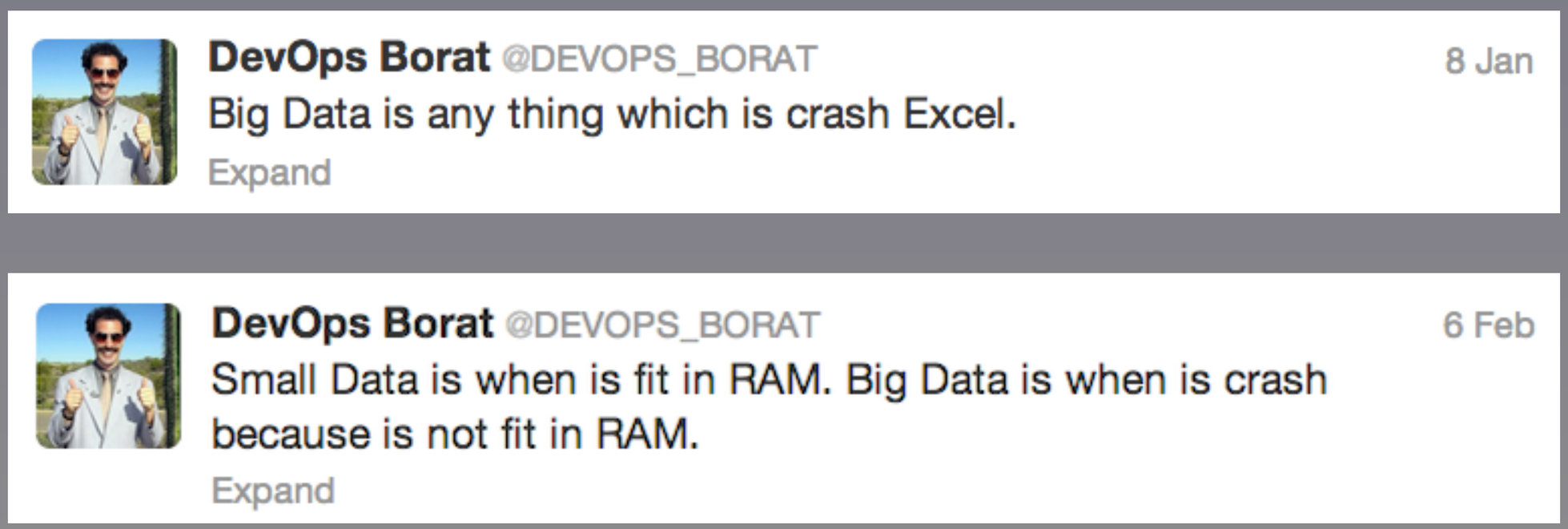
Copious Data: The "Killer App" for Functional Programming

Detroit Tech Watch
March 8, 2022
dean@deanwampler.com
@deanwampler
polyglotprogramming.com/talks

What Is Big … err… "Copious" Data?



Copious Data

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

Hat tip: Bob Korbus

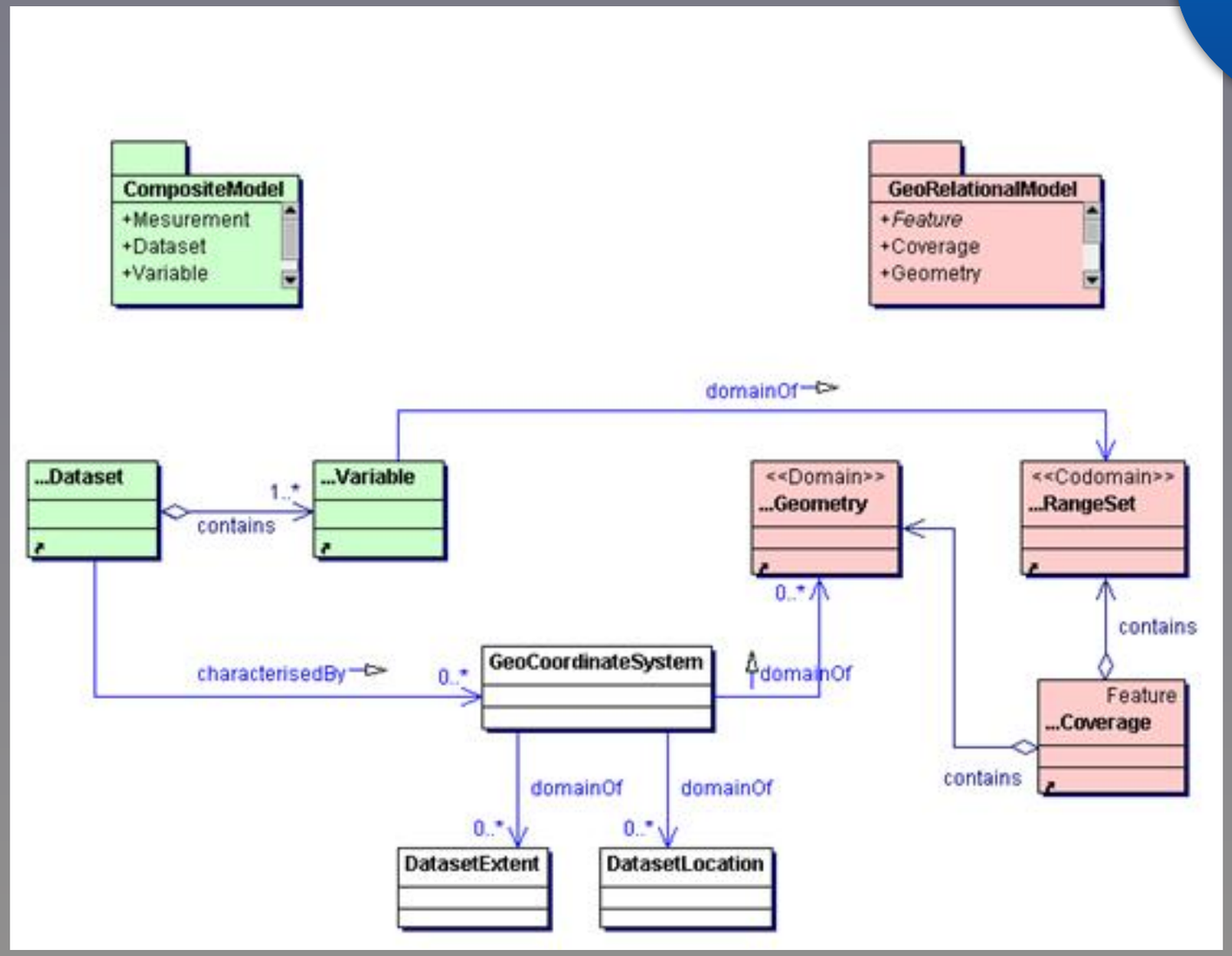
3 Trends

Data Size ⬆

2022 update:
Or is it?

Formal Schemas ⬇

2022 update:
Or are they?



Data-Driven Programs ⬆

2022 update:
Still true!

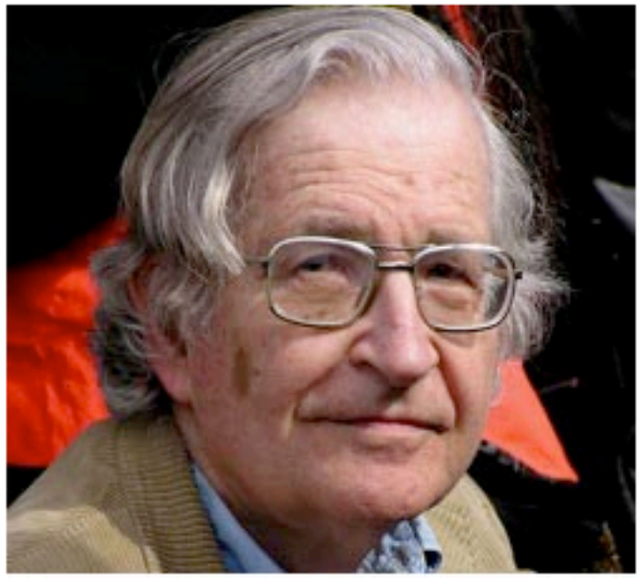
Probabilistic Models vs. Formal Grammars

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

What is MapReduce?

