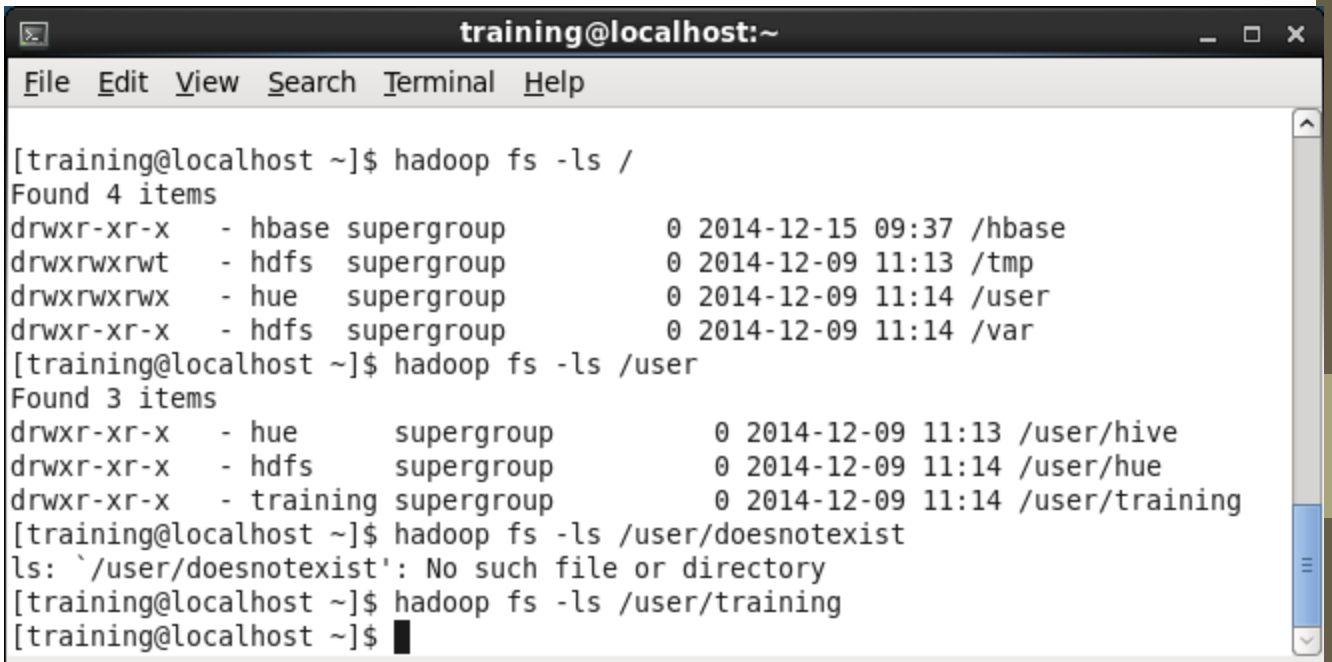
Hadoop-2 Labs

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%."

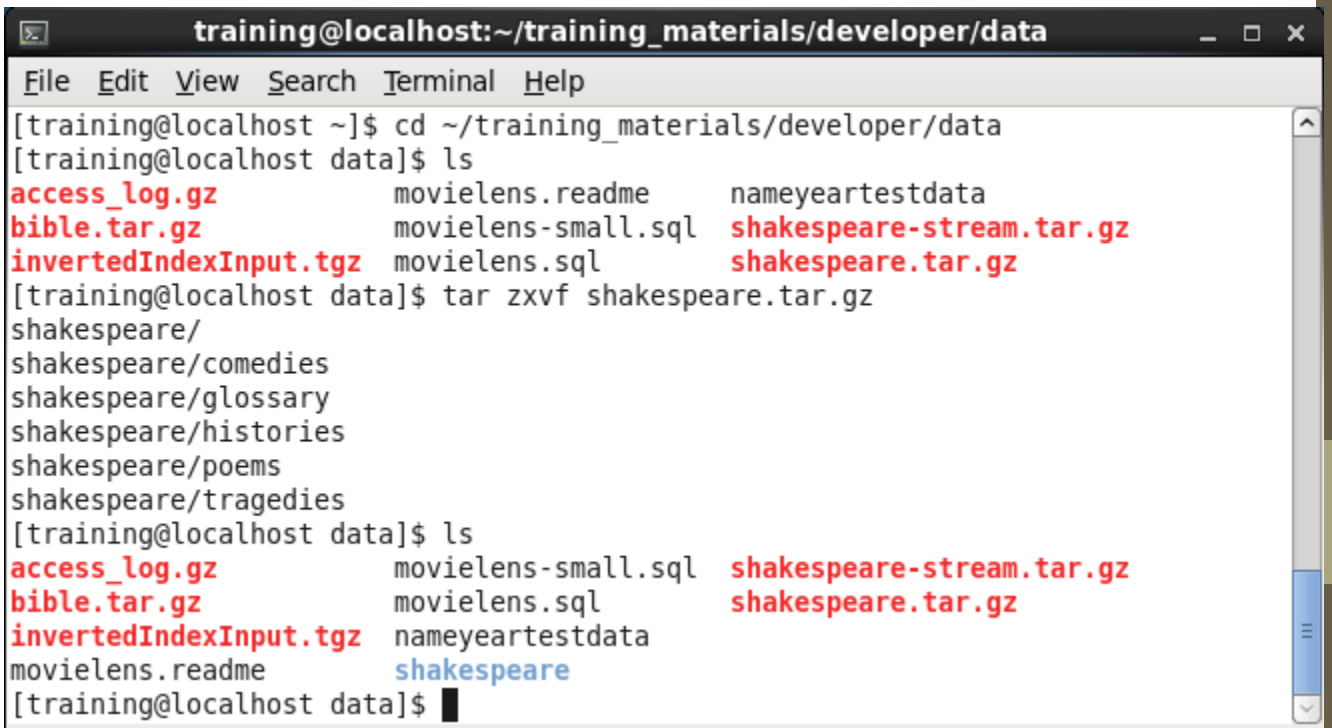
HDFS Lab (1)

- 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.



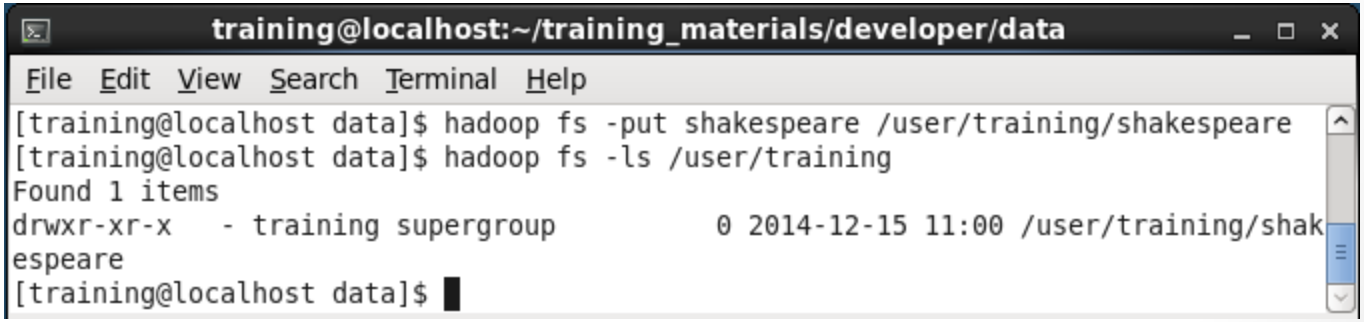
HDFS Lab (2)

- 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`**



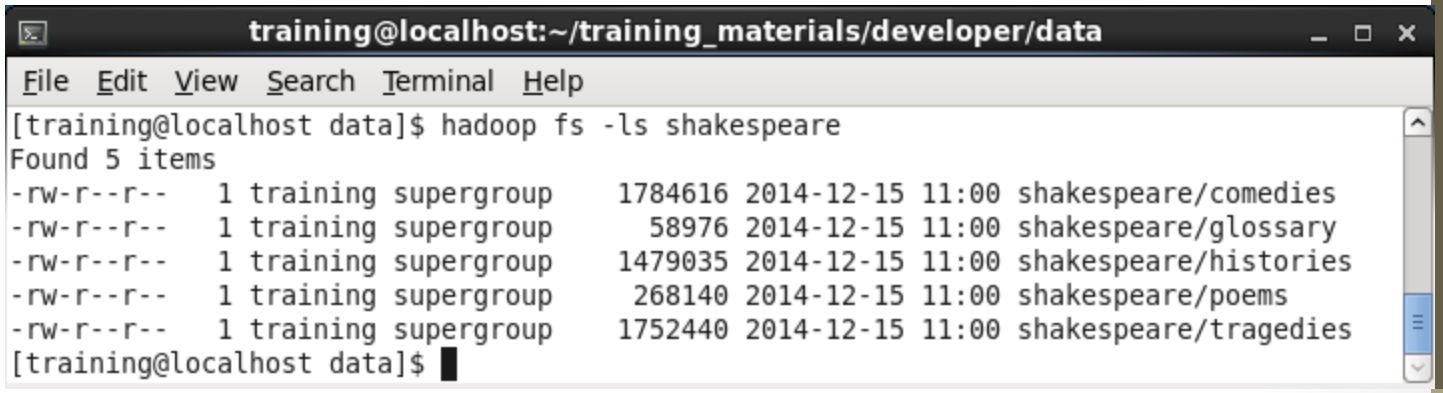
HDFS Lab (3)

