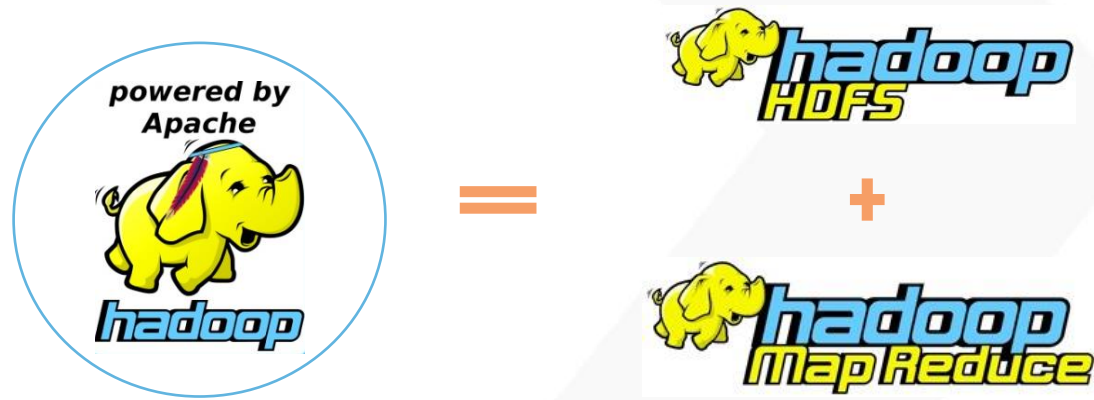


Hadoop Core Components



Disclaimer: This material is protected under copyright act AnalytixLabs ©, 2011-2015. Unauthorized use and/ or duplication of this material or any part of this material including data, in any form without explicit and written permission from AnalytixLabs is strictly prohibited. Any violation of this copyright will attract legal actions.

Hadoop Core Components



HADOOP-YARN-MAPREDUCE



What is MapReduce?

Break a large problem into sub-solutions

Map

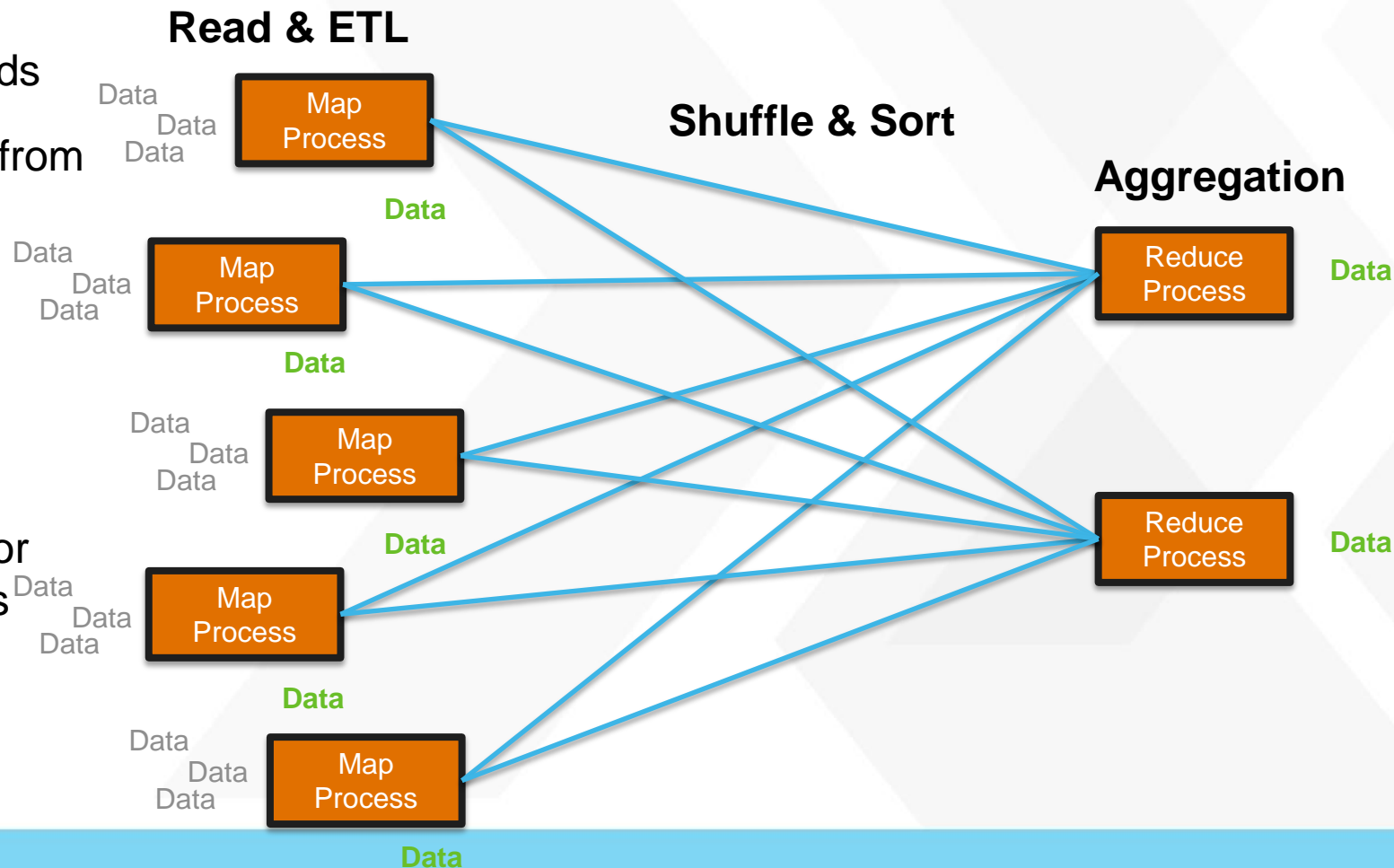
- Iterate over a large # of records
- Extract something of interest from each record

Shuffle

- Sort Intermediate results

Reduce

- Aggregate, summarize, filter or transform intermediate results
- Generate final output



MapReduce Paradigm – Map & Reduce

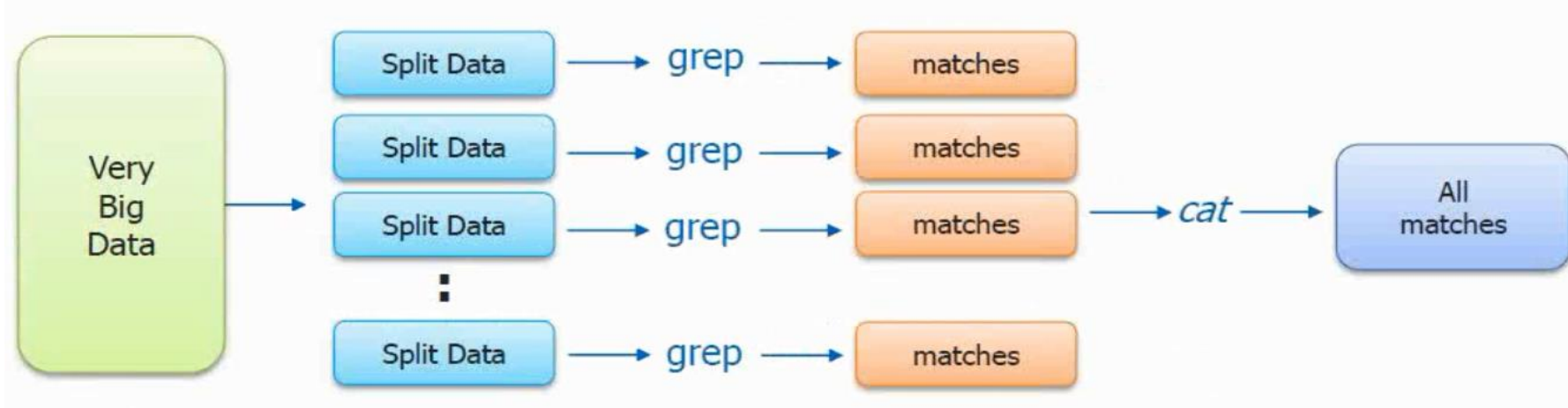
- MapReduce model consists two phases
- Map phase where input data is split into discrete chunks to be processed.
- Reduce phase where the output of the map phase is aggregated to produce the desired result.
- The simple nature of the programming model lends itself to very efficient and extremely large-scale implementations across thousands of cheap, commodity nodes
- Apache Hadoop MapReduce is an open-source, Apache Software Foundation project
- It is an implementation of the MapReduce programming paradigm

Features of MapReduce Programming

- Distributed and Parallel Processing
- Data Locality/Code Mobility
- Fault tolerance
- Speculative Execution
- Scalability (Scale Out Architecture)
- Reliability
- Batch Processing
- Sequential Access
- Optimized Scheduling
- Flexibility with coding language
- Security and Authentication
- Resiliency and High Availability

