

MapReduce

Scaling a simple program manually

WordCount Example

- “Do as I say, not as I do”

Word	Count
as	2
do	2
i	2
not	1
say	1
as	2

```
define wordCount as Multiset;
for each document in documentSet {
    T = tokenize(document);
    for each token in T {
        wordCount[token]++;
    }
}
display(wordCount);
```

Distributed Version

- When sets of documents become large
 - spam filter to know the words frequently used in the millions of spam emails you've received.

```
define wordCount as Multiset;
for each document in documentSubset {
    T = tokenize(document);
    for each token in T {
        wordCount[token]++;
    }
}
sendToSecondPhase(wordCount);
```

```
define totalWordCount as Multiset;
for each wordCount received from firstPhase {
    multisetAdd (totalWordCount, wordCount);
}
```

Issues

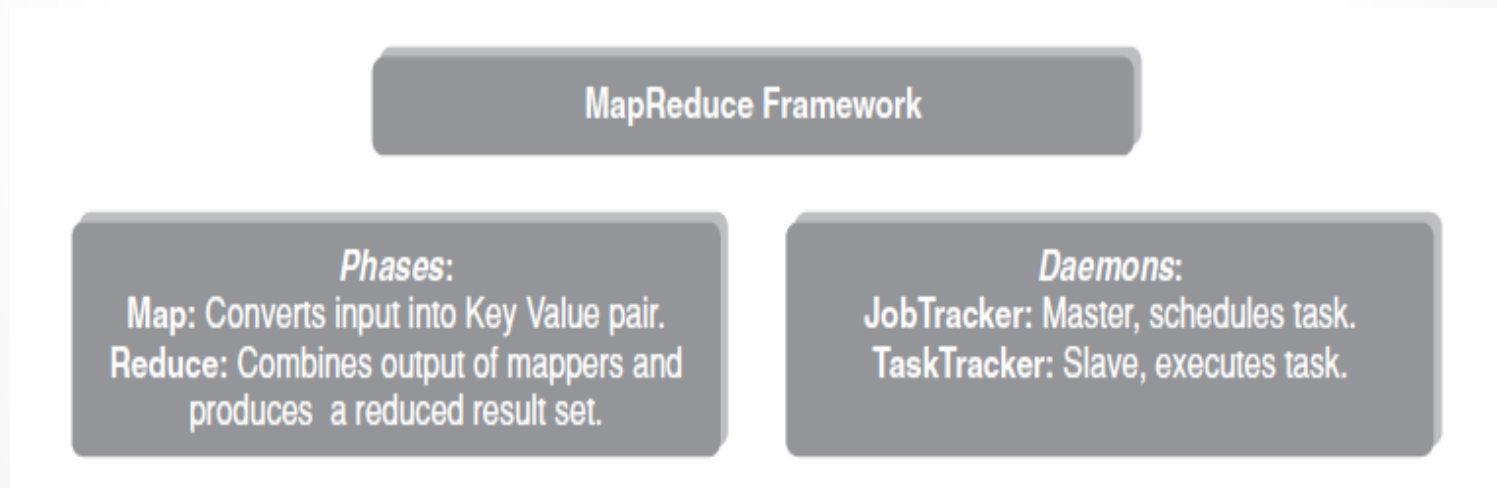
- Performance requirement of reading in the documents
- wordCount (and totalWordCount) are stored in memory. What if wordCount may not fit in memory.
- Phase two will become the bottleneck because it has only one machine
 - partition wordCount after phase one such that each machine in phase two only has to handle one partition.

