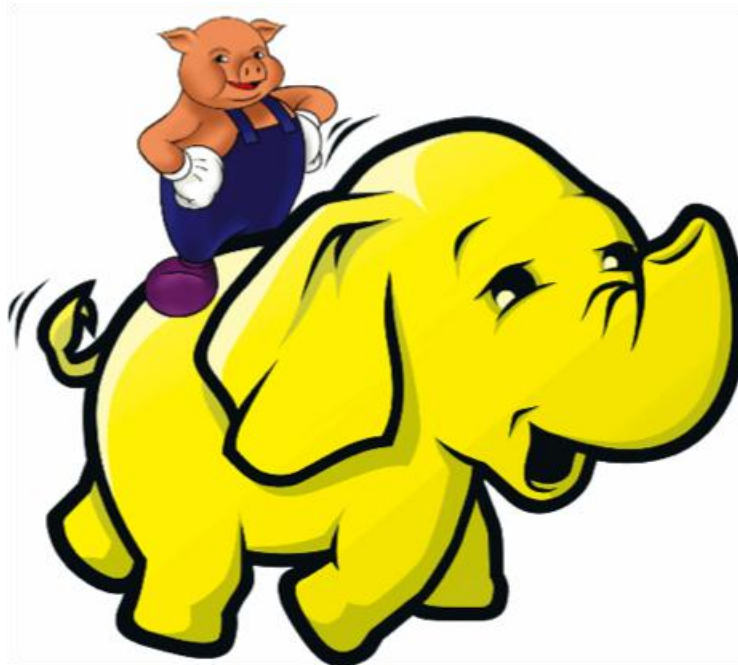# Lecture 26: Pig: Making Hadoop Easy (Some Slides provided by: Alan Gates, Yahoo!Research)

Friday, Dec 3, 2010

What is Pig?

- An engine for executing programs on top of Hadoop

- It provides a language, Pig Latin, to specify these programs

- An Apache open source project
http://hadoop.apache.org/pig/

Map-Reduce

- Computation is moved to the data

- A simple yet powerful programming model
  - Map: every record handled individually
  - Shuffle: records collected by key
  - Reduce: key and iterator of all associated values

- User provides:
  - input and output (usually files)
  - map Java function
  - key to aggregate on
  - reduce Java function

- Opportunities for more control: partitioning, sorting, partial aggregations, etc.
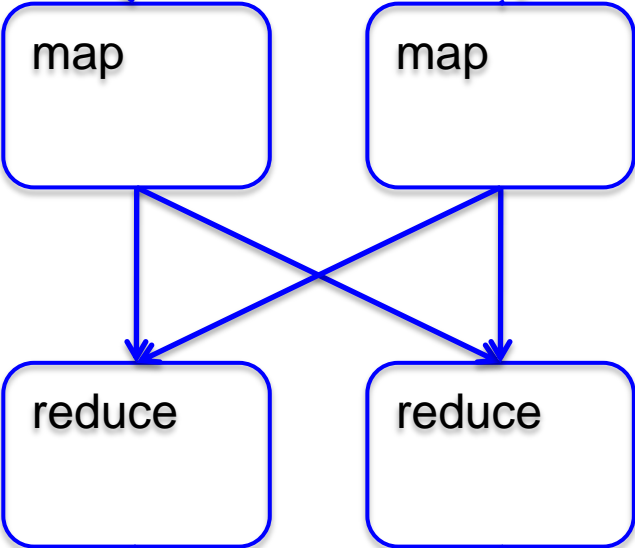
Map Reduce Illustrated

Romeo, Romeo, wherefore art thou Romeo?

What, art thou hurt?

