From Distributed Systems to Data Science

William C. Benton
Red Hat Emerging Technology

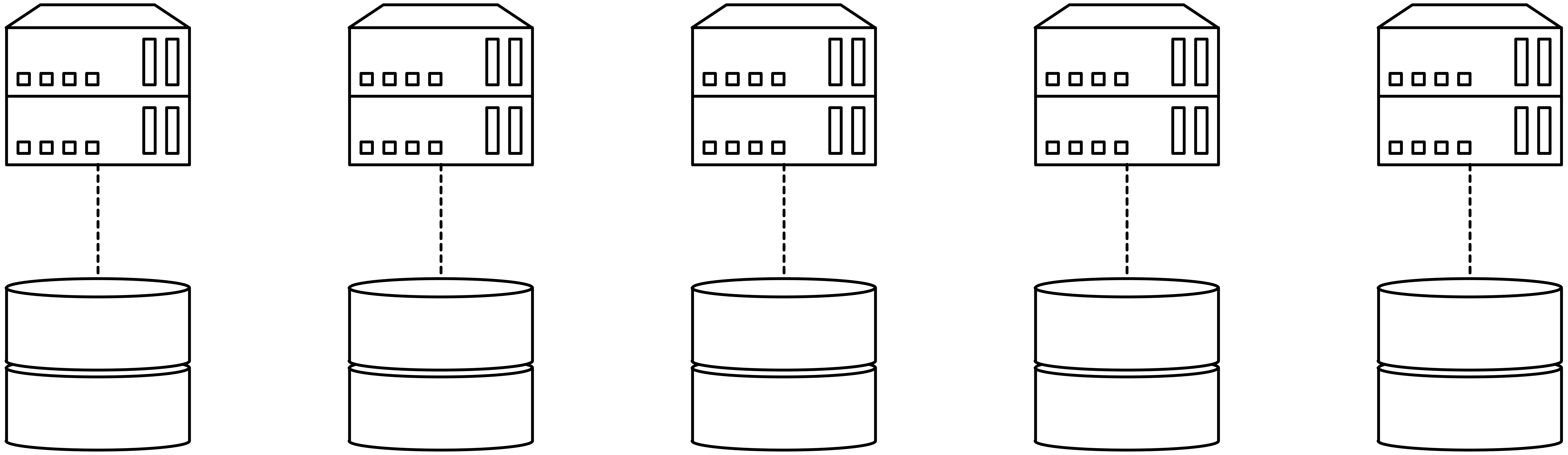
About me

At Red Hat: scheduling, configuration management, RPC, Fedora, data engineering, data science.

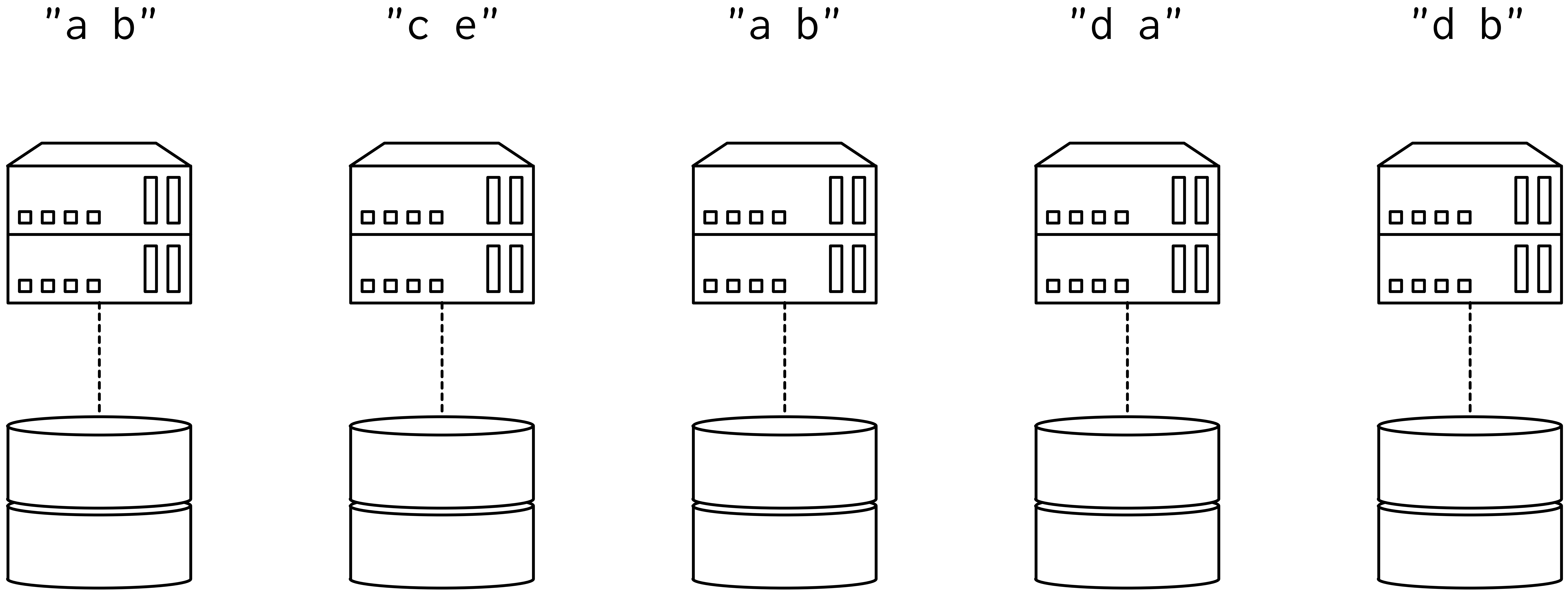
Before Red Hat: programming language research.

