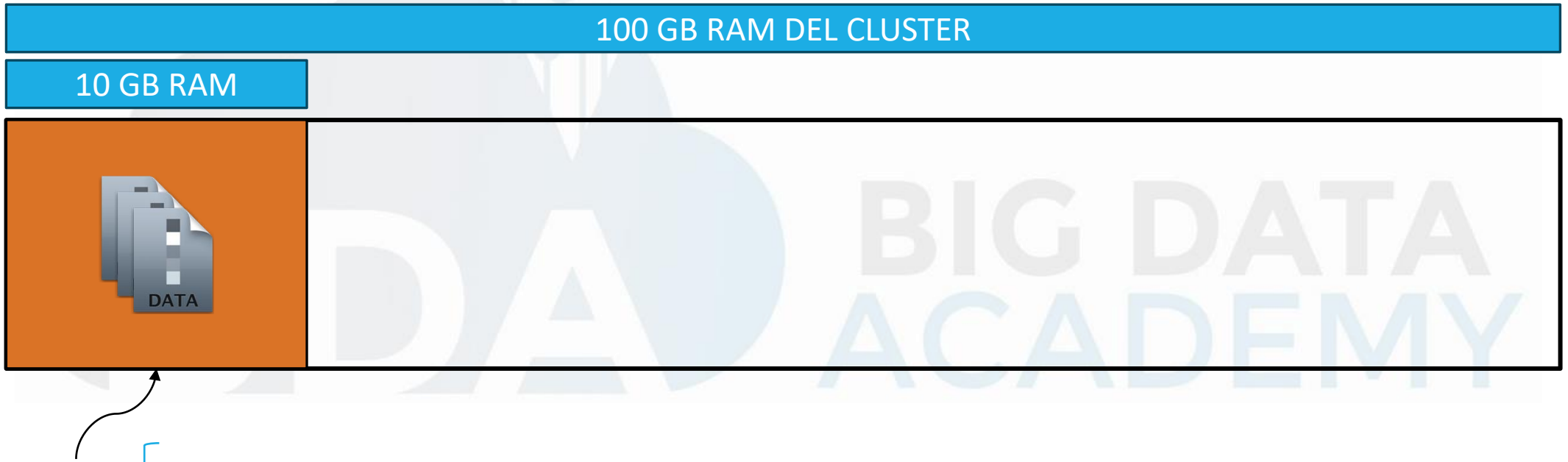
Gestión de Memoria Ram en Spark

Garbage Collector

Dataframes como punteros



- En SPARK, las variables dataframes son realmente “punteros”
- **Un puntero es una variable que apunta a una zona de memoria RAM** en donde se ha colocado un archivo de datos

¿Cómo liberamos memoria RAM?

df1 → **NULL**

- Si ejecutamos **la sentencia "df1 = null", no estamos liberando memoria RAM**, sólo estamos haciendo que el puntero "df1" deje de apuntar a la zona de memoria RAM inicial
- **La zona de memoria RAM inicial sigue estando ocupada**

Garbage Collector

Es el módulo que gestiona la memoria RAM, se encarga de liberar zonas de memoria RAM que ya no están siendo referenciadas (apuntadas) por una variable. Funciona de la siguiente manera:

---

df1 ⟶ **NULL**

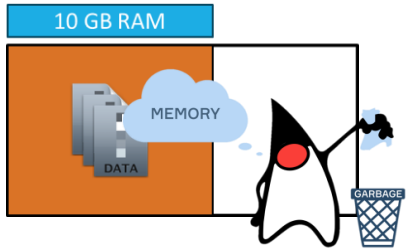
1. Asignamos la variable en “NULL”

GC [Garbage Collector]

2. El Garbage Collector (**GC**) recorre todas las zonas de memoria RAM para liberar las variables “NULL”. **La liberación no es instantánea**, si el GC está cerca a nuestra zona de memoria, puede tardar un par de segundos en liberarla, en el peor de los casos puede tardar hasta 120 minutos en llegar a nuestra zona de memoria RAM

[45 minutos]

3. Por ejemplo, **puede ser que el GC está lejos de nuestra zona de memoria** y llega a ella luego de 45 minutos de haber hecho “df1 = null”.



