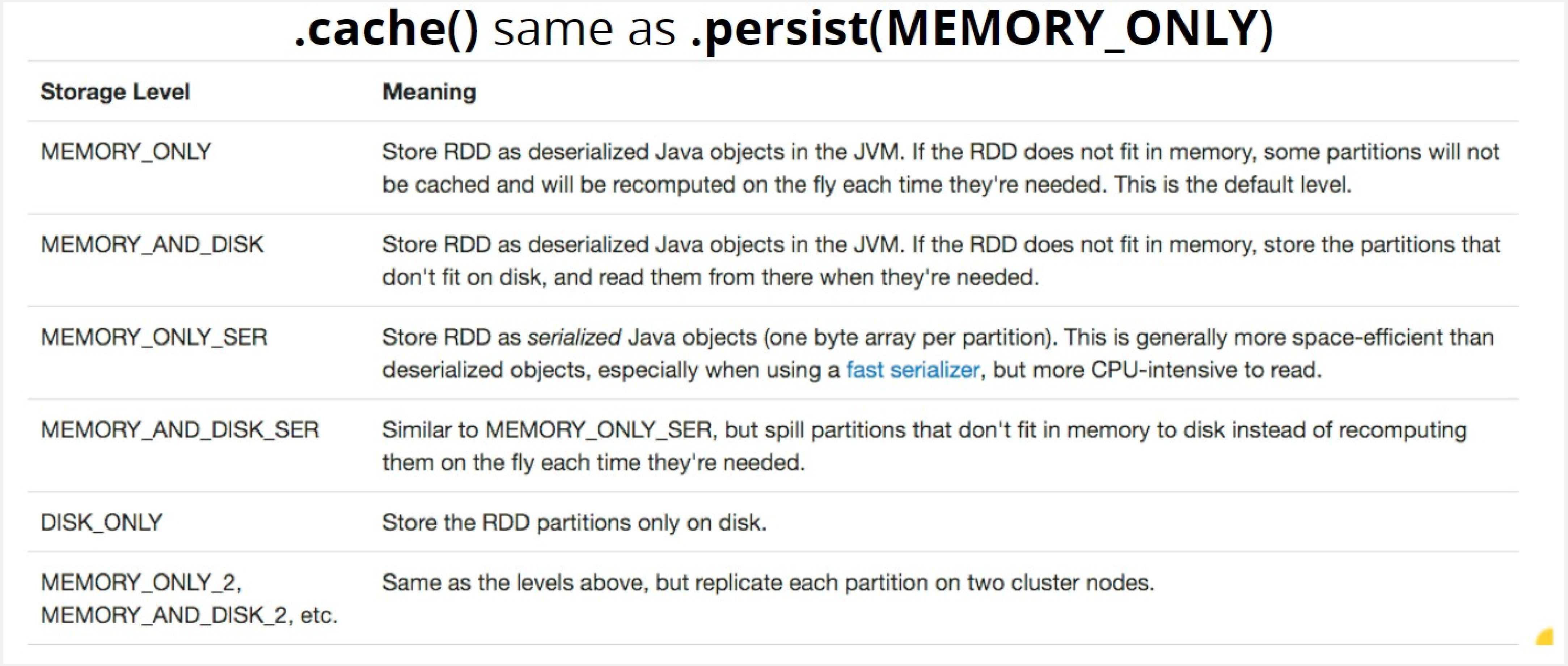
Levels of Persistence

- Caching is useful for retaining intermediate results
- On the other hand, caching can consume a lot of memory
- If memory is exhausted, caches can be eliminated, spilled to disk etc.
- If needed again, cache is recomputed or read from disk.
- The generalization of .cache() is called .persist() which has many options.

Storage Levels



.cache() same as **.persist(MEMORY_ONLY)**

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	Store RDD as *serialized* 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	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.

http://spark.apache.org/docs/latest/programming-guide.html#rdd-persistence

Checkpointing

- Spark is fault tolerant. If a slave machine crashes, it's RDD's will be recomputed.
- If hours of computation have been completed before the crash, all the computation needs to be redone.
- Checkpointing reduces this problem by storing the materialized RDD on a remote disk.
- On Recovery, the RDD will be recovered from the disk.
- It is recommended to cache an RDD before checkpointing it.

http://takwatanabe.me/pyspark/generated/generated/pyspark.RDD.checkpoint.html

Checkpoint vs. Persist to Disk

- Cache materializes the RDD and keeps it in memory (and/or disk). But the lineage （computing chain） of RDD (that is, seq of operations that generated the RDD) **will be remembered**, so that if there are node failures and parts of the cached RDDs are lost, they can be regenerated.
- Checkpoint saves the RDD to an HDFS file and actually forgets the lineage completely. This allows long lineages to be truncated and the data to be saved reliably in HDFS, which is naturally fault-tolerant by replication.
- https://github.com/JerryLead/SparkInternals/blob/master/markdown/english/6-CacheAndCheckpoint.md