HISTORY and MYTHOLOGY

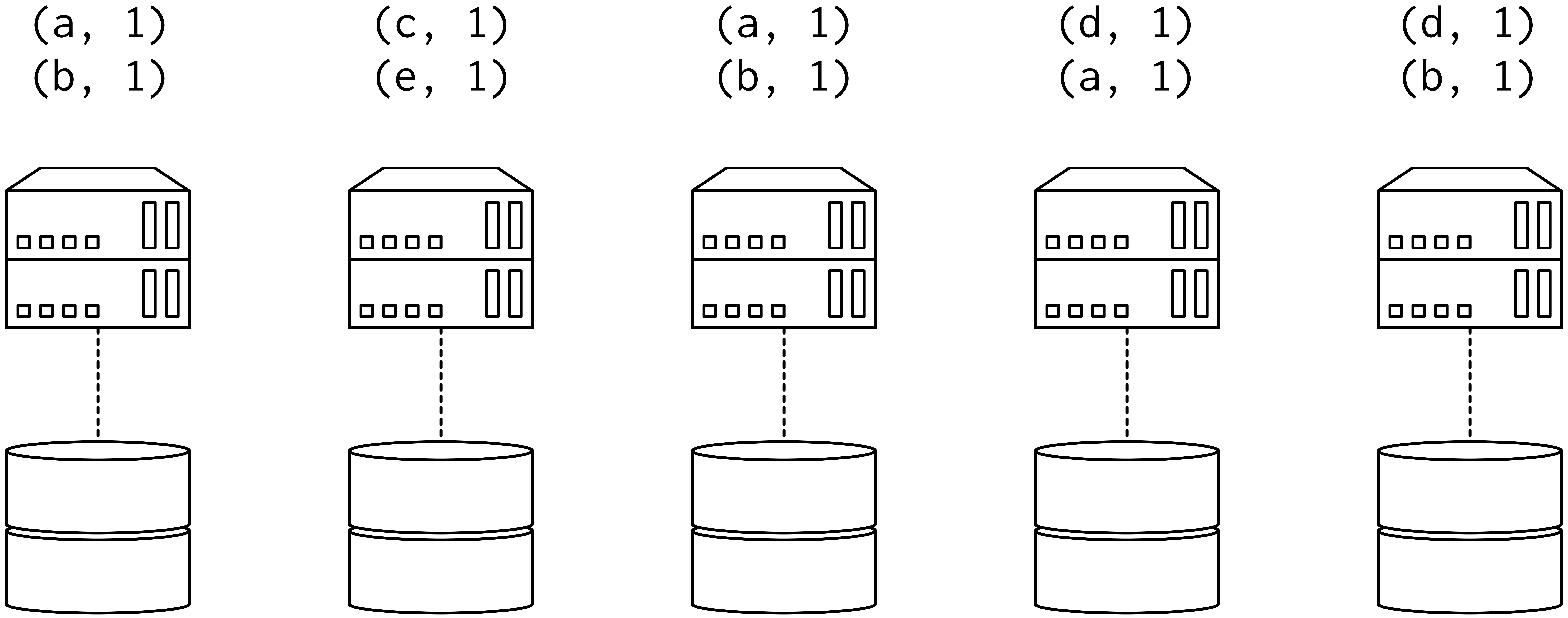
MapReduce (2004)



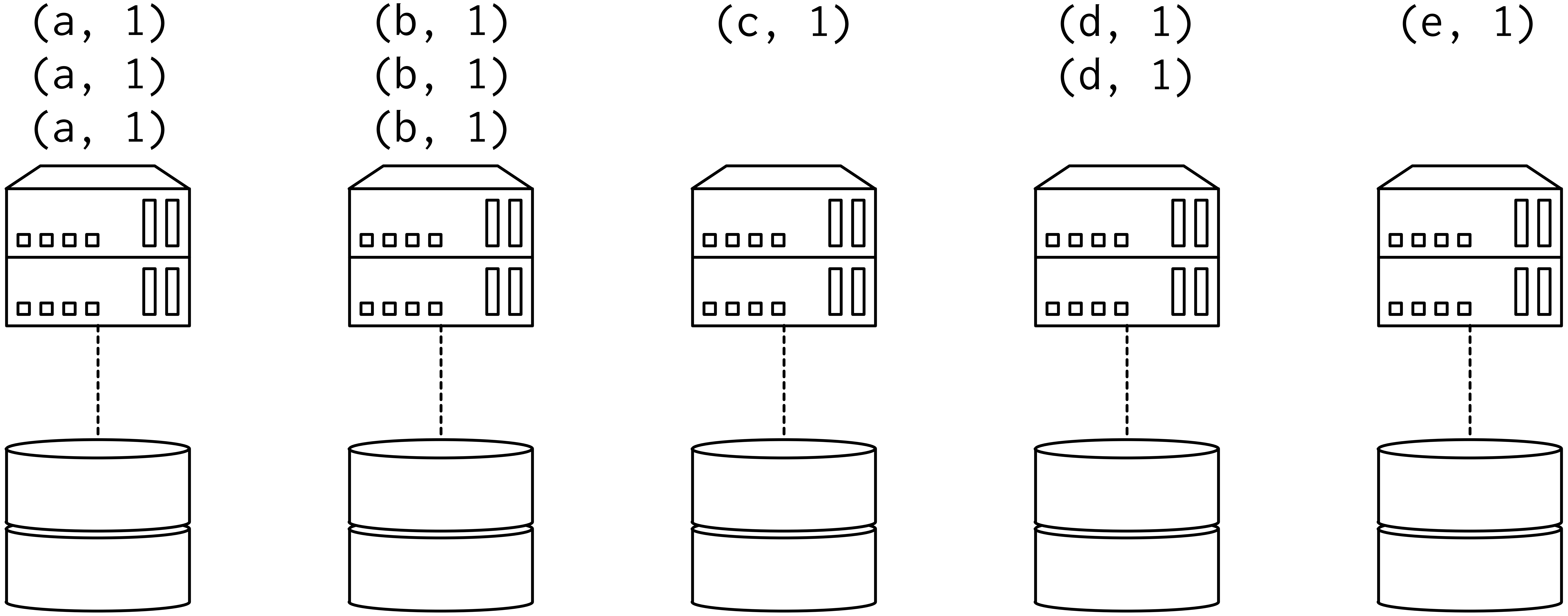
MapReduce (2004)



