Formación senior en Spark & Scala

PARTE DE LA FORMACIÓN DE BIG DATA ACADEMY

FORMADOR: ALONSO MELGAREJO [alonsoraulmgs@gmail.com]

Sobre el profesor

Soy un profesional con 12 años de experiencia empresarial en el mundo de las TI y 8 años de experiencia en Big Data, Cloud & Machine Learning, en la parte académica soy investigador en temas de computación cuántica y asesor de tesis de Big Data & Machine Learning

ACTUALMENTE SOY

- Director de Big Data & Analytics en **Big Cloud Dev**
- Formador de cursos de Big Data en **Big Data Academy**

CONTACTO: alonsoraulmgs@gmail.com **LINKEDIN**: https://www.linkedin.com/in/alonsoraulmg/

Concepto

Spark

Es un **motor de procesamiento distribuido paralelo in-memory**. Proporciana **apis en Java, Scala, Python y R**. Spark mantiene la **escalabilidad lineal y la tolerancia a fallos** de MapReduce, pero amplía sus bondades gracias a varias funcionalidades.

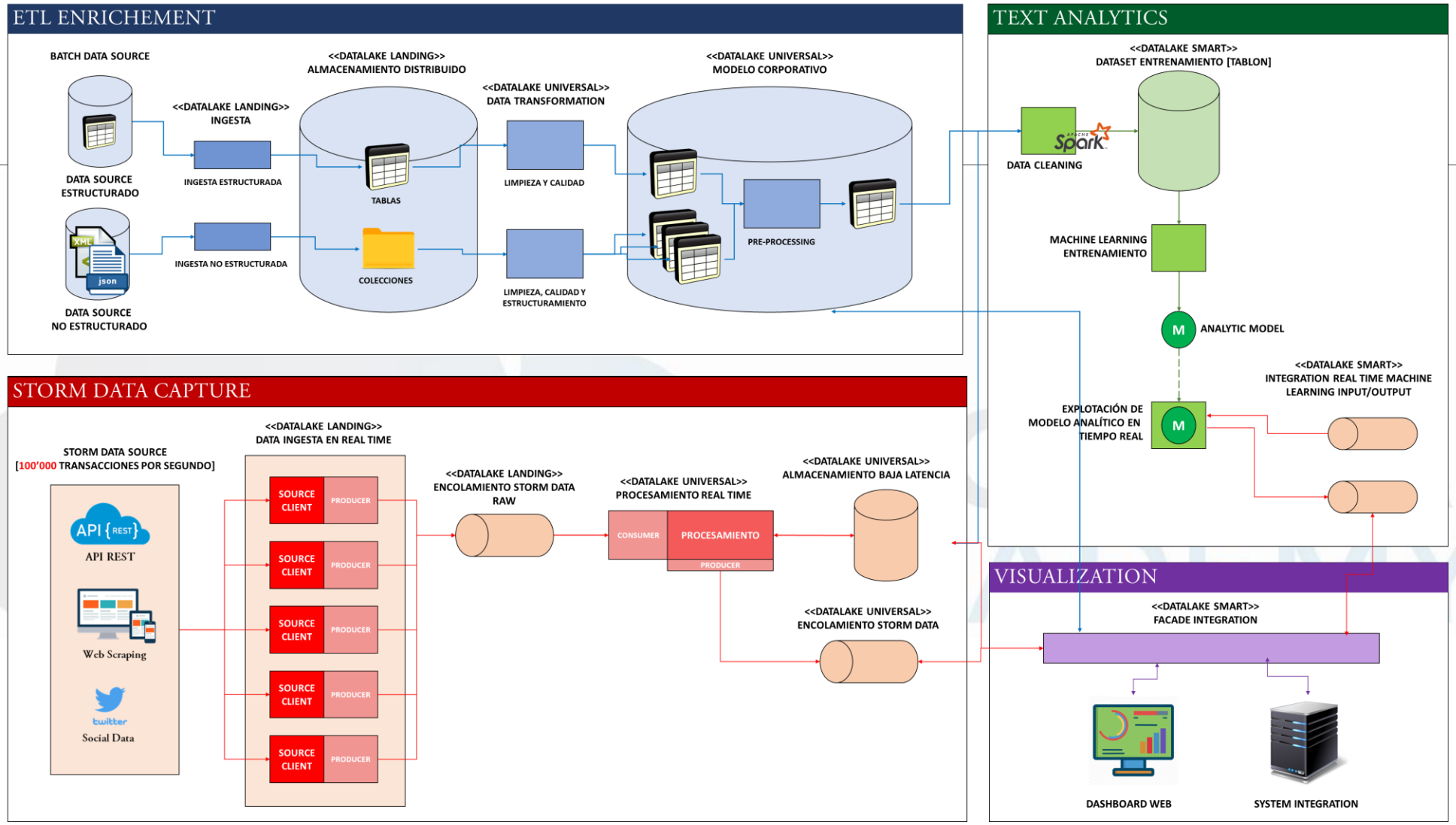
Objetivo fundamental

Objetivo fundamental de Spark

