GENERAL ASSEMBLY

INTRO TO DATA SCIENCE

LECTURE 16: MAP-REDUCE

I. BIG DATA

II. HADOOP ECOSYSTEM

III. MAP-REDUCE PROGRAMMING MODEL

EXERCISE:

V. MAP-REDUCE USING PIG

I. BIG DATA

As you have probably heard, **big data** *is a hot topic these days.*

Q: What does "big data" actually refer to?

As you have probably heard, **big data** *is a hot topic these days.*

Q: What does “big data” actually refer to?

A: Scalability; in particular, storing & processing web-scale (multi-terabyte) datasets…

One approach would be to get a huge supercomputer.

One approach would be to get a huge supercomputer.

But this has some obvious drawbacks:

- *expensive*
- *difficult to maintain*
- *scalability is bounded*

Instead of one huge machine, what if we got a bunch of regular (commodity) machines?

Instead of one huge machine, what if we got a bunch of regular (commodity) machines?

This has obvious benefits!

- cheaper

- easier to maintain

- scalability is unbounded (just add more nodes to the cluster)

Now we can give a complete answer to our earlier question.

Q: What does "big data" actually refer to?

Now we can give a complete answer to our earlier question.

Q: What does "big data" actually refer to?

A: Scalability; in particular, storing & processing web-scale (multi-terabyte) datasets using clusters of multiple computing nodes.

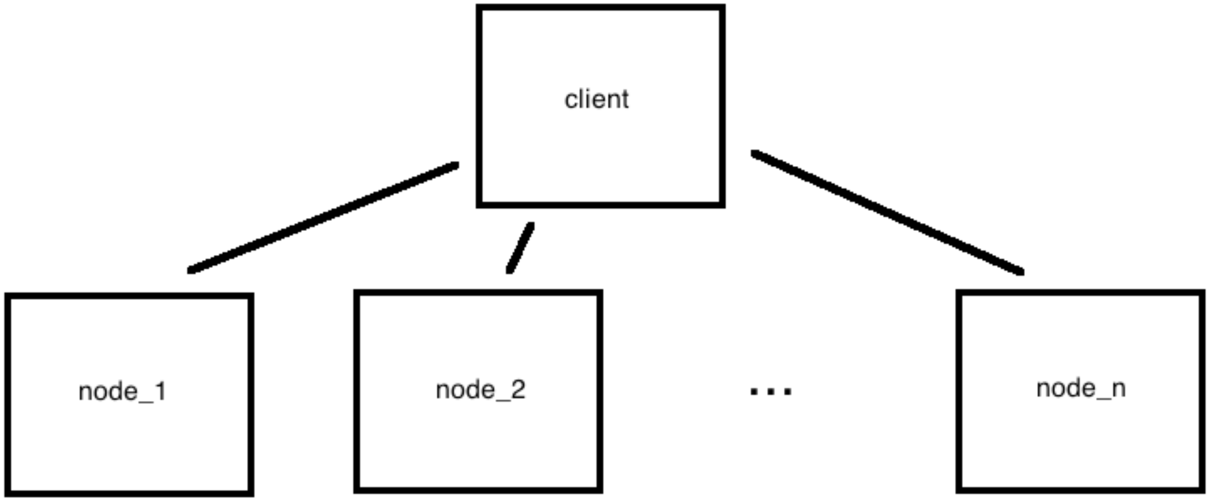
Now we can give a complete answer to our earlier question.

Q: What does "big data" actually refer to?

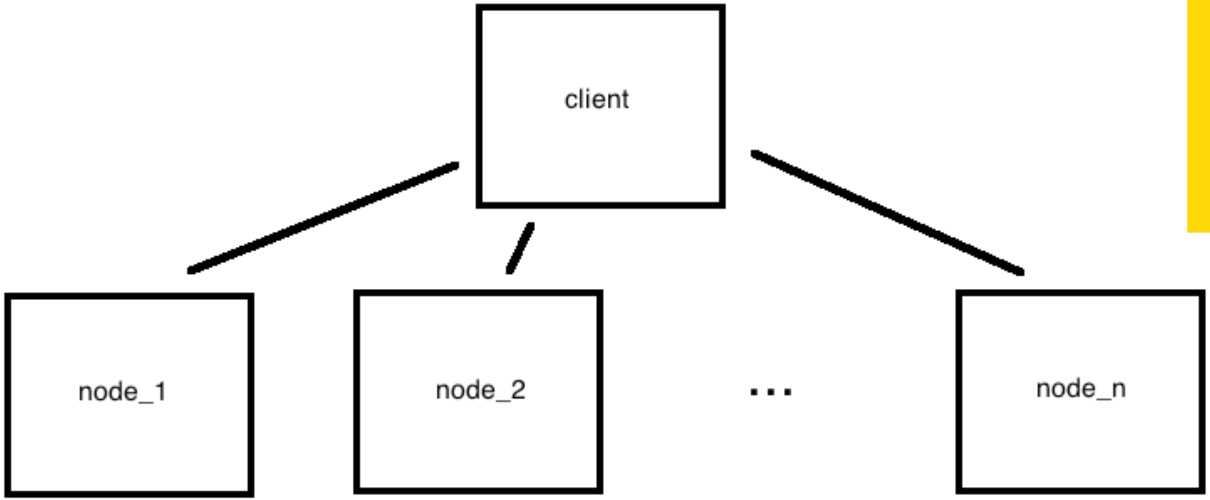
A: Scalability; in particular, storing & processing web-scale (multi-terabyte) datasets using clusters of multiple computing nodes.

"Scale out vs scale up!"

We can visualize this horizontal cluster architecture as a single client-multiple server relationship



We can visualize this horizontal cluster architecture as a single client-multiple server relationship



NOTE

A horizontally distributed system also has better fault tolerance than a single machine.

How do we process data in a distributed architecture?

- move code to data

- map-reduce → less overhead (network traffic, disk I/O)

“Computing nodes are the same as storage nodes.”