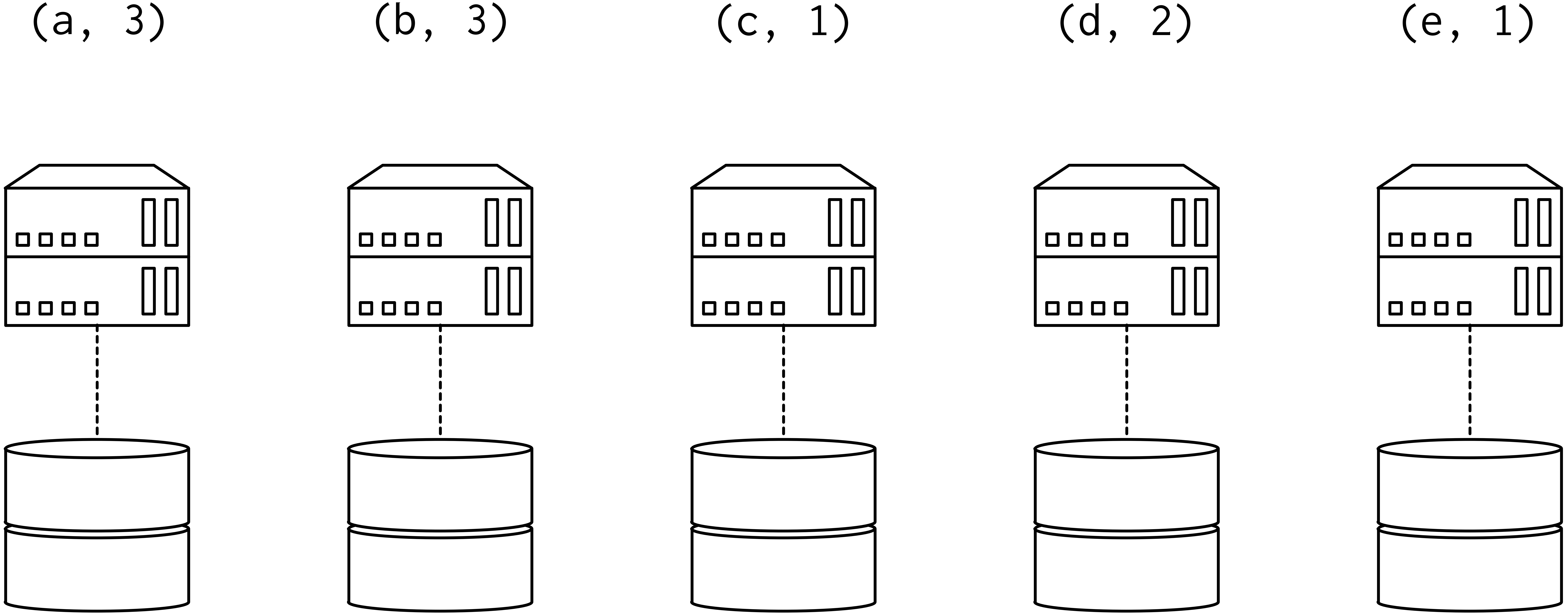
MapReduce (2004)



MapReduce (2004)



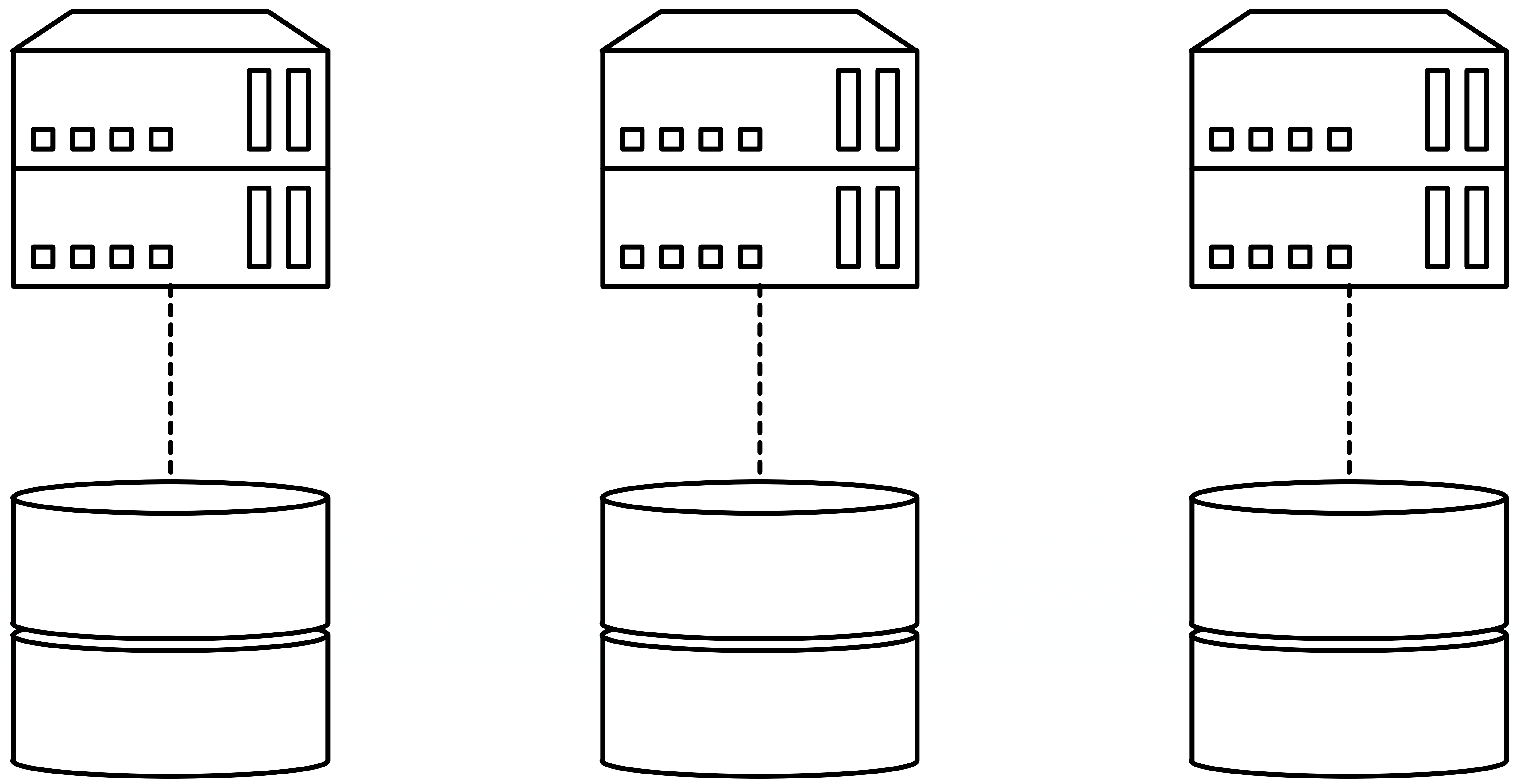
MapReduce (2004)



Hadoop (2005)

Hadoop (2005)

HDFS



Hadoop (2005)

Hadoop
MapReduce

Hadoop (2005)

Hadoop MapReduce

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

 public static class Map extends Mapper<LongWritable, Text, Text,
IntWritable> {
```

Hadoop (2005)

```
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

 public static class Map extends Mapper<LongWritable, Text, Text,
IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context
context) throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
 }

 public static class Reduce extends Reducer<Text, IntWritable,
```

Hadoop MapReduce

Hadoop (2005)

```
 }

 public static class Reduce extends Reducer<Text, IntWritable,
Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values,
Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
 }

 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
```

Hadoop MapReduce

Hadoop (2005)

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
}
}
```

Hadoop MapReduce

“Facts” (motivated by Hadoop)

Data federation is **the biggest problem**; solve it with distributed storage.

