Big Data Engineering

Ardhi Putra Pratama Hartono

Data Mining for Big Data

Pusat Ilmu Komputer Universitas Indonesia

16-20 Juli 2018

Outline

- Motivation
- MapReduce Framework
- Implementing *simple* MapReduce programs
- Hive
- Pig

The Problem

Big data means :

Lots of data, lots of hard drives

The Problem

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

Example: Given a **very large** text data we want to index the words, counting the frequency etc

Parallelization is **difficult**

Parallelization Challenges

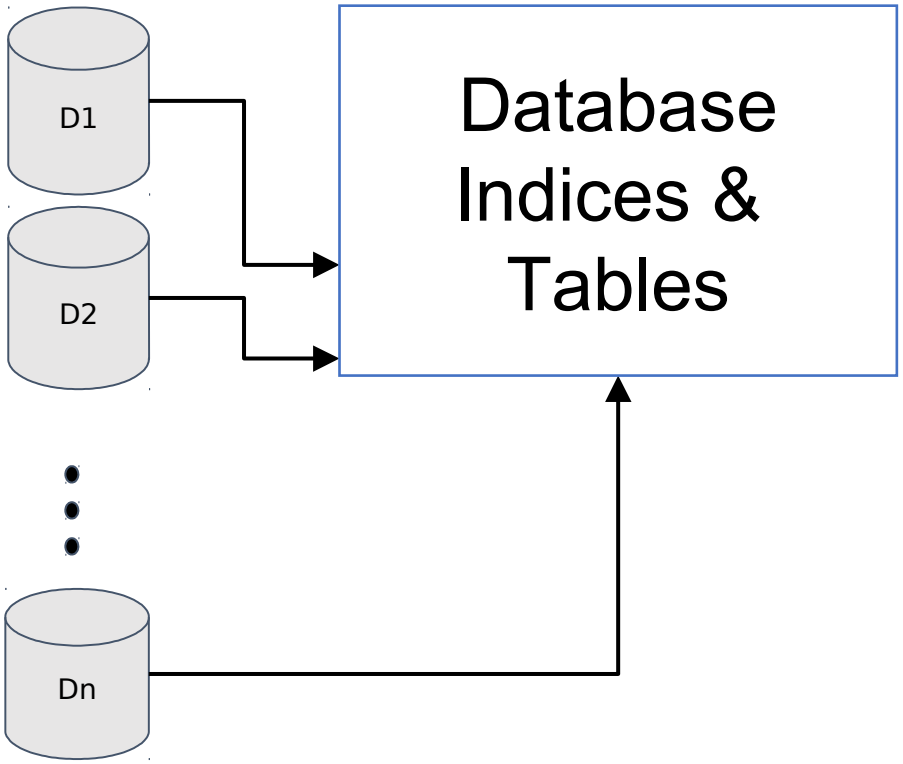
- How do we assign work units to workers?
- What if we have more work units than workers?
- What if workers need to share partial results?
- How do we aggregate partial results?
- How do we know all the workers have finished?
- What if workers die?

The solution

- We should bring computation to data
- Process data sequentially, avoid random access

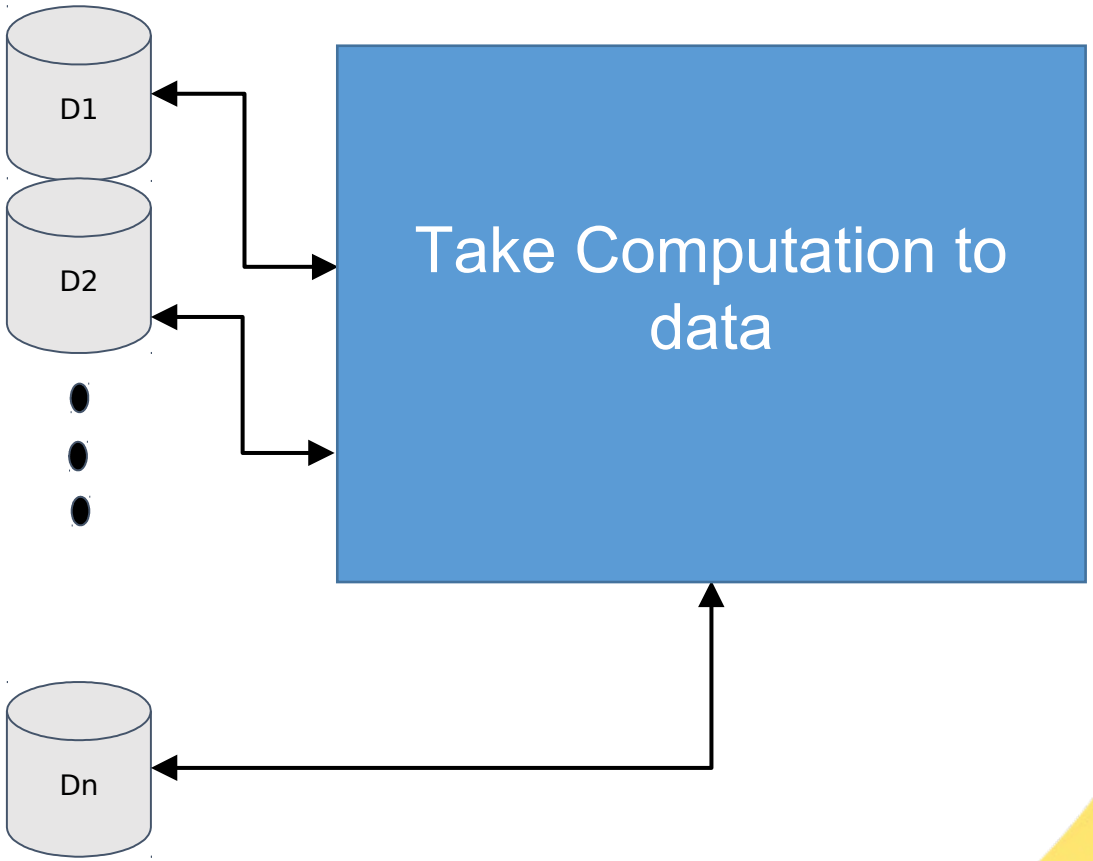
Possibilities when we have “Big Data”

Case 1 : Data needs updating



Possibilities when we have “Big Data”

Case 2 : need to sweep through data so:



The Map-Reduce

Hadoop handles the distribution and execution

MapReduce

- Programming model for Hadoop Ecosystem
- Large-Scale Data Processing
  - Want to use 1000s of CPUs but, don't want the hassle of **managing** things
- Traditional Parallel Programming
  - Require expertise on computing and system concepts (High Learning Curve)
    - Semaphores
    - Threads
    - Shared Memory
  - Incorrect use can crash the program

