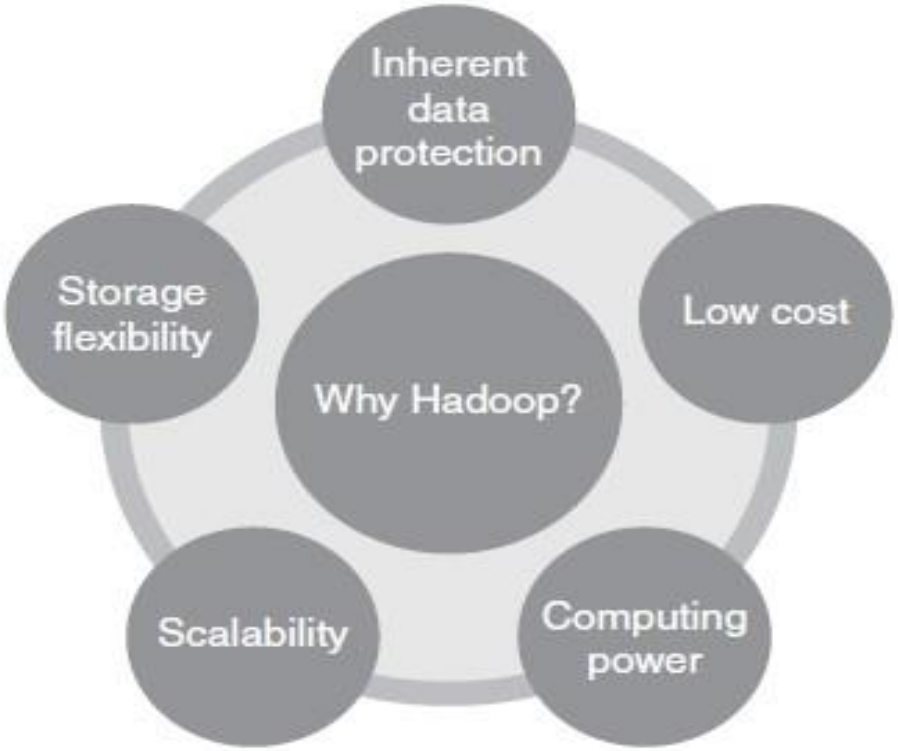
Introduction to Hadoop – (HDFS, MR, YARN)

Why Hadoop

Key Considerations are:

- Capability to handle large amount of data
- different categories of data
- fairly quickly

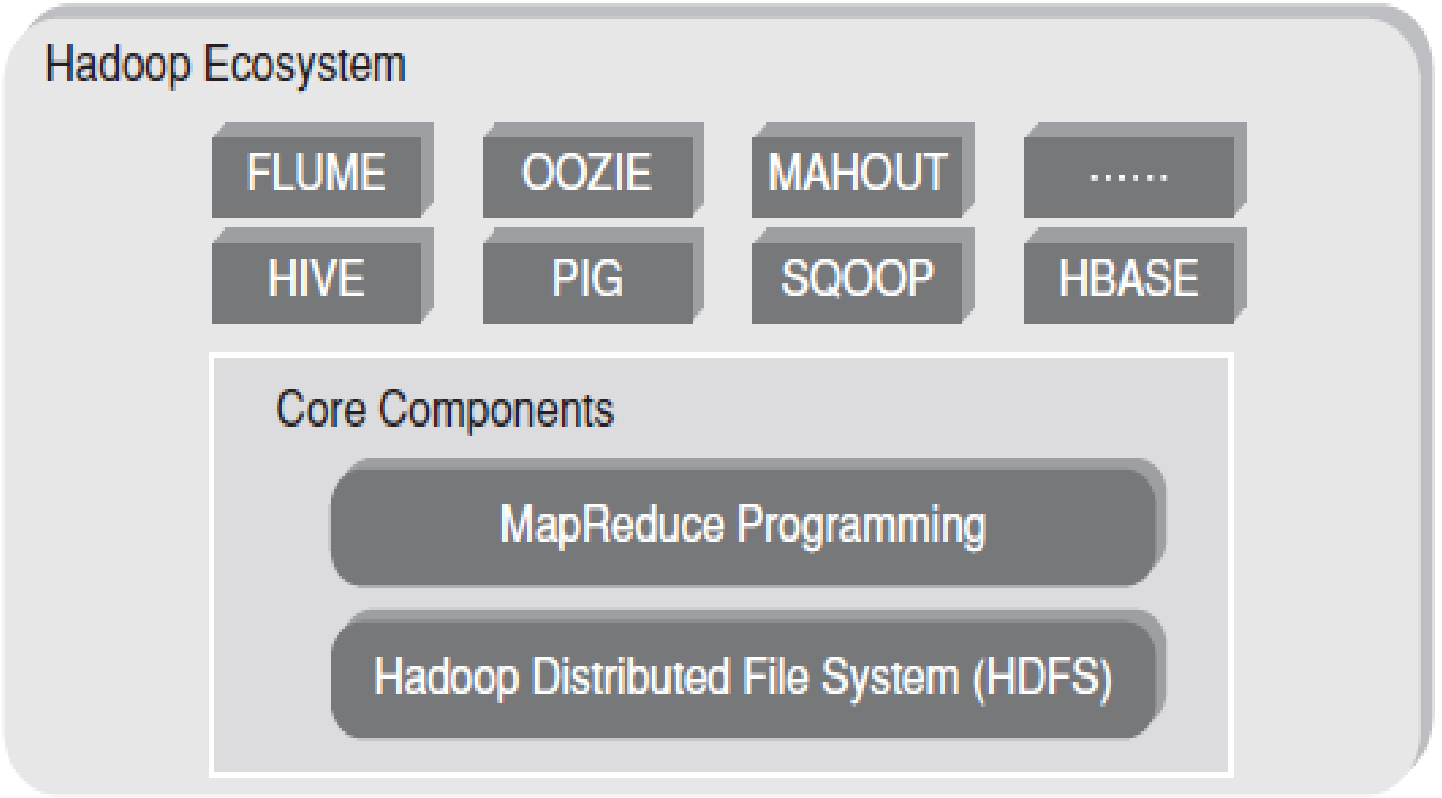


Why not RDBMS

RDBMS versus HADOOP

PARAMETERS	RDBMS	HADOOP
System	Relational Database Management System.	Node Based Flat Structure.
Data	Suitable for structured data.	Suitable for structured, unstructured data. Supports variety of data formats in real time such as XML, JSON, text based flat file formats, etc.
Processing	OLTP	Analytical, Big Data Processing
Choice	When the data needs consistent relationship.	Big Data processing, which does not require any consistent relationships between data.
Processor	Needs expensive hardware or high-end processors to store huge volumes of data.	In a Hadoop Cluster, a node requires only a processor, a network card, and few hard drives.
Cost	Cost around $10,000 to $14,000 per terabytes of storage.	Cost around $4,000 per terabytes of storage.

Hadoop Components



Hadoop Components

Hadoop Core Components:

HDFS:
(a) Storage component.
(b) Distributes data across several nodes.
(c) Natively redundant.

MapReduce:
(a) Computational framework.
(b) Splits a task across multiple nodes.
(c) Processes data in parallel.

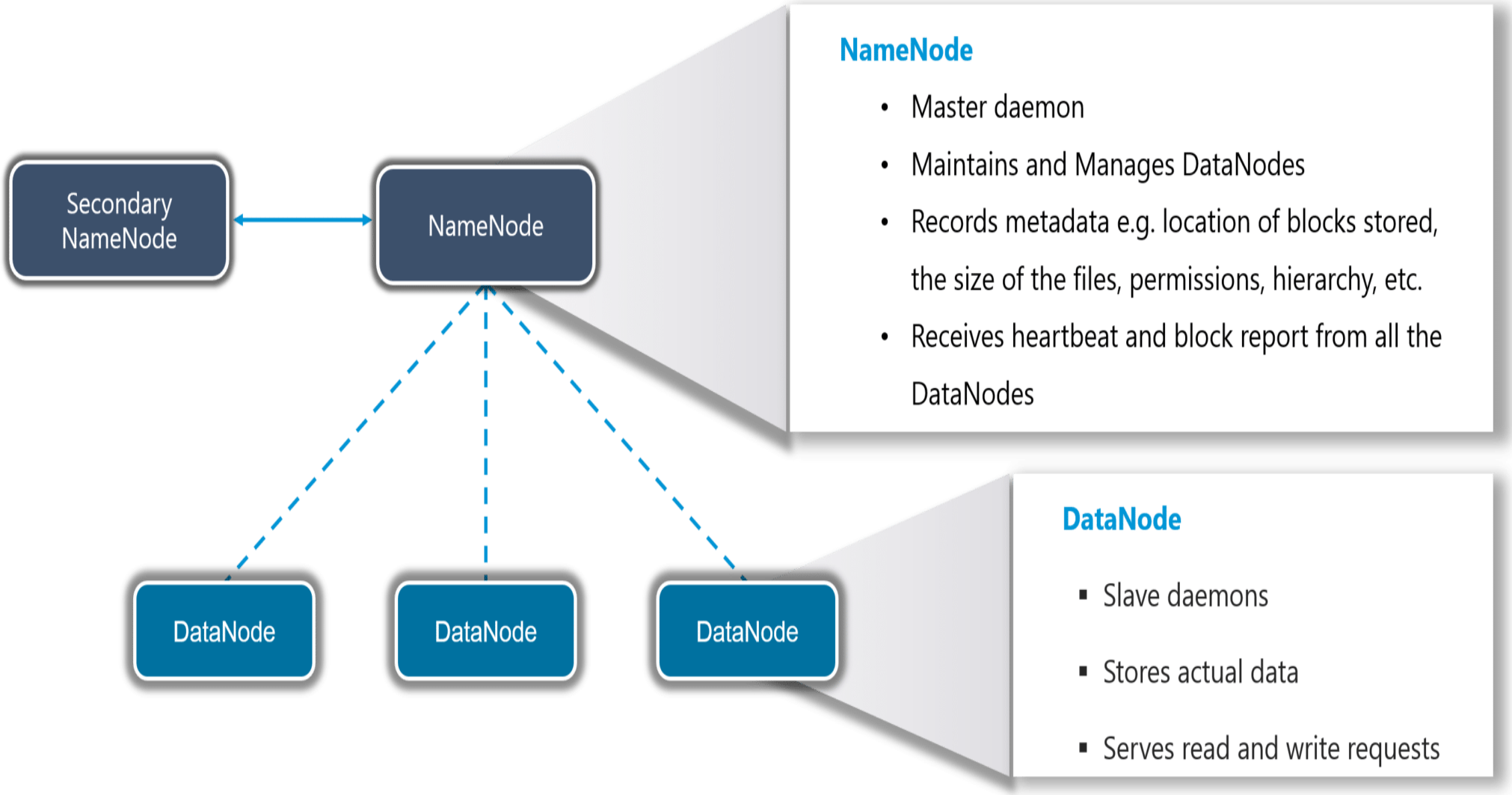
Why HDFS ?

- If we have a distributed processing frameworks where multiple tasks are launched in parallel, but a centralized file system which has to be accessed by the parallel tasks, there will be issues such as:
  - High network traffic from nodes running tasks to node on which file resides.
  - Synchronization issues since multiple tasks are accessing the same file.
  - Availability issues if the node containing the file goes down.

HDFS: The Hadoop Distributed File System

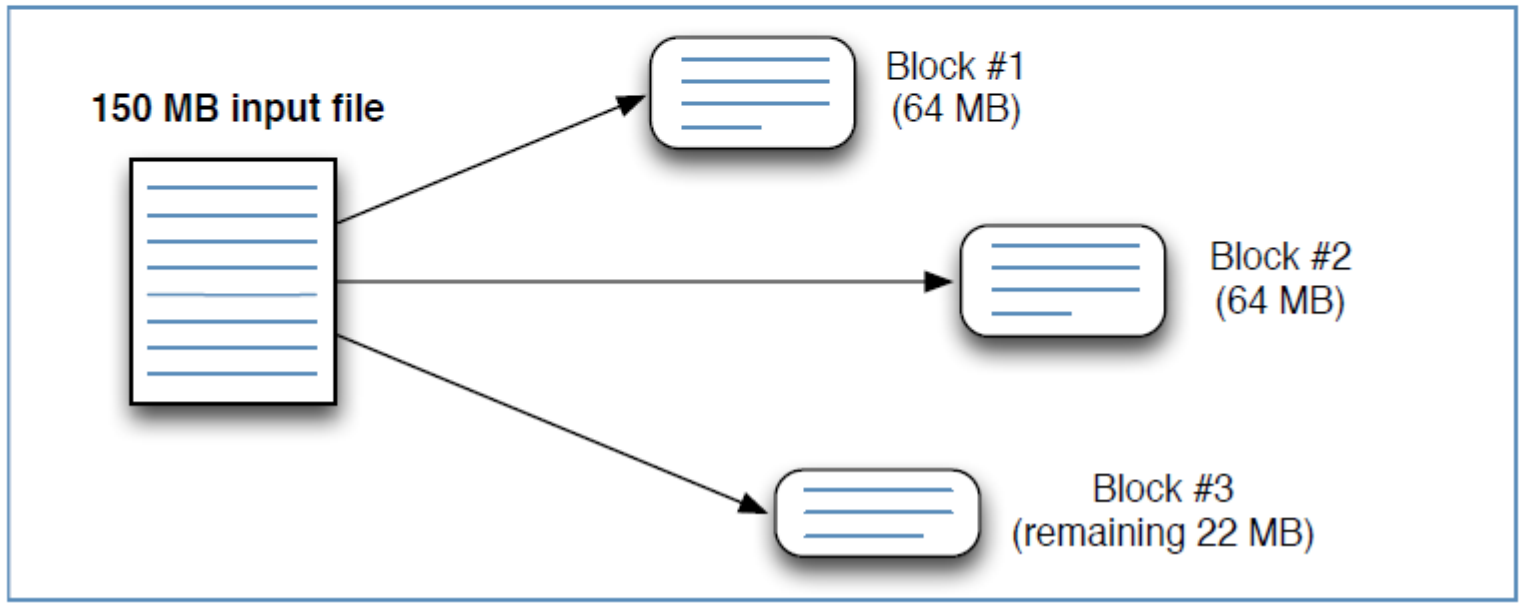
- **Based on Google's GFS (Google File System)**
- **Provides redundant storage for massive amounts of data**
  - Using industry-standard hardware
- **At load time, data is distributed across all nodes**
  - Provides for efficient processing

HDFS Daemons



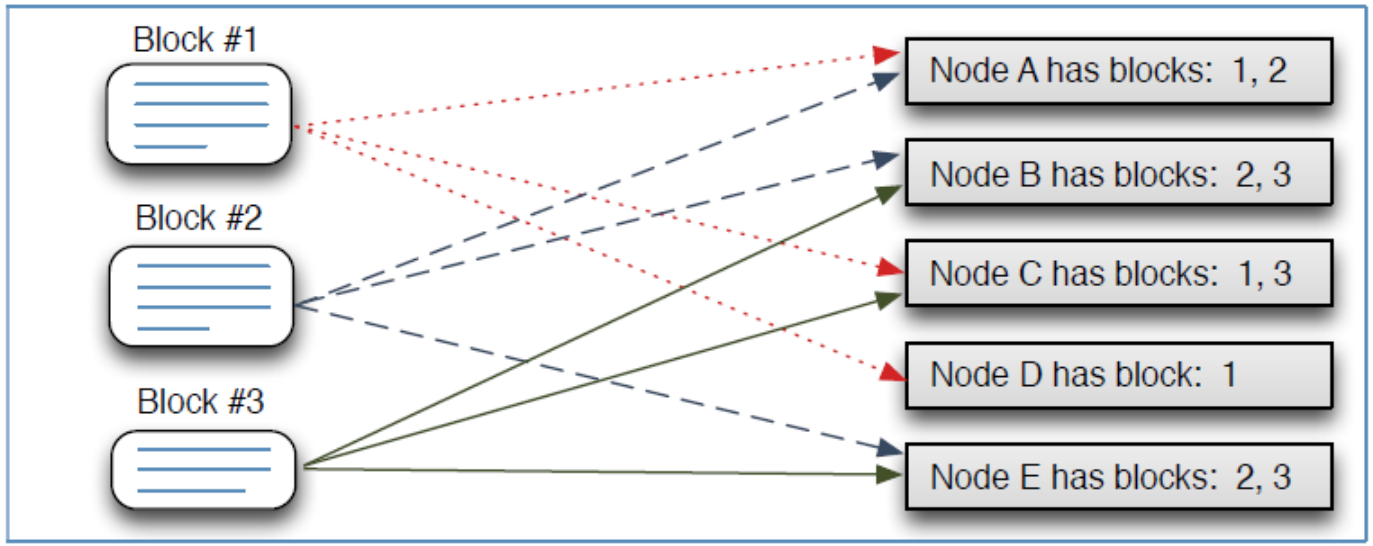
HDFS Blocks

- **When a file is added to HDFS, it is split into blocks**
- **This is a similar concept to native filesystems**
  - HDFS uses a *much* larger block size
  - Default block size is 64 MB (configurable)



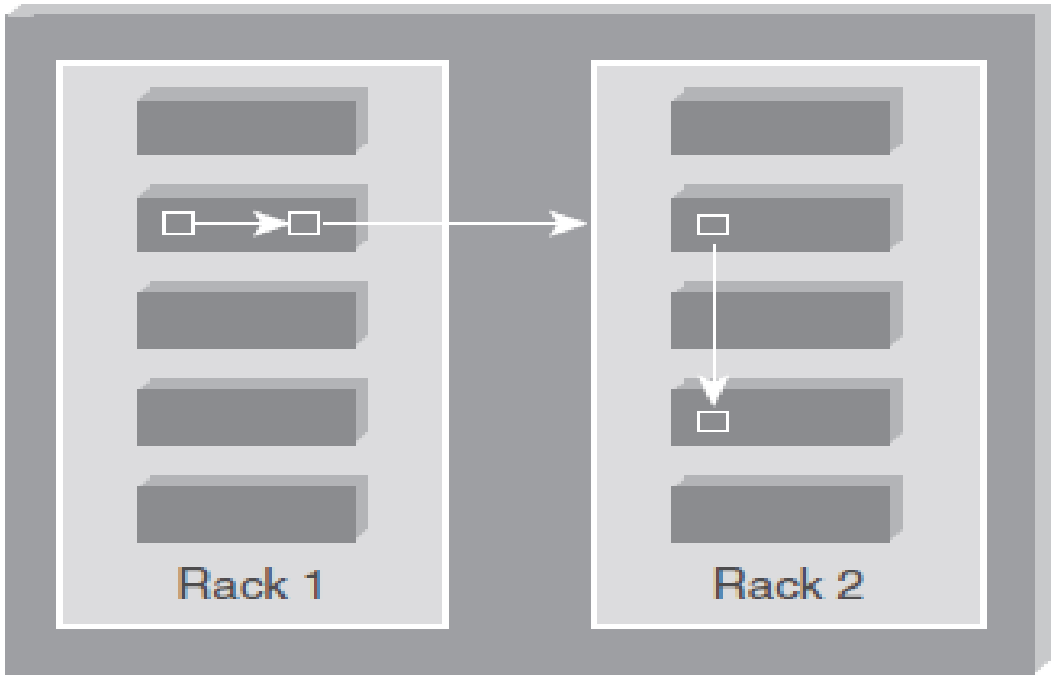
Block Replication

- **These blocks are replicated to nodes throughout the cluster**
  - Based on the *replication factor* (default is three)
- **Replication increases reliability and performance**
  - Reliability: data can tolerate loss of all but one replica
  - Performance: more opportunities for data locality



Replica Placement Strategy

As per the Hadoop Replica Placement Strategy, first replica is placed on the same node as the client. Then it places second replica on a node that is present on different rack. It places the third replica on the same rack as second, but on a different node in the rack. Once replica locations have been set, a pipeline is built. This strategy provides good reliability.



Terms to consider

Blocks- The smaller pieces of large file
Rack ID- Used to identify the data nodes in the rack
Rack- Collection of Data Nodes within the cluster
File System Namespace- Collection of files in the cluster
FSImage- Is file containing File system Namespace and file properties
EditLog- Transaction log to record every transaction that happens to file system metadata

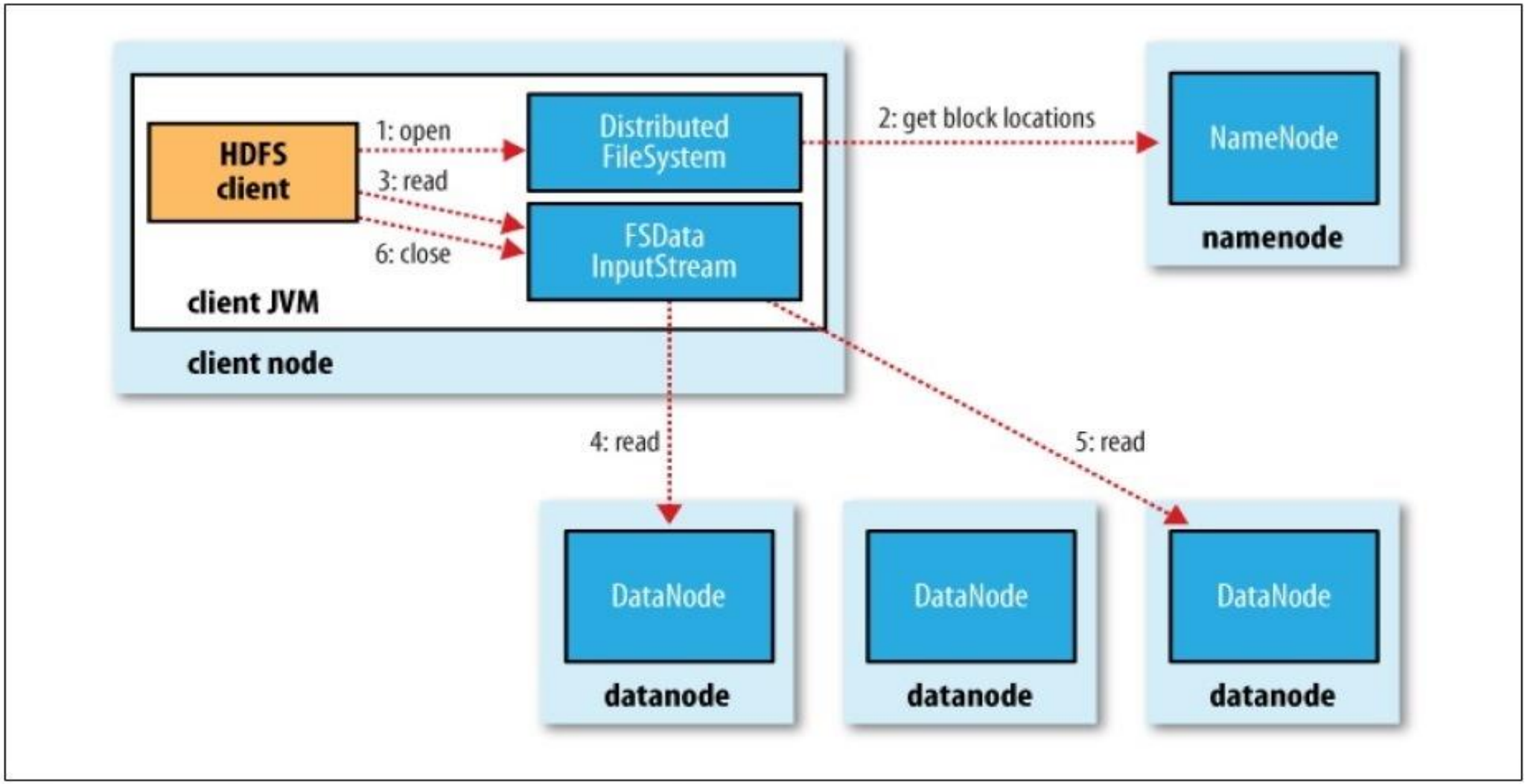
The NameNode

- **The NameNode stores all *metadata***
  - Information about file locations in HDFS
  - Information about file ownership and permissions
  - Names of the individual blocks
  - Locations of the blocks
- **Metadata is stored on disk and read when the NameNode daemon starts up**

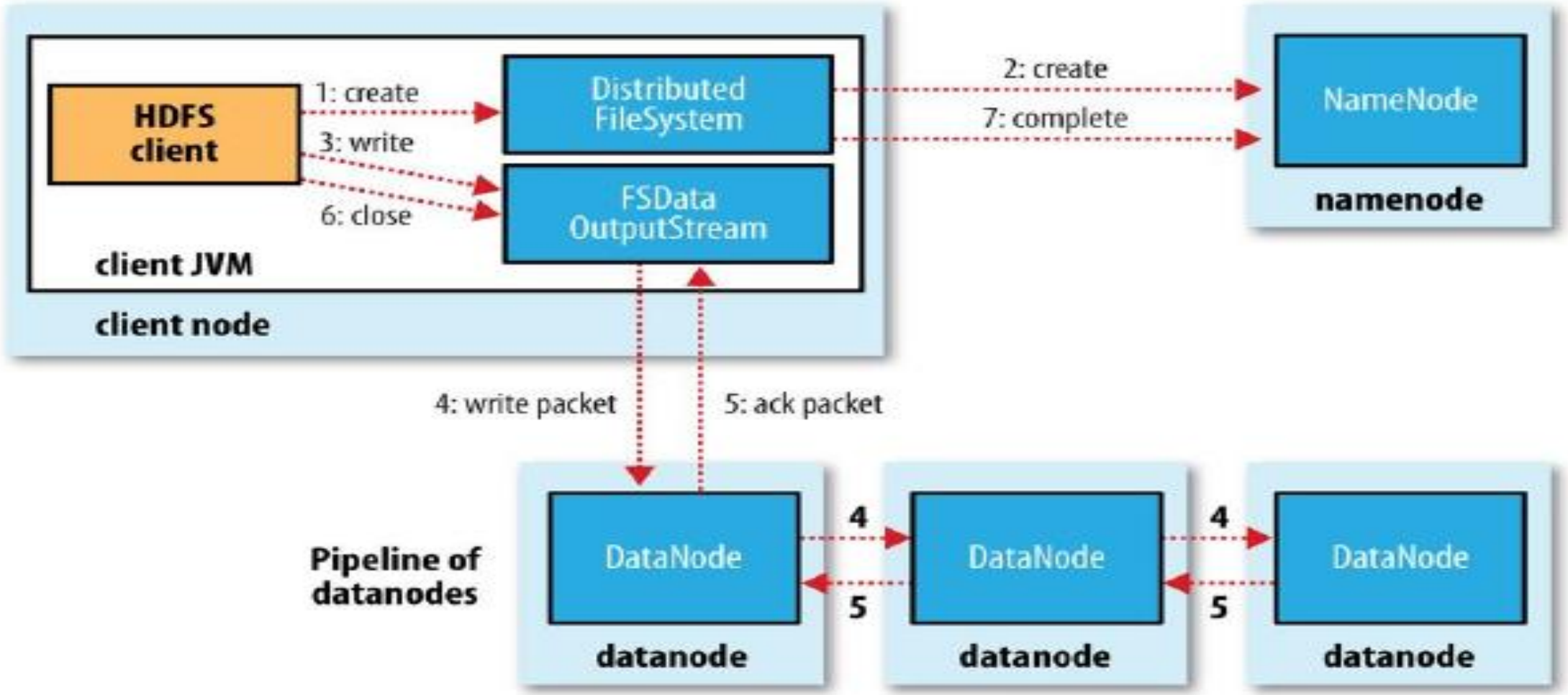
The DataNode

- **Actual contents of the files are stored as *blocks* on the slave nodes**
- **Blocks are simply files on the slave nodes' underlying filesystem**
  - Named `blk_xxxxxxx`
- **Each block is stored on multiple different nodes for redundancy**
  - Default is three replicas
- **Each slave node runs a DataNode daemon**
  - Controls access to the blocks
  - Communicates with the NameNode

File Read from HDFS



File Write into HDFS



Getting data in and out of HDFS

- Let us say we have HDFS running on a UNIX cluster.
- The UNIX files are not part of HDFS.
- To a copy a file from UNIX filesystem to HDFS, do the following:
  - **hadoop fs –mkdir /hdfsfldr**
  - **hadoop fs -put /root/unixfldr/input.txt /hdfsfldr/input.txt**
- To copy a file from HDFS to UNIX filesystem:
  - **Hadoop fs –get /hdfsfldr/input.txt /root/unixfldr/input.txt**
- To list the contents of a HDFS directory:
  - **Hadoop fs –ls –R /hdfsfldr**

- There are powerful tools to ingest data into HDFS: Sqoop(from RDMS), Flume etc.

