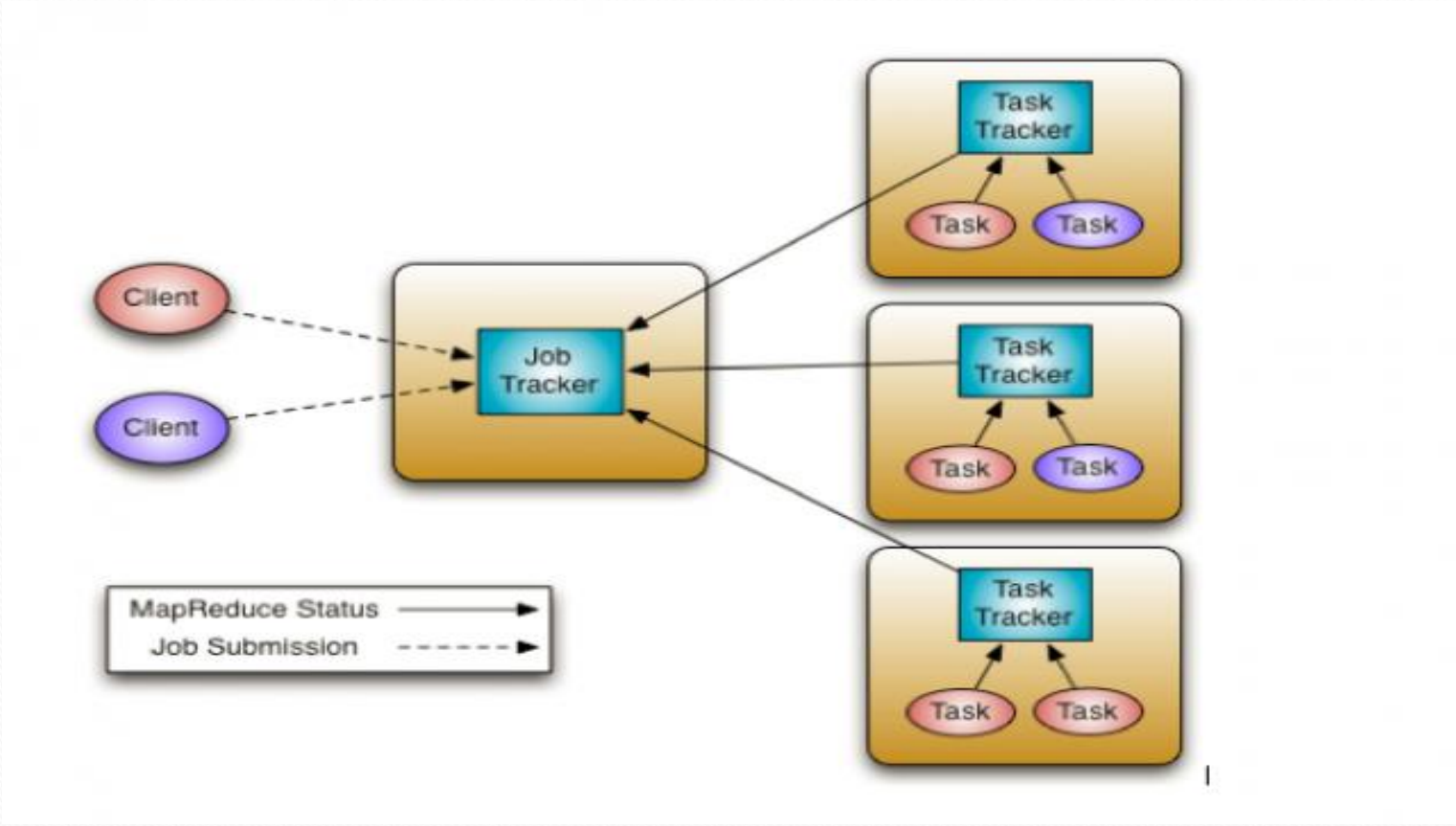
Map Reduce

Map Reduce

- A MapReduce is a data processing tool which is used to process the data parallelly in a distributed form.
- It was developed in 2004, on the basis of paper titled as "MapReduce: Simplified Data Processing on Large Clusters," published by Google.
- The MapReduce is a paradigm which has two phases, the mapper phase, and the reducer phase.
- In the Mapper, the input is given in the form of a key-value pair.
- The output of the Mapper is fed to the reducer as input.
- The reducer runs only after the Mapper is over.
- The reducer too takes input in key-value format, and the output of reducer is the final output.

