Big Data Hadoop and Spark Developer

simplilearn

Distributed Processing: MapReduce Framework and Pig

Learning Objectives

By the end of this lesson, you will be able to:

- Perform distributed processing in MapReduce
- Understand issues with distributed processing and how MR reduces them
- Implement MapReduce paradigm, divide-and-conquer
- Perform Word Count program - Hello World of big data
- Understand Pig
- Implement Pig Latin commands

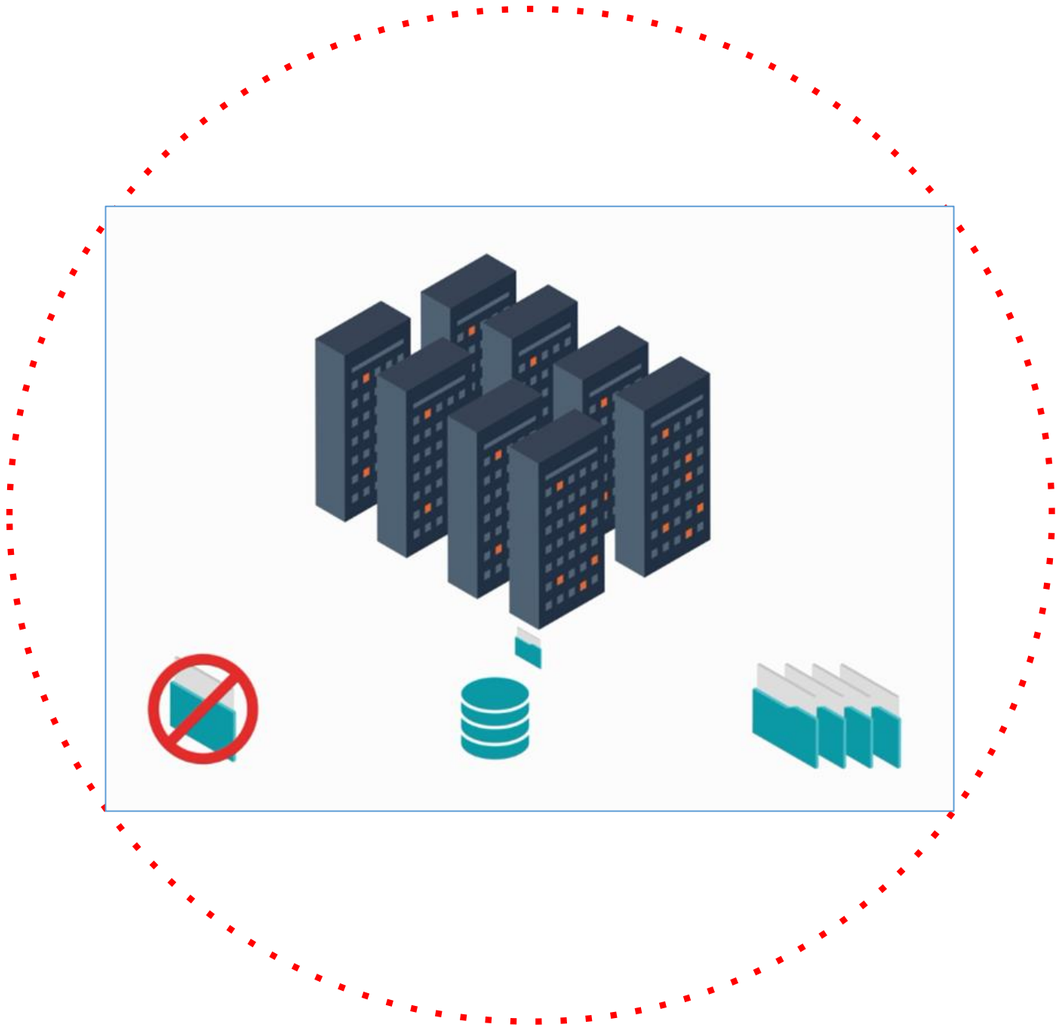
DATA AND
ARTIFICIAL INTELLIGENCE

Distributed Processing in MapReduce

Introduction to MapReduce

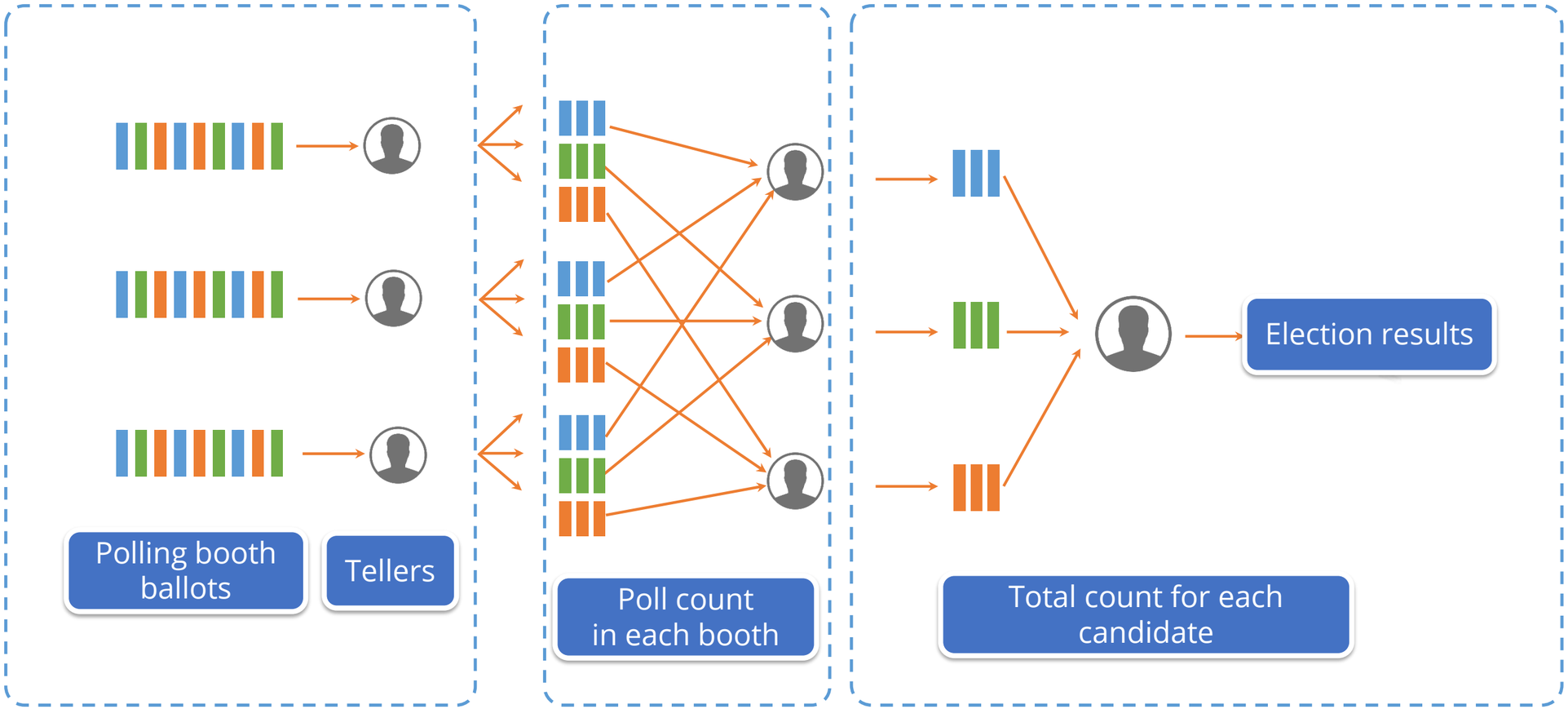
MapReduce is a programming model that simultaneously processes and analyzes huge data sets logically into separate clusters. While **Map** sorts the data, **Reduce** segregates it into logical clusters, thus removing the bad data and retaining the necessary information.

Why MapReduce?



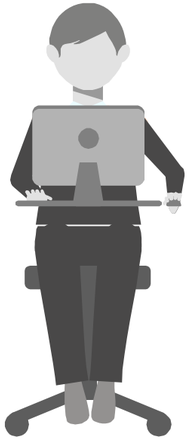
Huge amounts of data were stored in single servers prior to 2004.

MapReduce: Analogy



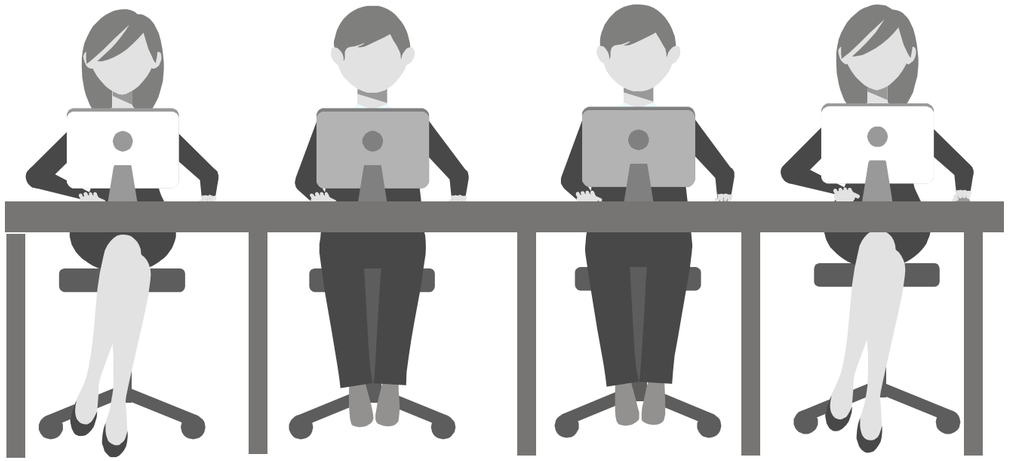
MapReduce: Analogy

One month to receive the election results



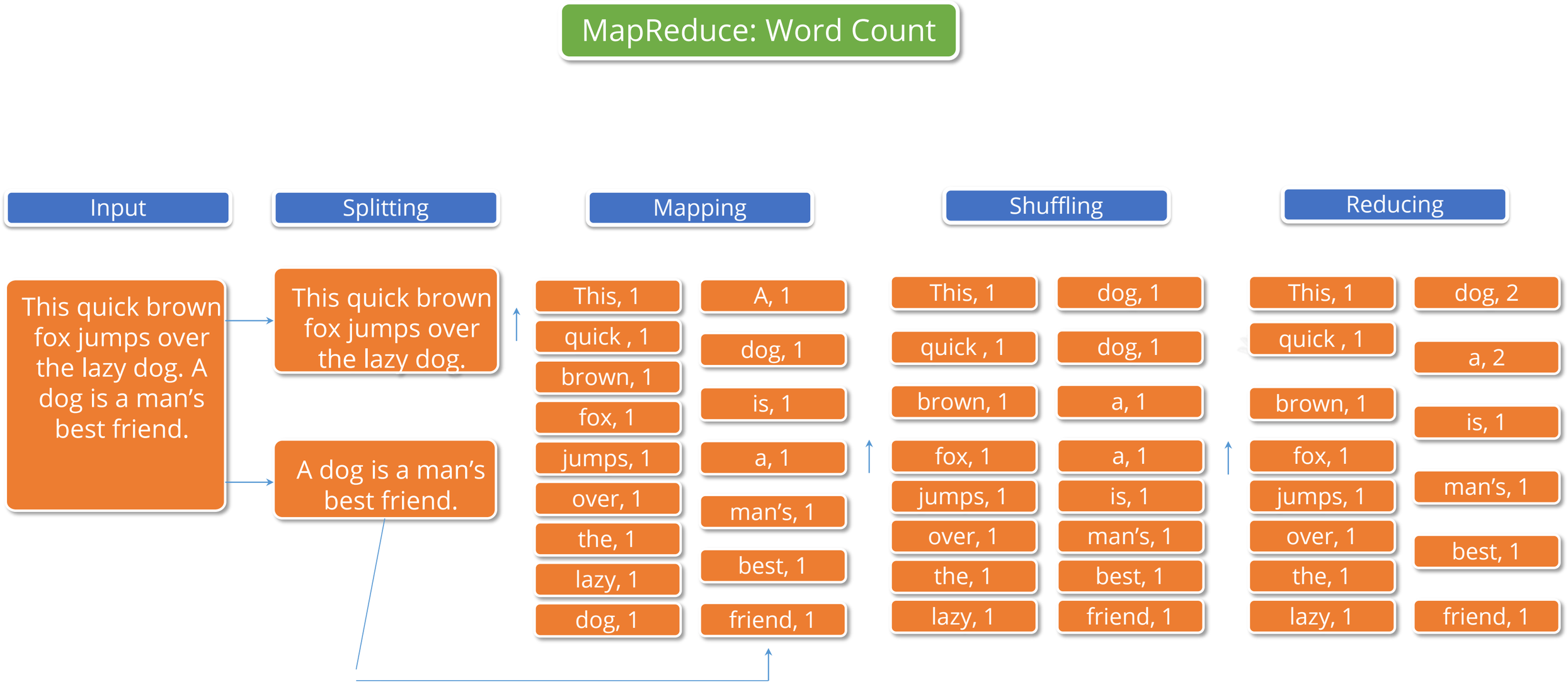
Individual Work

The results are obtained in one or two days



Parallel Work

Word Count Example



DATA AND
ARTIFICIAL INTELLIGENCE

Map Execution Phases

Map Execution Phases

Map phase

- Reads assigned input split from HDFS
- Parses input into records as key-value pairs
- Applies map function to each record
- Informs master node of its completion

Partition phase

- Each mapper must determine which reducer will receive each of the outputs
- For any key, the destination partition is the same
- Number of partitions = Number of reducers

Shuffle phase

- Fetches input data from all map tasks for the portion corresponding to the reduce tasks bucket

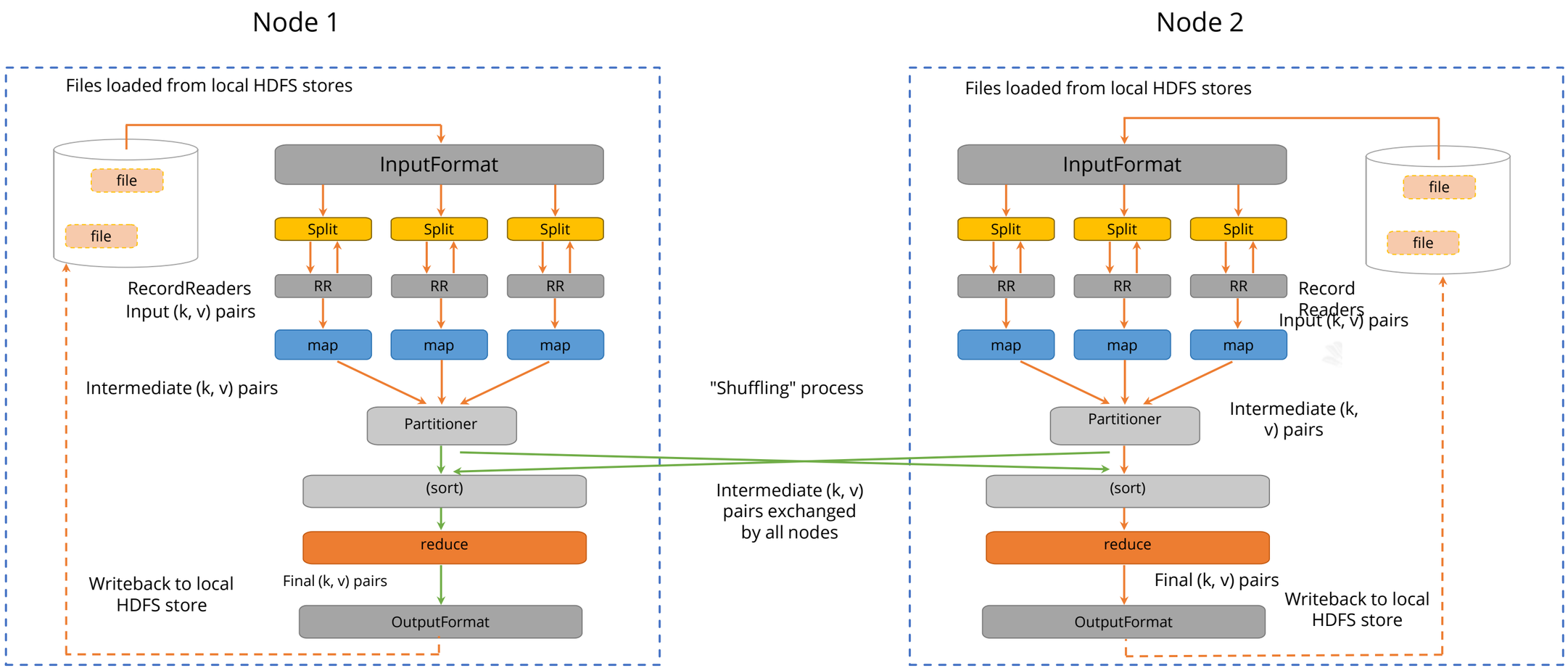
Sort phase

- Merge sorts all map outputs into a single run

Reduce phase

- Applies user-defined reduce function to the merged run

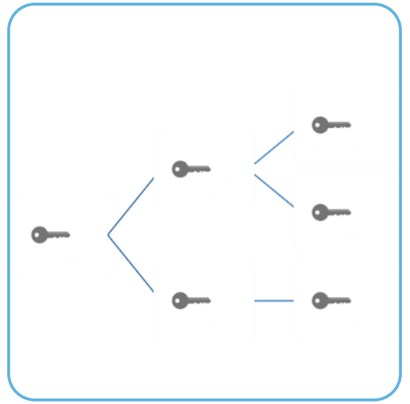
Map Execution: Distributed Two Node Environment



MapReduce Essentials

The job input is specified in key-value pairs:

A user defined map function is applied to each input record to produce a list of intermediate key-value pairs.