- 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`



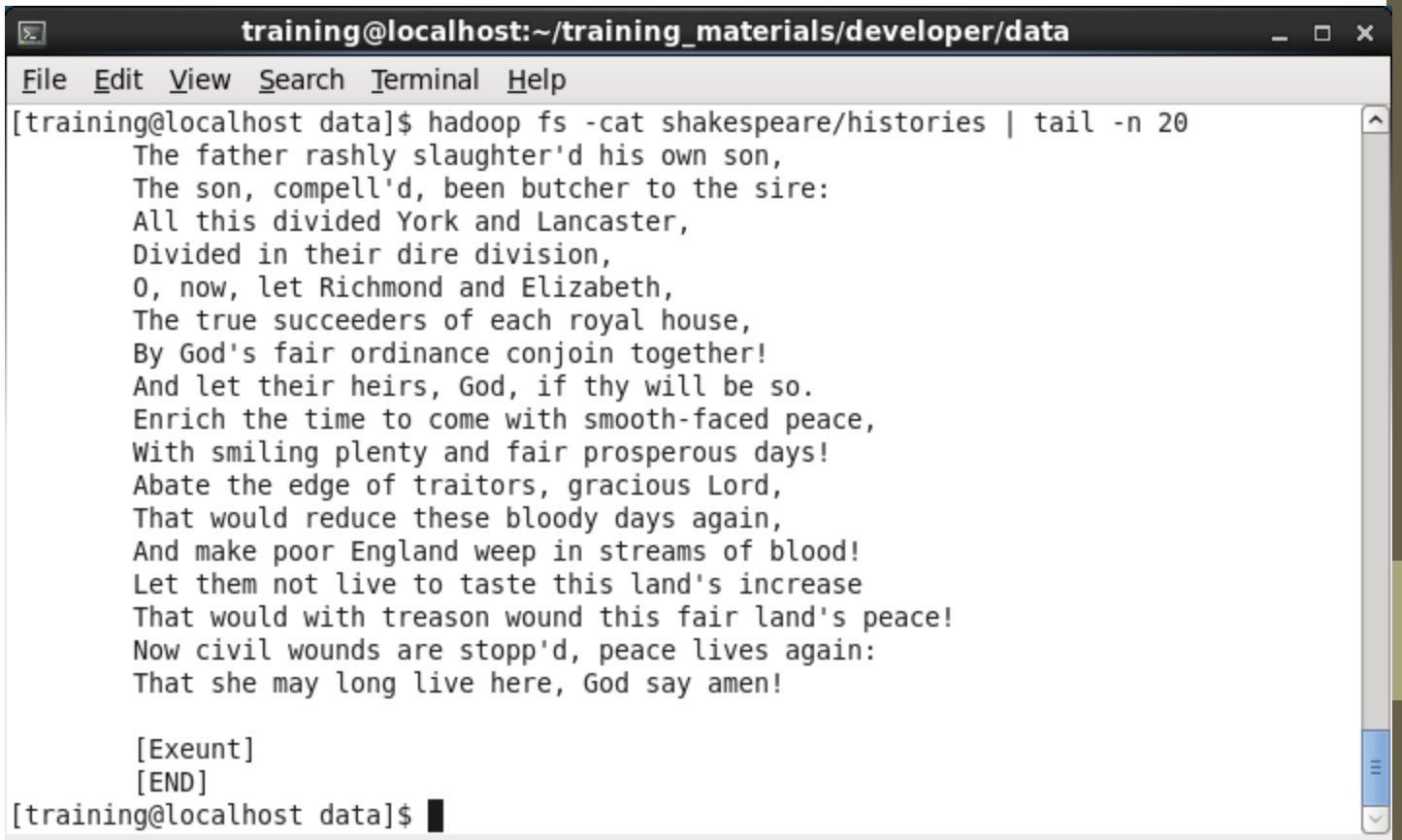
HDFS Lab (4)

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



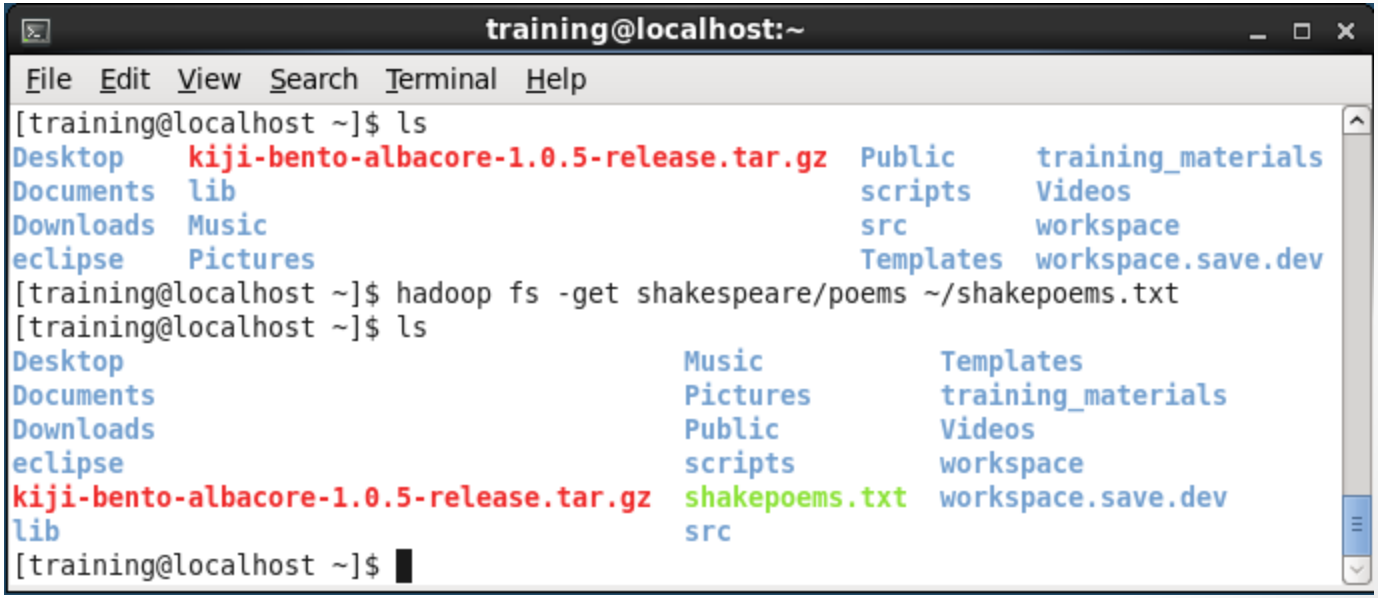
HDFS Lab (5)

- 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`



HDFS Lab (6)

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



Sqoop Lab

- **Sq**l + Had**oop** → **Sqoop**
- Use Sqoop to share between a RDBMS and Hadoop

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 data from a mysql table to an HDFS table. 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**

To run this command a second time you will need to avoid a naming collision:
sqoop import --connect jdbc:mysql://localhost/movielens --username training --password training --fields-terminated-by '\t' --table movie --target-dir /movie2

Sqoop Lab (3)

- Use this Sqoop command to import data from a mysql table to an HDFS table. 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`**

To run this command a second time you will need to avoid a naming collision:
sqoop import --connect jdbc:mysql://localhost/movielens --username training --password training --fields-terminated-by '\t' --table movierating --target-dir /movierating2

Hive Lab

"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';`

To run this command a second time you will need to avoid a naming collision:
hive> DROP TABLE **movie**
hive> DROP TABLE **movie**rating

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;**

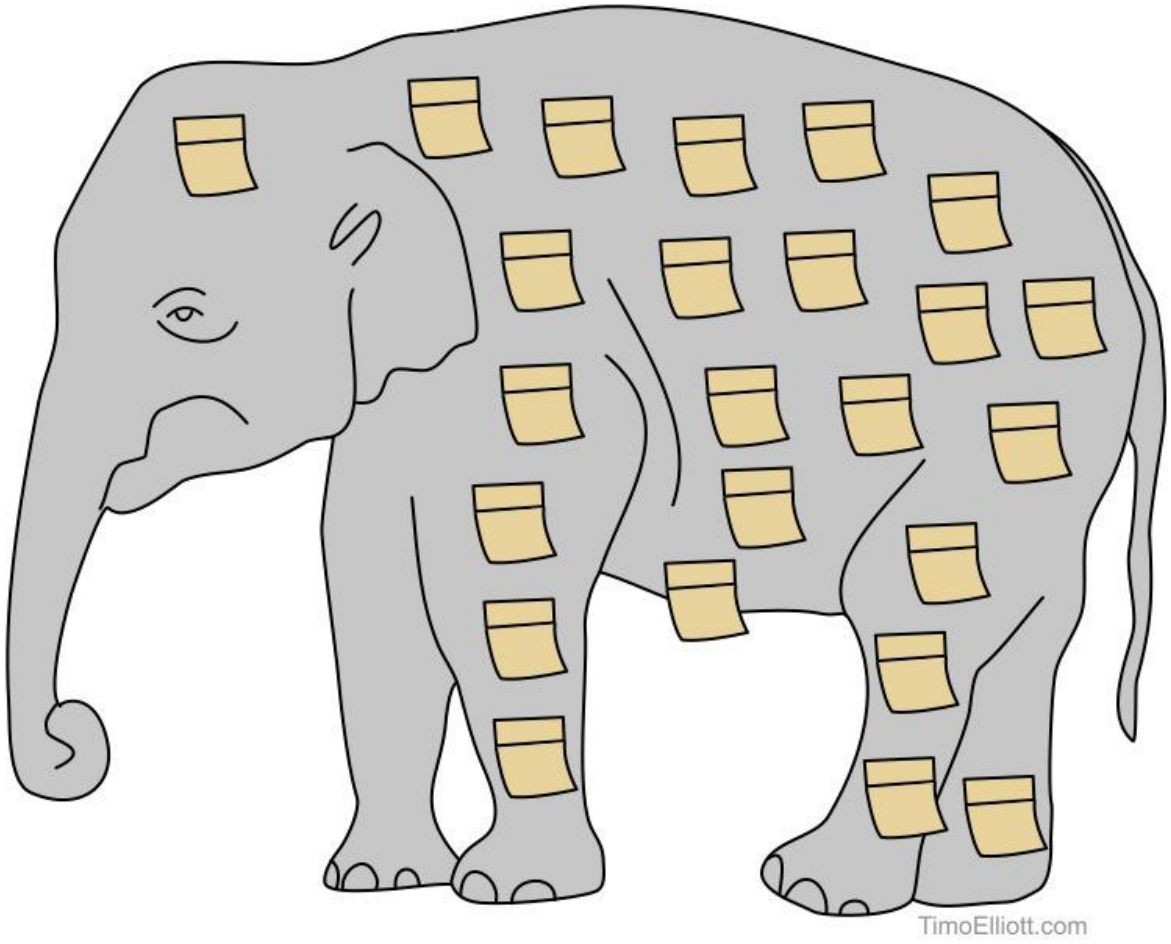
Hive Lab (4)

