

TechMesh London 2012
December 5, 2012
dean@deanwampler.com
polyglotprogramming.com/talks

MapReduce and Its Discontents



SOME RIGHTS RESERVED

Monday, April 22, 13

Beyond MapReduce: it's been a useful technology, but has a "first generation" feel. What's next?

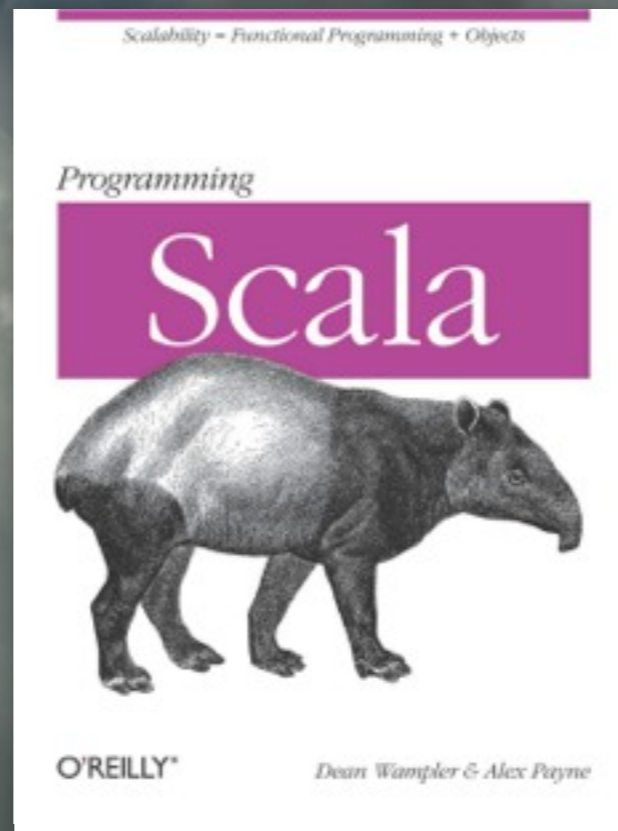
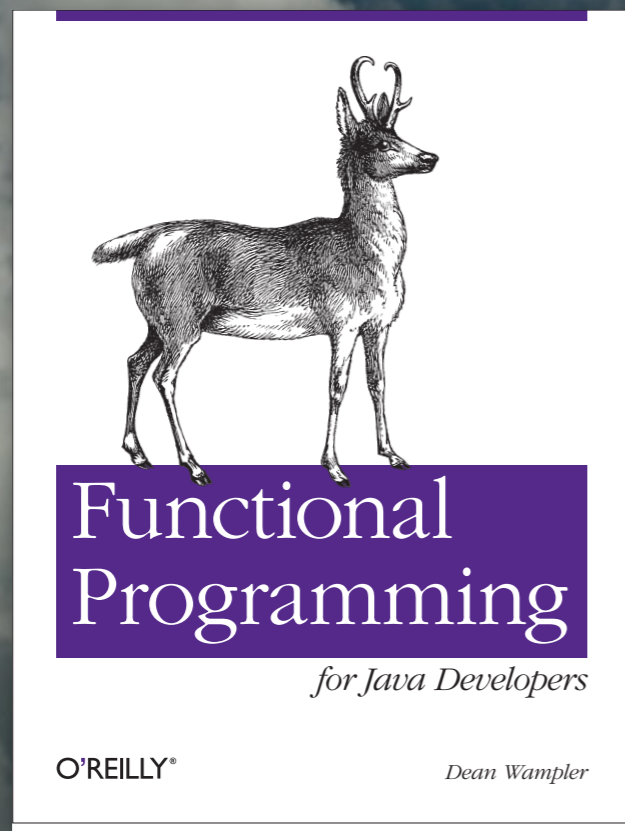
Copyright © Dean Wampler, 2011–2013, All Rights Reserved. Photos can only be used with permission. Otherwise, the content is free to use.

About Me...

dean@deanwampler.com

[@deanwampler](https://twitter.com/deanwampler)

github.com/deanwampler



Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

My books and contact information.

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.



3 Trends

4

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

Three trends influence my thinking...

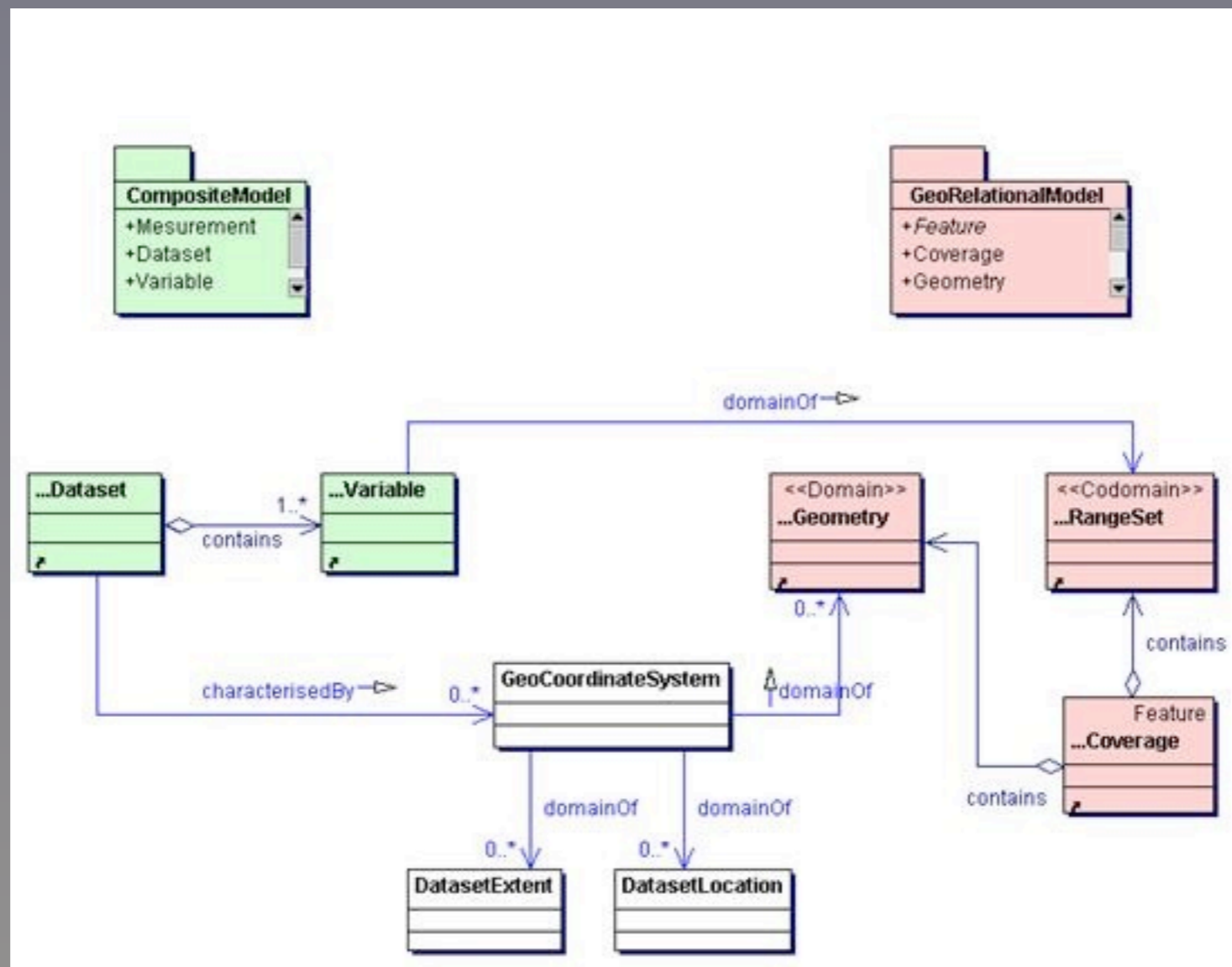
Data Size ↑



Monday, April 22, 13

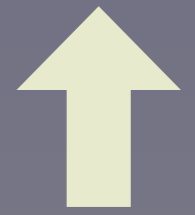
Data volumes are obviously growing... rapidly.
Facebook now has over 600PB (Petabytes) of data in Hadoop clusters!

Formal Schemas ↓



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, and we need to integrate so many disparate sources of data. So, using relatively-agnostic software, e.g., collections of things where the software is more agnostic about the structure of the data and the domain, tends to be faster to develop, test, and deploy. Put another way, we find it more useful to build somewhat agnostic applications and drive their behavior through data...

Data-Driven Programs



Monday, April 22, 13

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, as you might expect...)