Why MapReduce Framework

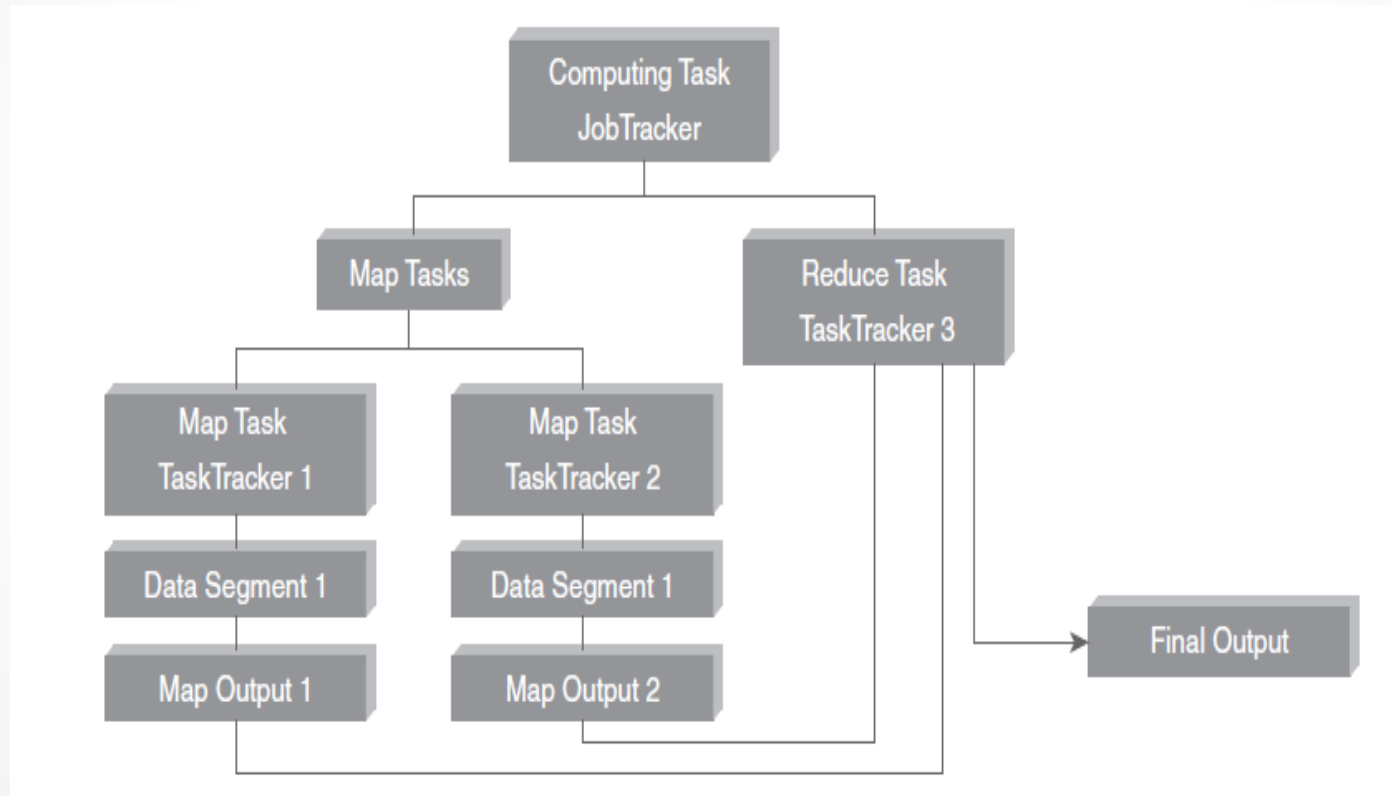
- To make wordcount work across a cluster of distributed machines
 - Store files over many processing machines (of phase one).
 - Write a disk-based hash table permitting processing without being limited by RAM capacity.
 - Partition the intermediate data (that is, wordCount) from phase one.
 - Shuffle the partitions to the appropriate machines in phase two.
 - Fault tolerance. (What if a machine fails in the middle of its task?)

