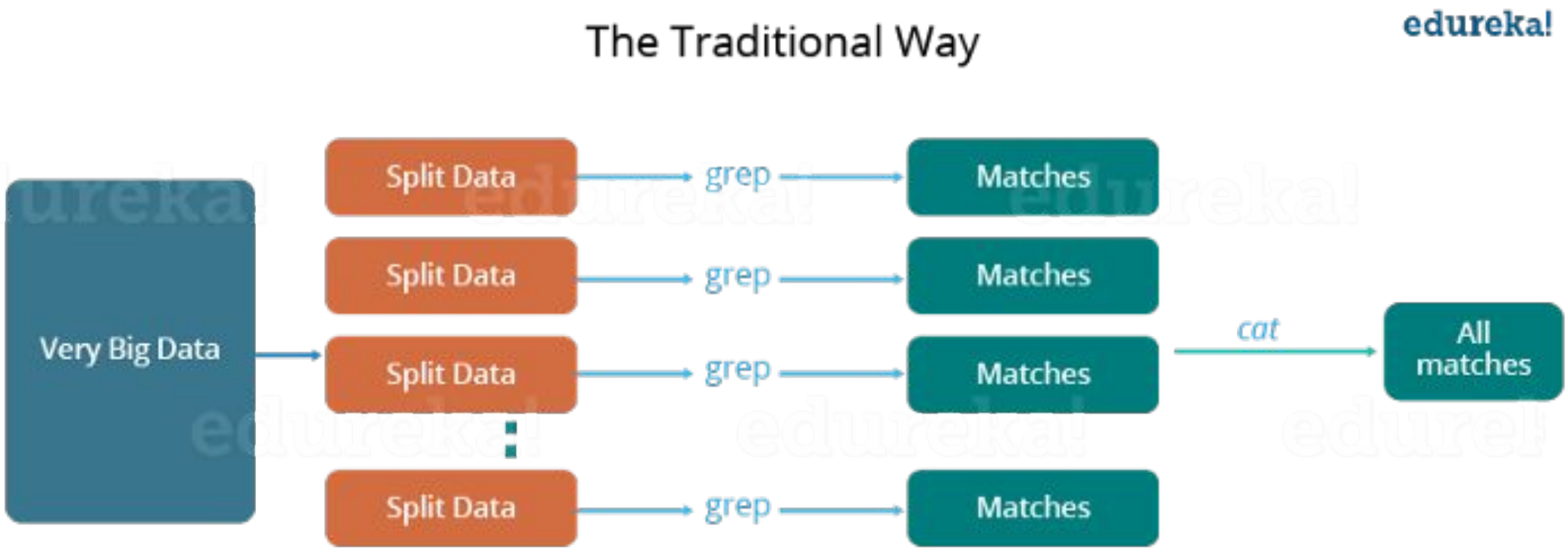
MapReduce Programming Model

Parallel Distributed computing



Design issues

- **Critical path problem:** It is the amount of time taken to finish the job without delaying the next milestone or actual completion date. So, if, any of the machines delay the job, the whole work gets delayed.

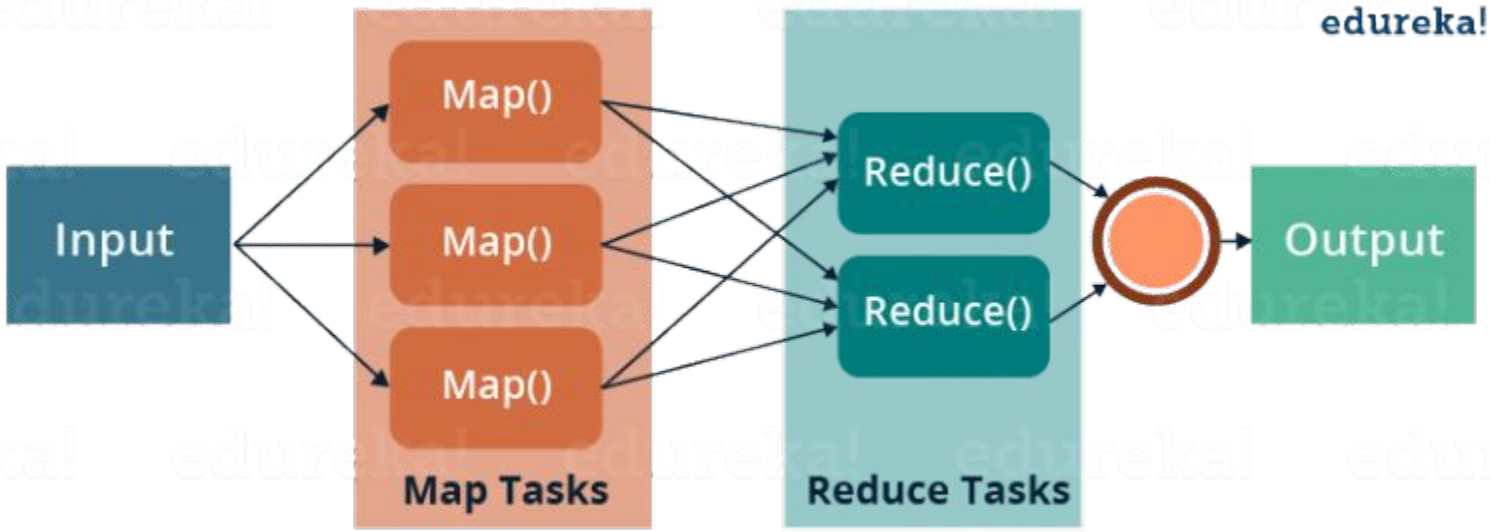
- **Reliability problem:** What if, any of the machines which are working with a part of the data fails? The management of this failover becomes a challenge.

Design issues

- **Equal split issue:** How will I divide the data into smaller chunks so that each machine gets an even part of the data to work with? In other words, how to equally divide the data such that no individual machine is overloaded or underutilized.
- **Single split may fail:** If any of the machines fail to provide the output, I will not be able to calculate the result. So, there should be a mechanism to ensure this fault tolerance capability of the system.
- **Aggregation of the result:** There should be a mechanism to aggregate the result generated by each of the machines to produce the final output.

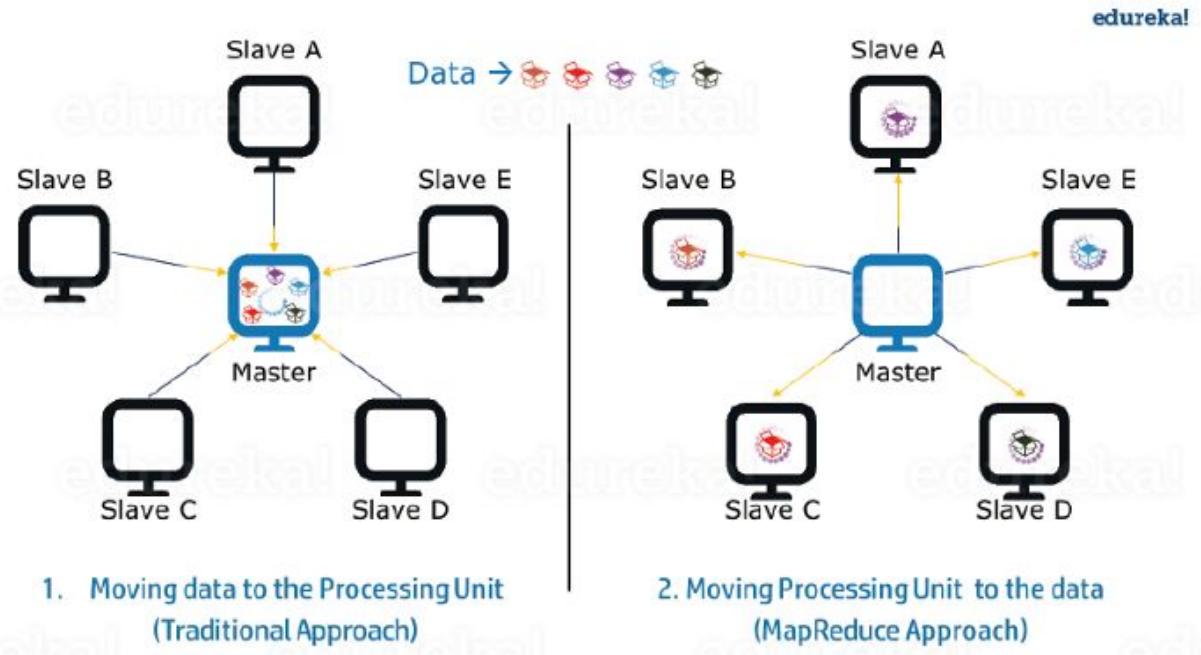
MapReduce

- MapReduce is a programming framework that allows us to perform distributed and parallel processing on large data sets in a distributed environment.