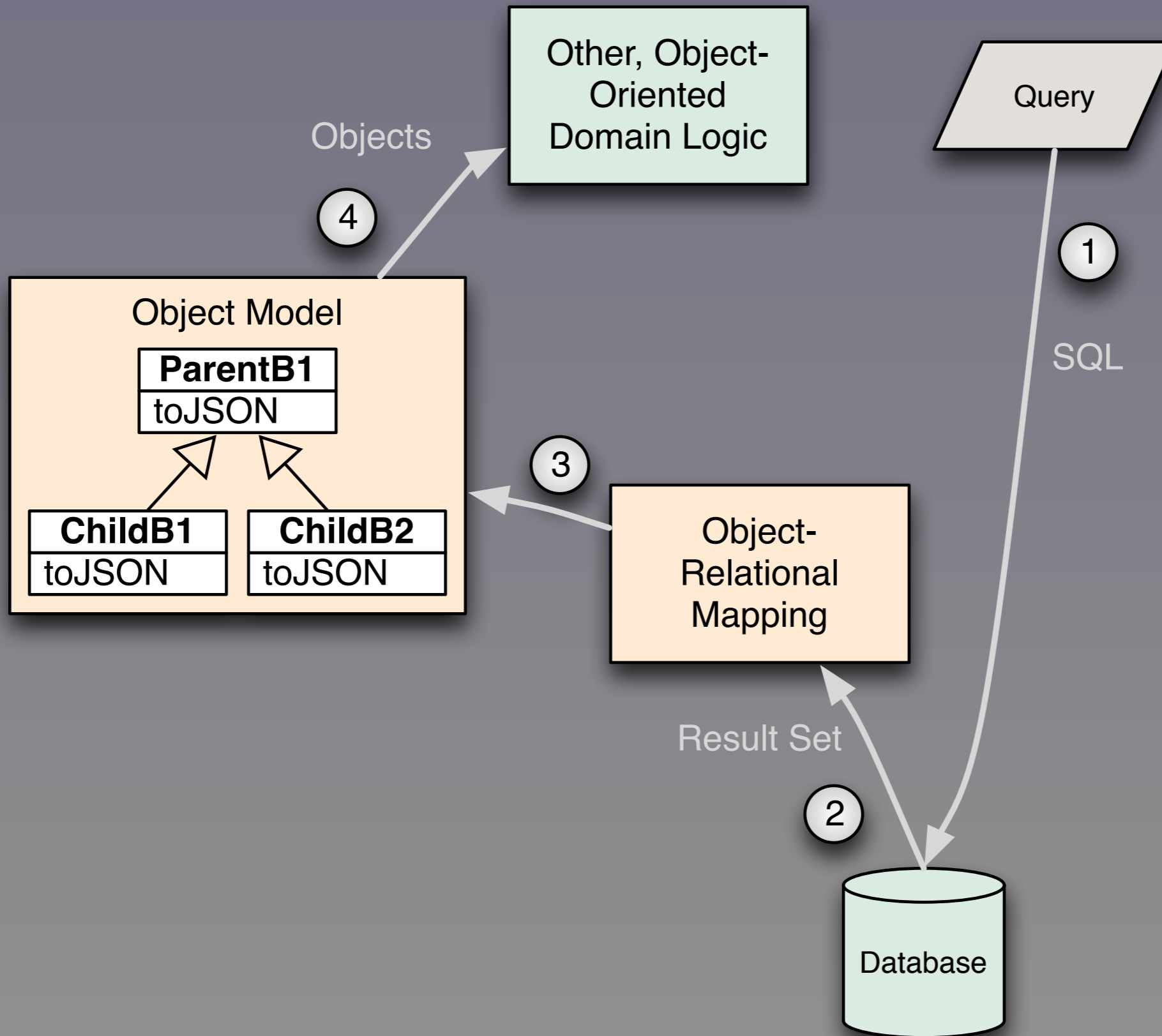
Big Data Architectures

8

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

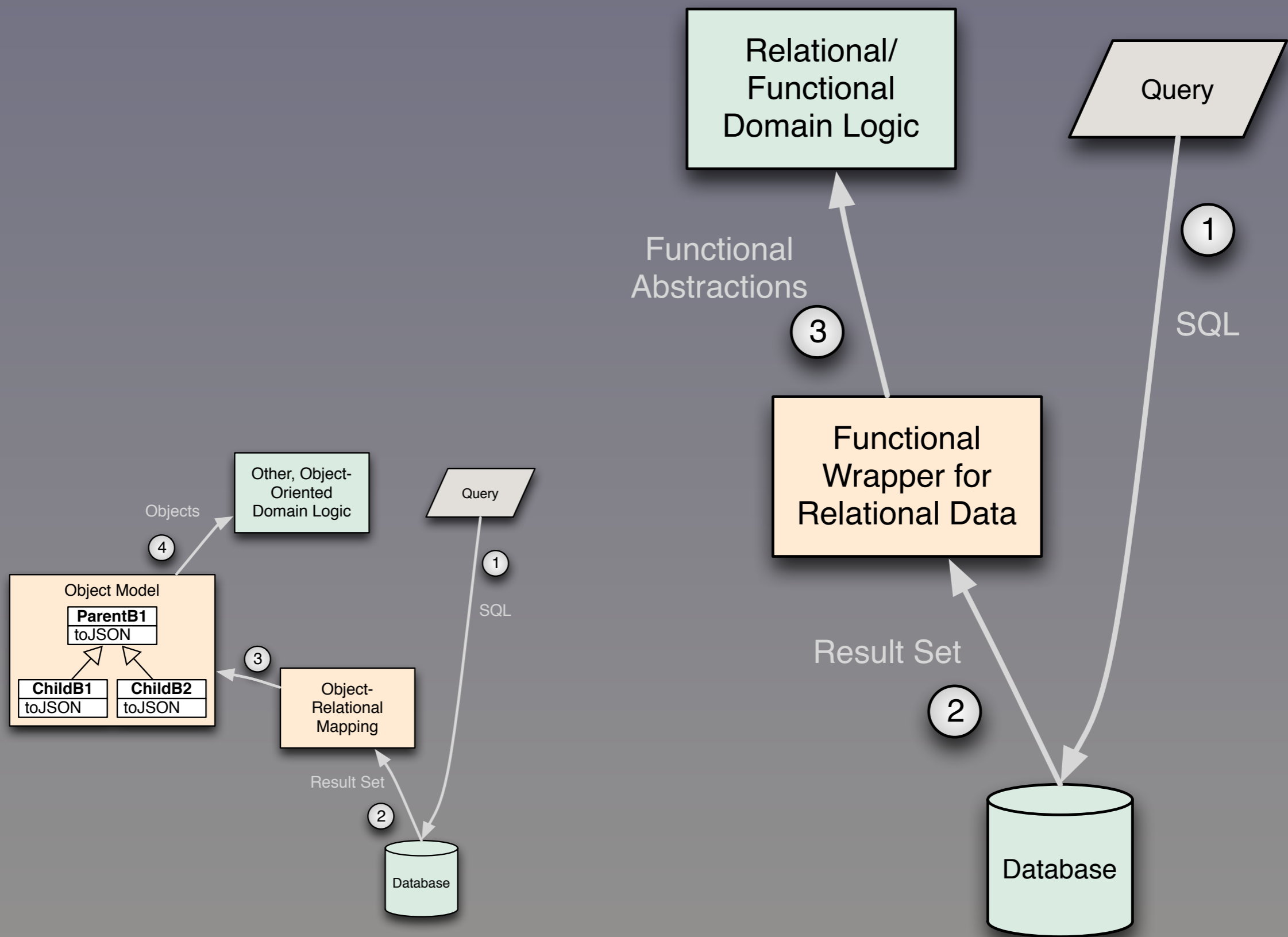
What should software architectures look like for these kinds of systems?



Monday, April 22, 13

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...)

The toJSON methods are there because we often convert these object graphs back into fundamental structures, such as the maps and arrays of JSON so we can send them to the browser!



Monday, April 22, 13

But the traditional systems are a poor fit for this new world: 1) they add too much overhead in computation (the ORM layer, etc.) and memory (to store the objects). Most of what we do with data is mathematical transformation, so we're far more productive (and runtime efficient) if we embrace fundamental data structures used throughout (lists, sets, maps, trees) and build rich transformations into those libraries, transformations that are composable to implement business logic.

- Focus on:

