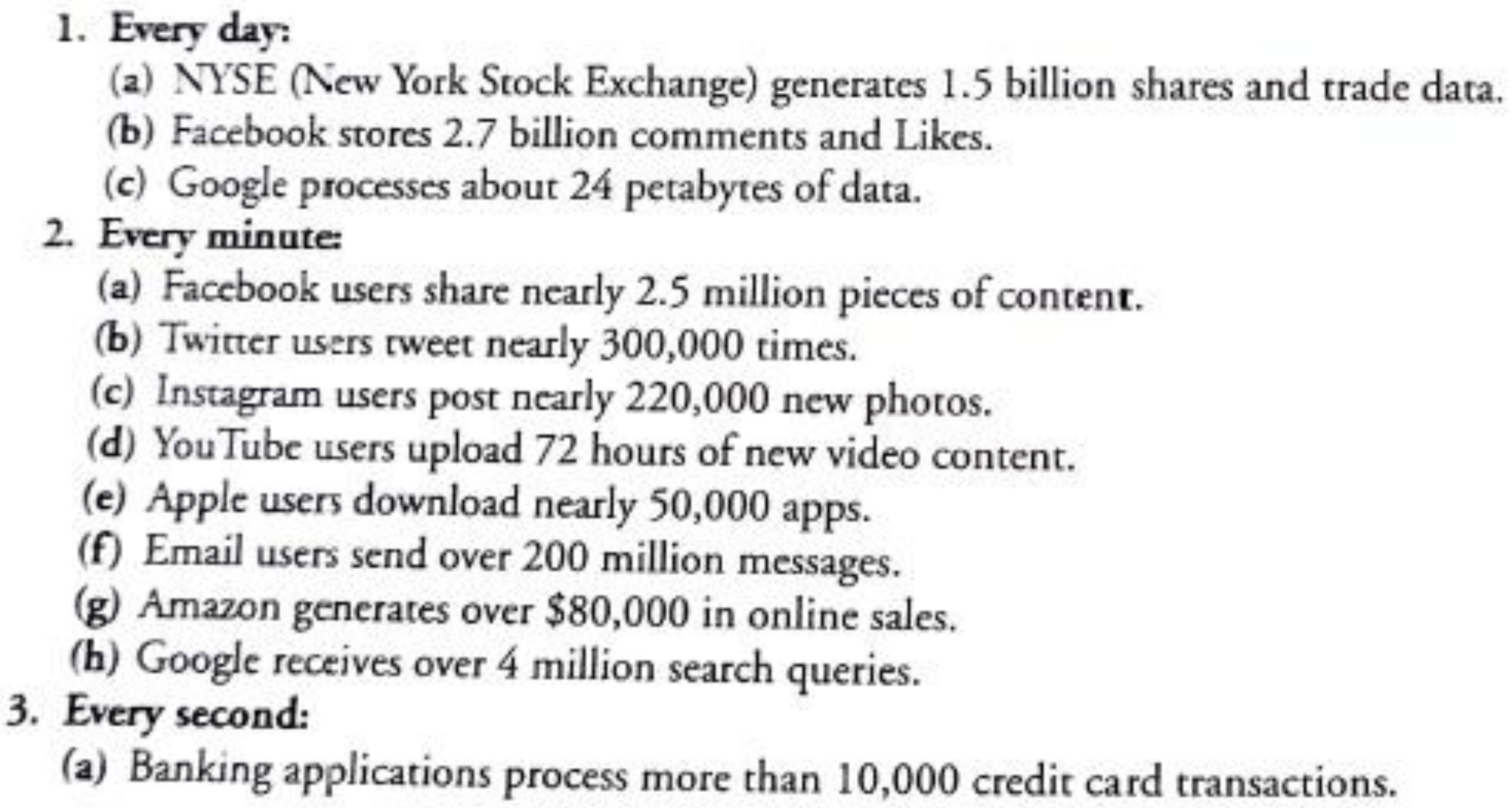
Module 2
Chapter 5 – Introduction to Hadoop

Prepared By:
Mr. Rajesh Nayak
Department of Artificial Intelligence and Data Science

5.1 Introducing Hadoop



1. **Every day:**
   (a) NYSE (New York Stock Exchange) generates 1.5 billion shares and trade data.
   (b) Facebook stores 2.7 billion comments and Likes.
   (c) Google processes about 24 petabytes of data.
2. **Every minute:**
   (a) Facebook users share nearly 2.5 million pieces of content.
   (b) Twitter users tweet nearly 300,000 times.
   (c) Instagram users post nearly 220,000 new photos.
   (d) YouTube users upload 72 hours of new video content.
   (e) Apple users download nearly 50,000 apps.
   (f) Email users send over 200 million messages.
   (g) Amazon generates over $80,000 in online sales.
   (h) Google receives over 4 million search queries.
3. **Every second:**
   (a) Banking applications process more than 10,000 credit card transactions.

- **<u>5.1.1. Data: The Treasure Trove:</u>**

1. Provides business advantages such as generating product recommendations, inventing new products, analysing the market, and many more,.
2. Provides few early key indicators that can turn the fortune of business.
3. Provide room for precise analysis. If we have more data for analysis, then we have grater precision of analysis.

- **<u>5.2. Why Hadoop?</u>**
- The key consideration for popularity of Hadoop is ***"Its capability to handle massive amounts of data, different categories of data-fairly quickly."***
- Other considerations: Low cost, computing power, scalability, storage flexibility, and inherent data protection.

5.1 Introducing Hadoop

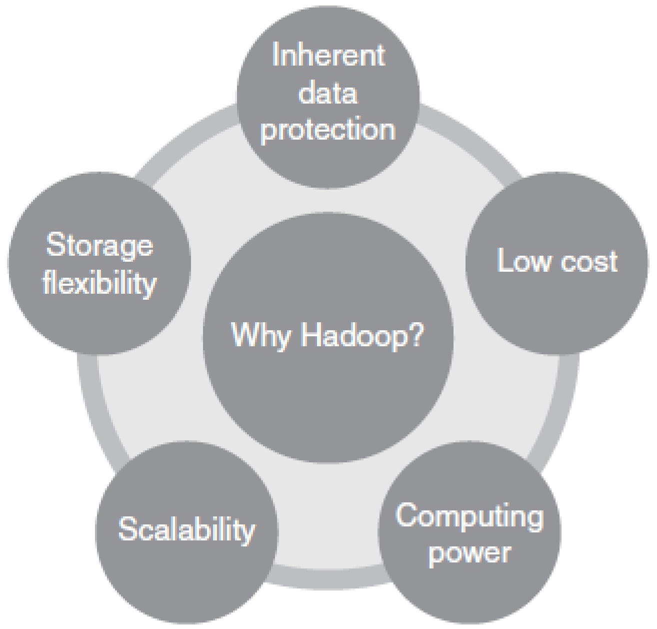


Figure 5.2 Key considerations of Hadoop

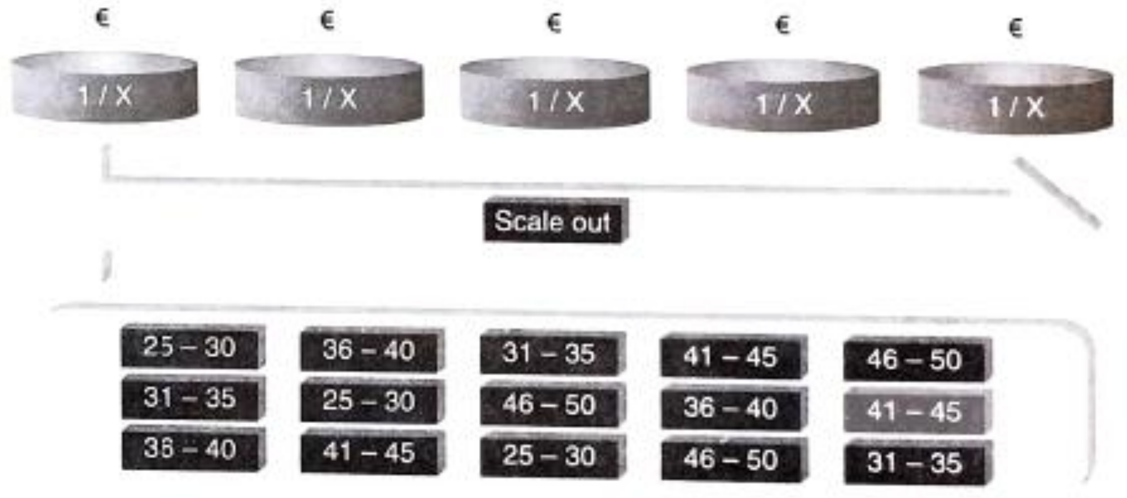


Figure 5.3 Hadoop framework (distributed file system, commodity hardware).

