


Is MapReduce the end of the “Big Data” story? Does it meet all our needs?

# Big Data




Hat tip: Bob Korbus



It’s a buzz word, but generally associated with the problem of data sets too big to manage with traditional SQL databases. A parallel development has been the NoSQL movement that is good at handling semistructured data, scaling, etc.

# Big Data

Data so big that traditional solutions are too slow, too small, or too expensive to use.




Hat tip: Bob Korbus



It's a buzz word, but generally associated with the problem of data sets too big to manage with traditional SQL databases. A parallel development has been the NoSQL movement that is good at handling semistructured data, scaling, etc.


Wednesday, November 14, 12

Three trends influence my thinking...

# 3 Trends



Wednesday, November 14, 12

Three trends influence my thinking...

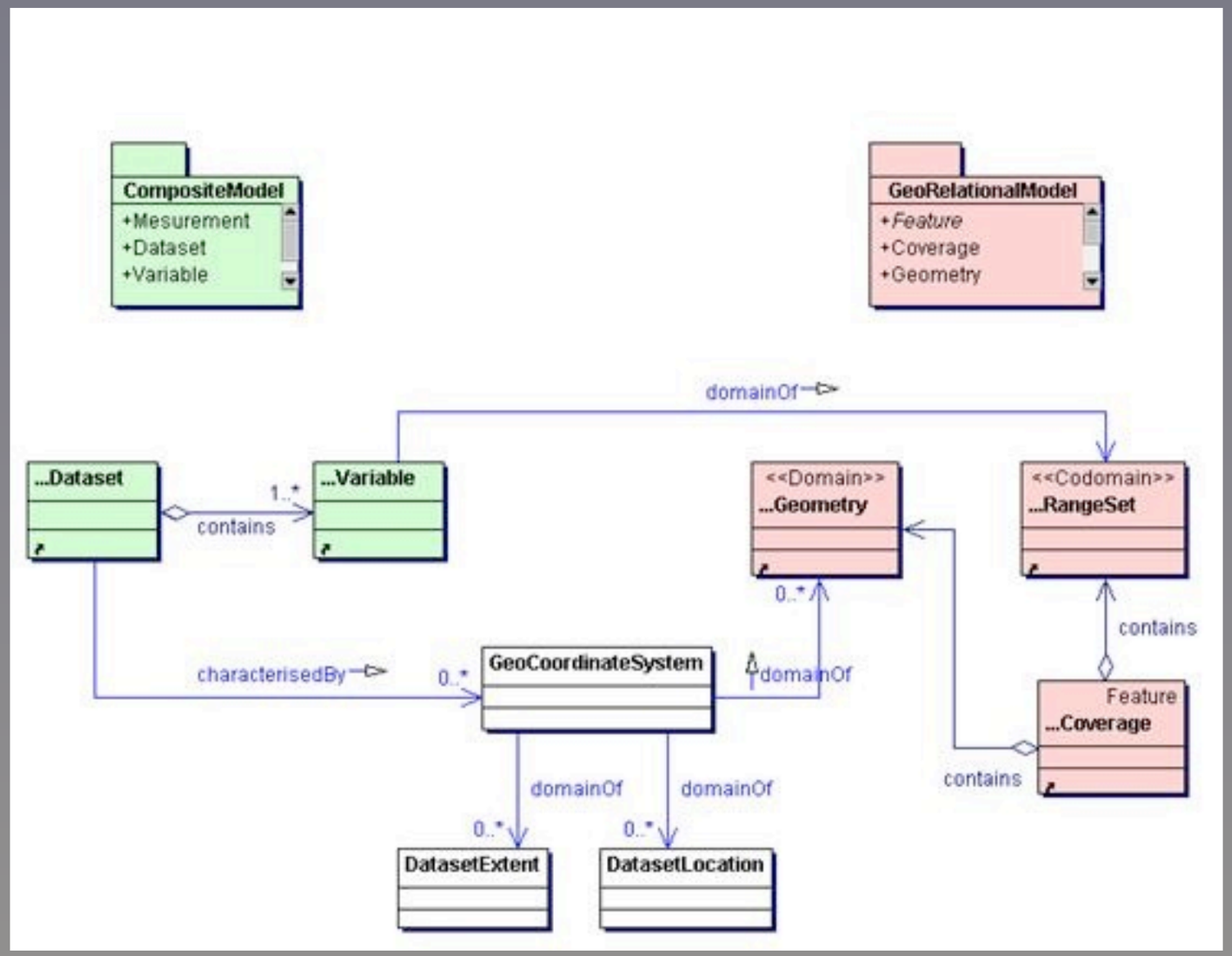
# Data Size ⬆



Wednesday, November 14, 12

Data volumes are obviously growing… rapidly.
Facebook now has over 600PB (Petabytes) of data!

# Formal Schemas ⬇






Wednesday, November 14, 12

There is less emphasis on “formal” schemas and domain models, i.e., both relational models of data and OO models, because data schemas and sources change rapidly. So, using relatively-agnostic software (e.g., collections of things where the software is agnostic about the contents) tends to be faster to develop and run. Put another way, we find it more useful to build somewhat agnostic applications and drive their behavior through data.

# Data-Driven Programs




Wednesday, November 14, 12

This is the 2nd generation “Stanley”, the most successful self-driving car ever built (by a Google-Stanford) team. Machine learning is growing in importance. Here, generic algorithms and data structures are trained to represent the “world” using data, rather than encoding a model of the world in the software itself. It’s another example of generic algorithms that produce the desired behavior by being application agnostic and data driven, rather than hard-coding a model of the world. (In practice, however, a balance is struck between completely agnostic apps and some engineering towards for the specific problem.



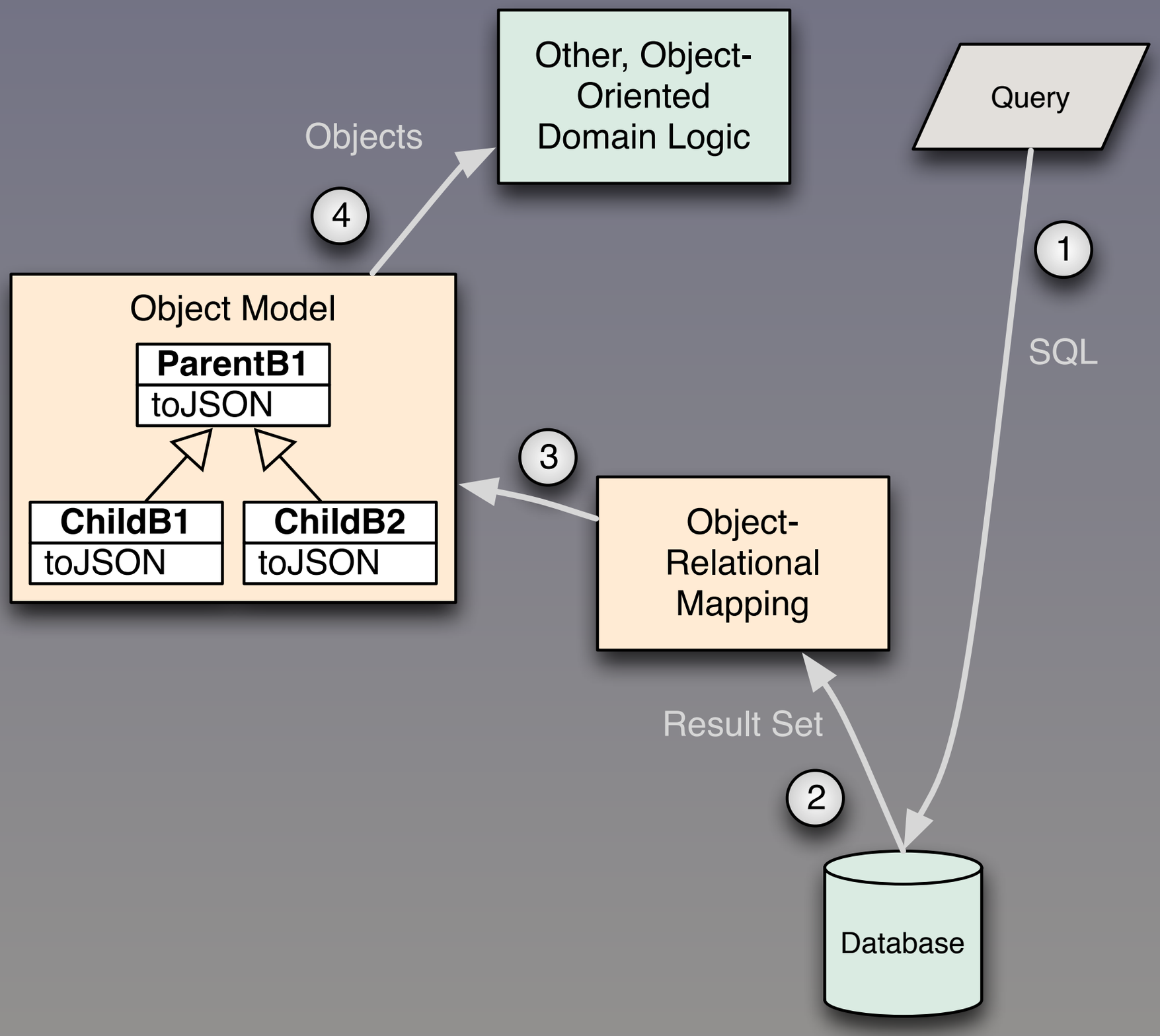

What should software architectures look like for these kinds of systems?

# Big Data Architectures





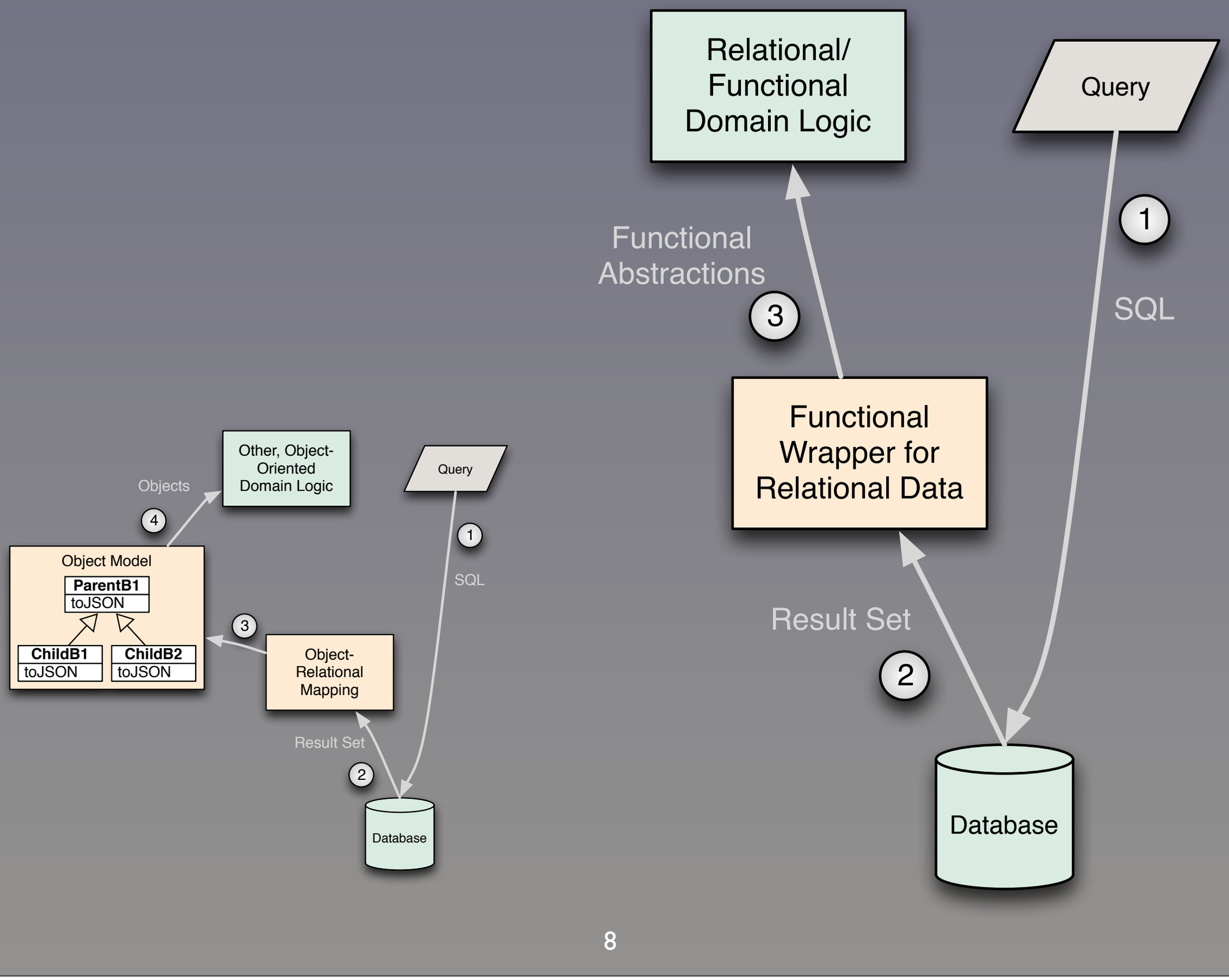
What should software architectures look like for these kinds of systems?






Wednesday, November 14, 12

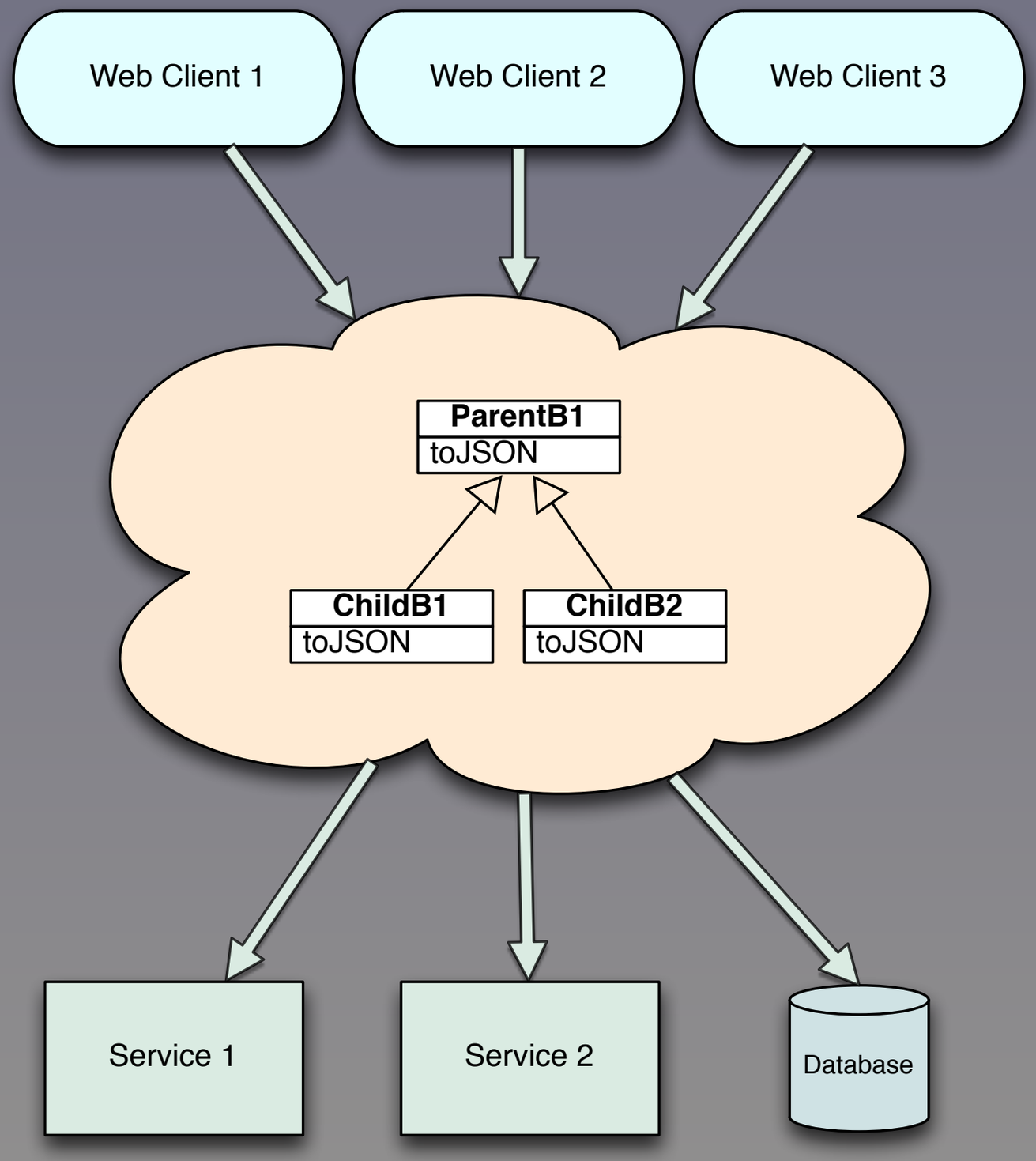
Traditionally, we've kept a rich, in-memory domain model requiring an ORM to convert persistent data into the model. This is resource overhead and complexity we can't afford in big data systems. Rather, we should treat the result set as it is, a particular kind of collection, do the minimal transformation required to exploit our collections libraries and classes representing some domain concepts (e.g., Address, StockOption, etc.), then write functional code to implement business logic (or drive emergent behavior with machine learning algorithms…)






