Big Data Analytics: Hadoop Paradigms for Managing Large Scale Dataset

Dr. Nitesh Funde,

Department of Artificial Intelligence,

Sardar Vallabhbhai National Institute of Technology, Surat

FDP: Artificial Intelligence and Data Science: Foundations, Pedagogy, Tools, and Emerging Research Trends

(15 December-19 December, 2025)

Content

- Introduction to Hadoop
- Comparison of Bigdata
- MapReduce
- Examples
- Hadoop Ecosystem

Introduction to Hadoop

- Internet started growing 2001.
- Before that data stored in rows and columns (Structured data)
- More solutions were available for structured data. As size is limited, thus it can store in single storage device required and single processing elements to process.
- But as internet starts growing, different types of data change. As we use such as image, video, text, email are available , thus volume of data increased and thus to process , it needs new solutions.
- Doug cutting started working in 2002 and it made public in 2008 by Yahoo and Apache foundation in 2012.

Comparison

Small Data

- Mostly structured
- Store in MB, GB, TB
- Increase gradually
- Locally present, centralized data
- Oracle, SQL server

Big Data

- Unstructured, Volume, Variety, Velocity
- Store in PB, EB
- Increase exponentially
- Globally present, distributed
- Hadoop, Spark
- More Data usually beats better algorithms

HD File System- Data nodes

- Rather than storing data on single node, Store data on different nodes distributedly and then we can access parallelly.
- Hadoop is an ecosystem / framework for storing and processing large-scale data in a distributed manner. The Hadoop ecosystem includes other tools like Hive, Pig, HBase, Spark.
- It has two main components.
  - 1) HDFS (Hadoop Distributed file system)-For distributed storage of data
  - 2) MapReduce: For distributed data processing. MapReduce is a programming model + processing framework for writing applications that process large amounts of data in parallel across a Hadoop cluster.
- Use of Java Programming Language.

Introduction to Hadoop

- Hadoop is an open source framework that allows us to store and process large data sets in a parallel and distributed manner.

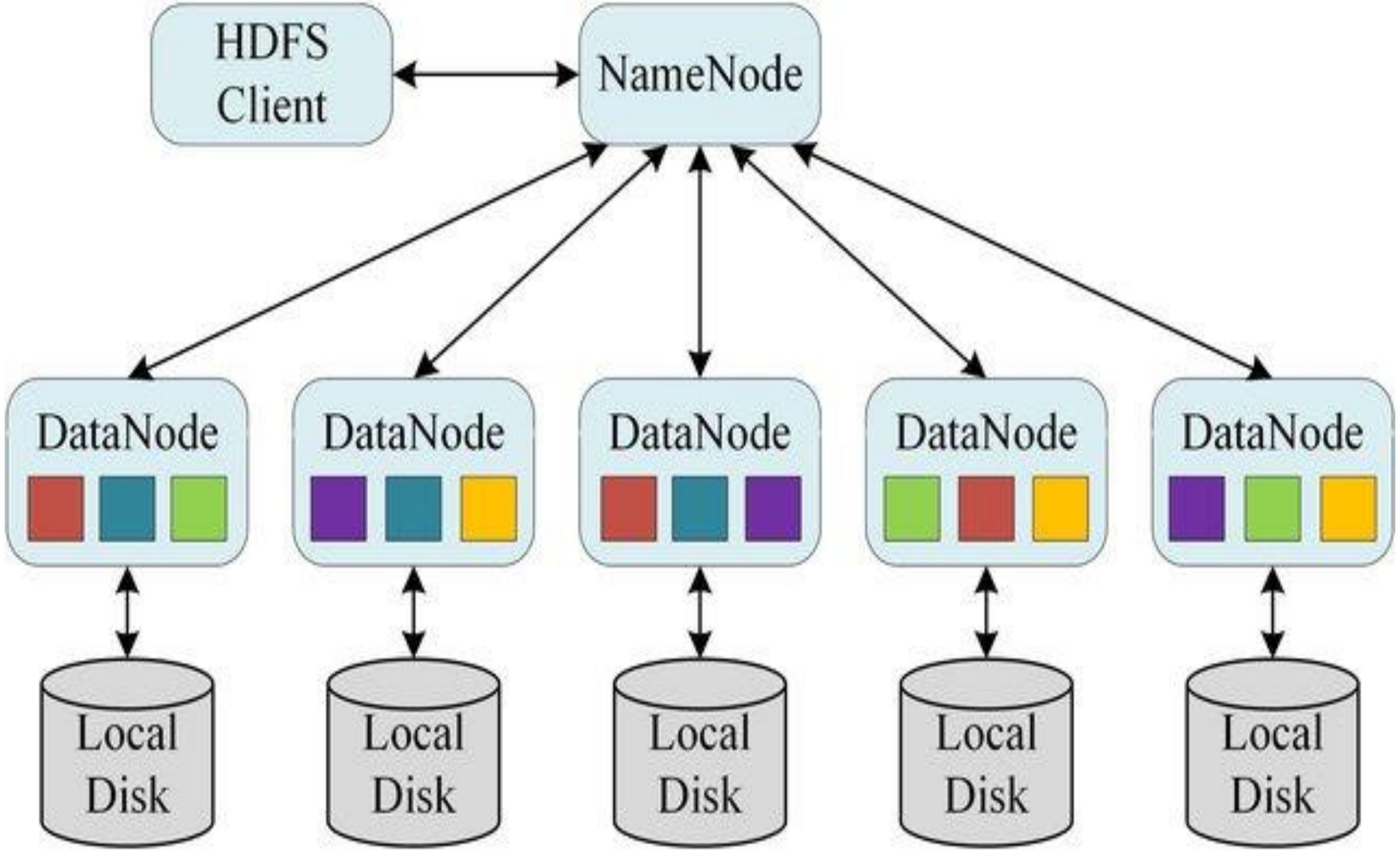
- Doug Cutting started the Hadoop project.

