

Hadoop Data Warehousing with Hive

Downloads!!

<http://bit.ly/QyvUyA>



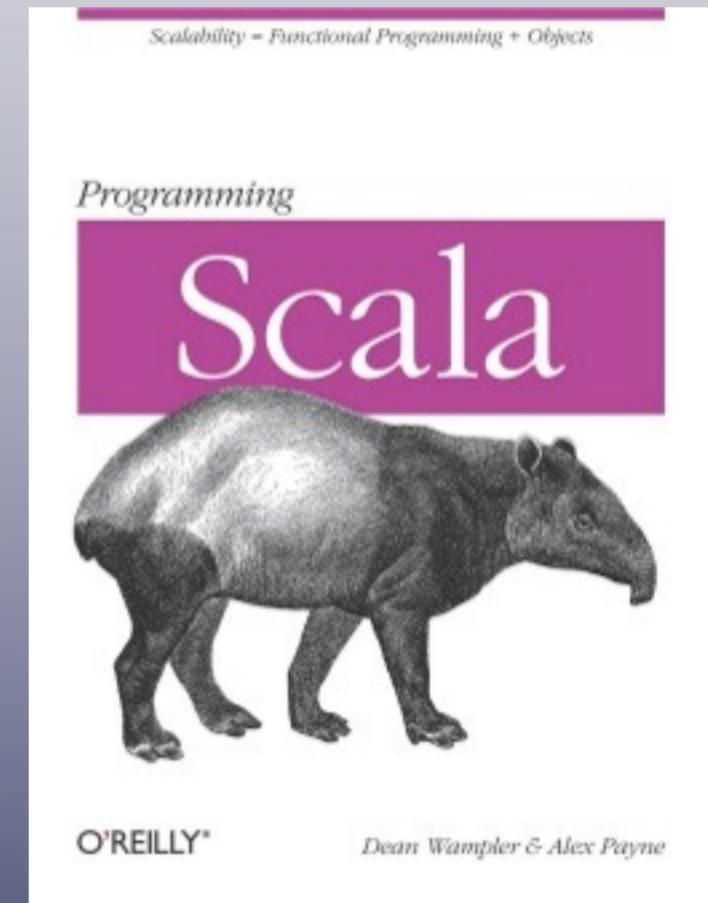
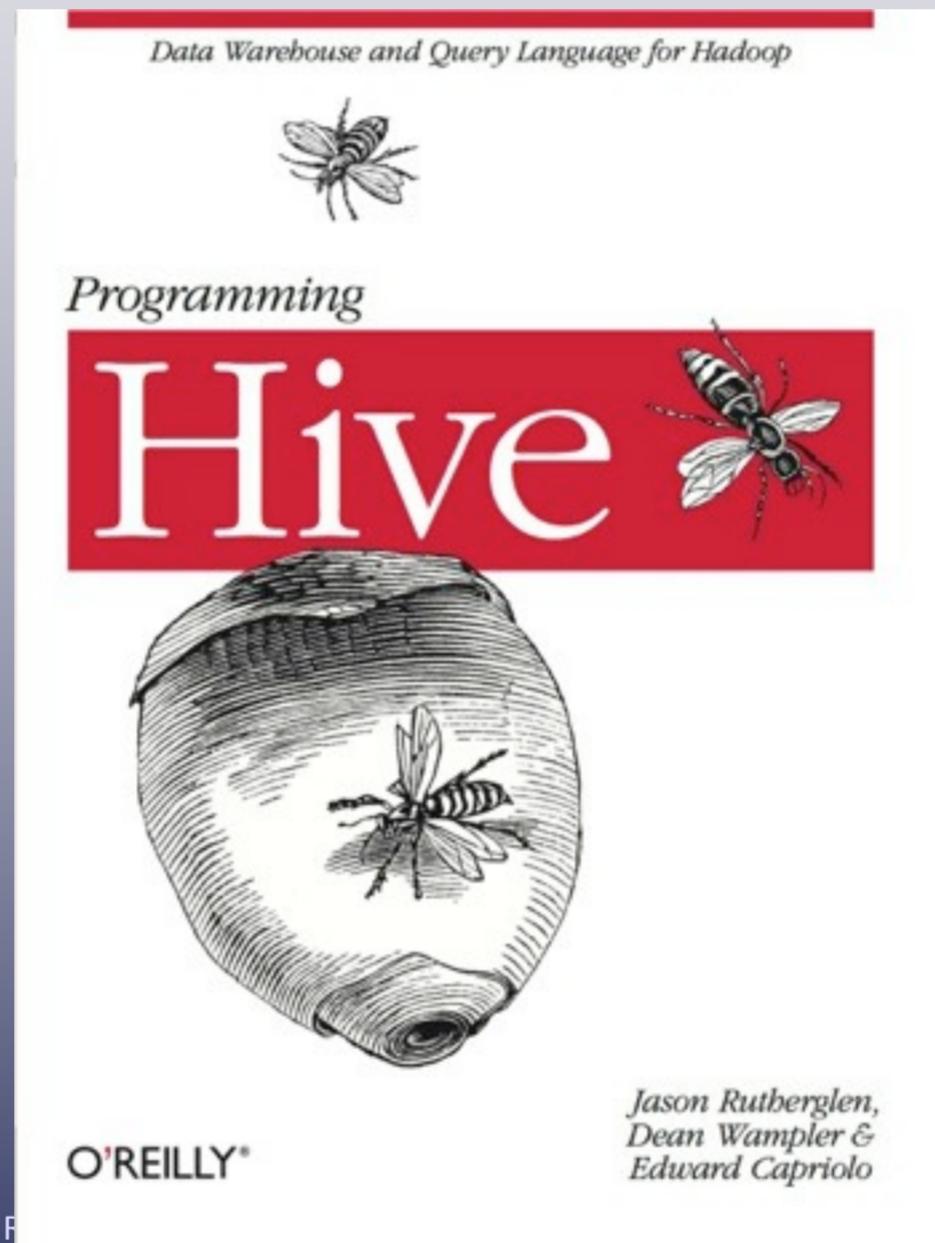
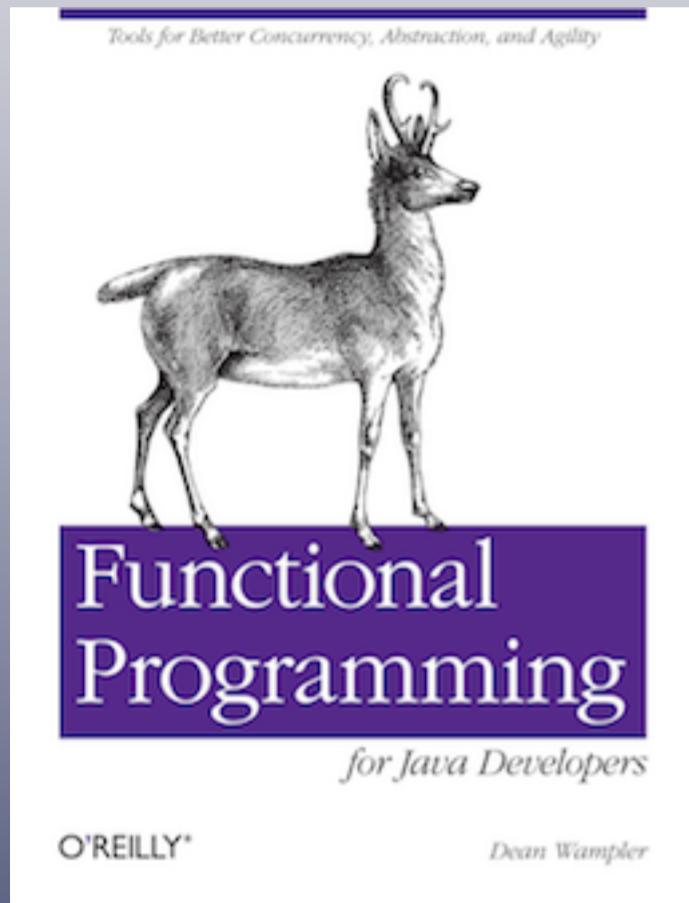
Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

Think Big Analytics provides Big Data consulting and training. This tutorial is based on our Hive courseware.

The bit.ly link takes you to the README for the exercises, etc.: <https://s3.amazonaws.com/thinkbigacademy/StrataHW2012/Hive-Tutorial/README.html>

dean.wampler@thinkbiganalytics.com



Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

My books. The latest, Programming Hive, is most relevant to this tutorial, of course.

Why *Hive*?

You've missed
your old friends...

SQL Databases



Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

If you've been in the Hadoop world a while, you might be missing the richness and maturity of SQL databases...

*Hadoop is
not a database,
but you can use SQL
to ask it *questions*!*

Hadoop: Designed for *Big Data*



Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

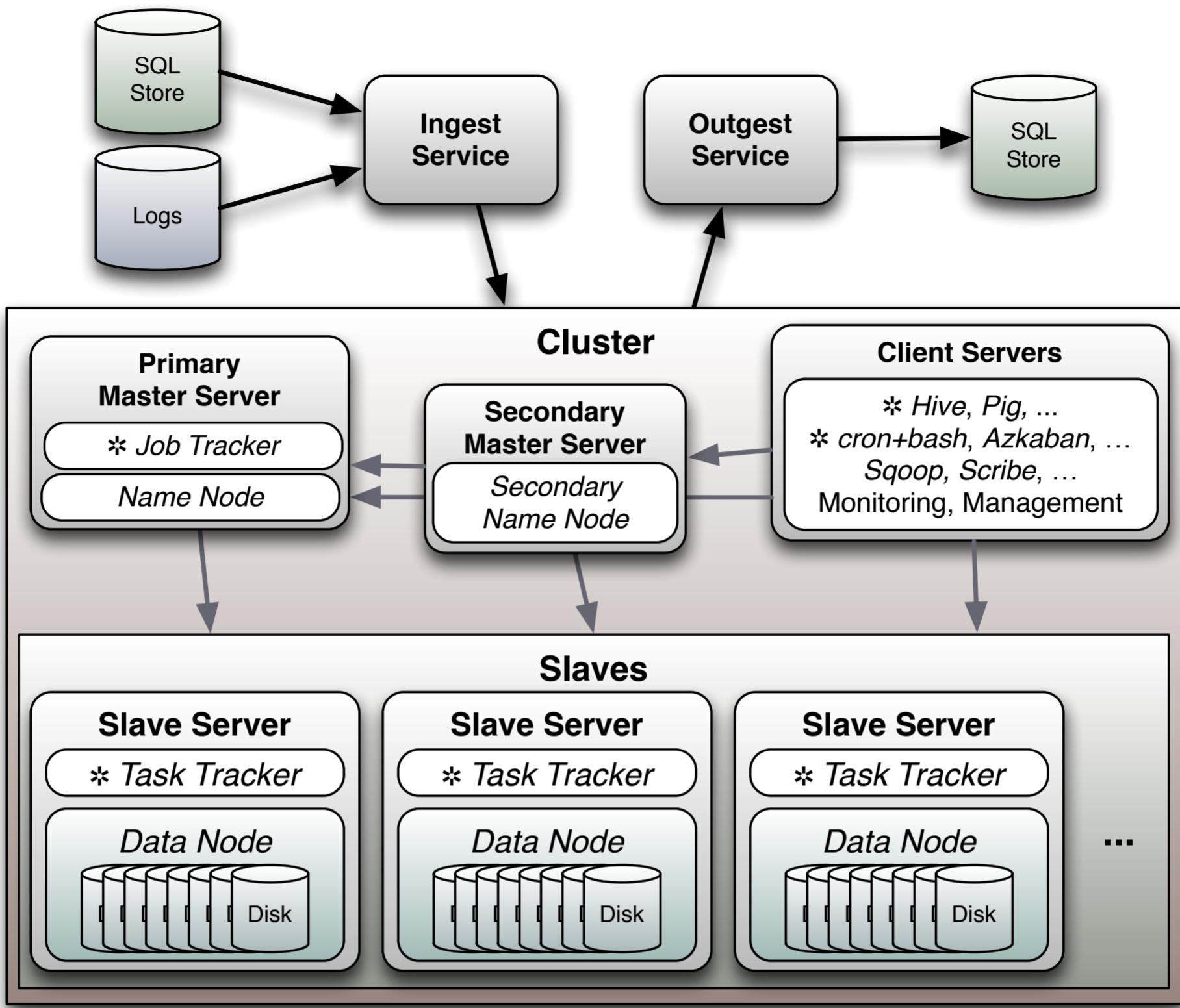
Tuesday, October 23, 12

A brief overview of Hadoop, so we're all on the “same page”.

Big Data: too big for *traditional* tools.

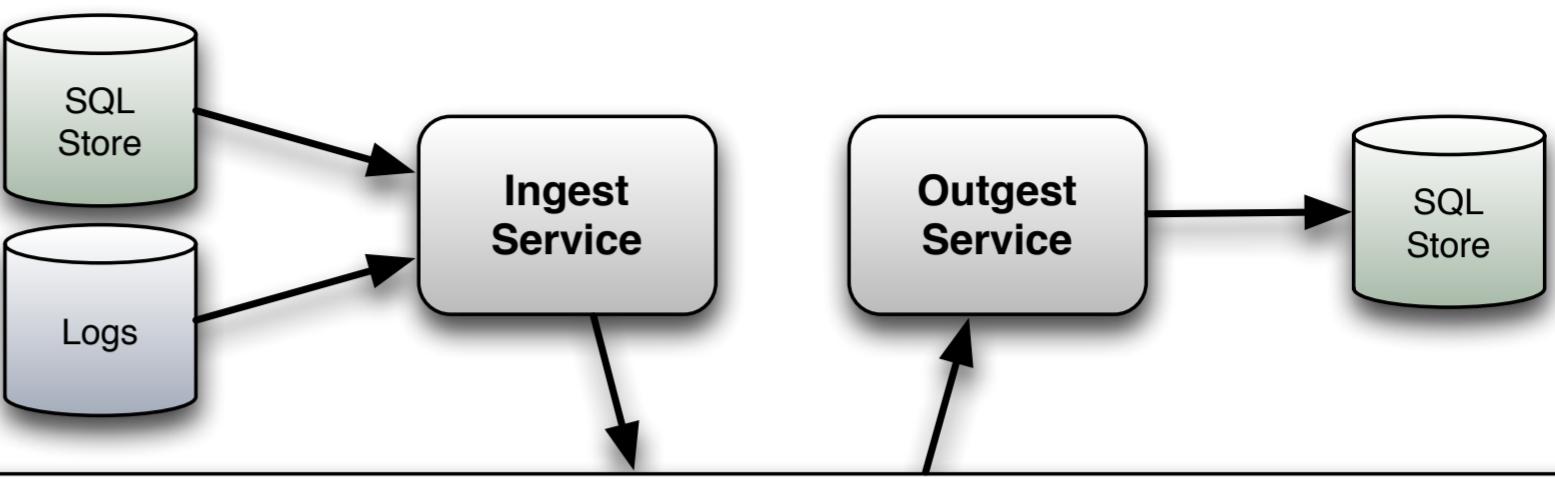
*So, scale horizontally,
for storage and CPU cycles.*

Hadoop Cluster

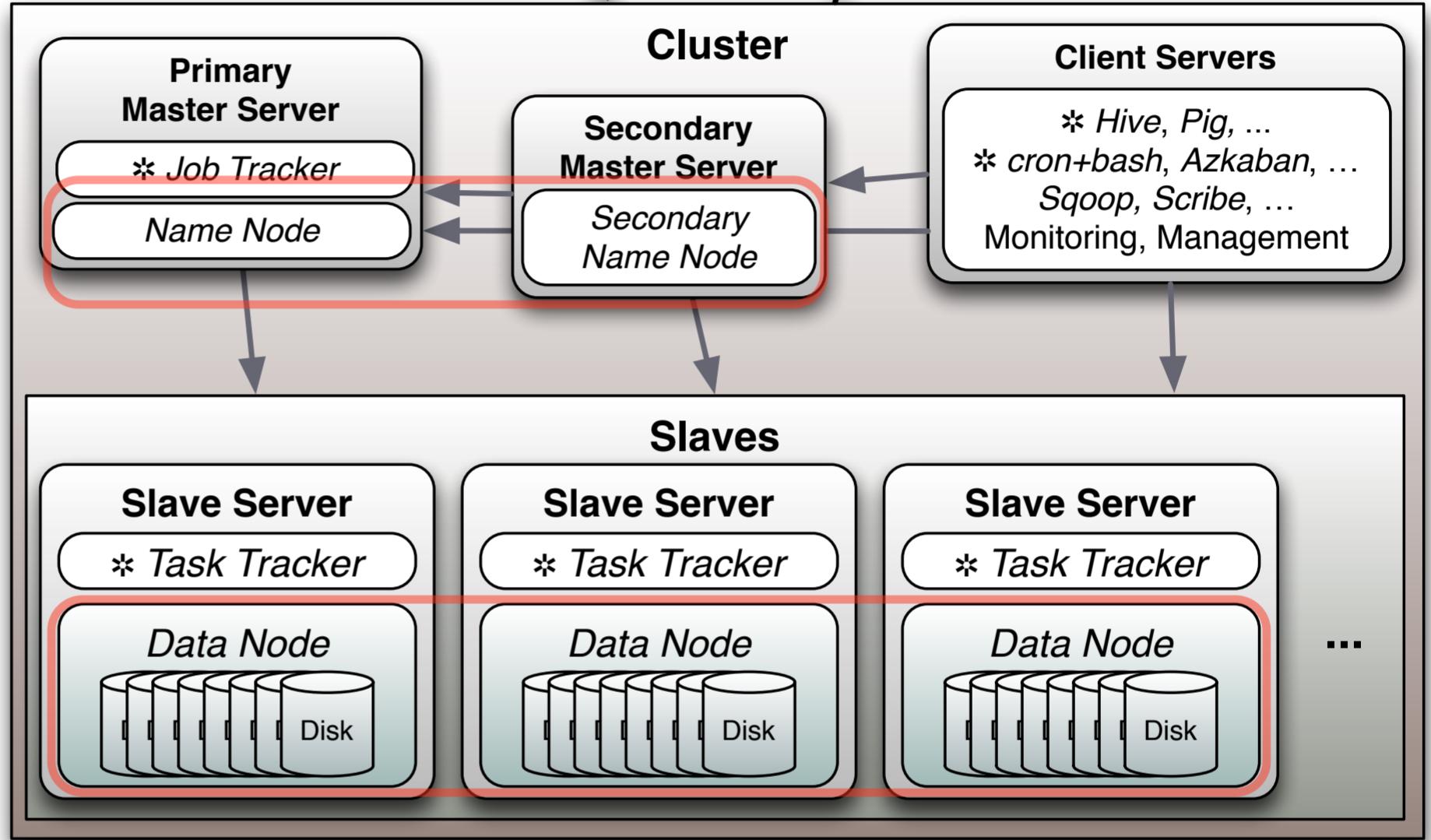


Here is a typical cluster configuration. (It would be slightly different for an Amazon Elastic MapReduce cluster, like the ones we are using for the exercises.) I marked the processes involved in running MapReduce (data crunching) jobs with a “gear” (okay, it’s an asterisk...). I also used italics for actual process names (although there may be subprocesses not shown).

Hadoop Distributed FS



- Services
 - Name Node
 - Secondary NN
 - Data Nodes
 - 64MB blocks
 - 3x replication



Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

At the bottom is the Hadoop Distributed File System. It sits on top of the native file system and uses 64MB blocks (default) to store large streams of data, so scanning the drives is very fast. Not so good, though, for small files! Also, since you'll regularly lose drives when you have 1000s of them in a large cluster, each block is replicated 3 times (default), with each block on a separate server.

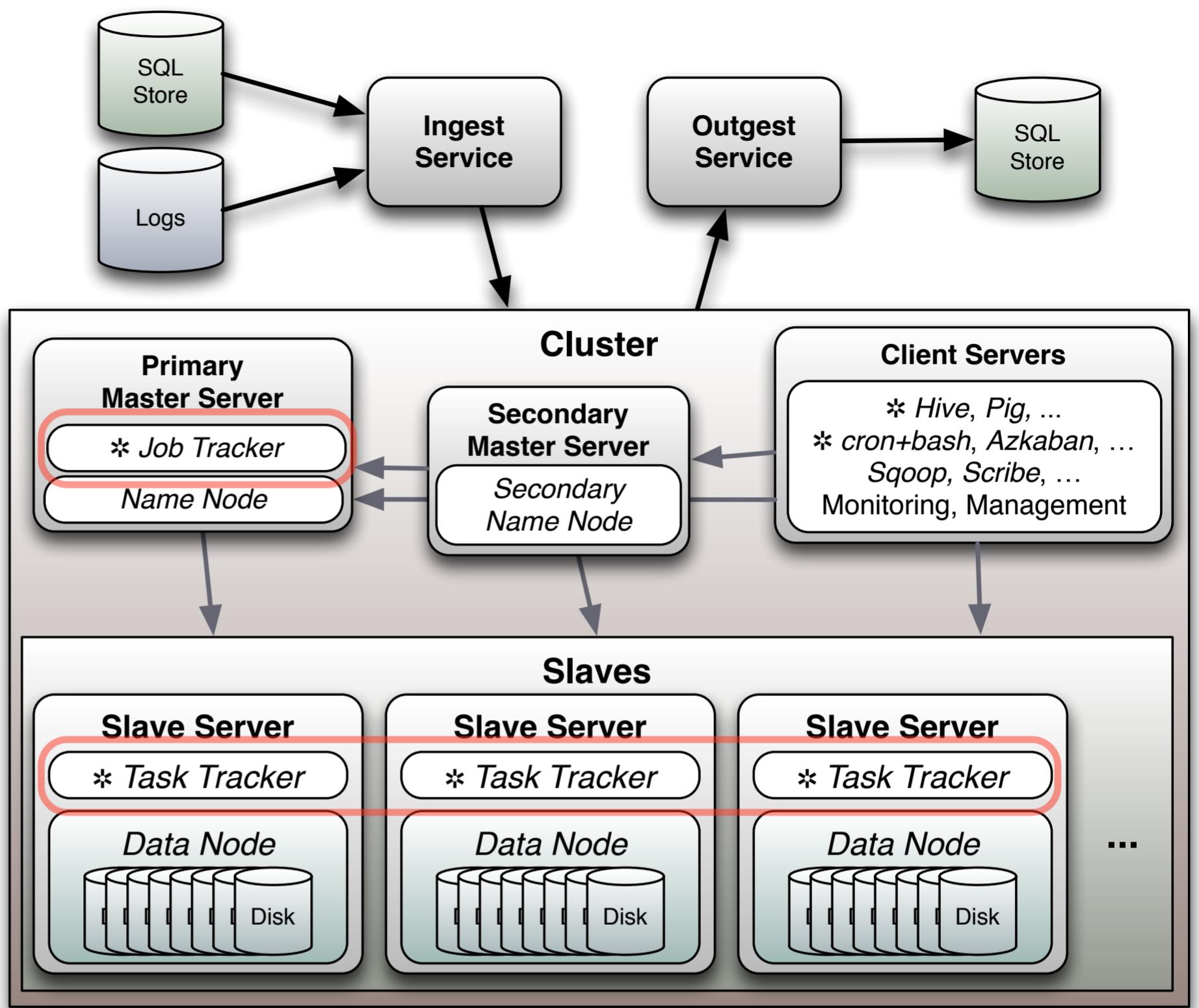
The Name Node maintains the metadata for the whole filesystem.