Wednesday, November 14, 12

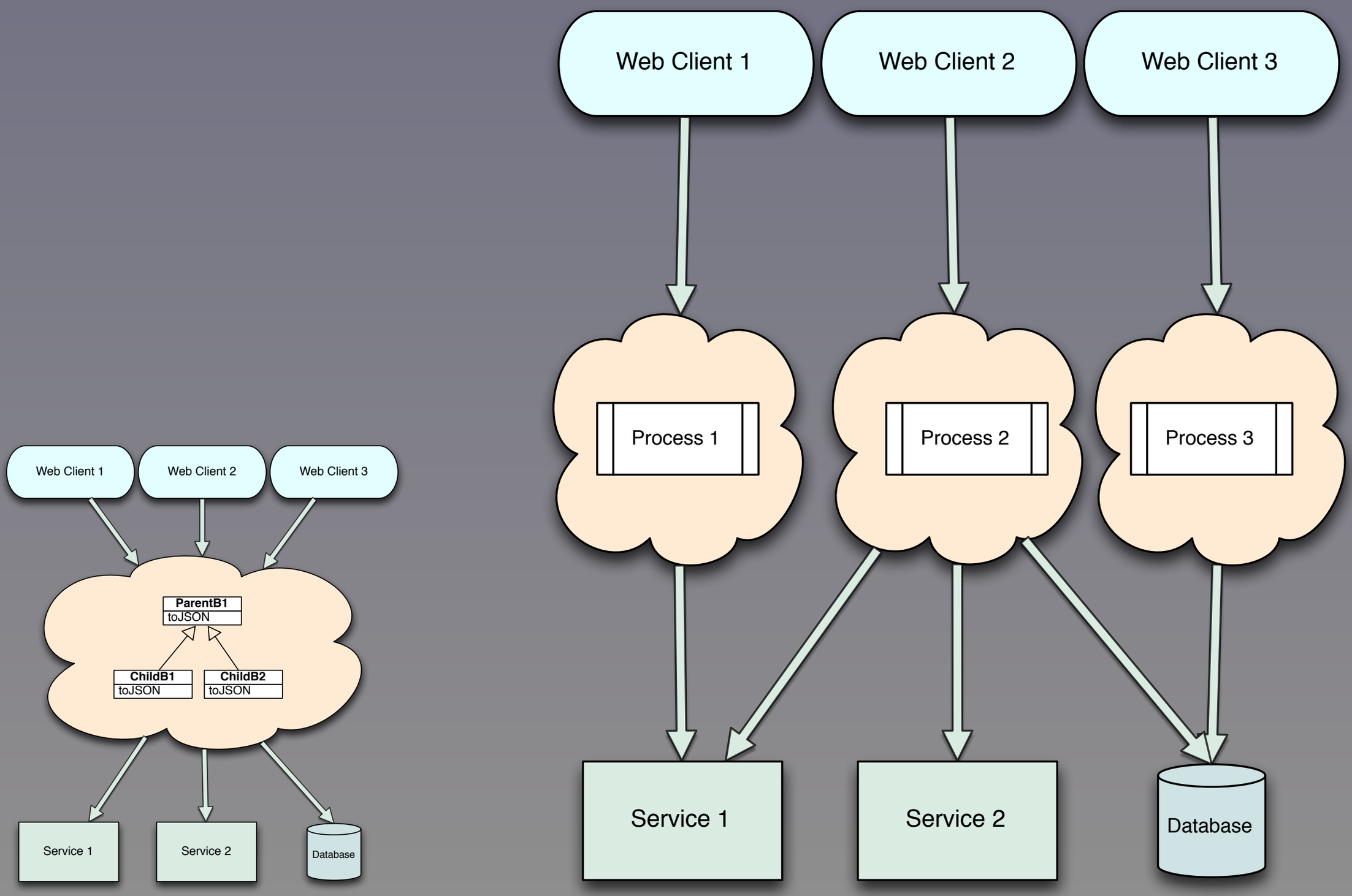
Traditionally, we've kept a rich, in-memory domain model requiring an ORM to convert persistent data into the model. This is resource overhead and complexity we can't afford in big data systems. Rather, we should treat the result set as it is, a particular kind of collection, do the minimal transformation required to exploit our collections libraries and classes representing some domain concepts (e.g., Address, StockOption, etc.), then write functional code to implement business logic (or drive emergent behavior with machine learning algorithms…)






Wednesday, November 14, 12

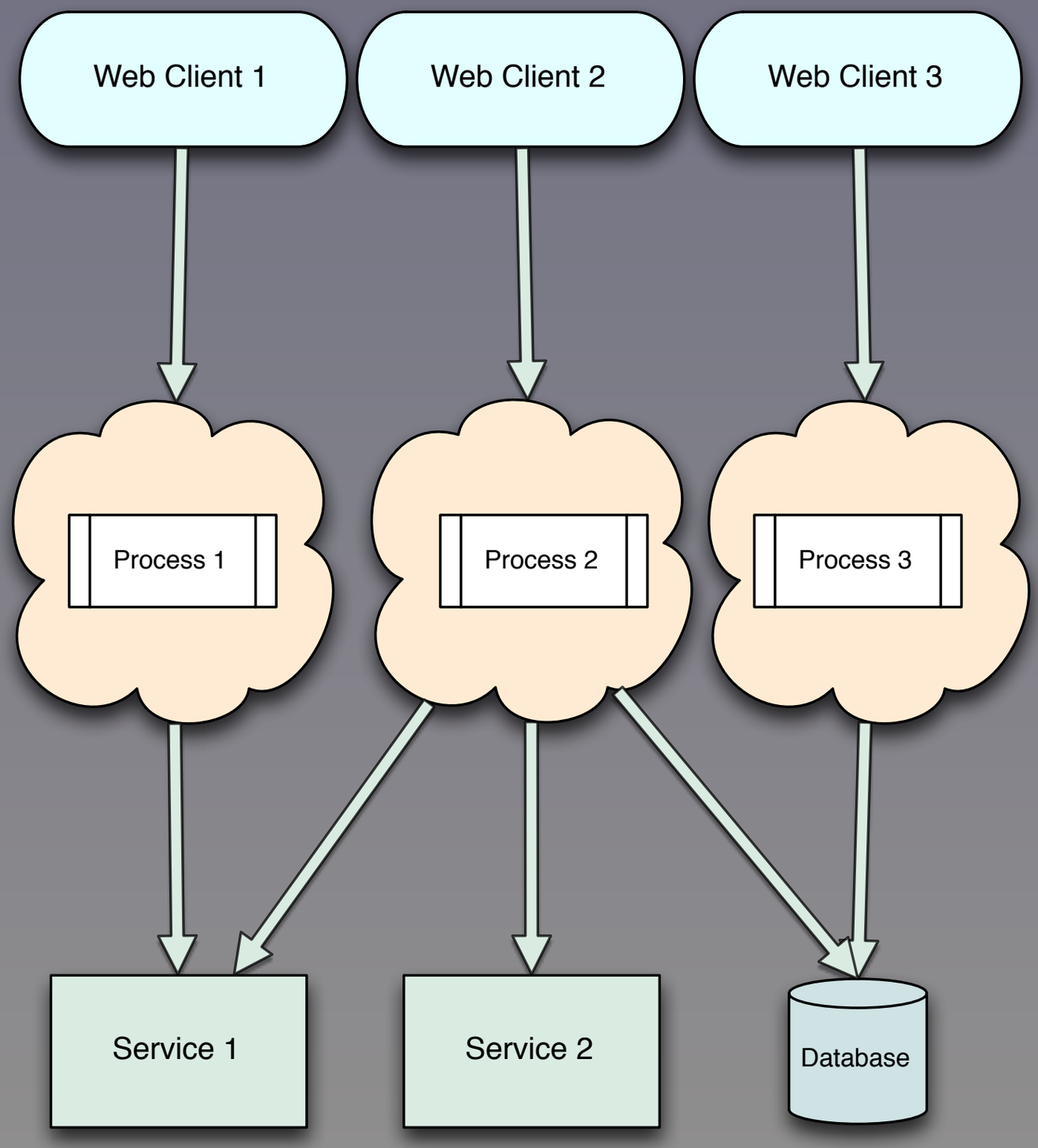
In a broader view, object models tend to push us towards centralized, complex systems that don’t decompose well and stifle reuse and optimal deployment scenarios. FP code makes it easier to write smaller, focused services that we compose and deploy as appropriate.






Wednesday, November 14, 12

In a broader view, object models tend to push us towards centralized, complex systems that don't decompose well and stifle reuse and optimal deployment scenarios. FP code makes it easier to write smaller, focused services that we compose and deploy as appropriate.

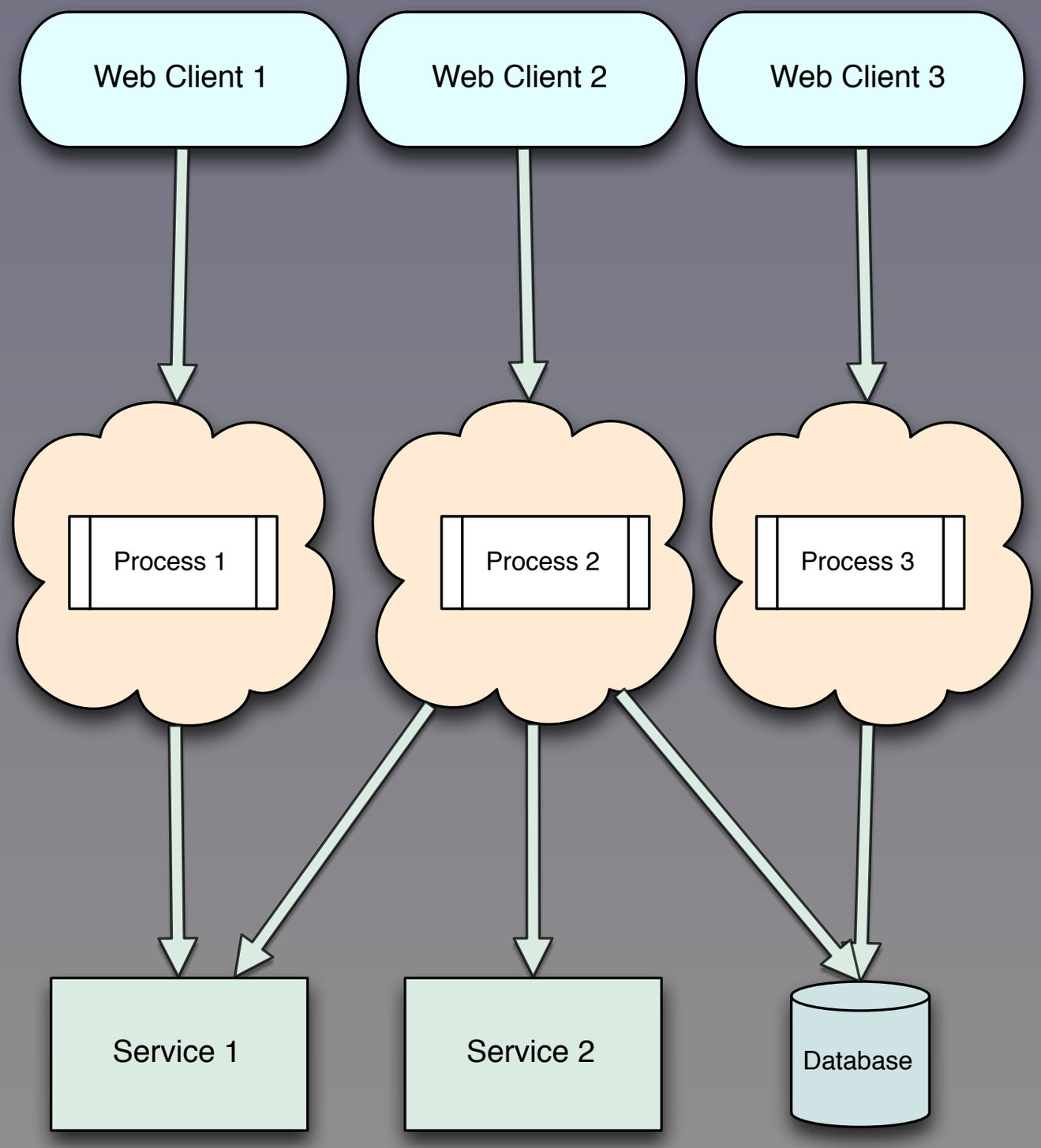





Wednesday, November 14, 12

Smaller, focused services scale better, especially horitzontally. They also don’t encapsulate more business logic than is required, and this (informal) architecture is also suitable for scaling ML and related algorithms.

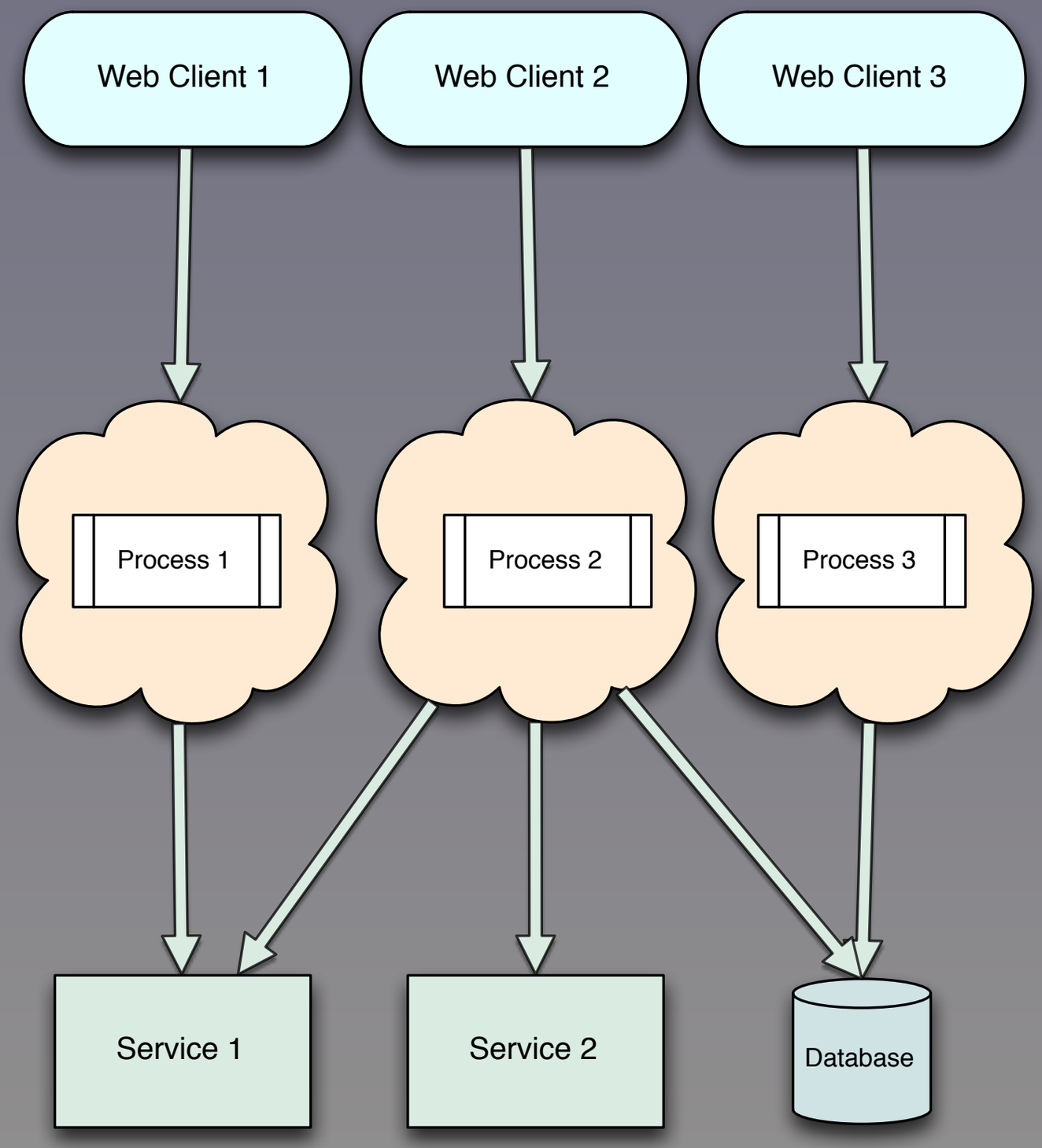
- Data Size ⬆
- Formal Schema ⬇
- Data-Driven Programs ⬆






Wednesday, November 14, 12

Smaller, focused services scale better, especially horitzontally. They also don't encapsulate more business logic than is required, and this (informal) architecture is also suitable for scaling ML and related algorithms.

