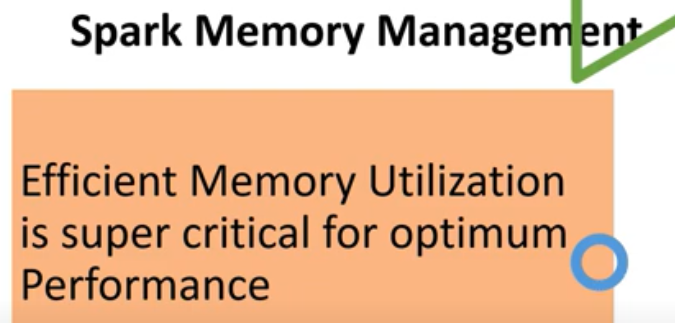
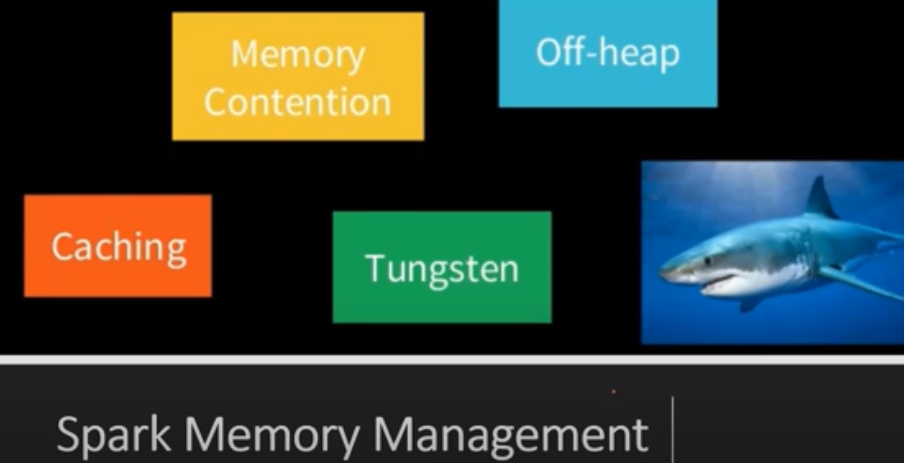
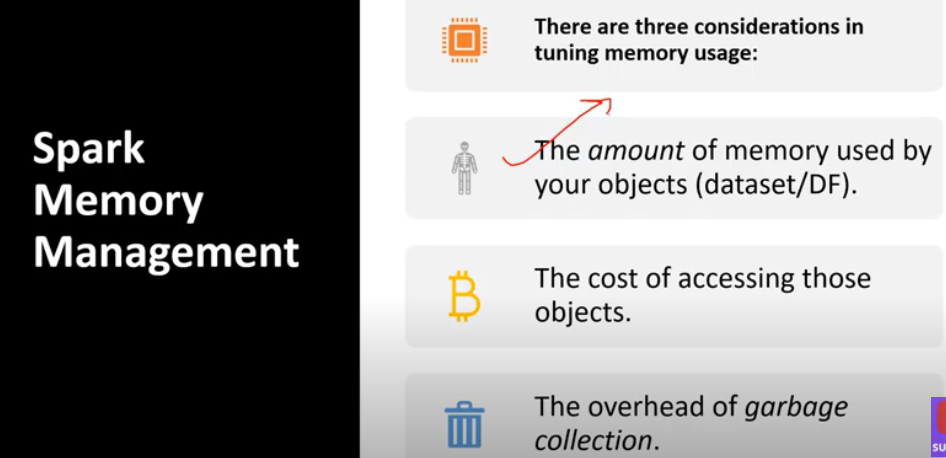
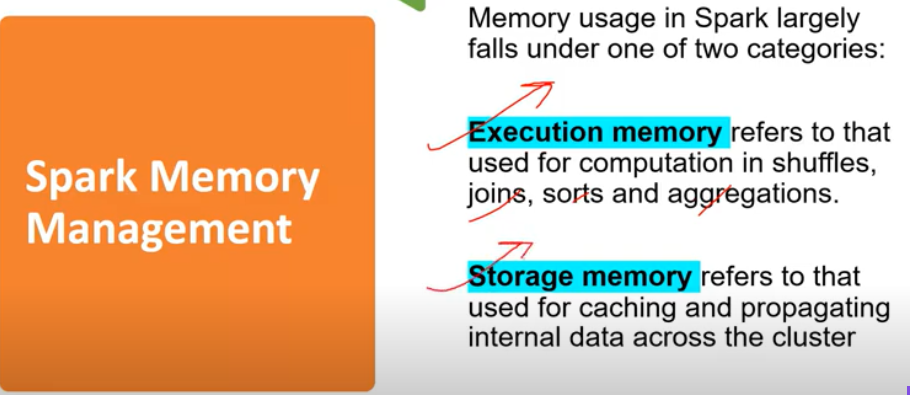
**SPARK MEMORY MANAGEMENT**

