Lecture 10:
Parallel Databases

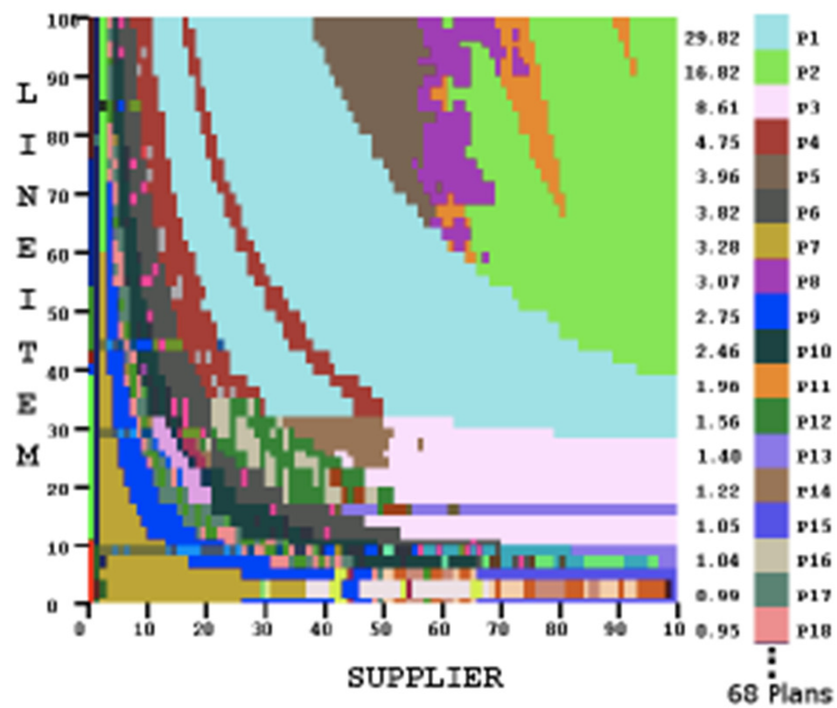
Wednesday, December 1st, 2010

Announcements

- Take-home Final: this weekend
- Next Wednesday: last homework due at midnight (Pig Latin)
- Also next Wednesday: last lecture (data provenance, data privacy)

Reading Assignment: "Rethinking the Contract"

- What is today's contract with the optimizer ?
- What are the main limitations in today's optimizers ?
- What is a "plan diagram" ?



Overview of Today's Lecture

- Parallel databases (Chapter 22.1 – 22.5)
- Map/reduce
- Pig-Latin
  - Some slides from Alan Gates (Yahoo!Research)
  - Mini-tutorial on the slides
  - Read manual for HW7
- Bloom filters
  - Use slides extensively !
  - Bloom joins are mentioned on pp. 746 in the book

Parallel v.s. Distributed Databases

- Parallel database system:
  - Improve performance through parallel implementation
  - Will discuss in class (and are on the final)

- Distributed database system:
  - Data is stored across several sites, each site managed by a DBMS capable of running independently
  - Will not discuss in class

Parallel DBMSs

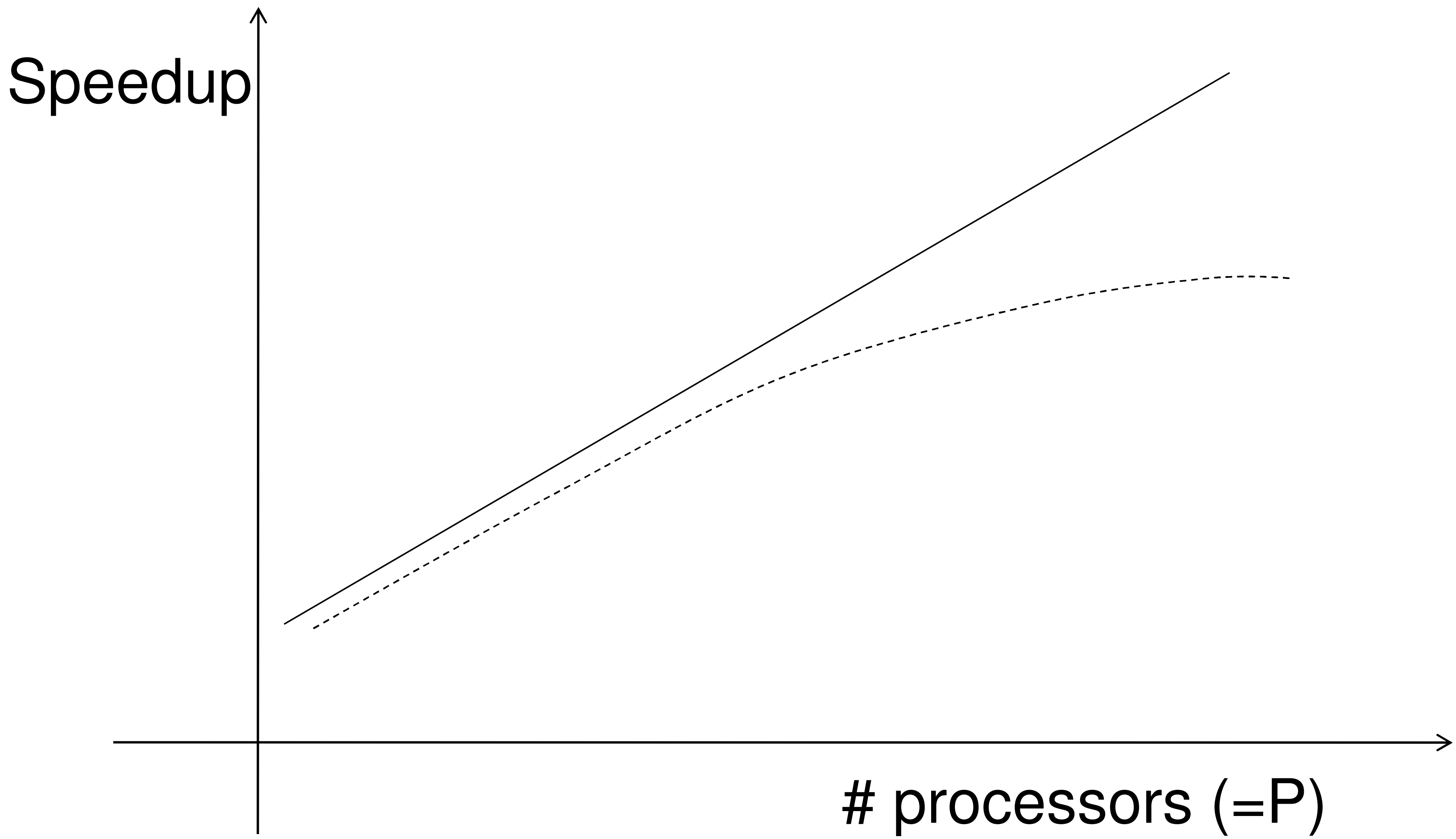
- Goal
  - Improve performance by executing multiple operations in parallel
- Key benefit
  - Cheaper to scale than relying on a single increasingly more powerful processor
- Key challenge
  - Ensure overhead and contention do not kill performance

Performance Metrics for Parallel DBMSs

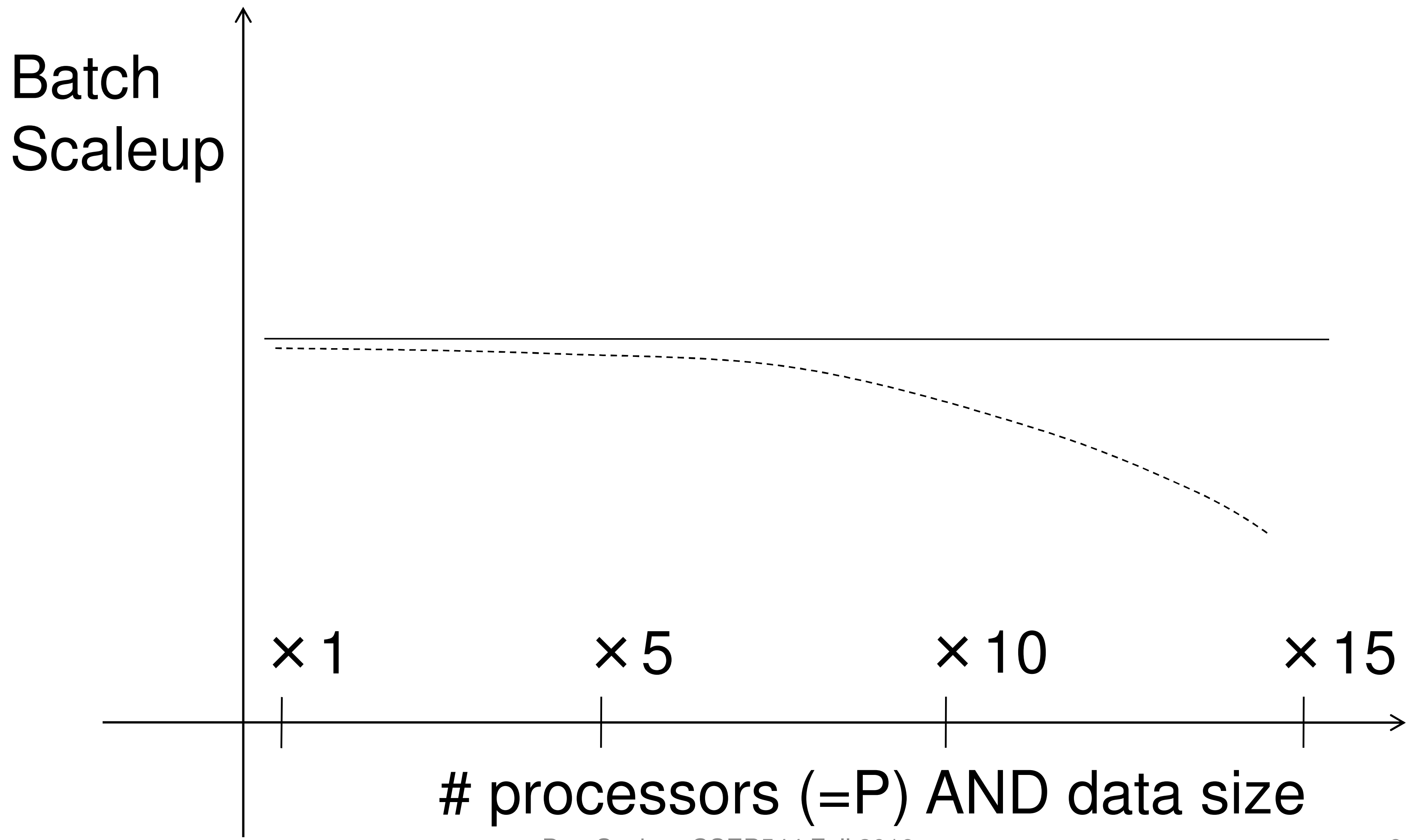
- Speedup
  - More processors ➔ higher speed
  - Individual queries should run faster
  - Should do more transactions per second (TPS)

- Scaleup
  - More processors ➔ can process more data
  - Batch scaleup
    - Same query on larger input data should take the same time
  - Transaction scaleup
    - N-times as many TPS on N-times larger database
    - But each transaction typically remains small

Linear v.s. Non-linear Speedup



Linear v.s. Non-linear Scaleup



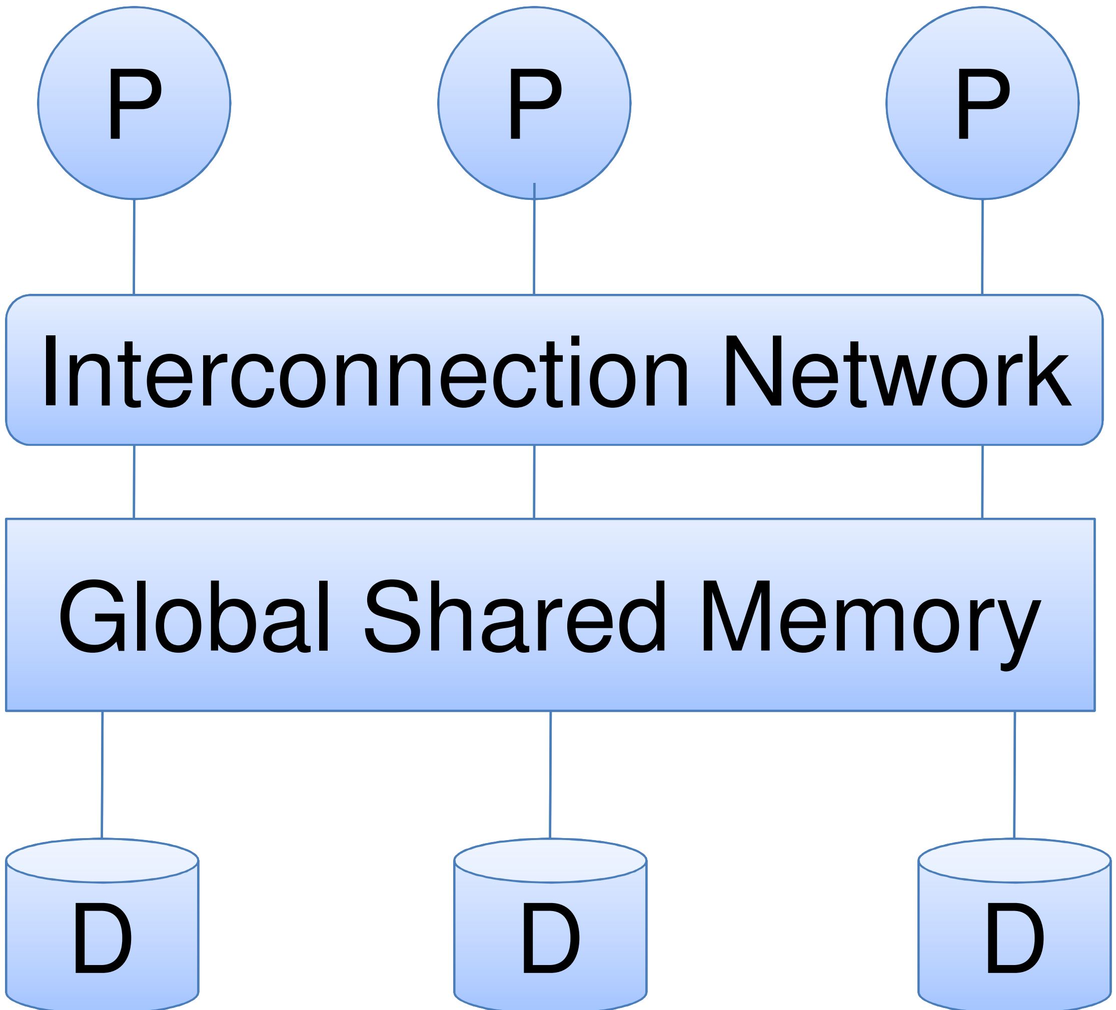
Challenges to Linear Speedup and Scaleup

- Startup cost
  – Cost of starting an operation on many processors
- Interference
  – Contention for resources between processors
- Skew
  – Slowest processor becomes the bottleneck