- **5.3. Why Not RDBMS?**
- **5.4 RDBMS versus Hadoop**

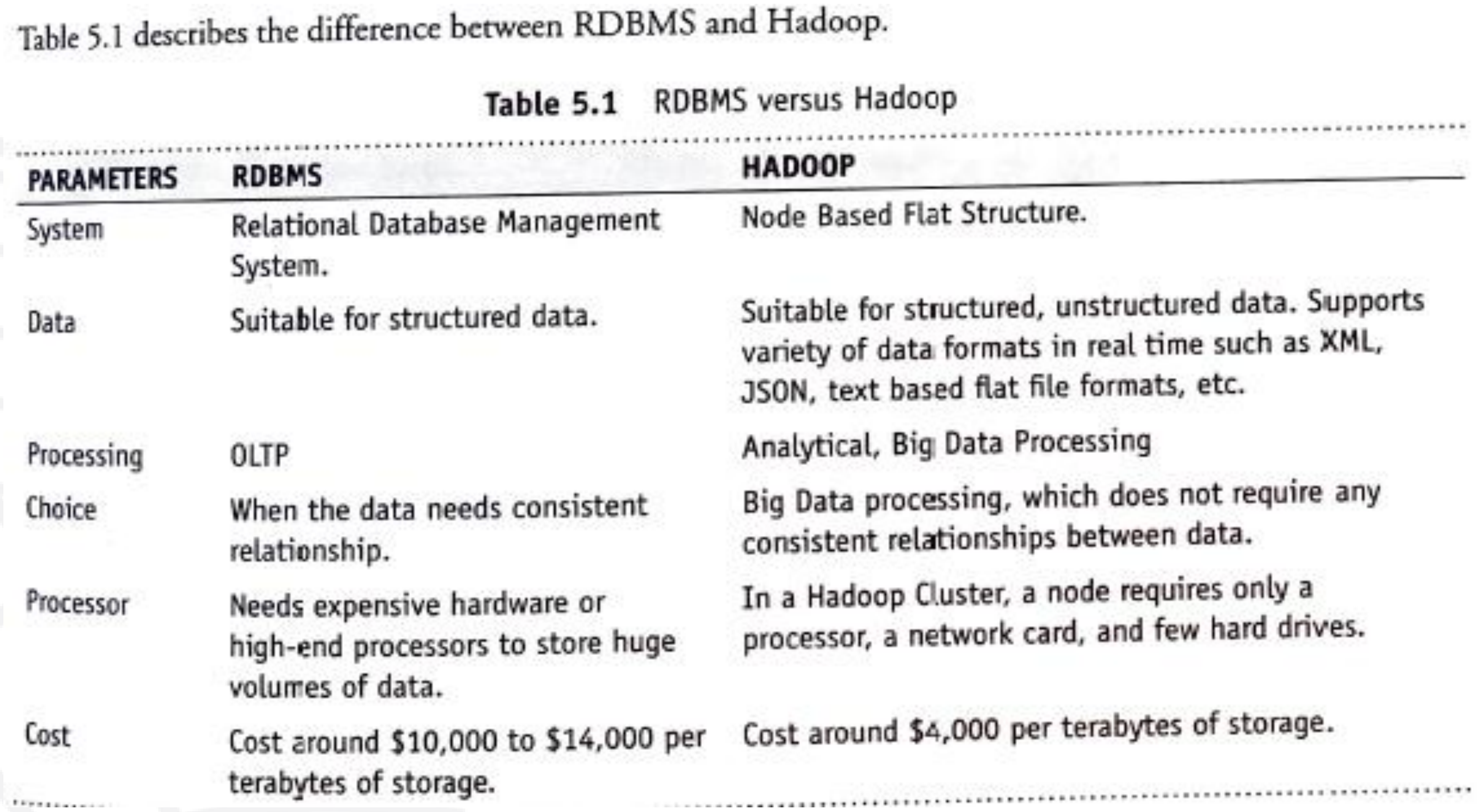


Table 5.1 describes the difference between RDBMS and Hadoop.

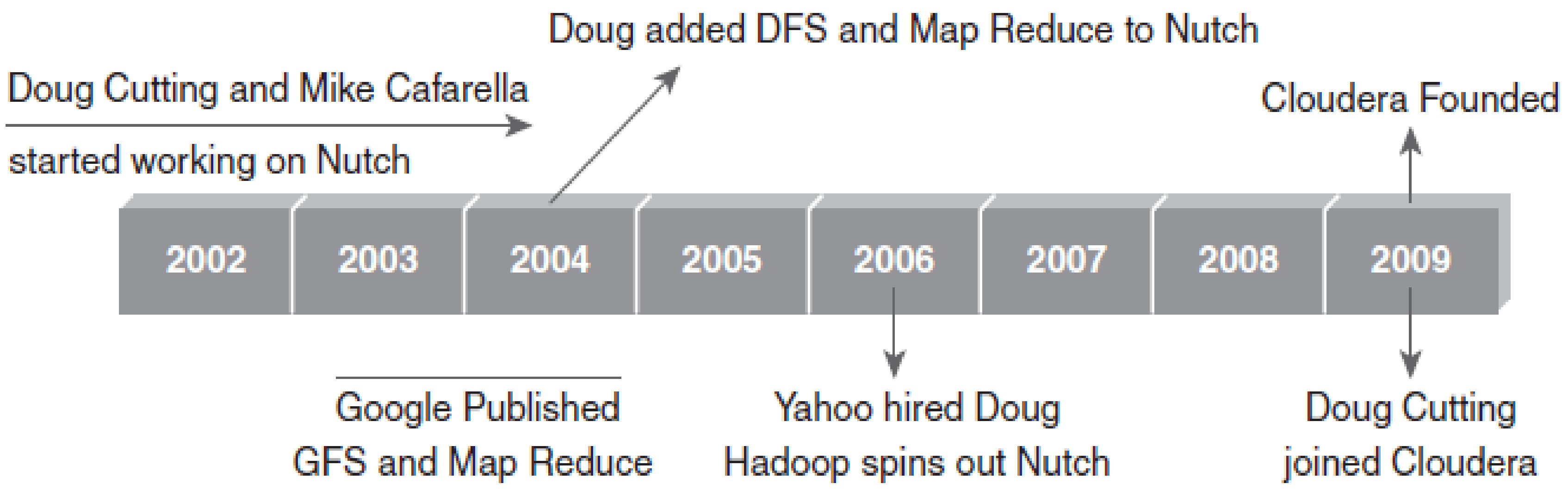
Table 5.1 RDBMS versus Hadoop

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.

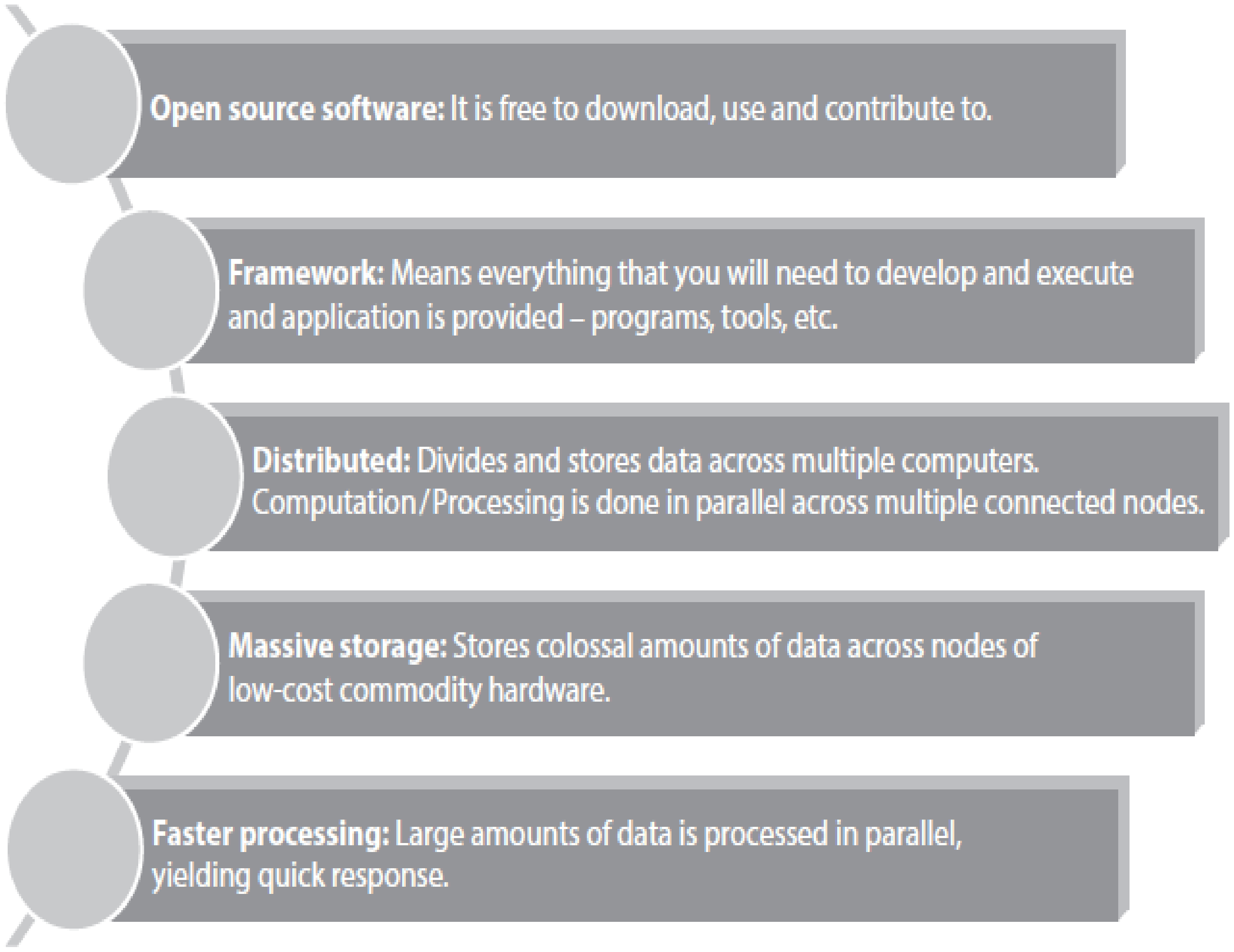
- **<u>5.5. Distributed computing challenges</u>**
- **5.5.1. Hardware Failure**
- In distributed system, several servers are networked together. This implies that more often than not, there may be a possibility of hardware failure.
- Hadoop uses ***Replication Factor*** to decide number of replication of the data.

- **5.5.2. How to process this Gigantic store of data?**
- In distributed system, data is spread across the network on several machines.
- Key challenge is to integrate the data.
- Hadoop uses MapReduce programming to solve this challenge.

- **5.6. History of Hadoop**


Doug added DFS and Map Reduce to Nutch
Doug Cutting and Mike Cafarella
started working on Nutch
Cloudera Founded
2002
2003
2004
2005
2006
2007
2008
2009
Google Published GFS and Map Reduce
Yahoo hired Doug Hadoop spins out Nutch
Doug Cutting joined Cloudera

- <u>**5.7. Hadoop Overview**</u>
- **5.7.1. Key aspects of Hadoop**



Open source software: It is free to download, use and contribute to.