4. Una vez que el GC llega a nuestra zona de memoria RAM verifica que no esté siendo apuntada por ninguna variable y libera la zona de memoria RAM

Transformations y Actions

Encadenamiento de procesos en SPARK

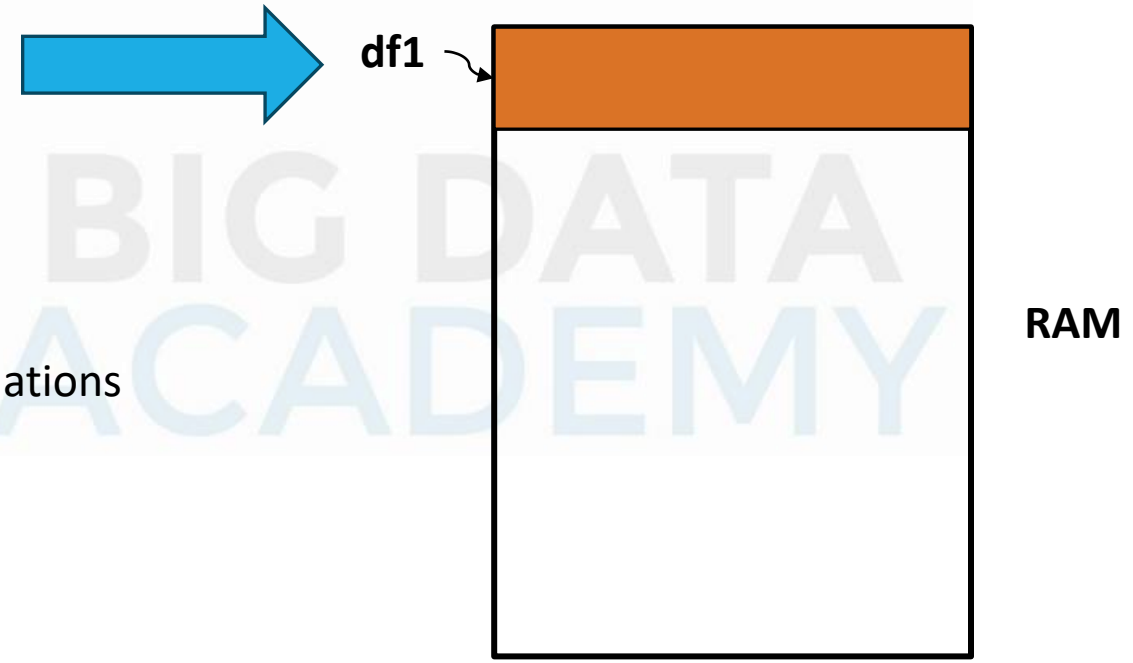
Una cadena de procesos es un conjunto de funciones “actions” y “transformations”. Los “actions” crean al dataframe en memoria RAM, **los “transformations” son “lazy” (perezosos), sólo definen cómo se creará el dataframe, pero no lo crea.**

Por ejemplo, en este código sólo se crea el “df1”, ya que es el único que es creado por un “action” (load)

```
#PASO 1: LEER
df1 = spark.read.format("csv").....

#PASO 2: FILTER
df2 = df1.filter(.....................)

#PASO 3: GROUP BY
df3 = df2.filter(.....................)

#PASO 4: FILTER
df4 = df3.filter(.....................)

#PASO 5: GROUP BY
df5 = df4.filter(.....................)
```

Action

Transformations



¿Cómo se crean los demás dataframes en la RAM?

Creación de los dataframes con “actions”