- 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

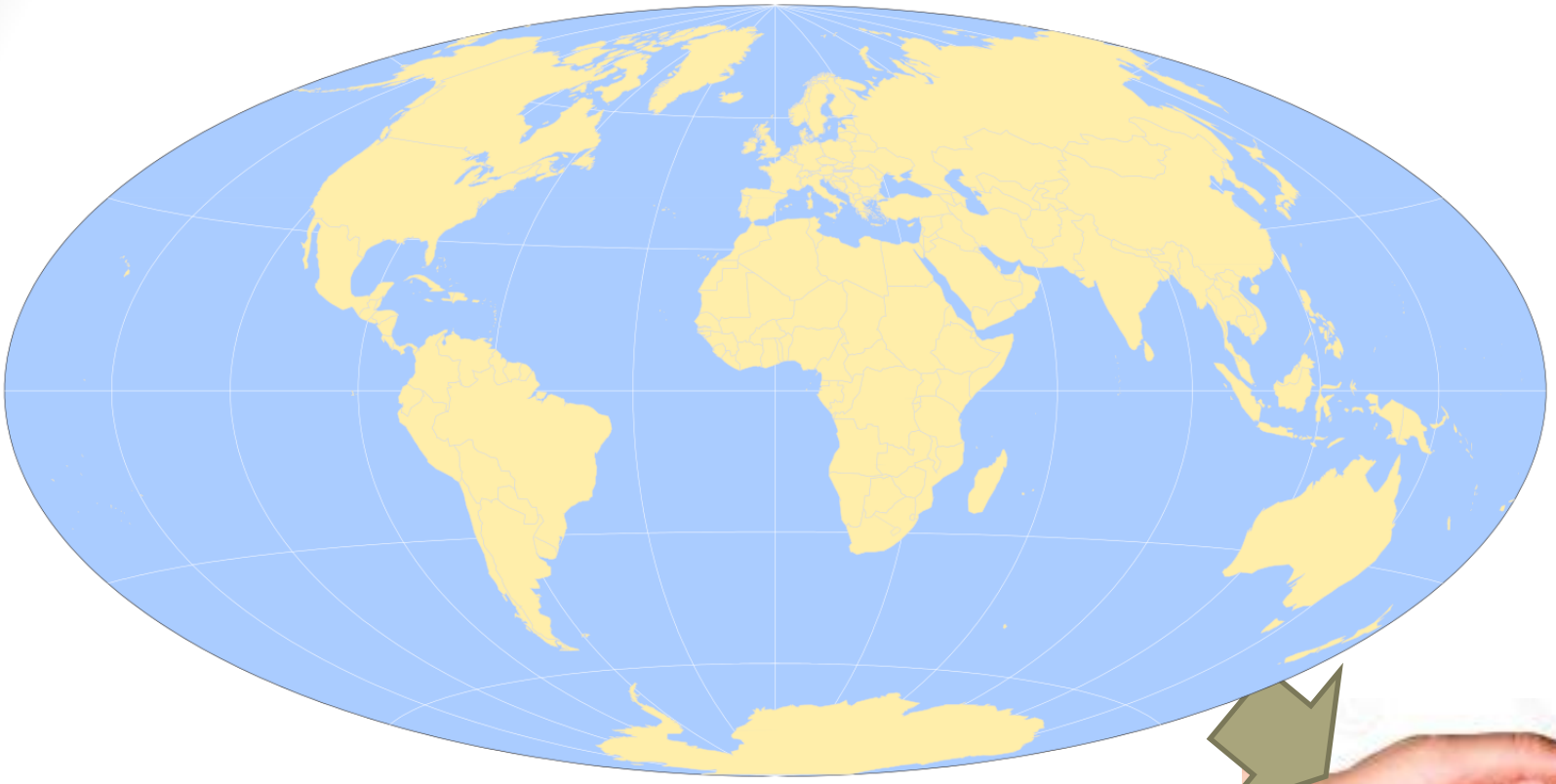
- Use Impala to execute “HiveQL”
  1. Open Firefox
  2. Start Hue by clicking on the Hue link in the Bookmarks toolbar (Username: training, Password: training)
  3. Select Impala and Query Editor Tab
  4. Enter Query into Query text box for Impala:
     - `select name, NumRatings , AvgRating from MovieRated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

Break



An early Hadoop prototype...

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)

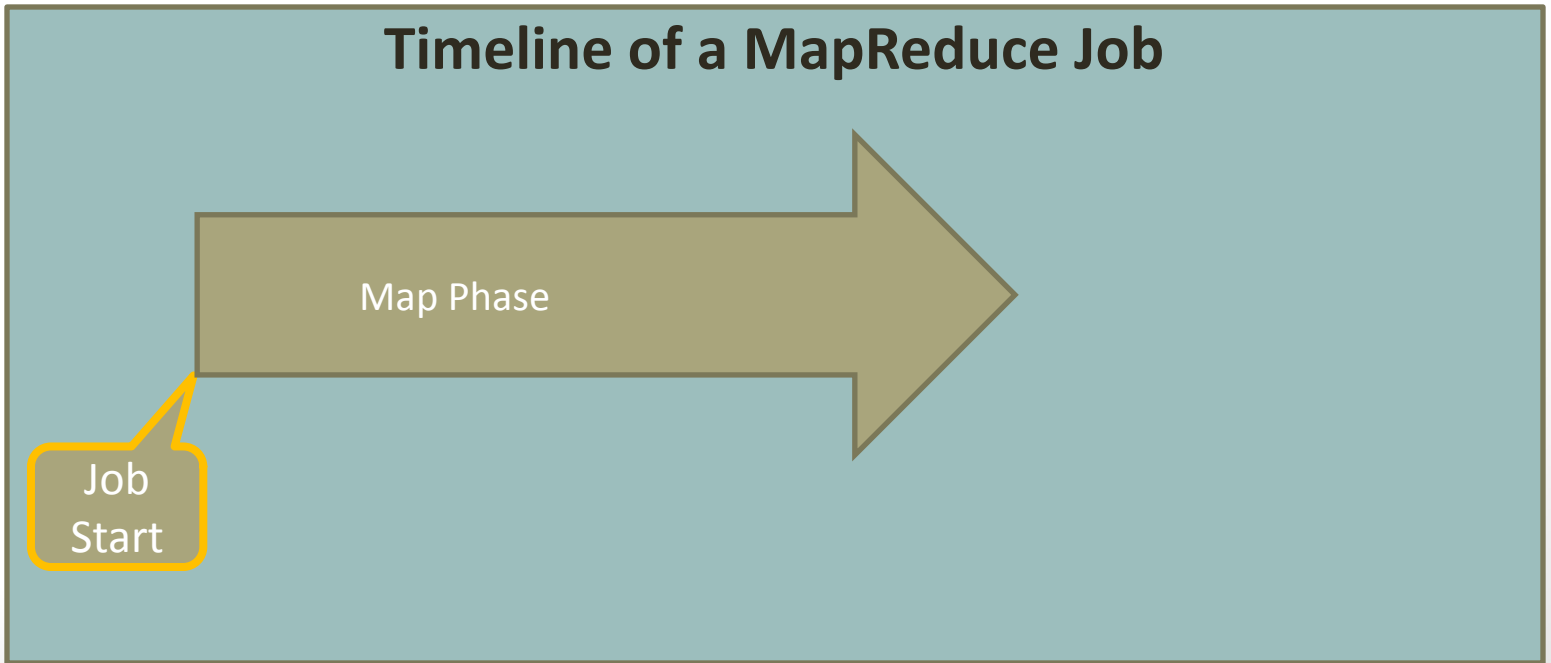
- The word MapReduce is made from **Map** and **Reduce**
- When a MapReduce job is started, then Hadoop sends **Map** and **Reduce** jobs to the data nodes. Hadoop manages all the details of data passing among data nodes.
  - The **Map** portion of the computation occurs on the individual data nodes 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

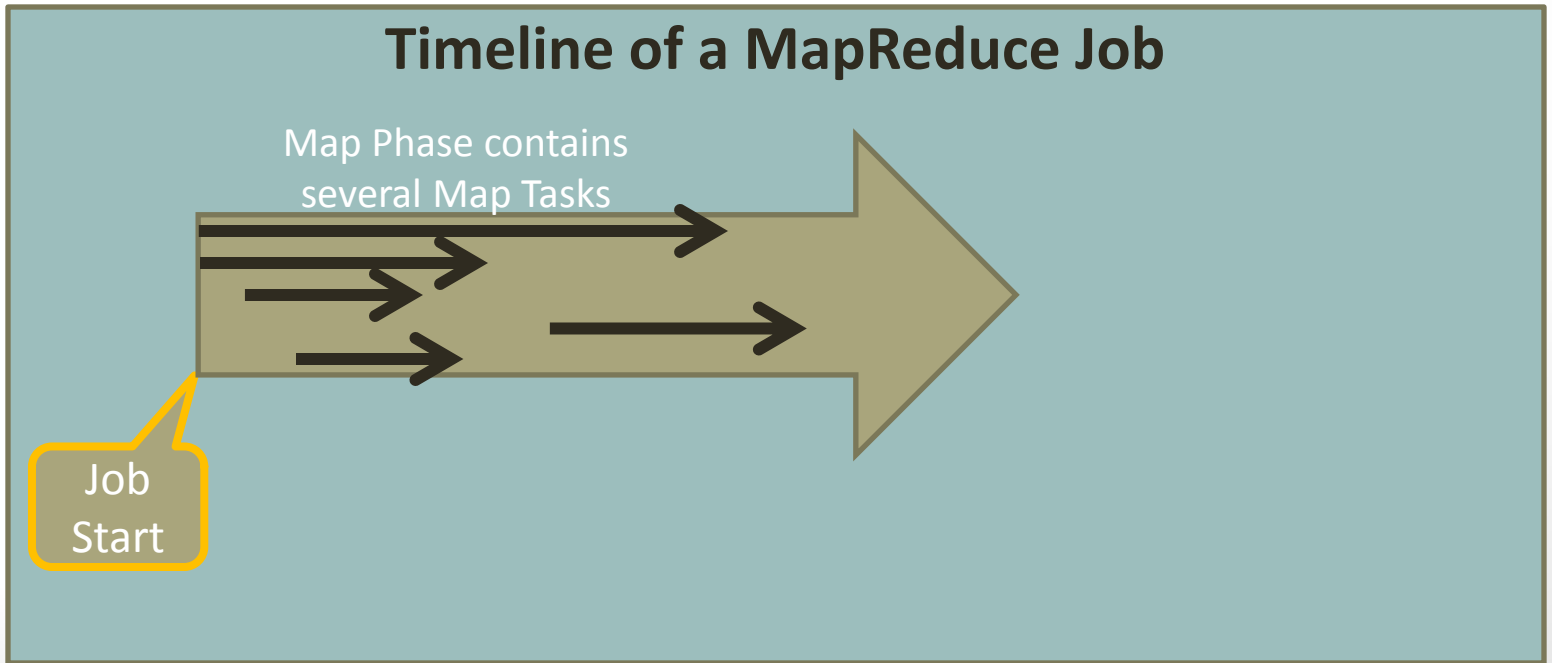
MapReduce (4)

- Parallelization Rules in MapReduce



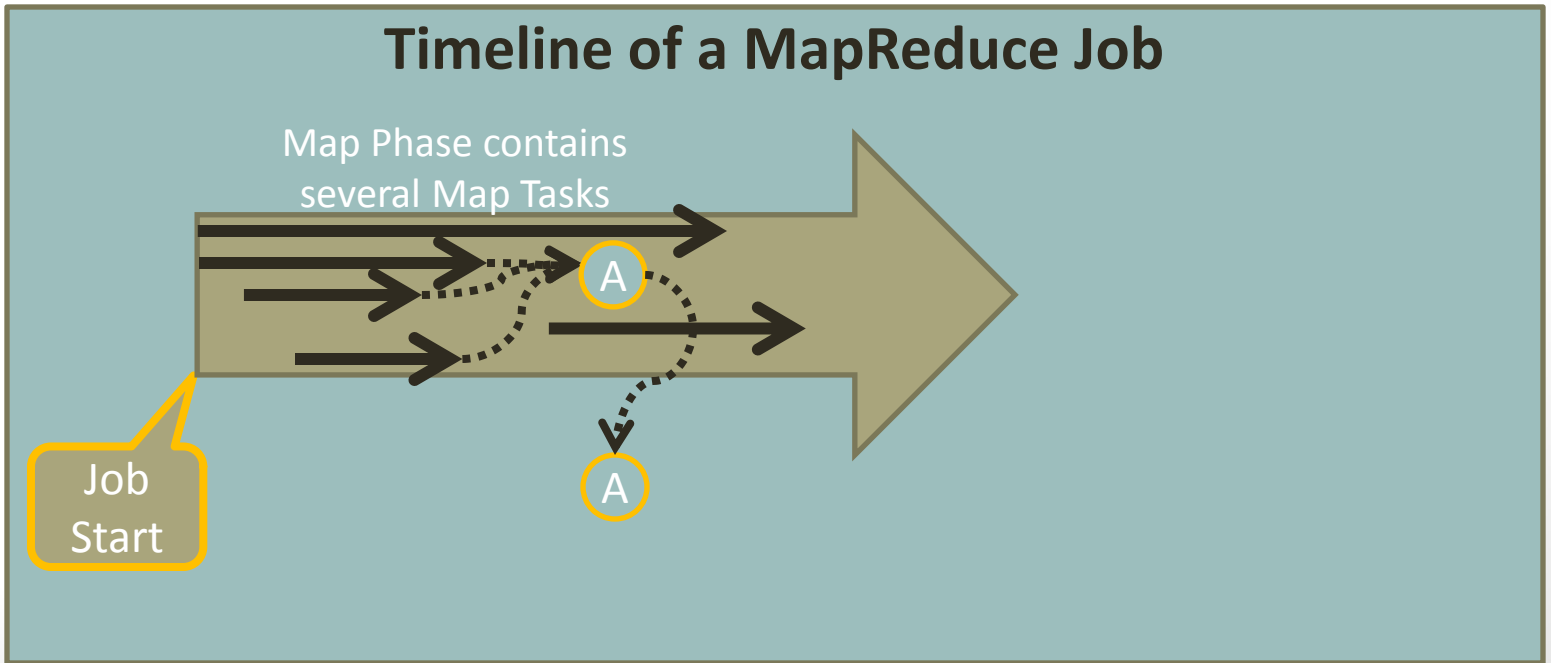
MapReduce (5)

