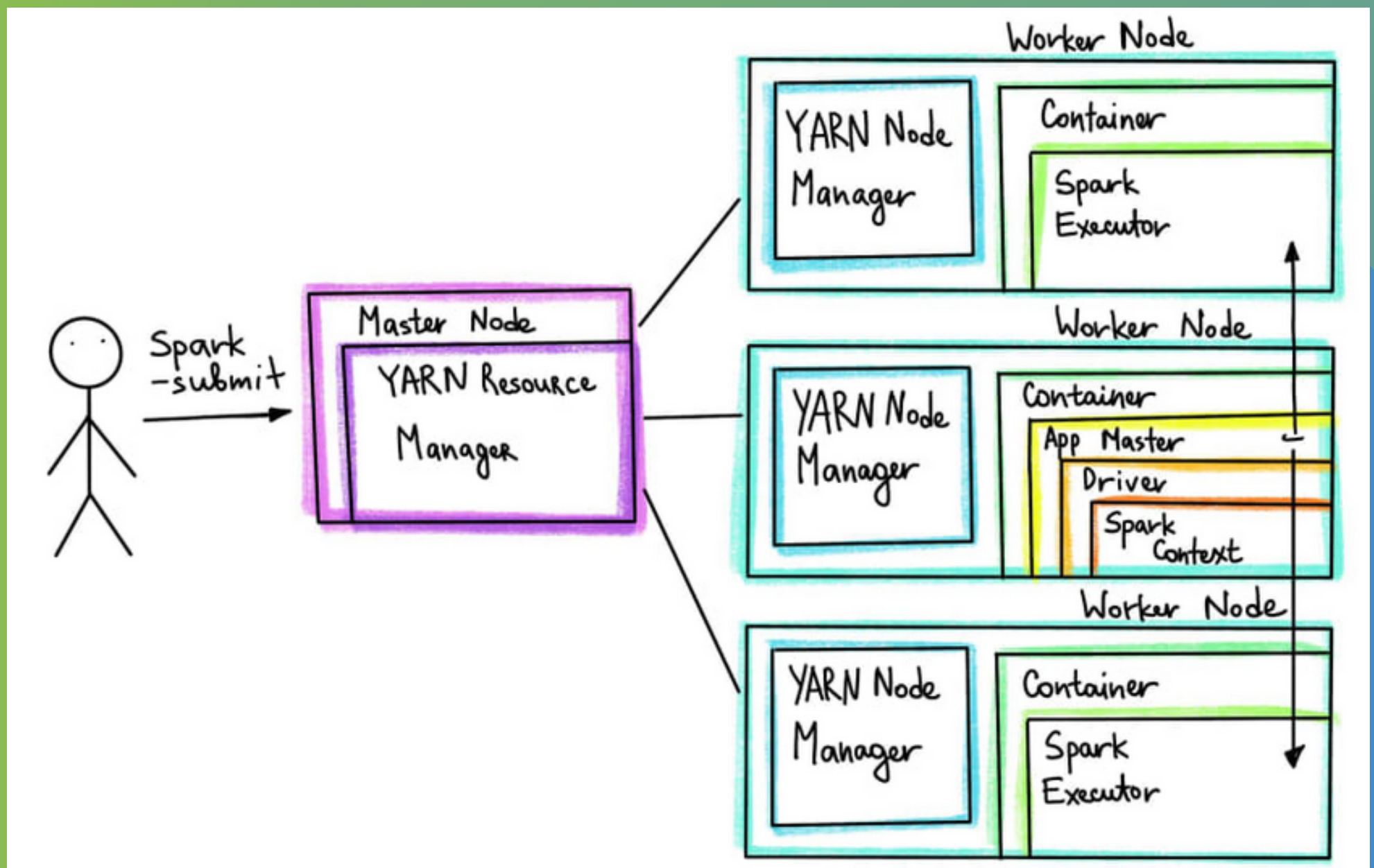


you submitted a Spark application in yarn cluster, the yarn resource manager will allocate an application master container and start the driver JVM in the container. The driver will start with some memory allocation, which you requested. You can ask for the driver memory using two configurations:

1. **spark.driver.memory**
2. **spark.driver.memoryOverhead**

Driver Memory Allocation

Assuming that you have set the Spark driver memory to 1 GB, and the default value for spark.driver.memoryOverhead is the maximum of 10% of the Spark driver memory or 384 MB, the total memory of the container will be approximately 1 GB + 384 MB. This means that the driver process will have 1 GB of JVM heap memory available to it, while the remaining 384 MB will be used for overhead memory.

