Big Data Analytics

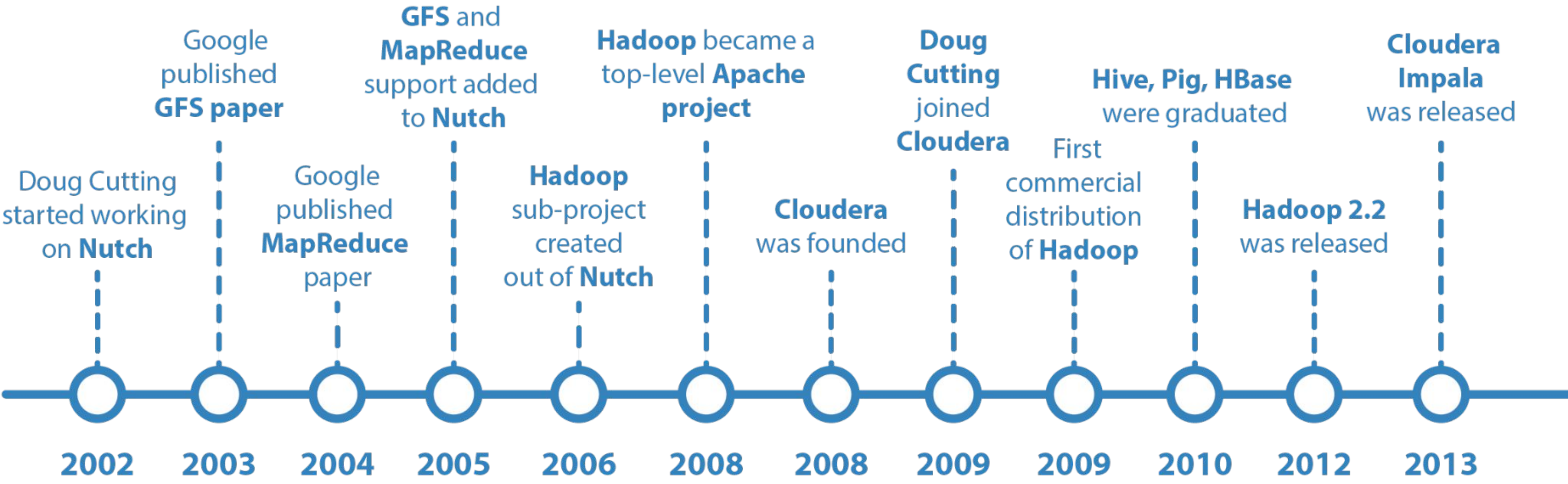
Unit 3

Small Data vs Big Data

BIG DATA tells us **what's** happening.

small data tells us **why** it's happening.

History



2002 — Doug Cutting started working on **Nutch**

2003 — Google published **GFS paper**

2004 — Google published **MapReduce** paper

2005 — **GFS** and **MapReduce** support added to **Nutch**

2006 — **Hadoop** sub-project created out of **Nutch**

2008 — **Hadoop** became a top-level **Apache project**

2008 — **Cloudera** was founded

2009 — **Doug Cutting** joined **Cloudera**

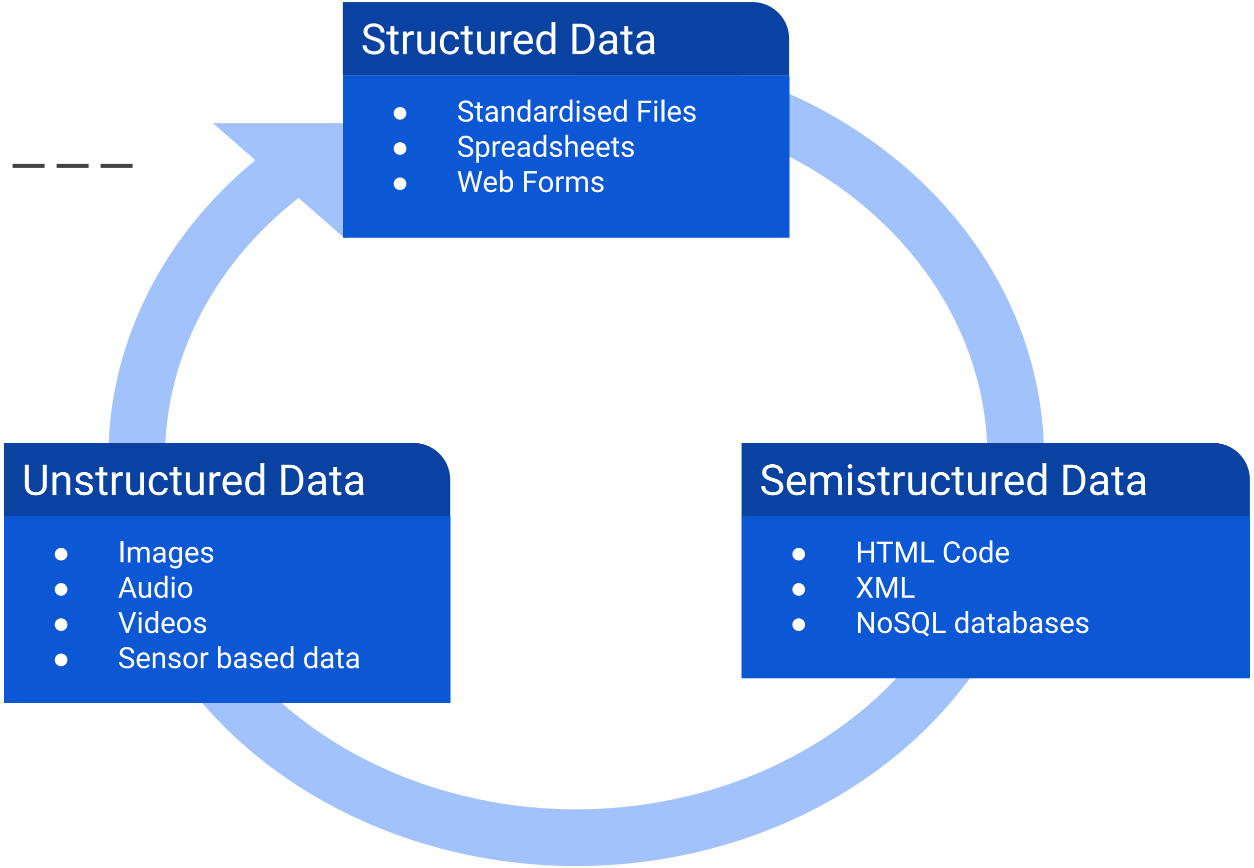
2009 — First commercial distribution of **Hadoop**

2010 — **Hive, Pig, HBase** were graduated

2012 — **Hadoop 2.2** was released

2013 — **Cloudera Impala** was released

Types of Data



Structured Data

- Standardised Files
- Spreadsheets
- Web Forms

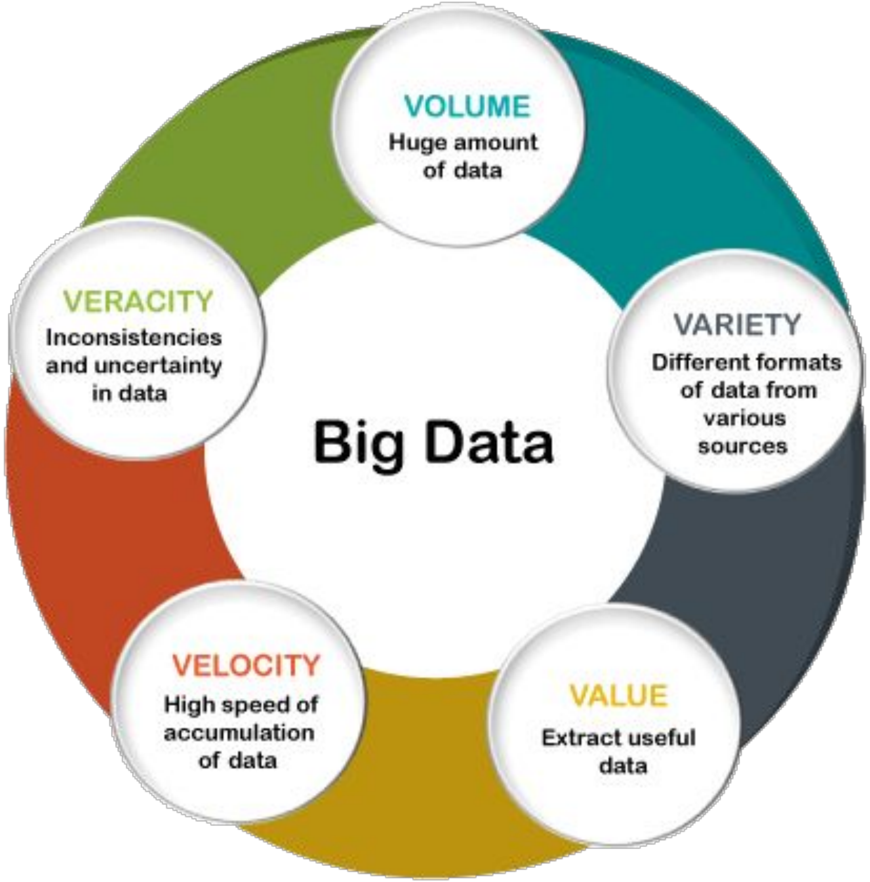
Semistructured Data

- HTML Code
- XML
- NoSQL databases

Unstructured Data

- Images
- Audio
- Videos
- Sensor based data

5 V’S of Big Data



Big Data

VOLUME
Huge amount of data

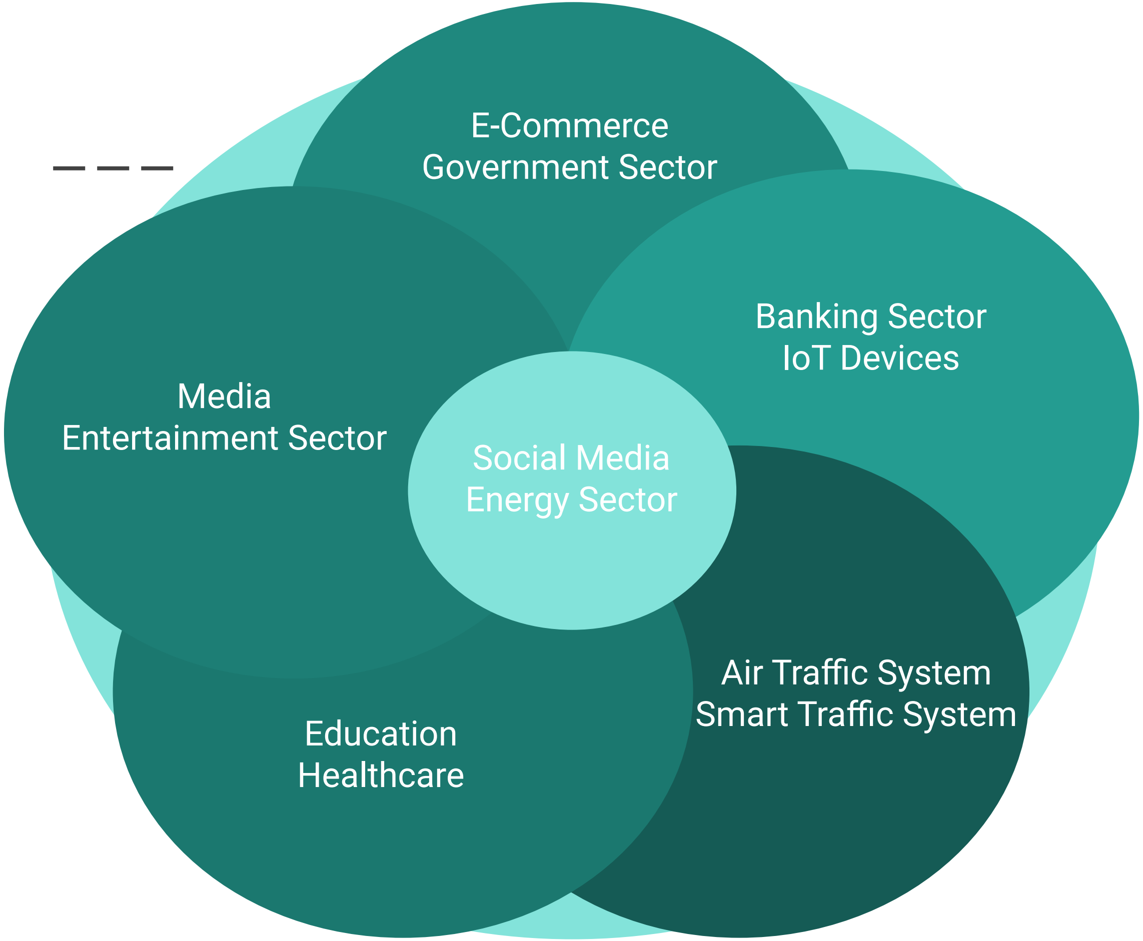
VARIETY
Different formats of data from various sources

