Lecture 20: NoSQL II

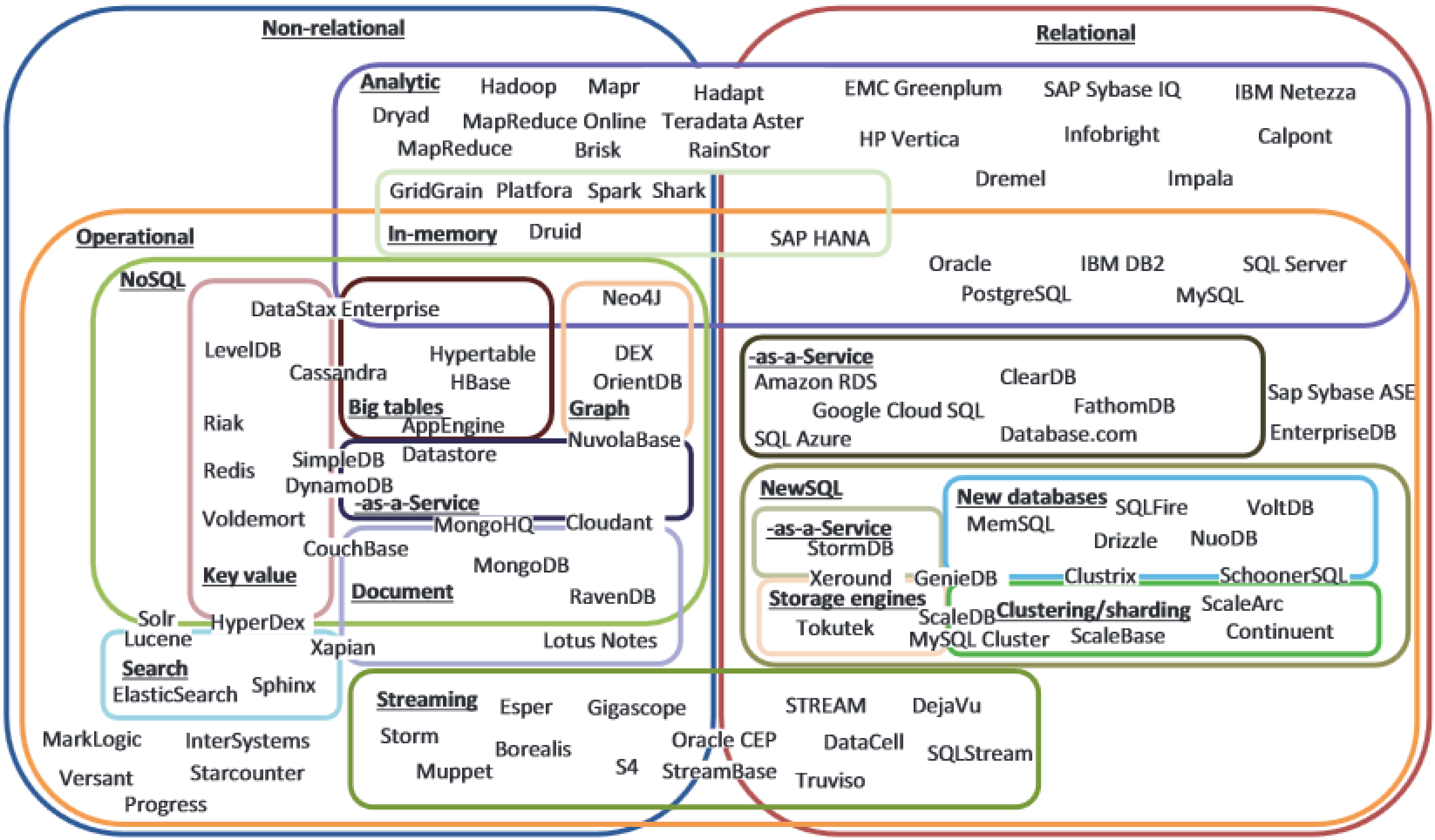
Monday, April 13, 2015

Announcements

- Today: MapReduce & flavor of Pig
- Next class: Cloud platforms and Quiz #6
- HW #4 is out and will be due 04/27

