Hadoop: HDFS and MapReduce

Prof. Vipul K. Dabhi
Department of Information Technology
D. D. University, Nadiad
vipul.k.dabhi@gmail.com

Outline

- Big Data
- Hadoop
  - Hadoop Architecture
  - HDFS
    - Name Node, Data Node, Secondary Name Node
  - Map Reduce
    - Job Tracker, Task Tracker
- Running Word count Example in Hadoop
- Map Reduce flowchart for Word count example
- Hadoop V1 and V2 (YARN)

Big Data and Problems

- Big Data Definition: Big data is a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools.
- Daily about 0.5 petabytes of updates are being made into FACEBOOK including 40 millions photos
- Daily, YOUTUBE is loaded with videos that can be watch for one year continuously.
- Large datasets are produced in fields like meteorology, biology and environmental research.
- This increase in size of data also affect Internet search, finance and business informatics

Why Big Data?

- Key enablers for the growth of Big Data are:
  - Increase of storage capacities (Floppy disks -> CD/DVD -> Hard Disk (500 GB) -> Hard Disk (1 TB)
  - Increase of processing power (technology to read the data has also taken a leap).
  - Increase in availability of data

How to Handle such Big Data

- Concept of Torrents
  - Reduce time to read by reading it from multiple sources simultaneously
    - Imagine if we had 100 drives, each holding one hundredth of data. Working in parallel, we could read the data in less than 2 minutes.
- How to handle system up and downs?
- How to combine the data from all systems?

Source: www.slideserve.com/jock/hadoop

How to Handle such Big Data

- How to handle system up and downs?
  - Commodity hardware for data storage
  - Chances of failure are very high
  - So have a redundant copy of same data across number of machines
  - In case of failure of one machine, you have other
  - Answer: Google File System (GFS)
- How to combine the data from all systems?
  - Analyze data across different machines. But how to merge them to get a meaningful outcome?
  - Data has to travel across network. Then only merging of data is possible.
  - Answer: Map Reduce

Source: www.slideserve.com/jock/hadoop

History of Hadoop

- Google was the first to launch GFS and MapReduce.
- They published papers of GFS and MapReduce in 2004. The technology was well proven in Google by 2004.
- Doug Cutting led the charge to develop an open source version of MapReduce system called Hadoop.
- Soon after, Yahoo and others supported this effort
- Today Hadoop is core part in:
  - FaceBook, Yahoo, LinkedIn, Twitter

Source: www.slideserve.com/jock/hadoop

What is Hadoop?

- Hadoop is an open-source software framework that supports data-intensive distributed applications.
- Goals of Hadoop:
  - Facilitate storage and processing of large and rapidly growing datasets
  - Highly scalable and available
  - Use commodity (cheap) hardware with redundancy
  - Move computation rather than data

When is Hadoop a Good Choice?

- When you must process lots of unstructured data
- When your processing can easily be made parallel
- When running batch jobs is acceptable
- When you have access to lots of cheap hardware

Source: www.tomwheeler.com/publications/2009/lambda_lounge_hadoop_200910/twheeler-hadoop-20091001-handouts.pdf

When is Hadoop Not a Good Choice?

- For calculations with little or no data
- When your processing can not be made parallel
- When you need interactive results

Source: www.tomwheeler.com/publications/2009/lambda_lounge_hadoop_200910/twheeler-hadoop-20091001-handouts.pdf

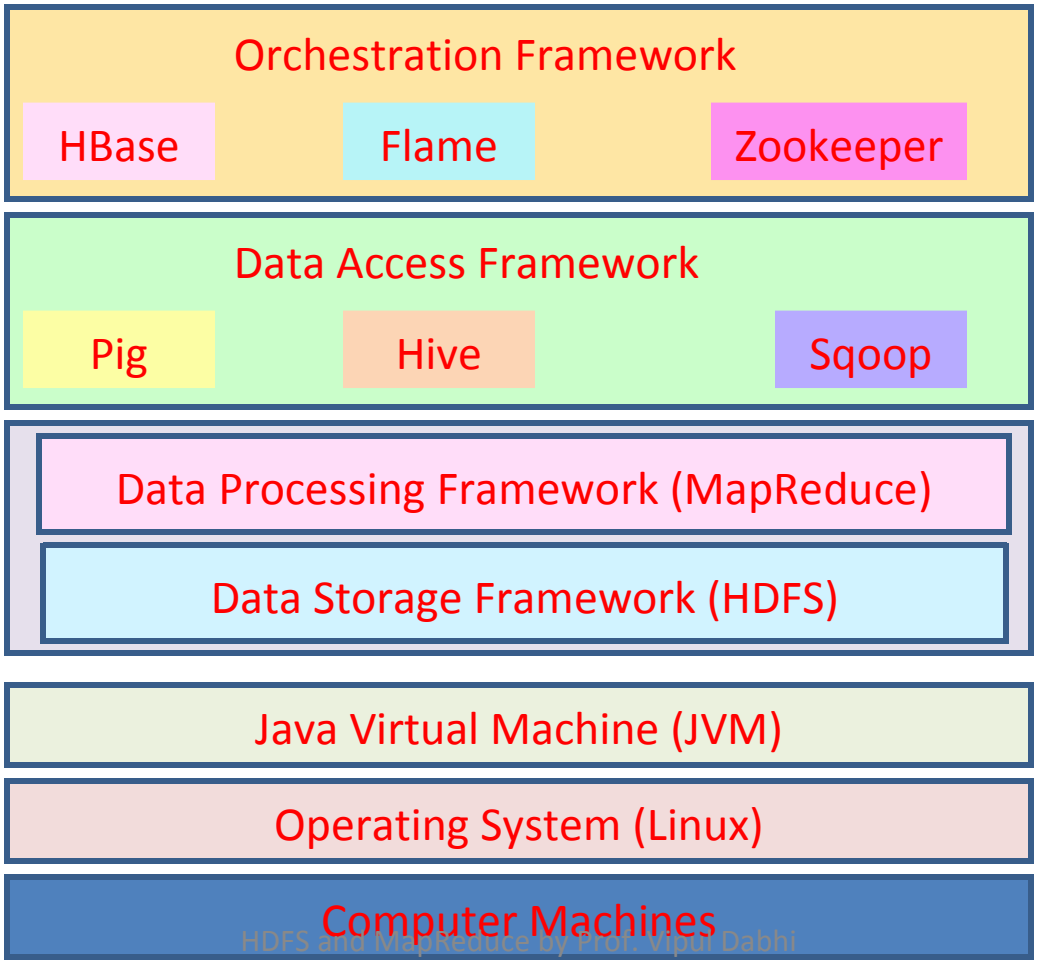
Core Hadoop Concepts

- Applications are written in high-level code
  - Developers do not worry about network programming
- Nodes talk to each other as little as possible
  - Developers should not write code which communicates between nodes
  - “Shared nothing” architecture
- Data is spread among machines in advance
  - Computation happens where the data is stored
  - Data is replicated multiple times on the system to improve reliability and availability.

Hadoop Components

- Hadoop consists of two main components
  - Hadoop Distributed File System (HDFS) (For Storing Data)
  - MapReduce Framework (For Processing Data)
- Hadoop EcoSystem
  - Projects developed around core Hadoop
  - Pig, Hive, Flume, Oozie, Sqoop
- Hadoop Cluster
  - A set of machines running HDFS and MapReduce
  - A cluster can have minimum one node and maximum as many as thousand nodes

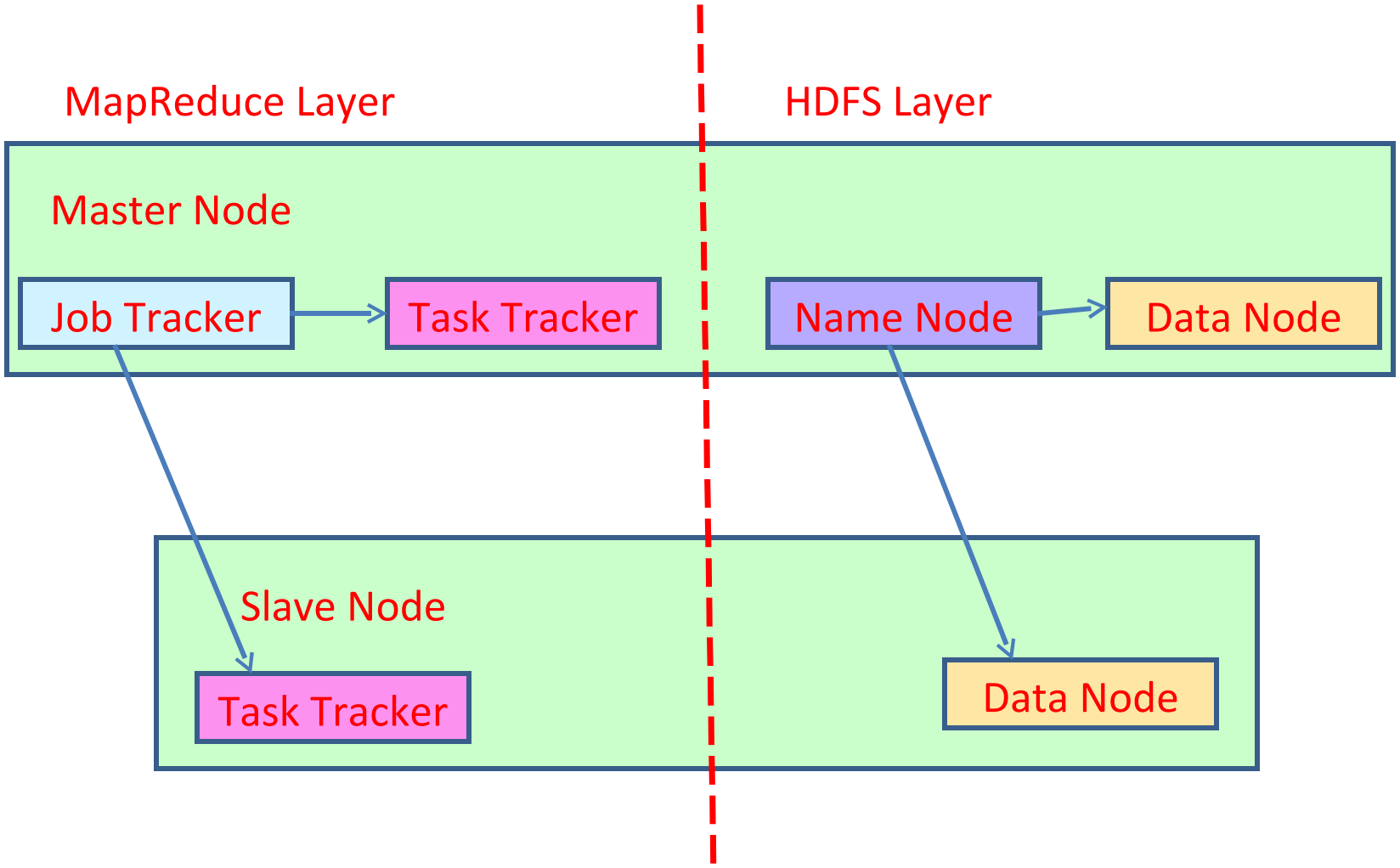
Hadoop Framework Tools



Hadoop Cluster

- Hadoop cluster integrates HDFS and MapReduce
- It follows master/slave architecture
- Master node contains
  - Job tracker node (MapReduce Layer)
  - Task tracker node (MapReduce Layer)
  - NameNode (HDFS layer)
  - DataNode (HDFS layer)
- Multiple slave nodes contain
  - Task tracker node (MapReduce Layer)
  - DataNode (HDFS Layer)

Hadoop Cluster



Hadoop Cluster: MapReduce Framework

- Per Cluster Node:
  - Single JobTracker per master
    - Responsible for scheduling the jobs' component tasks on the slaves
    - Monitors slave progress
    - Re-executes failed tasks
  - Single TaskTracker per slave
    - Execute the tasks as directed by the master

HDFS

- Hadoop Distributed File System (HDFS) is based on Google's GFS
- HDFS is implemented using Java
- HDFS sits on top of native filesystem (Ex. ext 3)
- HDFS can be part of a Hadoop cluster or can be a stand-alone general purpose distributed file system.
- Provides redundant storage for massive amounts of data using cheap and unreliable hardware

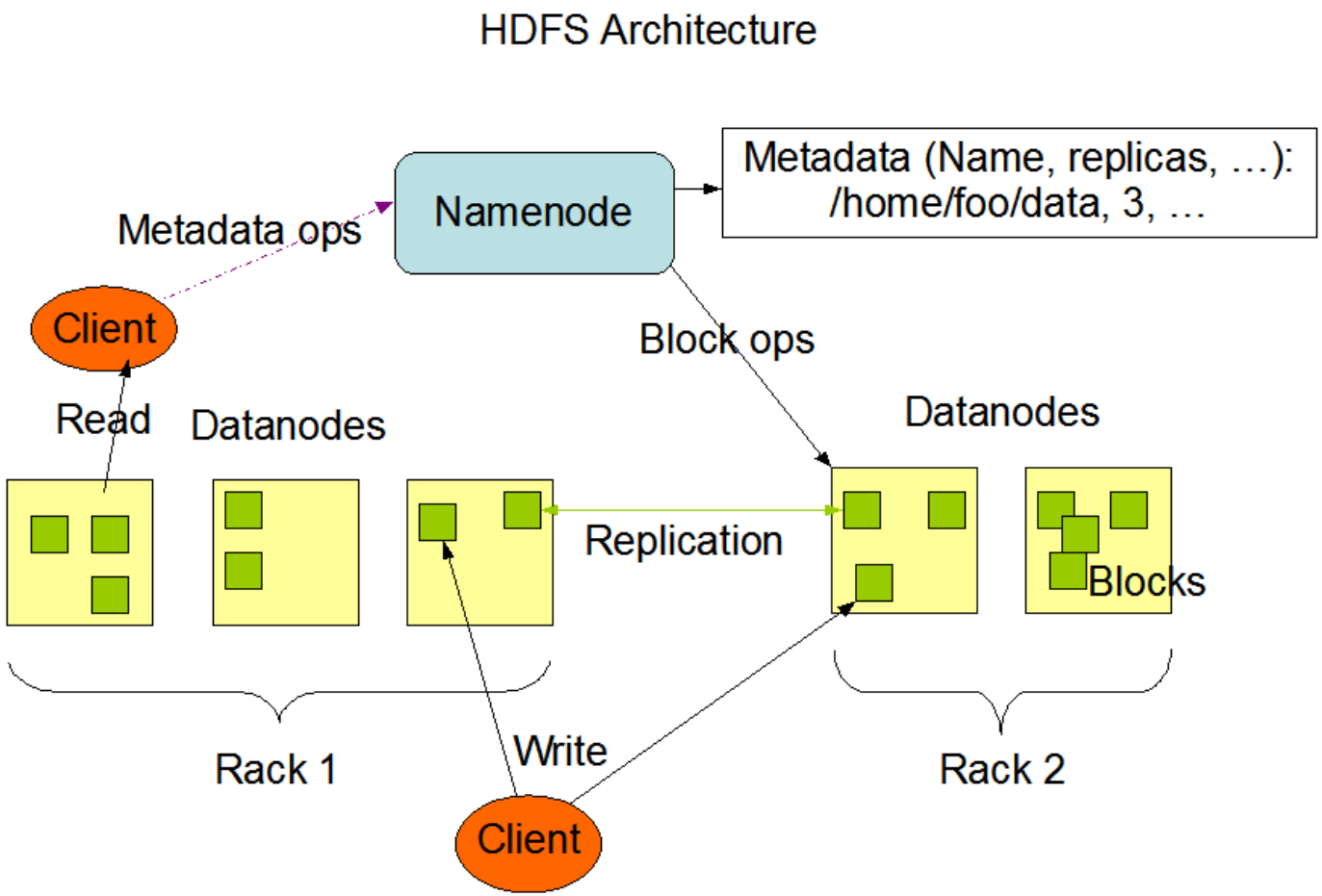
HDFS

- Hadoop Distributed File System is responsible for storing data on the cluster
- Data files are split into blocks and distributed across multiple nodes in the cluster
  - Block Size: UNIX = 4 KB vs HDFS = 64 or 128 MB
- Each block is replicated multiple times on multiple machines (DataNodes) to achieve reliability and availability
  - Default is to replicate each block three times
  - Replicas are stored on different nodes
- A master node called, NameNode keeps track of which blocks make up a file and where those blocks are located (known as metadata)