VALUE
Extract useful data

VELOCITY
High speed of accumulation of data

VERACITY
Inconsistencies and uncertainty in data

Real Time Applications



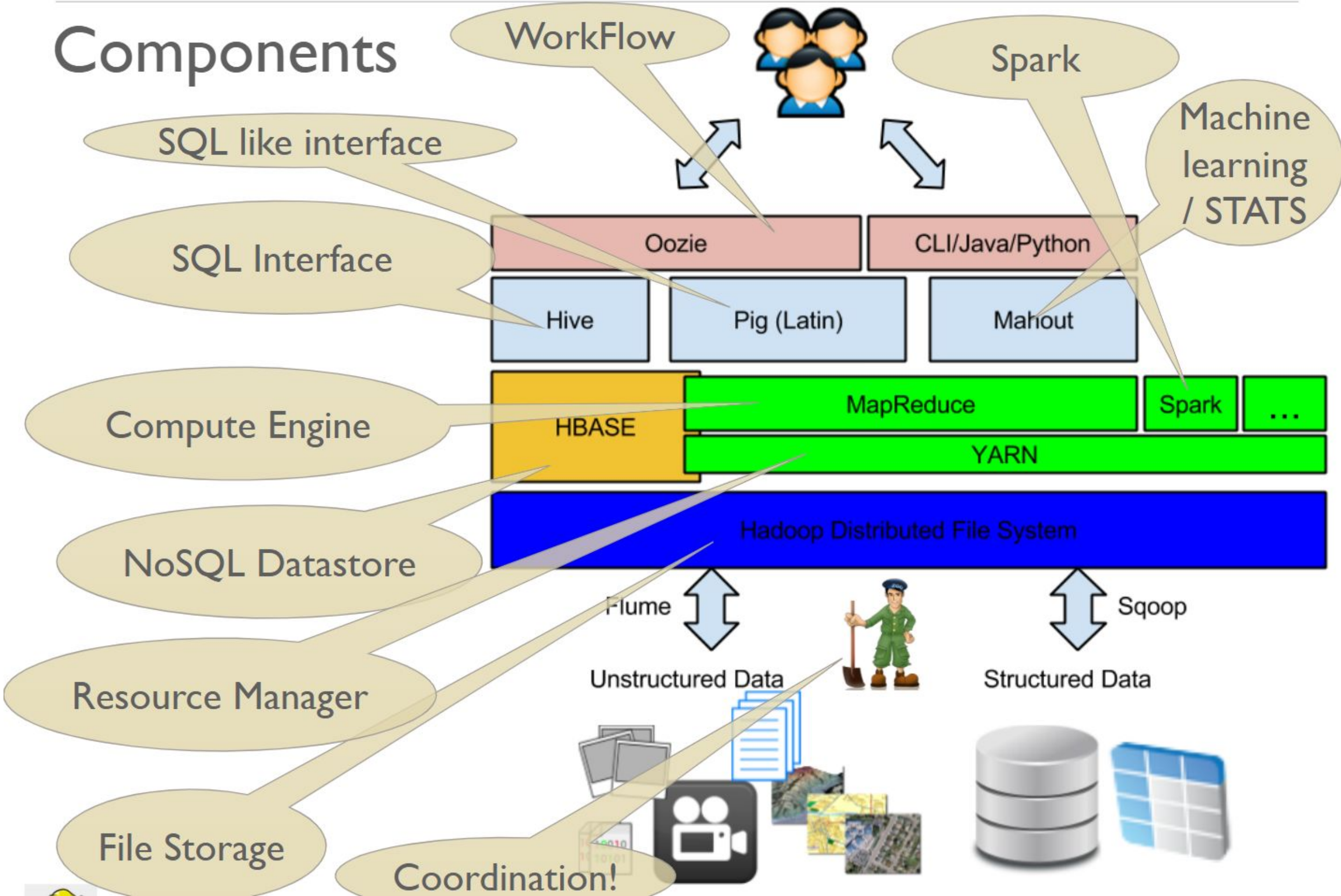
Advantages of Big Data

- Better Decision Making
- Reduce cost of business process
- Fraud Detection
- Increased Productivity
- Improved Customer Service
- Enhanced Speed and Agility

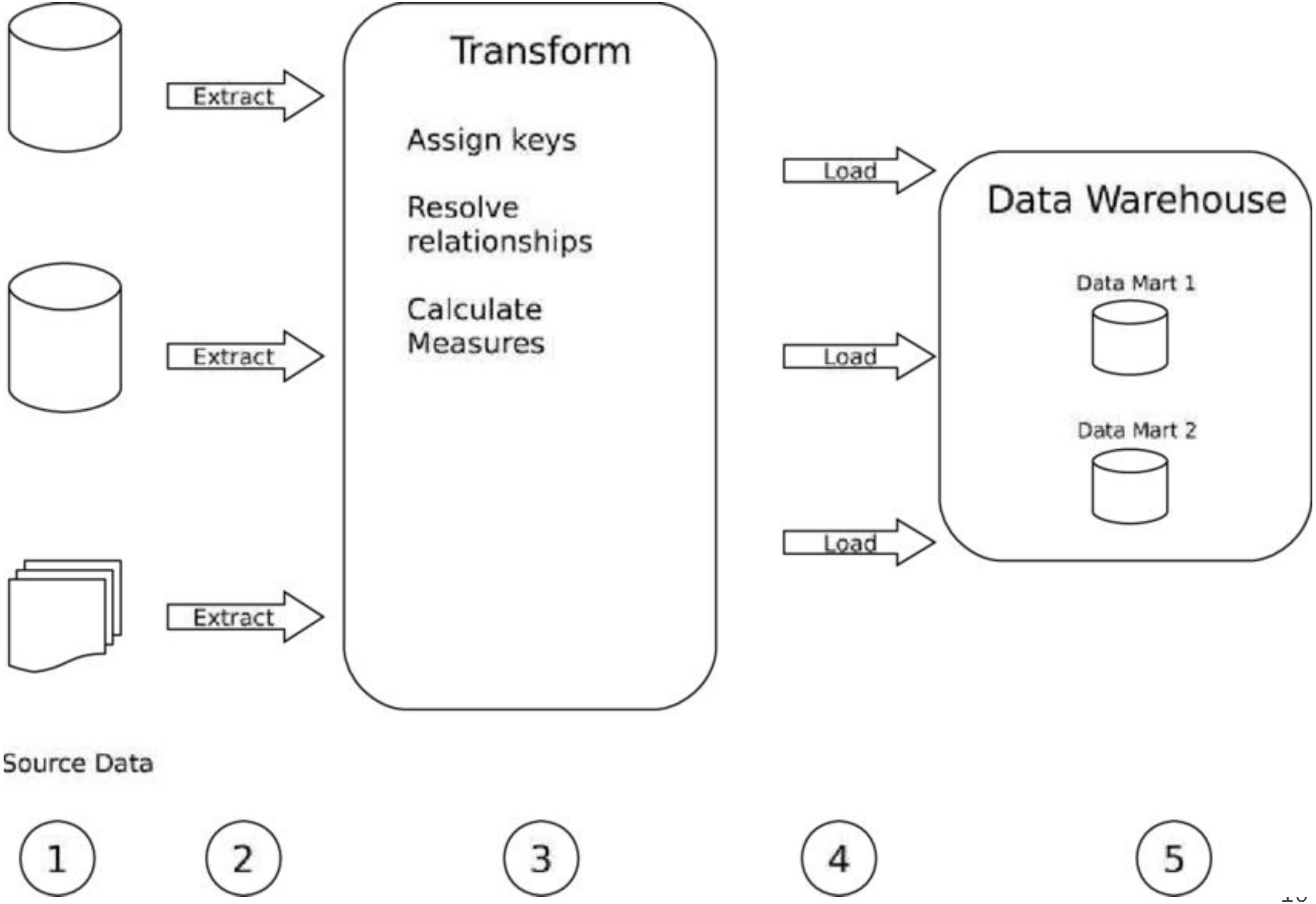
Disadvantages of Big Data

- Talent Gap
- Security Hazard
- High Cost
- Data Quality
- Rapid Change in technology at market place

Components


WorkFlow
Spark
Machine learning / STATS
SQL like interface
SQL Interface
Compute Engine
NoSQL Datastore
Resource Manager
File Storage
Coordination!
Oozie
CLI/Java/Python
Hive
Pig (Latin)
Mahout
HBASE
MapReduce
Spark
...
YARN
Hadoop Distributed File System
Flume
Sqoop
Unstructured Data
Structured Data

ETL Process


Transform
Extract
Extract
Extract
Assign keys
Resolve relationships
Calculate Measures
Load
Load
Load
Data Warehouse
Data Mart 1
Data Mart 2
Source Data
1
2
3
4
5

Why we need DFS?

- DFS stands for the distributed file system
- It is a concept of storing the file in multiple nodes in a distributed manner.
- Disk capacity of a system can only increase up to an extent.
- Processing large datasets on a single machine is not efficient.

Contd...

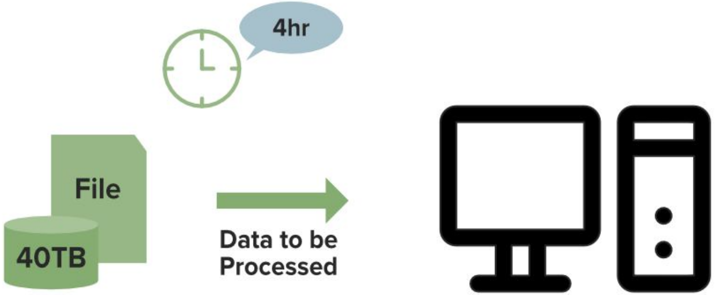


Fig. Local File System Processing

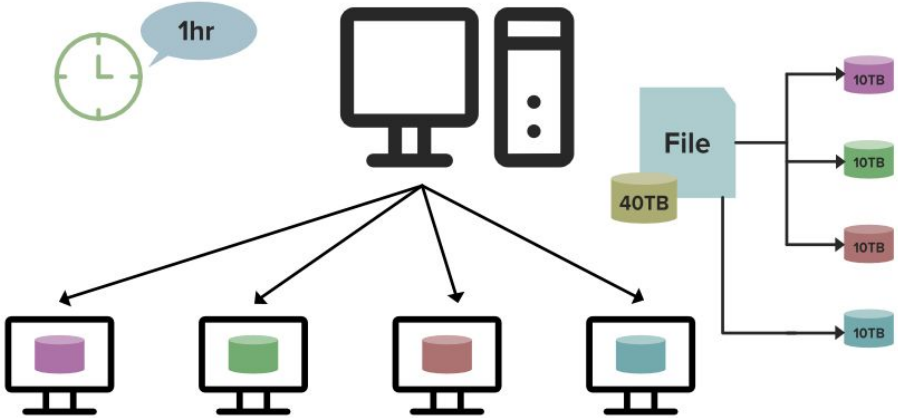


Fig. Distributed File System Processing

Hadoop

- It is an **open source framework** overseen by Apache Software Foundation which is written in **Java** for **storing and processing** of huge datasets with the cluster of commodity hardware.

- There are mainly three components of Hadoop which are **Hadoop Distributed File System (HDFS)**, **Yet Another Resource Negotiator(YARN) and MapReduce**

- Namenode, Secondary NameNode, and Resource Manager work on a Master System while the Node Manager and DataNode work on the Slave machine. **[start-all.sh and jps]**

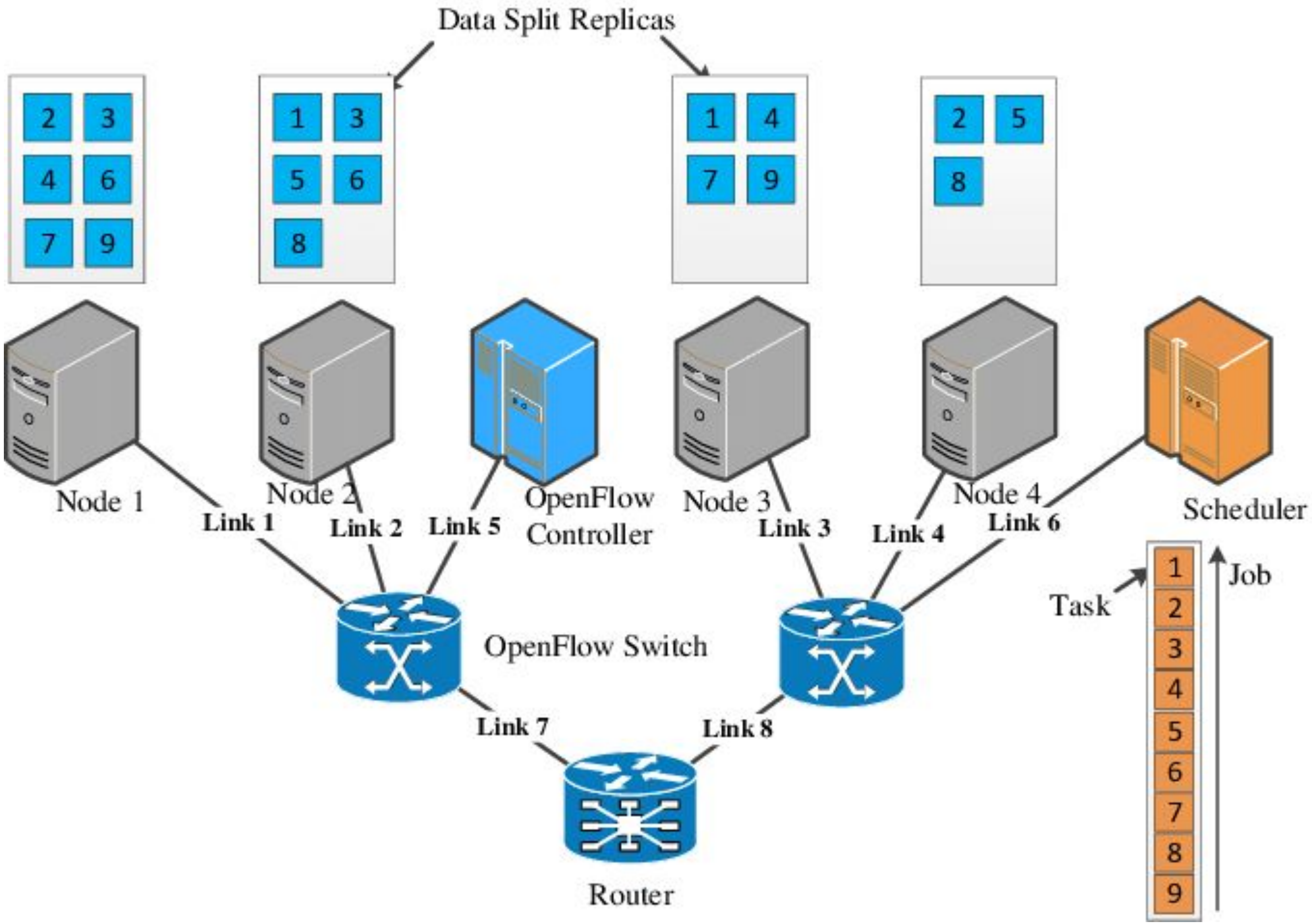
- **It is loosely coupled and easy to integration with other tools**

HDFS

- Hadoop Distributed File System is **a dedicated file system** to store big data with a cluster of commodity hardware or cheaper hardware with streaming access pattern

- Data to be stored at **multiple nodes** in the cluster which ensures **data security and fault tolerance**

Contd....



