Parallel & Distributed Computing (WQD7008)

Week 10

Large Scale Data Analysis with MapReduce Presentation Model

2019/2020 Semester 1

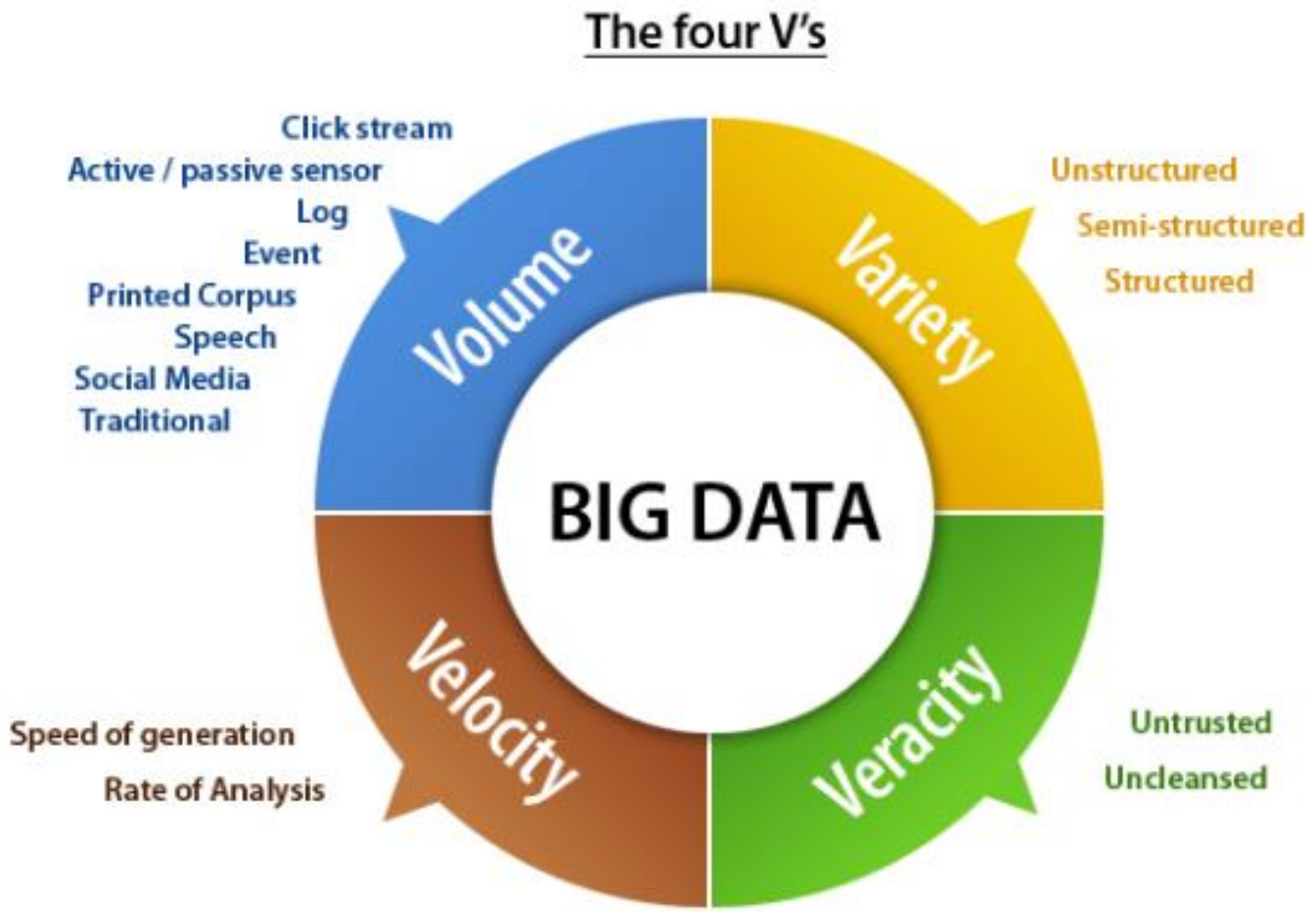
Dr. Hamid Tahaei

Introduction

What is Big Data?

- Big Data is a collection of large datasets that cannot be processed using traditional computing techniques.
- For example, the volume of data from Facebook or YouTube, to collect and manage on a daily basis, can fall under the category of Big Data.
- However, Big Data is not only about scale and volume, it also involves one or more of the following aspects – Velocity, Variety, Volume, and Complexity.
- Big data is a term that describes the large volume of data – both structured and unstructured – that inundates a business on a day-to-day basis.
- But it's not the amount of data that's important. It's what organizations do with the data that matters.
- Big data can be analyzed for insights that lead to better decisions and strategic business moves.

Introduction



Introduction

What is Big Data?

- **Volume.** Organizations collect data from a variety of sources, including business transactions, social media and information from sensor or machine-to-machine data. In the past, storing it would've been a problem – but new technologies (such as Hadoop) have eased the burden.
- **Velocity.** Data streams in at an unprecedented speed and must be dealt with in a timely manner. RFID tags, sensors and smart metering are driving the need to deal with torrents of data in near-real time.
- **Variety.** Data comes in all types of formats – from structured, numeric data in traditional databases to unstructured text documents, email, video, audio, stock ticker data and financial transactions.

- **Variability.** In addition to the increasing velocities and varieties of data, data flows can be highly inconsistent with periodic peaks. Is something trending in social media? Daily, seasonal and event-triggered peak data loads can be challenging to manage. Even more so with unstructured data.
- **Complexity.** Today's data comes from multiple sources, which makes it difficult to link, match, cleanse and transform data across systems. However, it's necessary to connect and correlate relationships, hierarchies and multiple data linkages or your data can quickly spiral out of control.

Introduction

Benefits of Big Data and Data Analytics

- Big data makes it possible for you to gain more complete answers because you have more information.
- More complete answers mean more confidence in the data—which means a completely different approach to tackling problems.
- Big data can help you address a range of business activities, from customer experience to analytics.

