

Introduction to Data Science

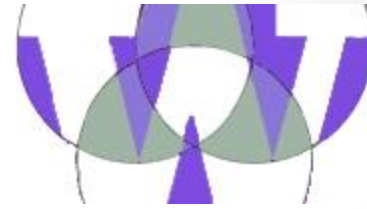
Lecture 8; November 30th, 2016

Ernst Henle

ErnstHe@UW.edu

Skype: ernst-henle

Agenda



- Announcements
 - Marius' after-hours meeting: Today at Applebee's (get directions from Marius)
 - Social Interactions on LinkedIn: Continue to discuss statistics, job prospects, etc.
- Quiz 08a NoSQL Introduction
- Hadoop Intro
- HDFS
- Quiz 08b Hadoop
- HDFS Lab
- Sqoop Lab
- Break
- Hive and Impala Lab
- MapReduce
- Break
- MapReduce Lab
- Predictive Faux Pas
- Assignment. See assignment slides at the end of the deck. Complete all assignments items from all assignment slides. Submit by this Saturday at 11:57 PM.

Quiz 08a NoSQL Introduction

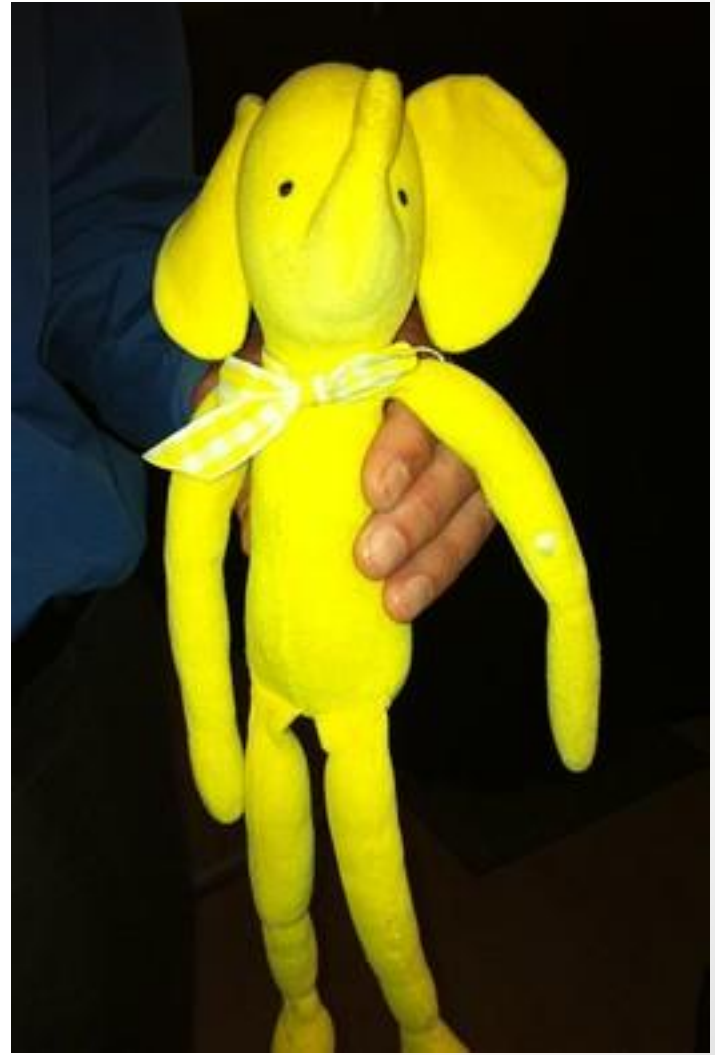


Movin' to Montana to raise me a crop of data (adapted from F. Zappa)

Hadoop-1 Intro

Hadoop Intro (1)

- Hadoop was created in 2008 at Yahoo by Doug Cutting and Mike Cafarella. Their work was based on two papers by Google:
 - **The Google File System**
 - **MapReduce: Simplified Data Processing on Large Clusters**



Hadoop Intro (2)

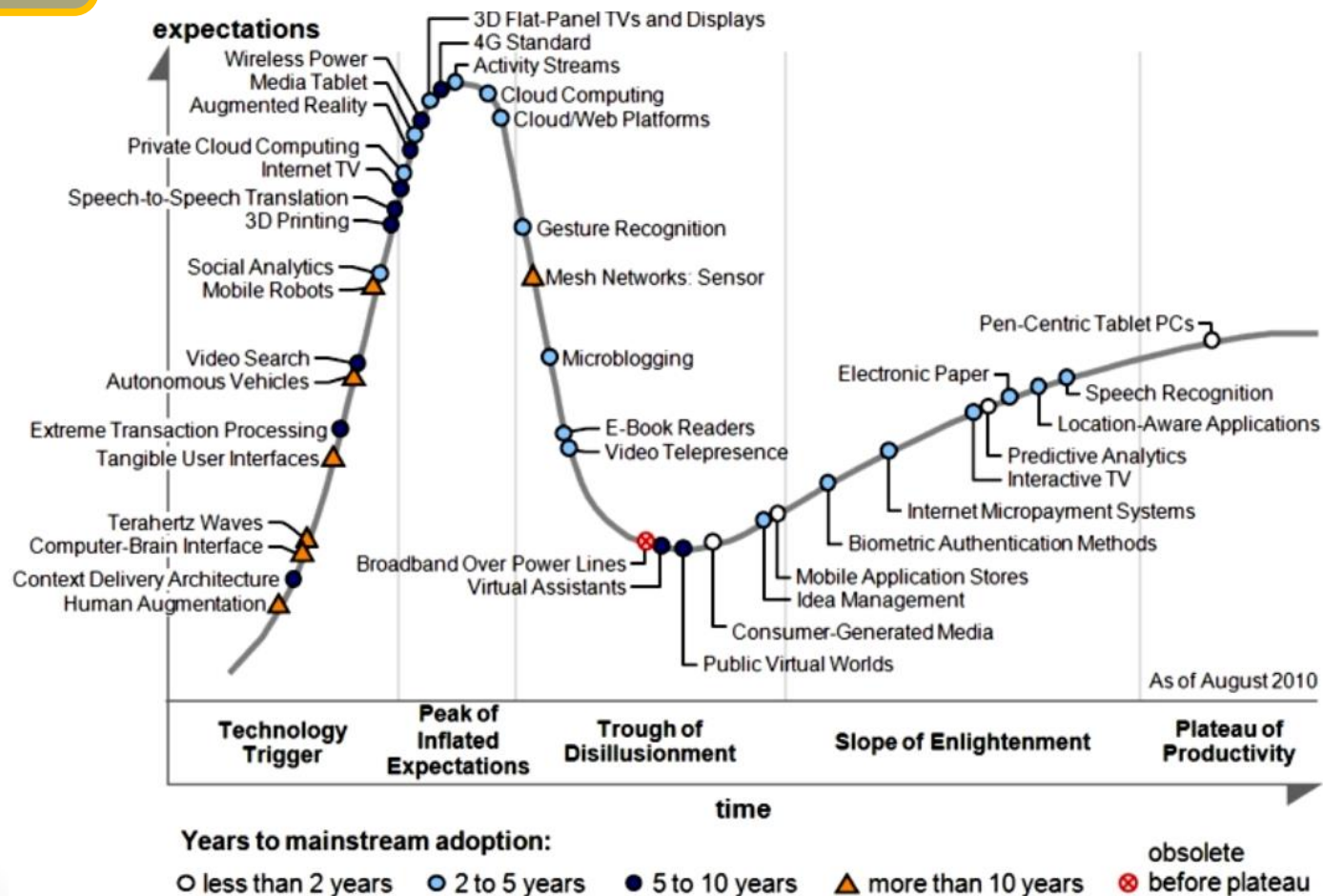
- Hadoop is
 - for big data
 - a framework for storing and processing large datasets (> 100 GB)
 - Apache open source software
 - inefficient and awkward for small datasets (< 100 GB)



Hadoop Intro (3)

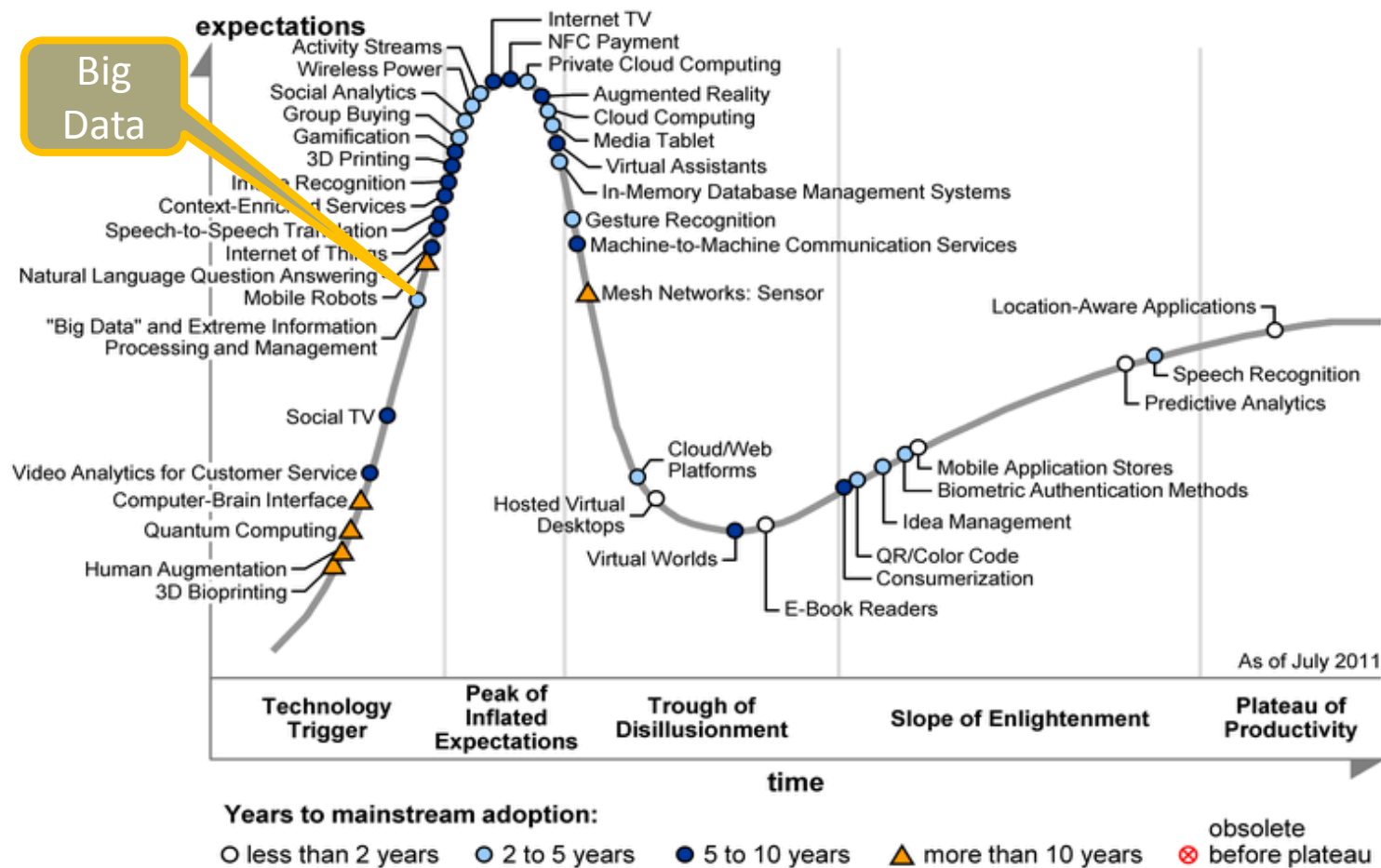
Emerging Technologies: 2010

Big
Data?



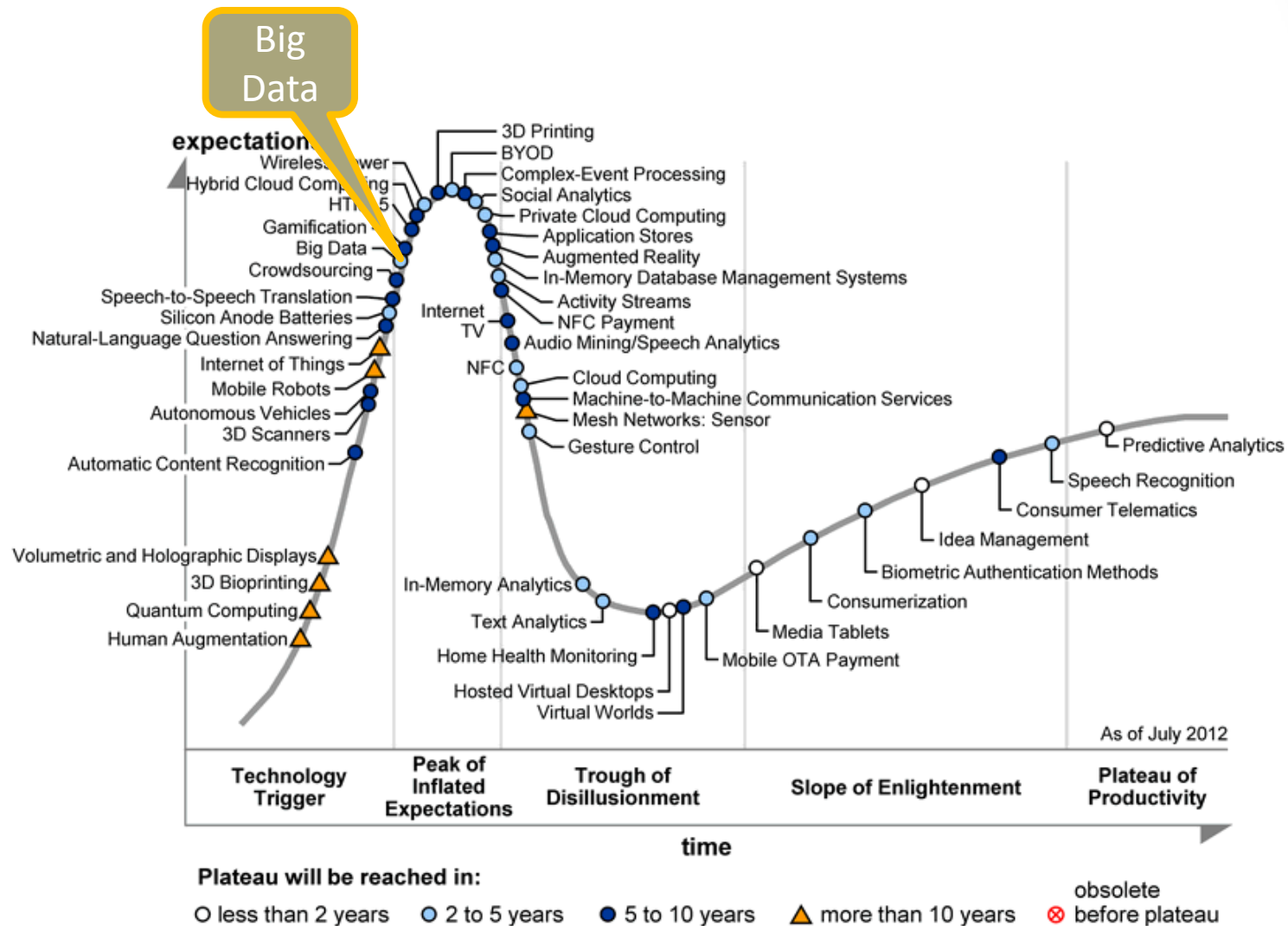
Hadoop Intro (4)

Emerging Technologies: 2011



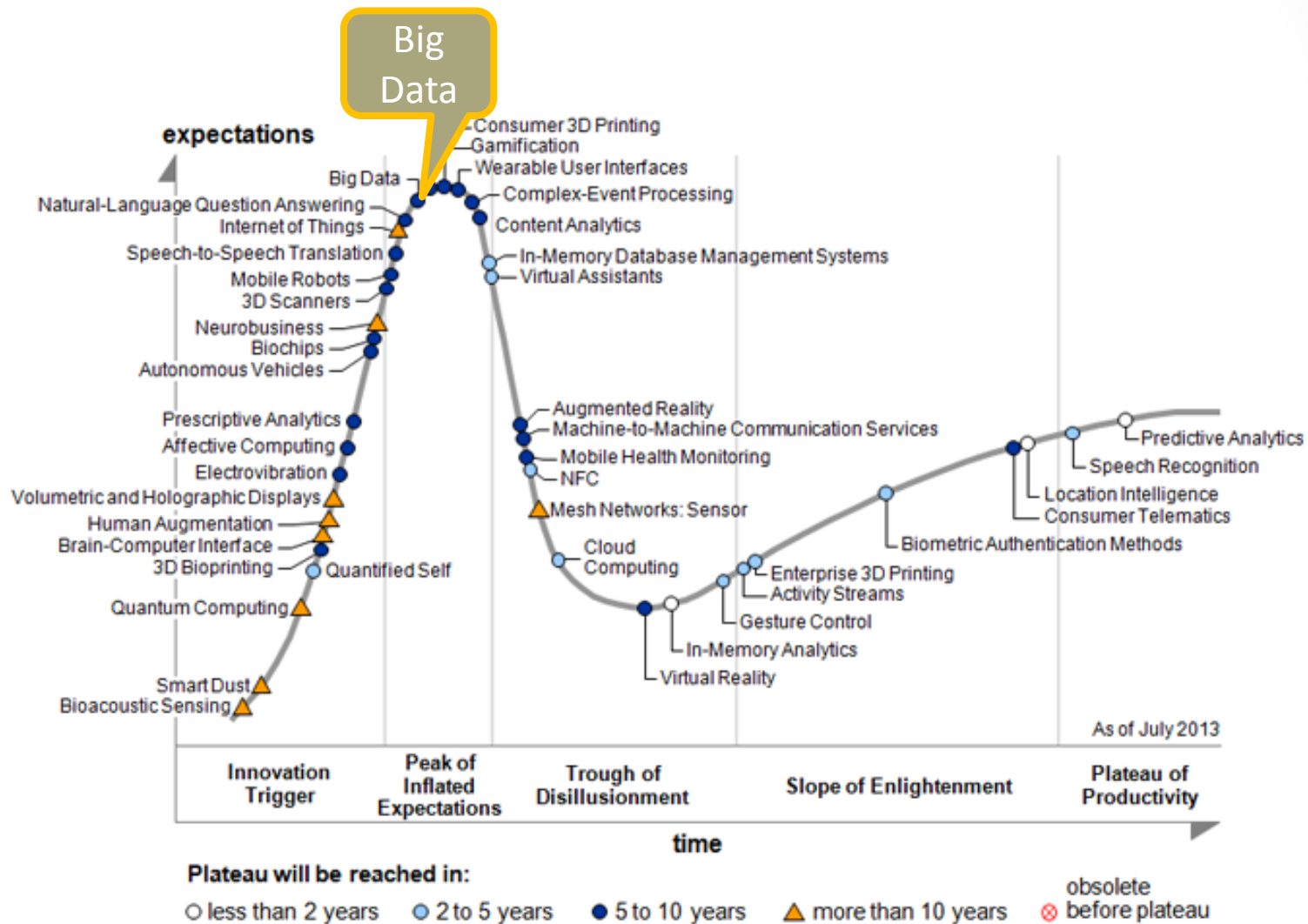
Hadoop Intro (5)

