DEPARTMENT OF MCA

I YEAR II SEM

23CAE717 – Cloud Computing

UNIT IV – PROGRAMMING MODEL

Topic 22: MapReduce , Twister and Iterative MapReduce

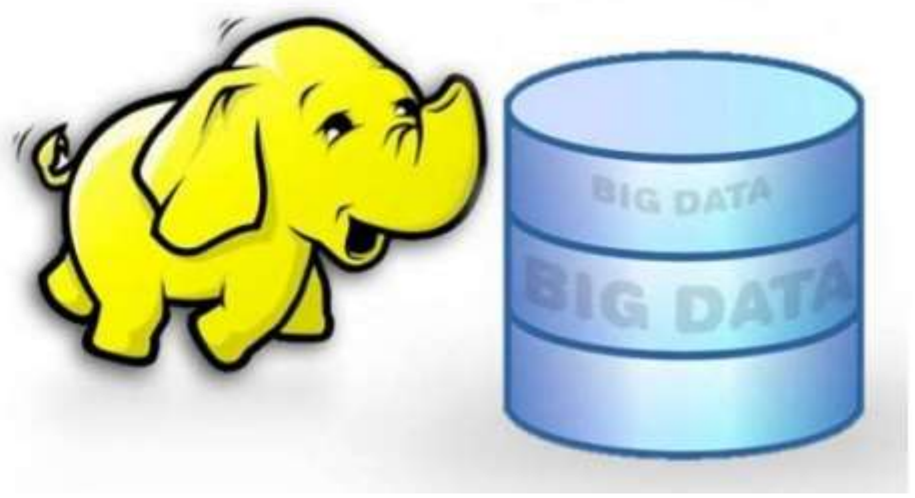
Solution for distributed system?

hadoop
Map Reduce

DRYAD

HADOOP

- ❑ A Software framework allows for distributed processing of large data set across cluster of commodity computers using a simple programming model

