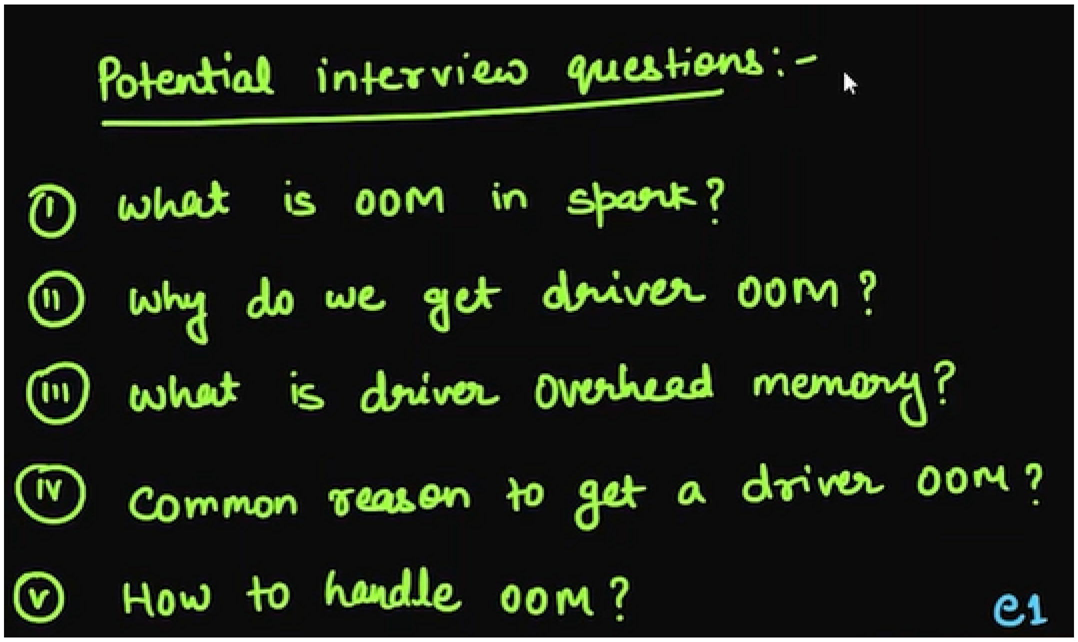
Spark Memory Management - Driver out of memory spark

1. Potential interview questions
2. 

Potential interview questions:-

(i) what is OOM in spark?

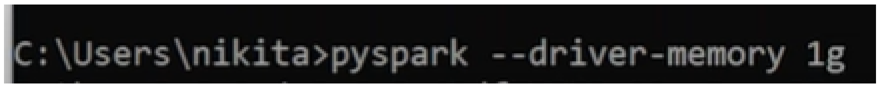
(ii) why do we get driver OOM?

(iii) what is driver overhead memory?

(iv) Common reason to get a driver OOM?

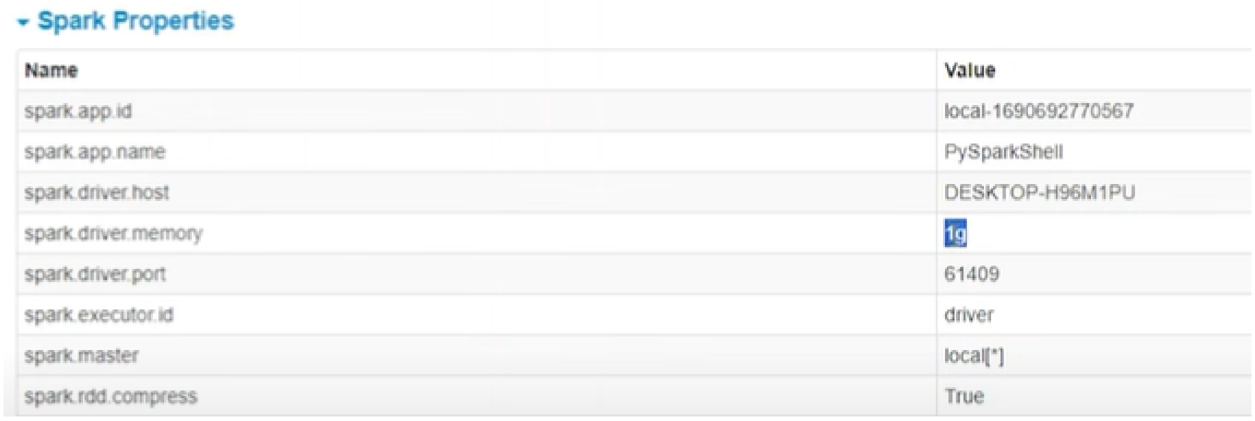
(v) How to handle OOM?

3. Lets learn this in practical way
4. Here we have initialized our driver-memory to 1gb in our local setup



```
C:\Users\nikita>pyspark --driver-memory 1g
```

5. And now if we go and check in spark ui..we can see it assigned 1 gb to out driver

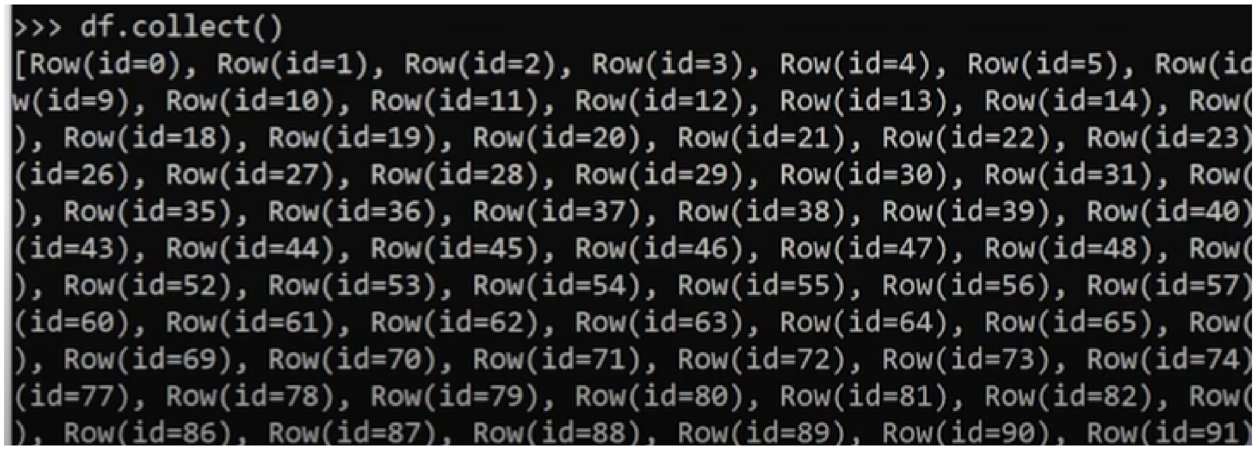


▾ Spark Properties

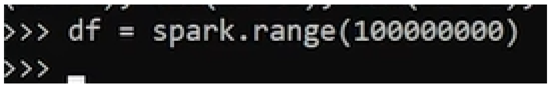
Name	Value
spark.app.id	local-1690692770567
spark.app.name	PySparkShell
spark.driver.host	DESKTOP-H96M1PU
spark.driver.memory	1g
spark.driver.port	61409
spark.executor.id	driver
spark.master	local[*]
spark.rdd.compress	True