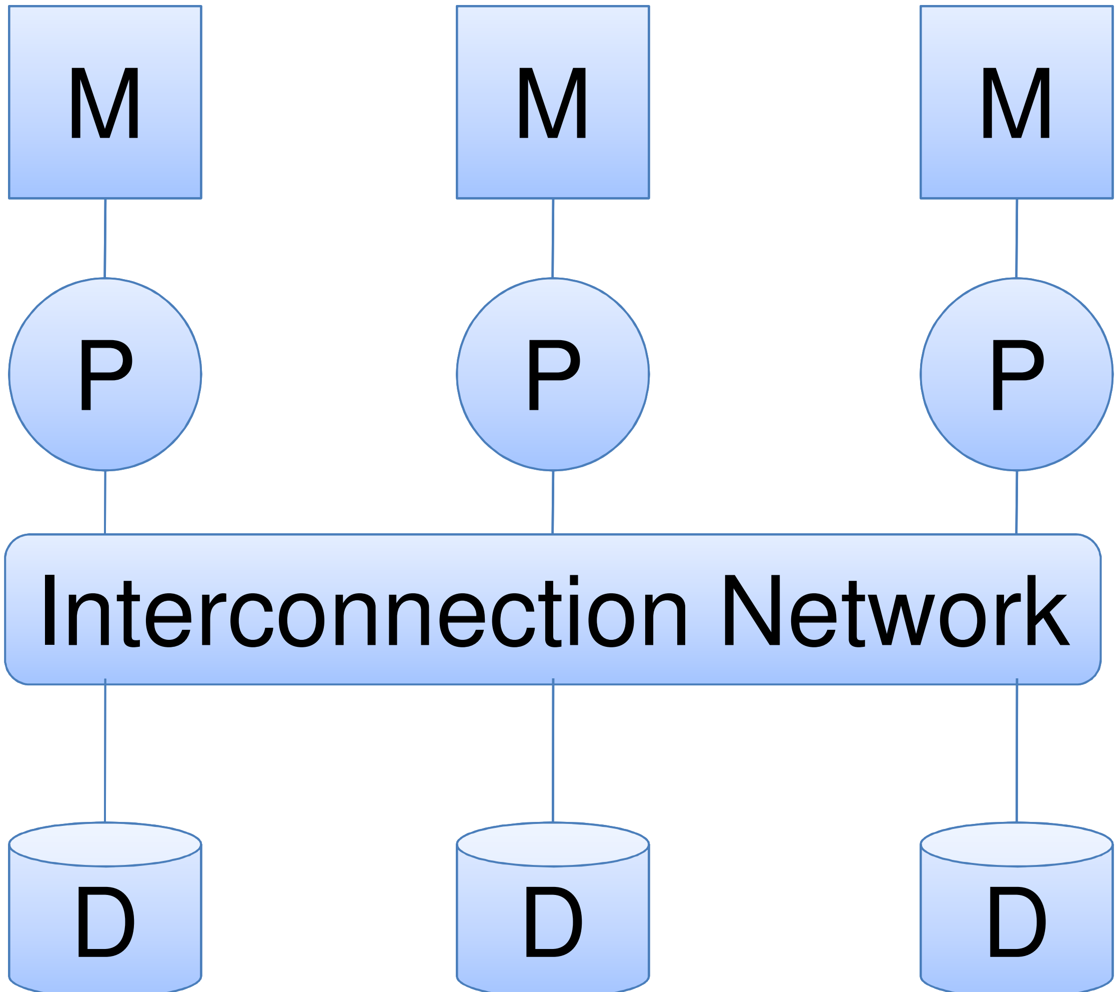
Architectures for Parallel Databases

- Shared memory
- Shared disk
- Shared nothing

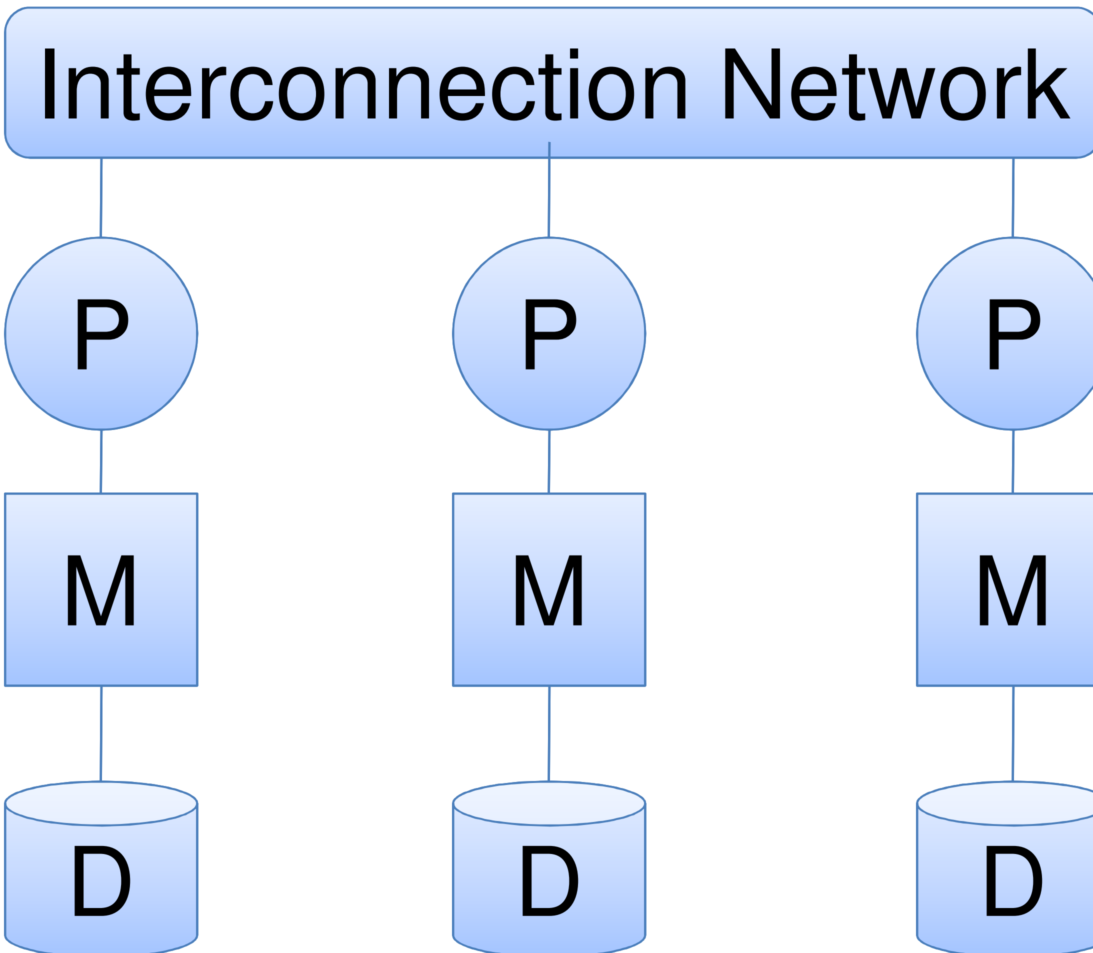
Shared Memory



Shared Disk



Shared Nothing



Shared Nothing

- Most scalable architecture
  - Minimizes interference by minimizing resource sharing
  - Can use commodity hardware

- Also most difficult to program and manage

- Processor = server = node
- P = number of nodes

We will focus on shared nothing

Taxonomy for Parallel Query Evaluation

- Inter-query parallelism
  - Each query runs on one processor
- Inter-operator parallelism
  - A query runs on multiple processors
  - An operator runs on one processor
- Intra-operator parallelism
  - An operator runs on multiple processors

We study only intra-operator parallelism: most scalable

Horizontal Data Partitioning

- Relation R split into P chunks $R_0, \ldots, R_{P-1}$, stored at the P nodes
- Round robin: tuple t_i to chunk (i mod P)
- Hash based partitioning on attribute A:
  - Tuple t to chunk h(t.A) mod P
- Range based partitioning on attribute A:
  - Tuple t to chunk i if $v_{i-1} < t.A < v_i$

Parallel Selection

Compute $\sigma_{A=v}(R)$, or $\sigma_{v1<A<v2}(R)$

- Conventional database:
  - Cost = B(R)
- Parallel database with P processors:
  - Cost = B(R) / P

Parallel Selection

Different processors do the work:

- Round robin partition: all servers do the work
- Hash partition:
  - One server for $\sigma_{A=v}(R)$,
  - All servers for $\sigma_{v1<A<v2}(R)$
- Range partition: one server does the work

Data Partitioning Revisited

What are the pros and cons ?

- Round robin
  - Good load balance but always needs to read all the data
- Hash based partitioning
  - Good load balance but works only for equality predicates and full scans
- Range based partitioning
  - Works well for range predicates but can suffer from data skew

Parallel Group By: $\gamma_{A,\ sum(B)}(R)$

Step 1: server i partitions chunk R_i using a hash function h(t.A): R_{i0}, R_{i1}, …, $R_{i,P-1}$

Step 2: server i sends partition R_{ij} to server j

Step 3: server j computes $\gamma_{A,\ sum(B)}$ on R_{0j}, R_{1j}, …, $R_{P-1,j}$

Cost of Parallel Group By

Recall conventional cost = 3B(R)

- Step 1: Cost = B(R)/P I/O operations
- Step 2: Cost = (P-1)/P B(R) blocks are sent
  - Network costs << I/O costs
- Step 3: Cost = 2 B(R)/P
  - When can we reduce it to 0 ?

Total = 3B(R) / P + communication costs

Parallel Join: $R \bowtie_{A=B} S$

Step 1

- For all servers in [0,k], server i partitions chunk R_i using a hash function h(t.A): R_{i0}, R_{i1}, …, $R_{i,P-1}$
- For all servers in [k+1,P], server j partitions chunk S_j using a hash function h(t.A): S_{j0}, S_{j1}, …, $R_{j,P-1}$

Step 2:

- Server i sends partition R_{iu} to server u
- Server j sends partition S_{ju} to server u

Steps 3: Server u computes the join of R_{iu} with S_{ju}

Cost of Parallel Join

- Step 1: Cost = (B(R) + B(S))/P
- Step 2: 0
  - (P-1)/P (B(R) + B(S)) blocks are sent, but we assume network costs to be << disk I/O costs
- Step 3:
  - Cost = 0 if small table fits in memory: B(S)/P <=M
  - Cost = 4(B(R)+B(S))/P otherwise

Parallel Query Plans

- Same relational operators
- Add special split and merge operators
  - Handle data routing, buffering, and flow control
- Example: exchange operator
  - Inserted between consecutive operators in the query plan

Map Reduce

- Google: paper published 2004
- Free variant: Hadoop

- Map-reduce = high-level programming model and implementation for large-scale parallel data processing

Data Model

Files !

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

A map-reduce program:

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

Step 1: the MAP Phase

User provides the MAP-function:

- Input: one `(input key, value)`
- Ouput: bag of `(intermediate key, value)` pairs

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

Step 2: the REDUCE Phase

User provides the REDUCE function:

- Input: `(intermediate key, bag of values)`
- Output: bag of output `values`

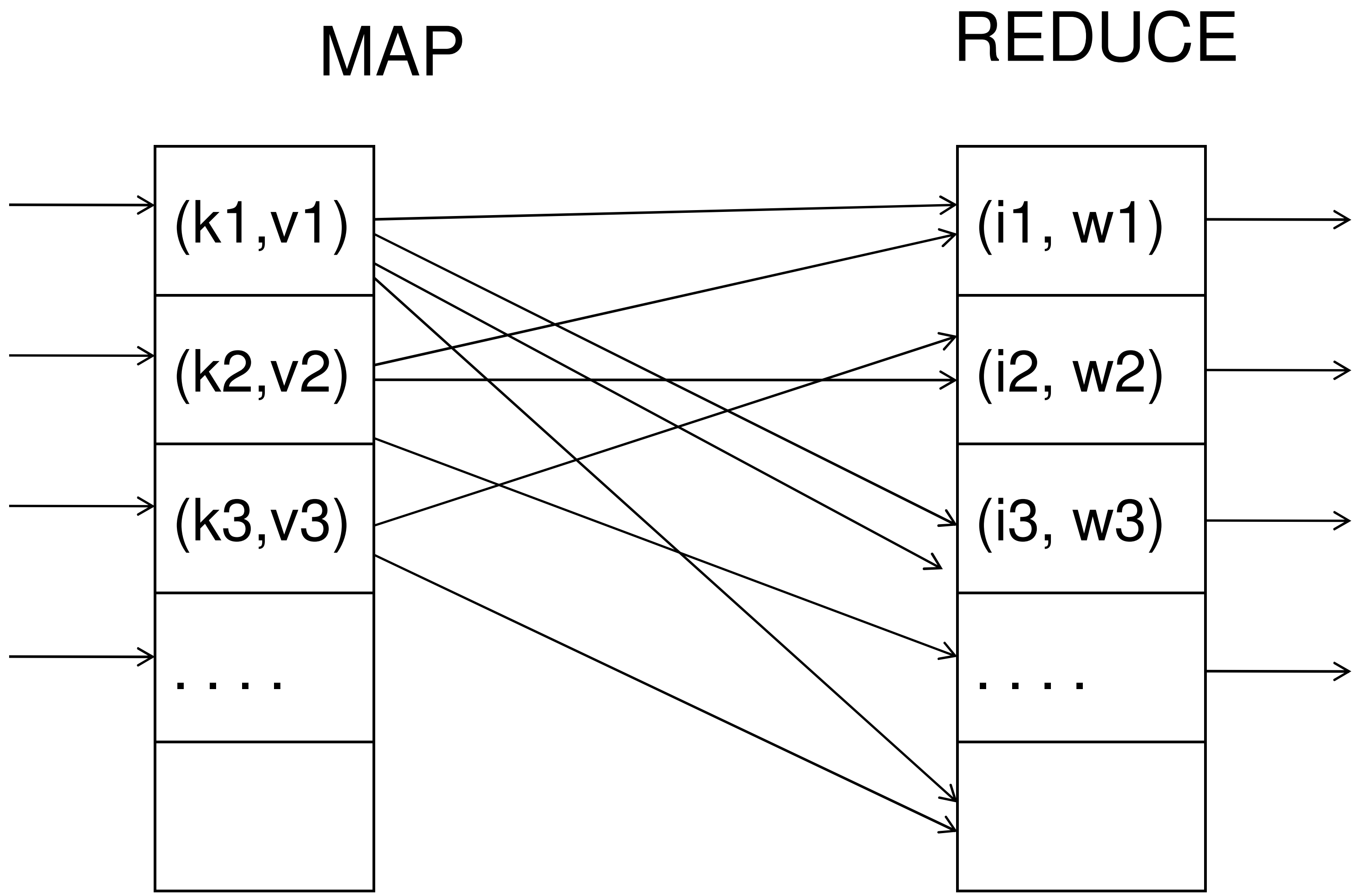
System groups all pairs with the same intermediate key, and passes the bag of values to the REDUCE function

Example

- Counting the number of occurrences of each word in a large collection of

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

```
reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```


MAP
REDUCE
(k1,v1)
(k2,v2)
(k3,v3)
. . . .
(i1, w1)
(i2, w2)
(i3, w3)
. . . .

Map = GROUP BY, Reduce = Aggregate

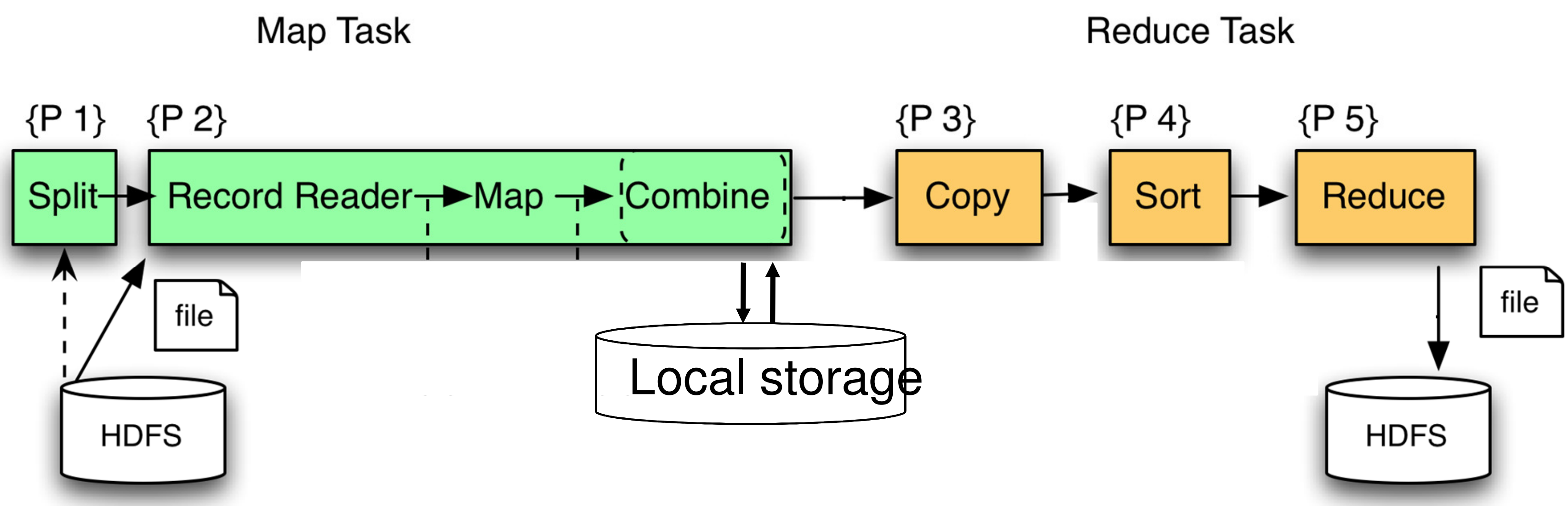
R(documentKey, word)

