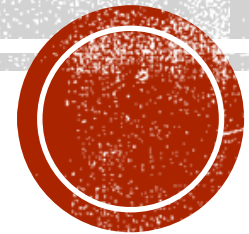


CLOUD PROGRAMMING MODEL

Dr. Manjunath Hegde
Dept. of Computer Applications
Manipal Institute of Technology, Manipal



CLOUD PROGRAMMING AND SYSTEMS

- Computing Platforms and Technologies, Eras of Computing.
- Parallel vs. Distributed Computing, Elements of Parallel Computing.
- Architectural Styles for Distributed Computing, Models for Inter-Process Communication.
- Elements of Distributed Computing.
- Cloud Application Architecture and Platforms.

PRINCIPLES OF PARALLEL AND DISTRIBUTED COMPUTING

- The term parallel computing and distributed computing are often used interchangeably, even though they mean slightly different things.
- The term parallel implies a tightly coupled system, where as distributed systems refers to a wider class of system, including those that are tightly coupled.
- More precisely, the term parallel computing refers to a model in which the computation is divided among several processors sharing the same memory.
- The architecture of parallel computing system is often characterized by the homogeneity of components: each processor is of the same type and it has the same capability as the others.

PRINCIPLES OF PARALLEL AND DISTRIBUTED COMPUTING

- The shared memory has a single address space, which is accessible to all the processors.
- Parallel programs are then broken down into several units of execution that can be allocated to different processors and can communicate with each other by means of shared memory.
- Originally parallel systems are considered as those architectures that featured multiple processors sharing the same physical memory and that were considered a single computer.
 - Over time, these restrictions have been relaxed, and parallel systems now include all architectures that are based on the concept of shared memory, whether this is physically present or created with the support of libraries, specific hardware, and a highly efficient networking infrastructure.

PRINCIPLES OF PARALLEL AND DISTRIBUTED COMPUTING

- The term distributed computing encompasses any architecture or system that allows the computation to be broken down into units and executed concurrently on different computing elements, whether these are processors on different nodes, processors on the same computer, or cores within the same processor.
- Distributed computing includes a wider range of systems and applications than parallel computing and is often considered a more general term.
- Even though it is not a rule, the term distributed often implies that the locations of the computing elements are not the same and such elements might be heterogeneous in terms of hardware and software features.
- Classic examples of distributed computing systems are
 - Computing Grids
 - Internet Computing Systems

WHAT IS PARALLEL PROCESSING?

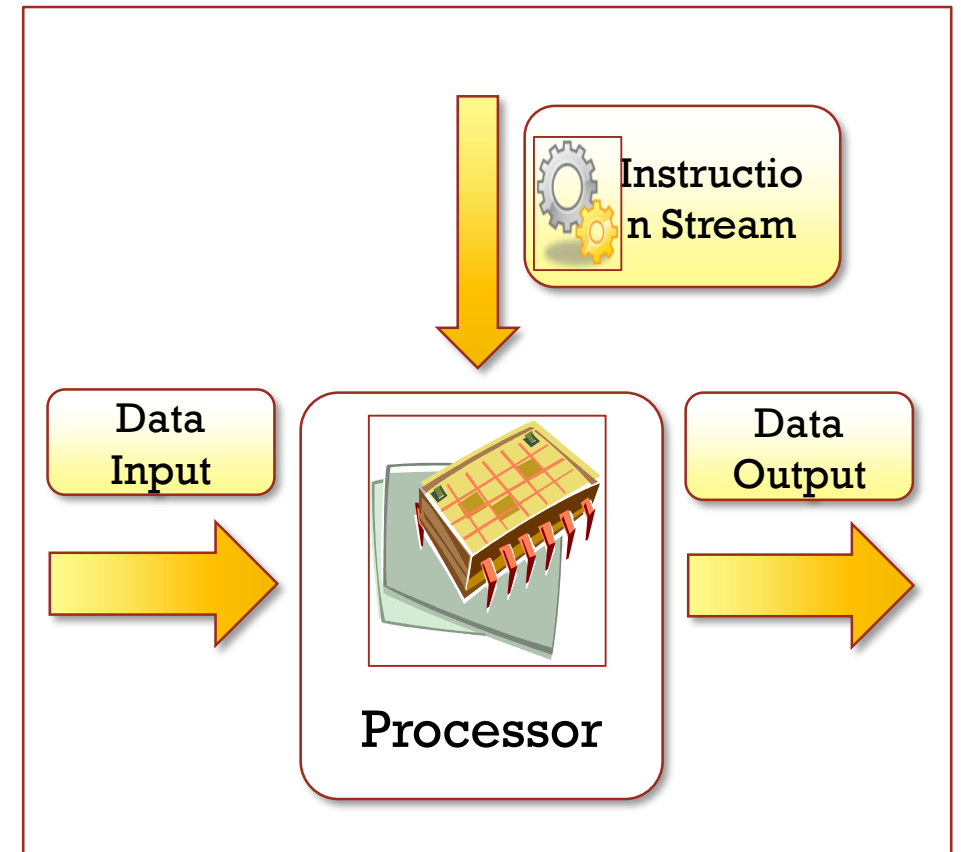
- Processing of multiple tasks simultaneously on multiple processors is called parallel processing.
- The parallel program consists of multiple active processes (tasks) simultaneously solving a given problem.
- A given task is divided into multiple subtasks using a divide-and-conquer technique, and each subtask is processed on a different central processing unit (CPU).
- Programming on multi processor system using the divide-and-conquer technique is called parallel programming.
- Many applications today require more computing power than a traditional sequential computer can offer.
- Parallel Processing provides a cost effective solution to this problem by increasing the number of CPUs in a computer and by adding an efficient communication system between them.
- The workload can then be shared between different processors. This setup results in higher computing power and performance than a single processor a system offers.

HARDWARE ARCHITECTURES FOR PARALLEL PROCESSING

- The core elements of parallel processing are CPUs. Based on the number of instructions and data streams, that can be processed simultaneously, computing systems are classified into the following four categories:
 - Single-instruction, Single-data (SISD) systems
 - Single-instruction, Multiple-data (SIMD) systems
 - Multiple-instruction, Single-data (MISD) systems
 - Multiple-instruction, Multiple-data (MIMD) systems

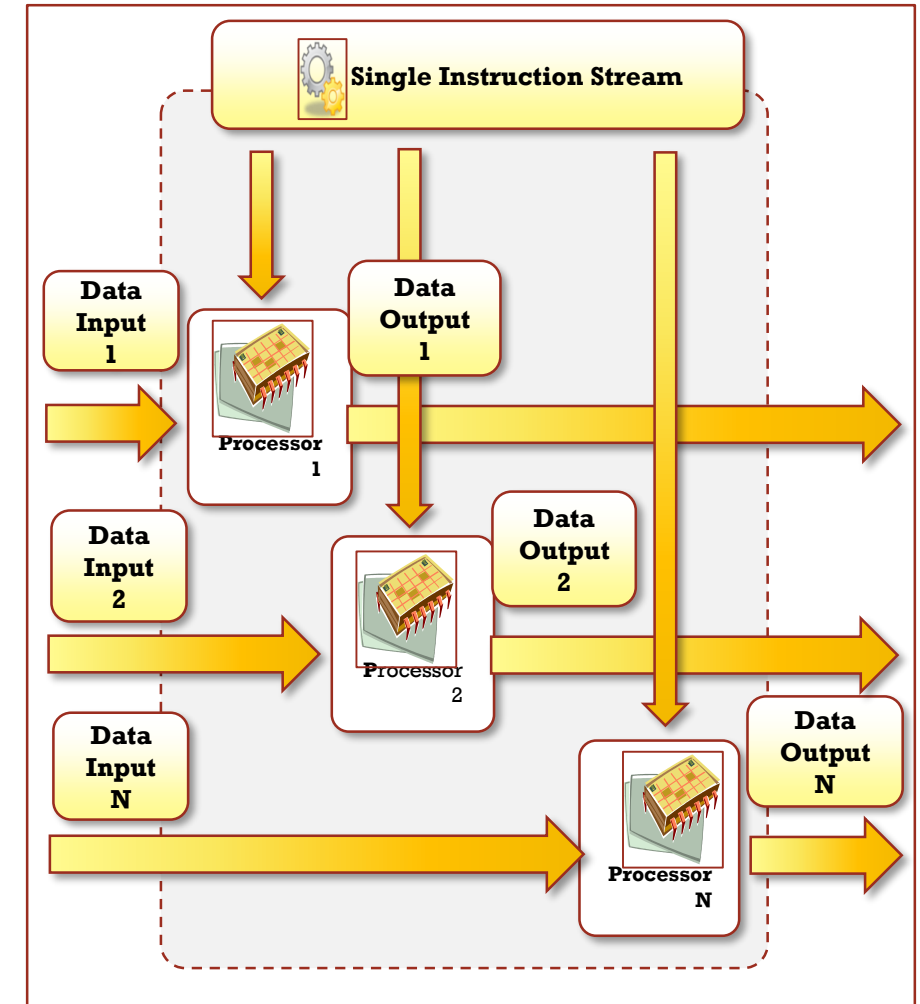
SINGLE – INSTRUCTION , SINGLE DATA (SISD) SYSTEMS

- SISD computing system is a uni-processor machine capable of executing a single instruction, which operates on a single data stream.
- Machine instructions are processed sequentially, hence computers adopting this model are popularly called sequential computers.
- Most conventional computers are built using SISD model.
- All the instructions and data to be processed have to be stored in primary memory.
- The speed of processing element in the SISD model is limited by the rate at which the computer can transfer information internally.
- Dominant representative SISD systems are IBM PC, Macintosh, and workstations.



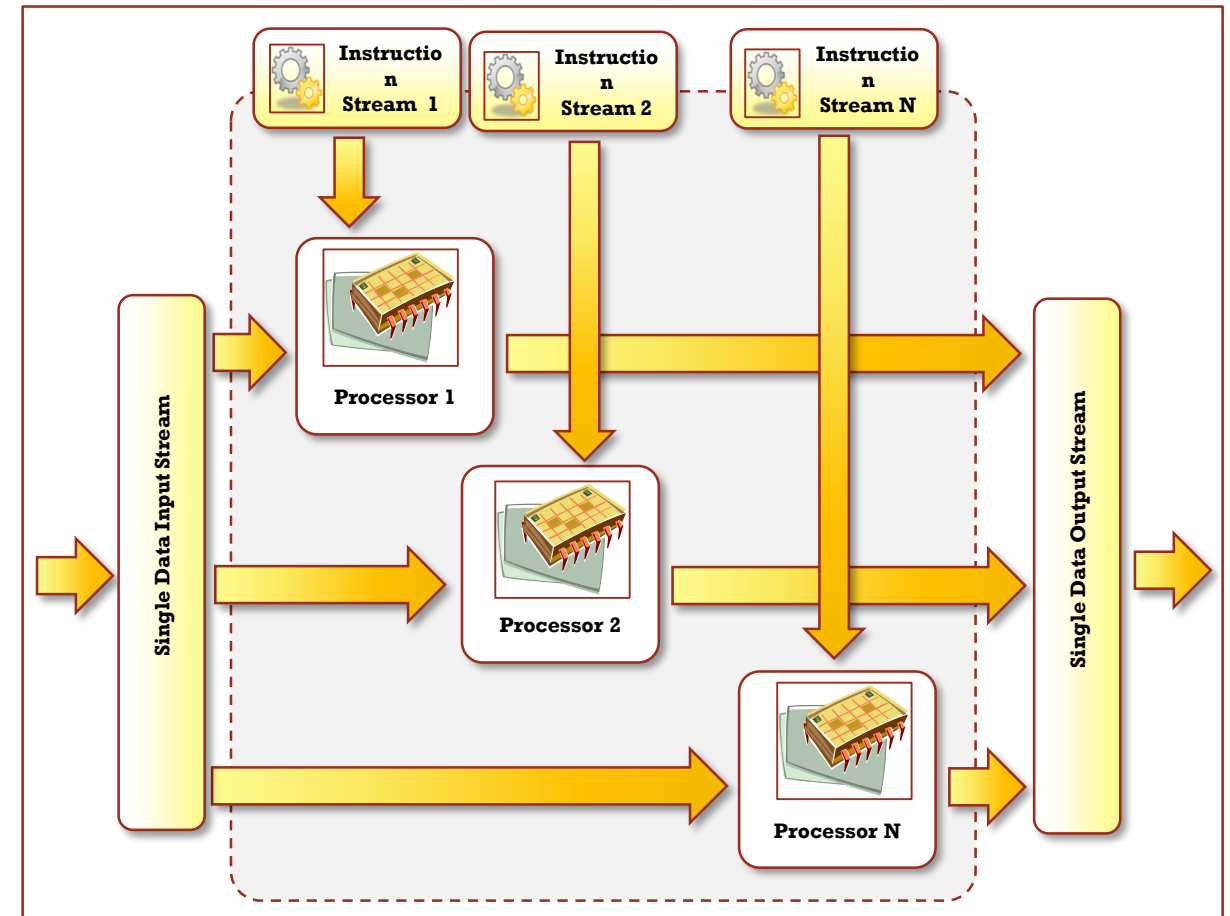
SINGLE – INSTRUCTION , MULTIPLE DATA (SIMD) SYSTEMS

- SIMD computing system is a multiprocessor machine capable of executing the same instruction on all the CPUs but operating on different data streams.
- Machines based on this model are well suited for scientific computing since they involve lots of vector and matrix operations.
- For instance statement $C_i = A_i * B_i$, can be passed to all the processing elements (PEs), organized data elements of vectors A and B can be divided into multiple sets (N- sets for N PE systems), and each PE can process one data set.
- Dominant representative SIMD systems are Cray's Vector processing machine.