- Two main components, Hadoop and MapReduce

- **Hadoop Distributed file system (HDFS)** is the primary data storage system used by Hadoop applications.

- **MapReduce** is the processing unit of Hadoop.

HDFS



HDFS Client
NameNode
DataNode
DataNode
DataNode
DataNode
DataNode
Local Disk
Local Disk
Local Disk
Local Disk
Local Disk

Functions of Name Node

- It records the metadata of all files stored in the cluster, e.g. location of blocks stored, the size of the files, permissions.

- Two files associated with metatdata:
  - FsImage: It contains complete state of the file system namespace since the start of the NameNode.
  - EditLogs: It contains all the recent modifications made to the file system.

- It regularly receives a communication messages (Heart beat) and block report from all the data nodes on the cluster to ensure that DataNodes are live.

Functions of Data Node

- The actual data is stored on DataNodes.
- The DataNodes perform the low-level read and write requests from the file system's client.
- Write operation is expensive on HDFS.

HDFS

- Rack awareness
- It is a physical collection of 30-40 nodes under 1 rack.
- Data store locally, but copies store remotely.

```
     Rack 1
----------------
Node A  (Data)
Node B (Replica)
Node C
```

```
     Rack 2
----------------
Node X (Replica)
Node Y
Node Z
```

HDFS

- In a data center, servers (nodes) are organized into racks (a rack is a cabinet that holds 30–40 machines/nodes physically).
- These racks are connected to a rack switch, and racks are connected to each other via higher-level network switches.
- Rack awareness means the system (like Hadoop’s HDFS) knows which node belongs to which rack.
- This helps the system make smart decisions about where to place data blocks and their replicas to balance performance and fault tolerance.

HDFS

- Data stored locally, but copies stored remotely
- When a file is stored in HDFS, it is broken into blocks (say 128 MB each).
- Each block is saved (replicated) **multiple times** across different nodes (default: 3 copies).
- Example with replication factor = 3:
- **One copy stored locally** on the node where data is written.
- **Second copy stored on a different node in the same rack** (to reduce network traffic).
- **Third copy stored on a node in a different rack** (to ensure data safety if a whole rack fails).