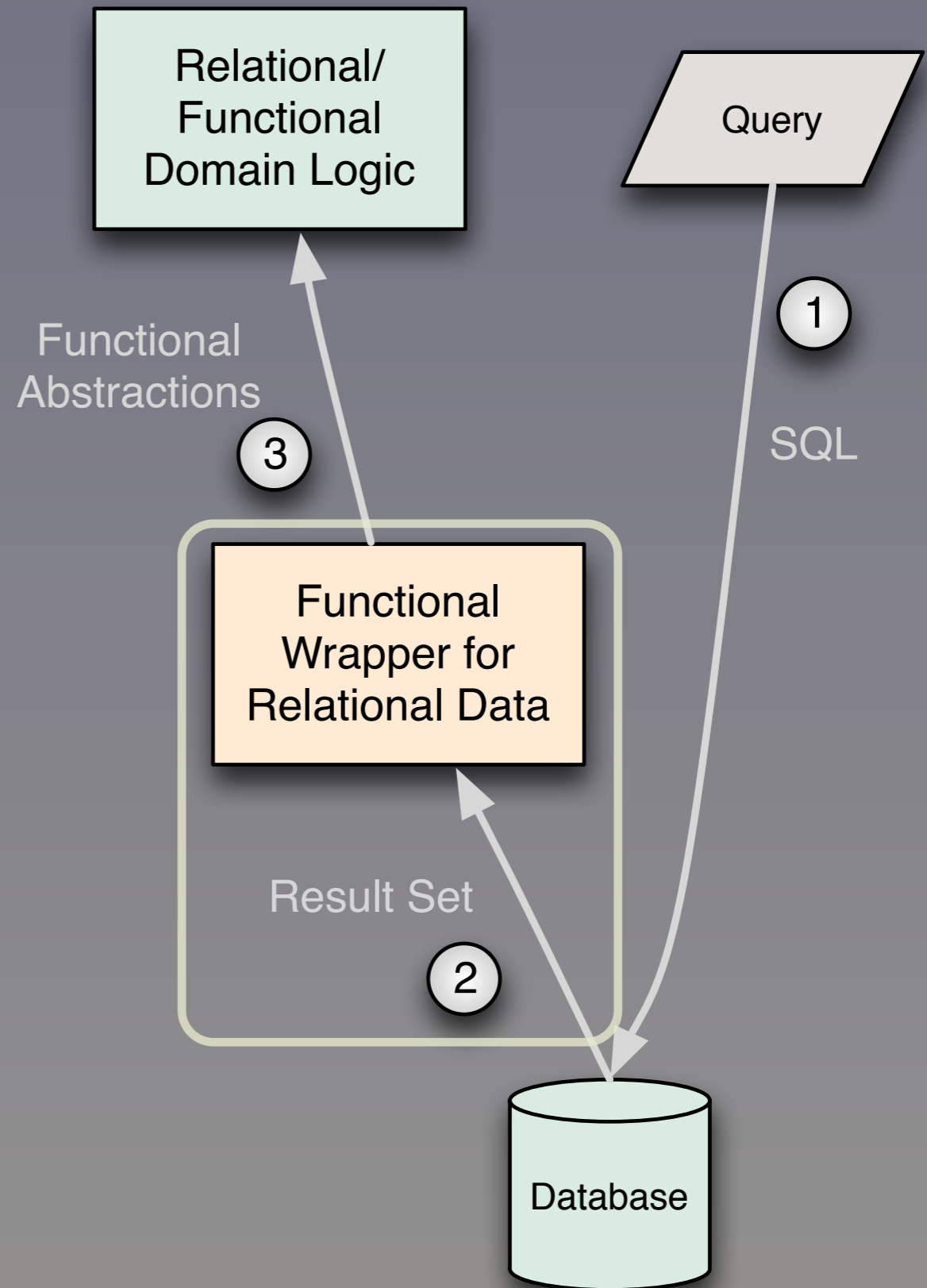
- Lists

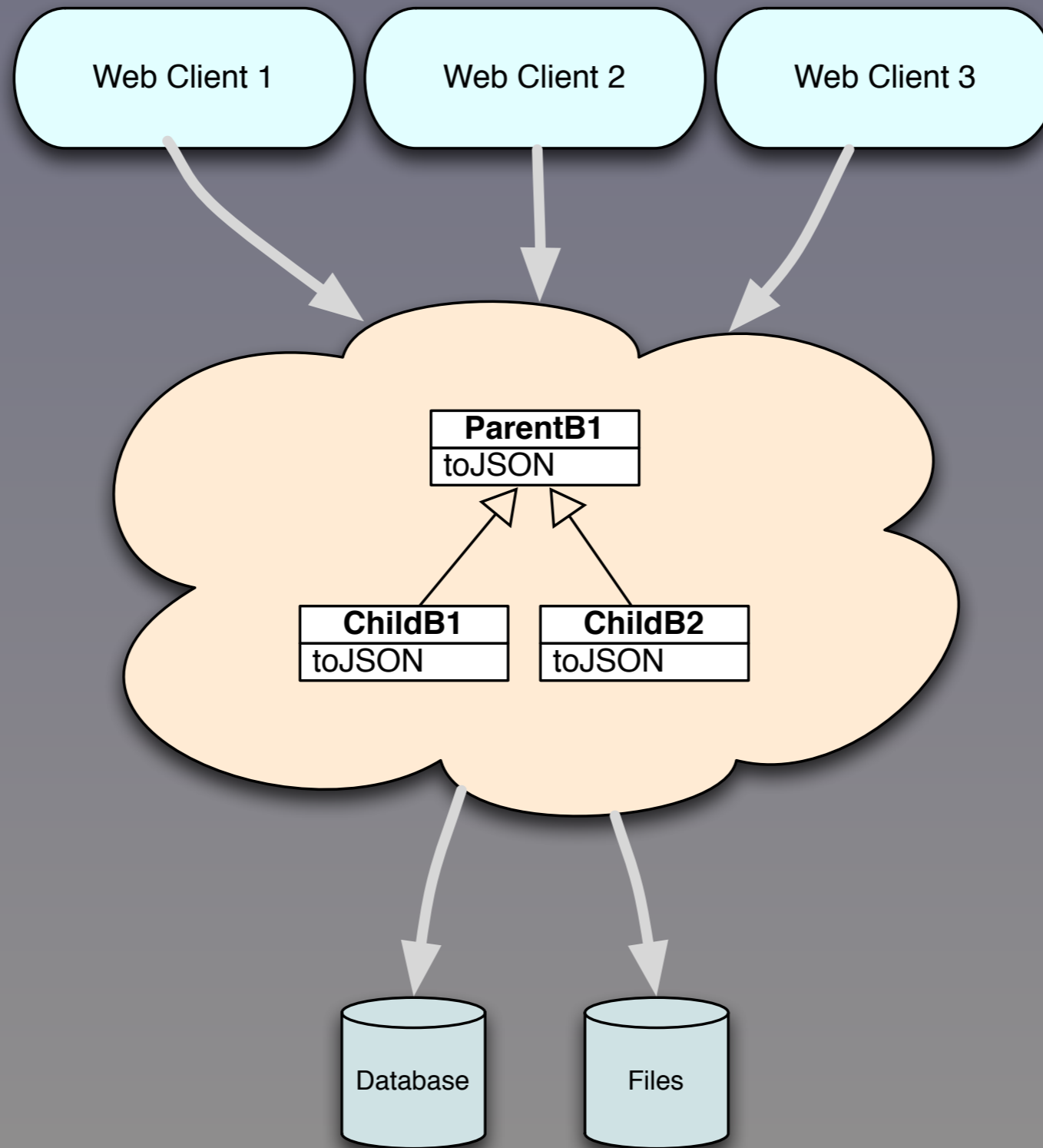
- Maps

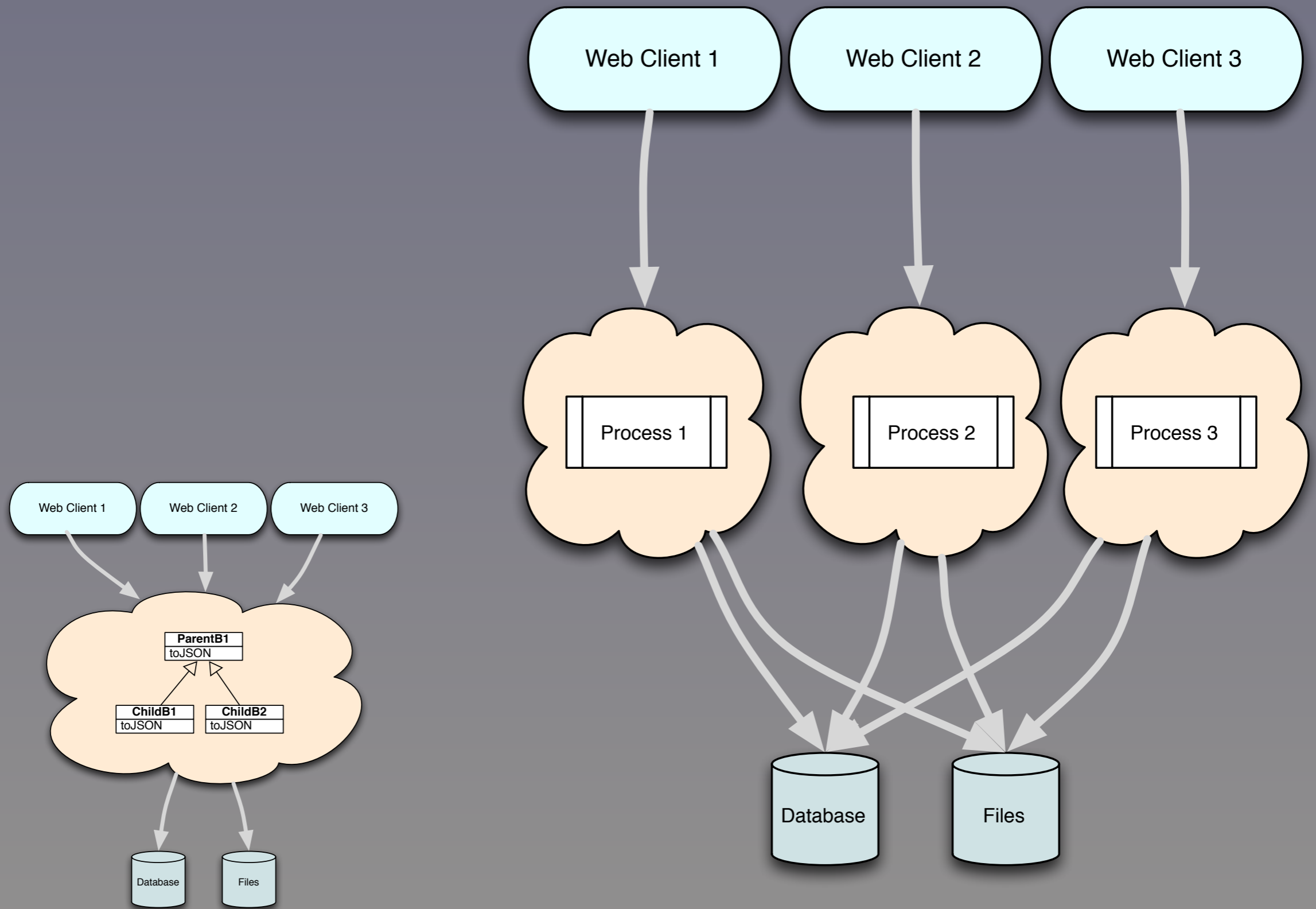
- Sets

- Trees

- ...





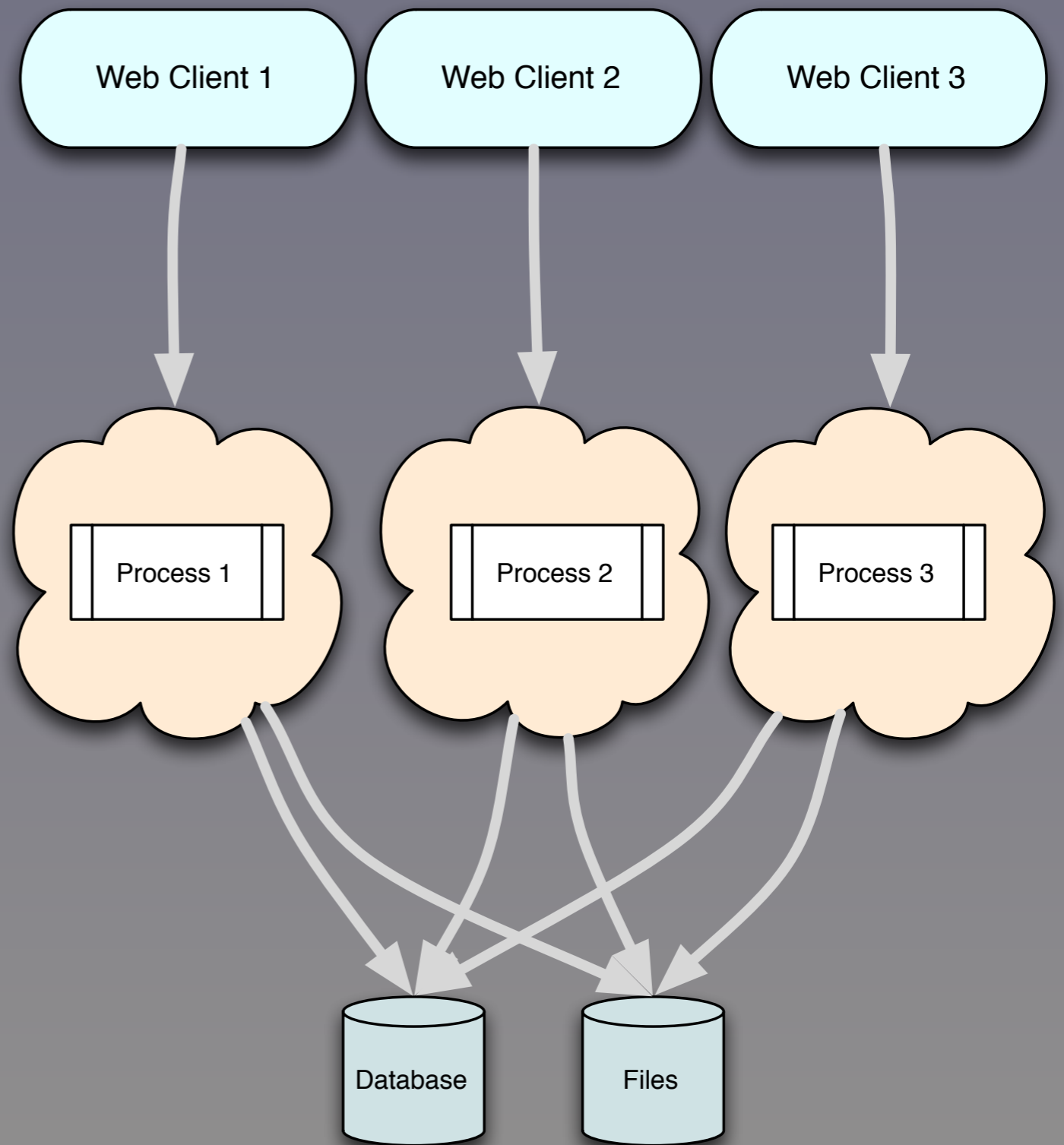


Monday, April 22, 13

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. Each "ProcessN" could be a parallel copy of another process, for horizontal, "shared-nothing" scalability, or some of these processes could be other services...

