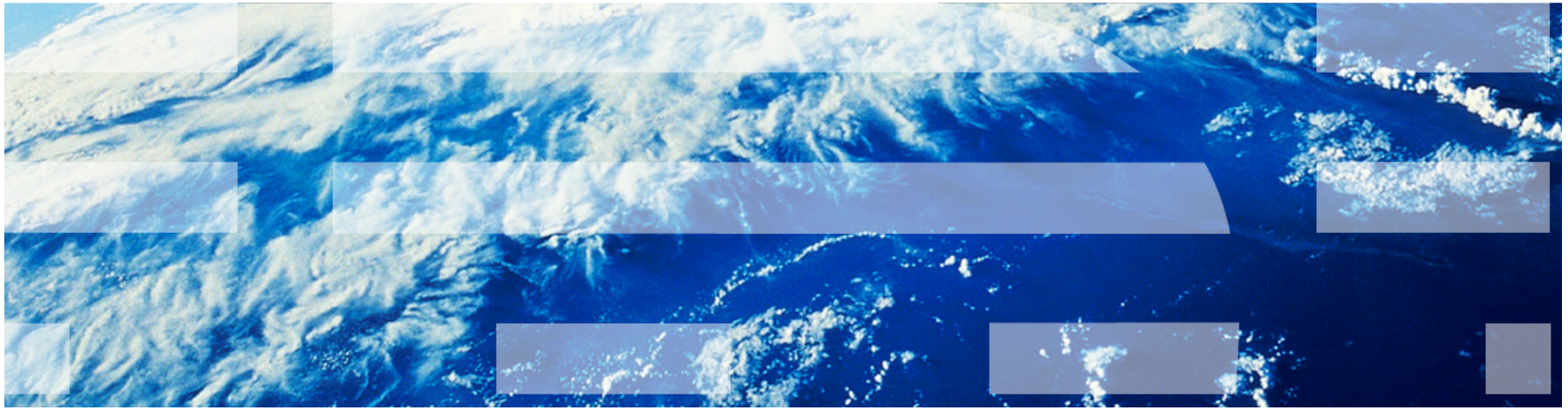
E6893 Big Data Analytics Lecture 2:

Big Data Analytics Platforms

Ching-Yung Lin, Ph.D.

Adjunct Professor, Dept. of Electrical Engineering and Computer Science

IBM Distinguished Researcher and Chief Scientist, Graph Computing



September 17th, 2015

Course information -- TAs

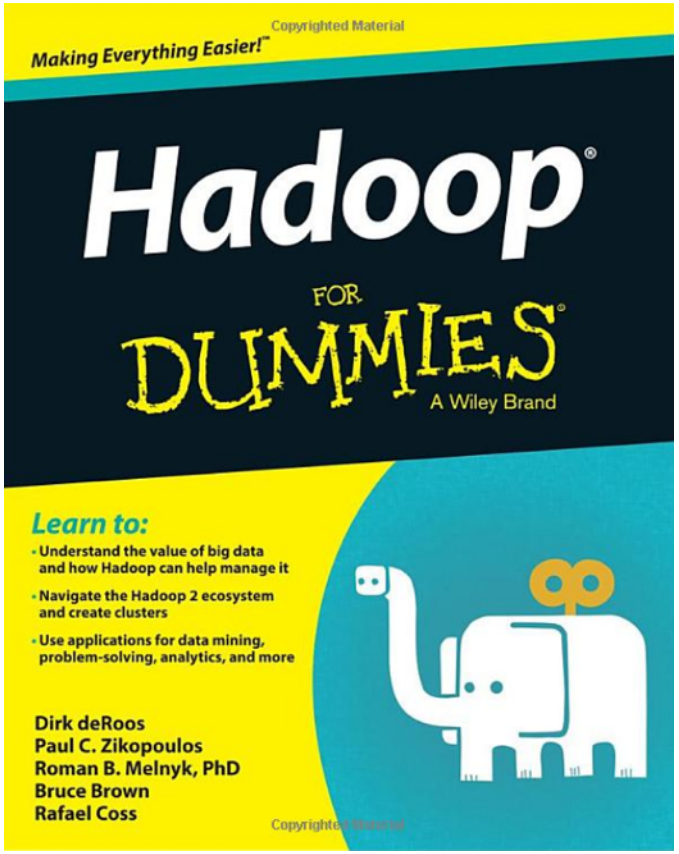
- 10 Teaching Assistants (Course Assistants/ Instructional Assistants):
  - Ghazel Fazelnia (EE)
  - Rishina Tah (CS)
  - Abhyday Polineni (CS)
  - Junkai Yan (EE)
  - Naman Jain (CS)
  - Siyuan Zhang (EE)
  - Rama Kompella (EE)
  - Liqun Chen (EE)
  - Yongchen Jiang (EE)
  - Tian Han (EE)

Students shall be divided into groups based on interest

- **Goal:** Align interest domain into groups in order to focus on use scenarios, datasets, requirements to create open source Big Data Analytics toolkits.

- **And also other fields that are not on the list:** Education, Social Science, etc.
- **Selection:** An online website will be opened to let all students (on-campus & CVN) submit preferences and description of personal education/work background towards the domain.
- **TAs** will be assigned to lead 10 groups. Some groups may have multiple fields. Some fields may be multiple groups.

Reading Reference for Lecture 2 & 3



Remind -- 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.

The project includes these modules:

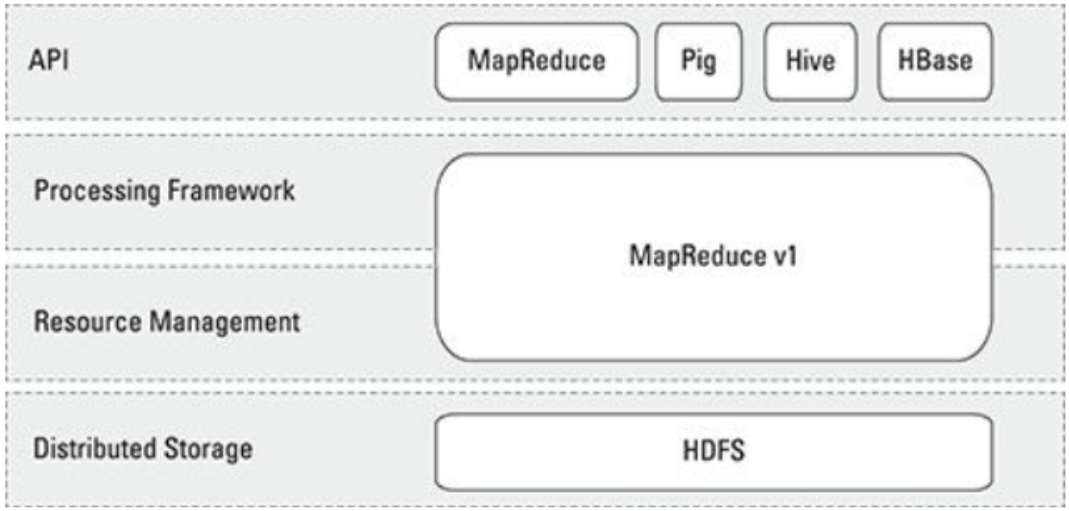
- **Hadoop Common**: The common utilities that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
- **Hadoop YARN**: A framework for job scheduling and cluster resource management.
- **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

http://hadoop.apache.org

Remind -- 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.

Four distinctive layers of Hadoop



Distributed storage: The Hadoop Distributed File System (HDFS) is the storage layer where the data, interim results, and final result sets are stored.

Resource management: In addition to disk space, all slave nodes in the Hadoop cluster have CPU cycles, RAM, and network bandwidth. A system such as Hadoop needs to be able to parcel out these resources so that multiple applications and users can share the cluster in predictable and tunable ways. This job is done by the JobTracker daemon.

Processing framework: The MapReduce process flow defines the execution of all applications in Hadoop 1. As we saw in Chapter 6, this begins with the map phase; continues with aggregation with shuffle, sort, or merge; and ends with the reduce phase. In Hadoop 1, this is also managed by the JobTracker daemon, with local execution being managed by TaskTracker daemons running on the slave nodes.

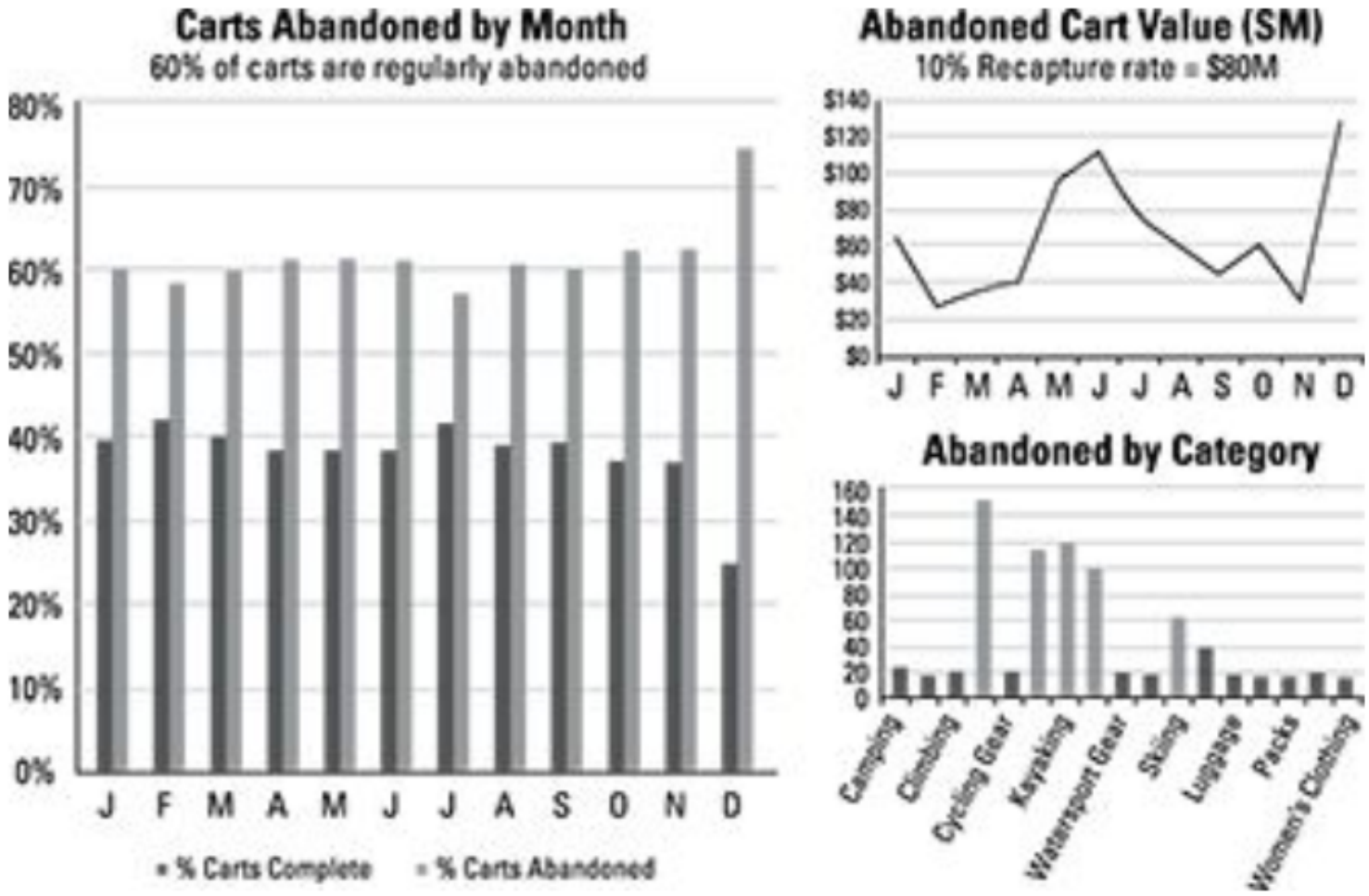
Application Programming Interface (API): Applications developed for Hadoop 1 needed to be coded using the MapReduce API. In Hadoop 1, the Hive and Pig projects provide programmers with easier interfaces for writing Hadoop applications, and underneath the hood, their code compiles down to MapReduce.