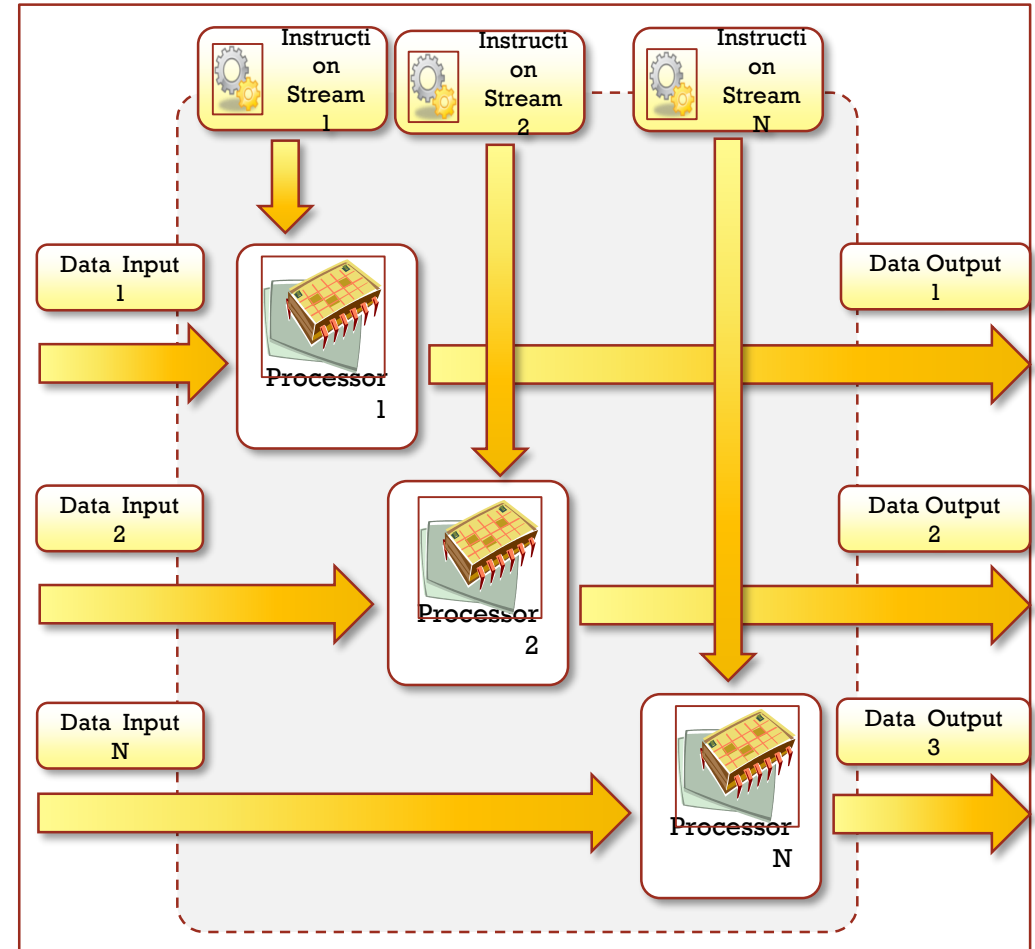
MULTIPLE – INSTRUCTION , SINGLE DATA (MISD) SYSTEMS

- MISD computing system is a multi processor machine capable of executing different instructions on different PEs all of them operating on the same data set.
- For example
 - $y = \sin(x) + \cos(x) + \tan(x)$
- Machines built using MISD model are not useful in most of the applications.
- Few machines are built but none of them available commercially.
- This type of systems are more of an intellectual exercise than a practical configuration.



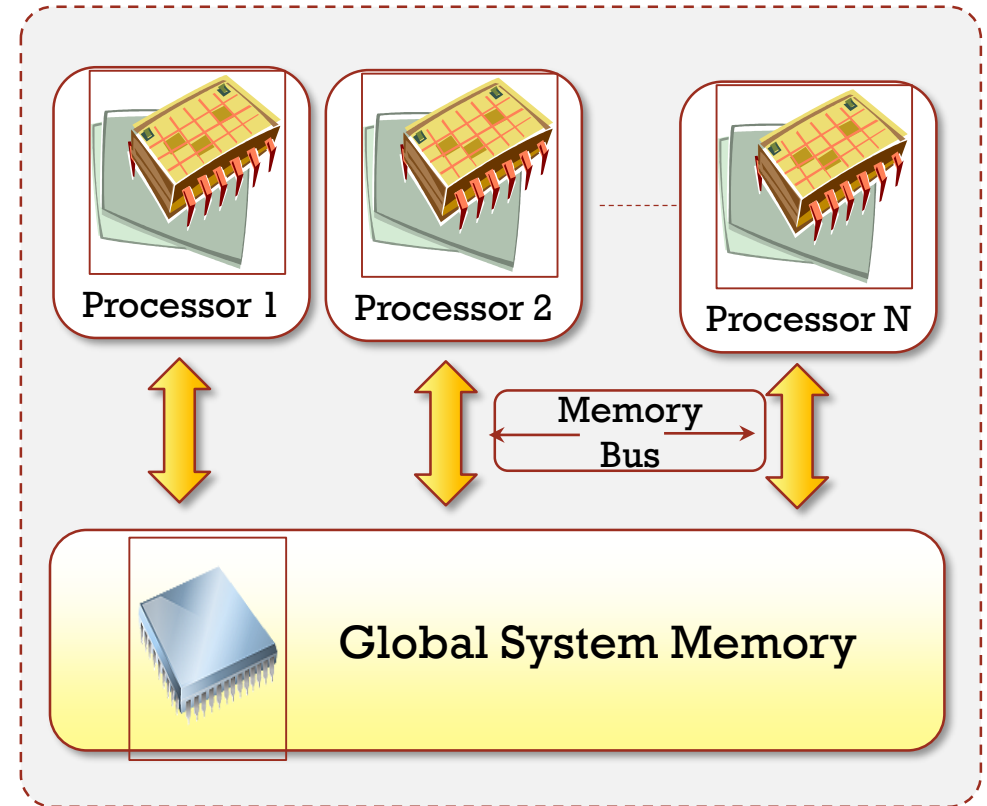
MULTIPLE – INSTRUCTION , MULTIPLE DATA (MIMD) SYSTEMS

- MIMD computing system is a multi processor machine capable of executing multiple instructions on multiple data sets.
- Each PE in the MIMD model has separate instruction and data streams, hence machines built using this model are well suited to any kind of application.
- Unlike SIMD, MISD machine, PEs in MIMD machines work asynchronously,
- MIMD machines are broadly categorized into shared-memory MIMD and distributed memory MIMD based on the way PEs are coupled to the main memory.



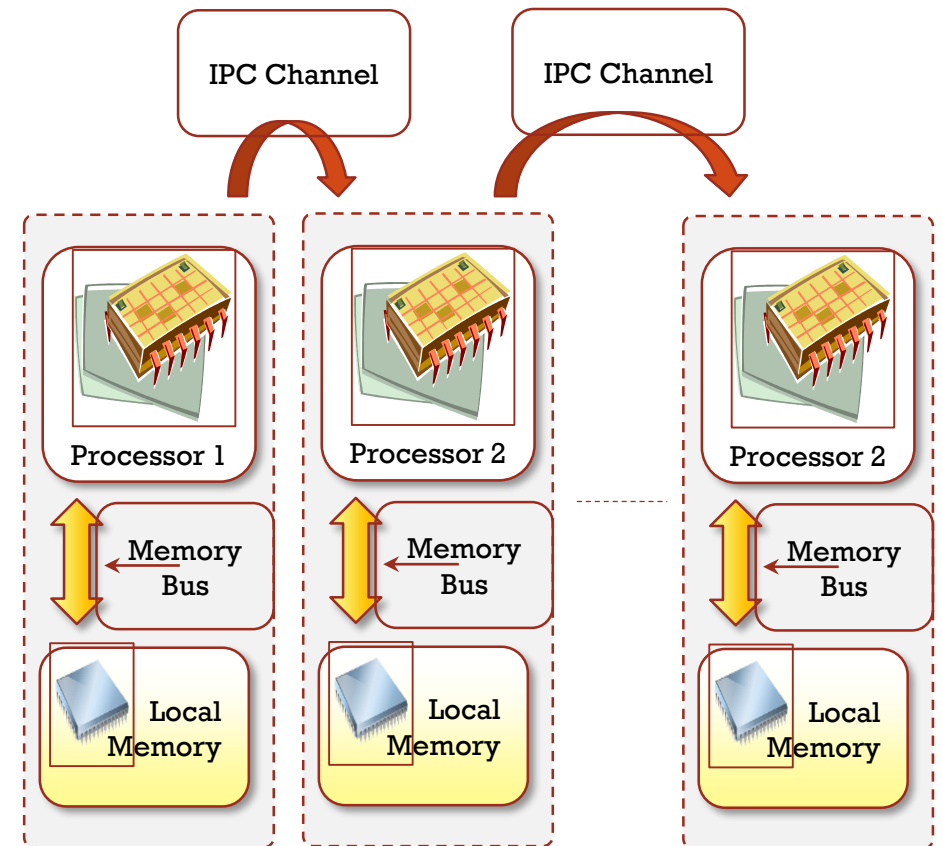
SHARED MEMORY MIMD MACHINES

- All the PEs are connected to a single global memory and they all have access to it.
- Systems based on this model are also called tightly coupled multi processor systems.
- The communication between PEs in this model takes place through the shared memory.
- Modification of the data stored in the global memory by one PE is visible to all other PEs.
- Dominant representative shared memory MIMD systems are silicon graphics machines and Sun/IBM SMP (Symmetric Multi-Processing).



DISTRIBUTED MEMORY MIMD MACHINES

- All PEs have a local memory. Systems based on this model are also called loosely coupled multi processor systems.
- The communication between PEs in this model takes place through the interconnection network, the inter process communication channel, or IPC.
- The network connecting PEs can be configured to tree, mesh, cube, and so on.
- Each PE operates asynchronously, and if communication/synchronization among tasks is necessary, they can do so by exchanging messages between them.



SHARED VS DISTRIBUTED MIMD MODEL

- The shared memory MIMD architecture is easier to program but is less tolerant to failures and harder to extend with respect to the distributed memory MIMD model.
- Failures, in a shared memory MIMD affect the entire system, whereas this is not the case of the distributed model, in which each of the PEs can be easily isolated.
- Moreover, shared memory MIMD architectures are less likely to scale because the addition of more PEs leads to memory contention.
- This is a situation that does not happen in the case of distributed memory, in which each PE has its own memory.
- As a result, distributed memory MIMD architectures are most popular today.

APPROACHES TO PARALLEL PROGRAMMING

- A sequential program is one that runs on a single processor and has a single line of control.
- To make many processors collectively work on a single program, the program must be divided into smaller independent chunks so that each processor can work on separate chunks of the problem.
- The program decomposed in this way is a parallel program.
- A wide variety of parallel programming approaches are available.

APPROACHES TO PARALLEL PROGRAMMING

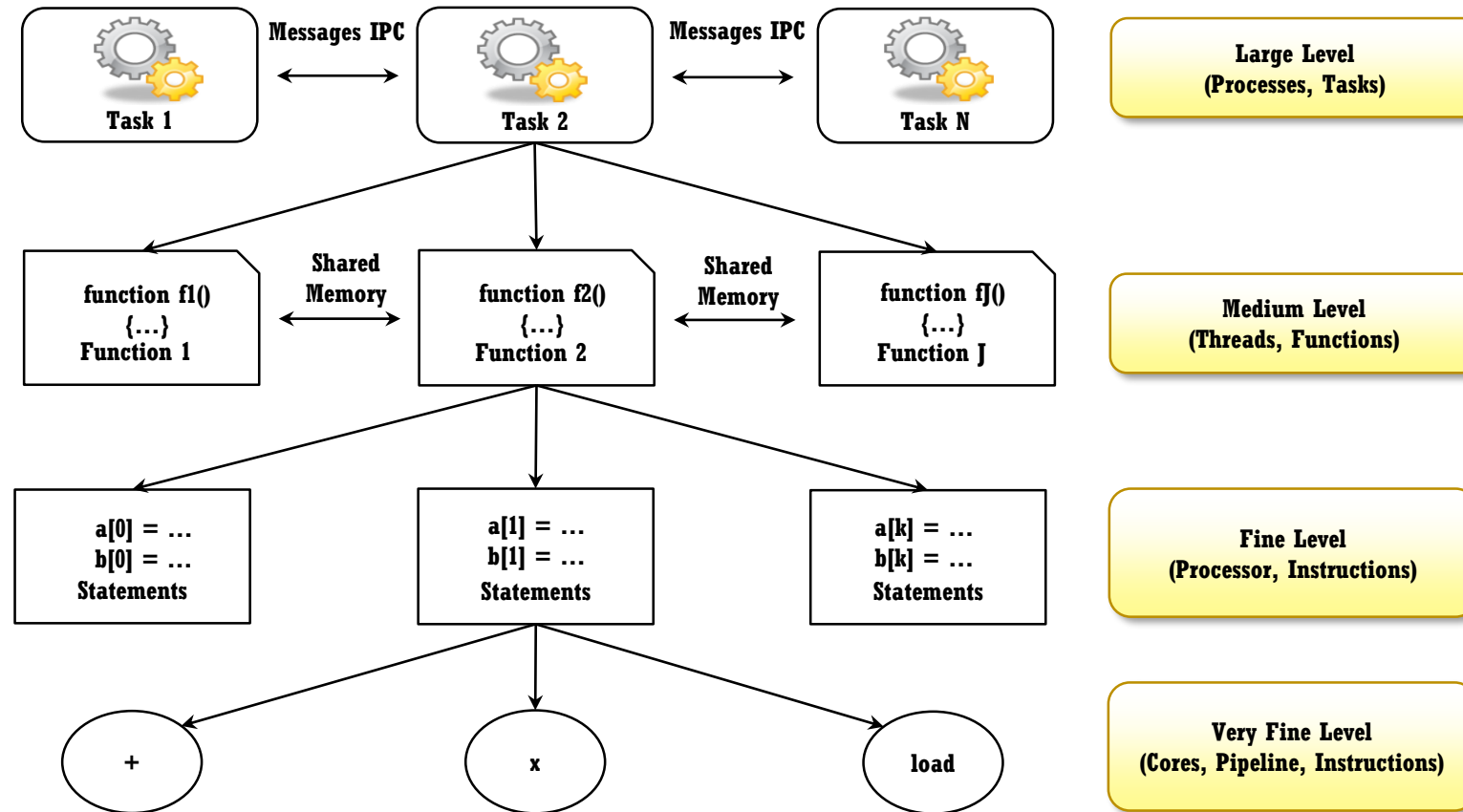
- The most prominent among them are the following.
 - Data Parallelism
 - Process Parallelism
 - Farmer-and-worker model
- The above said three models are suitable for task-level parallelism. In the case of data level parallelism, the divide-and-conquer technique is used to split data into multiple sets, and each data set is processed on different PEs using the same instruction.
- This approach is highly suitable to processing on machines based on the SIMD model.
- In the case of Process Parallelism, a given operation has multiple (but distinct) activities that can be processed on multiple processors.
- In the case of Farmer-and-Worker model, a job distribution approach is used, one processor is configured as master and all other remaining PEs are designated as slaves, the master assigns the jobs to slave PEs and, on completion, they inform the master, which in turn collects results.
- These approaches can be utilized in different levels of parallelism.

LEVELS OF PARALLELISM

- Levels of Parallelism are decided on the size of code (grain size) that can be a potential candidate of parallelism.
- The table shows the levels of parallelism.
- All these approaches have a common goal
 - To boost processor efficiency by hiding latency.
 - To conceal latency, there must be another thread ready to run whenever a lengthy operation occurs.
- The idea is to execute concurrently two or more single-threaded applications. Such as compiling, text formatting, database searching, and device simulation.