Hadoop is the dominant copious data platform today.

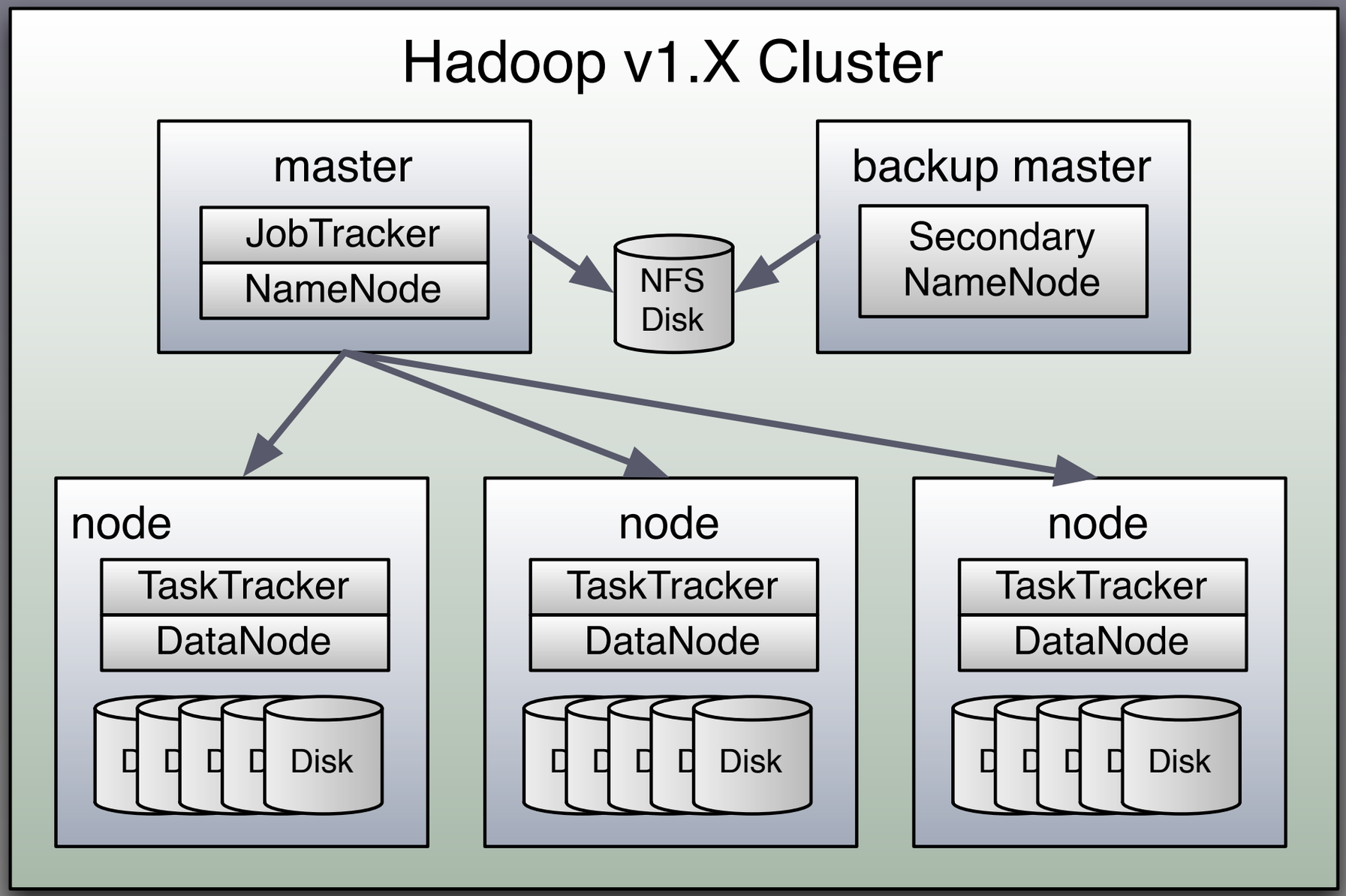
2022 update

was

Hadoop ~~is~~ the dominant copious data platform ~~today~~.