Data Split Replicas
2
3
4
6
7
9
1
3
5
6
8
1
4
7
9
2
5
8
Node 1
Node 2
OpenFlow Controller
Node 3
Node 4
Scheduler
Link 1
Link 2
Link 5
Link 3
Link 4
Link 6
OpenFlow Switch
Link 7
Link 8
Router
Task
Job
1
2
3
4
5
6
7
8
9

NameNode

- NameNode works as a Master in a Hadoop cluster
- Used for **storing the Metadata** i.e. the data about the data.
- Meta Data can also be the name of the file, size and the information about the location(Block number, Block ids) of Datanode
- Namenode instructs the DataNodes with the operation like delete, create, Replicate, etc.
- Hadoop Command: **hadoop-daemon.sh start namenode**
- Name Node is running on 50070 port number

DataNode

- DataNode works as a Slave DataNodes are mainly utilized for **storing the data** in a Hadoop cluster
- The number of DataNode can be scalable from 1 to any number.
- Data in HDFS is always stored in terms of blocks.
- The single block of data is divided into multiple blocks of size 128MB which is default
- Hadoop Command: **hadoop-daemon.sh start datanode**
- Data Node is running on **50075** port number

Contd...

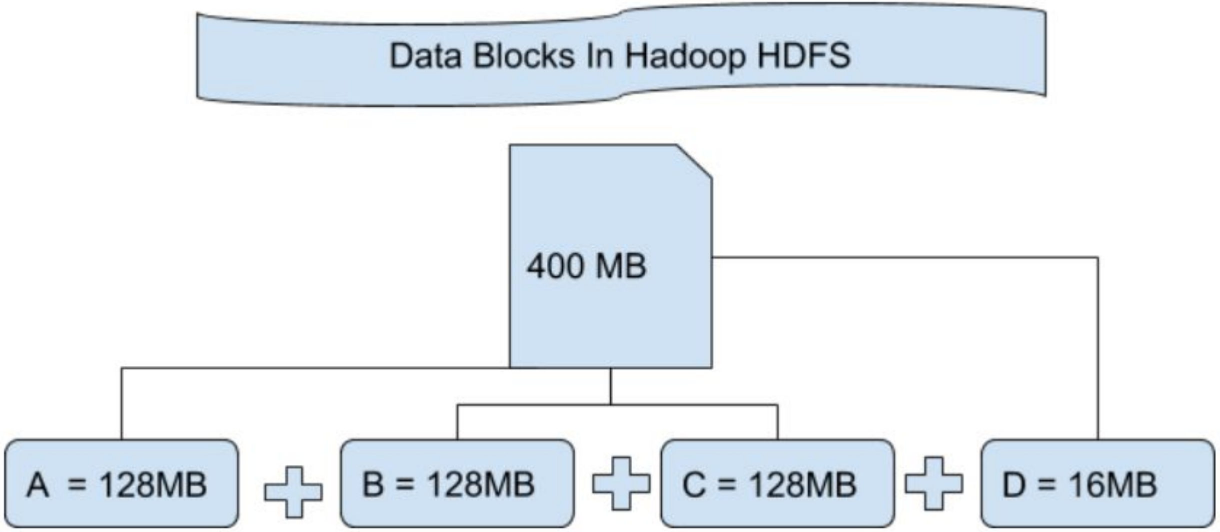


Fig. Data storage in blocks

Secondary Namenode

- Secondary NameNode is used for taking the hourly **backup of the data** and store this data into a file name *fsimage*.
- This file then gets transferred to new Master system if any crash happens.
- It continuously reads the MetaData from the RAM of NameNode and writes into the Hard Disk.
- secondary Namemode is running on **50090** port number

Contd….

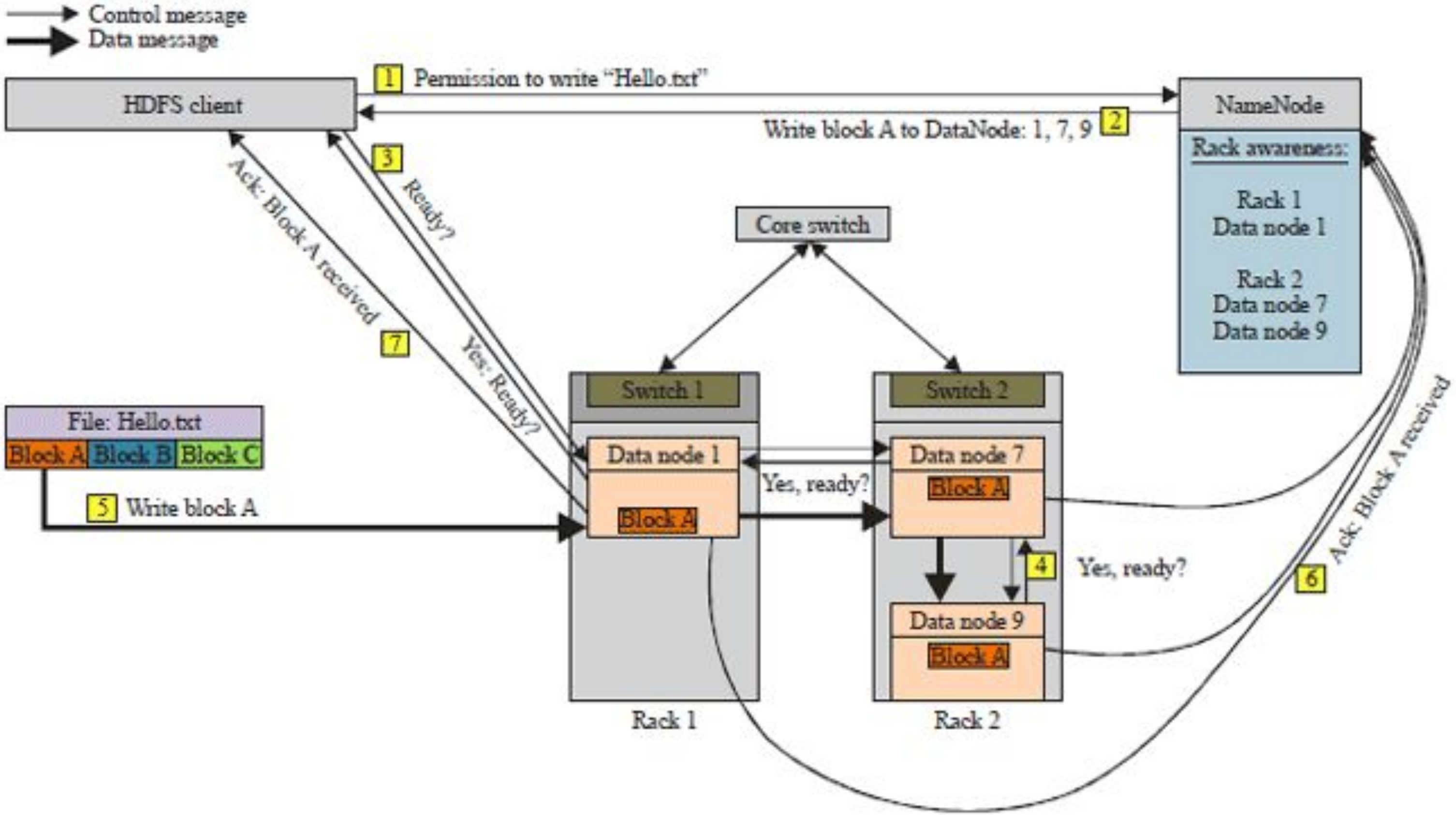


Fig. Datanode and Namenode Internal working

Resource Manager

- Resource Manager is also known as the Global Master Daemon that works on the Master System.
- The Resource Manager **manages the resources for the applications** that are running in a Hadoop Cluster.
- The Resource Manager Mainly consists of 2 things.

1. **Applications Manager**
2. **Scheduler**

- Port Number: 8088

Node Manager

- The Node Manager works on the Slaves System that manages the **memory resource** within the Node and Memory Disk.

- Each Slave Node in a Hadoop cluster has a single NodeManager Daemon running in it. It also sends this monitoring information to the Resource Manager.

- Hadoop Command: yarn-daemon.sh start nodemanager

- Port Number: 8042

Contd...