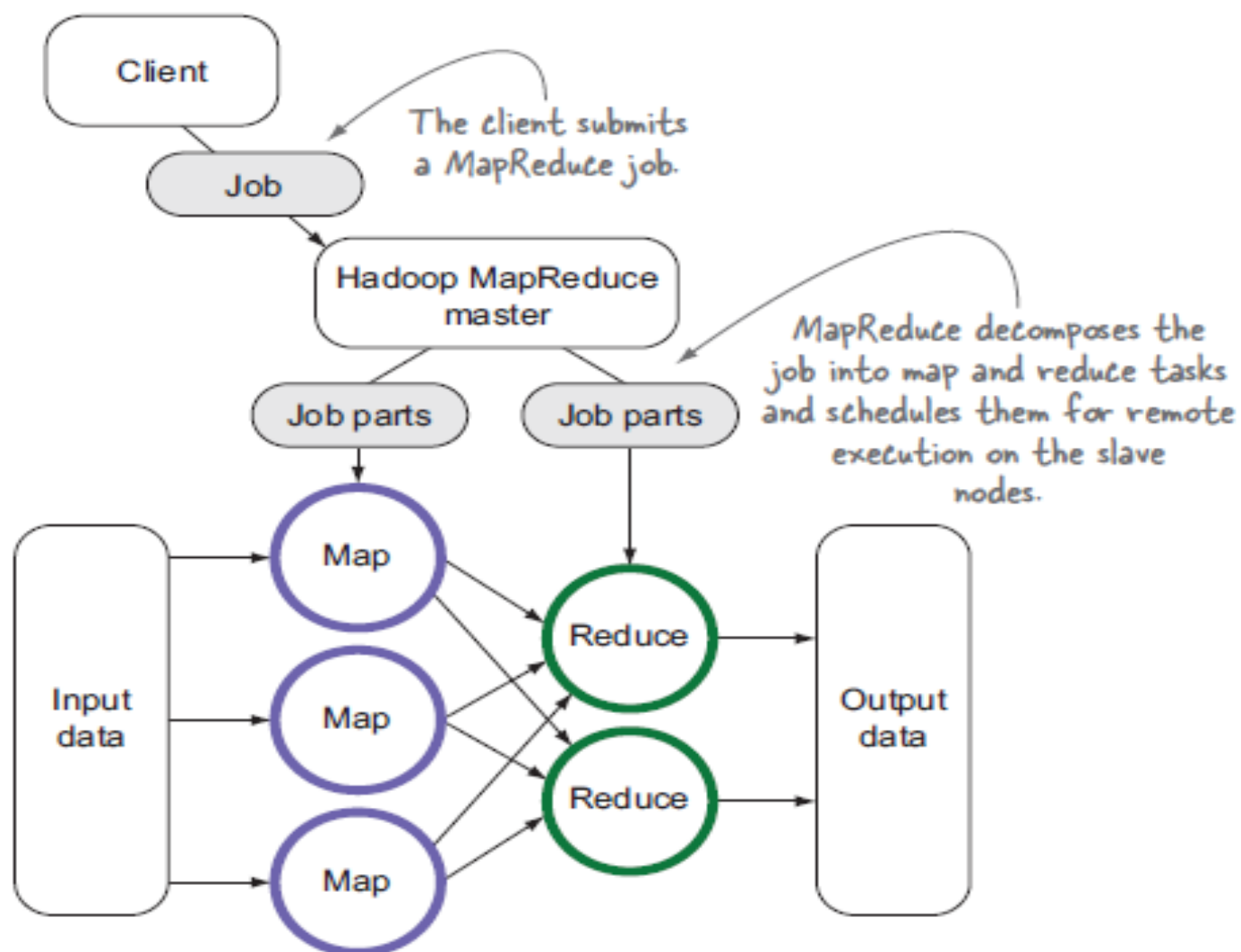
What is MapReduce Programming?

- MapReduce Programming is a software framework. MapReduce Programming helps you to process massive amounts of data in parallel



How MapReduce Programming Works





Mapper

The map function takes as input a key/value pair, which represents a logical record from the input data source.

In the case of a file, this could be a line, or if the input source is a table in a database, it could be a row.

map(key1, value1) → list(key2, value2)

The map function produces zero or more output key/value pairs for one input pair. For example, if the map function is a filtering map function, it may only produce output if a certain condition is met. Or it could be performing a demultiplexing operation, where a single key/value yields multiple key/value output pairs.

logical view of the map function that takes a key/value pair as input

Reducer

The reduce function is called once per unique map output key.