HDFS characteristics

- Very Large Files
  - MBs to PBs
- Streaming
  - Write once read many times
  - After huge data being placed -> we tend to use the data not modify it
  - Time to read the whole data is more important
- Commodity Cluster
  - No High end Servers
  - High chance of failure (HDFS is tolerant enough)
  - Replication is done

HDFS Architecture



Source: www.cse.buffalo.edu/faculty/bina/MapReduce/HDFS.ppt

HDFS Components

- NameNode
- DataNode
- SecondaryNameNode
- Master /Slave architecture
- HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients
- There are number of DataNodes

Name Node

- NameNode stores metadata for files
- The server holding NameNode instance is very crucial (Single point of Failure): Runs on highly reliable hardware
- Handles creation of more replica blocks when necessary after a DataNode failure

NameNode Metadata

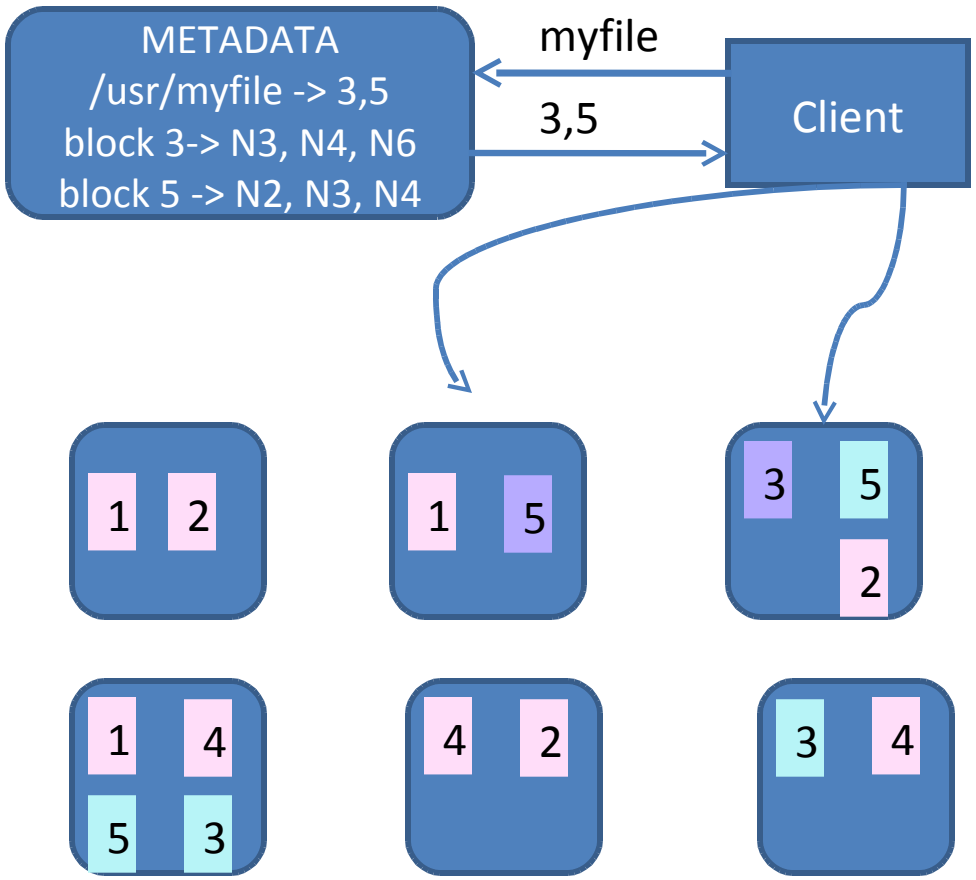
- Contents of NameNodeMetadata
  - List of files
  - List of blocks for each file
  - List of DataNodes for each block
  - File attributes, e.g.: creation time, replication factor
  - Transaction Log: records file creation, file deletion

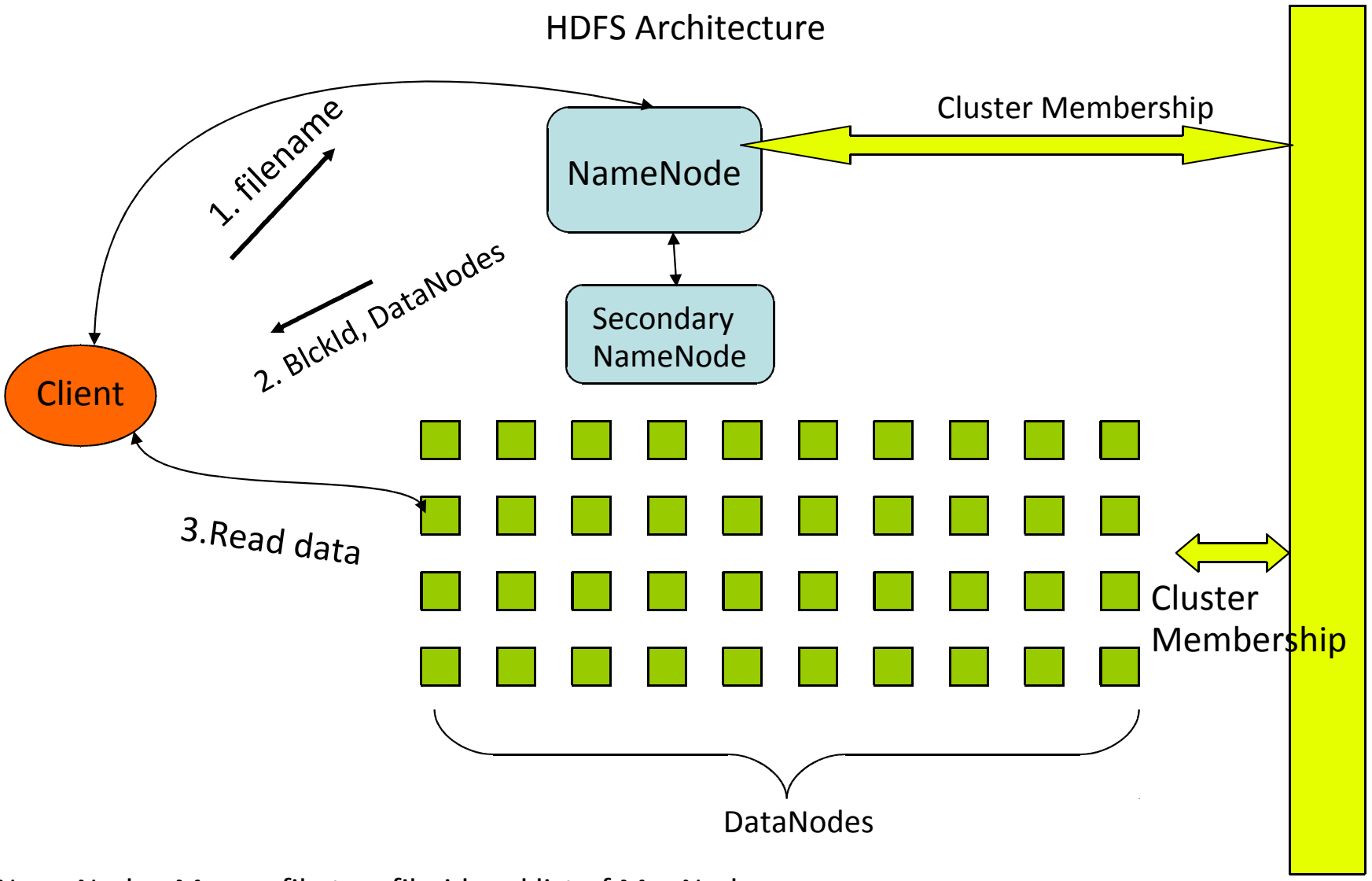
DataNode

- DataNode stores actual data in HDFS
- Runs on underlying file system (ext3, NTFS)
- Periodically sends a report of all existing blocks to NameNode
- DataNodes send heartbeat to the NameNode once every 3 seconds
- NameNode uses heartbeats to detect DataNode failure

Reading a File

- When a client application wants to read a file:
  - It communicates with the NameNode to determine which blocks make up the file, and which DataNodes those block reside on
  - It then communicates directly with the DataNodes to read the data





NameNode : Maps a file to a file-id and list of MapNodes
DataNode : Maps a block-id to a physical location on disk
SecondaryNameNode: Periodic merge of Transaction log

Source: www.csie.ndhu.edu.tw/~showyang/Cloud2010f/08Hadoop&Hive.pdf

Data Node: Pipelining of Data

- Client retrieves a list of DataNodes on which to place replicas of a block
- Client writes block to the first DataNode
- The first DataNode forwards the data to the next DataNode in the pipeline
- When all replicas are written, the client moves on to write the next block in file

Data Node Failure

- A subset of DataNodes may lose connectivity to the NameNode
- NameNode detects this condition by the absence of a Heartbeat message
- NameNode marks DataNodes without Heartbeat and does not send any IO requests to them
- Any data stored on failed DataNode is not available to the HDFS- cause replication factor of some of the blocks to fall below their specified value.

Getting Data in and out of HDFS

- Hadoop API
  - Use hadoop fs command
    - hadoop fs –ls
    - hadoop fs –mkdir TempDir
    - hadoop fs –put SampleTextFile.txt (local_dir) HadoopTextFile.txt (hdfs_dir)
    - hadoop fs -cat HadoopTextFile.txt
    - Hadoop fs –rm HadoopTextFile.txt
  - Ecosystem Projects
    - Flume: Collects data from log generating sources (Websites)
    - Sqoop: Extracts and/or inserts data between HDFS and RDBMS.

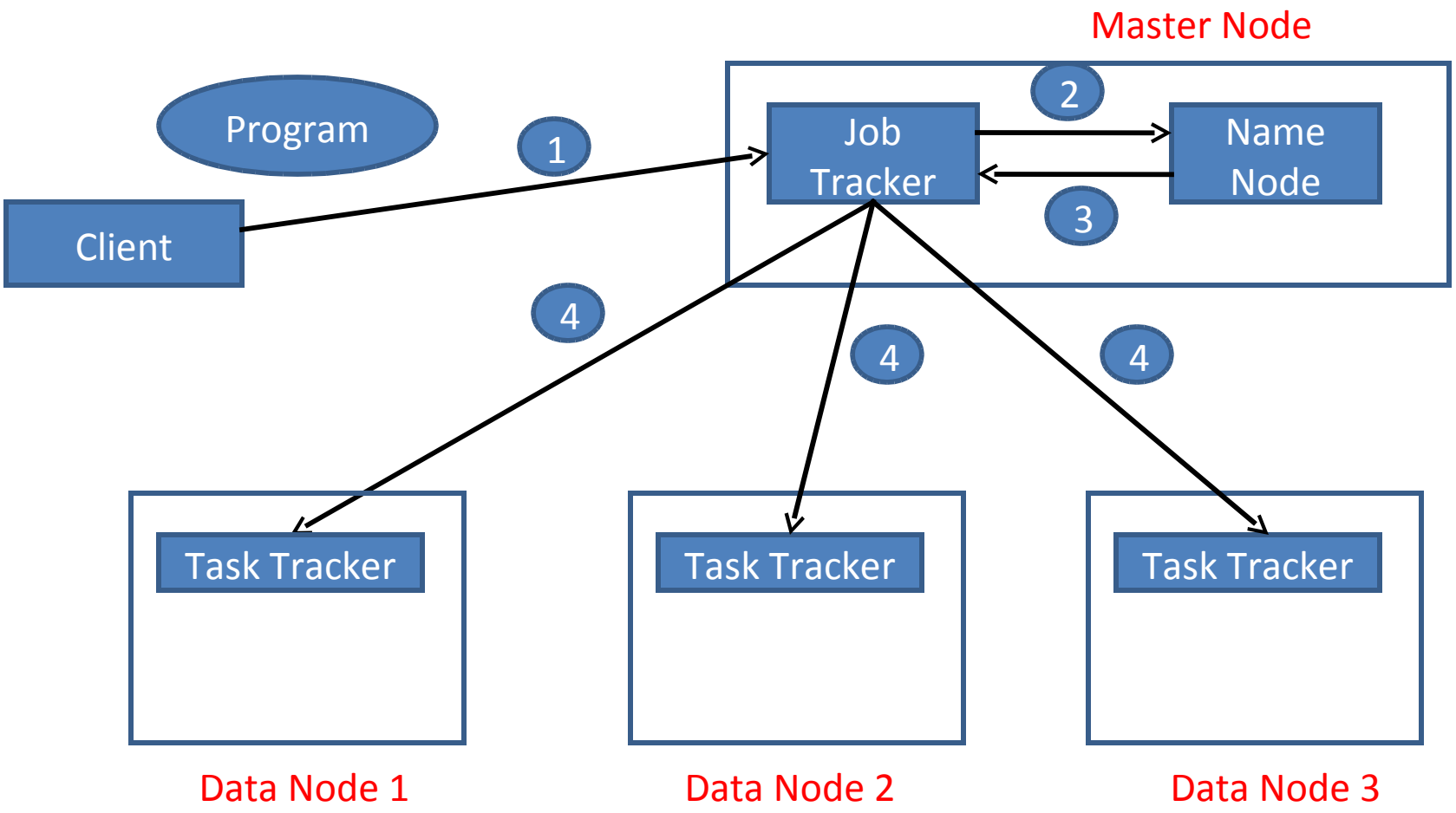
What is MapReduce

- MapReduce is a programming model and software framework developed by Google
- Aims at processing of vast amounts of data in parallel on large clusters (multiple nodes) of commodity hardware
- Each node processes data stored on that node
- Consists of two phases:
  - Map
  - Reduce

MapReduce core Functionality

- MapReduce code is usually written in Java. However, we can write MapReduce program using other programming languages (Ruby, C++, Python).
- Two phases:
  - Map Step:
    - Master node takes large problem input and slices it into smaller sub problems; distributes these to worker nodes.
    - Worker node may do this again; leads to a multi-level tree structure
    - Worker processes smaller problem and hands back to master
  - Reducer Step:
    - Master node takes the answers to the sub problems and combines them in a predefined way to get the output to original problem.

