Hadoop

Setup and Configuration

By

CHITRAKANT BANCHHOR

IT,MITCOE, Pune

cobanchhor@gmail.com

Hadoop : First step

awareness about Distributed computing and system

Cluster Computing

What is Cluster Computing ?

- A collection of computers configured in such a way that they can be used to solve a problem by means of parallel processing.


Sun

The First Beowulf

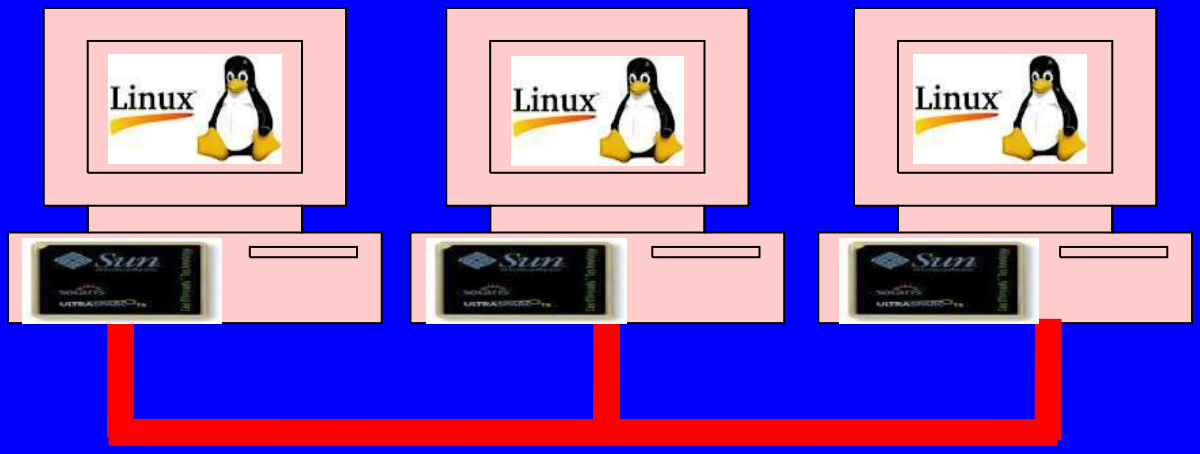
- 16 x 486DX4, 100MHz processors
- 16MB of RAM each, 256MB in total
- Ethernet (2 x 10Mbps)

Current Beowulf

- Faster processors, faster interconnect, but the idea remains the same
- Top cluster: 1.433 TFLOPS peak

Homogeneous Cluster

- The cluster in which every single node is exactly same, from the motherboard and the memory, to the disk drives and the NIC .



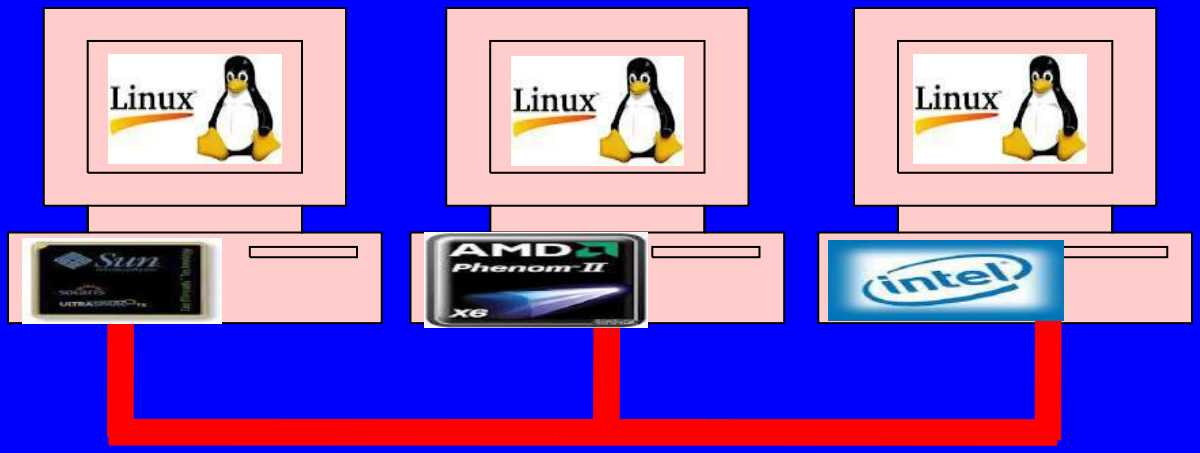
- Advantages:

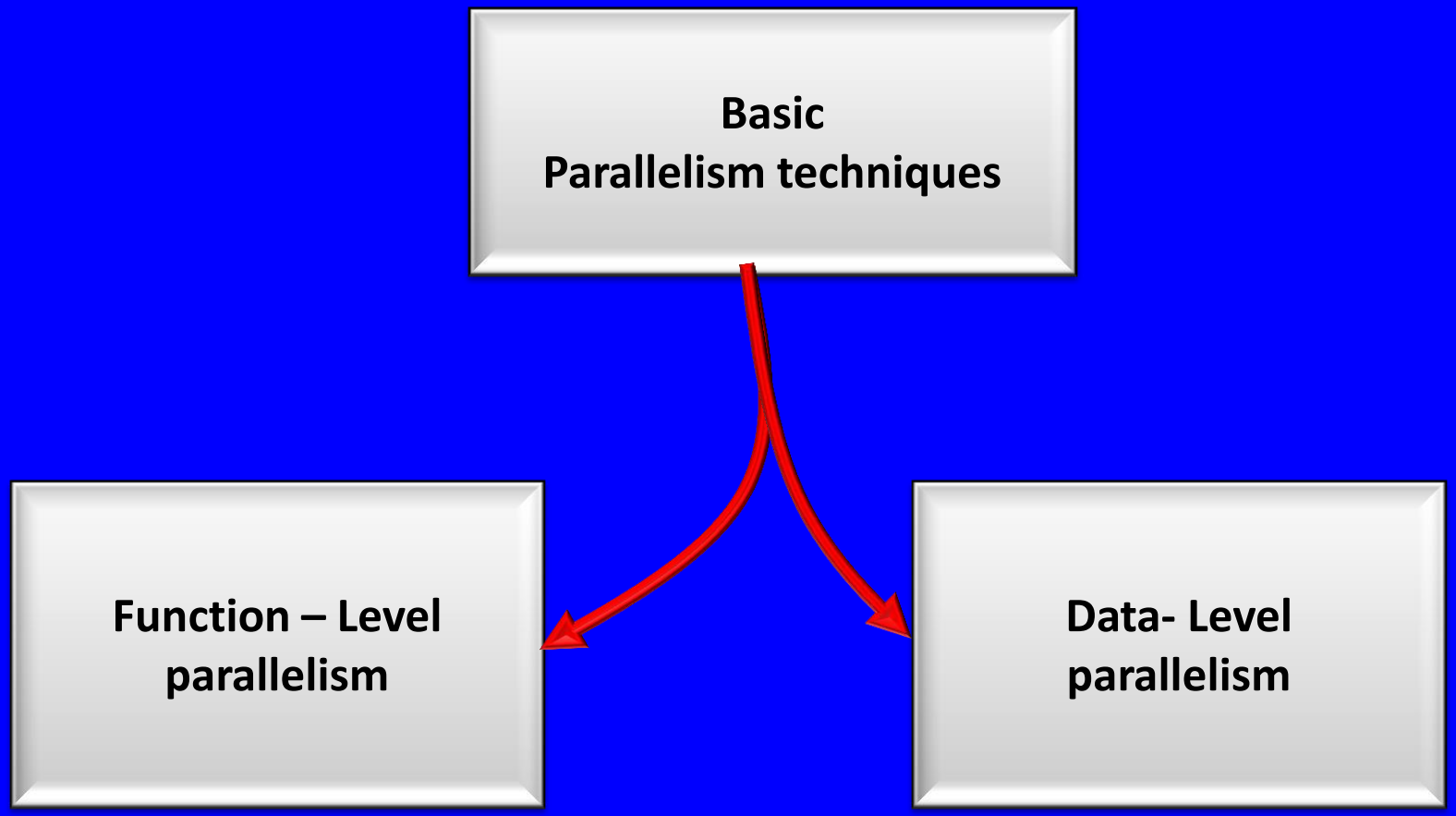
1. The use of homogeneous system will ensure that software will operate same on each node.

2. The nodes within the system are interchangeable

Heterogeneous Cluster

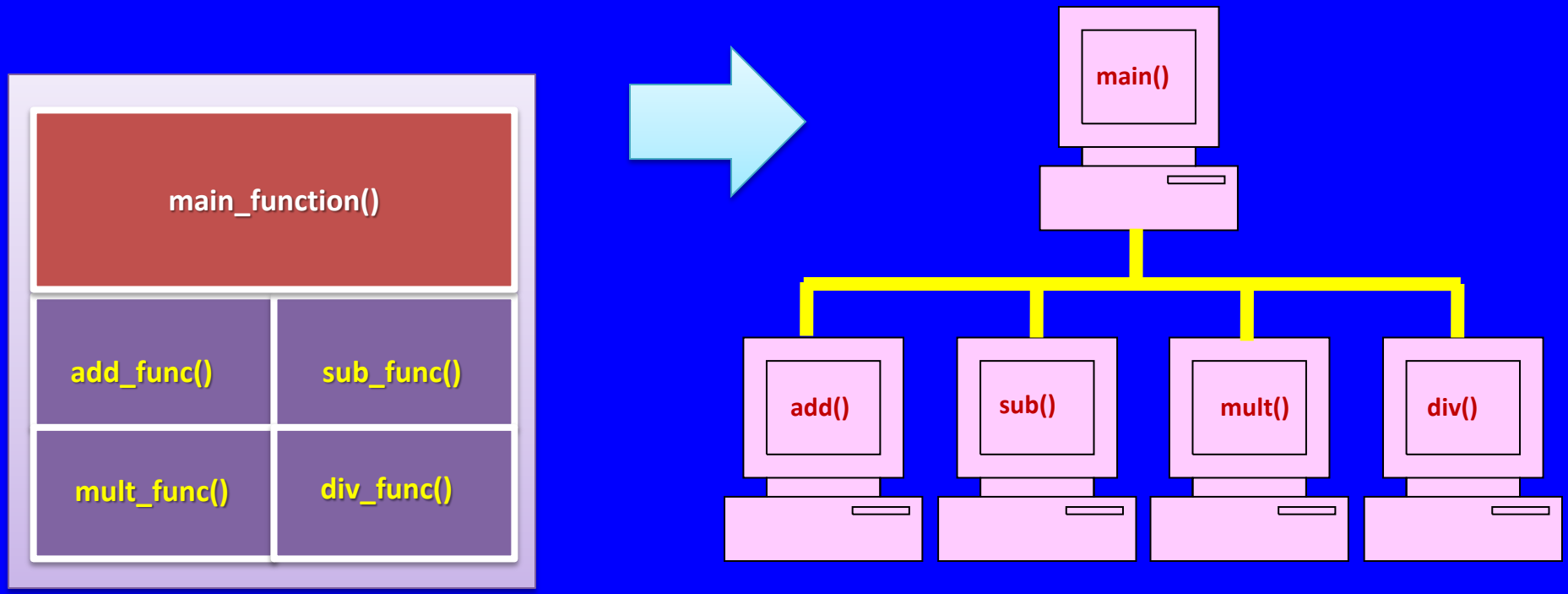
- Made from different kinds of computers, SPARC, DEC ALPHA




Basic
Parallelism techniques
Function – Level
parallelism
Data- Level
parallelism

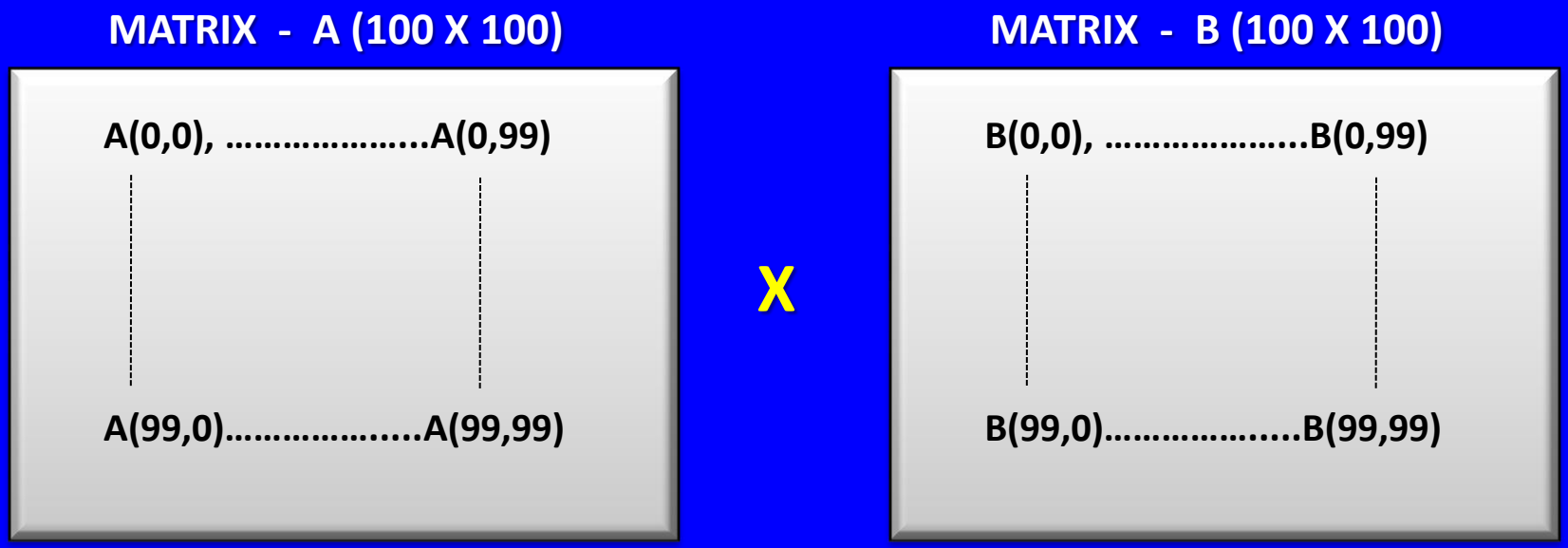
Functional parallelism

- An application is parallelized along its functions



Data Parallelism (SPMD)

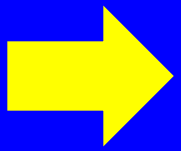
- All the tasks are same, but each one only solves the small part of the data.


MATRIX - A (100 X 100)
A(0,0), ……..A(0,99)
A(99,0)……..A(99,99)
X
MATRIX - B (100 X 100)
B(0,0), ……..B(0,99)
B(99,0)……..B(99,99)

MATRIX - A (100 X 100)

A(0,0), ……..A(0,99)

A(99,0)……..A(99,99)



A(0,0) .. A(0,24)

A(24,0) .. A(24,24)

A(25,25) .. A(25,49)

A(49,25) .. A(49,49)

A(50,50) .. A(50,74)

A(74,50) .. A(74,74)

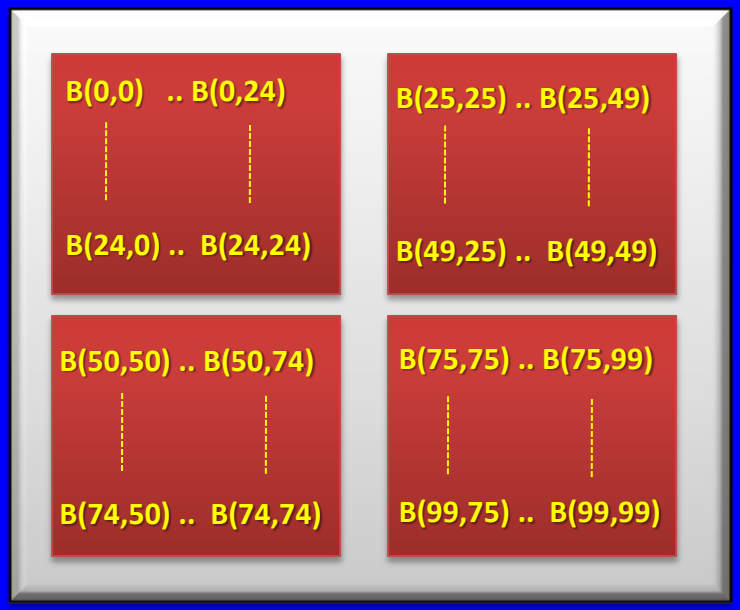
A(75,75) .. A(75,99)

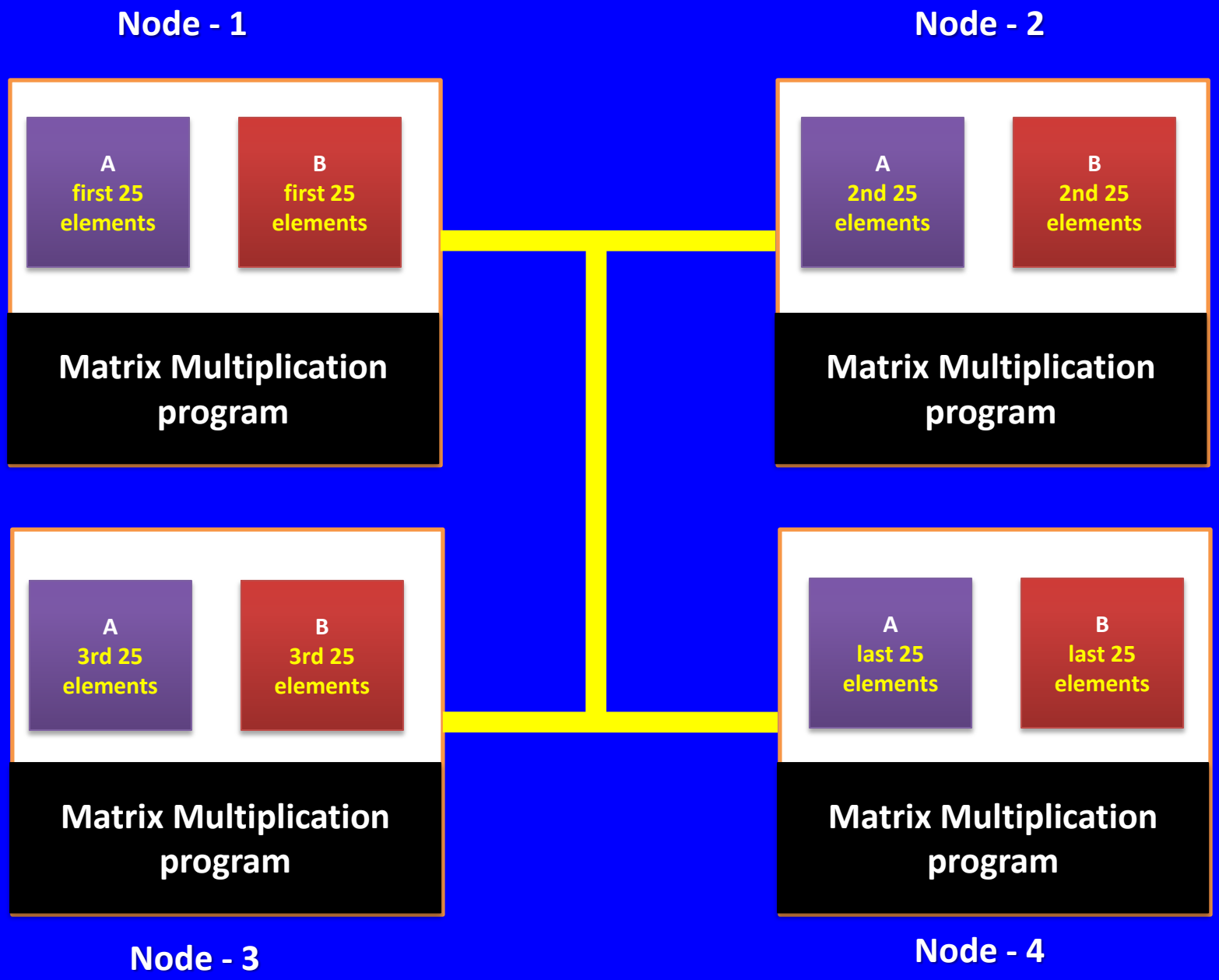
A(99,75) .. A(99,99)

MATRIX - B (100 X 100)

B(0,0), ……..B(0,99)

B(99,0)……..B(99,99)




Node - 1
A
first 25
elements
B
first 25
elements
Matrix Multiplication
program
Node - 2
A
2nd 25
elements
B
2nd 25
elements
Matrix Multiplication
program
A
3rd 25
elements
B
3rd 25
elements
Matrix Multiplication
program
Node - 3
A
last 25
elements
B
last 25
elements
Matrix Multiplication
program
Node - 4

Common parallel programming Paradigm

- Master – slave model

master.c

1. Get input data typically from file
2. Divide the data in separate chunks
3. Distribute each chunk into different slave programs in different nodes.
4. Wait for intermediate results from various slaves.
5. Combine these intermediate results and produce the final result

slave.c

1. Get data upon which algorithm operates from the master.
2. Performs defined task on the set of data.
3. Return intermediate result to master.

Step - I

Hadoop : Second step

Background HDFS and MapReduce

Hadoop
An Introduction

References

1. Chuck Lam, “Hadoop in Action”, December, 2010, Manning publication
2. Alex Holmes, “Hadoop in Practice”, October”, 2012, Manning publication
3. Tom White, “Hadoop: The Definitive Guide”, May 2009, O'Reilly Media
4. Eric Sammer, “Hadoop Operations”, September 2012, O'Reilly Media

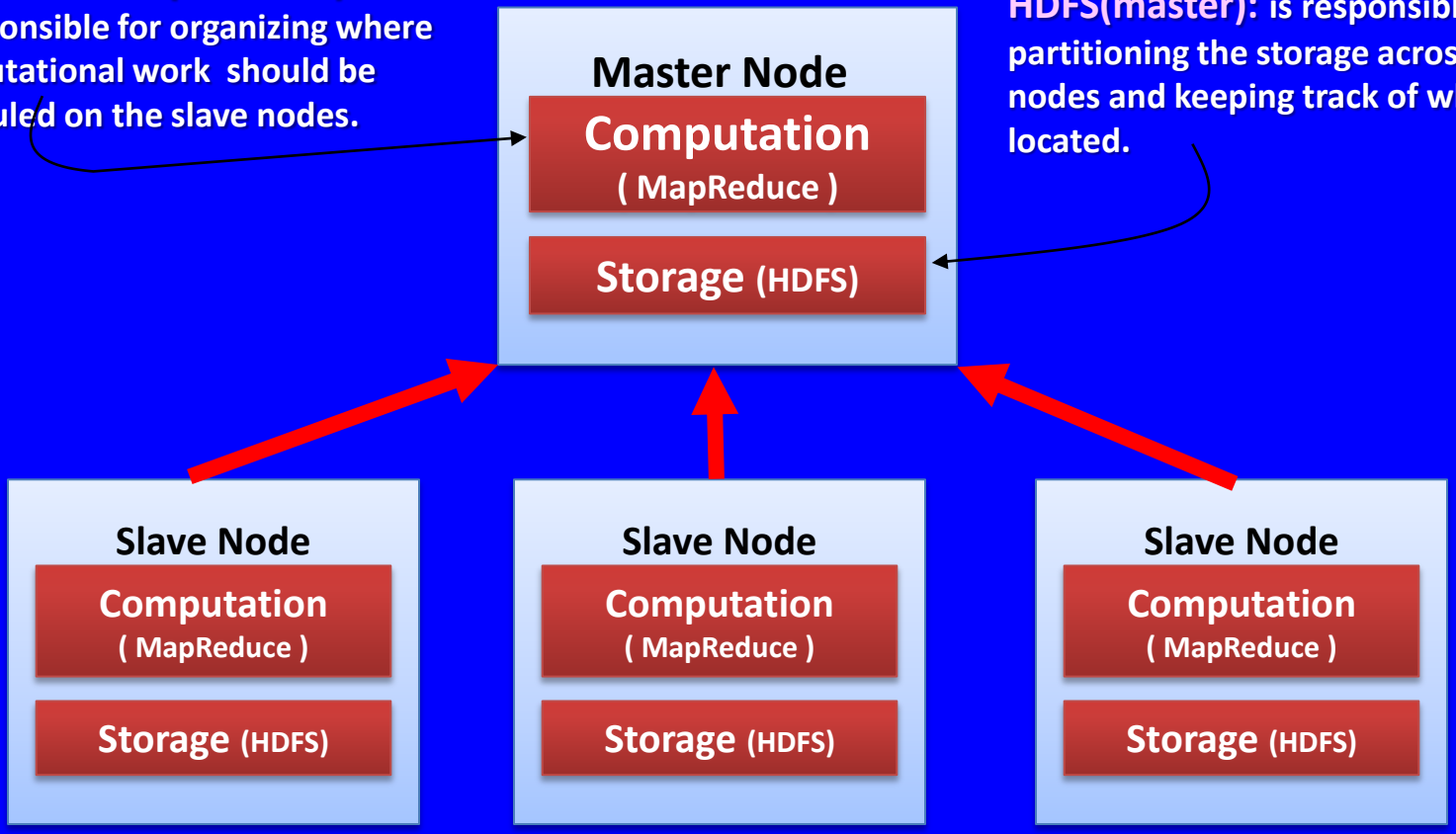
What is Hadoop?

- **Software Library:** Hadoop is a software library for distributed computing.
- **Distributed Storage:** Hadoop platform provides distributed storage.
- **Computational Capabilities:** Hadoop platform provides distributed computational capabilities.

High-level Hadoop architecture

MapReduce(master):
is responsible for organizing where computational work should be scheduled on the slave nodes.

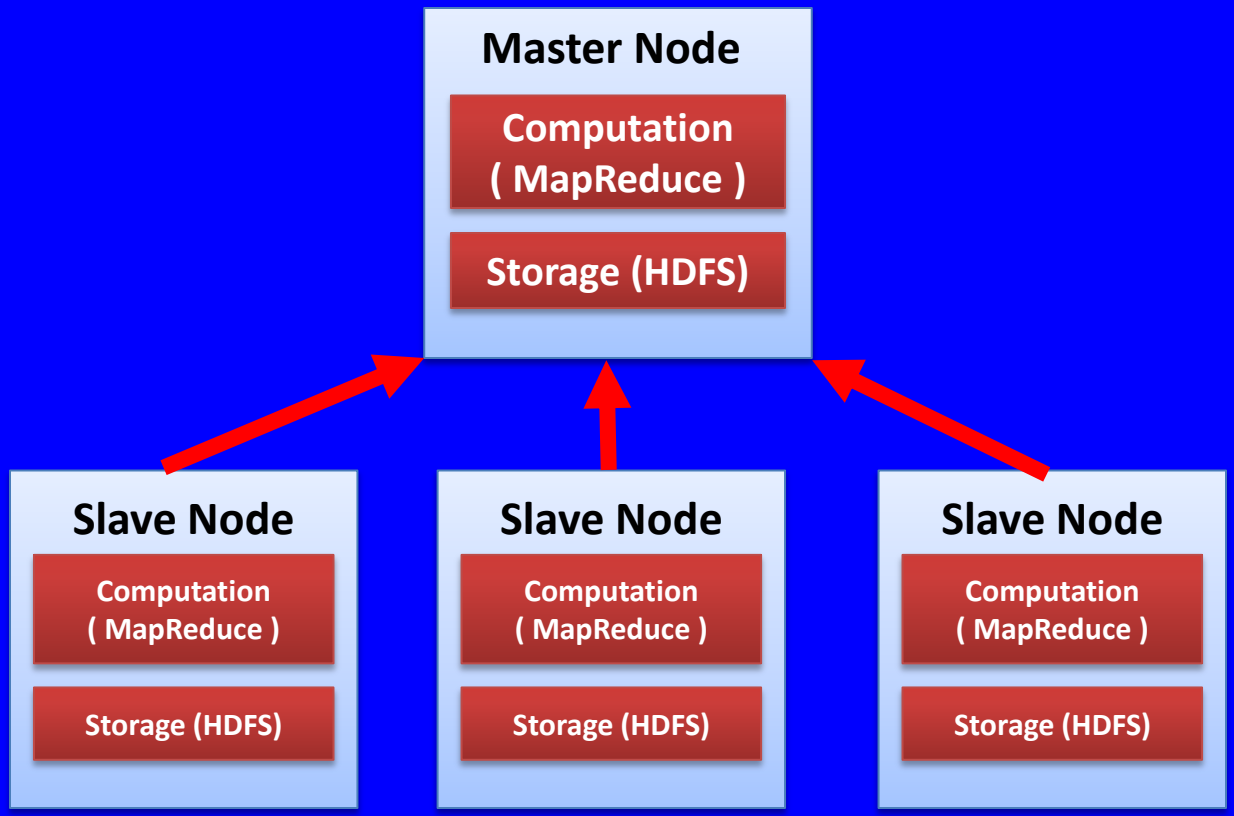
HDFS(master): is responsible for partitioning the storage across the slave nodes and keeping track of where data is located.



❖ Hadoop is a distributed master-slave architecture.

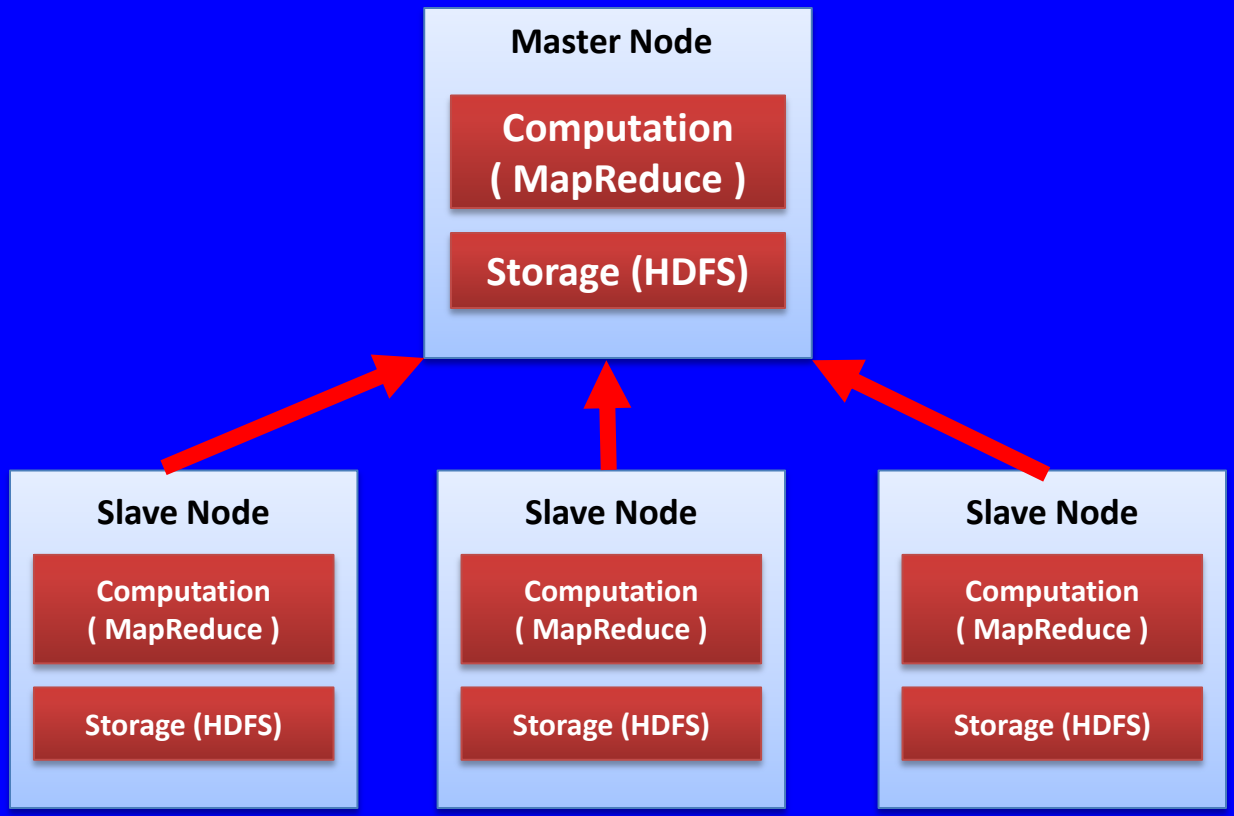
❖ Hadoop consists :

- ➢ HDFS for storage
- ➢ Map-Reduce for computational capabilities.