```
SELECT word, sum(1)
FROM R
GROUP BY word
```

Implementation

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

MR Phases



Map Task
Reduce Task
{P 1}
{P 2}
{P 3}
{P 4}
{P 5}
Split
Record Reader
Map
Combine
Copy
Sort
Reduce
file
HDFS
Local storage
file
HDFS

Interesting Implementation Details

- Worker failure:
  - Master pings workers periodically,
  - If down then reassigns its splits *to all other* workers → good load balance
- Choice of M and R:
  - Larger is better for load balancing
  - Limitation: master needs $O(M \times R)$ memory

Interesting Implementation Details

Backup tasks:

- *Straggler* = a machine that takes unusually long time to complete one of the last tasks. Eg:
  - Bad disk forces frequent correctable errors (30MB/s → 1MB/s)
  - The cluster scheduler has scheduled other tasks on that machine
- Stragglers are a main reason for slowdown
- Solution*: pre-emptive backup execution of the last few remaining in-progress tasks*

Map-Reduce Summary

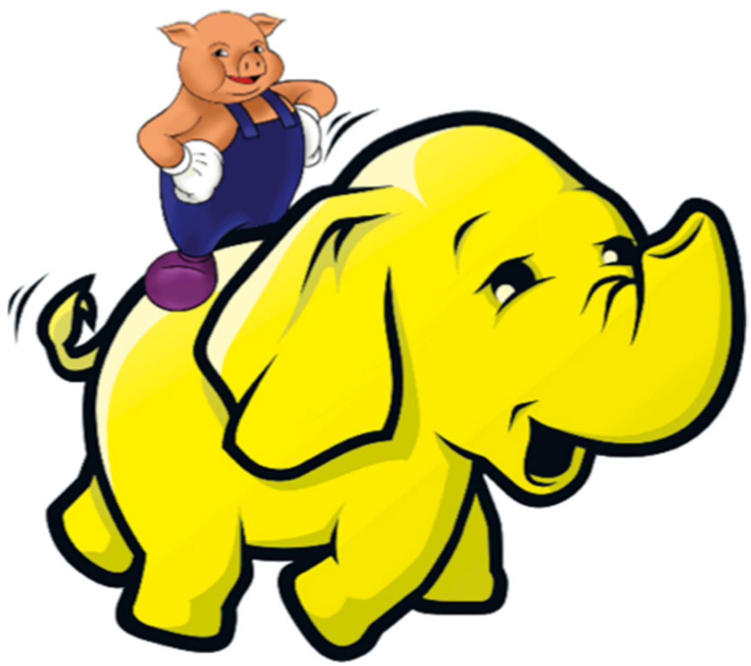
- Hides scheduling and parallelization details
- However, very limited queries
  - Difficult to write more complex tasks
  - Need multiple map-reduce operations
- Solution:

PIG-Latin !

Following Slides provided by:
Alan Gates, Yahoo!Research

What is Pig?

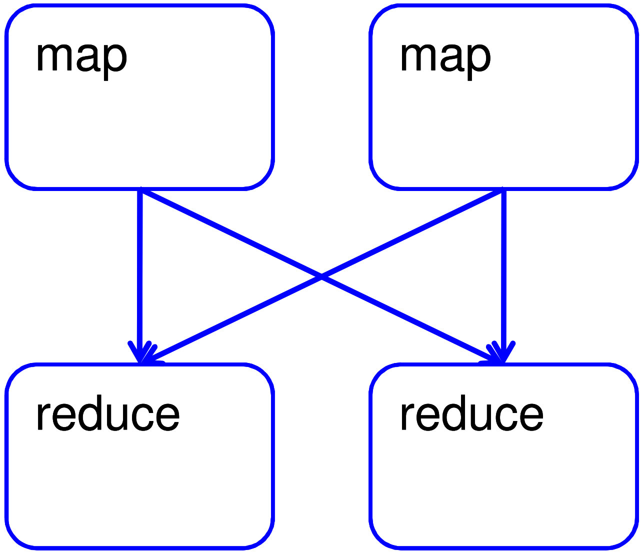
- An engine for executing programs on top of Hadoop
- It provides a language, Pig Latin, to specify these programs
- An Apache open source project
  http://hadoop.apache.org/pig/



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.

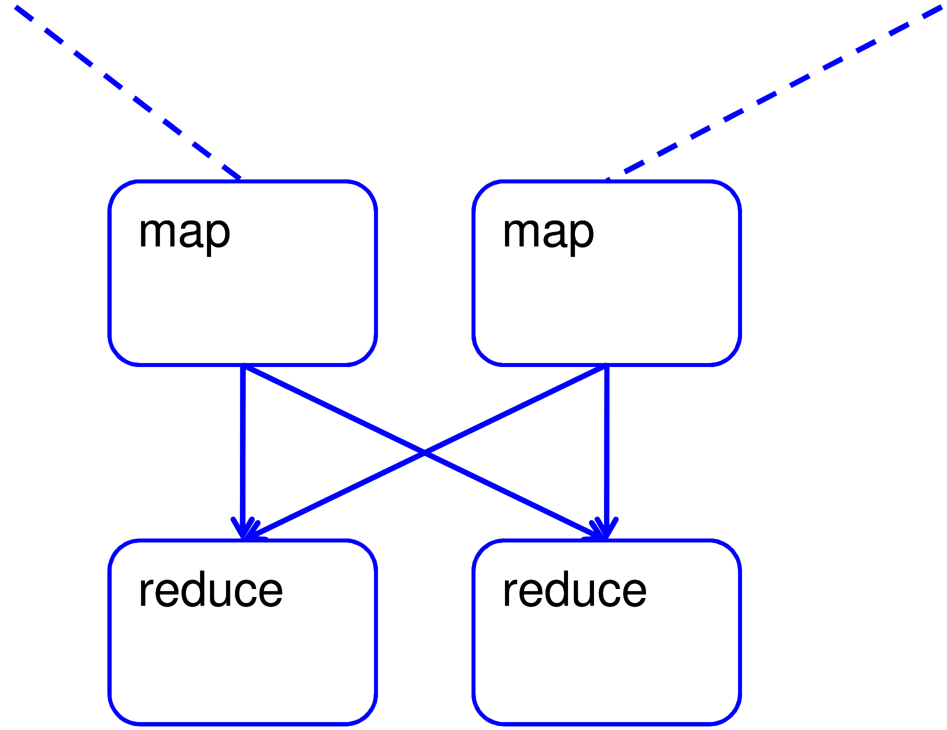
Map Reduce Illustrated



Map Reduce Illustrated

Romeo, Romeo, wherefore art thou Romeo?

What, art thou hurt?

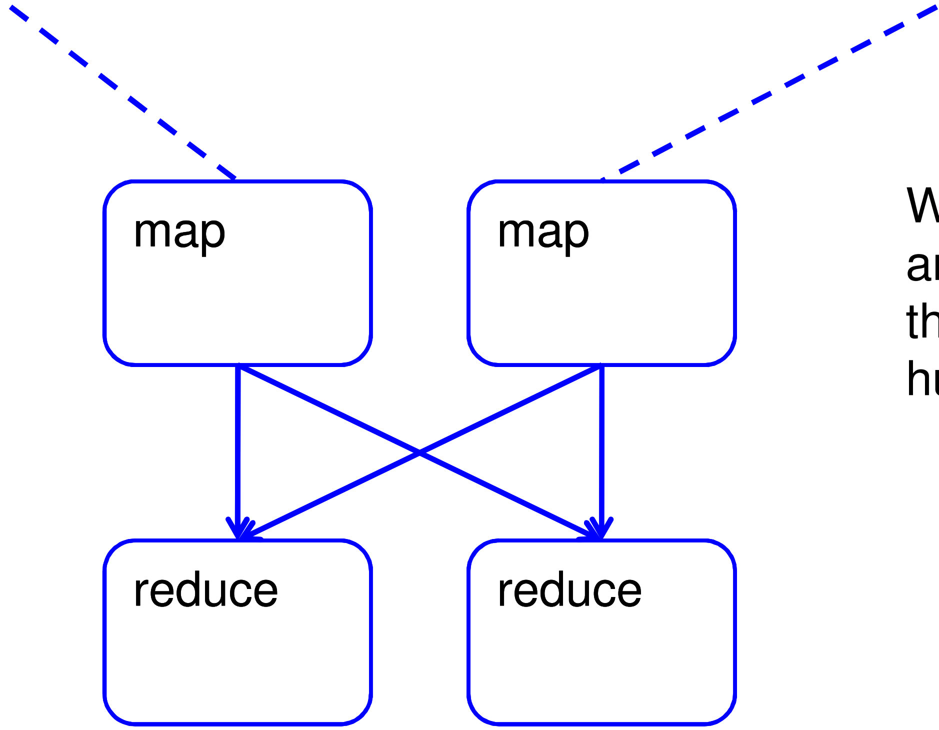


Map Reduce Illustrated

Romeo, Romeo, wherefore art thou Romeo?

What, art thou hurt?

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



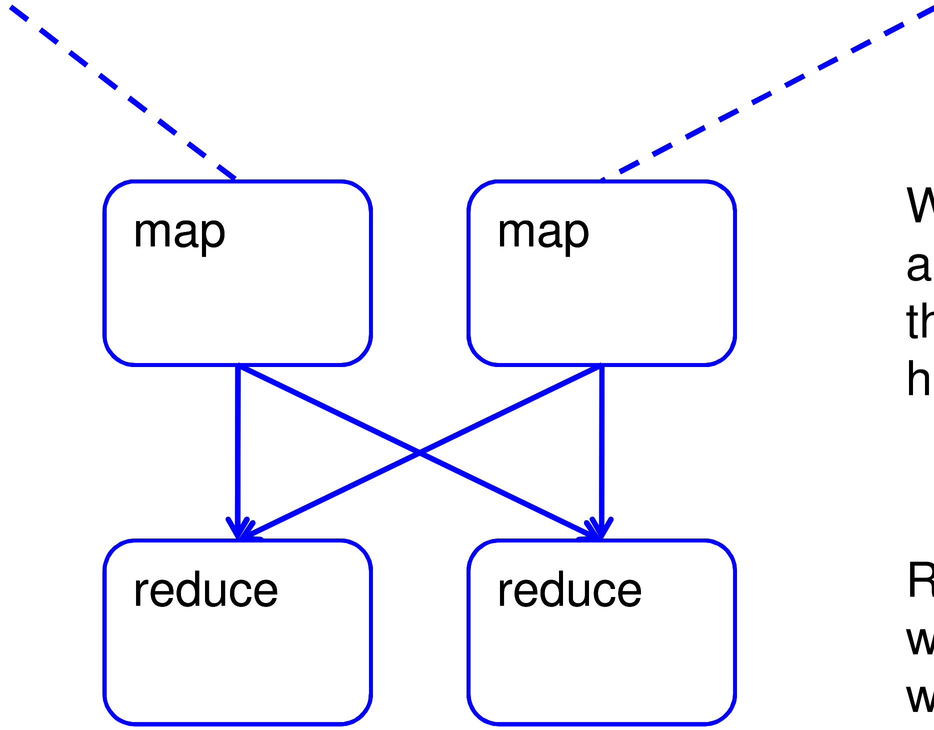
What, 1
art, 1
thou, 1
hurt, 1

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



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

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

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

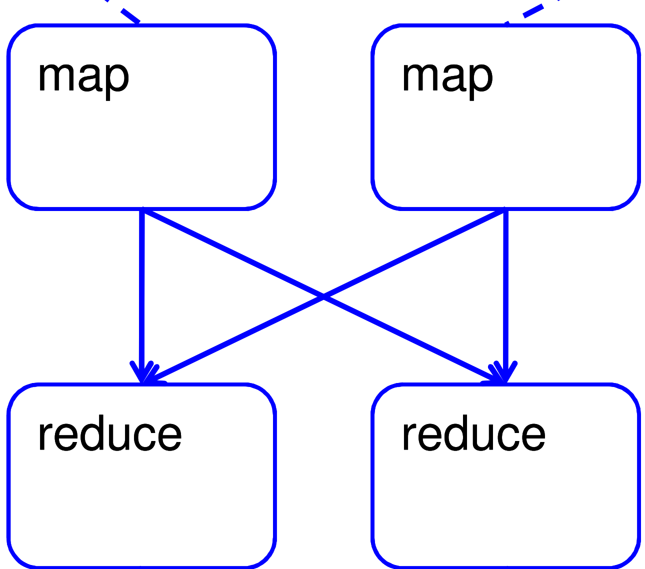
Map Reduce Illustrated

Romeo, Romeo, wherefore art thou Romeo?

What, art thou hurt?

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

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



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

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

art, 2
hurt, 1
thou, 2

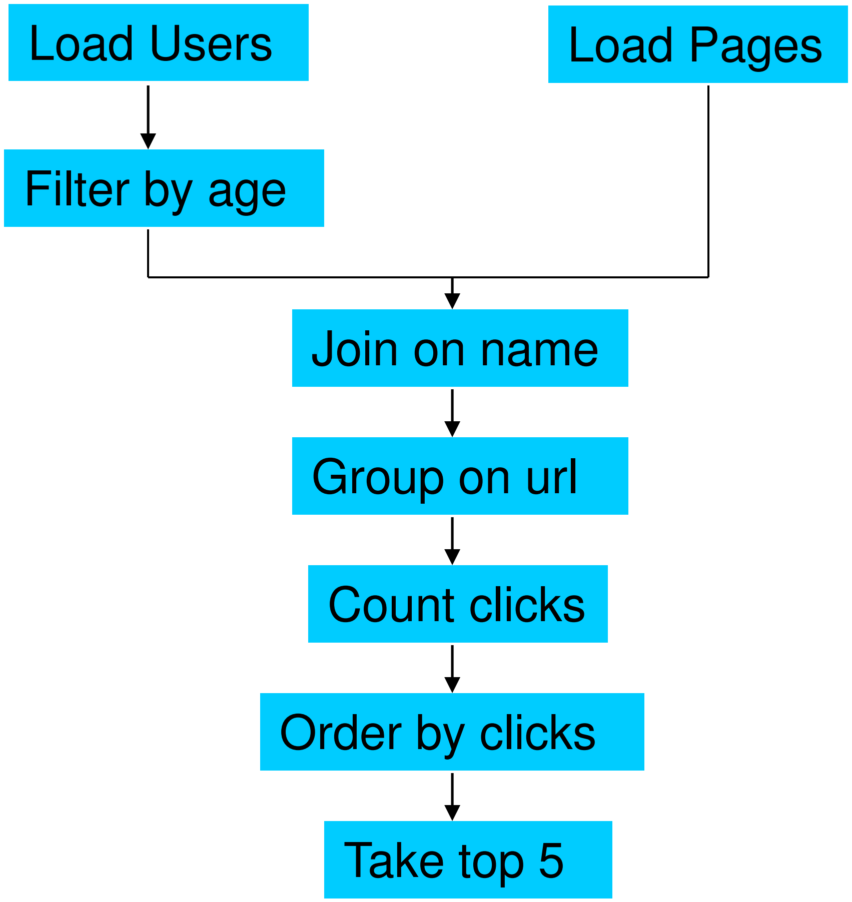
Romeo, 3
wherefore, 1
what, 1

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.



In Map-Reduce

```
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.a    pache.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;
imp ort 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.JobC              ontrol;
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.sub            string(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 w                e 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.to           String();
                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,
                OutputColle         ctor<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', first                  Comma);
            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 ke        y,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