Grain Size	Code Item	Parallelized By
Large	Separate and heavy weight process	Programmer
Medium	Function or procedure	Programmer
Fine	Loop or instruction block	Parallelizing compiler
Very Fine	Instruction	Processor

LEVELS OF PARALLELISM



ELEMENTS OF DISTRIBUTED COMPUTING

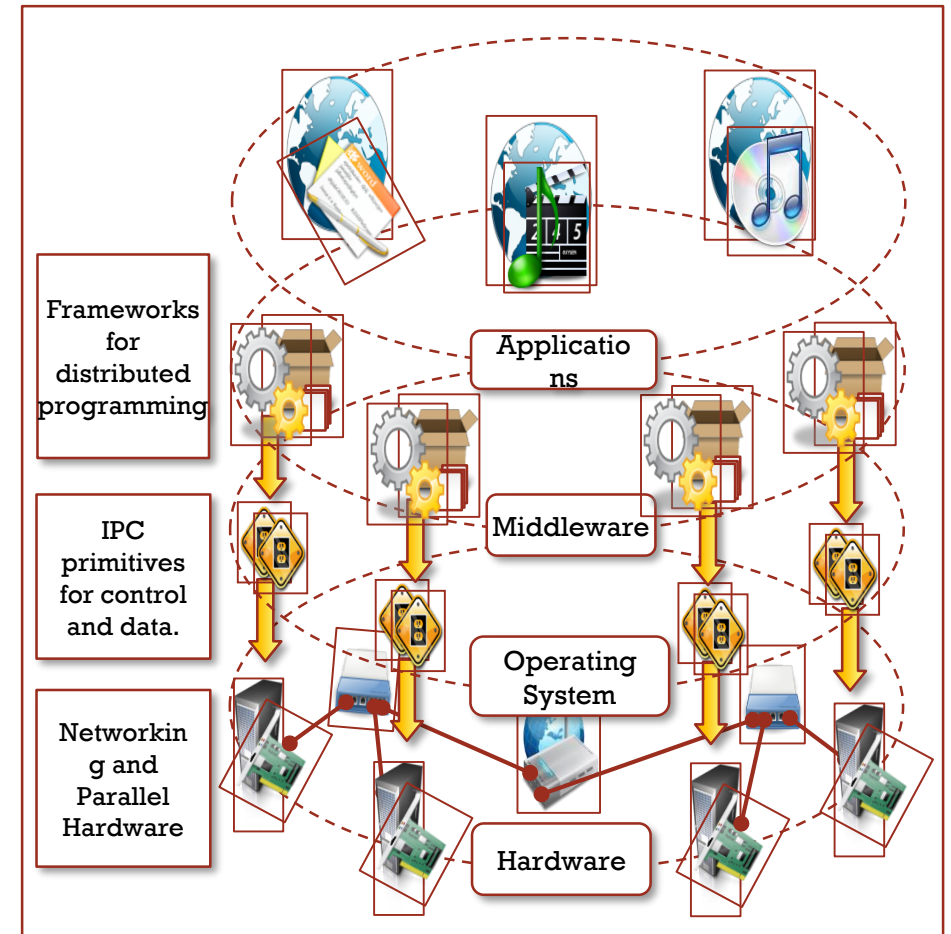
- In the previous section we discussed techniques and architectures that allow introduction of parallelism within a single machine or system and how parallelism operates at different levels of the computing stack.
- Here extend these concepts and explore how multiple activities can be performed by leveraging systems composed of multiple heterogeneous machines and systems.
- We discuss what is generally referred to as distributed computing and more precisely introduce the most common guidelines and patterns for implementing distributed computing systems from the perspective of the software designer.

GENERAL CONCEPTS AND DEFINITIONS

- Distributed computing studies the models, architectures, and algorithms used for building and managing distributed systems.
- As general definition of the term distributed system, we use the one proposed by Tanenbaum
 - A distributed system is a collection of independent computers that appears to its users as a single coherent system.
- This definition is general enough to include various types of distributed computing systems that are especially focused on unified usage and aggregation of distributed resources.
- Here, we focus on the architectural models, that are used to harness independent computers and present them as a whole coherent system.
- Communications is another fundamental aspect of distributed computing. Since distributed systems are composed of more than one computer that collaborate together, it is necessary to provide some sort of data and information exchange between them, which generally occurs through the network.
 - A distributed system is one in which components located at networked computers communicate and coordinate their action only by passing messages.
- As specified in this definition, the components of a distributed system communicate with some sort of message passing. This is a term that encompasses several communication models.

COMPONENTS OF DISTRIBUTED SYSTEM

- A distributed system is the result of the interaction of several components that traverse the entire computing stack from hardware to software.
- It emerges from the collaboration of several elements that- by working together- give users the illusion of a single coherent system.
- The figure provides an overview of the different layers that are involved in providing the services of a distributed system.
- At the very bottom layer, computer and network hardware constitute the physical infrastructure; these components are directly managed by the operating system, which provides the basic services for inter process communication (IPC), process scheduling and management, and resource management in terms of file system and local devices.
- Taken together these two layers become the platform on top of which specialized software is deployed to turn a set of networked computers into a distributed system.



ARCHITECTURAL STYLES FOR DISTRIBUTED COMPUTING

- Although a distributed system comprises the interaction of several layers, the middleware layer is the one that enables distributed computing, because it provides a coherent and uniform runtime environment for applications.
- There are many different ways to organize the components that, taken together, constitute such an environment.
- The interactions among these components and their responsibilities give structure to the middleware and characterize its type or, in other words, define its architecture.
- Architectural styles aid in understanding the classifying the organization of the software systems in general and distributed computing in particular.

ARCHITECTURAL STYLES FOR DISTRIBUTED COMPUTING

- The use of well-known standards at the operating system level and even more at the hardware and network levels allows easy harnessing of heterogeneous components and their organization into a coherent and uniform system.
- For example; network connectivity between different devices is controlled by standards, which allow them into interact seamlessly.
- Design patterns help in creating a common knowledge within the community of software engineers and developers as to how to structure the relevant of components within an application and understand the internal organization of software applications.
- Architectural styles do the same for the overall architecture of software systems.
- The architectural styles are classified into two major classes
 - Software Architectural styles : Relates to the logical organization of the software.
 - System Architectural styles: styles that describe the physical organization of distributed software systems in terms of their major components.

SOFTWARE ARCHITECTURAL STYLES

- Software architectural styles are based on the logical arrangement of software components.
- They are helpful because they provide an intuitive view of the whole system, despite its physical deployment.
- They also identify the main abstractions that are used to shape the components of the system and the expected interaction patterns between them.

Category	Most common Architectural Styles
Data Centered	Repository Blackboard
Data Flow	Pipe and filter Batch Sequential
Virtual Machine	Rule based Interpreter
Call and return	Main program and subroutine call/top-down systems Layered Systems
Independent Components	Communicating Processes Event Systems

TECHNOLOGIES FOR DISTRIBUTED COMPUTING

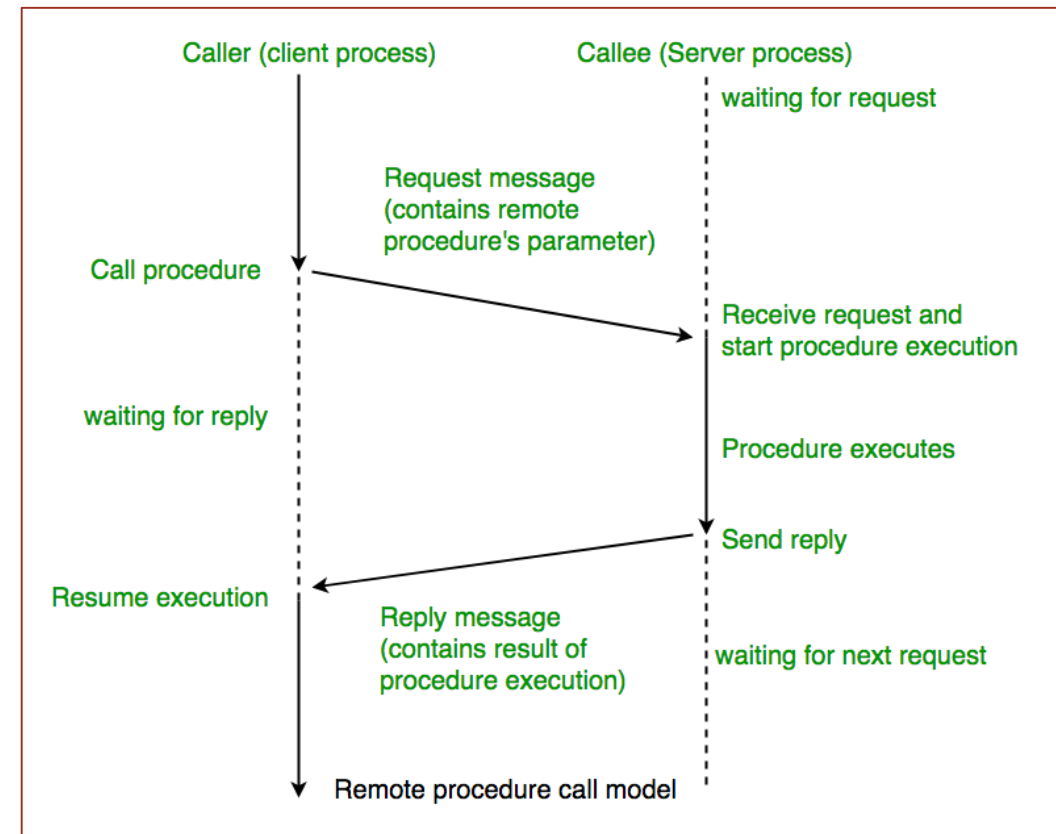
- Remote Procedure Call (RPC)
 - RPC is the fundamental abstraction enabling the execution procedures on clients' request.
 - RPC allows extending the concept of a procedure call beyond the boundaries of a process and a single memory address space.
 - The called procedure and calling procedure may be on the same system or they may be on different systems.
 - The important aspect of RPC is marshalling and unmarshalling.
- Distributed Object Frameworks
 - Extend object-oriented programming systems by allowing objects to be distributed across a heterogeneous network and provide facilities so that they can be coherently act as though they were in the same address space.

REMOTE PROCEDURE CALL (RPC)

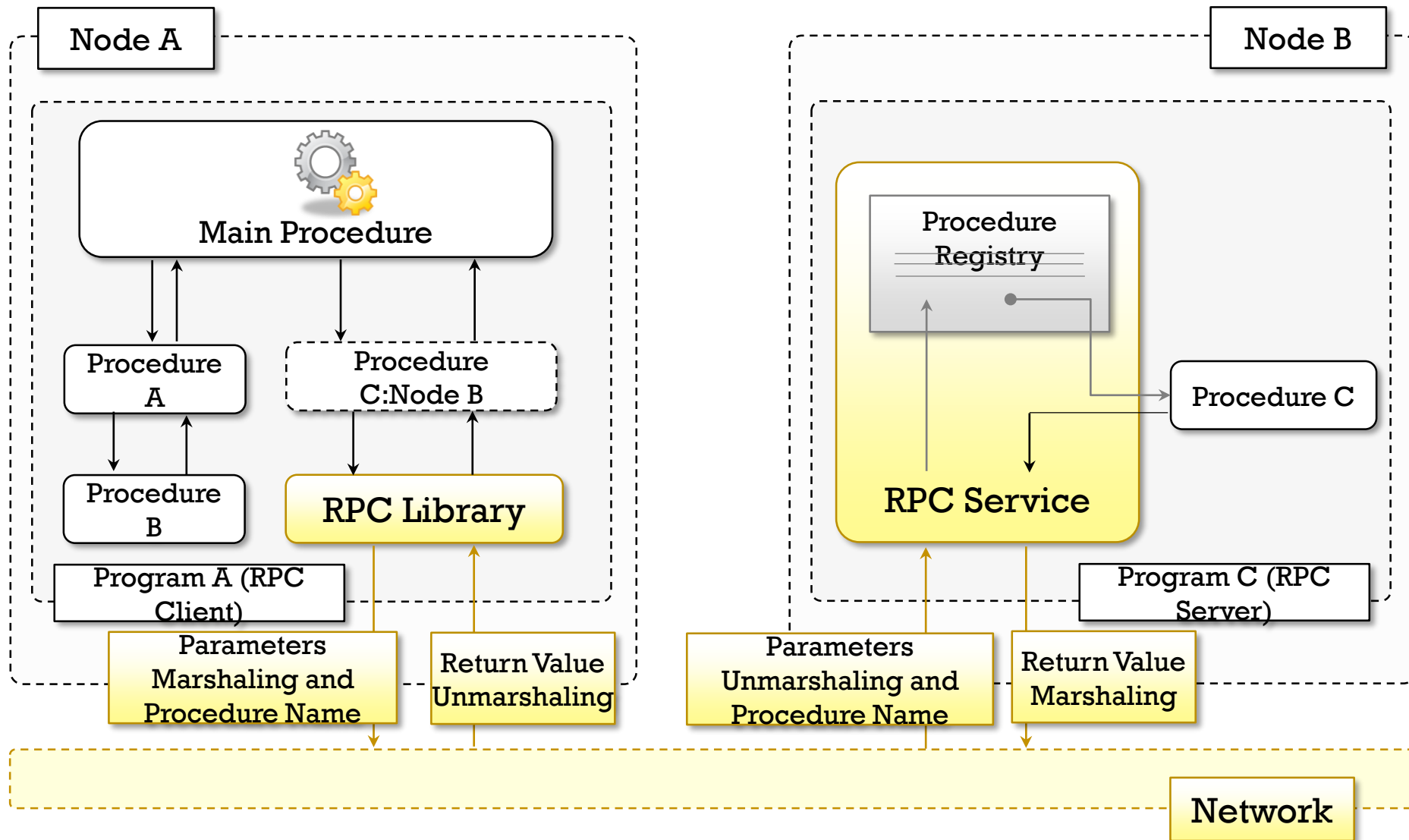
- Remote Procedure Call (RPC) is a technique for constructing distributed, client-server based applications. It is based on extending the conventional local procedure calling so that the called procedure need not exist in the same address space as the calling procedure.
- The two processes may be on the same system, or they may be on different systems with a network connecting them.

REMOTE PROCEDURE CALL (RPC)

- The calling environment is suspended, procedure parameters are transferred across the network to the environment where the procedure is to execute, and the procedure is executed there.
- When the procedure finishes and produces its results, its results are transferred back to the calling environment, where execution resumes as if returning from a regular procedure call.