ASIDE: DIVIDE AND CONQUER ALGORITHMS

Divide and conquer is a fundamental algorithmic technique for solving a given task, whose steps include:

Divide-and-conquer *is a fundamental algorithmic technique for solving a given task, whose steps include:*

1) *split task into subtasks*
2) *solve these subtasks independently*
3) *recombine the subtask results into a final result*

ASIDE: DIVIDE AND CONQUER ALGORITHMS

Map-reduce leverages the **divide-and-conquer** *approach by splitting a large dataset into several smaller datasets and performing a computation on each of these in parallel.*

ASIDE: DIVIDE AND CONQUER ALGORITHMS

The defining characteristic of a problem that is suitable for the divide and conquer approach is that it can be broken down into independent subtasks.

ASIDE: DIVIDE AND CONQUER ALGORITHMS

The defining characteristic of a problem that is suitable for the divide and conquer approach is that it can be broken down into independent subtasks.

Tasks that can be parallelized in this way include:

- *count, sum, average*
- *grep, sort, inverted index*
- *graph traversals, **some** ML algorithms*

II. HADOOP ECOSYSTEM

Hadoop *is a popular open-source Java-based implementation of the map-reduce framework (including file storage for input/output).*

HADOOP

Google	Open-source	Function
GFS	HDFS	Distributed file system
MapReduce	MapReduce	Batch distributed data processing
Bigtable	HBase	Distributed DB/key-value store
Protobuf/Stubby	Thrift or Avro	Data serialization/RPC
Pregel	Giraph	Distributed graph processing
Tenzing	Hive	Scalable SQL on MapReduce
Dremel/F1	Cloudera Impala	Scalable interactive SQL (MPP)
FlumeJava	Crunch	Abstracted data pipelines on Hadoop

Hadoop *is a popular open-source Java-based implementation of the map-reduce framework (including file storage for input/output).*

You can download Hadoop and configure a set of machines to operate as a map-reduce cluster, or you can run it as a service via Amazon's Elastic Map-Reduce.

Frequently when people say “map-reduce” they’re referring to Hadoop, but there are some exceptions:

Frequently when people say “map-reduce” they’re referring to Hadoop, but there are some exceptions:

- *many NoSQL databases support native map-reduce queries*
- *commercial distributions (Cloudera, MapR, etc)*
- *Google’s internal implementation*

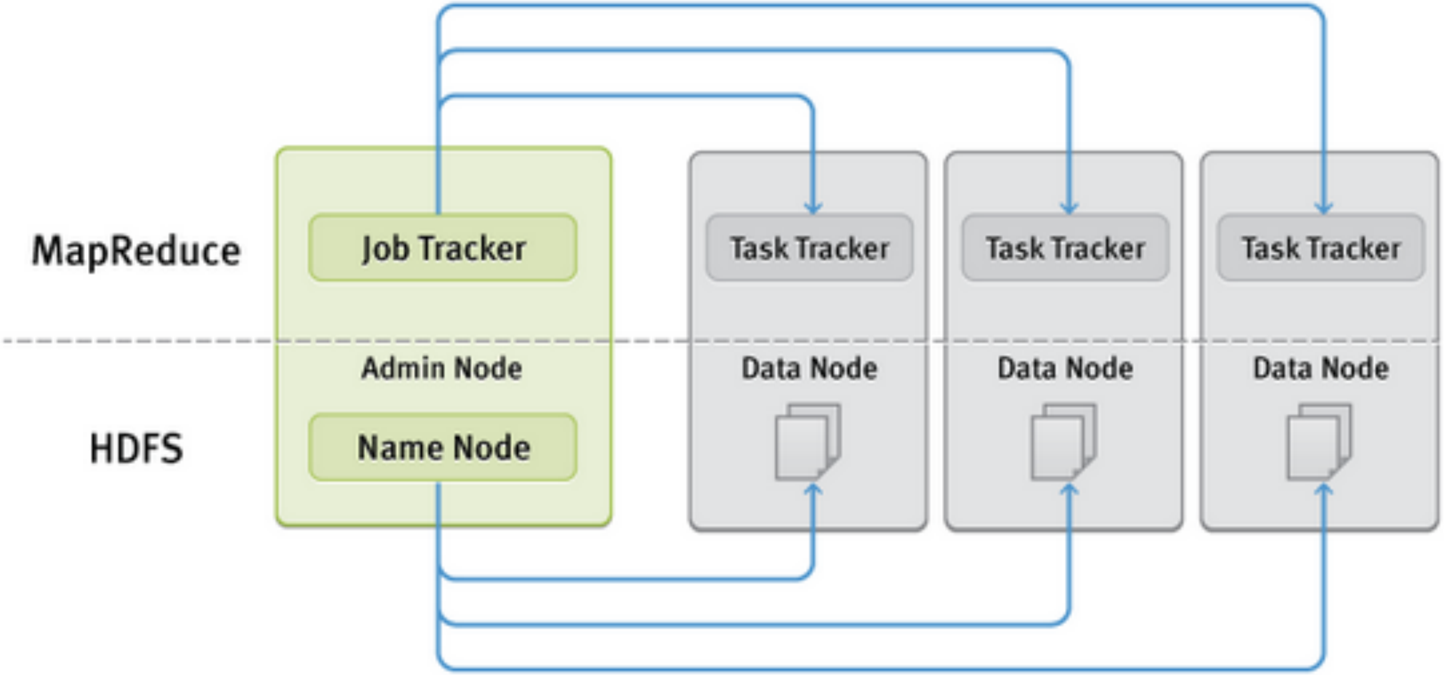
Hadoop *is a popular open-source Java-based implementation of the map-reduce framework (including file storage for input/output).*

Hadoop *is a popular open-source Java-based implementation of the map-reduce framework (including file storage for input/output).*

More often, Hadoop refers to the ecosystem of tools around distributed computing with two main components:

- distributed filesystem (HDFS)

- map-reduce job scheduler


MapReduce
HDFS
Job Tracker
Admin Node
Name Node
Task Tracker
Data Node
Task Tracker
Data Node
Task Tracker
Data Node

HADOOP

Google	Open-source	Function
GFS	HDFS	Distributed file system
MapReduce	MapReduce	Batch distributed data processing
Bigtable	HBase	Distributed DB/key-value store
Protobuf/Stubby	Thrift or Avro	Data serialization/RPC
Pregel	Giraph	Distributed graph processing
Tenzing	Hive	Scalable SQL on MapReduce
Dremel/F1	Cloudera Impala	Scalable interactive SQL (MPP)
FlumeJava	Crunch	Abstracted data pipelines on Hadoop

The Google File System (GFS) was developed alongside map-reduce to serve as the native file system for this type of processing.

