CS 644:
Introduction to Big Data

Daqing Yun (daqing.yun@njit.edu)
New Jersey Institute of Technology

Outline

- Big Data Ecosystem
  - *Hadoop*
  - *Pig*
  - *Spark*
  - Hive
  - Tez
  - Storm
  - Oozie
  - HBase

Big Data (Hadoop) Ecosystem

Big Data Applications/Domains
(Healthcare, insurance, finance, social networks, transportation, sciences, etc.)

Big Data Analytics
(Methods: AI, machine learning, visualization, etc. Modules: Pig, Hive, Mahout, etc.)

Big Data Computing
(YARN, MapReduce, Spark, Storm, Oozie, etc.)

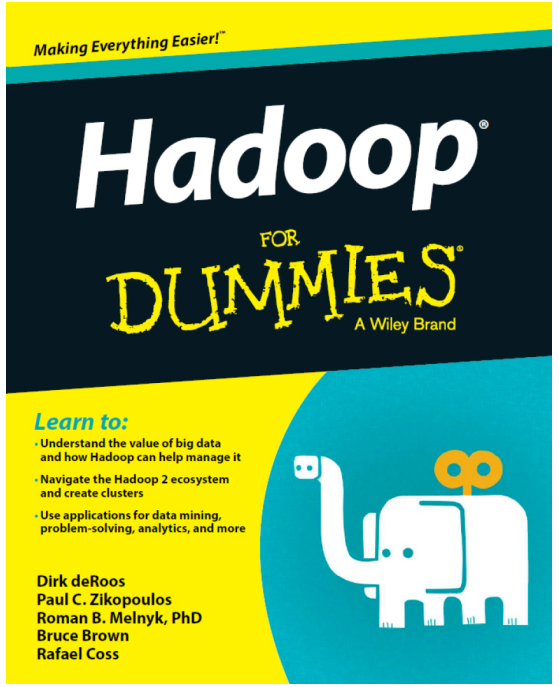
Big Data Management
(Structures: Key-Value, Document, Graph, etc. Systems: MongoDB, Hbase, Cassandra, etc.)

Big Data Storage
(HDFS)

Big Data Networking
(HPN, SDN, etc.)

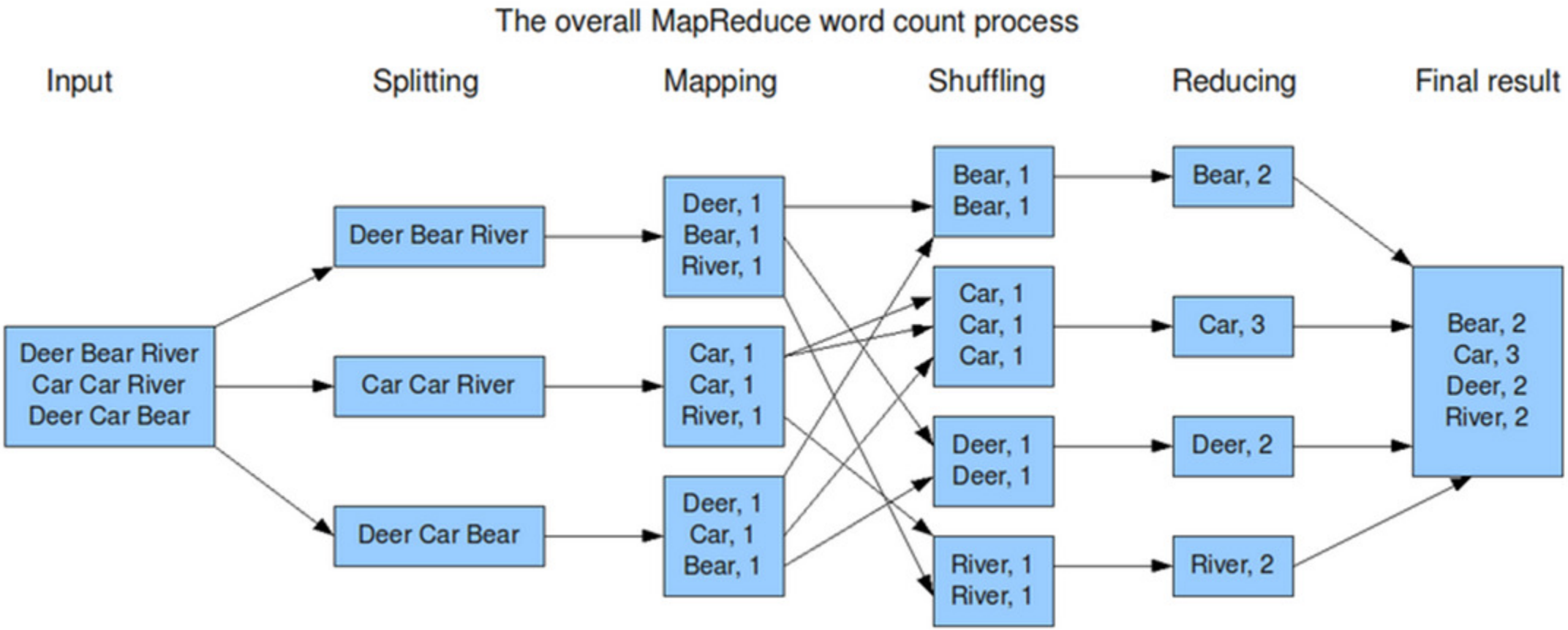
Use Cases for Big Data in Hadoop

- Log Data Analysis
  - Most common, fits perfectly for HDFS scenario: Write once & Read often.
- Data Warehouse Modernization
- Fraud Detection
- Risk Modeling
- Social Sentiment Analysis
- Image Classification
- Graph Analysis
- Beyond

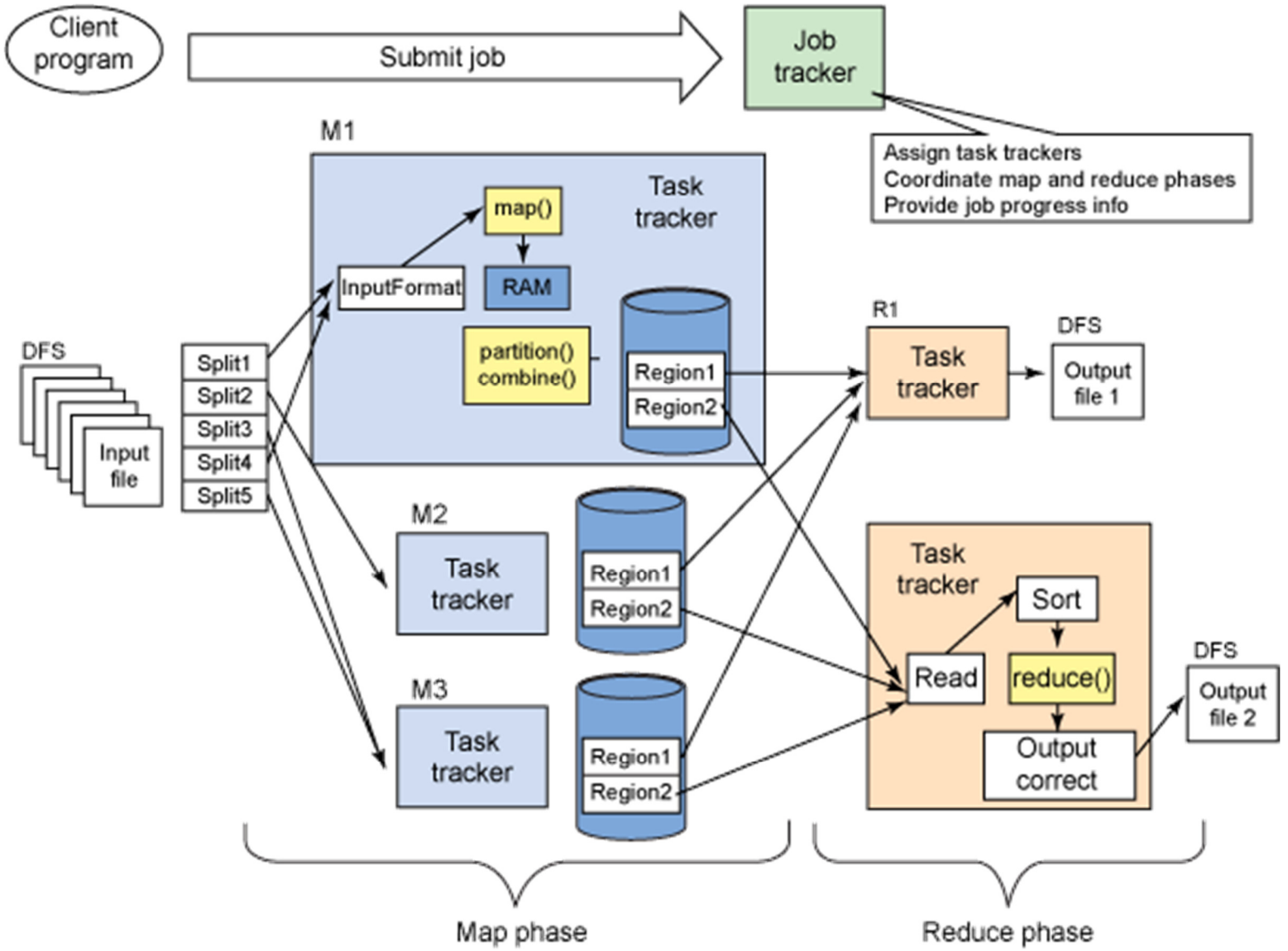


D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

MapReduce Example



MapReduce Data Flow



http://www.ibm.com/developerworks/cloud/library/cl-openstack-deployhadoop/

Apache Hadoop

- The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.
- The Apache Hadoop software library is **a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models**. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.
- This project includes these modules:
  - **Hadoop Common**: The common utilities that support the other Hadoop modules.
  - **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
  - **Hadoop YARN**: A framework for job scheduling and cluster resource management.
  - **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

http://hadoop.apache.org

Hadoop-related Apache Projects

- **Ambari™**: A web based tool for provisioning, managing, and monitoring Hadoop clusters. It also provides a dashboard for viewing cluster health and ability to view MapReduce, Pig, and Hive applications visually.
- **Avro™**: A data serialization system.
- **Cassandra™**: A scalable multi-master database with no single points of failure.
- **Chukwa™**: A data collection system for managing large distributed systems.
- **HBase™**: A scalable, distributed database that supports structured data storage for large tables.
- **Hive™**: A data warehouse infrastructure that provides data summarization and ad hoc querying
- **Mahout™**: A scalable machine learning and data mining library.
- **Pig™**: A high-level data-flow language and execution framework for parallel computation
- **Spark™**: A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
- **Tez™**: A generalized data-flow programming framework, built on Hadoop YARN, which provides a powerful and flexible engine to execute an arbitrary DAG of tasks to process data for both batch and interactive use-cases.
- **ZooKeeper™**: A high-performance coordination service for distributed applications.

Hadoop Distributed File System (HDFS)

- HDFS is a java-based file system that provides the scalable, fault-tolerant, cost-efficient storage for big data
  - The file content is split into large blocks (typically 128 megabytes), each of which is independently replicated at multiple DataNodes
  - The NameNode maintains the namespace tree (in RAM) and the mapping of blocks to DataNodes

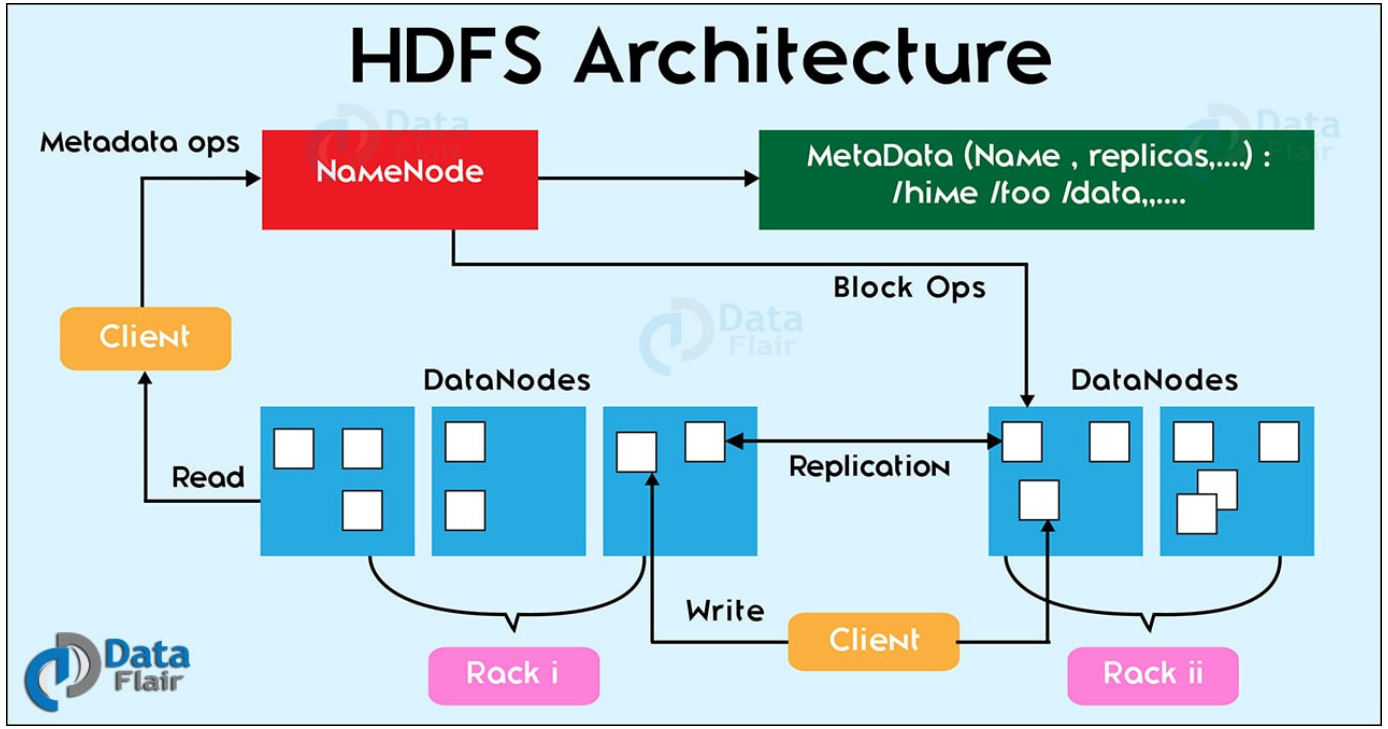
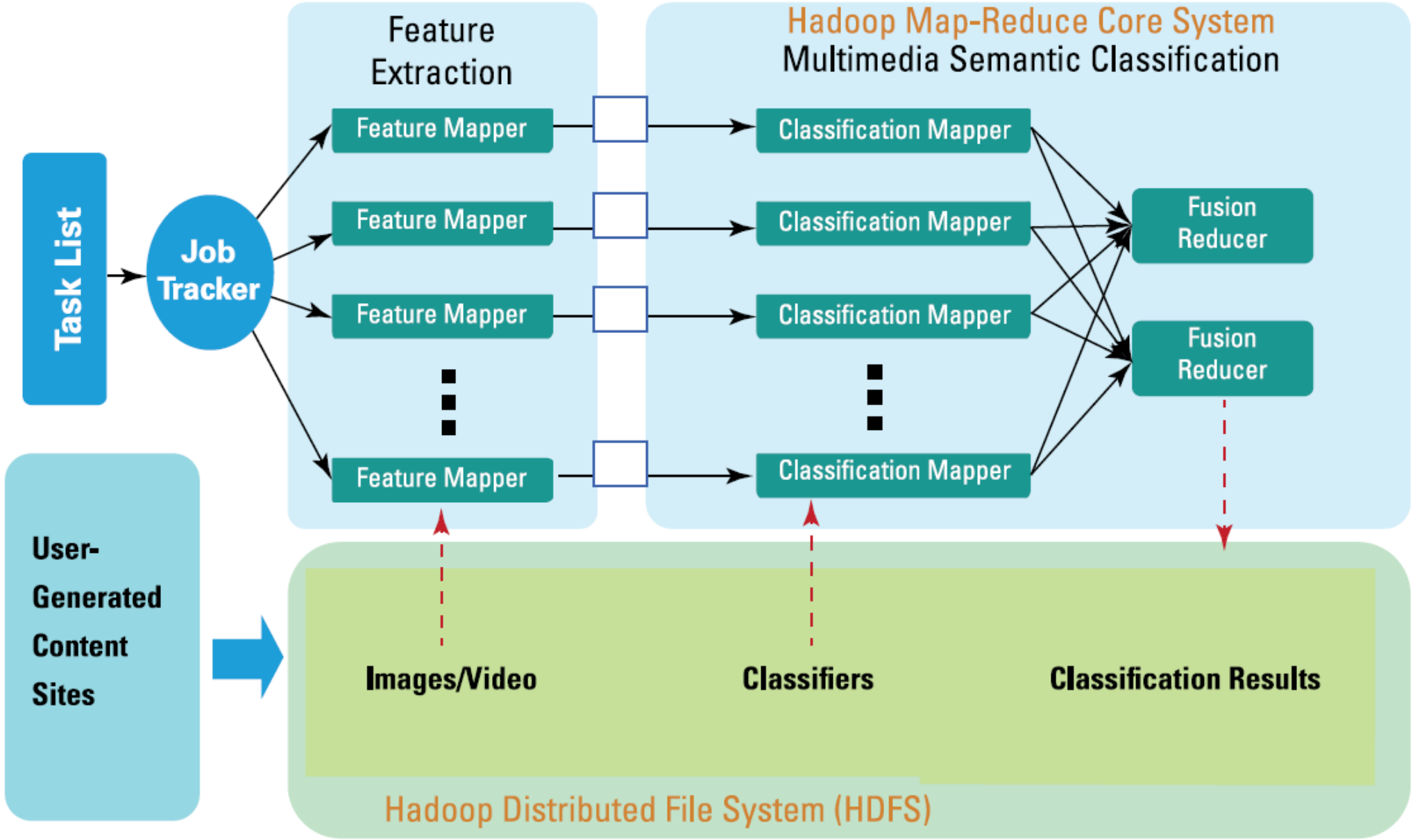


Image source: https://data-flair.training/blogs/hadoop-hdfs-architecture/

How Hadoop Works



Data Storage Operations on HDFS

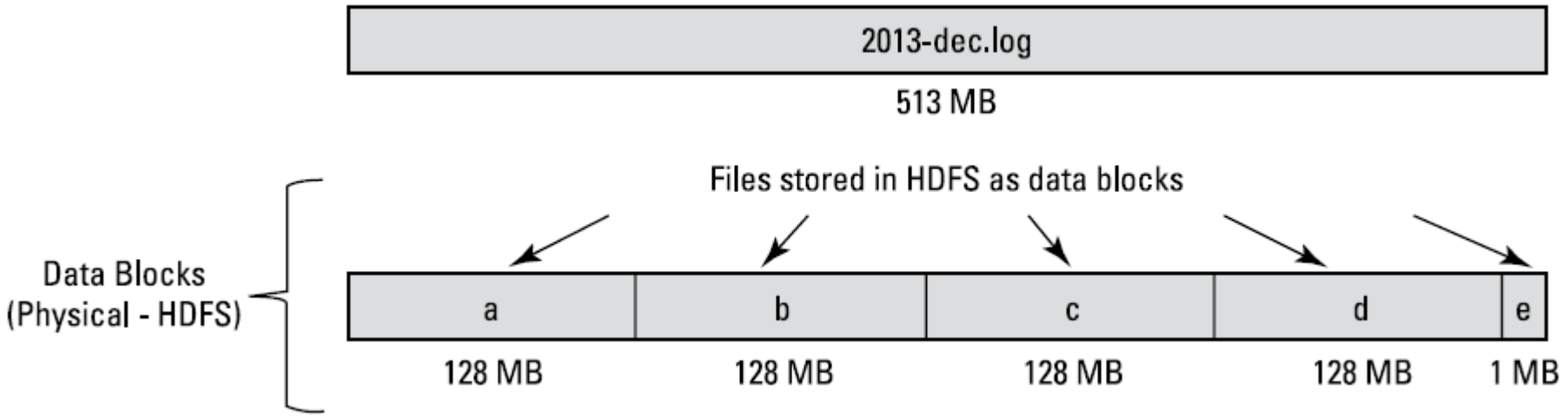
- Hadoop is designed to work best with a modest number of extremely large files.
- Average file sizes ➔ larger than 500MB.