Ejecutar procesos **lo más rápido posible**

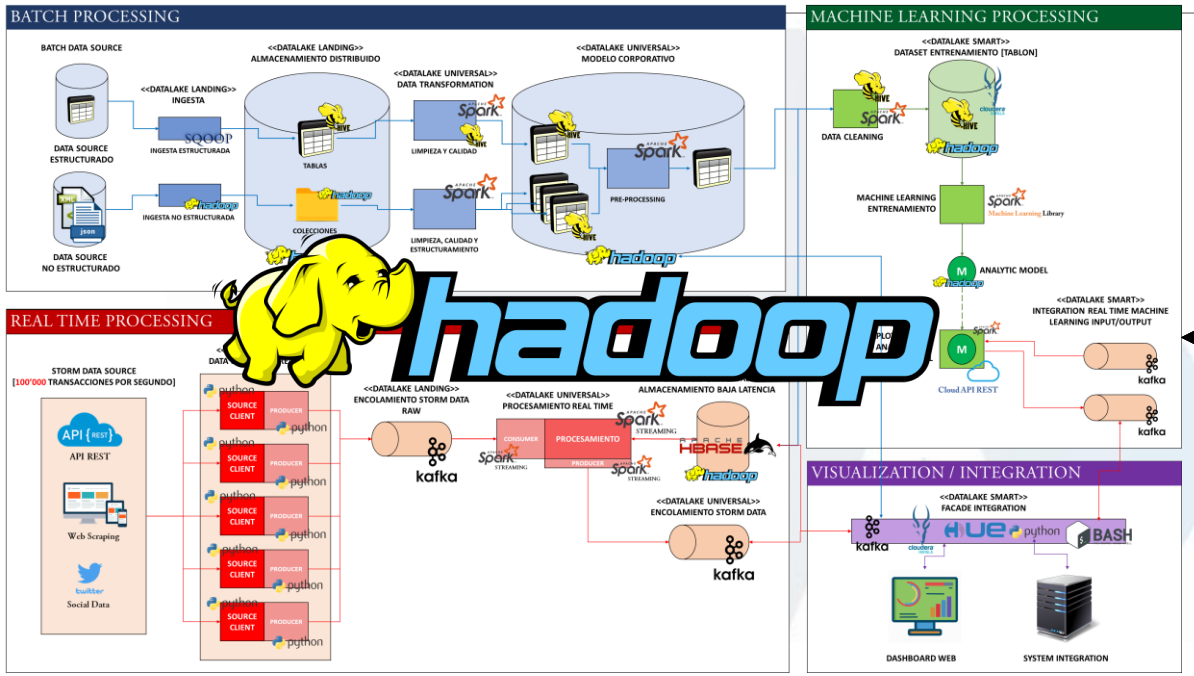
Naturaleza de funcionamiento

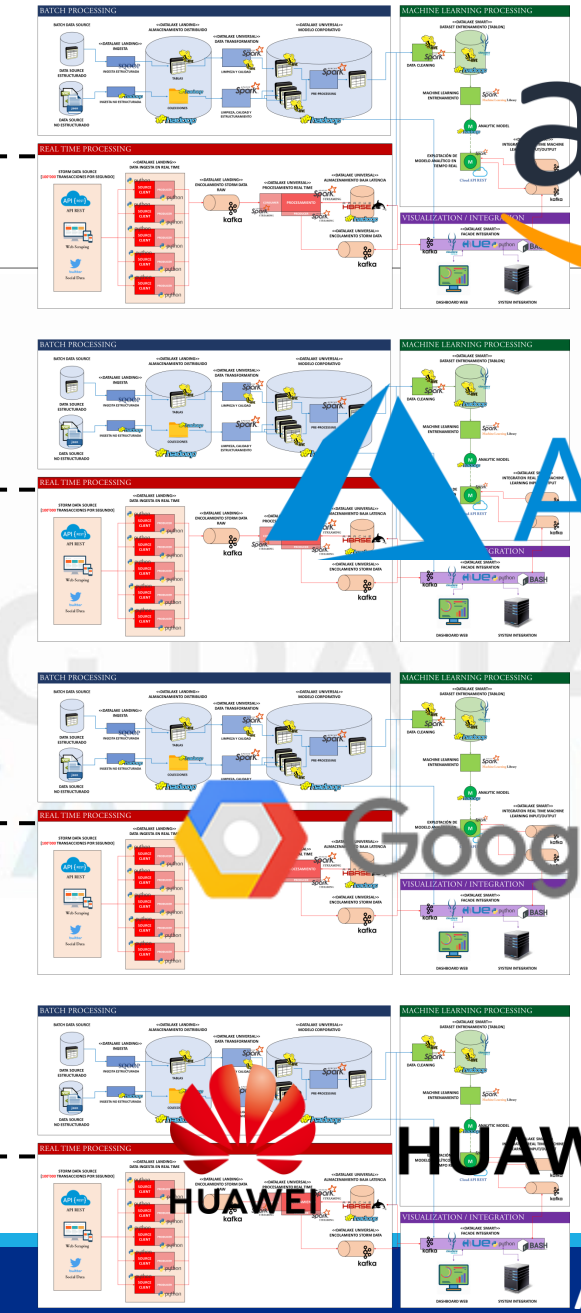
Vista Conceptual con Patrones de Diseño de Big Data



¡Este diagrama sí lo podemos codificar!

El ecosistema estándar basado en Hadoop





Basado en el Ecosistema **Estándar Hadoop**

Los patrones de diseño son agnósticos al ecosistema tecnológico

Herramientas cloud equivalentes

	ECOSISTEMA ESTÁNDAR	aws	Google Cloud Platform	Azure
Almacenamiento	hadoop	amazon S3	Cloud Storage	Azure Blob Storage
Interfaz SQL	HIVE	Amazon Athena	Google Big Query	Azure Synapse Analytics
ETL	HIVE	AWS Glue	Data Fusion	DATAFLOW
Soluciones	APACHE Spark	amazon EMR APACHE Spark	Cloud Dataproc APACHE Spark	HDInsight APACHE Spark
Workflow	OOZIE	Apache Airflow aws	Google Cloud Composer	Azure Data Factory

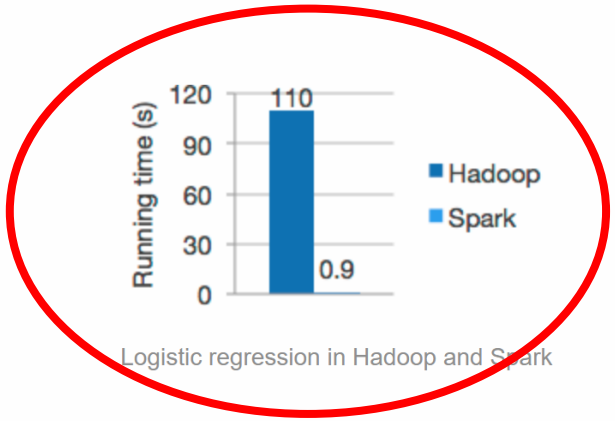
Spark vs Hadoop

Apache Spark™ is a unified analytics engine for large-scale data processing.