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

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        i   mplements 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.se    tJobName("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.s    etJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.add         InputPath(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(IdentityMap            per.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.se         tOutputPath(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(MRE            xample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFi            leOutputFormat.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(SequenceFileOutputF                ormat.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();
    }
}
```

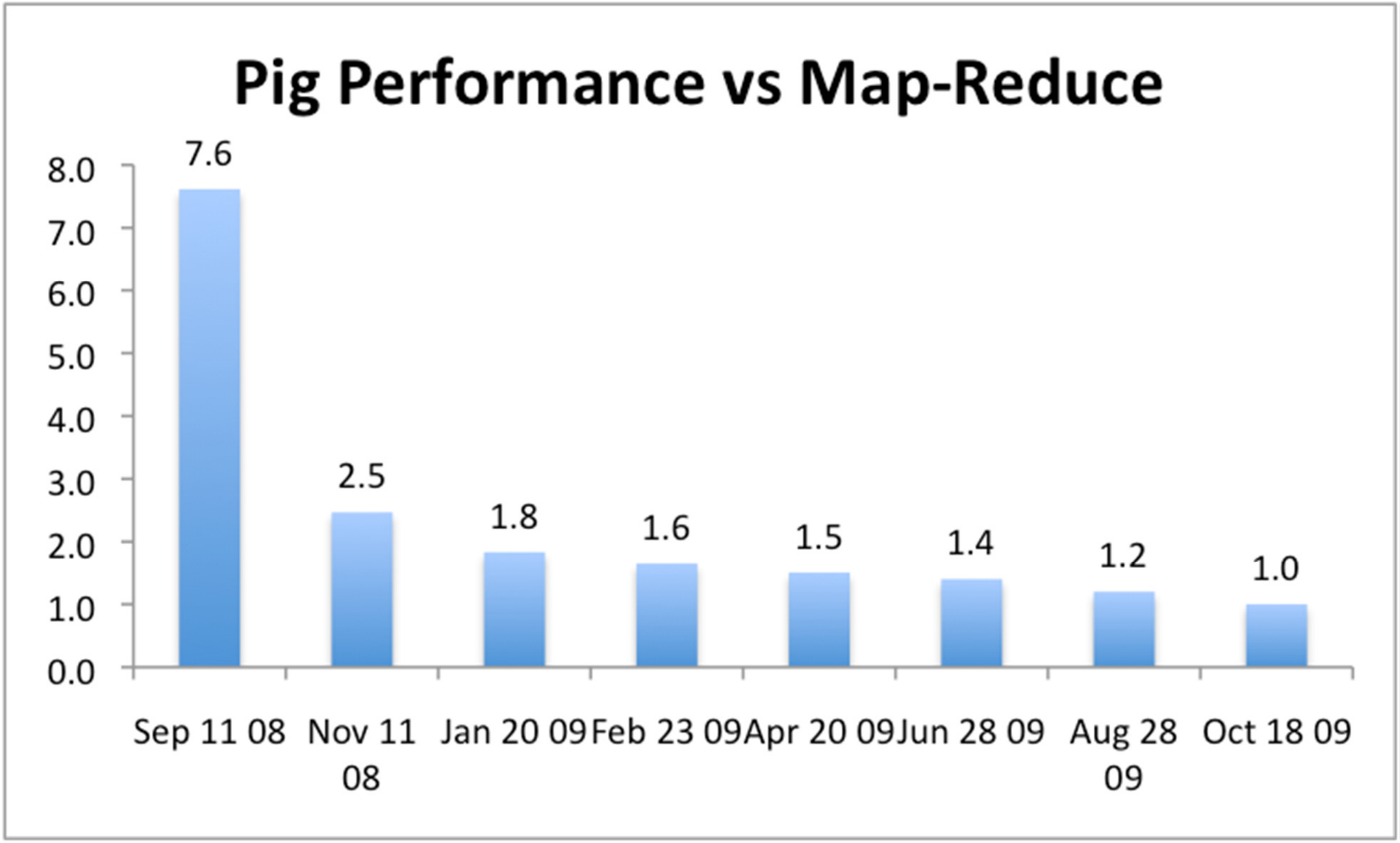
170 lines of code, 4 hours to write

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

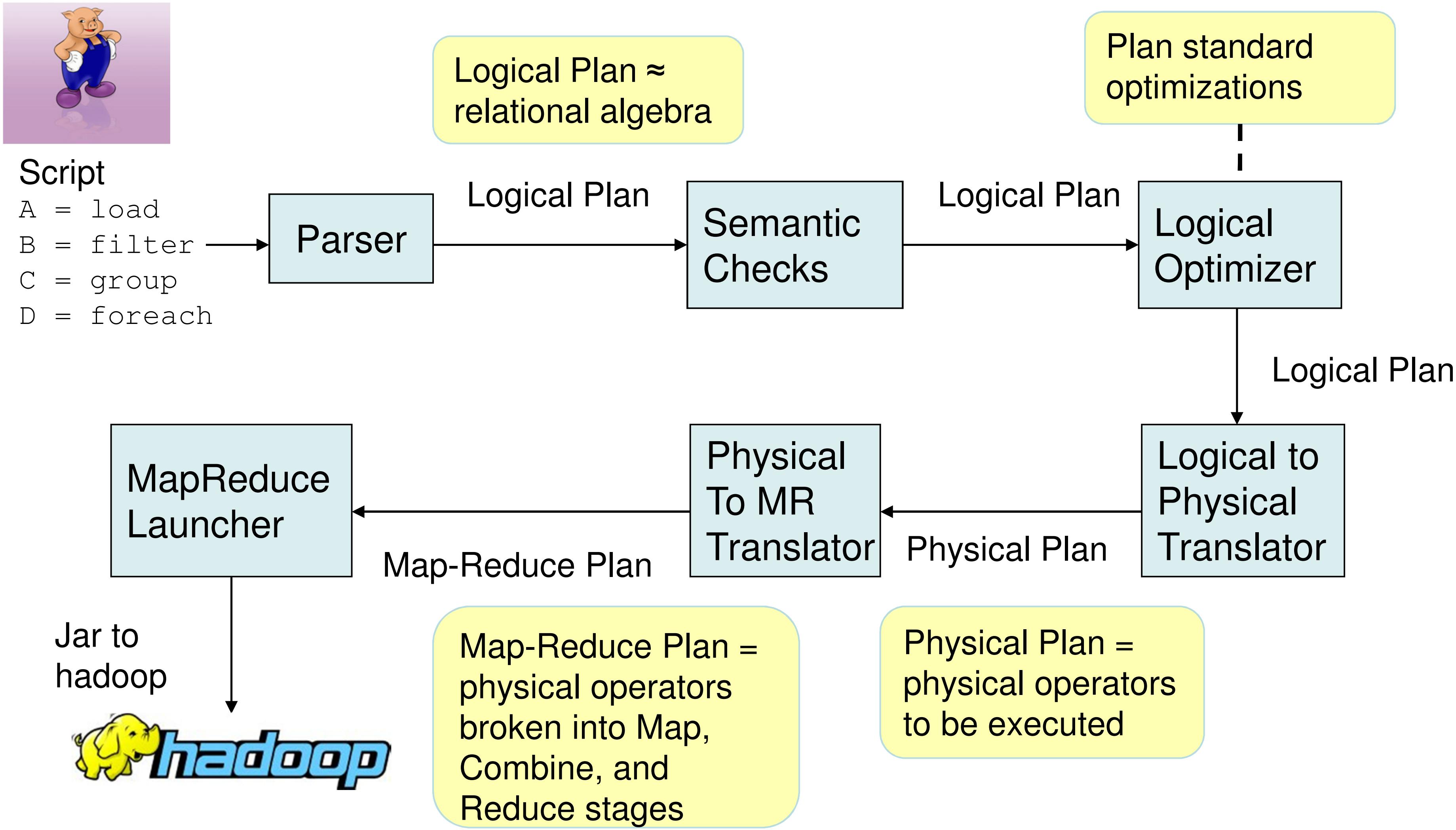
But can it fly?



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

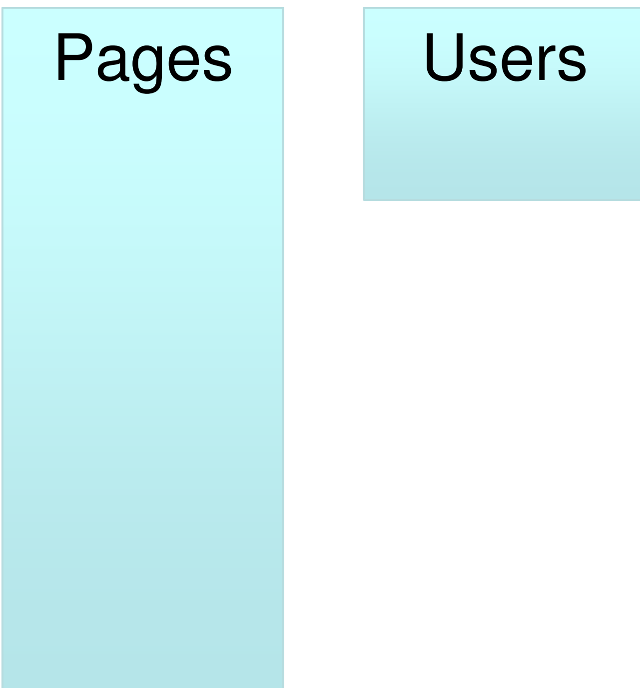

Logical Plan ≈ relational algebra
Plan standard optimizations
Script
A = load
B = filter
C = group
D = foreach
Parser
Logical Plan
Semantic Checks
Logical Plan
Logical Optimizer
Logical Plan
Logical to Physical Translator
Physical Plan
Physical To MR Translator