- Write Once, Read Often model.
- Content of individual files cannot be modified, other than appending new data at the end of the file.

- What we can do:
  - Create a new file
  - Append content to the end of a file
  - Delete a file
  - Rename a file
  - Modify file attributes like owner

HDFS Blocks

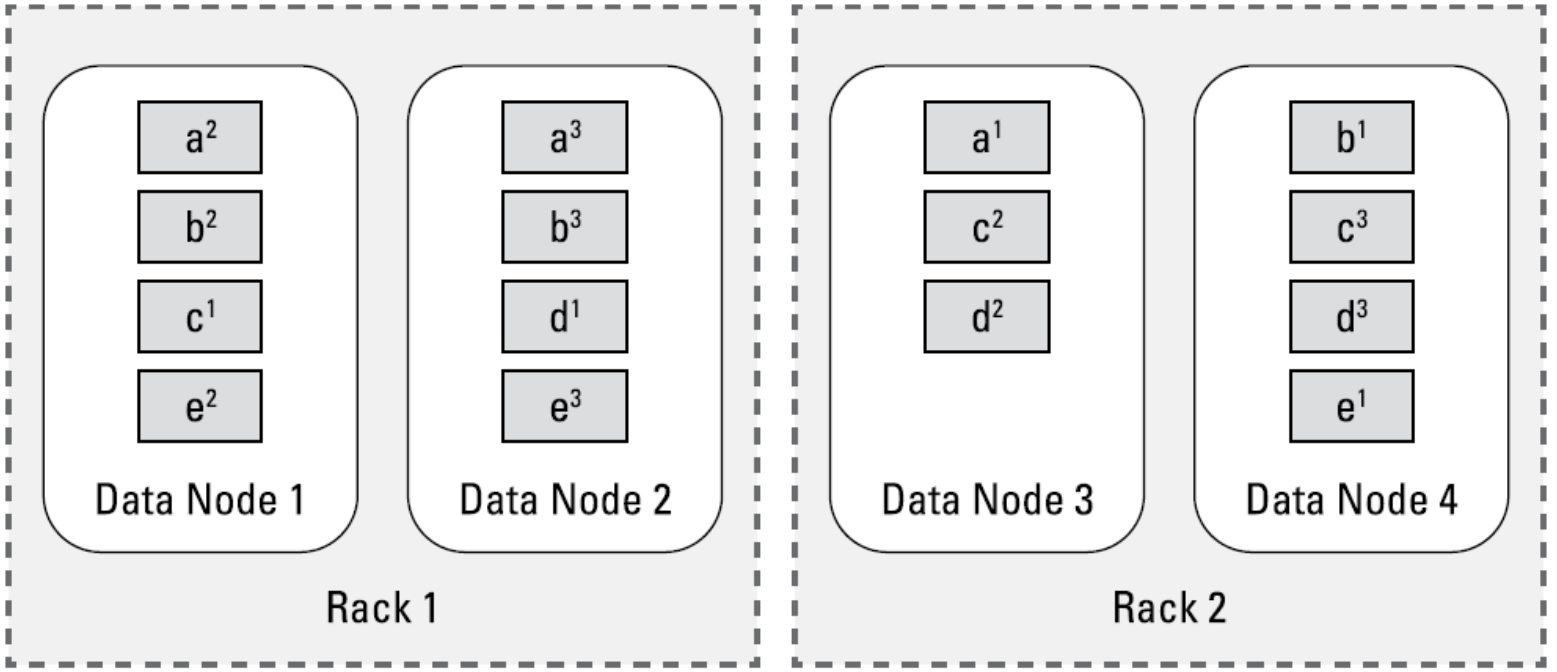
- File is divided into blocks (default: 64MB) and duplicated in multiple places (default: 3)



- Dividing into blocks is normal for a file system, e.g., the default block size in Linux is 4KB. The difference of HDFS is the scale.
- Hadoop was designed to operate at the petabyte scale.
- Every data block stored in HDFS has its own metadata and needs to be tracked by a central server.

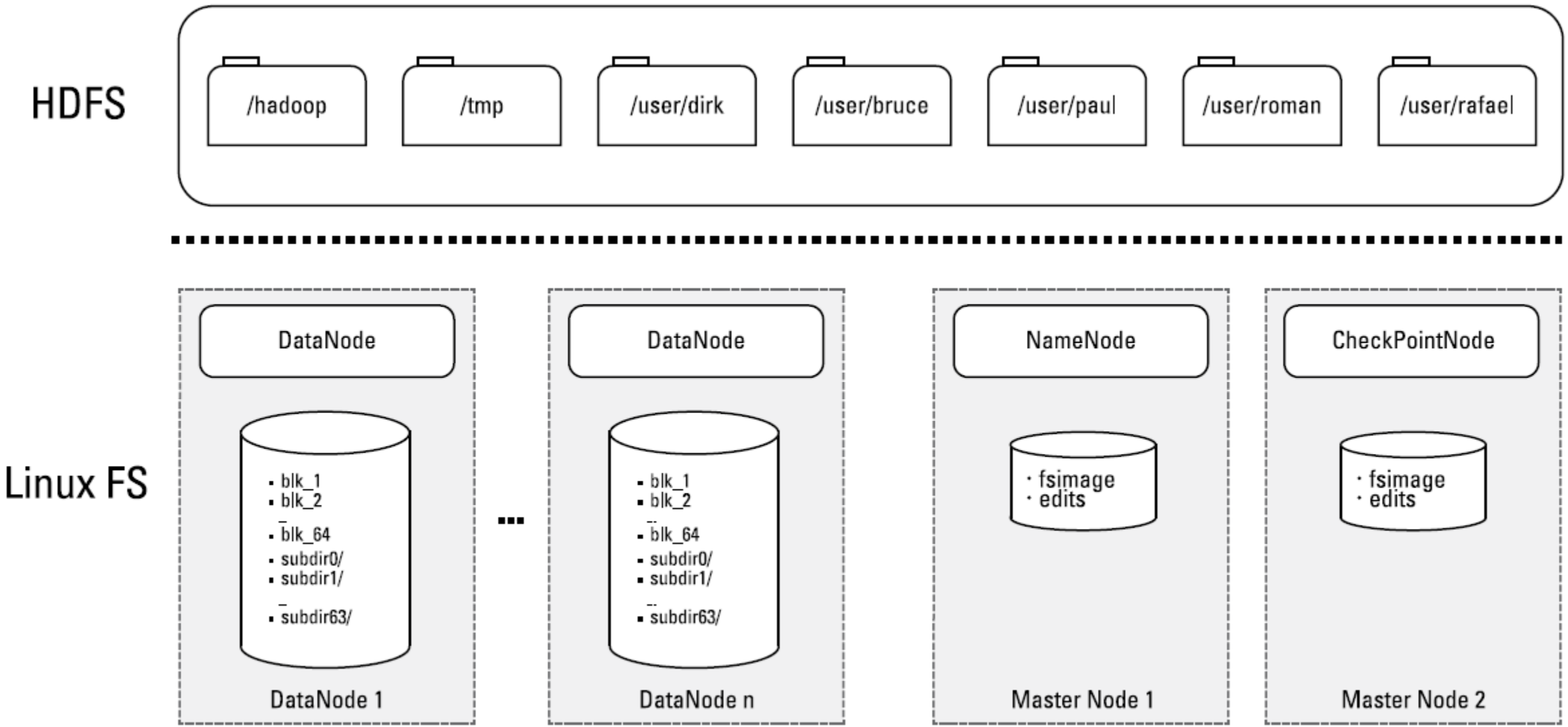
HDFS Blocks

- Replication patterns of data blocks in HDFS.



- When HDFS stores the replicas of the original blocks across the Hadoop cluster, it tries to ensure that the block replicas are stored in different failure points.

HDFS is User-Space-Level File System



Interaction between HDFS components

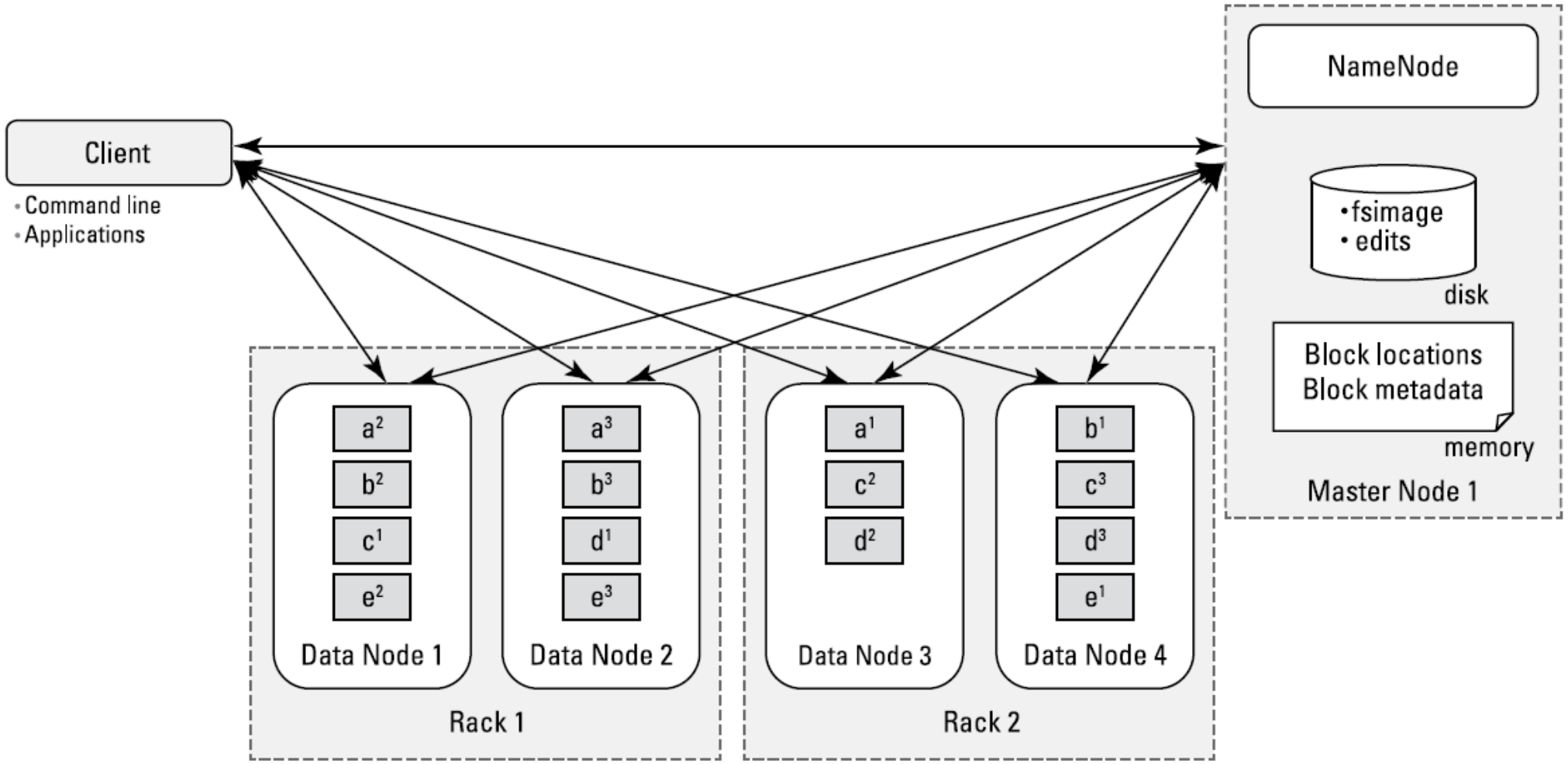
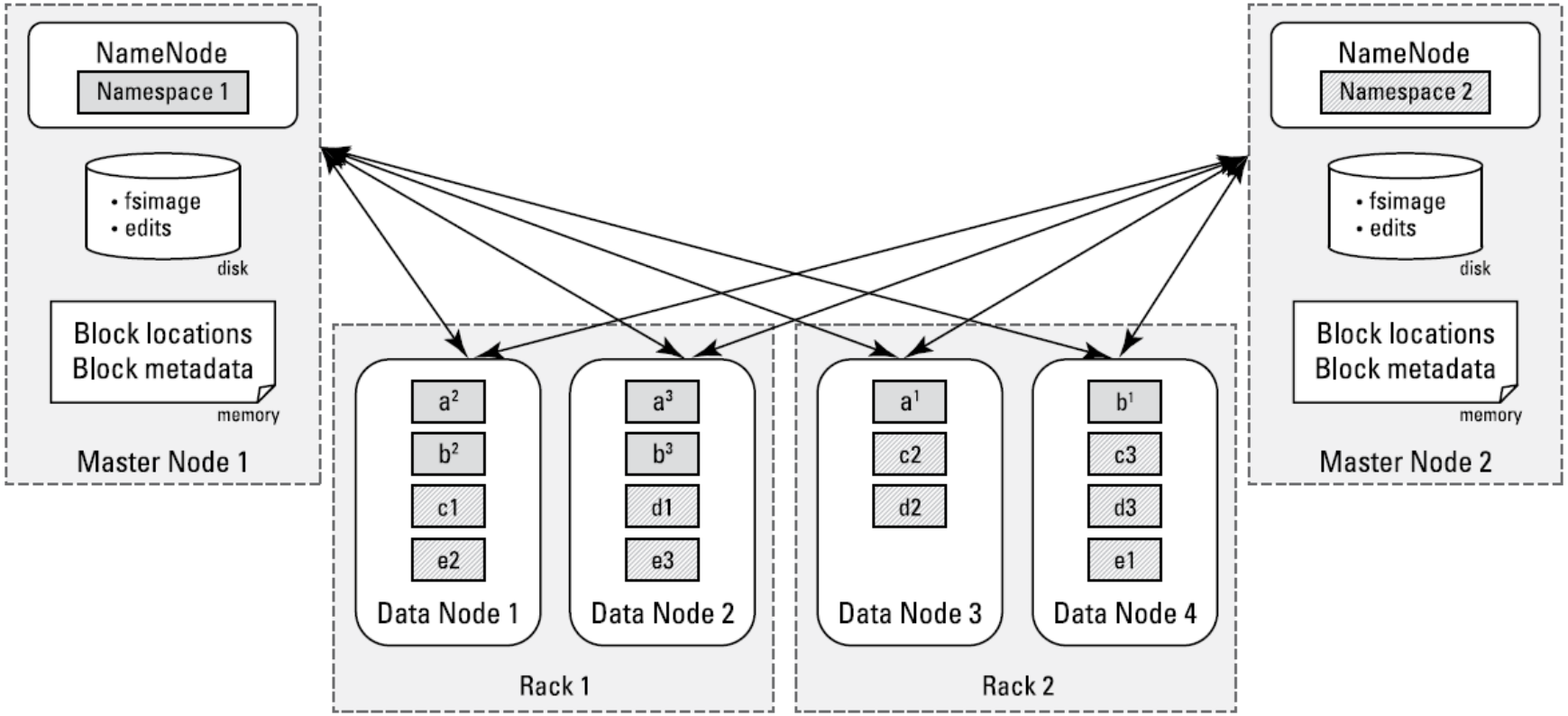


Image source: D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

HDFS Federation

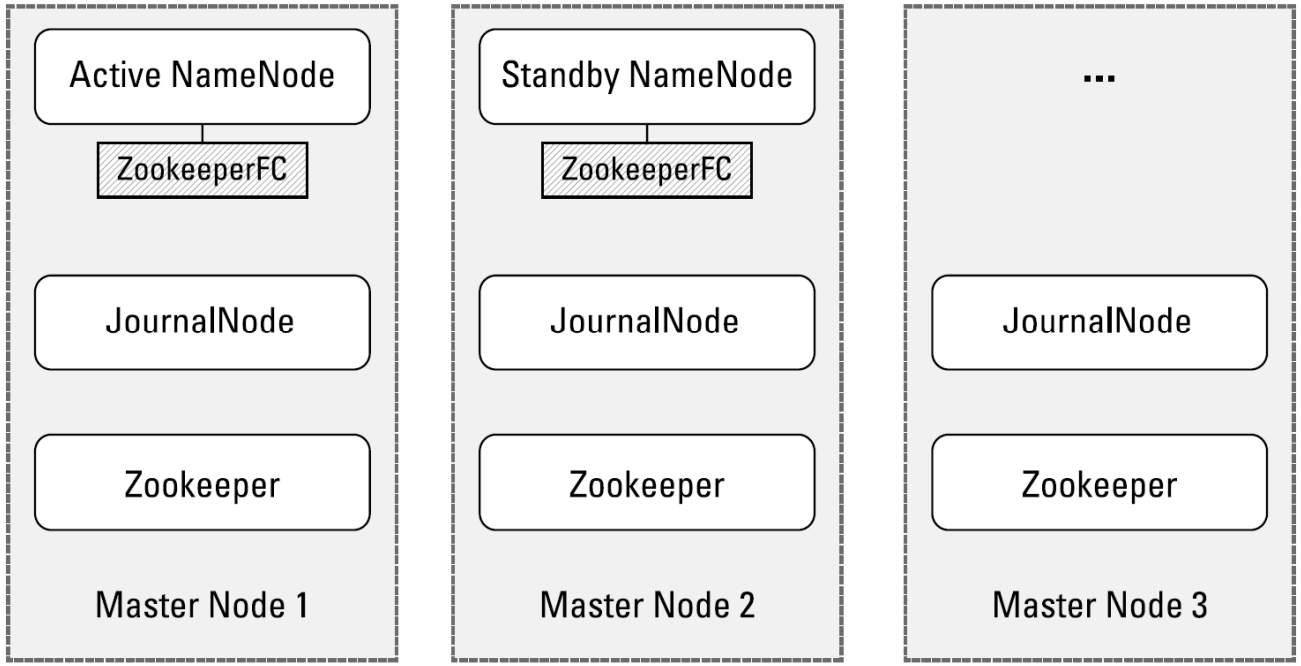
- Before Hadoop 2.0
  - NameNode was a single point of failure and operation limitation.
  - Hadoop clusters usually have fewer clusters that were able to scale beyond 3,000 or 4,000 nodes.
- In Hadoop 2.x
  - Multiple NameNodes can be used (High Availability feature – one is in an Active state, the other one is in a Standby state).



https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithNFS.html

High Availability of the NameNodes

- Active NameNode
- Standby NameNode – keeping the state of the block locations and block metadata in memory → HDFS checkpointing responsibilities



- JournalNode – if a failure occurs, the Standby Node reads all completed journal entries to ensure the new Active NameNode is fully consistent with the state of cluster
- Zookeeper – provides coordination and configuration services for distributed systems

https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithNFS.html

Data Compression in HDFS

Codec	*File Extension*	*Splittable?*	*Degree of Compression*	*Compression Speed*
Gzip	`.gz`	No	Medium	Medium
Bzip2	`.bz2`	Yes	High	Slow
Snappy	`.snappy`	No	Medium	Fast
LZO	`.lzo`	No, unless indexed	Medium	Fast

Several Useful Commands for HDFS

- All hadoop commands are invoked by the bin/hadoop script in format:

  `hadoop [--config confdir] [COMMAND][GENERIC_OPTIONS][COMMAND_OPTIONS]`

- For example,

  `$hadoop fsck / -files -blocks`

  → List the blocks that make up each file in HDFS

- For HDFS, the schema name is `hdfs`, and for the local file system, the schema name is `file`

- A file or directory in HDFS can be specified in a fully qualified way, such as

  `$hdfs://namenodehost/parent/child or $hdfs://namenodehost`

Several Useful Commands for HDFS

- The HDFS file system shell command is similar to Linux file commands, with the following general syntax:
  `$hadoop hdfs -file_cmd`
- For instance mkdir runs as:

  ```
  $hadoop hdfs dfs -mkdir /user/directory_name
  ```

- To create a folder named “joanna”, run

  ```
  $hadoop hdfs dfs -mkdir /user/joanna
  ```

- Copy a file from your local file system to HDFS, use put, run

  ```
  $hadoop hdfs dfs -put file_name /user/login_user_name
  ```

- Copy the file “data.txt” to the newly created directory, run

  ```
  $hadoop hdfs dfs -put data.txt /user/joanna
  ```