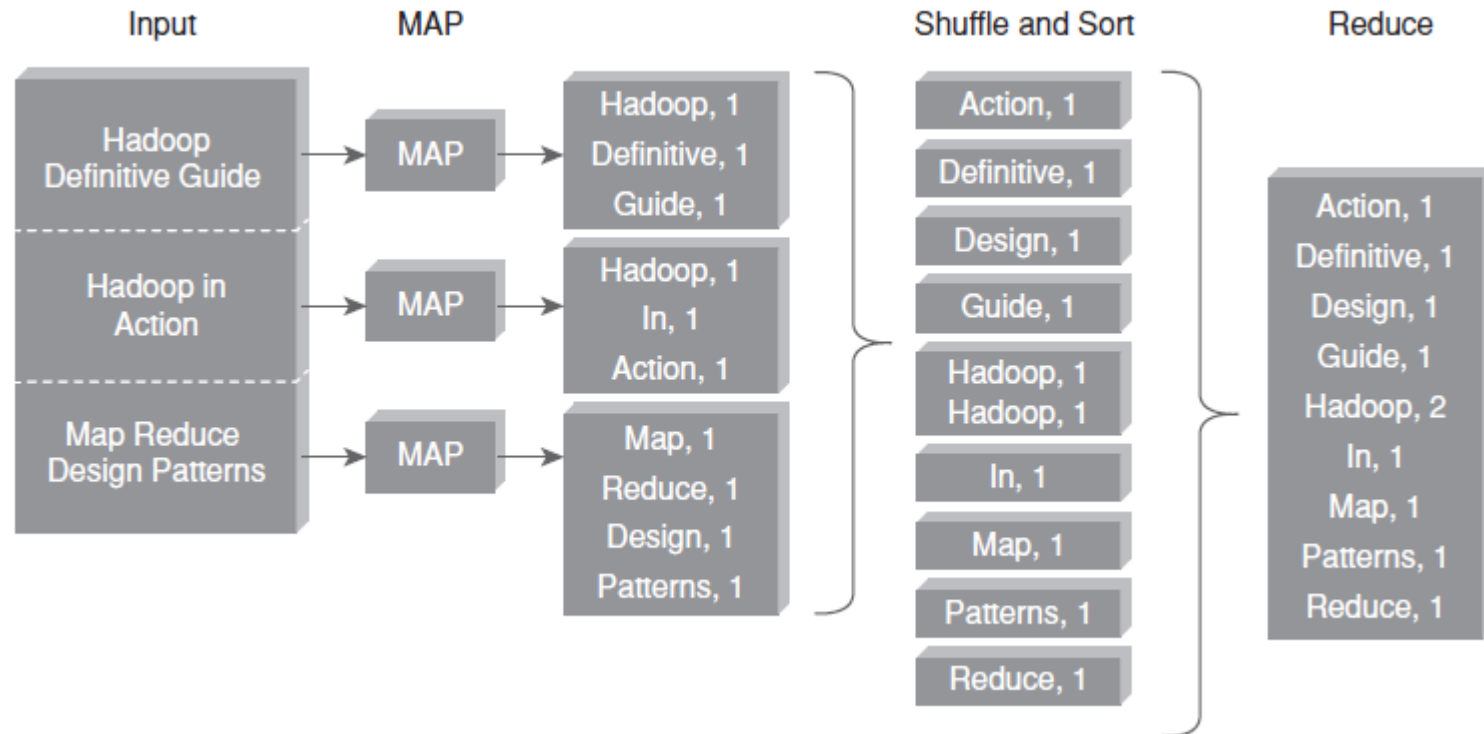
All of the map output values that were emitted across all the mappers for "key2" are provided in a list.

reduce (key2, list (value2's)) → list(key3, value3)