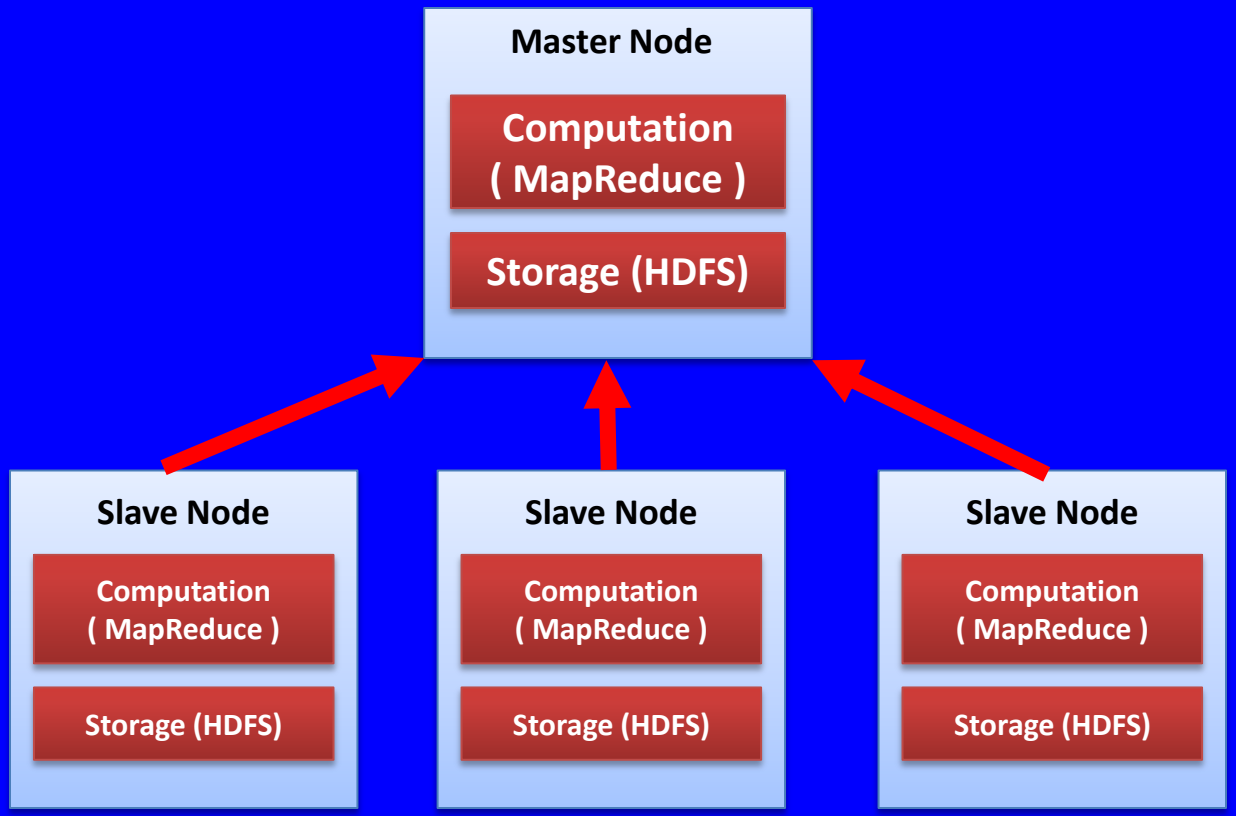
❖ HDFS(master):

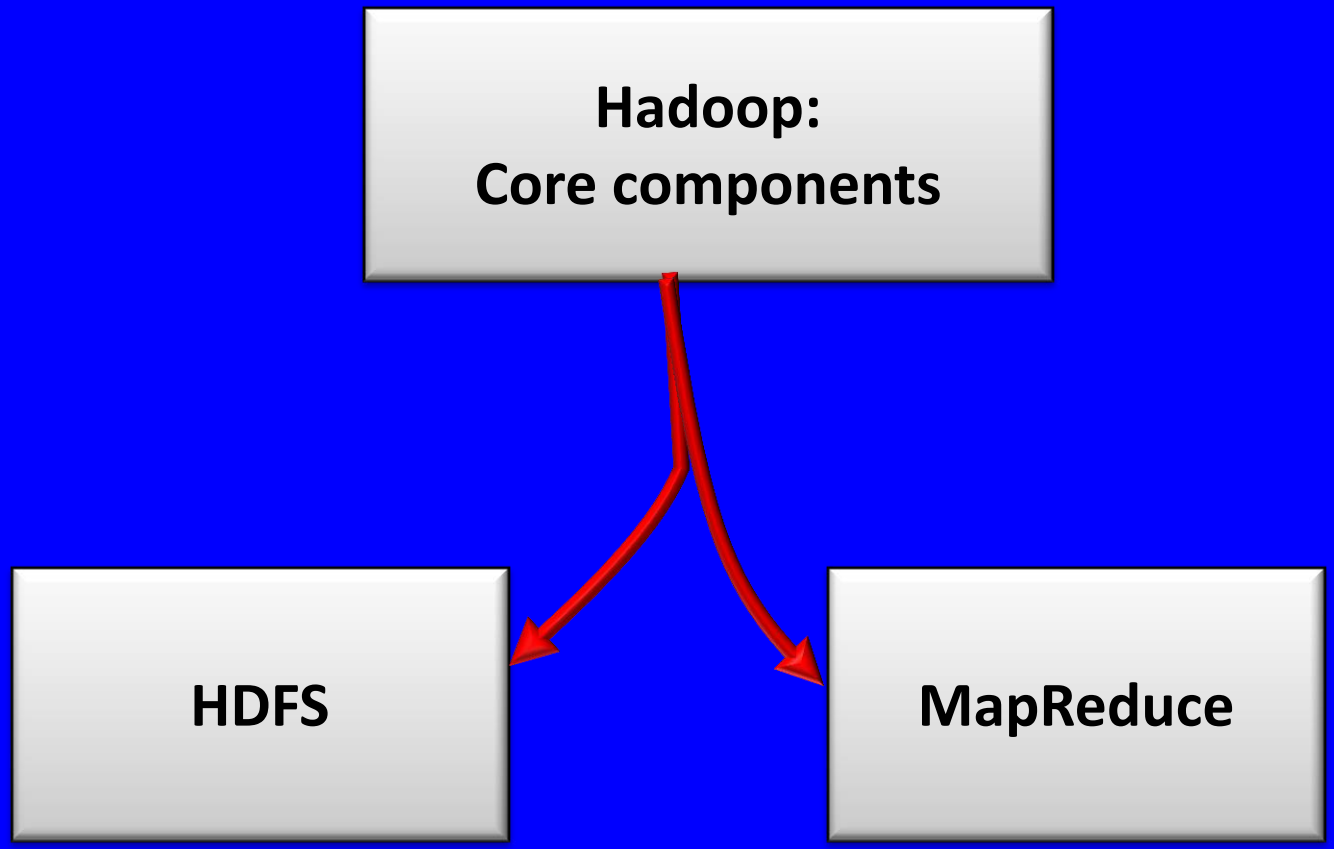
- Partitions the overall storage across the slave nodes.
- Keeps track of locations of data.



❖ MapReduce(master):

- **Organizing slave nodes to schedule computational tasks.**




Hadoop:
Core components
HDFS
MapReduce

Hadoop Distributed File System
(HDFS)

HDFS

- **HDFS:** Hadoop's implementation of distributed file system.
- **File system facilities:** hdfs provides all facilities of a file system

HDFS Characteristics

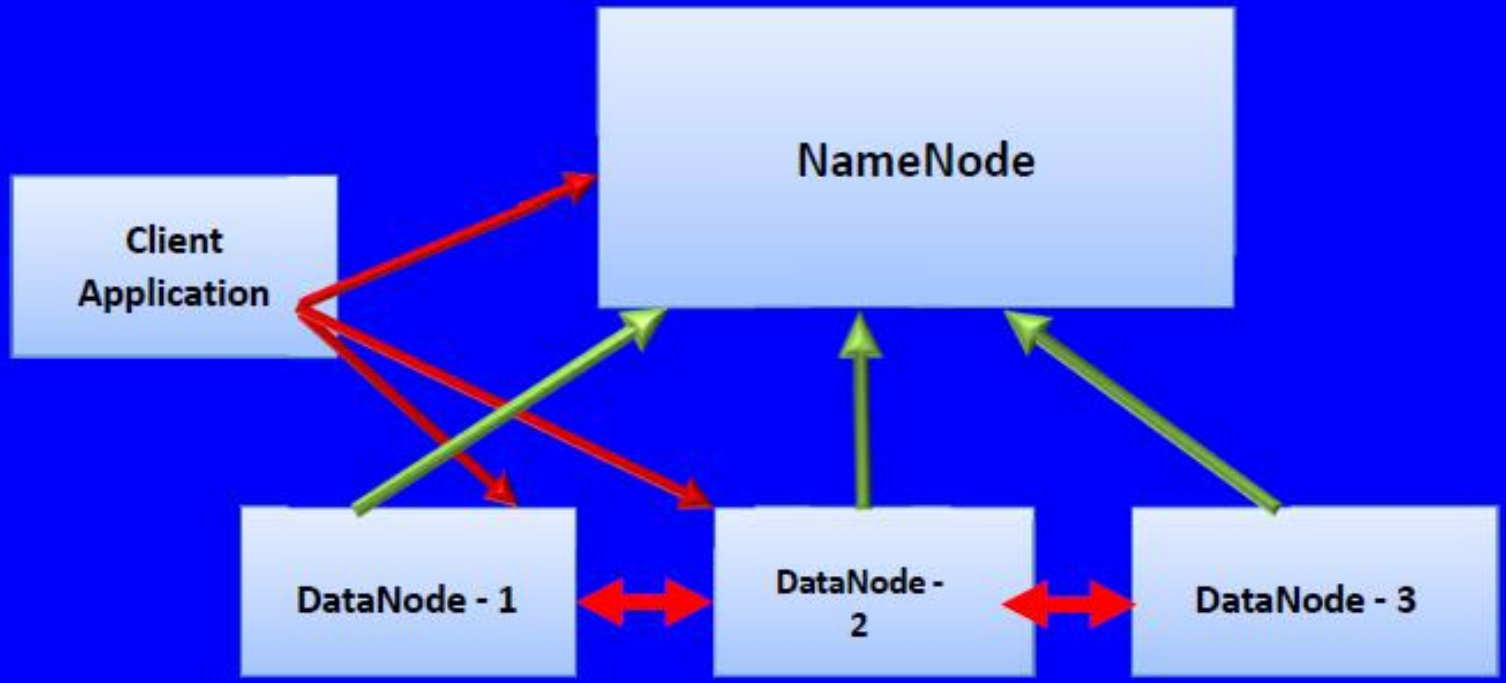
1. large files: works best when reading and writing large files (gigabytes and larger).

2. Scalability: achieved in part due to data replication.

3. High availability: achieved in part due to fault tolerance.

HDFS architecture

- **Consider an HDFS client communicating with the master NameNode and slave DataNodes.**

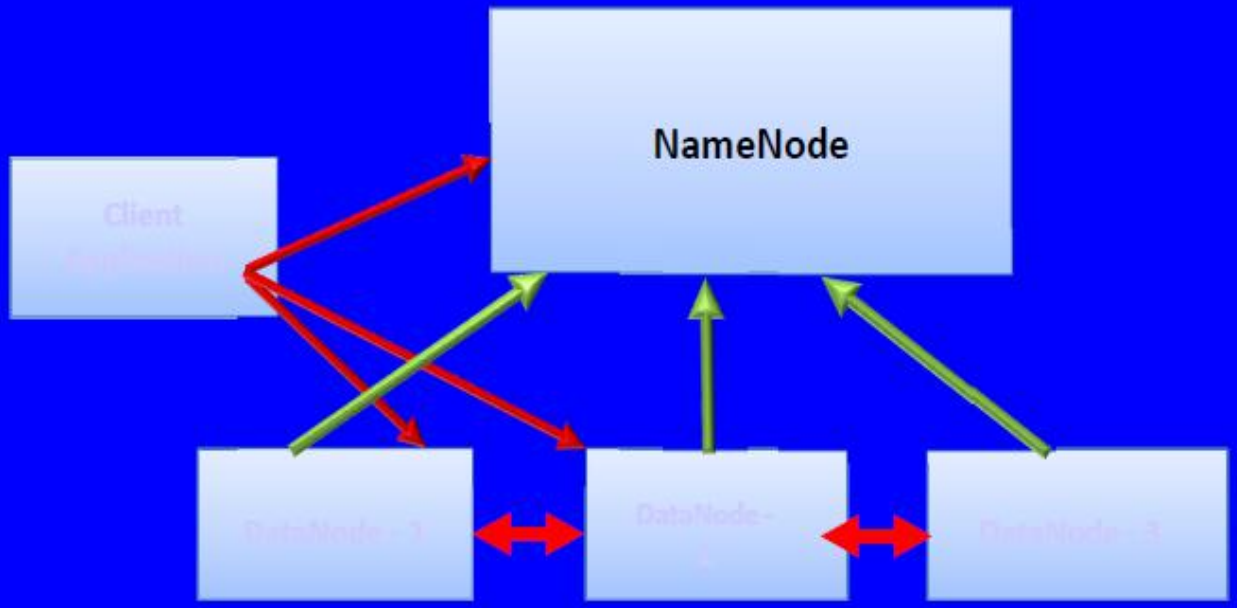


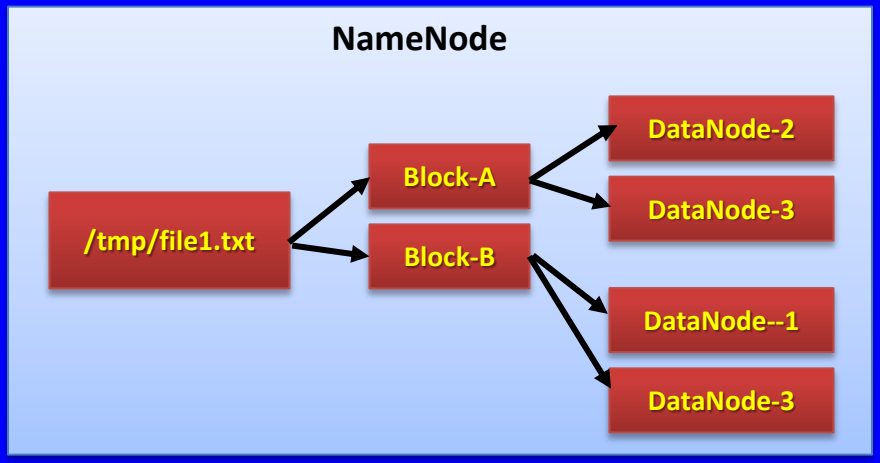
- **Master-Slave Architecture**

HDFS NameNode

- HDFS cluster consists of a single Namenode.
- It is a master server.

❖ Single point of failure : if namenode goes down - whole filesystem is offline



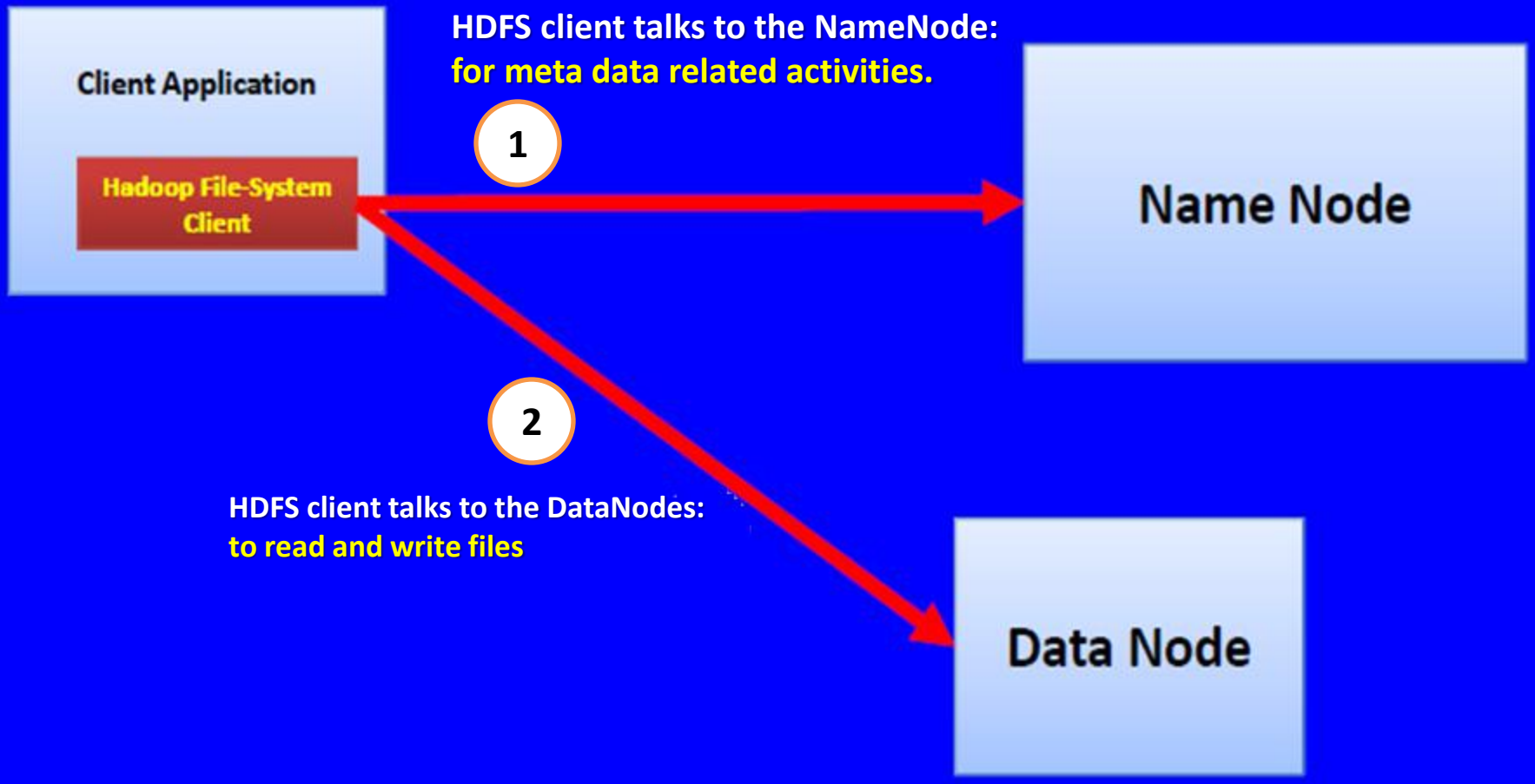


❖ NameNode contains: (The meta data about the file system)

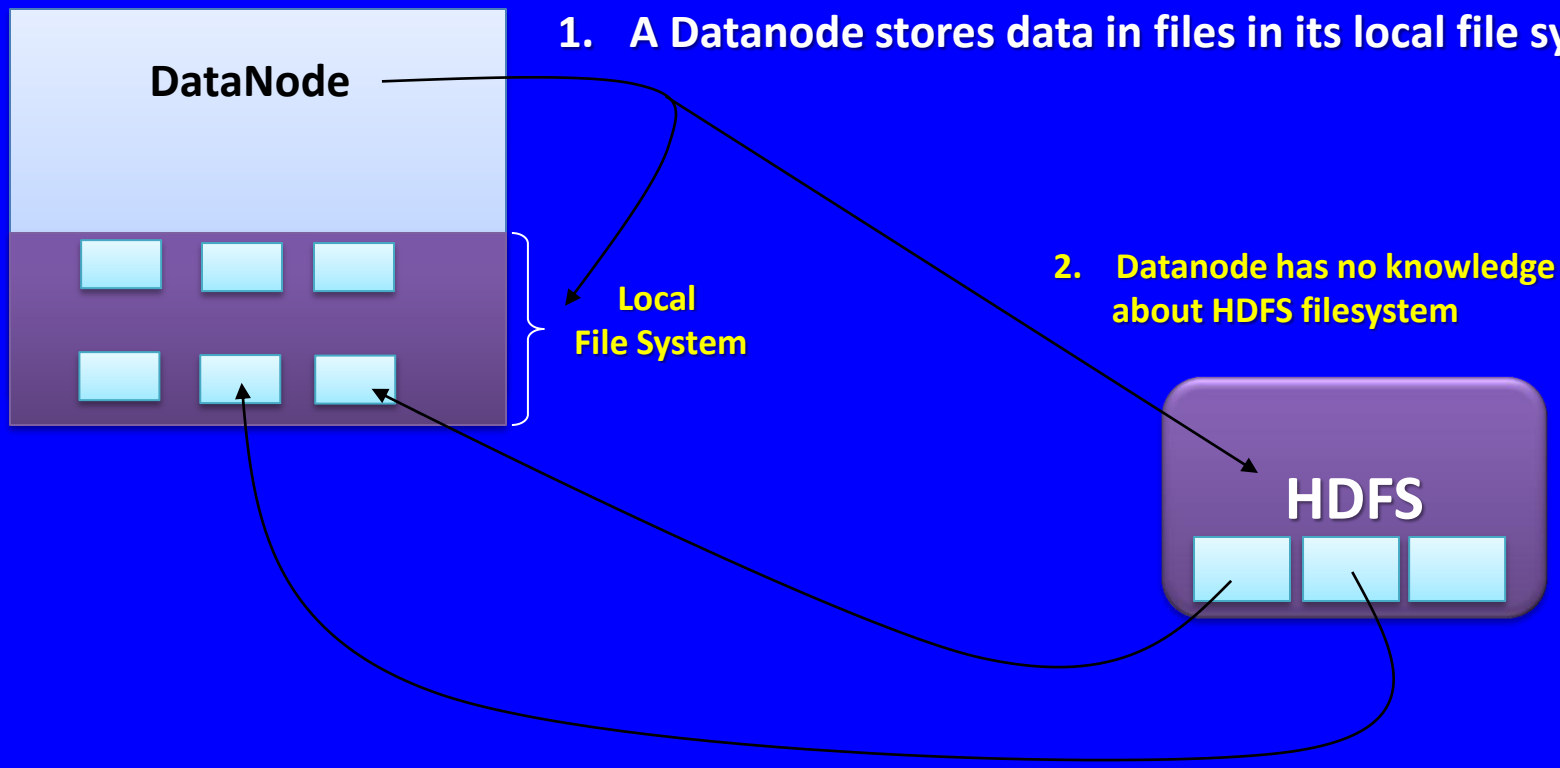
1. Information regarding a block's location.
2. The information of entire directory structure and files

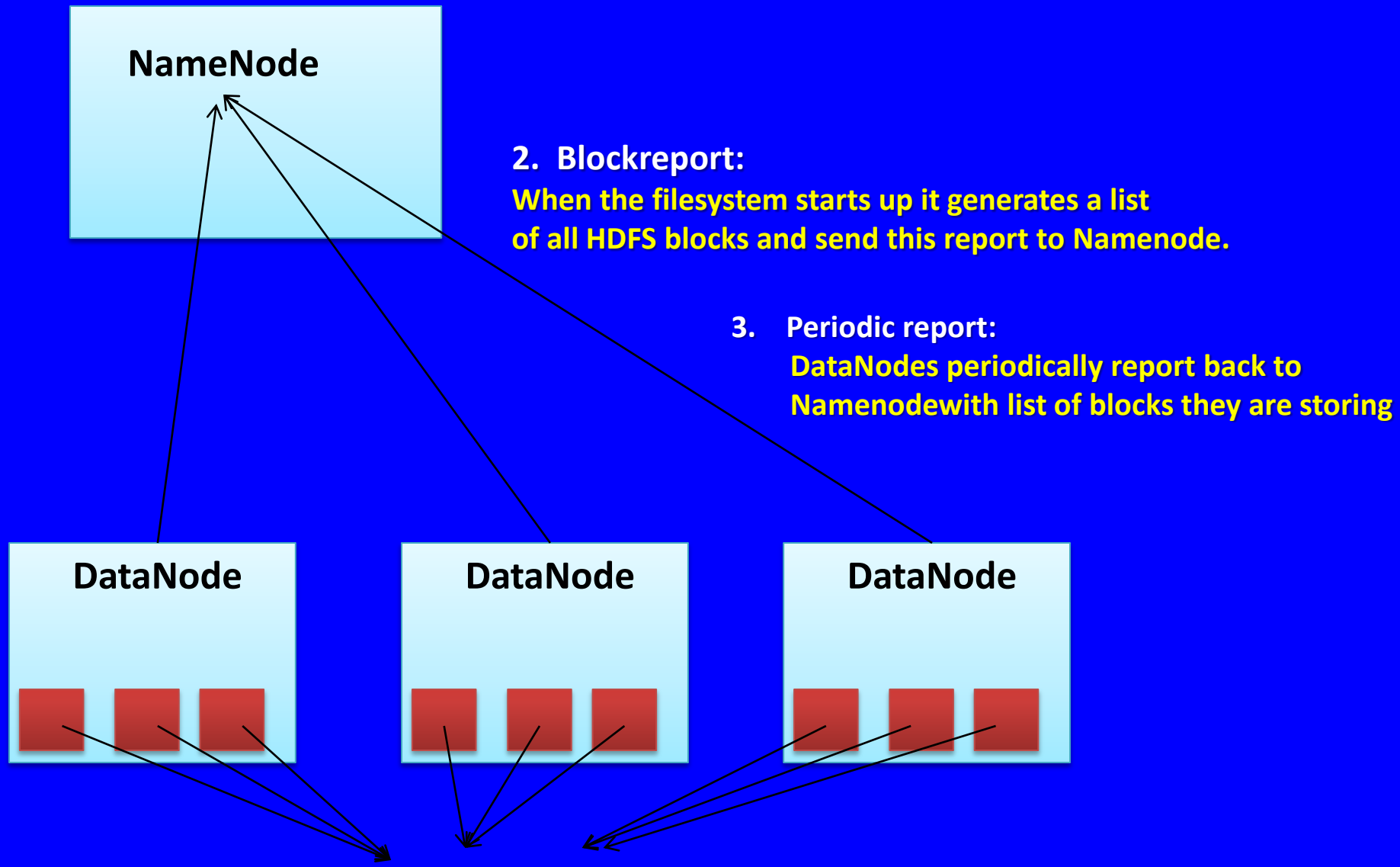
❖Meta data about the file system: which data nodes manage blocks for each file

❖HDFS Client

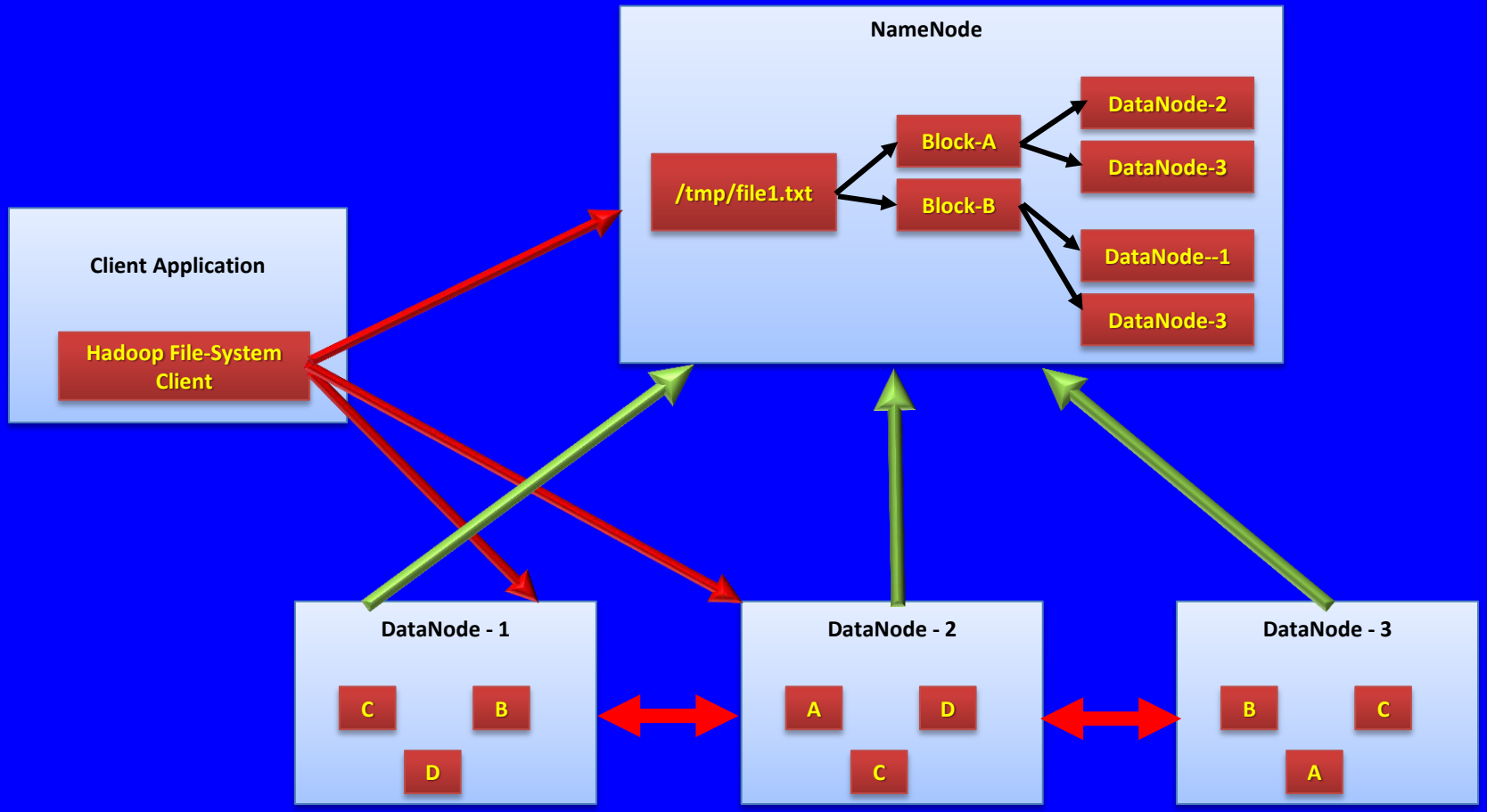

Client Application
Hadoop File-System Client
HDFS client talks to the NameNode:
for meta data related activities.
1
Name Node
2
HDFS client talks to the DataNodes:
to read and write files
Data Node

HDFS DataNodes




NameNode
2. Blockreport:
When the filesystem starts up it generates a list
of all HDFS blocks and send this report to Namenode.
3. Periodic report:
DataNodes periodically report back to
Namenodewith list of blocks they are storing
DataNode
DataNode
DataNode
1. Actual blocks of data:
Datanodes store actual blocks of data

- Logical representation of components: (NameNode, DataNode, ClientApp)
  - Figure shows a logical representation of the components in HDFS:



Map-Reduce

MapReduce Algorithm

Overview

MapReduce is a Distributed Data Processing Algorithm

MapReduce Algorithm

- **Useful to process huge amount of data in parallel**
- **Reliable and efficient way in cluster environments**

MapReduce: Internals

Divide & Conquer

- MapReduce technique for processing of large amount of data

- ❖ Divides input task into smaller and manageable sub-tasks
- ❖ Subtasks execute independently in-parallel

MapReduce Algorithm Steps

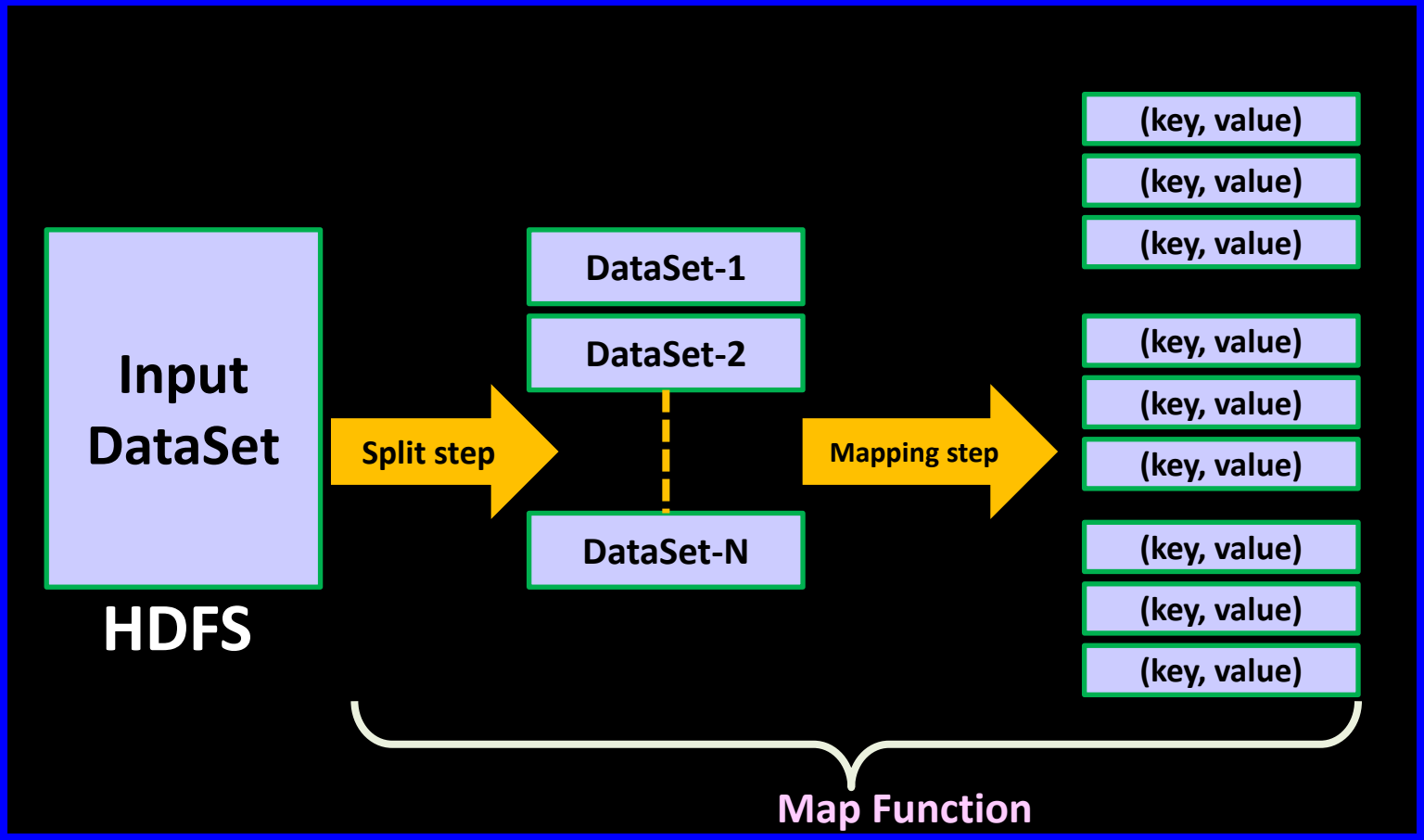
Main steps

1. Map Function
2. Shuffle Function
3. Reduce Function

Map Function

1. Accepts input tasks (DataSets)
2. Divides input tasks into smaller sub-tasks
3. Perform required computation on each sub – task in parallel
4. Perform subtasks
   - Splitting: Gets input DataSet from Source and divide into smaller Sub-DataSets
   - Mapping: Gets smaller Sub-DataSets and do required computation on each Sub-DataSet

Map function output → A set of key and value pairs as <Key, Value>



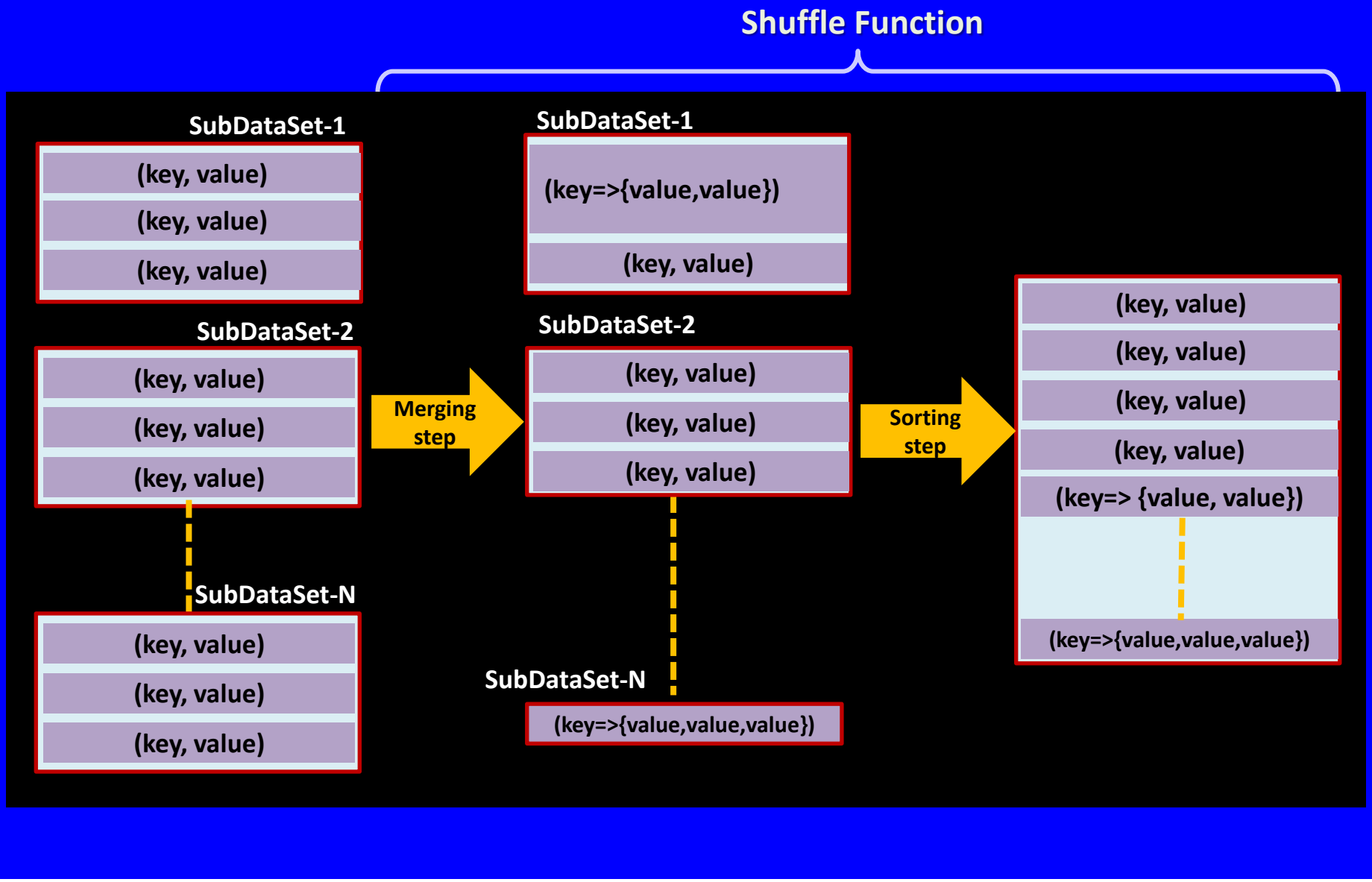
MapReduce First Step Output: Output of Map Function = List of <key, Value> Pairs