HDFS Features

- **High performance**
- **Fault tolerance**
- **Relatively simple centralized management**
  - Master-slave architecture
- **Security**
  - Two levels from which to choose
- **Optimized for distributed processing**
  - Data locality

HDFS can optimize data placement so it is local to the tasks.

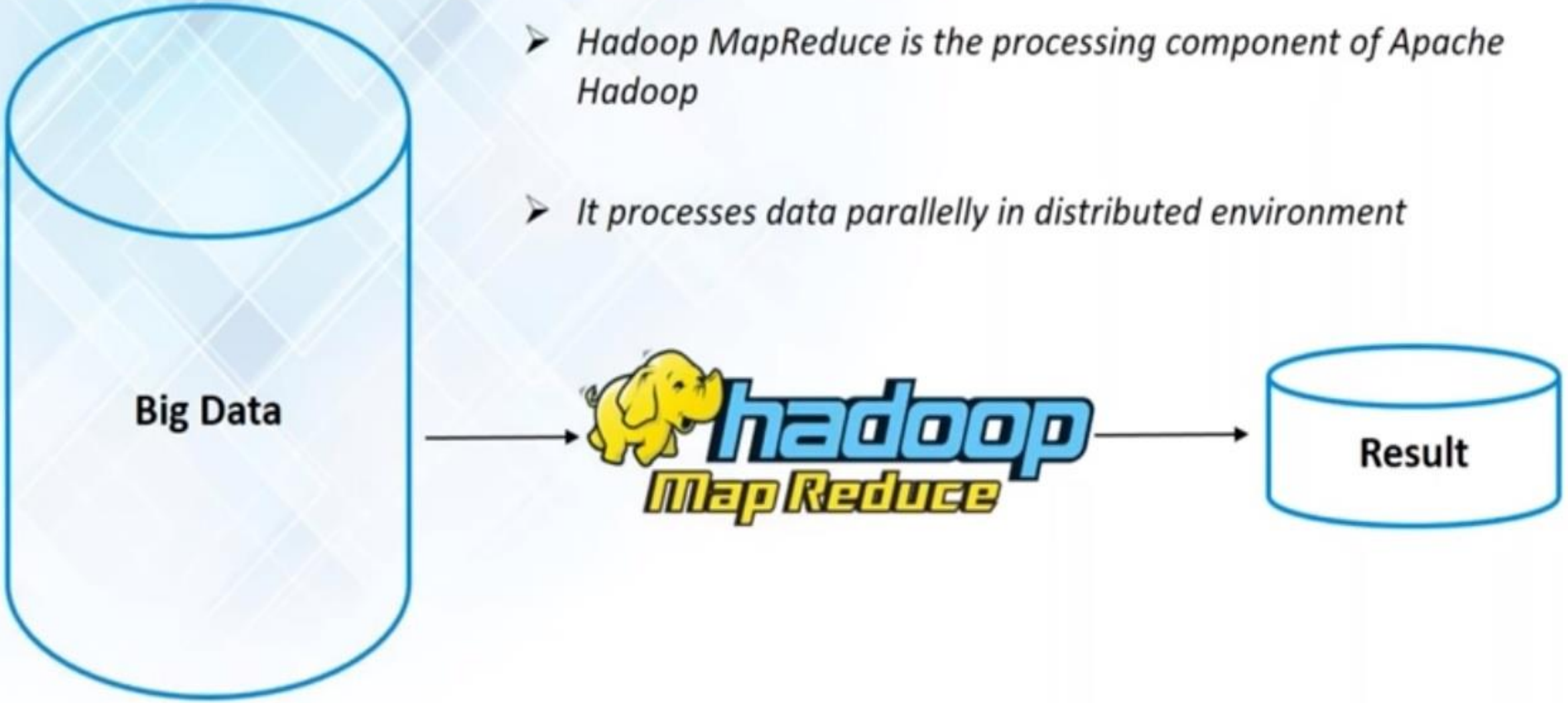
- **Scalability**

Review Questions 2

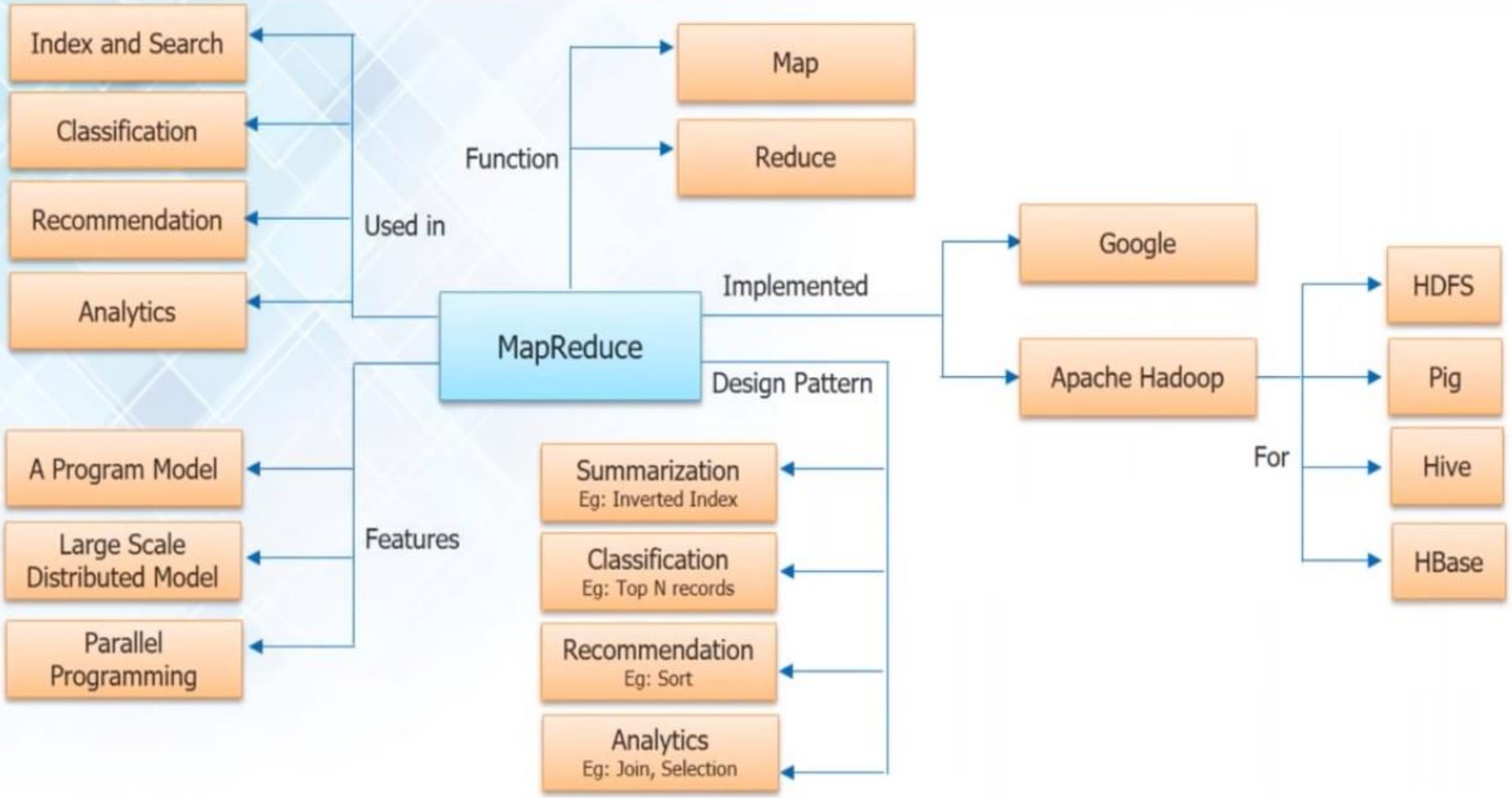
- How is a file stored in HDFS to support parallel processing as well as support fault tolerance.
- What are the HDFS Daemons and what are their responsibilities ?
- What assumptions were made in the design of HDFS ? Do these assumptions make sense ? Discuss why.

MapReduce: Data Processing Using Programming

- *Hadoop MapReduce is the processing component of Apache Hadoop*
- *It processes data parallelly in distributed environment*



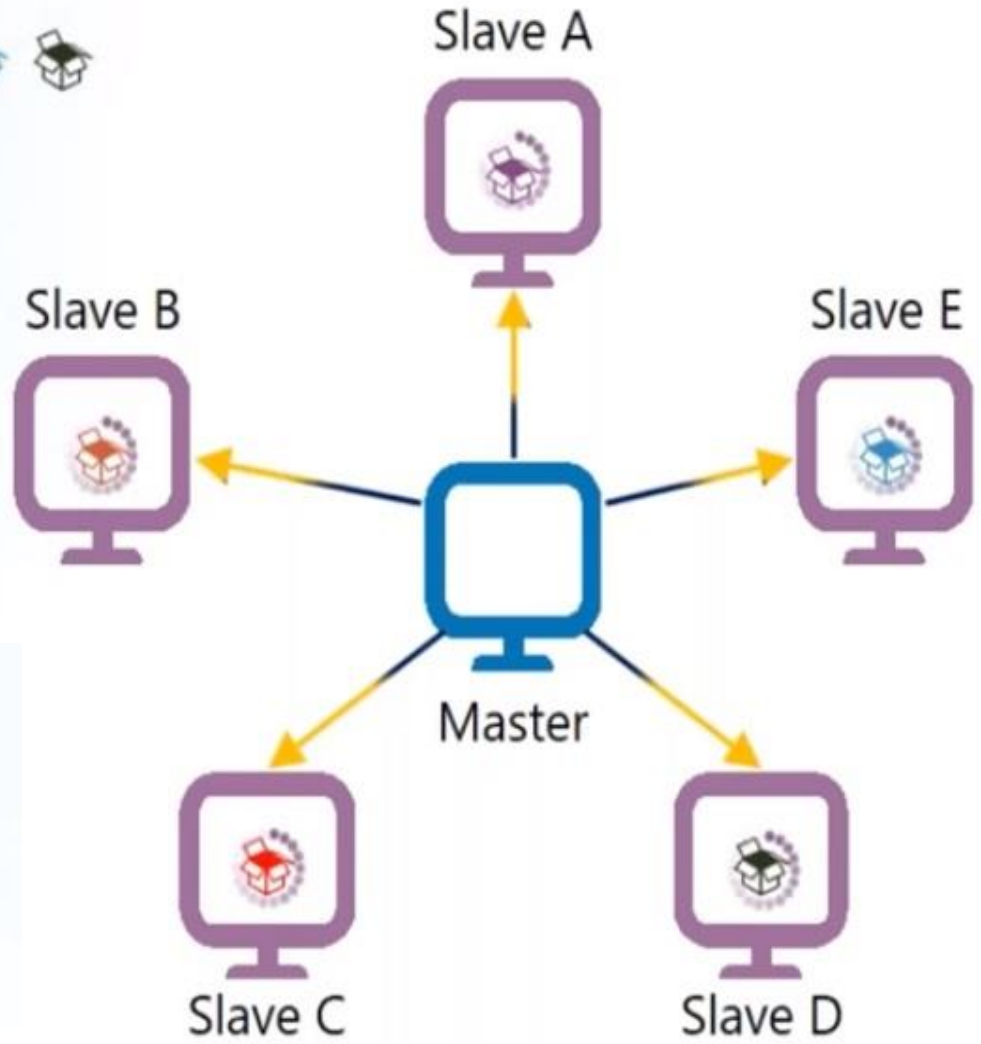
MapReduce In Nutshell



Advantage 1: Parallel Processing

Data →

- Data is processed in parallel
- Processing becomes fast
- Moving Data to processing is very costly
- In MapReduce, we move processing to Data

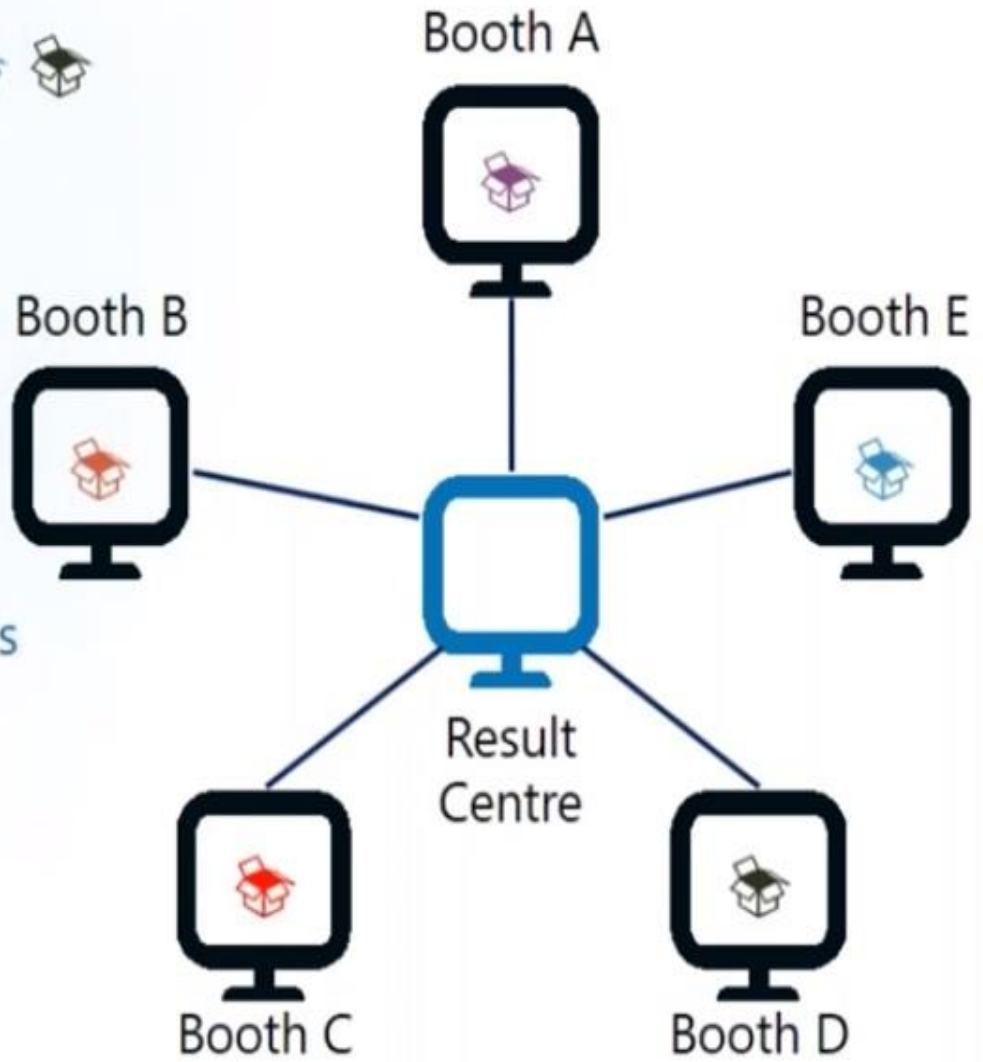


Election Votes Counting

Data →

Election Votes Casting

- Votes is stored at different Booths
- Result Centre has the details of all the Booths

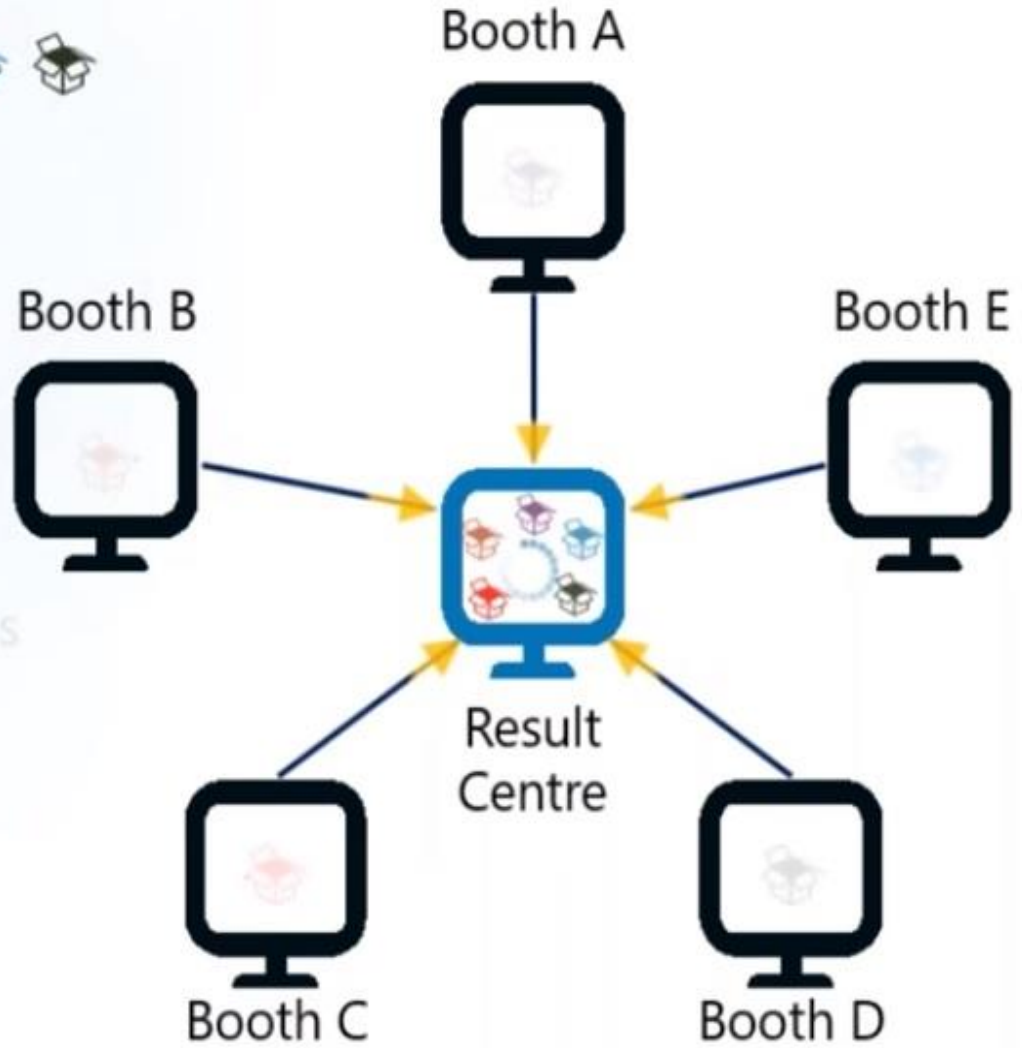


Election Votes Counting – Traditional Way

Data →

Counting – Traditional Approach

- Votes are moved to Result Centre for counting
- Result Centre has the details of all the Booths
- Moving all the votes to Centre is costly
- Result Centre is over-burdened
- Counting takes time

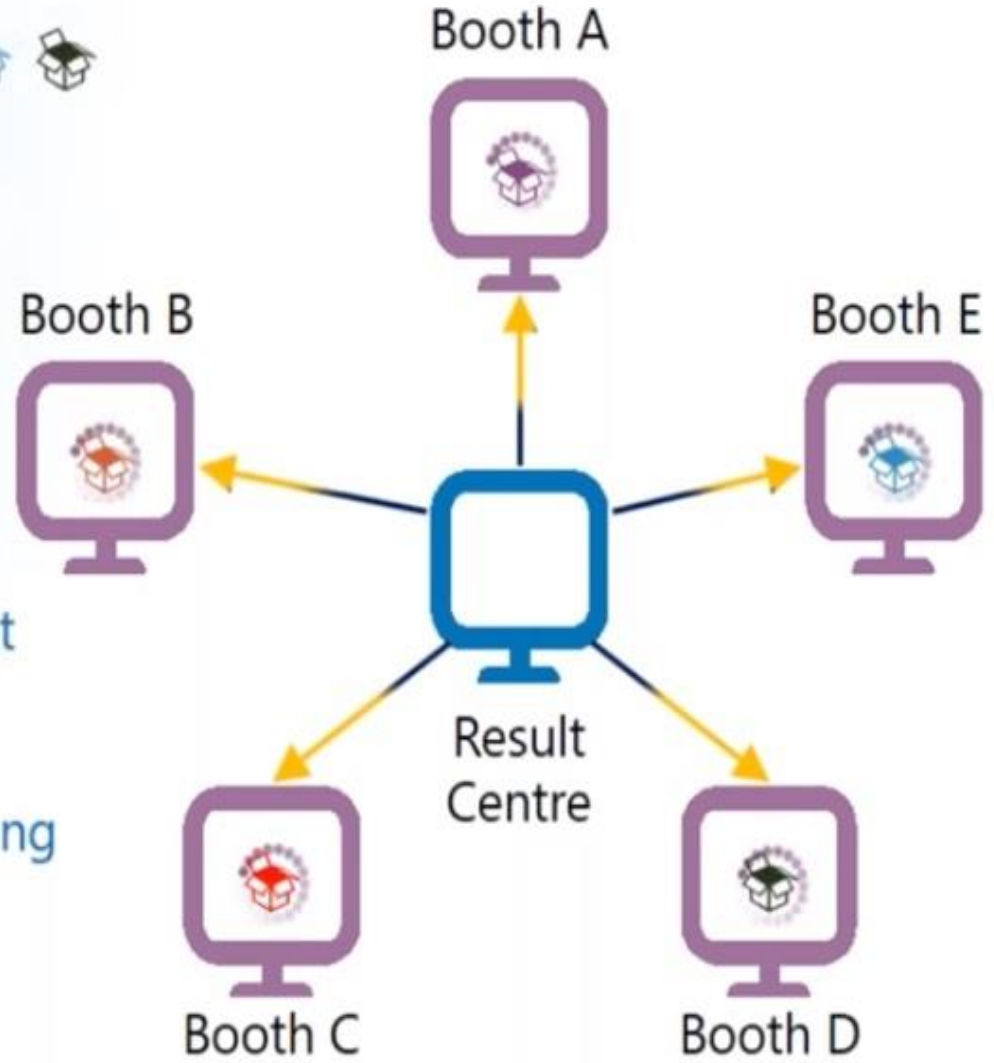


Election Votes Counting – MapReduce Way

Votes→

Counting – MapReduce Approach

- Votes are counted at individual booths
- Booth-wise results are sent back to the result centre
- Final Result is declared easily and quickly using this way



Hadoop Map Reduce

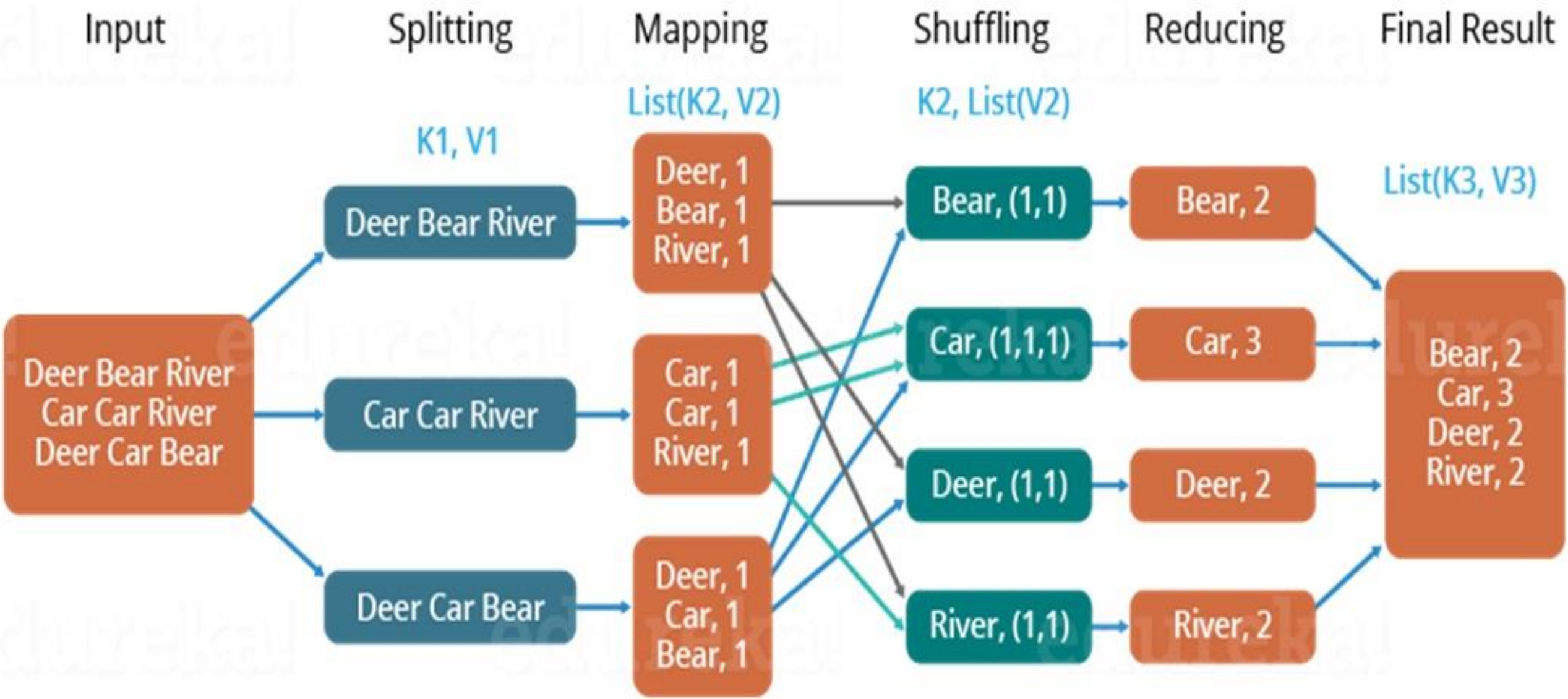
JobTracker

- Accepts Map-Reduce tasks from the Users
- Assigns tasks to the **Task Trackers** & monitors their status

TaskTracker

- Runs Map-Reduce tasks
- Sends heart-beat to Job Tracker
- Retrieves Job resources from HDFS

The Overall MapReduce Word Count Process



Review Questions 3

- What are the main functions of MapReduce ?
- What are the two main phases of the MapReduce programming ? Explain through an example.