Data is replicated in the (distributed) file system across several nodes.

Data is replicated in the (distributed) file system across several nodes.

This permits locality optimization (and fault tolerance) by allowing the mapper tasks to run on the same nodes where the data resides.

Data is replicated in the (distributed) file system across several nodes.

This permits locality optimization (and fault tolerance) by allowing the mapper tasks to run on the same nodes where the data resides.

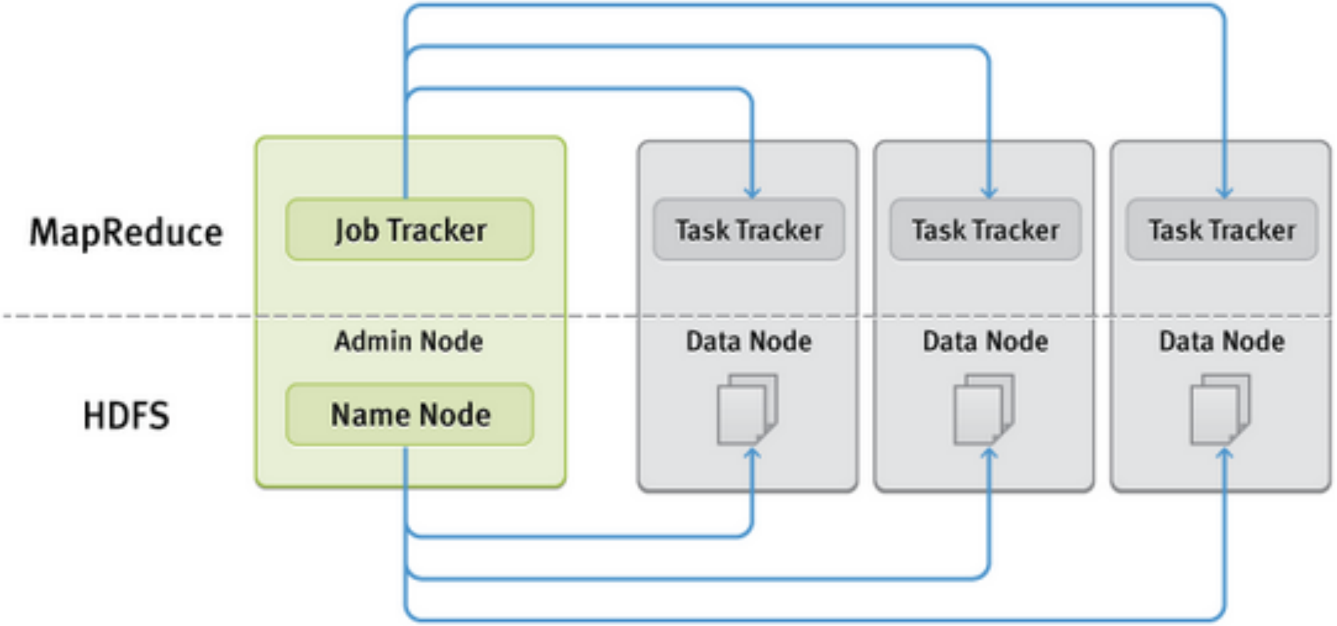
So we move code to data (instead of data to code), thus avoiding a lot of network traffic and disk I/O.

HDFS benefits include:

- Push compute tasks to data nodes to avoid data transfer

- Data replication: data is replicated so if a single machine fails another still contains the data

HADOOP



Source: http://www.ndm.net/datawarehouse/Greenplum/hadoop-components hulu™

Log storage and analysis

last.fm

Used for charts calculation, royalty reporting, log analysis, A/B testing, dataset merging

Also used for large scale audio feature analysis over millions of tracks

The New York Times
NYTIMES.COM

Large-scale image conversions

IMPLEMENTATION DETAILS

The map-reduce framework handles a lot of messy details for you:

IMPLEMENTATION DETAILS

The map-reduce framework handles a lot of messy details for you:

- parallelization & distribution (eg, input splitting)

- partitioning (shuffle/sort/redirect)

- fault-tolerance (fact: tasks/nodes will fail!)

- I/O scheduling

- status and monitoring

The map-reduce framework handles a lot of messy details for you:

- parallelization & distribution (eg, input splitting)

- partitioning (shuffle/sort/redirect)

- fault-tolerance (fact: tasks/nodes will fail!)

- I/O scheduling

- status and monitoring

This (along with the functional semantics) allows you to focus on solving the problem instead of accounting & housekeeping details.

It's possible to overlay the map-reduce framework with an additional declarative syntax.

This makes operations like select & join easier to implement and less error prone.

Popular examples include Pig and Hive.

HADOOP

Google	Open-source	Function
GFS	HDFS	Distributed file system
MapReduce	MapReduce	Batch distributed data processing
Bigtable	HBase	Distributed DB/key-value store
Protobuf/Stubby	Thrift or Avro	Data serialization/RPC
Pregel	Giraph	Distributed graph processing
Tenzing	Hive	Scalable SQL on MapReduce
Dremel/F1	Cloudera Impala	Scalable interactive SQL (MPP)
FlumeJava	Crunch	Abstracted data pipelines on Hadoop

Apache Hive

- SQL language to query data on HDFS

- Queries are translated behind the scenes into map-reduce jobs

- Data is stored on HDFS, but a metadata database contains the table schemas

Cloudera Impala

*- **ANOTHER** SQL language to query data on HDFS*

- Similar interface to Hive

BUT:

- Impala contains its own scheduling engine, queries are not translated map-reduce jobs