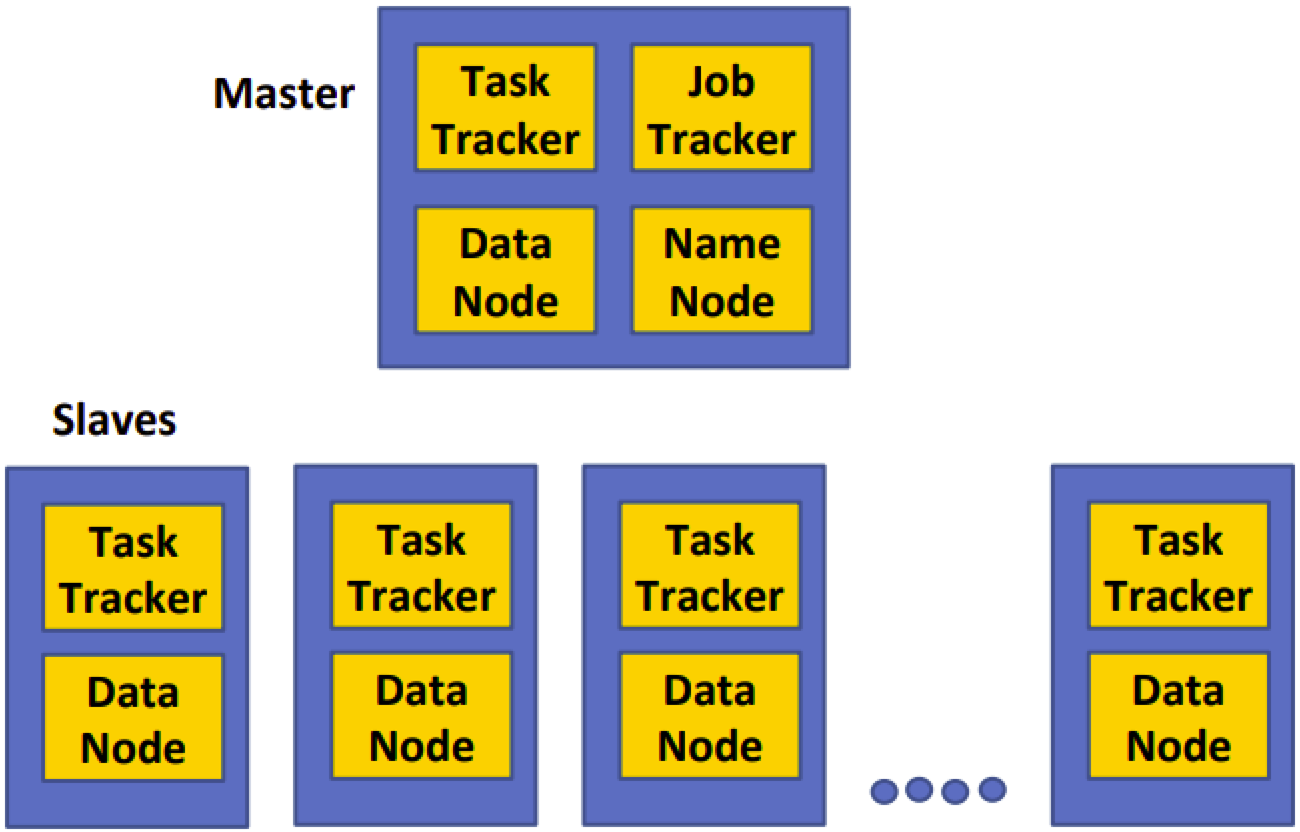
Why is this important

- Fault tolerance: If one node crashes, or even if an entire rack fails, data is still available.
- Efficiency: Keeping one copy in the same rack reduces cross-rack network traffic (which is more expensive).
- Balance: Spreading copies across racks ensures high availability and disaster recovery

MapReduce

- MapReduce perform the processing of large data sets in a distributed and parallel manner.
- MapReduce consists of main two tasks- map and reduce
- Two main daemons of MapReduce- Job Tracker and Task Tracker

Map Reduce


Master
Task Tracker
Job Tracker
Data Node
Name Node
Slaves
Task Tracker
Data Node
Task Tracker
Data Node
Task Tracker
Data Node
Task Tracker
Data Node