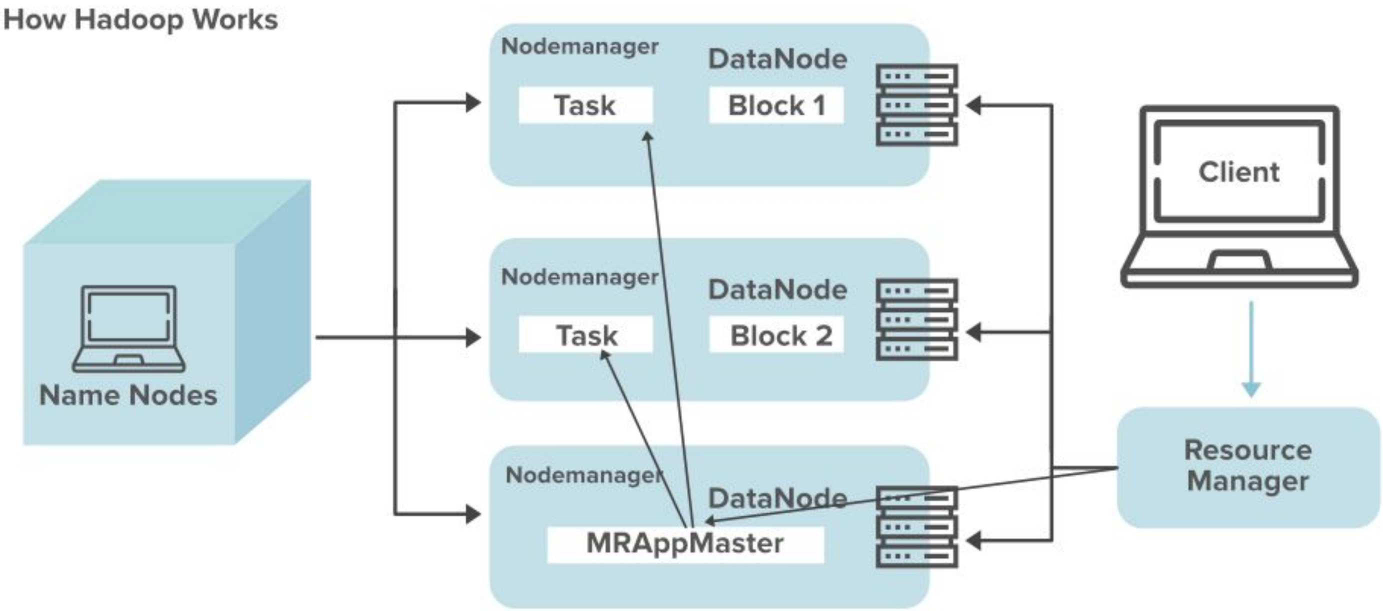
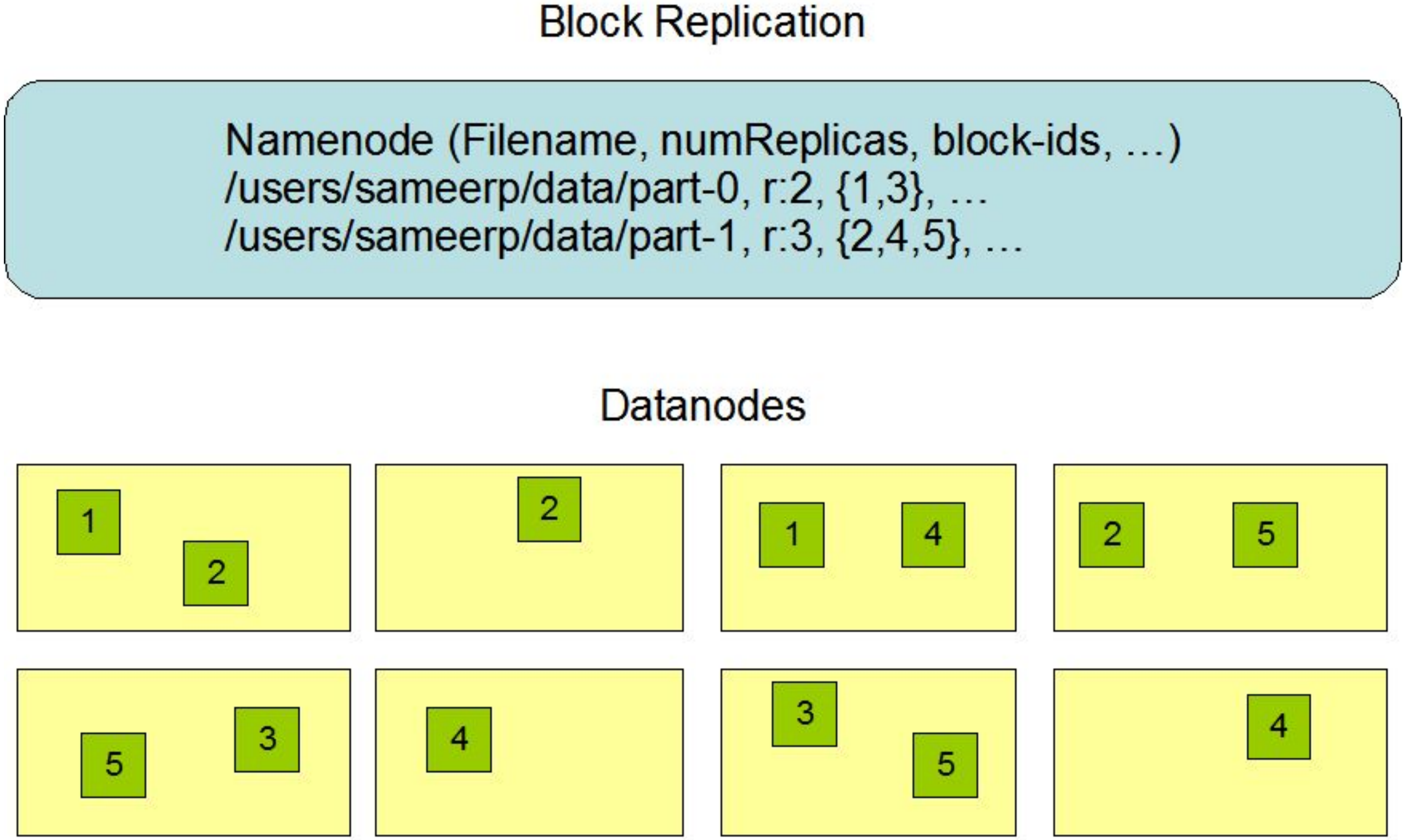


Fig: Internal Processing of Hadoop

Replication

- Replication ensures the availability of the data
- By default, the Replication Factor for Hadoop is set to 3 which can be configured
- We have made 4 file blocks which means that 3 Replica or copy of each file block is made as total of 4×3 = 12 blocks are made for the backup purpose.

Contd...



Block Replication

Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

Datanodes

Rack Awareness

- The rack is nothing but just the physical collection of datanodes in our Hadoop cluster (maybe 30 to 40)

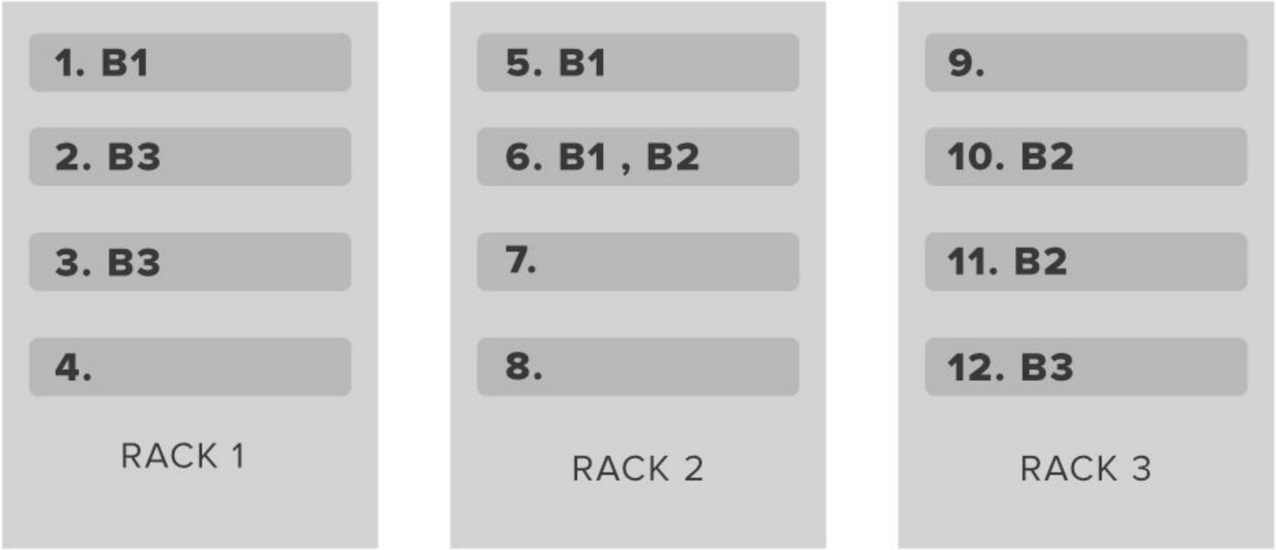
- With the help of this Racks information, Namenode chooses the closest Datanode to achieve the maximum performance while performing the read/write information which reduces the Network Traffic

- **There should not be more than 1 replica on the same Datanode.**
- **More than 2 replica’s of a single block is not allowed on the same Rack.**
- **The number of racks used inside a Hadoop cluster must be smaller than the number of replicas.**

Contd...

Replica Placement via Rack Awareness:

==> Block 1 , Block 2 , Block 3



1. B1
2. B3
3. B3
4.

RACK 1

5. B1
6. B1 , B2
7.
8.

RACK 2

9.
10. B2
11. B2
12. B3

RACK 3

Benefits of Rack Awareness

- We store the data in different Racks so no way to lose our data.

- Helps to **maximize the network bandwidth** because the data blocks transfer within the Racks.

- Improves the **cluster performance** and provides high data availability.

Map Reduce

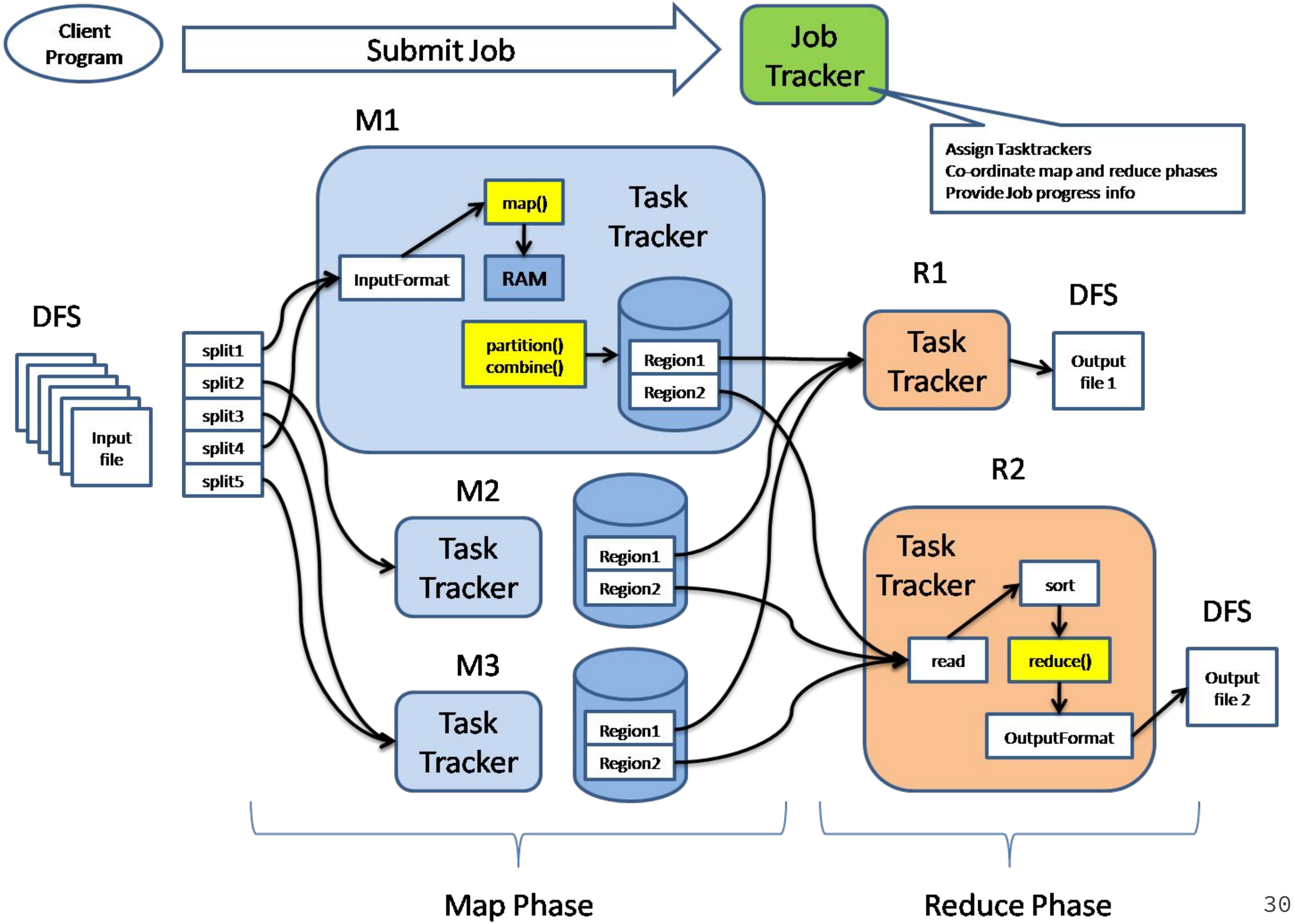
- Programming model and the processing component of Apache Hadoop
- The key reason to perform mapping and reducing is to speed up the job execution of the specific process.

- This can be done by splitting a process into number of tasks, thus **enabling parallelism**

- Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer.

- The map function filter out missing, suspect, or erroneous values

Map-Reduce Framework



Map Phase

- The input data is first split into smaller blocks.

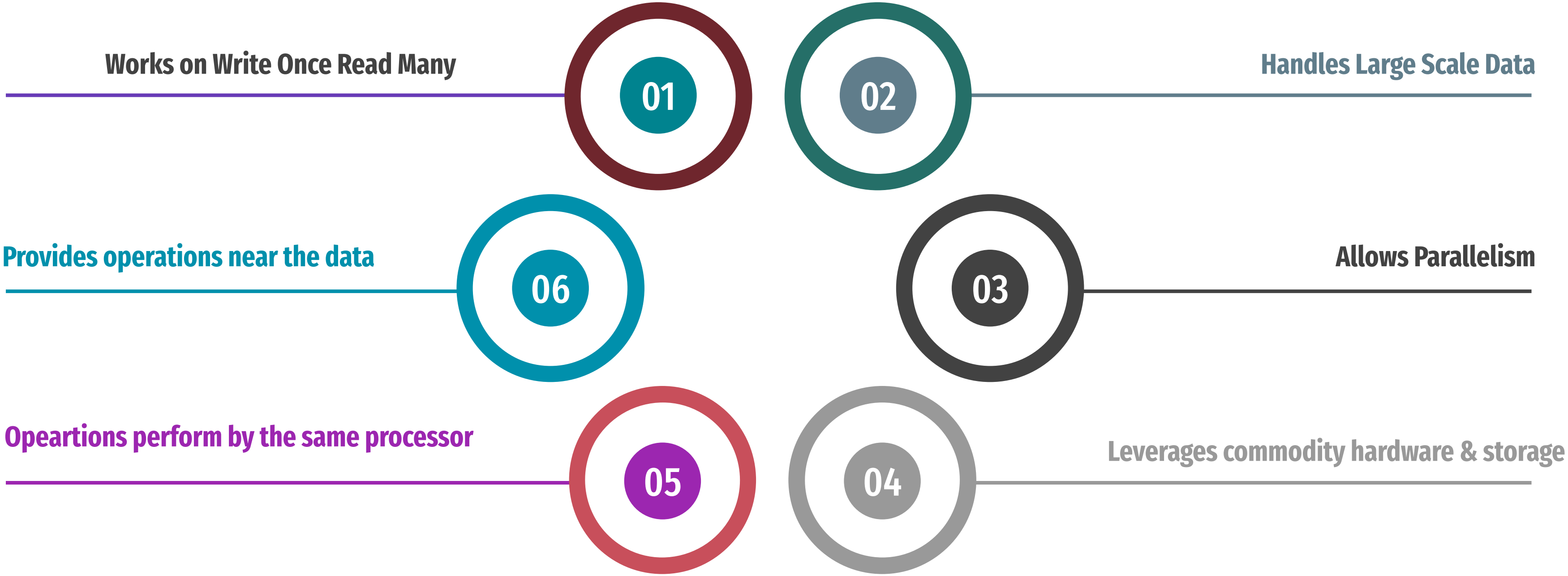
- Each block is then assigned to a mapper for processing.