Why YARN ? (Yet Another Resource Negotiator)

- How does MapReduce know how many nodes are there in the cluster ?
- How does MapReduce know which nodes are alive ?
- How does MapReduce know if a node crashes while executing a task or the node “hangs” ?
- Who is going to handle cluster security ?

- YARN does all of the above.
  - MapReduce runs tasks generated by a MapReduce program with the help of YARN.
  - YARN can be used by multiple MapReduce Applications.

Limitations of Hadoop 1.0 Architecture

1. Single NameNode is responsible for managing entire namespace for Hadoop Cluster.

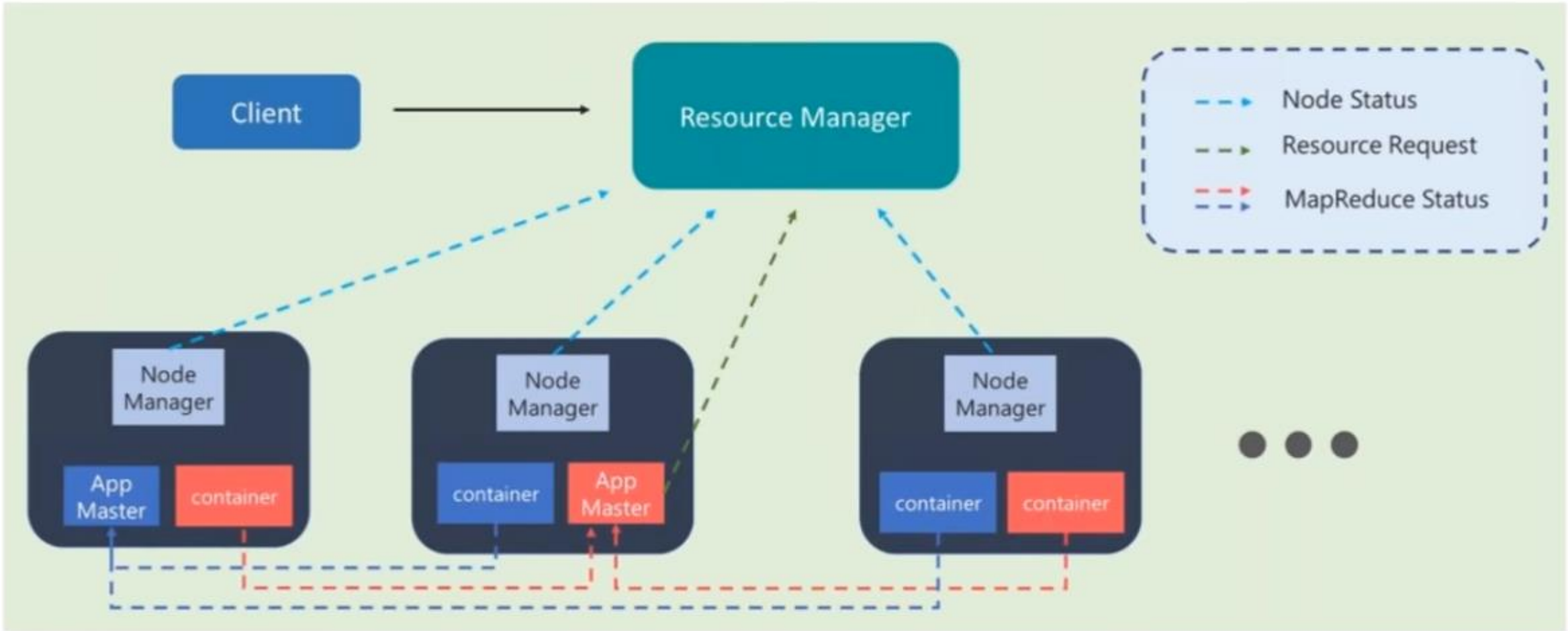
2. It has a restricted processing model which is suitable for batch-oriented MapReduce jobs.

3. Hadoop MapReduce is not suitable for interactive analysis.

4. Hadoop 1.0 is not suitable for machine learning algorithms, graphs, and other memory intensive algorithms.

5. **MapReduce** is responsible for **cluster resource management and data processing**.

Entities in YARN



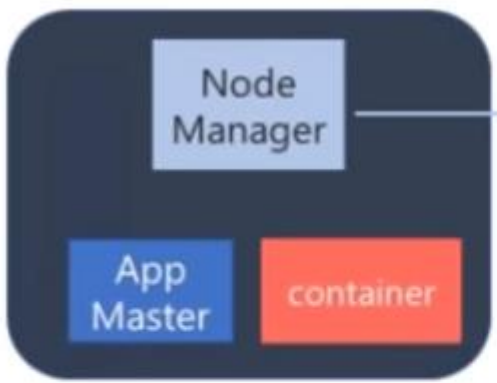
ResourceManager

Scheduler is *responsible* for *allocating resources* to *applications* & does not offers *guarantees* about *restarting failed tasks*. *Scheduler* has a *pluggable policy* like *CapacityScheduler* and the *FairScheduler*

ApplicationsManager is responsible for *accepting job-submissions*, *negotiating* the *first container* for *executing* the application specific *ApplicationMaster* and provides the service for *restarting* the *ApplicationMaster* container *on failure*.

NodeManager

NodeManager takes care of individual compute nodes in a Hadoop cluster

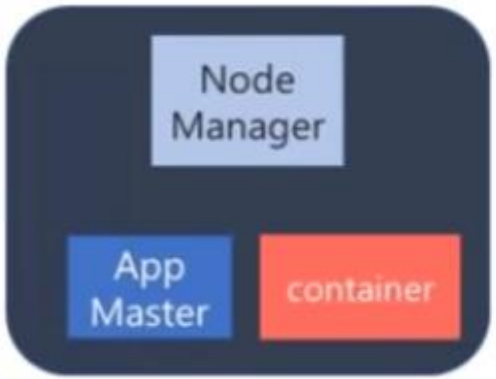


Node Manager manages following:

- *Container Lifecycle Management*
- *Container Dependencies*
- *Container Leases Node & Container Resource Usage*
- *Node Health*
- *Log Management*
- *Reporting Node & container status to RM*

ApplicationMaster

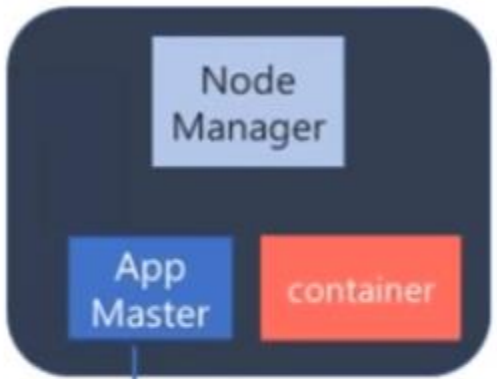
ApplicationMaster is the process that coordinates an application's execution in the cluster



Each application has its own unique ApplicationMaster, which is tasked with negotiating resources (containers) from the ResourceManager and working with the NodeManager to execute and monitor the tasks.

ApplicationMaster

ApplicationMaster is the process that coordinates an application's execution in the cluster



Application Master:

- *Manages the Application Lifecycle*
- *Makes dynamic adjustment to resource consumption*
- *Manages execution flow*
- *Manages faults*
- *Provides status & metrics to the RM*
- *Interacts with NodeManager & RM*

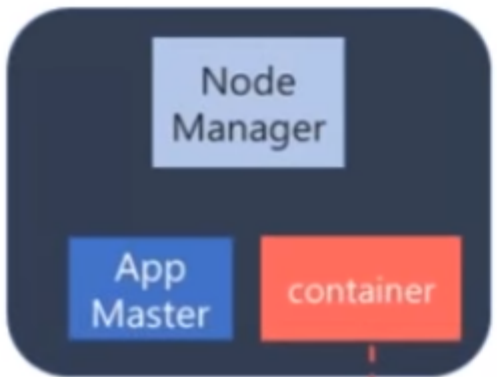
Container

Container is a collection of physical resources such as RAM, CPU cores, and disks on a single node.

- *There can be multiple containers on a single node*
- *Every node in the system is composed of multiple containers*
- *The ApplicationMaster can request any container so as to occupy a multiple of the minimum size*

Container

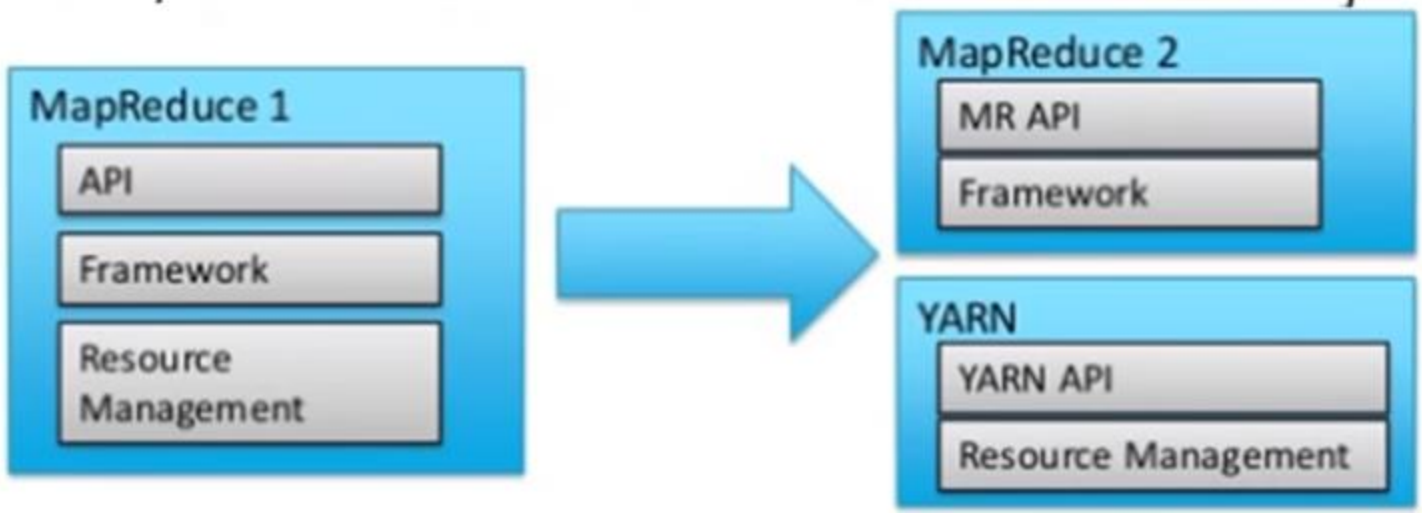
Container is a collection of physical resources such as RAM, CPU cores, and disks on a single node.



Container Launch Context (CLC) includes

- *Environment Variable*
- *Dependencies, i.e. local resources*
- *Security Tokens*
- *Command necessary to create the process, that application wants to launch*

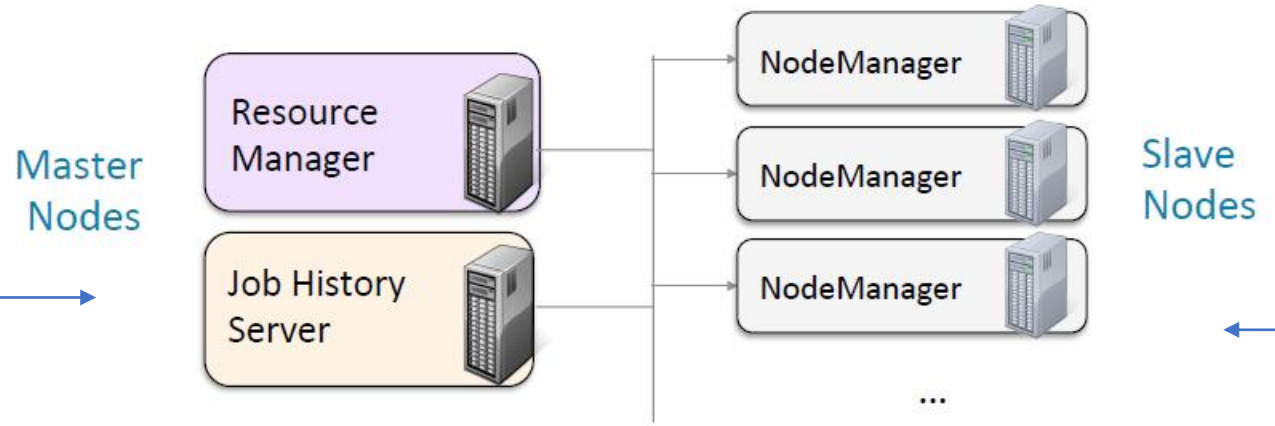
- It is a layer that separates the resource management layer and the processing components layer.
- MapReduce2 moves Resource management (like infrastructure to monitor nodes, allocate resources and schedule jobs)into YARN



YARN Daemons

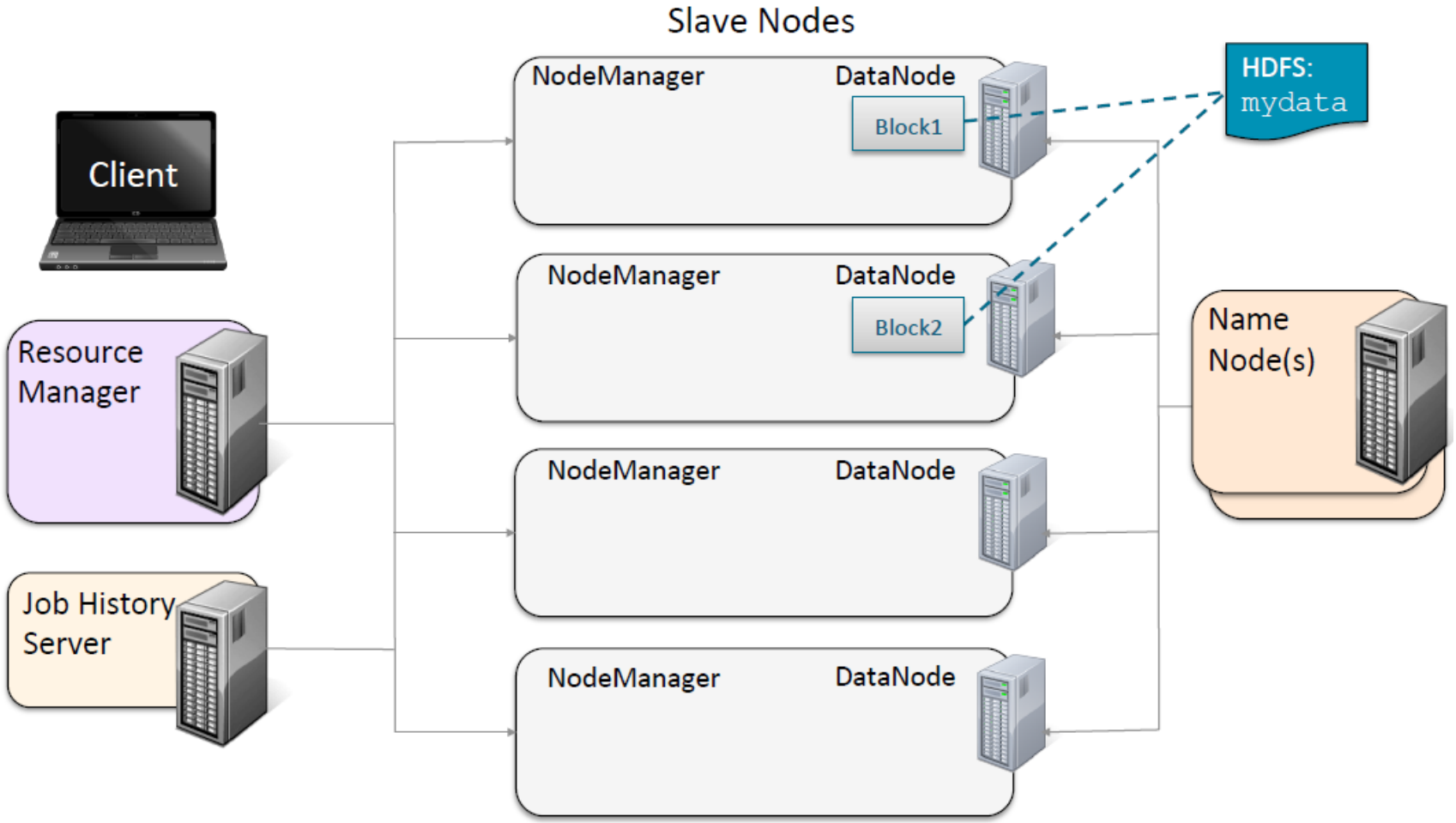
- **YARN daemons**
  - ResourceManager – one per cluster
    - Initiates application startup, schedules resource usage on slave nodes
  - NodeManager – one per slave node
    - Starts application processes, manages resources on slave nodes
  - JobHistoryServer – one per cluster
    - Archives jobs' metrics and metadata

In HDFS, NameNode Daemon runs on the master node, in YARN Resource Manager runs on the master.

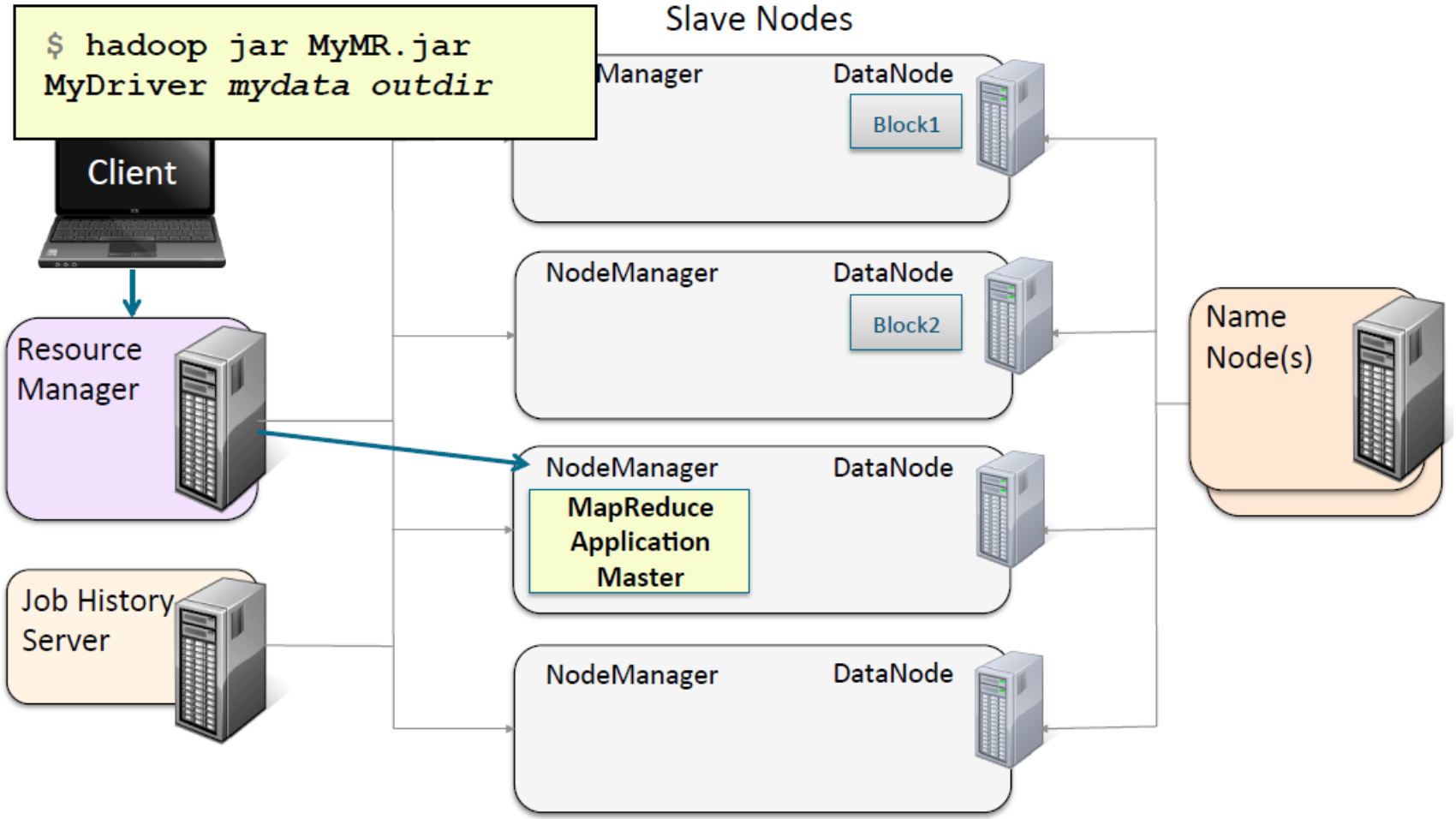


In HDFS, DataNode daemons run on the slave nodes, in YARN, NodeManagers run on the slaves.

