Vishal Gupta

SPARK CORE FUNDAMENTALS

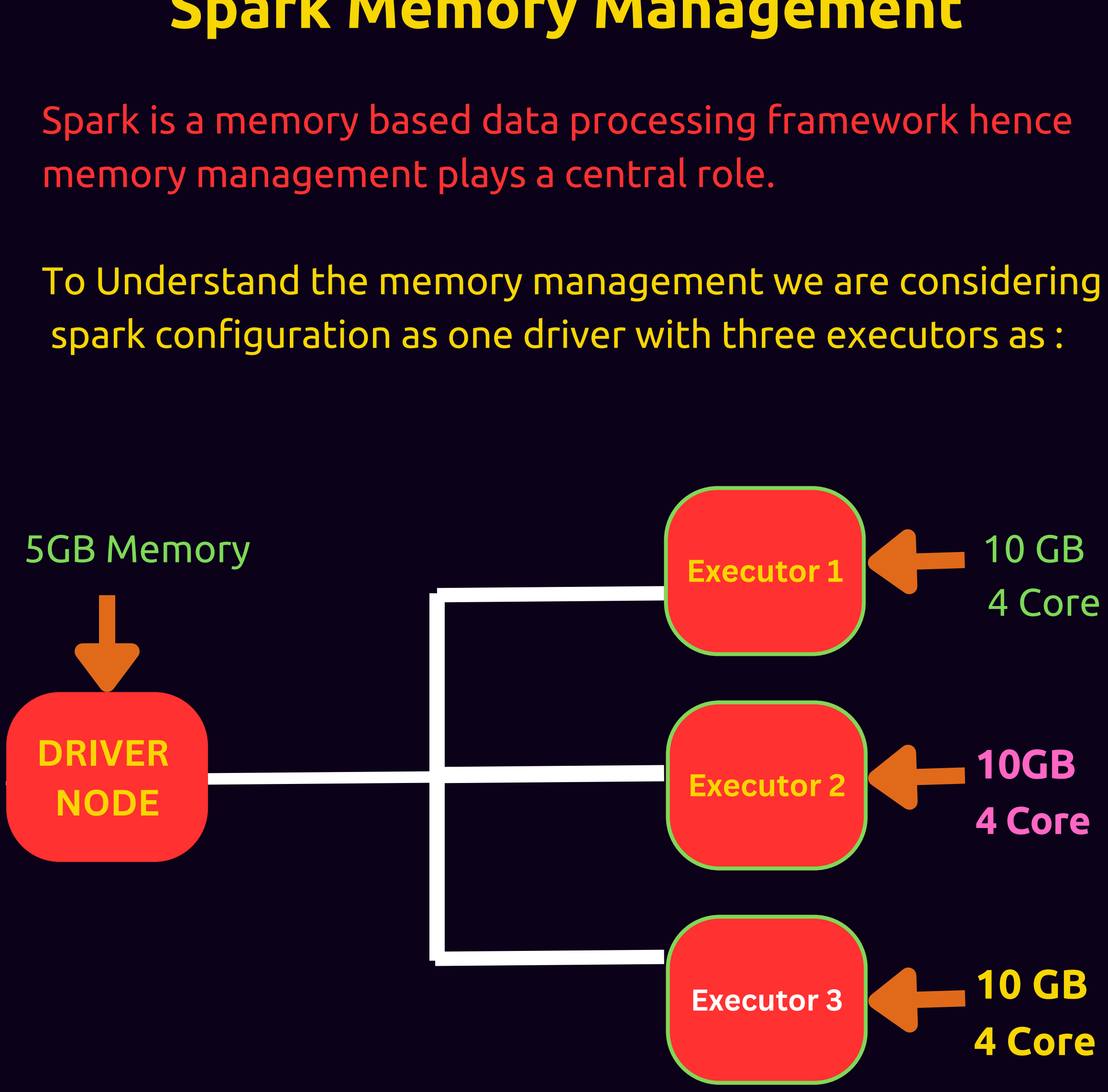
UnWrap All concept in SImplest way

SPARK MEMORY MANAGEMENT

Spark Memory Management

Spark is a memory based data processing framework hence memory management plays a central role.

To Understand the memory management we are considering spark configuration as one driver with three executors as :



The diagram illustrates a setup where the driver node is allocated 5GB of memory, while each of the three executors is assigned 10GB of memory and equipped with a 4-core CPU.

