Hadoop

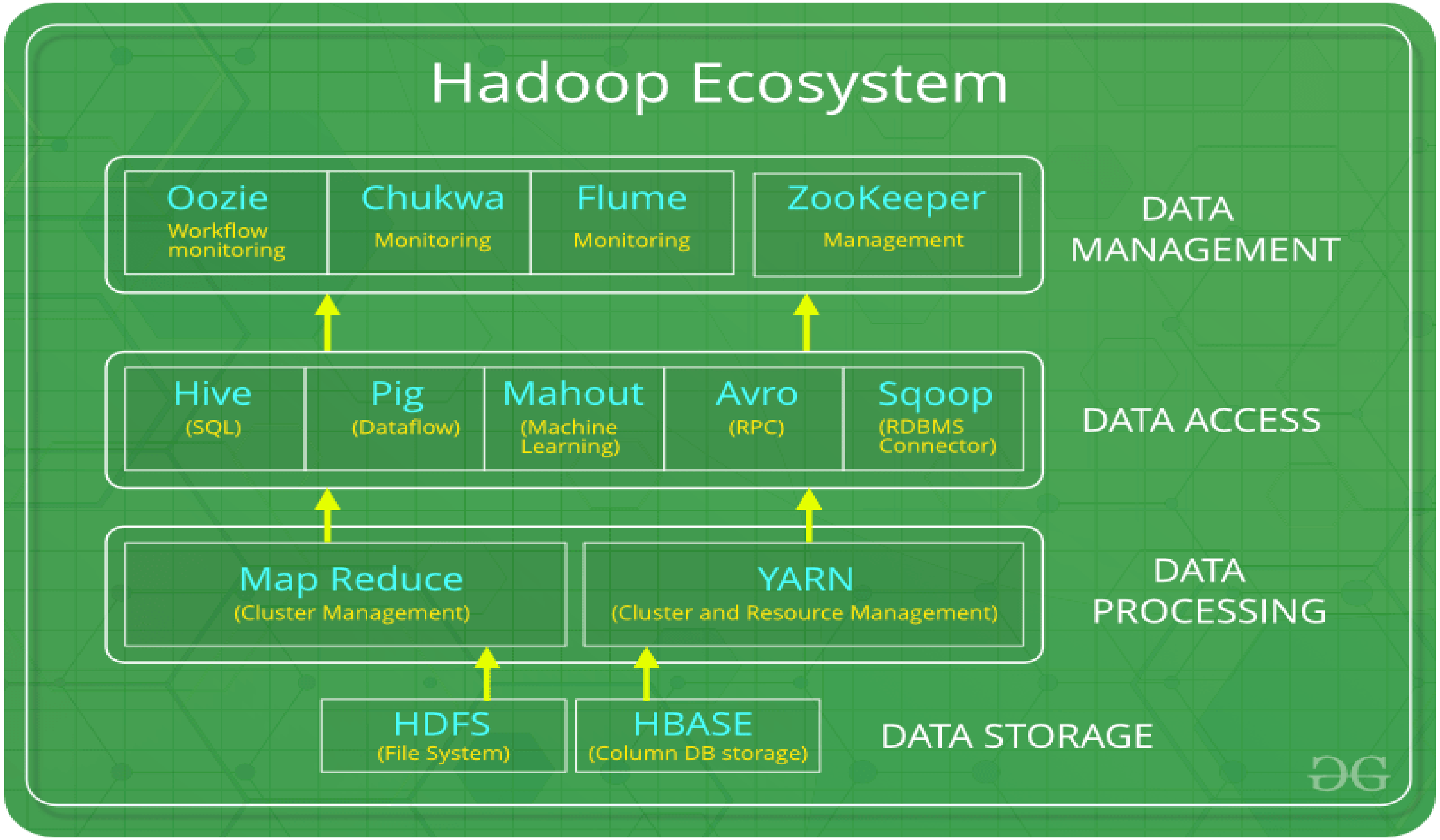
Apache Hadoop

Framework to handle Big Data

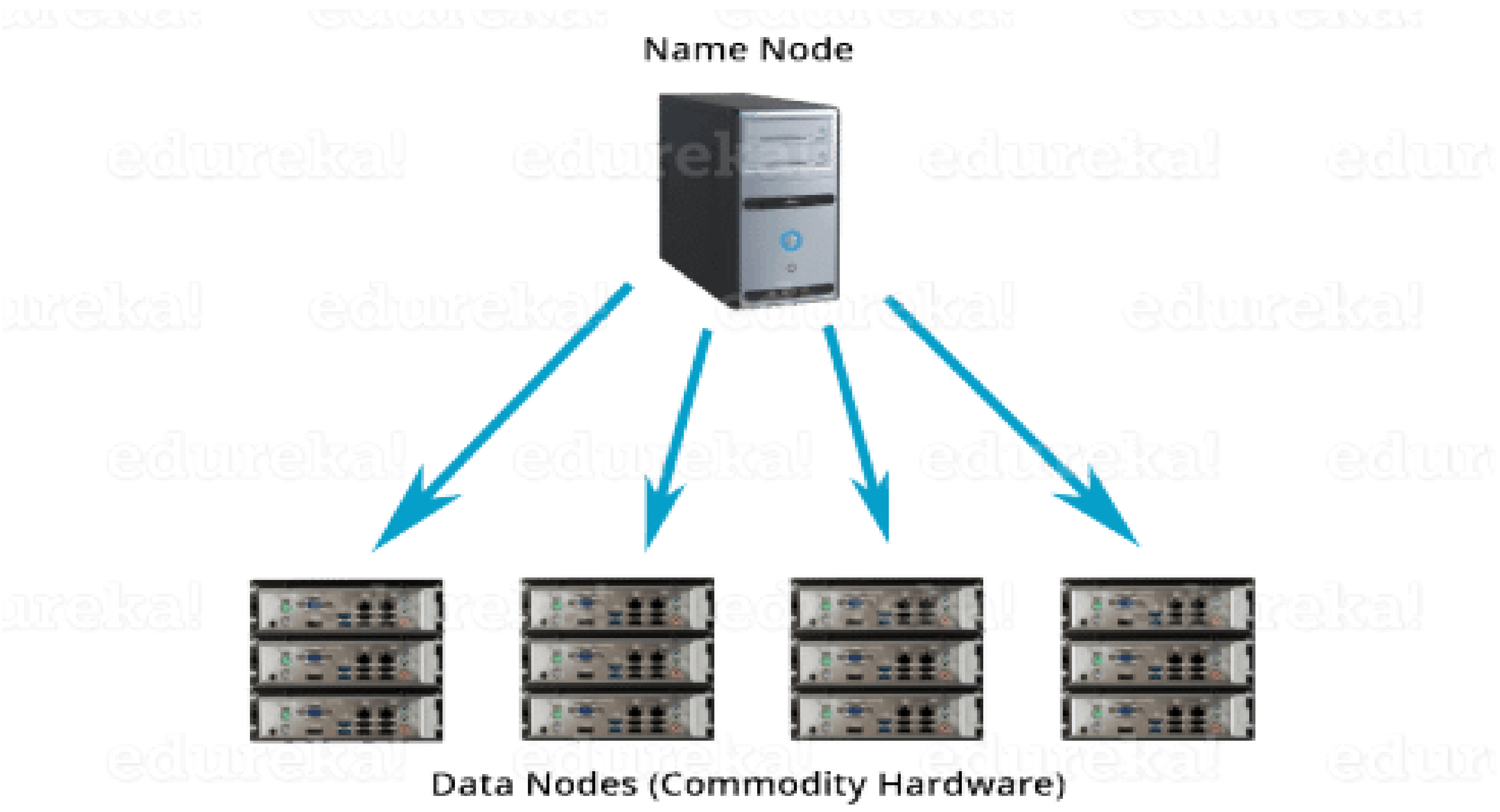
Three characteristics:

- Distributed
- Scalable
- Reliable

Hadoop Suite of components



HDFS



Name Node

Manages the filesystem namespace

Maintains the filesystem tree and the metadata for all the files and directories in the tree

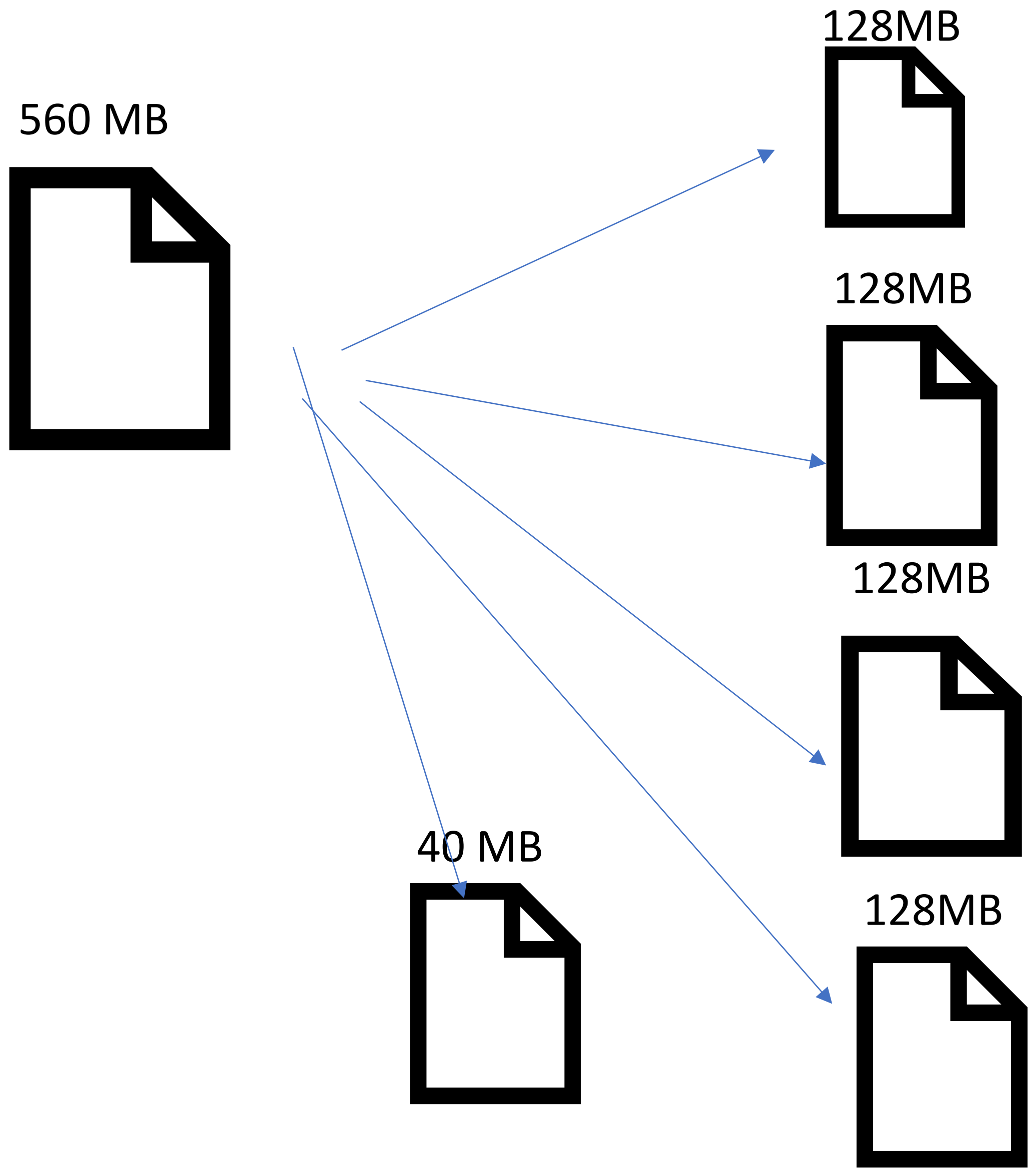
Stored in RAM