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

map

map

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

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

reduce

reduce

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

art, 2
hurt, 1
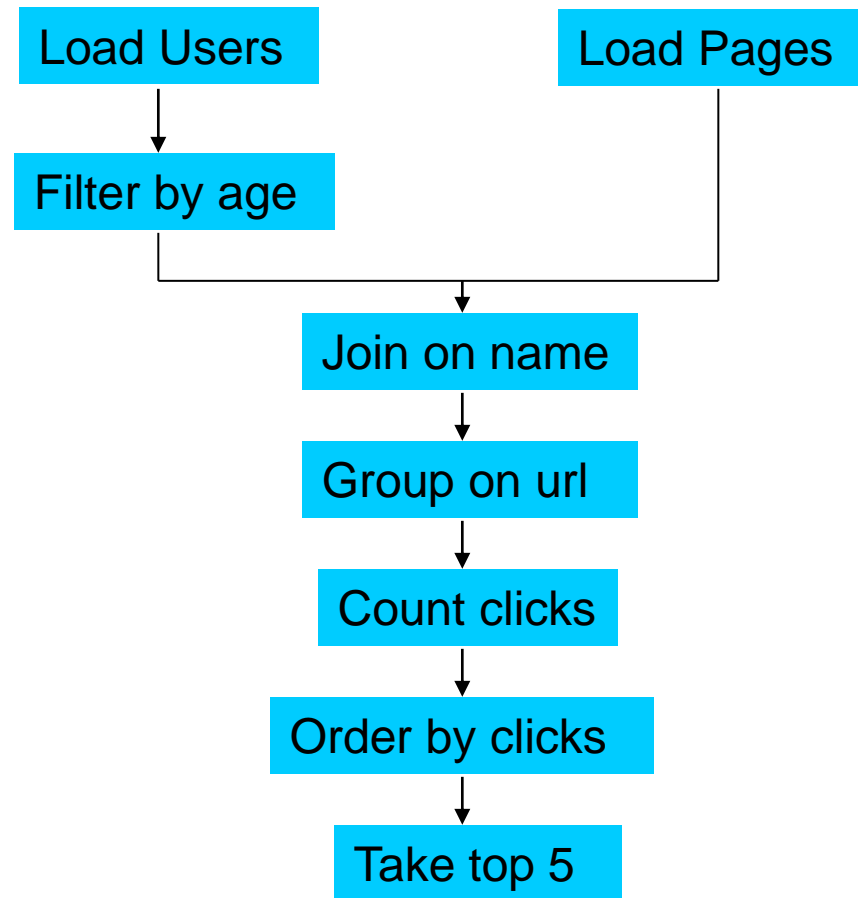thou, 2

Romeo, 3
wherefore, 1
what, 1

Making Parallelism Simple

- Sequential reads = good read speeds

- In large cluster failures are guaranteed; Map Reduce handles retries

- Good fit for batch processing applications that need to touch all your data:
  - data mining
  - model tuning

- Bad fit for applications that need to find one particular record

- Bad fit for applications that need to communicate between processes; oriented around independent units of work

Why use Pig?

Suppose you have user data in one file, website data in another, and you need to find the top 5 most visited sites by users aged 18 - 25.

```
Load Users                    Load Pages
    |                             |
Filter by age                    |
    |_____|_____|
                      |
                 Join on name
                      |
                 Group on url
                      |
                  Count clicks
                      |
                 Order by clicks
                      |
                  Take top 5
```

```
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();
    }
}
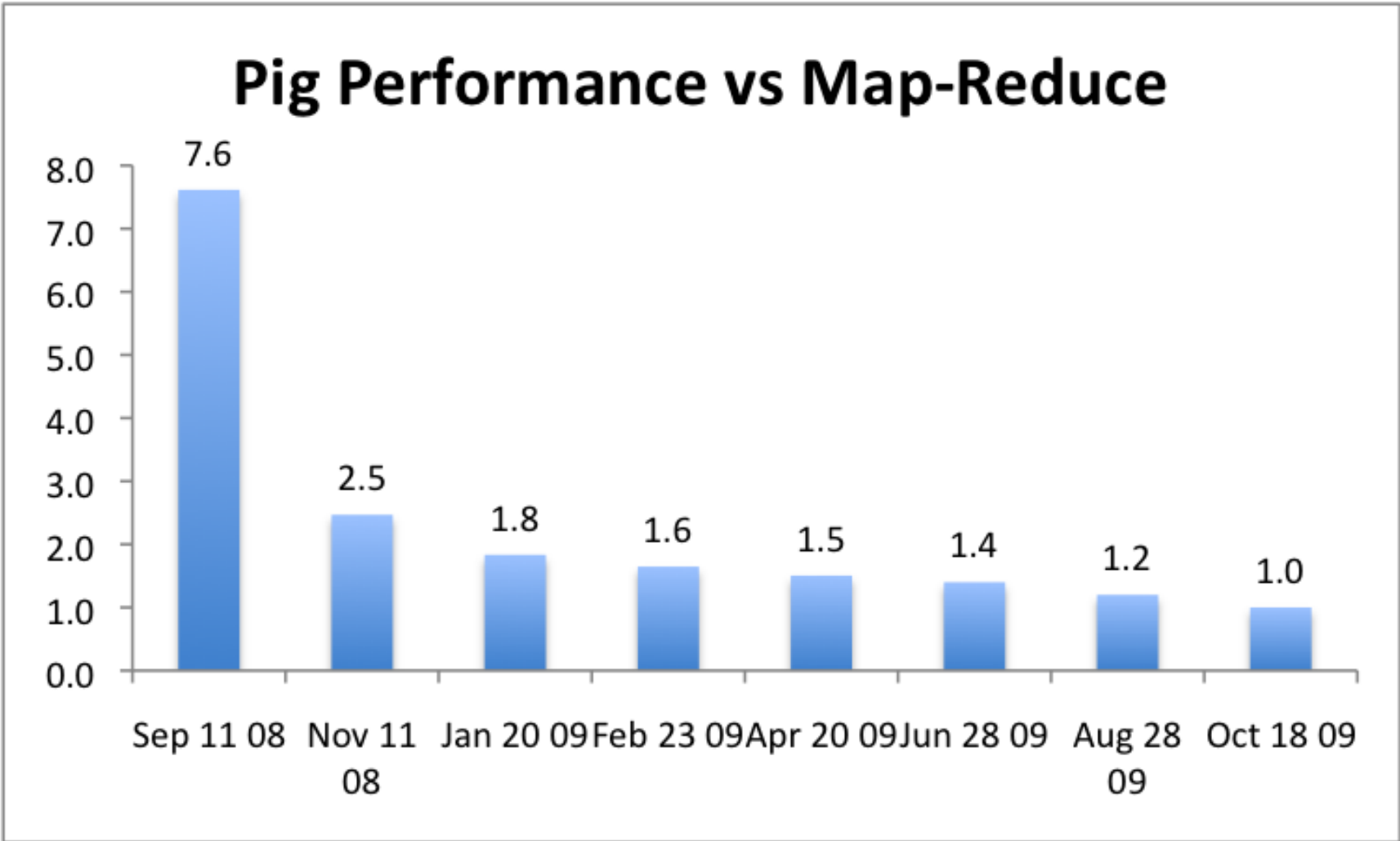```

170 lines of code, 4 hours to write

# 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';
```