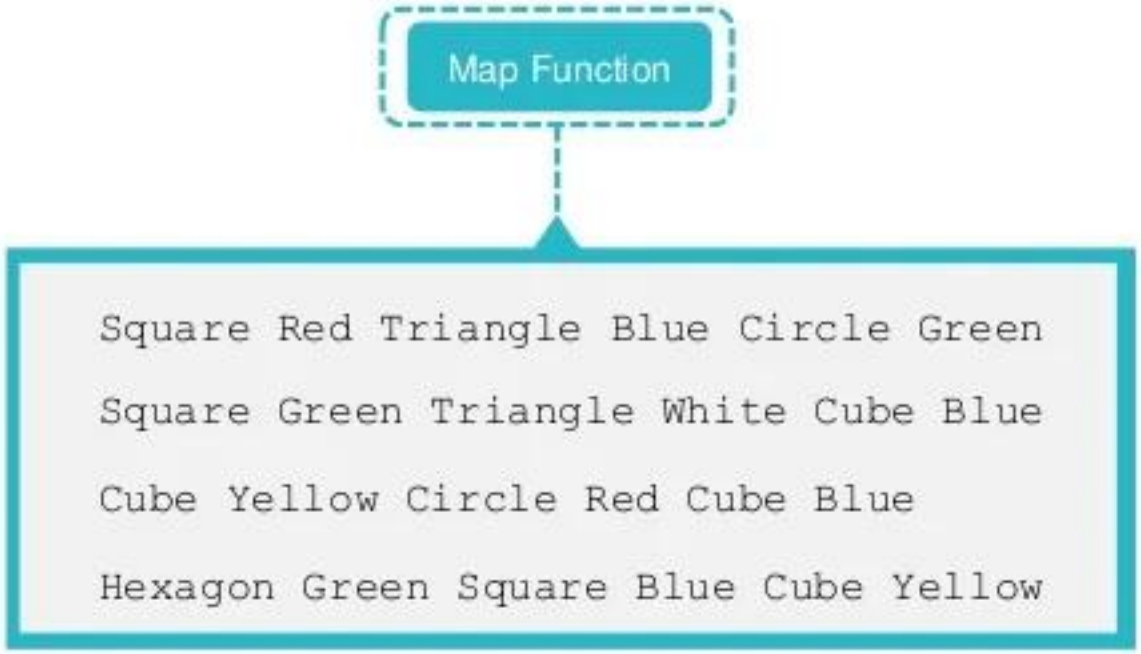
Advantages

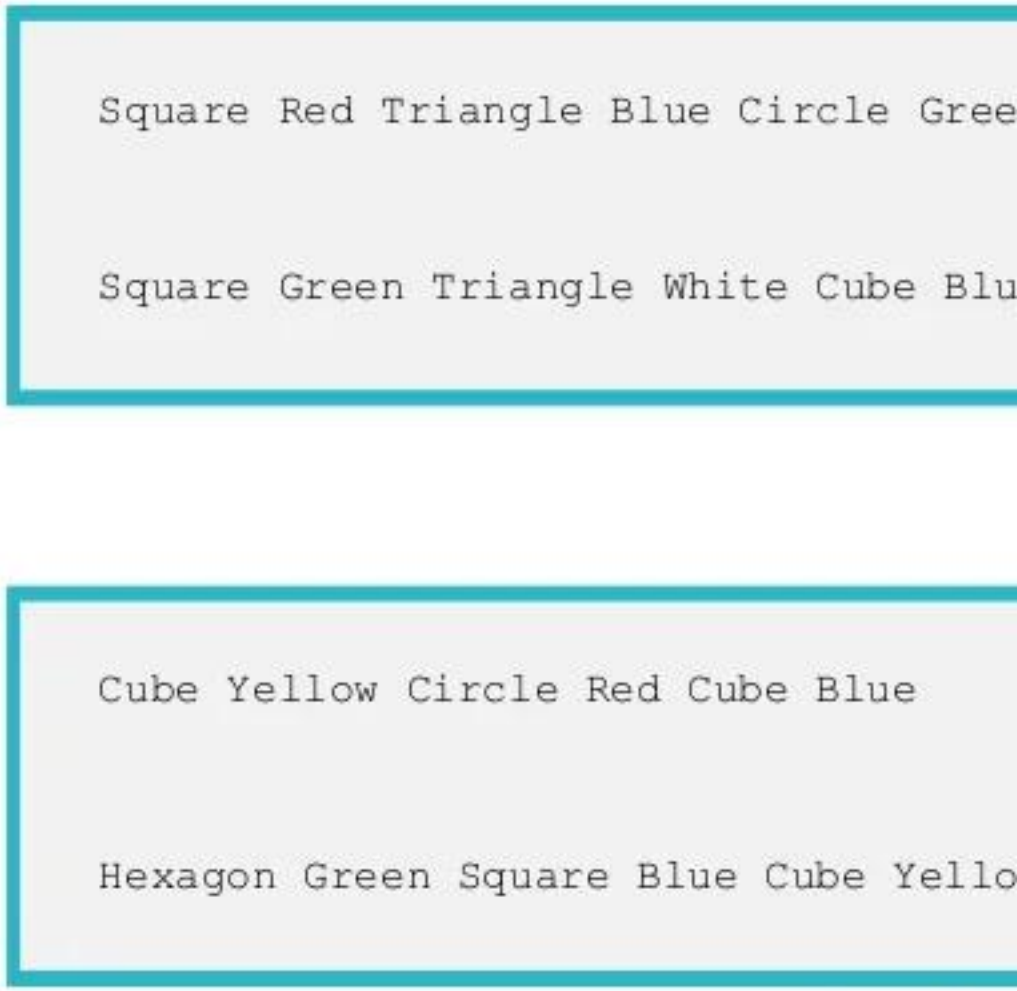
- MapReduce framework which allows programmers to provide **flexibility** to perform parallel computations and write code logic **without bothering about the design issues of the system like reliability, fault tolerance, etc.**
- Move computation to data
- Data locality (compute node and Storage node are the same)



MapReduce Example

MapReduce Example





MapReduce Example

Square Red Triangle Blue Circle Green

Square Green Triangle White Cube Blue

Cube Yellow Circle Red Cube Blue

Hexagon Green Square Blue Cube Yellow

Map step

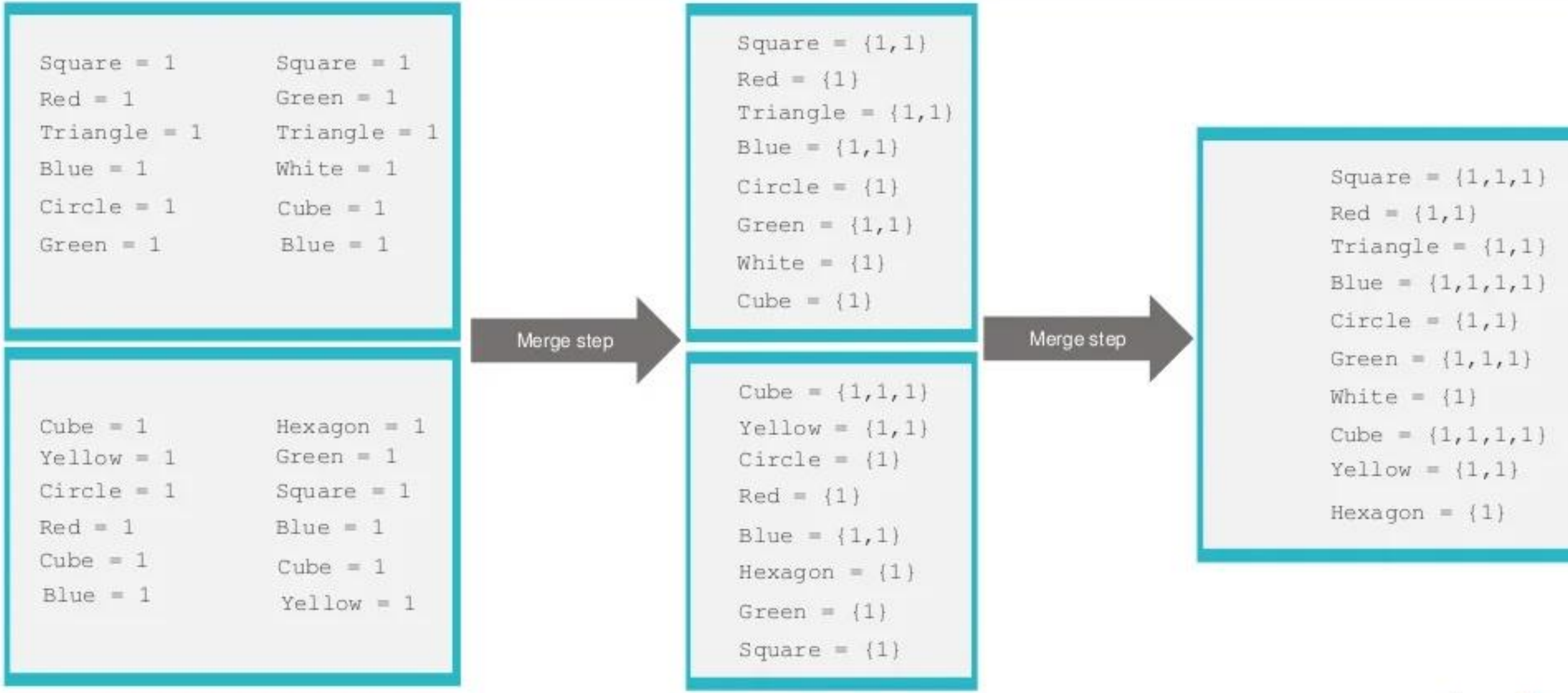
Square = 1
Red = 1
Triangle = 1
Blue = 1
Circle = 1
Green = 1

Square = 1
Green = 1
Triangle = 1
White = 1
Cube = 1
Blue = 1

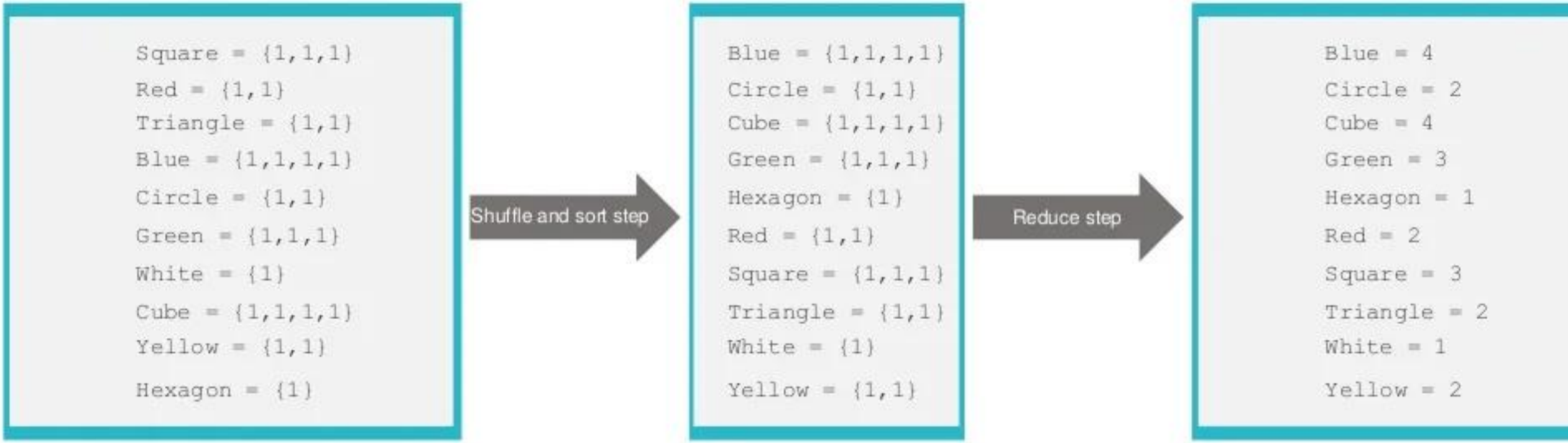
Cube = 1
Yellow = 1
Circle = 1
Red = 1
Cube = 1
Blue = 1