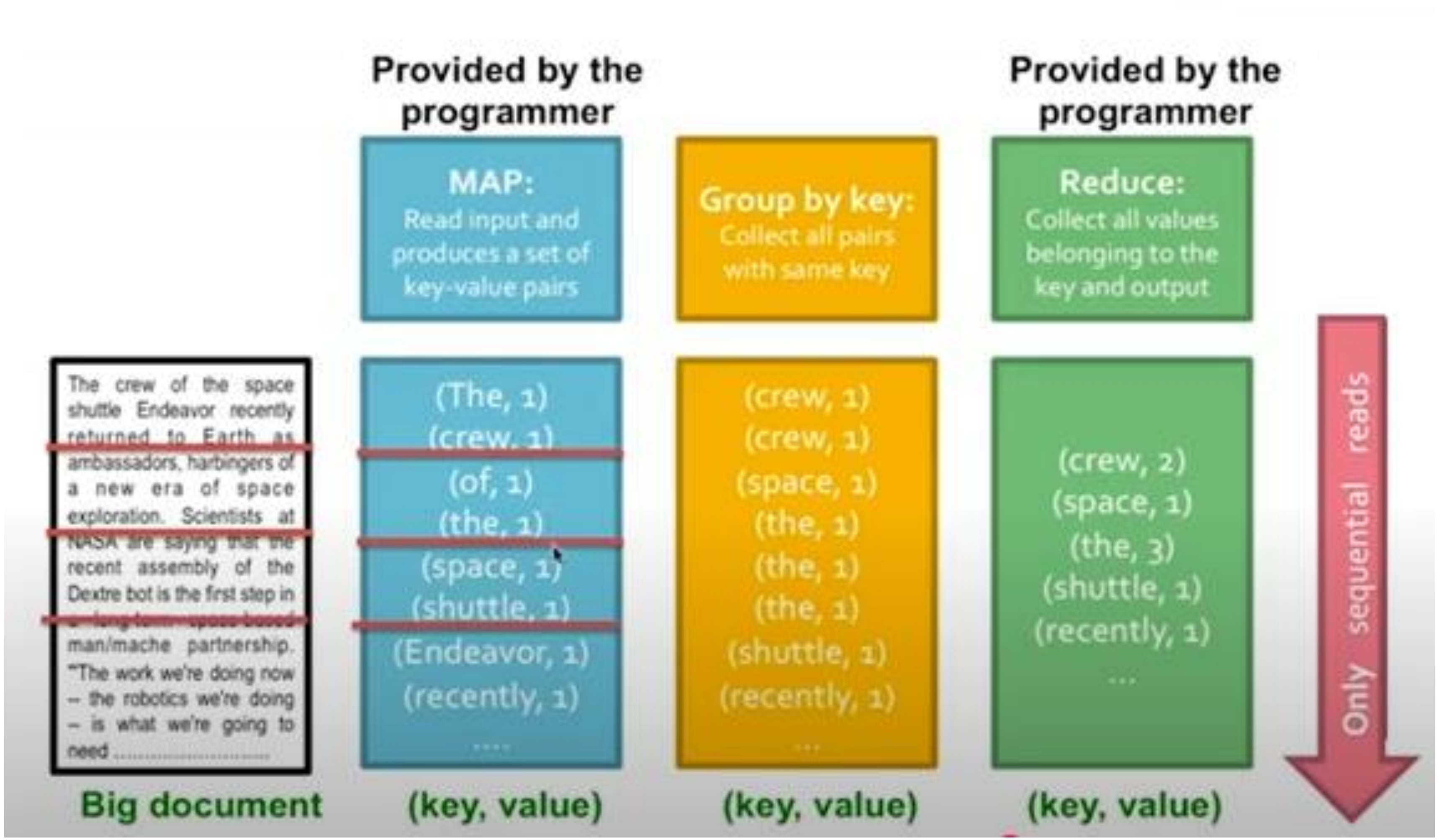
Job Tracker

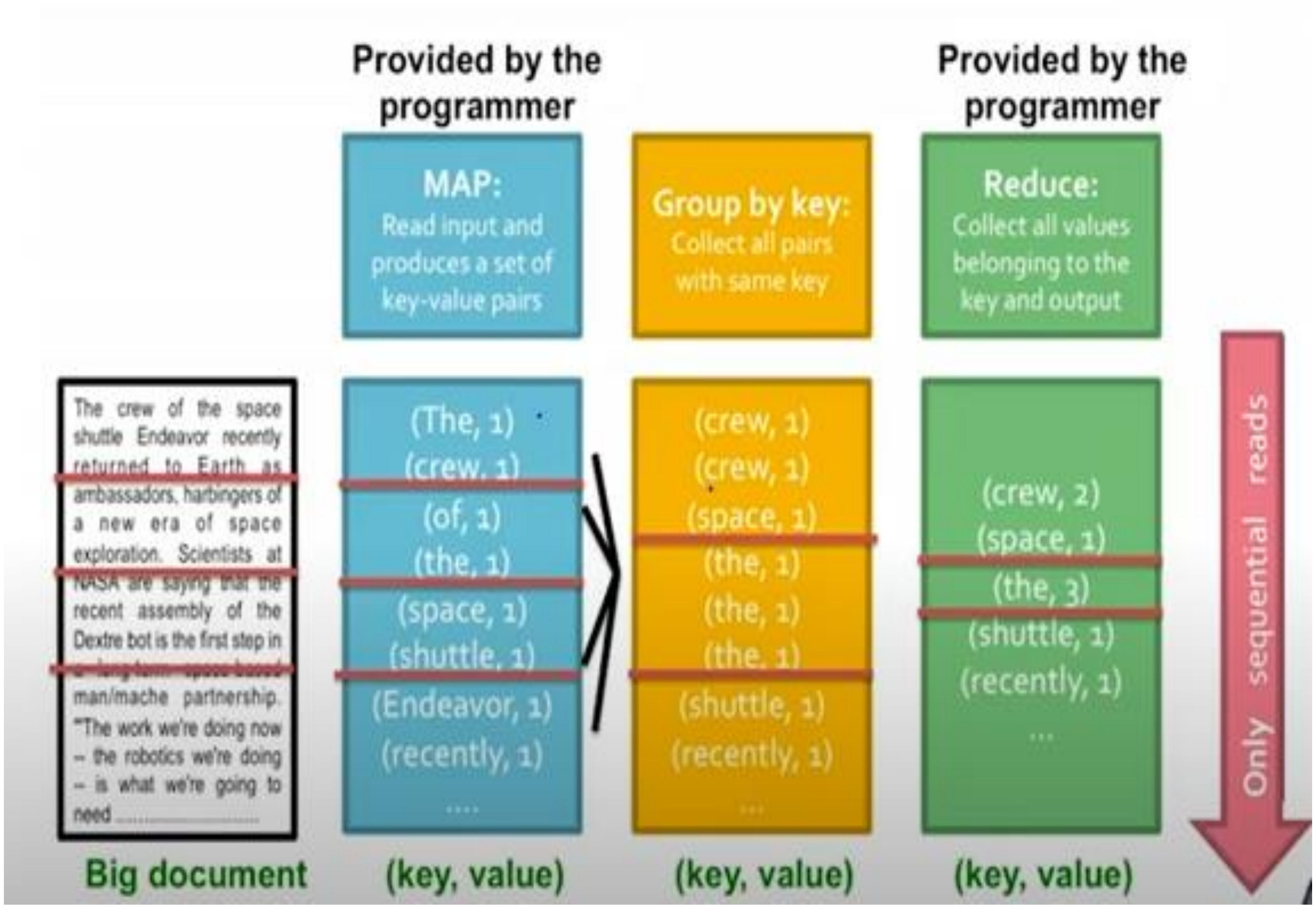
- The JobTracker is the master daemon that runs on the NameNode (master node) in a Hadoop cluster.
- Its main role is to manage and monitor MapReduce jobs submitted by clients.
- It decides how to split the job into smaller tasks and assigns them to different worker nodes.
- It keeps track of task progress (map tasks and reduce tasks) and reassigns them if a worker fails.
- It schedules tasks close to where the data resides (data locality) to minimize network usage. The JobTracker maintains overall job status (running, failed, completed) and communicates this back to the client.