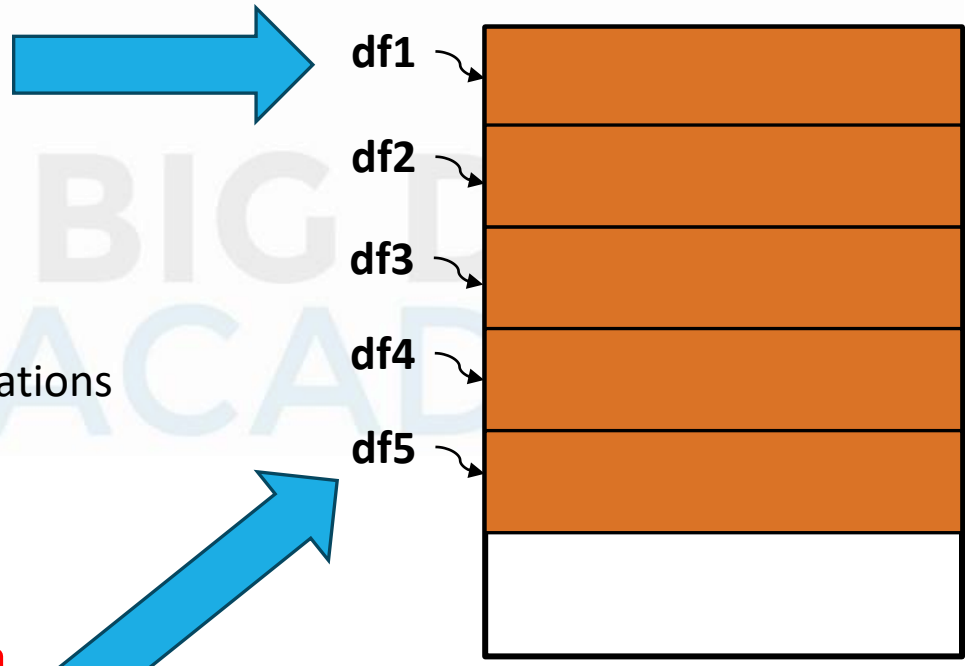
Sobre alguno de los dataframes de la cadena de transformations, **deberemos llamar a un “action”**, para que se creen todos los dataframes de la cadena. Por ejemplo, podemos llamar al action “write” sobre el “df5” para crear todos los dataframes

```
#PASO 1: LEER
df1 = spark.read.format("csv").....

#PASO 2: FILTER
df2 = df1.filter(.....................)

#PASO 3: GROUP BY
df3 = df2.filter(.....................)

#PASO 4: FILTER
df4 = df3.filter(.....................)

#PASO 5: GROUP BY
df5 = df4.filter(.....................)

#PASO 6: ESCRITURA
df5.write.format("csv").............
```

Action

Transformations

Action



Supongamos que cada DF demora 1 hora en crearse, en total **el action demoró 4 horas en ejecutar la cadena de procesos**

El problema de los actions y el Garbage Collector

¿Cómo se liberan los dataframes desde la memoria RAM?

En SPARK, cuando una cadena de dataframes es creada por un “action”, **todos los dataframes se marcan para ser liberados por el Garbage Collector,** cuando el Garbage Collector pase por esa zona de memoria RAM los eliminará.

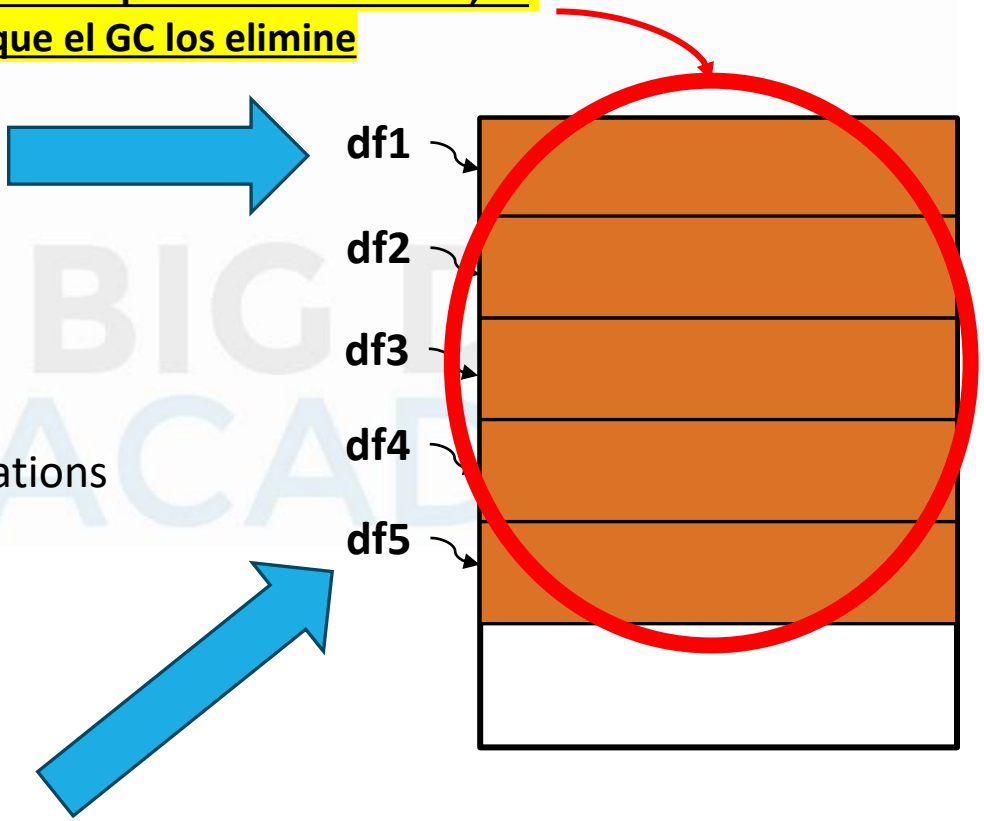
Inmediatamente después de ser creados, se marcan para que el GC los elimine

```
#PASO 1: LEER
df1 = spark.read.format("csv").....

#PASO 2: FILTER
df2 = df1.filter(.....................)

#PASO 3: GROUP BY
df3 = df2.filter(.....................)

#PASO 4: FILTER
df4 = df3.filter(.....................)

#PASO 5: GROUP BY
df5 = df4.filter(.....................)

#PASO 6: ESCRITURA
df5.write.format("csv").................
```