Hexagon = 1
Green = 1
Square = 1
Blue = 1
Cube = 1
Yellow = 1

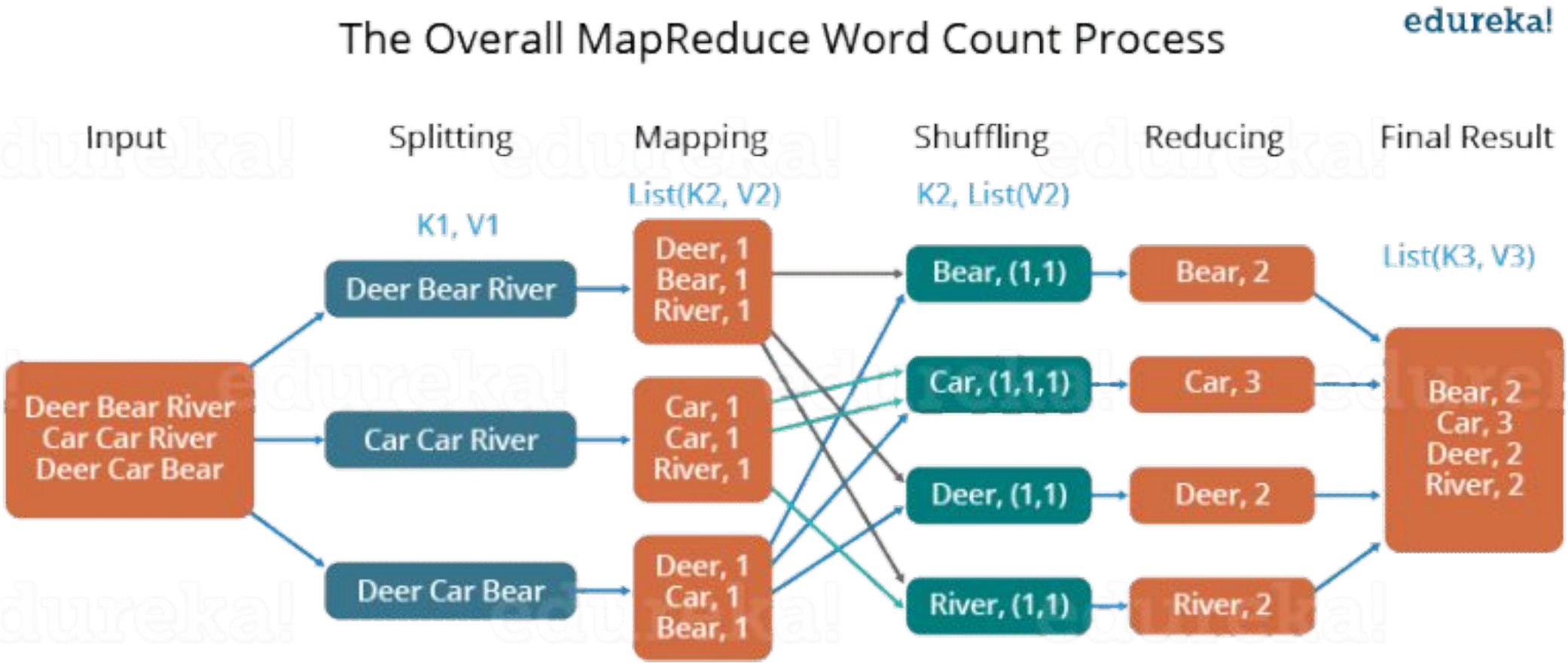
MapReduce Example



MapReduce Example



Word count example - number of occurrences of each word



RDBMS vs. MapReduce

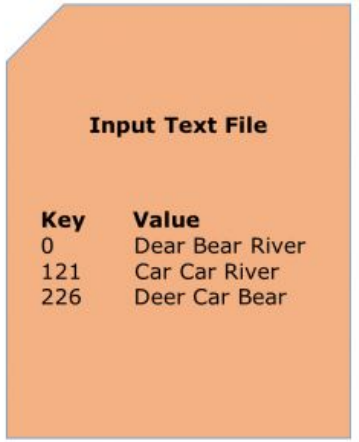
	Traditional RDBMS	MapReduce
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Transactions	ACID	None
Structure	Schema-on-write	Schema-on-read
Integrity	High	Low
Scaling	Nonlinear	Linear

Illustration

The entire MapReduce program can be fundamentally divided into three parts:

- Mapper Phase Code
- Reducer Phase Code
- Driver Code

Illustration – Mapper Phase



Both the input and output of the Mapper is a key/value pair.

Input:

- The *key* is nothing but the offset of each line in the text file: ***LongWritable***
- The *value* is each individual line (as shown in the figure at the right): ***Text***

Output:

- The *key* is the tokenized words: ***Text***
- We have the hard coded *value* in our case which is 1: ***IntWritable***
- Example — Dear 1, Bear 1, etc.

Illustration – Reducer Phase

Both the input and the output of the Reducer is a key-value pairs.

Input:

- The *key* is nothing but those unique words that have been generated after the sorting and shuffling phase: ***Text***
- The *value* is a list of integers corresponding to each key: ***IntWritable***
- Example — **Bear, [1, 1], etc.**

Output:

- The *key* is all the unique words present in the input text file: ***Text***
- The *value* is the number of occurrences of each of the unique words: ***IntWritable***
- Example — **Bear, 2; Car, 3, etc.**

Illustration – Driver code

- In the driver class, we set the configuration of our MapReduce job to run in Hadoop.
- We specify the **name of the job**, and the **data type of input/output** of the mapper and reducer.
- We also specify the **names of the mapper and reducer classes**.
- The **path of the input and output folder** is also specified.
- The method setInputFormatClass () is used for specifying how a Mapper will read the input data or what will be the unit of work.
  - Here, we have chosen **TextInputFormat** so that a single line is read by the mapper at a time from the input text file.
- The main () method is the entry point for the driver. In this method, we instantiate a new Configuration object for the job.

Example - Counting words of different lengths

Input file :

hi how are you?
Welcome to Nirma University.

Output file :

2: 2 , 3:3 , 5:1, 7:1 , 10:1

Illustration

hi:2, how:3, are:3, you:3,welcome:7,to:2,Nirma:5,University:10

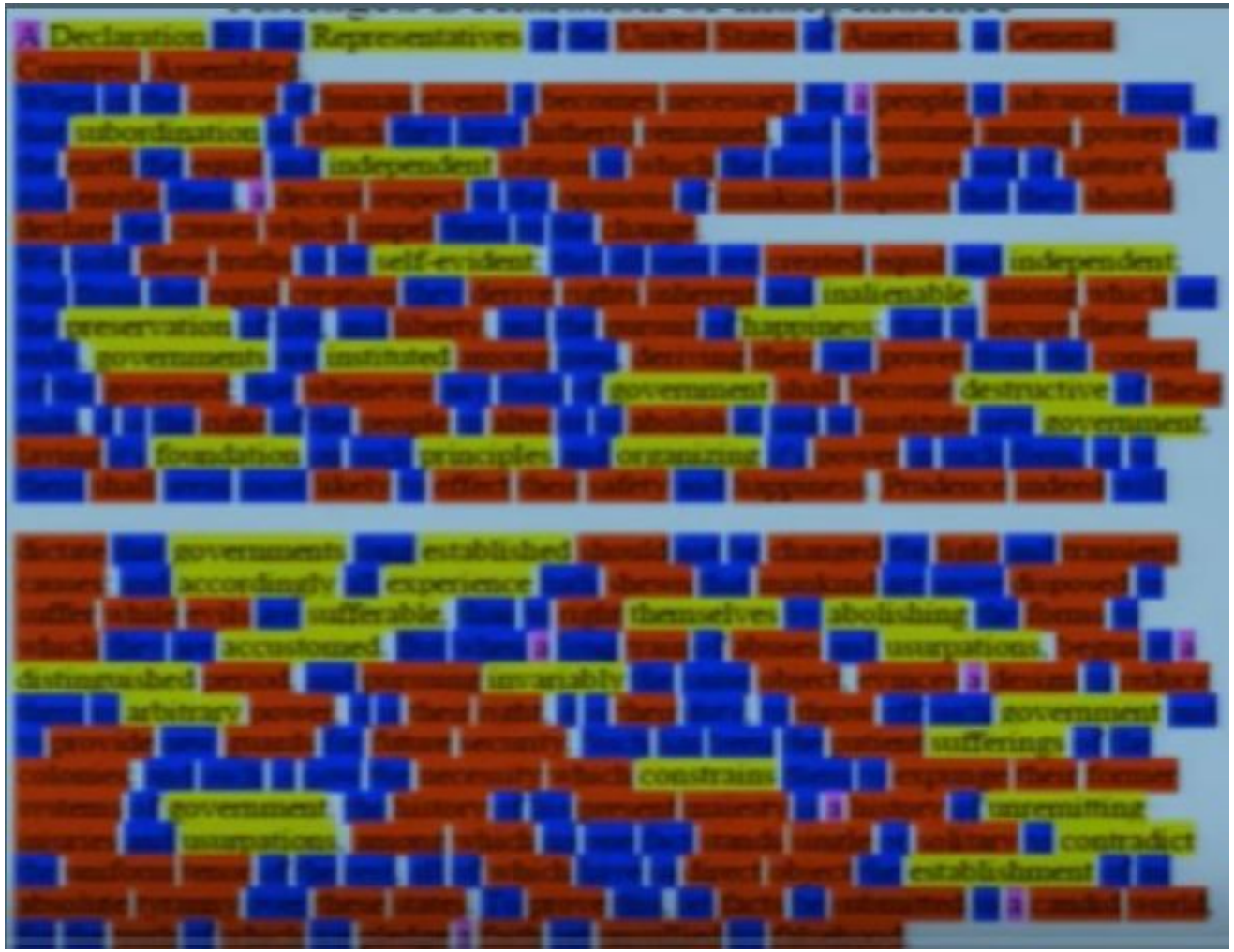
- Mapper Task :
  - Emit(2,hi),(2,to)(3,how)……
- Reducer Task :
  - (2:[hi,to])
  - (3:[how,are,you])
  - ……

Find out the word length histogram

Find how many big, medium, small and tiny words are there in a particular document? ?

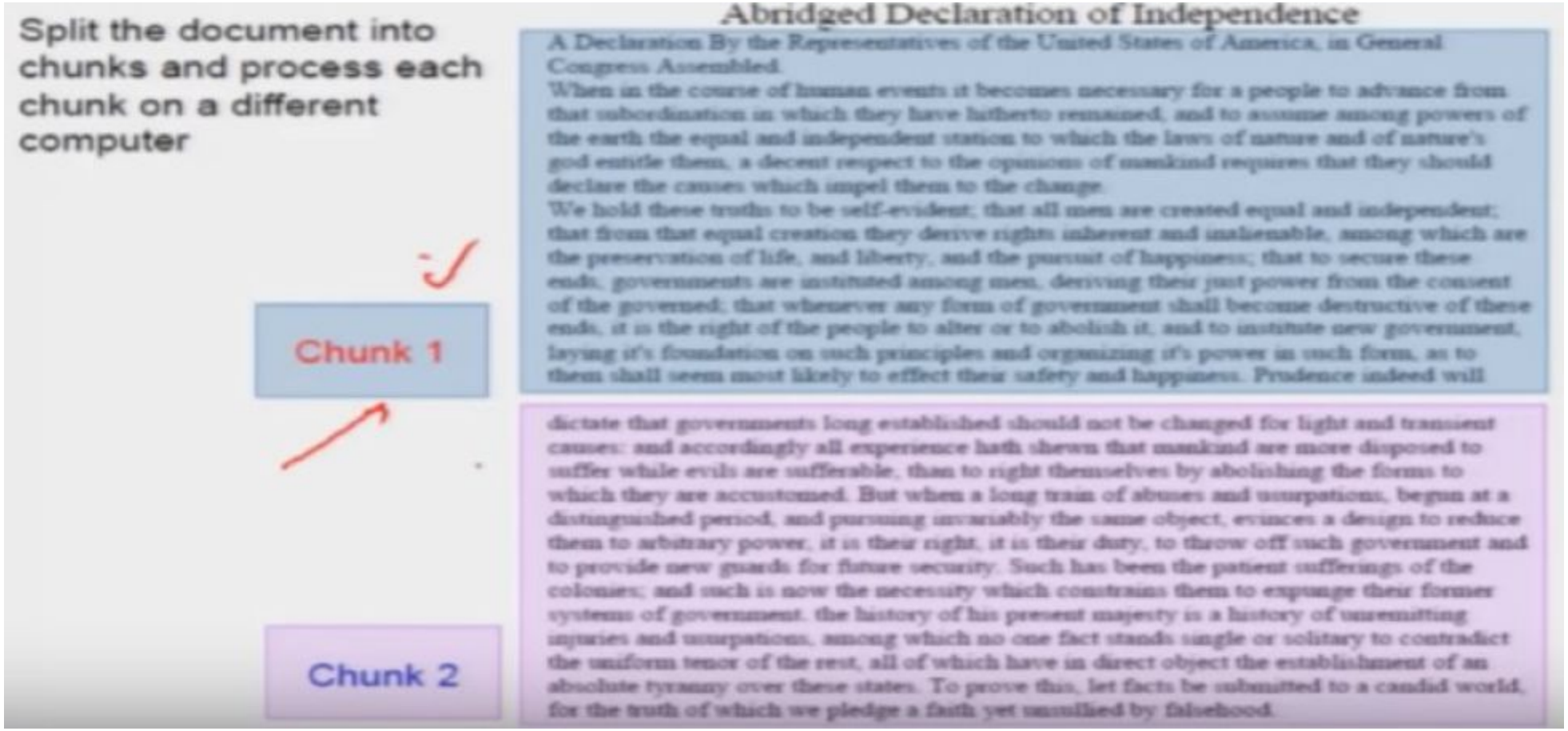
A Declaration By the Representatives of the United States of America, in General Congress Assembled. When in the course of human events it becomes necessary for a people to advance from that subordination in which they have hitherto remained, and to assume among powers of the earth the equal and independent station to which the laws of nature and of nature's god entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the change. We hold these truths to be self-evident; that all men are created equal and independent; that from that equal creation they derive rights inherent and inalienable, among which are the preservation of life, and liberty, and the pursuit of happiness; that to secure these ends, governments are instituted among men, deriving their just power from the consent of the governed; that whenever any form of government shall become destructive of these ends, it is the right of the people to alter or to abolish it, and to institute new government, laying it's foundation on such principles and organizing it's power in such form, as to them shall seem most likely to effect their safety and happiness. Prudence indeed will dictate that governments long established should not be changed for light and transient causes: and accordingly all experience hath shewn that mankind are more disposed to suffer while evils are sufferable, than to right themselves by abolishing the forms to which they are accustomed. But when a long train of abuses and usurpations, begun at a distinguished period, and pursuing invariably the same object, evinces a design to reduce them to arbitrary power, it is their right, it is their duty, to throw off such government and to provide new guards for future security. Such has been the patient sufferings of the colonies; and such is now the necessity which constrains them to expunge their former systems of government. the history of his present majesty is a history of unremitting injuries and usurpations, among which no one fact stands single or solitary to contradict the uniform tenor of the rest, all of which have in direct object the establishment of an absolute tyranny over these states. To prove this, let facts be submitted to a candid world, for the truth of which we pledge a faith yet unsullied by falsehood.