Let’s expand any of **one executor** to understand this in detail

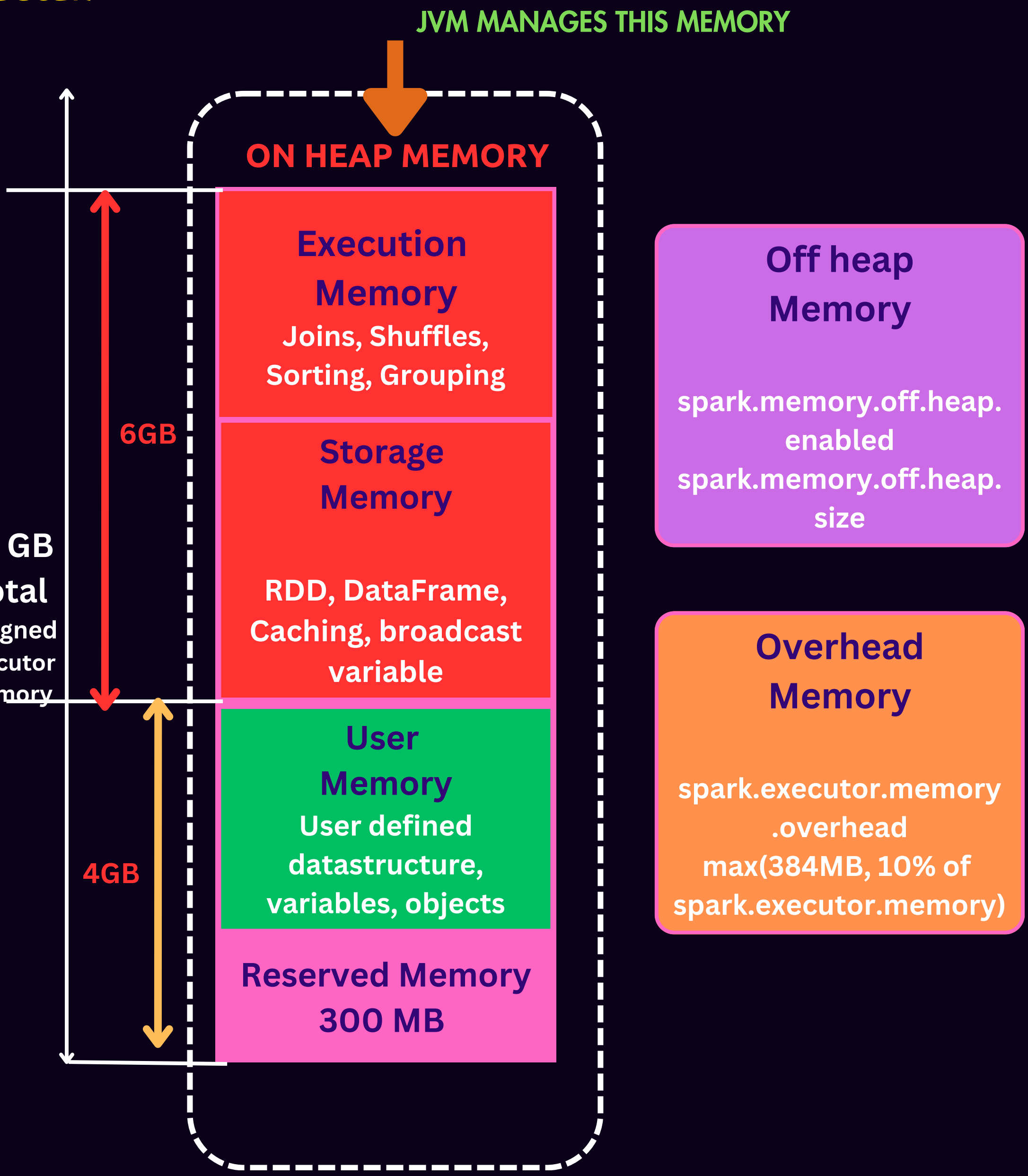


DIAGRAM - SPARK EXECUTOR CONTAINER WITH ALL AVAILABLE MEMORY

If we look into spark executor container we primarily have three blocks

1. On-heap memory
2. Off-heap memory
3. Overhead Memory

Most of the opration will run on **on-heap memory** which gets managed by **spark JVM**.

Note - JVM stands for Java Virtual machine and it is like a virtual computer that is used to run java programs. Spark is written in scala which runs on the JVM . JVM serves as execution environment to scala.

