Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

# CET4001B Big Data Technologies

**School of Computer Engineering and Technology**

# Unit-III

## Introduction to Hadoop

- **What is Hadoop:**
- **Hadoop Overview**
- **HDFS** - Introduction, Concepts, Design, HDFS interfaces, HDFS Read Architecture, HDFS Write Architecture,
- Managing Resources and Applications with Hadoop YARN (Yet Another Resource Negotiator)
- **Introduction to MAP REDUCE Programming,** Mapper, Reducer, Combiner, Partitioner, Searching, Sorting , Job Chaining.

# Why Hadoop?

- Big Data analytics and the Apache Hadoop open source project are rapidly emerging as the preferred solution to address business and technology trends that are disrupting traditional data management and processing.
- **Enterprises can gain a competitive advantage by being early adopters of big data analytics.**

Gartner

# Industries Are Embracing Big Data

## Retail

- CRM – Customer Scoring
- Store Siting and Layout
- Fraud Detection / Prevention
- Supply Chain Optimization

## Advertising & Public Relations

- Demand Signaling
- Ad Targeting
- Sentiment Analysis
- Customer Acquisition

## Financial Services

- Algorithmic Trading
- Risk Analysis
- Fraud Detection
- Portfolio Analysis

## Media & Telecommunications

- Network Optimization
- Customer Scoring
- Churn Prevention
- Fraud Prevention

## Manufacturing

- Product Research
- Engineering Analytics
- Process & Quality Analysis
- Distribution Optimization

## Energy

- Smart Grid
- Exploration

## Government

- Market Governance
- Counter-Terrorism
- Econometrics
- Health Informatics

## Healthcare & Life Sciences

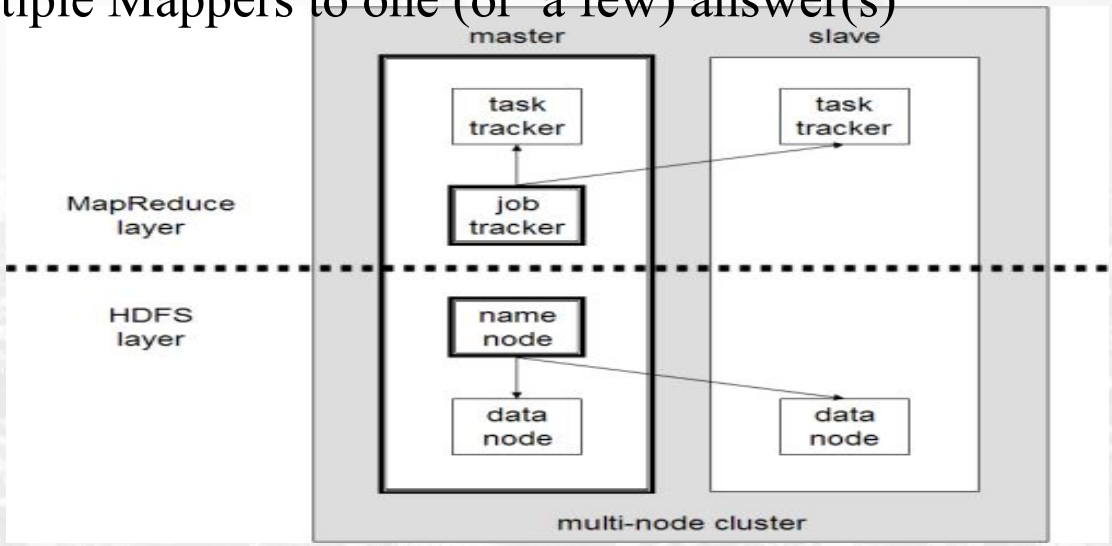
- Pharmaco-Genomics
- Bio-Informatics
- Pharmaceutical Research
- Clinical Outcomes Research

# What is Hadoop?

- Hadoop is a software framework for distributed processing of large datasets across large clusters of computers
- A scalable fault-tolerant distributed system for data storage and processing
- Operates on unstructured and structured data .
- A large and active ecosystem
- Open source under the friendly Apache License.
- Hadoop is open-source implementation for Google MapReduce
- Hadoop is based on a simple data model, *any data will fit*

Contd..

- Hadoop framework consists on two main layers
  - **Hadoop Distributed file system (HDFS) :** self-healing, high-bandwidth clustered storage, redundant, distributed file system optimized for large files
  - **Execution engine (MapReduce) :** fault-tolerant distributed processing, Programming model for processing sets of data Mapping inputs to outputs and reducing the output of multiple Mappers to one (or a few) answer(s)



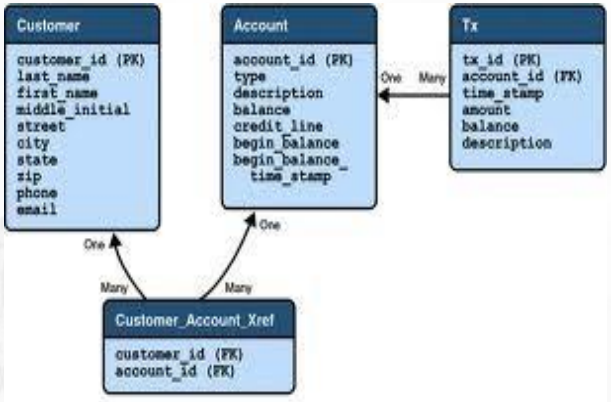
# Hadoop Infrastructure

- Hadoop is a ***distributed*** system like ***distributed databases***

- **However, there are several key differences between the two infrastructures**
  - **Data model**
  - **Computing model**
  - **Cost model**
  - **Design objectives**

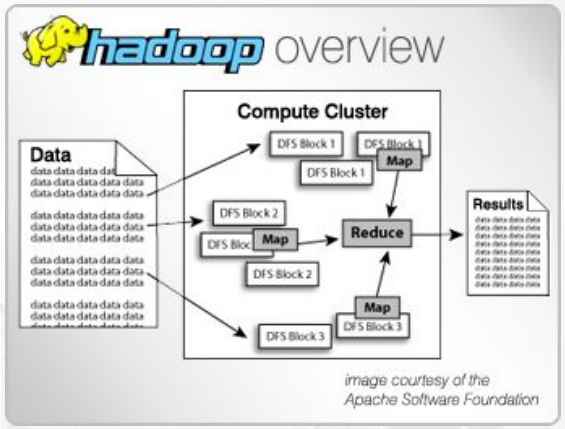
# How Data Model is Different?

## Distributed Databases




- Deal with tables and relations
- Must have a schema for data
- Data fragmentation & partitioning

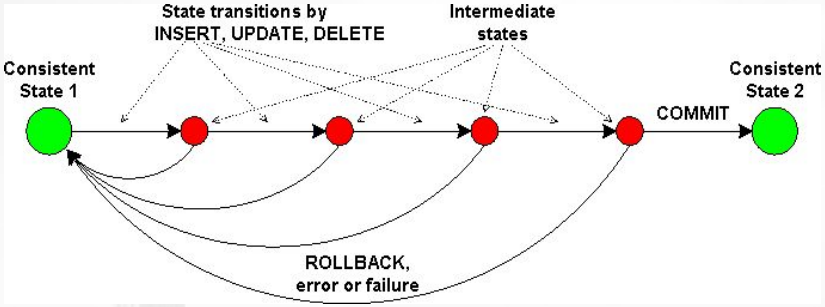
## Hadoop




- Deal with flat files in any format
- No schema for data
- Files are divide automatically into blocks

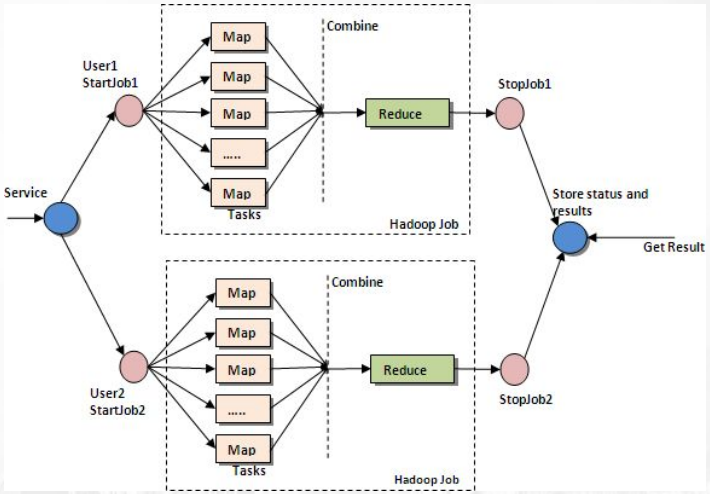
# How Computing Model is Different?

## Distributed Databases




- Notion of a transaction
- Transaction properties ACID
- Distributed transaction

## Hadoop

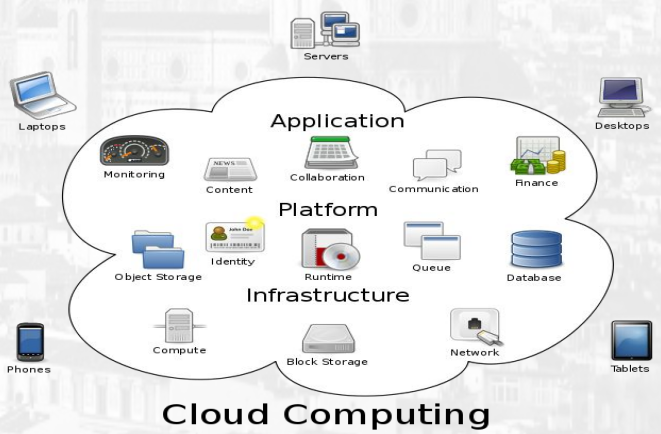



- Notion of a job divided into tasks
- Map-Reduce computing model
- Every task is either a map or reduce

# Summary: Hadoop vs. Other Systems

| | Distributed Databases | Hadoop |
|---|---|---|
| **Computing Model** | - Notion of transactions<br>- Transaction is the unit of work<br>- ACID properties, Concurrency control | - Notion of jobs<br>- Job is the unit of work<br>- No concurrency control |
| **Data Model** | - Structured data with known schema<br>- Read/Write mode | - Any data will fit in any format<br>- (un)(semi)structured<br>- ReadOnly mode |
| **Cost Model** | - Expensive servers | - Cheap commodity machines |
| **Fault Tolerance** | - Failures are rare<br>- Recovery mechanisms | - Failures are common over thousands of machines<br>- Simple yet efficient fault tolerance |
| **Key Characteristics** | - Efficiency, optimizations, fine-tuning | - Scalability, flexibility, fault tolerance |

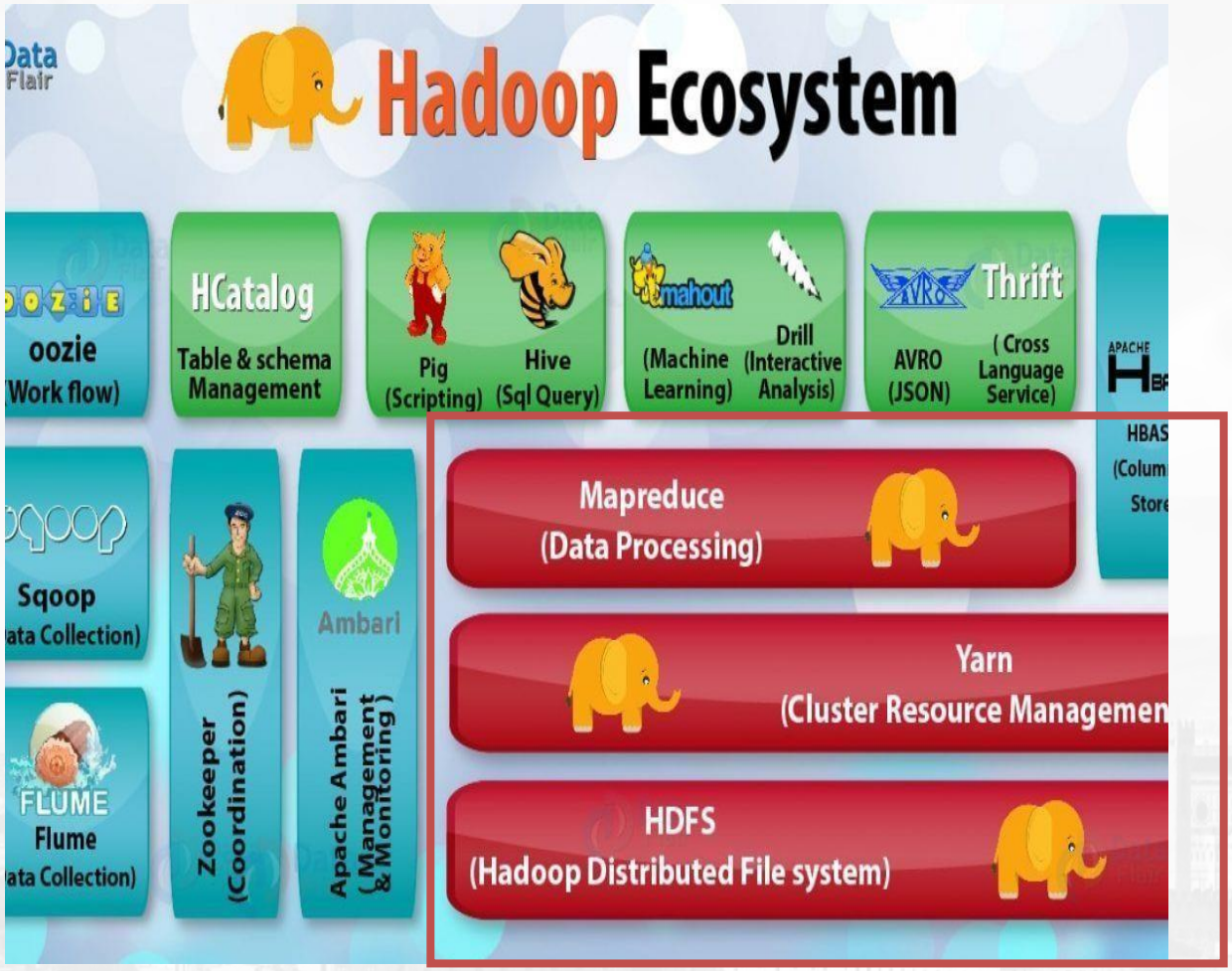
- **Cloud Computing**
  - A computing model where any computing infrastructure can run on the cloud
  - Hardware & Software are provided as remote services
  - Elastic: grows and shrinks based on the user's demand
  - Example: Amazon EC2



# Features of hadoop.

- **Open Source** – Hadoop is an open source framework which means it is available free of cost. Also, the users are allowed to change the source code as per their requirements.
- **High Availability** – The data stored in Hadoop is available to access even after the hardware failure. In case of hardware failure, the data can be accessed from another node.
- **Distributed Processing** – Hadoop supports distributed processing of data i.e. faster processing. The data in Hadoop HDFS is stored in a distributed manner and MapReduce is responsible for the parallel processing of data.
- **Reliability** – Hadoop stores data on the cluster in a reliable manner that is independent of machine. So, the data stored in Hadoop environment is not affected by the failure of the machine.
- **Fault Tolerance** – Hadoop is highly fault-tolerant. It creates three replicas for each block (default) at different nodes.
- **Scalability** – It is compatible with the other hardware and we can easily add/remove the new hardware to the nodes.

