
# Hadoop MapReduce

*Divide and Conquer...*

# What is **MapReduce?**

**MapReduce** is the processing layer of **Hadoop**.

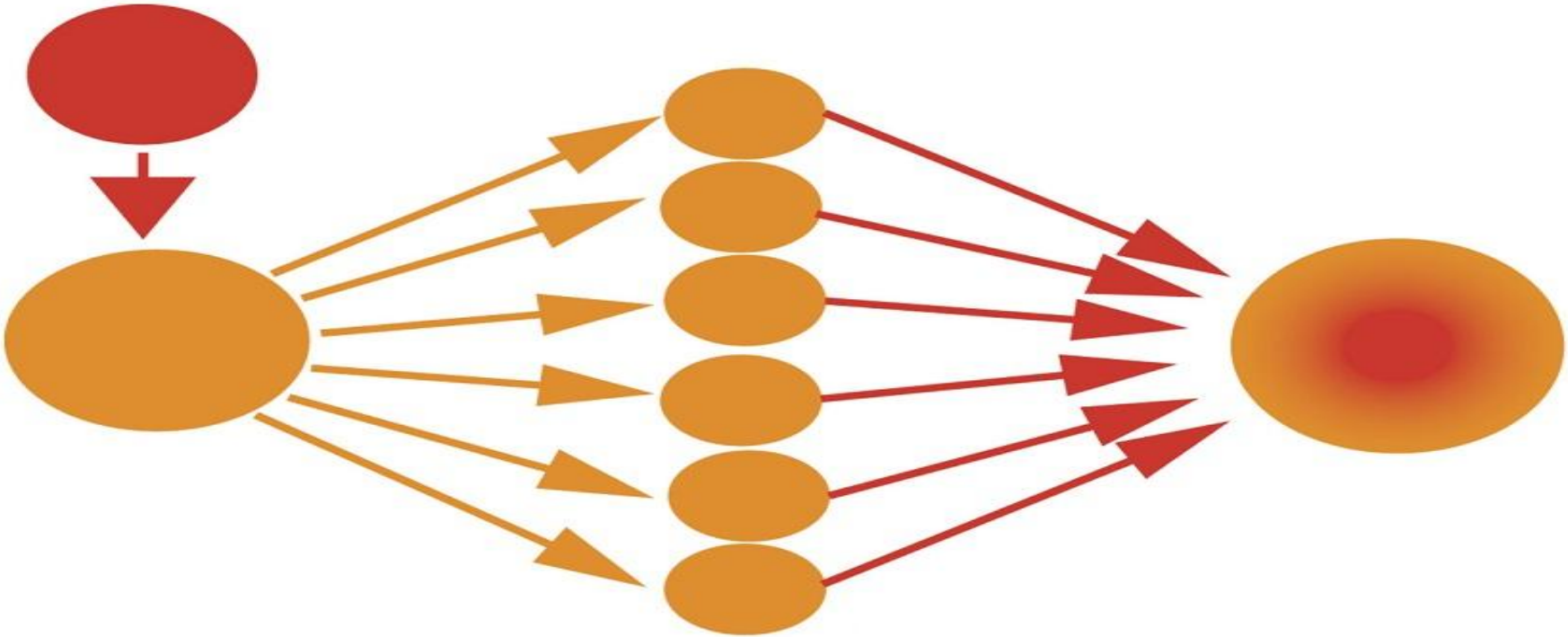
MapReduce programming model is designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks.
You need to put business logic in the way MapReduce works and rest things will be taken care by the framework.
Work (complete job) which is submitted by the user to master is divided into small works (tasks) and assigned to slaves.
MapReduce programs are written in a particular style influenced by functional programming constructs, specifical idioms for processing lists of data.
Here in MapReduce, we get inputs from a list and it converts it into output which is again a list.
It is the heart of Hadoop.
Hadoop is so much powerful and efficient due to MapRreduce as here parallel processing is done.