Whenever we write code in pyspark we are using wrapper around java API's of Spark. This is how inner execution of pyspark happens on JVM. It justifies how In-heap memory gets managed by JVM.

On heap memory is again divided into 4 sections -

1. Execution Memory -
   Joining, Shuffles, Sorting and Groupby operations happens using this memory

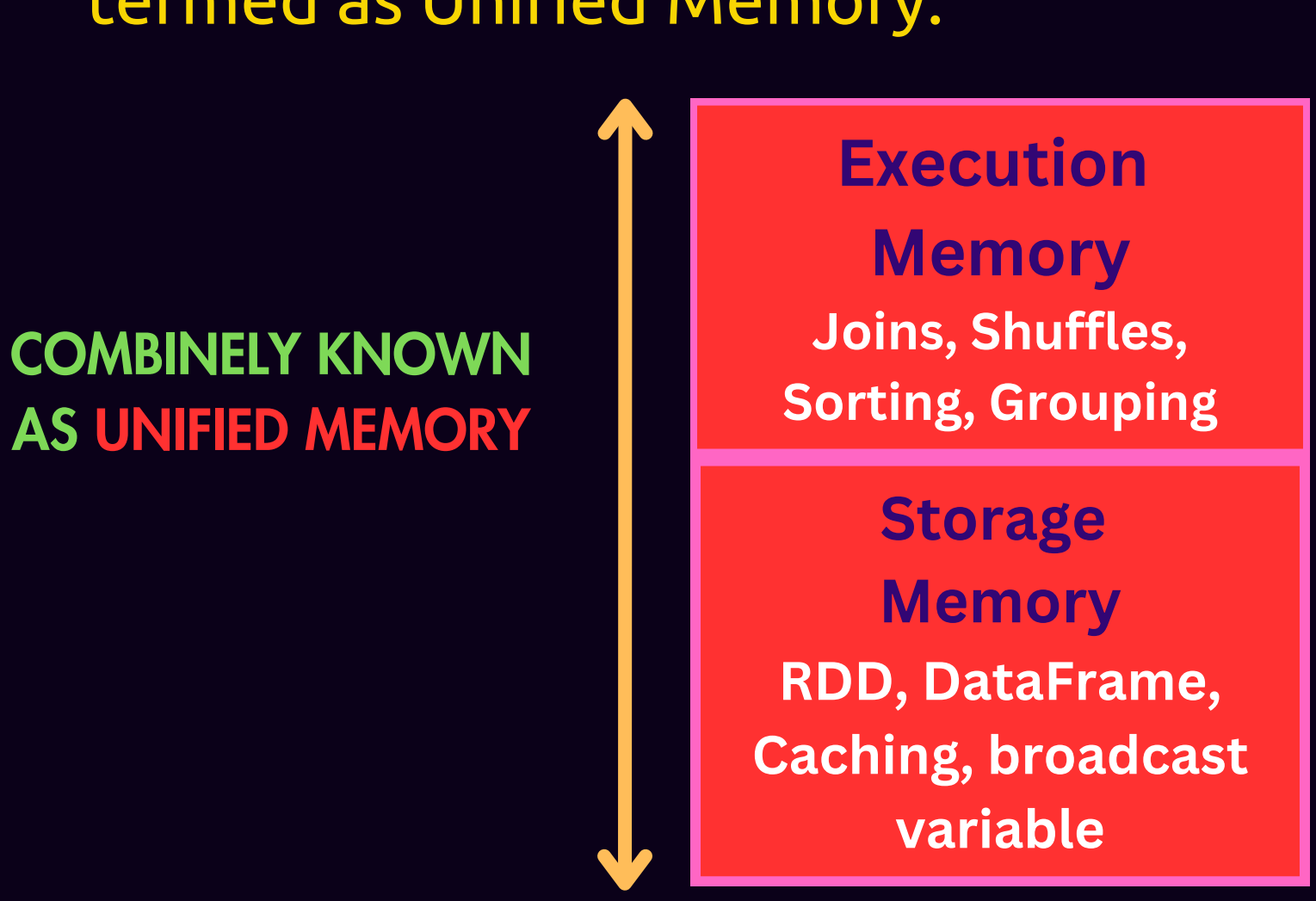
2. Storage Memory -
   RDD, Dataframes caching happens on this memory.
   This also gets used to store broadcast variables .

3. User Memory -
   User Objects, Variables and UDF's gets stored on this.

4. Reserved Memory -
   This is reserved part of memory that spark uses to run itself and for storing Internal objects

5. Overhead Memory - It is used for some Internal system level operations.

Unified Memory -
Execution memory combined with Storage memory termed as Unified Memory.