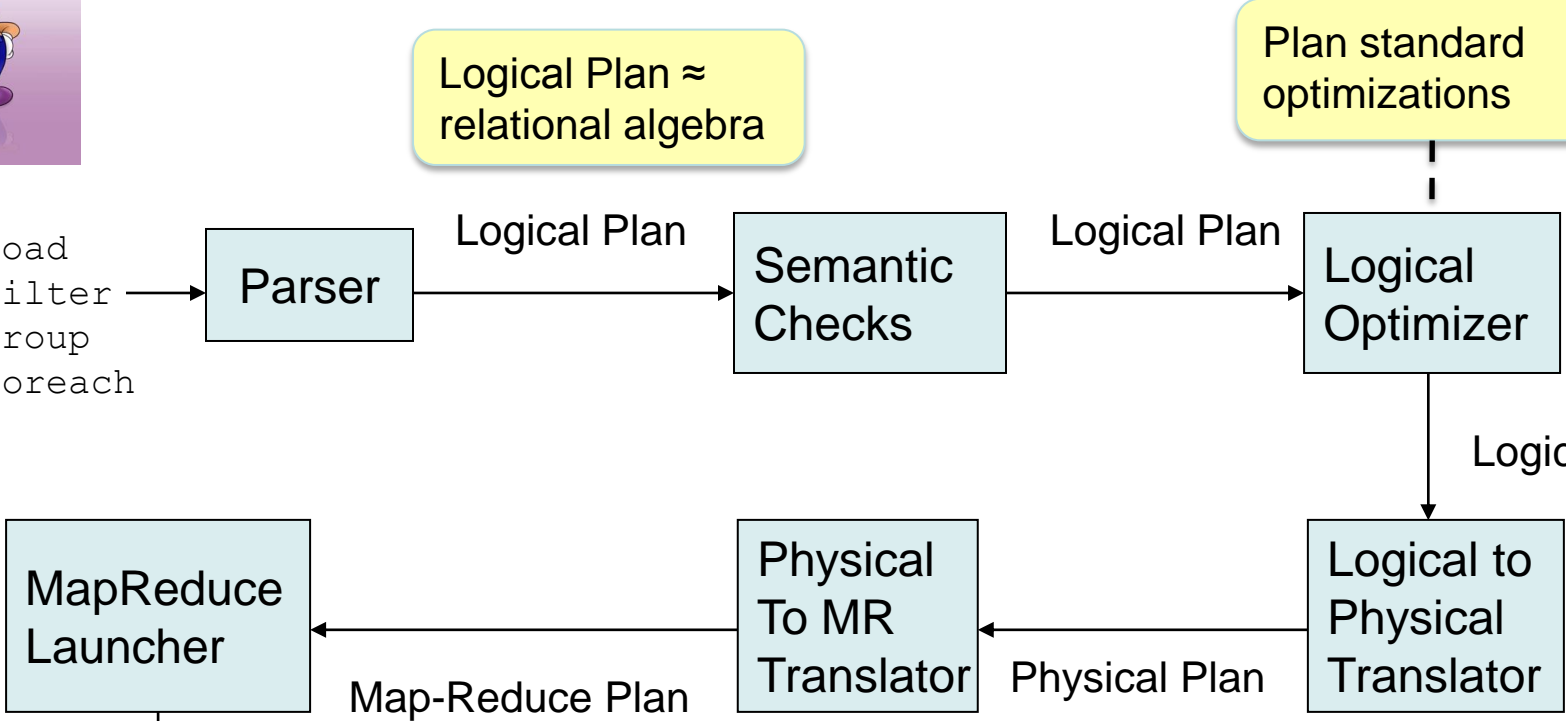9 lines of code, 15 minutes to write

But can it fly?