Shuffle Function (Combine Function)

Steps

1. Merging
2. Sorting

- Takes a list of outputs coming from “Map Function”
- Perform two sub-steps (merging, sorting) on each and every key-value pair



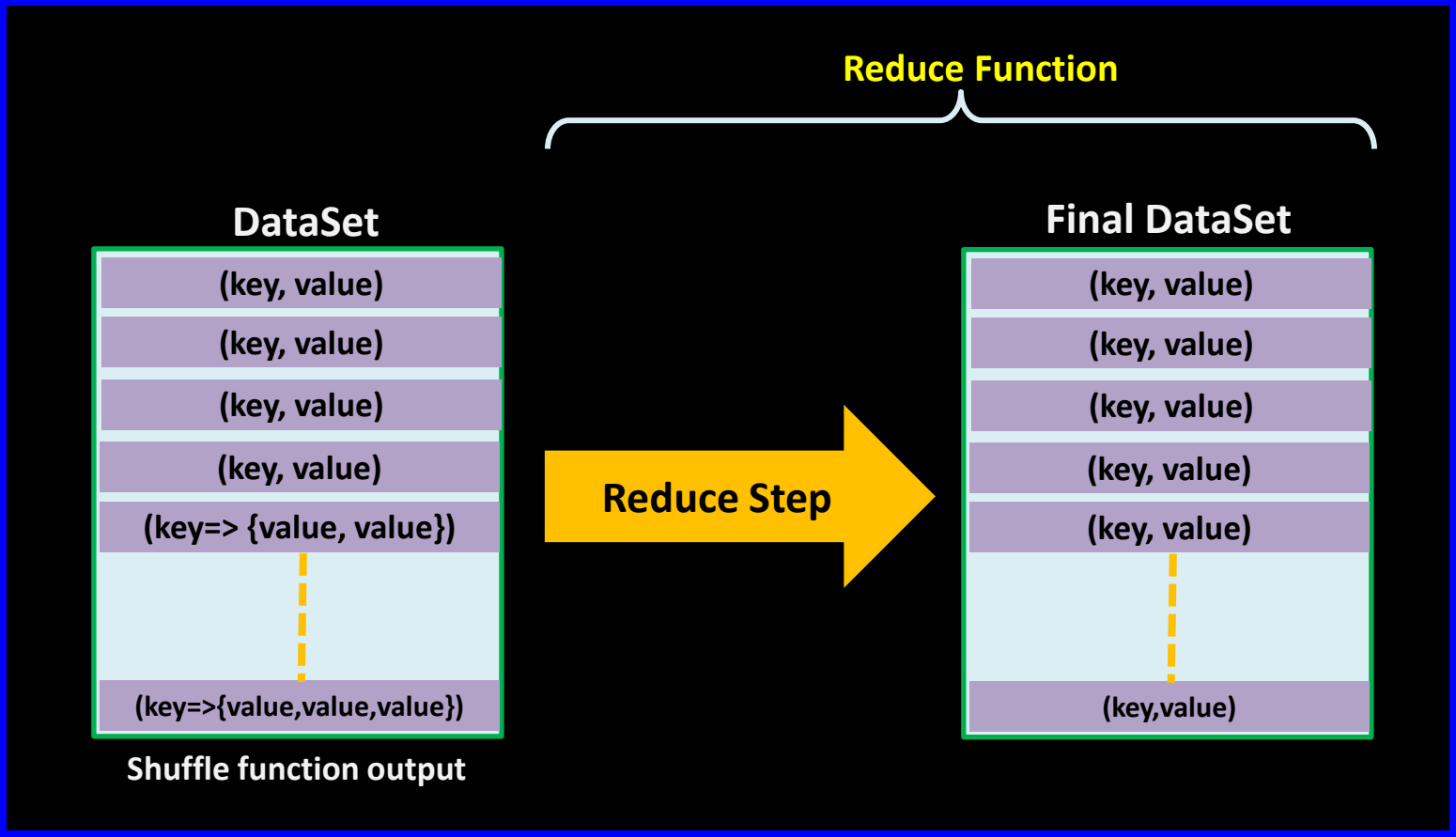
MapReduce Second Step Output: shuffle function output = List of<key, List<value>> pairs

Reduce Function

Reduce

- ❑ **Final step in MapReduce Algorithm**
- ❑ **Performs Only one step: Reduce step**

- **Takes a list of <Key, List<Value>> sorted pairs from shuffle function**
- **Performs reduce operation**



MapReduce final Step Output: **Reduce function output = List of<key, value> pairs**

MapReduce with word count

Problem statement

- **Counting:** **the number of occurrences of each word available in a Data-Set.**

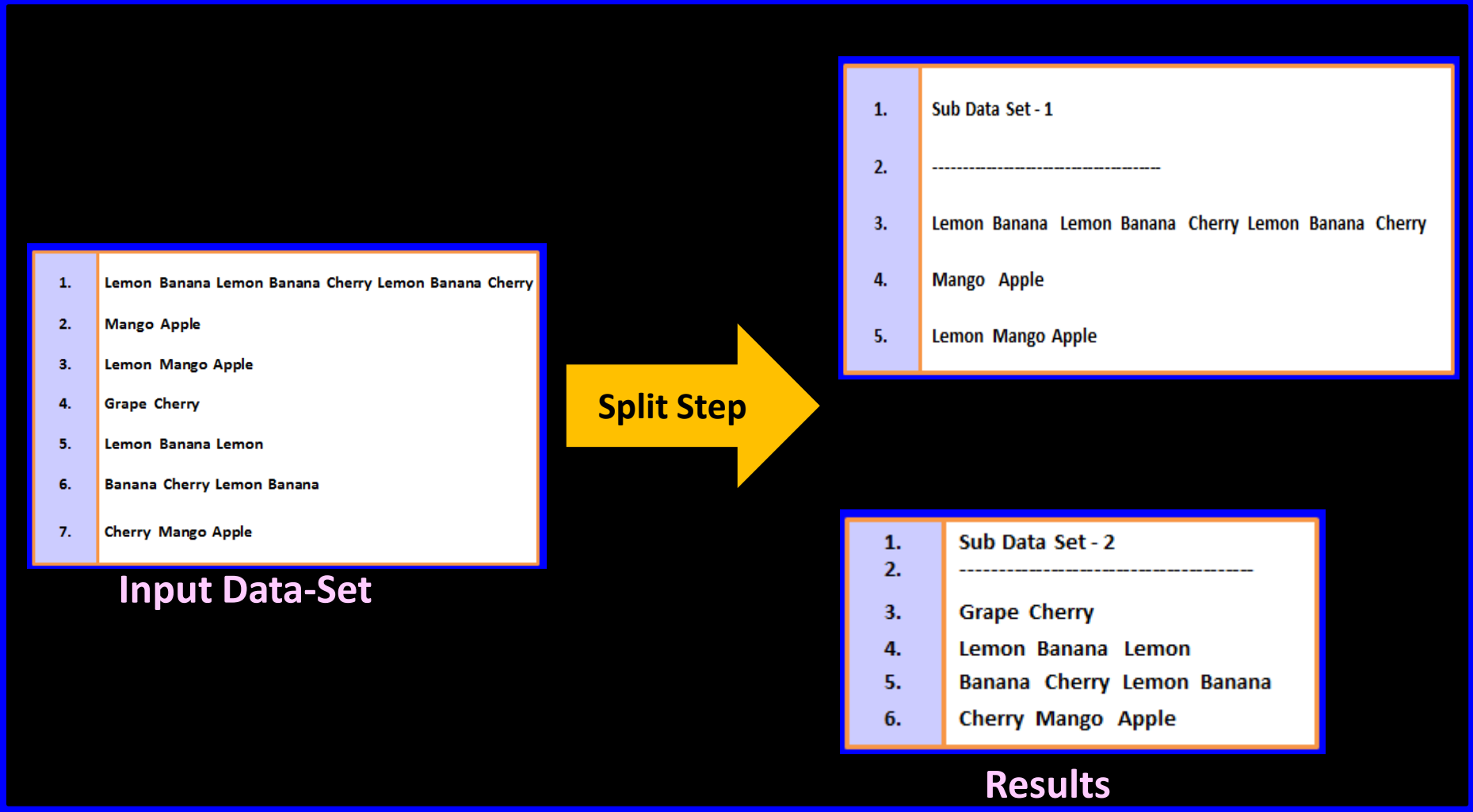
Input Data-Set

1. Lemon Banana Lemon Banana Cherry Lemon Banana Cherry
2. Mango Apple
3. Lemon Mango Apple
4. Grape Cherry
5. Lemon Banana Lemon
6. Banana Cherry Lemon Banana
7. Cherry Mango Apple

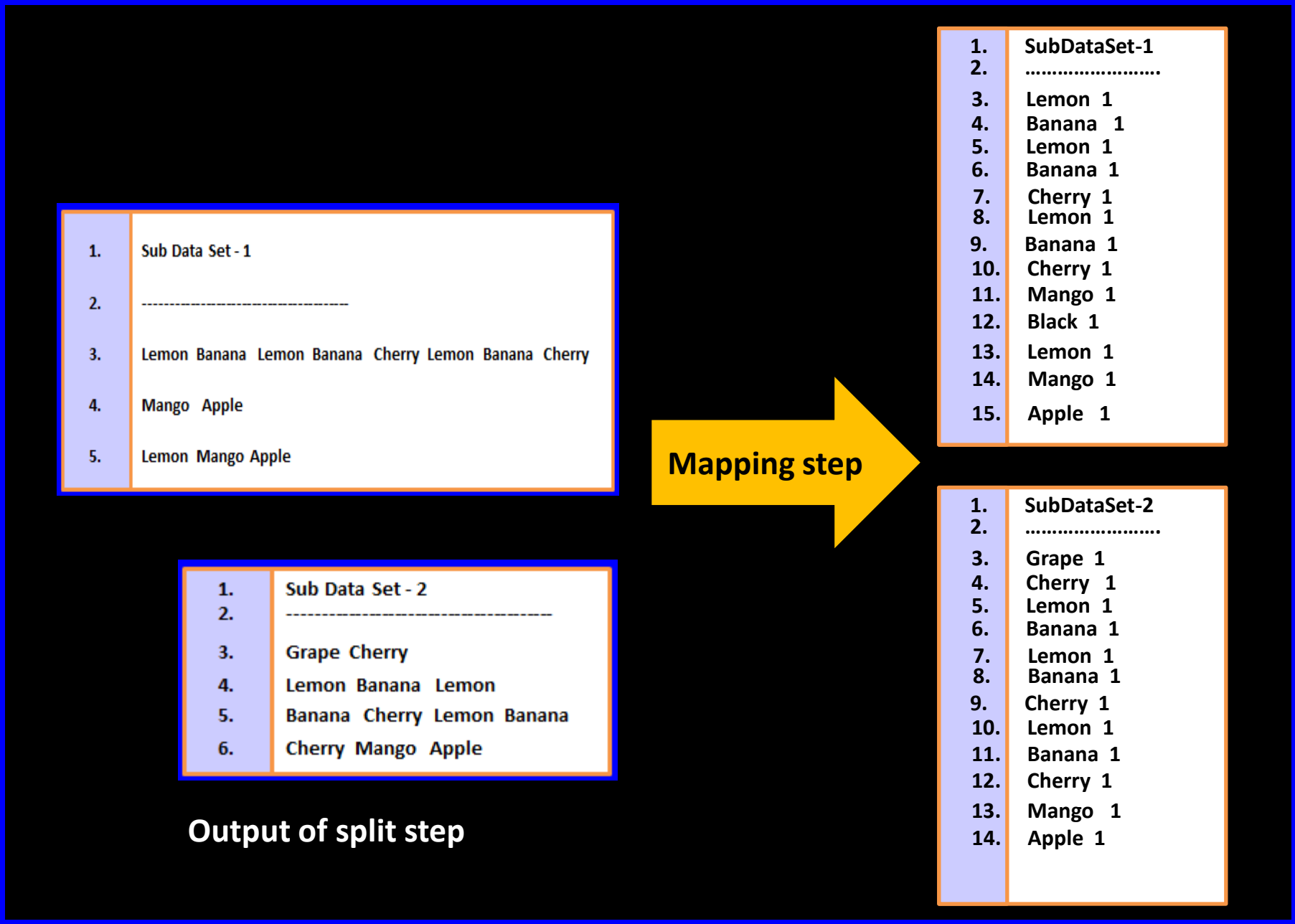
Expected output

1. Apple 3
2. Banana 6
3. Cherry 5
4. Grape 1
5. Lemon 7
6. Mango 3

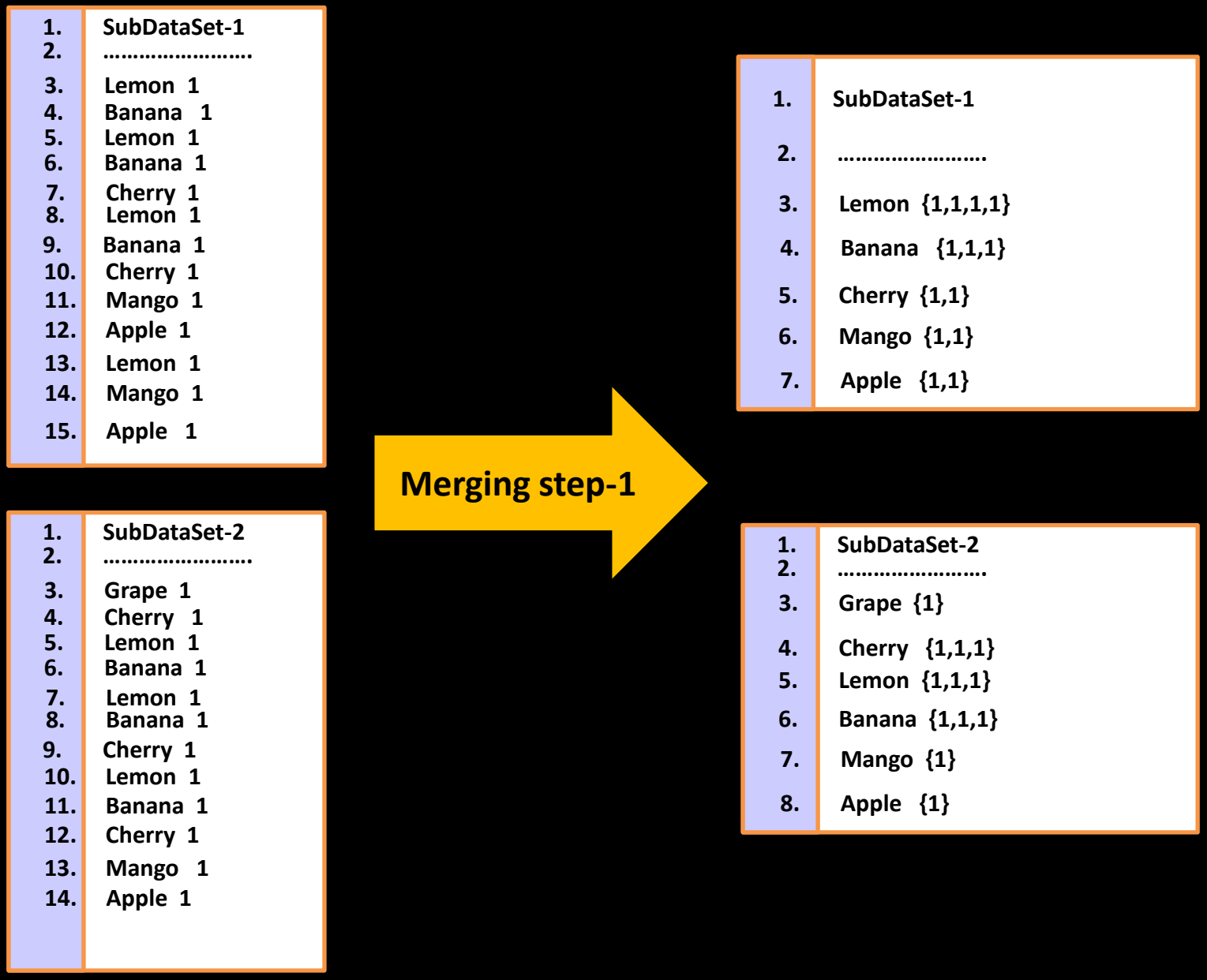
- **Map Function (Split Step)**



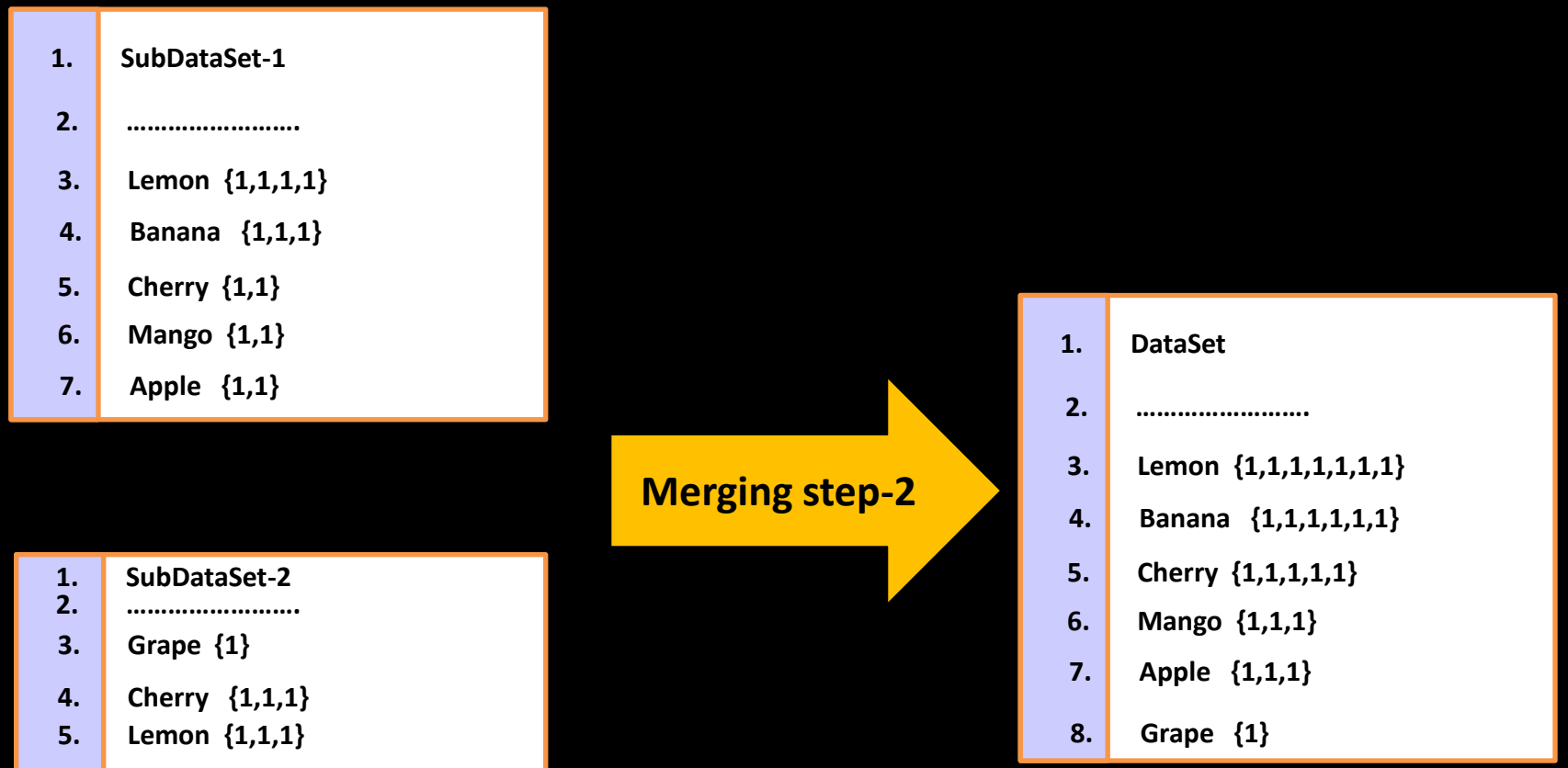
• Map Function (Mapping Step)



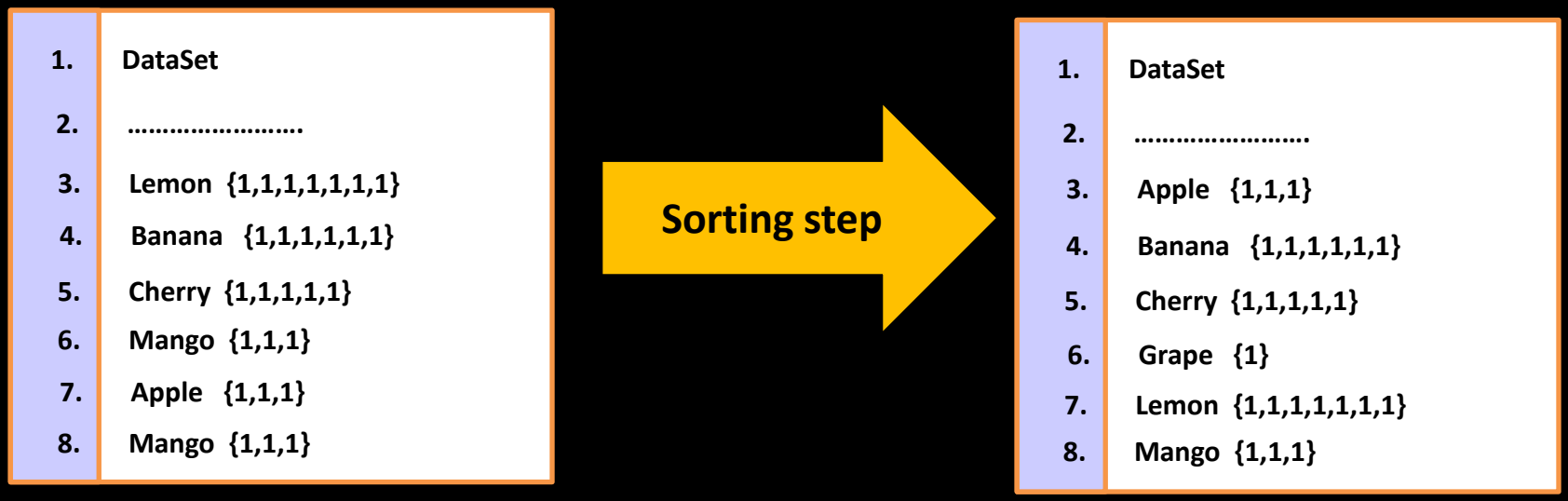
- **Shuffle Function (Merge Step-1)**



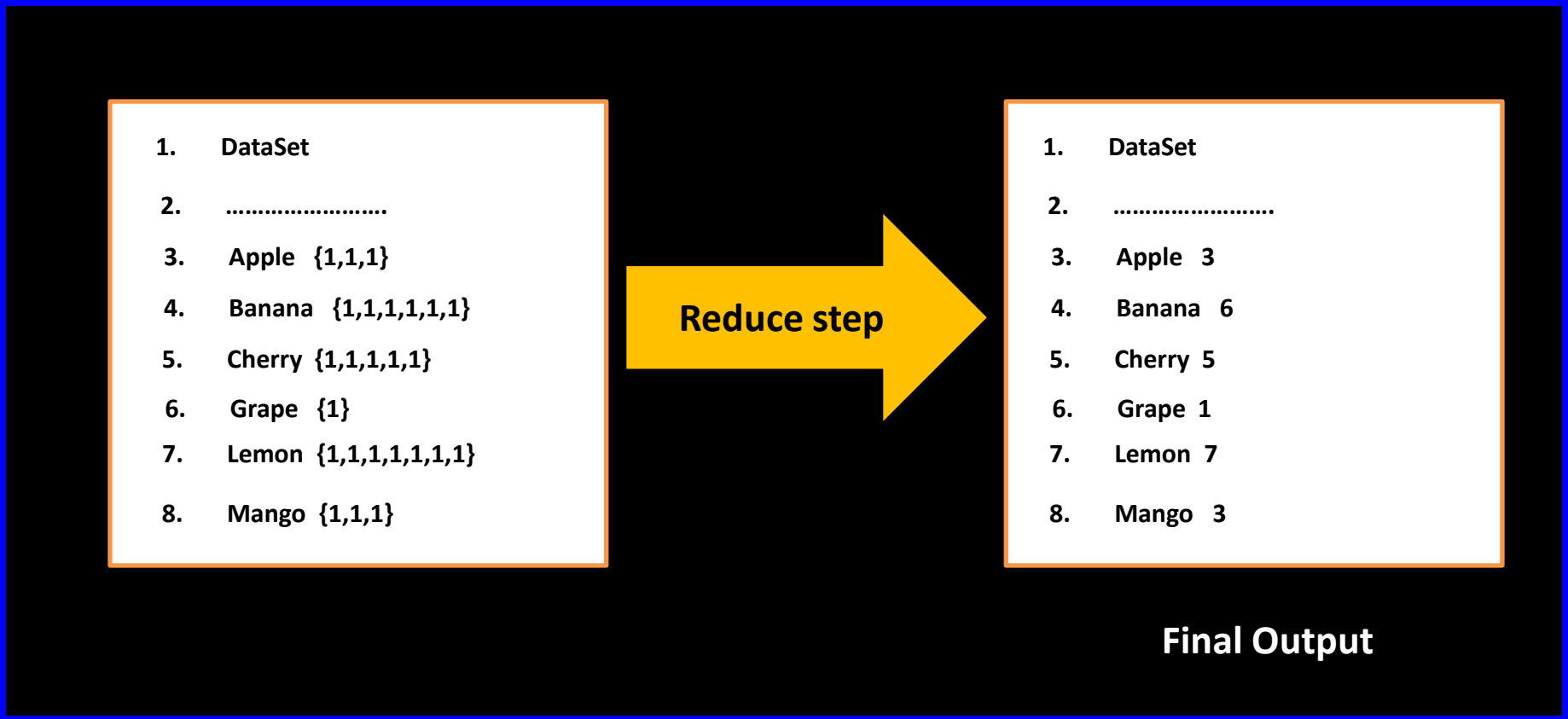
- **Shuffle Function (Merge Step-2)**



- **Shuffle Function (Sorting Step)**



- **Reduce Function (Reduce Step)**



What is Map-Reduce

Map
Reduce

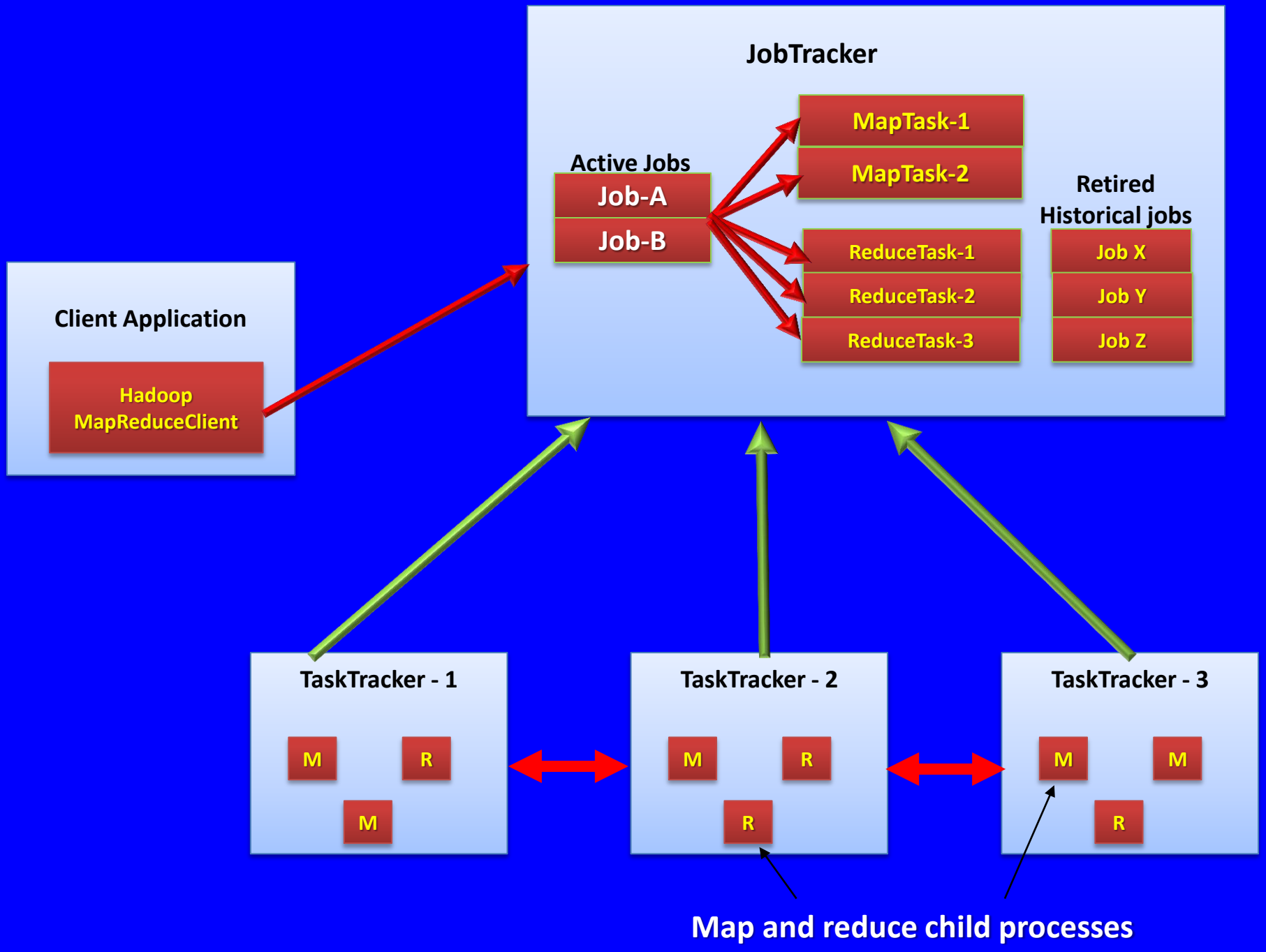
- **Distributed computing framework:** batch based.
- **Parallel works:** over a large amount of raw data.

MapReduce logical architecture

- Hadoop's MapReduce architecture is similar to the master-slave model in HDFS.

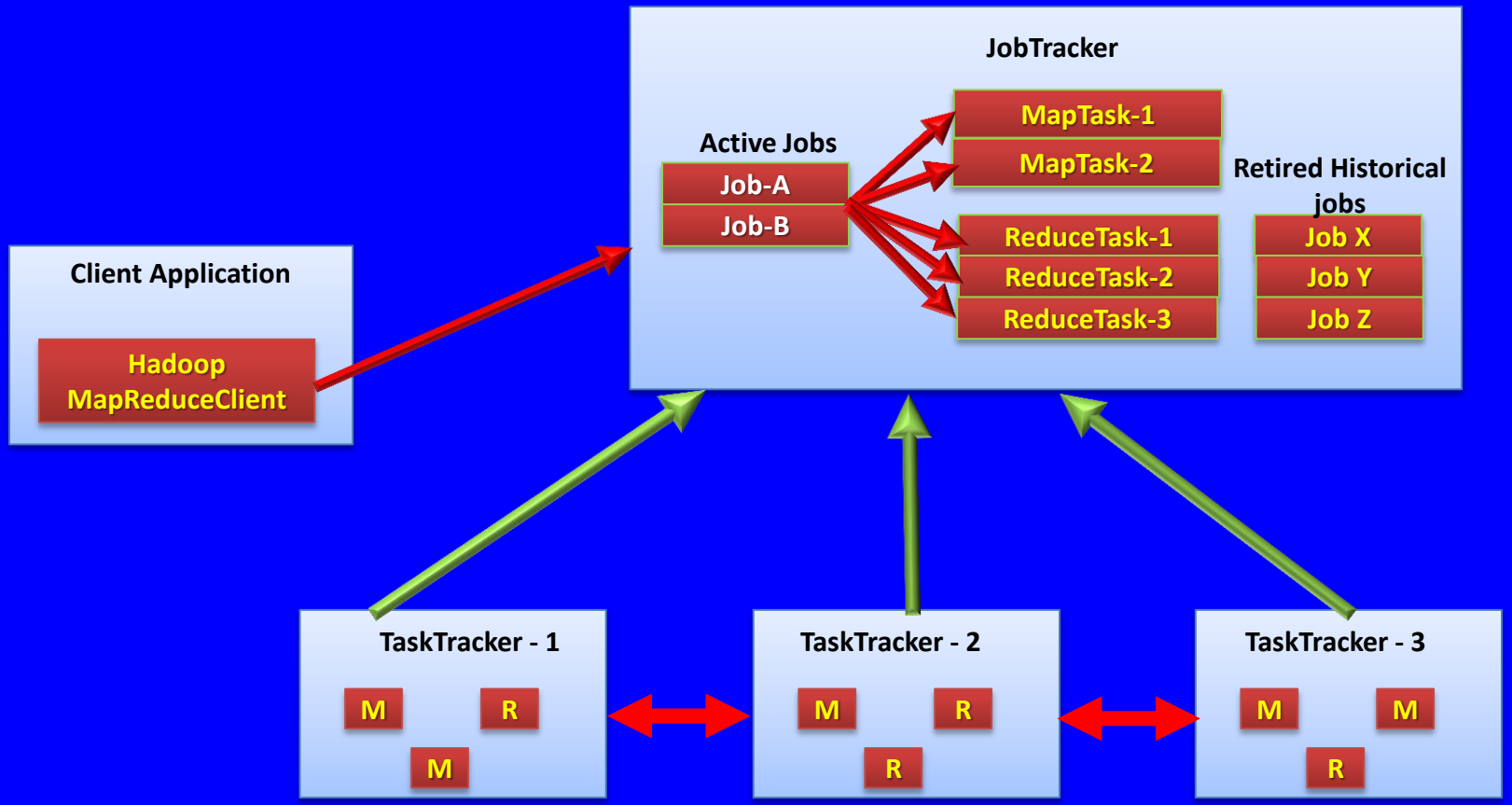
- The main components of MapReduce:

1. Hadoop Map Reduce client
2. Task Tracker
3. Job Tracker


JobTracker
Active Jobs
Job-A
Job-B
MapTask-1
MapTask-2
ReduceTask-1
ReduceTask-2
ReduceTask-3
Retired
Historical jobs
Job X
Job Y
Job Z
Client Application
Hadoop
MapReduceClient
TaskTracker - 1
TaskTracker - 2
TaskTracker - 3
M
R
M
M
R
R
M
M
R
Map and reduce child processes

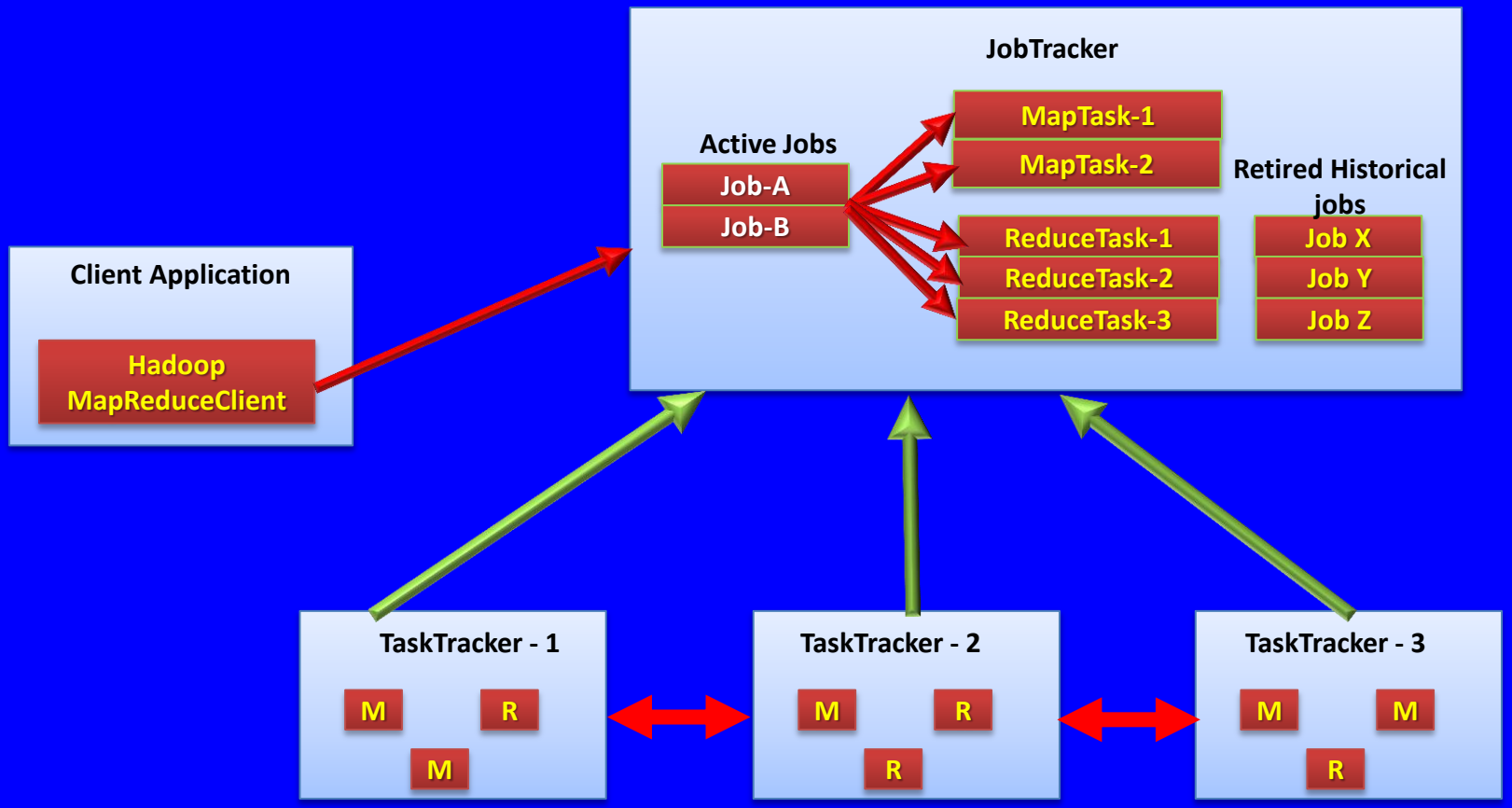
JobTracker

- Number of JobTrackers: There is one JobTracker running on a master node.
- Single point of failure: It is also single point of failure.



