Map-reduce

- Map-reduce is a generic multi-phase data aggregation modality for processingquantities of data.
- MongoDB provides map-reduce with the ***mapReduce*** database command.
- MongoDB applies the map phase to each input document (i.e. the documents in the collection that match the query condition). The map function emits keyvalue pairs. For those keys that have multiple values,
- MongoDB applies the reduce phase, which collects and condenses the aggregated data. MongoDB then stores the results in a collection. Optionally, the output of the reduce function may pass through a finalize function to further condense or process the results of the aggregation
- All map-reduce functions in MongoDB are JavaScript and run within the mongod process. Map-reduce operations take the documents of a single collection as the input and can perform any arbitrary sorting and limiting before beginning the map stage.
- mapReduce can return the results of a map-reduce operation as a document, or may write the results to collections.

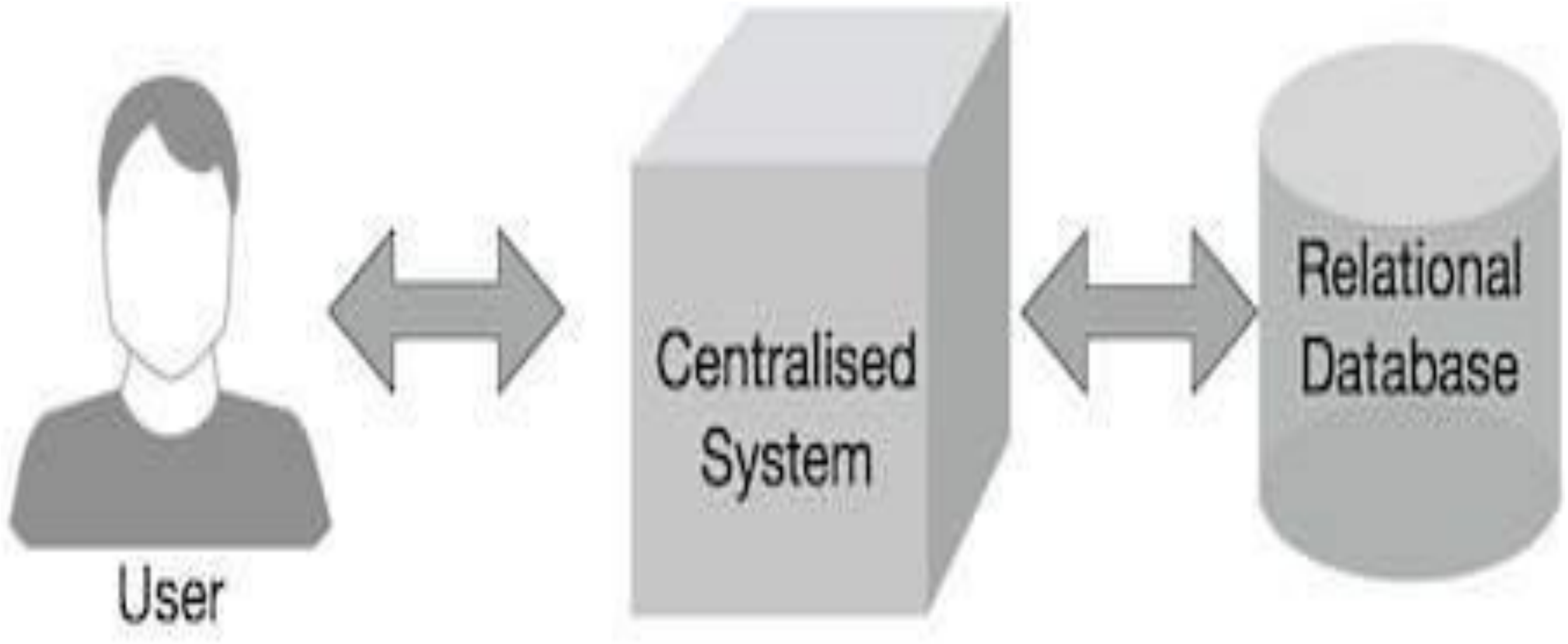
- As per the MongoDB documentation, **Map-reduce** is a data processing paradigm for condensing large volumes of data into useful aggregated results. MongoDB uses **mapReduce** command for map-reduce operations. MapReduce is generally used for processing large data sets.