- Hadoop is a  software framework from Apache Software Foundation that is used to store and process Big Data.
- Main Components of Hadoop EcoSystem :

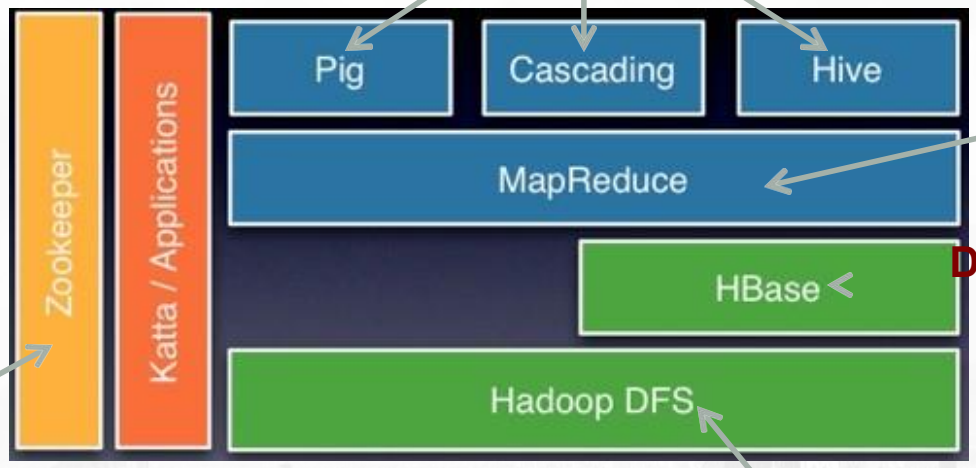



Application Layer

Resource Manager Layer

Storage layer

# Hadoop: Big Picture




**High-level languages**

**Execution engine**

**Distributed light-weight DB**

**Centralized tool for coordination**

**Distributed File system**

HDFS + MapReduce are enough to have things working

# Hadoop Architecture

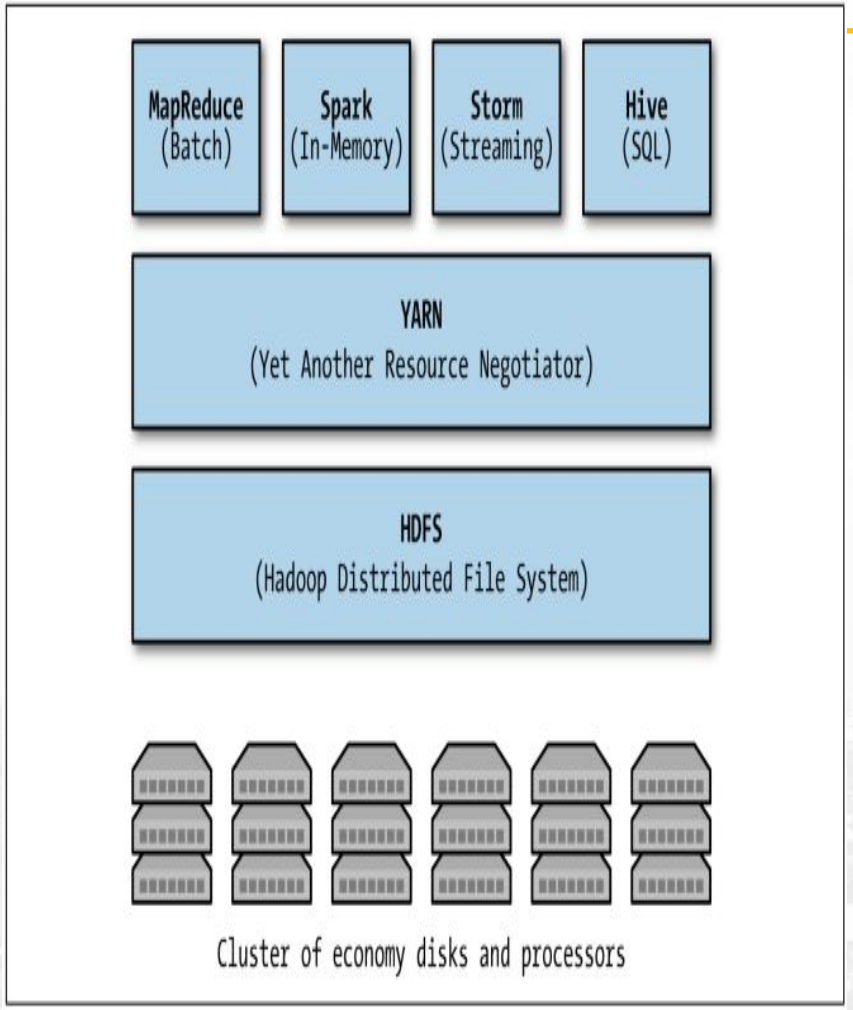



*Figure 2-1. Hadoop is made up of HDFS and YARN*

- Hadoop is composed of **two primary components** : **HDFS and YARN.**
- HDFS (**DFS**) is the Hadoop Distributed File System, **responsible** for managing data stored on disks across the cluster and built on top of a native (ext3, xfs, etc..) file system.
- YARN acts as a **cluster resource manager**, allocating computational assets (processing availability and memory on worker nodes) to applications that wish to perform a distributed computation.
- The **architectural stack in** figure 2.1 shows that the original Map‑ Reduce application is **implemented** on top of YARN as well as other new distributed computation applications like the graph processing engine Apache Giraph, and the in-memory computing platform Apache Spark.
- HDFS and YARN work in concert **to minimize** the amount of network traffic in the cluster primarily by **ensuring** that data is local to the required computation.
- Duplication of both data and tasks ensures fault tolerance, recoverability, and consistency. Moreover, the cluster **is centrally managed** to provide scalability and to abstract lowlevel **clustering programming** details.

# Contd…

- HDFS and YARN are flexible and in HDFS are ‘write once’.
- HDFS is optimized for large, streaming reads of files.
- Like a good OS, other **data storage systems** aside from HDFS **can be integrated** into the Hadoop framework such as Amazon S3 or Cassandra.
- Alternatively, data storage systems can be built directly on top of HDFS to provide more features than a simple file system.

Eg. **HBase** is a columnar data store built on top of HDFS and is one the most advanced analytical applications that leverage distributed storage.

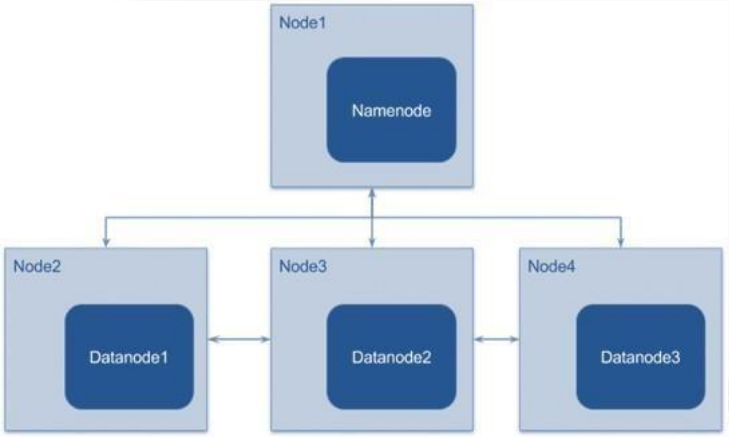
- In earlier versions of Hadoop, applications that wanted to leverage distributed computing on a Hadoop cluster had to translate user-level implementations into MapReduce jobs.
- However, YARN now allows richer abstractions of the cluster utility, making new data processing applications for machine learning, graph analysis, SQL-like querying of data, or even streaming data services faster and more easily implemented.
- As a result, a rich eco‐system of tools and technologies has been built up around Hadoop, specifically on top of YARN and HDFS.

# Hadoop Cluster

- Hadoop is not hardware that you have to maintain but the name of **the software that** runs on a cluster—namely, the distributed file system, HDFS, and the cluster resource manager, YARN,
- YARN and HDFS are implemented by **several daemon processes**—that is, software that runs in the background and does not require user input.
- Hadoop processes are services, meaning they run all the time on a cluster node, accept input, and deliver output through the network, similar to how an HTTP server works.
- Each of these processes runs inside of its own Java Virtual Machine (JVM) so each daemon has its own system resource allocation and is managed independently by the operating system.

# HDFS

- Hadoop Distributed File System (HDFS), its storage system has a master-slave architecture where the master node is called NameNode and slave node is called DataNode.
- A cluster contains Master(Namenode) and multiple Slave (Datanodes) as shown in diagram below




- Name node : contain the Metadata about the data and namespaces whereas Data nodes contain the actual data. Data is replicated on multiple data nodes for Fault Tolerance and Availability.

Eg : 3 Replicas(copies) of data can be maintained on 3 data nodes.

# Contd..

- Both HDFS and YARN have multiple master services responsible for coordinating worker services that run on each worker node.
- Worker nodesimplement both the HDFS and YARN worker services.
- For HDFS, the master and worker services are as follows:
- NameNode (Master) Stores
  - the directory tree of the file system,
  - file metadata
  - the locations of each file in the cluster.
- Clients wanting to access HDFS must first locate the appropriate storage nodes by requesting information from the NameNode.
- Secondary NameNode (Master) Performs housekeeping tasks and checkpointing on behalf of the NameNode. Despite its name, it is not a backup NameNode.
- DataNode (Worker) Stores and manages HDFS blocks on the local disk. Reports health and status of individual data stores back to the NameNode.

# Data Management

- At a high level, when data is accessed from HDFS, a client application must first make a request to the NameNode to locate the data on disk.
- The NameNode will reply with a list of DataNodes that store the data, and the client must then directly request each block of data from the DataNode.
- Note that the NameNode does not store data, nor does it pass data from DataNode to client, instead acting like a traffic cop, pointing clients to the correct DataNodes.

# Basic Concepts- Hadoop

• Data is distributed immediately when added to the cluster and stored on multiple nodes. Nodes prefer to process data that is stored locally in order to minimize traffic across the network.

• Data is stored in blocks of a fixed size (usually 64 MB/128 MB) and each block is duplicated multiple times across the system to provide redundancy and data safety.

• A computation is usually referred to as a job; jobs are broken into tasks where each individual node performs the task on a single block of data.

- Jobs are written at a high level without concern for network programming, time, or low-level infrastructure.
- Jobs are fault tolerant usually through task redundancy, such that if a single node or task fails, the final computation is not incorrect or incomplete.

• Each task should be independent and nodes should not have to communicate with each other during processing to avoid interprocess dependencies that could lead to deadlock.

• Master programs allocate work to worker nodes such that many worker nodes can operate in parallel, each on their own portion of the larger dataset

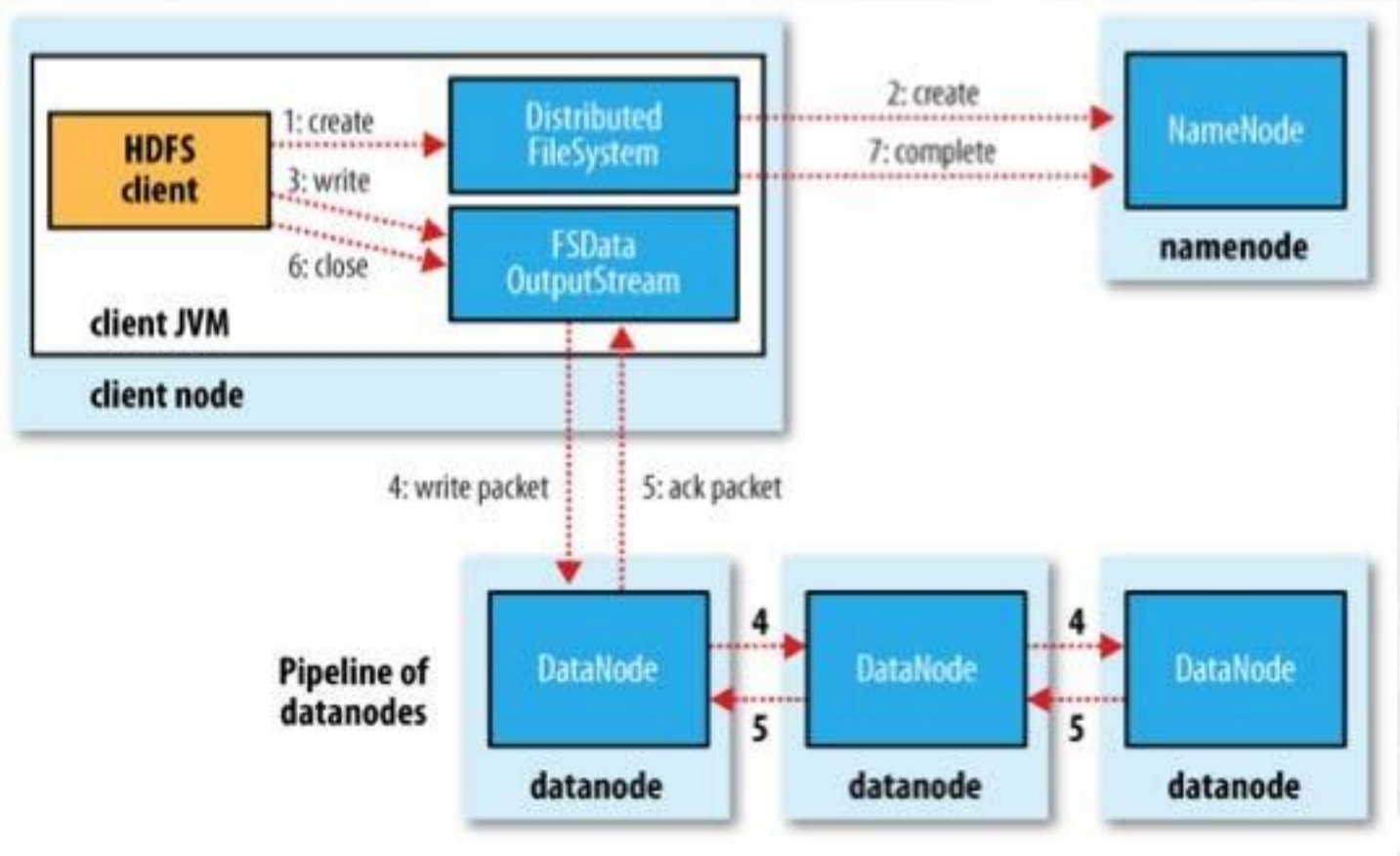
# Blocks

- Blocks are large files to be split across and distributed to many machines at run time.
- Different blocks from the same file will be stored on different machines to provide for more efficient distributed processing.
- In fact, there is a one-to-one connection between a task and a block of data and they will be replicated across the DataNodes.
- By default, the replication is three-fold, but this is configurable at runtime.
- Therefore, each block exists on three different machines and three different disks, and even if two nodes fail, the data will not be lost.