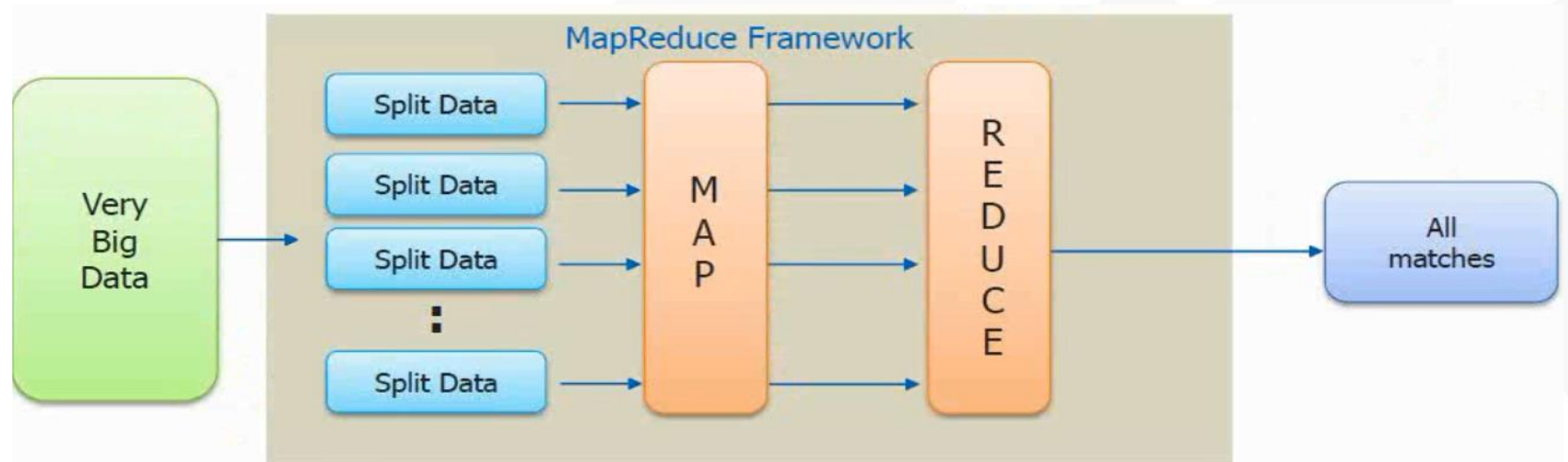
Traditional way Vs. Map Reduce Way

Lets take example of identify matching records in a data set with specific pattern



Traditional Way

Map Reduce Way



Map Reduce Components

- MapReduce project can be broken down into the following major facets:
- The end-user ***MapReduce API*** for programming the desired MapReduce application.
- The ***MapReduce framework***, which is the runtime implementation of various phases such as the map phase, the sort/shuffle/merge aggregation and the reduce phase.
- The ***MapReduce system***, which is the backend infrastructure required to run the user's MapReduce application, manage cluster resources, schedule thousands of concurrent jobs

Map Reduce System

Hadoop 1.X

- MapReduce system is composed of JobTracker and TaskTrackers.
- JobTracker is master process and its runs on the master node. One per cluster
- TaskTrackers are the slaves. On each node one task tracker runs

Hadoop 2.x

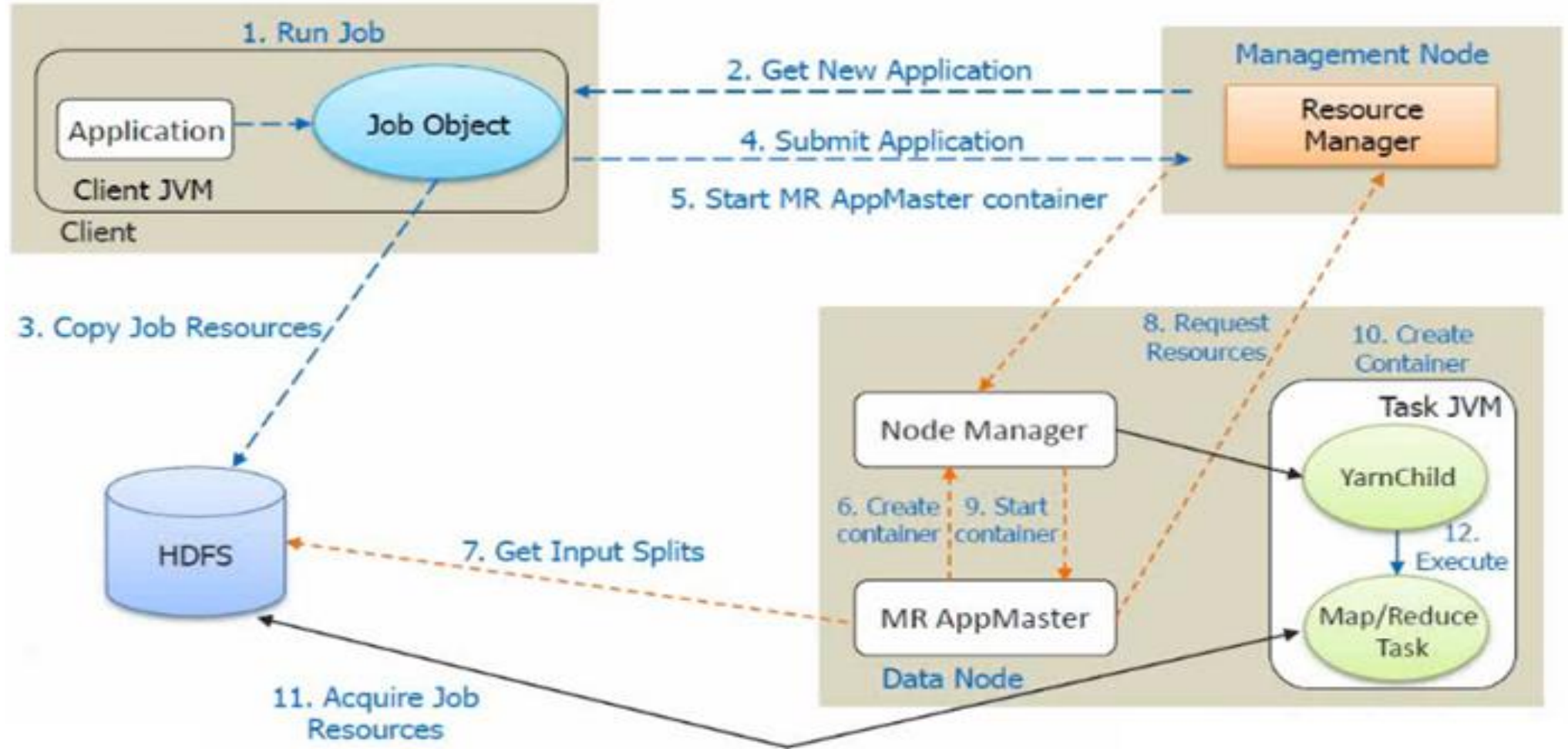
- MapReduce system is composed of Resource Manager and Node Manager
- Resource Manager is Master process and it runs on Master Node. One active RM per cluster
- Node Manager is slaves process and its runs on slaves/data nodes.

Hadoop MapReduce Requirements

The user/developer is required to set the framework with the following parameters:

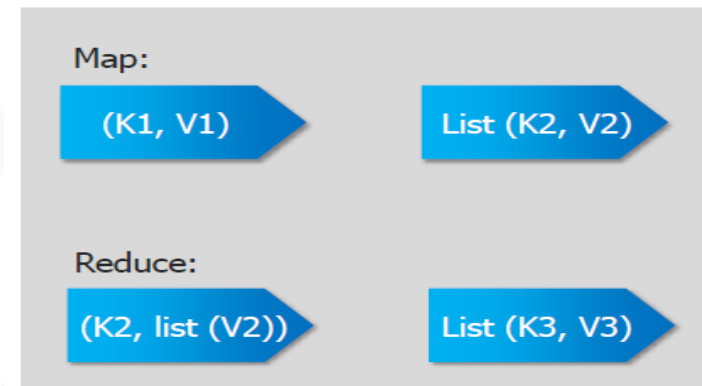
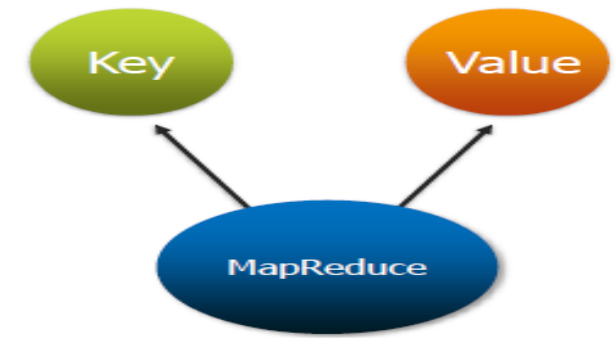
- ✓ The location(s) of the input datasets to be processed in HDFS
- ✓ The location of the job output in the HDFS
- ✓ The input format
- ✓ The output format
- ✓ The class containing the map function (Mapper Class)
- ✓ The class containing the reduce function (Reducer Class)
- ✓ Custom Partitioner class if required (optional)
- ✓ Combiner Class if required (Optional)