then

A Hadoop Cluster

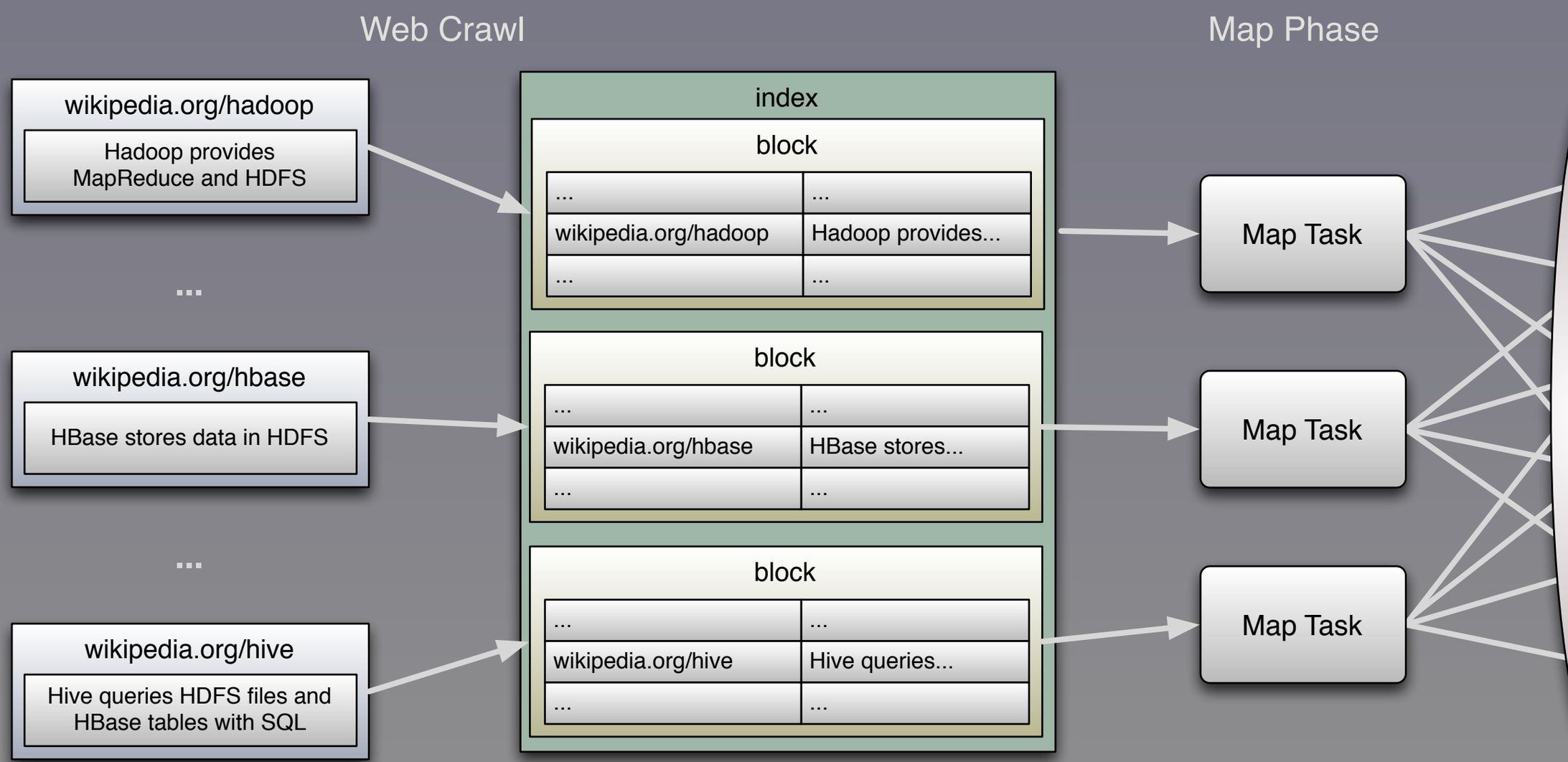


MapReduce in Hadoop

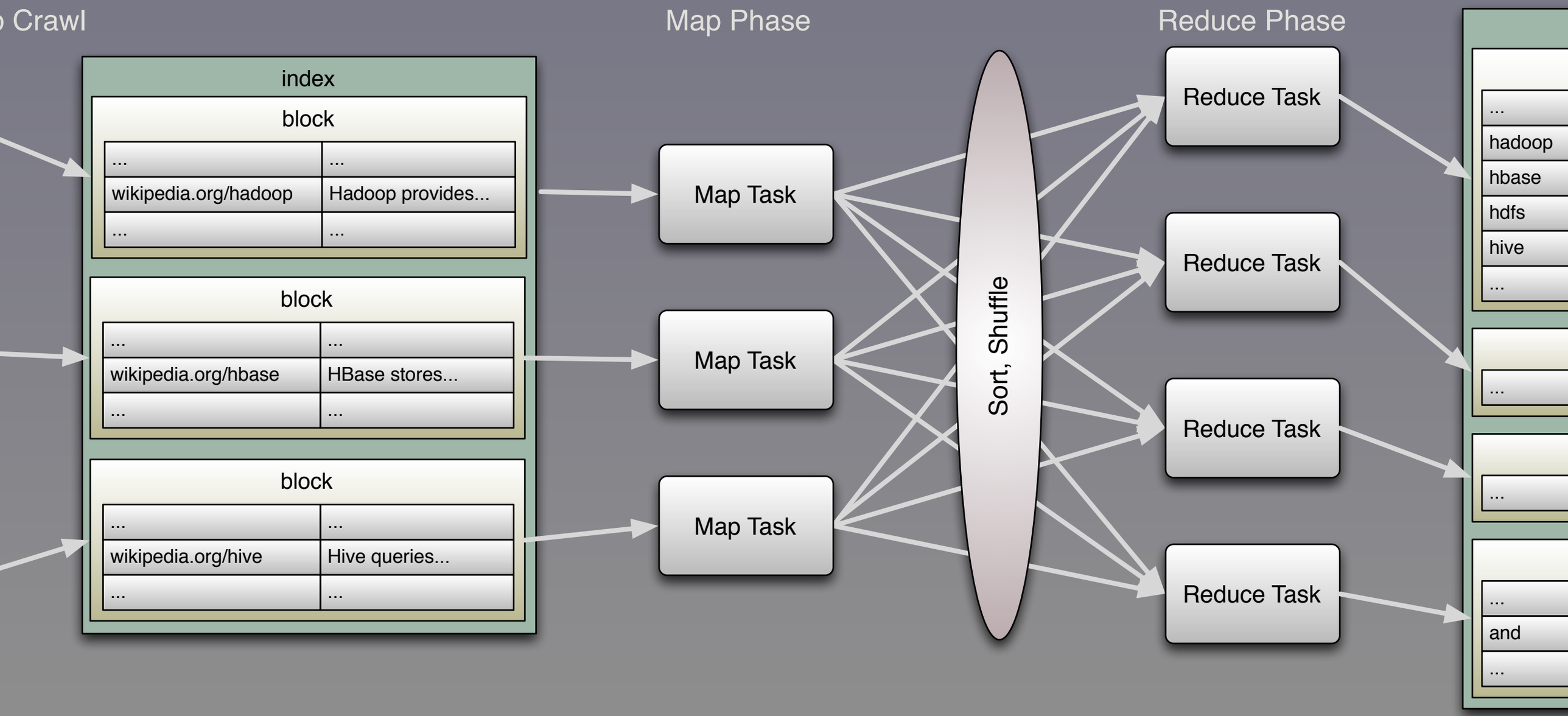
Let's look at a MapReduce algorithm: Inverted Index.

Used for text/web search.