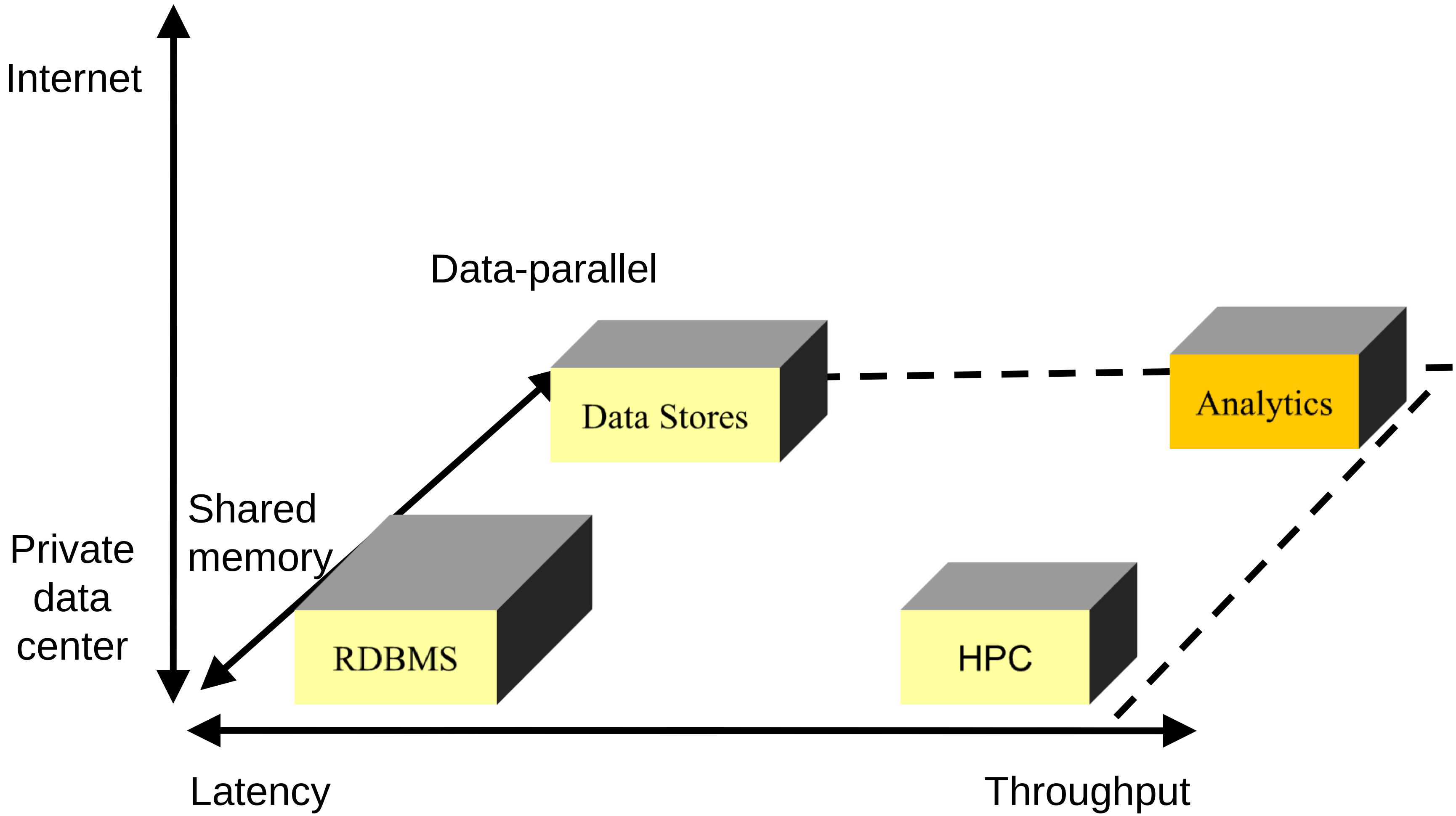
- Grading questions:
  - Class participation
  - Homeworks
  - Quizzes
  - Class project

“Data Systems” Landscape



Source: Lim et al, “How to Fit when No One Size Fits”, CIDR 2013.

