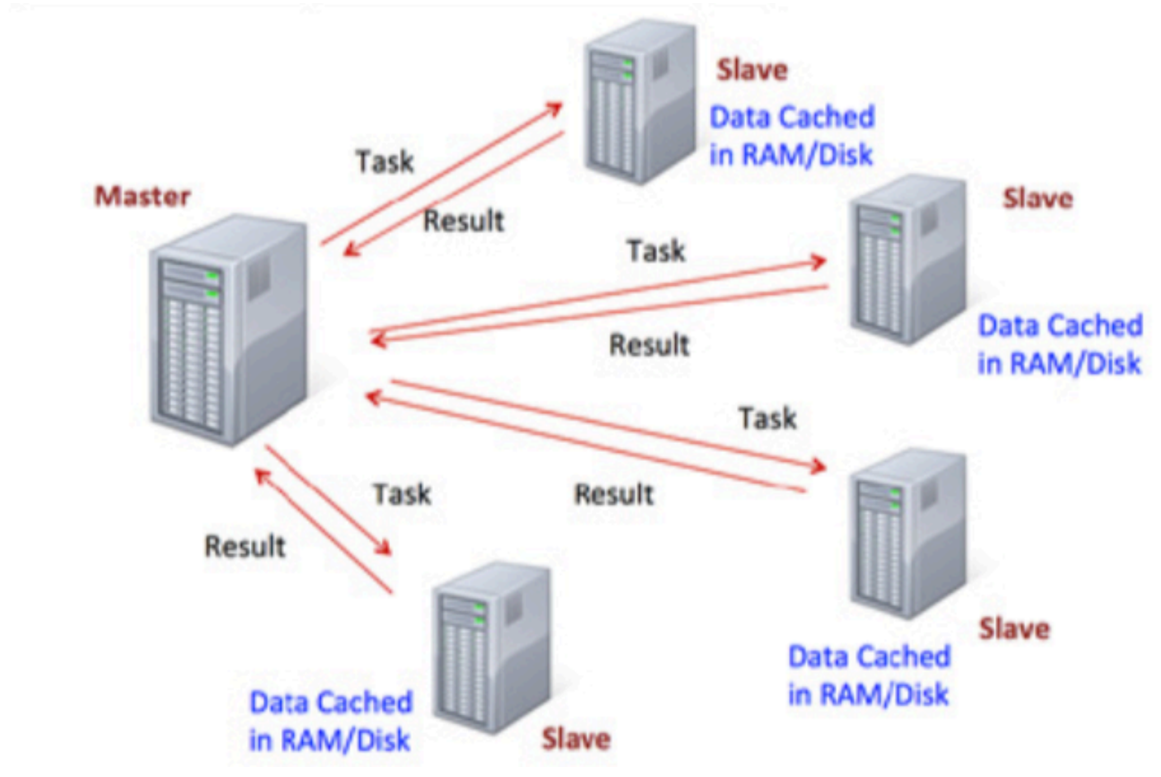
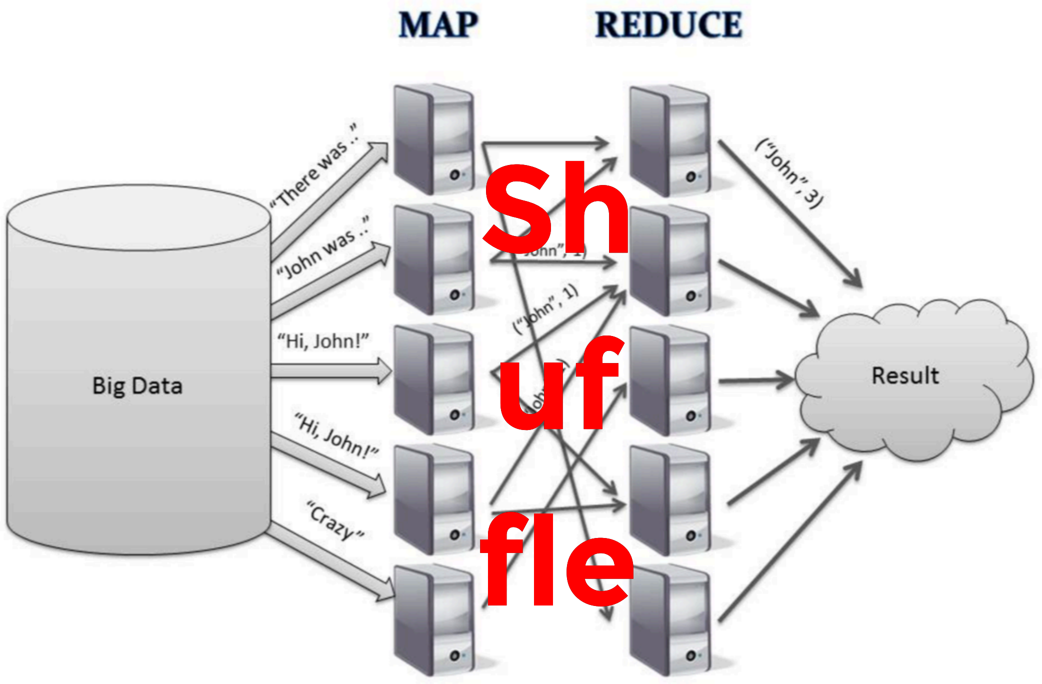
Spark architecture

