Introduction to Hadoop and HDFS

Lecture 2
October 19th, 2017

Jae W. Lee (jaewlee@snu.ac.kr)
Computer Science and Engineering
Seoul National University

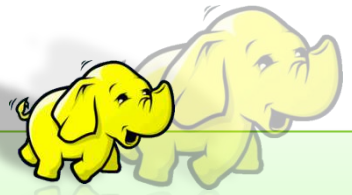
***Slide credits**: Rafael Coss (Hortonworks); Owen O'Malley (Hortonworks); DarrelAucoin (Intro to Hadoop); Tom White (Hadoop, The Definitive Guide)*

Outline

- **What is Apache Hadoop?**
- Key Components of Hadoop Stack
- Hadoop Distributed File System (HDFS)
- HDFS File System Shell Commands
- Apache Spark on HDFS

What is Apache Hadoop?

- **Solution for Big Data**
  - Designed for volume, velocity, variety & complexity of data
- **Data Platform Deployed on Commodity Hardware that**
  - Stores petabytes of data reliably
  - Runs huge distributed applications
  - Enables a rational economics model
- **Set of Open Source Projects**
  - Apache Software Foundation
  - Loosely coupled, ship early/often

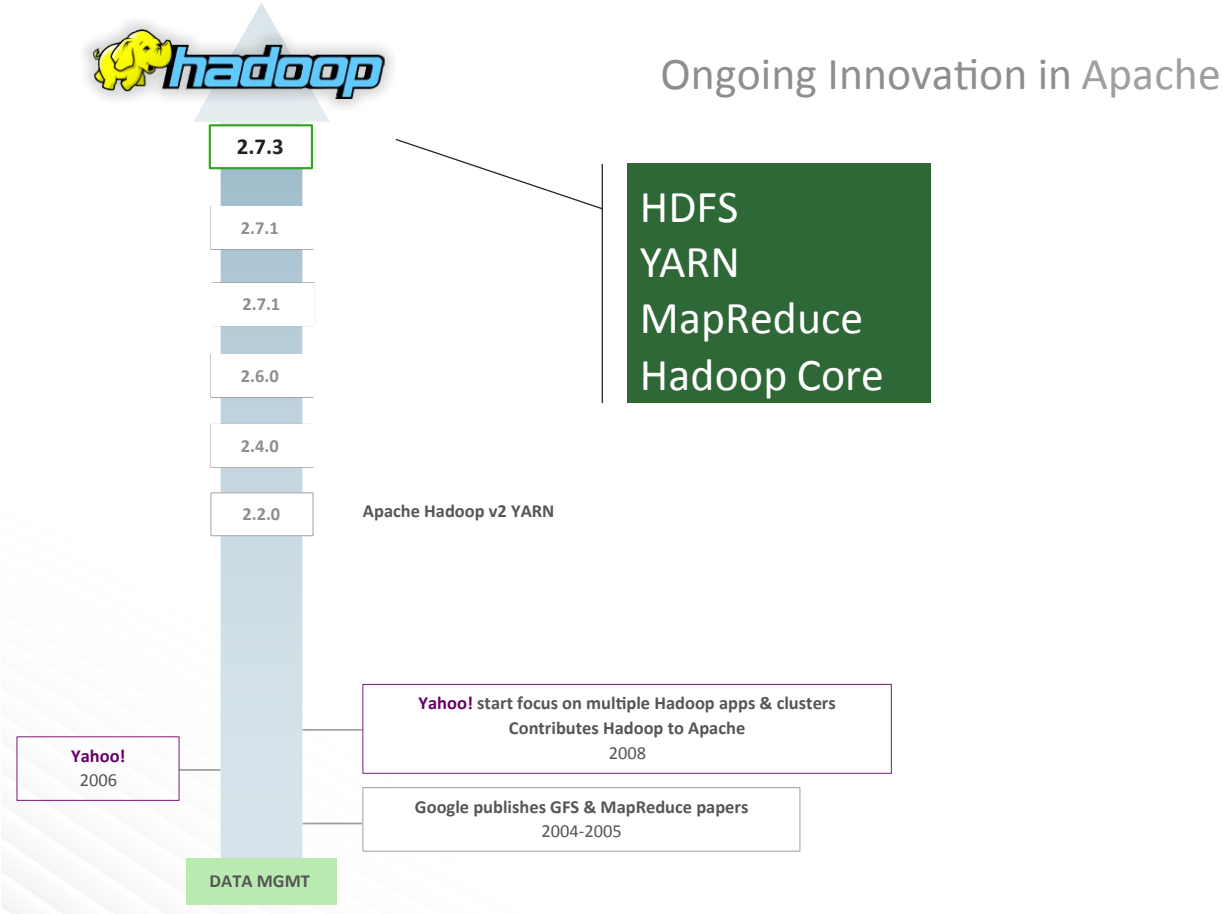


One of the best examples of open source driving innovation and creating a market

Source: Hortonworks

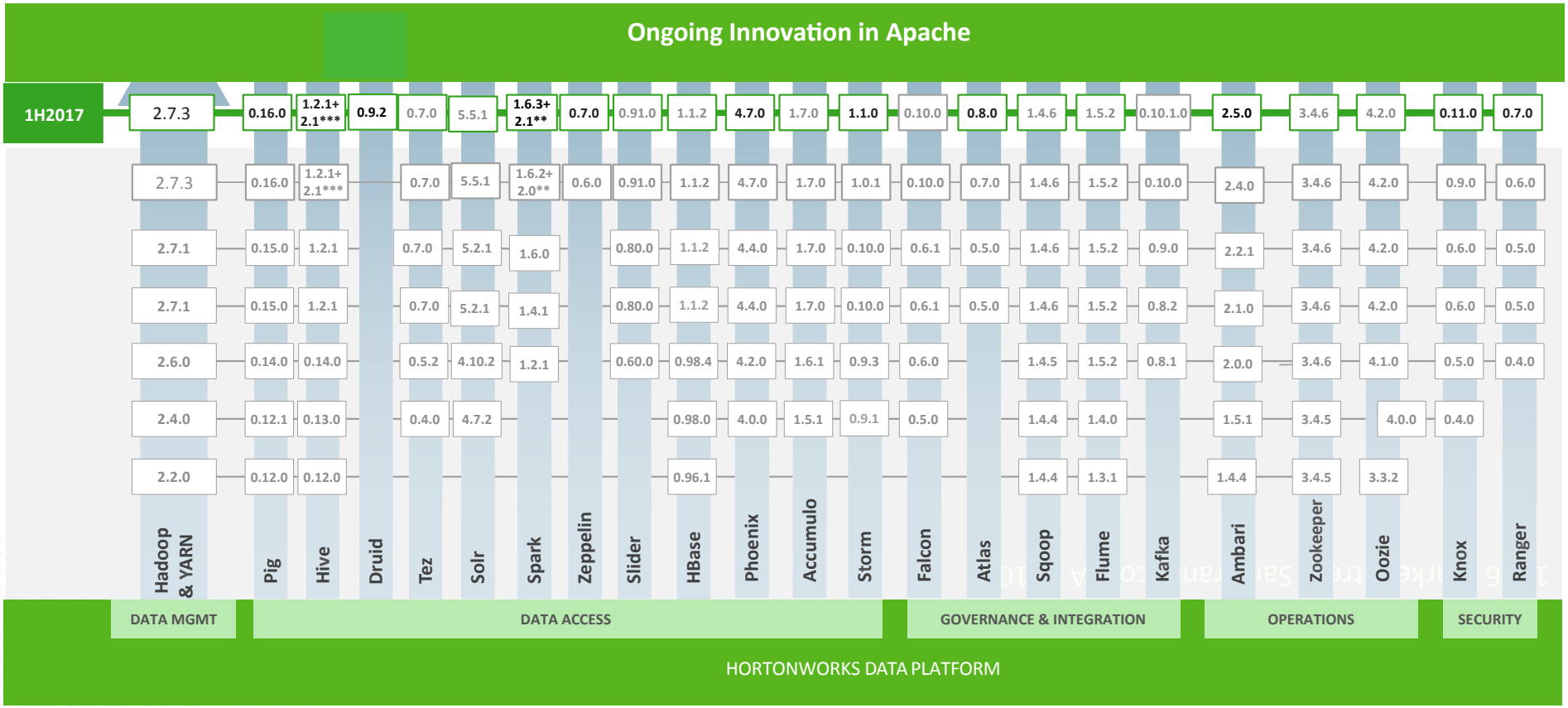
What is Apache Hadoop?

- **A brief history**



What is Apache Hadoop?

- **A brief history: Versions**



** HDP 2.6 – Shows current Apache branches being used. Final component version subject to change based on Apache release process.*

*** Spark 1.6.3+ Spark 2.1 – HDP 2.6 supports both Spark 1.6.3 and Spark 2.1 as GA*

**** Hive 2.1 is GA within HDP 2.6.*

What is Apache Hadoop?

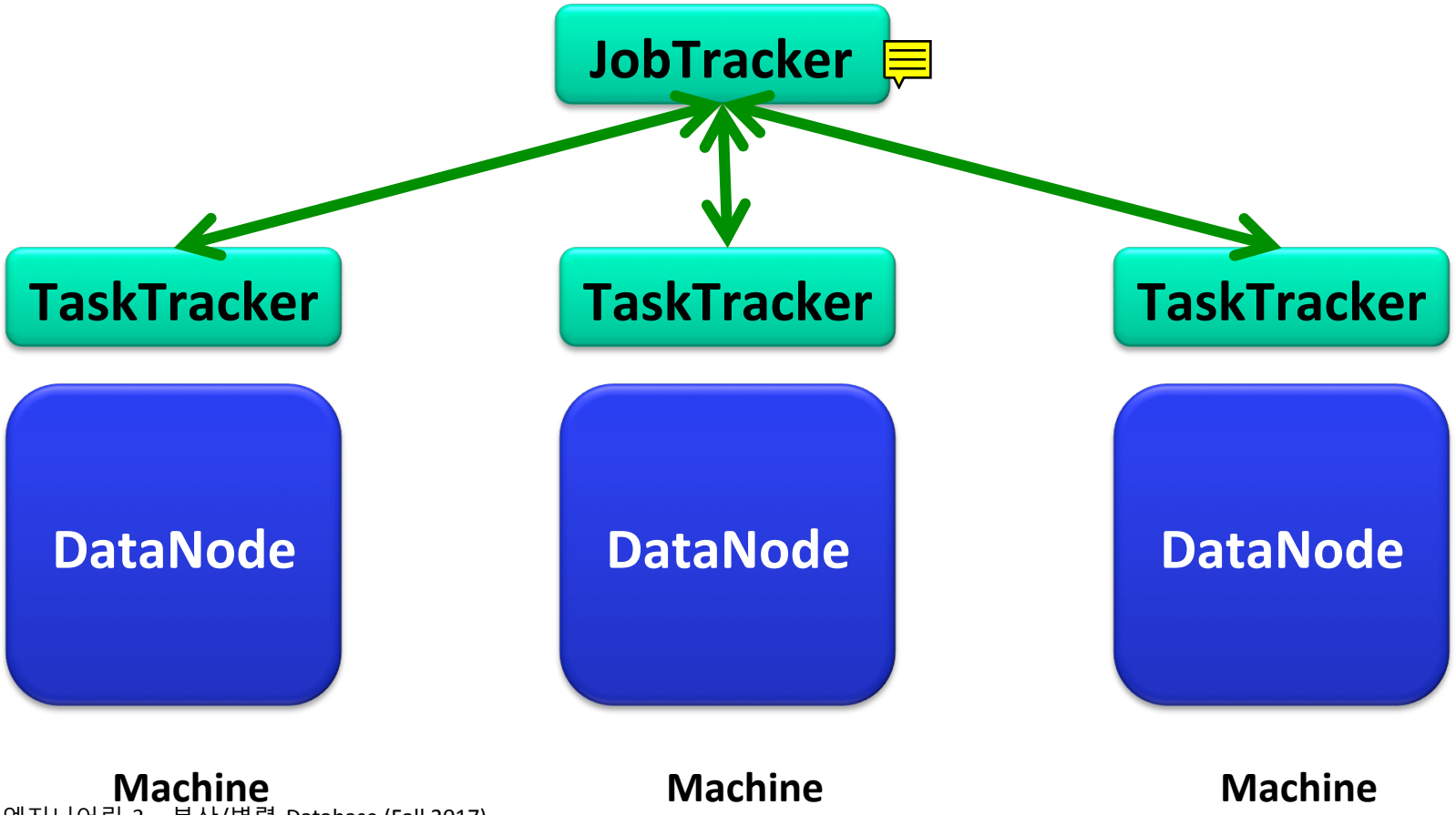
- **Open Source Apache Project**
- **Hadoop Core includes:**
  - MapReduce: the processing part of Hadoop
  - Hadoop File System (HDFS): the data part of Hadoop
- **Written in Java**
- **Runs on**
  - Linux, Mac OS/X, Windows, and Solaris
  - Commodity hardware
- **Key attributes**
  - Redundant and reliable (no data loss)
  - Extremely powerful/scalable
  - Batch processing centric
  - Easy to program distributed applications

Hadoop Structure

- **What's running on the machine?**
  - The MapReduce server on a typical machine is called a TaskTracker.
  - The HDFS server on a typical machine is called a DataNode.