Till here, we understood the description of each memory under executor memory layout

Let’s understand which portion of memory is going to take how much space

Initially we assumed spark executor memory as 10 GB

spark.executor.memory = 10GB (Total assigned memory)

spark.memory.fraction = 60 % of spark.executor.memory

spark.memory.fraction **= 6 GB (Unified memory)**

spark.storage.memory.fraction = 50 % of spark.memory.fraction

spark.storage.memory.fraction = 50% of 6GB

spark.storage.memory.fraction = 3GB

Left memory = 4 GB out of which

User Memory = 3.7 GB

Reserved memory

Let’ see these portions in diagram

UNIFIED MEMORY (0.6 of total memory) = 6GB

SPARK.MEMORY.EXECUTOR = 10GB

SPARK.MEMORY.FRACTION = 6GB

ON HEAP MEMORY

Execution Memory
Joins, Shuffles, Sorting, Grouping

Storage Memory
RDD, DataFrame, Caching, broadcast variable

SPARK.STORAGE.MEMORY=3GB

User Memory
User defined datastructure, variables, objects

Reserved Memory
300 MB

4 GB

4 GB
Out of this 4GB
300 MB reserved
User memory=3.7GB

Till here, we calculated

spark.memory.fraction that is 6GB
spark.storage.memory that is 3GB
User Memory that is 3.7GB
Reserved memory that is 300MB

Overhead Memory - Overhead memory will be extra from spark.memory.executor memory.

It will be calculated as
max(384,10% of spark executor memory)
hence max would be 1 GB

Overhead Memory would be 1GB

Off heap memory - Off heap memory will be disabled by default but we can eneable it by assigning about 10 - 20 percent of executor memory.

Again Off heap memory = 1GB (If we enable it)

Final Numbers are -

On-heap memory = 10GB
Off heap memory = 1GB
Overhead memory = 1GB

NOW YOU MIGHT BE THINKING ??

WE ALLOCATED 10GB MEMORY TO ON-HEAP MEMORY & HAVE NOT ALLOCATED ANYTHING TO OFF-HEAP AND OVERHEAD MEMORY THEN HOW DOES THESE TWO PORTIONS GETS MEMORY ? 

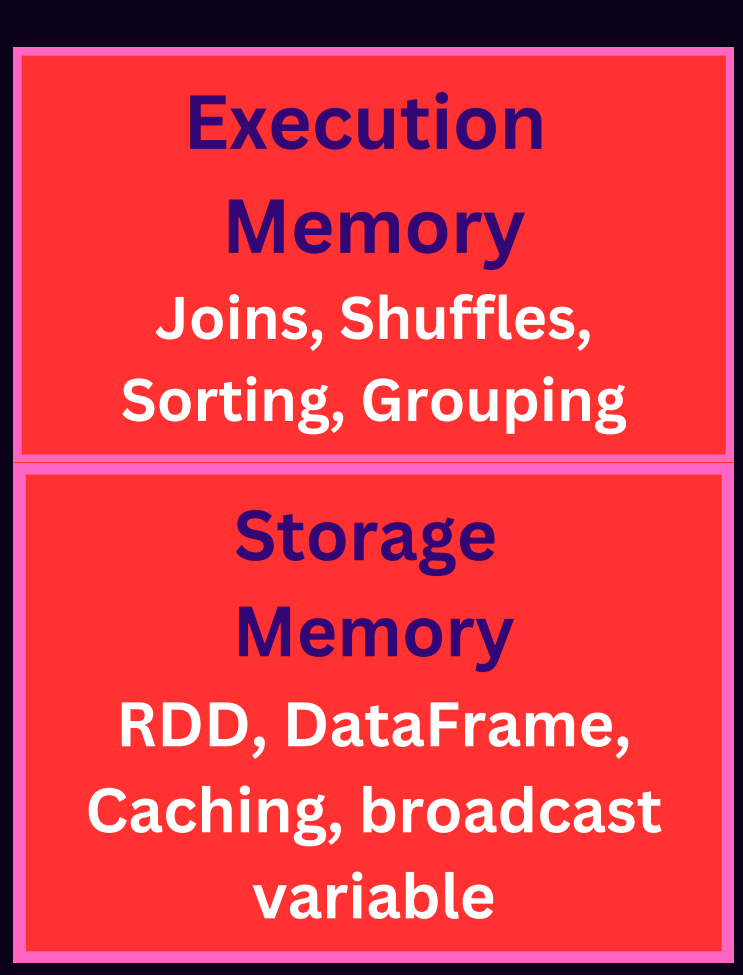
Answer - Executor allocates memory only to on-heap but when spark requests the cluster manager (YARN) for memory it is going to request added offheap and overhead memory as