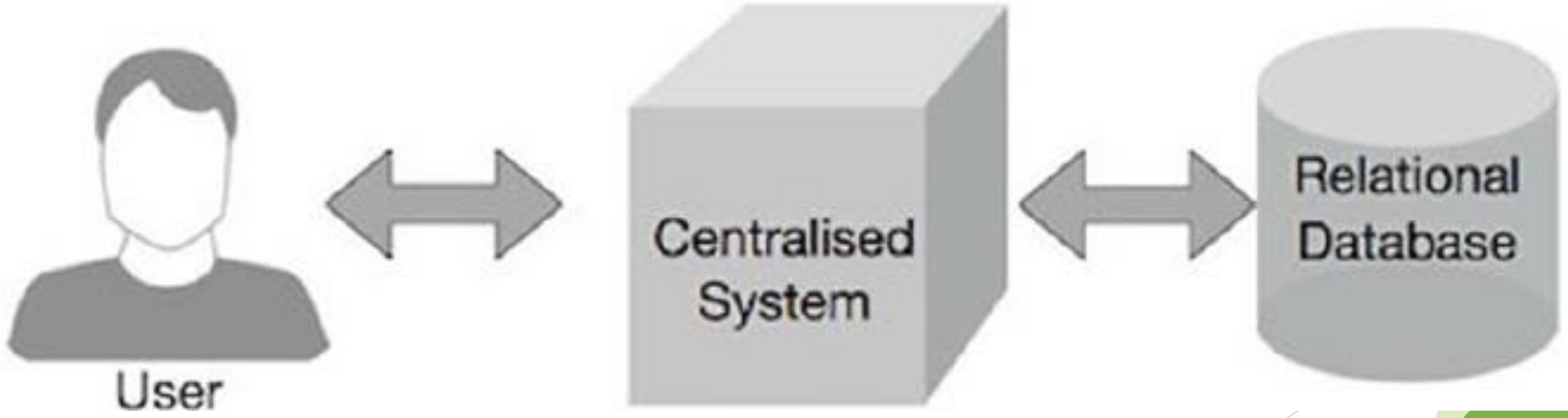
Some use cases:

Product Development, Predictive Maintenance, Customer Experience, Fraud and Compliance, Machine Learning, Operational Efficiency, Drive Innovation

Introduction

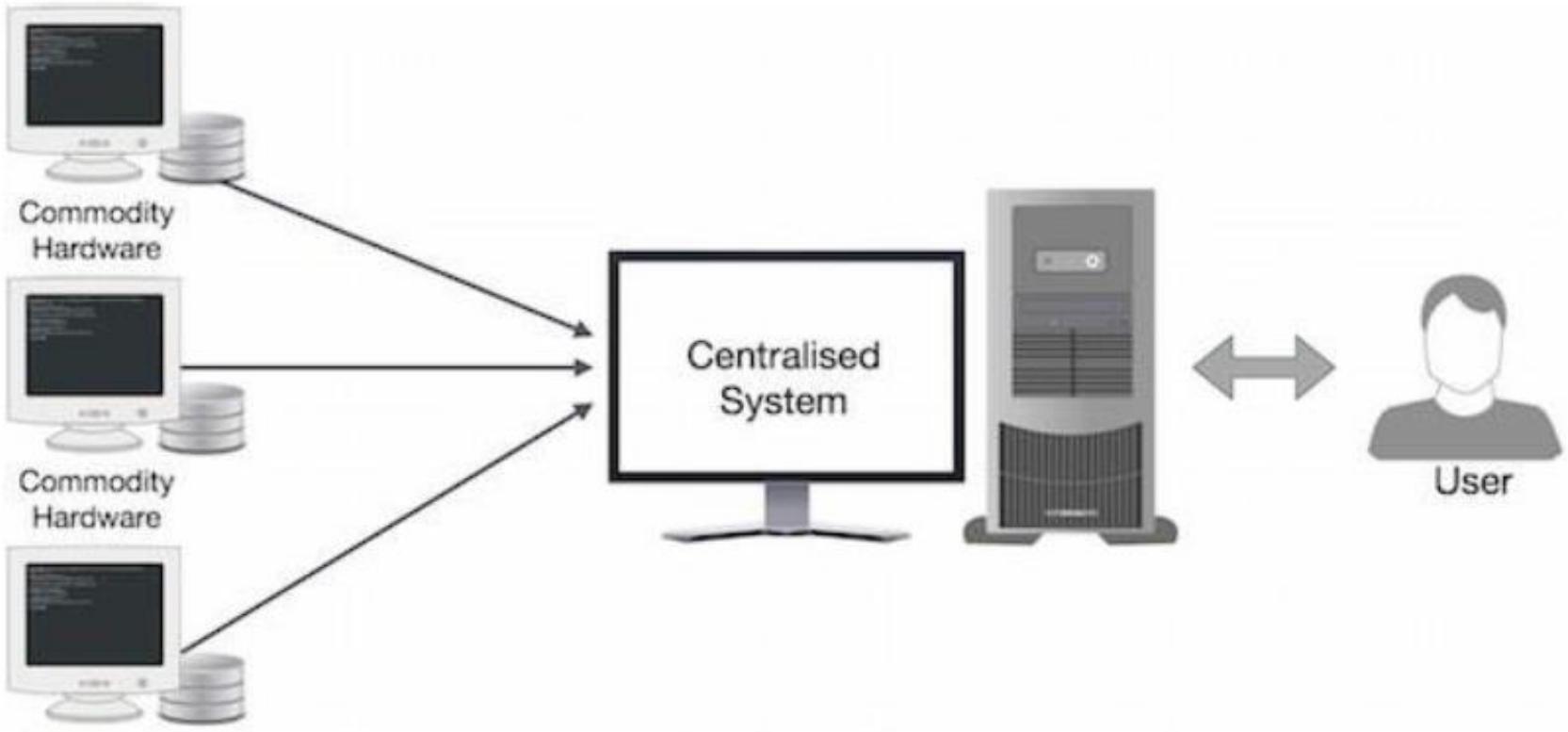
What is MapReduce and Why MapReduce?

- Traditional Enterprise Systems normally have a centralized server to store and process data.
- 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.