Common 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

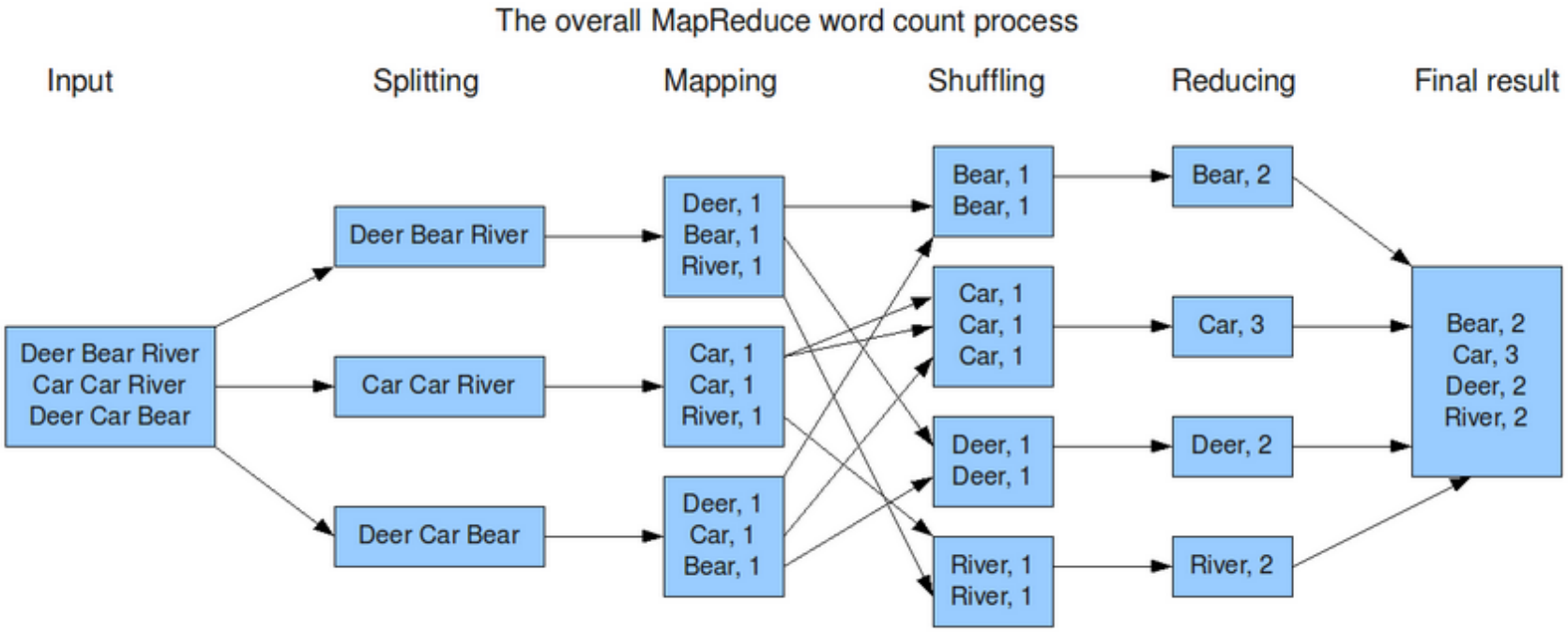
D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

Example: Business Value of Log Analysis – "Struggle Detection"



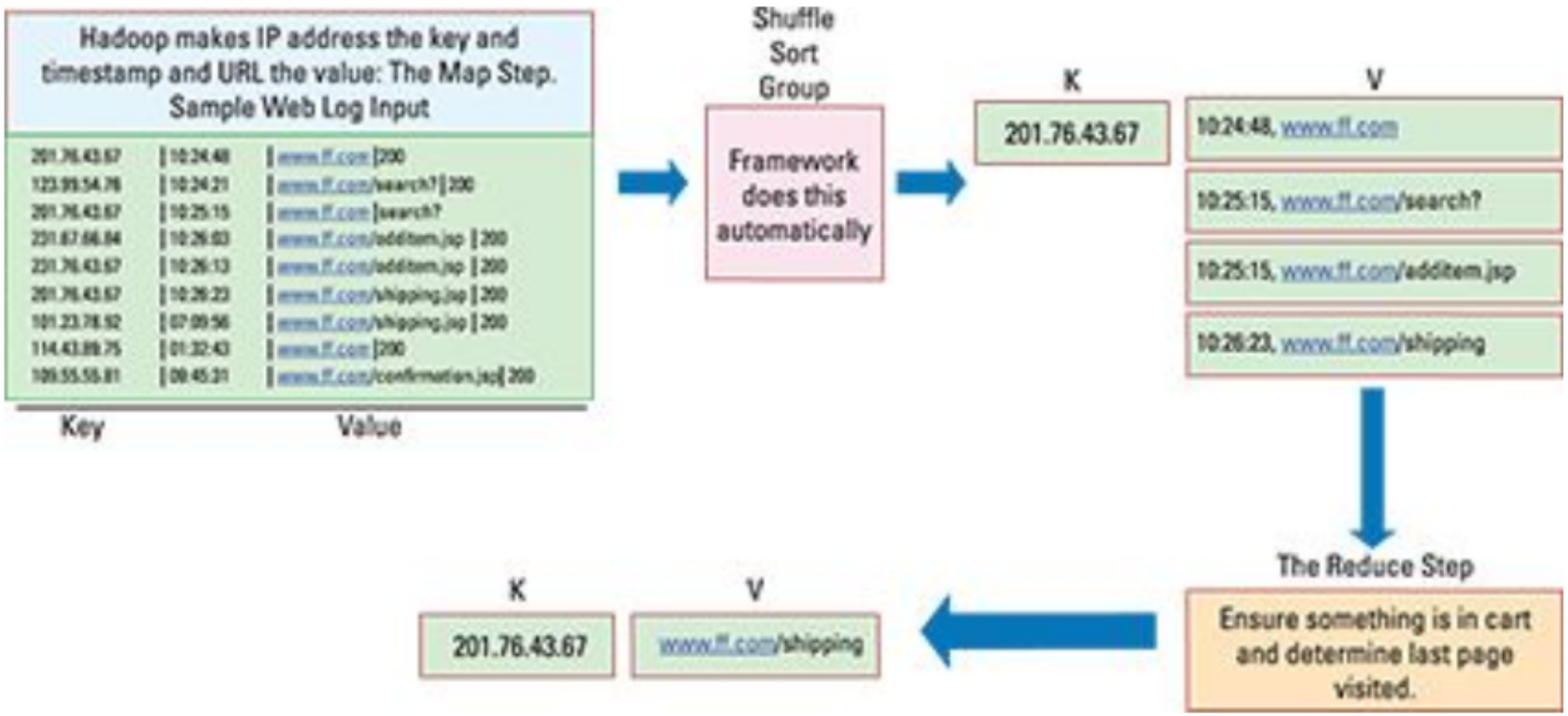
D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

Remind -- MapReduce example



http://www.alex-hanna.com

MapReduce Process on User Behavior via Log Analysis



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

Setting Up the Hadoop Environment

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

Setting Up the Hadoop Environment – Pseudo-distributed mode

Columbia University

Configuration

Use the following:

conf/core-site.xml:

```
<configuration>
     <property>
         <name>fs.defaultFS</name>
         <value>hdfs://localhost:9000</value>
     </property>
</configuration>
```

conf/hdfs-site.xml:

```
<configuration>
     <property>
         <name>dfs.replication</name>
         <value>1</value>
     </property>
</configuration>
```

conf/mapred-site.xml:

```
<configuration>
     <property>
         <name>mapred.job.tracker</name>
         <value>localhost:9001</value>
     </property>
</configuration>
```

Setup passphraseless ssh

Now 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 dsa -P '' -f ~/.ssh/id_dsa
$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

http://hadoop.apache.org/docs/stable2/hadoop-project-dist/hadoop-common/SingleNodeSetup.html

Setting Up the Hadoop Environment – Pseudo-distributed mode

Columbia University

Execution

Format a new distributed-filesystem:

```
$ bin/hadoop namenode -format
```

Start the hadoop daemons:

```
$ bin/start-all.sh
```

The hadoop daemon log output is written to the `$HADOOP_LOG_DIR` directory (defaults to `$HADOOP_PREFIX/logs`).

Browse the web interface for the NameNode and the JobTracker; by default they are available at:

- NameNode - `http://localhost:50070/`
- JobTracker - `http://localhost:50030/`

Copy the input files into the distributed filesystem:

```
$ bin/hadoop fs -put conf input
```

Run some of the examples provided:

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

Examine the output files:

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

```
$ bin/hadoop fs -get output output
$ cat output/*
```

or

View the output files on the distributed filesystem:

```
$ bin/hadoop fs -cat output/*
```

