

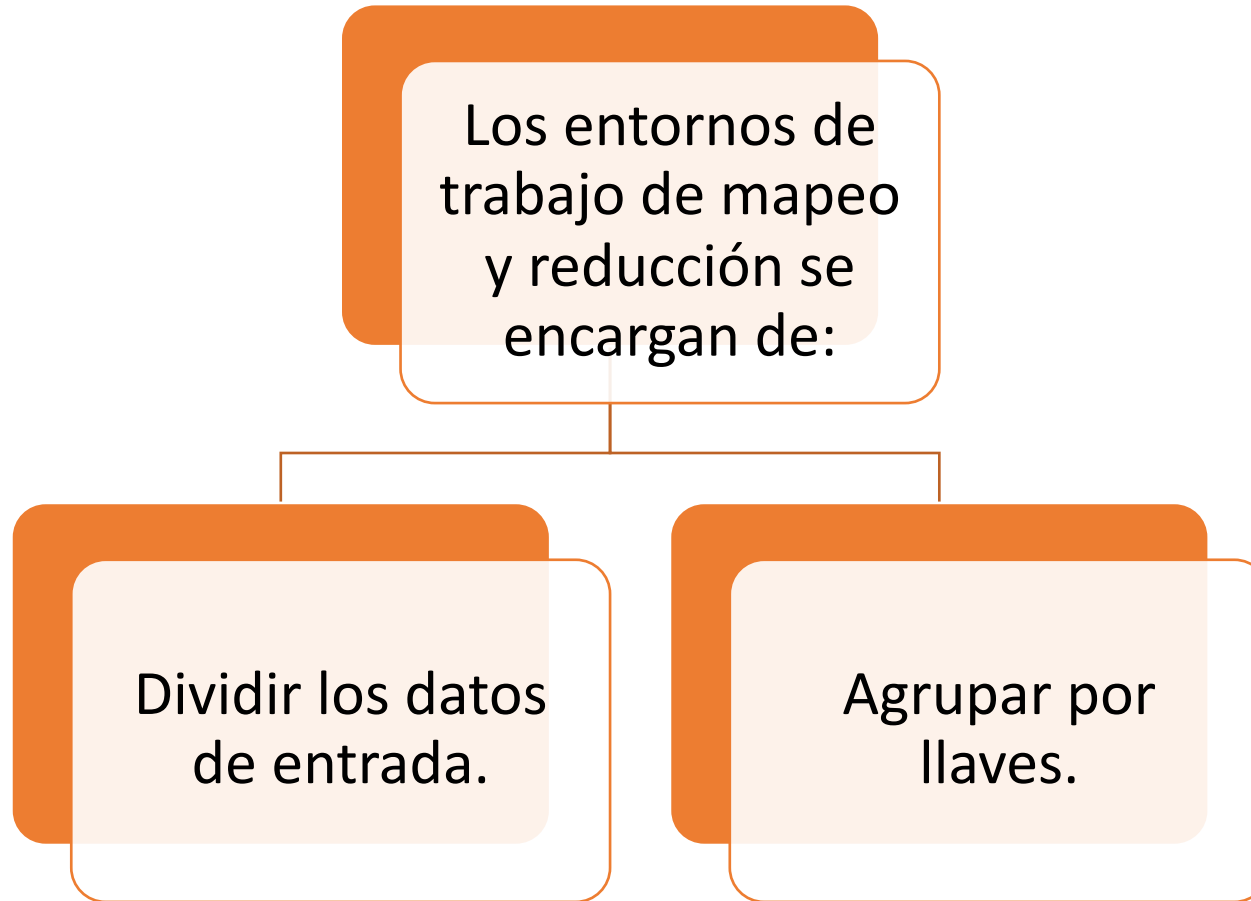
DATOS MASIVOS I

UNIDAD II MODELO DE MAPEO Y REDUCCIÓN

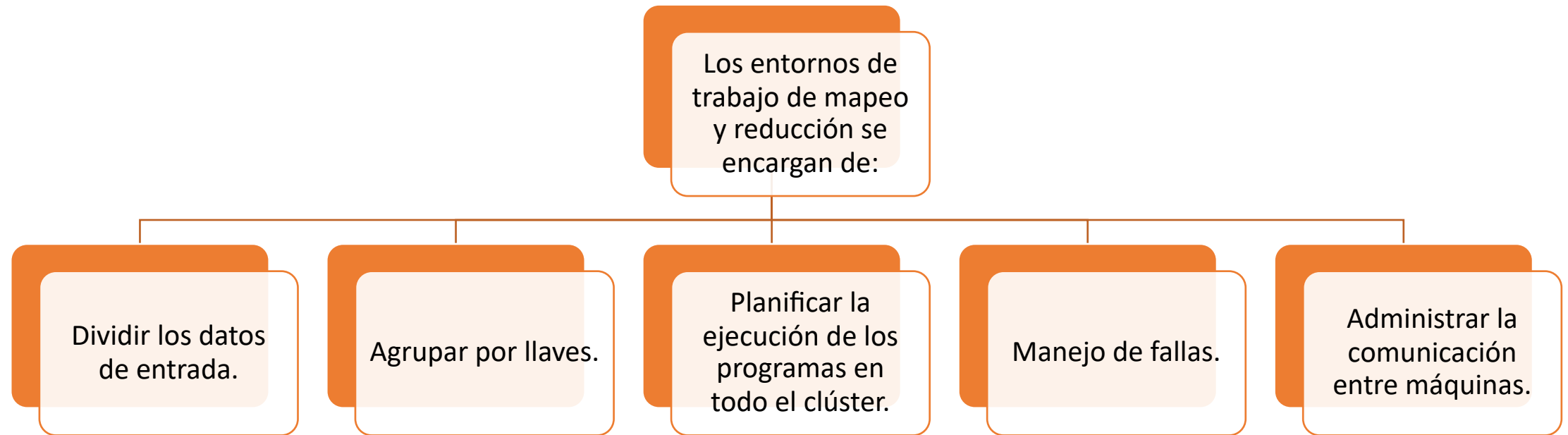
EXTENSIONES DE MAP – REDUCE

17 de Febrero de 2023

Entorno de Trabajo



Entorno de Trabajo



Map – Reduce en Paralelo

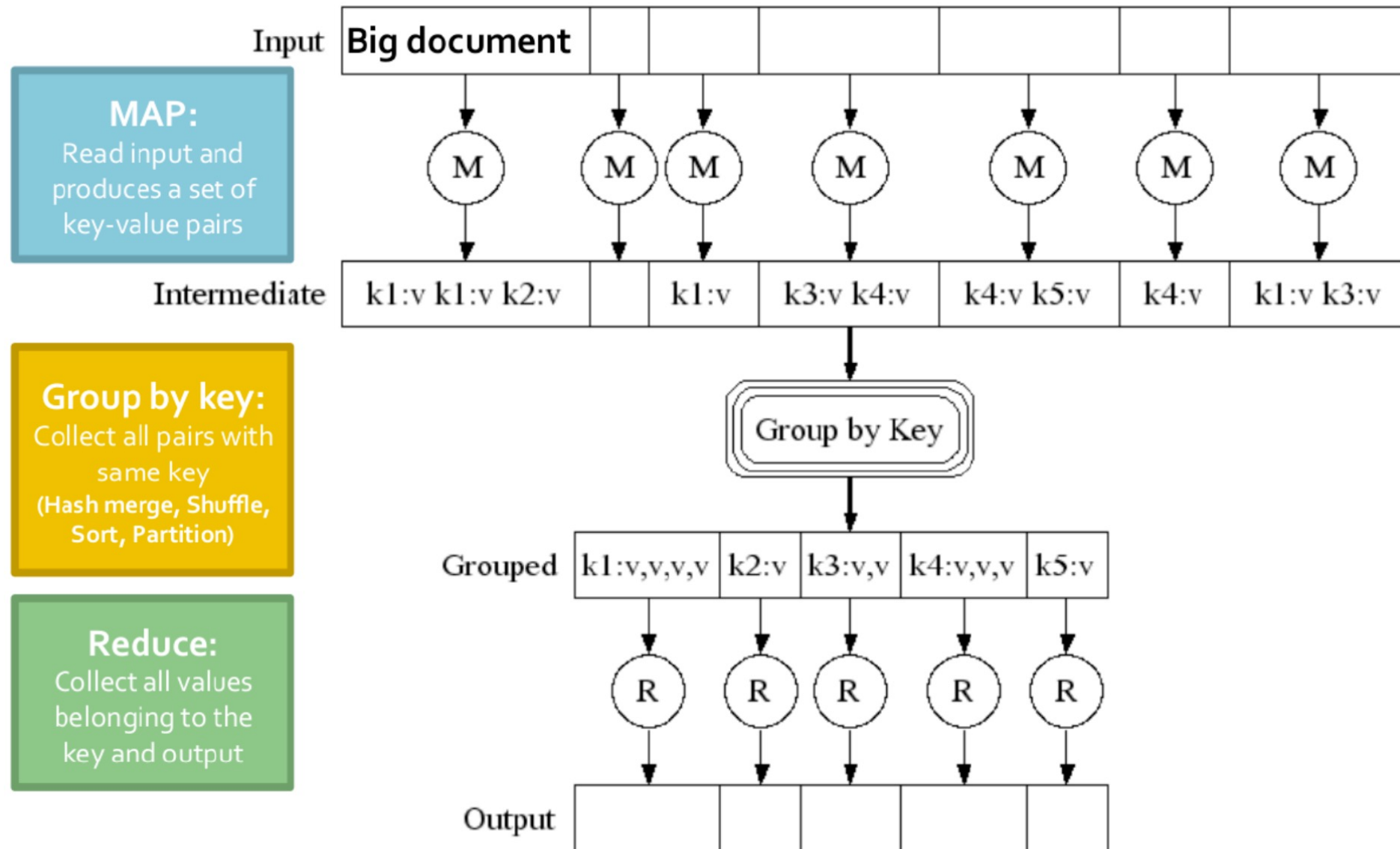


Imagen tomada de J. Leskovec, A. Rajaraman, J. Ullman.

Se encarga de la coordinación de las actividades en el clúster de cómputo.

- Planifica las tareas de mapeo y reducción.

Se encarga de la coordinación de las actividades en el clúster de cómputo.

- Planifica las tareas de mapeo y reducción.
- Para detectar fallas, el nodo maestro periódicamente se comunica con demás nodos.
- Estatus de las tareas / nodos: esperando, en progreso, completada.

Se encarga de la coordinación de las actividades en el clúster de cómputo.

- Cuando una tarea de mapeo está completa, el nodo maestro envía la localización y tamaño de los archivos intermedios a los nodos de reducción.

Flujo de Trabajo del Modelo Map – Reduce



Las entradas y salidas son almacenadas en el sistema de archivo distribuido (DFS).



Las tareas de mapeo son planificadas en nodos cercanos a la localización física de los datos.