Crawl teh Interwebs



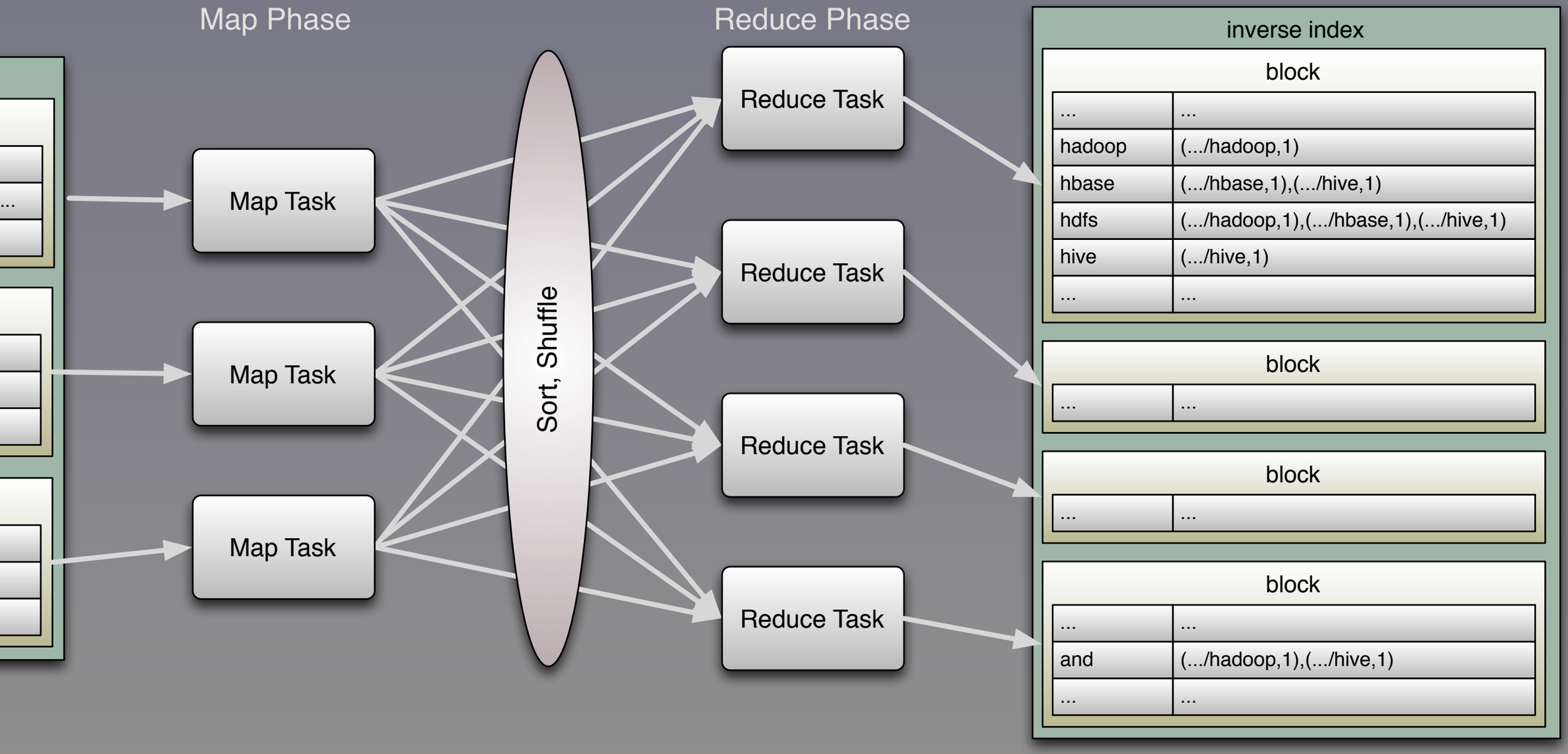
Compute Inverse Index



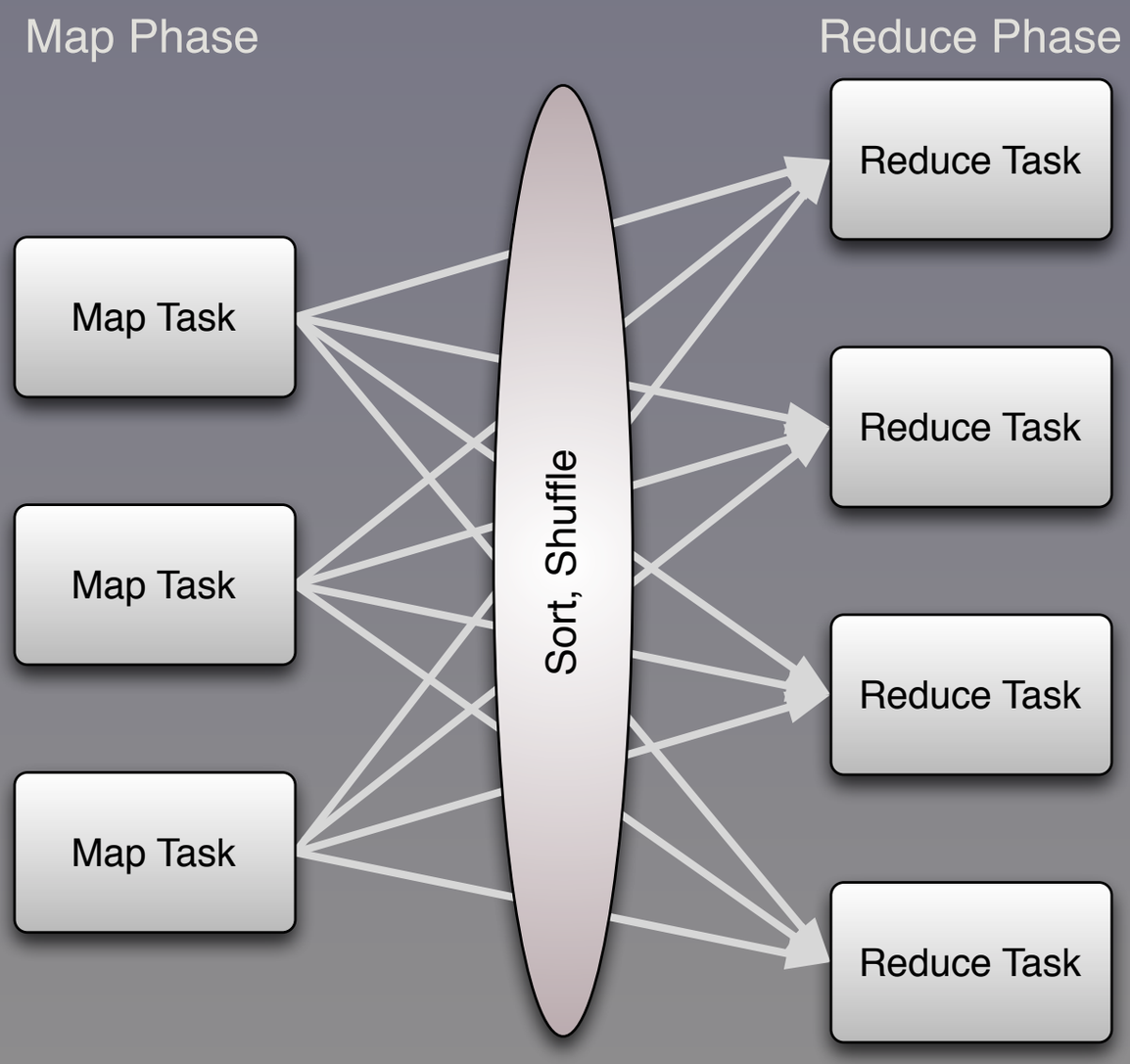
Compute Inverse Index

Key-values output
by first map task

Compute Inverse Index



Anatomy: MapReduce Job



Map (or *Flatmap*):

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

Reduce:

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

MapReduce and Its Discontents

It's hard to implement many algorithms in MapReduce.

MapReduce is very course-grained.

1-Map, 1-Reduce phase...

Multiple MR jobs required for some algorithms.

Each one flushes its results to disk!

MapReduce is designed for offline, batch-mode analytics.

High latency; not suitable for event processing.

The Hadoop Java API is hard to use.

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

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

```
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));
  }
}
```

```
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

```
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 ‘90s called. They want their EJBs back!

Use Cascalog (Clojure)

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

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

Datalog-style queries

Use Spark

```
import org.apache.spark.SparkContext

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

Also small and concise!

Spark replaced MapReduce:

2022 update: Much faster now!

- Distributed computing with in-memory caching.
- ~30x faster than MapReduce (in part due to caching of intermediate data).

Spark replaced MapReduce:

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

Use SQL when you can!

- Hive: SQL on top of MapReduce.
- Spark SQL: high perf. Spark API.
- Impala & Presto: HiveQL with new, faster back ends.

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;
```

... and similarly for the other SQL tools.

We're in the era where

The SQL Strikes Back!

(with apologies to George Lucas...)

Combinators

Why were the Scala, Clojure, and SQL solutions so concise and appealing??

Data problems are fundamentally Mathematics!

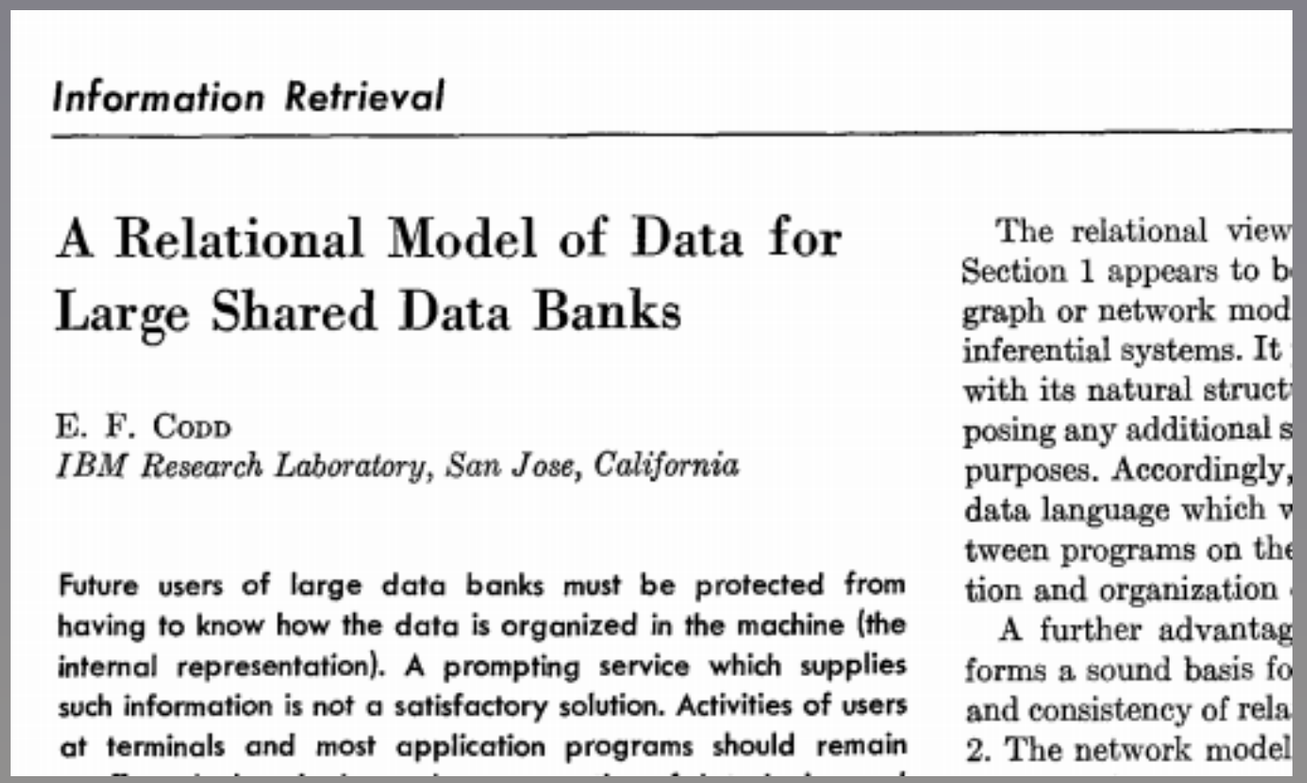
evanmiller.org/mathematical-hacker.html

Combinators

- Functions that are side-effect free.
- They get all their information from their inputs and write all their work to their outputs.

Set Theory and First-Order Logic

- Relational Model.
- Data organized into tuples, grouped by relations.