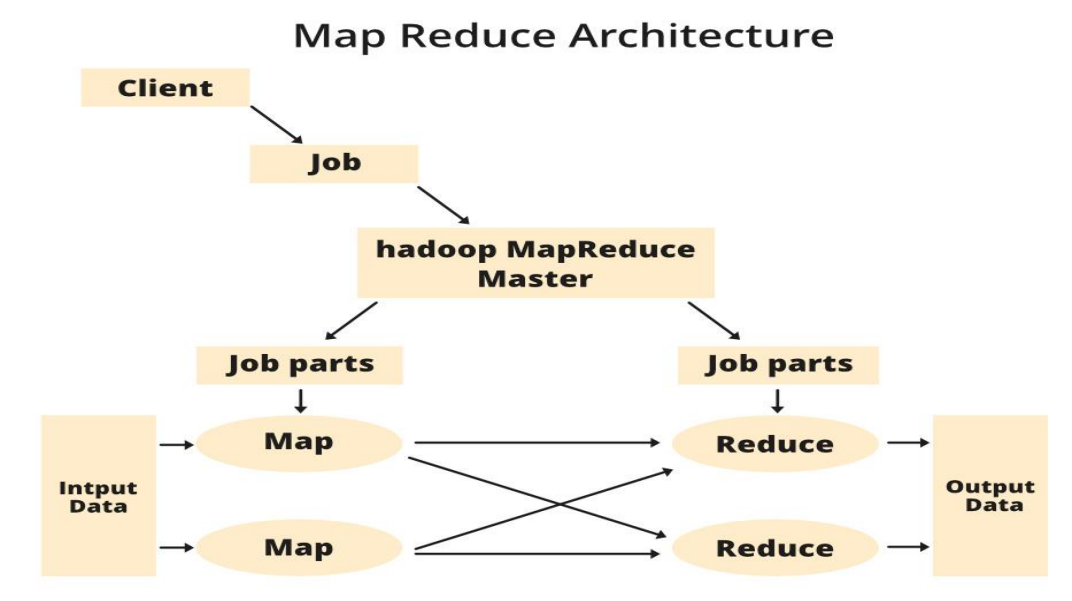
Map Reduce Architecture



- How the Job tracker and the task tracker deal with MapReduce:
- Job Tracker:
  - The work of Job tracker is to manage all the resources and all the jobs across the cluster and also to schedule each map on the Task Tracker running on the same data node since there can be hundreds of data nodes available in the cluster.
- Task Tracker:
  - The Task Tracker can be considered as the actual slaves working on the instruction given by the Job Tracker.
  - This Task Tracker is deployed on each of the nodes available in the cluster that executes the Map and Reduce task as instructed by Job Tracker.

- Single JobTracker per master
  - Responsible for scheduling the jobs' component tasks on the slaves
  - Monitors slave progress
- Re-executing failed tasks
  - Single TaskTracker per slave
  - Execute the tasks as directed by the master

Components of MapReduce Architecture:



Components of MapReduce Architecture:

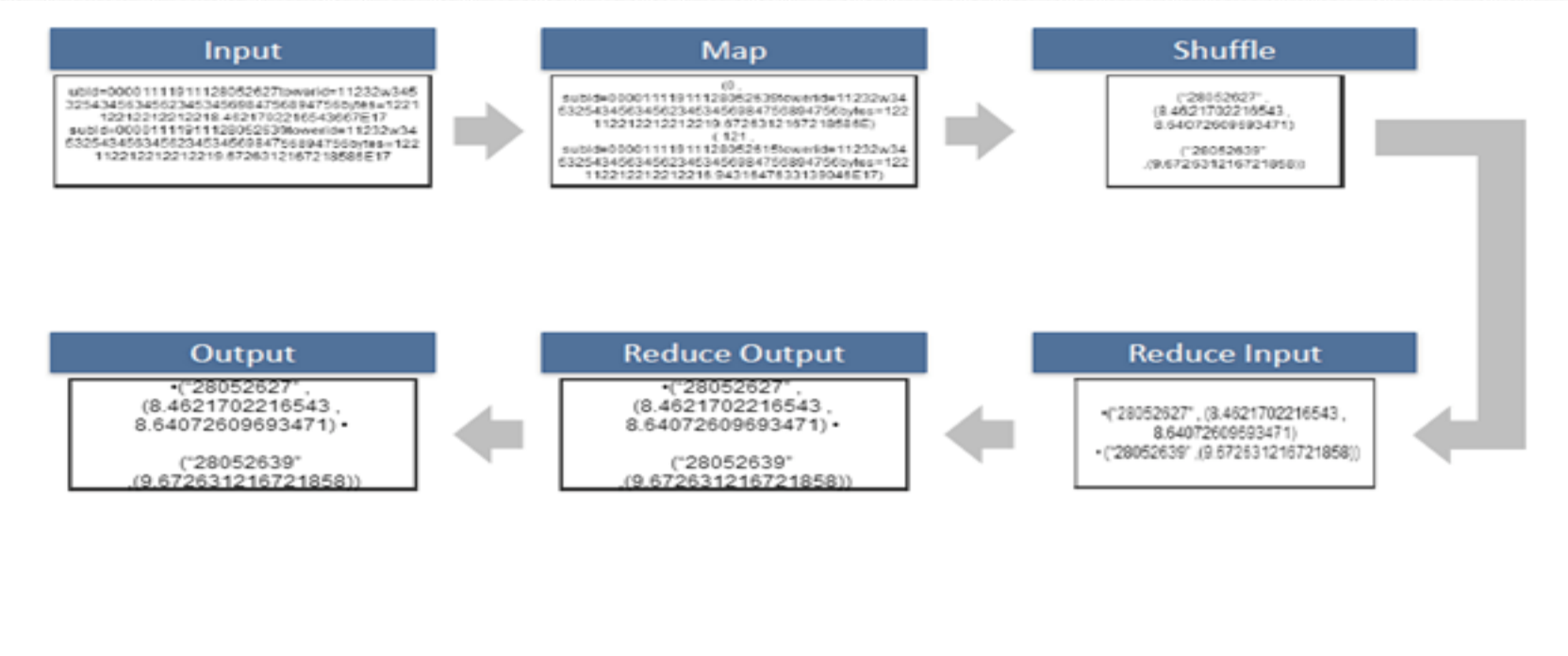
1. **Client:** The MapReduce client is the one who brings the Job to the MapReduce for processing. There can be multiple clients available that continuously send jobs for processing to the Hadoop MapReduce Manager.
2. **Job:** The MapReduce Job is the actual work that the client wanted to do which is comprised of so many smaller tasks that the client wants to process or execute.
3. **Hadoop MapReduce Master:** It divides the particular job into subsequent job-parts.
4. **Job-Parts:** The task or sub-jobs that are obtained after dividing the main job. The result of all the job-parts combined to produce the final output.
5. **Input Data:** The data set that is fed to the MapReduce for processing.
6. **Output Data:** The final result is obtained after the processing.

Explanation

- The client will submit the job of a particular size to the Hadoop MapReduce Master. Now, the MapReduce master will divide this job into further equivalent job-parts.
- These job-parts are then made available for the Map and Reduce Task.
- This Map and Reduce task will contain the program as per the requirement of the use-case that the particular company is solving.
- The developer writes their logic to fulfill the requirement that the industry requires.
- The input data which we are using is then fed to the Map Task and the Map will generate intermediate key-value pair as its output.
- The output of Map i.e. these key-value pairs are then fed to the Reducer and the final output is stored on the HDFS.
- There can be n number of Map and Reduce tasks made available for processing the data as per the requirement.
- The algorithm for Map and Reduce is made with a very optimized way such that the time complexity or space complexity is minimum.

Steps in Map Reduce

- The map takes data in the form of pairs and returns a list of <key, value> pairs. The keys will not be unique in this case.
- Using the output of Map, sort and shuffle are applied by the Hadoop architecture. This sort and shuffle acts on these list of <key, value> pairs and sends out unique keys and a list of values associated with this unique key <key, list(values)>.
- An output of sort and shuffle sent to the reducer phase. The reducer performs a defined function on a list of values for unique keys, and Final output <key, value> will be stored/displayed.