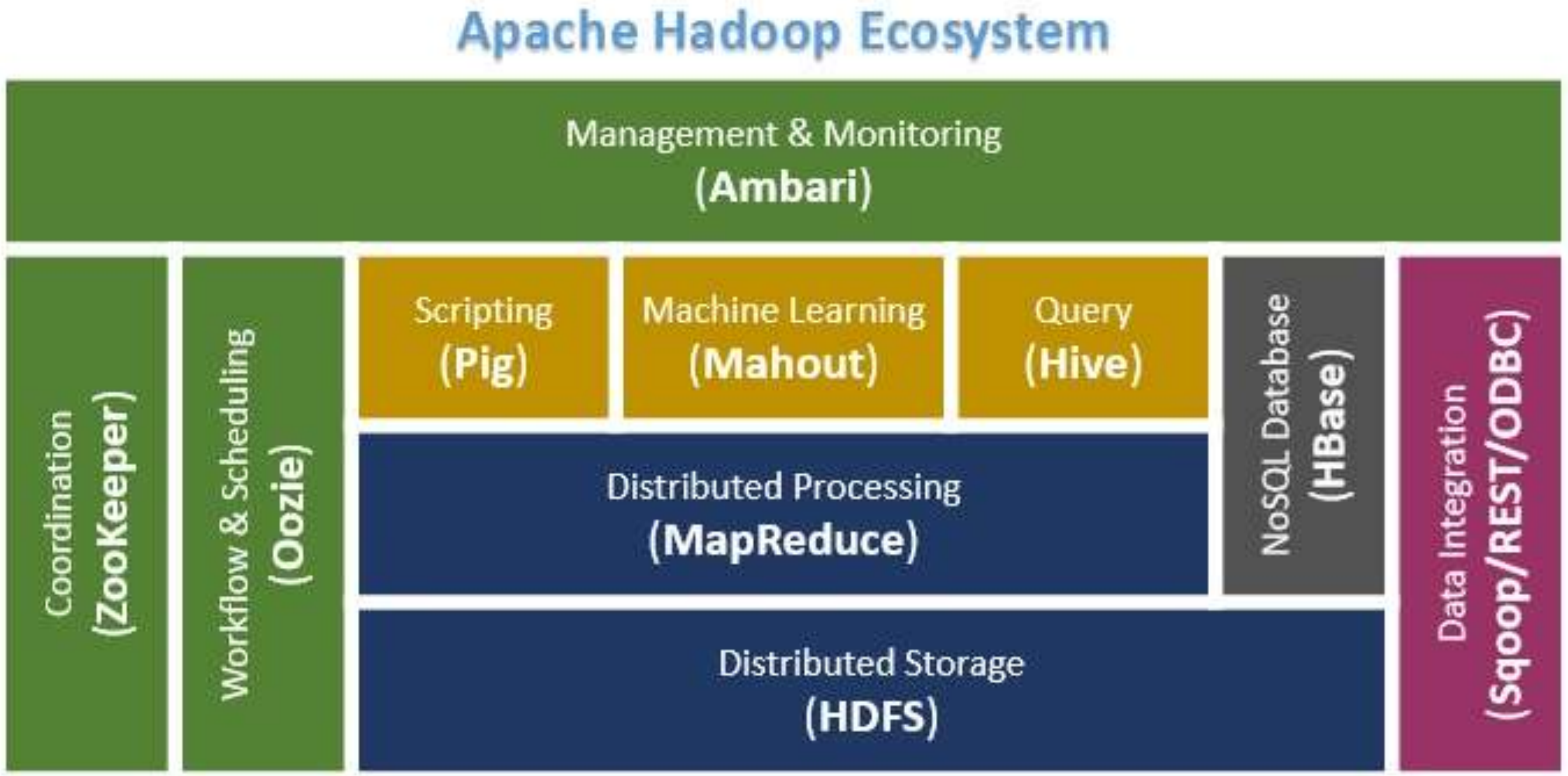


- ❑ Open source implementation by
- ❑ Apache foundation
- ❑ Written in java
- ❑ Contains massive storage and processing power

HADOOP

- ❑ Created by Doug Cutting and Mike Carafella in 2005
- ❑ Cutting named the program after his son's toy elephant
- ❑ USES
  - Data-intensive text processing
  - Graph mining
  - Machine learning and data mining
  - Large scale social network analysis

HADOOP ARCHITECTURE



HADOOP Ecosystem

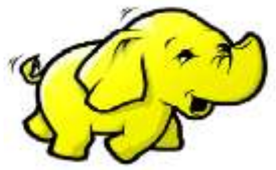
Hadoop Common	• Contains Libraries and other modules
HDFS	• Hadoop Distributed File System • Write once, read-many access model
Hadoop YARN	• Yet Another Resource Negotiator
Hadoop MapReduce	• A programming model for large scale data processing

MapReduce

- Programming model for writing distributed applications devised at Google for processing large amount of data
- Large cluster of commodity hardware in a reliable, flexible manner

MAP REDUCE

- Software framework which supports parallel and distributed computing on large data sets
- abstracts the data flow of running a parallel program on a distributed computing system by providing two functions: Map and Reduce
- hides the implementation of all data flow steps such as data partitioning, mapping, synchronization, communication, and scheduling
- two main functions (Map and Reduce) can be overridden by the user to achieve specific objectives



