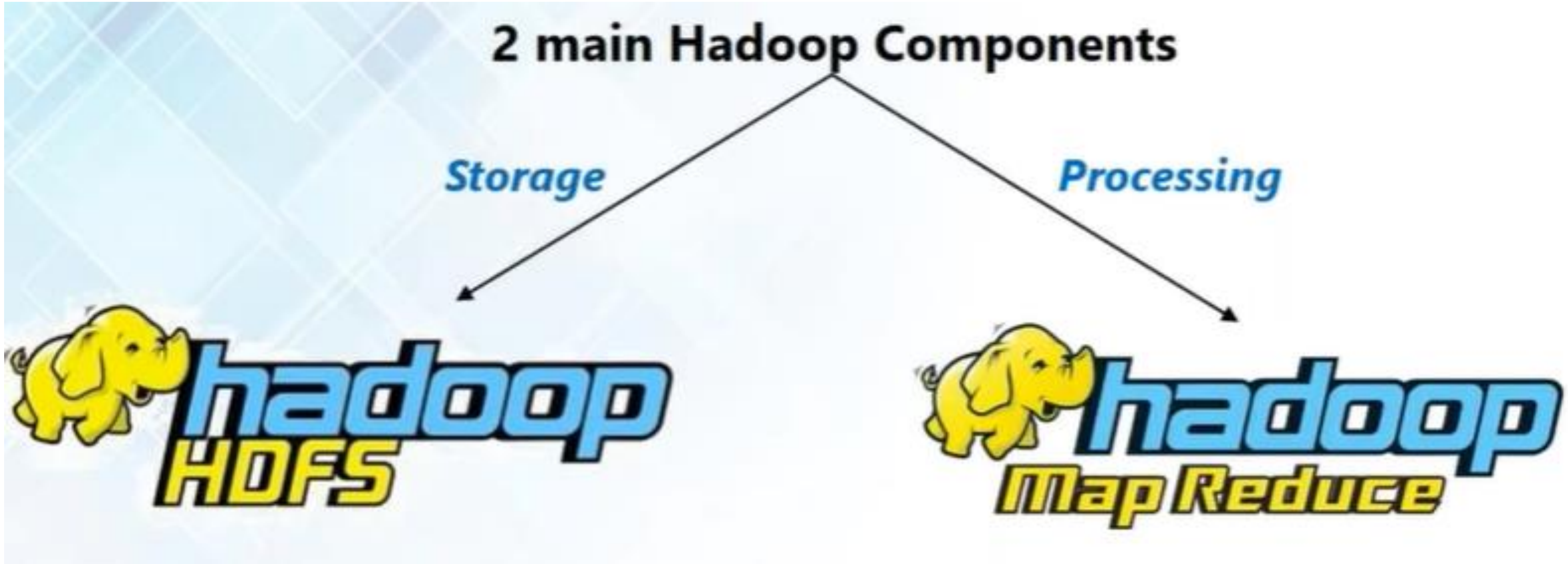
# MapReduce Part - 3

BY : DR.RASHMI L MALGHAN & MS. SHAVANTREVVA S BILAKERI

# AGENDA

1. What is Hadoop MapReduce?
2. MapReduce In Nutshell
3. Two Advantages of MapReduce
4. Hadoop MapReduce Approach with an Example
5. Hadoop MapReduce/YARN Components
6. YARN With MapReduce
7. Yarn Application Workflow