knows the data nodes on which all the blocks for a given file are located

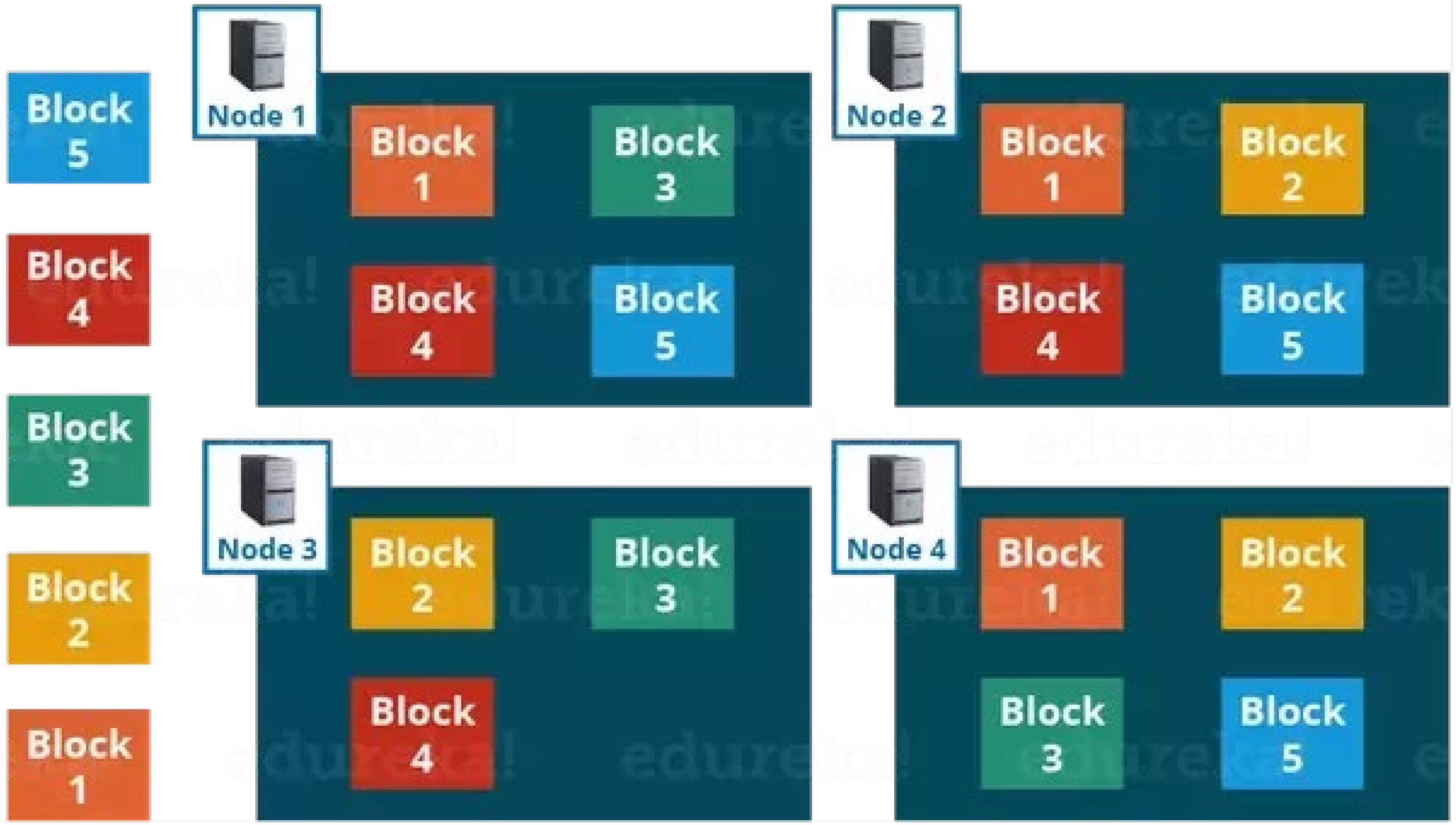
Data Node

- Data nodes store the actual data

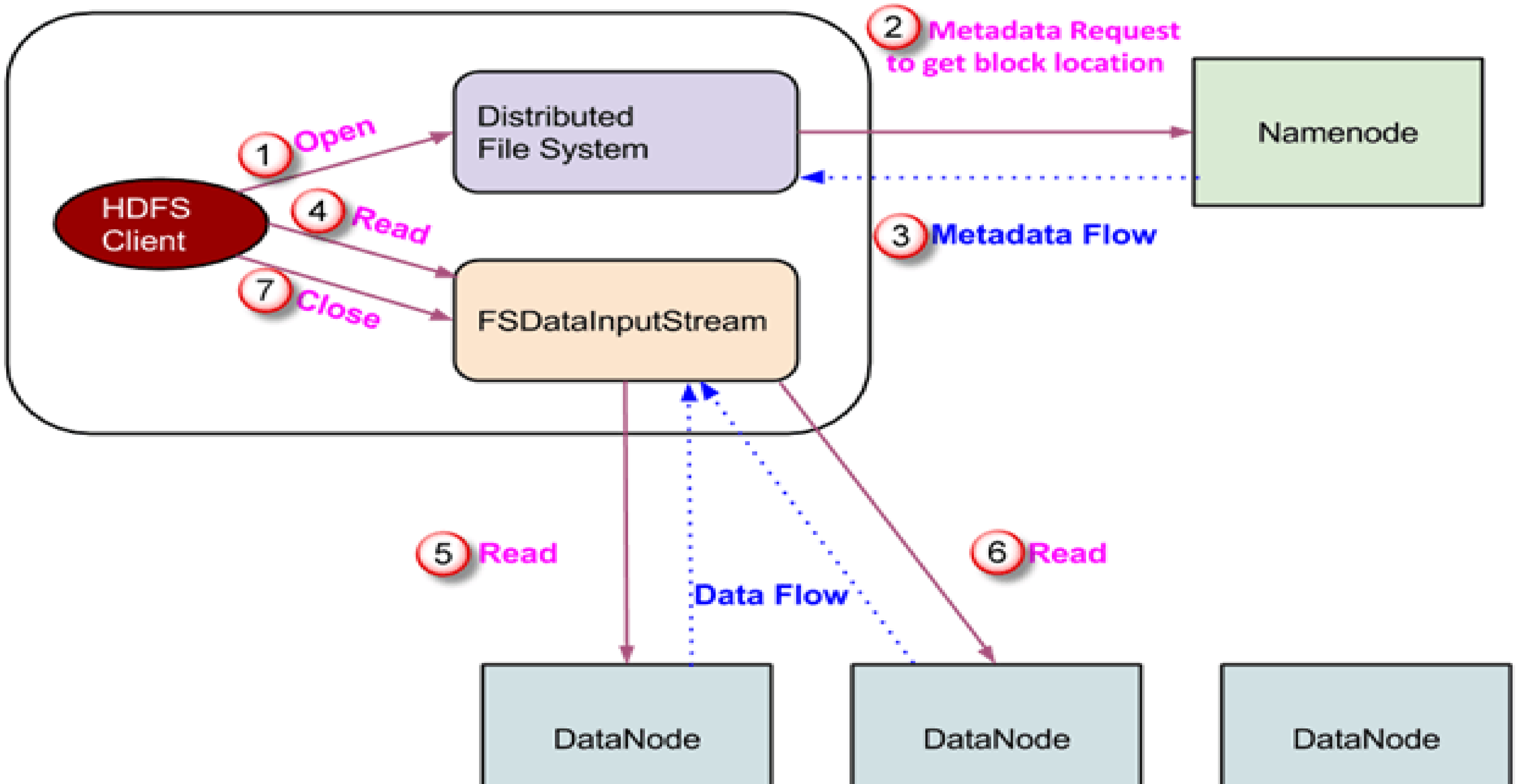
HDFS blocks



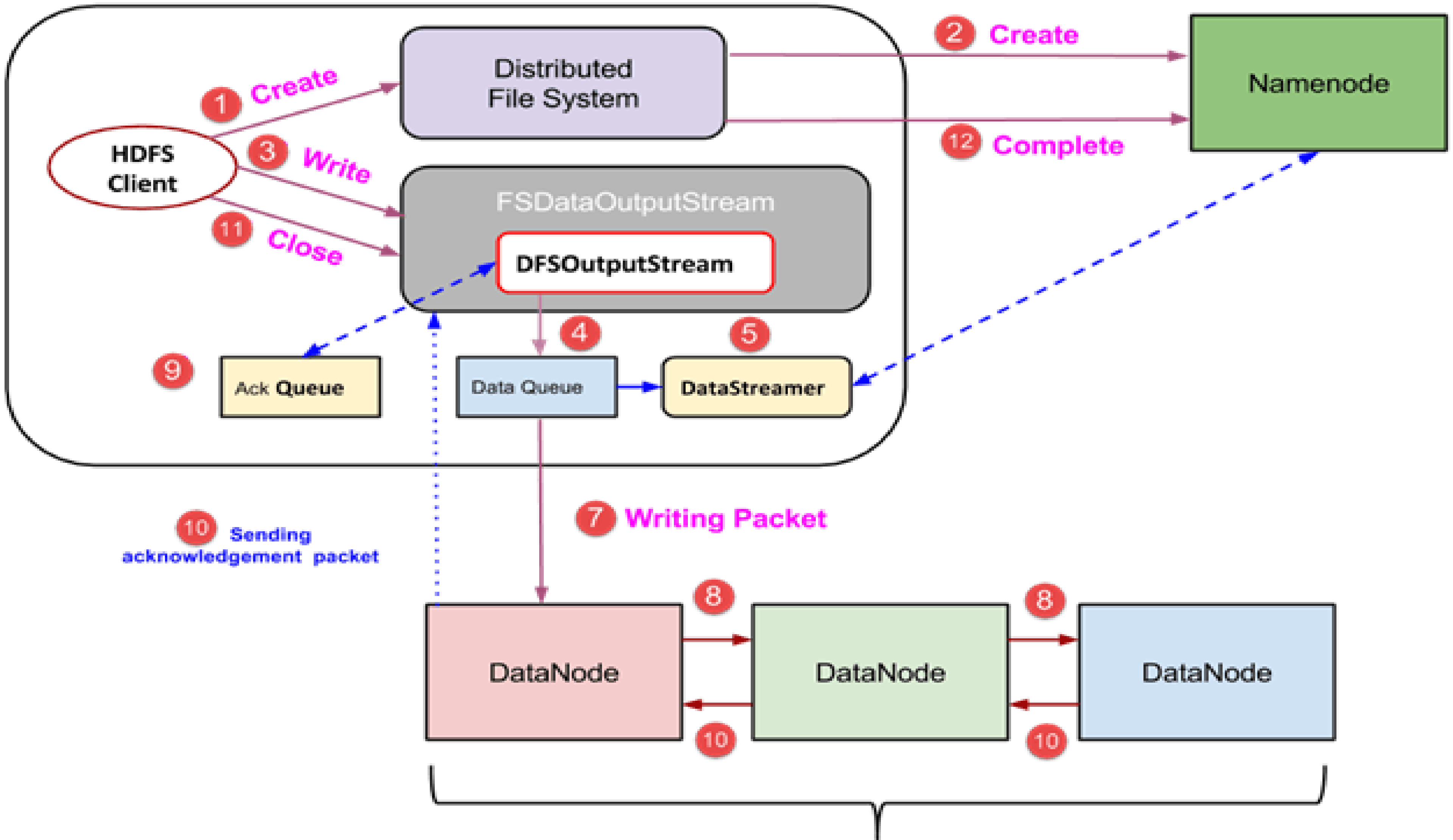