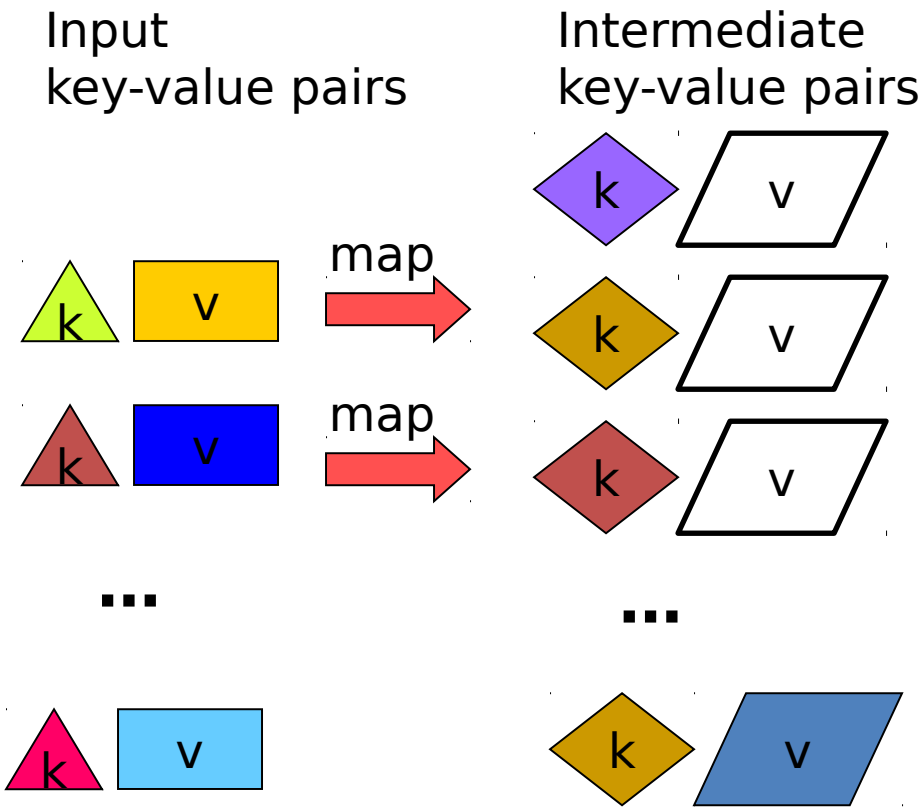
MapReduce Model

- Input & Output: a set of key/value pairs
- Two primitive operations
  - **Map** = apply operation to all elements

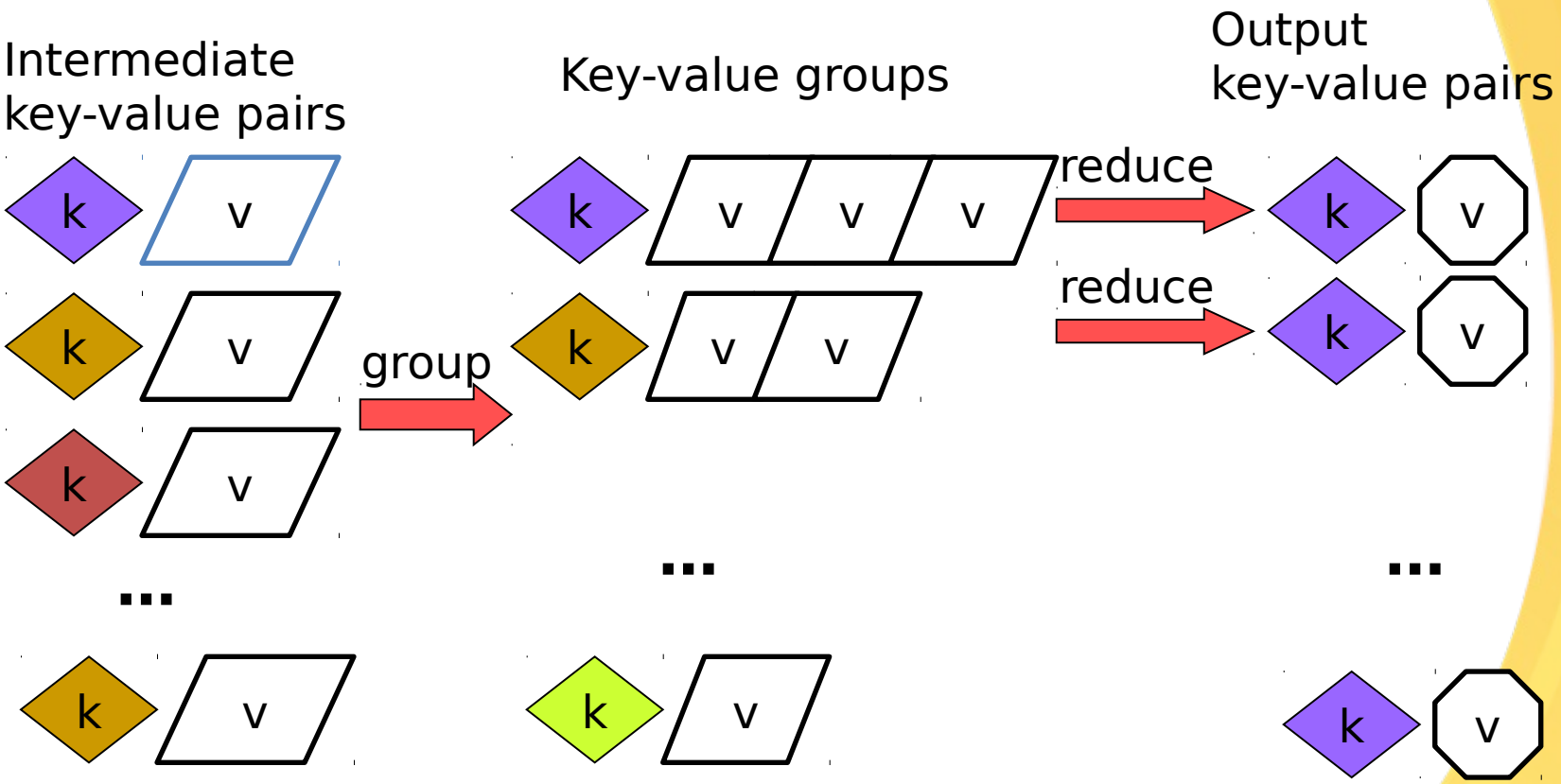
    (k_1, v_1) -> $list(k_2, v_2)$
  - **Reduce** = summarize operation on elements

    $(k_2, list(v_2))$ -> $list(k_3, v_3)$
- **Map** processes one input key/value pair and produces a set of key/value pairs
- **Reduce**
  - Merges all intermediate values for a particular key
  - Produce final key/value pairs
- **Shuffling and Sorting:**
  - Hidden phase between mappers and reducers
  - Groups all similar keys from all mappers, sorts and passes them to a certain reducer in the form of <key, <list of values>>

MapReduce: The Map Step



MapReduce: The Reduce Step



Map Reduce Framework

- User defines:
  - <*key, value*> pair
  - Mapper & Reducer functions
- Hadoop handle the logistics