The Secondary Name Node is a misnomer and it is being replaced in a forthcoming release of Hadoop. It is really a housekeeping service that offloads work from the name node, namely the persisting of in-memory metadata information. We'll investigate further later shortly.

Each Data Node knows how it stores its blocks (e.g., native file names...).

Jobs and Tasks

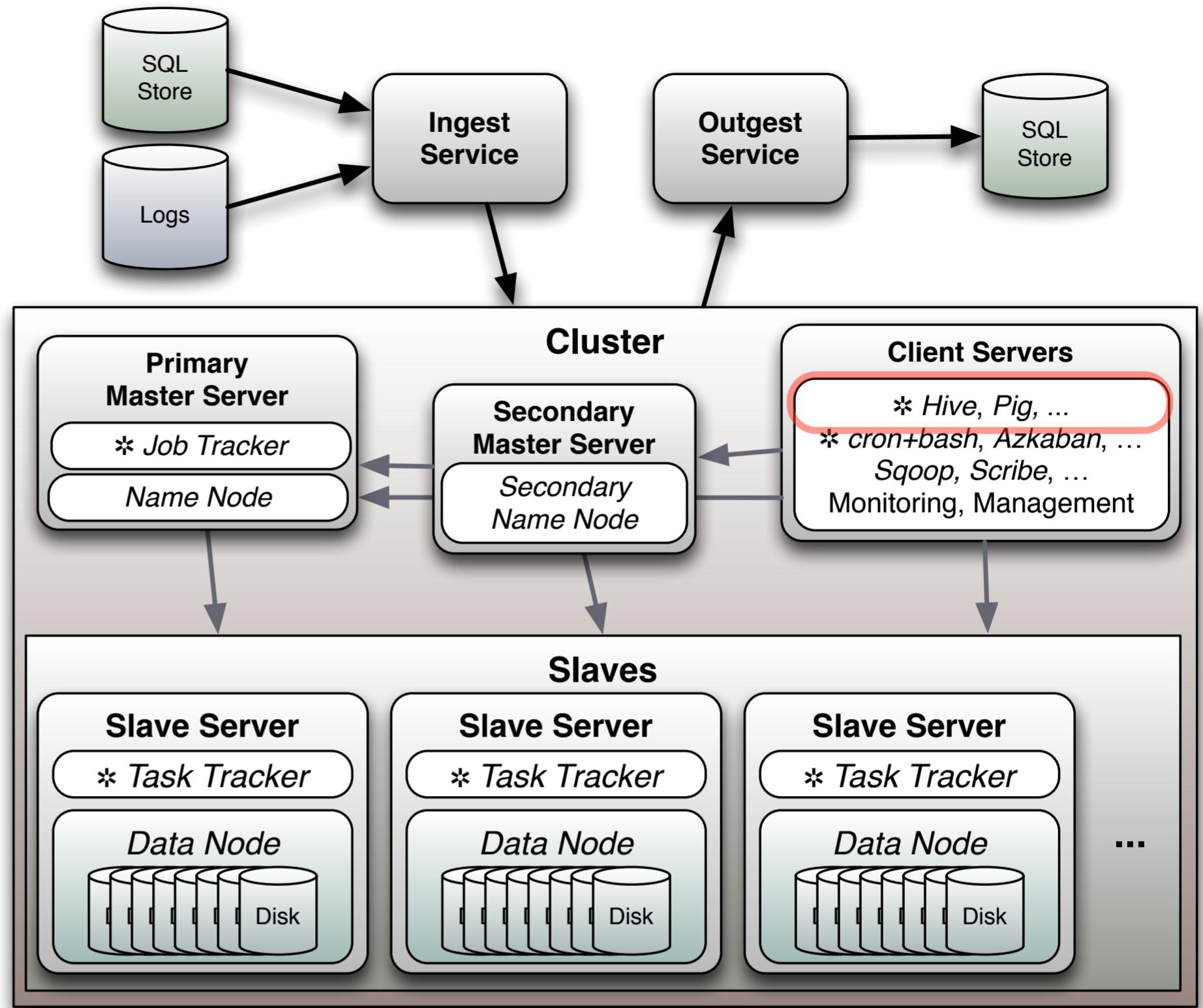
- Services
- *Job Tracker*
- *Task Trackers*



“Jobs” are controlled by the Job Tracker “master” process. It sends “tasks” to individual Task Trackers on the “slave” nodes. It tries to send a task to the server that already has the data for that task, to minimize streaming lots of data over the network.
The Job Tracker does the bookkeeping to ensure that every task completes successfully. It will restart failed tasks (for whatever reason) and it can even restart tasks that appear hung, but not yet failed.

Creating Apps

- Java API
- Hive
- Pig
- Cascading, ...
- Mahout
- others...



As we'll see, the MapReduce API is the assembly language of this ecosystem. It's great when you need the fine-grained control, but otherwise, it can be very difficult to map data flows and complex queries into MR. Also, non-developers can't write Java code and the Streaming API isn't much better for them. So, higher-level tools exist that present a better DSL for users and drive MR to execute the work. Here are some of the options.

Hive is a SQL-like language invented at Facebook. It's ideal for data warehouse applications. Pig is a data flow language that is great for complex transformations, but is harder to learn for people with a SQL background.

Both Hive and Pig can be extended with "User Defined Functions" (UDFs) and in other ways, as we'll see.

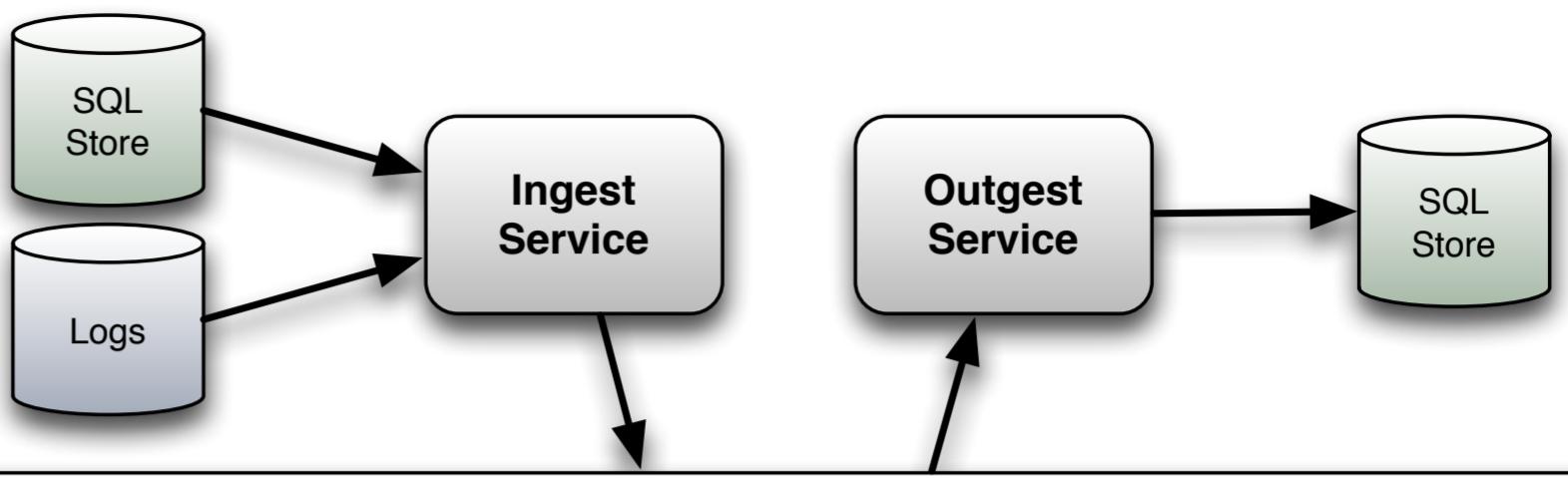
We'll spend most of tomorrow on Hive and most of the next day on Pig.

Cascading is a Java API providing higher-level abstractions. There are very good Clojure and Scala wrappers that further improve productivity.

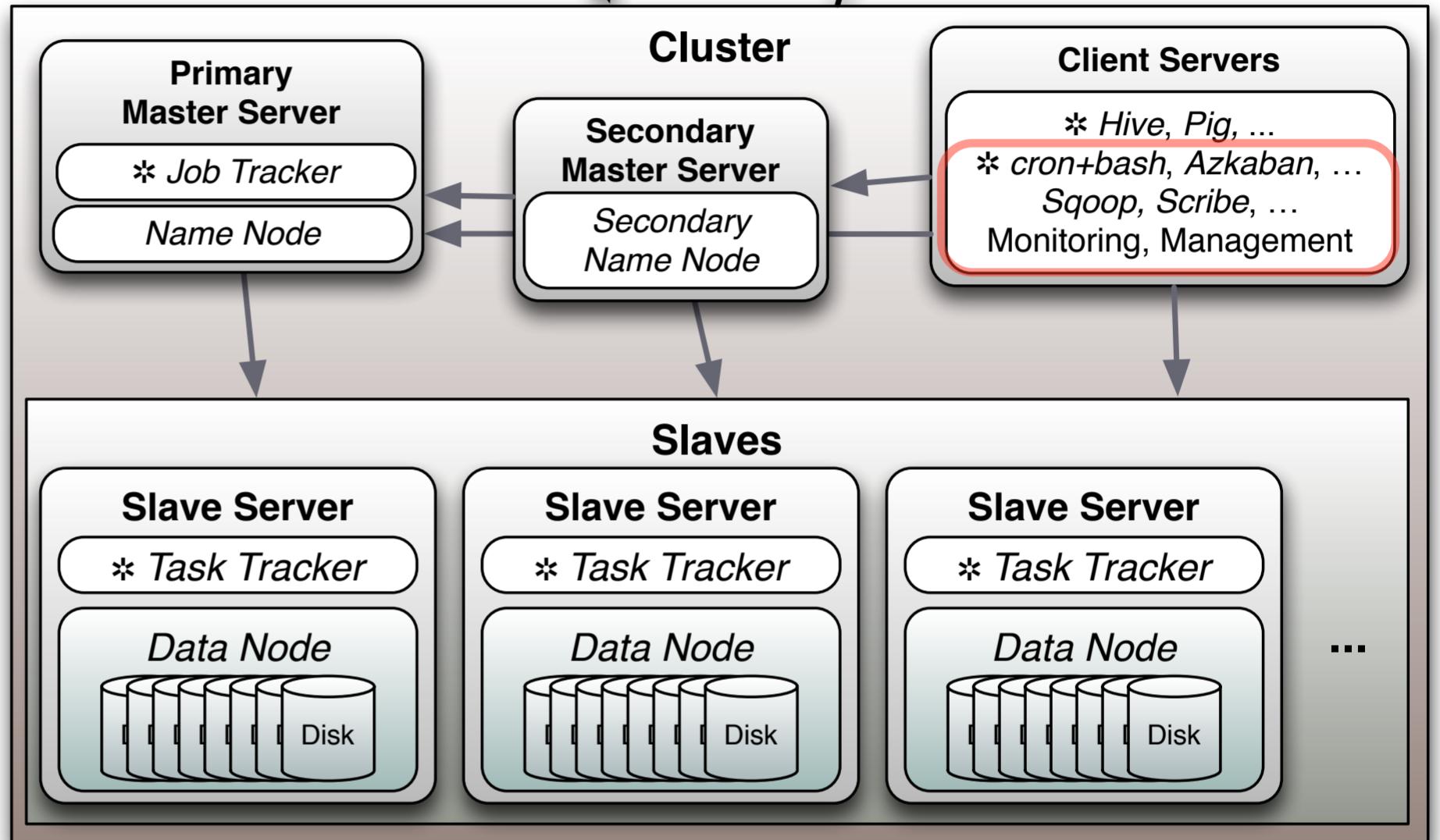
Mahout is a Machine Learning library that runs on Hadoop (although it also has standalone components).

The 3rd-party tools include developer-oriented tools and commercial query tools.

Enterprise Integration, etc.



- ETL
- *Scoop, Scribe, Flume, ...*
- Monitoring, ...



Other tools include extract, transform, and load tools (ETL) for moving log and DB data into Hadoop and exporting back to DBs, as needed. There are job scheduling and monitoring tools, too...

Hadoop Design Goals: Maximize Disk I/O!!

Oh, and run on commodity,
server-class hardware.

Hadoop Design Goals:

Batch Processing:

JVM spin-up for big tasks,
lots of disk sector scans.

Hadoop Design Goals:

Batch Processing:

Not so great for
“real-time” event processing.

Use Cases (Today):

Storing large data sets.
Long-running jobs *crunch*
that data in *parallel*.

Use Cases (Tomorrow):

Better event processing,
incremental updates.

The batch orientation is a great start, but long term, Hadoop will need to be better at incremental algorithms and suitability for “real time”, event processing. People want answers now, not overnight.

HBase + Hadoop is currently the best Hadoop-oriented pseudo-real-time option.

What is *MapReduce*?

MapReduce in Hadoop

Let's look at a
MapReduce algorithm:
WordCount.

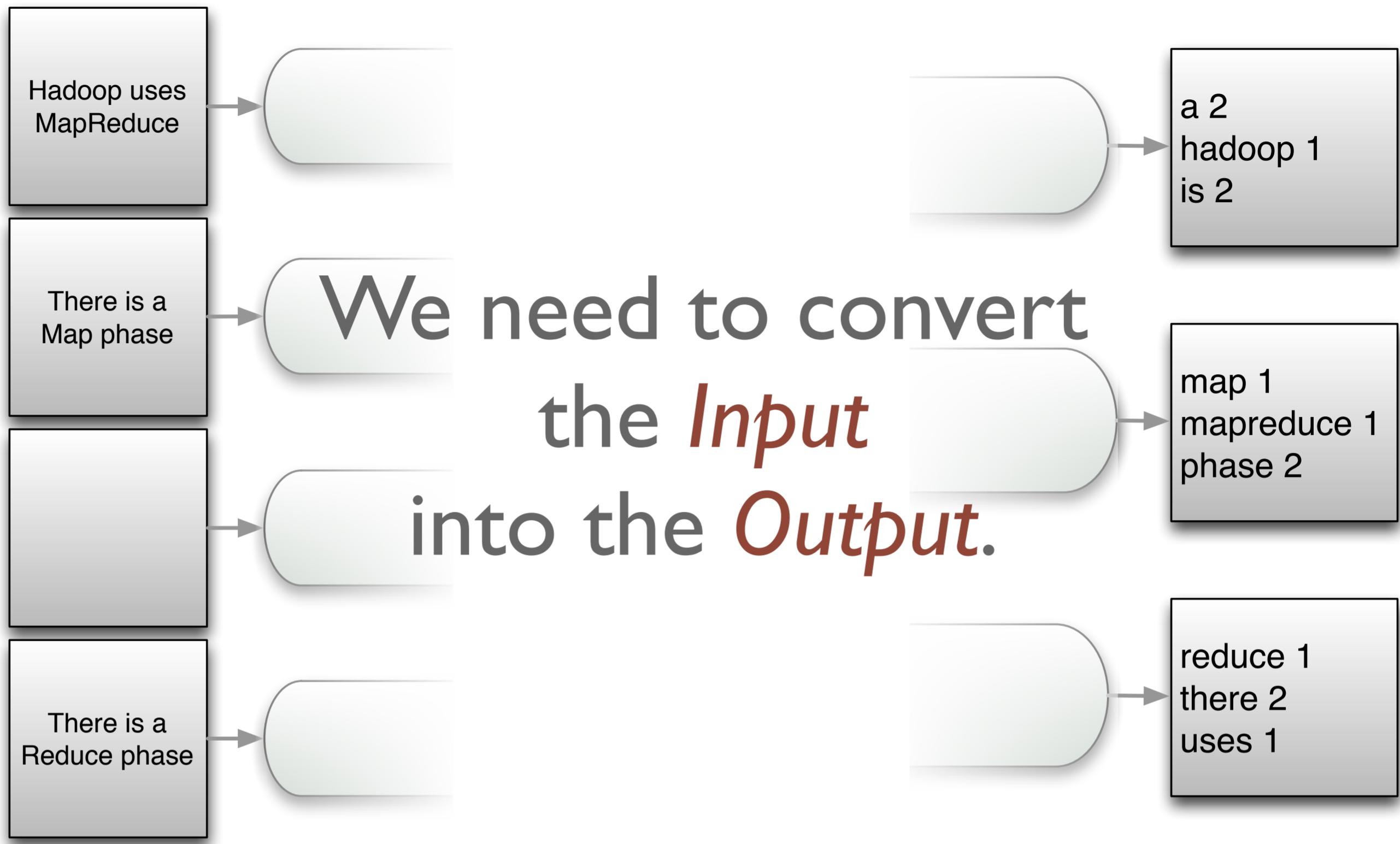
(The *Hello World* in big data...)

Input

Mapper

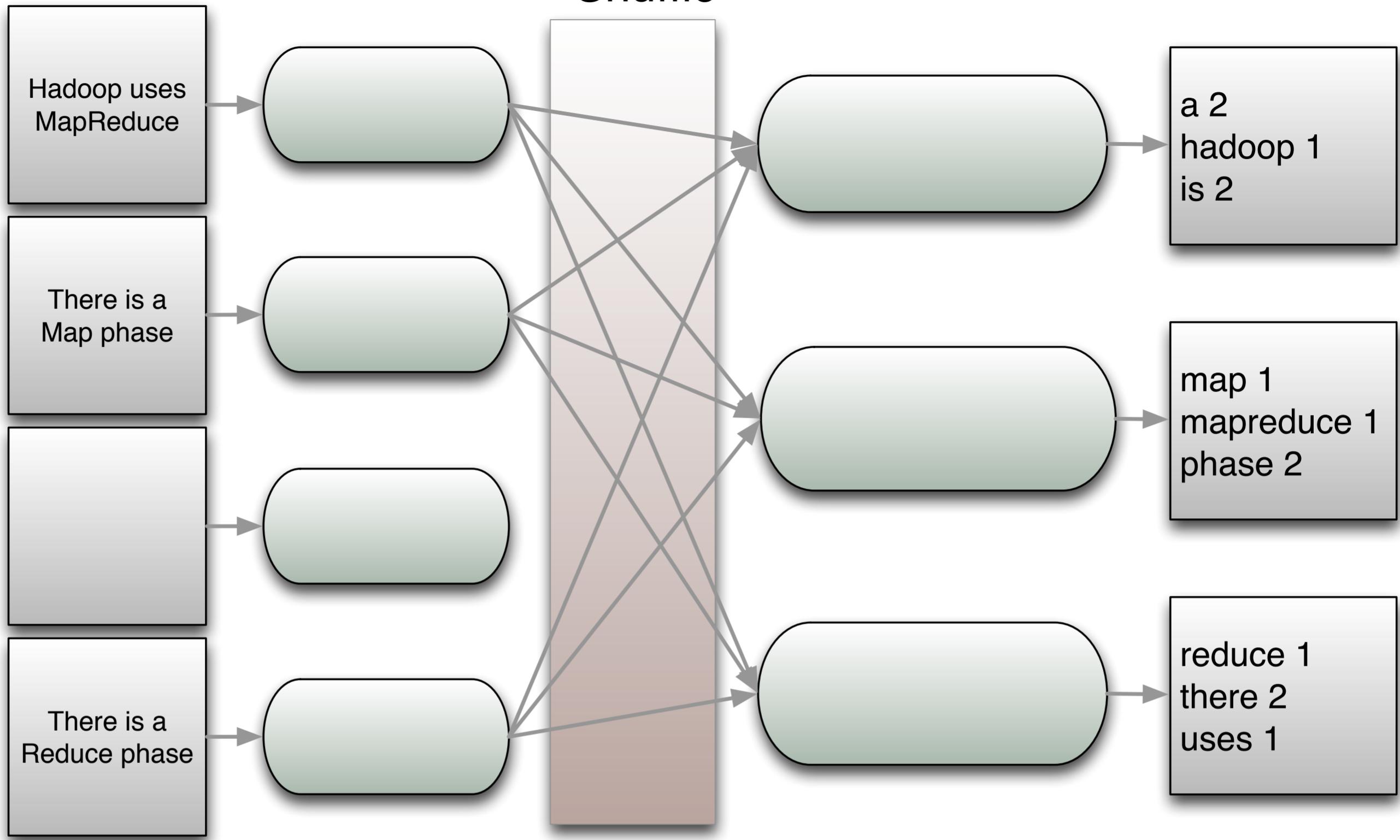
Reducers

Output



Four input documents, one left empty, the others with small phrases (for clarity...). The word count output is on the right (we'll see why there are three output "documents"). We need to get from the input on the left-hand side to the output on the right-hand side.

