goto; conference aarhus

FROM BIG DATA TO BIG INFORMATION

Dean Wampler (@deanwampler)
Concurrent Thought

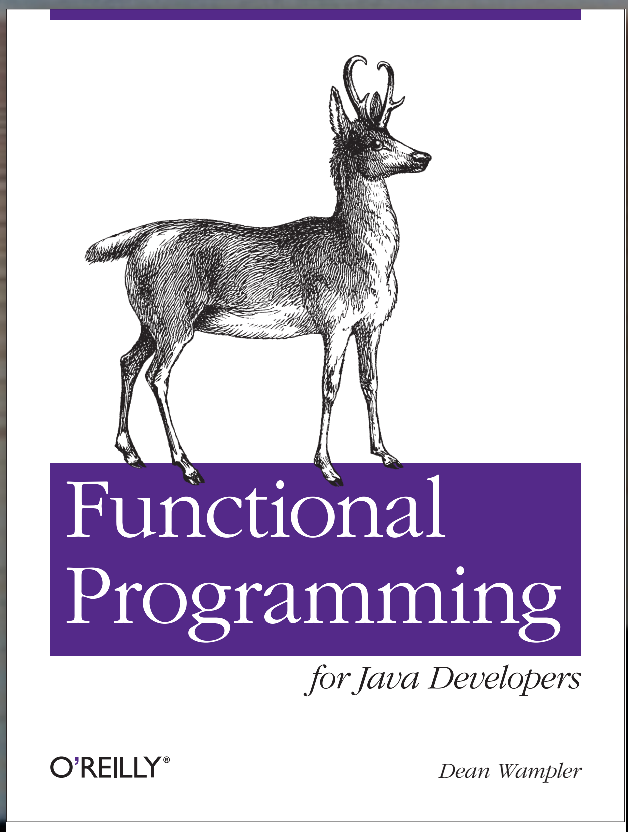
INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

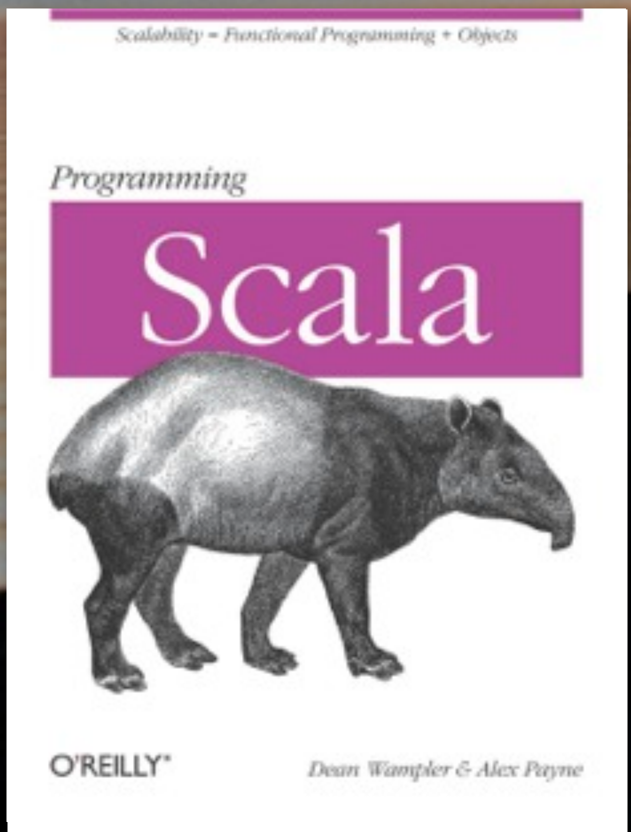
gotocon.com

Tuesday, October 1, 13

Who am I?

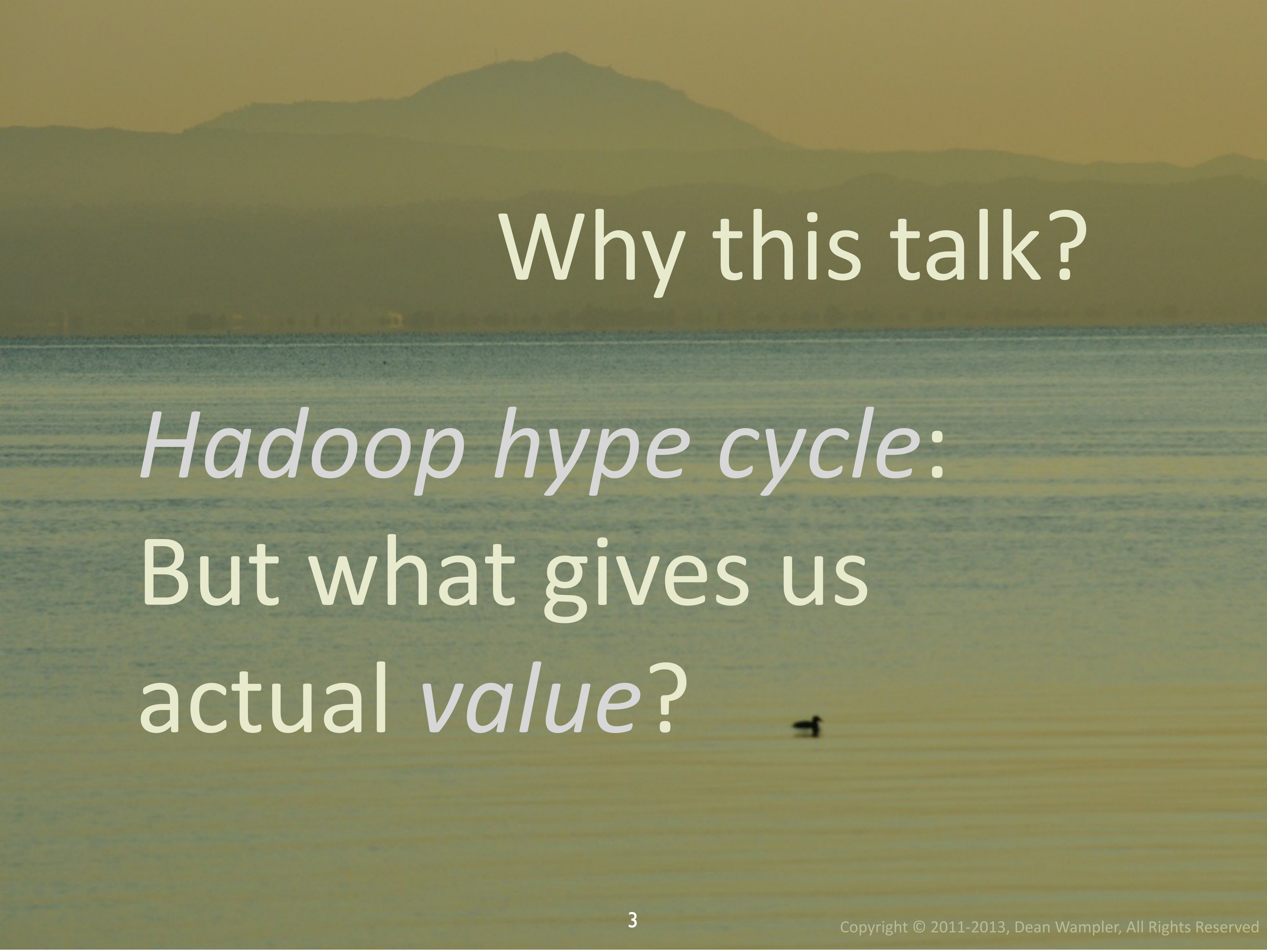
dean@concurrentthought.com
@deanwampler
polyglotprogramming.com/talks





Tuesday, October 1, 13

Photo: San Francisco Bay, just south of the airport in Burlingame, before sunrise.



This talk reflects my experiences working on “Big Data” projects, mostly using Hadoop, with many clients.
People sometimes jump on this bandwagon to avoid “being left behind” or to boost their career. Sometimes, it doesn’t make sense for their actual needs. Big Data itself doesn’t do you any good. It’s the information we extract that’s important, so let’s see how different technologies address specific problems.

Photo: Photo: San Francisco Bay, Burlingame, around sunrise.

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

Tuesday, October 1, 13

Photo: Photo: San Francisco Bay, Burlingame, around sunrise.

Big Data

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

Hat tip: Bob Korbus

Tuesday, October 1, 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.

Tuesday, October 1, 13

Three trends to organizer our thinking…

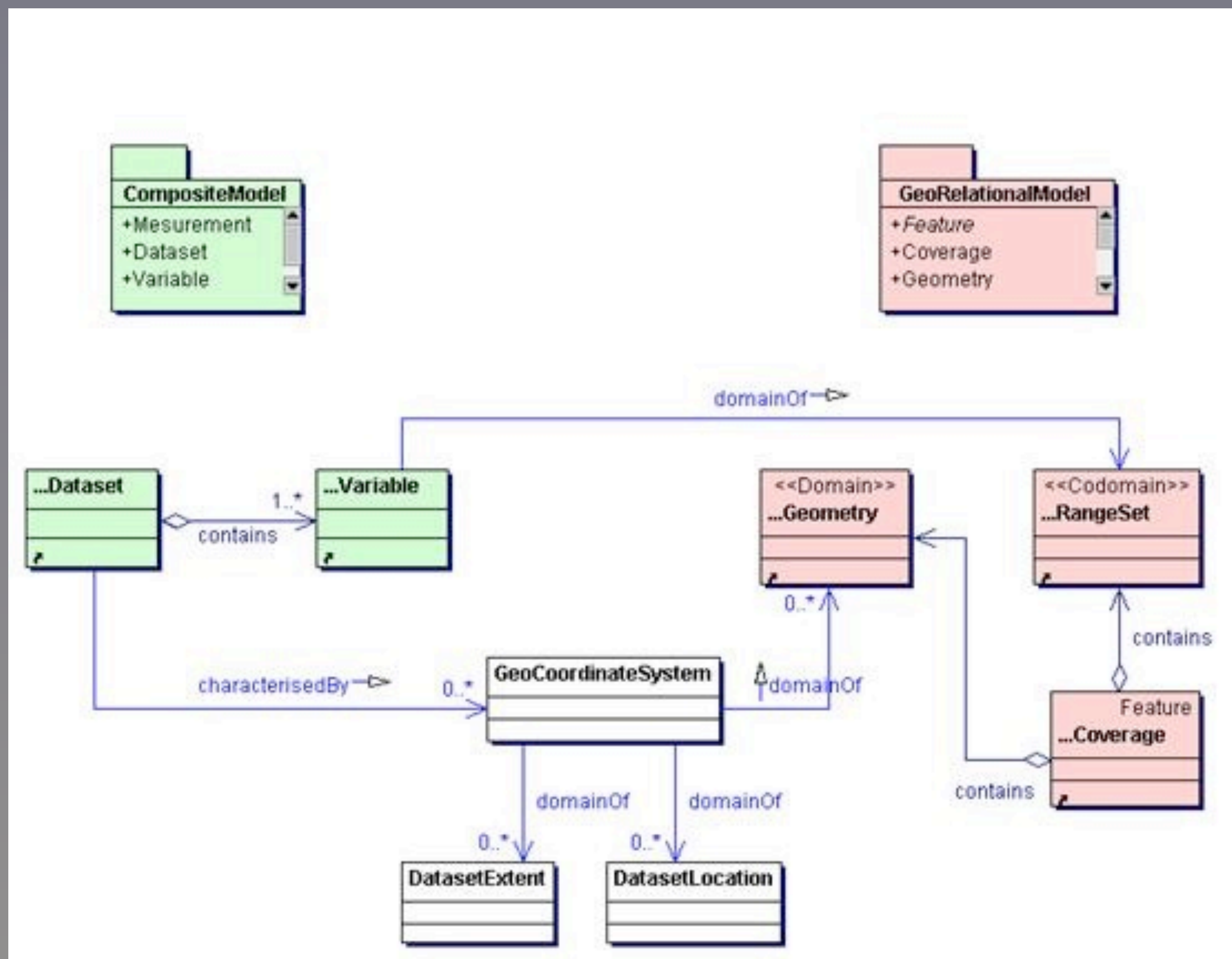
Photo: Gull on a pier near Fort Mason, SF

Data Size ⬆

Tuesday, October 1, 13

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

Formal Schemas ⬇



Tuesday, October 1, 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 ⬆

Tuesday, October 1, 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...)

