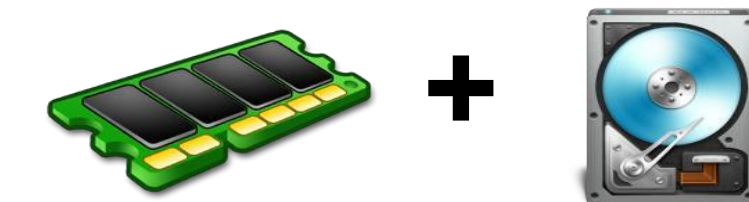
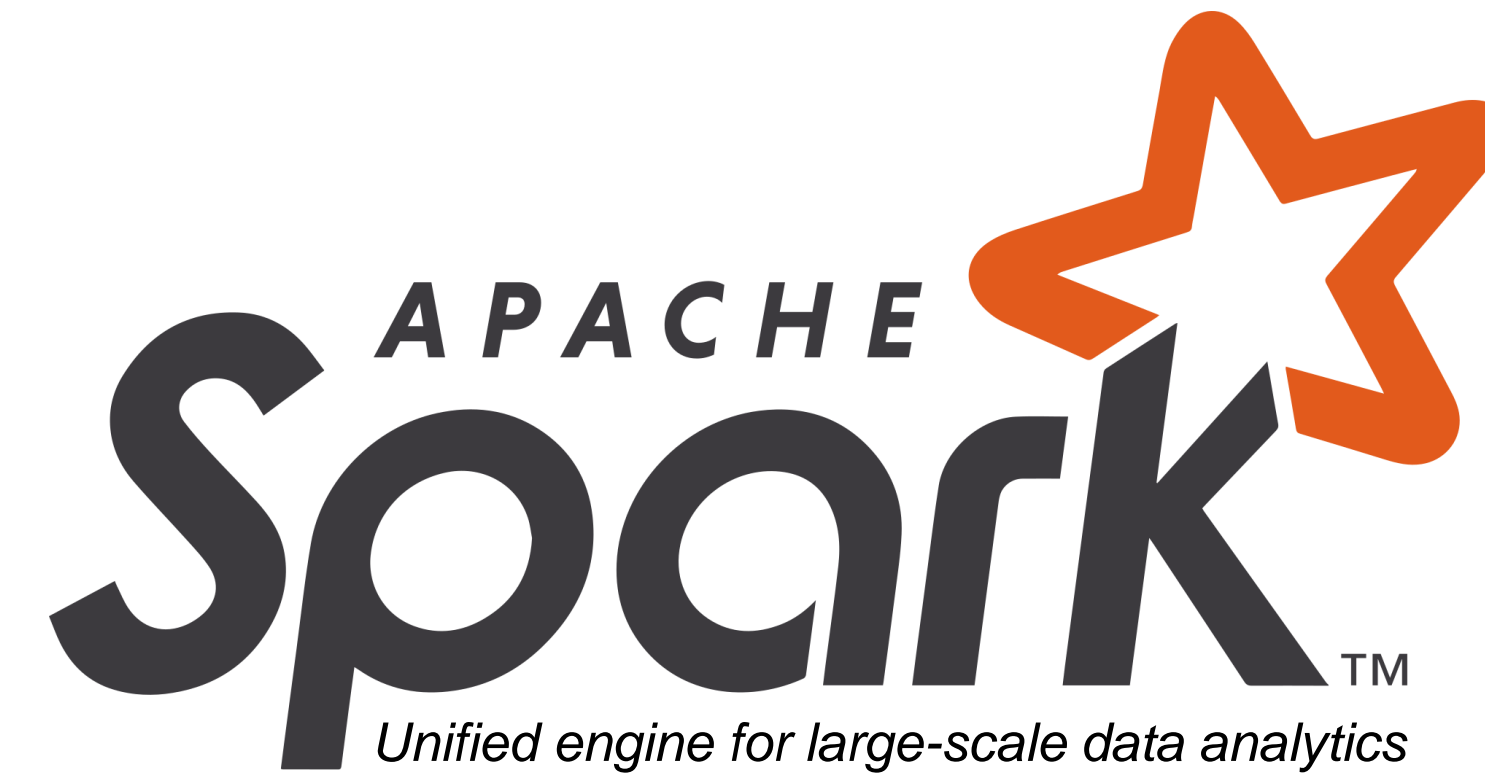