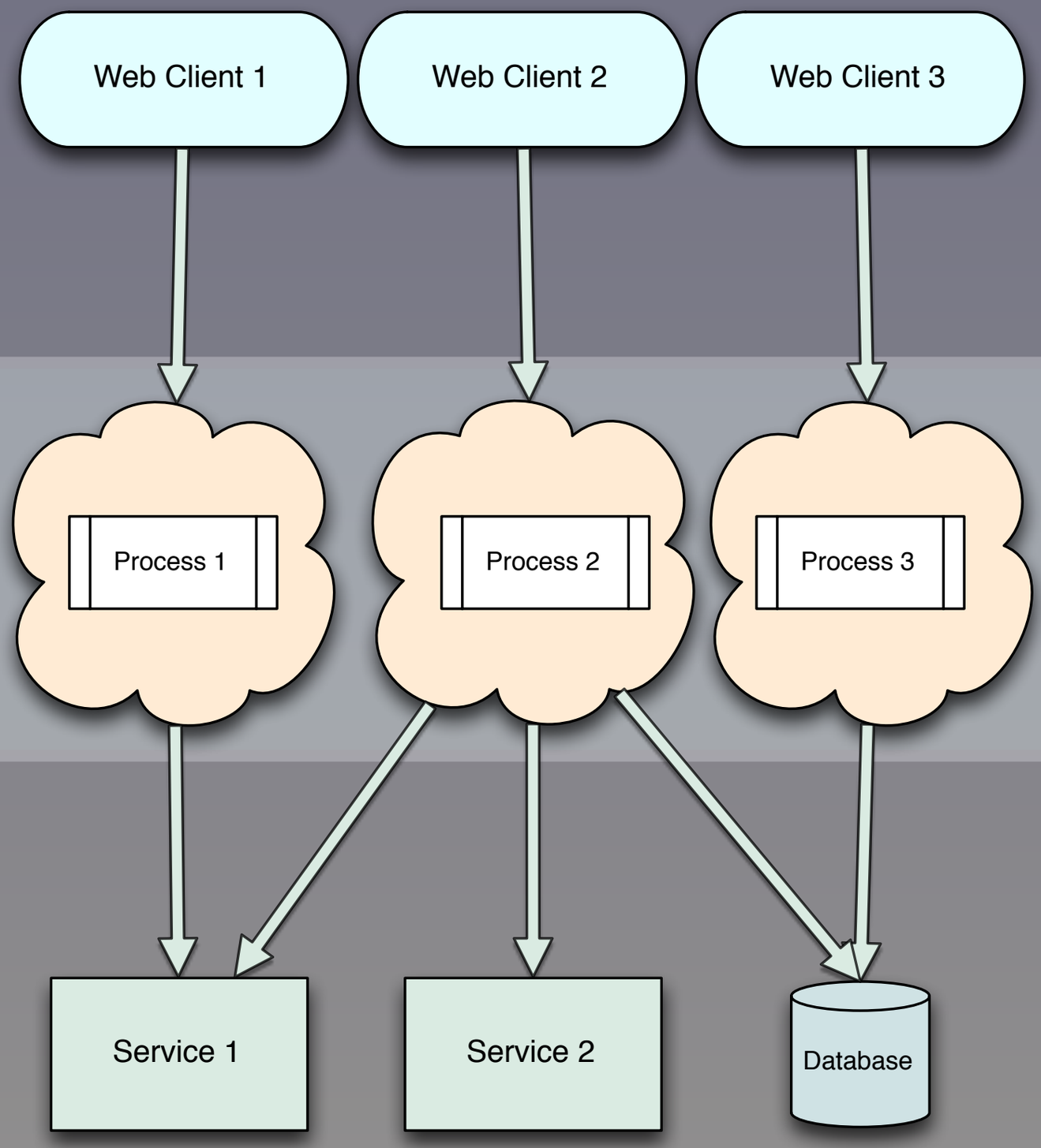





Wednesday, November 14, 12

And MapReduce + a distributed file system, like the Hadoop DFS, fit this model.

- MapReduce

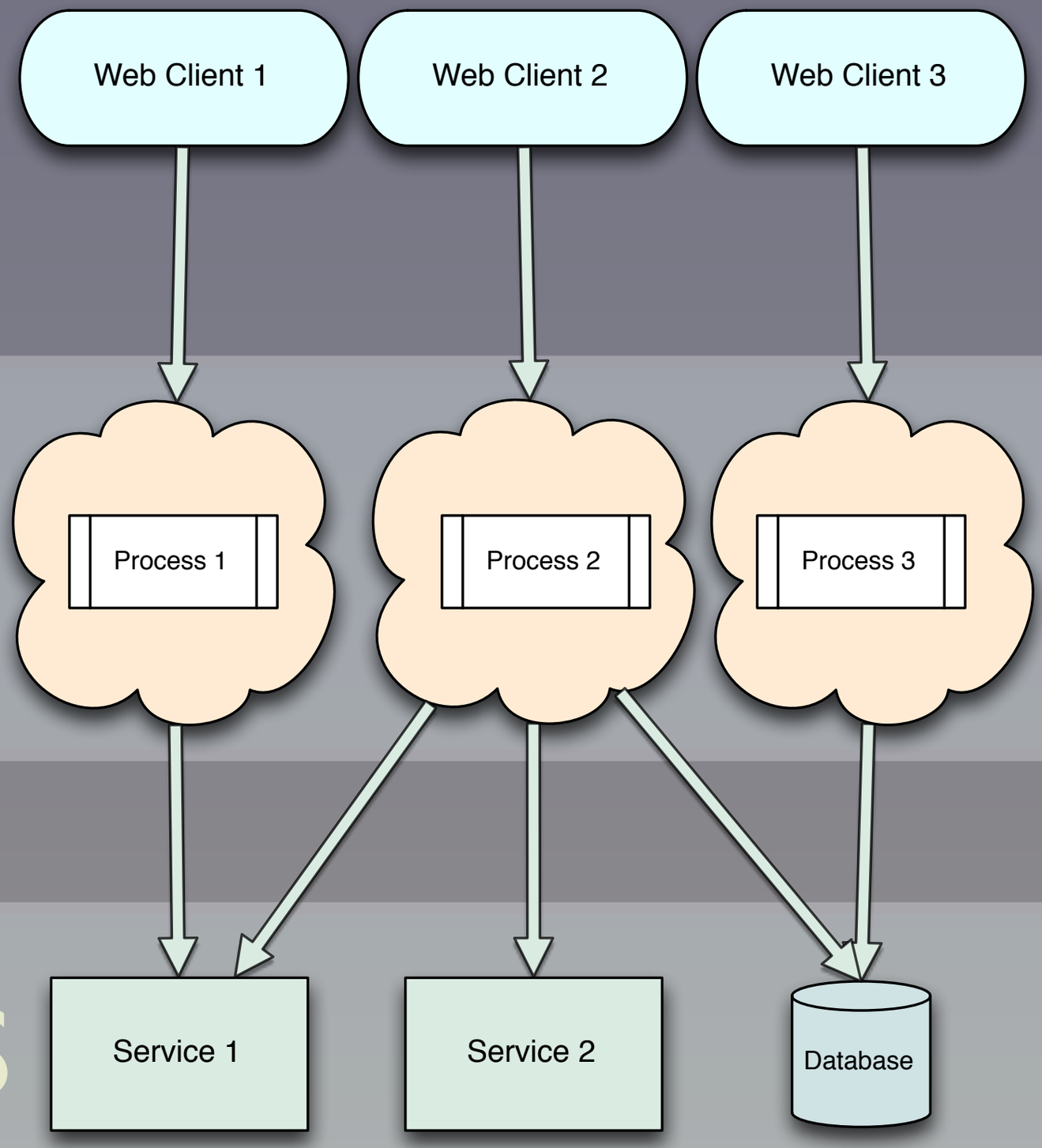





Wednesday, November 14, 12

And MapReduce + a distributed file system, like the Hadoop DFS, fit this model.

- MapReduce




- Distributed FS

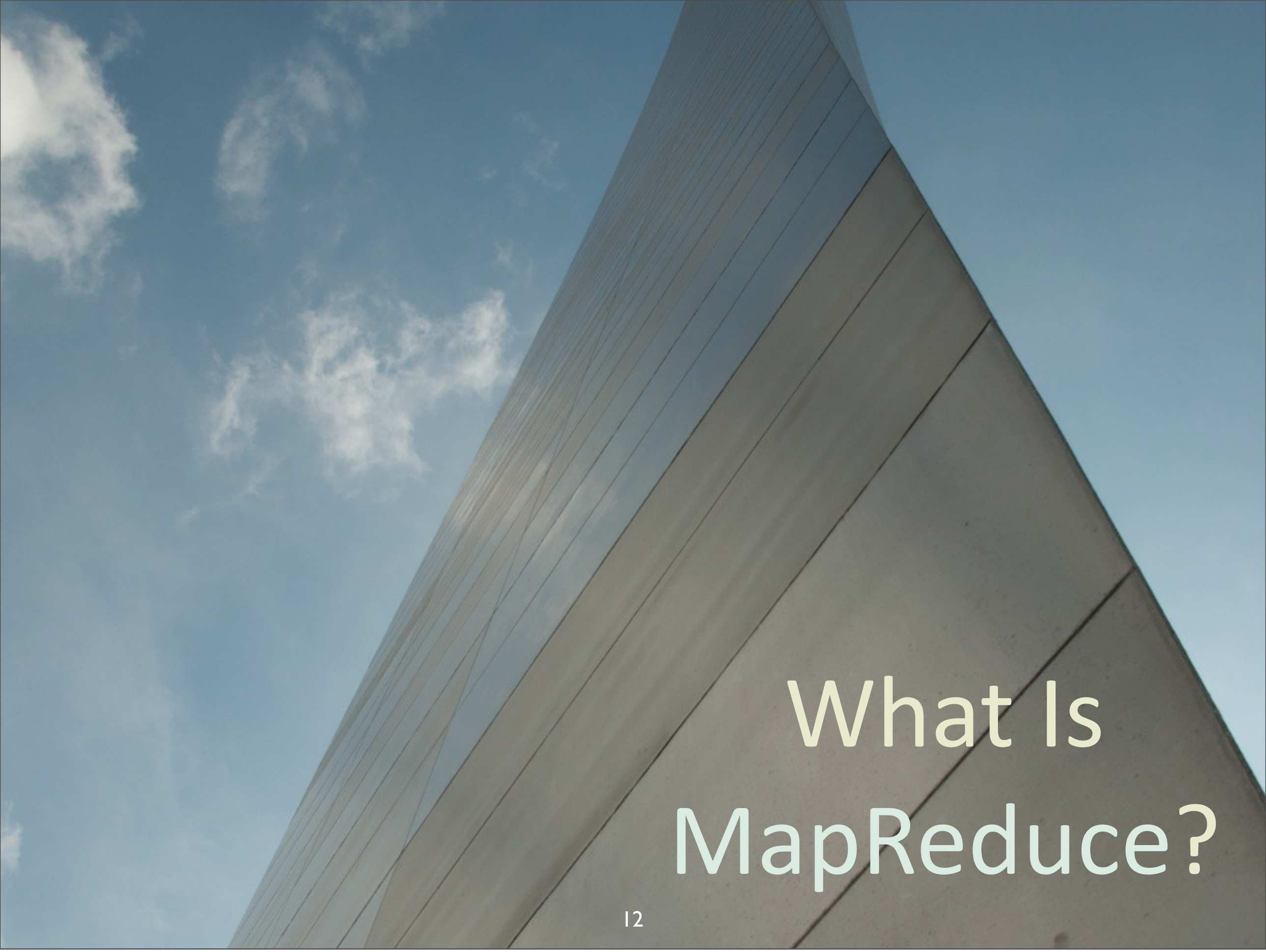


Wednesday, November 14, 12

And MapReduce + a distributed file system, like the Hadoop DFS, fit this model.




Let’s walk through the “Hello World” of MapReduce; the Word Count algorithm...






Wednesday, November 14, 12

Let's walk through the "Hello World" of MapReduce; the Word Count algorithm...

# MapReduce in Hadoop

Let's look at a *MapReduce* algorithm: *WordCount*.



Let's walk through this algorithm at a conceptual level...

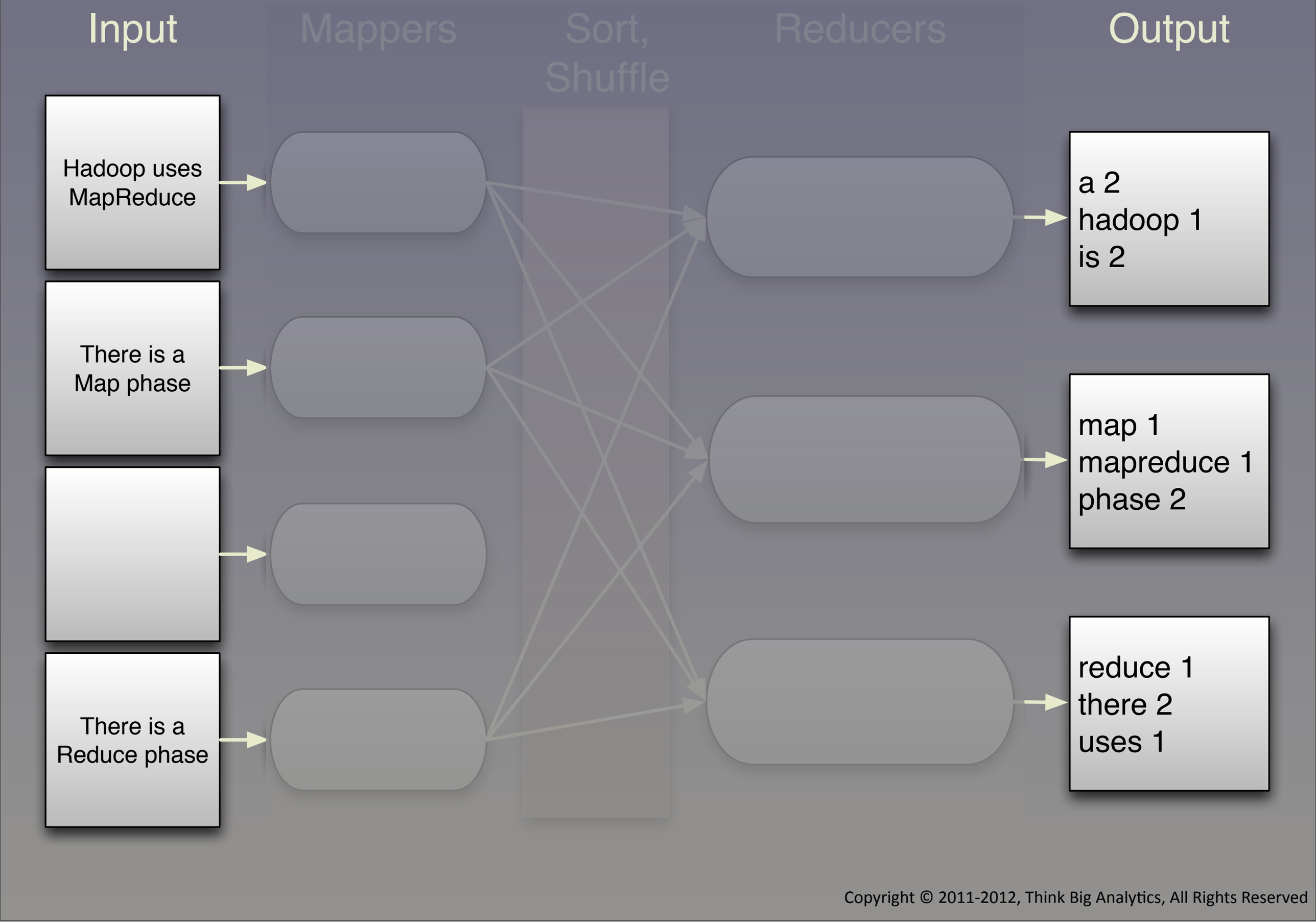
# MapReduce in Hadoop

Let's look at a *MapReduce* algorithm: *WordCount*.