You need to be able to **scale out** to many nodes to handle real-world data analytics problems.

Your network and disks **probably aren’t fast enough.**

“…at least two analytics production clusters (at Microsoft and Yahoo!) have median job input sizes under 14 GB and 90% of jobs on a Facebook cluster have input sizes under 100gb.”

Appuswamy et al., “Nobody ever got fired for buying a cluster.” (Microsoft Research Technical Report)

“Contrary to our expectations … CPU (and not I/O) is often the bottleneck [and] improving network performance can improve job completion time by a median of at most 2%.”

Ousterhout et al., “Making Sense of Performance in Data Analytics Frameworks.” (USENIX NSDI ’15)

“Facts,” revised for reality

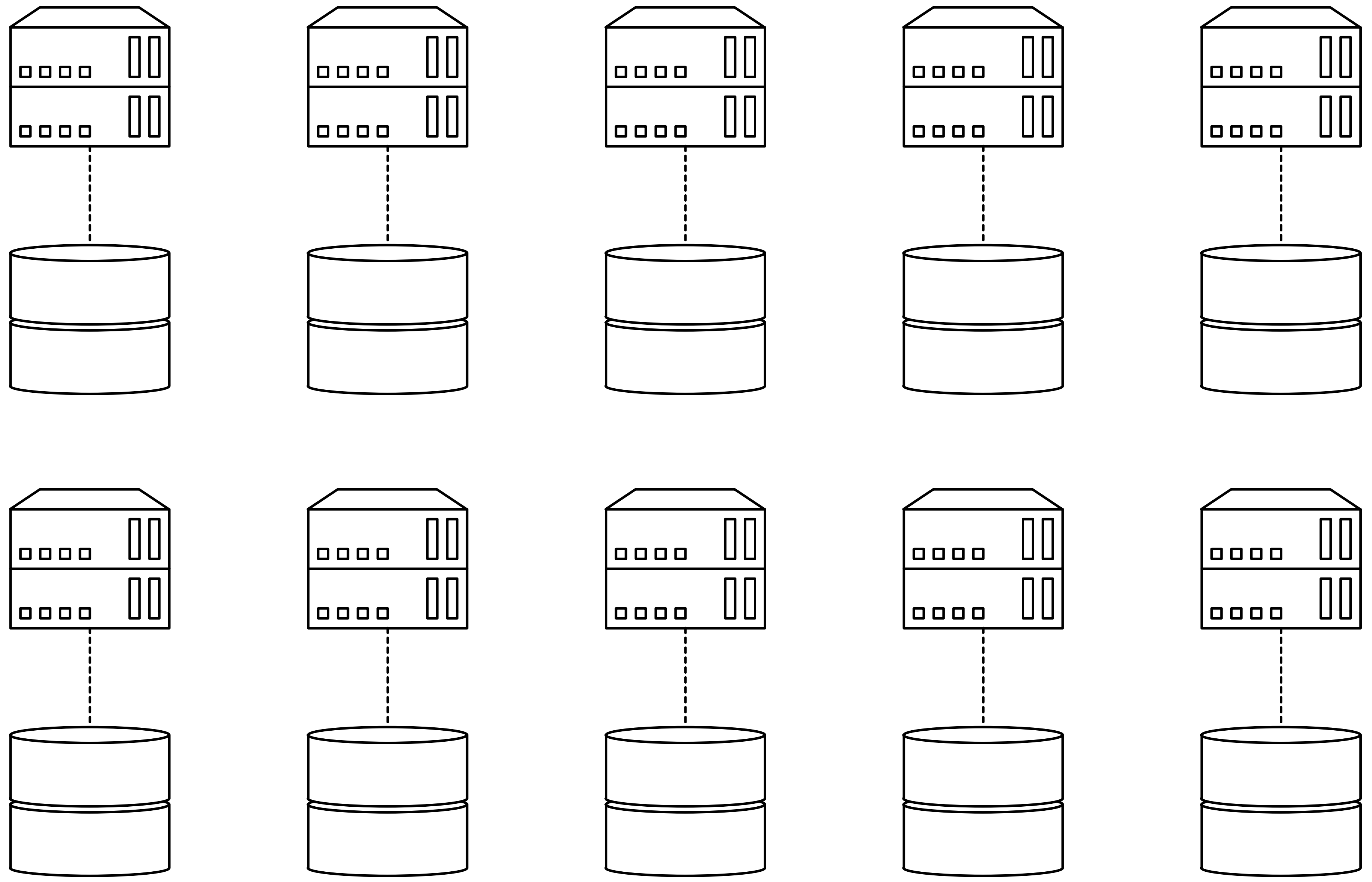
Data federation is absolutely a problem, but you have **enormous flexibility** as to **where you solve it.**