Hardware organization



In local installation, cores serve as master & slaves

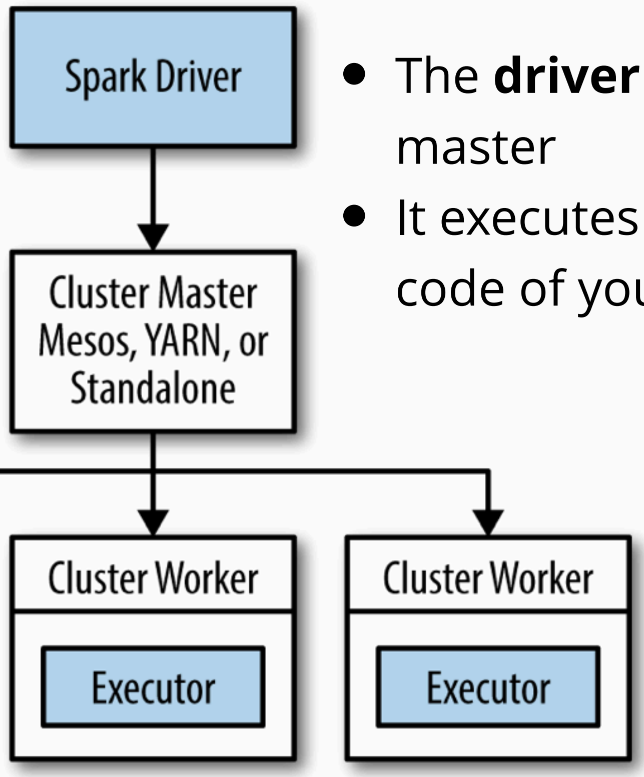
Communication



- Same machines are used for both map and reduce (decreases communication but only slightly)
- Communication between slaves is the toughest bottleneck.
- Design your computation to minimize communication.