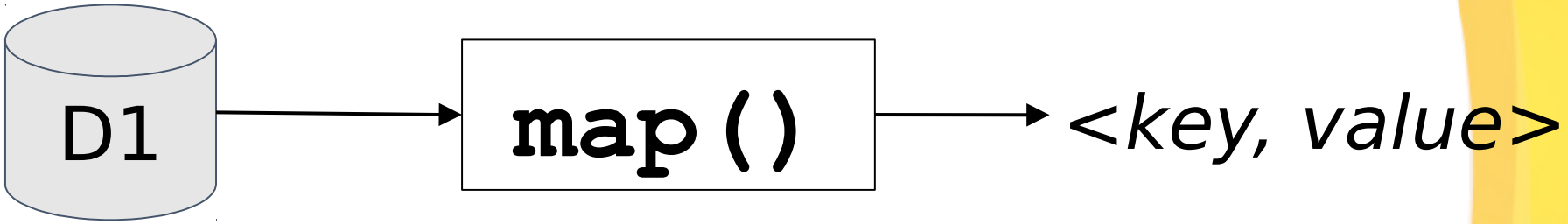
Map Reduce Flow

- User defines a map function

```
map()
```

Map Reduce Flow

- map() reads data and outputs <key, value>



Map/Reduce Flow

- User defines a reduce function

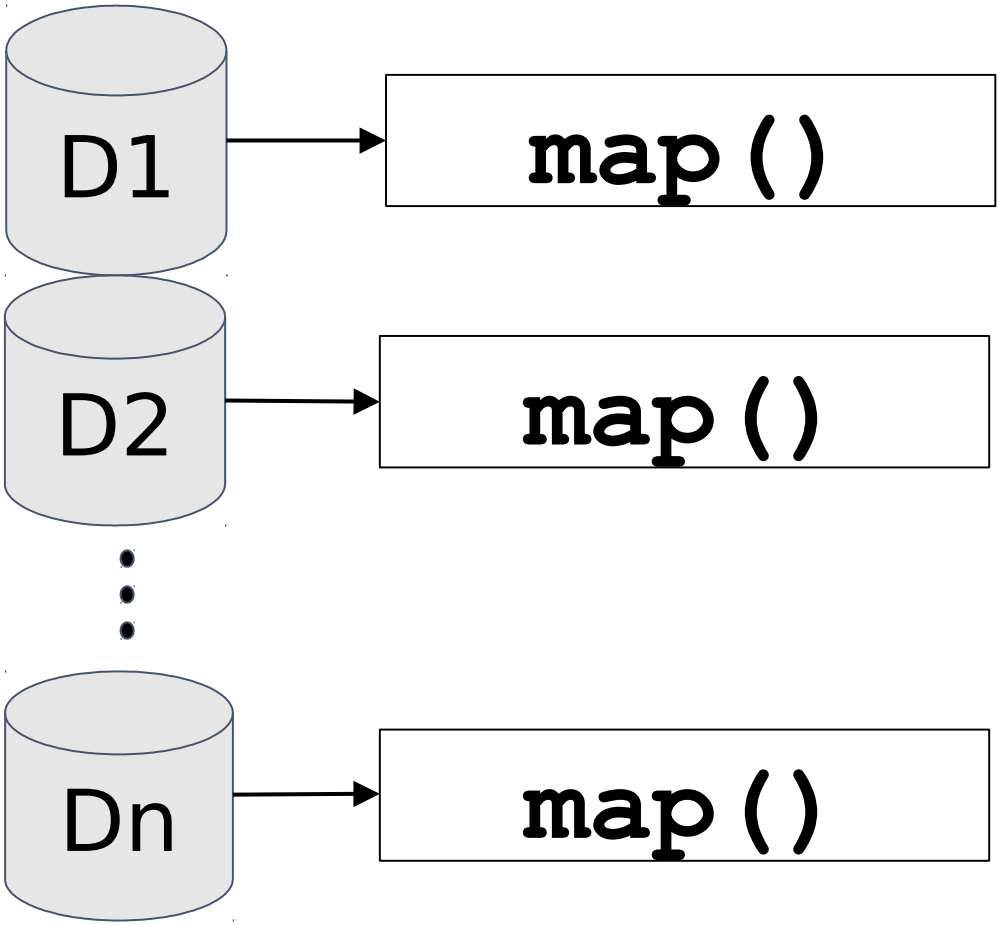
```
reduce()
```

Map/Reduce Flow

- reduce reads <key, value> and outputs your result

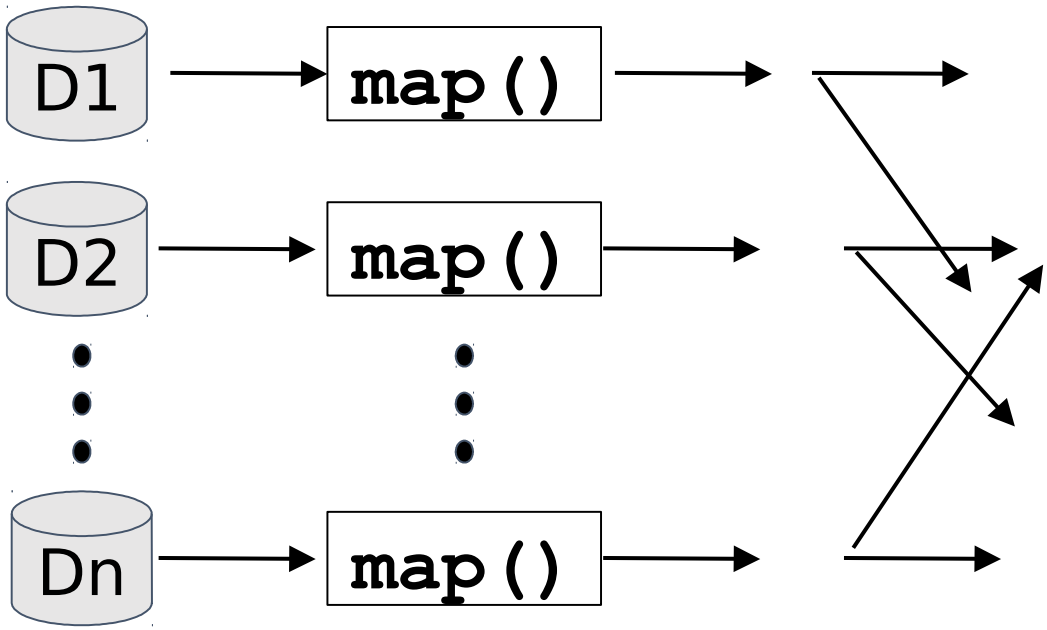
Map/Reduce Flow

- Hadoop distributes map() to data



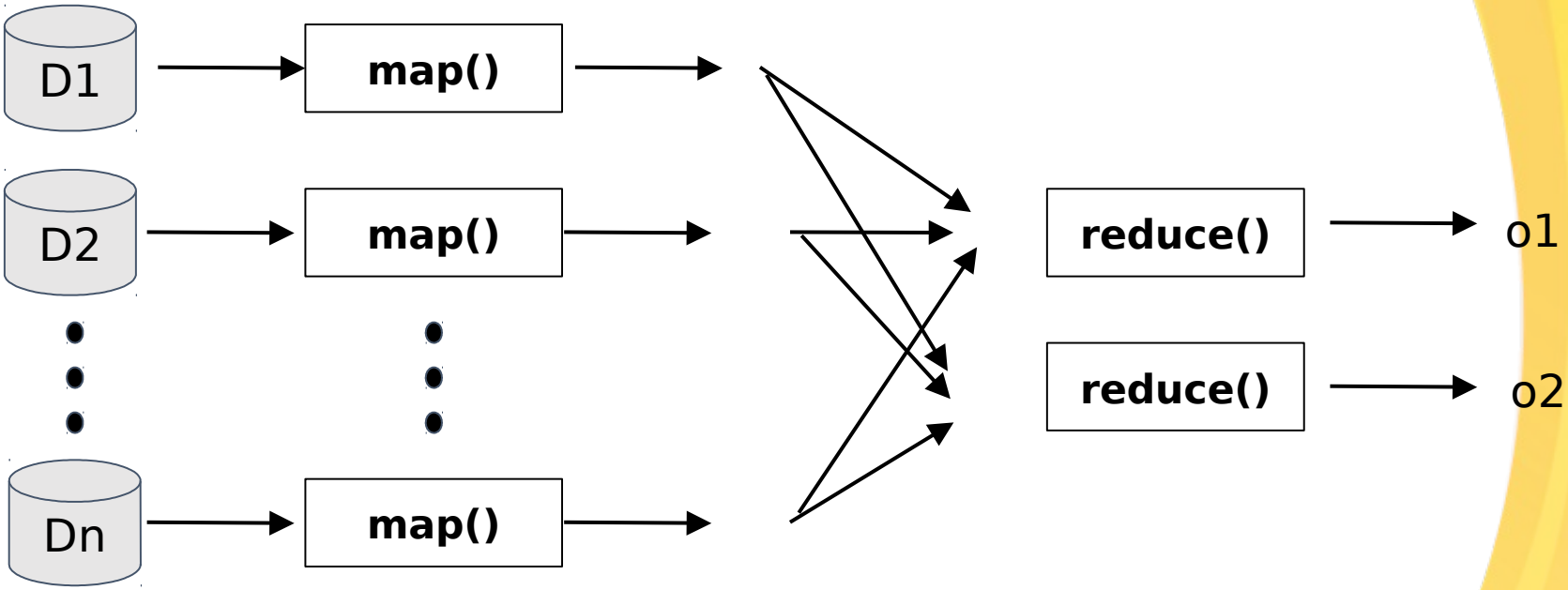
Map/Reduce Flow

- Hadoop groups <key, value> data



Map/Reduce Flow

- Hadoop distributes groups to reducers()



Map/Reduce Example

- “Hello world” : Count word frequencies

Input

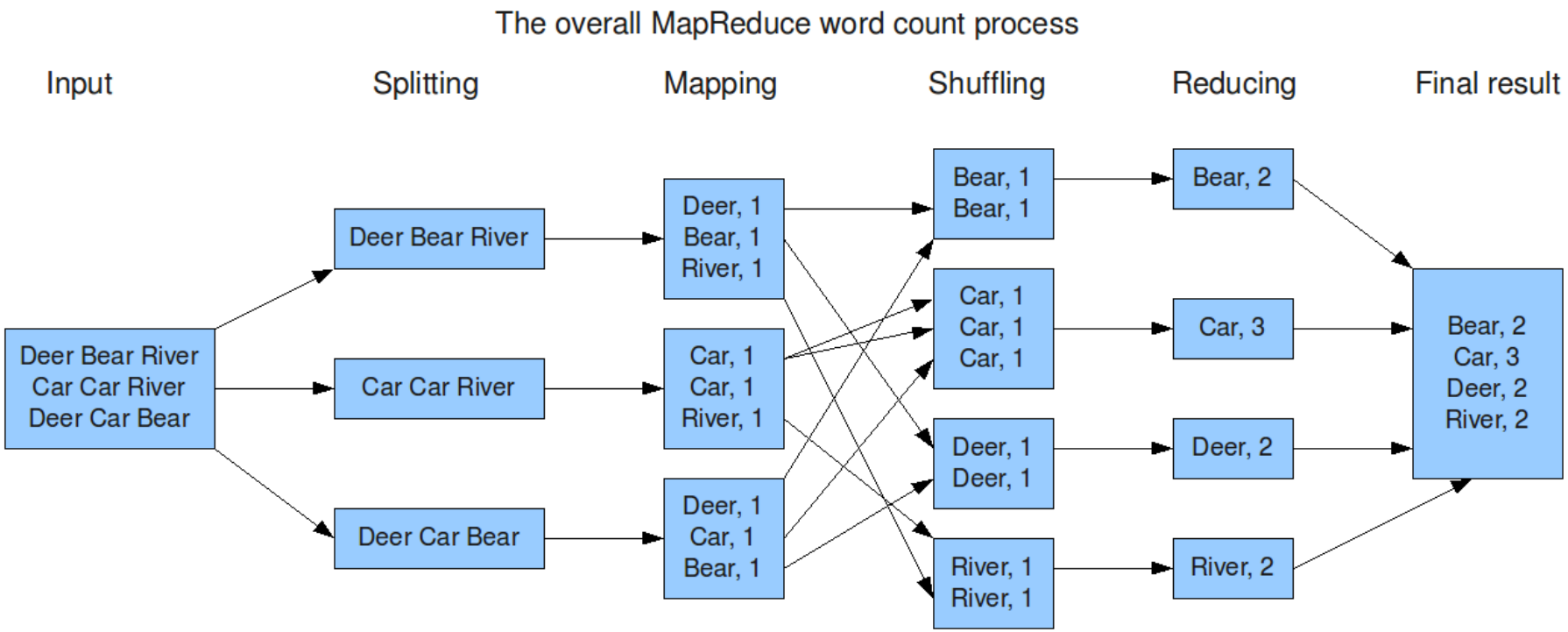