A user-defined reduce function is called once for each distinct key in the map output.

MapReduce Essentials

The essentials of each MapReduce phase are as follows:

- The number of reduce tasks can be defined by the users.
- Each reduce task is assigned a set of record groups, that is, intermediate records corresponding to a group of keys.
- For each group, a user-defined reduce function is applied to the recorded values.
- The reduce tasks are read from every map task, and each read returns the record groups for that reduce task.

Reduce phase cannot start until all mappers have finished processing.

DATA AND
ARTIFICIAL INTELLIGENCE

MapReduce Jobs

MapReduce Jobs

A job is a MapReduce program that causes multiple map and reduce functions to run parallelly over the life of the program.

ApplicationMaster and NodeManager functions:

Application Master

- Responsible for the execution of a single application or MapReduce job
- Divides job requests into tasks and assigns them to NodeManagers running on the slave node

NodeManager

- Has many dynamic resource containers
- Executes each active map or reduce task
- Communicates regularly with the Application Master

MapReduce and Associated Tasks

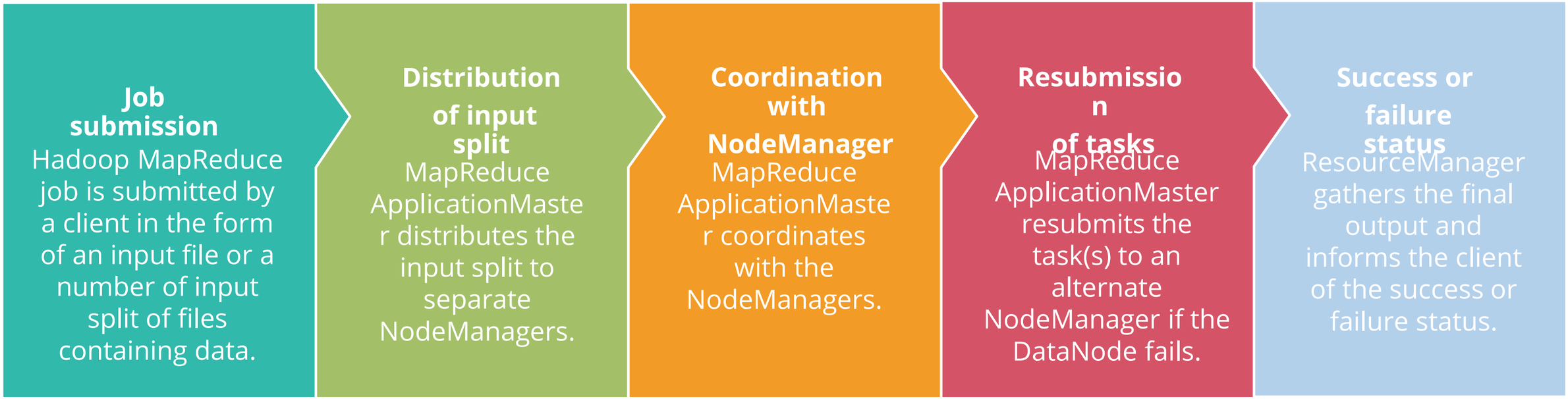
Map process
An initial ingestion and transformation step where initial input records are processed in parallel

Application Master
Keeps track of a MapReduce job

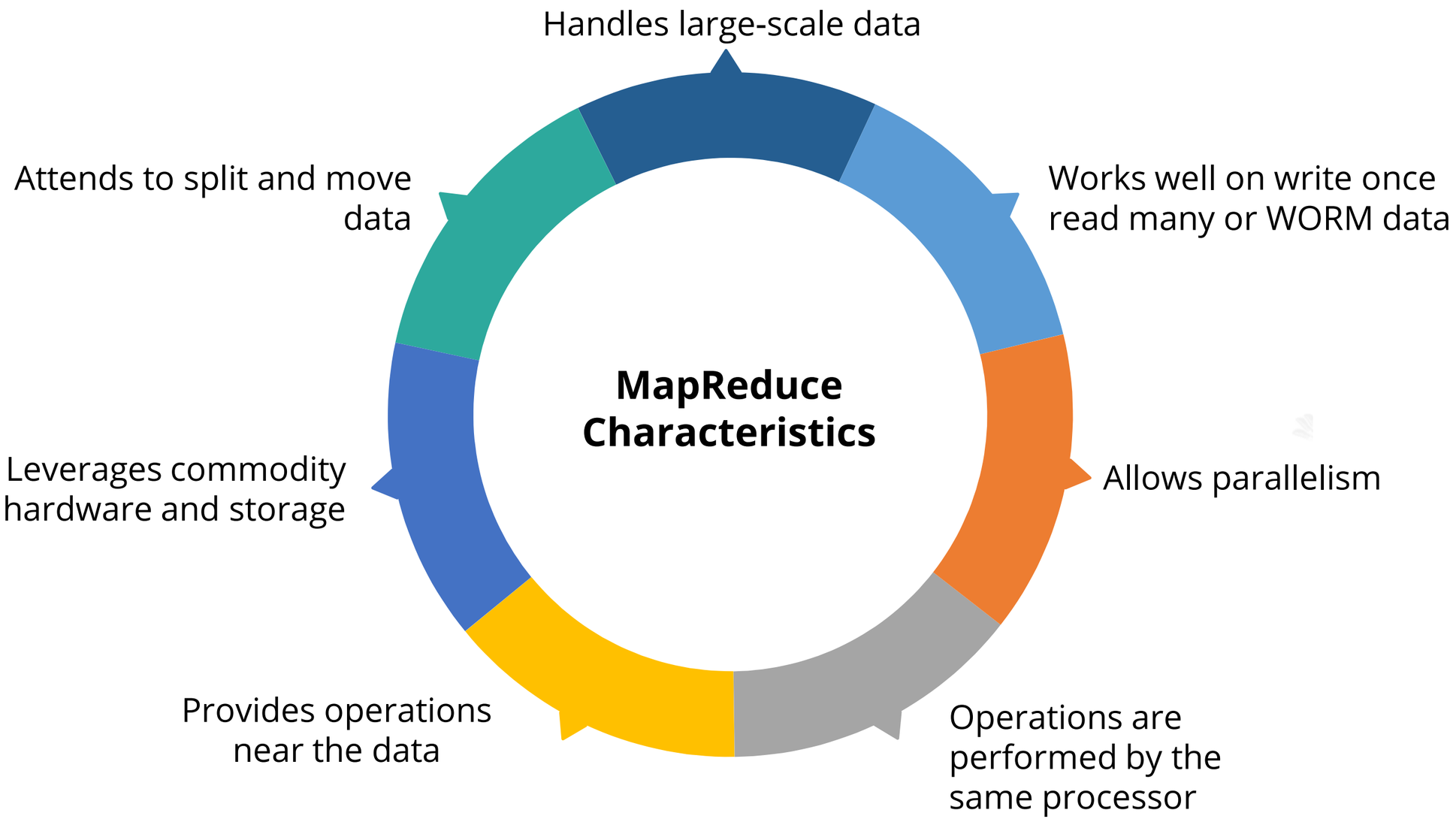
Reduce process
An aggregation or summarization step in which all associated records are processed together

Node Manager
Keeps track of individual map tasks and can run in parallel

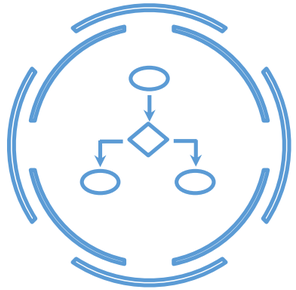
Hadoop MapReduce Job Work Interaction



Characteristics of MapReduce



Real-Time Uses of MapReduce



Algorithms

Sorting

Data mining

Search engine operations

Enterprise analytics

Gaussian analysis

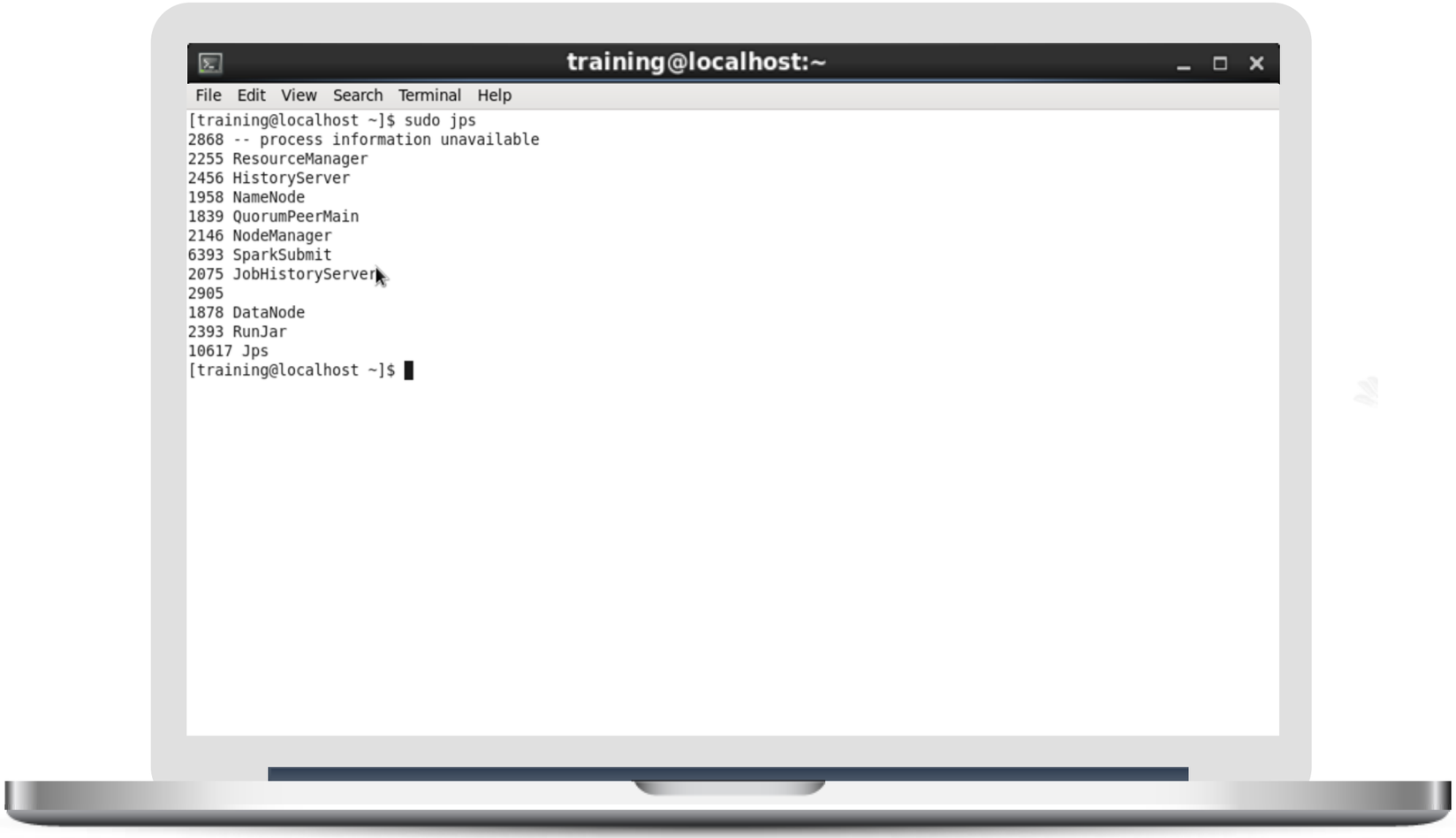
Semantic web 3.0

DATA AND
ARTIFICIAL INTELLIGENCE

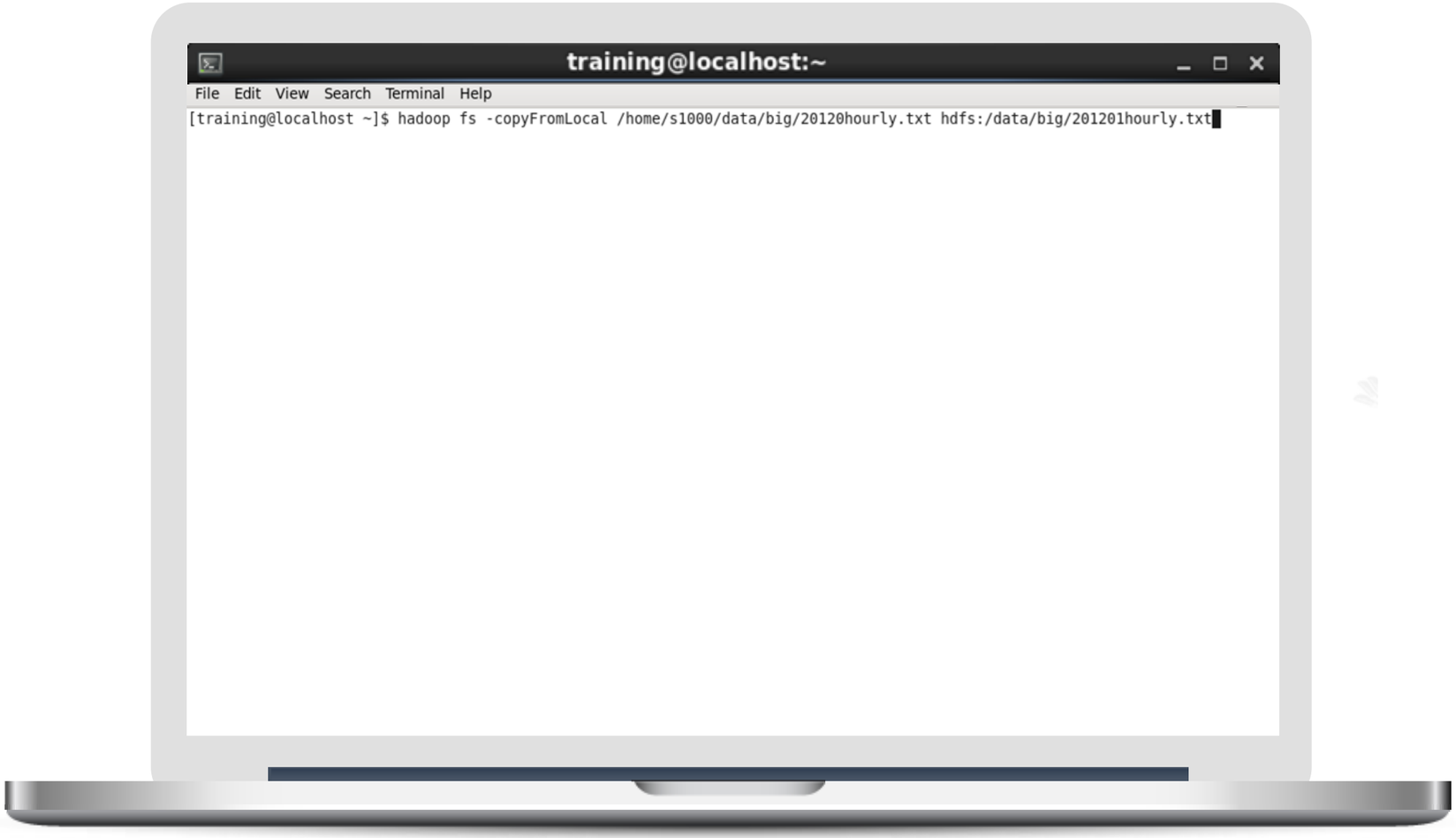
Setting Up the Environment for MapReduce Development

Setting Up the Environment for MapReduce Development

Ensure that all Hadoop services are live and running.