Map-Reduce Plan
MapReduce Launcher
Jar to hadoop
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

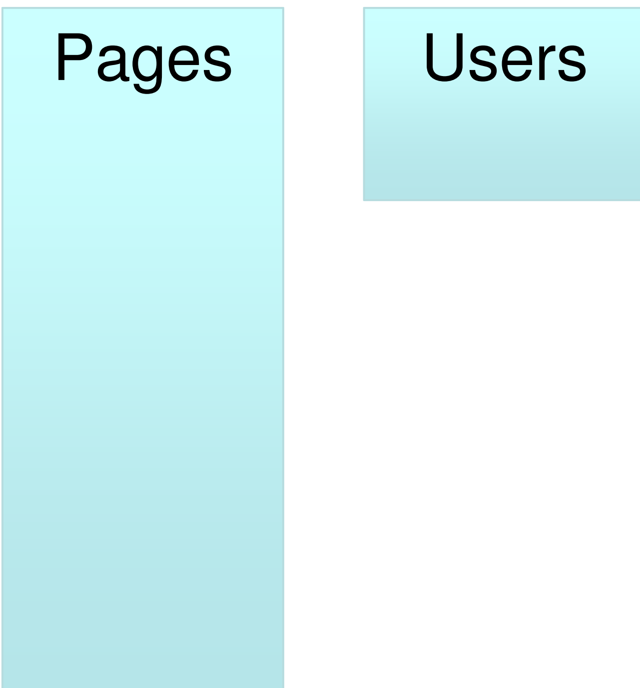
Aka
“Broakdcast Join”



Fragment Replicate Join

Aka
“Broakdcast Join”

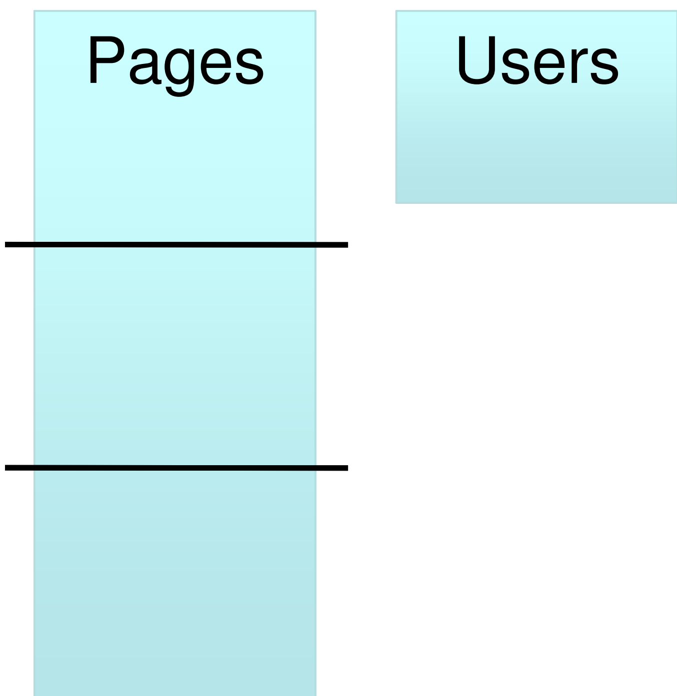
```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Pages by user, Users by name using "replicated";
```



Fragment Replicate Join

Aka
“Broakdcast Join”

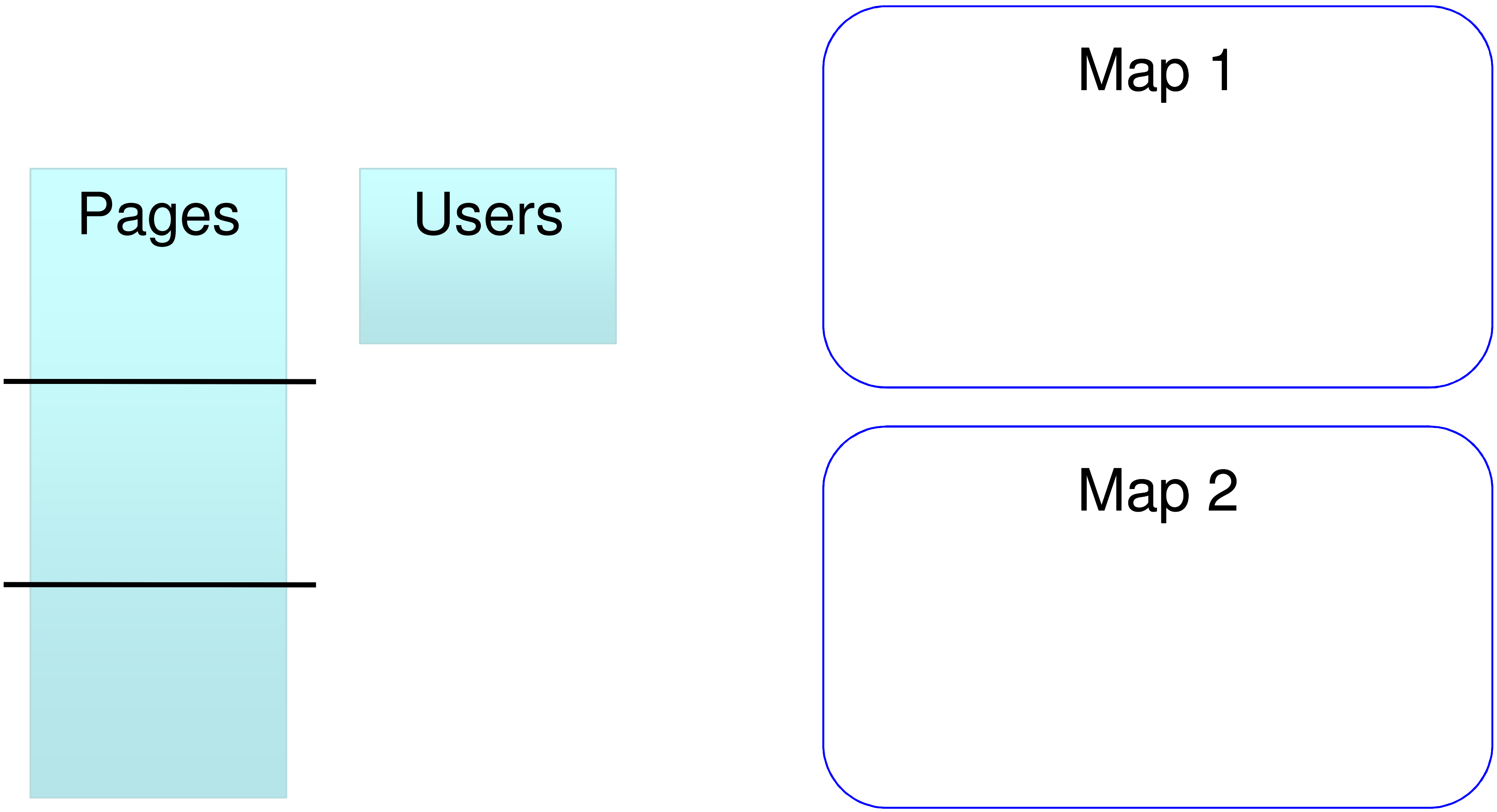
```
Users = load ‘users’ as (name, age);
Pages = load ‘pages’ as (user, url);
Jnd = join Pages by user, Users by name using “replicated”;
```



Fragment Replicate Join

Aka
“Broakdcast Join”

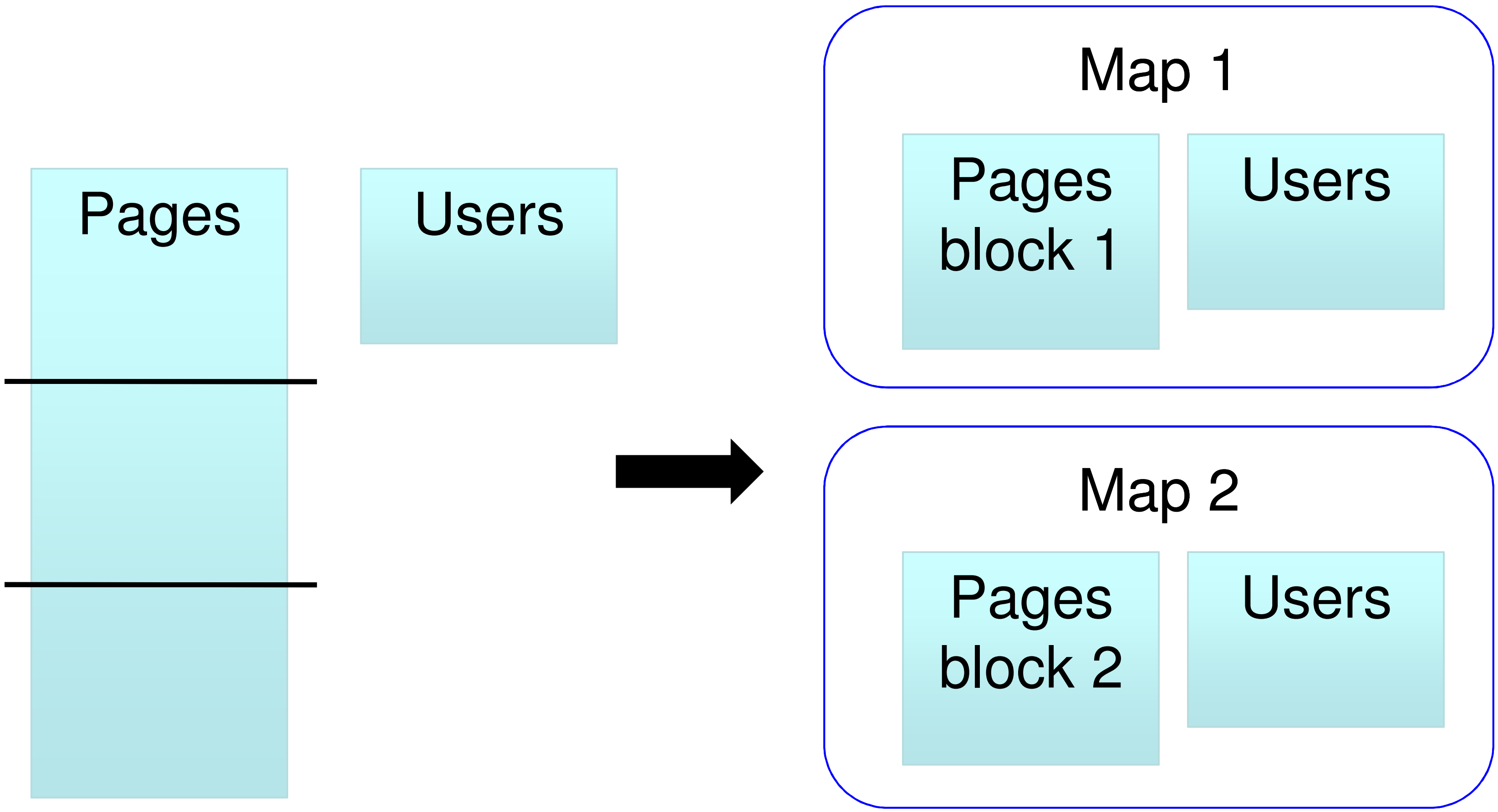
```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Pages by user, Users by name using "replicated";
```



Fragment Replicate Join

Aka
“Broakdcast Join”

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



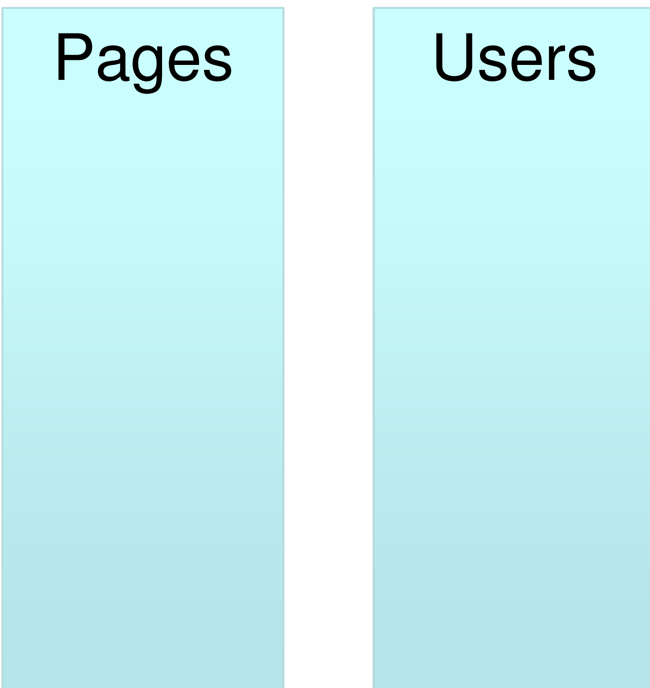
Hash Join

Pages

Users

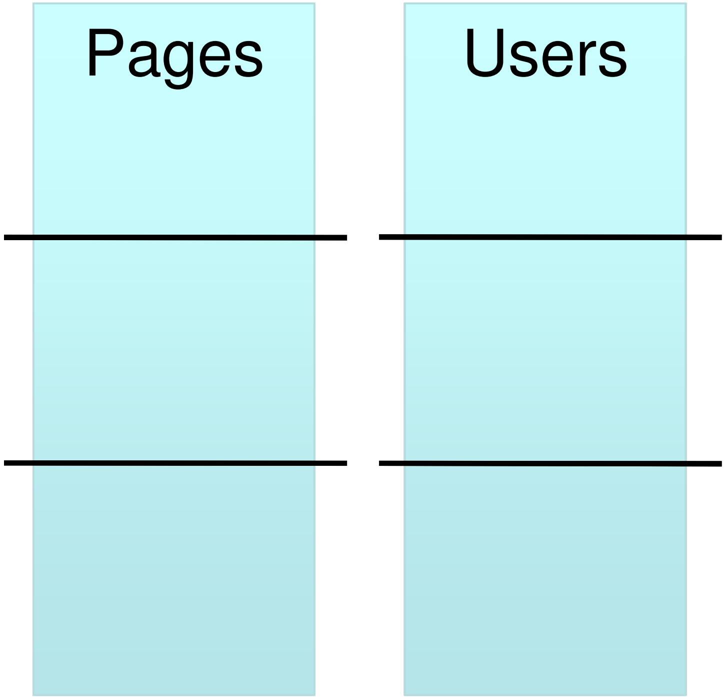
Hash Join

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



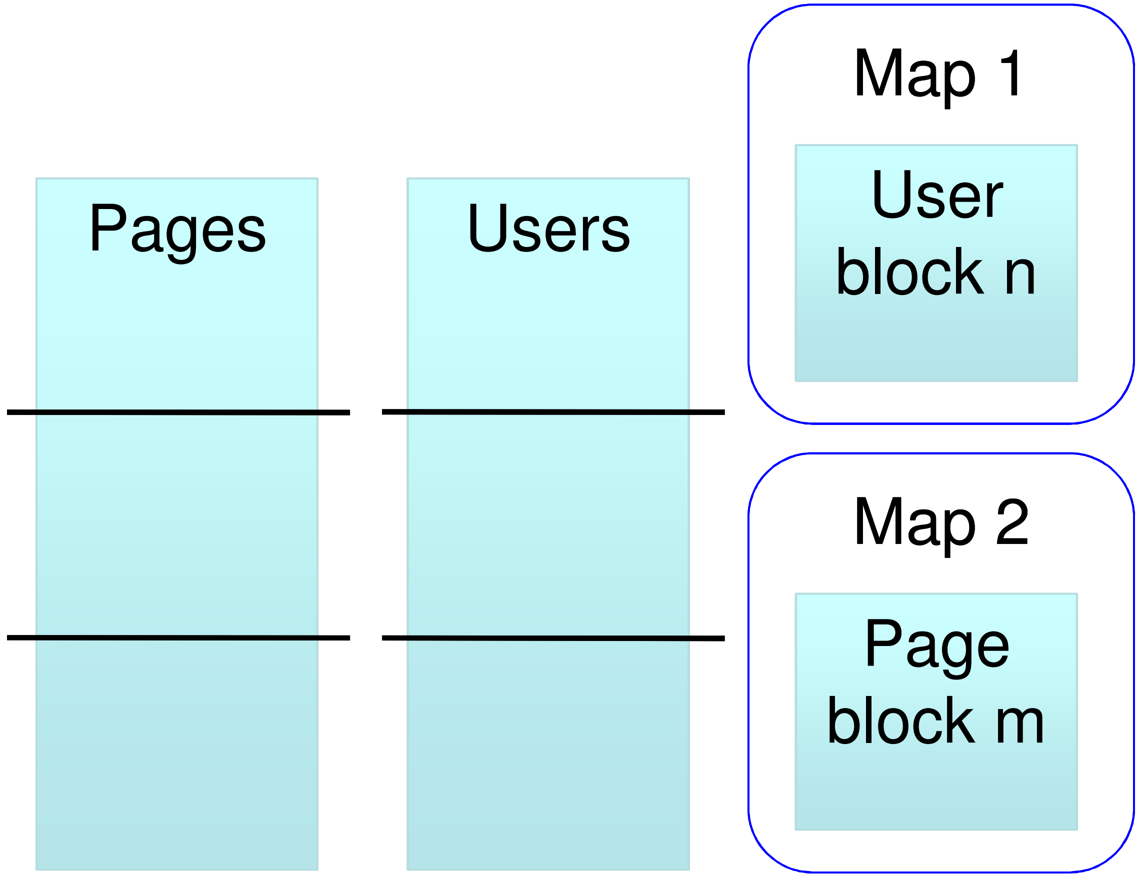
Hash Join

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



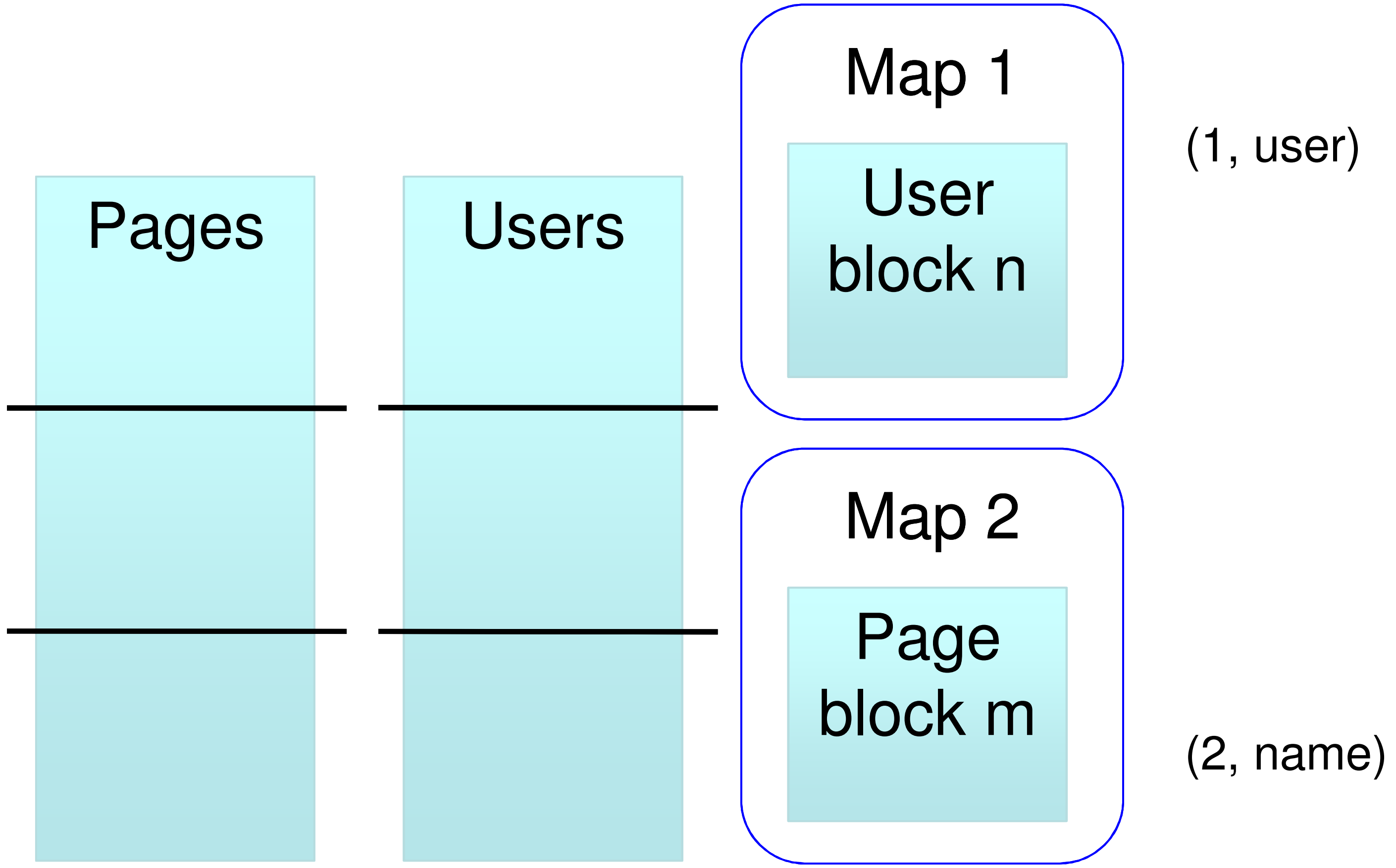
Hash Join

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



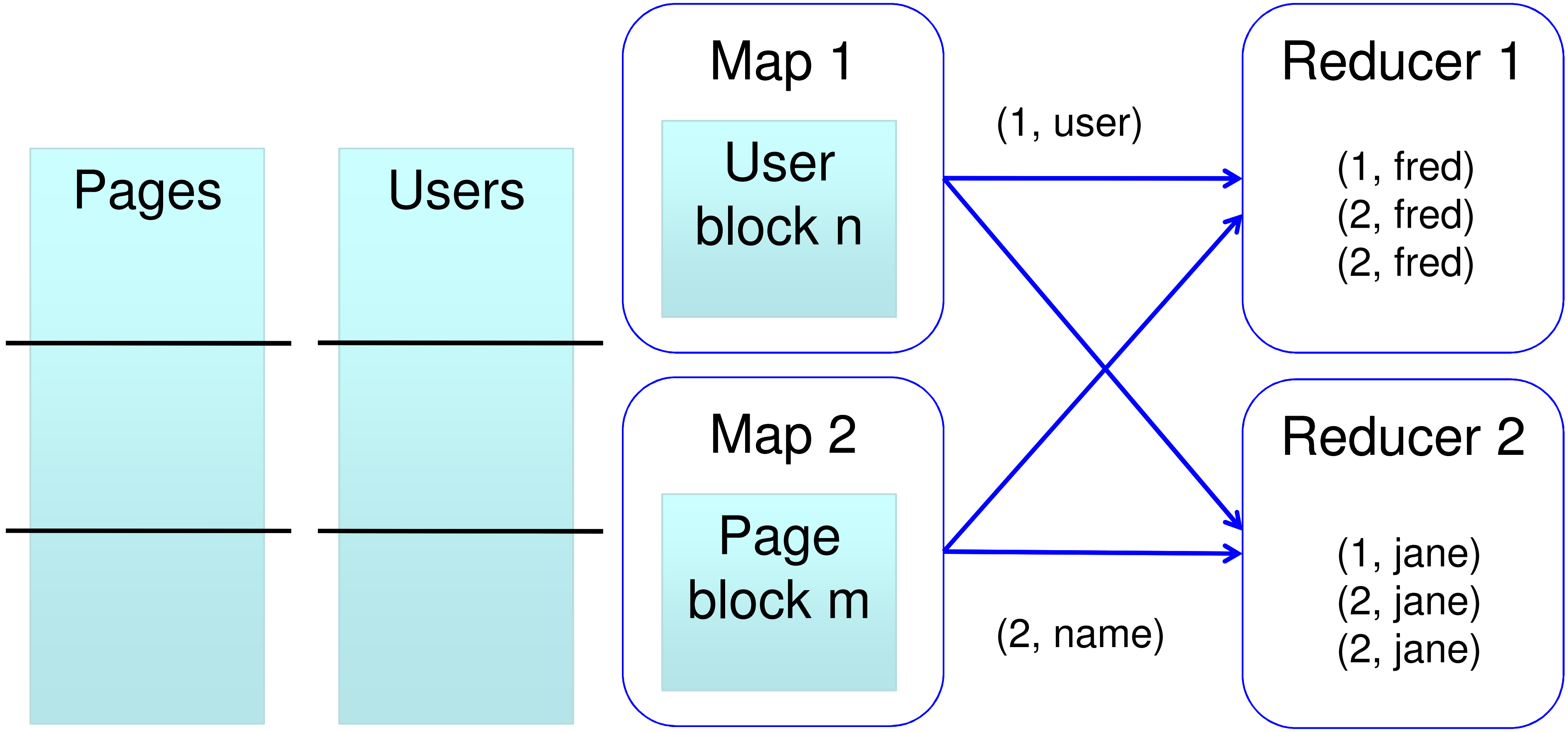
Hash Join