# How MapReduce work?

Map-Reduce divides the work into small parts, each of which can be done in parallel on the cluster of servers. A problem is divided into a large number of smaller problems each of which is processed to give individual outputs. These individual outputs are further processed to give final output.

Hadoop Map-Reduce is scalable and can also be used across many computers. Many small machines can be used to process jobs that could not be processed by a large machine.

# Parallel Processing



# MapReduce Terminologies

**MapReduce** is the data processing component of Hadoop. MapReduce programs transform lists of input data elements into lists of output data elements. A MapReduce program will do this twice, using two different list processing idioms:

- **Map**
- **Reduce**

In between Map and Reduce, there is small phase called **Shuffle** and **Sort** in MapReduce.

# What is a **MapReduce** Job?

**MapReduce** Job or a A “full program” is an execution of a Mapper and Reducer across a data set.
It is an execution of 2 processing layers i.e mapper and reducer.
A MapReduce job is a work that the client wants to be performed.
It consists of the input data, the MapReduce Program, and configuration info.

So client needs to submit input data, he needs to write MapReduce program and set the configuration info (These were provided during Hadoop setup in the configuration file and also we specify some configurations in our program itself which will be specific to our mapReduce job).

# What is **Task** in **MapReduce**?

A task in MapReduce is an execution of a Mapper or a Reducer on a slice of data.

It is also called Task-In-Progress (TIP).

It means processing of data is in progress either on mapper or reducer.

# What is **Task Attempt**?

**Task Attempt** is a particular instance of an attempt to execute a task on a node.
There is a possibility that anytime any machine can go down.
For example, while processing data if any node goes down, framework reschedules the task to some other node.
This rescheduling of the task cannot be infinite.
There is an upper limit for that as well.
The **default value** of task attempt is **4**.
If a task (Mapper or reducer) fails 4 times, then the job is considered as a failed job.
For high priority job or huge job, the value of this task attempt can also be

# Map Abstraction

Let us understand the abstract form of **Map** in **MapReduce**, the first phase of MapReduce paradigm, what is a map/mapper, what is the input to the mapper, how it processes the data, what is output from the mapper?
The map takes key/value pair as input. Whether data is in structured or unstructured format, framework converts the incoming data into key and value.

- Key is a reference to the input value.
- Value is the data set on which to operate.

**Map Processing**:

- A function defined by user – user can write custom business logic according to his need to process the data.
- Applies to every value in value input.

**Map produces a new list of key/value pairs**:

- An output of Map is called intermediate output.
- Can be the different type from input pair.
- An output of map is stored on the local disk from where it is shuffled to reduce nodes.

# Reduce Abstraction

**Reduce** takes intermediate Key / Value pairs as input and processes the output of the mapper. Usually, in the reducer, we do aggregation or summation sort of computation.