- The Hadoop framework decides how many mappers to use
  - size of the data to be processed
  - memory block available on each mapper server.

Reduce Phase

- After all the mappers complete processing, the framework shuffles and sorts the results before passing them on to the reducers.
- **A reducer cannot start while a mapper is still in progress.**
- All the map output values have the same key which is assigned to a single reducer, which then aggregates the values for that key.

Map Reduce Characteristic



01 Works on Write Once Read Many

02 Handles Large Scale Data

03 Allows Parallelism

04 Leverages commodity hardware & storage

05 Opeartions perform by the same processor

06 Provides operations near the data

Case Study : Log Data from payment gateway

- **MAP PHASE**

Mapper 1:<Exception A,1>,<Exception B,1>,<Exception A,1>,<Exception C,1>,
<Exception A,1>

Mapper 2:<Exception B,1>, <Exception B, 1>,<Exception A,1>,<Exception A, 1>

Mapper 3:<Exception A,1>, <Exception C, 1>,<Exception A,1>,<Exception B, 1>,
<Exception A, 1>

Mapper 4:<Exception B, 1>, <Exception C, 1>, <Exception C, 1>, <Exception A, 1>

- **COMBINE PHASE**

Combiner 1: <Exception A, 3>, <Exception B, 1>, <Exception C, 1>

Combiner 2: <Exception A, 2> <Exception B, 2>

Combiner 3: <Exception A, 3> <Exception B, 1> <Exception C, 1>

Combiner 4: <Exception A, 1> <Exception B, 1> <Exception C, 2>

Contd...

- **PARTITION PHASE - with combiner**

Reducer 1: <Exception A> {3,2,3,1}
Reducer 2: <Exception B> {1,2,1,1}
Reducer 3: <Exception C> {1,1,2}

- **PARTITION PHASE - without combiner**

Reducer 1: <Exception A> {1,1,1,1,1,1,1,1,1}
Reducer 2: <Exception B> {1,1,1,1,1}
Reducer 3: <Exception C> {1,1,1,1}

- **Reducer Phase**

Reducer 1: <Exception A, 9>
Reducer 2: <Exception B, 5>
Reducer 3: <Exception C, 4>

Case study for high temperature forecast

- **To visualize the way the map works, consider the following sample lines of input data**
  0043011990999991950051512004...*9999999N9+00221+99999999999...*
  0043011990999991950051518004...*9999999N9-00111+99999999999...*
  0043012650999991949032412004...*0500001N9+01111+99999999999...*
  0043012650999991949032418004...*0500001N9+00781+99999999999...*
- **These lines are presented to the map function as the key-value pairs:**
  (0, 006701199099999**1950051507004...9999999N9+00001+99999999999...)**
  (106, 004301199099999**1950051512004...9999999N9+00221+99999999999...)**
  (212, 004301199099999**1950051518004...9999999N9-00111+99999999999...)**
  (318, 004301265099999**1949032412004...0500001N9+01111+99999999999...)**
  (424, 004301265099999**1949032418004...0500001N9+00781+99999999999...)**

Contd...

- **The map function merely extracts the year and the air temperature**

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

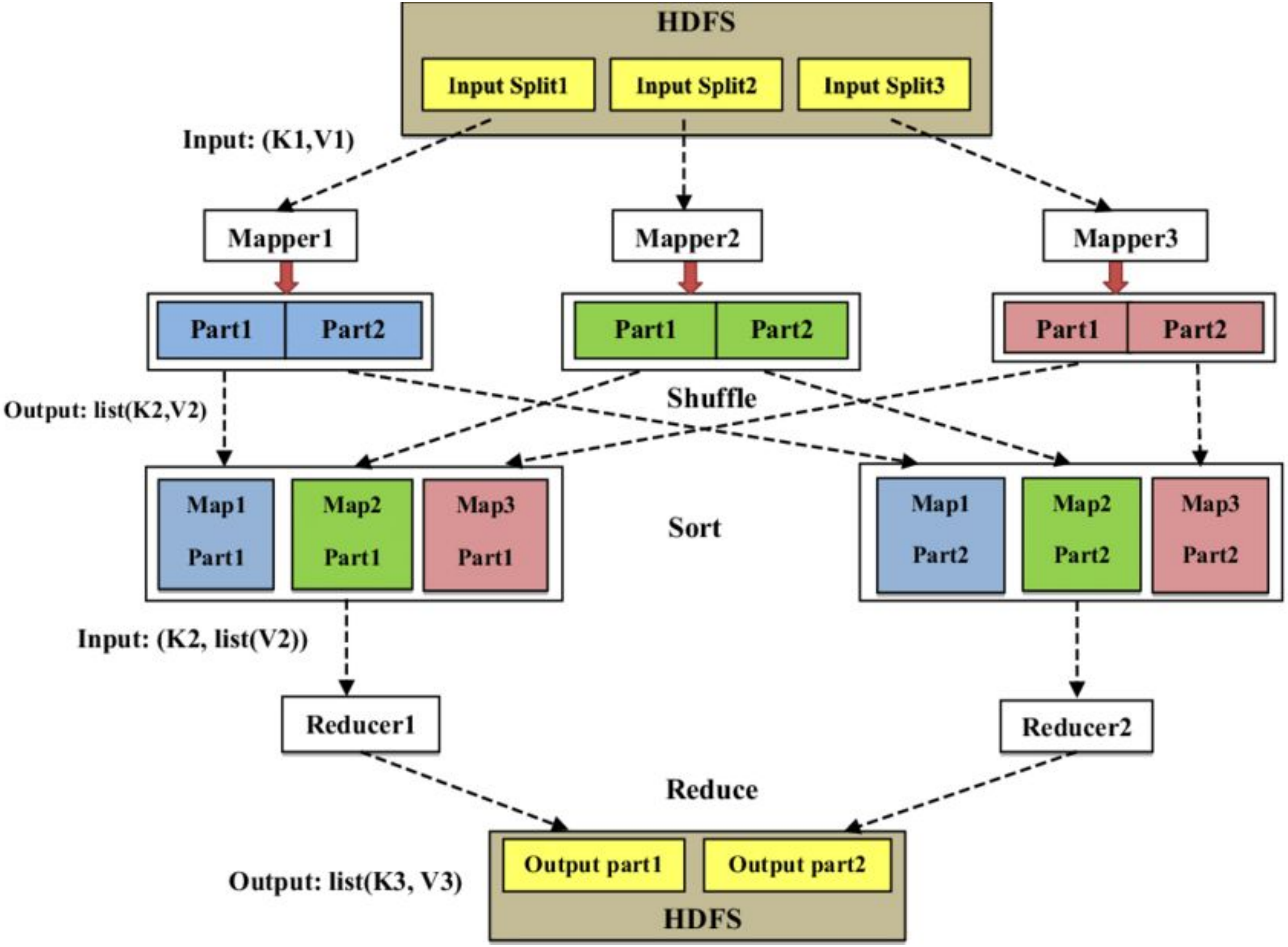
- **This processing sorts and groups the key-value pairs by key.**

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

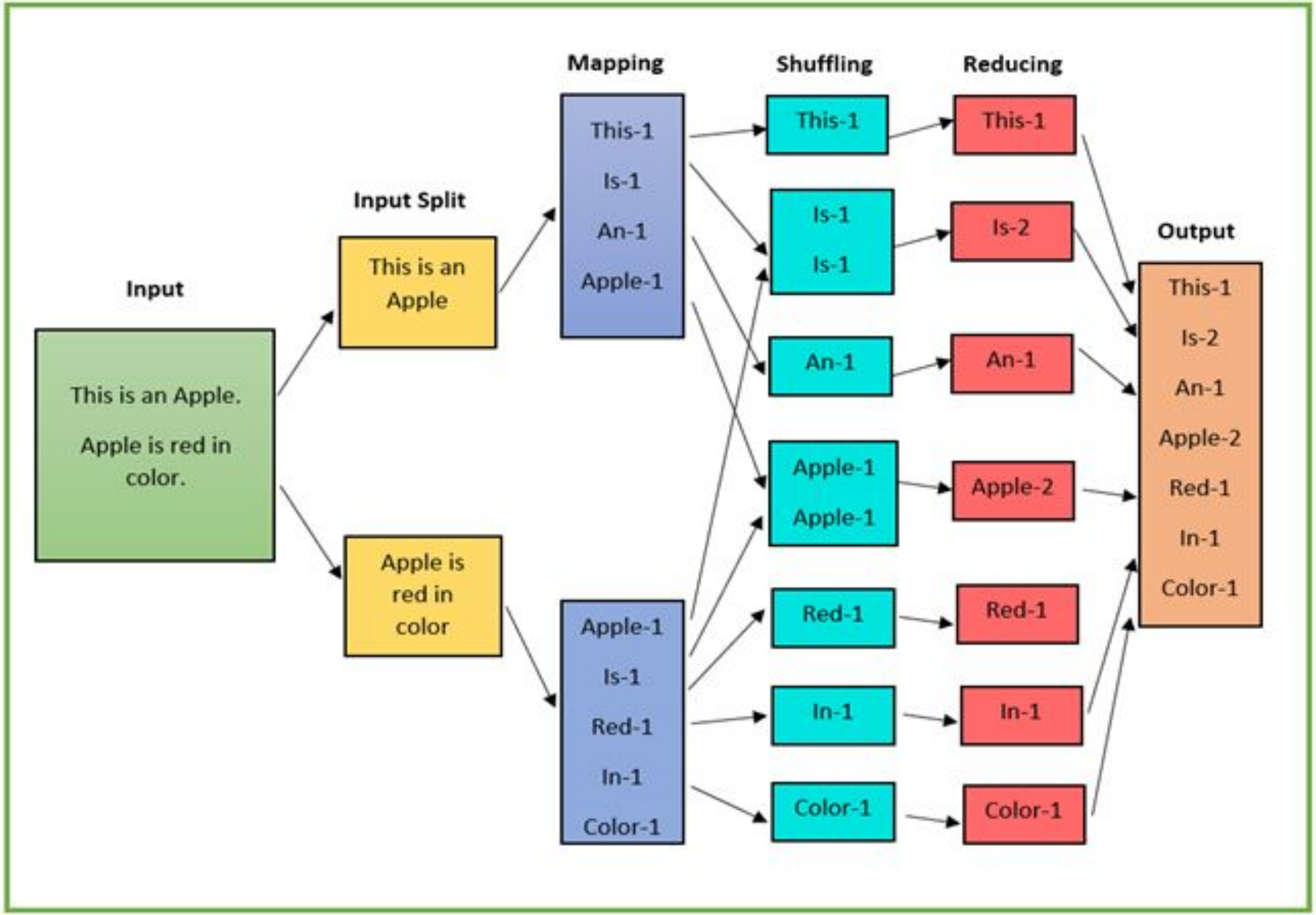
- **All the reduce function has to pick up the maximum temperature recorded in each year.**

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

Contd...

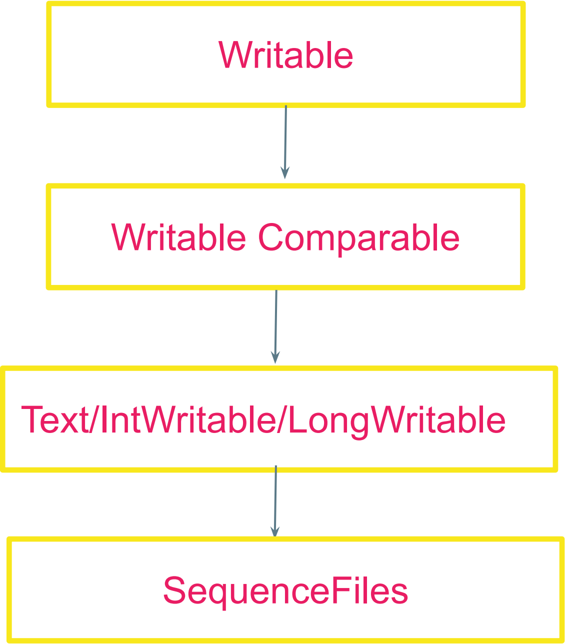

HDFS
Input Split1
Input Split2
Input Split3
Input: (K1,V1)
Mapper1
Mapper2
Mapper3
Part1
Part2
Part1
Part2
Part1
Part2
Output: list(K2,V2)
Shuffle
Map1
Part1
Map2
Part1
Map3
Part1
Sort
Map1
Part2
Map2
Part2
Map3
Part2
Input: (K2, list(V2))
Reducer1
Reducer2
Reduce
Output part1
Output part2
Output: list(K3, V3)
HDFS

Contd...