Action

Transformations

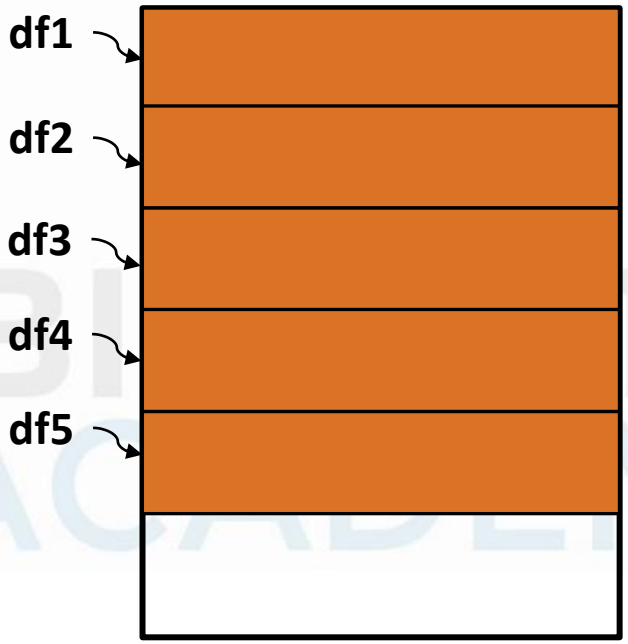
Action

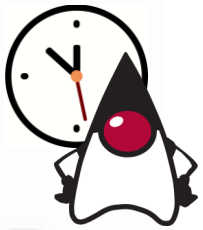


¿Cuándo elimina el GC a los dataframes?

Depende de qué tan cercano está de nuestra zona de memoria, puede que elimine a todos los dataframes 5 segundos después de ser creados por el action, o puede eliminarlos aún dentro de 50 minutos

```
#PASO 1: LEER
df1 = spark.read.format("csv").....

#PASO 2: FILTER
df2 = df1.filter(.....)

#PASO 3: GROUP BY
df3 = df2.filter(.....)

#PASO 4: FILTER
df4 = df3.filter(.....)

#PASO 5: GROUP BY
df5 = df4.filter(.....)

#PASO 6: ESCRITURA
df5.write.format("csv").....

#PASO 7: MOSTRAR
df5.show()
```





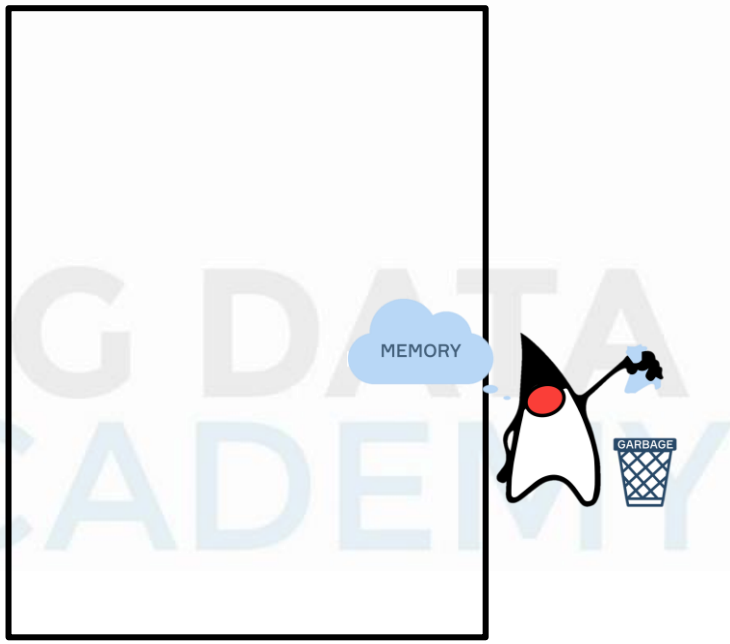
[El GC está lejos, pasará aún en 30 minutos]

Por ejemplo, si el GC aún pasará por nuestra zona de memoria dentro de 30 minutos y antes de eso ejecuto el action “show”, **el action “show” no vuelve a crear todos los dataframes, porque ya están en la RAM**

¿Y qué pasa luego de esos 30 minutos?