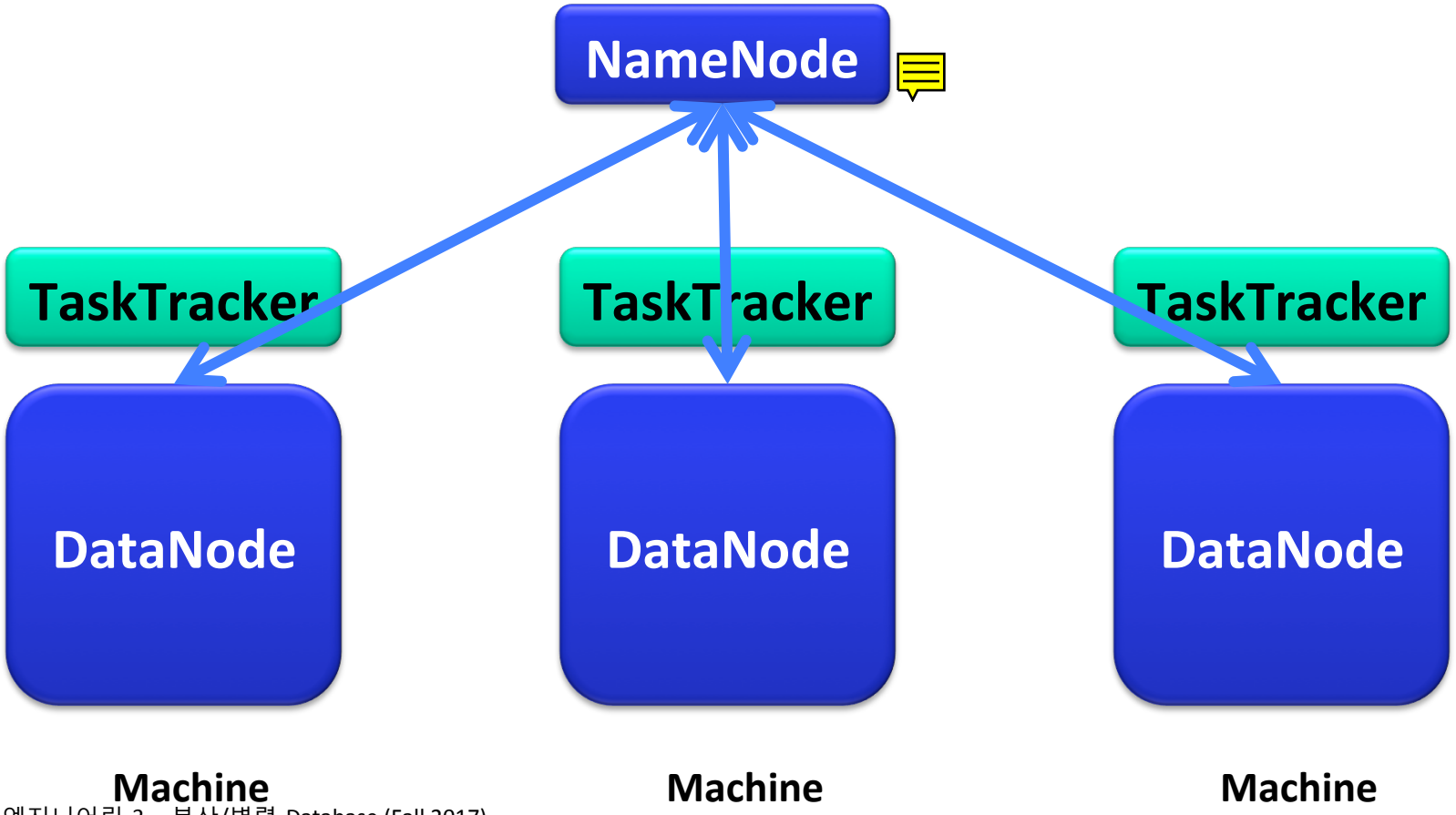
Hadoop Structure

- **Hadoop cluster: MapReduce**
  - Having multiple machines with Hadoop creates a cluster.
  - JobTracker keeps track of jobs being run.



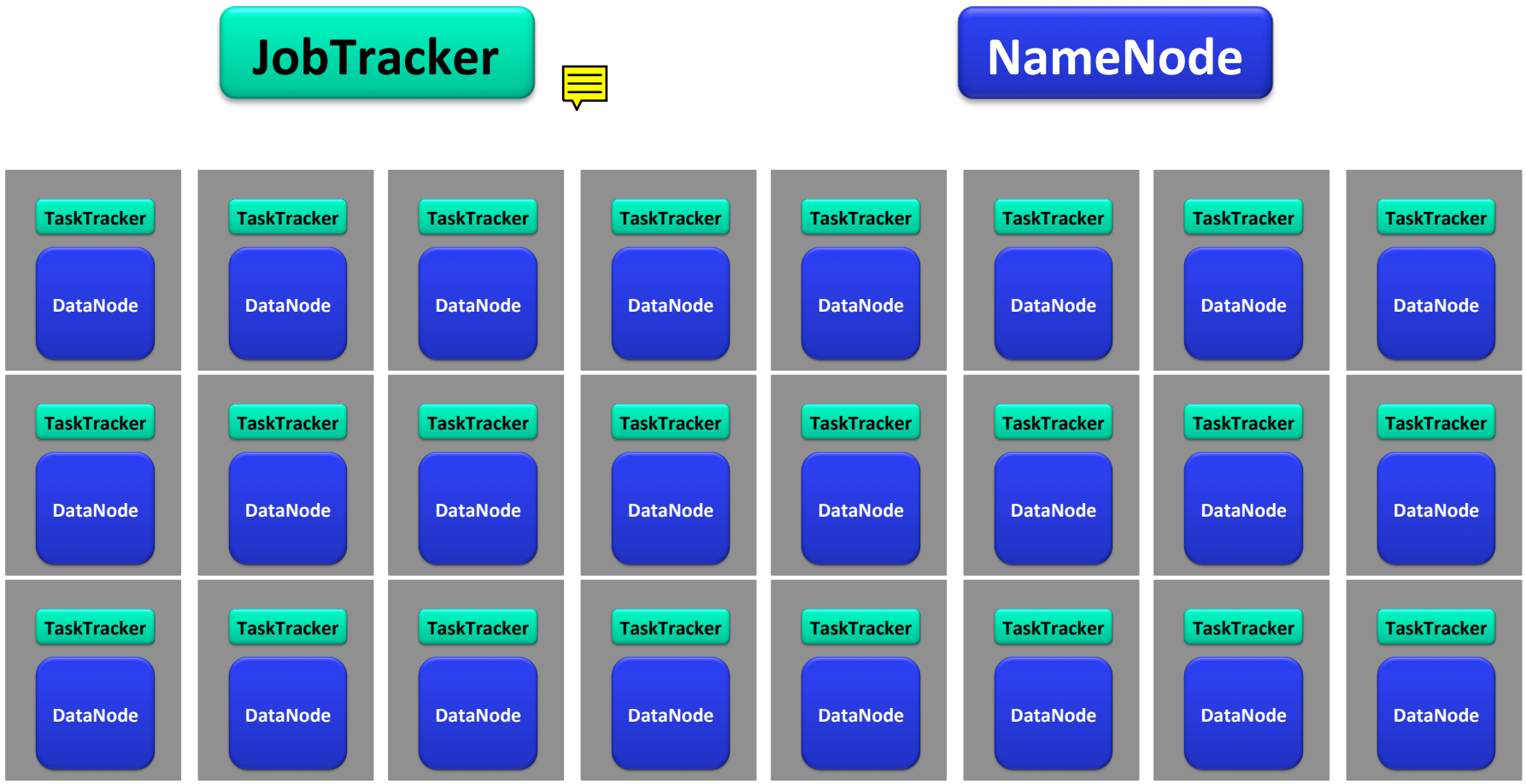
Hadoop Structure

- **Hadoop cluster**
  - NameNode keeps information on data location



Hadoop Structure

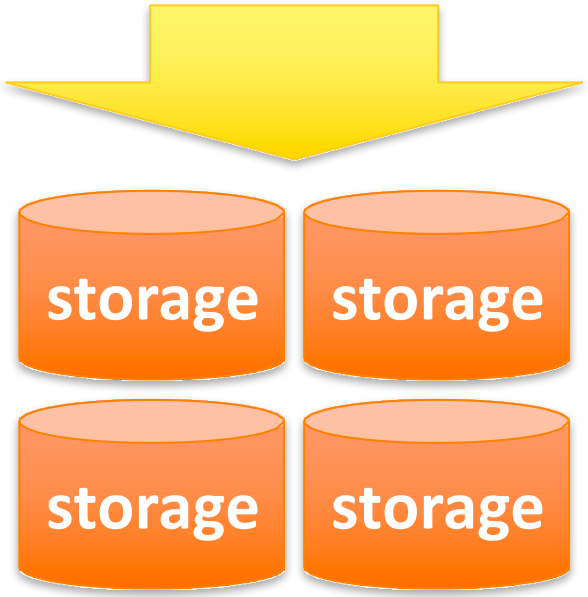
- Having many nodes..



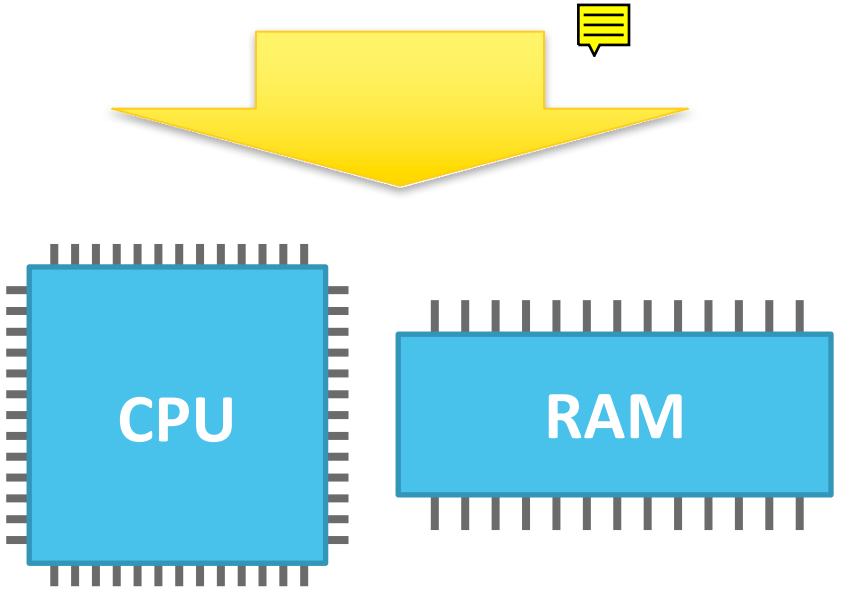
Hadoop Structure

- **Apache Hadoop = Storage + Compute**

Hadoop Distributed File System (HDFS)

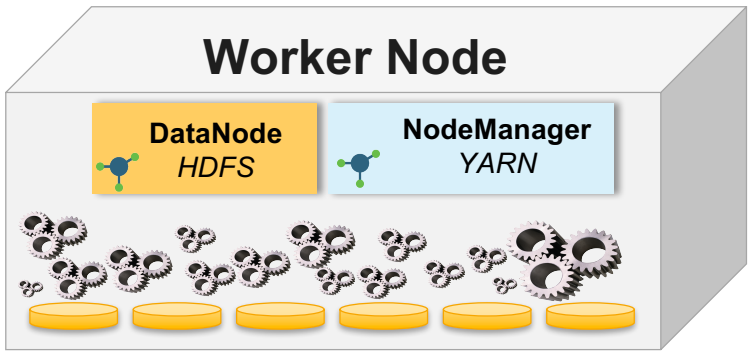


Yet Another Resource Negotiator (YARN)



Hadoop is ...