6. Now to get OOM..we create a df of 100 rows

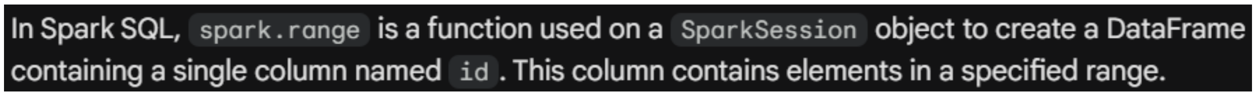
7. To see all the rows ..we use df.collect()



```
>>> df.collect()
[Row(id=0), Row(id=1), Row(id=2), Row(id=3), Row(id=4), Row(id=5), Row(id
w(id=9), Row(id=10), Row(id=11), Row(id=12), Row(id=13), Row(id=14), Row(
), Row(id=18), Row(id=19), Row(id=20), Row(id=21), Row(id=22), Row(id=23)
(id=26), Row(id=27), Row(id=28), Row(id=29), Row(id=30), Row(id=31), Row(
), Row(id=35), Row(id=36), Row(id=37), Row(id=38), Row(id=39), Row(id=40)
(id=43), Row(id=44), Row(id=45), Row(id=46), Row(id=47), Row(id=48), Row(
), Row(id=52), Row(id=53), Row(id=54), Row(id=55), Row(id=56), Row(id=57)
(id=60), Row(id=61), Row(id=62), Row(id=63), Row(id=64), Row(id=65), Row(
), Row(id=69), Row(id=70), Row(id=71), Row(id=72), Row(id=73), Row(id=74)
(id=77), Row(id=78), Row(id=79), Row(id=80), Row(id=81), Row(id=82), Row(
), Row(id=86), Row(id=87), Row(id=88), Row(id=89), Row(id=90), Row(id=91)
```

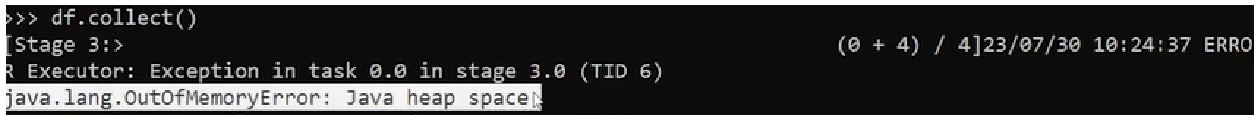
8. Now we have create a df with 1m rows  using spark.range

```
>>> df = spark.range(100000000)
>>>
```



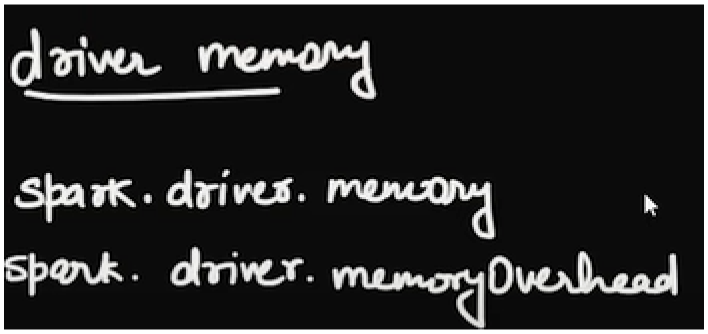
In Spark SQL, `spark.range` is a function used on a `SparkSession` object to create a DataFrame containing a single column named `id`. This column contains elements in a specified range.

9. Now if we do df.collect()..to show 1m rows.
10. Then we get OOM error



```
>>> df.collect()
[Stage 3:>                                                   (0 + 4) / 4]23/07/30 10:24:37 ERRO
R Executor: Exception in task 0.0 in stage 3.0 (TID 6)
java.lang.OutOfMemoryError: Java heap space
```