REMOTE PROCEDURE CALL (RPC)

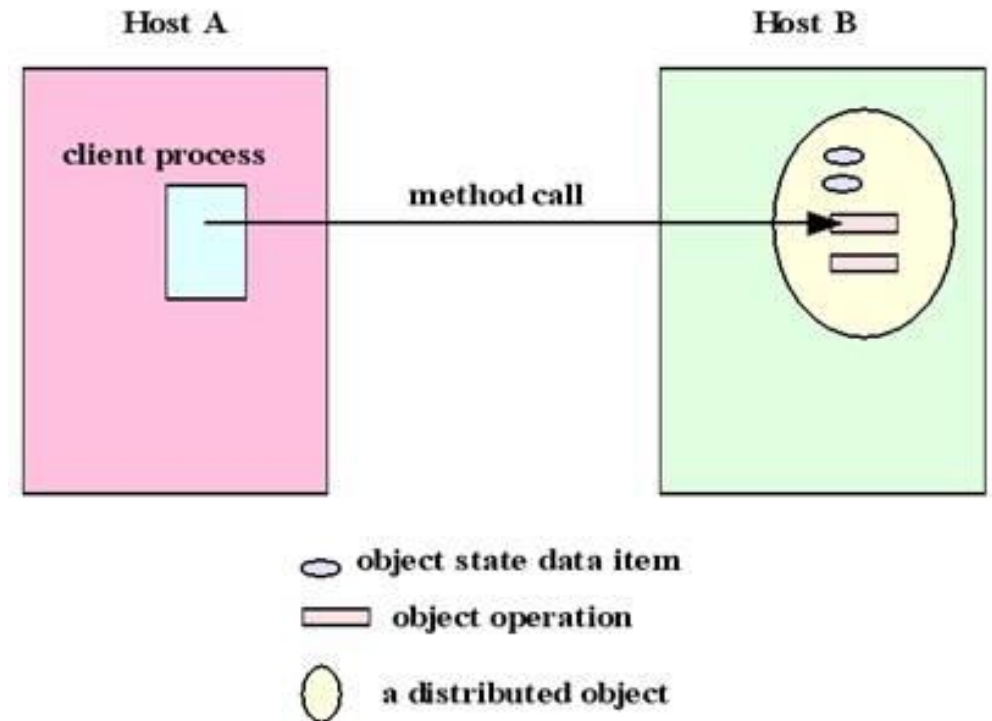


DISTRIBUTED OBJECT

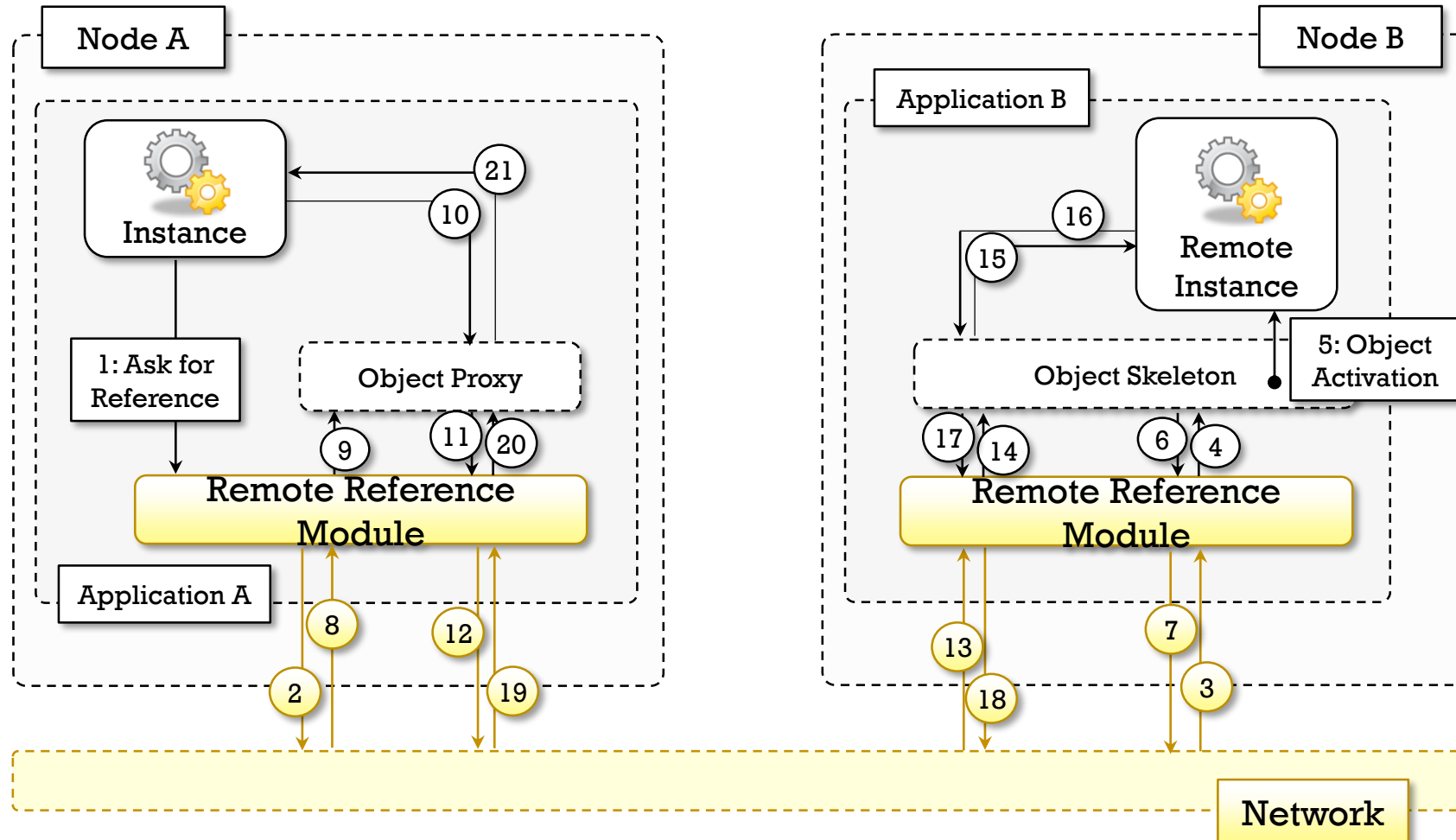
- The distributed object paradigm
 - provides abstractions beyond those of the message-passing model.
 - In object-oriented programming, objects are used to represent an entity significant to an application.
- Each object encapsulates:
 - the state or data of the entity: in Java, such data is contained in the instance variables of each object;
 - the operations of the entity, through which the state of the entity can be accessed or updated.
- Local Objects vs. Distributed Objects
 - Local objects are those whose methods can only be invoked by a local process, a process that runs on the same computer on which the object exists.
 - A distributed object is one whose methods can be invoked by a remote process, a process running on a computer connected via a network to the computer on which the object exists.

THE DISTRIBUTED OBJECT PARADIGM

- In a distributed object paradigm, network resources are represented by distributed objects.
- To request service from a network resource, a process invokes one of its operations or methods, passing data as parameters to the method.
- The method is executed on the remote host, and the response is sent back to the requesting process as a return value.



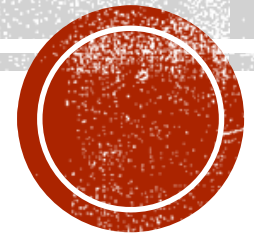
DISTRIBUTED OBJECT PROGRAMMING MODEL



EXAMPLES OF DISTRIBUTED OBJECT FRAMEWORKS

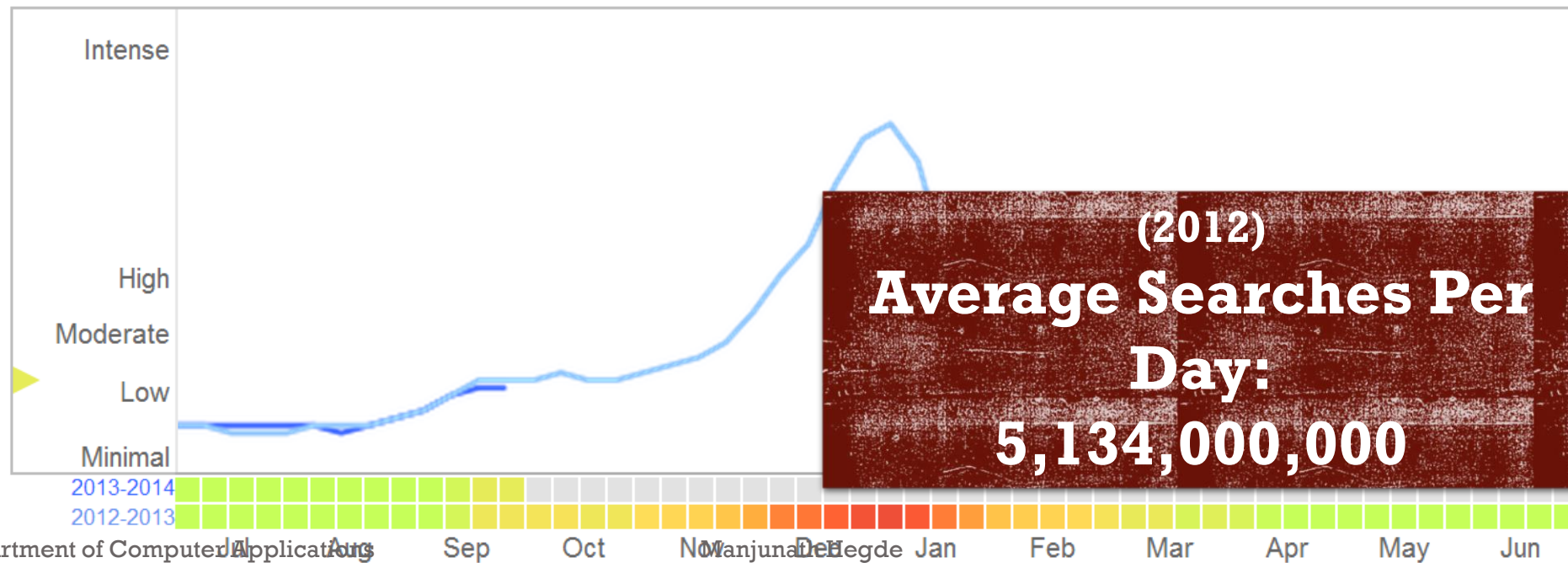
- Common Object Request Broker Architecture (CORBA): cross platform and cross language interoperability among distributed components.
- Distributed Component Object Model (DCOM/COM+) : Microsoft technology for distributed object programming before the introduction of .NET technology.
- Java Remote Method Invocation (RMI): technology provided by Java for enabling RPC among distributed Java objects.
- .NET Remoting: IPC among .NET applications, a uniform platform for accessing remote objects from within any application developed in any of the languages supported by .NET.

MAPREDUCE



[Canada](#) > Ontario

● 2013-2014 ● 2012-2013 ▼

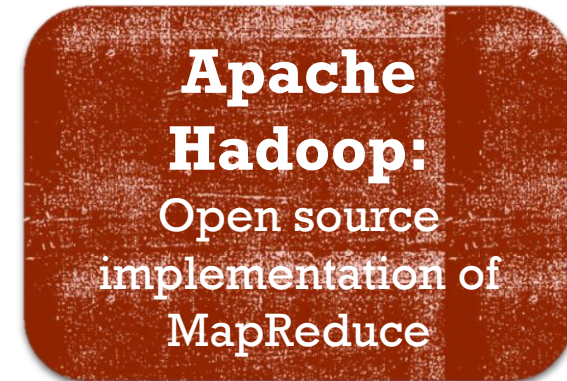


MOTIVATION

- Process lots of data
 - Google processed about **24 petabytes** of data per day in 2009.
- **A single machine** cannot serve all the data
 - You need a distributed system to store and process **in parallel**
- Parallel programming?
 - **Threading** is hard!
 - How do you facilitate **communication** between nodes?
 - How do you **scale to more machines**?
 - How do you handle machine **failures**?

MAPREDUCE

- MapReduce [OSDI'04] provides
 - Automatic parallelization, distribution
 - I/O scheduling
 - Load balancing
 - Network and data transfer optimization
 - Fault tolerance
 - Handling of machine failures
- **Need more power: Scale out, not up!**
 - Large number of commodity servers as opposed to some high end specialized servers



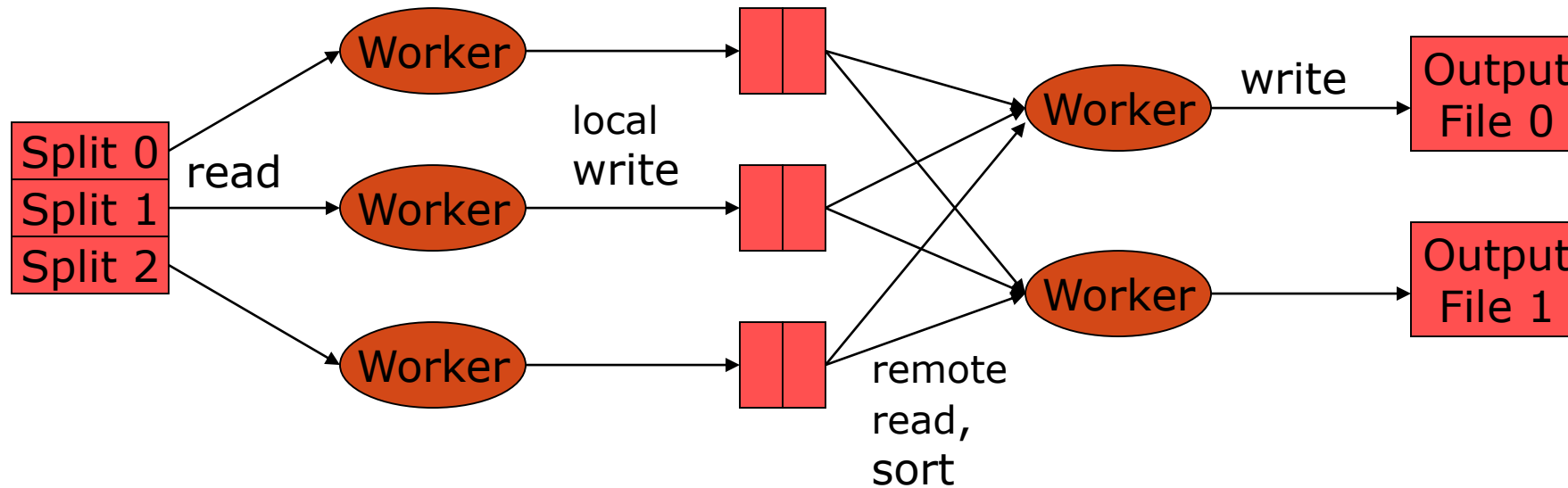
TYPICAL PROBLEM SOLVED BY MAPREDUCE

- Read a lot of data
- **Map**: extract something you care about from each record
- Shuffle and Sort
- **Reduce**: aggregate, summarize, filter, or transform
- Write the results

MAPREDUCE WORKFLOW

Input Data

Output Data



Map

extract something
you care about from
each record

Reduce

aggregate,
summarize,
filter, or
transform

MAPPERS AND REDUCERS

- Need to handle **more data**? Just add **more Mappers/Reducers**!
- No need to handle **multithreaded code** 😊
 - Mappers and Reducers are typically single threaded and **deterministic**
 - **Determinism** allows for **restarting of failed jobs**
 - Mappers/Reducers run **entirely independent** of each other
 - In Hadoop, they run in **separate JVMs**

E

Input Files

Apple Orange Mango
Orange Grapes Plum

Apple Plum Mango
Apple Apple Plum

<http://kickstarthadoop.blogspot.ca/2011/04/word-count-hadoop-map-reduce-example.html>

MAPPER

- Reads in **input pair** <Key, Value>

- Outputs a pair <K', V'>

- Let's count number of each word in user queries (or Tweets/Blogs)

- The input to the mapper will be <queryID, QueryText>:

<Q1, "The teacher went to the store. The store was closed; the store opens in the morning. The store opens at 9am." >

- The output would be:

<The, 1> <teacher, 1> <went, 1> <to, 1> <the, 1> <store, 1> <the, 1> <store, 1> <was, 1> <closed, 1> <the, 1> <store, 1> <opens, 1> <in, 1> <the, 1> <morning, 1> <the 1> <store, 1> <opens, 1> <at, 1> <9am, 1>

REDUCER

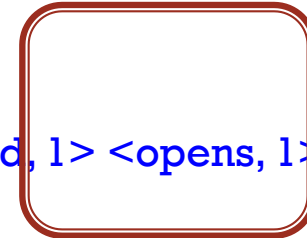
- Accepts the Mapper output, and aggregates values on the key

- For our example, the reducer input would be:

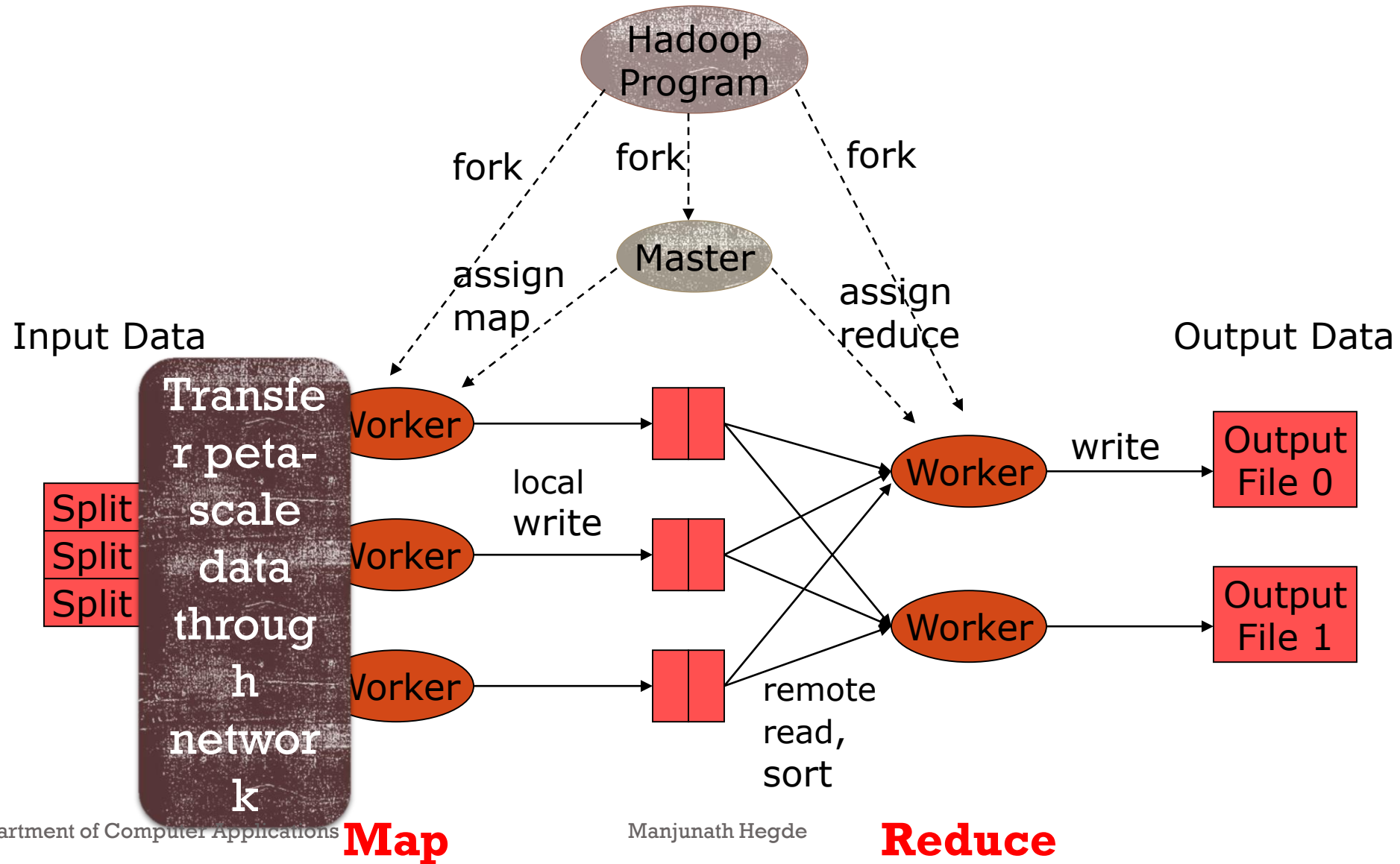
<The, 1> <teacher, 1> <went, 1> <to, 1> <the, 1> <store, 1> <the, 1> <store, 1> <was, 1>
<closed, 1> <the, 1> <store, 1> <opens, 1> <in, 1> <the, 1> <morning, 1> <the 1> <store, 1>
<opens, 1> <at, 1> <9am, 1>

- The output would be:

<The, 6> <teacher, 1> <went, 1> <to, 1> <store, 3> <was, 1> <closed, 1> <opens, 1> <morning, 1> <at, 1> <9am, 1>



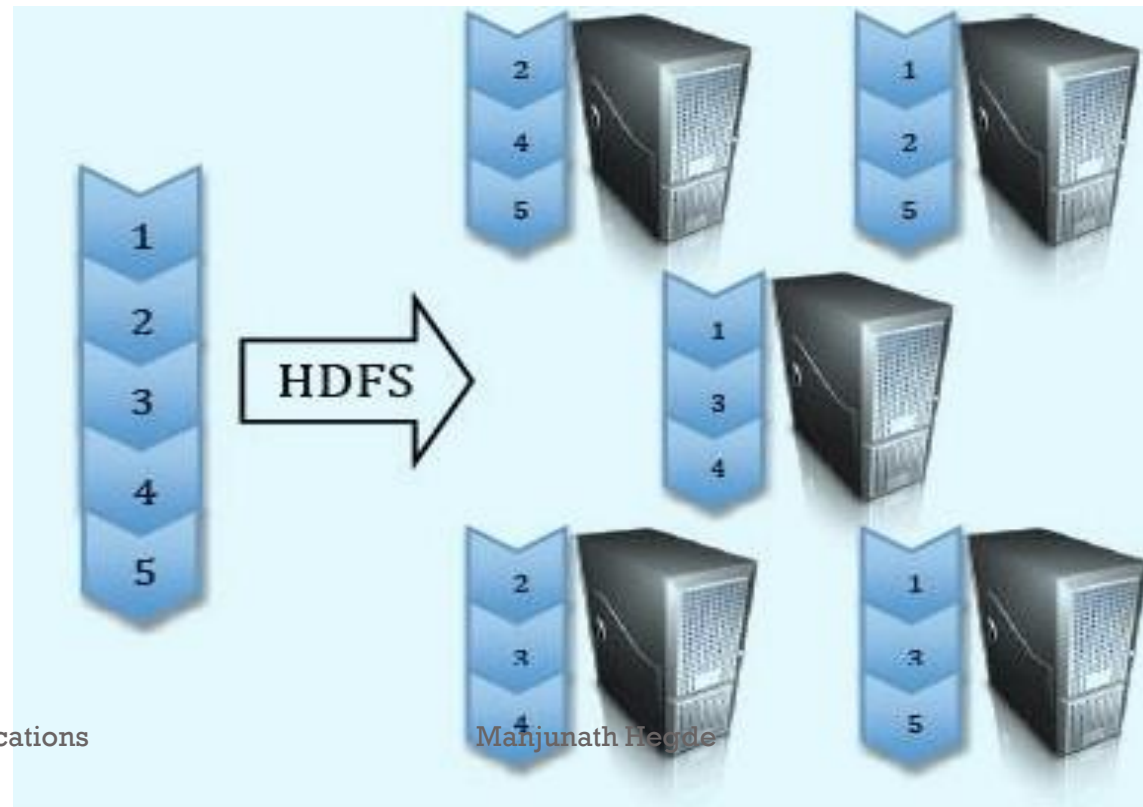
MAPREDUCE



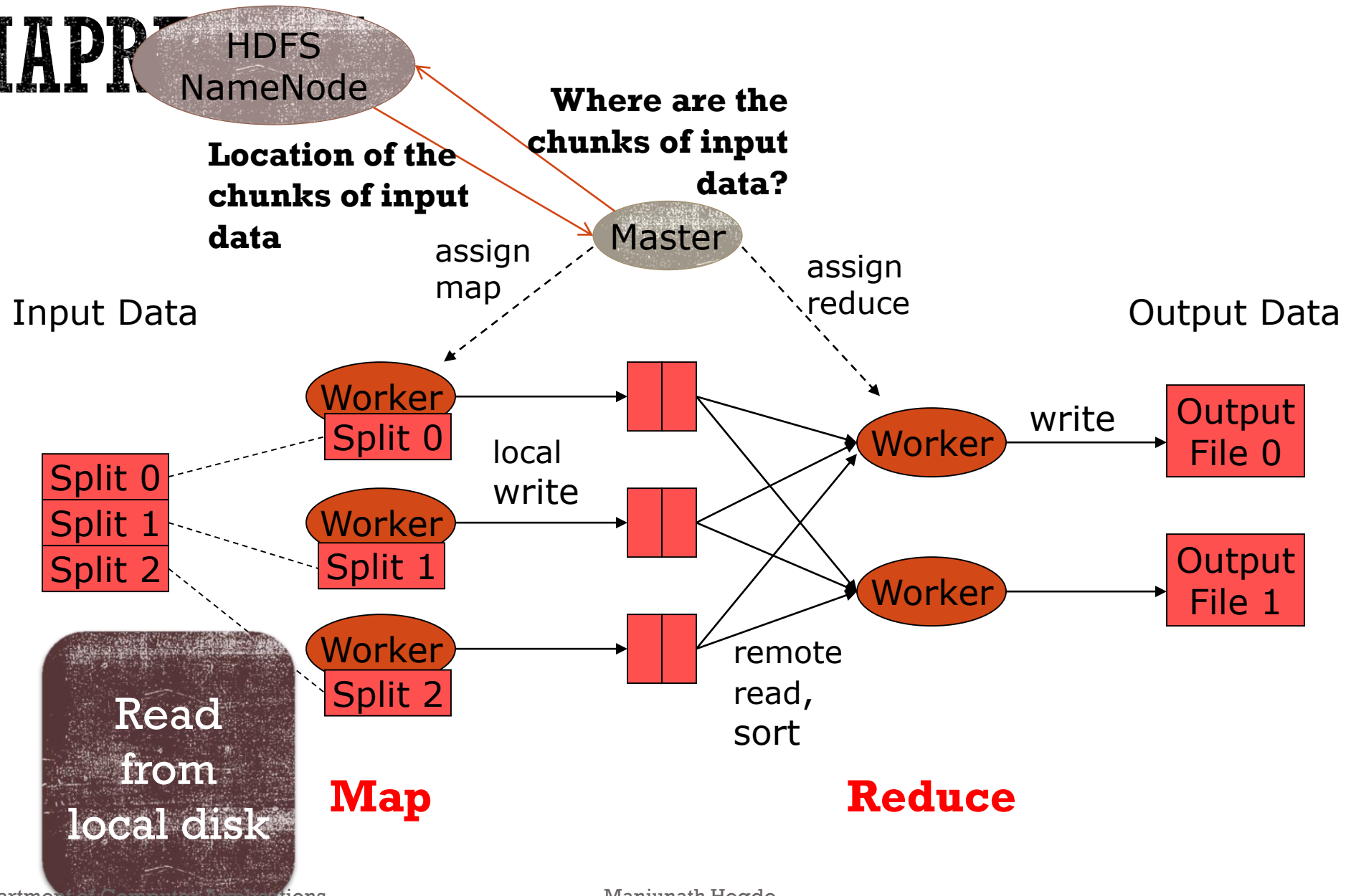
GOOGLE FILE SYSTEM (GFS)

HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

- Split data and store 3 replica on commodity servers



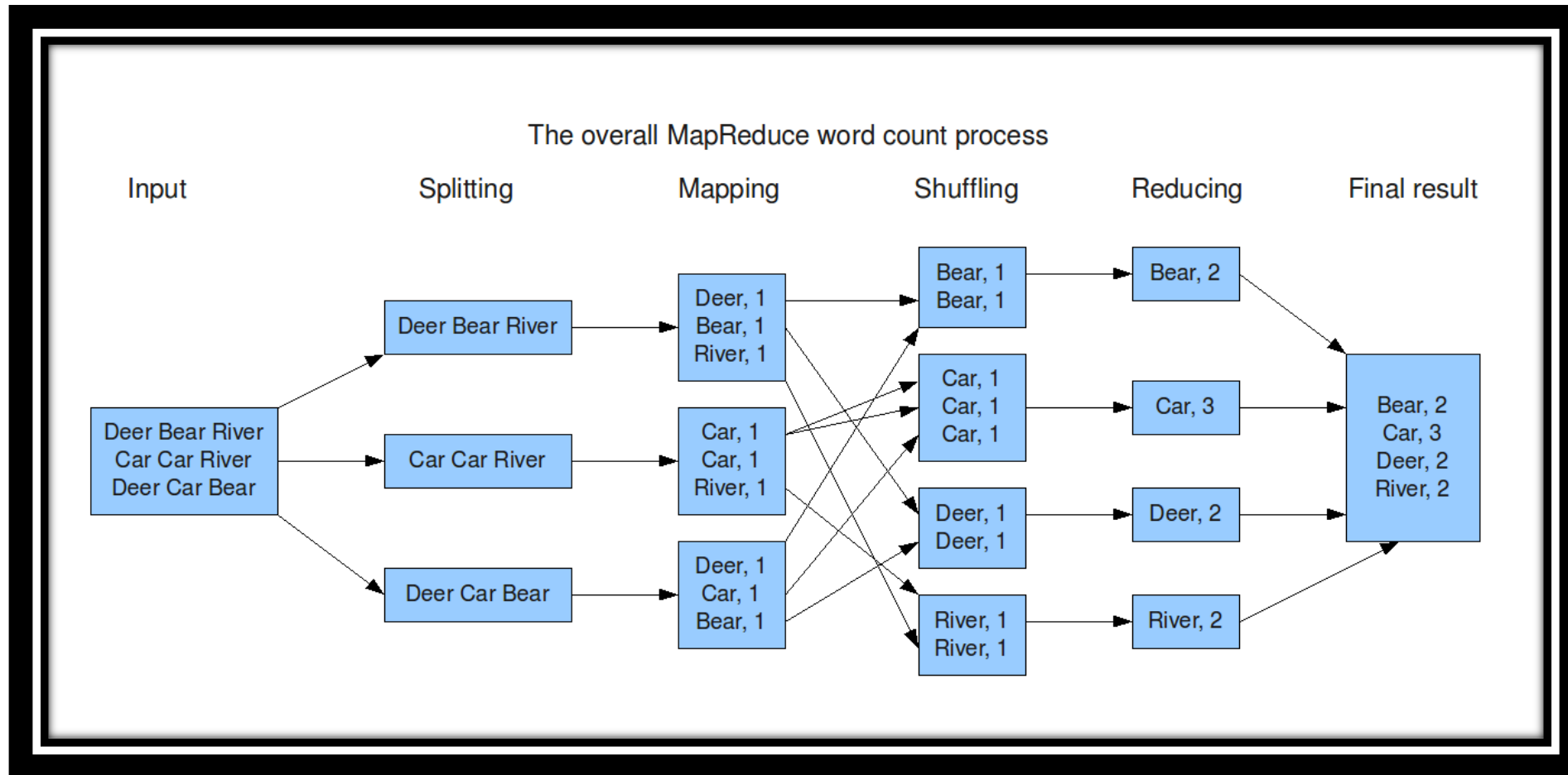
MAPREDUCE



FAILURE IN MAPREDUCE

- **Failures** are **norm** in commodity hardware
- **Worker failure**
 - Detect failure via periodic **heartbeats**
 - **Re-execute** in-progress map/reduce tasks
- **Master failure**
 - Single point of failure; Resume from Execution Log
- **Robust**
 - Google's experience: **lost 1600 of 1800 machines once!**, but **finished fine**

MAPREDUCE: WORD COUNT



```
public class WordCount {
```

```
    public static class Map extends MapReduceBase implements  
        Mapper<LongWritable, Text, Text, IntWritable> {  
        private final static IntWritable one = new IntWritable(1);  
        private Text word = new Text();
```

```
    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable>  
        output, Reporter reporter) throws IOException {  
        String line = value.toString();  
        StringTokenizer tokenizer = new StringTokenizer(line);  
        while (tokenizer.hasMoreTokens()) {  
            word.set(tokenizer.nextToken());  
            output.collect(word, one);  
        }  
    }  
}
```

Mapper

```
    public static class Reduce extends MapReduceBase implements  
        Reducer<Text, IntWritable, Text, IntWritable> {
```

```
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,  
        IntWritable> output, Reporter reporter) throws IOException {  
        int sum = 0;  
        while (values.hasNext()) { sum += values.next().get(); }  
        output.collect(key, new IntWritable(sum));  
    }  
}
```

Reducer

```
    public static void main(String[] args) throws Exception {  
        JobConf conf = new JobConf(WordCount.class);  
        conf.setJobName("wordcount");  
        conf.setOutputKeyClass(Text.class);  
        conf.setOutputValueClass(IntWritable.class);  
        conf.setMapperClass(Map.class);  
        conf.setCombinerClass(Reduce.class);  
        conf.setReducerClass(Reduce.class);  
        conf.setInputFormat(TextInputFormat.class);  
        conf.setOutputFormat(TextOutputFormat.class);  
        FileInputFormat.setInputPaths(conf, new Path(args[0]));  
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));  
  
        JobClient.runJob(conf);  
    }
```

Run this program as
a MapReduce job

SUMMARY

- MapReduce
 - Programming paradigm for data-intensive computing
 - Distributed & parallel execution model
 - Simple to program
 - The framework automates many tedious tasks (machine selection, failure handling, etc.)

