Wednesday, December 4, 13

Photo: John Hancock Center, Michigan Ave. Chicago, Illinois USA

Wednesday, December 4, 13

Typesafe builds tools for creating Reactive Applications, http://typesafe.com/platform. See also the Reactive Manifesto, http://www.reactivemanifesto.org/

Photo: The Chicago River

Wednesday, December 4, 13

I've been doing Scala for 6 years and Big Data for 3.5 years.

Wednesday, December 4, 13

My books…

What Is Big Data?

DevOps Borat @DEVOPS_BORAT 8 Jan

Big Data is any thing which is crash Excel.

Expand

DevOps Borat @DEVOPS_BORAT 6 Feb

Small Data is when is fit in RAM. Big Data is when is crash because is not fit in RAM.

Expand

Big Data

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

Hat tip: Bob Korbus

Wednesday, December 4, 13

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, December 4, 13

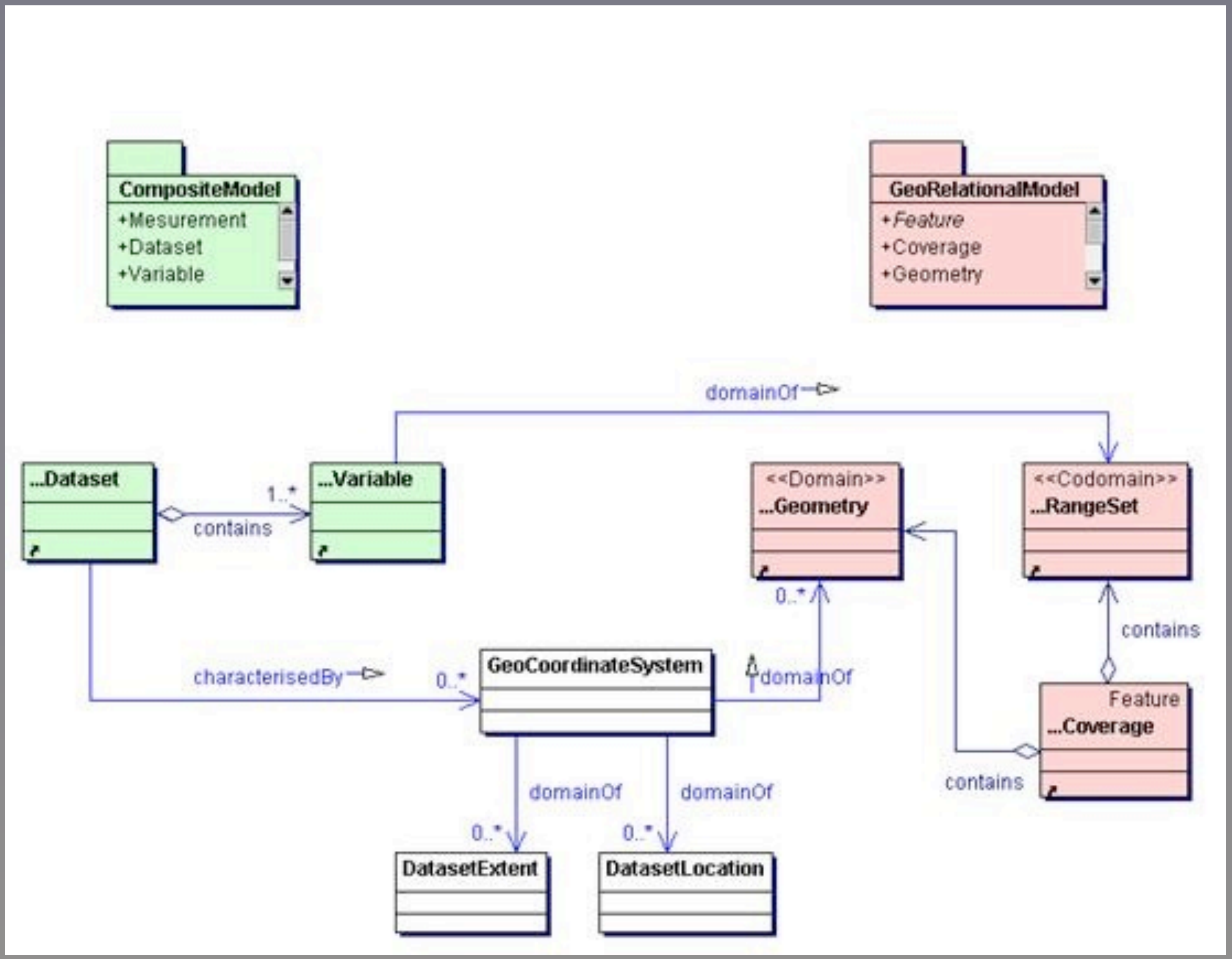
Three prevailing trends driving data-centric computing.
Photo: Prizker Pavilion, Millenium Park, Chicago (designed by Frank Gehry)

Data Size ⬆

Wednesday, December 4, 13

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

Formal Schemas ⬇



Wednesday, December 4, 13

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 ⬆

Wednesday, December 4, 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...)

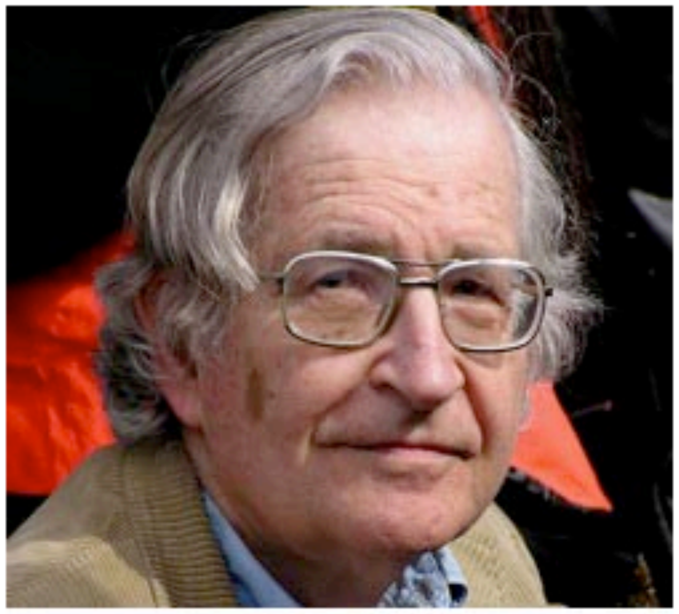
Probabilistic Models vs. Formal Grammars

[tor.com/blogs/...](http://www.tor.com/blogs/2011/06/norvig-vs-chomsky-and-the-fight-for-the-future-of-ai)

Norvig vs. Chomsky and the Fight for the Future of AI

KEVIN GOLD

When the Director of Research for Google compares one of the most highly regarded linguists of all time to Bill O'Reilly, you know it is *on*. Recently, Peter Norvig, Google's Director of Research and co-author of the most popular artificial intelligence textbook in the world, wrote a webpage extensively criticizing Noam Chomsky, arguably the most influential linguist in the world. Their disagreement points to a revolution in artificial intelligence that, like many revolutions, threatens to destroy as much as it improves. Chomsky, one of the old guard, wishes for an elegant theory of intelligence and language that looks past human fallibility to try to see simple structure underneath. Norvig, meanwhile, represents the new philosophy: truth by statistics,



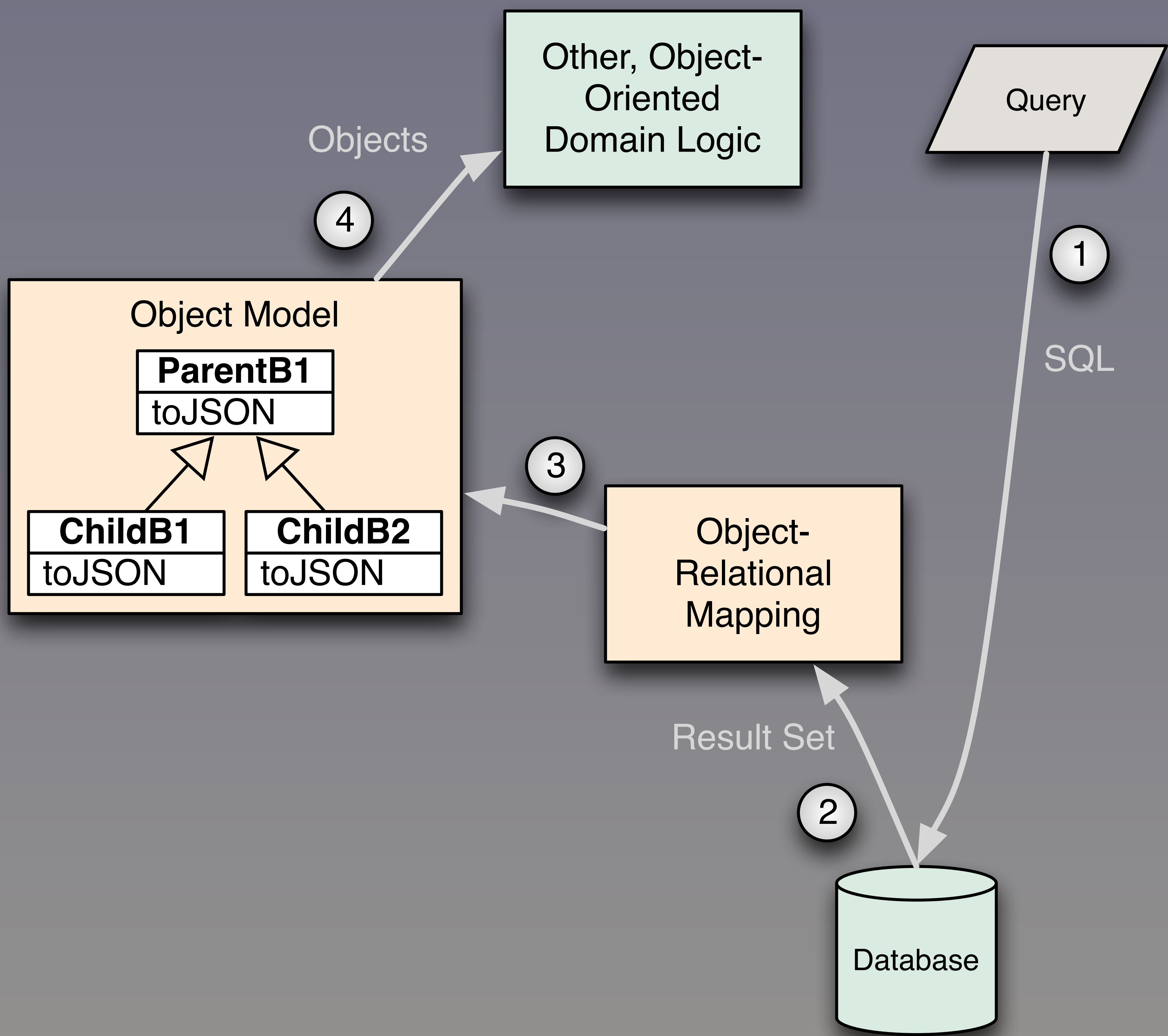
Chomsky photo by Duncan Rawlinson and his Online Photography School. Norvig photo by Peter Norvig