Blocks Replication

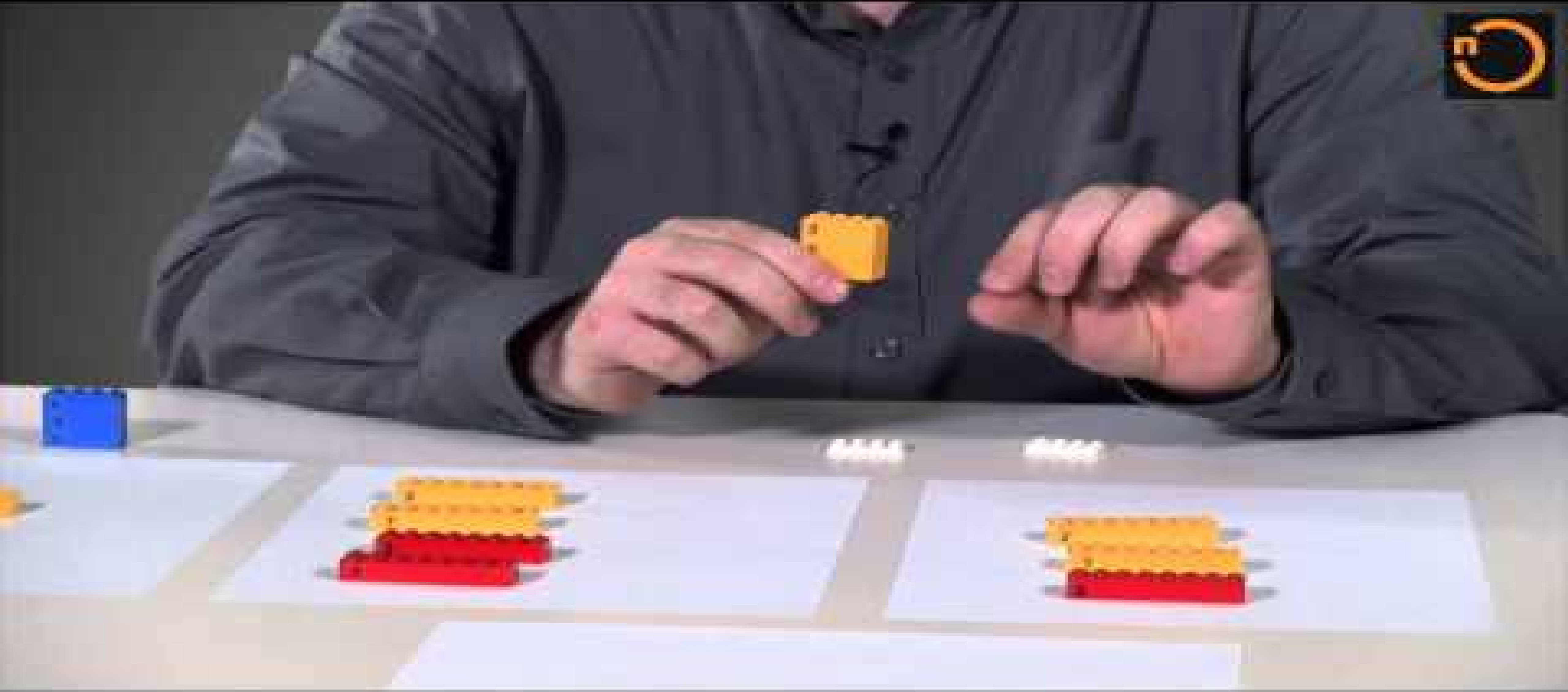


HDFS: File reading



HDFS: File Writing





MapReduce

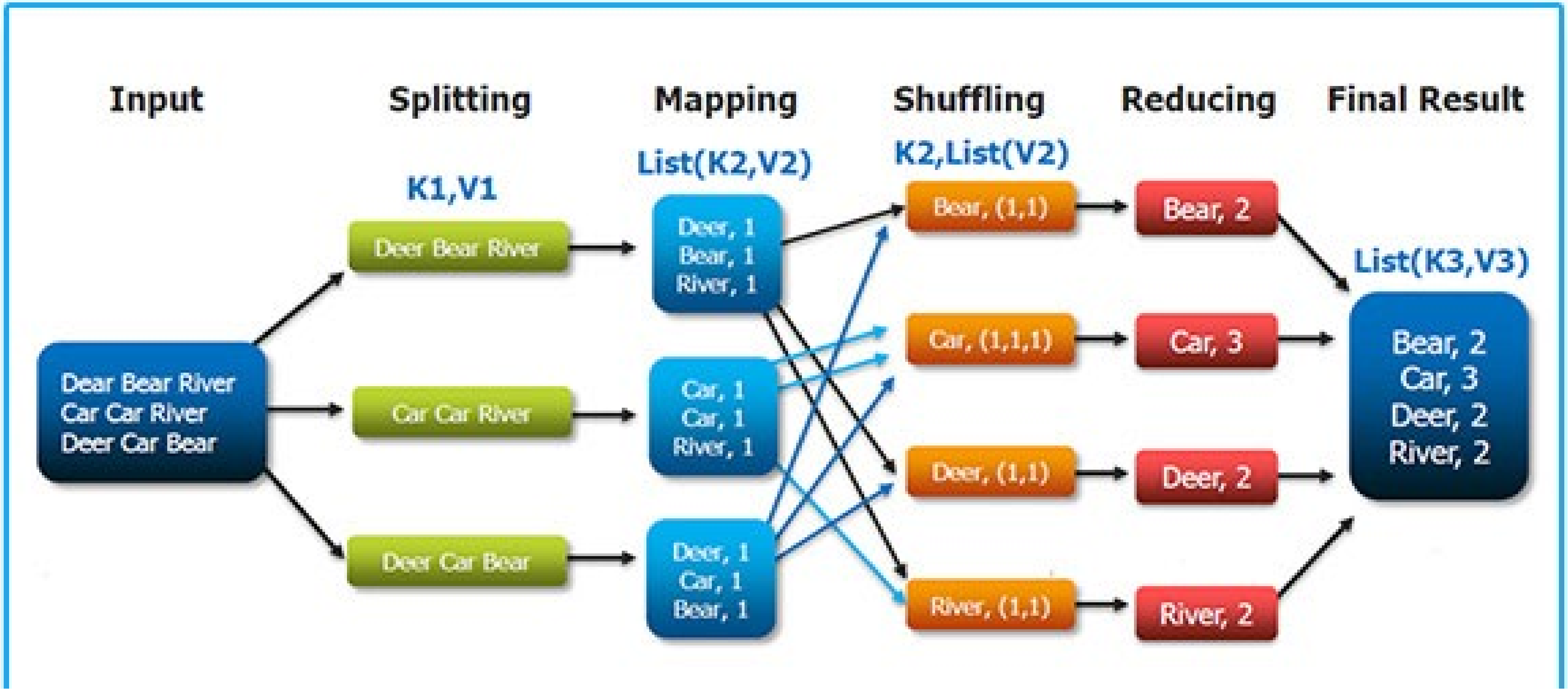
Programming Paradigm

- To help solve Big data problems
- Specifically sorting intensive jobs or disk read intensive