11. Lets learn how it is happening
12. In driver memory…we have 2 types of memory

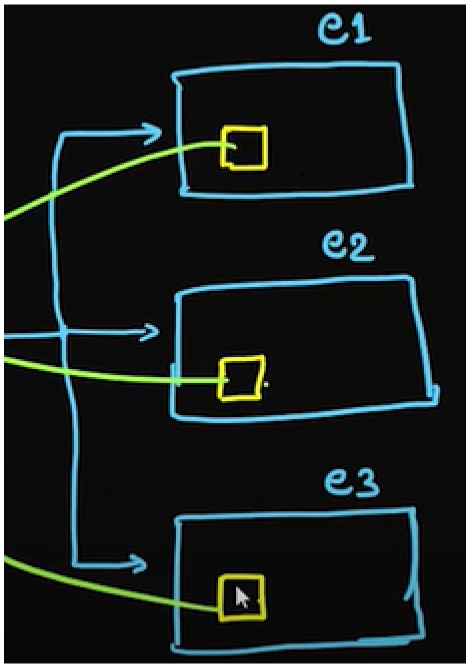


driver memory

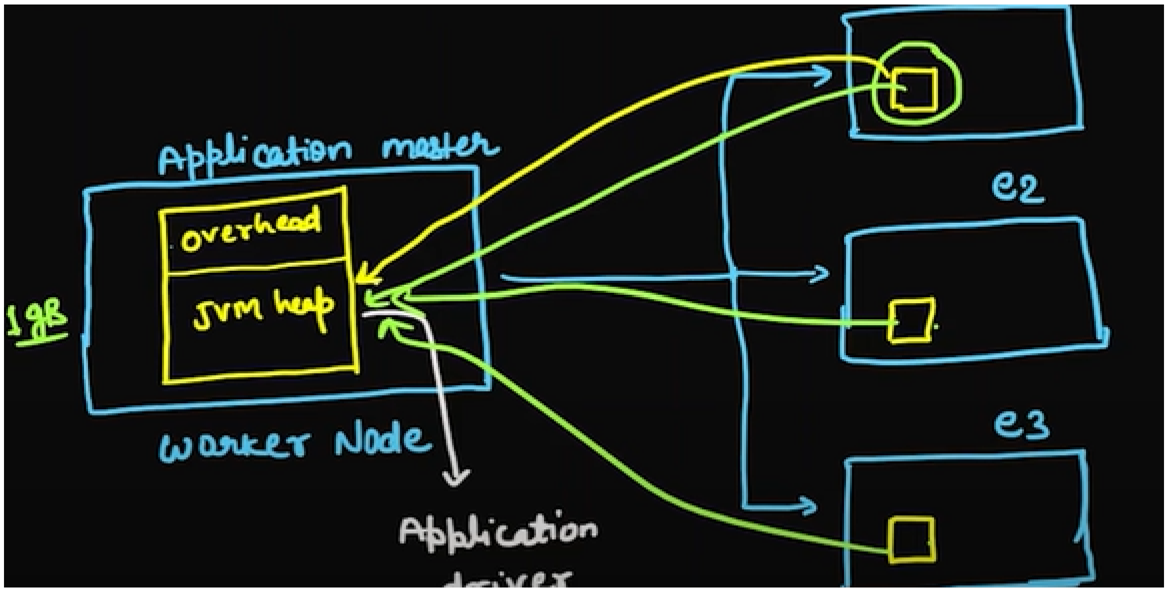
spark.driver.memory

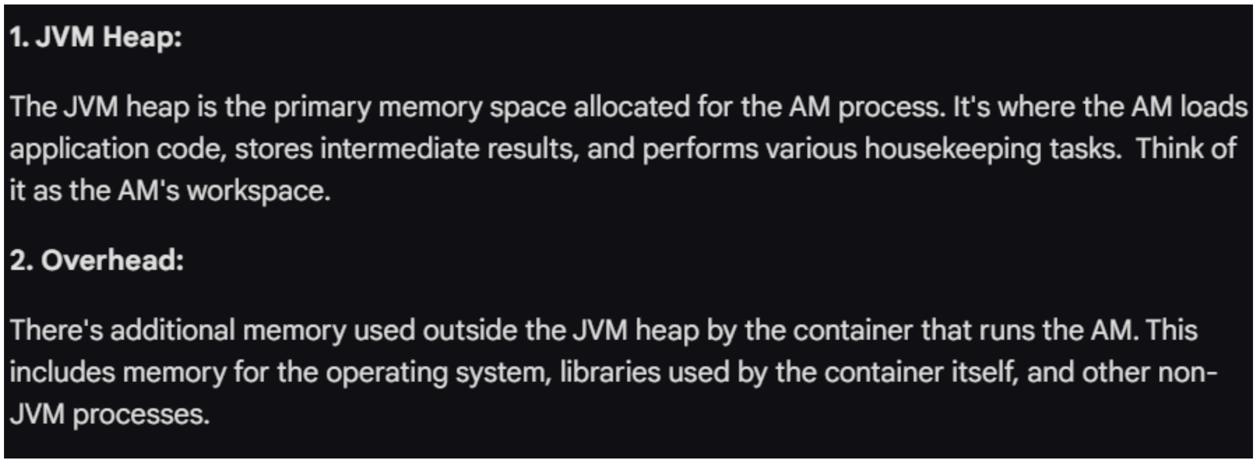
spark.driver.memoryOverhead

13. Now lets see why show() didn't gave error and y collect() gave error
14. Assume our 1m rows of data has partitioned and split into 3 executor nodes



15. Now whenever we hit collect..then these partitions will move to drive memory..
16. But as the driver memory is 1gb and all partitions are of 1.5gb..then we get driver OOM error
17. But in the case of show()...it only moves 1st partition data..as we need only few rows



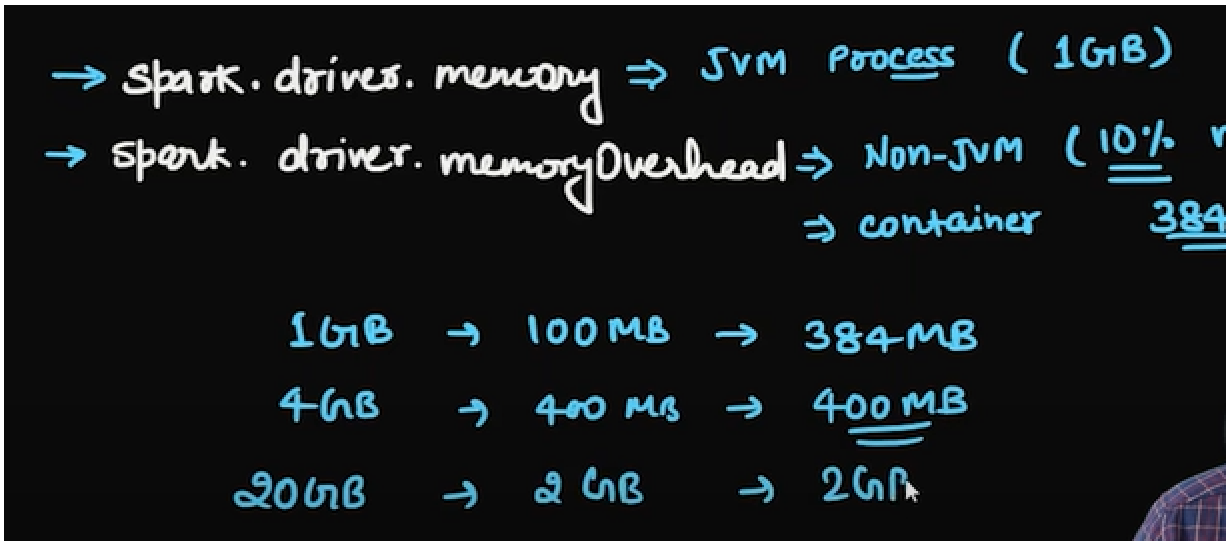
18. 

1. JVM Heap:

The JVM heap is the primary memory space allocated for the AM process. It's where the AM loads application code, stores intermediate results, and performs various housekeeping tasks. Think of it as the AM's workspace.

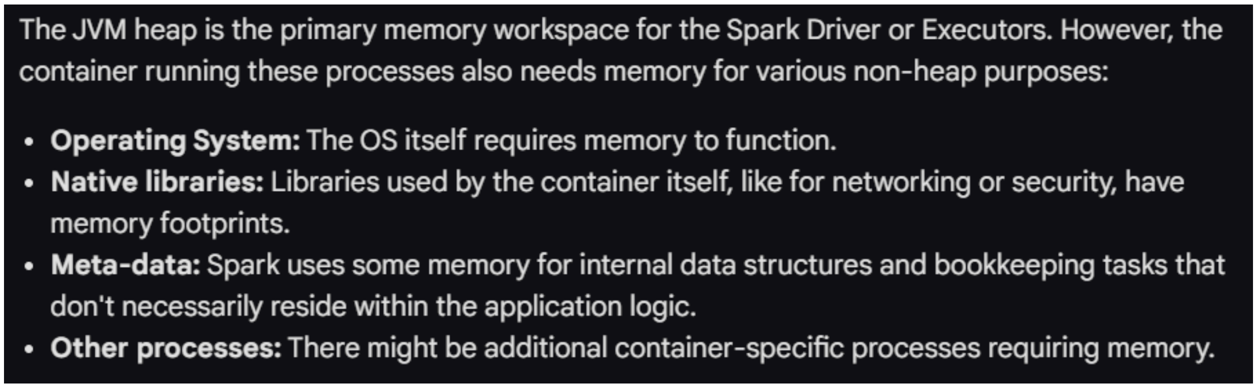
2. Overhead:

There's additional memory used outside the JVM heap by the container that runs the AM. This includes memory for the operating system, libraries used by the container itself, and other non-JVM processes.

19. 