MapReduce Application Execution

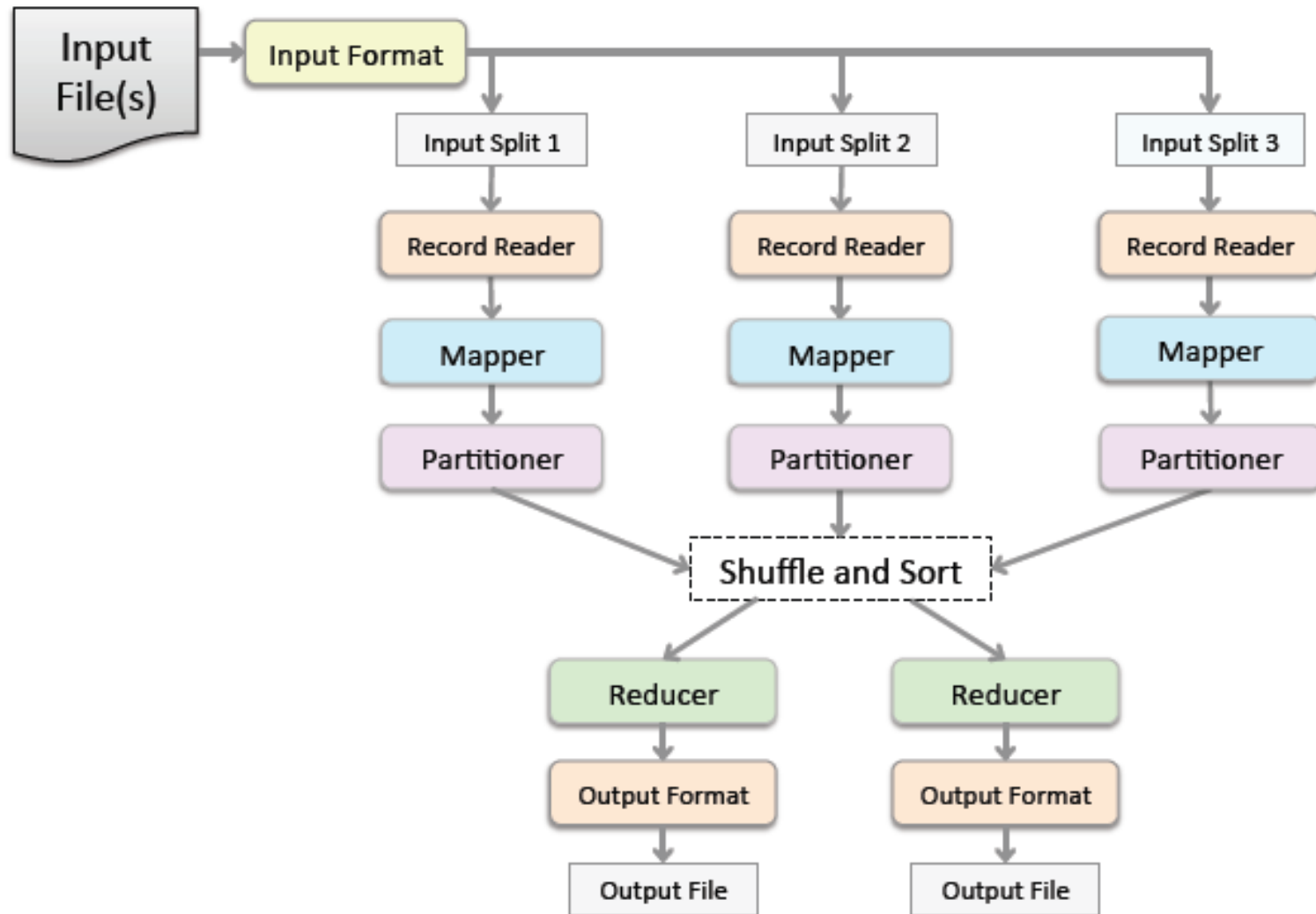


MapReduce Model

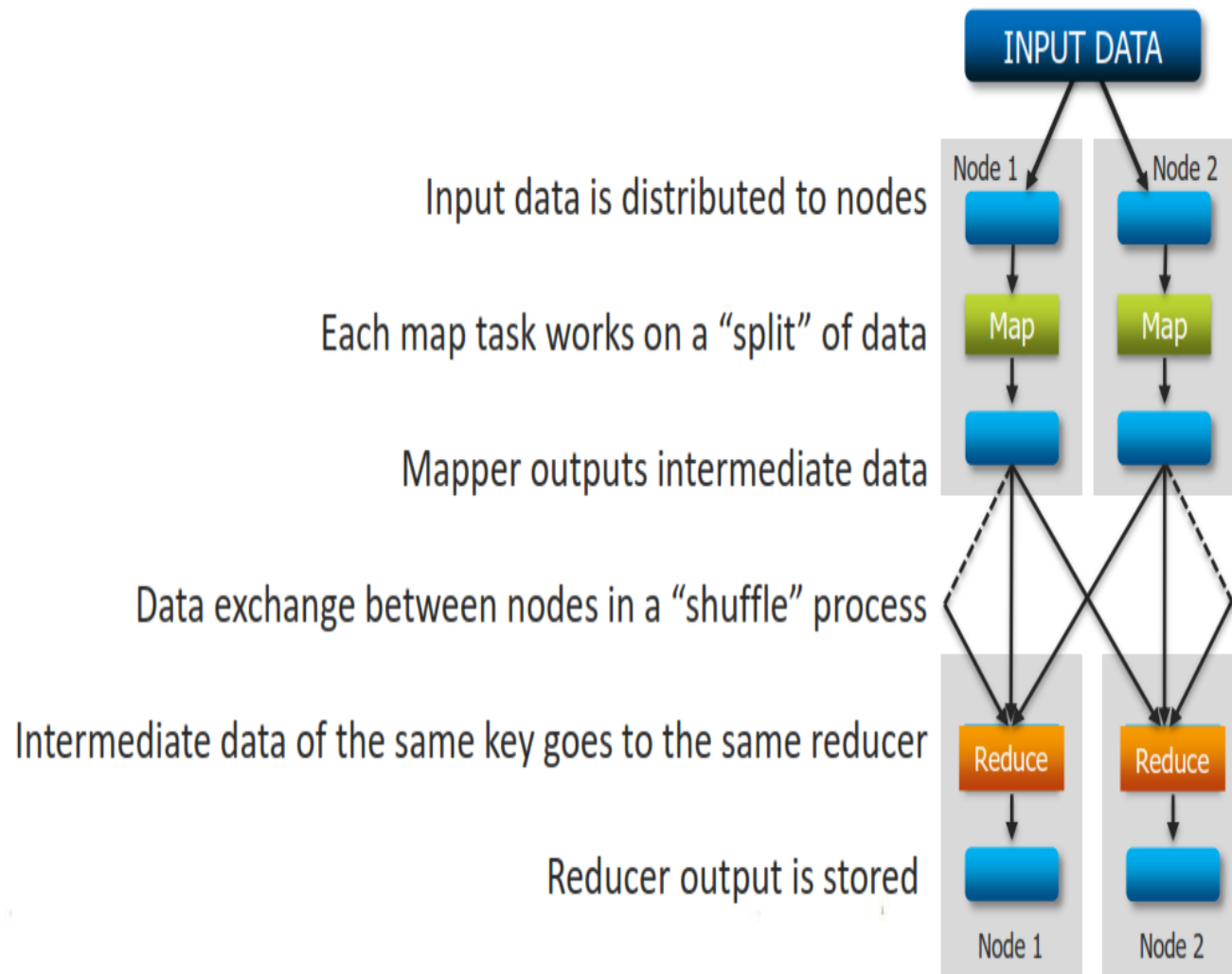
- **Divided in two phases**
 - Map phase
 - Reduce phase
- **Both phases use key-value pairs as input and output**
- **Defines map and reduce functions**
map: $(K1, V1) \rightarrow \text{list}(K2, V2)$
reduce: $(K2, \text{list}(V2)) \rightarrow \text{list}(K3, V3)$
 1. Map function is applied to every input key-value pair
 2. Map function generates intermediate key-value pairs
 3. Intermediate key-values are sorted and grouped by key
 4. Reduce is applied to sorted and grouped intermediate key-values
 5. Reduce emits result key-values