Information Retrieval

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain

The relational view
Section 1 appears to b
graph or network mod
inferential systems. It
with its natural struct
posing any additional s
purposes. Accordingly,
data language which w
tween programs on the
tion and organization
A further advantag
forms a sound basis fo
and consistency of rela
2. The network model

http://dl.acm.org/citation.cfm?doid=362384.362685

Let's look at
a few relational operators
and the corresponding
functional combinators.

Recall our Word Counts:

```
CREATE TABLE word_counts (
  word   CHARACTER(64),
  count  INTEGER);
```

(ANSI SQL syntax)

```
val word_counts: Stream[(String,Int)]
```

(Scala)

Restrict

```
SELECT * FROM word_counts
WHERE word = 'Chicago';
```

vs.

```
word_counts.filter {
  case (word, count) =>
    word == "Chicago"
}
```

Project

```
SELECT word FROM word_counts;
```

vs.

```
word_counts.map {
  case (word, count) =>
    word
}
```

Group By

```
SELECT count, size(word) AS size
FROM word_counts
GROUP BY count
ORDER BY size DESC;
```

vs.

```
word_counts.groupBy {
  case (word, count) => count
}.toList.map {
  case (count, words) => (count, words.size)
}.sortBy {
 case (count, size) => -size
}
```

Example

```
scala> val word_counts = List(
("a", 1), ("b", 2), ("c", 3),
("d", 2), ("e", 2), ("f", 3))

scala> val out = word_counts.groupBy {
  case (word, count) => count
}.toList.map {
  case (count, words) => (count, words.size)
}.sortBy {
 case (count, size) => -size
}

out: List[(Int,Int)] = List((2,3), (3,2), (1,1))
```

We could go on, but you get the point. Declarative, functional combinators are a natural tool for data.

SQL vs. FP

- SQL
  - Optimized for data operations.
- FP
  - Turing complete.
  - More combinators.
  - First class functions!

FP to the Rescue!

Popular Claim:

Multicore concurrency is driving FP adoption.

My Claim:

Data will drive the next wave of widespread FP adoption.

2022 update: Mostly true, in terms of # of developers…

2022 Postscript

Hadoop and *Data Lakes (swamps?)* are passé.

2022 Postscript

SQL is triumphant (again) for most data, because *structured data* is what most people want.

2022 Postscript

NoSQL databases are still used, but more cautiously.

2022 Postscript

SQL has driven adoption of *Spark SQL + Delta Lakes* and new data warehouses like *Snowflake*.

2022 Postscript

ML/AI is still a home for *(semi|uns)tructured data*.

2022 Postscript

Like SQL, *Python* is ascendant again for *ML/AI*, even though it is less "functional" than *Scala, Clojure, etc.*

2022 Postscript

Still for *data engineering*, like *ETL pipelines*, *Scala* is still very popular.

Questions?

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

Bonus Slides

Other branches of Mathematics that are very useful for Software

Category Theory

- Monads - Structure.
- Abstracting over collections.
- Control flow and mutability containment.

Category Theory

- Monoids, Groups, Rings, etc.
- Abstracting over addition, subtraction, multiplication, and division.

Monoid: Addition

- (a + b) + (c + d) for some a, b, c, d.
- "Add All the Things", Avi Bryant, StrangeLoop 2013.

infoq.com/presentations/abstract-algebra-analytics

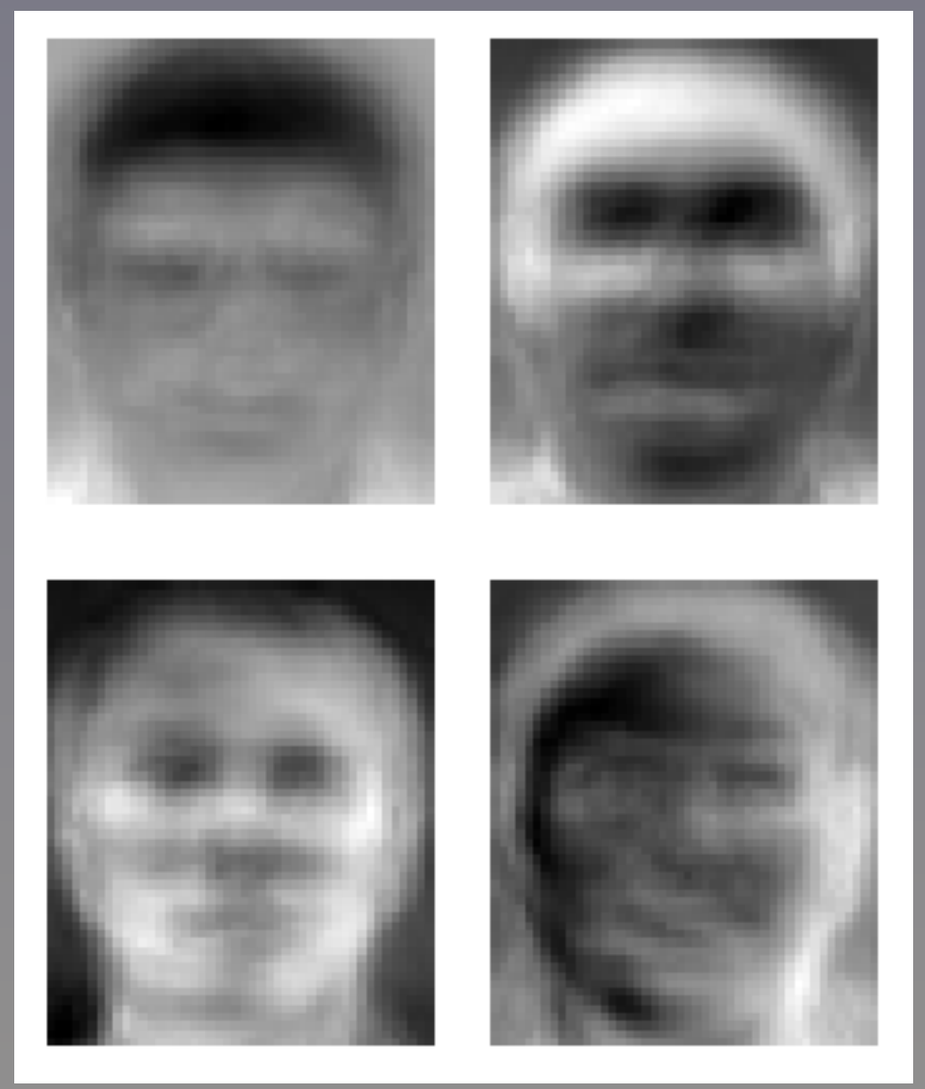
Linear Algebra

- Eigenvector and Singular Value Decomposition.
- Essential tools in machine learning.

$$Av = \lambda v$$

Example: Eigenfaces

- Represent images as vectors.
- Solve for "modes".
- Top N modes approx. faces!



http://en.wikipedia.org/wiki/File:Eigenfaces.png

Join

```
CREATE TABLE dictionary (
  word       CHARACTER(64),
  definition CHARACTER(256));
```

Table for join examples.