Most **real-world analytics workloads** will fit **in memory.**

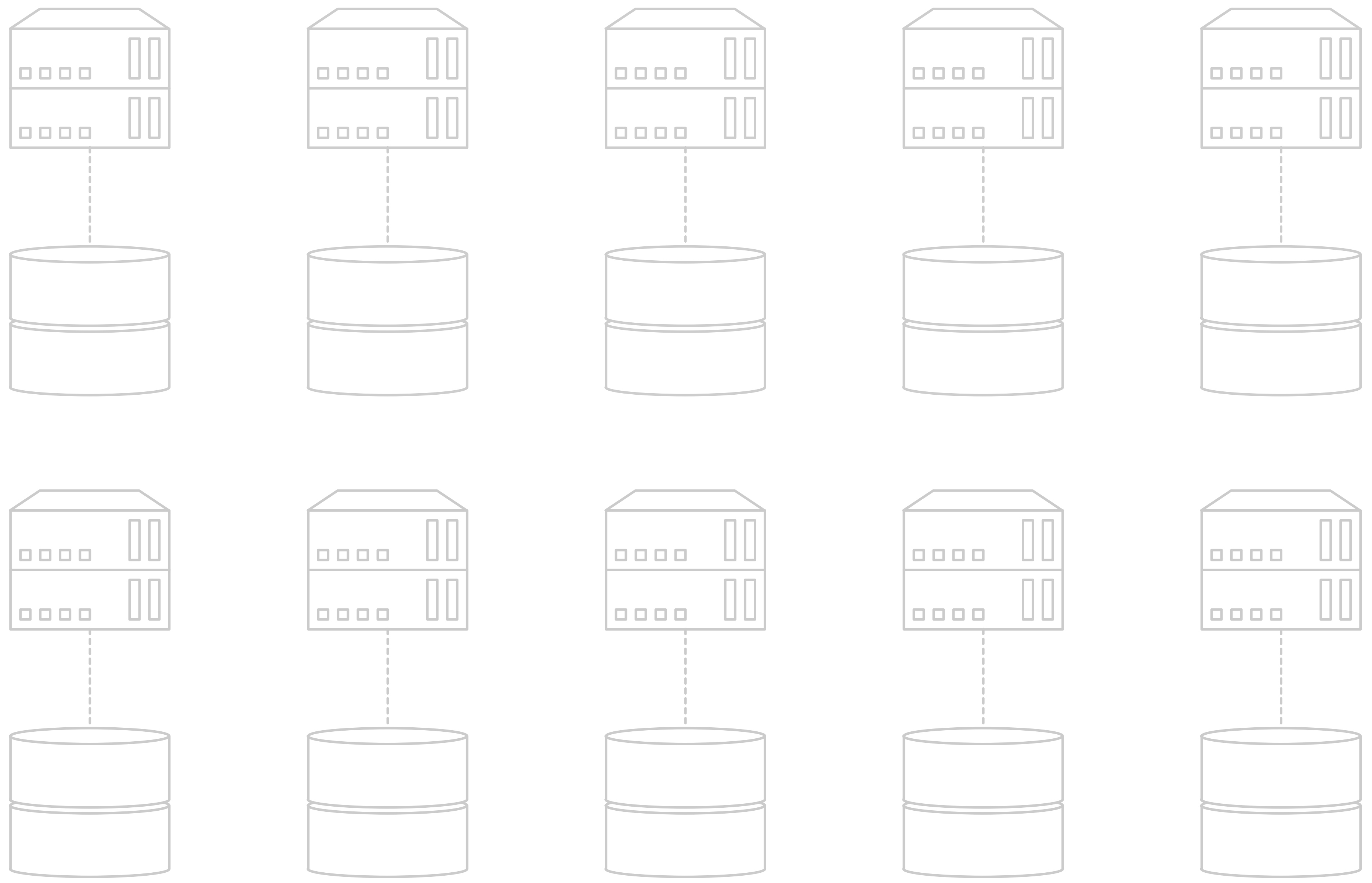
Your network and disks **probably aren’t the problem.**

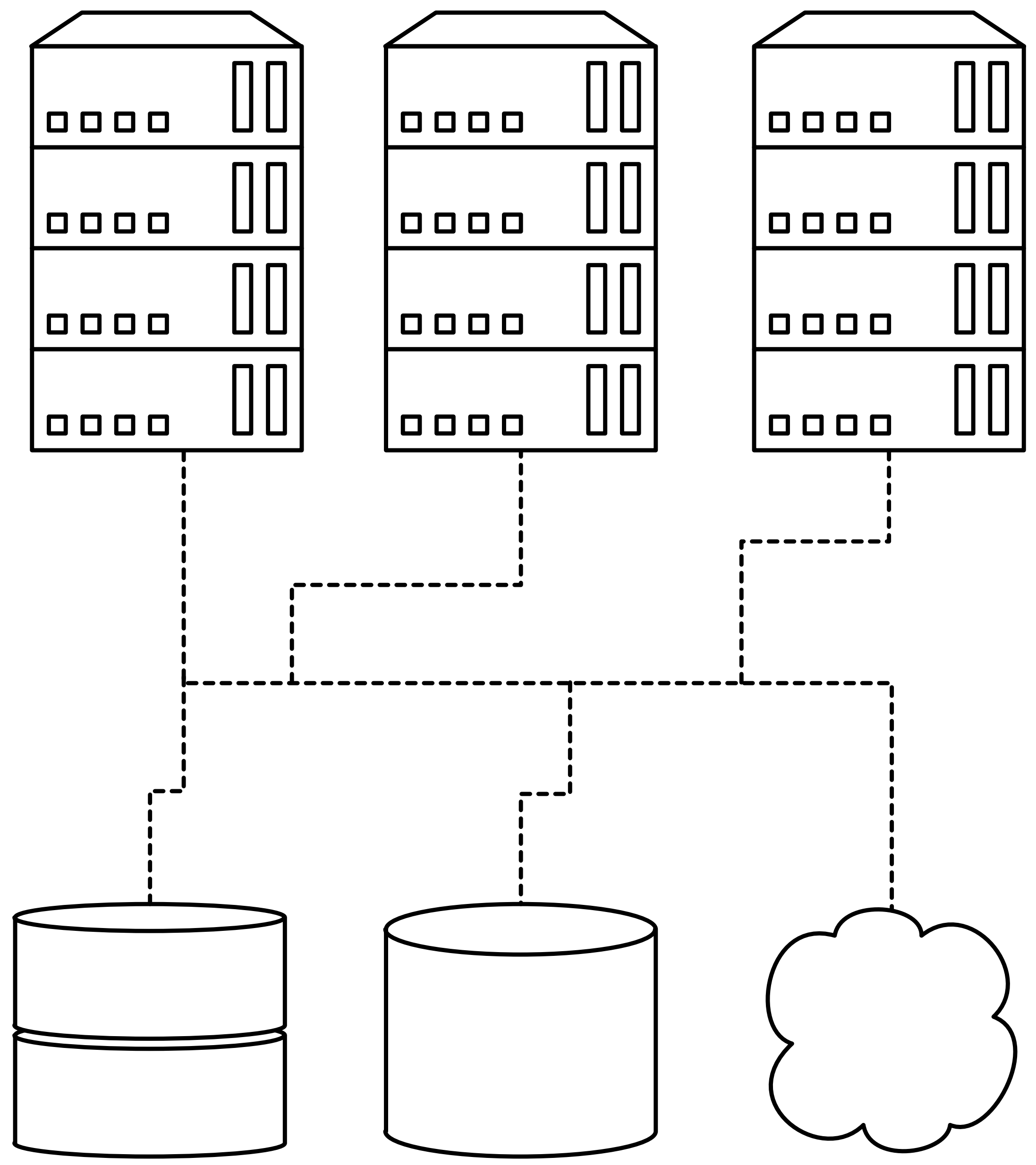
DATA PROCESSING with ELASTIC RESOURCES

Instead of...