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



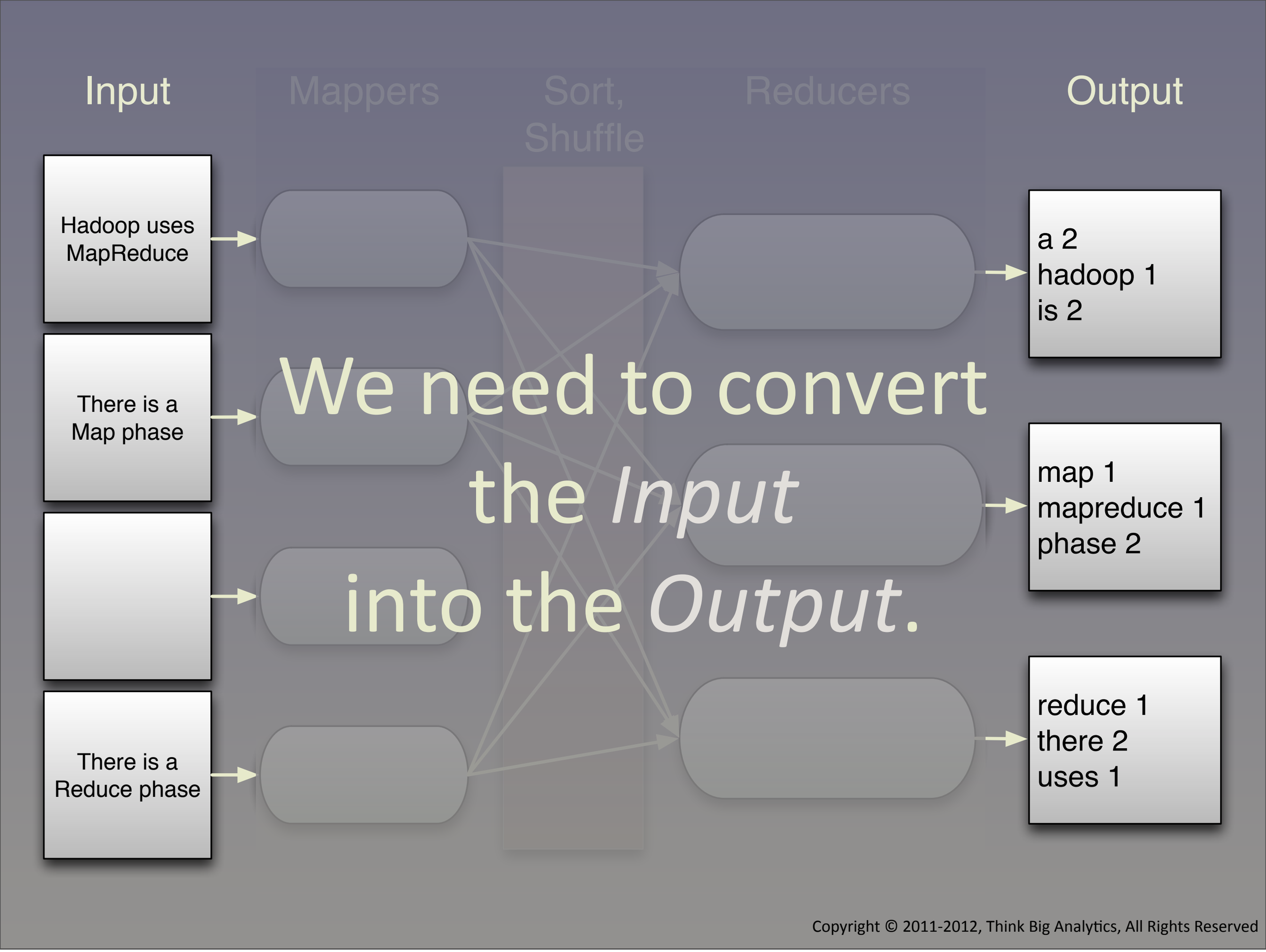
Let's walk through this algorithm at a conceptual level...







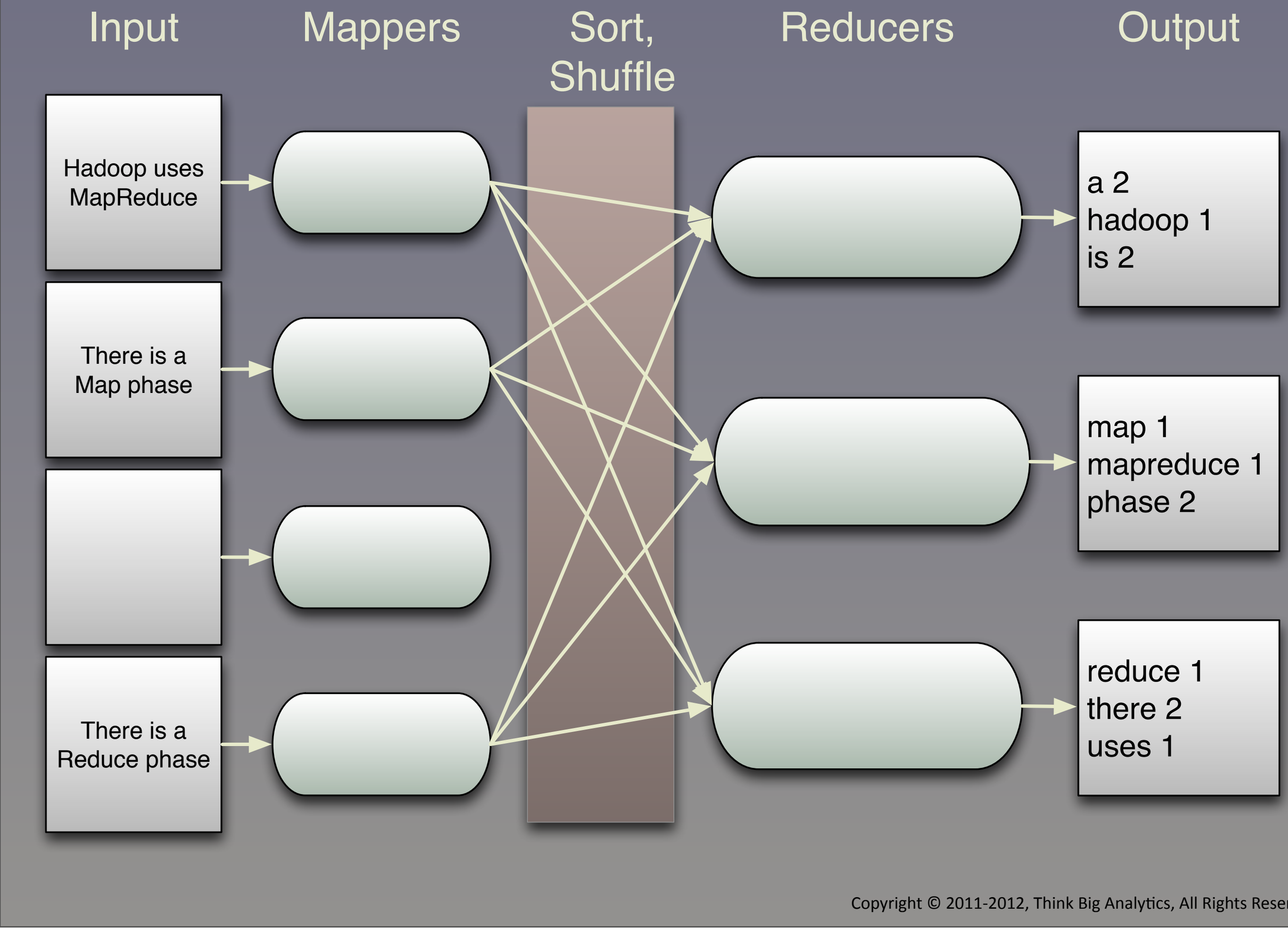
Four input documents, one left empty, the others with small phrases (for simplicity…). The word count output is on the right (we'll see why there are three output "documents"). We need to get from the input on the left-hand side to the output on the right-hand side.







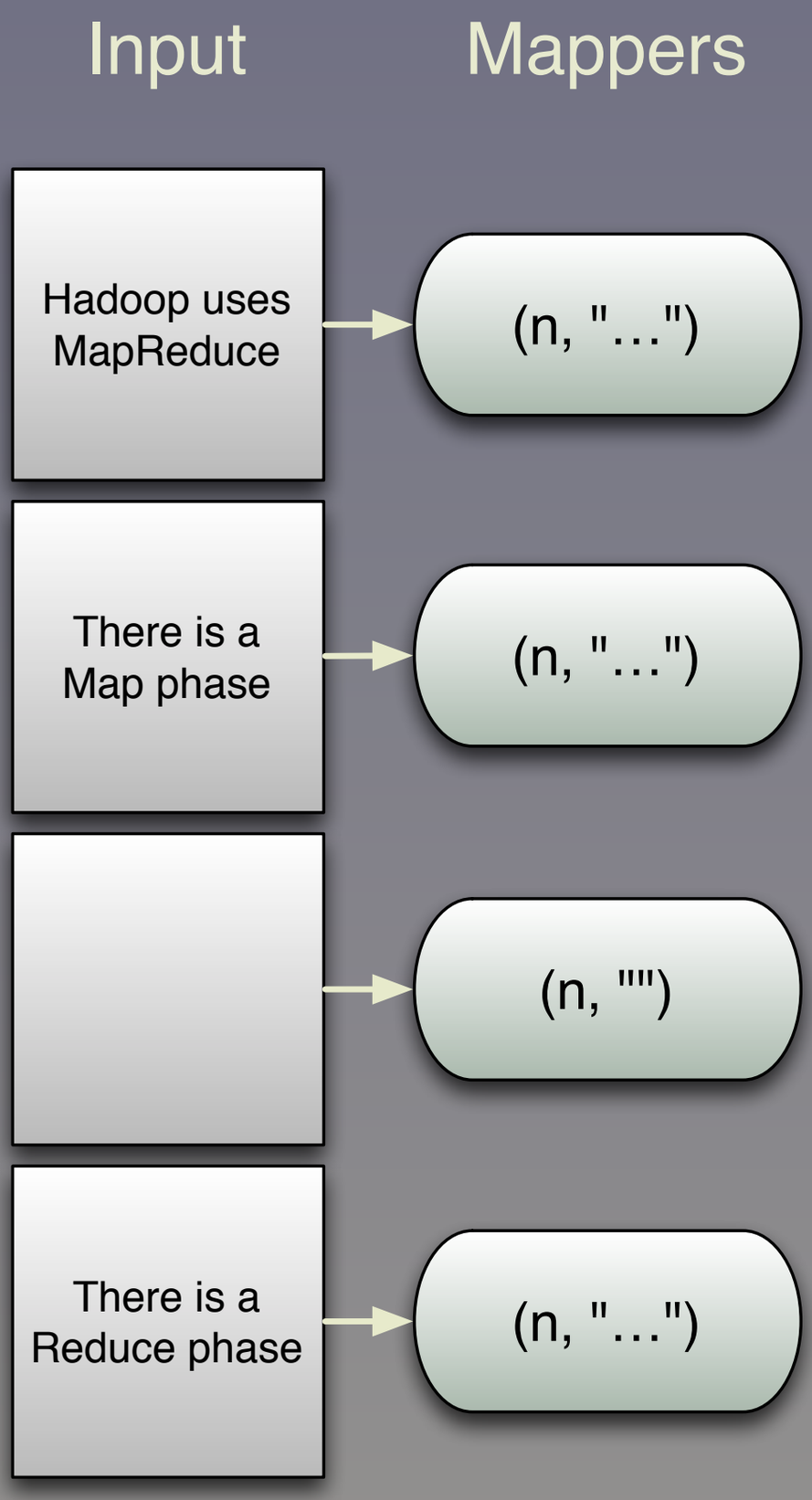
Four input documents, one left empty, the others with small phrases (for simplicity…). The word count output is on the right (we'll see why there are three output "documents"). We need to get from the input on the left-hand side to the output on the right-hand side.







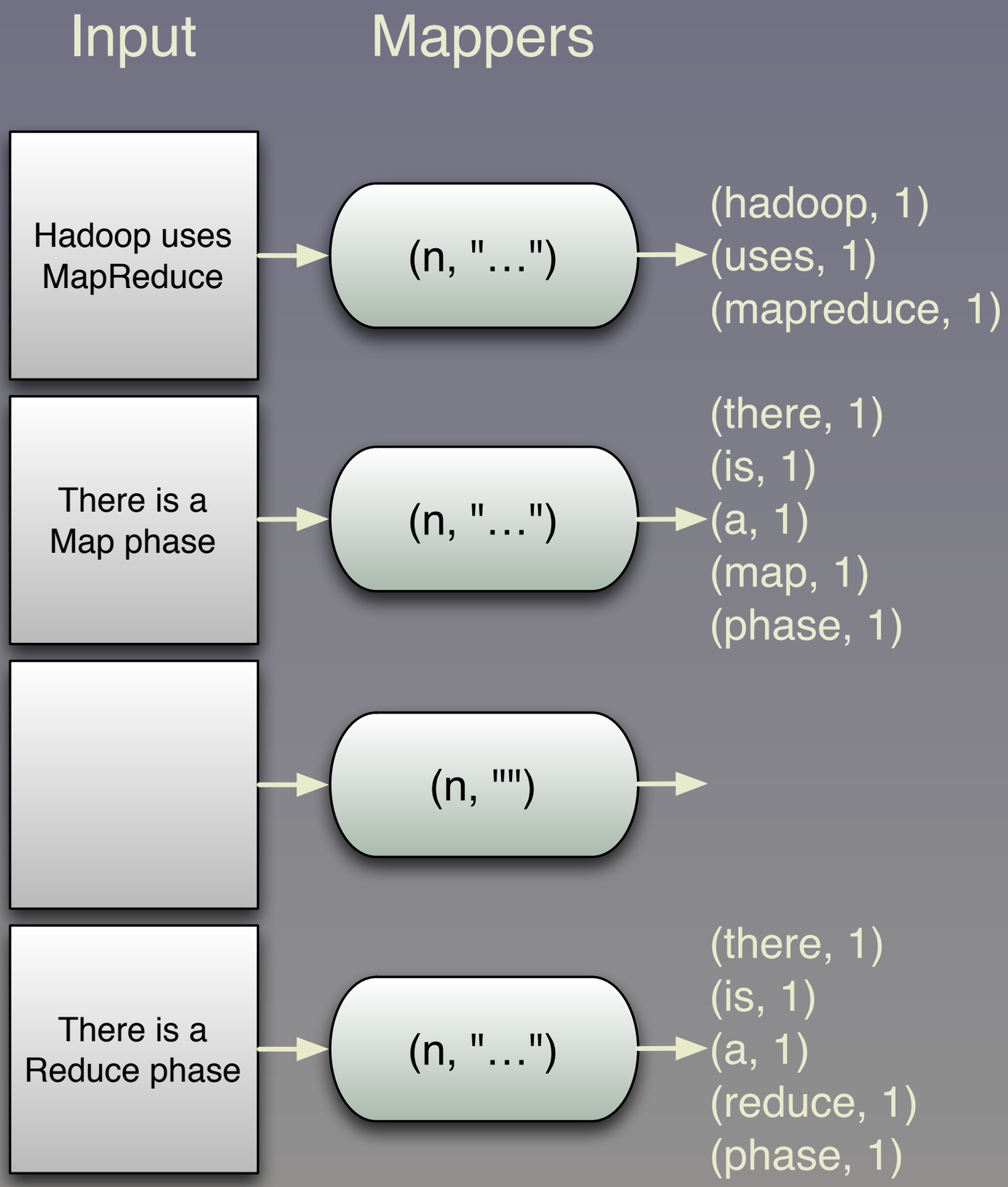
Here is a schematic view of the steps in Hadoop MapReduce. Each Input file is read by a single Mapper process (default: can be many-to-many, as we'll see later).
The Mappers emit key-value pairs that will be sorted, then partitioned and "shuffled" to the reducers, where each Reducer will get all instances 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).