Executing Program on Data



WordCount Example: MapReduce Flowchart

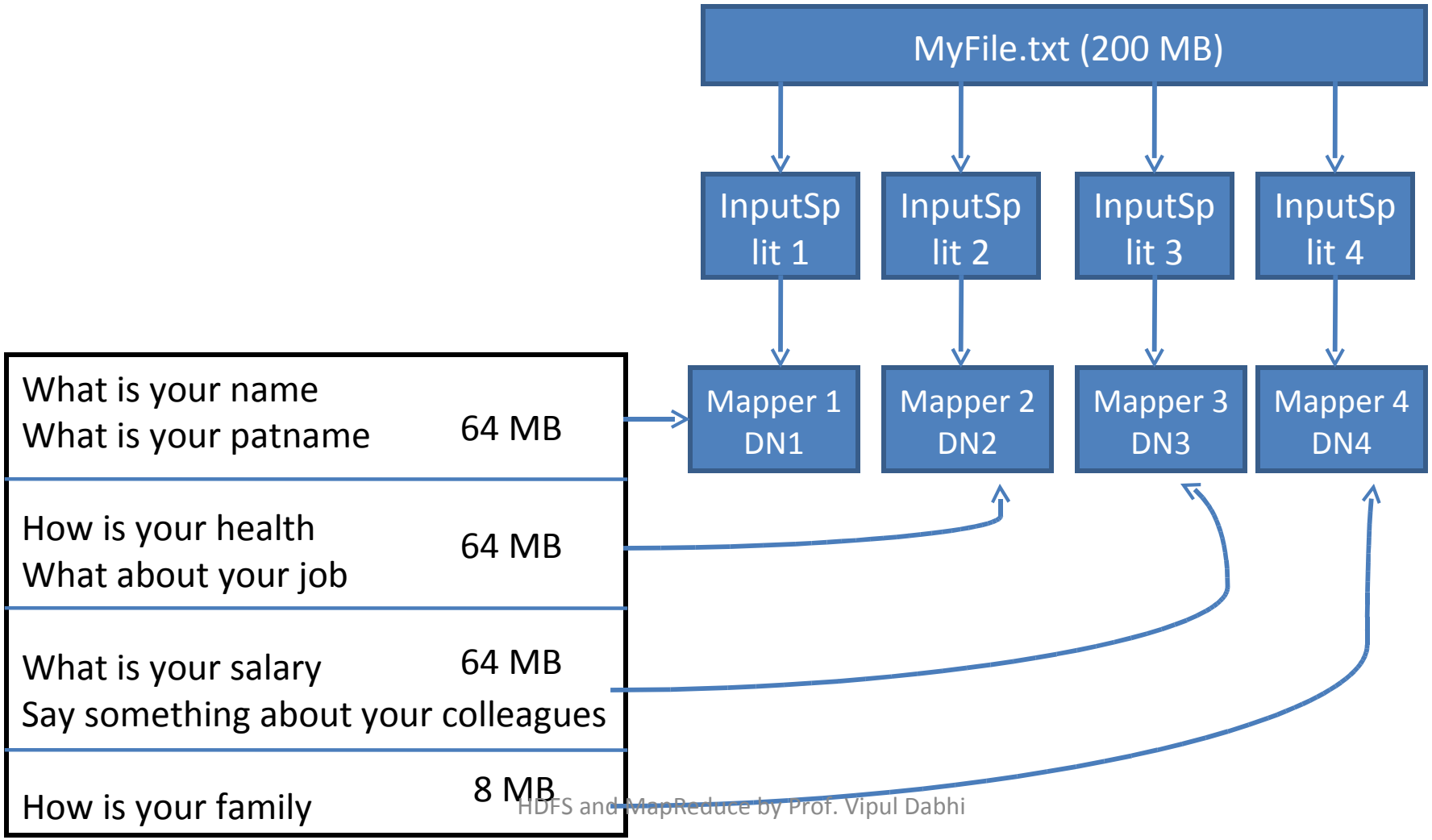
- Counting number of occurrences of words in given file
- Consider a file name “MyFile.txt” with the size of 200 MB and HDFS block size is 64 MB

MyFile.txt

What is your name
What is your patname
How is your health
What about your job
What is your salary
Say something about your colleagues
How is your family

What is your name What is your patname	64 MB
How is your health What about your job	64 MB
What is your salary Say something about your colleagues	64 MB
How is your family	8 MB

WordCount Example: MapReduce Flowchart

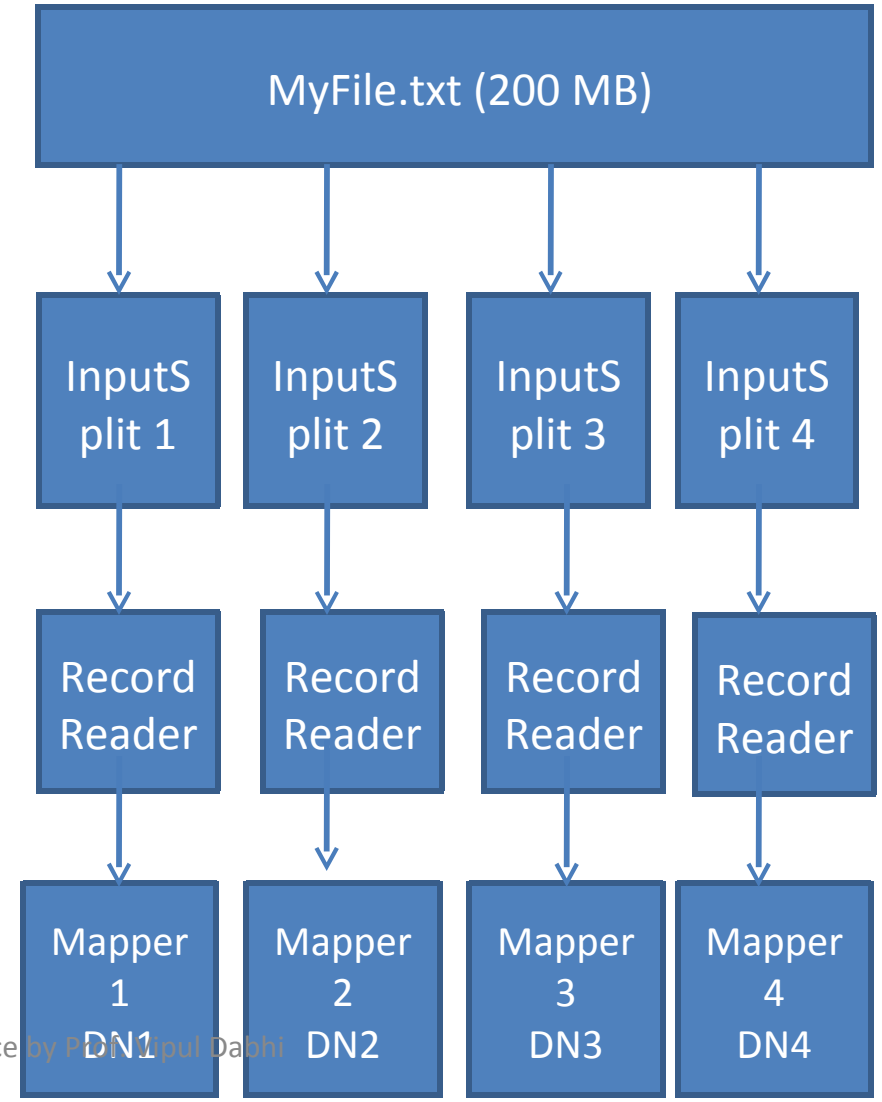


WordCount Example: MapReduce Flowchart

- Number of mappers will be equal to the number of input splits.
- Each Mapper will be executed by TaskTracker (MapReduce: MR) on DataNode (HDFS)
- TaskTracker (MR) will receive this job from JobTracker (MR).

WordCount Example: MapReduce Flowchart

- Both Mappers and Reducers can work with <key,value> pairs only.
- So, there is a need to convert text line (of MyFile.txt> to <key,value> pairs.
- For doing this conversion there is an interface named RecordReader.
- The implementation of RecordReader is done by Hadoop framework itself.



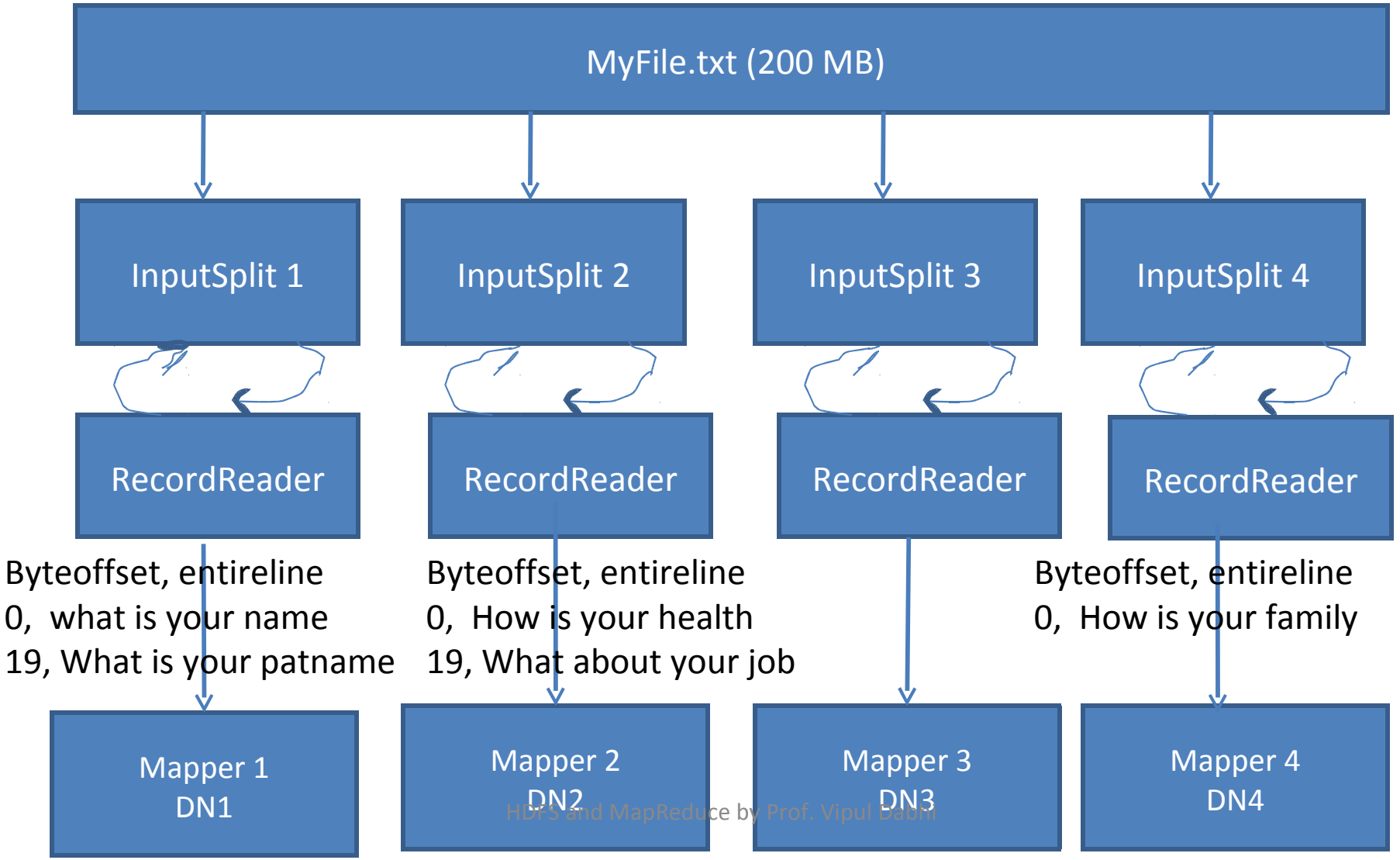
WordCount Example: MapReduce Flowchart

- In Hadoop terminology, we refer every line as one Record.
- There are two records for first three InputSplits and one record for last InputSplit.
- On what basis the RecordReader converts these records into <key,value> pairs depends on format of input file.
- RecordReader knows how to read one Record from corresponding InputSplit, that is why the name RecordReader.

WordCount Example: MapReduce Flowchart

- When the RecordReader tries to convert one Record to <key,value> pairs, it considers the format of file.
- Four formats (Predefined classes) of File are:
  - TextInputFormat
  - KeyValueTextInputFormat
  - SequenceFileInputFormat
  - SequenceFileAsTextInputFormat
- The input file should be either of these four formats.