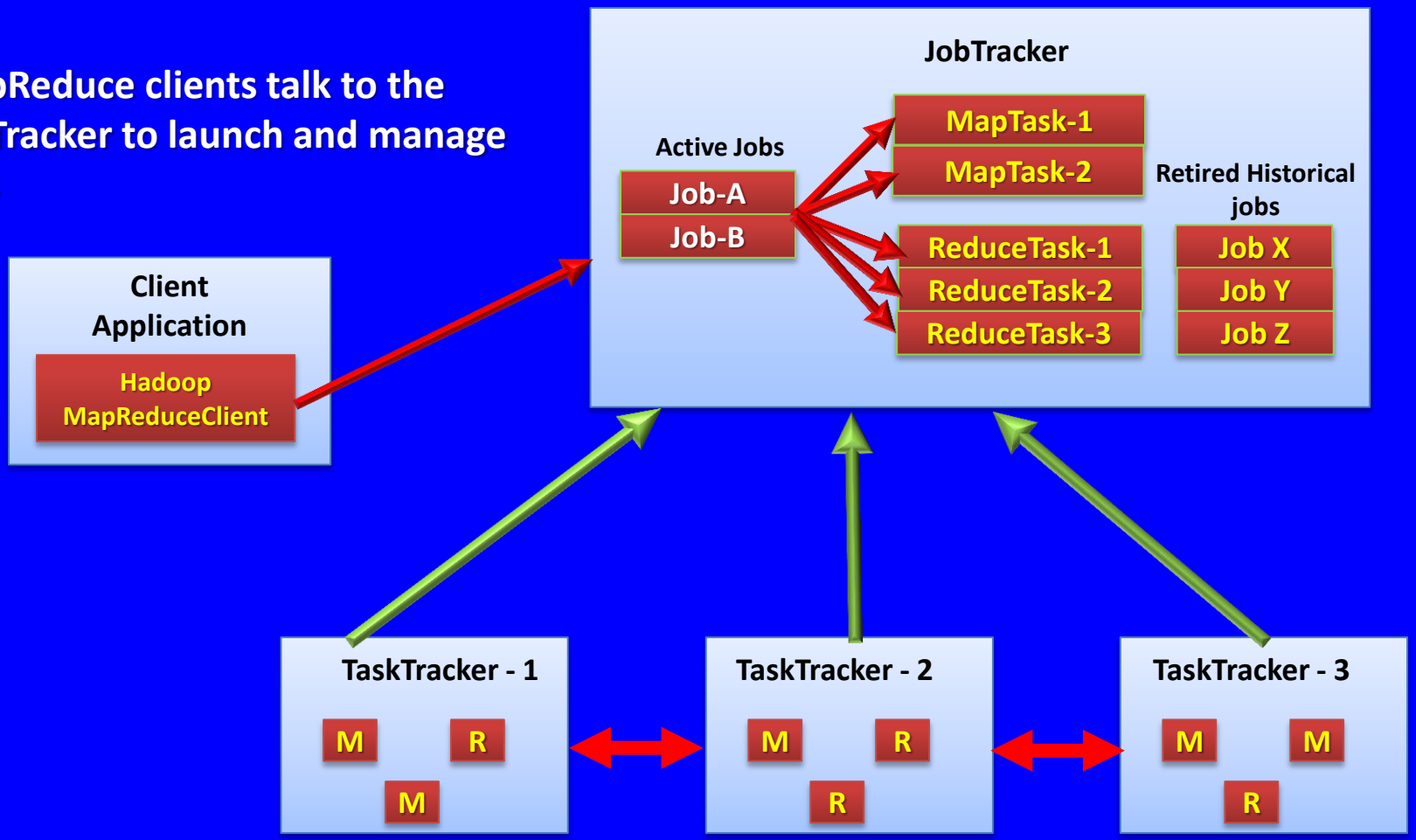
TaskTrackers

- Number of TaskTrackers: several TaskTracker running on slave nodes.

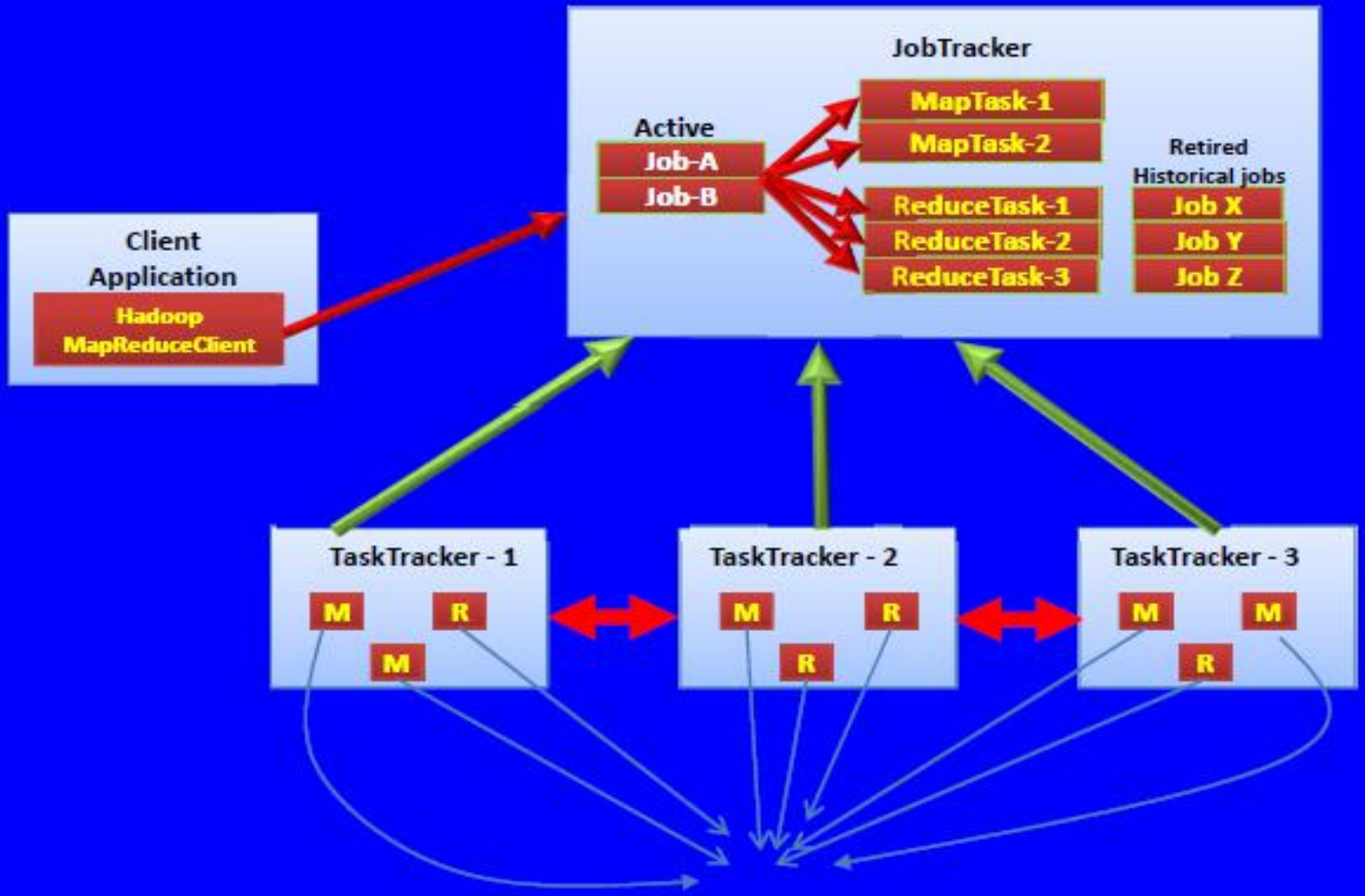


MapReduce Clients

MapReduce clients talk to the JobTracker to launch and manage jobs

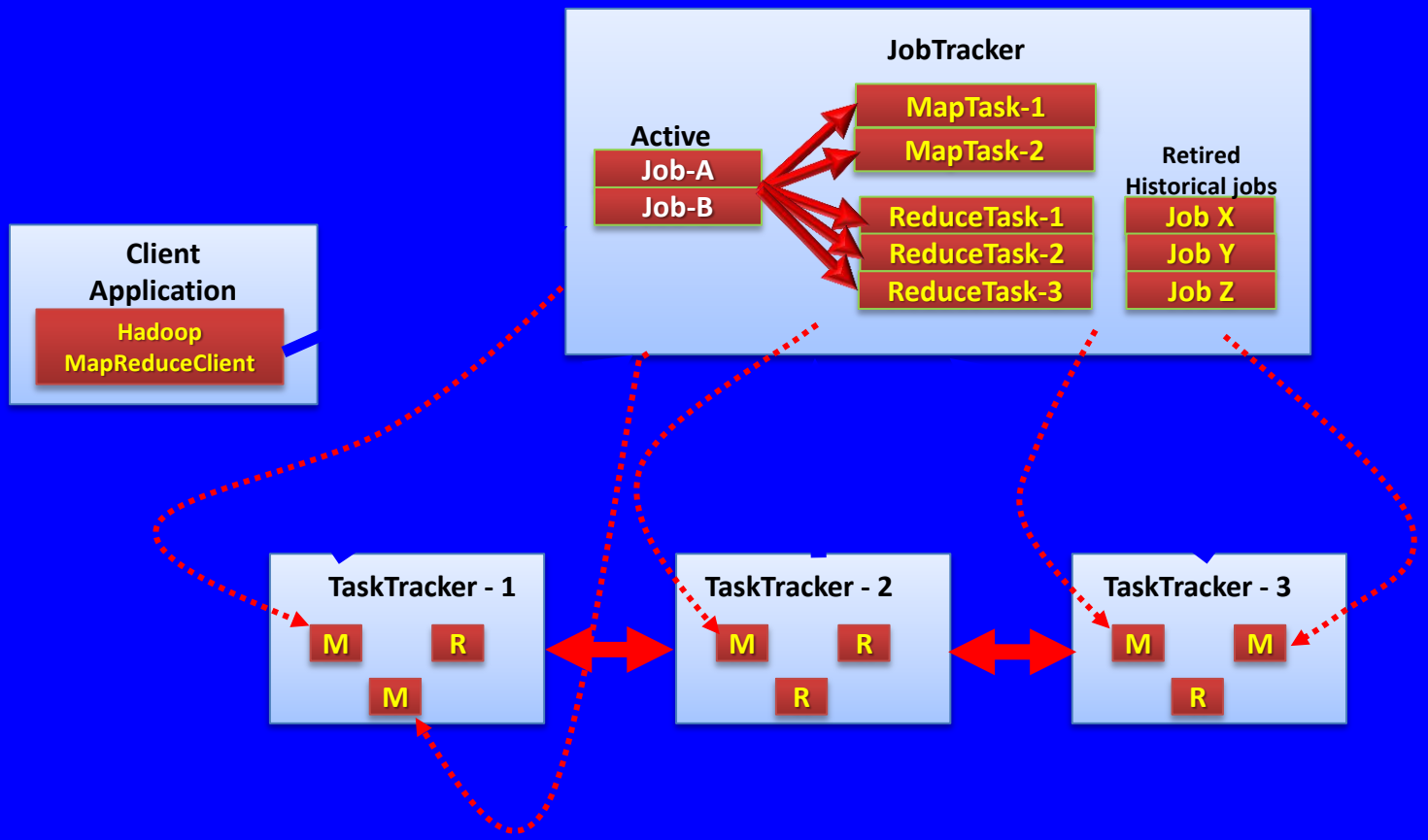


TaskTracker: performs

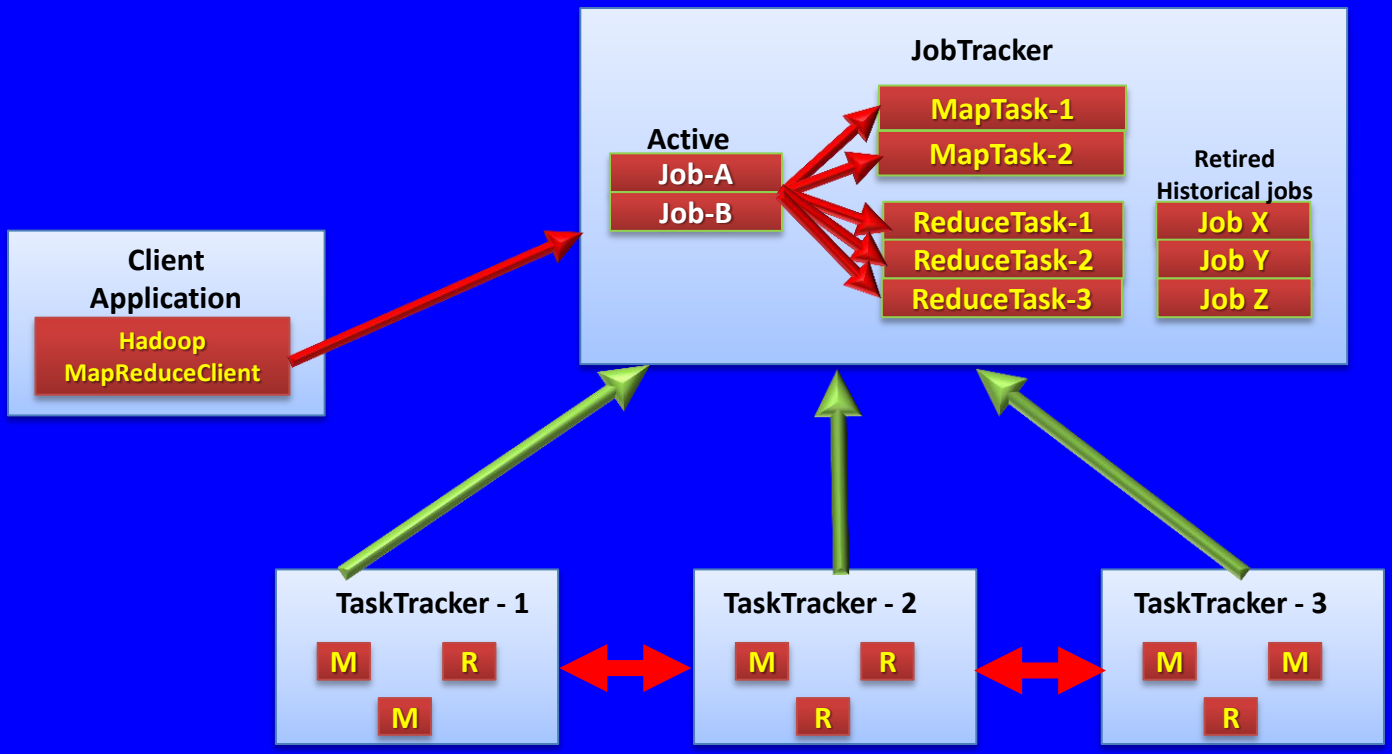


- Child processes spawned by TaskTracker:
  to perform the actual map or reduce work

- Map tasks input: read their input from HDFS
- Map task output: Write their output to the local disk.

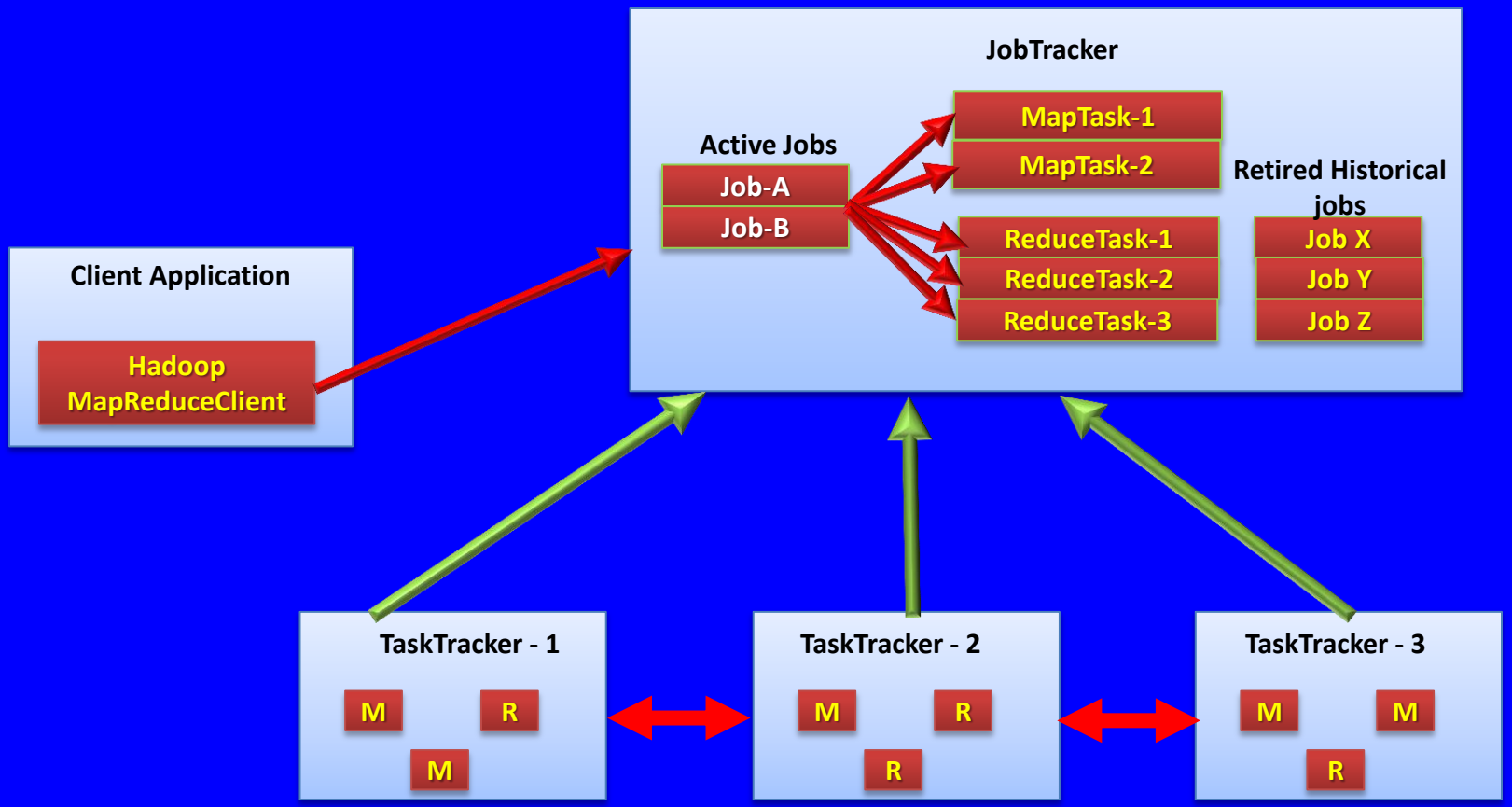


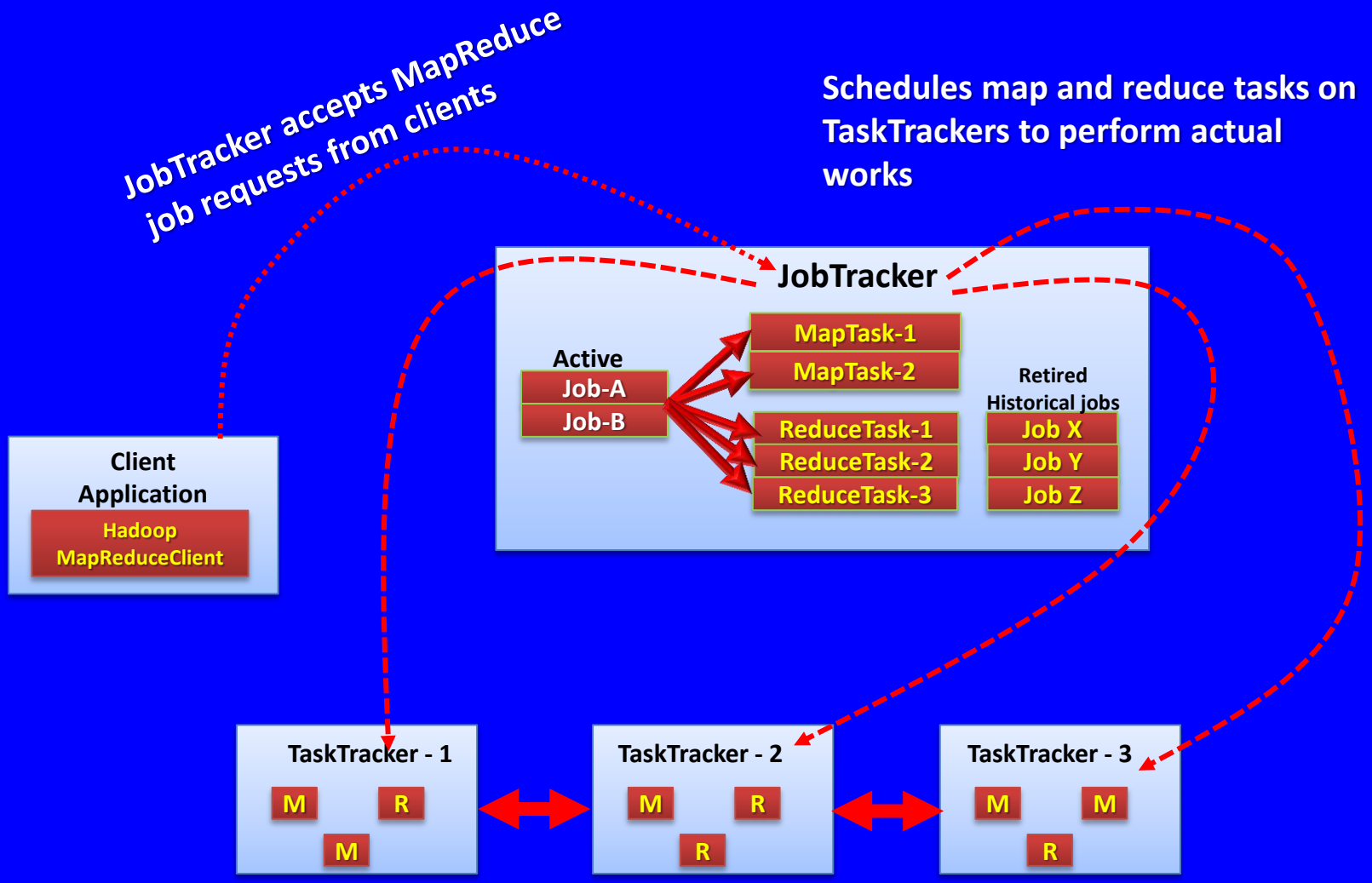
- Reduce tasks inputs: Reduce tasks read the map outputs over the network.
- Reduce tasks outputs: write their outputs back to HDFS



Job Tracker: performs

- Coordination amongst TaskTrackers:
  - JobTracker coordinates activities across the active TaskTrackers.




JobTracker accepts MapReduce job requests from clients
Schedules map and reduce tasks on TaskTrackers to perform actual works
JobTracker
Active
Job-A
Job-B
MapTask-1
MapTask-2
ReduceTask-1
ReduceTask-2
ReduceTask-3
Retired
Historical jobs
Job X
Job Y
Job Z
Client Application
Hadoop MapReduceClient
TaskTracker - 1
M
R
M
TaskTracker - 2
M
R
R
TaskTracker - 3
M
M
R

Step - II

Hadoop: Third step

Configuration and Setups

Single node cluster

Setting Hadoop cluster On Ubuntu Linux

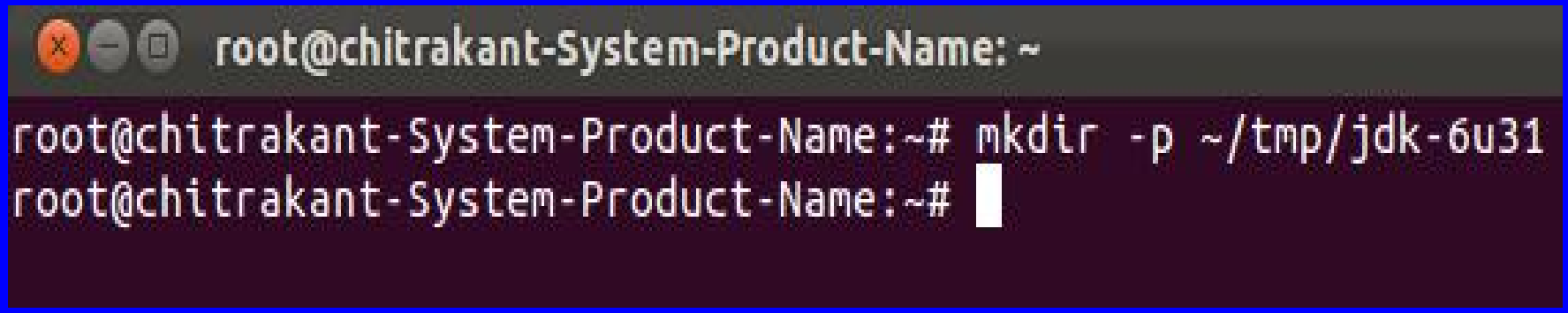
Step – 1

Install Sun Java

Installing the JDK manually

- **Download the adequate JDK from the Oracle Web site.**

- Create temporary directory to keep the installer:



❖ **Copy the file to the tmp location:**

```
root@chitrakant-System-Product-Name: ~/tmp/jdk-6u31
root@chitrakant-System-Product-Name:~/tmp/jdk-6u31# ls
jdk-6u31-linux-i586.bin
root@chitrakant-System-Product-Name:~/tmp/jdk-6u31# cp /root/Desktop/Required_So
ftwares/java/jdk-6u31-linux-i586.bin .
```

❖ **Provide various permissions (make it executable):**

```
root@chitrakant-System-Product-Name: ~/tmp/jdk-6u31
root@chitrakant-System-Product-Name:~/tmp/jdk-6u31# ls
jdk-6u31-linux-i586.bin
root@chitrakant-System-Product-Name:~/tmp/jdk-6u31# chmod 777 jdk-6u31-linux-i58
6.bin
root@chitrakant-System-Product-Name:~/tmp/jdk-6u31# ls
jdk-6u31-linux-i586.bin
root@chitrakant-System-Product-Name:~/tmp/jdk-6u31#
```

❖ Create target directory for java:

```
root@chitrakant-System-Product-Name: /
root@chitrakant-System-Product-Name:/# ls
bin   dev   initrd.img  media  proc  sbin      sys  var
boot  etc   lib         mnt    root  selinux   tmp  vmlinuz
cdrom home  lost+found  opt    run   srv       usr
root@chitrakant-System-Product-Name:/# mkdir java
root@chitrakant-System-Product-Name:/#
```

1

❖ Copy executable file into the created java directory:

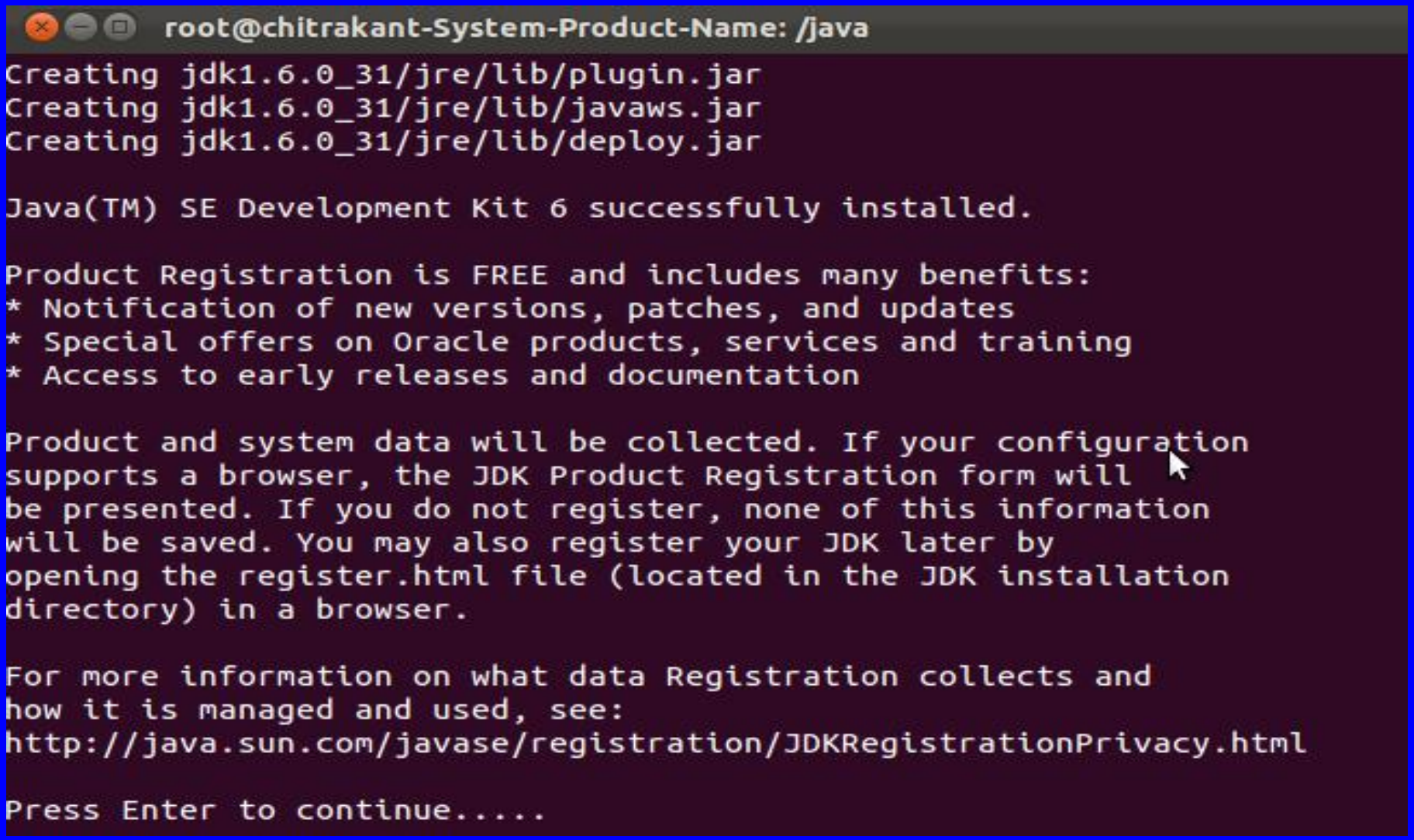
2

```
root@chitrakant-System-Product-Name:~/tmp/jdk-6u31# cp jdk-6u31-linux-i586.bin /
java/
root@chitrakant-System-Product-Name:~/tmp/jdk-6u31#
```

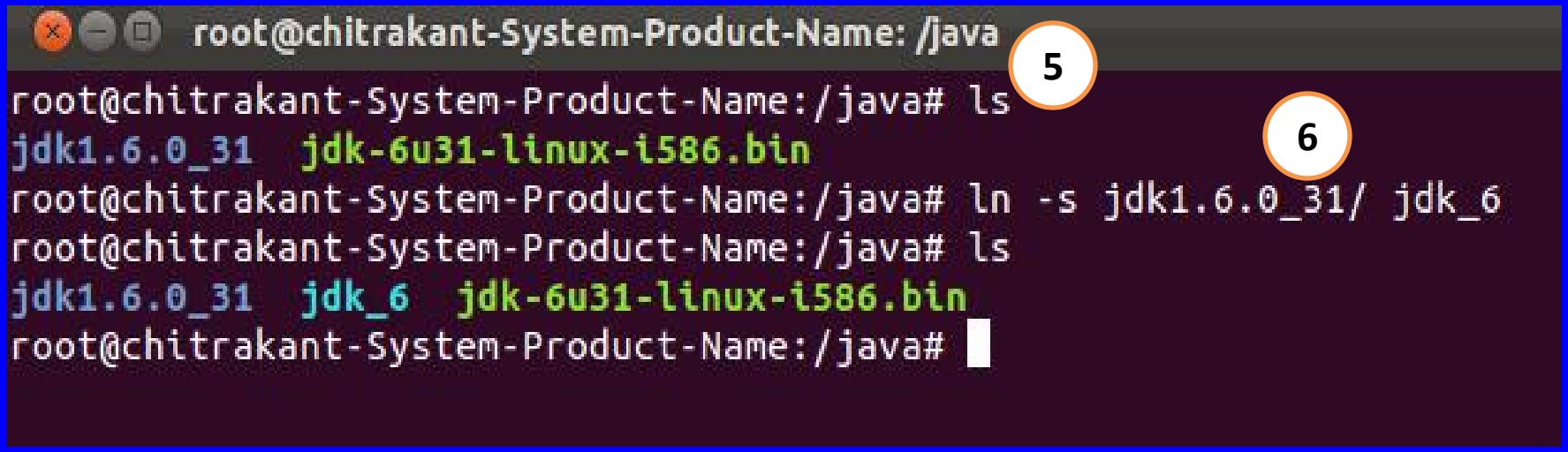
❖ Start java installation :

```
$ ./jdk-6u31-linux-i586.bin
```