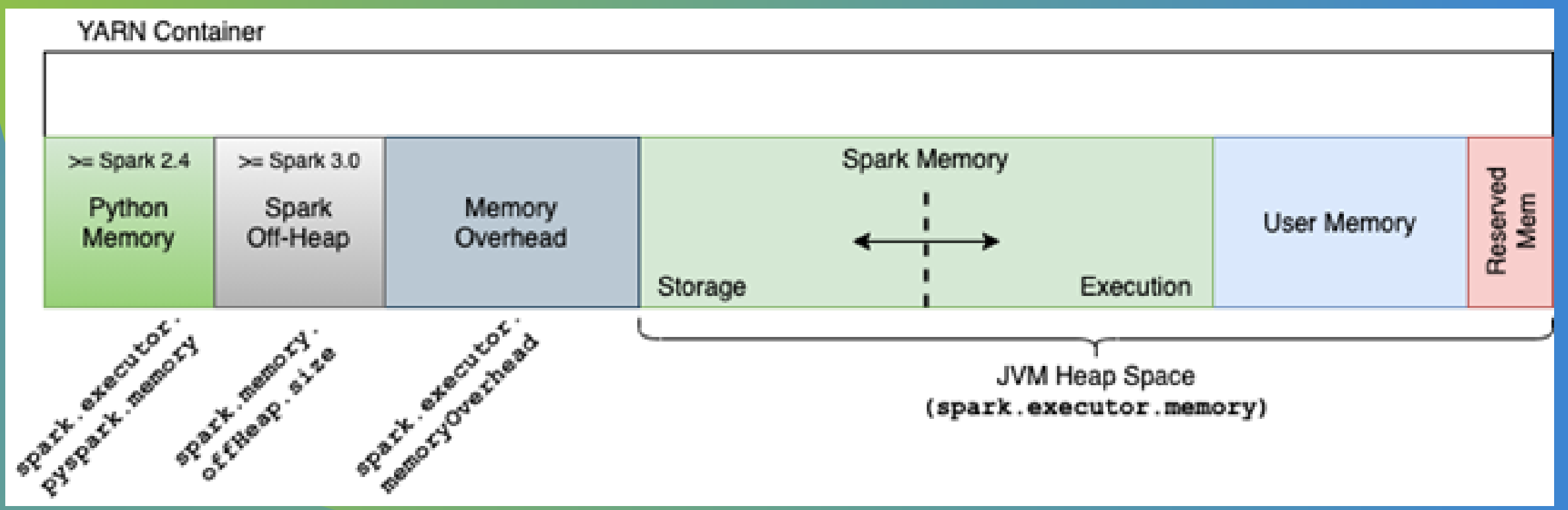
The purpose of this overhead memory is to ensure that the container has enough resources to run efficiently and avoid out-of-memory errors. The driver process itself will use all of the JVM heap memory allocated to it, but none of the overhead memory. Therefore, it's important to set the spark.driver.memoryOverhead parameter appropriately based on the size of the driver memory and the requirements of the application.

https://www.linkedin.com/in/mrabhijitsahoo/

Executor Memory Allocation

After the driver has requested executable containers from YARN, the YARN resource manager will allocate a set of containers. The amount of memory allocated to each container is the sum of several components:

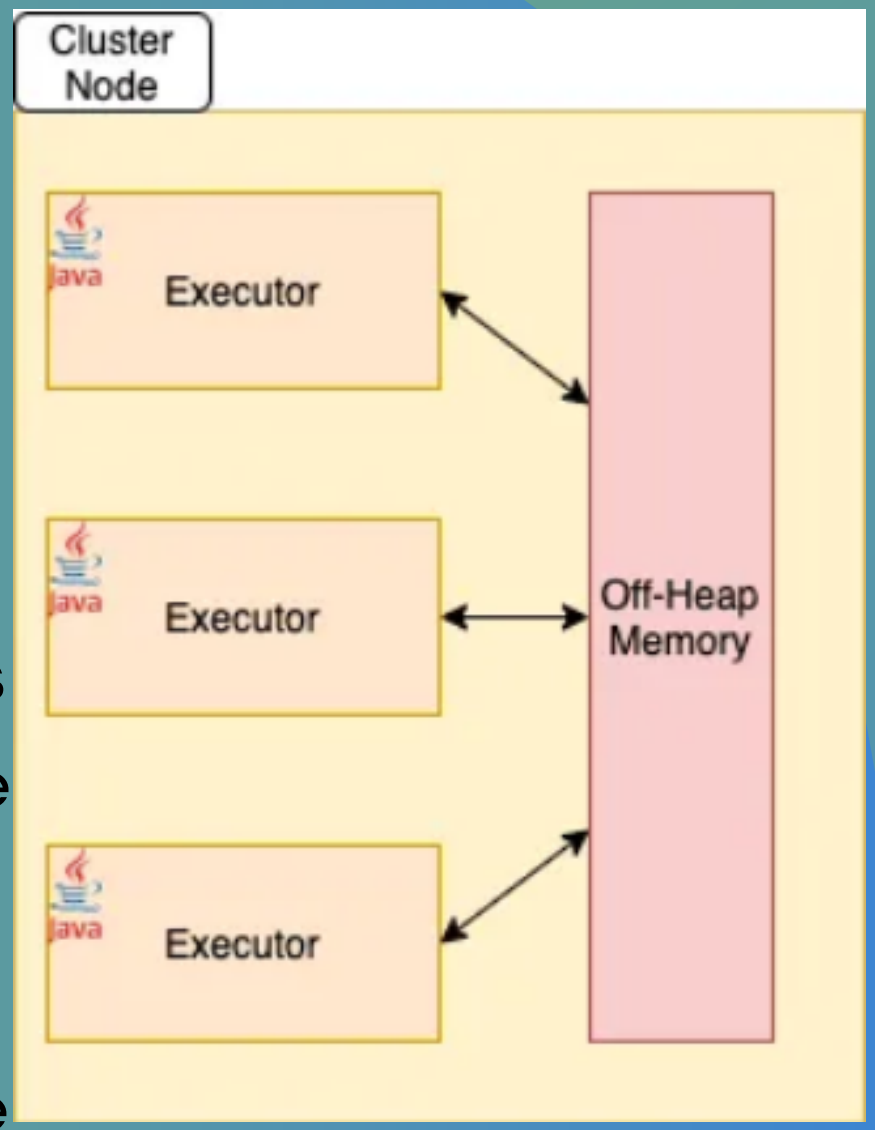
1. Off heap Memory - **spark.memory.offheap.size**
2. Overhead Memory - **spark.executor.memoryOverhead**
3. Pyspark Memory - **spark.executor.pyspark.memory**
4. heap Memory - **spark.executor.memory**



[Total Memory Request to YARN for each container]

1. off-heap Memory :

- Apache Spark uses off-heap memory to store data outside of the Java Virtual Machine (JVM) heap. This allows Spark to allocate and manage larger amounts of memory than it would be able to using the JVM heap alone.
- Off-heap memory is used for tasks such as caching, serialization, and managing large data structures. By storing this data outside of the JVM heap, Spark can avoid the overhead of garbage collection on the JVM heap, which can be time-consuming and can cause performance issues in large-scale applications.



[JVM Heap vs Off-Heap Memory]

- The off-heap memory is managed separately from the JVM heap and can be configured using the **spark.memory.offHeap.enabled** and **spark.memory.offHeap.size** configuration parameters.