Harry watched Dumbledore striding up and down in front of him, and thought. He thought of his mother, his father and Sirius. He thought of Cedric Diggory. .....

Output

Harry 1
He 2
Watched 1
Dumbledore 1
…
…

Map/Reduce Example : Word Count

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



Word Count serial code

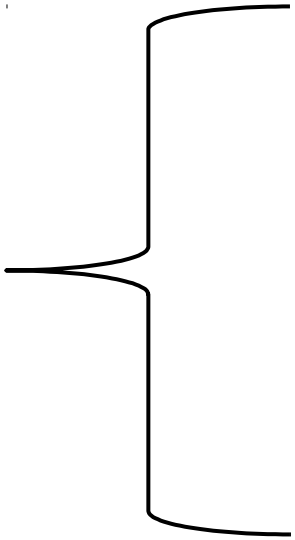
- In a nutshell:
  - Get word
  - Look up word in table (Hashtable etc)
    - **if** the word does not exist
      - Add the word to the table, set count to 1
    - **else** :
      - Add 1 to count

Word Count: Map/Reduce Strategy

- Let <word, 1> be the <key,value>
- Let Hadoop do the *hard work*

Word Count: Map/Reduce Strategy

Loop
Until
Done



1. Get word
2. Emit word

<word, 1>

What one mapper does?