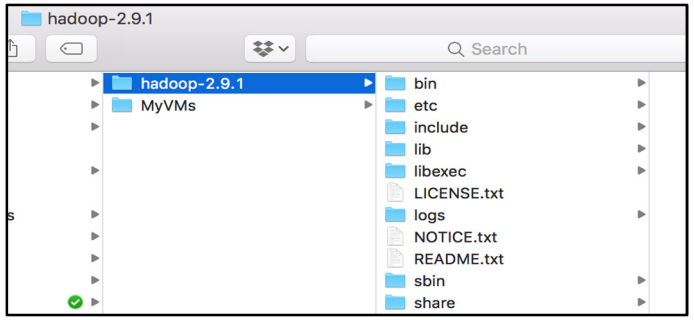
- To show a list of HDFS file, run

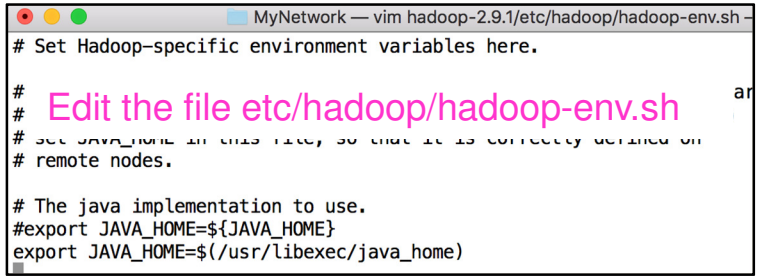
  ```
  $hadoop hdfs dfs -ls
  ```

Setting up the Hadoop environment

- Local (standalone) mode
- Pseudo-distributed mode
- Fully-distributed mode

Download binary package & extract





Setup passphraseless ssh

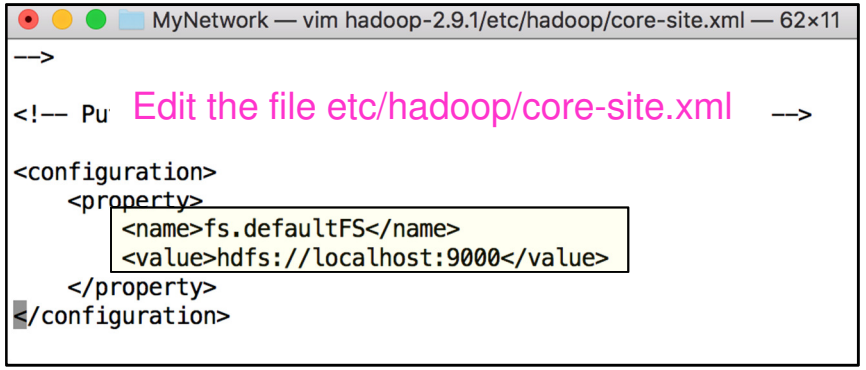
check that you can ssh to the localhost without a passphrase:

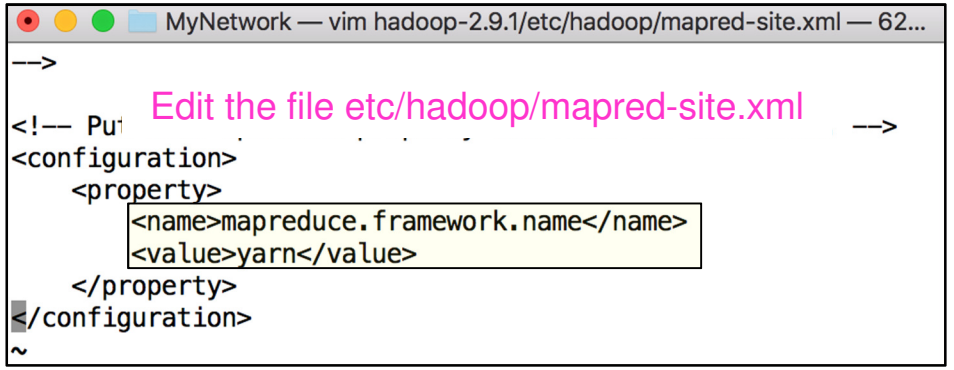
```
$ ssh localhost
```

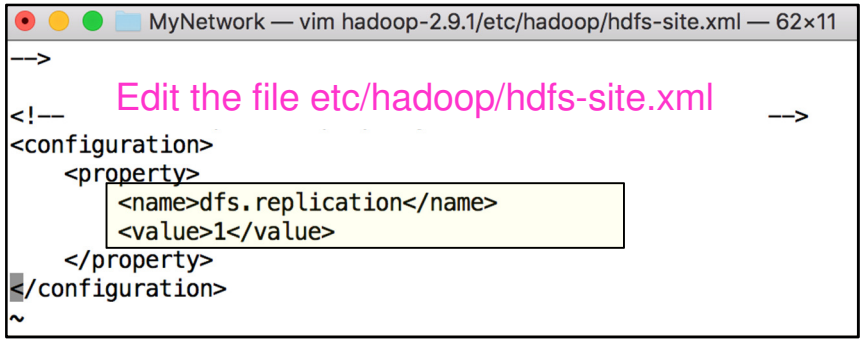
If you cannot ssh to localhost without a passphrase, execute the following commands:

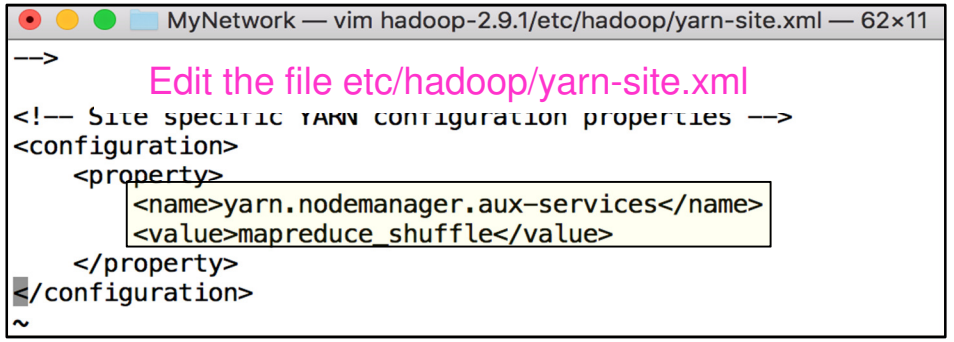
```
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
```

authorized_keys:
used by the SSH server to storethe public keys of clients for client authentication
known_hosts:
used by the SSH client to storethe public keys of servers for server authentication









Format and start HDFS and YARN

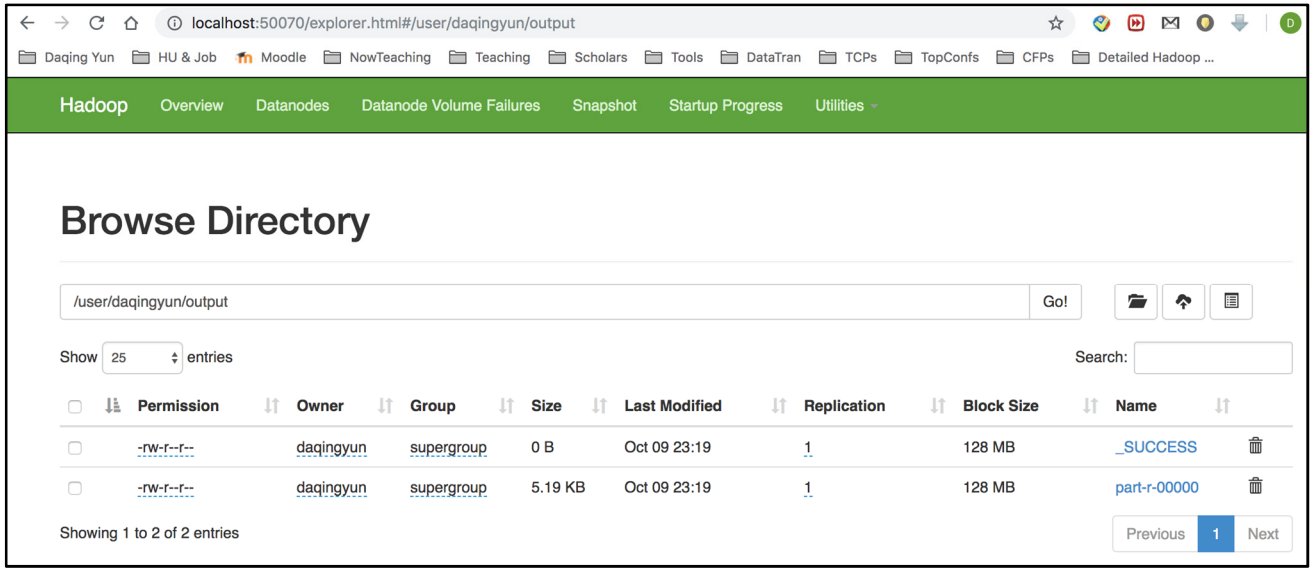
```
$ cd {your hadoop distribution directory}
```

Format the filesystem:

```
$ bin/hdfs namenode -format
```

Start NameNode daemon and DataNode daemon:

```
$ sbin/start-dfs.sh
```



Now you can browse the web interface for the NameNode at - http://localhost:50070/

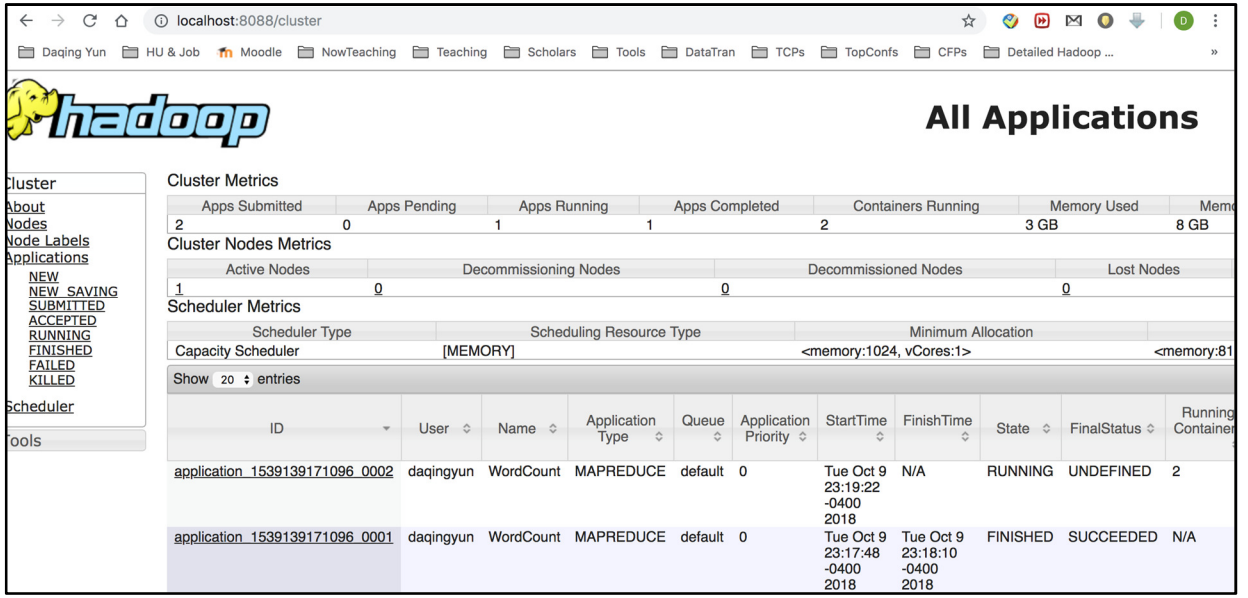
Make the HDFS directories required to execute MapReduce jobs:

```
$ bin/hdfs dfs -mkdir /user
$ bin/hdfs dfs -mkdir /user/{username}
```

Start ResourceManager daemon and NodeManager daemon:

```
$ sbin/start-yarn.sh
```

Browse the web interface for the ResourceManager at http://localhost:8088/



Test examples code that came with the hadoop version
Copy the input files into the distributed filesystem:

```
$ bin/hdfs dfs -put etc/hadoop input
```

Run some of the examples provided:

```
$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.0.jar grep input output 'dfs[a-z.]+'
```

This example counts the words starting with “dfs” in the input.

Examine the output files:
Copy the output files from the distributed filesystem to the local filesystem and examine them:

```
$ bin/hdfs dfs -get output output
$ cat output/*
```

or View the output files on the distributed filesystem:

```
$ bin/hdfs dfs -cat output/*
```

Stop YARN and HDFS
When you're done, stop the daemons with:

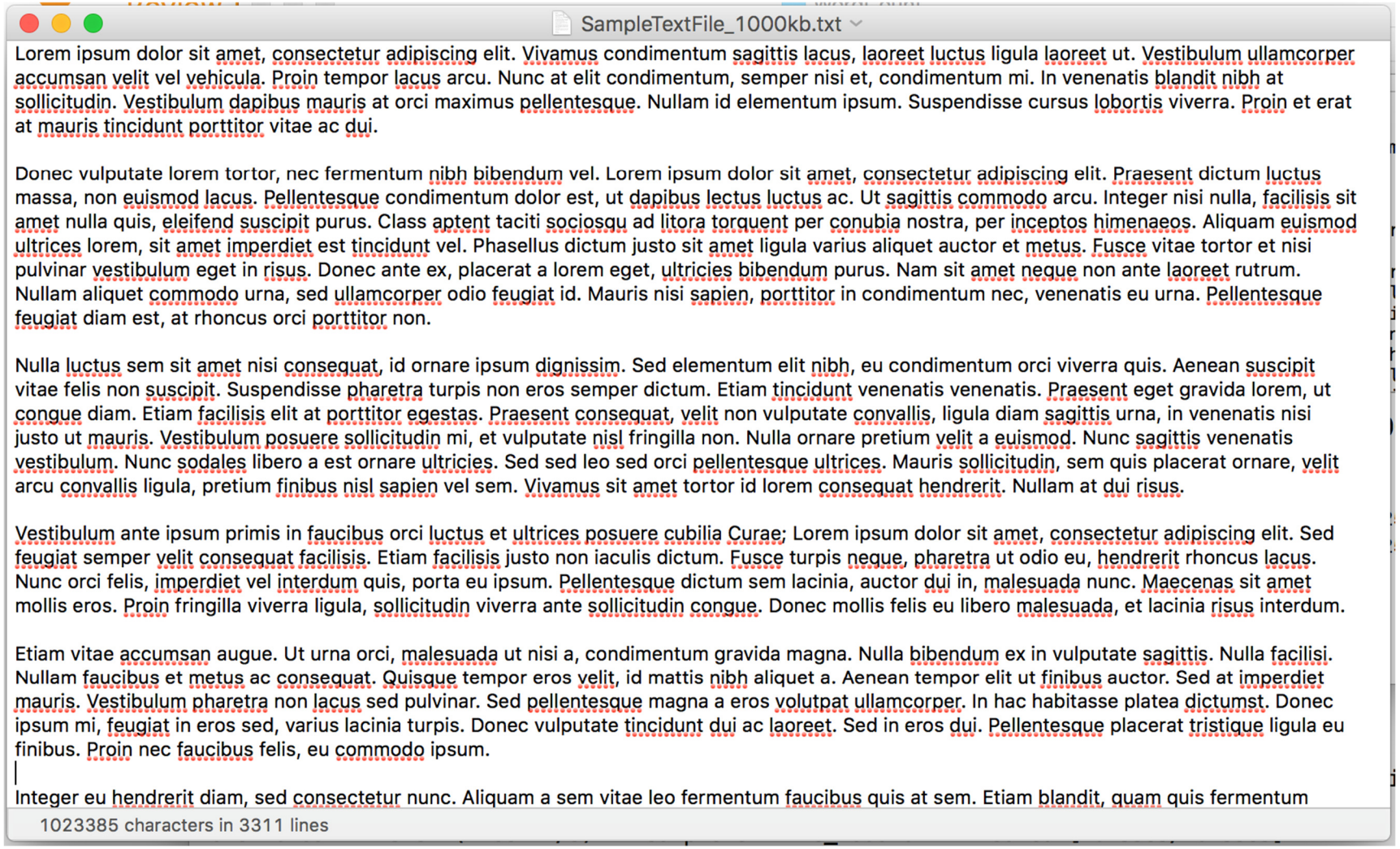
```
$ sbin/stop-yarn.sh
$ sbin/stop-dfs.sh
```

```
Daqings-MacBook-Pro:hadoop-2.9.1 daqingyun$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.9.1.jar grep input output 'dfs[a-z.]+'
18/10/09 23:36:21 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/10/09 23:36:22 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/10/09 23:36:23 INFO input.FileInputFormat: Total input files to process : 30
18/10/09 23:36:23 INFO mapreduce.JobSubmitter: number of splits:30
18/10/09 23:36:23 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
18/10/09 23:36:23 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1539139171096_0004
18/10/09 23:36:24 INFO impl.YarnClientImpl: Submitted application application_1539139171096_0004
18/10/09 23:36:24 INFO mapreduce.Job: The url to track the job: http://Daqings-MacBook-Pro.local:8088/proxy/application_1539139171096_0004/
18/10/09 23:36:24 INFO mapreduce.Job: Running job: job_1539139171096_0004
18/10/09 23:36:31 INFO mapreduce.Job: Job job_1539139171096_0004 running in uber mode : false
18/10/09 23:36:31 INFO mapreduce.Job:  map 0% reduce 0%
18/10/09 23:36:40 INFO mapreduce.Job:  map 3% reduce 0%
18/10/09 23:36:44 INFO mapreduce.Job:  map 7% reduce 0%
18/10/09 23:36:47 INFO mapreduce.Job:  map 10% reduce 0%
18/10/09 23:36:48 INFO mapreduce.Job:  map 13% reduce 0%
18/10/09 23:36:49 INFO mapreduce.Job:  map 17% reduce 0%
18/10/09 23:36:50 INFO mapreduce.Job:  map 20% reduce 0%
18/10/09 23:36:52 INFO mapreduce.Job:  map 23% reduce 0%
18/10/09 23:36:56 INFO mapreduce.Job:  map 27% reduce 0%
18/10/09 23:37:00 INFO mapreduce.Job:  map 30% reduce 0%
18/10/09 23:37:01 INFO mapreduce.Job:  map 33% reduce 0%
18/10/09 23:37:02 INFO mapreduce.Job:  map 37% reduce 0%
18/10/09 23:37:04 INFO mapreduce.Job:  map 40% reduce 0%
18/10/09 23:37:07 INFO mapreduce.Job:  map 43% reduce 0%
18/10/09 23:37:11 INFO mapreduce.Job:  map 47% reduce 0%
18/10/09 23:37:12 INFO mapreduce.Job:  map 53% reduce 0%
Daqings-MacBook-Pro:hadoop-2.9.1 daqingyun$ bin/hdfs dfs -get output ./
18/10/09 23:39:49 WARN util.NativeCodeLoader: Unable to load native-hadoop
tform... using builtin-java classes where applicable
Daqings-MacBook-Pro:hadoop-2.9.1 daqingyun$ cat output/part-r-00000
6       dfs.audit.logger
4       dfs.class
3       dfs.logger
3       dfs.server.namenode.
2       dfs.audit.log.maxbackupindex
2       dfs.period
2       dfs.audit.log.maxfilesize
1       dfs.log
1       dfs.file
1       dfs.servers
1       dfsadmin
1       dfsmetrics.log
1       dfs.replication
Daqings-MacBook-Pro:hadoop-2.9.1 daqingyun$
```

World Count Problem:
Hands-on MapReduce Programming Guide

- Version: Hadoop 2.9.1
- Mode: Pseudo-Distributed Mode
- OS: Mac

World Count Problem: Hands-on MapReduce Programming Guide – Input (a locally stored file)



World Count Problem: Hands-on MapReduce Programming Guide – MapReduce program

Source code: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example:_WordCount_v1.0

WordCount.java

```
1  import java.io.IOException;
2  import java.util.StringTokenizer;
3
4  import org.apache.hadoop.conf.Configuration;
5  import org.apache.hadoop.fs.Path;
6  import org.apache.hadoop.io.IntWritable;
7  import org.apache.hadoop.io.Text;
8  import org.apache.hadoop.mapreduce.Job;
9  import org.apache.hadoop.mapreduce.Mapper;
10 import org.apache.hadoop.mapreduce.Reducer;
11 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
12 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
13
14 public class WordCount {
15
16   public static class TokenizerMapper
17        extends Mapper<Object, Text, Text, IntWritable>{
18
19     private final static IntWritable one = new IntWritable(1);
20     private Text word = new Text();
21
22     public void map(Object key, Text value, Context context
23                     ) throws IOException, InterruptedException {
24       StringTokenizer itr = new StringTokenizer(value.toString());
25       while (itr.hasMoreTokens()) {
26         word.set(itr.nextToken());
27         context.write(word, one);
28       }
29     }
30   }
```