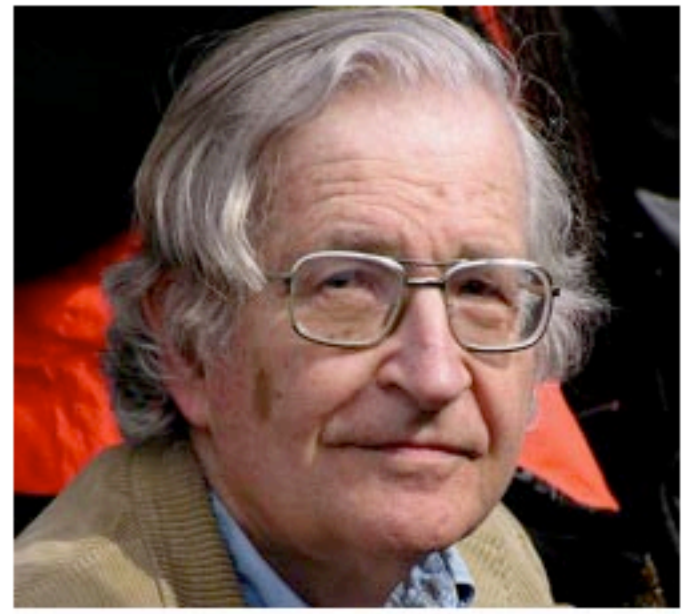
Probabilistic Models vs. Formal Grammars

tor.com/blogs/...

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,



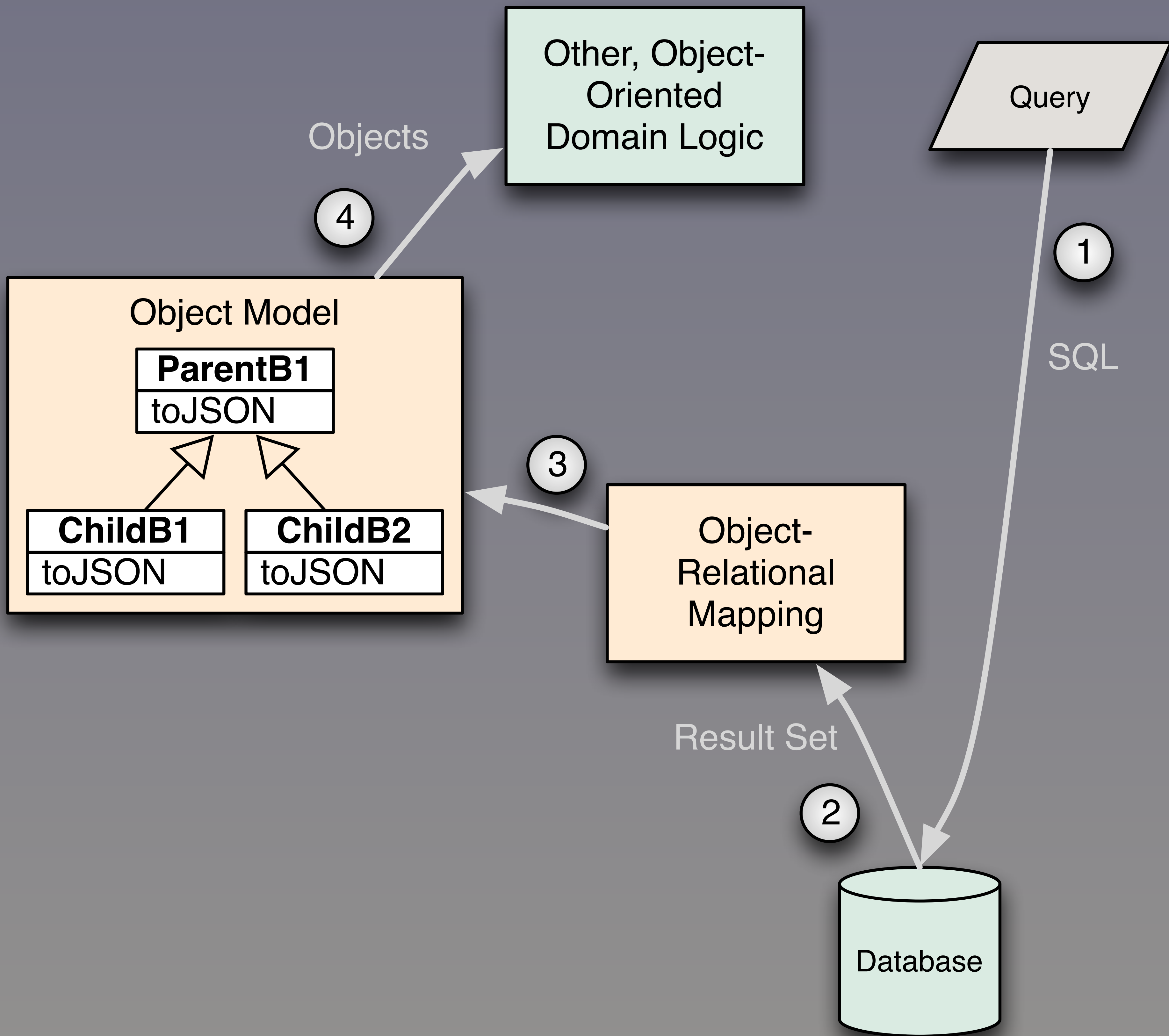
Chomsky photo by Duncan Rawlinson and his Online Photography School. Norvig photo by Peter Norvig

Tuesday, October 1, 13

An interesting manifestation of the last two points 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

Tuesday, October 1, 13

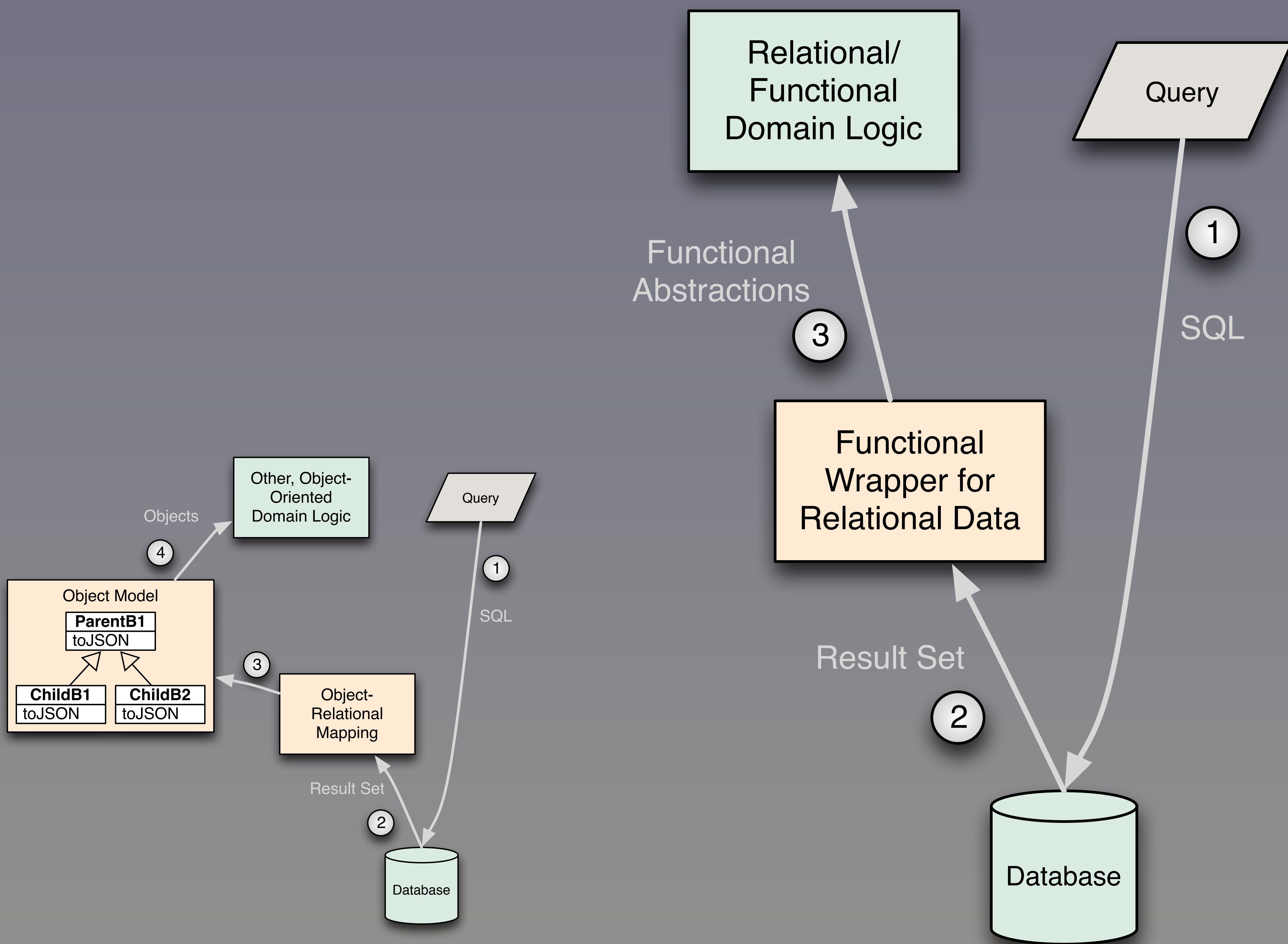
What should software architectures look like for these kinds of systems?
Photo: Light on the Boardwalk and Coit Tower.