Speed

Run workloads 100x faster.

Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.



Logistic regression in Hadoop and Spark

Ease of Use

Write applications quickly in Java, Scala, Python, R,

```
df = spark.read.json("logs.json")
df.where("age > 21")
  .select("name.first").show()
```

Latest News

Spark 2.3.3 relea

Spark 2.2.3 relea

Spark+AI Summ
2019, San Franc
(Dec 19, 2018)

Spark 2.4.0 relea

APACI
North Ar
Las Vega

Downl

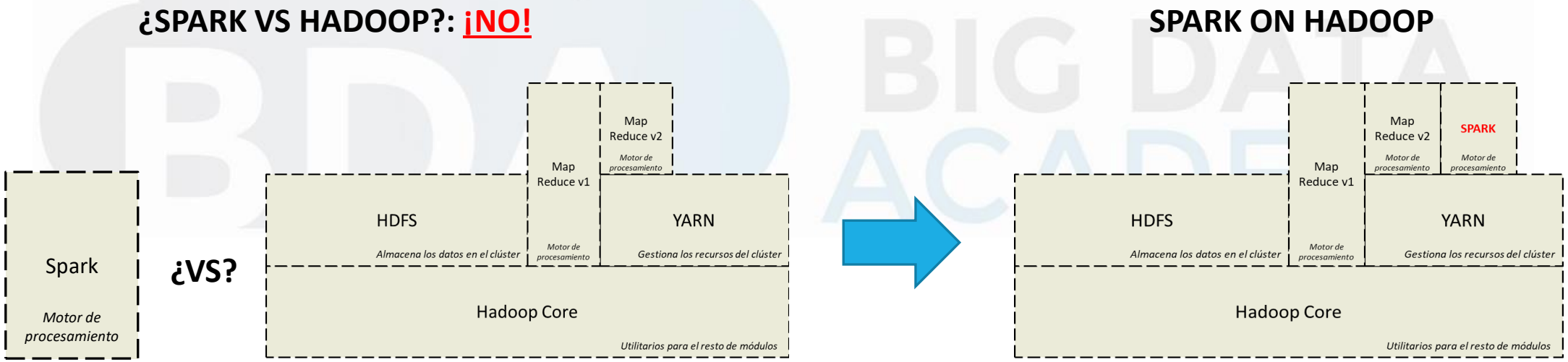
Built-in Libraries

SQL and DataFrar

Spark puede llegar a ser hasta **100 veces más rápido que Hadoop**

Si Spark es “mejor” entonces, ¿reemplaza a Hadoop?

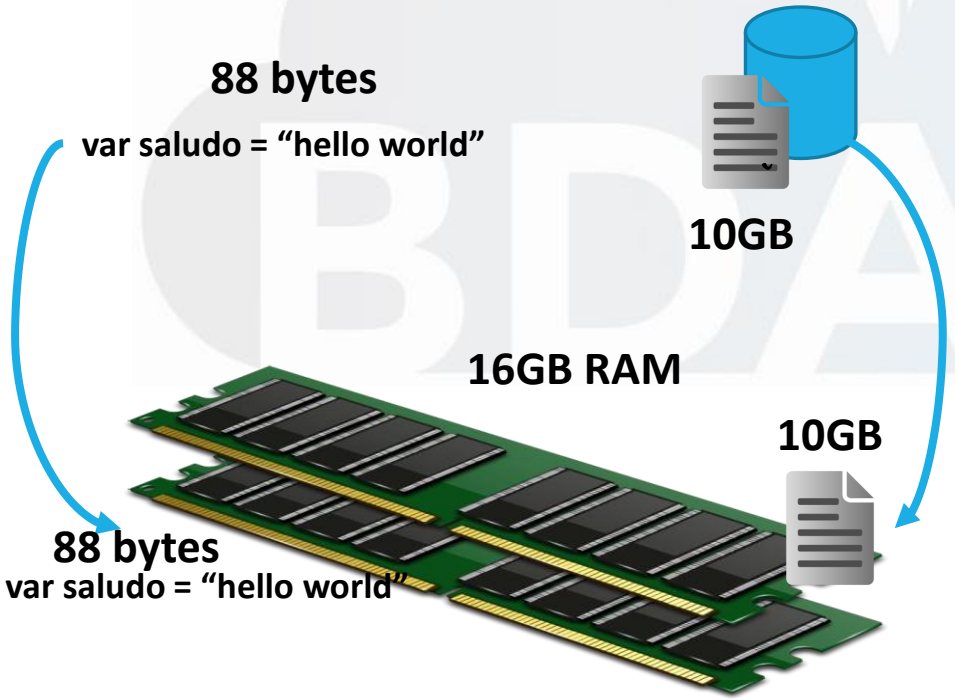
No, Spark es un motor de procesamiento, Hadoop es un ecosistema que incluye un motor de procesamiento llamado MapReduce v2 y otros componentes (HDFS para almacenar en disco duro, YARN para gestionar recursos). **Spark no reemplaza a Hadoop, sino que lo potencia.**