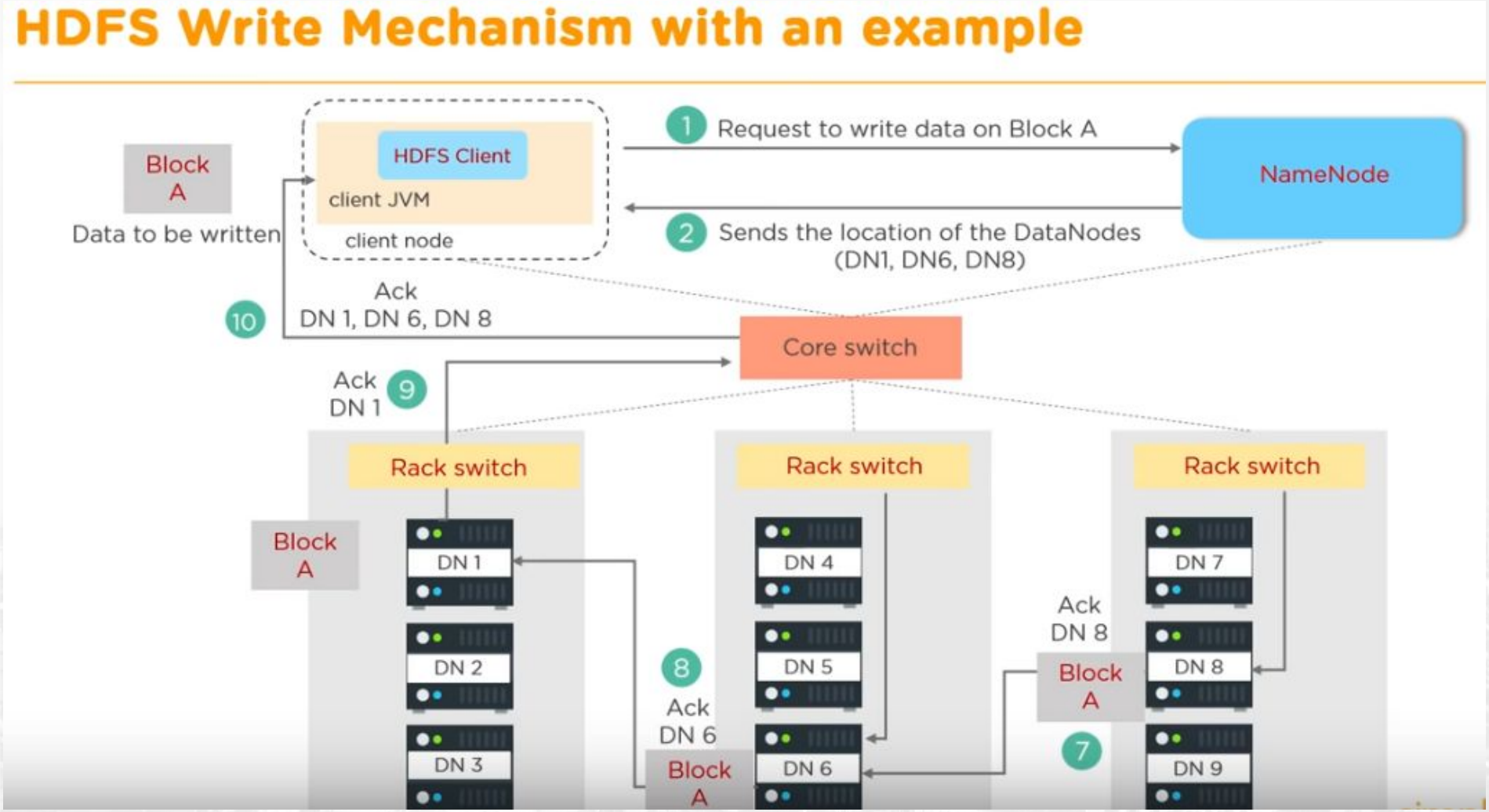
# Contd..

- HDFS is designed for storing very large files with streaming data access, and as such, it comes with a few caveats:
  - HDFS performs best with a modest number of very large files—for example, millions of large files (100 MB or more) rather than billions of smaller files that might occupy the same volume.
  - HDFS implements the WORM pattern—write once, read many. No random writes or appends to files are allowed.
  - HDFS is optimized for large, streaming reading of files, not random reading or selection.
- HDFS is best suited –
  - For storing raw input data for computation,
  - Intermediary results between computational stages,
  - Final results for the entire job.
- It is not a good fit as a data backend for applications that require updates in real-time, interactive data analysis, or record-based transactional support.
- Thus HDFS users tend to create large stores of heterogeneous data to aid in a variety of different computations and analytics , are sometimes called "data lakes" .
- Datalakes, simply hold all data about a known problem in a recoverable and fault-tolerant manner.

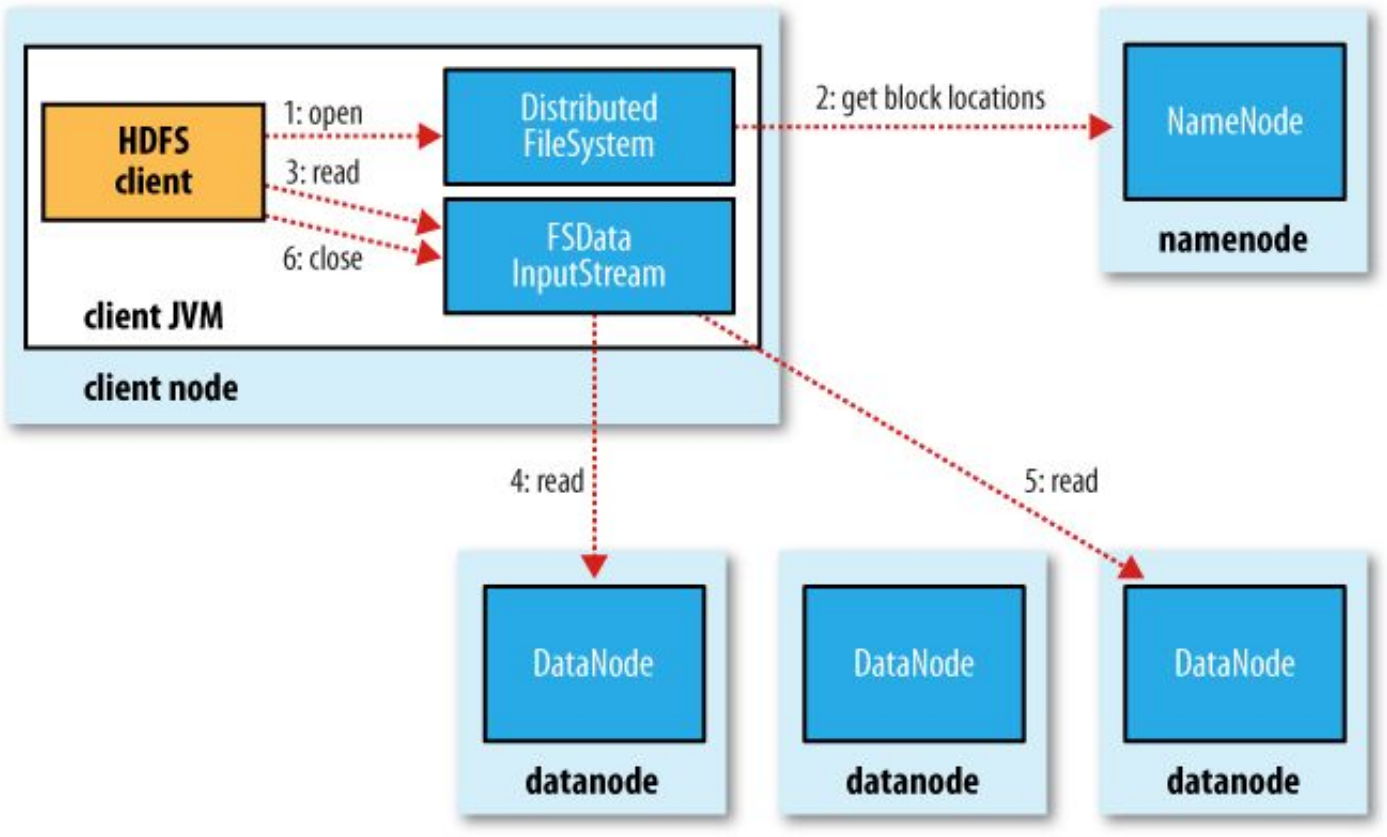
# HDFS File Write Operation



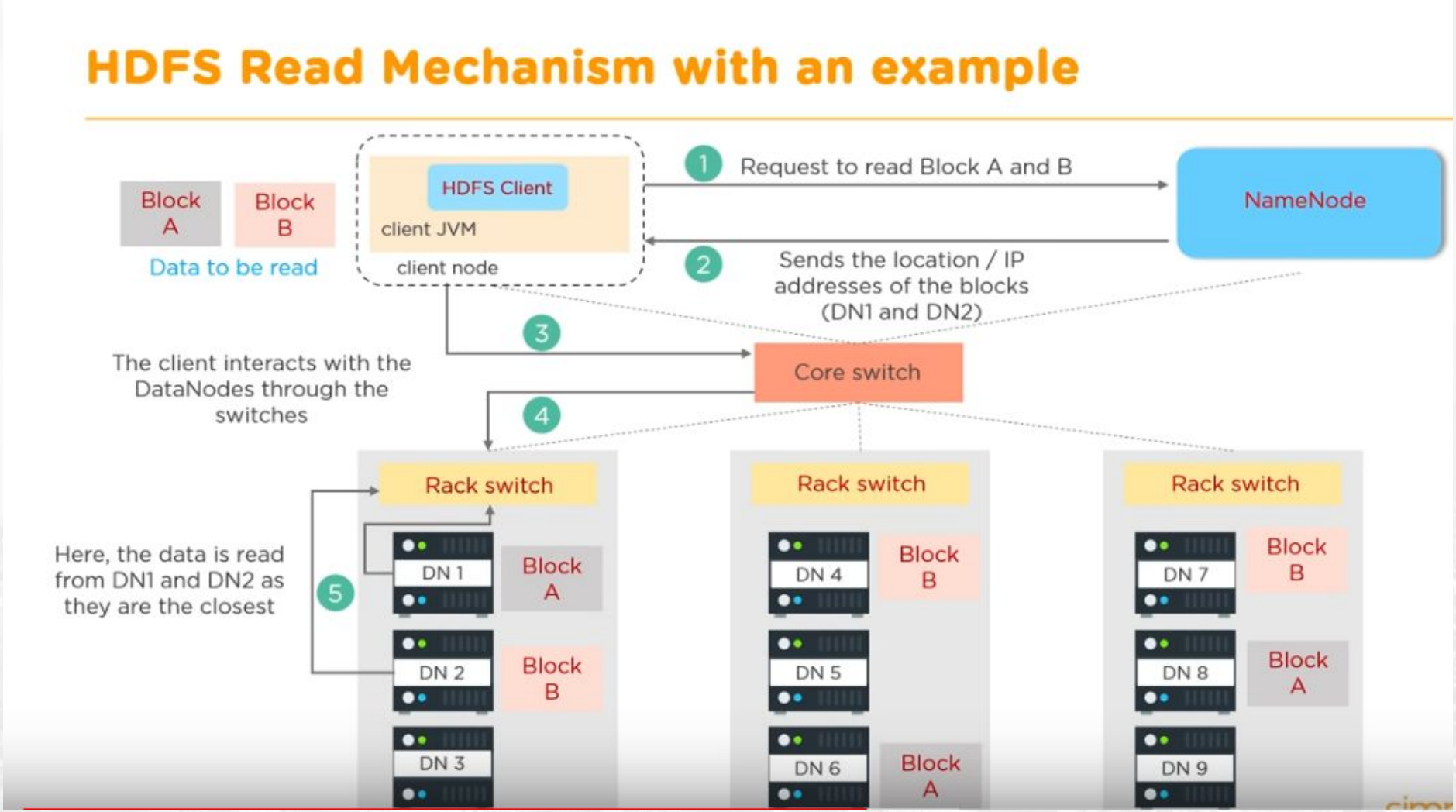
# HDFS Architecture



# HDFS File Read Operation



# HDFS Architecture



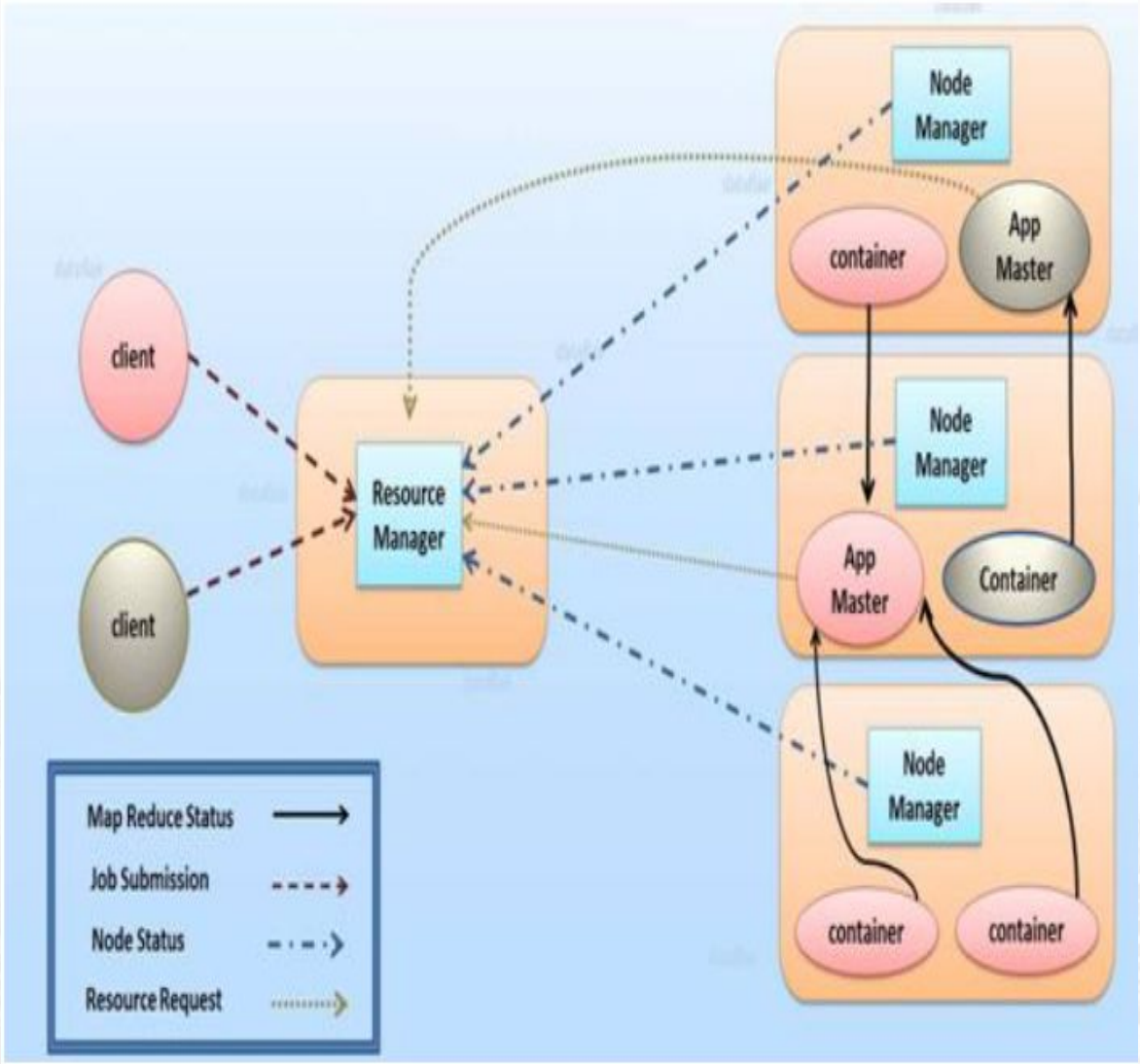
# Introducing YARN

- Hadoop 1.0 is popularized **MapReduce** and made large-scale distributed processing offering MapReduce on HDFS.
- The MapReduce job/workload management functions were highly coupled to the cluster/resource management Hadoop Architecture.
- So for other processing models or applications to utilize the cluster infrastructure for other distributed workloads gets decreased.
- MapReduce can be very efficient for large-scale batch workloads, but it's also quite I/O intensive, and due to the batch-oriented nature of HDFS and MapReduce, faces significant limitations in support for interactive analysis, graph processing, machine learning, and other memory-intensive algorithms.
- As such, it is impossible to repurpose the same cluster for these other distributed workloads.
- **Hadoop 2** addresses these limitations by introducing YARN, which decouples workload management from resource management so that multiple applications can share a centralized, common resource management service.
- By providing generalized job and resource management capabilities in YARN, Hadoop is no longer a singularly focused MapReduce framework but a full-fledged multi-application, big data operating system.
- In Hadoop 2.x, **HDFS architecture** has a single active NameNode and a single Standby NameNode, so if a NameNode goes down then we have standby NameNode to count on.