Wednesday, December 4, 13

An interesting manifestation of this trend is the public argument between Noam Chomsky and Peter Norvig on the nature of language. Chomsky long ago proposed a hierarchical model of formal language grammars. Peter Norvig is a proponent of probabilistic models of language. Indeed all successful automated language processing systems are probabilistic.
http://www.tor.com/blogs/2011/06/norvig-vs-chomsky-and-the-fight-for-the-future-of-ai

Wednesday, December 4, 13

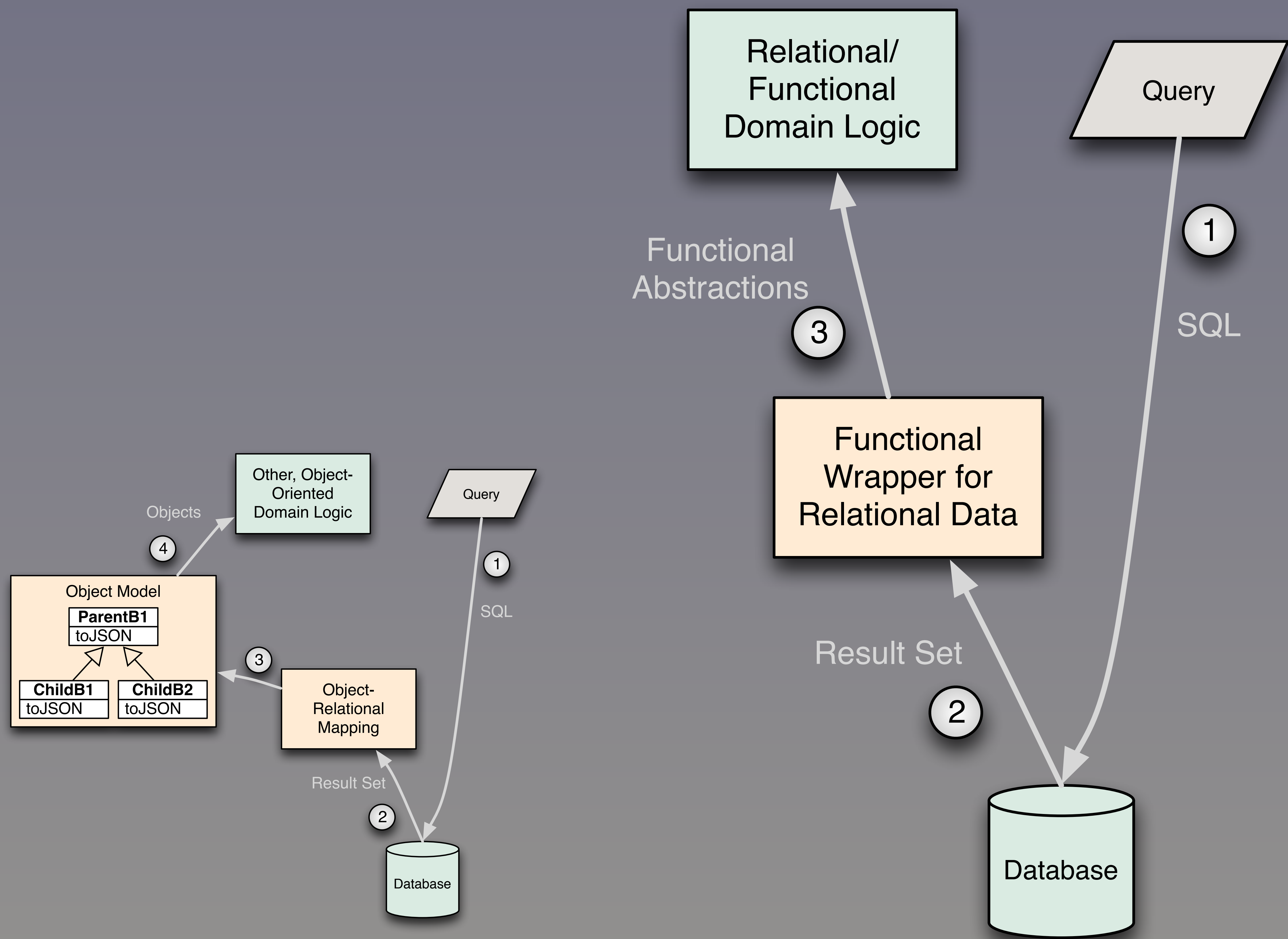
What should software architectures look like for these kinds of systems?
Photo: Cloud Gate (a.k.a. “The Bean”) in Millenium Park, Chicago, on a cloudy day.



Wednesday, December 4, 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…)

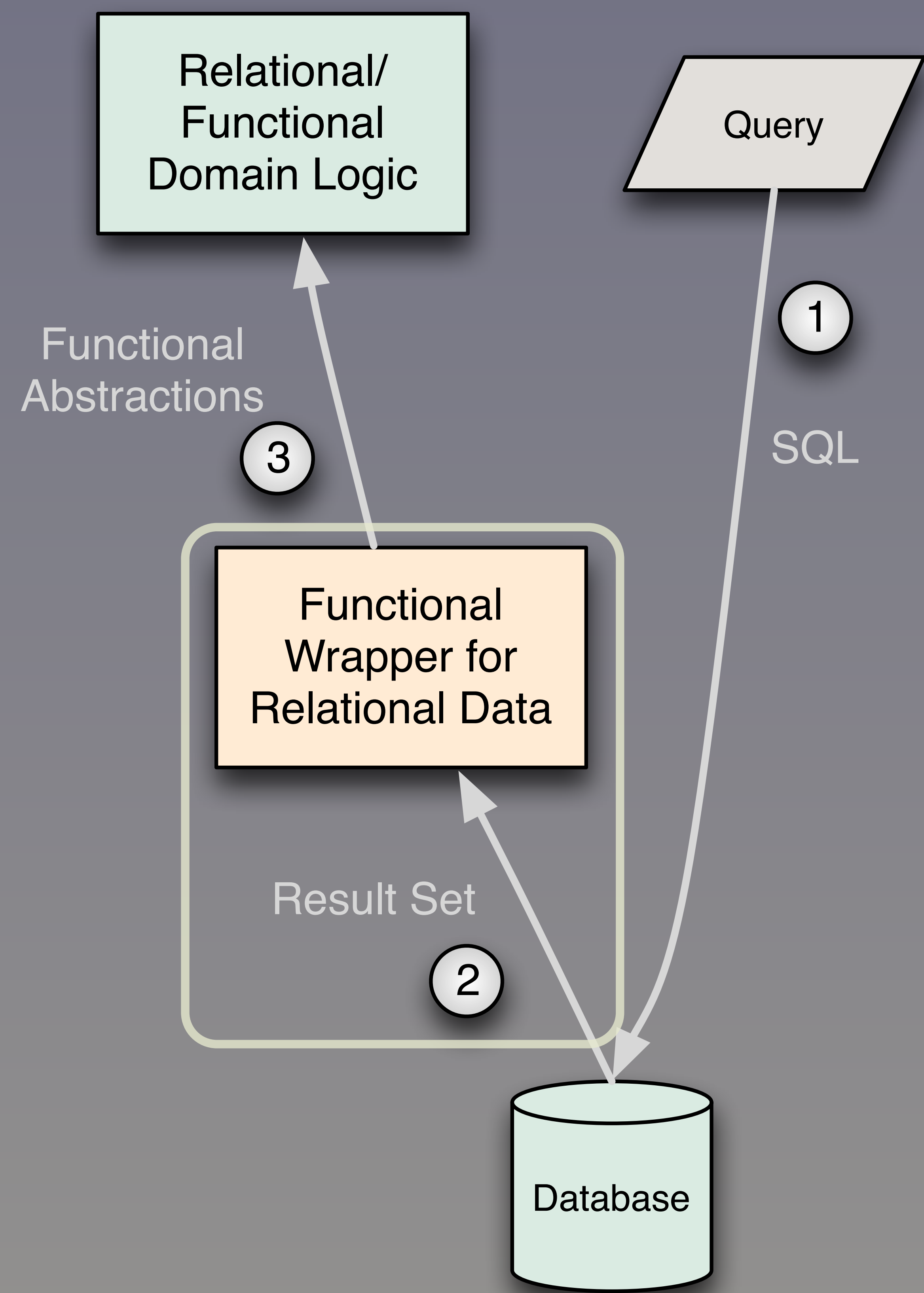
The toJSON methods are there because we often convert these object graphs back into fundamental structures, such as the maps and arrays of JSON so we can send them to the browser!



Wednesday, December 4, 13

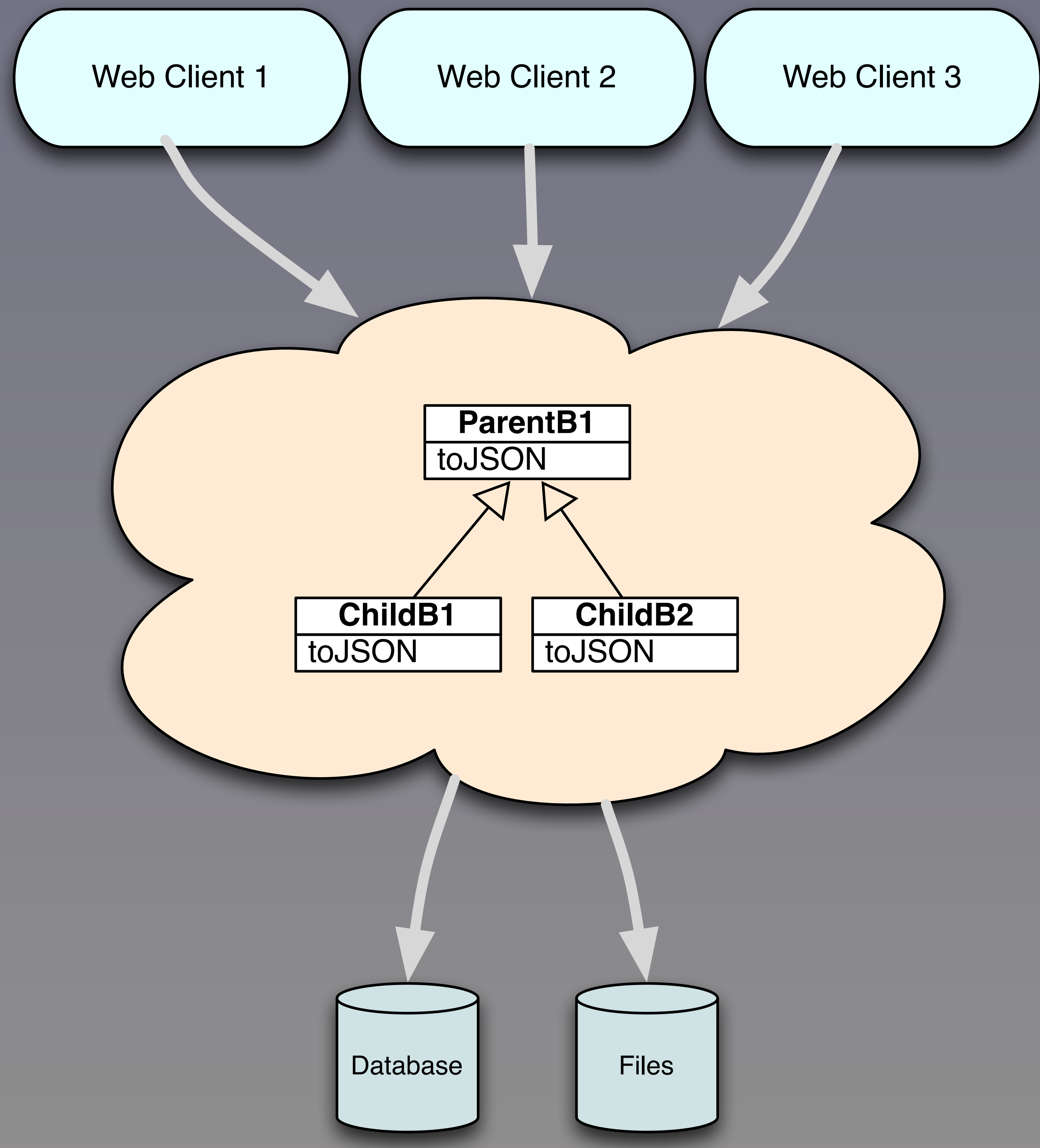
But the traditional systems are a poor fit for this new world: 1) they add too much overhead in computation (the ORM layer, etc.) and memory (to store the objects). Most of what we do with data is mathematical transformation, so we're far more productive (and runtime efficient) if we embrace fundamental 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
  - ...