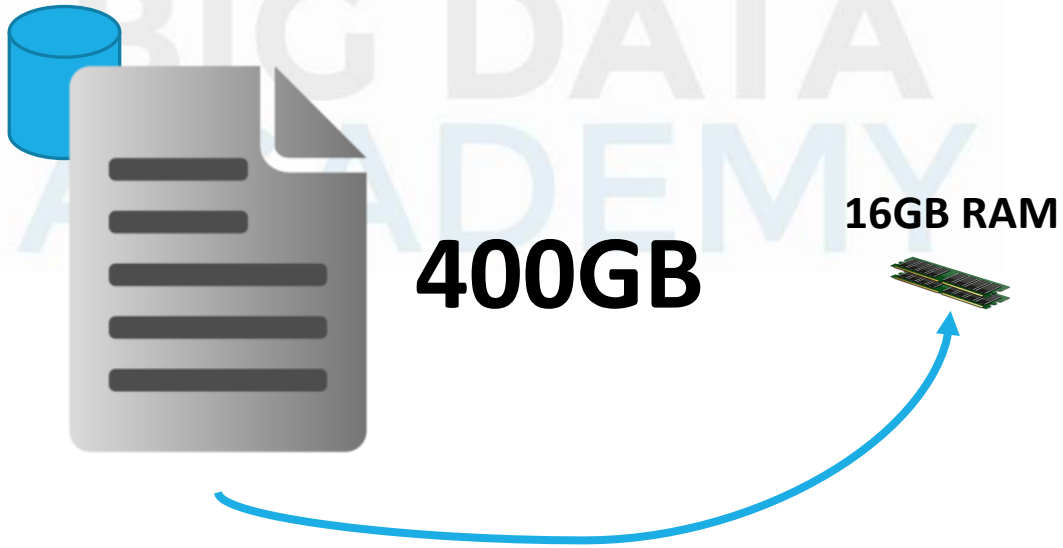
Variables in-memory

Variables en memoria

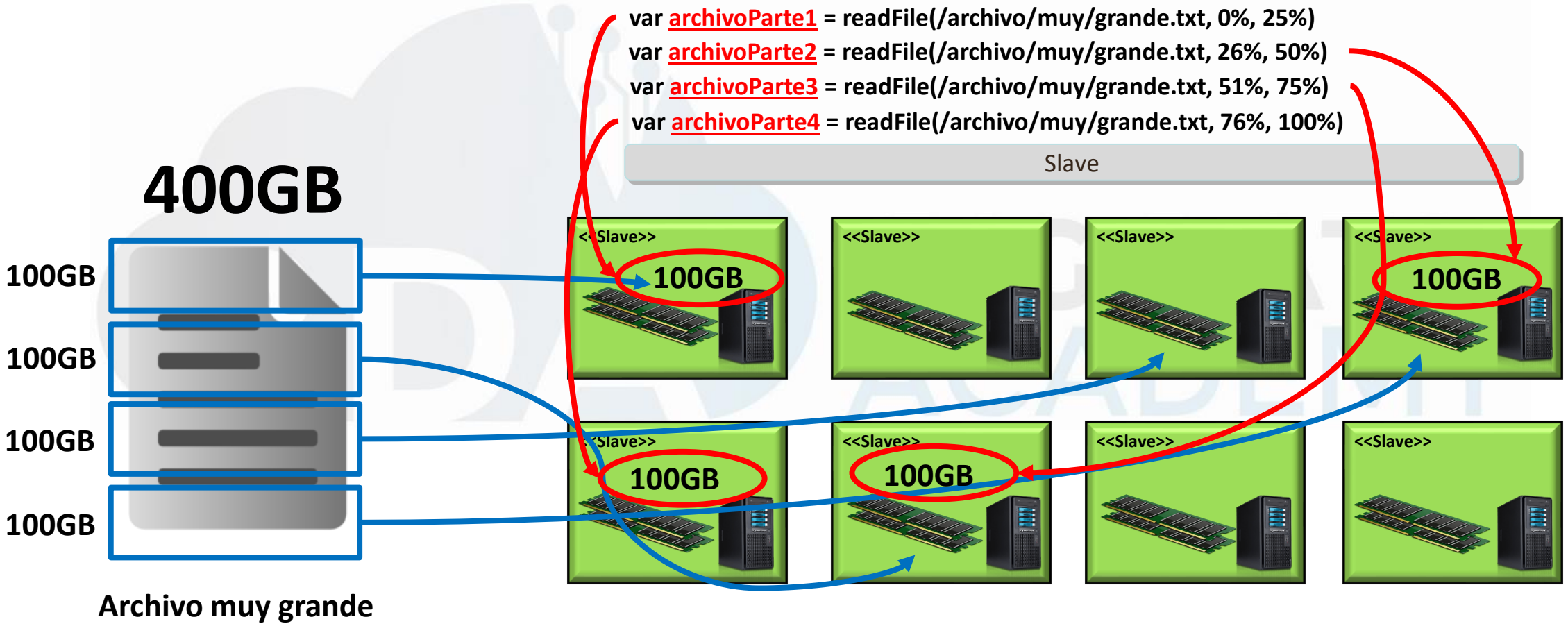
¿Cómo se crea una variable en memoria?



¿Y si tengo un archivo muy grande?

