")
```
- ```
df = sqlContext.write \  
    .format("parquet") \  
    .mode("append") \  
    .saveAsTable("...")
```



# Spark SQL

- Seleccionar columnas y filtrar
- Realizar `joins`
- Agregaciones
- Funciones analíticas
- Graficación
  - usando `pandas` ó `ggplot`

# Spark DataFrames

- Soporta una gran cantidad de formatos de datos y sistemas de almacenamiento.
  - Json, parquet, avro, csv, etc.
- Optimización con Spark SQL Catalyst
  - De hecho son más eficientes que los RDD
- APIs en Python, Java, Scala, R.
- Integrados con toda la infraestructura de Spark.
  - ML, *Streaming*, GraphX, etc.

# Spark DataFrames

- Al estar construidas sobre Spark RDD, puedes usar operaciones de RDD sobre un DataFrame, pero no es recomendable, ya que obtendrías un RDD de regreso.
- Los DataFrames (debido al optimizador Catalyst) son mucho más rápidos que los RDDs, además de que esa velocidad ganada es igual en todos los lenguajes.
  - Esto no sucede con los RDDs, un RDD en Scala es más rápido que un RDD en Python.

# Spark DataFrames

- Por *default* Spark utiliza el `SQLContext` nativo, pero, si estás conectado a un *cluster* con Apache Hive, puedes utilizar el `HiveContext` en su lugar.
- Esto te permitirá usar todo el `HiveQL`, sus `udf`'s y escribir/leer a tablas en Hive.
- En este taller **NO** tenemos un `HiveContext`
  - :

# Spark DataFrames

- Los DataFrames tienen esquema.
  - Se puede ver con `printSchema()`.
  - Spark se encarga de extraerlo (Parquet, tabla) o inferirlo (json).
- Un DataFrame Column es una abstracción, ya que no todas las fuentes están en formato columnar.
- A pesar de la sintaxis, es importante recordar que una Column **NO** es igual a un DataFrame

# Spark DataFrames

- Al igual que los `RDDs`, los `DataFrames` pueden ser operados mediante transformaciones y acciones.
- Las *Transformaciones* son *lazy* pero contribuyen a la planeación de la ejecución del query, las **Acciones** provocan la ejecución del query.
- Esto último significa que al ejecutarse la acción, Spark lee el data source y los datos fluyen a través del **DAG** generado por el optimizador, al concluir el resultado de la acción se despliega en el `Driver`.

# Spark DataFrame

**Demo**

Spark ML



# Spark ML

- Apareció en Spark 1.2
  - `spark.ml`
- Su objetivo es estandarizar una API para algoritmos de aprendizaje de máquina, de tal manera que sea más fácil combinarlos en un *pipeline* o *workflow*.
- Usa Spark DataFrames
  - **IMPORTANTE:** `spark.mllib` usa RDDs, `spark.ml` usa DataFrames.
  - No veremos `spark.mllib`

# Spark ML: Transformers

- Es un algoritmo que transforma un DataFrame en otro DataFrame, generalmente agregando una o más columnas.
- Su método más importante es `transform()`
- Incluye *features transformers* y *learning models*.