spatial software organization

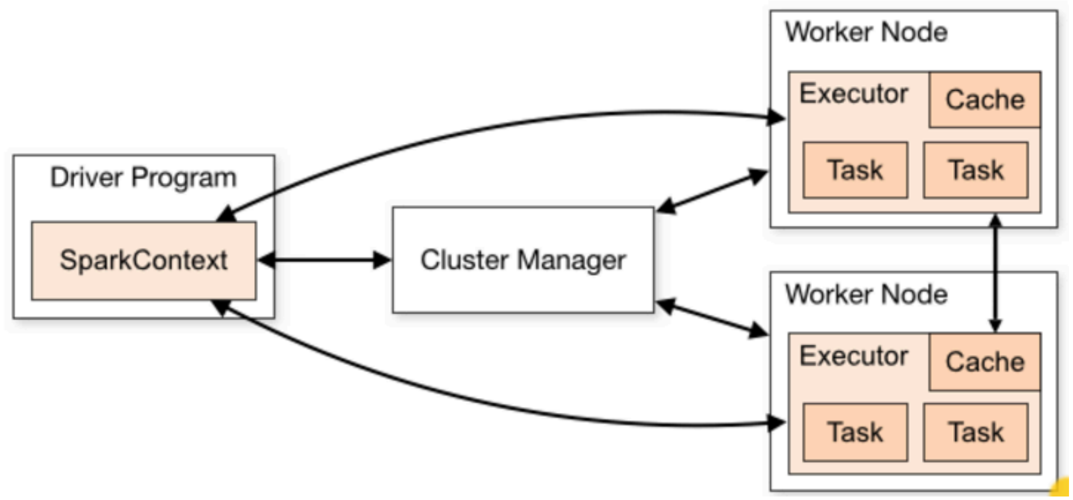
- The Cluster Master manages the computation resources.
- Mesos and Yarn are resource management programs for clusters.



- The **driver** runs on the master
- It executes the "main()" code of your program.

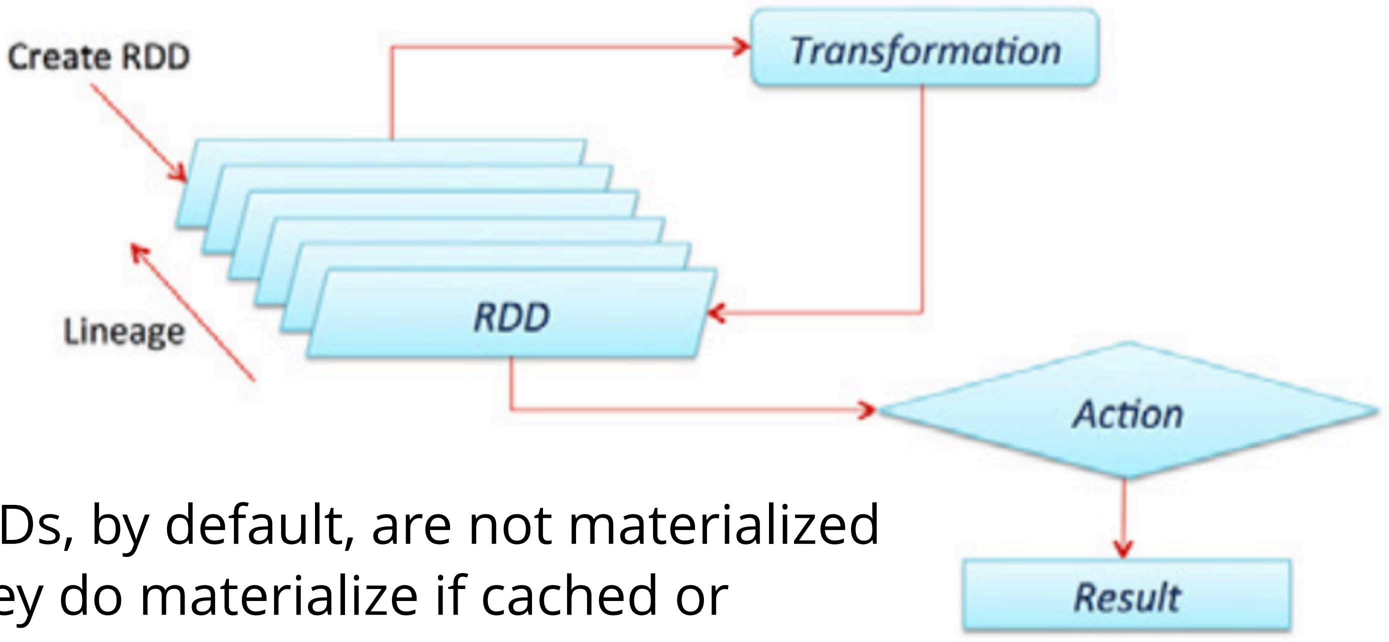
- **Workers** run on the slaves (usually one per core)
- Each RDD is **partitioned** among the workers,
- Workers manage **partitions** and **Executors**
- Executors execute **tasks** on their partition, are myopic.