Data Systems Design Space

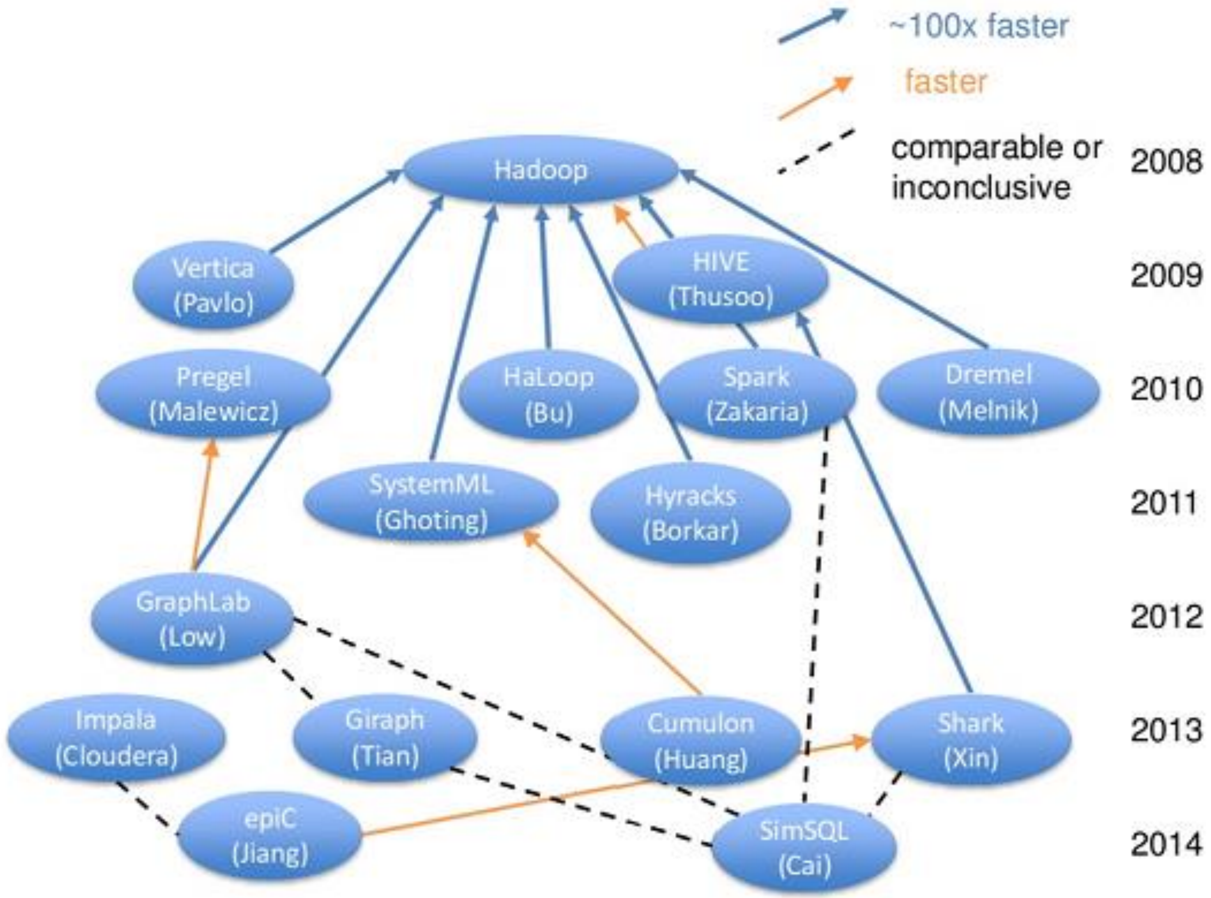


Source: Adapted from Michael Isard, Microsoft Research.

MapReduce

- MapReduce = high-level programming model and implementation for large-scale parallel data processing
- Inspired by primitives from Lisp and other functional programming languages
- History:
  - 2003: built at Google
  - 2004: published in OSDI (Dean & Ghemawat)
  - 2005: open-source version Hadoop
  - 2005 - 2014: very influential in DB community

MapReduce Literature



Source: David Maier and Bill Howe, "Big Data Middleware", CIDR 2015.

Data Model

MapReduce knows files!

A file = a bag of `(key, value)` pairs

A MapReduce program:

- Input: a bag of `(inputkey, value)` pairs
- Output: a bag of `(outputkey, values)` pairs

Step 1: Map Phase

- User provides the map function:
  - Input: one `(input key, value)` pair
  - Output: bag of `(intermediate key, value)` pairs

- MapReduce system applies the map function in parallel to all `(input key, value)` pairs in the input file

- Results from the Map phase are stored to disk and redistributed by the intermediate key during the Shuffle phase

Step 2: Reduce Phase

- MapReduce system groups all pairs with the same intermediate key, and passes the bag of values to the Reduce function

- User provides the Reduce function:
  - Input: `(intermediate key, bag of values)`
  - Output: `bag of output values`

- Results from Reduce phase stored to disk

Canonical Example

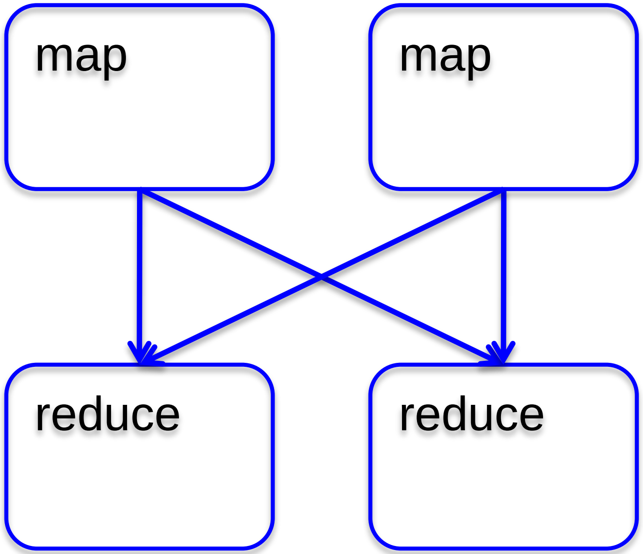
Pseudocode for counting the number of occurrences of each word in a large collection of documents

```
map(String key, String input_value):
  // key: document name
  // input_value: document contents
  for each word in input_value:
      EmitIntermediate(word, "1");
```

```
reduce(String inter_key, Iterator inter_values):
  // inter_key: a word
  // inter_values: a list of counts
  int sum = 0;
  for each value in inter_values:
      sum += ParseInt(value);
  EmitFinal(inter_key, sum);
```

Source: Adapted from “MapReduce: Simplified Data Processing on Large Clusters” (original MapReduce paper).