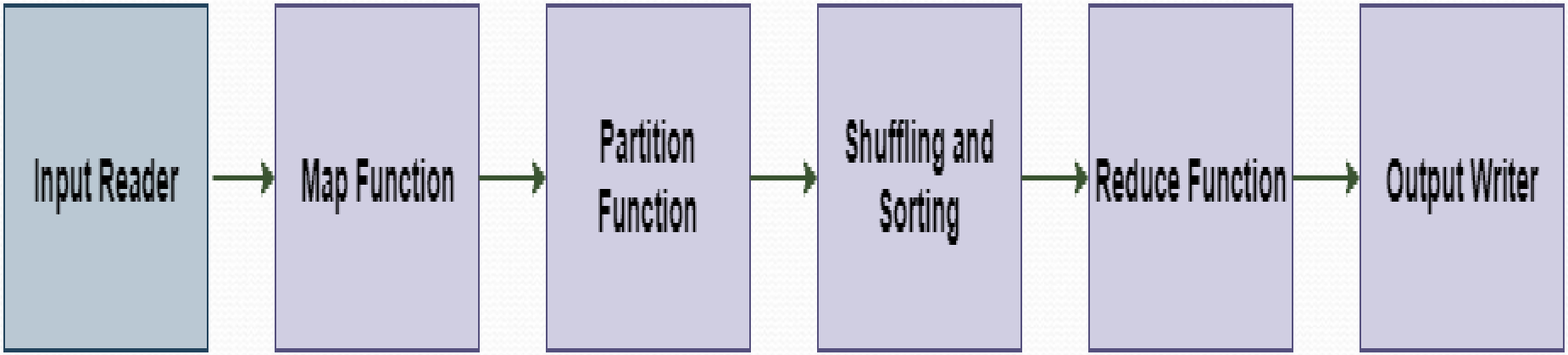

Input
Map
Shuffle
Reduce Input
Reduce Output
Output

Sort and Shuffle

- The sort and shuffle occur on the output of Mapper and before the reducer.
- When the Mapper task is complete, the results are sorted by key, partitioned if there are multiple reducers, and then written to disk.
- Using the input from each Mapper <k2,v2>, we collect all the values for each unique key k2.
- This output from the shuffle phase in the form of <k2, list(v2)> is sent as input to reducer phase.