Data Analytics using



RDD Persistence

Sukeshini

Big Data Analytics & Machine Learning Team
C-DAC Bengaluru
sukeshini@cdac.in



RDD Persistence



+



- Spark can ***persist* a dataset in memory/in Disk** across operations
- **When you persist an RDD, each node stores its partitions in memory and reuses it**
- Allows future **actions to be much faster**
- Use **`persist()`** or **`cache()`** methods on RDD.
- **Spark's cache is fault-tolerant** – If any partition of an RDD is lost, it will automatically be recomputed using the transformations that originally created it.
- Each persisted RDD can be stored using a different ***storage level (Set by StorageLevel)***
 - *Persist on Disk*
 - *Persist it in memory*
 - *Replicate it across nodes*



RDD Persistence



+



Available Storage Levels in Python:

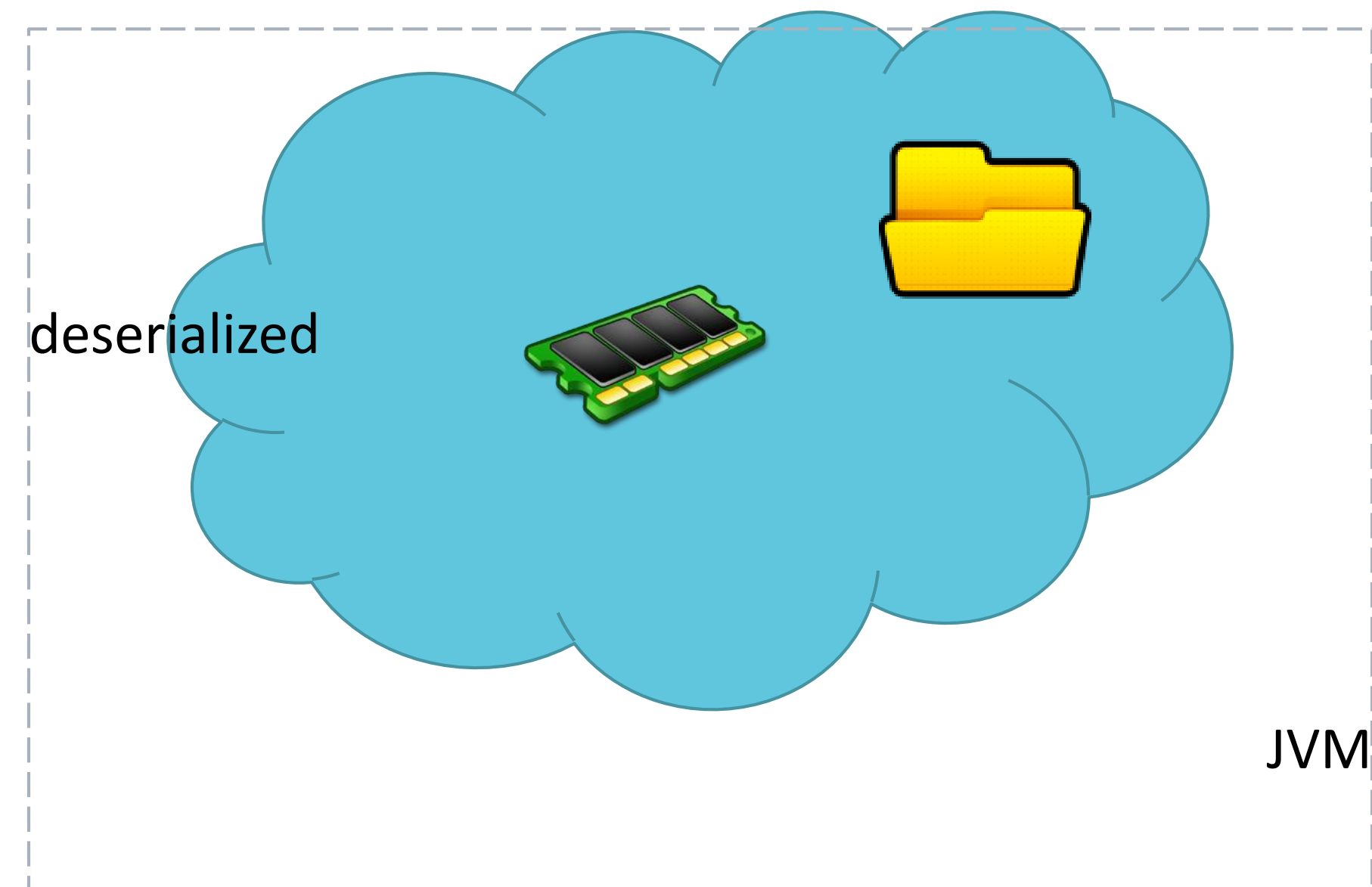
- MEMORY_ONLY
- MEMORY_AND_DISK
- MEMORY_ONLY_SER (Java and Scala)
- MEMORY_AND_DISK_SER (Java and Scala)
- DISK_ONLY
- MEMORY_ONLY_2
- MEMORY_AND_DISK_2
- OFF_HEAP (experimental)