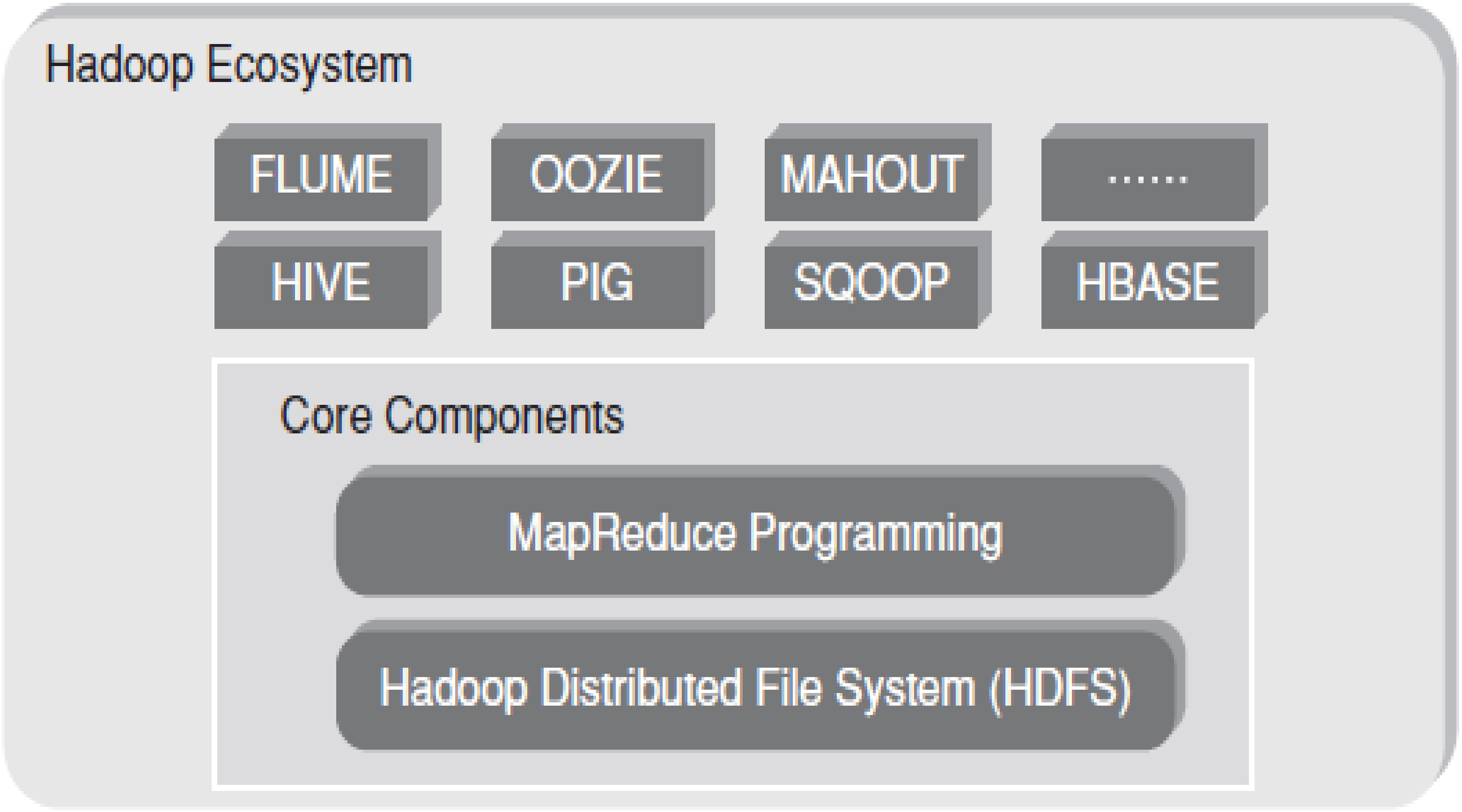
Framework: Means everything that you will need to develop and execute and application is provided – programs, tools, etc.

Distributed: Divides and stores data across multiple computers. Computation/Processing is done in parallel across multiple connected nodes.

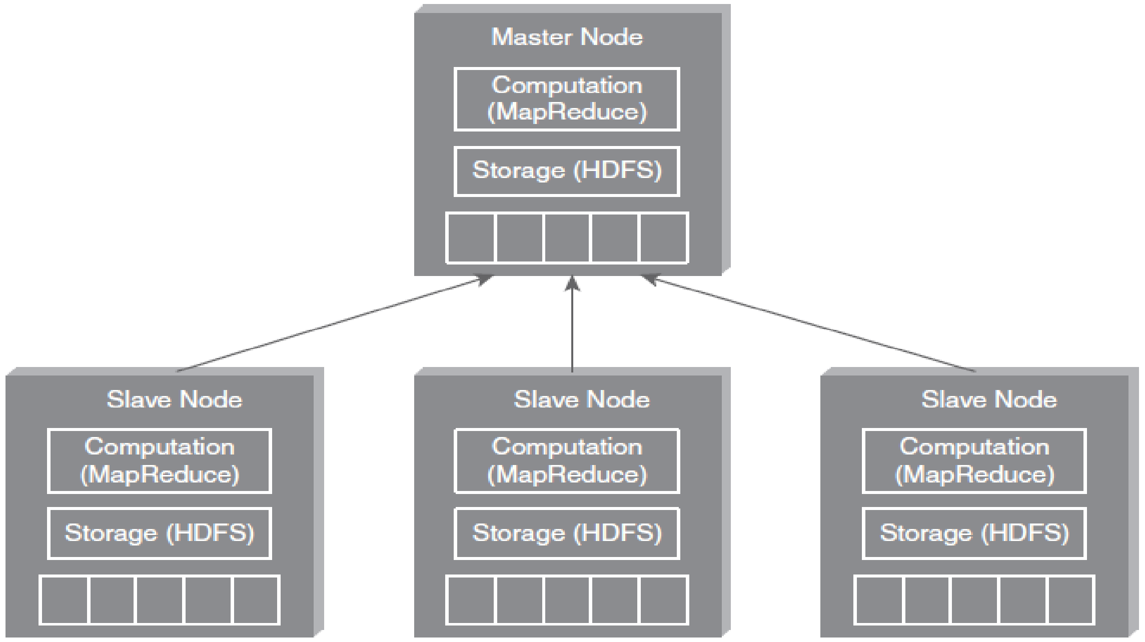
Massive storage: Stores colossal amounts of data across nodes of low-cost commodity hardware.

Faster processing: Large amounts of data is processed in parallel, yielding quick response.

- **<u>5.7. Hadoop Overview</u>**
- **5.7.2. Hadoop Components**

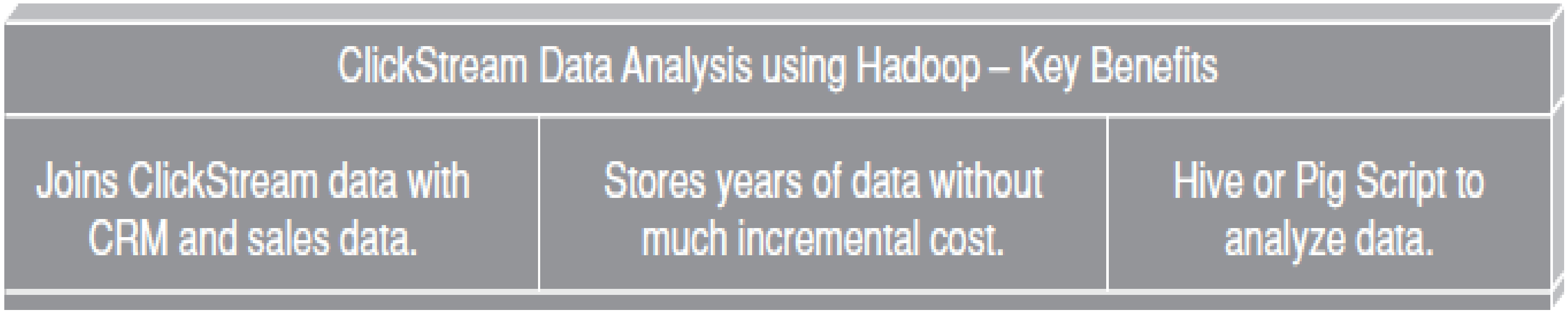


- **5.7.3. Hadoop Conceptual Layer**
- It involves Data Storage Layer and Data Processing Layer.
- **5.7.4 High-level Architecture of Hadoop**



- **Master HDFS** partitions the data storage across the slave nodes. It also keep track of locations of data on DataNodes.
- **Master MapReduce** decides and schedules computation task on slave nodes.

- **5.8. Use Case of Hadoop**
- ClickStream Data: It’s a mouse click data that helps you to understand the purchasing behaviour of customers. It helps online marketers to optimize their product web pages, promotional content, etc. to improve their business.



ClickStream Data Analysis using Hadoop – Key Benefits		
Joins ClickStream data with CRM and sales data.	Stores years of data without much incremental cost.	Hive or Pig Script to analyze data.

- **5.10. HDFS (Hadoop Distributed File System)**
- Key points of HDFS:

1. Storage component of Hadoop.
2. Distributed File System.
3. Modelled after Google File System.
4. Optimized for high throughput.
5. Can replicate a file for a configured number of times, which is tolerant in terms of both software and hardware.
6. Re-replicates data blocks automatically on nodes that have failed.
7. Helps in read an write of large files.
8. Sits on top of native file system such as ext3 and ext4.