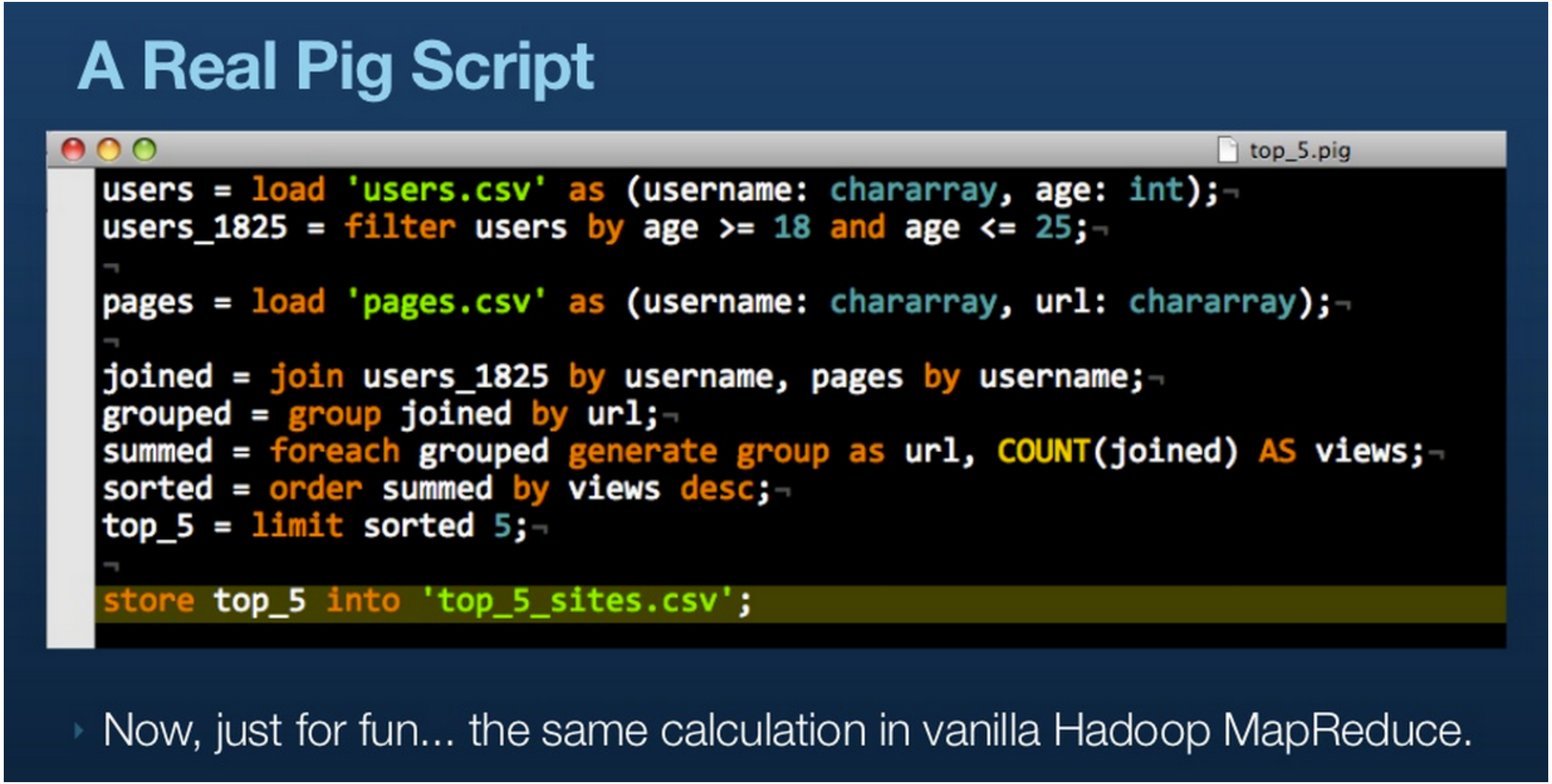
- Leads to faster queries, but no fault tolerance

MAP-REDUCE

Why Pig?

- Because I bet you can read the following script.

source: http://www.slideshare.net/kevinweil/hadoop-pig-and-twitter-nosql-east-2009