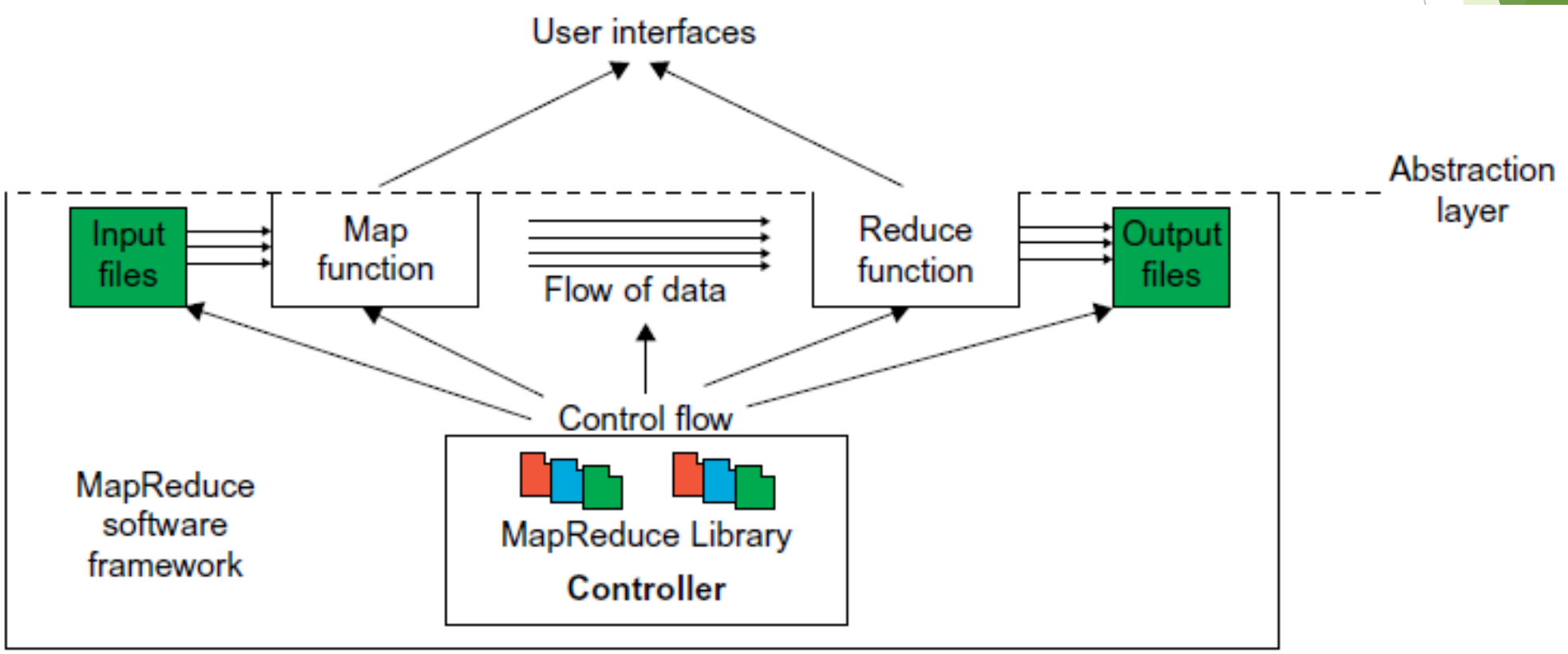
Introduction

- Google solved this bottleneck issue using an algorithm called MapReduce.
- MapReduce is a programming model for writing applications that can process Big Data in parallel on multiple nodes. MapReduce provides analytical capabilities for analyzing huge volumes of complex data.
- 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.



Introduction

- MapReduce is a software framework which supports parallel and distributed computing on large data sets. This software framework abstracts the data flow of running a parallel program on a distributed computing system by providing users with two interfaces in the form of two functions: Map and Reduce. Users can override these two functions to interact with and manipulate the data flow of running their programs.