Emerging Technologies: 2012



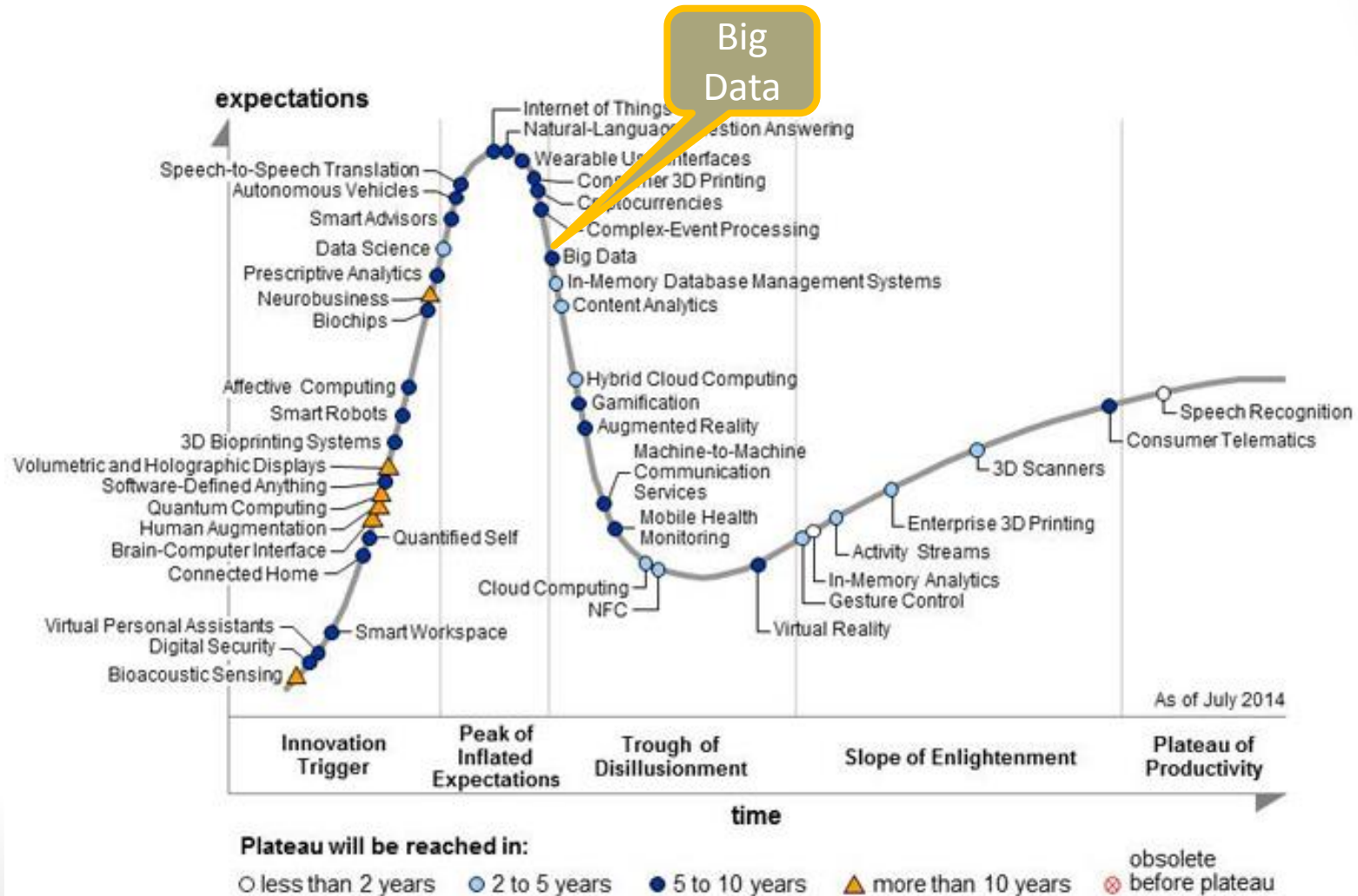
Hadoop Intro (6)

Emerging Technologies: 2013



Hadoop Intro (7)

Emerging Technologies: 2014



Hadoop Intro (8)

Emerging Technologies: 2015

Big
Data?



Hadoop Intro (9)

Emerging Technologies: 2016

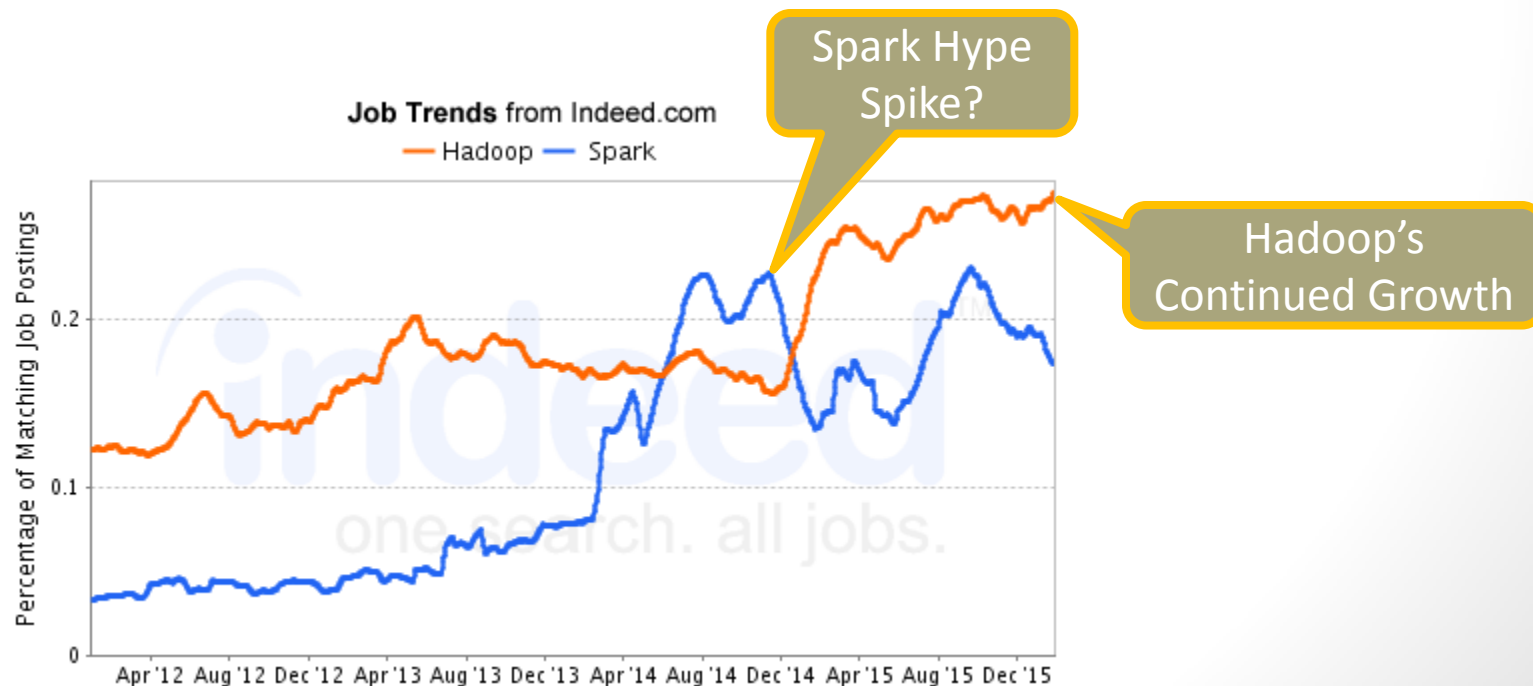
Big
Data?



Hadoop Intro (10)

Job outlooks

- Besides Hadoop's MapReduce there are other Big Data solutions. Spark is the most promising competitor due to in-memory computing.
- Where should we focus our efforts?



Hadoop Intro (11)

- Hadoop's most important two components
 1. HDFS for storage (Hadoop Distributed File System)
 2. MapReduce for processing (Supported programming pattern for Hadoop tasks)
- Other required Hadoop components
 1. YARN (Resource management of MapReduce tasks)
 2. Common (Basic Hadoop libraries)

Hadoop Intro (12)

- Commodity Hardware (typical: 2 processors, 4 GB)
- Scalable (Vary number of nodes)
- Distributed processing to increase processing power
- Data replication to make storage more secure
- Avoid Data roundtrips
- Write Once Read Many and Update as rarely as possible!
([http://en.wikipedia.org/wiki/Write once read many](http://en.wikipedia.org/wiki/Write_once_read_many))
- Streaming Access Pattern (Sequential Access of large chunks)
- Large block size (64 MB) enables fast sequential read. Data are streamed off the storage device while maintaining the maximum read rate. File sizes are in integer multiples of the block size. (Compare to common 4 KB block)

Hadoop Intro (13)

- Hadoop daemons:
 - Master Services:
 - NameNode: Manages HDFS
 - Secondary NameNode (Helps NameNode)
 - JobTracker: Manages TaskTrackers and MapReduce for a cluster
 - Slave Services:
 - DataNode: Executes MapReduce
 - TaskTracker: Manages MapReduce for a DataNode



Hadoop Intro (14)

- Ecosystem
 - Hive: Data warehouse on Hadoop
 - HiveQL SQL-like language for Hive
 - Impala: SQL-like query engine (Cloudera)
 - Sqoop: SQL to Hadoop. Transfers data between Hadoop and a relation database.
 - Oozie: Workflow scheduler for Hadoop. Scheduling is arranged like a DFD with DAG constraints
 - Flume: Collects and aggregates log files in Hadoop
 - Pig: Dataflow language and programming tool for MapReduce programs on Hadoop
 - Pig Latin: Language for Pig.
 - Mahout: Machine Learning algorithms that utilize the MapReduce pattern and run on Hadoop
 - Hue: Web-based interactive file browser

Hadoop Intro (15)

HDFS: How it works

- See: [HowHDFSworksByManeeshVarshney.pdf](#)

Quiz Hadoop

- You need to view the projected slide to answer questions 1 and 2



Hadoop-1 Intro

Hadoop-2 Labs

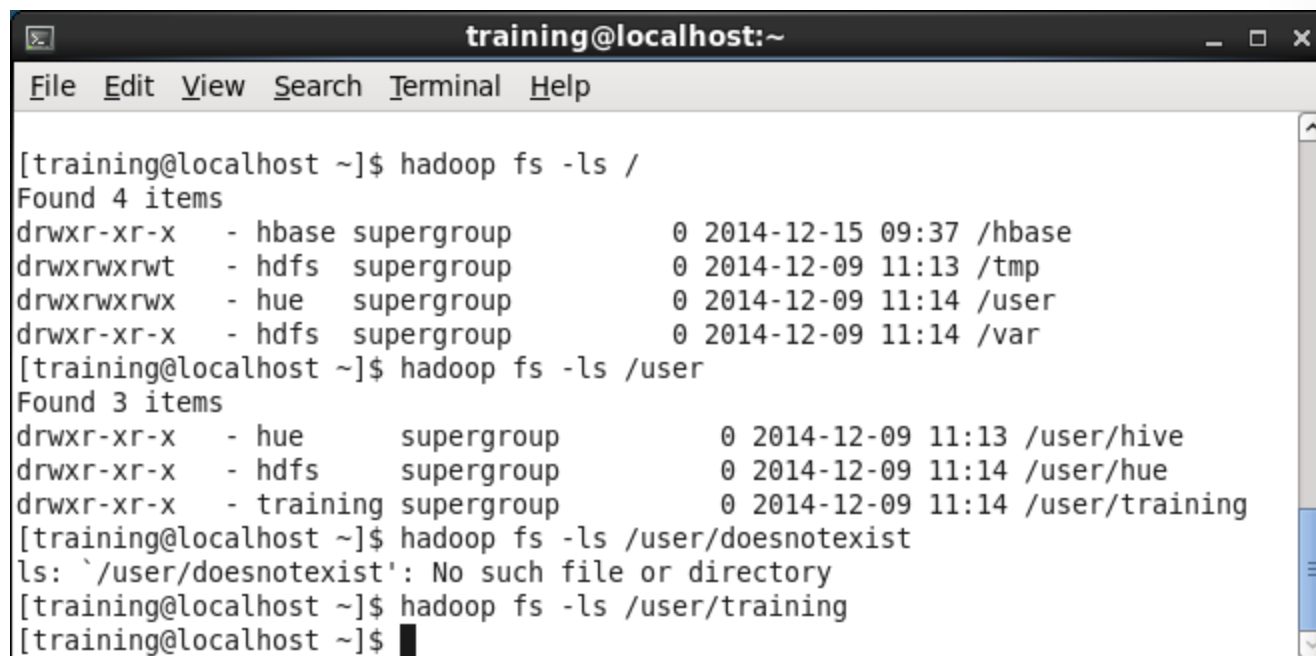
HDFS Lab (0)



“So you want to hire me as a Data Scientist for Intelligent Virtualized Deep Machine Learning Real-time Big Data in the Cloud for Social Networks? Ok, but if you also want Hadoop, increase my salary by 50%.”

HDFS Lab (1)

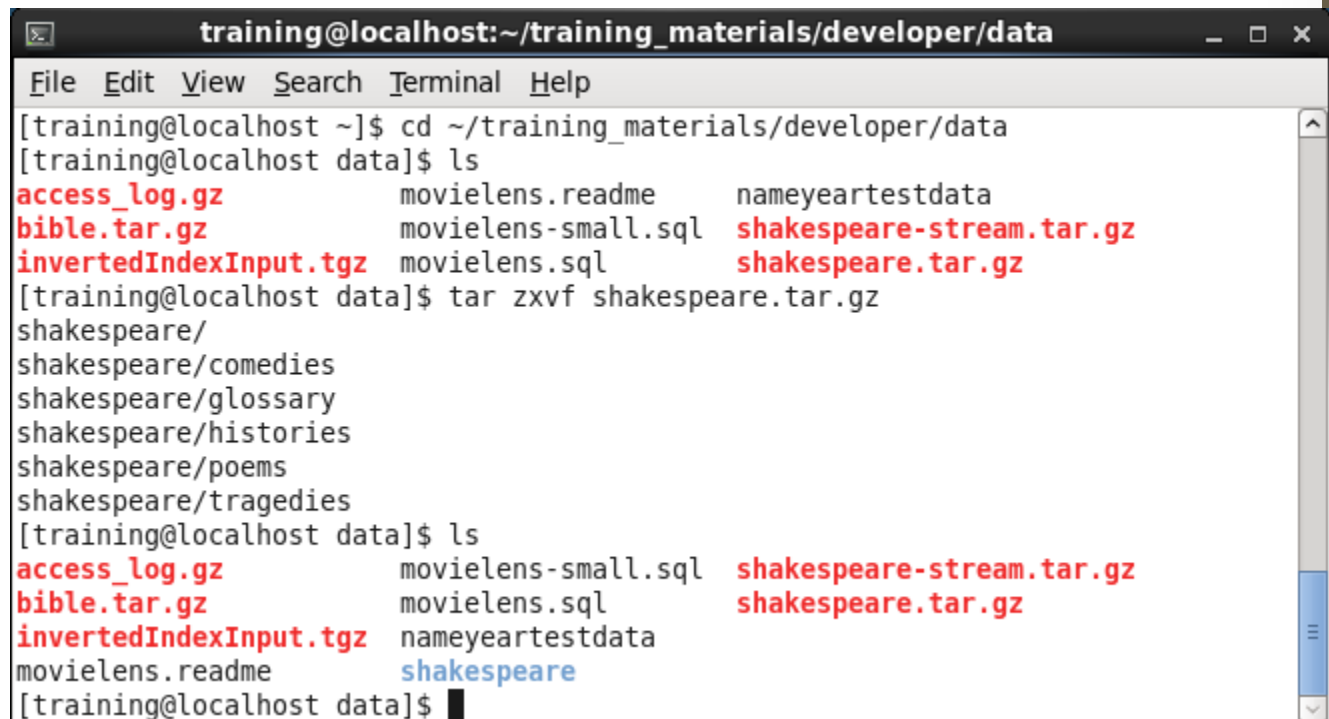
- Enter these commands into the console to list some folders and files inside HDFS:
 - `$ hadoop fs -ls /`
 - `$ hadoop fs -ls /user`
 - `$ hadoop fs -ls /user/doesnotexist`
 - `$ hadoop fs -ls /user/training`
- “fs” stands for file system (HDFS). “ls” is an argument to list HDFS files.



```
training@localhost:~  
File Edit View Search Terminal Help  
[training@localhost ~]$ hadoop fs -ls /  
Found 4 items  
drwxr-xr-x - hbase supergroup          0 2014-12-15 09:37 /hbase  
drwxrwxrwt - hdbs supergroup           0 2014-12-09 11:13 /tmp  
drwxrwxrwx - hue supergroup            0 2014-12-09 11:14 /user  
drwxr-xr-x - hdbs supergroup           0 2014-12-09 11:14 /var  
[training@localhost ~]$ hadoop fs -ls /user  
Found 3 items  
drwxr-xr-x - hue supergroup            0 2014-12-09 11:13 /user/hive  
drwxr-xr-x - hdbs supergroup           0 2014-12-09 11:14 /user/hue  
drwxr-xr-x - training supergroup       0 2014-12-09 11:14 /user/training  
[training@localhost ~]$ hadoop fs -ls /user/doesnotexist  
ls: '/user/doesnotexist': No such file or directory  
[training@localhost ~]$ hadoop fs -ls /user/training  
[training@localhost ~]$
```

HDFS Lab (2)