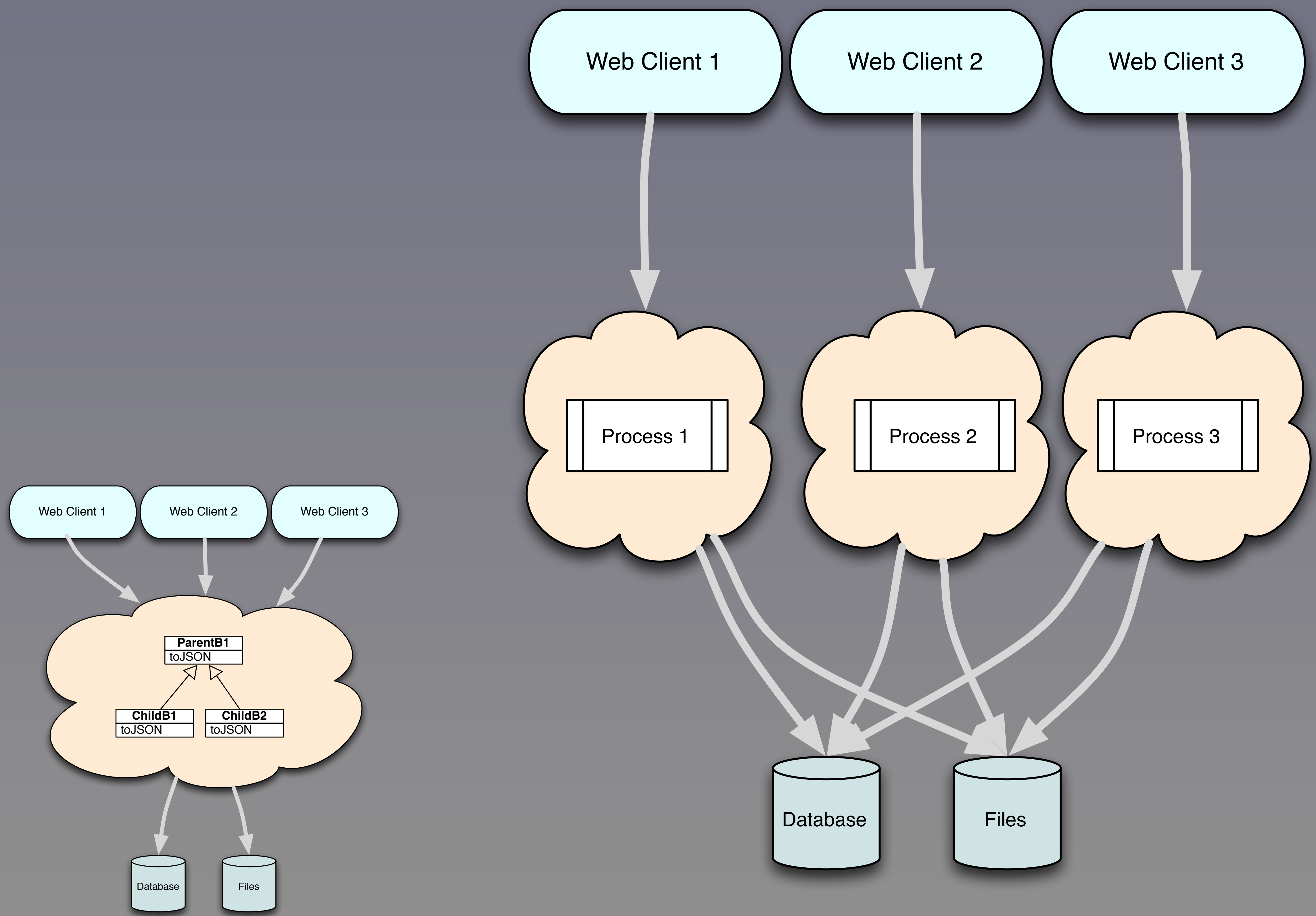
Wednesday, December 4, 13

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.



Wednesday, December 4, 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.

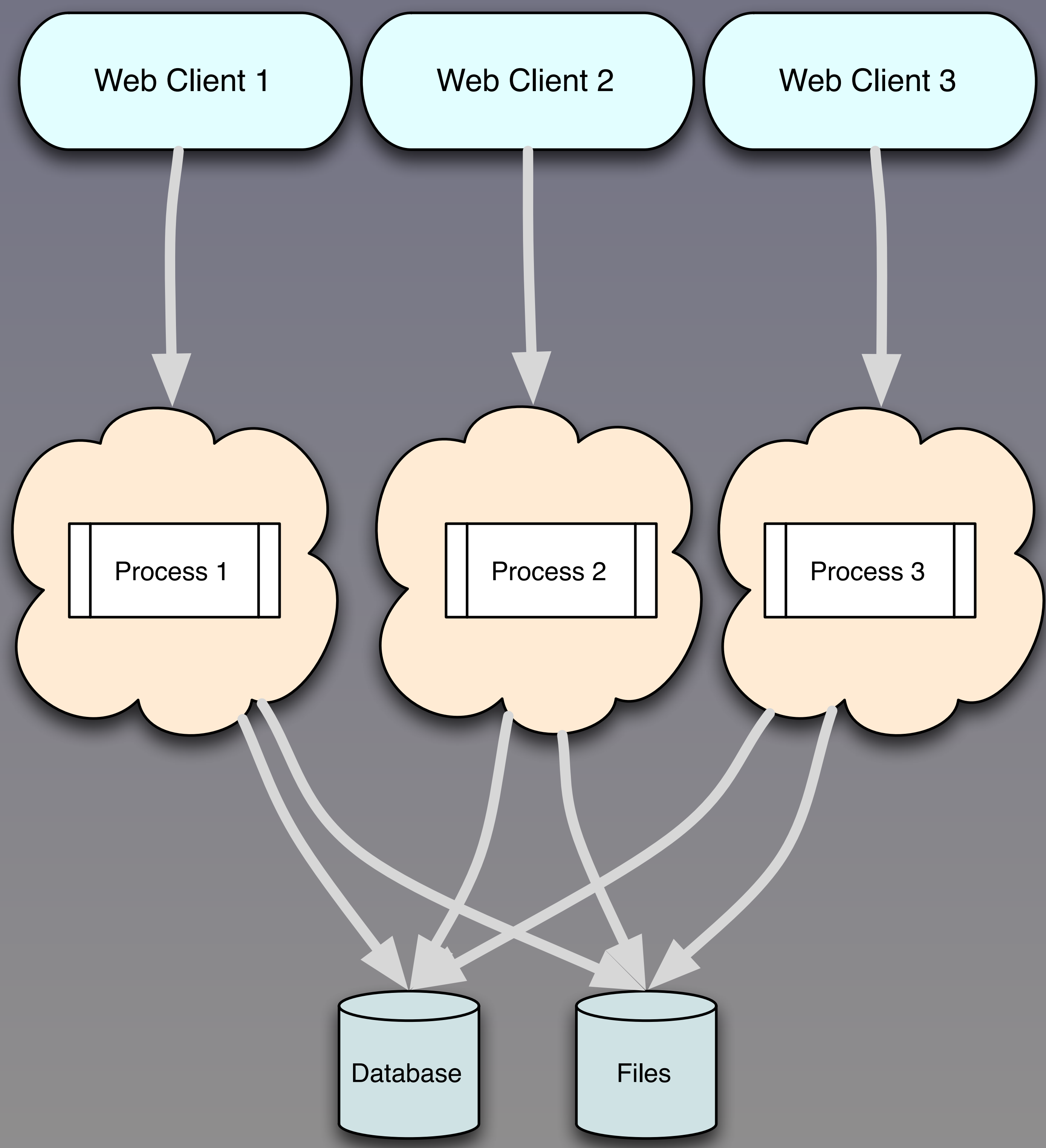


Wednesday, December 4, 13

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.