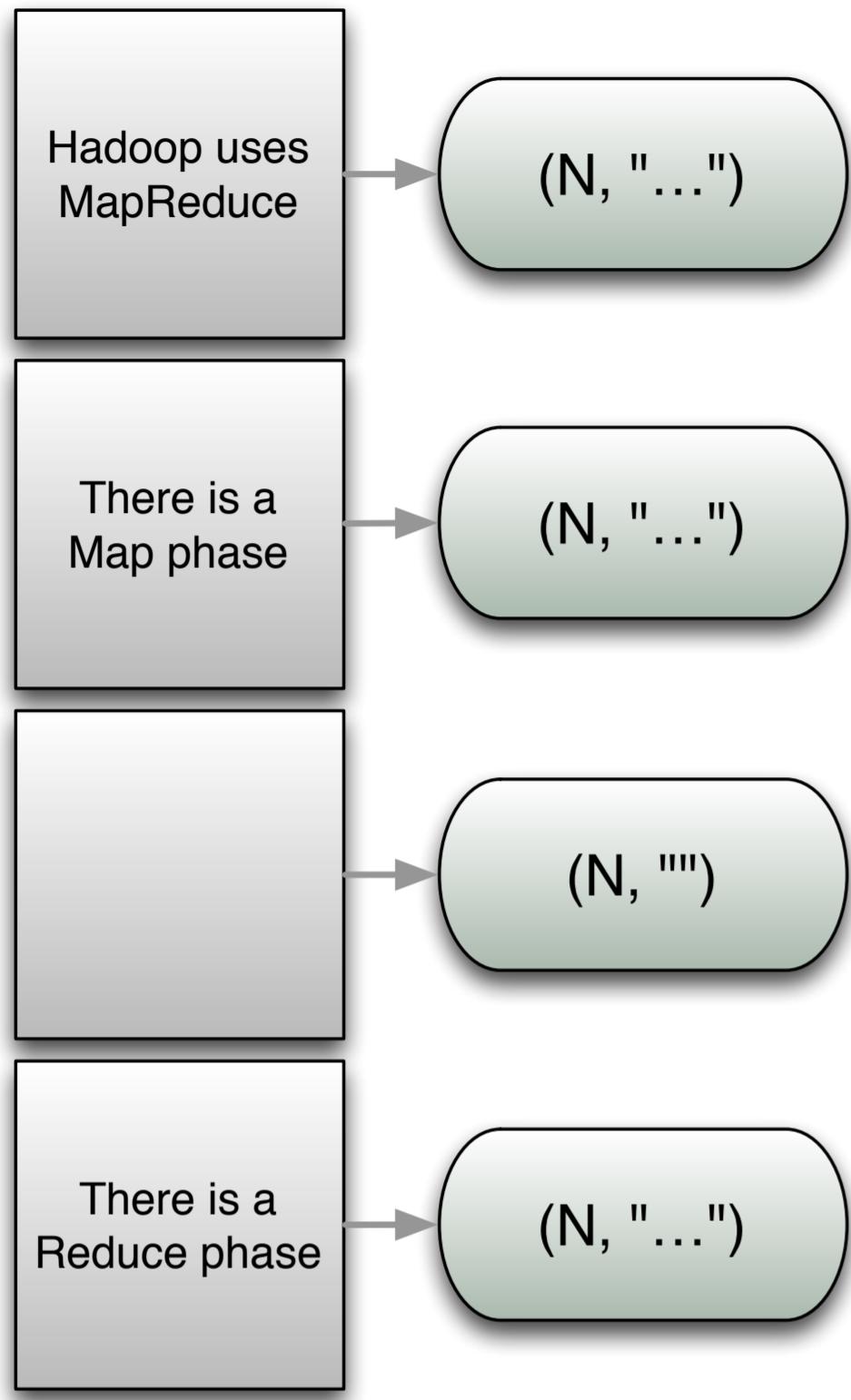
Input Mappers Sort, Shuffle Reducers Output



Here is a schematic view of the steps in Hadoop MapReduce. Each Input file is read by a single Mapper process (default: can be many-to-many, as we'll see later).
The Mappers emit key-value pairs that will be sorted, then partitioned and “shuffled” to the reducers, where each Reducer will get all instances of a given key (for 1 or more values).
Each Reducer generates the final key-value pairs and writes them to one or more files (based on the size of the output).

Input

Mappers

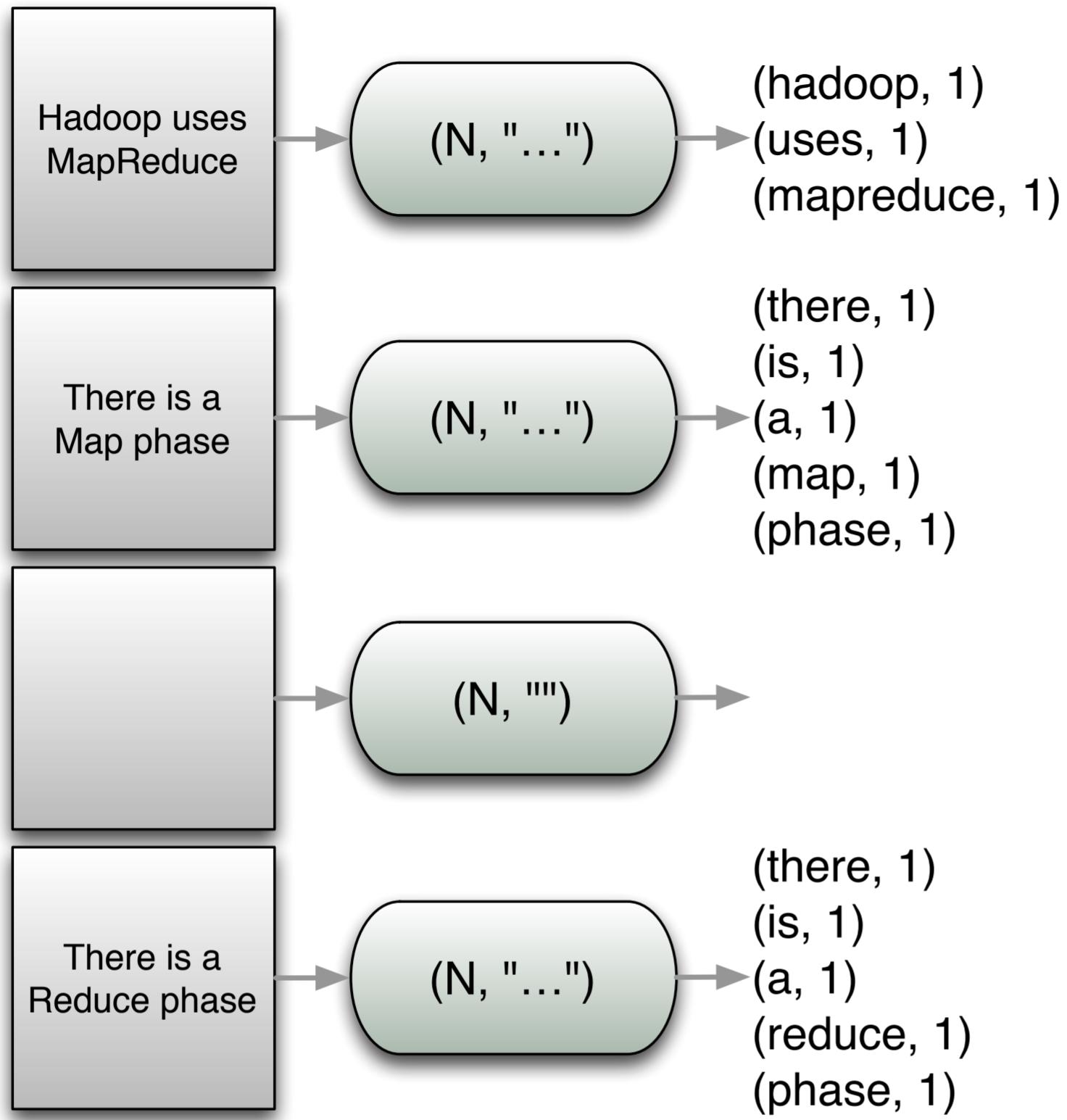


Each document gets a mapper. I'm showing the document contents in the boxes for this example. Actually, large documents might get split to several mappers (as we'll see). It is also possible to concatenate many small documents into a single, larger document for input to a mapper.

Each mapper will be called repeatedly with key-value pairs, where each key is the position offset into the file for a given line and the value is the line of text. We will ignore the key, tokenize the line of text, convert all words to lower case and count them...

Input

Mappers



The mappers emit key–value pairs, where each key is one of the words, and the value is the count. In the most naive (but also most memory efficient) implementation, each mapper simply emits (word, 1) each time “word” is seen.

The mappers themselves don’t decide to which reducer each pair should be sent. Rather, the job setup configures what to do and the Hadoop runtime enforces it during the Sort/Shuffle phase, where the key–value pairs in each mapper are sorted by key (that is locally, not globally or “totally”) and then the pairs are routed to the correct reducer, on the current machine or other machines.

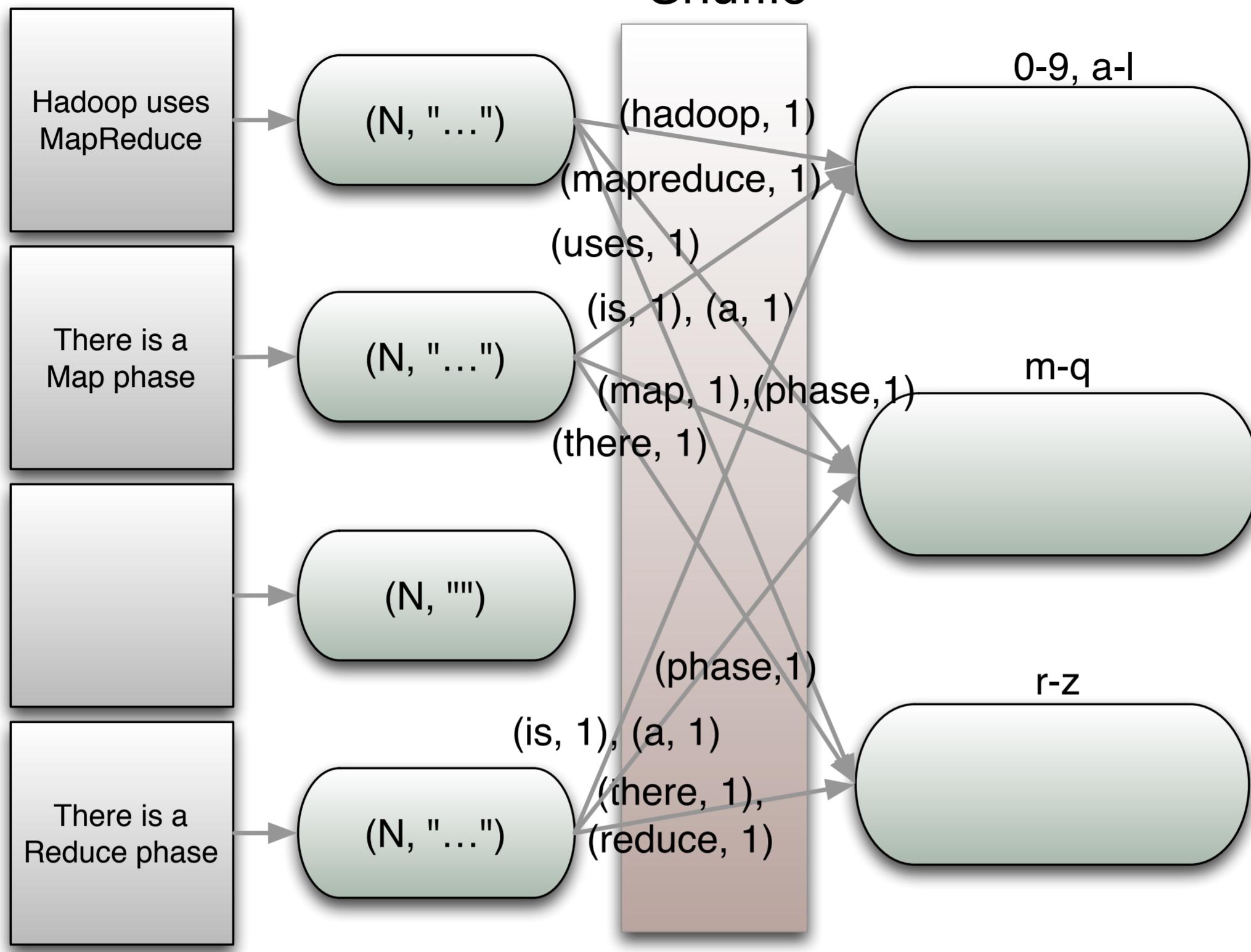
Note how we partitioned the reducers (by first letter of the keys). Also, note that the mapper for the empty doc. emits no pairs, as you would expect.

Input

Mappers

Sort, Shuffle

Reducers



The mappers emit key-value pairs, where each key is one of the words, and the value is the count. In the most naive (but also most memory efficient) implementation, each mapper simply emits (word, 1) each time “word” is seen.

The mappers themselves don't decide to which reducer each pair should be sent. Rather, the job setup configures what to do and the Hadoop runtime enforces it during the Sort/Shuffle phase, where the key-value pairs in each mapper are sorted by key (that is locally, not globally or “totally”) and then the pairs are routed to the correct reducer, on the current machine or other machines.

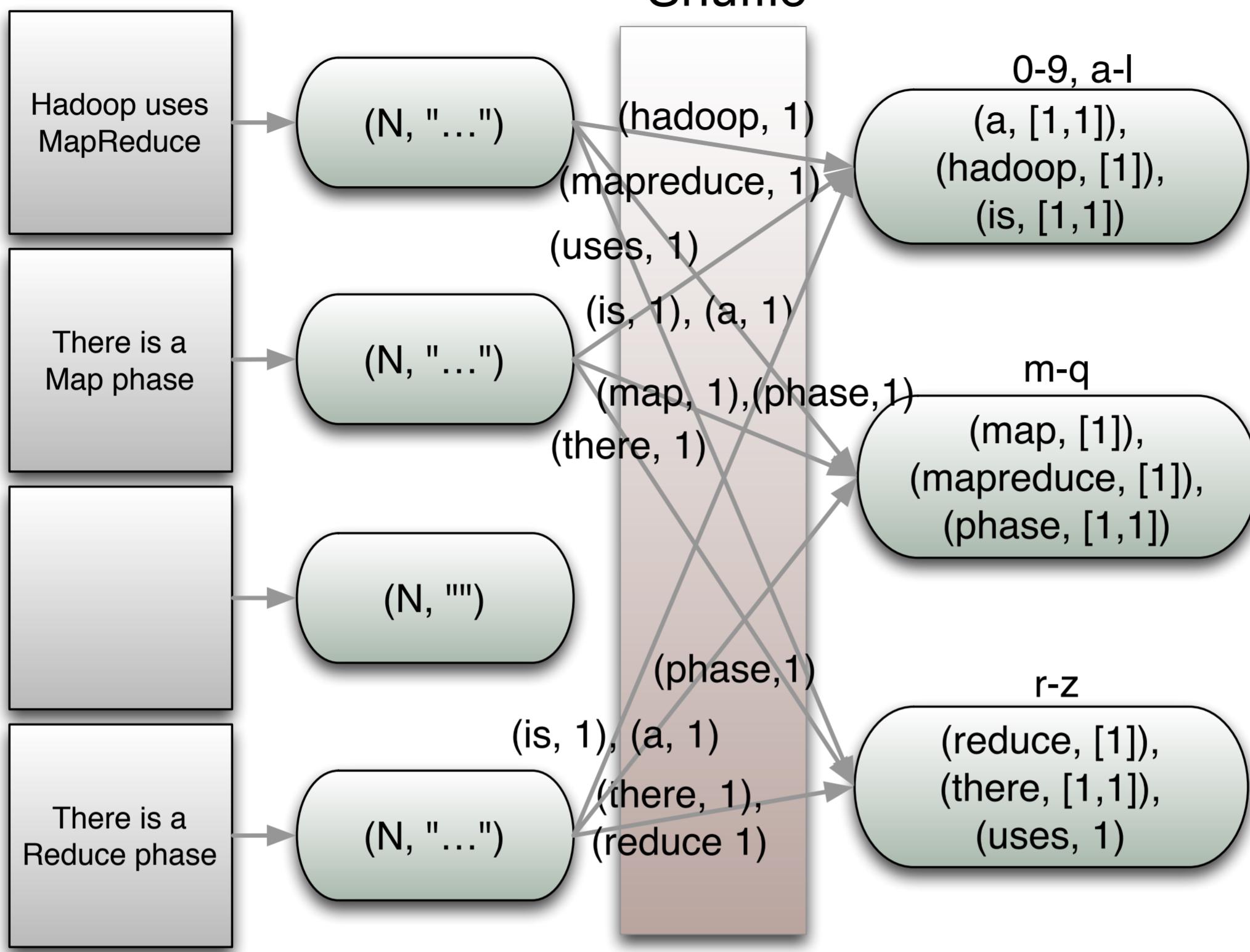
Note how we partitioned the reducers (by first letter of the keys). Also, note that the mapper for the empty doc. emits no pairs, as you would expect.

Input

Mappers

Sort, Shuffle

Reducers

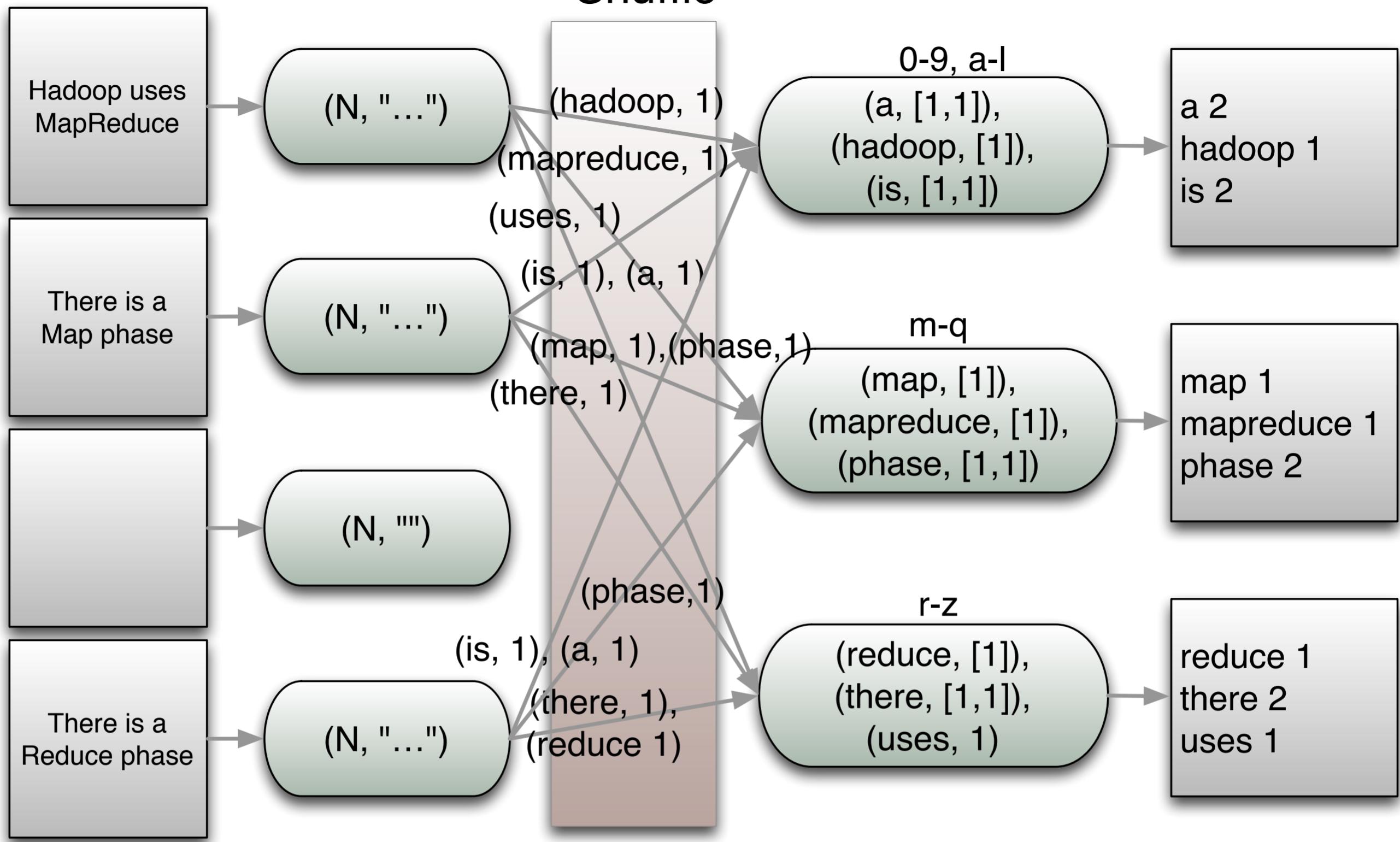


The mappers emit key-value pairs, where each key is one of the words, and the value is the count. In the most naive (but also most memory efficient) implementation, each mapper simply emits (word, 1) each time “word” is seen.

The mappers themselves don’t decide to which reducer each pair should be sent. Rather, the job setup configures what to do and the Hadoop runtime enforces it during the Sort/Shuffle phase, where the key-value pairs in each mapper are sorted by key (that is locally, not globally or “totally”) and then the pairs are routed to the correct reducer, on the current machine or other machines.

Note how we partitioned the reducers (by first letter of the keys). Also, note that the mapper for the empty doc. emits no pairs, as you would expect.