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



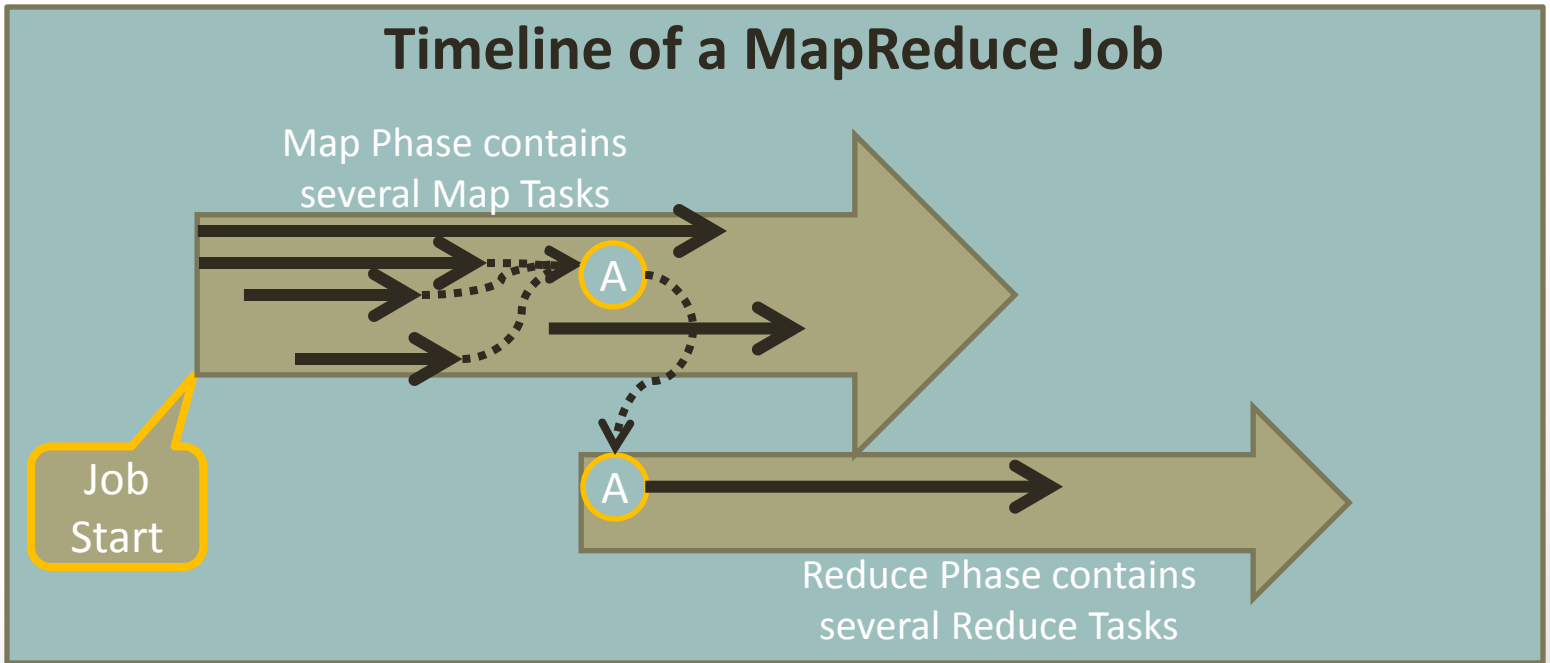
MapReduce (6)

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



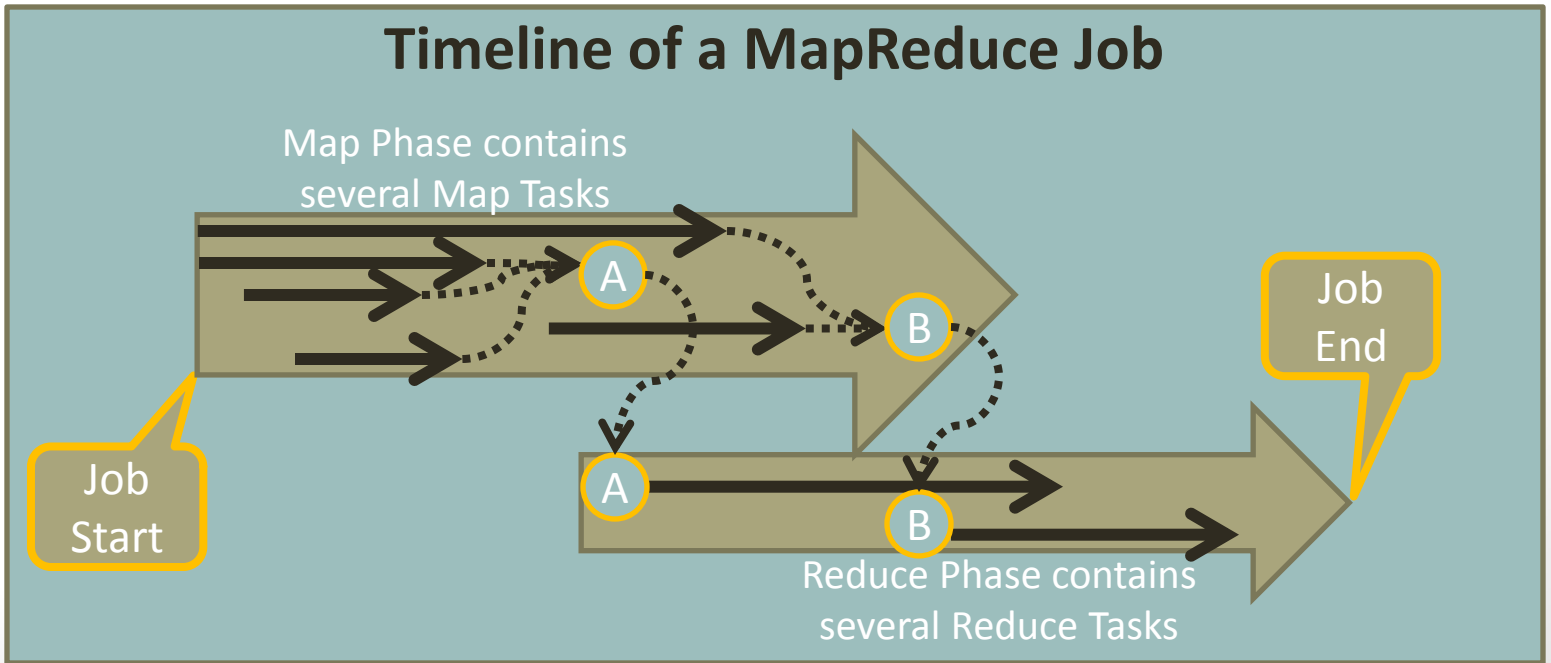
MapReduce (7)

- 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)

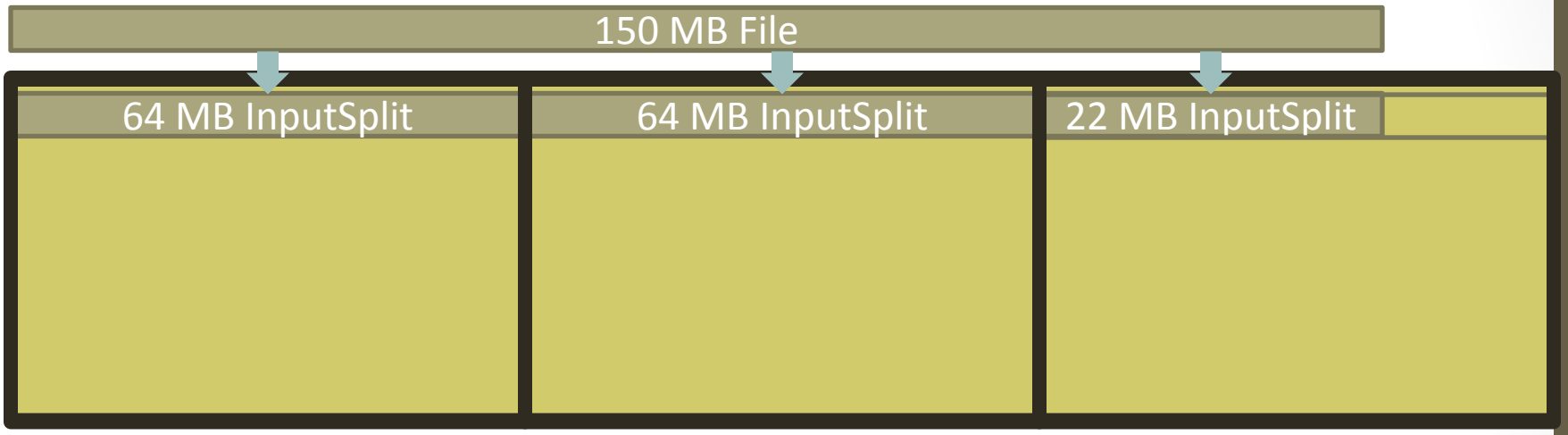
- Parallelization Rules in MapReduce
  - Map tasks occur in parallel independent of each other
  - Map tasks terminate in “Shuffle and Sort”
  - Some Reduce tasks can start before all Mappers are done.
  - Reduce tasks occur in parallel independent of each other
  - Reducers cannot complete before all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