2. overhead Memory :

- Memory overhead in Apache Spark refers to the additional memory required by the executor beyond the user-defined executor memory.
- This overhead is used to store metadata, internal data structures(such as bookkeeping information for shuffle operations and broadcast variables) and for inter-process communication (IPC) between the executor and driver. It is also used for shuffle exchange or network read buffer.In other words, it is the memory required for the Spark framework to function properly.
- This is determined by the **spark.executor.memoryOverhead** parameter.By default, 'spark.executor.memoryOverhead' is the maximum of 10% of the executor memory or 384 MB, whichever is larger.

The memory overhead is separate from the executor memory, but it is still allocated from the same memory pool as the executor memory. When configuring the executor memory, you need to take into account both the executor memory and the memory overhead to ensure that the executor has enough memory to run your Spark application.

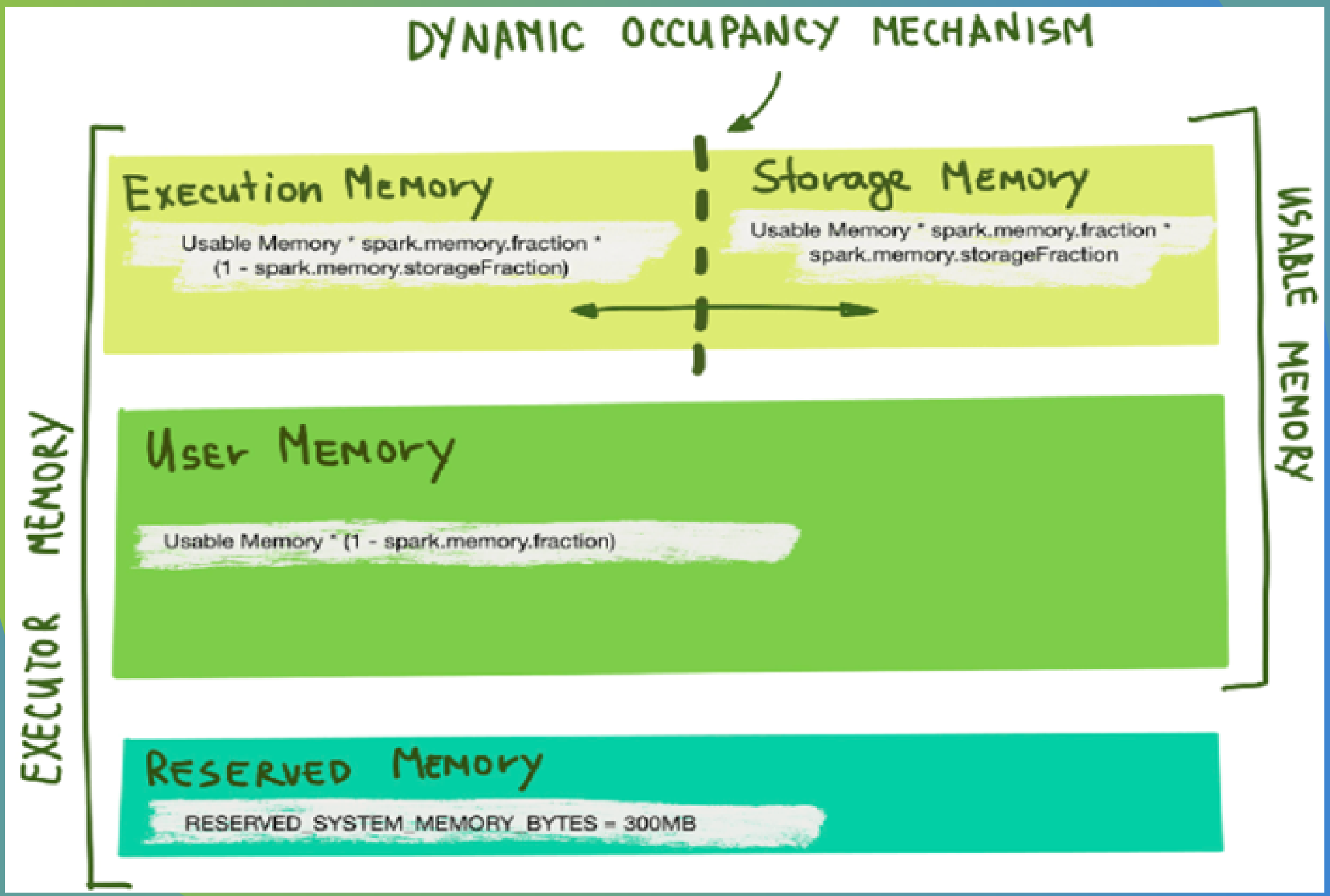
3. PySpark Memory :