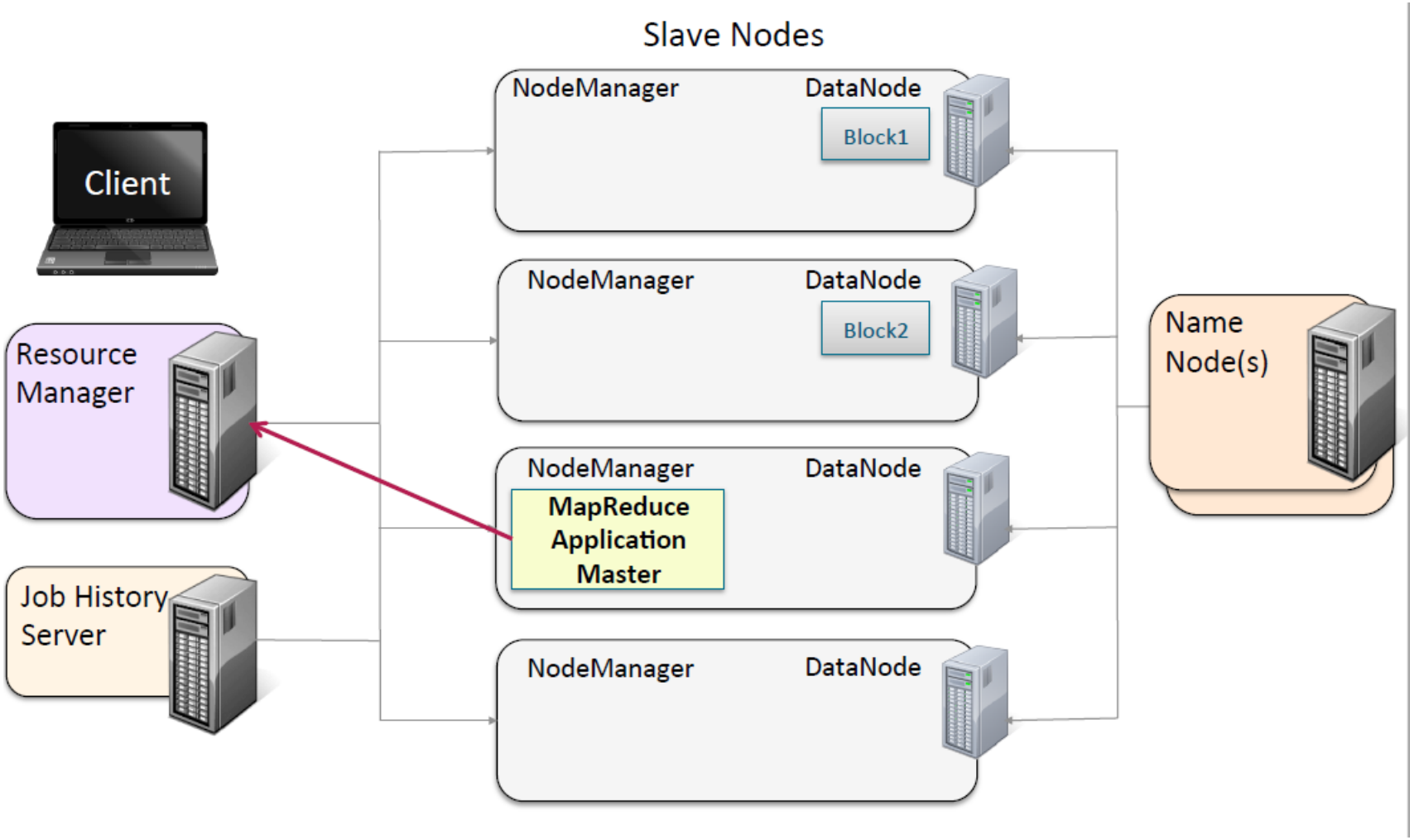
Running a Job on a YARN Cluster



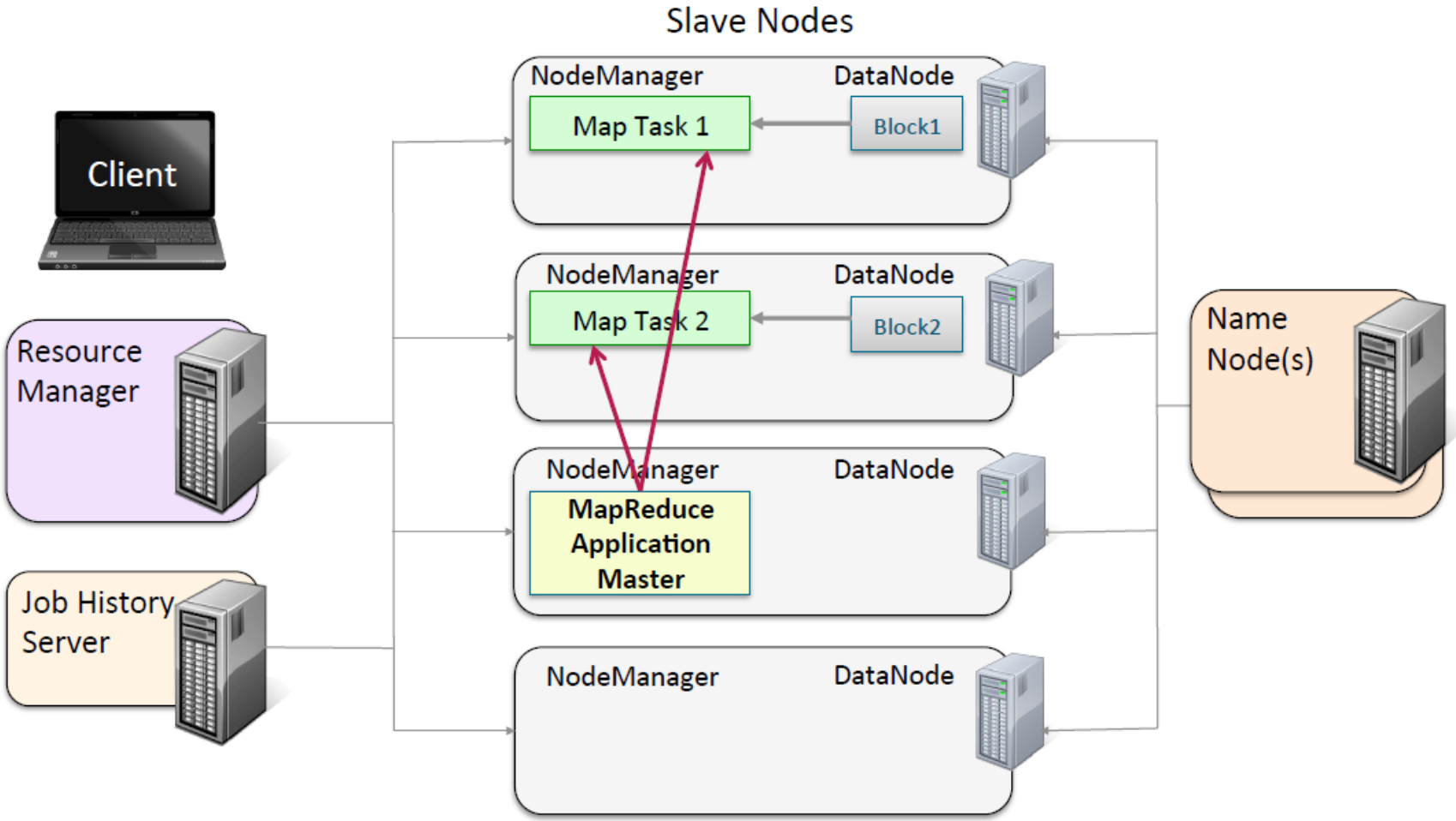
Running a Job on a YARN Cluster: Job Submission



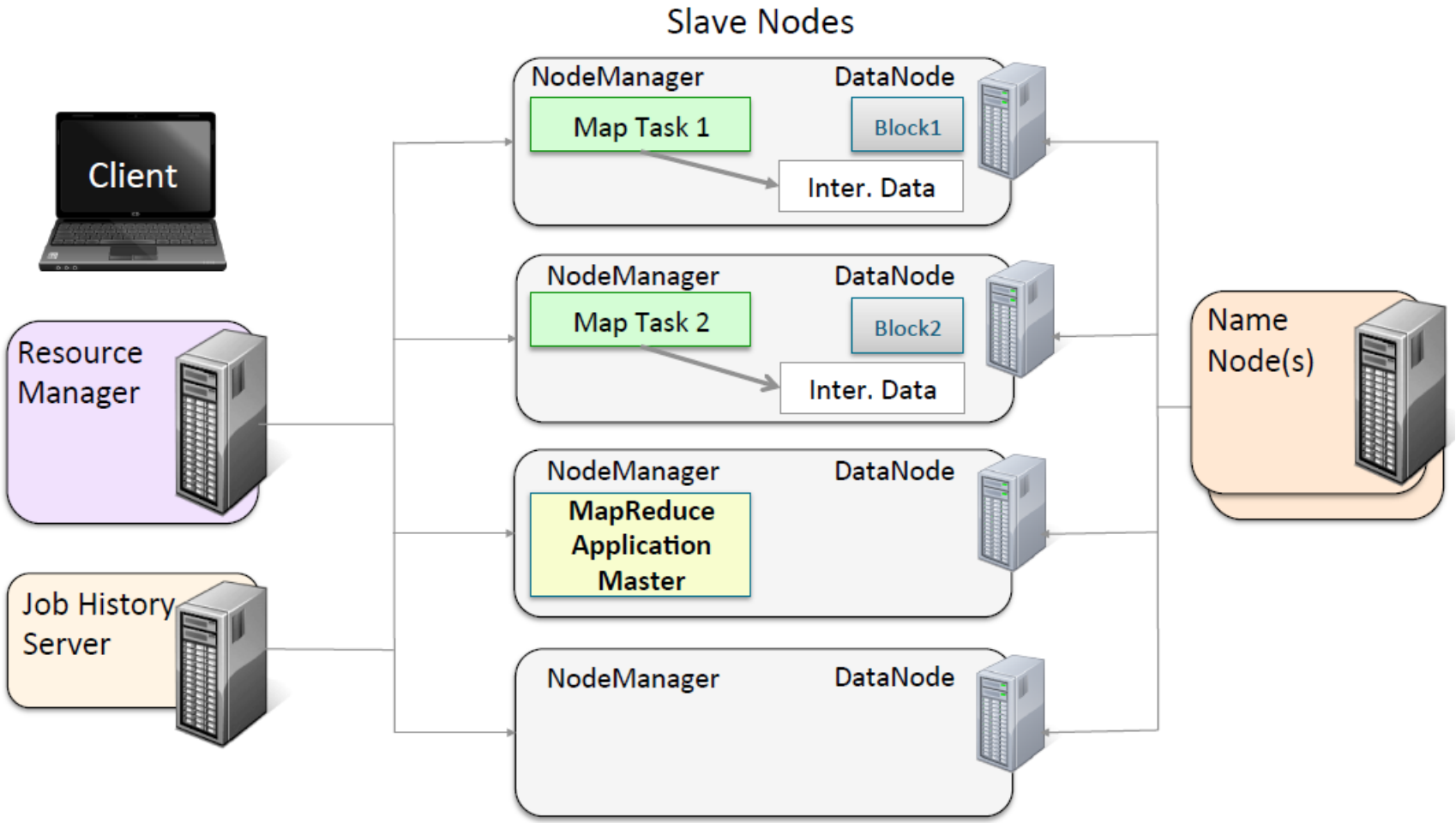
Running a Job on a YARN Cluster:Map TaskResource Request



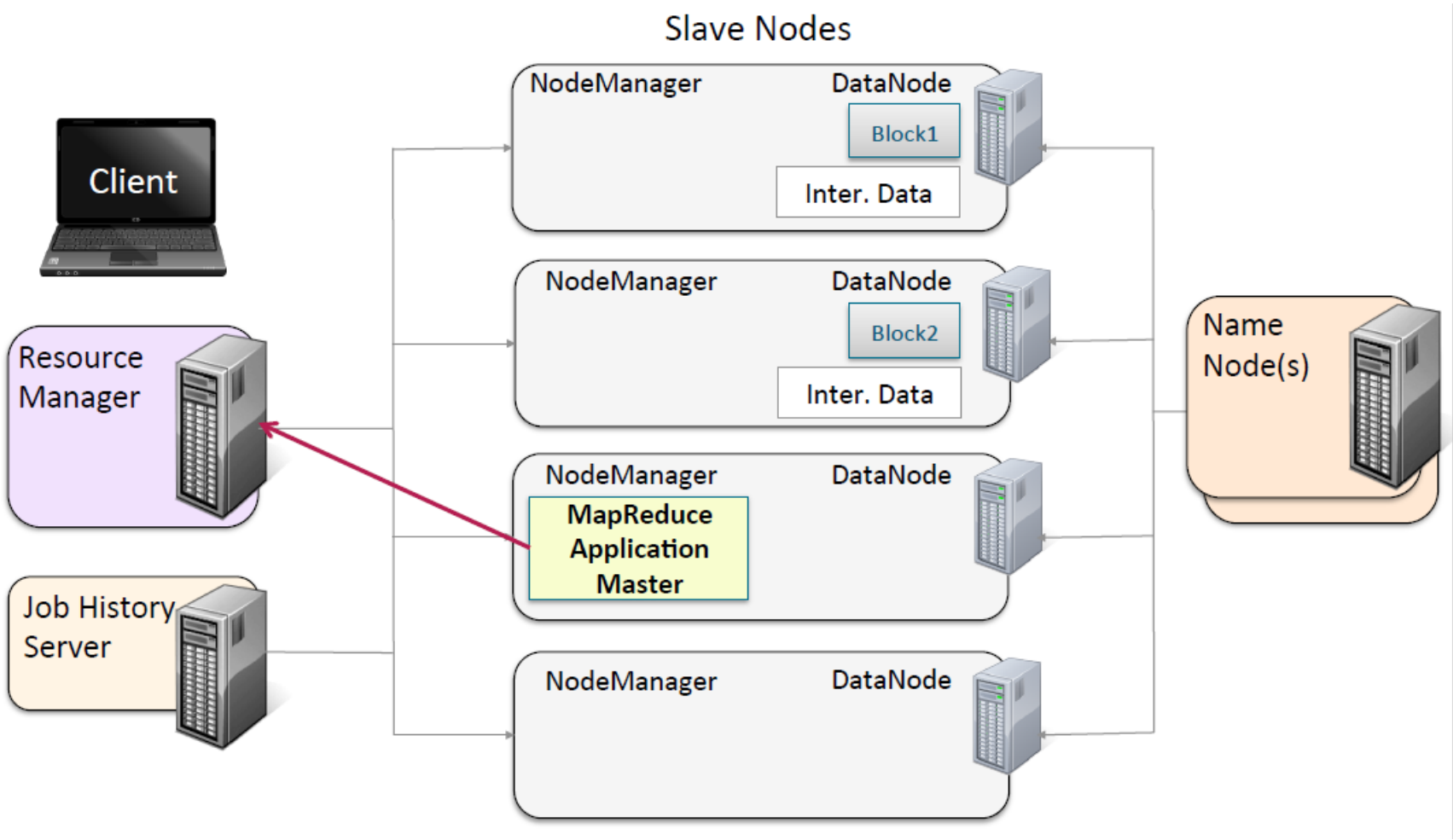
Running a Job on a YARN Cluster: Map Task Launch



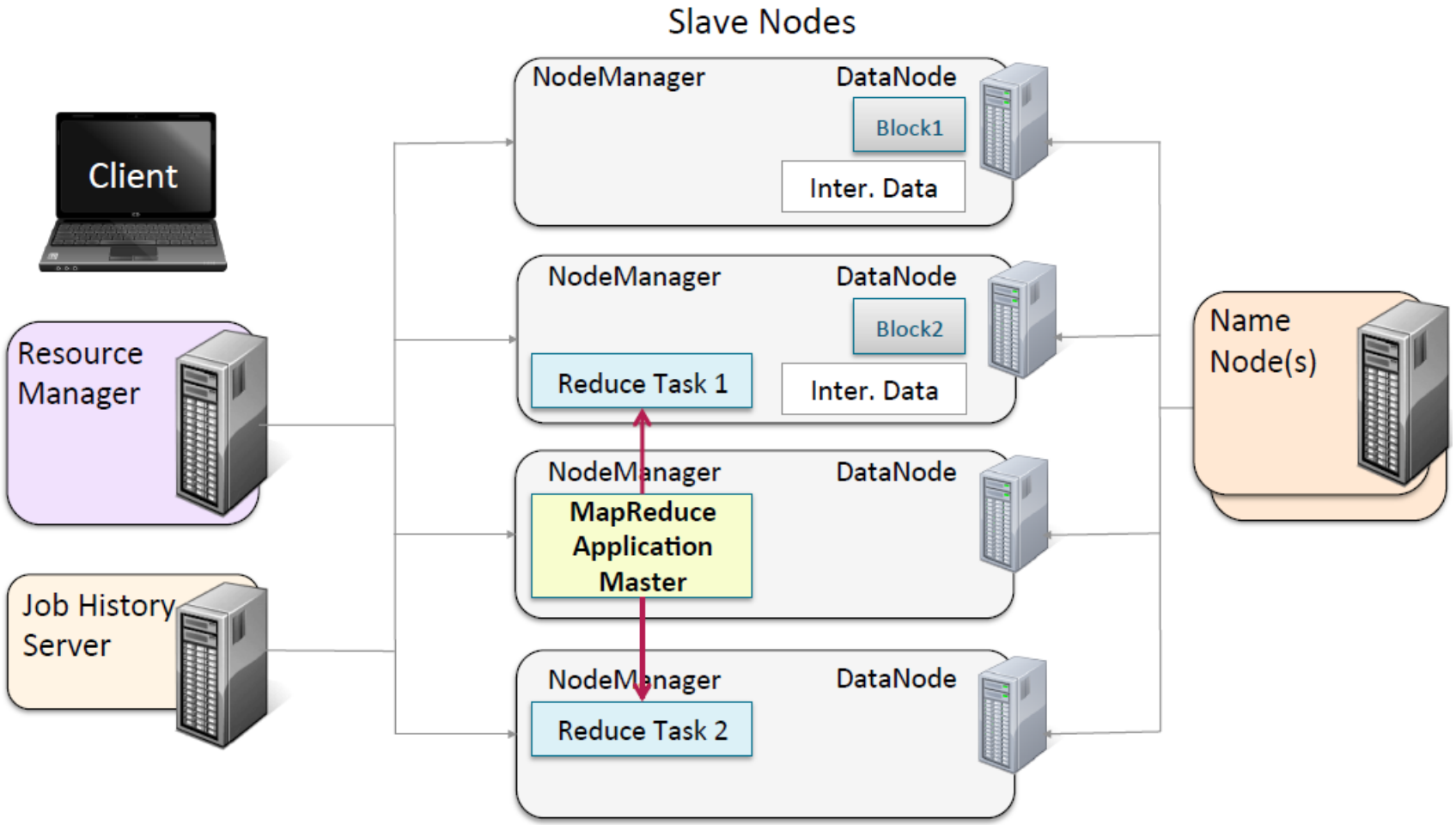
Running a Job on a YARN Cluster:Intermediate Data



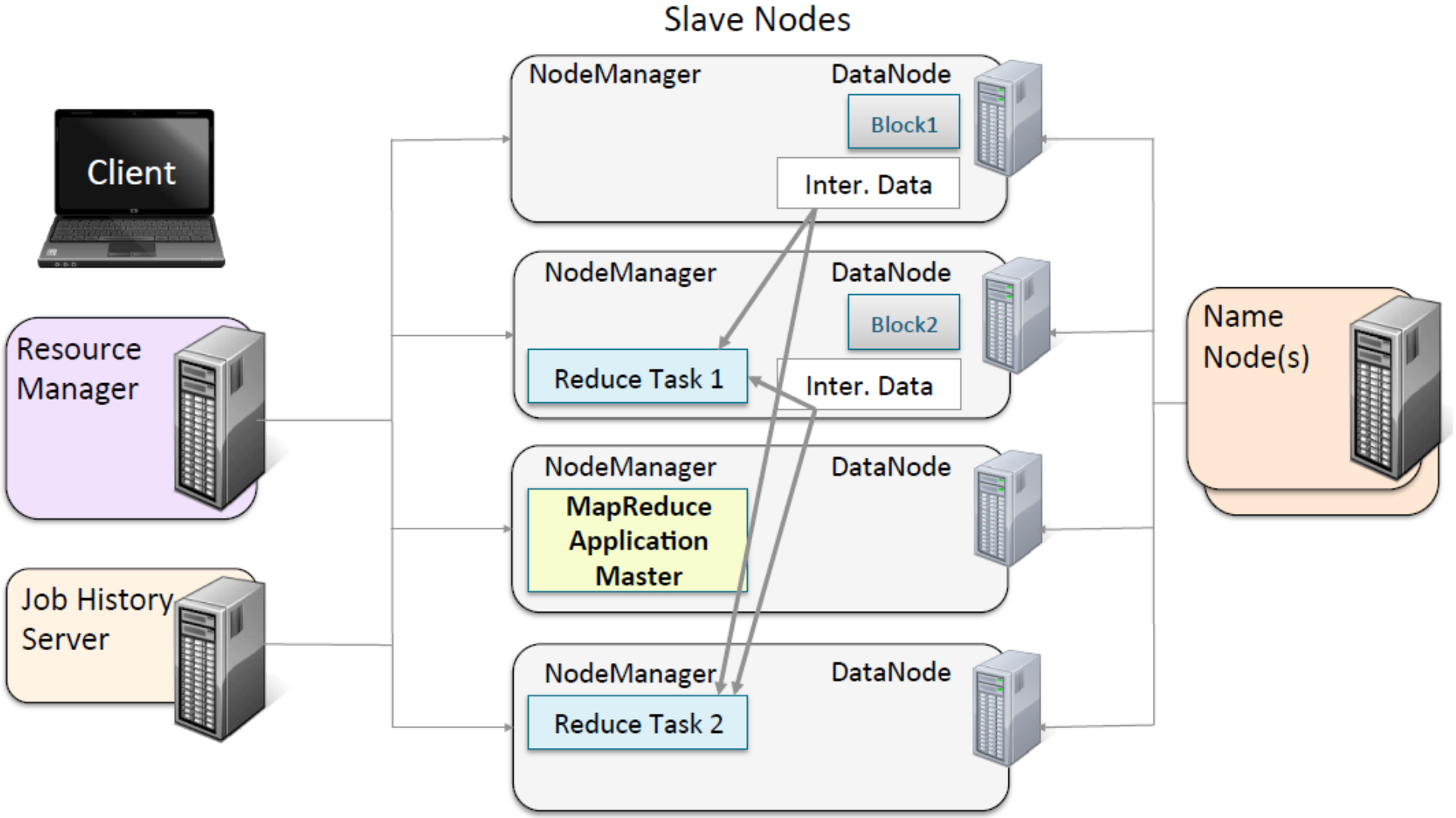
Running a Job on a YARN Cluster:Reduce Task Resource Request



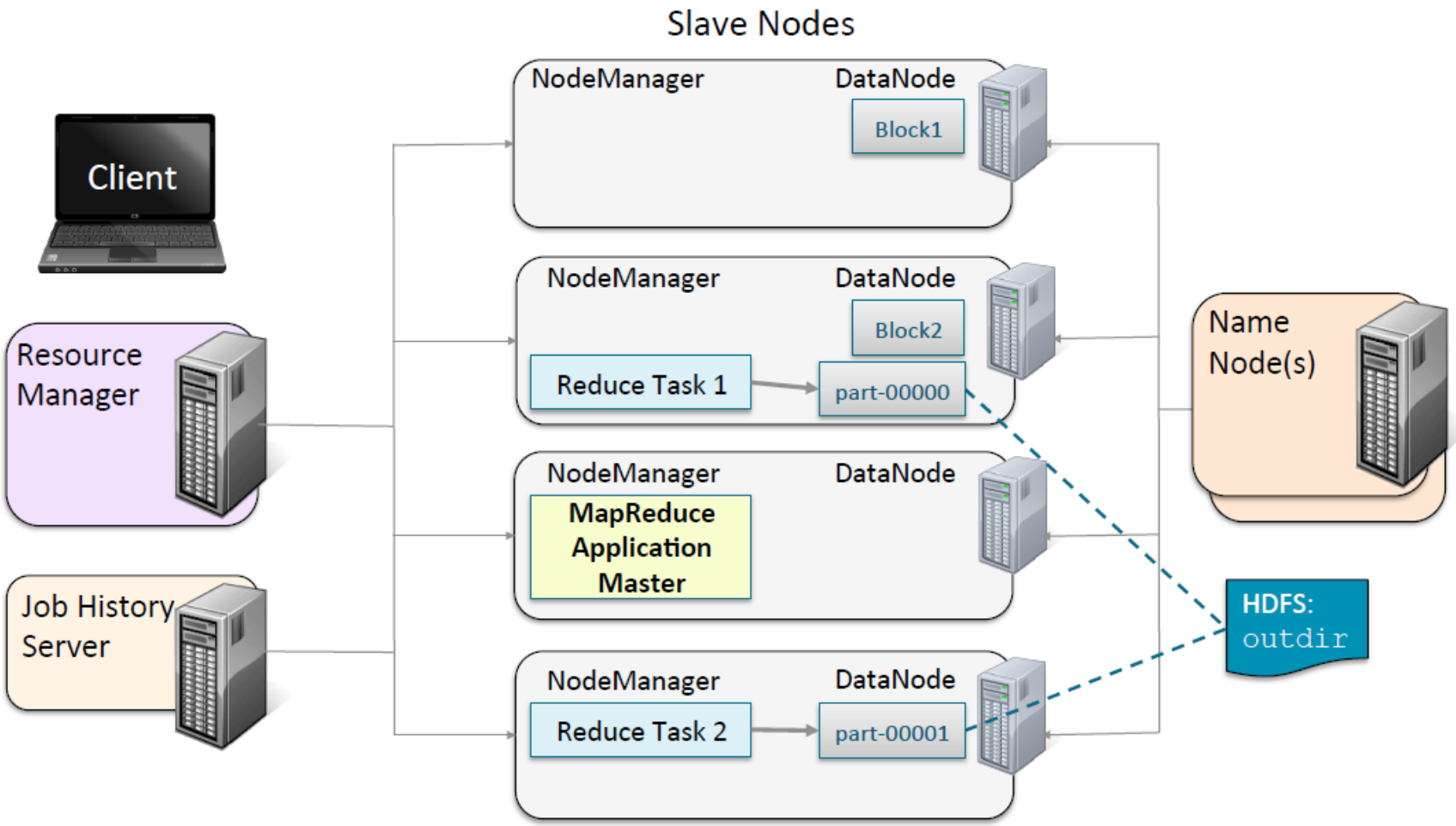
Running a Job on a YARN Cluster:Reduce Task Launch