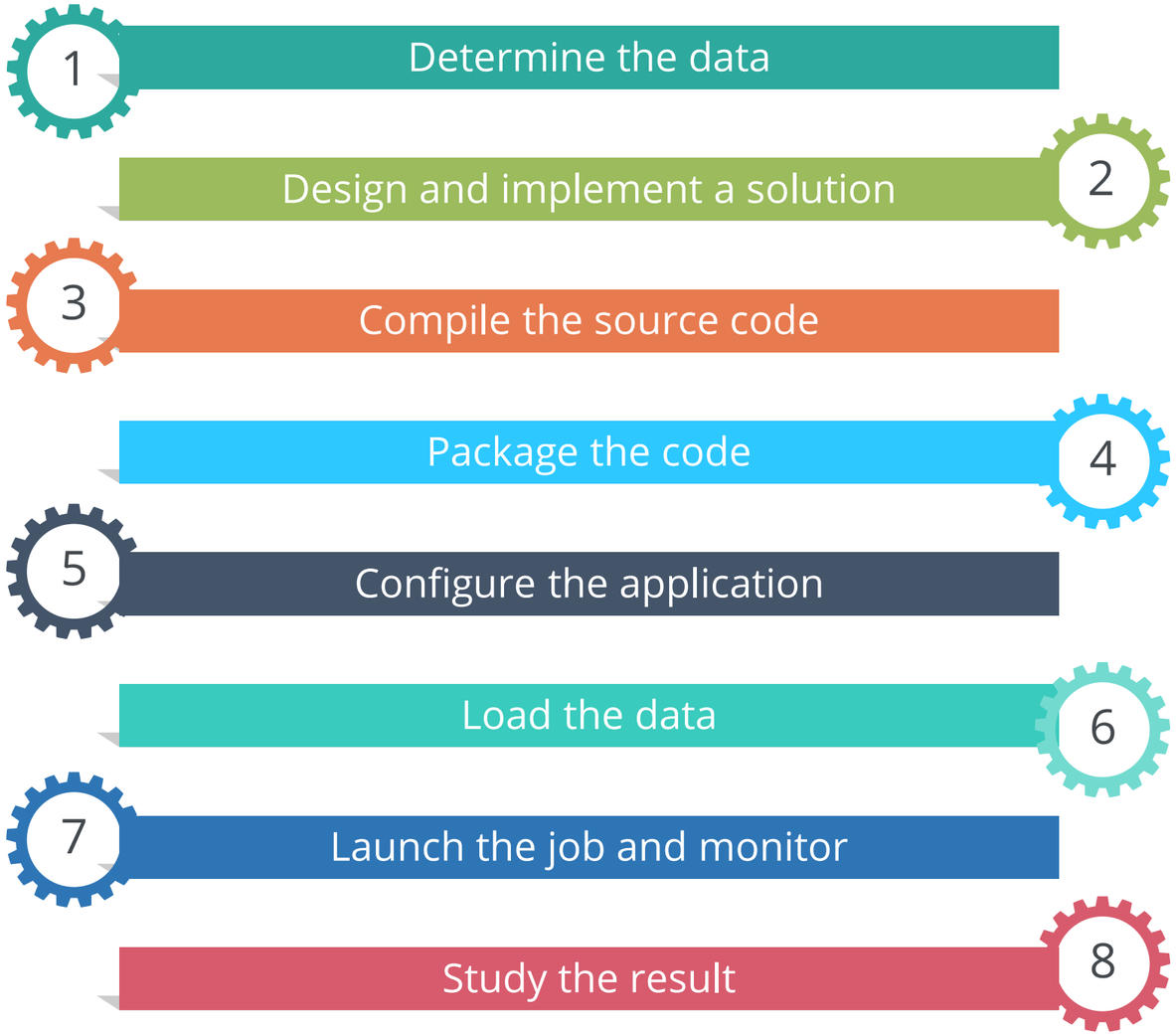


Uploading Big Data and Small Data



Building a MapReduce Program

The steps to build a MapReduce program are as follows:



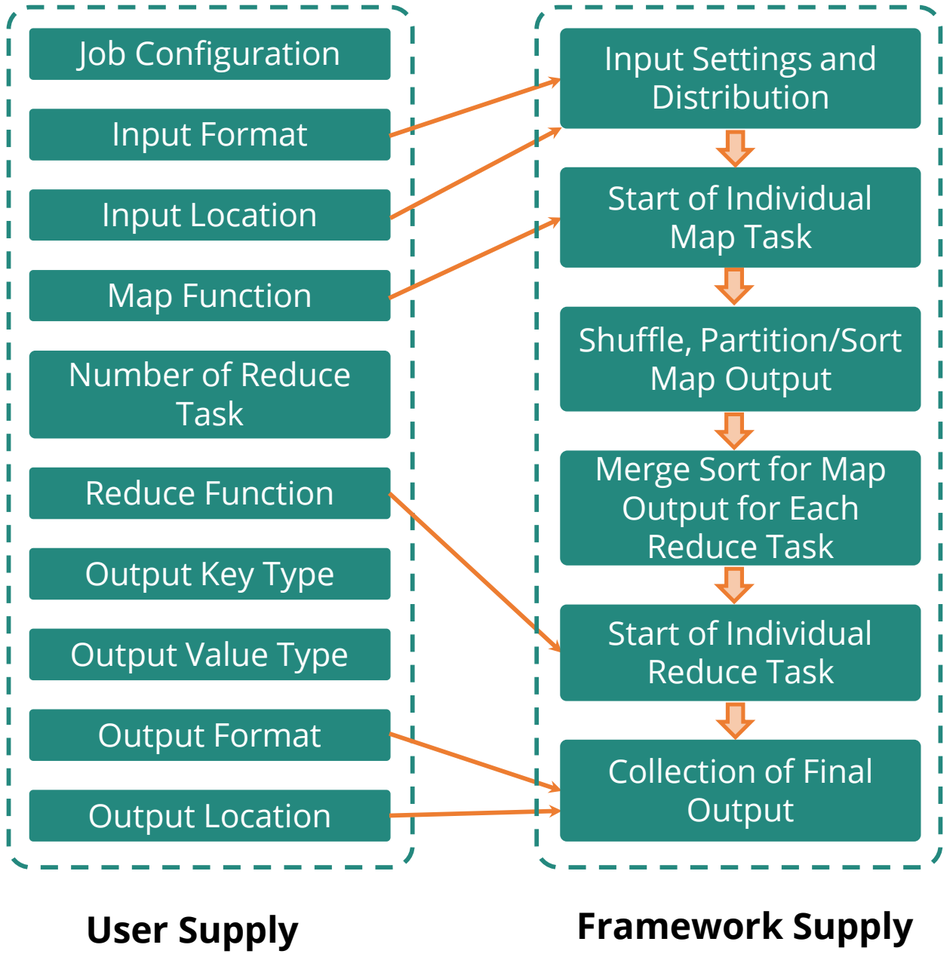
Hadoop MapReduce Requirements

The user or developer is required to set the framework with the following parameters:

- Locations of the job input
- Locations of the job output
- Input format
- Output format
- The class containing the map function
- The class containing the reduce function

Set of Classes

The image shows the set of classes under the user supply and the framework supply.



- ResourceManager accepts the input and hands over the job to ApplicationMaster, which divides the job into tasks.
- NodeManager completes the assignment by executing the map task as part of container execution.
- The reducer starts the merging process.
- The output is collected.

MapReduce - Responsibilities

Responsibilities

Developer

- Setting up the job
- Specifying the input location
- Ensuring correct format and location of input

Framework

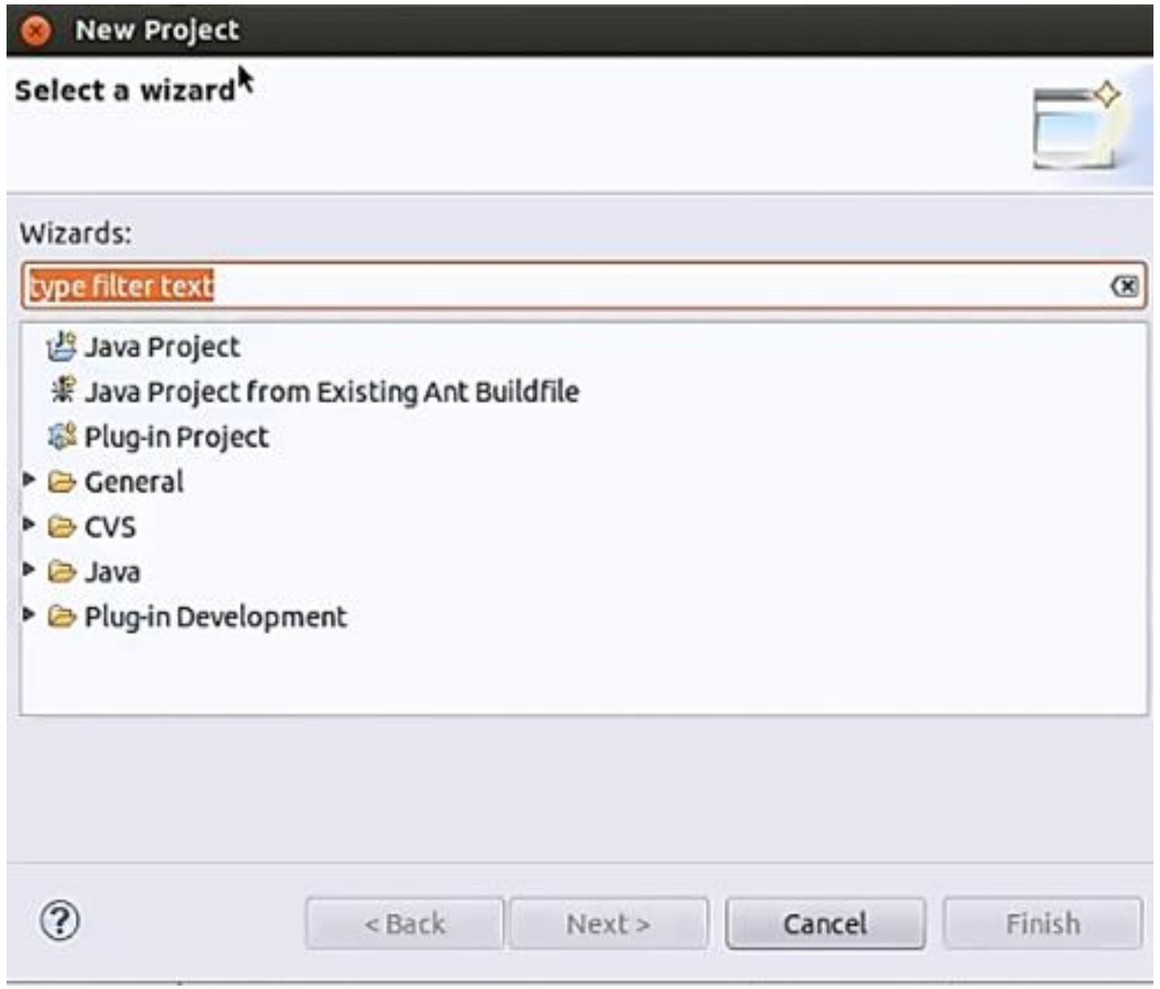
- Distributing the job
- Running the map operation
- Performing shuffling and sorting
- Reducing phases
- Placing the output
- Informing the user of job-completion status

DATA AND
ARTIFICIAL INTELLIGENCE

Creating a New Project

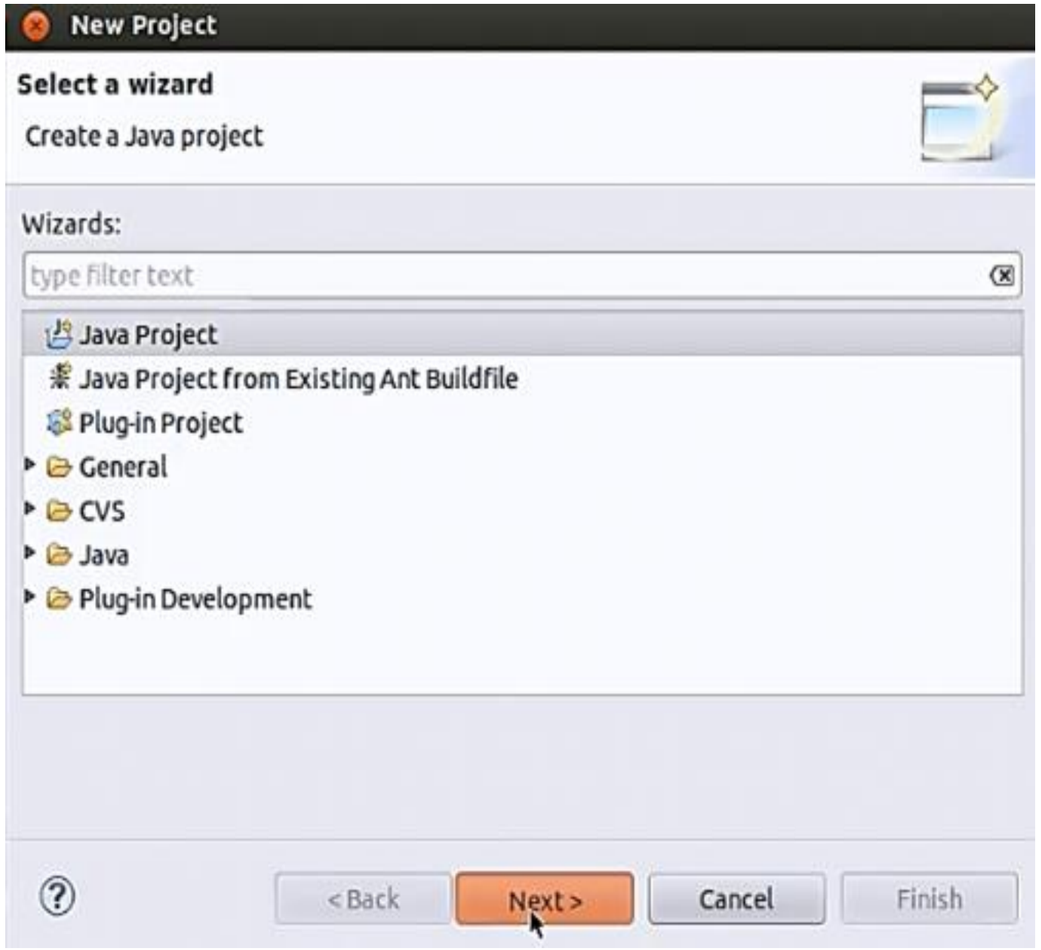
Create a New Project: Step 1

Create a new project and add essential jar files to run MapReduce programs.



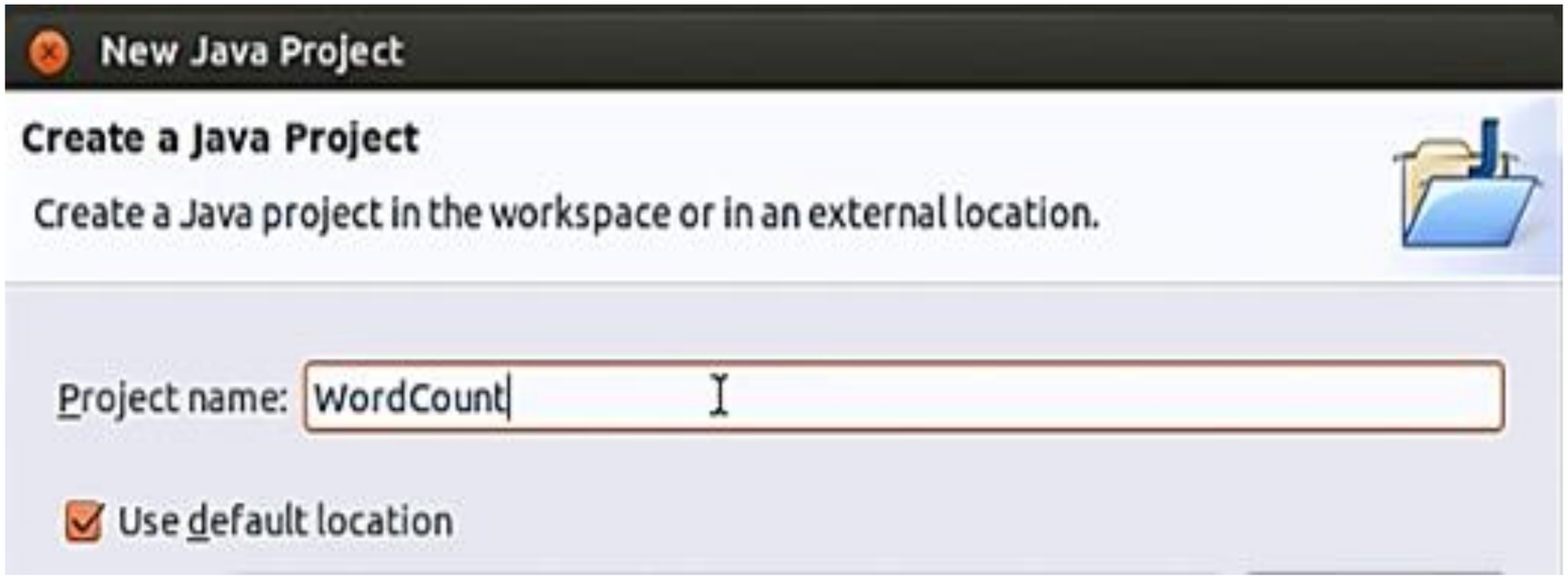
Create a New Project: Step 2

Select Java Project. Then click the Next button to continue.