spatial organization (more detail)



- SparkContext (sc) is the abstraction that encapsulates the cluster for the driver node (and the programmer).
- Worker nodes manage resources in a single slave machine.
- Worker nodes communicate with the cluster manager.
- Executors are the processes that can perform **tasks**.
- Cache refers to the local memory on the slave machine.

RDD Processing

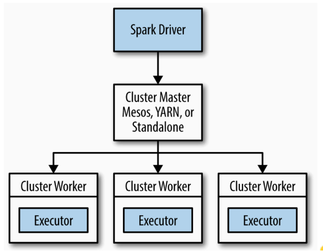


- RDDs, by default, are not materialized
- They do materialize if cached or otherwise persisted.

Temporal organization

RDD Graph and Physical plan

Recall Spatial organization



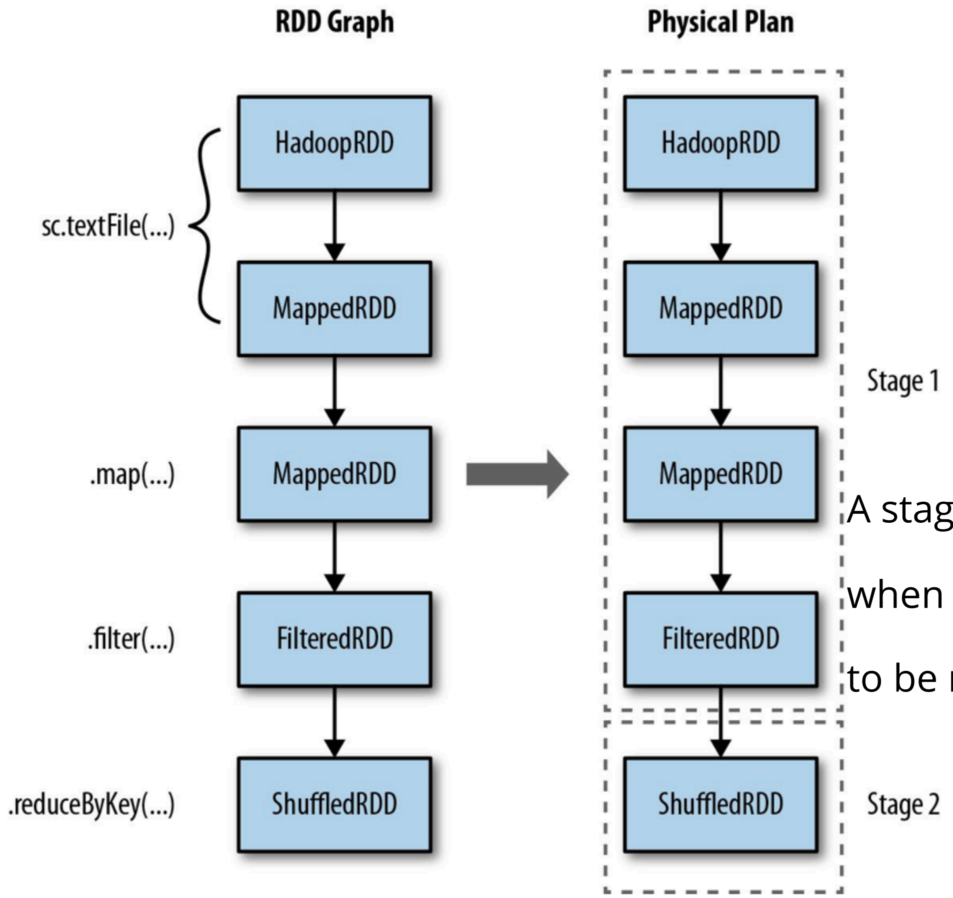


Figure 8-1. RDD transformations pipelined into physical stages