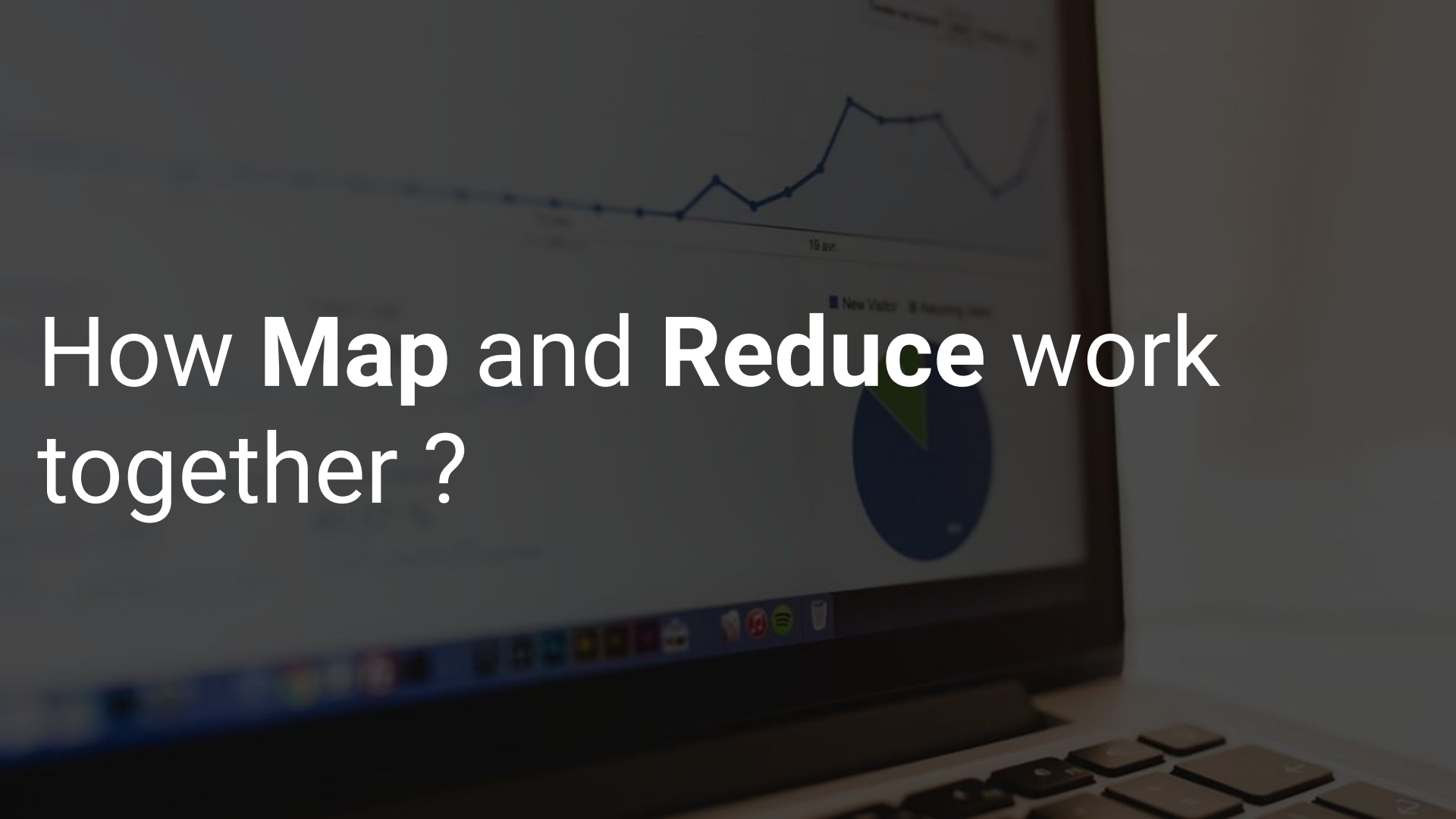
- Input given to reducer is generated by Map (intermediate output)
- Key / Value pairs provided to reduce are sorted by key