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

Data Types in Map Reduce



Writable

Writable Comparable

Text/IntWritable/LongWritable

SequenceFiles

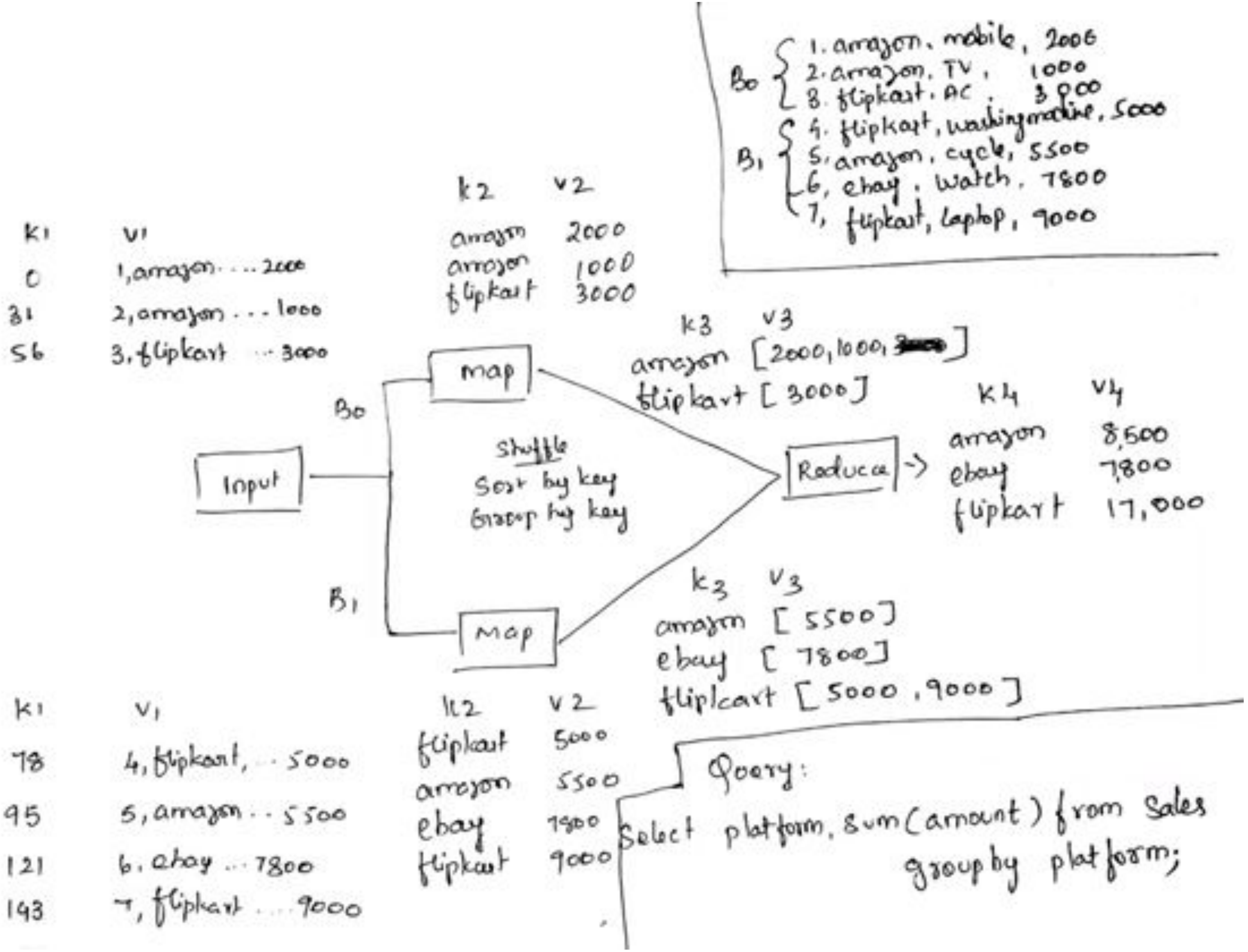
Every data type in hadoop is writable.
It defines a serialization or deserialization protocol.

All key must be of this type.
It defines a sort order.

These are a concrete classes for different data types.

It refers to a binary encoded with a sequence of <key,value> pairs

Example



Bo { 1. amazon, mobile, 2000
2. amazon, TV, 1000
3. flipkart, AC, 3000 }

B1 { 4. flipkart, washingmachine, 5000
5, amazon, cycle, 5500
6, ebay, watch, 7800
7, flipkart, laptop, 9000 }

k1	v1
0	1,amazon....2000
31	2,amazon...1000
56	3,flipkart ...3000

k2	v2
amazon	2000
amazon	1000
flipkart	3000

k3	v3
amazon	[2000,1000, ~~[illegible]~~]
flipkart	[3000]

k4	v4
amazon	8,500
ebay	7,800
flipkart	17,000

k3	v3
amazon	[5500]
ebay	[7800]
flipkart	[5000 ,9000]

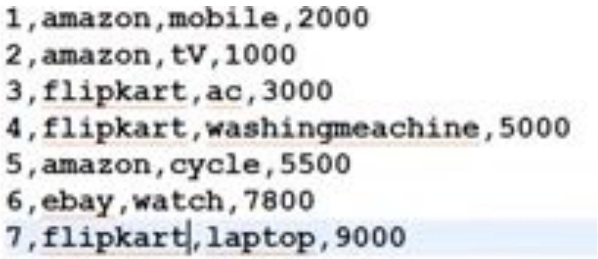
k1	v1
78	4, flipkart, ...5000
95	5, amazon..5500
121	6, ebay ...7800
143	7, flipkart 9000

k2	v2
flipkart	5000
amazon	5500
ebay	7800
flipkart	9000

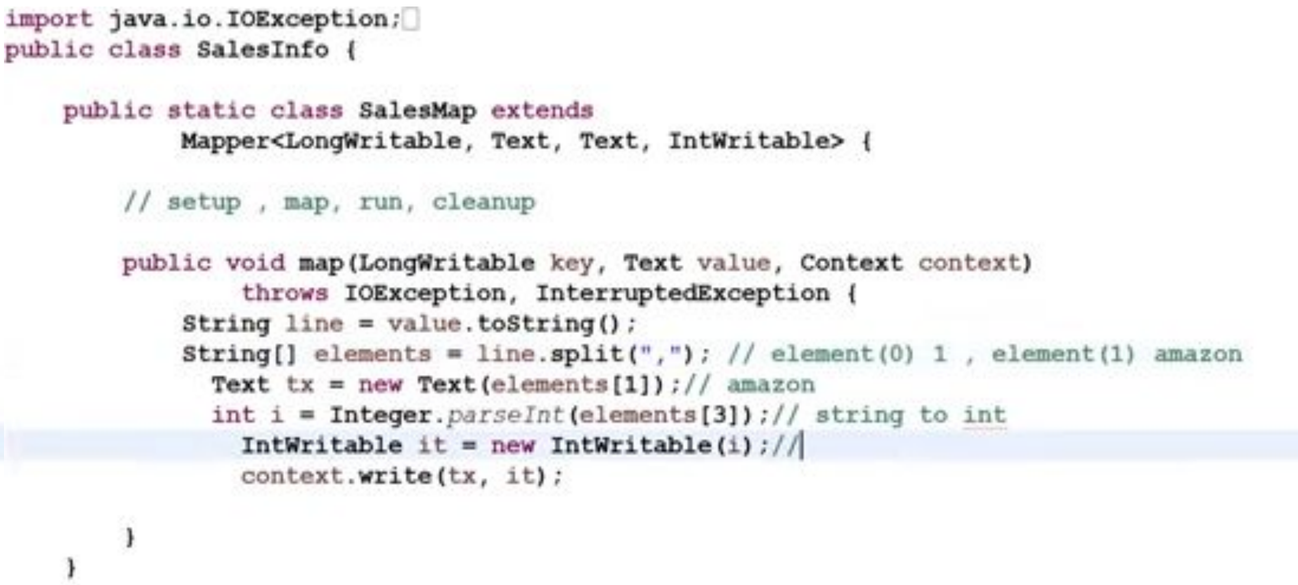
Query:
Select platform, sum(amount) from Sales
group by platform;

Mapper Class

Input.txt

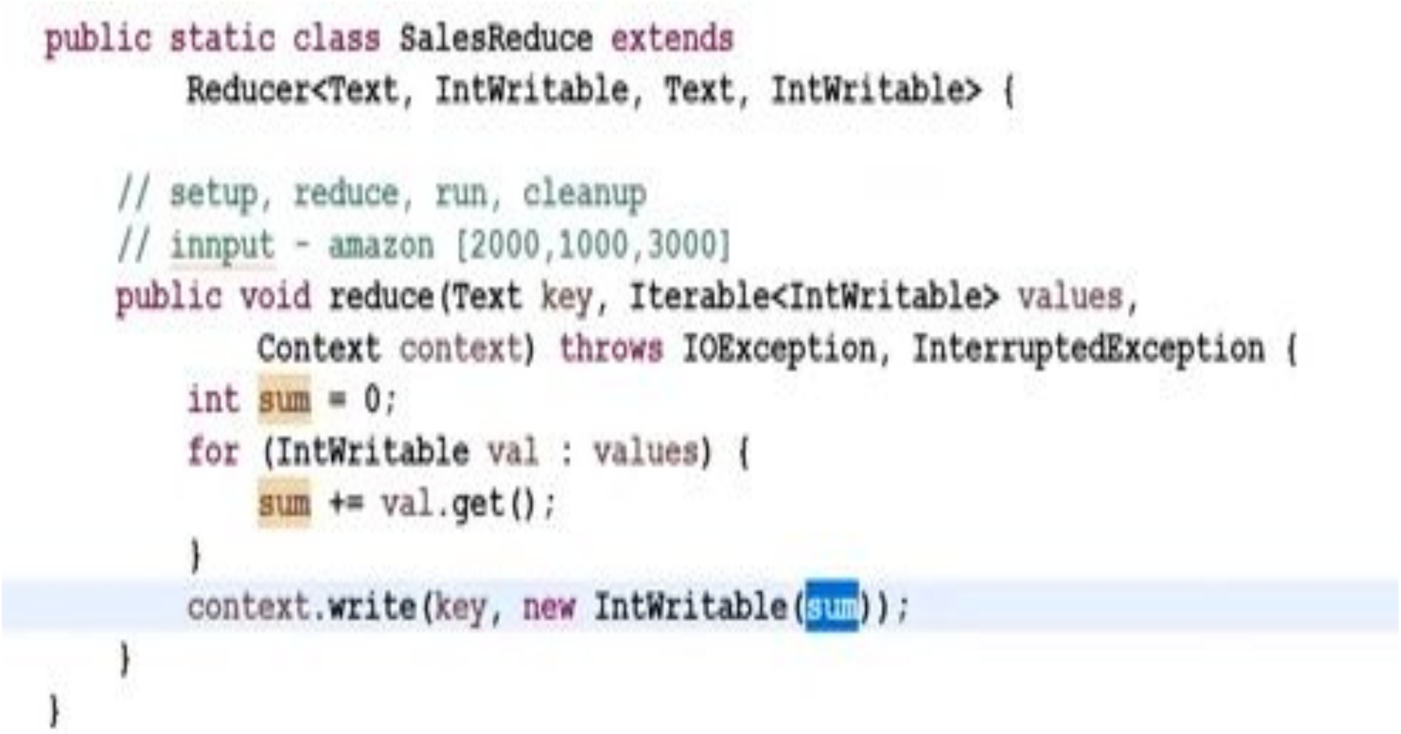


```
1,amazon,mobile,2000
2,amazon,tV,1000
3,flipkart,ac,3000
4,flipkart,washingmeachine,5000
5,amazon,cycle,5500
6,ebay,watch,7800
7,flipkart,laptop,9000
```