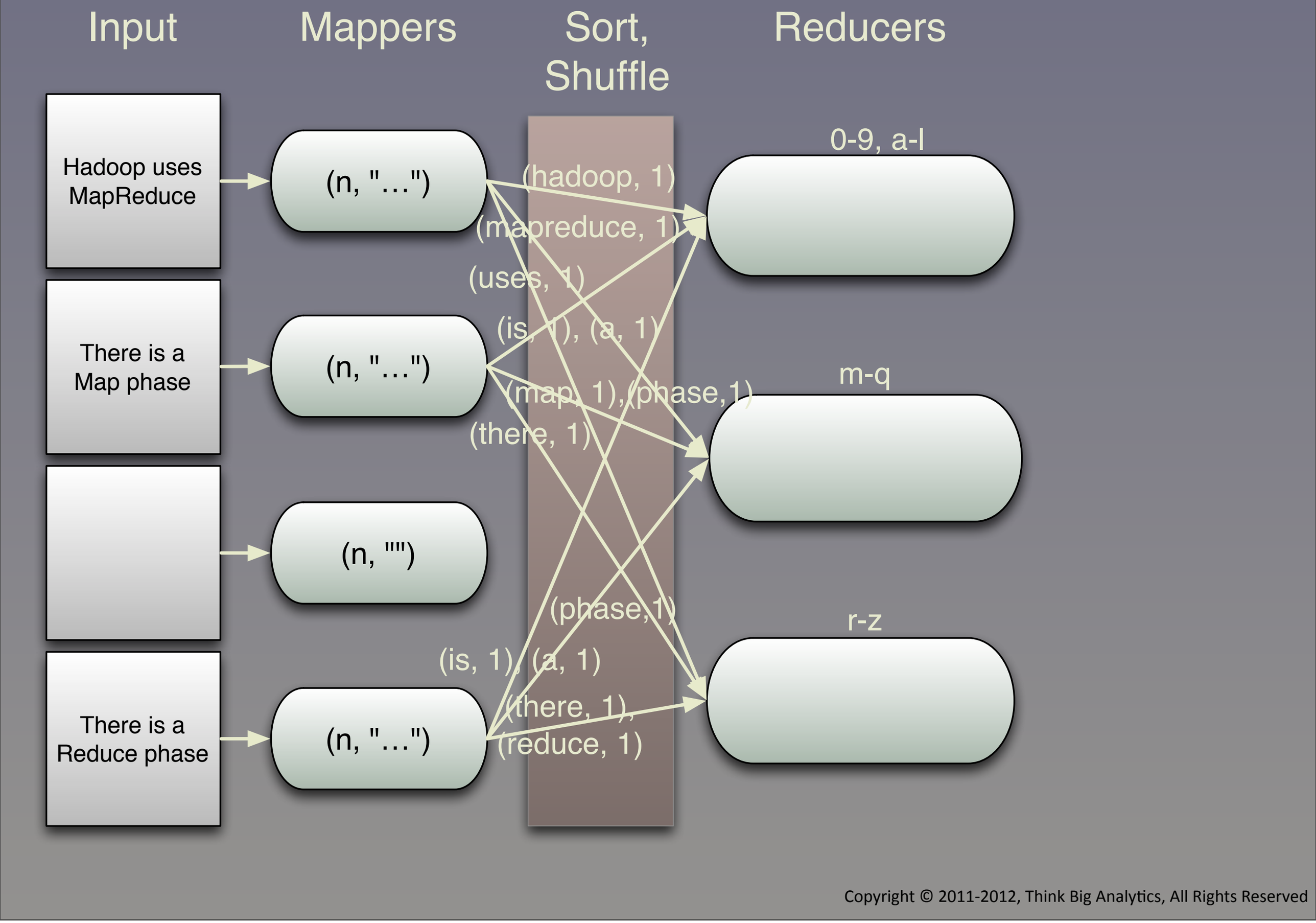

Each document gets a mapper. All data is organized into key-value pairs; each line will be a value and the offset position into the file will be the key, which we don't care about. I'm showing the document contents in the boxes and 1 mapper task (JVM process) per document. Large documents might get split to several mappers.
Each mapper will tokenize each line, one at a time, convert all words to lower case and count them...








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. However, this is IO inefficient!
Note that the mapper for the empty doc. emits no pairs, as you would expect.

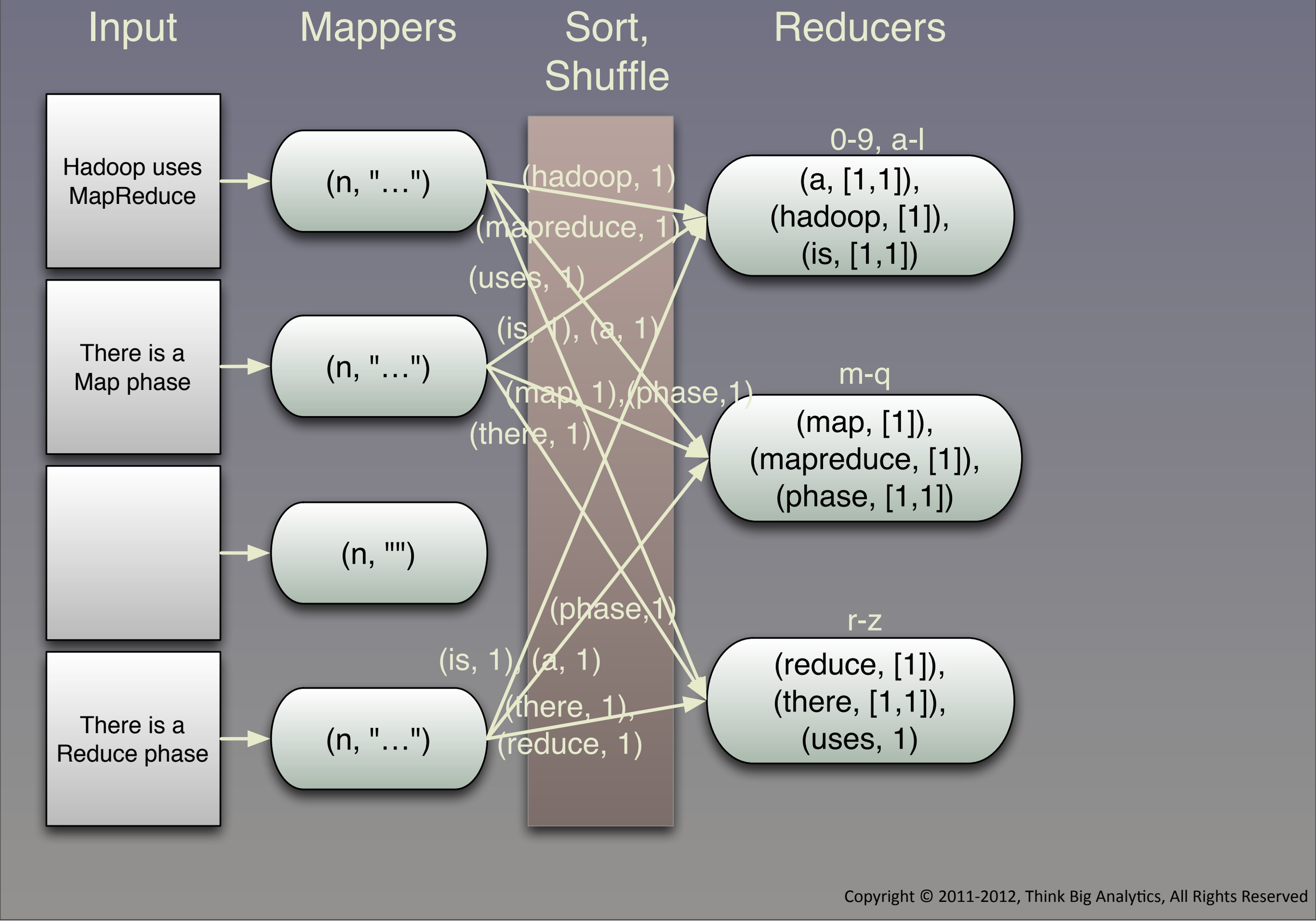




Wednesday, November 14, 12

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) and then the pairs are routed to the correct reducer, on the current machine or other machines.

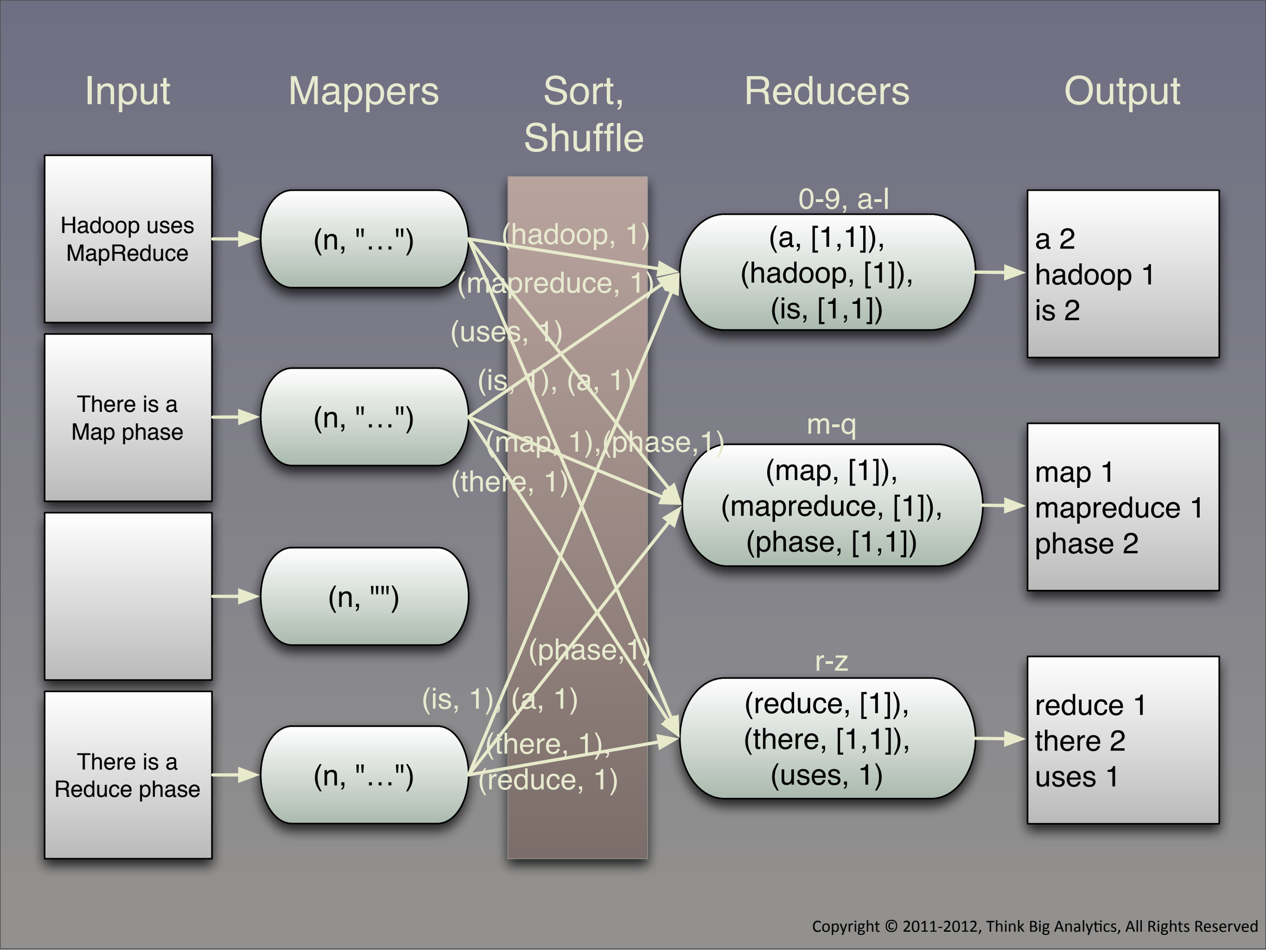
Note how we partitioned the reducers, by first letter of the keys. (By default, MR just hashes the keys and distributes them modulo # of reducers.)







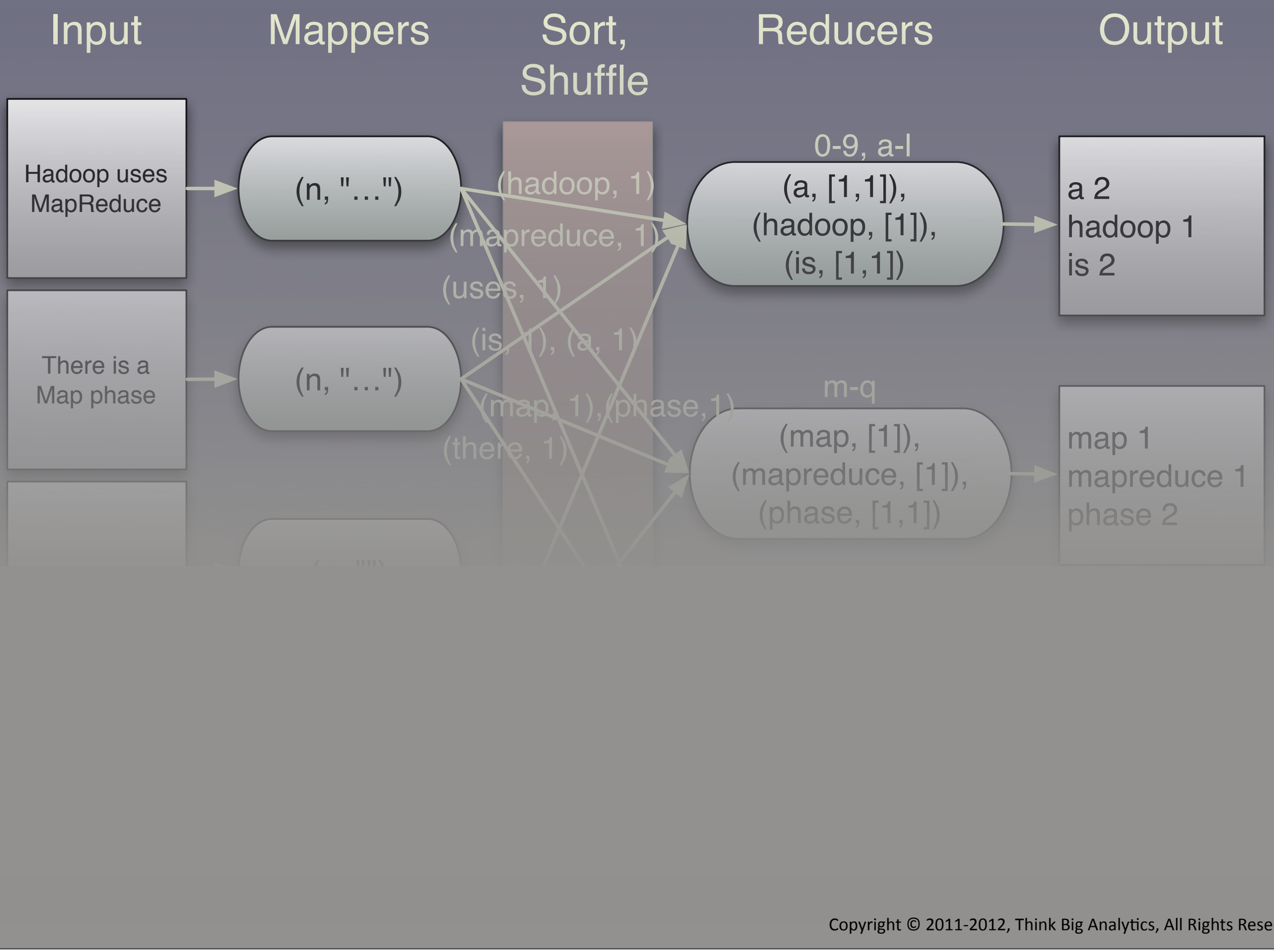

Our reducers are passed each key and a collection of all the values for that key. (The MR framework creates this collection for us.)