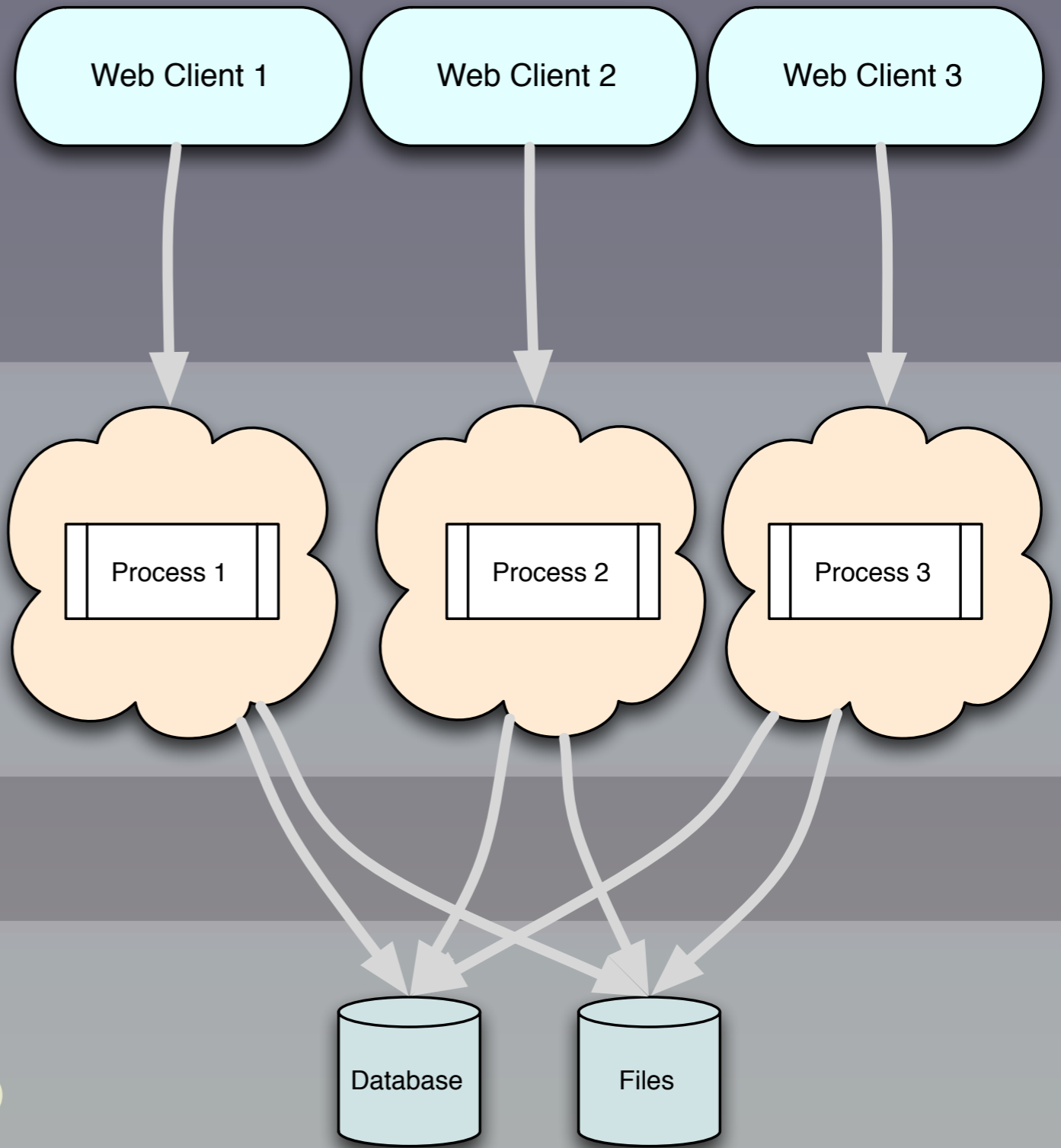
Smaller, focused services scale better, especially horizontally. They also don't encapsulate more business logic than is required, and this (informal) architecture is also suitable for scaling ML and related algorithms.

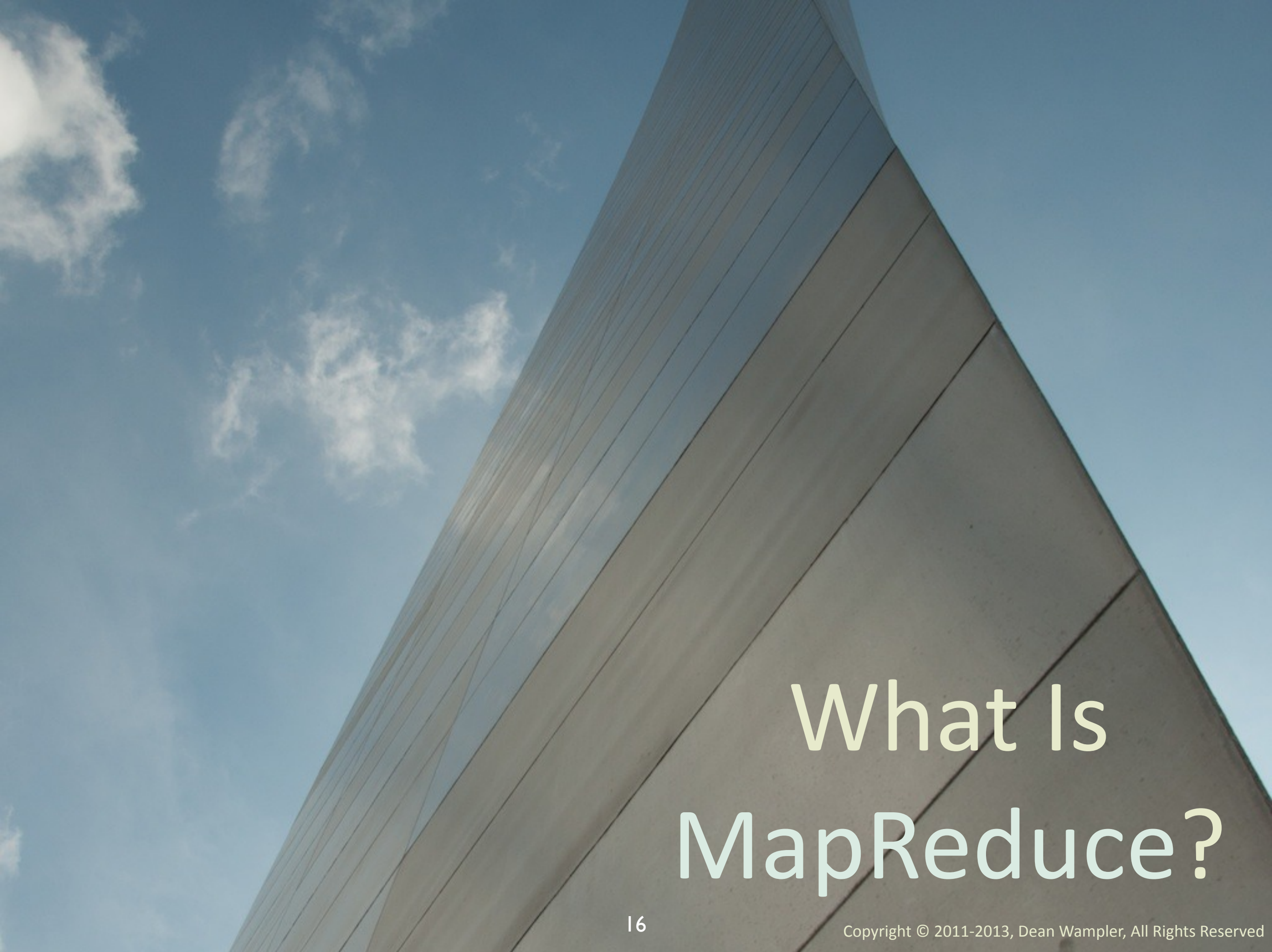
- Data Size ↑
- Formal Schema ↓
- Data-Driven Programs ↑



- MapReduce

- Distributed FS





What Is MapReduce?

16

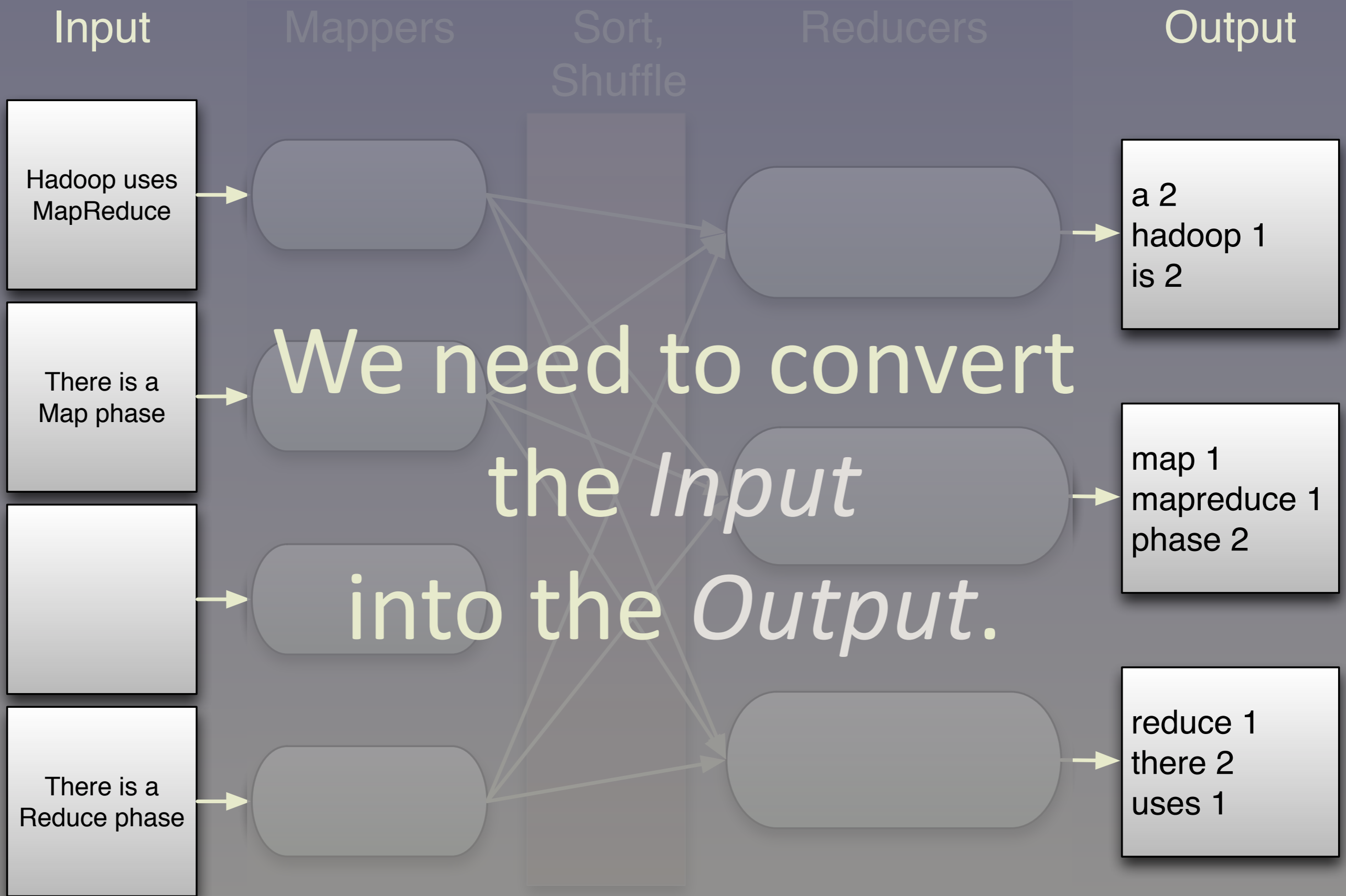
Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