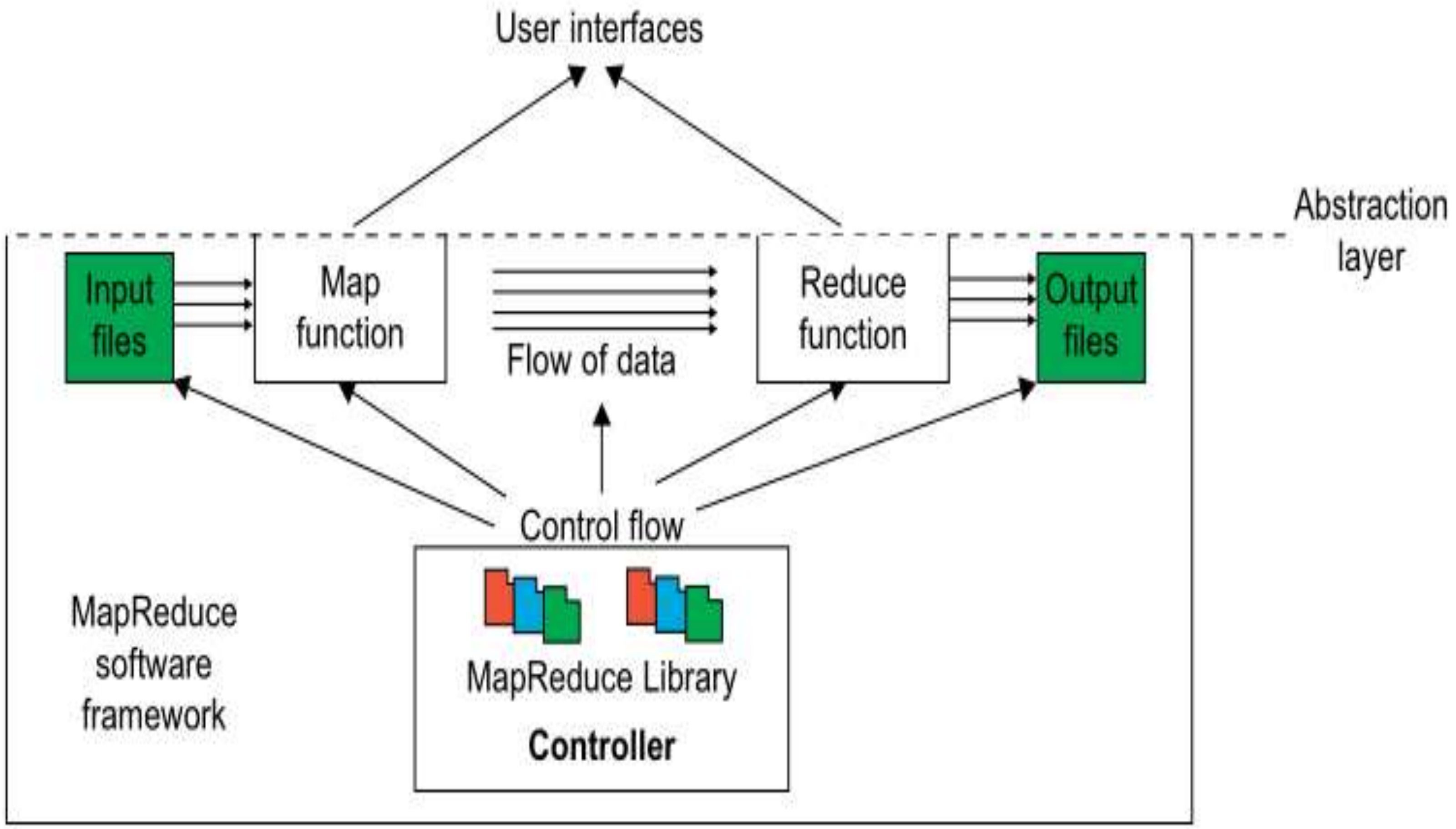
HADOOP High level view

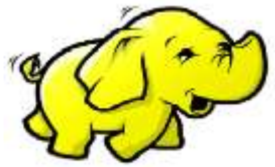
- ❑ When data is loaded onto the system it is divided into blocks
  - Typically 64MB or 128MB
- ❑ Tasks are divided into two phases
  - Map tasks which are done on small portions of data where the data is stored
  - Reduce tasks which combine data to produce the final output
- ❑ A master program allocates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

HADOOP 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

DATA & CONTROL FLOW





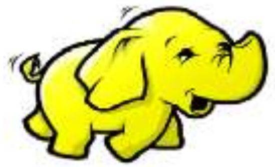
LOGICAL & DATA FLOW

Map function

- Input data is in the form of a (key, value) pair
- The output is called intermediate (key, value) pairs