Flujo de Trabajo del Modelo Map – Reduce

Los resultados intermedios son almacenados en sistemas locales de archivos (nodos de mapeo y reducción). Con el objetivo de:

Flujo de Trabajo del Modelo Map – Reduce

Los resultados intermedios son almacenados en sistemas locales de archivos (nodos de mapeo y reducción). Con el objetivo de:



Evitar tráfico en la red.



Sobrecargar de datos.

Manejo de Fallas



CUANDO FALLA EL NODO DE
MAPEO:



LAS TAREAS DE MAPEO
COMPLETADAS O EN
PROGRESO SE VUELVEN A
PROGRAMAR PARA SU
EJECUCIÓN.



LOS NODOS DE REDUCCIÓN
SON NOTIFICADOS
INDICANDO QUE LA TAREA
DE MAPEO FUE
REPROGRAMADA.

Cuando falla el nodo de reducción.



Las tareas en progreso son reprogramadas hacia otro nodo.

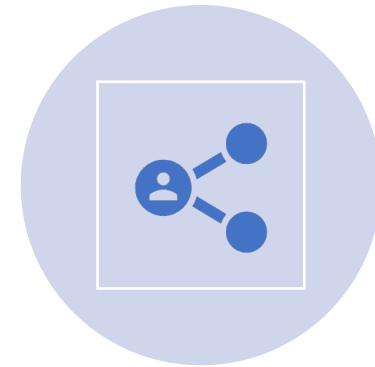
Manejo de Fallas



CUANDO FALLA EL NODO DE
MAESTRO.



TODAS LAS TAREAS DE
MAPEO Y REDUCCIÓN SON
ABORTADAS / CANCELADAS.



EL CLIENTE ES NOTIFICADO.

Flujo de Trabajo del Modelo Map – Reduce

- Las salidas de los reducer son frecuentemente la entrada hacia otra tarea de mapeo.

¿Cuántas Tareas de Map – Reduce se deben Planificar?

Objetivo. identificar M (número de tareas de mapeo) y R (número de tareas de reducción) que deben ser planificadas.

- El número de tareas M usualmente es más grande que el número de nodos en el clúster.

¿Cuántas Tareas de Map – Reduce se deben Planificar?



Usualmente R es más pequeño que M .



El archivo de salida se reparte en los R nodos.

¿Cuántas Tareas de Map – Reduce se deben Planificar?



Usualmente R es más pequeño que M .



El archivo de salida se reparte en los R nodos.

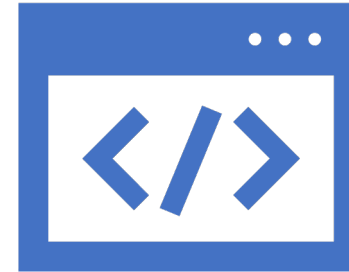


Mejora el balanceo de cargas y acelera la recuperación en caso de fallas.

Problemas con el Modelo Map – Reduce



Dificultad para programar
directamente.



Muchos algoritmos no se
describen fácilmente con
funciones de mapeo y
reducción.

- Cuellos de botella.
 - Incurre en gastos considerables debido a la replicación de datos, E/S de disco y serialización.

- Cuellos de botella.

- Incurre en gastos considerables debido a la replicación de datos, E/S de disco y serialización.
- Para preservar la persistencia de los datos, el tiempo que se toma en dividir y almacenar los datos es considerable.

Extensiones



BASADAS EN UN SISTEMA DE
ARCHIVOS DISTRIBUIDO.



PROCESAMIENTO SE REALIZA DE
FORMA DISTRIBUIDA CON
MUCHAS TAREAS REPETITIVAS
QUE SON INSTANCIAS DE
FUNCIONES DEFINIDAS POR EL
USUARIO.



INCORPORAN ESTRATEGIAS DE
MANEJO DE FALLAS.

Sistemas de Flujo de Trabajo



El cómputo se expresa como un grafo acíclico que representa el flujo de trabajo de un conjunto de funciones.



Que un vértice a se conecte a otro vértice b , representa que la salida de la función a es la entrada de la función b .

Sistemas de Flujo de Trabajo



Que un vértice a se conecte a otro vértice b , representa que la salida de la función a es la entrada de la función b .



El modelo Map – Reduce es un sistema de flujo de trabajo con dos pasos.

Sistemas de Flujo de Trabajo



Los datos fluyen de una función a otra y cada función se puede realizar en múltiples tareas con distintas partes de los datos de entrada.

Sistemas de Flujo de Trabajo



Un nodo maestro se encarga de dividir las tareas en distintos nodos y resolver posibles fallas.



Los datos fluyen de una función a otra y cada función se puede realizar en múltiples tareas con distintas partes de los datos de entrada.



Apache SPARK extiende el modelo de programación Map – Reduce.

Extensión del modelo Map – Reduce.

- Se basa en un sistema de operaciones (transformaciones – acciones) realizadas sobre colecciones de datos distribuidos (RDD).
- Actualmente, es el sistema más popular de flujo de datos.

Más rápido.

- Evita guardar resultados intermedios en disco.
- Activa la caché de datos para consultas repetitivas.

Apache SPARK

Presenta
funciones extras
(más allá de Map
– Reduce).

Compatible con
Apache Hadoop.

Es código abierto
(Apache
Foundation).

Apache SPARK



Soporta Java, Scala y Python.



Principal contribución: Conjunto de datos distribuidos y resilientes (RDD).



Integra *APIs* de alto nivel.

En las versiones más recientes de Spark incluye Dataframes y Datasets. Ofrece *APIs* para agregar datos, lo cual permite soportar SQL.

Conjunto de Datos (Dataset)

Puede contener cualquier tipo de información.