MapReduce Illustrated

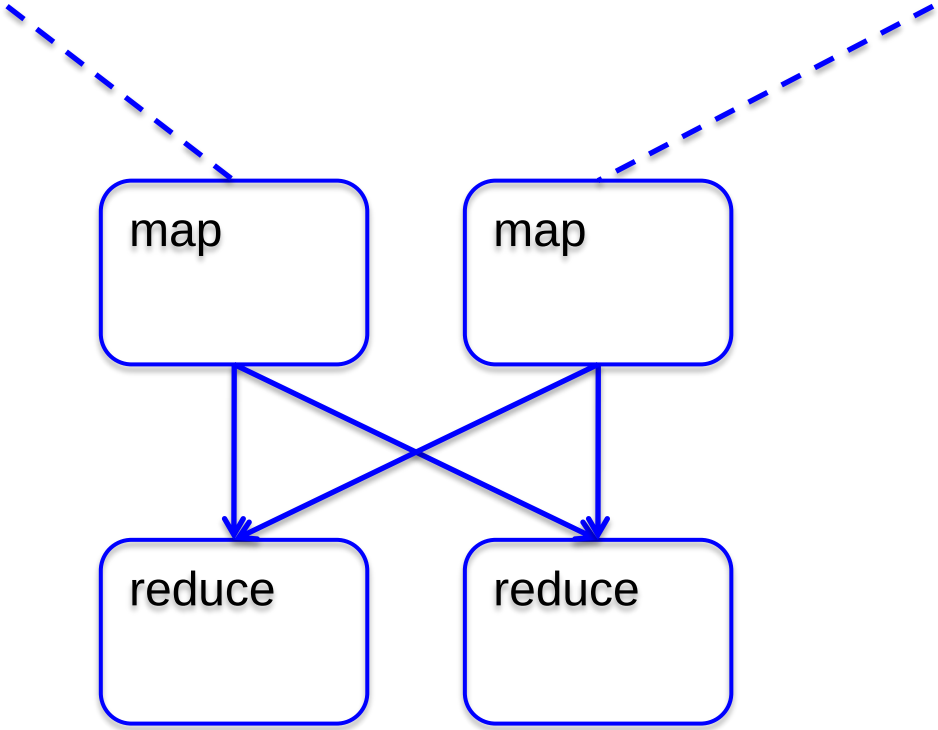


Source: Yahoo! Pig Team

MapReduce Illustrated

Romeo, Romeo, wherefore art thou Romeo?

What, art thou hurt?



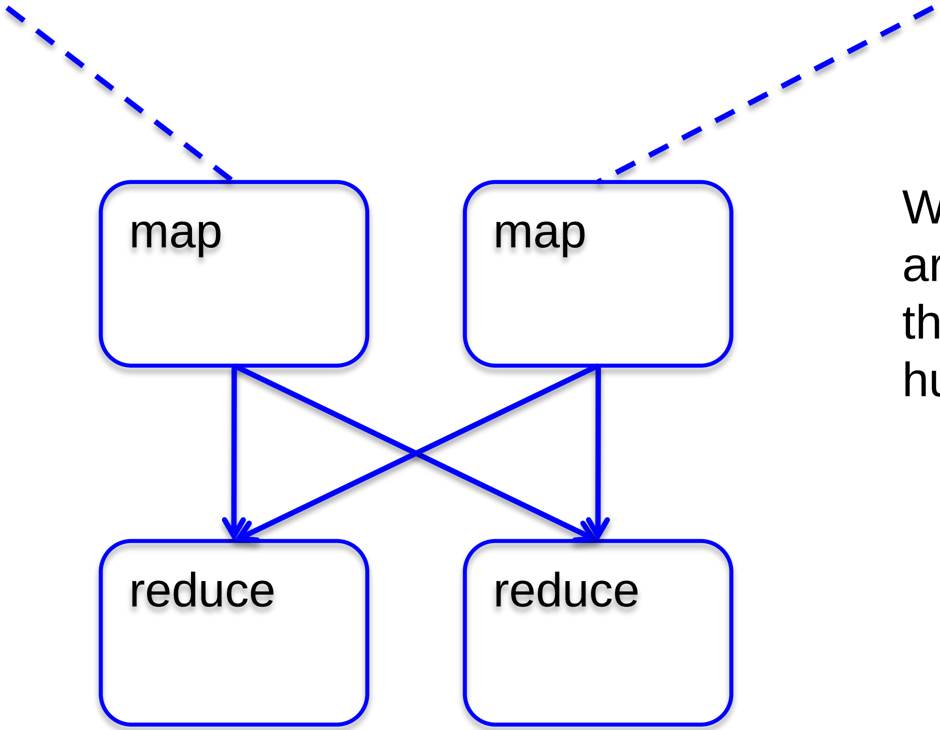
Source: Yahoo! Pig Team

MapReduce Illustrated

Romeo, Romeo, wherefore art thou Romeo?

What, art thou hurt?