MapReduce Command:

```
>db.collection.mapReduce(
 function() {emit(key,value);}, //map function
 function(key,values) {return reduceFunction},  //reduce
 function
 {
   out: collection, query: document,
   sort: document, limit: number
 }
)
```

- The map-reduce function first queries the collection, then maps the result documents to emit key- value pairs which is then reduced based on the keys that have multiple values.
- In the above syntax:
  - **map** is a javascript function that maps a value with a key and emits a key-valur pair
  - **reduce** is a javscript function that reduces or groups all the documents having the same key
  - **out** specifies the location of the map-reduce query result
  - **query** specifies the optional selection criteria for selecting documents
  - **sort** specifies the optional sort criteria
  - **limit** specifies the optional maximum number of documents to be returned

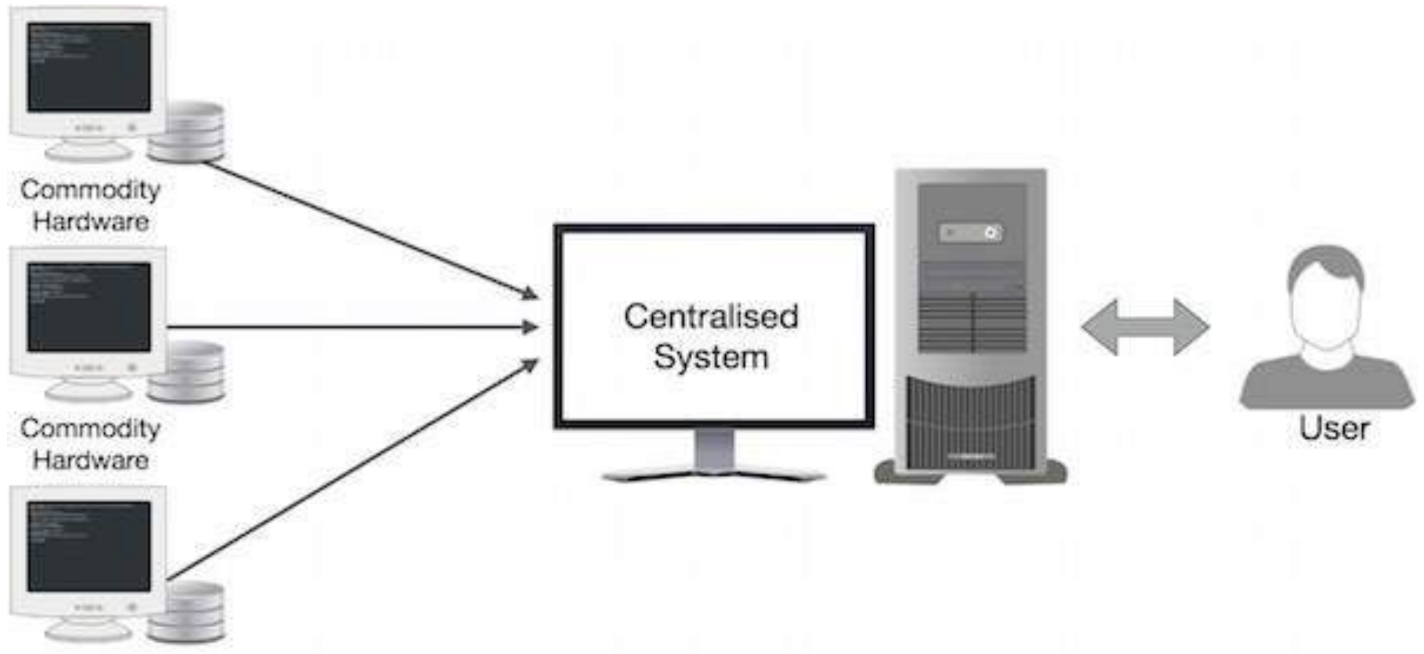
Why MapReduce?