WordCount.java

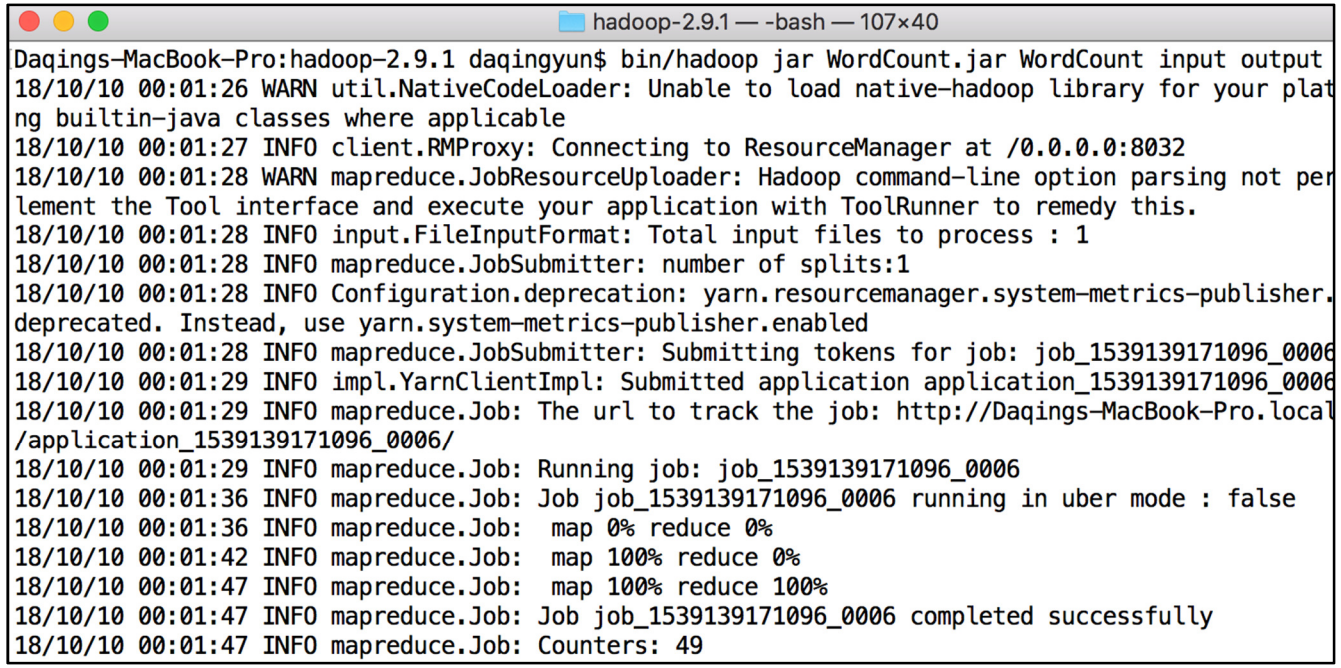
```
32   public static class IntSumReducer
33        extends Reducer<Text,IntWritable,Text,IntWritable> {
34     private IntWritable result = new IntWritable();
35
36     public void reduce(Text key, Iterable<IntWritable> values,
37                        Context context
38                        ) throws IOException, InterruptedException {
39       int sum = 0;
40       for (IntWritable val : values) {
41         sum += val.get();
42       }
43       result.set(sum);
44       context.write(key, result);
45     }
46   }
47
```

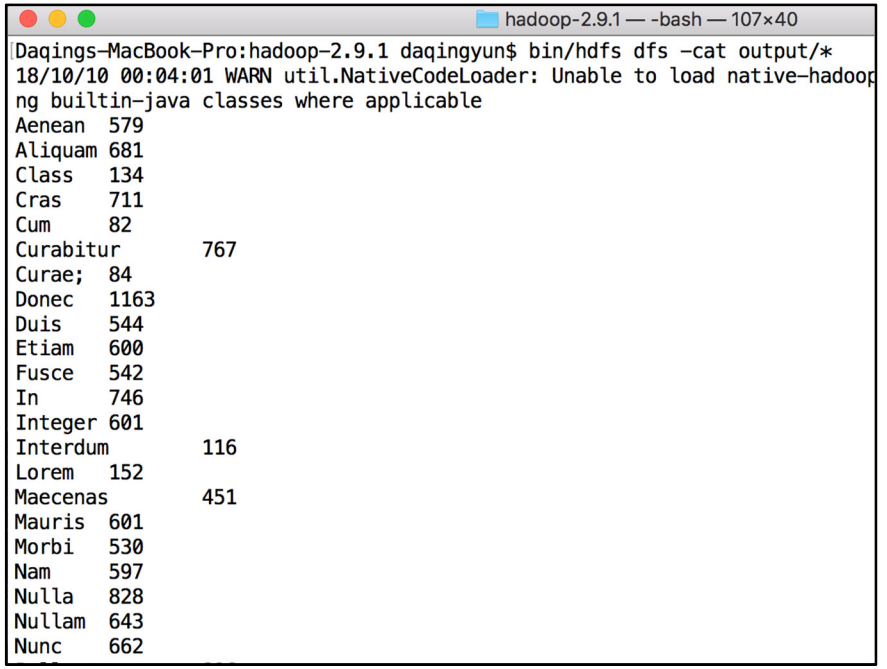
WordCount.java

```
49
50   public static void main(String[] args) throws Exception {
51     Configuration conf = new Configuration();
52     Job job = Job.getInstance(conf, "WordCount");
53     job.setJarByClass(WordCount.class);
54     job.setMapperClass(TokenizerMapper.class);
55     job.setCombinerClass(IntSumReducer.class);
56     job.setReducerClass(IntSumReducer.class);
57     job.setOutputKeyClass(Text.class);
58     job.setOutputValueClass(IntWritable.class);
59     FileInputFormat.addInputPath(job, new Path(args[0]));
60     FileOutputFormat.setOutputPath(job, new Path(args[1]));
61     System.exit(job.waitForCompletion(true) ? 0 : 1);
62   }
63 }
```

World Count Problem: Hands-on MapReduce Programming Guide – MapReduce program

- bin/hadoop com.sun.tools.javac.Main WordCount.java
- jar cf WordCount.jar WordCount*.class
- bin/hadoop jar WordCount.jar WordCount input output
- bin/hdfs dfs -cat output/*





World Count Problem: Hands-on MapReduce Programming Guide

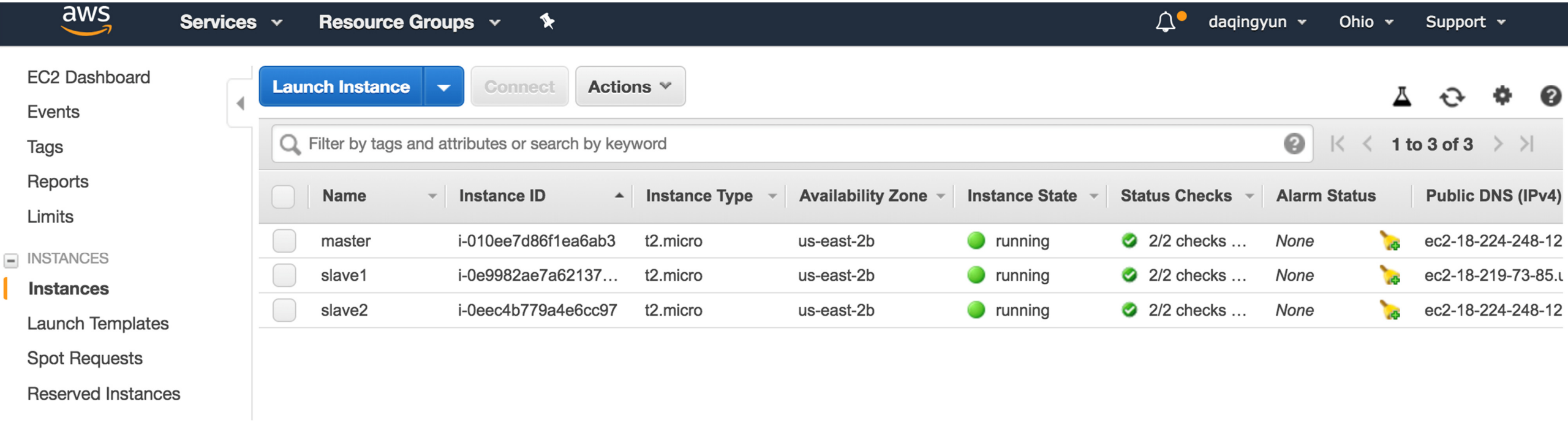
- Version: Hadoop 2.9.1
- Mode: Fully-Distributed Mode
- Cloud: Amazon Web Services

If we want to run a MapReduce program in a Fully-Distributed Mode on a Hadoop cluster, for example, in a public cloud environment, we can upload the JAR file to the master node of the cluster (or compile it there) and execute the program by using the following command:

$ bin/hadoop jar WordCount.jar WordCount /user/user_name/input /user/user_name/output

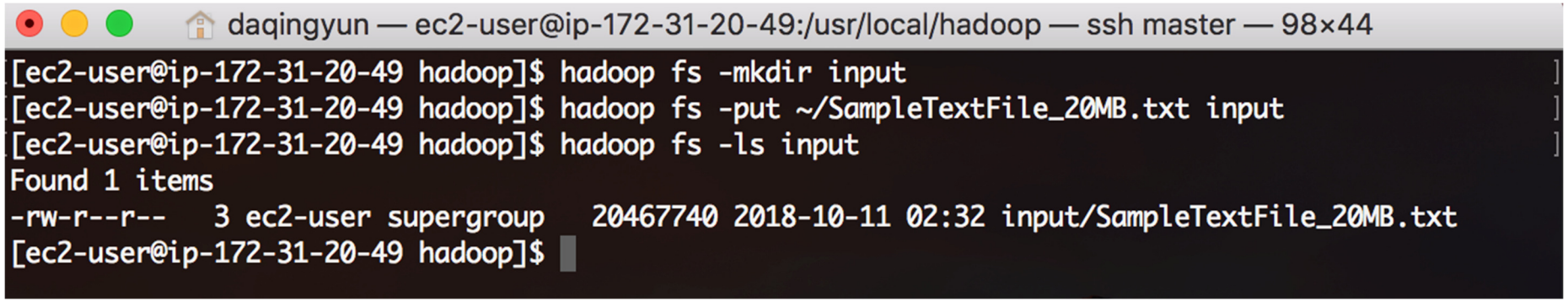
World Count Problem:
Hands-on MapReduce Programming Guide

- Three homogenous VM instances: one master node, and two slave nodes



World Count Problem:
Hands-on MapReduce Programming Guide

- Create the input directory in HDFS and upload the input data from local to this directory



World Count Problem: Hands-on MapReduce Programming Guide – MapReduce program

Compile jar package, upload input data, execute, and check out results

- Assuming environments are set
  - export JAVA_HOME=/usr/java/default
  - export PATH=${JAVA_HOME}/bin:${PATH}
  - export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
- bin/hadoop com.sun.tools.javac.Main WordCount.java
- jar cf WordCount.jar WordCount*.class
- bin/hdfs dfs -mkdir /user
- bin/hdfs dfs -mkdir /user/ec2-user
- bin/hdfs dfs -put SampleTextFile_20MB.txt
- bin/hadoop jar WordCount.jar WordCount input output
- bin/hdfs dfs -cat output/*

https://youtu.be/cr5RmnyWbYw
https://youtu.be/lRQqR0Fm1oE
https://youtu.be/u7flf_R-gaM
https://youtu.be/Z0cOE2SRo5c
https://youtu.be/PF1dUAKs_bg
https://youtu.be/k-DGPIwfitM
https://youtu.be/Z_5pJMCoeM8

World Count Problem: Hands-on MapReduce Programming Guide – MapReduce program

http://ec2-18-224-248-120.us-east-2.compute.amazonaws.com:50070/explorer.html#/user/ec2-user/input

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse Directory

/user/ec2-user/input Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	ec2-user	supergroup	19.52 MB	Oct 10 22:32	3	16 MB	SampleTextFile_20MB.txt

Showing 1 to 1 of 1 entries Previous 1 Next

Hadoop, 2018.

World Count Problem: Hands-on MapReduce Programming Guide – MapReduce program

http://ec2-18-224-248-120.us-east-2.compute.amazonaws.com:8088/cluster/scheduler

NEW,NEW_SAVING,SUBMITTED,ACCEPTED

- Cluster
  - About
  - Nodes
  - Node Labels
  - Applications
    - NEW
    - NEW_SAVING
    - SUBMITTED
    - ACCEPTED
    - RUNNING
    - FINISHED
    - FAILED
    - KILLED
  - Scheduler
- Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memo
10	1	1	8	3	4 GB	16 GB

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes
2	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:81

Dump scheduler logs 1 min

Application Queues

Legend: Capacity | Used | Used (over capacity) | Max Capacity | Users Requesting Resources

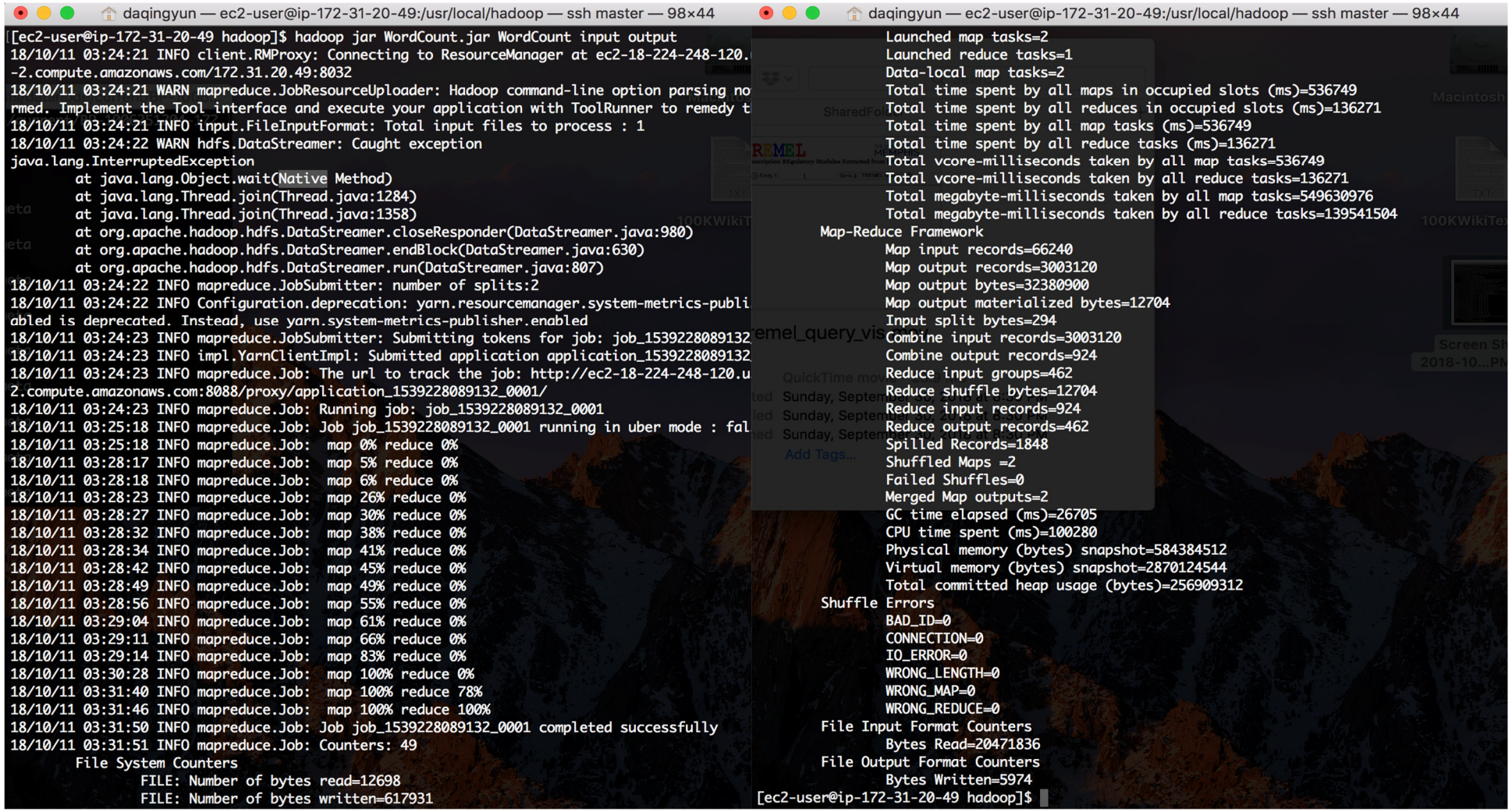
- Queue: root
  - Queue: default

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers

World Count Problem: Hands-on MapReduce Programming Guide – MapReduce program

- Execute WordCount.jar



World Count Problem: Hands-on MapReduce Programming Guide – MapReduce program

- Download the output folder from HDFS to the master node. Find out the size of the output is <6K, and display the word counting results

daqingyun — ec2-user@ip-172-31-20-49:/usr/local/hadoop/output — ssh master — 98×44

```
[ec2-user@ip-172-31-20-49 output]$ hadoop fs -ls -h output
Found 2 items
-rw-r--r--   3 ec2-user supergroup          0 2018-10-11 02:35 output/_SUCCESS
-rw-r--r--   3 ec2-user supergroup      5.8 K 2018-10-11 02:35 output/part-r-00000
[ec2-user@ip-172-31-20-49 output]$ hadoop fs -get output .
[ec2-user@ip-172-31-20-49 output]$ ls -h output/*
output/part-r-00000  output/_SUCCESS
[ec2-user@ip-172-31-20-49 output]$ ll -h output/*
-rw-r--r-- 1 ec2-user ec2-user 5.9K Oct 11 02:45 output/part-r-00000
-rw-r--r-- 1 ec2-user ec2-user    0 Oct 11 02:45 output/_SUCCESS
[ec2-user@ip-172-31-20-49 output]$
```

daqingyun — ec2-user@ip-172-31-20-49:/usr/local/hadoop/output — ssh master — 98×44

```
[ec2-user@ip-172-31-20-49 output]$ cat output/part-r-00000 | more
Aenean  11580
Aliquam 13620
Class   2680
Cras    14220
Cum     1640
Curabitur       15340
Curae;  1680
Donec   23260
Duis    10880
Etiam   12000
Fusce   10840
In      14920
Integer 12020
Interdum        2320
```

- Check slave nodes

daqingyun — ec2-user@ip-172-31-25-239:/usr/local/hadoop/data/hdfs/datar

slave2

```
[ec2-user@ip-172-31-25-239 subdir0]$ ll -h
total 20M
-rw-rw-r-- 1 ec2-user ec2-user  16M Oct 11 03:23 blk_1073741825
-rw-rw-r-- 1 ec2-user ec2-user 129K Oct 11 03:23 blk_1073741825_1001.meta
-rw-rw-r-- 1 ec2-user ec2-user 3.6M Oct 11 03:23 blk_1073741826
-rw-rw-r-- 1 ec2-user ec2-user  29K Oct 11 03:23 blk_1073741826_1002.meta
-rw-rw-r-- 1 ec2-user ec2-user 5.9K Oct 11 03:31 blk_1073741833
-rw-rw-r-- 1 ec2-user ec2-user   55 Oct 11 03:31 blk_1073741833_1009.meta
-rw-rw-r-- 1 ec2-user ec2-user  354 Oct 11 03:31 blk_1073741834
-rw-rw-r-- 1 ec2-user ec2-user   11 Oct 11 03:31 blk_1073741834_1010.meta
-rw-rw-r-- 1 ec2-user ec2-user  41K Oct 11 03:31 blk_1073741835
-rw-rw-r-- 1 ec2-user ec2-user  331 Oct 11 03:31 blk_1073741835_1011.meta
-rw-rw-r-- 1 ec2-user ec2-user 191K Oct 11 03:31 blk_1073741836
-rw-rw-r-- 1 ec2-user ec2-user 1.5K Oct 11 03:31 blk_1073741836_1012.meta
[ec2-user@ip-172-31-25-239 subdir0]$ head -3 blk_1073741826
osuere nulla a convallis. Nunc aliquam, mi vel faucibus dapibus, risus leo al
m lacus, sit amet semper dui mauris a sem. Etiam pretium elit erat, vitae bla
 metus aliquam in. Pellentesque non nisl at mauris porta scelerisque. Cras vi
malesuada tellus.

Fusce ac velit nunc. Phasellus eu aliquet justo. Aliquam id aliquam odio. Viv
 sollicitudin pharetra pharetra. Morbi nec blandit sem, quis fermentum erat.
entesque sed risus sit amet nunc eleifend tincidunt. Curabitur sit amet males
 libero. In pretium rutrum mauris vitae suscipit. Nulla tristique posuere eni
is mattis. Vestibulum tincidunt, tortor quis efficitur convallis, mi nisi fau
s sem, vel cursus mauris lectus eu nunc. Fusce at malesuada nulla, ut mattis
 Nunc porta tristique laoreet. Aenean non est augue. Phasellus congue massa c
, ac volutpat sapien viverra ut. Pellentesque habitant morbi tristique senect
t netus et malesuada fames ac turpis egestas.
[ec2-user@ip-172-31-25-239 subdir0]$ head -3 blk_1073741833
Aenean  11580
Aliquam 13620
Class   2680
[ec2-user@ip-172-31-25-239 subdir0]$
```

daqingyun — ec2-user@ip-172-31-20-104:/usr/local/hadoop/data/hdfs/datanode/...