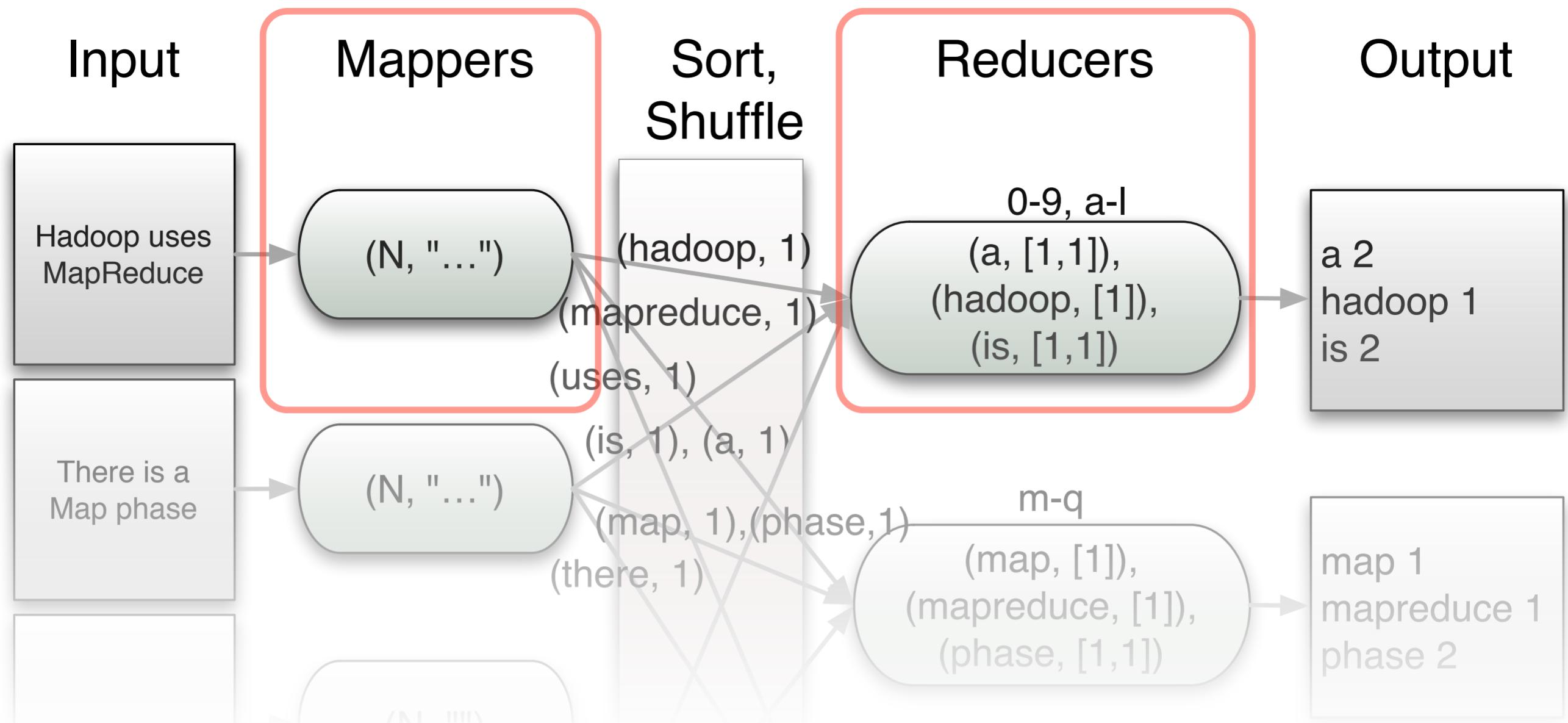
Input Mappers Sort, Shuffle Reducers Output



The final view of the WordCount process flow for our example.

We'll see in more detail shortly how the key-value pairs are passed to the reducers, which add up the counts for each word (key) and then writes the results to the output files.

The the output files contain one line for each key (the word) and value (the count), assuming we're using text output. The choice of delimiter between key and value is up to you. (We'll discuss options as we go.)



Map:

- Transform *one* input to *0-N* outputs.

Reduce:

- Collect *multiple* inputs into *one* output.

To recap, a “map” transforms one input to one output, but this is generalized in MapReduce to be one to 0-N. The output key-value pairs are distributed to reducers. The “reduce” collects together multiple inputs with the same key into

Hive

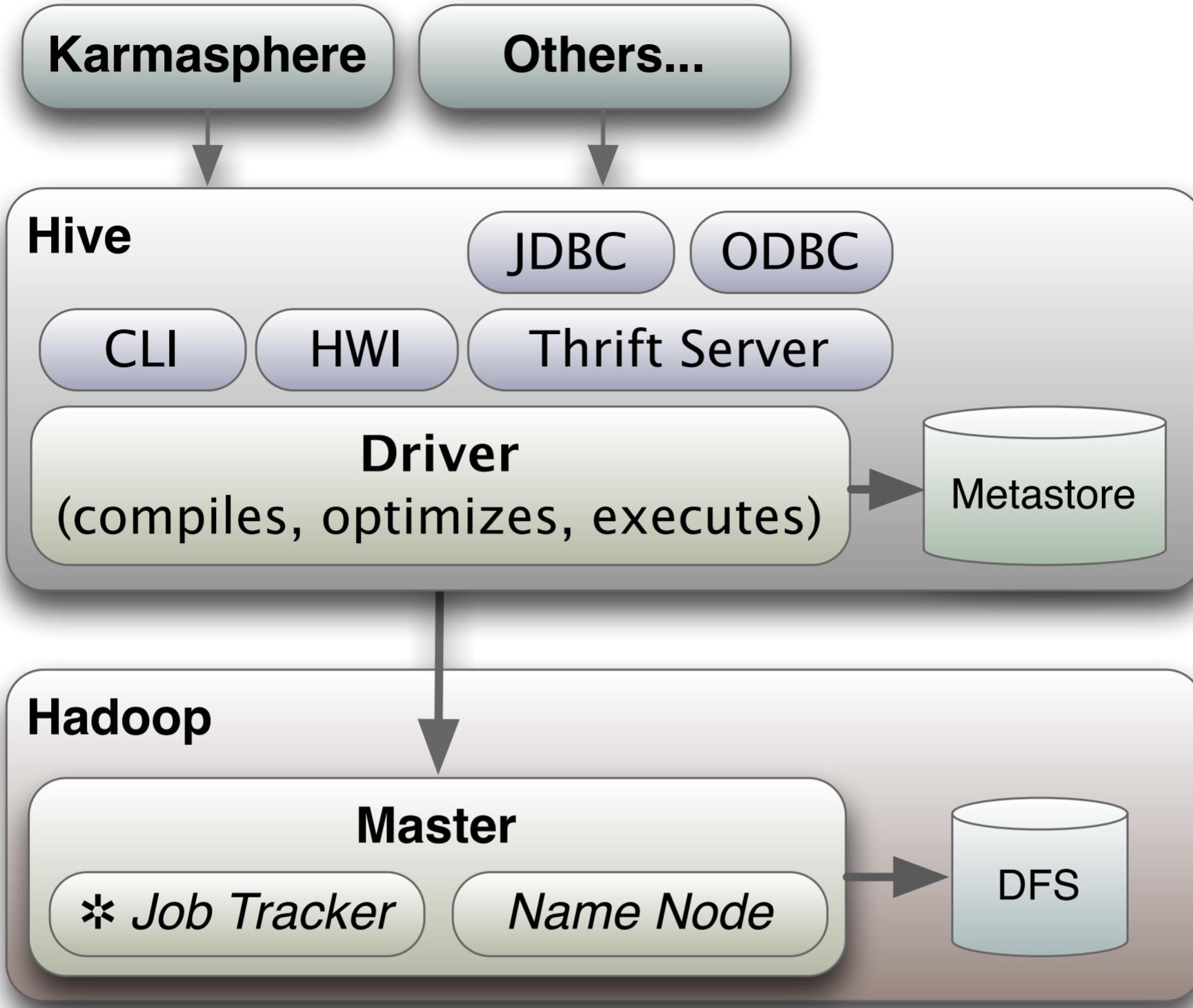


Tuesday, October 23, 12

Now let's move to Hive.

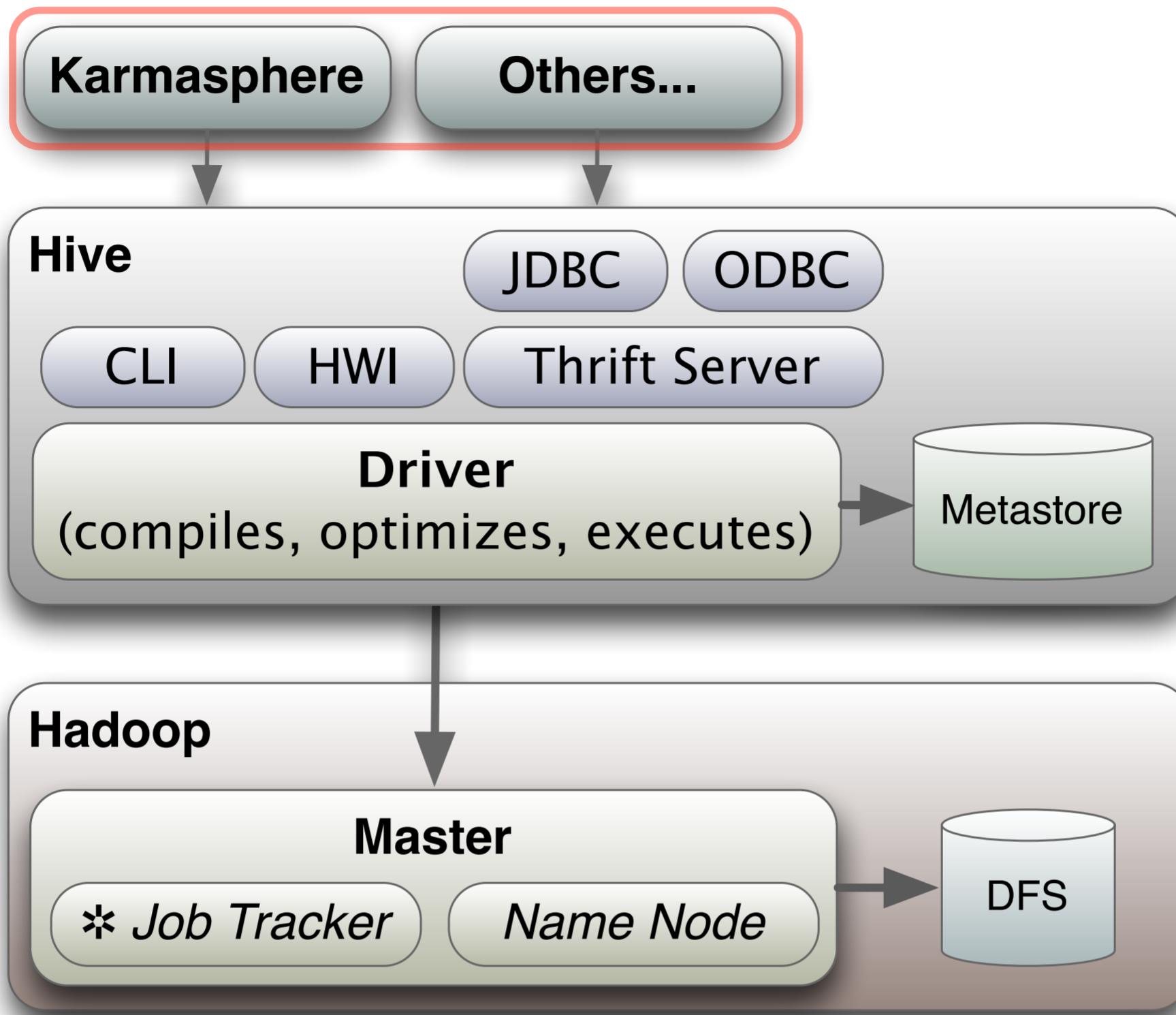
Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Hive + Hadoop



Most Hive queries generate MapReduce jobs. (Some operations don't invoke MapReduce, e.g., those that just write updates to the metastore and "select * from table;" queries.) We've omitted some arrows within the Hive bubble for clarity. They go "down", except for the horizontal connection between the driver and the metastore.

Hive + Hadoop

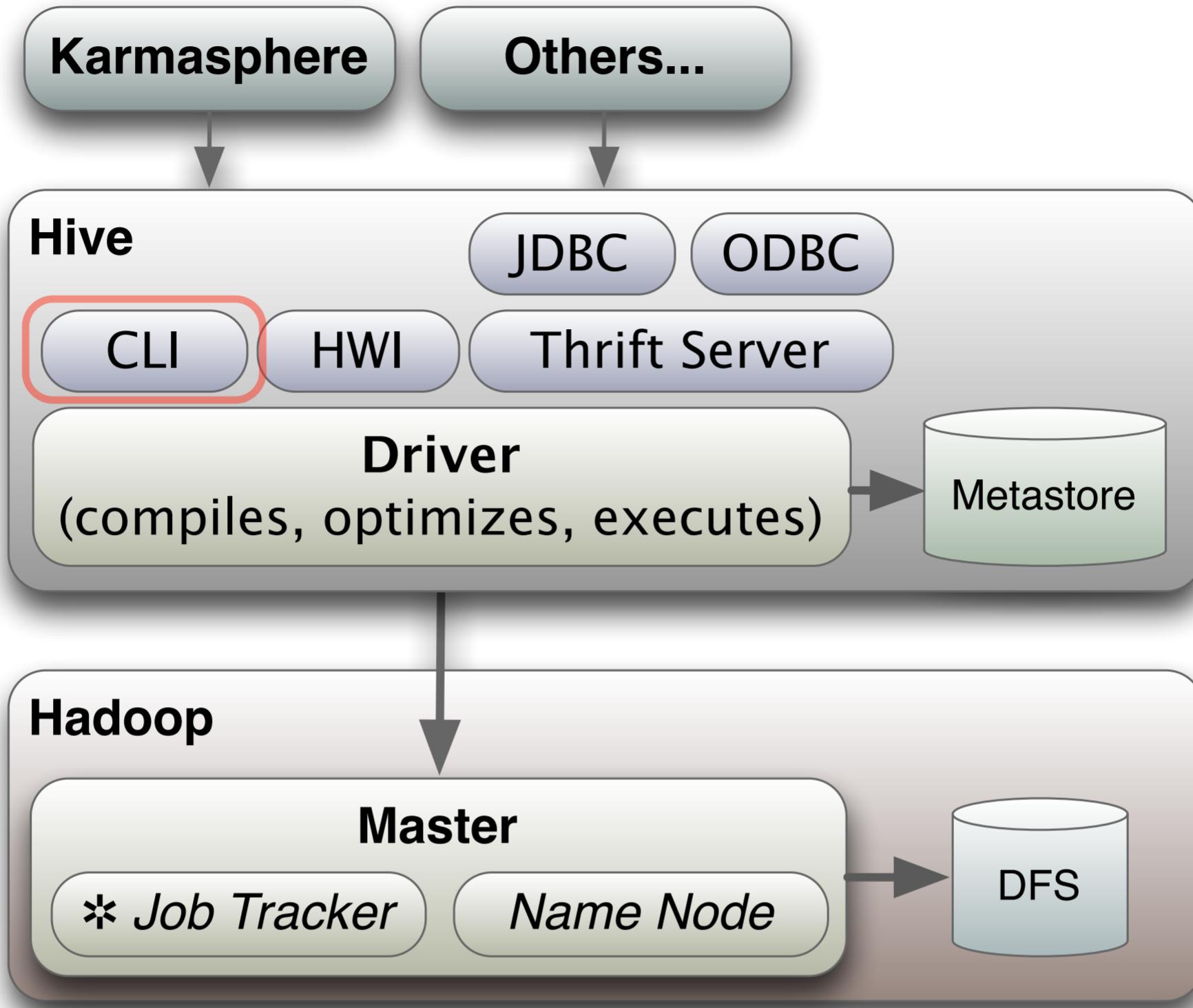


CLI = Command Line Interface.

HWI = Hive Web Interface.

We'll discuss the operation of the driver shortly.

Hive + Hadoop

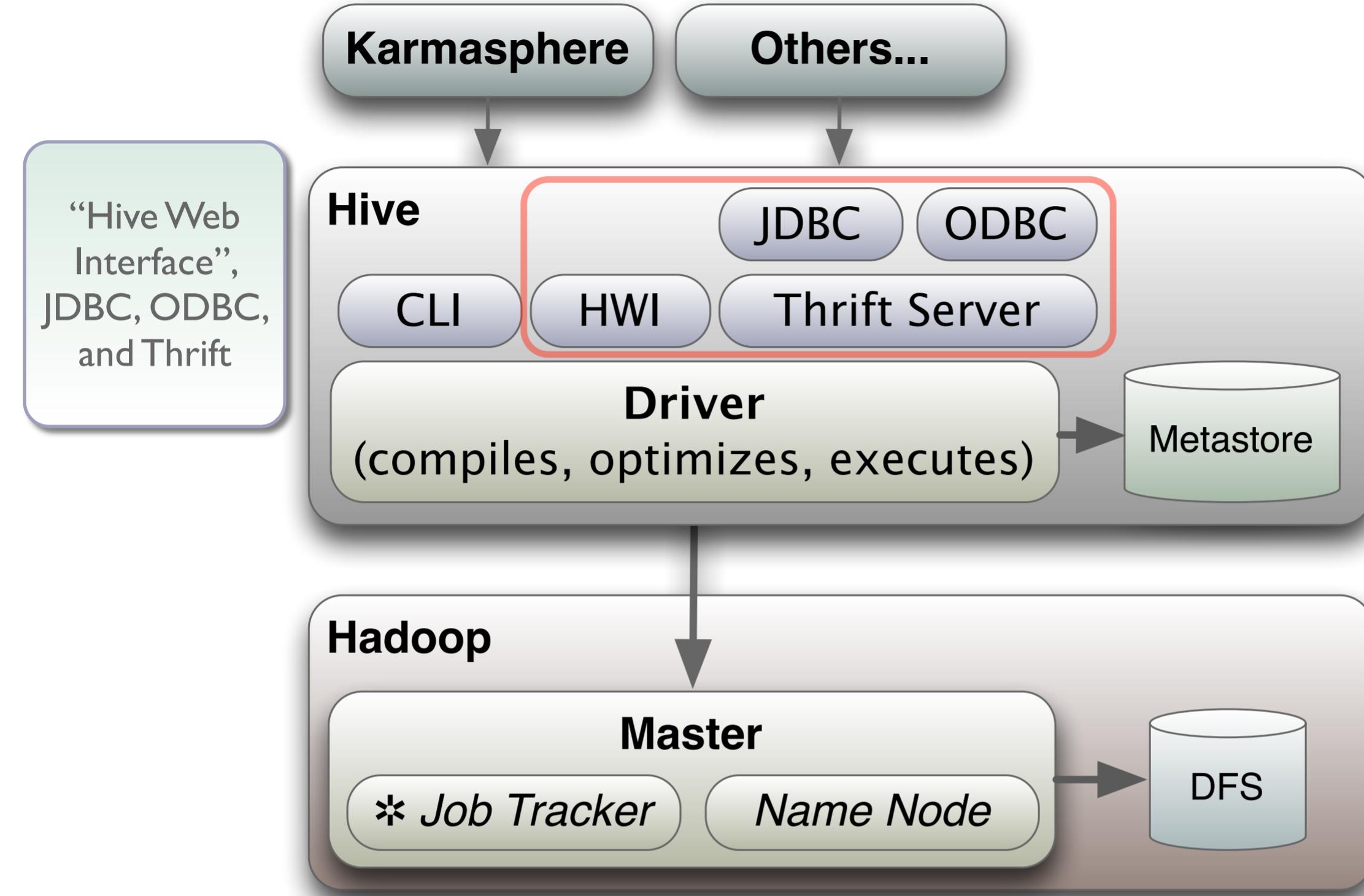


CLI = Command Line Interface.

HWI = Hive Web Interface.

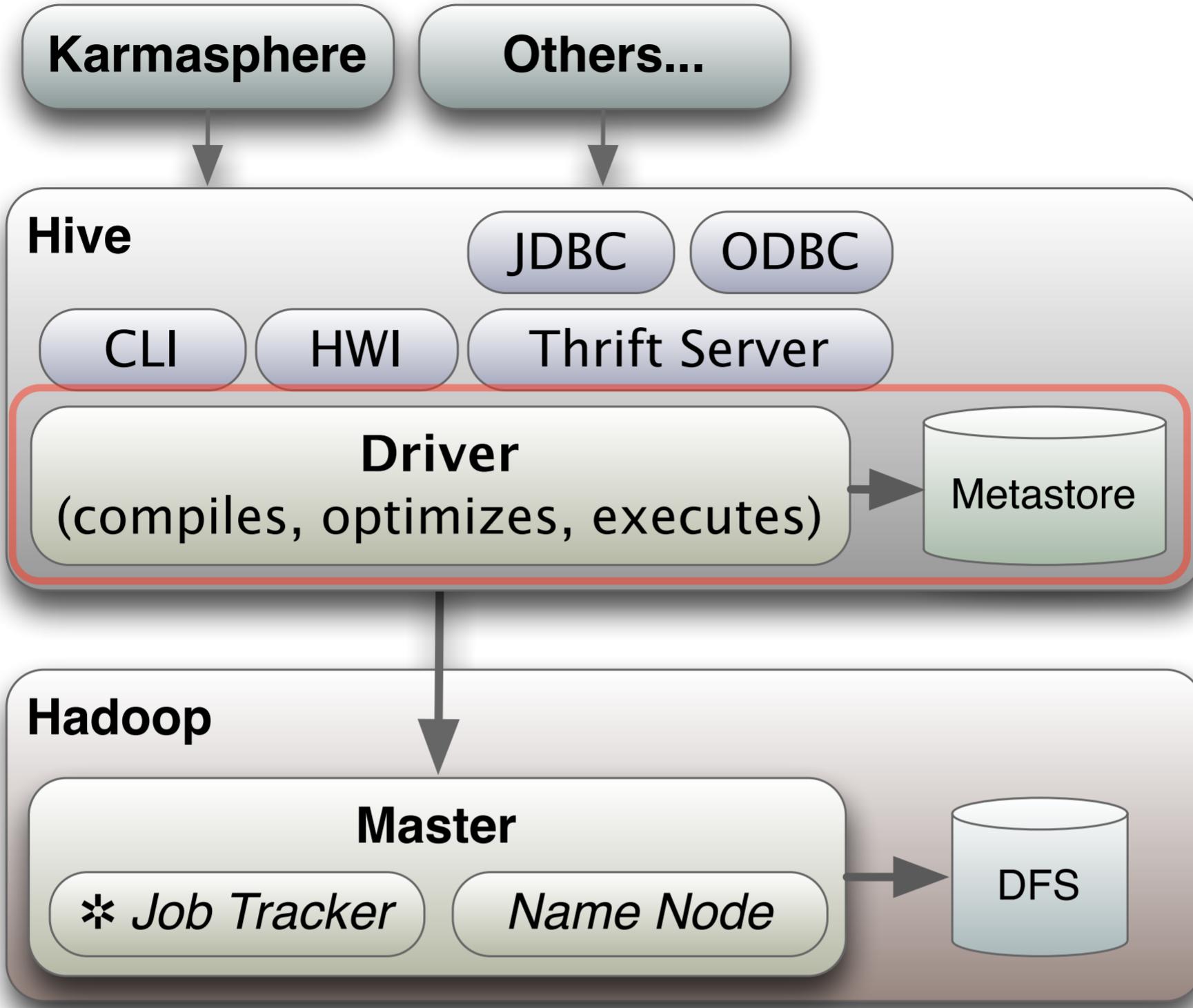
We'll discuss the operation of the driver shortly.