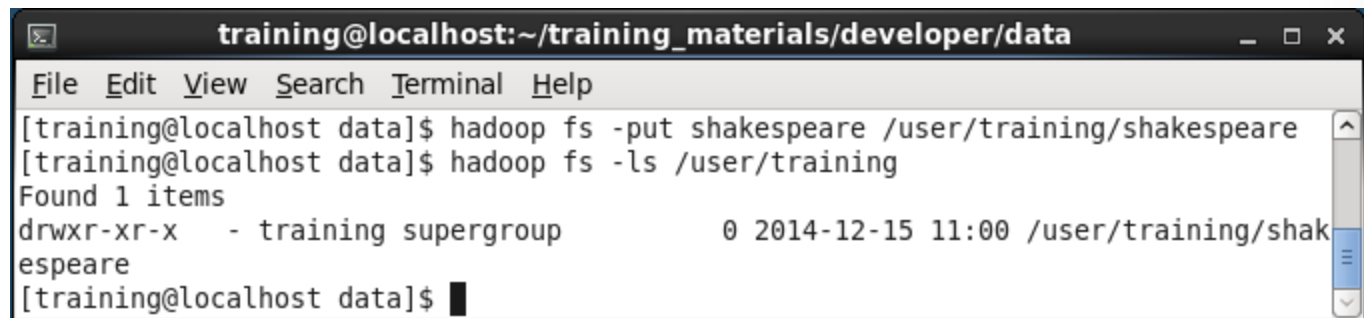
- Enter these commands into the console to find and expand the shakespeare tar on the Linux OS :
 - `$ cd ~/training_materials/developer/data`
 - `$ ls`
 - `$ tar zxvf shakespeare.tar.gz`
 - `$ ls`



```
training@localhost:~/training_materials/developer/data
File Edit View Search Terminal Help
[training@localhost ~]$ cd ~/training_materials/developer/data
[training@localhost data]$ ls
access_log.gz          movielens.readme      nameyeartestdata
bible.tar.gz           movielens-small.sql  shakespeare-stream.tar.gz
invertedIndexInput.tgz movielens.sql         shakespeare.tar.gz
[training@localhost data]$ tar zxvf shakespeare.tar.gz
shakespeare/
shakespeare/comedies
shakespeare/glossary
shakespeare/histories
shakespeare/poems
shakespeare/tragedies
[training@localhost data]$ ls
access_log.gz          movielens-small.sql  shakespeare-stream.tar.gz
bible.tar.gz           movielens.sql        shakespeare.tar.gz
invertedIndexInput.tgz nameyeartestdata
movielens.readme       shakespeare
[training@localhost data]$
```

HDFS Lab (3)

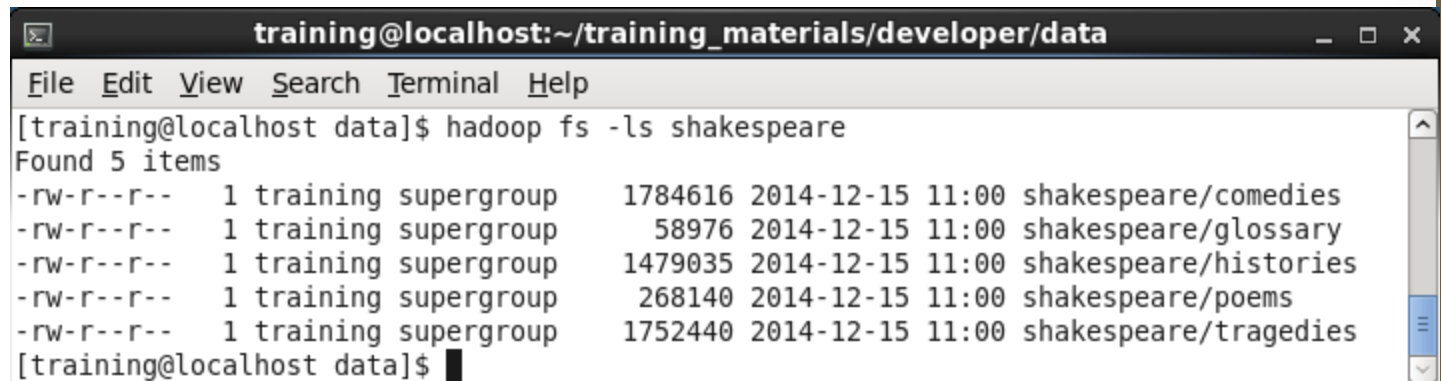
- Enter these commands into the console to place the shakespeare directory into HDFS under training and then verify that the directory is in HDFS:
 - `$ hadoop fs -put shakespeare /user/training/shakespeare`
 - `$ hadoop fs -ls /user/training`

A terminal window titled 'training@localhost:~/training_materials/developer/data'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the following commands and output:

```
[training@localhost data]$ hadoop fs -put shakespeare /user/training/shakespeare
[training@localhost data]$ hadoop fs -ls /user/training
Found 1 items
drwxr-xr-x  - training supergroup          0 2014-12-15 11:00 /user/training/shak
espeare
[training@localhost data]$
```

HDFS Lab (4)

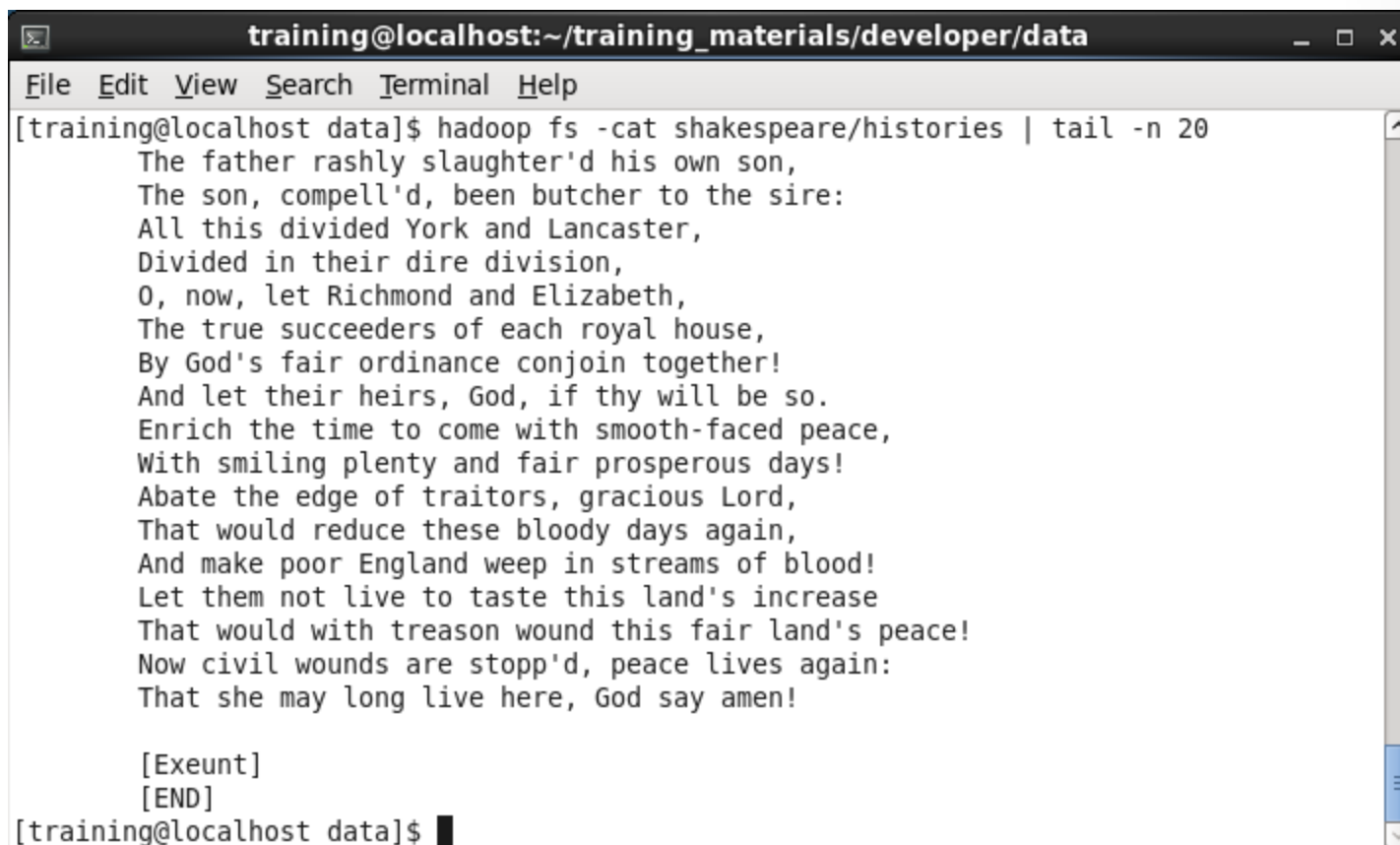
- Enter this command into the console to list the 5 files in the shakespeare directory of HDFS:
 - `$ hadoop fs -ls /user/training/shakespeare`

A terminal window titled 'training@localhost:~/training_materials/developer/data' with a menu bar (File, Edit, View, Search, Terminal, Help). The command '[training@localhost data]\$ hadoop fs -ls shakespeare' has been executed, resulting in the output 'Found 5 items' followed by a list of five files with their permissions, owner, group, size, timestamp, and path.

```
training@localhost:~/training_materials/developer/data
File Edit View Search Terminal Help
[training@localhost data]$ hadoop fs -ls shakespeare
Found 5 items
-rw-r--r--  1 training supergroup 1784616 2014-12-15 11:00 shakespeare/comedies
-rw-r--r--  1 training supergroup  58976 2014-12-15 11:00 shakespeare/glossary
-rw-r--r--  1 training supergroup 1479035 2014-12-15 11:00 shakespeare/histories
-rw-r--r--  1 training supergroup  268140 2014-12-15 11:00 shakespeare/poems
-rw-r--r--  1 training supergroup 1752440 2014-12-15 11:00 shakespeare/tragedies
[training@localhost data]$
```

HDFS Lab (5)

- Enter this command into the console to read the last 20 lines of the histories file in HDFS:
 - `$hadoop fs -cat shakespeare/histories | tail -n 20`

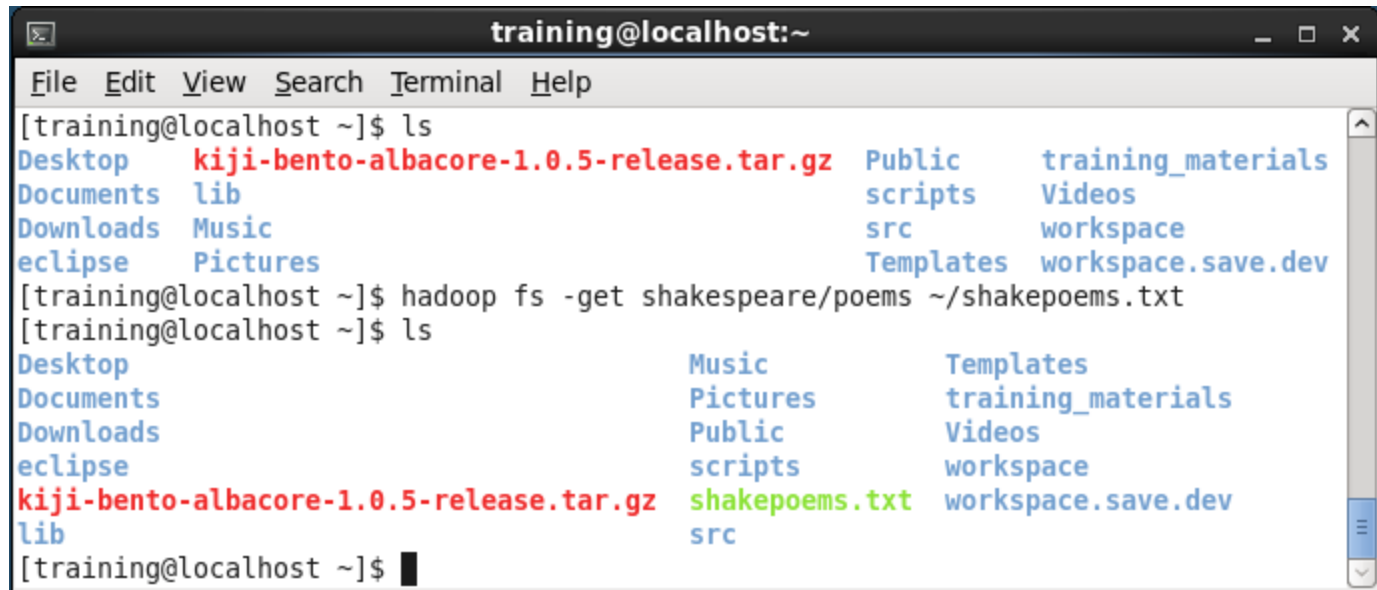


```
training@localhost:~/training_materials/developer/data
File Edit View Search Terminal Help
[training@localhost data]$ hadoop fs -cat shakespeare/histories | tail -n 20
    The father rashly slaughter'd his own son,
    The son, compell'd, been butcher to the sire:
    All this divided York and Lancaster,
    Divided in their dire division,
    O, now, let Richmond and Elizabeth,
    The true succeeders of each royal house,
    By God's fair ordinance conjoin together!
    And let their heirs, God, if thy will be so.
    Enrich the time to come with smooth-faced peace,
    With smiling plenty and fair prosperous days!
    Abate the edge of traitors, gracious Lord,
    That would reduce these bloody days again,
    And make poor England weep in streams of blood!
    Let them not live to taste this land's increase
    That would with treason wound this fair land's peace!
    Now civil wounds are stopp'd, peace lives again:
    That she may long live here, God say amen!

    [Exeunt]
    [END]
[training@localhost data]$
```

HDFS Lab (6)

- Enter the following commands to retrieve files from HDFS and list them in the Linux OS:
 - `$ cd ~`
 - `$ ls`
 - `$ hadoop fs -get shakespeare/poems ~/shakepoems.txt`
 - `$ ls`



```
training@localhost:~  
File Edit View Search Terminal Help  
[training@localhost ~]$ ls  
Desktop      kiji-bento-albacore-1.0.5-release.tar.gz  Public      training_materials  
Documents    lib                                         scripts     Videos  
Downloads    Music                                     src          workspace  
eclipse      Pictures                                  Templates   workspace.save.dev  
[training@localhost ~]$ hadoop fs -get shakespeare/poems ~/shakepoems.txt  
[training@localhost ~]$ ls  
Desktop      Music      Templates  
Documents    Pictures   training_materials  
Downloads    Public     Videos  
eclipse      scripts    workspace  
kiji-bento-albacore-1.0.5-release.tar.gz  shakepoems.txt  workspace.save.dev  
lib          src  
[training@localhost ~]$
```


Sqoop Lab (0)

- Sql + Hadoop → Sqoop
- Use Sqoop to share between a RDBMS and Hadoop



"Finance here - we're not sure about this Hadoop thing... Could you just dump it all into Excel for us?"

TimoElliott.com

Sqoop Lab (1)

- Use this Sqoop command to familiarize yourself with Sqoop:
 - `$ sqoop help`
- Use these sqoop commands to list mysql databases on localhost and then tables in one of those databases:
 - `$ sqoop list-databases --connect jdbc:mysql://localhost --username training --password training`
 - `$ sqoop list-tables --connect jdbc:mysql://localhost/movielens --username training --password training`

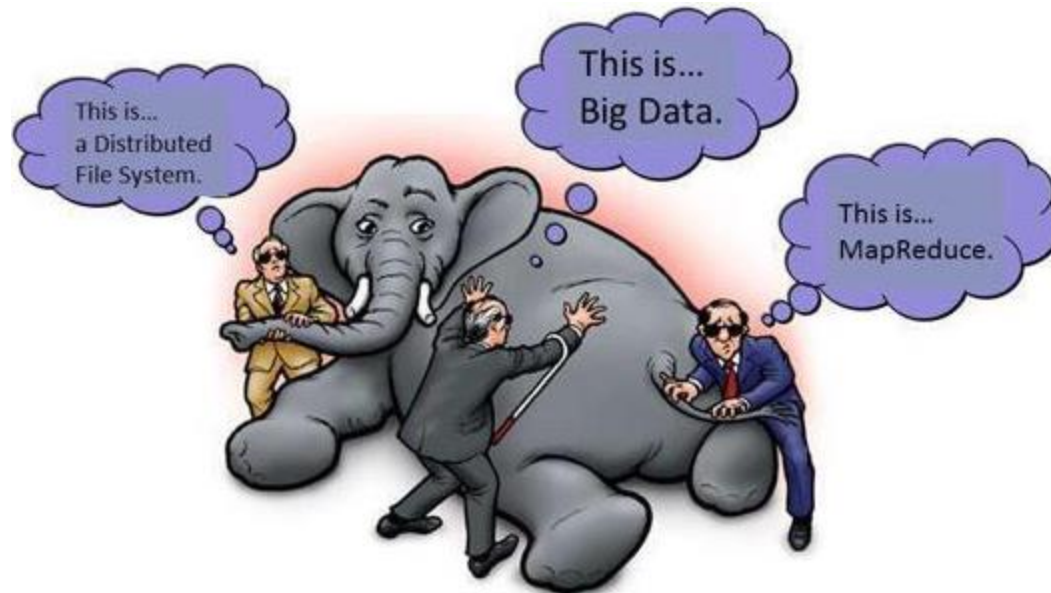
Sqoop Lab (2)

- Use this Sqoop command to import the movie table from mysql to an HDFS file. The fields of a table record (row) will be separated by tabs in the corresponding record of the HDFS file:
 - `$ sqoop import --connect jdbc:mysql://localhost/movielens --username training --password training --fields-terminated-by '\t' --table movie`
- Note that Sqoop constructs some SQL statements for MySQL and that it converts the SQL statement into a MapReduce job that only has a Map component:
 - `Test SQL: SELECT t.* FROM `movie` AS t LIMIT 1`
 - `SELECT MIN(`id`), MAX(`id`) FROM `movie``
- Use this HDFS command to list the newly imported file(s) and then view the last part of one of the files:
 - `$ hadoop fs -ls movie`
 - `$ hadoop fs -tail movie/part-m-00000`