Join - SQL

```
SELECT w.word, d.definition
FROM   word_counts AS w
       dictionary AS d
WHERE  w.word = d.word;
```

Join

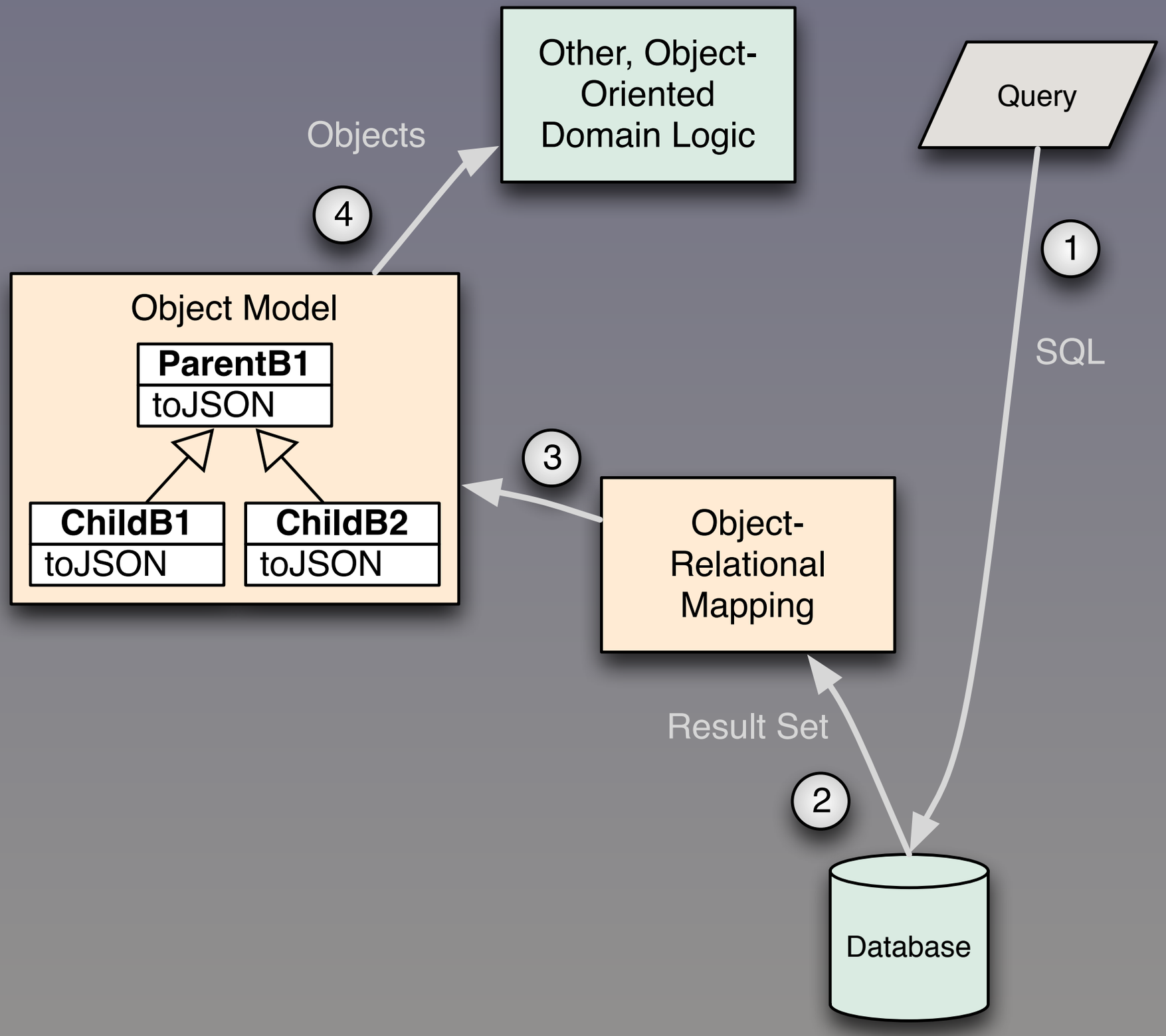
```
SELECT w.word, d.definition
FROM   word_counts AS w
       dictionary AS d
WHERE  w.word = d.word;
```

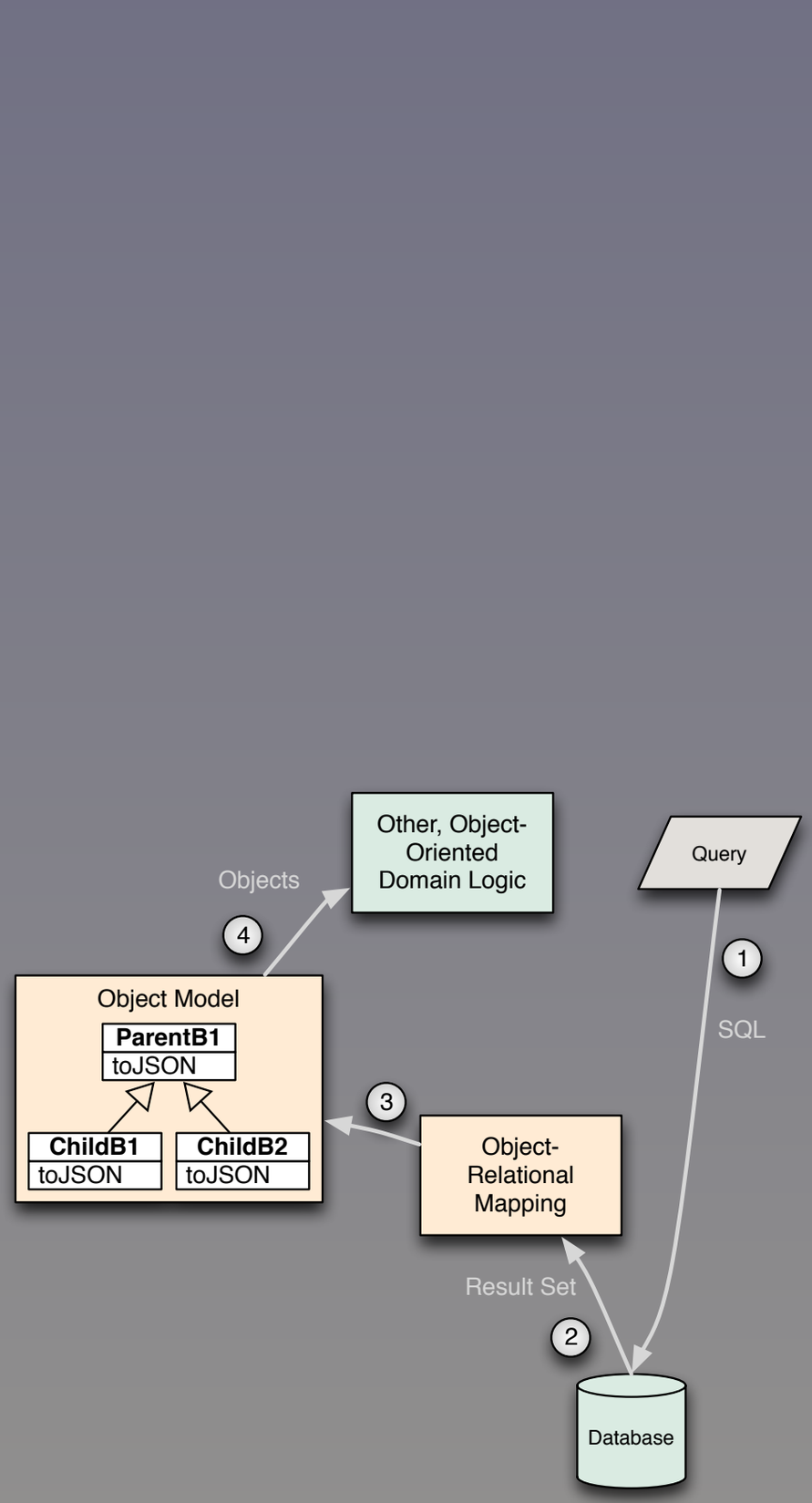
vs.