slave1

```
[ec2-user@ip-172-31-20-104 subdir0]$ ll -h
total 20M
-rw-rw-r-- 1 ec2-user ec2-user  16M Oct 11 03:23 blk_1073741825
-rw-rw-r-- 1 ec2-user ec2-user 129K Oct 11 03:23 blk_1073741825_1001.meta
-rw-rw-r-- 1 ec2-user ec2-user 3.6M Oct 11 03:23 blk_1073741826
-rw-rw-r-- 1 ec2-user ec2-user  29K Oct 11 03:23 blk_1073741826_1002.meta
-rw-rw-r-- 1 ec2-user ec2-user 5.9K Oct 11 03:31 blk_1073741833
-rw-rw-r-- 1 ec2-user ec2-user   55 Oct 11 03:31 blk_1073741833_1009.meta
-rw-rw-r-- 1 ec2-user ec2-user  354 Oct 11 03:31 blk_1073741834
-rw-rw-r-- 1 ec2-user ec2-user   11 Oct 11 03:31 blk_1073741834_1010.meta
-rw-rw-r-- 1 ec2-user ec2-user  41K Oct 11 03:31 blk_1073741835
-rw-rw-r-- 1 ec2-user ec2-user  331 Oct 11 03:31 blk_1073741835_1011.meta
-rw-rw-r-- 1 ec2-user ec2-user 191K Oct 11 03:31 blk_1073741836
-rw-rw-r-- 1 ec2-user ec2-user 1.5K Oct 11 03:31 blk_1073741836_1012.meta
[ec2-user@ip-172-31-20-104 subdir0]$ head -5 blk_1073741826
osuere nulla a convallis. Nunc aliquam, mi vel faucibus dapibus, risus leo aliquam
lacus, sit amet semper dui mauris a sem. Etiam pretium elit erat, vitae blandit met
us aliquam in. Pellentesque non nisl at mauris porta scelerisque. Cras vitae malesu
ada tellus.

Fusce ac velit nunc. Phasellus eu aliquet justo. Aliquam id aliquam odio. Vivamus s
ollicitudin pharetra pharetra. Morbi nec blandit sem, quis fermentum erat. Pellente
sque sed risus sit amet nunc eleifend tincidunt. Curabitur sit amet malesuada liber
o. In pretium rutrum mauris vitae suscipit. Nulla tristique posuere enim quis matti
s. Vestibulum tincidunt, tortor quis efficitur convallis, mi nisi faucibus sem, vel
 cursus mauris lectus eu nunc. Fusce at malesuada nulla, ut mattis dui. Nunc porta
tristique laoreet. Aenean non est augue. Phasellus congue massa dolor, ac volutpat
sapien viverra ut. Pellentesque habitant morbi tristique senectus et netus et males
uada fames ac turpis egestas.

Integer cursus turpis non leo vehicula molestie. Cras pellentesque sagittis eros eu
 vulputate. Nullam in lacinia diam. Aenean iaculis vulputate imperdiet. Ut dictum a
ccumsan sapien, ac egestas quam aliquam eget. Mauris malesuada mauris ac scelerisqu
e dapibus. Nunc est augue, imperdiet sit amet viverra eu, dictum quis sapien. Nulla
m pretium non ligula non interdum. Curabitur vitae lorem vitae orci ullamcorper luc
tus malesuada eget ligula. Integer sagittis porttitor eleifend. Quisque consectetur
 mi et auctor dictum. Pellentesque auctor nulla magna, non tincidunt purus malesuad
a eu. Cras consequat nisl nunc. Duis maximus at risus sit amet luctus. Sed sit amet
 dui ac velit egestas dictum. Vestibulum eu tellus vulputate, mattis nunc suscipit,
 porta nisl.
[ec2-user@ip-172-31-20-104 subdir0]$ head -3 blk_1073741833
Aenean  11580
Aliquam 13620
Class   2680
```

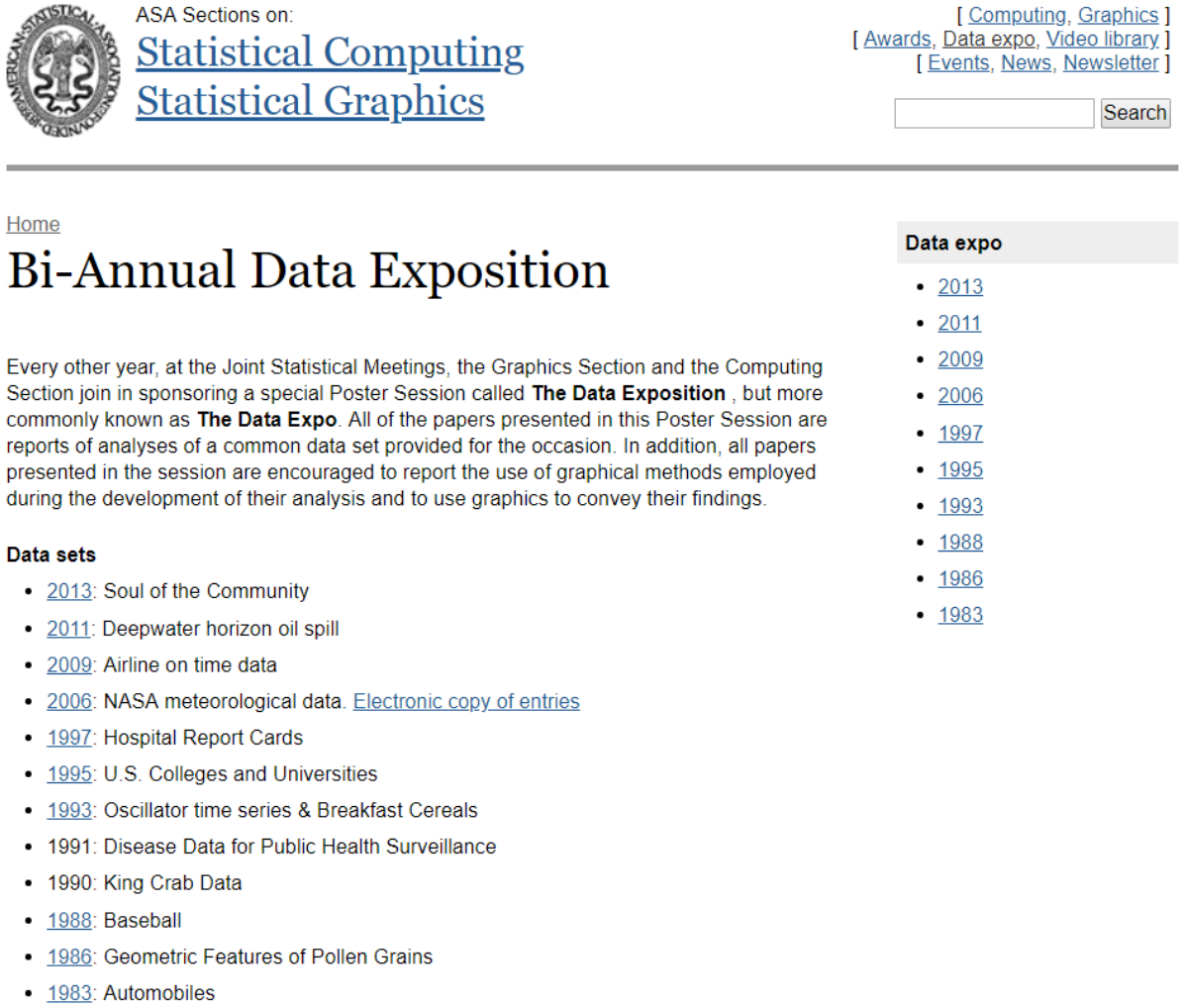
MapReduce Use Case Example Flight Data Analysis

- Airline On-time Performance data set (flight data set)
  - All the logs of domestic flights from the period of October 1987 to April 2008
  - Each record represents an individual flight where various details are captured
    - Time and date of arrival and departure
    - Originating and destination airports
    - Amount of time taken to taxi from the runway to the gate

- Download it from Statistical Computing:
  - http://stat-computing.org/dataexpo/2009/

Other datasets available from Statistical Computing

http://stat-computing.org/dataexpo/



ASA Sections on:
Statistical Computing
Statistical Graphics

[Computing, Graphics]
[Awards, Data expo, Video library]
[Events, News, Newsletter]

Search

Home

Bi-Annual Data Exposition

Every other year, at the Joint Statistical Meetings, the Graphics Section and the Computing Section join in sponsoring a special Poster Session called **The Data Exposition** , but more commonly known as **The Data Expo**. All of the papers presented in this Poster Session are reports of analyses of a common data set provided for the occasion. In addition, all papers presented in the session are encouraged to report the use of graphical methods employed during the development of their analysis and to use graphics to convey their findings.

Data sets

- 2013: Soul of the Community
- 2011: Deepwater horizon oil spill
- 2009: Airline on time data
- 2006: NASA meteorological data. Electronic copy of entries
- 1997: Hospital Report Cards
- 1995: U.S. Colleges and Universities
- 1993: Oscillator time series & Breakfast Cereals
- 1991: Disease Data for Public Health Surveillance
- 1990: King Crab Data
- 1988: Baseball
- 1986: Geometric Features of Pollen Grains
- 1983: Automobiles

Data expo

- 2013
- 2011
- 2009
- 2006
- 1997
- 1995
- 1993
- 1988
- 1986
- 1983

Flight Data Scheme

Variable descriptions

	Name	Description
1	Year	1987-2008
2	Month	1-12
3	DayofMonth	1-31
4	DayOfWeek	1 (Monday) - 7 (Sunday)
5	DepTime	actual departure time (local, hhmm)
6	CRSDepTime	scheduled departure time (local, hhmm)
7	ArrTime	actual arrival time (local, hhmm)
8	CRSArrTime	scheduled arrival time (local, hhmm)
9	UniqueCarrier	unique carrier code
10	FlightNum	flight number
11	TailNum	plane tail number
12	ActualElapsedTime	in minutes
13	CRSElapsedTime	in minutes
14	AirTime	in minutes
15	ArrDelay	arrival delay, in minutes
16	DepDelay	departure delay, in minutes
17	Origin	origin IATA airport code
18	Dest	destination IATA airport code
19	Distance	in miles
20	TaxiIn	taxi in time, in minutes
21	TaxiOut	taxi out time in minutes
22	Cancelled	was the flight cancelled?
23	CancellationCode	reason for cancellation (A = carrier, B = weather, C = NAS, D = security)
24	Diverted	1 = yes, 0 = no
25	CarrierDelay	in minutes
26	WeatherDelay	in minutes
27	NASDelay	in minutes
28	SecurityDelay	in minutes
29	LateAircraftDelay	in minutes

ASA Sections on:
Statistical Computing
Statistical Graphics

Data expo '09

http://stat-computing.org/dataexpo/

MapReduce Use Case Example Flight Data Analysis

- Count the number of flights for each carrier
  - Serial way (not MapReduce):

Pseudocode for Calculating The Number of Flights By Carrier Serially

```
create a two-dimensional array
  create a row for every airline carrier
    populate the first column with the carrier code
    populate the second column with the integer zero

for each line of flight data
  read the airline carrier code
  find the row in the array that matches the carrier code
    increment the counter in the second column by one

print the totals for each row in the two-dimensional array
```

D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

MapReduce Use Case Example Flight Data Analysis

- Count the number of flights for each carrier
  - Parallel way (MapReduce):

Pesudocode for Calculating The Number of Flights

```
Map Phase:
  for each line of flight data
    read the current record and extract the airline carrier code
    output the airline carrier code and the number one as a key/value pair

Shuffle and Sort Phase:
  read the list of key/value pairs from the map phase
  group all the values for each key together
    each key has a corresponding array of values
  sort the data by key
  output each key and its array of values

Reduce Phase:
  read the list of carriers and arrays of values from the shuffle and sort phase
  for each carrier code
    add the total number of ones in the carrier code's array of values together

print the totals for each row in the two-dimensional array
```

D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

MapReduce Application Flow

1. Determine the exact data sets to process from the data blocks. This involves calculating where the records to be processed are located within the data blocks.

2. Run the specified algorithm against each record in the data set until all the records are processed. The individual instance of the application running against a block of data in a data set is known as a mapper task. (This is the mapping part of MapReduce.)

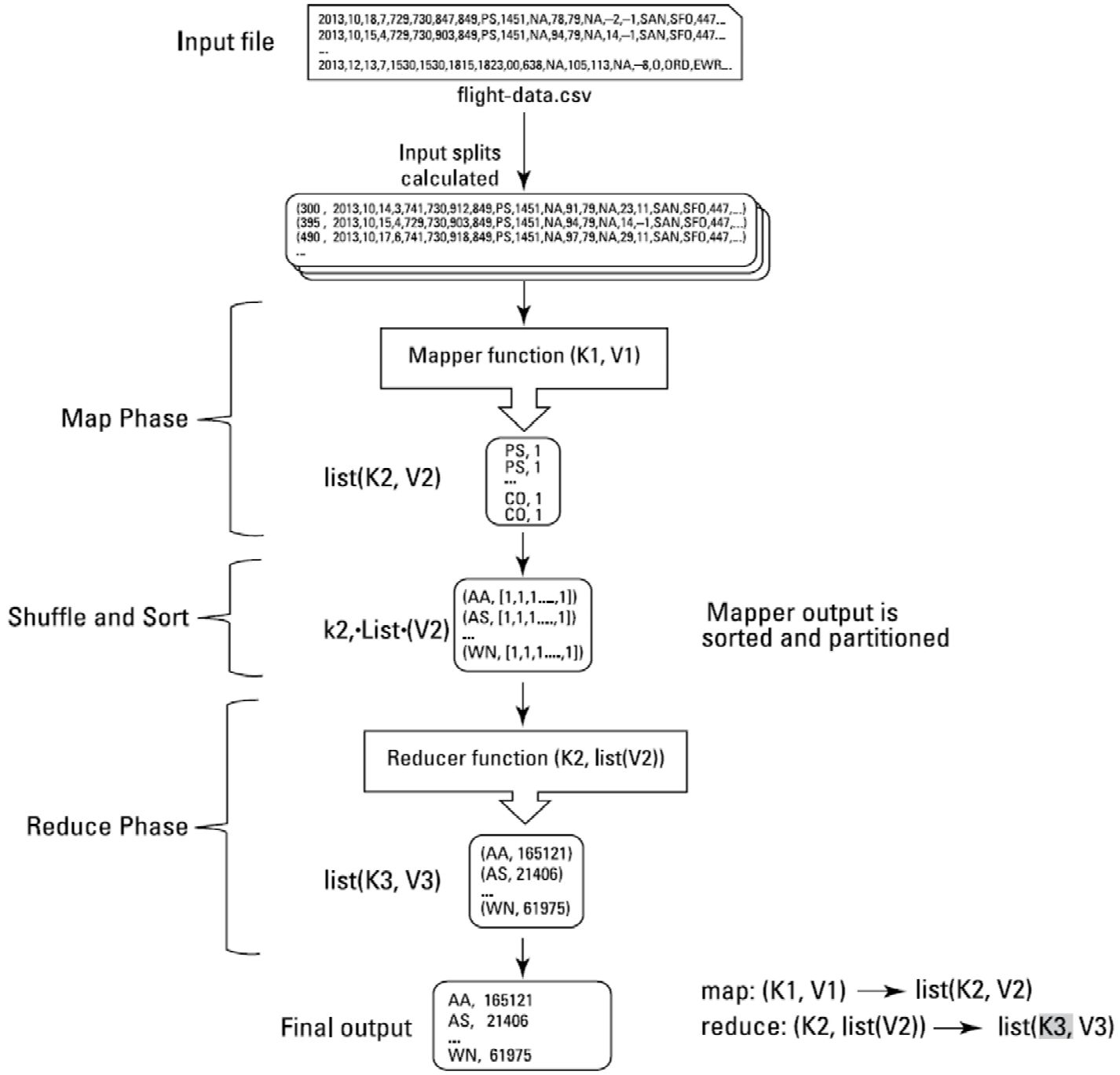
3. Locally perform an interim reduction of the output of each mapper. (The outputs are provisionally combined, in other words.) This phase is optional because, in some common cases, it isn't desirable.

4. Based on partitioning requirements, group the applicable partitions of data from each mapper's result sets.

5. Boil down the result sets from the mappers into a single result set — the Reduce part of MapReduce. An individual instance of the application running against mapper output data is known as a reducer task.

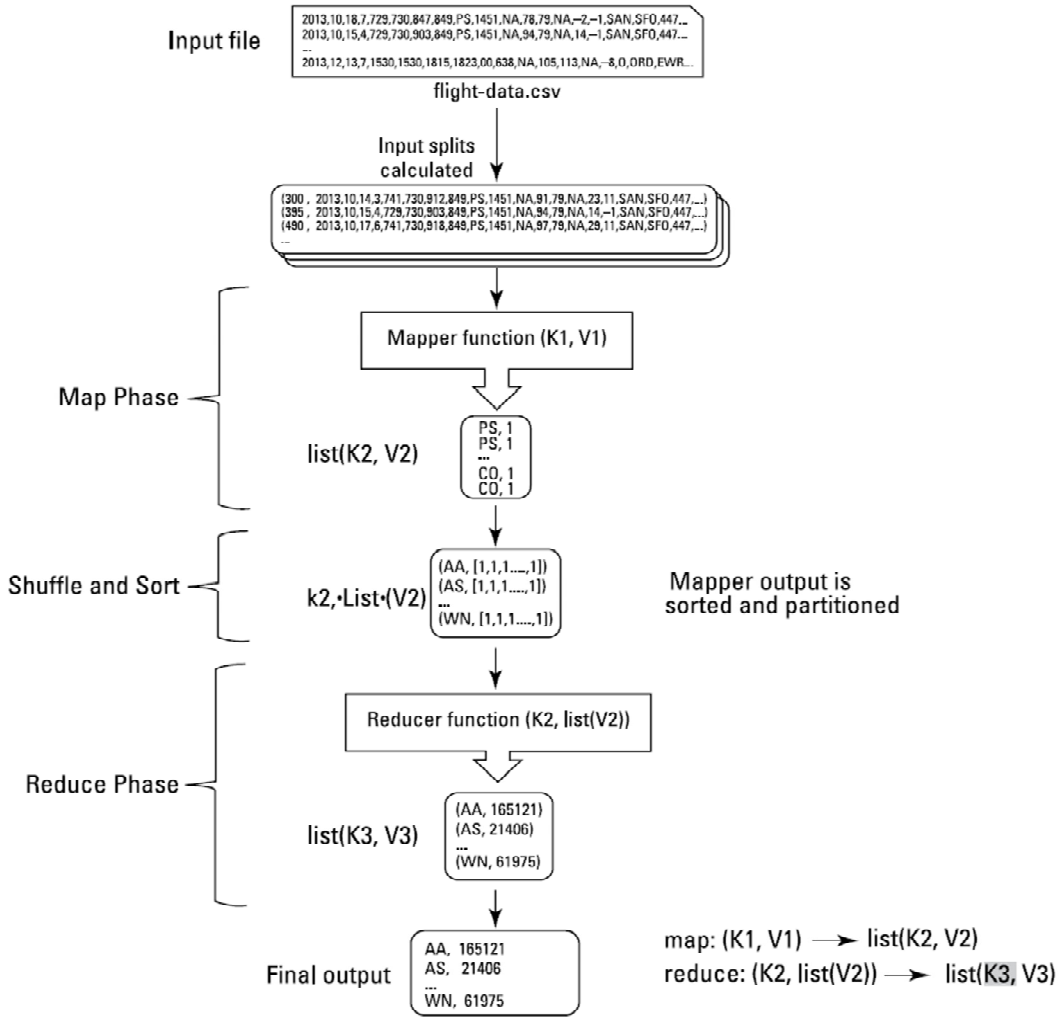
D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