# Hadoop Main Components:

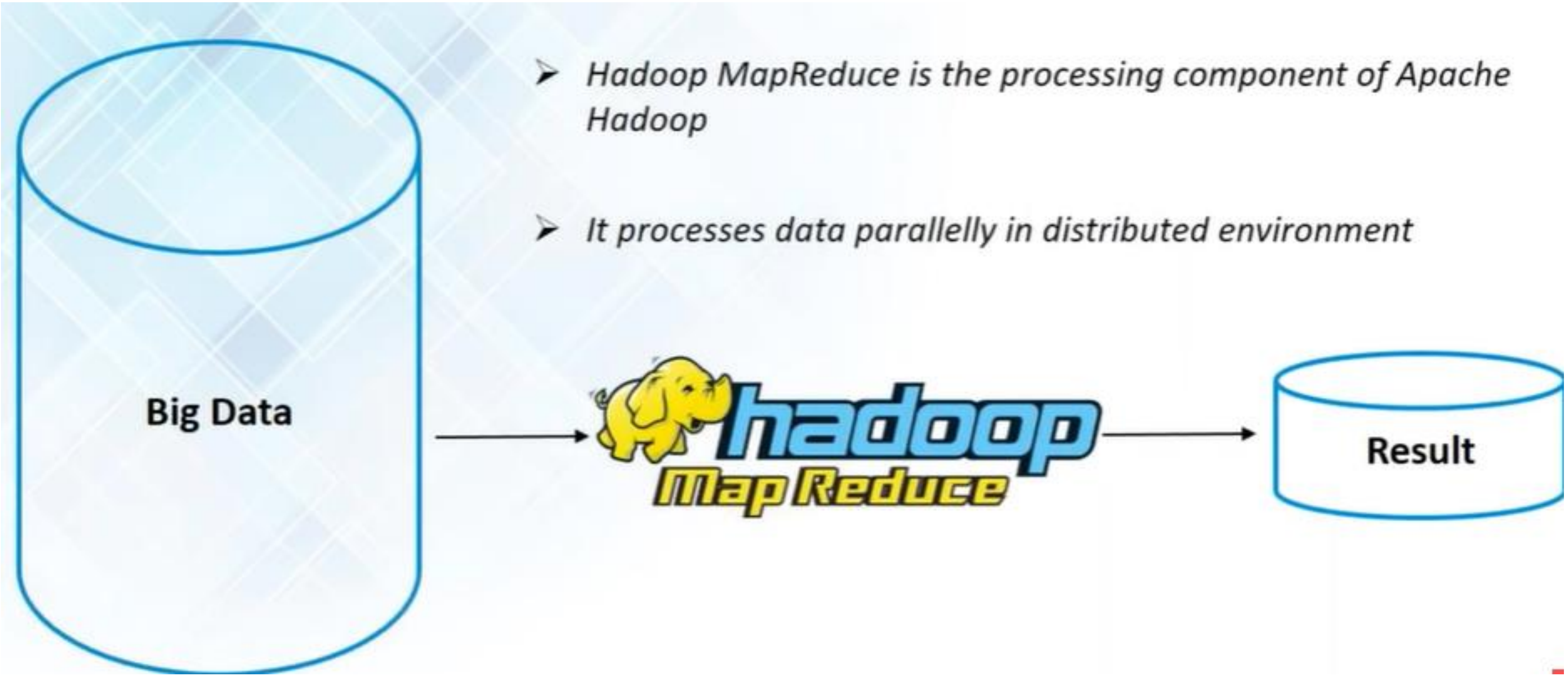


# Comparison: Convention Vs. MapReduce

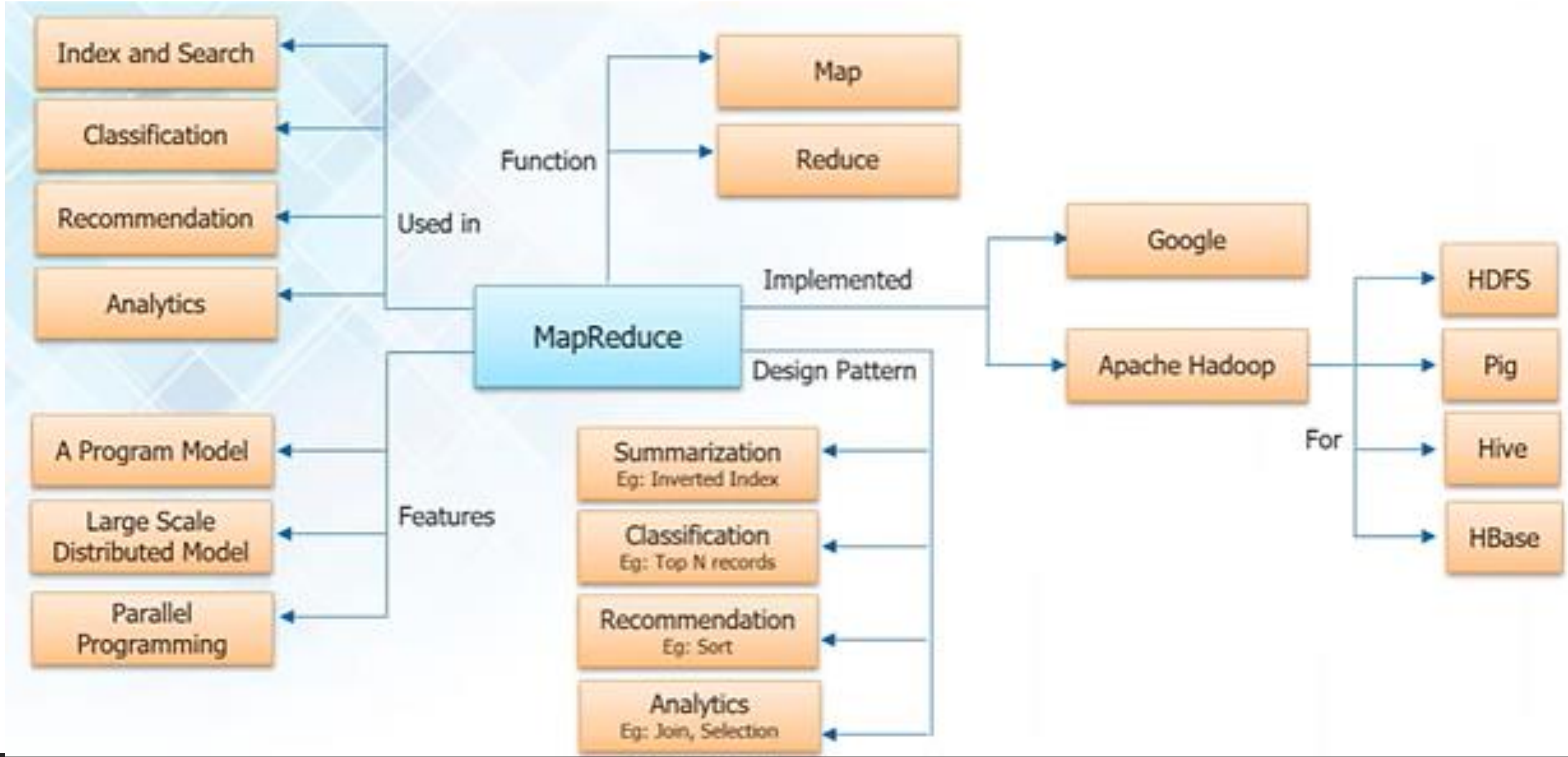
- **Conventional Approach**
  - **Single-machine** processing.
  - Suitable for **small datasets**.
  - Limited **scalability**.
  - May become a **bottleneck** for big data.

- **MapReduce Approach**
  - **Distributed** processing across a cluster.
  - Scalable for **large datasets**.
  - Handles **parallel processing** efficiently.
  - Tackles the **challenges of big data**

# MapReduce:


Hadoop MapReduce is the processing component of Apache Hadoop
It processes data parallelly in distributed environment
Big Data
hadoop
MapReduce
Result

# MapReduce - Nutshell



# Advantage 1: Parallel Processing


- In Hadoop data gets divided in to small chunks called as HDFS blocks.

  - Scalability
  - Fault Tolerance
  - Flexibility
  - Efficient Resource Utilization

# Advantage 2: Data Locality


- Processing – Master sends the logic to salves that require for its job processing.
- Smaller chunk of data is getting processed in multiple locations in parallel.
- It saves "time" & "network bandwidth"- required to move/transfer large amount of data from one point/location to another.
- Results will be sent back to master machines and sent to client machine

- **Data Locality -** MapReduce is processing at "Location" where data is stored rather bringing data to "centralised server"
- Client Sends/submits data – To Resource Manager decides – Data usually resides on "Nearest Data Node" to reduce the network bandwidth.
- Master = Name Node – In Hdfs [Storage] , Master – Consideration -resources manager - **Processing**

# MapReduce: Phases and Deamons

MapReduce Framework

Phases:

Map(): Converts input into Key Value pair

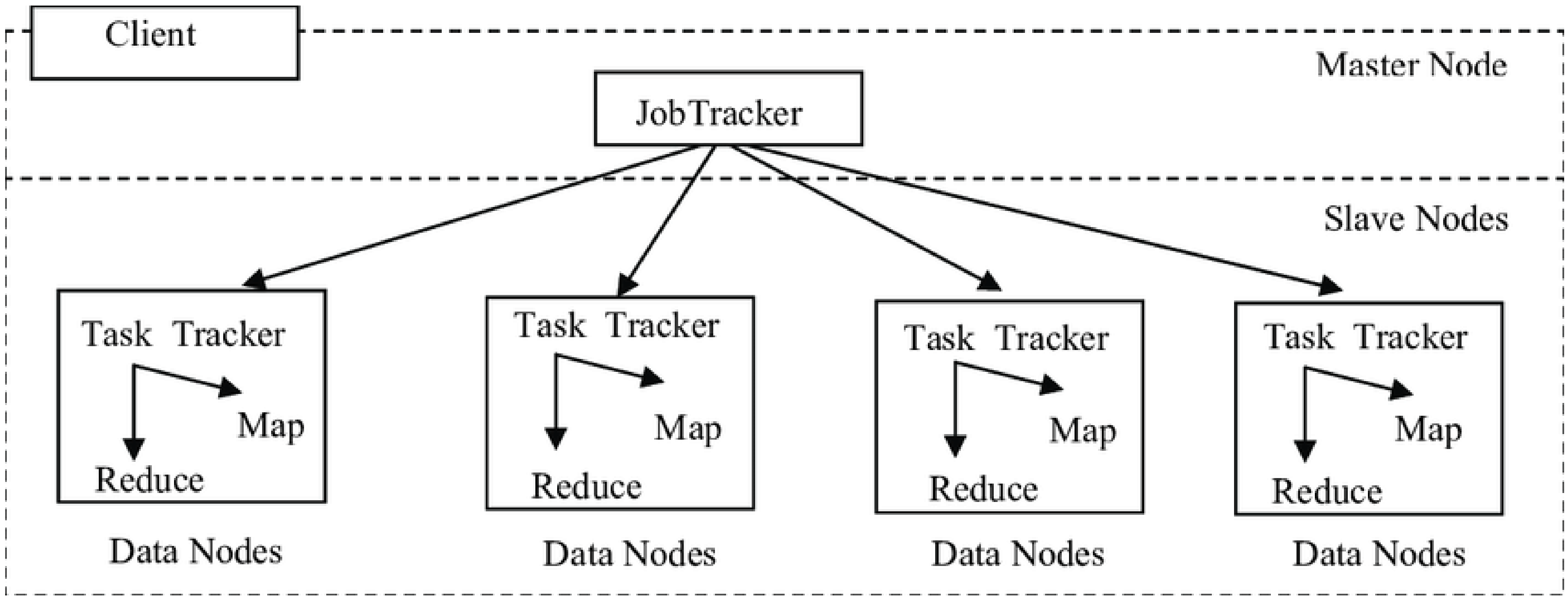
Reduce(): Combines output of mappers and produces a reduced result set.

Daemons:

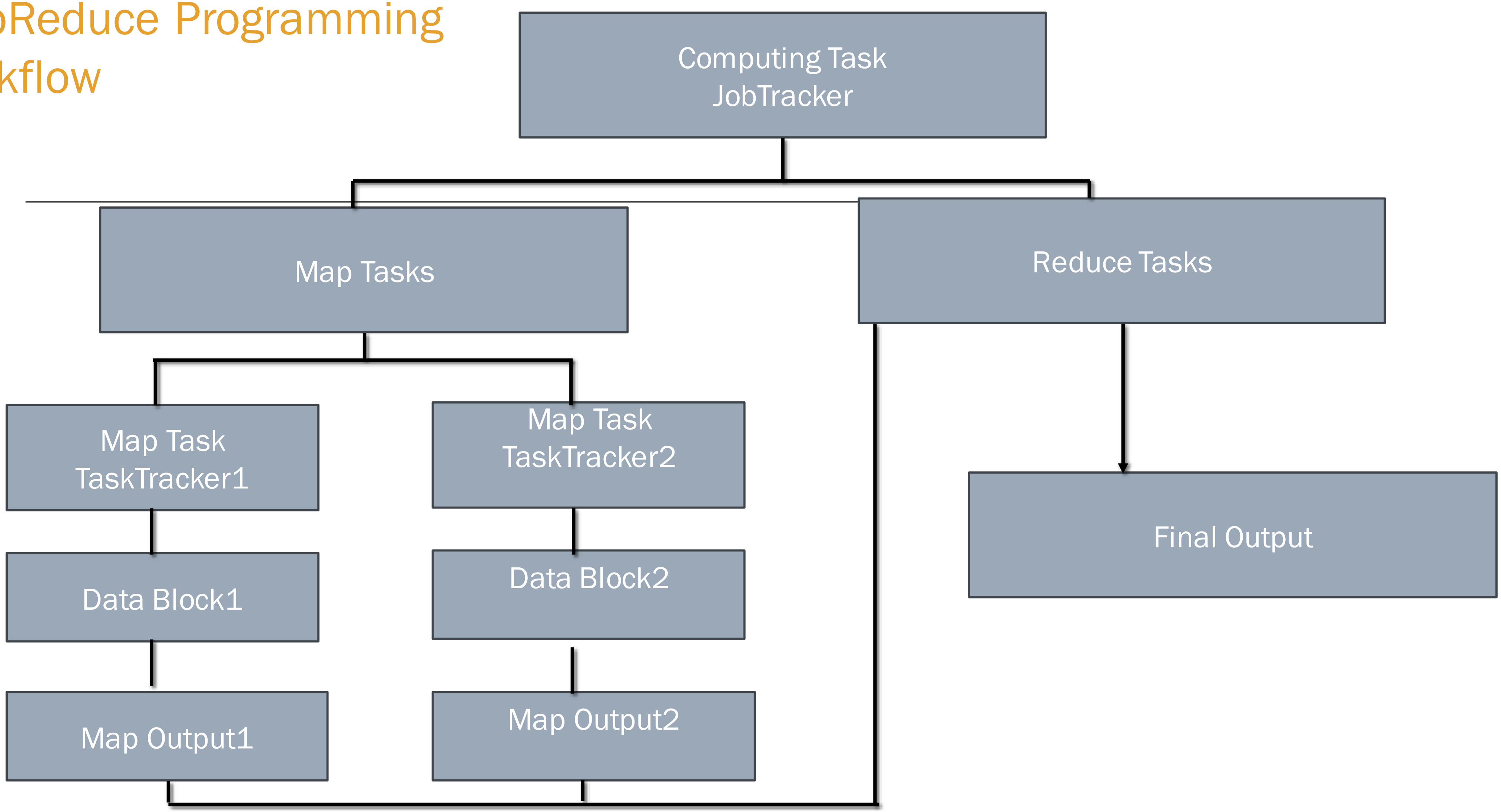
JobTracker: Master schedules task

TaskTracker: Slave executes task

# JobTracker and TaskTracker Interaction



# MapReduce Programming Workflow



# Phases in Map and Reduce Tasks

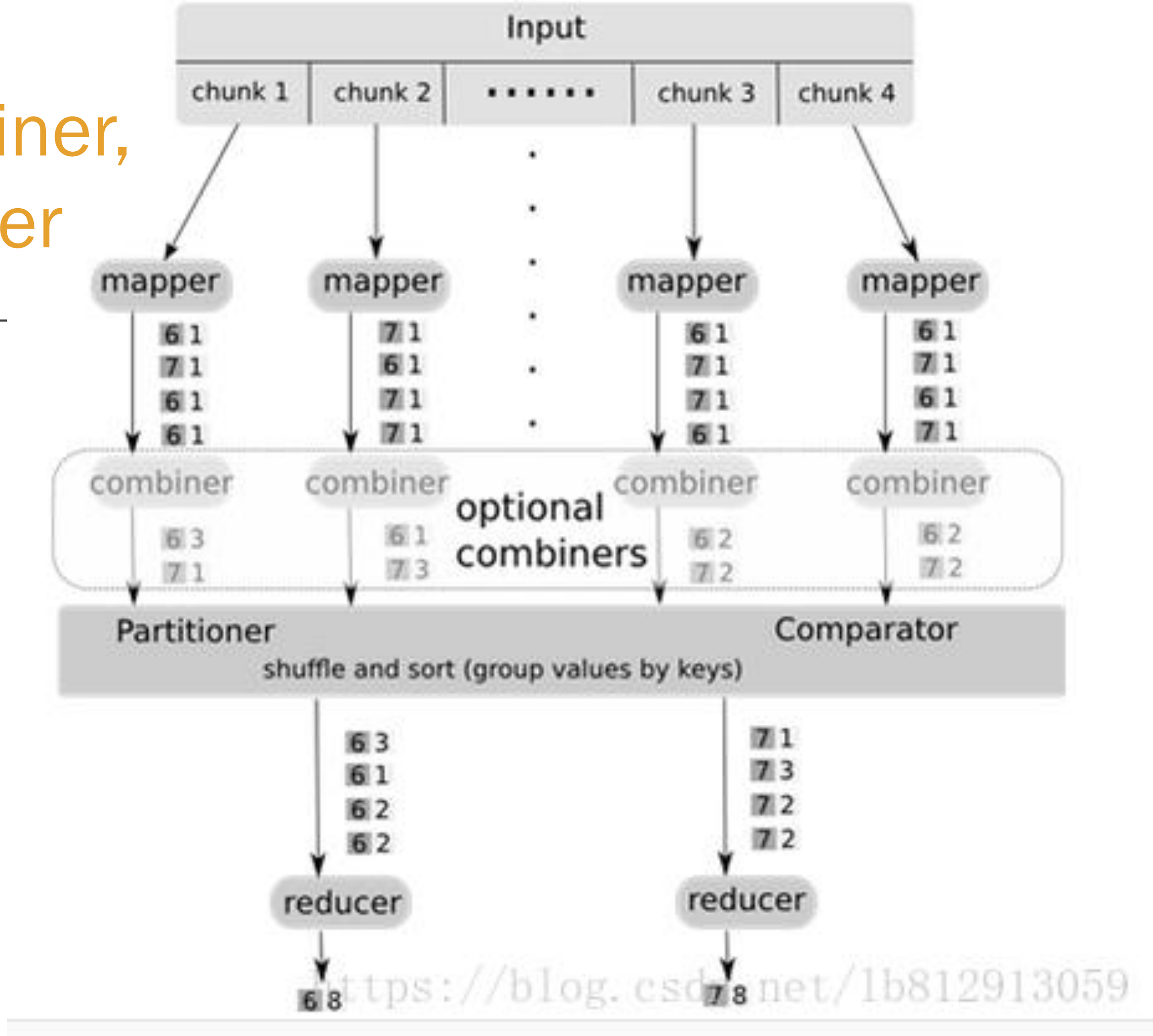
## Map Phases

1. Record Reader
2. Mapper
3. Combiner
4. Partitioner

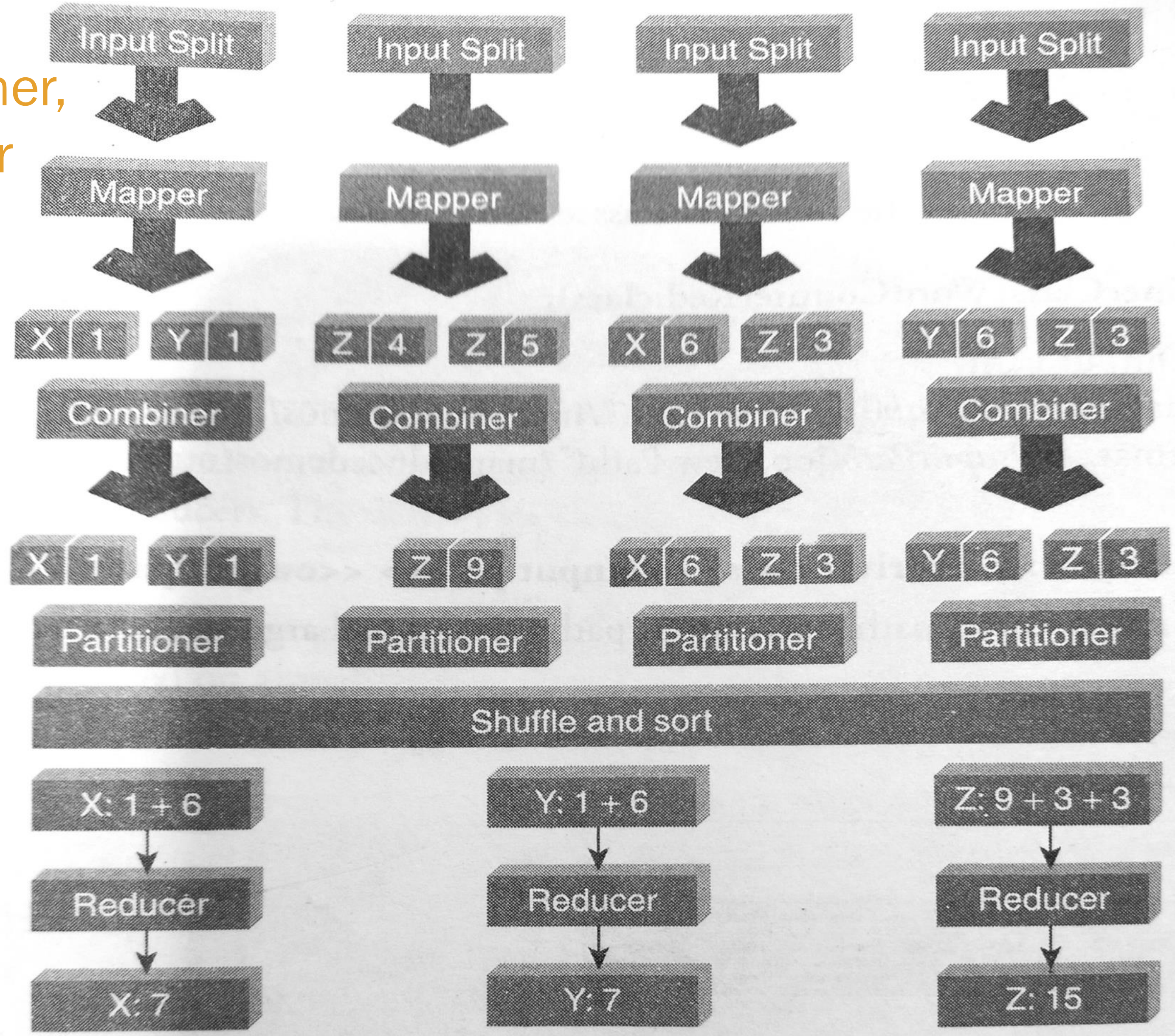
## Reduce Phases

1. Shuffle
2. Sort
3. Reducer
4. Output Format

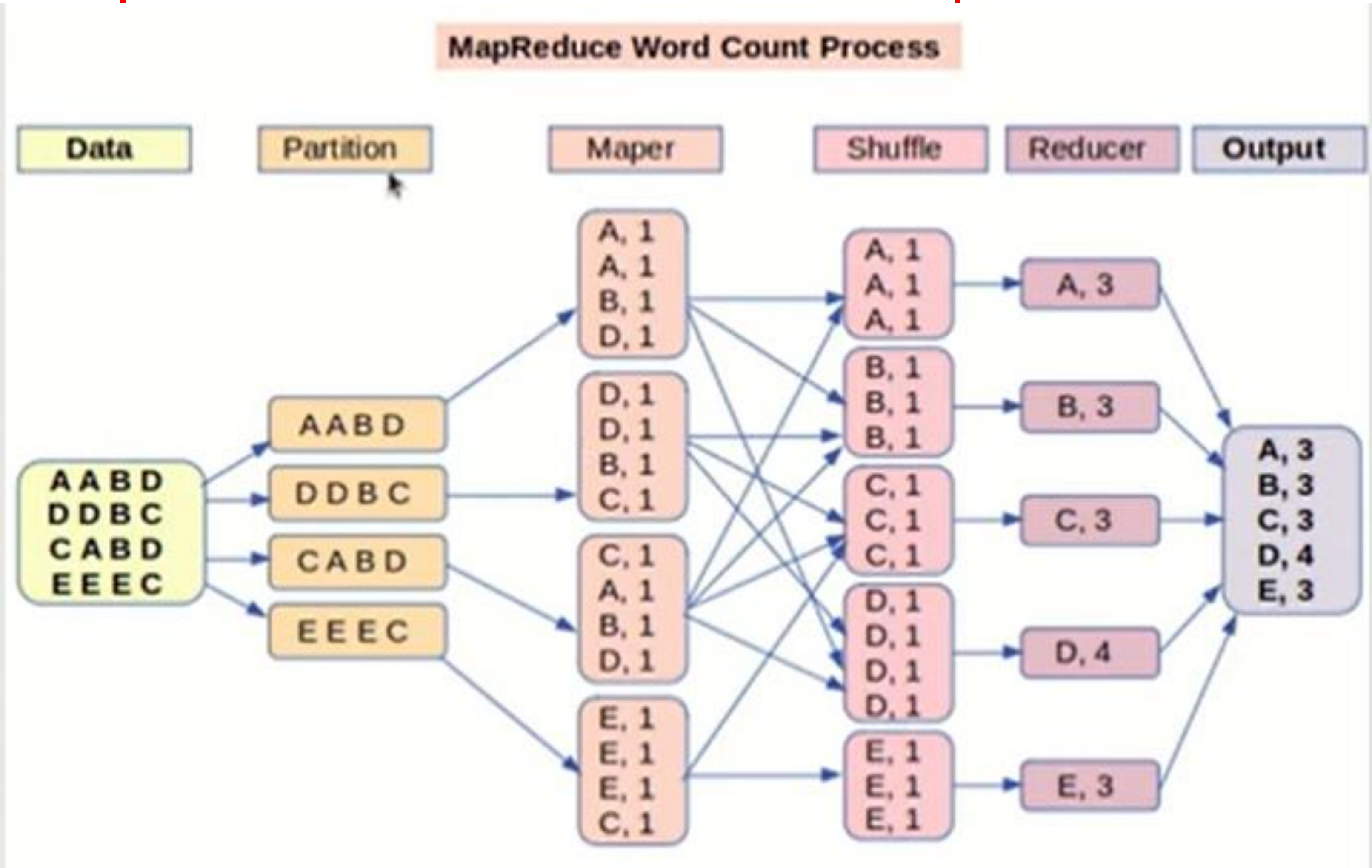
# Chores of Mapper, Combiner, Partitioner, and Reducer



# Chores of Mapper, Combiner, Partitioner, and Reducer



# MapReduce: Word count problem



# Word Count Program using MapReduce

**Step 1:** Create a file with the name *word_count_data.txt* and add some data to it

```
dikshant@dikshant-Inspiron-5567:~/Documents$ touch word_count_data.txt
dikshant@dikshant-Inspiron-5567:~/Documents$ nano word_count_data.txt
dikshant@dikshant-Inspiron-5567:~/Documents$ cat word_count_data.txt
geeks for geeks is best online coding platform
welcome to geeks for geeks hadoop streaming tutorial
dikshant@dikshant-Inspiron-5567:~/Documents$
```

**Step 2:** Create a **mapper.py** file that implements the mapper logic.

## Mapper.py

```python
#!/usr/bin/env python

# import sys because we need to read and write data to STDIN and
STDOUT
import sys

# reading entire line from STDIN (standard input)
for line in sys.stdin:
    # to remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()

    # we are looping over the words array and printing the word
    # with the count of 1 to the STDOUT
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        print '%s\t%s' % (word, 1)
```

**#!** is known as shebang and used for interpreting the script.

# Test **mapper.py** locally

```