Pig Performance vs Map-Reduce

Date	Value
Sep 11 08	7.6
Nov 11 08	2.5
Jan 20 09	1.8
Feb 23 09	1.6
Apr 20 09	1.5
Jun 28 09	1.4
Aug 28 09	1.2
Oct 18 09	1.0

Essence of Pig

- Map-Reduce is too low a level to program, SQL too high

- Pig Latin, a language intended to sit between the two:

  - Imperative

  - Provides standard relational transforms (join, sort, etc.)

  - Schemas are optional, used when available, can be defined at runtime

  - User Defined Functions are first class citizens

  - Opportunities for advanced optimizer but optimizations by programmer also possible

How It Works

Script
```
A = load
B = filter
C = group
D = foreach
```

Logical Plan ≈ relational algebra

Plan standard optimizations

Parser → Logical Plan → Semantic Checks → Logical Plan → Logical Optimizer

Logical Plan

MapReduce Launcher ← Map-Reduce Plan ← Physical To MR Translator ← Physical Plan ← Logical to Physical Translator

Jar to hadoop

Map-Reduce Plan = physical operators broken into Map, Combine, and Reduce stages

Physical Plan = physical operators to be executed

Cool Things We've Added In the Last Year

- Multiquery – Ability to combine multiple group bys into a single MR job (0.3)

- Merge join – If data is already sorted on join key, do join via merge in map phase (0.4)

- Skew join – Hash join for data with skew in join key. Allows splitting of key across multiple reducers to handle skew. (0.4)

- Zebra – Contrib project that provides columnar storage of data (0.4)

- Rework of Load and Store functions to make them much easier to write (0.7, branched but not released)

- Owl, a metadata service for the grid (committed, will be released in 0.8).

Fragment Replicate Join

```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Pages by user, Users by name using "replicated";
```