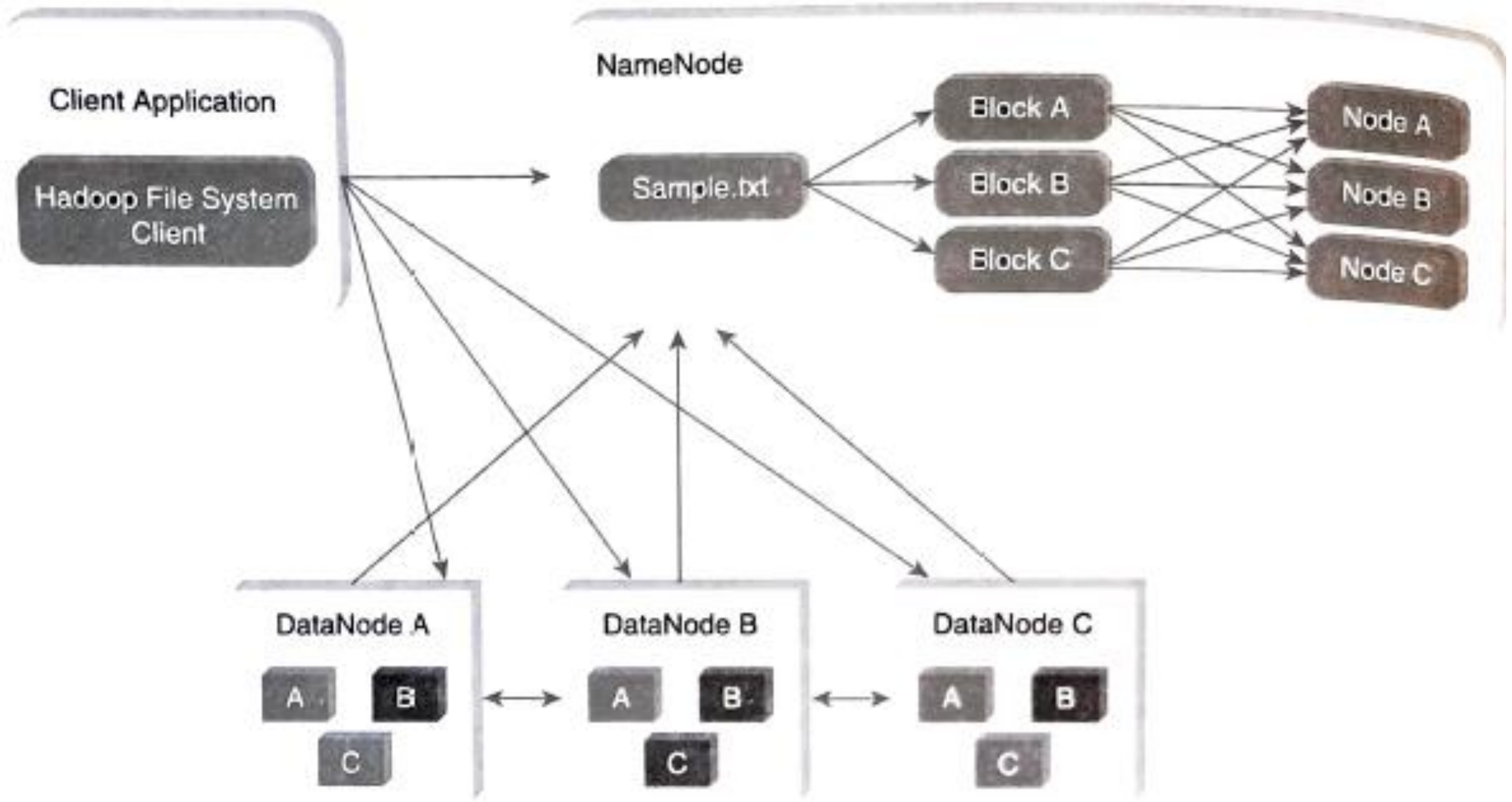


Figure 5.15 Hadoop Distributed File System Architecture.
Reference: Hadoop in Practice, Alex Holmes.

- **5.10.1 HDFS Daemons**
- **NameNode:** HDFS divides large file into smaller pieces called blocks.
- NameNode uses rack ID to identify DataNodes in the rack.
- A rack is a collection of DataNodes within the cluster.
- NameNode keeps tracks of blocks of a file as it is placed on various DataNodes.
- NameNode manages file-related operations such as read, write, create, and delete.
- Its main job is managing the file system namespace, which is a collection of files in the cluster. NameNode stores HDFS namespace.
- There is single NameNode per cluster.

- **5.10.1 HDFS Daemons**
- **DataNode:** There are multiple DataNodes per cluster. During Pipeline read and write DataNodes communicate with each other. A DataNode also continuously sends "heartbeat" message to NameNode to ensure the connectivity between the NameNode and DataNode.
- In case there is no heartbeat from DataNode, the NameNode replicates that DataNode within the cluster and keeps on running as if nothing had happened.

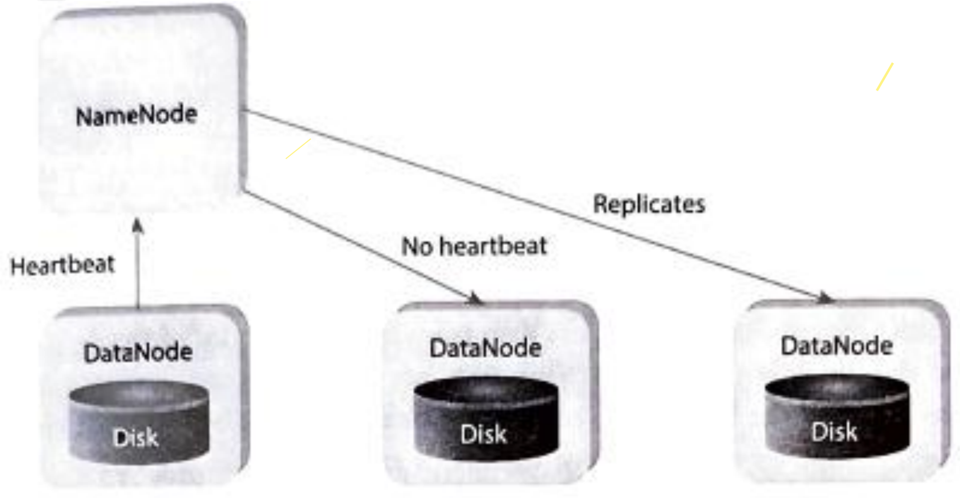
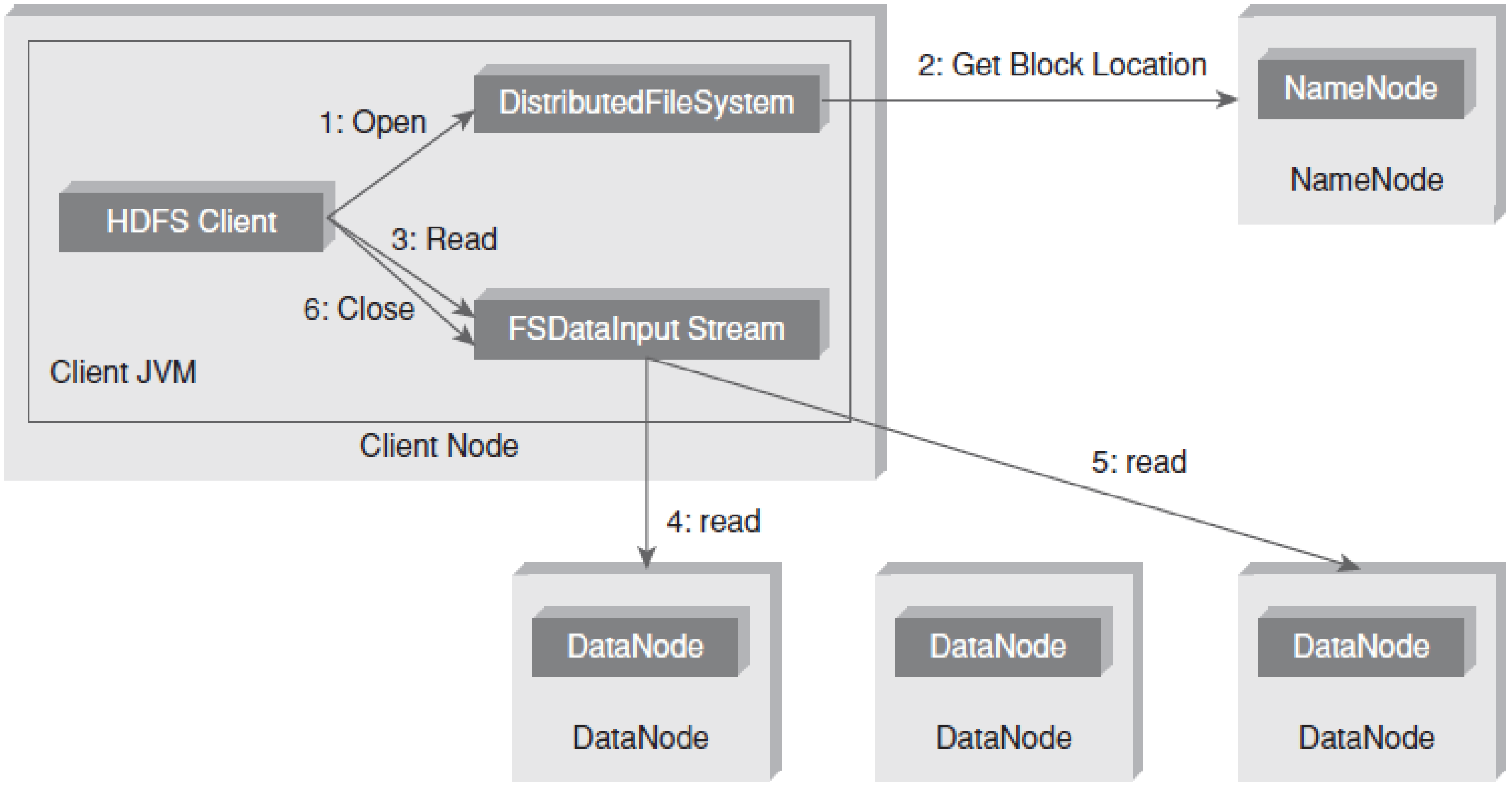


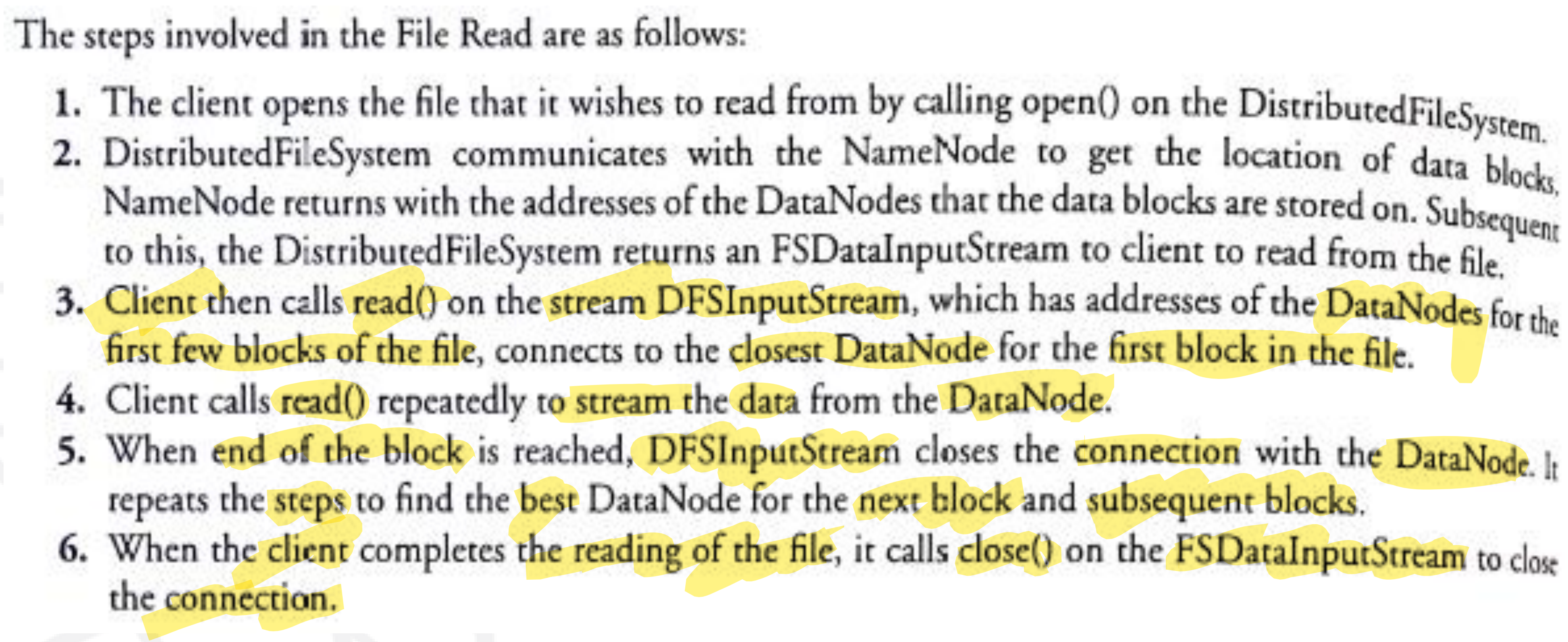
Figure 5.17 NameNode and DataNode Communication.