A Real Pig Script

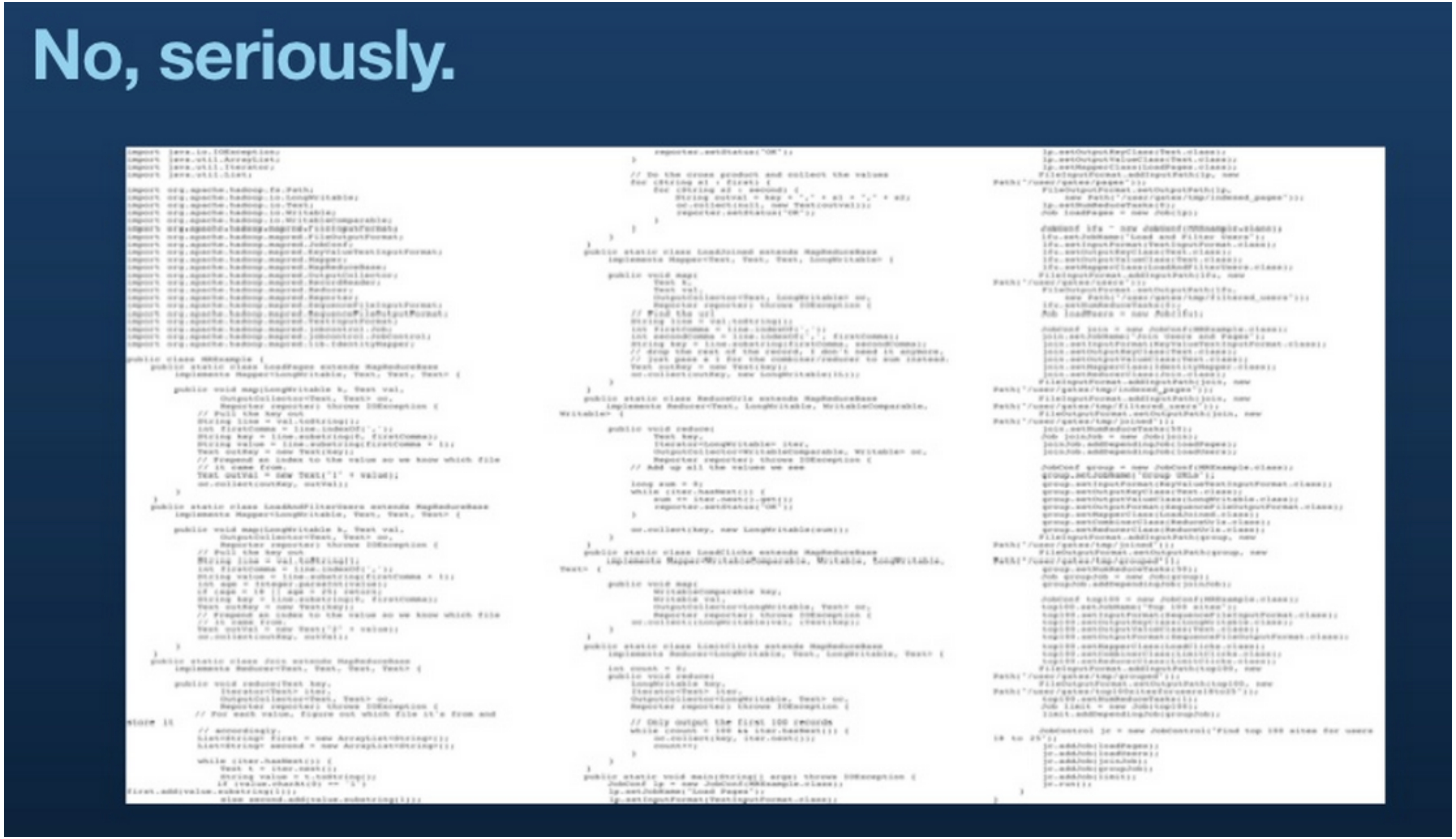
```
users = load 'users.csv' as (username: chararray, age: int);
users_1825 = filter users by age >= 18 and age <= 25;

pages = load 'pages.csv' as (username: chararray, url: chararray);

joined = join users_1825 by username, pages by username;
grouped = group joined by url;
summed = foreach grouped generate group as url, COUNT(joined) AS views;
sorted = order summed by views desc;
top_5 = limit sorted 5;

store top_5 into 'top_5_sites.csv';
```

- Now, just for fun... the same calculation in vanilla Hadoop MapReduce.



No, seriously.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

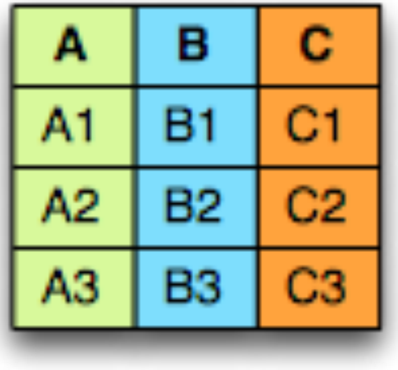
Parquet Data Storage

- *Nested columnar data storage*
- *Based on Google Dremel paper*
- *Open-sourced by Cloudera and Twitter in July 2013*

“Ideal for tables containing many columns, where most queries only refer to a small subset of the columns”

Parquet Data Storage

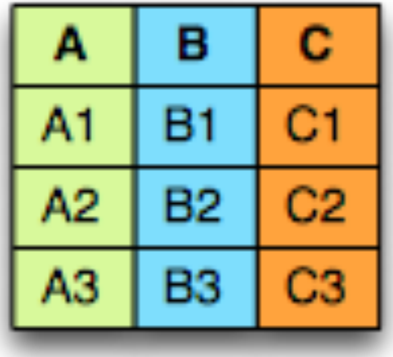
DATASET



A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

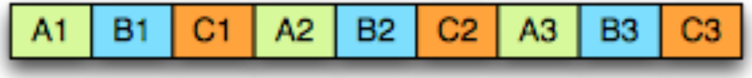
Parquet Data Storage

DATASET



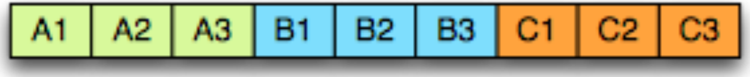
A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

ROW-ORIENTED STRUCTURE



A1	B1	C1	A2	B2	C2	A3	B3	C3

COLUMN-ORIENTED STRUCTURE



A1	A2	A3	B1	B2	B3	C1	C2	C3

Parquet Data Storage

Advantages

- *Limit I/O depending on columns queried*
- *Filter rows before reading all columns*
- *Efficient compression*

III. MAP-REDUCE PROGRAMMING

As we've discussed, the map-reduce approach involves splitting a problem into subtasks and processing these subtasks in parallel.

As we’ve discussed, the map-reduce approach involves splitting a problem into subtasks and processing these subtasks in parallel.

This takes place in two phases:

As we’ve discussed, the map-reduce approach involves splitting a problem into subtasks and processing these subtasks in parallel.

This takes place in two phases:

1) *the* **mapper** *phase*
2) *the* **reducer** *phase*

As we've discussed, the map-reduce approach involves splitting a problem into subtasks and processing these subtasks in parallel.

This takes place in (approximately) two phases:

1) the **mapper** *phase*
1.5) shuffle/sort
2) the **reducer** *phase*

Map-reduce uses a functional programming paradigm. The data processing primitives are mappers and reducers, as we've seen.

Map-reduce uses a functional programming paradigm. The data processing primitives are mappers and reducers, as we've seen.