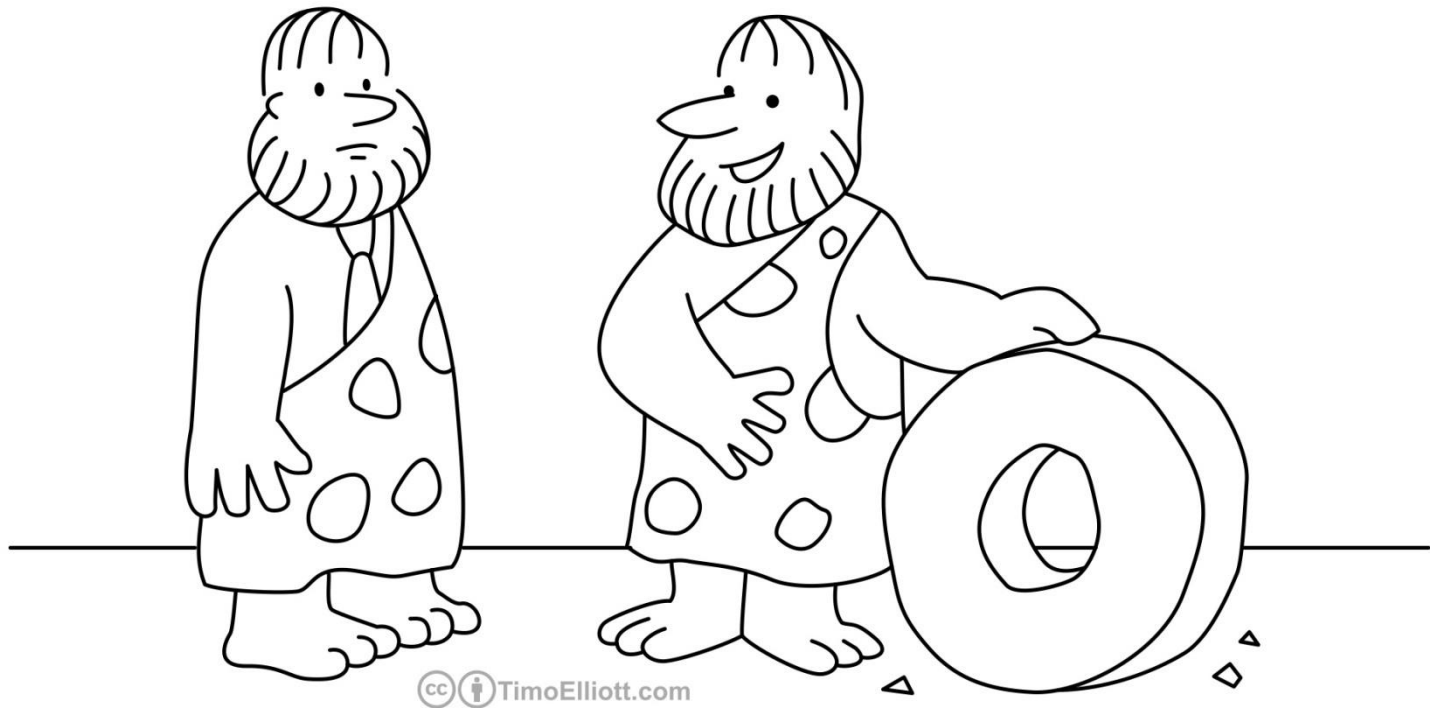
Sqoop Lab (3)

- Use this Sqoop command to import the movierating table from mysql to an HDFS file. The fields of a table record (row) will be separated by tabs in the corresponding record of the HDFS file:
 - `$ sqoop import --connect jdbc:mysql://localhost/movielens --username training --password training --fields-terminated-by '\t' --table movierating`
- Use these HDFS commands to list the newly imported file(s) and then view the last part of one of the files:
 - `$ hadoop fs -ls movierating`
 - `$ hadoop fs -tail movierating/part-m-00000`

Break



Hive Lab (0)



*“It does look similar—but this one
is powered by Hadoop”*

Hive Lab (1)

- Check that we have movie and movierating in HDFS

- `$ hadoop fs -cat /movie/part-m-00000 | head`
- `$ hadoop fs -cat /movierating/part-m-00000 | head`

- Start Hive

- `$ hive`



- Tell Hive that the HDFS files, movie and movierating, are tables:

- `hive> CREATE EXTERNAL TABLE movie (id INT, name STRING, year INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION '/user/training/movie';`
- `hive> CREATE EXTERNAL TABLE movierating (userid INT, movieid INT, rating INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION '/user/training/movierating';`

Hive Lab (2)

- Ask Hive to show tables and provide metadata on these tables
 - `hive> SHOW TABLES;`
 - `hive> DESCRIBE movie;`
 - `hive> DESCRIBE movierating;`
- Use Hive to find information from your tables
 - `hive> SELECT * FROM movie WHERE year < 1925;`
 - `hive> SELECT * FROM movie WHERE year < 1925 AND year != 0 ORDER BY name;`
 - `hive> SELECT * FROM movierating WHERE userid=149;`
 - `hive> SELECT AVG(rating) FROM movierating WHERE userid=149;`
 - `hive> SELECT userid, COUNT(userid),AVG(rating) FROM movierating where userid < 10 GROUP BY userid;`

Hive Lab (3)

- Use Hive to create a new table

- `hive> CREATE TABLE MovieRated (MovieID INT, NumRatings INT, AvgRating FLOAT);`
- `hive> insert overwrite table MovieRated SELECT movieid,COUNT(movieid),AVG(rating) FROM movierating GROUP BY movieid;`

- Query the new table

- `hive> SELECT COUNT(*) FROM MovieRated;`
- `hive> SELECT * FROM MovieRated WHERE AvgRating > 4.4 and NumRatings > 1000;`

Hive Lab (4)

- Query with Join

- `hive> select name, NumRatings , AvgRating from MovieRated join movie on MovieRated.movieid=movie.id where AvgRating > 4.4 and NumRatings > 500 ORDER BY AvgRating;`

- Quit Hive

- `hive> QUIT;`

- Take-home Exercise

- What is the name of the oldest movie in the database that has a top rating?

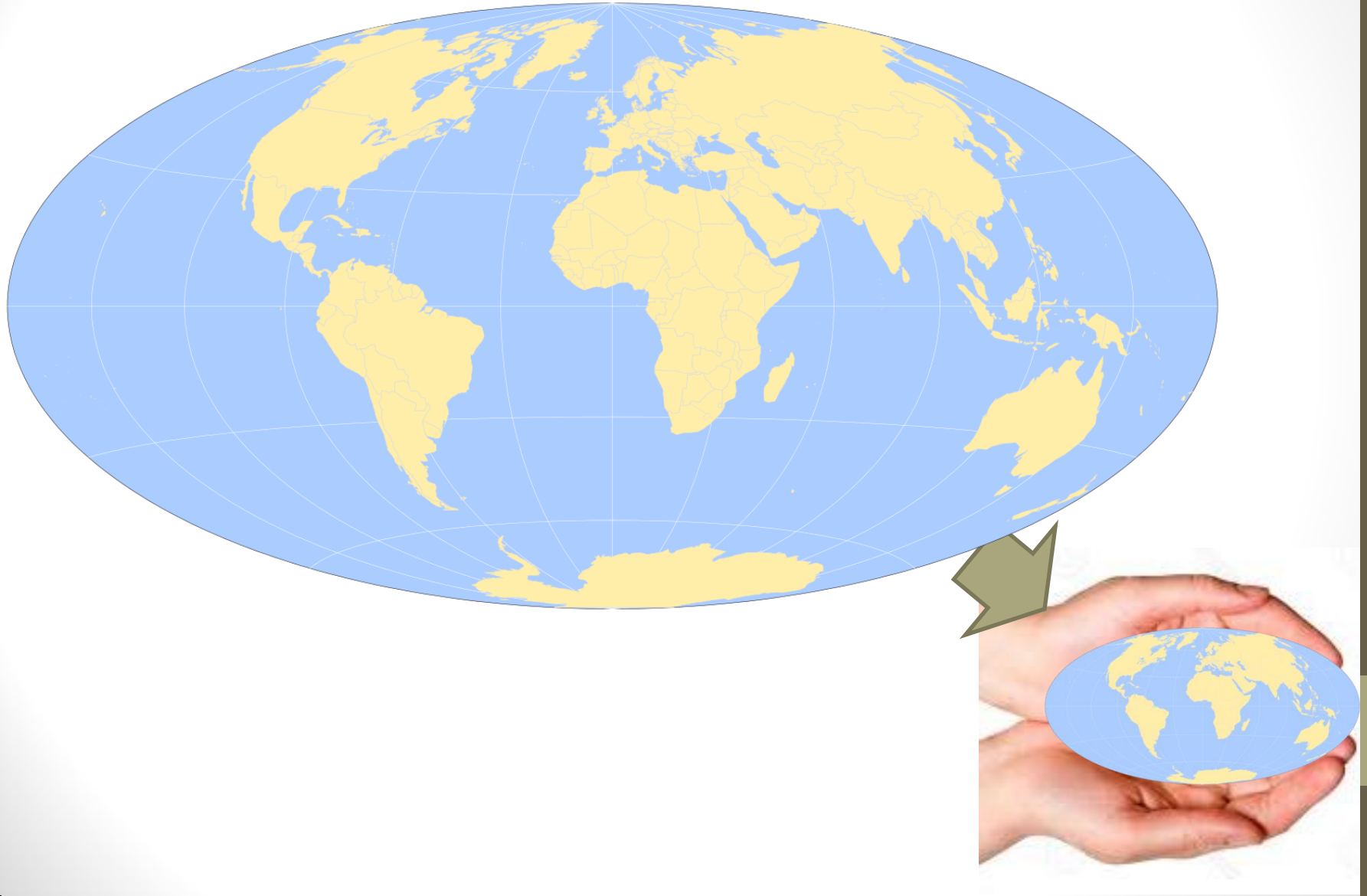
Hive Lab (5) Impala

- Use Impala to execute “HiveQL”

1. Open Firefox
2. Start Hue by clicking on the Hue link in the Bookmarks toolbar (Username: training, Password: training)
3. Select Impala and Query Editor Tab
4. Enter Query into Query text box for Impala:
 - **`select name, NumRatings , AvgRating from MovieRated join movie on MovieRated.movieid=movie.id where NumRatings > 500 ORDER BY AvgRating DESC limit 20;`**
5. Click on Execute



MapReduce (0)



MapReduce (1)

- The purpose of Hadoop is to facilitate scale-out computing.
- MapReduce is Hadoop's answer to scale-out processing (Like HDFS is Hadoop's answer to scale-out persistence).
- The ideas behind distributed processing:
 - Send the program to the data as opposed to the data to the program
 - Make use of distributed data where each chunk of data already has its own CPUs
 - Allow independent, asynchronous processing of data
- MapReduce is a pattern (programming model) designed to make coding of distributed processes easy.

MapReduce (2)

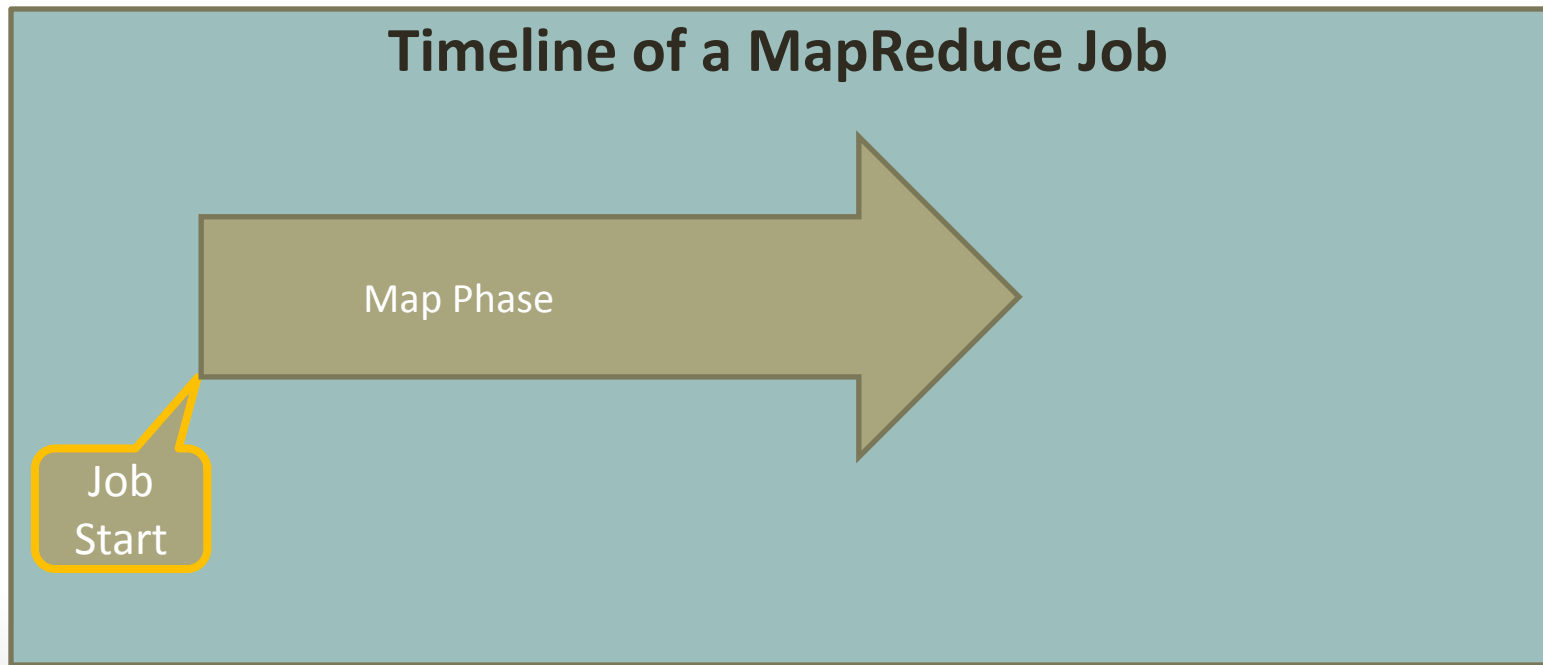
- The word MapReduce is made from **Map** and **Reduce**
- When a MapReduce job is started, then Hadoop sends **Map** and **Reduce** jobs to the data nodes. Hadoop manages all the details of data passing among data nodes.
 - The **Map** portion of the computation occurs on the individual data nodes where the data already reside. There is no need for the data to travel.
 - The **Reduce** portion of the computation occurs on some of the data nodes (not necessarily where the data reside). In some cases part of the **Reduce** operation can occur on the data node where the data already reside.

MapReduce (3)

- MapReduce uses three stages:
 1. Map
 2. Shuffle
 3. Reduce
- Then, why don't we call MapReduce: MapShuffleReduce?
 - Because: In most cases, the programmer need only be concerned with Map and Reduce. The Shuffle and Sorting is taken care of by Hadoop.
 - To achieve scalable programs, Hadoop takes care off:
 - Creating tasks on the various data nodes
 - Tracking Tasks
 - Moving data among data nodes
 - File I/O, networking, process synchronization, recovery from failures, re-running jobs, splitting input, Moving Key-Value-Pairs from Map to Reduce
 - Outputs results

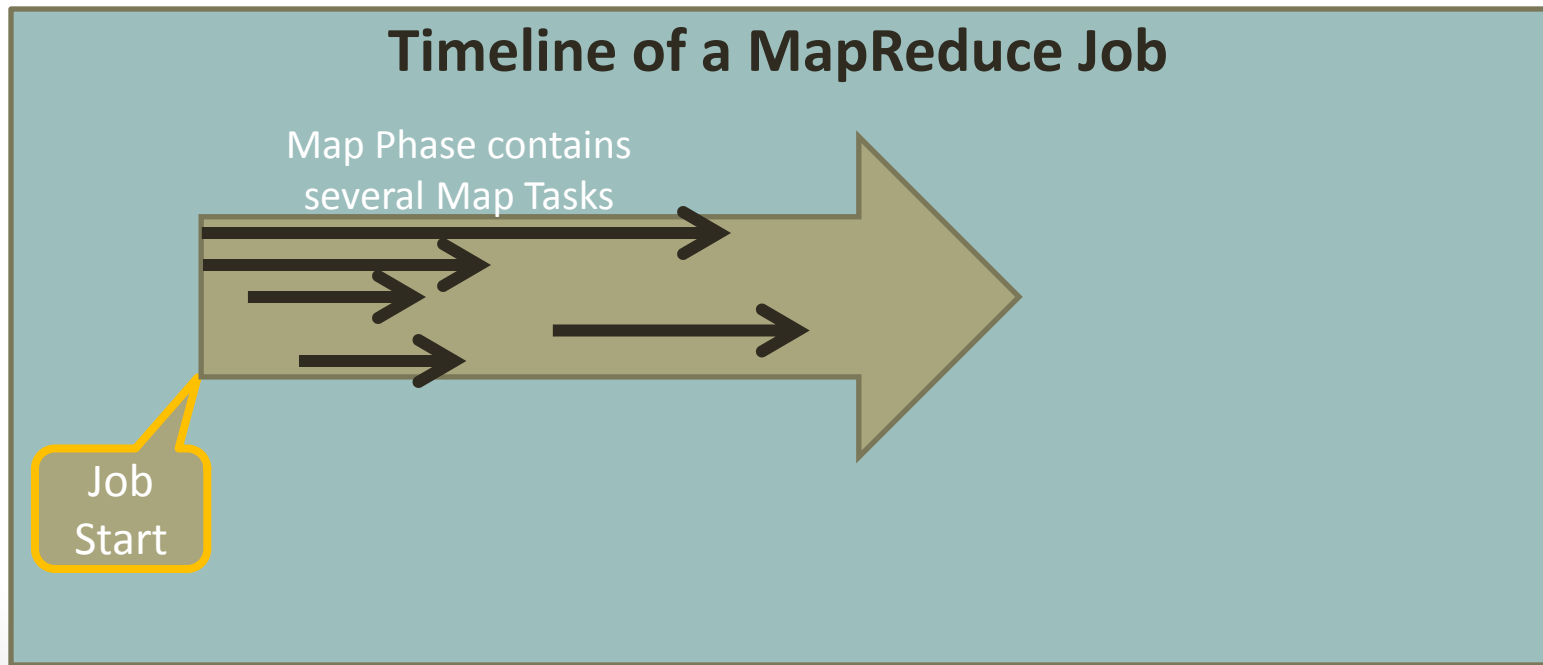
MapReduce (4)

- Parallelization Rules in MapReduce



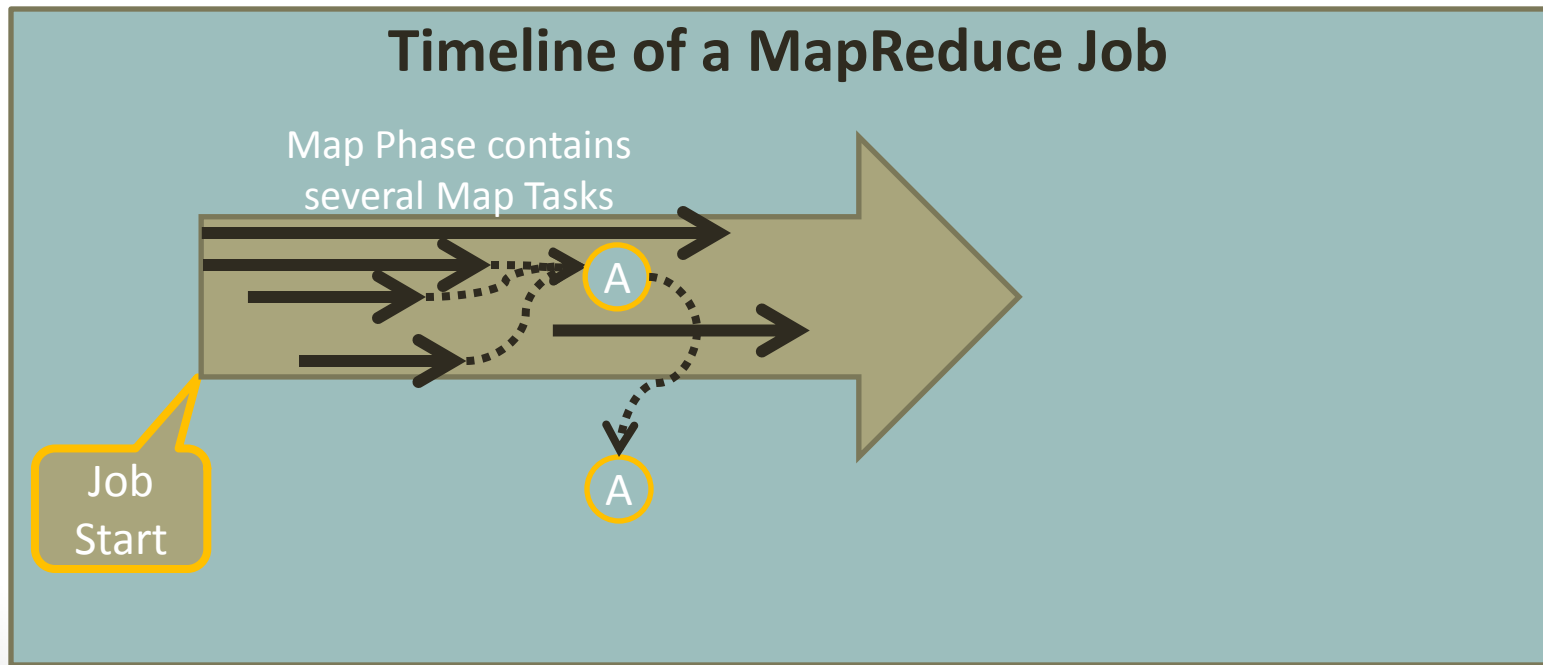
MapReduce (5)