Map Reduce: Big Picture



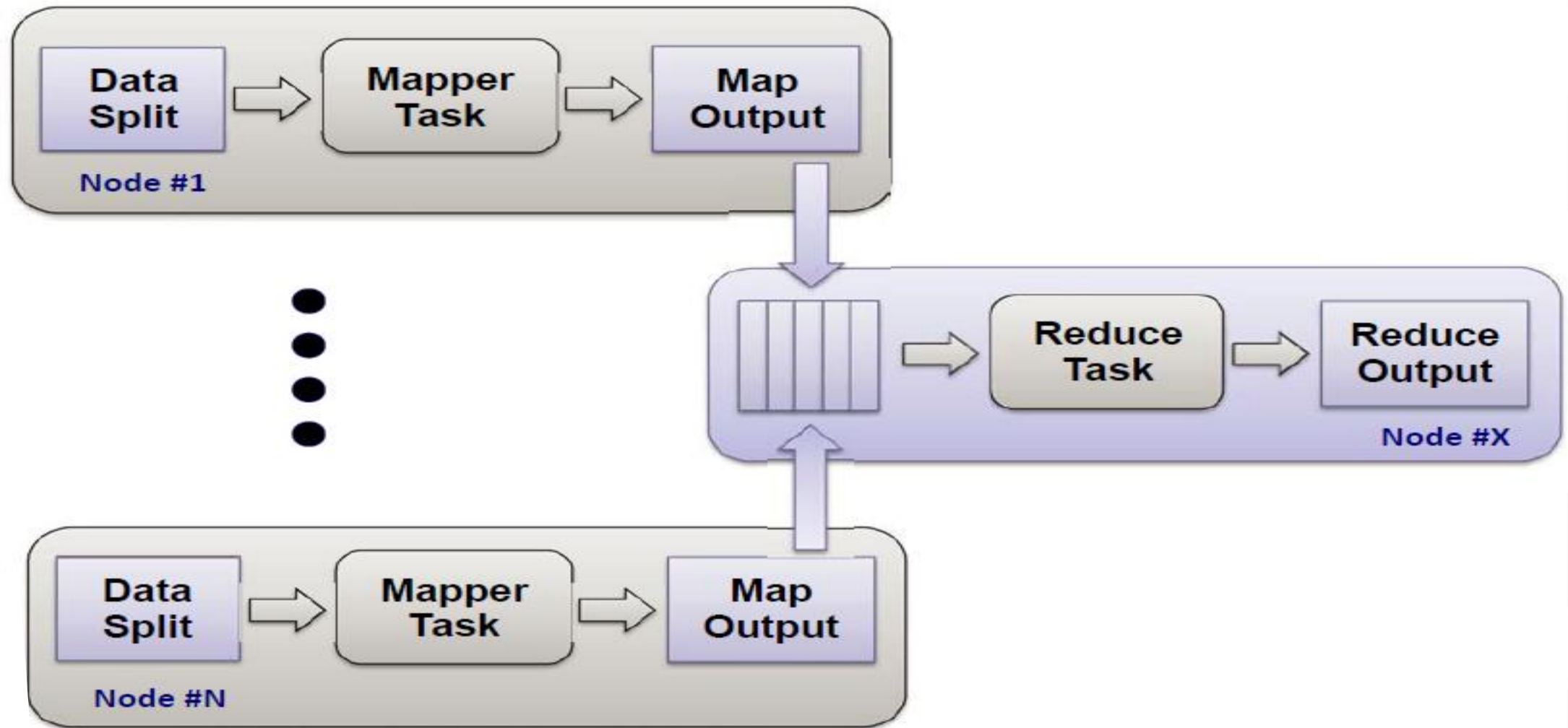
Flow of MapReduce



Input splits→Map phase→Reduce phase

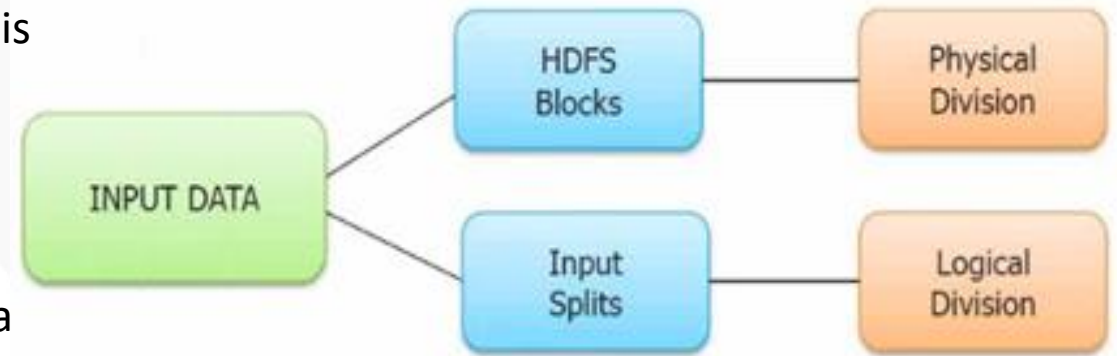
- ✓ Map phase: The input is split into sub jobs and these sub jobs are mapped to different CPU's/processors.
- ✓ Mappers work on processors, meaning, for every input split there will be a mapper working on it.
- ✓ After the Map phase, there is something called Sorting and Shuffling, where all the keys and values of each mapper will be shuffled and arranged in a sorted order based on the keys. Lastly, all the output of the processors (Mappers) are collected and sent to the Reduce phase.
- ✓ In the Mapper phase, key and values will be prepared, and in the Reduce phase, all the values for each unique key will be reduced to get the final result.

Map Reduce Flow of Data



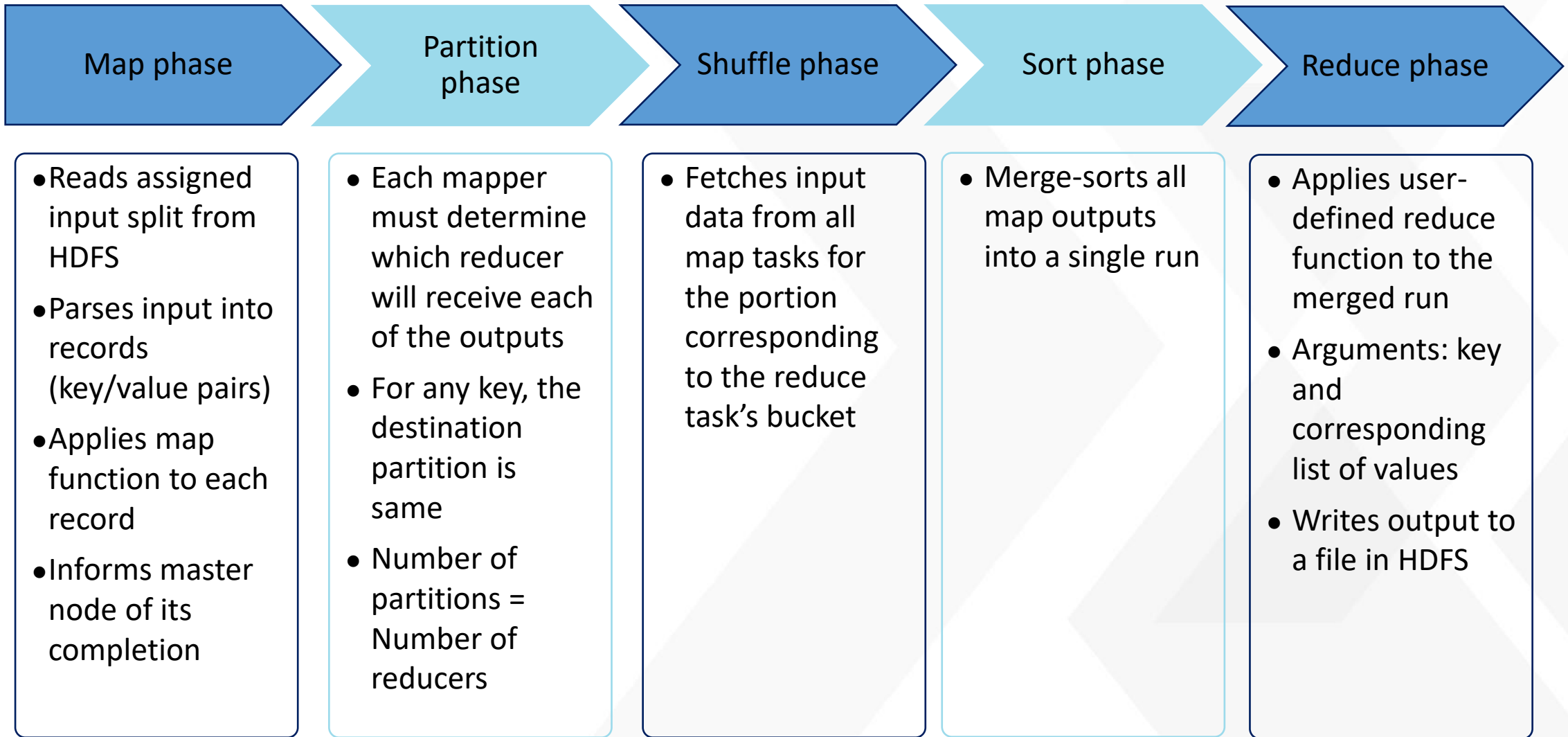
Input Splits Vs. HDFS Blocks

- ✓ Block is a continuous location on the hard drive where data is stored. File system stores data as collection of blocks.
- ✓ Input Split is the chunk of the input data sets or input file to be processed by MapReduce framework. The split is divided into records and each record is processed by the map
- ✓ Block is physical notation of data. It means it does store data physically.
- ✓ Whereas Split is logical notation of data. It does not store data but reference to data. And it honors logical boundaries of records