Hive + Hadoop



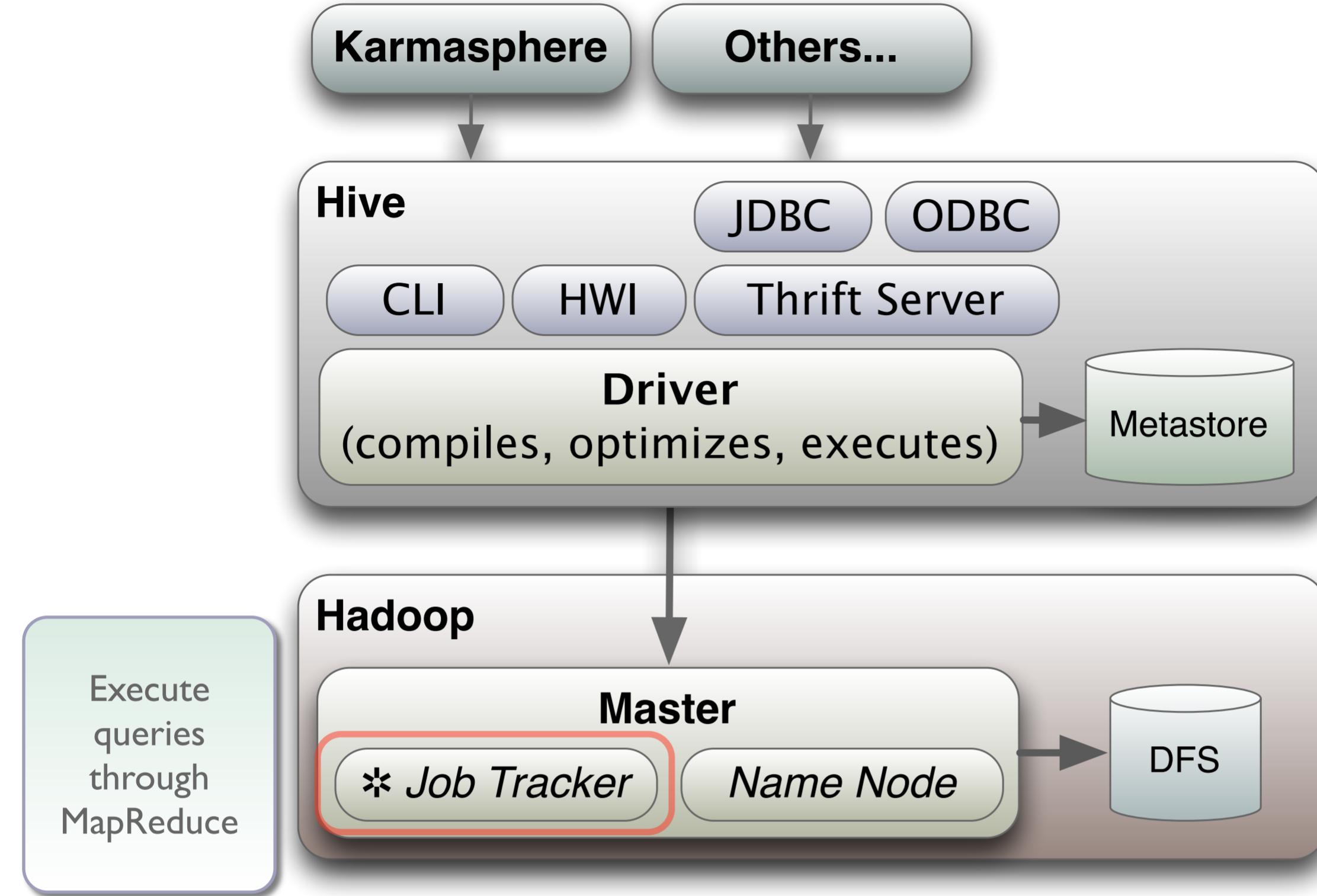
The HWI is very primitive and not very useful. You can also drive Hive from Java programs using JDBC and other languages using ODBC. These interfaces sit on top of a Thrift server, where Thrift is an RPC system invented by Facebook.

Hive + Hadoop



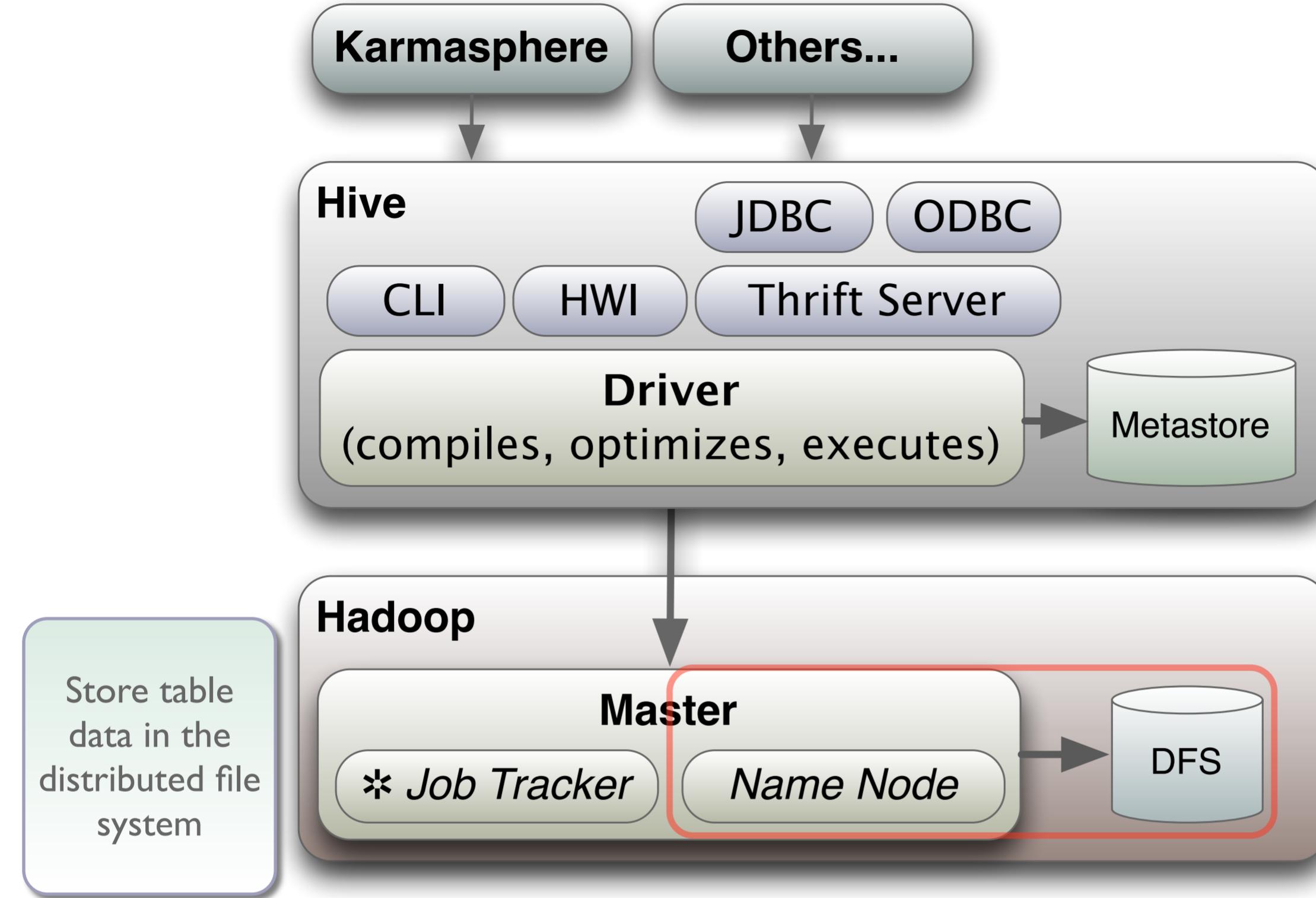
The Driver compiles the queries, optimizes them, and executes them, by *usually* invoking MapReduce jobs, but not always, as we'll see.

Hive + Hadoop



Most Hive queries generate MapReduce jobs. (Some operations don't invoke MapReduce, e.g., those that just write updates to the metastore and "select * from table;" queries.)

Hive + Hadoop



Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

Most Hive queries generate MapReduce jobs. (Some operations don't invoke MapReduce, e.g., those that just write updates to the metastore and "select * from table;" queries.) We've omitted some arrows within the Hive bubble for clarity. They go "down", except for the horizontal connection between the driver and the metastore.

- Due to *HDFS* and *MapReduce* foundations:
 - No Row-level *updates* nor *transactions*.
 - + Parallelized *queries* over *massive* data sets.

Hive-1-Walkthrough

```
WordCountWordCountBuffering.scala
WordCountBufferingFlushing.scala
WordCountNoBuffering.scala
WordCountNoBufferingTokenizing.scala
WordCountReduce.scala
Referenced Libraries
Scala Library [2.9.1.final]
JRE System Library [Java SE 6 (MacOS)
lib
lib_managed
project
src
target
all-shakespeare.txt
README.textile
results
run.sh
sbt
setup.sh
WordCount.iml

private[mapper] val mapper: MapperClass = oneOfTheFollowingOptions:
  1 | no | no-buffer      Simplest algorithm, but least efficient.
  2 | not | no-buffer-use-tokenizer Like 'no', but uses a less efficient
  3 | buffer            Buffer the counts and emit just one key-count pair
  4 | buffer-flush     Like 'buffer', but flushes data more often to limit
and
-c | --use-combiner Use the reducer as a combiner."
}

def main(args: Array[String]) {
  val (mapper, useCombiner, inputPath, outputPath) = parseArgs(args)
  case Settings(Some(m), useC, Some(in), Some(out)) => (m, useC, in, out)
  case _ => error("Invalid settings returned by parseArgs for main")
}
println(mapper.getClass.getName)
val conf = new JobConf(this.getClass)
conf.setJobName("Word Count without buffering")
FileInputFormat.addInputPath(conf, new Path(inputPath))
FileOutputFormat.setOutputPath(conf, new Path(outputPath))

conf.setMapperClass(mapper)
conf.setReducerClass(classOf[Reduce])
```

<http://bit.ly/QyvUyA>

The first exercise; an instructor-lead walkthrough of Hive on the EMR clusters. The bit.ly link from the previous slide is repeated...

Hive Schema

Specifying Table Schemas

- Example:

```
CREATE TABLE demo1(  
    id INT,  
    name STRING);
```

- Two columns:
 - One of type INT.
 - One of type STRING (not CHARARRAY).

Simple Data Types

TINYINT, SMALLINT, INT, BIGINT	1, 2, 4, and 8 byte integers
FLOAT, DOUBLE	4 byte (single precision), and 8 byte (double precision) floating point numbers
BOOLEAN	Boolean
STRING	Arbitrary-length String
TIMESTAMP	(v0.8.0) Date string: “yyyy-mm-dd hh:mm:ss.fffffffff” (The “fs” are nanosecs.)
BINARY	(v0.8.0) a limited VARBINARY type

All the types reflect underlying Java types. TIMESTAMP and BINARY are new to v0.8.0. Use a string for pre-0.8.0 timestamps (or BIGINT for Unix epoch seconds, etc.). BINARY has limited support for representing VARBINARY objects. Note that this isn't a BLOB type, because those are stored separately, while BINARY data is stored within the record.

Complex Data Types

ARRAY	Indexable list of items of the same type. Indices start at 0: <code>orders[0]</code>
MAP	Keys and corresponding values. <code>address['city']</code>
STRUCT	Like C-struct or Java object. <code>name.first, name.last</code>

Complex Schema

```
CREATE TABLE employees (
    name STRING,
    salary FLOAT,
    subordinates ARRAY<STRING>,
    deductions MAP<STRING, FLOAT>,
    address STRUCT<street:STRING,
              city:STRING, state:STRING, zip:INT>
);

```

- Uses Java-style “generics” syntax.
 - ARRAY<STRING>
 - MAP<STRING, FLOAT>
 - STRUCT<street:STRING, ...>

Complex Schema

```
CREATE TABLE employees (
    name STRING,
    salary FLOAT, name and salary
    subordinates ARRAY<STRING>,
    deductions MAP<STRING, FLOAT>,
    address STRUCT<street:STRING,
    city:STRING, state:STRING, zip:INT>
);
```

Complex Schema

```
CREATE TABLE employees (
    name STRING,
    salary FLOAT,
    subordinates ARRAY<STRING>,
    deductions MAP<STRING, FLOAT>,
    address STRUCT<street:STRING,
    city:STRING, state:STRING, zip:INT>
);
```

Arrays of the
same type

Complex Schema

```
CREATE TABLE employees (
    name STRING,
    salary FLOAT,
    subordinates ARRAY<STRING>,
    deductions MAP<STRING, FLOAT>,
    address STRUCT<street:STRING,
    city:STRING, state:STRING, zip:INT>
) ;
```

Paycheck
deductions:
(name, %)

Complex Schema

```
CREATE TABLE employees (
    name STRING,
    salary FLOAT,
    subordinates ARRAY<STRING>,
    deductions MAP<STRING, FLOAT>,
    address STRUCT<street:STRING,
    city:STRING, state:STRING, zip:INT>
);

```

Home
address

Normal Form??

```
subordinates ARRAY<STRING>,  
deductions MAP<STRING, FLOAT>,  
address STRUCT<street:STRING,  
city:STRING, state:STRING, zip:INT>
```

- We're trading normal form for faster access to all the data.
- *Essential for multi-TB data sets stored on hard drives!*

Storage Format

- So far, we've used a *plain-text file format*.
- Let's explore it's properties.
- We'll see other formats later.

Terminators (Delimiters)

' \n '	Between rows (records)
^A ('\001')	Between fields (columns)
^B ('\002')	Between ARRAY and STRUCT elements and MAP key-value pairs
^C ('\003')	Between each MAP key and value

Hive uses the term “terminators” in table definitions, but they are really delimiters or separators between “things”.

“^A” means “control-A”. The corresponding ‘\001’ is the “octal code” for how you write the control character in CREATE TABLE statements.

Specifying Encodings

```
CREATE TABLE employees (
    name STRING,
    salary FLOAT,
    subordinates ARRAY<STRING>,
    deductions MAP<STRING, FLOAT>,
    address STRUCT<street:STRING,
    city:STRING, state:STRING, zip:INT>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
COLLECTION ITEMS TERMINATED BY '\002'
MAP KEYS TERMINATED BY '\003'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

All the
defaults shown
explicitly!

The Actual File Format

John Doe^A100000.0^AMary Smith^BTodd
Jones^AFederal Taxes^C.2^BState Taxes^C.
05^BInsurance^C.1^A1 Michigan
Ave.^BChicago^BIL^B60600
...
One line of text.

CREATE TABLE employees (
 name STRING,
 salary FLOAT,
 subordinates ARRAY<STRING>,
 deductions MAP<STRING, FLOAT>,
 address STRUCT<street:STRING,
 city:STRING, state:STRING, zip:INT>)

The schema, for comparison

```

WordCountJob
▶ S WordCountBuffering.scala
▶ S WordCountBufferingFlushing.scala
▶ S WordCountNoBuffering.scala
▶ S WordCountNoBufferingTokenizing.scala
▶ S WordCountReduce.scala
▶ Referenced Libraries
  ▶ Scala Library [2.9.1.final]
  ▶ JRE System Library [Java SE 6 (Mac OS X)]
  ▶ lib
  ▶ lib_managed
  ▶ project
  ▶ src
  ▶ target
    all-shakespeare.txt
    README.textile
    results
    run.sh
    sbt
    setup.sh
    WordCount.iml

private[mapper] val settings = one of the following options:
1 | no | no-buffer      Simplest algorithm, but least efficient.
2 | not | no-buffer-use-tokenizer Like 'no', but uses a less efficient
3 | buffer            Buffer the counts and emit just one key-count pair
4 | buffer-flush     Like 'buffer', but flushes data more often to limit
and
-c | --use-combiner Use the reducer as a combiner."""

def main(argv: Array[String]): Unit = {
  val (mapper, useCombiner, inputPath, outputPath) = parseArgs(argv)
  case Settings(Some(m), useC, Some(in), Some(out)) => (m, useC, in, out)
  case _ => error("Invalid settings returned by parseArgs for main")
  println(mapper.getClass.getName)

  val conf = new JobConf(this.getClass)
  conf.setJobName("Word Count without Buffering")
  FileInputFormat.addInputPath(conf, new Path(inputPath))
  FileOutputFormat.setOutputPath(conf, new Path(outputPath))

  conf.setMapperClass(mapper)
  conf.setReducerClass(classOf[Reduce])
}

Problems @ Javadoc Declaration
0 items
Description Resource Path Location Type

```

Hive-2-Tables-Schemas

Table *Partitioning*

Partitioning

- Improve query *performance*.
- *Organize* your data.

Partitioning

- Separate *directories* for each partition *column*.

```
CREATE TABLE message_log (
    status STRING, msg STRING, hms STRING)
PARTITIONED BY (
    year INT, month INT, day INT);
```

On disk:

```
message_log/year=2011/month=12/day=31/
message_log/year=2012/month=01/day=01/
...
message_log/year=2012/month=01/day=31/
message_log/year=2012/month=02/day=01/
```

Partitioning

- Speed *queries* by limiting scans to the correct partitions specified in the WHERE clause.

```
SELECT * FROM message_log  
WHERE year = 2012 AND  
      month = 01 AND  
      day   = 31;
```

Without “Partition Filtering”

```
SELECT * FROM message_log;
```

ALL these
directories are read.

```
...
message_log/year=2011/month=12/day=31/
message_log/year=2012/month=01/day=01/
...
message_log/year=2012/month=01/day=31/
message_log/year=2012/month=02/day=01/
...
...
```

Without a WHERE clause that limits the result set, Hive has to read the files in EVERY DIRECTORY ever created for “message_log”. Sometimes, that’s what you want, but the point is that often, you’re likely to do queries between time ranges, so scanning all the data is wasteful.

Filtering by Year

```
SELECT * FROM message_log  
WHERE year = 2012;
```

Just 366 directories
are read.

...

message_log/year=2011/month=12/day=31/
message_log/year=2012/month=01/day=01/

...

message_log/year=2012/month=01/day=31/
message_log/year=2012/month=02/day=01/

...

Filtering by Month

```
SELECT * FROM message_log  
WHERE year = 2012 AND  
month = 01;
```

Just 31 directories
are read.

...

```
message_log/year=2011/month=12/day=31/  
message_log/year=2012/month=01/day=01/
```

...

```
message_log/year=2012/month=01/day=31/  
message_log/year=2012/month=02/day=01/
```

...

Filtering by Day

```
SELECT * FROM message_log  
WHERE year = 2012 AND  
      month = 01 AND  
      day   = 31;
```

Just **one** directory
is read.

...

```
message_log/year=2011/month=12/day=31/  
message_log/year=2012/month=01/day=01/
```

...

```
message_log/year=2012/month=01/day=31/  
message_log/year=2012/month=02/day=01/
```

...

Finally, if you filter by all three, year, month, and day, Hive only has to read one directory! The point is that partitions drastically reduce the amount of data Hive has to scan through, but it's only useful if you pick a partitioning scheme that represents common WHERE clause filtering, like date ranges in this example.

External Tables

(vs. *Managed* Tables)

So far, we have used “managed” tables, where Hive owns the data. What if we want to share data with other tools and not give ownership to Hive? That’s where external tables come in...

External Tables

- When you *manage the data yourself*:
 - The data is used by other tools.
 - You have a custom ETL process.
 - It's common to customize the file format, too...

```
CREATE EXTERNAL TABLE employees (
    name STRING,
    ...)
LOCATION '/data/employees/input' ;
```

Creating *External* Tables

- Example for plain text files:

```
CREATE EXTERNAL TABLE employees (
    name STRING,
    ...
) LOCATION '/data/employees/input';
```

No **scheme** prefix, e.g., `hdfs://server/...`
So, defaults to directory in the cluster.
We own and manage that directory.

Creating *External* Tables

- The locations can be *local*, in *HDFS*, or in *S3*.
- *Joins* can join table data from *any* such source!

The URI's *scheme*.

```
...  
LOCATION 'file:///path/to/data';...  
...  
LOCATION 'hdfs://server:port/path/to/data';  
...  
LOCATION 's3n://mybucket/path/to/data';
```

So, you might have a table pointing to “hot” data in HDFS, a table pointing to a local temporary file created by an ETL staging process, and some longer-lived data in S3 and do joins on all of them!

Dropping *External* Tables

- Because you *manage* the data *yourself*:
 - The table *data are not deleted* when you drop the table.
 - The table *metadata are deleted* from the *metastore*.

External, Partitioned Tables

- You can *partition* your *external* tables.
 - See the exercise...