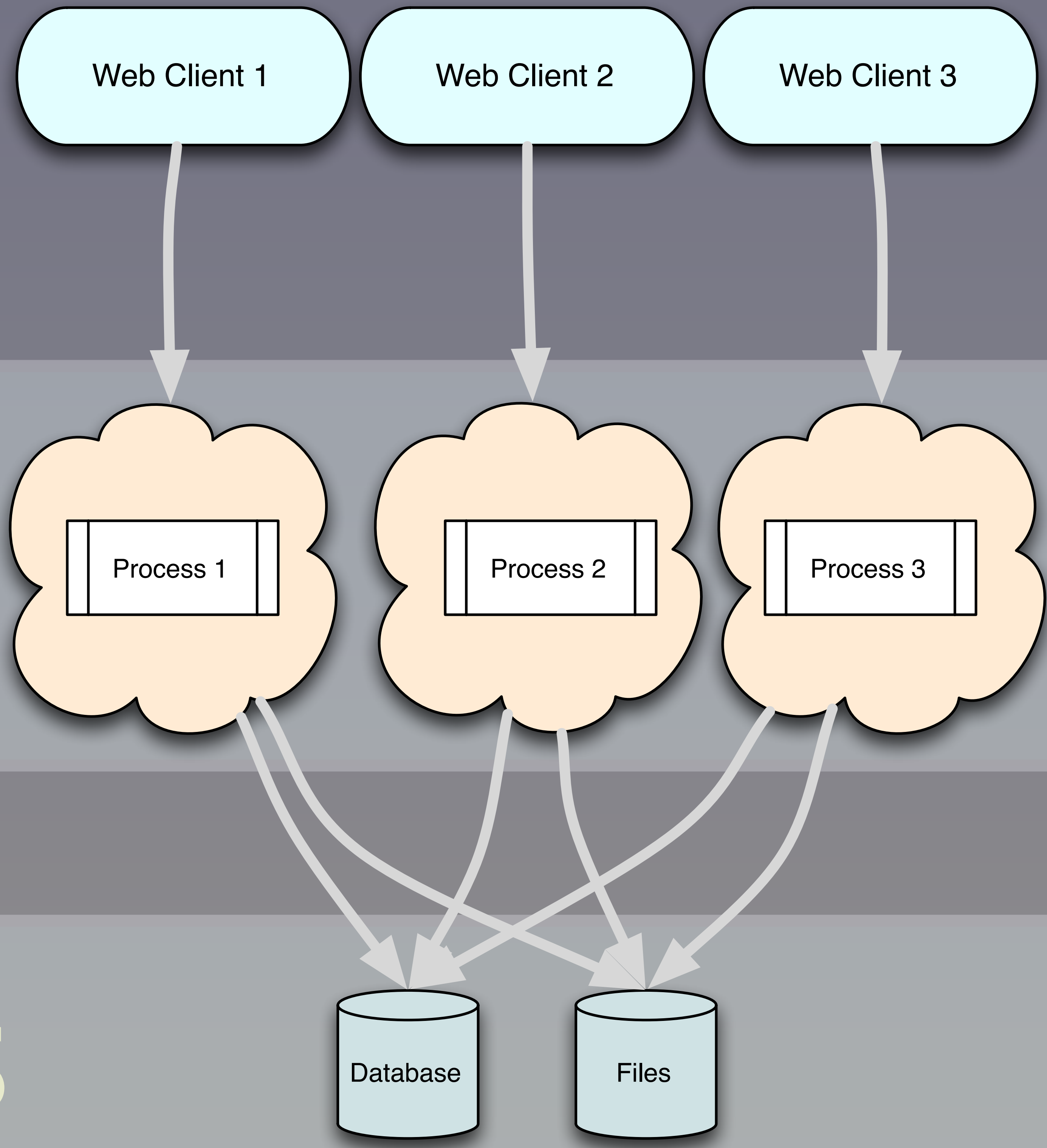
- Data Size ⬆
- Formal Schema ⬇
- Data-Driven Programs ⬆



Wednesday, December 4, 13

And this structure better fits the trends I outlined at the beginning of the talk.

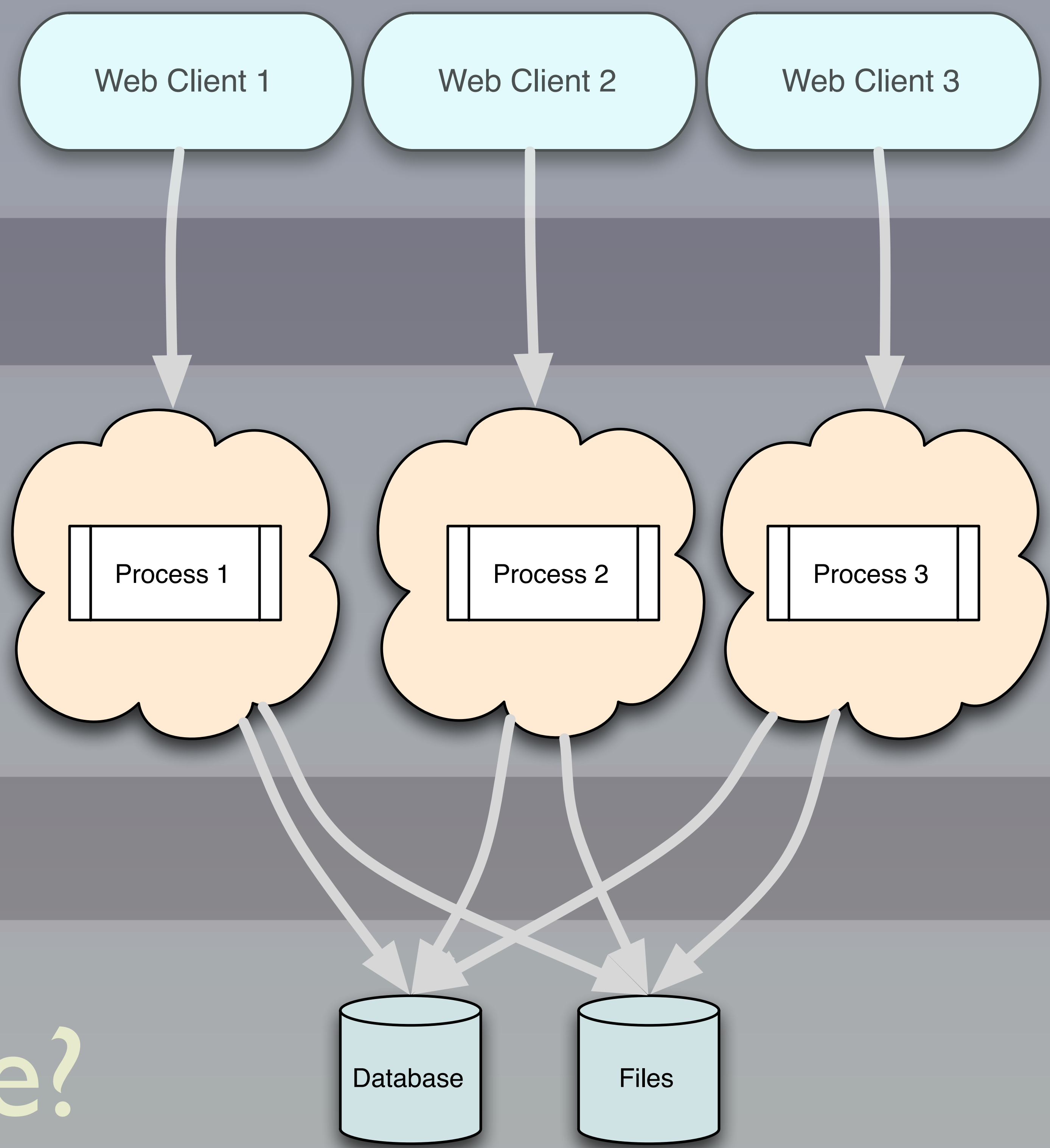
- MapReduce
- Distributed FS



Wednesday, December 4, 13

And MapReduce + a distributed file system, like Hadoop's MapReduce and HDFS, fit this model.

- JSON
- Node.js?
- JSON database?



Wednesday, December 4, 13

One interesting incarnation of this is JavaScript through the full stack, with JSON as the RPC format, stored directly (more or less) in a database like Mongo, CouchBase, and RethinkDB. Node gives you JS in the mid-tier, and JSON is obviously a browser tool.

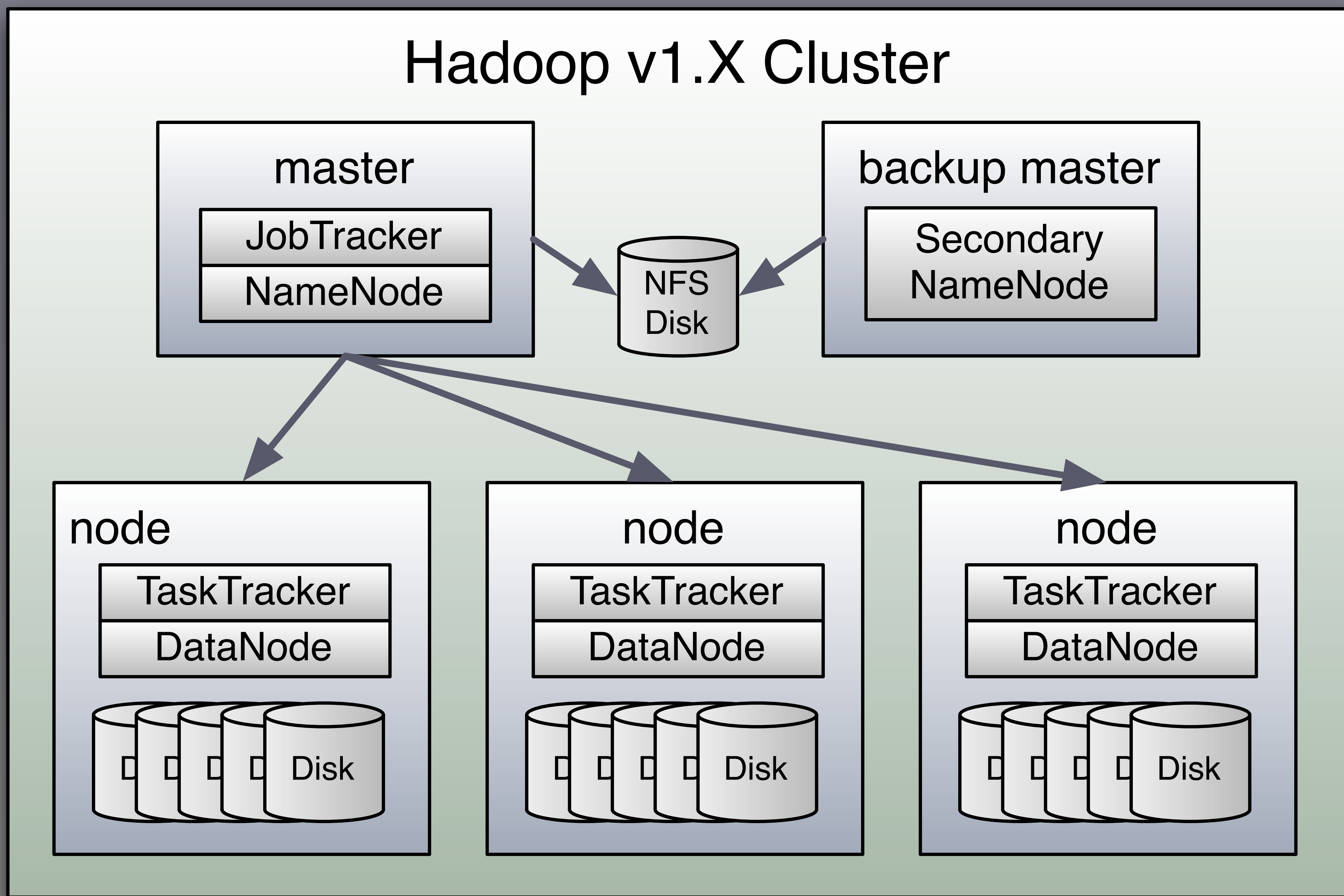
What Is MapReduce?

Wednesday, December 4, 13

Cloud Gate – “The Bean” – in Millenium Park, Chicago, on a sunny day – with some of my relatives ;)

Hadoop is the dominant Big Data platform today.

A Hadoop Cluster



Wednesday, December 4, 13

A Hadoop v1.X cluster. (V2.X introduces changes in the master processes, including support for high-availability and federation...). In brief:
JobTracker (JT): Master of submitted MapReduce jobs. Decomposes job into tasks (each a JVM process), often run where the "blocks" of input files are located, to minimize net IO.
NameNode (NN): HDFS (Hadoop Distributed File System) master. Knows all the metadata, like block locations. Writes updates to a shared NFS disk (in V1) for use by the Secondary NameNode.
Secondary NameNode (SNN): periodically merges in-memory HDFS metadata with update log on NFS disk to form new metadata image used when booting the NN and SNN.
TaskTracker: manages each task given to it by the JT.
DataNode: manages the actual blocks it has on the node.
Disks: By default, Hadoop just works with "a bunch of disks" - cheaper and sometimes faster than RAID. Blocks are replicated 3x (default) so most HW failures don't result in data loss.