Core hadoop



Disk, CPU, Memory

+

NameNode
HDFS

`/directory/structure/in/memory.txt`

ResourceManager
YARN

Resource management + scheduling

Hadoop daemon

User application

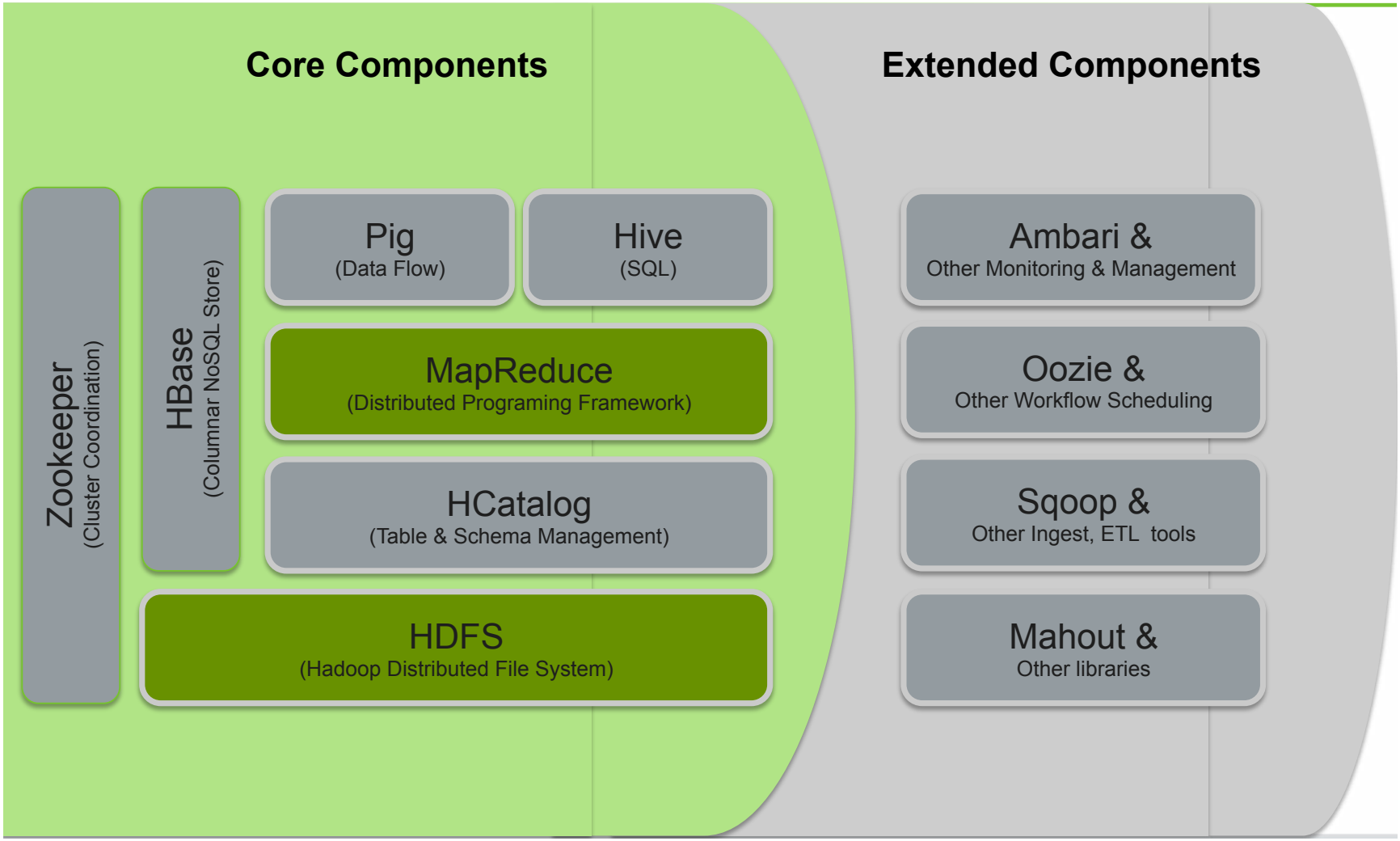
Hadoop is ...

- **Reliable**
  - Data is typically held on multiple DataNodes
  - Tasks that fail are redone
- **Scalable**
  - Same program runs on 1, 1000, or 4000 machines
  - Scales linearly
- **Simple**
  - Having simple APIs
- **Very powerful**
  - You can process in parallel masssive amounts of data
    - Petabytes of data
  - Processing in parallel allows for the timely processing of massive amounts of data

Outline

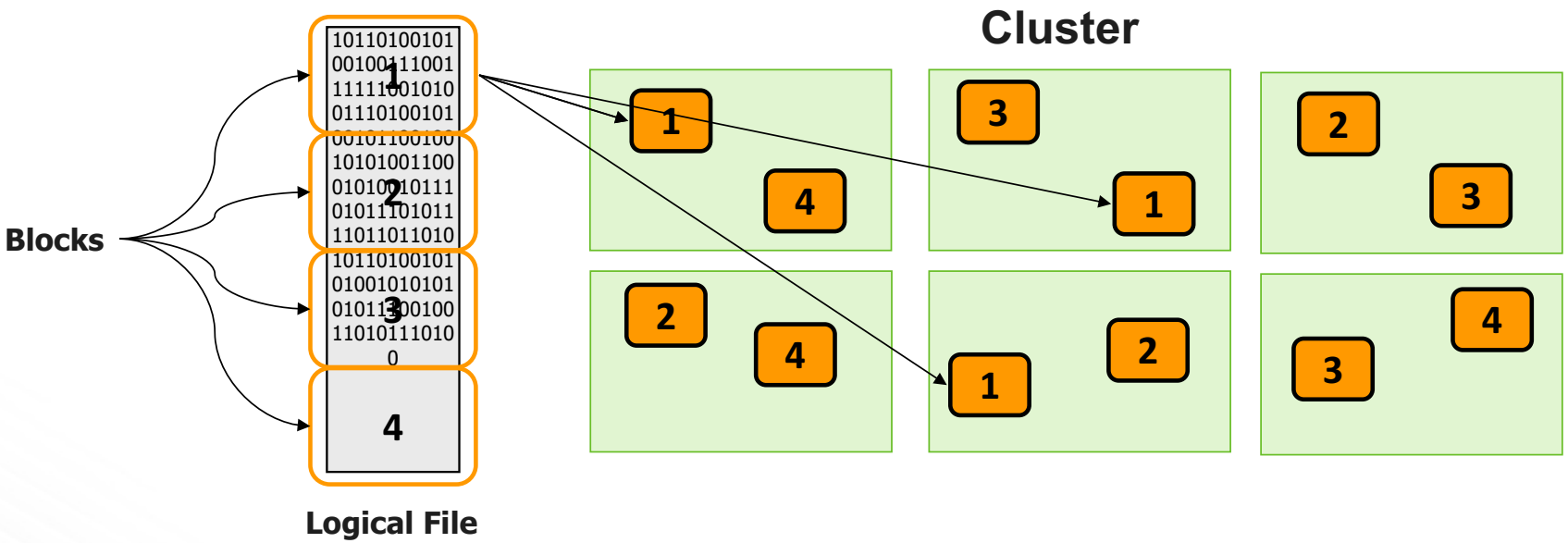
- What is Apache Hadoop?
- **Key Components of Hadoop Stack**
- Hadoop Distributed File System (HDFS)
- HDFS File System Shell Commands
- Apache Spark on HDFS

Key Components of Hadoop Stack



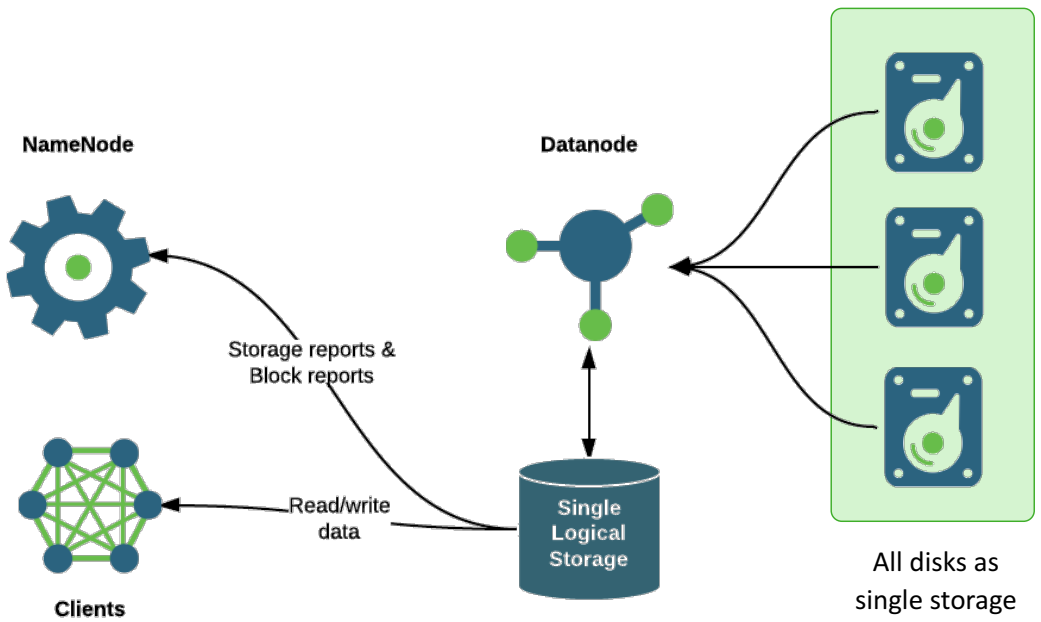
Key Components of Hadoop Stack

- **Hadoop Distributed File System (HDFS): Fault Tolerant Distributed Storage**
  - Divide files into big blocks and distribute 3 copies randomly across the cluster
  - Processing Data Locality
    - Not Just storage but computation



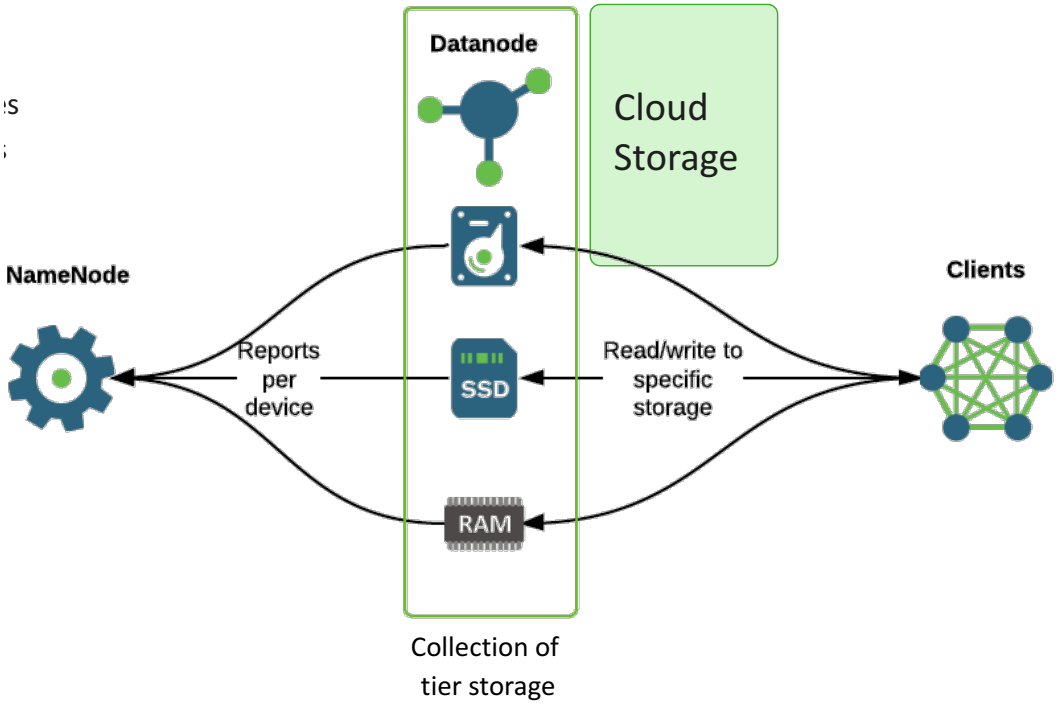
Key Components of Hadoop Stack

- **Hadoop Distributed File System (HDFS): Storage Architecture – Before**
  - DataNode is a single storage
  - Storage is uniform - Only storage type Disk
  - Storage types hidden from the file system