10GB+1GB+1GB =12GB

Hence It will request 12GB of memory

Unified Memory

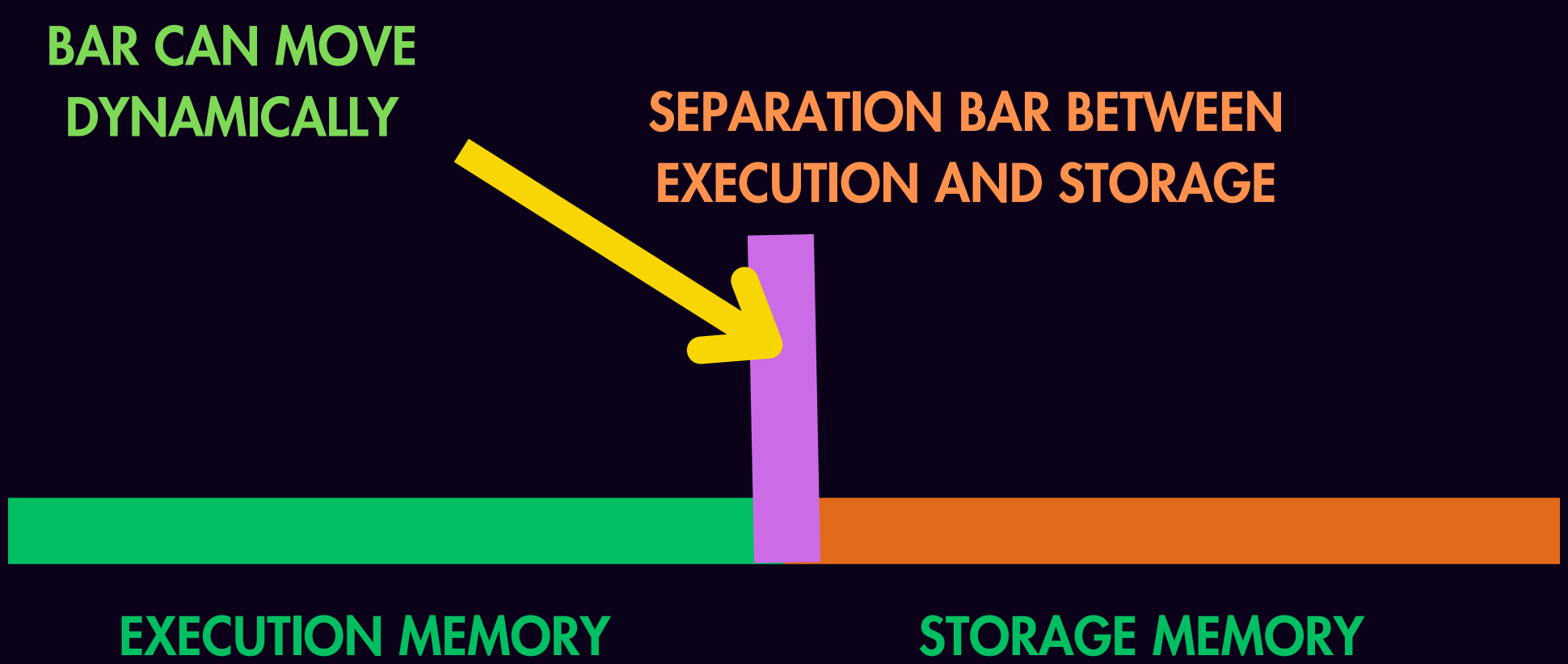
Execution and Storage memory together termed as Unified memory. They both combined termed as unified because of Dynamic memory strategy.



COMBINELY KNOWN AS UNIFIED MEMORY

What is this Dynamic memory Strategy ?

This feture allows to adjust memory between both of them
If Execution wants more memory It can simply use some of the storage memory & If storage memory wants more memory it can use execution memory



As represented in above diagram this separation bar can move as per the requirement from execution or storage memory.

Small Request-

If you find my content helpful like, comment and reshare with others
it will surely help other and mine too.

Vishal Gupta
Data Engineer
vishal.sss202@gmail.com

Vishal Gupta