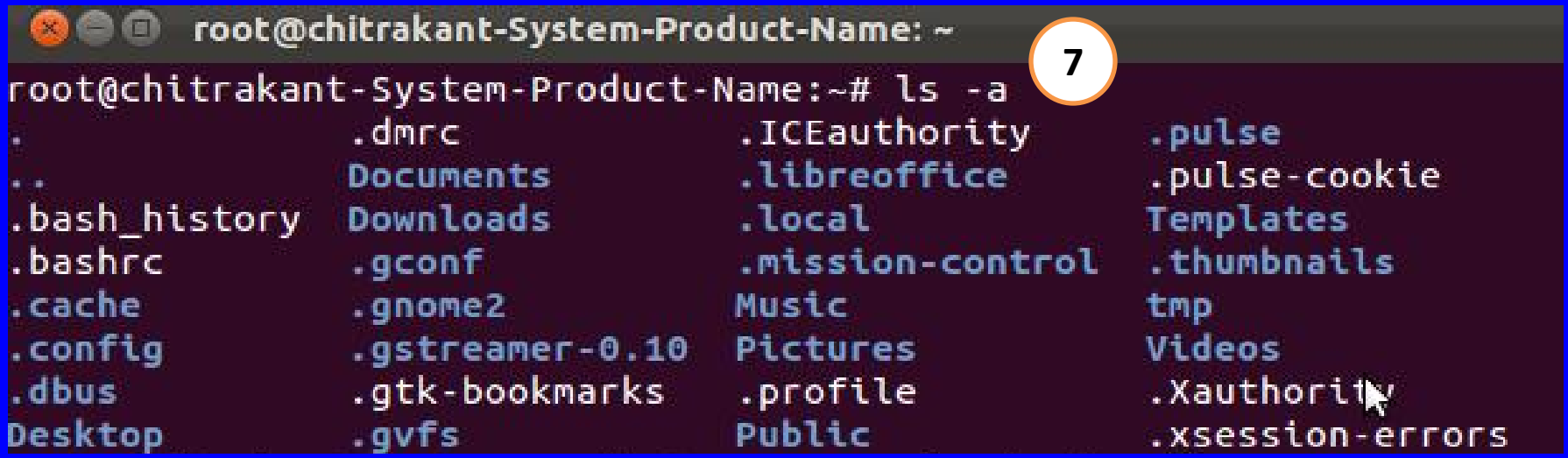
3



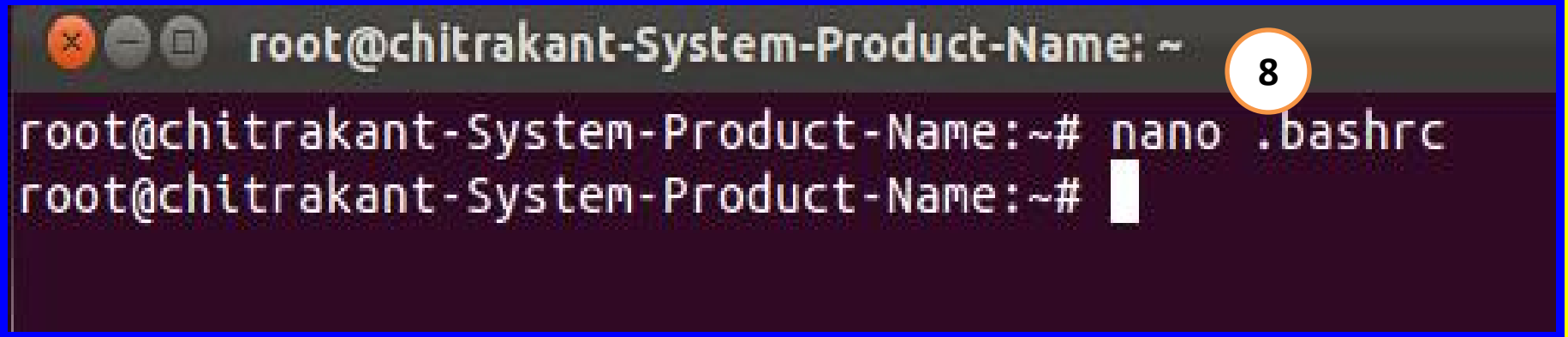
❖ create a symbolic link:



- Add to the ~/.bashrc the path to our JDK binary files

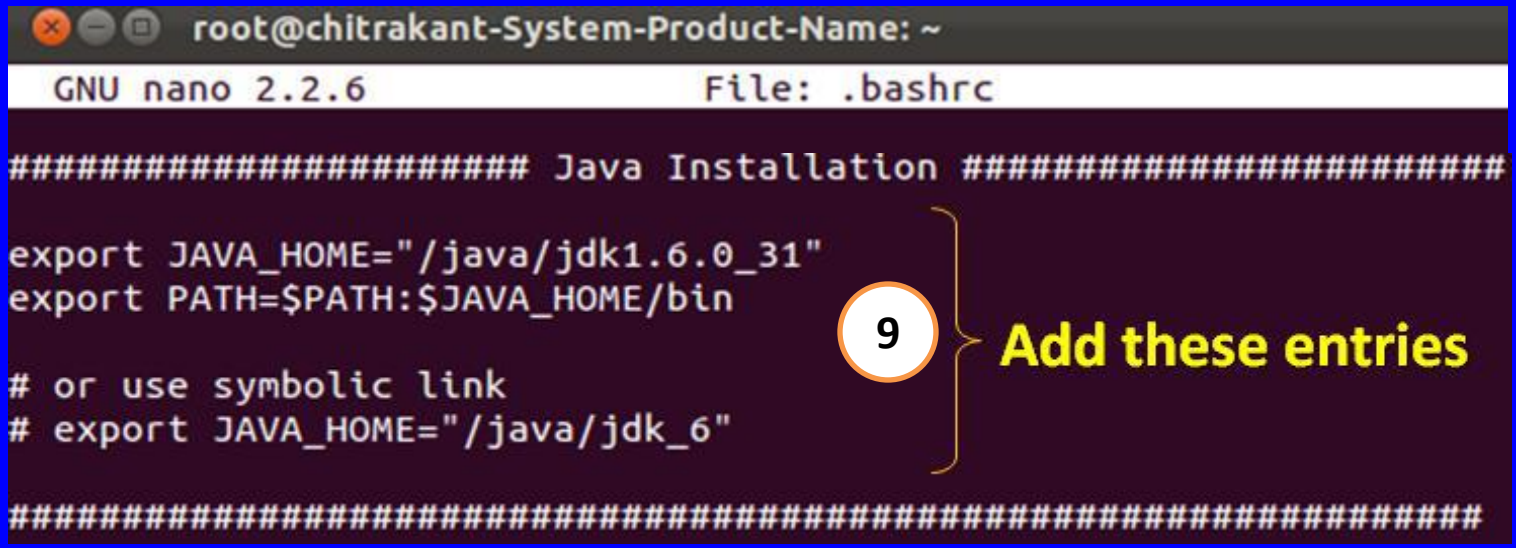


❖ Open .bashrc file to add java installation

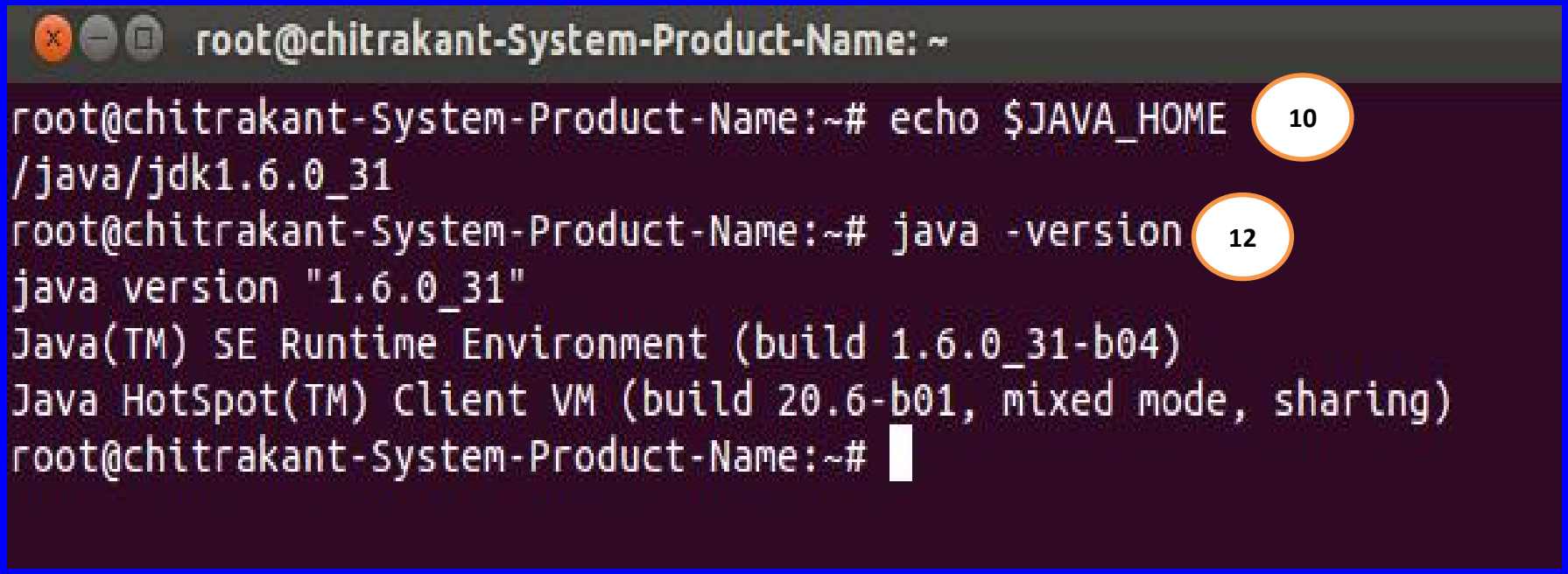


Set JAVA_HOME

- **Now set the JAVA_HOME environment variable to allow applications to find where my JDK is installed.**



- Close the current terminal and start new terminal.
- Apply the command : java -version



Step - 1

Step – 2

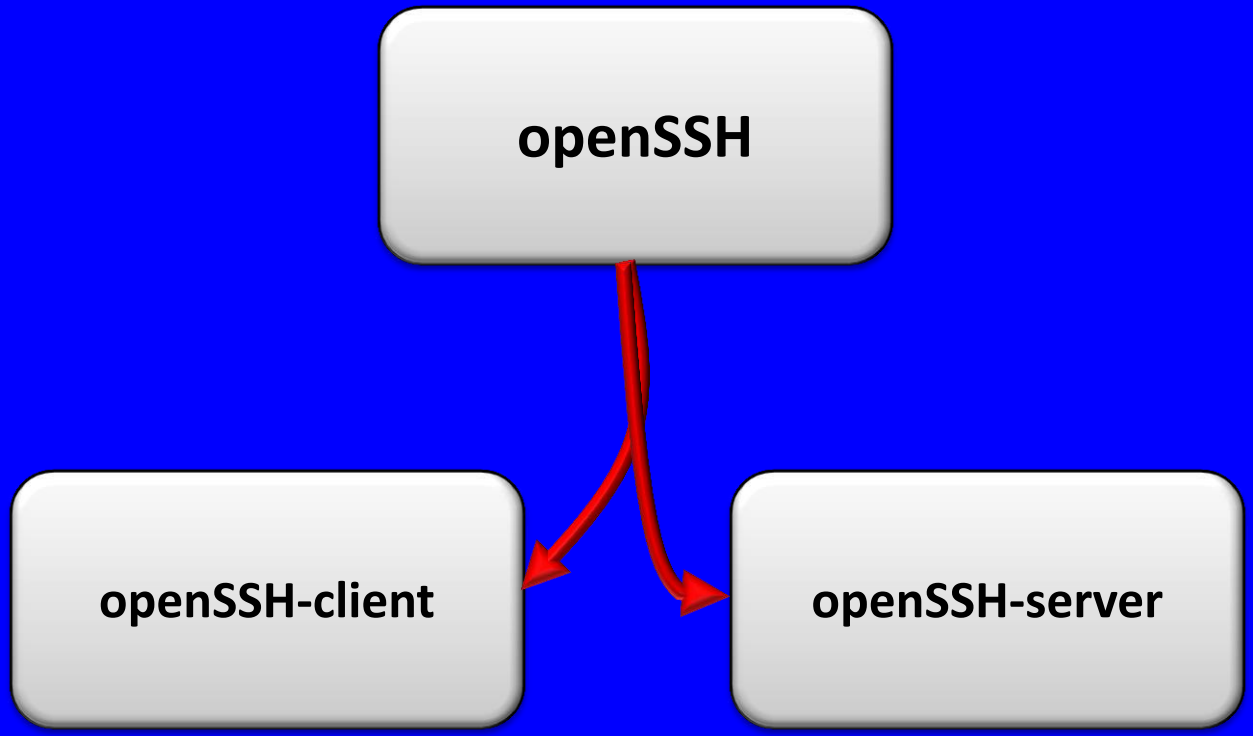
Install SSH

SSH

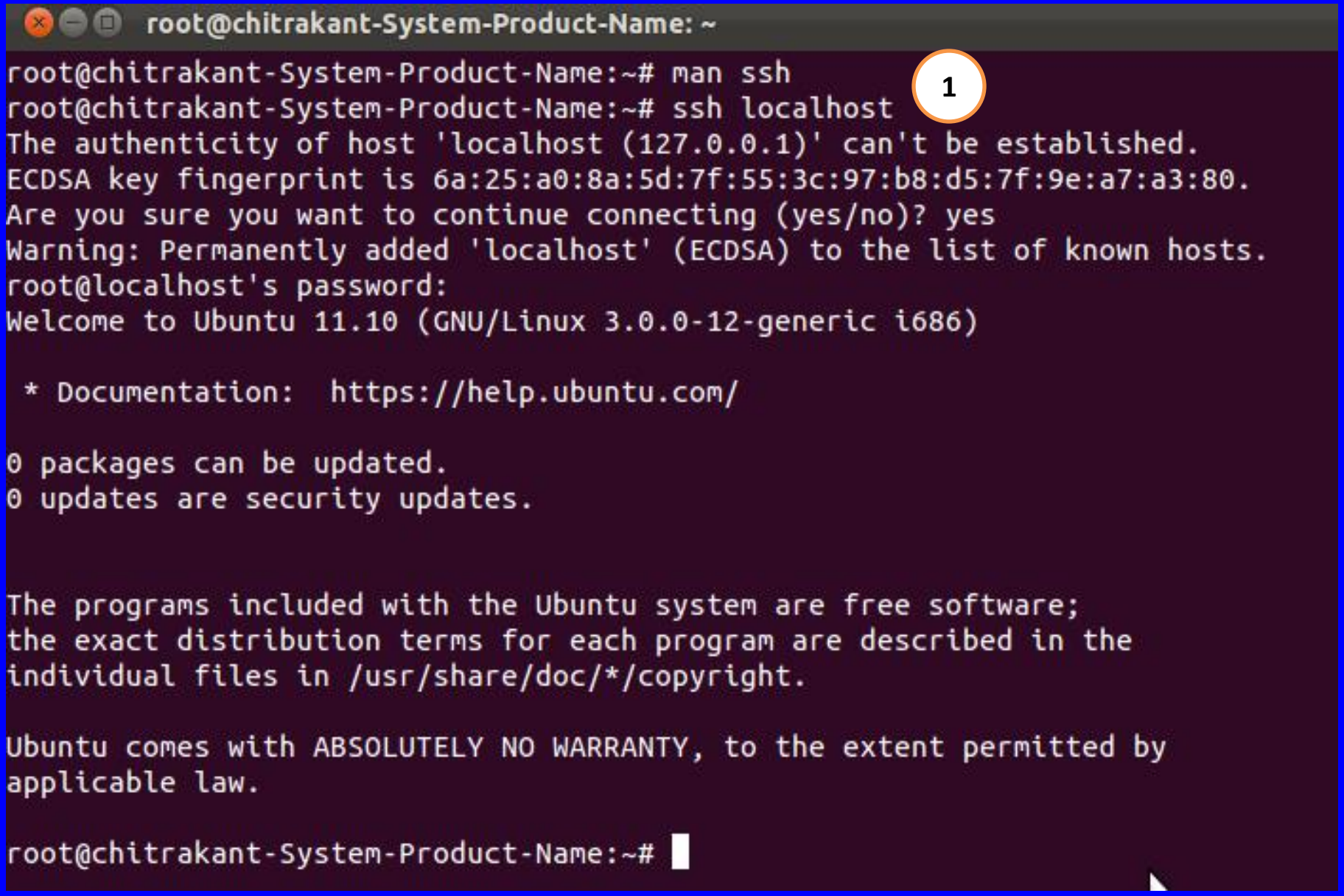
- Hadoop requires SSH access to manage its nodes:
  - Remote machines
  - Or local machine if used Hadoop on it.
- For single-node Hadoop setup: configure SSH access to localhost .

OpenSSH

- The Ubuntu flavor of SSH is called OpenSSH.



ssh–client already installed

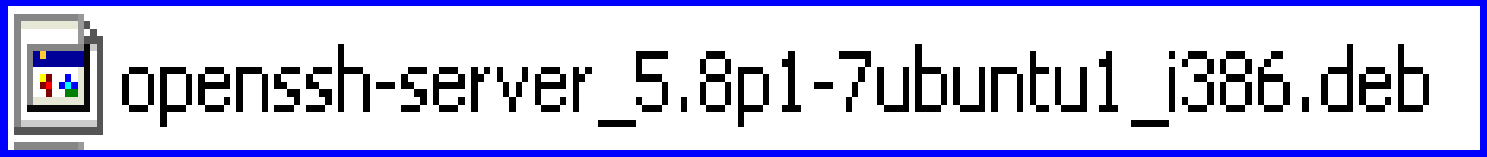


❖ Install openssh online:

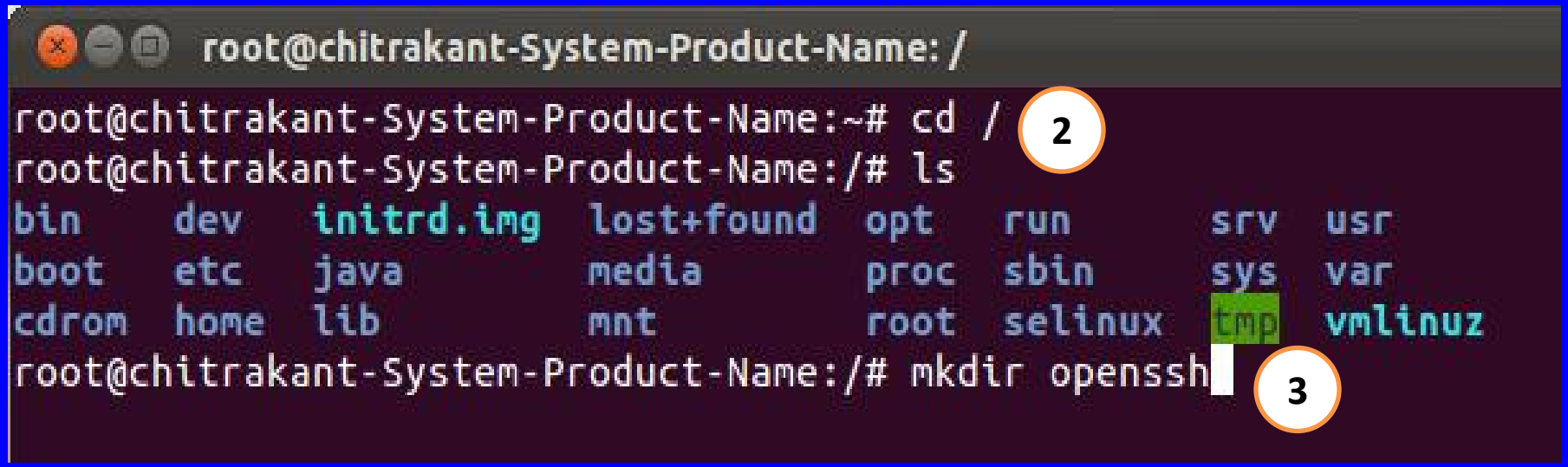
```
$ apt-get   install   openssh-client  openssh-server
```

❖ Manually Install openSSH:

❖ Download openssh installer from their web site:



❖ Create a directory to store openssh installer:



❖ Copy openssh installer into openssh directory:

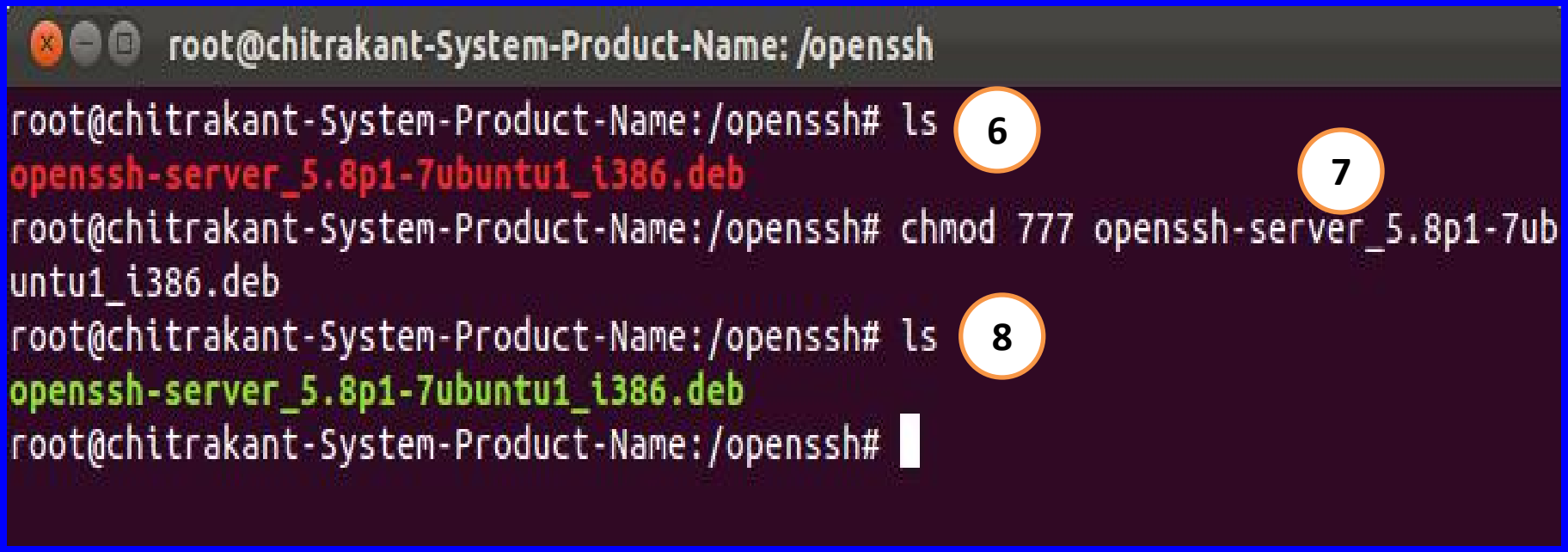
4

```
root@chitrakant-System-Product-Name:/# cd openssh/
root@chitrakant-System-Product-Name:/openssh# ls
root@chitrakant-System-Product-Name:/openssh# cp /root/Desktop/Required_Software
s/other_tools/openssh/openssh-server_5.8p1-7ubuntu1_i386.deb .
root@chitrakant-System-Product-Name:/openssh# ls
openssh-server_5.8p1-7ubuntu1_i386.deb
root@chitrakant-System-Product-Name:/openssh#
```

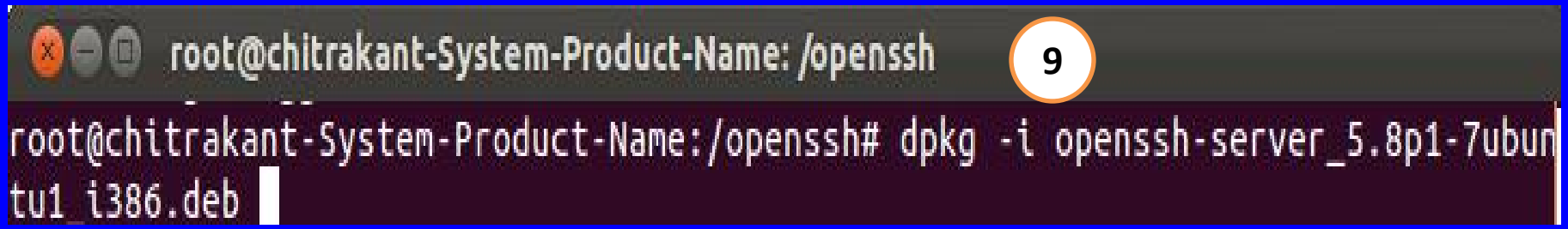
5

5.1

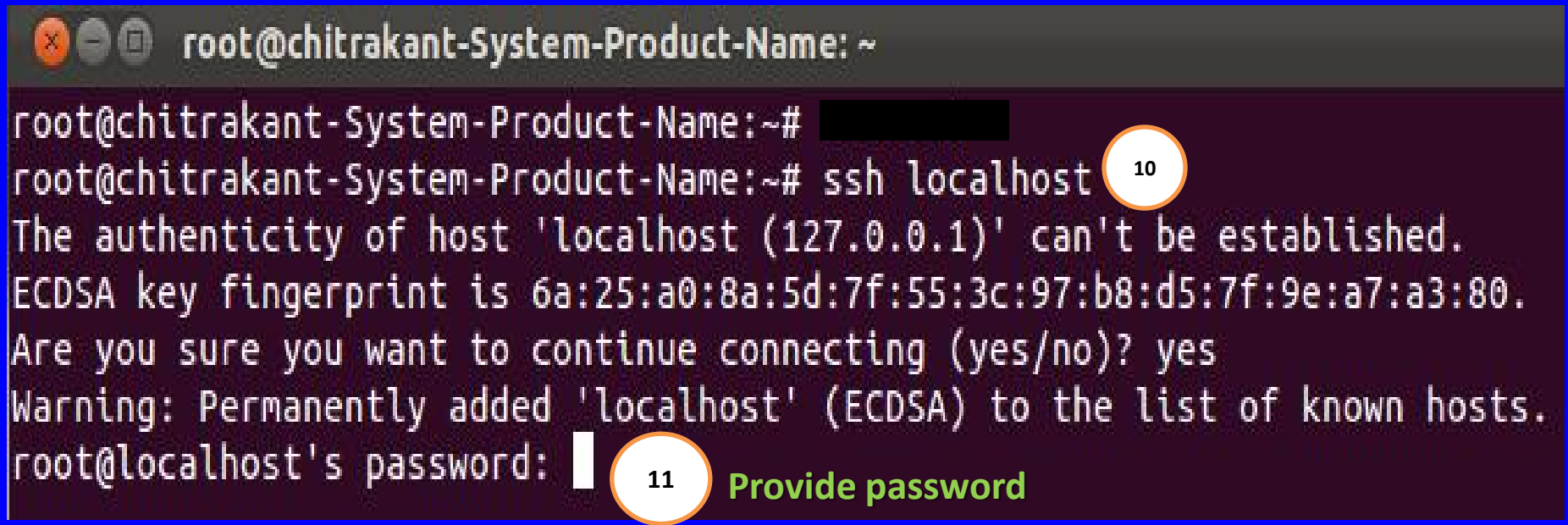
❖ **Make openssh installer as executable:**



❖ **Install openssh using the command: dpkg -i**



❖ Apply ssh command to connect with the localhost:



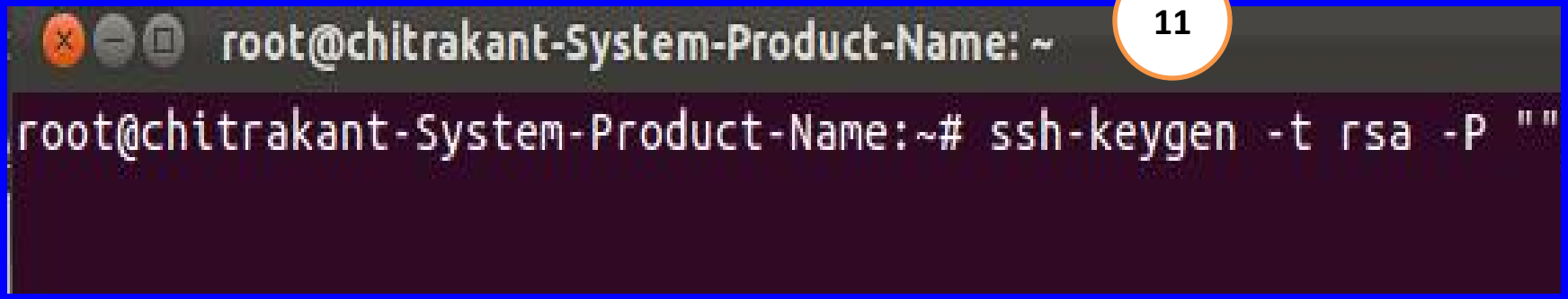
- ❖ Asks for the password and connected.
- ❖ But it should be able to connect without providing password.

Generate an SSH key

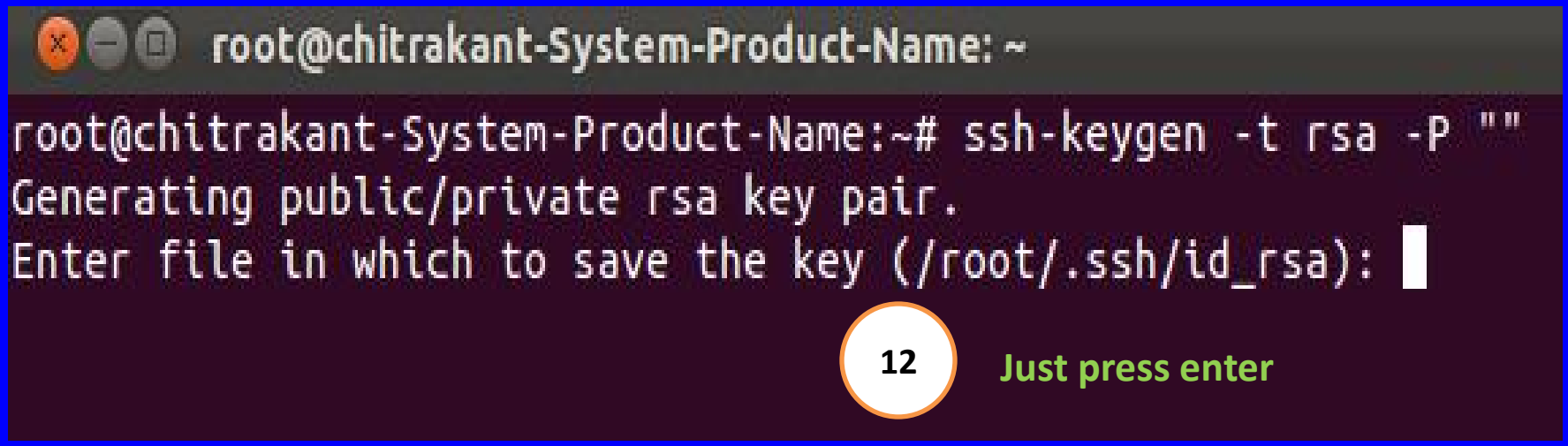
- First, we have to generate an SSH key for the user account
- Here we use for root account.

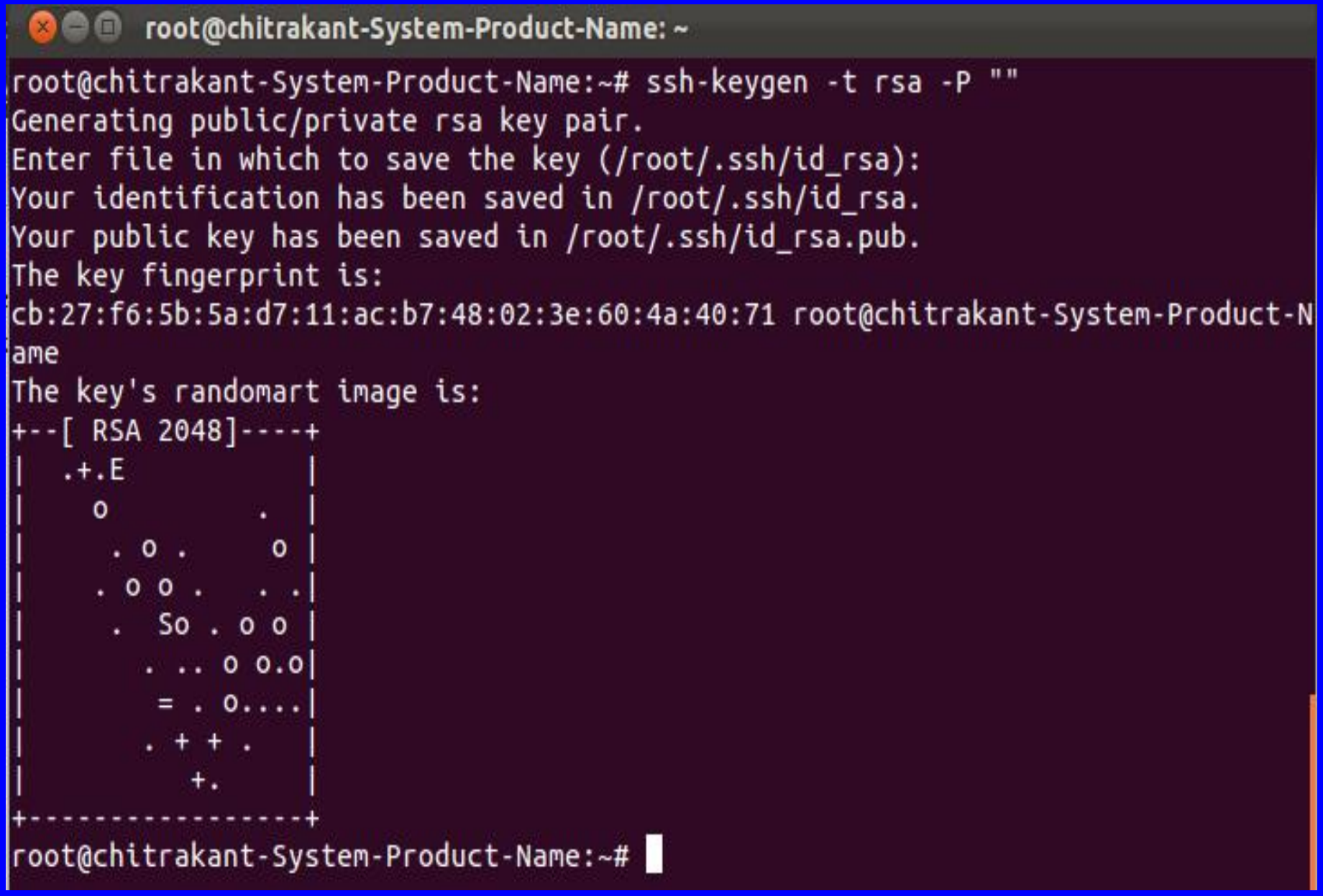
❖ ssh-keygen program:

- The ssh-keygen program generates a public/private key pair.
- That generated key allows password-less access to any remote server which has our public key stored in its authorized_keys file.