## Contd..

- YARN has multiple master services and a worker service as follows:
- ResourceManager (Master) Allocates and monitors available cluster resources (e.g., physical assets like memory and processor cores) to applications as well as handling scheduling of jobs on the cluster.
- ApplicationMaster (Master) Coordinates a particular application being run on the cluster as scheduled by the ResourceManager.
- NodeManager (Worker) Runs and manages processing tasks on an individual node as well as reports the health and status of tasks as they're running.
- The clients that wish to execute a job must first request resources from the ResourceManager, which assigns an application-specific ApplicationMaster for the duration of the job.
- The ApplicationMaster tracks the execution of the job, while the ResourceManager tracks the status of the nodes, and each individual NodeManager creates containers and executes tasks within them.
- There may be other processes/services running on the Hadoop cluster as well. eg. JobHistory servers or ZooKeeper coordinators, primary softwares running in a Hadoop cluster.

# YARN Architecture




- JobTracker's responsibility is of resource management and job scheduling/monitoring into separate daemons**.**
- **Application** is a job submitted to system.

Ex: MapReduce job.

- **Container:** Basic unit of allocation. Replaces fixed map/reduce slots. Fine-grained resource allocation across multiple resource type

Eg. Container_0: 2GB, 1CPU

Container_1: 1GB, 6CPU

# YARN Execution steps

The steps involved in YARN architecture are:

1. The client program submits an application.

2. The Resource Manager launches the Application Master by assigning some container.

3. The Application Master registers with the Resource manager.

4. On successful container allocations, the application master launches the container by providing the container launch specification to the NodeManager.

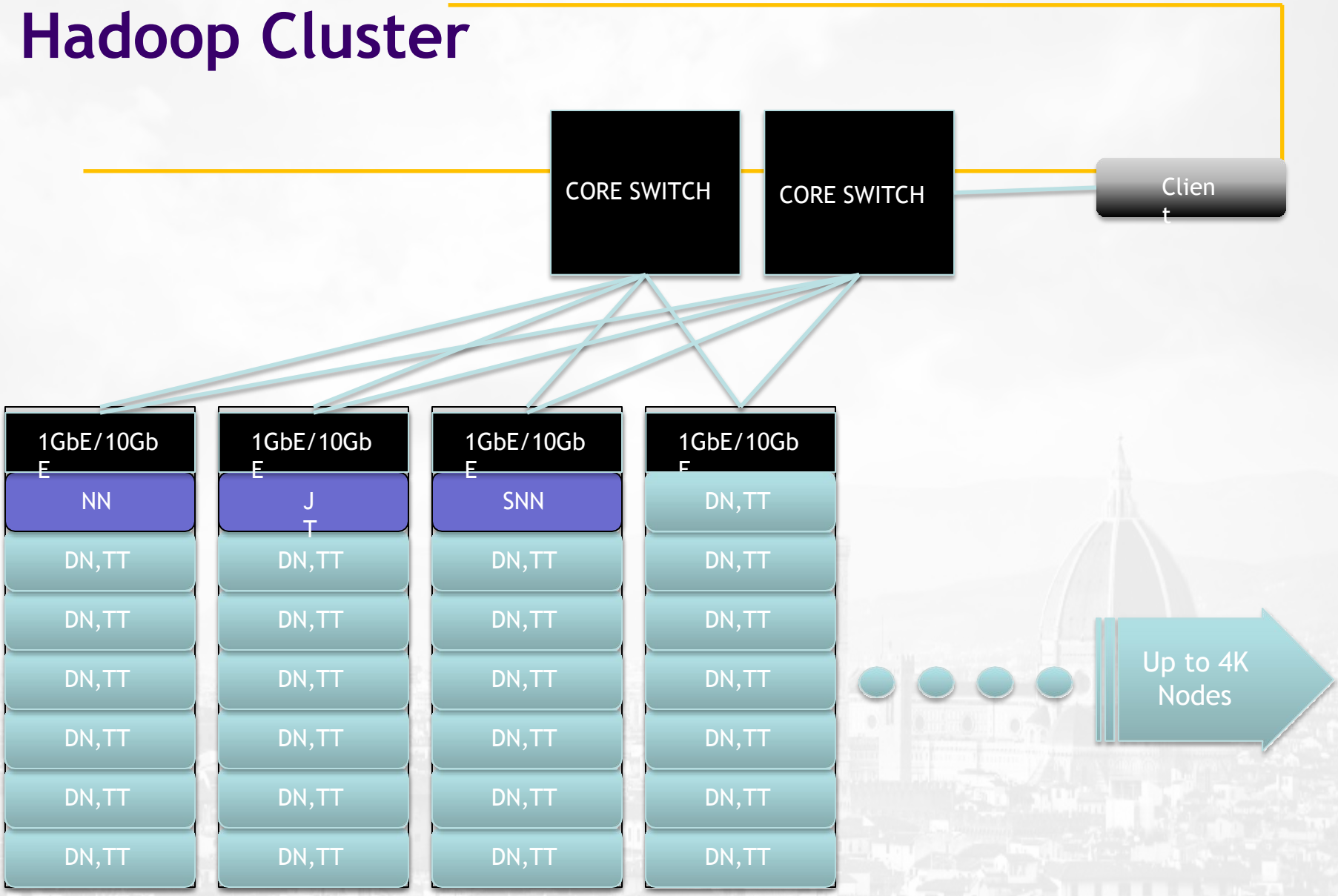
5. The NodeManager executes the application code.

6. During the application execution, the client that submitted the job directly communicates with the Application Master to get status, progress updates.

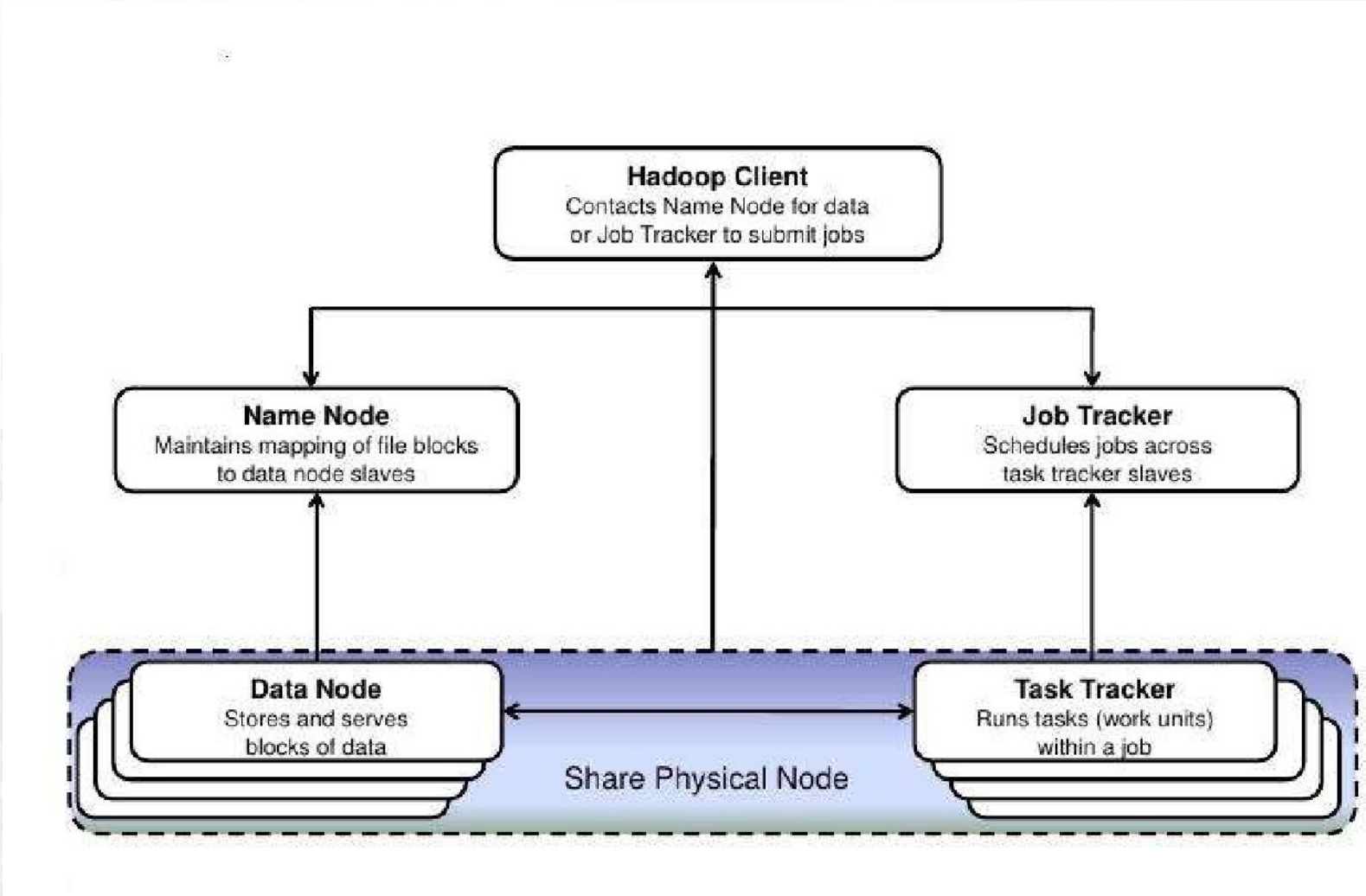
7. Once the application has been processed completely, the application master deregisters with the ResourceManager and shuts down allowing its own container to be repurposed.

# Hadoop Server Roles

# Hadoop Cluster



# Hadoop High Level Architecture



# Working with a Distributed File System

- For the most part, interaction with HDFS is performed through a command-line interface (CLI) that will be familiar with POSIX interfaces on Unix or Linux.
- Additionally, there is an HTTP interface to HDFS, as well as a programmatic interface written in Java.
- However, because the command-line interface is mostly used as it is familiar to developers.

# HDFS commands

- To see the available commands in the fs shell,

  $ hadoop fs –help

1. Creating a directory:
   - Syntax: hdfs dfs –mkdir <path>
   - Eg. hdfs dfs –mkdir /first
2. Remove a file in specified path:
   - Syntax: hdfs dfs –rm <src>
   - Eg. hdfs dfs –rm / first /abc.txt
3. Copy file from local file system to hdfs:
   - Syntax: hdfs dfs –copyFromLocal <src> <dst>
   - Eg. hdfs dfs –copyFromLocal /home/hadoop/sample.txt / first /abc1.txt
4. To display list of contents in a directory:
   - Syntax: hdfs dfs –ls <path>
   - Eg. hdfs dfs –ls / first
5. To display contents in a file:
   - Syntax: hdfs dfs –cat <path>
   - Eg. hdfs dfs –cat / first /abc1.txt

# Contd..

6. Copy file from hdfs to local file system:
   - Syntax: hdfs dfs –copyToLocal <src <dst>
   - Eg. hdfs dfs –copyToLocal / first /abc1.txt /home/hadoop/Desktop/sample.txt
7. To display last few lines of a file:
   - Syntax: hdfs dfs –tail <path>
   - Eg. hdfs dfs –tail / first /abc1.txt
8. Display aggregate length of file in bytes:
   - Syntax: hdfs dfs –du <path>
   - Eg. hdfs dfs –du / first
9. To count no.of directories, files and bytes under given path:
   - Syntax: hdfs dfs –count <path>
   - Eg. hdfs dfs –count / first
   - o/p: 1 1 60
10. Remove a directory from hdfs
    - Syntax: hdfs dfs –rmr <path>
    - Eg. hdfs dfs rmr / first

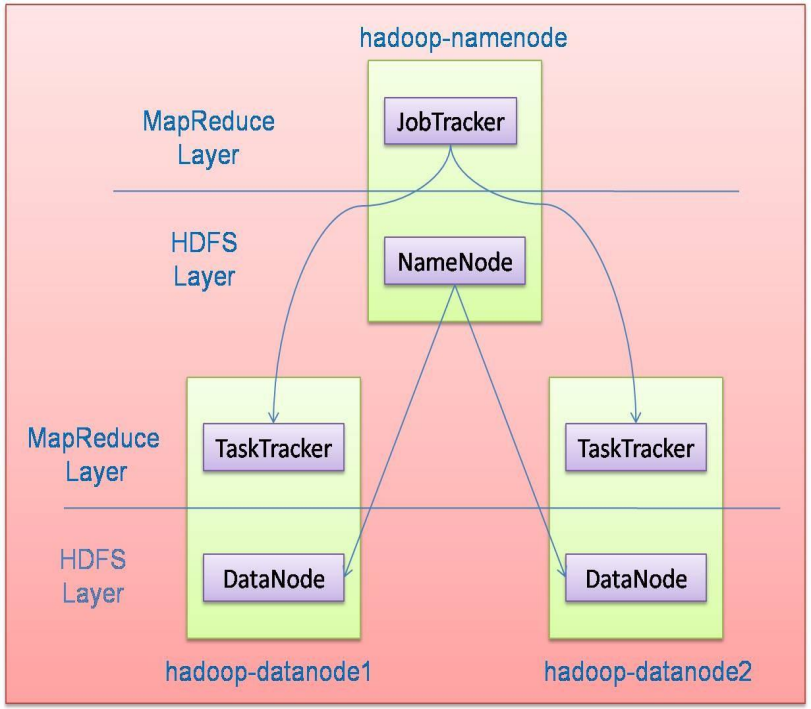
# HDFS Interfaces

- Programmatic access to HDFS is made available to software developers through a Java API, and any serious data ingestion into a Hadoop cluster should consider utilizing that API.
- There are also other tools for integrating HDFS with other file systems or network protocols—like FTP or Amazon S3.
- There are also HTTP interfaces to HDFS, which can be used for routine administration of the cluster file system and programmatic access to HDFS with Python.
- HTTP access to HDFS comes in two primary interfaces:
  - direct access through the HDFS daemons that serve HTTP requests,
  - proxy servers that expose an HTTP interface and then directly access HDFS on the client's behalf using the Java API.
- Eg. of proxies include HttpFS, Hoop, and WebHDFS, that allows RESTful network access to the Hadoop cluster, which is easily programmed against using Python.