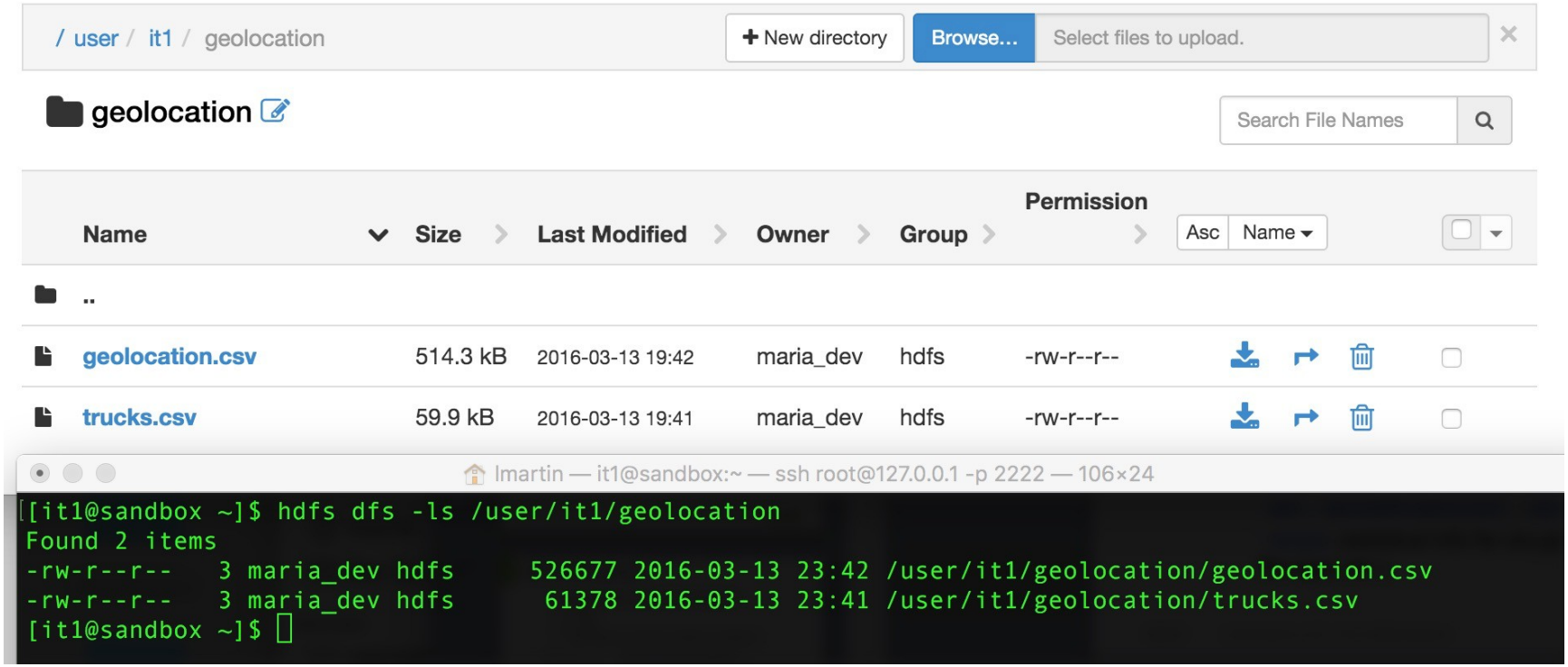
Key Components of Hadoop Stack

- **Hadoop Distributed File System (HDFS): Storage Architecture – Now**
  - New Architecture
    - DataNode is a collection of storages
    - Support different types of storages
      - Disk, SSDs, Memory
  - Block Storage Policies
    - Describes how to store data blocks in HDFS



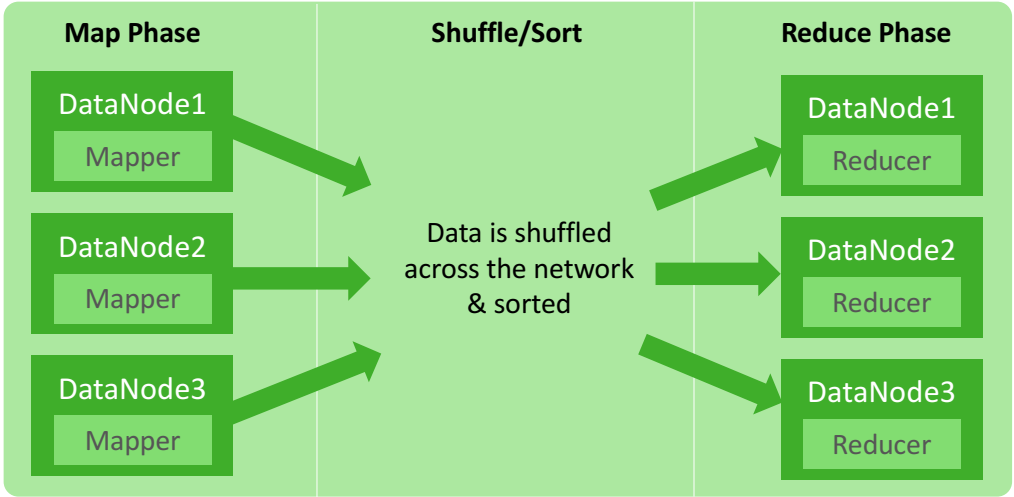
Key Components of Hadoop Stack

- **Hadoop Distributed File System (HDFS)**
  - It looks like a file system.



Key Components of Hadoop Stack

- **MapReduce: Batch access to data**
  - Framework
    - Made for developing distributed applications to process vast amounts of data in-parallel on large clusters
  - Proven
    - Reliable interface to Hadoop which works from GB to PB. But, batch oriented – Speed is not it's strong point.
  - Ecosystem
    - Ported to Hadoop 2 to run on YARN. Supports original investments in Hadoop by customers and partner ecosystem.



Key Components of Hadoop Stack

- **MapReduce: What is it?**
  - Break a large problem into sub-solutions

Map

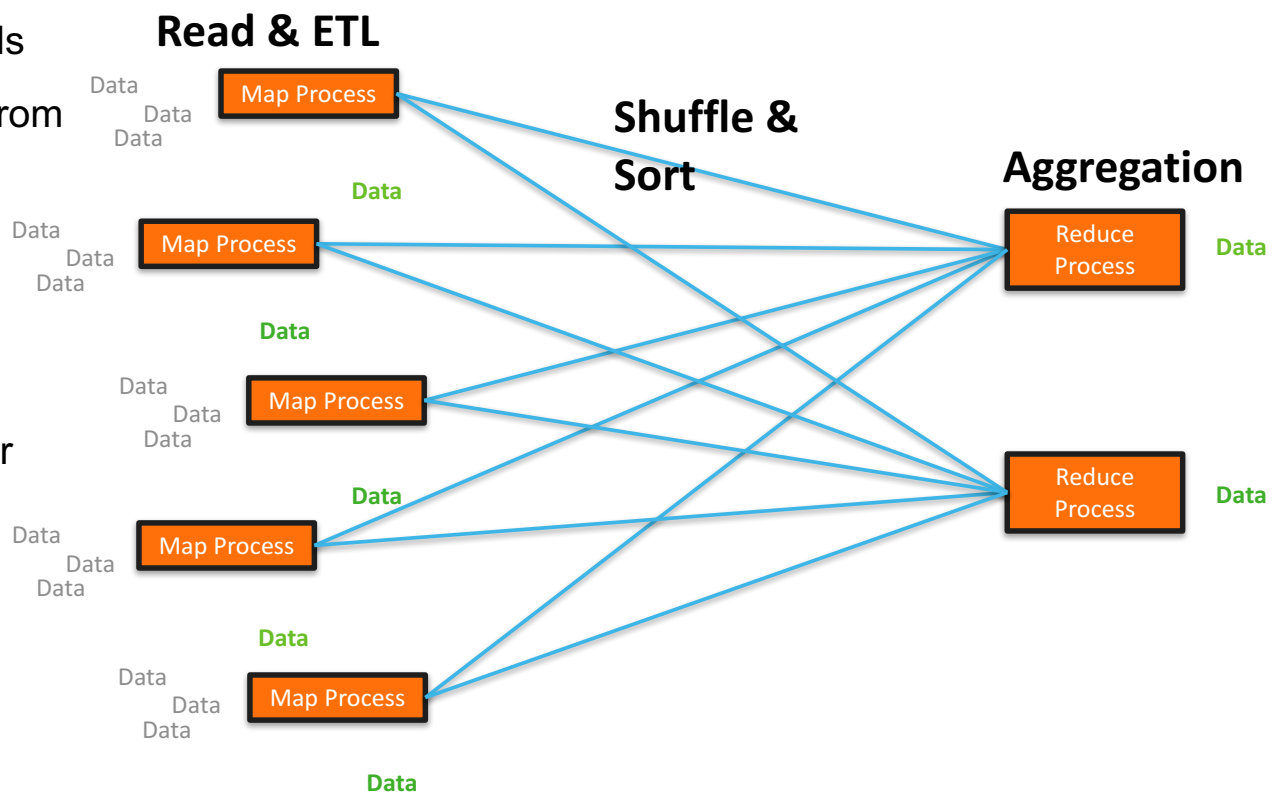
- Iterate over a large # of records
- Extract something of interest from each record

Shuffle

- Sort Intermediate results

Reduce

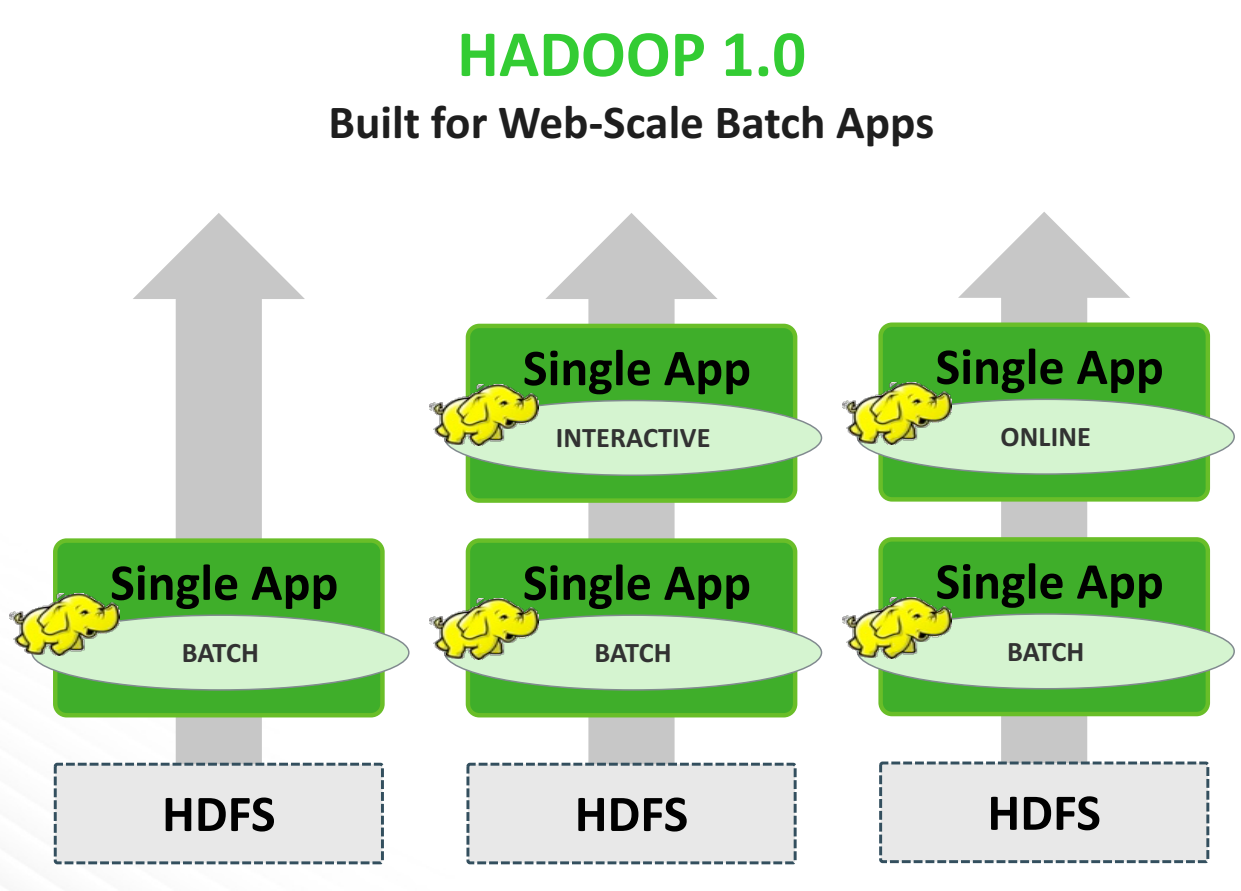
- Aggregate, summarize, filter or transform intermediate results
- Generate final output