- It asks for the file name, but for simple case do not provide any file name.




root@chitrakant-System-Product-Name: ~
root@chitrakant-System-Product-Name:~# ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
cb:27:f6:5b:5a:d7:11:ac:b7:48:02:3e:60:4a:40:71 root@chitrakant-System-Product-N
ame
The key's randomart image is:
+--[RSA 2048]----+
| .+.E |
| o   . |
| . o .   o |
| . o o .   . .|
| . So . o o |
| . .. o o.o|
| = . o....|
| . + + . |
| +. |
+-----------------+
root@chitrakant-System-Product-Name:~#

❖ **Enable SSH access to our local machine with this newly created key:**

- **To access ssh command without requiring a password:**
- **Following steps are used for enabling public-key authentication, but instead of entering a passphrase when prompted, simply press Enter.**

```
root@chitrakant-System-Product-Name:~# echo $HOME
/root
root@chitrakant-System-Product-Name:~# cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

```
root@chitrakant-System-Product-Name:~# echo $HOME
/root
root@chitrakant-System-Product-Name:~# cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/a
uthorized_keys
root@chitrakant-System-Product-Name:~# ssh localhost
Welcome to Ubuntu 11.10 (GNU/Linux 3.0.0-12-generic i686)

 * Documentation:  https://help.ubuntu.com/

0 packages can be updated.
0 updates are security updates.

Last login: Tue Jan  1 07:14:18 2002 from localhost
root@chitrakant-System-Product-Name:~#
```

Step - 2

Step – 3

Disabling IPv6

Why?

- There’s no practical point in enabling IPv6 on a box when we are not connected to any IPv6 network.

- Hence, simply disabled IPv6 on Ubuntu machine.

How To Disable IPv6 Under Ubuntu 10.04/10.10/11.04

1. On Ubuntu (10.04/10.10/11.04), launch the terminal and issue this command to check whether IPv6 is enabled or not:

```
$ cat /proc/sys/net/ipv6/conf/all/disable_ipv6
```

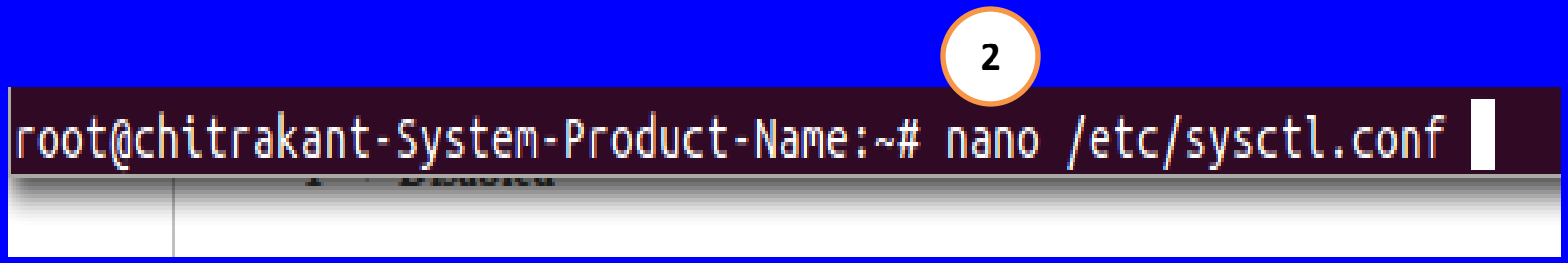
0 --> Enabled
1 --> Disabled

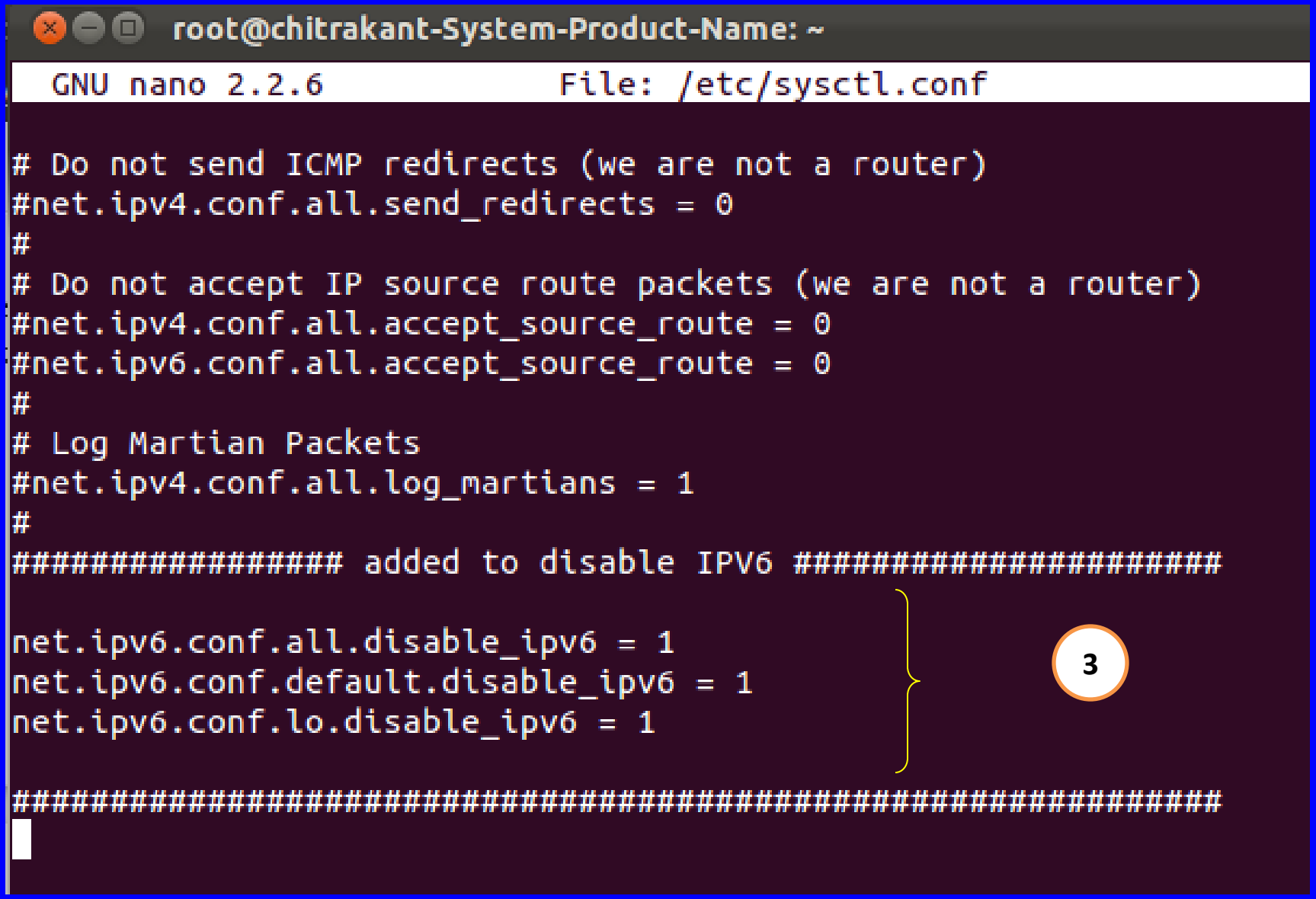
```
root@chitrakant-System-Product-Name:~# cat /proc/sys/net/ipv6/conf/all/disable_ipv6
```

```
root@chitrakant-System-Product-Name:~# nano /etc/sysctl.conf
root@chitrakant-System-Product-Name:~# nano /etc/sysctl.conf
root@chitrakant-System-Product-Name:~# sysctl -p
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
root@chitrakant-System-Product-Name:~# cat /proc/sys/net/ipv6/conf/all/disable_i
pv6
1
root@chitrakant-System-Product-Name:~#
```

2. To disable IPv6, we need to edit the 'sysctl.conf' file.

- So, via the terminal, issue this command:




root@chitrakant-System-Product-Name: ~
GNU nano 2.2.6 File: /etc/sysctl.conf
Do not send ICMP redirects (we are not a router)
#net.ipv4.conf.all.send_redirects = 0
#
Do not accept IP source route packets (we are not a router)
#net.ipv4.conf.all.accept_source_route = 0
#net.ipv6.conf.all.accept_source_route = 0
#
Log Martian Packets
#net.ipv4.conf.all.log_martians = 1
#
###################### added to disable IPV6 ##########################
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
3
##

3. Issue now this command to reload your configuration:

```
root@chitrakant-System-Product-Name:~# sysctl -p
```

4

IPv6 is now disabled, you can check its status with the first command given above (1 --> disabled).

```
root@chitrakant-System-Product-Name:~# sysctl -p
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
root@chitrakant-System-Product-Name:~#
```

```
root@chitrakant-System-Product-Name:~# cat /proc/sys/net/ipv6/conf/all/disable_ipv6
```

5

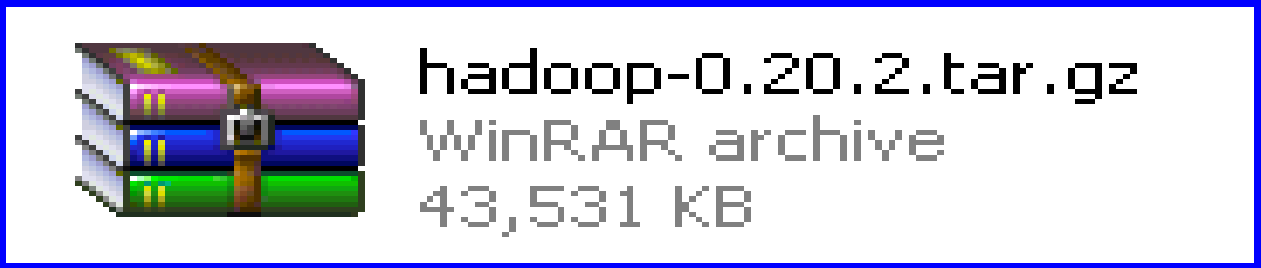
Step - 3

Step – 4

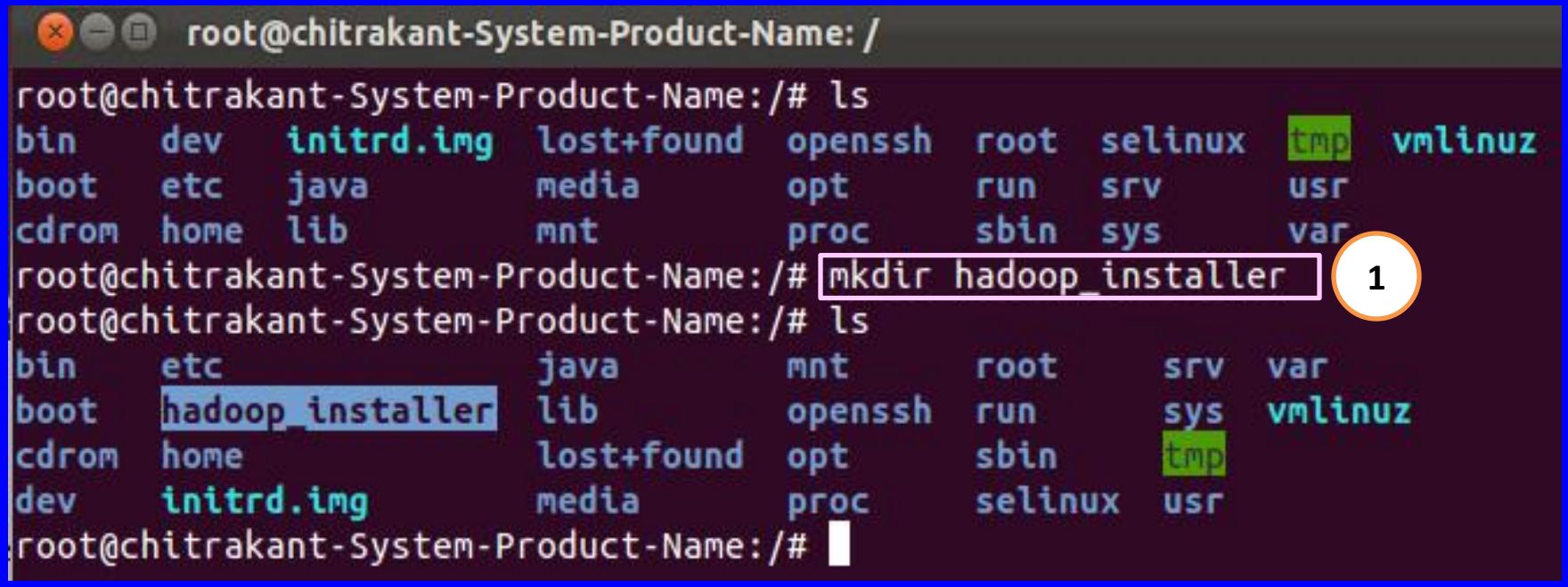
HADOOP installation and configuration

1. Installation

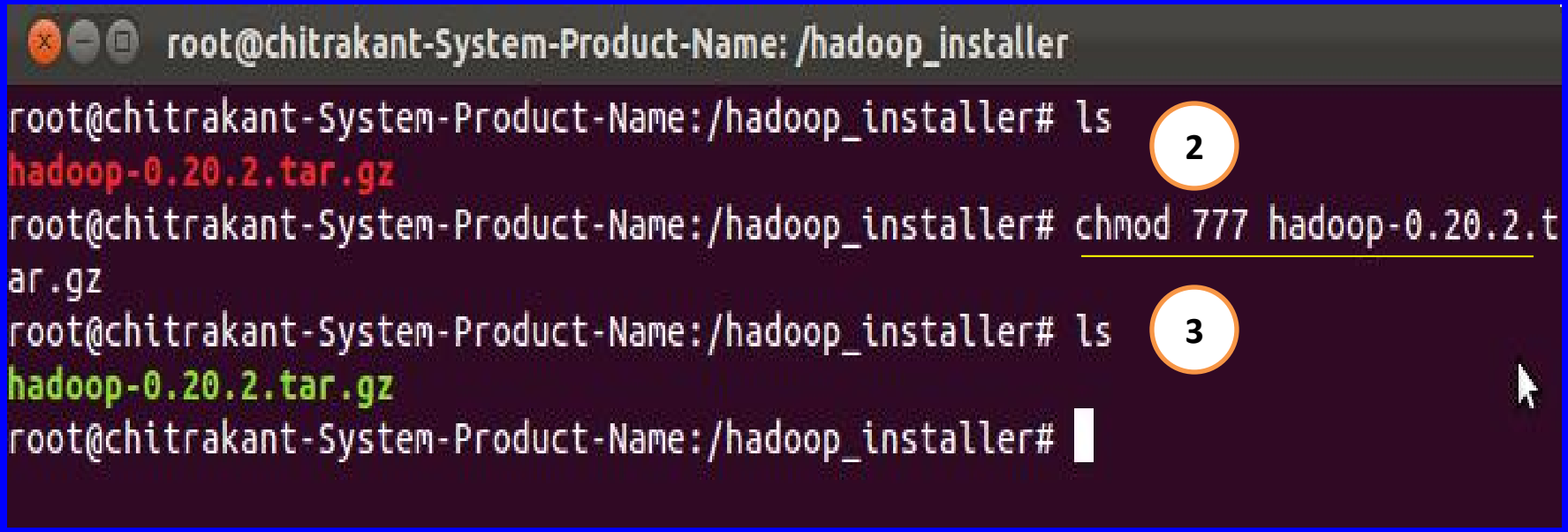
- **Download Hadoop from the Apache Download Mirrors.**



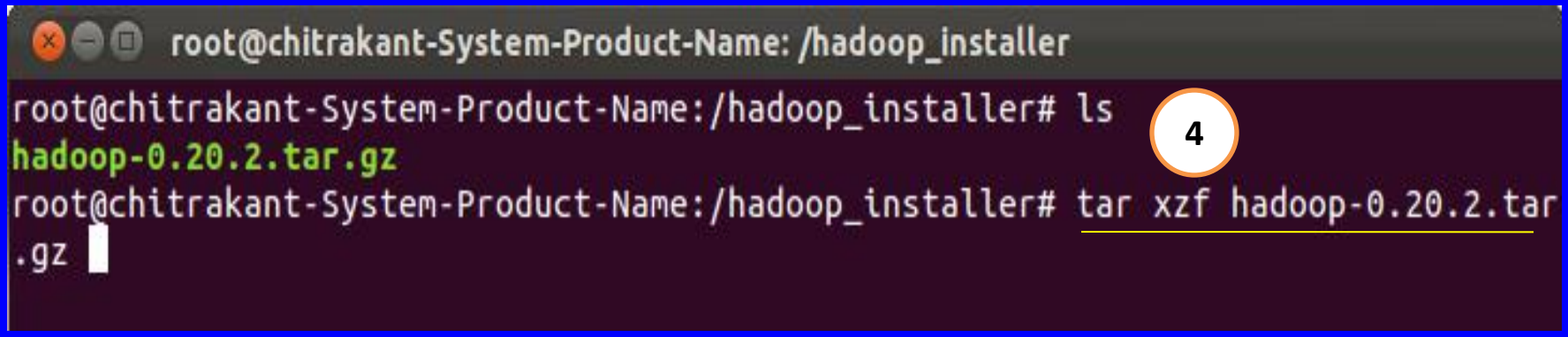
❖ create directory to hold hadoop installer:



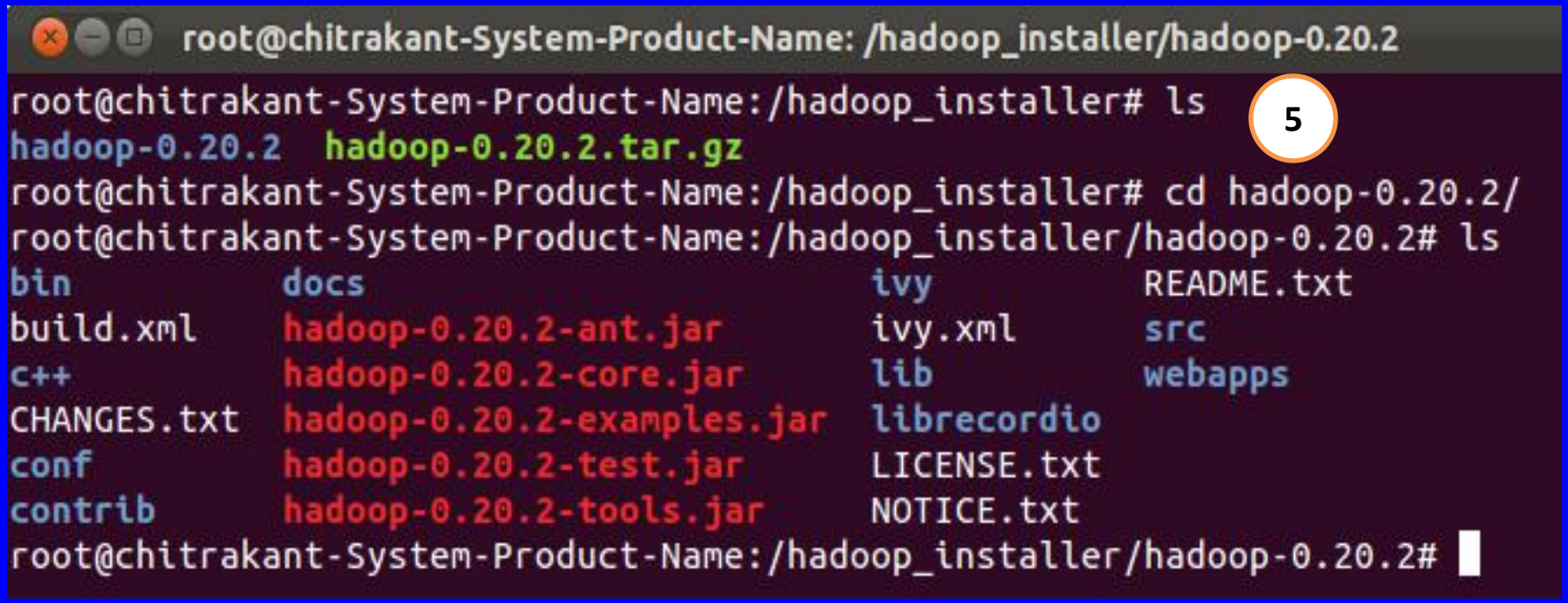
❖ **Copy hadoop installer into the target folder and make it executable:**



❖ Uncompress it using tar command:



❖ Hadoop components in the destination folder:



2. Update $HOME/.bashrc

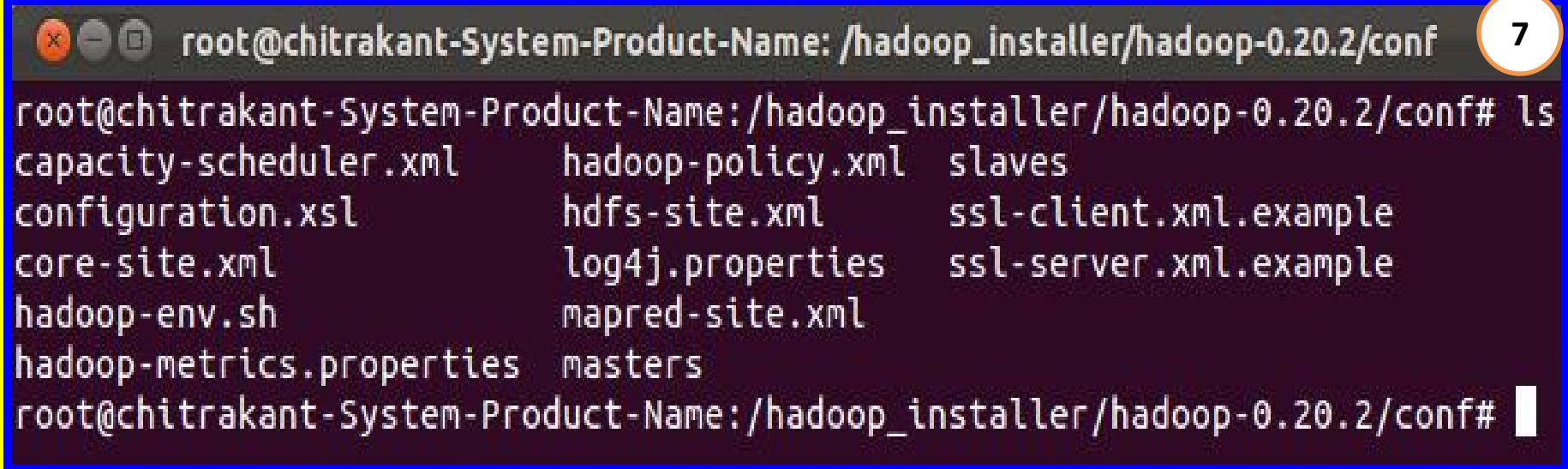
❖ **Set Hadoop environment in .bashrc file:**

```
root@chitrakant-System-Product-Name: ~
  GNU nano 2.2.6                                File: .bashrc


export JAVA_HOME="/java/jdk1.6.0_31"
export PATH=$PATH:$JAVA_HOME/bin

# or use symbolic link
# export JAVA_HOME="/java/jdk_6"

#####################################################################

############### Hadoop environment setting ##########################

export HADOOP_HOME=/hadoop_installer/hadoop-0.20.2

# Add Hadoop bin/ directory to PATH
export PATH=$PATH:$HADOOP_HOME/bin

########
# aliases for running Hadoop-related commands
unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -ls"
#####################################################################
```

6

3. Configuration

- hadoop-env.sh
- conf/*-site.xml



The Hadoop XML Configuration Files

- Hadoop's primary configuration files:

1. Core-site.xml,
2. hdfs-site.xml,
3. mapred-site.xml.

- Hadoop uses a simple XML file format for configuration files

Sample Hadoop XML configuration file

```
<configuration>
        <!-- Set 'some.property.name' to the value 'some-value'.
        <property>
                <name>some.property.name</name>
                <value>some-value</value>
        </property>
<!--Set 'foo.bar.baz' to the value '42' and prevent it from being overridden by marking it final.-->
        <property>
                <name>foo.bar.baz</name>
                <value>42</value>
                <final>true</final>
        </property>
        <!-- Additional property elements... -->
</configuration>
```

- These files control the configuration of the common libraries used by:
  - Hadoop
  - HDFS
  - MapReduce

Example : Hadoop XML configuration file

```
<?xml version="1.0"?>
<!-- core-site.xml -->
<configuration>
        <property>
                <name>fs.default.name</name>
                <value>hdfs://namenode/</value>
                <final>true</final>
        </property>
</configuration>
```

Significance of configuration files

- **Overrides built-in default values:**

- **Properties defined in each of these three files override built in default values which are contained within the main Hadoop jar file.**

hadoop-env.sh

- Environment-specific settings store in this file.
- Configure JAVA_HOME in this file.

```
$ nano /conf/hadoop-env.sh
```

❖ Set the JAVA_HOME environment variable to the Sun JDK/JRE 6 directory

```
# The java implementation to use.
export JAVA_HOME=/jdk1
```

root@chitrakant-System-Product-Name: /hadoop_installer/hadoop-0.20.2/conf

GNU nano 2.2.6 File: hadoop-env.sh

```
# Set Hadoop-specific environment variables here.

# The only required environment variable is JAVA_HOME.  All others are
# optional.  When running a distributed configuration it is best to
# set JAVA_HOME in this file, so that it is correctly defined on
# remote nodes.

# The java implementation to use.  Required.
# export JAVA_HOME=/usr/lib/j2sdk1.5-sun

# Extra Java CLASSPATH elements.  Optional.
# export HADOOP_CLASSPATH=

# The maximum amount of heap to use, in MB. Default is 1000.
# export HADOOP_HEAPSIZE=2000
```

Uncomment and provide java installation path

8

conf/*-site.xml

- **Directory configuration:**

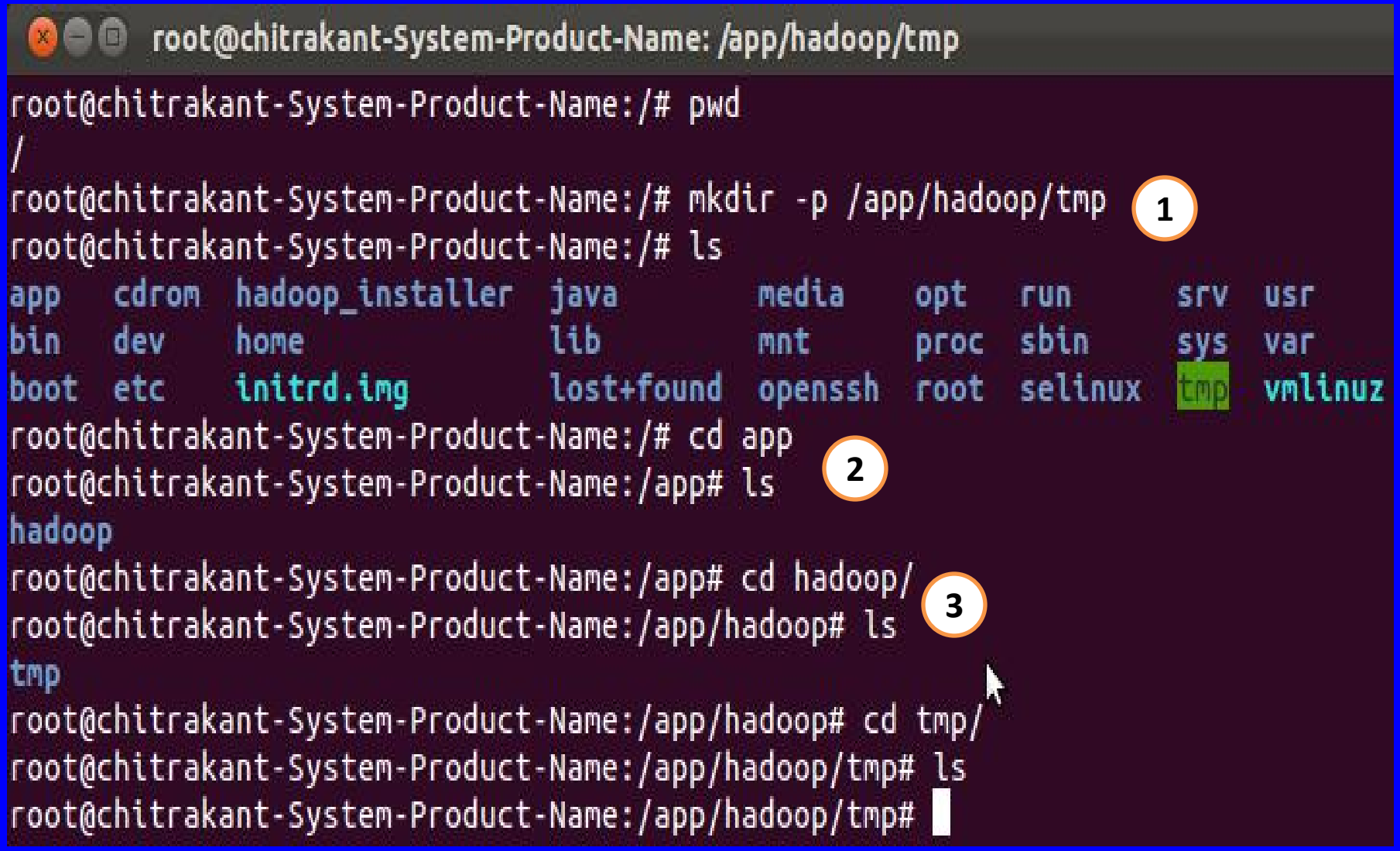
- **Now configure the directory where Hadoop will store its data files.**
- **The network ports it listens to, etc.**

- **HDFS:**

- **This setup will use Hadoop's Distributed File System, HDFS, even it has single node.**

Create directory for Hadoop data

- Here, we use the directory /app/hadoop/tmp in this setup.



core-site.xml

- Contains system-level Hadoop configuration items:
  - The HDFS URL,
  - The Hadoop temporary directory.

❖ Edit the file $HADOOP_HOME/conf/core-site.xml and make sure it looks like the following:

Hadoop created tmp directory we create our own /app/hadoop/tmp

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

        <configuration>
                <property>
                        <name>hadoop.tmp.dir</name>
                        <value>/usr/local/hadoop/tmp</value>
                </property>
                <property>
                        <name>fs.default.name</name>
                        <value>hdfs://localhost:8020</value>
                </property>
        </configuration>
```

❖ The property fs.default.name is a HDFS filesystem URI

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

        <configuration>
                <property>
                        <name>hadoop.tmp.dir</name>
                        <value>/usr/local/hadoop/tmp</value>
                </property>
                <property>
                        <name>fs.default.name</name>
                        <value>hdfs://localhost:8020</value>
                </property>
        </configuration>
```

- Host is the namenode's hostname or IP address
- Port is the port that the namenode will listen on for RPCs.

❖ If no port is specified, the default of 8020 is used

root@chitrakant-System-Product-Name: /hadoop_installer/hadoop-0.20.2/conf

GNU nano 2.2.6 File: core-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
        <property>
                <name>hadoop.tmp.dir</name>
                <value>/app/hadoop/tmp</value>
                <description>A base for other temporary directories.</description>
        </property>
         <property>
                <name>fs.default.name</name>
                <value>hdfs://localhost:54310</value>
                <description>
                        The name of the default file system.
                        A URI whose scheme and authority determine
                        the FileSystem implementation. The uri's
                        scheme determines the config property
                        (fs.SCHEME.impl) naming the FileSystem
                        implementation class. The uri's authority
                        is used to determine the host, port, etc.
                        for a filesystem.
                </description>
         </property>
</configuration>
```

conf/mapred-site.xml

- mapred-site.xml we specify the hostname and port of the JobTracker.

hostname (or IP address)

Port numner on which the jobtracker listens for RPC communication

```
<!-- In: conf/mapred-site.xml -->
<property>
        <name>mapred.job.tracker</name>
        <value>localhost:54311</value>
        <description>
                The host and port that the MapReduce job tracker runs  at.
                If "local", then jobs are run in-process as a single map  and
                reduce task.
        </description>
</property>
```

mapred.job.tracker

- mapred.job.tracker provides the locations of:
  - namenode,
  - datanodes
  - clients
- for tasktrackers and MapReduce clients.

GNU nano 2.2.6 File: mapred-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
        <property>
                <name>mapred.job.tracker</name>
                <value>localhost:54311</value>
                <description>
                    The host and port that the MapReduce job tracker runs
                    at. If "local", then jobs are run in-process as a single map
                    and reduce task.
                </description>
        </property>
</configuration>
```

hdfs-site.xml

- Contains HDFS settings :
  - The default file replication count,
  - The block size,
  - Whether permissions are enforced.

❖ In hdfs-site.xml we specify the default replication factor for HDFS, which should only be one because we're running on only one node.

❖ Edit the file $HADOOP_HOME/conf/hdfs-site.xml and make sure it looks like the following:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
            <name>dfs.replication</name>
            <value>1</value>
   </property>
   <property>
<!-- specify this so that running 'hadoop namenode -format' formats the right dir -->
            <name>dfs.name.dir</name>
            <value>/usr/local/hadoop/cache/hadoop/dfs/name</value>
    </property>
</configuration>
```

GNU nano 2.2.6 File: hdfs-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
                <description>
                 Default block replication.
                 The actual number of replications can be specified when the file is created.
                 The default is used if replication is not specified in create time.
                </description>
        </property>
</configuration>
```

Formatting the HDFS file-system via the NameNode

- **Hadoop File System:**

- Implemented on top of the local file-system of our “cluster”.
- (Single node cluster: on top of one local machine’s file system)
- (Multi node cluster: on top of all machine’s file system)

- **Format the Hadoop File System:**

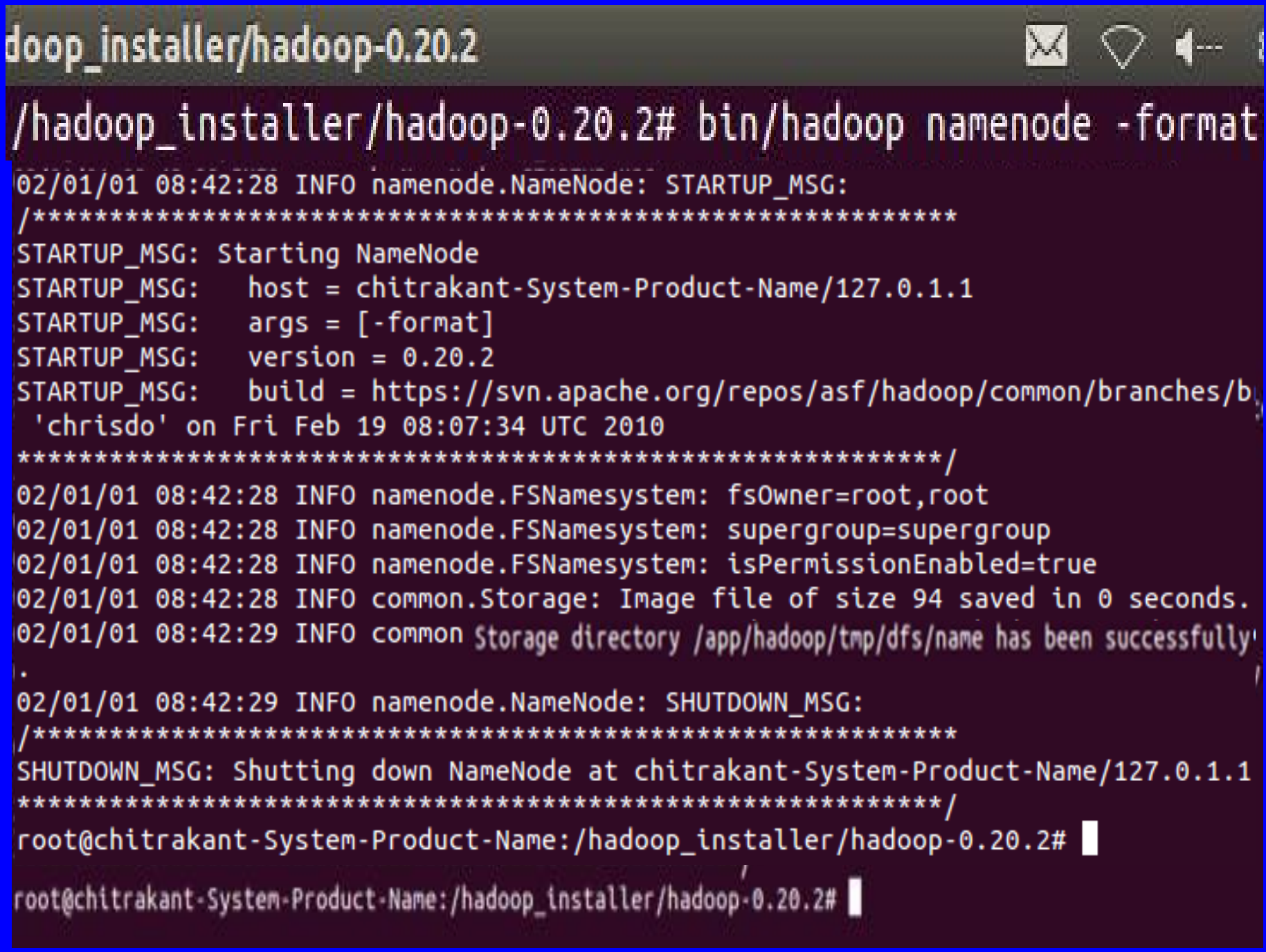
- Format only at first time when set up a Hadoop cluster.

- Format when system is running : data will be lost, not recommended.

- Format the file system: simply initializes the directory specified by dfs.name.dir variable.