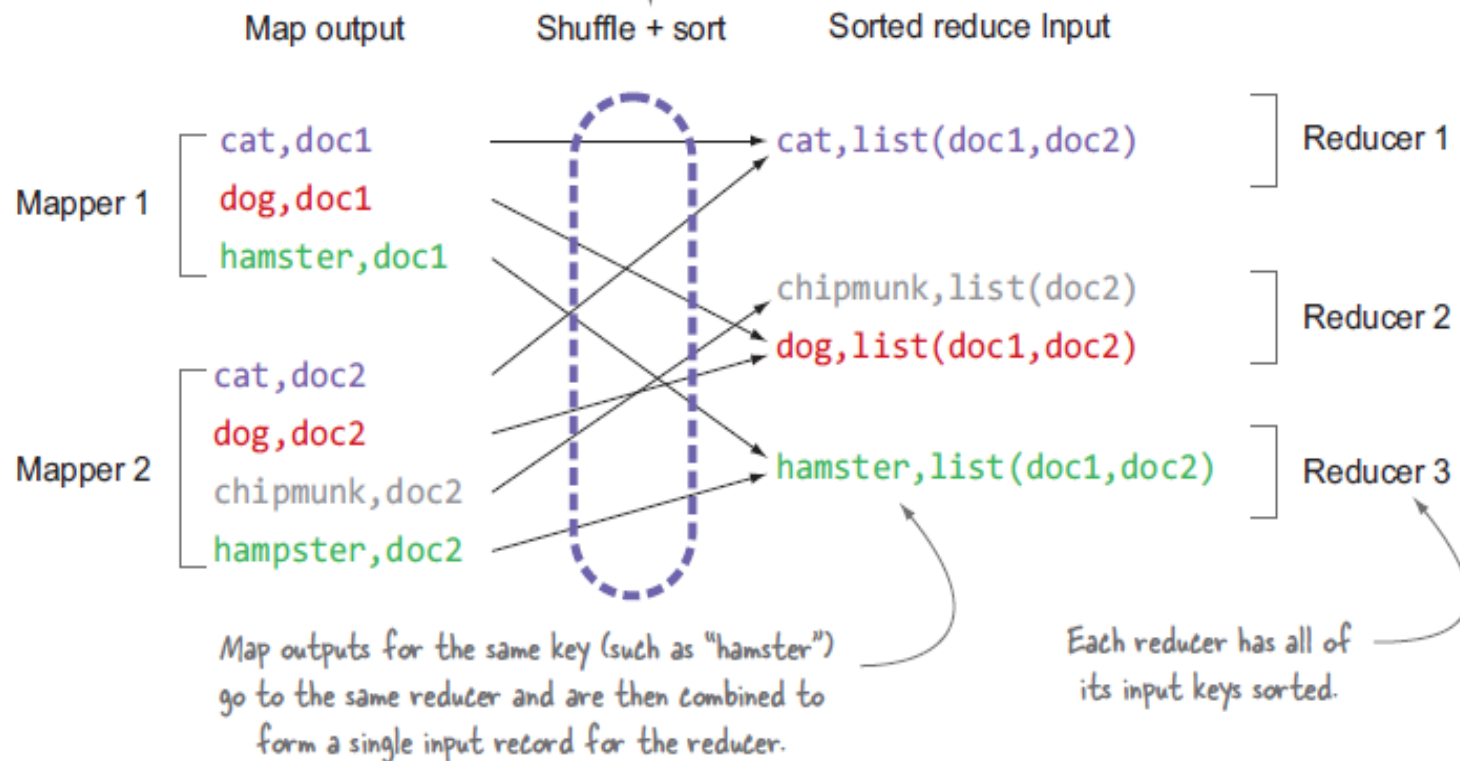
Like the map function, the reduce can output zero-to-many key/value pairs. Reducer output can write to flat files in HDFS, insert/update rows in a NoSQL database, or write to any data sink, depending on the requirements of the job.