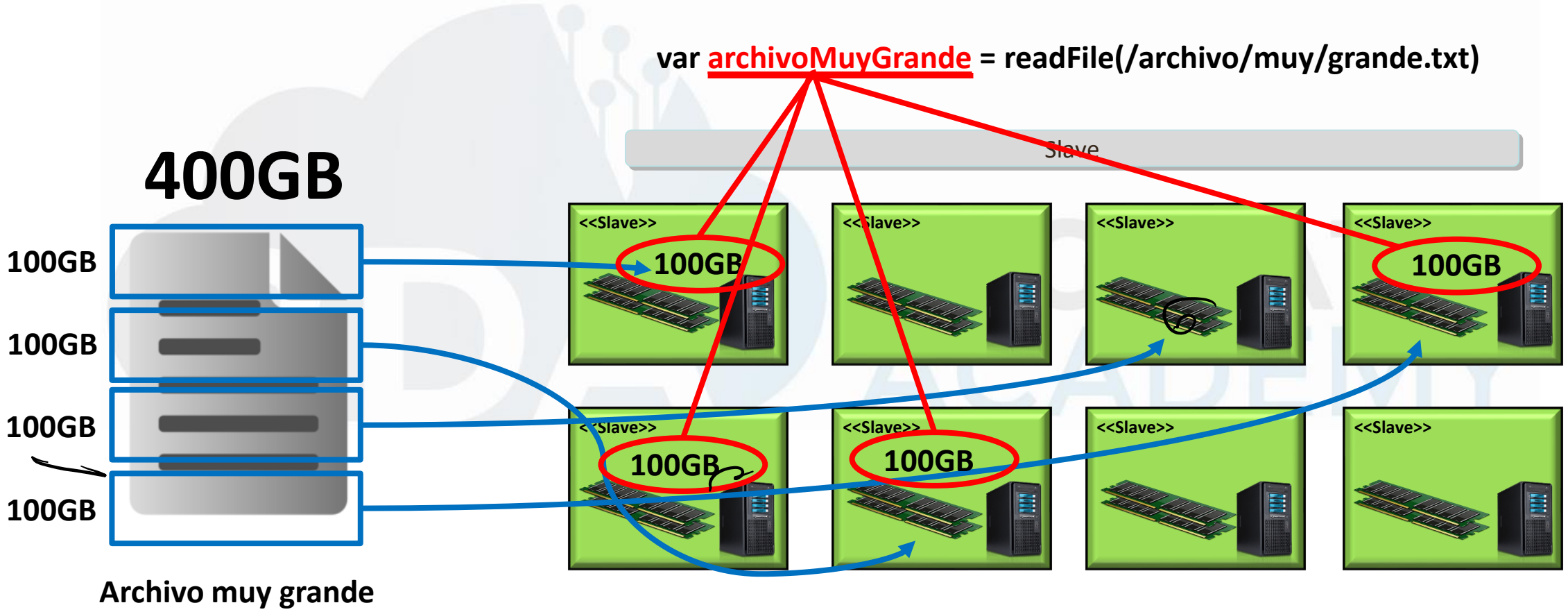


En un clúster clásico



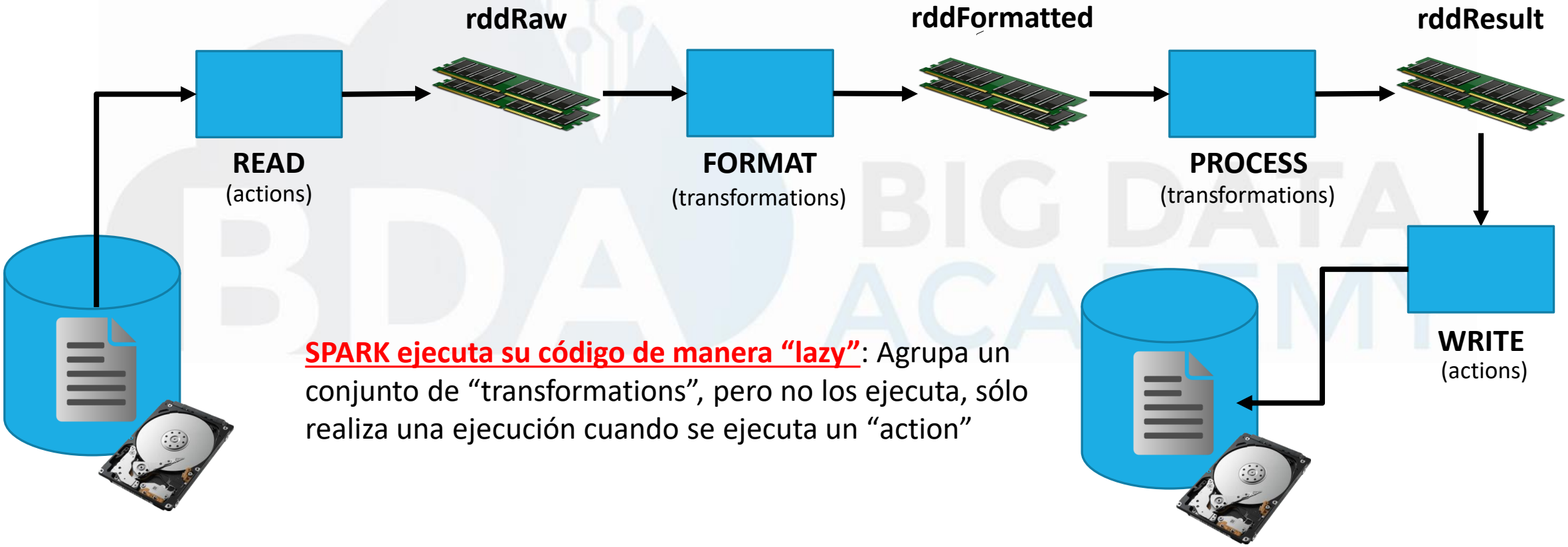
Servidores de 256GB de RAM cada uno

RDD: Resilient Distributed Dataset



Servidores de 256GB de RAM cada uno

Arquetipo de procesamiento en SPARK



SPARK ejecuta su código de manera "lazy": Agrupa un conjunto de "transformations", pero no los ejecuta, sólo realiza una ejecución cuando se ejecuta un "action"

Equivalente de un GROUP BY

```
rddPersonaGroupBy = sc.textFile("/dataset/persona.data"). \
map(lambda line : line.split("|")). \
filter(lambda register : register[0] != "ID"). \
map(lambda register : ( \
int(register[7]), \
( \
1, \
float(register[6]), \
float(register[6]), \
float(register[6]), \
int(register[5]) \
) \
) \
). \
reduceByKey(lambda v1, v2 : ( \
v1[0] + v2[0], \
v1[1] + v2[1], \
max(v1[2], v2[2]), \
min(v1[3], v2[3]), \
v1[4] + v2[4] \
)). \
map(lambda register : (( \
register[0], \
( \
register[1][0], \
register[1][1], \
register[1][2], \
register[1][3], \
register[1][4]/register[1][0] \
) \
))). \
sortBy(lambda register : register[0])
```

```
df5 = dfData.\
groupBy("EDAD").\
agg(\
f.count("EDAD"), \
f.min("FECHA_INGRESO"), \
f.sum("SALARIO"), \
f.max("SALARIO")\
)
```