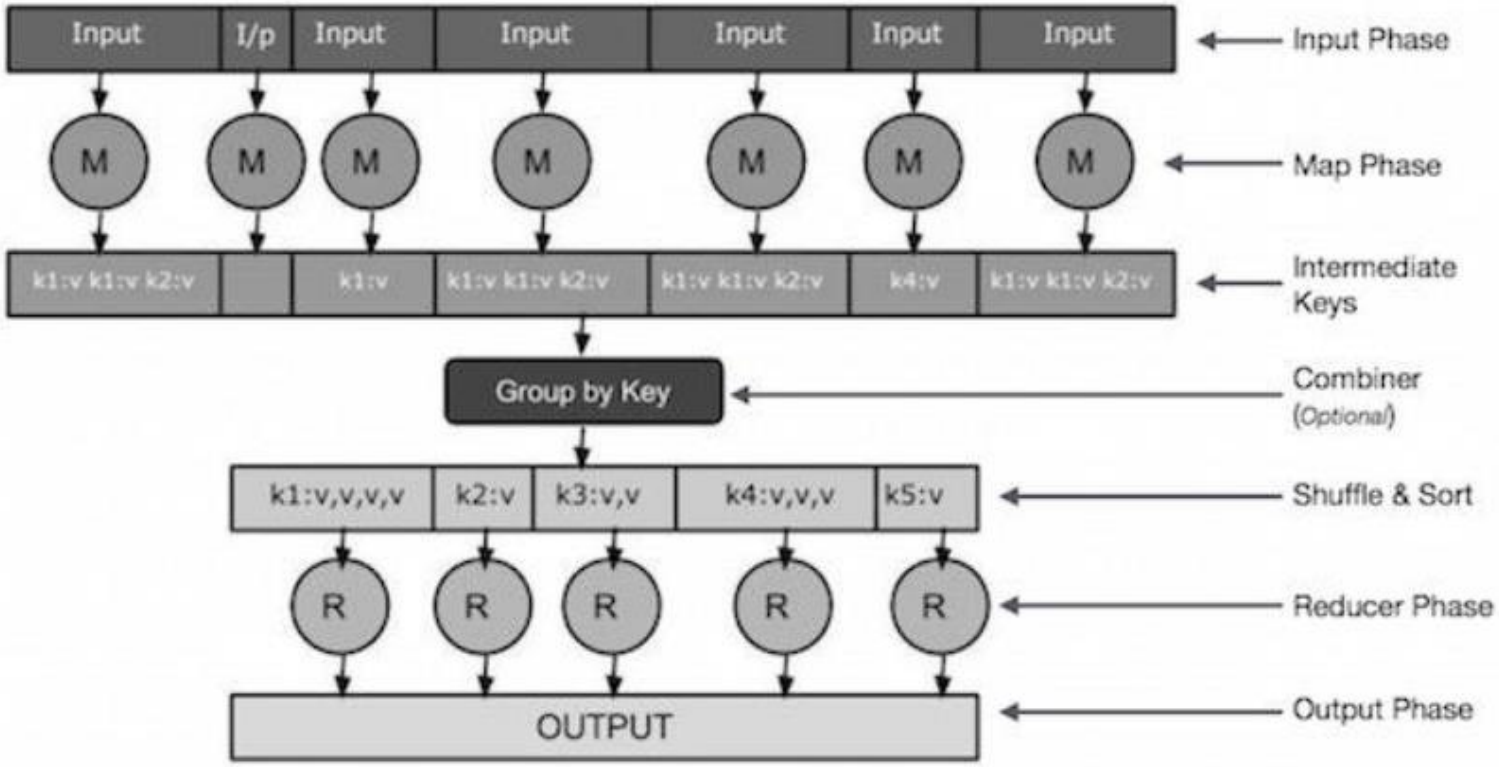


MapReduce workflow

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.



MapReduce Workflow (Control Flow)

How MapReduce Works?

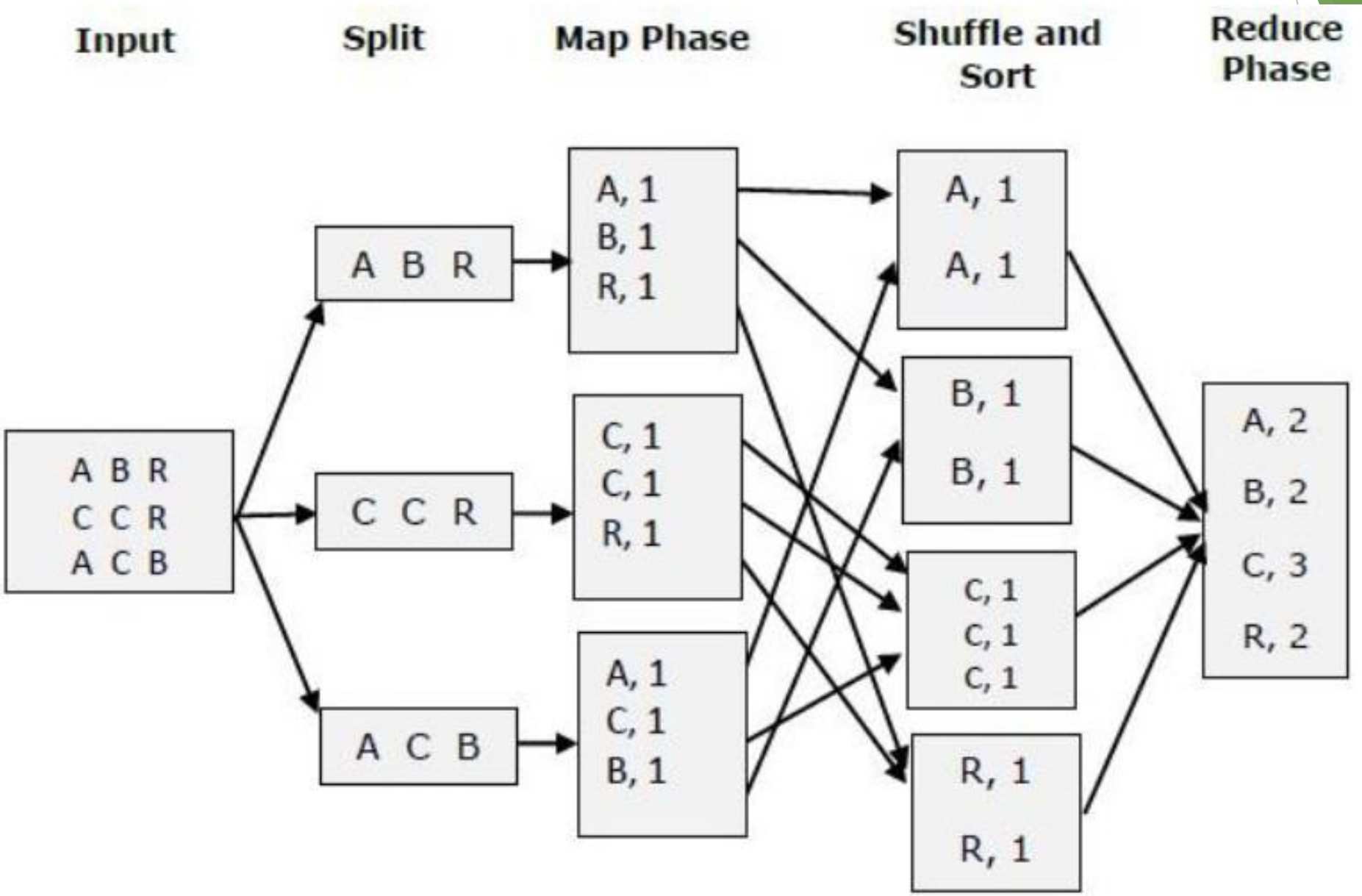
- **Input Phase** – There is a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.
- **Map** – Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.
- **Intermediate Keys** – The 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.

MapReduce workflow

How MapReduce Works?

- **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, there is an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