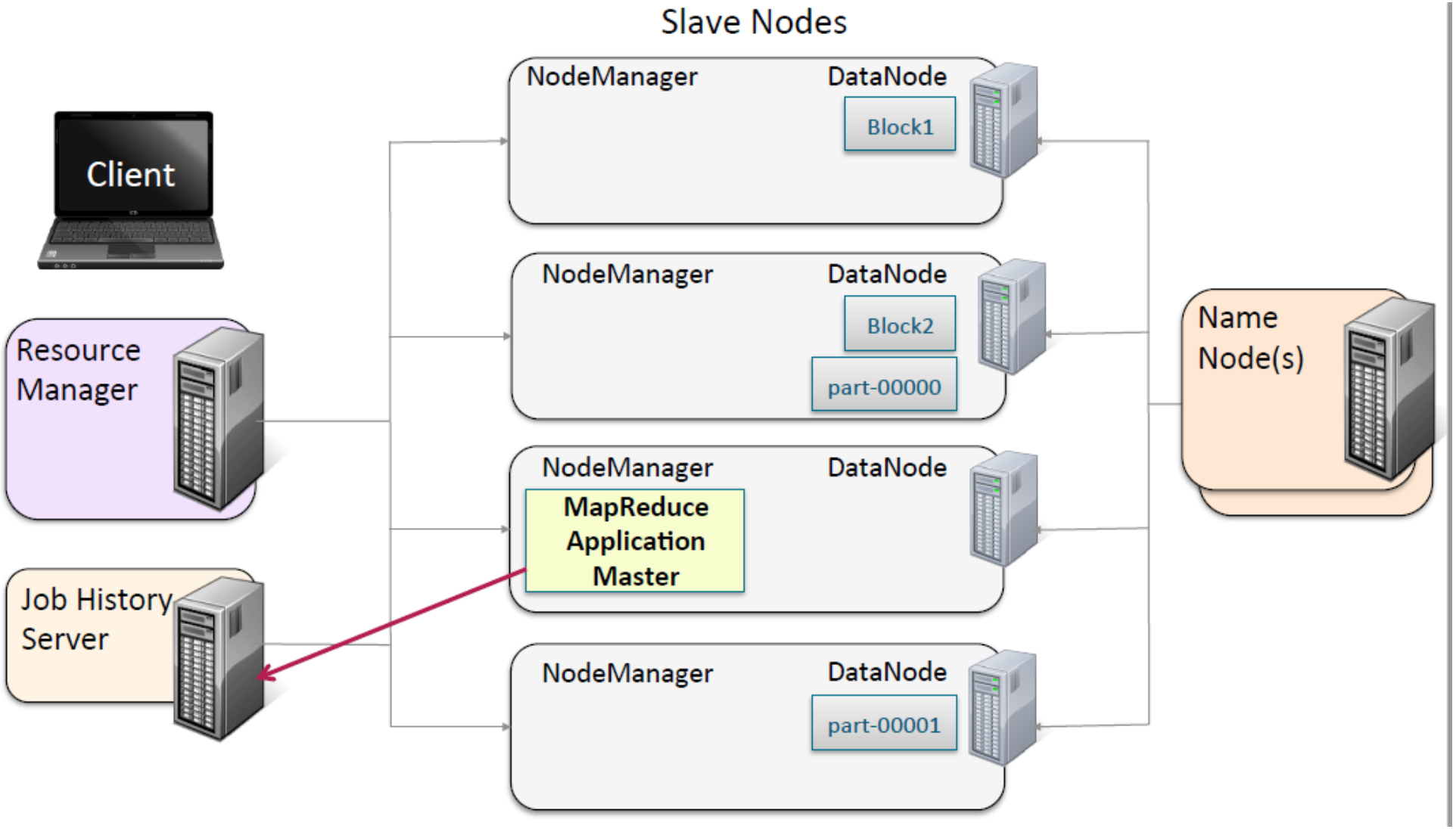
Running a Job on a YARN Cluster:Shuffle and Sort



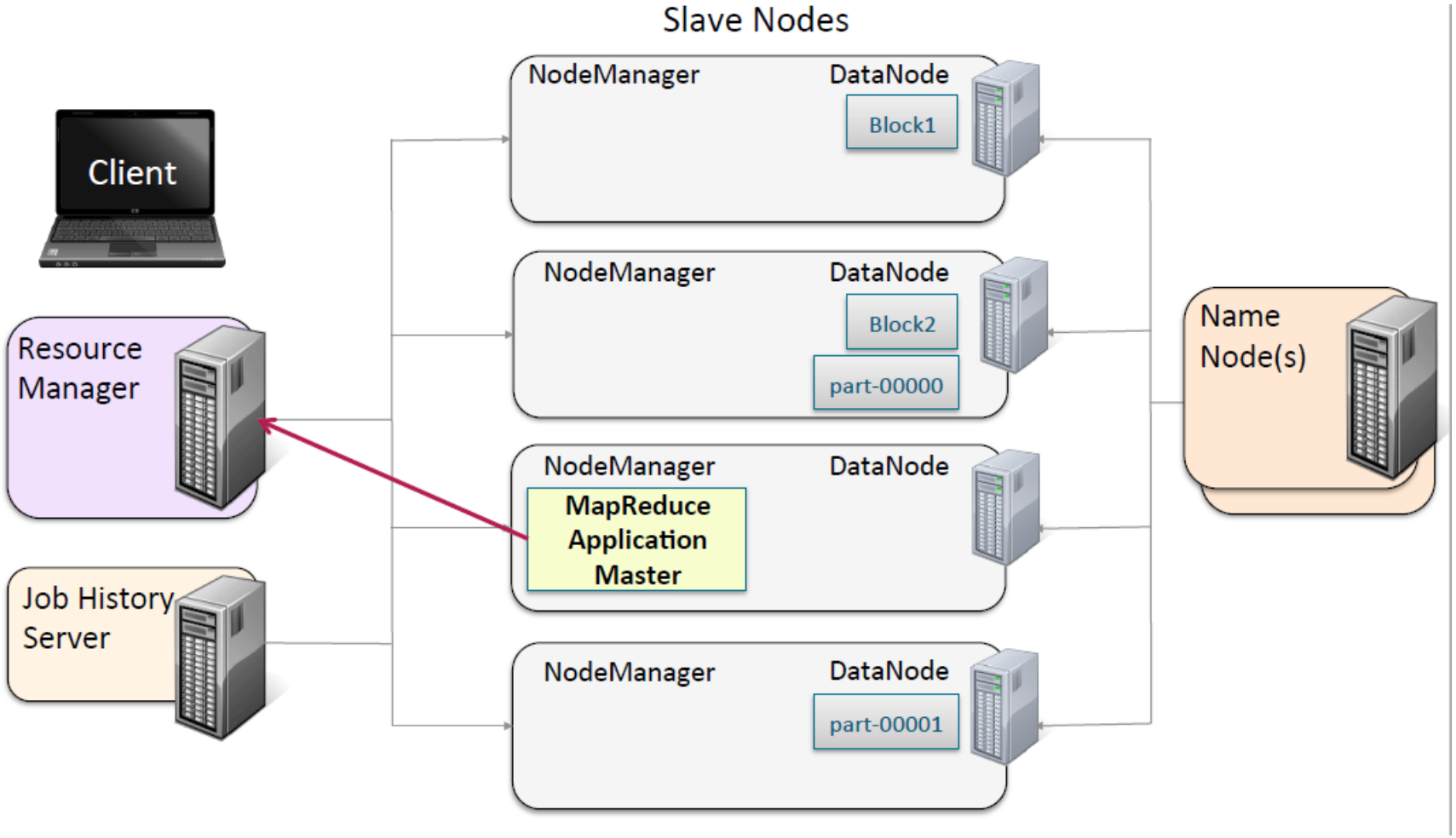
Running a Job on a YARN Cluster:Job Output



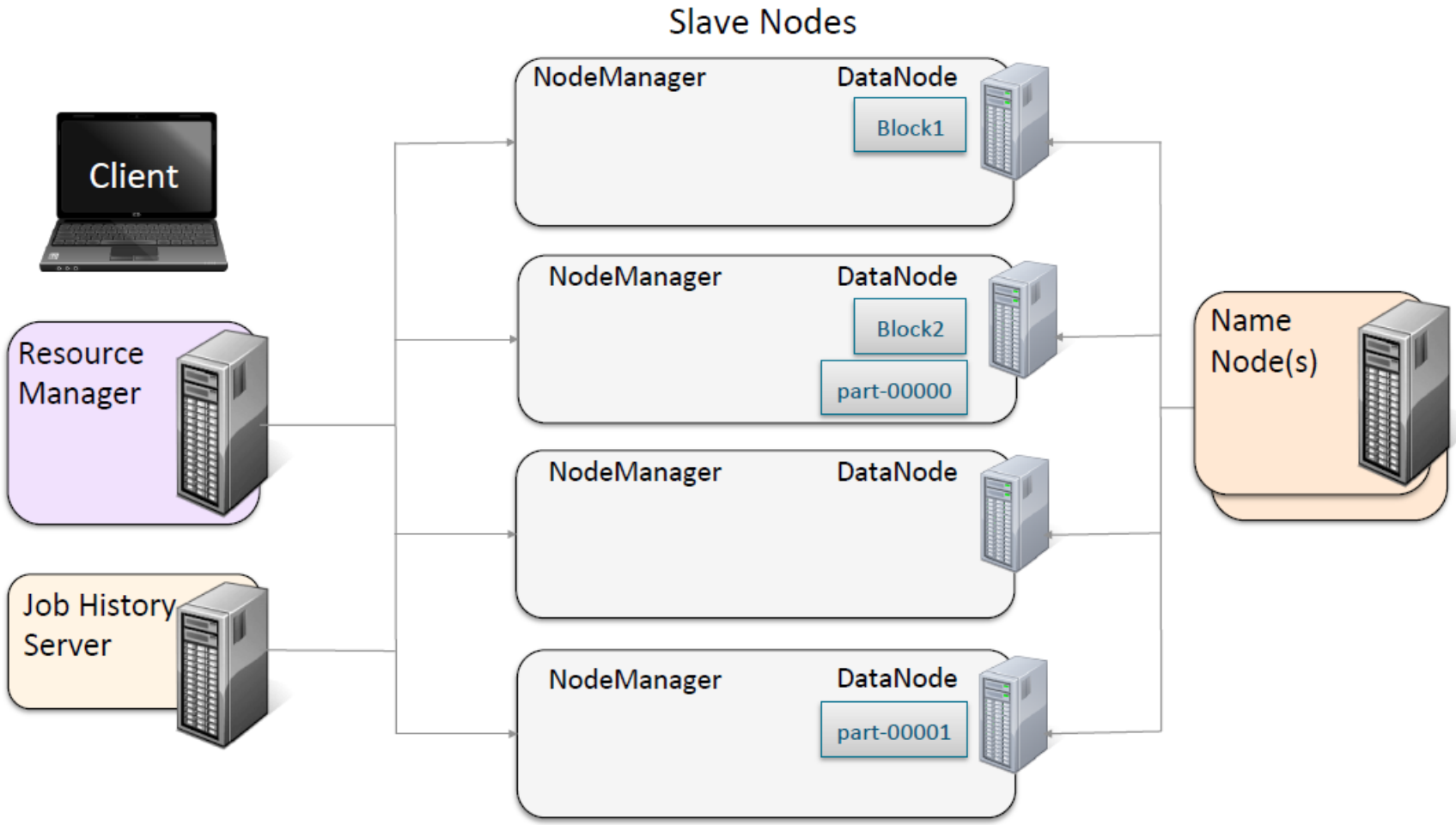
Running a Job on a YARN Cluster:Job History



Running a Job on a YARN Cluster:Job Completion



Running a Job on a YARN Cluster:Resources Released



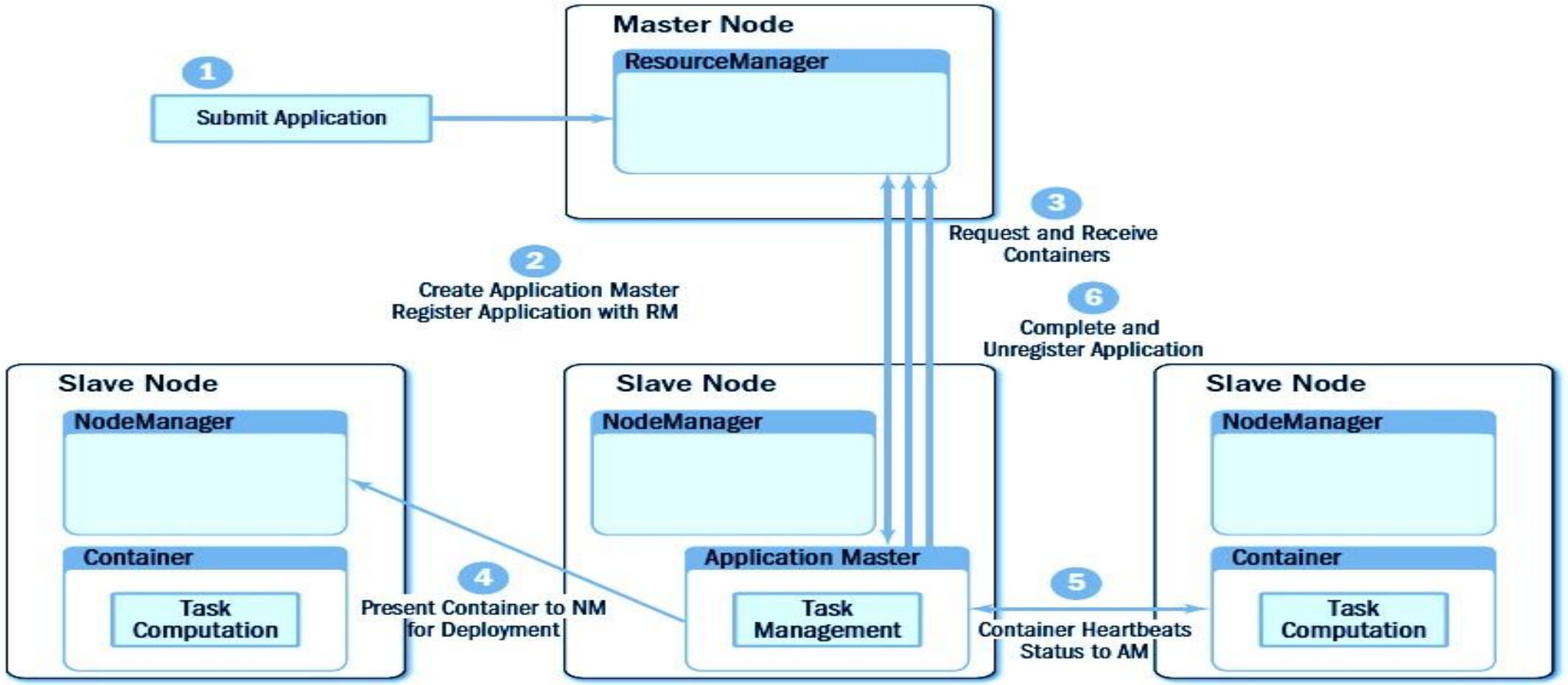
YARN Resource Manager

- Manages"nodes“
  - Tracks heartbeats from NodeManagers
  - Handles ApplicationMasters (e.g. MapReduce) requests for resources
  - Deallocates nodes when the tasks on them expire.
- Manages cluster/level security"

YARN Node Manager

- Communicates with the ResourceManager.
- Registers and provides info on node resources.
- Sends heartbeats and task status to Resource Manager.
- Launches Application Masters on request from the Resource Manager.
- Launches tasks on request from Application Masters.
- Monitors resource usage by tasks; kills runaway processes.
- Aggregates logs for an application and saves them to HDFS.
- Maintains node level security.

Brief on YARN workflow



http://www.bigdataqubes.com/posts/yarnarchitecture.php

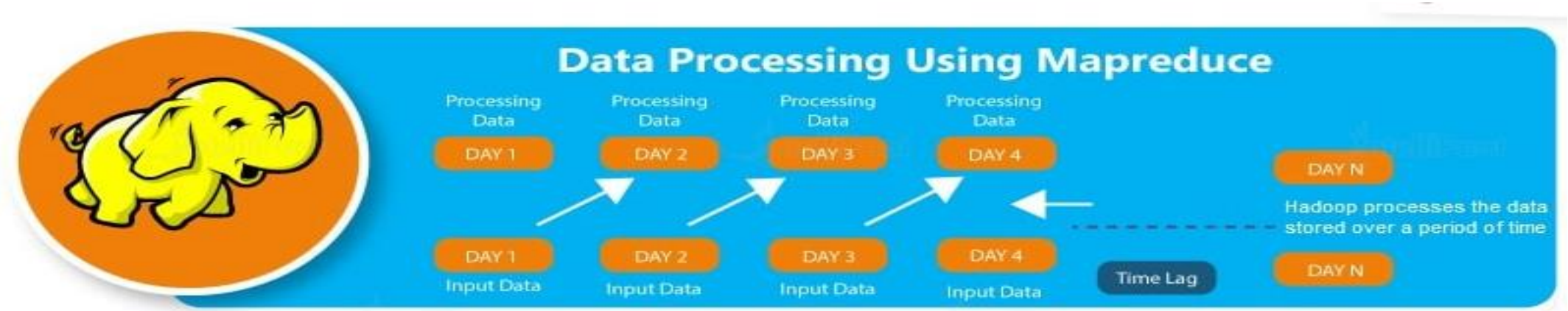
Review Questions 4

- What are the functions of YARN ?
- What are the YARN daemons and what are their functions ?
- Explain how MapReduce, YARN and HDFS work together. Diagrams are not necessary for the explanation, but the sequence of handshakes between the components should be clearly explained.

Hadoop vs Spark

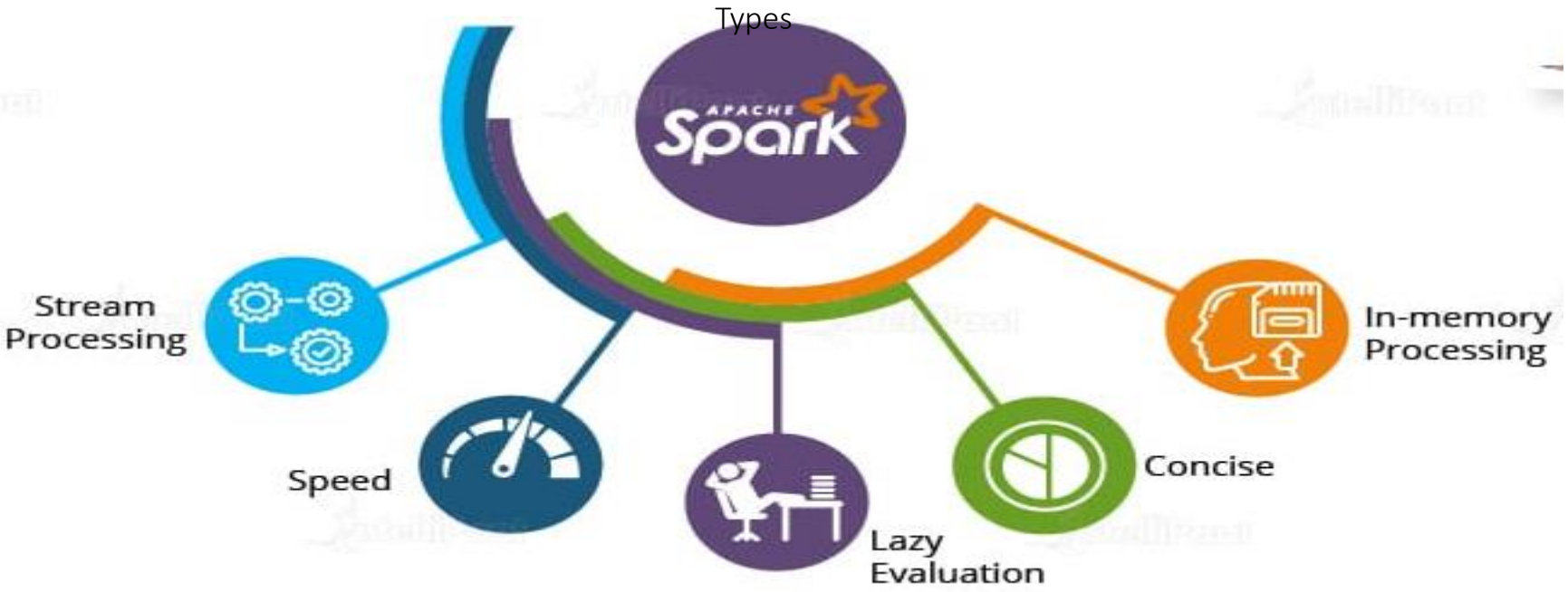
Draw backs of Hadoop

- **Low Processing Speed**
- **Batch Processing**
- **No Data Pipelining**
- **Not Easy to Use**
- **Latency**
- **Lengthy Line of Code**

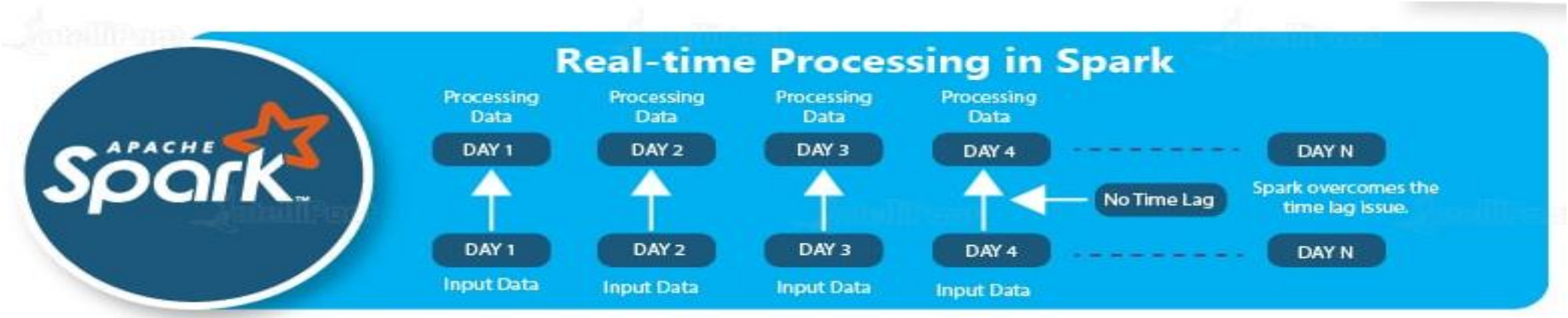


How Spark Is Better than Hadoop?