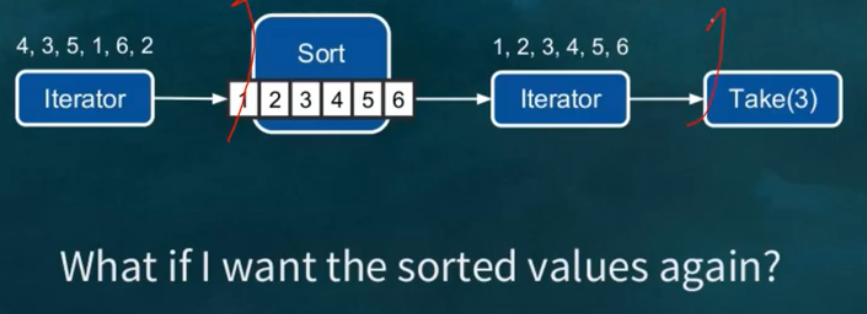


Spark memory related issues:--



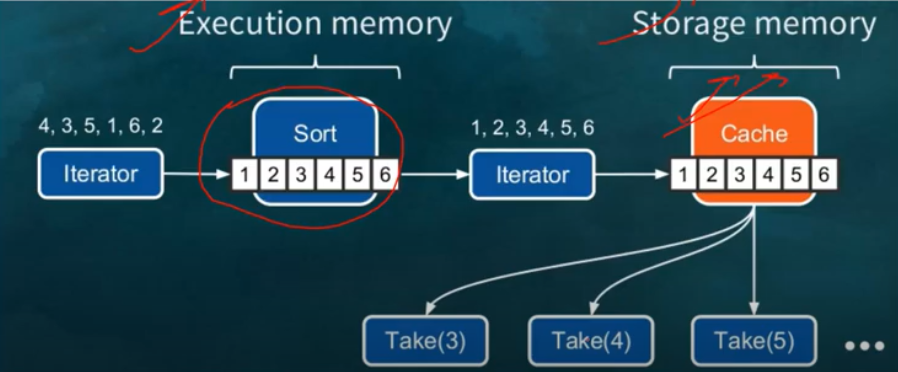


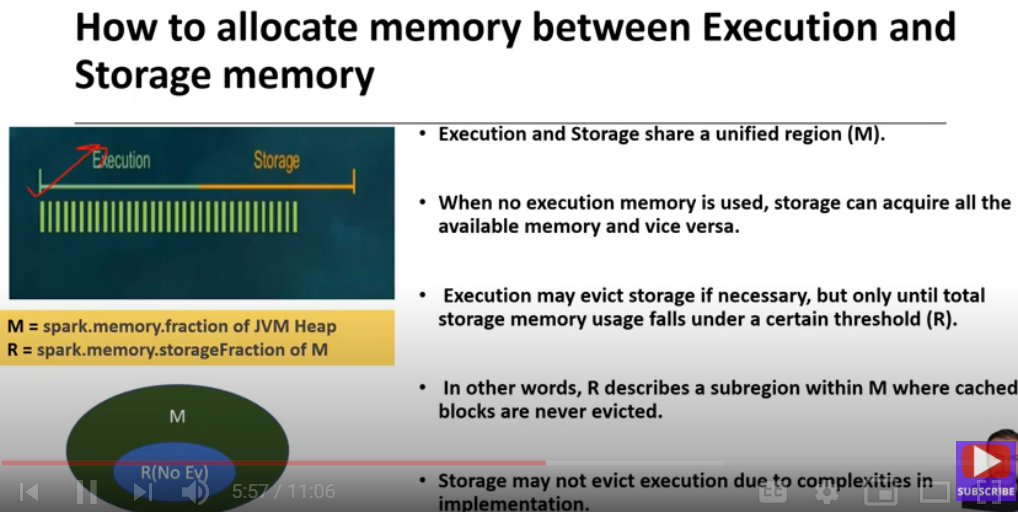




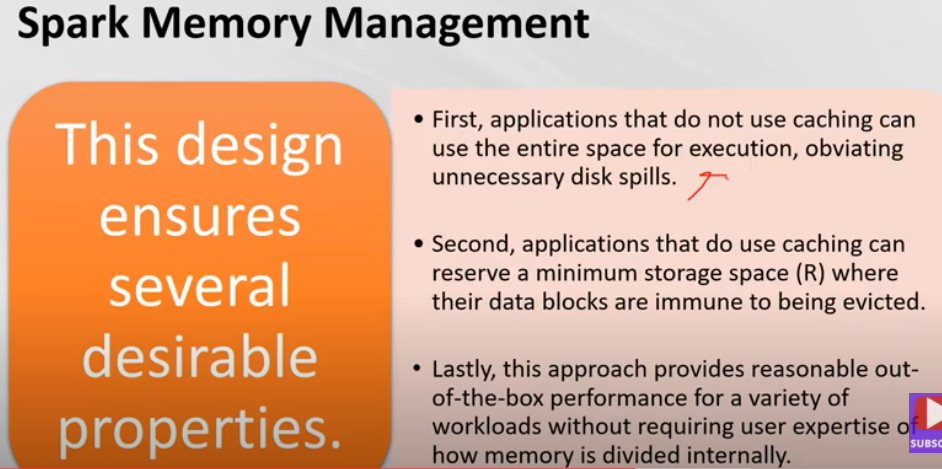
In above example we take 3 values, now we take 4 value or 5 values after that so we compute