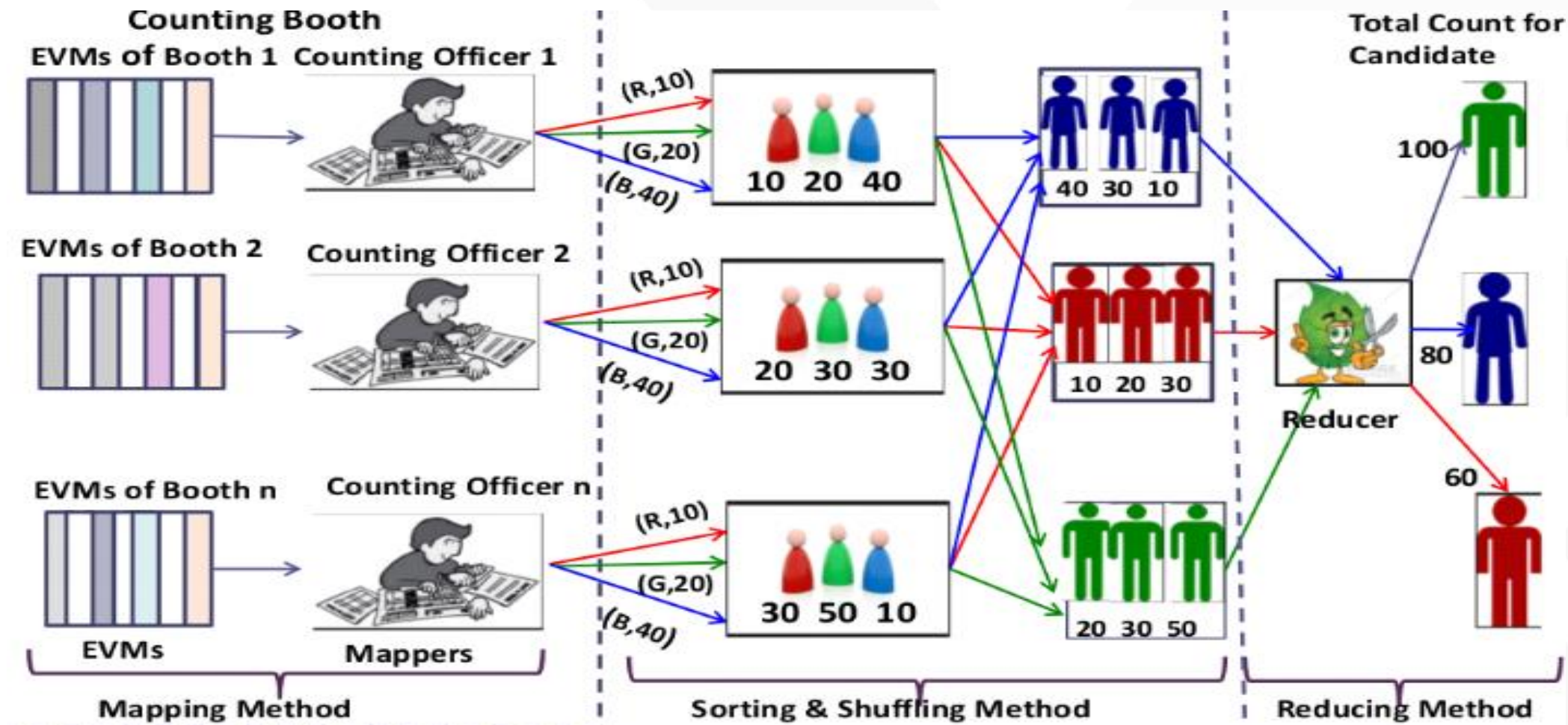


- Logical records do not fit neatly into the HDFS blocks.
- Logical records are lines that cross the boundary of the blocks.
- First split contains line 5 although it spans across blocks.

Typical Stages of Mapreduce



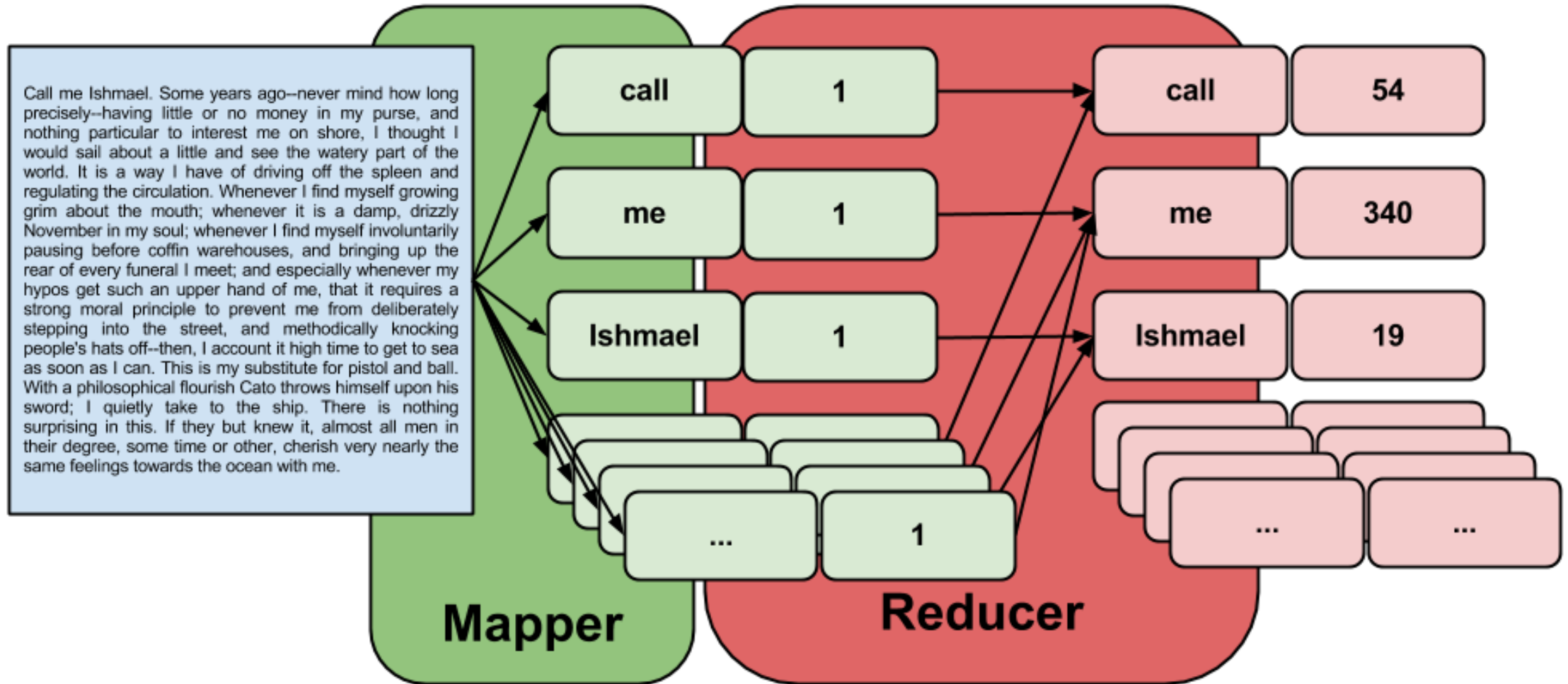
Example: Comparison of Election Counting with Mapreduce