- In-memory Processing
- Stream Processing
- Less Latency
- Lazy Evaluation
- Less Lines of Code



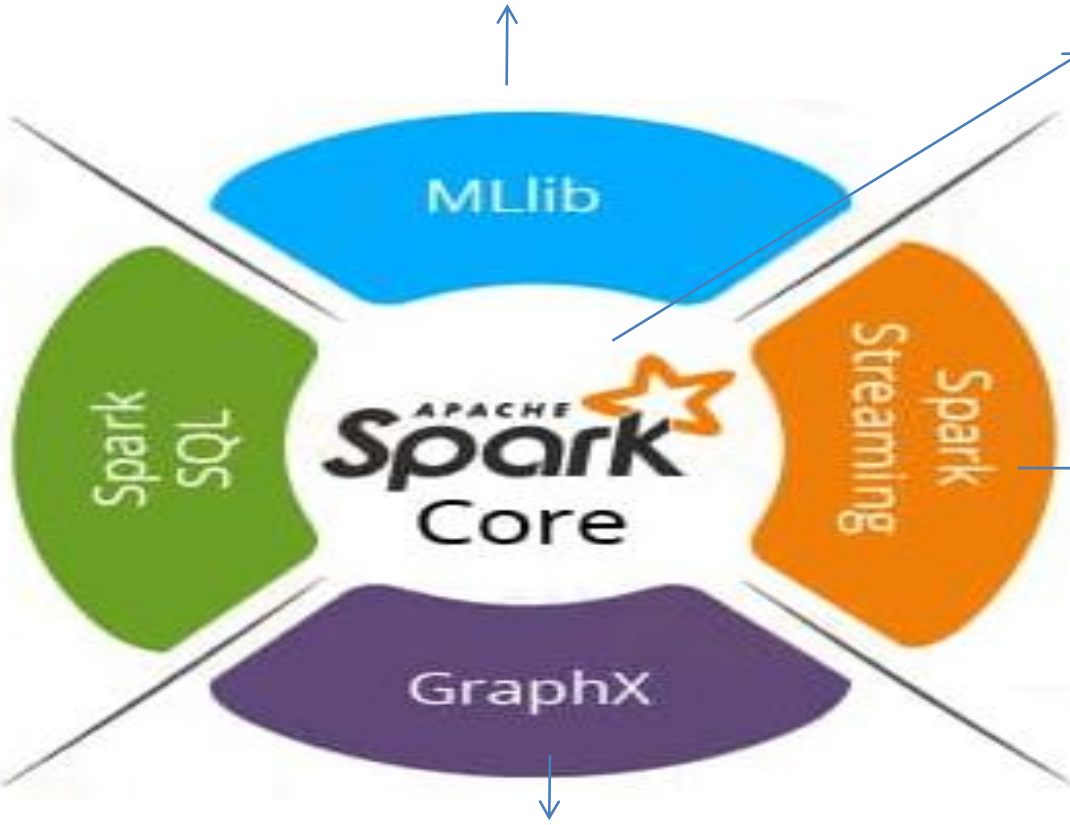
Real time processing in Spark



Components of Spark

Apache Spark comes up with a library containing common Machine Learning (ML) services called MLli

Basic building block ,Job scheduling, performing various memory operations, fault tolerance,

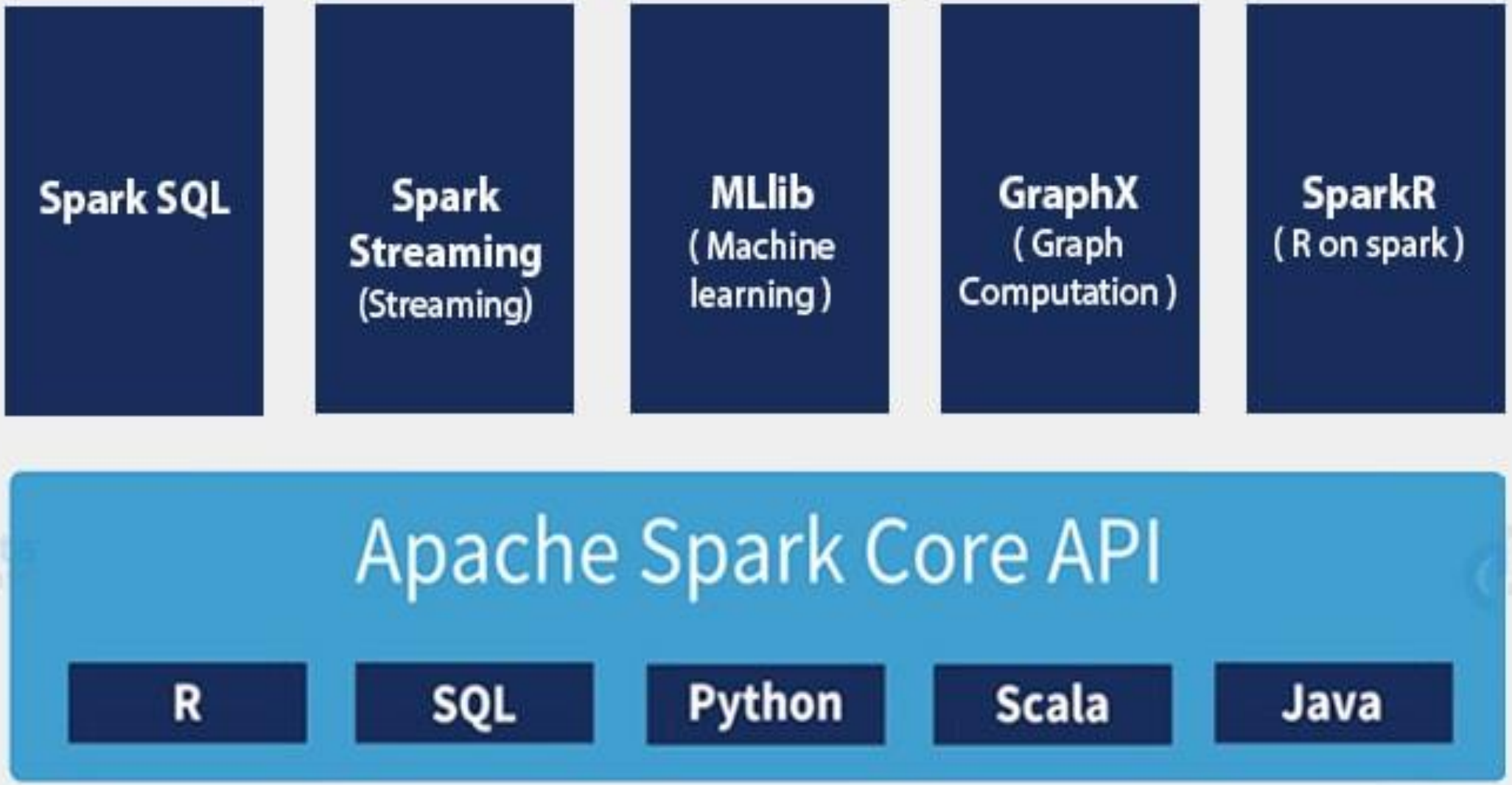


Spark SQL allows programmers to combine SQL queries with **programmable changes**

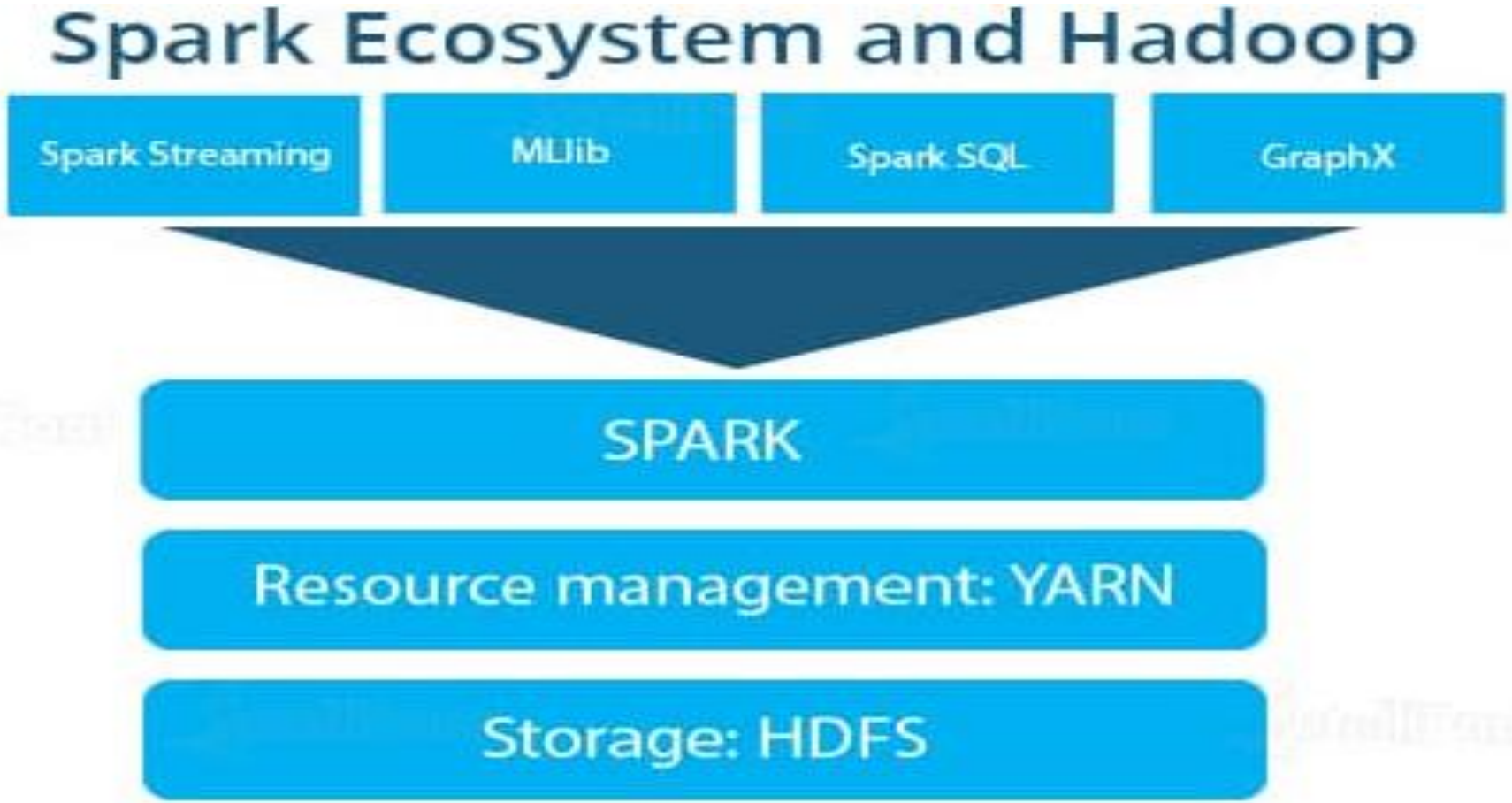
Spark Streaming processes live streaming of data

Apache Spark's library for enhancing graphs and enabling graph-parallel computation.

APACHE SPARK ECOSYSTEM



Spark Ecosystem and Hadoop



Why Use Hadoop and Spark Together?

- Spark can run on Hadoop
- Spark's MLlib components provide capabilities that are not easily achieved by Hadoop's MapReduce
- Hadoop provides enhanced security, which is a critical component for production workloads
- Spark does not have its own distributed file system. By combining Spark with Hadoop, you can make use of various Hadoop capabilities. For example, resources are managed via **YARN Resource Manager**. You can integrate Hadoop with Spark to perform Cluster Administration and Data Management.

Applications of Apache Spark

Netflix

E-bay, Alibaba

MyFitness pal

Review Exercises 5

1. What shortcomings of Hadoop motivated the development of Spark. How were the shortcomings addressed by Spark ?
2. Is Spark completely different from Hadoop ? If not, what is same between the two and what is different ?
3. Spark clusters may be more expensive than Hadoop. Explain why.
4. Find out about Spark use cases. (You can refer to https://www.dezyre.com/article/top-5-apache-spark-use-cases/271)

Scala Introduction

Introduction to Scala

- Scala overview. Motivation for Scala.
- The REPL shell.
- Values and Variables, Types, Operators
- Control structures, Functions
- Tuples
- Collections: Mappings, Iterators, Comprehensions.

- Classes, Objects.
- Pattern matching.
- Functional Programming.

We will be learning only a small subset of Scala constructs as required for the spark programming assignments. Some Scala constructs can be very sophisticated. We will learn mostly the simple use cases.

- Short for **Sca**lable **La**nguage.
- Created by Martin Odersky.
- Is a fusion of Object Oriented and Functional Programming.
- Scala is statically typed like Java, but the programmer has to supply type information in only a few places; Scala can infer type information.
- Scala programs run on JVM.
- Designed for concurrent hence scalable applications.
- Concise syntax.
- Has a REPL (Read-Eval-Print-Loop) shell.

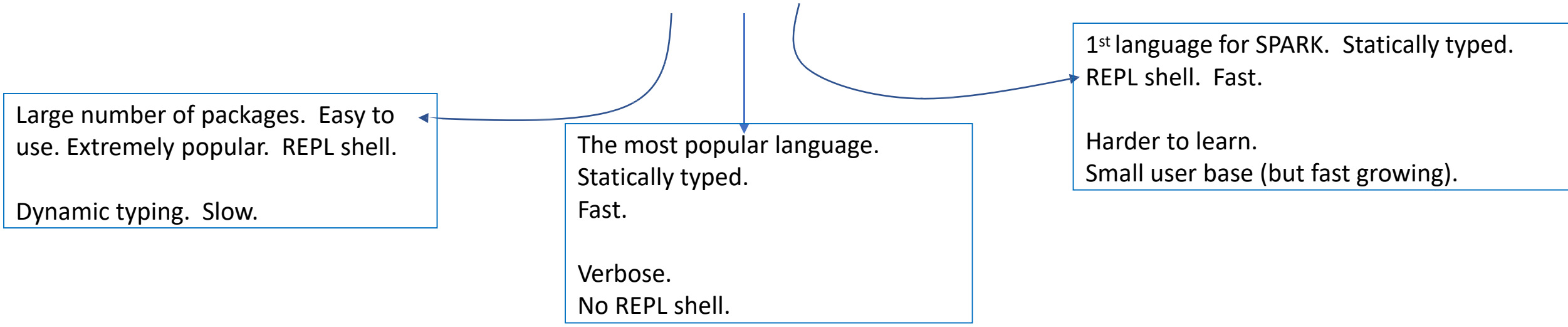
Top Programming Languages for Big Data

Apache Spark written in Scala

Apache Spark Programming

Written in Scala.

Can be programmed in Python, Java, Scala. (Also R)



Sbt(Simple Build Tool)

- sbt is an open source build tool for building scala projects.
  - It is interactive, and supports a scala shell.

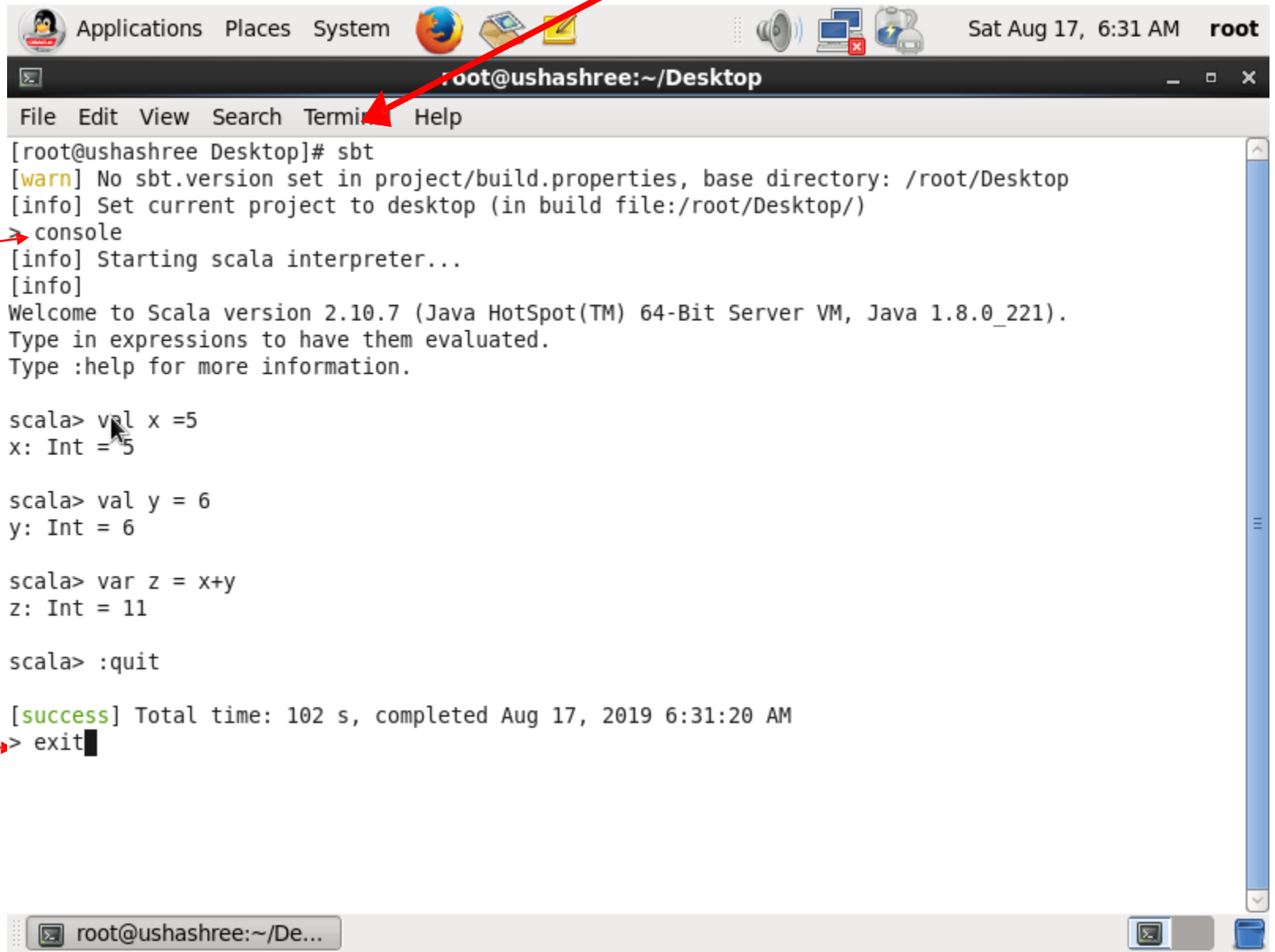
The Scala REPL Shell through sbt

starting a scala shell through sbt.

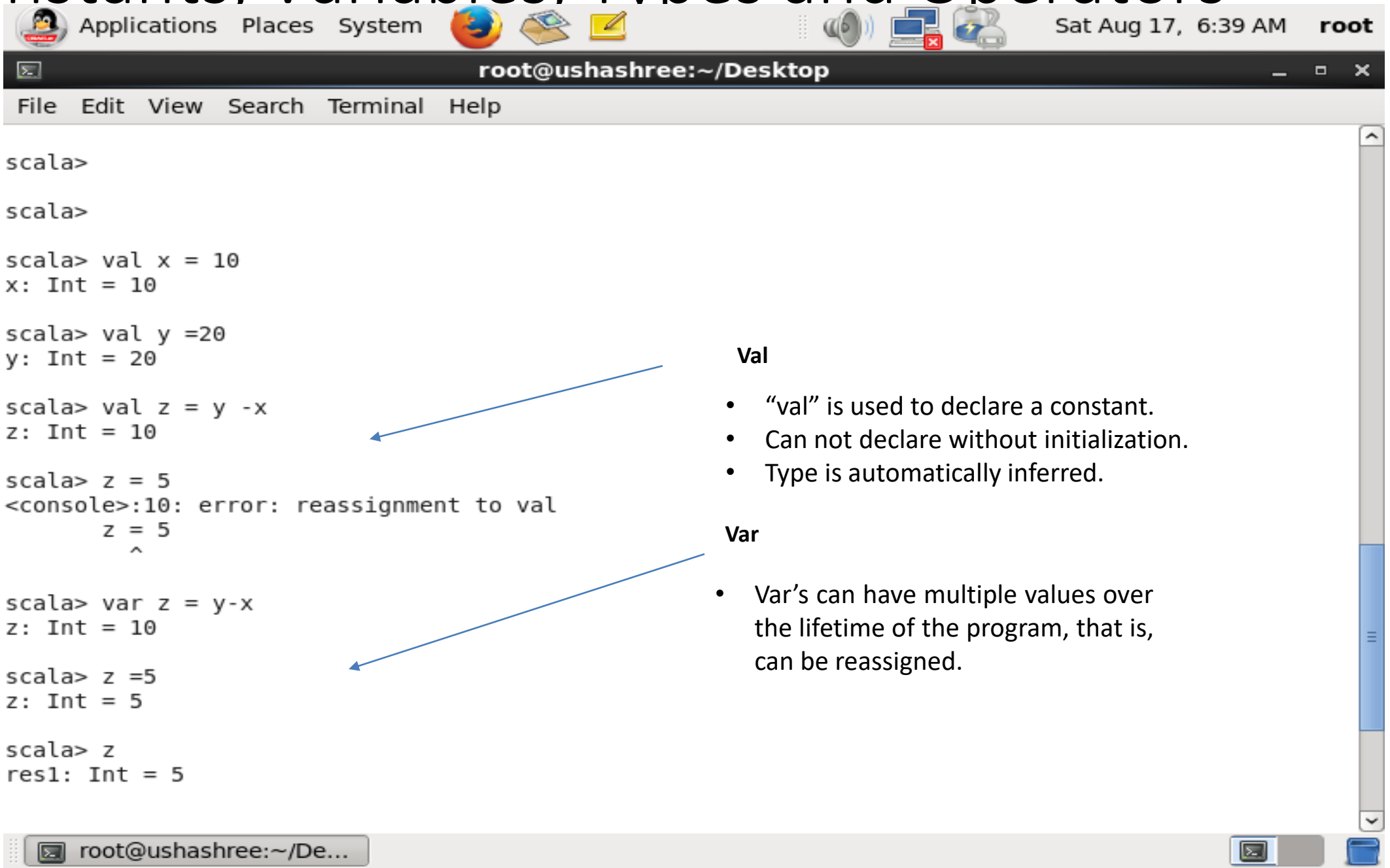
Typing "console" in sbt shell starts scala REPL shell.

":quit" quits scala shell.

"exit" exits the sbt shell.



Constants, Variables, Types and Operators

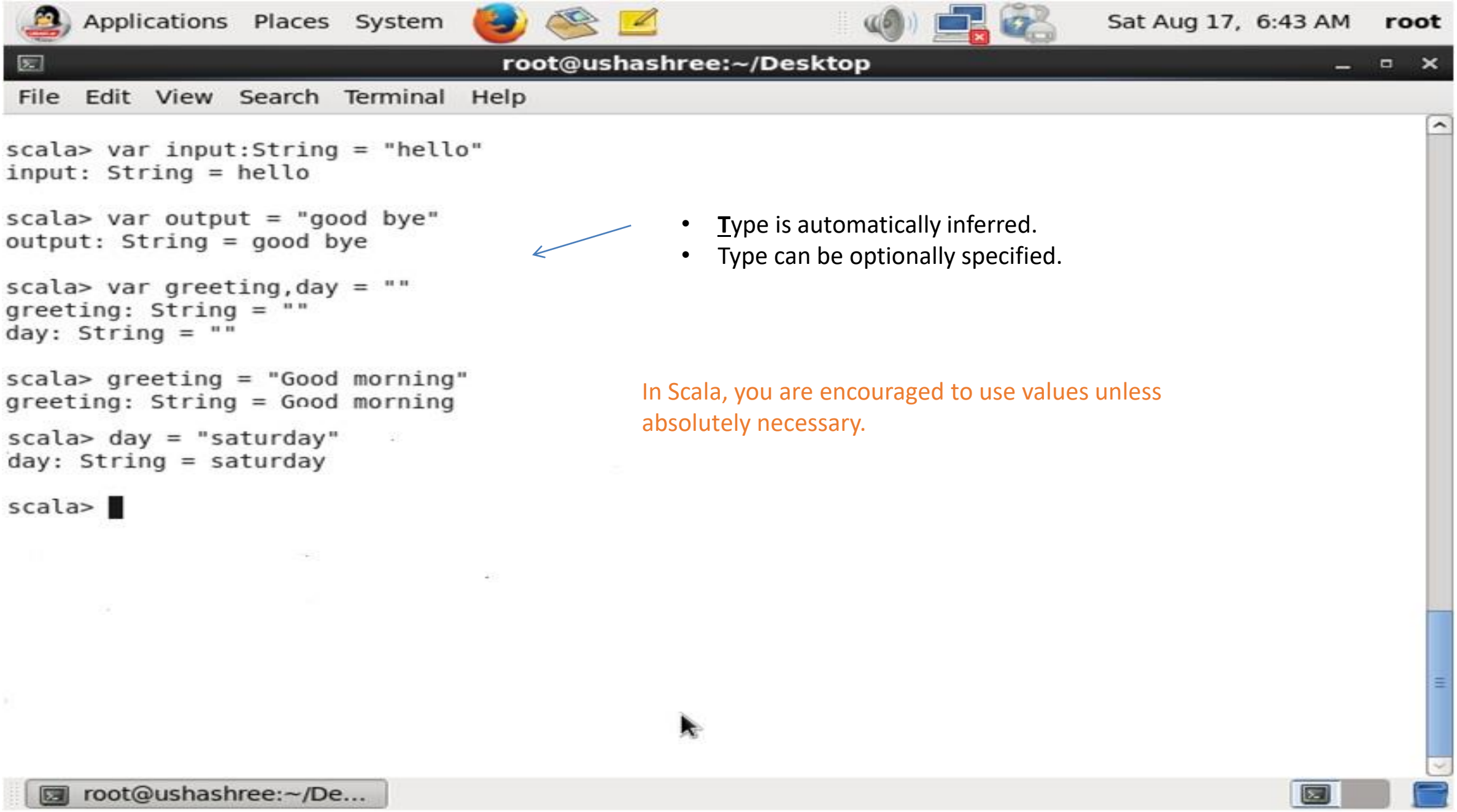


Val

- "val" is used to declare a constant.
- Can not declare without initialization.
- Type is automatically inferred.

Var

- Var's can have multiple values over the lifetime of the program, that is, can be reassigned.



- Type is automatically inferred.
- Type can be optionally specified.

In Scala, you are encouraged to use values unless absolutely necessary.

Types

- Commonly used types: Byte, Char, Short, Int, Long, Float, Double, Boolean, Char, String.
- For arbitrary precision: BigInt, BigDecimal.
- The “Any” type. When variable type is unknown. This should be used on an exception basis.
- The “Unit” type. Value is (), equivalent to void in Java.

- All types are actually classes. Hence all operators for the type are actually methods.
  - e.g. a + b is actually a shothand for a.+(b)

```
scala> val y =10
y: Int = 10

scala> val x =20
x: Int = 20

scala> val z = x.+(y)
z: Int = 30

scala> val w = x+ y
w: Int = 30
```

```
scala> "hello".union("world")
res2: String = helloworld

scala> "hello".intersect("world")
res3: String = lo

scala> "excited".sorted
res4: String = cdeeitx

scala> val s = "education"
s: String = education

scala> s(6)
res5: Char = i
```

- Additional operators/ methods defined for Strings types.
- As in Java and C++, e.g. for numbers: +, -, /, * etc.
- No ++. Instead use += 1, i.e. n += 1.
- Methods do not need round brackets or “()” if there are no parameters
- The ith character of an array is accessed as A(i).

- Math package has many math functions such as sqrt, min etc.

.

```
scala> import scala.math._
import scala.math._

scala> sqrt(2)
res8: Double = 1.4142135623730951

scala> min(x,y)
res9: Double = 1.4142135623730951
```

Value of Expressions and Statements

```
scala> 3*4
res10: Int = 12

scala> var x = 3*4
x: Int = 12

scala> var z = x = 5 *4
z: Unit = ()

scala> x
res11: Int = 20
```

3 * 4 is an expression. Has the value 12.

var x = 3 * 4 is a statement. The action is "12" being assigned to x.
The value of x is printed by the scala shell(not the value of the statement).

The value of the statement is assigned to z. Note that the shell prints the value of z as Unit() which in Scala is same as void.

Review Questions and Practise Exercises 1

- In the Scala REPL, type “3.” and then hit the TAB key. What do you see ? Note: do not ignore the “.” after the 3.
- Do the same as above by typing “Hello.” followed by the TAB key. Note: do not ignore the “.” after “Hello”. Repeat by typing “Hello.s” and then applying the TAB key.
- In the Scala REPL, compute the square root of 3, and then square that value. What do you observe and how can you explain your observation ?
- Scala lets you multiply a string with a number—try out "crazy" * 3 in the REPL. What does this operation do?
- How do you get the first character of a string in Scala? The last character?
- What is Lazy Initialization.