Task Tracker

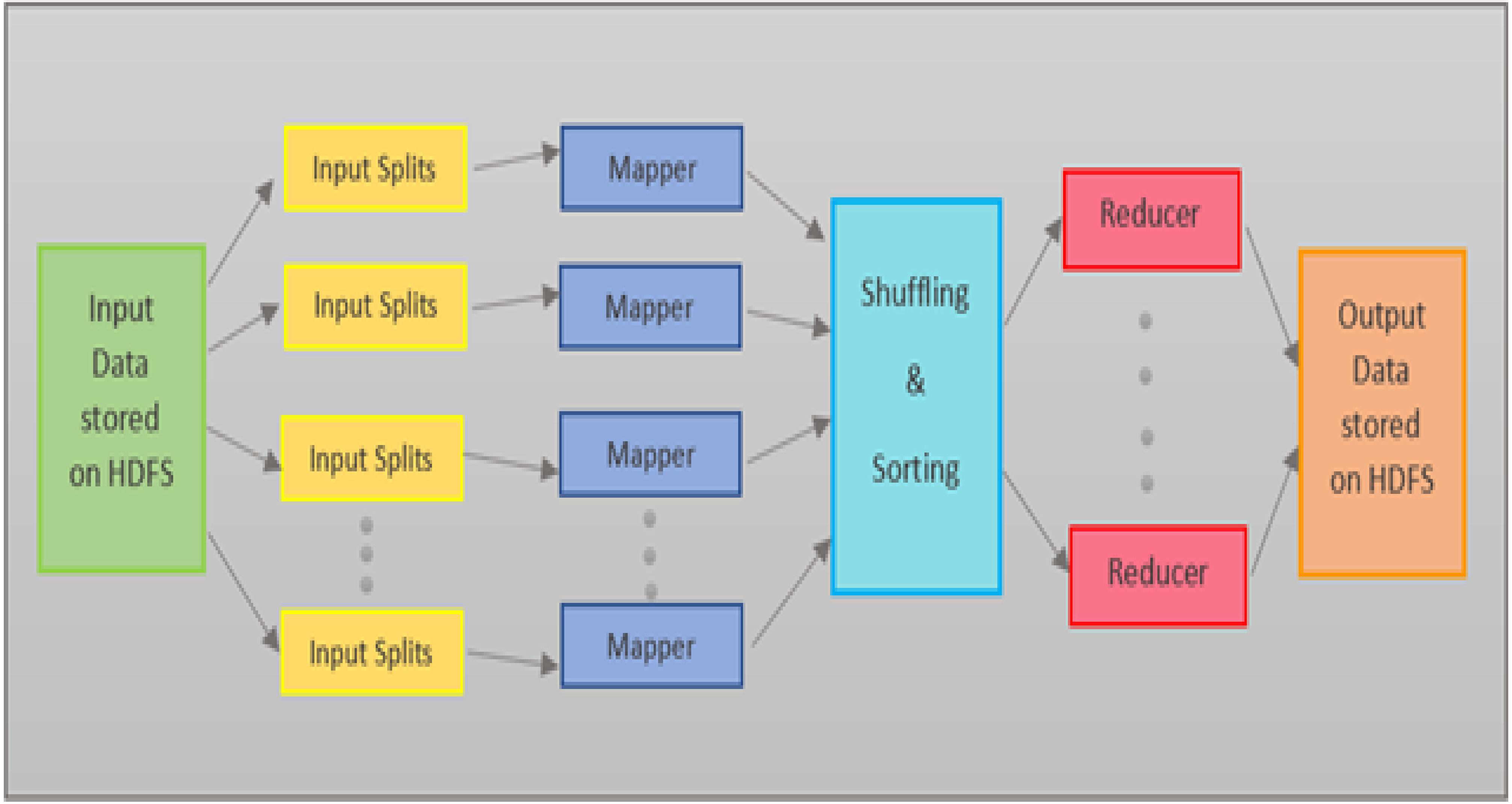
- The TaskTracker is the worker daemon that runs on each DataNode (slave node) in the Hadoop cluster.
- It receives task assignments (map or reduce tasks) from the JobTracker and executes them.
- Each TaskTracker has a fixed number of task slots (for running multiple tasks in parallel).
- It continuously sends heartbeat signals to the JobTracker to report task progress and availability.
- If the JobTracker does not receive a heartbeat in time, it considers the TaskTracker dead and reassigns its tasks elsewhere.
- It manages task execution, handles failures, and reports results back to the JobTracker. Essentially, the TaskTracker is the executor of work given by the JobTracker.