Romeo, 1
Romeo, 1
wherefore, 1
art, 1
thou, 1
Romeo, 1



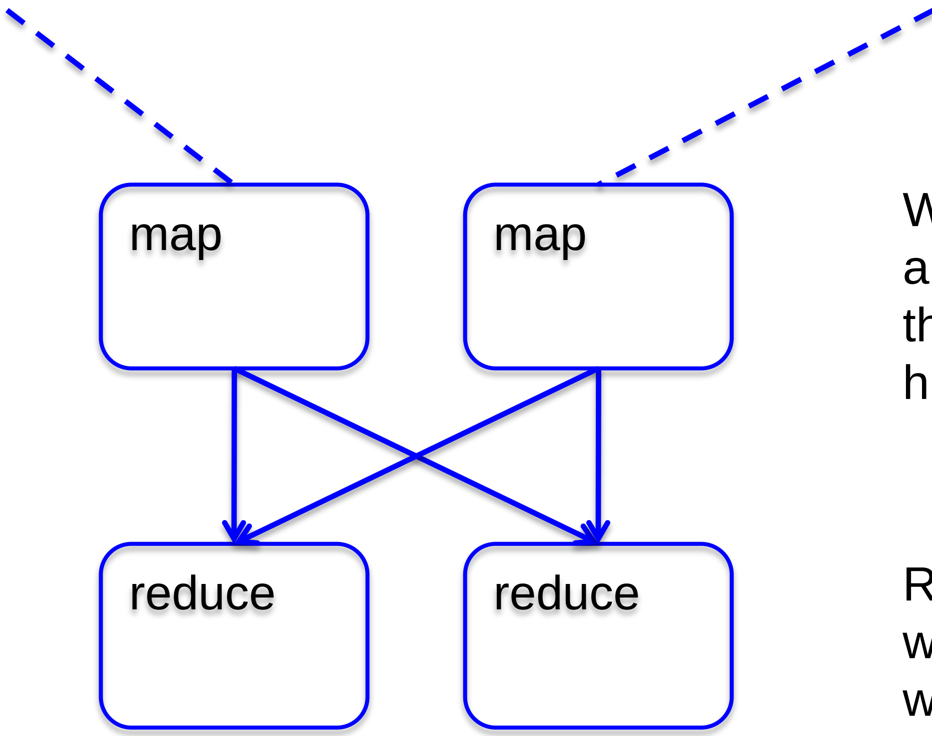
What, 1
art, 1
thou, 1
hurt, 1

Source: Yahoo! Pig Team

MapReduce Illustrated

Romeo, Romeo, wherefore art thou Romeo?

What, art thou hurt?



Romeo, 1
Romeo, 1
wherefore, 1
art, 1
thou, 1
Romeo, 1

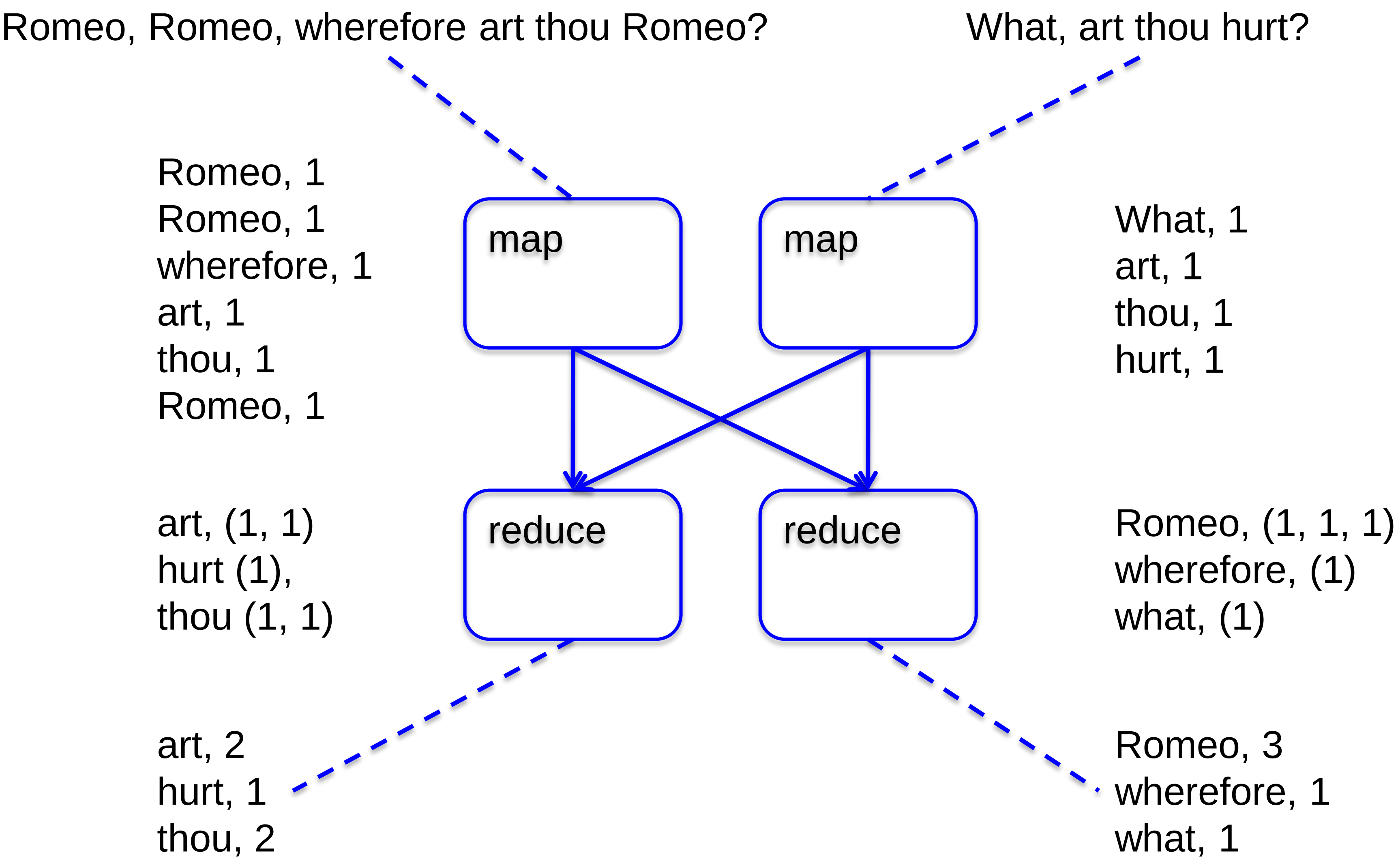
What, 1
art, 1
thou, 1
hurt, 1

art, (1, 1)
hurt (1),
thou (1, 1)

Romeo, (1, 1, 1)
wherefore, (1)
what, (1)