MapReduce in Hadoop

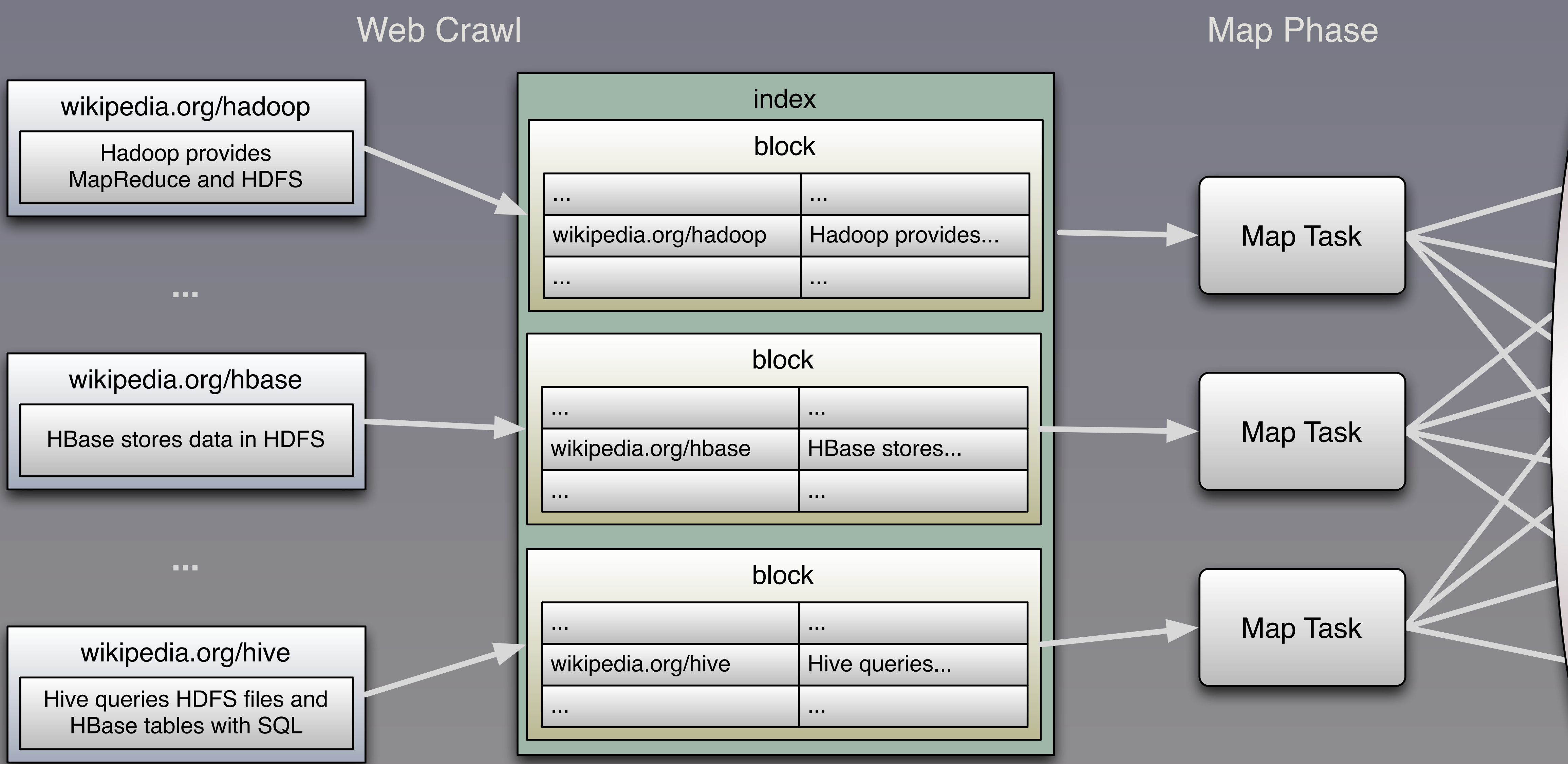
Let's look at a MapReduce algorithm: Inverted Index.

Used for text/web search.

Wednesday, December 4, 13

Let's walk through a simple version of computing an inverted index. Imagine a web crawler has found all docs on the web and stored their URLs and contents in HDFS. Now we'll index it; build a map from each word to all the docs where it's found, ordered by term frequency within the docs.

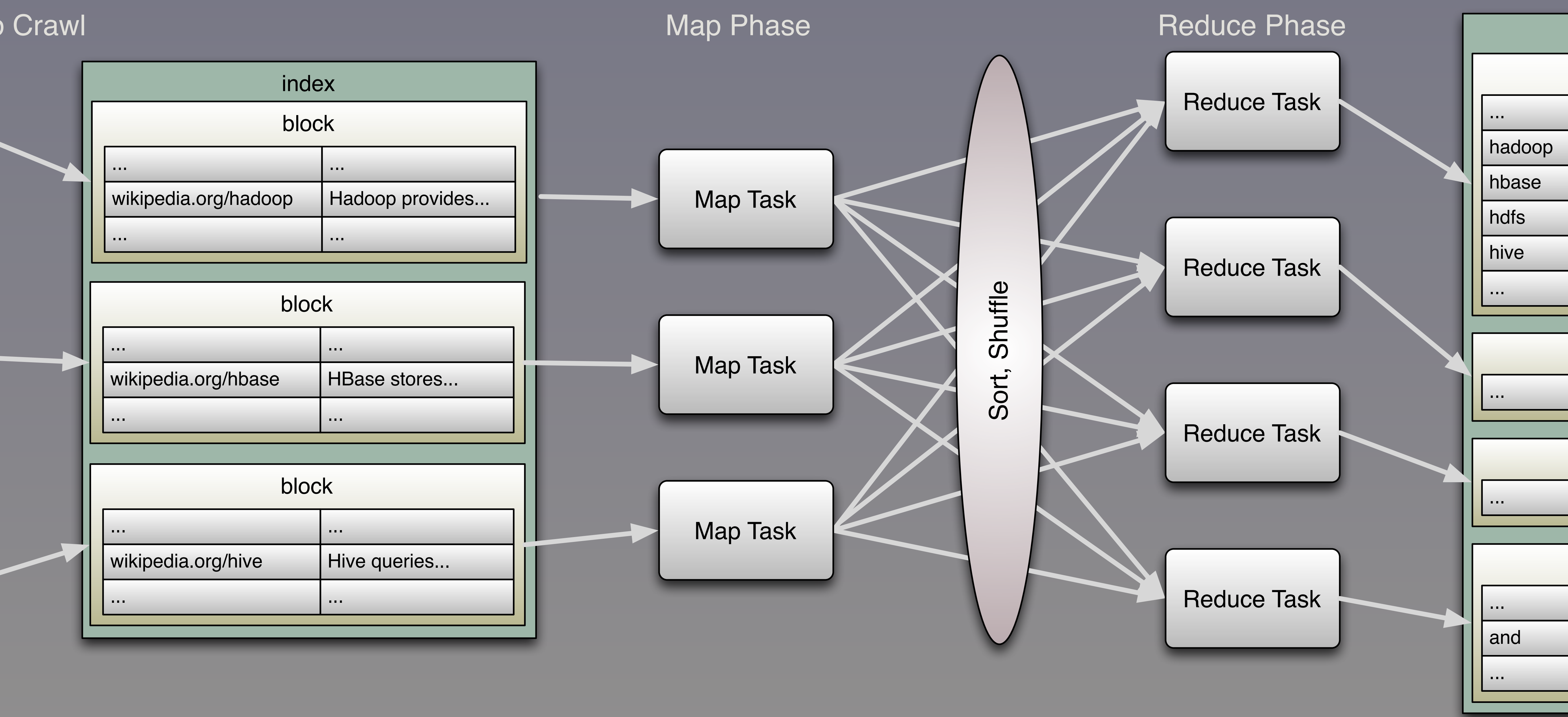
Crawl teh Interwebs



Wednesday, December 4, 13

Crawl pages, including Wikipedia. Use the URL as the document id in our first index, and the contents of each document (web page) as the second "column". in our data set.

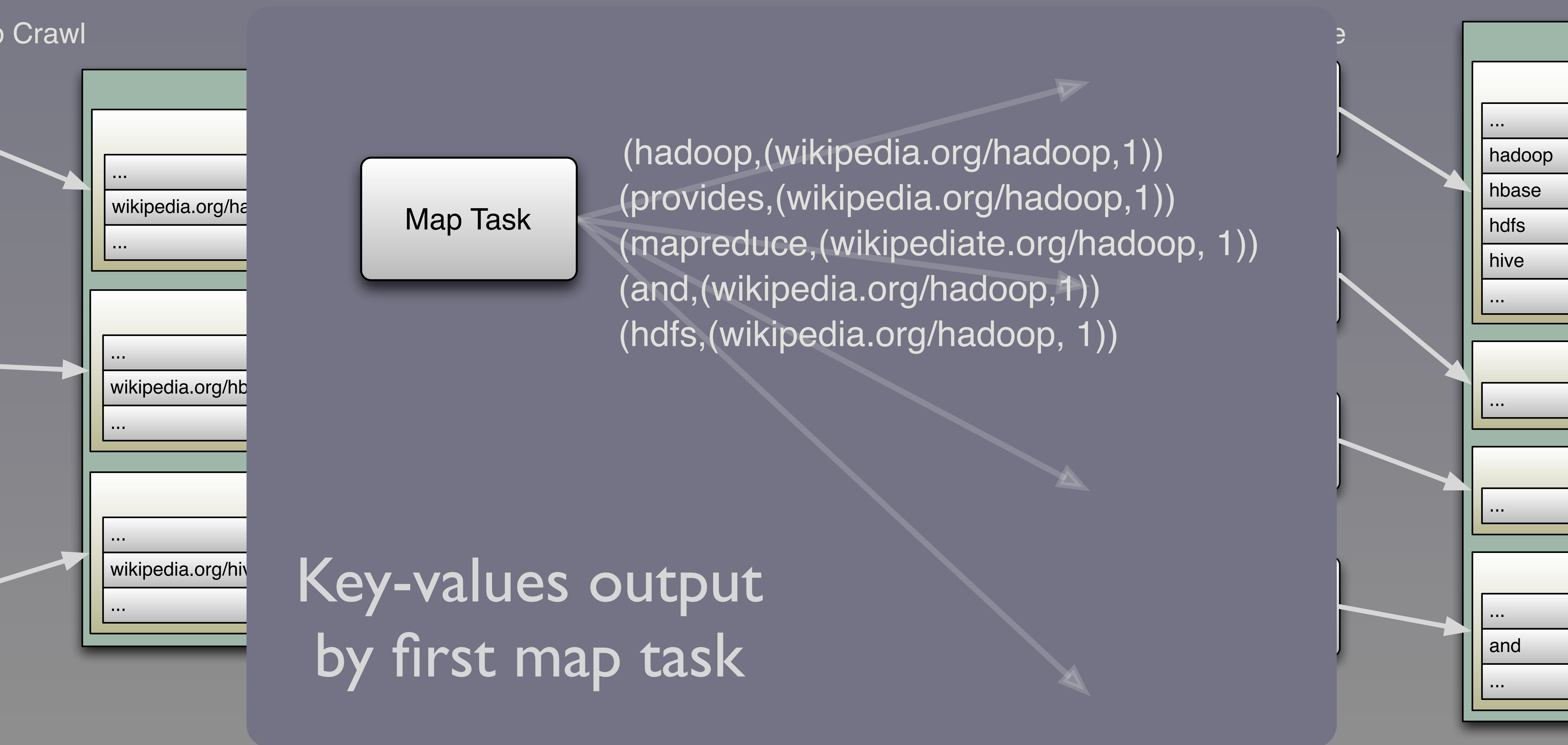
Compute Inverse Index



Wednesday, December 4, 13

Now run a MapReduce job, where a separate Map task for each input block will be started. Each map tokenizes the content in to words, counts the words, and outputs key-value pairs...