# Spark ML: Transformers

## Feature Transformer

- Toma un dataset
- Lee una columna
  - texto
- La convierte en una nueva columna de *features* y la agrega al dataset
  - vectorización del texto
- Devuelve un dataset nuevo

## Learning model

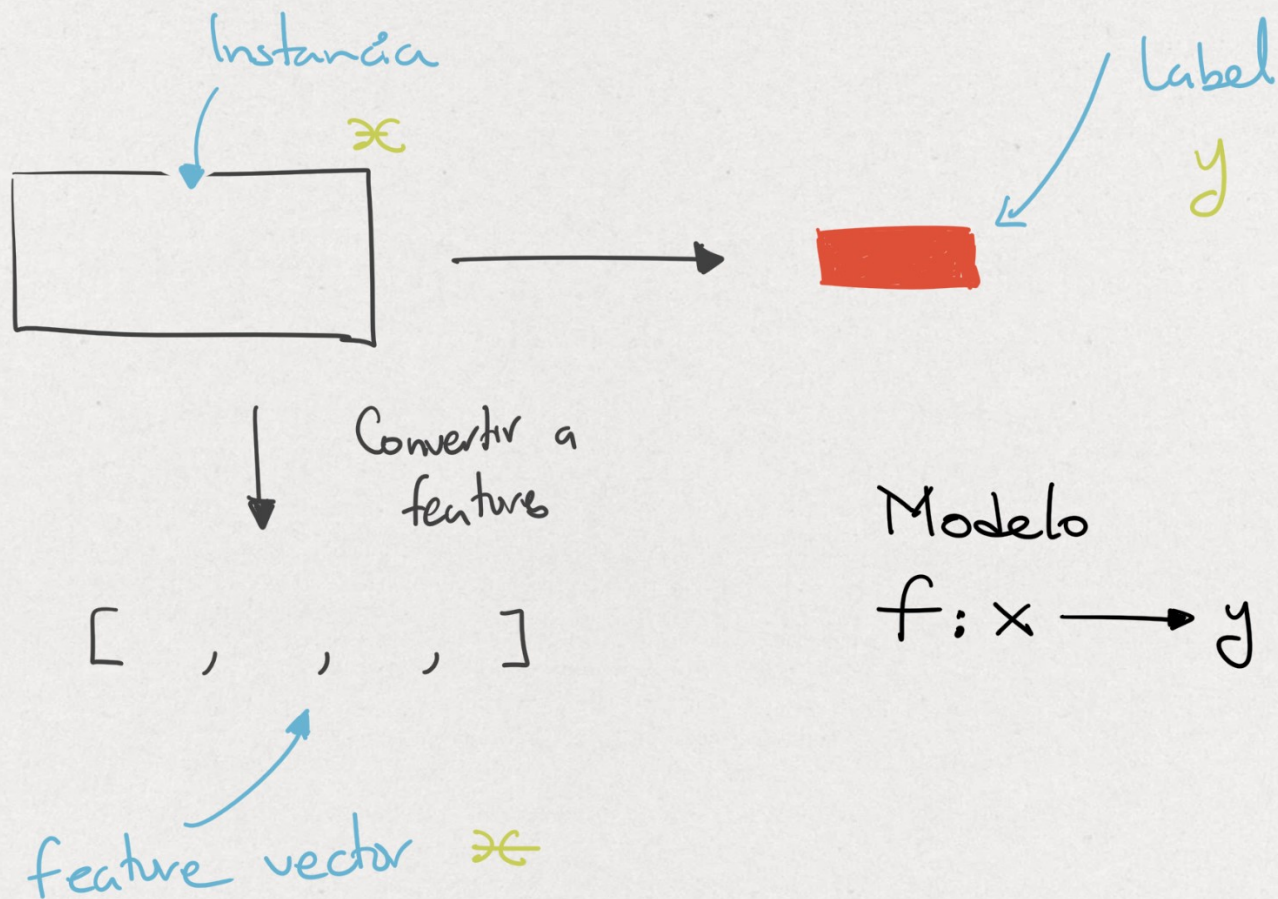
- Toma un dataset
- Lee la columna que contiene los *features*
- Predice las etiquetas para cada vector de *features*
- Agrega esa columna de etiquetas
- Devuelve un dataset nuevo

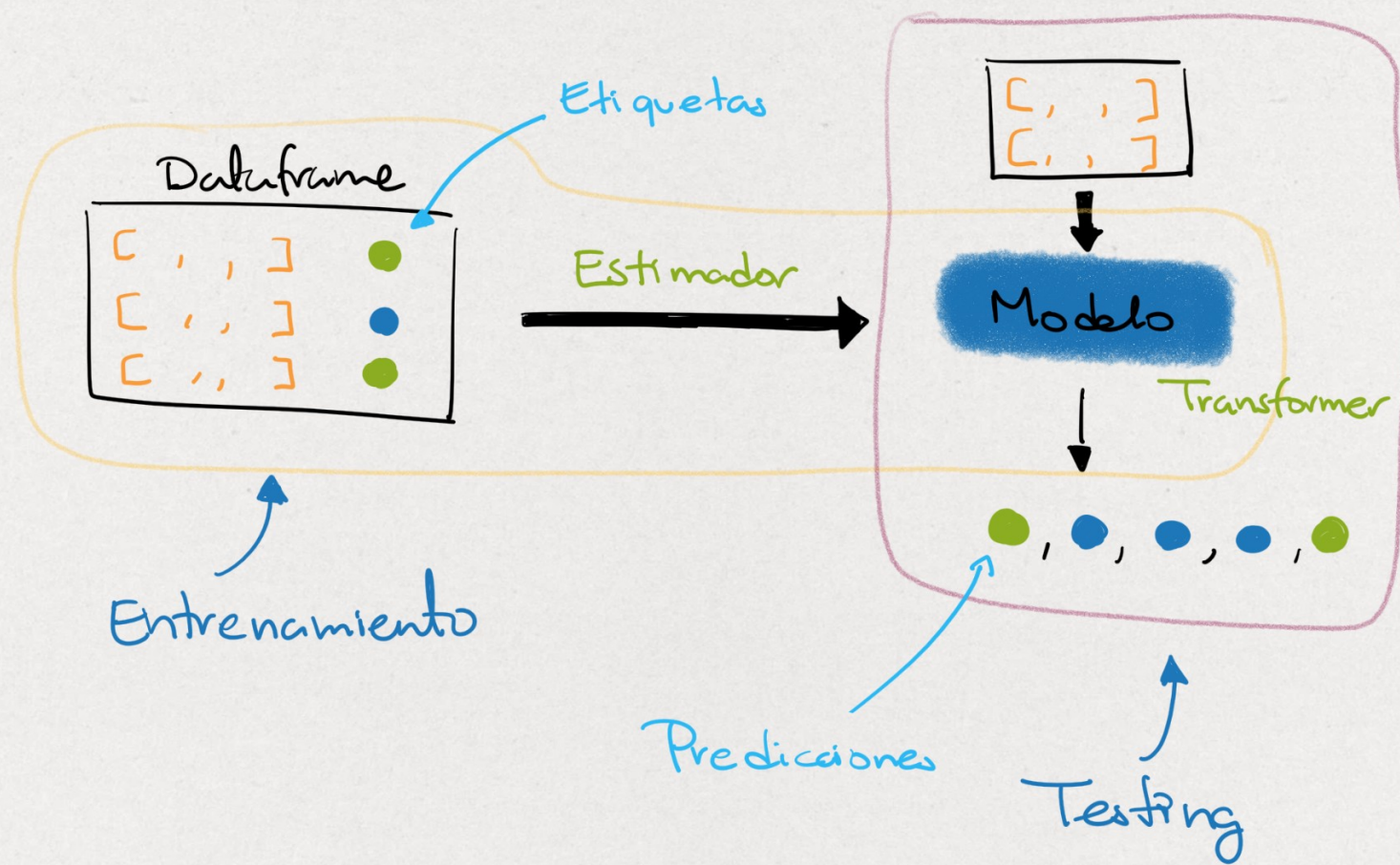
# Spark ML: Estimators

- Es un algoritmo que puede ser ajustado en un DataFrame para producir un **Transformer**.
- Un algoritmo de aprendizaje de máquina es un **Estimator** que entrena en el dataset para obtener un modelo.
- Implementan el método `fit()`.
- La regresión logística es un Estimator e invocando `fit()` entrena un modelo de regresión logística que es un Transformer.

# Spark ML: Pipeline

- En aprendizaje de máquina, es común ejecutar los algoritmos en una secuencia para procesar y aprender de los datos.
- Spark ML abstraer este *workflow* en un Pipeline, el cual consiste en una secuencia de Transformers y Estimators.





Spark ML: Workflow

# Spark ML

## Demo



# Spark Streaming

# Spark Streaming

**Demo**

# Spark GraphX

# Spark GraphX

**Demo**