CONTENTS

- **Motivation**
- Design overview
 - Write Example
 - Record Append
- Fault Tolerance & Replica Management
- Conclusions

MOTIVATION: LARGE SCALE DATA STORAGE

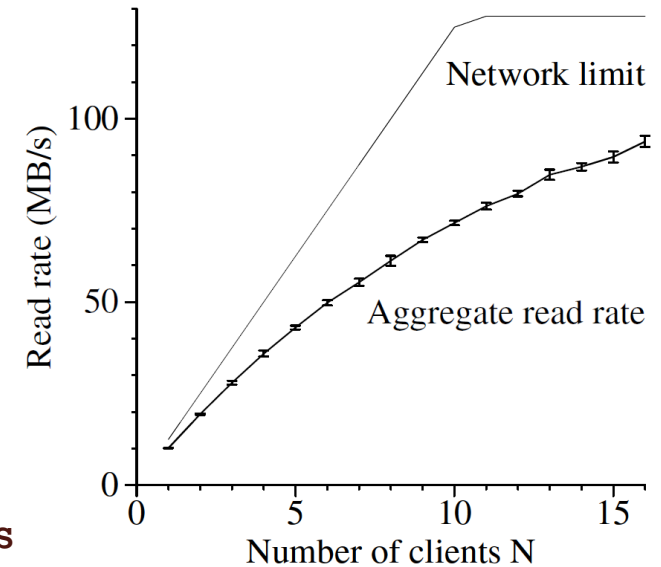
- Manipulate large (**Peta Scale**) sets of data
- Large number of machine with **commodity hardware**
- Component failure is the norm
- Goal: **Scalable, high performance, fault tolerant** distributed file system

WHY A NEW FILE SYSTEM?

- None designed for their failure model
- Few scale as highly or dynamically and easily
- Lack of special primitives for large distributed computation

WHAT SHOULD EXPECT FROM GFS

- Designed for Google's application
 - Control of both file system and application
 - Applications use a few specific access patterns
 - Append to large files
 - Large streaming reads
 - **Not** a good fit for
 - low-latency data access
 - lots of small files, multiple writers, arbitrary file modifications
- Not POSIX*, although mostly traditional
 - Specific operations: RecordAppend

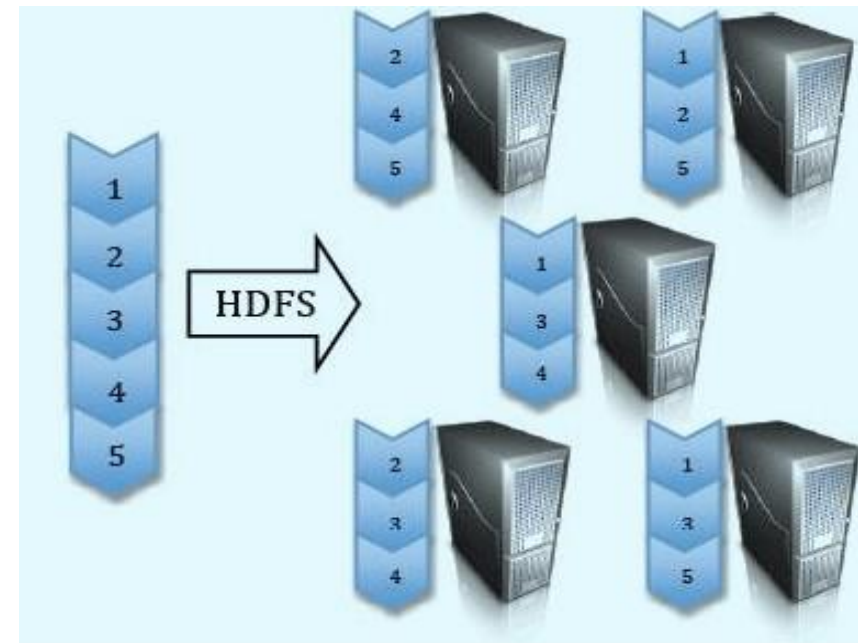


***POSIX** stands for Portable Operating System Interface

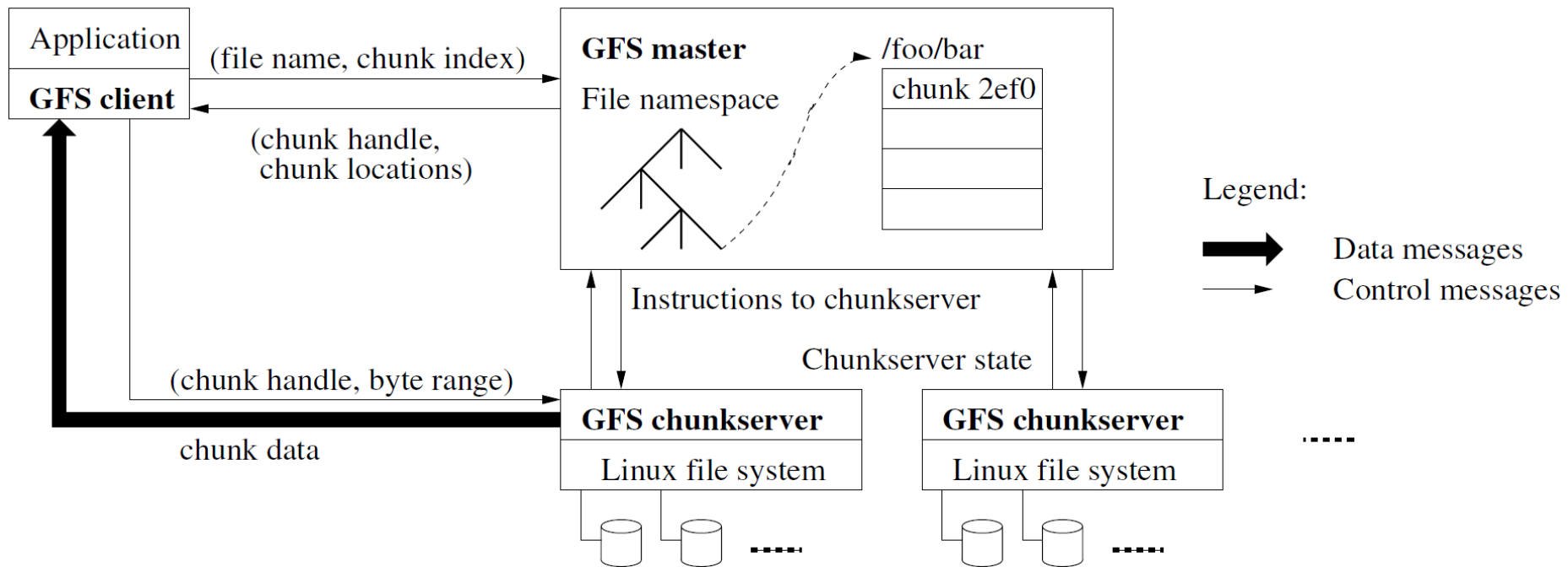
- **Different** characteristic than **transactional** or the “customer order” data : “**write once read many (WORM)**”
 - e.g. web logs, web crawler’s data, or healthcare and patient information
 - WORM inspired MapReduce programming model
- **Google used this characteristics in its Google file system**
- **Apache Hadoop: Open source project**

COMPONENTS

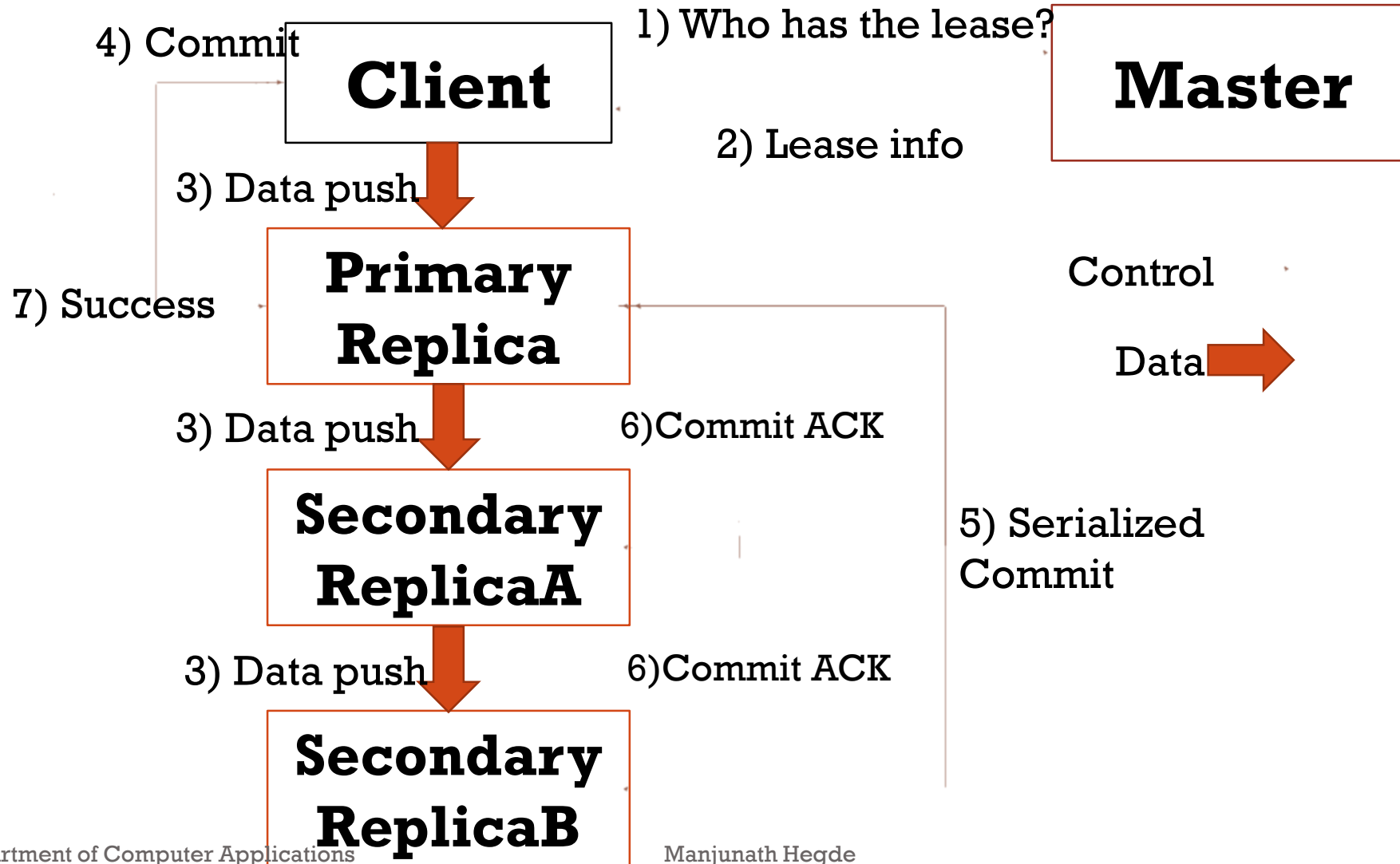
- **Master (NameNode)**
 - Manages metadata (namespace)
 - Not involved in data transfer
 - Controls allocation, placement, replication
- **Chunkserver (DataNode)**
 - Stores chunks of data
 - No knowledge of GFS file system structure
 - Built on local linux file system



GFS ARCHITECTURE



WRITE(FILENAME, OFFSET, DATA)



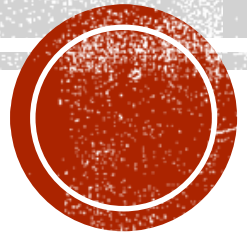
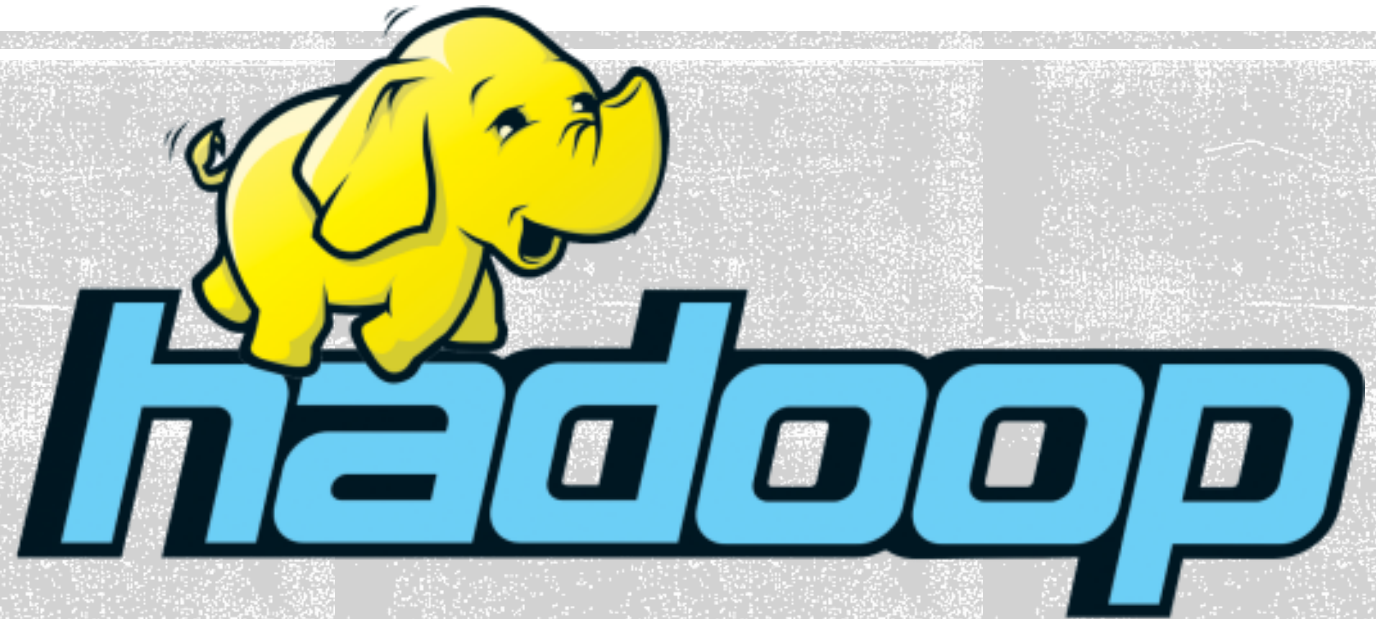
FAULT TOLERANCE

■ Replication

- High availability for reads
- User controllable, default 3 (non-RAID)
- Provides read/seek bandwidth
- Master is responsible for directing re-replication if a data node dies

LIMITATIONS

- Master is a central point of failure
- Master can be a scalability bottleneck
- Latency when opening/stating thousands of files
- Security model is weak



WHAT IS APACHE HADOOP?