WordCount Example: MapReduce Flowchart



RecordReader

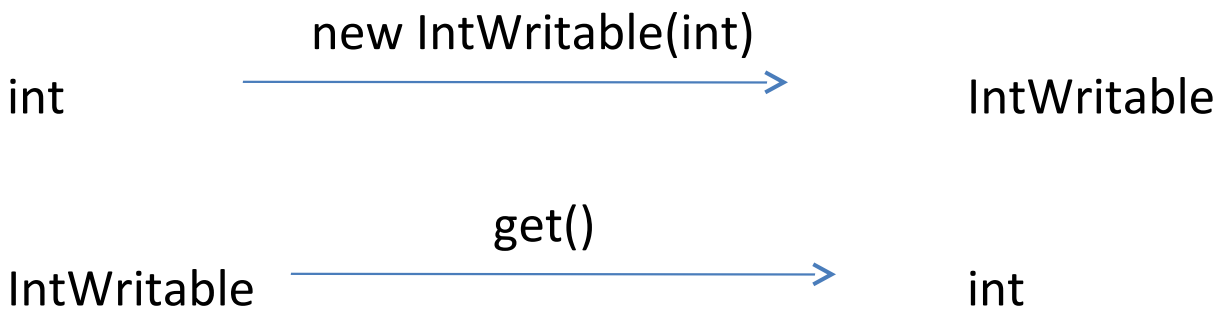
- For number of <key,value> pairs generated by RecordReader those many times a Mapper will get executed.
- For example, in this scenario, RecordReader1 produces two <key,value> pairs so Mapper1 gets executed twice.
- The Mapper is getting <key,value> pairs. These <key,value> pairs must be of type Object.

Input of map(): Box Classes

- Collection Framework can only work with Objects.
- Java has introduced wrapper classes corresponding to its primitive data types.
- Similar to wrapper classes in Java, Hadoop has introduced Box classes.
- map() function of Mapper class takes <key,value> pairs, object types must be any of box classes.

Box Classes in Hadoop

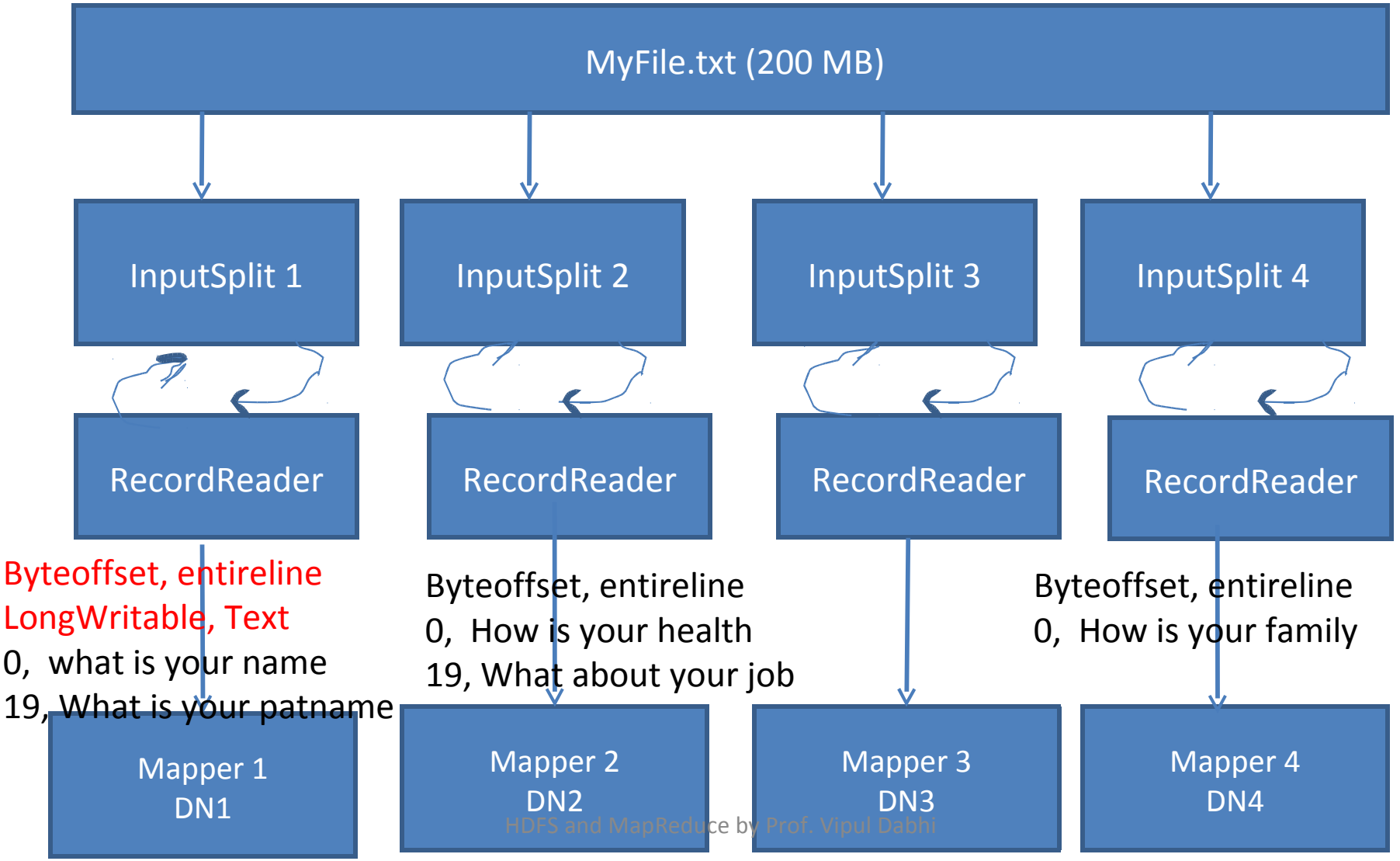
Primitive Data Type	Wrapper Class in Java	Box Class in Hadoop
int	Integer	IntWritable
long	Long	LongWritable
float	Float	FloatWritable
double	Double	DoubleWritable
String	String	Text
char	Character	Text



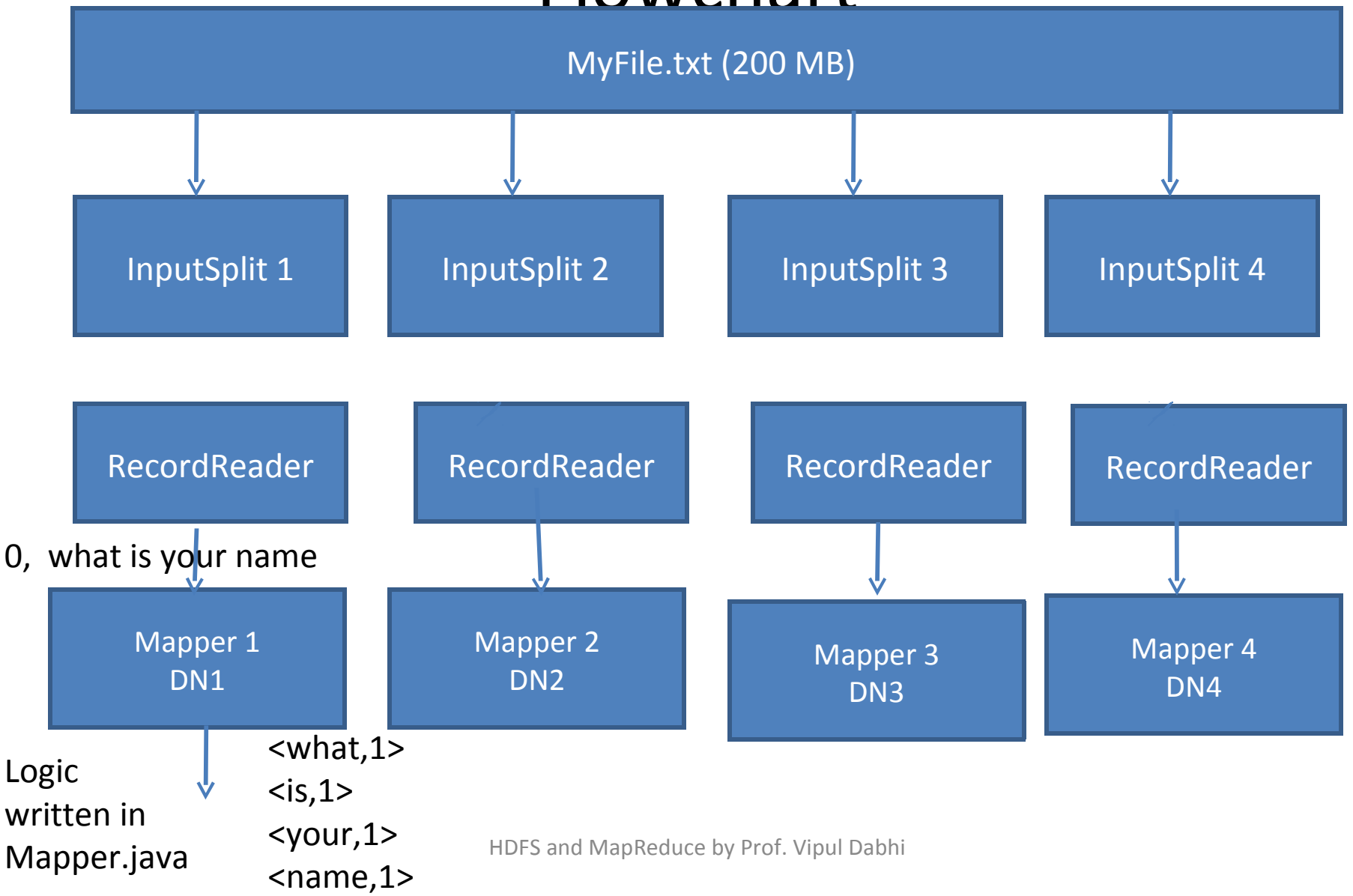
Up to Java version 1.4, we have used utility methods like parseInt(), valueOf() for conversion of wrapper class to primitive data types and vice versa.

But from version 1.4 onwards, autoboxing and autounboxing can be used for this conversion.