- When you use PySpark, your application may require a Python worker to execute Python code on each node in the Spark cluster. These Python workers cannot use the JVM heap memory as they are not part of the JVM runtime environment. Instead, they use off-heap memory to store data and intermediate results.
- If your PySpark application requires more off-heap memory than the default configuration, you can set the extra memory requirement using the **spark.executor.pyspark.memory** configuration property. However, note that Spark does not have a default value for this property, as most PySpark applications do not require additional memory.

4. Heap Memory :

This is the memory specified by the **--executor-memory** flag or the **spark.executor.memory** parameter. It represents the maximum JVM heap memory (Xmx) and is the memory where objects are managed by the garbage collector (GC). This is commonly referred to as Executor Memory.

Deep Dive into JVM Heap memory or executor memory



https://www.linkedin.com/in/mrabhijitsahoo/

One important parameter is **spark.executor.memory** (or **--executor-memory** for spar-submit) which specifies how much memory should be allocated to each executor in the cluster. This memory will split in 3 parts between:
reserved memory, **user memory**, **spark memory** (execution memory + storage memory).

4.a. Reserved memory

Reserved memory is a portion of the executor's memory that is reserved for non-Spark tasks, such as operating system and JVM overheads, off-heap memory, and other miscellaneous tasks. This memory cannot be used by Spark tasks, and it is managed by the executor itself. By default, Spark reserves 300MB of off-heap memory for its internal operations. This value is hardcoded in Spark's source code and is not directly configurable through a user-defined parameter.

4.b. User memory

- User memory is the portion of the executor's memory that is available for use by the user's code. This memory can be used to store data and objects, and it is managed by the user's application.
- You can store here your custom data structure, UDFs, RDD operation directly in your code, RDD lineage & dependency, spark internal metadata.
- If you are not using UDFs you can decrease user memory to give more memory to spark memory, but you can't make it 0 because you need it for metadata & other internal things.
- User memory will calculate as :

  (spark.executor.memory - reserved memory) * (1 - spark.memory.fraction)

4.c. Spark memory

Spark memory is the portion of the executor's memory that is used by Spark tasks. It consists of two regions: *execution memory* and *storage memory*.

i. Execution memory is used to store data that is needed during the execution of Spark tasks. This includes data that is generated during operations such as shuffles, sorts, joins, and aggregations. The amount of execution memory available is determined by the **spark.memory.fraction** parameter, which specifies the fraction of the total Spark memory that should be allocated to execution memory. By default, this value is set to 0.6, which means that 60% of the Spark memory will be used for execution memory.

ii. Storage memory is used to cache data that will be reused across multiple Spark tasks. This includes data that is cached using Spark's RDD or Dataset APIs, as well as broadcast variables. The amount of storage memory available is determined by the **spark.memory.fraction** parameter, as well as the **spark.memory.storageFraction** parameter. **spark.memory.storageFraction** specifies the fraction of the Spark memory that should be used for storage memory. **By default, this value is set to 0.5**, which means that half of the Spark memory allocated will be used for storage memory. The remaining half will be used for execution memory.

Together, the spark.memory.fraction and spark.memory.storageFraction parameters determine how much memory is allocated to execution memory and storage memory, respectively.

The total amount of Spark memory available is calculated as **(spark.executor.memory - reserved memory) * spark.memory.fraction**

and this memory is split between execution memory and storage memory according to spark.memory.storageFraction.

Follow me On in

https://www.linkedin.com/in/mrabhijitsahoo/

FOLLOW