Why MapReduce?

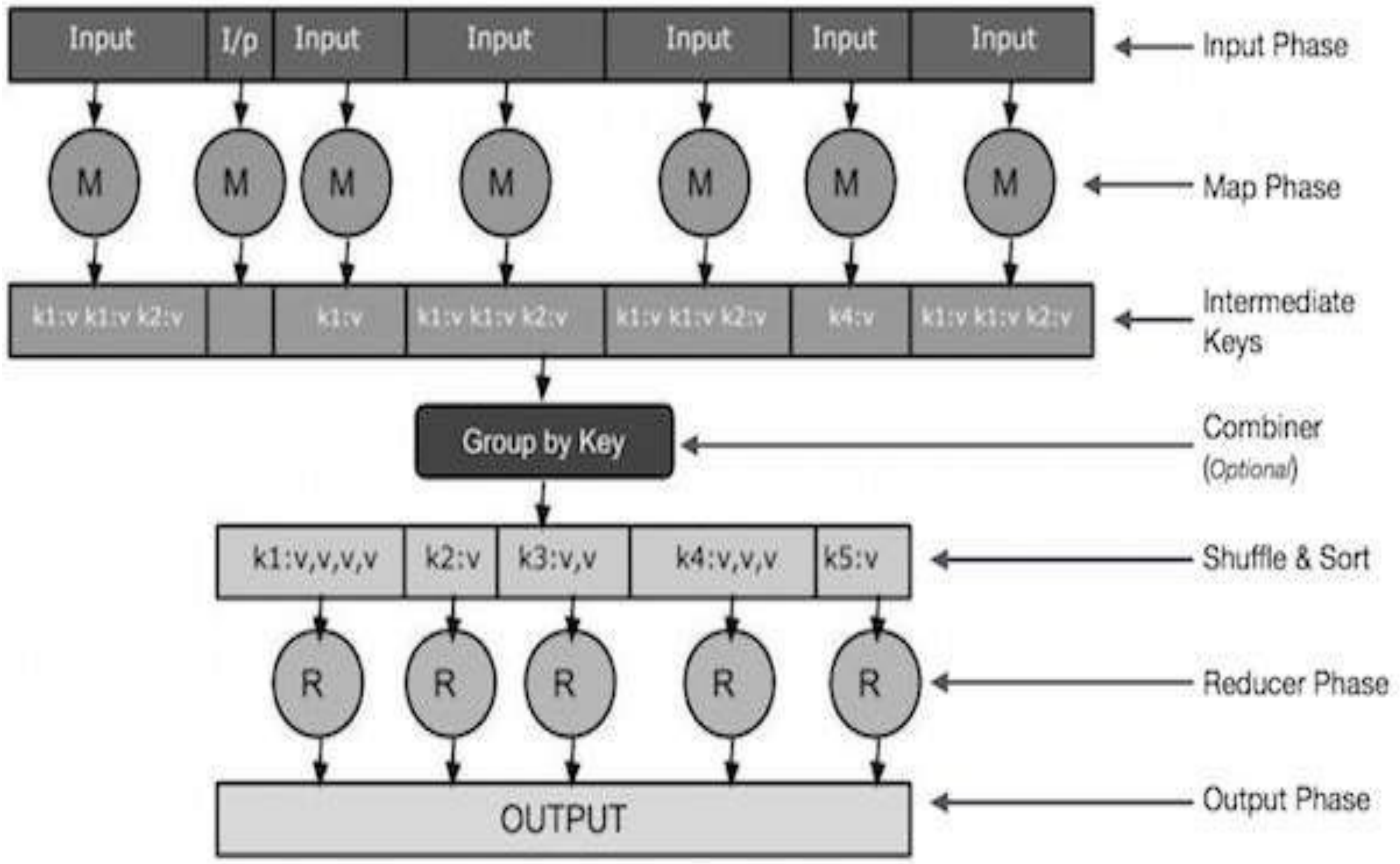
- Traditional Enterprise Systems normally have a centralized server to store and process data. The following illustration depicts a schematic view of a traditional enterprise system. Traditional model is certainly not suitable to process huge volumes of scalable data and cannot be accommodated by standard database servers. Moreover, the centralized system creates too much of a bottleneck while processing multiple files simultaneously.