input

He thought of his mother, his father and Sirius

keys

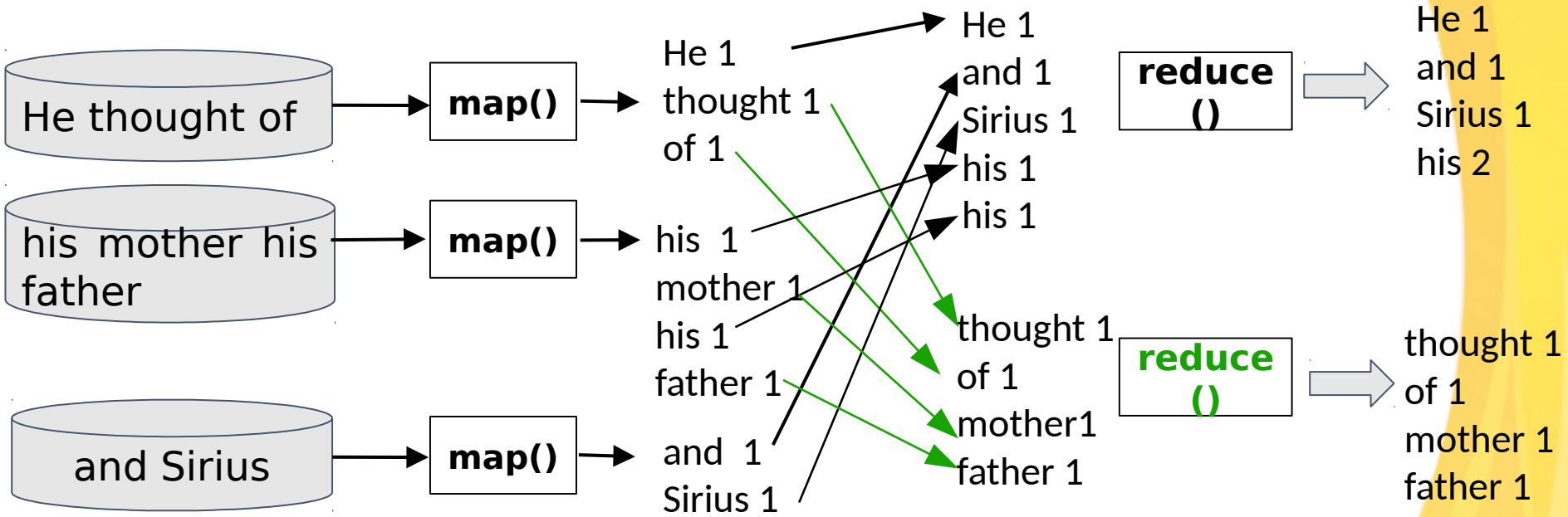
He
thought
of
his
mother
...
...

Emit <key, value>

<He,1>
<thought,1>
<of,1>
<his,1>
<mother,1>
<his,1>
<father, 1>
...

Reducer

Putting it all together



Implementation (Java) - Mapper

```
public void map(Object key, Text value, Context
  context) throws IOException, InterruptedException {
  StringTokenizer itr = new
  StringTokenizer(value.toString());
  IntWritable one = new IntWritable(1);
  Text word = new Text();
  while (itr.hasMoreTokens()) {
     word.set(itr.nextToken());
     context.write(word, one);
  }
}
```

Implementation (Java) - Reducer

```
public void reduce(Text key, Iterable<IntWritable>
  values, Context context) throws IOException,
  InterruptedException {
  int sum = 0;
  IntWritable result = new IntWritable();
  for (IntWritable val : values) {
     sum += val.get();
  }
  result.set(sum);
  context.write(key, result);
}
```

Implementation (Java) – Run It

Prerequisites :

1. **Buat direktori pribadi anda pada local (misal :** /home/bd/mydir/<ardhi>**)**
2. **Buat direktori pribadi anda pada hdfs (misal :** */user/bd/BSSN/<ardhi>***)***
3. ***Salin lirik lagu ke hdfs (lihat contoh)***

```
$ hadoop com.sun.tools.javac.Main WordCount.java
$ jar cf wc.jar WordCount*.class        --> ambil semua .class ke wc.jar
$ hadoop jar wc.jar WordCount inputbalon/
  outputbalonj/
```

Implementation (Python) - Mapper

```
#!/usr/bin/python
  import sys

for line in sys.stdin:
      line = line.strip()
      words = line.split()

      for word in words:
          print '%s\t%s' % (word, 1)
```

Implementation (Python) - Reducer

```
#!/usr/bin/python

    from operator import itemgetter
    import sys

    current_word = None
    current_count = 0
    word = None

    for line in sys.stdin:
        line = line.strip()
        word, count = line.split('\t', 1)

        try:
            count = int(count)
        except ValueError:
            continue

        if current_word == word:
            current_count += count
        else:
            if current_word:
                print '%s\t%s' % (current_word, current_count)
            current_count = count
            current_word = word

    if current_word == word:
        print '%s\t%s' % (current_word, current_count)
```

Implementation (Python) – Run It

```
cat input1.txt | ./mapper.py | sort | ./reducer.py

echo "foo foo quux labs foo bar quux" | ./mapper.py |
  sort | ./reducer.py
```

```
hadoop jar /opt/hadoop-
  2.7.3/share/hadoop/tools/lib/hadoop-streaming-
  2.7.3.jar
     -files mapper.py,reducer.py
     -mapper ./mapper.py
     -reducer ./reducer.py
     -input inputbalon/
     -output outputbalon
```

Don’t forget to prepare the directory!

High-Level Languages

Needs for High-Level Languages

- Hadoop is great for large-data processing!
  - But writing Java programs for everything is verbose and slow
  - Not everyone wants to (or can) write Java code
- Solution: develop higher-level data processing languages
  - Hive: HQL is like SQL
  - Pig: Pig Latin is a bit like Perl

Hive and Pig

- Hive: data warehousing application in Hadoop
  - Query language is HQL, variant of SQL
  - Tables stored on HDFS as flat files
  - Developed by Facebook, now open source
- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Developed by Yahoo!, now open source
  - Roughly 1/3 of all Yahoo! internal jobs
- Common idea:
  - Provide higher-level language to facilitate large-data processing
  - Higher-level language “compiles down” to Hadoop jobs

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

pusilkom ui

Hive

Hive

- Apache Hive is a **data warehouse infrastructure** built on top of **Hadoop** for providing data **summarization, query and analysis**.
- Using Hadoop was not easy for end users, especially for the ones who were not familiar with MapReduce framework. End users had to write map/reduce programs for simple tasks like getting raw counts or averages.
- Hive was created to make it possible for analysts with strong SQL skills (but meager Java programming skills) to run queries on the huge volumes of data to extract patterns and meaningful information..