Diagram illustrating the structure of a dataset (Conjunto de Datos) with columns and rows.

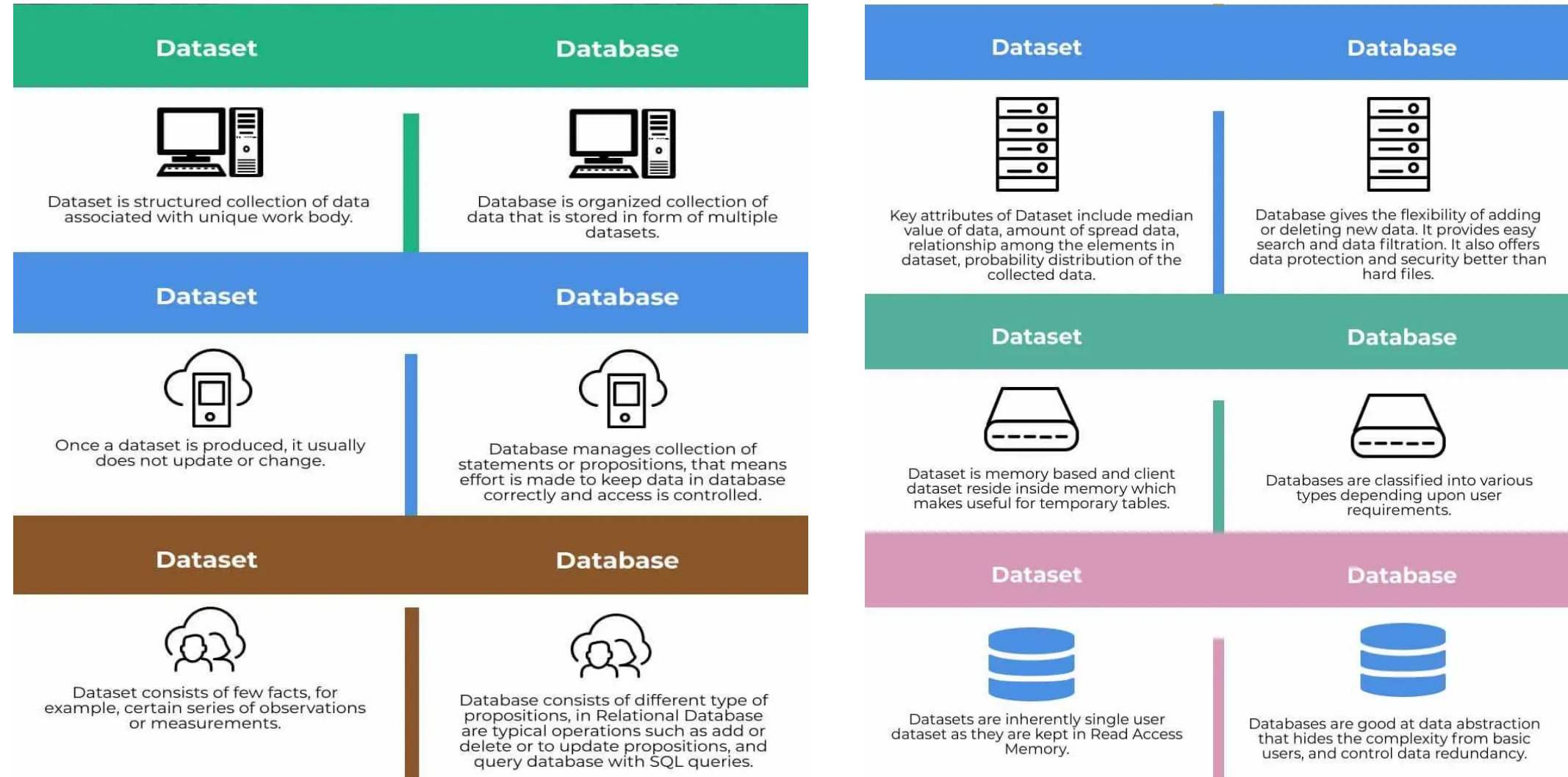
Columns: Name, Team, Number, Position, Age