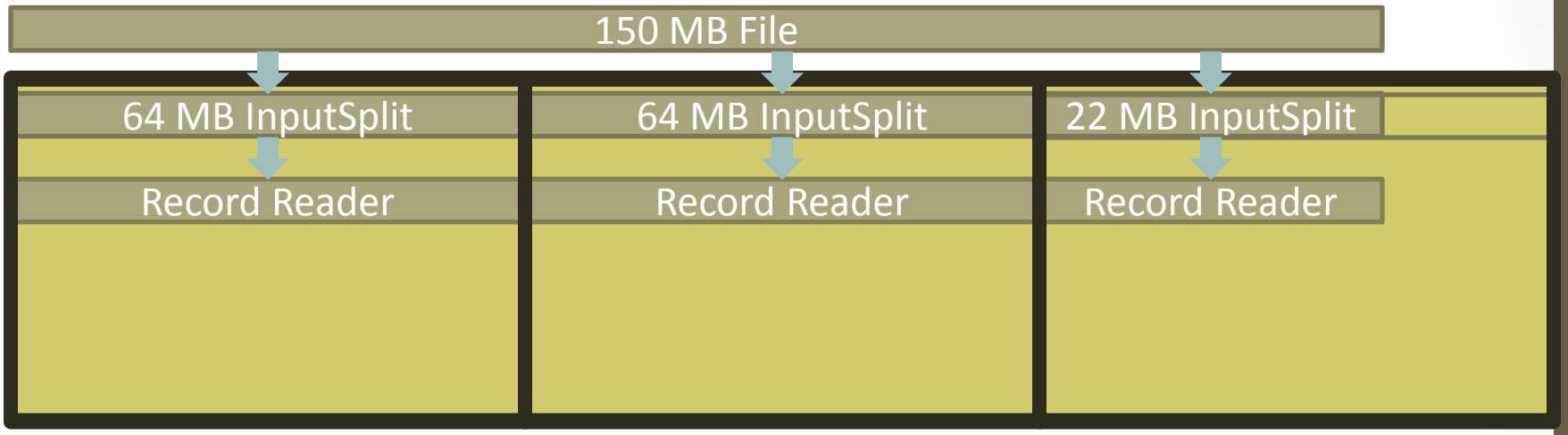
MapReduce (9)

MapReduce (10)

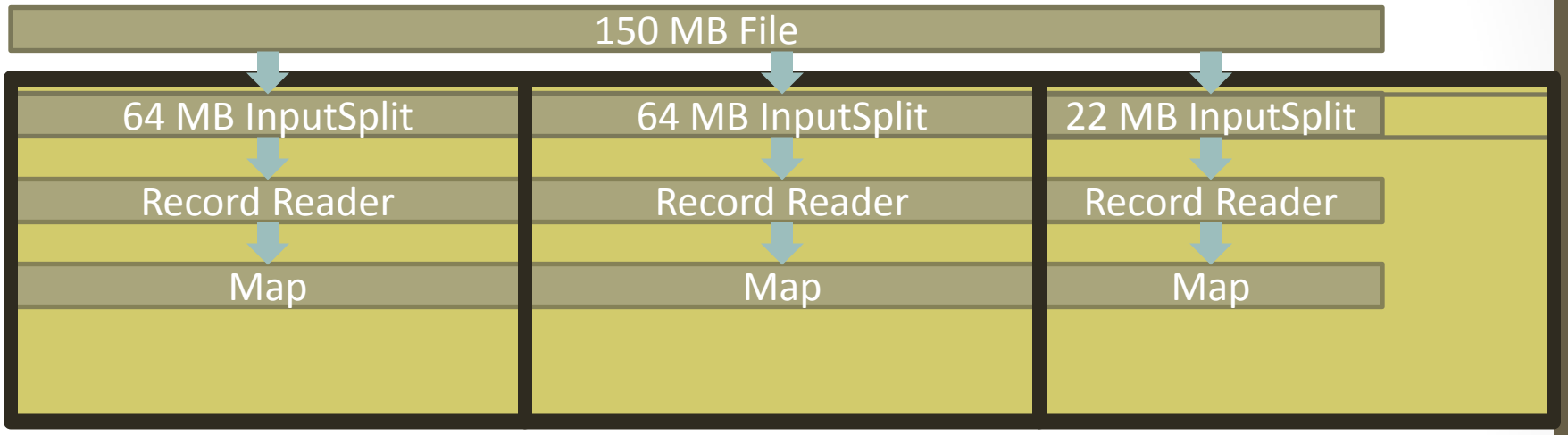
MapReduce (11)



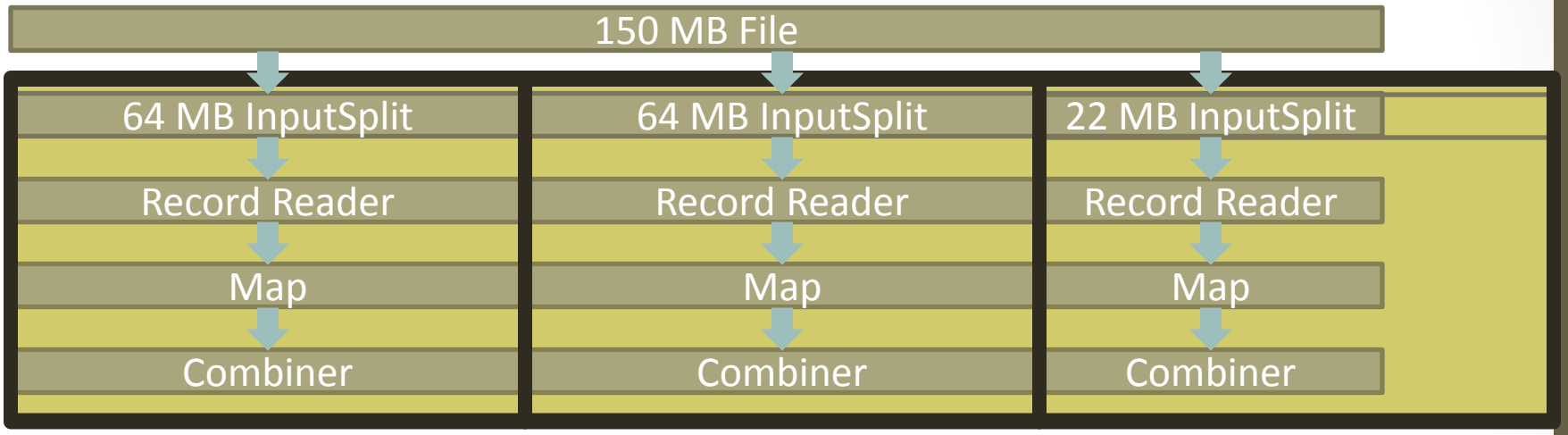
MapReduce (12)



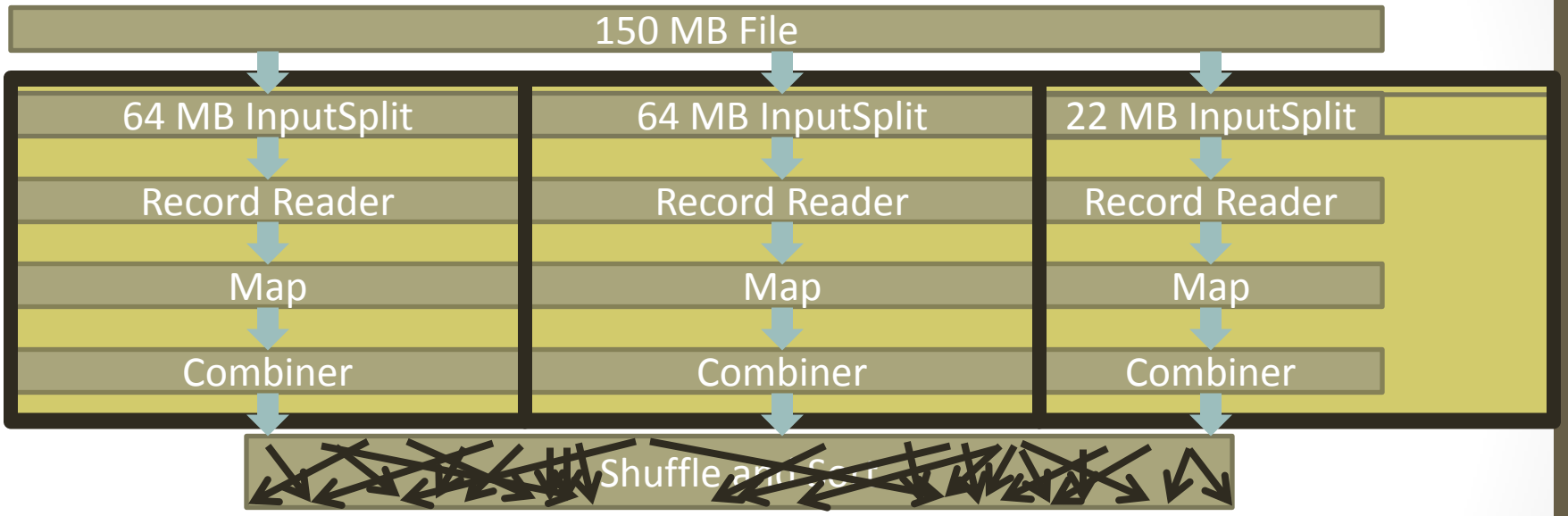
MapReduce (13)



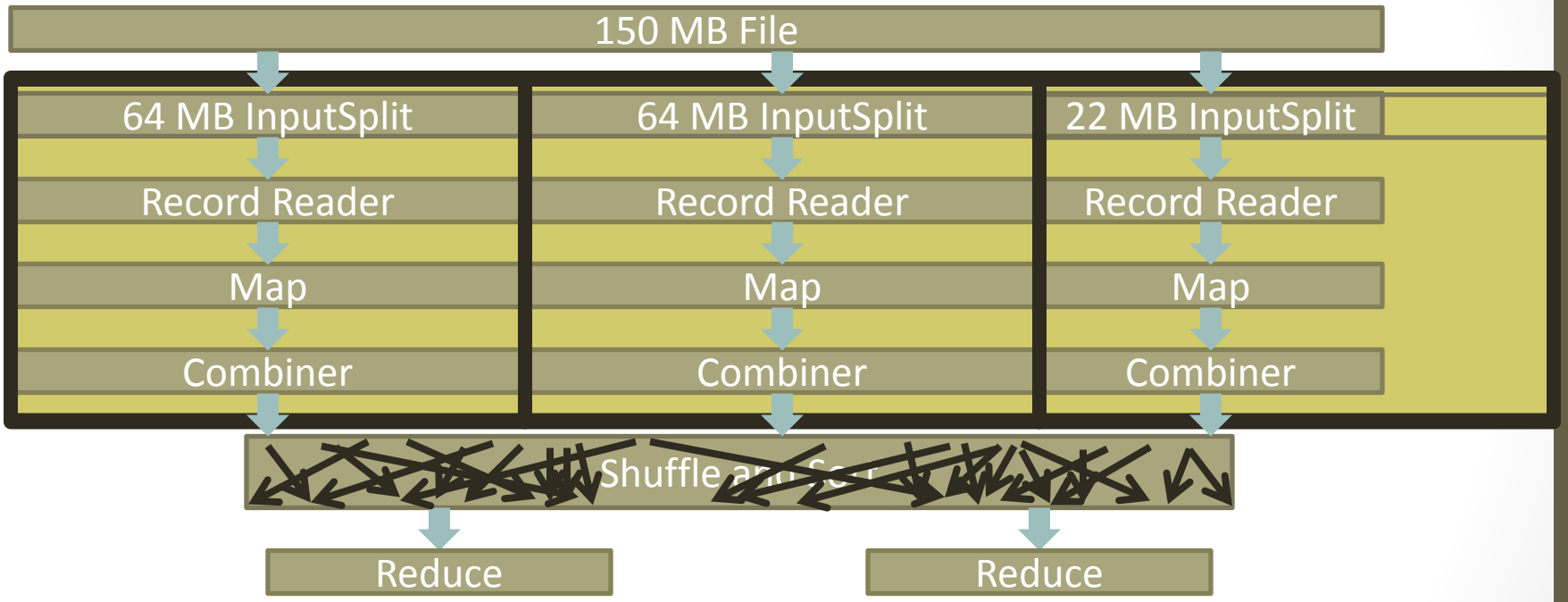
MapReduce (14)