**Reduce processing**:

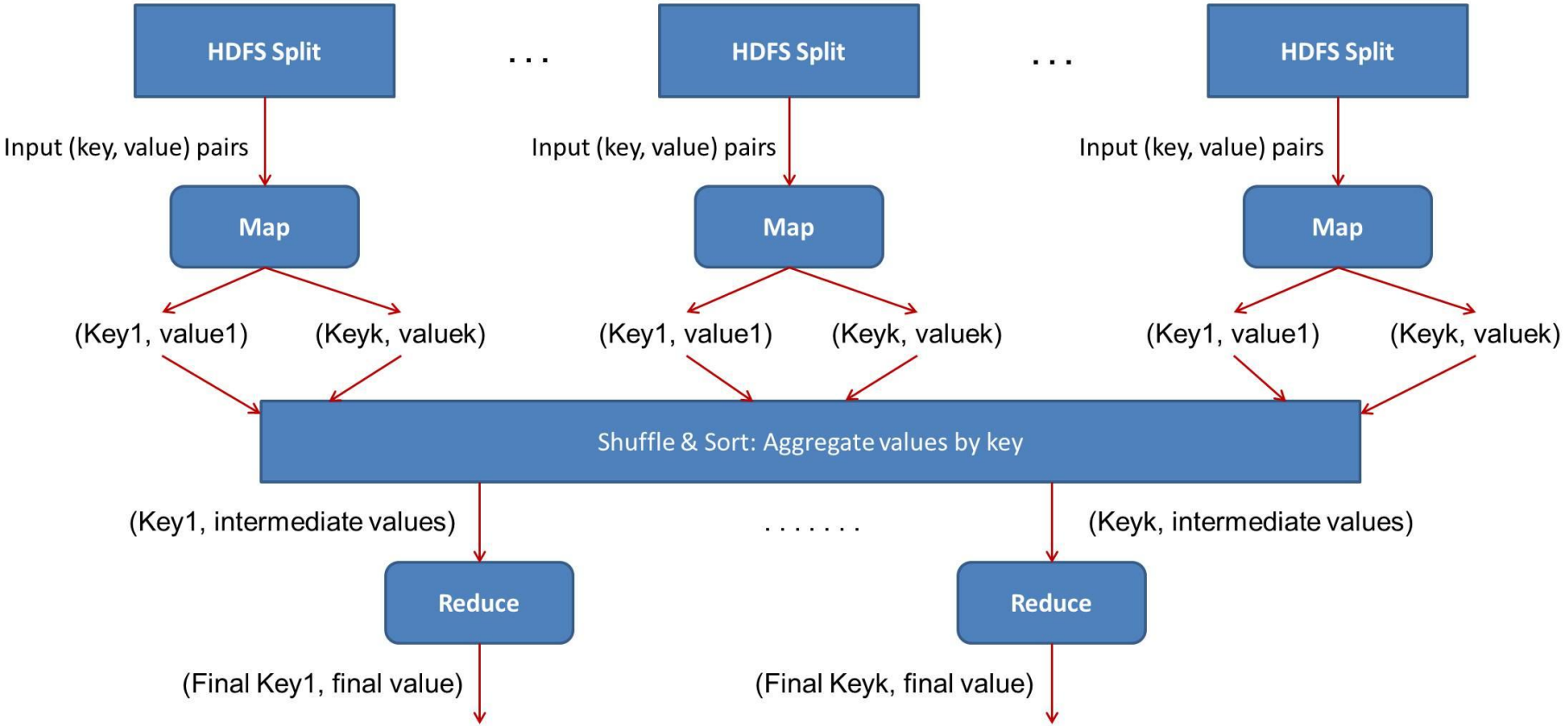
- A function defined by user – Here also user can write custom business logic and get the final output.
- Iterator supplies the values for a given key to the Reduce function.

**Reduce produces a final list of key/value pairs**:

- An output of Reduce is called Final output.
- It can be a different type from input pair.
- An output of Reduce is stored in HDFS.