- Here, the goal is to process all input (key, value) pairs to the Map function in parallel

Reduce function

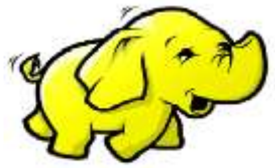
- Reduce function receives the intermediate (key, value) pairs in the form of a group of intermediate values associated with one intermediate key, (key, [set of values])



LOGICAL & DATA FLOW

MapReduce Framework

- Groups by first sorting the intermediate (key, value) pairs and then grouping values with the same key
- Reduce function processes each (key, [set of values]) group and produces a set of (key, value) pairs as output
- User overrides the Map and Reduce functions first, then
- MapReduce (Spec, & Results) function from the library to start the flow of data



MapReduce

1. Input

2. Map

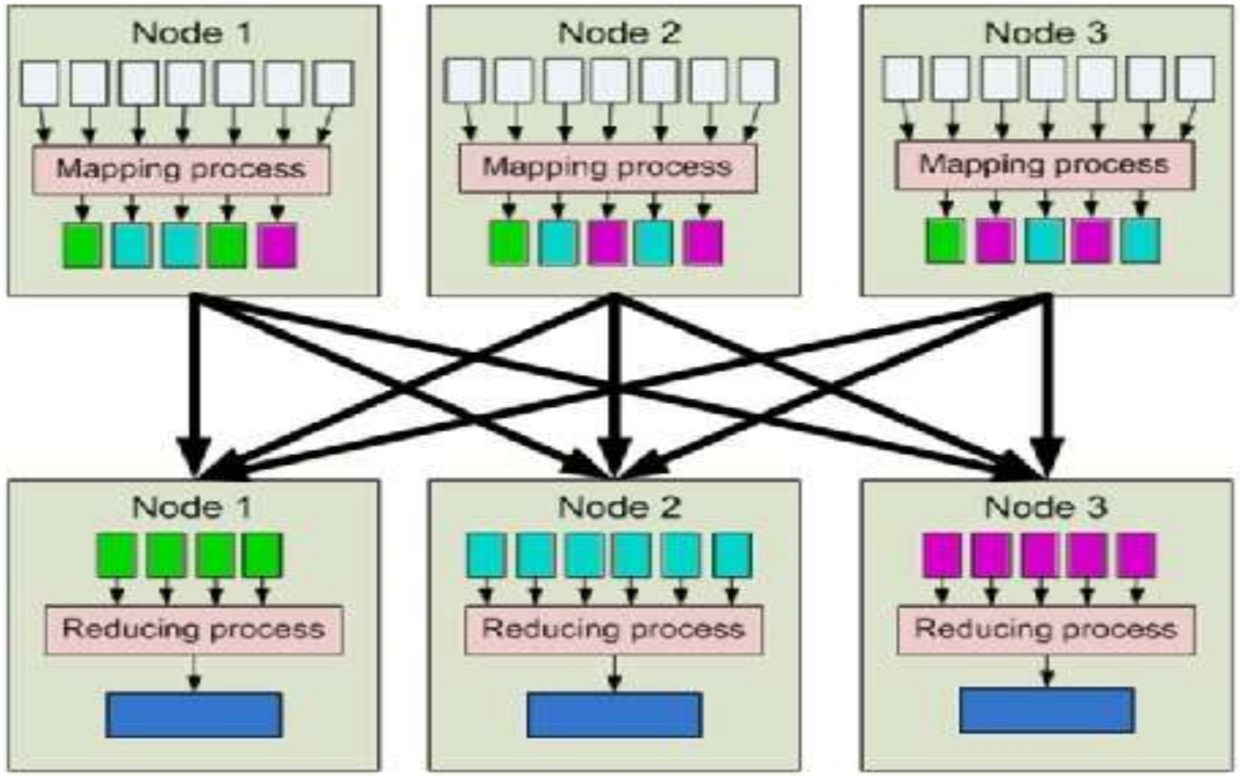
3. Partition [Sort]