MapReduce steps for flight data computation



D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

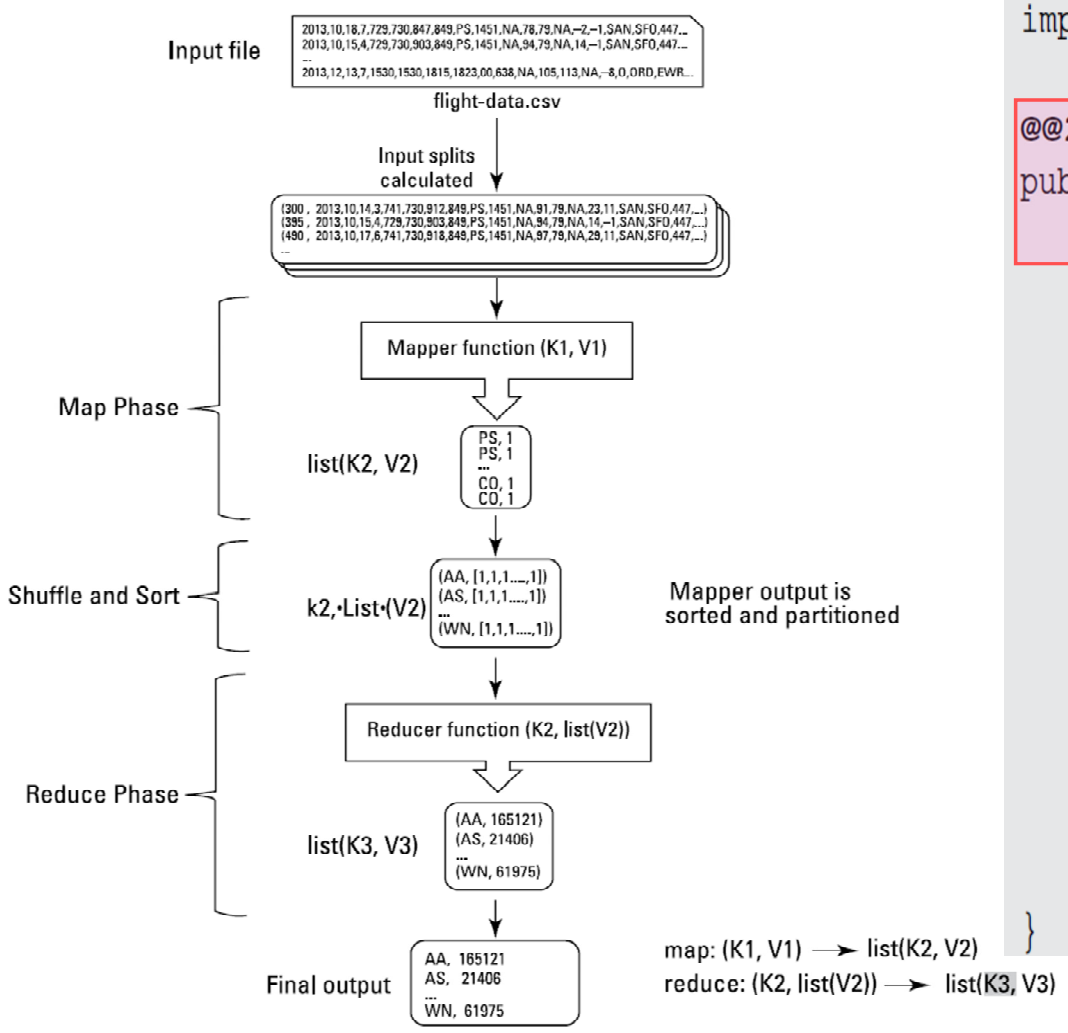
FlightsByCarrier Driver Application




```
@@1
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class FlightsByCarrier {
       public static void main(String[] args) throws Exception {
              @@2
              Job job = new Job();
              job.setJarByClass(FlightsByCarrier.class);
              job.setJobName("FlightsByCarrier");

              @@3
              TextInputFormat.addInputPath(job, new Path(args[0]));
              job.setInputFormatClass(TextInputFormat.class);

              @@4
              job.setMapperClass(FlightsByCarrierMapper.class);
              job.setReducerClass(FlightsByCarrierReducer.class);

              @@5
              TextOutputFormat.setOutputPath(job, new Path(args[1]));
              job.setOutputFormatClass(TextOutputFormat.class);
              job.setOutputKeyClass(Text.class);
              job.setOutputValueClass(IntWritable.class);
              @@6
              job.waitForCompletion(true);
       }
}
```

D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

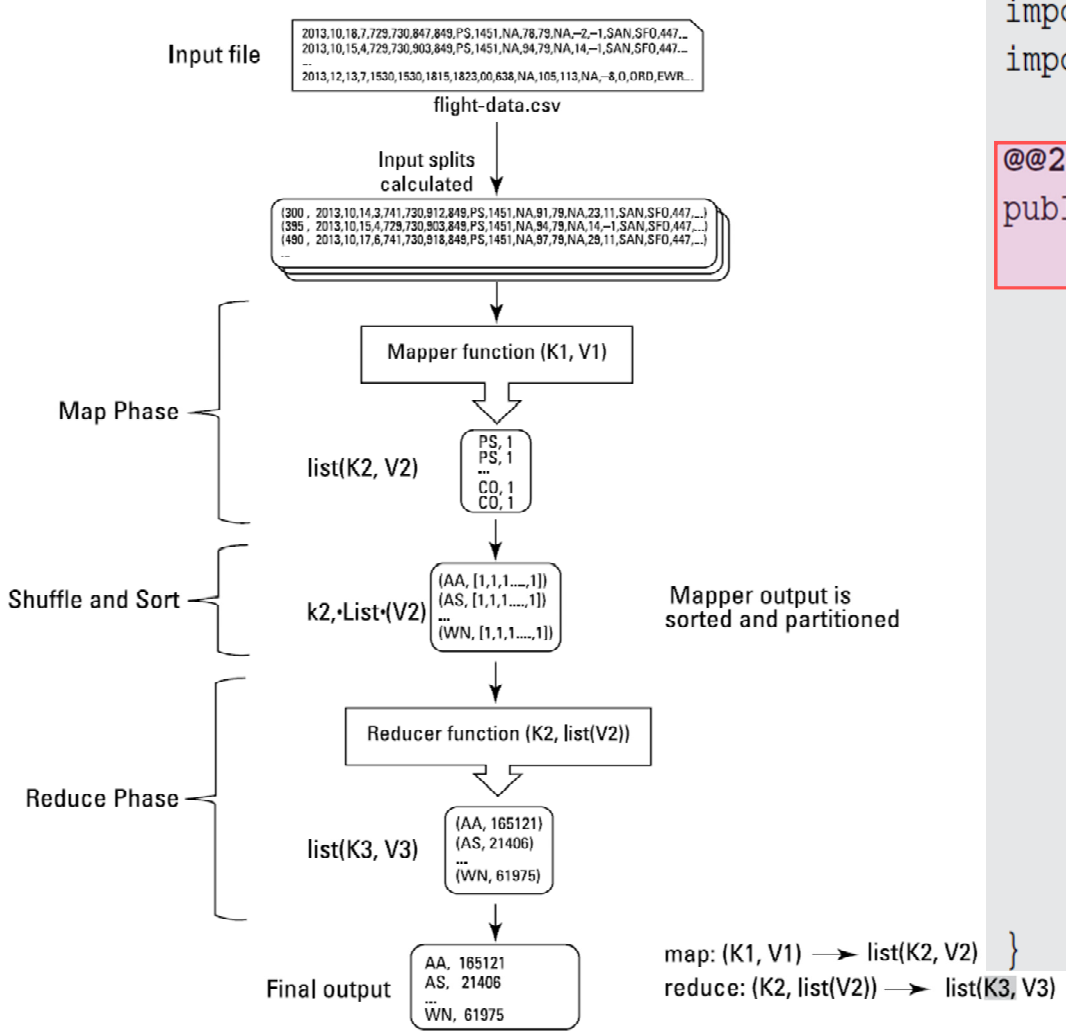
FlightsByCarrier Mapper Code




```
@@1
import java.io.IOException;
import au.com.bytecode.opencsv.CSVParser;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Mapper;

@@2
public class FlightsByCarrierMapper extends
                                Mapper<LongWritable, Text, Text, IntWritable> {
    @Override
    @@3
    protected void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            @@4
            if (key.get() > 0) {
                String[] lines = new
            CSVParser().parseLine(value.toString());
                @@5
                context.write(new Text(lines[8]), new IntWritable(1));
            }
    }
}
```

D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

FlightsByCarrier Reducer Code




```
@@1
import java.io.IOException;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Reducer;

@@2
public class FlightsByCarrierReducer extends
                Reducer<Text, IntWritable, Text, IntWritable> {
        @Override
        @@3
        protected void reduce(Text token, Iterable<IntWritable> counts,
                        Context context) throws IOException, InterruptedException {
                int sum = 0;

                @@4
                for (IntWritable count : counts) {
                        sum+= count.get();
                }
                @@5
                context.write(token, new IntWritable(sum));
        }
}
```

D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

Execution

To run the FlightsByCarrier application, follow these steps

1. Go the directory with your Java code and compile it using the following command

```
javac -classpath $CLASSPATH MapRed/FlightsByCarrier/*.java
```

2. Build a JAR file for the application by using this command

```
jar cvf FlightsByCarrier.jar *.class
```

3. Run the driver application by using this command:

```
hadoop jar FlightsByCarrier /user/root/airline-data/2008.csv \
/user/root/output/flightsCount
```

D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

See Results

4. Show the job's output file from HDFS by running the command

```
hadoop fs -cat /user/root/output/flightsCount/part-r-00000
```

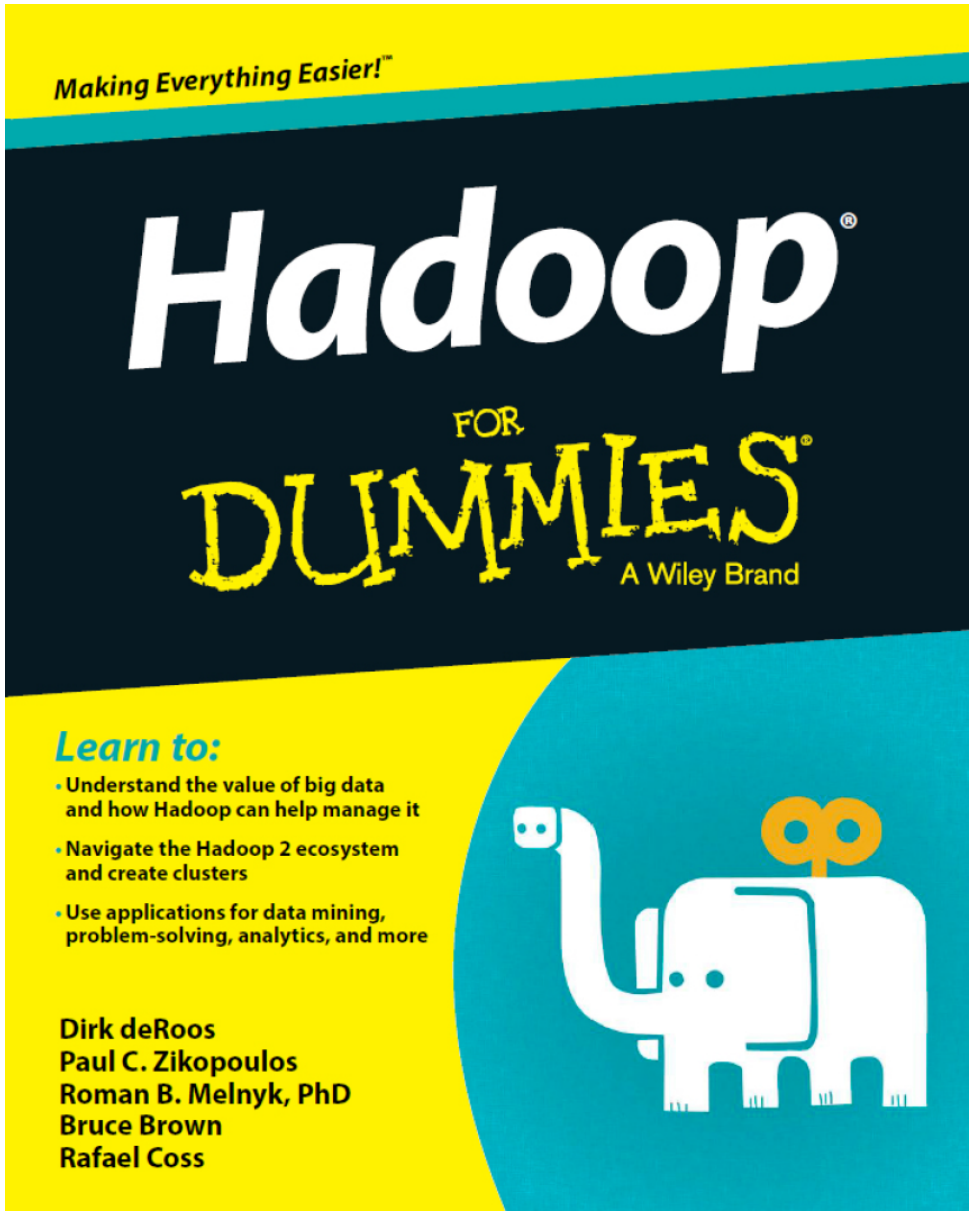
The total counts of all flights completed for each of the carriers in 2008:

```
AA          165121
AS          21406
CO          123002
DL          185813
EA          108776
HP          45399
NW          108273
PA (1)      16785
PI          116482
PS          41706
TW          69650
UA          152624
US          94814
WN          61975
```

Further Reading

- K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *Proc. of the 26th IEEE Symp. on Mass Storage Systems and Technologies*, Washington, DC, pp. 1-10.
  - https://goo.gl/WT90cp

Reading Reference



Java programming

- How to MapReduce programming with Apache
  - https://www.javaworld.com/article/2077907/open-source-tools/mapreduce-programming-with-apache-hadoop.html

- If you need more details, the following book helps
  - https://piazza-resources.s3.amazonaws.com/ist3pwd6k8p5t/iu5gqbsh8re6mj/OReilly.Hadoop.The.Definitive.Guide.4th.Edition.2015.pdf

- The following tutorial shows you how to use Eclipse to write, compile, and execute the export .jar file for the word counting problem in Hadoop in detail
  - https://www.dezyre.com/hadoop-tutorial/hadoop-mapreduce-wordcount-tutorial

Hadoop configuration

- Standalone mode
  - https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html#Standalone_Operation
- Pseudo-distributed mode
  - https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html#Pseudo-Distributed_Operation
- Fully-distributed mode
  - https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/ClusterSetup.html
- Another useful tutorial
  - https://www.edureka.co/blog/install-hadoop-single-node-hadoop-cluster

Resources

- Hadoop documentation
  - http://hadoop.apache.org/docs/current/
- Projects powered by Apache Hadoop
  - https://wiki.apache.org/hadoop/PoweredBy

Further simplify programming?

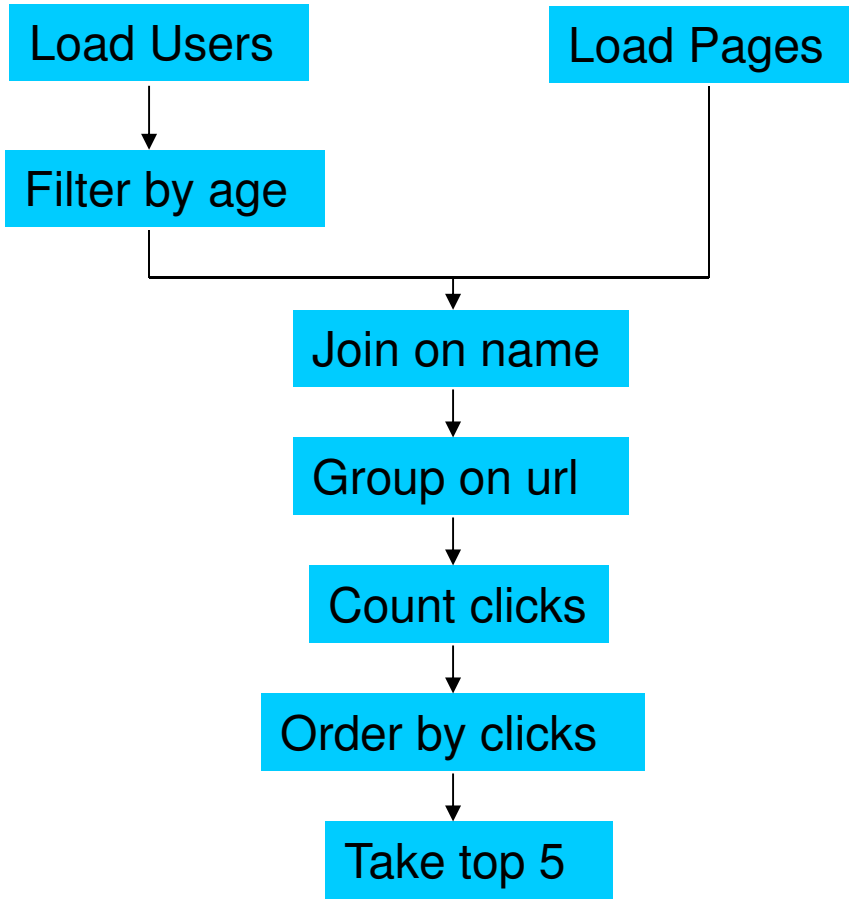
- MapReduce is powerful
  - Many algorithms can be expressed as a series of MapReduce jobs
- But it is fairly low-level
  - Must think about keys, values, partitioning, etc.
- Can we capture common “job patterns”?