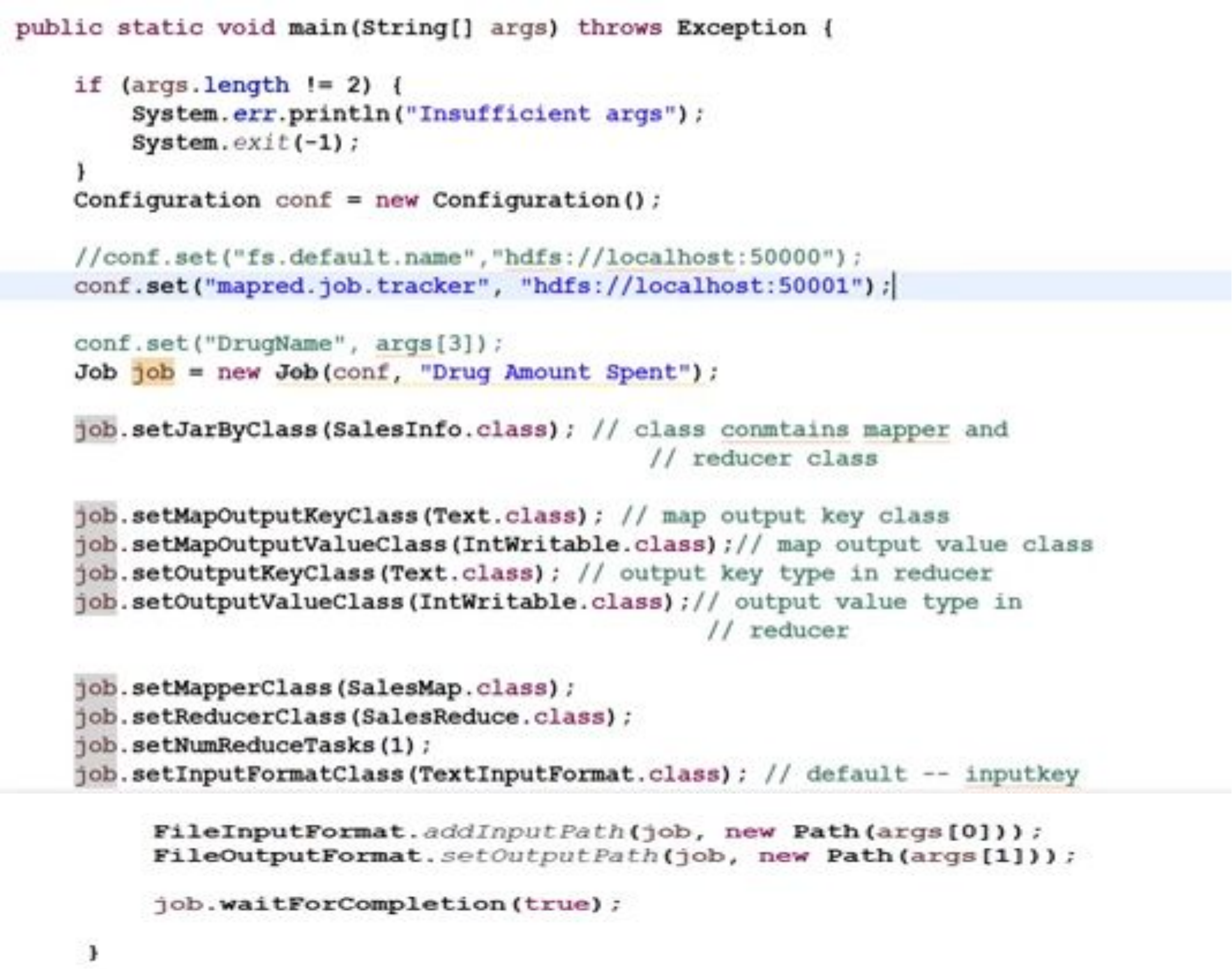
```
import java.io.IOException;
public class SalesInfo {

    public static class SalesMap extends
            Mapper<LongWritable, Text, Text, IntWritable> {

        // setup , map, run, cleanup

        public void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            String line = value.toString();
            String[] elements = line.split(","); // element(0) 1 , element(1) amazon
              Text tx = new Text(elements[1]);// amazon
              int i = Integer.parseInt(elements[3]);// string to int
                IntWritable it = new IntWritable(i);//
                context.write(tx, it);

        }
    }
```

Reducer Class



```
public static class SalesReduce extends
        Reducer<Text, IntWritable, Text, IntWritable> {

    // setup, reduce, run, cleanup
    // innput - amazon [2000,1000,3000]
    public void reduce(Text key, Iterable<IntWritable> values,
            Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

Driver Class



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

    if (args.length != 2) {
        System.err.println("Insufficient args");
        System.exit(-1);
    }
    Configuration conf = new Configuration();

    //conf.set("fs.default.name","hdfs://localhost:50000");
    conf.set("mapred.job.tracker", "hdfs://localhost:50001");

    conf.set("DrugName", args[3]);
    Job job = new Job(conf, "Drug Amount Spent");

    job.setJarByClass(SalesInfo.class); // class conmtains mapper and
                                          // reducer class

    job.setMapOutputKeyClass(Text.class); // map output key class
    job.setMapOutputValueClass(IntWritable.class);// map output value class
    job.setOutputKeyClass(Text.class); // output key type in reducer
    job.setOutputValueClass(IntWritable.class);// output value type in
                                                 // reducer

    job.setMapperClass(SalesMap.class);
    job.setReducerClass(SalesReduce.class);
    job.setNumReduceTasks(1);
    job.setInputFormatClass(TextInputFormat.class); // default -- inputkey

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

        job.waitForCompletion(true);

 }
```

Serialization and Deserialization

- *Serialization is the process of turning structured objects into a byte stream for transmission* over a network or for writing to persistent storage.

- *Deserialization is the reverse* process of turning a byte stream back into a series of structured objects.

- Serialization appears in two quite distinct areas of distributed data processing
  - inter-process communication and
  - persistent storage

Contd...

- The RPC protocol uses serialization to render the message into a binary stream to be sent to the remote node, which then deserializes the binary stream into the original message
- **The lifespan of an RPC is less than a second, whereas persistent data may be read years after it was written**
- We want the storage format to be
  - **compact** (to make efficient use of storage space),
  - **fast** (overhead in reading/writing terabytes of data is minimal),
  - **extensible** (transparently read data written in an older format),
  - **interoperable** (read/write persistent data using different languages).

Avro

- Apache Avro is a language-neutral data serialization system.
- The project was created by Doug Cutting to address the major downside of Hadoop Writables: lack of language portability.
- Having a data format that can be processed by many languages (currently C, C++, Java, Python, and Ruby)
- **Avro schemas are usually written in JSON, and data is usually encoded using a binary format**

YARN

- In the previous version Hadoop 1.0, which is also known as MapReduce version 1 (MRV1) use to perform both the task of process and resource management by itself.

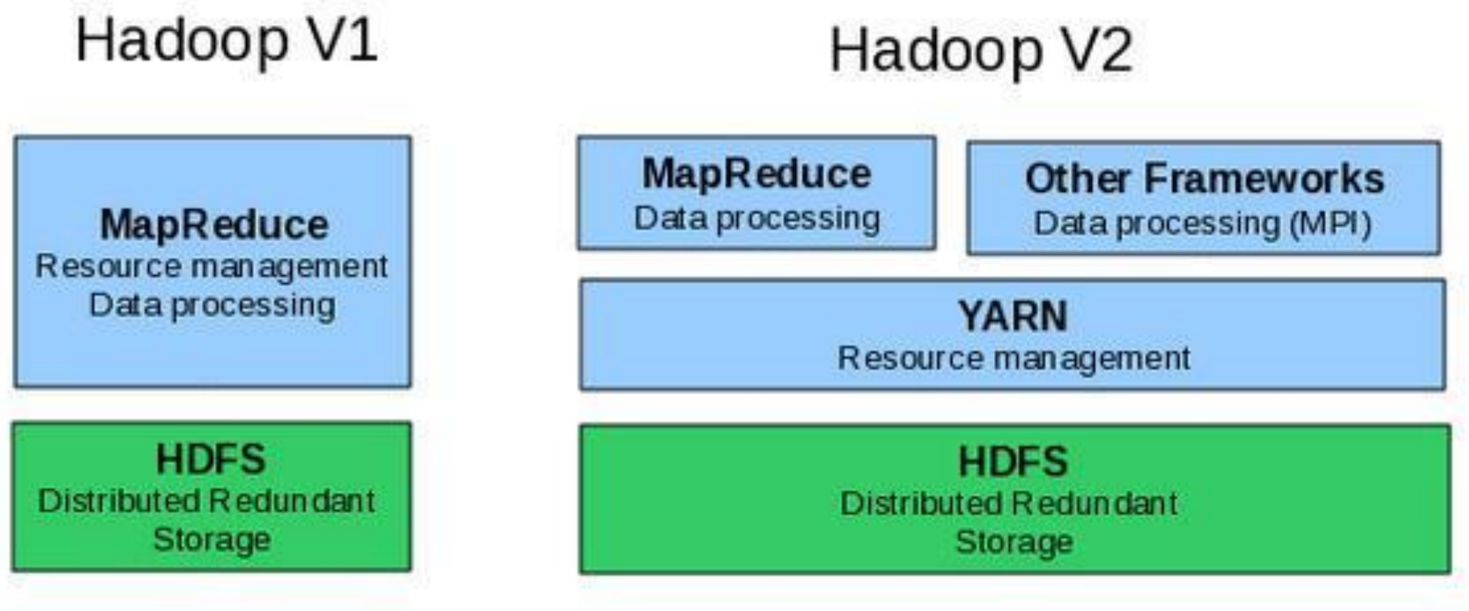
- **YARN is the acronym for Yet Another Resource Negotiator.**

- YARN, a vital core component in its successor Hadoop version 2.0 which was introduced in the year 2012 by Yahoo and Hortonworks

- The basic idea behind this is separating MapReduce from Resource Management and Job scheduling instead of a single master.

- In Hadoop 2.0, The concept of Application Master and Resource Manager was introduced by YARN

Contd....



Hadoop V1

MapReduce
Resource management
Data processing

HDFS
Distributed Redundant Storage

Hadoop V2

MapReduce
Data processing

Other Frameworks
Data processing (MPI)

YARN
Resource management

HDFS
Distributed Redundant Storage

Features of YARN

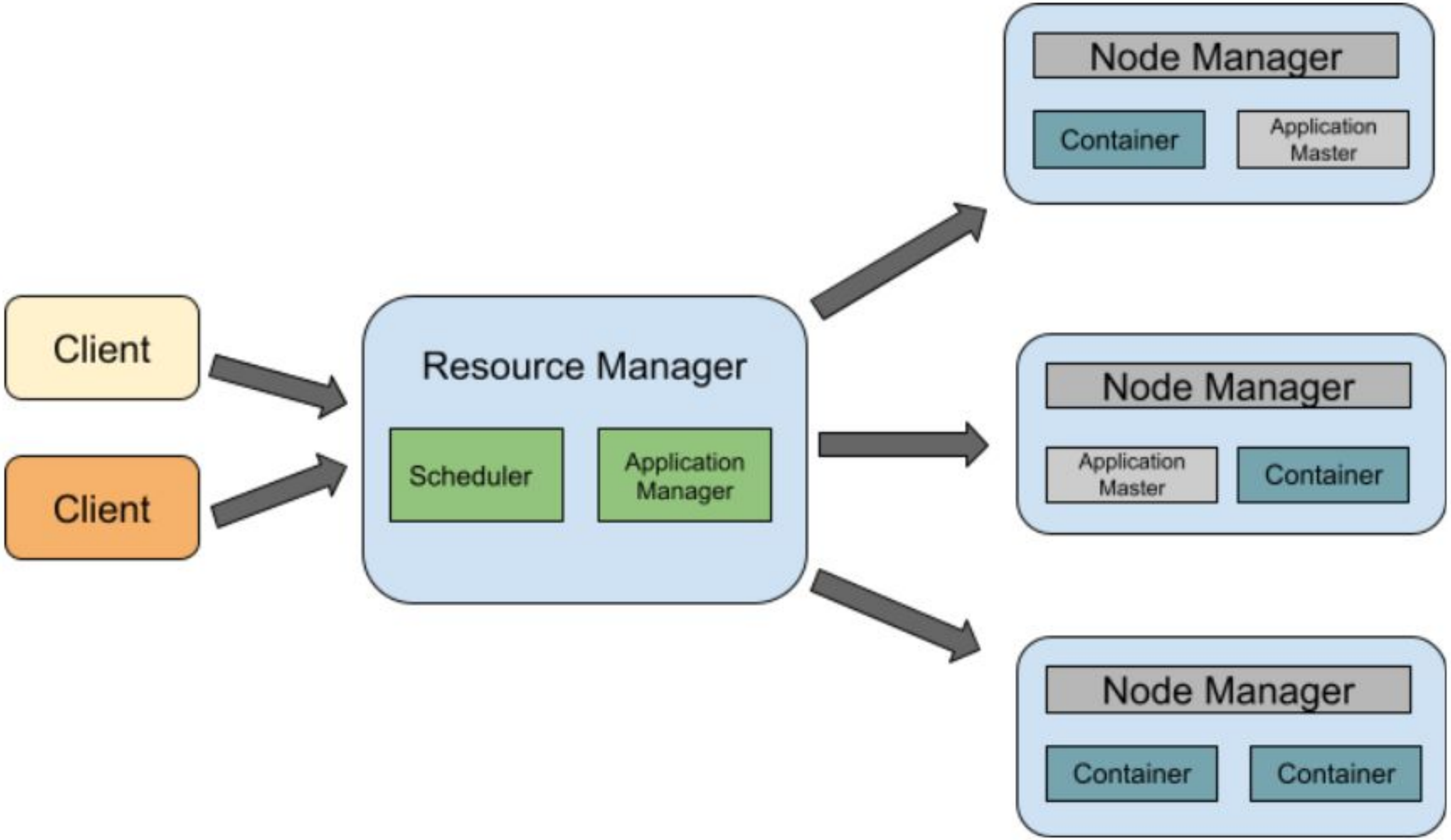
- **Multi-tenancy:** YARN has allowed access to multiple data processing engines such as batch processing engine, stream processing engine, interactive processing engine, graph processing engine and much more

- **Cluster Utilization**

- **Compatibility**

- **Scalability**

YARN Architecture



Components of YARN

- **Container**
  - **In the Container, user can find physical resources like a disk on a single node, CPU cores, RAM, Network**.
  - Data about the dependencies, security tokens, environment variables which are maintained as a record known as Container Launch Context (CLC) which is used to invoke containers
- **Application Master**
  - In a framework, when a single job is submitted, it is called an application.
  - Monitoring the application progress, application status tracking, negotiation of resources with resource manager is the responsibility of the application manager.