Pages

Users

Map 1

Pages block 1
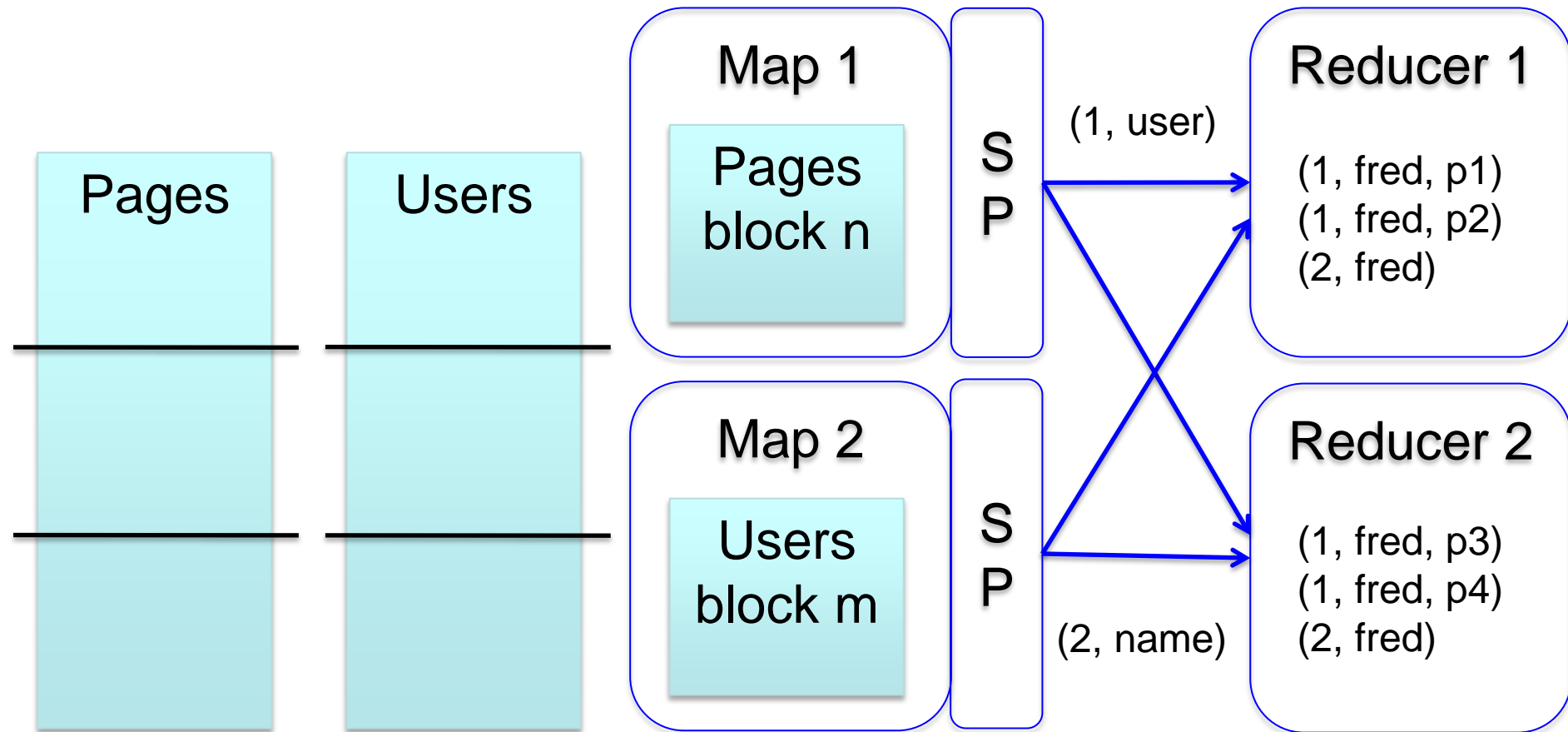
Users

Map 2

Pages block 2
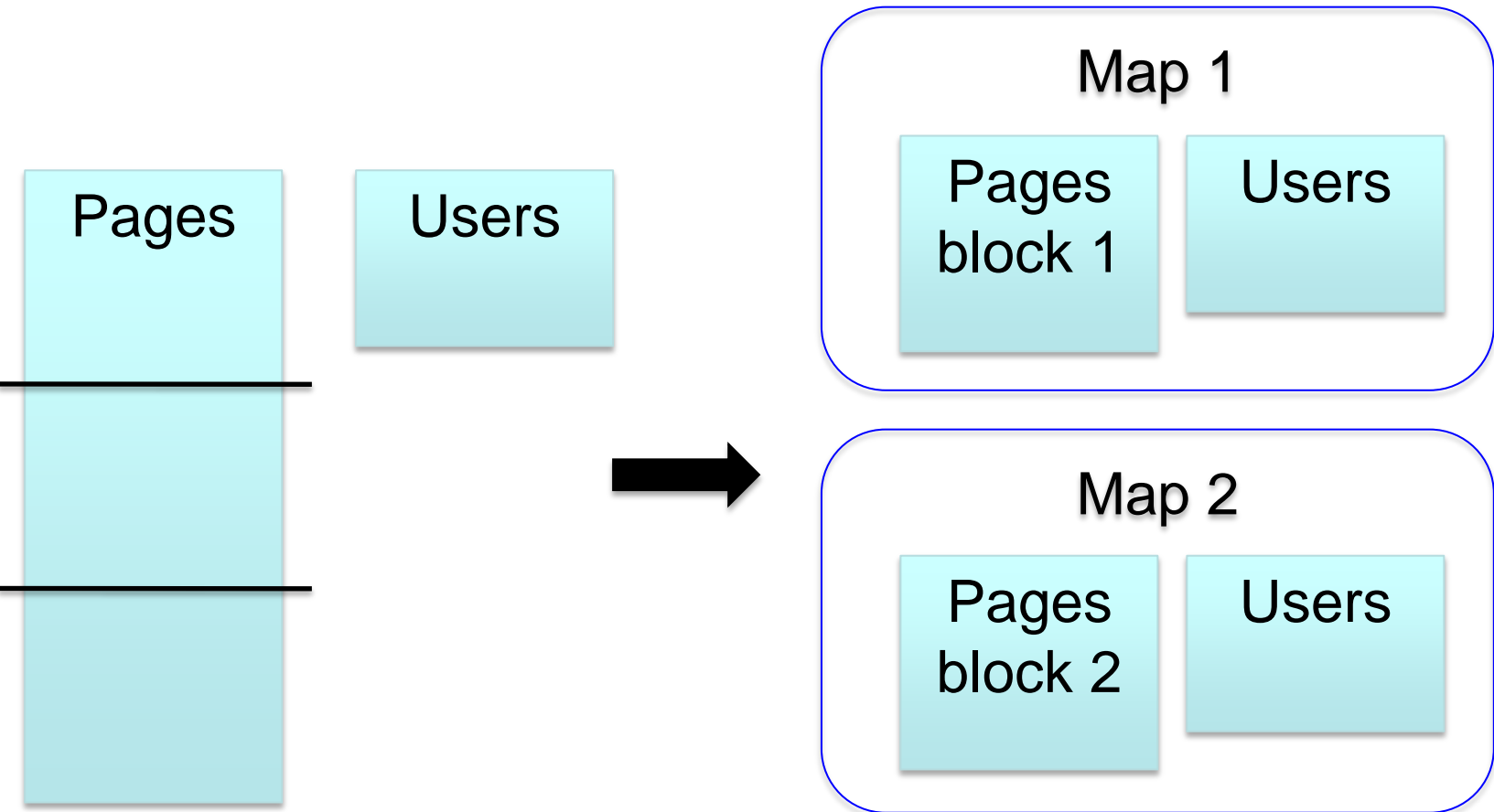
Users

Hash Join

```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Users by name, Pages by user;