Hive

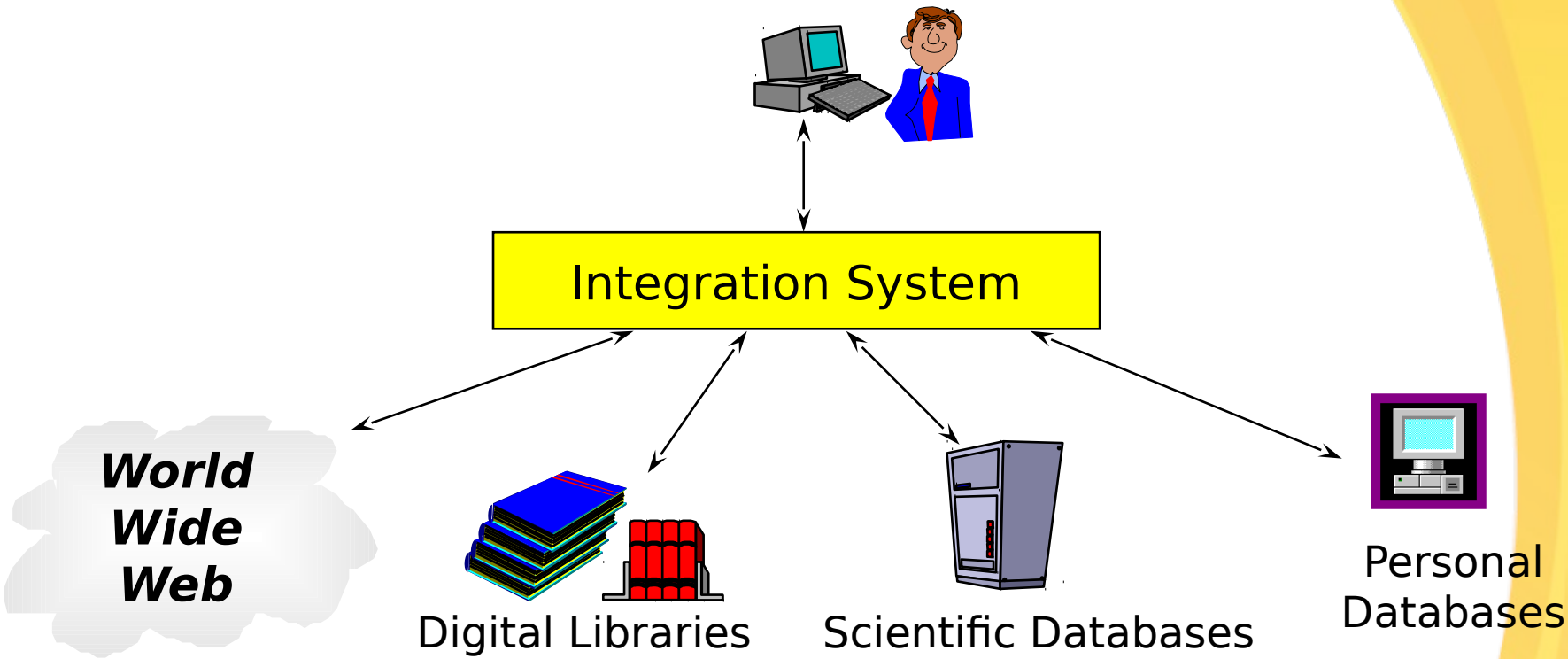
A data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis.

- ETL (Extract Transform Load)
- Structure.
- Access to different storage.
- Query execution via MapReduce.

Key Building Principles:

- SQL is a familiar language
- Extensibility – Types, Functions, Formats, Scripts
- Performance

Goal: Unified Access to Data



- Collects and combines information
- Provides integrated view, uniform user interface
- Supports sharing
- Solution: Use **data warehouse!**

What is a Data Warehouse?
A Practitioners Viewpoint

"A data warehouse is simply a single, complete, and consistent **store of data obtained from a variety of sources** and made **available to end users** in a way they can understand and use it in a business context."

-- **Barry Devlin, IBM Consultant**

Motivation

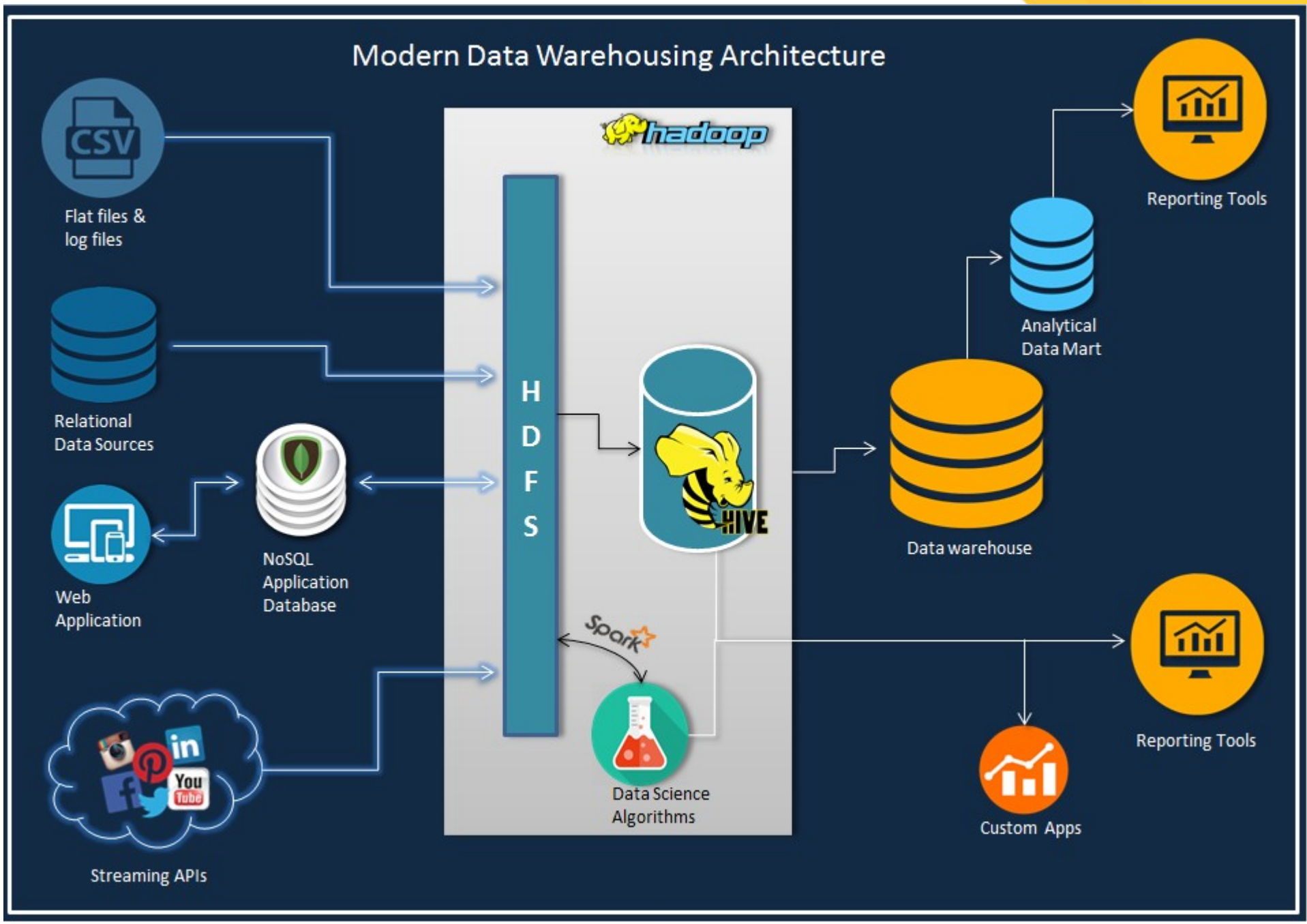
- Analysis of Data made by both engineering and non-engineering people.
- The data are growing fast. In 2007, the volume was 15TB and it grew up to 200TB in 2010.
- Current RDBMS can NOT handle it.
- Current solution are not available, not scalable, Expensive and Proprietary.

Motivation (cont.)

- Hadoop supports data-intensive distributed applications.

However...