- Open source software framework designed for storage and processing of large scale data on clusters of commodity hardware
- Created by Doug Cutting and Mike Carafella in 2005.
- Cutting named the program after his son's toy elephant.

Uses for Hadoop

- Data-intensive text processing
- Assembly of large genomes
- Graph mining
- Machine learning and data mining
- Large scale social network analysis

WHO USES HADOOP?



THE HADOOP ECOSYSTEM

Hadoop Common

- Contains Libraries and other modules

HDFS

- Hadoop Distributed File System

Hadoop YARN

- Yet Another Resource Negotiator

Hadoop MapReduce

- A programming model for large scale data processing



MOTIVATIONS FOR HADOOP

What considerations led to its design

MOTIVATIONS FOR HADOOP

- What were the limitations of earlier large-scale computing?
- What requirements should an alternative approach have?
- How does Hadoop address those requirements?

EARLY LARGE SCALE COMPUTING

- Historically computation was processor-bound
 - Data volume has been relatively small
 - Complicated computations are performed on that data
- Advances in computer technology has historically centered around improving the power of a single machine

DISTRIBUTED SYSTEMS

- Allows developers to use multiple machines for a single task



DISTRIBUTED SYSTEM: PROBLEMS

- Programming on a distributed system is much more complex
 - Synchronizing data exchanges
 - Managing a finite bandwidth
 - Controlling computation timing is complicated
 - Distributed systems must be designed with the expectation of failure

DISTRIBUTED SYSTEM: DATA STORAGE

- Typically divided into Data Nodes and Compute Nodes
- At compute time, data is copied to the Compute Nodes
- Fine for relatively small amounts of data
- Modern systems deal with far more data than was gathering in the past

HOW MUCH DATA?

- Facebook
 - 500 TB per day
- Yahoo
 - Over 170 PB
- eBay
 - Over 6 PB
- Getting the data to the processors becomes the bottleneck

REQUIREMENTS FOR HADOOP

- Must support partial failure
- Must be scalable
- Failure of a single component must not cause the failure of the entire system only a degradation of the application performance
 - ▶ Failure should not result in the loss of any data

RECOVERY

- If a component fails, it should be able to recover without restarting the entire system
- Component failure or recovery during a job must not affect the final output

SCALABILITY

- Increasing resources should increase load capacity
- Increasing the load on the system should result in a graceful decline in performance for all jobs
 - Not system failure

HADOOP

- Based on work done by Google in the early 2000s
 - “The Google File System” in 2003
 - “MapReduce: Simplified Data Processing on Large Clusters” in 2004
- The core idea was to distribute the data as it is initially stored
 - Each node can then perform computation on the data it stores without moving the data for the initial processing

CORE HADOOP CONCEPTS

- Applications are written in a high-level programming language
 - No network programming or temporal dependency
- Nodes should communicate as little as possible
 - A “shared nothing” architecture
- Data is spread among the machines in advance
 - Perform computation where the data is already stored as often as possible

HIGH-LEVEL OVERVIEW

- When data is loaded onto the system it is divided into blocks
 - Typically 64MB or 128MB
- Tasks are divided into two phases
 - Map tasks which are done on small portions of data where the data is stored
 - Reduce tasks which combine data to produce the final output
- A master program allocates work to individual nodes

FAULT TOLERANCE

- Failures are detected by the master program which reassigns the work to a different node
- Restarting a task does not affect the nodes working on other portions of the data
- If a failed node restarts, it is added back to the system and assigned new tasks
- The master can redundantly execute the same task to avoid slow running nodes

HADOOP DISTRIBUTED FILE SYSTEM

78

HDFS

OVERVIEW

- Responsible for storing data on the cluster
- Data files are split into blocks and distributed across the nodes in the cluster
- Each block is replicated multiple times

HDFS BASIC CONCEPTS

- HDFS is a file system written in Java based on the Google's GFS
- Provides redundant storage for massive amounts of data

HDFS BASIC CONCEPTS

- HDFS works best with a smaller number of large files
 - Millions as opposed to billions of files
 - Typically 100MB or more per file
- Files in HDFS are write once
- Optimized for streaming reads of large files and not random reads

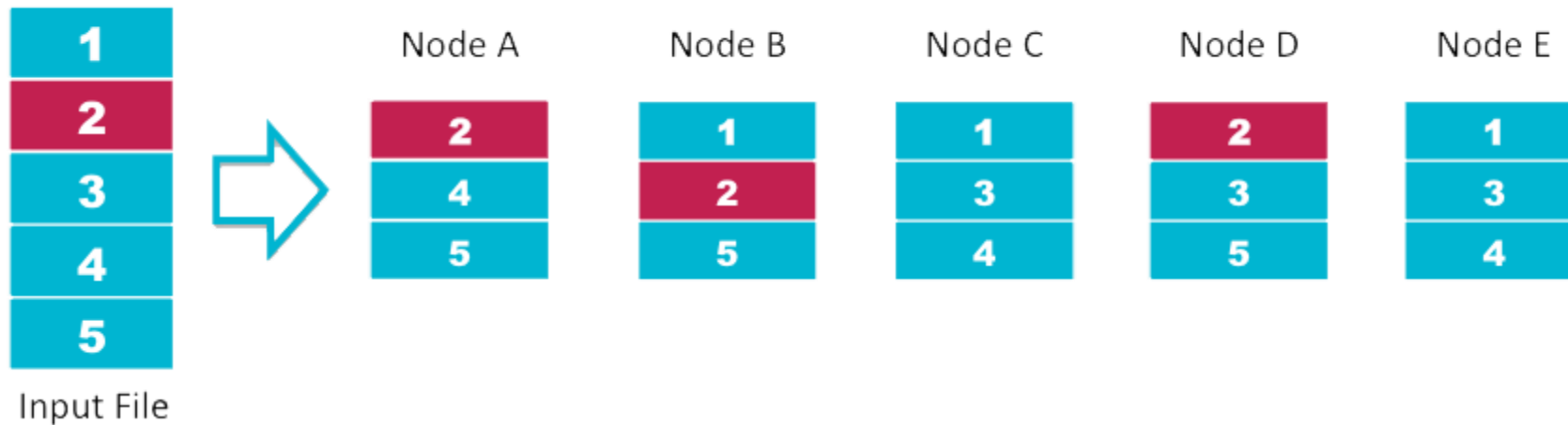
HOW ARE FILES STORED

- Files are split into blocks
- Blocks are split across many machines at load time
 - Different blocks from the same file will be stored on different machines
- Blocks are replicated across multiple machines
- The NameNode keeps track of which blocks make up a file and where they are stored

DATA REPLICATION

- Default replication is 3-fold

HDFS Data Distribution



DATA RETRIEVAL

- When a client wants to retrieve data
 - Communicates with the NameNode to determine which blocks make up a file and on which data nodes those blocks are stored
 - Then communicated directly with the data nodes to read the data

85

ANATOMY OF A CLUSTER

What parts actually make up a Hadoop cluster

OVERVIEW

- **NameNode**
 - Holds the metadata for the HDFS
- **Secondary NameNode**
 - Performs housekeeping functions for the NameNode
- **DataNode**
 - Stores the actual HDFS data blocks
- **JobTracker**
 - Manages MapReduce jobs
- **TaskTracker**
 - Monitors individual Map and Reduce tasks

THE NAMENODE

- Stores the HDFS file system information in a fsimage
- Updates to the file system (add/remove blocks) do not change the fsimage file
 - They are instead written to a log file
- When starting the NameNode loads the fsimage file and then applies the changes in the log file

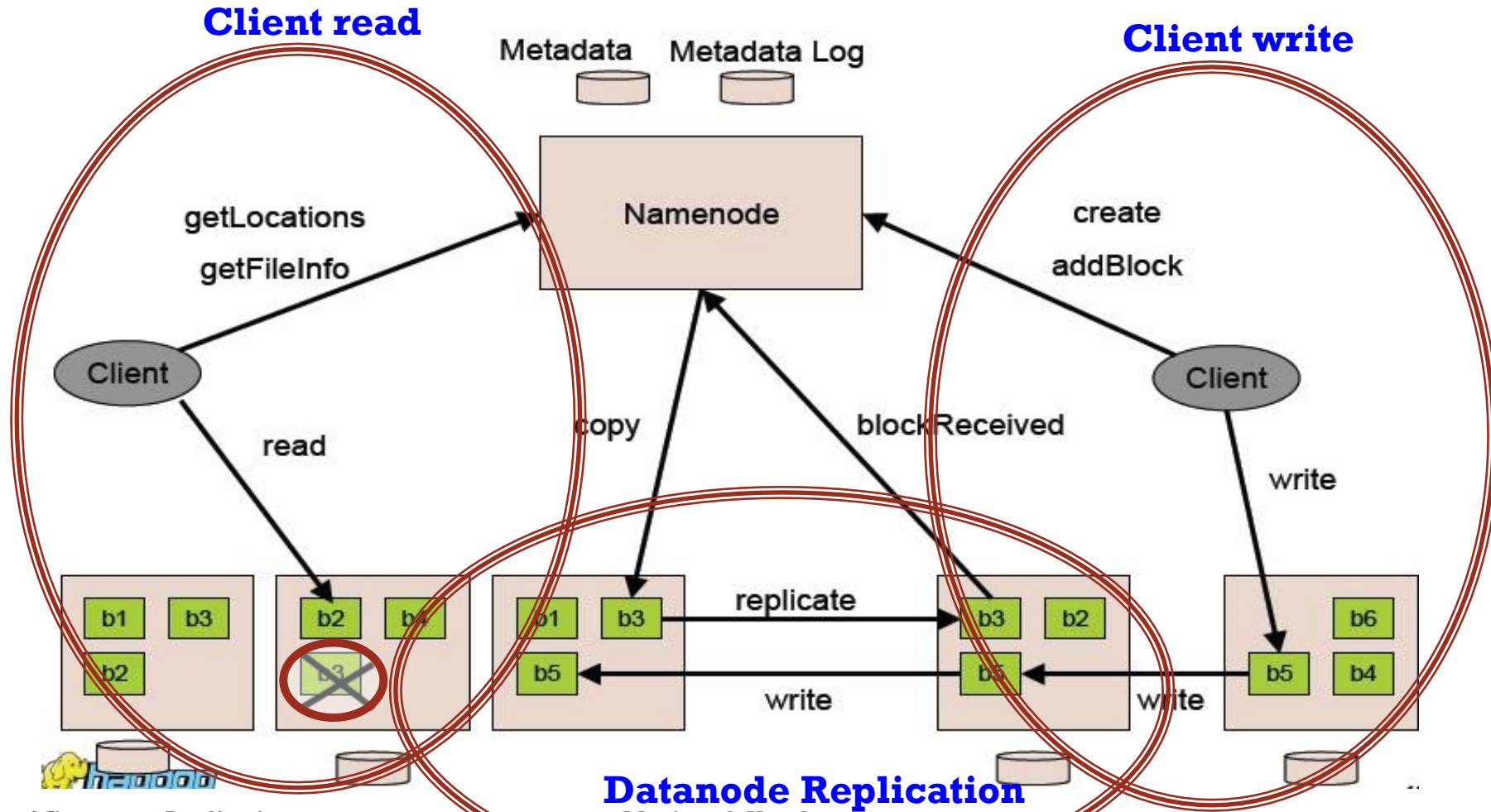
THE SECONDARY NAMENODE

- **NOT** a backup for the NameNode
- Periodically reads the log file and applies the changes to the fsimage file bringing it up to date
- Allows the NameNode to restart faster when required

JOBTRACKER AND TASKTRACKER

- JobTracker
 - Determines the execution plan for the job
 - Assigns individual tasks
- TaskTracker
 - Keeps track of the performance of an individual mapper or reducer

HDFS ARCHITECTURE





HADOOP ECOSYSTEM

Other available tools

WHY DO THESE TOOLS EXIST?

- MapReduce is very powerful, but can be awkward to master
- These tools allow programmers who are familiar with other programming styles to take advantage of the power of MapReduce

OTHER TOOLS

- Hive
 - Hadoop processing with SQL
- Pig
 - Hadoop processing with scripting
- Cascading
 - Pipe and Filter processing model
- HBase
 - Database model built on top of Hadoop
- Flume
 - Designed for large scale data movement

MAPREDUCE — INSTALLATION & IMPLEMENTATION

94

- MapReduce works only on Linux flavored operating systems and it comes inbuilt with a Hadoop Framework.
- Hadoop needs to be installed on VirtualBox if installing Hadoop on any OS other than Linux.
- We need to perform the following steps in order to install Hadoop framework.

JAVA INSTALLATION

- Java must be installed on your system before installing Hadoop. Use the following command to check whether you have Java installed on your system.

\$ java -version

- If Java is already installed on your system, you get to see the following response

java version "1.8.0_161"

Java(TM) SE Runtime Environment (build 1.8.0_161-b12)

Java HotSpot(TM) 64-Bit Server VM (build 25.161-b12, mixed mode)

- Download the latest version of Java
- After downloading, you can locate the file *jdk-8u<latest-version>-linux-x64.tar.gz* in your Downloads folder.

- Use the following commands to extract the contents of `jdk-8u<latest-version>-linux-x64.tar.gz`.

```
$ cd Downloads/
```

```
$ ls
```

```
jdk-8u<latest-version>-linux-x64.tar.gz
```

```
$ tar xzf jdk-8u<latest-version>-linux-x64.tar.gz
```

```
$ ls
```

```
jdk-8u<latest-version>-linux-x64.tar.gz
```

- To make Java available to all the users, you have to move it to the location “/usr/local/”. Go to root and type the following commands –

```
$ su
```

```
password:
```

```
# mv jdk1.7.0_71 /usr/local/java
```

```
# exit
```

- For setting up PATH and JAVA_HOME variables, add the following commands to ~/.bashrc file.

```
export JAVA_HOME=/usr/local/java  
export PATH=$PATH:$JAVA_HOME/bin
```

- Apply all the changes to the current running system.

```
$ source ~/.bashrc
```

- Use the following commands to configure Java alternatives –

- *# alternatives --install /usr/bin/java java usr/local/java/bin/java 2*
- *# alternatives --install /usr/bin/javac javac usr/local/java/bin/javac 2*
- *# alternatives --install /usr/bin/jar jar usr/local/java/bin/jar 2*
- *# alternatives --set java usr/local/java/bin/java*
- *# alternatives --set javac usr/local/java/bin/javac*
- *# alternatives --set jar usr/local/java/bin/jar*

- Now verify the installation using the command `java -version` from the terminal.

VERIFYING HADOOP INSTALLATION

- Hadoop must be installed on your system before installing MapReduce. Let us verify the Hadoop installation using the following command –

\$ hadoop version

- If Hadoop is already installed on your system, then you will get the following response –

Hadoop 2.4.1

--

Subversion <https://svn.apache.org/repos/asf/hadoop/common> -r 1529768

Compiled by hortonmu on 2013-10-07T06:28Z

Compiled with protoc 2.5.0

From source with checksum 79e53ce7994d1628b240f09af91e1af4

- If Hadoop is not installed on your system, then proceed with the following steps.