Rows (Filas): 0 to 6

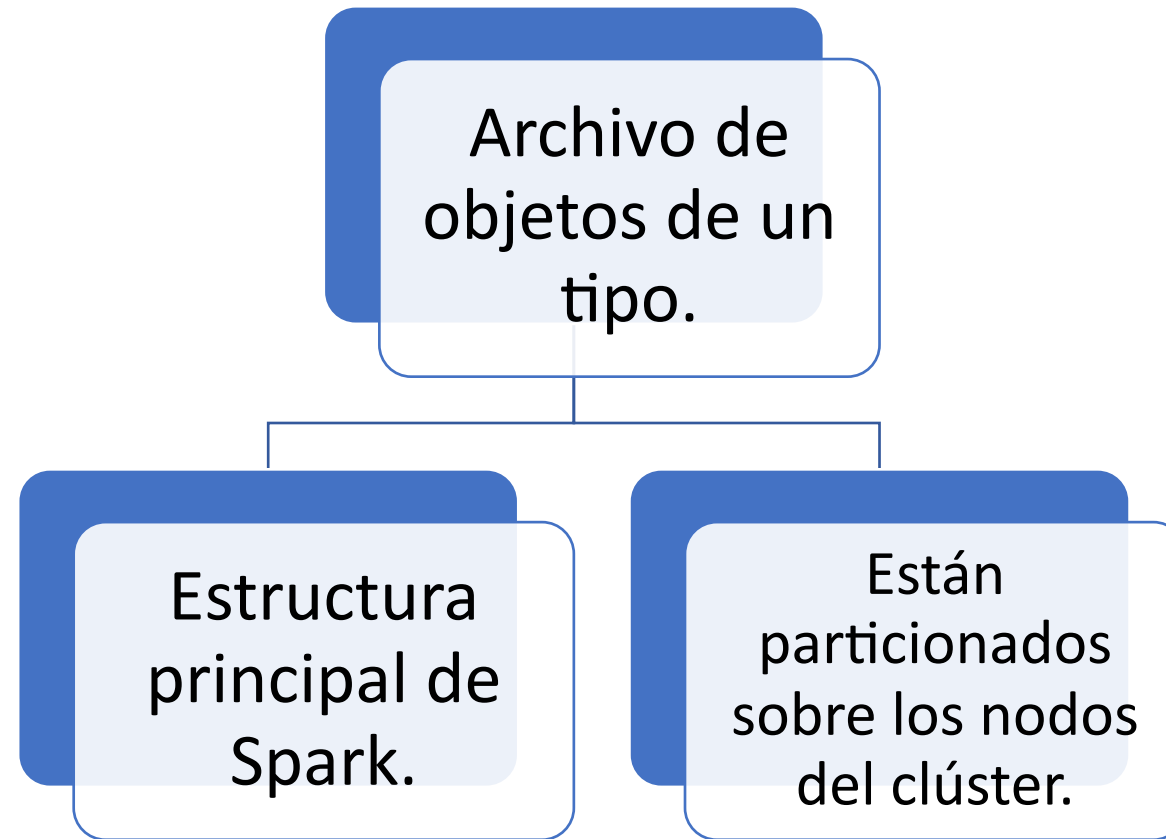
	<i>Name</i>	<i>Team</i>	<i>Number</i>	<i>Position</i>	<i>Age</i>
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Conjunto de Datos (Dataset)

Puede contener cualquier tipo de información.



Resilient Distributed Dataset (RDD)



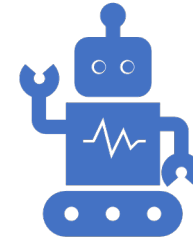
Resilient Distributed Dataset (RDD)



Son inmutables: cuando transformamos un *RDD*, realmente estamos creando uno nuevo.



Tolerante a fallos.



Pueden ser creados desde Hadoop.

¿Cómo se Crea un RDD?

Existen dos formas comunes para crear un *RDD*:

A través del objeto *SparkContext*.

A partir de conjuntos de datos externos.

Creando un RDD usando *SparkContext*

El método *SparkContext.parallelize* nos permite crear un *RDD* a partir de una lista o tupla:

```
lista = ['en', 'un', 'lugar', 'de', 'la', 'mancha']  
listardd = sc.parallelize(lista, 4)
```

- **sc** es una instancia de la clase *SparkContext*.

Creando un RDD usando *SparkContext*

- La lista se pasa como argumento a **sc** y se paralelizará automáticamente por *Spark*.
- El programador puede decidir en cuántas partes debe paralelizarse un *RDD* (por ejemplo 4).

Creando un RDD usando Conjuntos de Datos Externos

A partir de una fuente de almacenamiento utilizando la función `textFile` del `SparkContext`:

```
texto = sc.textFile("loremipsum.txt")
```

- Como argumento se pasa un archivo (texto) almacenado en disco.
- El método *textFile* cargaría el archivo como un *RDD*.

- ✓ Los RDDs no son valiosos solamente por los datos que contienen, sino por las operaciones que podemos realizar sobre ellos.

- ✓ Spark proporciona un conjunto de acciones para procesar y extraer información:
 - **collect ()**
 - **reduce()**
 - **count()**
 - **first**
 - **foreach()**

Acción: *collect()*

collect() retorna todos los elementos de un *RDD*.

```
rdd = sc.parallelize([4, 1, 2, 6, 1, 5, 3, 3, 2])
```

```
lista = rdd.collect()
```

```
print (“El tercer elemento de la lista es%d”%  
lista[2])
```

>> El tercer elemento de la lista es ?

Importante: si el RDD es muy grande, se podría tener problemas al poner toda la colección en memoria.

Acción: *count()*

count() retorna el número elementos del *RDD*.

```
rdd = sc.parallelize([4, 1, 2, 6, 1, 5, 3, 3, 2])
```

```
print("El RDD contiene %d elementos" %  
rdd.count())
```

```
>> El RDD contiene ? elementos
```

Acción: *countByValue()*

countByValue() retorna un diccionario con el número de apariciones de cada elemento en un RDD.

```
rdd = sc.parallelize([4, 1, 2, 6, 1, 5, 3, 3, 2])
```

```
rdd.countByValue()
```

```
>> ?
```

Acción: *countByValue()*

countByValue() retorna un diccionario con el número de apariciones de cada elemento en un RDD.

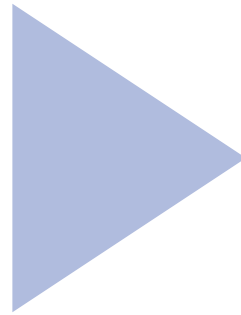
```
rdd = sc.parallelize([4, 1, 2, 6, 1, 5, 3, 3, 2])
```

```
rdd.countByValue()
```

```
>> 1:2, 2:2, 3:2, 4:1, 5:1, 6:1
```

Acción: *reduce()*

reduce(func)
agrega los
elementos de un
RDD según la
función que se le
pase como
parámetro.



La función debe
cumplir con
algunas
propiedades para
que pueda ser
calculada en
paralelo.