cat word_count_data.txt | python mapper.py
```

```
dikshant@dikshant-Inspiron-5567:~/Documents$ cat word_count_data.txt | python mapper.py
geeks	1
for	1
geeks	1
is	1
best	1
online	1
coding	1
platform	1
welcome	1
to	1
geeks	1
for	1
geeks	1
hadoop	1
streaming	1
tutorial	1
dikshant@dikshant-Inspiron-5567:~/Documents$
```

**Step 3:** Create a ***reducer.py*** file that implements the reducer logic.

```
Content: #!/usr/bin/python3

"reducer.py"
import sys
current_word = None
current_count = 0

for line in sys.stdin:
    # remove leading and trailing whitespaces
    line = line.strip()
    # parse the input we got from mapper.py
    word, count = line.split('\t')
    count = int(count)

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print ('%s\t%s' % (current_word, current_count))
        current_count = count
        current_word = word
if current_word == word:
    print ('%s\t%s' % (current_word, current_count))
```

## Test mapper and reduce

cat word_count_data.txt | python mapper.py | sort  | python reducer.py

```
dikshant@dikshant-Inspiron-5567:~/Documents$ cat word_count_data.txt | python mapper.py | sort -k1,1 | pyt
hon reducer.py
best    1
coding  1
for     2
geeks   4
hadoop  1
is      1
online  1
platform        1
streaming       1
to      1
tutorial        1
welcome 1
dikshant@dikshant-Inspiron-5567:~/Documents$
```

```
dikshant@dikshant-Inspiron-5567:~/Documents$ cat word_count_data.txt | python mapper.py | sort -k1,1 | python reducer.py
best    1
coding  1
for     2
geeks   4
hadoop  1
is      1
online  1
platform        1
streaming       1
to      1
tutorial        1
welcome 1
dikshant@dikshant-Inspiron-5567:~/Documents$
```