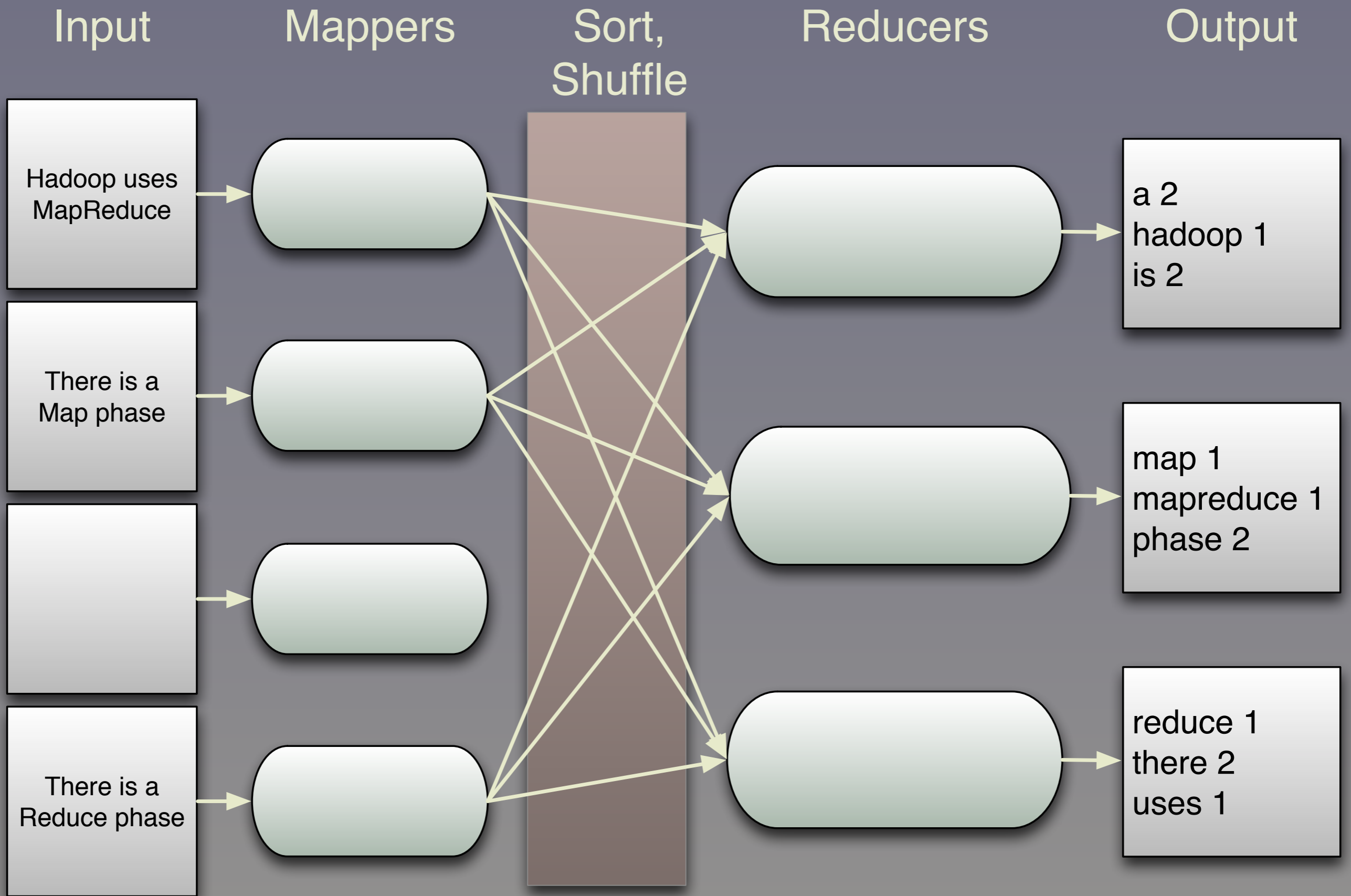
MapReduce in Hadoop

Let's look at a
MapReduce algorithm:
WordCount.

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



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.



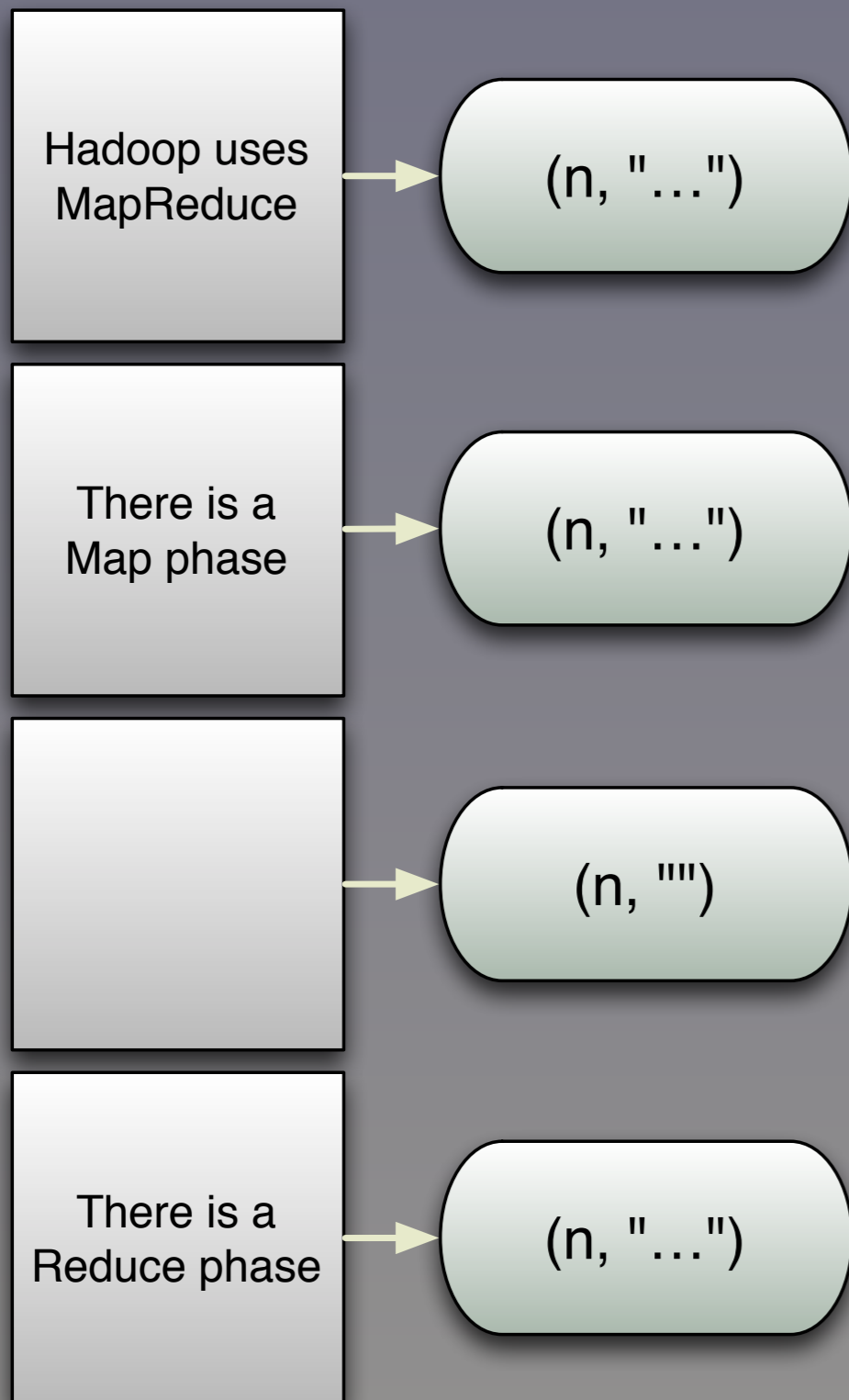
Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

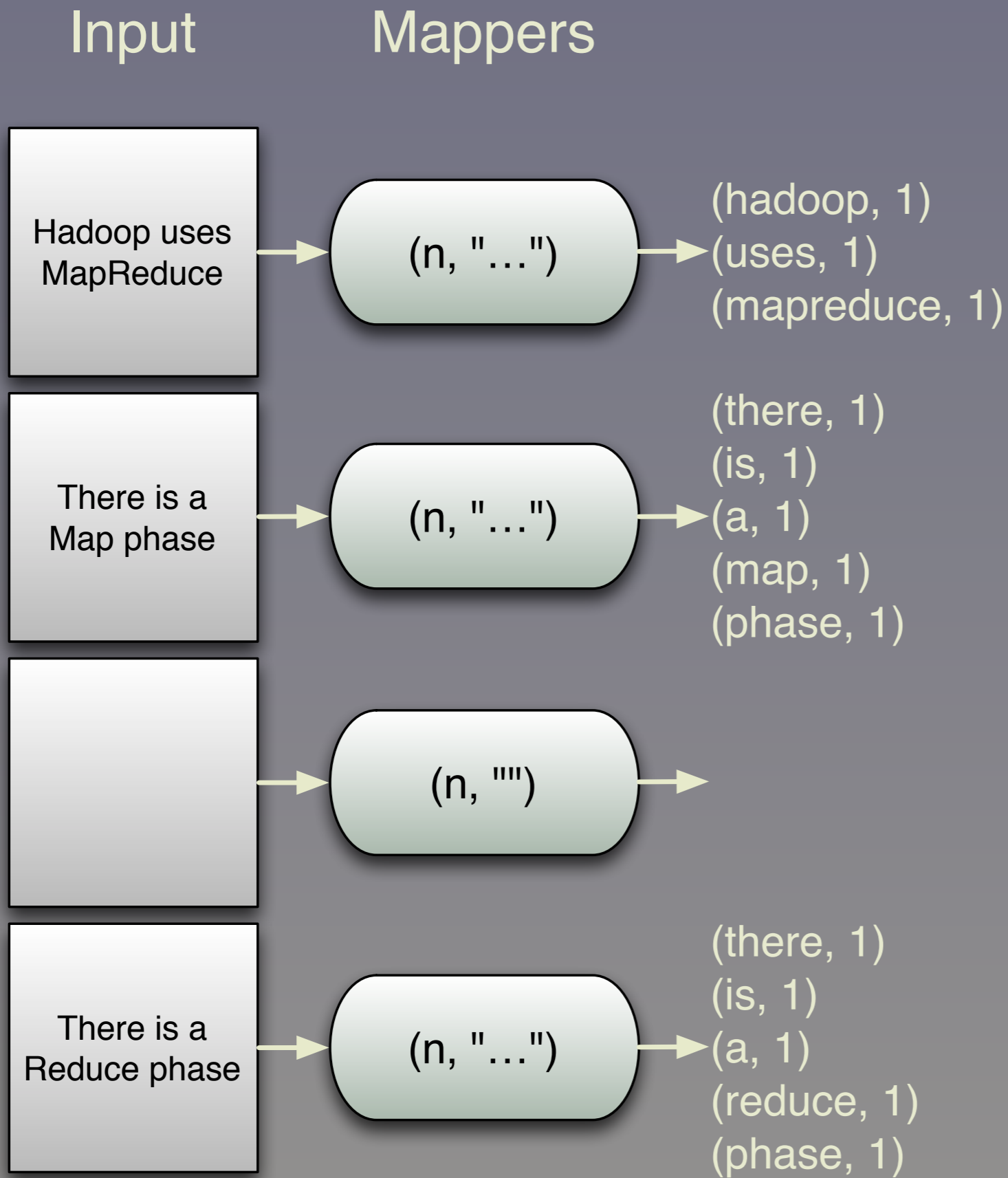
Here is a schematic view of the steps in Hadoop MapReduce. Each Input file is read by a single Mapper process (default: can be many-to-many, as we'll see later). The Mappers emit key-value pairs that will be sorted, then partitioned and "shuffled" to the reducers, where each Reducer will get all instances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