```
$  /usr/local/hadoop/bin/hadoop namenode -format
```

Command for format


doop_installer/hadoop-0.20.2
/hadoop_installer/hadoop-0.20.2# bin/hadoop namenode -format
02/01/01 08:42:28 INFO namenode.NameNode: STARTUP_MSG:
/**
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = chitrakant-System-Product-Name/127.0.1.1
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 0.20.2
STARTUP_MSG: build = https://svn.apache.org/repos/asf/hadoop/common/branches/b
'chrisdo' on Fri Feb 19 08:07:34 UTC 2010
**/
02/01/01 08:42:28 INFO namenode.FSNamesystem: fsOwner=root,root
02/01/01 08:42:28 INFO namenode.FSNamesystem: supergroup=supergroup
02/01/01 08:42:28 INFO namenode.FSNamesystem: isPermissionEnabled=true
02/01/01 08:42:28 INFO common.Storage: Image file of size 94 saved in 0 seconds.
02/01/01 08:42:29 INFO common Storage directory /app/hadoop/tmp/dfs/name has been successfully
.
02/01/01 08:42:29 INFO namenode.NameNode: SHUTDOWN_MSG:
/**
SHUTDOWN_MSG: Shutting down NameNode at chitrakant-System-Product-Name/127.0.1.1
**/
root@chitrakant-System-Product-Name:/hadoop_installer/hadoop-0.20.2#
root@chitrakant-System-Product-Name:/hadoop_installer/hadoop-0.20.2#

Starting single-node cluster

- **Run the command:**

```
$ /usr/local/hadoop/bin/start-all.sh
```

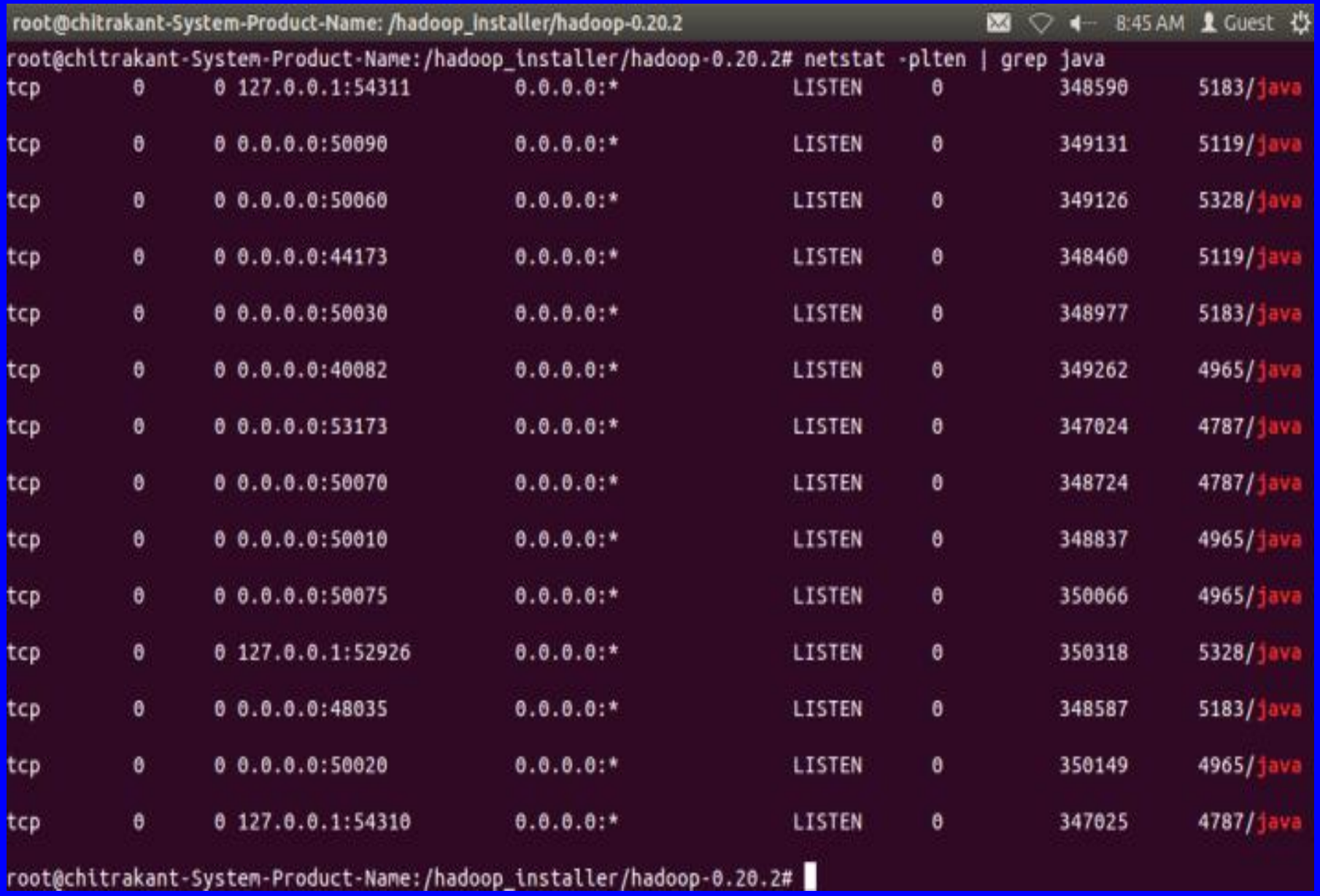
- **This will startup:**

1. Namenode,
2. Datanode,
3. Jobtracker
4. Tasktracker
5. SecondaryNamenode

❖ **Use jps to list out running hadoop processes**

```
root@chitrakant-System-Product-Name:/hadoop_installer/hadoop-0.20.2# jps
4965 DataNode
5328 TaskTracker
5183 JobTracker
5119 SecondaryNameNode
5493 Jps
4787 NameNode
root@chitrakant-System-Product-Name:/hadoop_installer/hadoop-0.20.2#
```

- **Check with netstat if Hadoop is listening on the configured ports.**



Stopping single-node cluster

```
root@chitrakant-System-Product-Name:/hadoop_installer/hadoop-0.20.2# bin/stop-all.sh
stopping jobtracker
localhost: stopping tasktracker
stopping namenode
localhost: stopping datanode
localhost: stopping secondarynamenode
root@chitrakant-System-Product-Name:/hadoop_installer/hadoop-0.20.2#
```

- To stop all the daemons running on the machine.

Step - 4

Step – 5

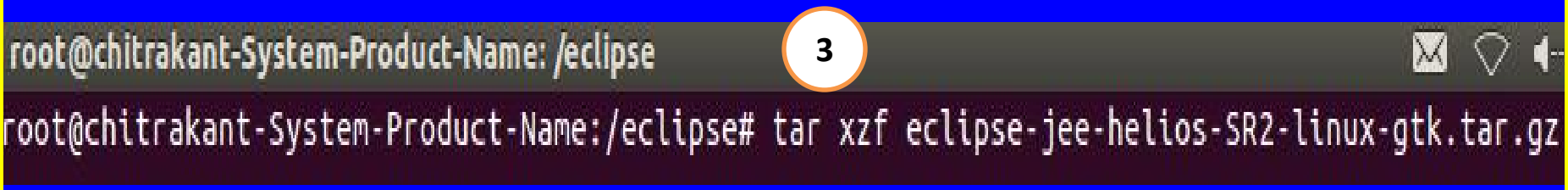
Installing eclipse

❖ Download eclipse installer

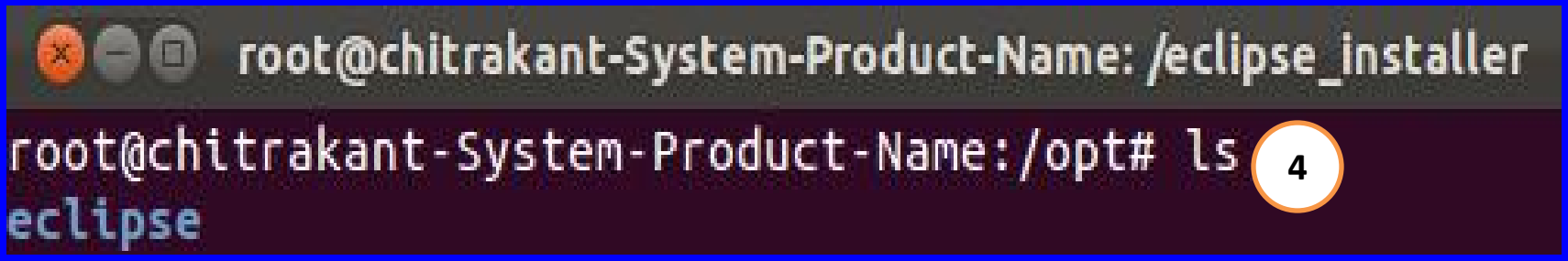

```
root@chitrakant-System-Product-Name: /eclipse
root@chitrakant-System-Product-Name:/# cd eclipse/
root@chitrakant-System-Product-Name:/eclipse# cp eclipse-jee-helios-SR2-linux-gtk.tar.gz .
```

2

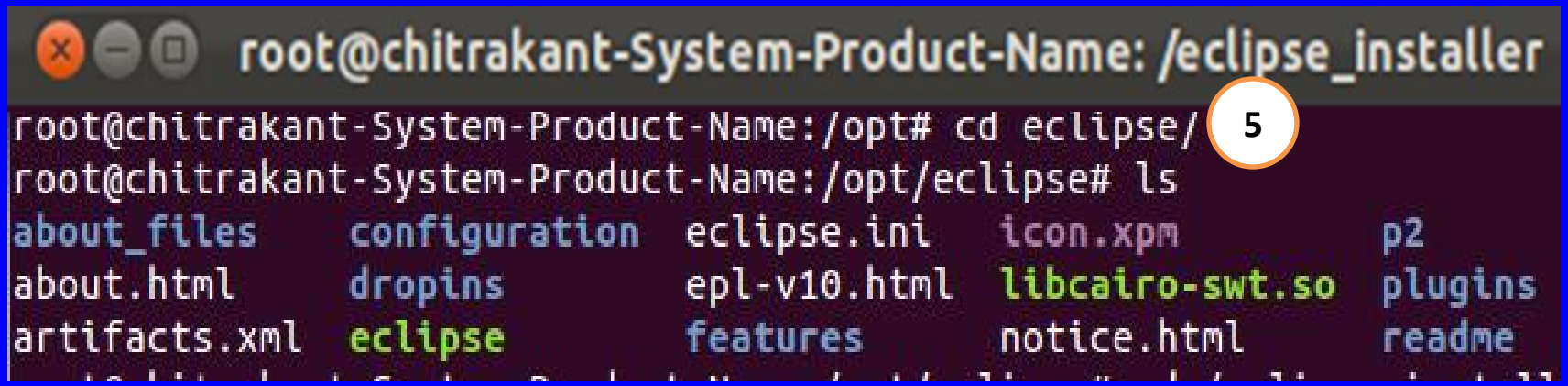
❖ Uncompress the eclipse installer file



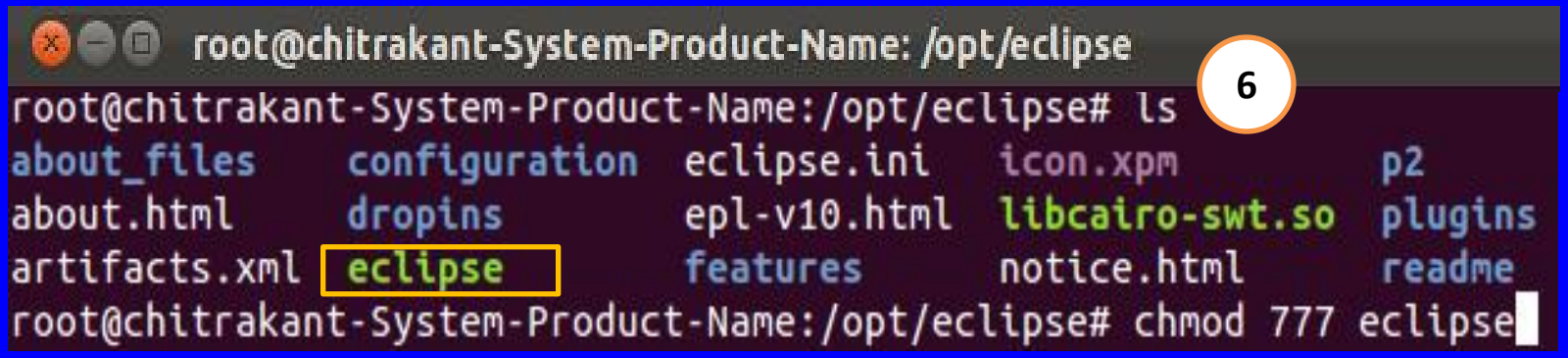
❖ create directory eclipse in /opt directory



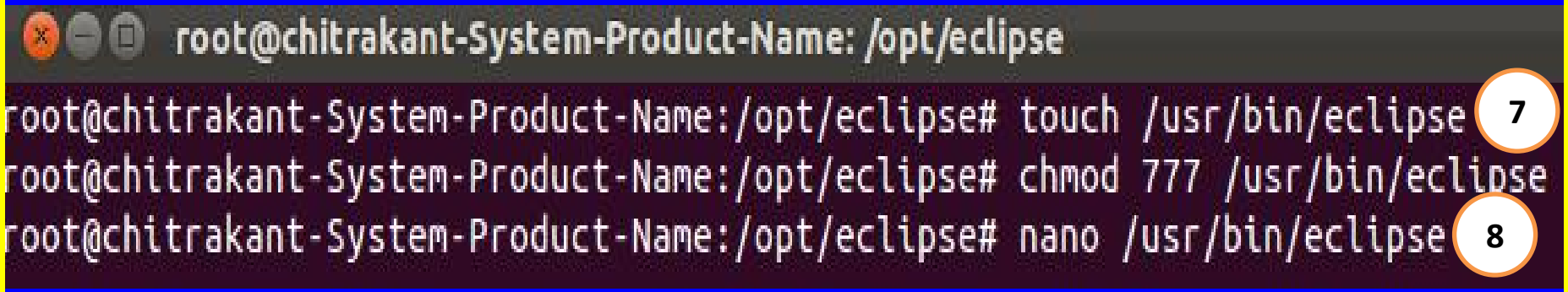
❖ move uncompressed eclipse into eclipse directory of /opt directory



❖ Provide execution permission to the eclipse executable

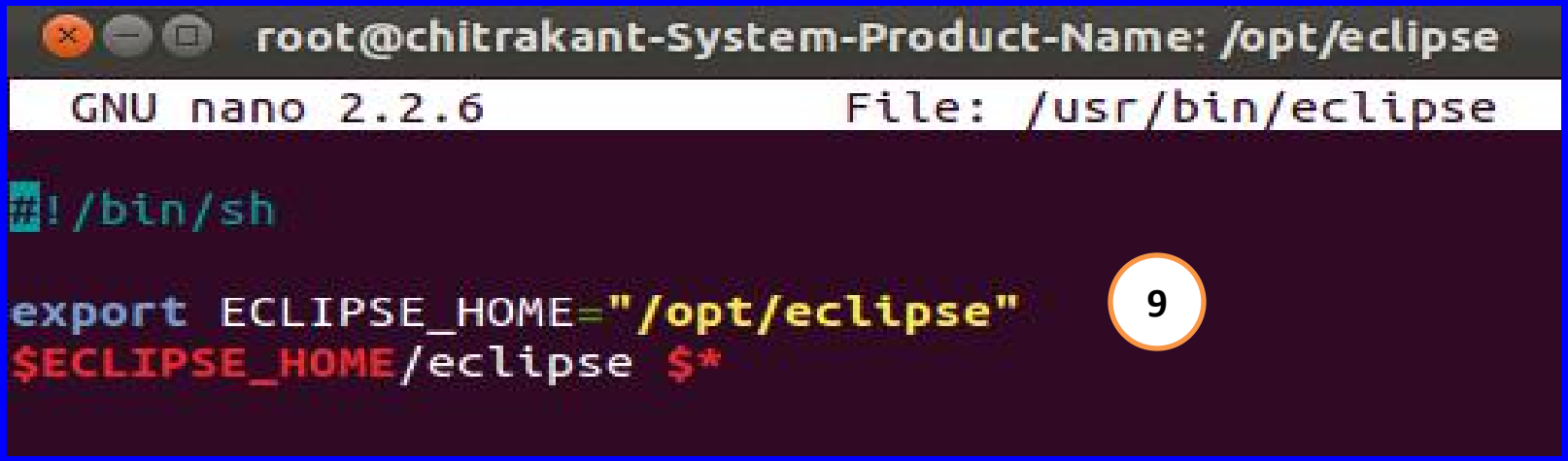


❖ Create an executable in home path:

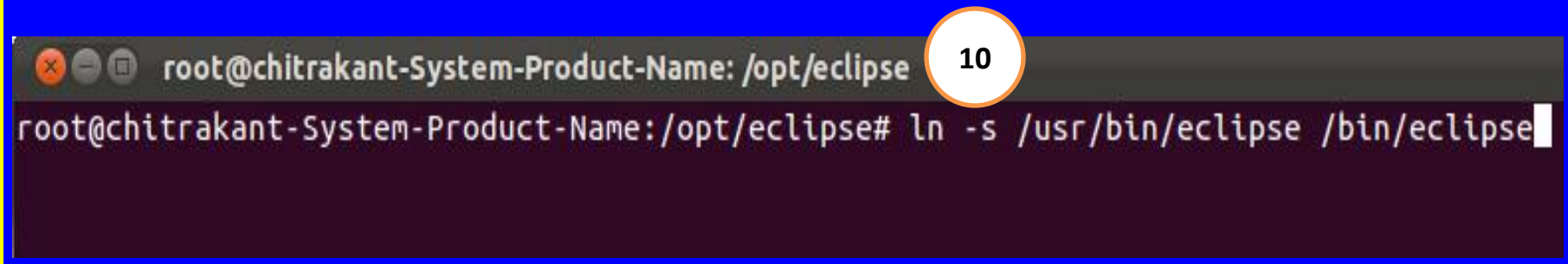


- **Write following steps in the eclipse file and exit from the editor**

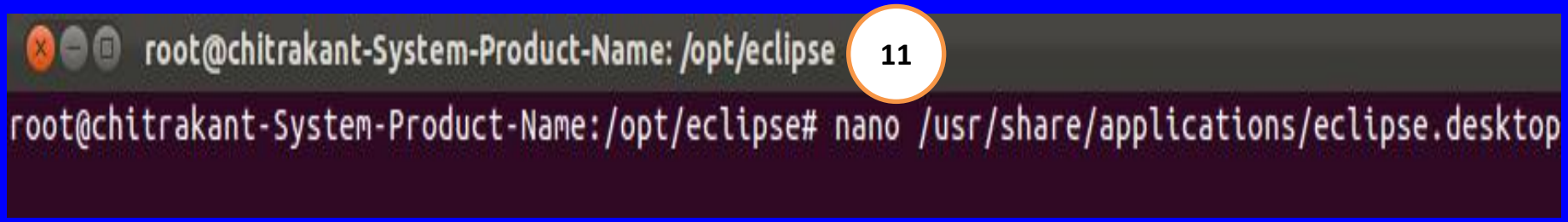
```
#!/bin/sh

ECLIPSE_HOME="/opt/eclipse" $ECLIPSE_HOME/eclipse   $*
```


root@chitrakant-System-Product-Name: /opt/eclipse
GNU nano 2.2.6 File: /usr/bin/eclipse
#!/bin/sh
export ECLIPSE_HOME="/opt/eclipse"
$ECLIPSE_HOME/eclipse $*
9

❖ Make eclipse executable everywhere by creating a symlink

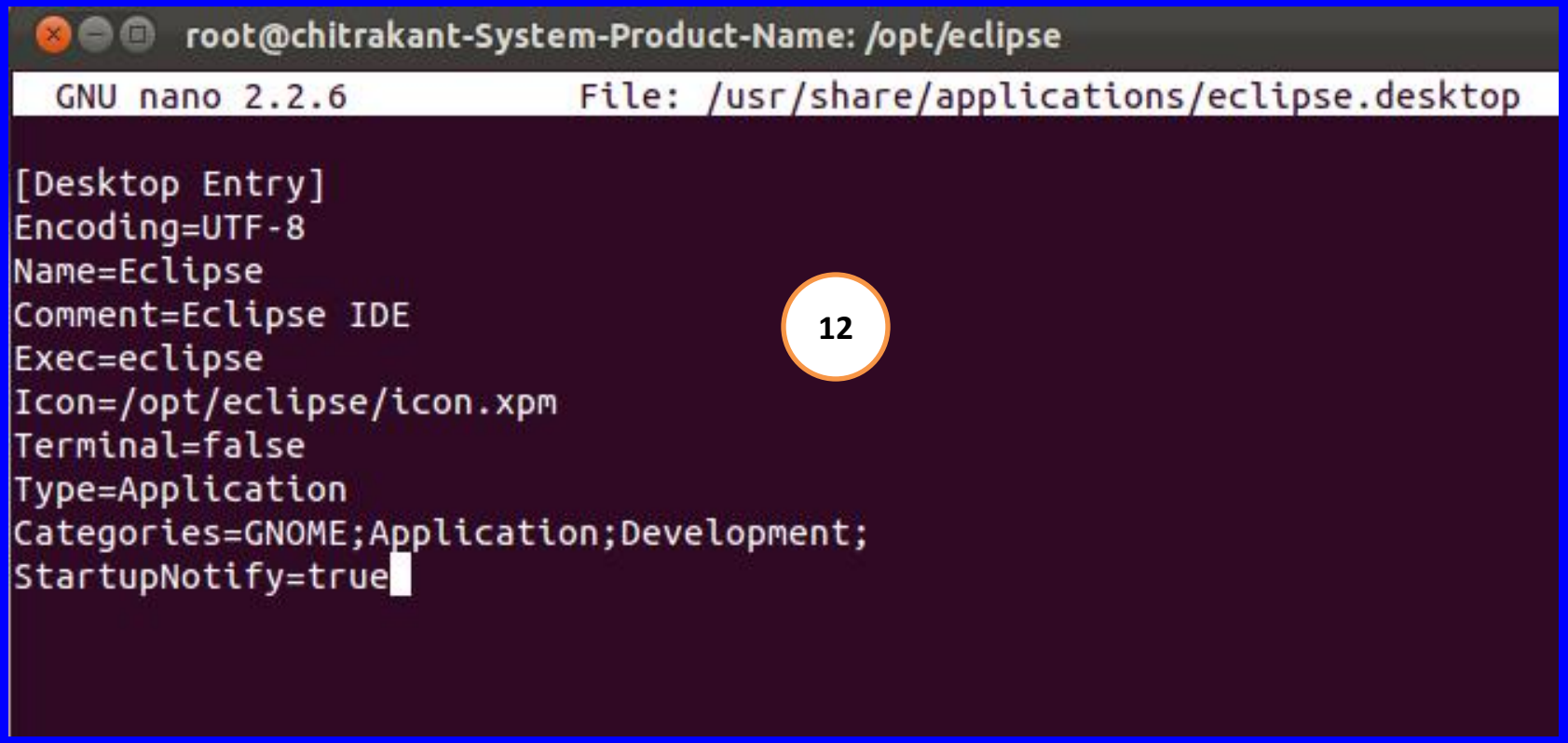


❖ create menu icon on the launcher

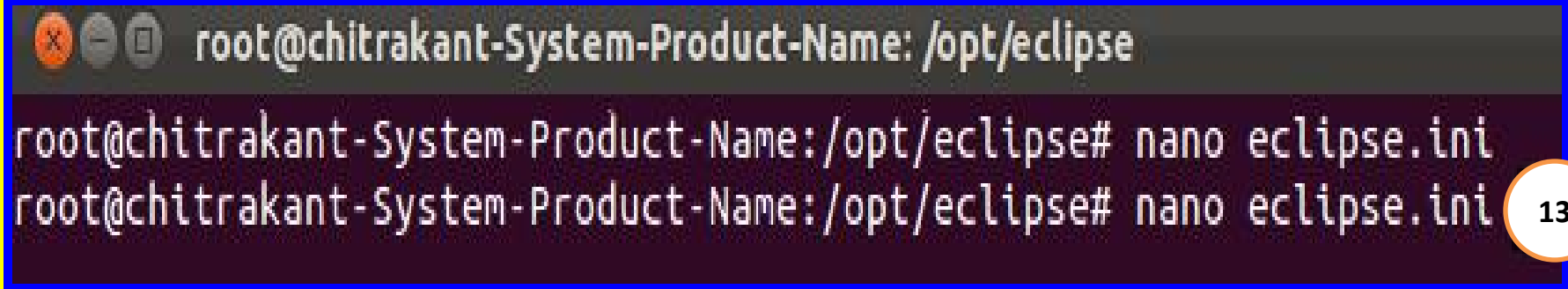


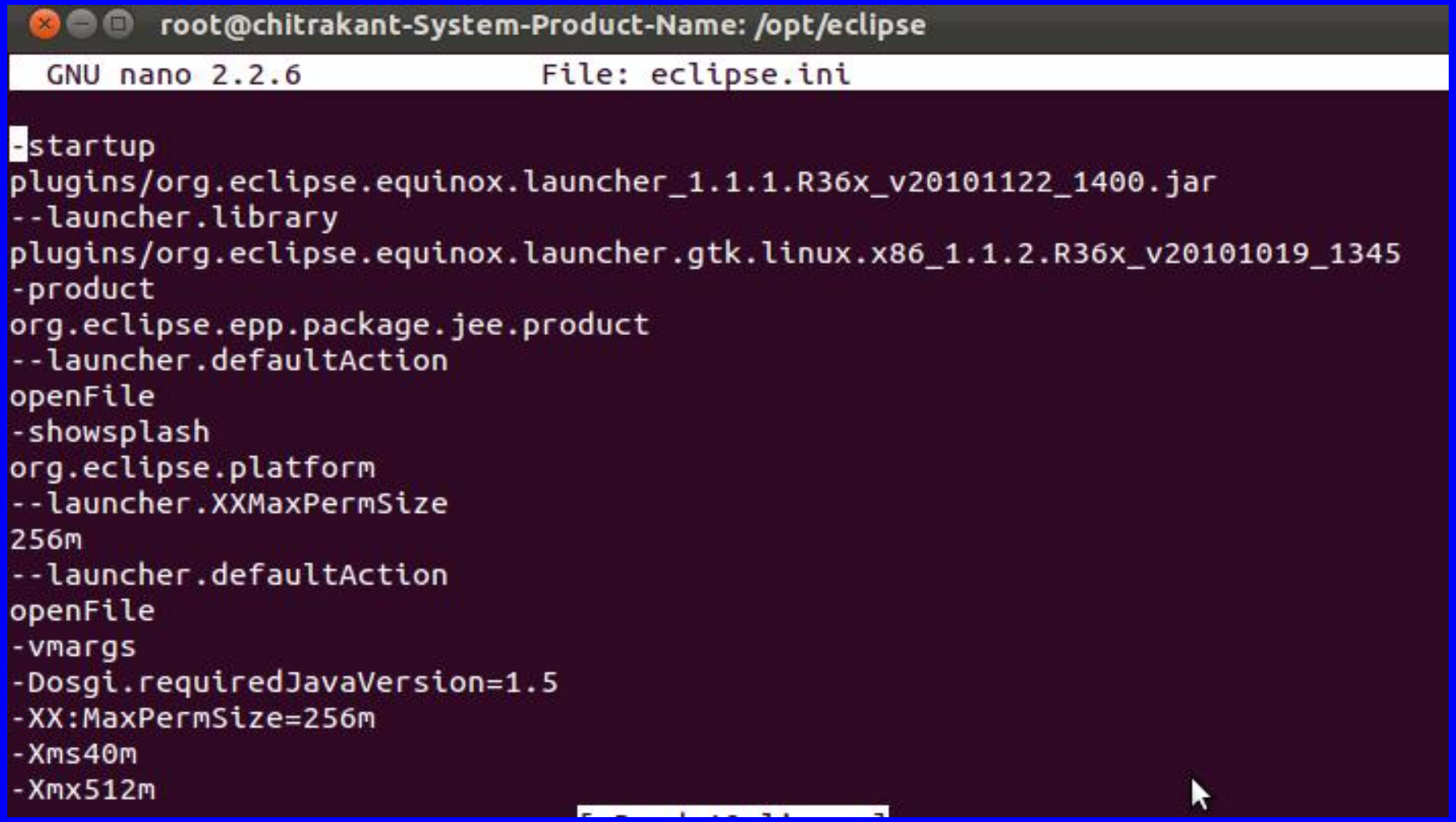
- **Write following contents into eclipse.Desktop file and save**

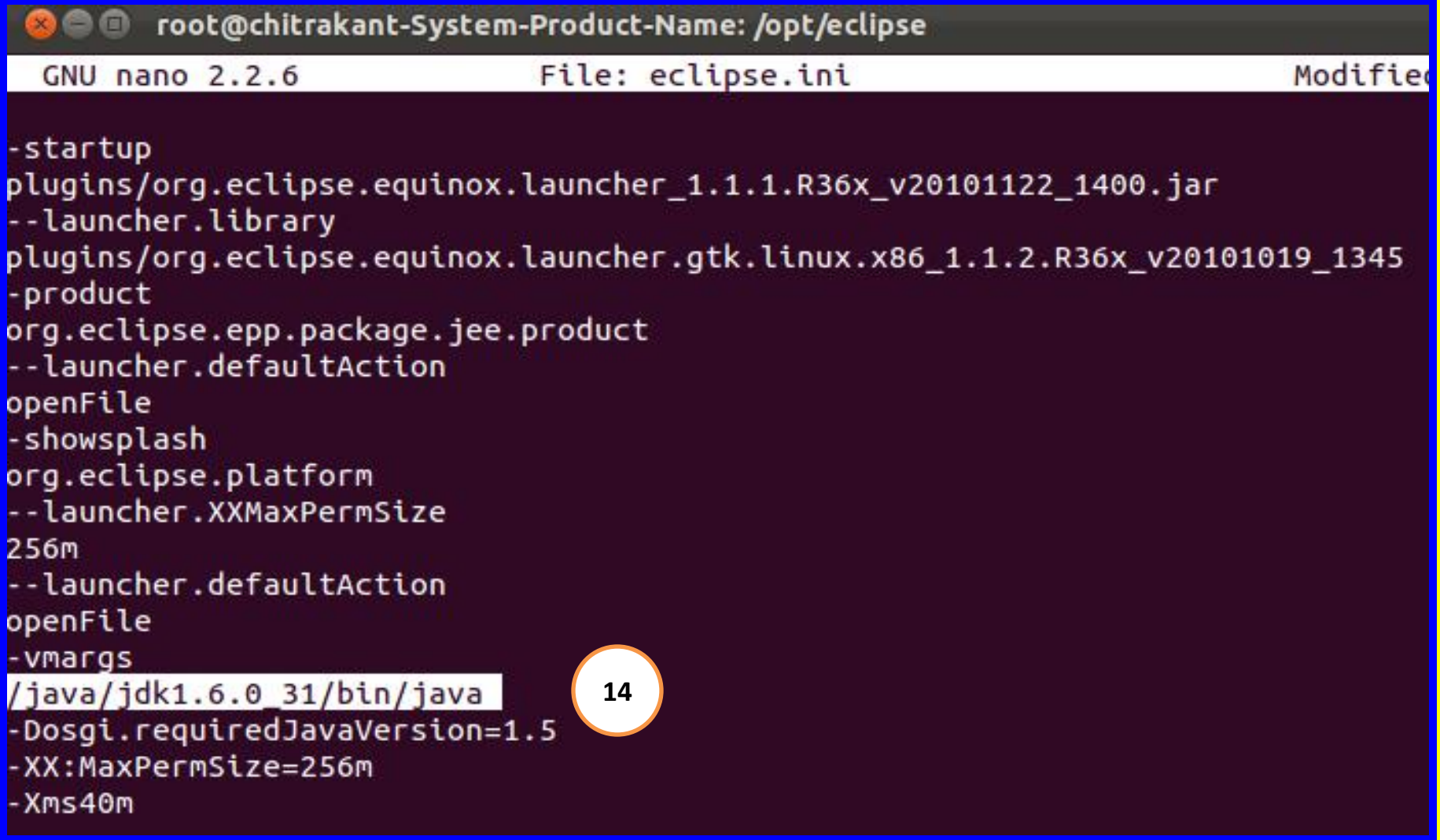
```
[Desktop Entry]
Encoding=UTF-8
Name=Eclipse
Comment=Eclipse IDE
Exec=eclipse
Icon=/opt/eclipse/icon.xpm
Terminal=false
Type=Application
Categories=GNOME;Application;Development;
StartupNotify=true
```


root@chitrakant-System-Product-Name: /opt/eclipse
GNU nano 2.2.6 File: /usr/share/applications/eclipse.desktop
[Desktop Entry]
Encoding=UTF-8
Name=Eclipse
Comment=Eclipse IDE
Exec=eclipse
Icon=/opt/eclipse/icon.xpm
Terminal=false
Type=Application
Categories=GNOME;Application;Development;
StartupNotify=true
12

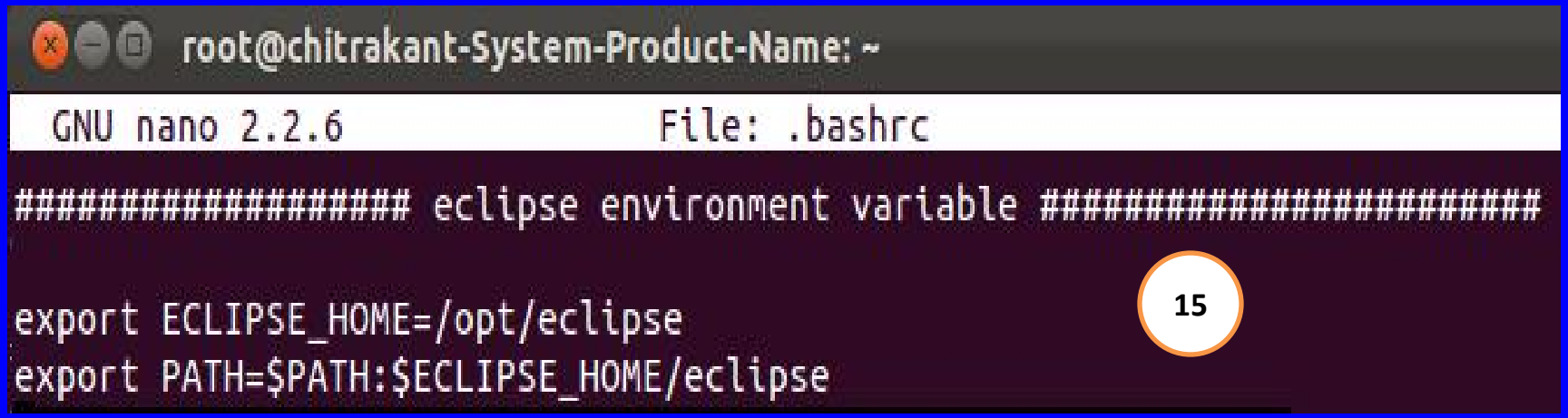
❖ Specifying the JVM




root@chitrakant-System-Product-Name: /opt/eclipse
GNU nano 2.2.6 File: eclipse.ini
-startup
plugins/org.eclipse.equinox.launcher_1.1.1.R36x_v20101122_1400.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.gtk.linux.x86_1.1.2.R36x_v20101019_1345
-product
org.eclipse.epp.package.jee.product
--launcher.defaultAction
openFile
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
--launcher.defaultAction
openFile
-vmargs
-Dosgi.requiredJavaVersion=1.5
-XX:MaxPermSize=256m
-Xms40m
-Xmx512m


root@chitrakant-System-Product-Name: /opt/eclipse
GNU nano 2.2.6 File: eclipse.ini Modifie
-startup
plugins/org.eclipse.equinox.launcher_1.1.1.R36x_v20101122_1400.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.gtk.linux.x86_1.1.2.R36x_v20101019_1345
-product
org.eclipse.epp.package.jee.product
--launcher.defaultAction
openFile
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
--launcher.defaultAction
openFile
-vmargs
/java/jdk1.6.0_31/bin/java
-Dosgi.requiredJavaVersion=1.5
-XX:MaxPermSize=256m
-Xms40m
14

❖ Set eclipse environment variables in the .bashrc file

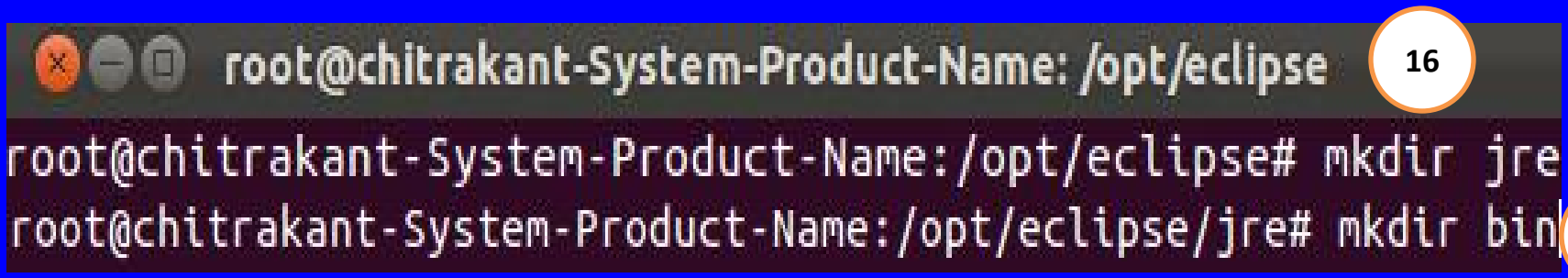


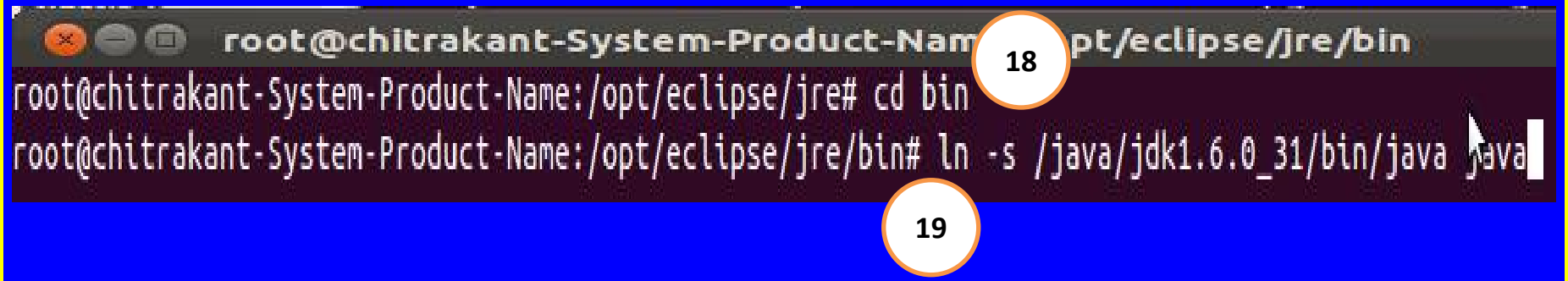
❖ Set eclipse launcher on the panel

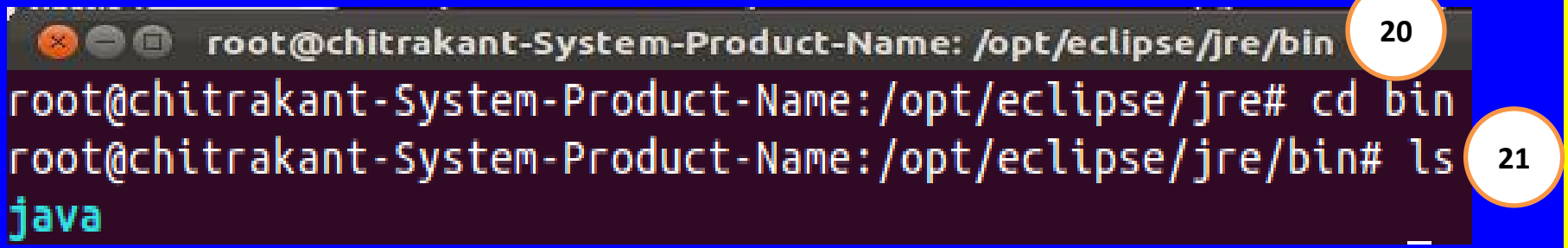
Step 1 : make directory jre inside /opt/eclipse directory

Step 2 : make directory bin inside /opt/eclipse/jre directory

Step 3 : create symbolic link for the /jdk/java/bin/java inside /opt/eclipse/jre/bin


root@chitrakant-System-Product-Name: /opt/eclipse
16
root@chitrakant-System-Product-Name:/opt/eclipse# mkdir jre
root@chitrakant-System-Product-Name:/opt/eclipse/jre# mkdir bin
17


root@chitrakant-System-Product-Nam pt/eclipse/jre/bin
18
root@chitrakant-System-Product-Name:/opt/eclipse/jre# cd bin
root@chitrakant-System-Product-Name:/opt/eclipse/jre/bin# ln -s /java/jdk1.6.0_31/bin/java java
19


root@chitrakant-System-Product-Name: /opt/eclipse/jre/bin
20
root@chitrakant-System-Product-Name:/opt/eclipse/jre# cd bin
root@chitrakant-System-Product-Name:/opt/eclipse/jre/bin# ls
21
java


Eclipse
Required_
Softwares
Tutorials
eclipse
HELIOS
(c) Copyright Eclipse contributors and others, 2000, 2011. All rights reserved. Eclipse is a trademark of the Eclipse Foundation, Inc. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.
21

Step - 5