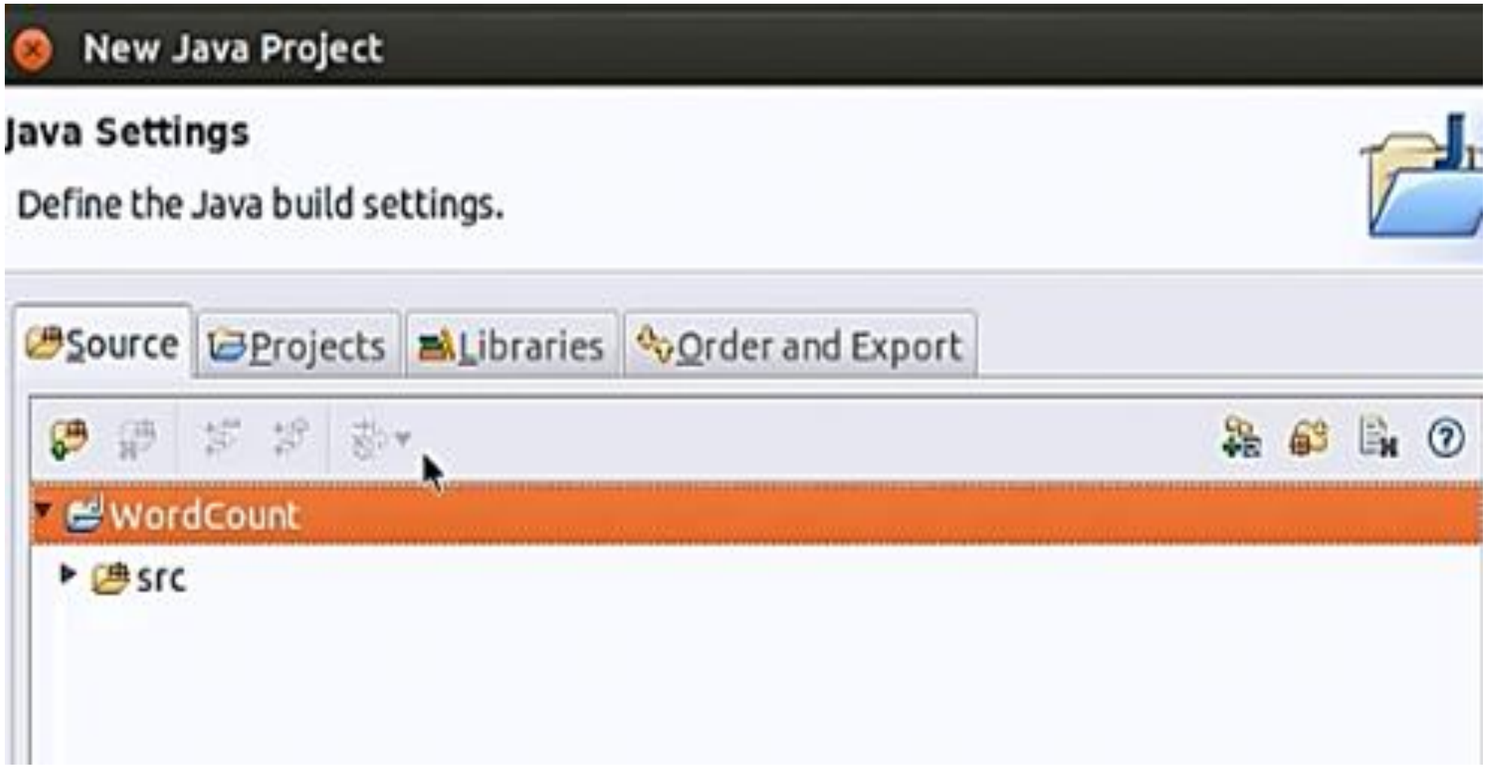
Create a New Project: Step 3

Enter the project name.



Create a New Project: Step 4

Include jar files from the Hadoop framework to ensure the programs locate the dependencies to one location.

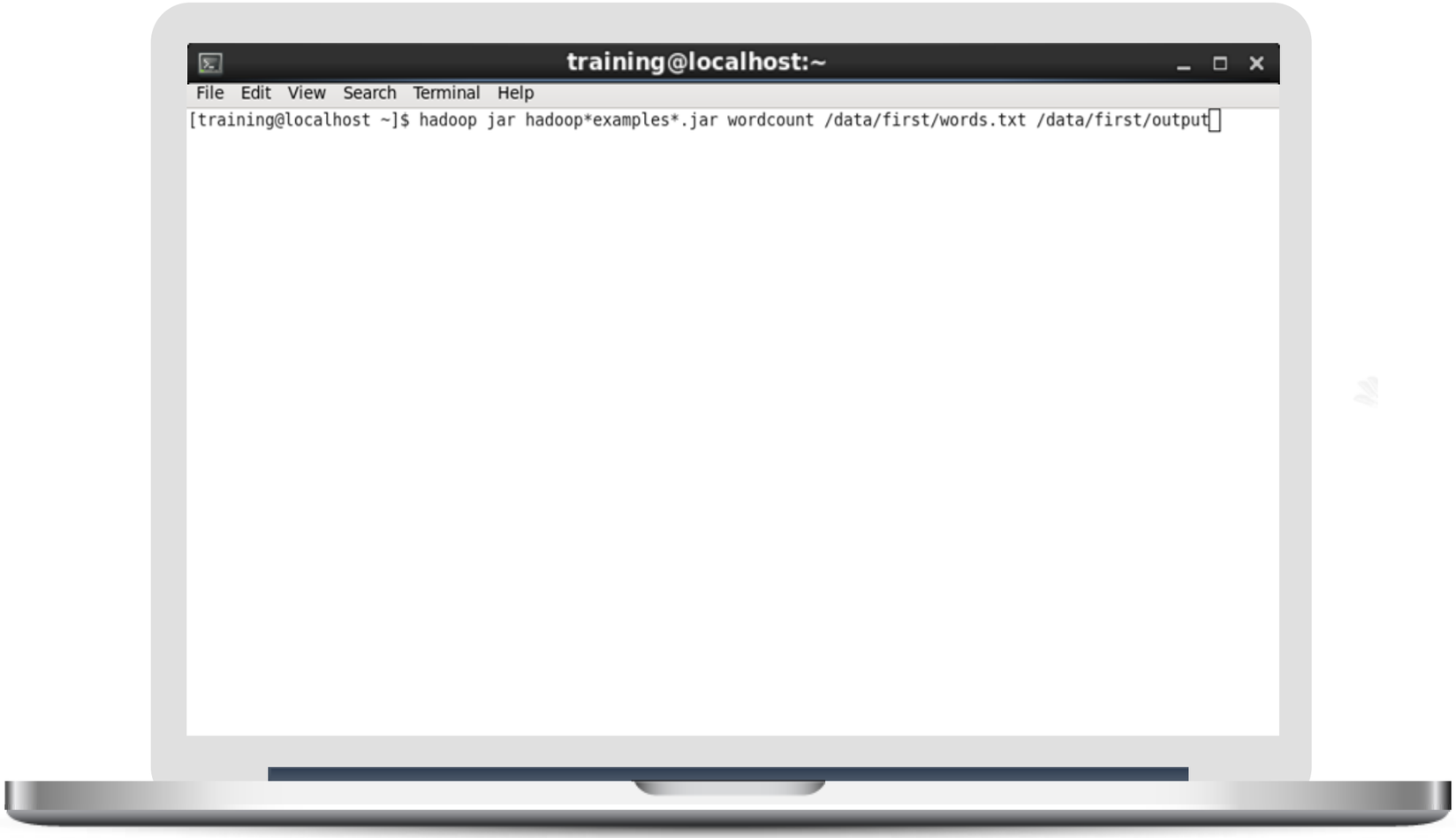


Create a New Project: Step 5

Add the essential jar files.

Checking Hadoop Environment for MapReduce

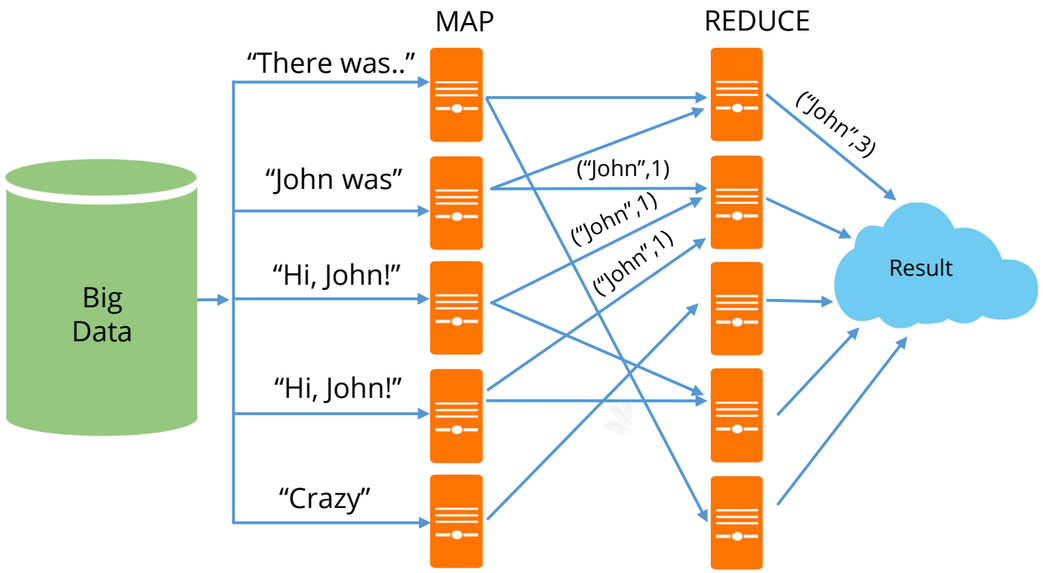
Ensure that the machine setup can perform MapReduce operations.



Advanced MapReduce

Hadoop MapReduce uses data types when it works with user-given mappers and reducers. The data is read from files into mappers and emitted by mappers to reducers. The processed data is sent back by the reducers. Data emitted by reducers goes into output files. At every step, data is stored in Java objects.

Writable data types: In the Hadoop environment, objects that can be put to or received from files and across the network must obey the Writable interface.



Interfaces

The interfaces in Hadoop are as follows:

- Writable
- Writable Comparable

A writable interface allows Hadoop to read and write the data in a serialized form for transmission.

```
interface Writable {
 public void readFields(DataInput in);
 public void write(DataOutput out);
}
```

Interfaces

The interfaces in Hadoop are as follows:

Writable

Writable Comparable

A WritableComparable interface extends the Writable interface so that the data can be used as a key and not as a value.

```
int compareTo(Object what)
int hashCode()
```

DATA AND
ARTIFICIAL INTELLIGENCE

Data Types in Hadoop

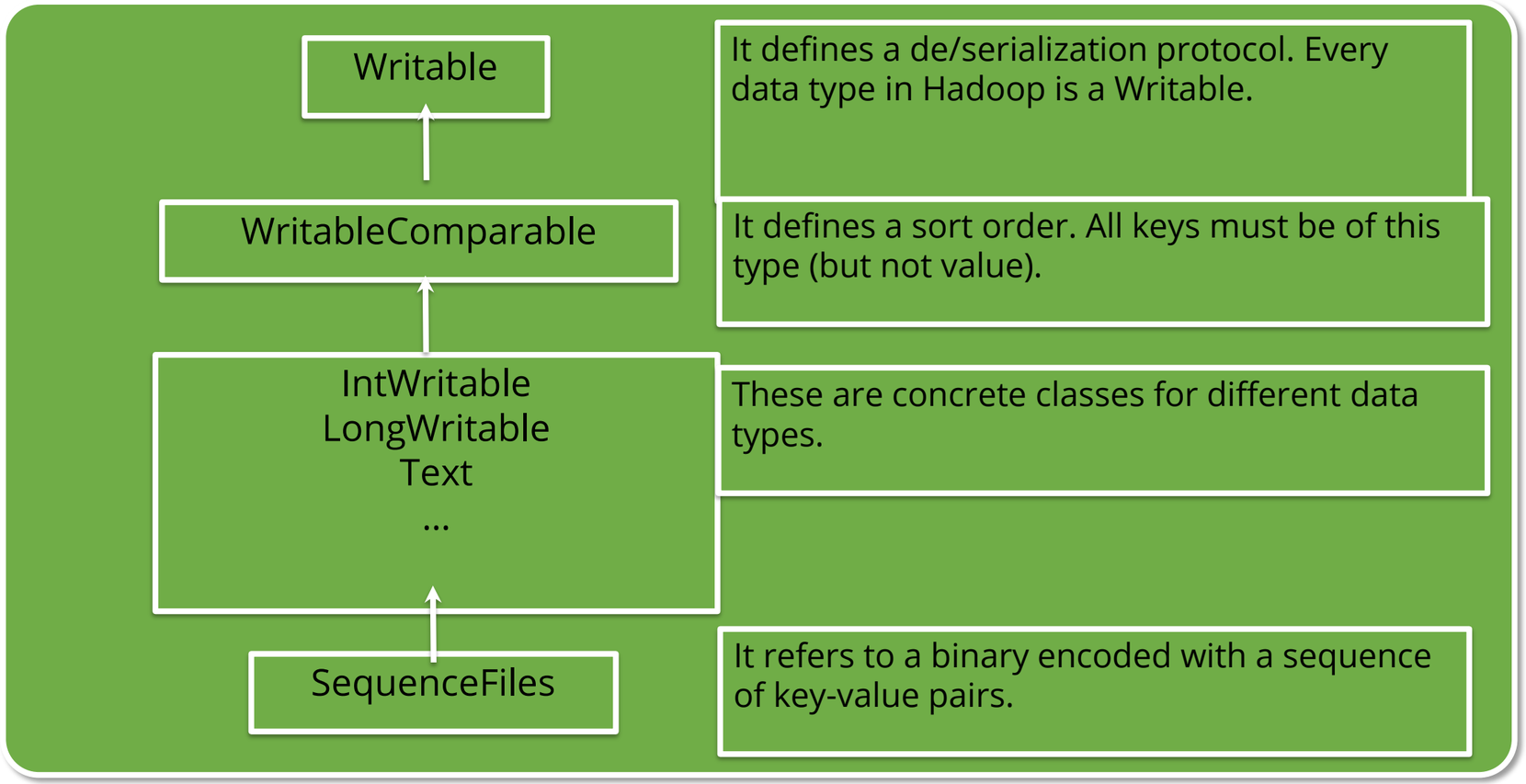
Data Types in Hadoop

The table lists a few important data types and their functions:

Data types	Functions
Text	Stores String data
IntWritable	Stores Integer data
LongWritable	Stores Long data
FloatWritable	Stores Float data
DoubleWritable	Stores Double data
BooleanWritable	Stores Boolean data
ByteWritable	Stores Byte data
NullWritable	Placeholder when value is not needed

Data Types in Hadoop

A sample data type related to the Writable interface is displayed here:



InputFormats in MapReduce

MapReduce can specify how its input is to be read by defining an InputFormat. The table lists some of the classes of InputFormats provided by the Hadoop framework:

InputFormat classes	Description
KeyValueTextInputFormat	One key-value pair per line
TextInputFormat	Key is the line number, and value is the line
NLineInputFormat	Similar to TextInputFormat, but the difference is that there are N number of lines that make an input split
MultiFileInputFormat	Input format that aggregates multiple files into one split
SequenceFileInputFormat	The input file is a Hadoop sequence file which contains a serialized key-value pair.

OutputFormats in MapReduce

The table lists some of the key classes of OutputFormats provided by the Hadoop framework:

OutputFormat classes	Description
TextOutputFormat	It is the default OutputFormat and writes records as lines of text. Each key-value pair is separated by a TAB character. This can be customized by using the mapred.textoutputformat.separator property. The corresponding InputFormat is KeyValueTextInputFormat.
SequenceFileOutputFormat	It writes sequence files to save the output. It is compact and compressed.
SequenceFileAsBinaryOutputFormat	It writes key and value in raw binary format into a sequential file container.
MapFileOutputFormat	It writes MapFiles as the output. The keys in a MapFile must be added in an order, and the reducer will emit keys in the sorted order.
MultipleTextOutputFormat	It writes data to multiple files whose names are derived from output keys and values.
MultipleSequenceFileOutputFormat	It creates output in multiple files in a compressed form.

Distributed Cache

Helps to boost efficiency when a map or a reduce task needs access to common data.

Allows a cluster node to read the imported files from its local file system instead of retrieving the files from other cluster nodes.

Allows both single files and archives (such as zip and tar.gz).

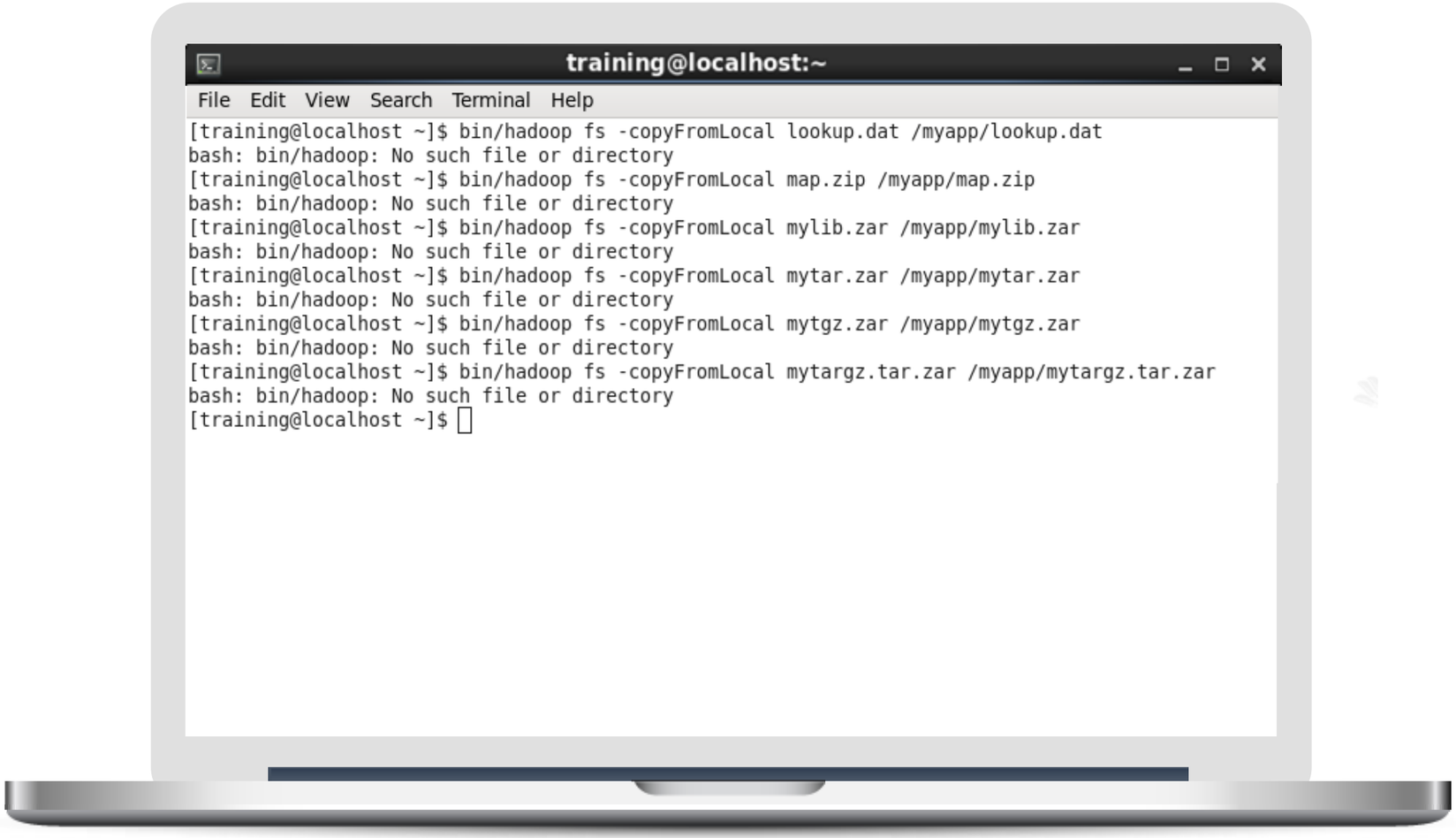
Copies files only to slave nodes. If there are no slave nodes in the cluster, distributed cache copies the files to the master node.