Illustration

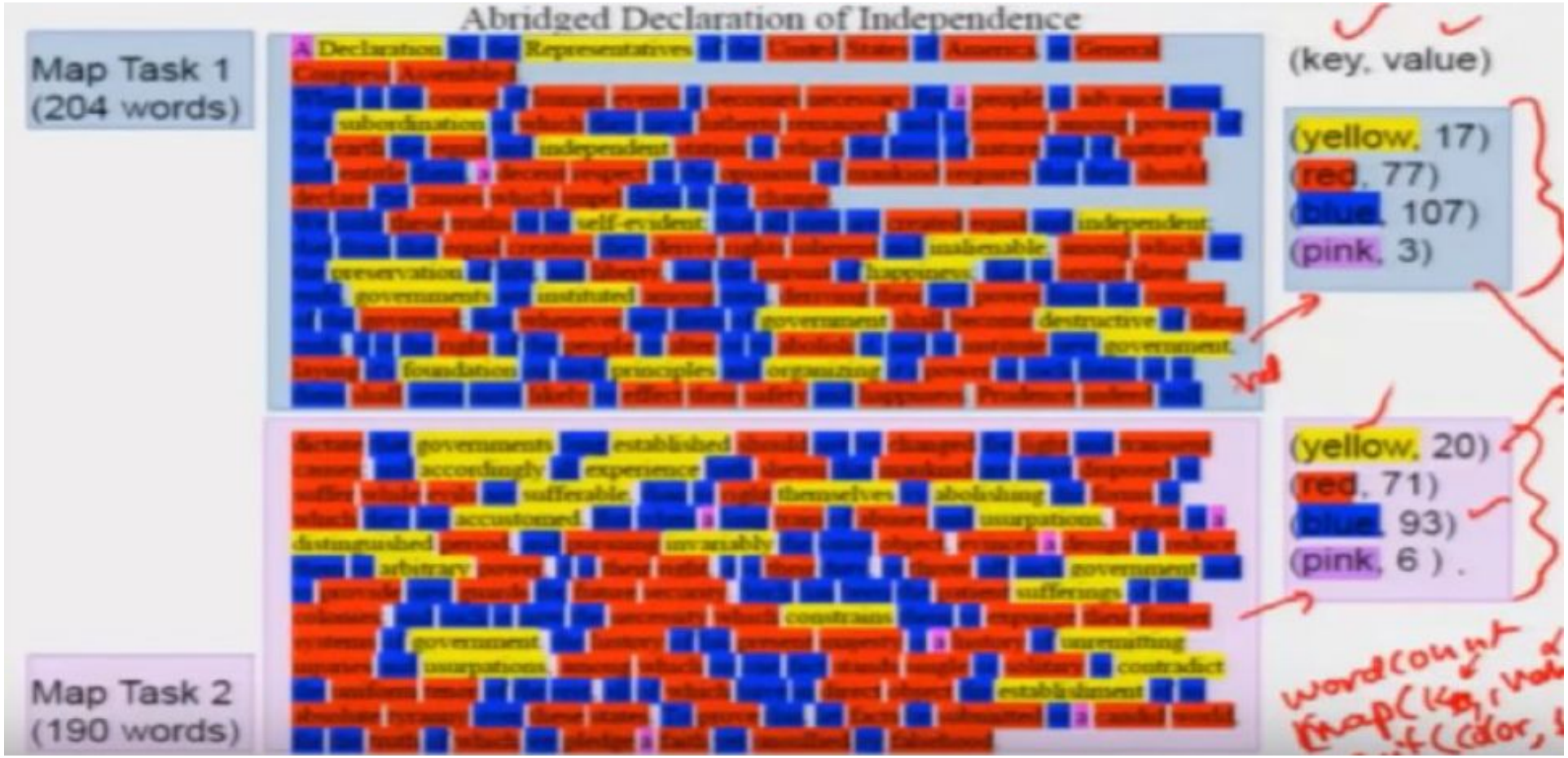


Big : Yellow : 10+
Medium: Red : 5 to 9
Small: Blue : 2to 4
Tiny: pink : 1

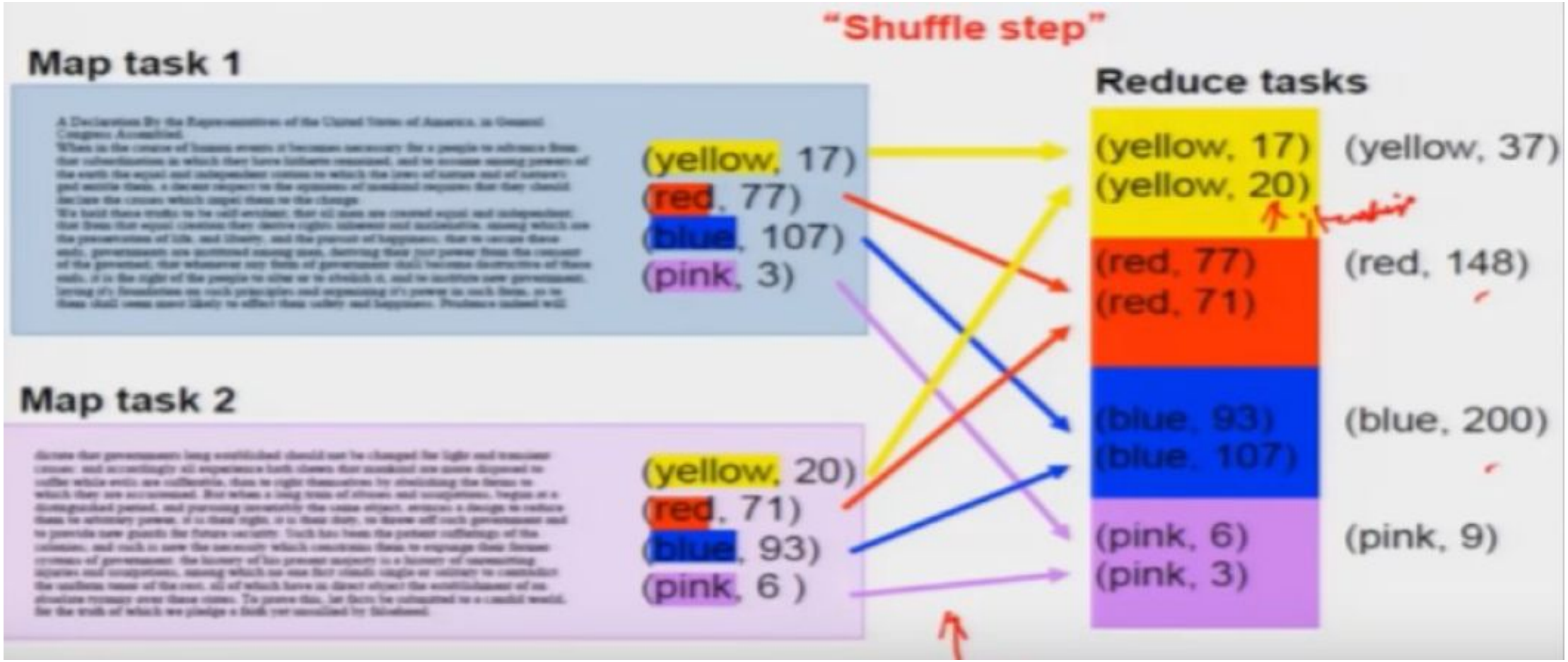
Illustration



Illustration

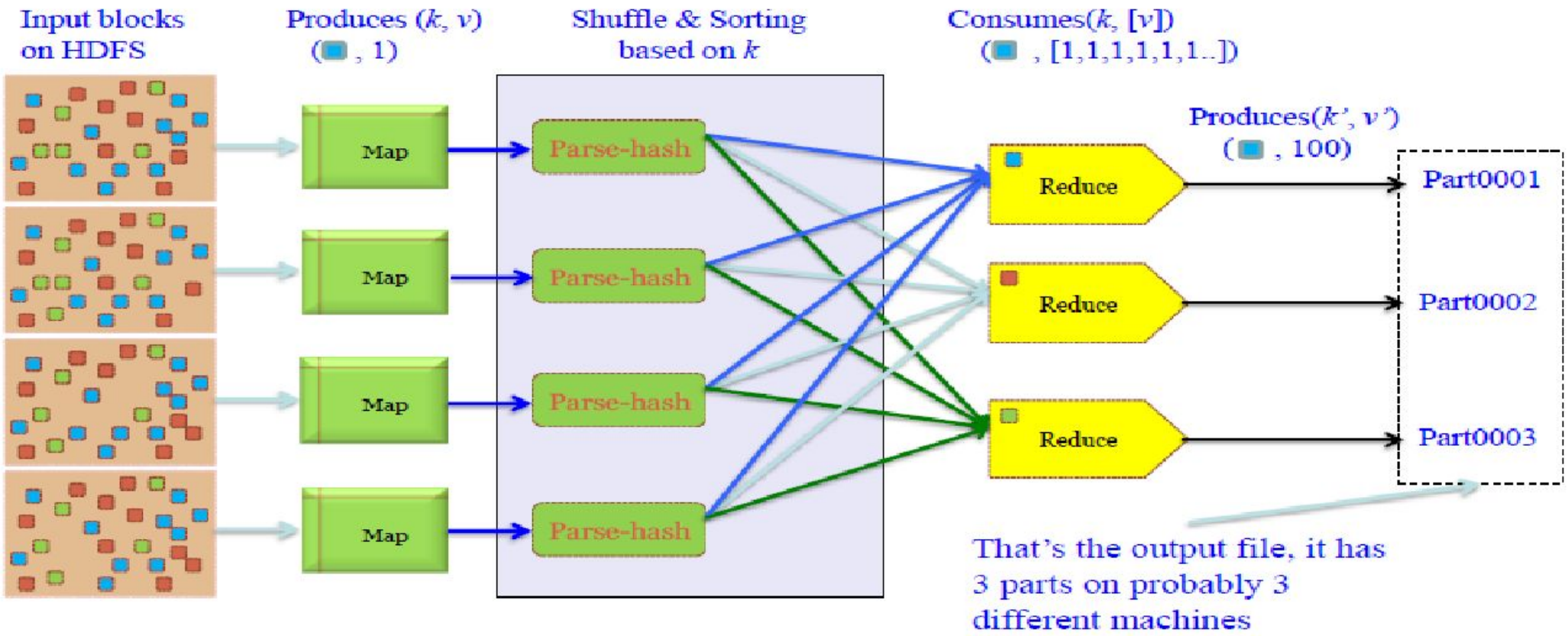


Illustration



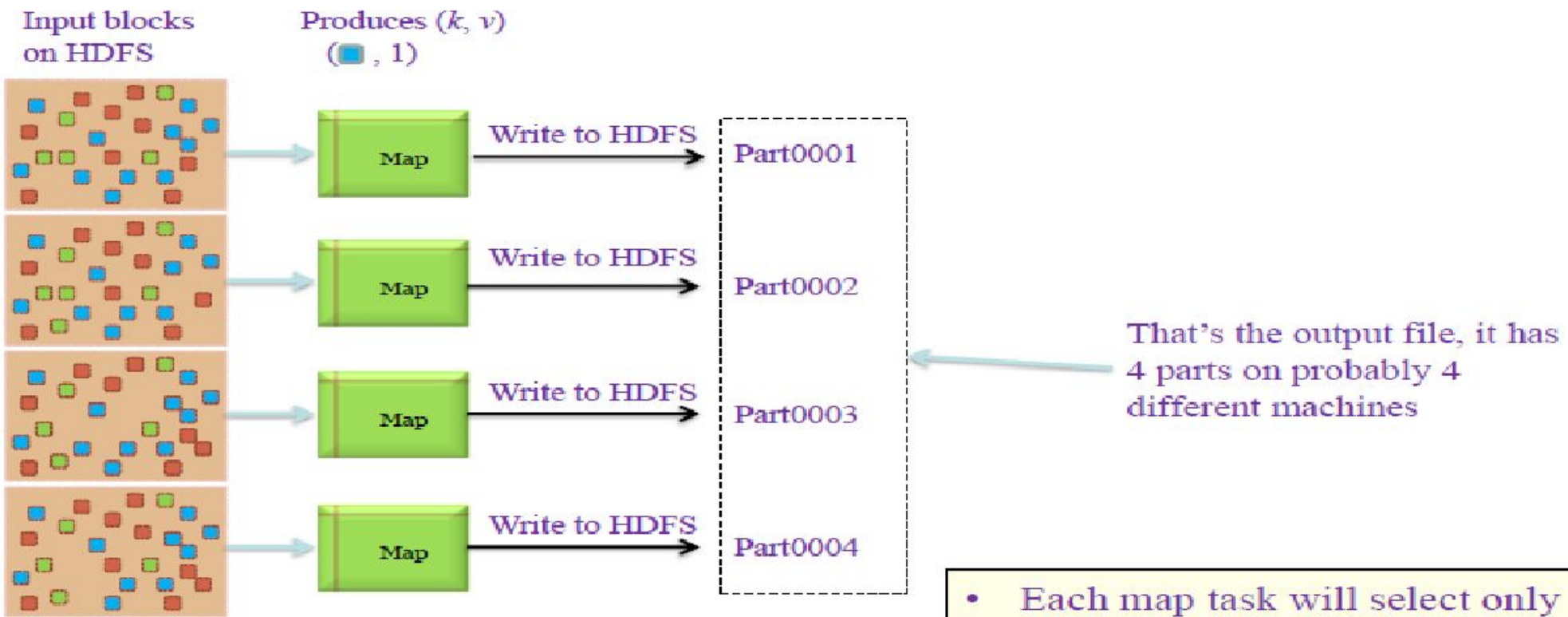
Example 2: Color Count

Job: Count the number of each color in a data set



Example 3: Color Filter

- Job: Select only the blue and the green colors



- Each map task will select only the blue or green colors
- No need for reduce phase

Inverted Index - Finding given word from search engine

Input:

Tweet1, “I love pancakes for breakfast”
Tweet2, “I dislike pancakes”
Tweet3, “What should I eat for breakfast?”
Tweet4, “I love to eat”

Output:

Pancakes(tweet1,tweet2)
Breakfast(tweet1,tweet3)
eat(tweet3,tweet4)
love(tweet1,tweet4)

Find out Mapper and Reducer task.

Example – Find the maximum number

Key questions

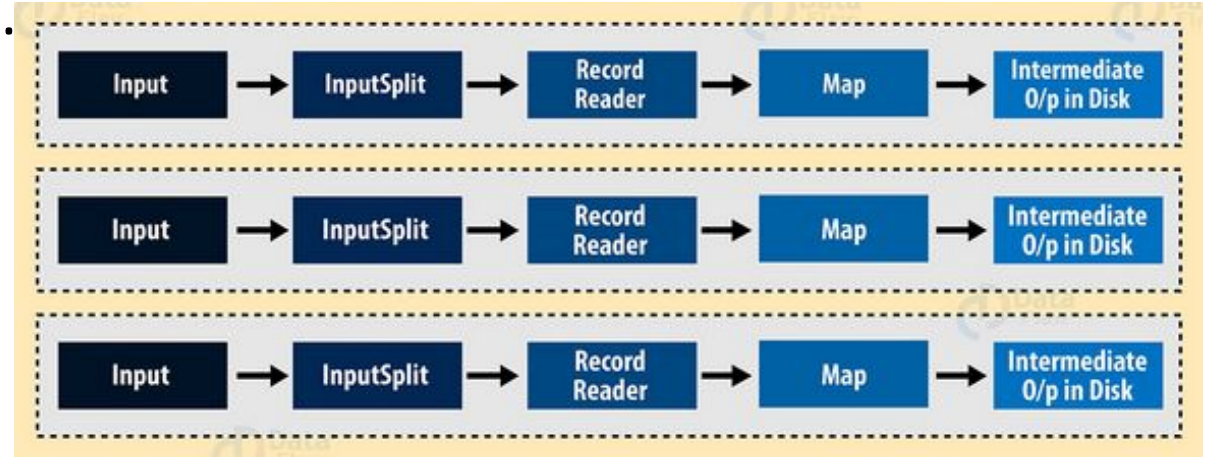
- What is the significance of key-value pairs in MapReduce programming?
- Is MapReduce programming model reading data from HDFS only??
- Is MapReduce programming model reading data from text format only??
- Is Reducer phase mandatory for map reduce programming model
- Is the output of each mapper is equal?

MapReduce - User Interfaces

- Let us first take the Mapper and Reducer interfaces. Applications typically implement them to provide the map and reduce methods.

Mapper

- Mapper maps input key/value pairs to a set of intermediate key/value pairs.
- Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records.
- A given input pair may map to zero or many output pairs.
- The Hadoop MapReduce framework spawns one map task for each InputSplit generated by the InputFormat for the job.