  - Map-reduce hard to program (users know sql/bash/python).
  - No schema.


Modern Data Warehousing Architecture
CSV
Flat files &
log files
Relational
Data Sources
Web
Application
NoSQL
Application
Database
Streaming APIs
hadoop
H
D
F
S
HIVE
Spark
Data Science
Algorithms
Analytical
Data Mart
Reporting Tools
Data warehouse
Reporting Tools
Custom Apps

Data Units

- Type
- Databases.
- Tables.

Type System

- Primitive types
  - Integers:TINYINT, SMALLINT, INT, BIGINT.
  - Boolean: BOOLEAN.
  - Floating point numbers: FLOAT, DOUBLE .
  - String: STRING.
- Complex types
  - Structs: {a INT; b INT}.
  - Maps: M['group'].
  - Arrays: ['a', 'b', 'c'], A[1] returns 'b'.

Examples – **Define Data**

- **CREATE TABLE** xx_sample (foo INT, bar STRING)

  - change xx to your name

- **SHOW TABLES** '*e';
- **DESCRIBE** xx_sample;
- **ALTER TABLE** xx_sample **ADD COLUMNS** (new_col INT);
- **DROP TABLE** xx_sample;

Examples – **Manipulate Data**

- **INSERT INTO TABLE** xx_sample **VALUES** (1, 'hello'), (2, 'world');

  Memasukkan data ke HDFS secara otomatis menggunakan MapReduce

- **LOAD DATA LOCAL INPATH** '/home/ardhi/bssn_18/hive1.txt' **OVERWRITE INTO TABLE** xx_sample;

  Need to append in table creation:

  ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';

SELECTS and FILTERS

- **SELECT** bar **FROM** xx_sample **WHERE** foo='5';
- **INSERT OVERWRITE DIRECTORY** '/tmp/hdfs_out' **SELECT** * **FROM** sample **WHERE** foo='4';
- **INSERT OVERWRITE LOCAL DIRECTORY** '/tmp/hive-sample-out' **SELECT** * **FROM** sample;

Aggregations and **Groups**

- **SELECT MAX**(foo) **FROM** sample;
- **SELECT** bar, **COUNT**(*), **SUM**(foo) **FROM** sample **GROUP BY** bar;
- **FROM** sample s **INSERT OVERWRITE TABLE** bar **SELECT** s.bar, count(*) WHERE s.foo > 0 **GROUP BY** s.bar;

JOIN

```
CREATE TABLE customer (id INT,name STRING,address STRING)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY '#';
CREATE TABLE order_cust (id INT,cus_id INT,prod_id INT,price INT)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

- **SELECT** * **FROM** customer c **JOIN** order_cust o **ON** (c.id=o.cus_id);
- **SELECT** c.id,c.name,c.address,ce.exp **FROM** customer c **JOIN (SELECT** cus_id,sum(price) AS exp **FROM** order_cust **GROUP BY** cus_id**)** ce ON (c.id=ce.cus_id);

Built-in Functions

- Mathematical: `round, floor, ceil, rand, exp...`
- Collection: `size, map_keys, map_values, array_contains.`
- Type Conversion: `cast.`
- Date: `from_unixtime, to_date, year, datediff...`
- Conditional: `if, case, coalesce.`
- String: `length, reverse, upper, trim...`

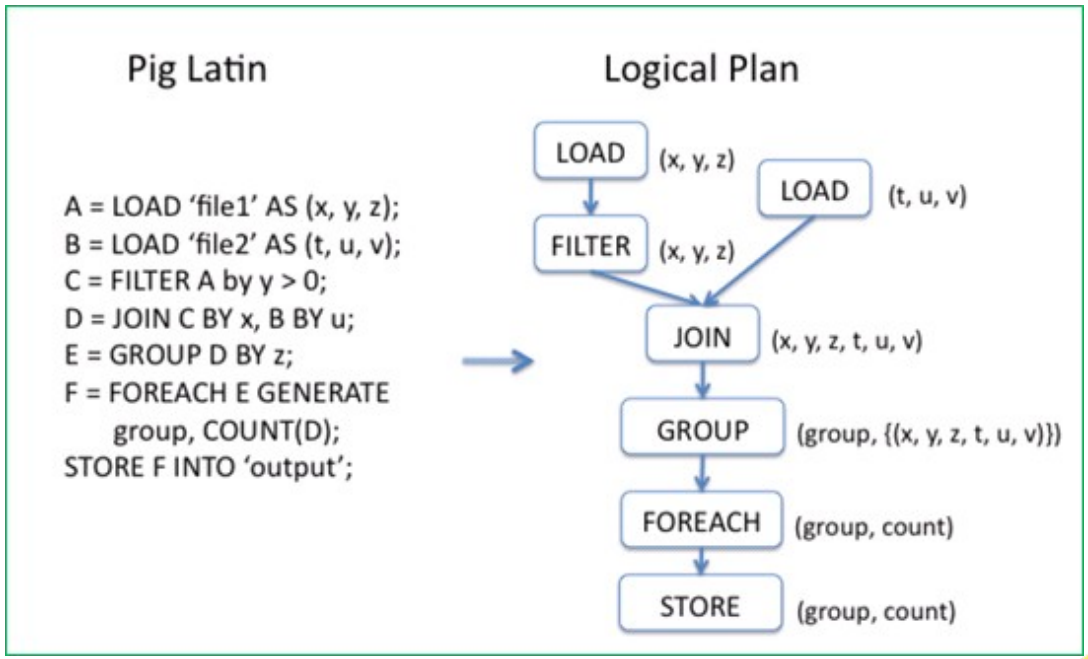
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

pusilkom ui

Pig

What is Pig?

- Framework for analyzing large un-structured and semi-structured data on top of Hadoop.
  - Pig Engine Parses, compiles Pig Latin scripts into MapReduce jobs run on top of Hadoop.
  - Pig Latin is declarative, SQL-like language; the high level language interface for Hadoop.



Motivation of Using Pig

- Faster development
  - Fewer lines of code (Writing map reduce like writing SQL queries)
  - Re-use the code (Pig library, Piggy bank)
- One test: Find the top 5 words with most high frequency
  - 10 lines of Pig Latin V.S 200 lines in Java
  - 15 minutes in Pig Latin V.S 4 hours in Java

Word Count using Java MapReduce

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordCount extends Configured implements Tool {

  public static class MapClass 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 itr = new StringTokenizer(line);
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        output.collect(word, one);
      }
    }
  }

  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));
    }
  }
```

```
  public int run(String[] args) throws Exception {
    JobConf conf = new JobConf(getConf(), WordCount.class);
    conf.setJobName("wordcount");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(MapClass.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    List<String> other_args = new ArrayList<String>();
    for(int i=0; i < args.length; ++i) {
      try {
        if ("-m".equals(args[i])) {
          conf.setNumMapTasks(Integer.parseInt(args[++i]));
        } else if ("-r".equals(args[i])) {
          conf.setNumReduceTasks(Integer.parseInt(args[++i]));
        } else {
          other_args.add(args[i]);
        }
      } catch (NumberFormatException except) {
        System.out.println("ERROR: Integer expected instead of " + args[i]);
        return printUsage();
      } catch (ArrayIndexOutOfBoundsException except) {
        System.out.println("ERROR: Required parameter missing from " +
                           args[i-1]);
        return printUsage();
      }
    }
    // Make sure there are exactly 2 parameters left.
    if (other_args.size() != 2) {
      System.out.println("ERROR: Wrong number of parameters: " +
                         other_args.size() + " instead of 2.");
      return printUsage();
    }
    FileInputFormat.setInputPaths(conf, other_args.get(0));
    FileOutputFormat.setOutputPath(conf, new Path(other_args.get(1)));
    JobClient.runJob(conf);
    return 0;
  }

  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(), new WordCount(), args);
    System.exit(res);
  }
}
```