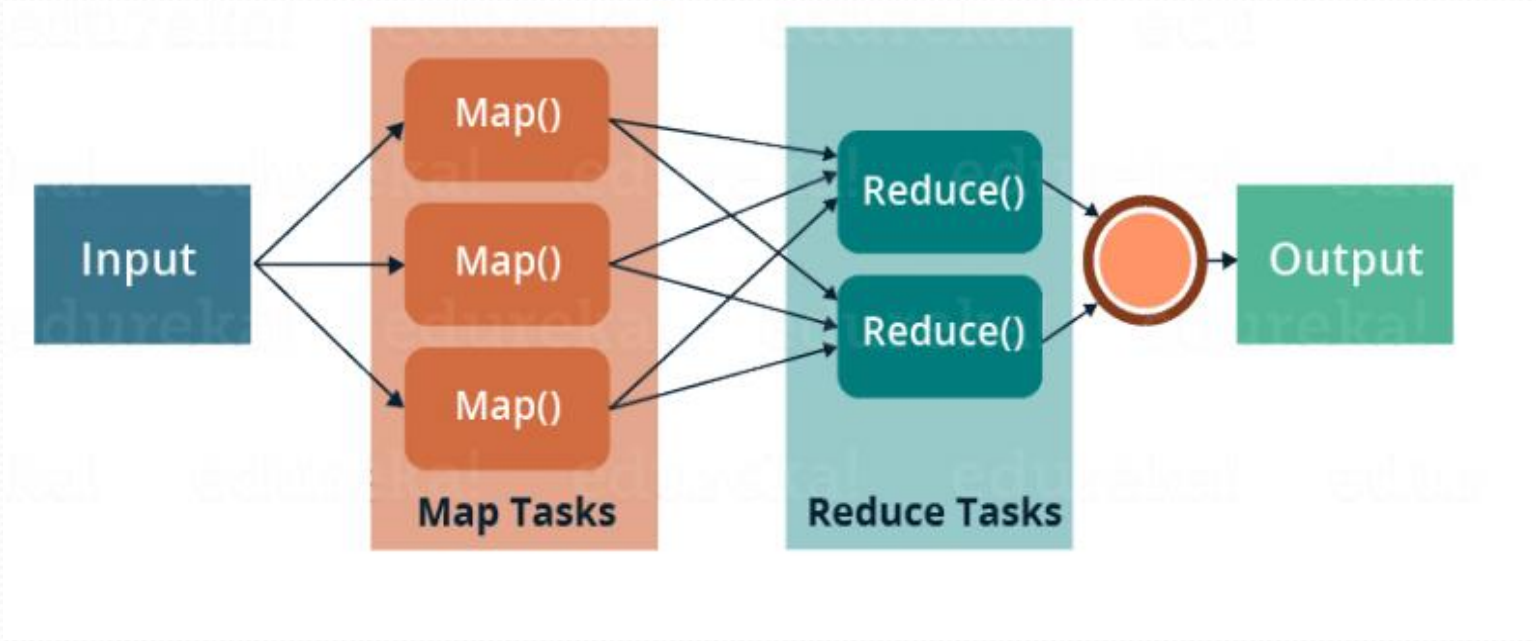
Data Flow In MapReduce

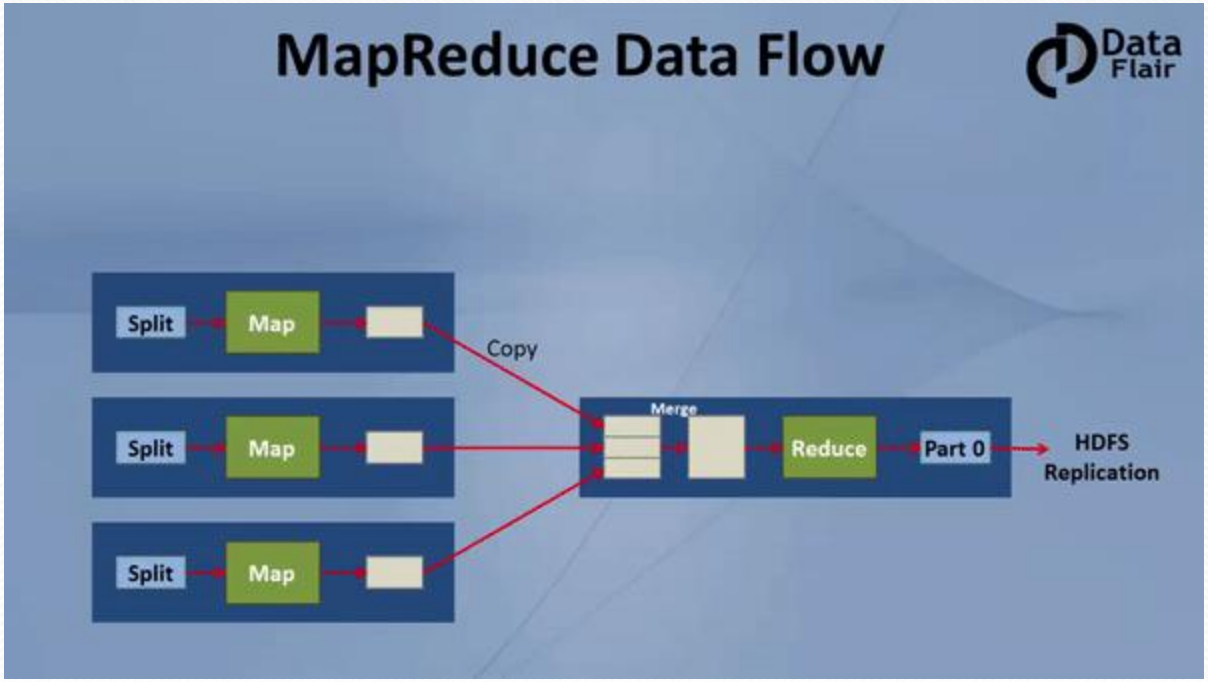
MapReduce is used to compute the huge amount of data . To handle the upcoming data in a parallel and distributed form, the data has to flow from various phases.