Note: In Python, stored objects will always be serialized with the [Pickle](#) library, no need to choose a serialized level

Storage Level	Meaning
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER (Java and Scala)	Store RDD as <i>serialized</i> Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer , but more CPU-intensive to read.
MEMORY_AND_DISK_SER (Java and Scala)	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Same as the levels above, but replicate each partition on two cluster nodes.
OFF_HEAP (experimental)	Similar to MEMORY_ONLY_SER, but store the data in off-heap memory . This requires off-heap memory to be enabled.

RDD Persistence

MEMORY_ONLY

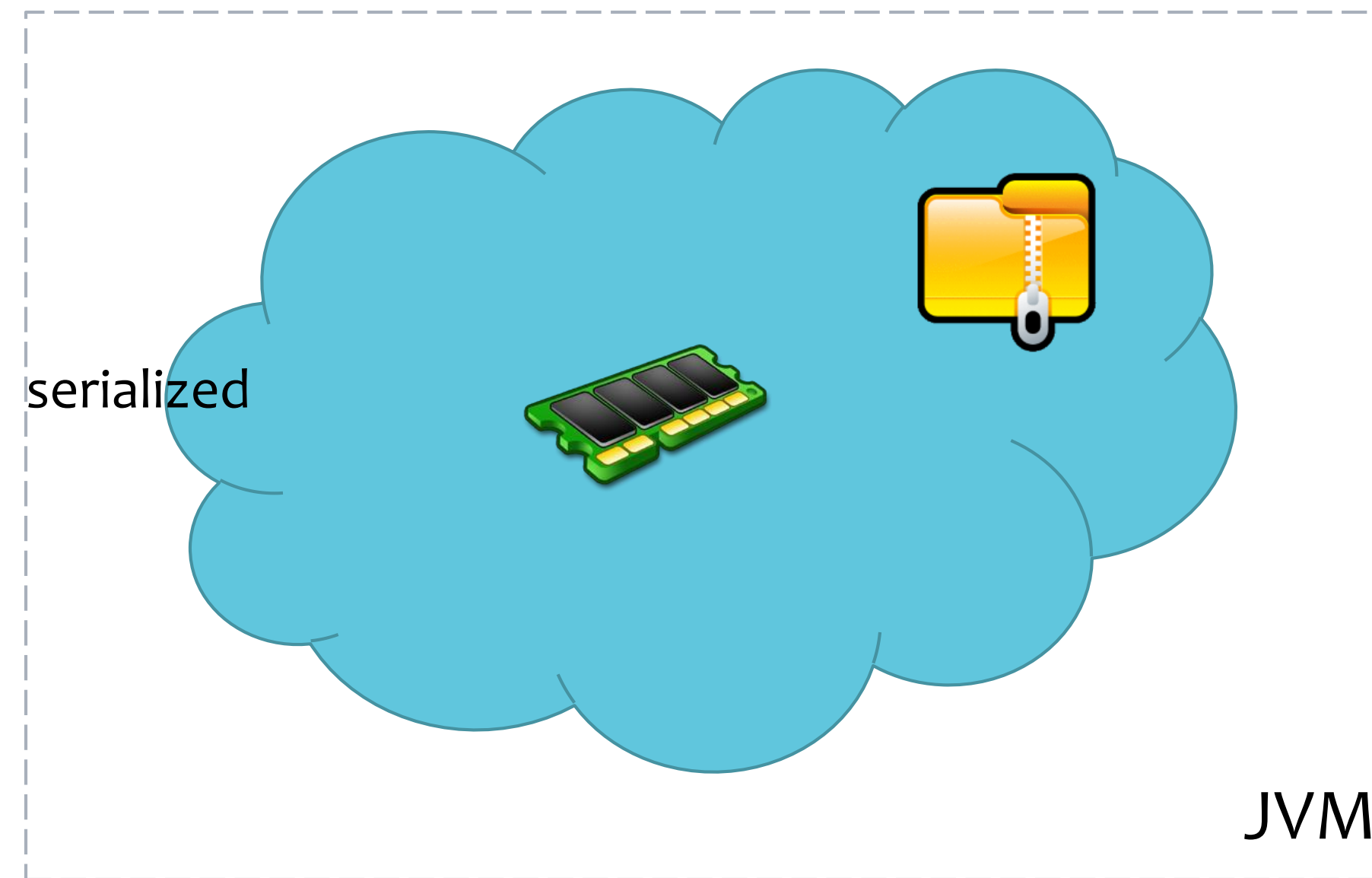


`RDD.cache() == RDD.persist(pyspark.StorageLevel.MEMORY_ONLY)`

most CPU-efficient option

RDD Persistence

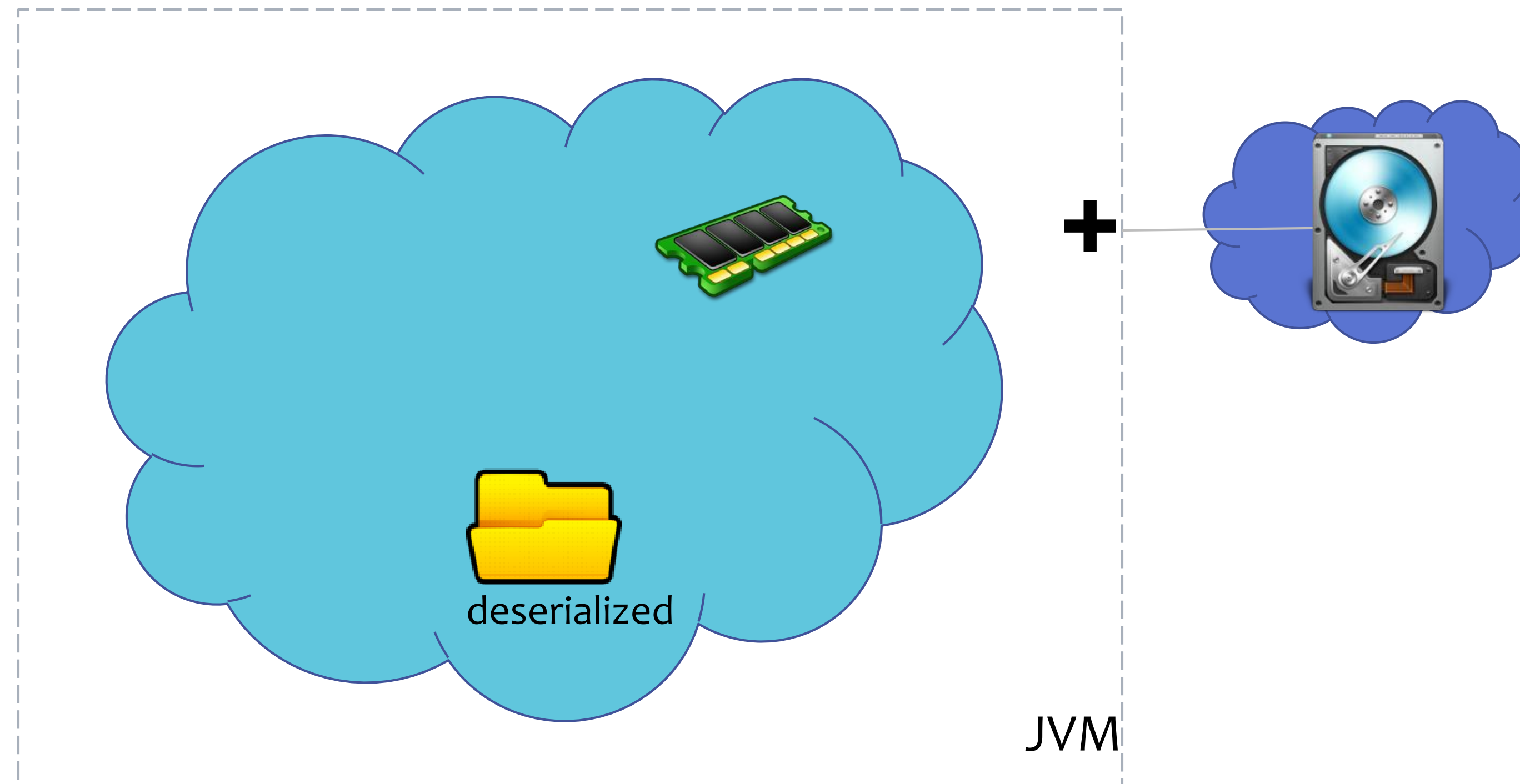
MEMORY_ONLY_SER



```
RDD.persist(pyspark.StorageLevel.MEMORY_ONLY_SER)
```