Conmutativa: $(A + B) = B + A$

- Asegurando que el resultado será independiente del orden de los elementos en el RDD.

Acción: *reduce()*

Conmutativa: $(A + B) = B + A$

- Asegurando que el resultado será independiente del orden de los elementos en el RDD.

Asociativa: $(A + B) + C = A + (B + C)$

- Asegurando que cualquiera de los dos elementos asociados en la agregación a la vez no afecta el resultado final.

Ejemplo: *reduce()*

Problema. Crear un *RDD* que multiplique por 2 sus valores y sumar los resultados:

```
rdd = sc.parallelize([1, 1, 1, 1, 2, 2, 2, 3, 3, 4])
```


Ejemplo: *reduce()*

Solución.

```
rddDos = rdd.map(lambda x: x*2)
```

```
finalSum = rddDos.reduce(lambda x, y: x + y)
```

```
print (finalSum)
```

```
>> 40
```

Ejemplo: *reduceByKey()*

Problema.

Crear un diccionario con elementos $(x, 1)$ y sumar las apariciones por elemento.

Ejemplo: *reduceByKey()*

```
rddText = sc.parallelize([ 'red', 'red', 'blue',  
'green', 'green', 'yellow' ])
```

```
rddAuxiliar = rddText.map( lambda x: (x, 1))
```

```
rddResult = rddAuxiliar.reduceByKey( lambda  
x, y: x + y)
```

```
print (rddResult.collect())
```

```
>> ?
```

Ejemplo: *reduceByKey()*

```
rddText = sc.parallelize([ 'red', 'red', 'blue',  
'green', 'green', 'yellow' ])
```

```
rddAuxiliar = rddText.map( lambda x: (x, 1))
```

```
rddResult = rddAuxiliar.reduceByKey( lambda  
x, y: x + y)
```

```
print (rddResult.collect())
```

```
>> [( 'blue', 1), ( 'green', 2), ( 'yellow', 1), ( 'red',  
2)]
```

Ejemplo: *foreach()*

foreach() ejecuta la función que se le pasa por parámetro sobre cada elemento del RDD.

Ejemplo: *foreach()*

```
rdd = sc.parallelize([ 4, 1, 2, 6, 1, 5, 3, 3, 2, 4  
1])
```

```
def impar(x):  
    if x% 2 == 1:  
        print ("%d es impar"% x)
```

```
rdd.foreach(impar)
```

```
>> ?
```

Ejemplo: *foreach()*

```
rdd = sc.parallelize([ 4, 1, 2, 6, 1, 5, 3, 3, 2, 4 ])  
def impar(x):  
    if x% 2 == 1:  
        print ("%d es impar"% x)  
rdd.foreach(impar)
```

```
>> 1 es impar  
    1 es impar  
    5 es impar  
    3 es impar  
    3 es impar
```

Ejemplo: *saveAsTextFile()*

saveAsTextFile(directorio) guarda el *RDD* como un conjunto de archivos de texto dentro del directorio.

Ejemplo: *collectAsMap()*

collectAsMap() retorna los elementos de un RDD clave/valor como un diccionario de python.

```
sc.parallelize([( 'a', 'b' ),( 'c', 'd' )]).collectAsMap()
```

```
>> ?
```

Ejemplo: *collectAsMap()*

collectAsMap() retorna los elementos de un RDD clave/valor como un diccionario de python.

```
sc.parallelize ([('a', 'b'), ('c', 'd')]).collectAsMap()
```

```
>> 'a': 'b', 'c': 'd'
```

Ejemplo: Otras Funciones

Función	Valor que retorna
<i>first()</i>	Devuelve el primer valor del <i>RDD</i> .
<i>mean()</i>	Devuelve el valor medio.
<i>variance()</i>	Devuelve la varianza.
<i>stdev()</i>	Devuelve la desviación estándar.
<i>take(n)</i>	Devuelve una lista con los <i>n</i> elementos del <i>RDD</i> .

No siempre se podrá ejecutar las acciones directamente sobre un RDD, debido a las siguientes razones:

- El *RDD* podría tener más datos de los necesarios a analizar.
- El *RDD* podría no contener todos los datos necesarios.

No siempre se podrá ejecutar las acciones directamente sobre un RDD, debido a las siguientes razones:

- El *RDD* podría tener más datos de los necesarios a analizar.
- El *RDD* podría no contener todos los datos necesarios.
- El *RDD* podría no estar en un formato adecuado.

Transformaciones

Antes de realizar acciones sobre los *RDD*, primero deben realizarse transformaciones sobre los *RDDs*.

Para garantizar que cada *RDD* contenga los datos unificados, filtrados y formateados para evitar errores.

Transformaciones

Al aplicar una transformación sobre un RDD original,

regresará un nuevo *RDD*.


Las transformaciones no modifican el *RDD* original.

Transformaciones


Al aplicar una transformación sobre un RDD original, regresará un nuevo *RDD*.



Las transformaciones no modifican el *RDD* original.



Spark evalúa las transformaciones de manera 'perezosa' (*lazy evaluation*).