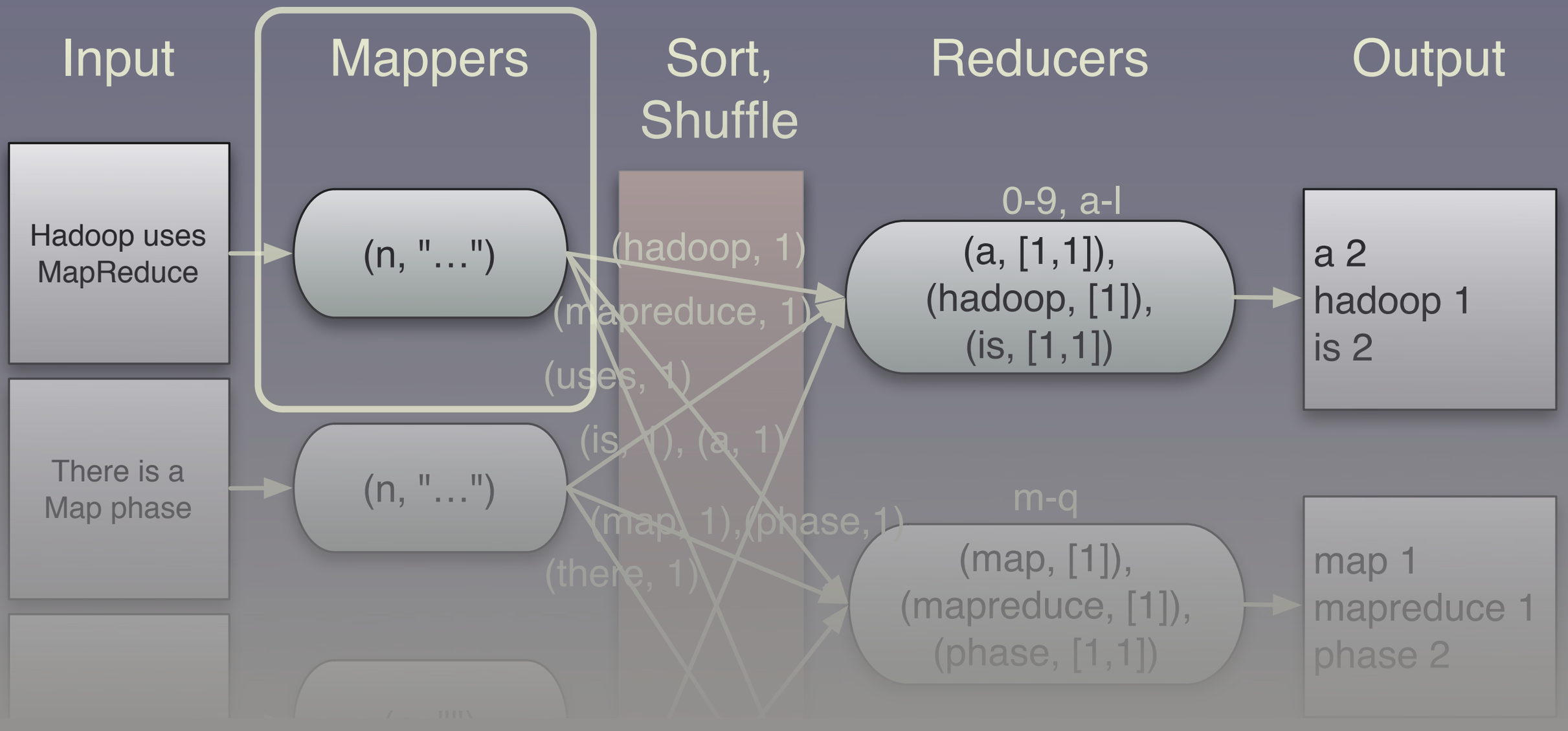
The final view of the WordCount process flow. The reducer just sums the counts and writes the output. The output files contain one line for each key (the word) and value (the count), assuming we're using text output. The choice of delimiter between key and value is up to you, but tab is common. (We'll discuss options as we go.)








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



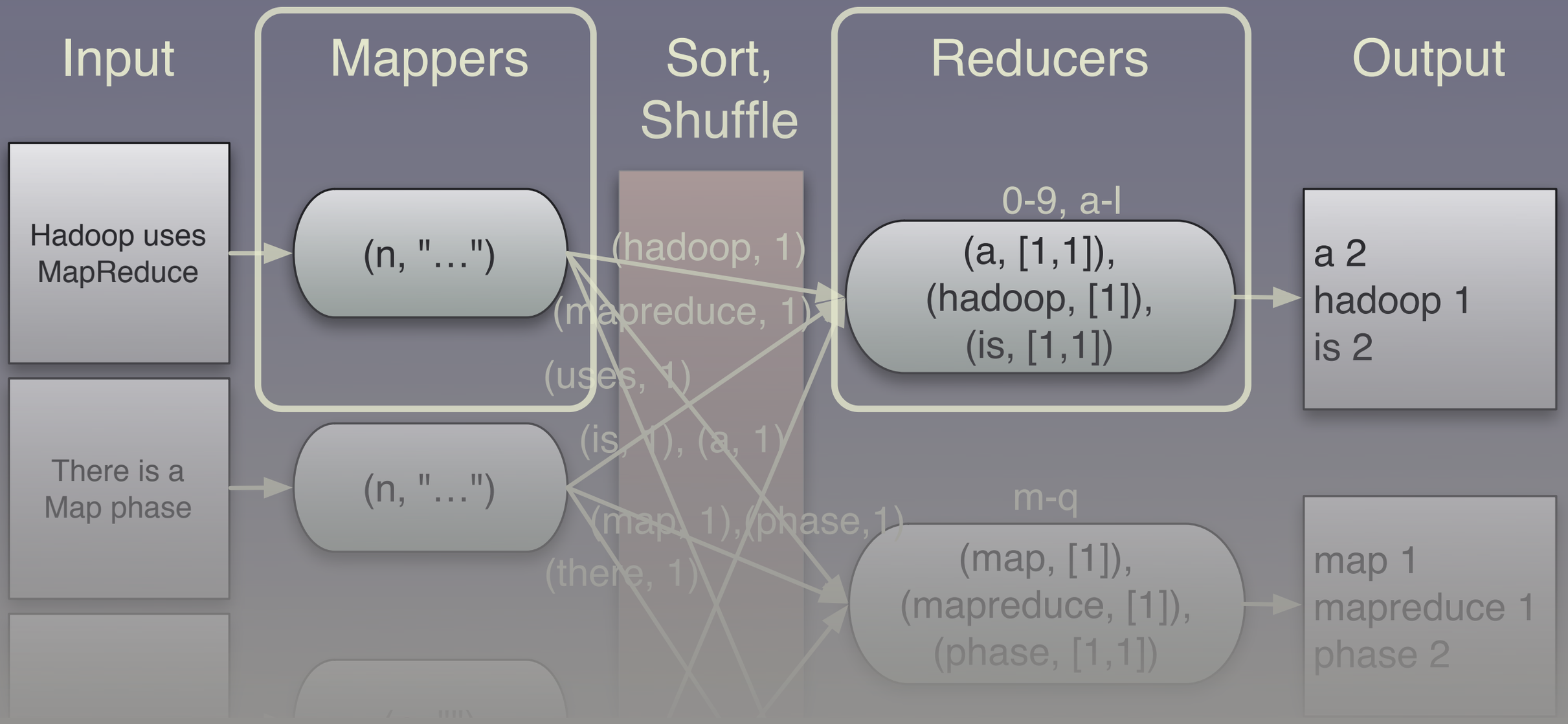

*Map*:

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



Wednesday, November 14, 12

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




*Map*:

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

*Reduce*:

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



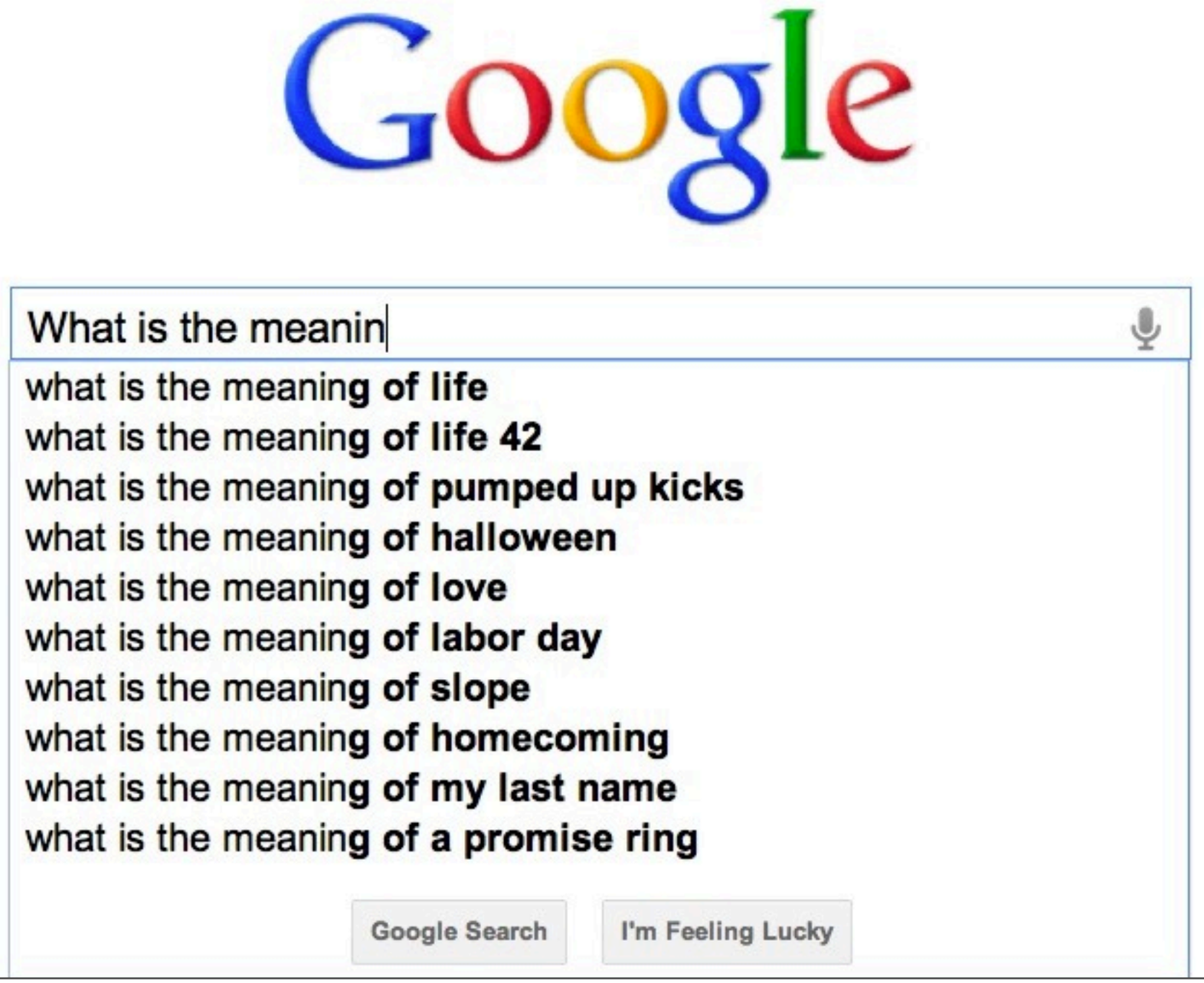


To recap, a “map” transforms one input to one output, but this is generalized in MapReduce to be one to 0-N. The output key-value pairs are distributed to reducers. The “reduce” collects together multiple inputs with the same key into

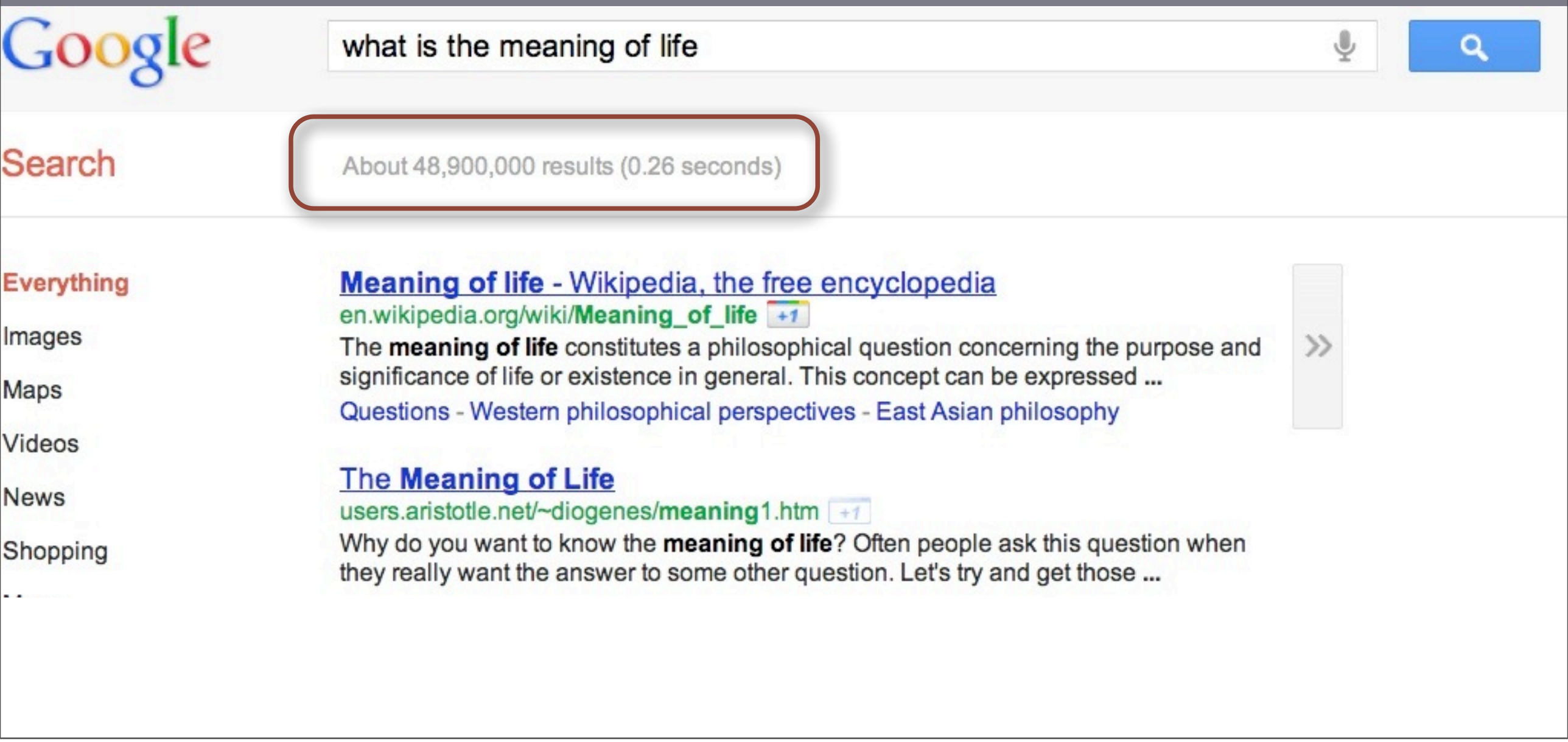



Let's review where MapReduce came from and its best-known, open-source incarnation, Hadoop.

# How would you *index* the *web*?



# How would you *index* the *web*?



You ask a *phrase* and the *search engine* finds the *best match* in *billions* of web pages.

Actually, *Google* computes the *index* that *maps* terms to pages in *advance*.

Actually, *Google* computes the *index* that *maps* terms to pages in *advance*.

Google's famous *Page Rank* algorithm.

# *Google File System*
# for Storage

## 2003

**The Google File System**

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

**ABSTRACT**

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological envi-

**1. INTRODUCTION**

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier



A distributed file system provides horizontal scalability and resiliency when file blocks are duplicated around the cluster.

# *MapReduce* for Computation

2004

**MapReduce: Simplified Data Processing on Large Clusters**

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

*Google, Inc.*

## Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to par-



A distributed file system provides horizontal scalability and resiliency when file blocks are duplicated around the cluster.

About this time, *Doug Cutting*, the creator of *Lucene* was working on *Nutch*...



Lucene is an open-source text search engine. Nutch is an open source web crawler.

He implemented clean-room versions of *MapReduce* and *GFS*...

By *2006* , they became part of a separate *Apache* project, called *Hadoop*.