- Google solved this bottleneck issue using an algorithm called MapReduce. MapReduce divides a task into small parts and assigns them to many computers. Later, the results are collected at one place and integrated to form the result dataset.



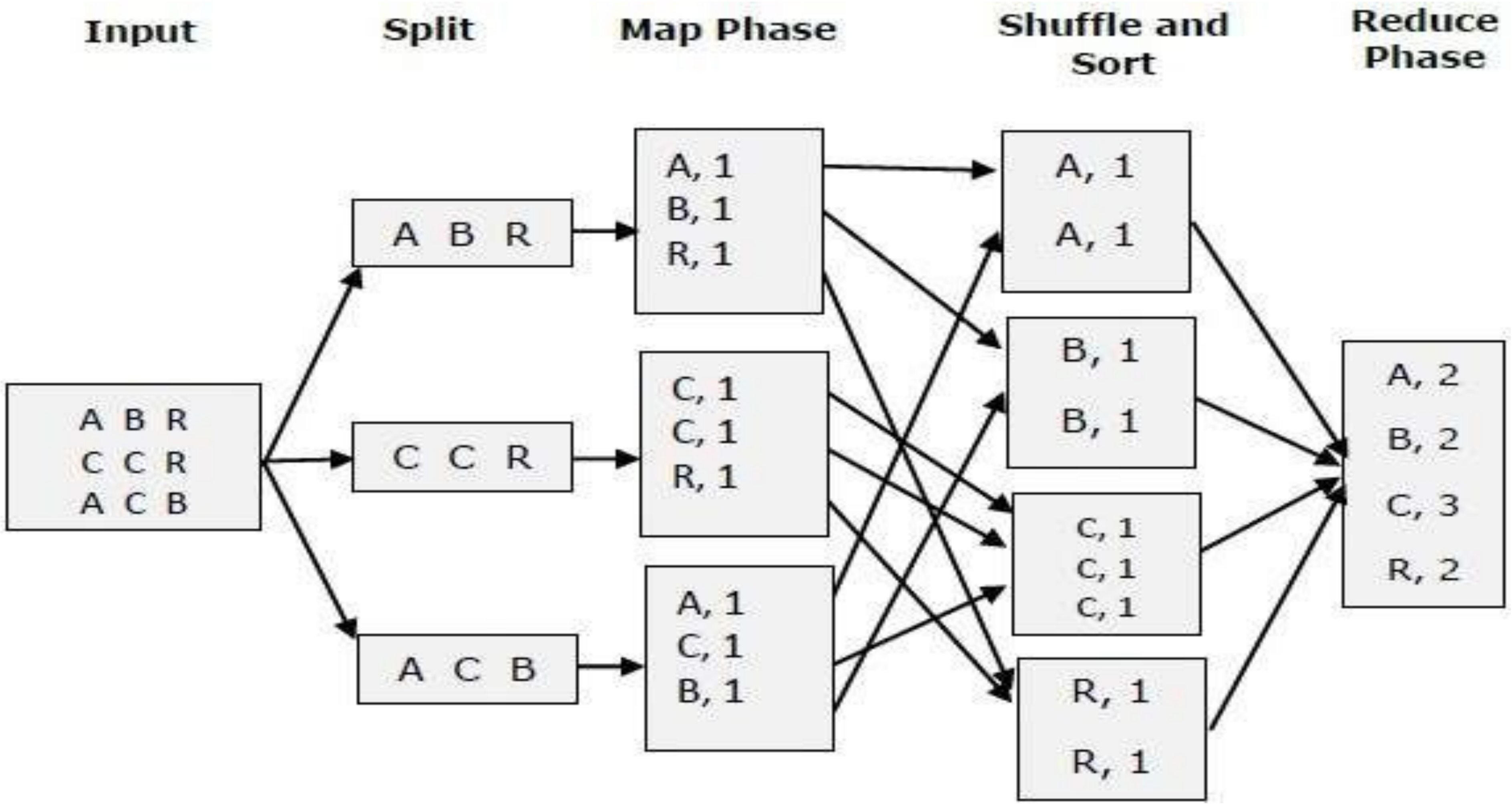
How MapReduce Works?