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

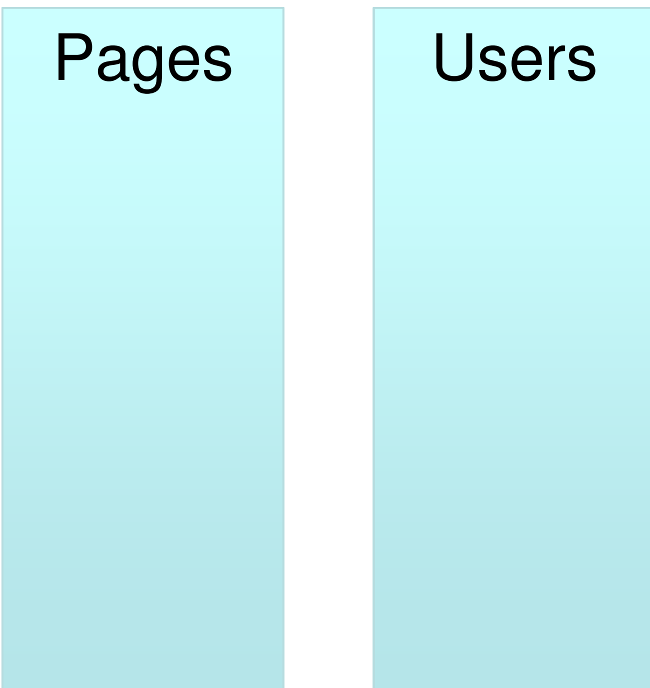


Hash Join

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

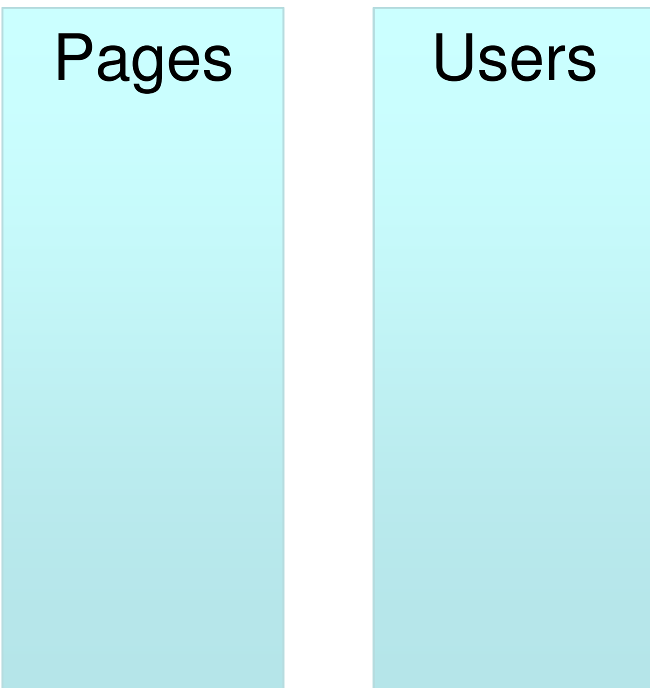


Skew Join



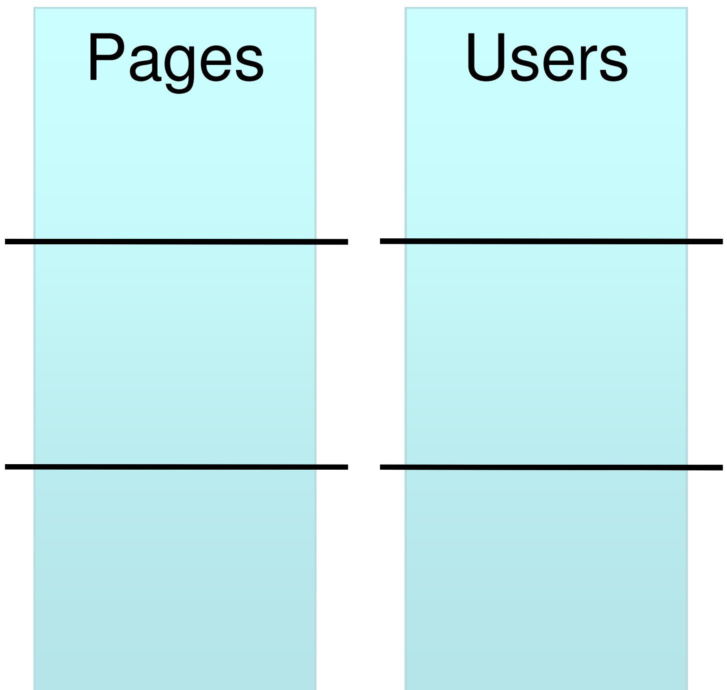
Skew Join

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



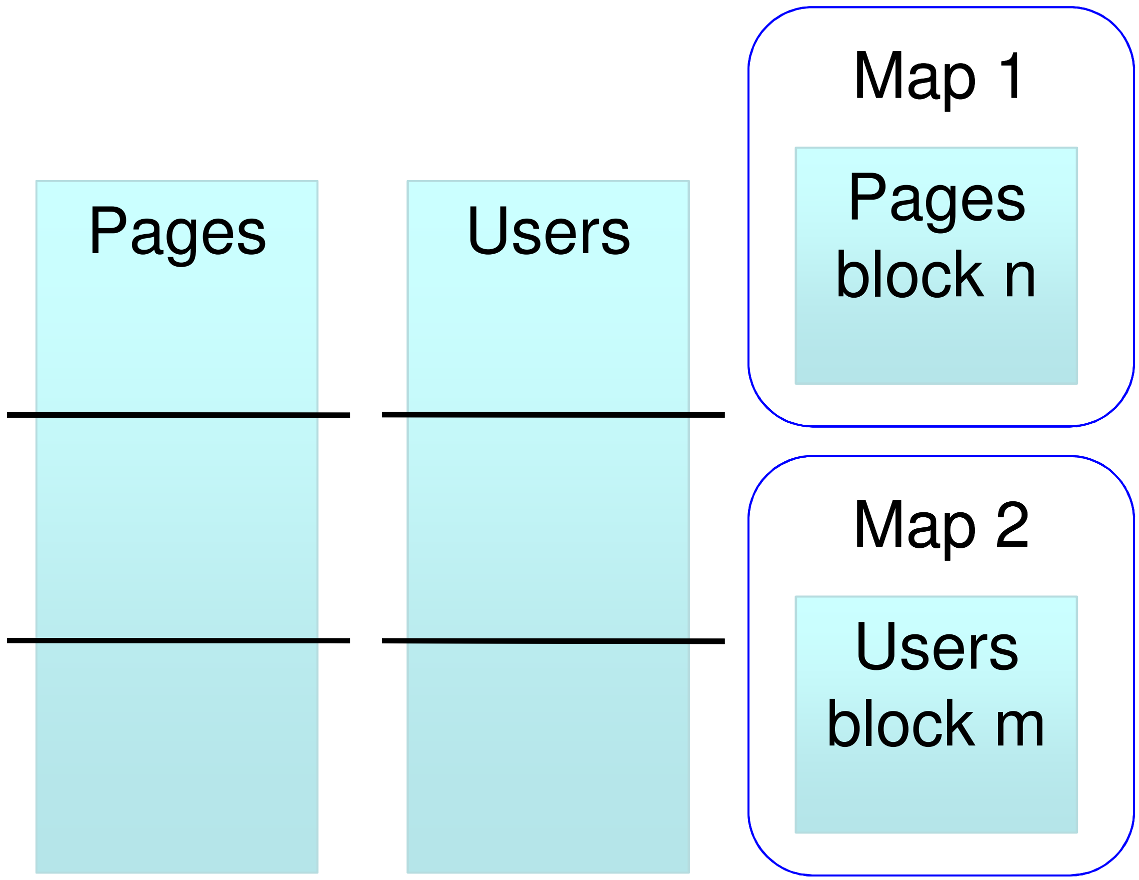
Skew Join

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



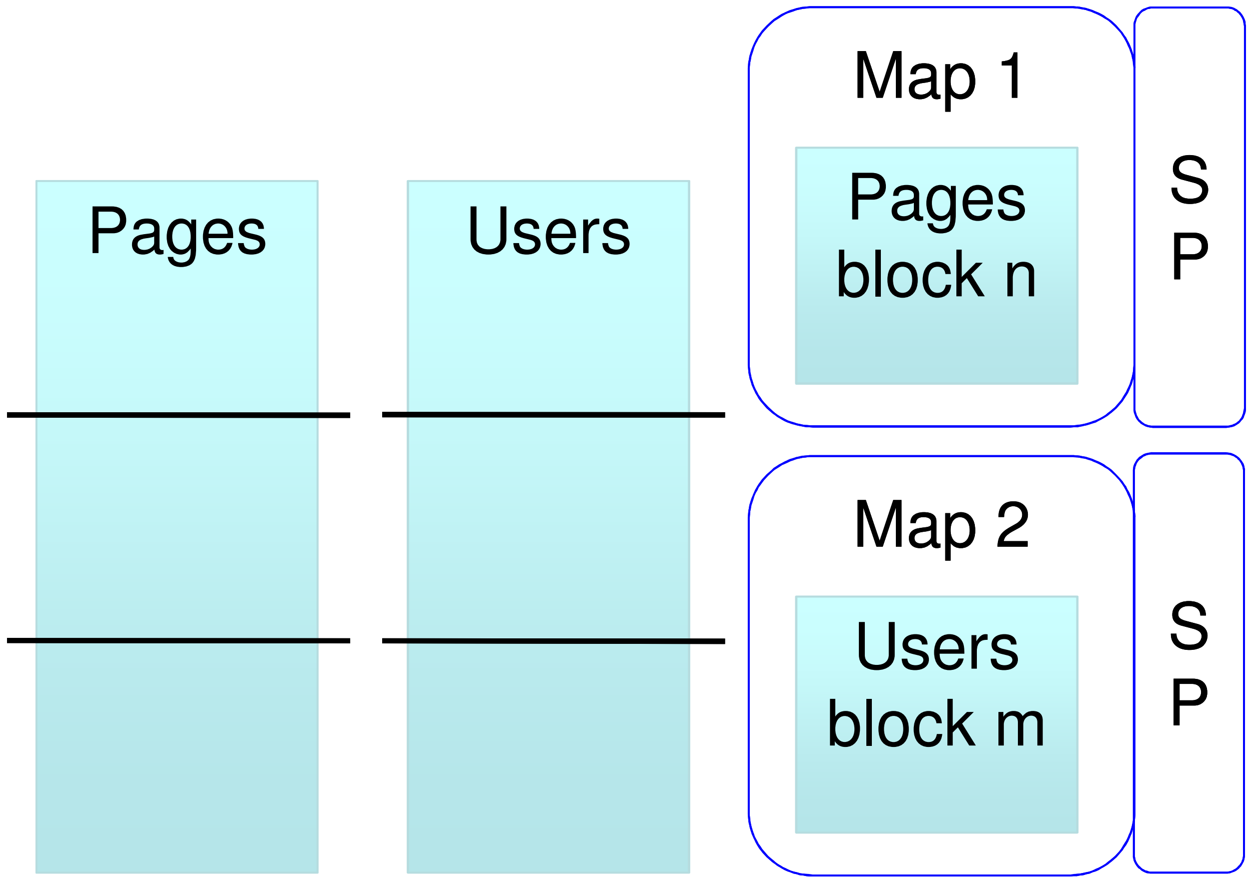
Skew Join

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



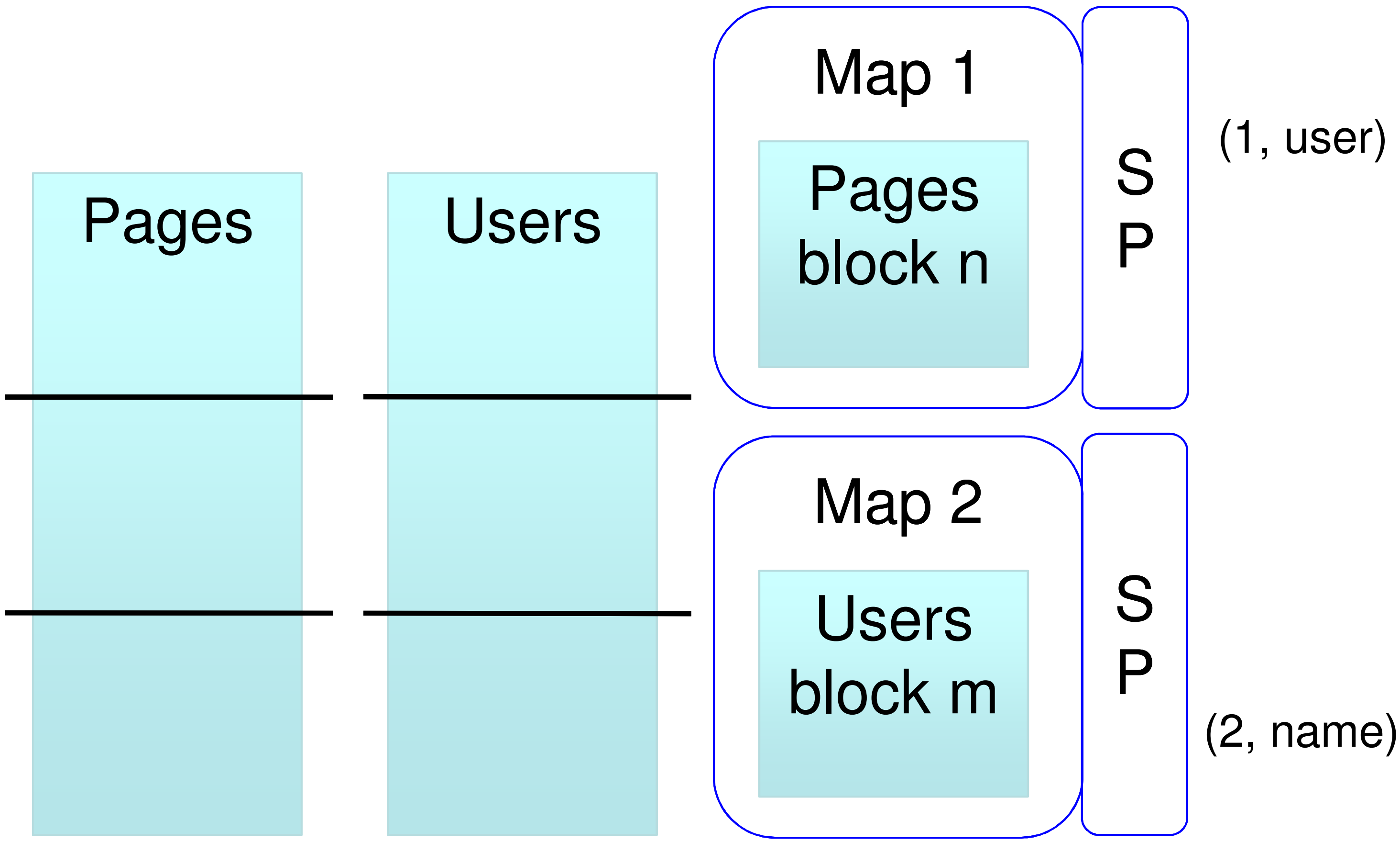
Skew Join

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



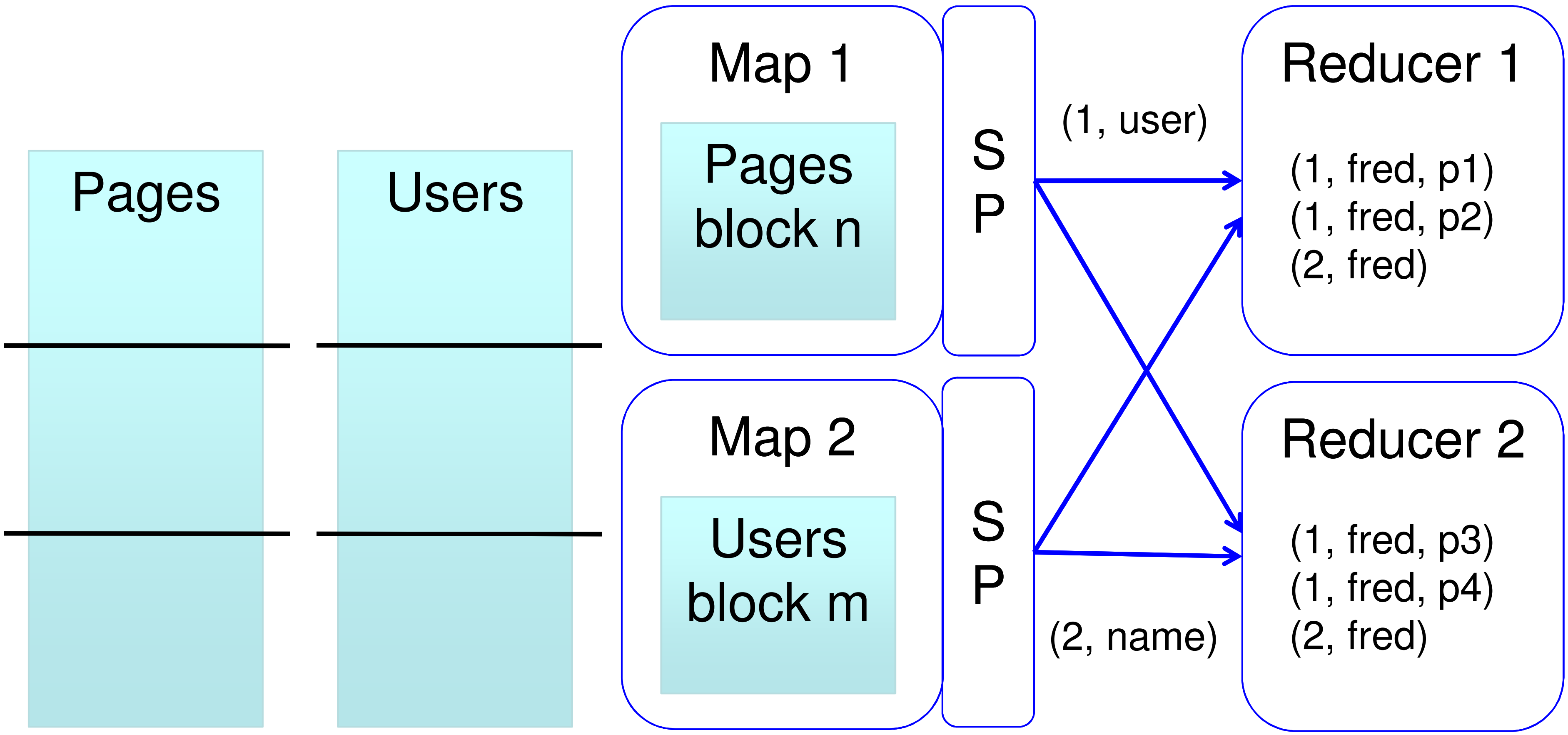
Skew Join

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



Skew Join

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

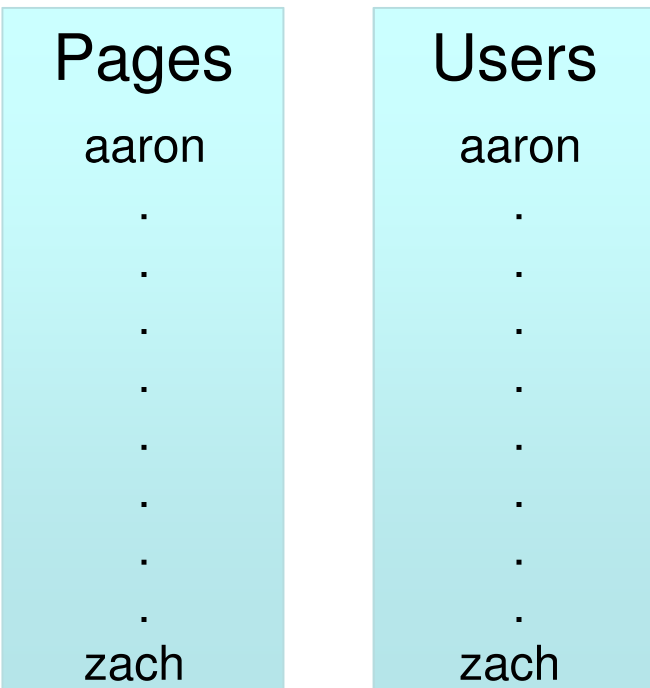


Merge Join

Pages	Users
aaron	aaron
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
zach	zach

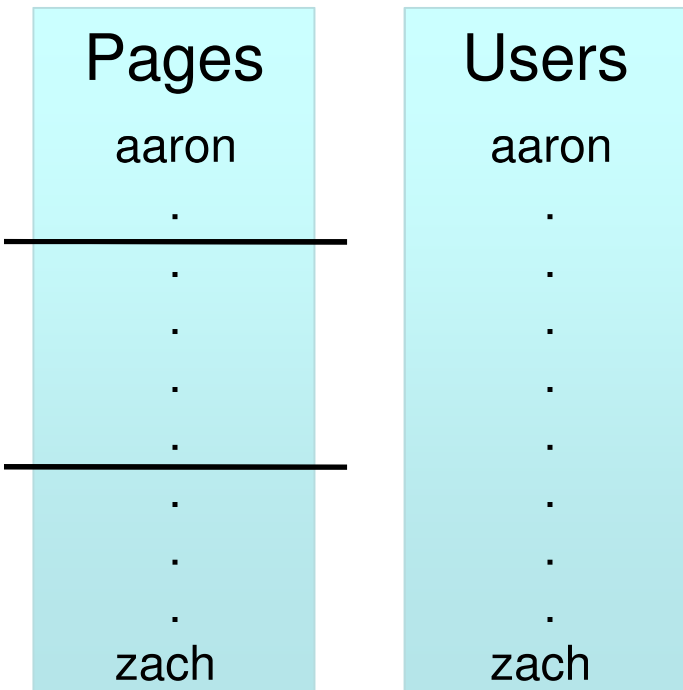
Merge Join

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



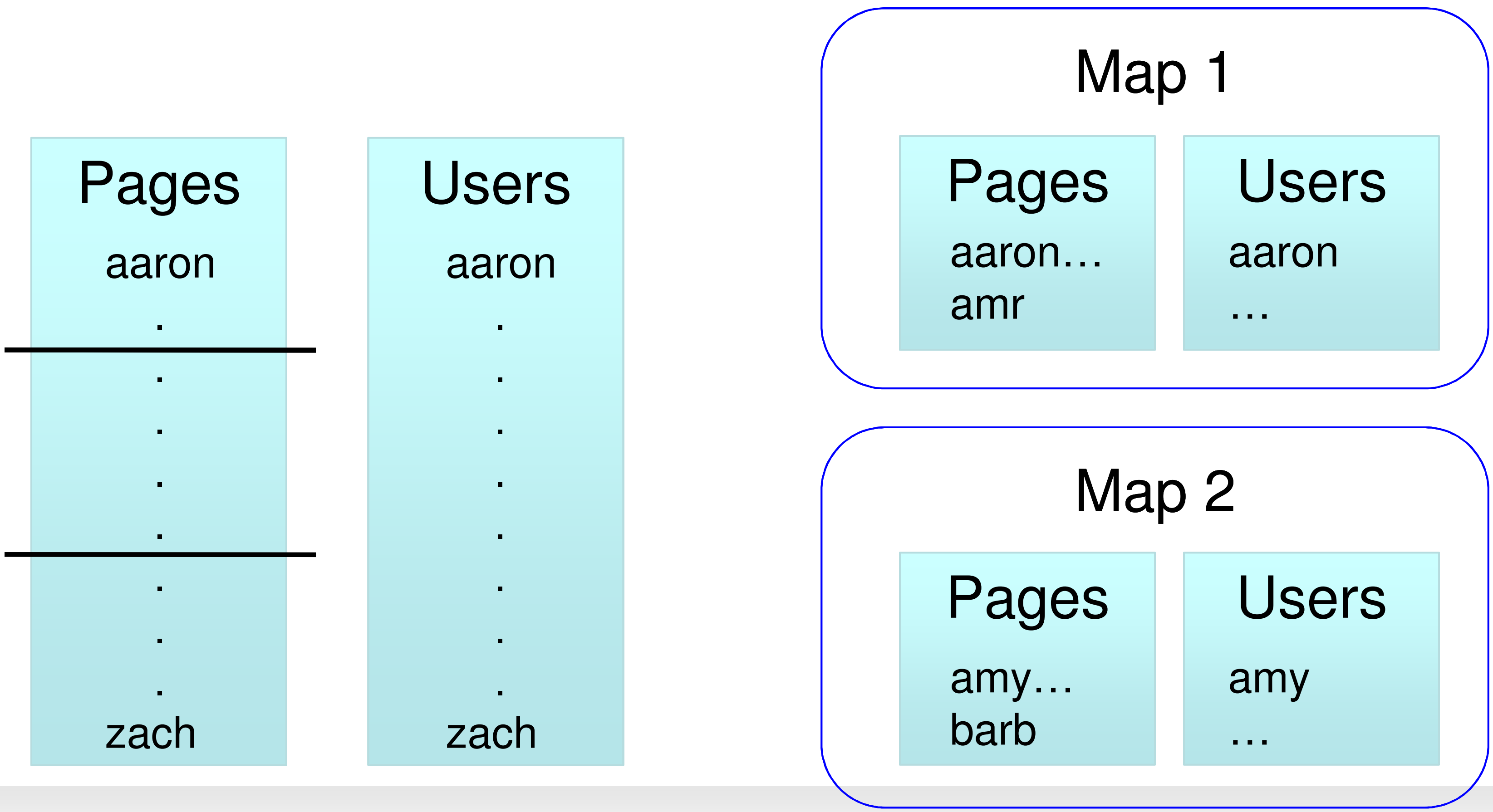
Merge Join

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



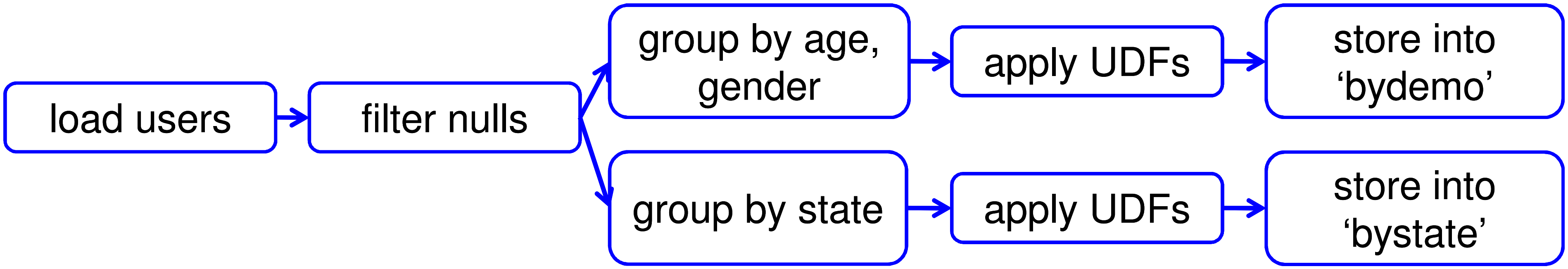
Merge Join

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

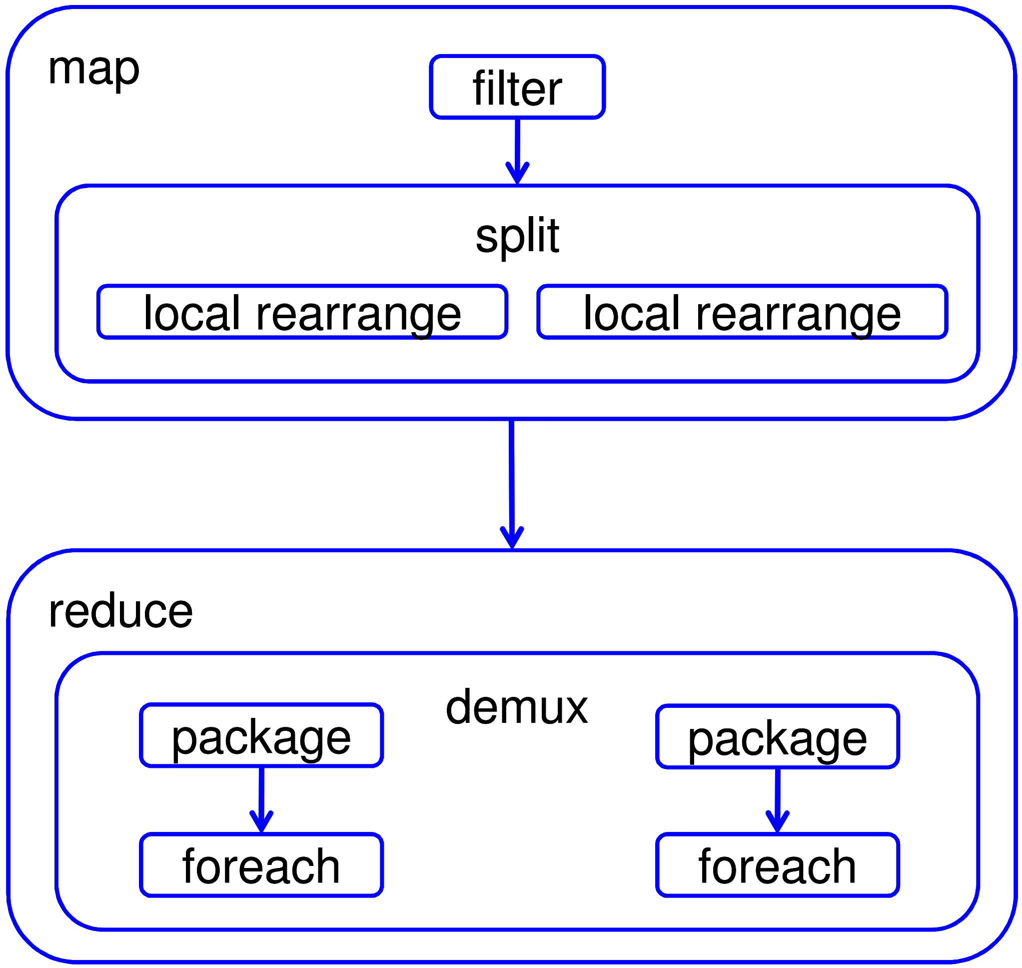


Multi-store script