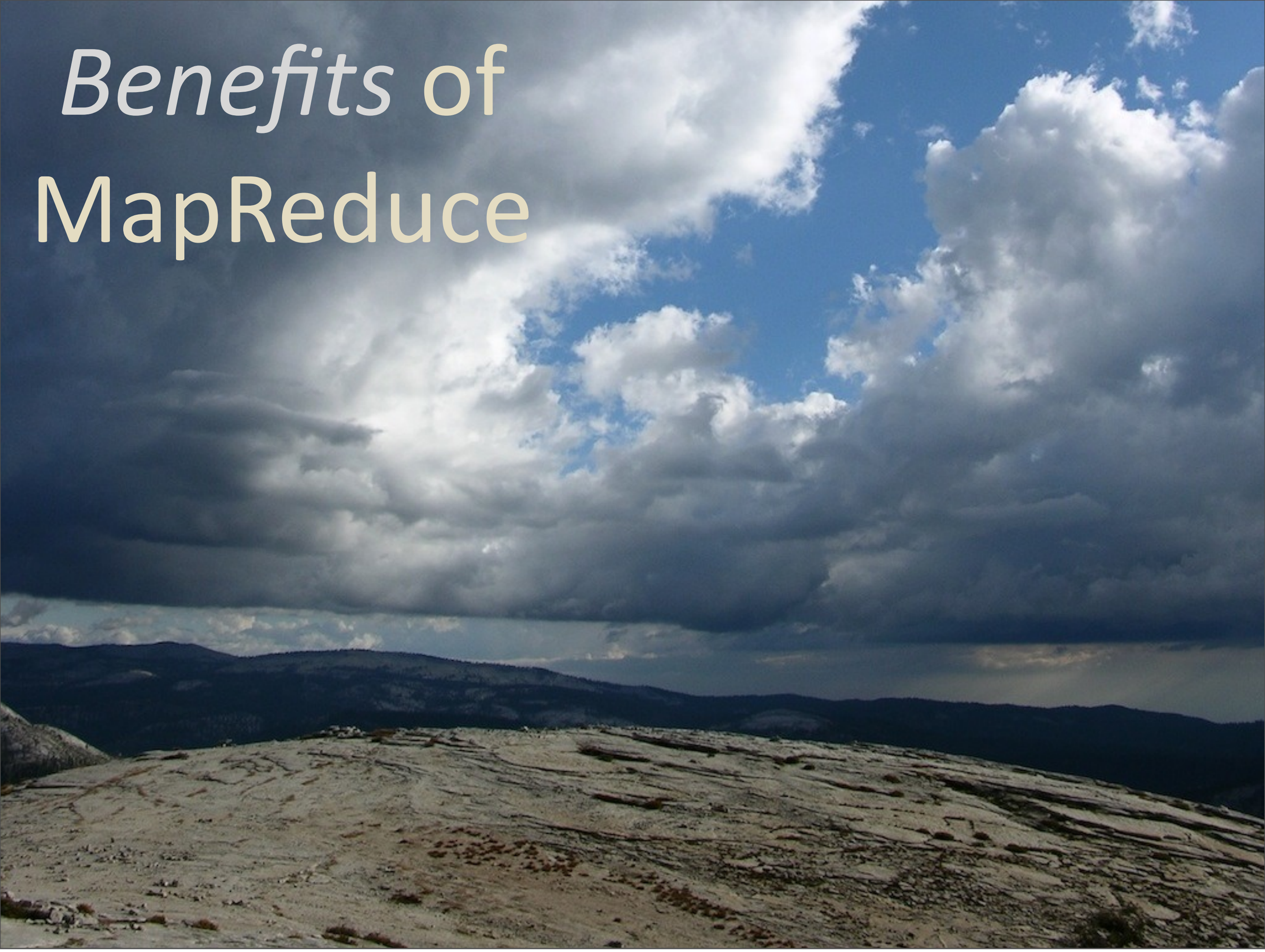
The name comes from a toy, stuffed elephant that Cutting's son owned at the time.

By *2006* , they became part of a separate *Apache* project, called *Hadoop*.




The name comes from a toy, stuffed elephant that Cutting's son owned at the time.

# *Benefits* of MapReduce



The best way to approach *Big Data* is to scale *Horizontally*.

# Hadoop Design Goals



Maximizing disk and network I/O is so important, because it's such a bottleneck, that it is a core design goal of Hadoop (both MR and HDFS). It affects the features and performance of everything in the stack, including high-level programming tools!

# Hadoop Design Goals

Maximize I/O!!



Maximizing disk and network I/O is so important, because it's such a bottleneck, that it is a core design goal of Hadoop (both MR and HDFS). It affects the features and performance of everything in the stack, including high-level programming tools!

# Hadoop Design Goals

Maximize I/O!!

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



Maximizing disk and network I/O is so important, because it's such a bottleneck, that it is a core design goal of Hadoop (both MR and HDFS). It affects the features and performance of everything in the stack, including high-level programming tools!

As a result, Hadoop is *great* for b*atch mode* data crunching.

# MapReduce and its Discontents



Is MapReduce the end of the story? Does it meet all our needs? Let's look at two problems...

# #1

# It's hard to *implement* many *Algorithms* in *MapReduce*.



Even word count is not "obvious". When you get to fancier stuff like joins, group-bys, etc., the mapping is not trivial at all.

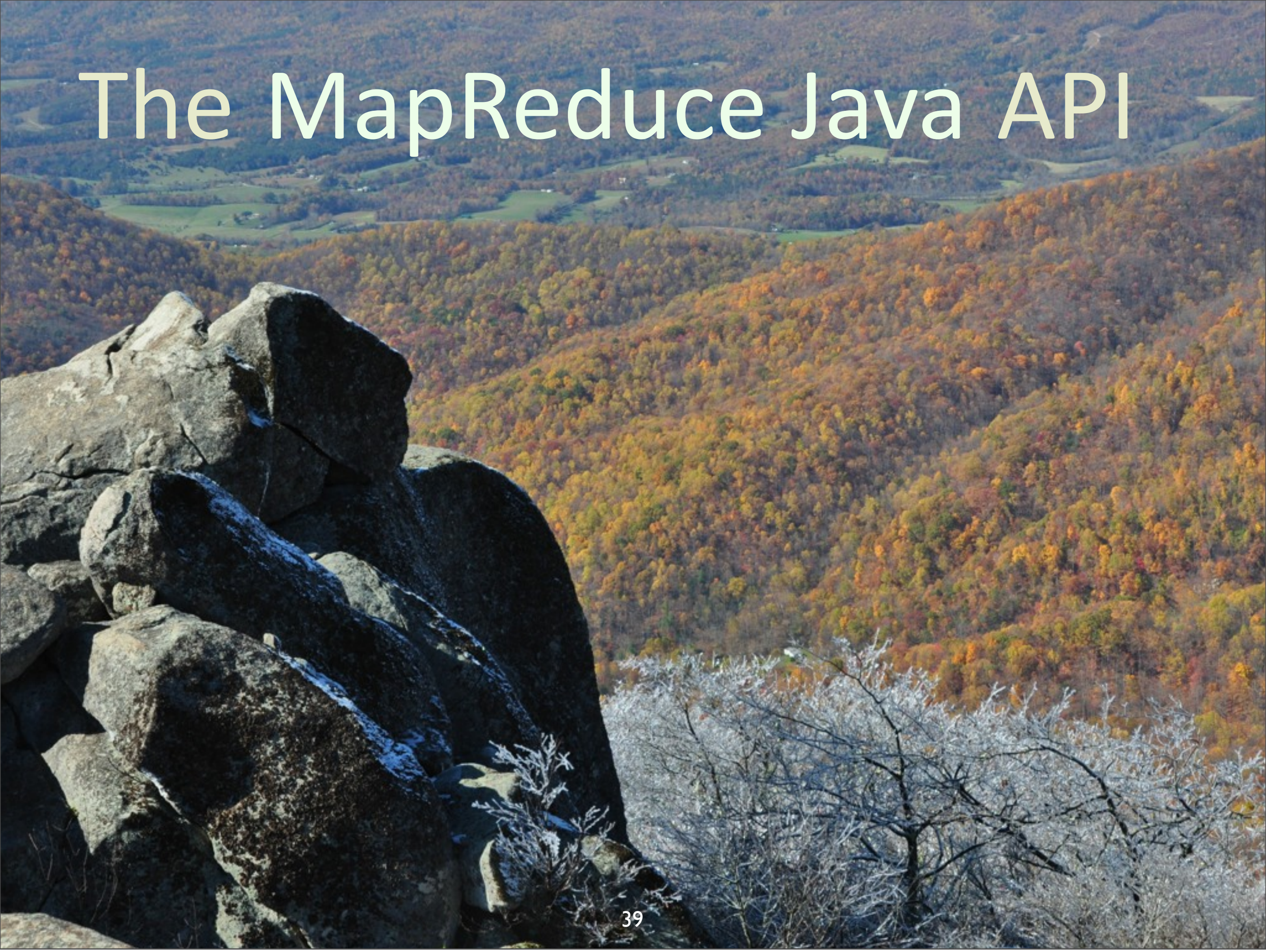
# #2

# What about *"real-time" event processing*?



Even Google now does not want to wait to compute a new index of the web. As soon as its web crawlers detect changes, Google would like to update its index as soon as possible.

# The MapReduce Java API

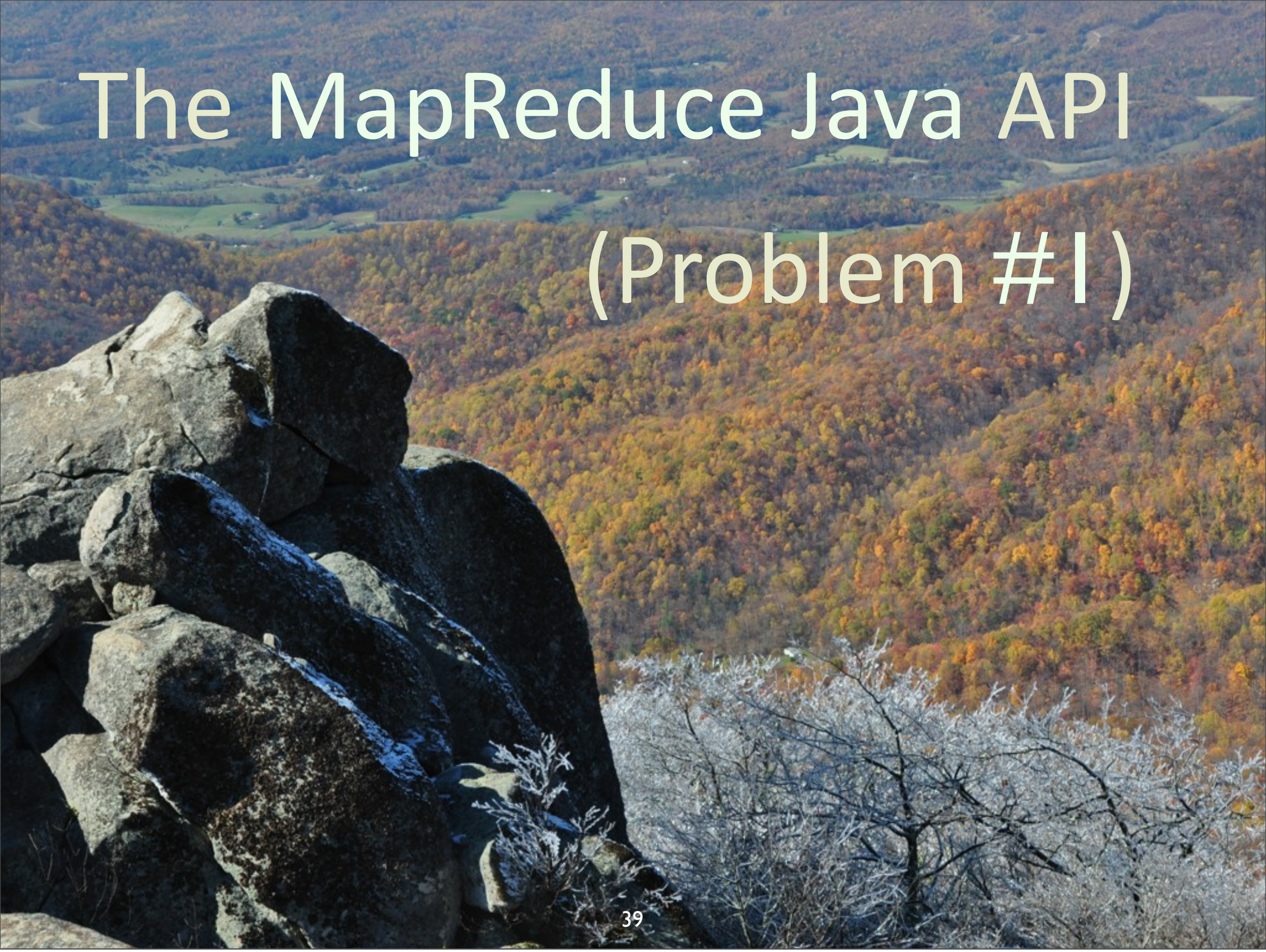






A Java API, but I'll show you Scala code; see my GitHub scala-hadoop project.

# The MapReduce Java API (Problem #1)







A Java API, but I'll show you Scala code; see my GitHub scala-hadoop project.

```
import org.apache.hadoop.io._
import org.apache.hadoop.mapred._
import java.util.StringTokenizer

object SimpleWordCount {

  val one  = new IntWritable(1)
  val word = new Text     // Value will be set in a non-thread-safe way!

  class WCMapper extends MapReduceBase with Mapper[LongWritable, Text, Text, IntWritable] {

    def map(key: LongWritable, valueDocContents: Text,
            output: OutputCollector[Text, IntWritable], reporter: Reporter): Unit = {
      val tokens = valueDocContents.toString.split("\\s+")
      for (wordString <- tokens) {
        if (wordString.length > 0) {
          word.set(wordString.toLowerCase)
          output.collect(word, one)
        }
      }
    }

    class Reduce extends MapReduceBase with Reducer[Text, IntWritable, Text, IntWritable] {

      def reduce(keyWord: Text, valuesCounts: java.util.Iterator[IntWritable],
                 output: OutputCollector[Text, IntWritable], reporter: Reporter): Unit = {
        var totalCount = 0
        while (valuesCounts.hasNext) {
            totalCount += valuesCounts.next.get
         }
         output.collect(keyWord, new IntWritable(totalCount))
      }
  }
```





This is intentionally too small to read. The algorithm is simple, but the framework is in your face. In the next several slides, notice which colors dominate. In this slide, it's green for types (classes), with relatively few yellow functions that implement actual operations.

Still, I've omitted many boilerplate details for configuring and running the job. This is just the "core" MapReduce code. In fact, Word Count is not too bad, but when you get to more complex algorithms, even conceptually simple ideas like relational-style joins and group-bys, the corresponding MapReduce code in this API gets complex and tedious very fast!




Cascading is a Java library that provides higher-level abstractions for building data processing pipelines with concepts familiar from SQL such as a joins, group-bys, etc. It works on top of Hadoop's MapReduce and hides the boilerplate from you.
See http://cascading.org.

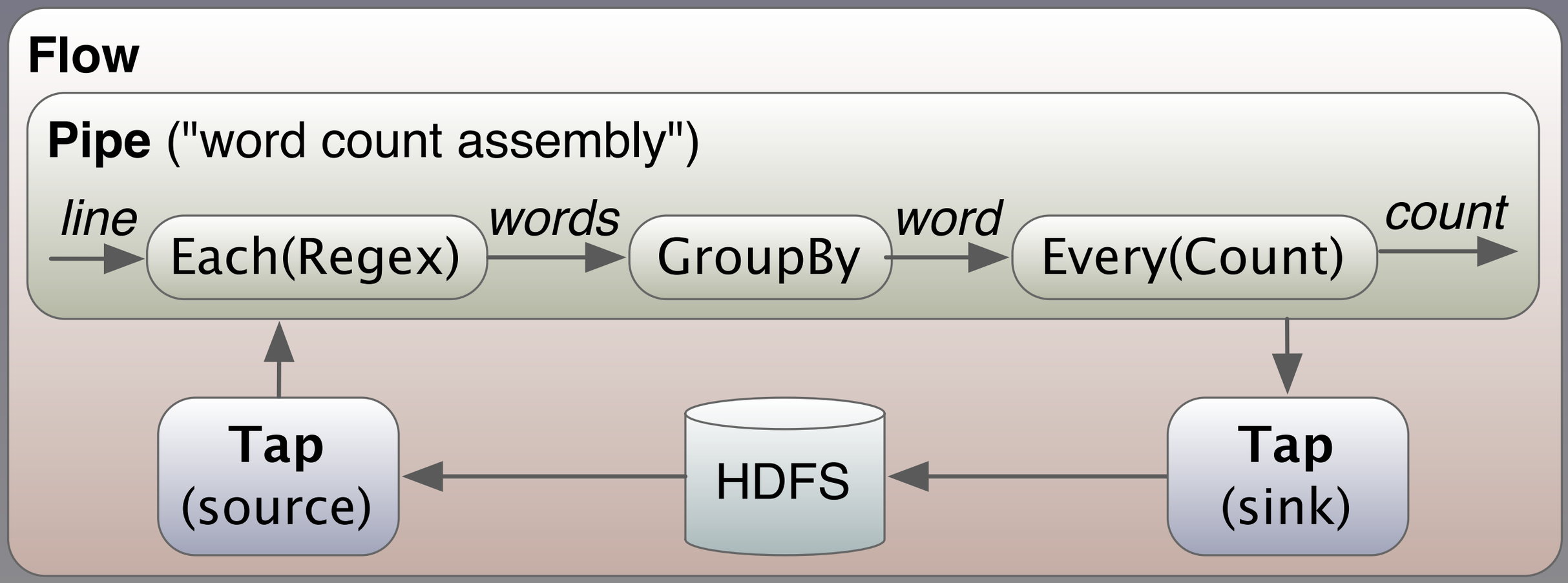

Wednesday, November 14, 12

Cascading is a Java library that provides higher-level abstractions for building data processing pipelines with concepts familiar from SQL such as a joins, group-bys, etc. It works on top of Hadoop's MapReduce and hides the boilerplate from you.
See http://cascading.org.