Compute Inverse Index

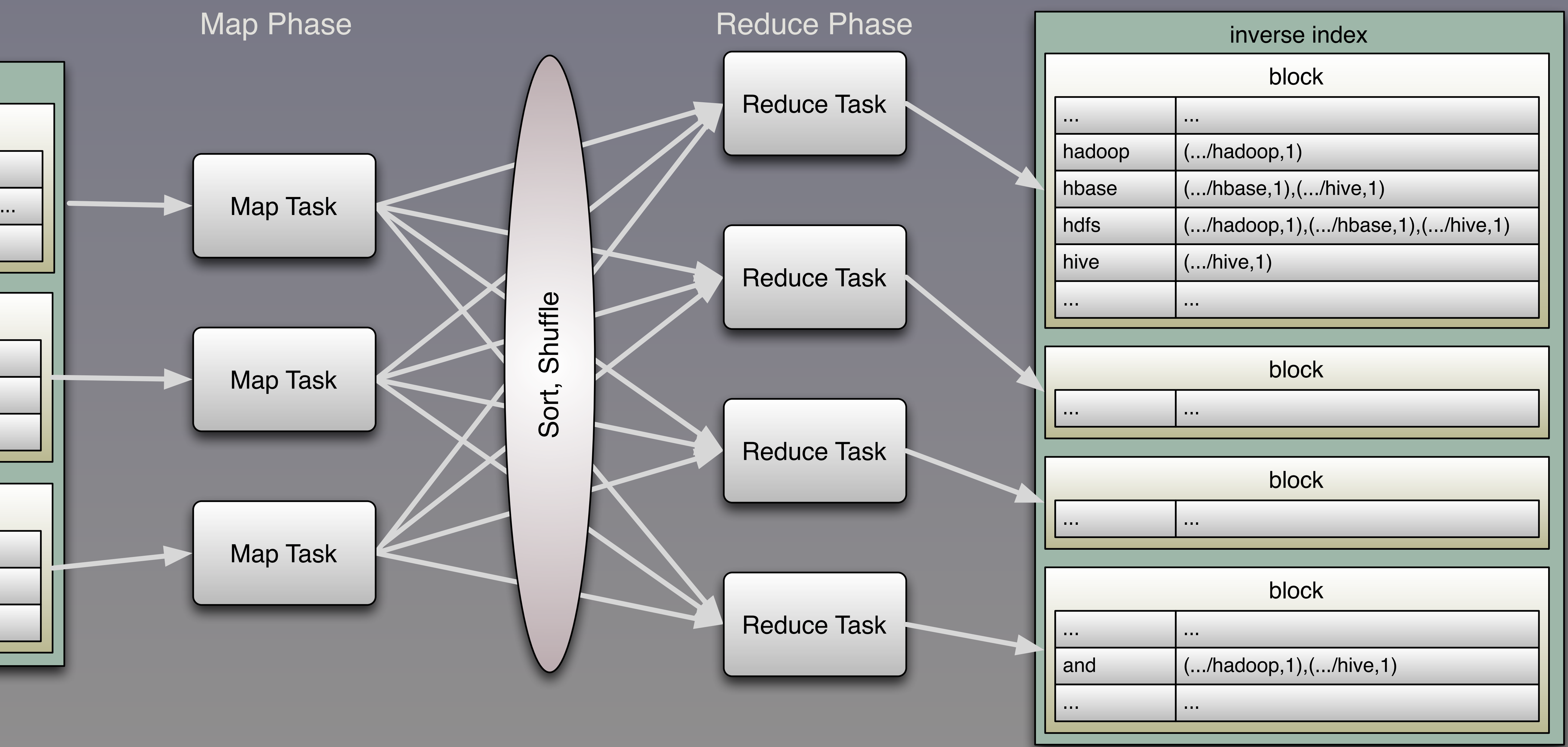


Wednesday, December 4, 13

Now run a MapReduce job, where a separate Map task for each input block will be started. Each map tokenizes the content in to words, counts the words, and outputs key-value pairs...

… Each key is a word that was found and the corresponding value is a tuple of the URL (or other document id) and the count of the words (or alternatively, the frequency within the document). Shown are what the first map task would output (plus other k-v pairs) for the (fake) Wikipedia “Hadoop” page. (Note that we convert to lower case…)

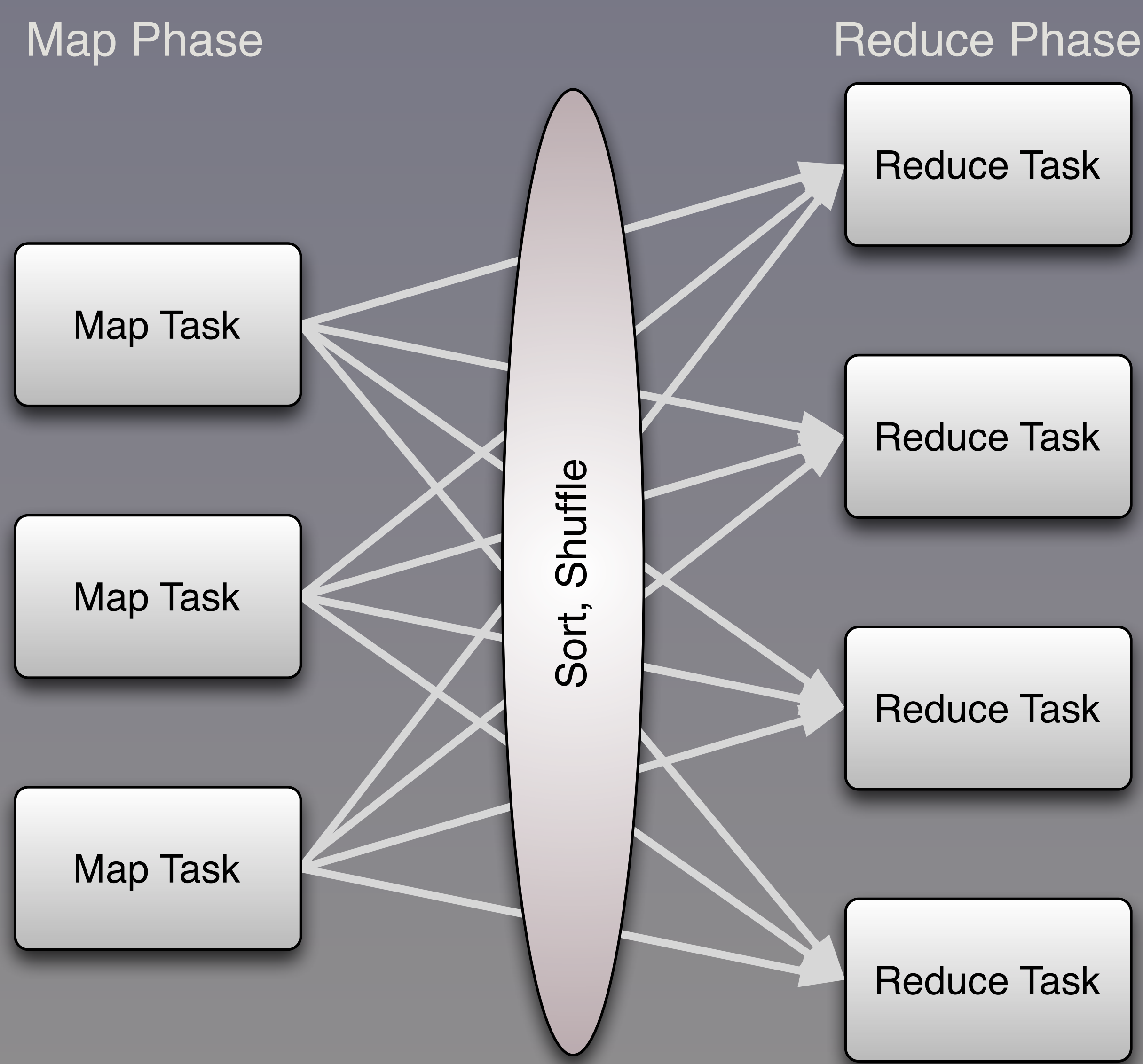
Compute Inverse Index



Wednesday, December 4, 13

Finally, each reducer will get some range of the keys. There are ways to control this, but we'll just assume that the first reducer got all keys starting with "h" and the last reducer got all the "and" keys. The reducer outputs each word as a key and a list of tuples consisting of the URLs (or doc ids) and the frequency/count of the word in that document, sorted by most frequent first. (All our docs have only one occurrence of any word, so the sort is moot...)

Anatomy: MapReduce Job



Map (or *Flatmap*):

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

Reduce:

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

Wednesday, December 4, 13

To recap, a true functional/mathematical "map" transforms one input to one output, but this is generalized in MapReduce to be one to 0-N. In other words, it should be "FlatmapReduce"!! The output key-value pairs are distributed to reducers. The "reduce" collects together multiple inputs with the same key into

Andrew Whang
@whangsf

Follow

MapReduce without the Reducer
pic.twitter.com/5lQSFYmhAT

Reply · Retweeted · Favorite · More

34 RETWEETS · 16 FAVORITES

Wednesday, December 4, 13

Quiz. Do you understand this tweet?

So, MapReduce is a mashup of our friends flatmap and reduce.

Wednesday, December 4, 13

Even in this somewhat primitive and coarse-grain framework, our functional data concepts are evident!

Today,

Hadoop is our best,

general-purpose tool

for horizontal scaling

of Big Data,

but...

Wednesday, December 4, 13

Is MapReduce the end of the story? Does it meet all our needs? Let’s look at a few problems…
Photo: Gratuitous Romantic beach scene, Ohio St. Beach, Chicago, Feb. 2011.

MapReduce doesn't fit all computation needs. HDFS doesn't fit all storage needs.

Wednesday, December 4, 13

Let's frame our discussion of where Big Data is going by contrasting needs and options with the current standard, MapReduce and HDFS (Hadoop Distributed File System).

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 from the algorithm to the implementation is not trivial at all. In fact, implementing algorithms in MR is now a specialized body of knowledge.

MapReduce is very course-grained.