* ETL: Extract, Transform, Load

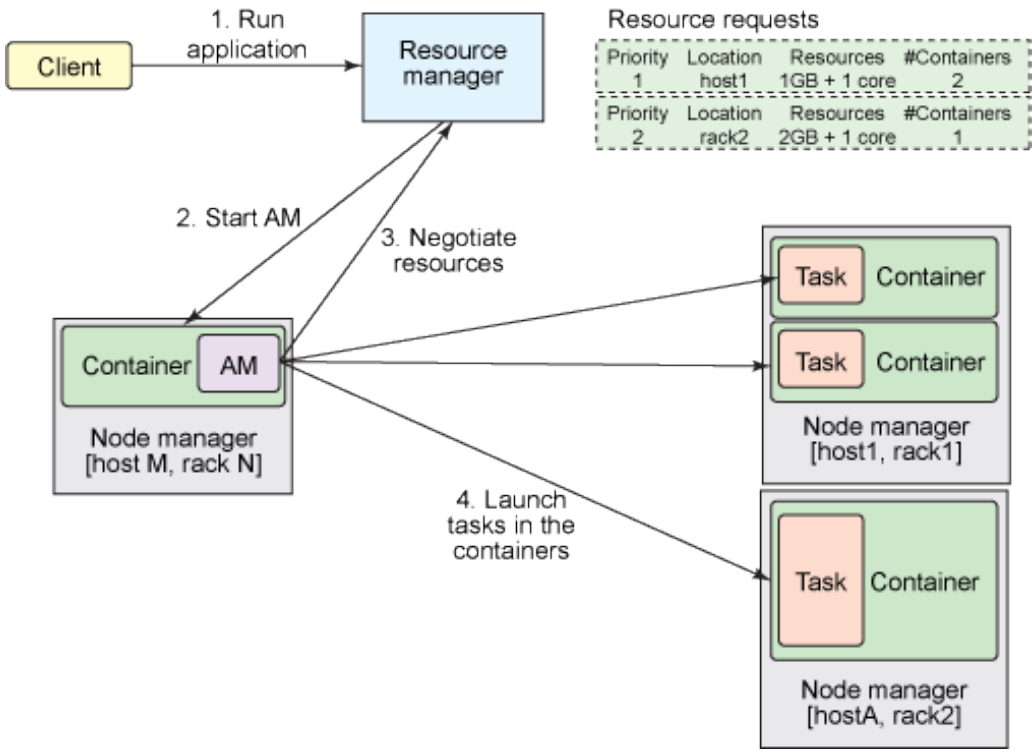
Key Components of Hadoop Stack

- **MapReduce: Key component for 1st Gen Hadoop**
  - Cost effective **batch** at scale



Key Components of Hadoop Stack

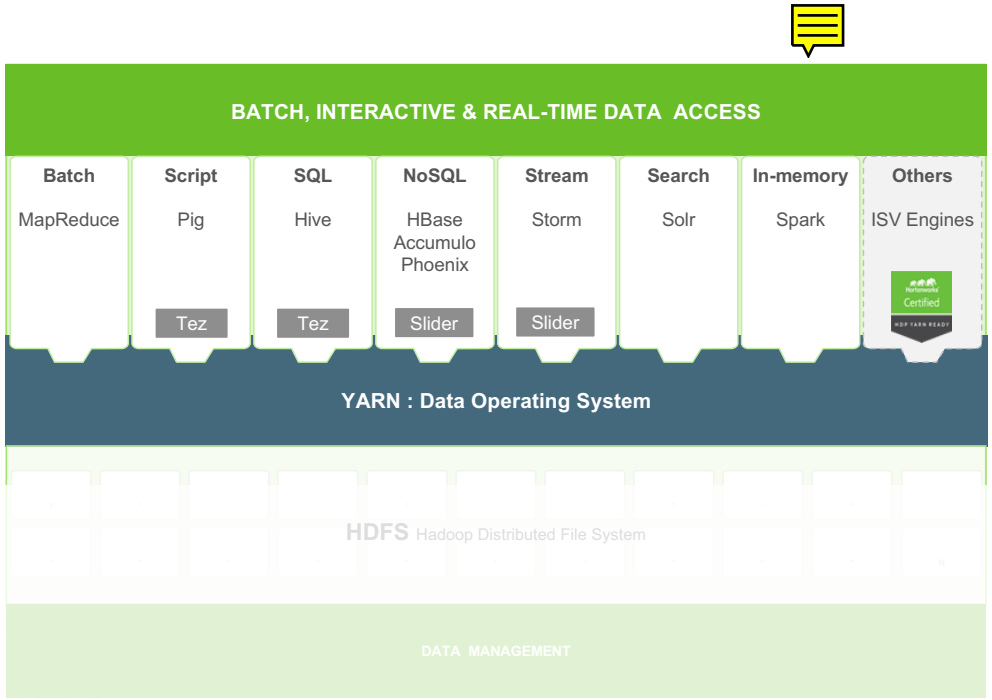
- **YARN: The resource manager for Apache Hadoop**
  - Being able to share resources not only across MapReduce jobs, but also across the other processing frameworks
    - Correspondence (MapReduce -> YARN):
      - Cluster manager -> ResourceManager
      - JobTracker -> ApplicatonMaster
      - TaskTracker -> NodeManager
      - MapReduce job -> Distributed application



Source: IBM

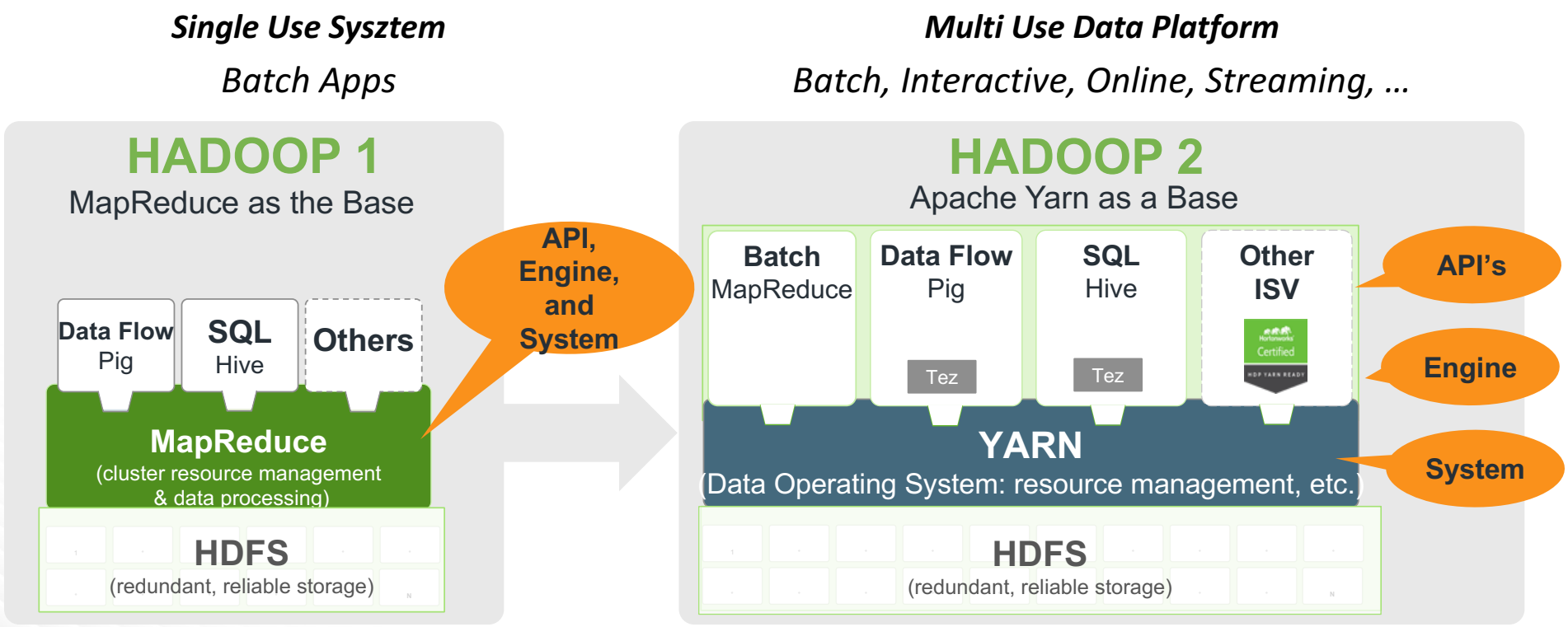
Key Components of Hadoop Stack

- **YARN: Data Operating System**
  - Architectural center of Hadoop
    - Common data platform, many applications
    - Support multi-tenant access & processing
    - Batch, interactive & real-time use cases