Transformaciones Más Comunes

Las transformaciones construyen *RDDs* a través de operaciones como las siguientes.

map()

filter()

distinct()

sample()

union()

Transformaciones: *map()*

map(func) retorna un nuevo RDD, resultado de pasar cada uno de los elementos de un *RDD* original como parámetro de la función.

```
rdd = sc.parallelize([ 4, 0, 2, 6, 1, 5, 3, 9, 7,  
8])
```

```
t1 = rdd.map(lambda x: x * 2)
```

```
t1.collect()
```

```
>> ?
```

Transformaciones: *map()*

map(func) retorna un nuevo RDD, resultado de pasar cada uno de los elementos de un *RDD* original como parámetro de la función.

```
rdd = sc.parallelize([ 4, 0, 2, 6, 1, 5, 3, 9, 7,  
8])
```

```
t1 = rdd.map(lambda x: x * 2)
```

```
t1.collect()
```

```
>> [ 8, 0, 4, 12, 2, 10, 6, 18, 14, 16]
```

Transformaciones: *filter()*

filter(func) retorna un nuevo *RDD* que contiene los elementos que cumplen la función.

```
num = sc.parallelize ([ 1, 2, 3, 4, 5, 6, 100,  
2000, 4000 ])
```

```
menor50 = num.filter(lambda x: x < 100)
```

```
menor50.collect()
```

```
>> ?
```

Transformaciones: *filter()*

filter(func) retorna un nuevo *RDD* que contiene los elementos que cumplen la función.

```
num = sc.parallelize ([ 1, 2, 3, 4, 5, 6, 100,  
2000, 4000 ])
```

```
menor50 = num.filter(lambda x: x < 100)
```

```
menor50.collect()
```

```
>> [ 1, 2, 3, 4, 5, 6]
```

Transformaciones: *distinct()*

distinct() retorna un nuevo *RDD* que contiene una sola copia de los diferentes elementos del *RDD*.

```
num = sc.parallelize([ 1, 2, 3, 4, 4, 3, 2, 5 ])
```

```
num.distinct().collect()
```

```
>> ?
```

Transformaciones: *distinct()*

distinct() retorna un nuevo *RDD* que contiene una sola copia de los diferentes elementos del *RDD*.

```
num = sc.parallelize([ 1, 2, 3, 4, 4, 3, 2, 5 ])
```

```
num.distinct().collect()
```

```
>> [ 4, 1, 5, 2, 3 ]
```

Transformaciones: *union()*

union() retorna un nuevo *RDD*, el cual contiene la unión de los elementos de un *RDD* y del *RDD* que se le pasa como argumento.

```
city1 = sc.parallelize([ 'Barcelona', 'Madrid',  
'Paris' ])
```

```
city2 = sc.parallelize([ 'Madrid', 'Londres',  
'Roma' ])
```


Transformaciones: *union()*

```
city1 = sc.parallelize([ 'Barcelona', 'Madrid',  
'Paris' ])
```

```
city2 = sc.parallelize([ 'Madrid', 'Londres',  
'Roma' ])
```

```
city1.union(city2).collect()
```

```
>> ?
```

Transformaciones: *union()*

```
city1 = sc.parallelize([ 'Barcelona', 'Madrid',  
'Paris' ])
```

```
city2 = sc.parallelize([ 'Madrid', 'Londres',  
'Roma' ])
```

```
city1.union(city2).collect()
```

```
>> [ 'Barcelona', 'Madrid', 'Paris', 'Madrid',  
'Londres', 'Roma' ]
```

Transformaciones: Otras Funciones

Función	Valor que retorna
<i>intersection()</i>	Devuelve la intersección de dos <i>RDDs</i> .
<i>keys()</i>	Devuelve únicamente las llaves del <i>RDD</i> .
<i>sortBy(func)</i>	Ordena un <i>RDD</i> según un criterio.

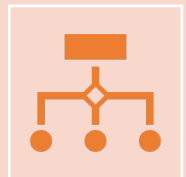
Características de un Grafo Acíclico Dirigido (DAG)



Cada tarea de *Spark* crea un DAG para que se ejecute en un clúster.

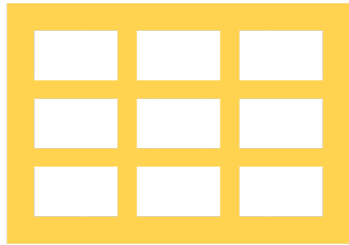


Los DAG pueden tener cualquier número de estados.

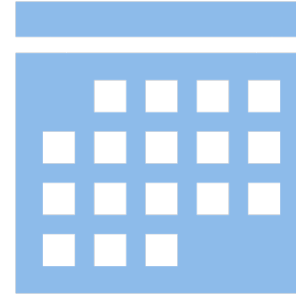


Map – Reduce tiene 2 estados predefinidos.

Características de un Grafo Acíclico Dirigido (DAG)