- Parallelization Rules in MapReduce
 - Map tasks occur in parallel independent of each other



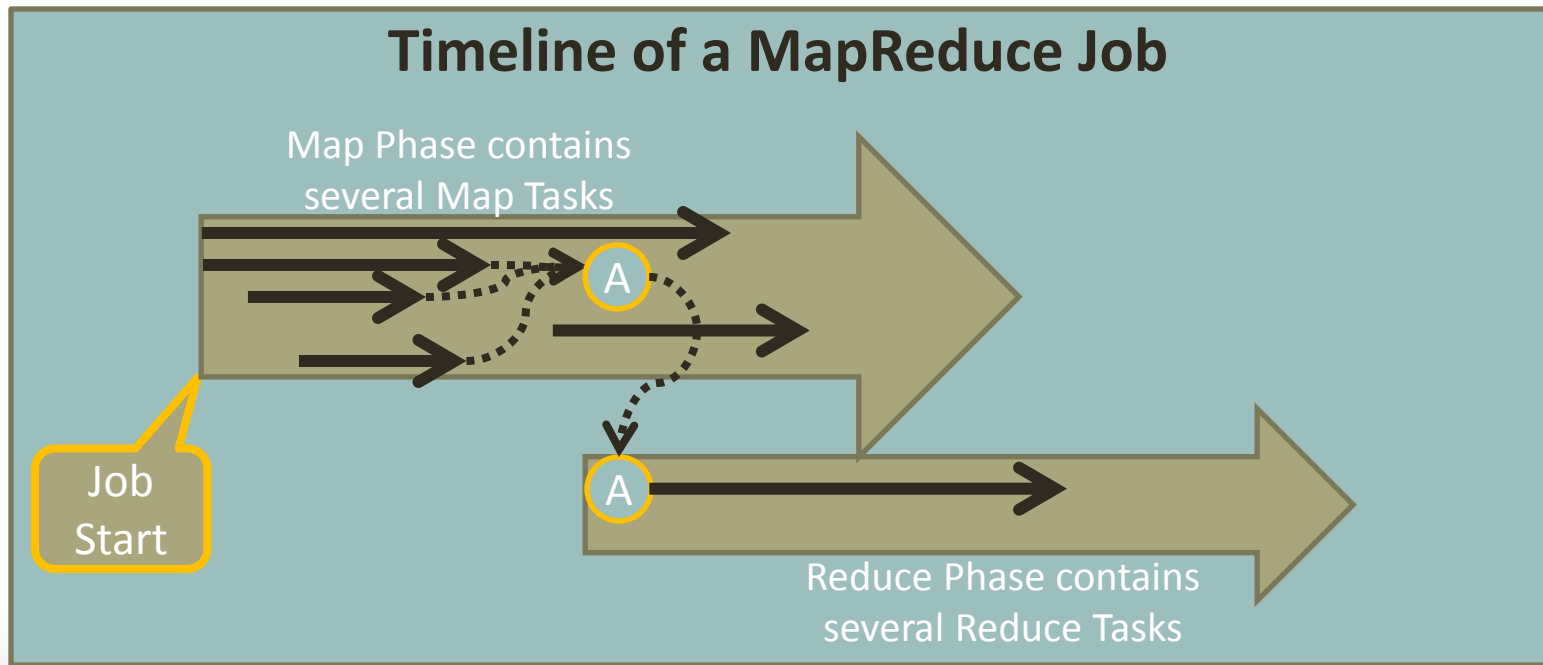
MapReduce (6)

- Parallelization Rules in MapReduce
 - Map tasks occur in parallel independent of each other
 - Map tasks terminate in “Shuffle and Sort”



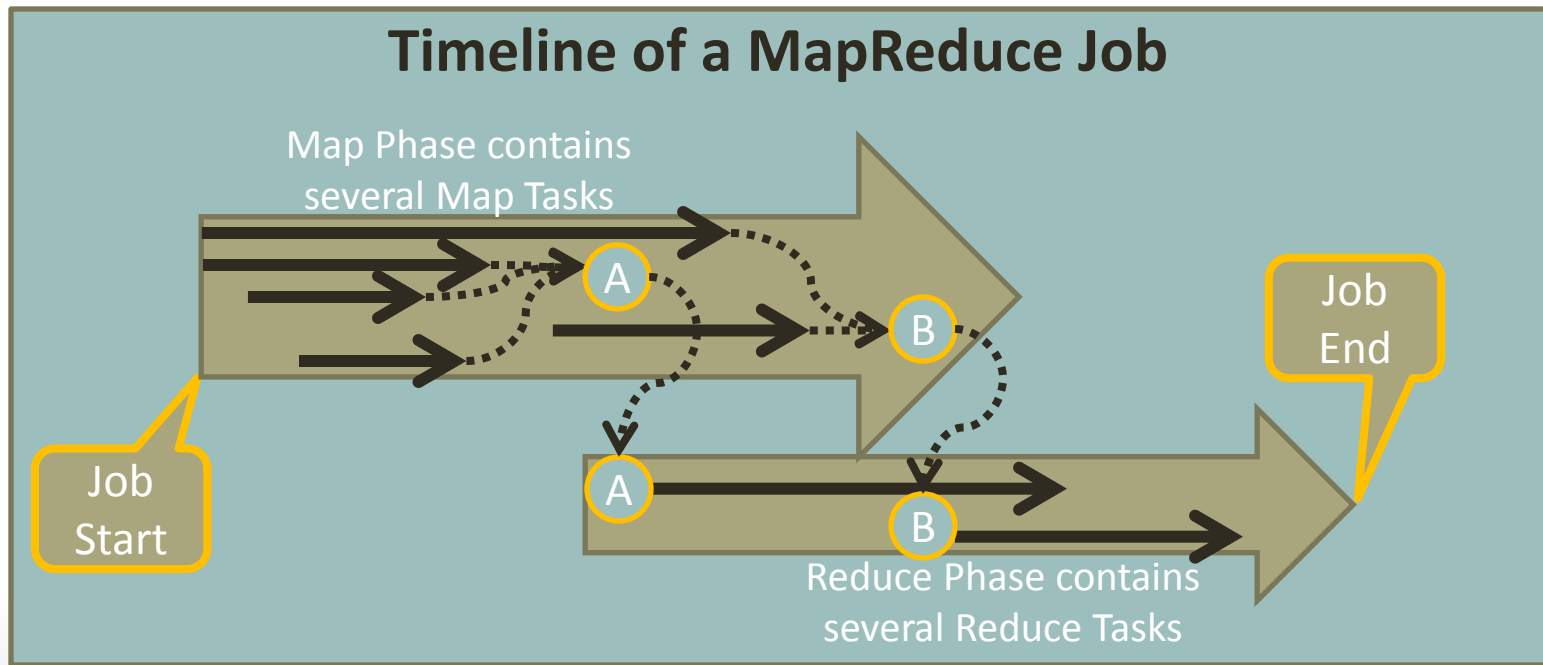
MapReduce (7)

- Parallelization Rules in MapReduce
 - Map tasks occur in parallel independent of each other
 - Map tasks terminate in “Shuffle and Sort”
 - Some Reduce tasks can start before all Mappers are done.



MapReduce (8)

- Parallelization Rules in MapReduce
 - Map tasks occur in parallel independent of each other
 - Map tasks terminate in “Shuffle and Sort”
 - Some Reduce tasks can start before all Mappers are done.
 - Reduce tasks occur in parallel independent of each other
 - Reducers cannot complete before all Mapping and Shuffle & Sort have completed.

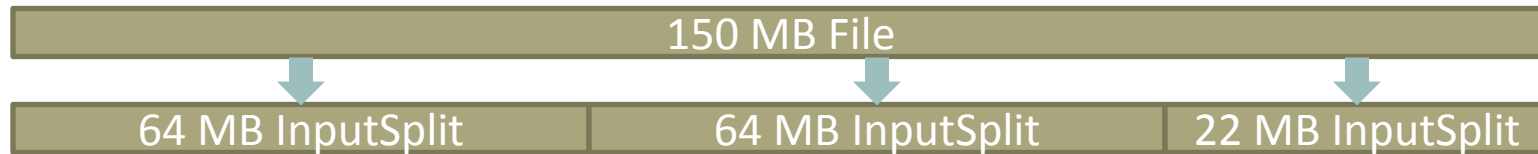


MapReduce (9)

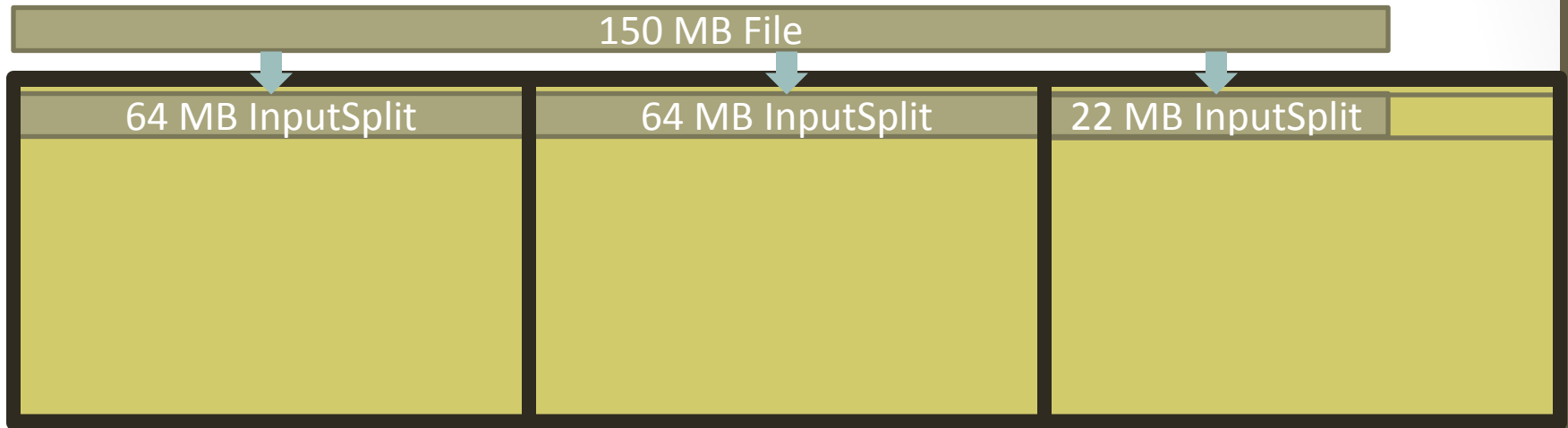
A horizontal bar with a dark olive green border and a lighter olive green fill. The text "150 MB File" is centered within the bar.

150 MB File

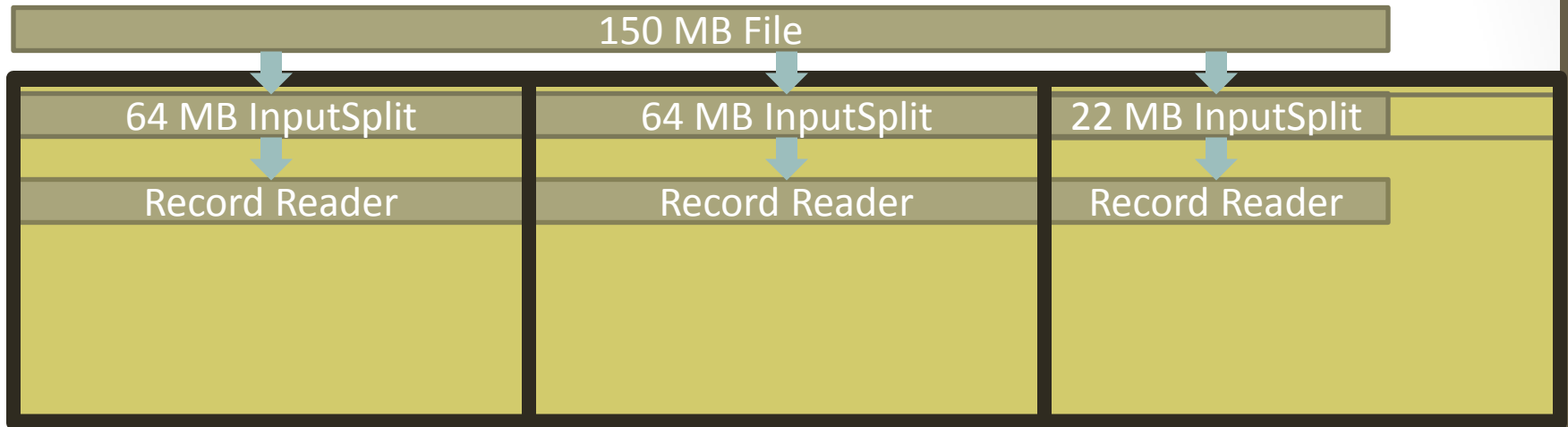
MapReduce (10)



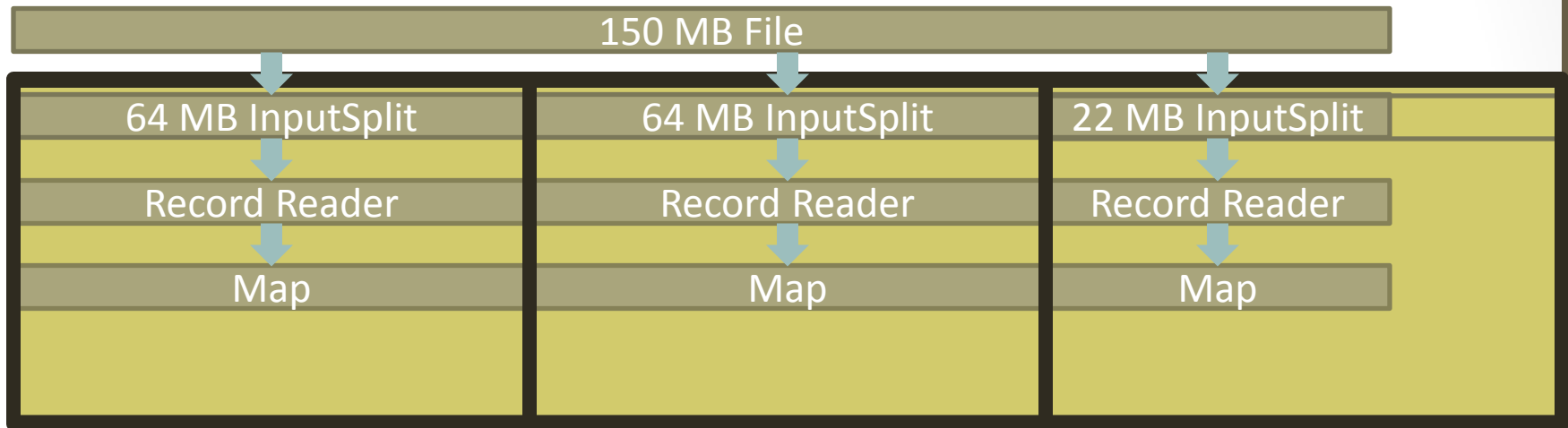
MapReduce (11)



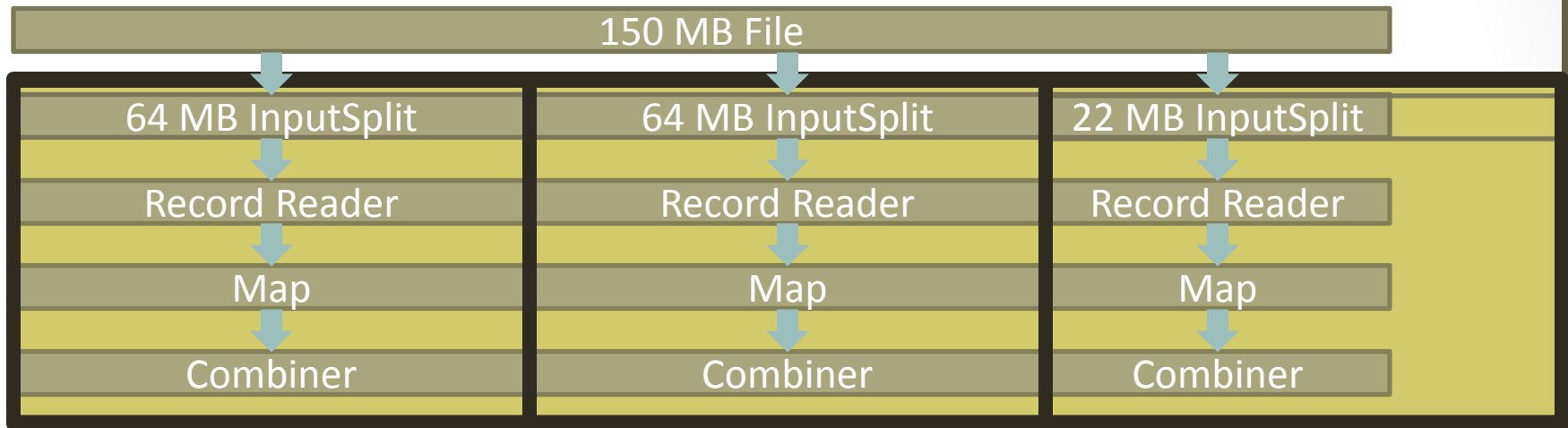
MapReduce (12)