Contd...

- **Node Manager**
    - The node manager takes care of individual nodes in the Hadoop cluster and also manages containers related to each specific node.
    - Sends each node’s health status to the Resource Manager.
    - The node manager also creates a container process when requested by the Application master.
    - It can also kill containers if directed by the Resources manager.

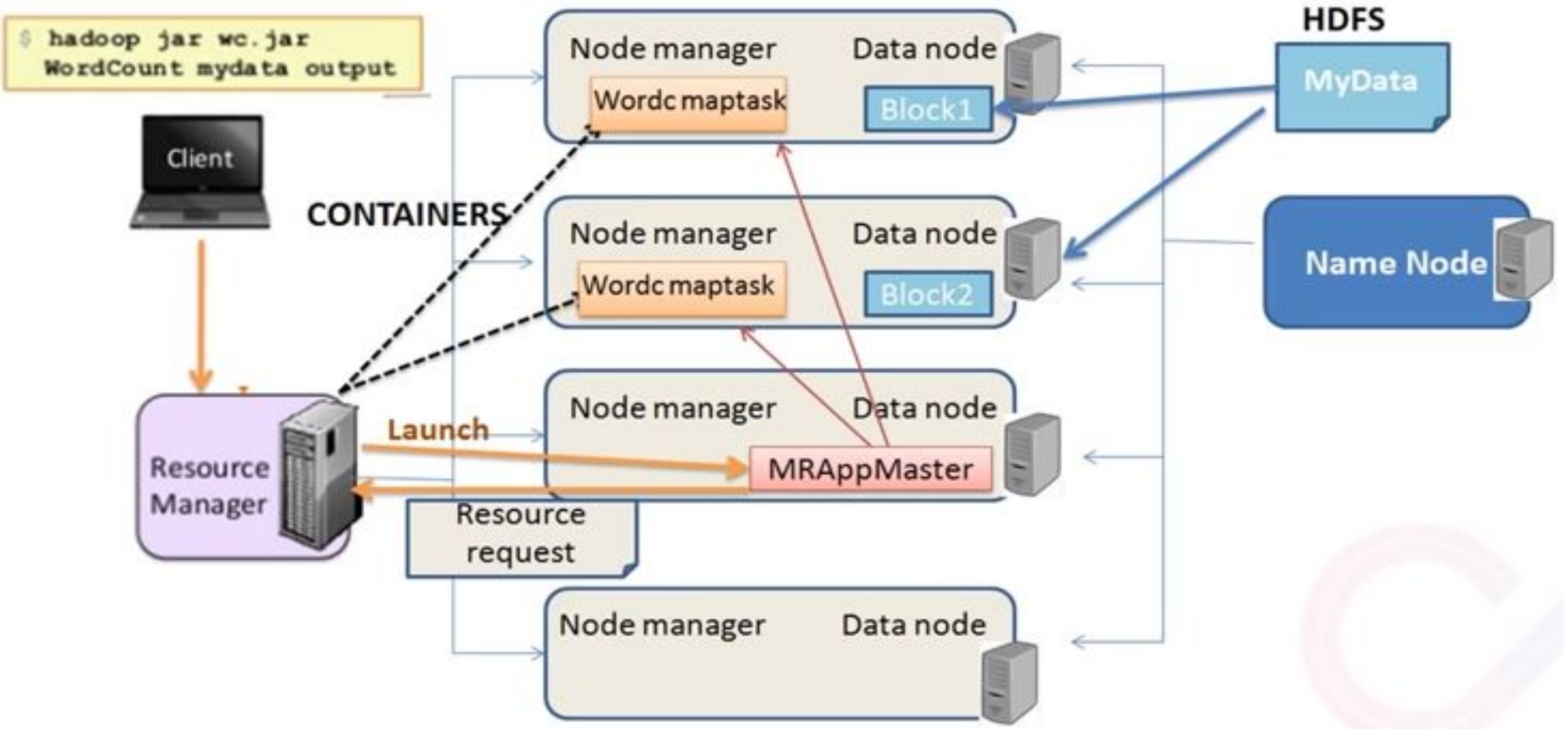
Contd...

- **Resource Manager**
  - Resource management and assignment of all the apps is the responsibility of Resource Manager and is the master daemon of YARN.
  - Requests received by the resource manager are forwarded to the corresponding node manager
  - Utilization of Cluster is optimized
  - Highest authority for allocation of available resources.

Contd...

- **Application Manager**
    - It first verifies and validates the submitted application's specifications and may reject the applications if there are not enough resources available.
    - No other application exists with the same ID
    - Forwards application after validation to the scheduler
    - Observes the states of applications and manages finished applications to save some Resource Manager's memory.
- **Scheduler**
    - There is no other task performed by scheduler like no restart of the job after failing, tracking or monitoring of tasks.

Word Count Example

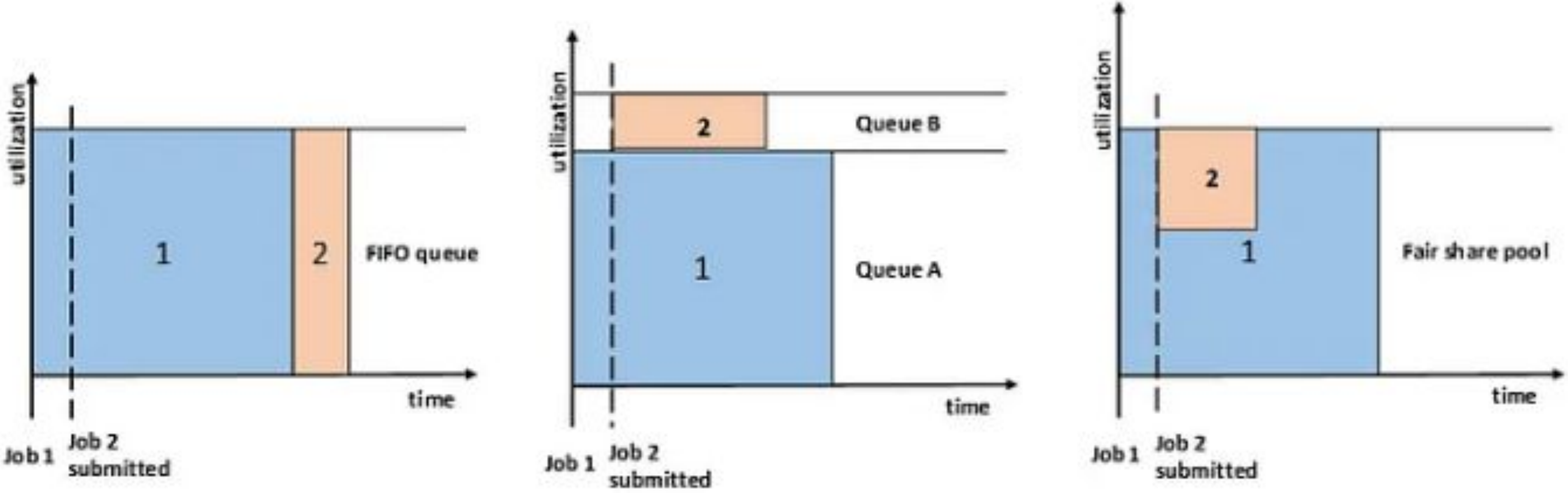


```
$ hadoop jar wc.jar
  WordCount mydata output
```

Types of Scheduler

- **FIFO Scheduler**
    - **First In First Out** is the scheduling policy
    - Irrespective of the size and priority, it gives more preferences to the application coming first than those coming later
    - Queue is used to place and execute incoming applications in the order of their submission
    - It does not take into account the balance of resource allocation between the long applications and short applications.
    - Starvation Problem

Contd....

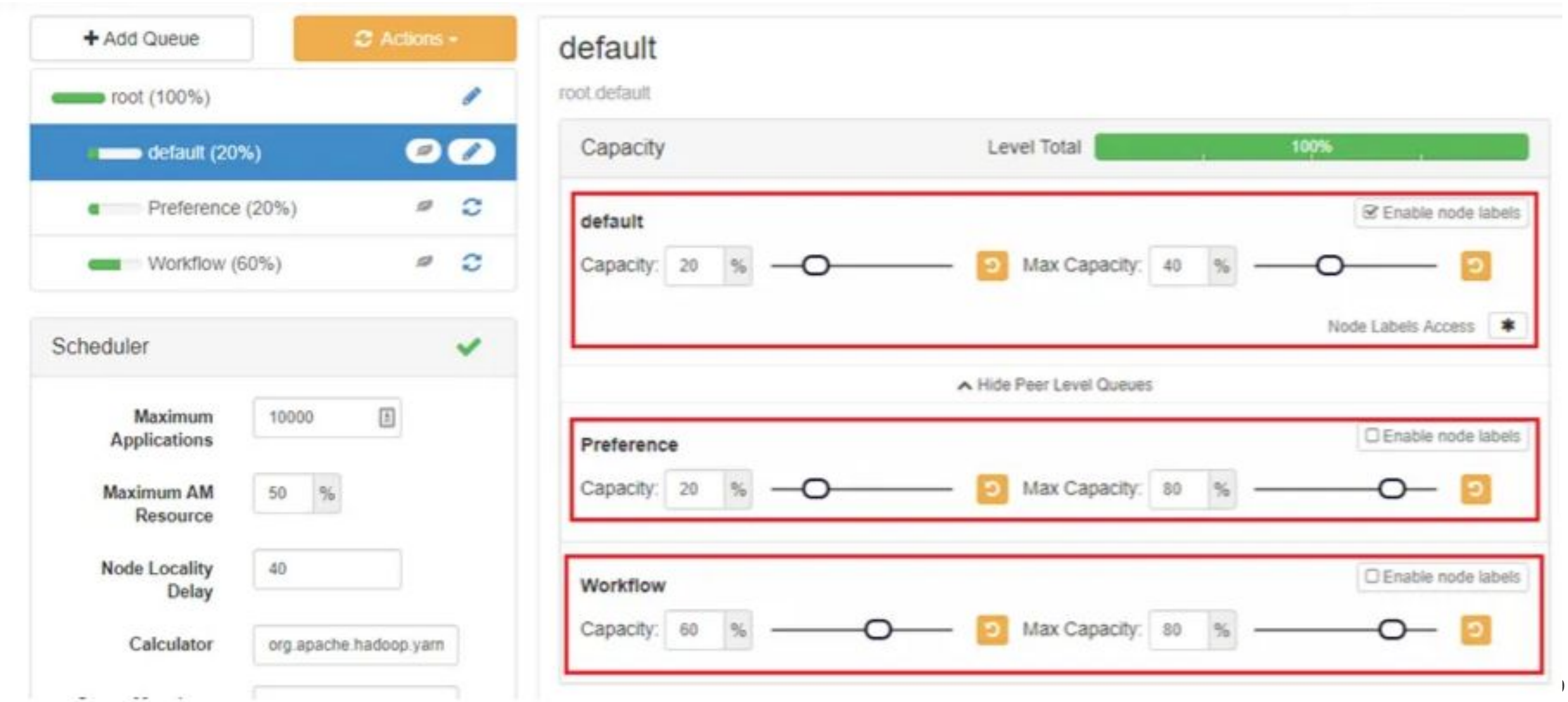


(a) FIFO scheduler

(b) Capacity Scheduler

(c) Fair Scheduler

Contd....



root (100%)
default (20%)
Preference (20%)
Workflow (60%)

Scheduler ✓

Maximum Applications: 10000

Maximum AM Resource: 50 %

Node Locality Delay: 40

Calculator: org.apache.hadoop.yarn

default

root.default

Capacity — Level Total: 100%

default — ☑ Enable node labels
Capacity: 20 % — Max Capacity: 40 %
Node Labels Access: *

Preference — ☐ Enable node labels
Capacity: 20 % — Max Capacity: 80 %

Workflow — ☐ Enable node labels
Capacity: 60 % — Max Capacity: 80 %

Contd….

- **Capacity Scheduler**
    - It is designed to run Hadoop applications in a shared, multi-node cluster while maximizing the throughput and the utilization of the cluster.
    - **A queue hierarchy contains three types of queues that are root, parent, and leaf.**
    - **The root queue represents the cluster itself, parent queue represents organization/group or sub-group and the leaf accepts application submission.**
    - When there is a demand for the free resources that are available on the queue who has completed its task, by the queues running below capacity, then these resources will be assigned to the applications on queues running below capacity. This provides elasticity for the organization in a cost-effective manner.

Contd...

- **Pros**
  - Maximizes the utilization of resources and throughput in the Hadoop cluster.
  - Provides elasticity for groups or organizations in a cost-effective manner.
  - Gives capacity guarantees and safeguards to the organization utilizing cluster.
- **Cons**
  - Complex amongst the other scheduler.

Contd...

- **Fair Scheduler**
    - With Fair Scheduler, there is no need for reserving a set amount of capacity because it will dynamically balance resources between all running applications.

    - It assigns resources to applications in such a way that all applications get, on average, an equal amount of resources over time.

    - When an app is present in the queue, then the app gets its minimum share, but when the queue doesn’t need its full guaranteed share, then the excess share is split between other running applications.