Tuesday, October 1, 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…)

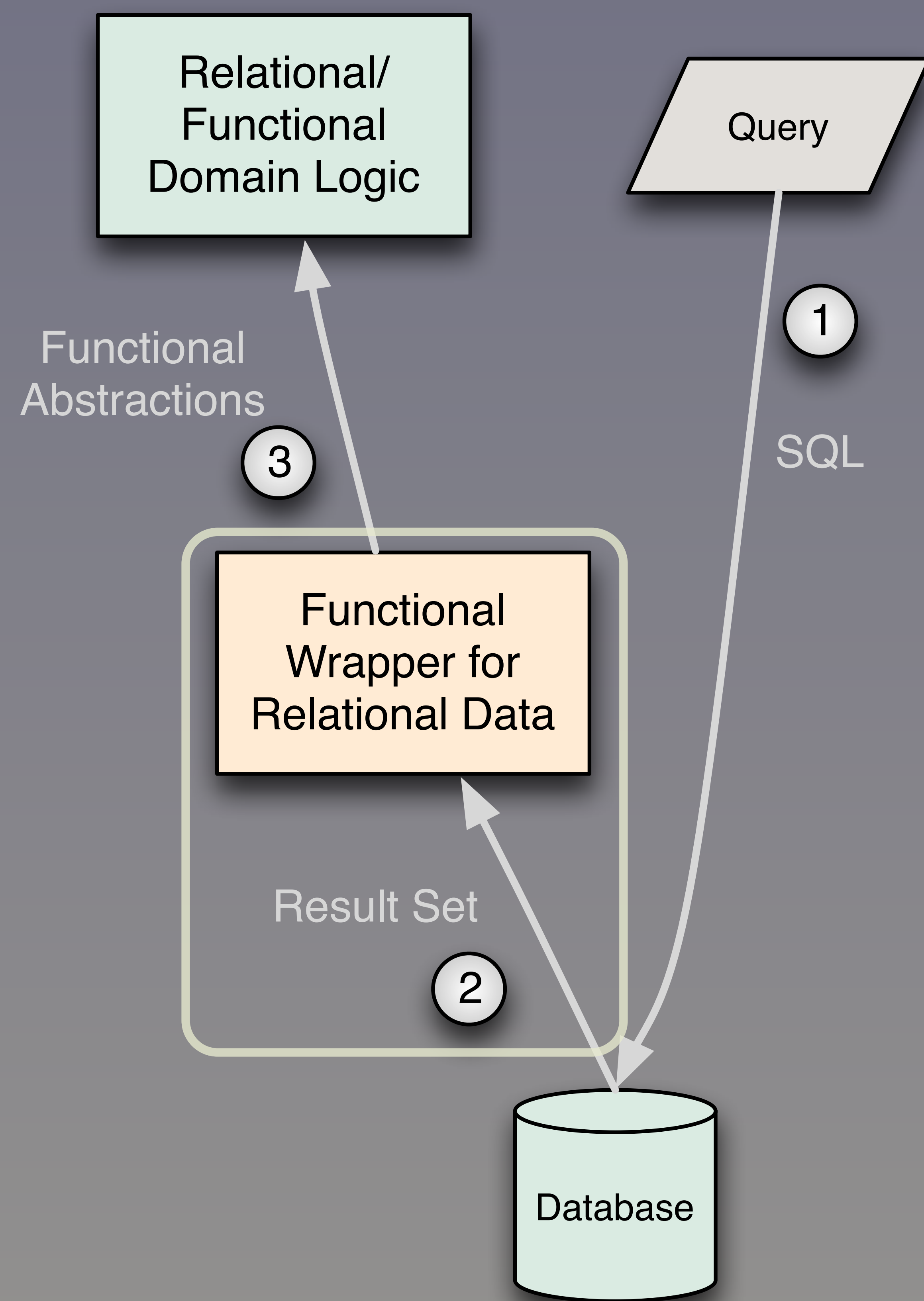
The toJSON methods are there because we often convert these object graphs back into fundamental structures, such as the maps and arrays of JSON so we can send them to the browser!



Tuesday, October 1, 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.

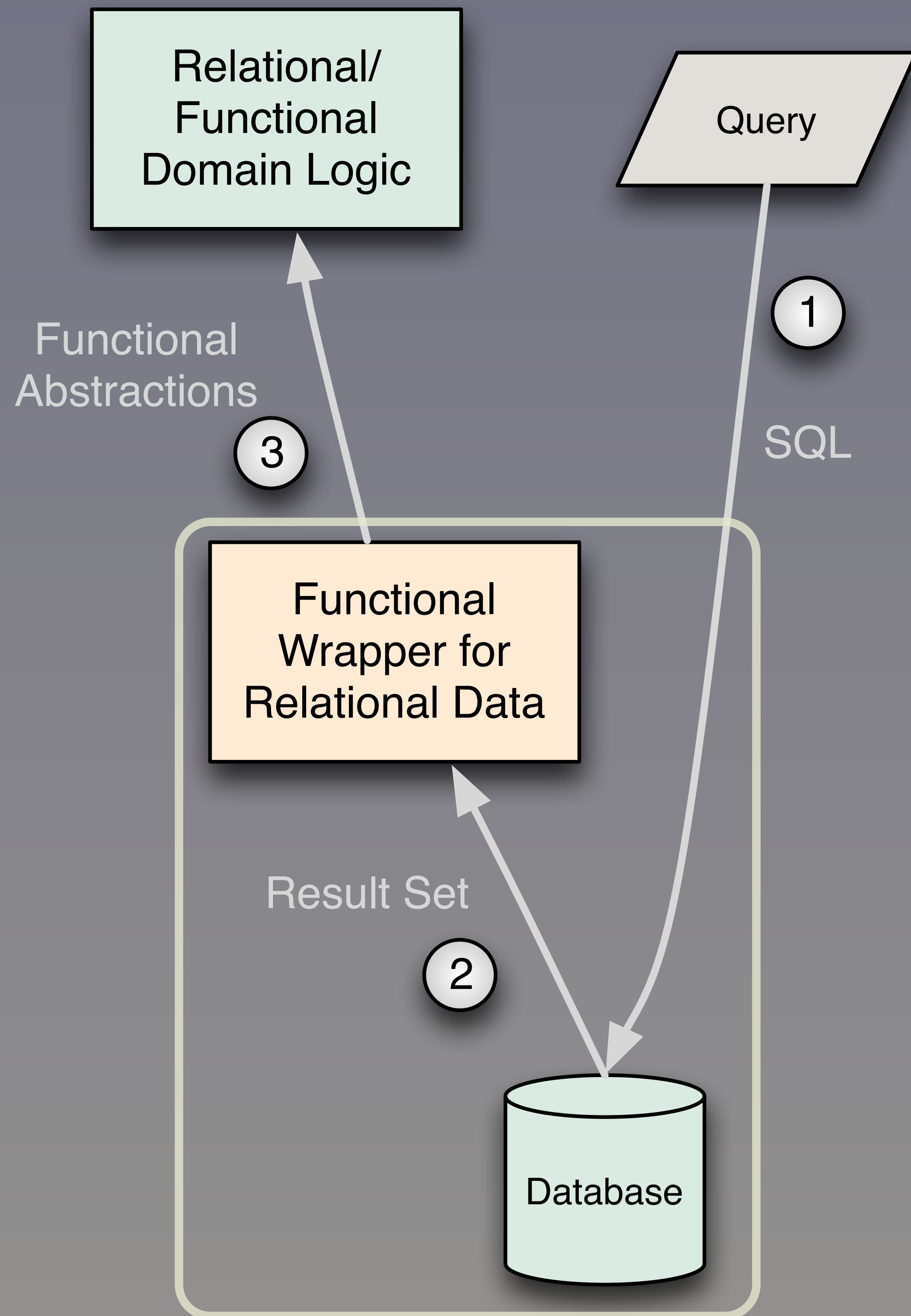
- Focus on:
  - Lists
  - Maps
  - Sets
  - Trees
  - ...



Tuesday, October 1, 13

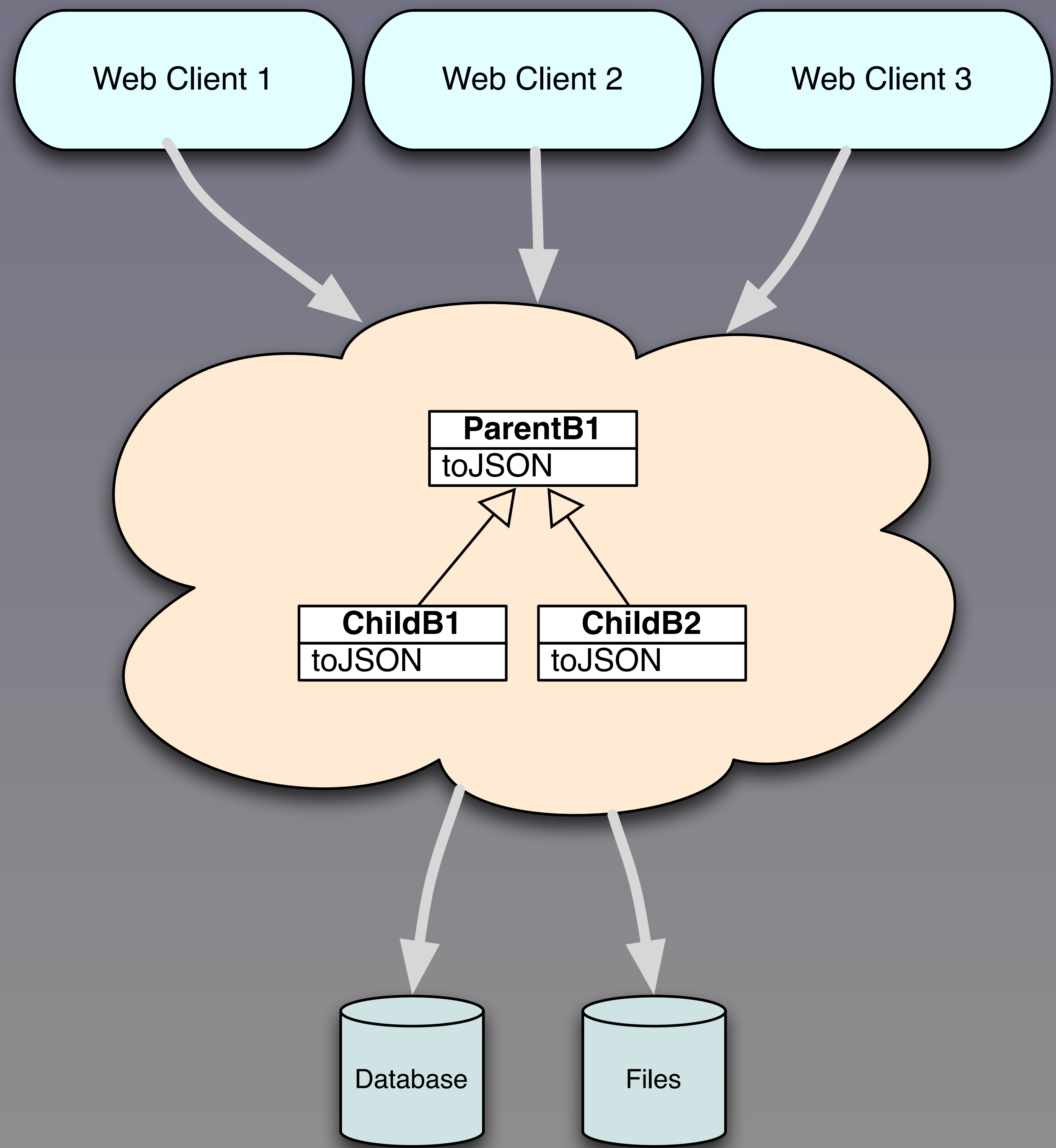
But the traditional systems are a poor fit for this new world: 1) they add too much overhead in computation (the ORM layer, etc.) and memory (to store the objects). Most of what we do with data is mathematical transformation, so we're far more productive (and runtime efficient) if we embrace fundamental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

- NoSQL?
  - Cassandra, HBase
  - Riak, Redis
  - MongoDB
  - Neo4J
  - ...