Supongamos que luego de 30 minutos el GC pasó por nuestra zona de memoria y eliminó todos los dataframes, ¿qué sucede si en ese momento ejecuto nuevamente el action “show”?

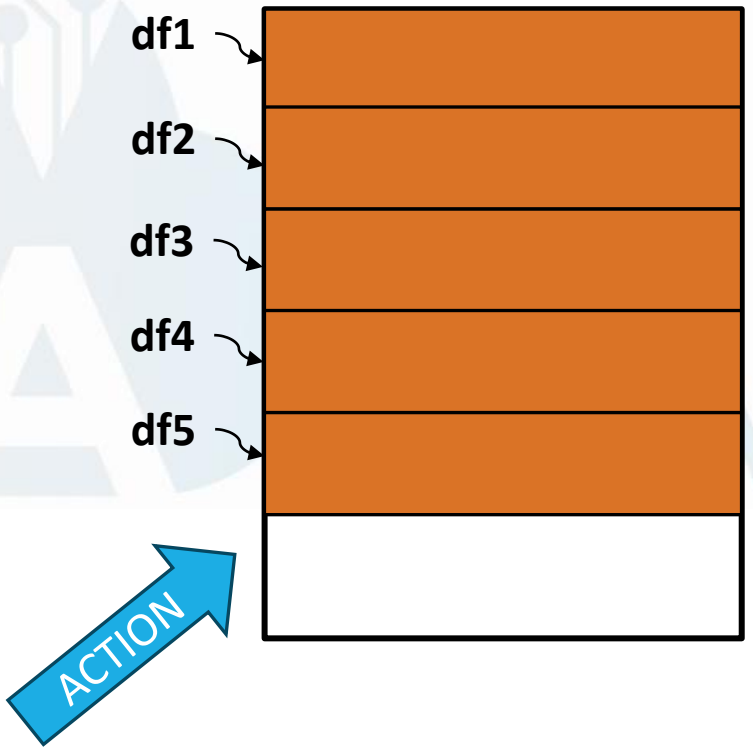
```
#PASO 1: LEER
df1 = spark.read.format("csv").....

#PASO 2: FILTER
df2 = df1.filter(.....................)

#PASO 3: GROUP BY
df3 = df2.filter(.....................)

#PASO 4: FILTER
df4 = df3.filter(.....................)

#PASO 5: GROUP BY
df5 = df4.filter(.....................)

#PASO 6: ESCRITURA
df5.write.format("csv").............

#PASO 7: MOSTRAR
df5.show()
```



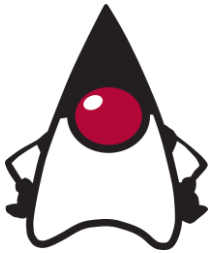
Como no hay nada en memoria RAM, **el “action” manda a ejecutar toda la cadena de dataframes que crean al “df5”**, deberemos esperar nuevamente 4 horas para que se creen todos los dataframes y ver los registros con el show

“Aleatoriedad” de la eliminación de los dataframes

```
#PASO 1: LEER
df1 = spark.read.format("csv").....

#PASO 2: FILTER
df2 = df1.filter(.....................)

#PASO 3: GROUP BY
df3 = df2.filter(.....................)

#PASO 4: FILTER
df4 = df3.filter(.....................)

#PASO 5: GROUP BY
df5 = df4.filter(.....................)

#PASO 6: ESCRITURA
df5.write.format("csv").................
```



*“Si una cadena de dataframes fue creada por un action, **puede que los borre todos dentro de 1 segundo o puede que los borre todos aún en 2 horas**”*



GC [Garbage Collector]

Dataframes en la caché

Marcando los dataframes en la “caché”

- Podemos evitar que un dataframe sea borrado de la RAM **marcándolo en la “caché”**
- Cuando un dataframe está en la “caché” y el GC pasa por su zona de memoria RAM no lo borra
- Un dataframe en la caché sólo se borra si lo des-marcamos de la caché o cuando el programa finaliza.