- The MapReduce algorithm contains two important tasks, namely Map and Reduce.
- The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).
- The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.
- The reduce task is always performed after the map job.
- Let us now take a close look at each of the phases and try to understand their significance.


Input
I/p
Input
Input
Input
Input
Input
Input Phase
M
Map Phase
k1:v k1:v k2:v
k1:v
k1:v k1:v k2:v
k1:v k1:v k2:v
k4:v
k1:v k1:v k2:v
Intermediate Keys
Group by Key
Combiner (Optional)
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
Shuffle & Sort
R
Reducer Phase
OUTPUT
Output Phase

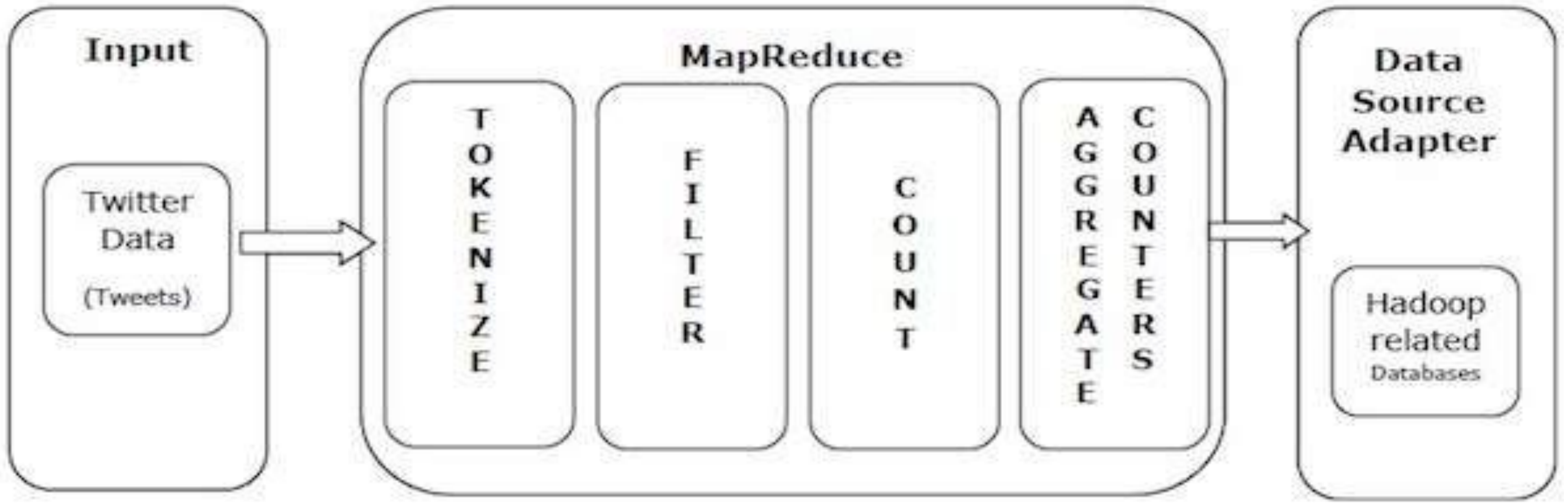
- **Input Phase** − Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.
- **Map** − Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.
- **Intermediate Keys** − They key-value pairs generated by the mapper are known as intermediate keys.
- **Combiner** − A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.
- **Shuffle and Sort** − The Reducer task starts with the Shuffle and Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.
- **Reducer** − The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.
- **Output Phase** − In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