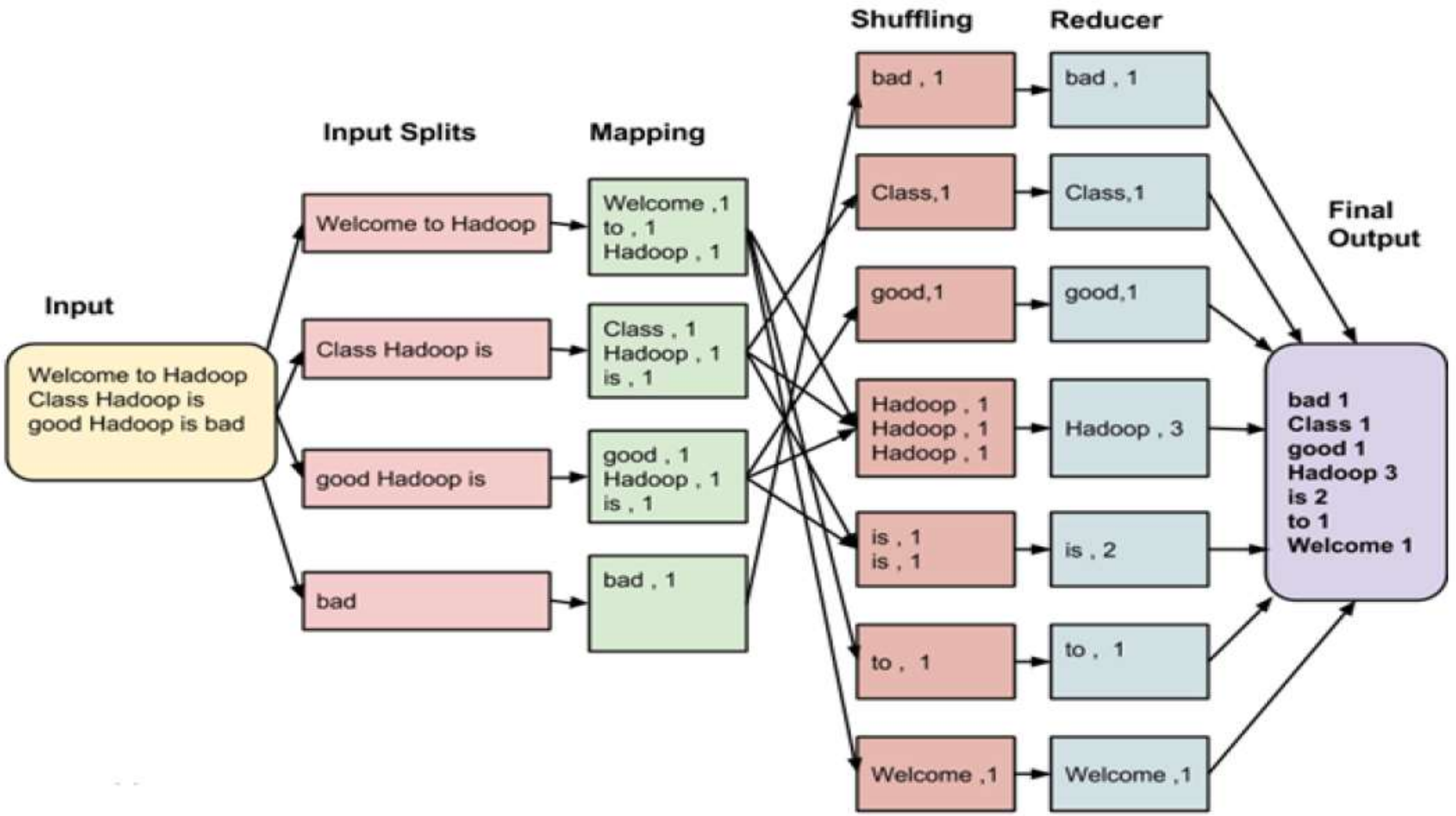
4. Shuffle

5. Merge Sort

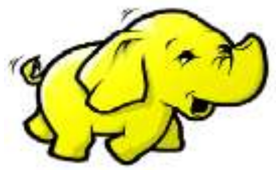
6. Reduce

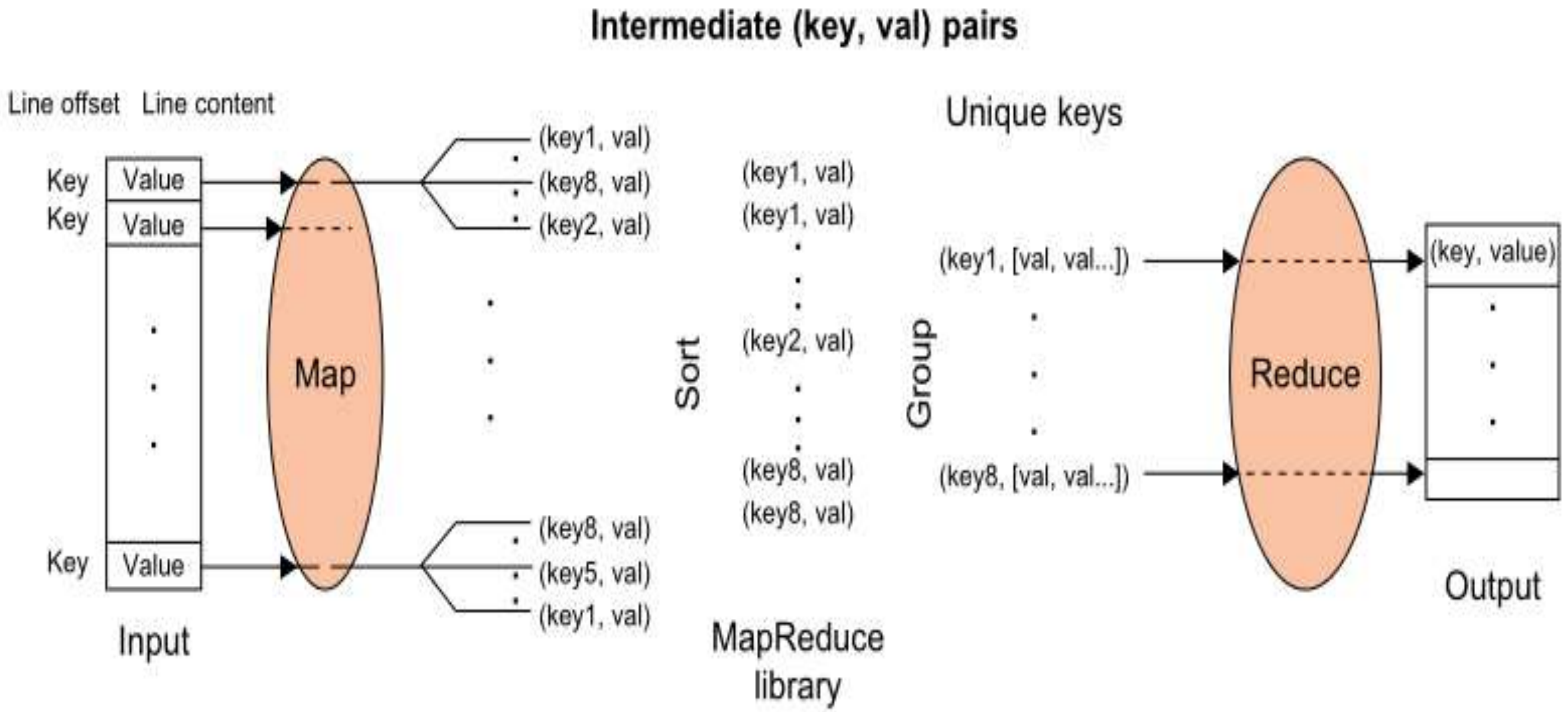
7. Output




Input
Welcome to Hadoop Class Hadoop is good Hadoop is bad
Input Splits
Welcome to Hadoop
Class Hadoop is
good Hadoop is
bad
Mapping
Welcome ,1 to , 1 Hadoop , 1
Class , 1 Hadoop , 1 is , 1
good , 1 Hadoop , 1 is , 1
bad , 1
Shuffling
bad , 1
Class,1
good,1
Hadoop , 1 Hadoop , 1 Hadoop , 1
is , 1 is , 1
to , 1
Welcome ,1
Reducer
bad , 1
Class,1
good,1
Hadoop , 3
is , 2
to , 1
Welcome ,1
Final Output
bad 1 Class 1 good 1 Hadoop 3 is 2 to 1 Welcome 1