- **5.10.1 HDFS Daemons**
- **Secondary NameNode:** It takes snapshot of HDFS metadata at intervals specified in the Hadoop configuration. Since memory requirements of Secondary NameNode are same as NameNode, its better to run NameNode and Secondary NameNode on different machines.
- Secondary NameNode does not record any real-time changes that happen to the HDFS metadata.

- ## 5.10.2 Anatomy of File Read



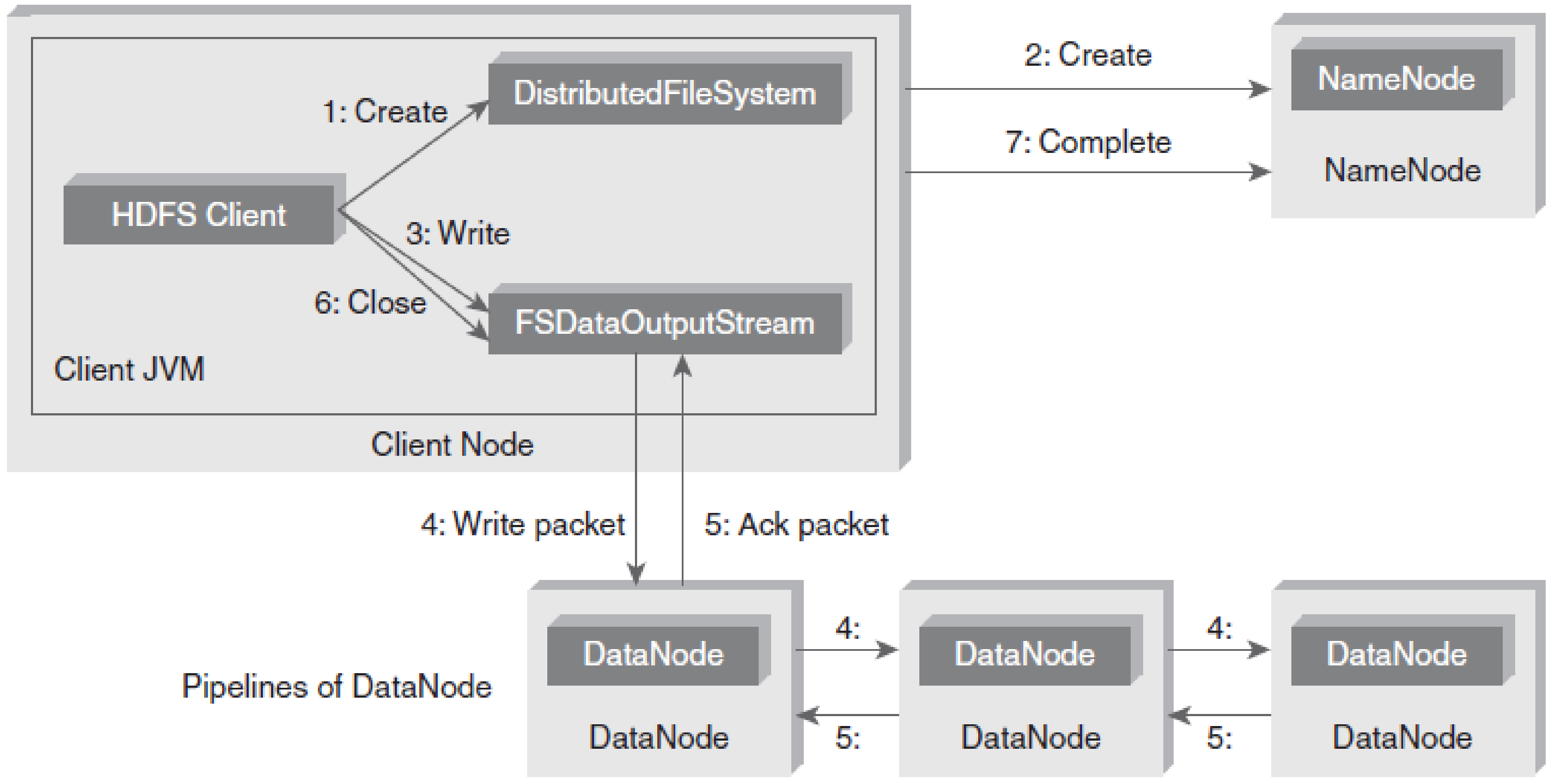
5.10.2 Anatomy of File Read



The steps involved in the File Read are as follows:

1. The client opens the file that it wishes to read from by calling open() on the DistributedFileSystem.
2. DistributedFileSystem communicates with the NameNode to get the location of data blocks. NameNode returns with the addresses of the DataNodes that the data blocks are stored on. Subsequent to this, the DistributedFileSystem returns an FSDataInputStream to client to read from the file.
3. Client then calls read() on the stream DFSInputStream, which has addresses of the DataNodes for the first few blocks of the file, connects to the closest DataNode for the first block in the file.
4. Client calls read() repeatedly to stream the data from the DataNode.
5. When end of the block is reached, DFSInputStream closes the connection with the DataNode. It repeats the steps to find the best DataNode for the next block and subsequent blocks.
6. When the client completes the reading of the file, it calls close() on the FSDataInputStream to close the connection.

- ## 5.10.2 Anatomy of File Write



• 5.10.2 Anatomy of File Write

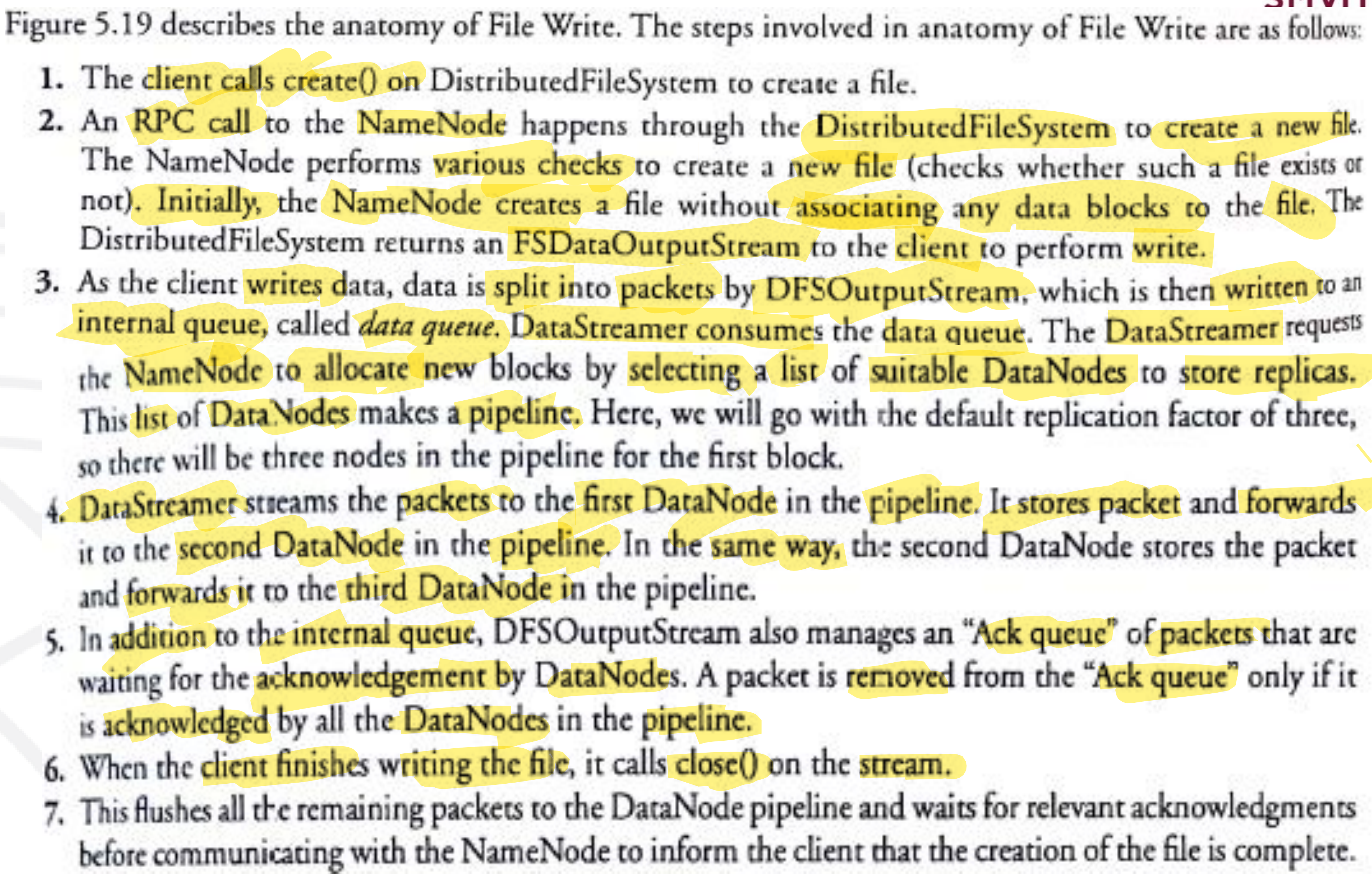
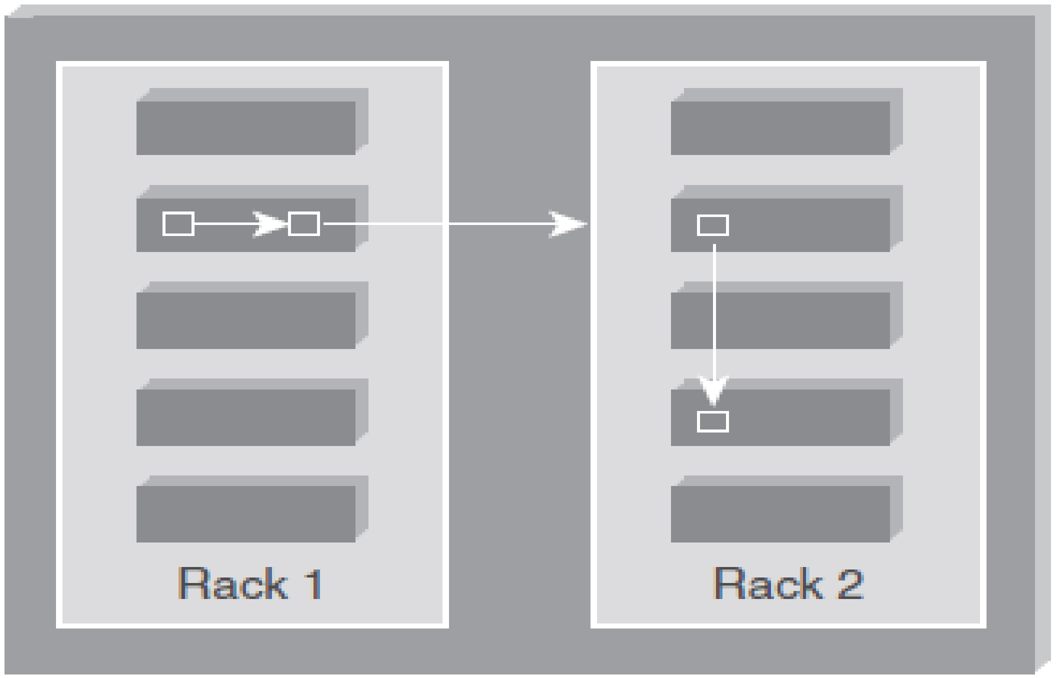