Let us try to understand the two tasks Map &f Reduce with the help of a small diagram −



MapReduce-Example

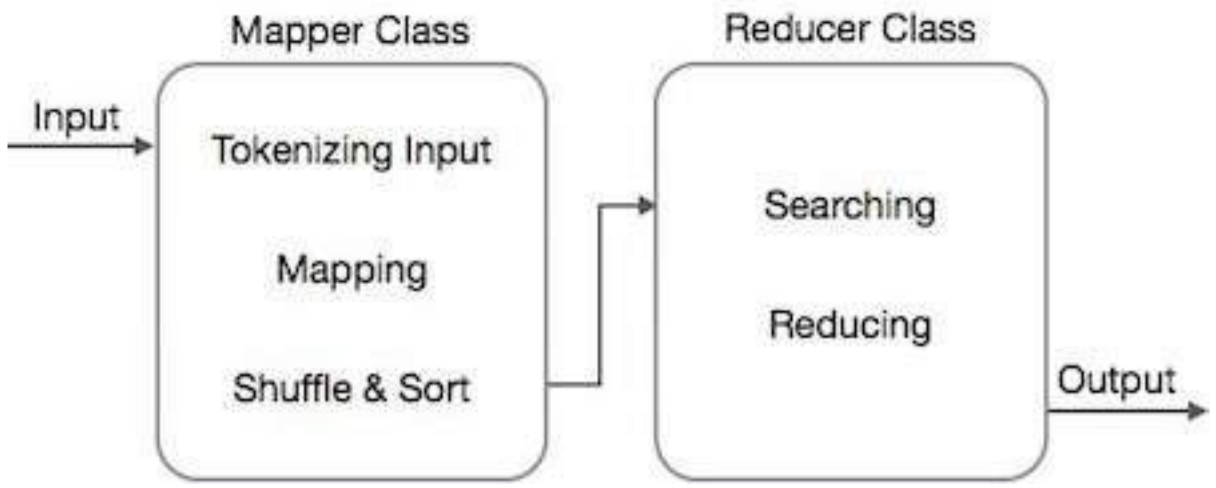
Let us take a real-world example to comprehend the power of MapReduce. Twitter receives around 500 million tweets per day, which is nearly 3000 tweets per second. The following illustration shows how Tweeter manages its tweets with the help of MapReduce.



- As shown in the illustration, the MapReduce algorithm performs the following actions −
- **Tokenize** − Tokenizes the tweets into maps of tokens and writes them as key-value pairs.
- **Filter** − Filters unwanted words from the maps of tokens and writes the filtered maps as key-value pairs.
- **Count** − Generates a token counter per word.
- **Aggregate Counters** − Prepares an aggregate of similar counter values into small manageable units.

MapReduce - Algorithm

- The MapReduce algorithm contains two important tasks, namely Map and Reduce.
- The map task is done by means of Mapper Class
- The reduce task is done by means of Reducer Class.
- Mapper class takes the input, tokenizes it, maps and sorts it. The output of Mapper class is used as input by Reducer class, which in turn searches matching pairs and reduces them.



- MapReduce implements various mathematical algorithms to divide a task into small parts and assign them to multiple systems. In technical terms, MapReduce algorithm helps in sending the Map & Reduce tasks to appropriate servers in a cluster.
- These mathematical algorithms may include the following −
- Sorting
- Searching
- Indexing