Pig

- Started at Yahoo! Research
- Runs about 50% of Yahoo!’s jobs
- Features:
  - Expresses sequences of MapReduce jobs
  - Data model: nested “bags” of items
  - Provides relational (SQL) operators (JOIN, GROUP BY, etc.)
  - Easy to plug in Java functions

An example problem

- Suppose you have user data in one file, website data in another, and you need to find the top 5 most visited pages by users aged 18-25



Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

In MapReduce

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

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

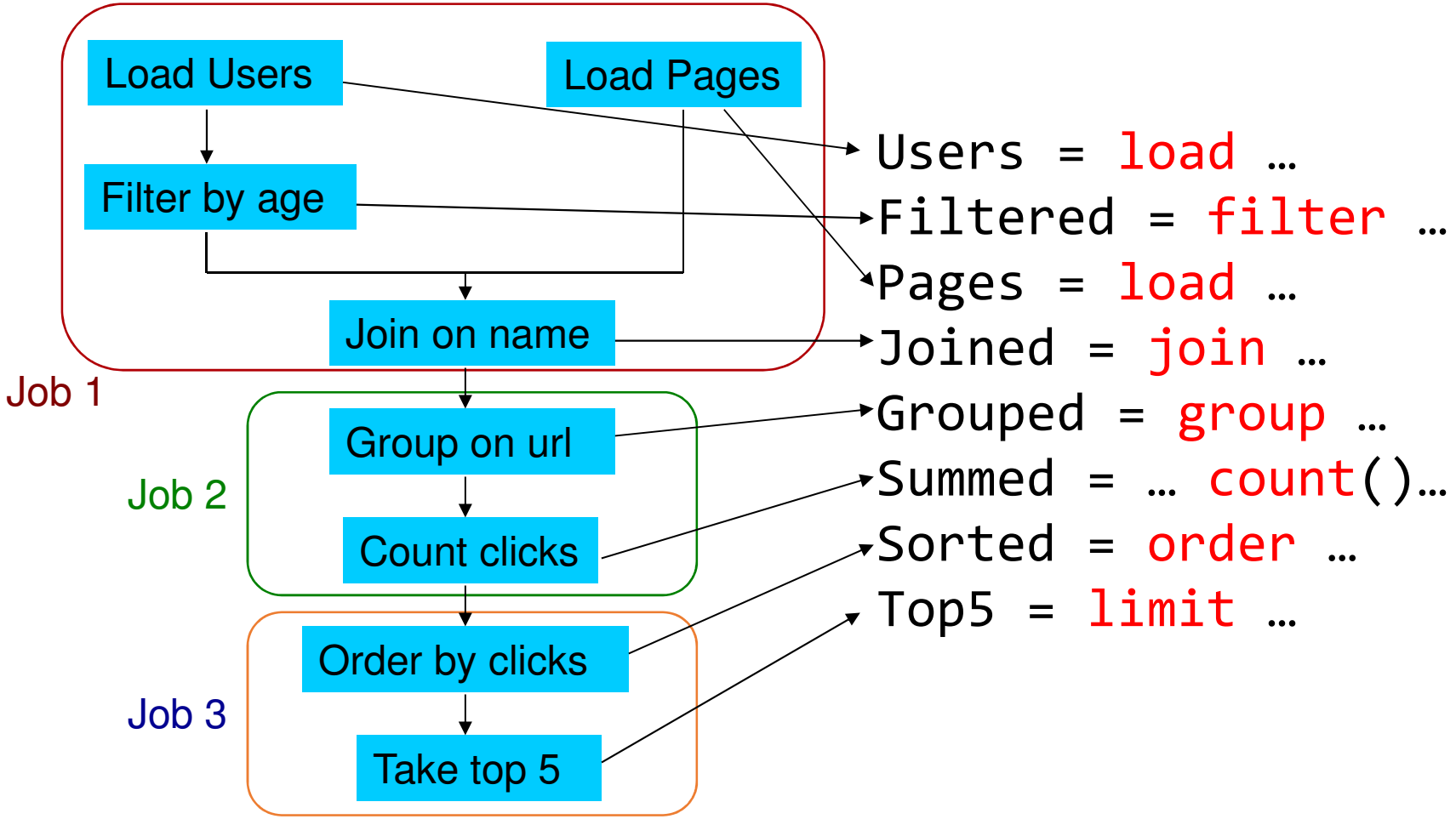
Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

In Pig Latin

```
Users    = load ‘users’ as (name, age);
Filtered = filter Users by
                  age >= 18 and age <= 25;
Pages    = load ‘pages’ as (user, url);
Joined   = join Filtered by name, Pages by user;
Grouped  = group Joined by url;
Summed   = foreach Grouped generate group,
                   count(Joined) as clicks;
Sorted   = order Summed by clicks desc;
Top5     = limit Sorted 5;

store Top5 into ‘top5sites’;
```

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

Translation to MapReduce



The components of the job naturally translate into Pig Latin

Using Pig Script -- Flight Data Analysis Example

- E.g.: totalmiles.pig: calculates the total miles flown for all flights flown in one year

```
records = LOAD '2013_subset.csv' USING PigStorage(',') AS
              (Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,\
              CRSArrTime,UniqueCarrier,FlightNum,TailNum,ActualElapsedTime,\
              CRSElapsedTime,AirTime,ArrDelay,DepDelay,Origin,Dest,\
              Distance:int,TaxiIn,TaxiOut,Cancelled,CancellationCode,\
              Diverted,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,\
              LateAircraftDelay);
milage_recs = GROUP records ALL;
tot_miles = FOREACH milage_recs GENERATE SUM(records.Distance);
STORE tot_miles INTO /user/root/totalmiles;
```

- Execute it: `pig totalmiles.pig`
- See result: `hdfs dfs -cat /user/root/totalmiles/part-r-00000` → **775009272**

D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

Summary

- MapReduce's data-parallel programming model hides complexity of distribution and fault tolerance
- Principal philosophies:
  - *Make it scale*, so you can throw hardware at problems
  - *Make it cheap*, saving hardware, programmer and administration costs (but necessitating fault tolerance)
- Pig further simplifies programming
- MapReduce is not suitable for all problems, but when it works, it may save you a lot of time…

Spark

Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud. It can access diverse data sources.

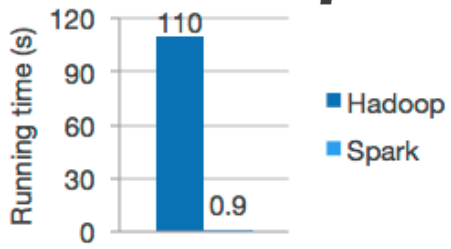
You can run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, on Mesos, or on Kubernetes. Access data in HDFS, Alluxio, Apache Cassandra, Apache HBase, Apache Hive, and hundreds of other data sources.

Apache Spark™ is a unified analytics engine for large-scale data processing.

Speed

Run workloads 100x faster.

Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.



Logistic regression in Hadoop and Spark

Ease of Use

Write applications quickly in Java, Scala, Python, R, and SQL.

Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it *interactively* from the Scala, Python, R, and SQL shells.

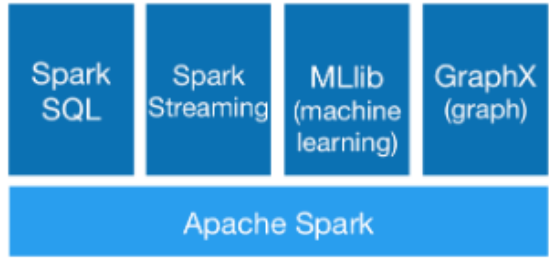
```
df = spark.read.json("logs.json")
df.where("age > 21")
  .select("name.first").show()
```

Spark's Python DataFrame API
Read JSON files with automatic schema inference

Generality

Combine SQL, streaming, and complex analytics.

Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.



Some of the slides about Spark are adopted from its author's talk at NSDI '12, https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia

Spark -- Motivation

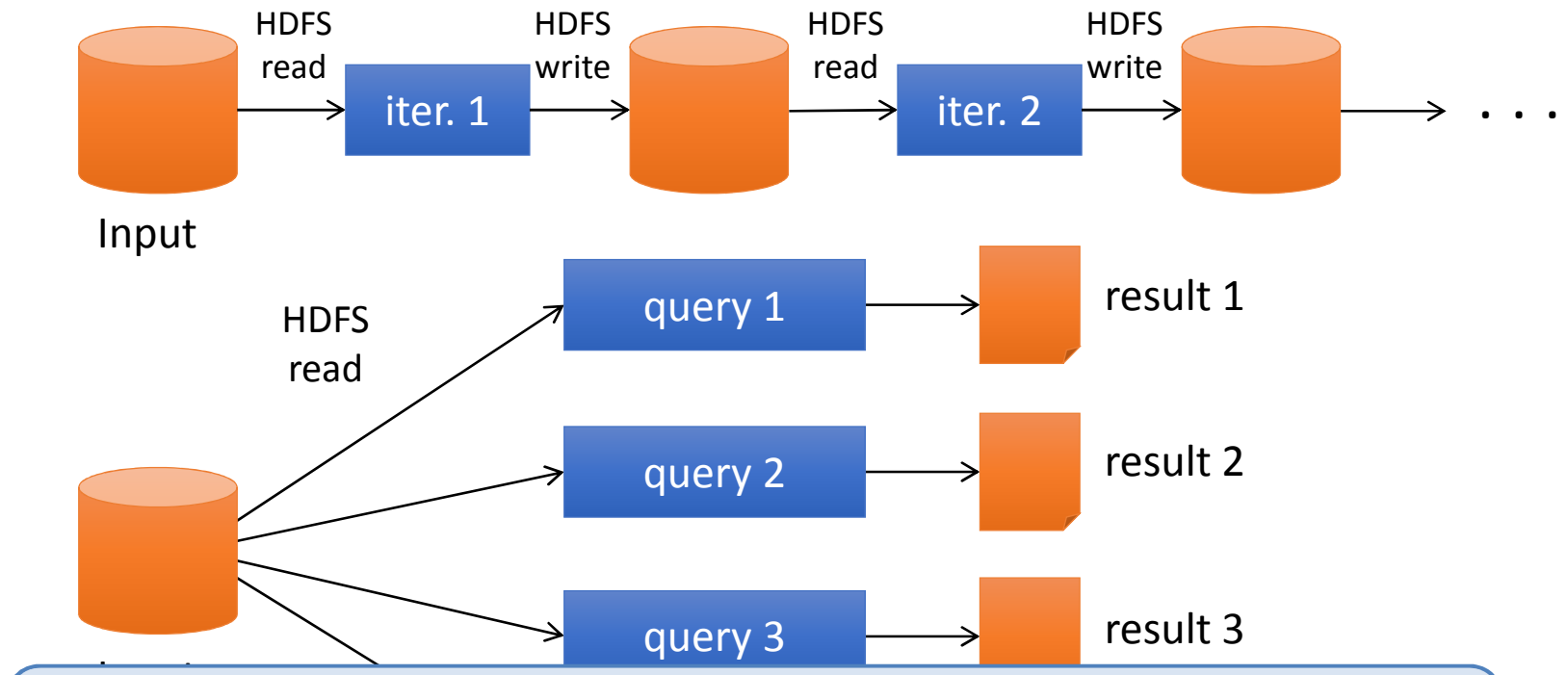
- MapReduce greatly simplified “Big Data” analysis on large, unreliable clusters

- But as soon as it got popular, users wanted more:
  - More **complex**, multi-stage applications
    - e.g. iterative machine learning, graph processing
  - More **interactive** ad-hoc queries
  - More **low-latency** online processing

- Spark’s response: *specialized* frameworks for some of these applications

Spark -- Motivation

- Complex apps and interactive queries both need one thing that MapReduce lacks: efficient primitives for **data sharing**
- In MapReduce, the only way to share data across jobs is stable storage → **slow**!



Slow due to replication and disk I/O, but necessary for fault tolerance

Spark -- Motivation

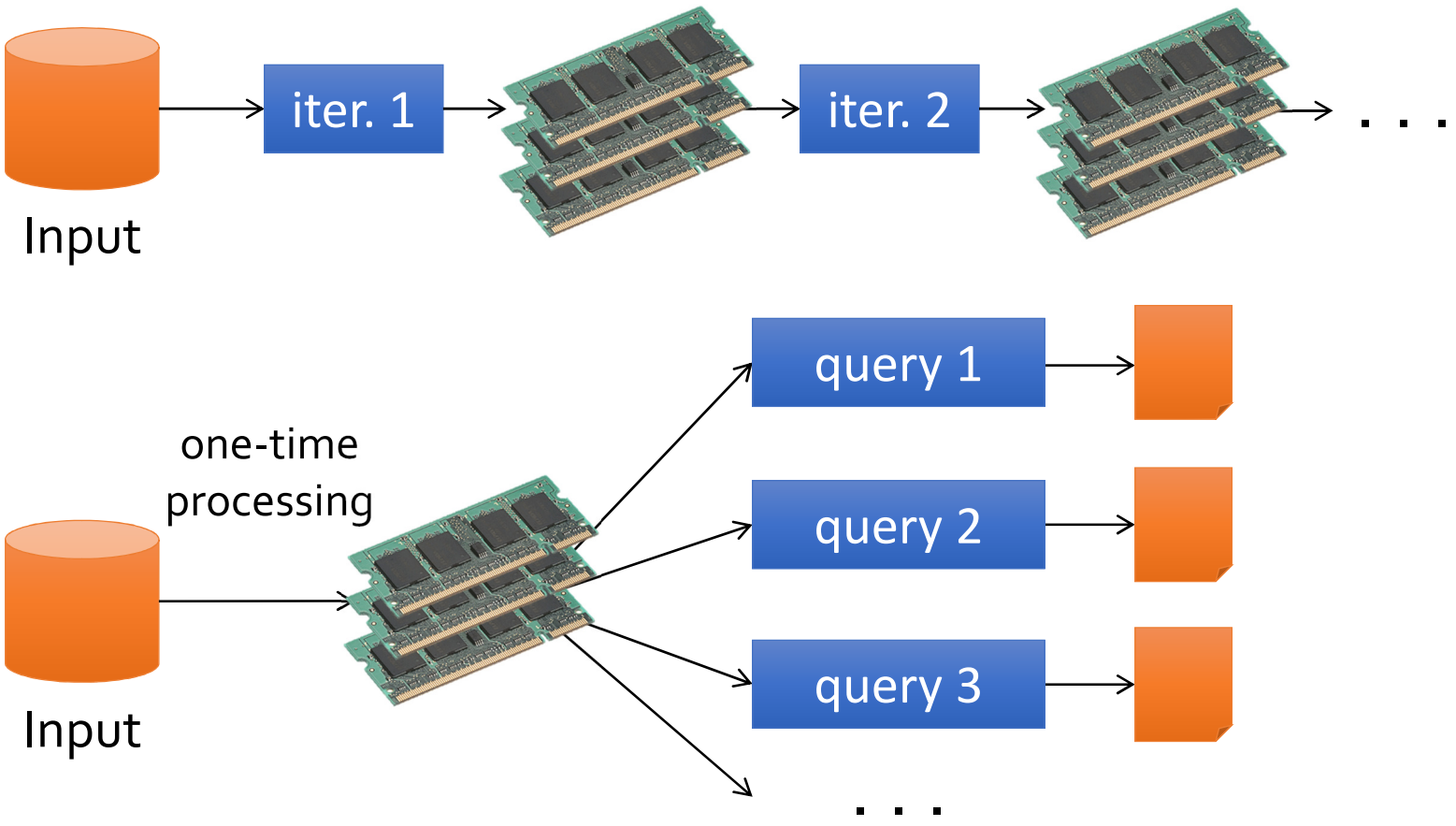
Applications that ***reuse*** intermediate results across multiple computations. Mainly, two types of applications:

1. Iterative machine learning & graph processing algorithms including: PageRank, K-means clustering, and logistic regression

2. Interactive data mining tools, where a user runs multiple ad hoc queries on the same subset of the data

With current frameworks, applications reload data from stable storage on each query

Goal: In-Memory Data Sharing



10-100× faster than network/disk, but how to get FT?

Resilient Distributed Datasets (RDDs)

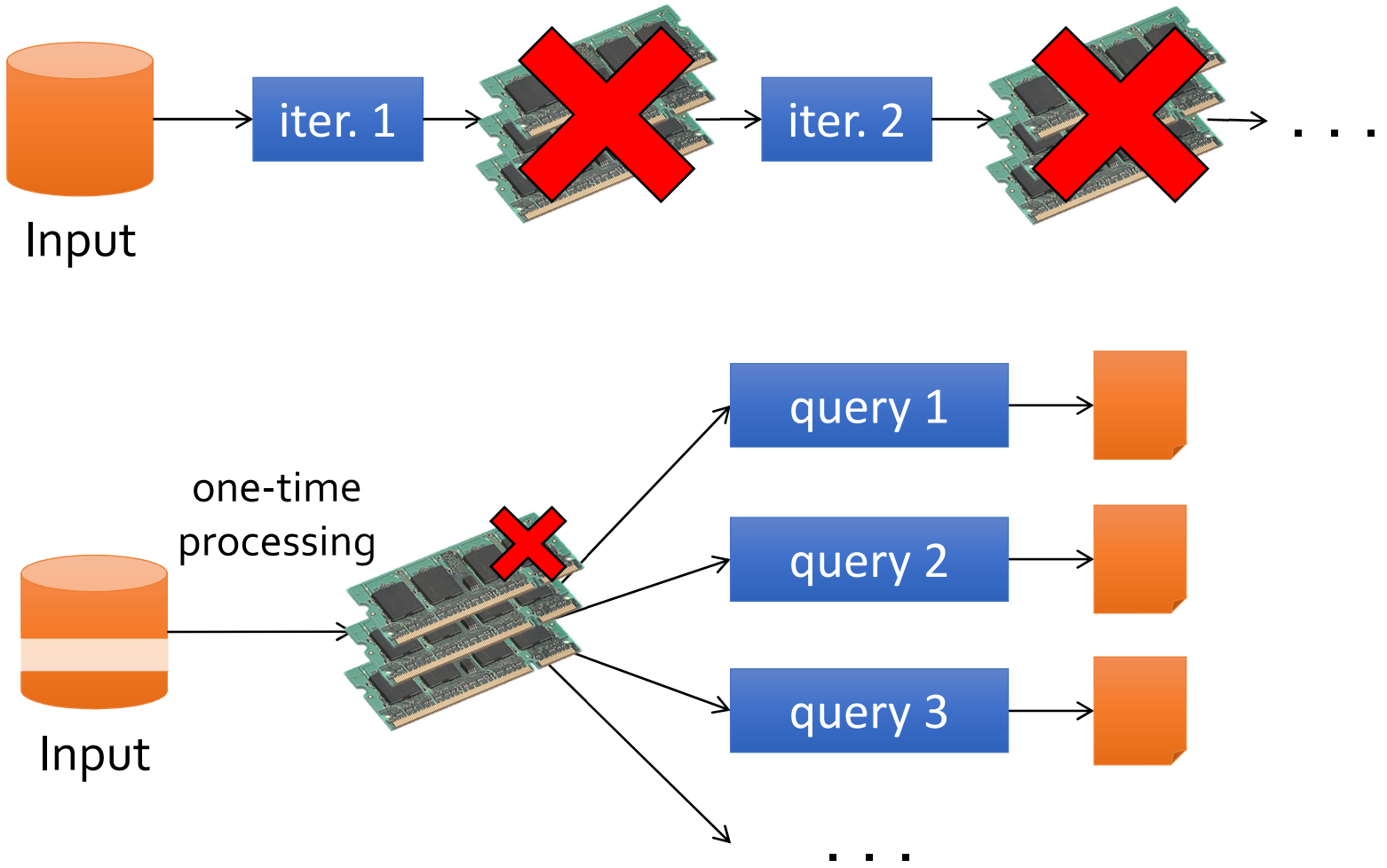
- Allow apps to keep working sets in memory for efficient reuse
- Retain the attractive properties of MapReduce
  - Fault tolerance, data locality, scalability
- Support a wide range of applications

Resilient Distributed Datasets (RDDs)

- Restricted form of distributed shared memory
  - Immutable, partitioned collections of records
  - Manipulated through a diverse set of transformations (*map*, *filter*, *join*, etc.)

- Efficient fault recovery using *lineage*
  - Remember the series of transformations that built an RDD (its *lineage*)
  - Log one operation to apply to many elements
  - Re-compute lost partitions on failure
  - No cost if nothing fails

RDD Recovery



Generality of RDDs

- Despite their restrictions, RDDs can express surprisingly many parallel algorithms
  - These naturally *apply the same operation to many items*

- Unify many current programming models
  - *Data flow models:* MapReduce, Dryad, SQL, …
  - *Specialized models* for iterative apps: BSP (Pregel), iterative MapReduce (Haloop), bulk incremental, …

- Support *new apps* that these models don't

Programming Model

- Resilient distributed datasets (RDDs)
  - Immutable, partitioned collections of objects
  - Created through parallel *transformations* (map, filter, groupBy, join, …) on data in stable storage
  - Can be *cached* for efficient reuse

- *Actions* on RDDs
  - Count, reduce, collect, save, …

Spark Programming Interface

- DryadLINQ-like API in the Scala language
- Usable interactively from Scala interpreter
- Provides:
  - Resilient distributed datasets (RDDs)
  - Operations on RDDs: *transformations* (build new RDDs), *actions* (compute and output results)
  - Control of each RDD's *partitioning* (layout across nodes) and *persistence* (storage in RAM, on disk, etc.)