A logical view of the reduce function that produces output for flat files, NoSQL rows, or any data sink

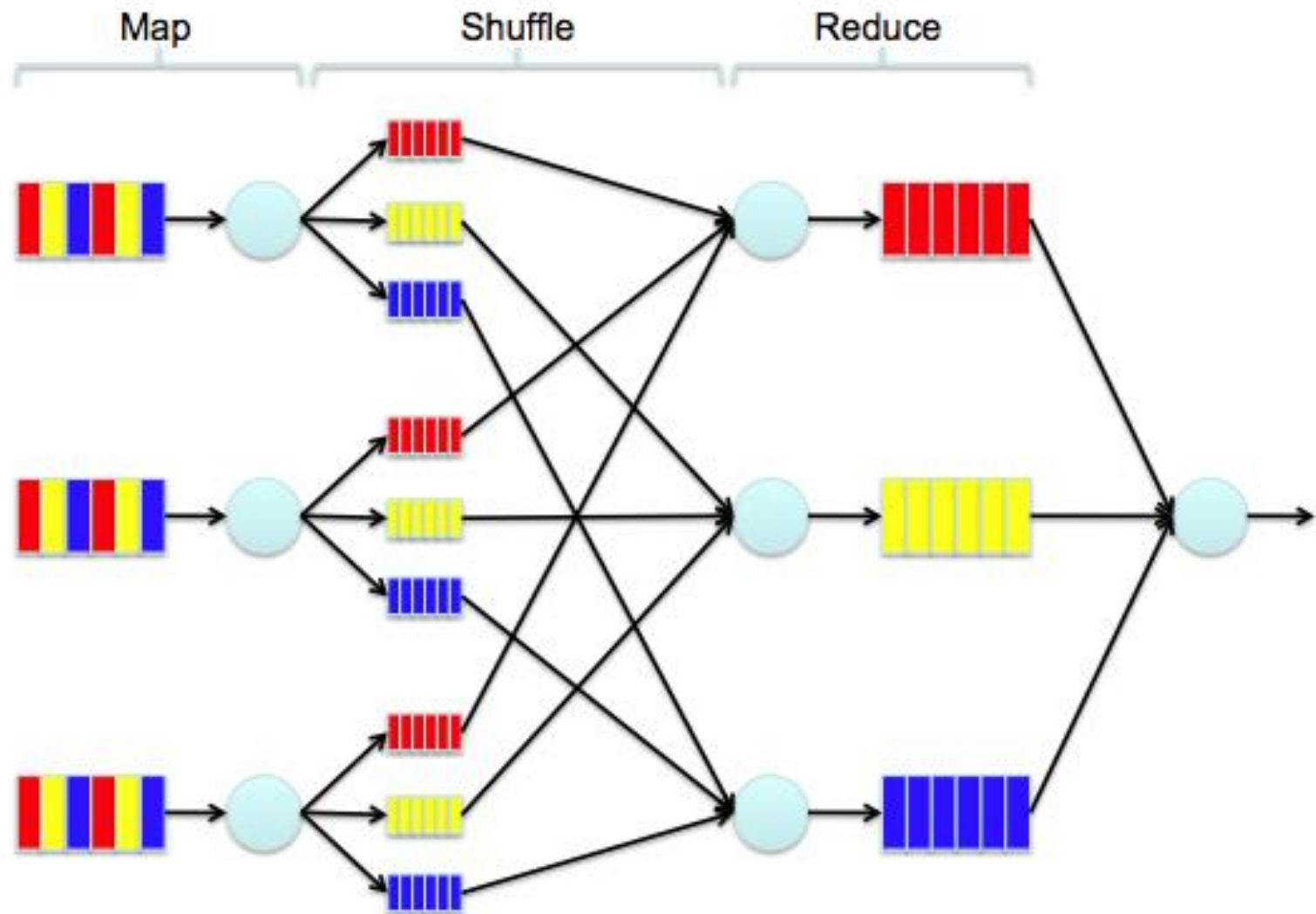
Wordcount Example



The shuffle and sort phases are responsible for two primary activities: determining the reducer that should receive the map output key/value pair (called partitioning); and ensuring that all the input keys for a given reducer are sorted.



MapReduce's shuffle and sort phases



Left outer join