1-Map, 1-Reduce phase...

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.

Multiple MR jobs required for some algorithms.

Each one flushes its results to disk!

If you have to sequence MR jobs to implement an algorithm, ALL the data is flushed to disk between jobs. There's no in-memory caching of data, leading to huge IO overhead.

MapReduce is designed for offline, batch-mode analytics.

High latency; not suitable for event processing.

Alternatives are emerging to provide event-stream ("real-time") processing.

The Hadoop Java API is hard to use.

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

Let's look at code for a simpler algorithm, Word Count.

(Tokenize as before, but ignore original document locations.)

Wednesday, December 4, 13

In Word Count, the mapper just outputs the word-count pairs. We forget about the document URL/id. The reducer gets all word-count pairs for a word from all mappers and outputs each word with its final, global count.

```java
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));
  }
}
```


Wednesday, December 4, 13

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 dominated by green for types (classes), with relatively few yellow functions that implement actual operations (i.e., do actual work).

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!

```java
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));
  }
}
```

The interesting bits

Wednesday, December 4, 13

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 dominated by green for types (classes), with relatively few yellow functions that implement actual operations (i.e., do actual work).

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!

```
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));
  }
}
```

2000 called. It wants its EJBs back!

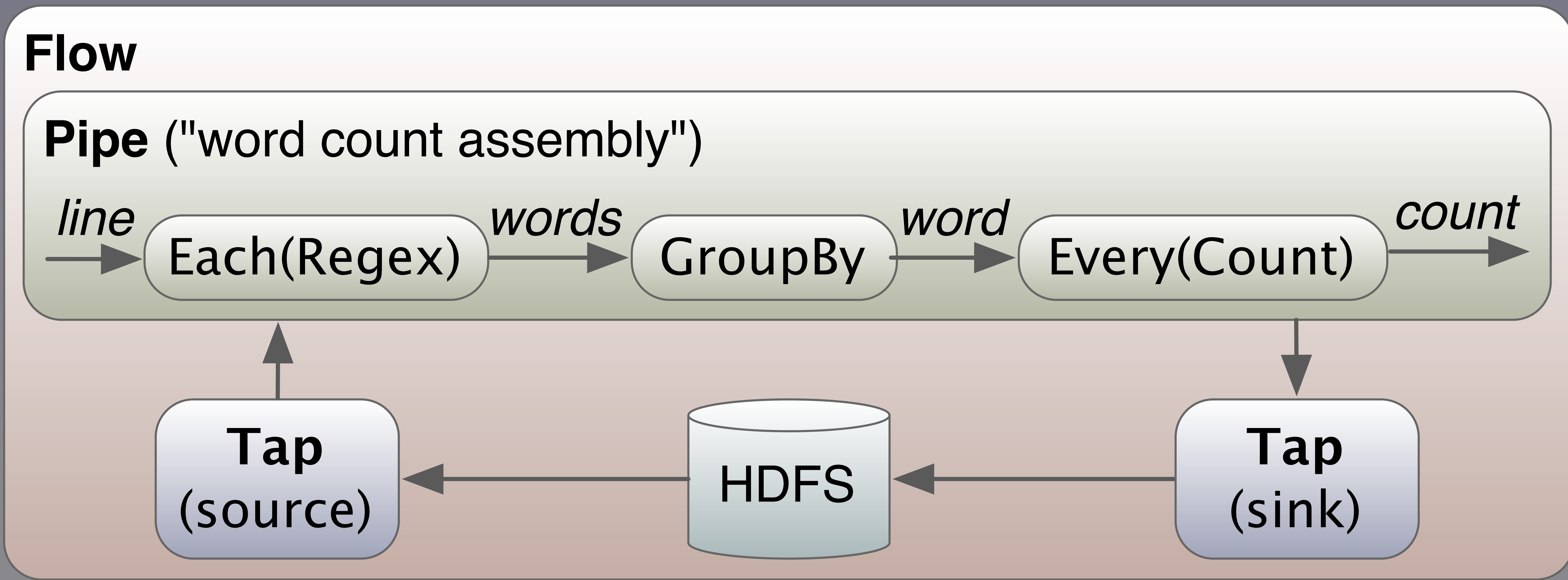
Wednesday, December 4, 13

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 dominated by green for types (classes), with relatively few yellow functions that implement actual operations (i.e., do actual work).
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!

Wednesday, December 4, 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.
Photo: Fermi Lab Office Building, Batavia, Illinois, USA (Fermi Lab is a large particle physics accelerator facility.)

Word Count: Cascading



Wednesday, December 4, 13

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

```java
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();
  }
}
```


Wednesday, December 4, 13

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.

Wednesday, December 4, 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.

```scala
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!!

Wednesday, December 4, 13

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.

Wednesday, December 4, 13

http://nathanmarz.com/blog/introducing-cascalog-a-clojure-based-query-language-for-hado.html
Clojure is a new JVM, lisp-based language with lots of important concepts, such as persistent datastructures.

```
(defn lowercase [w] (.toLowerCase w))

(?<- (stdout) [?word ?count]
 (sentence ?s)
   (split ?s :> ?word1)
 (lowercase ?word1 :> ?word)
   (c/count ?count))
```

Datalog-style queries

Wednesday, December 4, 13

Cascalog embeds Datalog-style logic queries. The variables to match are named ?foo.

Wednesday, December 4, 13

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

Why isn't it more widely used? 1) lack of commercial support, 2) only recently emerged out of academia.

```scala
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 small and concise!

Wednesday, December 4, 13

This spark example is actually closer in a few details, i.e., function names used, to the original Hadoop Java API example, but it cuts down boilerplate to the bare minimum.

Spark is a Hadoop MapReduce alternative:

- Distributed computing with in-memory caching.
- ~10-100x faster than MapReduce (in part due to caching of intermediate data).

Spark also addresses the lack of flexibility for the MapReduce model.

Spark is a Hadoop MapReduce alternative:

- Originally designed for machine learning applications.
- Developed by Berkeley AMP.

Wednesday, December 4, 13

Using SQL when you can! Here are 5 (and growing!) options, some of which still use MapReduce, while others have introduced new, faster runtimes.

Here, we’re discussing SQL as a tool for computation and not discussing the storage aspect of SQL database systems.

Use SQL when you can!

- Hive: SQL on top of MapReduce.
- Shark: Hive ported to Spark.
- Impala & Presto: HiveQL with new, faster back ends.
- Lingual: ANSI SQL on Cascading.

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. http://www.facebook.com/notes/facebook-engineering/presto-interacting-with-petabytes-of-data-at-facebook/10151786197628920 for Presto. Impala & Presto are relatively new.