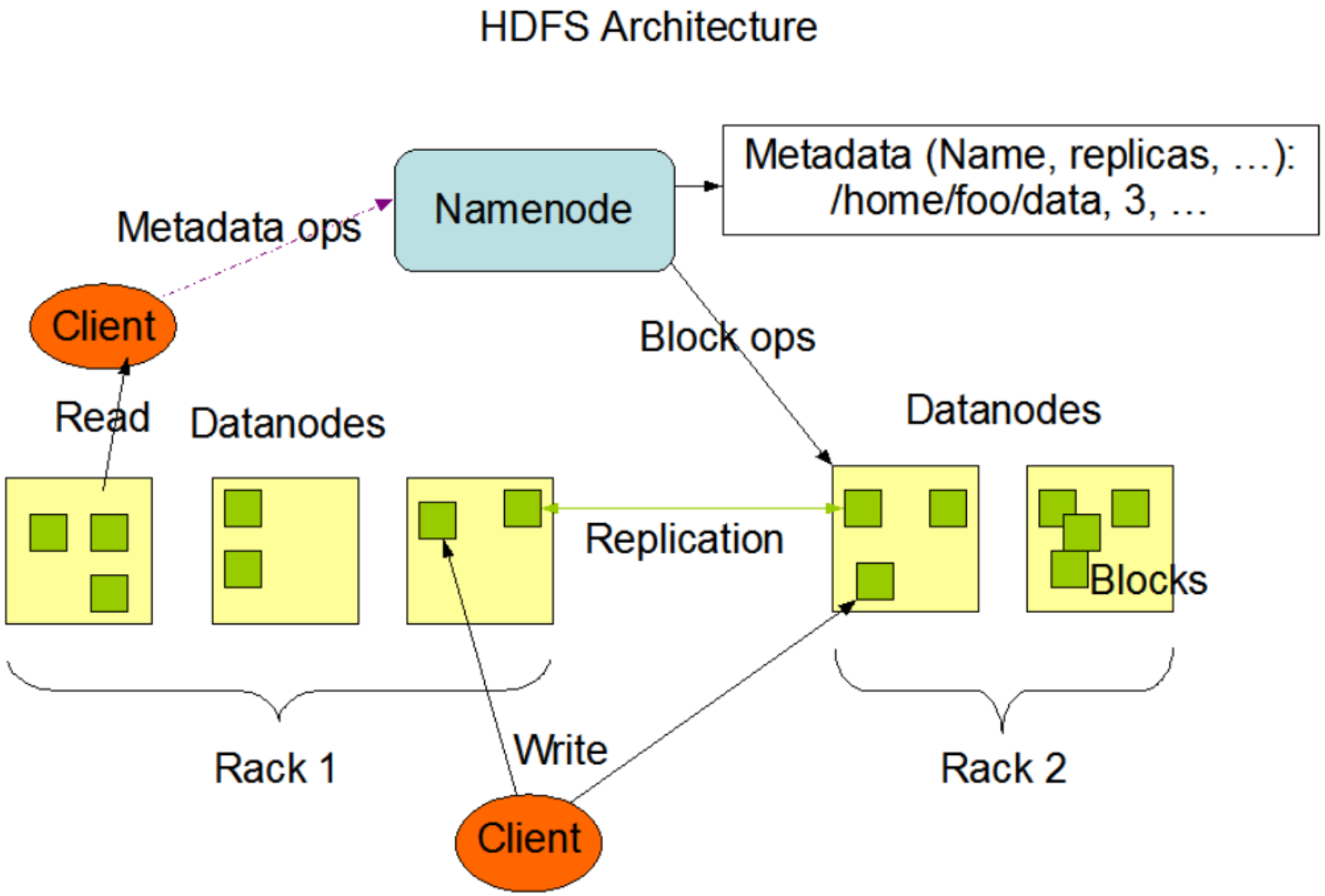
Data Storage Operations on HDFS

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

- Write One, 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

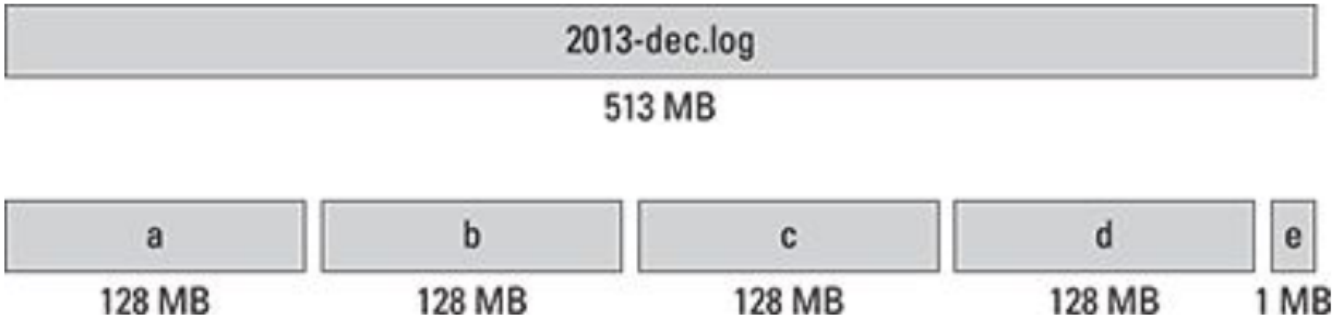
Remind -- Hadoop Distributed File System (HDFS)



http://hortonworks.com/hadoop/hdfs/

HDFS blocks

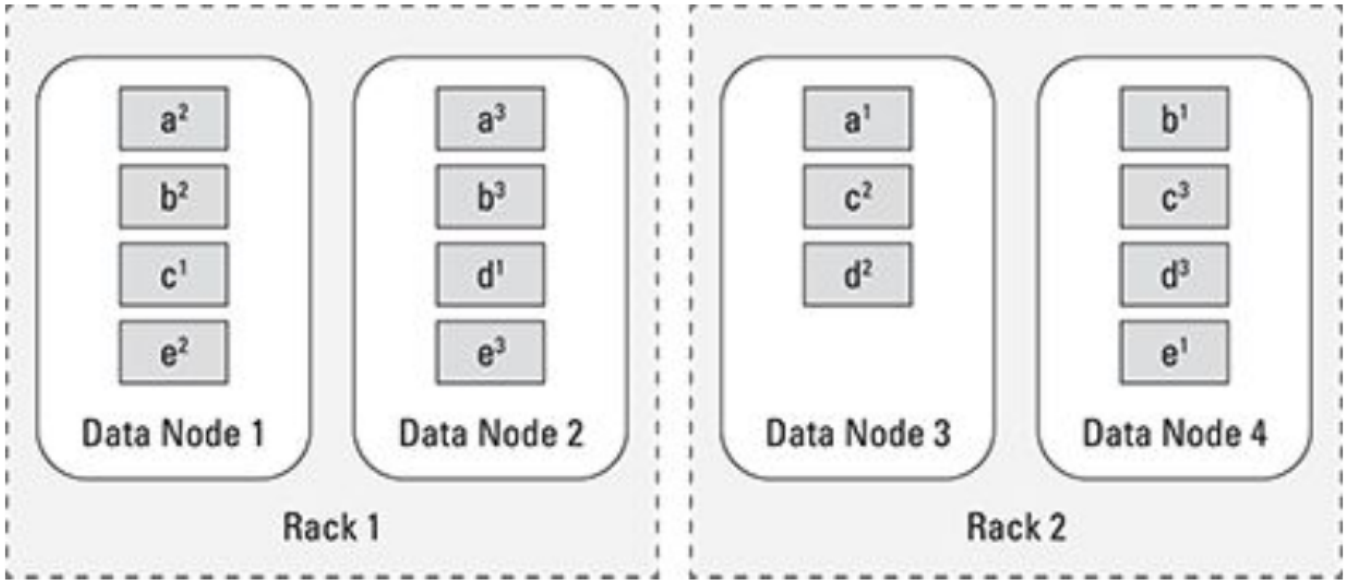
- File is divided into blocks (default: 64MB) and duplicated in multiple places (default: 3)



- 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.

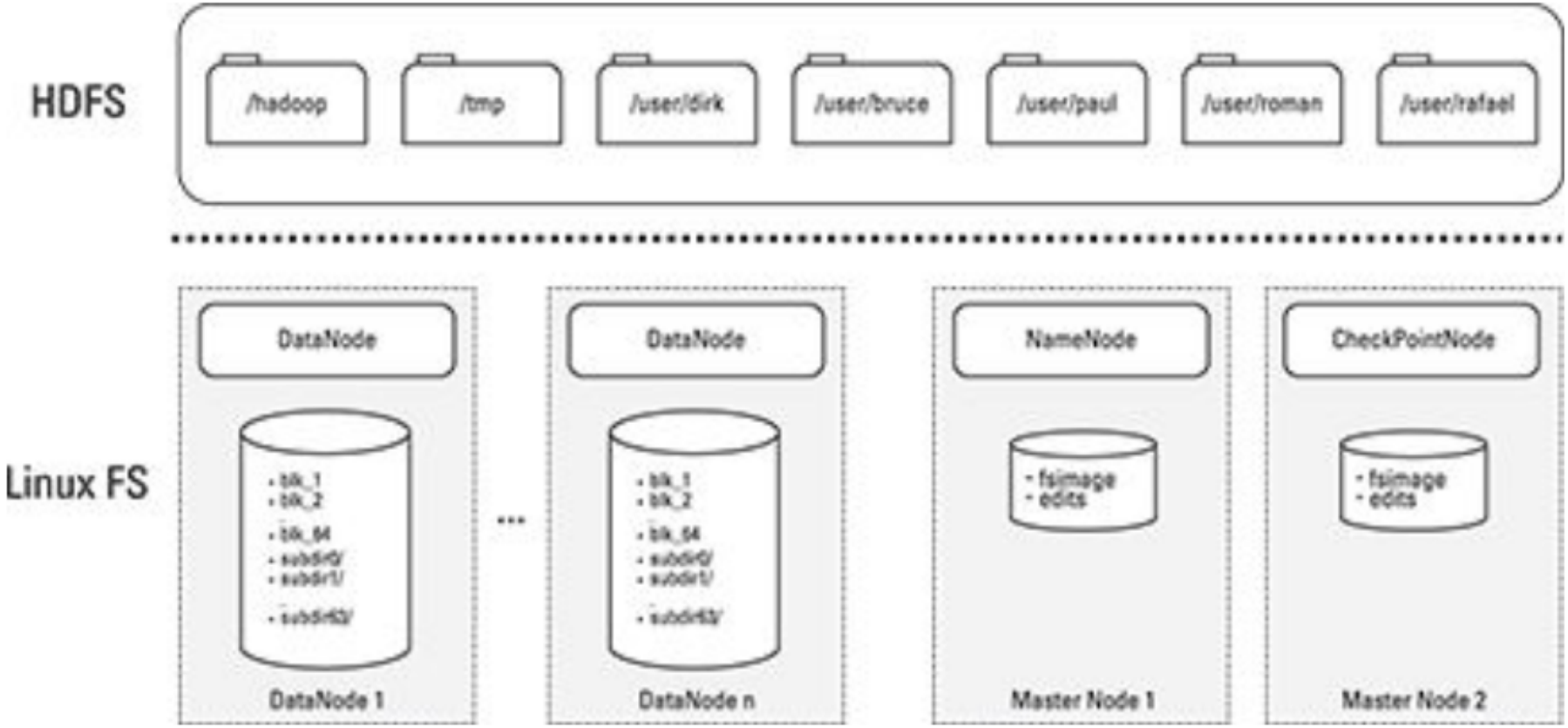
HDFS blocks

- Replication patterns of data blocks in HDFS.