Spark Operations

Transformations (define a new RDD)	map filter sample groupByKey reduceByKey sortByKey	flatMap union join cogroup cross mapValues
Actions (return a result to driver program)	collect reduce count save lookupKey	

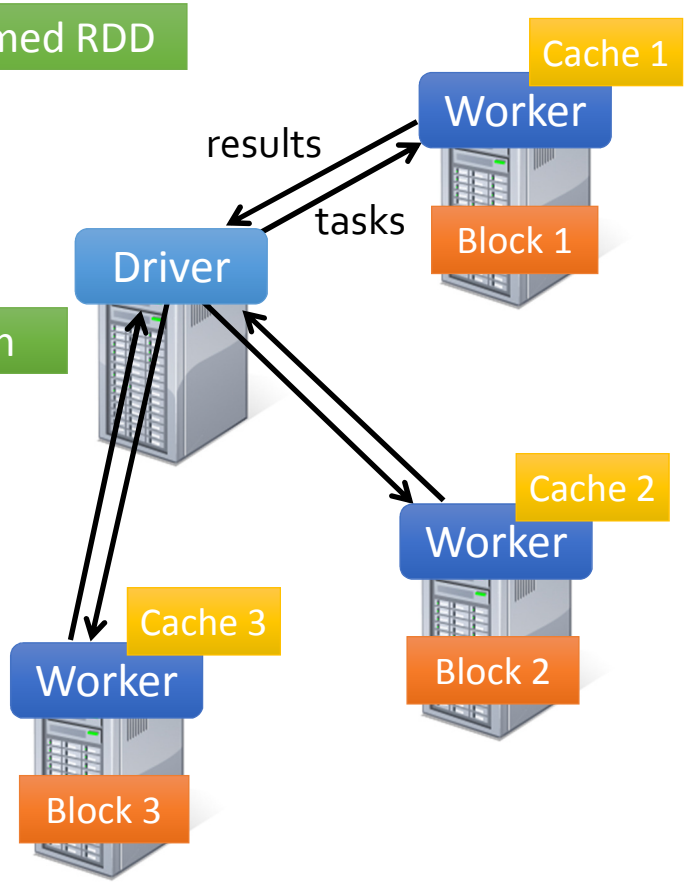
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
```

Transformed RDD

Action



Result: full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)

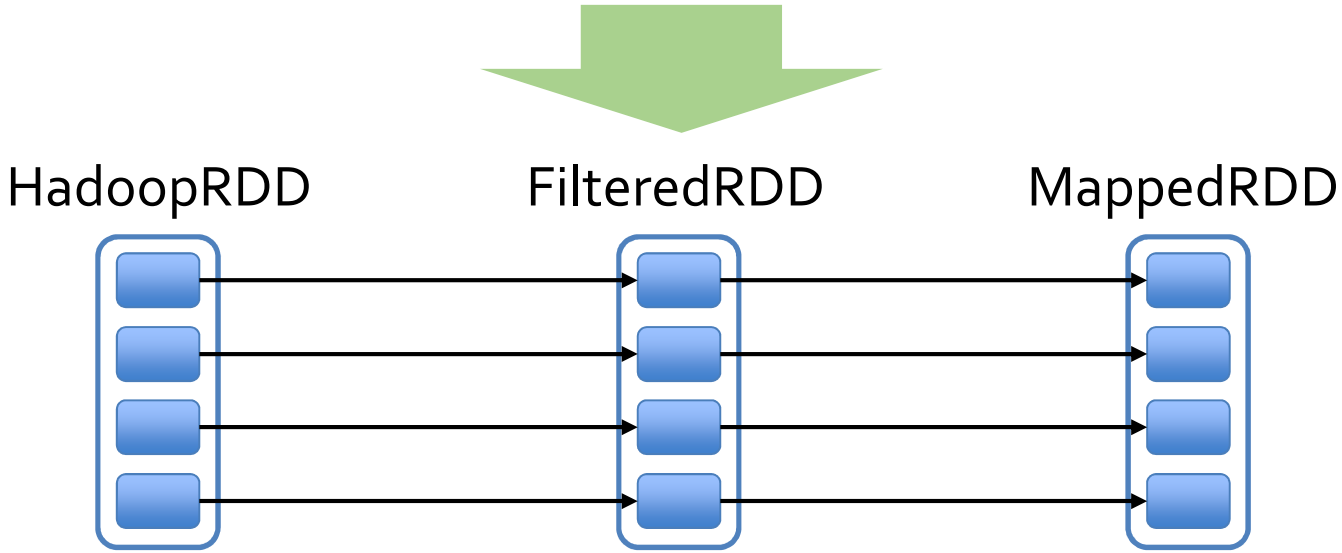
Result: scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)

RDD Fault Tolerance

- RDDs track the graph of transformations that built them (their *lineage*) to rebuild lost data

E.g.,:

```
messages = textFile(...).filter(_.contains("error"))
                        .map(_.split('\t')(2))
```

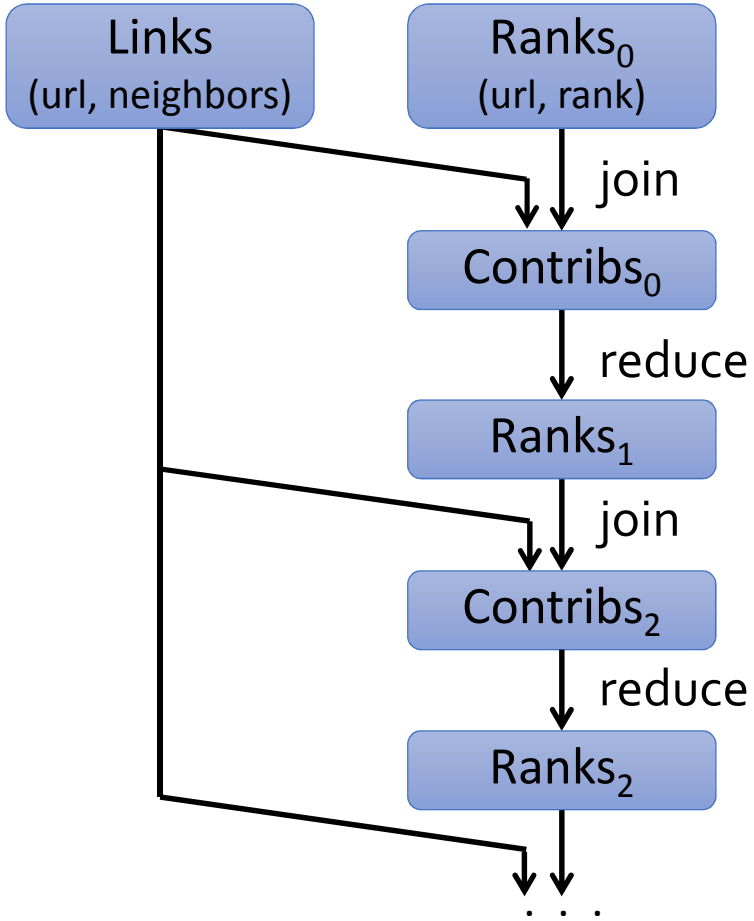


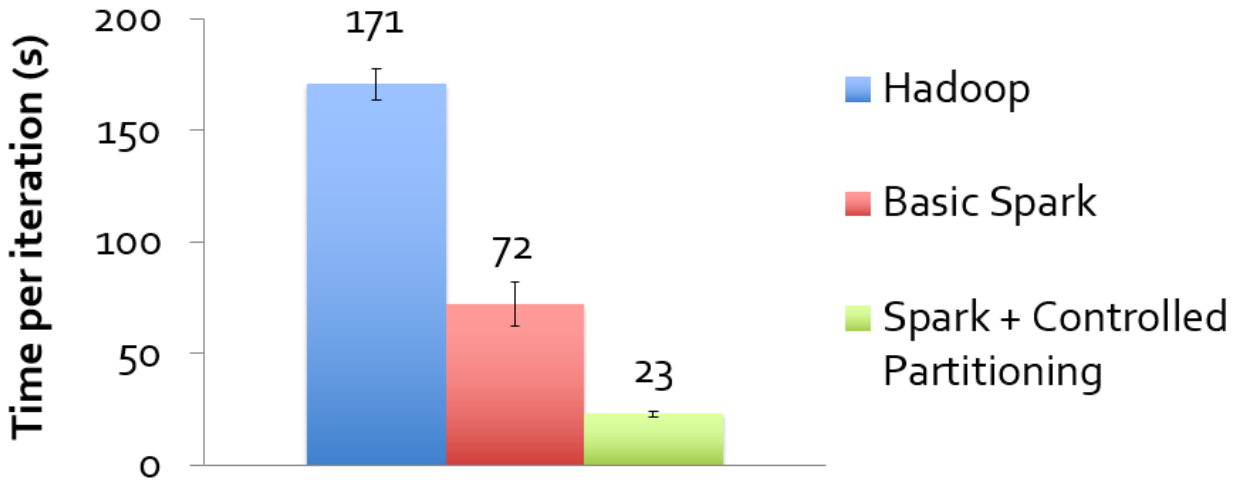
Example: PageRank

1. Start each page with a rank of 1
2. On each iteration, update each page's rank to
   $\Sigma_{i \in neighbors} rank_i / |neighbors_i|$

```
links = // RDD of (url, neighbors) pairs
ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  ranks = links.join(ranks).flatMap {
    (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }.reduceByKey(_ + _)
}
```

Optimizing Placement

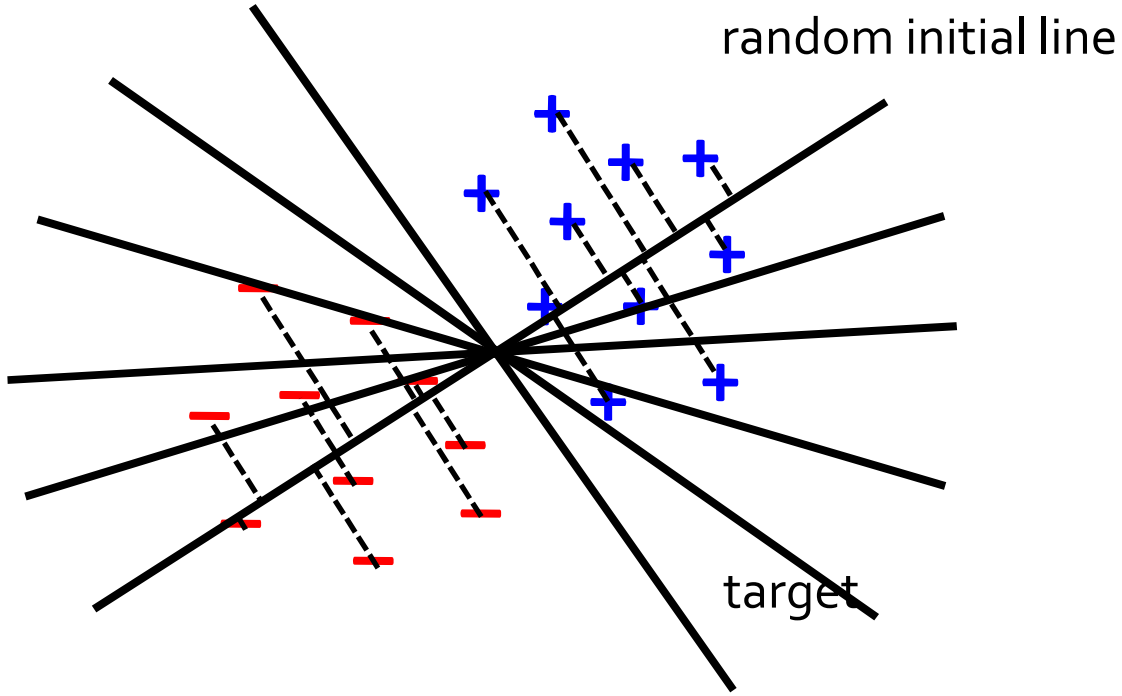




- `links` & `ranks` repeatedly joined
- Can *co-partition* them (e.g., hash both on URL) to avoid shuffles
- Can also use app knowledge, e.g., hash on DNS name
- `links = links.partitionBy(new URLPartitioner())`

Example: Logistic Regression

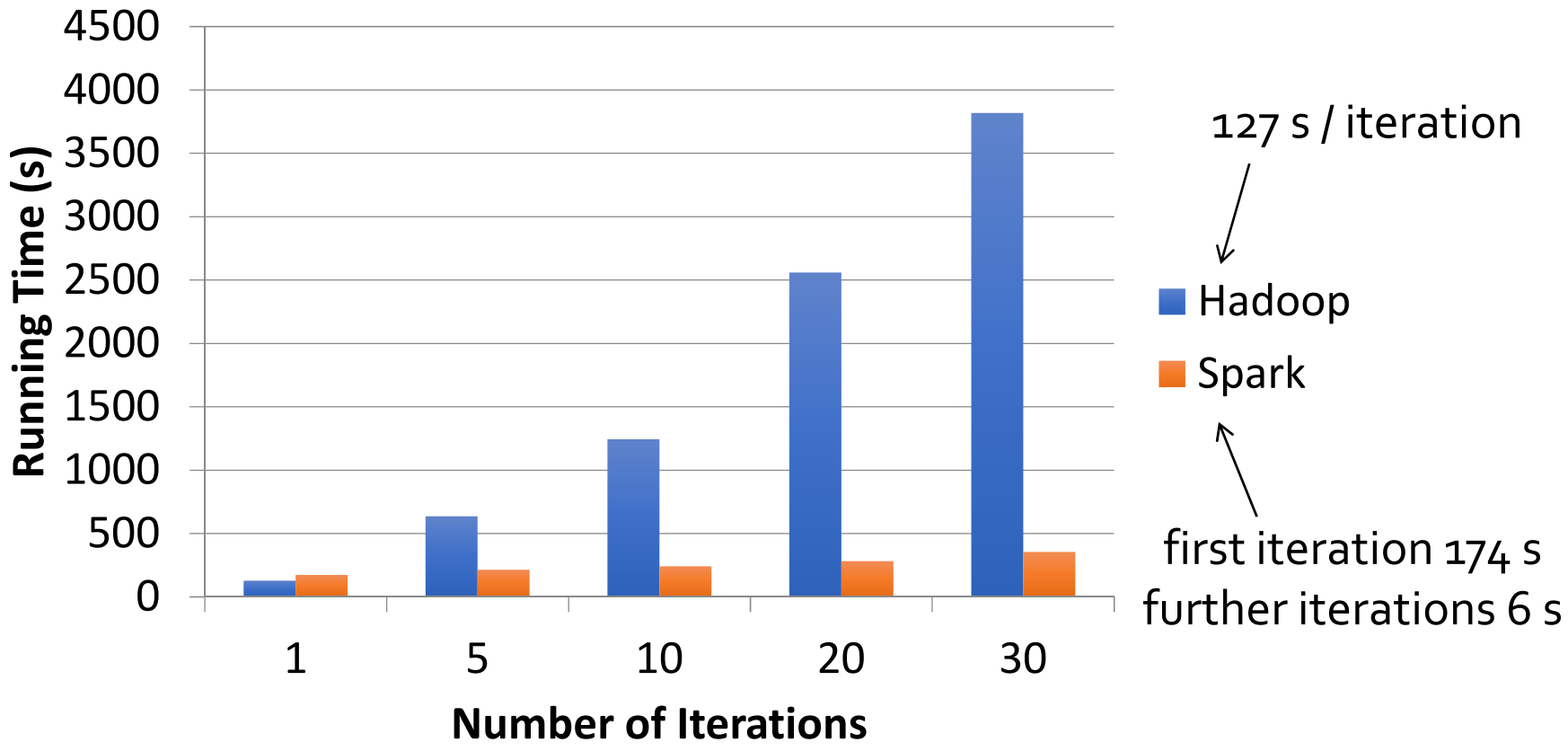
- Goal: find best line separating two sets of points



Logistics Regression Code

```
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

Logistic Regression Performance



Spark Applications

- In-memory data mining on Hive data (Conviva)
- Predictive analytics (Quantifind)
- City traffic prediction (Mobile Millennium)
- Twitter spam classification (Monarch)
- Collaborative filtering via matrix factorization
- …

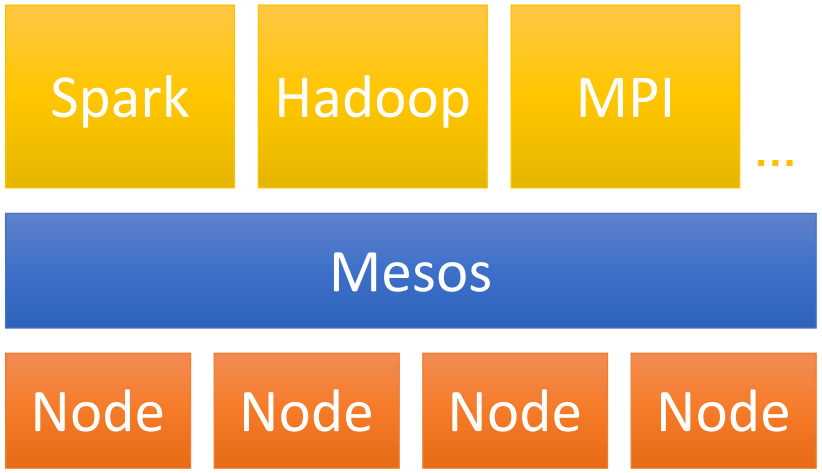
Framework built on Spark

- Pregel on Spark (Bagel)
  - Google message passing model for graph computation
  - 200 lines of code

- Hive on Spark (Shark)
  - 3000 lines of code
  - Compatible with Apache Hive
  - ML operators in Scala

Implementation

- Runs on Apache Mesos to share resources with Hadoop & other apps
- Can read from any Hadoop input source (e.g. HDFS)
- No changes to Scala compiler

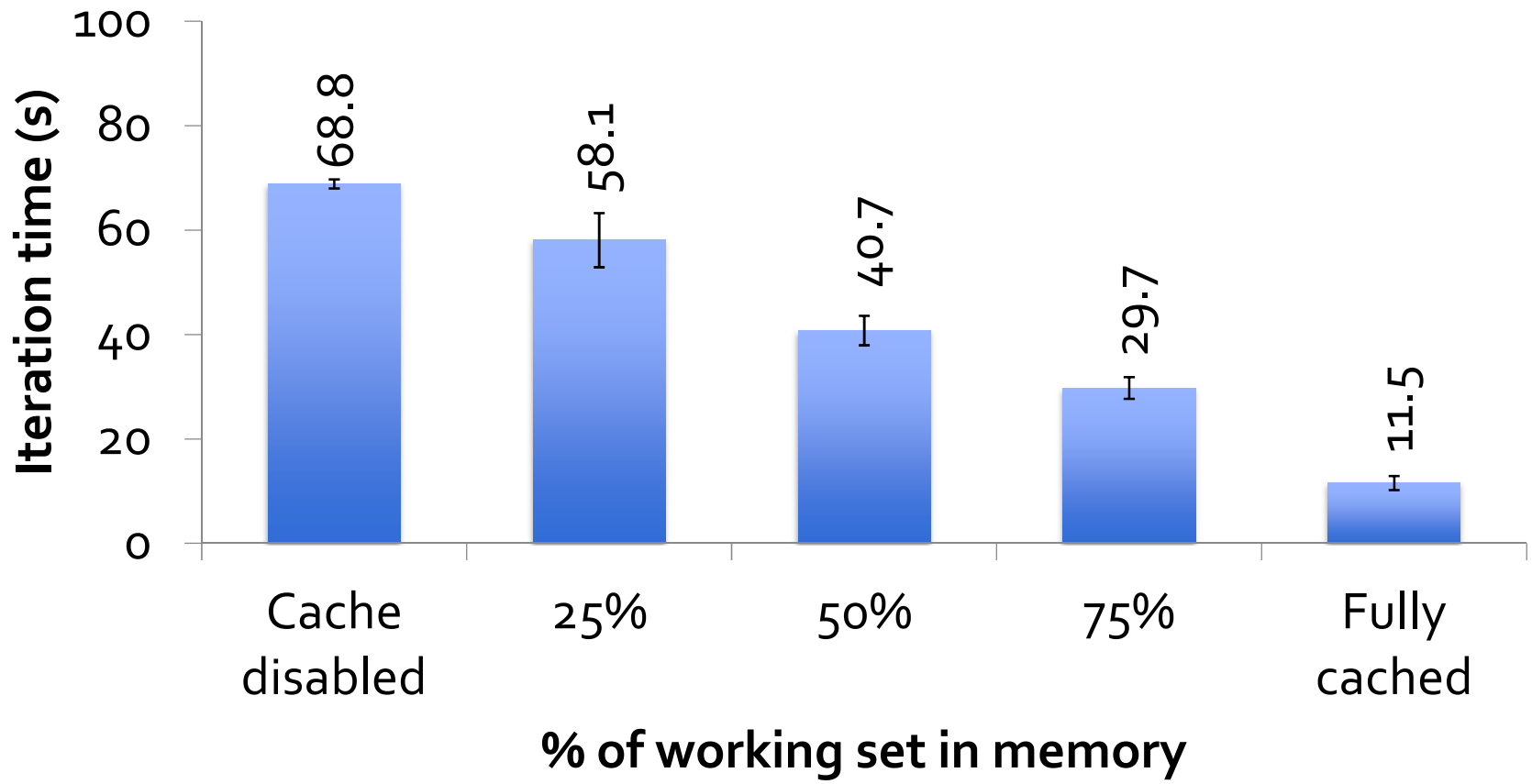


Interactive Spark

- Modified Scala interpreter to allow Spark to be used interactively from the command line

- Required two changes:
  - Modified wrapper code generation so that each line typed has references to objects for its dependencies
  - Distribute generated classes over the network

Behavior with Not enough RAM



Note – If the Distributed memory (RAM) is not sufficient to store intermediate results, then it will store those results on the disk.

Summary

- RDDs offer a simple and efficient programming model for a broad range of applications
- Leverage the coarse-grained nature of many parallel algorithms for low-overhead recovery
- Spark: http://spark-project.org

Thanks ! ☺

Questions ?