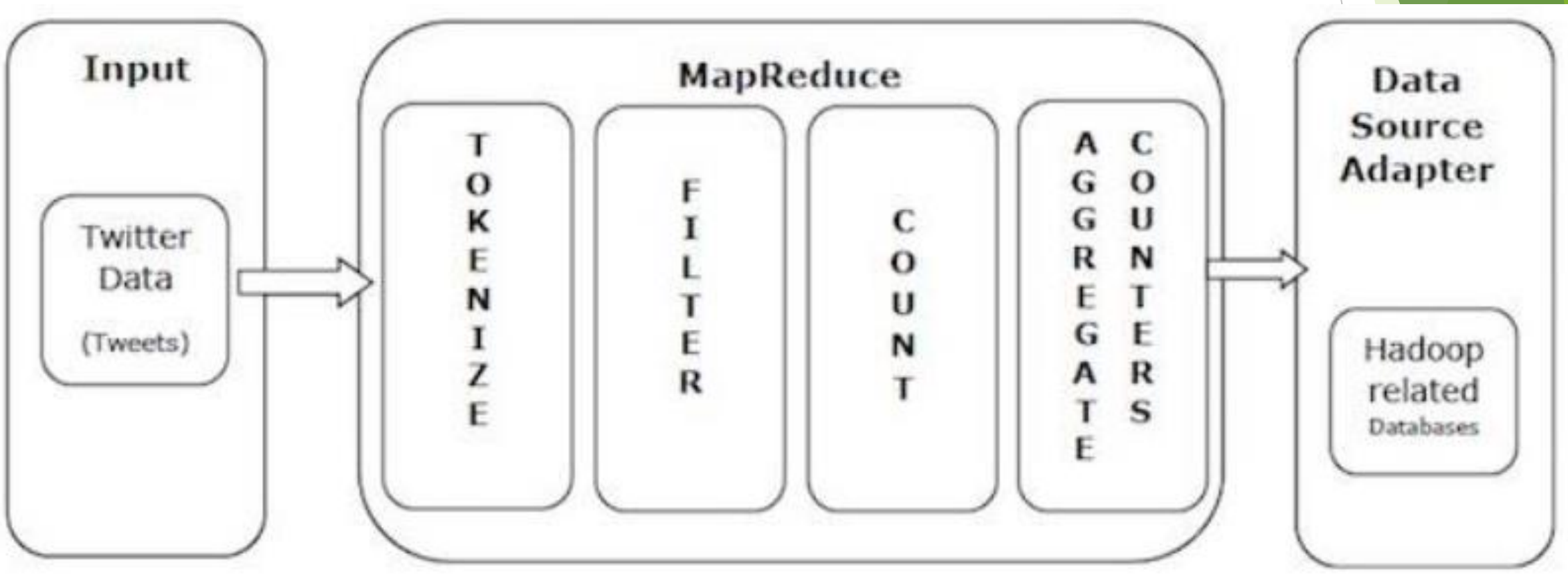
MapReduce workflow



MapReduce workflow

Example of a real-world

- 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.



MapReduce workflow

The MapReduce algorithm performs the following actions

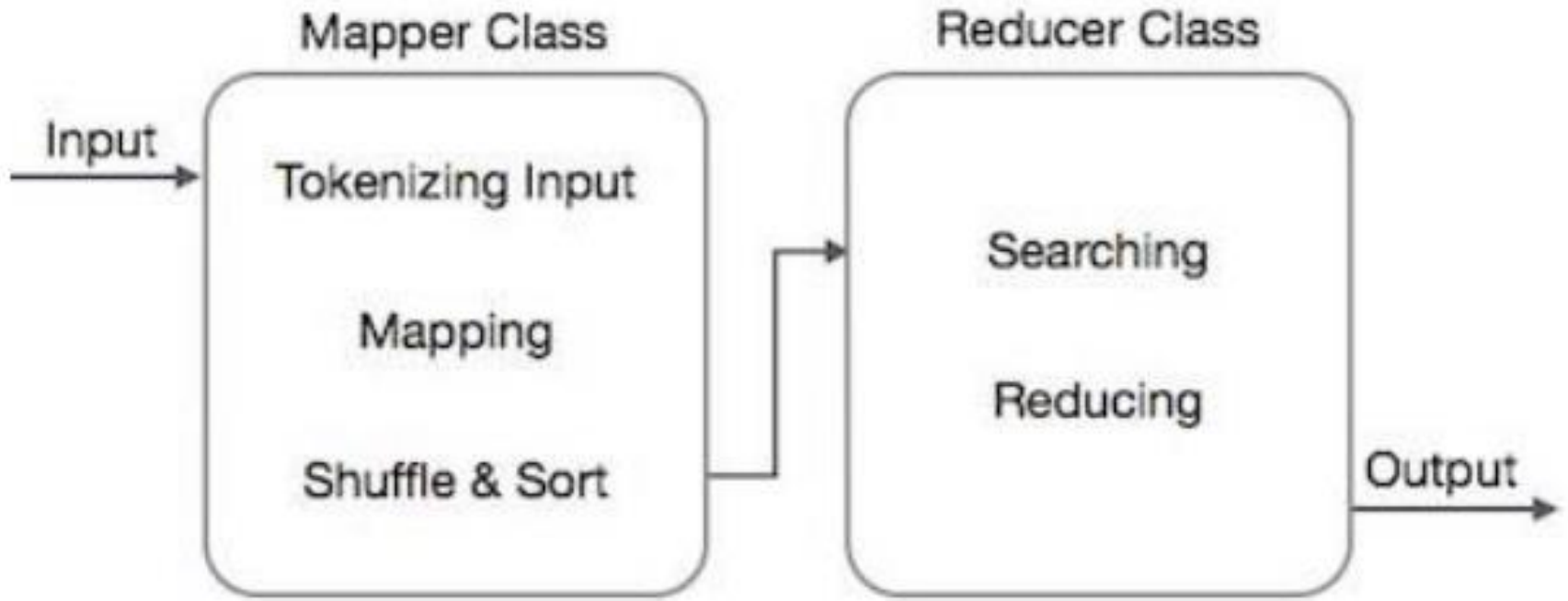
- **Tokenize** – Tokenizes the tweets into maps of tokens and writes them as key-value pairs.
- **Filter** – Filters unwanted words from the maps of tokens and writes the filtered maps as key-value pairs.
- **Count** – Generates a token counter per word.
- **Aggregate Counters** – Prepares an aggregate 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 – ALGORITHM

- 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, TF-IDF(Term Frequency – Inverse Document Frequency).

Sorting

- Sorting is one of the basic MapReduce algorithms to process and analyze data. MapReduce implements sorting algorithm to automatically sort the output key-value pairs from the mapper by their keys.
- Sorting methods are implemented in the mapper class itself.
- The set of intermediate key-value pairs for a given Reducer is automatically sorted by Hadoop to form key-values (K2, {V2, V2…}) before they are presented to the Reducer.

MAPREDUCE – ALGORITHM

Searching

- Searching plays an important role in MapReduce algorithm. It helps in the combiner phase (optional) and in the Reducer phase.