→ Spark.driver.memory ⇒ JVM Process (1GB)
→ spark.driver.memoryOverhead ⇒ Non-JVM (10%
⇒ container 384
1GB → 100MB → 384MB
4GB → 400 MB → 400 MB
20GB → 2 GB → 2GB

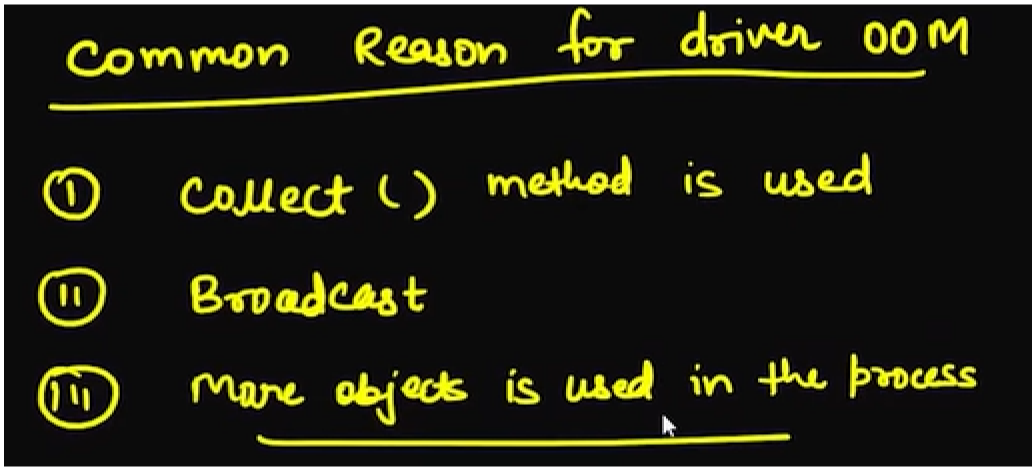
20. Here if we assign 1gb of memory to driver..then 10% or 384MB which ever is higher…goes to memory overhead for performing



The JVM heap is the primary memory workspace for the Spark Driver or Executors. However, the container running these processes also needs memory for various non-heap purposes:

- **Operating System:** The OS itself requires memory to function.
- **Native libraries:** Libraries used by the container itself, like for networking or security, have memory footprints.
- **Meta-data:** Spark uses some memory for internal data structures and bookkeeping tasks that don't necessarily reside within the application logic.
- **Other processes:** There might be additional container-specific processes requiring memory.

21. Common reasons to get OOM

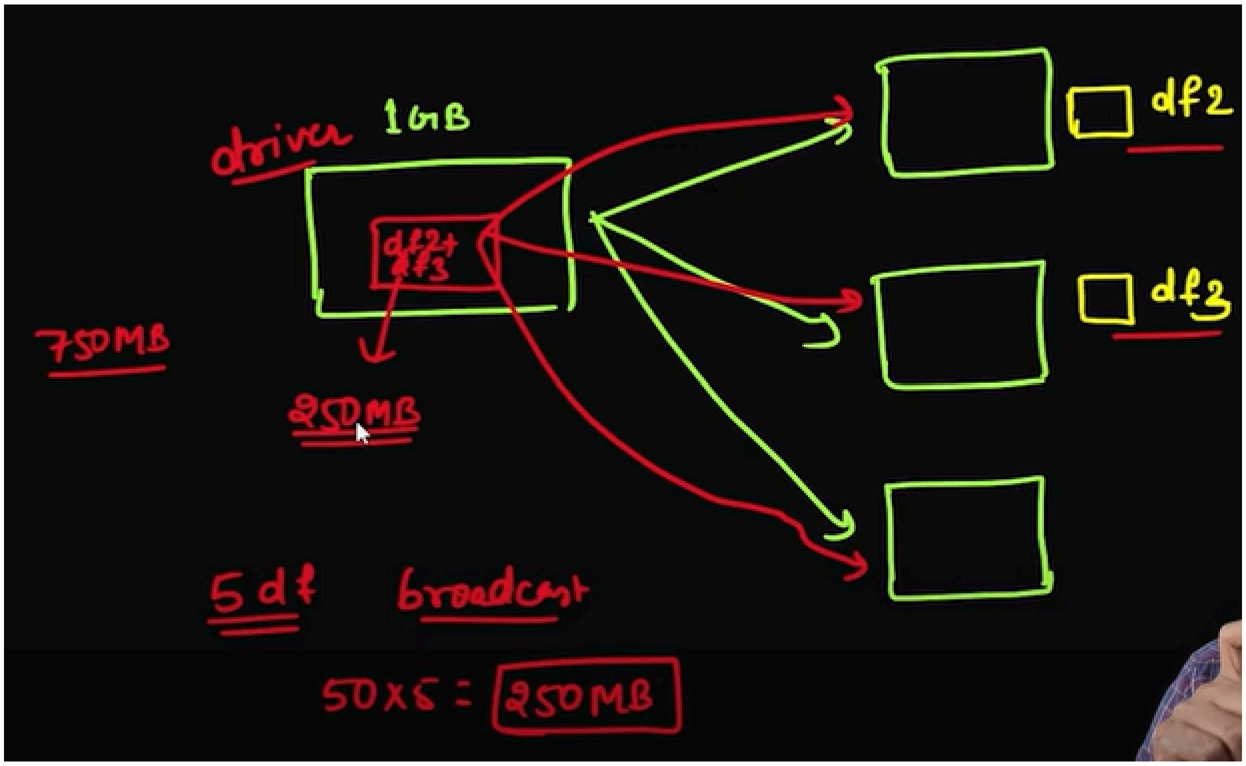
22. 

Common Reason for driver OOM

(i) Collect ( ) method is used

(ii) Broadcast

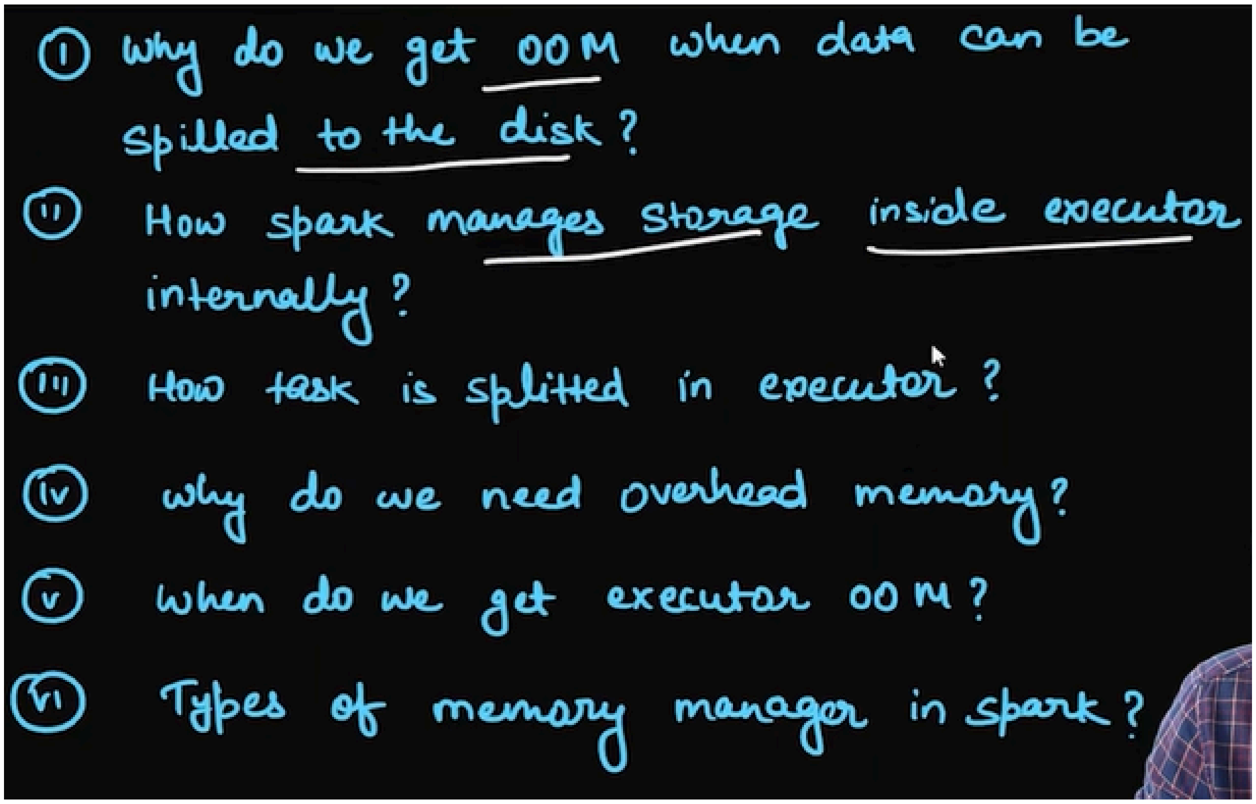
(iii) More objects is used in the process