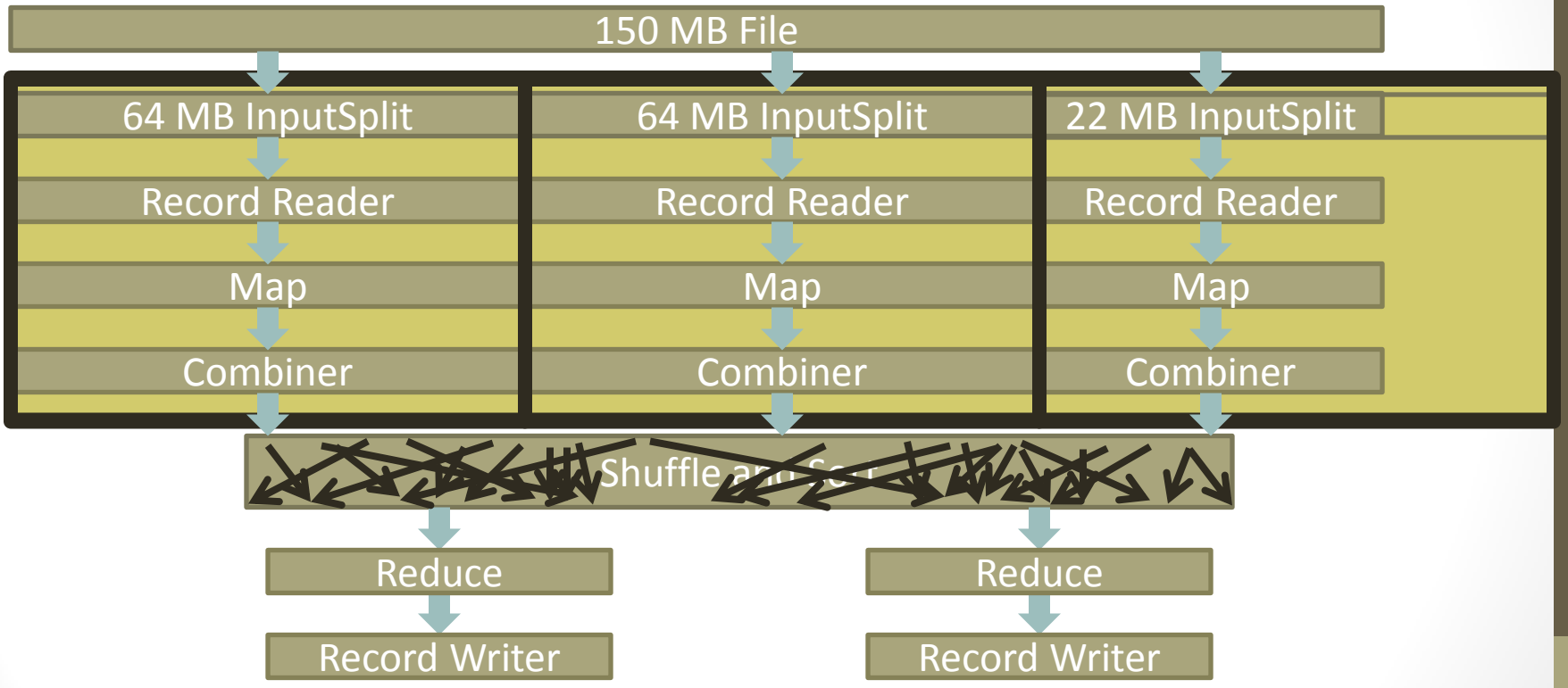
MapReduce (15)



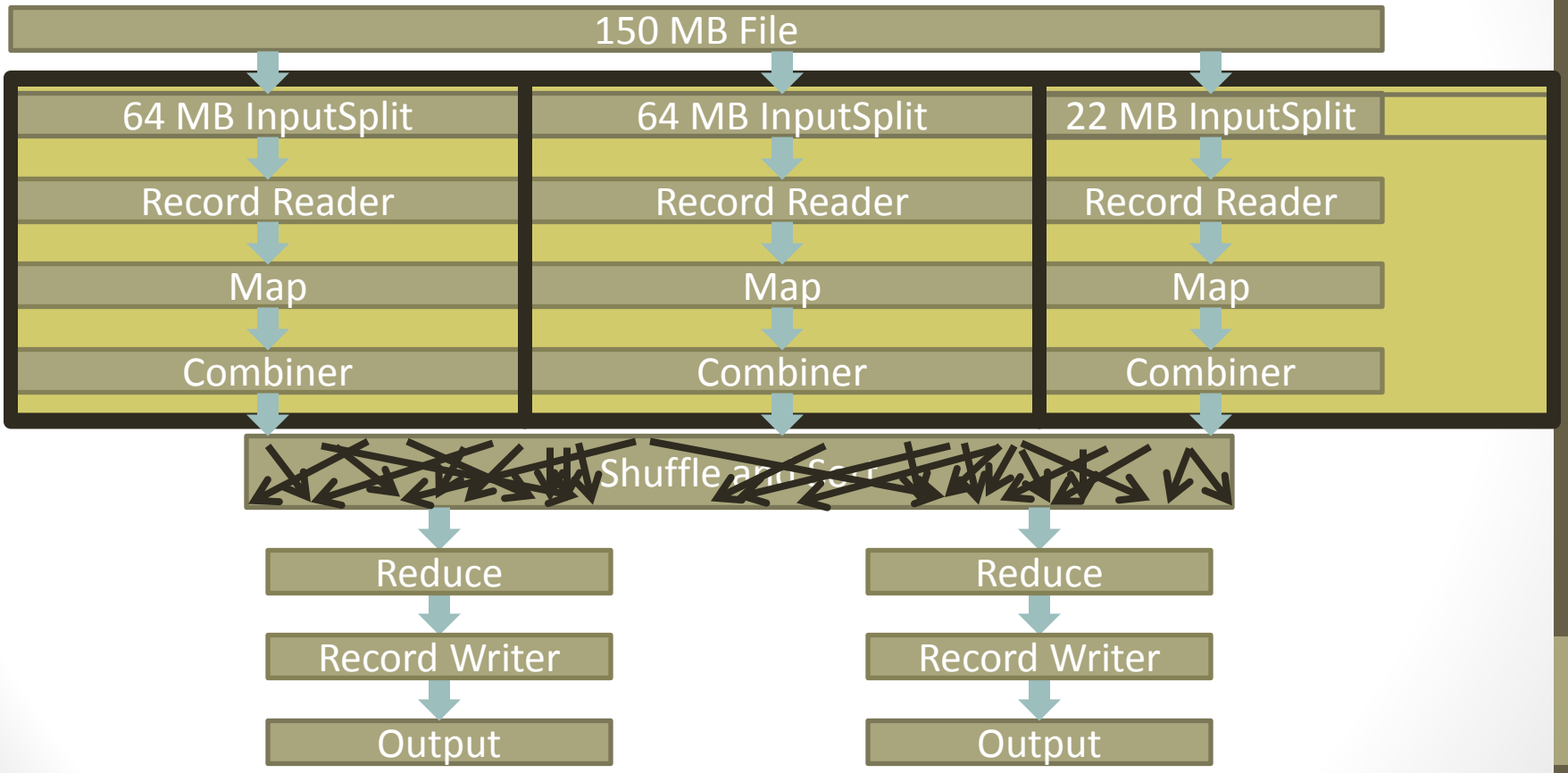
MapReduce (16)



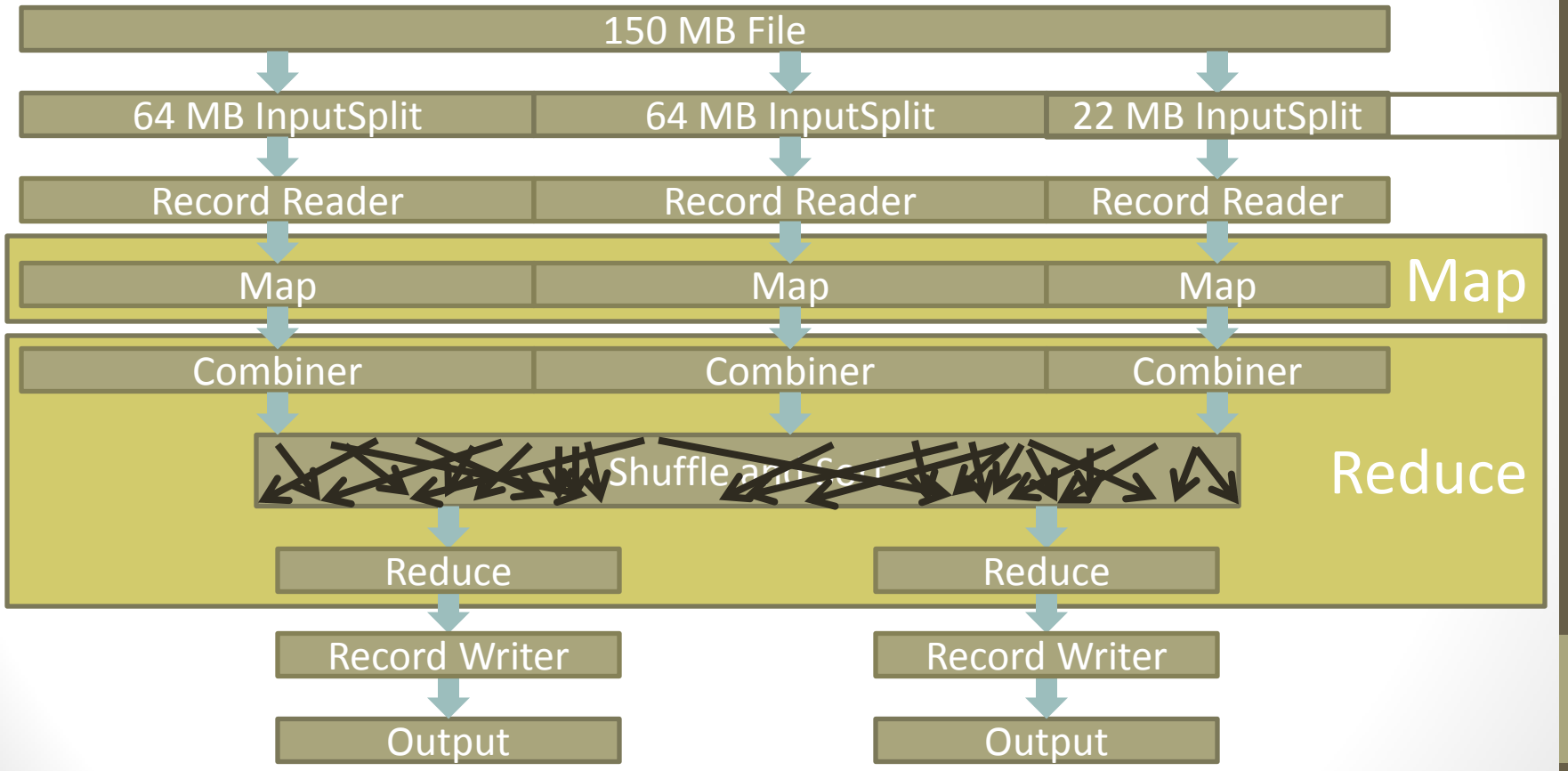
MapReduce (17)