Allows access to the cached files from mapper or reducer applications to make sure that the current working directory (./) is added into the application path.

Allows one to reference the cached files as though they are present in the current working directory.

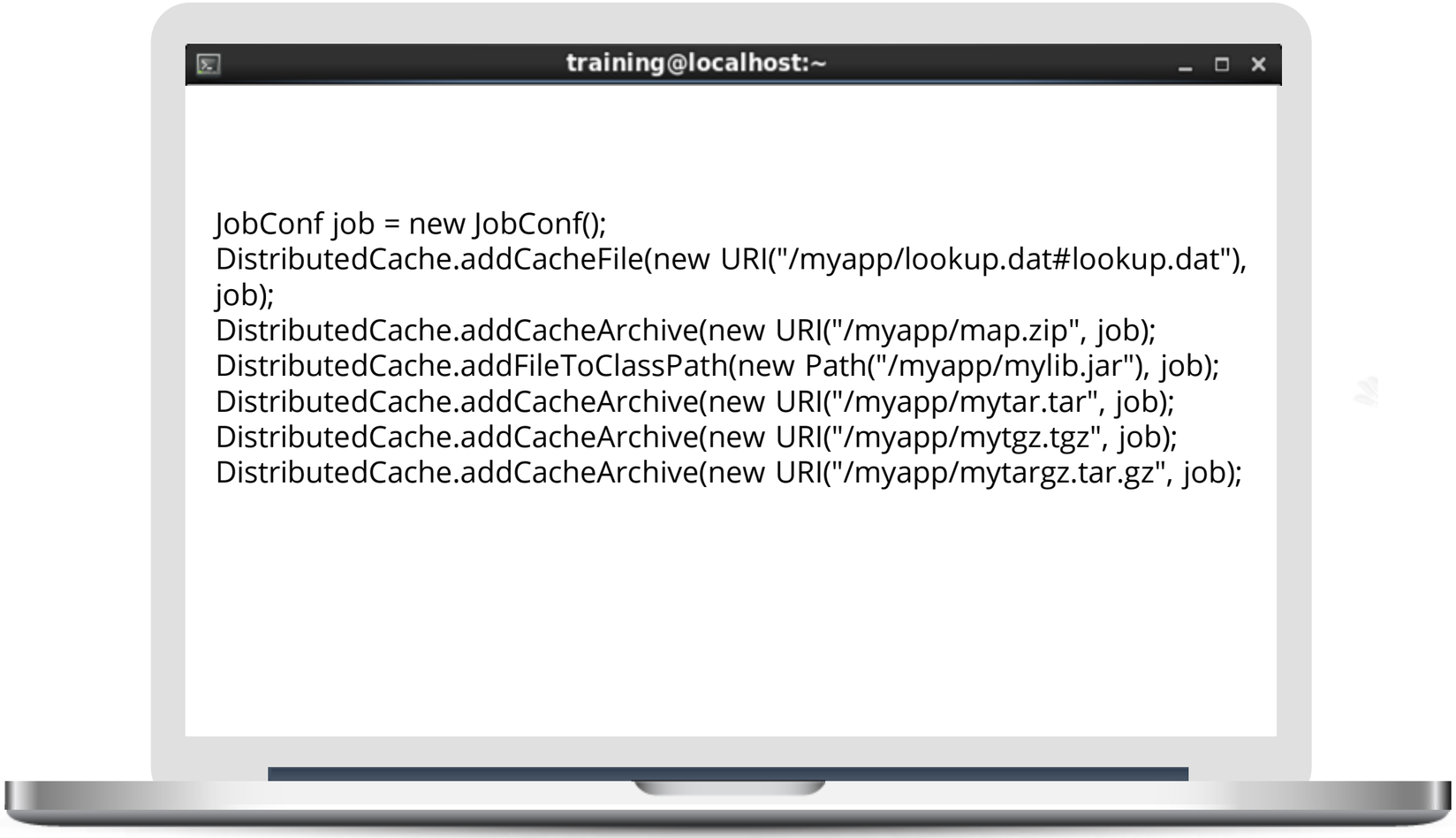
Using Distributed Cache - Step 1

Set up the cache by copying the requisite files to the FileSystem.



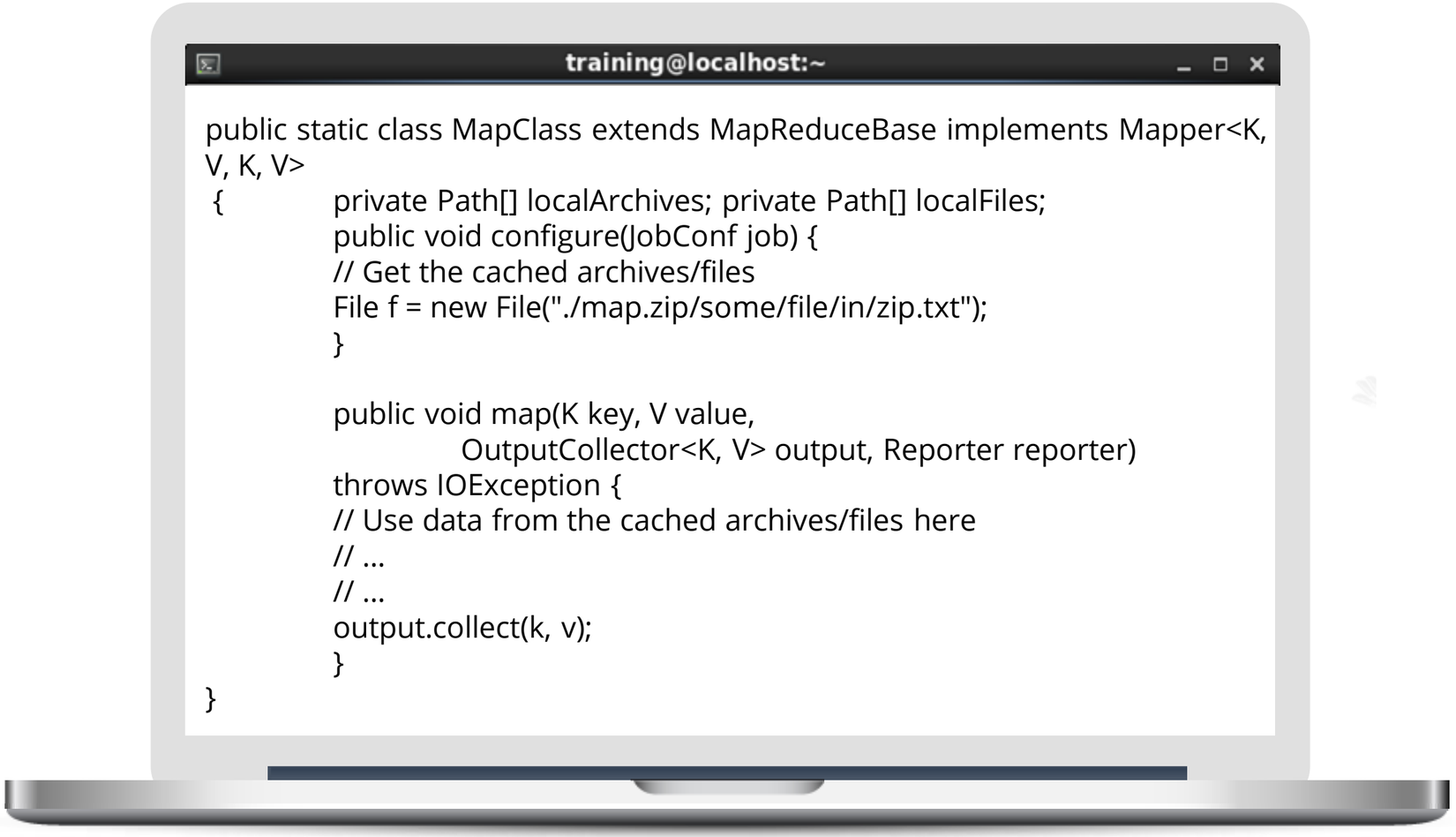
Using Distributed Cache - Step 2

Set up the application's JobConf as shown below.



Using Distributed Cache - Step 3

Use the cached files in the mapper or reducer.



DATA AND
ARTIFICIAL INTELLIGENCE

Joins in MapReduce

Joins in MapReduce

Joins are relational constructs to combine relations. In MapReduce, joins are used to combine two or more datasets. A join is performed either in the map phase or in the reduce phase by taking advantage of the MapReduce Sort-Merge architecture.

Cartesian product

It is a map-side join where every single record is paired up from another dataset.

Reduce side join

It is used for joining two or more large datasets by the same foreign key using any kind of join operation.

Composite join

It is a map-side join on very large formatted input datasets sorted and partitioned by a foreign key.

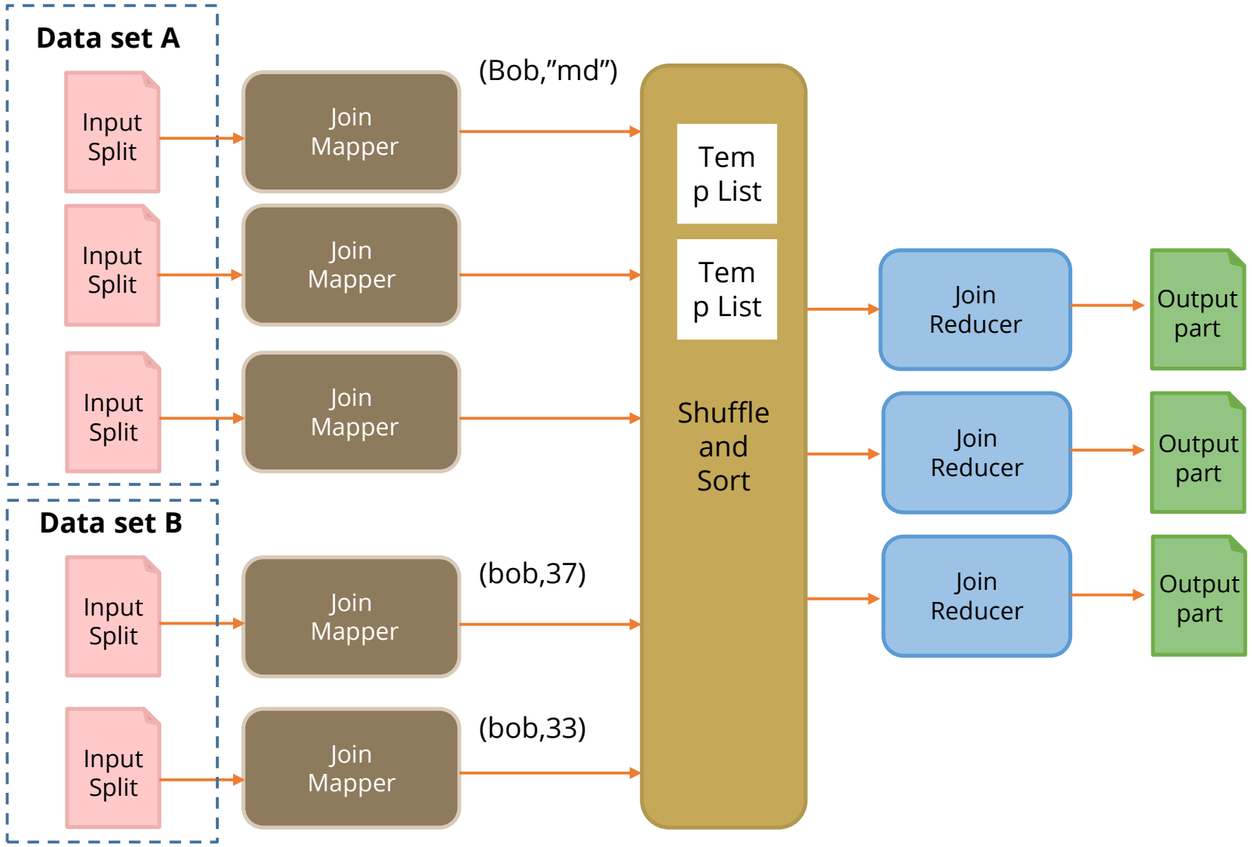
Replicated join

It is a map-side join that works in situations where one of the datasets is small enough to cache.

Reduce Side Join

A reduce side join works in the following ways:

- The mapper prepares for join operations:
  - It takes each input record from every dataset.
  - It emits a foreign key-record pair.
- The reducer performs join operation:
  - It stores the values of each input group in temporary lists.
  - The temporary lists are then iterated over, and the records from both sets are joined.



Reduce Side Join

A reduce side join should be used in the following conditions:

When multiple large datasets are joined by a foreign key

When flexibility is needed to execute any join operation

When a large amount of network bandwidth is available

When there is no limitation on the size of datasets

SQL analogy

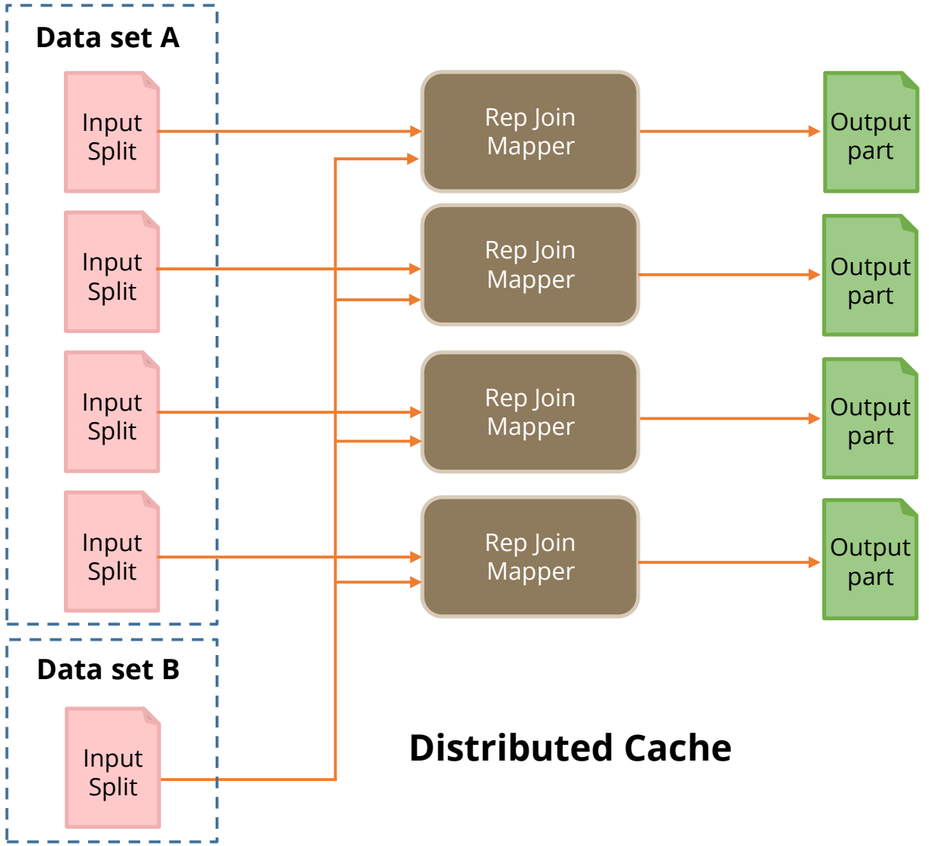
SELECT *users.ID, users.****Location****, comments.upVotes*
FROM *users*
*[****INNER****|****LEFT****|****RIGHT****]* ***JOIN*** *comments*
ON *users.ID=comments.UserID*

Output of a reduce side join:
The number of part files equals the number of reduce tasks.