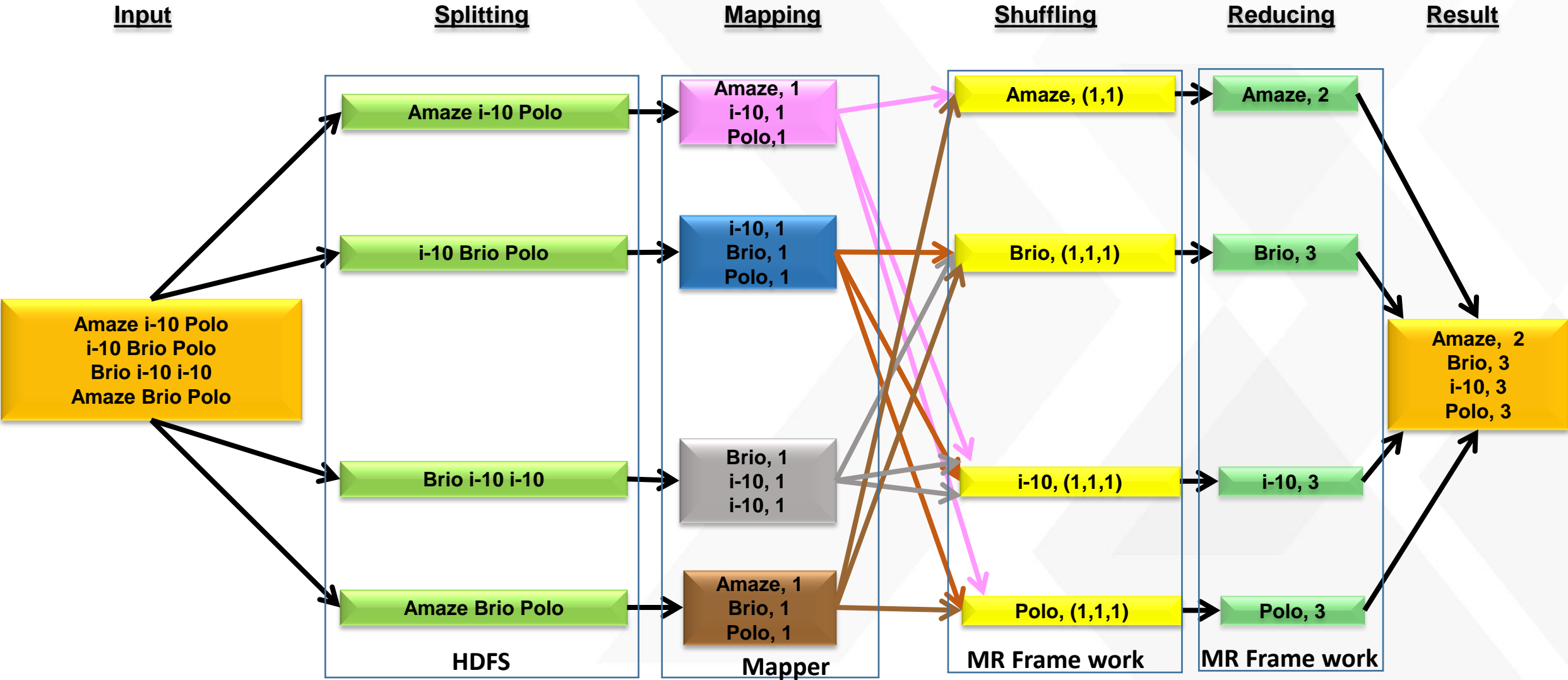
Example: Comparison of Election Counting with Mapreduce

- ✓ In India, after the elections, all the EVM's are brought to one place for counting and then Polling officers perform the count of the votes stored in EVM.
- ✓ This means the actual work is done by polling officer but that work is performed on EVM machine.
- ✓ Let's now link the components of the election with the actual components of MapReduce.
- ✓ **Input Splits:** Here, Input split is the EVM's that corresponds to one polling booth and votes stored in one EVM is calculated by one polling officer.
- ✓ **Map Phase:** In the Map phase, each Polling officer gets the ballot count of each candidate, in his respective polling booth. This is done simultaneously for each polling booth. From here, each candidate will become the key and the number of votes for the candidate will be the values.
- ✓ **Reduce Phase:** In the Reduce phase, the ballot count for each booth under a parliament seat position is taken and results are generated for each candidate.
- ✓ Which means that all the individual results of each polling booth will be collected and counted based on the keys. Finally, the total number of votes generated for that candidate will be calculated.

Example: Word count-MapReduce Algorithm

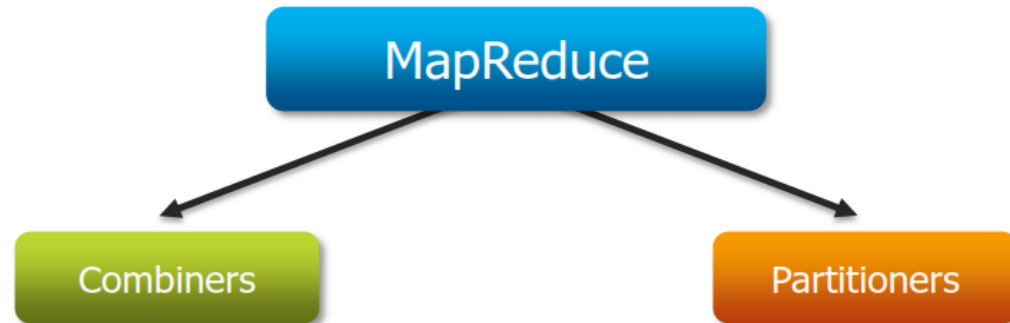


Example: Word count-MapReduce Algorithm



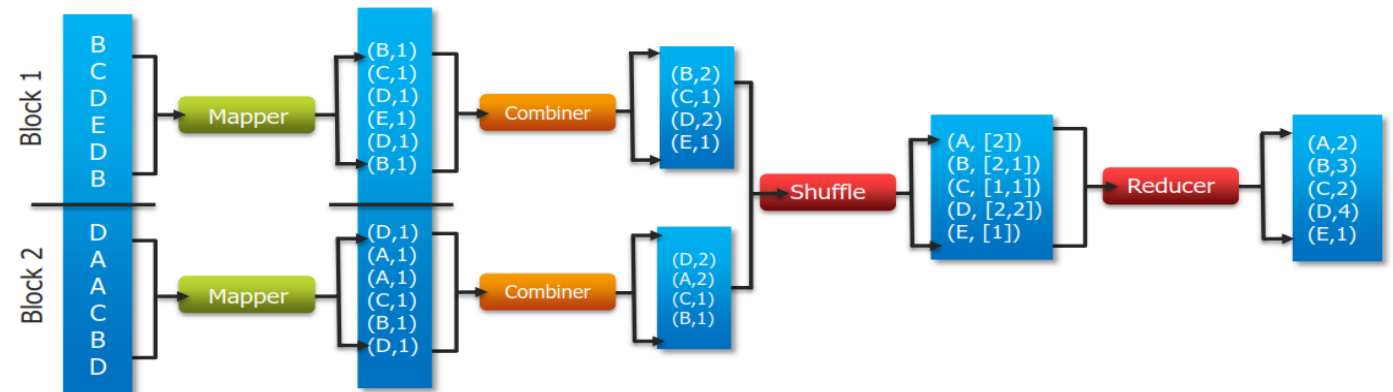
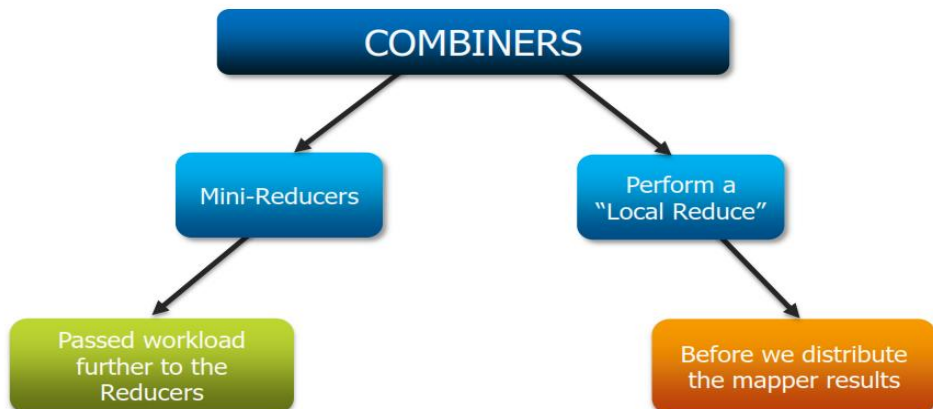
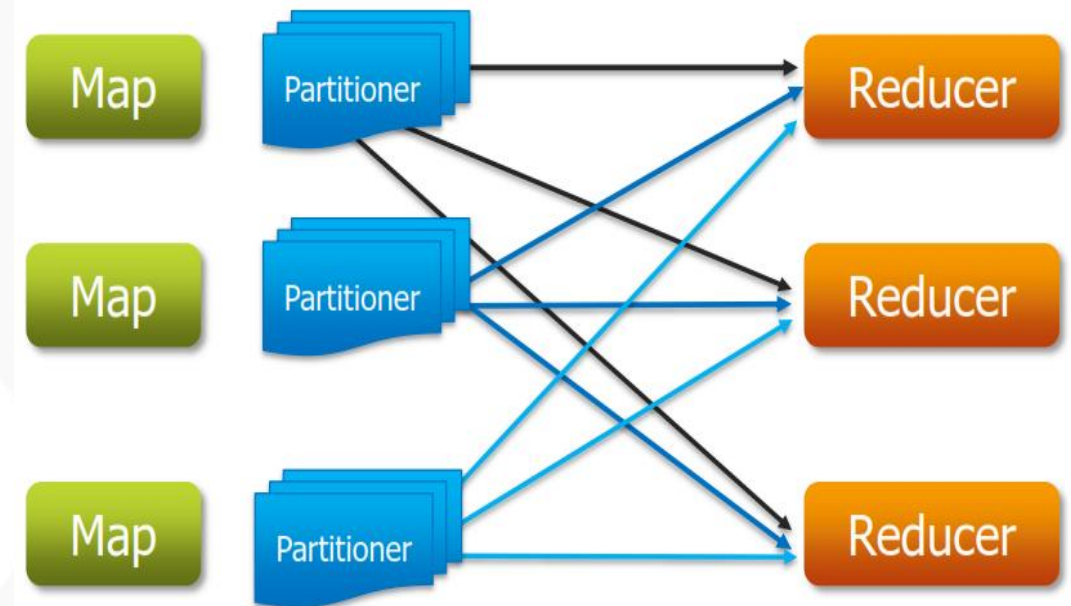
Other topics of MR