Example - How MapReduce employs Searching algorithm to find out the details of the employee who draws the highest salary in a given employee dataset.

name, salary
satish, 26000
Krishna, 25000
Satishk, 15000
Raju, 10000

name, salary
gopal, 50000
Krishna, 25000
Satishk, 15000
Raju, 10000

name, salary
satish, 26000
kiran, 45000
Satishk, 15000
Raju, 10000

name, salary
satish, 26000
Krishna, 25000
manisha, 45000
Raju, 10000

MAPREDUCE – ALGORITHM

Searching

- Lets assume there are employee data in four different files – A, B, C, and D. Let us also assume there are duplicate employee records in all four files because of importing the employee data from all database tables repeatedly.
- **The Map phase** processes each input file and provides the employee data in key-value pairs (<k, v> : <emp name, salary>).
- **The combiner phase** (searching technique) will accept the input from the Map phase as a key-value pair with employee name and salary. Using searching technique, the combiner will check all the employee salary to find the highest salaried employee in each file.

MAPREDUCE - ALGORITHM

```
<k: employee name, v: salary>
Max= the salary of an first employee. Treated as max salary

if(v(second employee).salary > Max){
   Max = v(salary);
}

else{
   Continue checking;
}
```

The expected result is as follows –

<satish, 26000>	<gopal, 50000>	<kiran, 45000>	<manisha, 45000>

MAPREDUCE - ALGORITHM

- **Reducer phase** − Form each file, you will find the highest salaried employee. To avoid redundancy, check all the <k, v> pairs and eliminate duplicate entries, if any. The same algorithm is used in between the four <k, v> pairs, which are coming from four input files. The final output should be as follows −

```
<gopal, 50000>
```

MAPREDUCE – ALGORITHM

Indexing

- Normally indexing is used to point to a particular data and its address. It performs batch indexing on the input files for a particular Mapper.
- The indexing technique that is normally used in MapReduce is known as **inverted index.** Search engines like Google and Bing use inverted indexing technique.

Example - The following text is the input for inverted indexing. Here T[0], T[1], and t[2] are the file names and their content are in double quotes.

```
T[0] = "it is what it is"
T[1] = "what is it"
T[2] = "it is a banana"
```

MAPREDUCE - ALGORITHM

Indexing

After applying the Indexing algorithm, we get the following output –

```
"a": {2}
"banana": {2}
"is": {0, 1, 2}
"it": {0, 1, 2}
"what": {0, 1}
```

Here "a": {2} implies the term "a" appears in the T[2] file. Similarly, "is": {0, 1, 2} implies the term "is" appears in the files T[0], T[1], and T[2].

MAPREDUCE – ALGORITHM

TF-IDF (Term Frequency – Inverse Document Frequency)

- TF-IDF is a text processing algorithm which is one of the common web analysis algorithms. The term 'frequency' refers to the number of times a term appears in a document.

Term Frequency (TF)

- It measures how frequently a particular term occurs in a document. It is calculated by the number of times a word appears in a document divided by the total number of words in that document.

```
TF(the) = (Number of times term the ‘the’ appears in a document) / (Total
number of terms in the document)
```

MAPREDUCE – ALGORITHM

Inverse Document Frequency (IDF)

- It measures the importance of a term. It is calculated by the number of documents in the text database divided by the number of documents where a specific term appears.
- While computing TF, all the terms are considered equally important. That means, TF counts the term frequency for normal words like “is”, “a”, “what”, etc. Thus we need to know the frequent terms while scaling up the rare ones, by computing the following

```
IDF(the) = log_e(Total number of documents / Number of documents with term
‘the’ in it).
```

MAPREDUCE – ALGORITHM

Example

- Consider a document containing 1000 words, wherein the word **hive** appears 50 times. The TF for **hive** is then (50 / 1000) = 0.05.
- Now, assume we have 10 million documents and the word **hive** appears in 1000 of these. Then, the IDF is calculated as log (10,000,000 / 1,000) = 4.
- The TF-IDF weight is the product of these quantities − 0.05 × 4 = 0.20.

MapReduce Actual Data and Control Flow

The main responsibility of the MapReduce framework is to efficiently run a user's program on a distributed computing system. Therefore, the MapReduce framework handles all partitioning, mapping, synchronization, communication, and scheduling details of such data flows.

- **Data partitioning** The MapReduce library splits the input data (files), already stored in GFS (Google File System), into M pieces that also correspond to the number of map tasks.
- **Computation partitioning** This is implicitly handled (in the MapReduce framework) by obliging users to write their programs in the form of the Map and Reduce functions. Therefore, the MapReduce library only generates copies of a user program (e.g., by a fork system call) containing the Map and the Reduce functions, distributes them, and starts them up on a number of available computation engines.
- **Determining the master and workers** The MapReduce architecture is based on a masterworker model. Therefore, one of the copies of the user program becomes the master and the rest become workers. The master picks idle workers, and assigns the map and reduce tasks to them. A map/reduce worker is typically a computation engine such as a cluster node to run map/ reduce tasks by executing Map/Reduce functions.

