MEMORY AND PERSISTENCE
Recommended to use at most only 75% of a machine's memory for Spark.

Minimum Executor heap size should be 8 GB.

Max Executor heap size depends... maybe 40 GB (watch GC)

Memory usage is greatly affected by storage level and serialization format.
RDD.cache() == RDD.persist(MEMORY_ONLY)

most CPU-efficient option
### Storage

<table>
<thead>
<tr>
<th>RDD Name</th>
<th>Storage Level</th>
<th>Cached Partitions</th>
<th>Fraction Cached</th>
<th>Size in Memory</th>
<th>Size on Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Memory Deserialized 1x Replicated</td>
<td>2</td>
<td>100%</td>
<td>55.6 KB</td>
<td>0.0 B</td>
</tr>
</tbody>
</table>
RDD.persist(MEMORY_ONLY_SER)
.persist(MEMORY_AND_DISK)
.persist(MEMORY_AND_DISK_SER)
.persist(DISK_ONLY)
RDD.persist(MEMORY_ONLY_2)
.persist(MEMORY_AND_DISK_2)
.persist(OFF_HEAP)
<table>
<thead>
<tr>
<th>Persistence</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORY_ONLY</td>
<td>Store RDD as deserialized Java objects in the JVM</td>
</tr>
<tr>
<td>MEMORY_AND_DISK</td>
<td>Store RDD as deserialized Java objects in the JVM and spill to disk</td>
</tr>
<tr>
<td>MEMORY_ONLY_SER</td>
<td>Store RDD as serialized Java objects (one byte array per partition)</td>
</tr>
<tr>
<td>MEMORY_AND_DISK_SER</td>
<td>Spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed</td>
</tr>
<tr>
<td>DISK_ONLY</td>
<td>Store the RDD partitions only on disk</td>
</tr>
<tr>
<td>MEMORY_ONLY_2, MEMORY_AND_DISK_2</td>
<td>Same as the levels above, but replicate each partition on two cluster nodes</td>
</tr>
<tr>
<td>OFF_HEAP</td>
<td>Store RDD in serialized format in Tachyon</td>
</tr>
</tbody>
</table>
- If RDD fits in memory, choose **MEMORY_ONLY**

- If not, use **MEMORY_ONLY_SER** w/ fast serialization library

- Don’t spill to disk unless functions that computed the datasets are very expensive or they filter a large amount of data. (recomputing may be as fast as reading from disk)

- Use replicated storage levels sparingly and only if you want fast fault recovery (maybe to serve requests from a web app)
Remember!

Intermediate data is automatically persisted during shuffle operations.
PySpark: stored objects will always be serialized with Pickle library, so it does not matter whether you choose a serialized level.
Default Memory Allocation in Executor JVM

- **Cached RDDs**: 60%
- **Shuffle memory**: 20%
- **User Programs (remainder)**: 20%

- `spark.storage.memoryFraction`
- `spark.shuffle.memoryFraction`
Spark uses memory for:

**RDD Storage:** when you call `.persist()` or `.cache()`. Spark will limit the amount of memory used when caching to a certain fraction of the JVM's overall heap, set by `spark.storage.memoryFraction`.

**Shuffle and aggregation buffers:** When performing shuffle operations, Spark will create intermediate buffers for storing shuffle output data. These buffers are used to store intermediate results of aggregations in addition to buffering data that is going to be directly output as part of the shuffle.

**User code:** Spark executes arbitrary user code, so user functions can themselves require substantial memory. For instance, if a user application allocates large arrays or other objects, these will contend for overall memory usage. User code has access to everything “left” in the JVM heap after the space for RDD storage and shuffle storage are allocated.
DETERMINING MEMORY CONSUMPTION

1. Create an RDD
2. Put it into cache
3. Look at SparkContext logs on the driver program or Spark UI

Logs will tell you how much memory each partition is consuming, which you can aggregate to get the total size of the RDD.

INFO BlockManagerMasterActor: Added rdd_0_1 in memory on mbk.local:50311 (size: 717.5 KB, free: 332.3 MB)