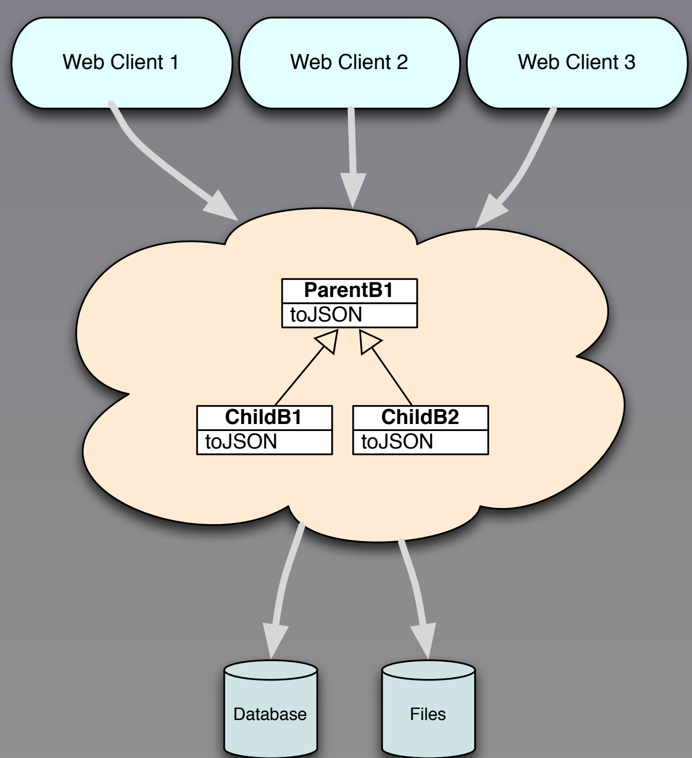
Tuesday, October 1, 13

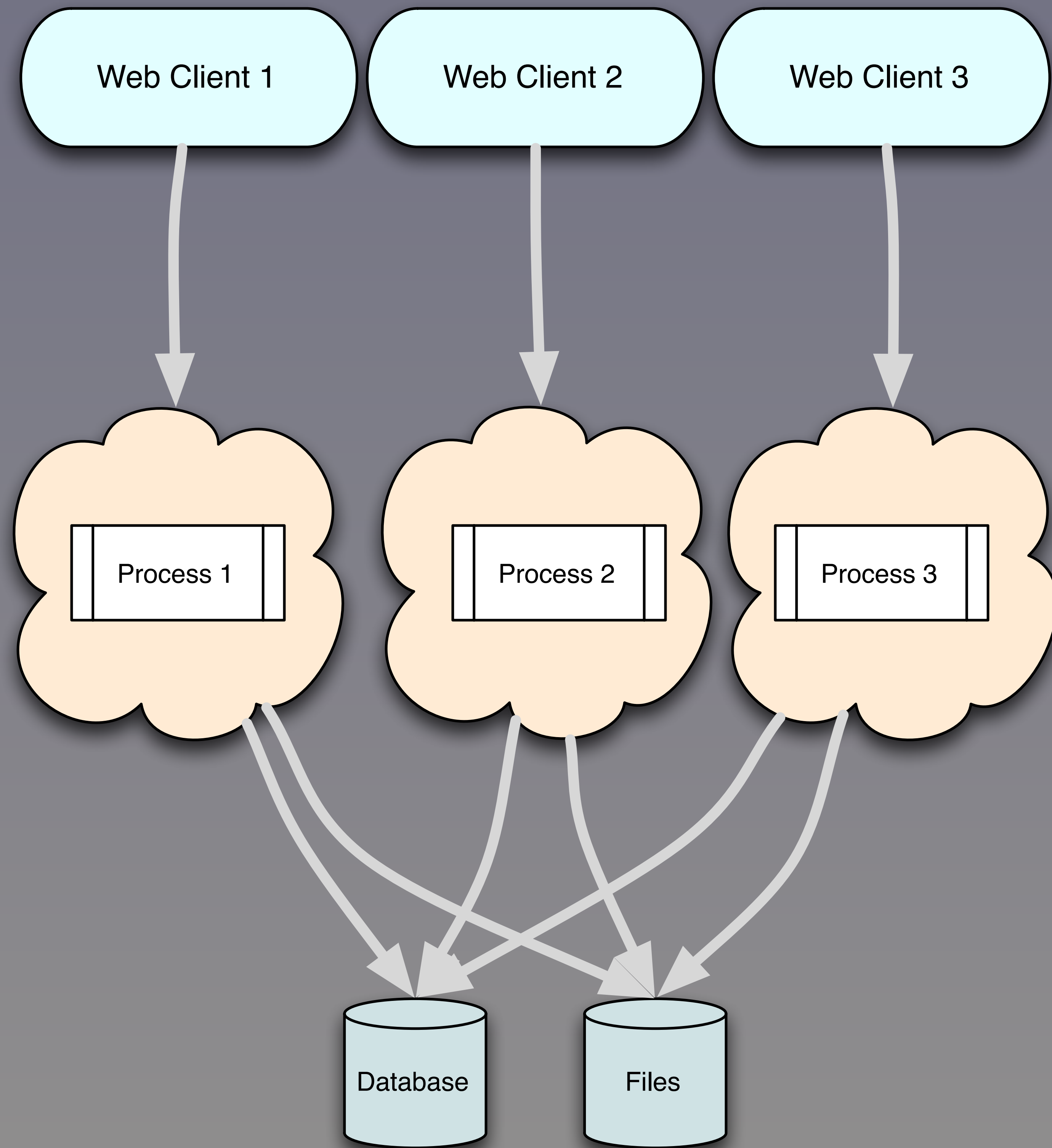
NoSQL databases fit this model well, with focused abstractions for their data model.



Tuesday, October 1, 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.



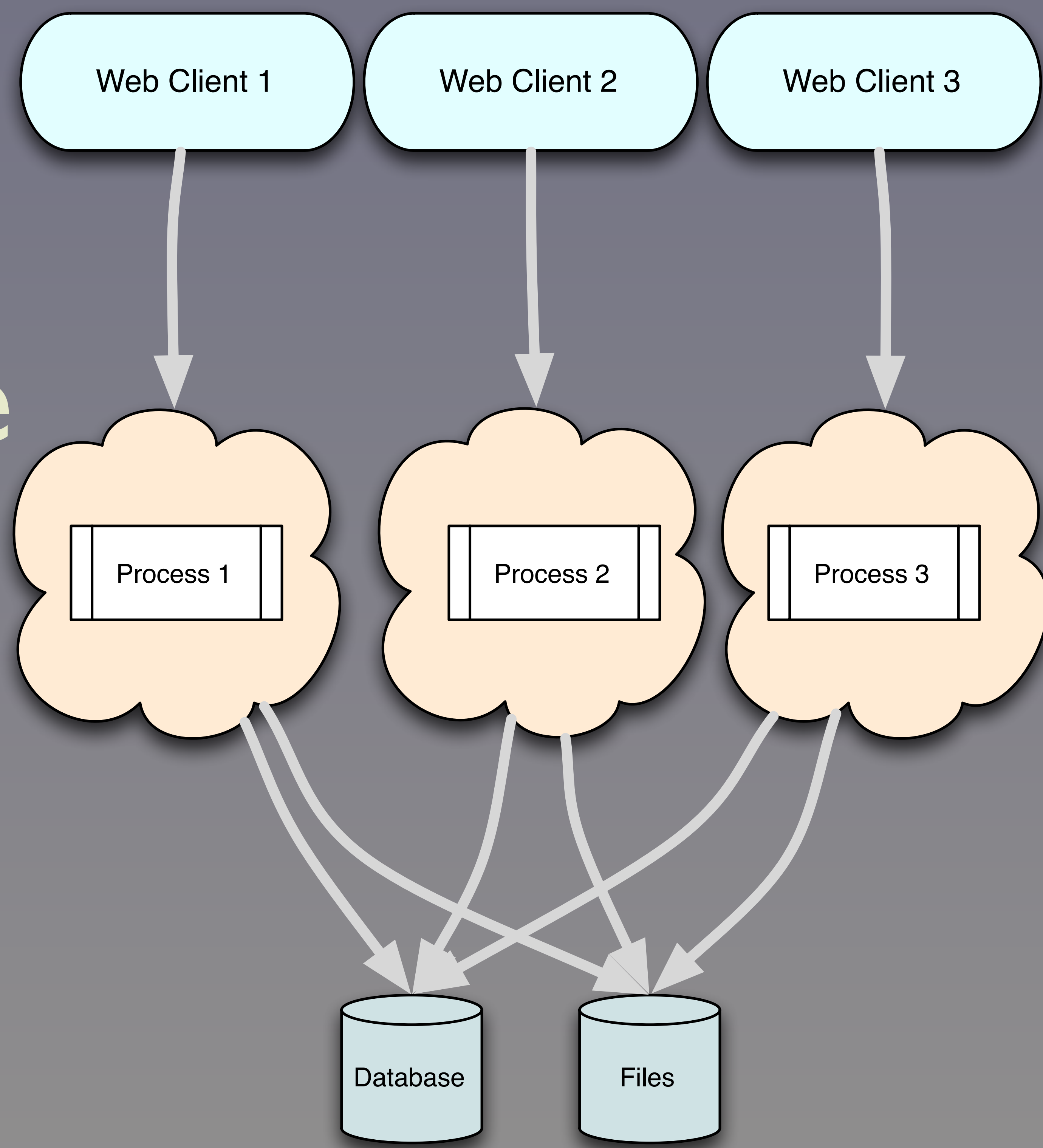


Tuesday, October 1, 13

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.