# MapReduce

- A data processing technique and a program model for distributed computing based on java contains two important tasks, namely Map and Reduce.
- **Map** : takes a set of raw data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).
- **Reduce** : takes the output from a map as an input and combines those data tuples into a smaller set of tuples.(eg. Output keys with aggregate values )
- Under the MapReduce model, the data processing primitives are called **mappers and reducers.**
- once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change.

# MapReduce: Hadoop Execution Layer




- Jobtracker knows everything about submitted jobs
- Divides jobs into tasks and decides where to run each task
- Continuously communicating with tasktrackers
- Tasktrackers execute tasks (multiple per node)
- Monitors the execution of each task
- Continuously sending feedback to Jobtracker

- MapReduce is a master-slave architecture
  - Master: JobTracker
  - Slaves: TaskTrackers (100s or 1000s of tasktrackers)
- Every datanode is running a tasktracker.

# Map-reduce Programming

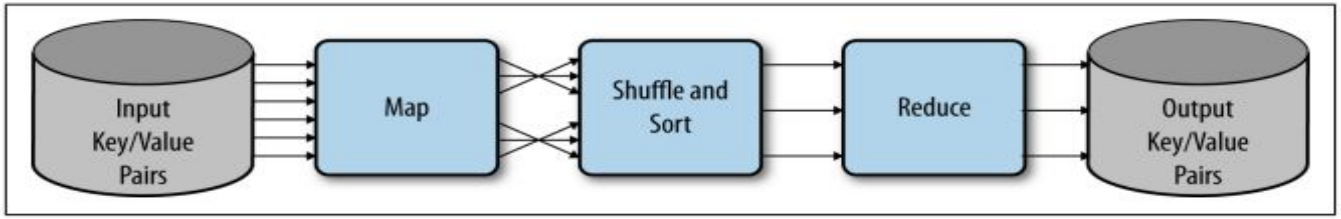


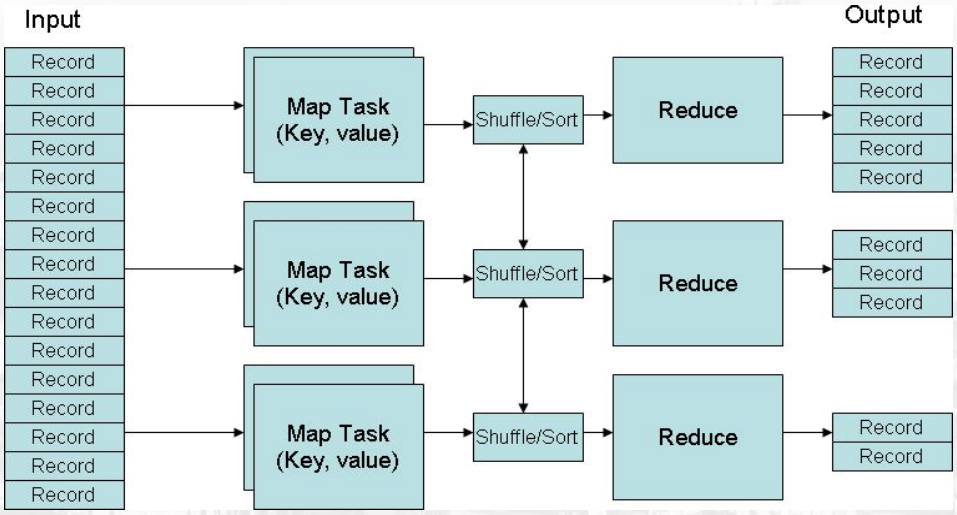

*Figure 2-5. Broadly, MapReduce is implemented as a staged framework where a map phase is coordinated to a reduce phase via an intermediate shuffle and sort*

- Phase 1 Local data is loaded into a mapping process as key/value pairs from HDFS.
- Phase 2 Mappers output zero or more key/value pairs, mapping computed values to a particular key.
- Phase 3 These pairs are then sorted and shuffled based on the key and are then passed to a reducer such that all values for a key are available to it.
- Phase 4 Reducers then must output zero or more final key/value pairs, which are the output (reducing the results of the map).

# Hadoop Computing Model

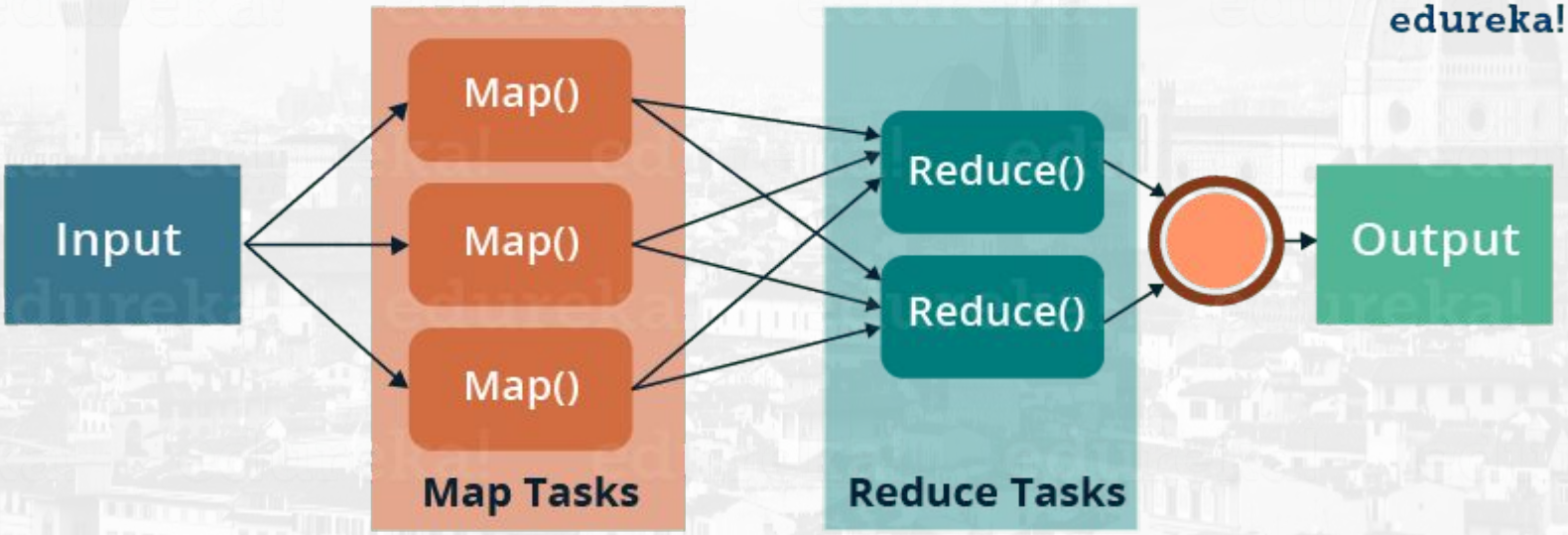
- Mapper and Reducers consume and produce *(Key, Value)* pairs.
- Users define the data type of the *Key* and *Value.*

- Shuffling & Sorting phase:
  - Map output is *shuffled* such that all same-key records go to the same reducer
  - Each reducer may receive multiple key sets
  - Each reducer *sorts* its records to group similar keys, then process each group



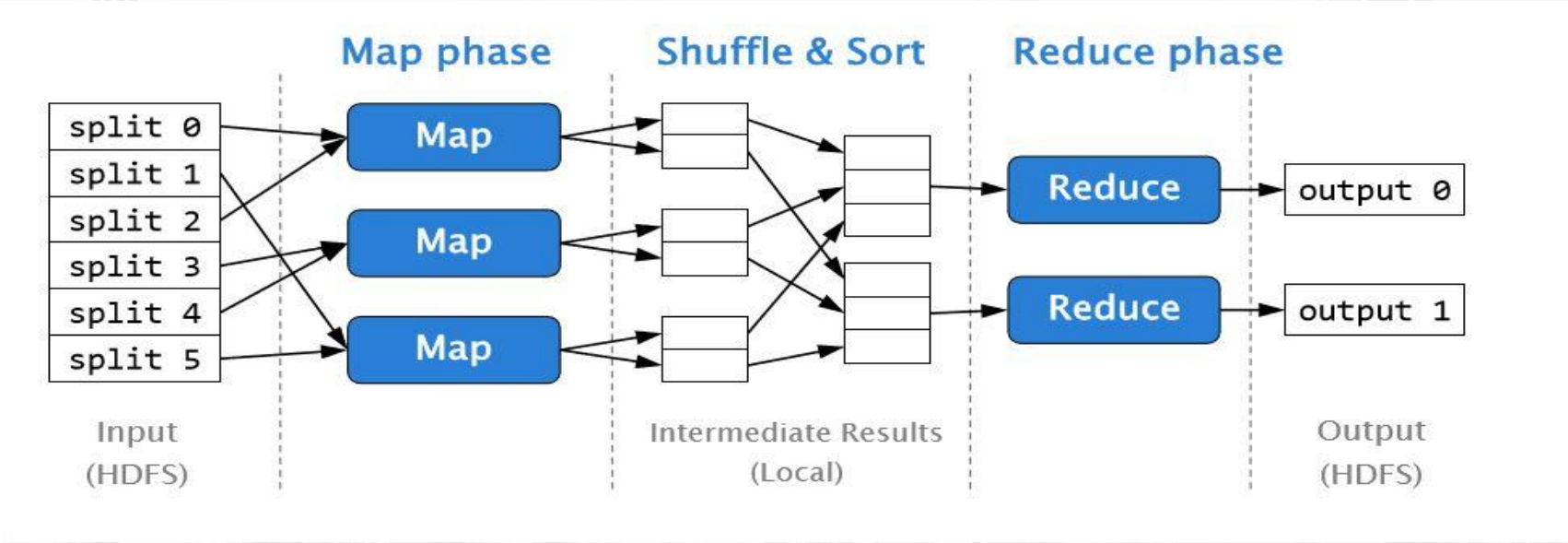
# MapReduce Workflow

- **Map Phase**
  - Raw data read and converted to key/value pairs
  - Map() function applied to any pair
- **Shuffle and Sort Phase**
  - All key/value pairs are sorted and grouped by their keys
- **Reduce Phase**
  - All values with a the same key are processed by within the same reduce() function



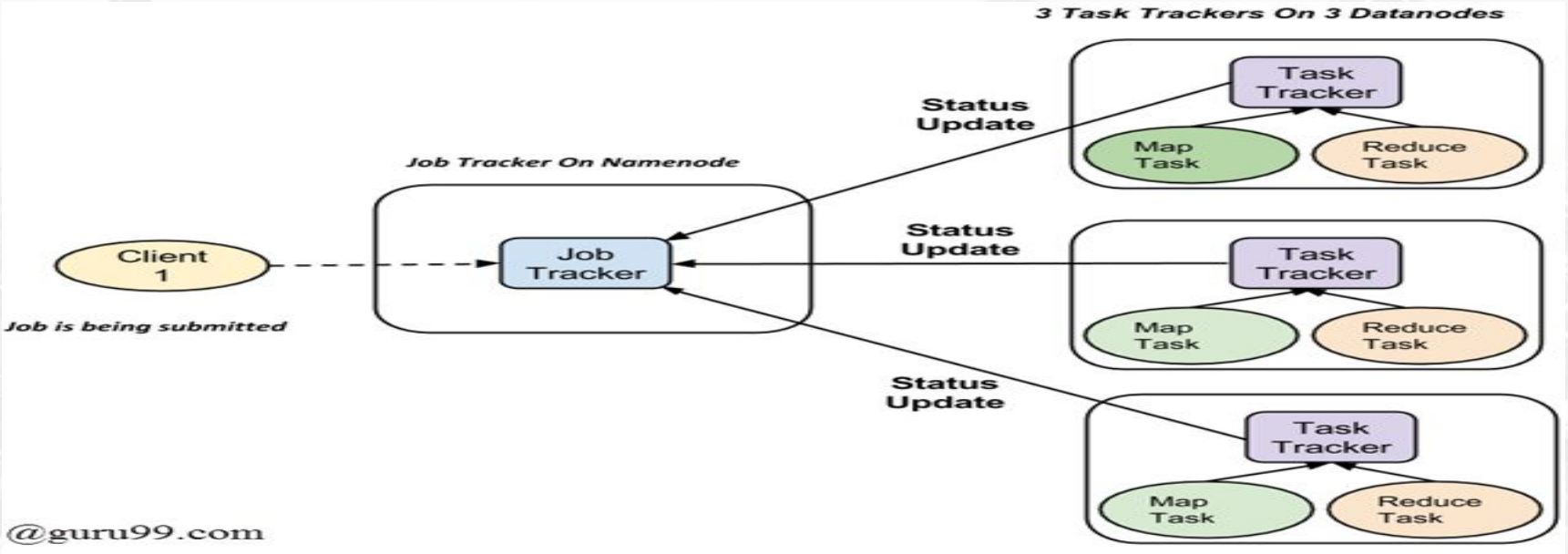
# MapReduce Programming Model

- Every MapReduce program must specify a **Mapper** and typically a **Reducer**
- The Mapper has a **map()** function that transforms input **(key, value)** pairs into any number of intermediate **(out_key, intermediate_value**) pairs
- The Reducer has a **reduce()** function that transforms intermediate **(out_key, list(intermediate_value))** aggregates into any number of output **(value')** pairs



# MapReduce Execution Details

- The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called a
  - **Jobtracker:** Acts like a master (responsible for complete execution of submitted job)
  - **Multiple Task Trackers:** Acts like slaves, each of them performing the job
- For every job submitted for execution in the system, there is one Jobtracker that resides on NameNode and there are multiple task trackers which reside on DataNode.