# Cascading Concepts

*Data flows* consist of source and sink *Taps* connected by *Pipes*.

# Word Count








Schematically, here is what Word Count looks like in Cascading. See http://docs.cascading.org/cascading/1.2/userguide/html/ch02.html for details.

```
import org.cascading.*;
...
public class WordCount {
  public static void main(String[] args) {
    String inputPath  = args[0];
    String outputPath = args[1];
    Properties properties = new Properties();
    FlowConnector.setApplicationJarClass( properties, Main.class );

    Scheme sourceScheme = new TextLine( new Fields( "line" ) );
    Scheme sinkScheme = new TextLine( new Fields( "word", "count" ) );
    Tap source = new Hfs( sourceScheme, inputPath );
    Tap sink   = new Hfs( sinkScheme, outputPath, SinkMode.REPLACE );

    Pipe assembly = new Pipe( "wordcount" );

    String regex = "(?<!\\pL)(?=\\pL)[^ ]*(?<=\\pL)(?!\\pL)";
    Function function = new RegexGenerator( new Fields( "word" ), regex );
    assembly = new Each( assembly, new Fields( "line" ), function );
    assembly = new GroupBy( assembly, new Fields( "word" ) );
    Aggregator count = new Count( new Fields( "count" ) );
    assembly = new Every( assembly, count );

    FlowConnector flowConnector = new FlowConnector( properties );
    Flow flow = flowConnector.connect( "word-count", source, sink, assembly);
    flow.complete();
  }
}
```





Here is the Cascading Java code. It's cleaner than the MapReduce API, because the code is more focused on the algorithm with less boilerplate, although it's not that much shorter. See http://docs.cascading.org/cascading/1.2/userguide/html/ch02.html for details; we won't discuss them here, but just go over highlights.

Note that there is still a lot of green for types, but at least the API emphasizes composing behaviors together.




Scalding is a Scala “DSL” (domain-specific language) that wraps Cascading providing an even more intuitive and more boilerplate-free API for writing MapReduce jobs. https://github.com/twitter/scalding


Wednesday, November 14, 12

Scalding is a Scala “DSL” (domain-specific language) that wraps Cascading providing an even more intuitive and more boilerplate-free API for writing MapReduce jobs. https://github.com/twitter/scalding

```
import com.twitter.scalding._

class WordCountJob(args: Args) extends Job(args) {
  TextLine( args("input") )
    .read
    .flatMap('line -> 'word) {
      line: String => line.trim.toLowerCase.split("\\W+")
    }
    .groupBy('word) { group => group.size('count) }
  }
    .write(Tsv(args("output")))
}
```





This Scala code is almost pure domain logic with very little boilerplate. There are a few minor differences in the implementation. You don't explicitly specify the "Hfs" (Hadoop Distributed File System) taps. That's handled by Scalding implicitly when you run in "non-local" model. Also, I'm using a simpler tokenization approach here, where I split on anything that isn't a "word character" [0-9a-zA-Z_].
There is little green, in part because Scala infers type in many cases. There is a lot more yellow for the methods that do real work!
What if MapReduce, and hence Cascading and Scalding, went obsolete tomorrow? This code is so short, I wouldn't care about throwing it away! I invested little time writing it, testing it, etc.

```
import com.twitter.scalding._

class WordCountJob(args: Args) extends Job(args) {
  TextLine( args("input") )
    .read
    .flatMap('line -> 'word) {
      line: String => line.trim.toLowerCase.split("\\W+")
    }
    .groupBy('word) { group => group.size('count) }
  .write(Tsv(args("output")))
}
```

# That's It!!





This Scala code is almost pure domain logic with very little boilerplate. There are a few minor differences in the implementation. You don't explicitly specify the "Hfs" (Hadoop Distributed File System) taps. That's handled by Scalding implicitly when you run in "non-local" model. Also, I'm using a simpler tokenization approach here, where I split on anything that isn't a "word character" [0-9a-zA-Z_].
There is little green, in part because Scala infers type in many cases. There is a lot more yellow for the methods that do real work!
What if MapReduce, and hence Cascading and Scalding, went obsolete tomorrow? This code is so short, I wouldn't care about throwing it away! I invested little time writing it, testing it, etc.

# You also see this functional improvement with other APIs:

- Crunch (Java)
- Scrunch (Scala)





See https://github.com/cloudera/crunch




Scalding is a Scala “DSL” (domain-specific language) that wraps Cascading providing an even more intuitive and more boilerplate-free API for writing MapReduce jobs. https://github.com/twitter/scalding


Wednesday, November 14, 12

Scalding is a Scala “DSL” (domain-specific language) that wraps Cascading providing an even more intuitive and more boilerplate-free API for writing MapReduce jobs. https://github.com/twitter/scalding

# Spark is a Hadoop MapReduce alternative:

- Distributed computing with in-memory caching.
- Up to 30x faster than MapReduce.





See http://www.spark-project.org/
Why isn't it more widely used; lack of commercial support...

# Spark is a Hadoop MapReduce alternative:

- Designed for machine learning applications.





See http://www.spark-project.org/

```
object WordCountSpark {
  def main(args: Array[String]) {
    val file = spark.textFile(args(0))
    val counts = file.flatMap(line => line.split("\\W+"))
                     .map(word => (word, 1))
                     .reduceByKey(_ + _)
    counts.saveAsTextFile(args(1))
  }
}
```





This spark example is actually closer in construction to the original Java API example, but it cuts down boilerplate to the bare minimum.

```
object WordCountSpark {
  def main(args: Array[String]) {
    val file = spark.textFile(args(0))
    val counts = file.flatMap(line => line.split("\\W+"))
                     .map(word => (word, 1))
                     .reduceByKey(_ + _)
    counts.saveAsTextFile(args(1))
  }
}
```

Also that's it!
Note it's similar to the MapReduce API,
but far more concise.



Wednesday, November 14, 12

This spark example is actually closer in construction to the original Java API example, but it cuts down boilerplate to the bare minimum.




Using SQL when you can! Here are 3 options.

# Use Hive, Shark, or Impala

# (Solution #3)





Using SQL when you can! Here are 3 options.

# Use SQL when you can:

- Hive: SQL on top of MapReduce.
- Shark: Hive ported to Spark.
- Impala: Hive SQL with new, faster back end.





See http://hive.apache.org/ or my book for Hive, http://shark.cs.berkeley.edu/ for shark, and http://www.cloudera.com/content/cloudera/en/products/cloudera-enterprise-core/cloudera-enterprise-RTQ.html for Impala.

# SQL!

```
CREATE TABLE docs (line STRING);
LOAD DATA INPATH '/path/to/docs' INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
(SELECT explode(split(line, '\W+')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```



Wednesday, November 14, 12

This is how you could implement word count in Hive. We're using some Hive built-in functions for tokenizing words in each "line", the one "column" in the docs table.

# SQL!

```
CREATE TABLE docs (line STRING);
LOAD DATA INPATH '/path/to/docs' INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
(SELECT explode(split(line, '\W+')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

## Word Count, again…
## … in HiveQL



Wednesday, November 14, 12

This is how you could implement word count in Hive. We're using some Hive built-in functions for tokenizing words in each "line", the one "column" in the docs table.

# Impala

- HiveQL front end.
- C++ and Java back end.
- Provides up to 100x performance improvement!
- Developed by Cloudera.





See http://www.cloudera.com/content/cloudera/en/products/cloudera-enterprise-core/cloudera-enterprise-RTQ.html.




A good summary presentation: http://www.slideshare.net/shatteredNirvana/pregel-a-system-for-largescale-graph-processing

# Use Graph Processing (Solution #4)





A good summary presentation: http://www.slideshare.net/shatteredNirvana/pregel-a-system-for-largescale-graph-processing

# Google's Page Rank

- Google invented MapReduce,
- … but MapReduce is suboptimal for Page Rank and other graph algorithms.

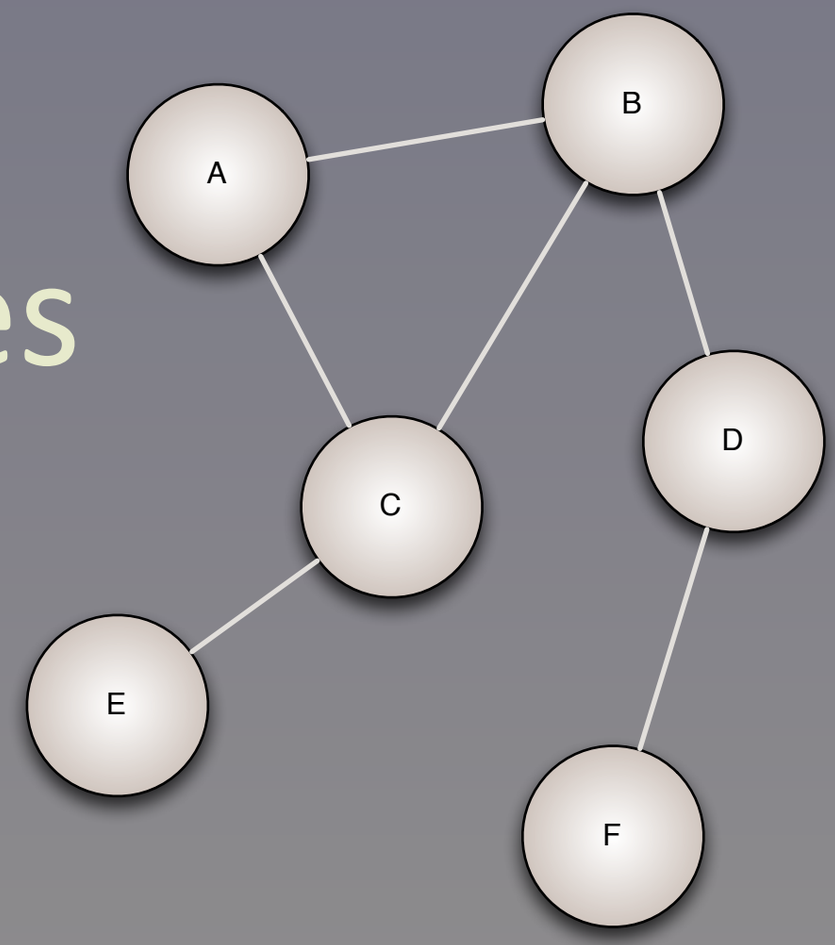




PageRank is the famous algorithm invented by Sergey Brin and Larry Page to index the web. It's the foundation of Google's search engine.

# Why not MapReduce?

- 1 MR job for each iteration that updates all n nodes/edges.
- Graph saved to disk after each iteration.
- …






Wednesday, November 14, 12

The presentation http://www.slideshare.net/shatteredNirvana/pregel-a-system-for-largescale-graph-processing
itemizes all the major issues with using MR to implement graph algorithms.

# Google's Pregel

- Page Rank now runs on a new graph framework: Pregel.
  - Bulk, Synchronous Parallel (BSP).
    - Graphs are first-class citizens.
    - Efficiently processes updates...





Pregel is the name of the river that runs through the city of Königsberg, Prussia (now called Kaliningrad, Ukraine). 7 bridges crossed the river in the city (including to 5 to 2 islands between river branches). Leonhard Euler invented graph theory when we analyzed the question of whether or not you can cross all 7 bridges without retracing your steps (you can't).

# Open-source Alternatives

- Apache Giraph.
- Apache Hama.
- Aurelius Titan.





http://incubator.apache.org/giraph/
http://hama.apache.org/
http://thinkaurelius.github.com/titan/
None is very mature nor has commercial support.




So, I think we have opportunities for a better way...




Wednesday, November 14, 12

So, I think we have opportunities for a better way...



I worked with EJBs a decade ago. The framework was completely invasive into your business logic. There were too many configuration options in XML files. The framework “paradigm” was a poor fit for most problems (like soft real time systems and most algorithms beyond Word Count). Internally, EJB implementations were inefficient and hard to optimize, because they relied on poorly considered object boundaries that muddled more natural boundaries. (I’ve argued in other presentations and my “FP for Java Devs” book that OOP is a poor modularity tool…)

The fact is, Hadoop reminds me of EJBs in almost every way. It works okay and people do get stuff done, but just as the Spring Framework brought an essential rethinking to Enterprise Java, I think there is an essential rethink that needs to happen in Big Data. The functional programming community, is well positioned to create it.




I worked with EJBs a decade ago. The framework was completely invasive into your business logic. There were too many configuration options in XML files. The framework “paradigm” was a poor fit for most problems (like soft real time systems and most algorithms beyond Word Count). Internally, EJB implementations were inefficient and hard to optimize, because they relied on poorly considered object boundaries that muddled more natural boundaries. (I’ve argued in other presentations and my “FP for Java Devs” book that OOP is a poor modularity tool…)

The fact is, Hadoop reminds me of EJBs in almost every way. It works okay and people do get stuff done, but just as the Spring Framework brought an essential rethinking to Enterprise Java, I think there is an essential rethink that needs to happen in Big Data. The functional programming community, is well positioned to create it.



Java is really the wrong language for this and the continued use by Big Data vendors is slowing down progress. It offers the wrong abstractions, objects instead mathematically-inspired functional programming, and it's incredibly verbose when compared to modern alternatives.

# Stop Using Java!



Java is really the wrong language for this and the continued use by Big Data vendors is slowing down progress. It offers the wrong abstractions, objects instead mathematically-inspired functional programming, and it's incredibly verbose when compared to modern alternatives.




Why is Functional Programming better for Big Data? The work we do with data is inherently mathematical transformations and FP is inspired by math. Hence, it's naturally a better fit, much more so than object-oriented programming. And, modern languages like Scala and Clojure are more concise and better at eliminating boilerplate.
SQL has succeeded all these years because it is also inspired by math, e.g., set theory.


Wednesday, November 14, 12

Why is Functional Programming better for Big Data? The work we do with data is inherently mathematical transformations and FP is inspired by math. Hence, it's naturally a better fit, much more so than object-oriented programming. And, modern languages like Scala and Clojure are more concise and better at eliminating boilerplate.
SQL has succeeded all these years because it is also inspired by math, e.g., set theory.



We already have the right model in the collection APIs that come with functional languages. They are far better engineered for intuitive data transformations. They provide the right abstractions and hide boilerplate. In fact, they make it relatively easy to optimization implementations for parallelization. The Scala collections offer parallelization with a tiny API call. Spark and Cascading transparently distribute collections across a cluster.

# Functional Collections.



We already have the right model in the collection APIs that come with functional languages. They are far better engineered for intuitive data transformations. They provide the right abstractions and hide boilerplate. In fact, they make it relatively easy to optimization implementations for parallelization. The Scala collections offer parallelization with a tiny API call. Spark and Cascading transparently distribute collections across a cluster.



We can start using new, more efficient compute models, like Spark, Pregel, and Impala today. Of course, you have to consider maturity, viability, and support issues in large organizations. So if you want to wait until these alternatives are more mature, then at least use better APIs for Hadoop!

# New Compute Models.



We can start using new, more efficient compute models, like Spark, Pregel, and Impala today. Of course, you have to consider maturity, viability, and support issues in large organizations. So if you want to wait until these alternatives are more mature, then at least use better APIs for Hadoop!

# Final Thought:

**Luke Wroblewski**

@lukew

Follow

The new tech mullet: simple mobile interactions up front, big data in the back.

12 Nov 12

Reply Retweet Favorite

# Questions?

QCon Motorola Solutions SABA
November 13, 2012

dean.wampler@thinkbiganalytics.com


THiNK BIG ANALYTICS

# Questions?

QCon Motorola Solutions SABA
November 13, 2012

dean.wampler@thinkbiganalytics.com


THiNK BIG ANALYTICS