**Step 4:** Now let’s start all our Hadoop daemons with the command:.

**start-all.sh**

hdfs dfs -mkdir /word_count_in_python

hdfs dfs -copyFromLocal /home/ssb/Documents/word_count_data.txt /word_count_in_python

chmod 777 mapper.py reducer.py

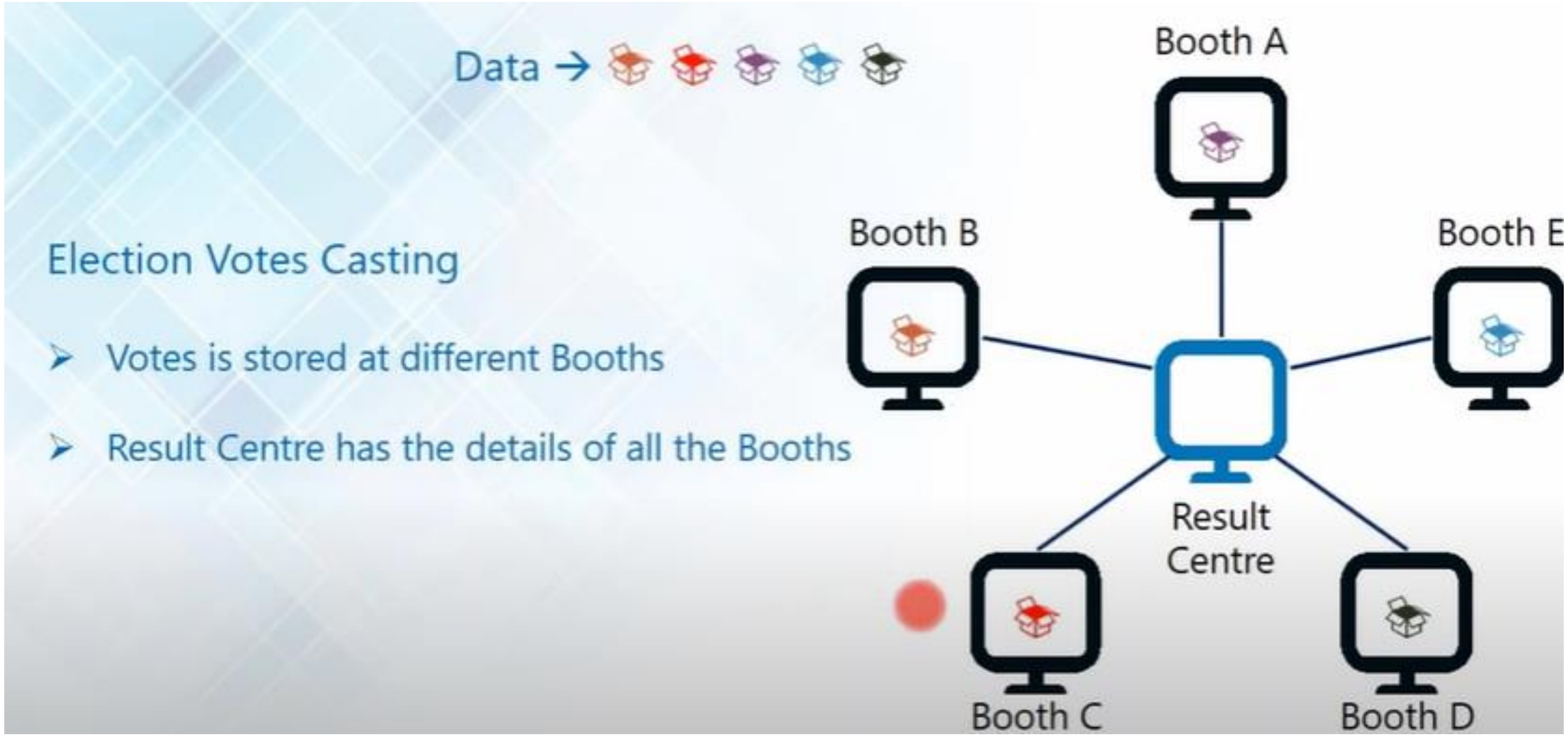
---

**Step 5:** Now download the latest **hadoop-streaming jar file.** Then place, this Hadoop,-streaming jar file to a place from you can easily access it.

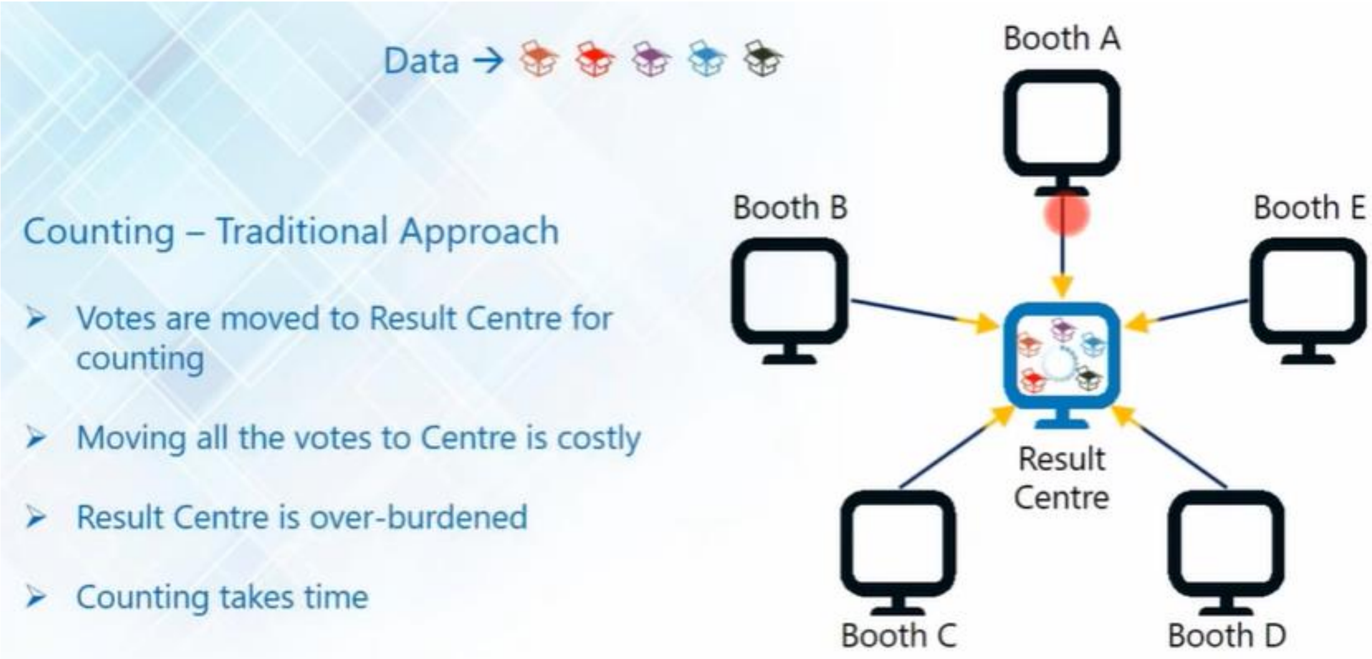
**Step6: Run our python files with the help of the Hadoop streaming utility**

```
hadoop jar /home/ssb/Documents/hadoop-streaming-
2.7.3.jar \

> -input /word_count_in_python/word_count_data.txt \

> -output /word_count_in_python/output \

> -mapper /home/ssb/Documents/mapper.py \

> -reducer /home/ssb/Documents/reducer.py
```