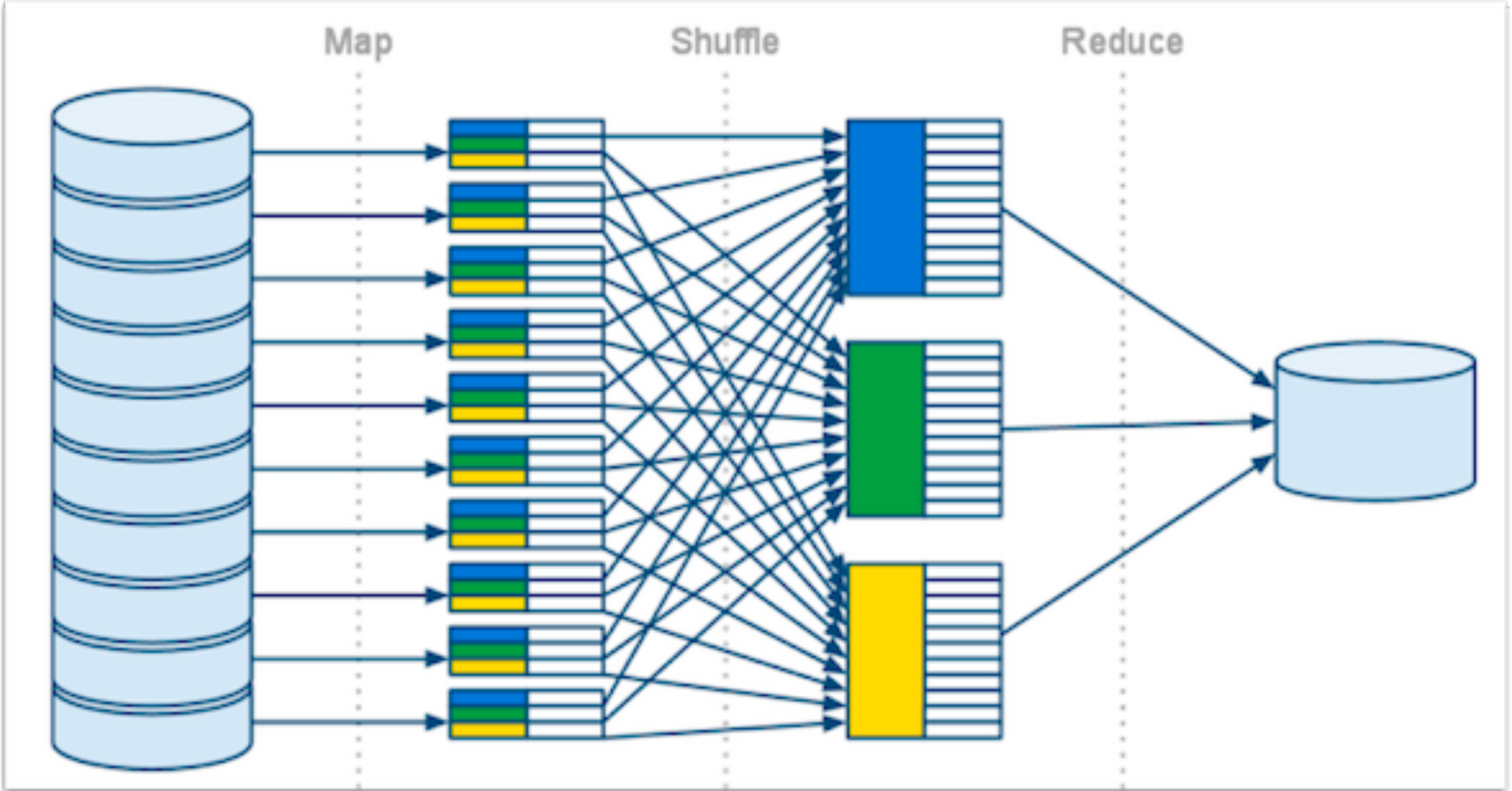
mappers – *filter & transform data*
reducers – *aggregate results*

Map-reduce uses a functional programming paradigm. The data processing primitives are mappers and reducers, as we've seen.

mappers – *filter & transform data*
reducers – *aggregate results*

The functional paradigm is good at describing how to solve a problem, but not very good at describing data manipulations (eg, relational joins).

MAP-REDUCE



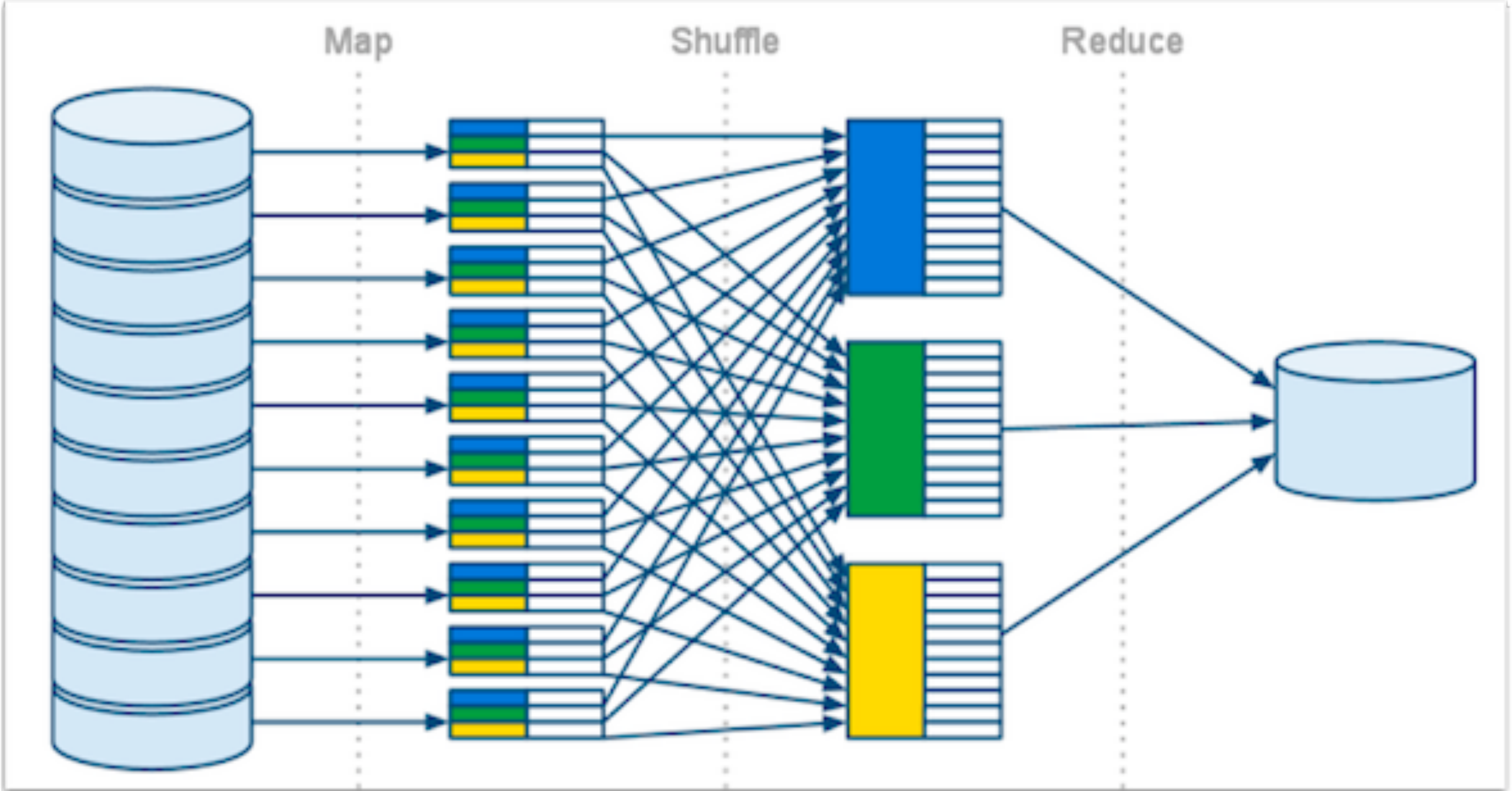
As our earlier diagram suggests, there are additional intermediate steps in a map-reduce workflow.