Control Structures and Functions

If - else

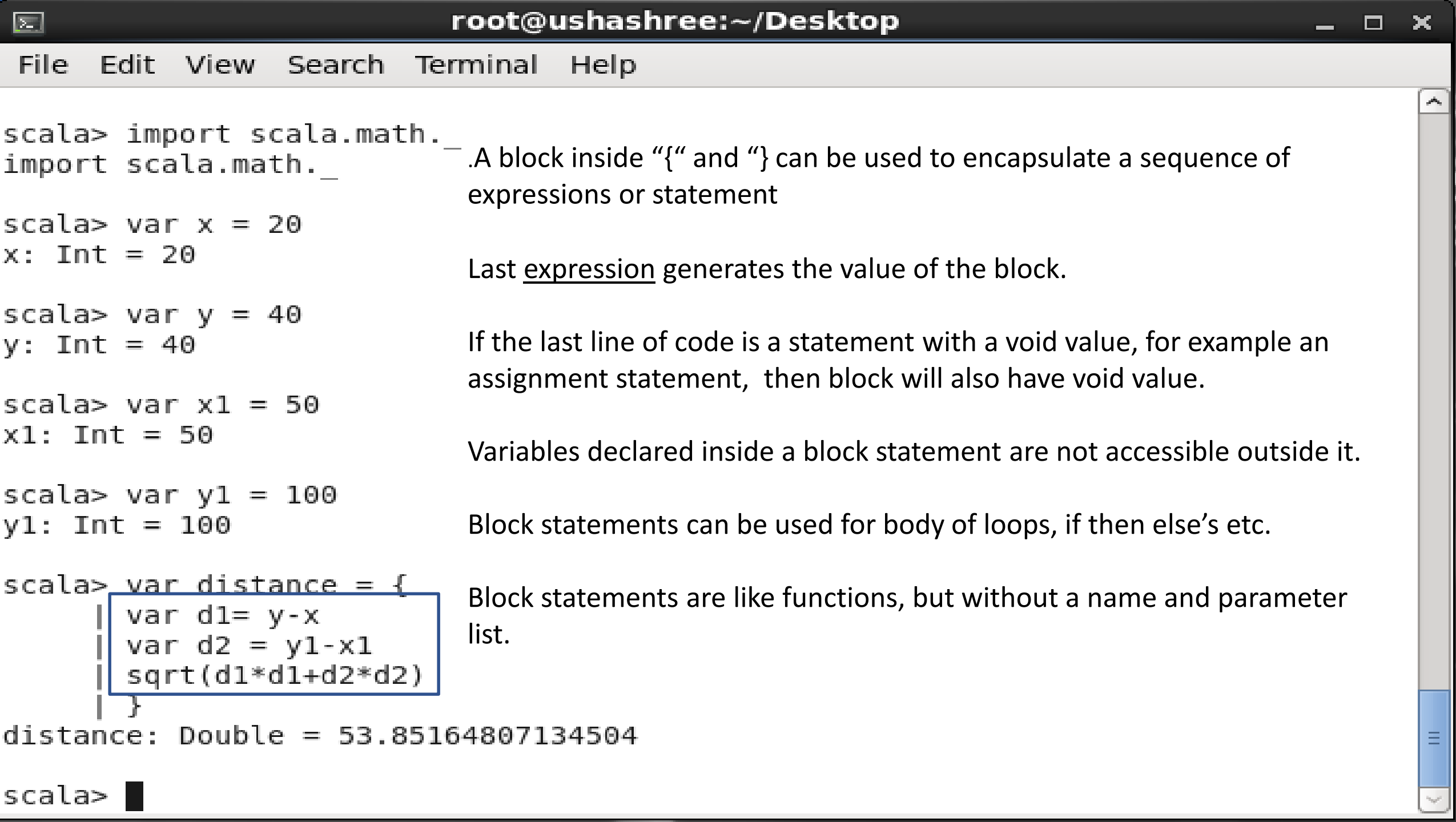
```
scala> var x = 5
x: Int = 5

scala> var z :Int  = _
z: Int = 0

scala> if(x>2) z = -1 else z =1

scala> val zz = if(x>10) -1 else 1
zz: Int = 1

scala> val zzz = if(x>2) z = 1 else -1
zzz: Unit = ()
```

variable z is initialized with a wildcard variable, which is set to default value of type.

Similar syntax as Java.

If statement can have a value if the then part and the else part have a value.

In this case if statement has the void value, since assignment statements in the then and else part have a Unit() or void value.


root@ushashree:~/Desktop
File Edit View Search Terminal Help
scala> import scala.math._
import scala.math._
scala> var x = 20
x: Int = 20
scala> var y = 40
y: Int = 40
scala> var x1 = 50
x1: Int = 50
scala> var y1 = 100
y1: Int = 100
scala> var distance = {
| var d1= y-x
| var d2 = y1-x1
| sqrt(d1*d1+d2*d2)
| }
distance: Double = 53.85164807134504
scala>
.A block inside “{“ and “} can be used to encapsulate a sequence of expressions or statement
Last expression generates the value of the block.
If the last line of code is a statement with a void value, for example an assignment statement, then block will also have void value.
Variables declared inside a block statement are not accessible outside it.
Block statements can be used for body of loops, if then else’s etc.
Block statements are like functions, but without a name and parameter list.

While and For Loops

```
scala> for(i<- 1 to 4)println(i)
1
2
3
4
```

- Syntax of for loop: for (I <- expr) body
- Sets “I” to every value of expression.

```
scala> for(i<- 1 until 4)println(i)
1
2
3
```

sets loop variable “ch” to every item of expression.

```
scala> var n = 5
n: Int = 5

scala> var r =1
r: Int = 1

scala> while(n>0)
     | {
     | r = r* n
     | println(r)
     | n -= 1
     | }
5
20
60
120
120
```

```
scala> var upcase = ""
upcase: String = ""

scala> for(ch <- "hello") upcase = upcase+ ch.toUpper

scala> println(upcase)
HELLO
```

Functions and Procedures

Functions

note “=“ after argument list and before body.

```
scala> def fact(n: Int)= {
     | var r =1
     | for ( i <- 1 to n) r = r * i
     | r
     | }
fact: (n: Int)Int

scala> fact(3)
res7: Int = 6

scala> fact(5)
res8: Int = 120
```

- can have variable number of arguments
- return type determined from type of last expression.
- “return” keyword not usually used.

```
scala> def recFact(n: Int): Int = if(n <= 0) 1 else n * recFact(n-1)
recFact: (n: Int)Int

scala> recFact(5)
res9: Int = 120

scala> def recFact(n: Int) = if(n <= 0) 1 else n * recFact(n-1)
<console>:9: error: recursive method recFact needs result type
       def recFact(n: Int) = if(n <= 0) 1 else n * recFact(n-1)
                                                   ^
```

return type has to be specified for recursive functions.

•Pure Functions:

The function always evaluates to the same result value given the same argument values. It cannot depend on any hidden state or value.

Evaluation of the result does not cause any semantically observable side effects.

Does not change value of parameters passed. Does not do any action (e.g. even a println) other than return a value. Scala encourages use of pure function.

The map method

```
scala> val Subject  = List("BDA","CC","CD","EDM")
Subject: List[String] = List(BDA, CC, CD, EDM)

scala> val subject = Subject.map(_.toLowerCase)
subject: List[String] = List(bda, cc, cd, edm)

scala> val subject1 = Subject.map(x=>x.toLowerCase)
subject1: List[String] = List(bda, cc, cd, edm)

scala> Subject
res11: List[String] = List(BDA, CC, CD, EDM)
```


Note: Subject remains unchanged.

- map applies a function to each element of a collection.
- We are using a List collection here as an illustration, but map can be applied to an Array, Map etc.
- map is often preferred to a for loop.

Functions as Parameters

```
scala> def recFact(n: Int): Int = if(n <= 0) 1 else n * recFact(n-1)
recFact: (n: Int)Int


scala> Array(4,6,3).map(recFact)
res12: Array[Int] = Array(24, 720, 6)
```


"map" is a method or function. Function recFact is passed as a parameter to the map method.

We say functions are first class citizens of a language if they can be passed as parameters to other functions.
Allows composition of functions, e.g. f(g(x)) much like mathematical functions.

Anonymous Functions

```
scala> def triple(x : Double)= 3 * x
triple: (x: Double)Double

scala> Array(2.1,3.7,5.5).map(triple)
res23: Array[Double] = Array(6.300000000000001, 11.100000000000001, 16.5)

scala> Array(2.1,3.7,5.5).map((x: Double)=> 2 *x)
res24: Array[Double] = Array(4.2, 7.4, 11.0)

scala> Array(2,3,5).map(
     | (x : Int) => {
     | var z = 2 * x
     | z +1
     | }
     | )
res25: Array[Int] = Array(5, 7, 11)
```

parameter on the LHS of "=>". function return value on the RHS.

(x:Double) => 3 * x, is an anonymous function: does not have a name.

Anonymous function with a block statement.

Procedure

s

```
scala> def box (s: String) {
     | val bor = "-" *s.length + "--\n"
     | println(bor + "|" + s + "|\n" + bor)
     | }
box: (s: String)Unit

scala> box("BDA")
-----
|BDA|
-----
```

- if no “=“ before body, then it is a procedure.
- Procedure returns returns Unit() or void type.
- Procedures are used for their side effect.

```
scala> def box (s: String = "HELLO" ) {
     | val bor = "-" *s.length + "--\n"
     | println(bor + "|" + s + "|\n" + bor)
     | }
box: (s: String)Unit

scala> box()
-------
|HELLO|
-------
```

Review Questions and Practise Exercises 2

1. Write a Scala equivalent for the Java loop for (int i = 10; i >= 0; i--) System.out.println(i);
2. Write a procedure countdown(n: Int) that prints the numbers from *n* to 0.
3. Write a for loop for computing the product of the Unicode codes of all letters in a string. For example, the product of the characters in "Hello" is 825152896.
4. Write a function that computes *x^n*, where *n* is an integer. Use the following recursive definition:
   x^n = *y^*2 if *n* is even and positive, where *y* = *x^(n* / 2).
   x^n = *x* * *x^(n* – 1) if *n* is odd and positive.
   *x^*0 = 1.
   x^n = 1 / *x^(–n)* if *n* is negative.
   Don’t use a return statement.

The Tuple

A Tuple is an aggregation of *n* elements, not necessarily of the same type.

```
scala> var x = (1,"one",1.1)
x: (Int, String, Double) = (1,one,1.1)

scala> x._1
res4: Int = 1

scala> x._1
res5: Int = 1

scala> val (first,second,third) = x
first: Int = 1
second: String = one
third: Double = 1.1
```

- Tuple elements are accessed using the ._ notation.
- Numbering starts from 1, not 0.

Accessing values using pattern matching.

Collections (Data Structures): Maps, Arrays and Lists

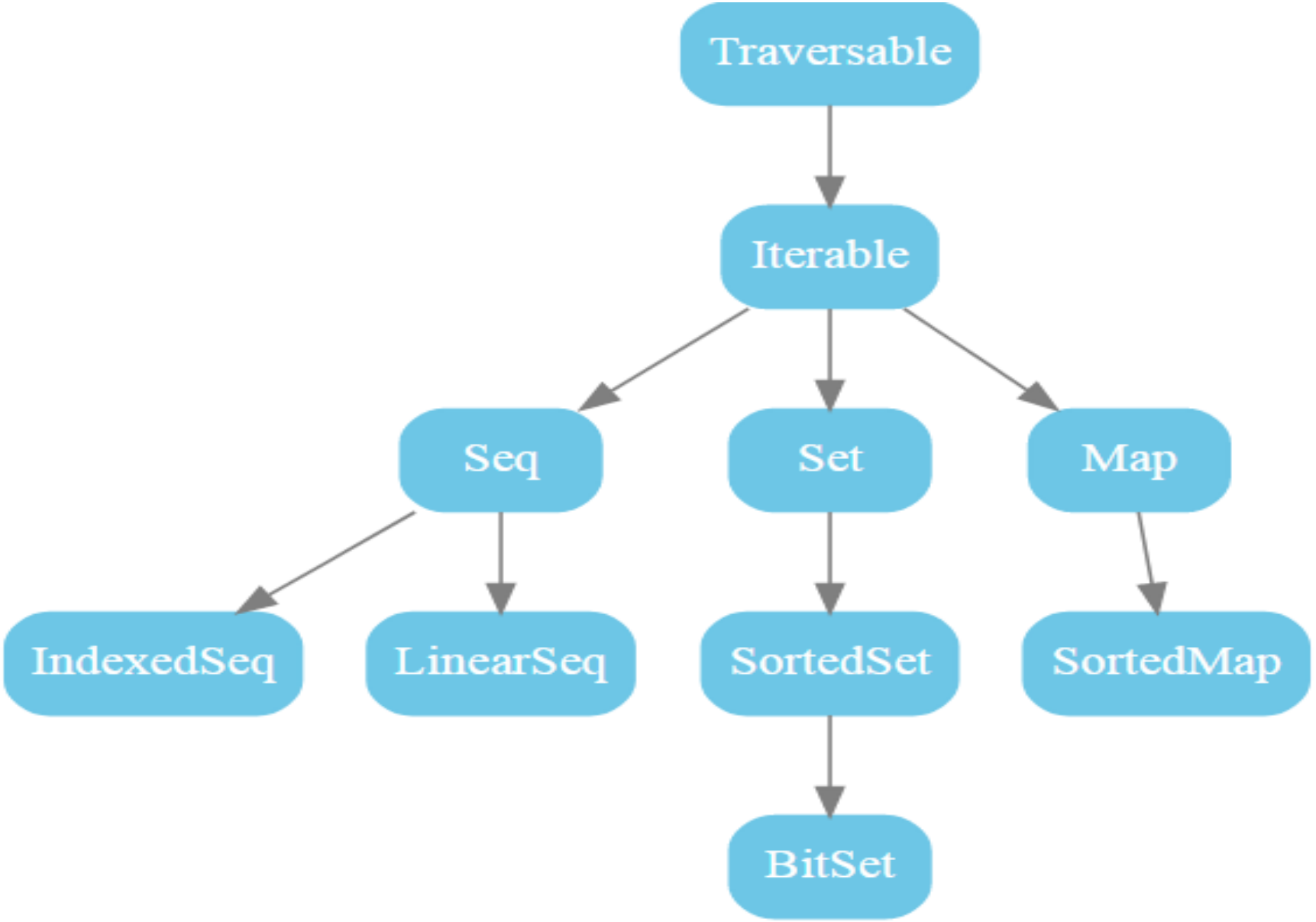
A ***mutable*** collection can be updated or extended in place. This means you can change, add, or remove elements of a collection as a side effect.

Immutable collections, by contrast, never change. You have still operations that simulate additions, removals, or updates, but those operations will in each case return a new collection and leave the old collection unchanged.

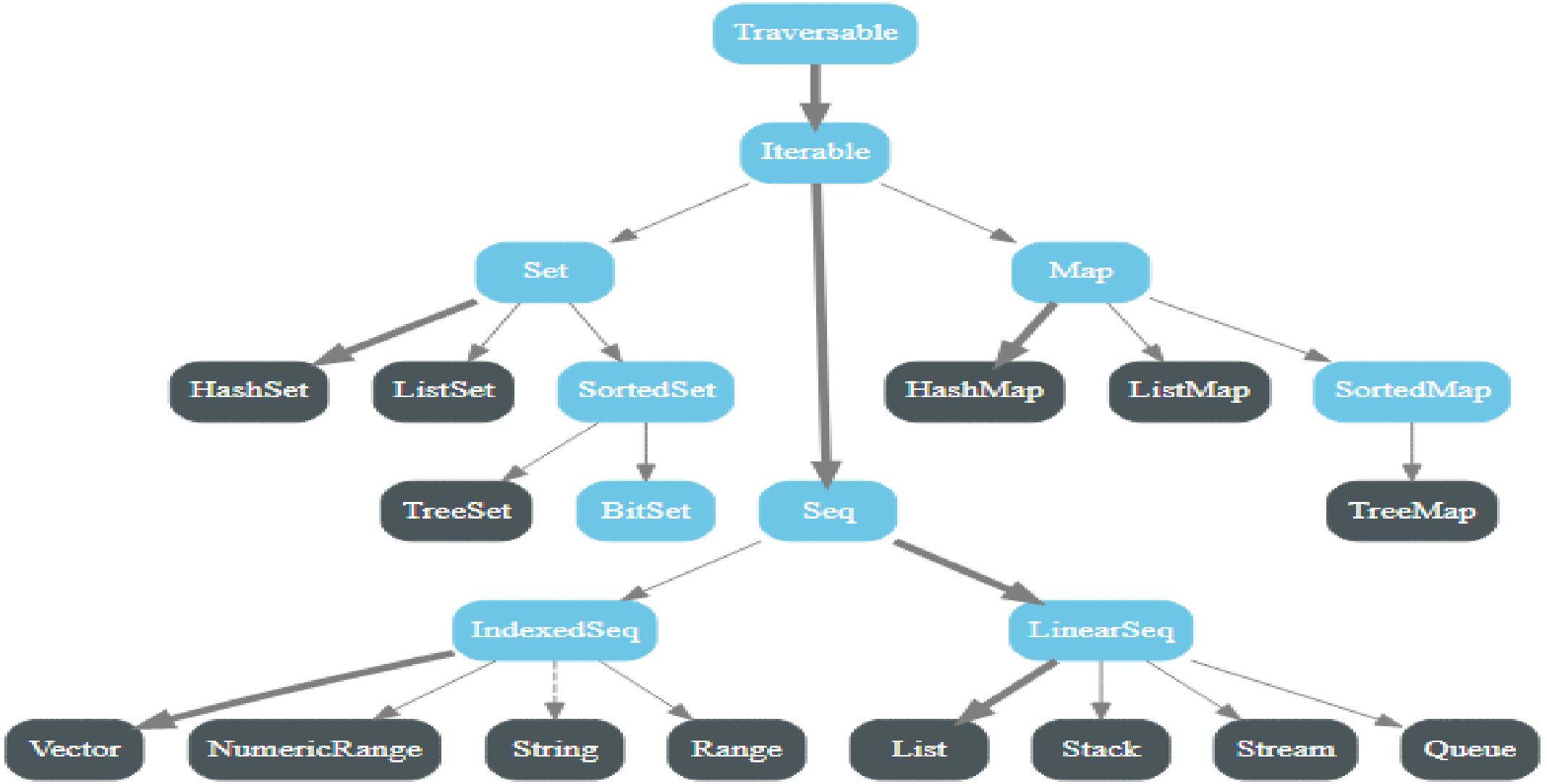
All collection classes are found in the package **`scala.collection`** or one of its sub-packages **`mutable`**, **`immutable`**, and **`generic`**.

By default, Scala always picks immutable collection

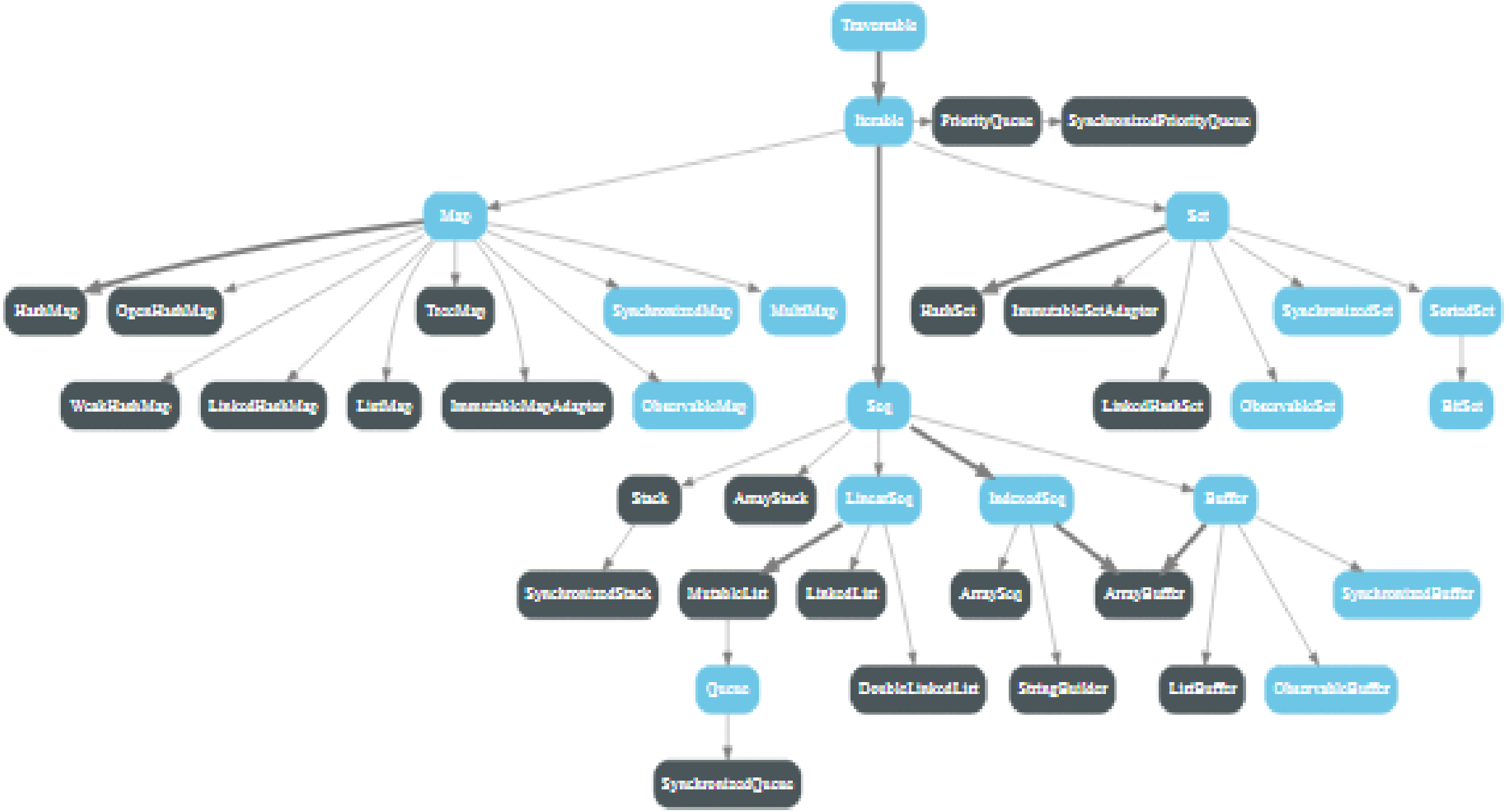
The following figure shows all collections in package `scala.collection`



The following figure shows all collections in package scala.collection.immutable.



The following figure shows all collections in package scala.collection.mutable.



Every kind of collection can be created by the same uniform syntax, writing the collection class name followed by its elements:

```
Traversable(1, 2, 3)
Iterable("x", "y", "z")
Map("x" -> 24, "y" -> 25, "z" -> 26)
Set(Color.red, Color.green, Color.blue)
SortedSet("hello", "world")
Buffer(x, y, z)
IndexedSeq(1.0, 2.0)
LinearSeq(a, b, c)
```

Maps: Map is a collection of (key, Value) pairs i.e. tuples with 2 elements.Like a dictionary.

key value

```
scala> val Marks = Map("Akshay"->25,"Mohan"->20,"Ramya"->27)
Marks: scala.collection.immutable.Map[String,Int] = Map(Akshay -> 25, Mohan -> 20, Ramya -> 27)

scala> var LabMarks = scala.collection.mutable.Map("Pavan"->45,"Nikhil"->35,"salman"->48)
LabMarks: scala.collection.mutable.Map[String,Int] = Map(Pavan -> 45, salman -> 48, Nikhil -> 35)
```

- Different ways of creating a map.
- Maps can be mutable or immutable.
- Default is immutable.

The Map data structure should be distinguished from the map method to be studied later.

Maps: Accessing Values Using Keys

```
scala> Marks("Mohan")
res7: Int = 20

scala> Marks.contains("Ramya")
res8: Boolean = true

scala> Marks.contains("Usha")
res9: Boolean = false
```

Map: Updating Values

A map can be updated only if mutable.

```
scala> LabMarks += ("Manoj"->40,"Vidya"->21)
res4: scala.collection.mutable.Map[String,Int] = Map(Pavan -> 45, Manoj -> 40, salman -> 48, Vidy
a -> 21, Nikhil -> 35)

scala> LabMarks("Pavan")
res5: Int = 45

scala> Marks += ("rishi" ->15)
<console>:9: error: value += is not a member of scala.collection.immutable.Map[String,Int]
              Marks += ("rishi" ->15)
```

Add new key value pairs, or update value of existing key.

```
scala> LabMarks -= ("Vidya")
res12: scala.collection.mutable.Map[String,Int] = Map(Pavan -> 45, Manoj -> 40, salman -> 48, Nik
hil -> 35)
```

Remove key, value pairs.

Map: Iteration

- the loop variable is actually a pair !!
- expression on the rhs of loop variable assignment i.e. “Marks” generates an **iterable**. To be explained later.
- The values of (k, v) are generated by iterating over the map.

```
scala> for((k,v)<-Marks) println((k,v))
(Akshay,25)
(Mohan,20)
(Ramya,27)

scala> LabMarks.foreach(println)
(Pavan,45)
(Manoj,40)
(salman,48)
(Vidya,21)
(Nikhil,35)
```

Arrays

- **Arrays are fixed length but mutable.**
- **Use *ArrayBuffer* if you want variable length.**

```
scala> var colors = new Array[String](7)
colors: Array[String] = Array(null, null, null, null, null, null, null)

scala> colors(0)= "RED"

scala> colors(1)= "Blue"

scala> var labs = Array(1,2,3,4,5,6)
labs: Array[Int] = Array(1, 2, 3, 4, 5, 6)

scala> for(x<-colors)println(x)
RED
Blue
null
null
null
null
null

scala> labs.foreach(println)
1
2
3
4
5
6
```

Different ways of declaring an Array.