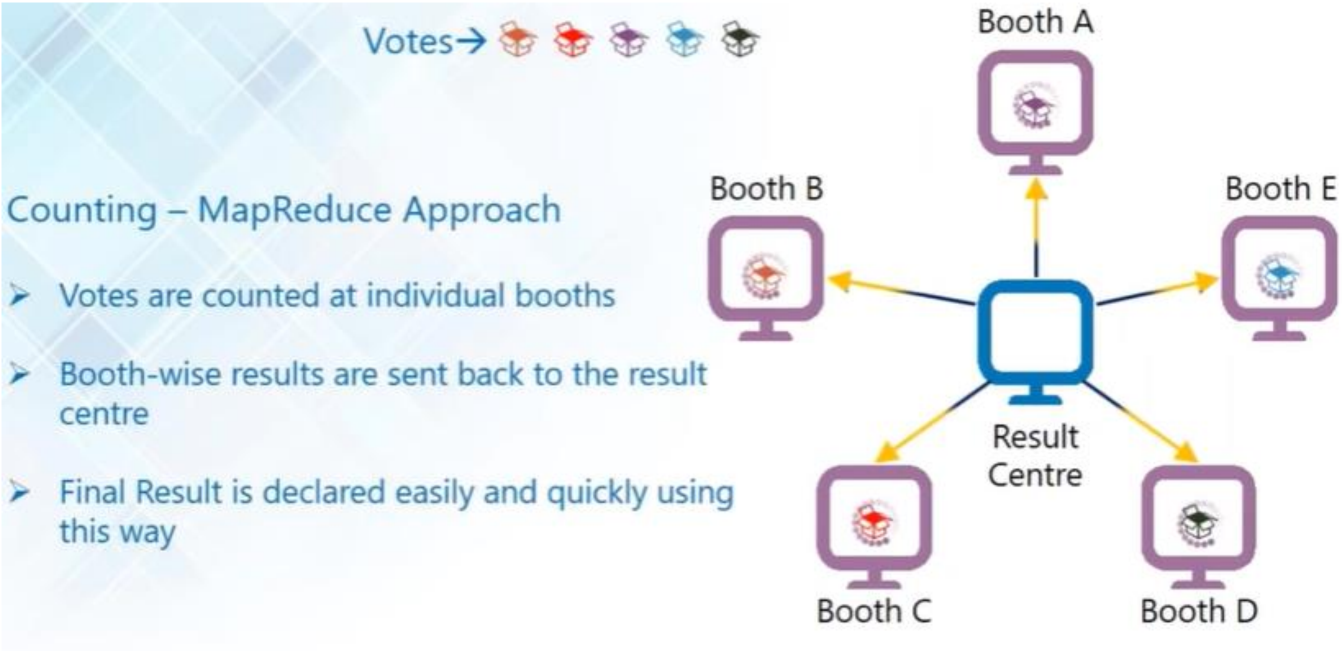
# Example: Election Votes Counting

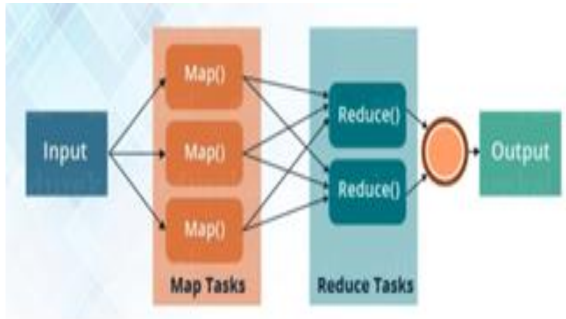


# Traditional Approach: Election Votes Counting



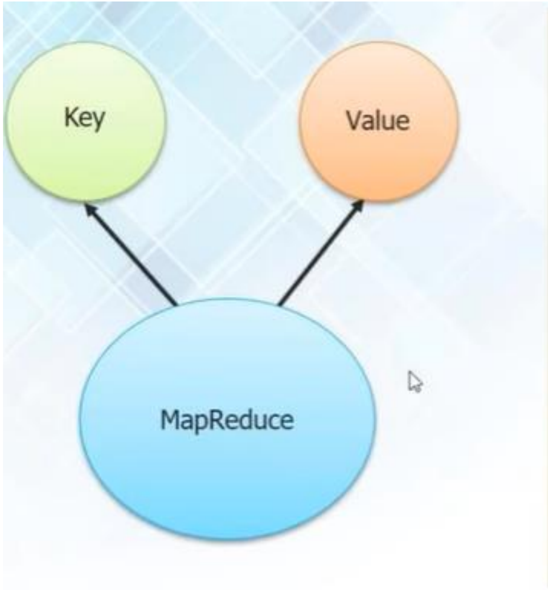
# MapReduce Approach: Election Votes Counting

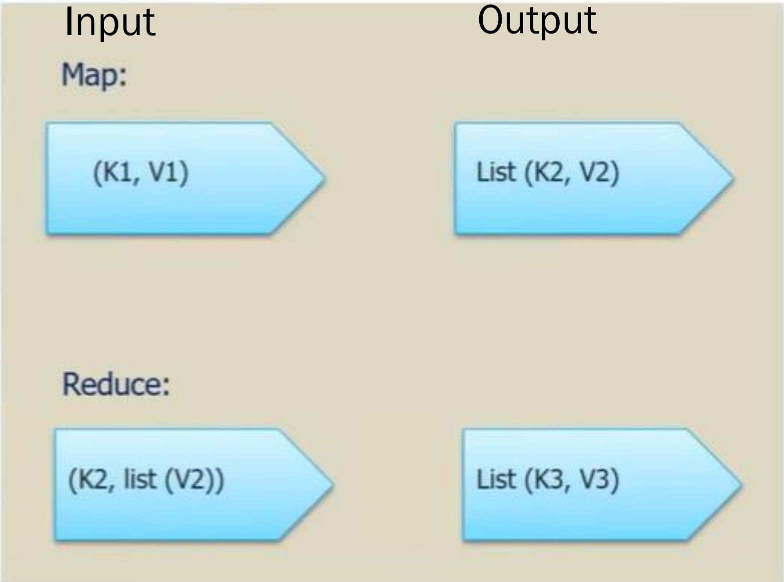



- Counting Part of respective booth was done by map function and sent to reducer.
- Combining of results was done by reducer function

# Anatomy of MapReduce:



# Exercise: MapReduce

1. Search specific keyword in a file

2. Sort data by student name

3. Arrange the data on user ID, then within user ID sort them in increasing order of page count

1. Write MapReduce program to find unit wise salary

# How to Interact with HDFS

1. hadoop namenode –format
2. hadoop secondarynamenode
3. hadoop namenode
4. hadoop datanode

start-all.sh

5. hadoop dfsadmin -report
6. hadoop mradmin -refreshNodes
7. hdfs dfsadmin -setBalancerBandwidth <bandwidth>
8. hdfs fsck /user/example/data
9. hdfs fsck /path/to/hdfs/directory -files -blocks –locations
10. hdfs fsck /path/to/hdfs/directory -files -blocks -locations -racks –replicaDetails
11. hdfs fsck /path/to/hdfs/directory -files -blocks -locations -racks -replicaDetails > fsck_report.txt

1. hadoop mradmin -refreshNodes -decommission <hostname>
2. hadoop mradmin -refreshNodes -commission <hostname>

---

3. hdfs dfs -ls /path/to/directory
4. hdfs dfs -copyToLocal /path/in/hdfs localfile
5. hdfs dfs -rm /path/to/file
6. hdfs dfs -rm -r /user/hadoop/data
7. hdfs dfs -mv /path/to/source /path/to/destination
8. hdfs dfs –put <src path> <dest path>
9. Hdfs –get <hdfs file path> <local path>

http://<TaskTracker-Hostname>:50060

http://<JobTracker-Hostname>:50030

hadoop balancer -h

# Interact with MapReduce Jobs

| Sr.No. | GENERIC_OPTION & Description |
|---|---|
| 1 | **-submit <job-file>**<br>Submits the job. |
| 2 | **-status <job-id>**<br>Prints the map and reduce completion percentage and all job counters. |
| 3 | **-counter <job-id> <group-name> <countername>**<br>Prints the counter value. |
| 4 | **-kill <job-id>**<br>Kills the job. |
| 5 | **-events <job-id> <fromevent-#> <#-of-events>**<br>Prints the events' details received by jobtracker for the given range. |
| 6 | **-history [all] <jobOutputDir> - history < jobOutputDir>**<br>Prints job details, failed and killed tip details. More details about the job such as successful tasks and task attempts made for each task can be viewed by specifying the [all] option. |
| 7 | **-list[all]**<br>Displays all jobs. -list displays only jobs which are yet to complete. |
| 8 | **-kill-task <task-id>**<br>Kills the task. Killed tasks are NOT counted against failed attempts. |
| 9 | **-fail-task <task-id>**<br>Fails the task. Failed tasks are counted against failed attempts. |
| 10 | **-set-priority <job-id> <priority>**<br>Changes the priority of the job. Allowed priority values are VERY_HIGH, HIGH, NORMAL, LOW, VERY_LOW |

# Yet Another Resource Negotiator (YARN)

- YARN stands for "Yet Another Resource Negotiator". It was introduced in Hadoop 2.0 **to remove the bottleneck on Job Tracker which was present in Hadoop 1.0.**

- YARN was described as a "*Redesigned Resource Manager*" at the time of its launching

- YARN architecture basically **separates resource management layer from the processing layer.**

- YARN also allows different data processing engines like graph processing, interactive processing, stream processing as well as batch processing.

- it can **dynamically allocate various resources** and schedule the application processing

# YARN Features

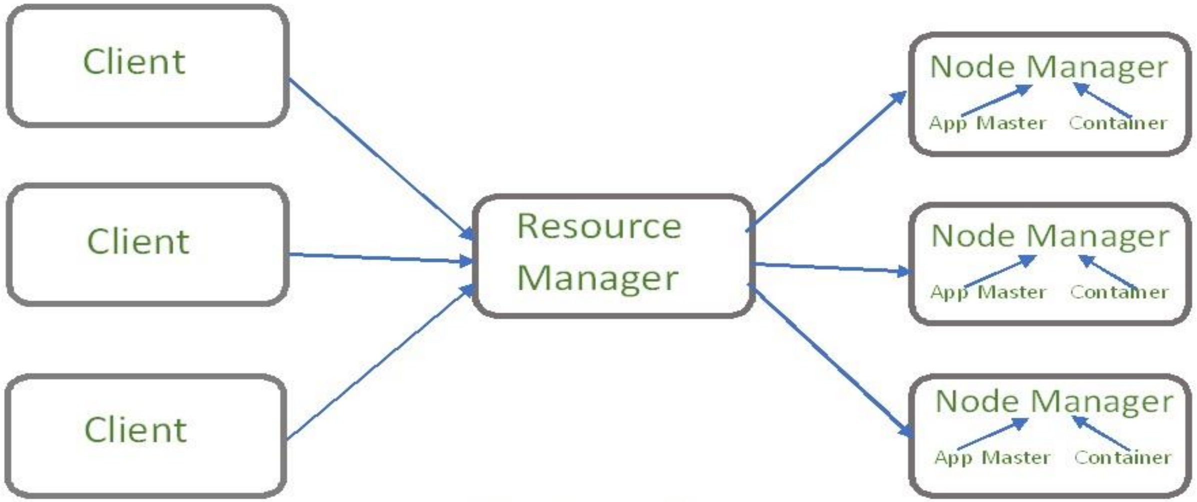
- **Scalability:** The scheduler in Resource manager of YARN architecture allows Hadoop to extend and manage thousands of nodes and clusters.