mappers – *filter & transform data*
reducers – *aggregate results*

As our earlier diagram suggests, there are additional intermediate steps in a map-reduce workflow.

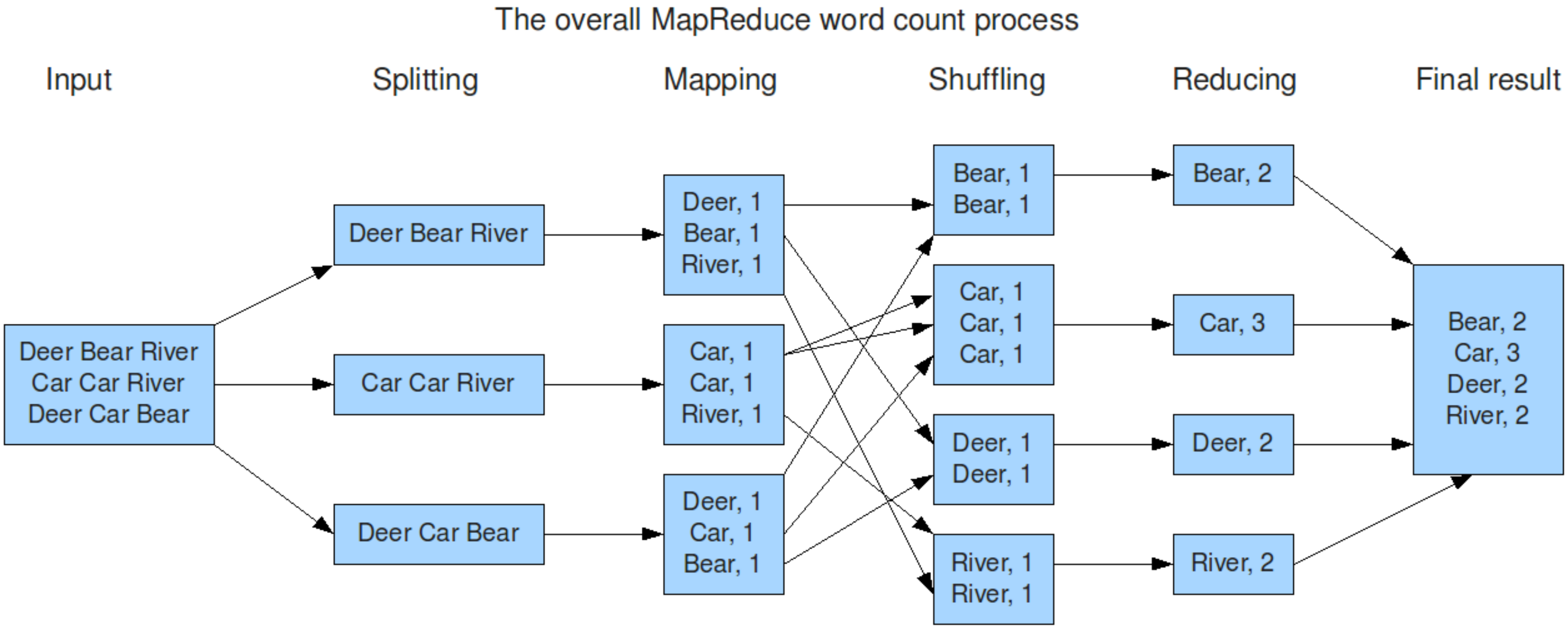
mappers – *filter & transform data*
combiners – *perform reducer operations on the mapper node (optional step, to reduce network traffic and disk I/O).*
partitioners – *shuffle/sort/redirect mapper output*
reducers – *aggregate results*

MAP-REDUCE



III. WORD COUNT EXAMPLE

MAP-REDUCE


The overall MapReduce word count process
Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

EXAMPLE

Map-reduce processes data in terms of key-value pairs:

```
input             <k1, v1>

mapper            <k1, v1> → <k2, v2>

(partitioner)     <k2, v2> → <k2, [all k2 values]>

reducer           <k2, [all k2 values]> → <k3, v3>
```

Using the following input, we can implement the “Hello World” of map-reduce: a word count.