Figure 5.19 describes the anatomy of File Write. The steps involved in anatomy of File Write are as follows:

1. The client calls create() on DistributedFileSystem to create a file.
2. An RPC call to the NameNode happens through the DistributedFileSystem to create a new file. The NameNode performs various checks to create a new file (checks whether such a file exists or not). Initially, the NameNode creates a file without associating any data blocks to the file. The DistributedFileSystem returns an FSDataOutputStream to the client to perform write.
3. As the client writes data, data is split into packets by DFSOutputStream, which is then written to an internal queue, called *data queue*. DataStreamer consumes the data queue. The DataStreamer requests the NameNode to allocate new blocks by selecting a list of suitable DataNodes to store replicas. This list of DataNodes makes a pipeline. Here, we will go with the default replication factor of three, so there will be three nodes in the pipeline for the first block.
4. DataStreamer streams the packets to the first DataNode in the pipeline. It stores packet and forwards it to the second DataNode in the pipeline. In the same way, the second DataNode stores the packet and forwards it to the third DataNode in the pipeline.
5. In addition to the internal queue, DFSOutputStream also manages an "Ack queue" of packets that are waiting for the acknowledgement by DataNodes. A packet is removed from the "Ack queue" only if it is acknowledged by all the DataNodes in the pipeline.
6. When the client finishes writing the file, it calls close() on the stream.
7. This flushes all the remaining packets to the DataNode pipeline and waits for relevant acknowledgments before communicating with the NameNode to inform the client that the creation of the file is complete.

- **5.10.4 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.

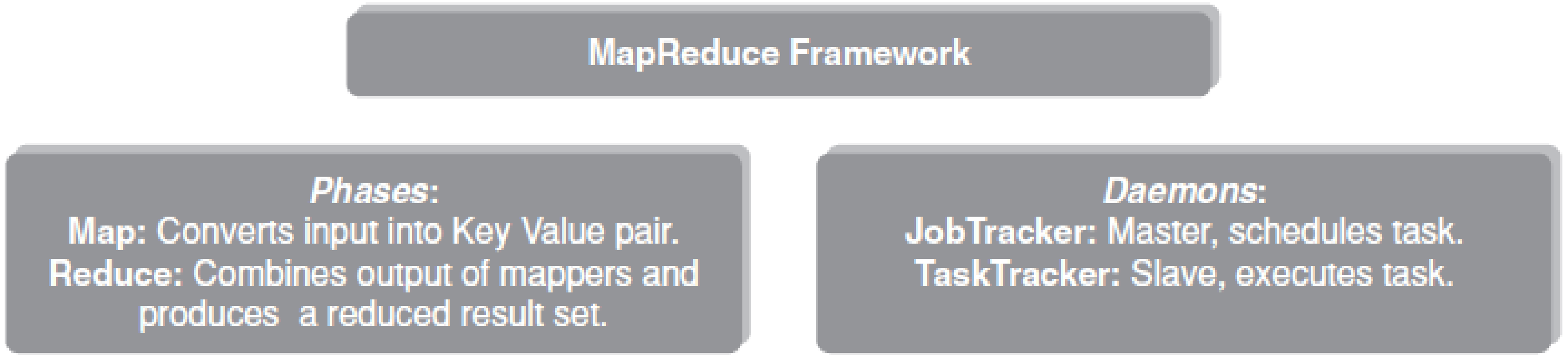


- **5.10.6 Special Features of HDFS**
- Data Replication: There is absolutely no need for a client application to track all blocks. It directs the client to the nearest replica to ensure high performance.

- Data Pipeline: A client application writes a block to the first DataNode in the pipeline. Then this DataNode takes over and forwards the data to the next node in the pipeline. This process continues for all the data blocks, and subsequently all the replicas are written to the disk.

- **<u>5.11 Processing Data with Hadoop</u>**
- In MapReduce programming, the input dataset is split into independent chunks. Map tasks process these independent chunks completely in a parallel manner. The output produced by the map tasks serves as intermediate data and is stored on the local disk of the server.
- The output of the mappers are automatically shuffled and sorted by the framework.
- MapReduce framework sorts the output based on keys. This sorted output becomes the input to the reduce task.
- Reduce task provides reduced output by combining the output of various mappers.
- Job inputs and outputs are stored in file system.
- MapReduce also handles tasks like scheduling, monitoring, re-executing failed tasks, etc.

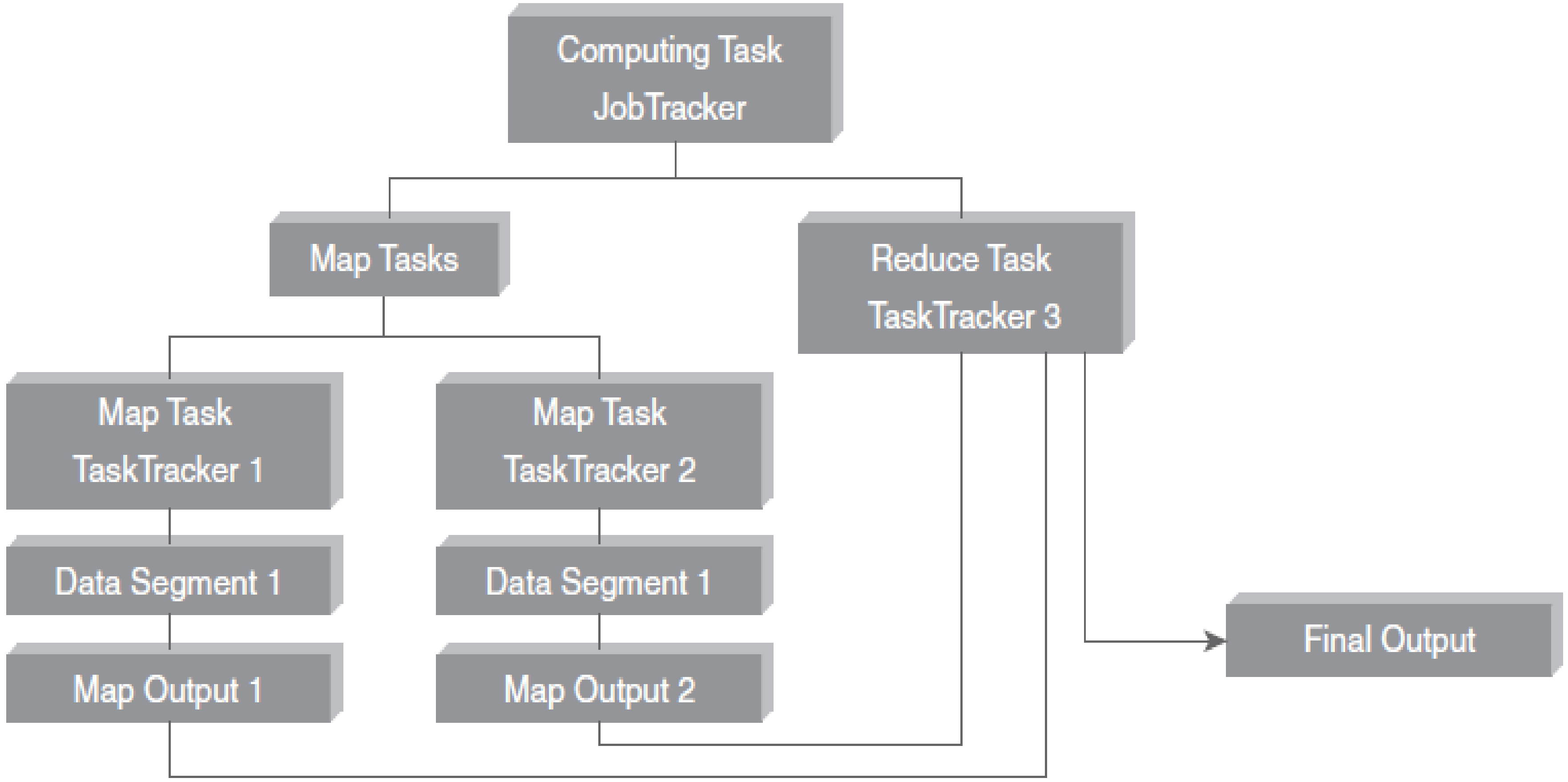
- **5.11 Processing Data with Hadoop**


MapReduce Framework
Phases:
Map: Converts input into Key Value pair.
Reduce: Combines output of mappers and produces a reduced result set.
Daemons:
JobTracker: Master, schedules task.
TaskTracker: Slave, executes task.

- **5.11.1 MapReduce Daemons**

1. JobTracker: It provides connectivity between Hadoop and user application. When user submit code to cluster, JobTracker creates the execution plan by deciding which task to assign to which node. It also monitors all running tasks.
2. TaskTracker: This daemon is responsible for executing individual tasks that is assigned by the JobTrackerr. There is a single TaskTracker per slave and spawns multiple Java Virtual Machines to handle multiple map and reduce tasks in parallel.

• 5.11.2 How Does MapReduce Work?



- **<u>5.12 Managing Resources and Applications with Hadoop YARN</u>**
- YARN is Hadoop 2.x based architecture. Which handles the resource management task.
- **5.12.1. 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 ML algorithms, graphs, and other memory intensive algorithms.
5. MapReduce is responsible for cluster resource management and data processing.