LOGICAL & DATA FLOW

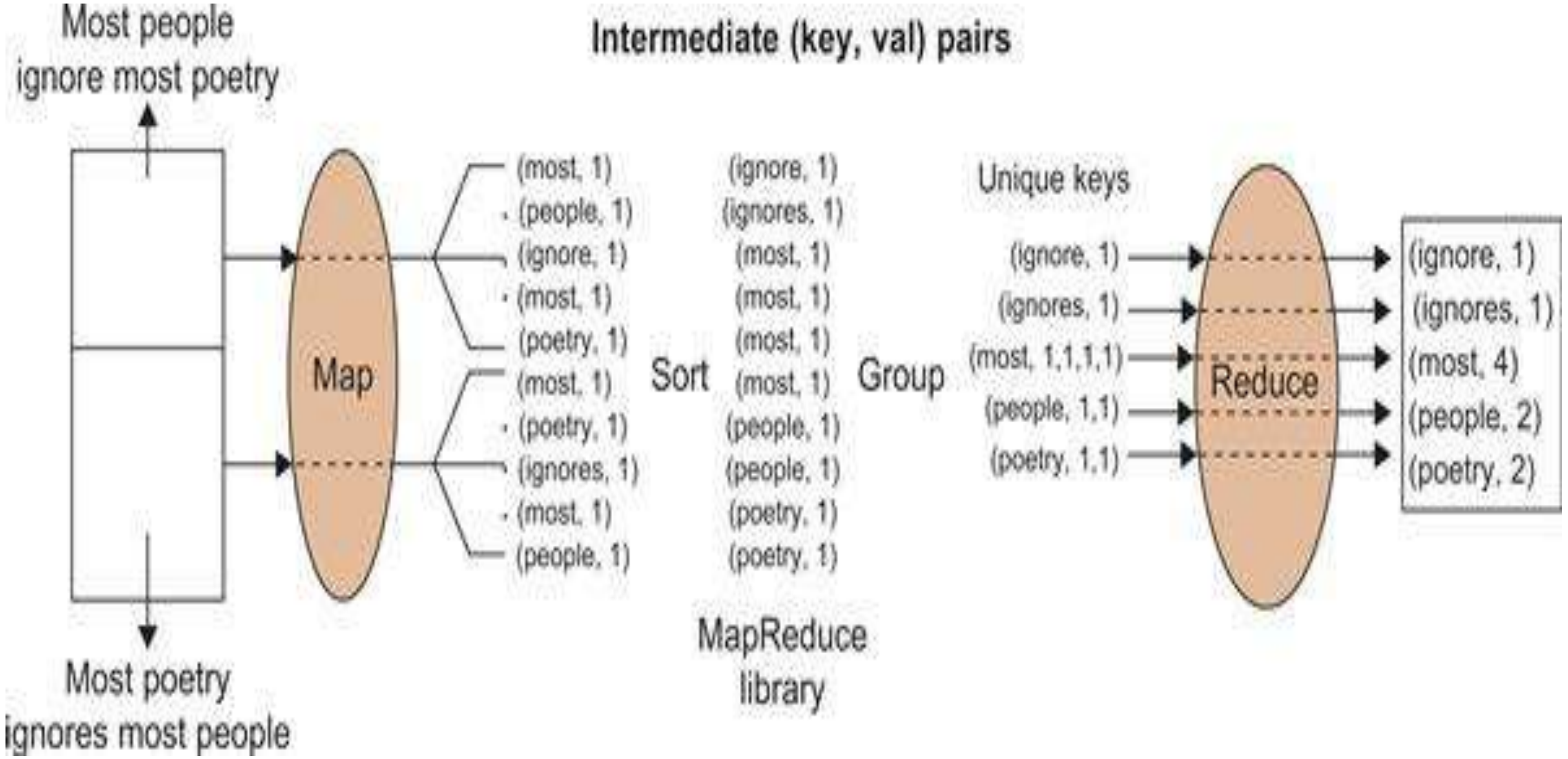




$$(key_1, val_1) \xrightarrow{\textit{Map Function}} \text{List}\,(key_2, val_2)$$

$$(key_2, \text{List}\,(val_2)) \xrightarrow{\textit{Reduce Function}} \text{List}\,(val_2)$$

EXAMPLE


Most people ignore most poetry
Intermediate (key, val) pairs
Map
(most, 1)
(people, 1)
(ignore, 1)
(most, 1)
(poetry, 1)
(most, 1)
(poetry, 1)
(ignores, 1)
(most, 1)
(people, 1)
Sort
(ignore, 1)
(ignores, 1)
(most, 1)
(most, 1)
(most, 1)
(most, 1)
(people, 1)
(people, 1)
(poetry, 1)
(poetry, 1)
Group
Unique keys
(ignore, 1)
(ignores, 1)
(most, 1,1,1,1)
(people, 1,1)
(poetry, 1,1)
Reduce
(ignore, 1)
(ignores, 1)
(most, 4)
(people, 2)
(poetry, 2)
MapReduce library
Most poetry ignores most people