DOWNLOADING HADOOP

- Download Hadoop 2.4.1 from Apache Software Foundation and extract its contents using the following commands.

```
$ su
```

```
password:
```

```
# cd /usr/local
```

```
# wget http://apache.claz.org/hadoop/common/hadoop-2.4.1/
```

```
hadoop-2.4.1.tar.gz
```

```
# tar xzf hadoop-2.4.1.tar.gz
```

```
# mv hadoop-2.4.1/* to hadoop/
```

```
# exit
```

INSTALLING HADOOP IN PSEUDO DISTRIBUTED MODE

- The pseudo-distributed mode is also known as a single-node cluster where both NameNode and DataNode will reside on the same machine.
- Setting up Hadoop
- You can set Hadoop environment variables by appending the following commands to ~/.bashrc file.

```
export HADOOP_HOME=/usr/local/hadoop  
export HADOOP_MAPRED_HOME=$HADOOP_HOME  
export HADOOP_COMMON_HOME=$HADOOP_HOME  
export HADOOP_HDFS_HOME=$HADOOP_HOME  
export YARN_HOME=$HADOOP_HOME  
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native  
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
```

- Apply all the changes to the current running system.
 - *\$ source ~/.bashrc*

HADOOP CONFIGURATION

- You can find all the Hadoop configuration files in the location “\$HADOOP_HOME/etc/hadoop”. You need to make suitable changes in those configuration files according to your Hadoop infrastructure.
\$ cd \$HADOOP_HOME/etc/hadoop
- In order to develop Hadoop programs using Java, you have to reset the Java environment variables in hadoop-env.sh file by replacing JAVA_HOME value with the location of Java in your system.
export JAVA_HOME=/usr/local/java
- You have to edit the following files to configure Hadoop –
 - core-site.xml
 - hdfs-site.xml
 - yarn-site.xml
 - mapred-site.xml

CORE-SITE.XML

- core-site.xml contains the following information–
 - Port number used for Hadoop instance
 - Memory allocated for the file system
 - Memory limit for storing the data
 - Size of Read/Write buffers
- Open the core-site.xml and add the following properties in between the `<configuration>` and `</configuration>` tags.

```
<configuration>
```

```
  <property>
```

```
    <name>fs.default.name</name>
```

```
    <value>hdfs://localhost:9000 </value>
```

```
  </property>
```

```
</configuration>
```

HDFS-SITE.XML

- hdfs-site.xml contains the following information –
 - Value of replication data
 - The namenode path
 - The datanode path of your local file systems (the place where you want to store the Hadoop infra)
- Open this file and add the following properties in between the <configuration>, </configuration> tags.

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.name.dir</name>

    <value>file:///home/hadoop/hadoopinfra/hdfs/name
node</value>
  </property>
  <property>
    <name>dfs.data.dir</name>

    <value>file:///home/hadoop/hadoopinfra/hdfs/datan
ode </value>
  </property>
</configuration>
```


YARN-SITE.XML

- This file is used to configure yarn into Hadoop. Open the yarn-site.xml file and add the following properties in between the <configuration>, </configuration> tags.

```
<configuration>  
  <property>  
    <name>yarn.nodemanager.aux-services</name>  
    <value>mapreduce_shuffle</value>  
  </property>  
</configuration>
```

MAPRED-SITE.XML

- This file is used to specify the MapReduce framework we are using. By default, Hadoop contains a template of yarn-site.xml. First of all, you need to copy the file from mapred-site.xml.template to mapred-site.xml file using the following command.

```
$ cp mapred-site.xml.template mapred-site.xml
```

- Open mapred-site.xml file and add the following properties in between the <configuration>, </configuration> tags.

```
<configuration>
```

```
<property>
```

```
<name>mapreduce.framework.name</name>
```

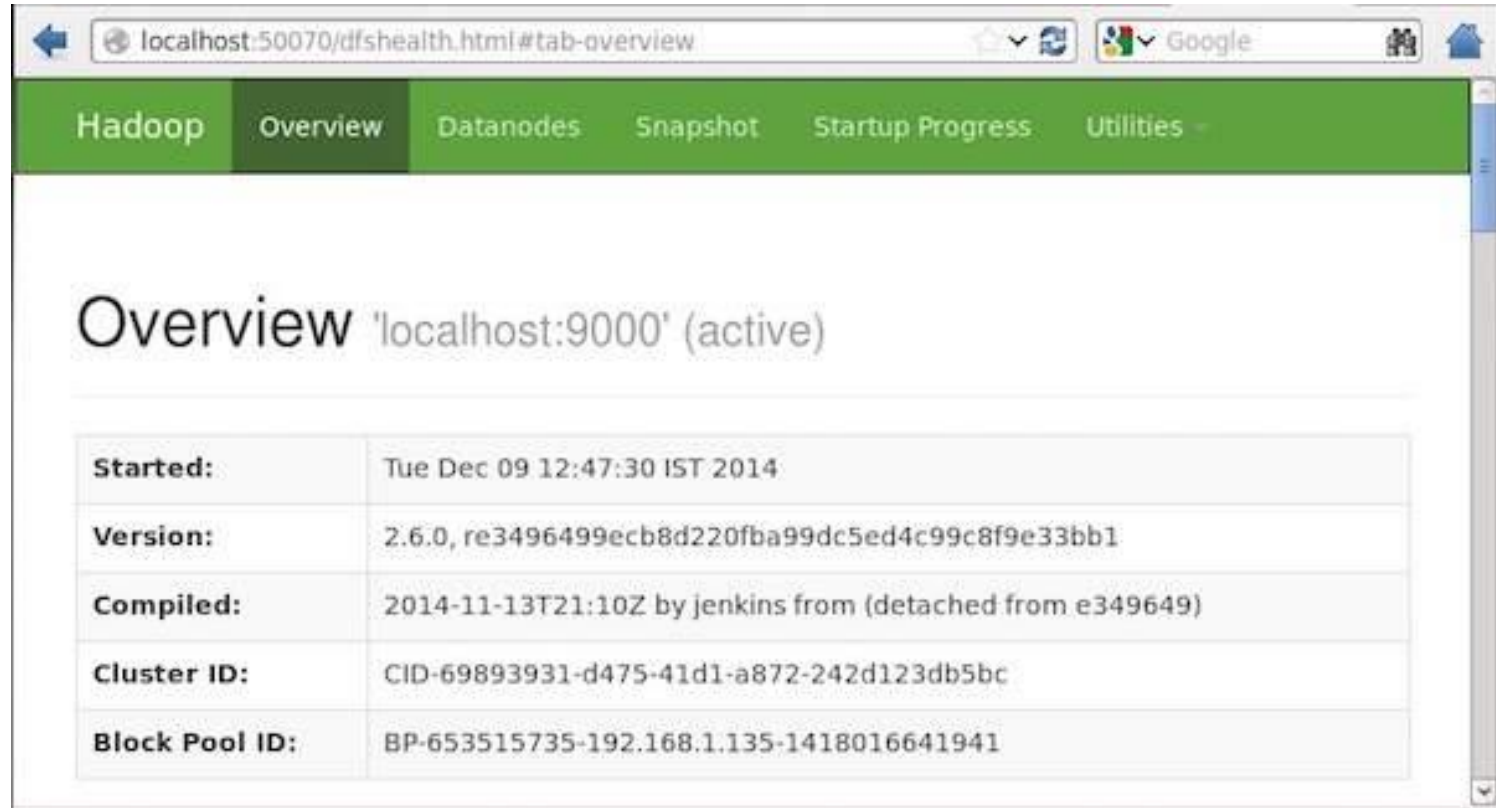
```
<value>yarn</value>
```

```
</property>
```

```
</configuration>
```

ACCESSING HADOOP ON BROWSER

- The default port number to access Hadoop is 50070. Use the following URL to get Hadoop services on your browser.
 - <http://localhost:50070/>



The screenshot shows a web browser window with the address bar displaying `localhost:50070/dfshealth.html#tab-overview`. The page has a green navigation bar with tabs: **Hadoop**, **Overview**, **Datanodes**, **Snapshot**, **Startup Progress**, and **Utilities**. The main content area is titled "Overview 'localhost:9000' (active)". Below the title is a table with the following information:

Started:	Tue Dec 09 12:47:30 IST 2014
Version:	2.6.0, re3496499ecb8d220fba99dc5ed4c99c8f9e33bb1
Compiled:	2014-11-13T21:10Z by jenkins from (detached from e349649)
Cluster ID:	CID-69893931-d475-41d1-a872-242d123db5bc
Block Pool ID:	BP-653515735-192.168.1.135-1418016641941

VERIFY ALL APPLICATIONS OF A CLUSTER

- The default port number to access all the applications of a cluster is 8088. Use the following URL to use this service.
 - <http://localhost:8088/>



The screenshot shows the Hadoop web interface at localhost:8088. The title is "All Applications". On the left, there is a sidebar with the Hadoop logo and a "Cluster" section containing a list of applications: "NEW", "SUBMITTED", "KILLED", "FAILED", "RUNNING", "FINISHED", "KILLED", and "SUCCEEDED". The main content area shows "Cluster Metrics" with a table of application statistics. Below this, there is a table of applications with columns for ID, User, Name, Application Type, Queue, StartTime, FirstTime, State, and Progress. The table is currently empty, showing "Showing 0 to 0 of 0 entries".

App Submitted	App Pending	App Running	App Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
0	0	0	0	0	0 B	8 GB	0 B	0	0	0	1	0	0	0	0

ID	User	Name	Application Type	Queue	StartTime	FirstTime	State	Progress	Tracking Info
No data available in table									

Showing 0 to 0 of 0 entries

MAPREDUCE - API

- The classes and their methods that are involved in the operations of MapReduce programming. Primarily focus on the following –
 - JobContext Interface
 - Job Class
 - Mapper Class
 - Reducer Class

JOBCONTEXT INTERFACE

- The JobContext interface is the super interface for all the classes, which defines different jobs in MapReduce.
- It gives a read-only view of the job that is provided to the tasks while they are running.
- The following are the sub-interfaces of JobContext interface.
- `MapContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT>`
 - Defines the context that is given to the Mapper.
- `ReduceContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT>`
 - Defines the context that is passed to the Reducer.

JOB CLASS

- The Job class is the most important class in the MapReduce API.
- It allows the user to configure the job, submit it, control its execution, and query the state.
- The set methods only work until the job is submitted, afterwards they will throw an `IllegalStateException`.
- Normally, the user creates the application, describes the various facets of the job, and then submits the job and monitors its progress.

EXAMPLE OF HOW TO SUBMIT A JOB —

```
// Create a new Job
Job job = new Job(new Configuration());
job.setJarByClass(MyJob.class);

// Specify various job-specific parameters
job.setJobName("myjob");
job.setInputPath(new Path("in"));
job.setOutputPath(new Path("out"));
job.setMapperClass(MyJob.MyMapper.class);
job.setReducerClass(MyJob.MyReducer.class);

// Submit the job, then poll for progress until the job is complete
job.waitForCompletion(true);
```


CONSTRUCTORS

Following are the constructor summary of Job class.

S.No	Constructor Summary
1	Job()
2	Job(Configuration conf)
3	Job(Configuration conf, String jobName)

METHODS

Some of the important methods of Job class are as follows

S.No	Method Description
1	getJobName() User-specified job name.
2	getJobState() Returns the current state of the Job.
3	isComplete() Checks if the job is finished or not.
4	setInputFormatClass() Sets the InputFormat for the job.
5	setJobName(String name) Sets the user-specified job name.
6	setOutputFormatClass() Sets the Output Format for the job.
7	setMapperClass(Class) Sets the Mapper for the job.
8	setReducerClass(Class) Sets the Reducer for the job.
9	setPartitionerClass(Class) Sets the Partitioner for the job.
10	setCombinerClass(Class) Sets the Combiner for the job.

MAPPER CLASS

- The Mapper class defines the Map job. Maps input key-value pairs to a set of intermediate key-value pairs.
- Maps are the individual tasks that transform the input records into intermediate records.
- The transformed intermediate records need not be of the same type as the input records.
- A given input pair may map to zero or many output pairs.
- Method
 - map is the most prominent method of the Mapper class. The syntax is defined below –
 - `map(KEYIN key, VALUEIN value, org.apache.hadoop.mapreduce.Mapper.Context context)`
 - This method is called once for each key-value pair in the input split.

REDUCER CLASS

- The Reducer class defines the Reduce job in MapReduce. It reduces a set of intermediate values that share a key to a smaller set of values.
- Reducer implementations can access the Configuration for a job via the `JobContext.getConfiguration()` method.
- A Reducer has three primary phases – Shuffle, Sort, and Reduce.
 - Shuffle – The Reducer copies the sorted output from each Mapper using HTTP across the network.
 - Sort – The framework merge-sorts the Reducer inputs by keys (since different Mappers may have output the same key). The shuffle and sort phases occur simultaneously, i.e., while outputs are being fetched, they are merged.
 - Reduce – In this phase the `reduce (Object, Iterable, Context)` method is called for each `<key, (collection of values)>` in the sorted inputs.
- Method
 - `reduce` is the most prominent method of the Reducer class. The syntax is defined below –
 - `reduce(KEYIN key, Iterable<VALUEIN> values, org.apache.hadoop.mapreduce.Reducer.Context context)`
 - This method is called once for each key on the collection of key-value pairs.

MAPREDUCE - HADOOP IMPLEMENTATION

- MapReduce is a framework that is used for writing applications to process huge volumes of data on large clusters of commodity hardware in a reliable manner.

MapReduce Algorithm

- Generally MapReduce paradigm is based on sending map-reduce programs to computers where the actual data resides.
 - During a MapReduce job, Hadoop sends Map and Reduce tasks to appropriate servers in the cluster.
 - The framework manages all the details of data-passing like issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
 - Most of the computing takes place on the nodes with data on local disks that reduces the network traffic.
 - After completing a given task, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

INPUTS AND OUTPUTS (JAVA PERSPECTIVE)

- The MapReduce framework operates on key-value pairs, that is, the framework views the input to the job as a set of key-value pairs and produces a set of key-value pair as the output of the job, conceivably of different types.
- The key and value classes have to be serializable by the framework and hence, it is required to implement the Writable interface.
- Additionally, the key classes have to implement the WritableComparable interface to facilitate sorting by the framework.
- Both the input and output format of a MapReduce job are in the form of key-value pairs –
 - (Input) $\langle k1, v1 \rangle \rightarrow \text{map} \rightarrow \langle k2, v2 \rangle \rightarrow \text{reduce} \rightarrow \langle k3, v3 \rangle$ (Output).

	Input	Output
Map	$\langle k1, v1 \rangle$	list ($\langle k2, v2 \rangle$)
Reduce	$\langle k2, \text{list}(v2) \rangle$	list ($\langle k3, v3 \rangle$)

MAPREDUCE IMPLEMENTATION

The following table shows the data regarding the electrical consumption of an organization. The table includes the monthly electrical consumption and the annual average for five consecutive years. We need to write applications to process the input data in the given table to find the year of maximum usage, the year of minimum usage, and so on.

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Avg
1979	23	23	2	43	24	25	26	26	26	26	25	26	25
1980	26	27	28	28	28	30	31	31	31	30	30	30	29
1981	31	32	32	32	33	34	35	36	36	34	34	34	34
1984	39	38	39	39	39	41	42	43	40	39	38	38	40
1985	38	39	39	39	39	41	41	41	00	40	39	39	45