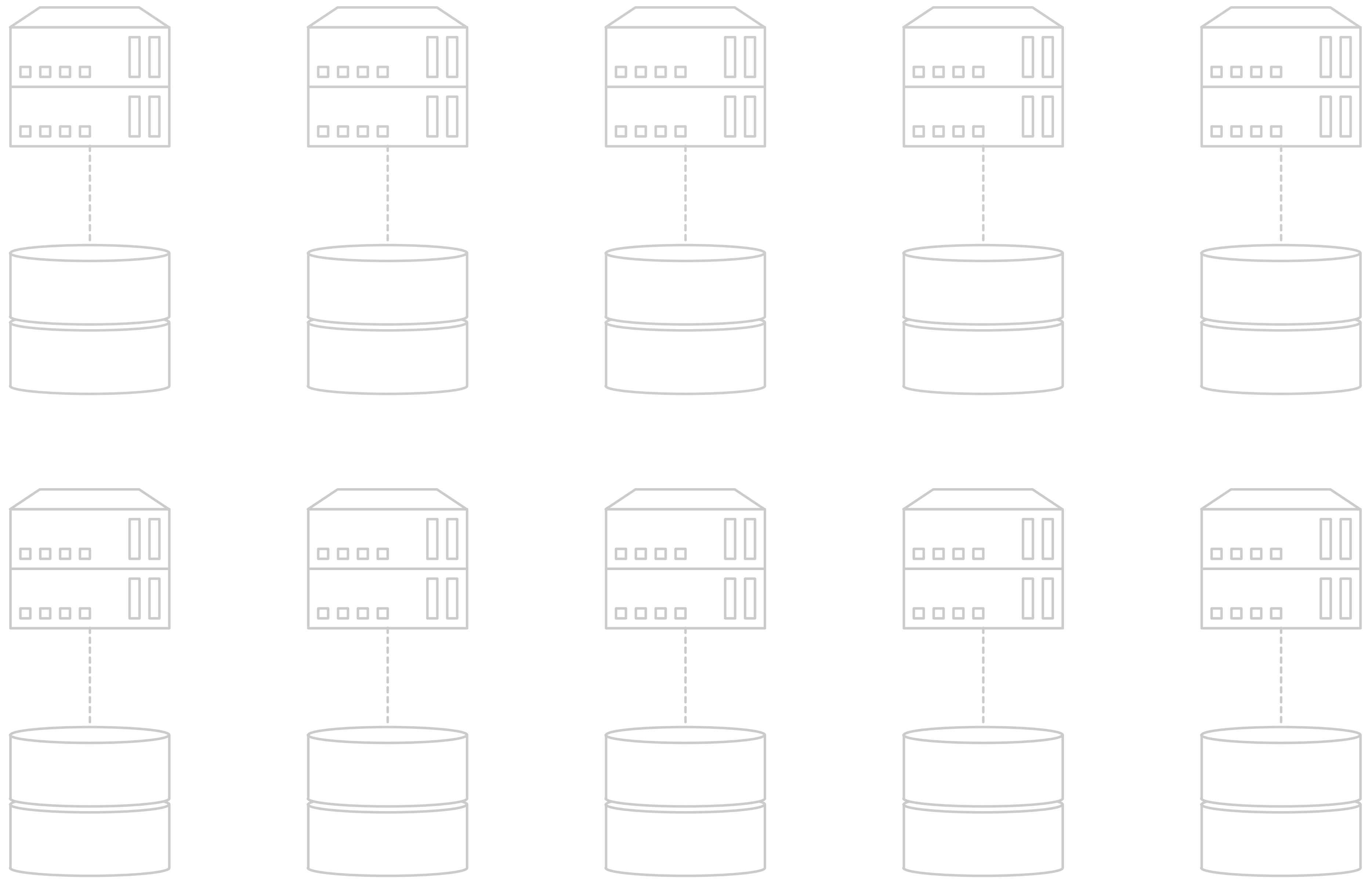


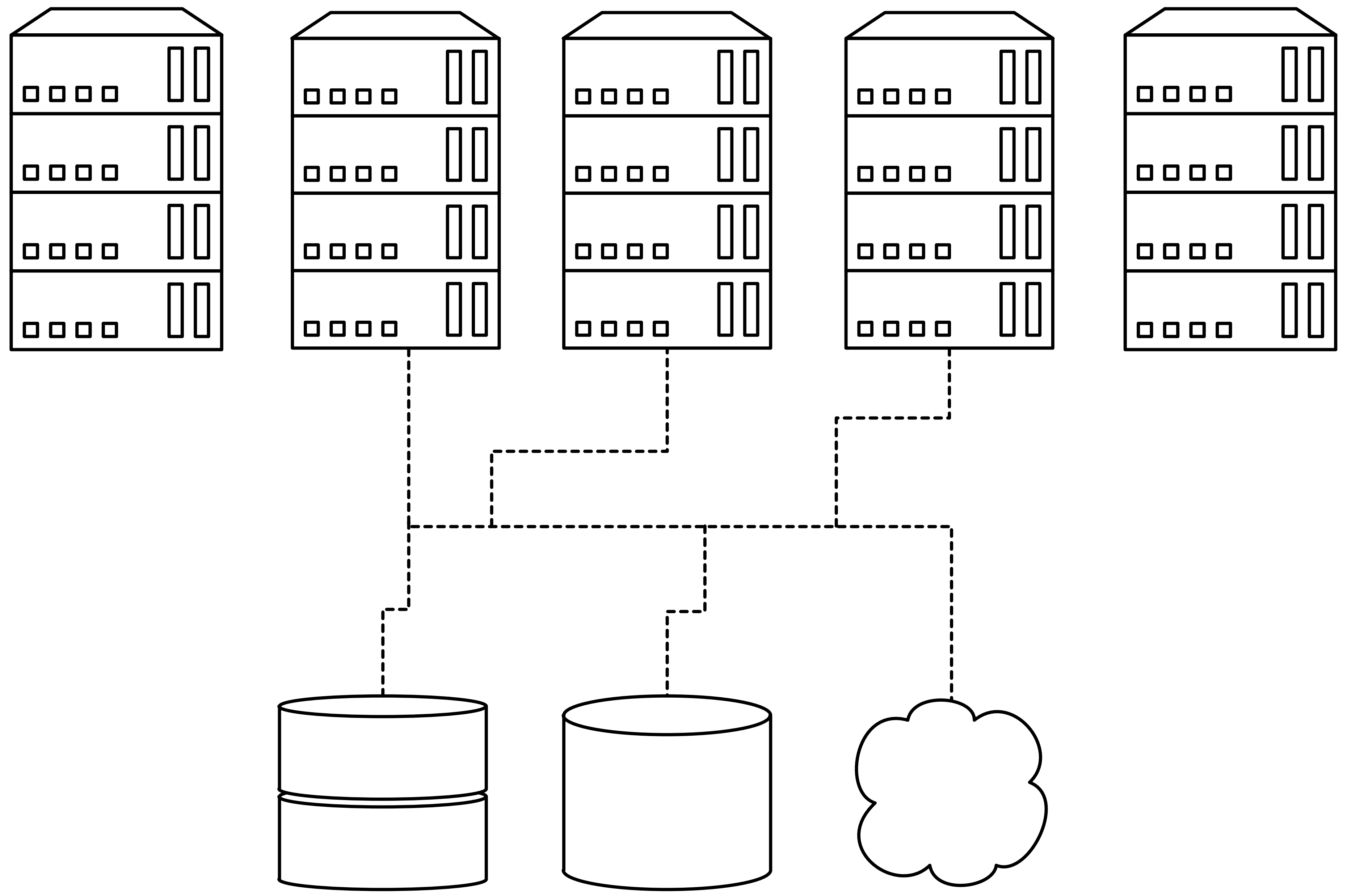
Instead of...