Mapper

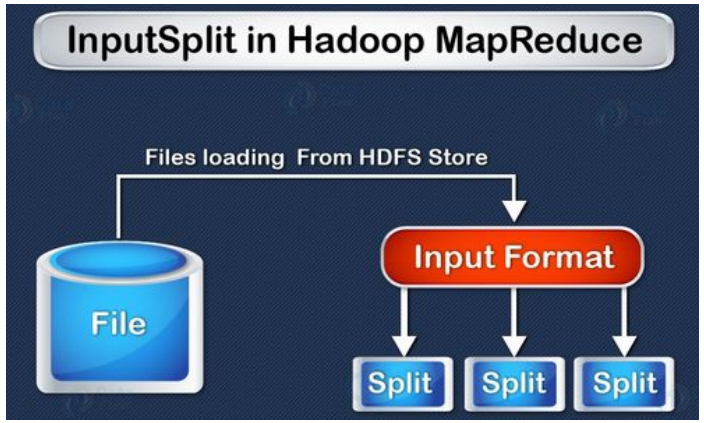
- **How is key value pair generated in Hadoop?**
  1. Input Split
  2. Record Reader
- The first stage in Data Processing using MapReduce is the **Mapper Class.** Here, RecordReader processes each Input record and generates the respective key-value pair. Hadoop's Mapper store saves this intermediate data into the local disk.
- **Input Split**

  It is the logical representation of data. It represents a block of work that contains a single map task in the MapReduce Program.
- **RecordReader**

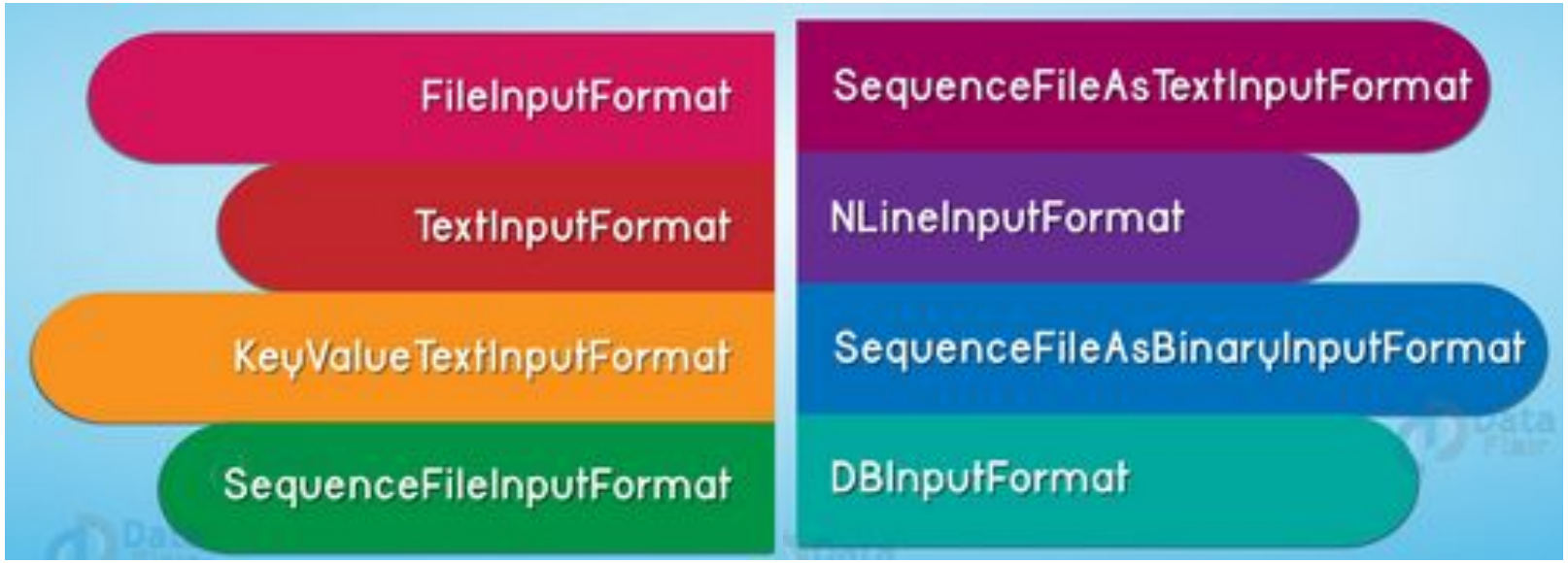
  It interacts with the Input split and converts the obtained data in the form of **Key-Value** Pairs.

InputSplit

- **InputSplit** in Hadoop **MapReduce** is the logical representation of data. It describes a unit of work that contains a single map task in a MapReduce program.
- As a user, we don't need to deal with InputSplit directly, because they are created by an **InputFormat**
- *mapred.min.split.size* parameter in *mapred-site.xml* we can control this value or by overriding the parameter in the Job object used to submit a particular **MapReduce job**.



Types of InputFormat in MapReduce



Mapper Class

- The client (running the job) can calculate the splits for a job by calling '**getSplit()**', and then sent to the application master, which uses their storage locations to schedule map tasks that will process them on the cluster.
- Then, map task passes the split to the *createRecordReader()* method on InputFormat to get **RecordReader** for the split and RecordReader generate record (key-value pair), which it passes to the map function.