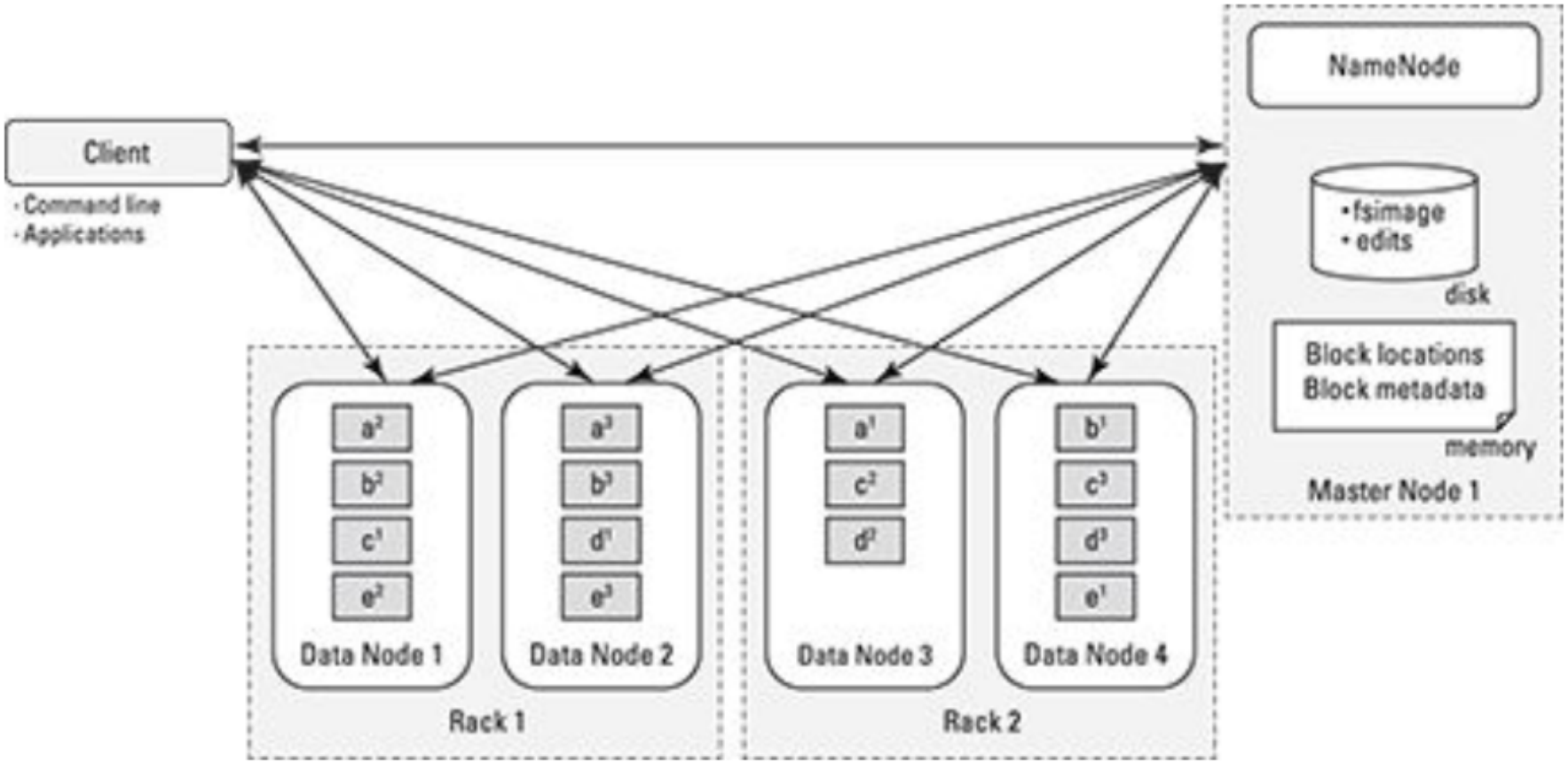


- When HDFS stores the replicas of the original blocks across the Hadoop cluster, it tries to ensure that the block replicas are stored in different failure points.

HDFS is a User-Space-Level file system

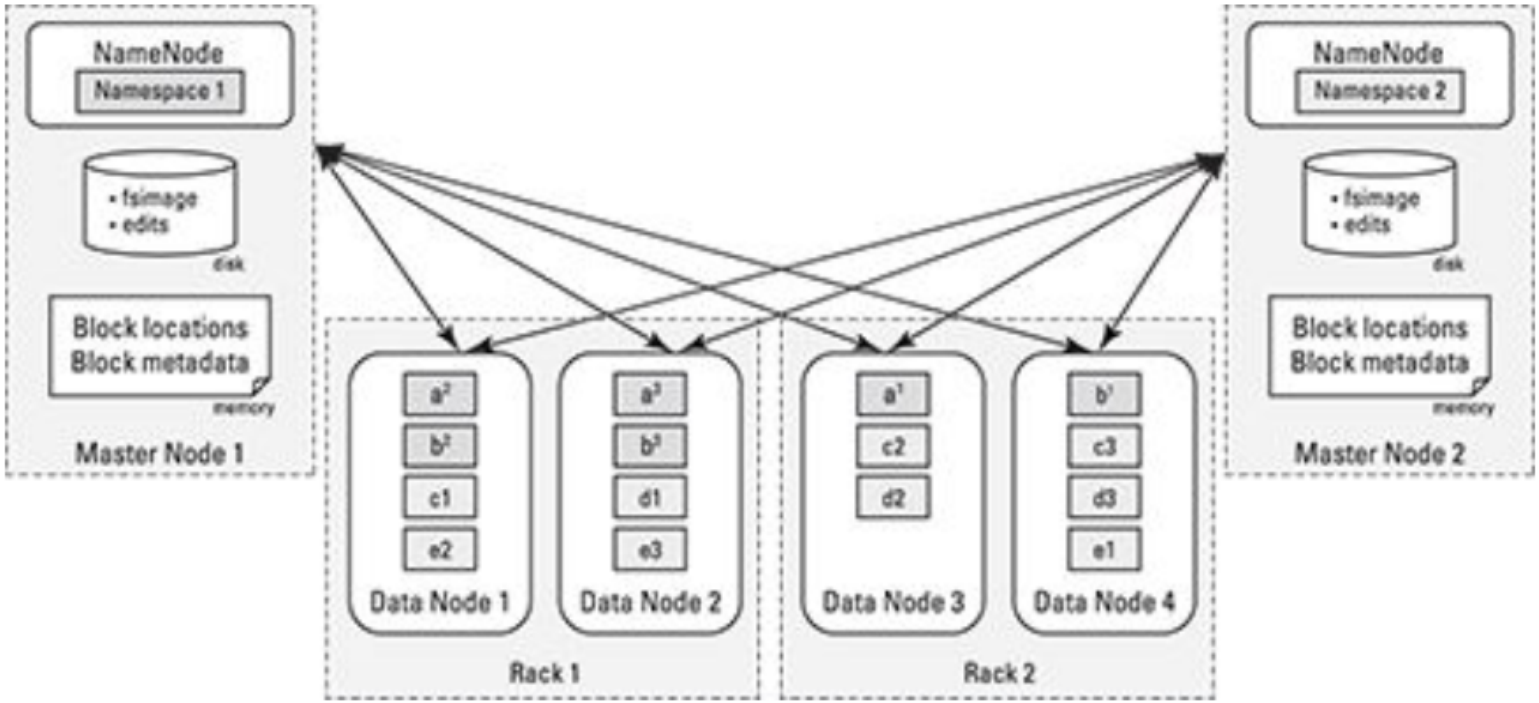


Interaction between HDFS components



HDFS Federation

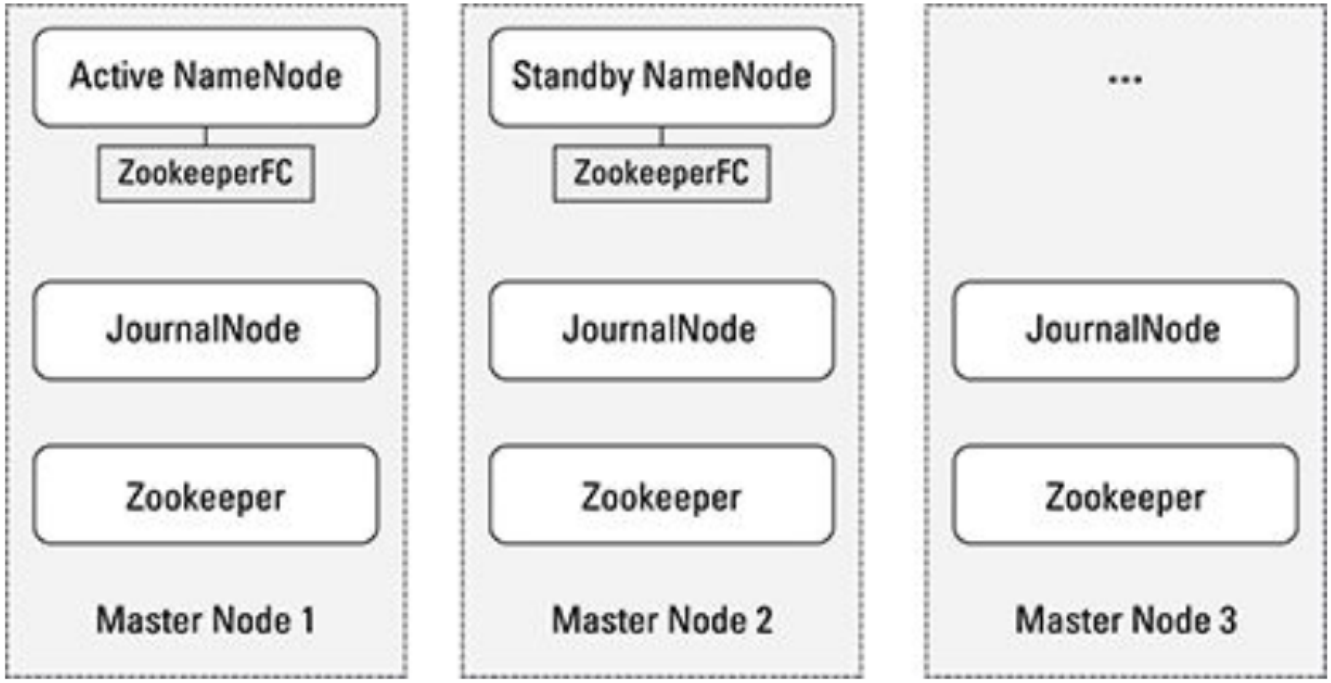
- Before Hadoop 2.0, NameNode was a single point of failure and operation limitation.
- Before Hadoop 2, Hadoop clusters usually have fewer clusters that were able to scale beyond 3,000 or 4,000 nodes.
- Multiple NameNodes can be used in Hadoop 2.x. (HDFS High Availability feature – one is in an Active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



http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/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.

Data Compression in HDFS

Columbia University

Table 5-1 Hadoop Codecs

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.

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

- % 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 director in HDFS can be specified in a fully qualified way, such as:
  hdfs://namenodehost/parent/child or hdfs://namenodehost

- 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

Several useful commands for HDFS -- II

For example, to create a directory named "joanna", run this mkdir command:

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

Use the Hadoop put command to copy a file from your local file system to HDFS:

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

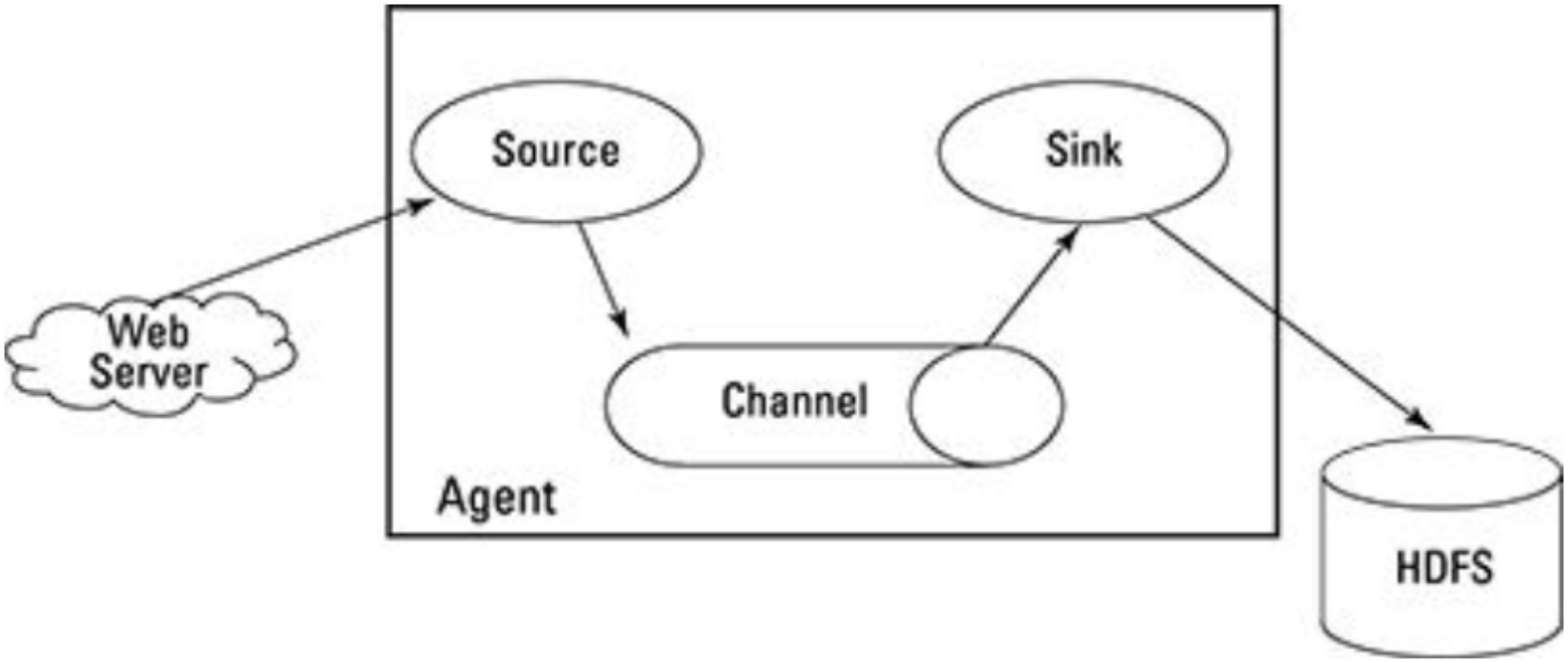
For example, to copy a file named data.txt to this new directory, run the following put command:

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

Run the ls command to get an HDFS file listing:

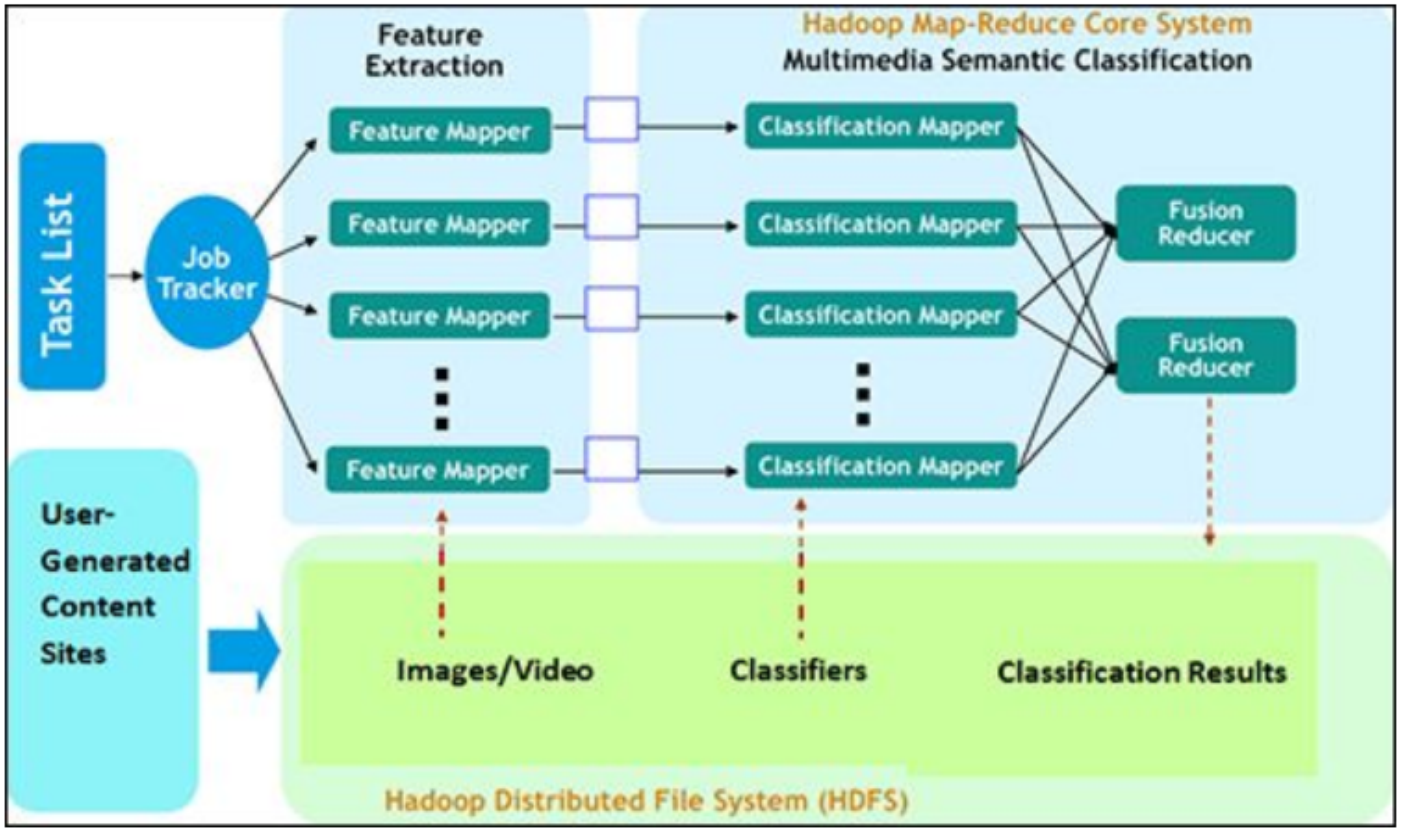
```
$ hadoop hdfs dfs -ls .
```

Ingesting Log Data -- Flume



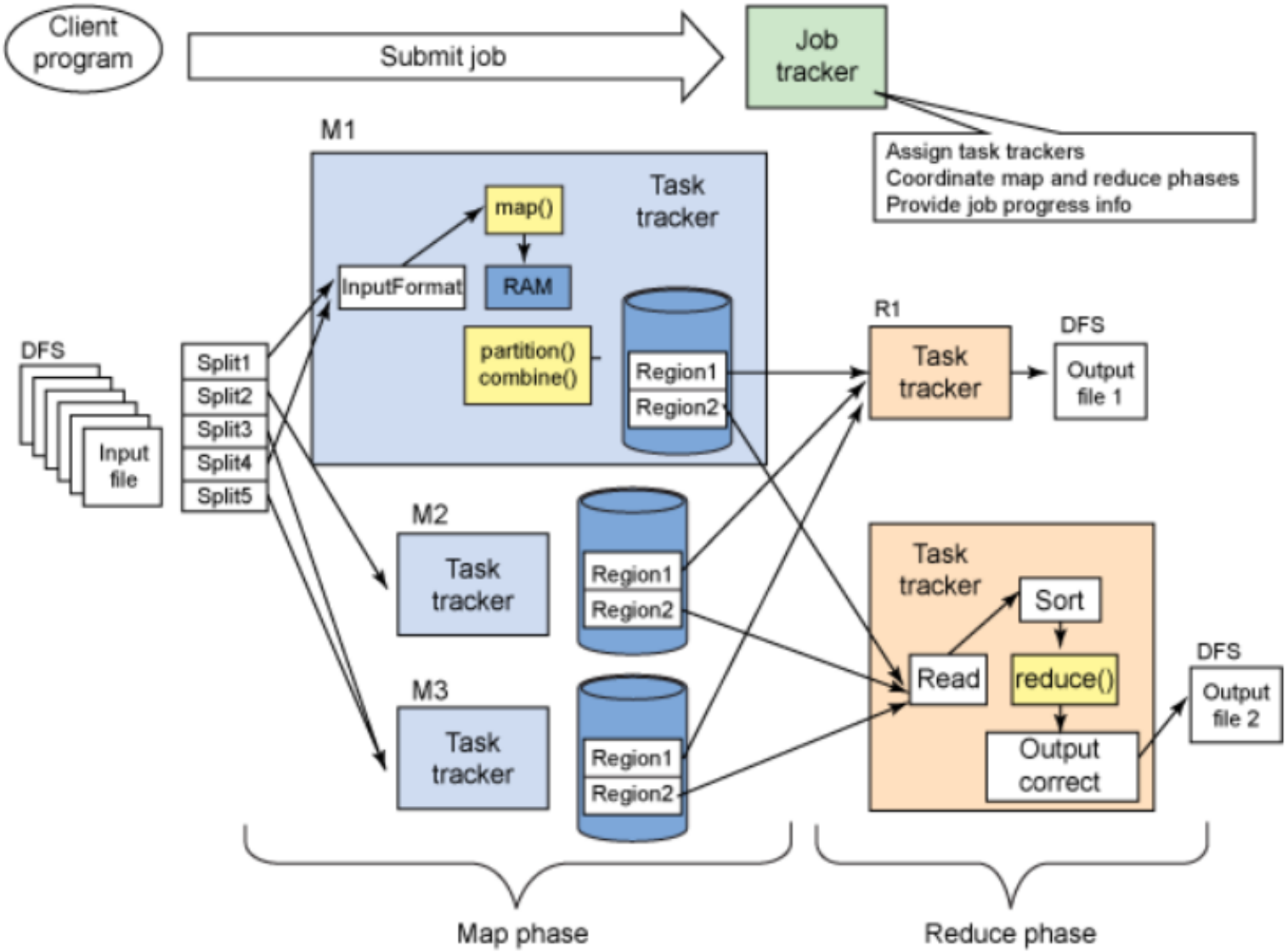
Ingesting stream data

How Hadoop Works



http://www.alex-hanna.com

Remind -- MapReduce Data Flow



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

MapReduce Use Case Example – flight data

- Data Source: 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/

Columbia University

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

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

Flight Data Schema

	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
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

MapReduce Use Case Example – flight data

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

Listing 6-1: 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
```

MapReduce Use Case Example – flight data

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

Listing 6-2: Pesudocode for Calculating The Number of Flights By Carrier in Parallel