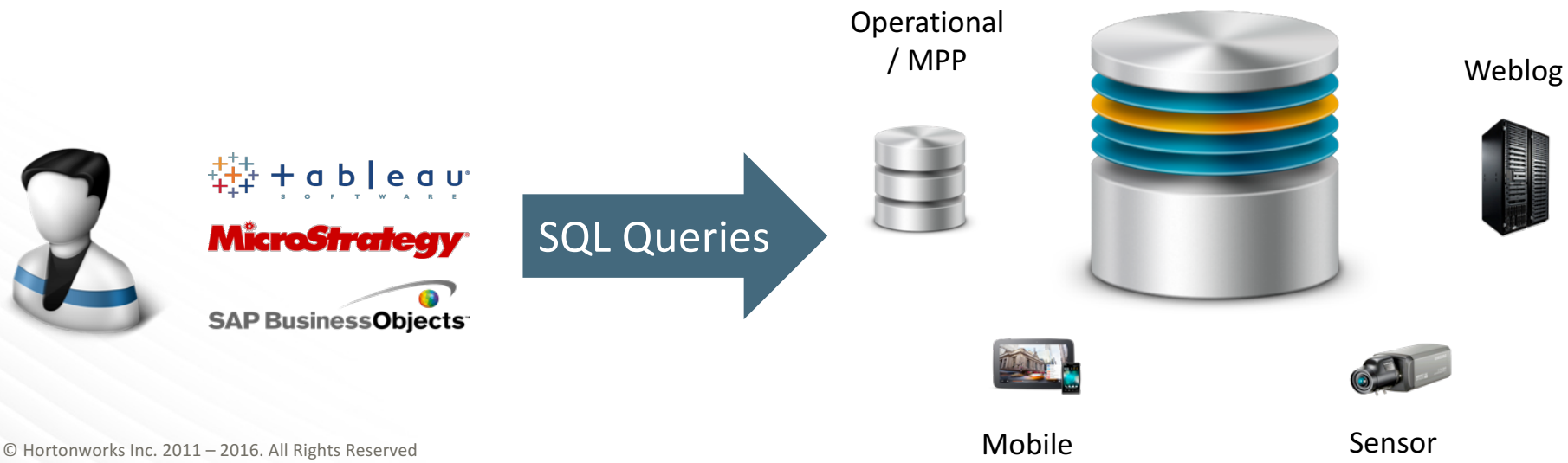
Key Components of Hadoop Stack

- **YARN: Hadoop beyond batch!**
  - A shift from the old to the new



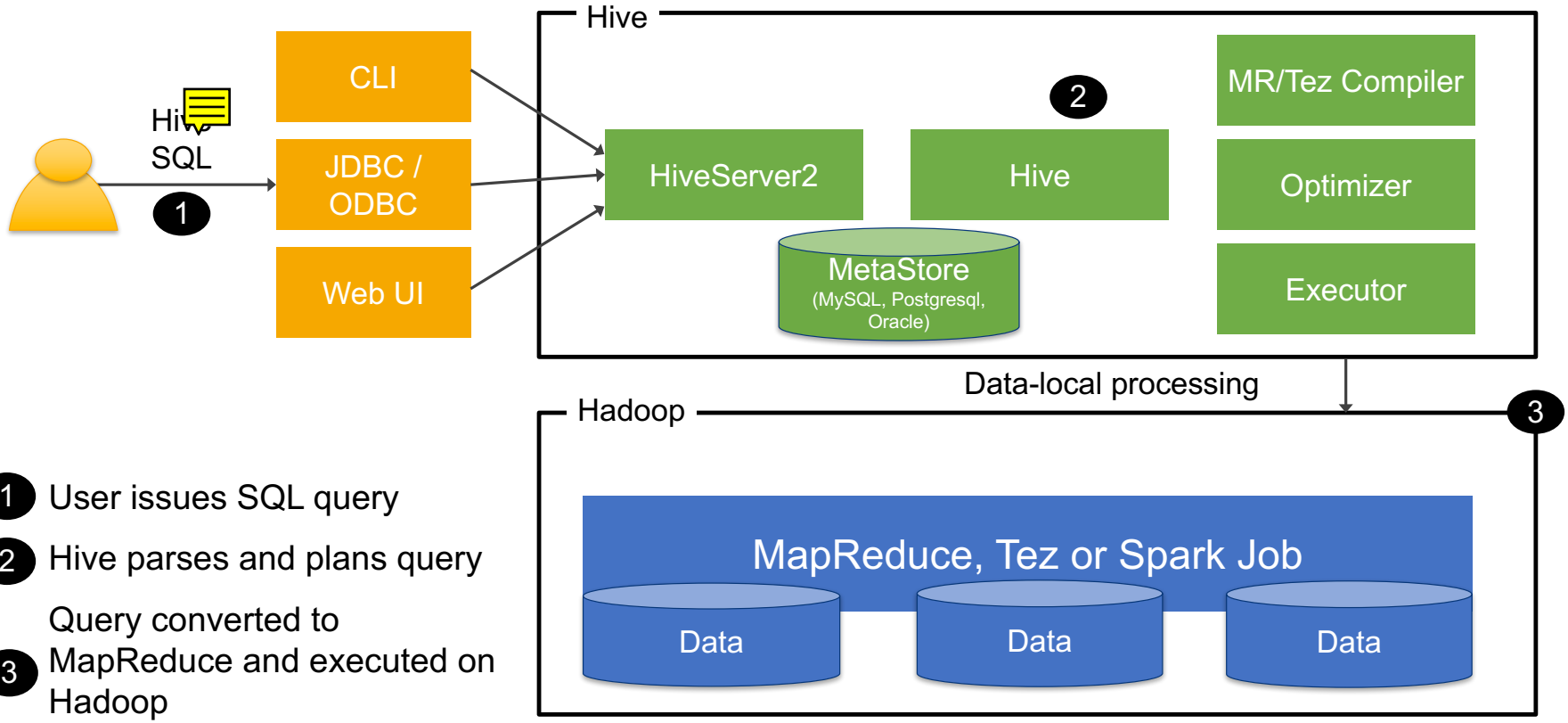
Key Components of Hadoop Stack

- **Apache Hive: SQL in Hadoop**
  - Created by a team at Facebook
  - Provides a standard SQL interface to data stored in Hadoop
  - Quickly find value in raw data files
  - Proven at petabyte scale
  - Compatible with ALL major BI tools such as Tableau, Excel, MicroStrategy, Business Objects, etc…



Key Components of Hadoop Stack

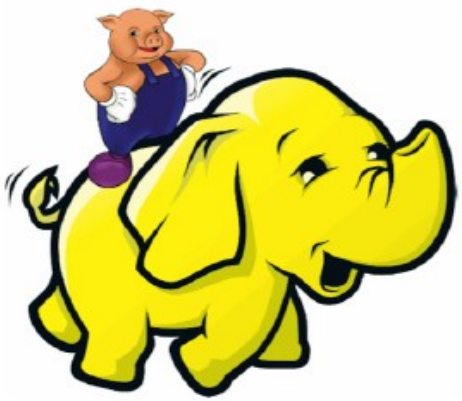
Apache Hive: Architecture



1. User issues SQL query
2. Hive parses and plans query
3. Query converted to MapReduce and executed on Hadoop

Key Components of Hadoop Stack

- **Apache Pig**
  - Data flow engine and scripting language (Pig Latin)
  - Allows you to transform data and datasets
  - Advantages over MapReduce
    - Reduces time to write jobs
    - Community support
    - Piggybank has a significant number of User-Defined Functions (UDF's) to help adoption
    - There are a large number of existing shops using PIG



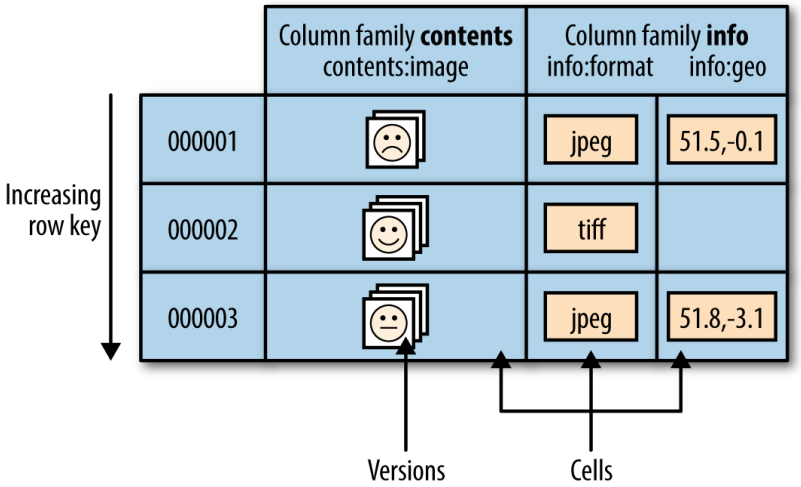
Key Components of Hadoop Stack

- **Apache Pig: Example**
  - Maybe we want to join two datasets, from different sources, on a common value, and want to filter, and sort, and get top 5 sites

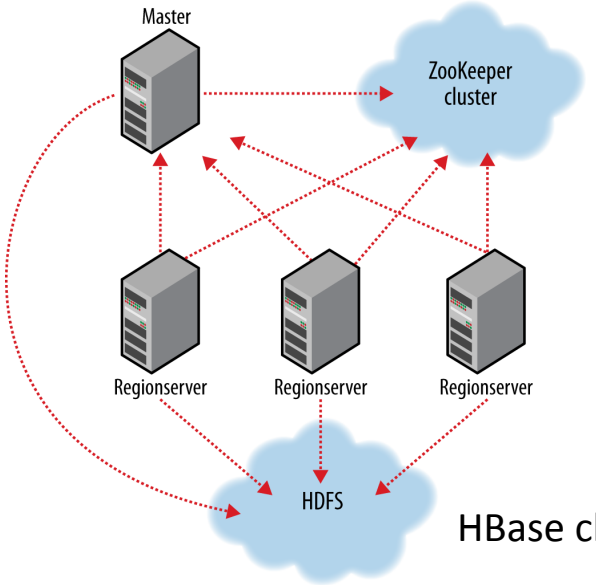
```
users = LOAD 'input/users' USING PigStorage(',')
        AS (name:chararray, age:int);

filtrd = FILTER users BY age >= 18 and age <= 25;

pages = LOAD 'input/pages' USING PigStorage(',')
        AS (user:chararray, url:chararray);

jnd = JOIN filtrd BY name, pages BY user;

grpd = GROUP jnd BY url;

smmd = FOREACH grpd GENERATE group, COUNT(jnd) AS clicks;

srtd = ORDER smmd BY clicks DESC;

top5 = LIMIT srtd 5;

STORE Top5 INTO 'output/top5sites' USING PigStorage(',');
```

Key Components of Hadoop Stack

- **HBase: Distributed column-oriented DB**
  - Provides *real-time* read/write random access to very large datasets
    - Note that the original HDFS only provides append-only storage optimized for batch processing.
  - Used for Facebook messages DB
  - Unlike RDBMS it does not support SQL, but can host very large, sparsely populated tables well on commodity cluster



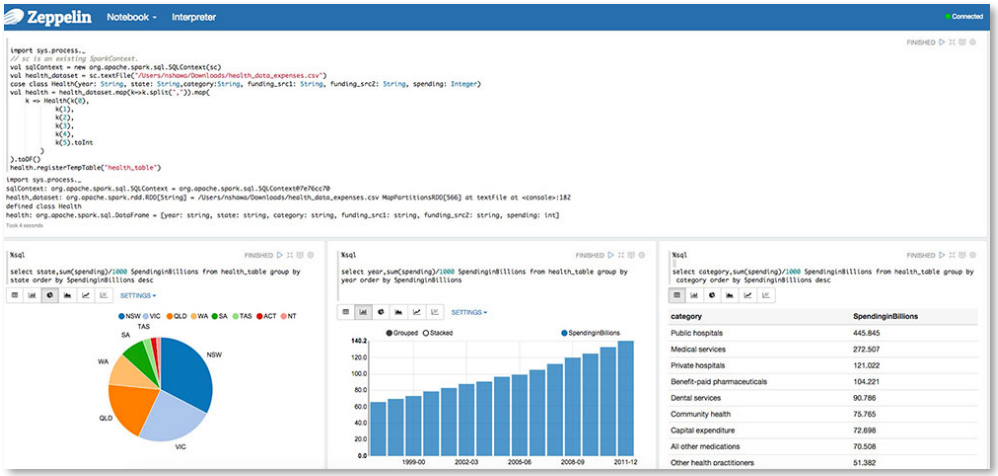
HBase data model (example: photo DB)



HBase cluster members

Key Components of Hadoop Stack

- **Apache Zepplin**
  - Web-based notebook for data engineers, data analysts and data scientists
    - Brings interactive data ingestion, data exploration, visualization, sharing and collaboration features to Hadoop and Spark
  - Modern data science studio
    - Scala with Spark
    - Python with Spark
    - SparkSQL
    - Apache Hive, and more.