23. If our broadcast join size is greater than driver memory size..then we may get OOM
24. 

25. Here if we want to join two dataframe and broadcast it to executors…then if these two df size is less than driver memory..then we dont get any issue
26. But if the df’s size exceeds the driver memory..then we get OOM
27. And also if by mistake we configured overhead memory less than 10%..then we may get OOM error
28. Common reasons explained : https://g.co/gemini/share/94b1d7105aa8

Executor OOM

1. Potential Interview Questions



(i) why do we get OOM when data can be spilled to the disk ?

(ii) How spark manages storage inside executor internally ?

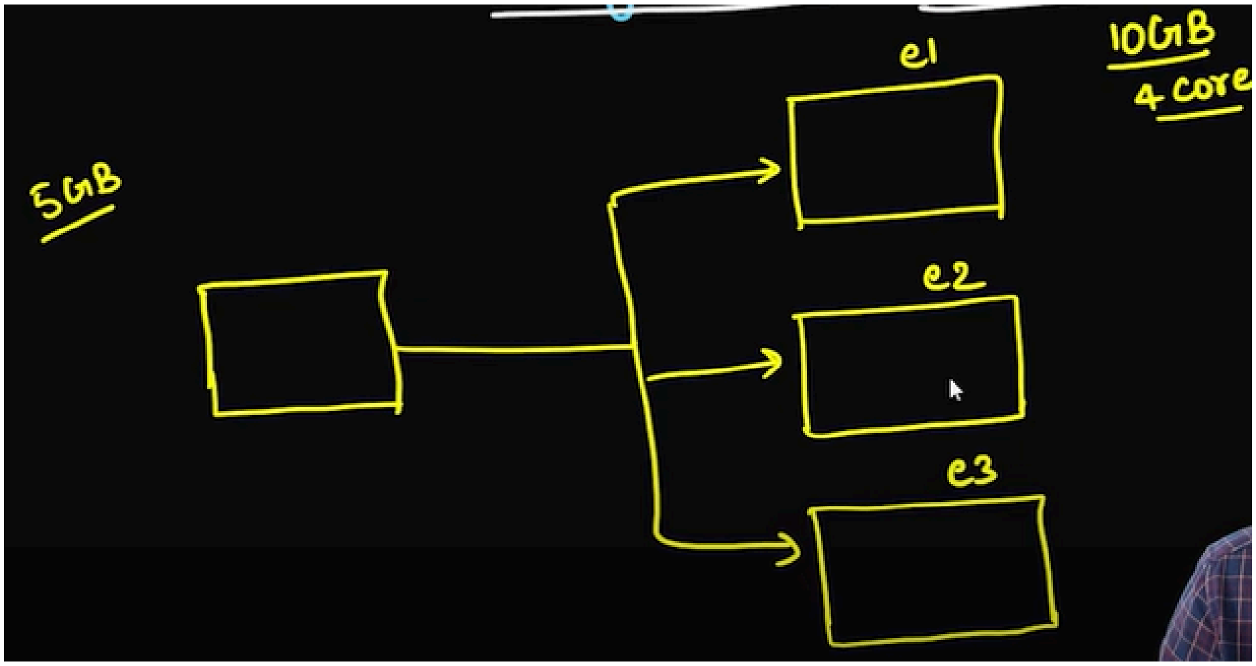
(iii) How task is splitted in executor ?

(iv) why do we need overhead memory?

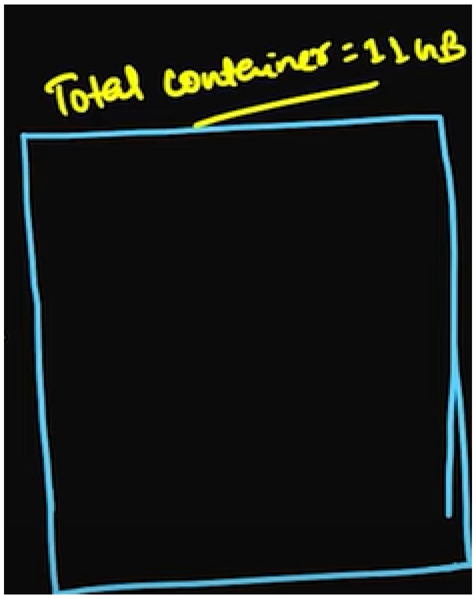
(v) when do we get executor OOM ?

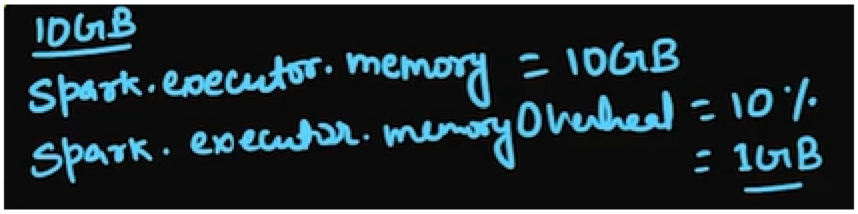
(vi) Types of memory manager in spark ?