Word Count using Pig

```
Lines=LOAD ‘input/hadoop.log’ AS (line: chararray);
Words = FOREACH Lines GENERATE  FLATTEN(TOKENIZE(line)) AS word;
Groups = GROUP Words BY word;
Counts = FOREACH Groups GENERATE  group, COUNT(Words);
Results = ORDER Words BY Counts DESC;
Top5 = LIMIT Results 5;
STORE Top5 INTO /output/top5words;
```

Who uses Pig for What

- 70% of production jobs at Yahoo (10ks per day)
- Twitter, LinkedIn, Ebay, AOL,…
- Used to
  - Process web logs
  - Build user behavior models
  - Process images
  - Build maps of the web
  - Do research on large data sets

Pig Hands-on

1. Accessing Pig
2. Basic Pig knowledge: (Word Count)
    1. Pig Data Types
    2. Pig Operations
    3. How to run Pig Scripts

Accessing Pig

- Accessing approaches:
  - Batch mode: submit a script directly
  - Interactive mode: Grunt, the pig shell
  - PigServer Java class, a JDBC like interface
- Execution mode:
  - Local mode: pig –x local
  - Mapreduce mode: pig –x mapreduce

Pig Data Types

- **Scalar Types:**
  - –Int, long, float, double, boolean, null, chararray, bytearry;
- **Complex Types**: fields, tuples, bags, relations;
  - –A Field is a piece of data
  - –A Tuple is an ordered set of fields
  - –A Bag is a collection of tuples
  - –A Relation is a bag

- Samples:
  - –Tuple -> Row in Database

```
( 0002576169, Tome, 20,  4.0)
```

  - –Bag -> Table or View in Database

```
{(0002576169 , Tome,  20,  4.0),
 (0002576170, Mike, 20, 3.6),
 (0002576171 Lucy, 19, 4.0), …. }
```

Pig Operations

- Loading data
  - **LOAD** loads input data

```
Lines=LOAD 'input/access.log' AS (line: chararray);
```

- Projection
  - takes a set of expressions and applies them to every record.

```
FOREACH … GENERATE … (similar to SELECT)
```

- Grouping
  - **GROUP** collects together records with the same key
- Dump/Store
  - **DUMP** displays results to screen, **STORE** save results to file system
- Aggregation
  - **AVG, COUNT, MAX, MIN, SUM**

Pig Operations - Foreach

- Foreach ... Generate
  - The `Foreach … Generate` statement iterates over the members of a bag

```
studentid = FOREACH  students  GENERATE  studentid, name;
```

  - The result of a Foreach is another bag
  - Elements are named as in the input bag

Pig Operations – Positional Reference

- Fields are referred to by positional notation or by name (alias).

```
> students = LOAD 'student.txt' AS (name:chararray, age:int,
gpa:float);
> DUMP A;
(John,18,4.0F)
(Mary,19,3.8F)
(Bill,20,3.9F)
> studentname = Foreach students Generate $1 as studentname;
```

	First Field	Second Field	Third Field
Data Type	chararray	int	float
Position notation	$0	$1	$2
Name (variable)	name	age	Gpa
Field value	Tom	19	3.9

Pig Operations- Group

- Groups the data in one or more relations
  - The GROUP and COGROUP operators are identical.
  - Both operators work with one or more relations.
  - For readability GROUP is used in statements involving one relation
  - COGROUP is used in statements involving two or more relations. Jointly Group the tuples from A and B.

```
B = GROUP A BY age;
C = COGROUP A BY name, B BY name;
```

Pig Operations – Dump&Store

- DUMP Operator:
  - display output results, will always trigger execution
- STORE Operator:
  - Pig will parse entire script prior to writing for efficiency purposes

```
A = LOAD ‘input/pig/multiquery/A’;
B = FILTER A by $1 == “apple”;
C = FILTER A by $1 == “apple”;
STORE B INTO “output/b”
STORE C INTO “output/c”
```

Pig Operations - Count

- Compute the number of elements in a bag
- Use the COUNT function to compute the number of elements in a bag.
- COUNT requires a preceding GROUP ALL statement for global counts and GROUP BY statement for group counts.

```
X = FOREACH B GENERATE COUNT(A);
```

Pig Operation - Order

- Sorts a relation based on one or more fields
- In Pig, relations are unordered. If you order relation A to produce relation X relations A and X still contain the same elements.

```
student = ORDER students BY gpa DESC;
```

How to run Pig Latin scripts

- Local mode
  - Local host and local file system is used
  - Neither Hadoop nor HDFS is required
  - Useful for prototyping and debugging
- MapReduce mode
  - Run on a Hadoop cluster and HDFS
- Batch mode - run a script directly
  - Pig –x local my_pig_script.pig
  - Pig –x mapreduce my_pig_script.pig
- Interactive mode use the Pig shell to run script
  - Grunt> Lines = LOAD '/input/input.txt' AS (line:chararray);
  - Grunt> Unique = DISTINCT Lines;
  - Grunt> DUMP Unique;

Hands-on: Word Count using Pig Latin

- **Batch mode**
  1. pig –x local wordcount.pig

- **Iterative mode**
  1. grunt> Lines=LOAD ‘input.txt’ AS (line: chararray);
  2. grunt> Words = FOREACH Lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
  3. grunt> Groups = GROUP Words BY word;
  4. grunt> counts = FOREACH Groups GENERATE group, COUNT(Words);
  5. grunt> DUMP counts;

TOKENIZE&FLATTEN

- TOKENIZE returns a new bag for each input; “FLATTEN” eliminates bag nesting
- `A:{line1, line2, line3…}`
- After Tokenize:
  `{{line1word1,line1word2,…}},`
  `{line2word1,line2word2…}}`
- After Flatten
  `{line1word1,line1word2,line2wor`
  `d1…}`

Accessing hive

- **pig -x local -useHCatalog**

- **In .bashrc :**
  1. `export HCAT_HOME="/opt/apache-hive-2.1.0-bin/hcatalog"`
  2. `export PIG_CLASSPATH=/opt/pig-0.16.0/lib/*:$HCAT_HOME/share/hcatalog/*:$HIVE_HOME/lib/*`
  3. `export HADOOP_USER_CLASSPATH_FIRST=true`

- **a = LOAD 'ardhi' USING org.apache.hive.hcatalog.pig.HcatLoader();**
- **describe a;**
- **dump a;**

References

- Tom White. *Hadoop The Definitive Guide 4th ed.* O'Reilly. 2015
- https://www.edureka.co/blog/mapreduce-tutorial/
- Hadoop Big Data Tutorial
- Coursera Big Data
- Hadoop Official Website