whole again again but its not a god approach so we cache that intermediate output so we include cache .

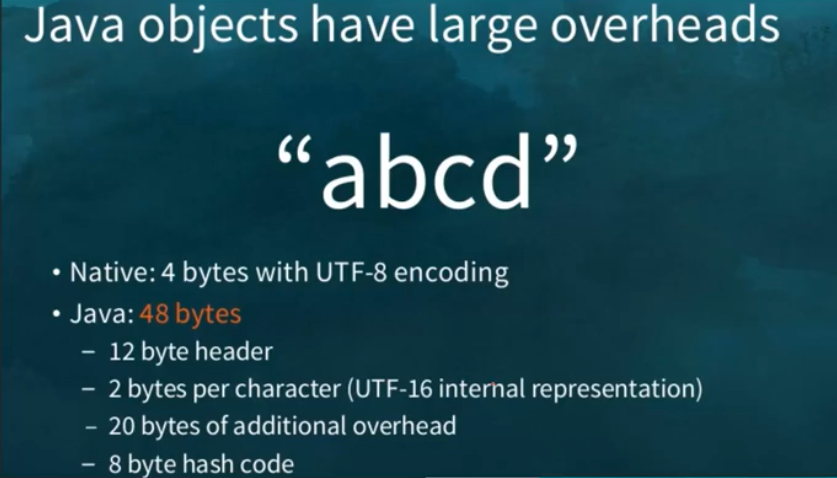
The memory used to perform operation like in above example sort , the memory used is executor memory and memory used for cache is storage memory.