A stage ends when the RDD needs to be materialized

Terms and concepts of execution

- RDDs are **partitioned** across workers, each worker manages a one **partition** of each RDD.
- RDD graph defines the **Lineage** of the RDDs.
- SparkContext divides the RDD graph into **stages** which defines the execution plan (or physical plan)
- A **task** corresponds to the to one stage, restricted to one partition.
- An **executor** is a process that can perform tasks.

Persistance

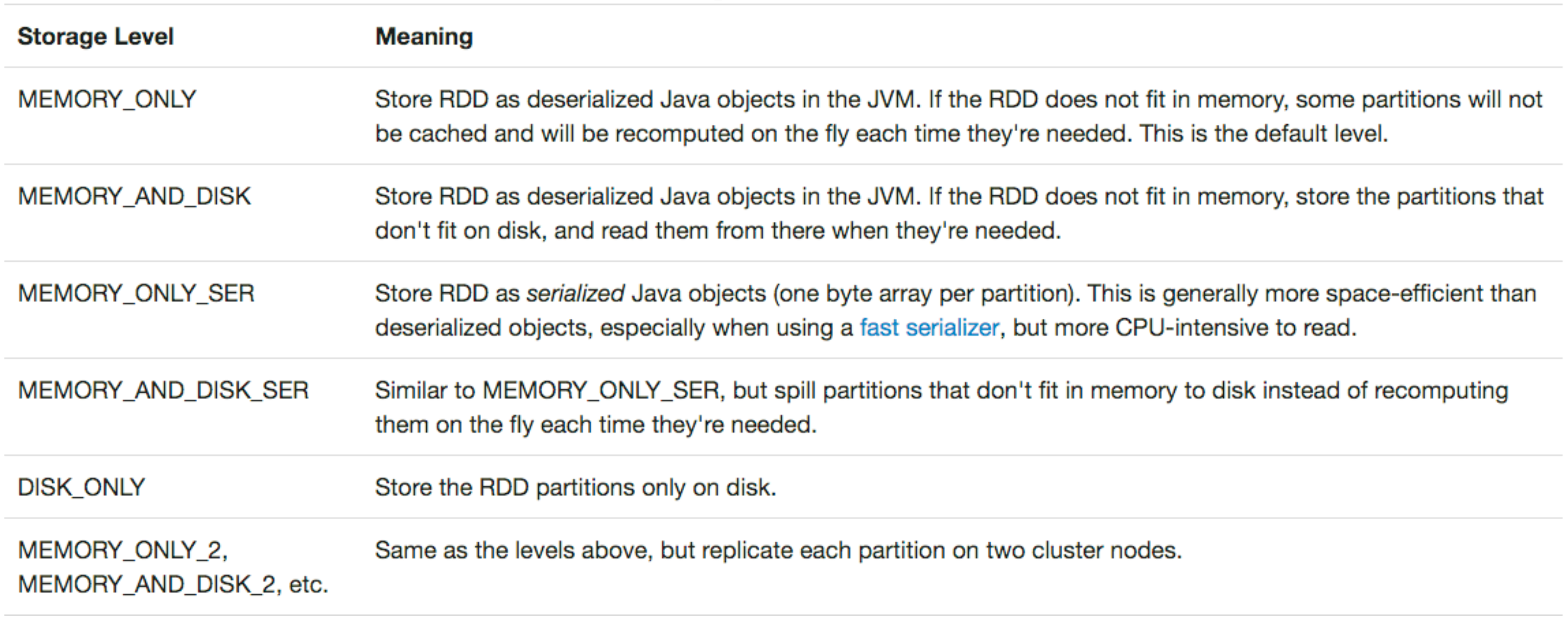
and Checkpointing

Levels of persistance

- 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.

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.