# How **Map** and **Reduce** work together ?

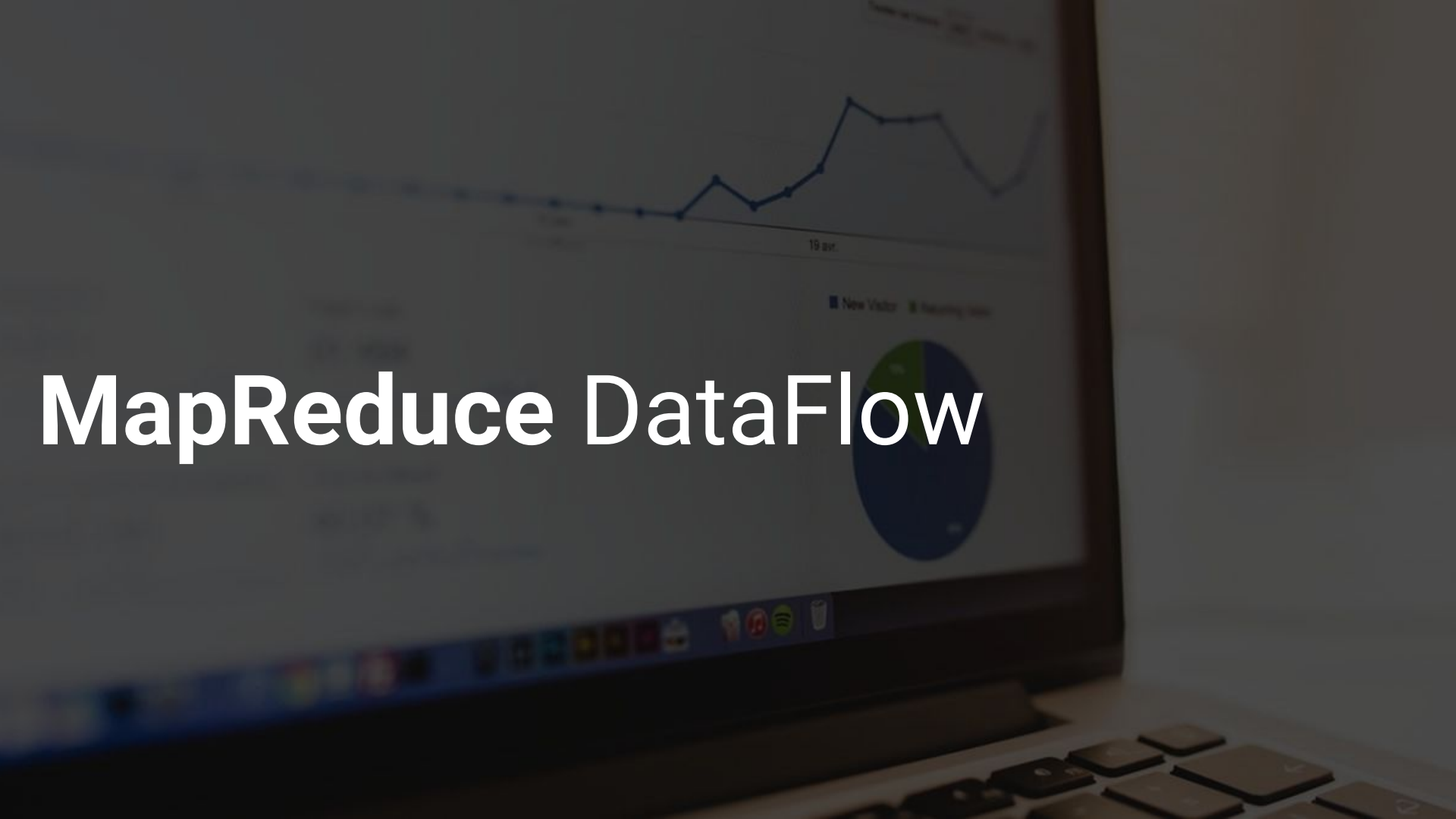

HDFS Split
HDFS Split
HDFS Split
Input (key, value) pairs
Input (key, value) pairs
Input (key, value) pairs
Map
Map
Map
(Key1, value1)
(Keyk, valuek)
(Key1, value1)
(Keyk, valuek)
(Key1, value1)
(Keyk, valuek)
Shuffle & Sort: Aggregate values by key
(Key1, intermediate values)
(Keyk, intermediate values)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

Input data given to mapper is processed through user defined function written at mapper.

All the required complex business logic is implemented at the mapper level so that heavy processing is done by the mapper in parallel as the number of mappers is much more than the number of reducers.