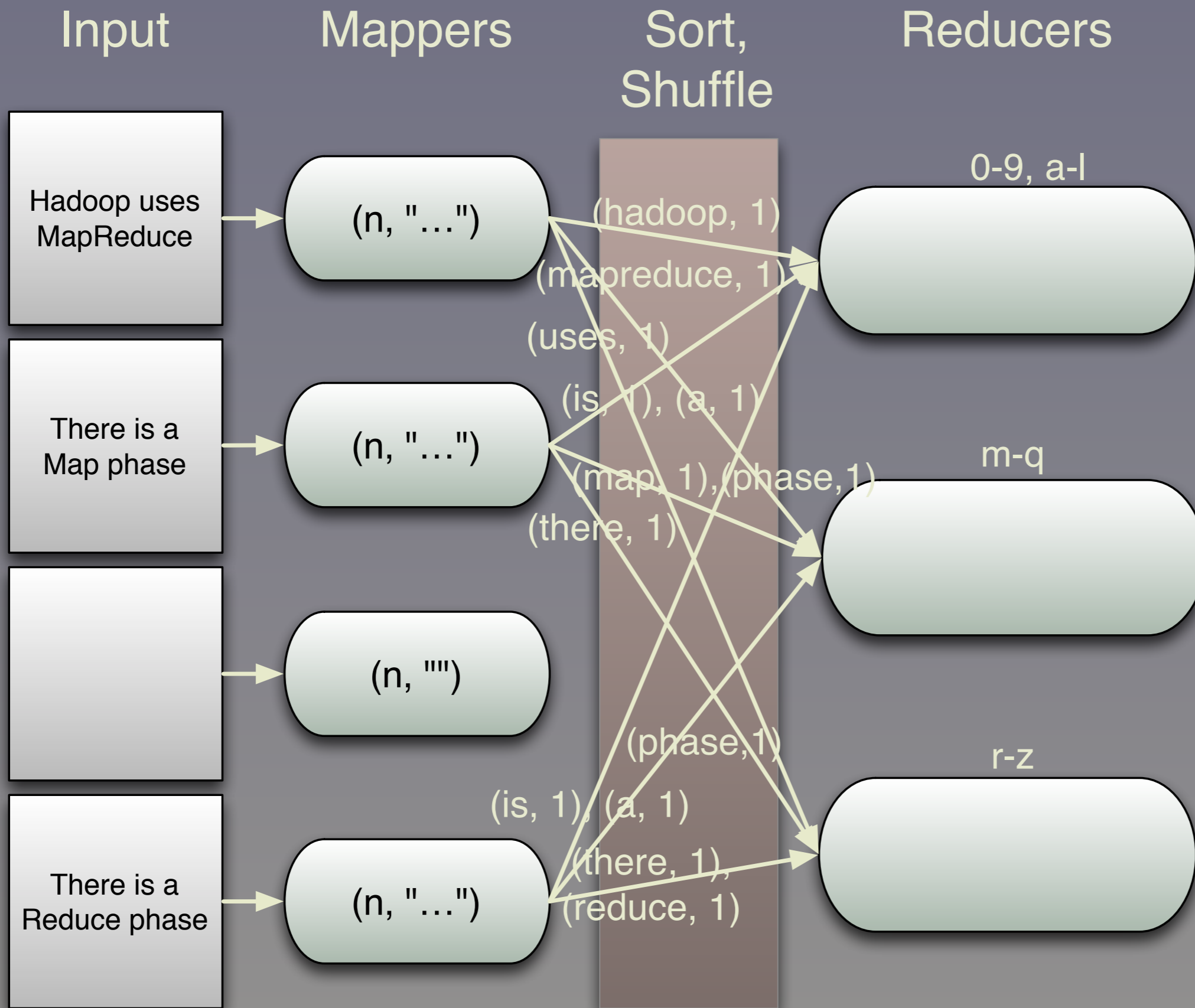
Mappers



Each document gets a mapper. 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 each document's contents in a box and 1 mapper task (JVM process) per document. Large documents might get split to several mapper tasks. The mappers tokenize each line, one at a time, converting all words to lower case and counting them...

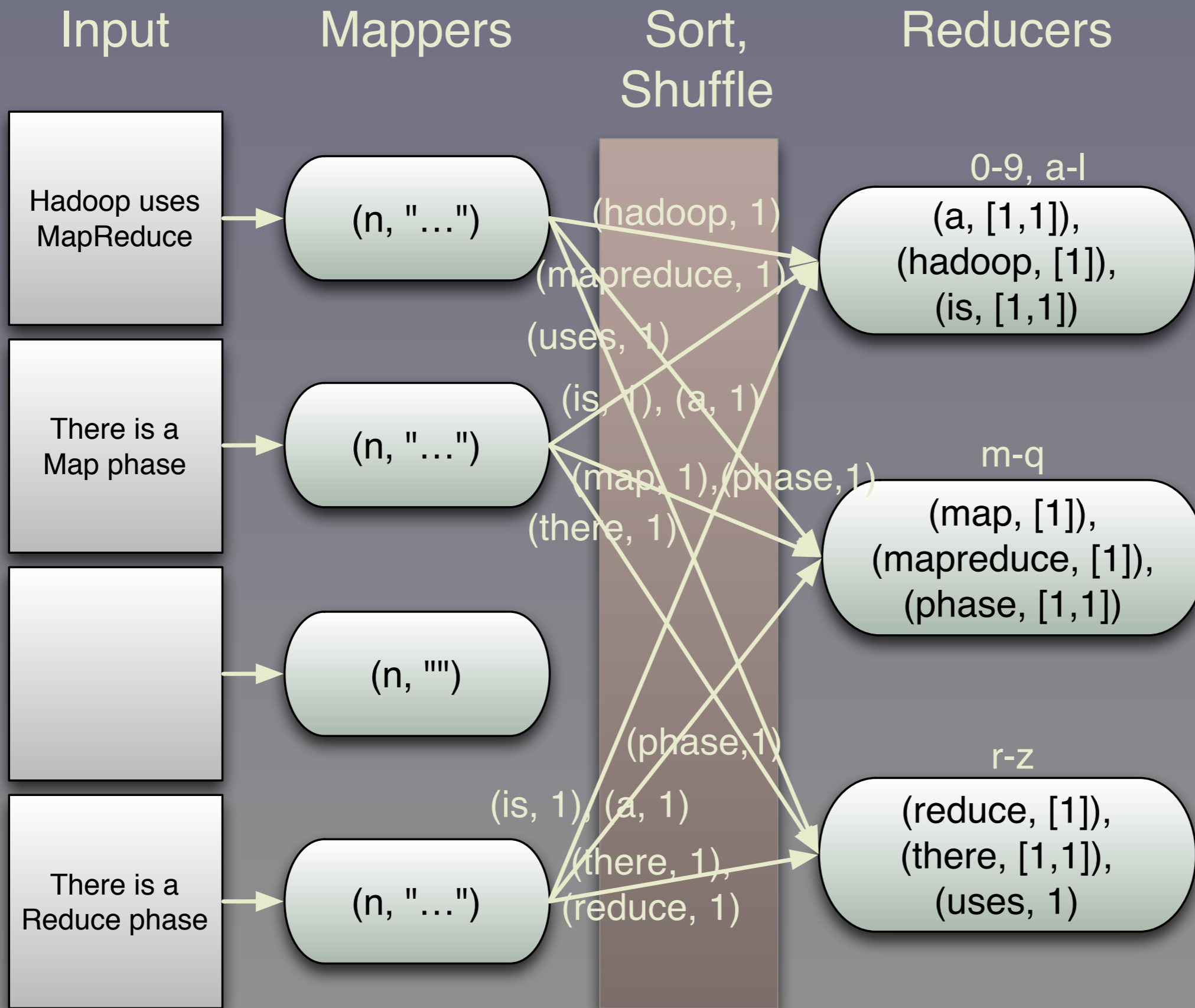


The mappers emit key-value pairs, where each key is one of the words, and the value is the count. In the most naive (but also most memory efficient) implementation, each mapper simply emits (word, 1) each time “word” is seen. However, this is IO inefficient! Note that the mapper for the empty doc. emits no pairs, as you would expect.