- **5.12.2. HDFS Limitations:**
- NameNode saves all its file metadata in main memory. Although the main memory is not as small and as expensive as before, still there is limit on the number of objects that one can have in the memory on a single NameNode.
- This problem is resolved with help of HDFS Federation in Hadoop 2.x.
- **5.12.3 Hadoop 2: HDFS**
- It consists of 2 major components: 1) namespace, 2) blocks storage service.
- Namespace service takes care of file-related operations, such as creating and modifying files and directories. Block storage service handles data node cluster management, replication.

- **HDFS 2 Features:**
  - Horizontal scalability – HDFS federation using multiple independent name nodes to support scalability. These name nodes need not to have coordination with each other
  - High availability – This is obtained with the help of Passive Standby NameNode. These NameNodes handle failover automatically. Passive NameNodes reads, edits from shared storage and keeps the metadata updated. In case of active NameNode failure, passive NameNode becomes active automatically.

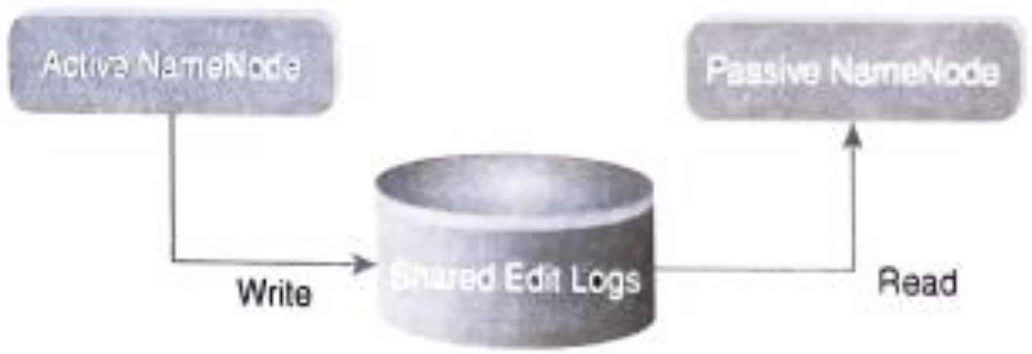
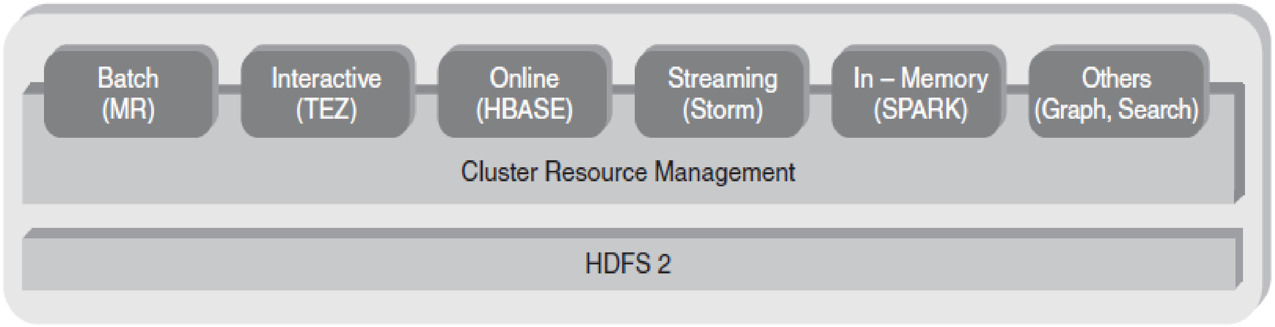


Figure 5.26 Active and Passive NameNode interaction.

5.12.4 Hadoop 2 YARN: Taking Hadoop beyond Batch:



- The fundamental idea behind this architecture is splitting the JobTracker responsibility of resource management and Job Scheduling/Monitoring into separate daemons. Daemons that are part of YARN Architecture are described below.
- A Global ResourceManager: Its main responsibility is to distribute resources among various applications in the system. It has two main components: Scheduler and Application Manager (Accept job, negotiate resource, restart).
- NodeManager: This is a per-machine slave daemon. NodeManager responsibility is launching the application containers for application execution. It monitors the resource usage such as memory, CPU, disk, network, etc. It then reports the usage of resources to the global Resource storage.
- Per-application ApplicationMaster: This is an application-specific entity. Its responsibility is to negotiate required resources for execution from the ResourceManager. It works along with the NodeManager for executing and monitoring component tasks.

- ## YARN Architecture:

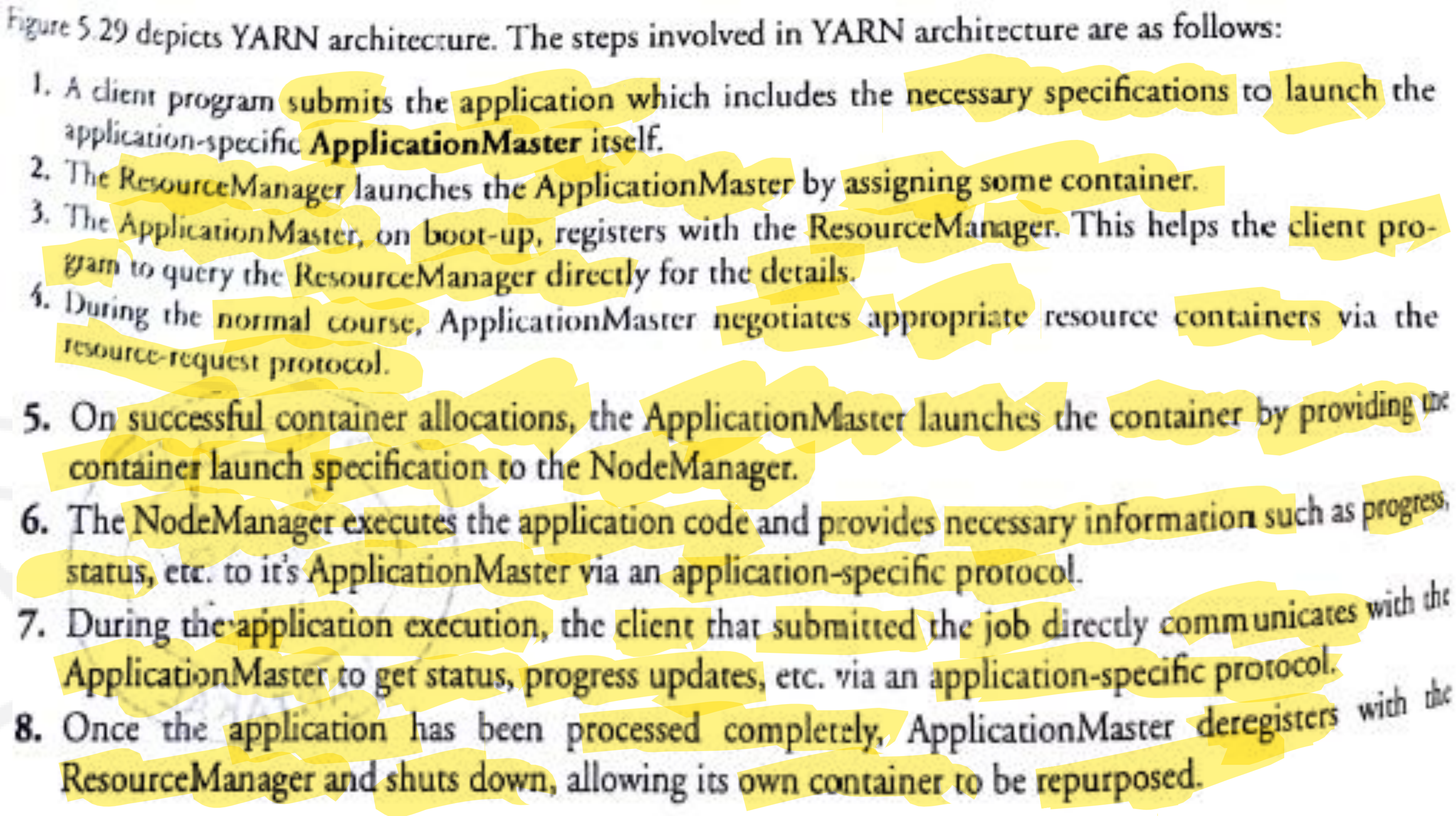


Figure 5.29 depicts YARN architecture. The steps involved in YARN architecture are as follows:

1. A client program submits the application which includes the necessary specifications to launch the application-specific **ApplicationMaster** itself.
2. The ResourceManager launches the ApplicationMaster by assigning some container.
3. The ApplicationMaster, on boot-up, registers with the ResourceManager. This helps the client program to query the ResourceManager directly for the details.
4. During the normal course, ApplicationMaster negotiates appropriate resource containers via the resource-request protocol.
5. On successful container allocations, the ApplicationMaster launches the container by providing the container launch specification to the NodeManager.
6. The NodeManager executes the application code and provides necessary information such as progress, status, etc. to it's ApplicationMaster via an application-specific protocol.
7. During the application execution, the client that submitted the job directly communicates with the ApplicationMaster to get status, progress updates, etc. via an application-specific protocol.
8. Once the application has been processed completely, ApplicationMaster deregisters with the ResourceManager and shuts down, allowing its own container to be repurposed.