You would have to code two functions:

- Mapper: converts input into key-value pairs
- Reducer: Aggregate all values for keys

Map-Reduce Problem: Word Count

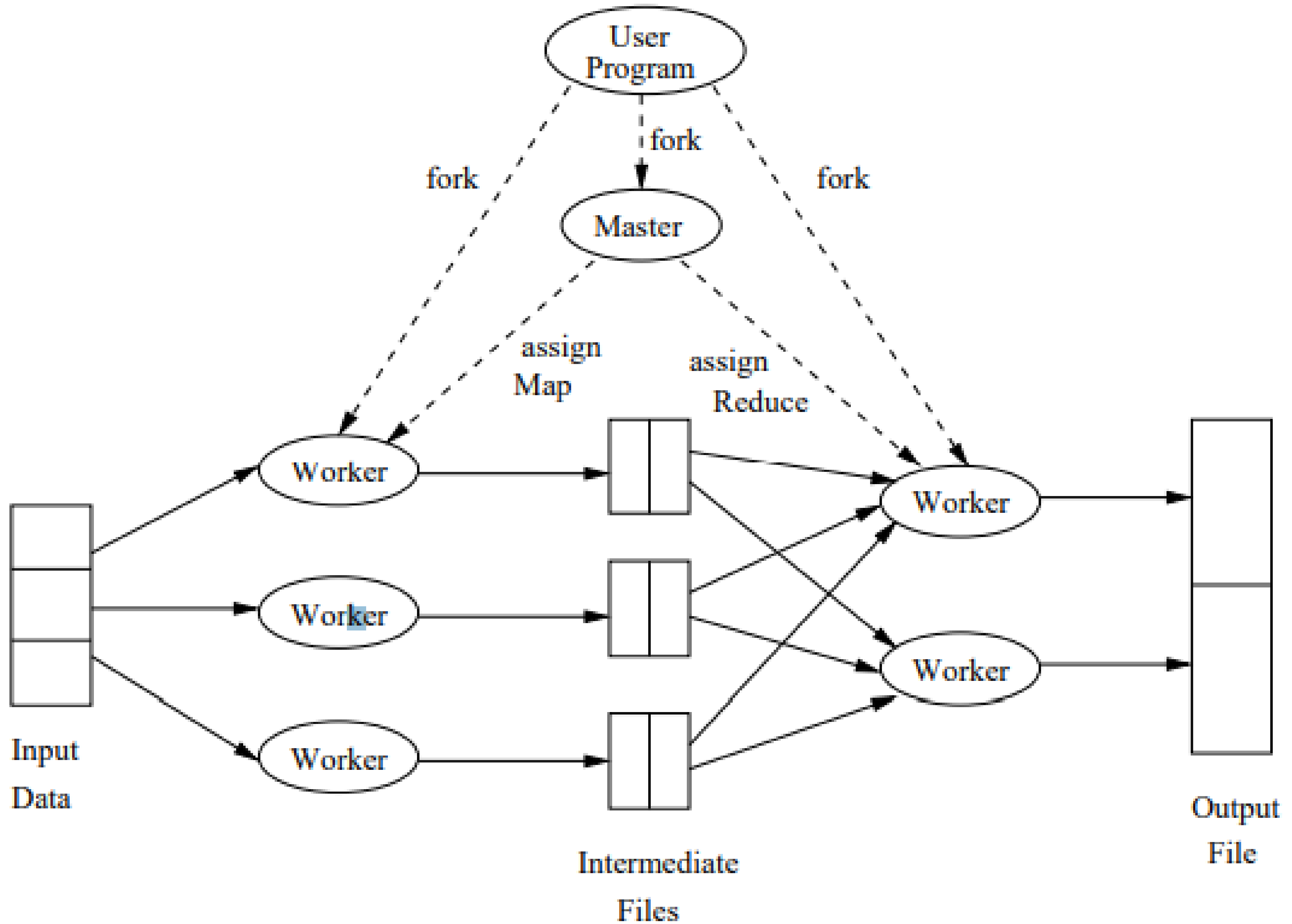


Map-Reduce: Environment

Map-Reduce environment takes care of:

- Partitioning the input data
- Scheduling the program's execution across a set of machines
- Performing the **group by key** step
- Handling machine failures
- Managing required inter-machine communication

MapReduce Programming Paradigm: Overview



Data Flow

- **Input and final output are stored on a distributed file system (FS):**
  - Scheduler tries to schedule map tasks "close" to physical storage location of input data
- **Intermediate results are stored on local FS of Map and Reduce workers**
- **Output is often input to another MapReduce task**