# MapReduce Data flow

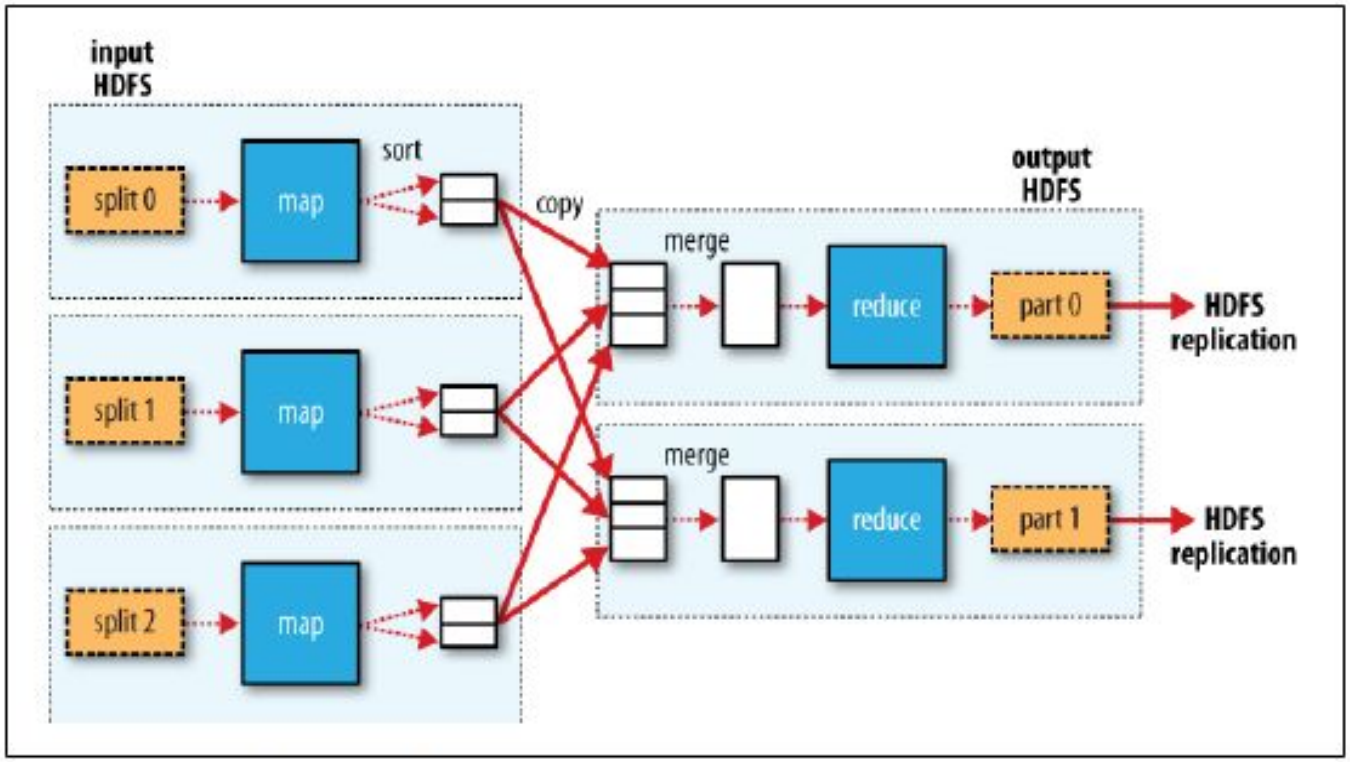
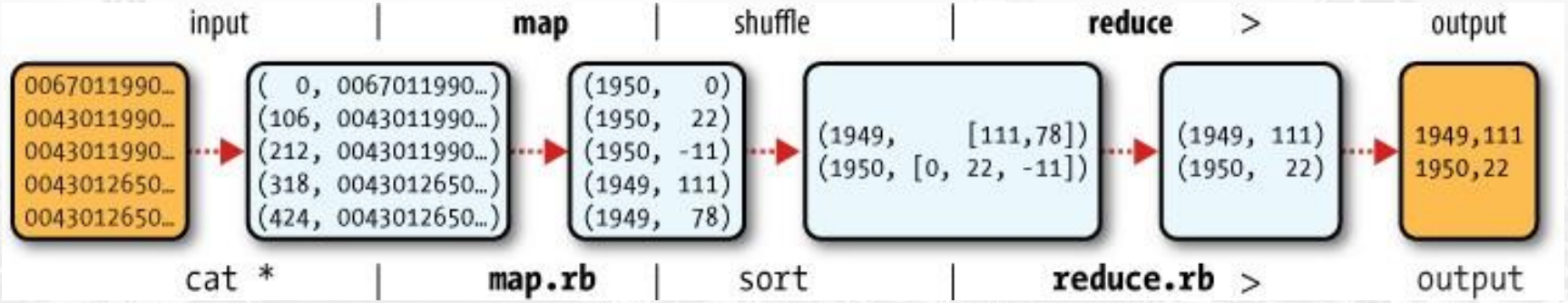


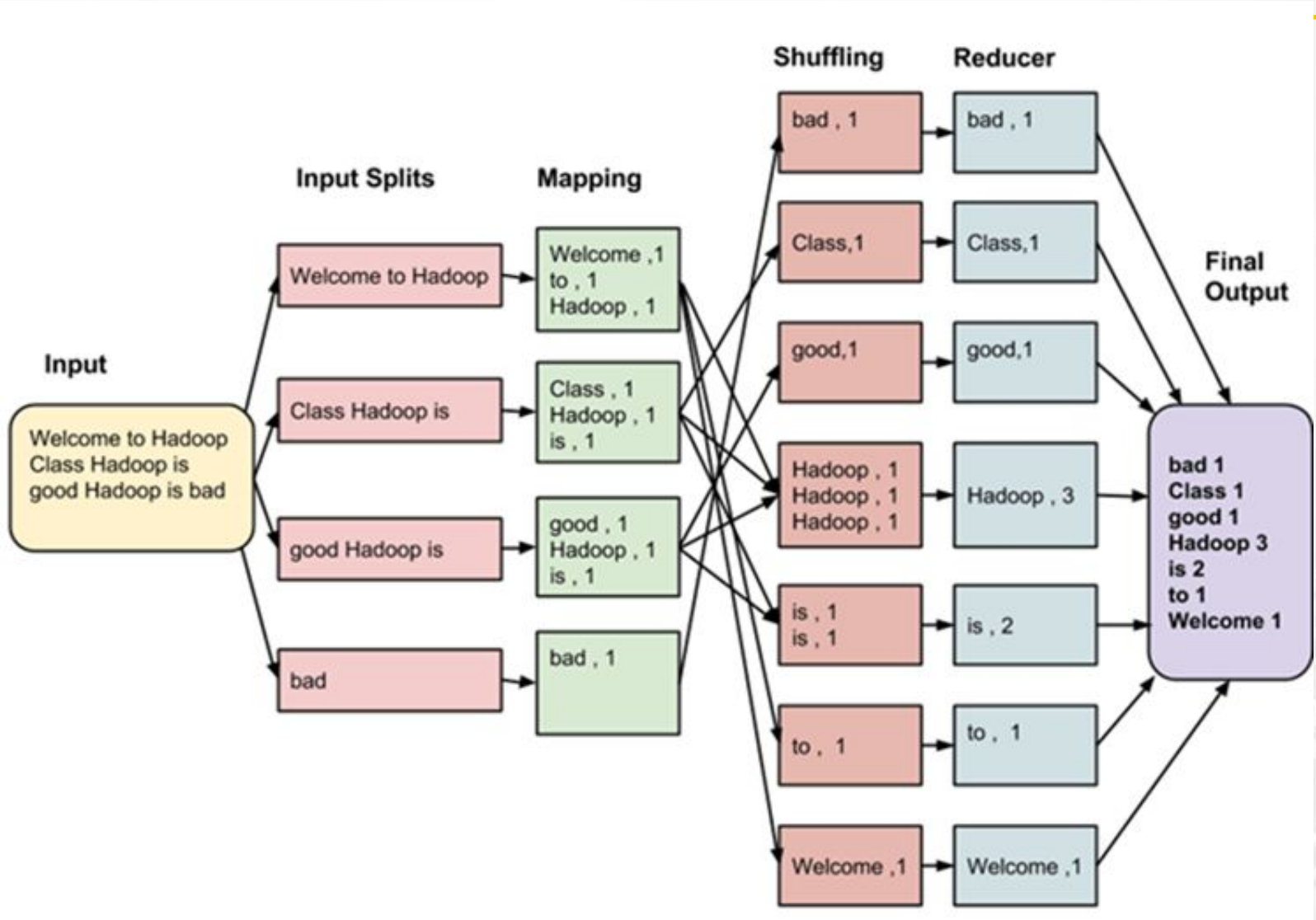

*Figure 2-4. MapReduce data flow with multiple reduce tasks*

# Ex-1. max/min temperature

What was the max/min temperature for the last century?

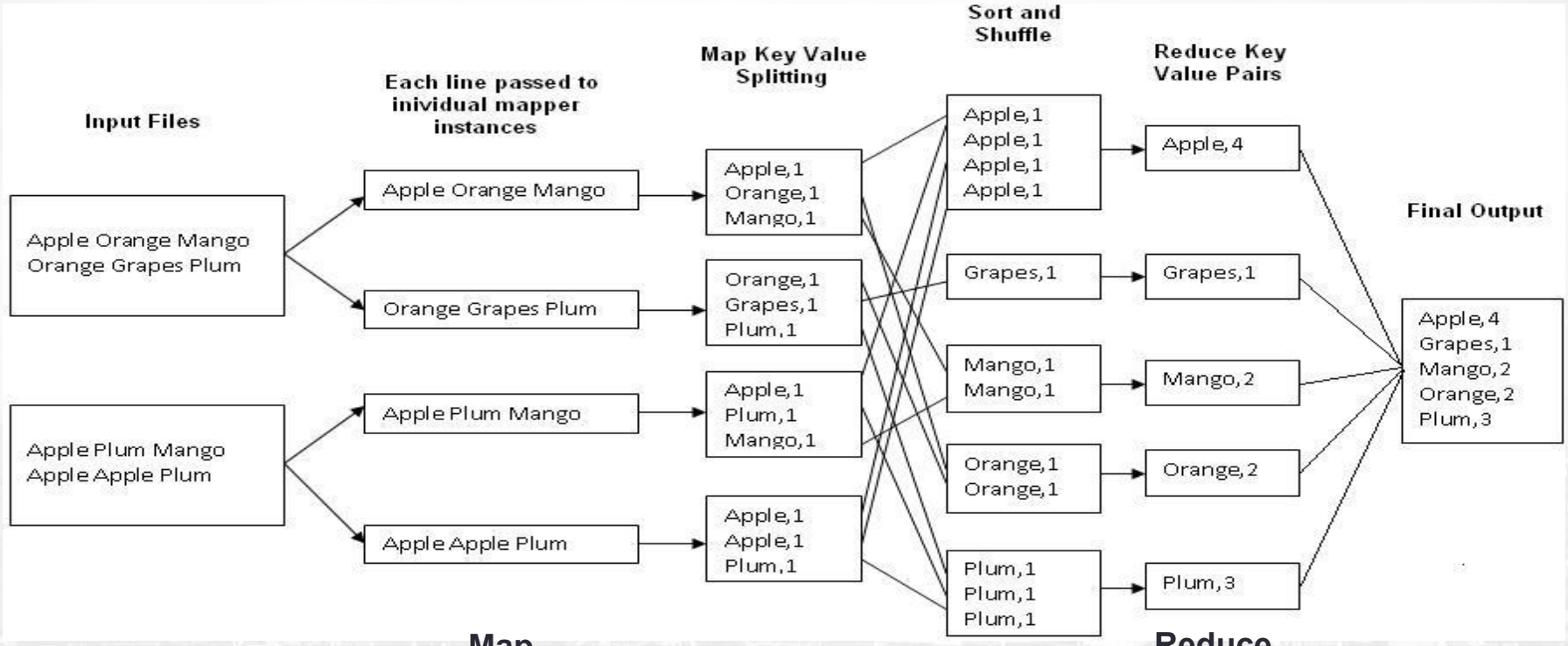


# MapReduce Programming Model



# Ex-2. Word Count

- Job: Count the occurrences of each word in a data set

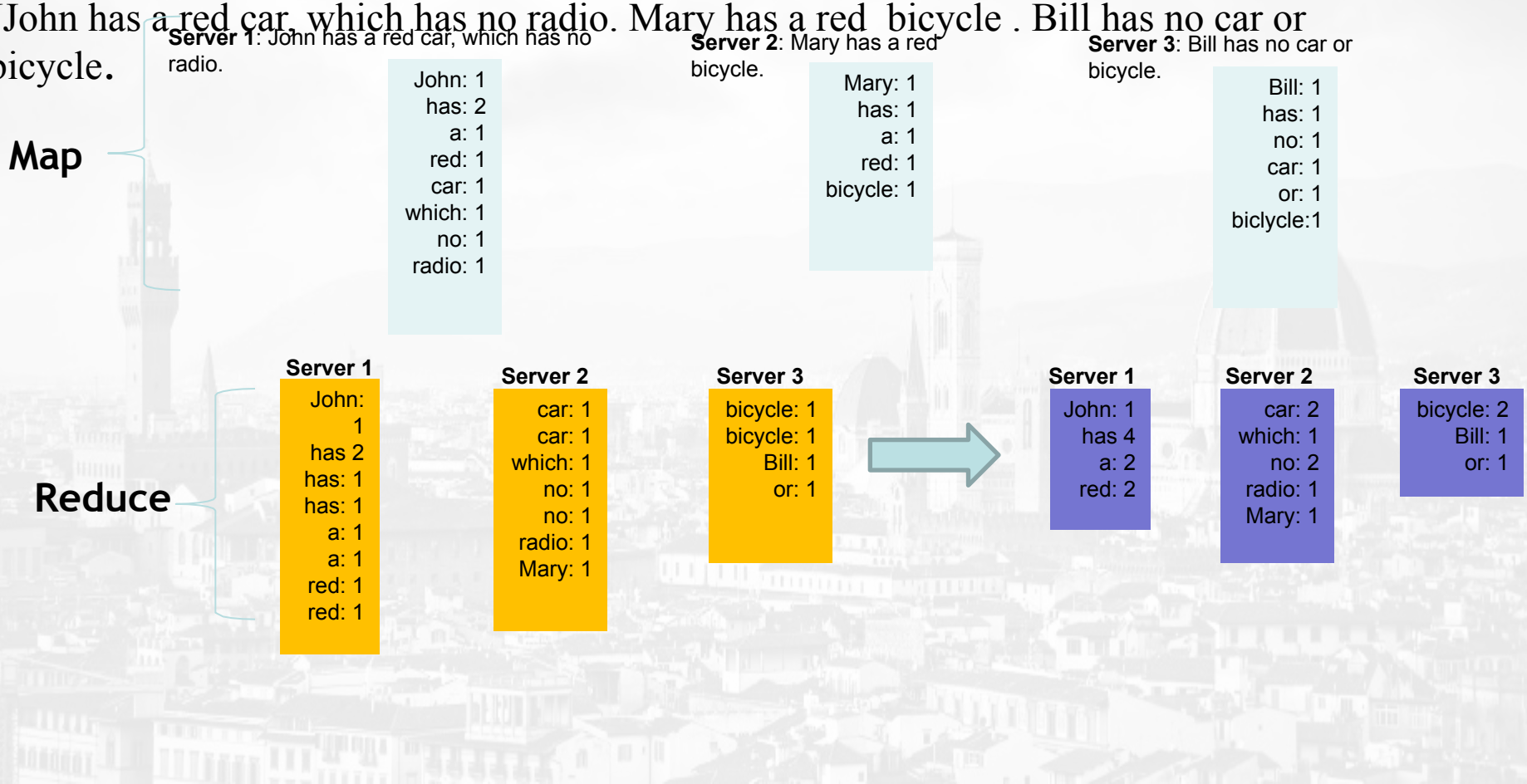



Reduce phase is optional: Jobs can be MapOnly

# Ex-3

**Problem**: Count the number of times that each word appears in the following paragraph:

JJohn has a red car, which has no radio. Mary has a red  bicycle . Bill has no car or bicycle.