Instead of...





Instead of...

```
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {
 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) { word.set(tokenizer.nextToken()); context.write(word, one);}
   }
 }
```

Instead of...

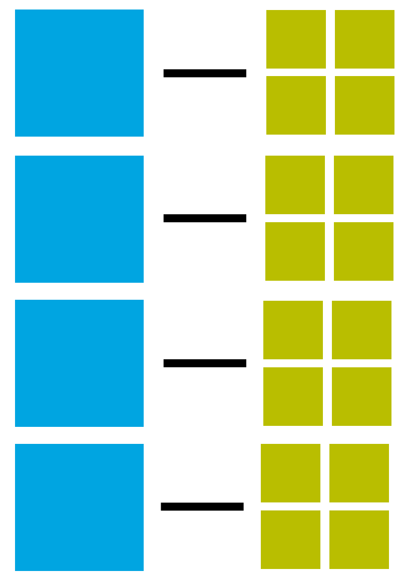
```java
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {
 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) { word.set(tokenizer.nextToken()); context.write(word, one);}
   }
 }

 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
   public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) { sum += val.get(); }
        context.write(key, new IntWritable(sum));
    }
 }

 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = new Job(conf, "wordcount");
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
 }
}
```

```scala
val file = spark.textFile("file://...")

val counts = file.flatMap(line =>
                   line.split(" "))
                  .map(word => (word, 1))
                  .reduceByKey(_ + _)


counts.saveAsTextFile("file://...")
```

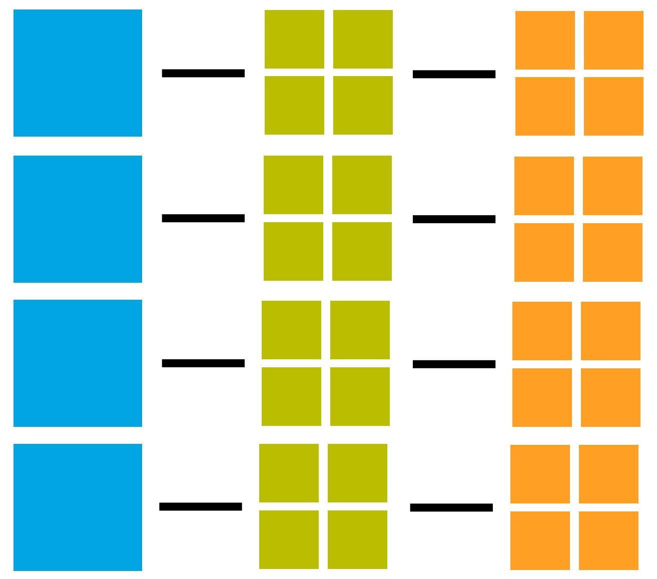
```scala
val file = spark.textFile("file://...")

val counts = file.flatMap(line =>
                    line.split(" "))
                  .map(word => (word, 1))
                  .reduceByKey(_ + _)


counts.saveAsTextFile("file://...")
```

```scala
val file = spark.textFile("file://...")

val counts = file.flatMap(line =>
                    line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)

counts.saveAsTextFile("file://...")
```

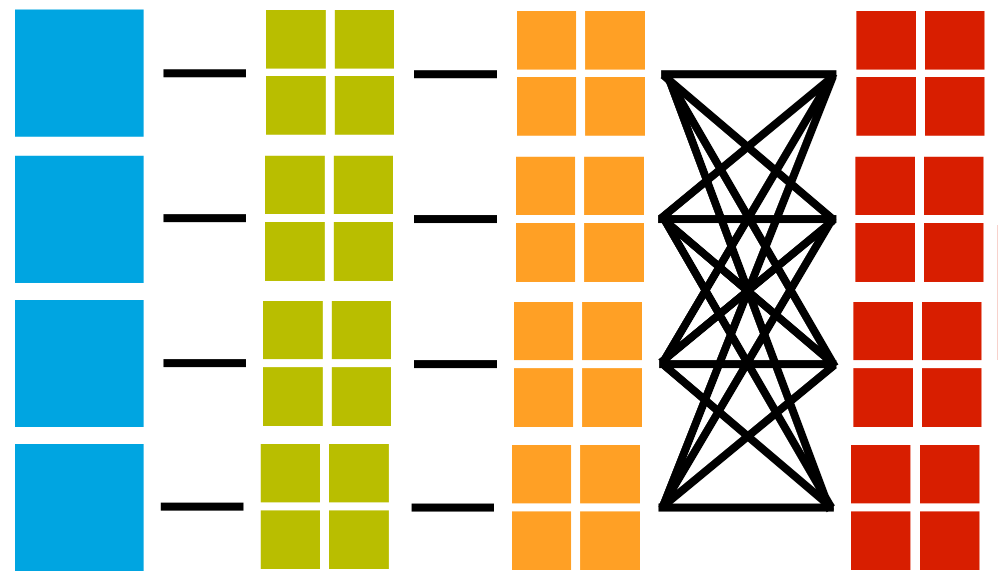
```
val file = spark.textFile("file://...")

val counts = file.flatMap(line =>
                    line.split(" "))
                .map(word => (word, 1))
                .reduceByKey(_ + _)

counts.saveAsTextFile("file://...")
```