2. Lets assume we have driver node of 5GB and 3 executors each with 10gb storage and 4 core

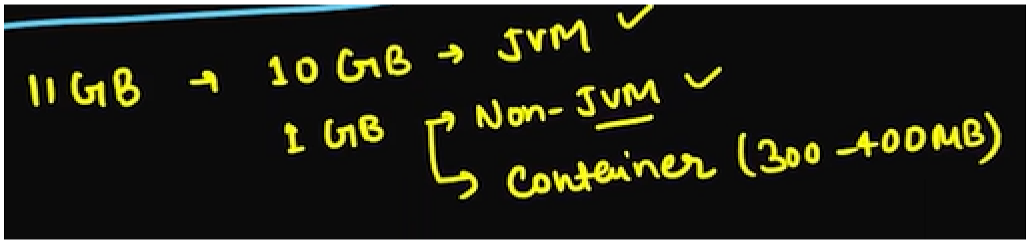


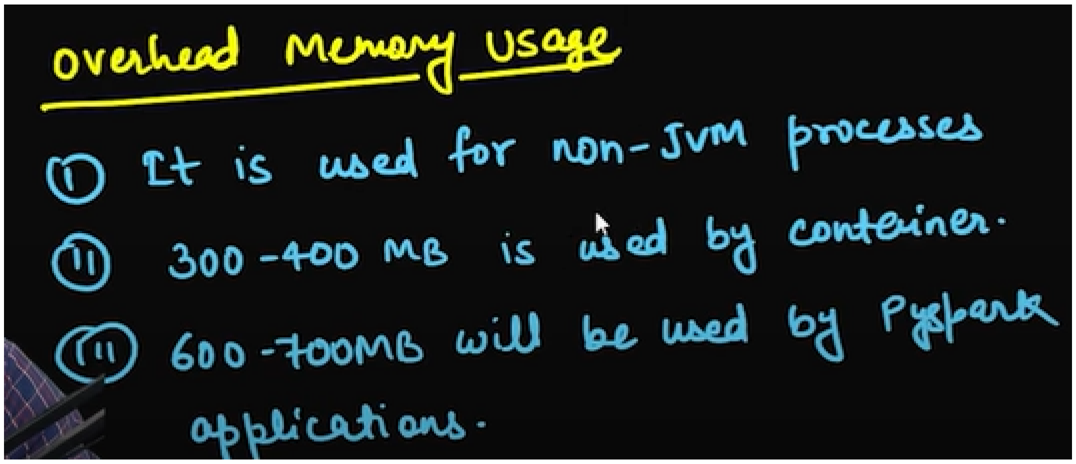
3. Lets deep dive into one executor

4. 


10GB
spark.executor.memory = 10GB
spark.executor.memoryOverhead = 10%
= 1GB

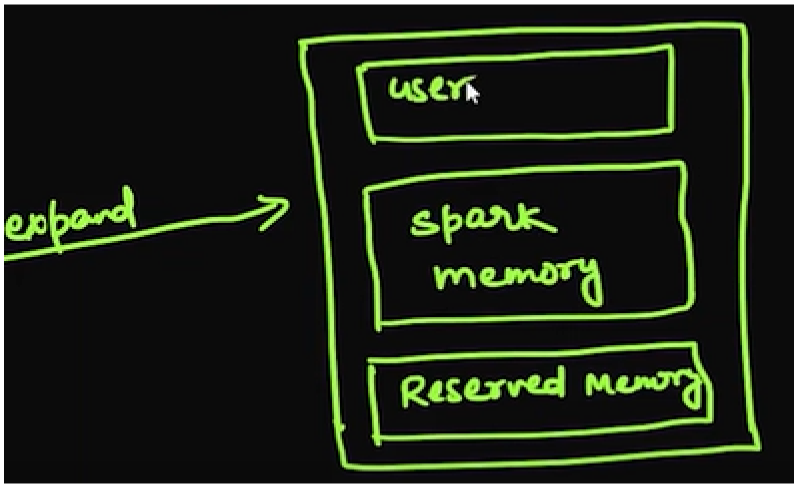
5. So if anything exceeds JVM memory or overhead memory..then we get Exe OOM error



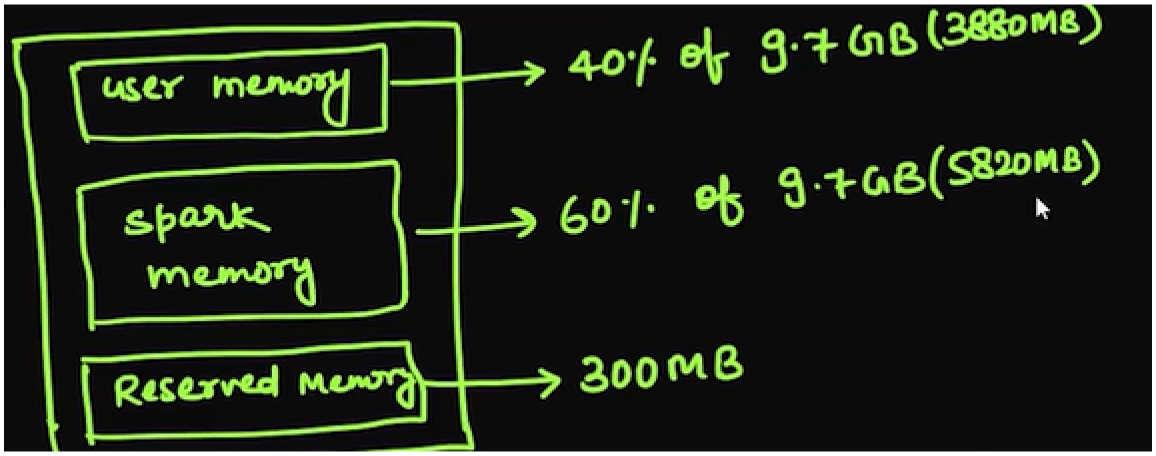

Overhead Memory Usage

(i) It is used for non-JVM processes
(ii) 300-400 MB is used by container.
(iii) 600-700MB will be used by Pyspark applications.

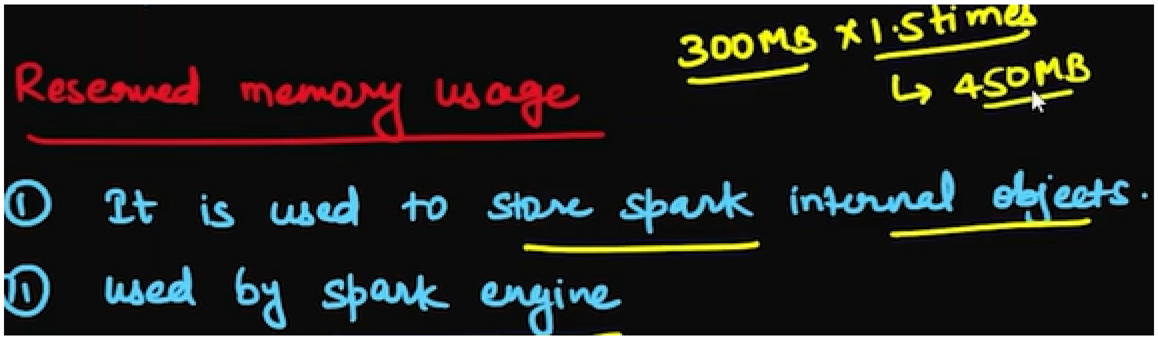
6. Now lets focus into JVM memory of 10gb