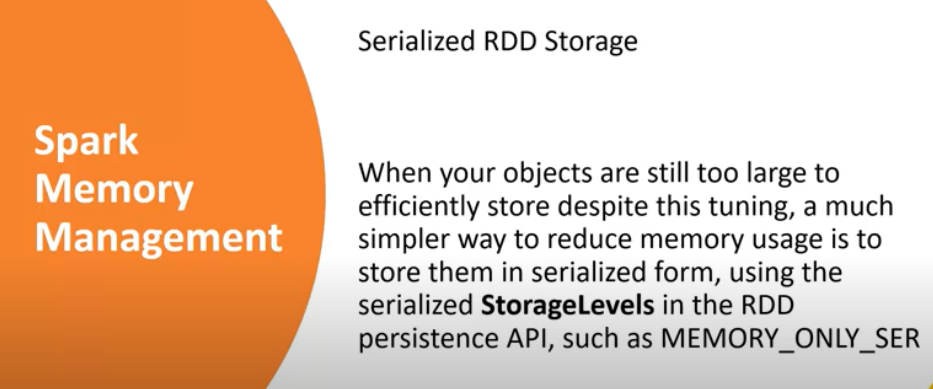


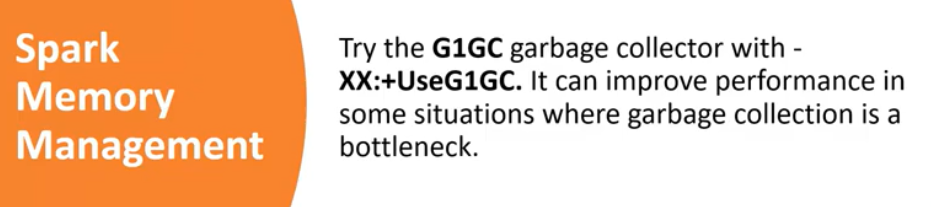


The execution memory can evict the storage of storage memory only when total storage memory usage falls under a certain threshold, as execution memory evict the block on the basis of LRU algorithm . all you least recently blocks removed from the storage and add in the disk.





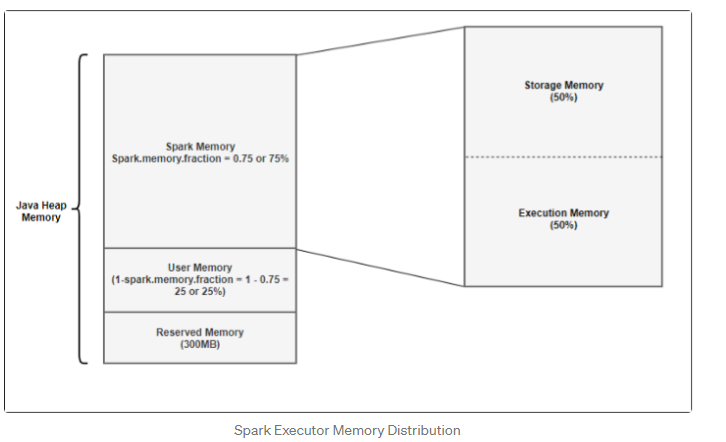




# Executor memory breakdown

Prior to spark 1.6, mechanism of memory management was different, this article describes about memory management in spark version 1.6 and above. Earlier memory management was done using StaticMemoryManager class, in spark 1.6 and above, memory management is done by UnifiedMemoryManager class.

Following diagram shows how the memory is segregated in Spark executor



For the sake of understanding, we will take an example of 4GB Memory allocated to an executor and leave the default configuration and see how much memory each segment gets.

Two main configurations that control executor memory allocation:

spark.memory.fraction — defaults to 0.75

spark.memory.storageFraction — defaults to 0.5

## **1. Reserved Memory**

This is the memory reserved by the system, and its size is hardcoded. As of Spark 1.6.0, its value is 300MB, which means that this 300MB of RAM cannot be changed unless spark is recompiled. This memory stores sparks internal objects. One thing to note is that, if the executor memory is less than 1.5 times of reserved memory, Spark will fail with a “please use larger heap size” error message.