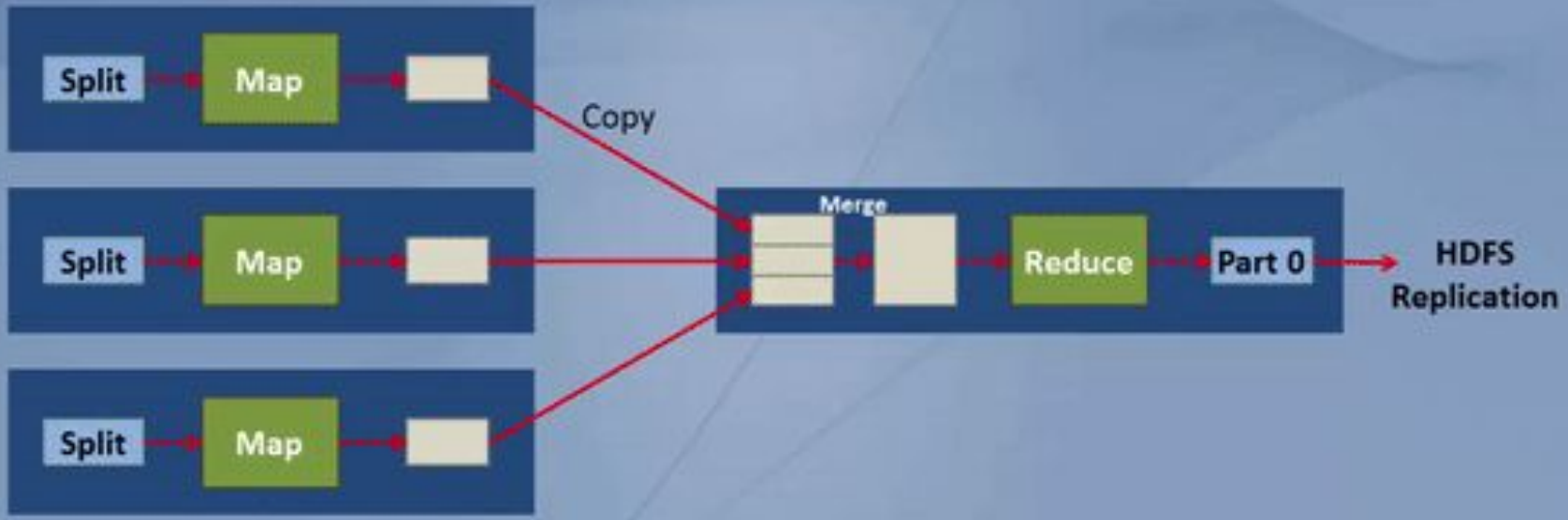
Mapper generates an output which is intermediate data and this output goes as input to reducer.

This intermediate result is then processed by user defined function written at reducer and final output is generated. Usually, in reducer very light processing is done. This final output is stored in HDFS and replication is done as usual.



# **MapReduce** DataFlow

# MapReduce Data Flow



As seen from the diagram of mapreduce workflow in Hadoop, the square block is a slave. There are 3 slaves in the figure. On all 3 slaves mappers will run, and then a reducer will run on any 1 of the slave. For simplicity of the figure, the reducer is shown on a different machine but it will run on mapper node only.

Let us now discuss the map phase:

An input to a mapper is 1 block at a time. (Split = block by default)