Record Reader

- The MapReduce RecordReader in Hadoop takes the byte-oriented view of input, provided by the InputSplit and presents as a record-oriented view for Mapper.
- Map task passes the split to the **createRecordReader()** method on InputFormat in task tracker to obtain a RecordReader for that split. The RecordReader load's data from its source and converts into key-value pairs suitable for reading by the **mapper.**

Types of Hadoop Record Reader in MapReduce

i. LineRecordReader

ii. SequenceFileRecordReader

- **Maximum size for a Single Record**
- conf.setInt("mapred.linerecordreader.maxlength", Integer.MAX_VALUE);
- A line with a size greater than this maximum value (default is 2,147,483,647) will be ignored.

Reducer Class

The Intermediate output generated from the mapper is fed to the reducer which processes it and generates the final output which is then saved in the **HDFS.**

Hadoop Record Writer

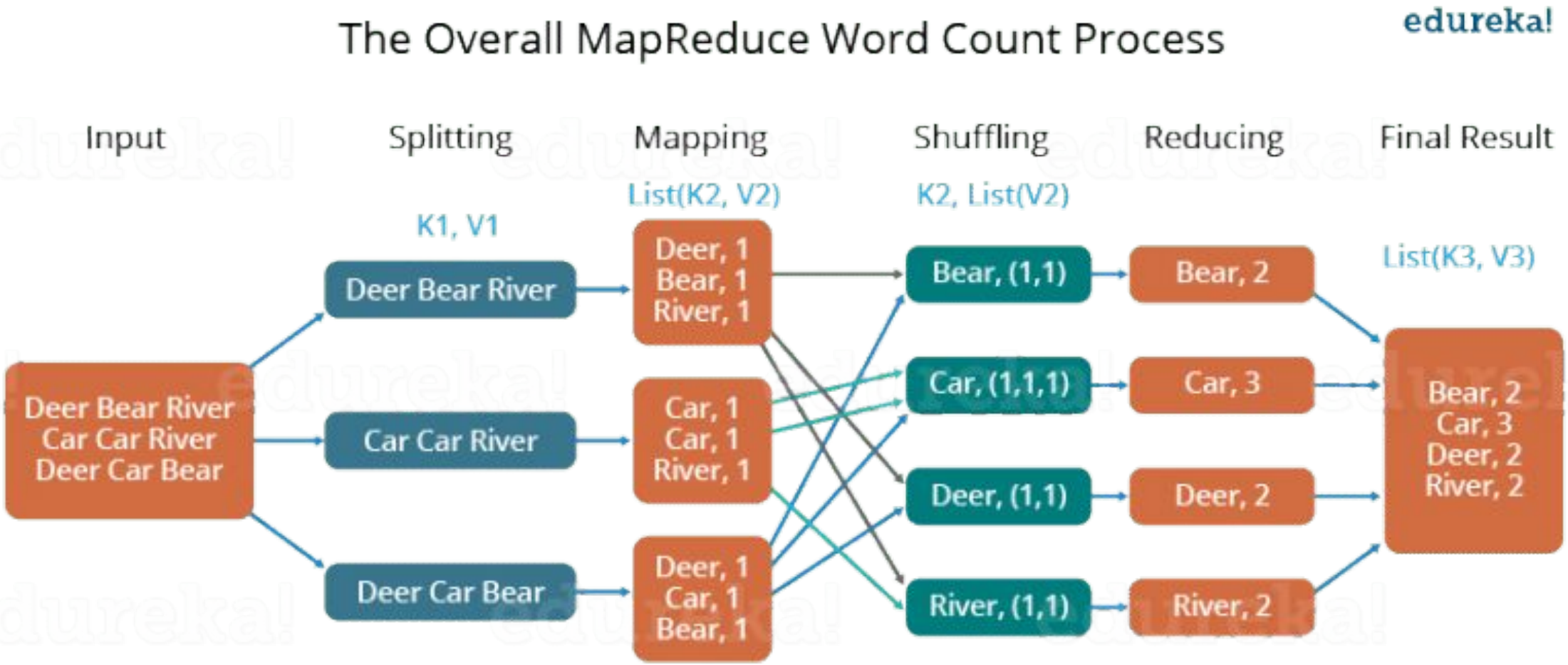
- Record Writer writes output key-value pairs from the Reducer phase to output files.
- **TextOutputFormat**
- **SequenceFileOutputFormat**
- **SequenceFileAsBinaryOutputFormat**
- **MapFileOutputFormat**
- **MultipleOutputs**
- **LazyOutputFormat**
- **DBOutputFormat**

Driver Class

The major component in a MapReduce job is a **Driver Class.** It is responsible for setting up a MapReduce Job to run-in Hadoop. We specify the names of **Mapper** and **Reducer** Classes long with data types and their respective job names.

Example

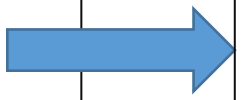
input : Dear, Bear, River, Car, Car, River, Deer, Car and Bear



Word count Job

Input : Text file
Output : count of words

Hi how are you?
how is your job?
how is your family
how is your sister
how is your brother
what is the time now
what is the strength of the Hadoop



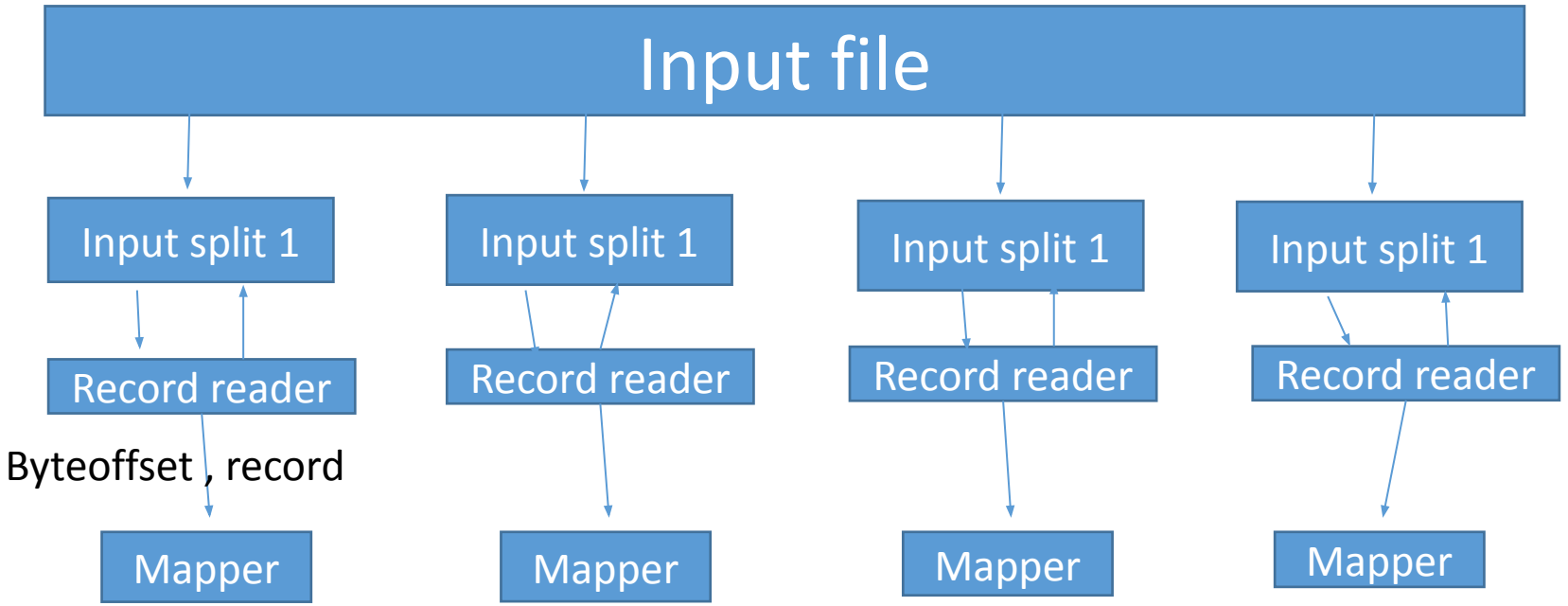
Hi how are you
how is your job

how is your family
how is your sister

How is your brother
what is the time now

what is the strength of the Hadoop

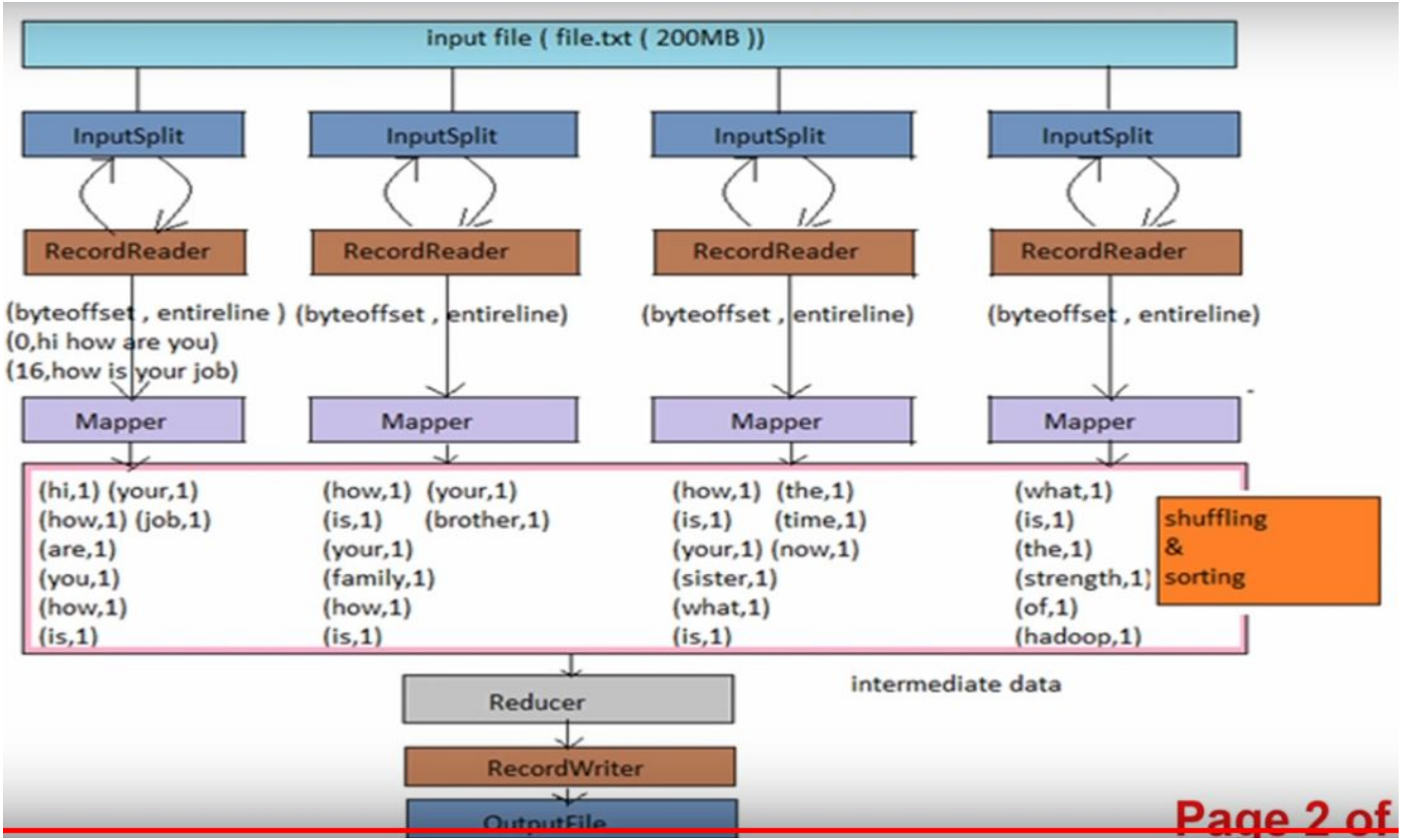
File.txt
Size :: 200MB



Text Input format
Key Value Text Input Format
Sequence File Input Format
SequenceFileAsTextInput Format

Primitive types	**Wrapper Class**	**Box Class**
int	Integer	IntWritable
long	Long	LongWritable
float	Float	FloatWritable
double	Double	DoubleWritable
string	String	StringWritable
char	Character	CharWritable
etc…	etc..	etc..