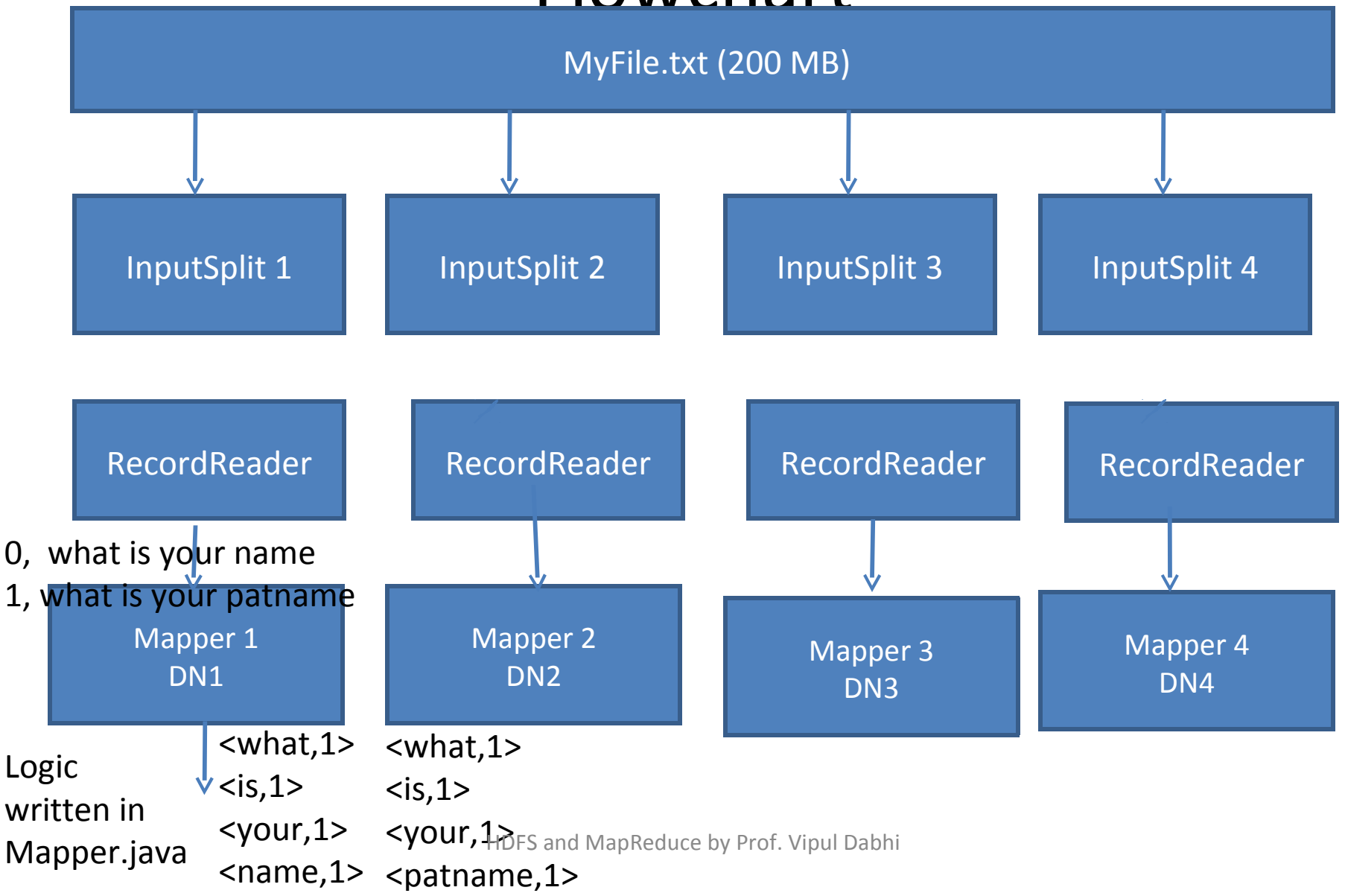
WordCount Example: MapReduce Flowchart



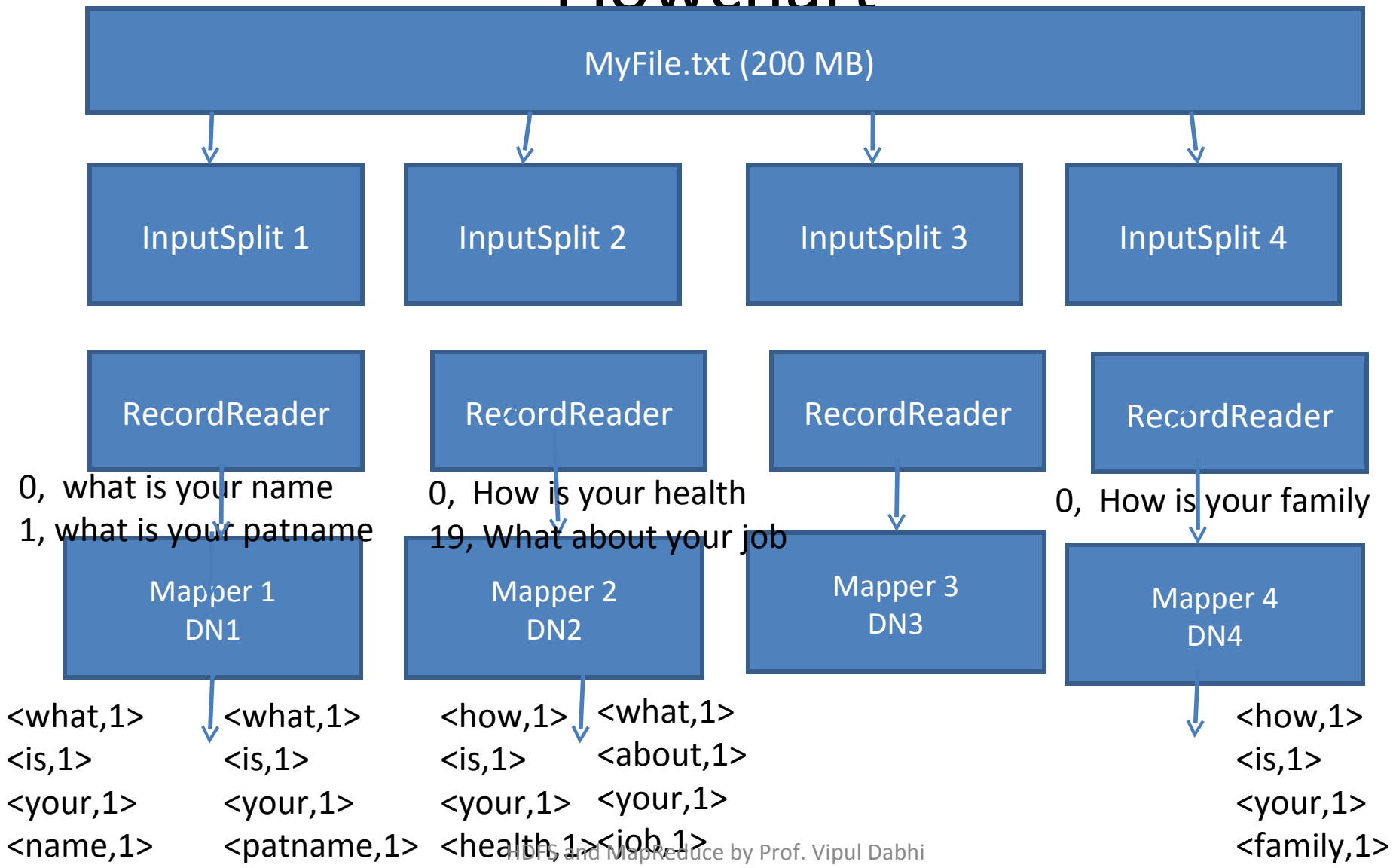
WordCount Example: MapReduce Flowchart



WordCount Example: MapReduce Flowchart



WordCount Example: MapReduce Flowchart



Intermediate Data

- The mapper produces <key,value> pairs.
- It produced <key,value> pairs are of Box class types.
- For this example,
  - Input to Mapper is of type <LongWritable, Text>
  - Output of Mapper is of type <Text, IntWritable>.
- The <key,value> pairs generated by Mappers are called Intermediate data.

Intermediate Data

- Intermediate Data: Data generated between Mapper and Reducer.
- We can have duplicate keys in Intermediate data. But ideally, you should not have duplicate keys but duplicate values.
- The intermediate data should pass through two more phases: shuffling and sorting.

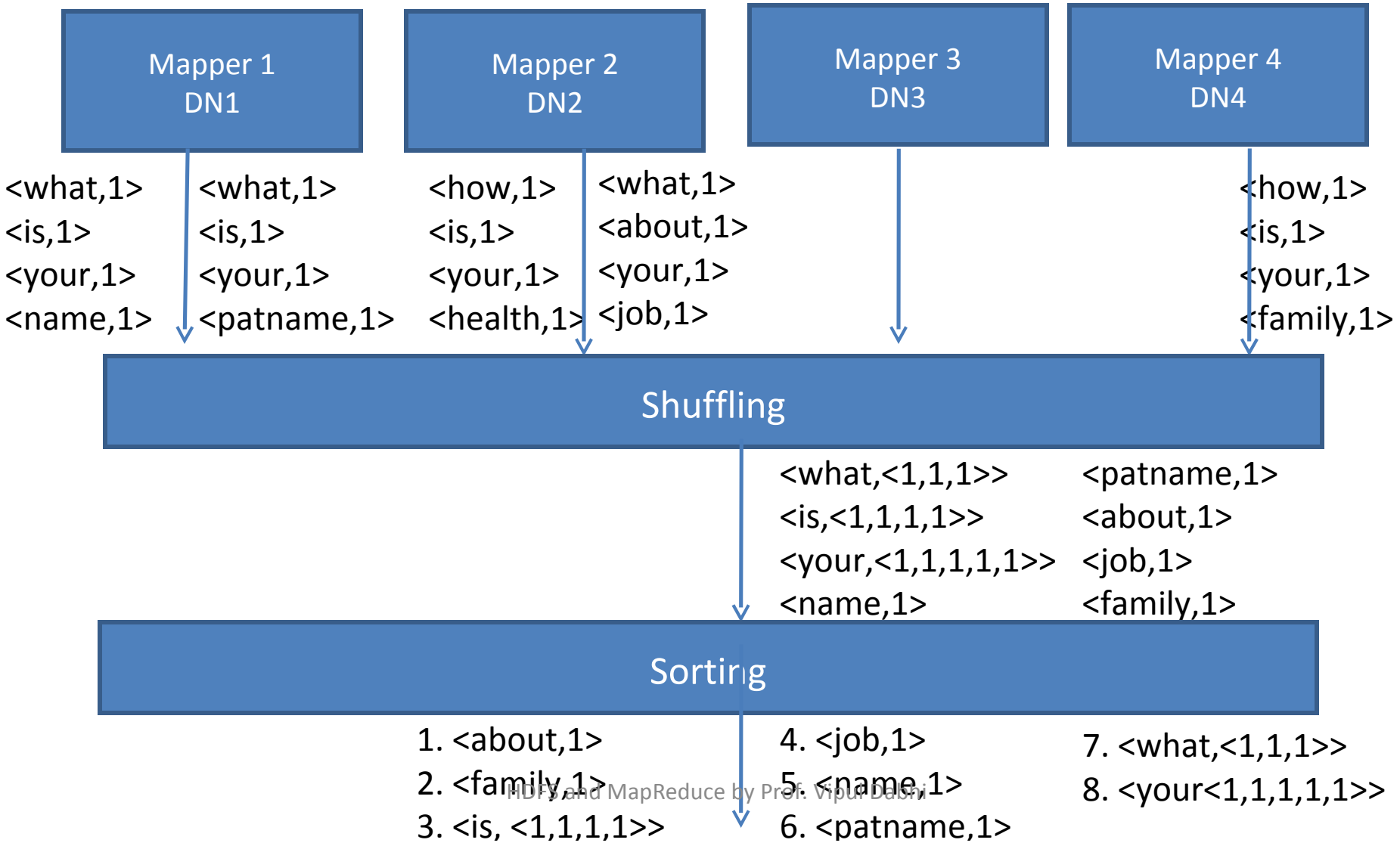
Shuffling and Sorting Phase

- Shuffling: Combine all values associated to single (unique) identical keys.

<what,1>	**<what,1>**	<how,1>	**<what,1>**	<how,1>
<is,1>	<is,1>	<is,1>	<about,1>	<is,1>
<your,1>	<your,1>	<your,1>	<your,1>	<your,1>
<name,1>	<patname,1>	<health,1>	<job,1>	<family,1>

  - <what, <1,1,1>>
  - <is,<1,1,1,1>>
- Shuffling phase makes sure that we will not have any duplicate keys.
- By comparing the keys, the <key,value> pairs can be sorted.

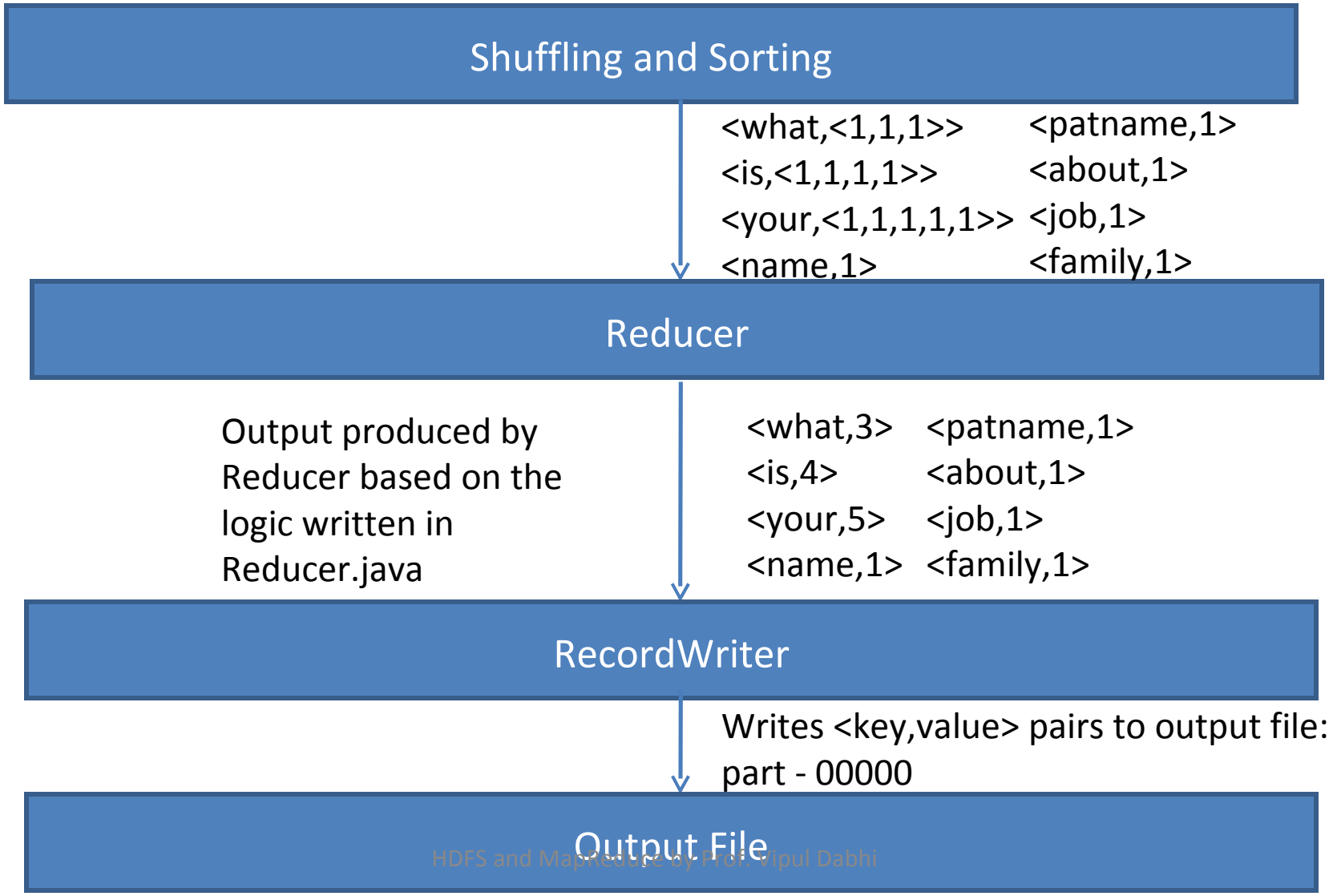
Shuffling and Sorting



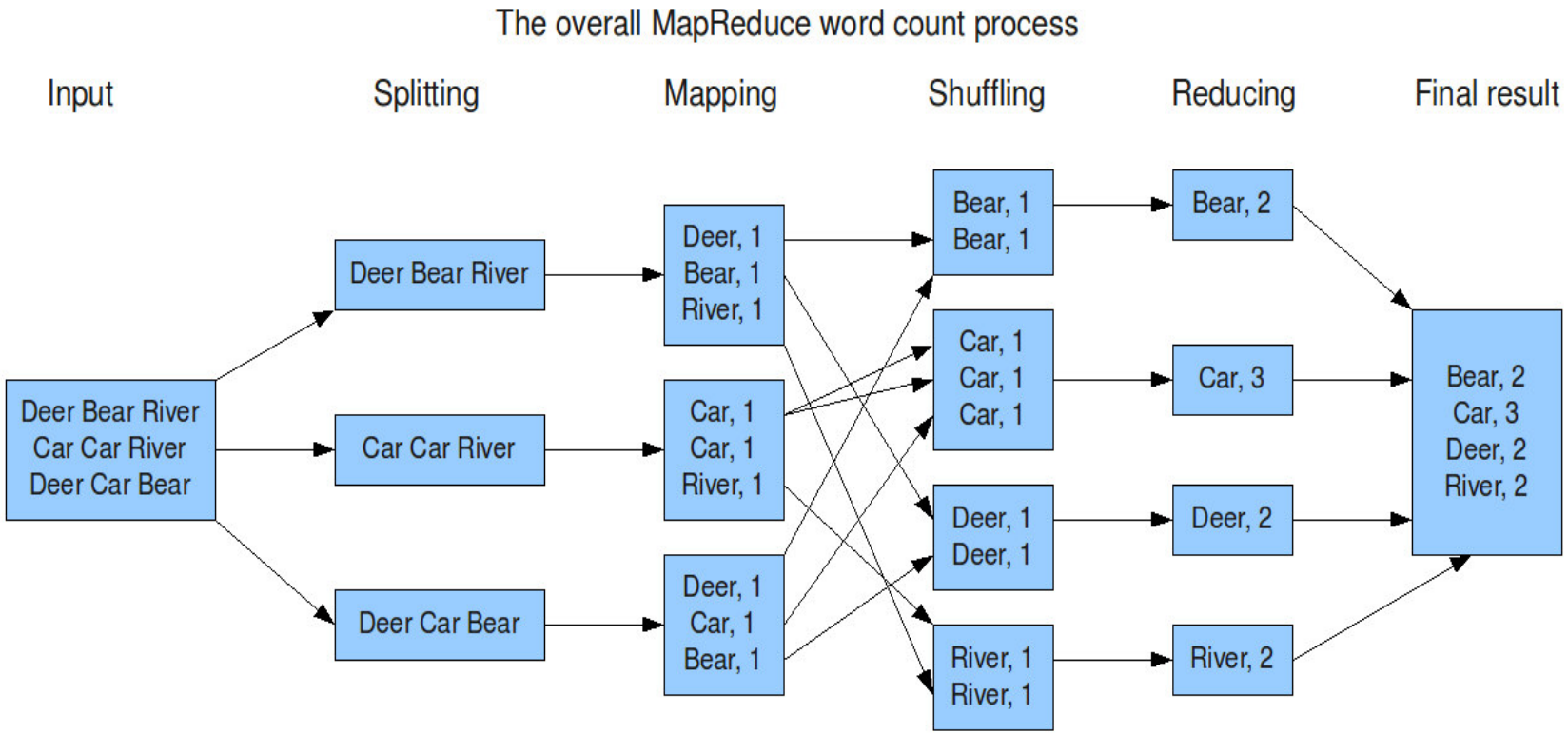
Reducer

- Reducer will combine the <key,value> pairs generated by different Mappers.
- The Reducer will get executed as per number of <key,value> pairs generated after Shuffling and Sorting phase.
- When you implement your own Reducer, automatically shuffling and sorting functionalities will be provided by Hadoop.
- If you use default Reducer, only sorting functionality will be provided by Hadoop framework.
- The output produced by Reducer is of type <Text, IntWritable>.