```
root@chitrakant-System-Product-Name:/hadoop_installer/hadoop-0.20.2# bin/hadoop dfs -put conf input
root@chitrakant-System-Product-Name:/hadoop_installer/hadoop-0.20.2# bin/hadoop jar hadoop-*-exampl
es.jar grep input output 'dfs[a-z.]+'
02/01/01 00:26:18 INFO mapred.FileInputFormat: Total input paths to process : 13
02/01/01 00:26:18 INFO mapred.JobClient: Running job: job_200201010003_0001
02/01/01 00:26:19 INFO mapred.JobClient:  map 0% reduce 0%
02/01/01 00:26:30 INFO mapred.JobClient:  map 15% reduce 0%
02/01/01 00:26:36 INFO mapred.JobClient:  map 30% reduce 0%
02/01/01 00:26:39 INFO mapred.JobClient:  map 46% reduce 10%
02/01/01 00:26:42 INFO mapred.JobClient:  map 61% reduce 10%
02/01/01 00:26:45 INFO mapred.JobClient:  map 76% reduce 10%
02/01/01 00:26:48 INFO mapred.JobClient:  map 92% reduce 15%
02/01/01 00:26:51 INFO mapred.JobClient:  map 100% reduce 15%
02/01/01 00:26:57 INFO mapred.JobClient:  map 100% reduce 100%
02/01/01 00:26:59 INFO mapred.JobClient: Job complete: job_200201010003_0001
02/01/01 00:26:59 INFO mapred.JobClient: Counters: 18
02/01/01 00:26:59 INFO mapred.JobClient:   Job Counters
02/01/01 00:26:59 INFO mapred.JobClient:     Launched reduce tasks=1
02/01/01 00:26:59 INFO mapred.JobClient:     Launched map tasks=13
02/01/01 00:26:59 INFO mapred.JobClient:     Data-local map tasks=13
02/01/01 00:26:59 INFO mapred.JobClient:   FileSystemCounters
02/01/01 00:26:59 INFO mapred.JobClient:     FILE_BYTES_READ=158
02/01/01 00:26:59 INFO mapred.JobClient:     HDFS_BYTES_READ=19146
02/01/01 00:26:59 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=804
02/01/01 00:26:59 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=280
02/01/01 00:26:59 INFO mapred.JobClient:   Map-Reduce Framework
02/01/01 00:26:59 INFO mapred.JobClient:     Reduce input groups=7
02/01/01 00:26:59 INFO mapred.JobClient:     Combine output records=7
02/01/01 00:26:59 INFO mapred.JobClient:     Map input records=580
02/01/01 00:26:59 INFO mapred.JobClient:     Reduce shuffle bytes=230
02/01/01 00:26:59 INFO mapred.JobClient:     Reduce output records=7
02/01/01 00:26:59 INFO mapred.JobClient:     Spilled Records=14
02/01/01 00:26:59 INFO mapred.JobClient:     Map output bytes=193
02/01/01 00:26:59 INFO mapred.JobClient:     Map input bytes=19146
02/01/01 00:26:59 INFO mapred.JobClient:     Combine input records=10
```

Step - III

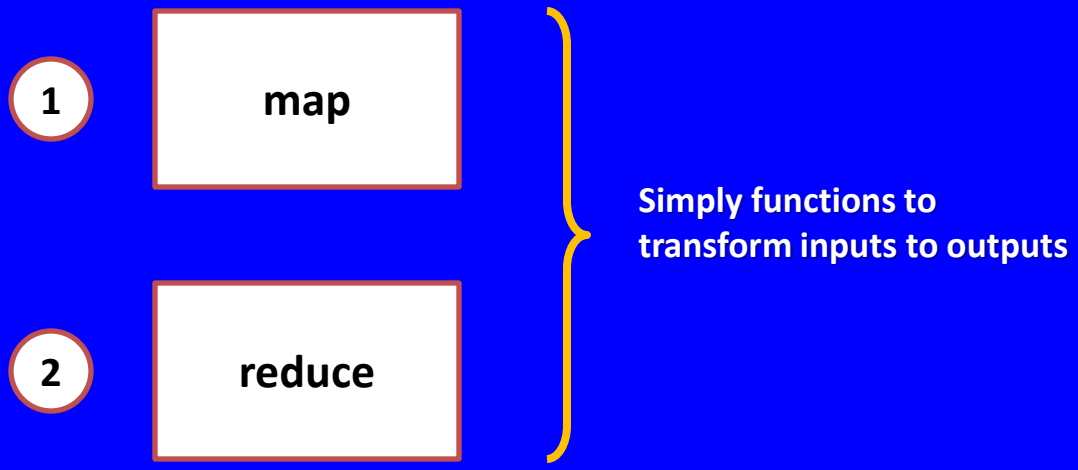
Hadoop : Fourth step

MapReduce Programming

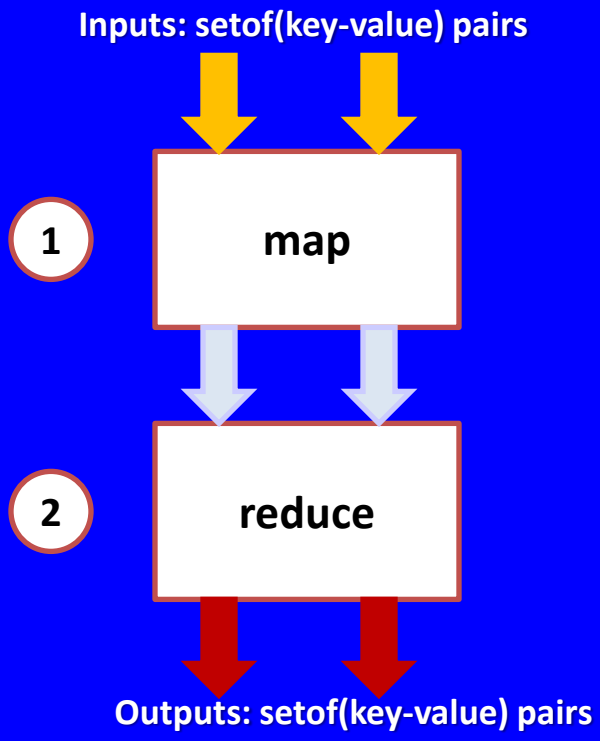
Understanding MapReduce Program

MapReduce

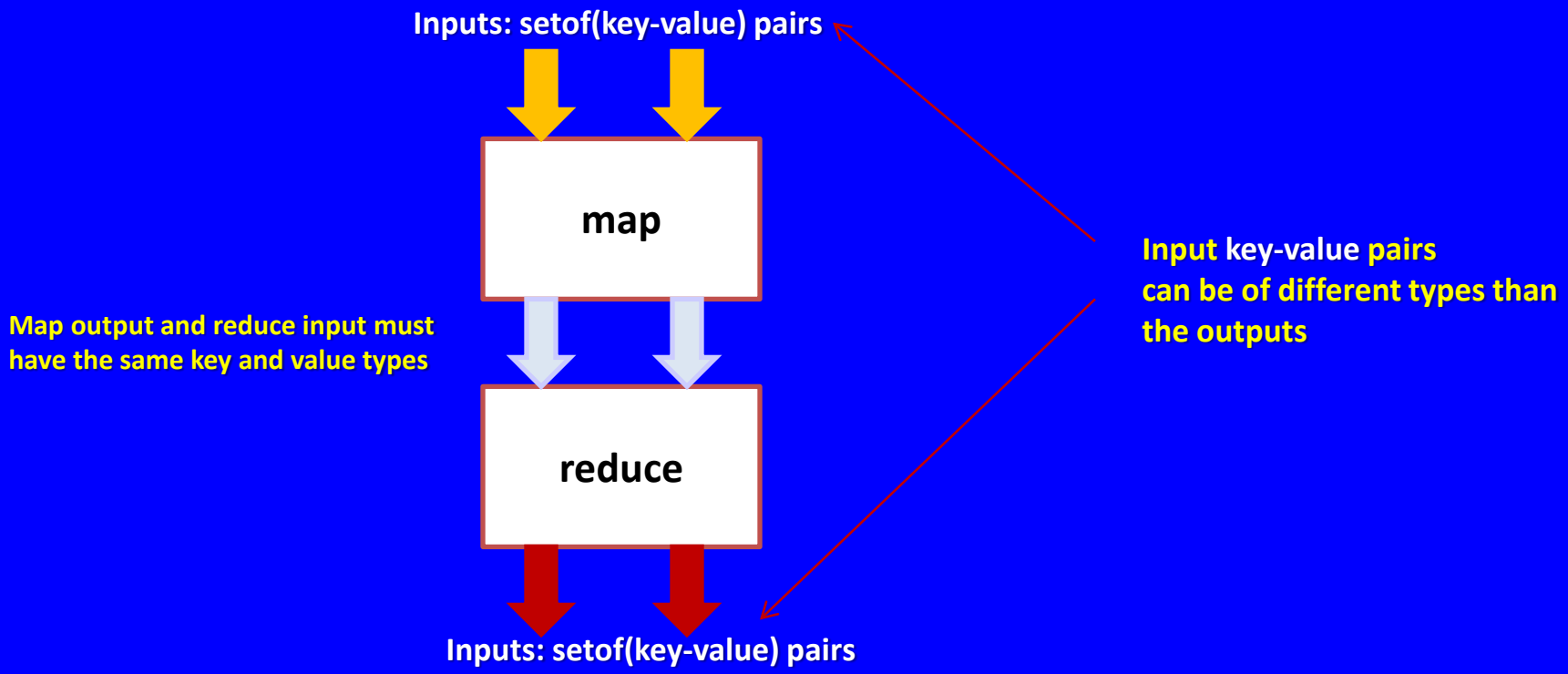
- A MapReduce job proceeds in two phases



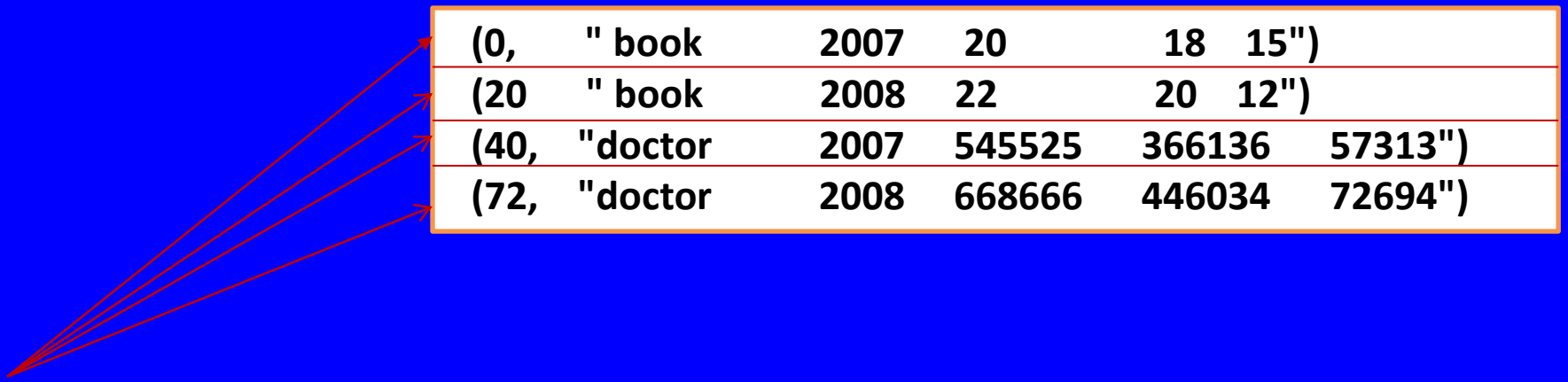
- **Inputs and outputs are key – value pairs**



- **MapReduce is typed**



Sample Input File



- Input comprises lines of text
- Represented as key-value pairs in Hadoop

Key	Value
✓ The offset of the line within the file	❖ The value is the line of text

- Ignore other fields and the key from input file.
- Only word and the number of occurrences required.

(0,	" book	2007	20	18	15")
(20	" book	2008	22	20	12")
(40,	"doctor	2007	545525	366136	57313")
(72,	"doctor	2008	668666	446034	72694")

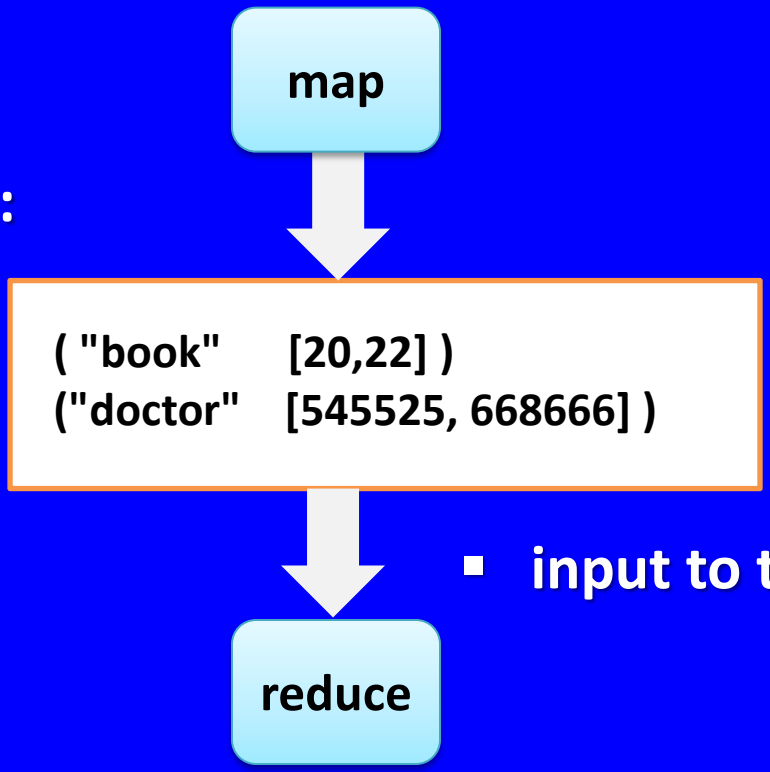
- The map emits →

Key: word Value : No. of occurrences

("book"	20)
(" book"	22)
("doctor"	545525)
("doctor"	668666)

- **Map does** → The values for a given key are brought together so that the reduce function processes them as a group

- The map output:



- input to the reduce function

Reduce does → **The reduce function iterates over the values for each key and does whatever processing it likes.**

- **This example** → **An aggregate count (simply sum the values)**

```
( "book"     [42] )
( "doctor"   [1214191] )
```

- **Final result** → **The number of times each word occurred in the whole dataset**

Java MapReduce

- Write the MapReduce job to find the total count for each word.

map function

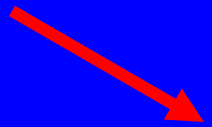
- Represented in Java by an instance of
  - ✓ org.apache.hadoop.mapreduce.Mapper

- Decision about input key-value pairs and output key-value pairs types

```
public class Mapper< KEYIN,  VALUEIN,  KEYOUT,  VALUEOUT >
```

Declaration of the Mapper class

- **TextInputFormat** → **Because we are processing text**



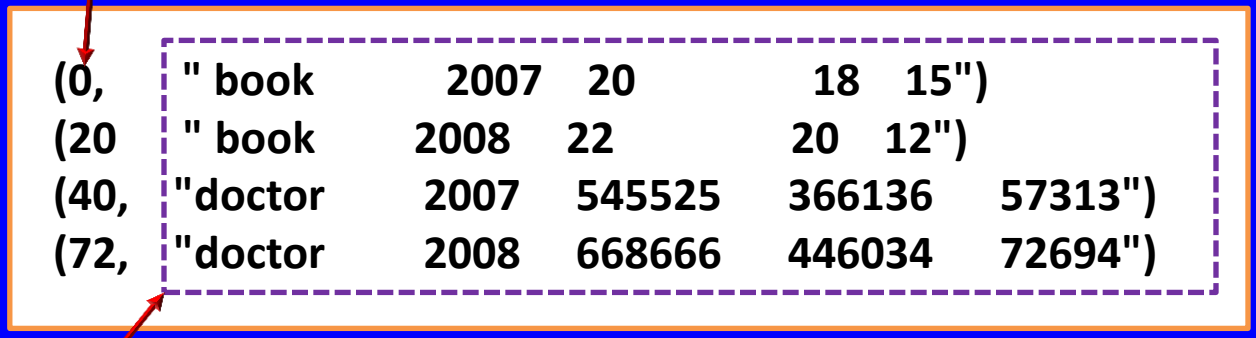
Determines the input types as LongWritable and Text

(in package: org.apache.hadoop.io)

- **Wrappers around standard Java types (long and String)**

- TextInputFormat presents the input to our mapper as (LongWritable, Text) pairs

Key → The offset within the file



value → The content of the line

Mapper

- Extract the word and the number of occurrences, and ignore everything else

- Mapper output is (word, count) pairs, of type (Text, LongWritable)

```
public class CountMapper extends Mapper<LongWritable, Text, Text, LongWritable>
```

- Our task → Write the implementation of the map() method

```
package myPackage;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class CountMapper extends
        Mapper<LongWritable, Text, Text, LongWritable> {

}
```

Import Library Packages

Every Map class must extend Mapper

```
package myPackage;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class CountMapper extends
        Mapper<LongWritable, Text, Text, LongWritable> {

    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {



    }
}
```

Import Library Packages

Every Map class must extend Mapper

Every Mapper class must provide definition of map function

```
package myPackage;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class CountMapper extends
                    Mapper<LongWritable, Text, Text, LongWritable> {

  private Text word = new Text();
  private LongWritable count = new LongWritable();

    protected void map(LongWritable key, Text value, Context context)
                                        throws IOException, InterruptedException {




    }

}
```

Instance variables
To store map output key and value

```
(0,    " book     2007   20      18      15")
(20    " book     2008   22      20      12")
(40,   "doctor    2007   545525  366136  57313")
(72,   "doctor    2008   668666  446034  72694")
 [0]    [1]       [2]    [3]     [4]     [5]
```

```
package myPackage;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class CountMapper extends
                    Mapper<LongWritable, Text, Text, LongWritable> {

   private Text word = new Text();
   private LongWritable count = new LongWritable();

   protected void map(LongWritable key, Text value, Context context)
                                        throws IOException, InterruptedException {

        /* Body of map()→ splits the tab – separated input line into fields
           and uses the first field as the word and the third as the count

    }

}
```

```
package myPackage;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class CountMapper extends
                    Mapper<LongWritable, Text, Text, LongWritable> {

  private Text word = new Text();
  private LongWritable count = new LongWritable();

  protected void map(LongWritable key, Text value, Context context)
                                        throws IOException, InterruptedException {

      context.write(word, count);
      /*The map output is written using the write method in Context*/

  }

}
```

```
public class CountMapper extends
                        Mapper<LongWritable, Text, Text, LongWritable> {
  private Text word = new Text();
  private LongWritable count = new LongWritable();

  protected void map(LongWritable key, Text value, Context context)
                                                  throws IOException, InterruptedException {
    /* value is tab separated values: word, year, occurrences, #books, #pages
    we project out (word, occurrences) so we can sum over all years  */
    String[] split = value.toString().split("\t+");
        word.set(split[0]);
        if (split.length > 2) {
            try {
                count.set(Long.parseLong(split[2]));
                context.write(word, count);
            } catch (NumberFormatException e) {
                // cannot parse - ignore
              }
        }//if
  }
}
```

- **Running through our dataset, the map output looks like this:**

```
( "book"     20 )
(" book"      22)
("doctor"     545525)
("doctor"      668666)
```

Reduce

("book" [20,22])
("doctor" [545525, 668666])

```
package myPackage;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class CountReducer extends
        Reducer<Text, LongWritable, Text,LongWritable> {

    protected void reduce(KEY key, Iterable<LongWritable>values,
            Context context) throws IOException, InterruptedException {



    }
}
```

Import Library Packages

Every Reducer class must extend Reducer

Every Reducer class must provide definition of reduce function

- **Notice that the reduce() method signature is different from map()**
- **It has an iterator over the values, rather than a single value.**
- **This reflects the grouping that the framework performs on the values for a key.**

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

      protected void reduce(KEY key, Iterable<LongWritable>values,
                        Context context) throws IOException, InterruptedException {









      }
}
```

```
public class CountReducer<KEY> extends Reducer<KEY, LongWritable,
                                               KEY,LongWritable> {

  private LongWritable result = new LongWritable();

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

    long sum = 0;

    for (LongWritable val : values) {
        sum += val.get();
    }

    result.set(sum);

    context.write(key, result);

  }
}
```

It sums the values, then writes the total out using the same key as the input

- **The output of the reducer will be:**

```
( "book"      [42] )
("doctor"      [1214191])
```

Code the MapReduce Application/Driver class (CountJob.java)

- Define a driver class.
- Create a new client job, configuration object and advertise Mapper and Reducer classes.
- The driver class is responsible for setting our MapReduce job to run in Hadoop.
- We specify job name, data type of input/output and names of mapper and reducer classes.

```
package myPackage;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
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;
import org.apache.hadoop.mapreduce.lib.reduce.LongSumReducer;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

- CountJob extends Hadoop's Configured class and implements the Tool interface's run()method
- This allows us to pass in configuration options at runtime

```
public class CountJob extends Configured implements Tool {

        public int run(String[] args) throws Exception {


        }


}
```

- The code to launch a job is in the run() method.
- It is managed by the Job class from the org.apache.hadoop.mapreduce package.

```
public class CountJob extends Configured implements Tool {

    public static void main(String[] args) throws Exception {

    }
    public int run(String[] args) throws Exception {

    }

}
```

1

2

```
public class CountJob extends Configured implements Tool {


        public static void main(String[] args) throws Exception {
                int   rc   =   ToolRunner . run( new  CountJob(), args );
                System.exit(rc);
        }
}
```

```
public class CountJob extends Configured implements Tool {

    public int run(String[] args) throws Exception {

        /* The code to launch a job is in the run() method */
        /* it is managed by the Job class from the org.apache.hadoop.mapreduce package. */

        /* The Job instance specifies various things about the job:
               A name for display purposes,
               The mapper and reducer classes
               The job output types, (Have to match the mapper and reducer output types)
        */

        setMapOutputKeyClass()
        setMapOutputValueClass()

    }
}
```

1

2

3

Should call when map output types are different from reduce output types

```
public class CountJob extends Configured implements Tool {


    public int run(String[] args) throws Exception {

2       setJarByClass()

4       FileInputFormat . addInputPath( job, new Path( args[0] ));
        FileOutputFormat . setOutputPath( job, new Path( args[1] ));


    }
}
```

tell Hadoop to look for the local JAR that contains the CountJob class

- Tell the job what input data to process and where to place the output.
- Do this via the static APIs on FileInputFormat and FileOutputFormat.
- Use command-line arguments to specify the file paths.

```
public class CountJob extends Configured implements Tool {

    public int run(String[] args) throws Exception {

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

5

- Launches the job and waits for it to complete.
- While it is running, it prints the progress on the console.

```
public class CountJob extends Configured implements Tool {

    public int run(String[] args) throws Exception {
        Job  job = new Job( getConf() );                          1
        job . setJarByClass( getClass() );

        FileInputFormat . addInputPath( job, new Path( args[0] ));   4
        FileOutputFormat . setOutputPath( job, new Path( args[1] ));

        job . setJobName( getClass() . getSimpleName( ) ) ;       2
        job . setMapperClass( CountMapper.class ) ;
        job . setCombinerClass( CountReducer.class ) ;
        job . setReducerClass( CountReducer.class ) ;

        job . setOutputKeyClass( Text . class );                  3
        job . setOutputValueClass( LongWritable . class );

        return  job . waitForCompletion ( true )  ?  0  :  1 ;    5
    }
}
```

Complete Driver code

```
public class CountJob extends Configured implements Tool {

    public static void main(String[] args) throws Exception {
        int rc = ToolRunner . run( new CountJob(), args );
        System.exit(rc);
    }
    public int run(String[] args) throws Exception {

        Job job = new Job( getConf() );
        job . setJarByClass( getClass() );

        FileInputFormat . addInputPath( job, new Path( args[0] ));
        FileOutputFormat . setOutputPath( job, new Path( args[1] ));

        job . setJobName( getClass() . getSimpleName( ) ) ;
        job . setMapperClass( CountMapper.class ) ;
        job . setCombinerClass( CountReducer.class ) ;
        job . setReducerClass( CountReducer.class ) ;
        job . setOutputKeyClass( Text . class );
        job . setOutputValueClass( LongWritable . class );
        return job . waitForCompletion ( true ) ? 0 : 1 ;
    }
}
```

Compiling the CountJob program

CountJob

- Folder contains CountJob program

CountMapper.java

- Contains the map function implementation

CountReducer.java

- Contains the reduce function implementation

CountJob.java

- Contains the code coordinating the execution of the map and reduce functions

- Execute the following commands in the CountJob folder:

compiles the program using the classes developed by Hadoop (hadoop-core-1.0.4.jar)

```
CountJob$ javac -cp hadoop-core-1.0.4.jar *.java
CountJob$ jar cvf CountJob.jar *.class
```

Creates a jar file called CountJob.jar to be used for running the Count program in Hadoop

Running the CountJob program in Hadoop

- Perform following task in the folder containing Hadoop installation

Starts the Hadoop services

```
hadoop $  bin/start-all.sh

hadoop $  ssh  localhost

hadoop $  mkdir input
```

Establishes a secure connection with the machine

Creates the directory to put input.txt file

- Copy the CountJob.jar and the input.txt file into the Hadoop installation directory

```
hadoop $ ls

  CountJob.jar
  Input.txt
  ..................................
```

```
hadoop-examples-1.0.4.jar
hadoop-minicluster-1.0.4.jar
hadoop-test-1.0.4.jar
hadoop-tools-1.0.4.jar
```

- Now prepare the input for the CountJob program:

Creates a directory called input in the HDFS

```
hadoop$ bin/hadoop   dfs   -mkdir   input

hadoop$ bin/hadoop   dfs   -put   input.txt   input
```

Copy input.txt into the input folder in HDFS

- **Finally execute the following commands:**

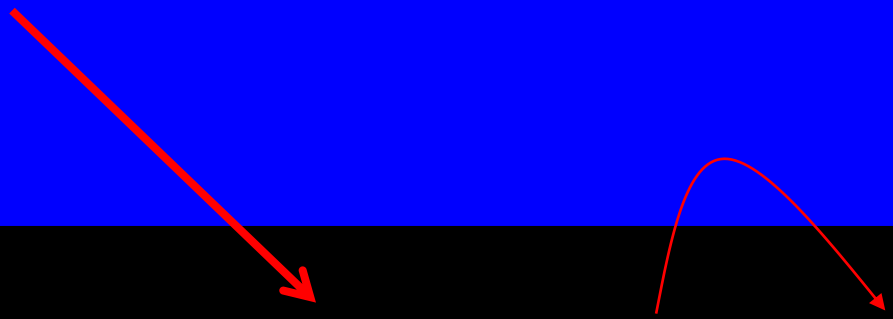
Run the CountJob program in Hadoop

```
hadoop$ bin/hadoop jar CountJob.jar   CountJob  input  output
```

HDFS folder where the input files reside

HDFS folder that will contain the output files

Copies the output folder from
HDFS to local machine



```
hadoop$ bin/hadoop  dfs  -get   output  output
```

- The result of the CountJob program in a file (probably) called part-00000

Step - IV

Assignment-1:

- **Write Character count mapper, reducer and driver classes using Hadoop APIs for the sample input file as shown below:**

SampleInputFile.txt

```
lmnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
mnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
mnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
mnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
mnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
lmnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
mnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
mnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
mnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
mnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
lmnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
mnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
mnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
mnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
mnmnlllmmlnnnmlllmnlmllmnlmmmnlmnnlmnlmnlnnnmlnmmmllmnmnmlnmnmlmnlmnllnmn
```

Sample Output format

l	236
m	111
n	134

Assignment-2:

- **Write a MapReduce program using Hadoop APIs for following task:**

Consider an input file containing integer numbers.
Provide this file as an input to the program and the program finds maximum number of two consecutive columns and produces output.

SampleInputFile.txt

```
8,4,5,6,10,9,2,3,5,6
1,2,4,3,7,6,8,9,4,5
3,5,2,3,7,5,8,9,9,0
.
.
.
.
.
.
.
5,7,7,4,8,6,5,7,9,0
```

Sample Output format

```
1st&2ndColumn 8
2nd&3rdColumn 6
.
.
.
.
9th&10thColumn 6
```

Thank You!