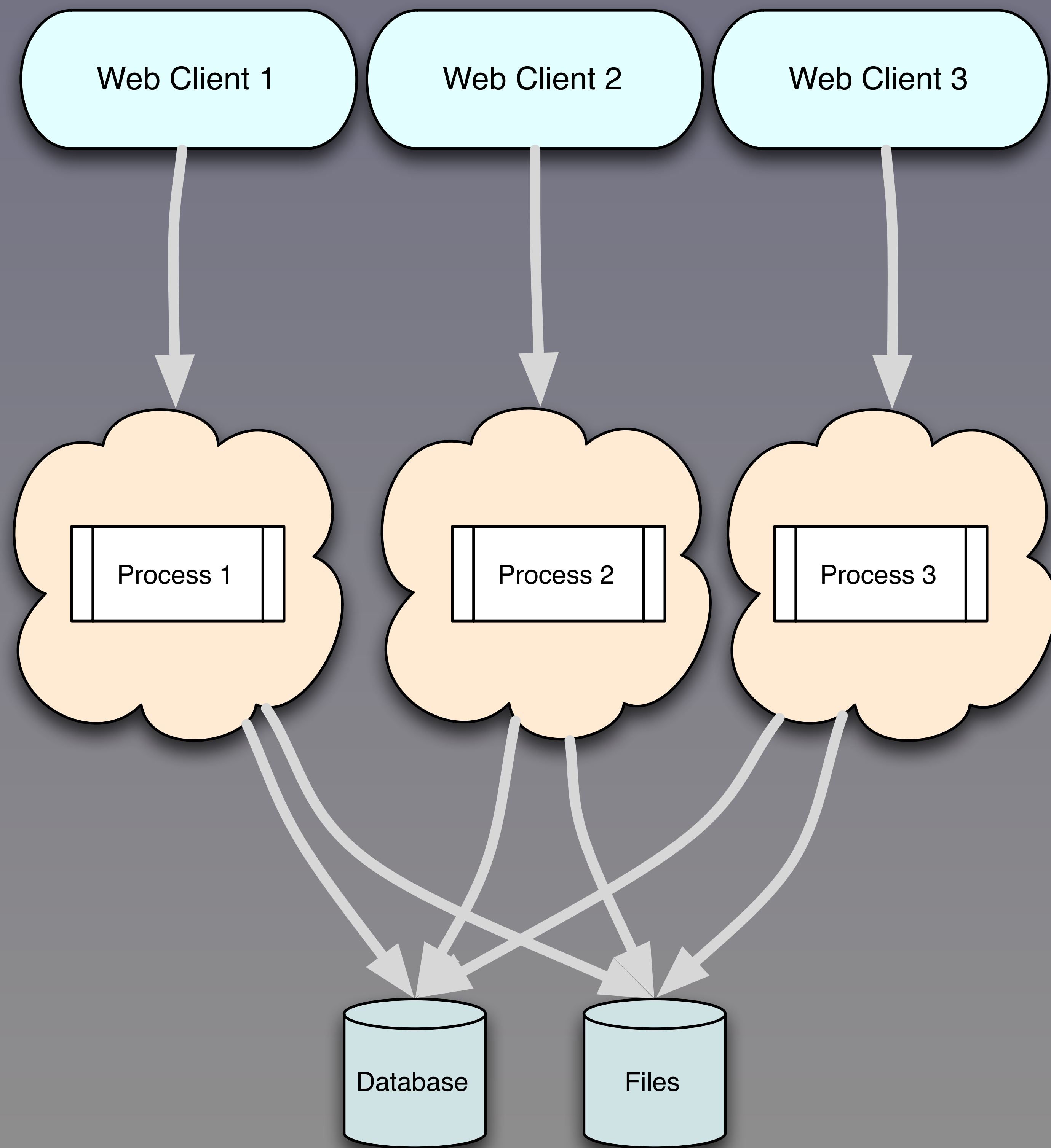
- Smaller, more focused middleware



Tuesday, October 1, 13

An important "software engineering" benefit is the way it promotes smaller, more focused middleware, avoiding bloat that is hard to evolve.