- **Compatibility:** YARN supports the existing map-reduce applications without disruptions thus making it compatible with Hadoop 1.0 as well.

- **Cluster Utilization:** Since YARN supports Dynamic utilization of cluster in Hadoop, which enables optimized Cluster Utilization.

- **Multi-tenancy:** It allows multiple engine access thus giving organizations a benefit of multi-tenancy.

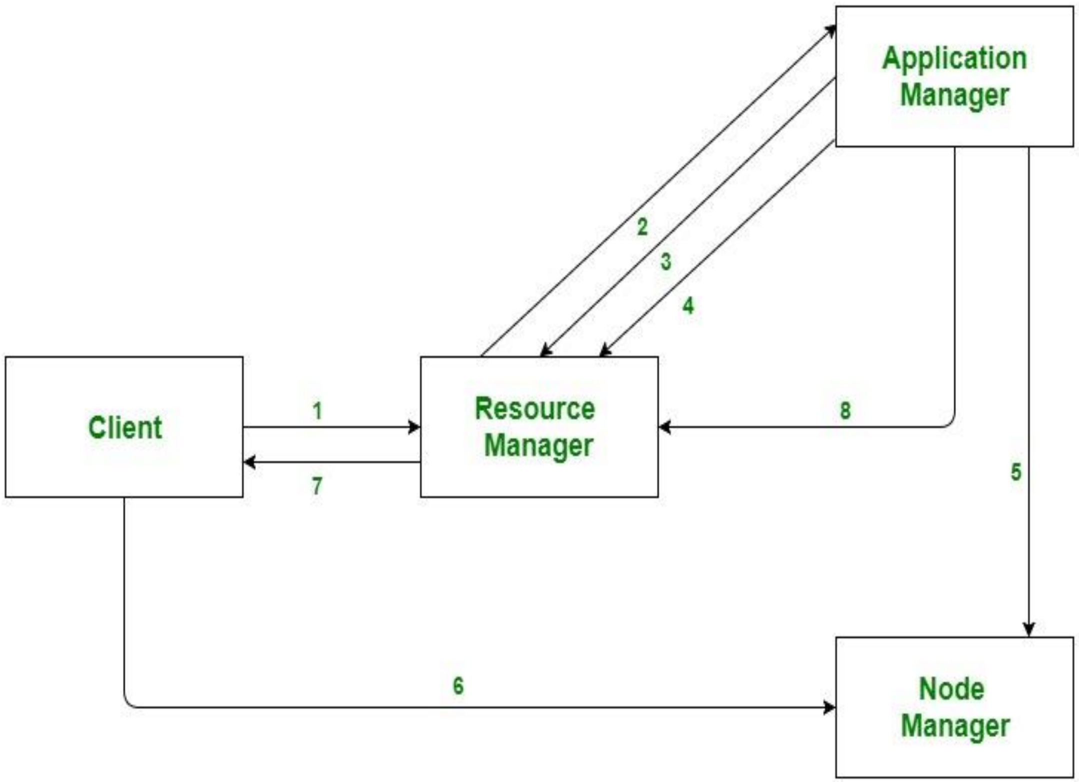
# The main components of YARN architecture




Hadoop Yarn architecture

- Client
- Resource Manager
  1. **Scheduler**
  2. **Application manager**
- Node Manager
- Application Master
- Container

# Application workflow in Hadoop YARN

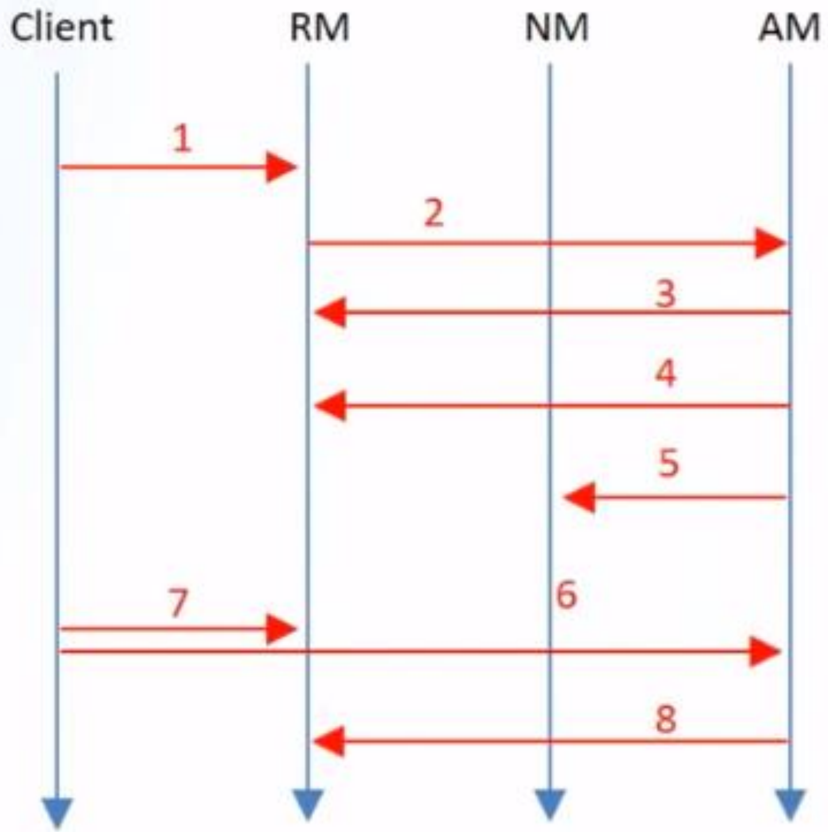



1. Client submits an application
2. The Resource Manager allocates a container to start the Application Manager
3. The Application Manager registers itself with the Resource Manager
4. The Application Manager negotiates containers from the Resource Manager
5. The Application Manager notifies the Node Manager to launch containers
6. Application code is executed in the container
7. Client contacts Resource Manager/Application Manager to monitor application's status
8. Once the processing is complete, the Application Manager un-registers with the Resource Manager

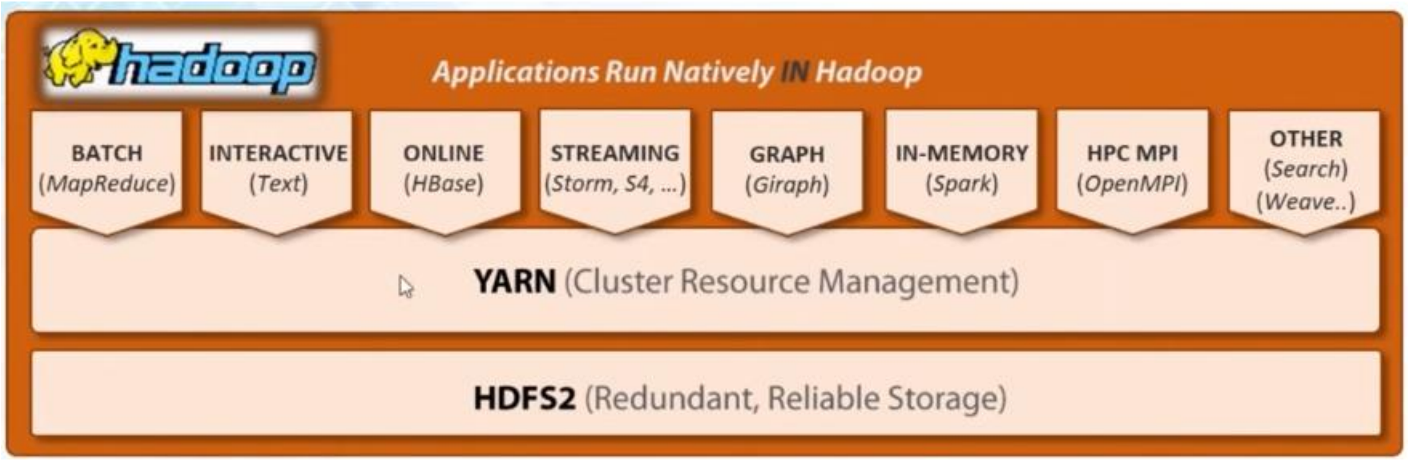
# Application : Workflow

→ Execution Sequence :