RDD Persistence

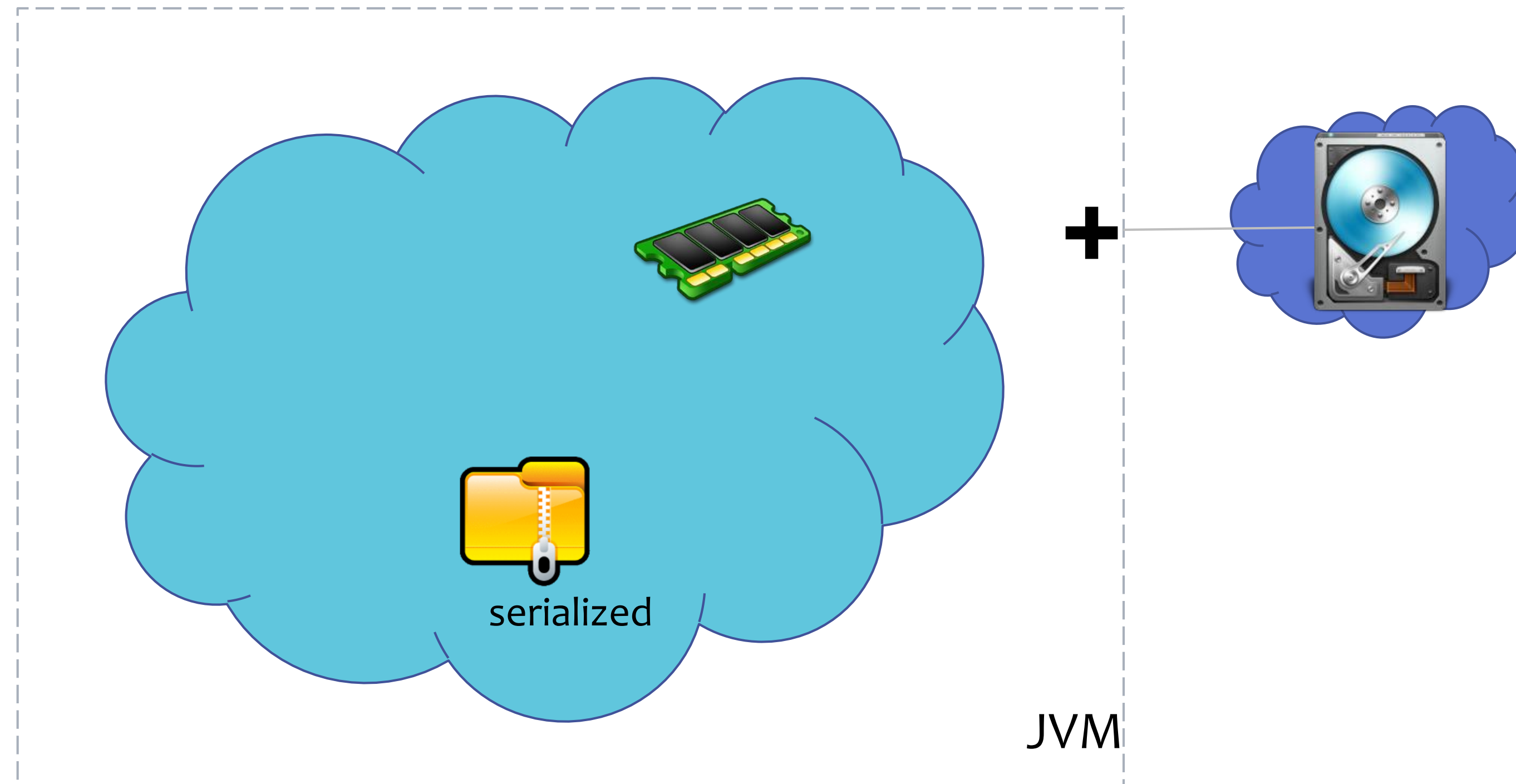
MEMORY_AND_DISK



`RDD.persist(pyspark.StorageLevel.MEMORY_AND_DISK)`

RDD Persistence

MEMORY_AND_DISK_SER



`RDD.persist(pyspark.StorageLevel.MEMORY_AND_DISK_SER)`

RDD Persistence

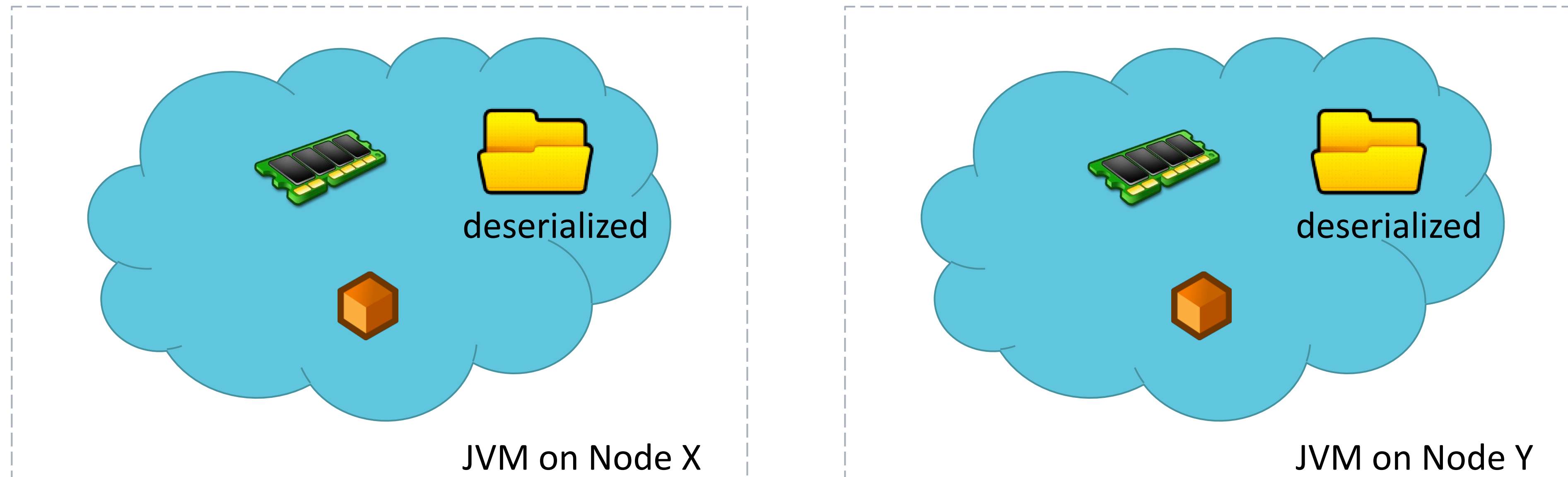
DISK_ONLY



`RDD.persist(pyspark.StorageLevel.DISK_ONLY)`