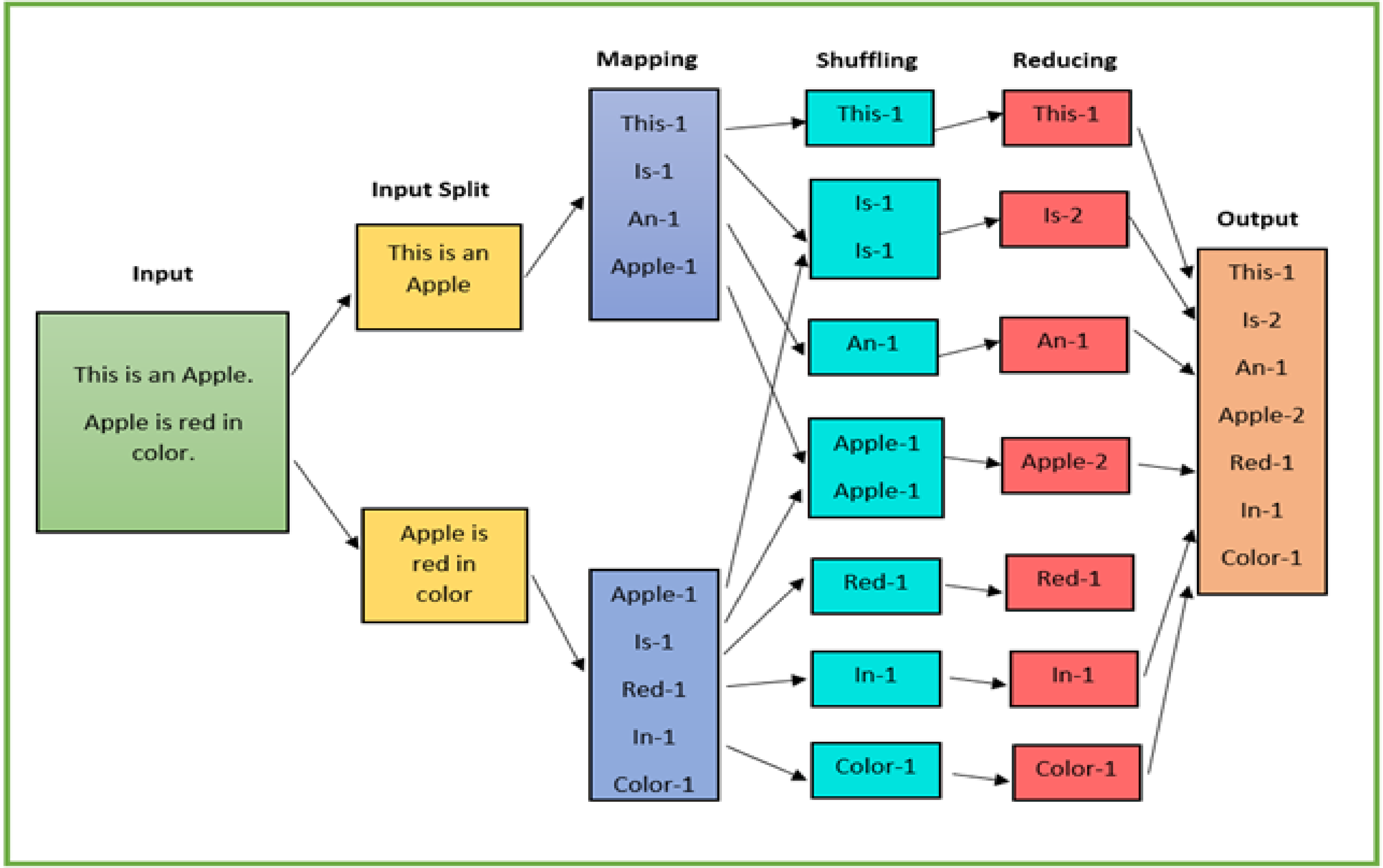
MapReduce: Word Couting





Provided by the programmer
Provided by the programmer
MAP:
Read input and produces a set of key-value pairs
Group by key:
Collect all pairs with same key
Reduce:
Collect all values belonging to the key and output
The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. "The work we're doing now -- the robotics we're doing -- is what we're going to need ....
(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
....
(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
...
(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
...
Only sequential reads
Big document
(key, value)
(key, value)
(key, value)


Input Data stored on HDFS
Input Splits
Input Splits
Input Splits
Input Splits
Mapper
Mapper
Mapper
Mapper
Shuffling & Sorting
Reducer
Reducer
Output Data stored on HDFS


Input
This is an Apple.
Apple is red in color.
Input Split
This is an Apple
Apple is red in color
Mapping
This-1
Is-1
An-1
Apple-1
Apple-1
Is-1
Red-1
In-1
Color-1
Shuffling
This-1
Is-1
Is-1
An-1
Apple-1
Apple-1
Red-1
In-1
Color-1
Reducing
This-1
Is-2
An-1
Apple-2
Red-1
In-1
Color-1
Output
This-1
Is-2
An-1
Apple-2
Red-1
In-1
Color-1

Map and Reduce Function

```
map(key, value):
// key: document name; value: text of the document
  for each word w in value:
     emit(w, 1)


reduce(key, values):
// key: a word; value: an iterator over counts
      result = 0
      for each count v in values:
            result += v
      emit(key, result)
```

MapReduce

- **MapReduce** is a programming model for data processing.
- Hadoop can run MapReduce programs written in various languages; such as Java, Ruby, and Python.
- Most importantly, MapReduce programs are inherently parallel.
- **A weather dataset :** a program that mines weather data. Weather **sensors collect data every hour at many locations across the globe and gather a large volume of log data,** which is a good candidate for analysis with MapReduce because we want to process all the data, and the data is semi-structured and record-oriented.
- To find maximum temperature for each year.

A Weather Dataset: Format of a National Climatic Data Center record

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...
```

Each line is a **fixed-width record** where different character positions encode things like station ID, year, month, day, temperature, etc.