```

WordCountWordCount
WordCountBuffering.scala
WordCountBufferingFlushing.scala
WordCountNoBuffering.scala
WordCountNoBufferingTokenizing.scala
WordCountReduce.scala
Referenced Libraries
Scala Library [2.9.1.final]
JRE System Library [Java SE 6 (MacOS)
lib
lib_managed
project
src
target
all-shakespeare.txt
README.textile
results
run.sh
sbt
setup.sh
WordCount.iml

private[mapper] val settings = one of the following options:
1 | no | no-buffer    Simplest algorithm, but least efficient.
2 | not | no-buffer-use-tokenizer Like 'no', but uses a less efficient
3 | buffer          Buffer the counts and emit just one key-count pair
4 | buffer-flush    Like 'buffer', but flushes data more often to limit
and
-c | --use-combiner Use the reducer as a combiner."
def main(args: Array[String]): Unit = {
  val (mapper, useCombiner, inputPath, outputPath) = parseArgs(args)
  case Settings(Some(m), useC, Some(in), Some(out)) => (m, useC, in, out)
  case _ => error("Invalid settings returned by parseArgs for main")
}
println(mapper.getClass.getName)
val conf = new JobConf(mapper.getClass)
conf.setJobName("Word Count without Buffering")
FileInputFormat.addInputPath(conf, new Path(inputPath))
FileOutputFormat.setOutputPath(conf, new Path(outputPath))
conf.setMapperClass(mapper)
conf.setReducerClass(WordCountReduce)
  
```

Exercise:

Hive-3-External-Partitioned-Tables

Problems @ Javadoc Declaration

0 items

Description	Resource	Path	Location	Type

Writable Smart Insert Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Select

Statements

[https://cwiki.apache.org/confluence/display/Hive/
LanguageManual+Select](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select)



Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

Mostly what you already know...

Select

- Many of the standard SQL features are supported.
- See the list of operators and functions in the *Hive Cheat Sheet* and here:
 - <https://cwiki.apache.org/confluence/display/Hive/Tutorial#Tutorial-Builtinoperatorsandfunctions>

Select

- We've already seen simple examples.
- Let's dive into the details in the exercise.

```

WordCount.scala
▶ S WordCountBuffering.scala
▶ S WordCountBufferingFlushing.scala
▶ S WordCountNoBuffering.scala
▶ S WordCountNoBufferingTokenizing.scala
▶ S WordCountReduce.scala
▶ Referenced Libraries
  ▶ Scala Library [2.9.1.final]
  ▶ JRE System Library [Java SE 6 (MacOS X)]
  ▶ lib
  ▶ lib_managed
  ▶ project
  ▶ src
  ▶ target
    all-shakespeare.txt
    README.textile
    results
    run.sh
    sbt
    setup.sh
    WordCount.iml

private[mapper] val settings = one of the following options:
1 | no | no-buffer      Simplest algorithm, but least efficient.
2 | not | no-buffer-use-tokenizer Like 'no', but uses a less efficient
3 | buffer           Buffer the counts and emit just one key-count pair
4 | buffer-flush     Like 'buffer', but flushes data more often to limit
and
-c | --use-combiner Use the reducer as a combiner."""

def main(argv: Array[String]): Unit = {
  val (mapper, useCombiner, inputPath, outputPath) = parseArgs(argv)
  case Settings(Some(m), useC, Some(in), Some(out)) => (m, useC, in, out)
  case _ => error("Invalid settings returned by parseArgs for main")
  println(mapper.getClass.getName)

  val conf = new JobConf(this.getClass)
  conf.setJobName("Word Count without Buffering")
  FileInputFormat.setInputPath(conf, new Path(inputPath))
  FileOutputFormat.setOutputPath(conf, new Path(outputPath))

  conf.setMapperClass(mapper)
  conf.setReducerClass(classOf[Reduce])
}

object Settings {
  type Mapper = String
  type UseCombiner = Boolean
  type InputPath = Path
  type OutputPath = Path
}

```

Problems @ Javadoc Declaration

0 items

Description	Resource	Path	Location	Type

Writable Smart Insert Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Exercise: Hive-4-Select

Key Points

- Use *partition filters* to prune sections of data from the search:

```
SELECT ymd, symbol FROM stocks  
WHERE exchange = 'NASDAQ' AND  
      symbol    = 'AAPL' ;
```

Key Points

- Use *partition filters* to prune sections of data from the search:

```
SELECT ymd, symbol FROM stocks  
WHERE exchange = 'NASDAQ' AND  
symbol = 'AAPL' ;
```

```
SELECT * FROM stocks  
WHERE exchange = 'NASDAQ' AND  
symbol = 'AAPL' ;
```

- The second query does *not* require a MR job!

Well chosen partitions can greatly improve the query performance of common queries, by narrowing the range of data that has to be scanned.
In fact, if you're showing all columns and filtering only on partitions, Hive skips MR altogether! However, if the WHERE clause includes non-partition clauses, then MR is required.
(For tables without partitions, "select * from tbl_name;" will also work without MR.)

Key Points

- Use of DISTINCT with *partition* columns does *not* work in Hive before v0.9.0:

```
hive> SELECT DISTINCT symbol FROM stocks;  
OK  
Time taken: 10.273 seconds
```

Key Points

- Use *aggregate functions* to “roll up” data:

```
SELECT count(*) FROM stocks  
WHERE exchange = 'NASDAQ' AND  
symbol      = 'AAPL' ;
```

```
SELECT avg(price_close) FROM stocks  
WHERE exchange = 'NASDAQ' AND  
symbol      = 'AAPL' ;
```

Key Points

- Use GROUP BY to cluster data together:
 - Return the yearly average price for AAPL.

```
SELECT year(ymd), avg(price_close)
FROM stocks
WHERE exchange = 'NASDAQ' AND
      symbol    = 'AAPL' ;
GROUP BY year(ymd);
```

Key Points

- Be aware of *gotchas* in FLOAT and DOUBLE comparisons!

```
SELECT name, deductions['... Taxes']
FROM employees
WHERE deductions['... Taxes'] > 0.2;
```

This happens because the 0.2 is a double, but the actual representation in binary is (roughly) 0.2000000012 (I made up the number of zeros and the “12”, but it’s some small delta above 0.2), while 0.2 as float is 0.2000012; the extra “12” occurs “sooner”. When Hive casts this float to double, it is greater than the literal 0.2 double!

Hive

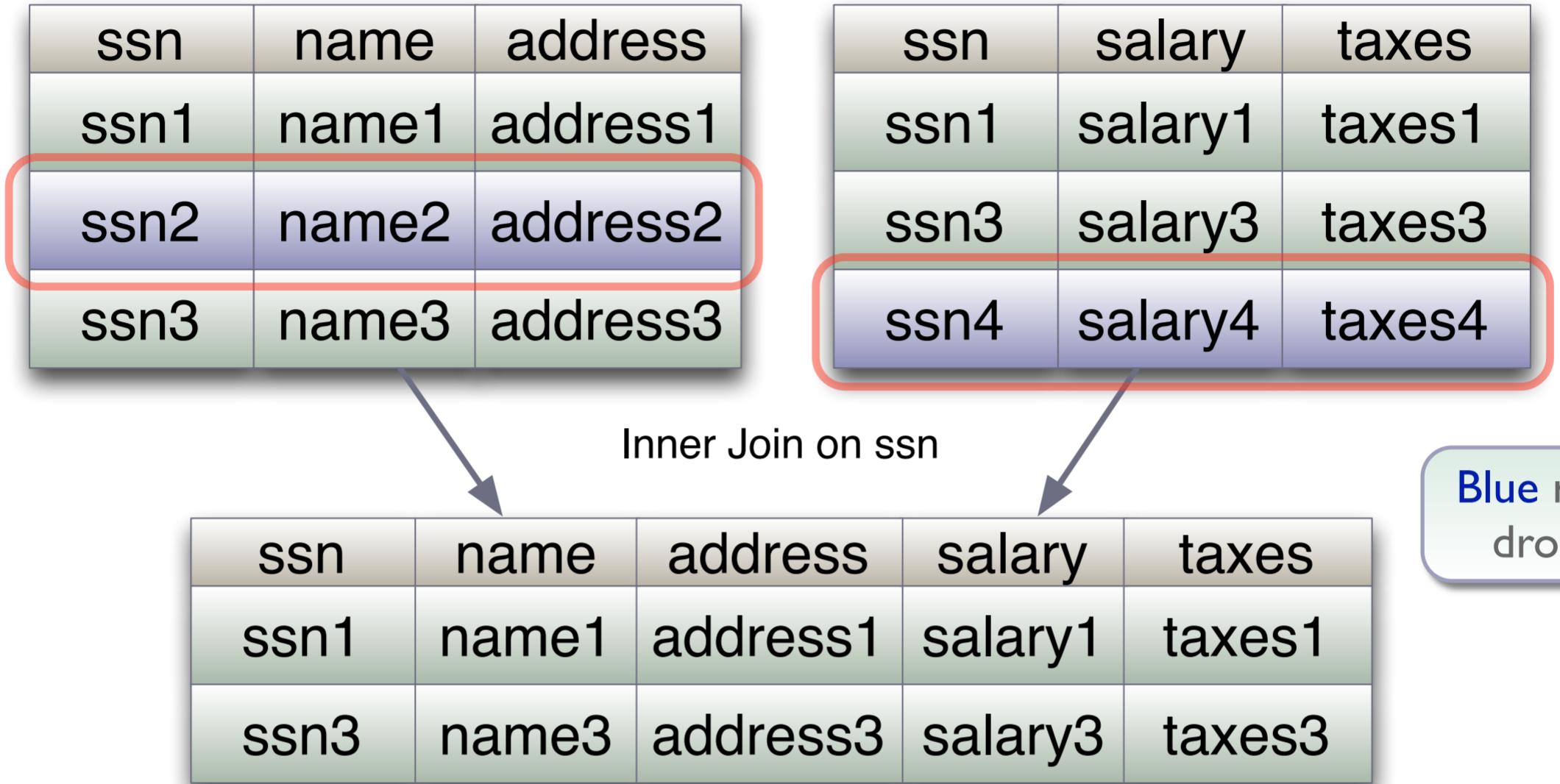
Joins

[https://cwiki.apache.org/confluence/display/Hive/
LanguageManual+Joins](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Joins)

Joins

- Four kinds supported:
 - *Inner* Joins.
 - *Outer* Joins.
 - *Left Semi* Joins (not discussed here).
 - *Map-side* Joins (an optimization of others).

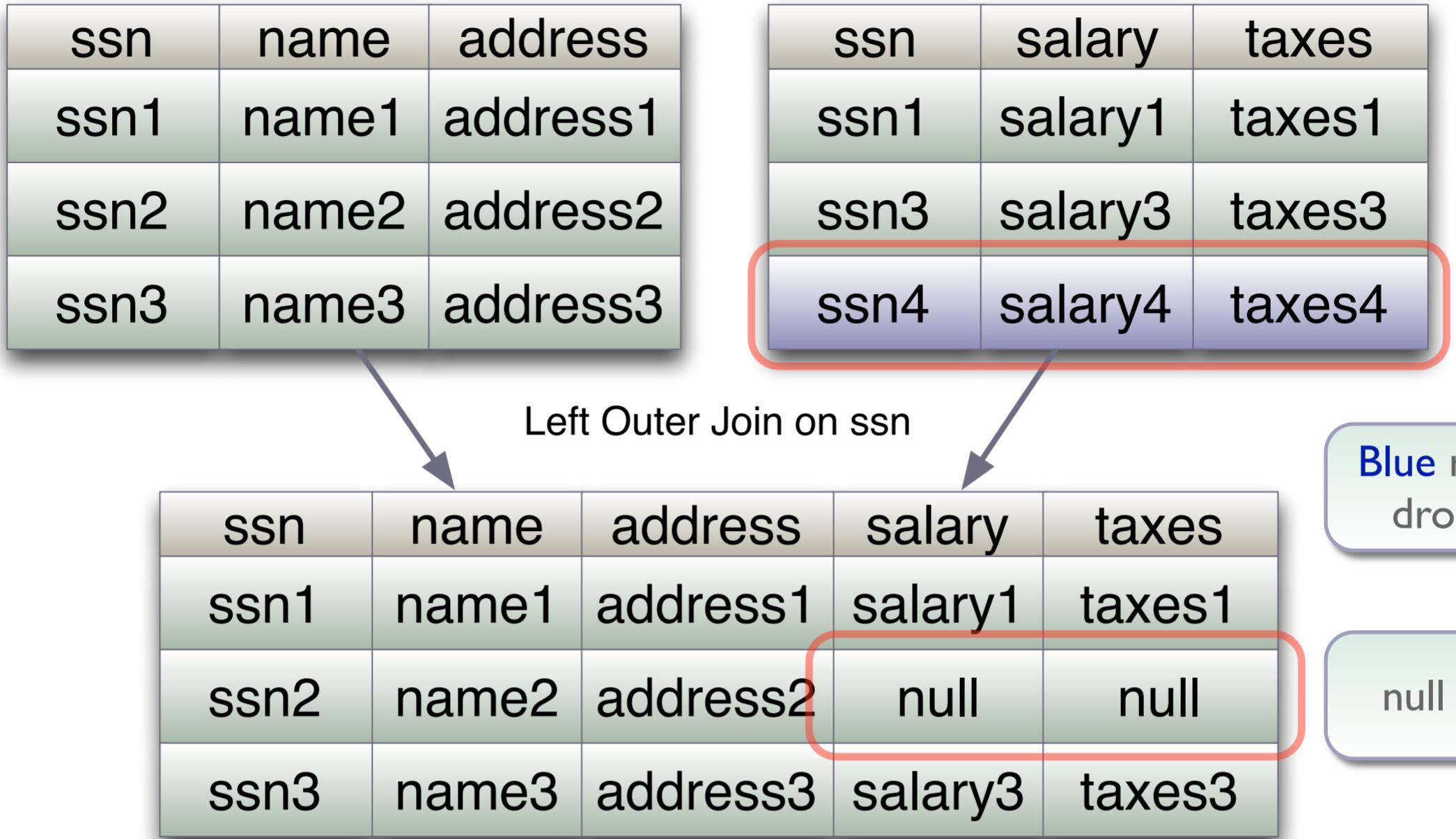
Inner Joins



We're doing a join that requires an identical ssn value in both tables. Records without a match in both tables are discarded.

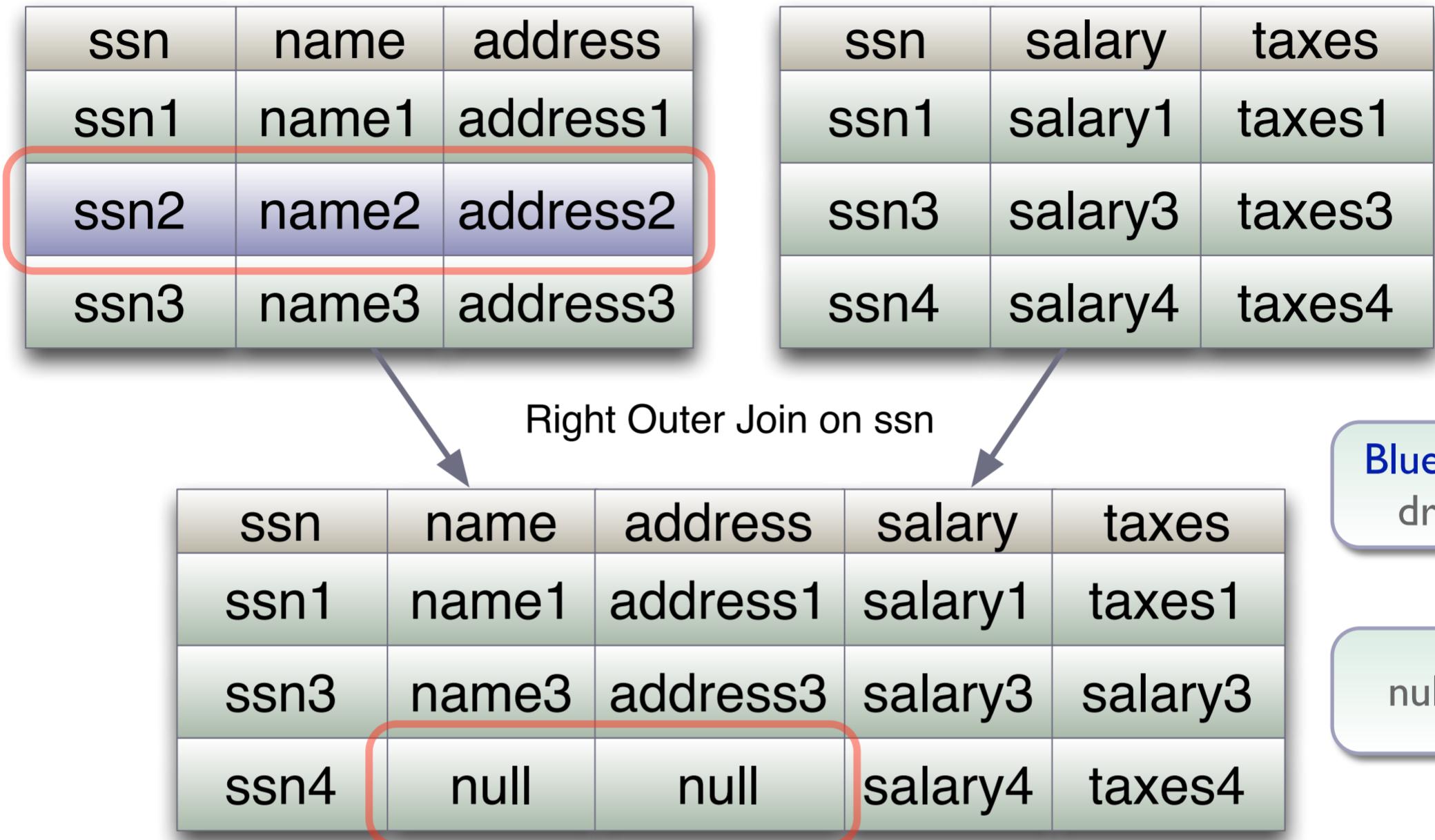
The tan color shows the column names. The green are records that make it through the join, while the blue are the records that are removed because they don't satisfy the match condition.

Left Outer Joins



Now, we keep all the left-hand side records and if there isn't a corresponding record in the right-hand side, we just use null for those fields.

Right Outer Joins



Now, we keep all the right-hand side records and if there isn't a corresponding record in the left-hand side, we just use null for those fields.

Full Outer Joins

Full Outer Join on ssn

ssn	name	address	salary	taxes
ssn1	name1	address1	salary1	taxes1
ssn2	name2	address2	null	null
ssn3	name3	address3	salary3	taxes3
ssn4	null	null	salary4	taxes4

No records dropped.

null fields!

Now, we keep all the right-hand side records and if there isn't a corresponding record in the left-hand side, we just use null for those fields.

Dividend Data

- Like the *stock* data we've been using, let's introduce *dividend* data.

```
CREATE EXTERNAL TABLE IF NOT EXISTS
  dividends (ymd STRING, dividend FLOAT)
PARTITIONED BY (
  exchange STRING, symbol STRING)
ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',';
```

General Features

- Syntax requires JOIN and ON clauses.

```
SELECT s.ymd, s.symbol,  
       s.price_close, d.dividend  
  FROM stocks s  
 JOIN dividends d  
 WHERE s.ymd = d.ymd AND  
       s.symbol = d.symbol  
  WHERE s.ymd > '2010-01-01';
```

Note that the a.ymd > ‘...’ is in the WHERE clause, not the ON clause for the JOIN.
Some SQLs would “infer” the JOIN from just the SELECT clause shown.
Actually you can omit the ON clause, but then you get a MxN cross product!!

General Features

- Only *equality* ($x = y$) conditions allowed.
- *Equi-joins*
- Hard to implement other conditions in MR.

```
SELECT s.ymd, s.symbol,  
       s.price_close, d.dividend  
FROM stocks s  
JOIN dividends d  
ON s.ymd = d.ymd AND  
   s.symbol = d.symbol  
WHERE s.ymd > '2010-01-01';
```

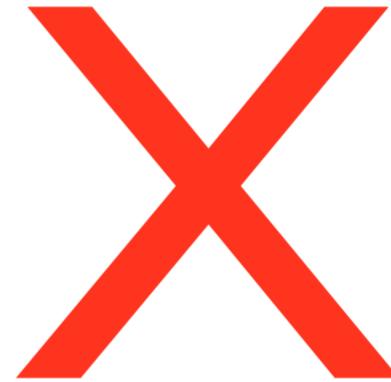
General Features

- Attempt to use general *theta join* condition.

FAILED: Error in semantic analysis: ...

Both left and right aliases encountered in JOIN 'ymd'

```
SELECT s.ymd, s.symbol,  
       s.price_close, d.dividend  
FROM stocks s  
JOIN dividends d  
ON s.ymd <> d.ymd AND  
   s.symbol = d.symbol  
WHERE s.ymd > '2010-01-01';
```



General Features

- Joining more than two tables.
- Will use *one* MR job if the *same* column for every table is used in the join clause.

```
SELECT ...
FROM stocks a
JOIN stocks b ON a.ymd = b.ymd
JOIN stocks c ON a.ymd = c.ymd
WHERE a.symbol = 'AAPL' AND
      b.symbol = 'IBM' AND
      c.symbol = 'INTC' AND
      a.ymd > '2010-01-01';
```

A *self join*.

Same column from a used,
same from b, and same from c.

If I used a different column from a in the second join, 2 MR jobs would be required. Note this doesn't mean I have to "key" for all tables, just the same column for each table throughout, not more than one column from any of the tables.

General Features

- Put the *biggest* table *last*.
- Reducer will *stream* the last table; *buffer* the others.

```
SELECT s.ymd, s.symbol,  
       s.price_close, d.dividend  
  FROM stocks s  
 JOIN dividends d  
    ON s.ymd = d.ymd AND  
       s.symbol = d.symbol  
 WHERE s.ymd > '2010-01-01';
```

Streamed column

General Features

- But you can override which table is *streamed*.

```
SELECT /*+ STREAMTABLE(s) */ ...
  s.price_close, d.dividend
FROM stocks s
JOIN dividends d
ON s.ymd = d.ymd AND
   s.symbol = d.symbol
WHERE s.ymd > '2010-01-01' ;
```

Inner Joins

- Row must exist in *both* tables (*inner join*).
- A *self join* on the *same* table.

No extra
keyword
=> inner join

```
SELECT s.ymd, s.symbol,  
       s.price_close, d.dividend  
  FROM stocks s  
    JOIN dividends d  
    ON s.ymd = d.ymd AND  
       s.symbol = d.symbol  
 WHERE s.ymd > '2010-01-01';
```

Left Outer Joins

- Returns all rows for left-hand table (subject to WHERE) even when there are no right-hand matches. (NULLs returned for those fields.)

