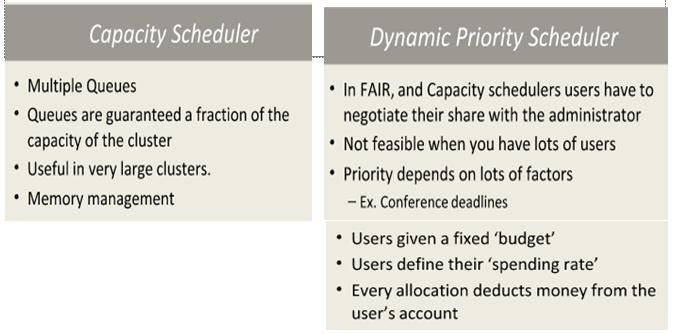
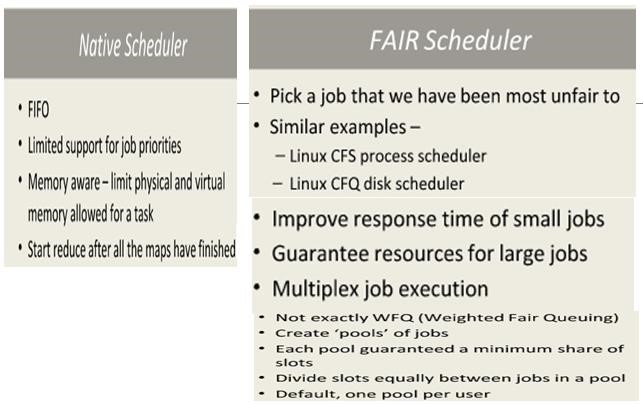


# Types of job schedulers



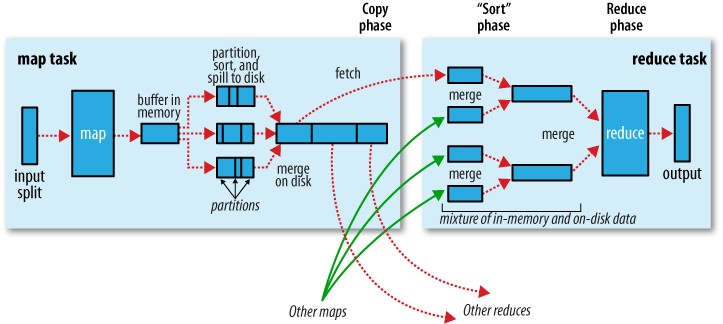
# Shuffle and Sort

* MapReduce makes the guarantee that the input to every reducer is sorted by key.
* The process by which the system performs the sort—and transfers the map outputs to the reducers as inputs—is known as the shuffle.
* The shuffle is an area of the codebase where refinements and improvements are continually being made.

# STEPS

1. **The Map Side**

When the map function starts producing output, it is not simply written to disk. The process is more involved, and takes advantage of buffering writes in memory and doing some presorting for efficiency reasons.



**Shuffle and sort in MapReduce**

The buffer is 100 MB by default, a size which can be tuned by changing the io.sort.mb property. When the contents of the buffer reaches a certain threshold size a background thread will start to spill the contents to disk. Map outputs will continue to be written to the buffer while the spill takes place, but if the buffer fills up during this time, the map will block until the spill is complete. Spills are written in round-robin fashion to the directories specified by the mapred.local.dir property, in a job-specific subdirectory. Before it writes to disk, the thread first divides the data into partitions corresponding to the reducers that they will ultimately be sent to. Within each partition, the background thread performs an in-memory sort by key, and if there is a combiner function, it is run on the output of the sort.

1. **The reduce side**

The map output file is sitting on the local disk of the machine that ran the map task. The reduce task needs the map output for its particular partition from several map tasks across the cluster.

During the reduce phase, the reduce function is invoked for each key in the sorted output. The output of this phase is written directly to the output filesystem, typically HDFS. In the case of HDFS, since the tasktracker node is also running a datanode, the first block replica will be written to the local disk.

1. **Configuration Tuning**

The general principle is to give the shuffle as much memory as possible. There is a trade-off, in that you need to make sure that your map and reduce functions get enough memory to operate. Write your map and reduce functions to use as little memory as possible. They should not use an unbounded amount of memory. The amount of memory given to the JVMs in which the map and reduce tasks run is set by the mapred.child.java.opts property.

On the map side, the best performance can be obtained by avoiding multiple spills to disk; one is optimal. On the reduce side, the best performance is obtained when the intermediate data can reside entirely in memory.