Mutability of an Array.

Array elements accessed using round brackets or “()”.

Traversing an array

Array Buffers

```
scala> var animals = scala.collection.mutable.ArrayBuffer[String]()
animals: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer()

scala> animals = scala.collection.mutable.ArrayBuffer("Pig")
animals: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer(Pig)

scala> animals += "Rat"
res20: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer(Pig, Rat)

scala> animals += "cat"
res21: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer(Pig, Rat, cat)

scala> animals -= "Pig"
res22: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer(Rat, cat)
```

Lists

```
scala> var Department = "CSE"::"ece"::"ISE"::"MECH"::Nil
Department: List[String] = List(CSE, ece, ISE, MECH)

scala> val rows = List(1,2,3,4)
rows: List[Int] = List(1, 2, 3, 4)
```

- Lists are immutable (and ofcourse fixed length).
- Use ListBuffer if you want a mutable List.

```
scala> var fruits  = scala.collection.mutable.ListBuffer("apple")
fruits: scala.collection.mutable.ListBuffer[String] = ListBuffer(apple)

scala> fruits += "Banana"
res23: scala.collection.mutable.ListBuffer[String] = ListBuffer(apple, Banana)

scala> fruits += "grapes"
res24: scala.collection.mutable.ListBuffer[String] = ListBuffer(apple, Banana, grapes)
```

Lists continued

Lists have many useful methods which are worth familiarizing oneself with.

```
scala> List().
++              copyToArray     genericBuilder      last
++:             copyToBuffer    groupBy             leng
+:              corresponds     grouped             leng
/:              count           hasDefiniteSize     lift
:+              diff            hashCode            map
::              distinct        head                mapC
:::             drop            headOption          max
:\              dropRight       indexOf             maxB
WithFilter      dropWhile       indexOfSlice        min
addString       endsWith        indexWhere          minB
aggregate       equals          indices             mkSt
andThen         exists          init                nonE
apply           filter          inits               orEl
applyOrElse     filterNot       intersect           padT
canEqual        find            isDefinedAt         par
collect         flatMap         isEmpty             part
collectFirst    flatten         isTraversableAgain  patc
combinations    fold            iterator            perm
```

```
scala> Department:::rows
res29: List[Any] = List(CSE, ece, ISE, MECH, 1, 2, 3, 4)

scala> rows:::Department
res30: List[Any] = List(1, 2, 3, 4, CSE, ece, ISE, MECH)
```

```
scala> var sum = 0
sum: Int = 0

scala> for(ele<-rows) sum += ele

scala> sum
res37: Int = 10
```

```
scala> mul
res46: Int = 0

scala> rows.foreach(mul += _)

scala> mul
res48: Int = 10

scala> ■
```

traversing a list

Performance of Scala Collections

- Scala has many different types of collections, mutable as well as immutable.

- The performance of the same operation, e.g. indexing, can vary significantly depending upon the collection type being used. e.g. indexing a List is much faster than a ListBuffer, but ListBuffer is mutable.

- We do not discuss performance of Scala collections in this brief introduction.

Some operations on Collections: map, reduce, Iterators, Comprehensions

The map method (Already discussed)

- map applies a function to each element of a collection.
- We are using a List collection here as an illustration, but map can be applied to an Array, Map etc.
- map is often preferred to a for loop.

```
scala> val Subject  = List("BDA","CC","CD","EDM")
Subject: List[String] = List(BDA, CC, CD, EDM)

scala> val subject = Subject.map(_.toLowerCase)
subject: List[String] = List(bda, cc, cd, edm)

scala> val subject1 = Subject.map(x=>x.toLowerCase)
subject1: List[String] = List(bda, cc, cd, edm)

scala> Subject
res11: List[String] = List(BDA, CC, CD, EDM)
```

The reduce method

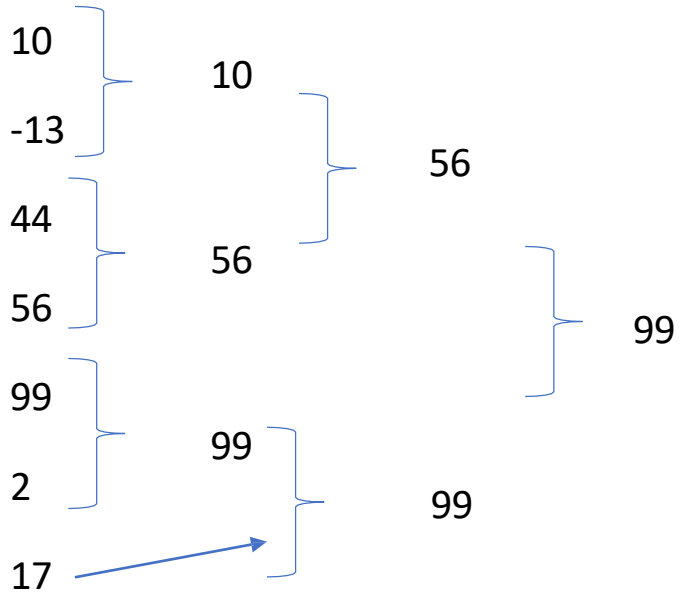
The reduce method takes all the elements in a collection and combines them using a binary operation to produce a single value.

reduce: The order in which elements are selected are not guaranteed. Hence the operation should be commutative.

reduceLeft / reduceRight: elements are combined from the left / right always, i.e. order is guaranteed.

```
scala> val ints : List[Int] = List(10,-13,20,40,100,45,67)
ints: List[Int] = List(10, -13, 20, 40, 100, 45, 67)

scala> ints.reduce((x,y)=>x max y)
res16: Int = 100

scala> ints.reduceRight(_ min _)
res17: Int = -13
```

Note the alternate way of specifying arguments.



```
scala> var intr = scala.collection.mutable.ArrayBuffer[String]("hello","apple","Java","zebra")
intr: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer(hello, apple, Java, zebra)

scala> intr += "education"
res20: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer(hello, apple, Java, zebra, educ
ation)

scala> intr.reduceLeft((x,y)=> if(x.length > y.length) x else y)
res21: String = education
```

```
scala> val mul = List(2.0 ,1.5,6.9)
mul: List[Double] = List(2.0, 1.5, 6.9)

scala> val divide=(n : Double, m: Double) => {
     | var result = n/m
     | result
     | }
divide: (Double, Double) => Double = <function2>

scala> mul.reduceLeft(divide)
res24: Double = 0.1932367149758454

scala> mul.reduceRight(divide)
res25: Double = 9.200000000000001
```

How to use Iterator in Java?

- ‘Iterator’ is an interface which belongs to collection framework. It allows us to traverse the collection, access the data element and remove the data elements of the collection.
  java.util package has **public interface Iterator** and contains three methods:
- **boolean hasNext()**: It returns true if Iterator has more element to iterate.
- **Object next()**: It returns the next element in the collection until the hasNext()method return true. This method throws ‘NoSuchElementException’ if there is no next element.
- **void remove()**: It removes the current element in the collection. This method throws ‘IllegalStateException’ if this function is called before next( ) is invoked.

```
import java.io.*;
import java.util.*;
 class Test {
    public static void main(String[] args)
    {
        ArrayList<String> list = new ArrayList<String>();
         list.add("A");
        list.add("B");
        list.add("C");
        list.add("D");
        list.add("E");
         // Iterator to traverse the list
        Iterator iterator = list.iterator();
         System.out.println("List elements : ");
         while (iterator.hasNext())
           System.out.print(iterator.next() + " ");

        System.out.println();
    }
}
```

Iterator

- Although using an iterator with hasNext() and next() is a common way to loop over a collection in Java, they aren't commonly used in Scala, because Scala collections have methods like map and foreach that let you implement algorithms more concisely.
- In for-each loop, we can't modify collection, it will throw a ConcurrentModificationException on the other hand with iterator we can modify collection.Modifying a collection simply means removing an element or changing content of an item stored in the collection.

```
scala> val names = List("Paul","Alex", "bob")
names: List[String] = List(Paul, Alex, bob)

scala> val nameite = names.iterator
nameite: Iterator[String] = non-empty iterator

scala> while(nameite.hasNext)println(nameite.next)
Paul
Alex
bob
```

- iterators are defined for collections. they allow stepping through the elements of a collection.
- Using map or foreach usually a much better option.

```
scala> val nite = scala.io.Source.fromFile("/root/Desktop/text.txt").getLines
nite: Iterator[String] = non-empty iterator

scala> while(nite.hasNext)println(nite.next)
hello
how r u
all well
ksjsd
djfjfnf
```

- Use iterators when collection is too big to place completely in memory. e.g. when processing large files.
- Iterator will only get one item at a time.
- Iterators can be thought of as pointers to elements of the collection.
- An important part of using an iterator is knowing that it's exhausted after you use it.

Review Questions and Practise Exercises 3

1. A map is like a dictionary. Explain.
2. Set up a map of gadgets that you want, along with their prices. Create a second map of the gadgets with a 10% discount. Print the second map and inspect its contents.
3. Use a mutable map to count the number of times each word appears in a sentence (provided as a string).
4. Write a function lteqgt(values: Array[Int], v:Int) that returns a triple containing the counts of values less than v, equal to v, and greater than v.
5. In the scala console, type “Hello”.zip(“World”). What does it do ? Give a scenario where you can use this method.

Review Questions and Practise Exercises 3

6. Explain what is a comprehension in your own words. Give an example of comprehensions in Scala and explain the correspondence between your explanation and example.
7. When does it make sense to use iterators ? Why is map / foreach usually a better option ?
8. Given an array of strings, find the sum of the length of all the strings.
9. Write a function that given a string, produces a map of the indexes of all the characters as a list. For example, indexes("Mississippi") should return a map associating 'M' with the List(0), 'i' with the List(1, 4, 7, 10) and so on. Use a mutable map of characters and ListBuffer's which are mutable, in place of List's which are not mutable.

Tips:

i. To use ListBuffer Class, you will have to import scala.collections.mutable.ListBuffer package.
ii. To add an element to a ListBuffer, the += operator has to be used.
iii. To generate the positions of the characters in the string, you can use the Range operator i.e. (0 until str.length).
iv. A possible way of generating the initial list of maps of characters and their positions, e.g. ('M', 0), ('I', 1), ('s', 2)… is using the zip operation on the string and the list generated as in (iii). From this list, the required map can be generated.

Scala Comprehensions – Simple Example

have to use “yield” keyword with for loop.

```
scala> val name1 = Array("chris","paul","alic")
name1: Array[String] = Array(chris, paul, alic)

scala> val namr = for(e<-name1) yield e.capitalize
namr: Array[String] = Array(Chris, Paul, Alic)
```

source (must be an iterable)

expression to generate elements of new set.

No filter is given in this example.

Destination type is same as source type (some rare exceptions exist).

Scala Comprehensions – Complex Example

```
scala> val user = List(("ashwin" ,30),("mohan",25),("Rajesh",40),("usha",22),("teja",24),("vivek"
,35),("eshwar",32))
user: List[(String, Int)] = List((ashwin,30), (mohan,25), (Rajesh,40), (usha,22), (teja,24), (viv
ek,35), (eshwar,32))

scala> val sr = for(r<-user; if(r._2 > 20 && r._2 < 30)) yield r._1
sr: List[String] = List(mohan, usha, teja)
```

Classes and Objects

```
scala> class Counter {
     | private var value = 0
     | def increment() { value += 1}
     | def current = value
     | }
defined class Counter
```

- Stylistically, methods which change values are written with “()”.
- Methods which don’t are written without “()”.

```
scala> val myCounter = new Counter
myCounter: Counter = Counter@6764201e

scala> myCounter.increment()

scala> println(myCounter.current)
1
```

Class: Getters and Setters

```
scala> class Person {
     | var age = 0
     | }
defined class Person

scala> val Rahul = new Person
Rahul: Person = Person@6079d219

scala> Rahul.age = 30
Rahul.age: Int = 30

scala> println(Rahul.age)
30
```

- Calls setter Rahul.age_ = 30
- Calls getter Rahul.age()
- A getter only is generated for a val
- A getter and setter is generated for
- every variable in the class

```
scala> class Student {
     | private var privateAge = 0
     | def age = privateAge
     | def age_=(newValue: Int) {
     | if(newValue > privateAge) privateAge =newValue;
     | }
     | }
defined class Student

scala> val fred = new Student
fred: Student = Student@7f088b5c

scala> fred.age = 30
fred.age: Int = 30

scala> fred.age = 21
fred.age: Int = 30
```

Getters and setters can be redefined.

- .

A private variable has private getters and setters.

Inheritance etc are beyond the scope of this short introduction.

Singleton Object

- Singleton object is an object which is declared by using object keyword instead by class. No object is required to call methods declared inside singleton object.

```
object Singleton{
   def main(args:Array[String]){
      SingletonObject.hello()        // No need to create object.
   }
}

object SingletonObject{
   def hello(){
```

```
scala> object Accounts {
     | private var lastNumber = 0
     | def newUniqueNumber() = { lastNumber += 1; lastNumber}
     | }
defined object Accounts

scala> val newAccountNumber = Accounts.newUniqueNumber
newAccountNumber: Int = 1

scala> val newAccountNumber = Accounts.newUniqueNumber
newAccountNumber: Int = 2
```

```
object Hello {
  def main(args: Array[String]) {
    println("Hello, World!")
  }
}
```

Each Scala program must start with an object's main method of type Array[String] => Unit

In scala, there is no static concept. So scala creates a singleton object to provide entry point for your program execution. If you don't create singleton object, your code will compile successfully but will not produce any output. Methods declared inside Singleton Object are accessible globally. A singleton object can extend classes and traits.

Review Questions and Practise Exercises 4

1. Write a class Time with read only properties hours and minutes and a method before(other: Time): Boolean that checks whether this time comes before the other. A Time object should be constructed as new Time(hrs, min), where hrs is in 24 hour format (0 to 23).
2. Write an object Conversions with methods inchesToCms and milesToKms. Use it to convert 5 inches to centimetres and 10 miles to kilometers.
3. What are getters and setters ? Explain with an example.

Pattern Matching

Pattern Matching in Java Switch Statement

```
int i=2;
switch(i)
{
   case 1:
     System.out.println("Case1 ");
     break;
   case 2:
     System.out.println("Case2 ");
     break;
   case 3:
     System.out.println("Case3 ");
     break;
   case 4:
     System.out.println("Case4 ");
     break;
   default:
     System.out.println("Default ");
}
```

variable

possible values of variable, have to be a constant.

Where is the pattern matching here ?
The value of the variable “I” is matched to constant values of the case statement.

breaks are important.

when variables value is not matched, default branch is executed.

Pattern matching in Scala is incredibly more powerful.

Pattern Matching in Scala

```
scala> var result = 1
result: Int = 1

scala> result match {
     | case 1 =>"You are Right"
     | case 0 =>"You are not wrong"
     | case _ => "You are Wrong"
     | }
res15: String = You are Right
```

match is equivalent of switch statement.

no break statements required.

default case is indicated by “_”.

In Scala, case statements also have a value similar to if statements.

Sequence Patterns

```
scala> val myList = List(1,2,3,4)
myList: List[Int] = List(1, 2, 3, 4)

scala> val printListType = myList match {
     | case List(0,_,_) => "a Three element list"
     | case List(1,_*) => "a list beginning with 1"
     | case _ => "not a know type list"
     | }
printListType: String = a list beginning with 1
```

Patterns in Variable Declarations

```
scala> val (teacher, teacherDetails) =
     | ("Ushashree", ("ushashree.sgs@nmit.ac.in","9254689331"))
teacher: String = Ushashree
teacherDetails: (String, String) = (ushashree.sgs@nmit.ac.in,9254689331)

scala> println(teacherDetails)
(ushashree.sgs@nmit.ac.in,9254689331)


scala> val (teacher, (email,cellNo)) =
     | ("Archana naik", ("archananaik@nmit.ac.in","9552629318"))
teacher: String = Archana naik
email: String = archananaik@nmit.ac.in
cellNo: String = 9552629318

scala> println(email)
archananaik@nmit.ac.in
```

Patterns in for Expressions

```
scala> import scala.collection.JavaConversions.propertiesAsScalaMap
import scala.collection.JavaConversions.propertiesAsScalaMap

scala> for((k,v)<-System.getProperties())
     | println(s"$k -> $v")
java.runtime.name -> Java(TM) SE Runtime Environment
sun.boot.library.path -> /usr/java/jdk1.8.0_221-amd64/jre/lib/amd64
java.vm.version -> 25.221-b11
java.vm.vendor -> Oracle Corporation
java.vendor.url -> http://java.oracle.com/
path.separator -> :
java.vm.name -> Java HotSpot(TM) 64-Bit Server VM
file.encoding.pkg -> sun.io
user.country -> US
```

Review Questions and Practise Exercises 5

1. Using pattern matching, write a function swap that swaps the first two elements of an array provided its length is atleast two. Refer to the example on List matching for hints.
2. Matching on case classes: Case classes are especially useful for pattern matching. Define two case classes as below:

```
abstract class Notification
case class Email(sender: String, title: String, body: String) extends Notification  case class
     SMS(caller: String, message: String) extends Notification
```

Define a function showNotification which takes as a parameter the abstract type Notification and matches on the type of Notification (i.e. it figures out whether it's an Email or SMS).
In the case it's an Email(email, title, _) return the string: s"You got an email from $email with title: $title"
In the case it's an SMS return the String: s"You got an SMS from $number! Message: $message"

Functional Programming (Optional)

Functional Programming

- Definition 1: In functional programming, programs are executed by evaluating *expressions*, in contrast with imperative programming where programs are composed of *statements* which change global *state* when executed. Functional programming typically avoids using mutable state. Benefit: Safer code. Immutability is especially desirable for data processing applications in a distributed framework.
- Definition 2: Functional programming requires that functions are *first-class*, which means that they are treated like any other values and can be passed as arguments to other functions or be returned as a result of a function. Benefit: Better abstraction, easier to understand code.

Scala strongly supports functional programming, but has OOP + functional features.

Features

- Higher-order functions (HOFs) are functions that take other functions as their arguments.
  - Scala example: map function.
- Immutable data, that is data specified in the code can not be changed.
  - Scala example: List, val.
- Purity where expressions do not yield actions and just have return values.
  - Input / output are actions. Since IO is required, its recommended to have a non-functional component for IO.

Features Continued

- Referential transparency: Pure computations yield the same value each time they are invoked. In other words there is no state which can change the value of the computation dependent upon the value of the state.
- Lazy evaluation: computations are deferred until they are needed as opposed to when they are specified.
  - Since pure computations are referentially transparent they can be performed at any time and still yield the same result. This makes it possible to defer the computation of values until they are needed, that is, to compute them *lazily*. *Lazy evaluation* avoids unnecessary computations and allows, for example, infinite data structures to be defined and used.
  - In Scala, one needs to explicitly specify lazy evaluation. In Spark, lazy evaluation is the default.

Quicksort: Imperative versus Functional Styles

```
/** Quick sort, imperative style */
object sort {

  /** Nested methods can use and even update everything
   *  visible in their scope (including local variables or
   *  arguments of enclosing methods).
   */
  def sort(a: Array[Int]) {

    def swap(i: Int, j: Int) {
      val t = a(i); a(i) = a(j); a(j) = t
    }

    def sort1(l: Int, r: Int) {
      val pivot = a((l + r) / 2)
      var i = l
      var j = r
      while (i <= j) {
        while (a(i) < pivot) i += 1
        while (a(j) > pivot) j -= 1
        if (i <= j) {
          swap(i, j)
          i += 1
          j -= 1
        }
      }
      if (l < j) sort1(l, j)
      if (j < r) sort1(i, r)
    }

    if (a.length > 0)
      sort1(0, a.length - 1)
  }

  def println(ar: Array[Int]) {
    def print1 = {
      def iter(i: Int): String =
        ar(i) + (if (i < ar.length-1) "," + iter(i+1) else "")
      if (ar.length == 0) "" else iter(0)
    }
    Console.println("[" + print1 + "]")
  }

  def main(args: Array[String]) {
    val ar = Array(6, 2, 8, 5, 1)
    println(ar)
    sort(ar)
    println(ar)
  }

}
```

```
/** Quick sort, functional style */
object sort1 {
  def sort(a: List[Int]): List[Int] = {
    if (a.length < 2)
      a
    else {
      val pivot = a(a.length / 2)
      sort(a.filter(_ < pivot)) :::
           a.filter(_ == pivot) :::
           sort(a.filter(_ > pivot))
    }
  }
  def main(args: Array[String]) {
    val xs = List(6, 2, 8, 5, 1)
    println(xs)
    println(sort(xs))
  }
}
```

Functional Programming Features Explained

- Pure functions and side effects
- Referential transparency
- First class functions and higher order functions
- Anonymous functions

FP - Elements of Functional Programming - Part-1.mp4

- Immutability
- Recursion and Tail Recursion
- Statements

FP - Elements of Functional Programming - Part 2.mp4

Functional Programming Features Explained

- Strict and Non-strict (Lazy) evaluations

FP - Strict and Lazy Evaluations.mp4

- Pattern matching

FP - Pattern Matching.mp4

- Closures

FP - Closures.mp4

Review Questions and Practise Exercises 6

1. What are some of the key features of Functional Programming.
2. Try to find out why Functional code is more amenable for parallel programming.
3. Explain how quicksort works. Implement Quicksort using functional programming style. (Lab Assignment)

FP – Why it is not widespread ? (From Quora)

- Travis Addair, Senior Software Engineer at Uber; prev SEMC, Storm8, Google, LLNL, USGS
- Answered Apr 13, 2016
- The reason that more businesses aren't using functional languages in production is simply that the **increased benefit/cost ratio claims are as yet unproven**.
- It may very well be true that functional programming languages are safer, more performant, more scalable, more elegant, and more versatile than any comparable OOP language used in production, but remember that the burden of proof here is on the functional language advocates to demonstrate why we *should* be using FP, not on industry to prove that we *shouldn't*.
- In order to convince industry that FP is the future and key to their business' continued growth and success, I believe the FP community needs to demonstrate the following:
- At least one successful $1+ billion businesses that built their core technology stack using a Functional Language (the **existence** proof).
- Show that this technology represents a competitive advantage in the business' market, without which they would not be able to compete (the **value** proof).
- Show that this technology would not have been feasible to develop at or below the order of magnitude cost using a more commonplace OOP language (the **opportunity cost** proof).
- Once you get to point number 3, I think it's a very difficult case to prove, but remember that from a business standpoint, the cost of switching to a new tech stack is expensive, particularly when we consider the added risks involved:
- What about all our legacy OOP code?

FP – Why is not widespread ? (From Quora)

- Will we be able to hire enough engineers who can get up to speed with this FP language quickly?
- What if this FP thing is a fad and we're left with unsupported tech in a few years?
- What kind of ecosystem exists to support this new FP tech stack?
- What kinds of unforeseen issues might arise with scaling this technology if it hasn't been done successfully by a business our size before?
- In my experience, the language is rarely the bottleneck preventing your business from scaling up to the next level. Many of the world's biggest and most successful tech enterprises were built on tech stacks that many language purists would consider shoddy at best. Facebook, for instance, was built on PHP, widely regarded as one of the worst languages around.
- And yet, here they are. Not only surviving, but thriving. Sure, they may be using other languages for other things today, but the core of their business is still built around PHP.
- So if Functional Programming wants to eat the world, it's going to need to take a bite out of some problem too tough for OOP to chew. Then we may just see a big surge in FP investment similar to the NoSQL boom of the past few years.

FP – Why is not widespread ? (From Quora)

- Ivan Fedorov, programmer / entrepreneur
- Answered Sep 15, 2015
- Facebook uses Haskell for spam prevention, Walmart uses Clojure for handling hundreds of thousands transactions per second. Nuff said?

I think its just because the swing into FP has just began, while the swing into OOP is 30-40 years old. (When I say the swing, I mean the mentions swing, not the programming languages development.

Daniel Philippus, Used several languages, mainly Haskell, Scala, C, Go

- Answered Apr 12, 2016
- It's not as popular, hence less support and fewer libraries, and difficult to get into.
- The startup I'm working with is writing our back-end in Haskell. It's a good choice: the language itself has some impressive capabilities, and the web framework Yesod that we're using makes excellent use of those for good performance, easy use of complex features, and superb reliability that few languages are capable of (e.g. type-safe URLs, in-site links that automatically update when you move a page).
- On the other hand, it's taking noticeably longer to get going than it would have with the alternative (which was Dlang and the Vibe.d framework). Haskell is an utterly bizarre language compared to what I've used before, and it's taking time to learn it (my previous projects all used imperative languages like Golang, Dlang, Python). There aren't as many online resources for me to draw on due to the smaller community. While the framework is powerful, the way it uses Haskell's capabilities makes much of my previous experience of little use due to differences. The other developer just has absolutely no idea what's going on with its complexities, so he's pretty much only going to be doing front-end work (JavaScript developer).

Miscellanous

- Scala cheat sheet:
  - http://www.cis.upenn.edu/~matuszek/Concise%20Guides/Concise%20Scala.html
- Functional Programming Discussion:
  - https://wiki.haskell.org/Functional_programming
  - https://en.wikipedia.org/wiki/Functional_programming