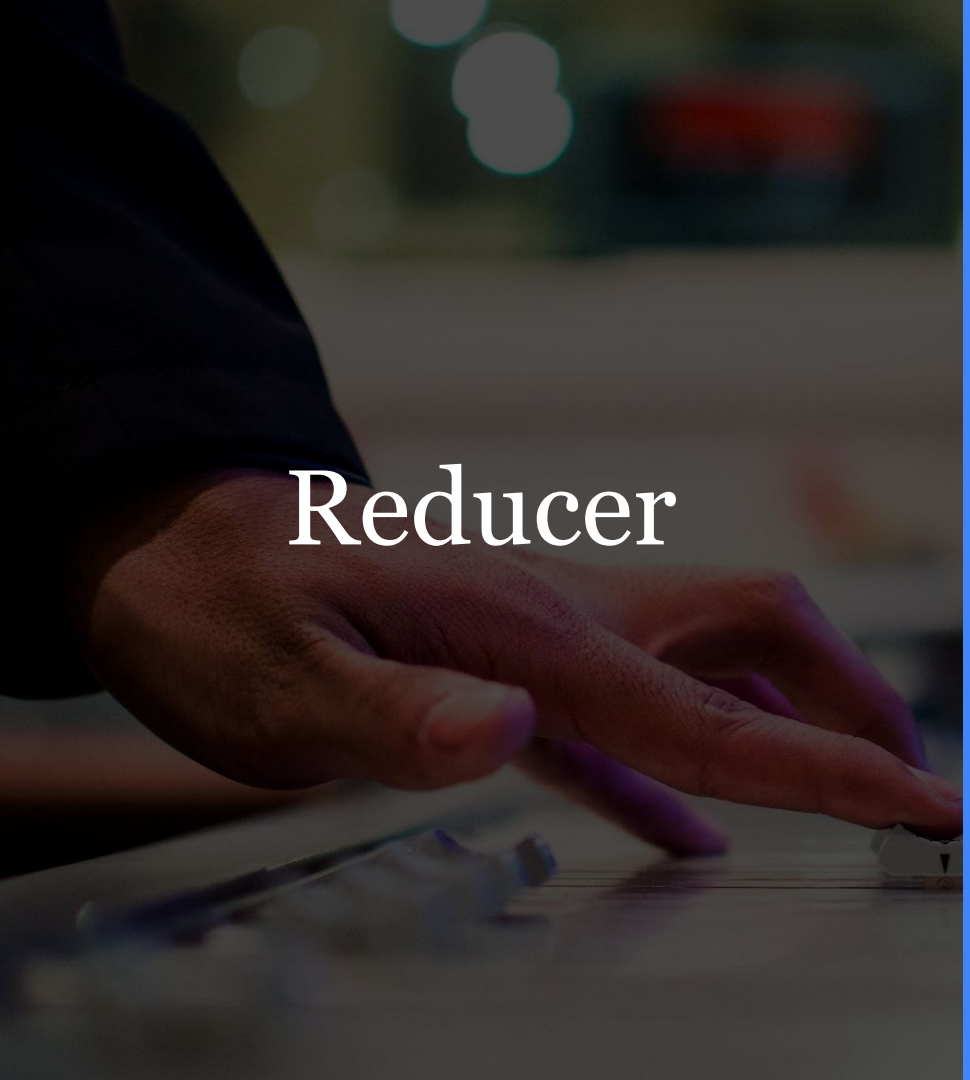
An output of mapper is written to a local disk of the machine on which mapper is running. Once the map finishes, this intermediate output travels to reducer nodes (node where reducer will run).

Reducer is the second phase of processing where the user can again write his custom business logic. Hence, an output of reducer is the final output written to HDFS.

By default on a slave, 2 mappers run at a time which can also be increased as per the requirements. It depends again on factors like datanode hardware, block size, machine configuration etc. We should not increase the number of mappers beyond the certain limit

# Mapper

**Mapper** in Hadoop **Mapreduce** writes the output to the local disk of the machine it is working. This is the temporary data. An output of mapper is also called intermediate output. All mappers are writing the output to the local disk. As First mapper finishes, data (output of the mapper) is traveling from mapper node to reducer node. Hence, this movement of output from mapper node to reducer node is called **shuffle**.



# Reducer

**Reducer** is also deployed on any one of the datanode only. An output from all the mappers goes to the reducer. All these outputs from different mappers are merged to form input for the reducer. This input is also on local disk. Reducer is another processor where you can write custom business logic. It is the second stage of the processing. Usually to reducer we write aggregation, summation etc. type of functionalities. Hence, Reducer gives the final output which it writes on **HDFS**.

**Map** and **reduce** are the stages of processing. They run one after other. After all, mappers complete the processing, then only reducer starts processing.

1 block is present at 3 different locations by default, but framework allows only 1 mapper to process 1 block. So only 1 mapper will be processing 1 particular block out of 3 replicas. The output of every mapper goes to every reducer in the cluster i.e every reducer receives input from all the mappers. Hence, framework indicates reducer that whole data has processed by the mapper and now reducer can process the data.

An output from mapper is partitioned and filtered to many partitions by the partitioner. Each of this partition goes to a reducer based on some conditions. Hadoop works with key value principle i.e **mapper** and **reducer** gets the input in the form of key and value and write output also in the same form.


Questions?