Reducer


Shuffling and Sorting
<what,<1,1,1>>
<is,<1,1,1,1>>
<your,<1,1,1,1,1>>
<name,1>
<patname,1>
<about,1>
<job,1>
<family,1>
Reducer
Output produced by Reducer based on the logic written in Reducer.java
<what,3>
<is,4>
<your,5>
<name,1>
<patname,1>
<about,1>
<job,1>
<family,1>
RecordWriter
Writes <key,value> pairs to output file: part - 00000
Output File

Word Count Example



Points need to be emphasized

- No reduce can begin until map is complete
- Master must communicate locations of intermediate files
- Tasks scheduled based on location of data
- If map worker fails any time before reduce finishes, task must be completely rerun
- MapReduce library does most of the hard work for us.

Source: grid.hust.edu.cn/xhshi/course/files/talk-MapReduce.ppt

Word Count Mapper

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

  public static 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.hasNext()) {
      word.set(tokenizer.nextToken());
      output.collect(word, one);
      }
    }
 }
```

Word Count Reducer

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

Word Count Example

- Jobs are controlled by configuring *JobConfs*
- JobConfs are maps from attribute names to string values
- The framework defines attributes to control how the job is executed
  - `conf.set("mapred.job.name", "MyApp");`
- Applications can add arbitrary values to the JobConf
  - `conf.set("my.string", "foo");`
  - `conf.set("my.integer", 12);`
- JobConf is available to all tasks

Putting it all together

- Create a launching program for your application (Drivercode: wordcount.java)
- The launching program configures:
  - The *Mapper* and *Reducer* to use
  - The output key and value types (input types are inferred from the *InputFormat*)
  - The locations for your input and output
- The launching program then submits the job and typically waits for it to complete

Putting it all together

```
public class WordCount extends Configured implements Tool {
//Driver Code class has user defined configuration. For ex. What mapper class, reducer class, input and ouput value types is to be used.

 public int run (String args[]) throws Exception {
// To give user-defined configuration, we must creat object of JobConf class.
JobConf conf = new JobConf(WordCount.class);  // ---- Drivercode class name as argument of constructor.
conf.setJobName("wordcount");   // ----- set job name

conf.setMapOutputKeyClass(Text.class); // --- set map output key class
conf.setMapOutputValueClass(IntWritable.class); // -- set map output value class

conf.setMapperClass(Map.class); // --- set mapper class
conf.setReducerClass(Reduce.class); // --- set reducer class

conf.setOutputKeyClass(Text.class); // --- set output key class
conf.setOutputValueClass(IntWritable.class);  //----- set output value class