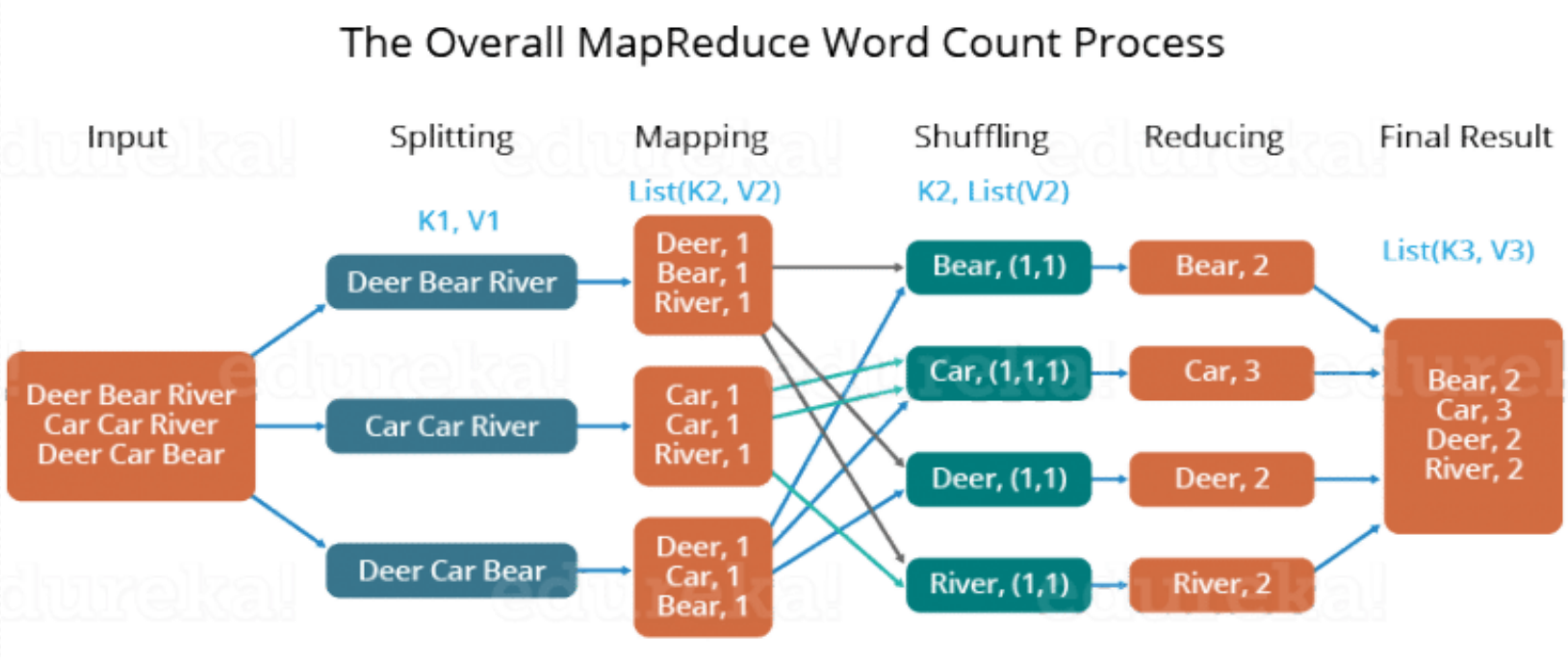
- Partition function
  - The partition function assigns the output of each Map function to the appropriate reducer. The available key and value provide this function. It returns the index of reducers.
- Shuffling and Sorting
  - The data are shuffled between/within nodes so that it moves out from the map and get ready to process for reduce function. Sometimes, the shuffling of data can take much computation time.
  - The sorting operation is performed on input data for Reduce function. Here, the data is compared using comparison function and arranged in a sorted form.