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

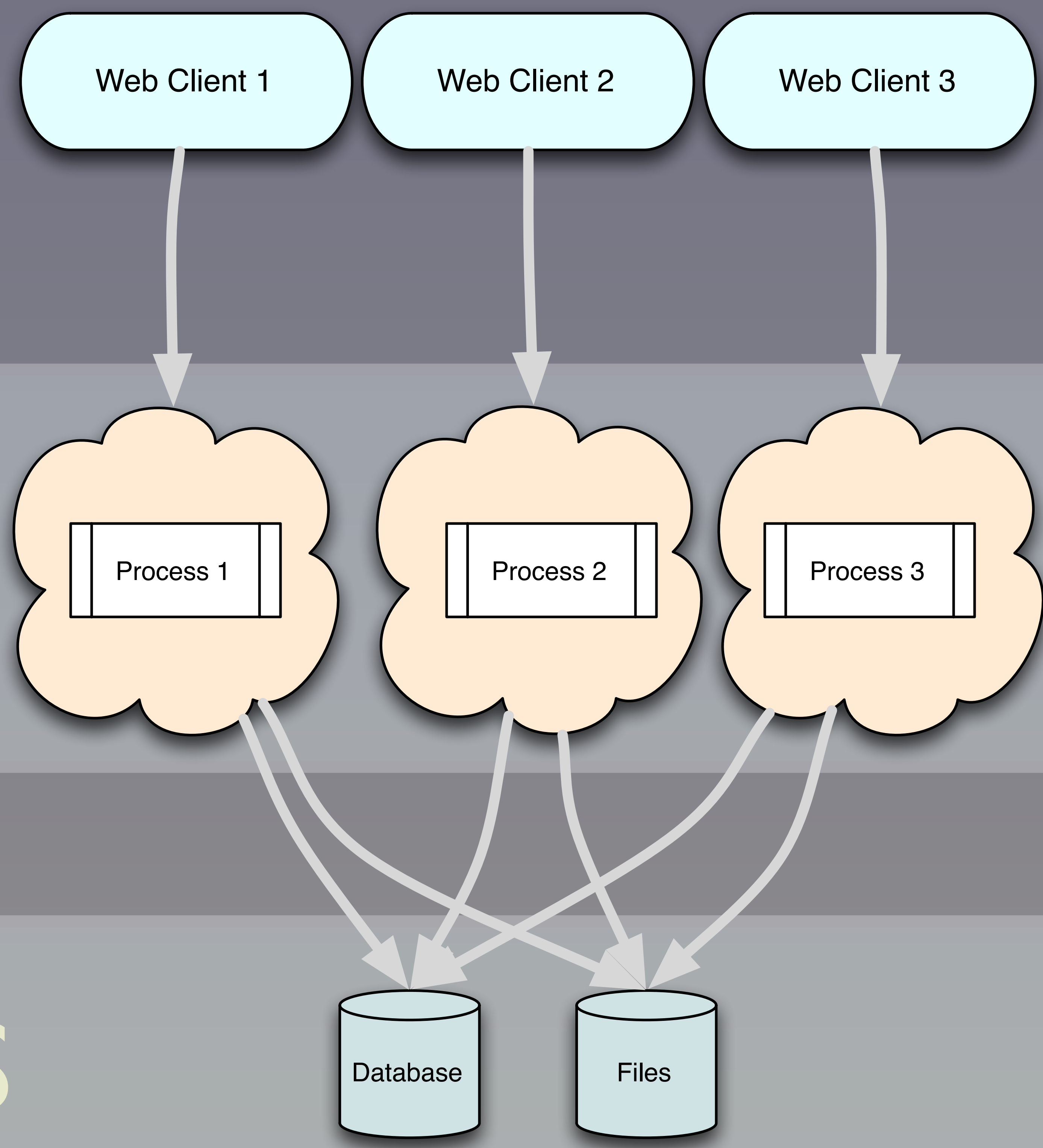


Tuesday, October 1, 13

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

- MapReduce

- Distributed FS

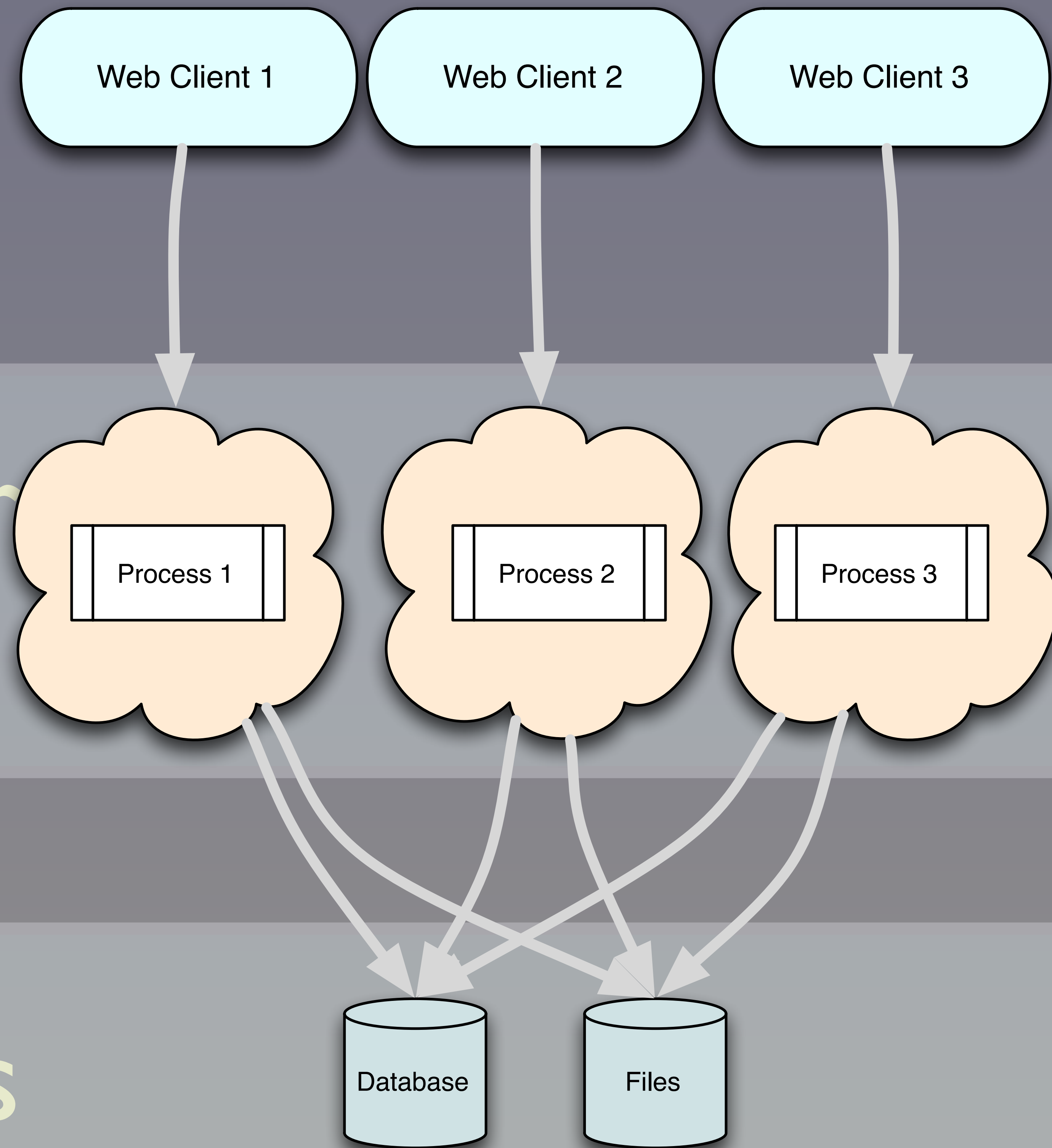


Tuesday, October 1, 13

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

- Hadoop, other middleware

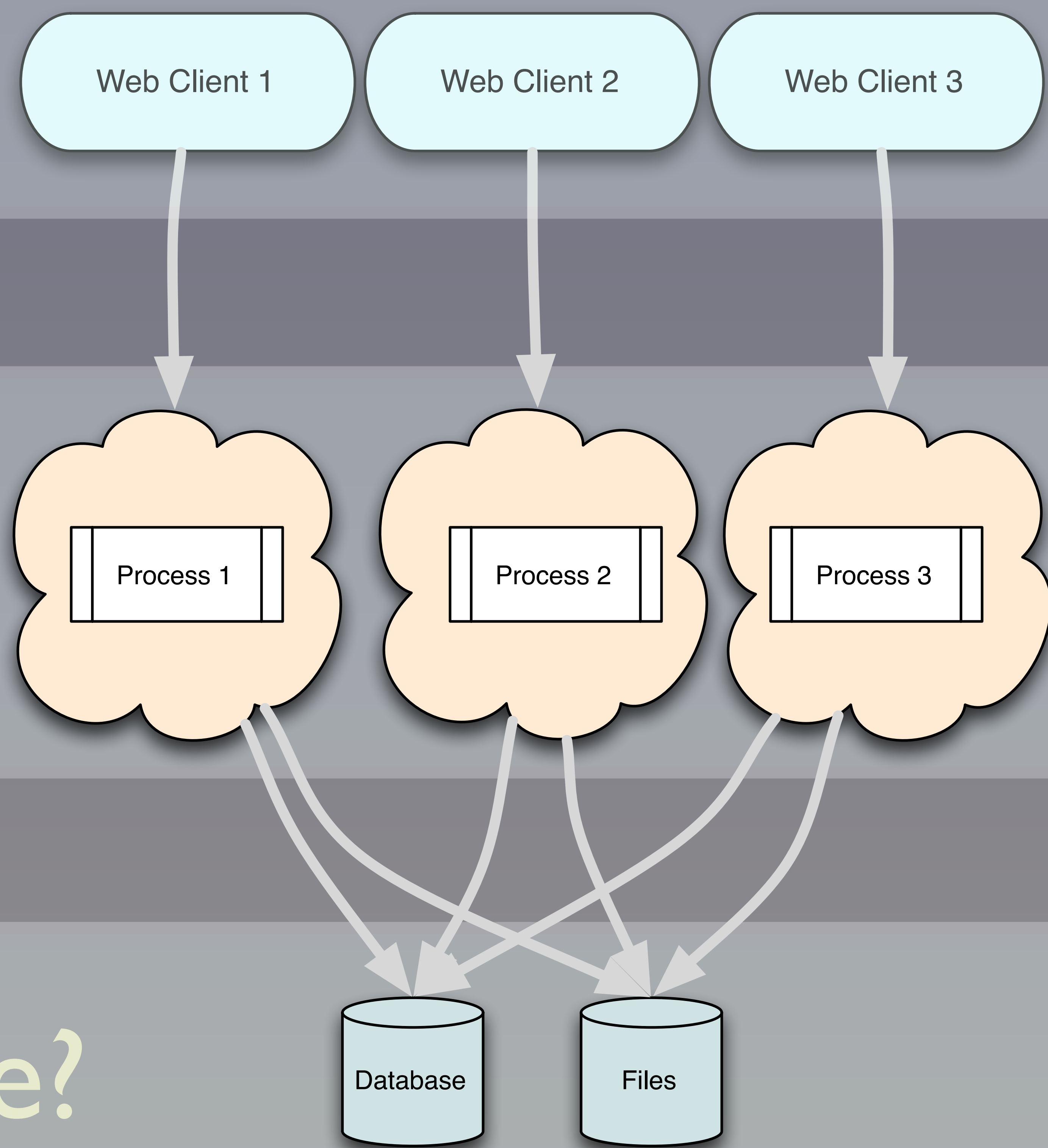
- NoSQL stores



Tuesday, October 1, 13

MapReduce and a “bare”, distributed file system aren’t the right model for many systems (as we’ll discuss). A similar model uses a NoSQL store and ad-hoc middleware. In this case, more logic is in the storage layer vs. more limited capabilities in a distributed file system.

- JSON
- Node.js?
- JSON database?



Tuesday, October 1, 13

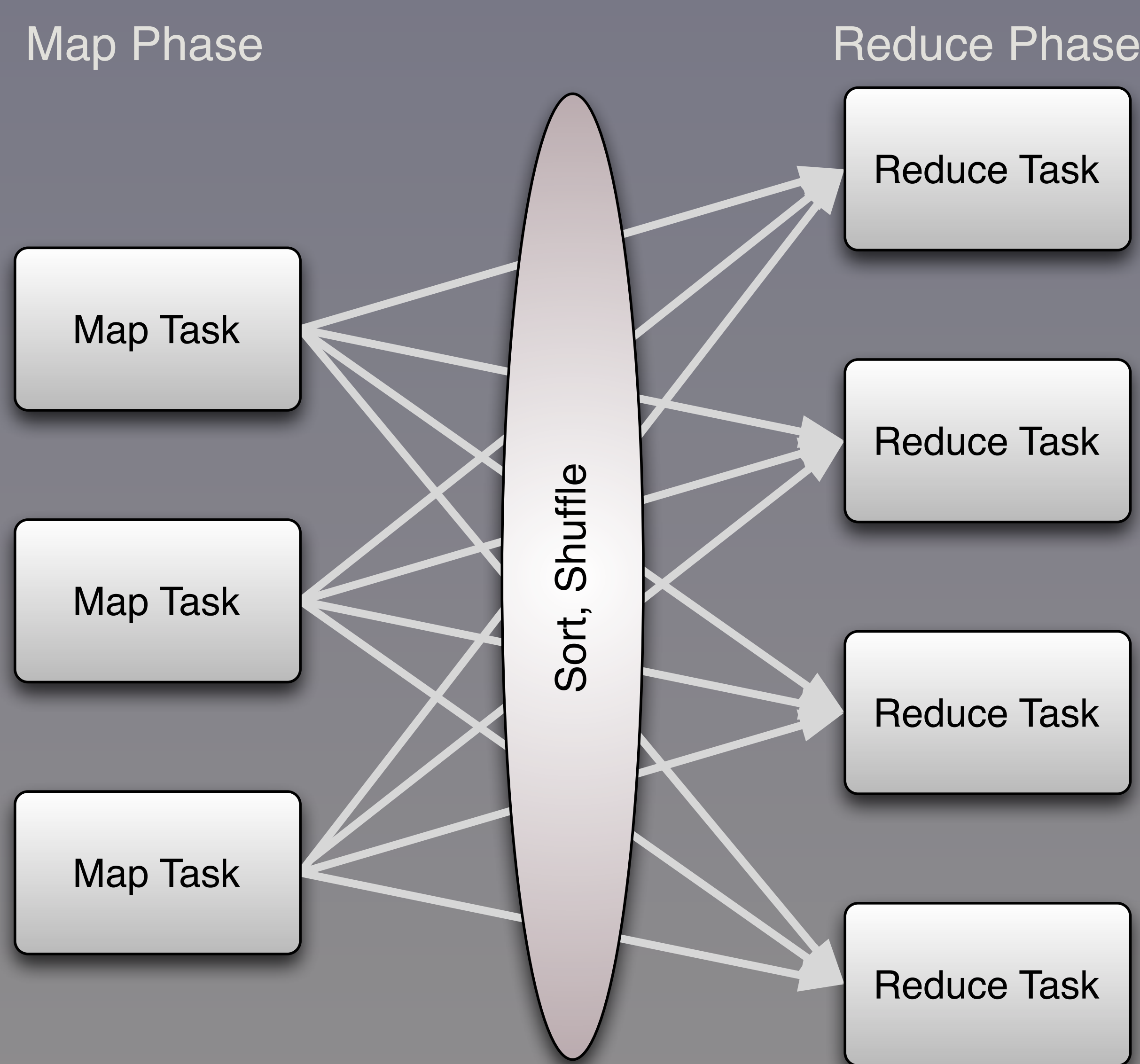
One interesting incarnation of this is JavaScript through the full stack, with JSON as the RPC format, stored directly (more or less) in a database like Mongo, CouchBase, and RethinkDB. Node gives you JS in the mid-tier, and JSON is obviously a browser tool.

Tuesday, October 1, 13

Let's be clear about what MR actually is.

Photo: Near the Maritime Museum

Anatomy: *MapReduce Job*



Map (or *Flatmap*):

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

Reduce:

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

Tuesday, October 1, 13

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

For your consideration...

Arguably, Hadoop is our best, generic* tool for scaling Big Data horizontally (at least today).

As I'll argue it's definitely more first-generation than "perfect". By generic, I mean beyond the more focused goals of a database, SQL or NoSQL, in the sense that it provides a very open-ended compute model and primitive storage, on which you can build a database, which is exactly what HBase is! However, HBase doesn't use MapReduce, just HDFS with its own compute engine. The asterisk is because MR is very general-purpose, but actually somewhat difficult, inflexible, and inefficient for many algorithms.

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

Not so great for *event-stream* processing.

Tuesday, October 1, 13

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

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

Not so great for *transactions*.

Tuesday, October 1, 13

Also, there's no built-in concept of transactional behavior in Hadoop.

MapReduce is very course-grained.

1-Map and
1-Reduce phase...

Tuesday, October 1, 13

I’m going to revisit this topic as we go...

Spark is a Hadoop MapReduce alternative:

- Distributed computing with in-memory caching.
- Up to 30x faster than MapReduce.
- Developed by Berkeley AMP.

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

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

Tuesday, October 1, 13

There are great higher-level APIs that relieve this burden. We can't discuss them today, but see Cascading, Scalding, Cascalog, for examples.

Word Count: Hadoop Java API

```
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 text,
        OutputCollector<Text, IntWritable> output, Reporter reporter) {
      String[] tokens = text.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 '90s called. They want their EJBs back!

Tuesday, October 1, 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!

Word Count: Scalding Scala API

```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!!

Tuesday, October 1, 13

This Scala code using Twitter's Scalding API for Hadoop is almost pure domain logic with very little boilerplate. There are a few minor differences in the implementation. There is little green for "type ceremony", in part because Scala infers types in many cases. There is a lot more yellow for the functions that do real work!

What if MapReduce, and hence 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.