FileInputFormat.setInputPaths(conf, new Path(args[0])); //---- set input file. Convert string (filename) into path.
FileOutputFormat.setOutputPath(conf, new Path(args[1])); // ---O/P directory should be only one
JobClient.runJob(conf); //--- Submit the job for running
return 0;
   }
}
```

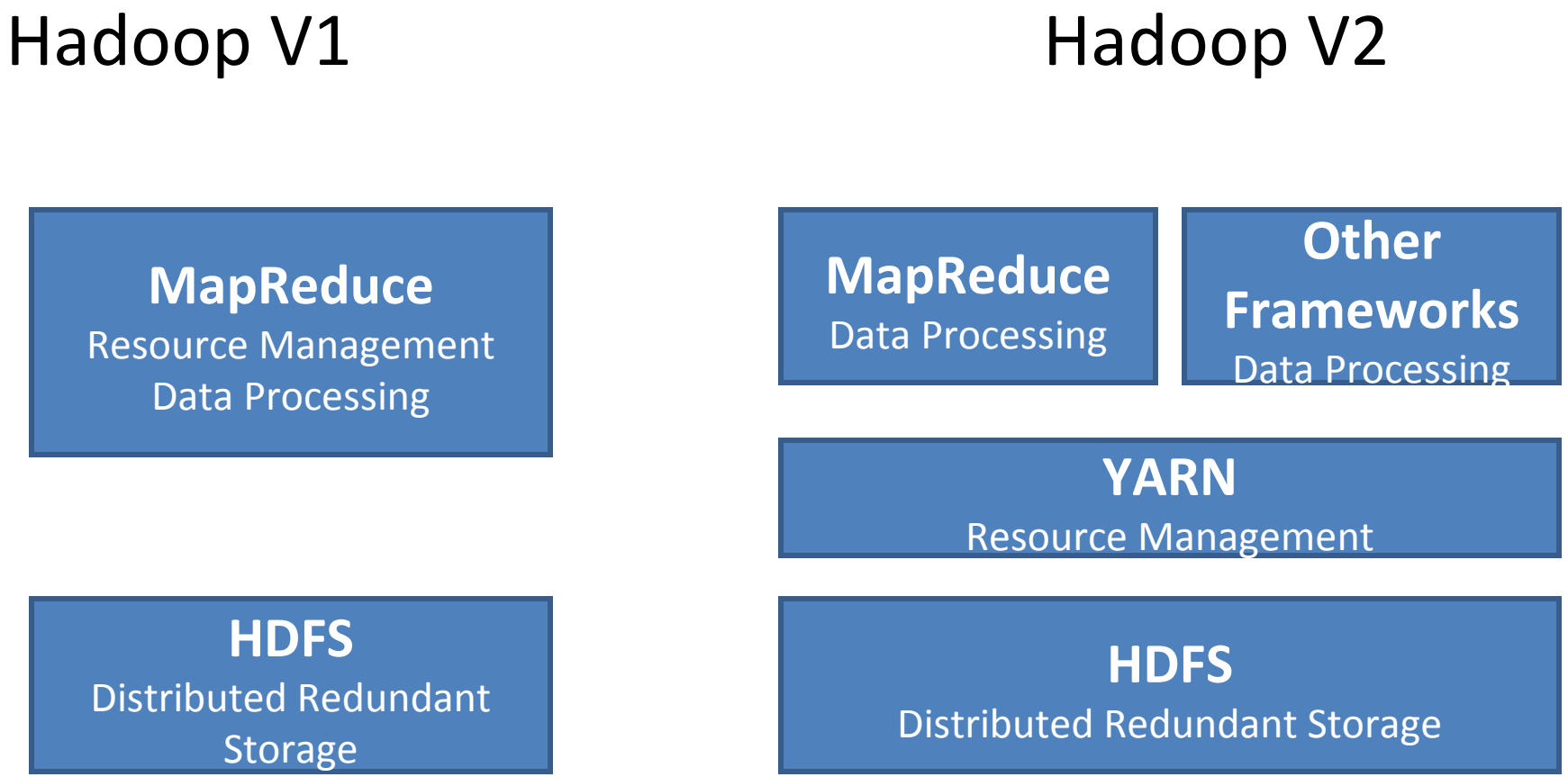
How to run the program

- $ hadoop jar wc.jar WordCount i/p o/p
  - WordCount is a driver code class.
  - Where i/p and o/p are command line arguments. Argument “i/p” specifies input file name and argument “o/p” specifies output directory.
  - You can have multiple input files but only one o/p directory.

How many Maps and Reduces

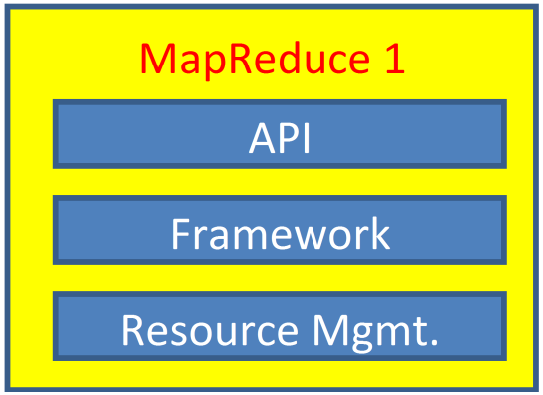
- Maps
  - Usually as many as the number of HDFS blocks being processed, this is the default
  - Else the number of maps can be specified as a hint
  - The number of maps can also be controlled by specifying the *minimum split size*
- Reduces
  - 2-3, Unless the amount of data being processed is small

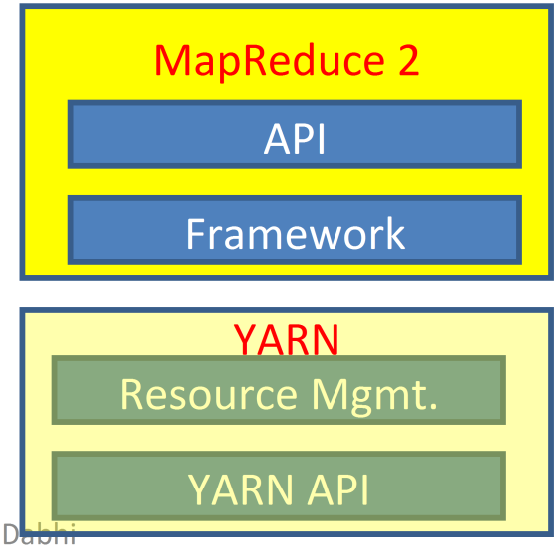
Hadoop V1 and V2



YARN

- YARN/MapReduce2 is core component of Hadoop V2 and added to improve performance of Hadoop.
- YARN stands for “Yet Another Resource Negotiator”.
- It is a layer that separates the resource management layer and processing component layer.
- MapReduce2, moves Resource Management (like infrastructure to monitor nodes, allocate resources and schedule jobs) into YARN.





Need for YARN?

- In MapReduce1, Jobtracker is responsible for both job scheduling and monitoring progress of tasks.
- Scalability bottleneck caused by having a single JobTracker. According to Yahoo, practical limits of such design are reached with a cluster of 5000 nodes and 40000 tasks running concurrently.
- The computational resources on each slave node are divided by a cluster administrator into a fixed number of map and reduce slots.
  - So a slave node can't run more map tasks than map slots at any given moment, even if no reduce tasks are running. This is not proper utilization of resources.
- Hadoop was designed to run MapReduce jobs only
  - JobTracker application is built to run MapReduce jobs only. So problem arises when non-mapreduce (other type of distributed) application tries to run in this framework.

Major Stable Releases of Hadoop

- Major stable releases
  - 1.0 release in 2011
  - 2.2 release in 2013 – YARN
  - 2.4 release in 2014 – Enterprise Features
  - Latest is 2.7.x in December 2017
  - Cloud based hadoop distribution
    - Virtual machine clusters

Known Applications of MapReduce

- Distributed sort
- Document clustering
- Analyzing and indexing textual information
- Analyzing similarities of user's behavior
- Building recommender system
- Image retrieval engine
- Generating Web graphs

Big Data Processing (Analytics)

Big Data Analytics is the process of examining large data sets to uncover hidden Patterns, unknown correlations, market trends and other useful business information.

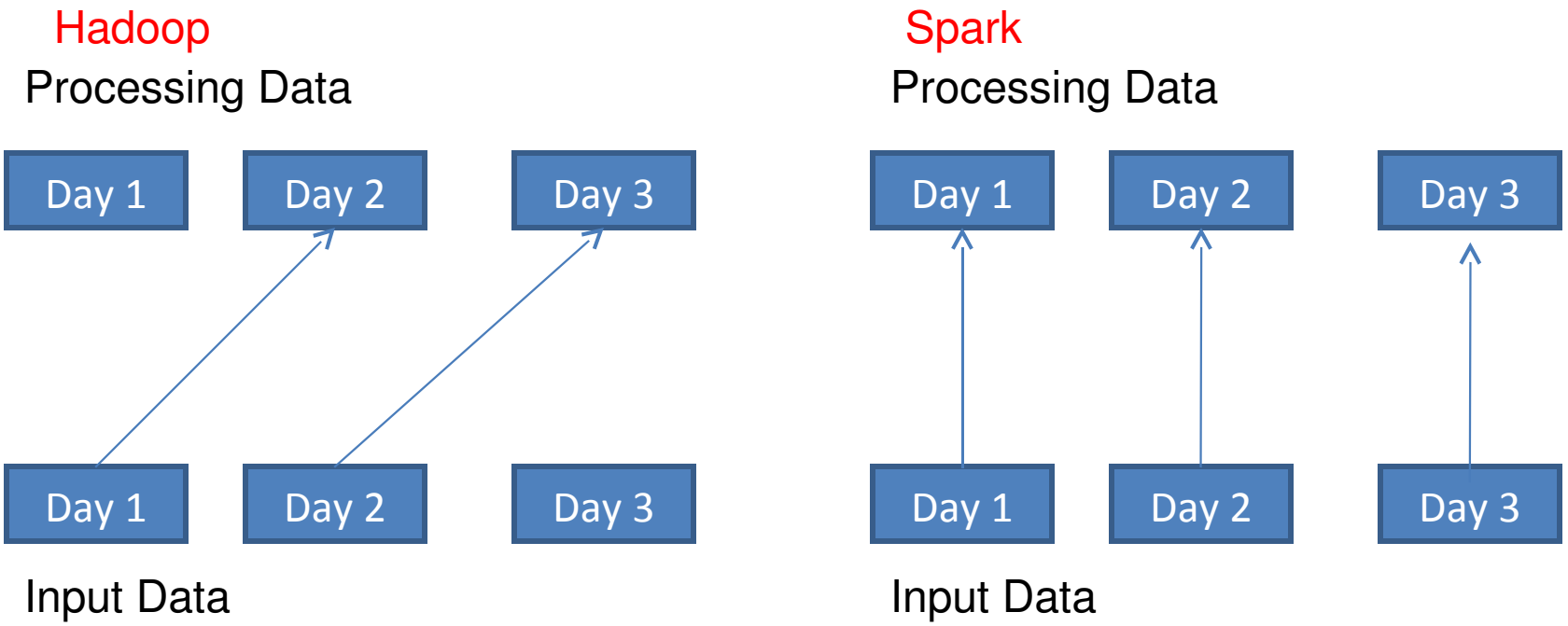
Big Data Analytics is of two types:
(1) Batch Analytics (2) Real-Time Analytics

Batch Processing	Real-Time (Stream) Processing
The collection and storage of data, for processing at a scheduled time when a sufficient amount of data has been accumulated.	The immediate processing of data after the transaction occurs, with the database being updated at the time of the event.
Examples: Cheque Clearing, Generation of Bills	Examples: Ticket Reservation Systems
Many transactions take place at once.	The act of processing data is repetitive.
Data takes time to be processed.	Data is processed immediately.

Apache spark can be used for near real time processing of data as well as batch processing of data.
Otherwise most of the time hadoop framework is used for batch processing of data.

Why Apache Spark?

- In Hadoop, you can only work on batch-time processing. Hadoop is not meant for real-time processing of data.
- Hadoop processes the data stored over a period of time. Spark overcomes the time lag issue.
- Spark can do both (batch and real time) kind of data processing.

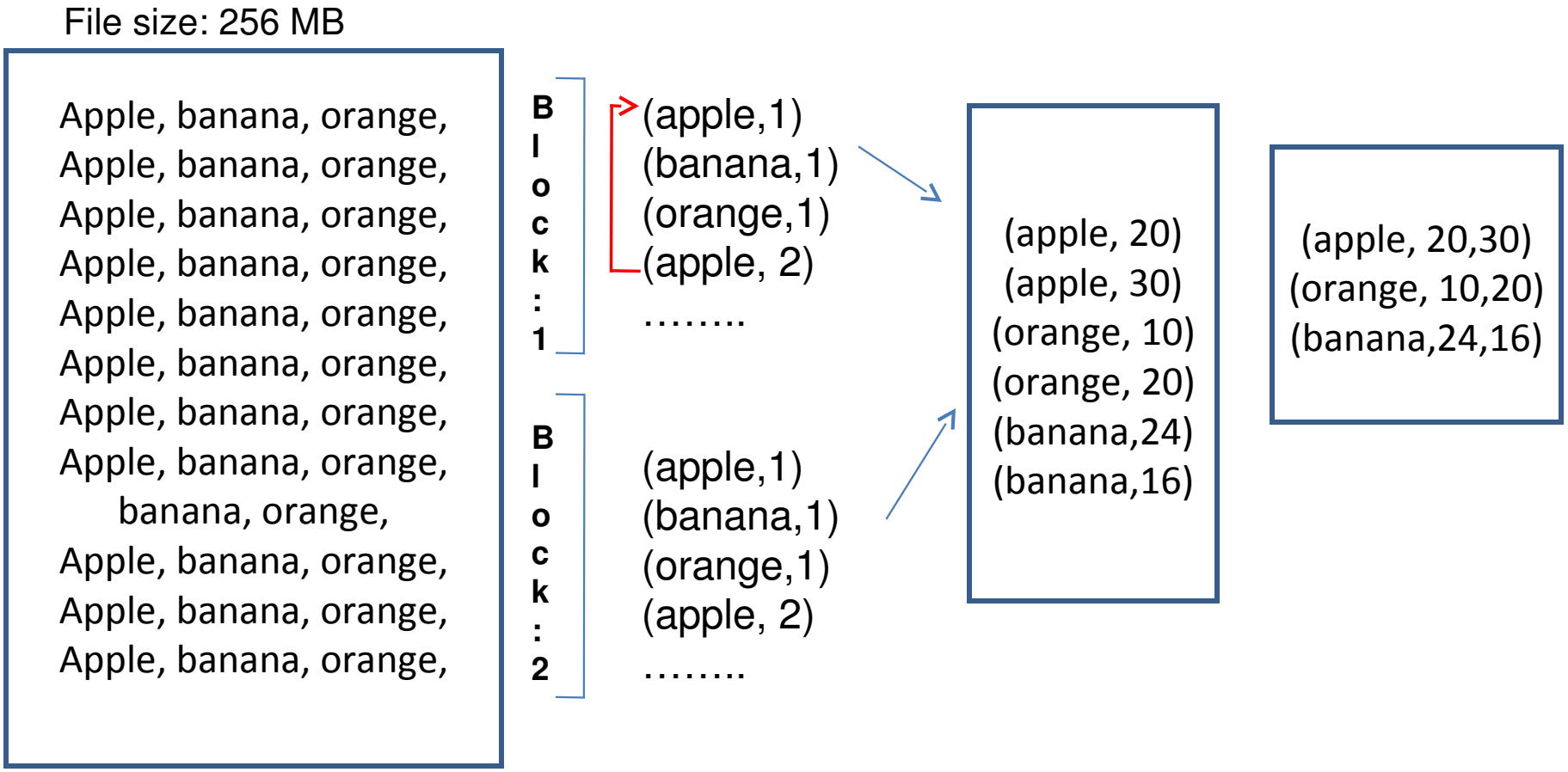


Advantages of Apache Spark

Requirements	Hadoop	Spark
Process data in real-time	No	Yes
Handle input from multiple sources	Yes	Yes
Easy to use	No	Yes
Faster Processing	No	Yes

Performance Bottleneck in WC Example

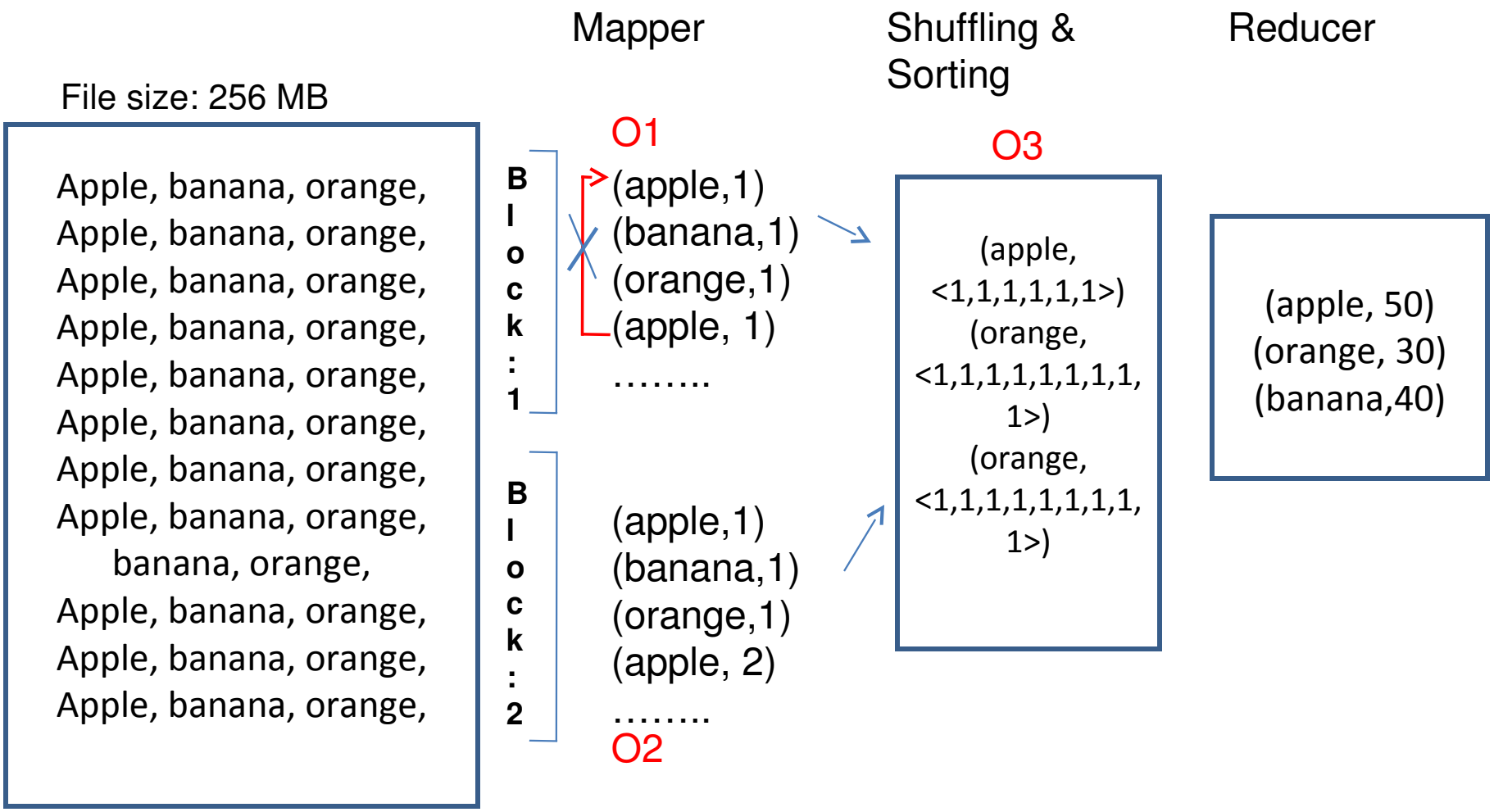
- Consider example of word count. Suppose we have a file having following content. We want to solve it without using mapreduce. Consider default block size is 128 MB.