Formula : **Reserved Memory = 300MB**

Calculation for 4GB : **Reserved Memory = 300MB**

## **2. User Memory**

This is the memory area that stores all the user defined data structures, any UDFs created by the user etc,. This memory segment is not managed by spark, spark will not be aware of/maintain this memory segment. The size of this memory pool can be calculated as (Java Heap — Reserved Memory) \* (1.0 — spark.memory.fraction), which is by default equal to (Java Heap — 300MB) \* 0.25.

Formula : **User Memory =** **(Java Heap — Reserved Memory) \* (1.0 — spark.memory.fraction)**

Calculation for 4GB : **User Memory = (4024MB — 300MB) \* (1.0–0.75) = 949MB**

## **3. Spark Memory**

This memory pool is managed by Spark. This is responsible for storing intermediate state while doing task execution like joins or to store the broadcast variables. All the cached/persisted data will be stored in this segment, specifically in the storage memory of this segment.

Formula : **(Java Heap — Reserved Memory) \* spark.memory.fraction**

Calculation for 4GB : **(4096MB -300MB) \* 0.75 = 2847MB**

This is broken into 2 segments Storage Memory and Execution Memory. We will briefly discuss these two segments:

**Storage Memory:**

Storage memory is used for storing all of the cached data, broadcast variables are also stored here. Any persist option which includes MEMORY in it, spark will store that data in this segment. Spark clears space for new cache requests by removing old cached objects based on Least Recently Used (LRU) mechanism. Once the cached data it is out of storage, it is either written to disk or recomputed based on configuration. Broadcast variables are stored in this segment with MEMORY\_AND\_DISK persistent level.

Formula: **Storage Memory = (Java Heap — Reserved Memory) \* spark.memory.fraction \* spark.memory.storageFraction**

Calculation for 4GB : **Storage Memory = (4096MB — 300MB) \* 0.75 \* 0.5 = ~1423MB**

**Execution Memory:**

This memory region is used by Spark for objects created during execution of a task. For example,it is used to store hash table for hash aggregation step, it is used to store shuffle intermediate buffer on the Map side in memory etc,. This pool also supports spilling on disk if not enough memory is available, but the blocks from this pool cannot be forcefully evicted by other tasks.

Formula: **Execution Memory = (Java Heap — Reserved Memory) \* spark.memory.fraction \* (1.0 — spark.memory.storageFraction)**

Calculation for 4GB : **Execution Memory = (4096MB — 300MB) \* 0.75 \* (1.0 — 0.5) = ~1423MB**

In spark 1.6 and above, there is no hard boundary between Execution memory and Storage memory. Due to the nature of Execution memory, blocks cannot be forcefully evicted from this pool, otherwise execution will break since the block it refers to won’t be found. But when it comes to Storage memory, blocks can be evicted from memory and written to disk or recomputed(if persistence level is MEMORY\_ONLY) as required. Few things to keep in mind about storage and execution memory:

1.Storage memory can borrow space from execution memory only if blocks are not used in Execution memory.

2.Execution memory can also borrow space from Storage memory if blocks are not used in Storage memory.

3.If blocks from Execution memory is used by Storage memory and Execution needs more memory, it can forcefully evict the excess blocks occupied by Storage Memory

4.If blocks from Storage Memory is used by Execution memory and Storage needs more memory, it cannot forcefully evict the excess blocks occupied by Execution Memory, it will end up having less memory area. It will wait until Spark releases the excess blocks stored by Execution memory and then occupies them.

## **Memory breakdown for 4GB executor memory assignment**

Reserved Memory — 300MB — 7.32%

User Memory — 949MB — 23.16%

Spark Memory — 2847MB —69.5%

This is the total memory break down, if you like to know what would be the space available to store your cached data (note that there is no hard boundary, this is the initial allocation):

Initial Storage Memory (50% of spark memory) — 1423MB — 34.75%

The percentage value is only for 4GB executor memory calculation, for a different executor memory configuration, these won’t hold good. It is only given for understanding purposes.