```
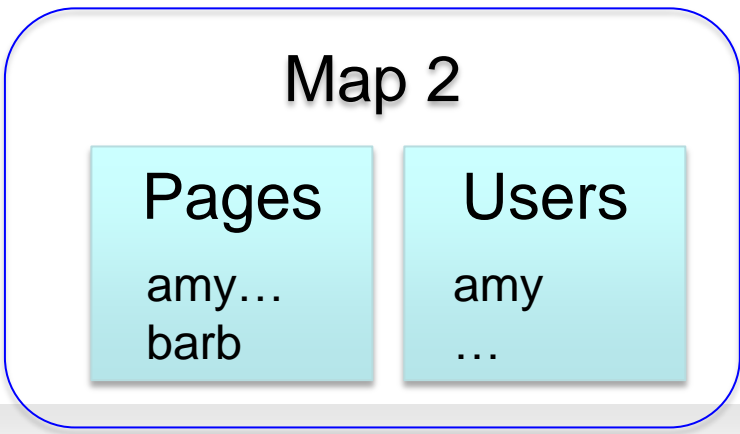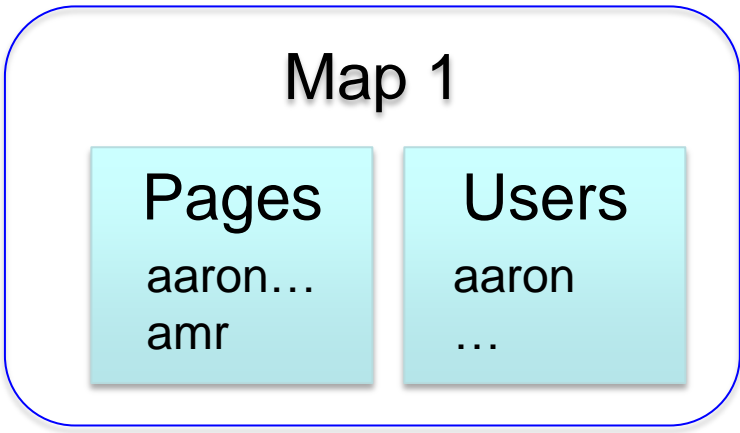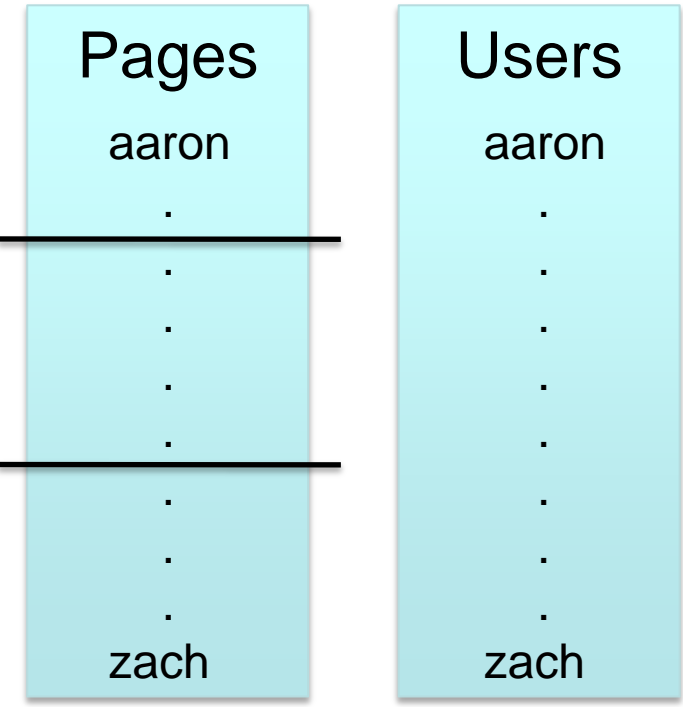
Pages

Users

Map 1

User block n

Map 2

Page block m

(1, user)

(2, name)

Reducer 1

(1, fred)
(2, fred)
(2, fred)