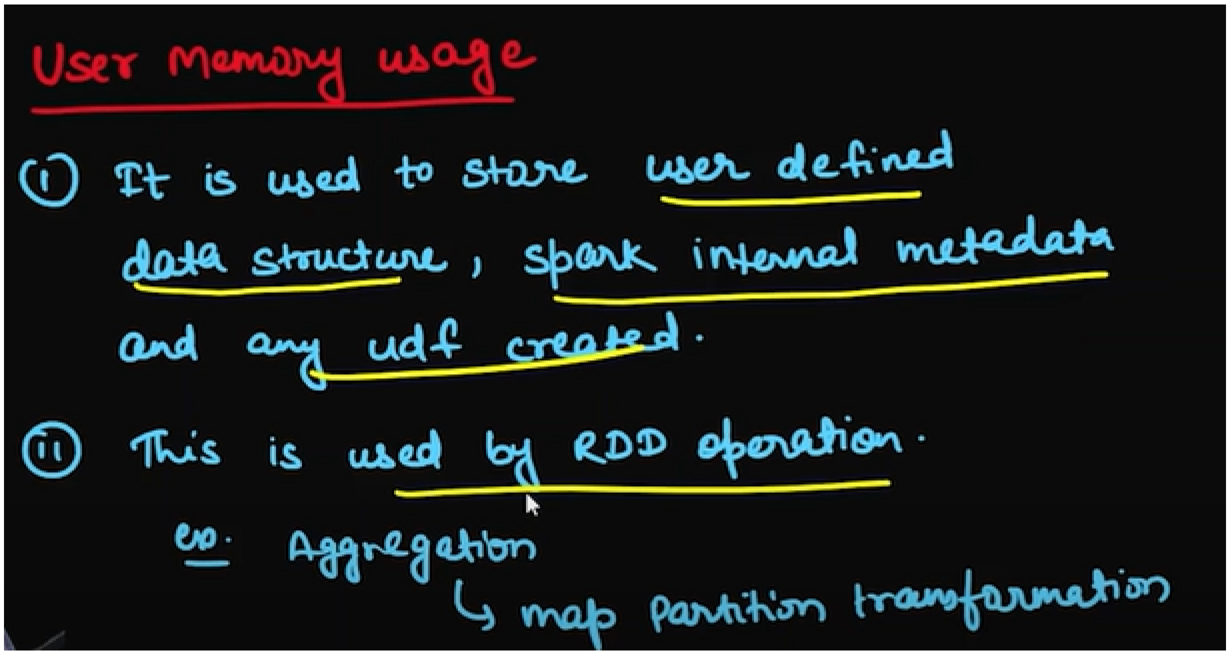


7. So this 10gb jvm memory ..is further divided into 3 memory types ..see above
8. And this 10GB will be splitted among the these 3 memory types ..in this way



9. So if our reserved memory usage is 300MB then executor memory must be atleast 450MB



10. 

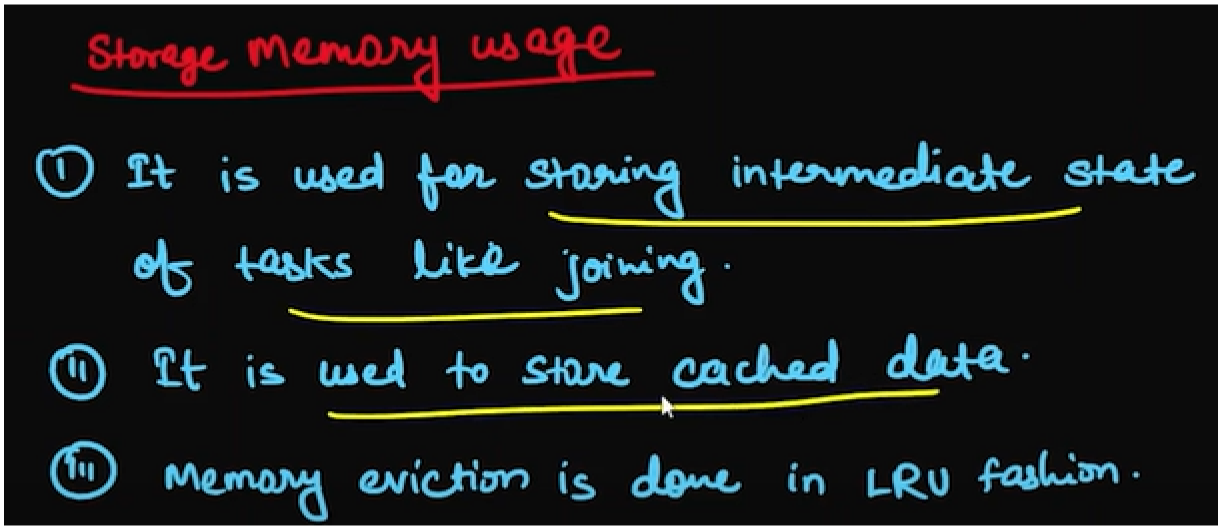
User Memory usage

(i) It is used to store user defined data structure, spark internal metadata and any udf created.

(ii) This is used by RDD operation.

ex. Aggregation

↳ map partition transformation

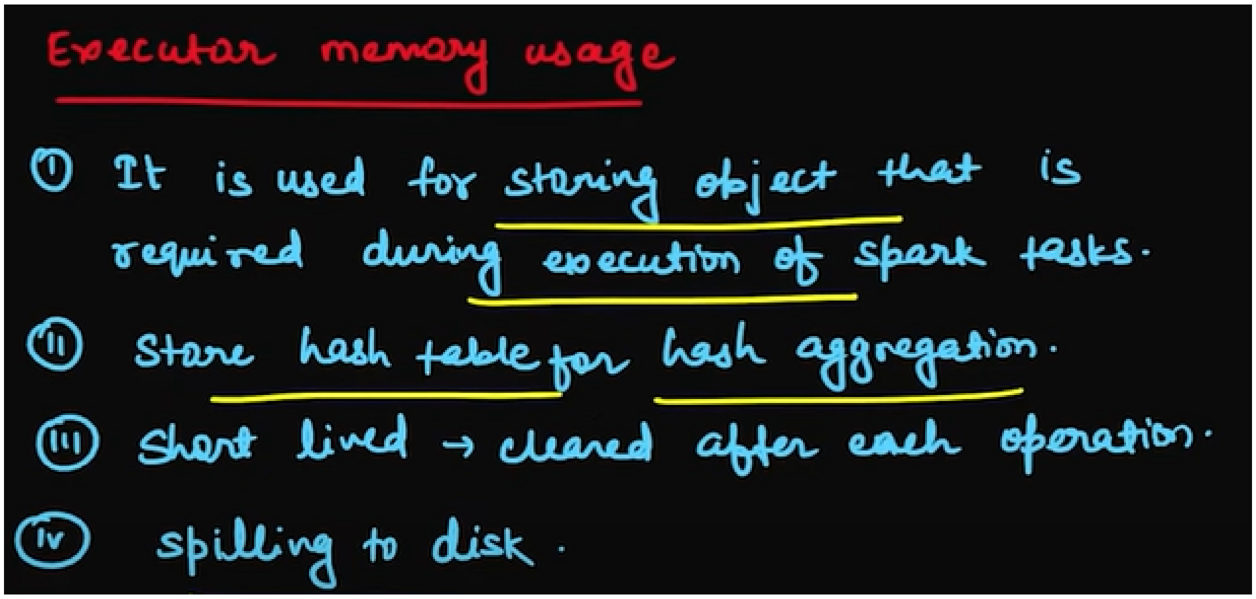
11. If we write any RDD operation in our code..then the user memory is used..
12. For dataframe operations spark memory is used
13. Spark memory is divided into two parts ..see below
14. 