RDD Persistence

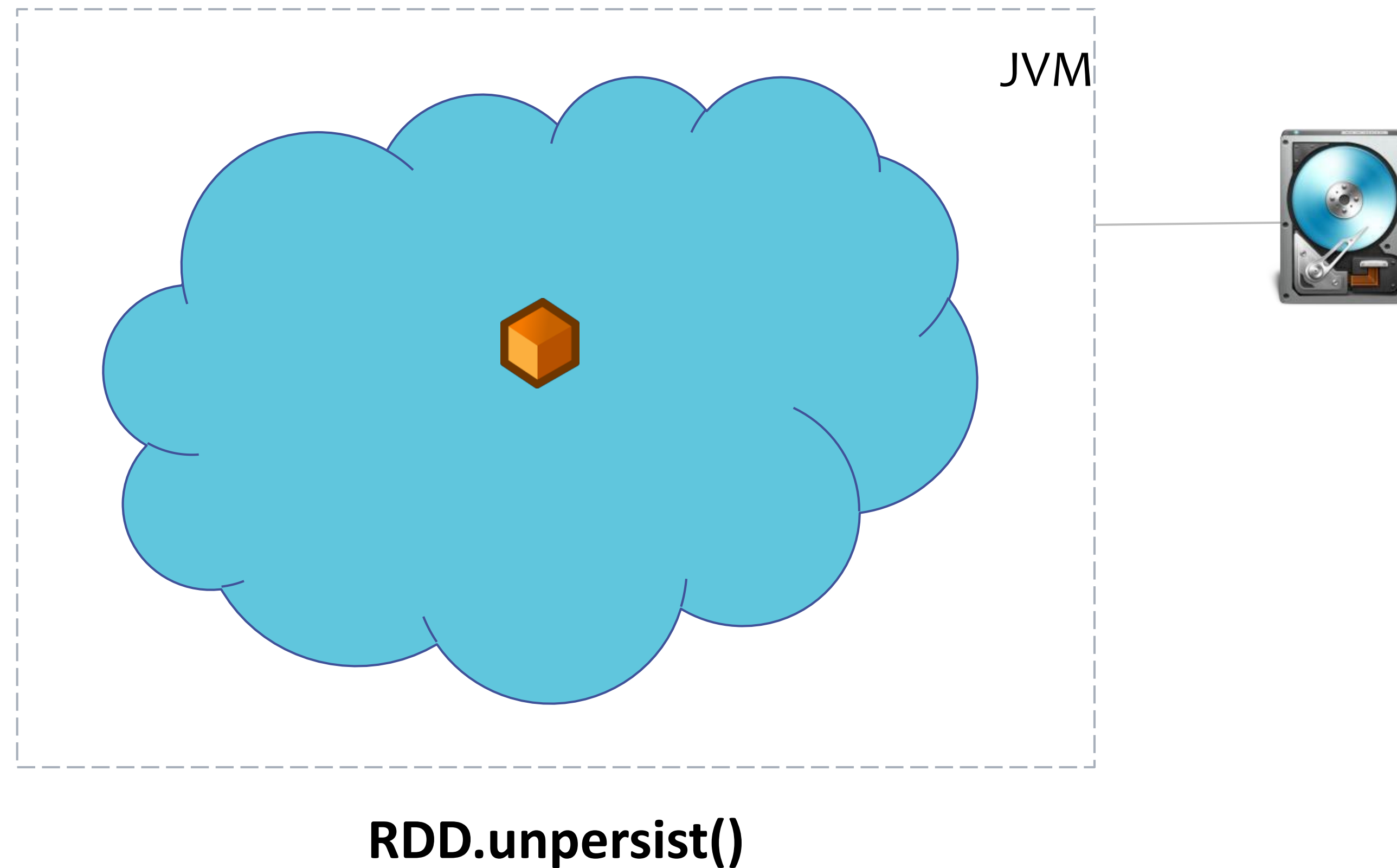
MEMORY_ONLY_2



`RDD.persist(pyspark.StorageLevel.MEMORY_ONLY_2)`

RDD Persistence

.unpersist()



RDD Persistence - Suggestions

- If RDD fits in the memory, choose MEMORY_ONLY
- If not, use MEMORY_ONLY_SER
- **Don't spill to disk** unless functions that computed the datasets are very expensive or they filter a large amount of data. (re-computing 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)

Spark Memory Usage

(a) 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`

Spark Memory Usage

(b) Spark uses memory for: **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.