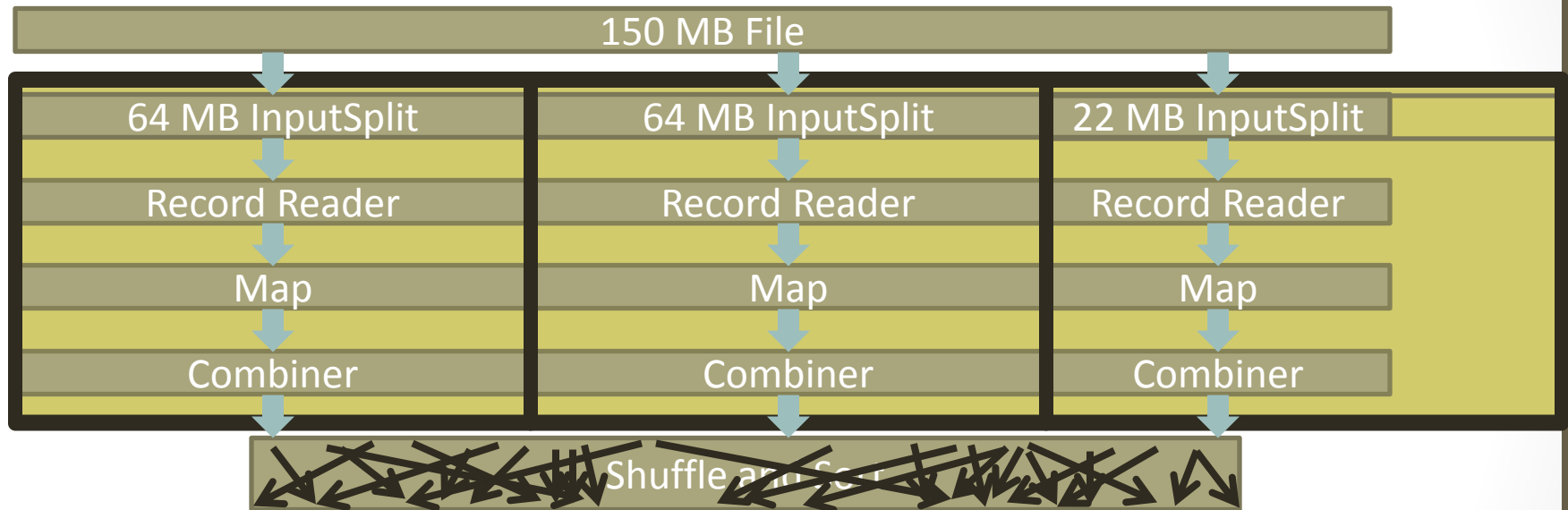
MapReduce (13)



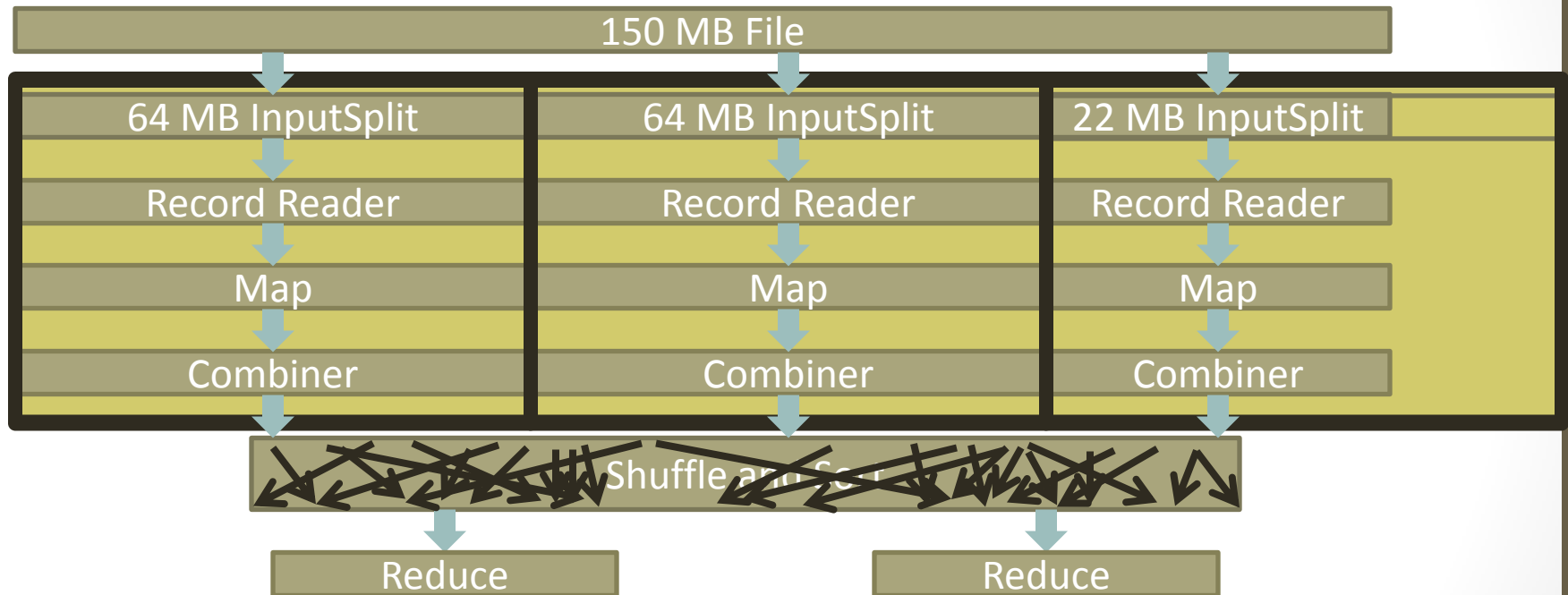
MapReduce (14)



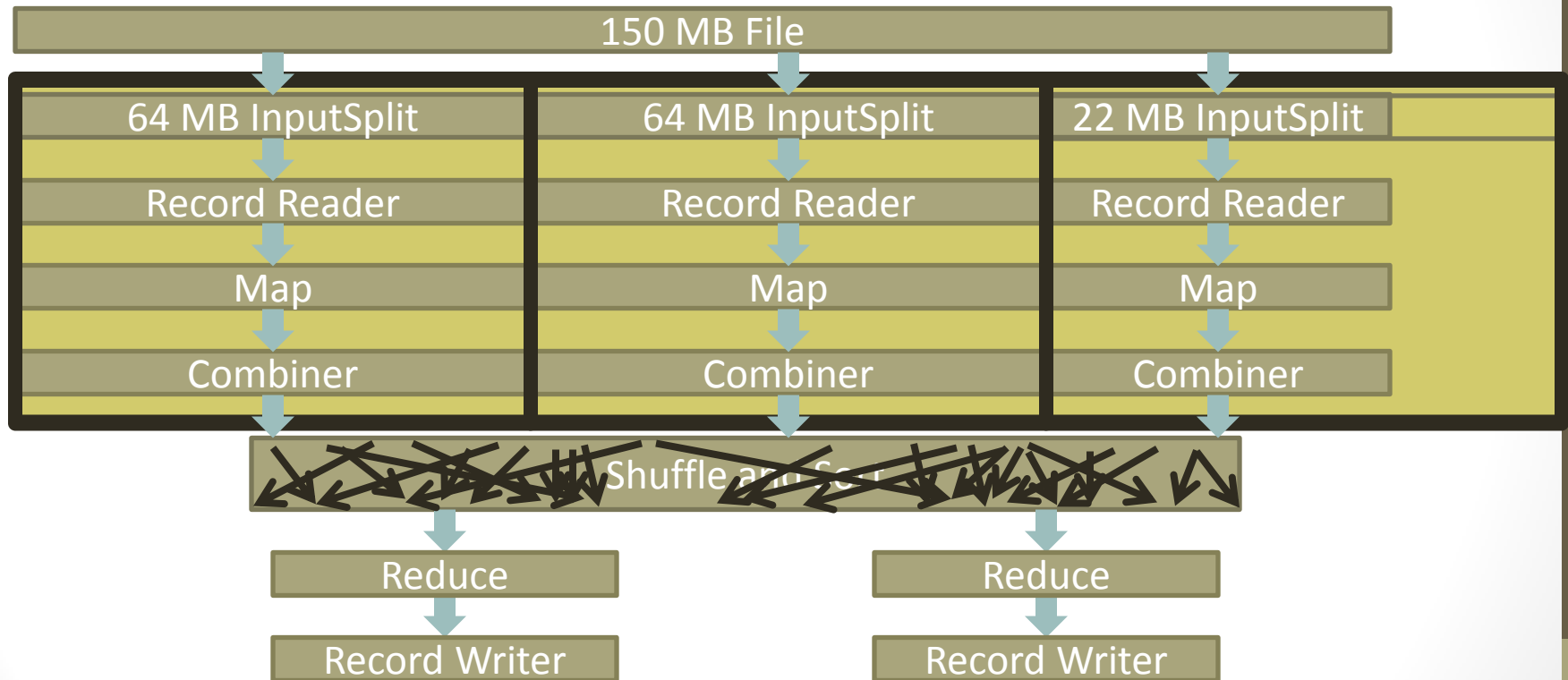
MapReduce (15)



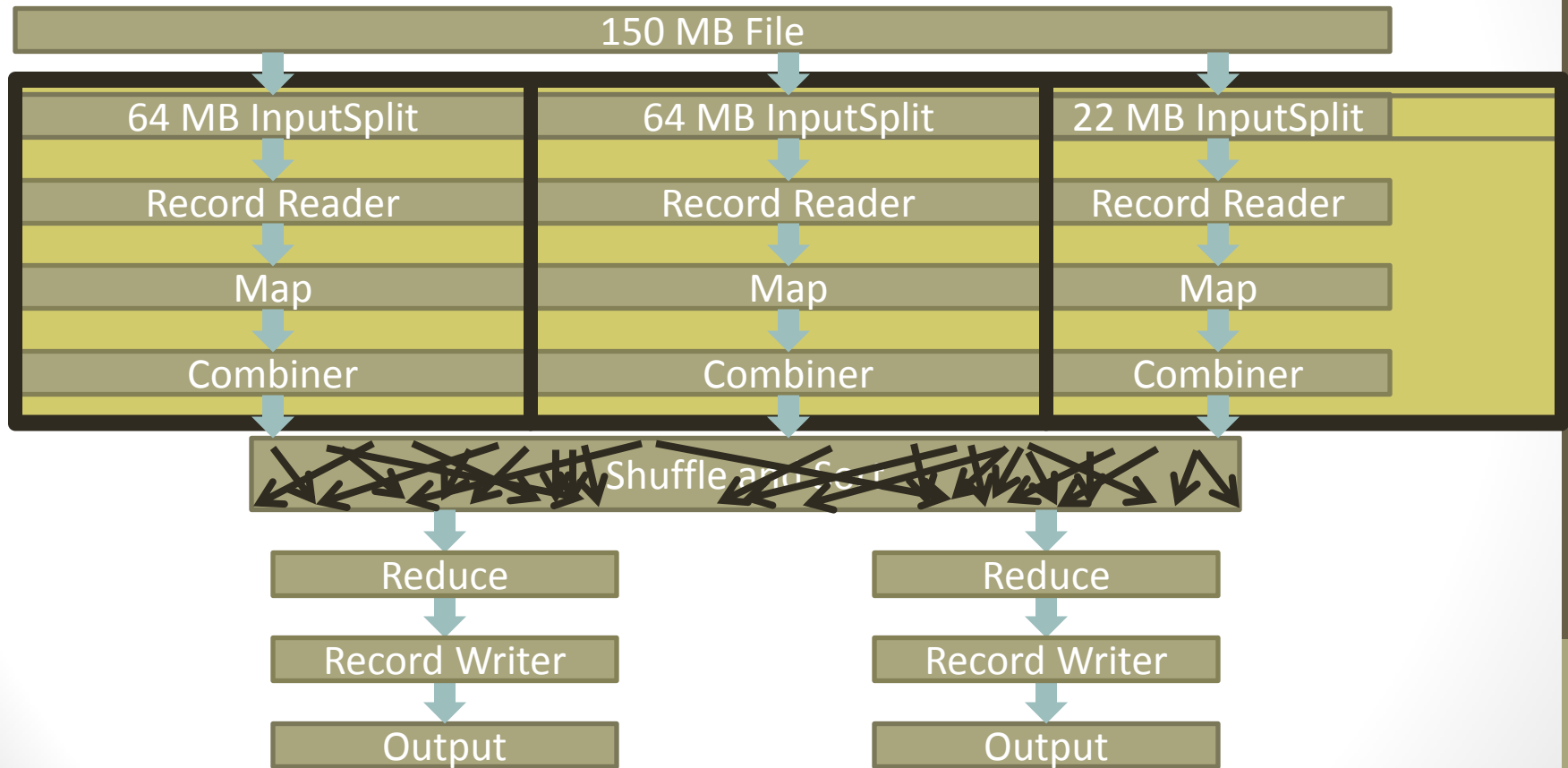
MapReduce (16)



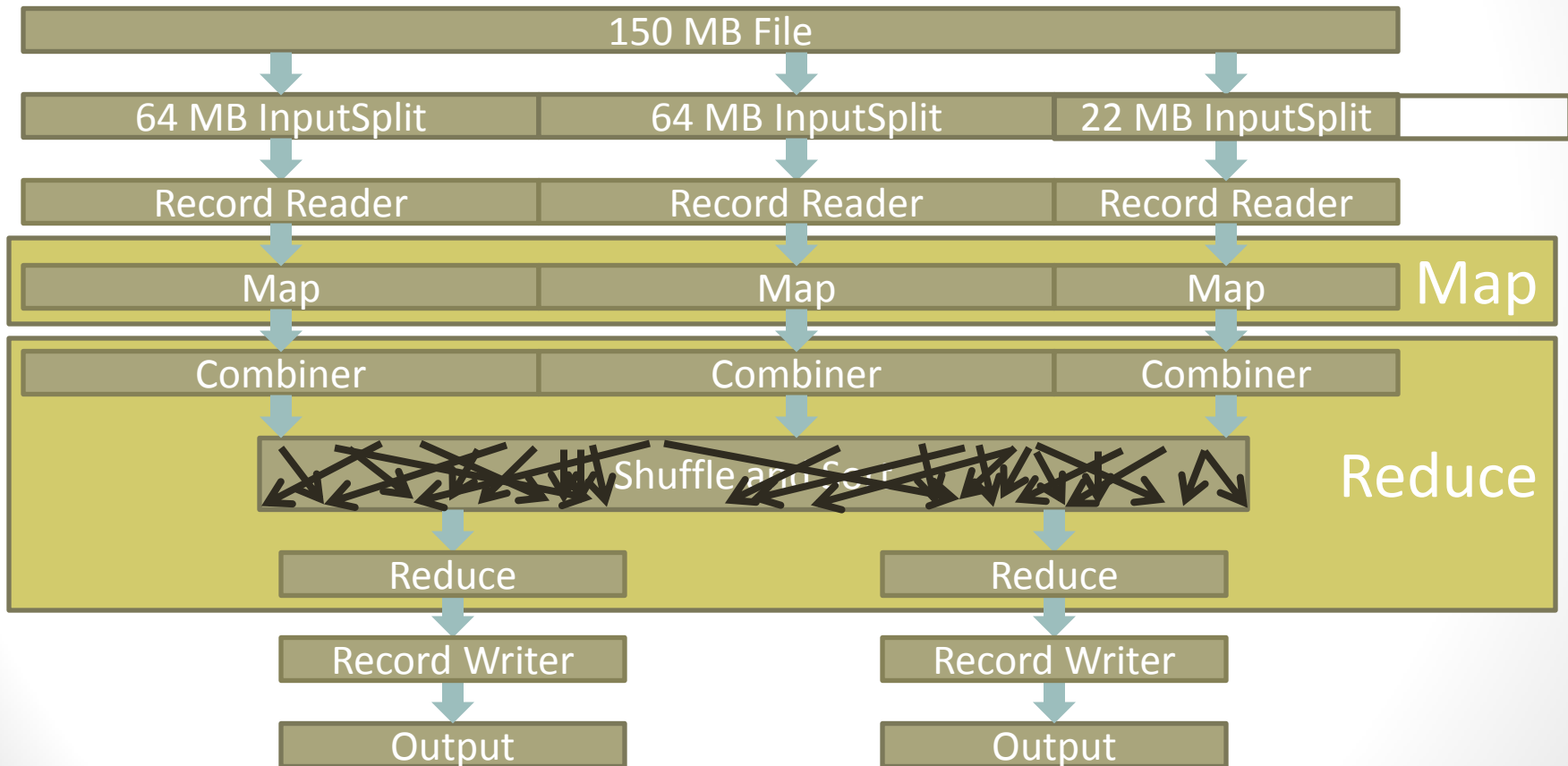
MapReduce (17)



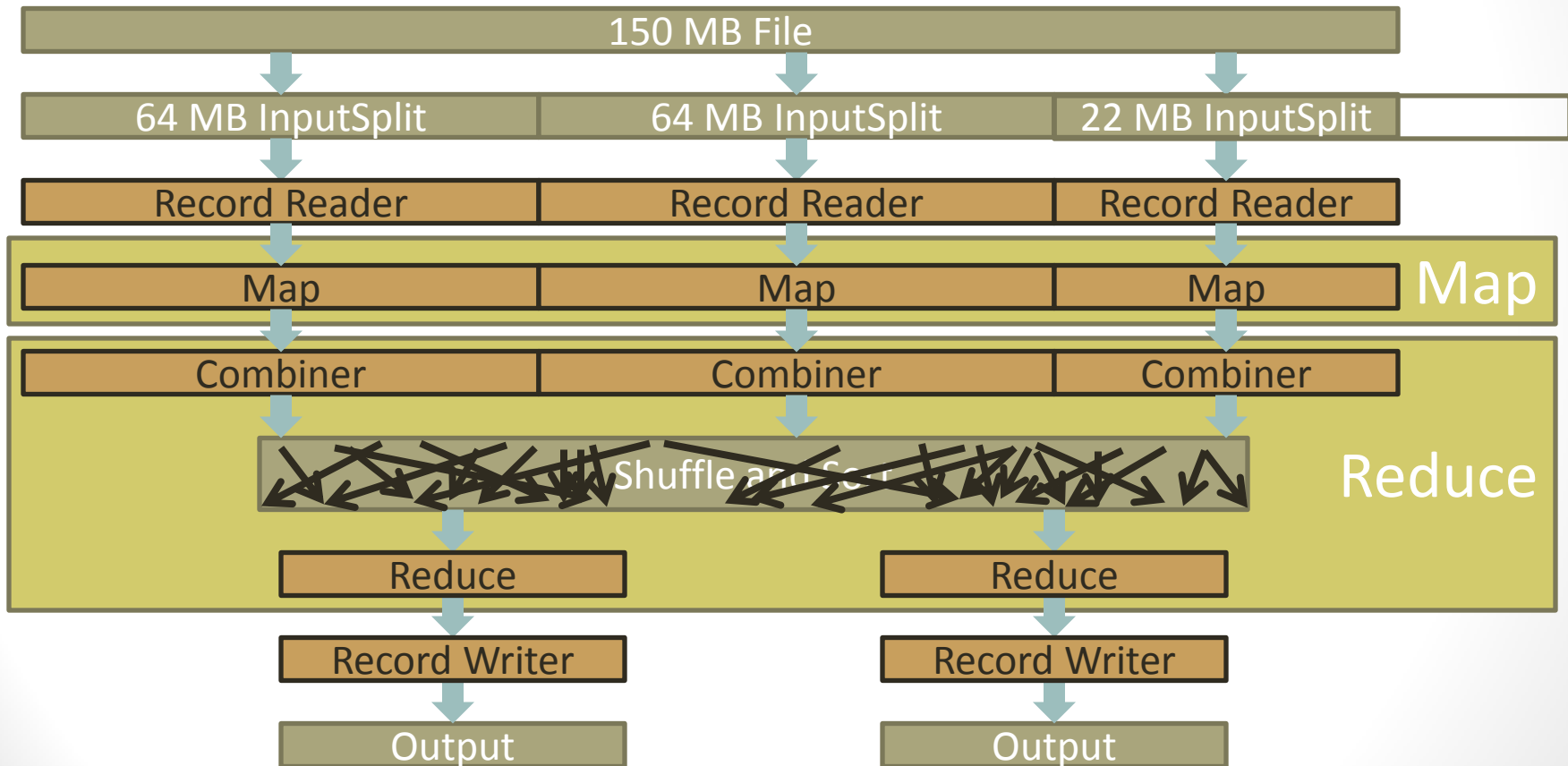
MapReduce (18)