Source: Yahoo! Pig Team

MapReduce Illustrated



Source: Yahoo! Pig Team

Rewritten as SQL

Documents(document_id, word)

```
SELECT word, COUNT(*)
FROM Documents
GROUP BY word
```

Observe:
Map + Shuffle Phases = Group By
Reduce Phase = Aggregate

More generally, each of the SQL operators that we have studied can be implemented in MapReduce

Relational Join

Employees(<u>emp_id</u>, last_name, first_name, dept_id)
Departments(<u>dept_id</u>, dept_name)

```
SELECT *
FROM Employees e, Departments d
WHERE e.dept_id = d.dept_id
```

Relational Join

Employees(emp_id, emp_name, dept_id)

emp_id	emp_name	dept_id
20	Alice	100
21	Bob	100
25	Carol	150

Departments(dept_id, dept_name)

dept_id	dept_name
100	Product
150	Support
200	Sales

```
SELECT e.emp_id, e.emp_name, d.dept_id, d.dept_name
FROM Employees e, Deparments d
WHERE e.dept_id = d.dept_id
```

emp_id	emp_name	dept_id	dept_name
20	Alice	100	Product
21	Bob	100	Product
25	Carol	150	Support

Relational Join

Employees(emp_id, emp_name, dept_id)

emp_id	emp_name	dept_id
20	Alice	100
21	Bob	100
25	Carol	150

Departments(dept_id, dept_name)

dept_id	dept_name
100	Product
150	Support
200	Sales

Input:
Employee, 20, Alice, 100
Employee, 21, Bob, 100
Employee, 25, Carol, 150
Departments, 100, Product
Departments, 150, Support
Departments, 200, Sales

Map →

Output:
k=100,v=(Employee, 20, Alice, 100)
k=100,v=(Employee, 21, Bob, 100)
k=150, v=(Employee, 25, Carol, 150)
k=100, v=(Departments, 100, Product)
k=150, v=(Departments, 150, Support)
k=200, v=(Departments, 200, Sales)

Relational Join

Employees(emp_id, emp_name, dept_id)

emp_id	emp_name	dept_id
20	Alice	100
21	Bob	100
25	Carol	150

Departments(dept_id, dept_name)

dept_id	dept_name
100	Product
150	Support
200	Sales

Input:

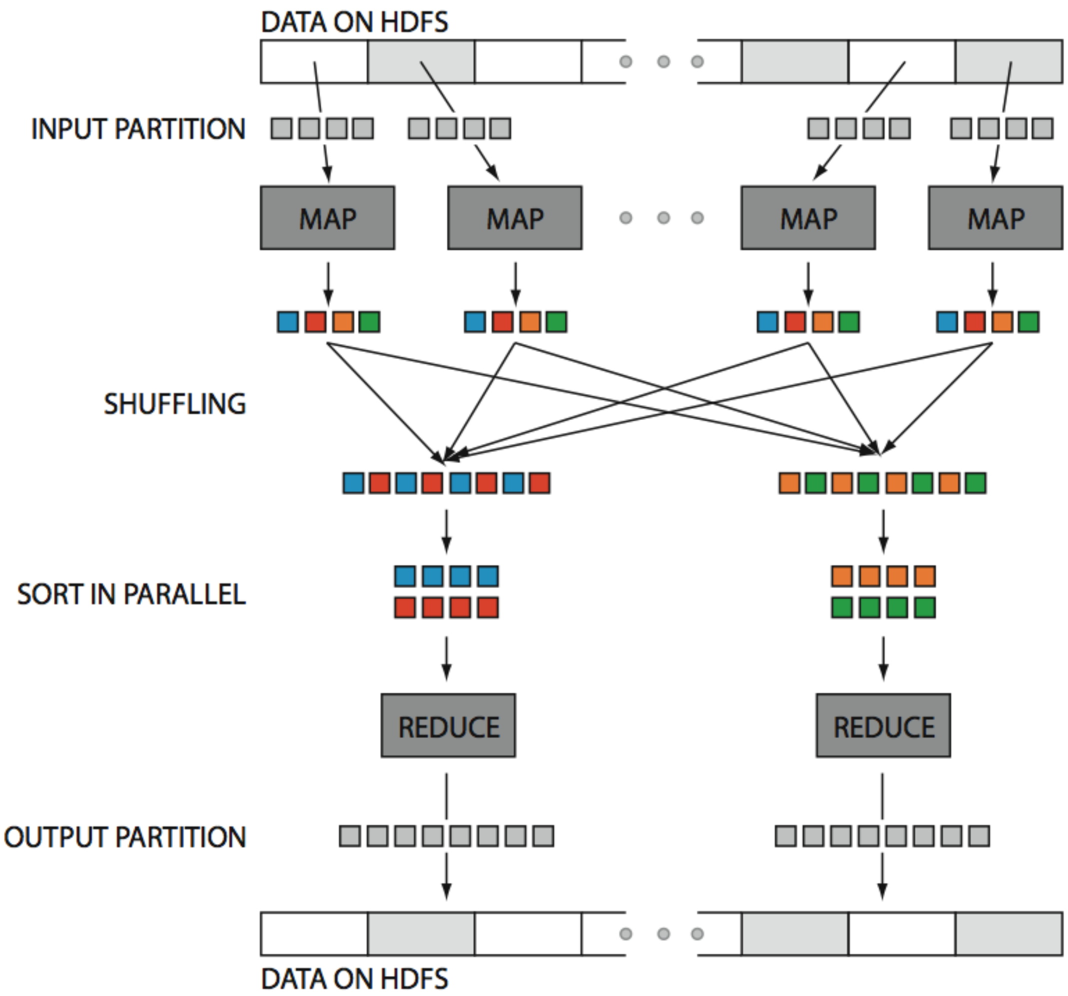
k=100,v=[(Employee, 20, Alice, 100),
(Employee, 21, Bob, 100),
(Departments, 100, Product)]
k=150, v=[(Employee, 25, Carol, 150),
(Departments, 150, Support)]
k=200, v=[(Departments, 200, Sales)]