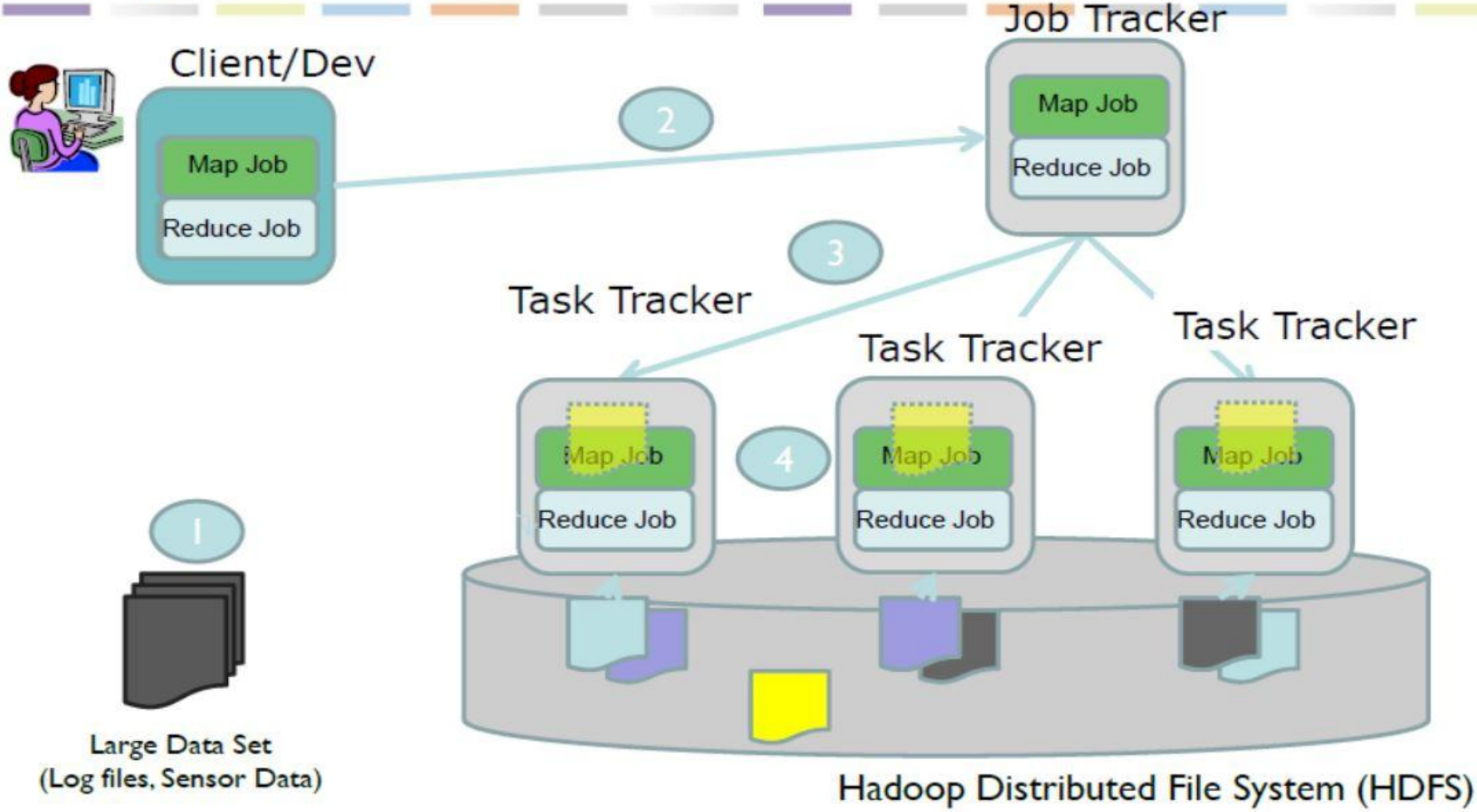


# Submitting a MapReduce Job to YARN

- The MapReduce API is written in Java, and therefore MapReduce jobs submitted to the cluster are going to be compiled Java Archive (JAR) files.
- Hadoop will transmit the JAR files across the network to each node that will run a task (either a mapper or reducer) and the individual tasks of the MapReduce job are executed.
- Connect the ResourceManager and send the ***.jar* file to be executed on all nodes of the cluster.
- The job will execute and outputs the status of mappers and reducers, and when completed will report statistics for the completion of the job. Once complete, the results of the job will be written to the directory(**wordcounts**), which can be viewed as follows:
  - hostname $ **hadoop fs –ls wordcounts**
- There are several output files named similarly to *part-00000*, and in fact, there should be one part file for each reducer that was used in the computation
- In order to read the result of the job, cat the part file from the remote file system and pipe it to less:
  - hostname $ **hadoop fs –cat wordcounts/part-00000 | less**
- The hadoop job command allows you to manage jobs currently running on the cluster. List all running jobs with the -list command:
  - hostname $ **hadoop job -list**
- Use the output to identify the job ID of the job you'd like to terminate, then kill the job issuing the -kill command:
  - hostname $ **hadoop job -kill $JOBID**

# Putting it all Together: MapReduce and HDFS

SNIA Education



# Advanced MapReduce Techniques

- Combiners:-- the primary MapReduce optimization technique
- Partitioners:-- technique for ensuring there is no bottleneck in the reduce step
- Job chaining:-- a technique for putting together larger algorithms and data flows.

# Combiners

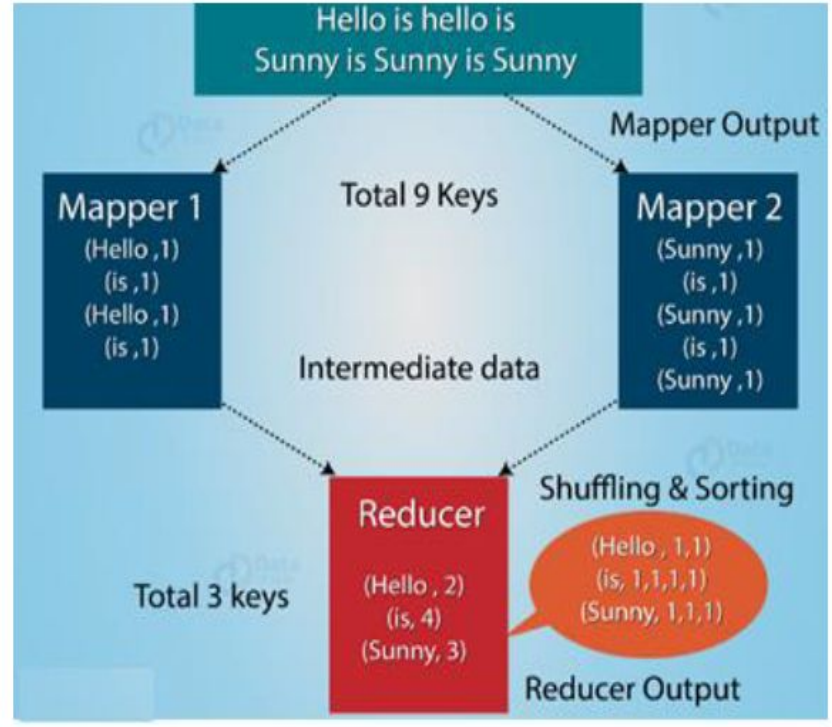
- Mappers produce a lot of intermediate data that must be sent over the network to be shuffled, sorted, and reduced.
- Because networking is a physical resource, large amounts of transmitted data can lead to job delays and memory bottlenecks e.g., there is too much data for the reducer to hold into memory.
- Combiners are the primary mechanism to solve this problem, and are essentially intermediate reducers that are associated with the mapper output.
- Combiners reduce network traffic by performing a mapper-local reduction of the data before forwarding it on to the appropriate reducer.

## Contd..

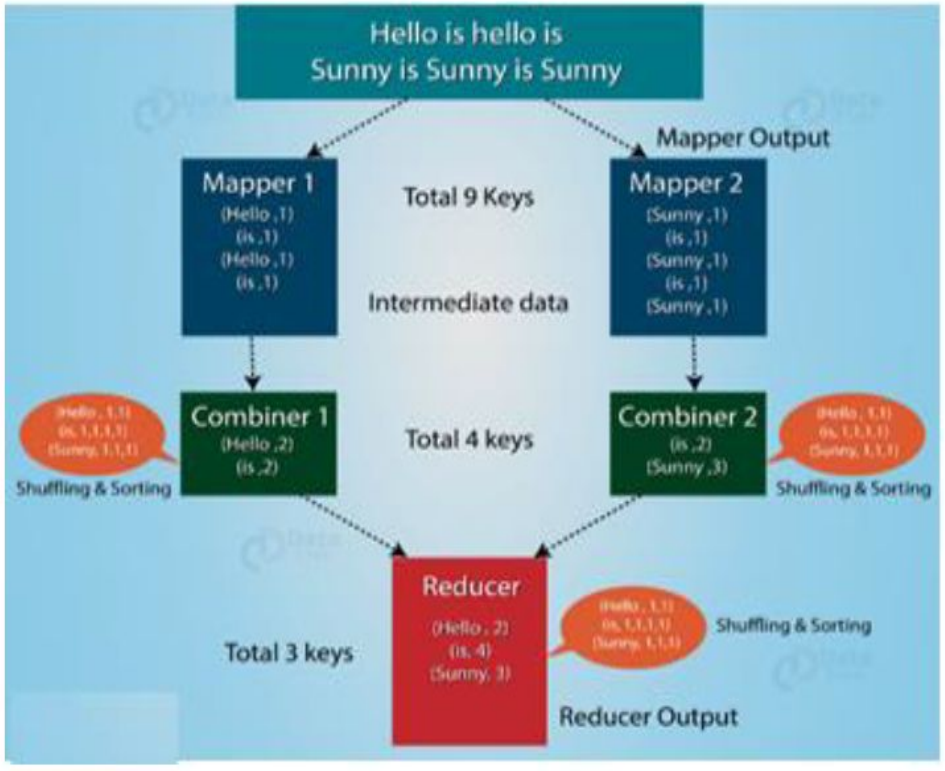
- Consider the following output from two mappers and a simple sum reduction.
  - Mapper 1 output: (IAD, 14.4), (SFO, 3.9), (JFK, 3.9), (IAD, 12.2), (JFK, 5.8)
  - Mapper 2 output: (SFO, 4.7), (IAD, 2.3), (SFO, 4.4), (IAD, 1.2)
  - Intended sum reduce output: (IAD, 29.1), (JFK, 9.7), (SFO, 13.0)
- Each mapper is emitting extra work for the reducer, namely in the duplication of the different keys coming from each mapper. A combiner that precomputes the sums for each key will reduce the number of key/value pairs being generated, and therefore the amount of network traffic.
- As there are fewer duplicate keys, the shuffle and sort operation also becomes faster.
- It is extremely common for the combiner and the reducer to be identical, which is possible if the operation is commutative and associative, but this is not always the case.
- The algorithms expressed both with a mapper, a reducer, and a combiner if the combiner has a different operation than the reducer.
- To specify a combiner in Hadoop Streaming, use the -combiner option, similar to specifying a mapper and reducer.
- In the microframework, a combiner class would simply the subclass- Reducer.

# Combiner

- It takes intermediate <keys, value> pairs provided by mapper and applies user specific aggregate function to only one mapper. It is also known as local Reducer to perform local aggregation on intermediate outputs.




MapReduce without Combiner class




MapReduce with Combiner class

# Contd..

- The main use of combiner is to reduce the number of key value pairs which is passed from mapper to reducer so that network traffic can be reduced.
- Combiner takes output of mapper as its input i,e output key and value type of mapper must be same as combiner's input key and value type. Thus There is one combiner for each mapper.
- But reducer's input key and value type is not same as mapper output type and by default there is one reducer for all the mappers which takes values from all the mappers and do some aggregation and summation on the inputs.
- Combiners execute whenever there is requirement ,its not compulsory that combiners will execute every time but in case reducers they execute everytime unless it is map only job.

# Partitioners

- Partitioners control how keys and their values get sent to individual reducers by dividing up the keyspace.
- The default behavior is the HashPartitioner, allocates keys evenly to each reducer by computing the hash of the key and assigning the key to a keyspace determined by the number of reducers.
- The issue arises when there is a key imbalance, such that a large number of values are associated with one key, and other keys are less likely where a significant portion of the reducers are underworked, and much of the benefit of reduction parallelism is lost.
- A custom partitioner can ease this problem by dividing the keyspace according to some other semantic structure besides hashing (which is usually domain specific).
- Custom partitioners can also be required for some types of MapReduce algorithms, most notably to implement a left outer join.
- The use of a custom partitioner also allows for clearer data organization, allowing you to write sectioned output to each file based on the partitioning criteria—for example, writing per-year output data.
- Unfortunately, custom partitioners can only be created with the Java API.
- However, Hadoop Streaming users can still specify a partitioner Java class either from the Hadoop library, or by writing their own Java partitioner and submitting it with their streaming job.