Reducer 2

(1, jane)
(2, jane)
(2, jane)
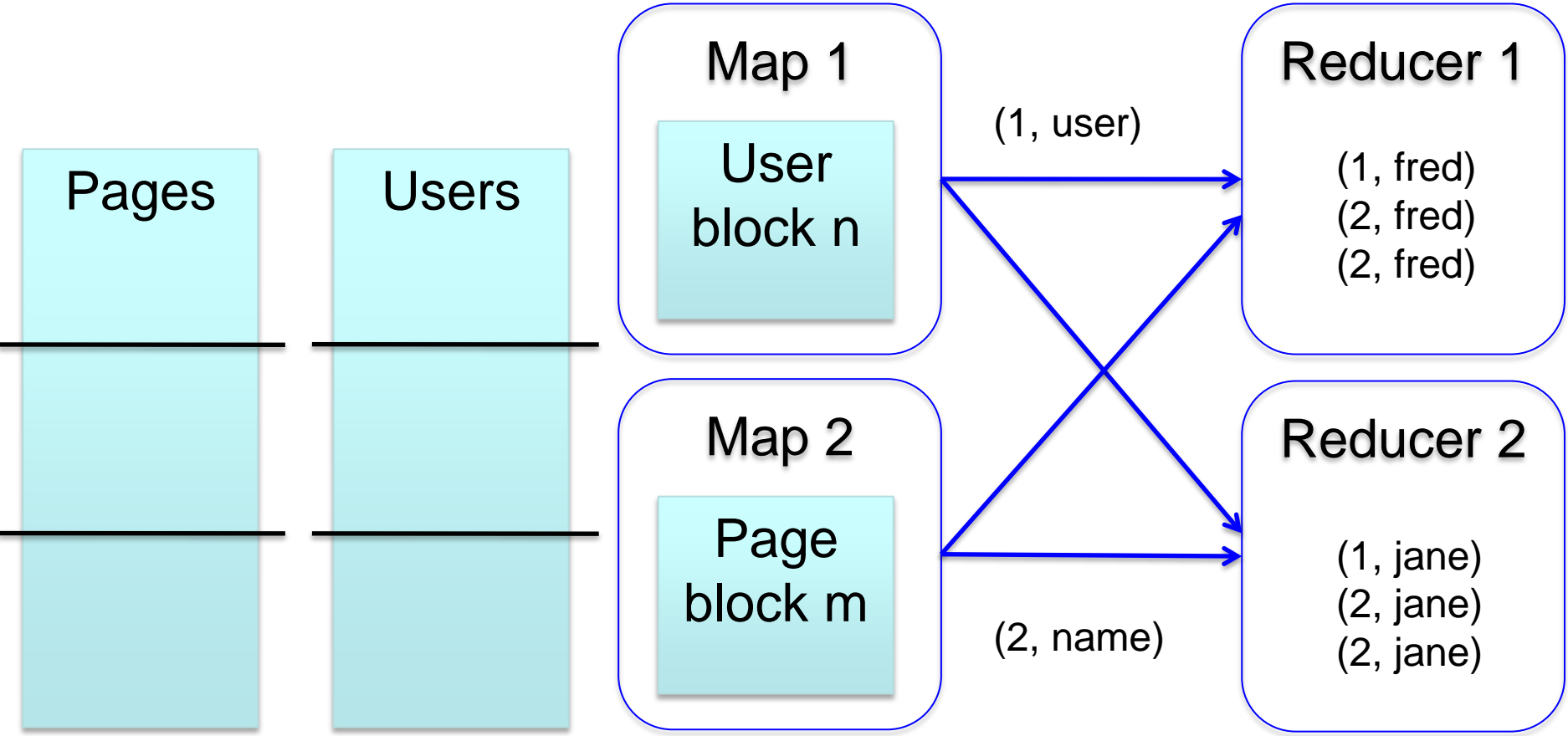
Skew Join

```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Pages by user, Users by name using "skewed";
```

Merge Join

```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Pages by user, Users by name using "merge";
```