Outline

- What is Apache Hadoop?
- Key Components of Hadoop Stack
- **Hadoop Distributed File System (HDFS)**
- HDFS File System Shell Commands
- Apache Spark on HDFS

HDFS

- Data is organized into files and directories
- Files are divided into uniform sized blocks (default 128MB) and distributed across cluster nodes
- HDFS exposes block placement so that computation can be migrated to data

HDFS

- Blocks are replicated (default 3) to handle hardware failure
- Replication for performance and fault tolerance (Rack-Aware placement)
- HDFS keeps checksums of data for corruption detection and recovery

HDFS

- Master-Worker Architecture
- Single NameNode
- Many (Thousands) DataNodes

HDFS Master (NameNode)

- Manages filesystem namespace
- File metadata (i.e. “inode”)
- Mapping inode to list of blocks + locations
- Authorization & Authentication
- Checkpoint & journal namespace changes

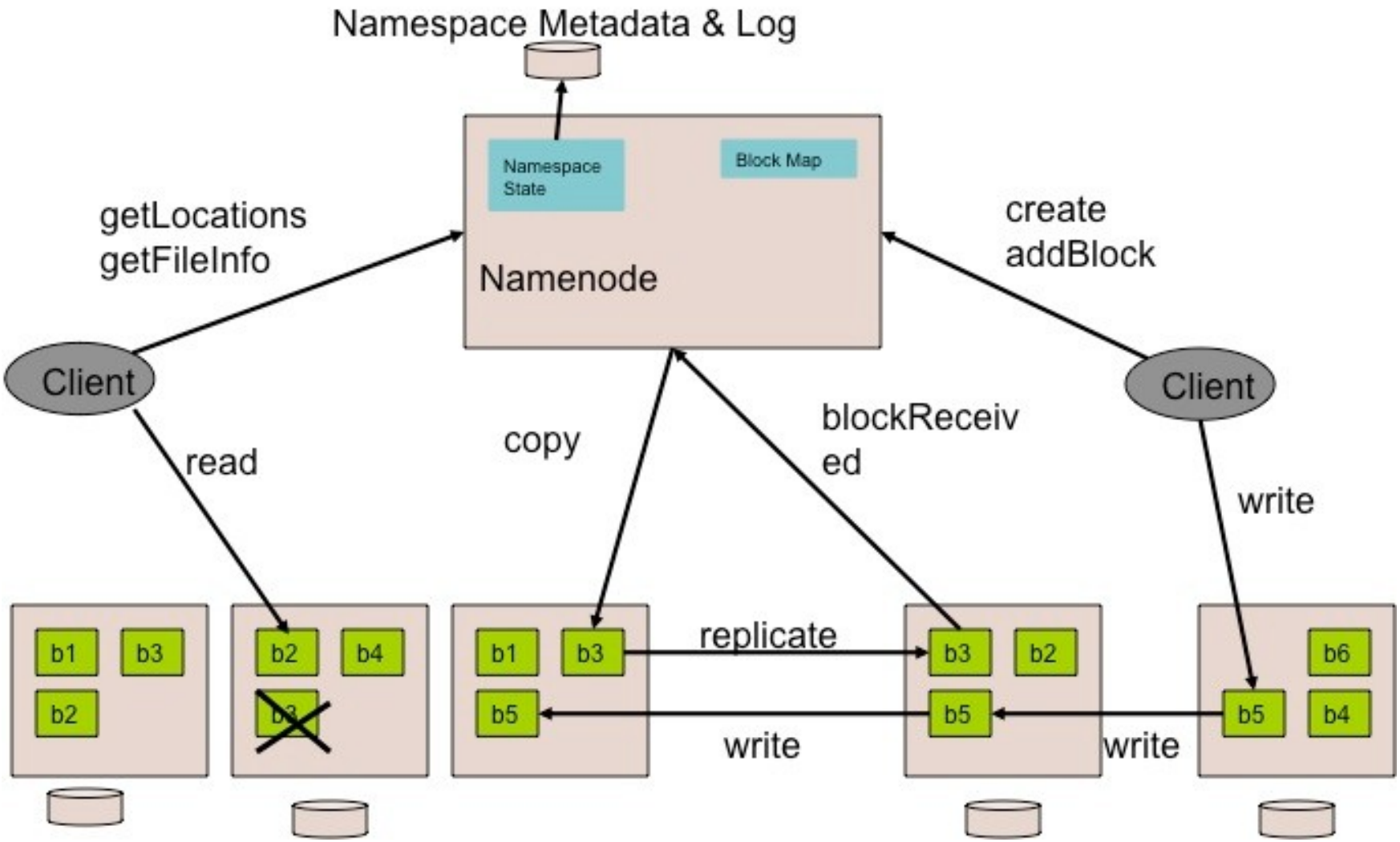
NameNode

- Mapping of datanode to list of blocks
- Monitor datanode health
- Replicate missing blocks
- Keeps ALL namespace in memory
- 60M objects (File/Block) in 16GB

DataNodes

- Handle block storage on multiple volumes & block integrity
- Clients access the blocks directly from data nodes
- Periodically send heartbeats and block reports to NameNode
- Blocks are stored as underlying OS's files

HDFS Architecture



Design of HDFS

- **HDFS is ...**
  - "a filesystem designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware"
- **Three key phrases in details**
  - Very large files
    - “Very large” in this context means files that are hundreds of megabytes, gigabytes, or terabytes in size

Design of HDFS

- **HDFS is ...**
  - "a filesystem designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware"
- **Three key phrases in details**
  - Streaming data access
    - Built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern
    - A dataset is typically generated or copied from source, and then various analyses are performed on that dataset over time
    - Each analysis will involve a large proportion, if not all, of the dataset
    - the time to read the whole dataset is more important than the latency in reading the first record

Design of HDFS

- **HDFS is ...**
  - "a filesystem designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware"
- **Three key phrases in details**
  - Commodity hardware
    - doesn't require expensive, highly reliable hardware
    - designed to run on clusters of commodity hardware (commonly available hardware that can be ob- tained from multiple vendors)
    - the chance of node failure across the cluster is high, at least for large clusters
    - designed to carry on working without a noticeable interruption to the user in the face of such failure

When HDFS Does NOT Work So Well?

- **For the following applications HDFS is NOT a good fit!**
  - Low-latency data access
    - Applications that require low-latency access to data, in the tens of milliseconds range, will not work well with HDFS (HDFS for high throughput, but not for low latency)
    - HBase is a better choice for the application
  - Lots of small files
    - Namenode holds filesystem metadata in memory, the limit to the num- ber of files in a filesystem is governed by the amount of memory on the namenode
    - As a rule of thumb, each file, directory, and block takes about 150 bytes
      - if you had one million files, each taking one block, you would need at least 300 MB of memory!

When HDFS Does NOT Work So Well?

- **For the following applications HDFS is NOT a good fit!**
  - Multiple writers, arbitrary file modifications
    - Files in HDFS may be written to by a single writer
    - Writes are always made at the end of the file, in append-only fashion
    - No support for multiple writers or for modifications at arbitrary offsets in the file

Example: Unigrams

- **Input: Huge text corpus**
  - Wikipedia Articles (40GB uncompressed)
- **Output: List of words sorted in descending order of frequency**

Unigrams

- Implementation in Linux shell (if data fits in one node)

```
$ cat ~/wikipedia.txt | \
sed -e 's/ /\n/g' | grep . | \
sort | \
uniq -c > \
~/frequencies.txt

$ cat ~/frequencies.txt | \  #
cat | \
sort -n -k1,1 -r |  #
cat > \
~/unigrams.txt
```

Unigrams

- **MapReduce for Unigrams: Pseudo-code (Phase #1)**
  - Counting # of occurrences for each word

```
mapper (filename, file-contents):
     for each word in file-
     contents:
          emit (word, 1)

reducer (word, values):
     sum = 0
     for each value in values:
          sum = sum + value
     emit (word, sum)
```

Unigrams

- **MapReduce for Unigrams: Pseudo-code (Phase #2)**
  - Generating a sorted list

```
mapper (word, frequency):
     emit (frequency, word)

reducer (frequency, words):
     for each word in words:
          emit (word, frequency)
```

Unigrams

- **Java Mapper**
  - Just for reference (don't need to understand)

```
public static class MapClass extends MapReduceBase
      implements Mapper<LongWritable, Text, Text, IntWritable> {

        public void map(LongWritable key, Text value,
            OutputCollector<Text, IntWritable> output,
            Reporter reporter) throws IOException {

            String line = value.toString();
            StringTokenizer itr = new StringTokenizer(line);
            while (itr.hasMoreTokens()) {
                  Text word = new Text(itr.nextToken());
                  output.collect(word, new IntWritable(1));
            }
      }
}
```

Unigrams

- **Java Reducer**
  - Just for reference (don't need to understand)

```
public static class Reduce extends MapReduceBase
      implements Reducer<Text, IntWritable, Text, IntWritable> {

      public void reduce(Text key,Iterator<IntWritable> values,
            OutputCollector<Text,IntWritable> output,
            Reporter reporter) throws IOException {

            int sum = 0;
            while (values.hasNext()) {
                  sum += values.next().get();
            }
            output.collect(key, new IntWritable(sum));
      }
}
```

Unigrams

- Driver (like main function)

```
public void run(String inputPath, String outputPath) throws
Exception
{
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("wordcount");
        conf.setMapperClass(MapClass.class);
        conf.setReducerClass(Reduce.class);
        FileInputFormat.addInputPath(conf, new Path(inputPath));
        FileOutputFormat.setOutputPath(conf, new Path(outputPath));
        JobClient.runJob(conf);
}
```

Configuration

- **Unified Mechanism for**
  - Configuring Daemons
  - Runtime environment for Jobs/Tasks
- **Defaults: *-default.xml**
- **Site-Specific: *-site.xml**
- ***final* parameters**

Configuration

- **Example**

```
<configuration>
     <property>
          <name>mapred.job.tracker</name>
          <value>head.server.node.com:9001</value>
     </property>
     <property>
          <name>fs.default.name</name>
          <value>hdfs://head.server.node.com:9000</value>
     </property>
     <property>
     <name>mapred.child.java.opts</name>
     <value>-Xmx512m</value>
     <final>true</final>
     </property>
....
</configuration>
```

Outline

- What is Apache Hadoop?
- Key Components of Hadoop Stack
- Hadoop Distributed File System (HDFS)
- **HDFS File System Shell Commands**
- Apache Spark on HDFS

HDFS File System Shell

- **HDFS File System (FS) shell**
  - includes various shell-like commands that directly interact with HDFS
  - also support other file systems like local FS, HFTP FS, S3 FS, and others.
  - is invoked by the following command:

```
$ bin/hadoop fs ...
```

  - If HDFS is being used, the following command does the same.

```
$ bin/hdfs dfs ...
```

  - If you need more information about supported shell commands:
    - Type `$ bin/hdfs dfs -help`
    - Look up HDFS User Guide: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/FileSystemShell.html

HDFS File System Shell Commands

- **ls: list files**
  - Usage: `hadoop fs -ls [-d] [-h] [-R] [<path> ...]`
  - Desc: List the contents that match the specified file pattern. If path is not specified, the contents of /user/<currentUser> will be listed. -R recursively lists subdirectories.
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -ls /user/hadoop
Found 1 items
-rw-r--r--   1 ubuntu supergroup          14 2017-10-17 16:40 /
user/hadoop/file1
```

HDFS File System Shell Commands

- **cat: display file content**
  - Usage: `hadoop fs -cat [-ignoreCrc] <src> ...`
  - Desc: Fetch all files that match the file pattern <src> and display their content on stdout
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -cat /user/hadoop/*
Hello world!
```

HDFS File System Shell Commands

- **mkdir: create a directory**
  - Usage: `hadoop fs -mkdir [-p] <path> ...`
  - Desc: Create a directory in specified location. (-p Do nto fail if the directory already exists).
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -mkdir /user/hadoop/mydir
$ bin/hadoop fs -ls /user/hadoop
Found 2 items
-rw-r--r--   1 ubuntu supergroup         14 2017-10-17 16:40 /
user/hadoop/file1
drwxr-xr-x   - ubuntu supergroup          0 2017-10-17 16:54 /
user/hadoop/mydir
```

HDFS File System Shell Commands

- **rm: delete files**
  - Usage: `hadoop fs –rm [-f] [-r|-R] [-skipTrash] <src> ...`
  - Desc: Delete all files that match the specified file pattern. Equivalent to the Unix command "rm <src>". –r (or –R) recursively deletes directories.
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -ls /user/hadoop
Found 2 items
-rw-r--r--   1 ubuntu supergroup         14 2017-10-17 16:40 /user/hadoop/file1
drwxr-xr-x   - ubuntu supergroup          0 2017-10-17 16:54 /user/hadoop/mydir
$ bin/hadoop fs -rm -r /user/hadoop/mydir
17/10/17 16:59:11 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion
interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /user/hadoop/mydir
$ bin/hadoop fs -ls /user/hadoop
Found 1 items
-rw-r--r--   1 ubuntu supergroup         14 2017-10-17 16:40 /user/hadoop/file1
```

HDFS File System Shell Commands

- **put (or copyFromLocal): copy files from local file system**
  - Usage: `hadoop fs -put [-f] [-p] [-l] <localsrc> ... <dst>`
  - Desc: Copy files from the local file system into fs. Copying fails if the file already exists, unless the -f flag is given.
  - Example (assume we're in $HADOOP_HOME directory)

```
# hello.txt and world.txt are in local file system
$ bin/hadoop fs -put hello.txt world.txt /user/hadoop
$ bin/hadoop fs -ls /user/hadoop
Found 3 items
-rw-r--r--   1 ubuntu supergroup          14 2017-10-17 16:40 /
user/hadoop/file1
-rw-r--r--   1 ubuntu supergroup          14 2017-10-17 17:09 /
user/hadoop/hello.txt
-rw-r--r--   1 ubuntu supergroup          28 2017-10-17 17:09 /
user/hadoop/world.txt
```

HDFS File System Shell Commands

- **get (or copyToLocal): copy files from local file system**
  - Usage: `hadoop fs -get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>`
  - Desc: Copy files that match the file pattern <src> to the local name. <src> is kept. When copying multiple files, the destination must be a directory.
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -get /user/hadoop/hello.txt hello.copy.txt
$ md5sum hello.txt hello.copy.txt  # compare two versions
5c06f4b1a10b425fb707d16bb4d5d9a9  hello.txt
5c06f4b1a10b425fb707d16bb4d5d9a9  hello.copy.txt
```

HDFS File System Shell Commands

- **getmerge: get files and merge them into local file system**
  - Usage: `hadoop fs -getmerge [-nl] <src> <localdst>`
  - Desc: Get all the files in the directories that match the source file pattern and merge and sort them to only one file on local fs. <src> is kept. -nl adds a newline character at the end of each file
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -getmerge /user/hadoop/*.txt helloworld.txt
$ cat helloworld.txt   # from local file system
Hello world!

To the world: Hello there!
```

HDFS File System Shell Commands

- **appendToFile: appends the contents of local files to dst file**
  - Usage: `hadoop fs –appendToFile <localsrc> ... <dst>`
  - Desc: Appends the contents of all the given local files to the given dst file. The dst file will be created if it does not exist. If <localSrc> is -, then the input is read from stdin.
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -appendToFile world.txt /user/hadoop/hello.txt
$ bin/hadoop fs -cat hello.txt
Hello world!