MAP-REDUCE EXAMPLE: MAPPER INPUT

Using the following input, we can implement the "Hello World" of map-reduce: a word count.

```
where
where in
where in the
where in the world
where in the world is
where in the world is carmen
where in the world is carmen sandiego
```

MAP-REDUCE EXAMPLE: MAPPER

The first processing primitive is the mapper, which filters & transforms the input data, and emits transformed key-value pairs.

MAP-REDUCE EXAMPLE: MAPPER

The first processing primitive is the mapper, which filters & transforms the input data, and emits transformed key-value pairs.

```
mapper(k1, v1):
// k1 = line number
// v1 = line contents (eg, space–delimited string)

    words = tokenize(v1)   // split string into words
    for word in words:
        emit (word, 1)
```

MAP-REDUCE EXAMPLE: MAPPER OUTPUT

The mapper emits key-value pairs for each word encountered in the input data.

MAP-REDUCE EXAMPLE: MAPPER OUTPUT

The mapper emits key-value pairs for each word encountered in the input data.

```
where 1
where 1
in    1
where 1
in    1
the   1
...
```

MAP-REDUCE EXAMPLE: PARTITIONER

The partitioner is internal to the map-reduce framework, so we don't have to write this ourselves. It shuffles & sorts the mapper output, and redirects all intermediate results for a given key to a single reducer.

MAP-REDUCE EXAMPLE: PARTITIONER OUTPUT

The partitioner is internal to the map-reduce framework, so we don't have to write this ourselves. It shuffles & sorts the mapper output, and redirects all intermediate results for a given key to a single reducer.

```
where        [1, 1, 1, 1, 1, 1, 1]
in           [1, 1, 1, 1, 1, 1]
the          [1, 1, 1, 1, 1]
world        [1, 1, 1, 1]
is           [1, 1 ,1]
carmen       [1, 1]
sandiego     [1]
```

MAP-REDUCE EXAMPLE: REDUCER

Finally, the reducer receives all values for a given key and aggregates (in this case, sums) the results.

MAP-REDUCE EXAMPLE: REDUCER

Finally, the reducer receives all values for a given key and aggregates (in this case, sums) the results.

```
reducer(k2, k2_vals):
// k2 = word
// k2_vals = word counts

    emit k2, sum(k2_vals)
```

MAP-REDUCE EXAMPLE: REDUCER OUTPUT

Reducer output is aggregated…

```
where       7
in          6
the         5
world       4
is          3
carmen      2
sandiego    1
```

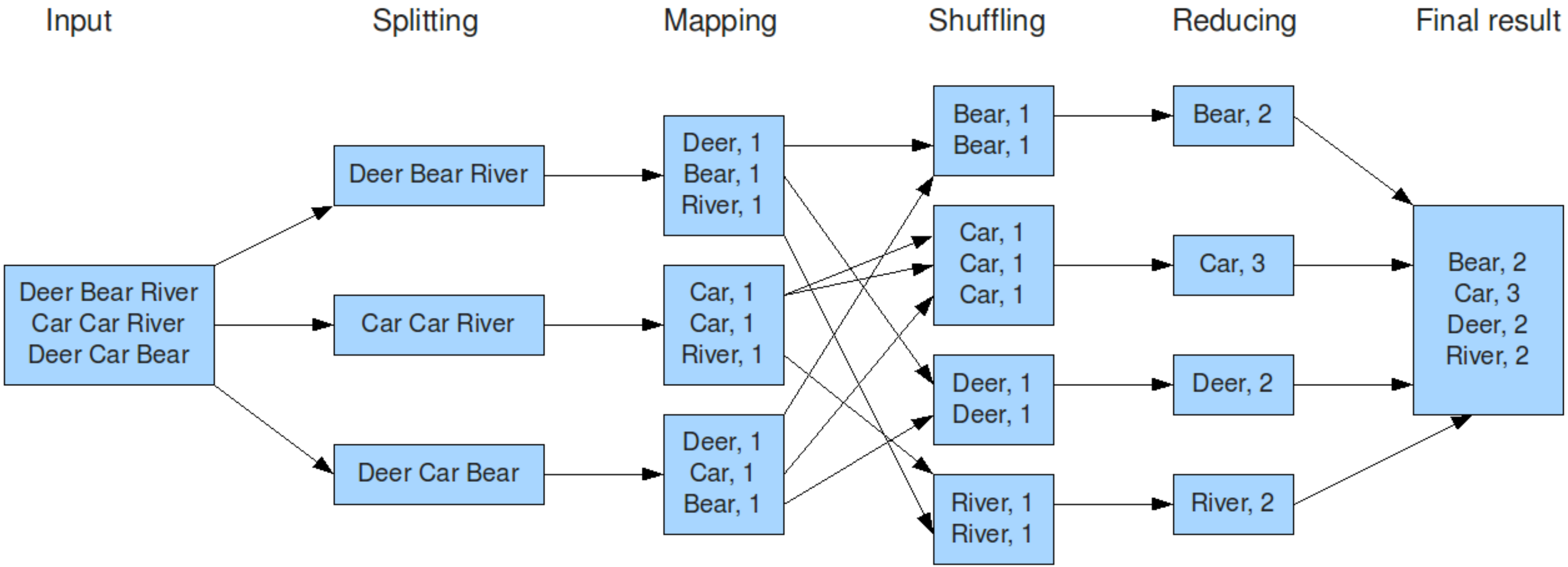
MAP-REDUCE EXAMPLE: REDUCER OUTPUT

Reducer output is aggregated & sorted by key.

```
carmen      2
is          3
in          6
the         5
sandiego    1
where       7
world       4
```

MAP-REDUCE

The overall MapReduce word count process



Input

Deer Bear River
Car Car River
Deer Car Bear

Splitting

Deer Bear River

Car Car River

Deer Car Bear

Mapping

Deer, 1
Bear, 1
River, 1

Car, 1
Car, 1
River, 1

Deer, 1
Car, 1
Bear, 1

Shuffling

Bear, 1
Bear, 1

Car, 1
Car, 1
Car, 1

Deer, 1
Deer, 1

River, 1
River, 1

Reducing

Bear, 2

Car, 3

Deer, 2

River, 2

Final result

Bear, 2
Car, 3
Deer, 2
River, 2