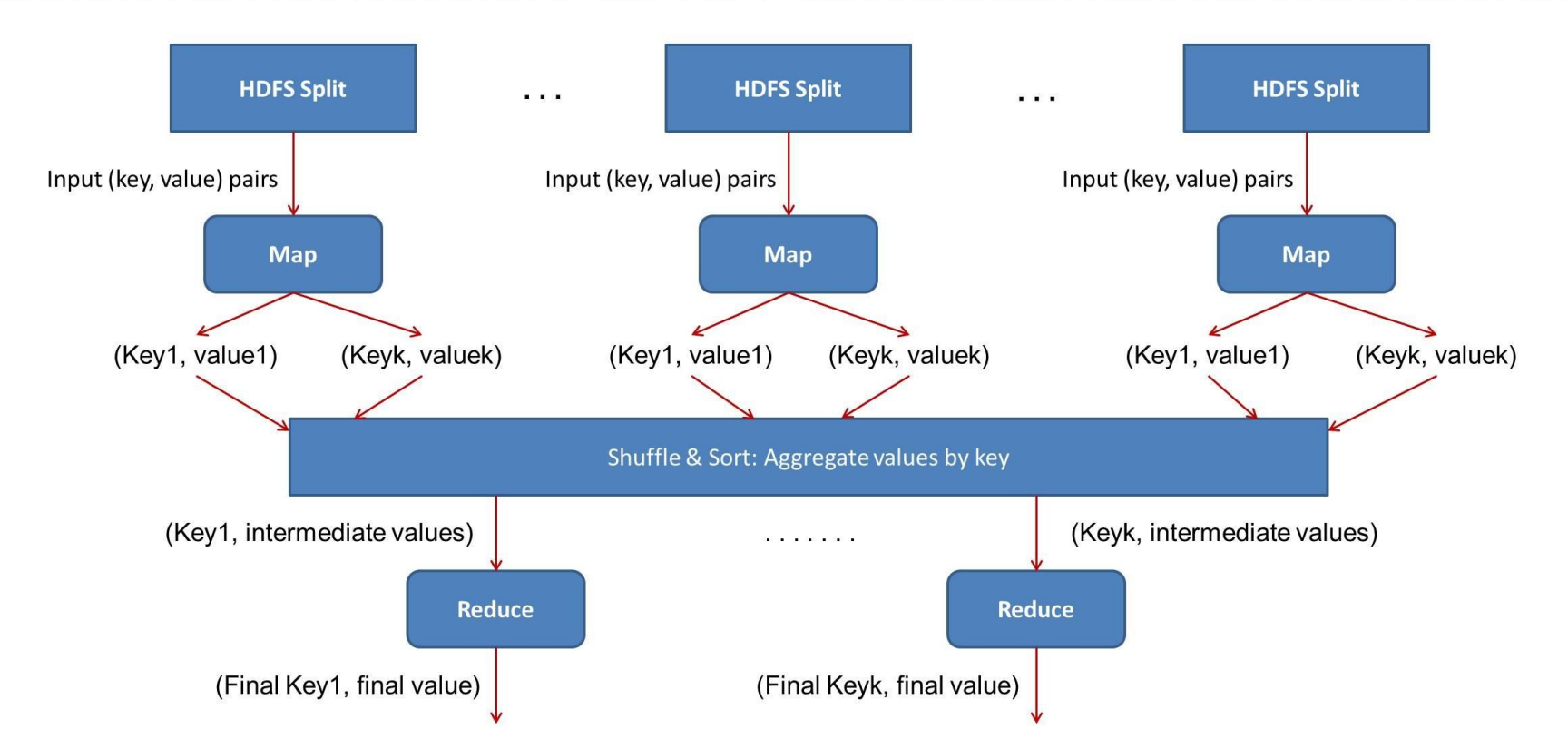
- Reduce function
  - The Reduce function is assigned to each unique key. These keys are already arranged in sorted order.
  - The values associated with the keys can iterate the Reduce and generate the corresponding output.
- Output writer
  - Once the data flow from all the above phases, Output writer executes.
  - The role of Output writer is to write the Reduce output to the stable storage.


Input
Map()
Map()
Map()
Map Tasks
Reduce()
Reduce()
Reduce Tasks
Output


MapReduce Data Flow
Data Flair
Split
Map
Split
Map
Split
Map
Copy
Merge
Reduce
Part 0
HDFS Replication

Example:Find the Word Count




HDFS Split
HDFS Split
HDFS Split
Input (key, value) pairs
Input (key, value) pairs
Input (key, value) pairs
Map
Map
Map
(Key1, value1)
(Keyk, valuek)
(Key1, value1)
(Keyk, valuek)
(Key1, value1)
(Keyk, valuek)
Shuffle & Sort: Aggregate values by key
(Key1, intermediate values)
(Keyk, intermediate values)
Reduce
Reduce
(Final Key1, final value)
(Final Keyk, final value)