Replicated Join

A replicated join is a map-only pattern. It works in the following ways:

- It reads all files from the distributed cache and stores them in in-memory lookup tables.
- The mapper processes each record and joins it with the data stored in memory.
- There is no data shuffled to the Reduce phase.
- The mapper gives the final output part.



Replicated Join

A replicated join should be used in the following conditions:

When all datasets except for the largest one can fit into the main memory of each map task that is limited by Java Virtual Machine (JVM) heap size

When there is a need for an inner join or a left outer join, with the large input dataset being the "left" part of the operation

SQL analogy

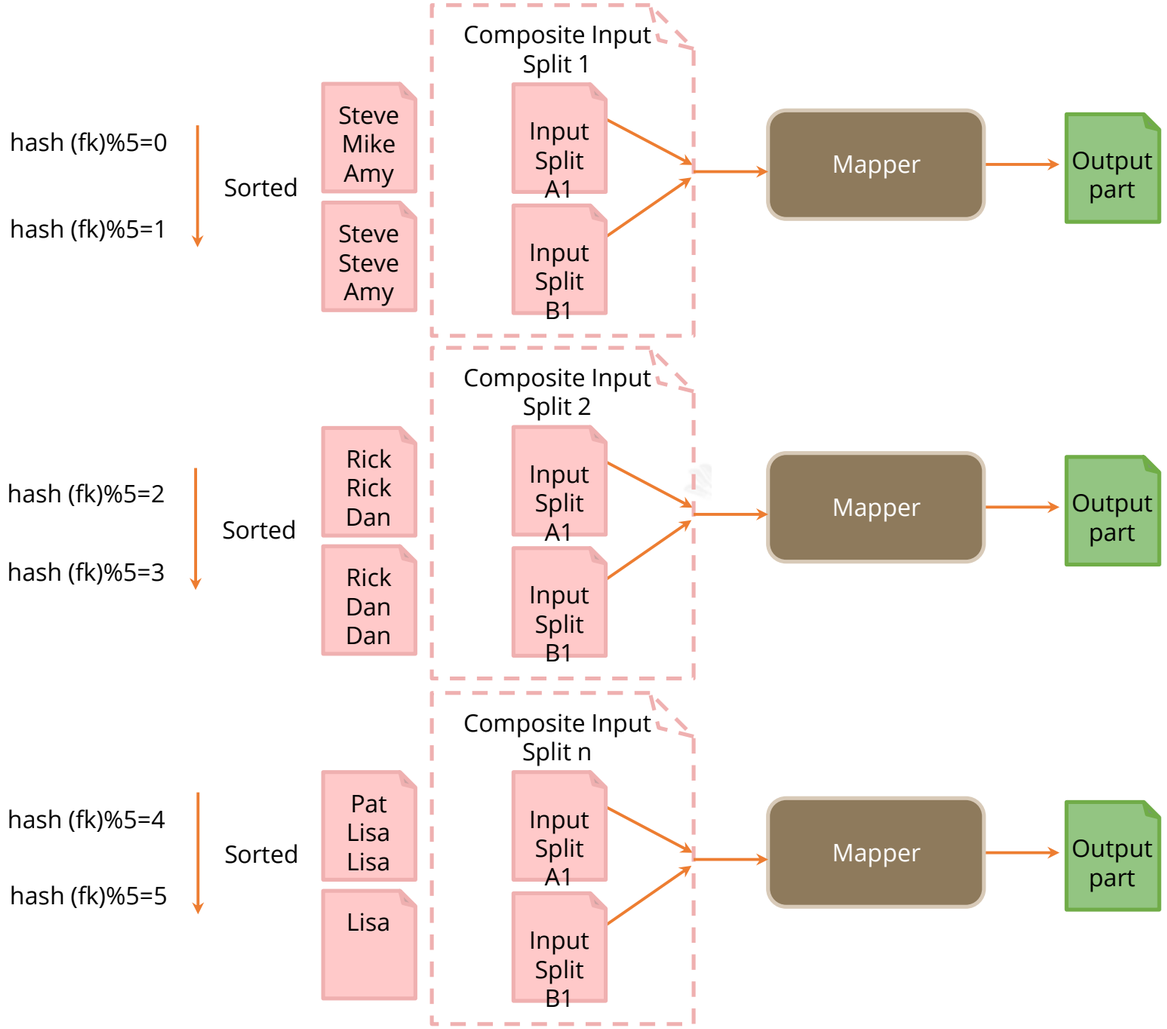
SELECT *users.ID, users.****Location****, comments.upVotes*
FROM *users*
*[****INNER****|****LEFT****|****RIGHT****]* ***JOIN*** *comments*
ON *users.ID=comments.UserID*

Output of a replicated join:
The number of part files equals the number of map tasks.

Composite Join

A composite join is a map-only pattern working in the following ways:

- All datasets are divided into the same number of partitions.
- Each partition of dataset is sorted by a foreign key, and all the foreign keys reside in the associated partition of each dataset.
- Two values are retrieved from the input tuple associated with each dataset; they are based on the foreign key and the output to the file system.



Composite Join

A composite join should be used in the following conditions:

When all datasets are sufficiently large

When there is a need for an inner join or a full outer join

SQL analogy

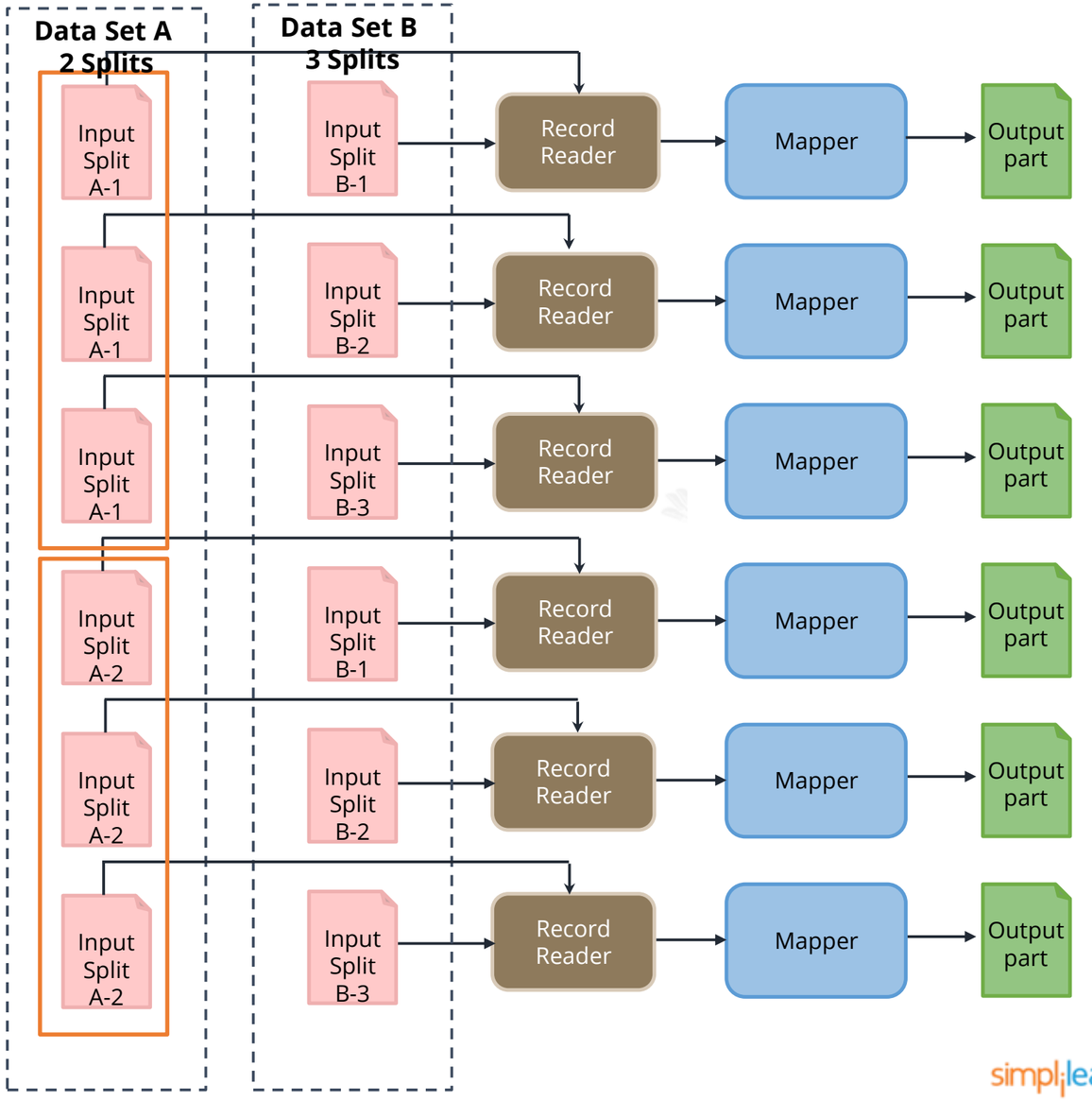
```
SELECT users.ID, users.Location,
comments.upVotes
FROM users
[INNER|LEFT|RIGHT] JOIN
comments
ON users.ID=comments.UserID
```

Output of a composite join:
The number of part files equals the number of map tasks.

Cartesian Product

A Cartesian product is a map-only pattern that works in the following ways:

- Datasets are split into multiple partitions.
- Each partition is fed to one or more mappers. For example, split A-1 and A-2 are fed to 3 mappers each, as shown in the image.
- A RecordReader reads every record of input split associated with the mapper.
- The mapper simply pairs every record of a dataset with every record of all other datasets.



Cartesian Product

A Cartesian product should be used in the following conditions:

When there is a need to analyze relationships between all pairs of individual records

When there are no constraints on the execution time

SQL analogy

SELECT ***
FROM *users*
*[**CROSS**]* ***JOIN*** *comments*

Output of a Cartesian product:
Every possible tuple combination from the input records is represented in the final output.

DATA AND
ARTIFICIAL INTELLIGENCE

Introduction to Pig

Introduction to Pig

Different fundamentals

Need for common operation

Coding difficulty

Rigid dataflow

What is Pig?

- Extensible
- Self-optimizing
- Easily programmed

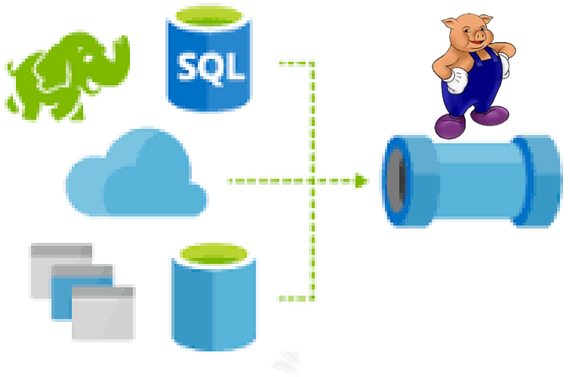
Pig is a scripting platform designed to process and analyze large data sets, and it runs on Hadoop clusters.

Pig—Example

Yahoo has scientists who use grid tools to scan through petabytes of data.

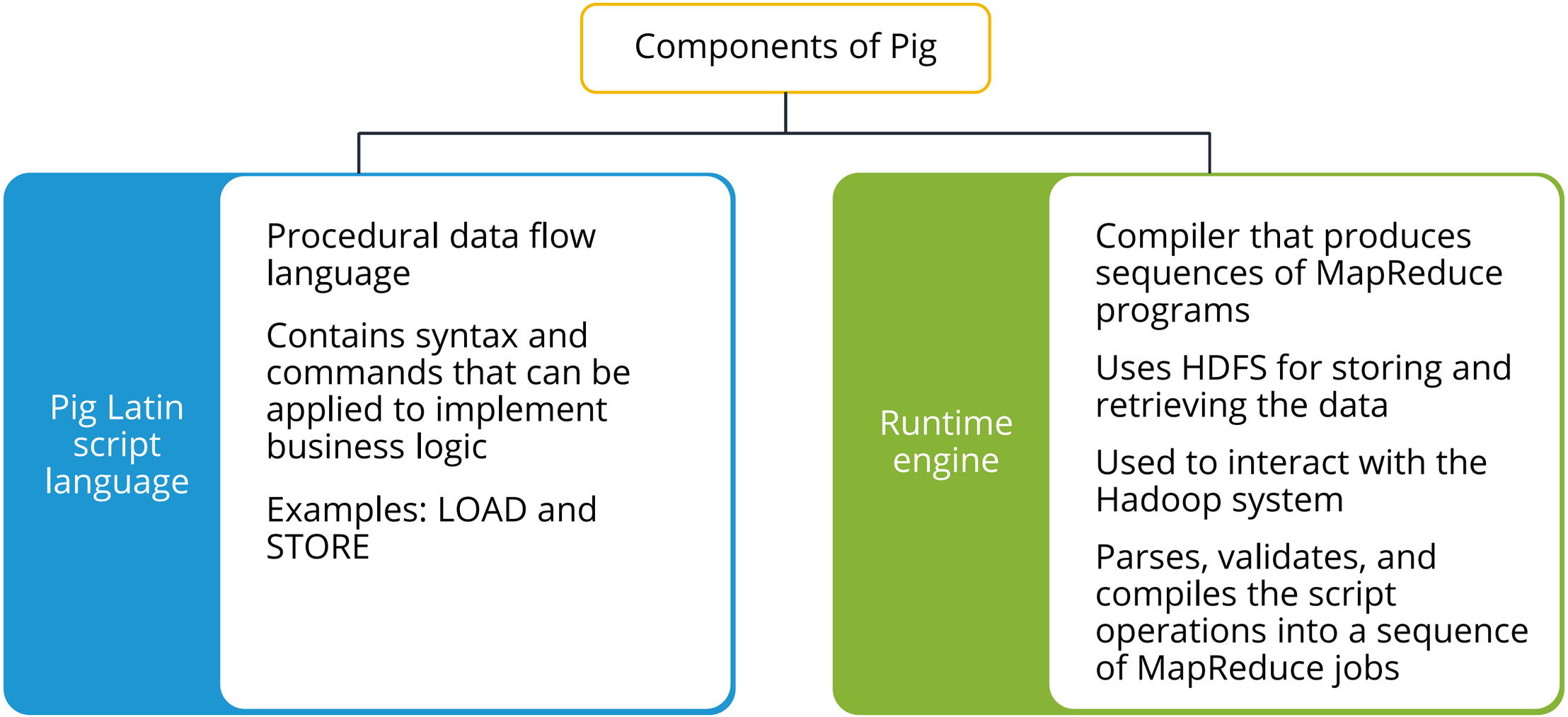
Write scripts to test a theory.

In the data factory, data may not be in a standardized state.



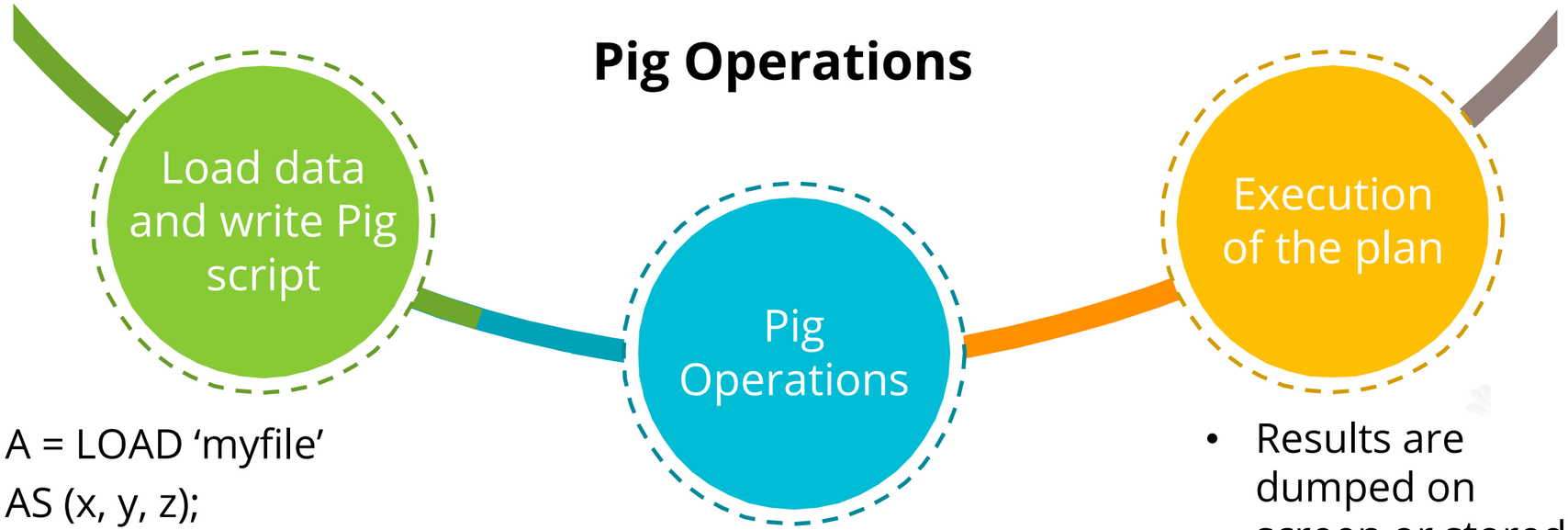
Pig supports data with partial or unknown schemas, and supports semi-structured or structured data.

