Out Of Memory (OOM) Errors

1️⃣ What is an OOM Error?

- OOM = Out Of Memory → The program requests more memory than available.
- In JVM-based systems (like Spark, Java, Scala):

java.lang.OutOfMemoryError: Java heap space

- In Python (Pandas, NumPy):

MemoryError: Unable to allocate X GiB for array

- Happens when:
  - Dataset too large to fit in memory
  - Inefficient operations that create huge intermediate results
  - Memory configuration too small for the workload

2️⃣ Types of OOM in Data Processing

Type	Example	Context
Driver OOM	java.lang.OutOfMemoryError: Java heap space	Spark driver memory too small to hold data collected/aggregated
Executor OOM	GC overhead limit exceeded	Executor tasks processing too much data in memory
Shuffle OOM	During wide transformations	Joins, groupBy that need big shuffle blocks
Broadcast OOM	Large broadcast variable	Broadcasting a “small” table that’s actually huge

---

3️⃣ Common Causes

1. Loading all data into memory
   - Example in Pandas:

```python
df = pd.read_csv("big_file.csv")  # Reads entire file at once
```

2. Exploding joins
   - Joining two large datasets without filters → huge intermediate table.
3. Wide transformations in Spark
   - groupBy, reduceByKey, or join without partitioning/filtering.
4. Incorrect broadcast join
   - Broadcasting a dataset larger than available executor memory.
5. High parallelism with low memory
   - Many tasks running in parallel → memory contention.

4️⃣ Example in Pandas

python
```
import pandas as pd
import numpy as np

# Create large dataset (~5GB in memory)
rows = 100_000_000
df = pd.DataFrame({    "id": np.arange(rows),     "value": np.random.rand(rows) })

# This will likely OOM in Colab / small memory environments
df['double'] = df['value'] * 2
```
The new column double doubles memory usage temporarily → risk of OOM.

5️⃣ Example in Spark (Driver OOM)

python
```
# Collecting too much data to driver
large_df = spark.range(0, 1_000_000_000)  # 1 billion rows
large_df.collect()  # ❌ Will likely cause driver OOM
```
- collect() pulls all rows into driver memory → not scalable.

6️⃣ How to Prevent OOM

In Pandas

- Chunked reading:

python
```
for chunk in pd.read_csv("big_file.csv", chunksize=100_000):
    process(chunk)
```
- Use Parquet (columnar) instead of CSV for faster, selective reads.
- Drop unnecessary columns early:

python
```
df = df[['needed_col1', 'needed_col2']]
```
In Spark

- Increase memory:

bash
```
--driver-memory 4g
--executor-memory 8g
```
- Filter early (pushdown):

python

```python
df = df.filter(df.date >= '2025-01-01')
```

- Avoid collecting large data to driver (collect(), toPandas()).
- Avoid broadcasting large tables:

sql

```sql
SET spark.sql.autoBroadcastJoinThreshold = -1;
```

7️⃣ Special Case — Broadcast OOM

If you force broadcast a dataset that's too large:

sql

CopyEdit

```sql
SELECT /*+ BROADCAST(big_table) */ ...
```

- Every executor will try to store it → total memory usage = size × num_executors.
- Fix:
  - Let Spark auto-decide broadcast.
  - Reduce dataset size before broadcasting.

8️⃣ Memory Debugging Tips

- In Spark: Use Spark UI → check "Storage" and "Executors" tabs.
- In Python: Use memory_usage() in Pandas:

python

CopyEdit

```python
print(df.memory_usage(deep=True).sum() / 1024**2, "MB")
```

- Monitor OS memory with htop or Colab's "RAM" bar.

9️⃣ Visual Diagram — How OOM Happens

pgsql

```
┌──────────────────────────────────────────────┐
│ Data Read from Disk          │
└──────────────────────────────────────────────┘
        │
   Stored in Memory
        │
┌────────────────▼──────────────┐
│ Memory Limit Reached │  ---> GC tries to free space
└────────────────▲──────────────┘
        │
  No more space left
        │
java.lang.OutOfMemoryError / MemoryError
```

🔟 Key Takeaways

- Load less data → filter early, use columns selectively.
- Right-size memory configs for Spark driver/executors.
- Avoid large intermediate objects in Pandas/Spark.
- In distributed systems, watch out for broadcast joins and shuffle-heavy operations.