Complete view of MapReduce, illustrating combiners and partitioners in addition to Mappers and Reducers



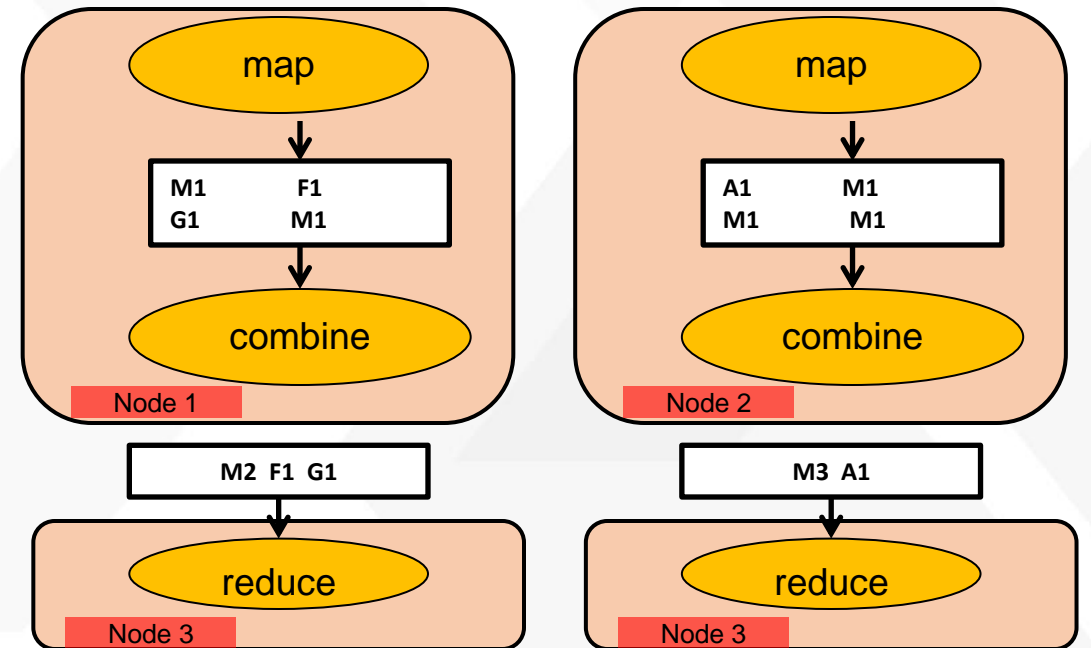
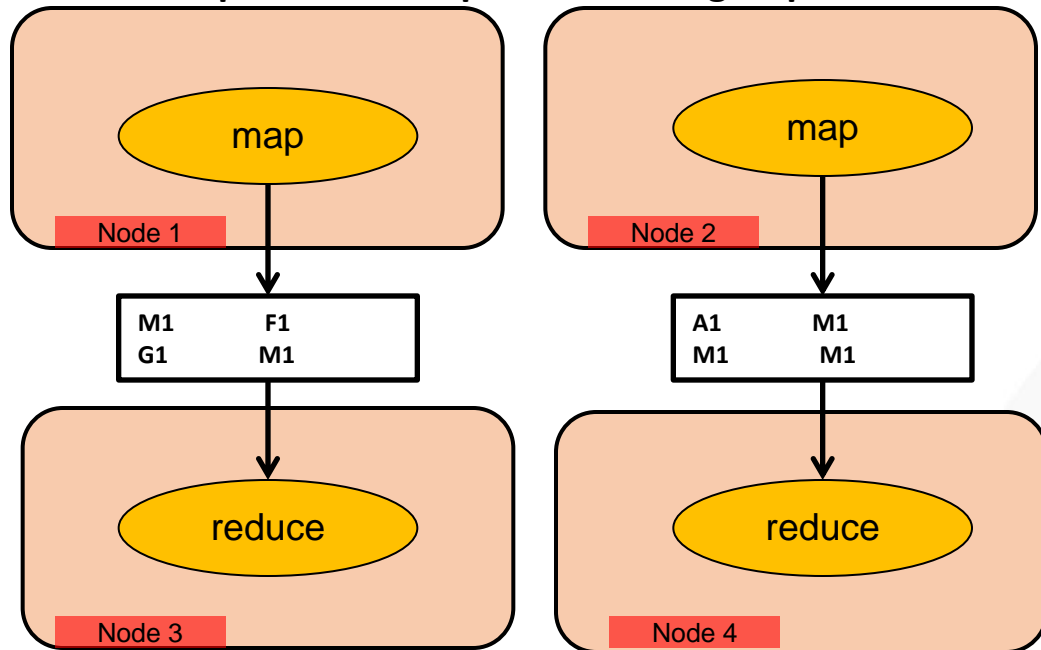
Combiners can be viewed as 'mini-reducers' in the Map phase.

Partitioners determine which reducer is responsible for a particular key.



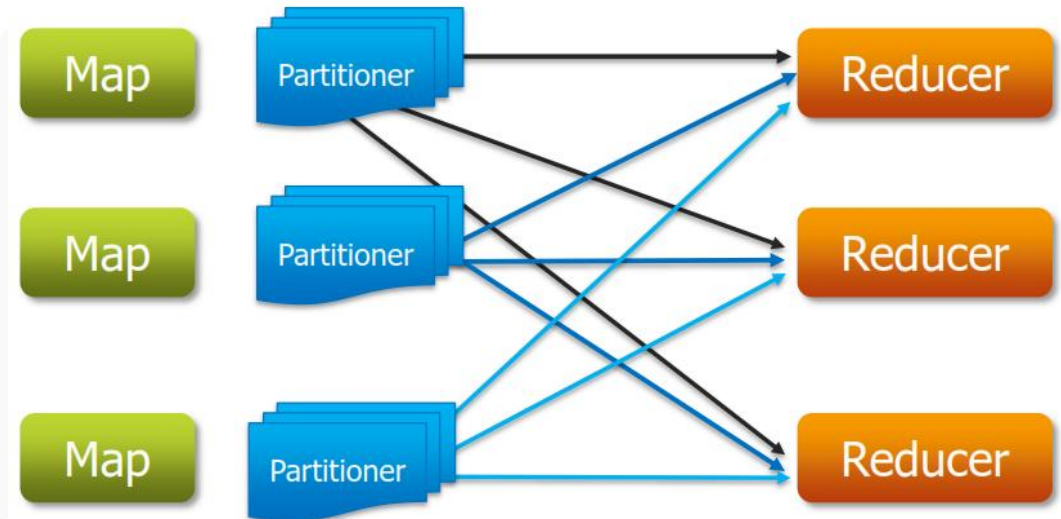
Combiner (Local Reducers)

- Runs on output of map function
- Produces output for reduce function
 - input (file) -> record(K1,V1)
 - map: (K1,V1) \rightarrow list (K2,V2)
 - combine: List (K2,V2) \rightarrow (K2 list(V2))
 - reduce: (K2,list(V2)) \rightarrow list (K3,V3)
- Often is the same class as Reducer
- Each combine processes output from a single split



Partitioners

- ✓ Partitioners determine which reducer is responsible for a particular key
- ✓ HashPartitioner is the default partitioner in Hadoop, which creates one Reduce task for each unique “key”.
- ✓ All the values with the same key goes to the same instance of your reducer, in a single call to the reduce function.
- ✓ If user is interested to store a particular group of results in different reducers, then the user can write his own partitioner(custom) implementation.
- ✓ It can be general purpose or custom made to the specific data types or values that you expect to use in user application.
 - ✓ **Custom Partitioner** is a process that allows you to store the results in different reducers, based on the user condition.
 - ✓ By setting a partitioner to partition by the key, we can guarantee that, records for the same key will go to the same reducer.
 - ✓ A partitioner ensures that only one reducer receives all the records for that particular key.