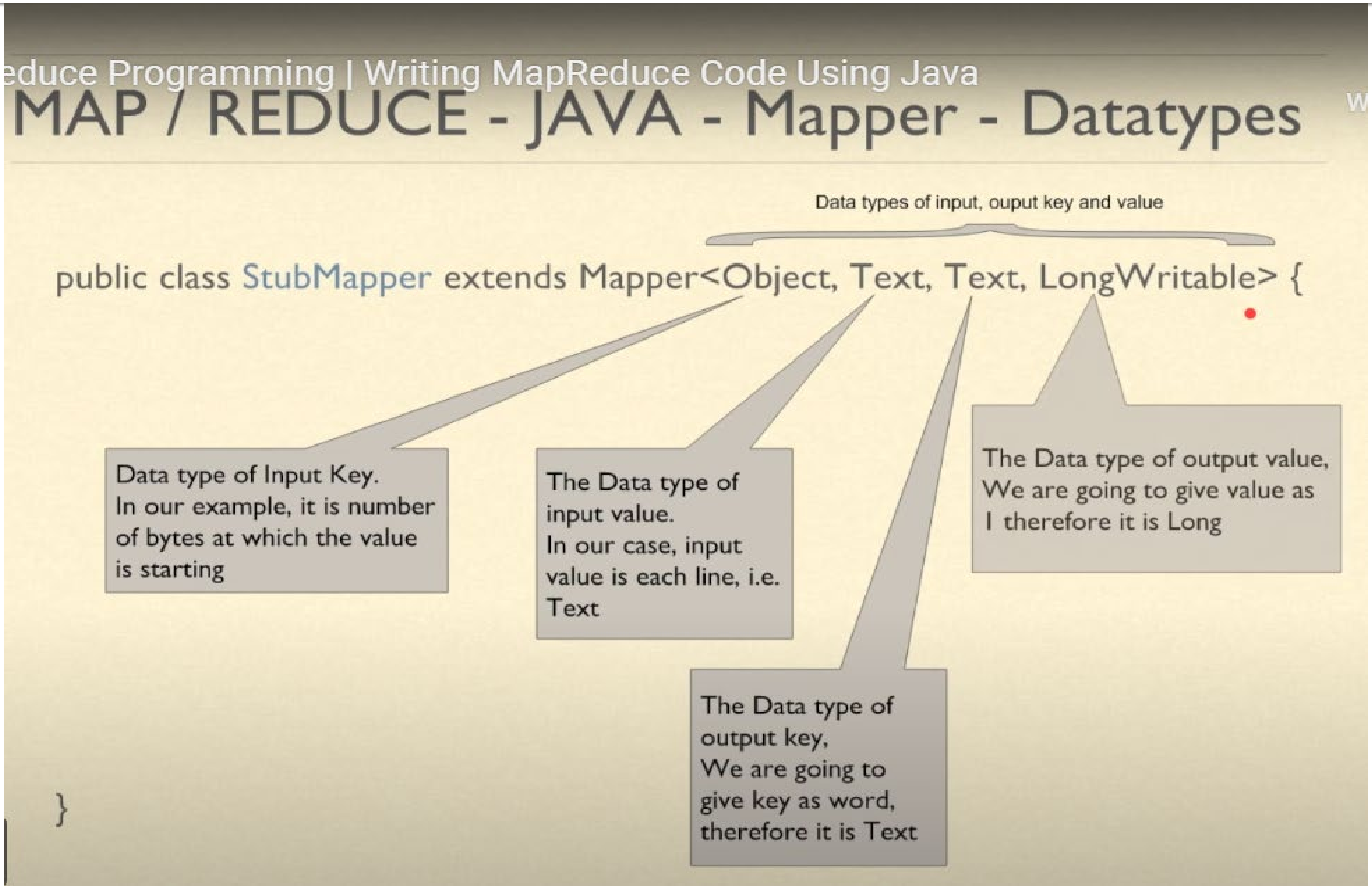
Coordination: Master

- Master node takes care of coordination:
  - **Task status:** (idle, in-progress, completed)
  - **Idle tasks** get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its *R* intermediate files, one for each reducer
  - Master pushes this info to reducers
- Master pings workers periodically to detect failures

Dealing with Failures

- **Map worker failure**
  - Map tasks completed or in-progress at worker are reset to idle
  - Reduce workers are notified when task is rescheduled on another worker
- **Reduce worker failure**
  - Only in-progress tasks are reset to idle
  - Reduce task is restarted
- **Master failure**
  - MapReduce task is aborted and client is notified

StubMapper.java



educe Programming | Writing MapReduce Code Using Java

MAP / REDUCE - JAVA - Mapper - Datatypes

Data types of input, ouput key and value

```
public class StubMapper extends Mapper<Object, Text, Text, LongWritable> {

}
```

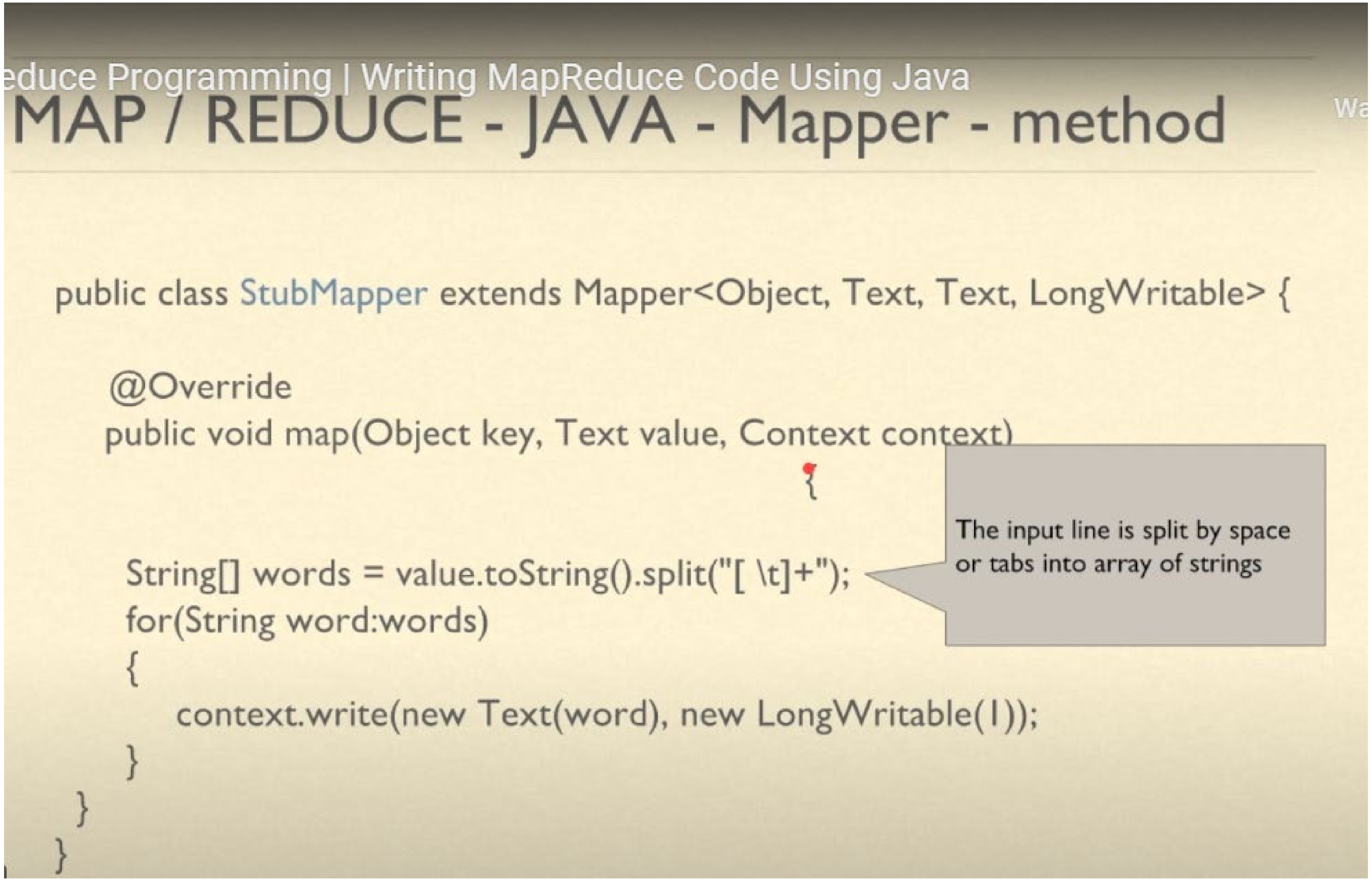
Data type of Input Key.
In our example, it is number of bytes at which the value is starting

The Data type of input value.
In our case, input value is each line, i.e. Text

The Data type of output key,
We are going to give key as word, therefore it is Text

The Data type of output value,
We are going to give value as 1 therefore it is Long