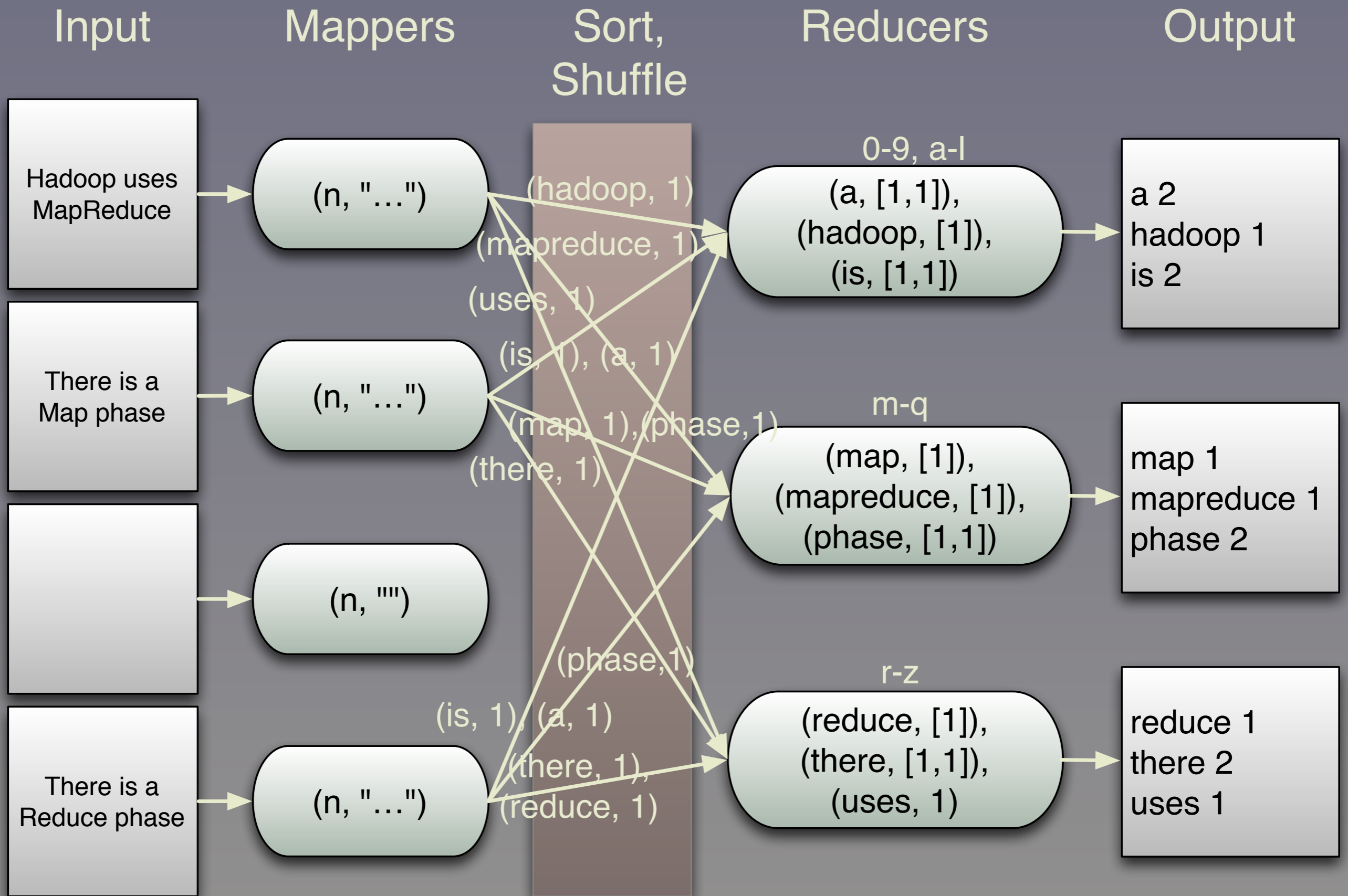


The mappers themselves don't decide to which reducer each pair should be sent. Rather, the job setup configures what to do and the Hadoop runtime enforces it during the Sort/Shuffle phase, where the key-value pairs in each mapper are sorted by key (that is locally, not globally) and then the pairs are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

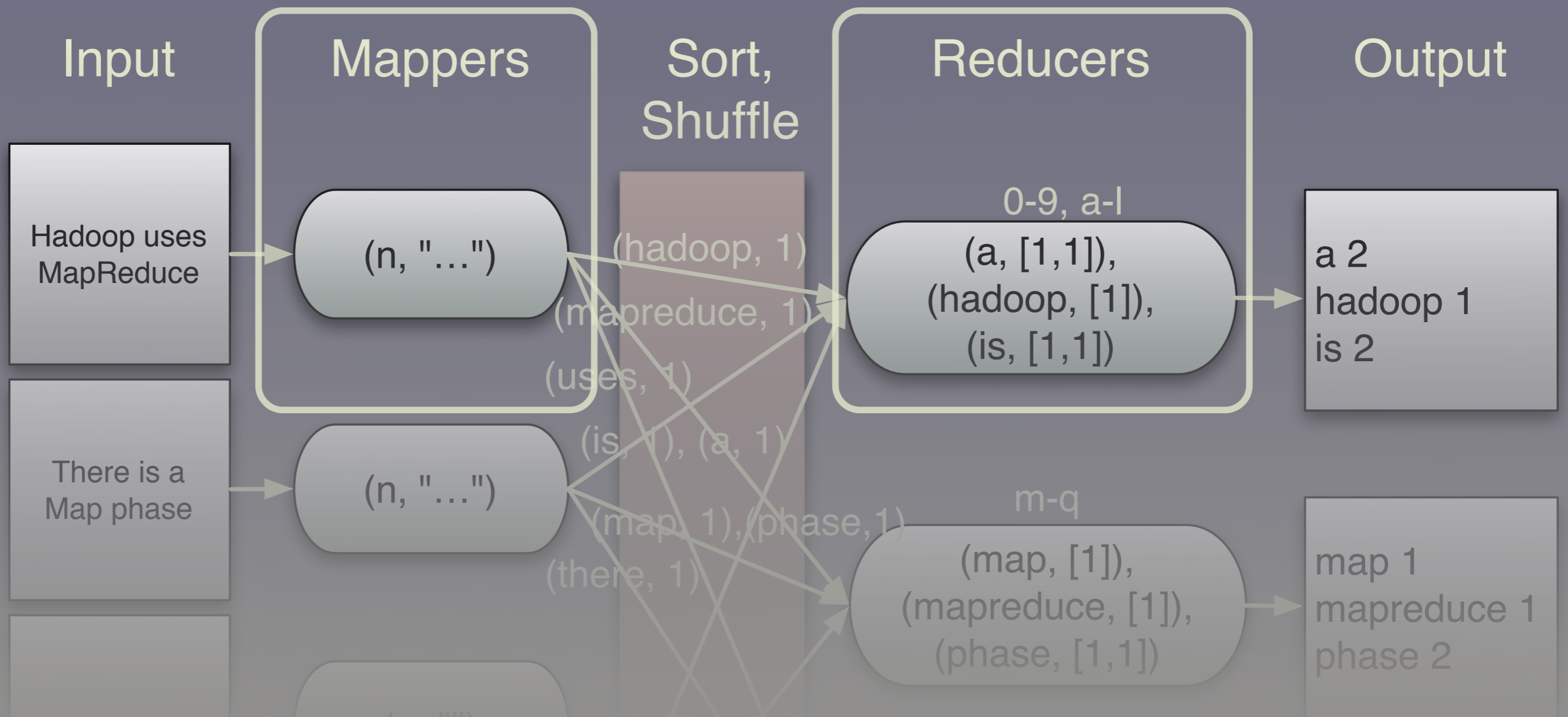
Note how we partitioned the reducers, by first letter of the keys. (By default, MR just hashes the keys and distributes them modulo # of reducers.)



The reducers are passed each key (word) and a collection of all the values for that key (the individual counts emitted by the mapper tasks). The MR framework creates these collections 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.



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



History of MapReduce

26

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

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

How would you *index* the web?

The Google logo is displayed in its characteristic multi-colored font: blue 'G', red 'o', yellow 'o', blue 'g', green 'l', and red 'e'.

What is the meanin| 

- what is the meaning of life
- what is the meaning of life 42
- what is the meaning of pumped up kicks
- what is the meaning of halloween
- what is the meaning of love
- what is the meaning of labor day
- what is the meaning of slope
- what is the meaning of homecoming
- what is the meaning of my last name
- what is the meaning of a promise ring

How would you *index the web?*

Google

what is the meaning of life



Search

About 48,900,000 results (0.26 seconds)

Everything

Images

Maps

Videos

News

Shopping

[Meaning of life - Wikipedia, the free encyclopedia](#)

en.wikipedia.org/wiki/Meaning_of_life

The **meaning of life** constitutes a philosophical question concerning the purpose and significance of life or existence in general. This concept can be expressed ...

[Questions - Western philosophical perspectives - East Asian philosophy](#)



[The Meaning of Life](#)

users.aristotle.net/~diogenes/meaning1.htm

Why do you want to know the **meaning of life**? Often people ask this question when they really want the answer to some other question. Let's try and get those ...

Monday, April 22, 13

Did Google search the entire web in 0.26 seconds to find these ~49M results?

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.

Google's famous *Page Rank* algorithm.

In the early 2000s,
Google invented server
infrastructure to support
PageRank, etc...

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

Monday, April 22, 13

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-

Monday, April 22, 13

The compute model for processing all that data is MapReduce. It handles lots of boilerplate, like breaking down jobs into tasks, distributing the tasks around the cluster, monitoring the tasks, etc. You write your algorithm to the MR programming model.

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

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

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

They started clean-
room versions of
MapReduce and *GFS*...

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



Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

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*.

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

We can't build vertical systems big enough and if we could, they would cost a fortune!

Hadoop Design Goals

Maximize I/O
Performance!!

And *parallelize execution,*
run on server-class,
commodity hardware.

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

By design, Hadoop is
great for batch mode
data crunching.

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

... but less so for “real-time” event handling, as we’ll discuss...

MapReduce and its Discontents

41

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

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

#1

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

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

Even word count is not “obvious”. When you get to fancier stuff like joins, group-bys, etc., the mapping from the algorithm to the implementation is not trivial at all. In fact, implementing algorithms in MR is now a specialized body of knowledge...

#2

For *Hadoop* in
particularly,
the *Java API* is
hard to use.

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

The Hadoop Java API is even more verbose and tedious to use than it should be.

```

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import java.util.StringTokenizer;

class WCMapper extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {

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

    @Override
    public void map(LongWritable key, Text valueDocContents,
        OutputCollector<Text, IntWritable> output, Reporter reporter) {
        String[] tokens = valueDocContents.toString.split("\\s+");
        for (String wordString: tokens) {
            if (wordString.length > 0) {
                word.set(wordString.toLowerCase());
                output.collect(word, one);
            }
        }
    }
}

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

    public void reduce(Text keyWord, java.util.Iterator<IntWritable> valuesCounts,
        OutputCollector<Text, IntWritable> output, Reporter reporter) {
        int totalCount = 0;
        while (valuesCounts.hasNext) {
            totalCount += valuesCounts.next.get();
        }
        output.collect(keyWord, new IntWritable(totalCount));
    }
}

```

This is intentionally too small to read and we're not showing the main routine, which doubles the code size. 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.

The main routine I've omitted contains 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!

Use Cascading (Java) (Solution #1a)

45

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

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 most of the boilerplate from you.

See <http://cascading.org>.

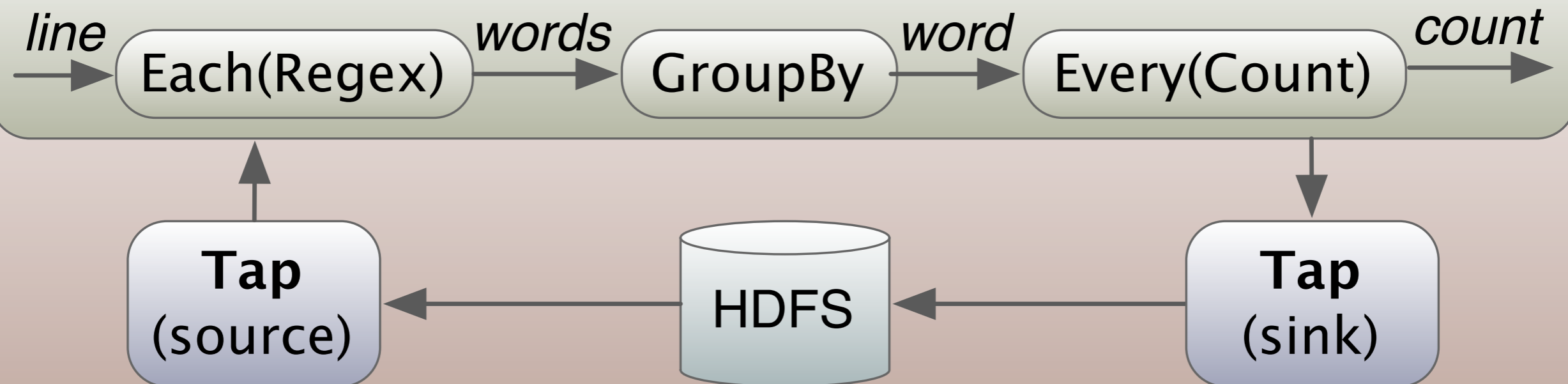
Cascading Concepts

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

Word Count

Flow

Pipe ("word count assembly")



```

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 looks like it's not that much shorter. HOWEVER, this is all the code, where as previously I omitted the setup (main) code. See <http://docs.cascading.org/cascading/1.2/userguide/html/ch02.html> for details of the API features used here; we won't discuss them here, but just mention some highlights.

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

Use Scalding (Scala) (Solution #1b)

Monday, April 22, 13

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>

Scala is a new JVM language that modernizes Java’s object-oriented (OO) features and adds support for functional programming, as we discussed previously and we’ll revisit shortly.