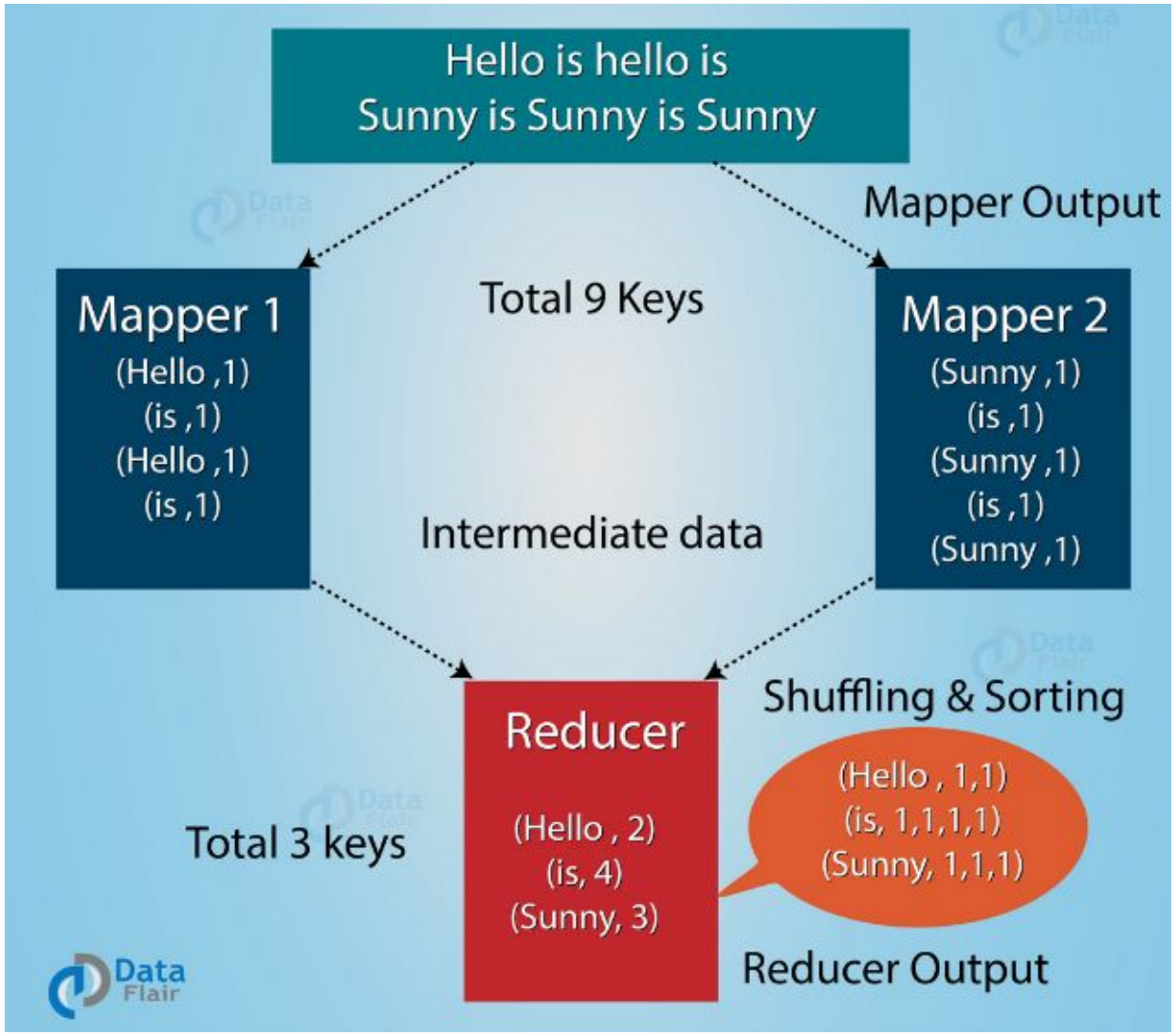
Because of collection framework , as it doesnot work on the primitive types, wrapper classes are created.
Collection framework Work with the object to type so object of wrapper class is to be created. Similar to Java as it has introduced wrapper class corresponding to primitive class. Hadoop has introduced Box classes.
For Java conversion from primitive to wrapper is done automatically but for Hadoop we need to explicitly mention that conversion.
Int - new IntWritable(int) and get () method for back


input file (file.txt (200MB))
InputSplit
InputSplit
InputSplit
InputSplit
RecordReader
RecordReader
RecordReader
RecordReader
(byteoffset , entireline)
(0,hi how are you)
(16,how is your job)
(byteoffset , entireline)
(byteoffset , entireline)
(byteoffset , entireline)
Mapper
Mapper
Mapper
Mapper
(hi,1) (your,1)
(how,1) (job,1)
(are,1)
(you,1)
(how,1)
(is,1)
(how,1) (your,1)
(is,1) (brother,1)
(your,1)
(family,1)
(how,1)
(is,1)
(how,1) (the,1)
(is,1) (time,1)
(your,1) (now,1)
(sister,1)
(what,1)
(is,1)
(what,1)
(is,1)
(the,1)
(strength,1)
(of,1)
(hadoop,1)
shuffling
&
sorting
intermediate data
Reducer
RecordWriter
OutputFile

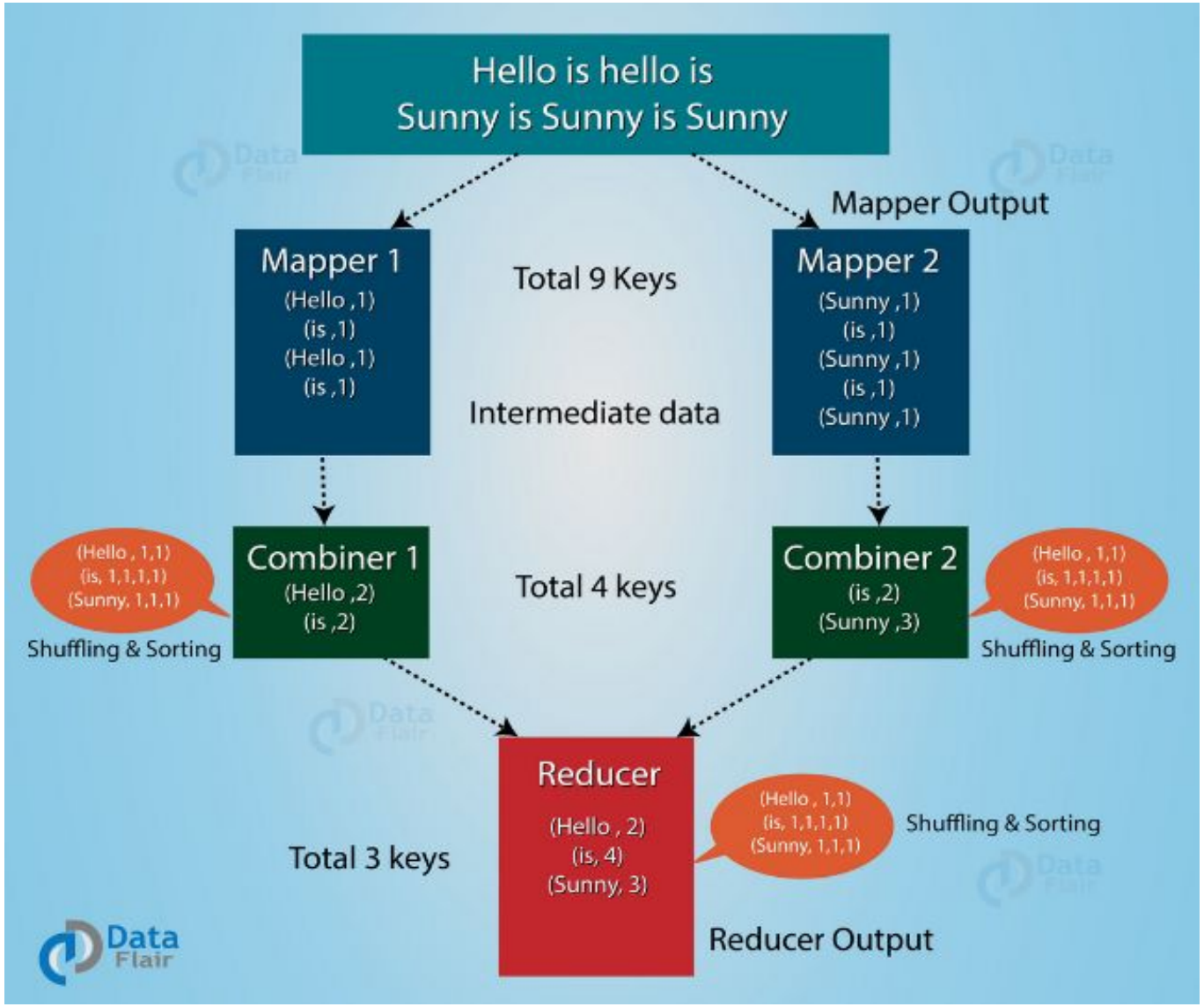
Hadoop/Map Reduce Combiners

- On a large dataset when we run MapReduce job, large chunks of intermediate data is generated by the Mapper and this intermediate data is passed on the Reducer for further processing, which leads to enormous **network congestion**. MapReduce framework provides a function known as **Hadoop Combiner** that plays a key role in reducing network congestion
- The combiner in MapReduce is also known as **'Mini-reducer'.** The primary job of **Combiner is to process the output data from the Mapper, before passing it to Reducer**. It runs after the mapper and before the Reducer and its use is optional

MapReduce program without Combiner



MapReduce program with Combiner



Advantages of MapReduce Combiner

- Hadoop Combiner reduces the time taken for data transfer between mapper and reducer.
- It decreases the amount of data that needs to be processed by the reducer.
- The Combiner improves the overall performance of the reducer.