```
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
```

MapReduce application flow

Columbia University

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.

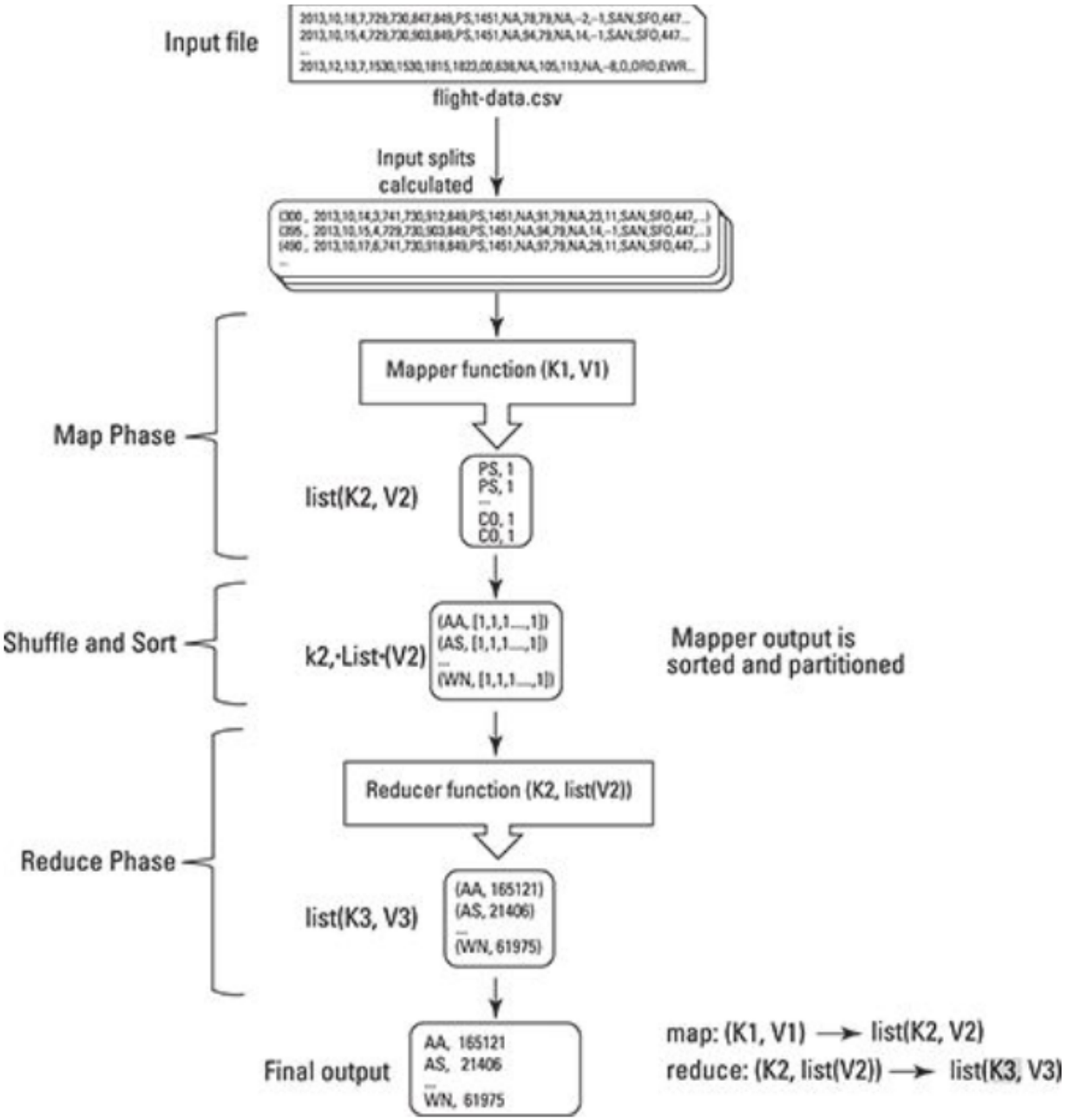
Run the specified algorithm against each record in the data set until all the records are processed. The individual instance of the application running against a block of data in a data set is known as a *mapper task*. (This is the mapping part of MapReduce.)

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.

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

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*.

MapReduce steps for f data computation

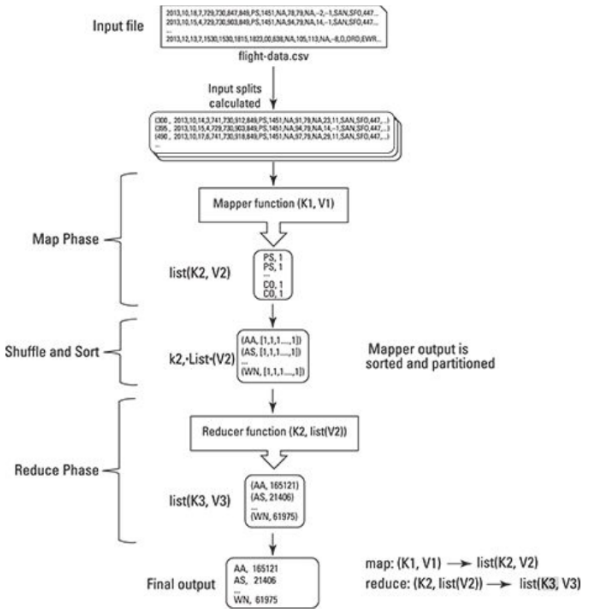


Create FlightsByCarrier.java:

Listing 6-3: The 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");
```



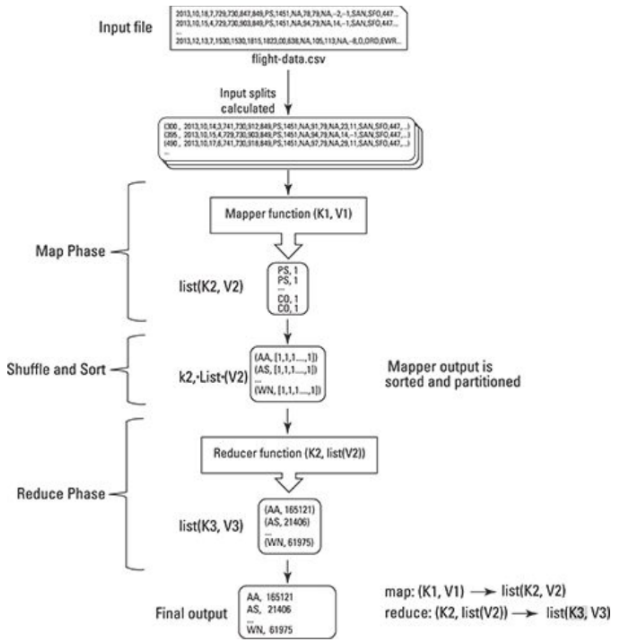
FlightsByCarrier application

```
@@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);
      }
}
```

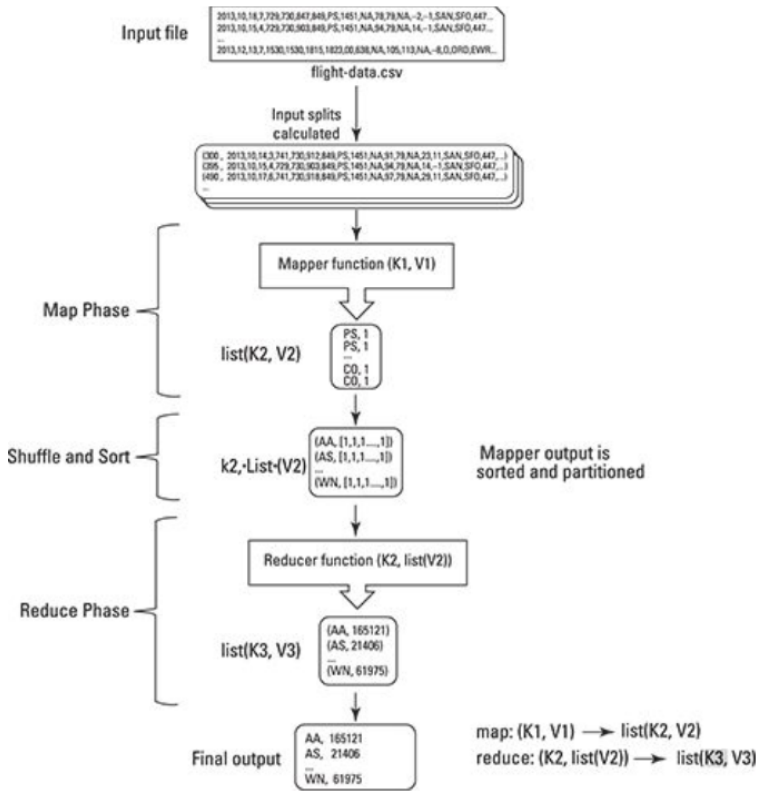


FlightsByCarrier Mapper

Listing 6-4: The 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));
      }
    }
}
```

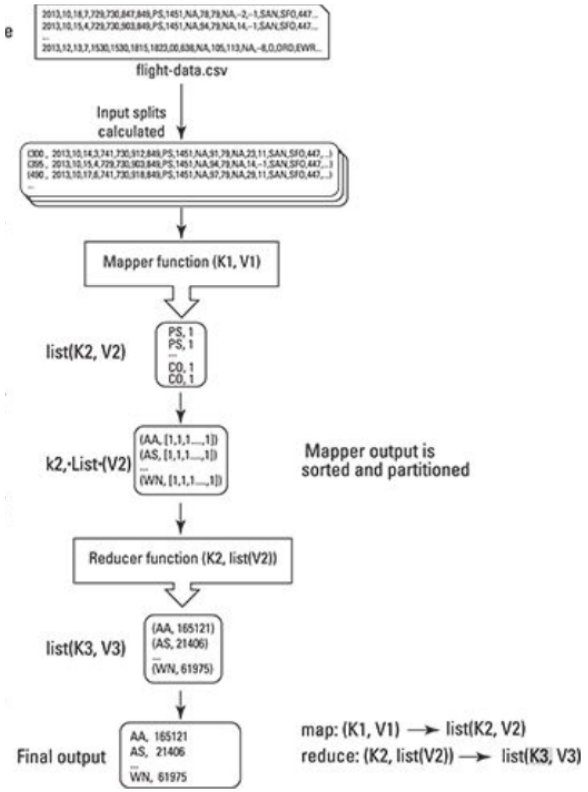


Listing 6-5: The FlightsByCarrier Reducer Code

```
 @@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));
    }
}
```



Run the code

To run the FlightsByCarrier application, follow these steps:

Go to the directory with your Java code and compile it using the following command:

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

Build a JAR file for the application by using this command:

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

Run the driver application by using this command:

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

See Result

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

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

You see 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
```