ACTUAL DATA & CONTROL FLOW

Data partitioning: splits the input data into M pieces that also corresponds to the number of map tasks

Computation partitioning: user write program in the form of the Map & Reduce functions. MapReduce library only generates copies of program, distributes and starts them up on a number of available computation engines

Determining the master and workers : one of the copies of the user program becomes the master and the rest become workers. The master picks idle workers, and assigns the map and reduce tasks to them

Reading the input data: Each map worker reads its corresponding input data split, and sends it to its Map function

Map function: receives the input data split and produce the intermediate(key, val) pairs

Combiner function: merges the local data of each map worker before sending it over the network to effectively reduce its communication costs.

Actual Data and control Flow

Partitioning function: intermediate pairs produced by each map worker are partitioned into R regions, equal to the number of reduce tasks, by the Partitioning function to guarantee that all (key, value) pairs with identical keys are stored in the same region

Synchronization : applies a simple synchronization policy to coordinate map workers with reduce workers, so communication between them starts when all map tasks finish

Communication : Reduce worker i, already notified of the location of region i of all map workers, uses a remote procedure call to read the data from the respective region of all map workers

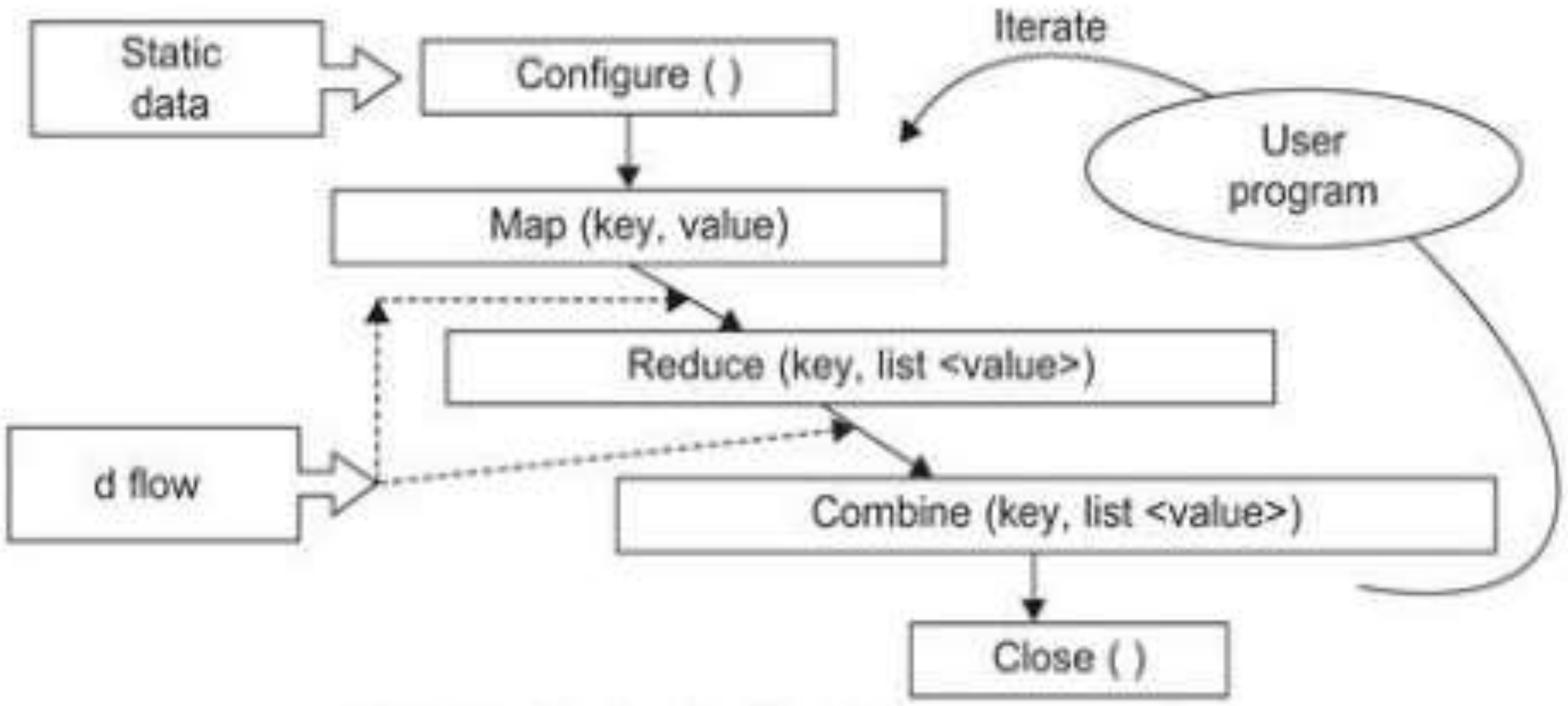
Sorting and Grouping : When reading the input data is finalized by a reduce worker, the data is initially buffered in the local disk of the reduce worker. Then the reduce worker groups intermediate (key, value) pairs by sorting the data based on their keys