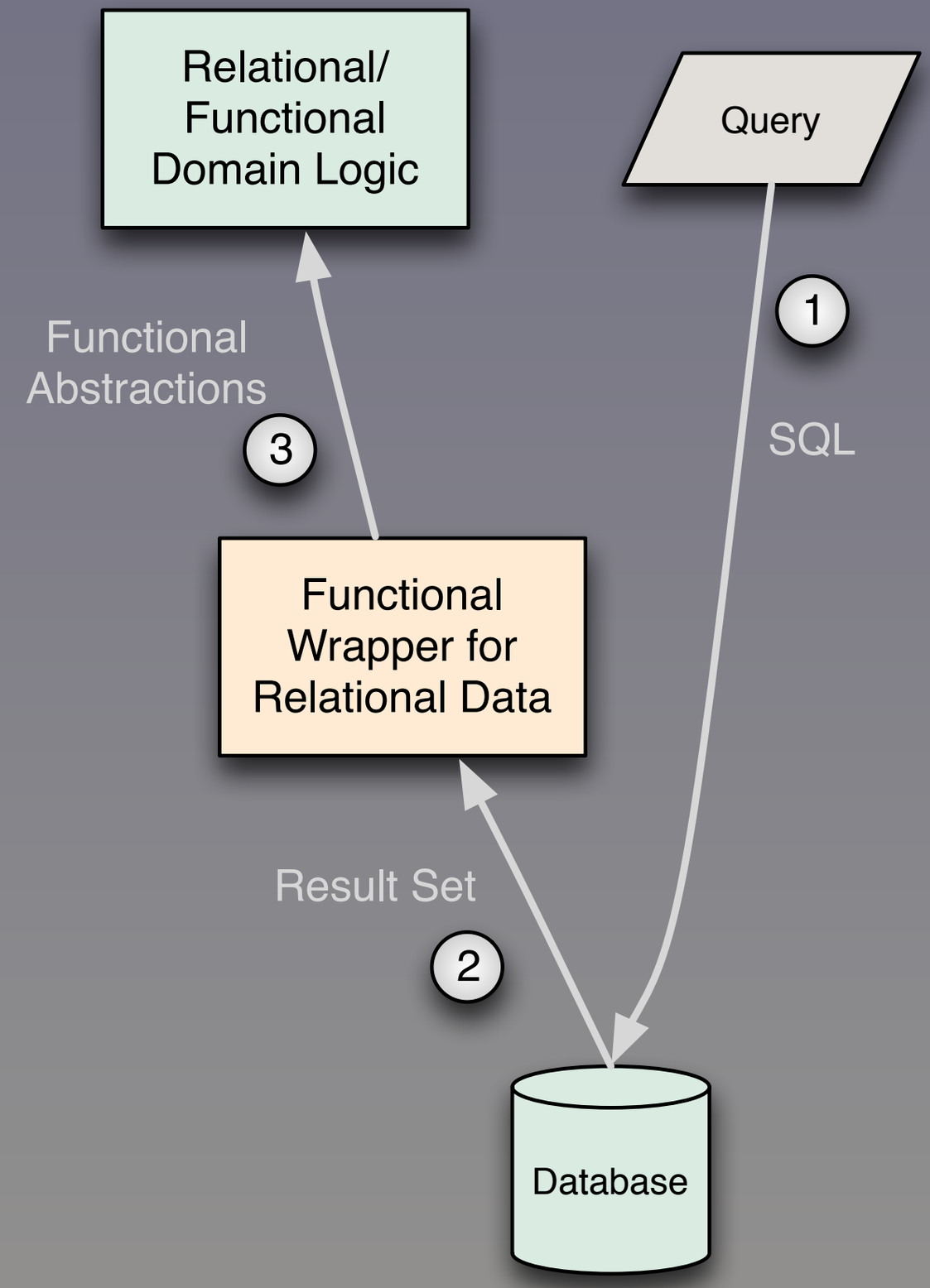
```
...
word_counts
  .joinWithLarger('wword -> 'dword,
    dictionary)
  .project('wword, 'definition)
```

Joins are expensive.
Your data system needs to exploit optimizations.