To the world: Hello there!
```

HDFS File System Shell Commands

- **mv: move files**
  - Usage: `hadoop fs -mv <src> ... <dst>`
  - Desc: Move files that match the specified file pattern <src> to a destination <dst>. When moving multiple files, the destination must be a directory.
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -mv /user/hadoop/hello.txt /user/hadoop/olleh.txt
$ bin/hadoop fs -ls /user/hadoop
Found 3 items
-rw-r--r--   1 ubuntu supergroup          14 2017-10-17 16:40 /
user/hadoop/file1
-rw-r--r--   1 ubuntu supergroup          14 2017-10-17 17:09 /
user/hadoop/olleh.txt
-rw-r--r--   1 ubuntu supergroup          28 2017-10-17 17:09 /
user/hadoop/world.txt
```

HDFS File System Shell Commands

- **chown: change ownership**
  - Usage: `hadoop fs –chown [-R] [OWNER][:[GROUP]] PATH...`
  - Desc: Changes owner and group of a file. This is similar to the shell's chown command with a few exceptions. -R modifies the files recursively.
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -ls /user/hadoop
Found 2 items
-rw-r--r--   1 ubuntu supergroup          14 2017-10-17 17:09 /
user/hadoop/hello.txt
-rw-r--r--   1 ubuntu supergroup          28 2017-10-17 17:09 /
user/hadoop/world.txt
$ bin/hadoop fs -chown jaewooklee /user/hadoop/*
$ bin/hadoop fs -ls /user/hadoop
Found 2 items
-rw-r--r--   1 jaewooklee supergroup          14 2017-10-17
17:09 /user/hadoop/hello.txt
-rw-r--r--   1 jaewooklee supergroup          28 2017-10-17
17:09 /user/hadoop/world.txt
```

HDFS File System Shell Commands

- **chmod: change permissions**
  - Usage: `hadoop fs –chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...`
  - Desc: Changes permissions of a file. This works similar to the shell's chmod command with a few exceptions. –R modifies the files recursively.
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -chmod -R g+rwx /user/hadoop
$ bin/hadoop fs -ls /user/hadoop
Found 2 items
-rw-rwxr--   1 jaewooklee supergroup         42 2017-10-18
00:09 /user/hadoop/hello.txt
-rw-rwxr--   1 jaewooklee supergroup         28 2017-10-17
17:09 /user/hadoop/world.txt
```

HDFS File System Shell Commands

- **expunge: empty trash**
  - Usage: hadoop fs –expunge
  - Desc: Delete files from the trash that are older than the retention threshold
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -expunge
17/10/17 17:33:57 INFO fs.TrashPolicyDefault: Namenode trash
configuration: Deletion interval = 0 minutes, Emptier interval =
0 minutes.
$ # trash emptied
```

HDFS File System Shell Commands

- **du: display lentgh of files**

  - Usage: `hadoop fs –du [-s] [-h] <path> ...`
  - Desc: Show the amount of space, in bytes, used by the files that match the specified file pattern. The output is in the form: size – name (path)
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -du /user/hadoop
14   /user/hadoop/file1
14   /user/hadoop/olleh.txt
28   /user/hadoop/world.txt
```

HDFS File System Shell Commands

- **setrep: change replication factor of file**
  - Usage: `hadoop fs –setrep [-R] [-w] <rep> <path> ...`
  - Desc: Set the replication level of a file. If <path> is a directory then the command recursively changes the replication factor of all files under the directory tree rooted at <path>. –w waits until the replication to complete (this potentially takes a very long time).
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -setrep -R 3 /user/hadoop
Replication 3 set: /user/hadoop/hello.txt
Replication 3 set: /user/hadoop/world.txt
$ bin/hadoop fs -ls /user/hadoop
Found 2 items
-rw-r--r--   3 ubuntu supergroup         42 2017-10-18 00:19 /
user/hadoop/hello.txt
-rw-r--r--   3 ubuntu supergroup         28 2017-10-17 17:09 /
user/hadoop/world.txt
```

HDFS File System Shell Commands

- **stat: show stat information**
  - Usage: `hadoop fs -stat [format] <path> ...`
  - Desc: Print statistics about the file/directory at <path> in the specified format. Format accepts filesize in blocks (%b), type (%F), group name of owner (%g), name (%n), block size (%o), replication (%r), user name of owner (%u), modification date (%y, %Y).
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -stat %b::%F::%g::%n::%o::%r::%y /user/hadoop/
world.txt
28::regular file::supergroup::world.txt:134217728::3::2017-10-17
17:09:56
```

HDFS File System Shell Commands

- **tail: display last part to stdout**
  - Usage: `hadoop fs –tail [-f] <file>`
  - Desc: Show the last 1KB of the file. -f shows appended data as the file grows.
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -tail /user/hadoop/hello.txt
Hello world!

To the world: Hello there!
```

HDFS File System Shell Commands

- **test: check attribute of file/dir**
  - Usage: `hadoop fs -test -[defsz] <path>`
  - Desc: Answer various questions about <path>, with result via exit status. (-d: return 0 if <path> is a directory, -e: return 0 if <path> exists, and so on.)
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -test -e /user/hadoop/hello.txt  # returns 0
$ bin/hadoop fs -test -e /user/hadoop/hi.txt     # returns 1
```

HDFS File System Shell Commands

- **text: outputs file in text format**
  - Usage: `hadoop fs -text [-ignoreCrc] <src> ...`
  - Desc: Takes a source file and outputs the file in text format. The allowed formats are zip and TextRecordInputStream and Avro.
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -text /user/hadoop/hello.zip
P|RKY???"*     hello.txtUT    ???Y???Yux
                                        ???H???W(?/?IQ??
?W(?H?p?<?r@??T?P|RKY???"*    ??hello.txtUT???Yux
                                              ??PKOeu
```

HDFS File System Shell Commands

- **touchz: create a zero-length file**
  - Usage: `hadoop fs –touchz <path> ...`
  - Desc: Creates a file of zero length at <path> with current time as the timestamp of that <path>. An error is returned if the file exists with non-zero length.
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -touchz /user/hadoop/zerolengh.txt
$ bin/hadoop fs -ls /user/hadoop
Found 3 items
-rw-r--r--   3 ubuntu supergroup         42 2017-10-18 00:19 /
user/hadoop/hello.txt
-rw-r--r--   3 ubuntu supergroup         28 2017-10-17 17:09 /
user/hadoop/world.txt
-rw-r--r--   1 ubuntu supergroup          0 2017-10-18 00:56 /
user/hadoop/zerolengh.txt
```

HDFS File System Shell Commands

- **count: count files, dirs, and bytes**
  - Usage: `hadoop fs –count [-q] [-h] <path> ...`
  - Desc: Count the number of directories, files and bytes under the paths that match the specified file pattern. The output columns are:
    DIR_COUNT FILE_COUNT CONTENT_SIZE FILE_NAME or
    (with –q flag) QUOTA REMAINING_QUOTA SPACE_QUOTA 
    REMAINING_SPACE_QUOTA DIR_COUNT FILE_COUNT CONTENT_SIZE FILE_NAME
  - Example (assume we're in $HADOOP_HOME directory)

```
$ bin/hadoop fs -count -q /user/hadoop
 none                 inf              none                 inf
1               3                   70 /user/hadoop
```

HDFS Commands

- **fsck: File system check**
  - Runs the HDFS filesystem checking utility

```
% hdfs fsck /
Connecting to namenode via http://localhost:50070/fsck?ugi=ubuntu&path=%2F
FSCK started by ubuntu (auth:SIMPLE) from /127.0.0.1 for path / at Tue Oct 17
17:44:22 UTC 2017
...Status: HEALTHY
 Total size:        56 B
 Total dirs:        3
 Total files:       3
 Total symlinks:              0
 Total blocks (validated):    3 (avg. block size 18 B)
 Minimally replicated blocks: 3 (100.0 %)
 Over-replicated blocks:      0 (0.0 %)
 Under-replicated blocks:     0 (0.0 %)
 Mis-replicated blocks:               0 (0.0 %)
 Default replication factor:  1
 Average block replication:   1.0
 Corrupt blocks:              0
 Missing replicas:            0 (0.0 %)
 Number of data-nodes:                1
 Number of racks:             1
FSCK ended at Tue Oct 17 17:44:22 UTC 2017 in 3 milliseconds

The filesystem under path '/' is HEALTHY
```

HDFS Commands

- **distcp: distributed copy**
  - Usage: hadoop distcp [options] <srcurl> <desturl>
  - It is used for large inter/intra-cluster copying. It uses MapReduce to effect its distribution, error handling and recovery, and reporting. It expands a list of files and directories into input to map tasks, each of which will copy a partition of the files specified in the source list.
  - Reference: https://hadoop.apache.org/docs/stable/hadoop-distcp/DistCp.html

```
$ bin/hadoop distcp hdfs://nn1:8020/foo/bar hdfs://nn2:8020/bar/foo
```

Outline

- What is Apache Hadoop?
- Key Components of Hadoop Stack
- Hadoop Distributed File System (HDFS)
- HDFS File System Shell Commands
- **Apache Spark on HDFS**

Collocating Spark and HDFS

- **Spark works well with HDFS.**
  - Gives applications a chance to achieve data locality for HDFS running on the same machines
- **Using Spark with HDFS is simple:**
  - Just specifying `hdfs://master:port/path` for your input and output suffices.
- **Caveat – version mismatch**
  - Spark and HDFS (Hadoop) versions should match!
  - The HDFS protocol changes across Hadoop versions.
  - If you build Spark from source, you can specify the Hadoop version number (different from the default version) by setting SPARK_HADOOP_VERSION environment variable.

Accessing HDFS from Spark

- **Example: Count the total words in text files**

```
1 >>> text_file = sc.textFile(“hdfs://localhost:9000/input”)
2 >>> counts = text_file.flatMap(lambda line: line.split(“ “)) \
3 ...              .count()
4 [Stage 0:>>>>>>>>>>>>>>>                                  (1 + 1) / 3]
5 >>>
```

- Line 1 creates an input RDD from an HDFS file (named "input")
- "hdfs://localhost:9000" is the file system URI (Pseudodistributed mode)
- The RDD will also be stored in a distributed manner.
- You will actually run this code tomorrow!

Summary

In this lecture we covered the following topics:

- What is Apache Hadoop?
- Key components constituting Apache Hadoop: It's much more than just MapReduce+HDFS!
- Basic eesign Hadoop Distributed File System (HDFS)
- HDFS file system shell commands
- Running example of using HDFS on Spark