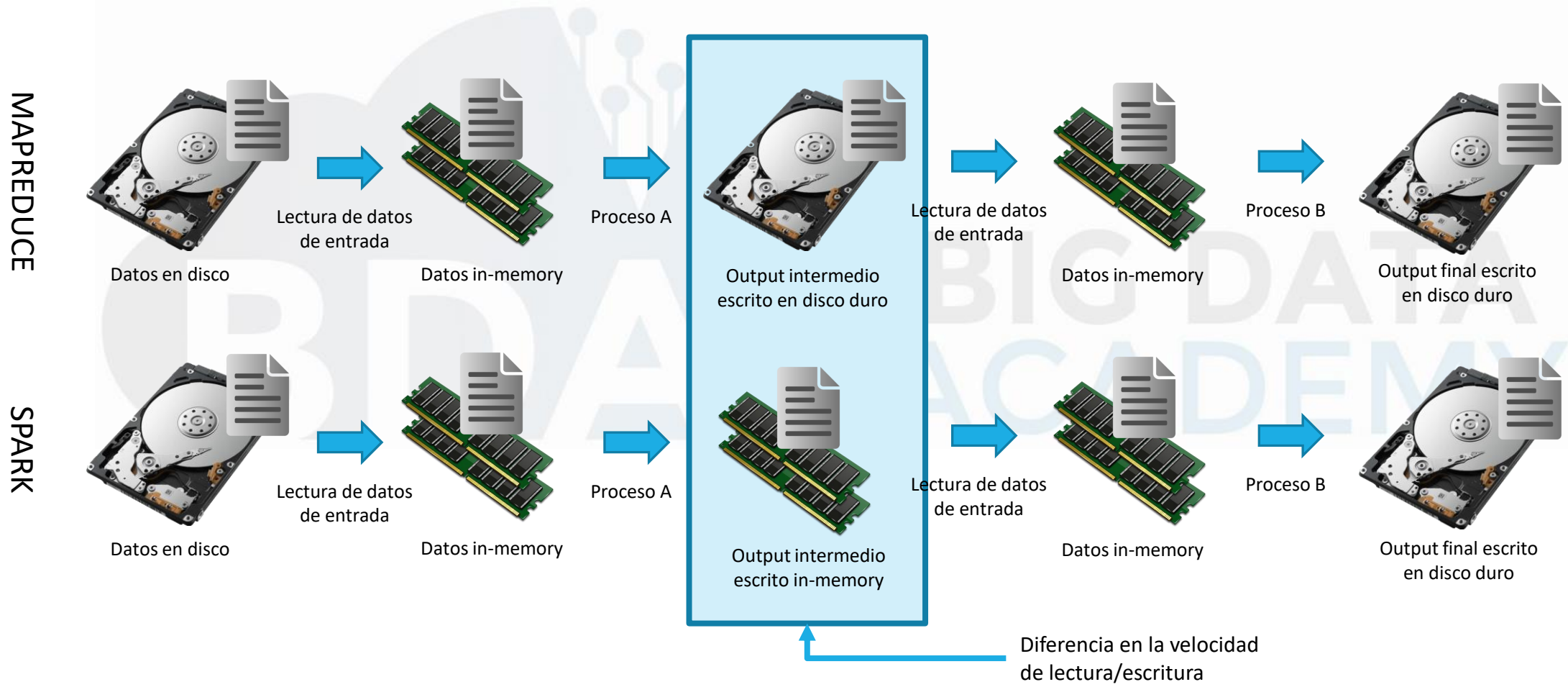
Agregando estructura a los RDD: Los Dataframes

RDD + **METADATA** = **DATAFRAME**

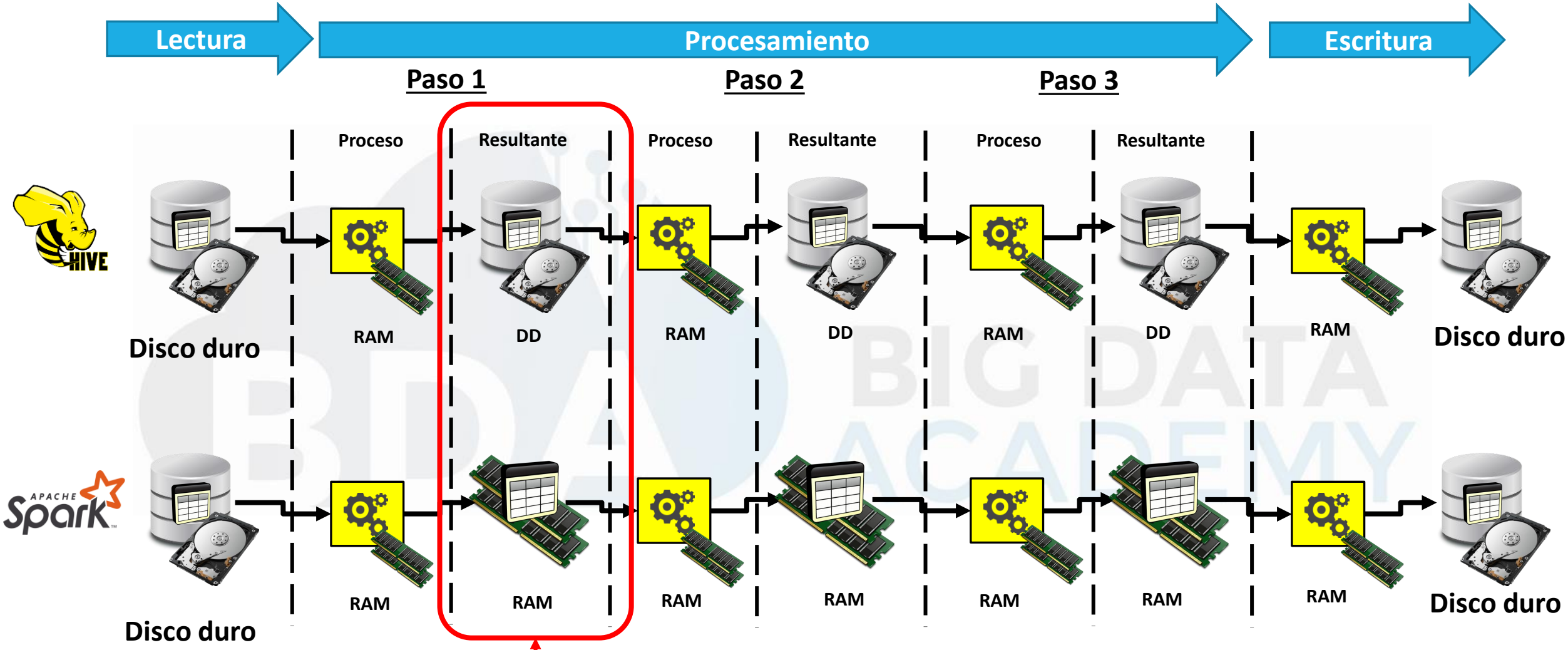
(CAMPO1 STRING,
CAMPO2 INT,
CAMPO3 DOUBLE,
…)