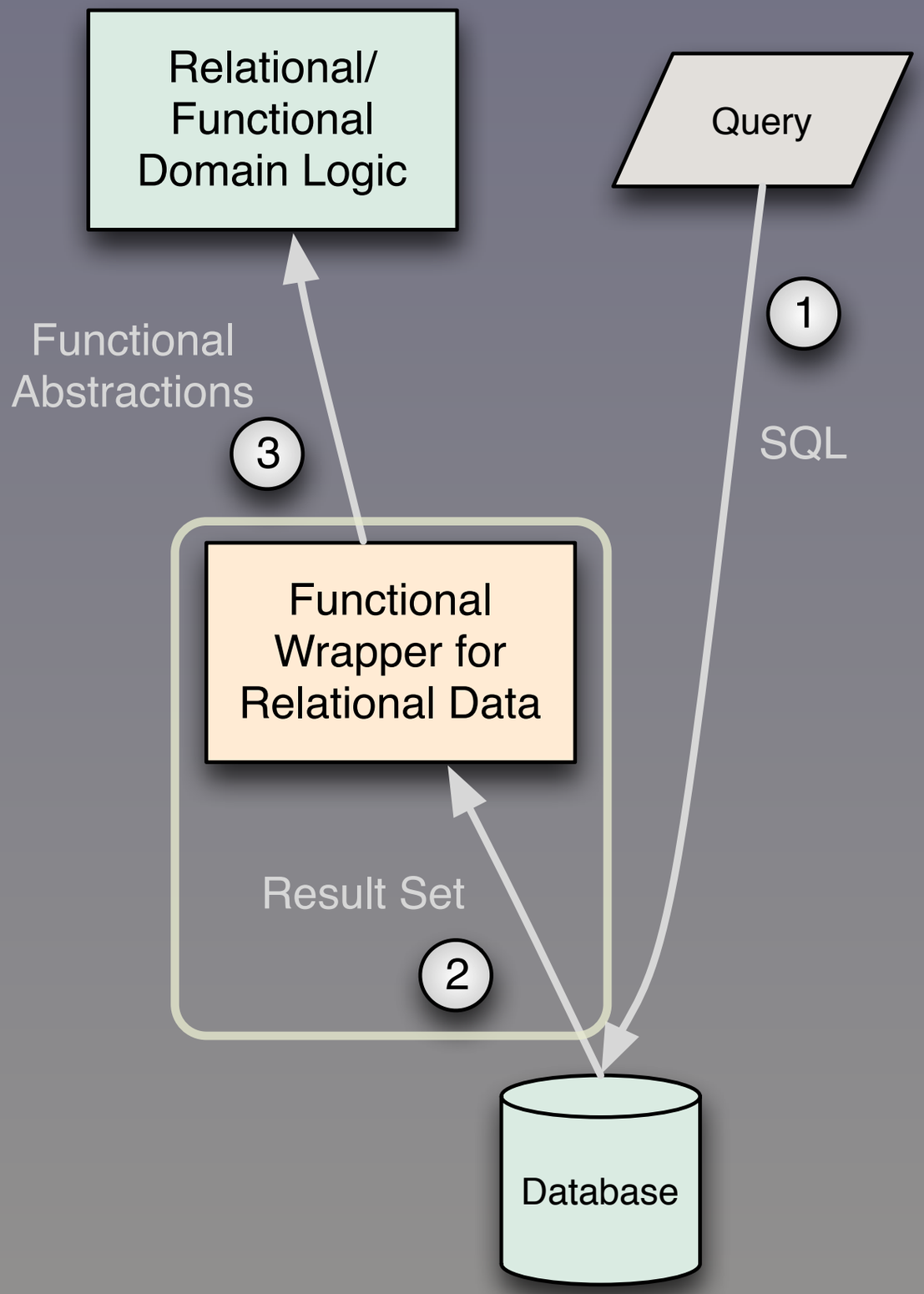
Data Architectures

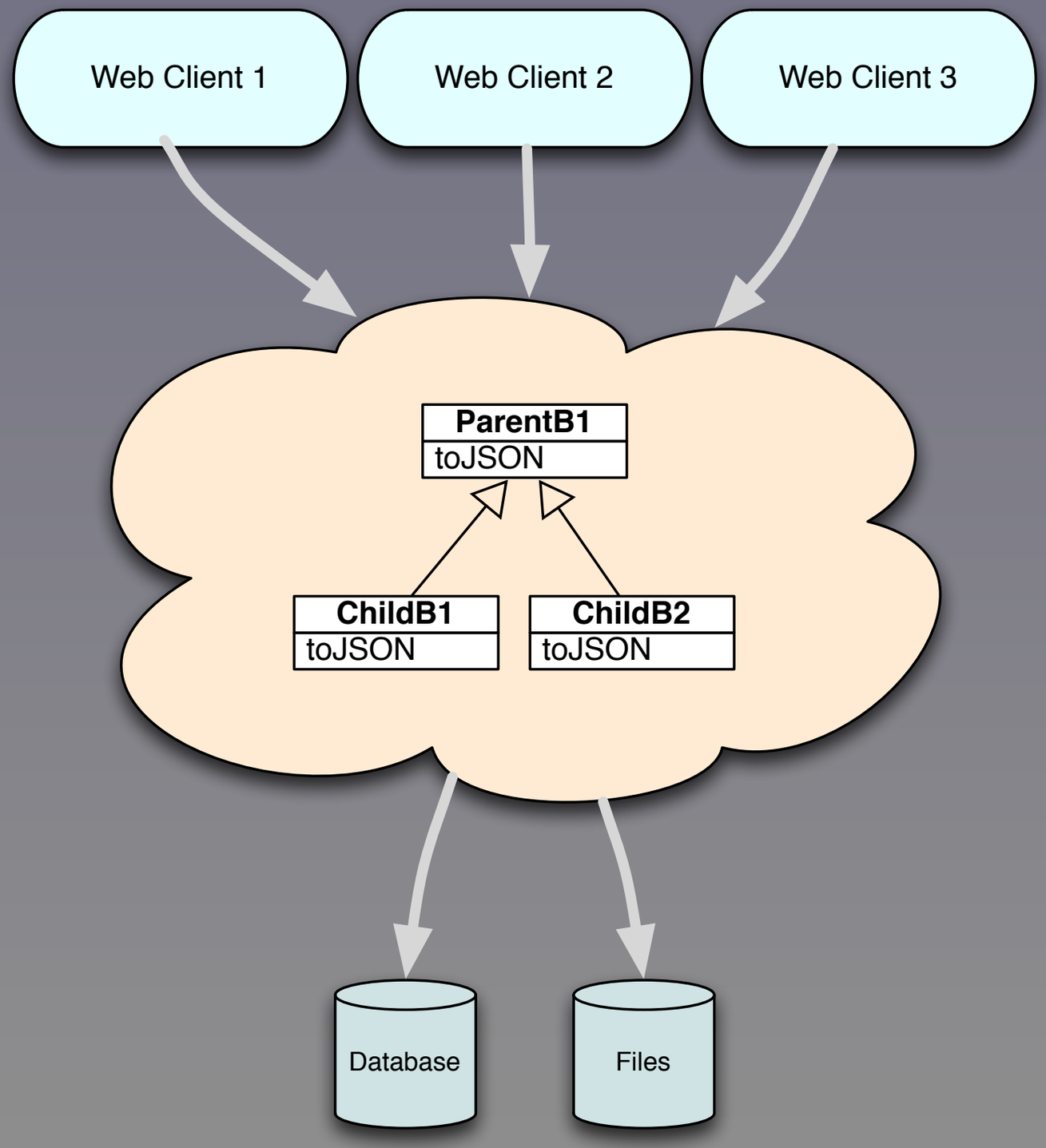


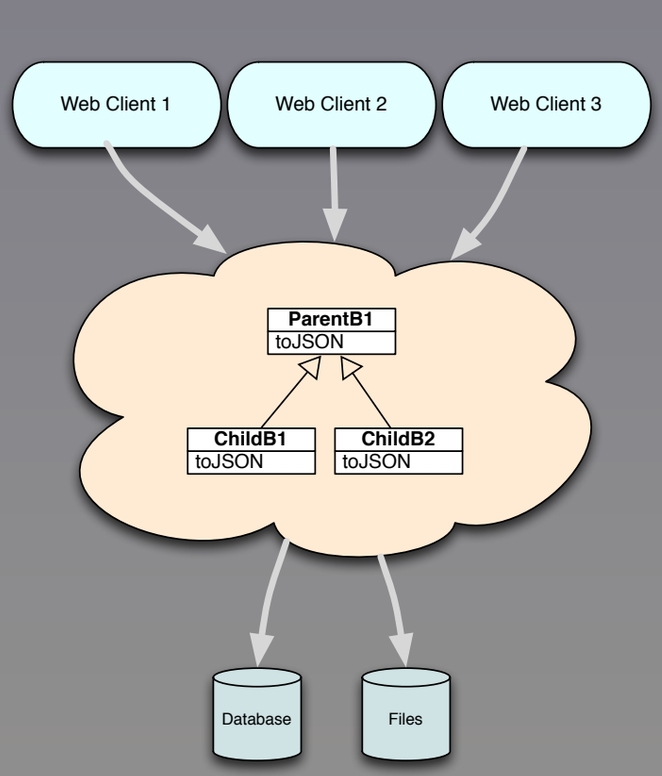


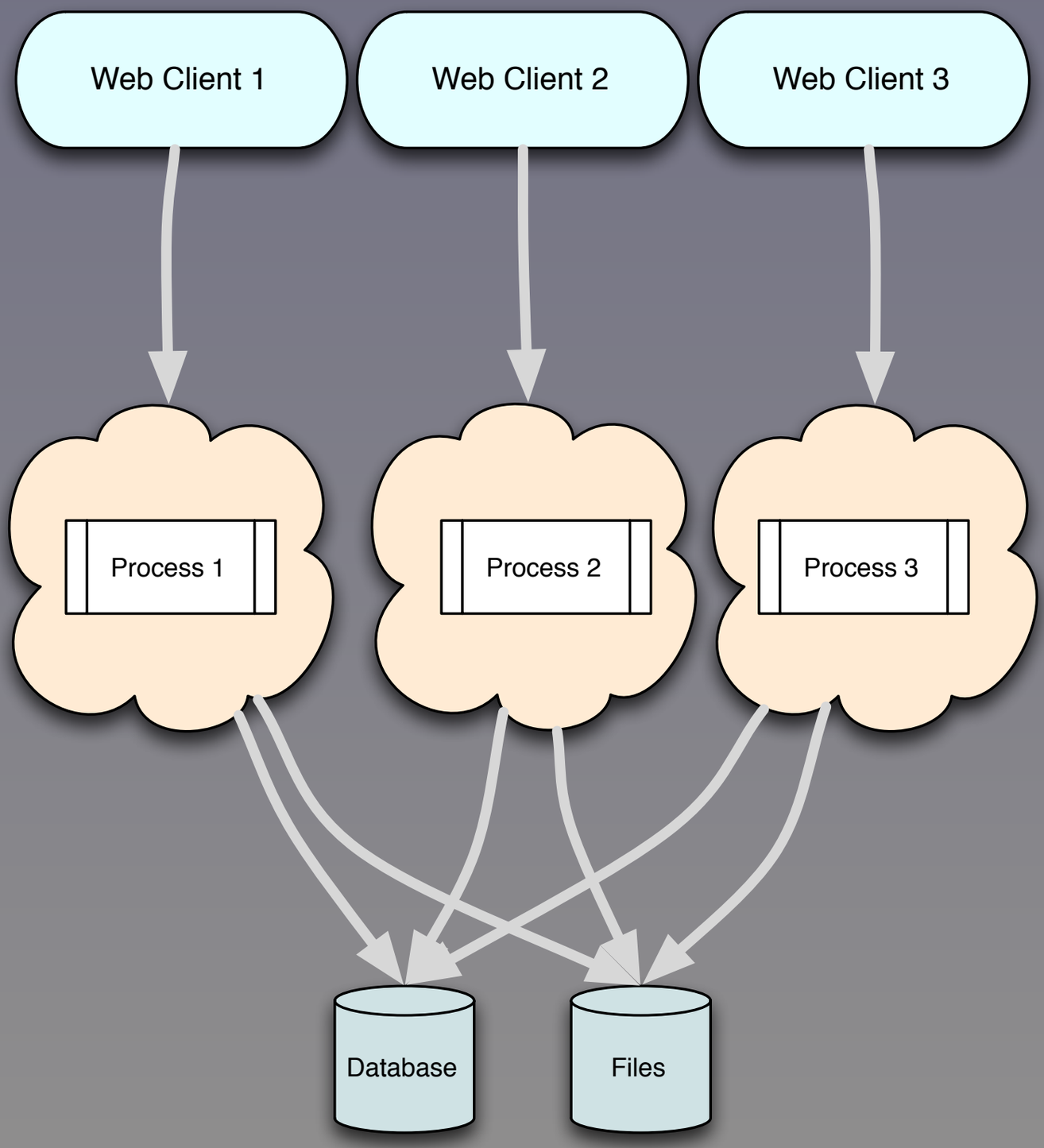


- Focus on:
  - Lists
  - Maps
  - Sets
  - Trees
  - ...









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