```
0057
332130   # USAF weather station identifier
99999    # WBAN weather station identifier
19500101 # observation date
0300     # observation time
4
+51317   # latitude (degrees x 1000)
+028783  # longitude (degrees x 1000)
FM-12
+0171    # elevation (meters)
99999
V020
320      # wind direction (degrees)
1        # quality code
N
0072
1
00450    # sky ceiling height (meters)
1        # quality code
C
N
010000   # visibility distance (meters)
1        # quality code
N
9
-0128    # air temperature (degrees Celsius x 10)
1        # quality code
-0139    # dew point temperature (degrees Celsius x 10)
1        # quality code
10268    # atmospheric pressure (hectopascals x 10)
1        # quality code
```

Analyzing the Data with Hadoop

- To take advantage of the parallel processing that Hadoop provides, we need to express our query as a MapReduce job.
- MapReduce works by breaking the processing into two phases: the map phase and the reduce phase.
- Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer.

Analyzing the Data with Hadoop

- The input to our map phase is the **raw NCDC data**. We choose a text input format that gives us each line in the dataset as **a text value**. The key is the offset of the beginning of the line from the beginning of the file, but as we have no need for this, we ignore it.
- Our map function is simple. We pull out the year and the air temperature, because these are the only fields we are interested in.
- In this case, the map function is just a data preparation phase, setting up the data in such a way that the reduce function can do its work on it: finding the maximum temperature for each year. The map function is also a good place to drop bad records: here we filter out temperatures that are missing, suspect, or erroneous.

Analyzing the Data with Hadoop

To visualize the way the map works, consider the following sample lines of input data

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...
```

These lines are presented as a input to the map function as the key-value pairs:

```
(0, 0067011990999991950051507004…9999999N9+00001+99999999999…)
(106, 0043011990999991950051512004…9999999N9+00221+99999999999…)
(212, 0043011990999991950051518004…9999999N9-00111+99999999999…)
(318, 0043012650999991949032412004…0500001N9+01111+99999999999…)
(424, 0043012650999991949032418004…0500001N9+00781+99999999999…)
```

Example

- The keys are the line offsets within the file, which we ignore in our map function. **The map function merely extracts the year and the air temperature** and emits them as its output (the temperature values have been interpreted as integers):

```
(1950, 0)
(1950, 22)
(1950, −11)
(1949, 111)
(1949, 78)
```

- The output from the map function is processed by the MapReduce framework before being sent to the reduce function.

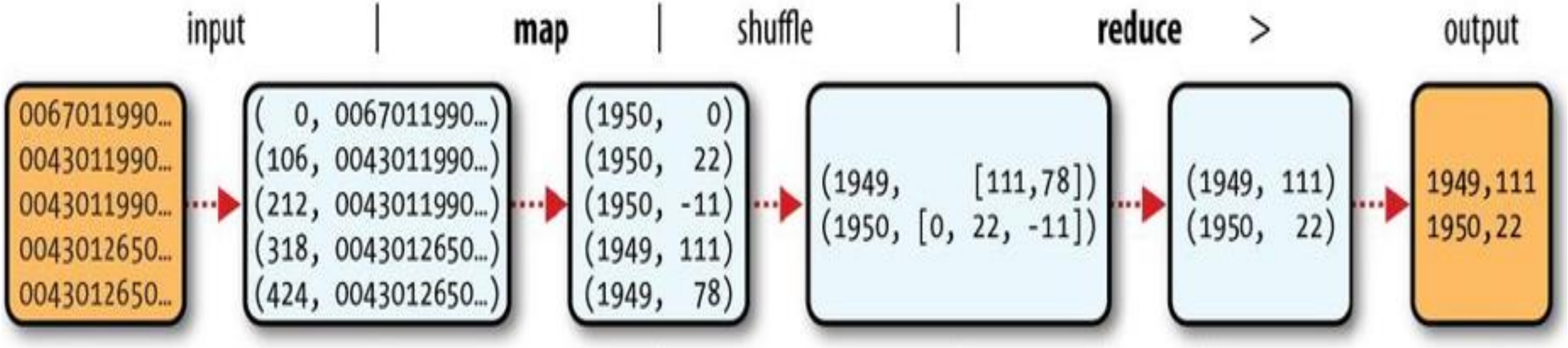
- This processing sorts and groups the key-value pairs by key. So, continuing the example, **reduce function sees the following input**:

```
(1949, [111, 78])
(1950, [0, 22, −11])
```

- Each year appears with a list of all its air temperature readings. All the reduce function has to do now is iterate through the list and **pick up the maximum reading:**

```
(1949, 111)
(1950, 22)
```

Example



input
map
shuffle
reduce
>
output
0067011990...
0043011990...
0043011990...
0043012650...
0043012650...
(0, 0067011990...)
(106, 0043011990...)
(212, 0043011990...)
(318, 0043012650...)
(424, 0043012650...)
(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)
(1949, [111,78])
(1950, [0, 22, -11])
(1949, 111)
(1950, 22)
1949,111
1950,22

Mapper Function

Mapper Class:

- Generic type with four formal type parameters.
- Specifies input key, input value, output key, and output value types for the map function.
- **For the example**:
  - Input key: long integer offset,
  - Input value: line of text,
  - Output key: year,
  - Output value: air temperature (integer).

Hadoop Basic Types:

- Uses Hadoop's own basic types .
- Found in the `org.apache.hadoop.io` package.
- Example types:
  - `LongWritable` (corresponds to Java Long),
  - `Text` (similar to Java String),
  - `IntWritable` (similar to Java Integer).

Mapper Function

map() Method:

- Takes a key and a value as parameters.
- Converts the `Text` value (containing the input line) into a Java `String`.
- Uses `substring()` method to extract the columns of interest from the line.