StubMapper.java



educe Programming | Writing MapReduce Code Using Java

MAP / REDUCE - JAVA - Mapper - method

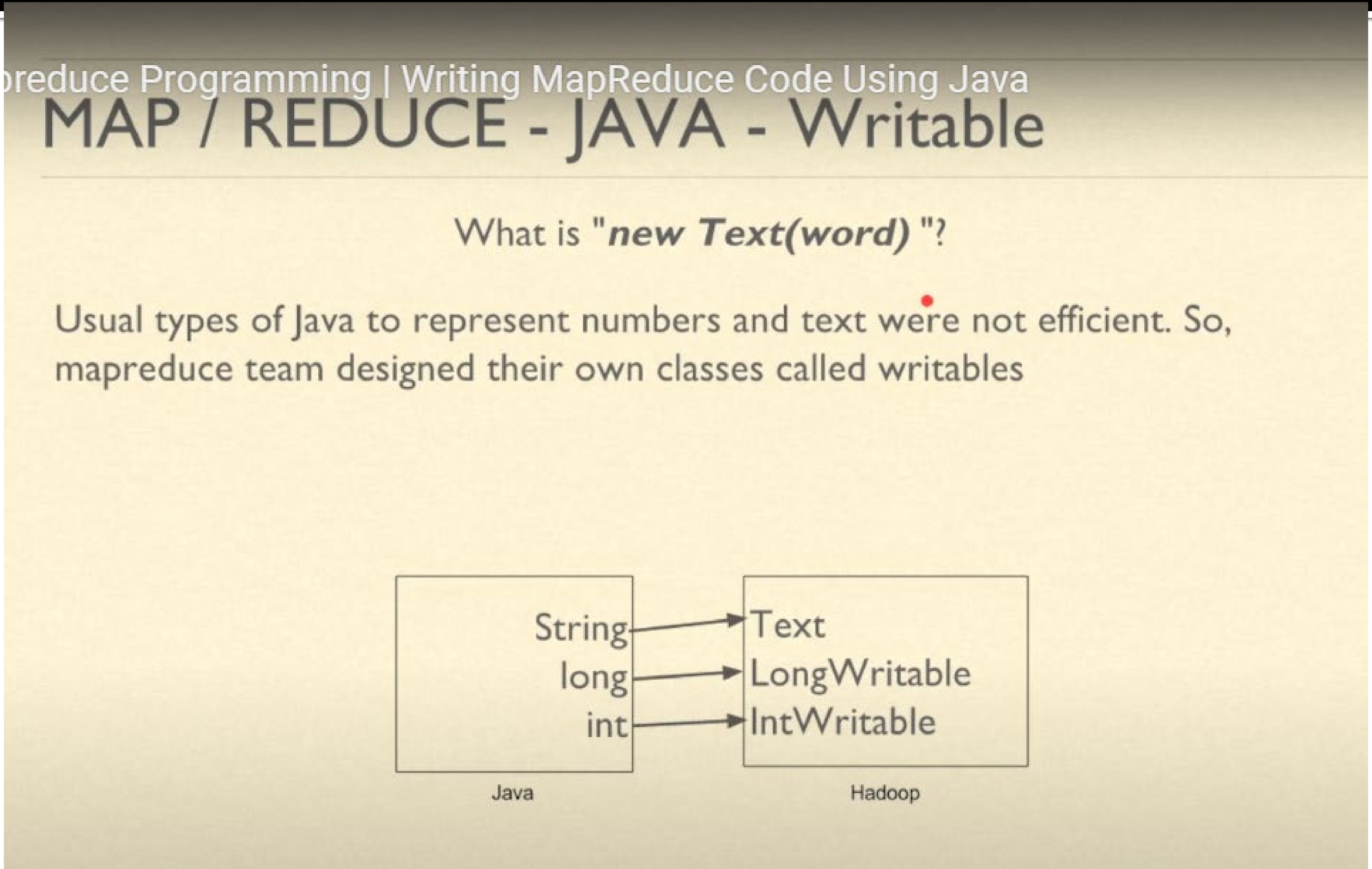
```
public class StubMapper extends Mapper<Object, Text, Text, LongWritable> {

    @Override
    public void map(Object key, Text value, Context context)
    {

      String[] words = value.toString().split("[ \t]+");
      for(String word:words)
      {
          context.write(new Text(word), new LongWritable(1));
      }
    }
}
```

The input line is split by space or tabs into array of strings

StubMapper.java



reduce Programming | Writing MapReduce Code Using Java

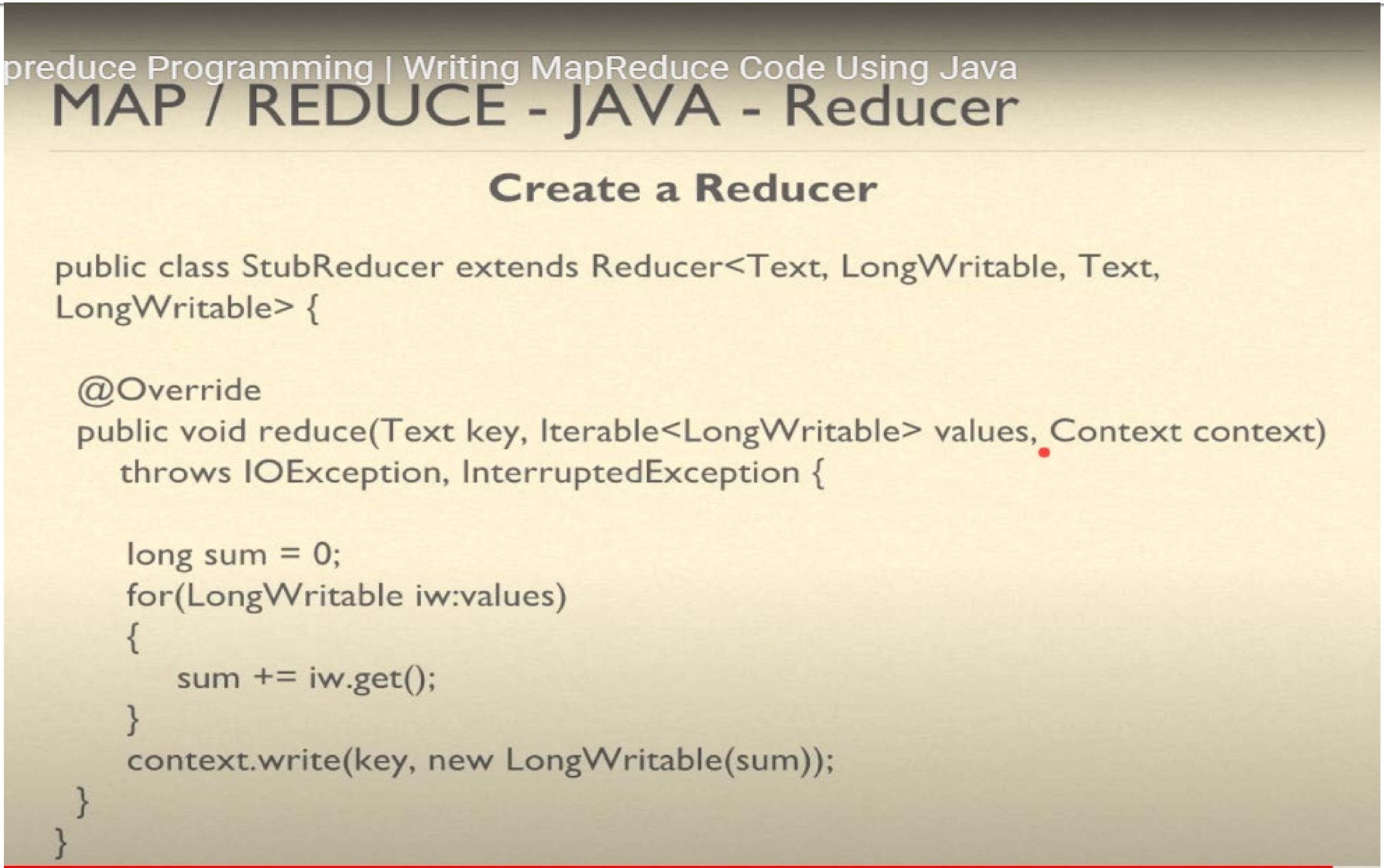
MAP / REDUCE - JAVA - Writable

What is "***new Text(word)*** "?