Reduce function : reduce worker iterates over the grouped (key, value) pairs, and for each unique key, it sends the key and corresponding values to the Reduce function. Then this function processes its input data and stores the output results in predetermined files in the user's program

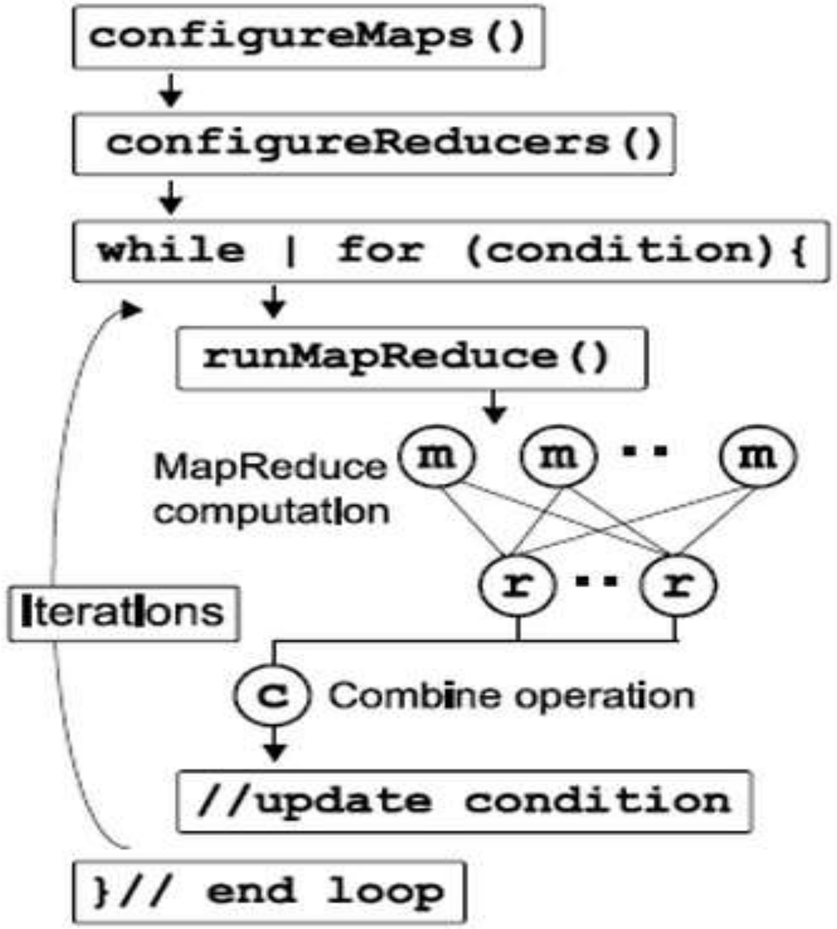
Twister: Iterative MapReduce

- Twister is a lightweight MapReduce runtime
- Features
  - Distinction on static and variable data
  - Configurable long running (cacheable) map/reduce tasks
  - Combine phase to collect all reduce outputs
  - Data access via local disks
  - Lightweight (~5600 lines of Java code)
  - Support for typical MapReduce computations
  - Tools to manage data

Twister: Architecture



Twister: Iterative MapReduce



Who uses Hadoop?

adconion MEDIA GROUP
box
King.com
explorys
SRA INTERNATIONAL, INC.
yp
Experian
TREND MICRO
ADGOOROO
TRION
mobile posse
Huron CONSULTING GROUP
OPOWER
U.S. ARMY
concurrent
aggregate knowledge
SOCIAL SECURITY ADMINISTRATION USA
NetApp
ORBITZ
rapleaf
Chevron
LYRIS
syncsort
DISA
CBS Interactive
tynt
SAMSUNG
BlackBerry
GROUPON
Aol Advertising.
Infinite Campus
Treato the voice of the patient
QUALCOMM
RelayHealth
KLOUT
DELL SecureWorks
ebay
Gravity
APOLLO GROUP
Skybox imaging
NAVTEQ
NATIONAL CANCER INSTITUTE
DATASIFT
rackspace HOSTING

References

1. Kai Hwang, Geoffrey C Fox, Jack G Dongarra, “Distributed and Cloud Computing, From Parallel Processing to the Internet of Things”, Morgan Kaufmann Publishers, 2012.
2. https://subscription.packtpub.com/book