1. Client submits an application
2. RM allocates a container to start AM
3. AM registers with RM
4. AM asks containers from RM
5. AM notifies NM to launch containers
6. Application code is executed in container
7. Client contacts RM/AM to monitor application's status
8. AM unregisters with RM

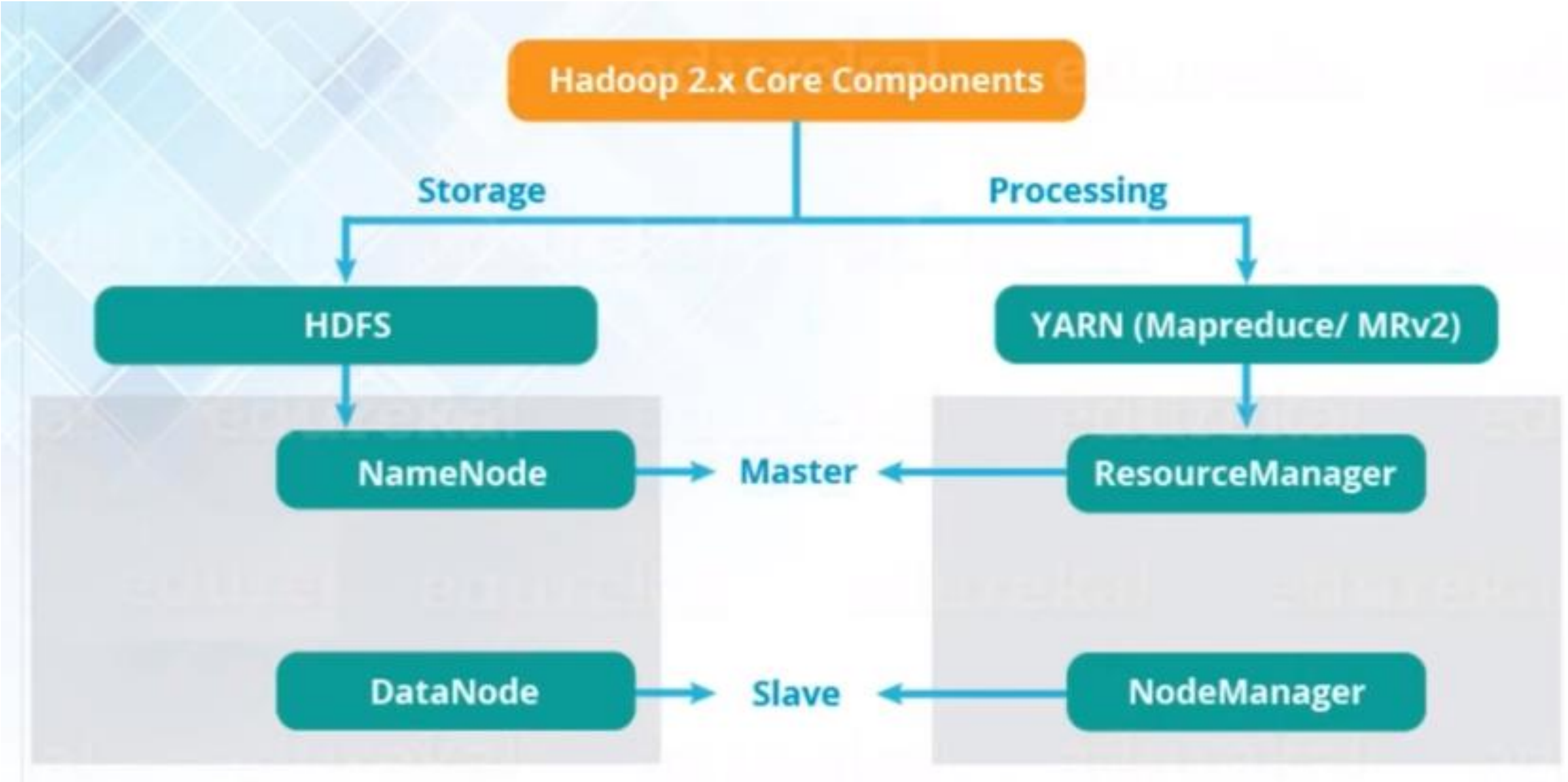


# YARN : MapReduce Beyond




- Using YARN lot of frameworks are able to utilize and connect to HDFS
- YARN – Open gate for frameworks, other search engines and even other big data applications also
- Application of "HDFS " increased because of YARN as resource provider

# Hadoop Daemons:



# Hadoop 2.X MapReduce Yarn Components:

→ Client

» Submits a MapReduce Job

→ Resource Manager

» Cluster Level resource manager
» Long Life, High Quality Hardware

→ Node Manager

» One per Data Node
» Monitors resources on Data Node

→ Job History Server

» Maintains information about submitted MapReduce jobs after their ApplicationMaster terminates

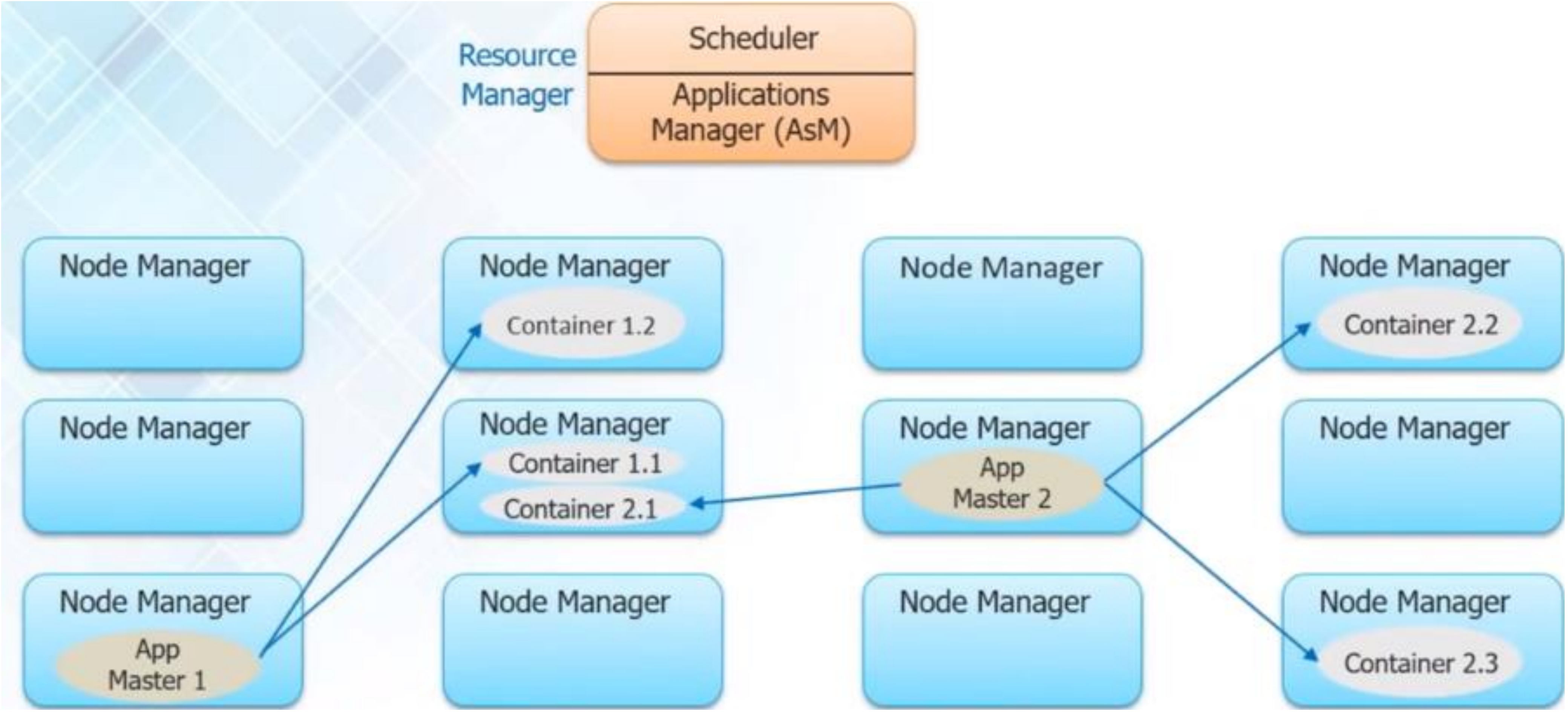
→ ApplicationMaster

» One per application
» Short life
» Coordinates and Manages MapReduce Jobs
» Negotiates with Resource Manager to schedule tasks
» The tasks are started by NodeManager(s)

→ Container

» Created by NM when requested
» Allocates certain amount of resources (memory, CPU etc.) on a slave node

# YARN: Workflow




THANK
YOU.