Using Pig Script

- 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;
```

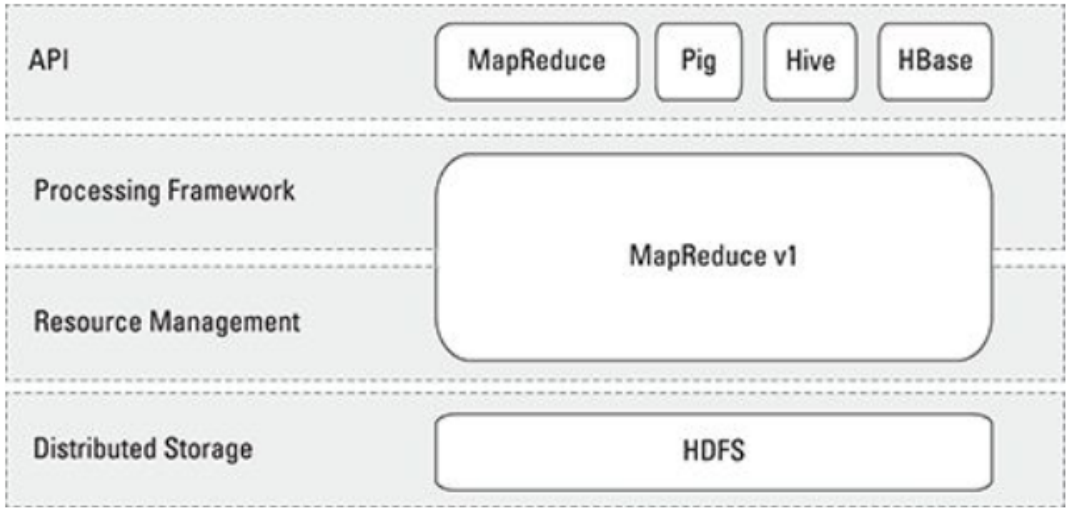
- Execute it: pig totalmiles.pig
- See result: hdfs dfs –cat /user/root/totalmiles/part-r-00000

  ➔ 775009272

- YARN – Yet Another Resource Negotiator:
  - A Tool that enables the other processing frameworks to run on Hadoop.
  - A general-purpose resource management facility that can schedule and assign CPU cycles and memory (and in the future, other resources, such as network bandwidth) from the Hadoop cluster to waiting applications.

➔YARN has converted Hadoop from simply a batch processing engine into a platform for many different modes of data processing, from traditional batch to interactive queries to streaming analysis.

Four distinctive layers of Hadoop



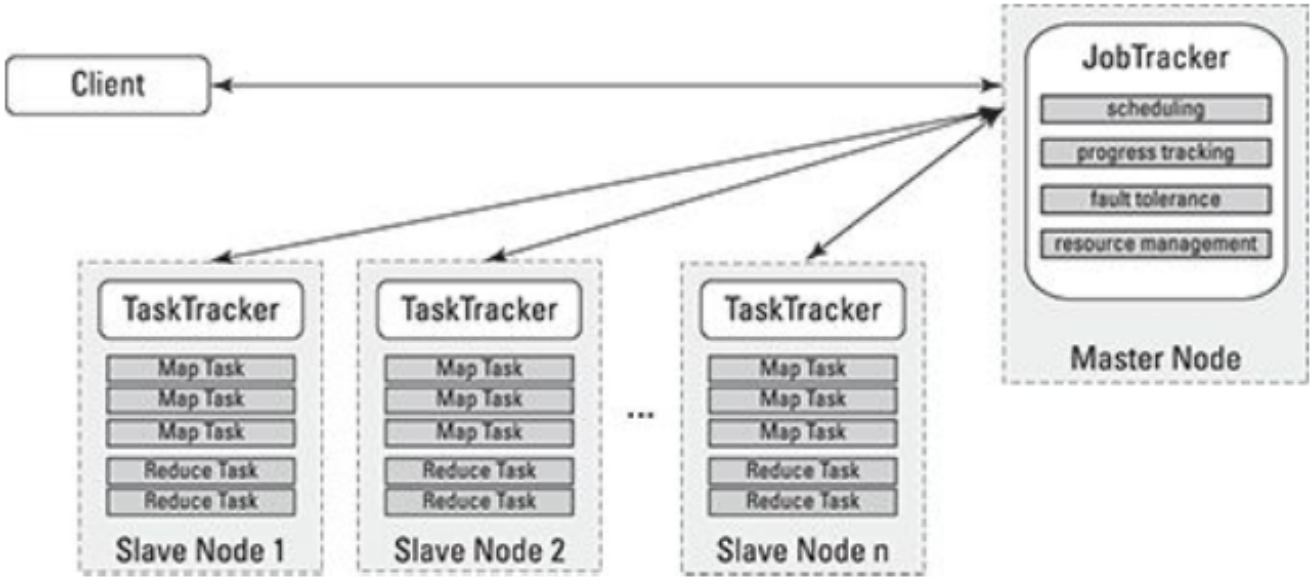
Distributed storage: The Hadoop Distributed File System (HDFS) is the storage layer where the data, interim results, and final result sets are stored.

Resource management: In addition to disk space, all slave nodes in the Hadoop cluster have CPU cycles, RAM, and network bandwidth. A system such as Hadoop needs to be able to parcel out these resources so that multiple applications and users can share the cluster in predictable and tunable ways. This job is done by the JobTracker daemon.

Processing framework: The MapReduce process flow defines the execution of all applications in Hadoop 1. As we saw in Chapter 6, this begins with the map phase; continues with aggregation with shuffle, sort, or merge; and ends with the reduce phase. In Hadoop 1, this is also managed by the JobTracker daemon, with local execution being managed by TaskTracker daemons running on the slave nodes.

Application Programming Interface (API): Applications developed for Hadoop 1 needed to be coded using the MapReduce API. In Hadoop 1, the Hive and Pig projects provide programmers with easier interfaces for writing Hadoop applications, and underneath the hood, their code compiles down to MapReduce.

Hadoop 1 execution



1. The client application submits an application request to the JobTracker.
2. The JobTracker determines how many processing resources are needed to execute the entire application.
3. The JobTracker looks at the state of the slave nodes and queues all the map tasks and reduce tasks for execution.
4. As processing slots become available on the slave nodes, map tasks are deployed to the slave nodes. Map tasks are assigned to nodes where the same data is stored.
5. The JobTracker monitors task progress. If failure, the task is restarted on the next available slot.
6. After the map tasks are finished, reduce tasks process the interim results sets from the map tasks.
7. The result set is returned to the client application.

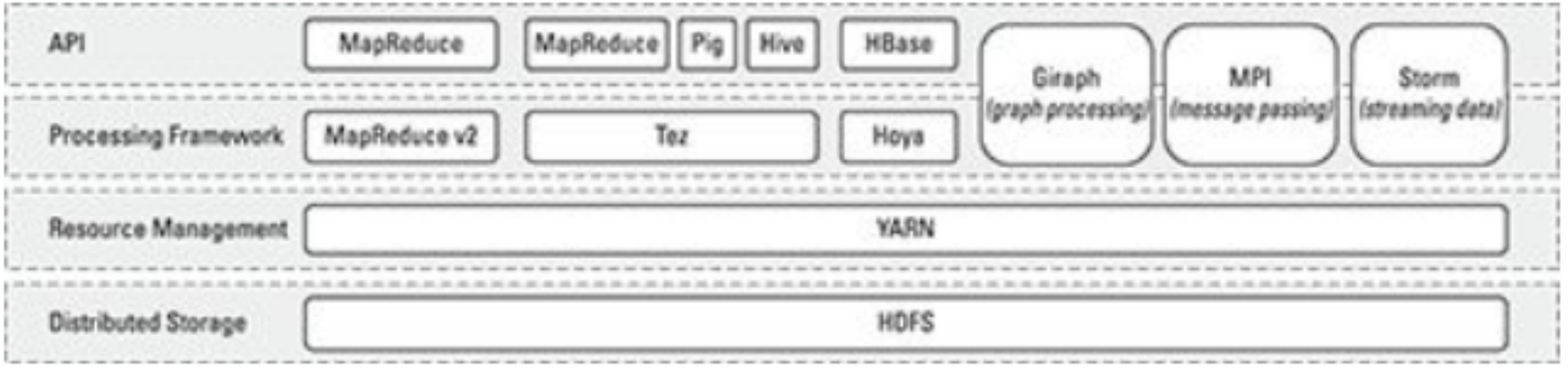
Limitation of Hadoop 1

- MapReduce is a successful batch-oriented programming model.
- A glass ceiling in terms of wider use:
  - Exclusive tie to MapReduce, which means it could be used only for batch-style workloads and for general-purpose analysis.
- Triggered demands for additional processing modes:
  - Graph Analysis
  - Stream data processing
  - Message passing

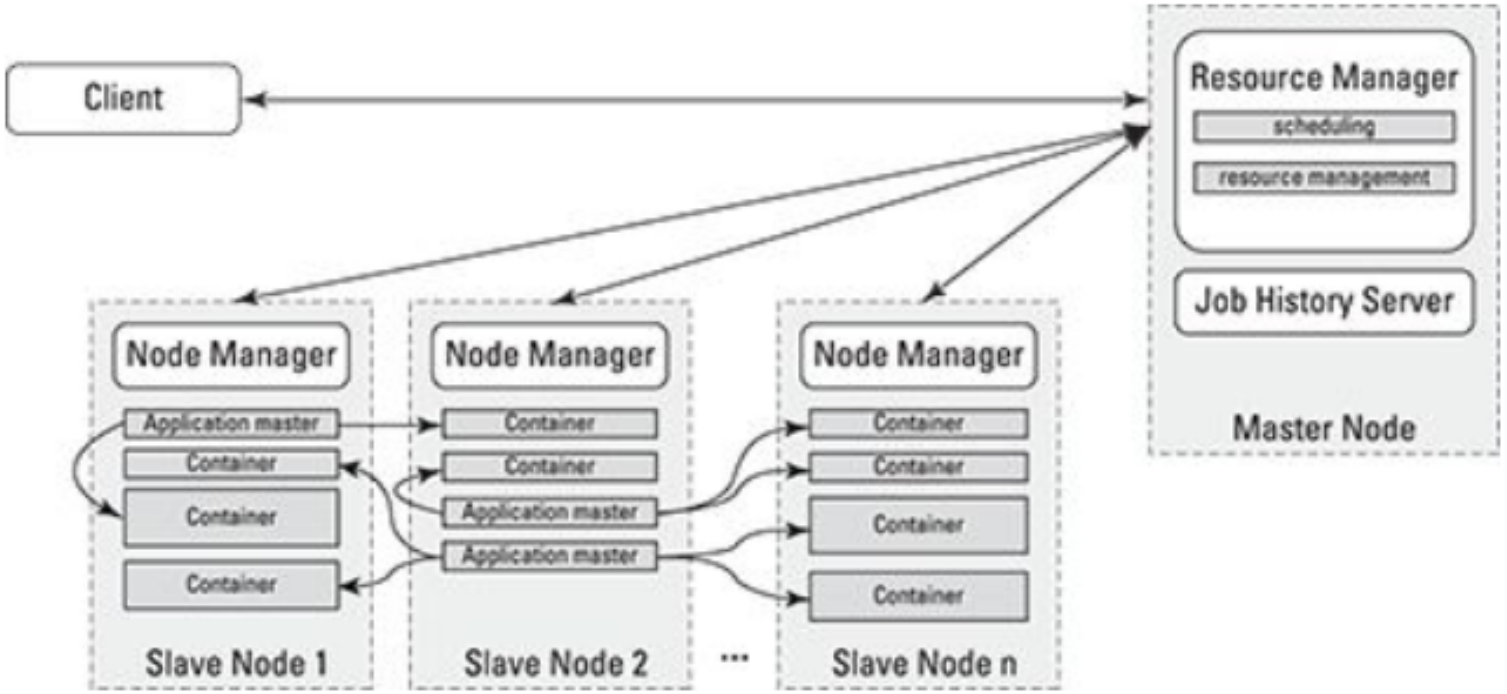
➔ Demand is growing for real-time and ad-hoc analysis
➔ Analysts ask many smaller questions against subsets of data and need a near-instant response.
➔ Some analysts are more used to SQL & Relational databases

YARN was created to move beyond the limitation of a Hadoop 1 / MapReduce world.

Hadoop 2 Data Processing Architecture

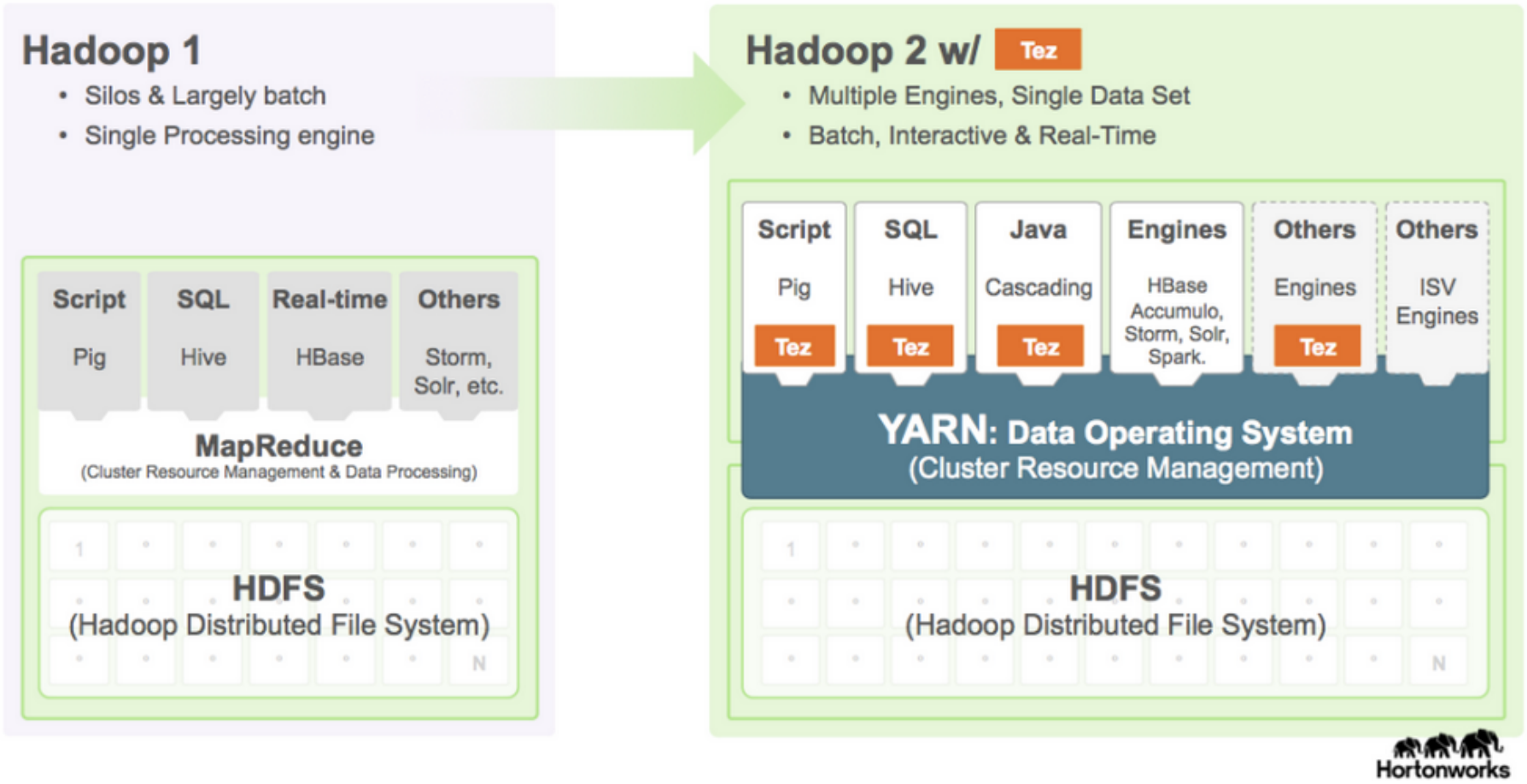


YARN's application execution

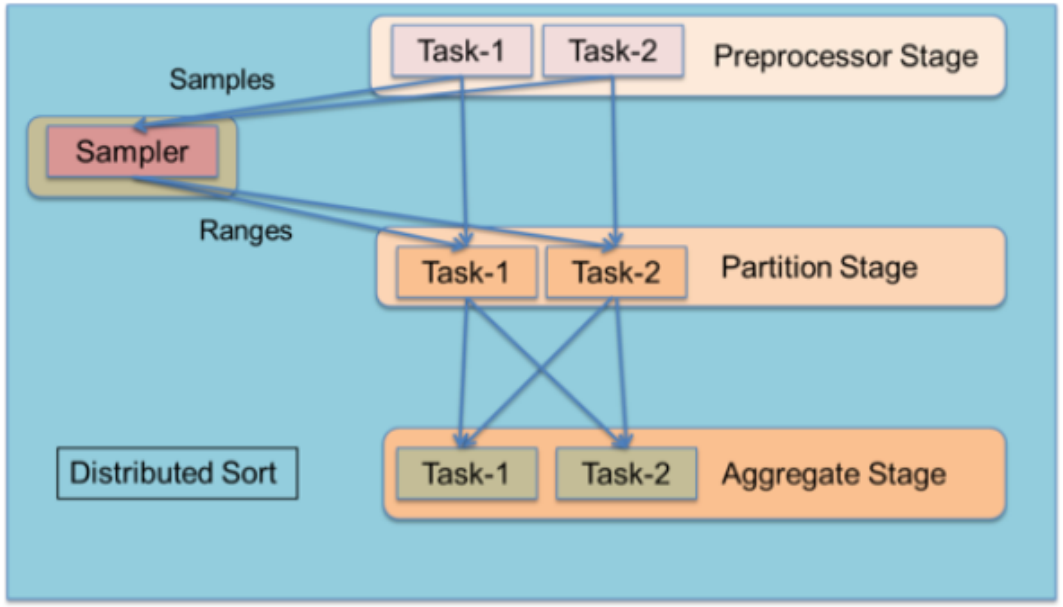


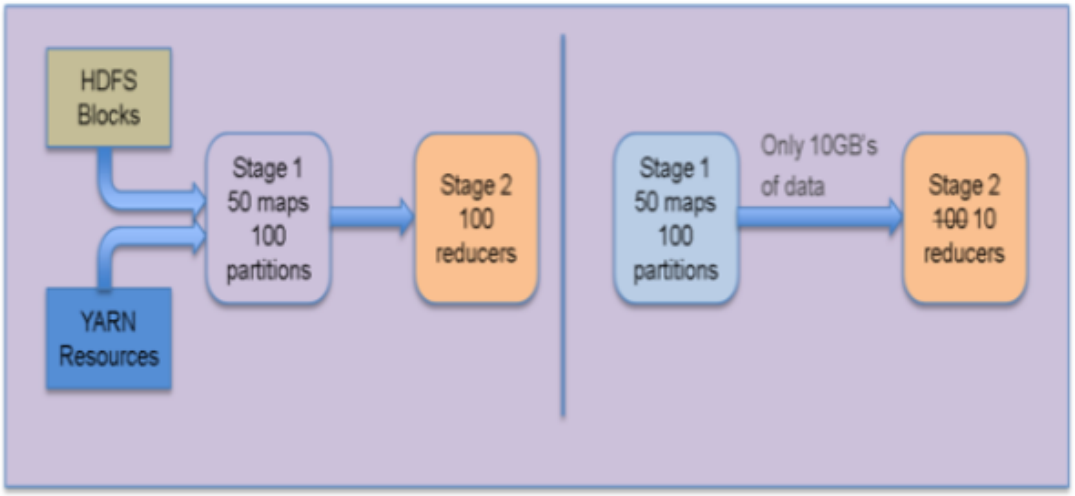
- Client submits an application to Resource Manager.
- Resource Manager asks a Node Manager to create an Application Master instance and starts up.
- Application Manager initializes itself and register with the Resource Manager
- Application manager figures out how many resources are needed to execute the application.
- The Application Master then requests the necessary resources from the Resource Manager. It sens heartbeat message to the Resource Manager throughout its lifetime.
- The Resource Manager accepts the request and queue up.
- As the requested resources become available on the slave nodes, the Resource Manager grants the Application Master leases for containers on specific slave nodes.
- …. ➔ only need to decide on how much memory tasks can have.

Apache Tez



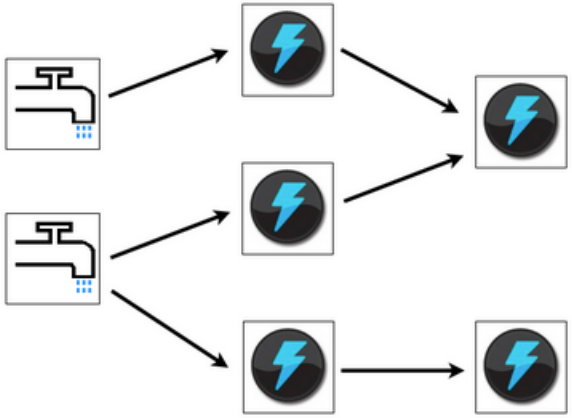
Tez's characteristics

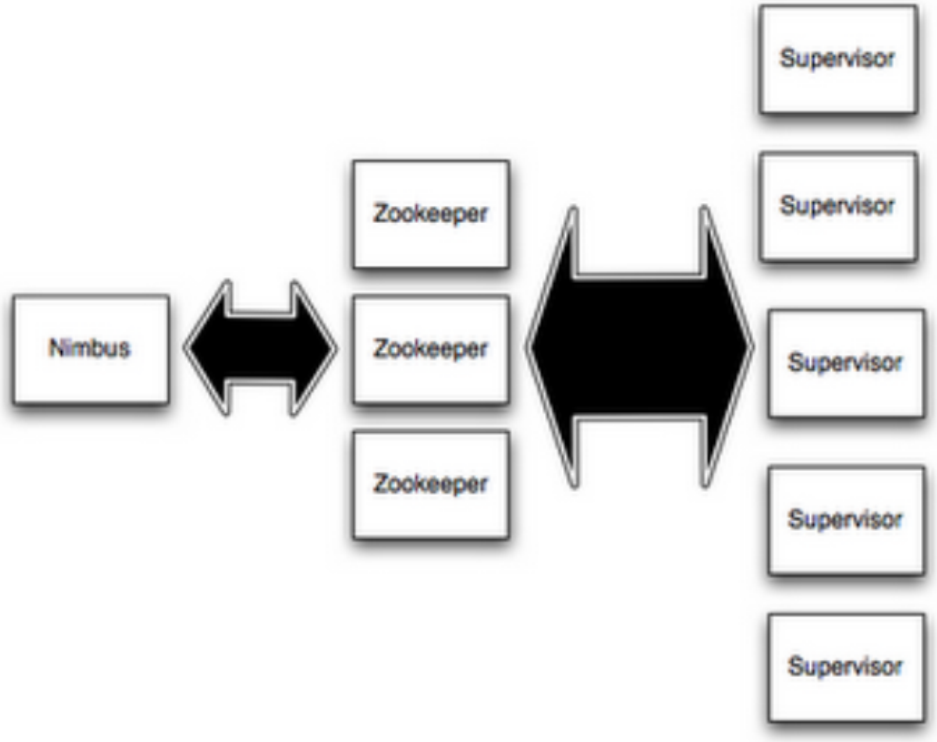




- Dataflow graph with vertics to express, model, and execute data processing logic
- Performance via Dynamic Graph Reconfiguration
- Flexible Input-Procesor-Output task model
- Optimal Resource Management

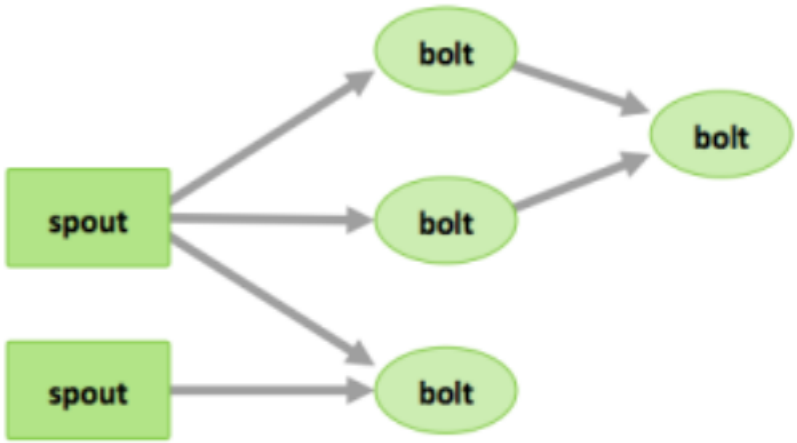
Stream Processing
-- On Hadoop, you run MapReduce jobs; On Storm, you run Topologies.
-- Two kinds of nodes on a Storm cluster:
  -- the master node runs “Nimbus”
  -- the worker nodes called the Supervisor.





How Storm processed data

- **Tuples**– an ordered list of elements. For example, a "4-tuple" might be (7, 1, 3, 7)
- **Streams** – an unbounded sequence of tuples.
- **Spouts** –sources of streams in a computation (e.g. a Twitter API)
- **Bolts** – process input streams and produce output streams. They can: run functions; filter, aggregate, or join data; or talk to databases.
- **Topologies** – the overall calculation, represented visually as a network of spouts and bolts (as in the following diagram)



Initiative Goals

Streams in HDP

Bringing stream data processing to enterprise Apache Hadoop and Hortonworks Data Platform.

Storm on YARN

Use the YARN Hadoop operating system to allow multiple workloads to be applied to Hadoop data simutaneously.

Enterprise Readiness

Bring baseline high availability, management, authentication and advanced scheduling to Storm.

Questions?