```
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

Pig Latin Mini-Tutorial

(will skip in class; please read in order to do homework 7)

Outline

Based entirely on *Pig Latin: A not-so-foreign language for data processing*, by Olston, Reed, Srivastava, Kumar, and Tomkins, 2008

Quiz section tomorrow: in CSE 403 (this is CSE, don't go to EE1)

Pig-Latin Overview

- Data model = loosely typed *nested relations*
- Query model = a sql-like, dataflow language
- Execution model:
  - Option 1: run locally on your machine
  - Option 2: compile into sequence of map/reduce, run on a cluster supporting Hadoop

Example

- Input: a table of urls:
  (url, category, pagerank)
- Compute the average pagerank of all sufficiently high pageranks, for each category
- Return the answers only for categories with sufficiently many such pages

First in SQL…

```
SELECT category, AVG(pagerank)
FROM urls
WHERE pagerank > 0.2
GROUP By category
HAVING COUNT(*) > 10^6
```

…then in Pig-Latin

```
good_urls = FILTER urls BY pagerank > 0.2
groups = GROUP good_urls BY category
big_groups = FILTER groups
                        BY COUNT(good_urls) > 10^6
output = FOREACH big_groups GENERATE
                        category, AVG(good_urls.pagerank)
```

Types in Pig-Latin

- Atomic: string or number, e.g. ‘Alice’ or 55
- Tuple: (‘Alice’, 55, ‘salesperson’)
- Bag: {(‘Alice’, 55, ‘salesperson’),
  (‘Betty’,44, ‘manager’), …}
- Maps: we will try not to use these

Types in Pig-Latin

Bags can be nested !

- {(‘a’, {1,4,3}), (‘c’,{ }), (‘d’, {2,2,5,3,2})}

Tuple components can be referenced by number

- $0, $1, $2, …

$$\texttt{t} = \left(\texttt{`alice'}, \left\{ \begin{array}{c} (\texttt{`lakers', 1}) \\ (\texttt{`iPod', 2}) \end{array} \right\}, \left[ \texttt{`age'} \rightarrow \texttt{20} \right] \right)$$

Let fields of tuple `t` be called `f1, f2, f3`

Expression Type	Example	Value for `t`
Constant	`'bob'`	Independent of `t`
Field by position	`$0`	`'alice'`
Field by name	`f3`	$[\texttt{`age'} \rightarrow \texttt{20}]$
Projection	`f2.$0`	$\left\{ \begin{array}{c} (\texttt{`lakers'}) \\ (\texttt{`iPod'}) \end{array} \right\}$
Map Lookup	`f3#'age'`	`20`
Function Evaluation	`SUM(f2.$1)`	`1 + 2 = 3`
Conditional Expression	`f3#'age'>18? 'adult':'minor'`	`'adult'`
Flattening	`FLATTEN(f2)`	`'lakers', 1` `'iPod', 2`

Loading data

- Input data = FILES !
  - Heard that before ?

- The LOAD command parses an input file into a bag of records
- Both parser (="deserializer") and output type are provided by user

Loading data

```
queries = LOAD 'query_log.txt'
          USING myLoad( )
          AS (userID, queryString, timeStamp)
```

Loading data

- USING userfuction( ) -- is optional
  - Default deserializer expects tab-delimited file
- AS type – is optional
  - Default is a record with unnamed fields; refer to them as $0, $1, …
- The return value of LOAD is just a handle to a bag
  - The actual reading is done in pull mode, or parallelized

FOREACH

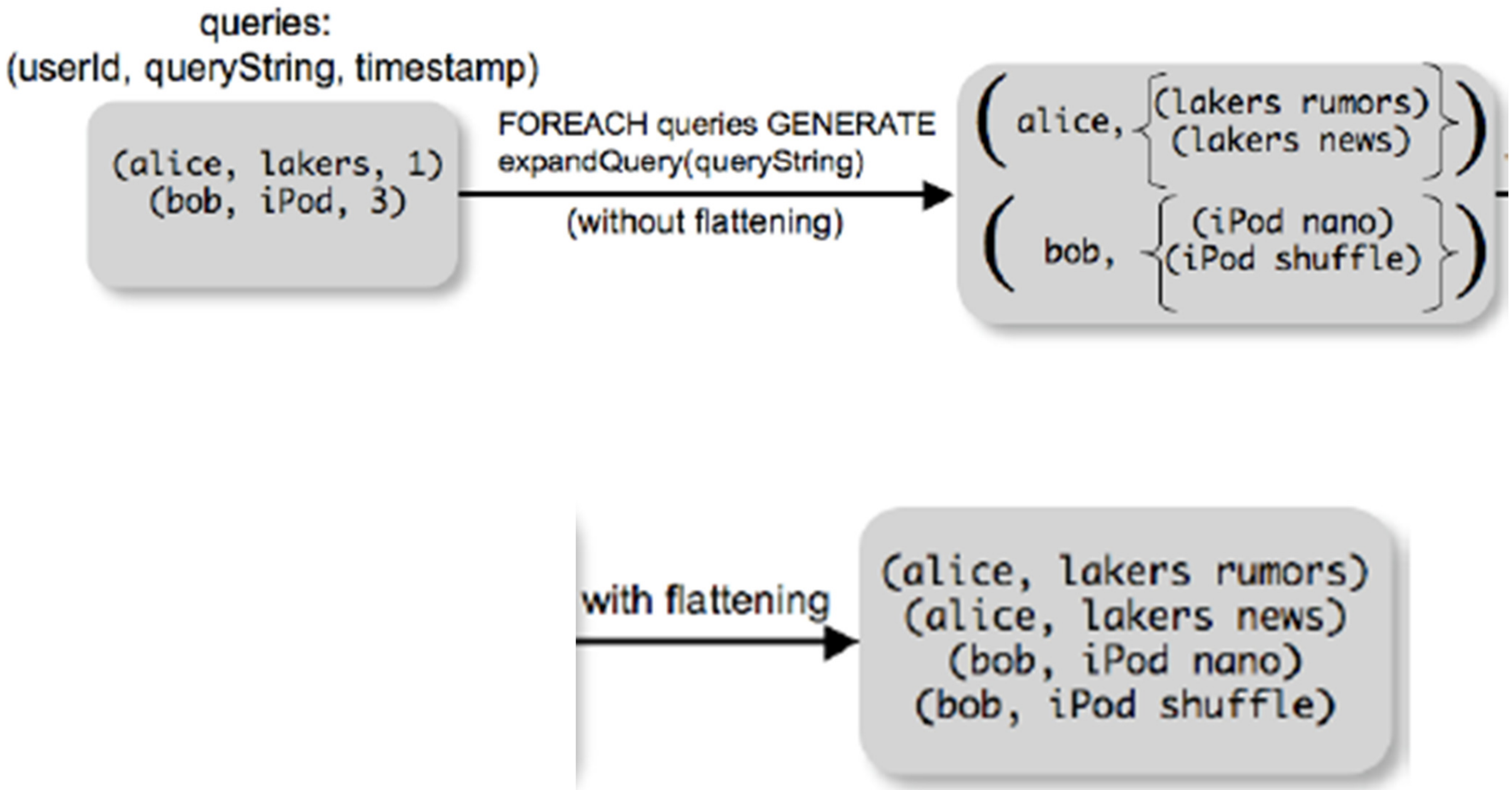
```
expanded_queries =
  FOREACH queries
  GENERATE userId, expandQuery(queryString)
```

expandQuery( ) is a UDF that produces likely expansions
Note: it returns a bag, hence expanded_queries is a nested bag

FOREACH

```
expanded_queries =
  FOREACH queries
  GENERATE userId,
             flatten(expandQuery(queryString))
```

Now we get a flat collection



queries:
(userId, queryString, timestamp)
(alice, lakers, 1)
(bob, iPod, 3)
FOREACH queries GENERATE
expandQuery(queryString)
(without flattening)
alice,
(lakers rumors)
(lakers news)
bob,
(iPod nano)
(iPod shuffle)
with flattening
(alice, lakers rumors)
(alice, lakers news)
(bob, iPod nano)
(bob, iPod shuffle)

FLATTEN

Note that it is NOT a first class function !

(that's one thing I don't like about Pig-latin)

- First class FLATTEN:
  - FLATTEN({{2,3},{5},{},{4,5,6}}) = {2,3,5,4,5,6}
  - Type: {{T}} → {T}
- Pig-latin FLATTEN
  - FLATTEN({4,5,6}) = 4, 5, 6
  - Type: {T} → T, T, T, …, T ?????

FILTER

Remove all queries from Web bots:

```
real_queries =  FILTER queries BY userId neq 'bot'
```

Better: use a complex UDF to detect Web bots:

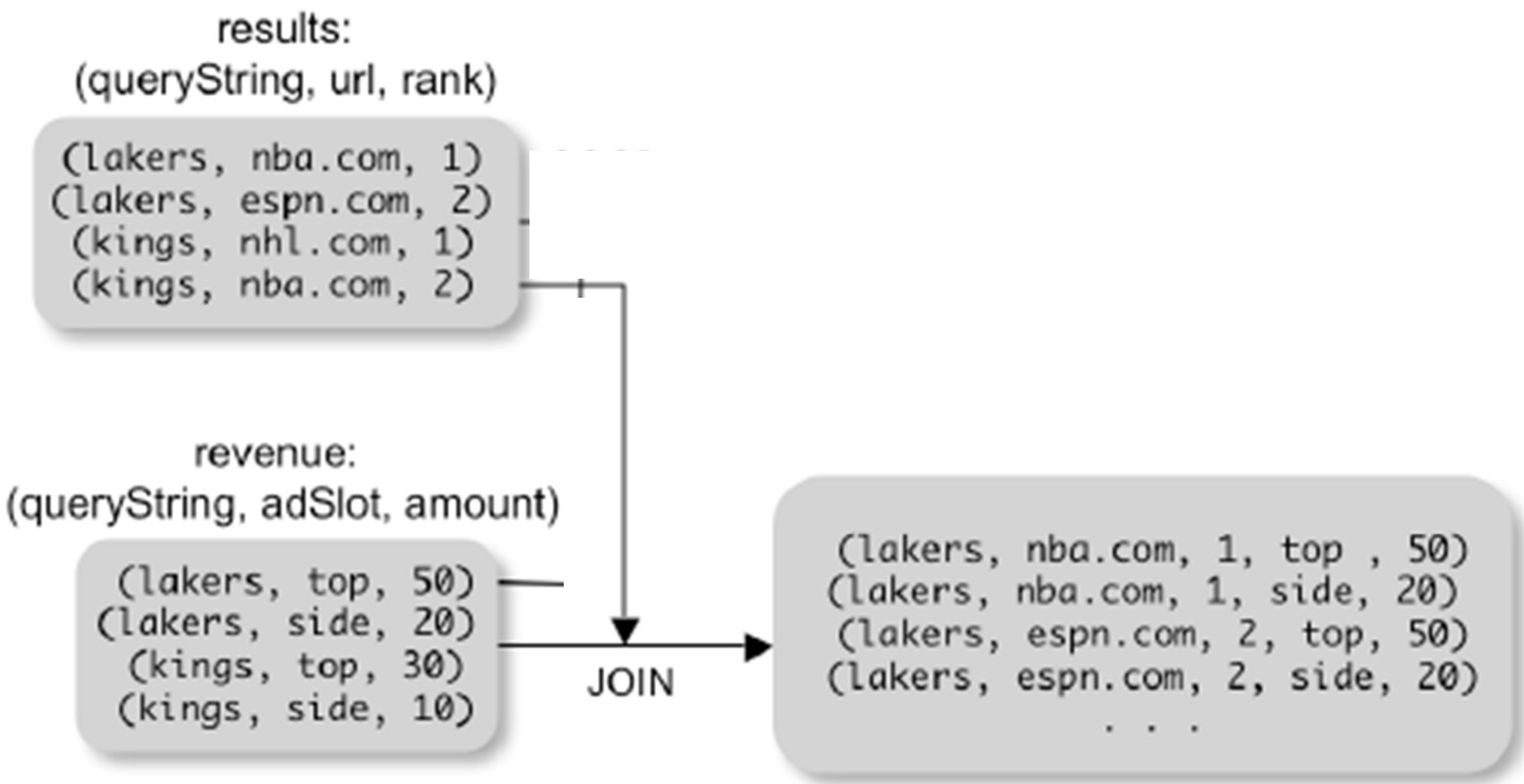
```
real_queries =  FILTER queries
                    BY NOT isBot(userId)
```

JOIN

results: {(queryString, url, position)}
revenue: {(queryString, adSlot, amount)}

```
join_result = JOIN results BY queryString
                   revenue BY queryString
```

join_result : {(queryString, url, position, adSlot, amount)}


results:
(queryString, url, rank)
(lakers, nba.com, 1)
(lakers, espn.com, 2)
(kings, nhl.com, 1)
(kings, nba.com, 2)
revenue:
(queryString, adSlot, amount)
(lakers, top, 50)
(lakers, side, 20)
(kings, top, 30)
(kings, side, 10)
JOIN
(lakers, nba.com, 1, top , 50)
(lakers, nba.com, 1, side, 20)
(lakers, espn.com, 2, top, 50)
(lakers, espn.com, 2, side, 20)
. . .