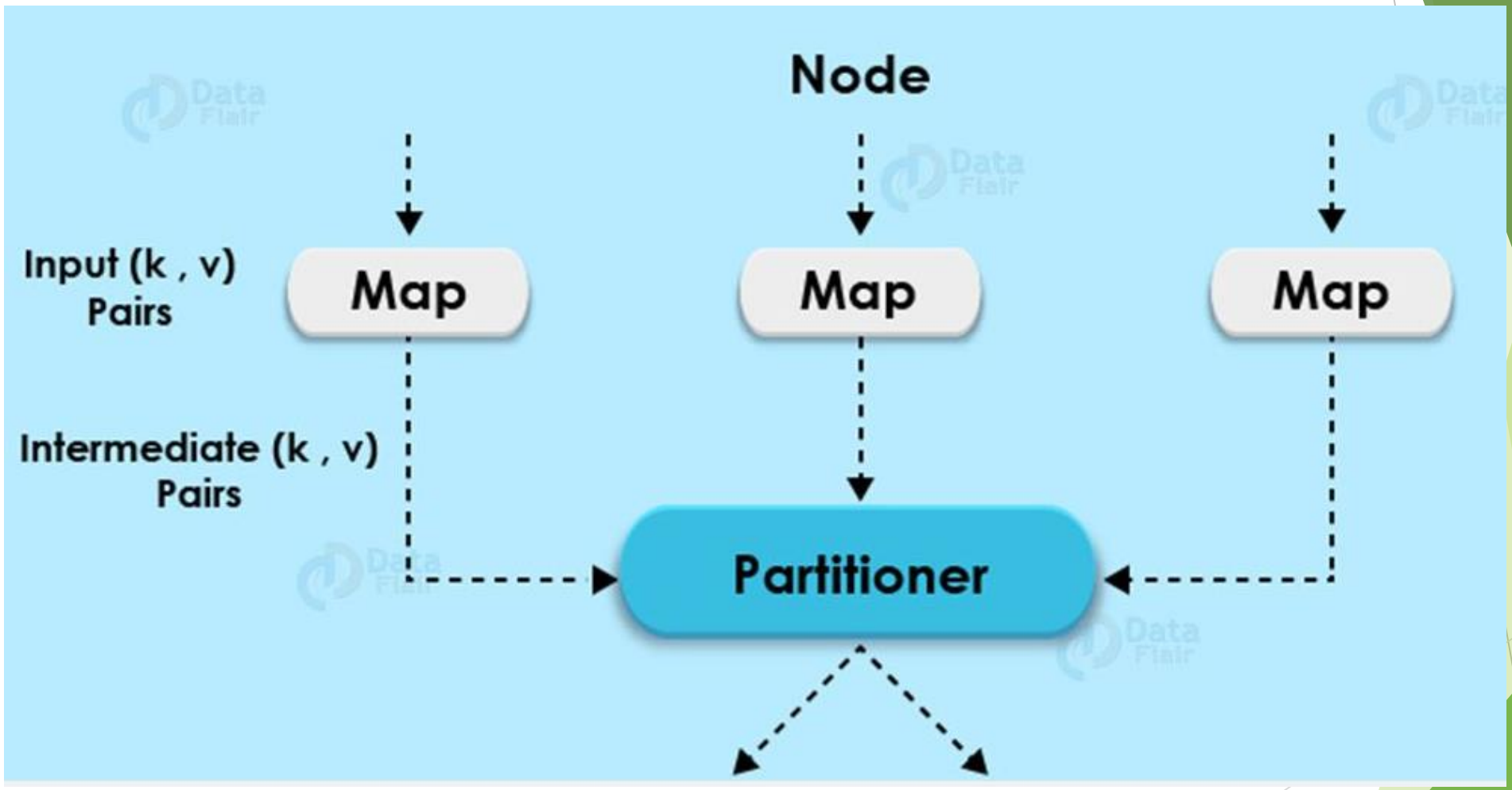
MapReduce Actual Data and Control Flow

- **Reading the input data (data distribution)** Each map worker reads its corresponding portion of the input data, namely the input data split, and sends it to its Map function. Although a map worker may run more than one Map function, which means it has been assigned more than one input data split, each worker is usually assigned one input split only.
- **Map function** Each Map function receives the input data split as a set of (key, value) pairs to process and produce the intermediated (key, value) pairs.
- **Combiner function** This is an optional local function within the map worker which applies to intermediate (key, value) pairs. The user can invoke the Combiner function inside the user program. The Combiner function runs the same code written by users for the Reduce function as its functionality is identical to it. The Combiner function merges the local data of each map worker before sending it over the network to effectively reduce its communication costs. The MapReduce framework sorts and groups the data before it is processed by the Reduce function. Similarly, the MapReduce framework will also sort and group the local data on each map worker if the user invokes the Combiner function.

MapReduce Actual Data and Control Flow

- **Partitioning function** The intermediate (key, value) pairs with identical keys are grouped together because all values inside each group should be processed by only one Reduce function to generate the final result. However, in real implementations, since there are M map and R reduce tasks, intermediate (key, value) pairs with the same key might be produced by different map tasks, although they should be grouped and processed together by one Reduce function only. Therefore, the intermediate (key, value) pairs produced by each map worker are partitioned into R regions, equal to the number of reduce tasks, by the Partitioning function to guarantee that all (key, value) pairs with identical keys are stored in the same region. As a result, since reduce worker i reads the data of region i of all map workers, all (key, value) pairs with the same key will be gathered by reduce worker i accordingly. To implement this technique, a Partitioning function could simply be a hash function (e.g., Hash(key) mod R) that forwards the data into particular regions. It is also worth noting that the locations of the buffered data in these R partitions are sent to the master for later forwarding of data to the reduce workers.