Components of Pig



How Pig Works

Pig's operation can be explained in the following 3 stages:



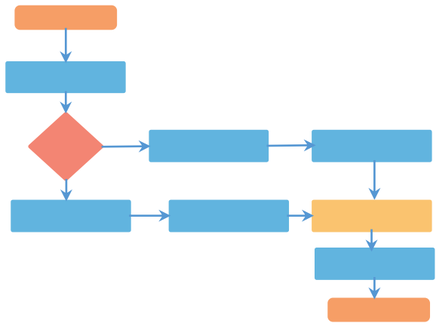

```
A = LOAD 'myfile'
AS (x, y, z);
B = FILTER A by x > 0;
C = GROUP B BY x;
D = FOREACH A GENERATE
x, COUNT(B);
STORE D INTO 'output';
```

- Parses and checks the script
- Optimizes the scripts
- Plans execution
- Submits to Hadoop
- Monitors job progress

- Results are dumped on screen or stored in HDFS

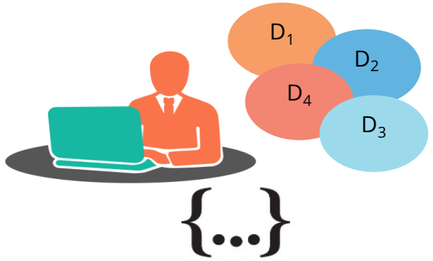
Salient Features

Developers and analysts like to use Pig as it offers many features.



Step-by-Step Procedural Control

Schemas can be Assigned Dynamically



Supports UDFs and Data Types

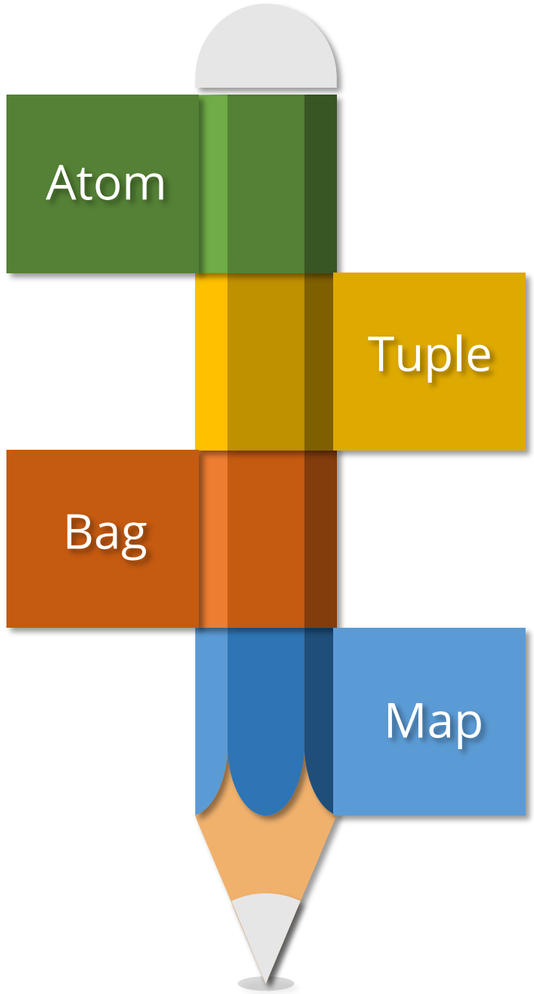
DATA AND
ARTIFICIAL INTELLIGENCE

Pig Data Model

Data Model

As part of its data model, Pig supports four basic types:

A simple atomic value
Example: ‘Mike’



A sequence of fields that can be of any data type
Example: (‘Mike’,43)

A collection of tuples of potentially varying structures that can contain duplicates
Example: {(‘Mike’), (‘Doug’, (43, 45))}

An associative array; the key must be a chararray, but the value can be of any type.
Example: [name#Mike,phone#5551212]

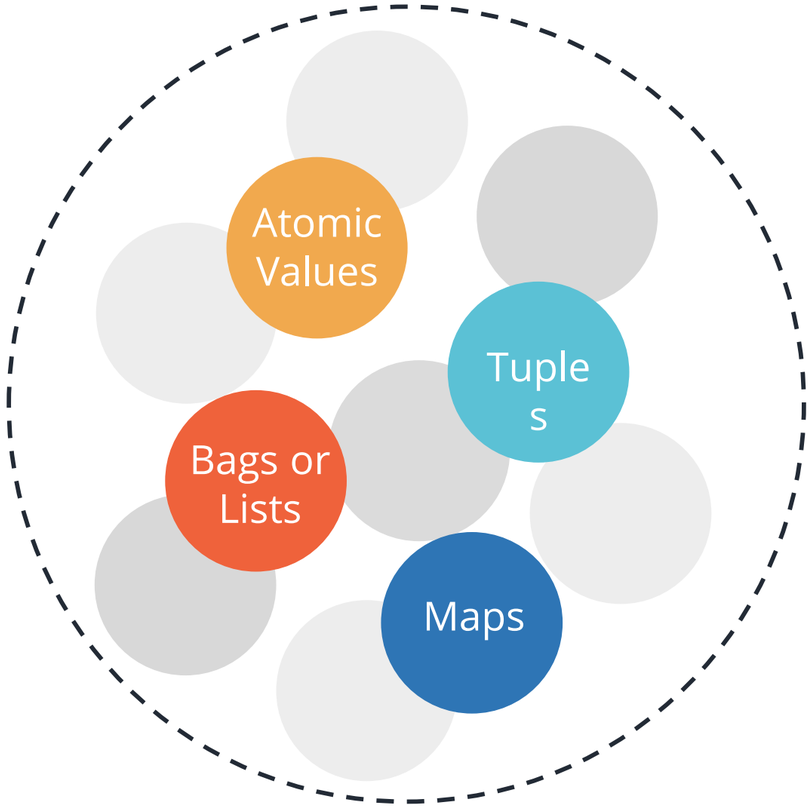
Data Model

- By default, Pig treats undeclared fields as ByteArrays.
- Pig can infer a field's type based on:
  - Use of operators that expect a certain type of field
  - User Defined Functions (UDFs) with a known or explicitly set return type
  - Schema information provided by a LOAD function or explicitly declared using an AS clause

Type conversion is lazy; the data type is enforced at execution only.

Nested Data Model

Pig Latin has a fully-nestable data model.



Advantages of nested data model

- More natural to programmers than flat tuples
- Avoids expensive joins

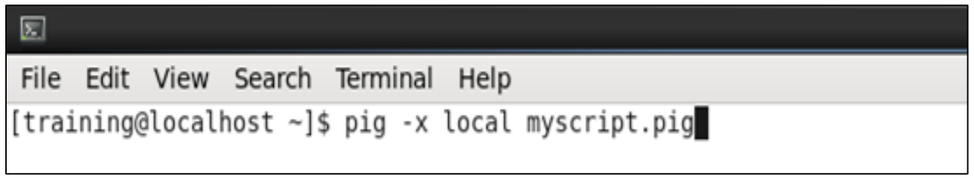
Pig Execution Modes

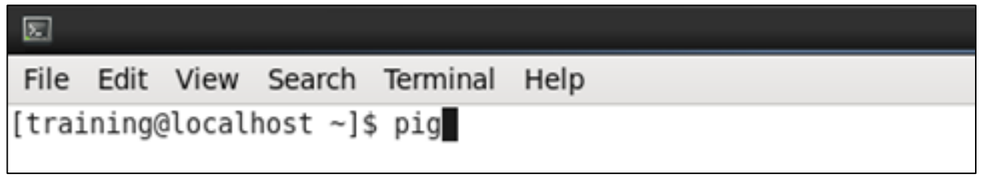
Pig works in two execution modes:

Pig execution modes

Local mode: Pig depends on the OS file system.

MapReduce mode: Pig relies on HDFS.



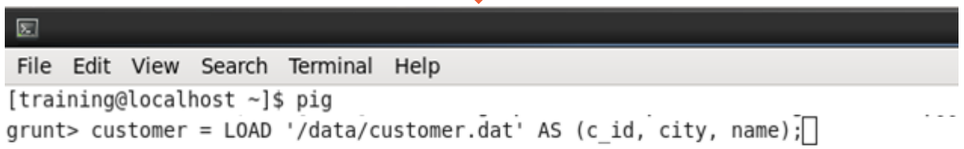


Pig Interactive Modes

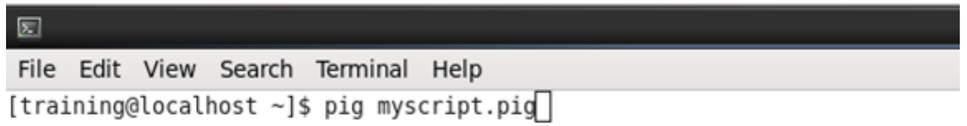
Pig Latin program can be written in two interactive modes:

Pig Interactive Modes

Interactive mode: A line by line code is written and executed.



Batch mode: A file containing Pig scripts is created and executed in a batch.



Pig vs. SQL

The differences between Pig and SQL are given below:

Difference	 Pig	 SQL
Definition	Scripting language used to interact with HDFS	Query language used to interact with databases
Query Style	Step-by-step	Single block
Evaluation	Lazy evaluation	Immediate evaluation
Pipeline Splits	Pipeline splits are supported	Requires the join to be run twice or materialized as an intermediate result

Pig vs. SQL: Example

Track customers in Texas who spend more than $2,000

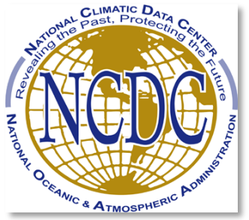
SQL	Pig

SELECT c_id , SUM(amount) AS CTotal
FROM customers c
JOIN sales s ON c.c_id = s.c_id
WHERE c.city = ‘Texas'
GROUP BY c_id
HAVING SUM(amount) > 2000
ORDER BY CTotal DESC
	customer = LOAD '/data/customer.dat' AS (c_id,name,city);
	sales = LOAD '/data/sales.dat' AS (s_id,c_id,date,amount);
	salesBLR = FILTER customer BY city == ‘Texas';
	joined= JOIN customer BY c_id, salesTX BY c_id;
	grouped = GROUP joined BY c_id;
	summed= FOREACH grouped GENERATE GROUP,
	SUM(joined.salesTX::amount);
	spenders= FILTER summed BY $1 > 2000;
	sorted = ORDER spenders BY $1 DESC;
	DUMP sorted;

Getting Datasets for Pig Development

Use the following URLs to download different datasets for Pig development:

Datasets	URL
Books	www.gutenberg.org (war_and_peace.txt)
Wikipedia database	http://dumps.wikimedia.org/enwiki/
Open data base from Amazon S3 data	https://aws.amazon.com/public-data-sets/
Open database from National climate data	http://cdo.ncdc.noaa.gov/qclcd_ascii/



DATA AND
ARTIFICIAL INTELLIGENCE

Pig Operations

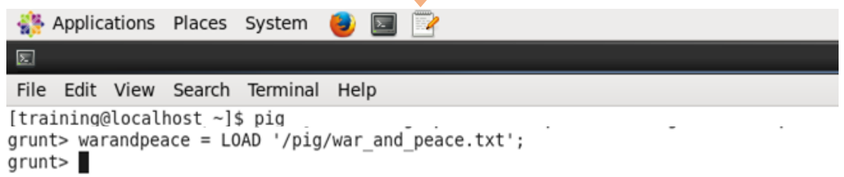
Loading and Storing Methods

Loading refers to loading relations from files in the Pig buffer.

Storing refers to writing outputs to the file system.

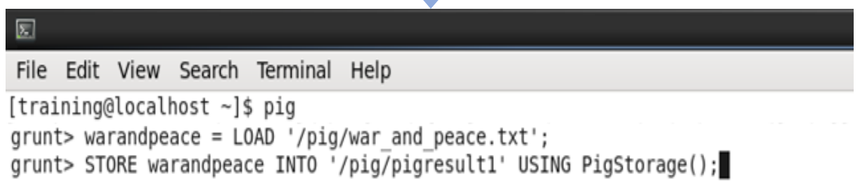
Load Data

Keyword:
LOAD



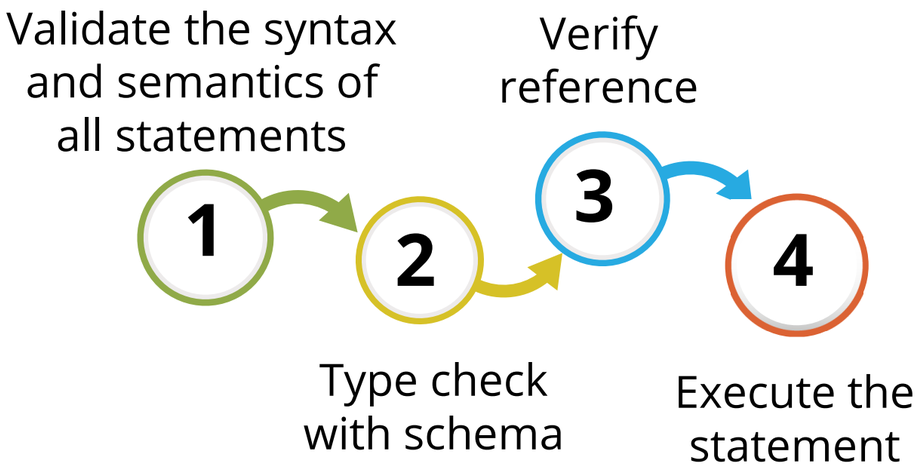
Store Data

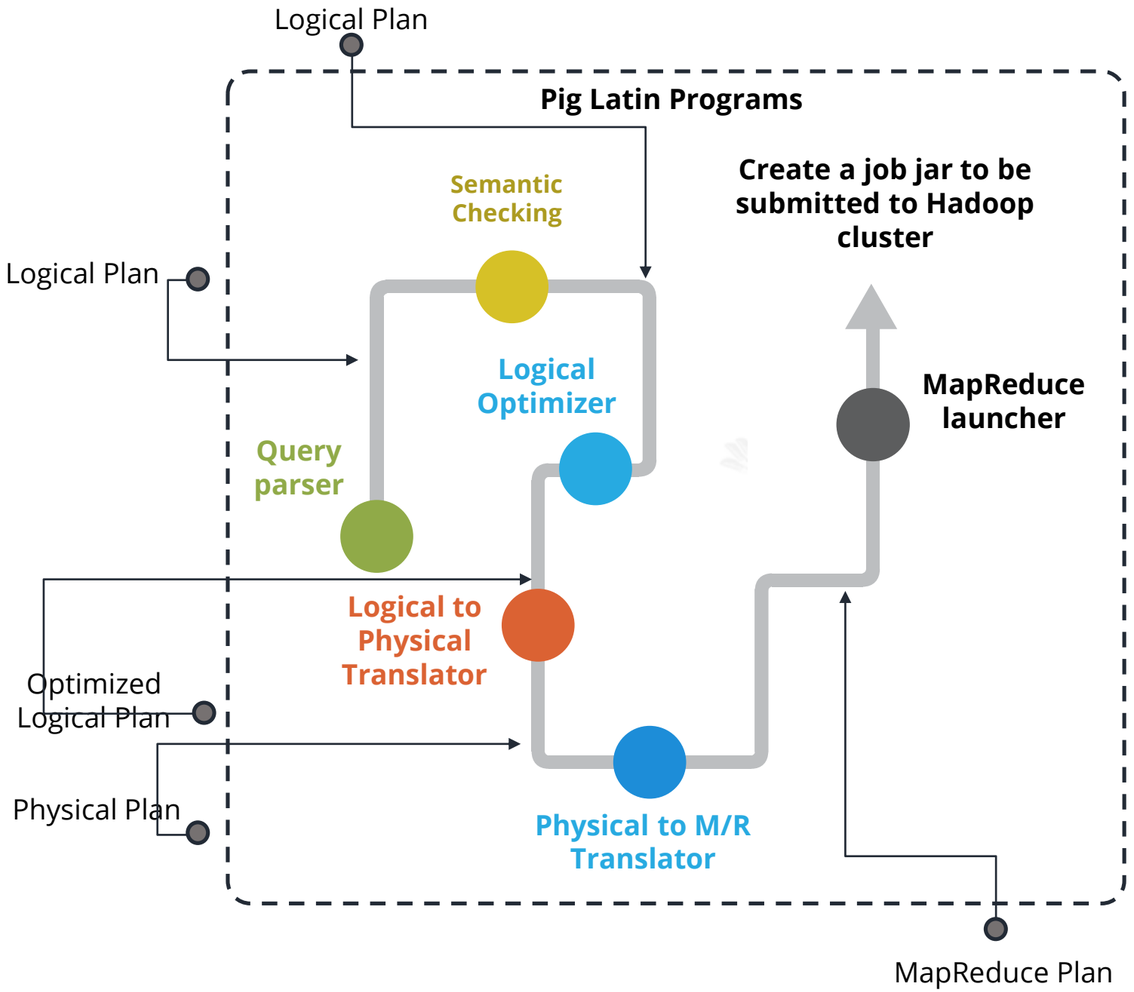
Keyword: STORE



Script Interpretation

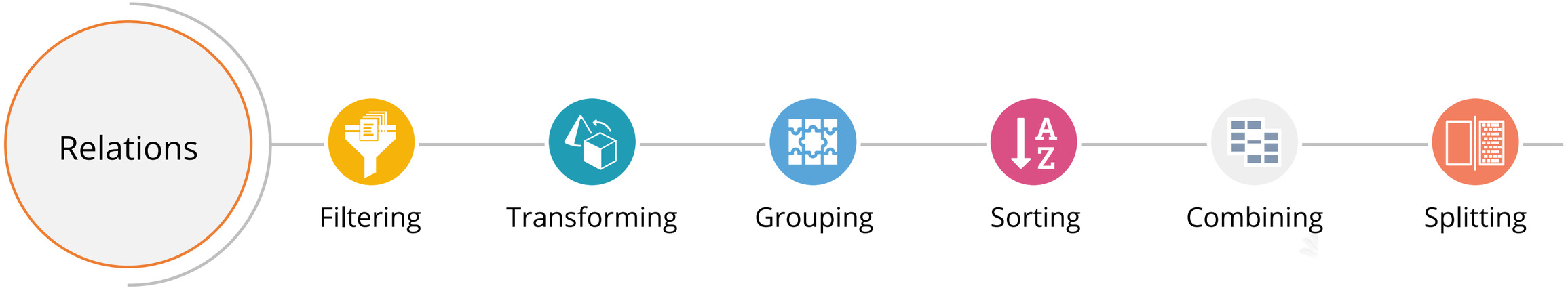
Pig processes Pig Latin statements in the following manner:





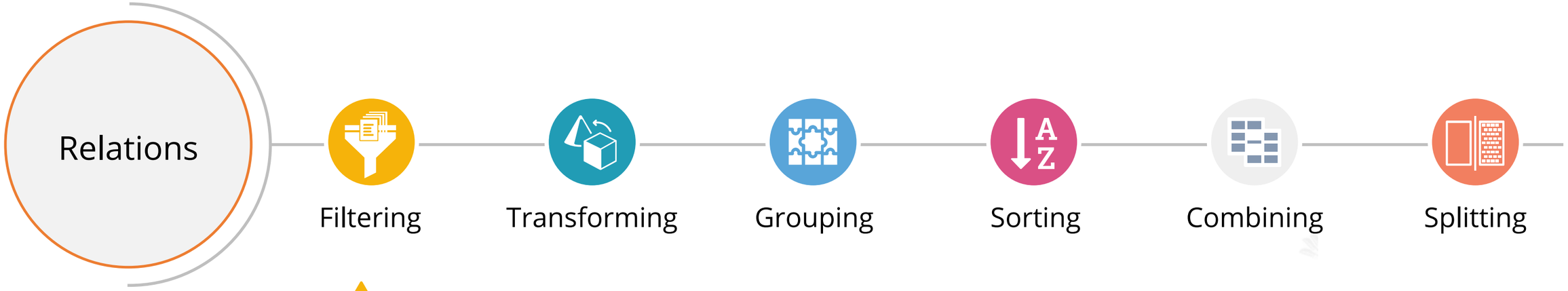
Various Relations Performed by Developers

Some of the relations performed by Big Data and Hadoop Developers are as follows:



Various Relations Performed by Developers

Some of the relations performed by Big Data and Hadoop Developers are as follows:

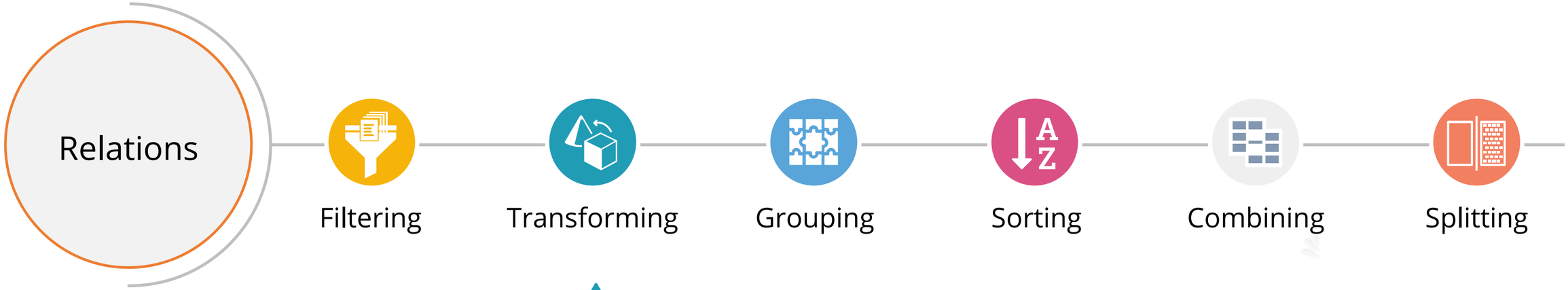


Filtering can be defined as filtering of data based on a conditional clause such as grade and pay.

```
training@localhost:~
File  Edit  View  Search  Terminal  Help
[training@localhost ~]$ pig
grunt> FILTER mbaresults by grade = 'A';
```

Various Relations Performed by Developers

Some of the relations performed by Big Data and Hadoop Developers are as follows:

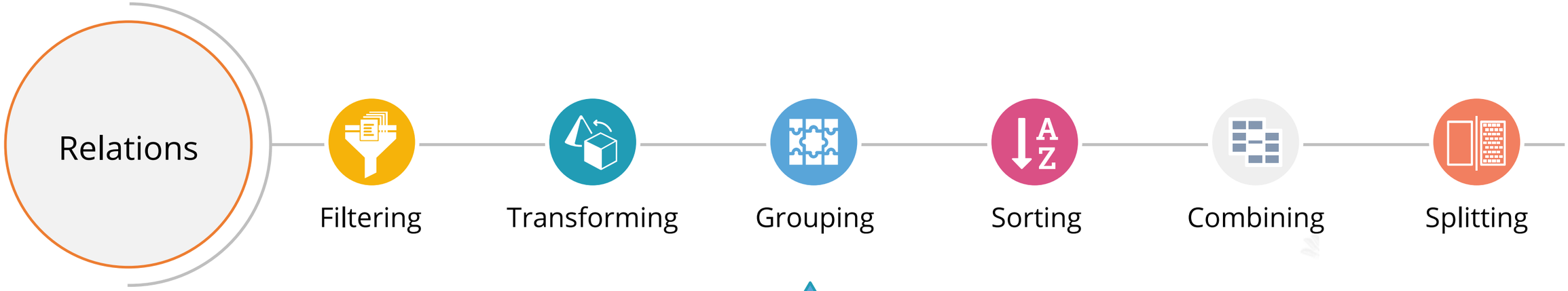


Transforming refers to making data presentable for the extraction of logical data.

```
training@localhost:~
File  Edit  View  Search  Terminal  Help
[training@localhost ~]$ pig
grunt> transform = FOREACH book GENERATE FLATTEN(TOKENIZE(lines)) as word;
```

Various Relations Performed by Developers

Some of the relations performed by Big Data and Hadoop Developers are as follows:

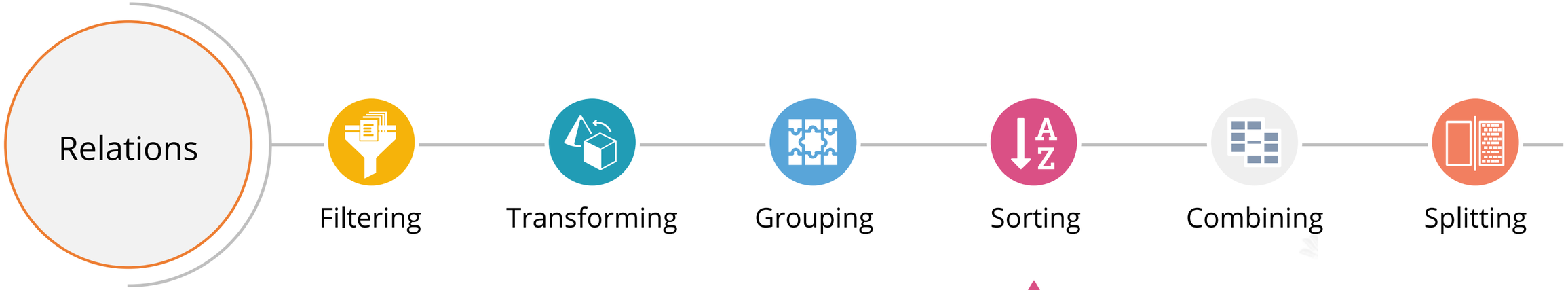


Grouping refers to generating a group of meaningful data.

```
training@localhost:~
File  Edit  View  Search  Terminal  Help
[training@localhost ~]$ pig
grunt> grouped = GROUP words by words;
```

Various Relations Performed by Developers

Some of the relations performed by Big Data and Hadoop Developers are as follows:

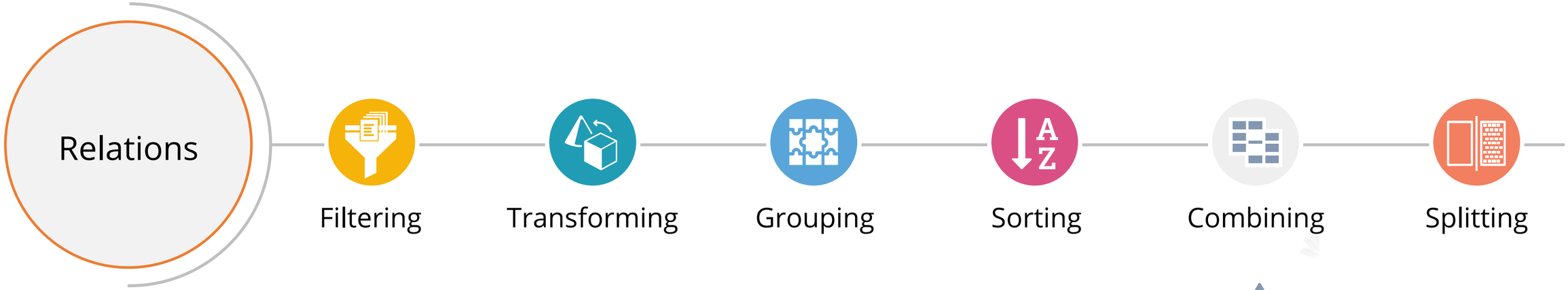


Sorting of data refers to arranging the data in either ascending or descending order.

```
training@localhost:~
File  Edit  View  Search  Terminal  Help
[training@localhost ~]$ pig
grunt> sorted = ORDER words by $1 DESC;
```

Various Relations Performed by Developers

Some of the relations performed by Big Data and Hadoop Developers are as follows:

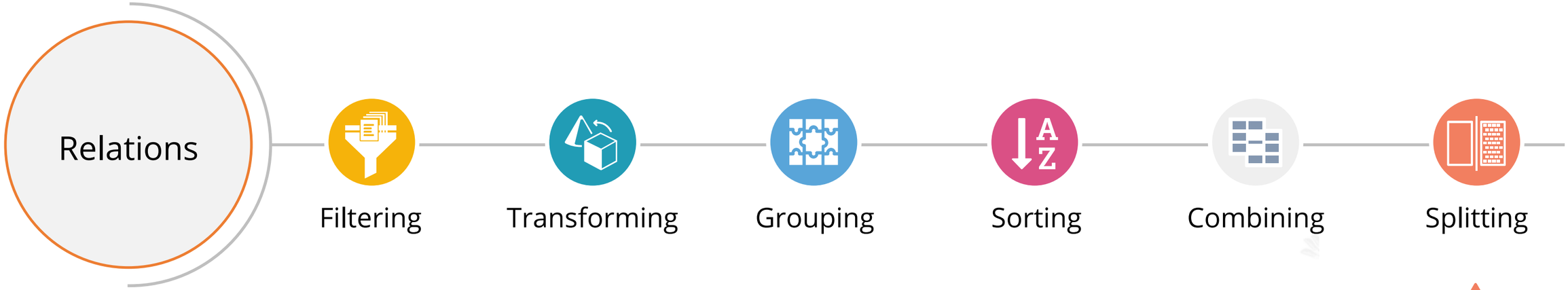


Combining refers to performing a union operation on the data stored in the variable.

```
training@localhost:~
File  Edit  View  Search  Terminal  Help
[training@localhost ~]$ pig
grunt> bookscombined = UNION book1, book2;
```

Various Relations Performed by Developers

Some of the relations performed by Big Data and Hadoop Developers are as follows:



Splitting refers to separating data that has logical meaning.

```
training@localhost:~
File  Edit  View  Search  Terminal  Help
[training@localhost ~]$ pig
grunt> SPLIT bookscombined INTO book1 IF SUBSTRING(isbn,4,6) == '1234', book2 IF SUBSTRING(isbn,4,6) == '3456';
```

Various Pig Commands

These are some of the Pig commands:

Pig command	Functions
load	Reads data from system
Store	Writes data to file system
foreach	Applies expressions to each record and outputs one or more records
filter	Applies predicate and removes records that do not return true
Group or cogroup	Collects records with the same key from one or more inputs
join	Joins two or more inputs based on a key
order	Sorts records based on a key
distinct	Removes duplicate records
union	Merges data sets
split	Splits data into two or more sets, based on filter conditions
stream	Sends all records through a user-provided binary
dump	Writes output to stdout
limit	Limits the number of records

Assisted Practice

MapReduce and Pig

Duration: 15 mins

Problem Statement: In this demonstration, you will analyze web log data using MapReduce and solve various real-world KPIs.

Access: Click on the **Practice Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

Assisted Practice

Apache Pig

Duration: 15 mins

Problem Statement: In this demonstration, you will analyze sales data and solve KPIs using Pig.

Access: Click on the **Practice Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

Unassisted Practice

Apache Pig

Duration: 15 mins

Problem Statement: Use pig to count the number of words from a text file.

Access: Click on the **Practice Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

Steps to Perform

- **Pig**

  $ pig -x local

  grunt> lines = LOAD "sampletextfile.txt" AS (line:chararray);

  grunt> words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;

  grunt> DUMP words

  grunt> grouped_words = GROUP words BY word;

  grunt> DUMP grouped_words

  grunt> wordcount = FOREACH grouped_words GENERATE group, COUNT(words);

  grunt> DUMP wordcount

Key Takeaways

You are now able to:

- Perform distributed processing in MapReduce
- Understand issues with distributed processing and how MR reduces them
- Implement MapReduce paradigm, divide-and-conquer
- Perform Word Count program - Hello World of big data
- Understand Pig
- Implement Pig Latin commands

Knowledge Check

Knowledge Check

1

Which of the following commands is used to start Pig in MapReduce mode?

a. Pig

b. Pig -x MapReduce

c. Pig -x local

d. Both Pig and Pig -x MapReduce

Knowledge Check

1

Which of the following commands is used to start Pig in MapReduce mode?

a. Pig

b. Pig -x MapReduce

c. Pig -x local

d. Both Pig and Pig -x MapReduce

The correct answer is **d.**

Pig or Pig -x MapReduce command can be used to run Pig in MapReduce mode.

Knowledge Check

2

Which of the following commands is used to start Pig in local mode?

a. Pig -x local

b. Pig -x MapReduce

c. Pig

d. Both b and c

Knowledge Check

2

Which of the following commands is used to start Pig in local mode?

a. Pig -x local

b. Pig -x MapReduce

c. Pig

d. Both b and c

The correct answer is **a.**

Pig -x local command is used to start Pig in local mode.

Knowledge Check

3

How many phases exist in MapReduce?

a. 4

b. 5

c. 6

d. 2

Knowledge Check

3

How many phases exist in MapReduce?

a. 4

b. 5

c. 6

d. 2

The correct answer is **b.**

MapReduce consists of five phases: Map phase, Partition phase, Shuffle phase, Sort phase, and Reduce phase.

Knowledge Check

4

In how many ways can Pig Latin program be written?

a. 2

b. 4

c. 3

d. 5

Knowledge Check

4

In how many ways can Pig Latin program be written?

a. 2

b. 4

c. 3

d. 5

The correct answer is **a.**

Pig Latin program can be written in two ways.

Knowledge Check

5

Which of the following is the first step to build a MapReduce program?

a. Launch the job and monitor

b. Package the code

c. Determine the data

d. None of the above

Knowledge Check

5

Which of the following is the first step to build a MapReduce program?

a. Launch the job and monitor

b. Package the code

c. Determine the data

d. None of the above

The correct answer is **c.**

Determining the data is the first step to build a Mapreduce program.

Knowledge Check

6

Which of the following is a real-time use case of MapReduce?

a. Enterprise Analytics

b. Gaussian analysis

c. Search engine operations

d. All of the above

Knowledge Check

6

Which of the following is a real-time use case of MapReduce?

a. Enterprise Analytics

b. Gaussian analysis

c. Search engine operations

d. All of the above

The correct answer is **d.**

Enterprise analytics, Gaussian analysis, and search engine operations are real-time use cases of MapReduce.

Lesson-End Project

Problem Statement:

The US Department of Transport collects statistics from all the airlines, which include airline details, airport details, and flight journey details.
These airlines have global presence and they operate almost in every country.
Flight data can help to decide which airline provides better service and find the routes in which flights are getting delayed.

The data collected is present in the files: flight.csv, airline.csv, and aiport.csv.
You are hired as a big data consultant to provide important insights.

Your task is to write MapReduce jobs and provide insights on:

1. The number of flights that get canceled in a month, every year
2. The airline names and the number of times their flights were canceled and diverted
3. The number of times flights were delayed for each airline

DATA AND
ARTIFICIAL INTELLIGENCE

Thank You