```

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 functions 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.

Other Improved APIs:

- Crunch (Java) & Scrunch (Scala)
- Scoobi (Scala)
- ...

Use Spark (Scala) (Solution #2)

Spark is a Hadoop MapReduce alternative:

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

Spark is a Hadoop MapReduce alternative:

- Originally designed for machine learning applications.

```
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.

Use Hive, Shark, or Impala (Solution #3)

Use SQL when you can!

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

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

Impala

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

#3

It's not suitable for
“real-time”
event processing.

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

For typical web/enterprise systems, “real-time” is up to 100s of milliseconds, so I’m using the term broadly (but following common practice in this industry). True real-time systems, such as avionics, have much tighter constraints.



Storm!

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

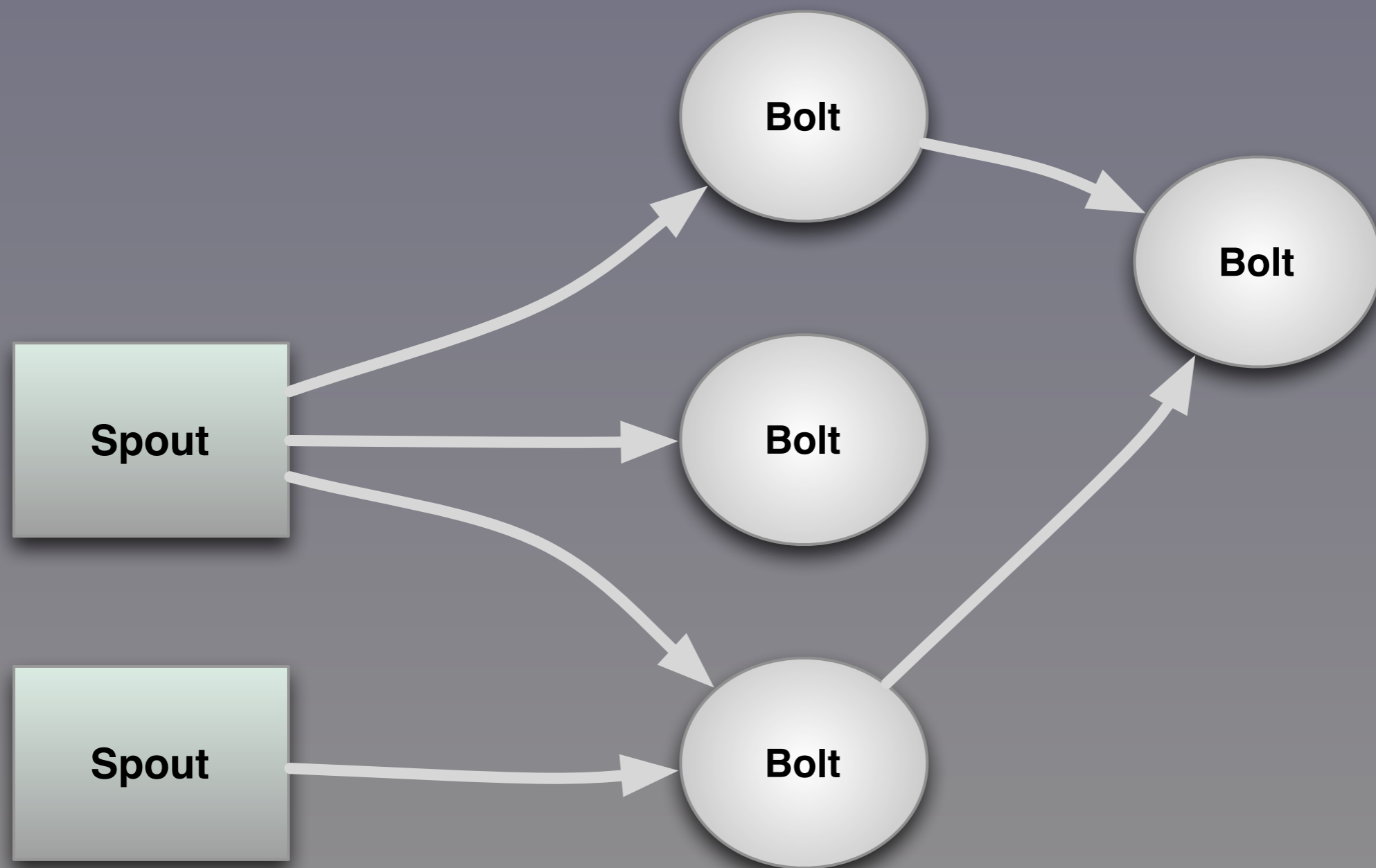
Monday, April 22, 13

Storm implements
reliable, distributed
“real-time”
event processing.

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

<http://storm-project.net/> Created by Nathan Marz, now at Twitter, who also created Cascalog, the Clojure wrapper around Cascading with added Datalog (logic programming) features.



In Storm terminology, Spouts are data sources and bolts are the event processors. There are facilities to support reliable message handling, various sources encapsulated in Sprouts and various targets of output. Distributed processing is baked in from the start.

Databases?



Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

SQL or NoSQL Databases?

- Since databases are designed for fast, transactional updates, consider a database for event processing.

#4

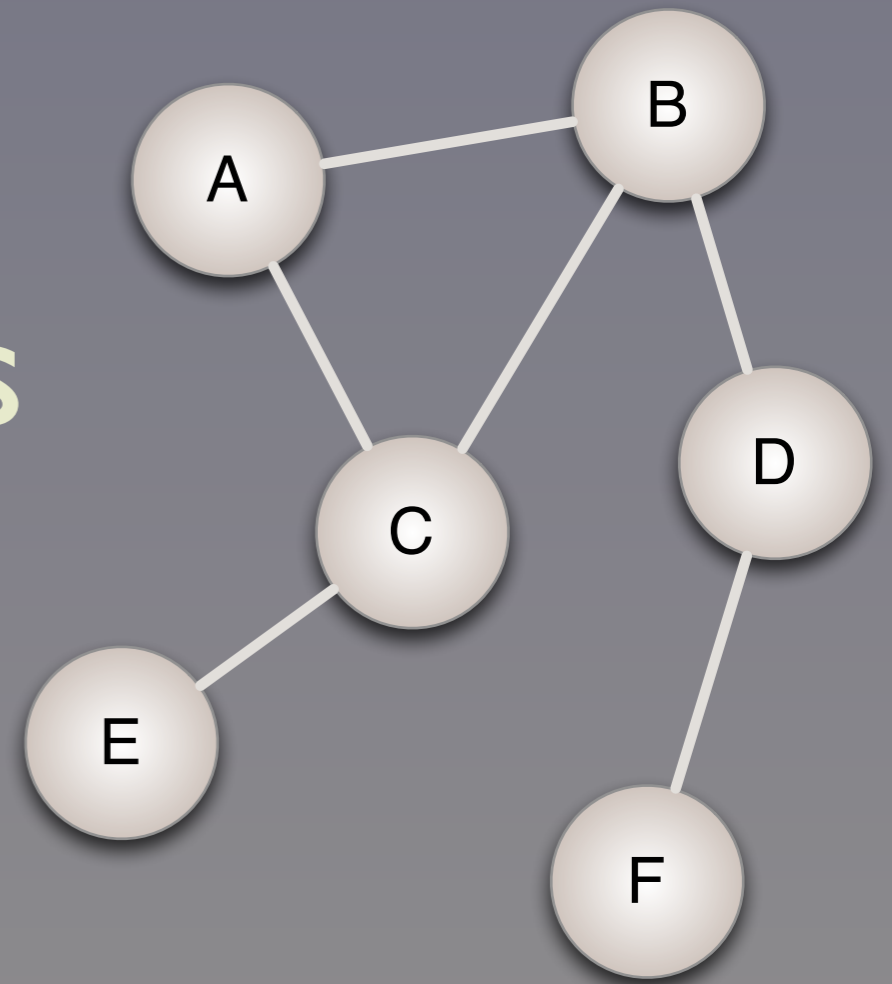
It's not ideal for
graph processing.

Google's Page Rank

- Google invented MapReduce,
- ... but MapReduce is not ideal for Page Rank and other graph algorithms.

Why not MapReduce?

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



Use Graph Processing (Solution #4)

Google's Pregel

- Pregel: New graph framework for Page Rank.
- 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.

All are
somewhat
immature.

A Manifesto...



72

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

To bring this altogether, I think we have opportunities for a better way...



Hadoop is the *Enterprise Java Beans* of our time.

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

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’s a 1st generation solution that mostly works okay and people do get work done with it, 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, specifically around Hadoop. The functional programming community, is well positioned to create it...

Stop using Java!



Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

Java has taken us a long way and the JVM remains one of our most valuable tools. But the language is really wrong language for data purposes and its continued use by Big Data vendors is slowing down overall progress, as well as application developer productivity, IMHO. Java emphasizes the wrong abstractions, objects instead of mathematically-inspired functional programming constructs, and Java encourages inflexible bloat because it's verbose compared to more modern alternatives and objects (at least class-based ones...) are far less reusable and flexible than people realize.



Functional Languages improve Big Data productivity!

75

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

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, Clojure, Erlang, F#, OCaml, and Haskell are more concise and better at eliminating boilerplate, while still providing excellent performance.

Note that one reason SQL has succeeded all these years is because it is also inspired by math, e.g., set theory.



Functional Collections.

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

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 optimize implementations for parallelization. The Scala collections offer parallelization with a tiny API call. Spark and Cascading transparently distribute collections across a cluster.

Erlang, Akka: Actor-based, Distributed Computation

Fine Grain Compute Models.

Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Monday, April 22, 13

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! For example, Erlang is a very mature language with the Actor model backed in. Akka is a Scala distributed computing model based on the Actor model of concurrency. It exposes clean, low-level primitives for robust, distributed services (e.g., Actors), upon which we can build flexible big data systems that can handle soft real time and batch processing efficiently and with great scalability.

Final Thought:



Luke Wroblewski

@lukew



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

12 Nov 12



ights Reserved

Monday, April 22, 13
A final thought about Big Data...

Questions?

TechMesh London 2012
December 5, 2012

[@deanwampler](#)

dean@deanwampler.com

79

Copyright © 2011-2013, Dean Wampler,



Monday, April 22, 13

All pictures Copyright © Dean Wampler, 2011-2013, All Rights Reserved. All other content is free to use, but attribution is requested.