Procesamiento in-memory

¿Por qué SPARK es más rápido que MapReduce?

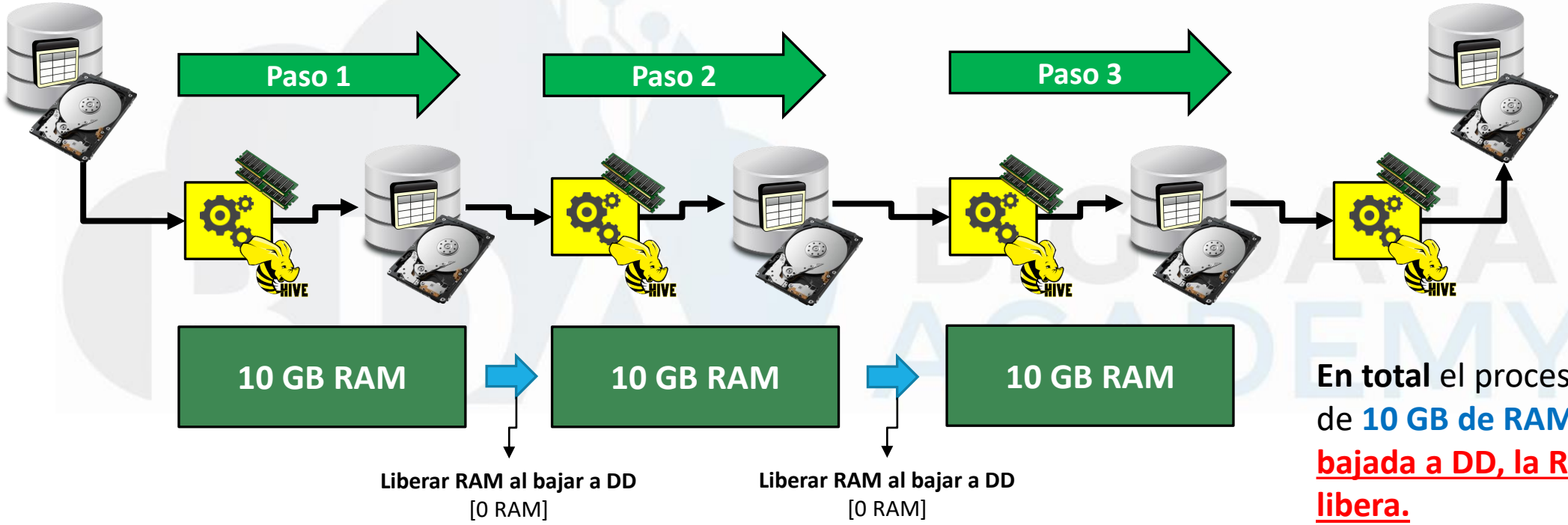


Procesamiento in-memory



Las resultantes intermedias se almacenan en RAM, y **la RAM es hasta 100 veces más rápida** que el disco duro

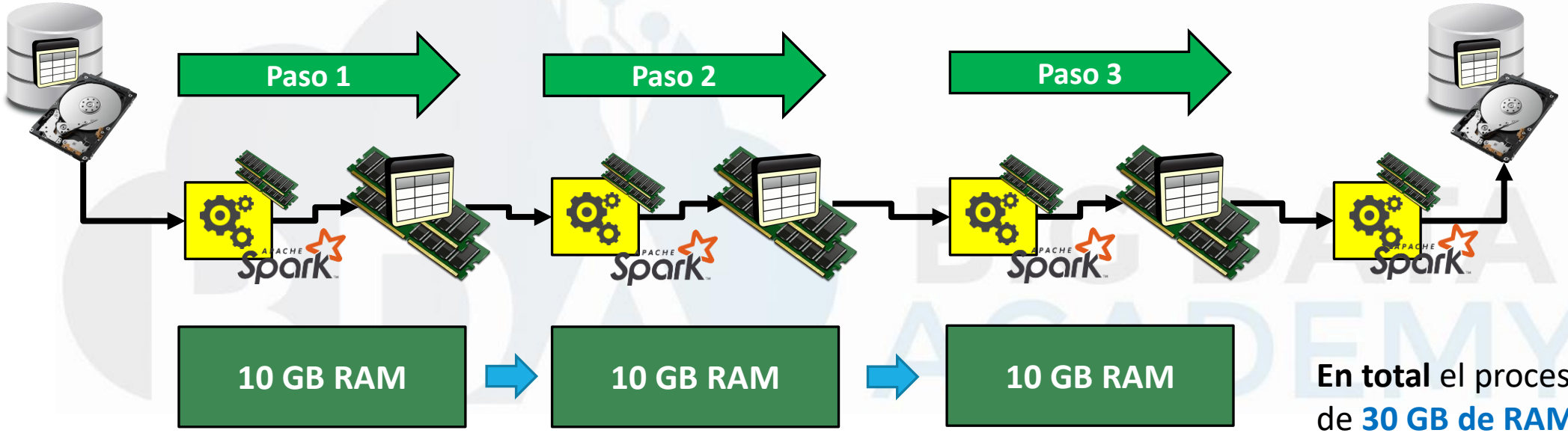
¿El procesamiento in-memory es mejor?