# Output

Output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

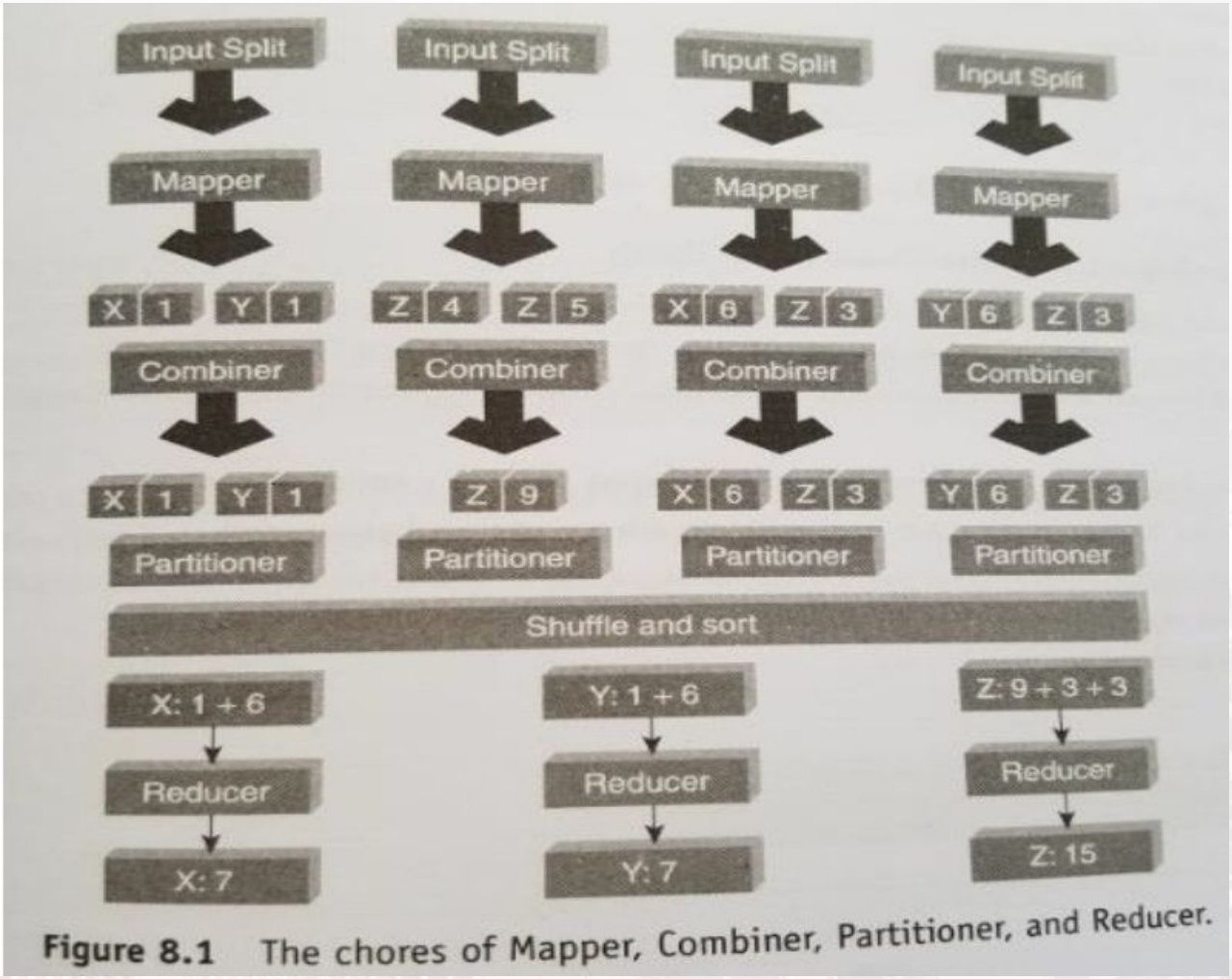



**Figure 8.1** The chores of Mapper, Combiner, Partitioner, and Reducer.

# Job Chaining

- Most complex algorithms , in order to implement more complex analytics, use a technique called *job chaining.*
- If a complex algorithm can be decomposed into several smaller MapReduce tasks, then these tasks can be chained together to produce a complete output.
- E.g. Consider a computation to compute the pairwise Pearson correlation coefficient for a number of variables in a dataset that requires a computation of the mean and standard deviation of each variable.
- Because this cannot be easily accomplished in a single MapReduce, we might employ the following strategy:
  1. Compute the mean and standard deviation of each (X, Y) pair.
  2. Use the output of the first job to compute the covariance and Pearson correlation coefficient.
- The mean and standard deviation can be computed in the initial job by mapping the total, the sum, and the sum of squares to a reducer that computes the mean and standard deviation.
- The second job takes the mean and standard deviation and computes the covariance by mapping the difference of the value and the mean, and their product for each pair, then reducing by appropriately summing and taking a square root.

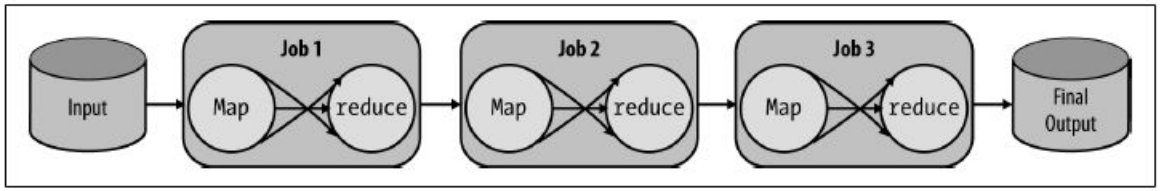



*Figure 3-2. Linear job chaining produces complete computations by sending the output of one or more MapReduce jobs as the input to another*

- Job chaining is therefore the combination of many smaller jobs into a complete computation by sending the output of one or more previous jobs into the input of another.
- In order to implement algorithms like this, the developer must think about how each individual step of a computation can be reduced to intermediary values, not just between mappers and reducers but also between jobs.
- Many jobs are typically thought of as *linear job chaining*. A linear dependency means that each MapReduce job is dependent only upon a single previous job.

- When considering job chains and job dependencies, it's helpful to note that the possibility of map-only jobs exists.
- Map-only jobs fall into two categories:
  - those where you would require no aggregation
  - and those where you are actively trying to avoid the shuffle and sort phase—either to maintain data order or to optimize the execution of the job.
- To execute a map-only job, simply set the number of reducers to 0.
- With Hadoop Streaming, you can specify the number of reducers via the –numReduceTasks flag.

# Map Reduce Real Time Use Cases

1. Merging Small Files into SEQUENCE Files
2. Visits Per Hour
3. Measuring the Page Rank
4. Word Search in Huge Log Files (Word Count also)

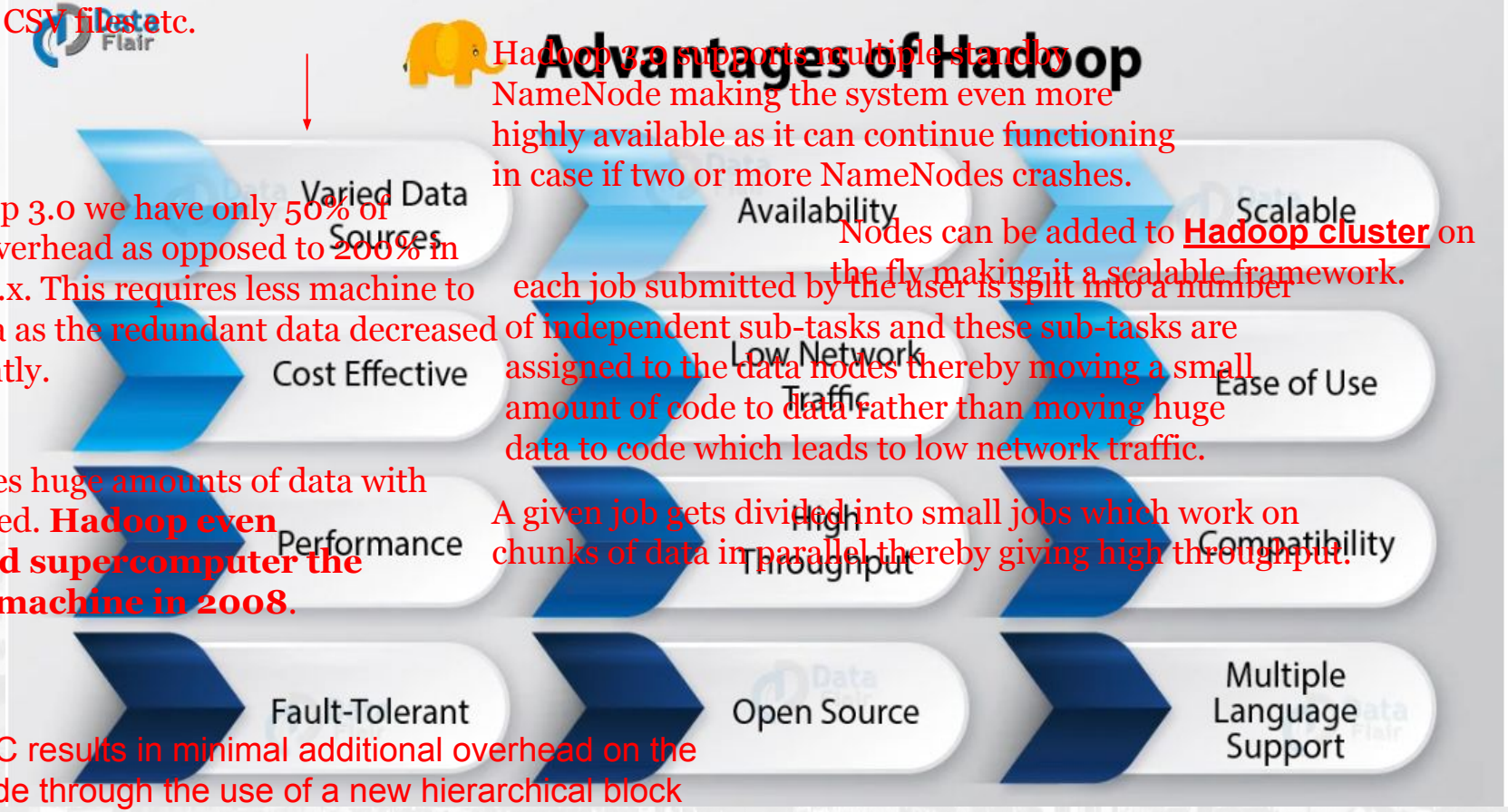
# Comparing RDBMS and MapReduce

| | Traditional RDBMS | MapReduce |
|---|---|---|
| Data Size | Gigabytes *(Terabytes)* | Petabytes *(Exabytes)* |
| Access | Interactive and Batch | Batch |
| Updates | Read / Write many times | Write once, Read many times |
| Structure | Static Schema | Dynamic Schema |
| Integrity | High (ACID) | Low |
| Scaling | Nonlinear | Linear |
| DBA Ratio | *1:40* | *1:3000* |

*Reference: Tom White's Hadoop: The Definitive Guide*

# Advantages of Hadoop 3.0:

Hadoop can accept data in a text file, XML file, images, CSV files etc.




Hadoop 3.0 supports multiple stand-by NameNode making the system even more highly available as it can continue functioning in case if two or more NameNodes crashes.

In Hadoop 3.0 we have only 50% of storage overhead as opposed to 200% in Hadoop2.x. This requires less machine to store data as the redundant data decreased significantly.

Nodes can be added to **Hadoop cluster** on the fly making it a scalable framework.

each job submitted by the user is split into a number of independent sub-tasks and these sub-tasks are assigned to the data nodes thereby moving a small amount of code to data rather than moving huge data to code which leads to low network traffic.

processes huge amounts of data with high speed. **Hadoop even defeated supercomputer the fastest machine in 2008**.

A given job gets divided into small jobs which work on chunks of data in parallel thereby giving high throughput.

HDFS-EC results in minimal additional overhead on the NameNode through the use of a new hierarchical block naming protocol,

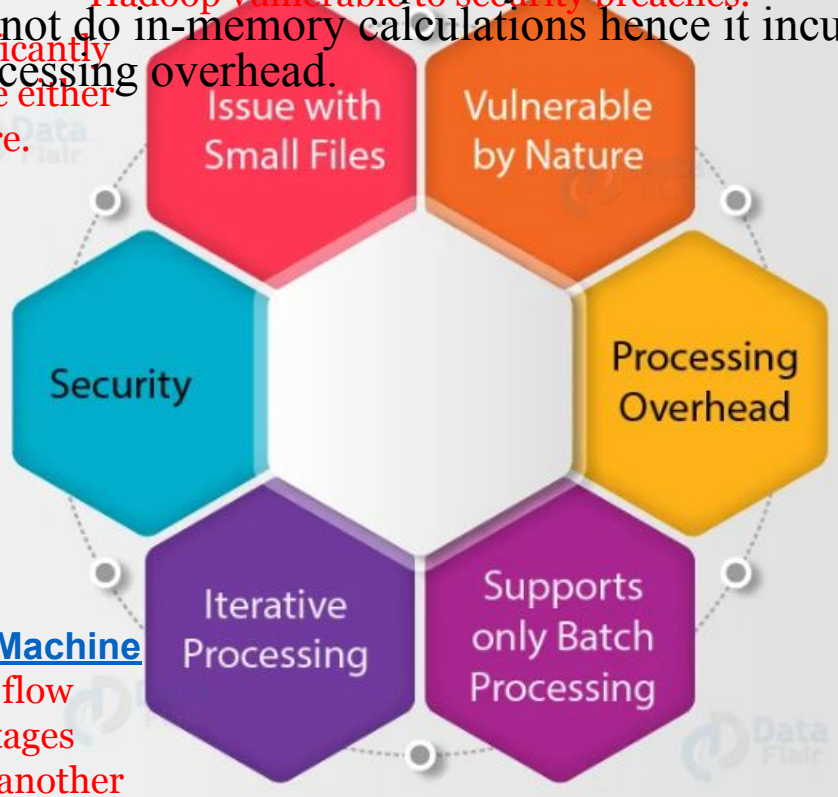
# Disadvantages of Hadoop 3.0:

- The data is read from the disk and written to the disk which makes read/write operations very expensive with tera and petabytes of data. Hadoop cannot do in-memory calculations hence it incurs processing overhead.

A small file is nothing but a file which is significantly smaller than Hadoop's block size which can be either 128MB or 256MB by default. Hadoop fails here.

It is easily exploited by cyber criminals which makes Hadoop vulnerable to security breaches.

Hadoop uses Kerberos authentication which is hard to manage with missing encryption at storage and network levels




Disadvantages of Hadoop

Hadoop cannot do iterative processing by itself. Machine learning or iterative processing has a cyclic data flow whereas Hadoop has data flowing in a chain of stages where output on one stage becomes the input of another stage.

not efficient in stream processing.
It cannot produce output in real-time with low latency.as it works on data which we collect and store in a file in advance before processing.

# References


1. Hadoop: The Definitive Guide, Tom White,3rd edition ,O'Reilly ,yahoo Press
2. Learning Spark-Lightning-Fast Big Data Analysis, Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia, O'Reilly, Copyright © 2015
3. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.pdf
4. https://www.knowledgehut.com/blog/big-data/5-best-data-processing-frameworks
5. https://opensourceforu.com/2018/03/a-quick-comparison-of-the-five-best-big-data-frameworks/
6. https://data-flair.training/blogs/hadoop-ecosystem-components/
7. http://dbis.informatik.uni-freiburg.de/content/courses/SS12/Praktikum/Datenbanken%20und%20Cloud%20Computing/slides/MapReduce.pdf
8. https://www.edureka.co/blog/mapreduce-tutorial/
9. Introduction to Analytics and Big Data-Hadoop, Thomas Rivera, Hitachi Data System, SNIA Education,2014
10. Hadoop: A Framework for Data-Intensive Distributed Computing, CS561-Spring 2012 WPI, Mohamed Y. Eltabakh

# Thank you