Context in map() Method:

- Provides an instance of `Context` to write the output.
- Wraps the temperature in an `IntWritable`.
- Outputs a record only if the temperature is present.

Mapper for the maximum temperature example

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class MaxTemperatureMapper
extends Mapper<LongWritable, Text, Text, IntWritable> {
        private static final int MISSING = 9999;
        @Override
public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
                String line = value.toString();
                String year = line.substring(15, 19);
                int airTemperature;
```

```
airTemperature = Integer.parseInt(line.substring(87, 92));
                if (airTemperature != MISSING)
context.write(new Text(year), new IntWritable(airTemperature));
        }
    }
}
```

Reducer Function

- Reducer Class:
  - Four formal type parameters specify input and output types for the reduce function.
- Input Types for Reduce Function:
  - **Must match the output types of the map function**.
  - Example input types: `Text` and `IntWritable`.
- Output Types for Reduce Function:
  - Example output types: `Text` and `IntWritable`.
  - Represents a year and its maximum temperature.
- Finding Maximum Temperature:
  - Iterates through temperatures.
  - Compares each temperature with the current record of the highest found so far.
  - Determines the maximum temperature for each year.

Reducer for the maximum temperature example

```java
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class MaxTemperatureReducer
        extends Reducer<Text, IntWritable, Text,
IntWritable> {
        @Override
        public void reduce(Text key, Iterable<IntWritable>
values, Context context)
throws IOException, InterruptedException {
                int maxValue = Integer.MIN_VALUE;
                for (IntWritable value : values) {
                maxValue = Math.max(maxValue, value.get());
                }
                context.write(key, new IntWritable(maxValue));
}
        }
```

Application to find the maximum temperature in the weather dataset

```java
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxTemperature {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperature <input path> <output path>");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperature.class);
    job.setJobName("Max temperature");
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setReducerClass(MaxTemperatureReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Application Class

- Job Object:
  - Specifies and controls how the job is run.
  - Packages code into a JAR file for distribution across the Hadoop cluster.
  - JAR file location is inferred using the class specified in the `setJarByClass()` method.
- Input and Output Paths:
  - Input path specified by `addInputPath()` method on `FileInputFormat`.
  - **Can be a single file, directory (containing all files), or file pattern.**
  - Output path specified by `setOutputPath()` method on `FileOutputFormat`.
  - Specifies the directory for the output files from the reduce function.

Application Class

- Map and Reduce Classes:
  - Map and reduce types specified by `setMapperClass()` and `setReducerClass()` methods.
  - Output types for the reduce function controlled by `setOutputKeyClass()` and `setOutputValueClass()` methods.
- Map Output Types:
  - Map output types default to the same types as the reducer.
- Input Types:
  - Input types controlled by the input format (using the default `TextInputFormat` in this case).

Application Class

- Running the Job:
  - `waitForCompletion()` method on Job submits the job and waits for completion.

- Return Value:
  - `waitForCompletion()` returns a Boolean indicating success (true) or failure (false).
  - Translated into the program's exit code (0 for success, 1 for failure).

- Output:
  - The output was written to the output directory, which contains one output file per reducer. The job had a single reducer, so we find a single file, named part-r-00000:

Basic File handling in HDFS

Command	Description	Usage Example
`hdfs dfs -ls`	List files and directories in a directory.	`hdfs dfs -ls /user/<username>/`
`hdfs dfs -mkdir`	Create a directory in HDFS.	`hdfs dfs -mkdir /user/<username>/new_directory`
`hdfs dfs -put`	Copy files from the local filesystem to HDFS.	`hdfs dfs -put local_file.txt /user/<username>/`
`hdfs dfs -get`	Copy files from HDFS to the local filesystem.	`hdfs dfs -get /user/<username>/file.txt local_dir`
`hdfs dfs -cat`	Display the contents of a file in HDFS.	`hdfs dfs -cat /user/<username>/file.txt`
`hdfs dfs -cp`	Copy files within HDFS.	`hdfs dfs -cp /user/<username>/file.txt /user/<username>/new_location/`
`hdfs dfs -mv`	Move files within HDFS.	`hdfs dfs -mv /user/<username>/old_location/file.txt /user/<username>/new_location/`
`hdfs dfs -rm`	Remove files or directories from HDFS.	`hdfs dfs -rm /user/<username>/file.txt`
`hdfs dfs -du`	Display disk usage of files and directories.	`hdfs dfs -du -h /user/<username>/`
`hdfs dfs -chmod`	Change permissions of files or directories.	`hdfs dfs -chmod 755 /user/<username>/file.txt`
`hdfs dfs -chown`	Change owner of files or directories.	`hdfs dfs -chown <new_owner>:<new_group> /user/<username>/file.txt`
`hdfs dfs -setrep`	Set the replication factor for a file.	`hdfs dfs -setrep 3 /user/<username>/file.txt`
`hdfs dfsadmin -report`	Display a summary of the HDFS cluster.	`hdfs dfsadmin -report`

Hadoop Framework

- The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.
- It is designed to scale up from single servers to thousands of machines, each offering local computation and storage

Hadoop Framework

- Hadoop Ecosystem: Hive, Pig, Sqoop, Oozie, HBase, Flume, Mahout