Storage Memory usage

(i) It is used for storing intermediate state of tasks like joining.

(ii) It is used to store cached data.

(iii) Memory eviction is done in LRU fashion.

15. 

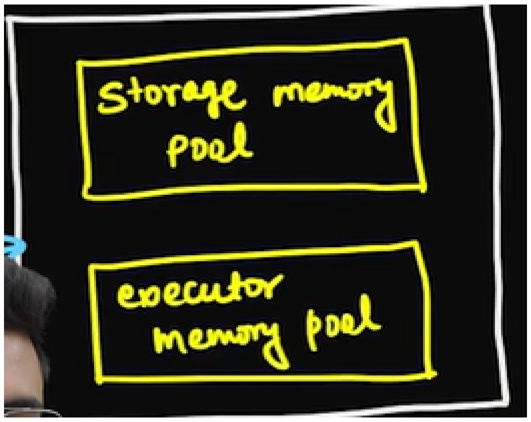
Executor memory usage

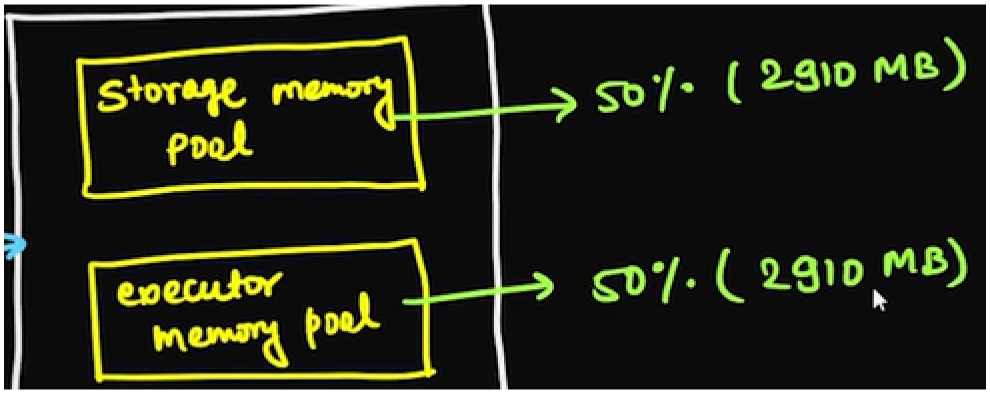
(i) It is used for storing object that is required during execution of spark tasks.

(ii) Store hash table for hash aggregation.

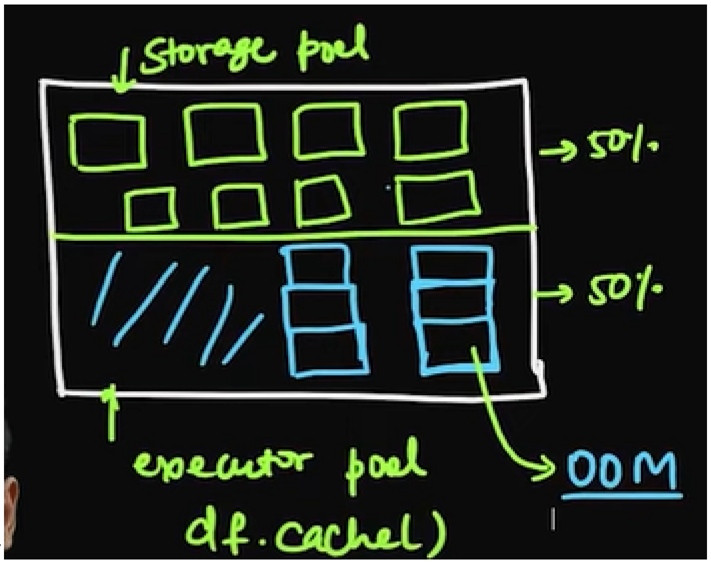
(iii) Short lived → cleared after each operation.

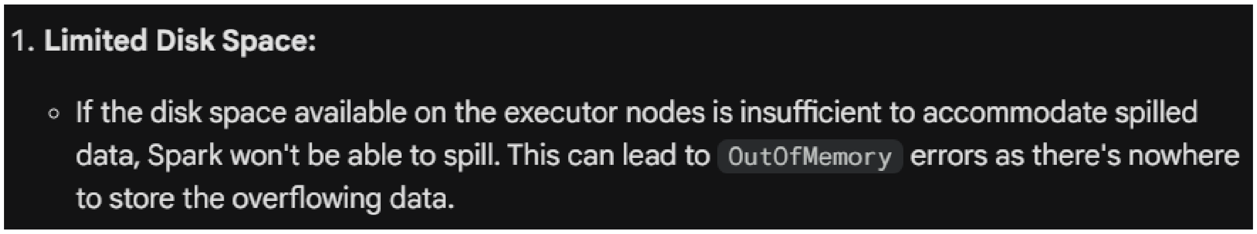
(iv) spilling to disk.

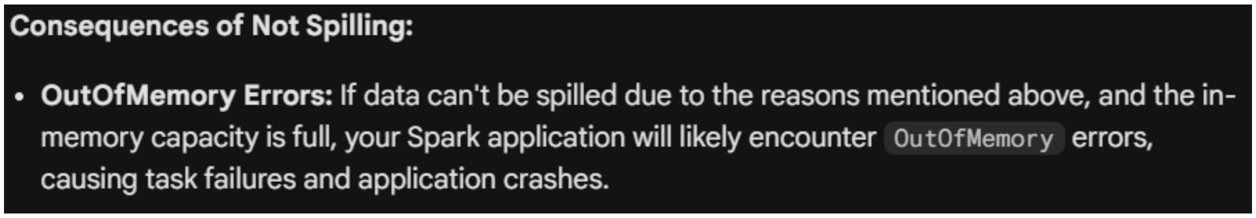
16. Executor memory is used by shuffle hash join
17. Lets expand spark memory 

18. As spark has 5820MB…these evenly distribute this data


19. We have 2 types of memory manager…explained here : https://g.co/gemini/share/d72aa12c3ce6

20. Now if our both storage and executor memory pools gets filled…then we will get OOM error

21. 
1. **Limited Disk Space:**
   - If the disk space available on the executor nodes is insufficient to accommodate spilled data, Spark won't be able to spill. This can lead to `OutOfMemory` errors as there's nowhere to store the overflowing data.


Consequences of Not Spilling:

- **OutOfMemory Errors:** If data can't be spilled due to the reasons mentioned above, and the in-memory capacity is full, your Spark application will likely encounter `OutOfMemory` errors, causing task failures and application crashes.