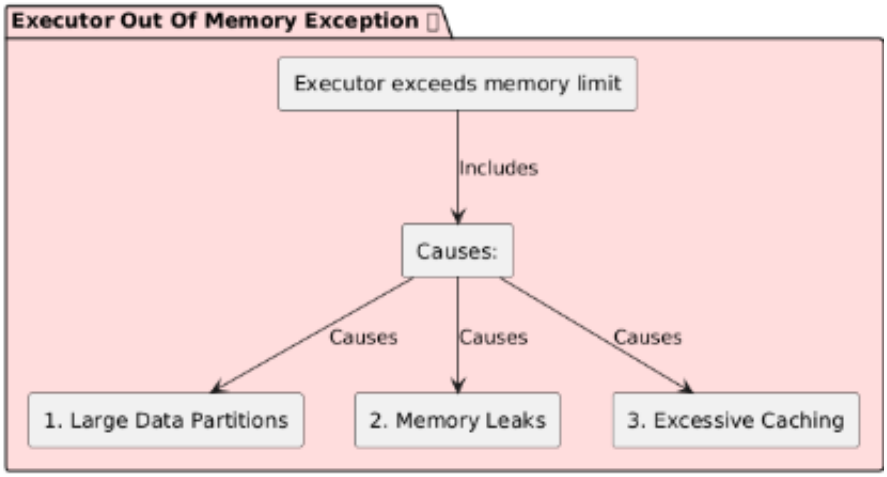
Understanding and Fixing Executor Out Of Memory Exception in Databricks

In Databricks, encountering an **Executor Out Of Memory Exception** indicates that a Spark executor has run out of memory during task execution. This can lead to job failures and reduced performance. Here's a detailed explanation of the issue and how to resolve it within the Databricks environment:

What is Executor Out Of Memory Exception?

The **Executor Out Of Memory Exception** in Databricks occurs when an executor exceeds its allocated memory limit. This issue can arise due to:

1. **Large Data Partitions**: Executors may be handling large data partitions that require more memory than is available.
2. **Memory Leaks**: Inefficient memory use or failure to release unused objects can cause memory exhaustion.
3. **Excessive Caching**: Caching too much data in memory can overwhelm the available memory on the executors.



How to Fix Executor Out Of Memory Exception in Databricks

1. **Increase Executor Memory**:
   - Allocate more memory to executors to better handle large datasets and complex computations. In Databricks, you can adjust executor memory settings via the cluster configuration.
   - Go to the **Clusters** tab, select your cluster, and click on **Edit**. Under the **Spark Config** section, add or adjust the following configuration:

     ```
     spark.executor.memory=4g
     ```

   - This setting increases the memory available to each executor to 4 gigabytes.

2. **Adjust Executor Instances**:
   - Increase the number of executor instances to spread the workload more evenly and reduce memory pressure on individual executors.
   - In the Databricks cluster configuration, adjust the number of worker nodes in the **Worker Type** section to increase the number of executors.

3. **Optimize Data Partitions**:
   - Repartition your data to ensure partitions are of manageable size, preventing executors from being overwhelmed by large partitions.
   - Example in a Databricks notebook:

     ```
     df = df.repartition(100)
     ```

   - This command repartitions the DataFrame into 100 partitions, distributing the data more evenly across executors.

4. **Review Caching Strategy**:
   - Cache only the essential data and ensure that you unpersist data that is no longer needed to avoid excessive memory usage.
   - Example:

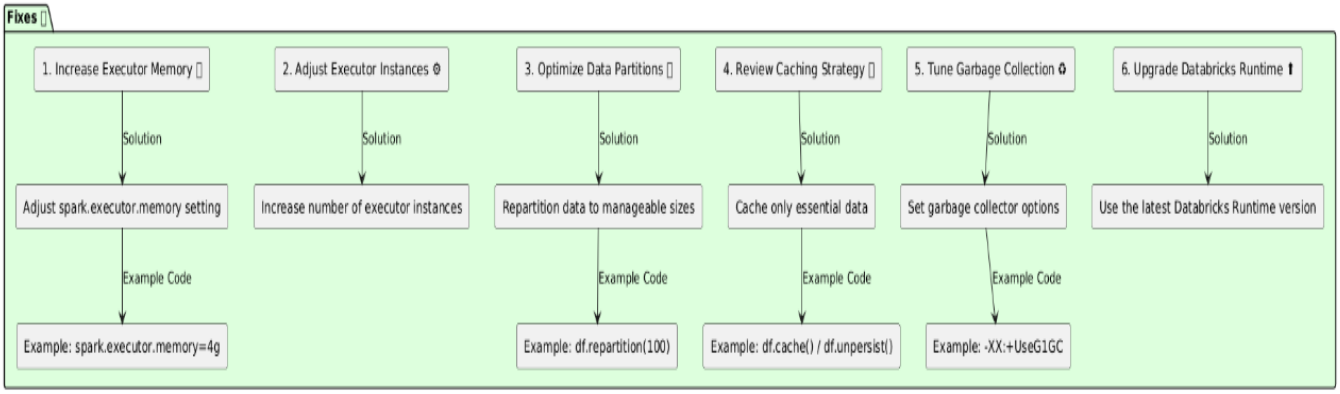
     ```
     df.cache()

     # After processing

     df.unpersist()
     ```

   - The `cache()` method stores the DataFrame in memory, and `unpersist()` helps free up memory when the DataFrame is no longer required.

5. **Tune Garbage Collection**:
   - Optimize garbage collection settings to improve memory management and reduce pauses caused by garbage collection.
   - In Databricks, add the following configuration in the **Spark Config** section of your cluster settings:

     ```
     spark.executor.extraJavaOptions=-XX:+UseG1GC
     ```

   - This configuration enables the G1 garbage collector, which can improve memory management efficiency.
6. **Upgrade Databricks Runtime**:
   - Ensure you are using the latest Databricks Runtime version, as newer versions come with performance improvements and bug fixes related to memory management.



By increasing executor memory, adjusting the number of executor instances, optimizing data partitions, refining caching strategies, tuning garbage collection, and using the latest Databricks Runtime, you can effectively manage memory and mitigate the Executor Out Of Memory Exception in Databricks.