En total el proceso necesito de **10 GB de RAM**, **en cada bajada a DD, la RAM se libera.**

Un proceso que intercala procesamiento entra RAM y DD, **no consume mucha memoria RAM**

¿El procesamiento in-memory es mejor?



En total el proceso necesito de **30 GB de RAM**

No hay liberación de memoria RAM, el uso de la RAM se va acumulando

Un proceso que usa intensivamente la RAM, **consume mucha memoria RAM**

Mientras más pasos, más memoria RAM

```python
#PASO 1
df1 = dfData.groupBy(dfData["EDAD"]).agg(
    f.count(dfData["EDAD"]).alias("CANTIDAD"),
    f.min(dfData["FECHA_INGRESO"]).alias("FECHA_CONTRATO_MAS_RECIENTE"),
    f.sum(dfData["SALARIO"]).alias("SUMA_SALARIOS"),
    f.max(dfData["SALARIO"]).alias("SALARIO_MAYOR")
)

#PASO 2
df2 = df1.filter(df1["EDAD"] > 35)

#...

#PASO 10
df10 = df9.filter(df2["SUMA_SALARIOS"] > 500000)

#RESULTANTE
dfResultado = df3.filter(df3["SALARIO_MAYOR"] > 1000)
dfResultado.show()
```

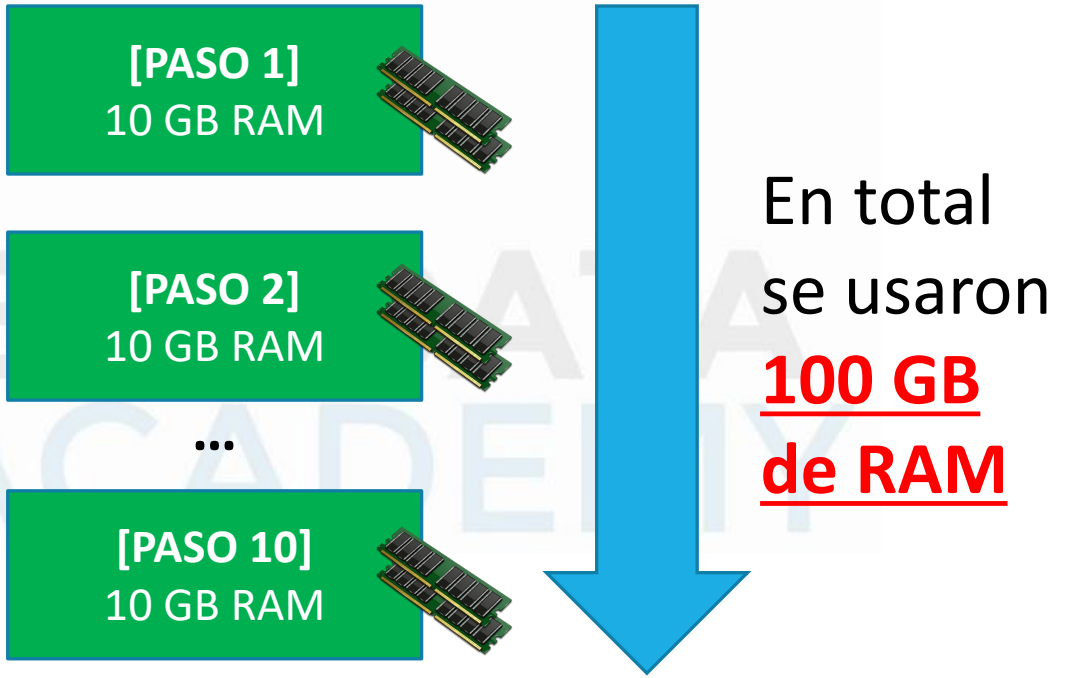
En total se usaron **100 GB de RAM**

¿**Encadenar todo el proceso en un único gran paso** lo soluciona?

¿Y si tengo **CIEN** pasos, o **MIL** pasos?

¿Y si encadeno procesos, optimizo?

```
1  dfResultado = dfData.groupBy(dfData["EDAD"]).agg( #PASO 1
2      f.count(dfData["EDAD"]).alias("CANTIDAD"),
3      f.min(dfData["FECHA_INGRESO"]).alias("FECHA_CONTRATO_M
4      f.sum(dfData["SALARIO"]).alias("SUMA_SALARIOS"),
5      f.max(dfData["SALARIO"]).alias("SALARIO_MAYOR")
6  ).alias("D").\
7  filter(f.col("D.EDAD") > 35).\ #PASO 2
8  ....
9  filter(f.col("D.SUMA_SALARIOS") > 5000).\ #PASO 10
10 filter(f.col("D.SALARIO_MAYOR") > 1000) #RESULTANTE
```



En total se usaron **100 GB de RAM**

Ejecutas un único gran paso, que **seguirá usando la misma cantidad de memoria RAM**, **no estamos optimizando.**

¿Nuestro script tiene la suficiente memoria RAM (100 GB) para funcionar?