Word Count in Hive 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;
```

Works for Hive, Shark, and Impala

Wednesday, December 4, 13

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, etc., etc.

We're in the era I call
The SQL Strikes Back!

(with apologies to
George Lucas...)

IT shops realize that NoSQL is useful and all, but people really, Really, REALLY love SQL. So, it's making a big comeback. You can see it in Hadoop, in SQL-like APIs for some "NoSQL" DBs, e.g., Cassandra and MongoDB's Javascript-based query language, as well as "NewSQL" DBs.

Hadoop owes a lot of its popularity to Hive!

Wednesday, December 4, 13

In large companies, the data analysts outnumber the developers by a large margin. Almost all of them know SQL (even if they happen to use SAS or similar tools more often...).

Some “NoSQL” databases have or are adding query languages (e.g., Cassandra, MongoDB).

Wednesday, December 4, 13

Cassandra’s is relatively new and based on SQL. MongoDB has always had one, based on a Javascript DSL.

“NewSQL” databases are bringing NoSQL performance to the relational model.

Examples

- Google Spanner and F1.
- NuoDB.
- VoltDB.

Spanner is the successor to BigTable. It is a globally-distributed database (consistency is maintained using the Paxos algo. and hardware synchronized clocks through GPS and atomic clocks!) Each table requires a primary key. F1 is an RDBMS built on top of it.
NuoDB is a cloud based RDBMS.
VoltDB is an example "in-memory" database, which are ideal for lots of small transactions that leverage indexing and rarely require full table scans.

(So, take what I said earlier:

Formal Schemas ⬇

with a grain of salt...)

MapReduce is not suitable for event processing (“real-time”).

Wednesday, December 4, 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.

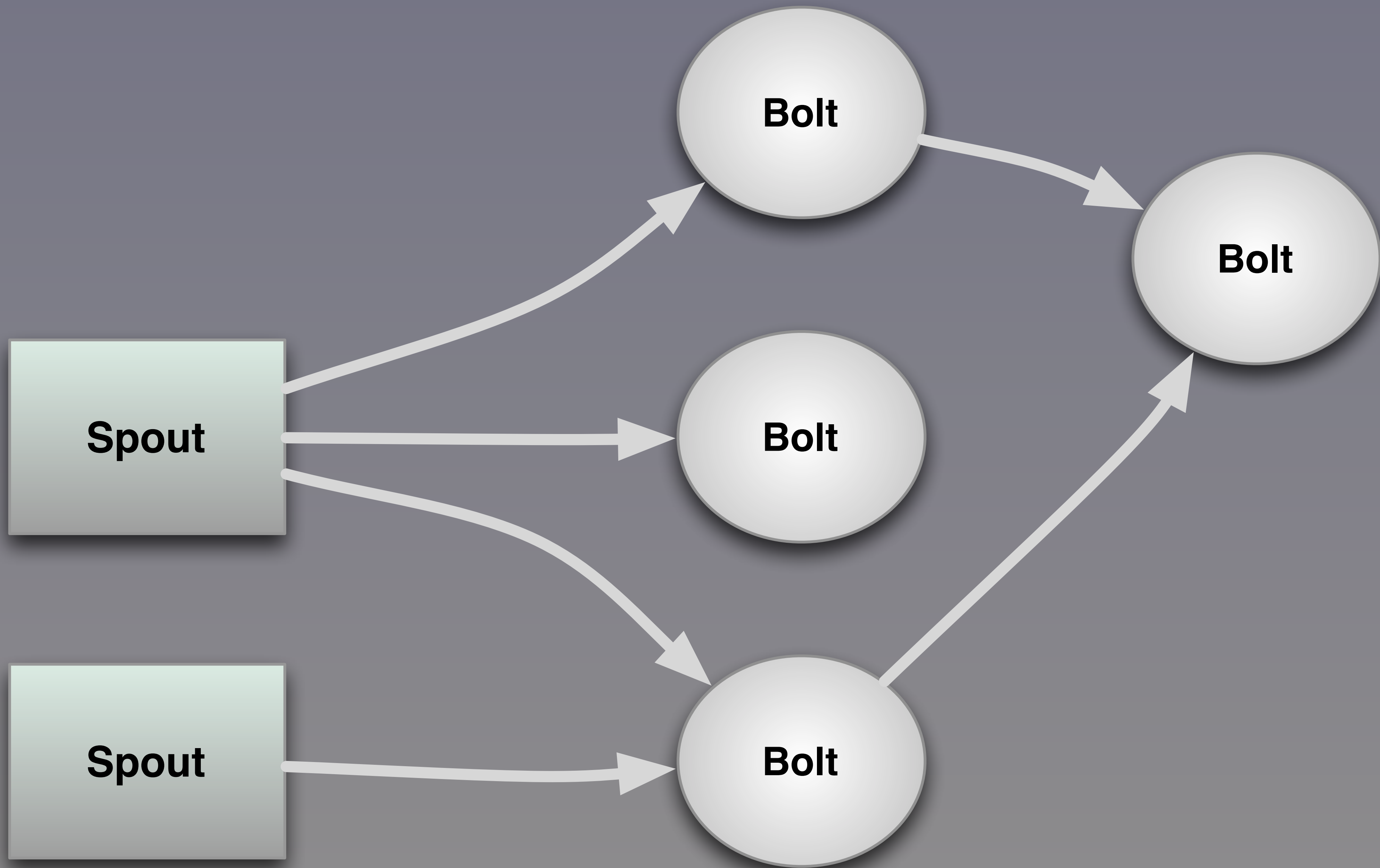
Wednesday, December 4, 13

Photo: Top of the AON Building, Chicago, after a Storm passed through.

Storm implements reliable, distributed event processing.

Wednesday, December 4, 13

http://storm-project.net/ Created by Nathan Marz, now at Twitter, who also created Cascalog.



Wednesday, December 4, 13

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.

Spark and Message Queues are also being used for distributed event processing.

http://storm-project.net/ Created by Nathan Marz, now at Twitter, who also created Cascalog.

Wednesday, December 4, 13

Databases as a real-time event processing option?

Photo: Outside my condo window, Chicago!

SQL or NoSQL Databases?

Capture events in a database with fast writes.

Wednesday, December 4, 13

Use a SQL database unless you need the scale and looser schema of a NoSQL database!

HDFS?

HDFS is oriented towards batch-mode reads and writes.

So, it's not suitable for incremental updates, like capturing events.

Databases are great for capturing individual records, especially in append-only scenarios.
So, storage is another important aspect of event processing.

Hadoop vs. SQL?

- Hadoop
- Very flexible compute model
- “Table” scans
- Batch mode

- NoSQL / SQL
- Focused on a data model
- Transactional
- Event driven

Wednesday, December 4, 13

What else can we say about Hadoop vs. SQL?

Problem:

Your current data warehouse can only store 6-months of data without a $1M upgrade.

Very common scenario. Numbers roughly correspond to a situation faced by a client...

Traditional DW

- *Pros*
- Mature
- Rich SQL, analytics
- Mid-size data

- *Cons*
- Expensive - $/TB
- Scalability limits

Data warehouses tend to be more scalable and a little less expensive than OLTP systems, which is why they are used to "warehouse" transactional data and perform analytics. However, their $/TB is ~10x–100x the cost on Hadoop and Hadoop scales to larger data sets.

SQL is very important for data warehouse applications, but transactions aren't.

NoSQL does give you the more cost-effective storage, but SQL is very important for most DW applications, so your "NoSQL" store would need a powerful query tool to support common DW scenarios. However, DW experts usually won't tolerate anything that isn't SQL. Note that Cassandra is one of several NoSQL and "NewSQL" databases with a SQL dialect.

- Traditional DW

+ Mature
+ Rich SQL, analytics
- Scalability
- $$/TB

- Hadoop

- Less mature
+ Improving SQL
+ Scalable!
+ Low $/TB

Data warehouses tend to be more scalable and a little less expensive than OLTP systems, which is why they are used to “warehouse” transactional data and perform analytics. However, their $/TB is ~10x the cost on Hadoop and Hadoop scales to larger data sets.

Hadoop has become a popular data warehouse supplement/replacement.

Many of my projects have offloaded an overburdened or expensive traditional data warehouse to Hadoop. Sometimes a wholesale replacement, but more often a supplemental strategy, at least for a transitional period of some duration.

MapReduce is not ideal for graph algorithms.

Wednesday, December 4, 13

A good summary presentation: http://www.slideshare.net/shatteredNirvana/pregel-a-system-for-largescale-graph-processing
Photo: Detail of the now-closed Esquire Movie Theater, a few blocks from here, Feb. 2011

Google's Page Rank

Google invented MapReduce,

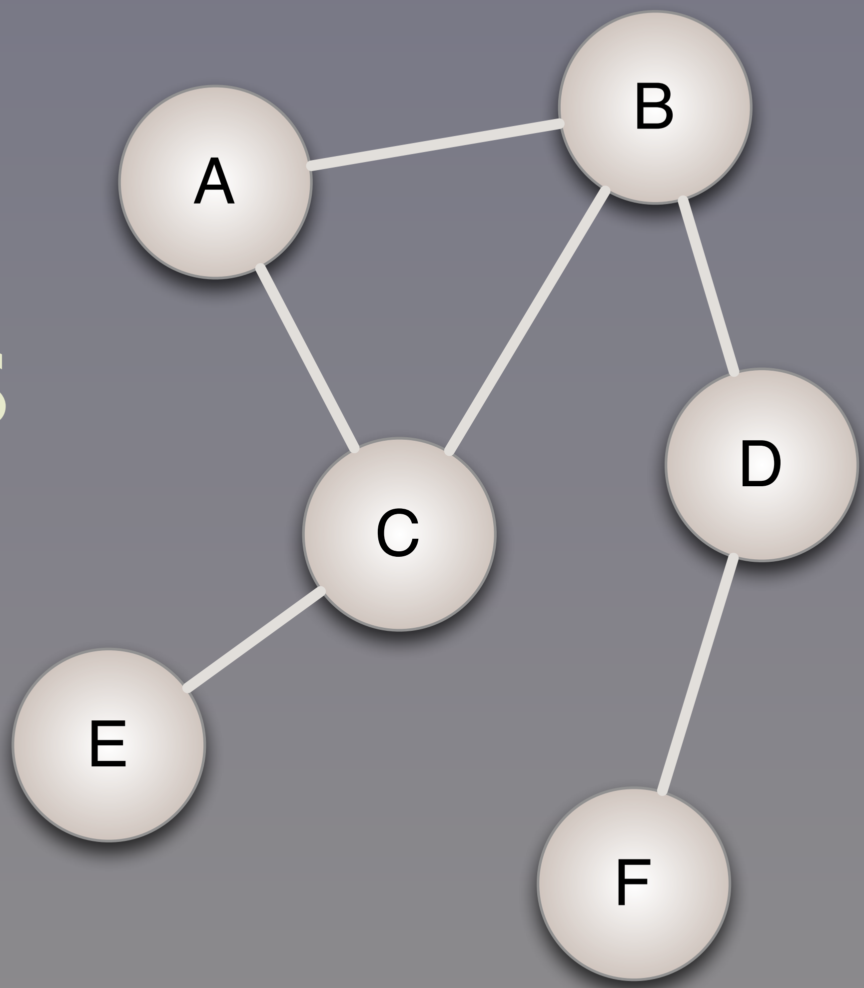
… but MapReduce is not ideal for Page Rank and other graph algorithms.

Wednesday, December 4, 13

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 (and total world domination ;).

Why not MapReduce?

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



Wednesday, December 4, 13

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.
In a nutshell, a job with a map and reduce phase is waaay to course-grained...

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.

Wednesday, December 4, 13

http://incubator.apache.org/giraph/
http://hama.apache.org/
http://thinkaurelius.github.com/titan/
None is very mature nor has extensive commercial support.
I didn't mention popular options like Neo4J because I'm focusing on cluster-oriented tools.

Sometimes, you need a specialized tool.

Purpose-built Tools

Wednesday, December 4, 13

I see that a trend where completely generic tooling is giving way to more “purpose-built” tooling...

Photo: Buildings along the Chicago River.

New compute engines (e.g., Impala, Presto).

New Hadoop file formats, optimize access (e.g., Parquet).

In the quest for ever better performance over massive datasets, the generic file formats in Hadoop and MapReduce are hitting a performance wall (although not everyone agrees). Parquet is column oriented & contains the data schema, like Thrift, Avro, and Protobuf. It will be exploited to optimize queries over massive data sets, much faster than the older file formats. Similarly, Impala is purpose built optimized query engine (that relies on Parquet).

Lucene with Solr and ElasticSearch

Example of solving a specific problem with a custom solution.

Wednesday, December 4, 13

This is an example of a specific problem domain and focused tools to solve it.
http://www.elasticsearch.org/, http://lucene.apache.org/solr/

MapReduce is not iterative, so Machine Learning algorithms perform poorly.

ML includes recommendation engines (e.g., the way Netflix recommends movies to you or Amazon recommends products), classification (e.g., SPAM classifiers, character and image recognition), and clustering. Other specialized examples include text mining and other forms of natural language processing (NLP).

Photo: Two famous 19th Century Buildings in Chicago.

- Recommendations: Netflix movies, Amazon products, ...
- Classification: SPAM filters, character recognition, ...
- Clustering: Find groups in social networks, ...

ML algorithms
tend to be iterative,
but they can be force fit
into MapReduce.

Wednesday, December 4, 13

Many ML algorithms iterate to a solution.

Machine Learning Tools

- Mahout: MapReduce algorithms.
- Pattern: PMML on Cascading.
- Spark: More flexible compute model.

Common Workflow

- Train ML models with Hadoop.
- Store model in a database.
- Predict based on user events.

Wednesday, December 4, 13

Since Hadoop is not suitable for event handling, it's common to train prediction, recommendation, etc. models with MapReduce, but store the model in a fast store, so the model can be used in real time to make predictions, etc.

Emerging: Probabilistic Programming

- Languages for Probabilistic Graphical Models??
  - Bayesian Networks.
  - Markov Chains.
  - ...

http://en.wikipedia.org/wiki/Bayesian_network
http://en.wikipedia.org/wiki/Markov_chain
PGMs are essential tools for many machine learning and artificial intelligence systems. But they require some expertise to build, both mastery of the PGM concepts and implementing them in conventional programming languages There is growing interest in designing languages that encapsulate this complexity.

Wednesday, December 4, 13

Now that we have cataloged some issues and solutions, let's recap and look forward.
Photo: Lake Michigan, near Ohio Street Beach, Chicago, Feb. 2011.

Wednesday, December 4, 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...

Wednesday, December 4, 13

We've seen a lot of issues with MapReduce. Already, alternatives are being developed, either general options, like Spark and Storm, or special-purpose built replacements, like Impala. Let's consider other options...

Emerging replacements are based on Functional Languages...

```scala
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")))
}
```


FP is such a natural fit for the problem that any attempts to build big data systems without it will be handicapped and probably fail.
Let's consider other MapReduce options...

… and SQL, which is roaring back!

```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;
```


FP is such a natural fit for the problem that any attempts to build big data systems without it will be handicapped and probably fail.
Let's consider other MapReduce options...

Why are Scala, Clojure, and SQL solutions so concise and appealing?

Big Data is Mathematics.

∴ Functional Languages are the best tools.

Concurrency
has been called
the killer app for FP.

Big Data is a
bigger killer app, IMHO.

I think big data may drive FP adoption just as much as concurrency concerns, if not more so. Why? Because I suspect more developers will need to get “good” at data, vs. good at concurrency.

Wednesday, December 4, 13

Photo: Building in fog on Michigan Avenue, Chicago.