Word Count: Spark Scala API

```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!

Tuesday, October 1, 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.

Usage Scenarios

Let's look at some usage scenarios to understand the strengths and weaknesses of Hadoop-oriented solutions vs. NoSQL-oriented solutions.

Photo: Transamerica Building in San Francisco

TL;DR

- *Hadoop*
- Very flexible compute model
- "Table" scans
- Batch mode

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

Top-level comparison. These are points of emphasis for these options. What I mean is that we tend to write Hadoop-centric or database-centric apps, depending on the scenario. (NoSQL doesn't force a particular data model, like the relational model, but usually the data has a particular "shape".)

TL;DR

Need both?
Hadoop is often used with a *NoSQL* store.

Just to be clear that there isn't a clear divide between these technologies, of course.
For example, it's not uncommon to integrate MR jobs with Cassandra, HBase, MongoDB, and even relational tables.

Low-cost Data Warehouse

Tuesday, October 1, 13

Data warehouse systems hit scalability limits and cost/TB concerns at large scale…
Some SQL-based data warehouse systems can cost 100x (per TB) what Hadoop costs. You trade off maturity for cost and flexibility.

Problem:

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

Very common scenario...

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.

Solution?

Replace the *data warehouse* with *NoSQL*?

Could you use a NoSQL store instead??

SQL is very *important* for *data warehouse* applications.

Tuesday, October 1, 13

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.

Solution?

Replace the *data warehouse* with *Hadoop*?

Could you use Hadoop instead??

- *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.

SQL on Hadoop

Tuesday, October 1, 13

Let's discuss the SQL options now, as I consider them very important. Otherwise, the majority of Data Analysts and casual SQL users would not find Hadoop very useful.

Use SQL when you can!

- Hive: SQL on top of MapReduce.
- Shark: Hive ported to Spark.
- Impala: HiveQL with new, faster back end.
- 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. Impala is very new. It doesn't yet support all Hive features.

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

Tuesday, October 1, 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.

Hive

- SQL dialect.
- Uses MapReduce (today...).
  - So annoying latency.
- First SQL on Hadoop.
- Developed by Facebook.

http://hive.apache.org
There is an active effort to wean Hive from its MR back end, replacing it with a must faster query engine for more rapid results. This is one of many examples where MR is slowly being replaced.

Shark

- HiveQL front end.
- Spark back end.
- Provides better performance.
- Developed by Berkeley AMP.

See http://www.cloudera.com/content/cloudera/en/products/cloudera-enterprise-core/cloudera-enterprise-RTQ.html. However, this was just announced a few ago (at the time of this writing), so it's not production ready quite yet...

Impala

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

Tuesday, October 1, 13

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

MapReduce EoL??

- Impala is one of many examples where MapReduce is being replaced with more performant and flexible compute models.

Tuesday, October 1, 13

MR is slowly being phased out… very slowly.

Lingual

- ANSI SQL front end.
- Cascading back end.
  - Same strengths/weaknesses for runtime performance as Hive.

Tuesday, October 1, 13

http://www.cascading.org/lingual/ A relatively new "API" for Cascading, still in Beta. Because Cascading runs on MapReduce (there's also a standalone "local mode" for small jobs and development), it will have the same perf. characteristics as Hive and other MR-based tools. However, Cascading may eventually replace its dependence on MR with a better compute engine.

Search

Tuesday, October 1, 13

Improving information search and retrieval, usually through some means of indexing. An example of a return to a technology customized for a particular class of problems, as opposed to relying on something very generic, like Hadoop.

Lucene with Solr or ElasticSearch

A *specific solution* for *search*.

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

Tuesday, October 1, 13

Data warehouse systems hit scalability limits and cost/TB concerns at large scale...

Recall, Hadoop is *great* for *batch mode* data crunching.

Not so great for *event-stream* processing.

Tuesday, October 1, 13

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

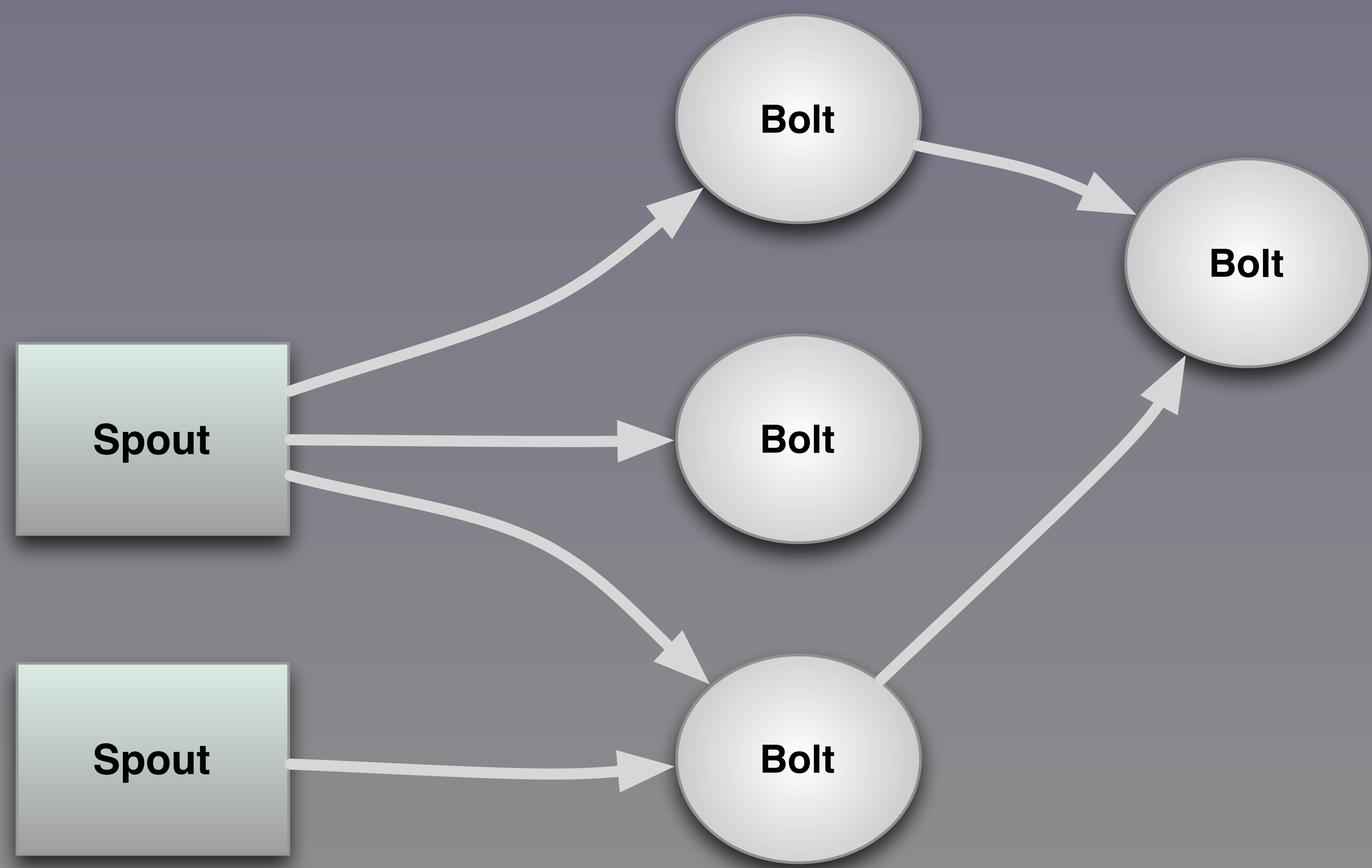
Tuesday, October 1, 13

Nathan Marz's Storm system for scalable, distributed event stream processing. A complement to Hadoop, as he describes in detail in his book "Big Data" (Manning).

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

Storm implements reliable, distributed event processing.

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



Tuesday, October 1, 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.

- *Hadoop*

+ Cheap

+ Scalable

+ Commercial Support