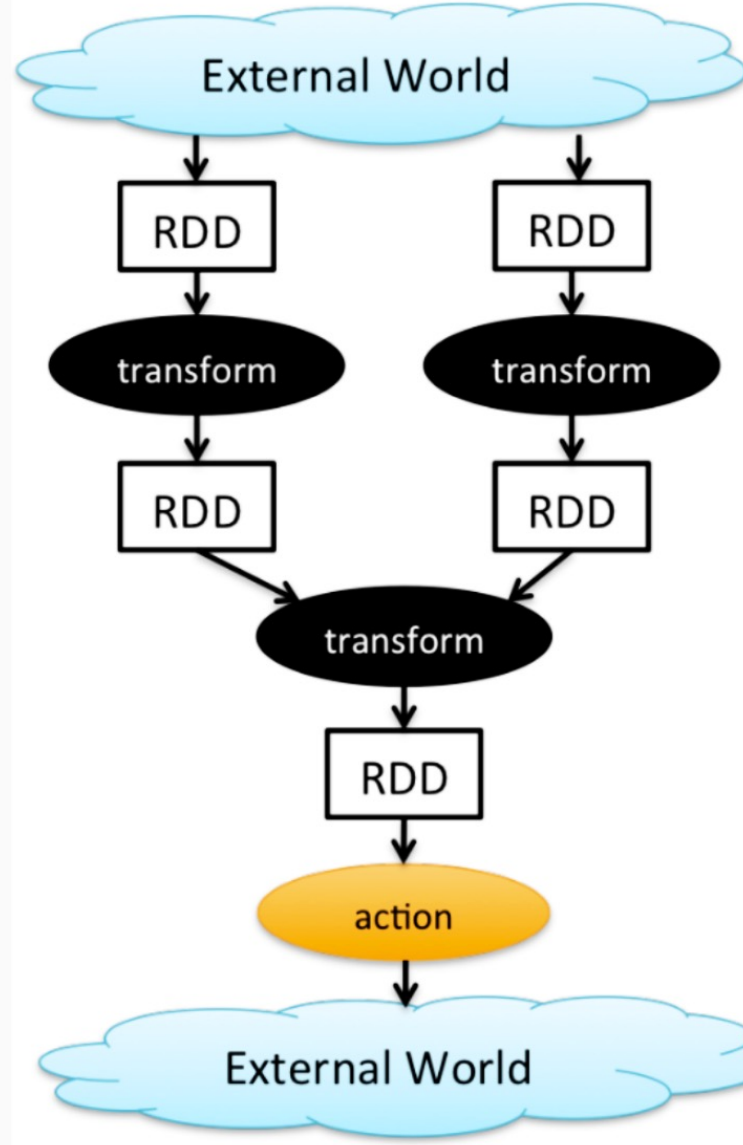


Caché de datos.

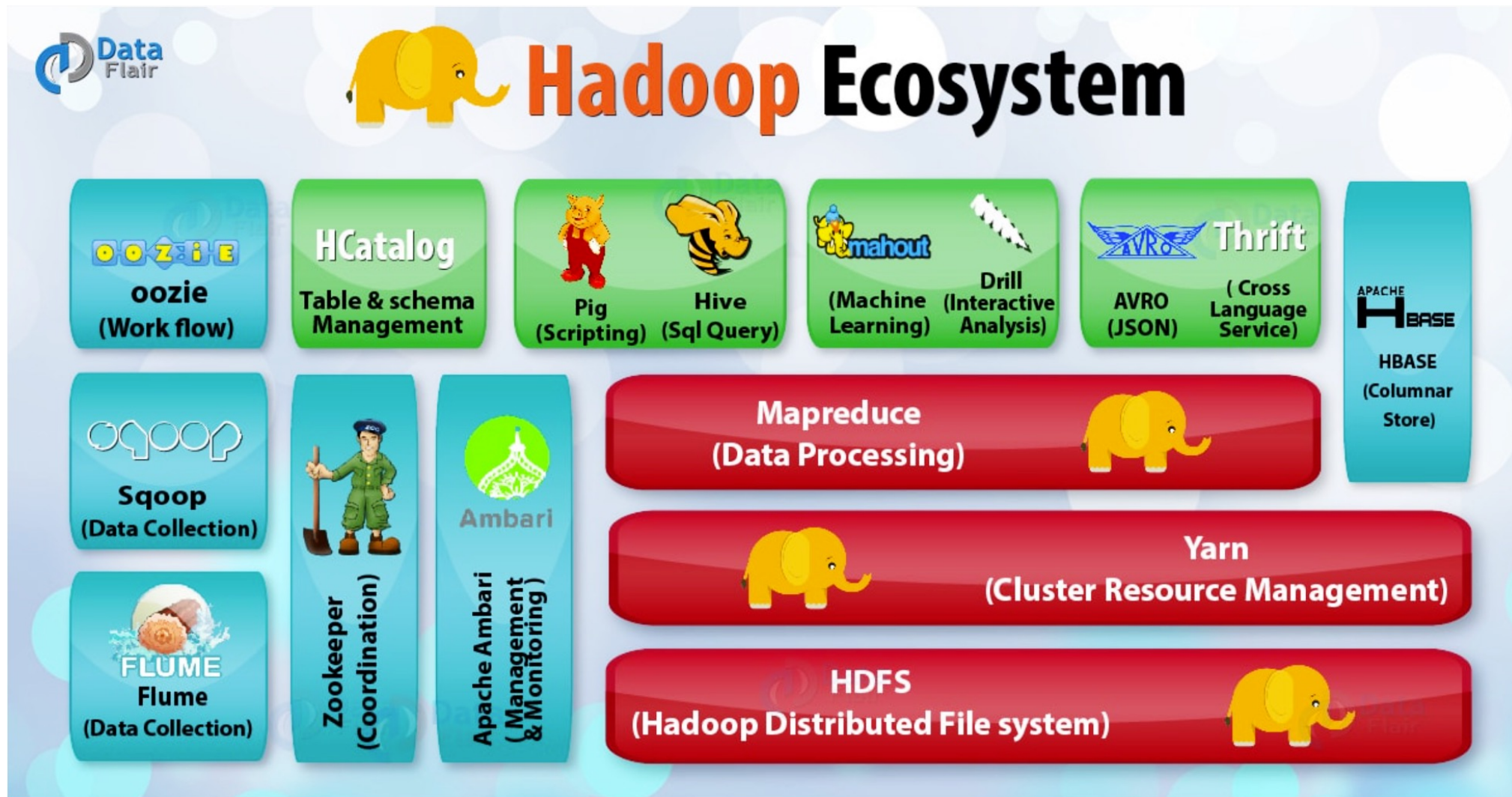


Los DAG permiten programar
hilos complejos de ejecución
en paralelo.

DAG



Componentes del Ecosistema HADOOP



Módulos de *Apache Spark*