GROUP BY

revenue: {(queryString, adSlot, amount)}

```
grouped_revenue = GROUP revenue BY queryString
query_revenues =
    FOREACH grouped_revenue
    GENERATE queryString,
            SUM(revenue.amount) AS totalRevenue
```

grouped_revenue: {(queryString, {(adSlot, amount)})}

query_revenues: {(queryString, totalRevenue)}

Simple Map-Reduce

input : {(field1, field2, field3, ...)}

```
map_result = FOREACH input
                 GENERATE FLATTEN(map(*))
key_groups = GROUP map_result BY $0
output = FOREACH key_groups
             GENERATE reduce($1)
```

map_result : {(a1, a2, a3, . . .)}
key_groups : {(a1, {(a2, a3, . . .)})}

Co-Group

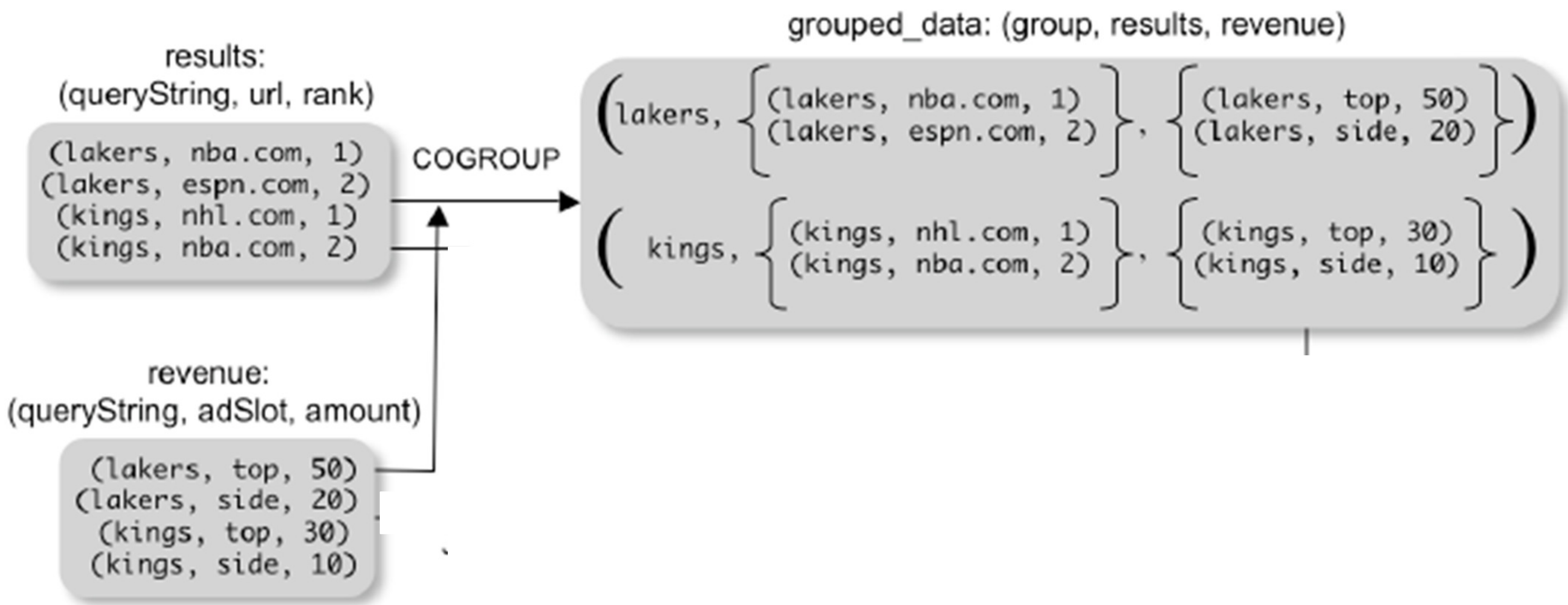
results: {(queryString, url, position)}
revenue: {(queryString, adSlot, amount)}

```
grouped_data =
    COGROUP results BY queryString,
            revenue BY queryString;
```

grouped_data: {(queryString, results:{(url, position)},
revenue:{(adSlot, amount)})}

What is the output type in general ?

Co-Group



Is this an inner join, or an outer join ?

Co-Group

grouped_data: {(queryString, results:{(url, position)},
revenue:{(adSlot, amount)})}

```
url_revenues = FOREACH grouped_data
      GENERATE
            FLATTEN(distributeRevenue(results, revenue));
```

distributeRevenue is a UDF that accepts search results and revenue information for a query string at a time, and outputs a bag of urls and the revenue attributed to them.

Co-Group v.s. Join

grouped_data: {(queryString, results:{(url, position)},
revenue:{(adSlot, amount)})}

```
grouped_data = COGROUP results BY queryString,
                       revenue BY queryString;
join_result = FOREACH grouped_data
              GENERATE FLATTEN(results),
                       FLATTEN(revenue);
```

Result is the same as JOIN

Asking for Output: STORE

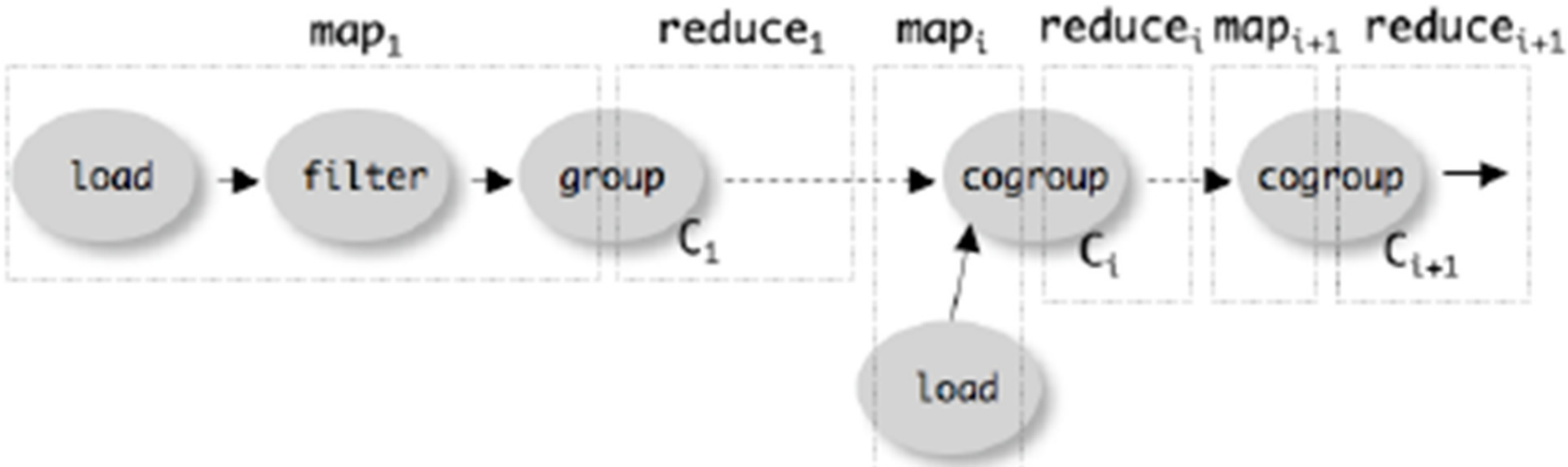
```
STORE query_revenues INTO `myoutput'
USING myStore();
```

Meaning: write query_revenues to the file ‘myoutput’

Implementation

- Over Hadoop !
- Parse query:
  - Everything between LOAD and STORE → one logical plan
- Logical plan → sequence of Map/Reduce ops
- All statements between two (CO)GROUPs → one Map/Reduce op

Implementation



Bloom Filters

We *WILL* discuss in class !

Lecture on Bloom Filters

Not described in the textbook !

Lecture based in part on:

- Broder, Andrei; Mitzenmacher, Michael (2005), "Network Applications of Bloom Filters: A Survey", Internet Mathematics 1 (4): 485–509
- Bloom, Burton H. (1970), "Space/time trade-offs in hash coding with allowable errors", Communications of the ACM 13 (7): 422–42

Pig Latin Example Continued

Users(name, age)
Pages(user, url)

```
SELECT Pages.url, count(*) as cnt
FROM Users, Pages
WHERE Users.age in [18..25]
      and Users.name = Pages.user
GROUP BY Pages.url
ORDER DESC cnt
```

Example

Problem: many Pages, but only a few visited by users with age 18..25

- Pig's solution:
  - MAP phase sends *all* pages to the reducers
- How can we reduce communication cost ?

Hash Maps

- Let $S = \{x_1, x_2, \ldots, x_n\}$ be a set of elements
- Let $m > n$
- Hash function $h : S \rightarrow \{1, 2, \ldots, m\}$

$$S = \{x_1, x_2, \ldots, x_n\}$$

H=

1	2										m
0	0	1	0	1	1	0	0	1	1	0	0

0	0	1	0	1	1	0	0	0	1	0	1

Hash Map = Dictionary

The hash map acts like a dictionary

- Insert(x, H) = set bit h(x) to 1
  - Collisions are possible
- Member(y, H) = check if bit h(y) is 1
  - False positives are possible
- Delete(y, H) = not supported !
  - Extensions possible, see later

0	0	1	0	1	1	0	0	0	1	0	1

Example (cont'd)

- Map-Reduce task 1
  - Map task: compute a hash map H of **User** names**,** where age in [18..25]. Several Map tasks in parallel.
  - Reduce task: combine all hash maps using OR. One single reducer suffices.
- Map-Reduce task 2
  - Map tasks 1: map each **User** to the appropriate region
  - Map tasks 2: map only **Pages** where user in H to appropriate region
  - Reduce task: do the join

Why don't we lose any Pages?

Analysis

- Let $S = \{x_1, x_2, \ldots, x_n\}$
- Let j = a specific bit in H ($1 \leq j \leq m$)
- What is the probability that j remains 0 after inserting all n elements from S into H ?
- Will compute in two steps

0	0	0	0	1	0	0	0	0	0	0	0

Analysis

- Recall |H| = m
- Let's insert only x_i into H

- What is the probability that bit j is 0 ?

0	0	0	0	1	0	0	0	0	0	0	0

Analysis

- Recall |H| = m
- Let's insert only x_i into H

- What is the probability that bit j is 0 ?

- Answer: p = 1 – 1/m

0	0	1	0	1	1	0	0	0	1	0	1

Analysis

- Recall $|H| = m$, $S = \{x_1, x_2, \ldots, x_n\}$
- Let's insert all elements from S in H

- What is the probability that bit j remains 0 ?

0	0	1	0	1	1	0	0	0	1	0	1

Analysis

- Recall $|H| = m$, $S = \{x_1, x_2, \ldots, x_n\}$
- Let's insert all elements from S in H

- What is the probability that bit j remains 0 ?

- Answer: $p = (1 - 1/m)^n$

0	0	1	0	1	1	0	0	0	1	0	1

Probability of False Positives

- Take a random element y, and check member(y,H)
- What is the probability that it returns *true* ?

0	0	1	0	1	1	0	0	0	1	0	1

Probability of False Positives

- Take a random element y, and check member(y,H)
- What is the probability that it returns *true* ?

- Answer: it is the probability that bit h(y) is 1, which is $f = 1 - (1 - 1/m)^n \approx 1 - e^{-n/m}$

0	0	1	0	1	1	0	0	0	1	0	1

Analysis: Example

- Example: $m = 8n$, then

  $f \approx 1 - e^{-n/m} = 1\text{-}e^{-1/8} \approx 0.11$

- A 10% false positive rate is rather high…
- Bloom filters improve that (coming next)

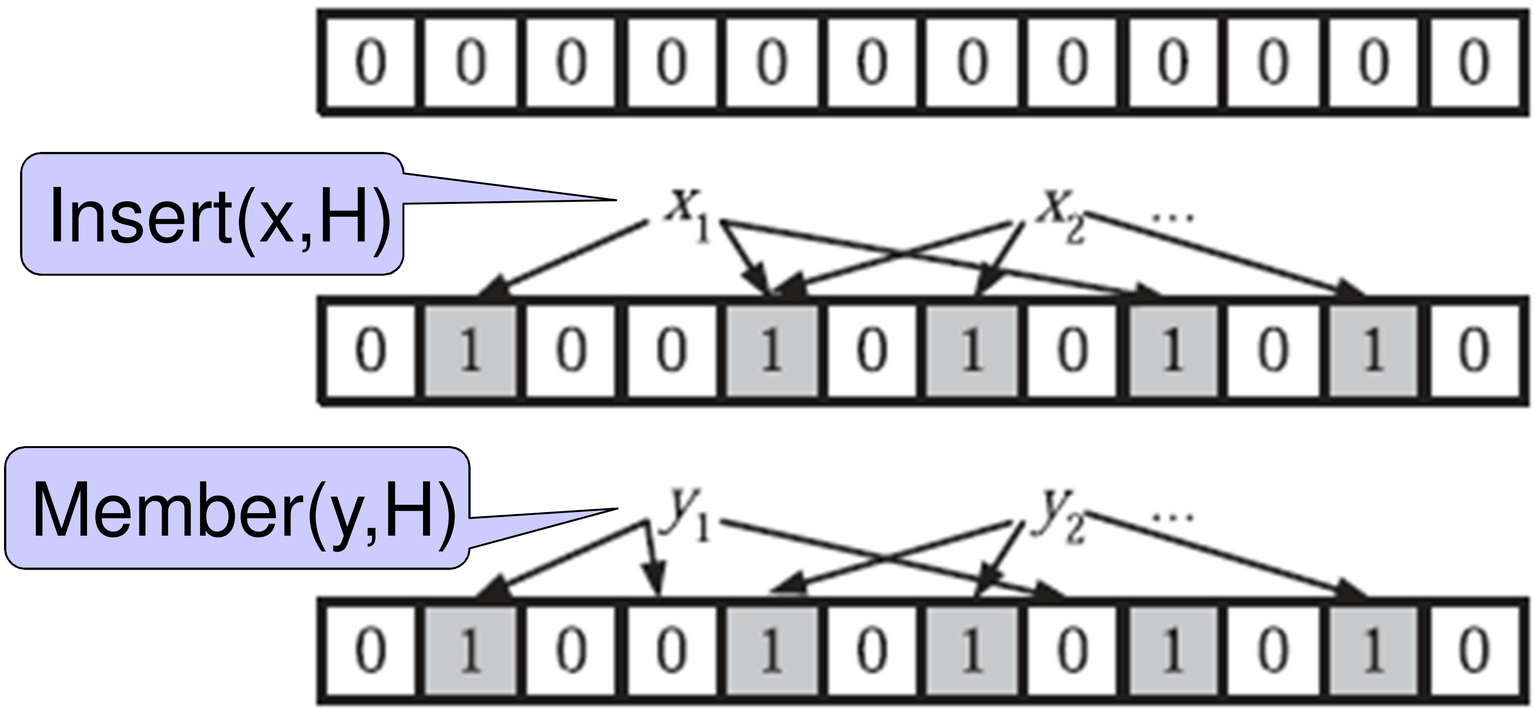
Bloom Filters

- Introduced by Burton Bloom in 1970
- Improve the false positive ratio
- Idea: use k independent hash functions

Bloom Filter = Dictionary

- Insert(x, H) = set bits $h_1(x), \ldots, h_k(x)$ to 1
  - Collisions between x and x' are possible
- Member(y, H) = check if bits $h_1(y), \ldots, h_k(y)$ are 1
  - False positives are possible
- Delete(z, H) = not supported !
  - Extensions possible, see later

Example Bloom Filter k=3



y_1 = is not in H (why ?); y_2 may be in H (why ?)

Choosing k

Two competing forces:

- If k = large
  - Test more bits for member(y,H) ➔ lower false positive rate
  - More bits in H are 1 ➔ higher false positive rate
- If k = small
  - More bits in H are 0 ➔ lower positive rate
  - Test fewer bits for member(y,H) ➔ higher rate

0	0	0	0	1	0	0	1	0	1	0	0

Analysis

- Recall $|H| = m$, #hash functions = k
- Let's insert only x_i into H

- What is the probability that bit j is 0 ?

0	0	0	0	1	0	0	1	0	1	0	0

Analysis

- Recall $|H| = m$, #hash functions = k
- Let's insert only x_i into H
- What is the probability that bit j is 0 ?
- Answer: $p = (1 - 1/m)^k$

0	0	1	0	1	1	0	1	0	1	0	0

Analysis

- Recall $|H| = m$, $S = \{x_1, x_2, \ldots, x_n\}$
- Let's insert all elements from S in H

- What is the probability that bit j remains 0 ?

0	0	1	0	1	1	0	1	0	1	0	0

Analysis

- Recall $|H| = m$, $S = \{x_1, x_2, \ldots, x_n\}$
- Let's insert all elements from S in H

- What is the probability that bit j remains 0 ?

- Answer: $p = (1 - 1/m)^{kn} \approx e^{-kn/m}$

Probability of False Positives

- Take a random element y, and check member(y,H)
- What is the probability that it returns *true* ?

Probability of False Positives

- Take a random element y, and check member(y,H)
- What is the probability that it returns *true* ?

- Answer: it is the probability that all k bits $h_1(y), \ldots, h_k(y)$ are 1, which is:

$$\boxed{f = (1-p)^k \approx (1 - e^{-kn/m})^k}$$

Optimizing k

- For fixed m, n, choose k to minimize the false positive rate f
- Denote g = ln(f) = k ln(1 – $e^{-kn/m}$)
- Goal: find k to minimize g

$$\frac{\partial g}{\partial k} = \ln\left(1 - \mathrm{e}^{-\frac{kn}{m}}\right) + \frac{kn}{m}\frac{\mathrm{e}^{-\frac{kn}{m}}}{1 - \mathrm{e}^{-\frac{kn}{m}}}$$

$$\boxed{k = \ln 2 \times m/n}$$

Bloom Filter Summary

Given $n = |S|$, $m = |H|$,
choose $k = \ln 2 \times m/n$ hash functions

Probability that some bit j is 1 $\boxed{p \approx e^{-kn/m} = ½}$

Expected distribution $\boxed{m/2 \text{ bits } 1,\ m/2 \text{ bits } 0}$

Probability of false positive

$$\boxed{f = (1-p)^k \approx (½)^k = (½)^{(\ln 2)m/n} \approx (0.6185)^{m/n}}$$

Bloom Filter Summary

- In practice one sets $m = cn$, for some constant c
  - Thus, we use c bits for each element in S
  - Then $f \approx (0.6185)^c$ = constant

- Example: $m = 8n$, then
  - $k = 8(\ln 2) = 5.545$ (use 6 hash functions)
  - $f \approx (0.6185)^{m/n} = (0.6185)^8 \approx 0.02$ (2% false positives)
  - Compare to a hash table: $f \approx 1 - e^{-n/m} = 1 - e^{-1/8} \approx 0.11$

The reward for increasing m is much higher for Bloom filters

Set Operations

Intersection and Union of Sets:

- Set S ➔ Bloom filter H
- Set S' ➔ Bloom filter H'

- How do we computed the Bloom filter for the intersection of S and S' ?

Set Operations

Intersection and Union:

- Set S ➔ Bloom filter H
- Set S' ➔ Bloom filter H'

- How do we computed the Bloom filter for the intersection of S and S' ?
- Answer: bit-wise AND: $H \wedge H'$

Counting Bloom Filter

Goal: support delete(z, H)

Keep a counter for each bit j

- Insertion ➔ increment counter
- Deletion ➔ decrement counter
- Overflow ➔ keep bit 1 forever

Using 4 bits per counter:

Probability of overflow $\leq 1.37\ 10^{-15} \times m$

Application: Dictionaries

Bloom originally introduced this for hyphenation

- 90% of English words can be hyphenated using simple rules
- 10% require table lookup
- Use “bloom filter” to check if lookup needed

Application: Distributed Caching

- Web proxies maintain a cache of (URL, page) pairs
- If a URL is not present in the cache, they would like to check the cache of other proxies in the network
- Transferring all URLs is expensive !
- Instead: compute Bloom filter, exchange periodically