Spark Memory Usage

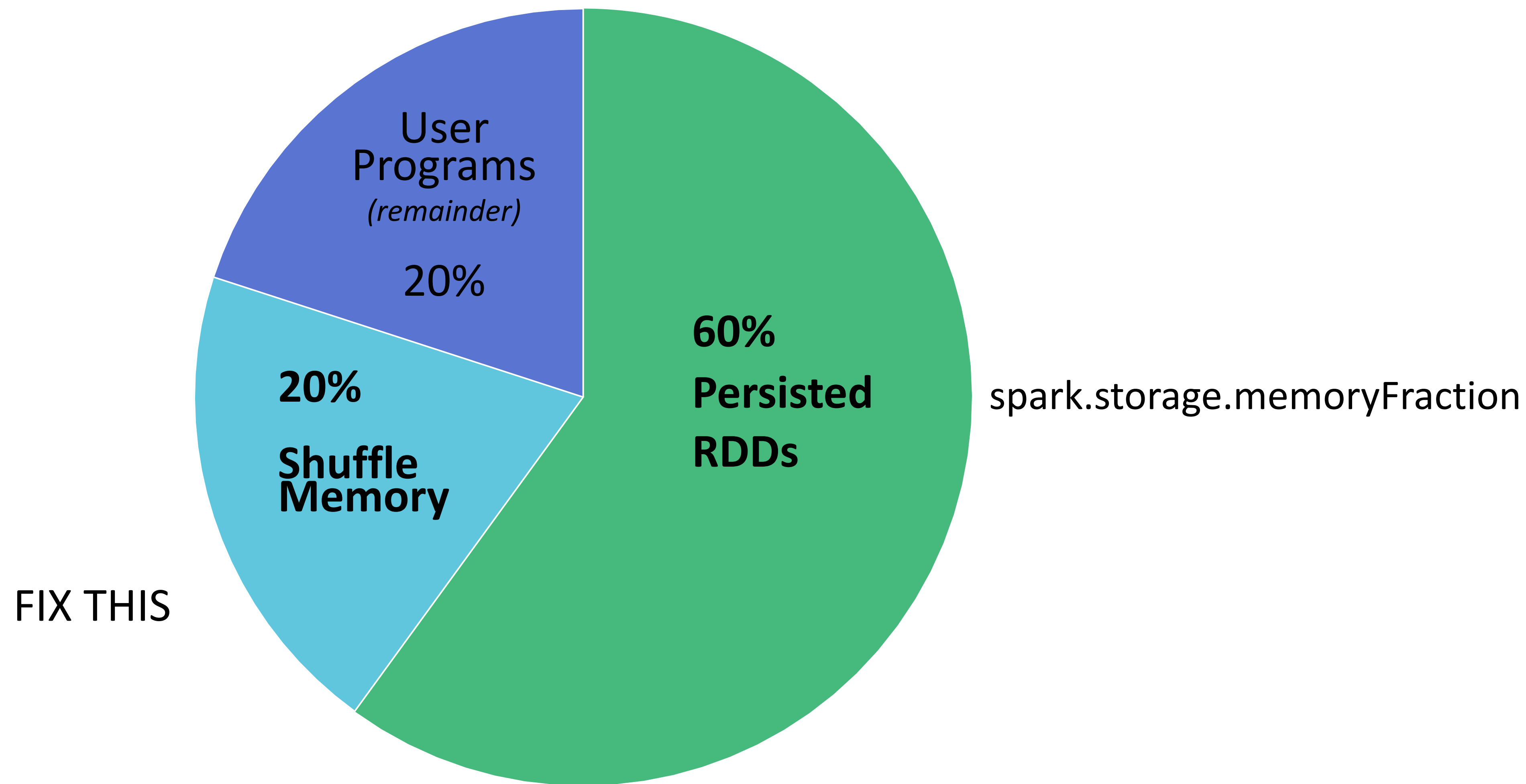
(c) Spark uses memory for: **User code**

Spark executes arbitrary user code, so user functions require substantial memory.

- For instance, if a user application allocates large arrays or other objects, these will content 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.



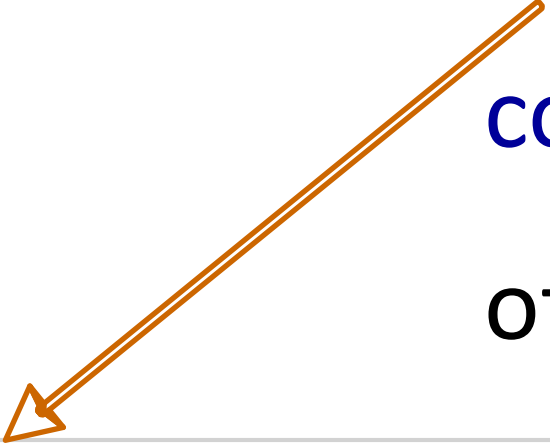
Default Memory Allocation in Executor JVM



Spark Memory Usage

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)
```

Thank You