Disadvantages of MapReduce Combiner

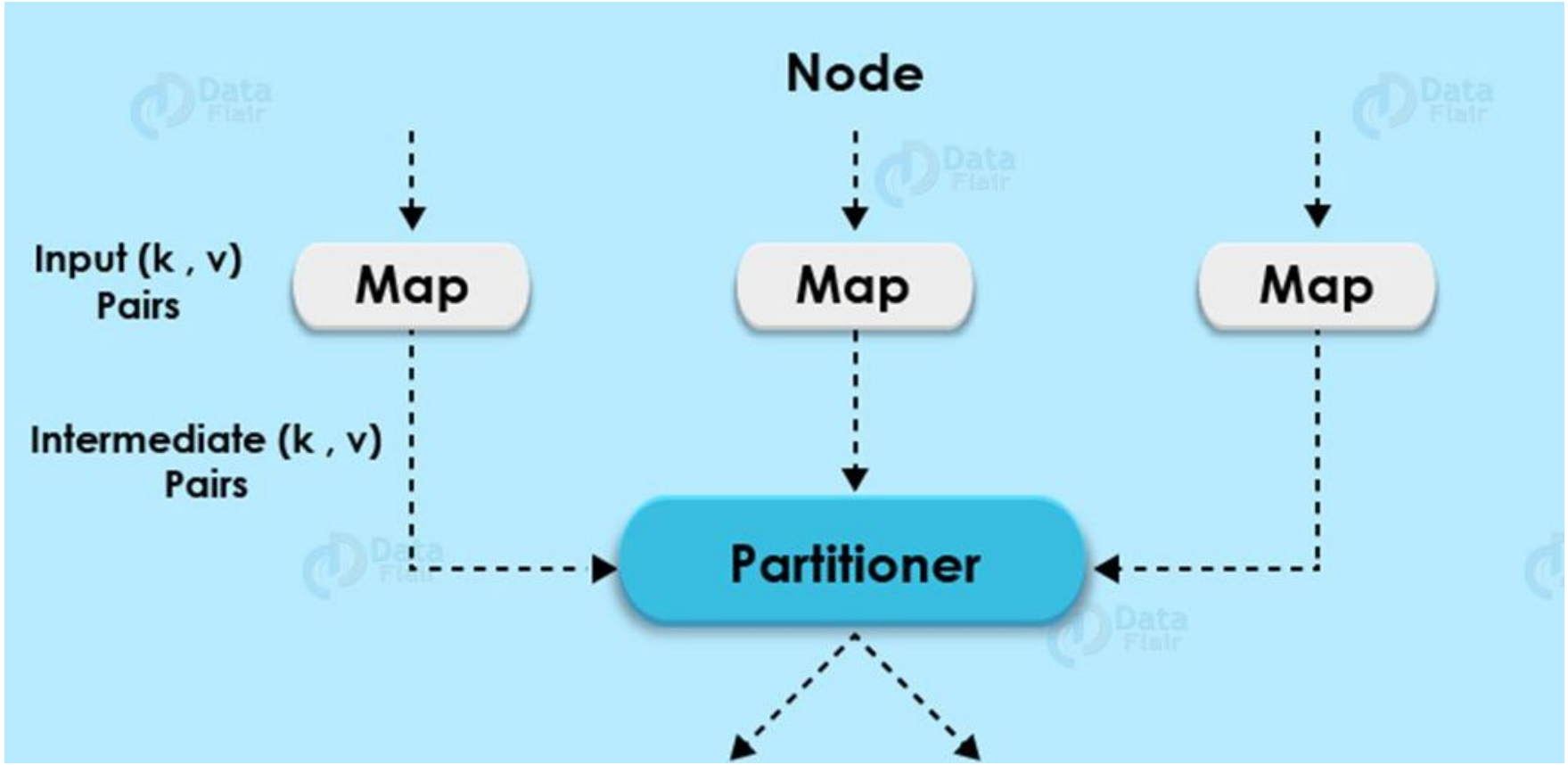
- MapReduce jobs cannot depend on the Hadoop combiner execution because there is no guarantee in its execution.
- In the local filesystem, the key-value pairs are stored in the Hadoop and run the combiner later which will cause expensive disk IO.

combiner Vs. reducer

- Combiner - mini reducer that performs local reduce task.
- It runs on the Map output and produces the output to reducers input.
- It is usually used for network optimization when the map generates greater number of outputs.
- **The combiner has a constraint that the input or output key and value types must match the output types of the Mapper.**
- Combiners can operate only on a subset of keys and values i.e. combiners can be executed **on functions that are commutative**.
- Combiner functions get their input from a single mapper whereas reducers can get data from multiple mappers as a result of partitioning.

Hadoop Partitioner / MapReduce Partitioner

- **Partitioning** of the keys of the intermediate map output is controlled by the Partitioner.



Hadoop MapReduce Partitioner

- MapReduce job takes an input data set and produces the list of the key-value pair which is the result of map phase in which input data is split and each task processes the split and each map, output the list of key-value pairs. Then, the output from the map phase is sent to reduce task which processes the user-defined reduce function on map outputs.

- Before reduce phase, partitioning of the map output take place on the basis of the key and sorted.

Hadoop MapReduce Partitioner

- This partitioning specifies that all the values for each key are grouped together and make sure that all the values of a single key go to the same reducer, thus allowing even distribution of the map output over the reducer.
- Partitioner in Hadoop MapReduce redirects the mapper output to the reducer by determining which reducer is responsible for the particular key.
- The Default Hadoop partitioner in Hadoop MapReduce is Hash Partitioner which computes a hash value for the key and assigns the partition based on this result.

How many Partitioners

- The total number of Partitioners that run in Hadoop is equal to the number of reducers i.e. Partitioner will divide the data according to the number of reducers which is set by *JobConf.setNumReduceTasks()* method.
- Thus, the data from a single partitioner is processed by a single reducer. And partitioner is created only when there are multiple reducers.

Poor Partitioning in Hadoop MapReduce

- If in data input one key appears more than any other key then
  1. The key appearing more will be sent to one partition.
  2. All the other key will be sent to partitions according to their hashCode().
- If **hashCode()** method does not uniformly distribute other keys data over partition range, then data will not be evenly sent to reducers.
- Poor partitioning of data means that some reducers will have more data input than other i.e. they will have more work to do than other reducers. So, the entire job will wait for one reducer to finish its extra-large share of the load.
- we can create Custom partitioner, which allows sharing workload uniformly across different reducers.

How to set number of reducers?

- By default no of reducer=1
- If you mention *JobConf.setNumReduceTasks(0) then no of reducers are 0 and process will be executed only using mappers. No sorting& shuffling will applied*
- *Methods to set no of reducers*
    1. ***Command line*** *(*bin/hadoop jar -Dmapreduce.job.maps=5 yourapp.jar..)
       mapred.map.tasks --> mapreduce.job.maps
       mapred.reduce.tasks --> mapreduce.job.reduces
    2. In the code, one can configure JobConf variables.
       job.setNumMapTasks(5); // 5 mappers
       job.setNumReduceTasks(2); // 2 reducers

How Many Maps?

- The number of maps is usually driven by the total size of the inputs, that is, the total number of blocks of the input files.

- The right level of parallelism for maps seems to be around 10-100 maps per-node, although it has been set up to 300 maps for very cpu-light map tasks. Task setup takes a while, so it is best if the maps take at least a minute to execute.

- Thus, if you expect 10TB of input data and have a blocksize of 128MB, you'll end up with 82,000 maps, unless Configuration.set(MRJobConfig.NUM_MAPS, int) (which only provides a hint to the framework) is used to set it even higher.

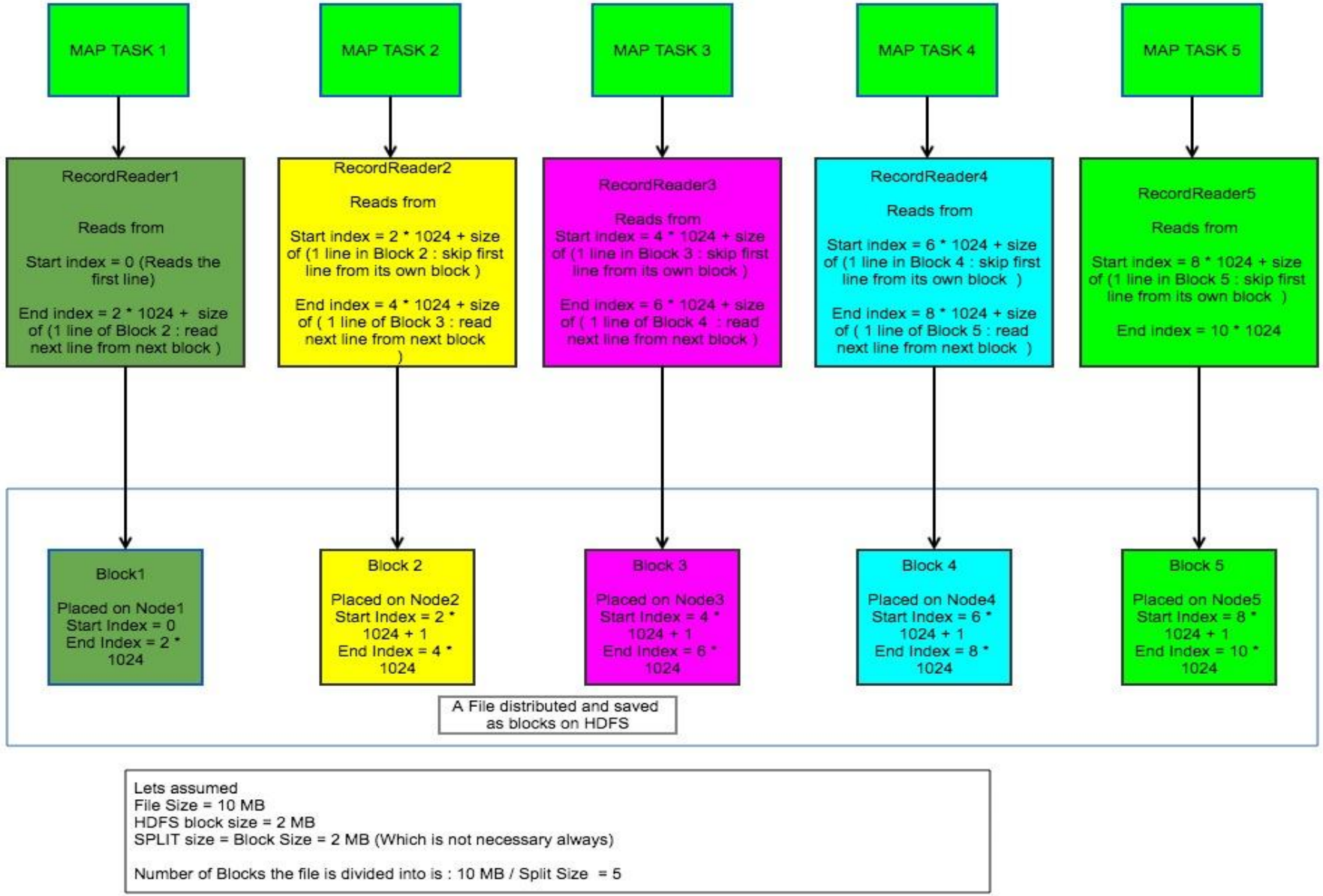
How Many Reducers?

- The right number of reduces seems to be 0.95 or 1.75 multiplied by (<no. of nodes> * <no. of maximum containers per node>).
- With 0.95 all of the reduces can launch immediately and start transferring map outputs as the maps finish. With 1.75 the faster nodes will finish their first round of reduces and launch a second wave of reduces doing a much better job of load balancing.
- Increasing the number of reduces increases the framework overhead, but increases load balancing and lowers the cost of failures.
- The scaling factors above are slightly less than whole numbers to reserve a few reduce slots in the framework for speculative-tasks and failed tasks.

Key question

- Is HDFS block and MapReduce InputSplit the same?

HDFS splits its blocks (byte-oriented view) so that each block is less than or equal to the block size configured. So it is considered to be not following a logical split. This means a part of the last record may reside in one block and the rest of it is in another block. This seems correct for storage. But At processing time, the partial records in a block cannot be processed as it is. So the record-oriented view comes into place. This will ensure to get the remaining part of the last record in the other block to make it a block of complete records. This is called input-split (record-oriented view).



Lets assumed
File Size = 10 MB
HDFS block size = 2 MB
SPLIT size = Block Size = 2 MB (Which is not necessary always)

Number of Blocks the file is divided into is : 10 MB / Split Size = 5

Where to define it?

In Driver class

- **job. Set Combiner Class(ReduceClass.class);**

Separate java file

- **public class Combiners Hadoop {**