Reduce →

Output:

20, Alice, 100, Product
21, Bob, 100, Product
25, Carol, 150, Support

Hadoop on One Slide



Source: Huy Vo, NYU Poly

MapReduce Internals

- Single master node
- Master partitions input file by key into **M splits** (> servers)
- Master assigns **workers** (=servers) to the **M map tasks**, keeping track of their progress
- Workers write their output to local disk, partition into **R regions** (> servers)
- Master assigns workers to the **R reduce tasks**
- Reduce workers read regions from the map workers' local disks

Key Implementation Details

- Worker failures:
  - Master pings workers periodically, looking for stragglers
  - When straggle is found, master reassigns splits to **other** workers
  - Stragglers are a main reason for slowdown
  - Solution: pre-emptive backup execution of last few remaining in-progress tasks
- Choice of M and R:
  - Larger than servers is better for load balancing

MapReduce Summary

- Hides scheduling and parallelization details
- Not most efficient implementation, but has great fault tolerance
- However, limited queries:
  - Difficult to write more complex tasks
  - Need multiple MapReduce operations
- Solution:
  - Use high-level language (e.g. Pig, Hive, Sawzall, Dremel, Tenzing) to express complex queries
  - Need optimizer to compile queries into MR tasks

MapReduce Summary

- Hides scheduling and parallelization details
- Not most efficient implementation, but has great fault tolerance
- However, limited queries:
  - Difficult to write more complex tasks
  - Need multiple MapReduce operations
- Solution:
  - Use high-level language (e.g. Pig, Hive, Sawzall, Dremel, Tenzing) to express complex queries
  - Need optimizer to compile queries into MR tasks

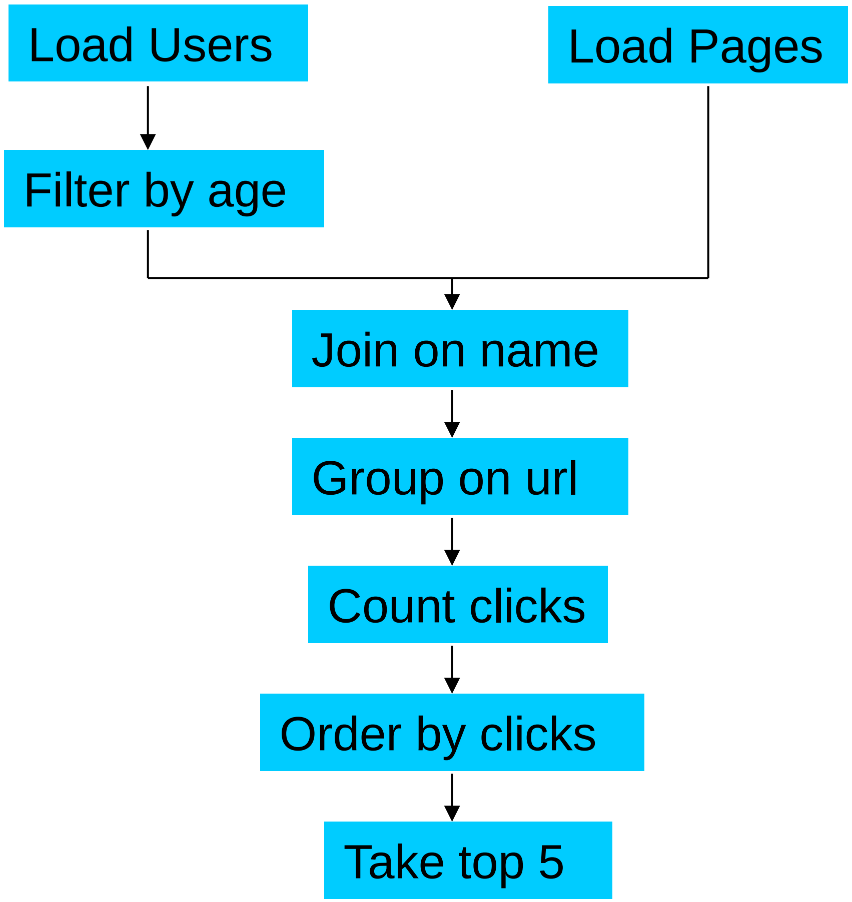
Pig & Pig Latin

- An engine and language for executing programs on top of Hadoop
- Logical plan → sequence of MapReduce ops
- Free and open-sourced (unlike some others) http://hadoop.apache.org/pig/
- ~70% of Hadoop jobs are Pig jobs at Yahoo!
- Being used at Twitter, LinkedIn, and other companies
- Available as part of Amazon, Hortonworks and Cloudera Hadoop distributions

Why use Pig?