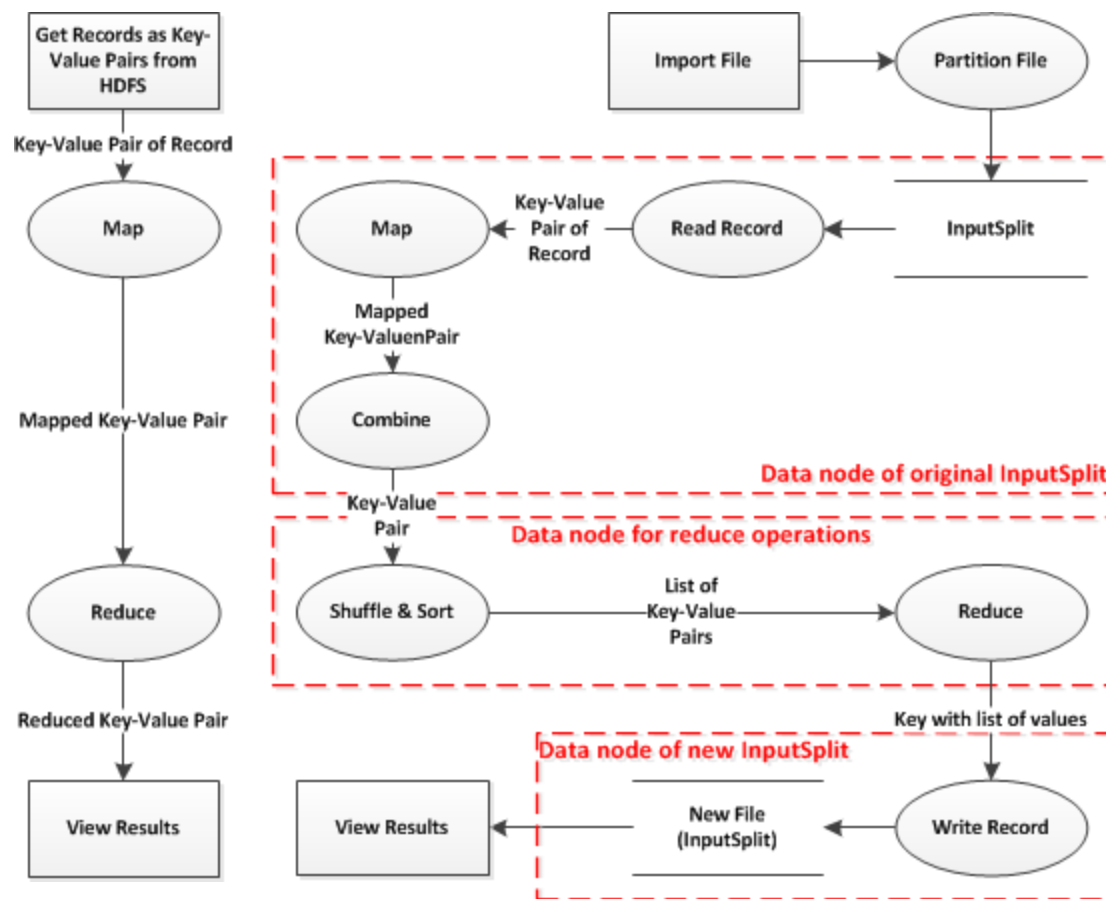
MapReduce(19)



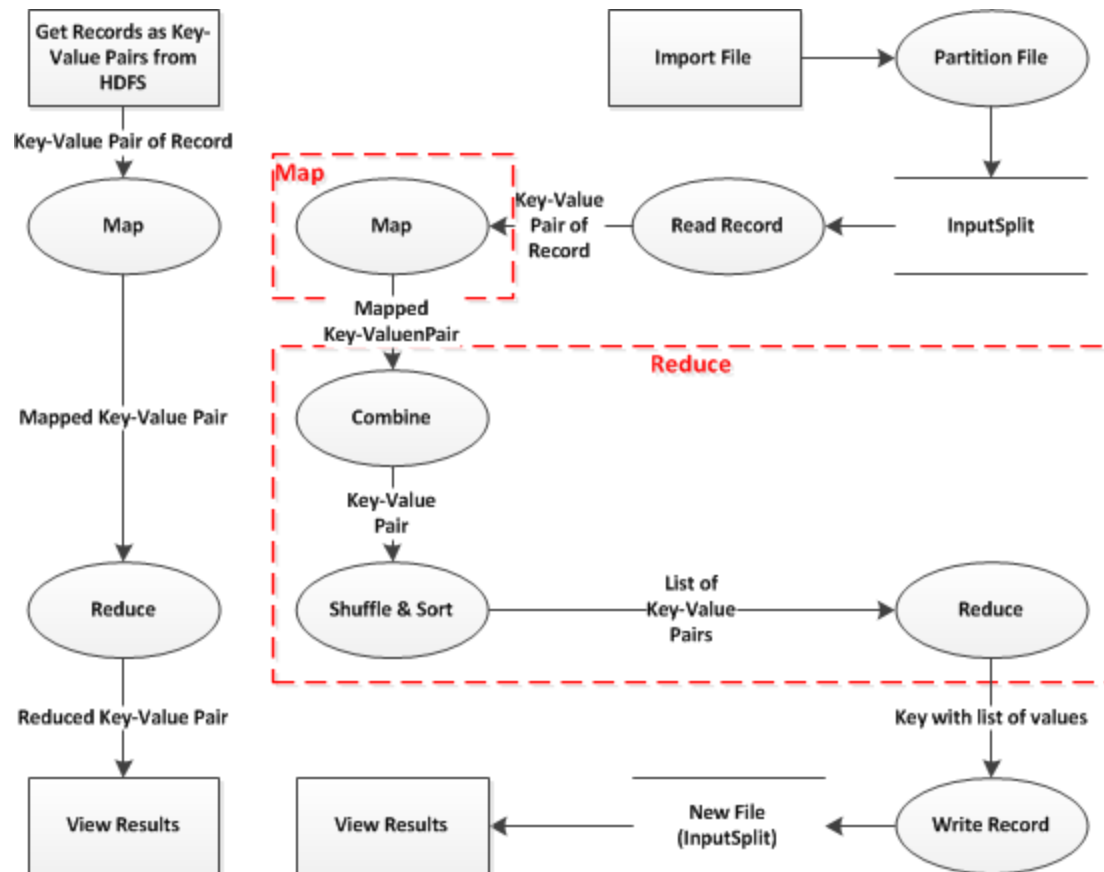
MapReduce (20)



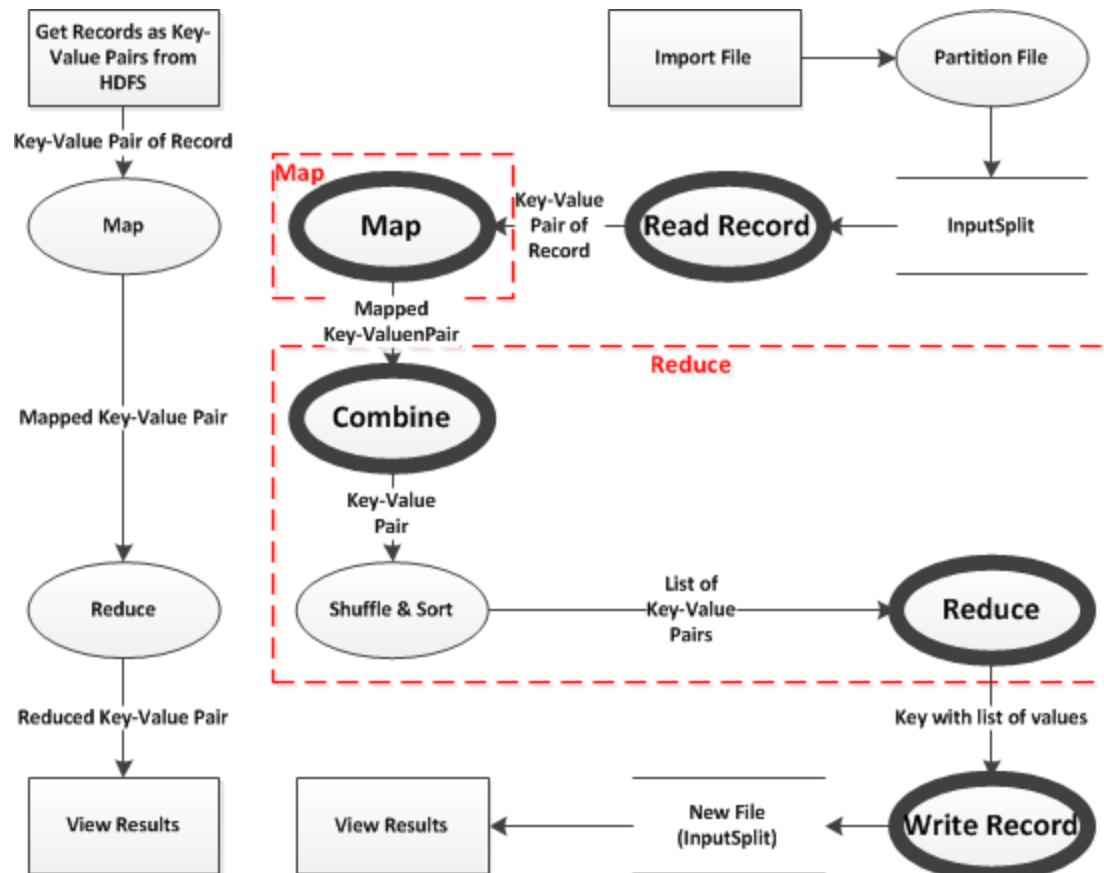
MapReduce (21)



MapReduce (22)



MapReduce (23)

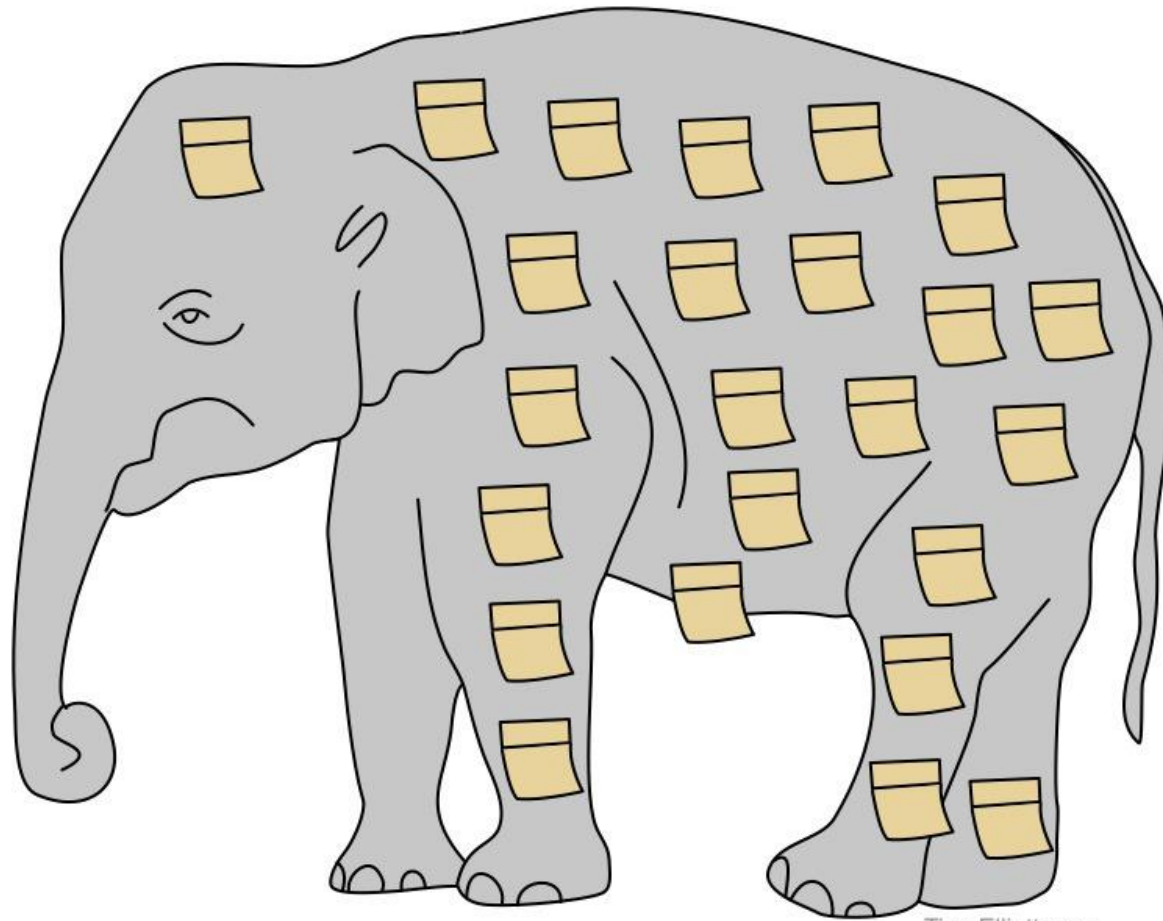


MapReduce (24)

- Record Read
 - Record $\rightarrow (K1, V1)$
 - to be or not to be $\rightarrow (0, \text{"to be or not to be"})$
- Map
 - $(K1, V1) \rightarrow \text{list}(K2, V2)$
 - $(0, \text{"to be or not to be"}) \rightarrow [(to, 1), (be, 1), (or, 1), (not, 1), (to, 1), (be, 1)]$
- Shuffle and Sort
 - $\text{list}(K2, V2) \rightarrow (K2, \text{list}(V2))$
 - $[(to, 1), (be, 1), (or, 1), (not, 1), (to, 1), (be, 1)] \rightarrow (to, [1, 1]), (be, [1, 1]), (or, [1]), (not, [1])$
- Reduce
 - $(K2, \text{list}(V2)) \rightarrow \text{list}(K3, V3)$
 - $(to, [1, 1]), (be, [1, 1]), (or, [1]), (not, [1]) \rightarrow [(to, 2), (be, 2), (or, 1), (not, 1)]$
- Record Write
 - $(K3, V3) \rightarrow \text{Records}$
 - $[(to, 2), (be, 2), (or, 1), (not, 1)] \rightarrow$

to,2
be,2
or,1
not,1

Break



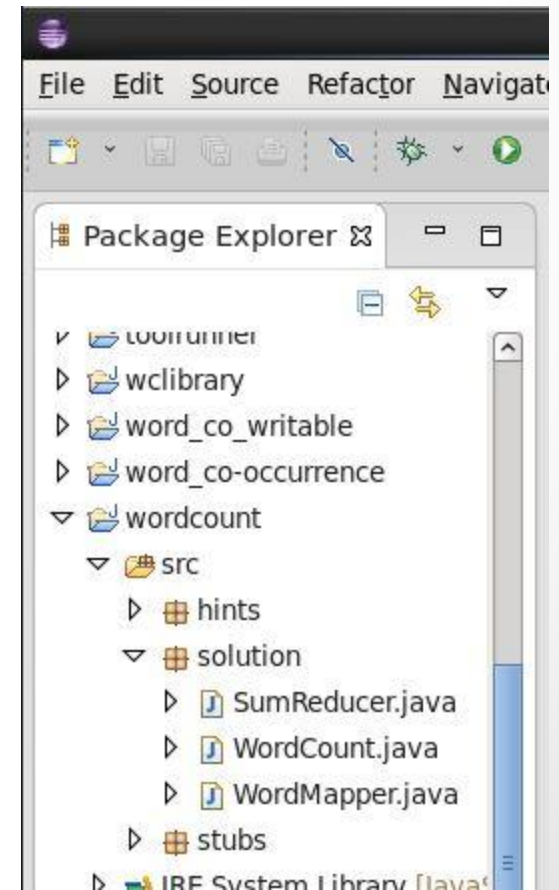
TimoElliott.com

An early Hadoop prototype...

MapReduce Lab (1)



- Open Eclipse by clicking on the Eclipse icon. Then, use the Package Explorer to navigate to wordcount.
- See the java files in wordcount\src\solution\
 - WordCount.java
 - WordMapper.java
 - SumReducer.java
- Study these java files



MapReduce Lab (2)

Input File

ByteOffset

Record

Mapper Class

```
extends Mapper<
MapperInputKeyType,
MapperInputValueType,
MapperEmitKeyType,
MapperEmitValueType>
```

```
map(
MapperInputKeyType
MapperInputKey,

MapperInputValueType
MapperInputValue,
Context context)
```

```
context.write(
MapperEmitKey,
MapperEmitValue);
```

Reducer Class

```
extends Reducer<
MapperEmitKeyType,
MapperEmitValueType,
OutputKeyType,
OutputValueType>
```

```
reduce(
MapperEmitKeyType
MapperEmitKey,
Iterable<
MapperEmitValueType>
MapperEmitValue,
Context context)
```

```
context.write(
OutputKey,
OutputValue);
```

main

```
job.setOutputKeyClass(
OutputKeyType.class);
job.setOutputValueClass(
OutputValueType.class);
```

Output File

```
OutputKey
OutputValue
```


MapReduce Lab (3)

Mapper input key is typically just the file byte offset. Ignore it.