MapReduce Actual Data and Control Flow

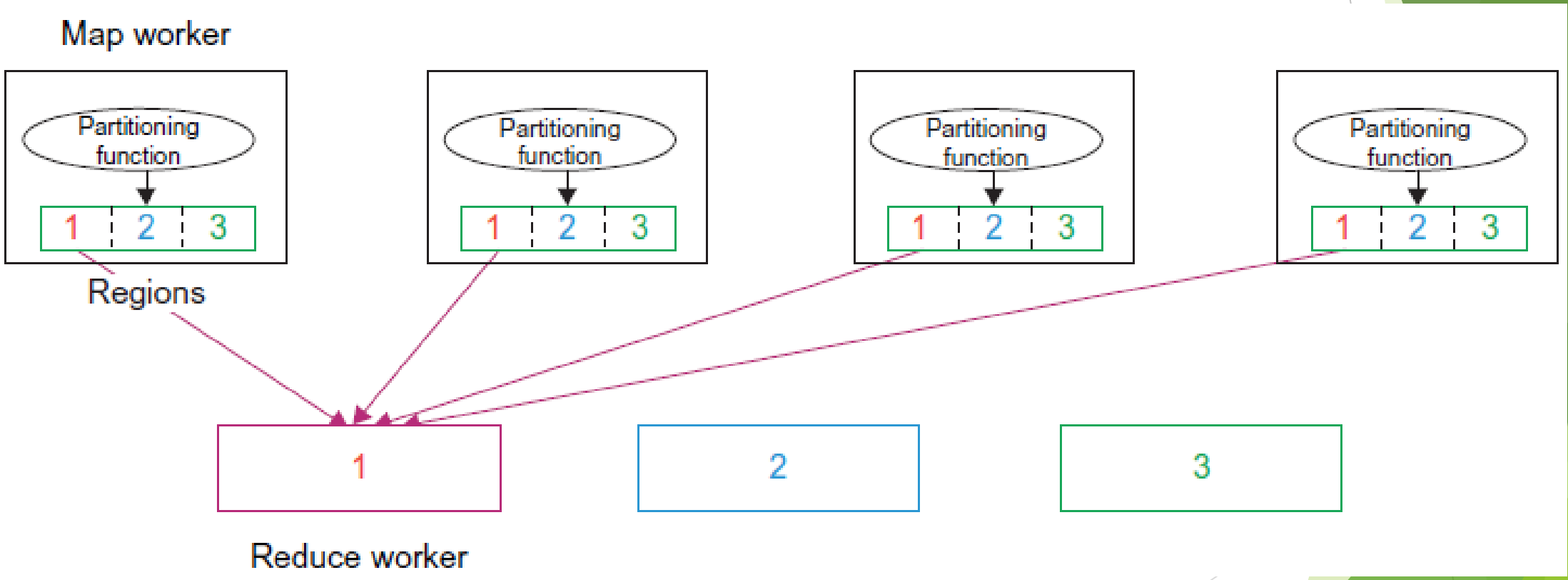


MapReduce Actual Data and Control Flow

- **Communication** Reduce worker i, already notified of the location of region i of all map workers, uses a remote procedure call to read the data from the respective region of all map workers. Since all reduce workers read the data from all map workers, all-to-all communication among all map and reduce workers, which incurs network congestion, occurs in the network. This issue is one of the major bottlenecks in increasing the performance of such systems.
- **Sorting and Grouping** When the process of reading the input data is finalized by a reduce worker, the data is initially buffered in the local disk of the reduce worker. Then the reduce worker groups intermediate (key, value) pairs by sorting the data based on their keys, followed by grouping all occurrences of identical keys. Note that the buffered data is sorted and grouped because the number of unique keys produced by a map worker may be more than R regions in which more than one key exists in each region of a map worker.
- **Reduce function** The reduce worker iterates over the grouped (key, value) pairs, and for each unique key, it sends the key and corresponding values to the Reduce function. Then this function processes its input data and stores the output results in predetermined files in the user's program.

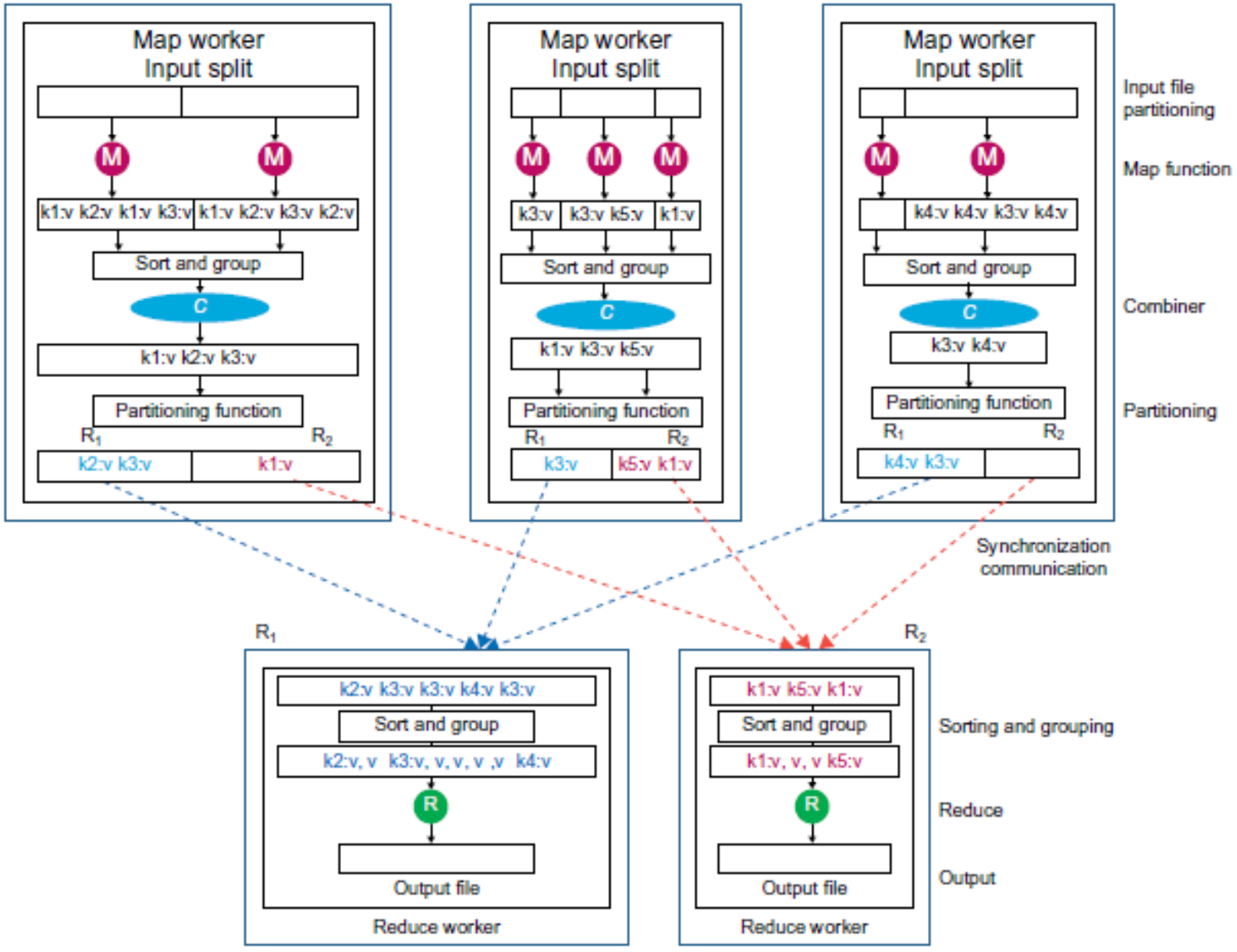
MapReduce Actual Data and Control Flow

- **Synchronization** MapReduce applies a simple synchronization policy to coordinate map workers with reduce workers, in which the communication between them starts when all map tasks finish.



Use of MapReduce partitioning function to link the Map and Reduce workers.

Data flow implementation of many functions in the Map workers and in the Reduce workers through multiple sequences of partitioning, combining, synchronization and communication, sorting and grouping, and reduce operations.



MAPREDUCE – ALGORITHM

Example – Word Count

Bus Car Train
Train Plane Car
Bus Bus Plane