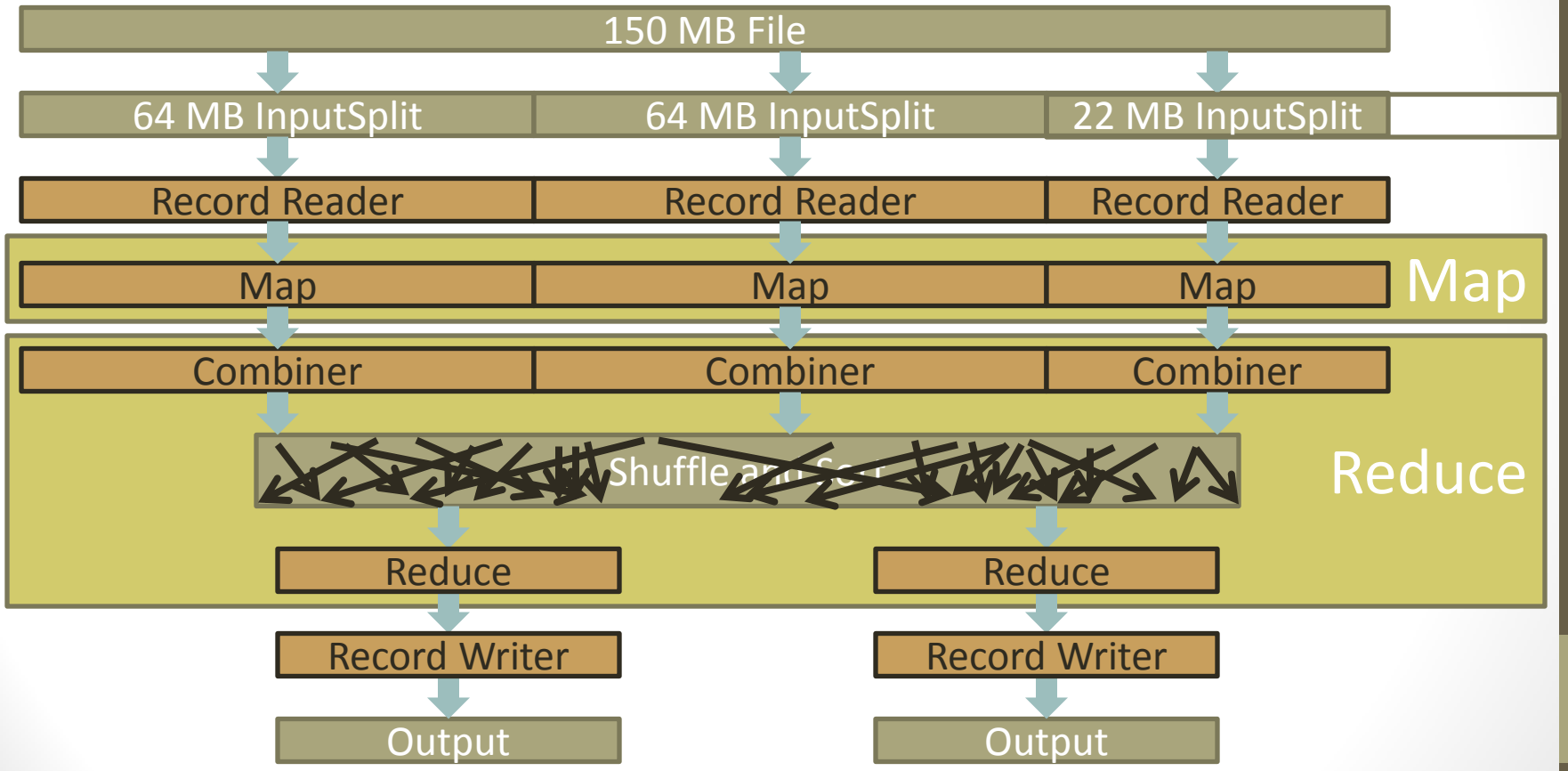
MapReduce (18)



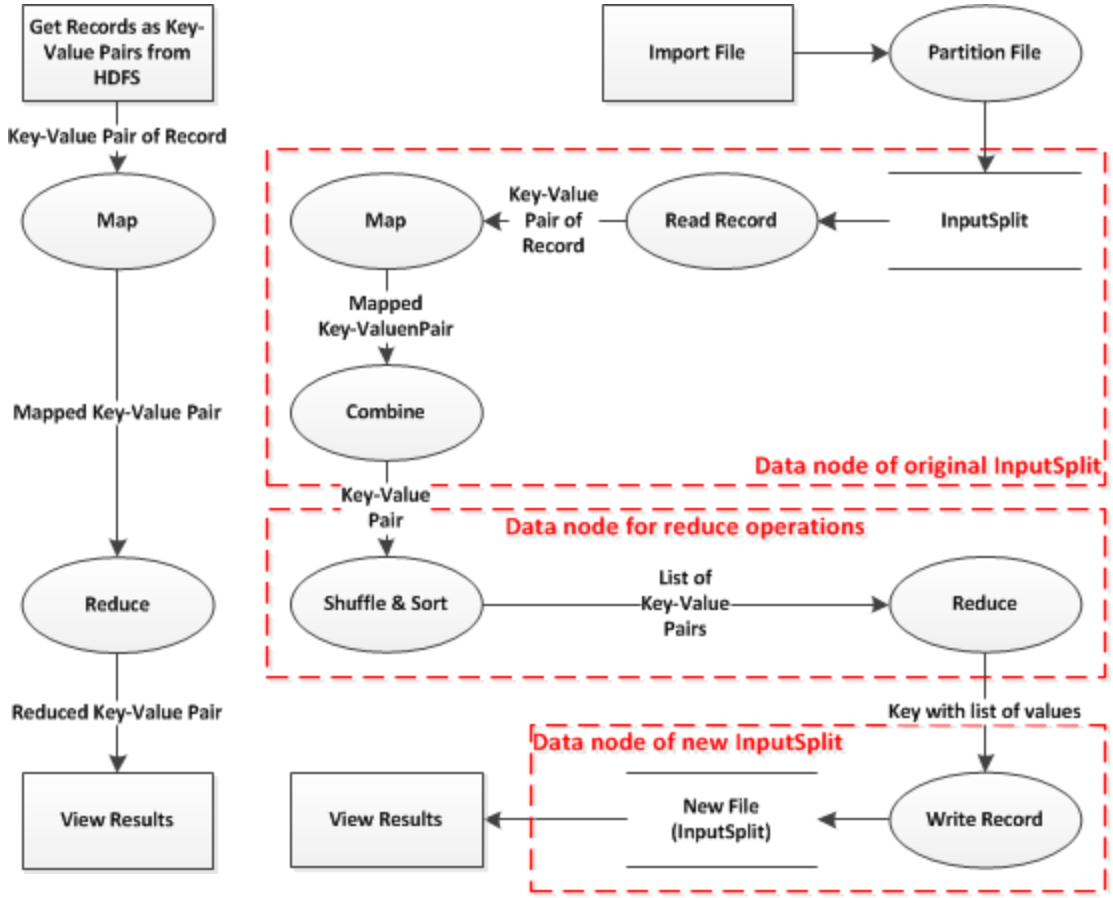
MapReduce(19)



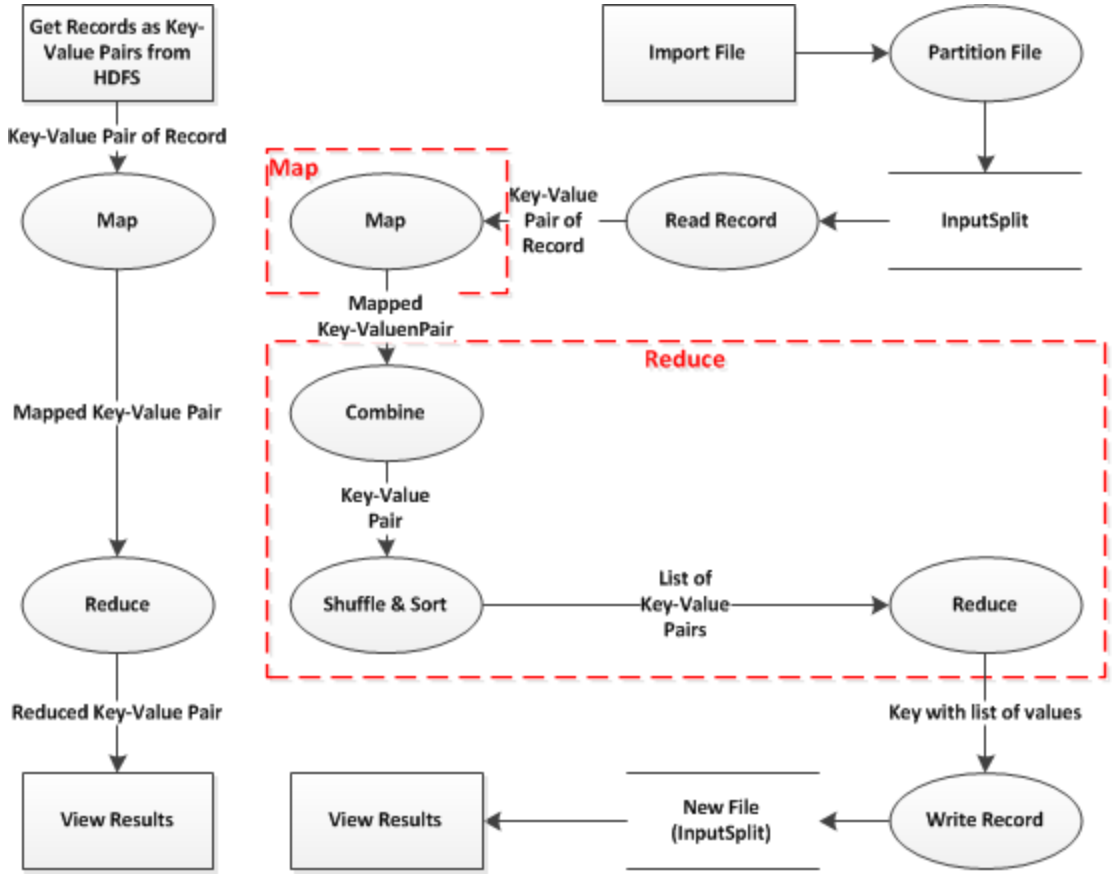
MapReduce (20)