Performance Bottleneck in WC Example

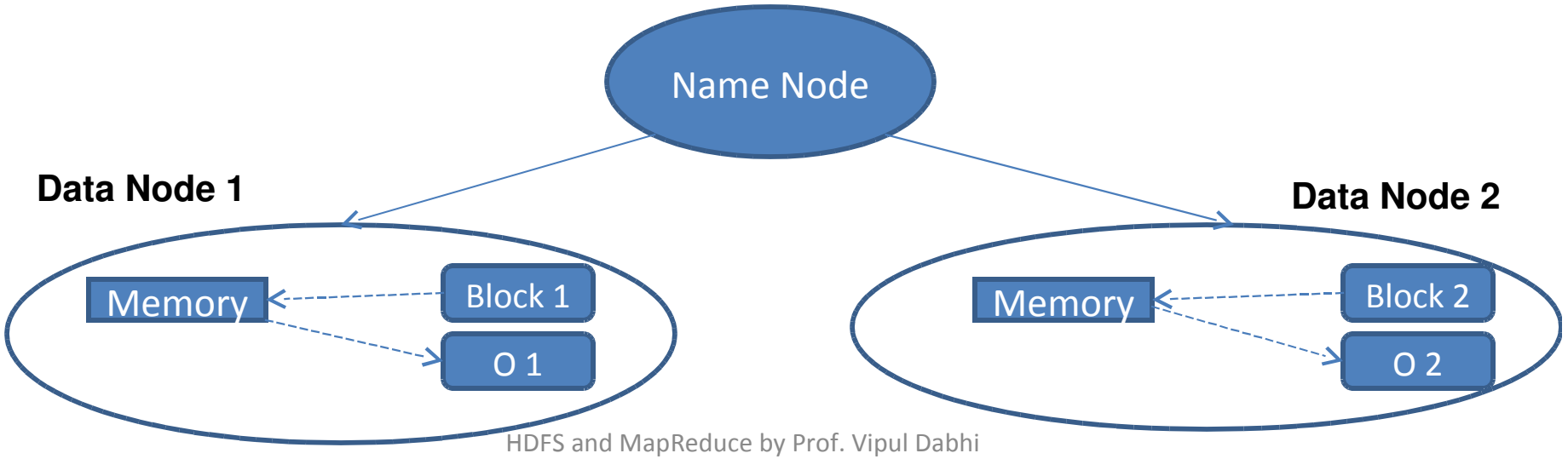
- When we are doing the first step, don't you think that we are decreasing the performance?
  - Because every time the element is coming, you are going and checking back whether that element has occurred before or not and secondly adding up 1 to the previous number.

How Map Reduce has overcome the problem?



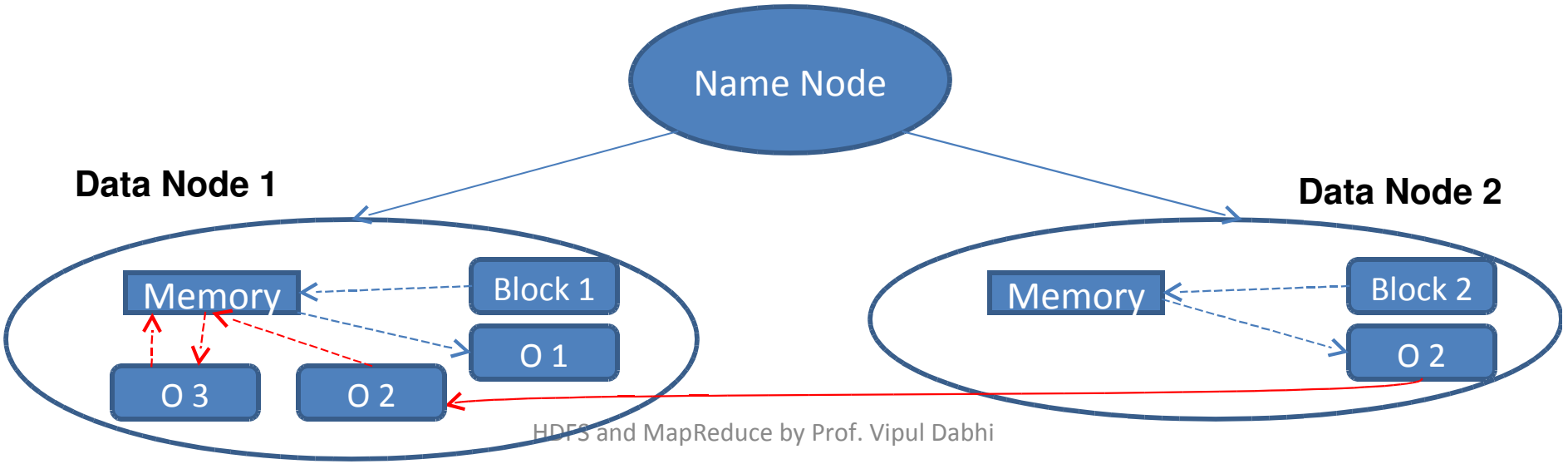
Why Map-Reduce Program is Slower?

- **Where do you perform the processing?**
  - **At Hard-disk level or at memory-level? At Memory-level.**
- **First, mapper code will run. When mapper code run on DN1, the block 1 will move out from disk to memory of DN 1. Similar for block 2 on DN2. We assume that we have memory of 128 MB on data nodes. You will get error in case of map-reduce if memory is less than 128 MB. But Spark can take care (overcome) of this situation.**
- **Whenever there is input/output operation (to/from disk), it degrades the performance because of disk seek etc..**
- **The output of mappers (O1 and O2) will be given back to disk. Again output operation.**

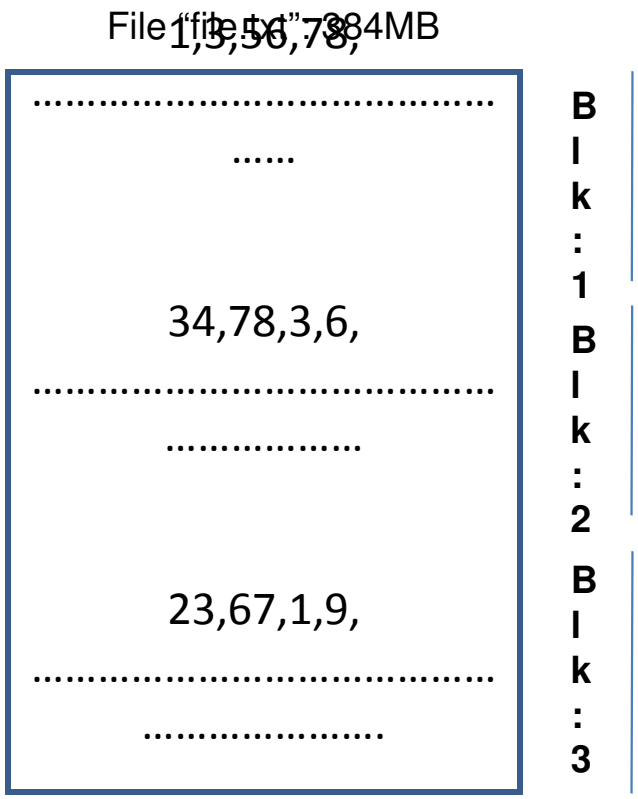


Why Map-Reduce Program is Slower?

- **Now, shuffle and sort will happen on one data node. Data is coming from multiple data nodes to Data Node 1. So, data will get networked transfer from DN2 to DN1.**
- **O2 will send to memory of DN1 and O3 (output of shuffle and sort phase) will get save to disk.**
- **After that reducer code will executed. Reducer code will bring O3 in the memory. Final output will get stored into disk.**
- **There are so many input-output operations happen in one program. Reason for map-reduce program are slower in nature.**



How Apache Spark solves the problem?

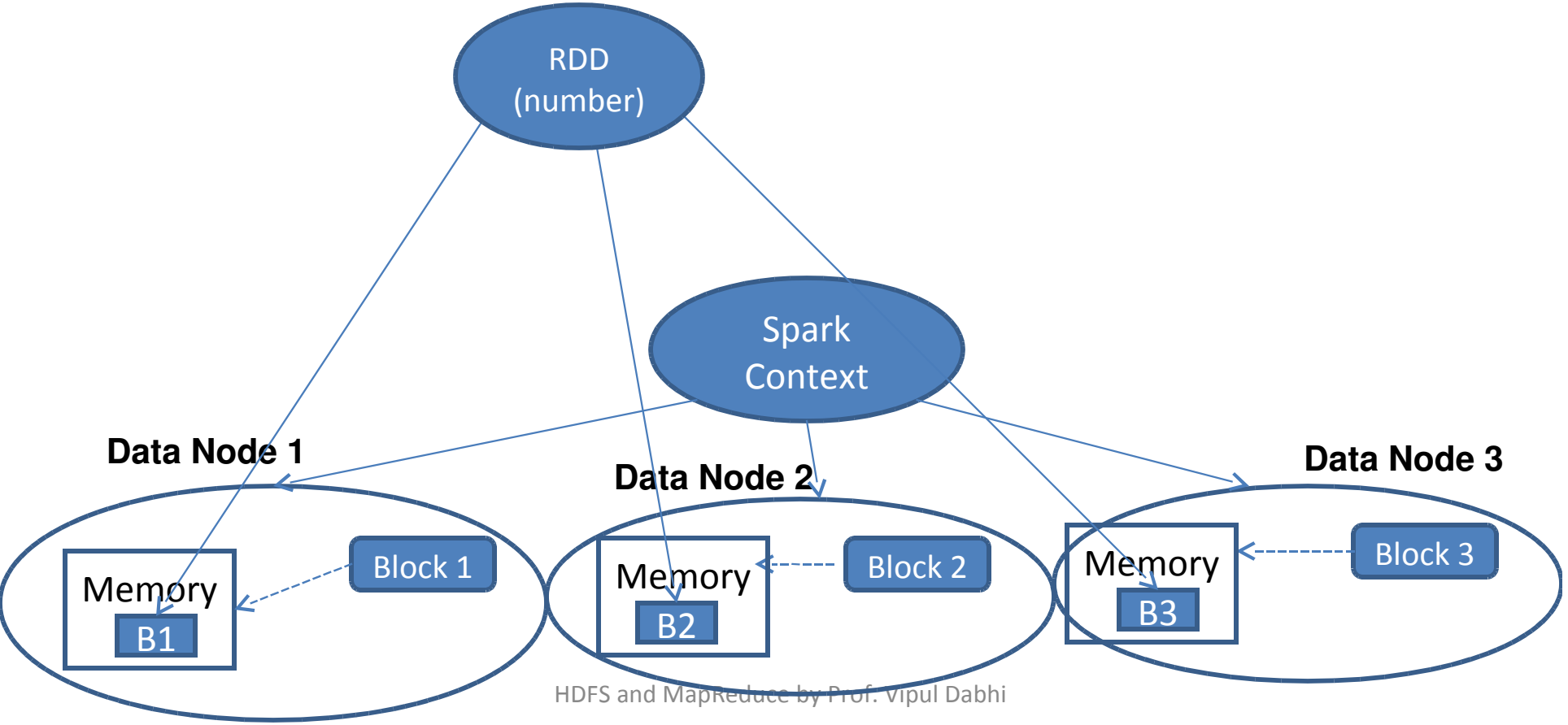


- RDD number = sc.textFile("file.txt")
- RDD filter1 = number.map("logic to find values < 10")

• Consider that we have file.txt residing in HDFS.
• As we have "main" function (entry point for program execution) in JAVA, we have Spark Context (SC) in Spark. The SC resides on Master node. SC will be Separate for each individual application.
• The textFile() function will locate file.txt file and load That file in memory of machines. If file.txt resides on 3 machines (DN), then blocks will be copied on corresponding machines (DNs).
• Whatever code you have written inside map() will get Executed.

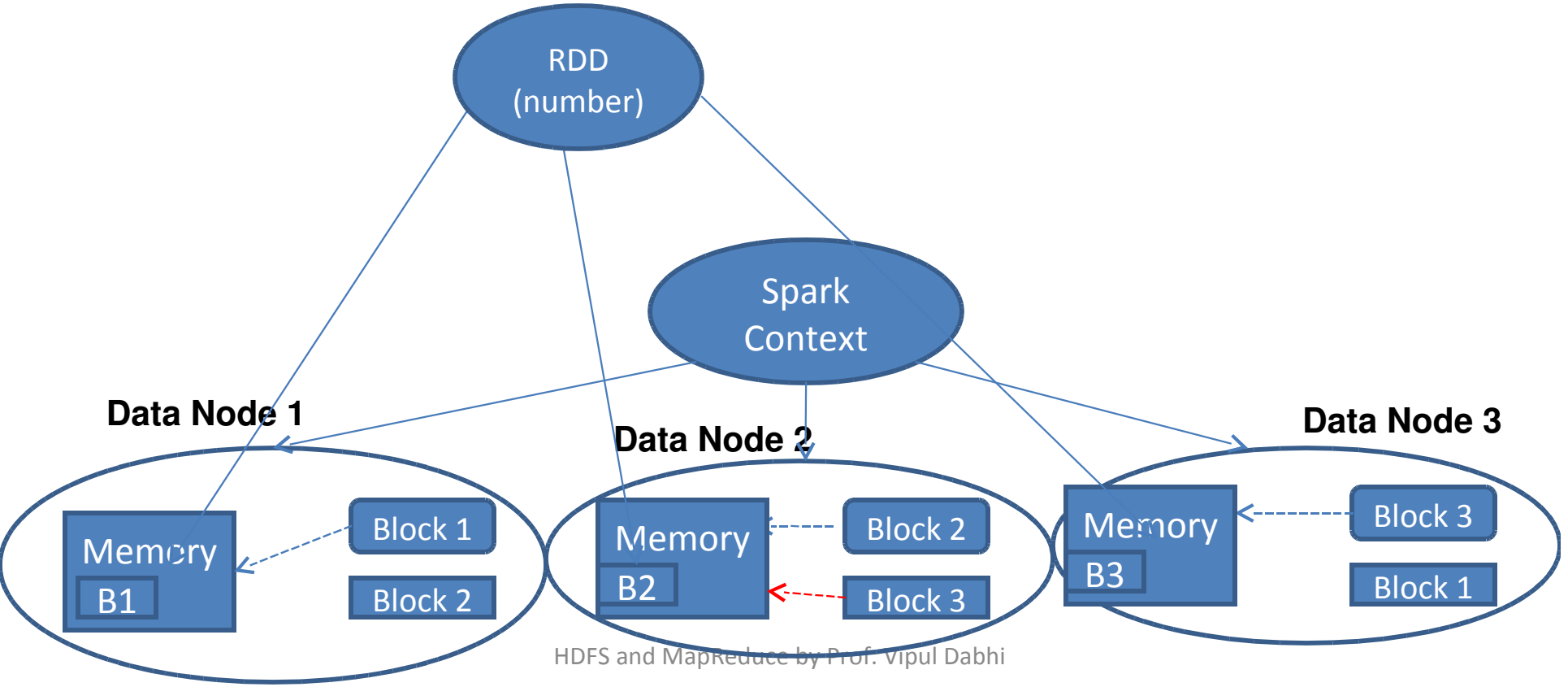
How Apache Spark solves the problem?

RDD (Resilient Distributed Data) is a distributed data sitting in memory of data nodes.
Resilient means Reliable. RDD are always immutable (will not be able to change any Block).
Memory (RAM) is volatile. As soon as we restart DN, we will lose data.
How can we say that it is reliable?

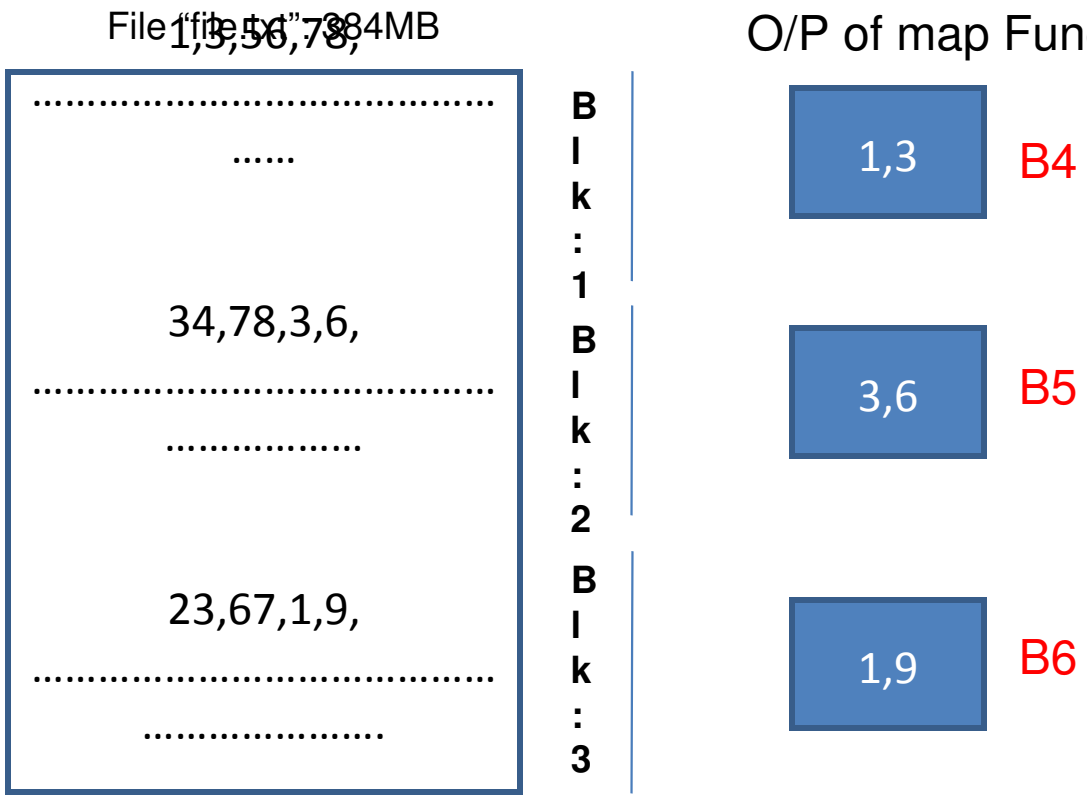


How Apache Spark solves the problem?

Let us say Replication Factor is “2”.
Suppose DN3 gets failed then we lost Block 3, as this block was loaded in memory on DN3.
Block 3 resides on disk of DN2, so it will immediately load Block 3 into memory of DN2. So B2 and B3 both will start residing together in DN2.
A new RDD will be created.



How Apache Spark solves the problem?



- The map() function will get executed on Block 1 and output will be available in new Block 4. Similarly, from Block 2, map() will generate Block 5 and so on.
- Both Blocks (Block 1 and Block 4) will start residing in memory on DN 1.
- Collectively all three blocks (B4, B5 and B6) will be again called an RDD. Which will be referred as "Filter 1" RDD.

How Apache Spark solves the problem?

- Are we doing many Input/Output operations just like in Map-Reduce?
  - No. Only input/output operation happens at first stage (when we execute sc.textFile()). After that out data is always sitting in memory. Reason for giving you faster output.

References

- Apache Hadoop Homepage
  - http://hadoop.apache.org/
- HDFS Documentation
  - http://hadoop.apache.org/hdfs/docs/current/index.html
- Hadoop API
  - http://hadoop.apache.org/core/docs/current/api
- HDFS Design
  - http://hadoop.apache.org/core/docs/current/hdfs_design.html
- Map/Reduce Tutorial
  - http://hadoop.apache.org/common/docs/current/mapred_tutorial.html
- Hadoop-MapReduce in Java
  - http://lucene.apache.org/hadoop
- http://code.google.com/edu/parallel/mapreduce-tutorial.html
- MapReduce Flow Chart
  - https://www.youtube.com/watch?v=6OemZEJdMp8
- http://labs.google.com/papers/mapreduce.html

Thank You