- **YARN Architecture:**

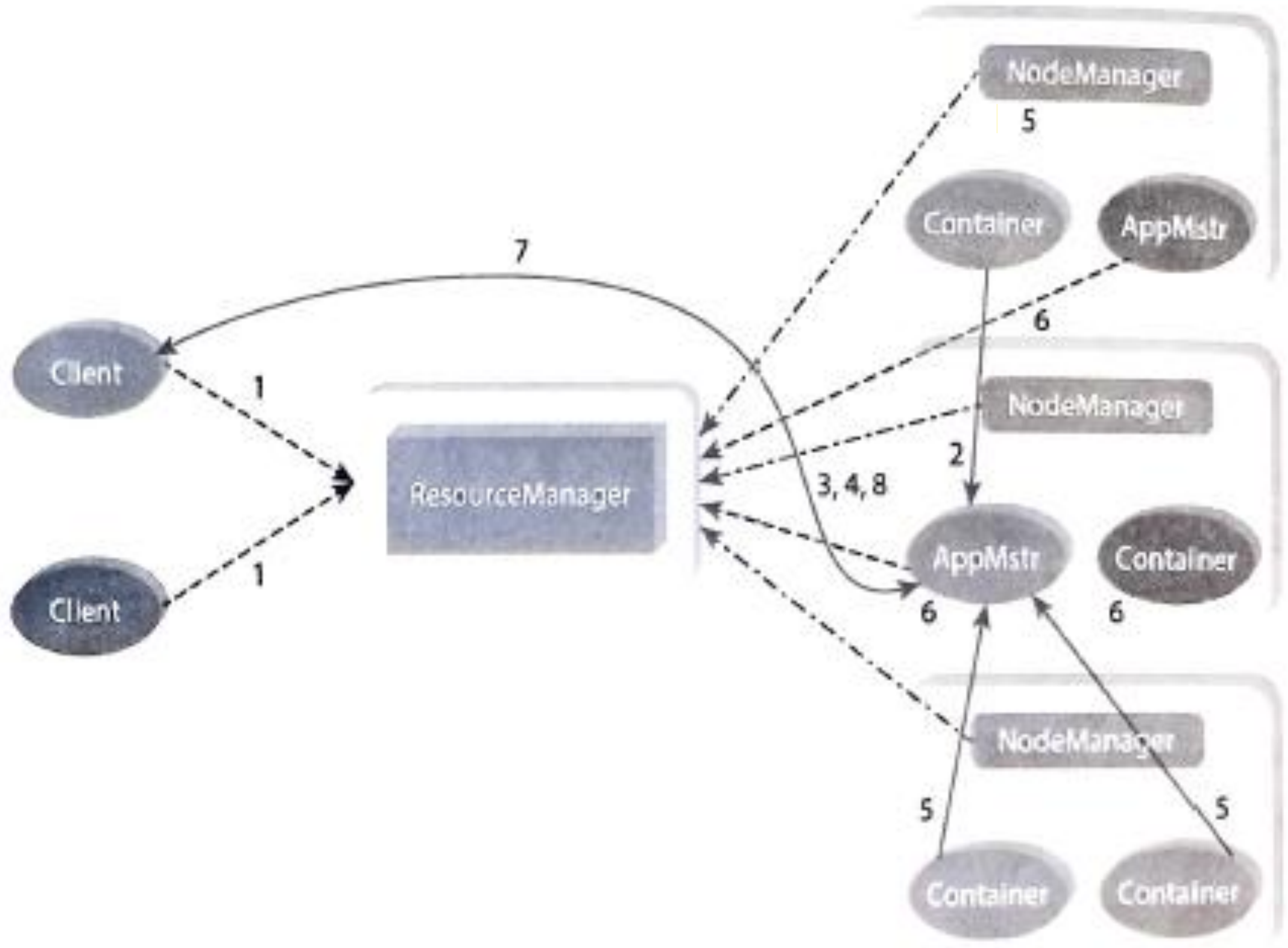


Figure 5.29 YARN architecture.

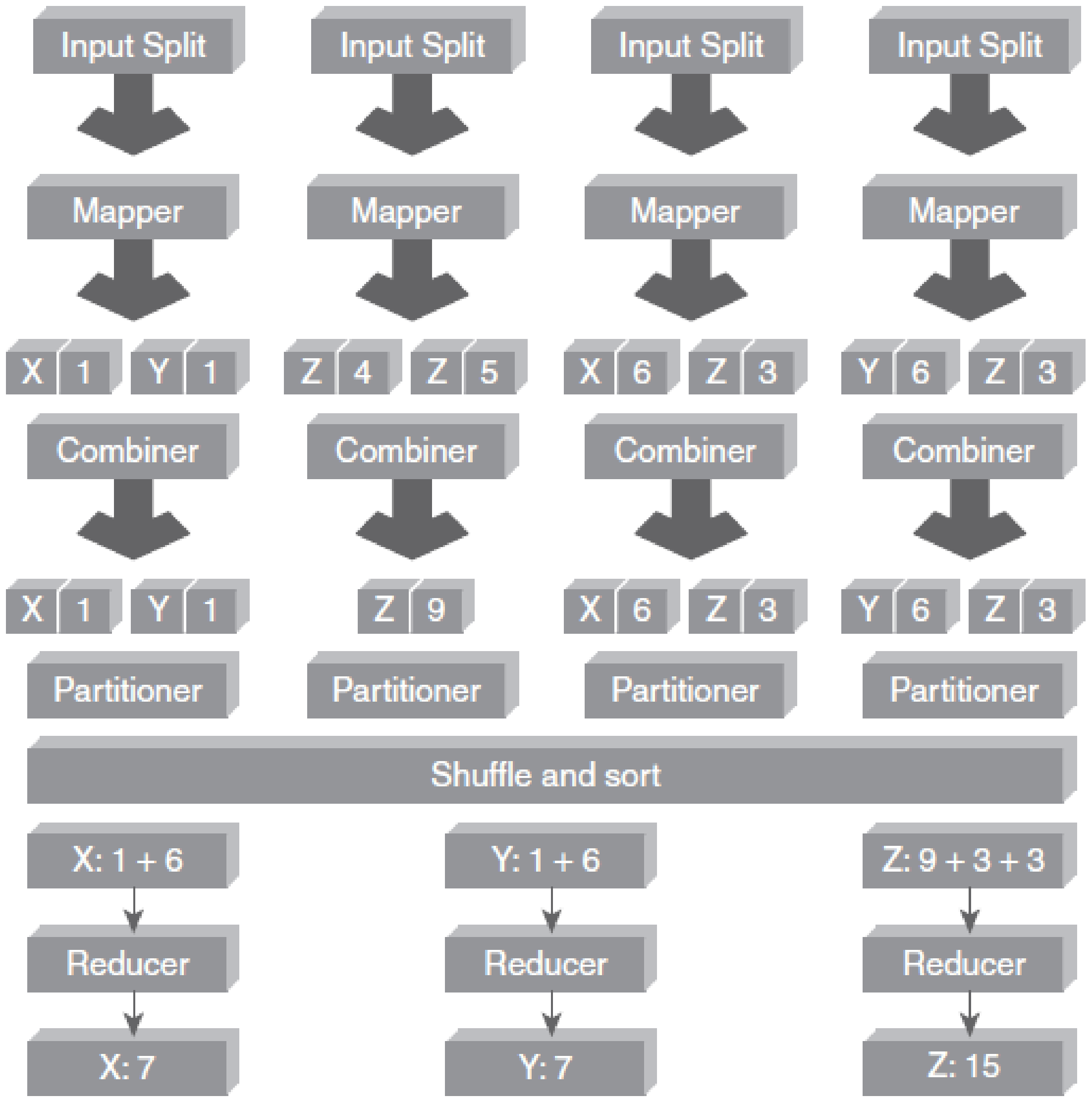
Module 2
Chapter 8 – Introduction to MapReduce Programming

- **8.1 Introduction**
- In MapReduce Programming, Jobs (Applications) are split into a set of map tasks and reduce tasks. Then these tasks arc executed in a distributed fashion on Hadoop cluster.
- Each task processes small subset of data that has been assigned to it. This way, Hadoop distributes the load across the cluster. MapReduce job takes a set of files that is stored in HDFS (Hadoop Distributed File System) as input.
- Map task takes care of loading. parsing, transforming, and filtering. The responsibility of reduce task is grouping and aggregating data that is produced by map tasks to generate final output.
- Each map task is broken into the following phases: 1) RecordReader. 2) Mapper. 3) Combiner. 4) Partitioner.
- The reduce tasks are broken into the following phases: 1) Shuffle. 2) Sort. 3) Reducer. 4) Output Format.

- **8.2 Mapper**
- A mapper maps the input key-value pairs into a set of intermediate key-value pairs. Maps are individual tasks that have the responsibility of transforming input records into intermediate key-value pairs.

1. RecordReader: RecordReader converts a byte-oriented view of the input (as generated by the Input Split) into a record-oriented view and presents it to the Mapper tasks. It presents the tasks with Keys and values. Generally the key is the positional information and value is a chunk of data that constitutes the record.
2. Map: Map function works on the key-value pair produced by Record Reader and generates zero or more intermediate key-value pairs. The MapReduce decides the key-value pair based on the context.
3. Combiner: It is an optional function but provides high performance in terms of network bandwidth and disk space. It takes intermediate key-value pair provided by mapper and applies user-specific aggregate function to only that mapper. It is also known as local reducer.
4. Partitioner: The partitioner takes the intermediate key-value pairs produced by the mapper, splits them into shard, and sends the shard to the particular reducer as per the user-specific code. Usually, the key with same values goes to the same reducer. The partitioned data of each map task is written to the local disk of that machine and pulled by the respective reducer.

- **8.3 Reducer**
- The primary chore of the Reducer is to reduce a set of intermediate values (the ones that share a common key) to a smaller set of values. The Reducer has three primary phases: Shuffle and Sort, Reduce, and Output Format.
  1. Shuffle and Sort: This phase takes the output of all the partitioners and downloads them into the local machine where the reducer is running. Then these individual data pipes are sorted by keys which produce larger data list. The main purpose of this sort is grouping similar words so that their values can be easily iterated over by the reduce task.
  2. Reduce: The reducer takes the grouped data produced by the shuffle and sort phase, applies reduce function, and processes one group at a time. The reduce function iterates all the values associated with that key. Reducer function provides various operations such as aggregation, filtering, and combining data. Once it is done, the output (zero or more key-value pairs) of reducer is sent to the output format.
  3. Output Format: The output format separates key-value pair with tab (default) and writes it out to a file using record writer.


Input Split
Input Split
Input Split
Input Split
Mapper
Mapper
Mapper
Mapper
X 1 Y 1
Z 4 Z 5
X 6 Z 3
Y 6 Z 3
Combiner
Combiner
Combiner
Combiner
X 1 Y 1
Z 9
X 6 Z 3
Y 6 Z 3
Partitioner
Partitioner
Partitioner
Partitioner
Shuffle and sort
X: 1 + 6
Y: 1 + 6
Z: 9 + 3 + 3
Reducer
Reducer
Reducer
X: 7
Y: 7
Z: 15

- **8.4 Combiner**
- It is an optimization technique for MapReduce Job. Generally, the reducer class is set to be the combiner class. The difference between combiner class and reducer class is as follows:
  1. Output generated by combiner is intermediate data and it is passed to the reducer.
  2. Output of the reducer is passed to the output file on disk.
- **8.5 Partitioner**
- The partitioning phase happens after map phase and before reduce phase. Usually the number of partitions are equal to the number of reducers. The default partitioner is hash partitioner.
- **8.6 Compression**
- In MapReduce programming, you can compress the MapReduce output file. Compression provides two benefits as follows:
  1. Reduces the space to store files.
  2. Speeds up data transfer across the network.