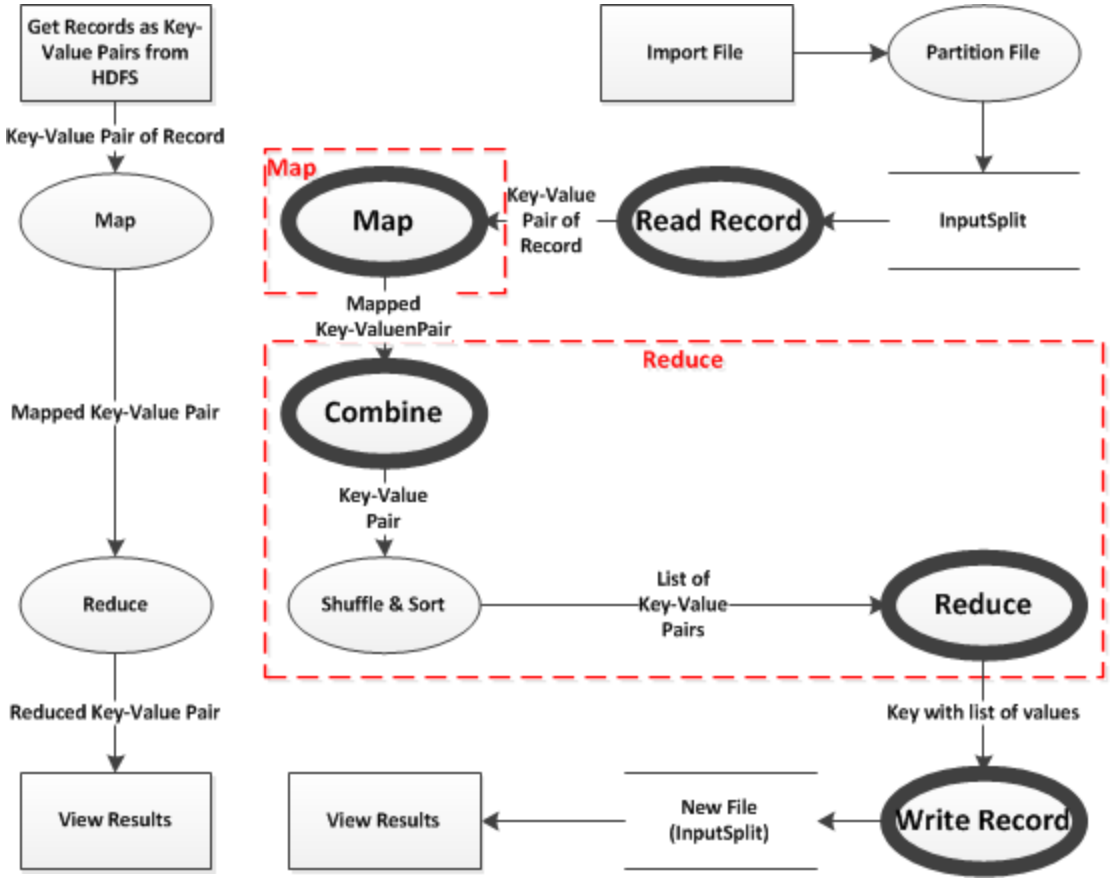
MapReduce (21)



MapReduce (22)



MapReduce (23)

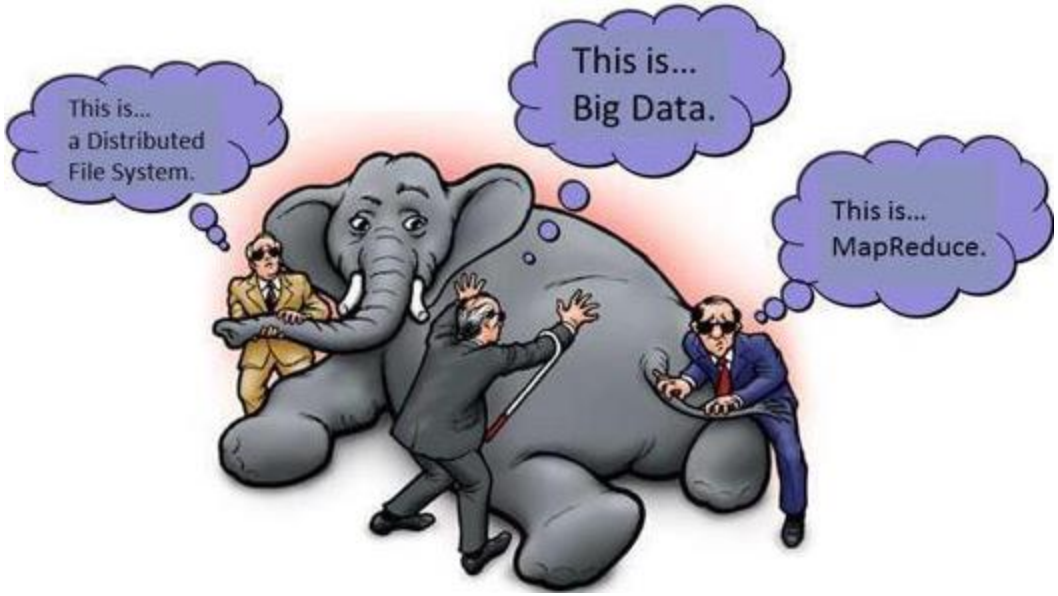


MapReduce (24)

- Record Read
  - Record → (K1,V1)
  - to be or not to be → (0,“to be or not to be”)
- Map
  - (K1,V1) → list(K2,V2)
  - (0,“to be or not to be”) → [(to,1),(be,1),(or,1),(not,1),(to,1),(be,1)]
- Shuffle and Sort
  - list(K2,V2) → (K2,list(V2))
  - [(to,1),(be,1),(or,1),(not,1),(to,1),(be,1)] → (to,[1,1]),(be,[1,1]),(or,[1]),(not,[1])
- Reduce
  - (K2,list(V2)) → list(K3,V3)
  - (to,[1,1]),(be,[1,1]),(or,[1]),(not,[1]) → [(to,2),(be,2),(or,1),(not,1)]
- Record Write
  - (K3,V3) → Records
  - [(to,2),(be,2),(or,1),(not,1)] →

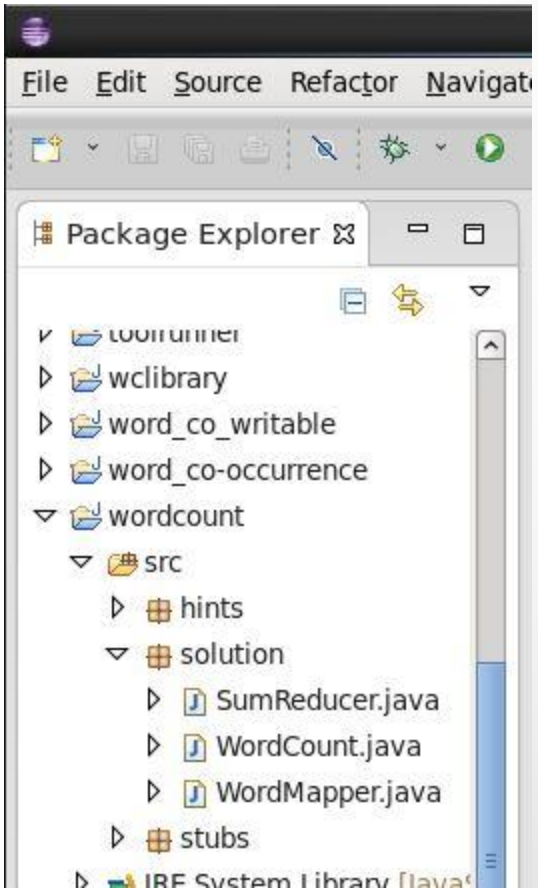
to,2
be,2
or,1
not,1

MapReduce Lab (0)



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)

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`

Note: the apostrophe in this statement is not the straight or slanted quote (' or ‘) but rather this character (`).

MapReduce Lab (3)

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`

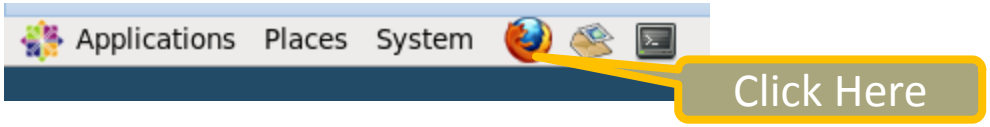
In Hadoop, run the wordcount program on Shakespeare's poems:

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

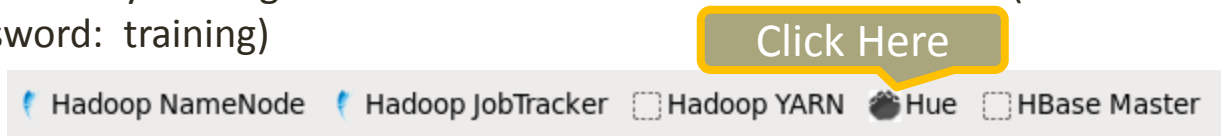
hadoop fs -rm -r wordcounts pwords

MapReduce Lab (4)

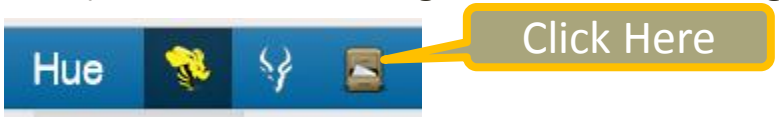
- Use Hue's HDFS file browser to browse and read the results
  1. Open Firefox



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



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



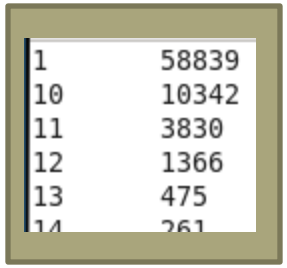
  4. Find wordcounts/part-r-00000

  5. View Contents of part-r-00000

Select Shakespeare > histories > Viewing Bytes 1163653 – 1168000; Read: **King Richard II**

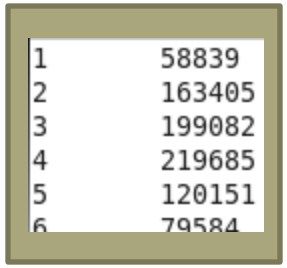
MapReduce Lab (5)

- This is part of today's homework assignment.
- Create a MapReduce program that lists word lengths and how often the occurrence of words of that length. 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 wordcounts`
  - View the results using the file browser
    - `hadoop fs -cat wordcounts/part-r-00000 | less`
  - Take a screen shot of the results. Similar to this:



MapReduce Lab (6)

- This is part of today's (optional) homework assignment.
- Create a MapReduce program that lists word lengths and how often the occurrence of words of that length. Modify package wordcount.solution:
  - Calculate the 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 wordcounts`
  - View the results using the file browser
    - `hadoop fs -cat wordcounts/part-r-00000 | less`
  - Take a screen shot of the results. Similar to this:




C'mon, Hadoop — it's bedtime!

Hadoop-2 Labs