```
SELECT s.ymd, s.symbol,  
       s.price_close, d.dividend  
  FROM stocks s  
LEFT OUTER JOIN dividends d  
    ON s.ymd = d.ymd AND  
       s.symbol = d.symbol  
 WHERE s.symbol = 'AAPL';
```

Left Outer Joins

- This query returns:

```
...
1988-02-10 AAPL 41.00 NULL
1988-02-11 AAPL 40.63 NULL
1988-02-12 AAPL 41.00 0.02
1988-02-16 AAPL 41.25 NULL
1988-02-17 AAPL 41.88 NULL
...
...
```

A *three-day* gap for
President's Day
weekend...

Left Outer Joins

- Let's optimize the query with a *partition filter*.

```
SELECT s.ymd, s.symbol,  
       s.price_close, d.dividend  
  FROM stocks s  
 LEFT OUTER JOIN dividends d  
    ON s.ymd = d.ymd AND  
       s.symbol = d.symbol  
 WHERE s.symbol = 'AAPL' AND  
       d.symbol = 'AAPL' AND  
       s.exchange = 'NASDAQ' AND  
       d.exchange = 'NASDAQ';
```

Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

We'll partition dividends like we do stocks, by exchange and symbol, so add the new clauses to the WHERE clause should speed it up!

Different Output!!

- What we got *before*:

...

1988-02-11 AAPL 40.63 NULL

1988-02-12 AAPL 41.00 0.02

1988-02-16 AAPL 41.25 NULL

...

- What we get *now*:

...

1987-11-17 AAPL 35.0 0.02

1988-02-12 AAPL 41.0 0.02

1988-05-16 AAPL 41.25 0.02

1988-08-15 AAPL 41.25 0.02

Only the days of a dividend payment!

...

Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

Our “optimization” changed the results!

Outer Join Gotcha

- **Why?** Because WHERE is evaluated *after* JOIN, so d.* are *NULL except on dividend days!*

```
SELECT s.ymd, s.symbol,  
       s.price_close, d.dividend  
FROM stocks s  
LEFT OUTER JOIN dividends d  
ON s.ymd = d.ymd AND  
   s.symbol = d.symbol  
WHERE s.symbol = 'AAPL' AND  
      d.symbol = 'AAPL' AND  
      s.exchange = 'NASDAQ' AND  
      d.exchange = 'NASDAQ';
```

Effectively
back to an
inner join!

Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

d.exchange will be NULL when there are no dividend records for a given day, so the WHERE clause will effectively discard ALL stock records on day's when AAPL didn't pay a dividend! So, we're back to an equi-join, not an outer join. This is actually common behavior in most SQLs.

Outer Join Gotcha Fix

- **Fix:** Remove the `d.*` predicates from the WHERE clause.

```
SELECT s.ymd, s.symbol,  
    s.price_close, d.dividend  
FROM stocks s  
LEFT OUTER JOIN dividends d  
ON s.ymd = d.ymd AND  
    s.symbol = d.symbol  
WHERE s.symbol = 'AAPL' AND  
    s.exchange = 'NASDAQ';
```

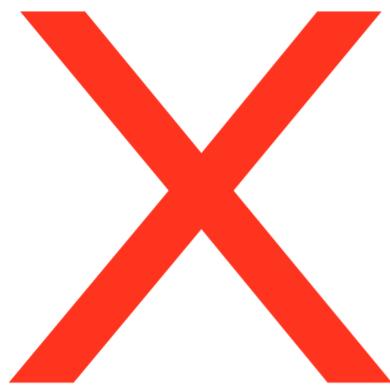
Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Outer Join Gotcha Fix

- **Doesn't work:** Move the JOIN s.exchange and d.exchange predicates *inside* the JOIN clause.

```
SELECT s.ymd, s.symbol,  
       s.price_close, d.dividend  
FROM stocks s  
LEFT OUTER JOIN dividends d  
ON s.ymd      = d.ymd AND  
   s.symbol    = d.symbol AND  
   s.symbol    = 'AAPL' AND  
   d.symbol    = 'AAPL' AND  
   s.exchange  = 'NASDAQ' AND  
   d.exchange  = 'NASDAQ';
```

Copyright © 2011-2012, Think Big Analytics, All Rights Reserved



Outer Join Gotcha Fix

- In general, not all WHERE clause predicates work in ON clauses.
- **Warning:** The Hive Wiki claims these should work!

...

```
s.symbol      = 'AAPL' AND  
d.symbol      = 'AAPL' AND  
s.exchange    = 'NASDAQ' AND  
d.exchange    = 'NASDAQ' ;
```

Or Use Nested Queries

```
SELECT s.ymd, s.symbol,
       s.price_close, d.dividend
  FROM (
    SELECT ymd, symbol, price_close
      FROM stocks WHERE
        exchange = 'NASDAQ' AND symbol = 'AAPL') s
 LEFT OUTER JOIN (
    SELECT ymd, symbol, dividend
      FROM dividends WHERE
        exchange = 'NASDAQ' AND symbol = 'AAPL') d
     ON s.ymd      = d.ymd AND
        s.symbol   = d.symbol;
```

Outer Join Gotcha

- This gotcha applies to all OUTER JOINS.

Right Outer Joins

- Returns all rows for right-hand table (subject to WHERE) even when there is no left-hand match. (NULLs returned for those fields.)

```
SELECT s.ymd, s.symbol,  
       s.price_close, d.dividend  
  FROM dividends d  
RIGHT OUTER JOIN stocks s  
    ON s.ymd = d.ymd AND  
       s.symbol = d.symbol  
 WHERE s.symbol = 'AAPL' AND  
       s.exchange = 'NASDAQ';
```

Full Outer Joins

- Returns all rows for both tables (subject to WHERE) even when records on either side don't match.

```
SELECT s.ymd, s.symbol,  
       s.price_close, d.dividend  
  FROM dividends d  
 FULL OUTER JOIN stocks s  
    ON s.ymd = d.ymd AND  
       s.symbol = d.symbol  
 WHERE s.symbol = 'AAPL' AND  
       s.exchange = 'NASDAQ';
```

This query's results will be the same as for the *right join* query.

Map-side Joins

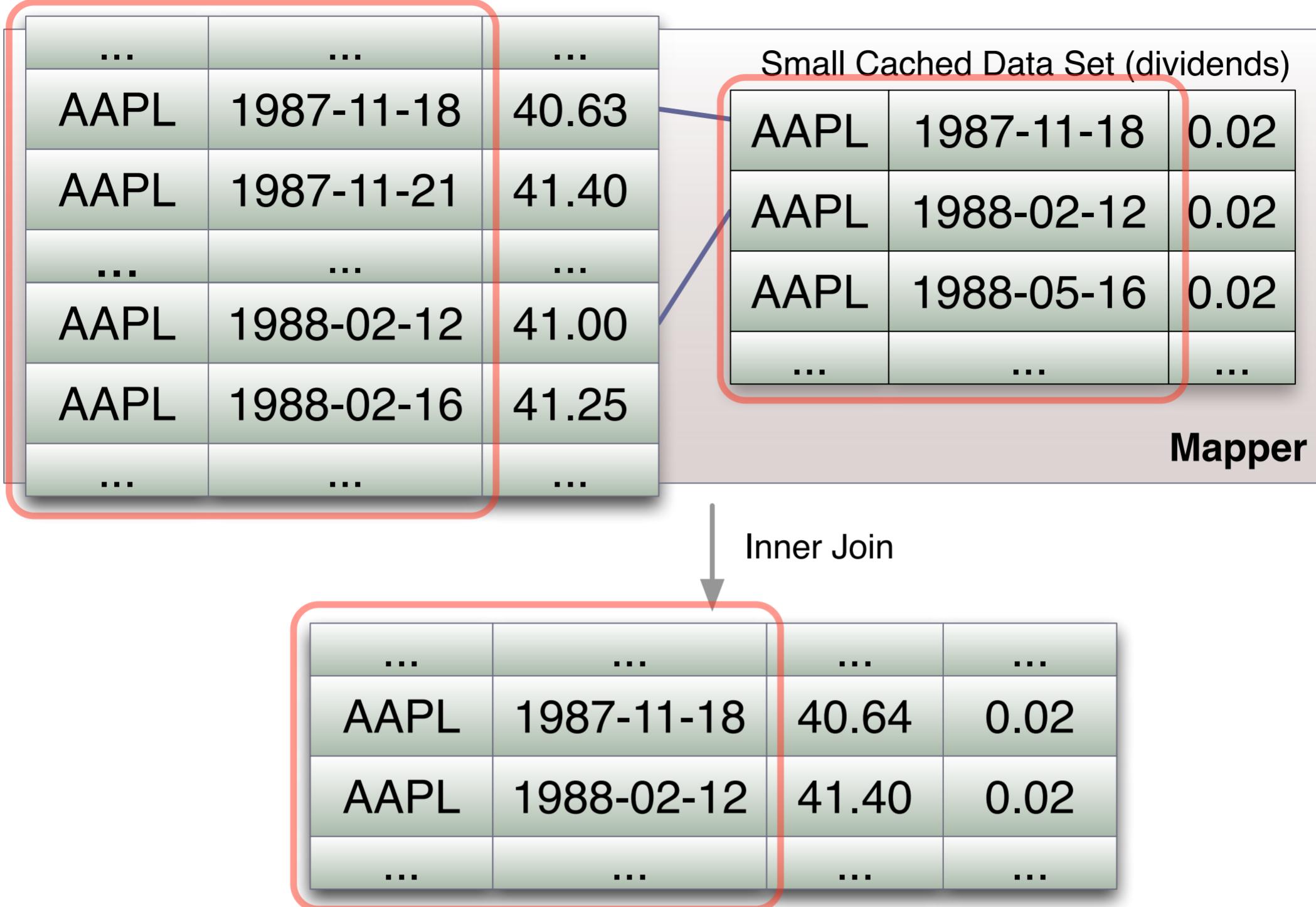
- Join tables in the mapper.
- Optimization that eliminates the reduce step.
- Useful if all but one table is small.
- Useful for sorted and “bucketized” data.

```
SELECT s.ymd, s.symbol,  
       s.price_close, d.dividend  
FROM dividends d  
JOIN stocks s  
ON s.ymd = d.ymd AND  
   s.symbol = d.symbol;
```

A small table, so Hive can apply the optimization.

This query prints all the closing prices for a stock on any day when it pays a dividend. If all but one table is small enough, the mapper can load the small tables in memory and do the joins there, rather than invoking an expensive reduce step.

Large Streamed Data Set (stocks)



Map-side Joins

- In versions of Hive before v0.7.0, you had to add this directive after SELECT:
 - `/*+ MAPJOIN(d) */` (now deprecated)
- The optimization is automatic if:
 - `set hive.auto.convert.join = true;`

Map-side Joins

- Can't be used with RIGHT/FULL OUTER joins.
- See the wiki page for additional optimizations possible when data is BUCKETED and/or SORTED.

```

private[mapper] val settings = one of the following options:
1 | no | no-buffer      Simplest algorithm, but least efficient.
2 | not | no-buffer-use-tokenizer Like 'no', but uses a less efficient
3 | buffer           Buffer the counts and emit just one key-count pair
4 | buffer-flush     Like 'buffer', but flushes data more often to limit
and
-c | --use-combiner Use the reducer as a combiner."
}

def main(argv: Array[String]): Unit = {
    val (mapper, useCombiner, inputPath, outputPath) = parseArgs(argv)
    case Settings(Some(m), useC, Some(in), Some(out)) => (m, useC, in, out)
    case _ => error("Invalid settings returned by parseArgs for main")
}
println(mapper.getClass.getName)

val conf = new JobConf(this.getClass)
conf.setJobName("Word Count without Buffering")
FileInputFormat.addInputPath(conf, new Path(inputPath))
FileOutputFormat.setOutputPath(conf, new Path(outputPath))

conf.setMapperClass(mapper)
conf.setReducerClass(classOf[Reduce])

```

Problems @ Javadoc Declaration

0 items

Description	Resource	Path	Location	Type

Writable Smart Insert Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Exercise: Hive-5-Joins

Built-in and User Defined Functions

[https://cwiki.apache.org/confluence/display/Hive/
LanguageManual+UDF](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF)



Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

Hive documentation labels ALL functions UDFs, including built-ins!

UDFs

```
hive> SHOW FUNCTIONS;  
!  
!=  
...  
abs  
acos  
...  
year  
...
```

UDFs;

```
hive> DESCRIBE FUNCTION year;  
year(date) - Returns the year of date
```

```
hive> DESCRIBE FUNCTION EXTENDED year;  
year(date) - Returns the year of date  
date is a string in the format of 'yyyy-MM-  
dd HH:mm:ss' or 'yyyy-MM-dd'.
```

Example:

```
> SELECT year('2009-30-07', 1) FROM src  
LIMIT 1;  
2009
```

User Defined Function

(UDF)

- Works on a single row.
- One or more columns or results from other UDFs.
- Example from Hive's built-in UDFs:

```
SELECT year(ymd) FROM stocks;
```

User Defined Aggregate Function (UDAF)

- Takes a collection of rows or values and aggregates them into a new row or value.
- Example from Hive's built-in UDFs:

```
SELECT year(ymd),  
       avg(price_close) FROM stocks  
      WHERE symbol = 'AAPL'  
    GROUP BY year(ymd);
```

User Defined Table Generating Function (UDTF)

- Takes a single row of values and generates multiple output rows, effectively a new table.
- Example from Hive's built-in UDTFs:

```
SELECT explode(subordinates) AS subs  
FROM employees;
```

Required Column
Alias!

Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

User Defined Table Generating Function

(UDTF)

- Limitations:

- No other columns allowed in SELECT.
- UDTFs can't be nested.
- GROUP BY, CLUSTER BY, etc. not supported.

Lateral Views

- More flexible way to use UDTFs:

```
SELECT name, sub
FROM employees
LATERAL VIEW explode(subordinates)
subView AS sub;
```

Lateral Views

- More flexible way to use UDTFs:

```
SELECT name, sub  
FROM employees  
LATERAL VIEW explode(subordinates)  
subView AS sub;
```

Whole FROM
Clause

View Alias

Column Alias

Have some limitations so they aren't used a lot. There is a feature we aren't discussing called Lateral Views (<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+LateralView>) that provides more flexibility for similar computations.

Writing Custom UDFs

```
// Java
import org.apache.hadoop.hive.ql.exec.UDF;

public class NowUDF extends UDF {
    public long evaluate() {
        return System.currentTimeMillis();
    }
}
```

- You compile this Java code and build a jar file...

Writing Custom UDFs

-- HQL

```
ADD JAR path_to_jar;
```

There is no CREATE
PERMANENT
FUNCTION...

```
CREATE TEMPORARY FUNCTION now  
AS 'com...NowUDF';
```

```
SELECT epoch_millis FROM ...  
WHERE epoch_millis < now() ...;
```

```

WordCount.scala
▶ S WordCountBuffering.scala
▶ S WordCountBufferingFlushing.scala
▶ S WordCountNoBuffering.scala
▶ S WordCountNoBufferingTokenizing.scala
▶ S WordCountReduce.scala
▶ Referenced Libraries
  ▶ Scala Library [2.9.1.final]
  ▶ JRE System Library [Java SE 6 (MacOS X)]
  ▶ lib
  ▶ lib_managed
  ▶ project
  ▶ src
  ▶ target
    all-shakespeare.txt
    README.textile
    results
    run.sh
    sbt
    setup.sh
    WordCount.iml

private[mapper] val settings = one of the following options:
1 | no | no-buffer      Simplest algorithm, but least efficient.
2 | not | no-buffer-use-tokenizer Like 'no', but uses a less efficient
3 | buffer            Buffer the counts and emit just one key-count pair
4 | buffer-flush     Like 'buffer', but flushes data more often to limit
and
-c | --use-combiner Use the reducer as a combiner."""

def main(argv: Array[String]): Unit = {
  val (mapper, useCombiner, inputPath, outputPath) = parseArgs(argv)
  case Settings(Some(m), useC, Some(in), Some(out)) => (m, useC, in, out)
  case _ => error("Invalid settings returned by parseArgs for main")
  println(mapper.getClass.getName)

  val conf = new JobConf(this.getClass)
  conf.setJobName("Word Count without Buffering")
  FileInputFormat.addInputPath(conf, new Path(inputPath))
  FileOutputFormat.setOutputPath(conf, new Path(outputPath))

  conf.setMapperClass(mapper)
  conf.setReducerClass(classOf[Reduce])
}

object Settings {
  type Mapper = String
  type UseCombiner = Boolean
  type InputPath = Path
  type OutputPath = Path
}

```

Problems @ Javadoc Declaration

0 items

Description	Resource	Path	Location	Type

Writable Smart Insert Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

File and Record Formats

<https://cwiki.apache.org/confluence/display/Hive/SerDe>

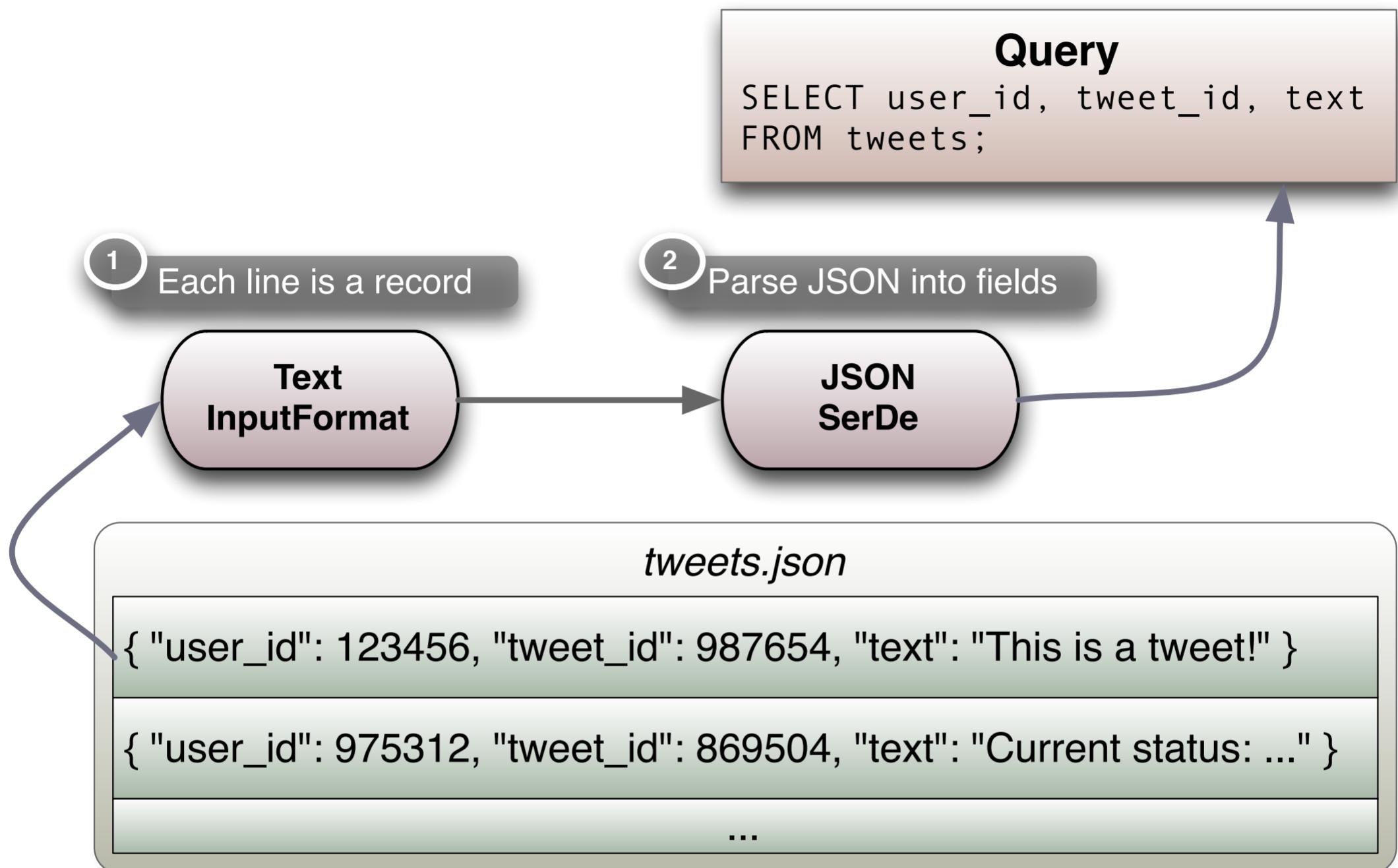


Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

How to support new file and record formats. We have used the defaults almost exclusively today, but the file and record formats, and whether or not (and how) you compress files have important benefits for disk space and network IO overhead, MR performance, how easy it is to work with files using other tools (either within or outside the Hadoop “ecosystem”), etc. These choices are usually made by the IT team when architecting the whole system and particular data usage scenarios. We discuss these choices in depth in our Developer Course aimed at Java Developers. Also, the Bonus Material section contains an expanded version of this section.

File and Record Formats



File and *Record* Formats

- INPUTFORMAT and OUTPUTFORMAT:
 - How *records* are stored in *files* and query results are written.
- SERDE: (serializer-deserializer)
 - How *records* are stored in *columns*.

This is an important distinction; how records are encoded in files and how columns/fields are encoded in records.

INPUTFORMATs are responsible for splitting an input stream into records.

OUTPUTFORMATs are responsible for writing records to an output stream (i.e., query results).

Two separate classes are used.

SERDEs are responsible for tokenizing a record into columns/fields and also encoding columns/fields into records. Unlike the *PUTFORMATs, there is one class for both tasks.

Built-in File Formats

- The default is TEXTFILE.

```
CREATE TABLE tbl_name (col1 TYPE, ...)
```

...

```
STORED AS TEXTFILE;
```

- There are other built-in formats...

We have been using TEXTFILE, the default, all day.

We'll discuss several of the most common options, but there are many more to choose from. In your projects, the whole development team will want to pick the most appropriate formats that balance the various concerns of disk space and network utilization, sharing with other tools, etc.

Built-in File Formats

- SEQUENCEFILE is a binary, space-efficient format supported by Hadoop.

```
CREATE TABLE tbl_name (col1 TYPE, ...)
```

...

```
STORED AS SEQUENCEFILE;
```

- Easiest to use with pre-existing SEQUENCEFILEs or INSERT ... SELECT.

Built-in File Formats

- Enable SEQUENCEFILE block compression.