Usual types of Java to represent numbers and text were not efficient. So, mapreduce team designed their own classes called writables

$$\begin{tikzcd}
\text{String} \arrow[r] & \text{Text} \\
\text{long} \arrow[r] & \text{LongWritable} \\
\text{int} \arrow[r] & \text{IntWritable} \\
\text{Java} & \text{Hadoop}
\end{tikzcd}$$

StubReducer.java



preduce Programming | Writing MapReduce Code Using Java

MAP / REDUCE - JAVA - Reducer

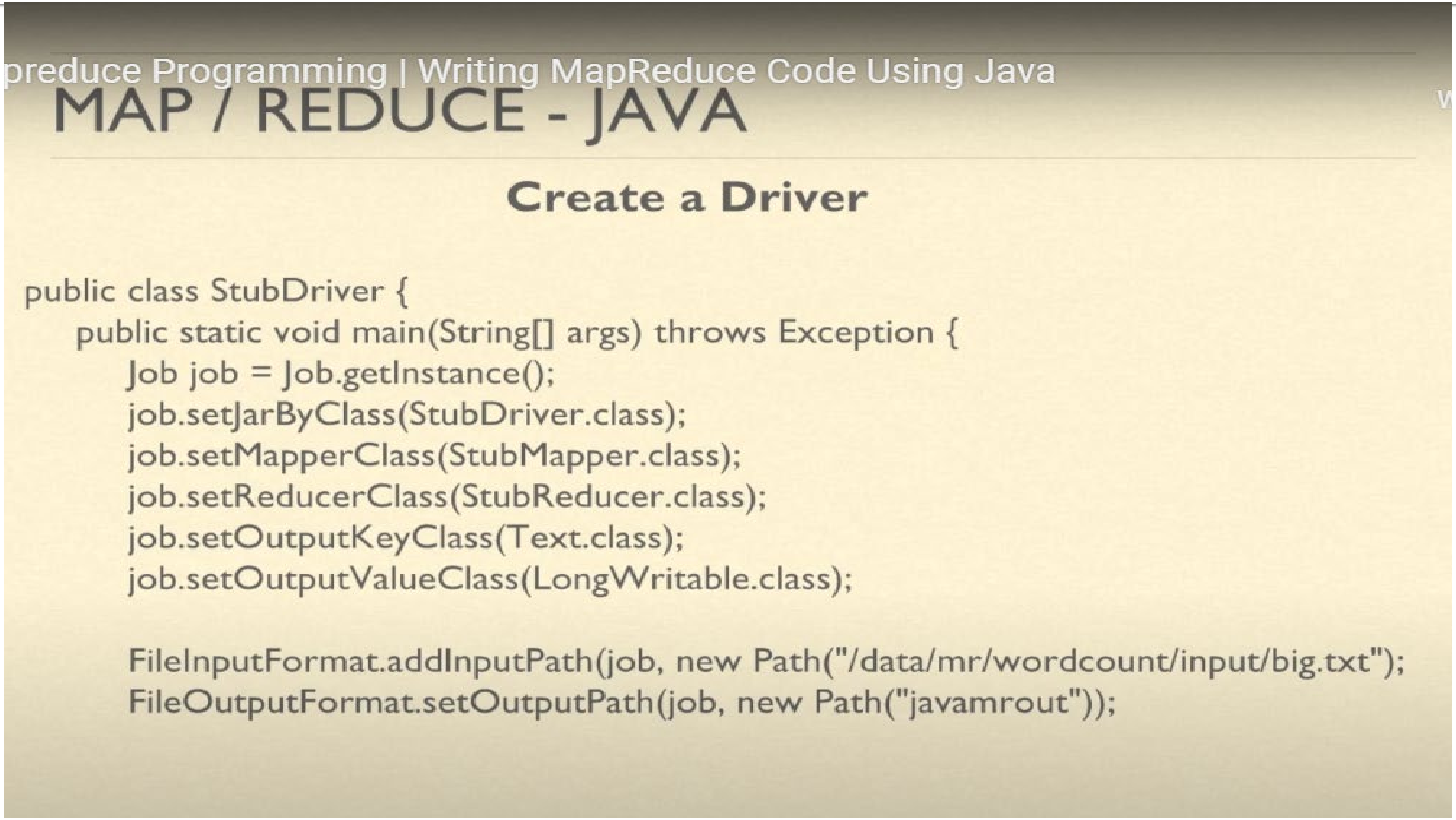
Create a Reducer

```
public class StubReducer extends Reducer<Text, LongWritable, Text,
LongWritable> {

  @Override
  public void reduce(Text key, Iterable<LongWritable> values, Context context)
      throws IOException, InterruptedException {

        long sum = 0;
        for(LongWritable iw:values)
        {
            sum += iw.get();
        }
        context.write(key, new LongWritable(sum));
  }
}
```

StubDriver.java


preduce Programming | Writing MapReduce Code Using Java

MAP / REDUCE - JAVA

Create a Driver

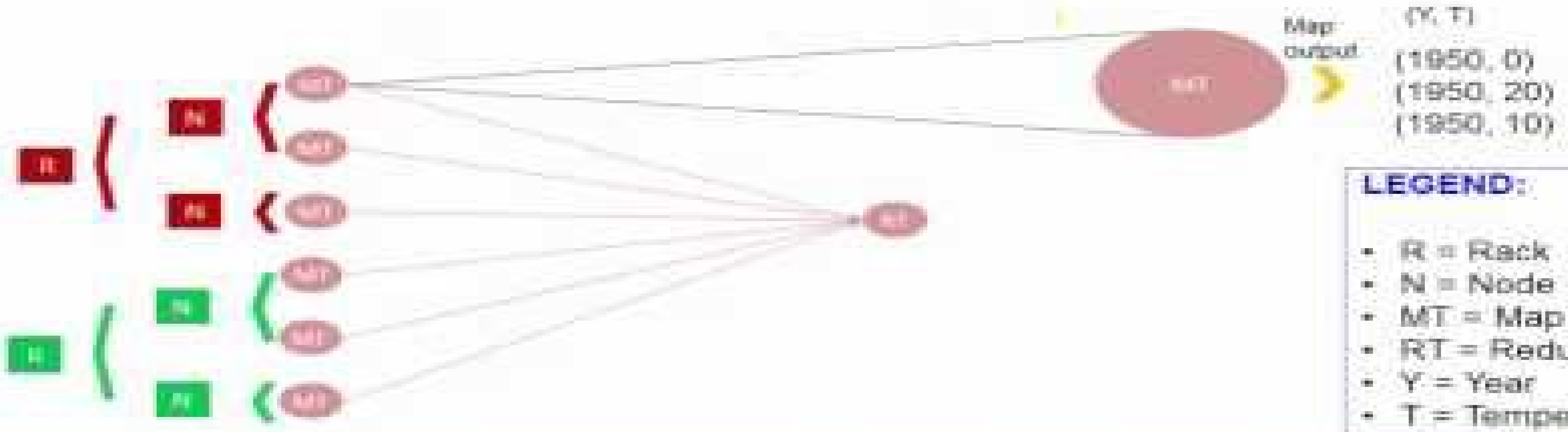
```
public class StubDriver {
    public static void main(String[] args) throws Exception {
        Job job = Job.getInstance();
        job.setJarByClass(StubDriver.class);
        job.setMapperClass(StubMapper.class);
        job.setReducerClass(StubReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(LongWritable.class);

        FileInputFormat.addInputPath(job, new Path("/data/mr/wordcount/input/big.txt"));
        FileOutputFormat.setOutputPath(job, new Path("javamrout"));
```

Visit PragProg.com/screencasts

Combiner

- Mapper generates {Year, Temperature} as (key, value). The objective of MapReduce job is to get the highest temperature each year



LEGEND:

- R = Rack
- N = Node
- MT = Map Task
- RT = Reduce Task
- Y = Year
- T = Temperature