Counters

A Counter is generally used to keep track of occurrences of any events. In Hadoop Framework, whenever any MapReduce job gets executed, the Hadoop Framework initiates counters to keep track of the job statistics like number of rows read, number of rows written as output etc.

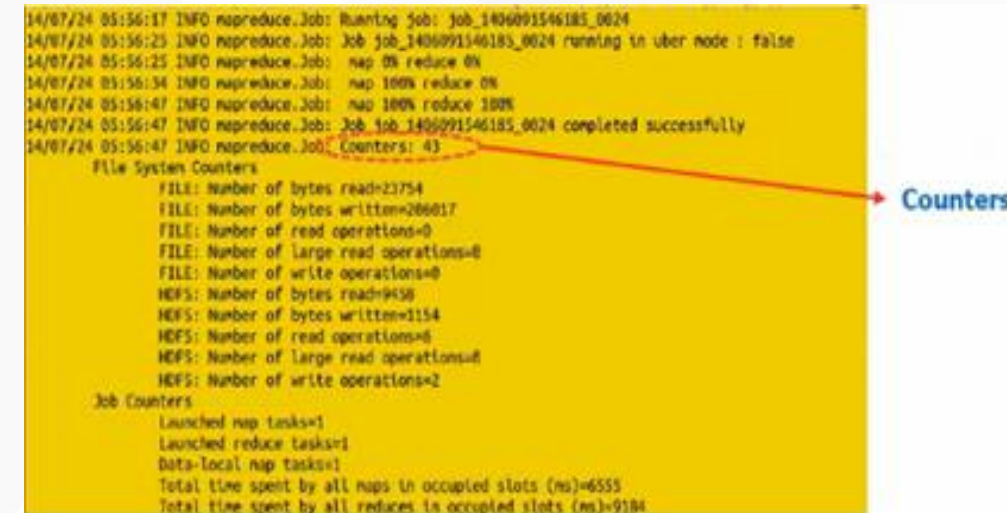
These are built in counters in Hadoop Framework. Additionally, we can also create and use our own custom counters.

Typically some of the operations of Hadoop counters are:

- ✓ Number of mapper and reducer launched..
 - ✓ The number of bytes was read and written
 - ✓ The number of tasks was launched and successfully ran
 - ✓ The amount of CPU and memory consumed is
- Appropriate or not for your job and cluster nodes

Built in counters are of three types:

- ✓ Mapreduce Task Counters
- ✓ File system counters
- ✓ Job Counters



```
14/07/24 05:56:17 INFO mapreduce.Job: Running job: job_1406091546185_0024
14/07/24 05:56:25 INFO mapreduce.Job: Job job_1406091546185_0024 running in uber mode : false
14/07/24 05:56:25 INFO mapreduce.Job: map 0% reduce 0%
14/07/24 05:56:34 INFO mapreduce.Job: map 100% reduce 0%
14/07/24 05:56:47 INFO mapreduce.Job: map 100% reduce 100%
14/07/24 05:56:47 INFO mapreduce.Job: Job job_1406091546185_0024 completed successfully
14/07/24 05:56:47 INFO mapreduce.Job: Counters: 43
  File System Counters
    FILE: Number of bytes read=23754
    FILE: Number of bytes written=266017
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=9456
    HDFS: Number of bytes written=2154
    HDFS: Number of read operations=0
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
  Job Counters
    Launched map tasks=1
    Launched reduce tasks=1
    Data-local map tasks=1
    Total time spent by all maps in occupied slots (ms)=6555
    Total time spent by all reduces in occupied slots (ms)=9184
```

Mapreduce Job Input-Output formats

Input

Input Format describes the input specification for a MapReduce job. The MapReduce framework relies on the InputFormat of the job to:

- Validate the input specification of the job.
- Split up the input file(s) into logical Input Split instances, each of which is then assigned to an individual Mapper.
- Provide the RecordReader implementation used to read input records from the logical InputSplit for processing by the Mapper .

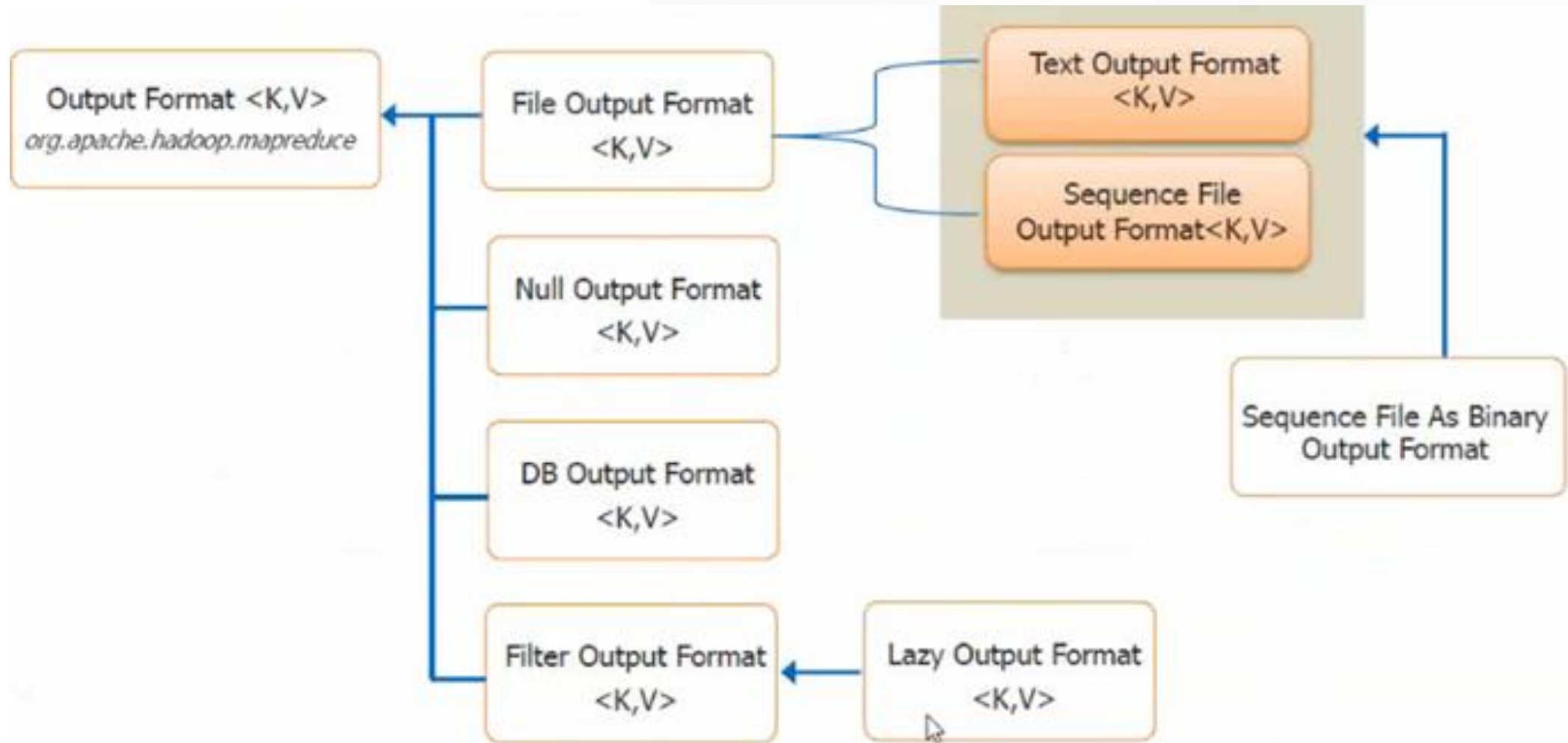
Output

OutputFormat describes the output-specification for a MapReduce job. The MapReduce framework relies on the OutputFormat of the job to:

- Validate the output-specification of the job; for example, check that the output directory doesn't already exist.
- Provide the RecordWriter implementation used to write the output files of the job. Output files are stored in a FileSystem. TextOutputFormat is the defaultOutputFormat.



Output Formats



Contact Us

Visit us on: <http://www.analytixlabs.in/>

For more information, please contact us: <http://www.analytixlabs.co.in/contact-us/>

Or email: info@analytixlabs.co.in

Call us we would love to speak with you: (+91) 9910509849

Join us on:

Twitter - <http://twitter.com/#!/AnalytixLabs>

Facebook - <http://www.facebook.com/analytixlabs>

LinkedIn - <http://www.linkedin.com/in/analytixlabs>

Blog - <http://www.analytixlabs.co.in/category/blog/>