Find the top 5 most visited sites by users aged 18 - 25.

Assume: user data stored in one file and website data in another file.



Source: Yahoo! Pig Team

In MapReduce

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

170 lines of code, 4 hours to write

Source: Yahoo! Pig Team

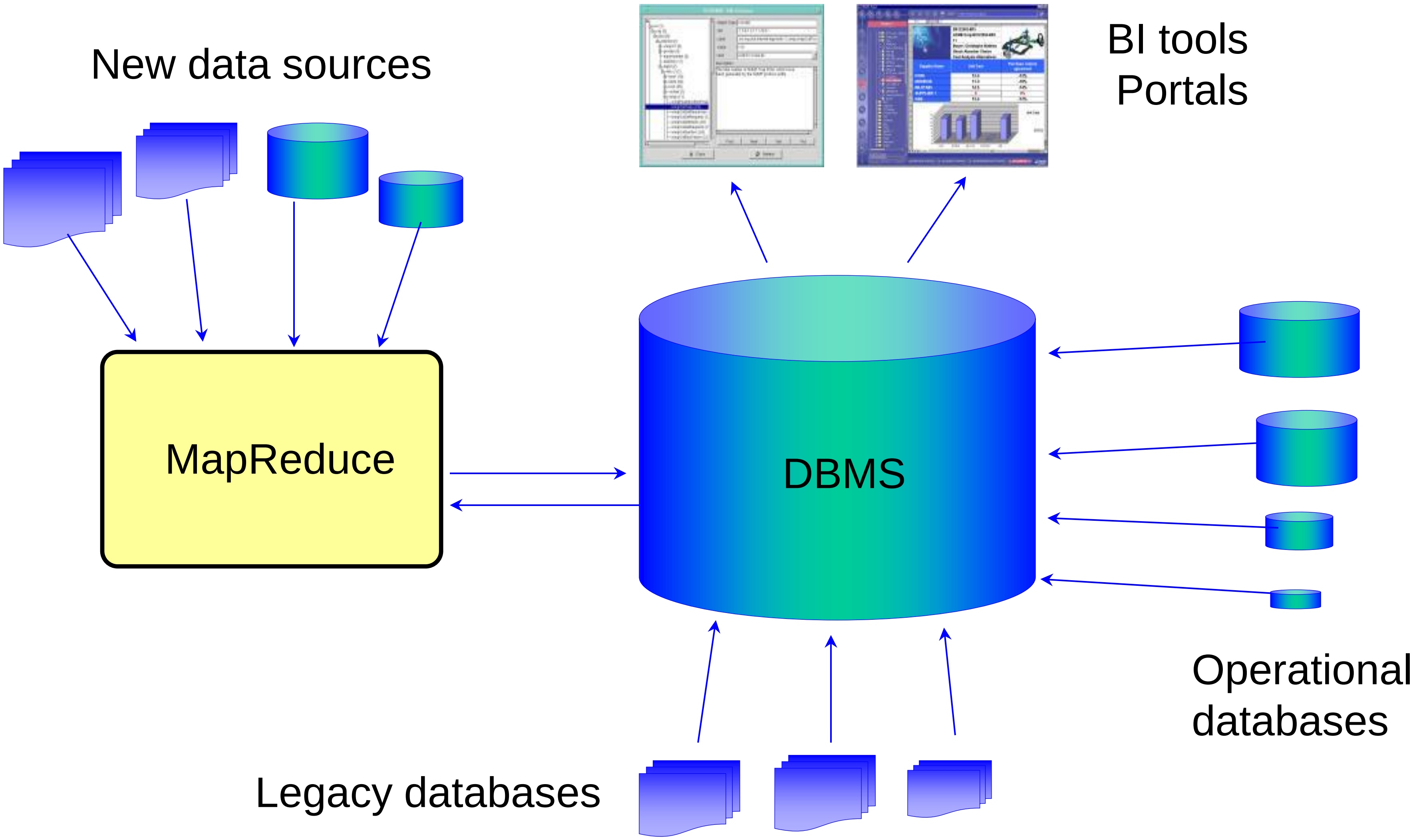
In Pig Latin

```
Users = load 'users' as (name, age);
Fltrd = filter Users by
        age >= 18 and age <= 25;
Pages = load 'pages' as (user, url);
Jnd = join Fltrd by name, Pages by user;
Grpd = group Jnd by url;
Smmd = foreach Grpd generate group,
       COUNT(Jnd) as clicks;
Srtd = order Smmd by clicks desc;
Top5 = limit Srtd 5;
store Top5 into 'top5sites';
```

9 lines of code, 15 minutes to write

Source: Yahoo! Pig Team

Emerging Analytics Pipeline



Optional References

MapReduce: Simplified Data Processing on Large Clusters [Dean & Ghemawarat OSDI ‘04]

Pig Latin: A Not-So-Foreign Language for Data Processing [Olston et. al. SIGMOD ‘08]

Hive – A Petabyte Scale Data Warehouse Using Hadoop [Thusoo VLDB ‘09]

Designs, Lessons and Advice from Building Large Distributed Systems [Dean LADIS ‘09]

Tenzing: A SQL Implementation On The MapReduce Framework [Chattopadhyay et. al. VLDB ‘11]

Next Class

- Cloud platforms (guest speaker Jacob Walcik)
- Quiz #6