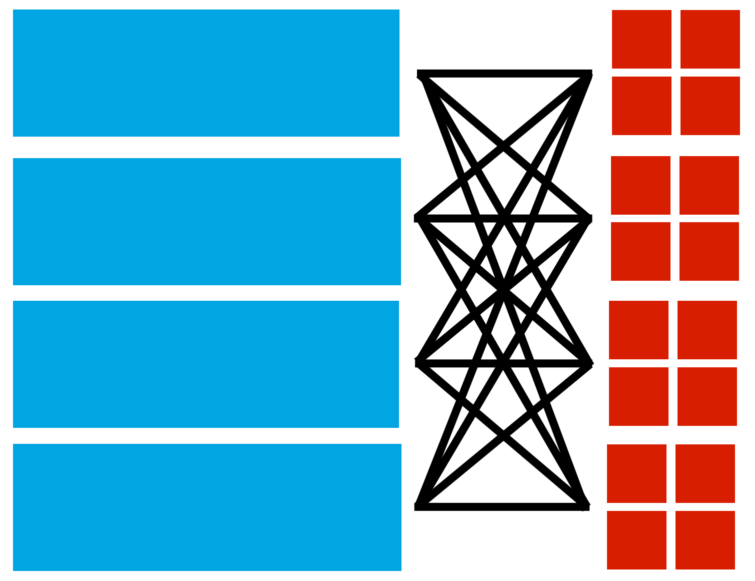
```scala
val file = spark.textFile("file://...")

val counts = file.flatMap(line =>
                 line.split(" "))
             .map(word => (word, 1))
             .reduceByKey(_ + _)

counts.saveAsTextFile("file://...")
```



```
val file = spark.textFile("file://...")

val counts = file.flatMap(line =>
                 line.split(" "))
               .map(word => (word, 1))
               .reduceByKey(_ + _)

counts.saveAsTextFile("file://...")
```



```scala
val file = spark.textFile("file://...")

val counts = file.flatMap(line =>
                   line.split(" "))
                .map(word => (word, 1))
                .reduceByKey(_ + _)
```



```scala
counts.saveAsTextFile("file://...")
```

Unlike Hadoop

Contemporary data frameworks like **Apache Spark** and **Apache Flink** provide **unified abstractions** and **rich standard libraries** for **machine learning**, **structured query with planning**, and **graph processing.**

DATA SCIENCE at RED HAT

Some of our recent projects

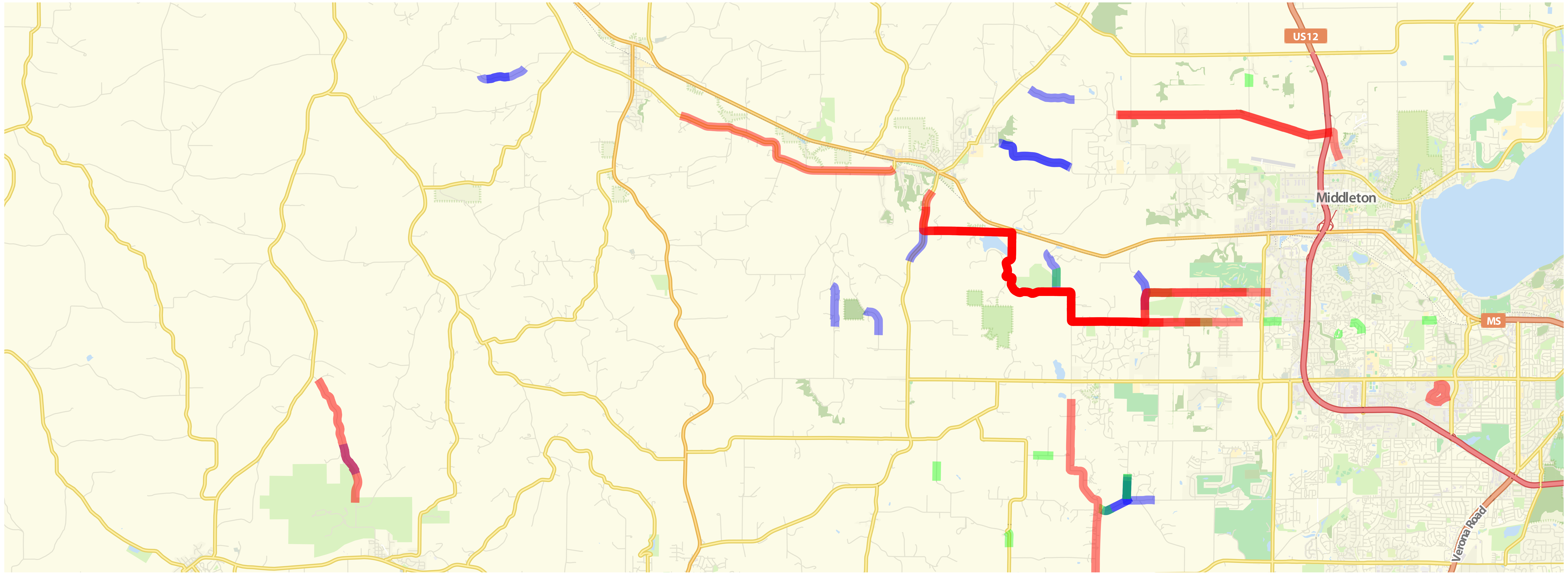
Analyzing open-source community health (Benton, Spark Summit ’15)

Customer segmentation with NLP (Nowling, Spark Summit East ’16)

Understanding benchmark results (Feddema, Apache: Big Data ’16)

Machine role analysis (Erlandson, Apache: Big Data ’16)

...and a not-so-recent one



Analyzing endurance-sports data (Benton, Spark Summit '14)

KEY TAKEAWAYS for your DATA SCIENCE INITIATIVES

When choosing a compute model, consider realistic working-set sizes rather than archive storage volume. (You may need petascale storage, but you probably don't even need terascale compute.)

Many real-world workloads benefit more from scale-up than scale-out. Choose a compute framework that lets you exploit both kinds of scale.

Don’t choose a compute model that forces you into unrealistic assumptions about persistent storage or requires dedicated resources.

Make sure that your data lifecycle facilitates generating reproducible results. Archive all unmodified source data and automate your processing and analysis workflows.

Predictive models are great, but predictions are only part of the story. Whenever possible, use human-interpretable models that help you understand something new about your problem.

THANKS!

@willb • willb@redhat.com
https://chapeau.freevariable.com