Mapper Class

extends Mapper<

MapperInputKeyType,
MapperInputValueType,
MapperEmitKeyType,
MapperEmitValueType>

map(
MapperInputKeyType
MapperInputKey,

MapperInputValueType
MapperInputValue,
Context context)

context.write(
MapperEmitKey,
MapperEmitValue);

Input File
ByteOffset

Record

Reducer Class

extends Reducer<

MapperEmitKeyType,
MapperEmitValueType,
OutputKeyType,
OutputValueType>

reduce(
MapperEmitKeyType
MapperEmitKey,
Iterable<
MapperEmitValueType>
MapperEmitValue,
Context context)

context.write(
OutputKey,
OutputValue);

main

job.setOutputKeyClass(
OutputKeyType.class);
job.setOutputValueClass(
OutputValueType.class);

Output File

OutputKey
OutputValue

MapReduce Lab (4)

Mapper input value is typically a file record. It is of type Text.

Mapper Class

extends Mapper<
MapperInputKeyType,
MapperInputValueType,
MapperEmitKeyType,
MapperEmitValueType>

map(
MapperInputKeyType
MapperInputKey,

MapperInputValueType
MapperInputValue,
Context context)

context.write(
MapperEmitKey,
MapperEmitValue);

Reducer Class

extends Reducer<
MapperEmitKeyType,
MapperEmitValueType,
OutputKeyType,
OutputValueType>

reduce(
MapperEmitKeyType
MapperEmitKey,
Iterable<
MapperEmitValueType>
MapperEmitValue,
Context context)

context.write(
OutputKey,
OutputValue);

main

job.setOutputKeyClass(
OutputKeyType.class);
job.setOutputValueClass(
OutputValueType.class);

Input File
ByteOffset

Record



Output File
OutputKey
OutputValue

MapReduce Lab (5)

Mapper output or emit key is the same the reducer input key.

Mapper Class

extends Mapper<
MapperInputKeyType,
MapperInputValueType,
MapperEmitKeyType,
MapperEmitValueType>

map(
MapperInputKeyType
MapperInputKey,

MapperInputValueType
MapperInputValue,
Context context)

context.write(
MapperEmitKey,
MapperEmitValue);

Input File
ByteOffset

Record

Reducer Class

extends Reducer<
MapperEmitKeyType,
MapperEmitValueType,
OutputKeyType,
OutputValueType>

reduce(
MapperEmitKeyType
MapperEmitKey,
Iterable<
MapperEmitValueType>
MapperEmitValue,
Context context)

context.write(
OutputKey,
OutputValue);

main

job.setOutputKeyClass(
OutputKeyType.class);
job.setOutputValueClass(
OutputValueType.class);

Output File
OutputKey
OutputValue

MapReduce Lab (6)

Mapper output or emit value is the reducer input value.

Mapper Class

extends Mapper<
MapperInputKeyType,
MapperInputValueType,
MapperEmitKeyType,
MapperEmitValueType>

map(
MapperInputKeyType
MapperInputKey,

MapperInputValueType
MapperInputValue,
Context context)

context.write(
MapperEmitKey,
MapperEmitValue)
;

Reducer Class

extends Reducer<
MapperEmitKeyType,
MapperEmitValueType,
OutputKeyType,
OutputValueType>

reduce(
MapperEmitKeyType
MapperEmitKey,
Iterable<
MapperEmitValueType>
MapperEmitValue,
Context context)

context.write(
OutputKey,
OutputValue);

main

job.setOutputKeyClass(
OutputKeyType.class);
job.setOutputValueClass(
OutputValueType.class);

Input File
ByteOffset

Record

Output File
OutputKey
OutputValue

MapReduce Lab (7)

Reducer output or emit key is the output key from main and the key of the Output File

Mapper Class

```
extends Mapper<
MapperInputKeyType,
MapperInputValueType,
MapperEmitKeyType,
MapperEmitValueType>
```

```
map(
MapperInputKeyType
MapperInputKey,

MapperInputValueType
MapperInputValue,
Context context)
```

```
context.write(
MapperEmitKey,
MapperEmitValue);
```

Input File
ByteOffset

Record

Reducer Class

```
extends Reducer<
MapperEmitKeyType,
MapperEmitValueType,
OutputKeyType,
OutputValueType>
```

```
reduce(
MapperEmitKeyType
MapperEmitKey,
Iterable<
MapperEmitValueType>
MapperEmitValue,
Context context)
```

```
context.write(
OutputKey,
OutputValue);
```

main

```
job.setOutputKeyClass(
OutputKeyType.class);
job.setOutputValueClass(
OutputValueType.class);
```

Output File
OutputKey
OutputValue

MapReduce Lab (8)

Reducer output or emit value is the output value from main and the value of the Output File

Input File
ByteOffset

Record

Mapper Class

extends Mapper<
MapperInputKeyType,
MapperInputValueType,
MapperEmitKeyType,
MapperEmitValueType>

map(
MapperInputKeyType
MapperInputKey,

MapperInputValueType
MapperInputValue,
Context context)

context.write(
MapperEmitKey,
MapperEmitValue);

Reducer Class

extends Reducer<
MapperEmitKeyType,
MapperEmitValueType,
OutputKeyType,
OutputValueType>

reduce(
MapperEmitKeyType
MapperEmitKey,
Iterable<
MapperEmitValueType>
MapperEmitValue,
Context context)

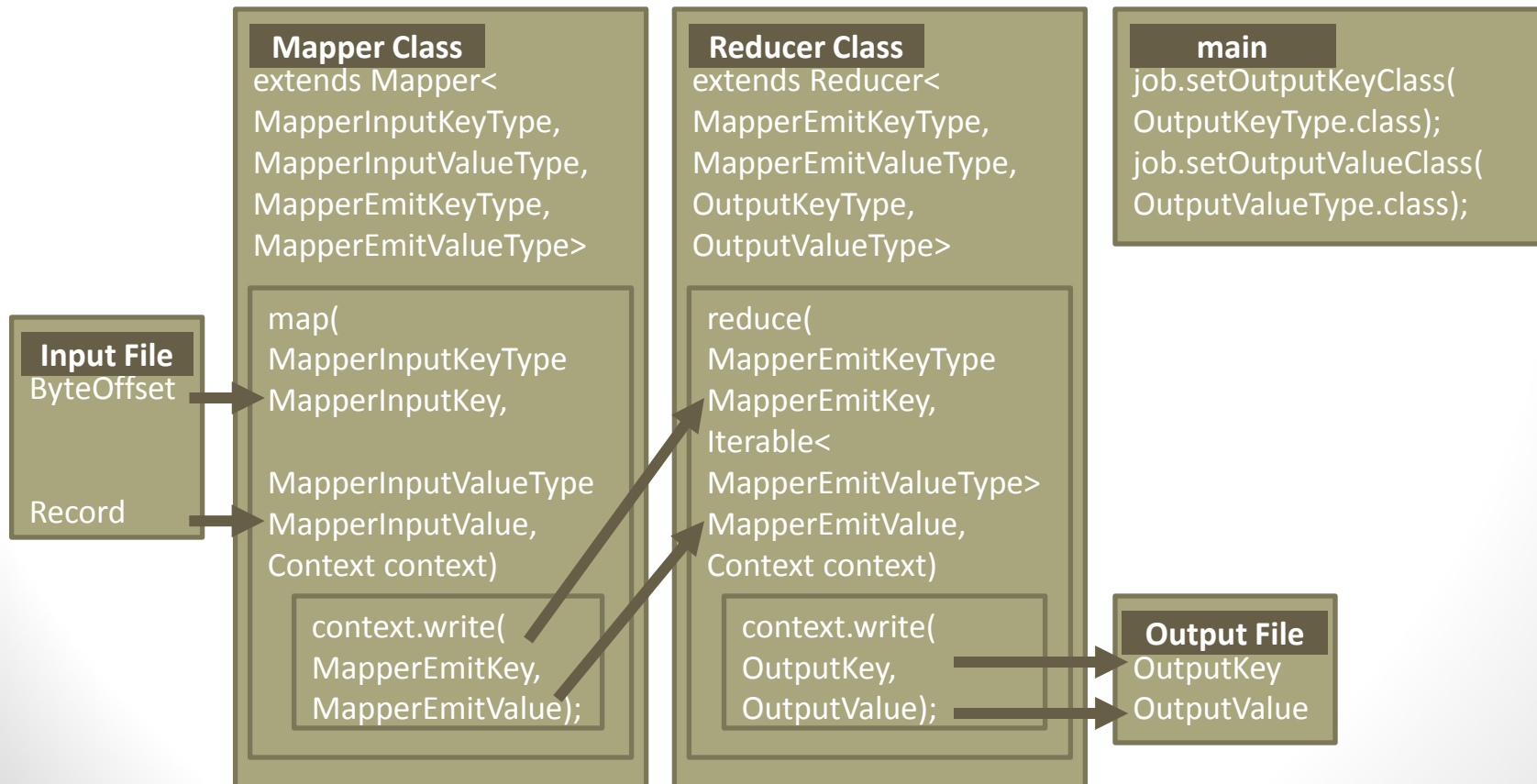
context.write(
OutputKey,
OutputValue);

main

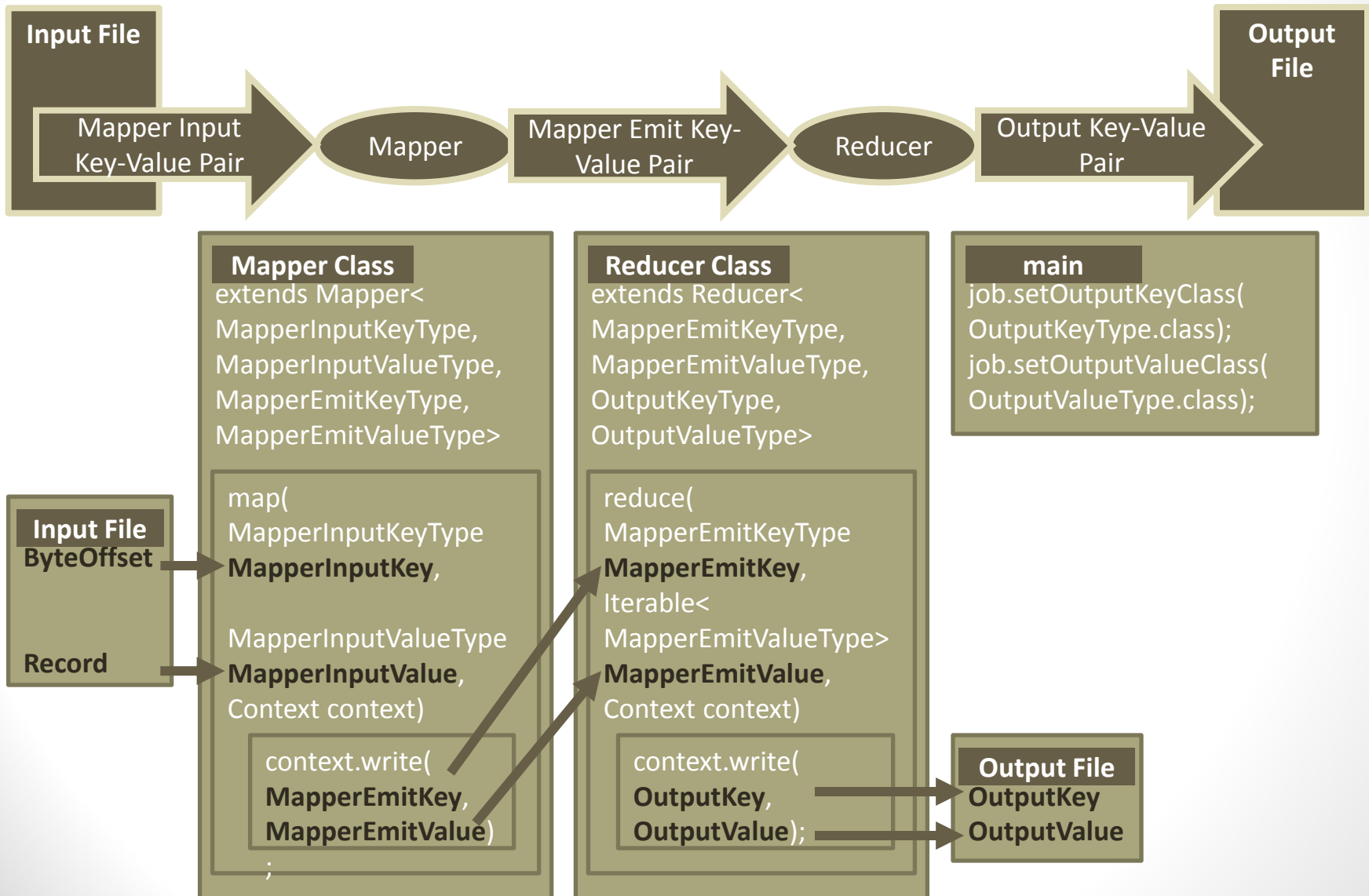
job.setOutputKeyClass(
OutputKeyType.class);
job.setOutputValueClass(
OutputValueType.class);

Output File
OutputKey
OutputValue

MapReduce Lab (9)



MapReduce Lab (10)



MapReduce Lab (11)



Open a terminal

In the terminal change the current directory:

- `$ cd ~/workspace/wordcount/src`

List the directories and files in the current directory:

- `$ ls`

List the files in the solution directory:

- `$ ls solution`

Show the classpath to get the Hadoop Jar and other required libraries:

- `$ hadoop classpath`

Compile the java files to class files:

- `$ javac -classpath `hadoop classpath` solution/*.java`

List the files in the solution directory (note the new class files):

- `$ ls solution`

MapReduce Lab (12)



In the terminal:

Create a JAR file containing the class files:

- `$ jar cvf wordcount.jar solution/*.class`

List the files in the current directory (note the new JAR file):

- `$ ls`

In Hadoop, run the wordcount program on all of Shakespeare's works:

- `$ hadoop jar wordcount.jar solution.WordCount shakespeare wordcounts`

In HDFS list the files that contain the results:

- `$ hadoop fs -ls wordcounts`

In HDFS list the results:

- `$ hadoop fs -cat wordcounts/part-r-00000 | less`

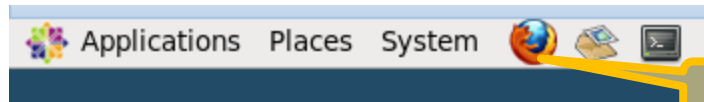
In Hadoop, run the wordcount program on Shakespeare's poems:

- `$ hadoop jar wc.jar solution.WordCount shakespeare/poems pwords`

MapReduce Lab (13)

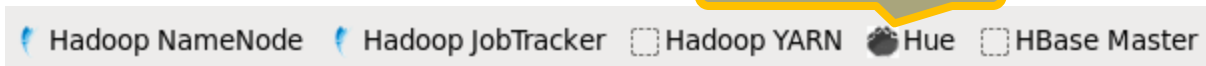
- Use Hue's HDFS file browser to browse and read the results

1. Open Firefox



Click Here

2. Start Hue by clicking on the Hue link in the Bookmarks toolbar (Username: training, Password: training)



Click Here

3. Start File Browser (Username: training, Password: training)



Click Here

4. Find wordcounts/part-r-00000



Click Here

5. View Contents of part-r-00000

<input type="checkbox"/>	Type	Name
<input type="checkbox"/>	Folder	..
<input type="checkbox"/>	File	_SUCCESS
<input type="checkbox"/>	Folder	_logs
<input type="checkbox"/>	File	part-r-00000

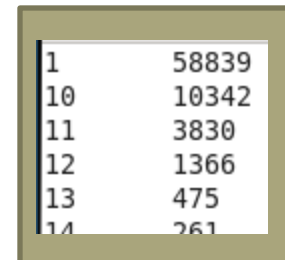
Click Here

MapReduce Lab (14)

- The next two slides describe the assignment for this week.

MapReduce Lab (15)

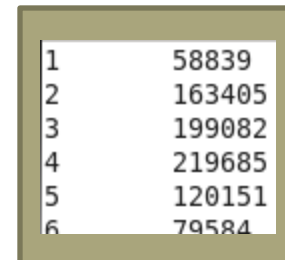
- Lab: Word Length Count 1
- This is part of today's homework assignment.
- Create a MapReduce program that lists word lengths and how often words of that length occur. Modify package wordcount.solution:
 - Calculate word length (`word.length()`) and cast to a Text.
 - Change map method to use the Text word length as the key.
 - Compile and run the program on shakespeare's works:
 - `hadoop jar wordcount.jar solution.WordCount shakespeare lengthcounts1`
 - Copy the modified java files (E.g. `MyFile.java` -> `MyFile_1.java`)
 - View the results using the file browser
 - `hadoop fs -cat lengthcounts1/part-r-00000 | less`
 - Take a screen shot of the results (approx. 18 rows). Similar to this:



1	58839
10	10342
11	3830
12	1366
13	475
14	261

MapReduce Lab (16)

- Lab: Word Length Count 2
- This is part of today's homework assignment.
- Create a MapReduce program that lists word lengths and how often words of that length occur. Modify package wordcount.solution:
 - Calculate word length (`word.length()`) and cast to `IntWritable`
 - Change map method to use the word length as the key
 - Compile and run the program on shakespeare's works:
 - `hadoop jar wordcount.jar solution.WordCount shakespeare lengthcounts2`
 - Copy the modified java files (E.g. `MyFile.java` -> `MyFile_2.java`)
 - View the results using the file browser
 - `hadoop fs -cat lengthcounts2/part-r-00000 | less`
 - Take a screen shot of the results (approx. 18 rows). Similar to this:



1	58839
2	163405
3	199082
4	219685
5	120151
6	79584



Hadoop-2 Labs

Assignment

Assignment

1. Comment in our LinkedIn Group on a discussion started by yourself or a fellow student about statistics. Copy the post to a text file called post.txt. This assignment item is similar to last week's assignment.
2. Create a MapReduce program that lists word lengths next to the number of times a word of that length occurs. The word lengths are sorted as texts, even though they are numbers. Specifically, modify package wordcount.solution as described in the MapReduce Lab for "Word Length Count 1". In this lab, the output keys are of type text. Save all code of the modified java file(s) and a screen shot of all the results (approx. 18 rows).
3. Create a MapReduce program that lists word lengths next to the number of times a word of that length occurs. The word lengths are sorted numerically. Specifically, modify package wordcount.solution as described in the MapReduce Lab for "Word Length Count 2". In this lab, the output keys are of type IntWritable (numeric). Save all code of the modified java file(s) and a screen shot of all the results (approx. 18 rows).
4. Submit to Canvas by Saturday 11:57 PM the following: post.txt. The code as *.java files from "Word Length Count 1". The screenshot of the results from "Word Length Count 1". The code as *.java files from "Word Length Count 2". The screenshot of the results from "Word Length Count 2".
5. Look through the Preview section, especially the SPARQL Exercises, prior to next week's lecture. (There will be a quiz).

Assignment

Introduction to Data Science