- Batch mode

- *Storm*

- Less mature

+ Robust

- Commercial Support

+ Event Streams

Tuesday, October 1, 13

If you need to respond to event streams, Hadoop simply isn't suitable. Storm is one possible solution. It was designed for this problem, but it is less mature and commercial support is just starting to appear.

Databases to the Rescue?

Tuesday, October 1, 13

Photo: Actually, this is another Chicago photo, looking out my condo window. (These heavy-lift helicopters are used to lift new heating and air conditioning units from the street to the tops of buildings.)

SQL or NoSQL Databases?

Databases are designed for fast, transactional updates.

So, consider a database for event processing.

Tuesday, October 1, 13

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

Using Hadoop?

Combine HBase

(a NoSQL database)

with Hadoop.

HBase uses HDFS for durable storage, so it's already integrated well with Hadoop. Teams that need both database operations and MR jobs often use this combination.

Tuesday, October 1, 13

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).

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

Arbitrary *ML algorithms* can be implemented using *MapReduce*, but they tend to be *iterative*.

Many ML algorithms iterate to a solution.

Pattern

A new toolkit for writing *models* in *SAS* , etc., then run them on *Cascading*.

Tuesday, October 1, 13

See details about Pattern on http://cascading.org
Hides much of the complexity of implementing predictive models, although it's still using Hadoop under the hood.

Spark

Alternative to *MapReduce* with more *flexible* and *composable* abstractions.

Tuesday, October 1, 13

http://spark-project.org/

Spark

Originally designed for *Machine Learning*.

Tuesday, October 1, 13

http://spark-project.org/

Train with *Hadoop*, etc.

Serve requests with *NoSQL* store.

Tuesday, October 1, 13

A common model is to use Hadoop to train models (e.g., recommendation engine) over very large data set, then store the model in an event-processing system (NoSQL, Storm) to serve requests in real time. (Problem: how do you update the model to reflect new inputs? Rerun training periodically or use an “online” algorithm – out of scope here!)

Social Network Data

How should you represent associations in social networks, e.g., friends on Facebook, followers on Twitter, ...

Google's Page Rank

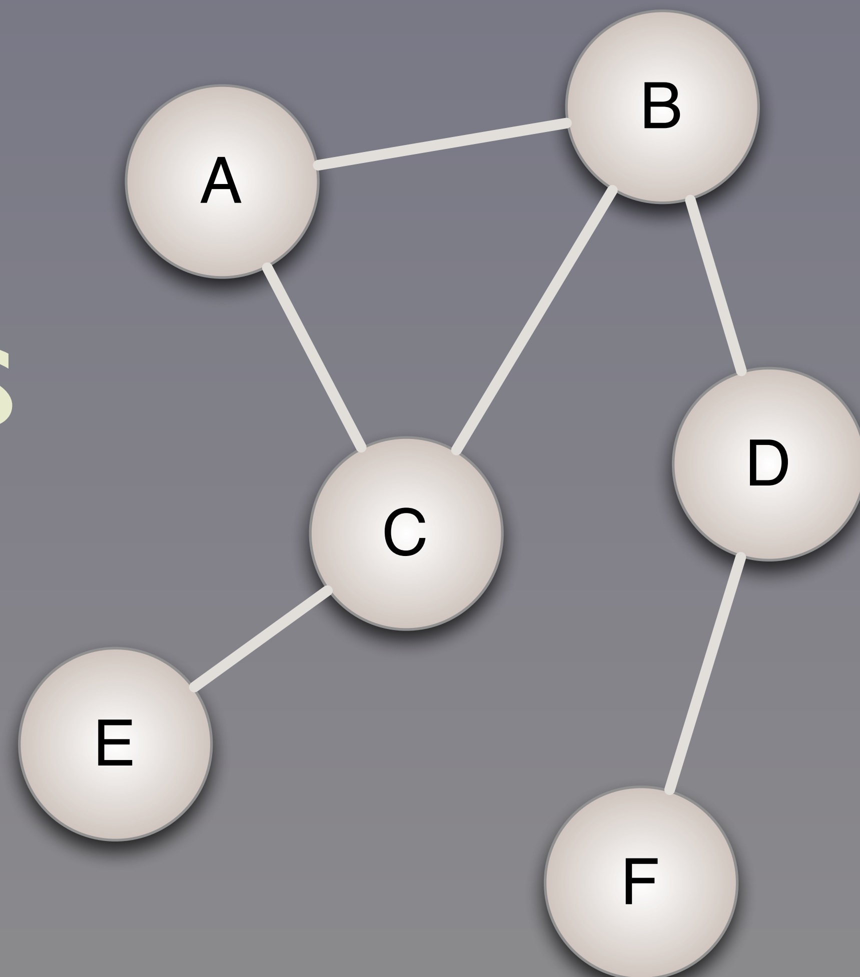
Google invented MapReduce,

… but MapReduce is not ideal 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 (and total world domination ;).

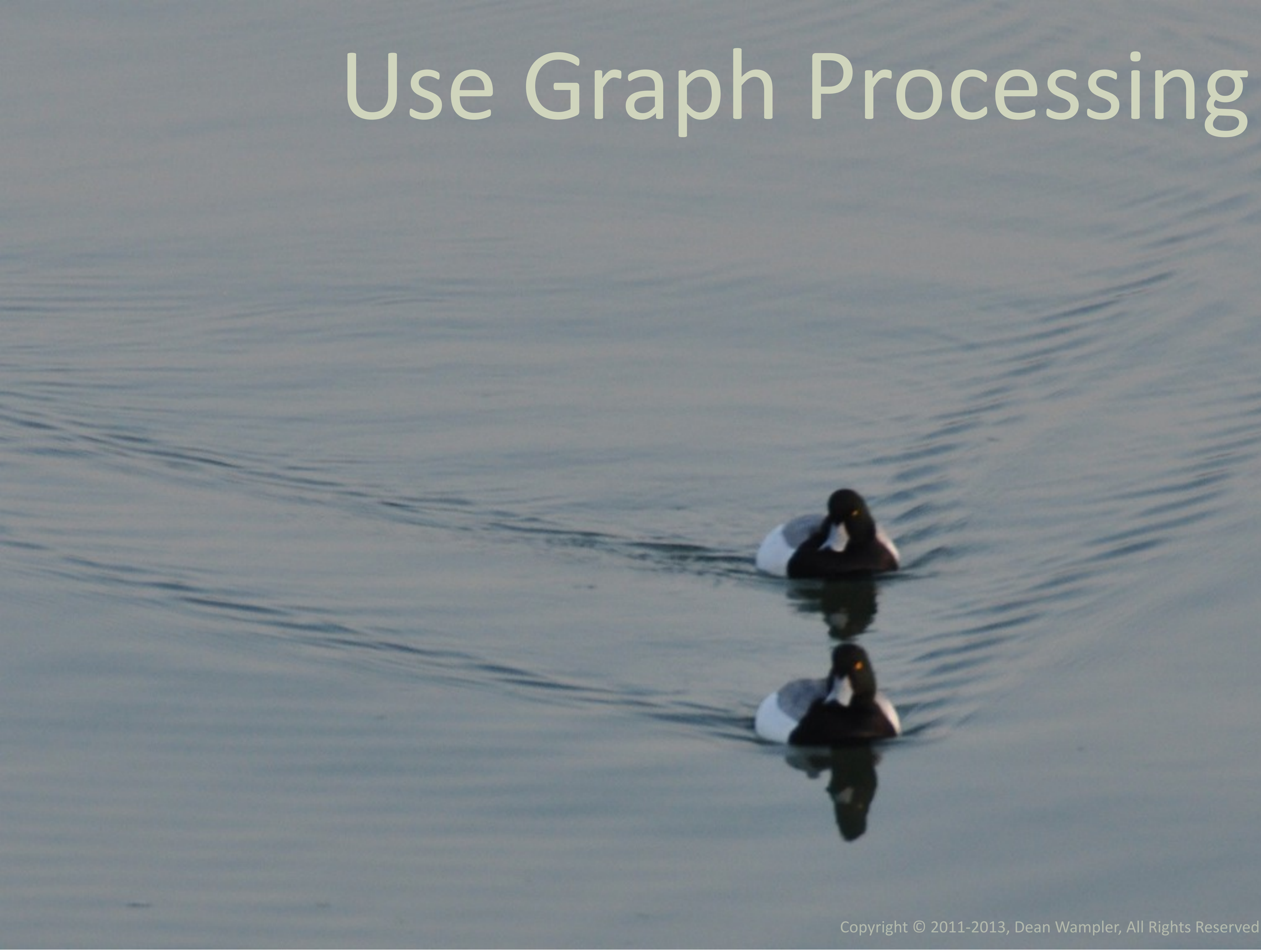
Why not MapReduce?

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



Tuesday, October 1, 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...



Tuesday, October 1, 13

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

Photo: San Francisco Bay

Google's Pregel

- Pregel: New graph framework for Page Rank.
  - Bulk, Synchronous Parallel (BSP).
    - Graphs are first-class citizens.
    - Efficiently processes updates...

Tuesday, October 1, 13

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 extensive commercial support.

Open-source Alternatives

- Neo4J.
  - Mature, single machine graphs.

Tuesday, October 1, 13

http://neo4j.org

Neo4J

- Not the same kind of distributed system like Pregel, but
  - More mature.
  - Commercially supported.
  - Maybe you don't need distributed?

Tuesday, October 1, 13

http://neo4j.org

Tuesday, October 1, 13

Some thoughts about where the industry is and where it's headed.

Hadoop works for batch-mode analytics.

Tuesday, October 1, 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...


NoSQL scales well, fits non-relational problems.
Copyright © 2011-2013, Dean Wampler, All Rights Reserved

Purpose-built Tools for special Scenarios

Tuesday, October 1, 13

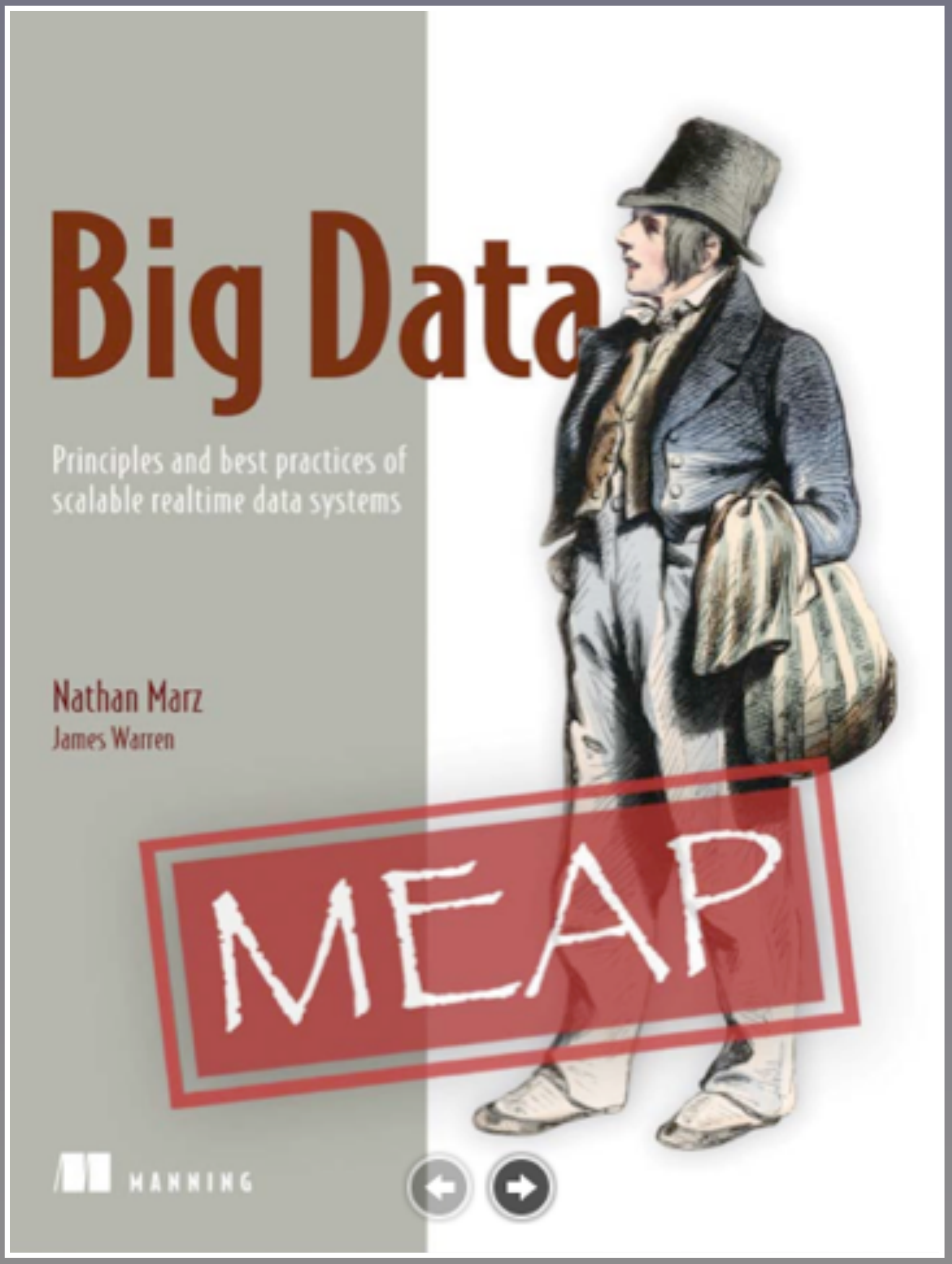
I see that a trend where completely generic tooling is giving way to more “purpose-built” tooling, e.g., search and event processing.

Tuesday, October 1, 13

Now that we have cataloged some issues and solutions, let’s recap and look forward.
Photo: Grebe and distorted post reflection.

“Big Data”

Nathan Marz’s vision for data systems...



Tuesday, October 1, 13

- Use Hadoop for batch-mode processing of very large data sets.
- Use Storm for event handling, e.g., increment updates.
- Use NoSQL for flexible, scalable storage.
- Stir...

Tuesday, October 1, 13

NoSQL solves meets lots of requirements better than traditional RDBMSs, but people loves them some SQL!!

Don't overlook SQL

- It's entrenched.
- Organizations want a SQL solution if they can have it.

Hadoop owes a lot of its popularity to Hive!

Tuesday, October 1, 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 query languages (e.g., Cassandra, MongoDB).

Tuesday, October 1, 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.

A Final Emerging Trend...

Tuesday, October 1, 13

Both are examples of technologies focused on particular problems.
photo: Trump hotel and residences on the Chicago River.

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.

Photo: Same sunrise, in Burlingame.