Pages

aaron
.
.
.
.
.
.
.
zach

Users

aaron
.
.
.
.
.
.
.
zach

Map 1

Pages
aaron…
amr

Users
aaron
…

Map 2

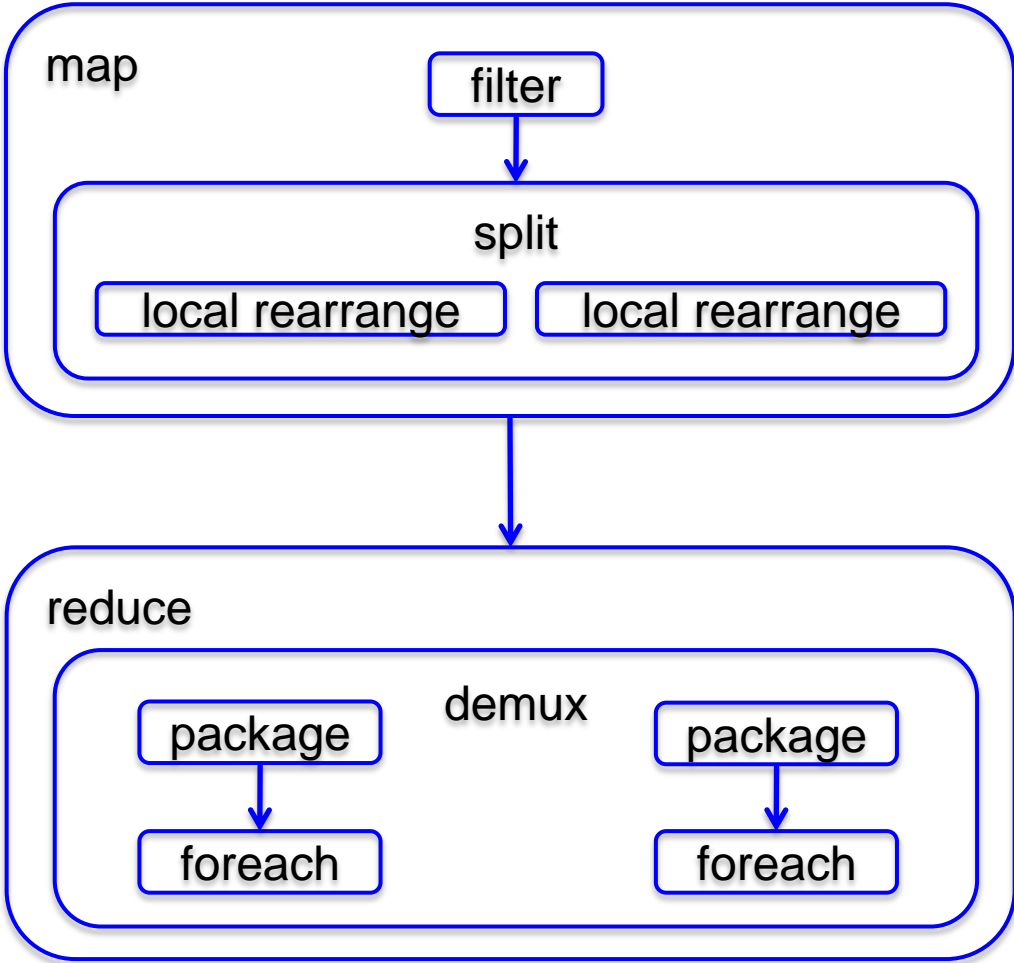**Pages**
amy…
barb

Users
amy
…

Multi-store script

```
A = load 'users' as (name, age, gender,
        city, state);
B = filter A by name is not null;
C1 = group B by age, gender;
D1 = foreach C1 generate group, COUNT(B);
store D into 'bydemo';
C2= group B by state;
D2 = foreach C2 generate group, COUNT(B);
store D2 into 'bystate';
```
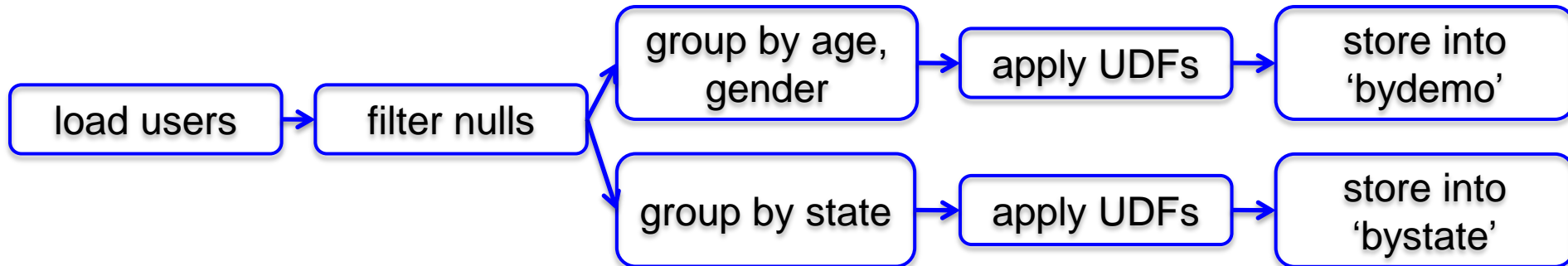
Multi-Store Map-Reduce Plan

What are people doing with Pig

- At Yahoo ~70% of Hadoop jobs are Pig jobs

- Being used at Twitter, LinkedIn, and other companies

- Available as part of Amazon EMR web service and Cloudera Hadoop distribution

- What users use Pig for:
    - Search infrastructure
    - Ad relevance
    - Model training
    - User intent analysis
    - Web log processing
    - Image processing
    - Incremental processing of large data sets

What We're Working on this Year

- Optimizer rewrite

- Integrating Pig with metadata

- Usability – our current error messages might as well be written in actual Latin

- Automated usage info collection

- UDFs in python

Research Opportunities

- Cost based optimization – how does current RDBMS technology carry over to MR world?

- Memory Usage – given that data processing is very memory intensive and Java offers poor control of memory usage, how can Pig be written to use memory well?

- Automated Hadoop Tuning – Can Pig figure out how to configure Hadoop to best run a particular script?

- Indices, materialized views, etc. – How do these traditional RDBMS tools fit into the MR world?

- Human time queries – Analysts want access to the petabytes of data available via Hadoop, but they don't want to wait hours for their jobs to finish; can Pig find a way to answer analysts question in under 60 seconds?

- Map-Reduce-Reduce – Can MR be made more efficient for multiple MR jobs?

- How should Pig integrate with workflow systems?

- See more: http://wiki.apache.org/pig/PigJournal

Learn More

- Visit our website: http://hadoop.apache.org/pig/

- On line tutorials

    – From Yahoo, http://developer.yahoo.com/hadoop/tutorial/

    – From Cloudera, http://www.cloudera.com/hadoop-training

- A couple of Hadoop books are available that include chapters on Pig, search at your favorite bookstore

- Join the mailing lists:

    – pig-user@hadoop.apache.org for user questions

    – pig-dev@hadoop.apache.com for developer issues

- Contribute your work, over 50 people have so far