```
SET io.seqfile.compression.type=BLOCK;  
CREATE TABLE tbl_name (col1 TYPE, ...)  
...  
STORED AS SEQUENCEFILE;
```

- *BZip2* supports block compression!

We won't discuss file compression in more detail here. (Our Hadoop training for Java Developers covers this topic in depth.) Compression gives further space and network IO savings. Compressing files by "block" (chunks of rows or bytes), rather than all at once has important practical consequences for MapReduce's ability to split a file into "splits", where each split is sent to a separate Map process. If a file can't be split, then no matter how big it is, it has to be sent to one task, reducing the benefit of a cluster! Block compressed files can be split by MR on block boundaries. Not all compression schemes support block compression. BZip2 does, but GZip does not. SEQUENCEFILEs lend themselves well to block compression.

Custom File Formats

- You might have your data in a *custom format*.

```
CREATE TABLE tbl_name (col1 TYPE, ...)
```

...

```
STORED AS INPUTFORMAT '...' OUTPUTFORMAT '...';
```

Custom File Formats

- Must specify both INPUTFORMAT and OUTPUTFORMAT.

```
CREATE TABLE tbl_name (col1 TYPE, ...)
```

...

STORED AS
INPUTFORMAT

The Hive defaults
for TEXTFILE

'org.apache.hadoop.mapreduce.lib.input.TextInputFormat'

OUTPUTFORMAT

Used for query results

'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat' ;

Note that the default INPUTFORMAT is a general Hadoop MapReduce type, while the OUTPUTFORMAT is Hive-specific type.

If you specify INPUTFORMAT, you must also specify OUTPUTFORMAT.

SerDes: JSON “Records”

```
CREATE TABLE tbl_name (id BIGINT, ...)
```

...

```
STORED AS  
ROW FORMAT SERDE
```

A SerDe for JSON

```
'org.apache.hadoop.hive.contrib.serde2.JsonSerde'  
WITH SERDEPROPERTIES (  
    "id"=".id",  
    "user"=".user.name", ...)  
) ;
```

Mechanism for
configuring SerDes
(when they
support it.)

JSON SerDe

- There are several versions of this “contrib” *JSON Serde*.
- *Think Big Analytics* extended it to support SERDEPROPERTIES (among other things).

<https://github.com/thinkbiganalytics/hive-json-serde>

The second project has an interesting JavaBean introspection library that makes it very easy to provide for more dynamic schemas, depending on what JavaBean properties are introspected.

We used the hive-json-serde just now, of course.

```

WordCountJob.scala
WordCountBuffering.scala
WordCountBufferingFlushing.scala
WordCountNoBuffering.scala
WordCountNoBufferingTokenizer.scala
WordCountReduce.scala
Referenced Libraries
Scala Library [2.9.1.final]
JRE System Library [Java SE 6 (MacOS)
lib
lib_managed
project
src
target
all-shakespeare.txt
README.textile
results
run.sh
sbt
setup.sh
WordCount.iml

private[mapper] val settings = one of the following options:
1 | no | no-buffer    Simplest algorithm, but least efficient.
2 | not | no-buffer-use-tokenizer Like 'no', but uses a less efficient
3 | buffer          Buffer the counts and emit just one key-count pair
4 | buffer-flush    Like 'buffer', but flushes data more often to limit
and
-c | --use-combiner Use the reducer as a combiner."
def main(argv: Array[String]): Unit = {
  val (mapper, useCombiner, inputPath, outputPath) = parseArgs(argv)
  case Settings(Some(m), useC, Some(in), Some(out)) => (m, useC, in, out)
  case _ => error("Invalid settings returned by parseArgs for main")
  println(mapper.getClass.getName)

  val conf = new JobConf(this.getClass)
  conf.setJobName("Word Count without Buffering")
  FileInputFormat.addInputPath(conf, new Path(inputPath))
  FileOutputFormat.setOutputPath(conf, new Path(outputPath))

  conf.setMapperClass(mapper)
  conf.setReducerClass(classOf[Reduce])
}

object WordCount extends Application
{
  def main(argv: Array[String]): Unit = {
    val (mapper, useCombiner, inputPath, outputPath) = parseArgs(argv)
    case Settings(Some(m), useC, Some(in), Some(out)) => (m, useC, in, out)
    case _ => error("Invalid settings returned by parseArgs for main")
    println(mapper.getClass.getName)

    val conf = new JobConf(this.getClass)
    conf.setJobName("Word Count without Buffering")
    FileInputFormat.addInputPath(conf, new Path(inputPath))
    FileOutputFormat.setOutputPath(conf, new Path(outputPath))

    conf.setMapperClass(mapper)
    conf.setReducerClass(classOf[Reduce])
  }
}

```

Exercise:
Hive-7-SerDe

Back to *UDFs*: *N-Grams* and *Statistics*



Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

Let's look at a few more built-in functions, for doing n-gram analysis of text, an important tool for natural-language processing (machine learning), and also statistics, specifically histogram calculations.

```

WordCountWordCount
WordCountBuffering.scala
WordCountBufferingFlushing.scala
WordCountNoBuffering.scala
WordCountNoBufferingTokenizing.scala
WordCountReduce.scala
Referenced Libraries
Scala Library [2.9.1.final]
JRE System Library [Java SE 6 (MacOS X)]
lib
lib_managed
project
src
target
all-shakespeare.txt
README.textile
results
run.sh
sbt
setup.sh
WordCount.iml

private[mapper] val argMapper: String = args(0)
private[mapper] val argReducer: String = args(1)

private[mapper] def main(argv: Array[String]): Unit = {
    val (mapper, useCombiner, inputPath, outputPath) = parseArgs(argv)
    val Settings(m, useC, Some(in), Some(out)) = Settings(m, useC, inputPath, outputPath)
    case _ => error("Invalid settings returned by parseArgs for main")
}

private[mapper] def parseArgs(args: Array[String]): (String, Boolean, Path, Path) = {
    val (m, useC, in, out) = args match {
        case Array(m, useC, in, out) if useC == "true" & in != null & out != null =>
            (m, true, Path(in), Path(out))
        case Array(m, useC, in, out) if useC == "false" & in != null & out != null =>
            (m, false, Path(in), Path(out))
        case _ => error("Usage: MapperClass [useCombiner] InputPath OutputPath")
    }
    (m, useC, in, out)
}

private[mapper] object Settings {
    def apply(m: String, useC: Boolean, in: Path, out: Path): Settings = new Settings(m, useC, in, out)
}

private[mapper] class Settings(m: String, useC: Boolean, in: Path, out: Path) {
    def getClass: Class[Mapper] = mapper.getClass
}

private[mapper] class Mapper extends Reducer[Text, IntWritable, Text, IntWritable] {
    private[mapper] var count = 0
    private[mapper] var previousKey: Option[Text] = None
    private[mapper] var buffer: mutable.StringBuilder = new mutable.StringBuilder()

    private[mapper] def reduce(key: Text, values: Iterator[IntWritable], context: Reducer.Context): Unit = {
        if (key != previousKey) {
            if (previousKey.isDefined) {
                context.write(previousKey.get, count)
            }
            count = 0
            previousKey = Some(key)
        }
        values.foreach(v => count += v.get)
    }

    private[mapper] def map(key: Text, value: IntWritable, context: Mapper.Context): Unit = {
        if (useCombiner) {
            context.write(key, value)
        } else {
            buffer.append(key + "\n")
            buffer.append(value.toString + "\n")
        }
    }

    private[mapper] def write(context: Reducer.Context): Unit = {
        if (useCombiner) {
            context.write(previousKey.get, count)
        } else {
            context.write(previousKey.get, buffer.toString)
        }
    }
}

```

Exercise:

Hive-8-NGrams-Stats

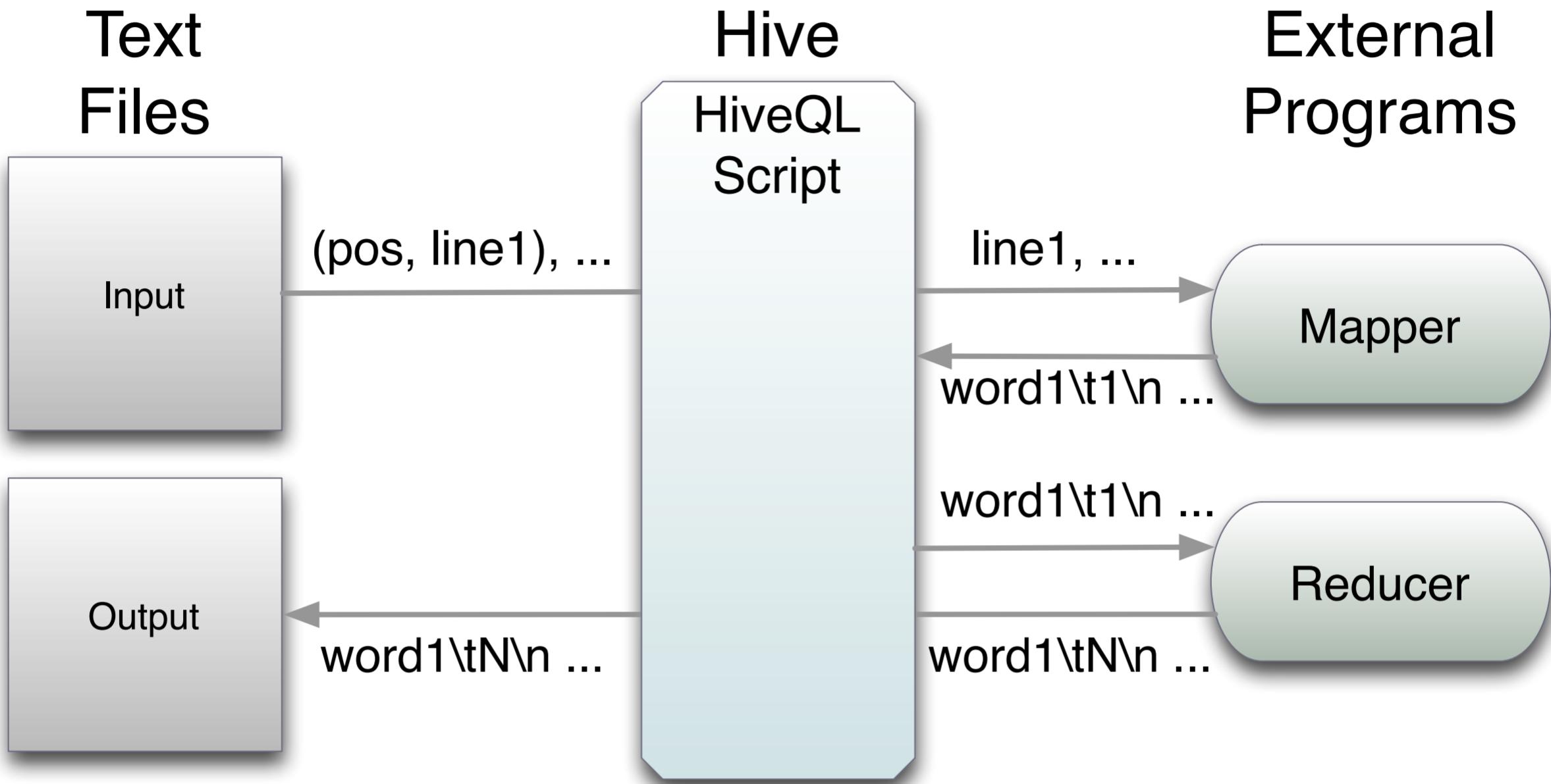
Hive

Select Transform or Map Reduce

Syntax

[https://cwiki.apache.org/confluence/display/Hive/
LanguageManual+Transform](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Transform)

Calling External Scripts



Select Transform or Map Reduce

- A technique for calling out to external programs to perform map and reduce operations.

Word Count Example

- We'll use *mapper* and *reducer* scripts written in *Python* to compute the *Word Count* for Shakespeare's plays.

Word Count Example

```
# mapper.py  
import sys
```

```
for line in sys.stdin:  
    words = line.strip().split()  
    for word in words:  
        print "%s\t1" % (word.lower())
```

The *details* don't matter for our purposes.

Don't worry if you don't know Python. We read input from "stdin" ("standard in"), a line at a time. For each line, we "strip" whitespace off the beginning and end, then "split" the string on whitespace to create a list of words. Finally, we iterate through the words and print the word in lower case, followed by a tab character, followed by the number 1. (We could rewrite the last line as

```
print "%s\t1" % (word.lower())
```

Word Count Example

- The *reducer.py* script:
 - Each key-value pair will be on a *separate* line in the input, but the *keys* will be *sorted*
 - So, we'll see this:

word1	1
word1	1
word2	1
word2	1
word2	1
word3	1

...

Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

Don't worry if you don't know Python. We read input from "stdin" ("standard in"), a line at a time. For each line, we "strip" whitespace off the beginning and end, then "split" the string on whitespace to create a list of words. Finally, we iterate through the words and print the word in lower case, followed by a tab character, followed by the number 1. (We could rewrite the last line as

```
print "%s\t1" % (word.lower())
```

Word Count Example

```
# reducer.py
import sys
(last_key, last_count) = (None, 0)
for line in sys.stdin:
    (key, count) = line.strip().split("\t")
    if last_key and last_key != key:
        print "%s\t%d" % (last_key, last_count)
        (last_key, count) = (key, int(count))
    else:
        last_key = key
        last_count += int(count)
if last_key:
    print "%s\t%d" % (last_key, last_count)
```

Mostly *bookkeeping details.*

This one is obviously more complicated, mostly because of the way we received the key-value pairs from the map process, as discussed on the previous slide. We have to keep track of the last key (from the previous line), so we can detect when we start getting lines with the next key. We also track the count for the last key. We loop through the input lines, removing leading and trailing whitespace, then splitting on the tab character to extract the current key and count (which is always 1 for our case).

Next, we test to see if the new key is different from the last key. If so, we print out the last key and its total count, then reset the last key and count to be what we just read in.

Otherwise, if we're still receiving the same key, we update the count and make sure the `last_key` is assigned (needed for the very first line on input). Finally, after the loop finishes, we write out whatever is left, if any.

(There are certainly improvements you could make to this script, like extracting the “print” statement into a function...)

Word Count Example

- First we need tables for the input text and output word count:

```
CREATE EXTERNAL TABLE  
    shakespeare_plays (line STRING)  
LOCATION '/data/shakespeare/input';
```

```
CREATE TABLE shakespeare_plays_wc (  
    word STRING, count INT)  
ROW FORMAT  
DELIMITED FIELDS TERMINATED BY '\t';
```

The “input” directory contains a file with all of Shakespeare’s plays. We will treat each line as a “record” with one field – the line of text.

The output table will have word-count records, a text file separated by tabs, which makes it easy to use with the Python scripts.

Word Count Example

```
ADD FILE ../../mapper.py;
ADD FILE ../../reducer.py;
FROM (
    FROM shakespeare_plays
    MAP line USING 'mapper.py'
    AS word, count
    CLUSTER BY word) wc
INSERT OVERWRITE TABLE shakespeare_plays_wc
REDUCE wc.word, wc.count USING 'reducer.py'
AS word, count;
```

Word Count Example

```
ADD FILE ../../mapper.py;
ADD FILE ../../reducer.py;
FROM (
  FROM shakespeare_plays
  MAP line USING 'mapper.py'
  AS word, count
  CLUSTER BY word) wc
```

Copy scripts
around cluster

Nested
query.

Must cluster
the map
output

```
INSERT OVERWRITE TABLE shakespeare_plays_wc
REDUCE wc.word, wc.count USING 'reducer.py'
AS word, count;
```

There is no SELECT clause; every “field” output by the MAP step will be “selected”.
The map step is invoked in a nested query of the same FROM form.
It is necessary to here to cluster the map output by word, which means we want all
(word,count) pairs for a given value of “word” to go to the same reducer step invocation, a
requirement for the WordCount algorithm to work.

Word Count Example

```
ADD FILE ./mapper.py;
ADD FILE ./reducer.py;
FROM (
  FROM shakespeare_plays
  MAP line USING 'mapper.py'
  AS word, count
  CLUSTER BY word) wc
INSERT OVERWRITE TABLE shakespeare_plays_wc
REDUCE wc.word, wc.count USING 'reducer.py'
AS word, count;
```

The *map* step.

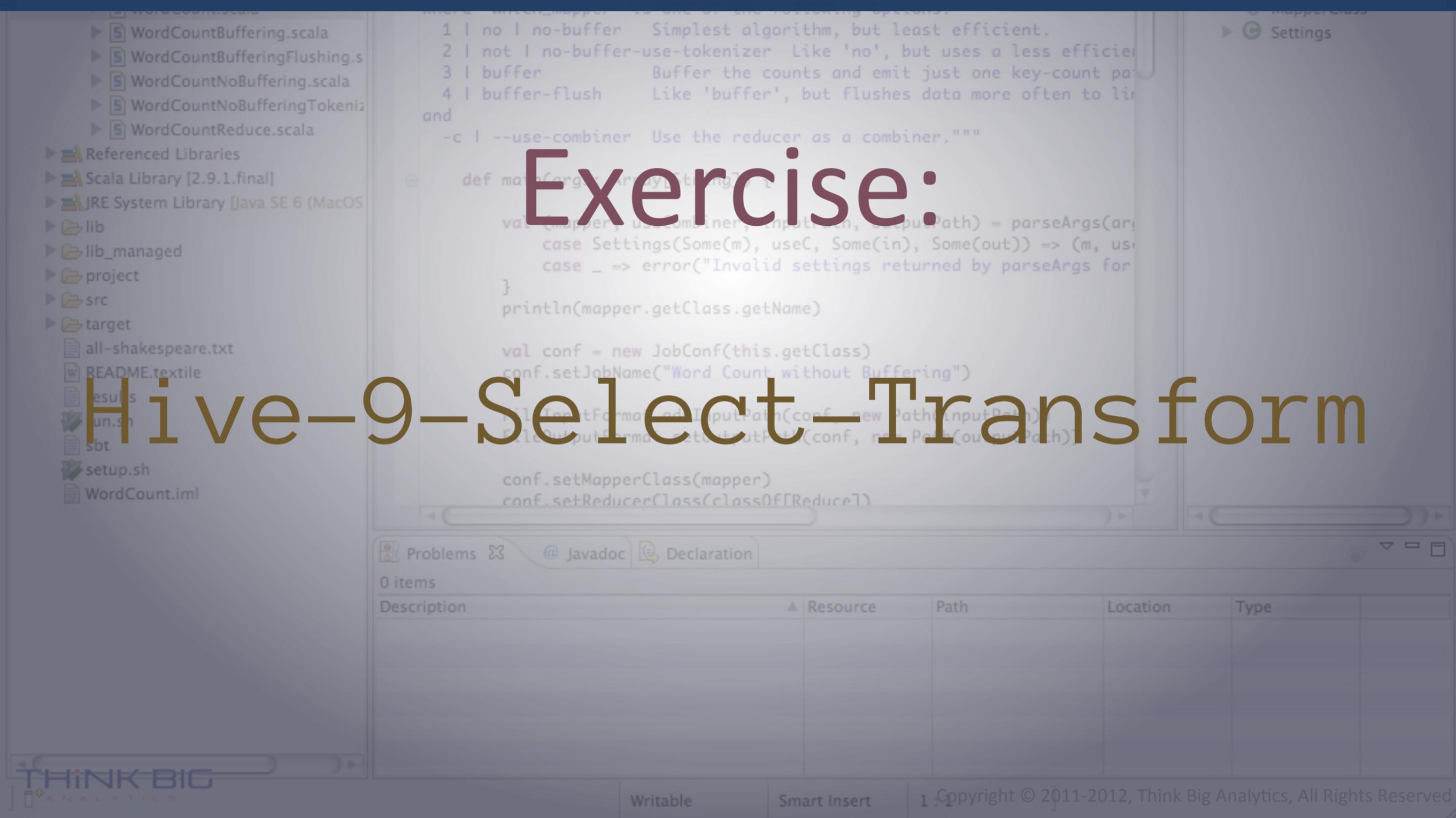
The *reduce* step.

Word Count Example

```
ADD FILE ./mapper.py;
ADD FILE ./reducer.py;
FROM (
    FROM shakespeare_plays
    SELECT TRANSFORM (line) USING 'mapper.py'
    AS word, count
    CLUSTER BY word) wc
INSERT OVERWRITE TABLE shakespeare_plays_wc
    SELECT TRANSFORM (wc.word, wc.count)
    USING 'reducer.py'
    AS word, count;
```

The more generic
SELECT
TRANSFORM
syntax.

Using MAP... and
REDUCE... is
actually *misleading*.



```

WordCountMapper
WordCountBuffering.scala
WordCountBufferingFlushing.scala
WordCountNoBuffering.scala
WordCountNoBufferingTokenizer.scala
WordCountReduce.scala
Referenced Libraries
Scala Library [2.9.1.final]
JRE System Library [Java SE 6 (MacOS)
lib
lib_managed
project
src
target
all-shakespeare.txt
README.textile
results
run.sh
sbt
setup.sh
WordCount.iml

private[mapper] val settings = one of the following options:
1 | no | no-buffer      Simplest algorithm, but least efficient.
2 | not | no-buffer-use-tokenizer Like 'no', but uses a less efficient
3 | buffer            Buffer the counts and emit just one key-count pair
4 | buffer-flush     Like 'buffer', but flushes data more often to limit
and
-c | --use-combiner Use the reducer as a combiner."""

def main(argv: Array[String]): Unit = {
    val (mapper, useCombiner, inputPath, outputPath) = parseArgs(argv)
    case Settings(Some(m), useC, Some(in), Some(out)) => (m, useC, in, out)
    case _ => error("Invalid settings returned by parseArgs for main")
}
println(mapper.getClass.getName)

val conf = new JobConf(this.getClass)
conf.setJobName("Word Count without Buffering")
FileInputFormat.setInputPath(conf, new Path(inputPath))
FileOutputFormat.setOutputPath(conf, new Path(outputPath))

conf.setMapperClass(mapper)
conf.setReducerClass(classOf[Reduce])

```

Problems @ Javadoc Declaration

0 items

Description	Resource	Path	Location	Type

Writable Smart Insert Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Exercise:

Hive-9-Select-Transform

Thanks for attending!

Questions?

info@thinkbiganalytics.com

<http://thinkbiganalytics.com>

Copyright © 2011-2012, Think Big Analytics, All Rights Reserved

Tuesday, October 23, 12

