CSE 544: Principles of Database Systems

MapReduce, PigLatin

Parallel DB Wrapup

Announcements

- HW2 (SimpleDB) was due last night
- HW3 (MR/PigLatin) posted today
- Review 6 (MapReduce) was due last night
- Review 7 (Datalog) due next Monday
- Project Milestone due next Friday

Overview of Today's Lecture

- Map/reduce (paper)
- Parallel query processing wrap-up
- PigLatin
  - Learn on your own, for HW3, by reading the slides

Recommended Reading

- Chapter 2 (Sections 1,2,3) of Mining of Massive Datasets, by Rajaraman and Ullman http://i.stanford.edu/~ullman/mmds.html

Cluster Computing

Cluster Computing

- Large number of commodity servers, connected by high speed, commodity network
- Rack: holds a small number of servers
- Data center: holds many racks

Cluster Computing

- Massive parallelism:
  - 100s, or 1000s, or 10000s servers
  - Many hours
- Failure becomes a fact of life:
  - If medium-time-between-failure is 1 year
  - Then 10000 servers have one failure / hour

Distributed File System (DFS)

Very large files: TBs, PBs

- 1 file = partitioned into chunks, e.g. 64MB
- 1 chunk = replicated several times (≥3)

Implementations:

- Google's DFS: GFS, proprietary
- Hadoop's DFS: HDFS, open source

Typical Problems Solved by MR

- Read a lot of data
- Map: extract something you care about from each record
- Shuffle and Sort
- Reduce: aggregate, summarize, filter, transform
- Write the results

Outline stays the same, map and reduce change to fit the problem

slide source: Jeff Dean

Data Model

Files !

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

A MapReduce 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: **`(input key, value)`**
- Ouput:
  bag of **`(intermediate key, value)`**

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)`**

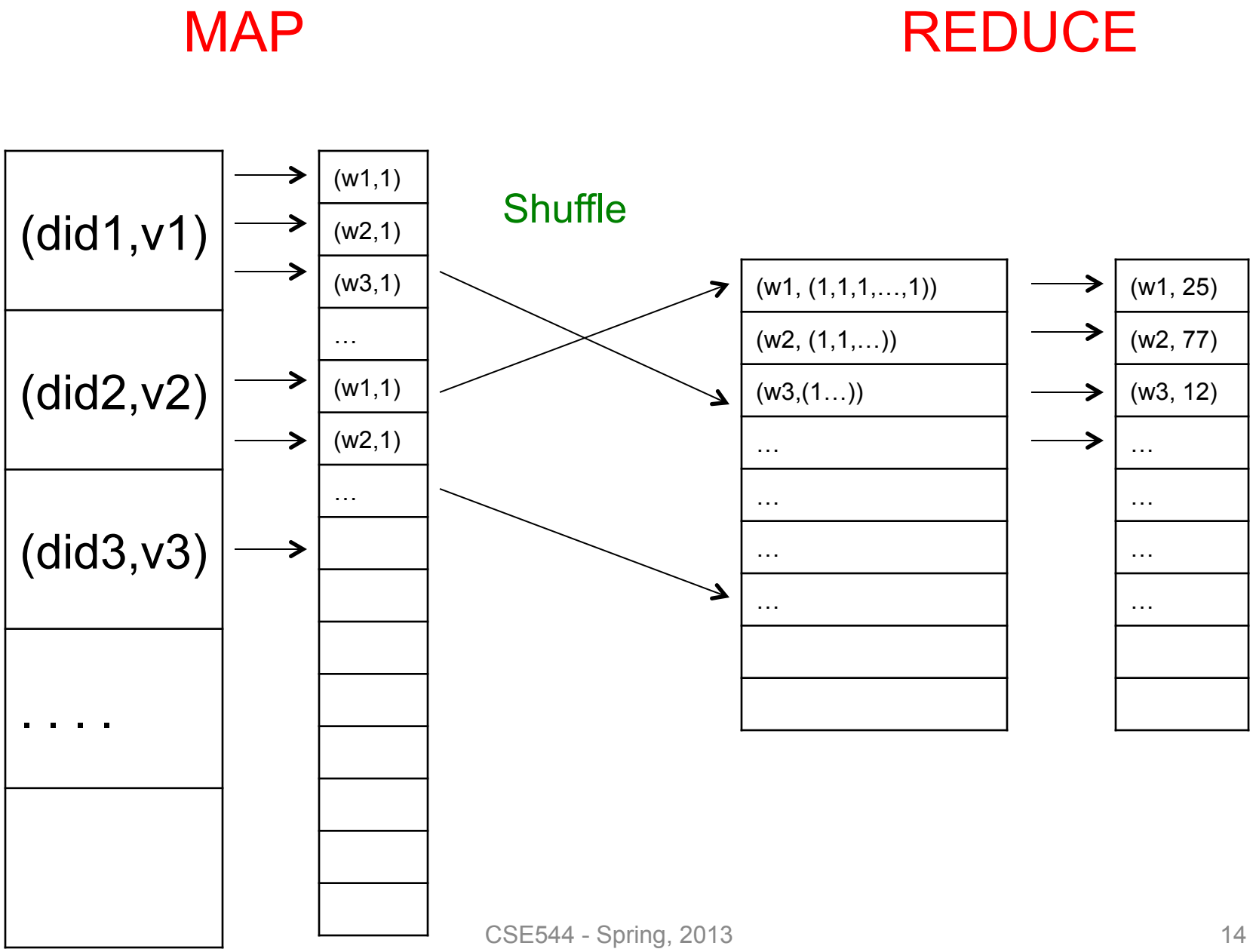
System groups all pairs with the same intermediate key, and passes the bag of values to the REDUCE function

Example

- Counting the number of occurrences of each word in a large collection of documents
- Each Document
  - The key = document id (did)
  - The value = set of words (word)

```
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
(did1,v1)
(did2,v2)
(did3,v3)
. . . .
(w1,1)
(w2,1)
(w3,1)
…
(w1,1)
(w2,1)
…
Shuffle
(w1, (1,1,1,…,1))
(w2, (1,1,…))
(w3,(1…))
…
…
…
…
(w1, 25)
(w2, 77)
(w3, 12)
…
…
…
…

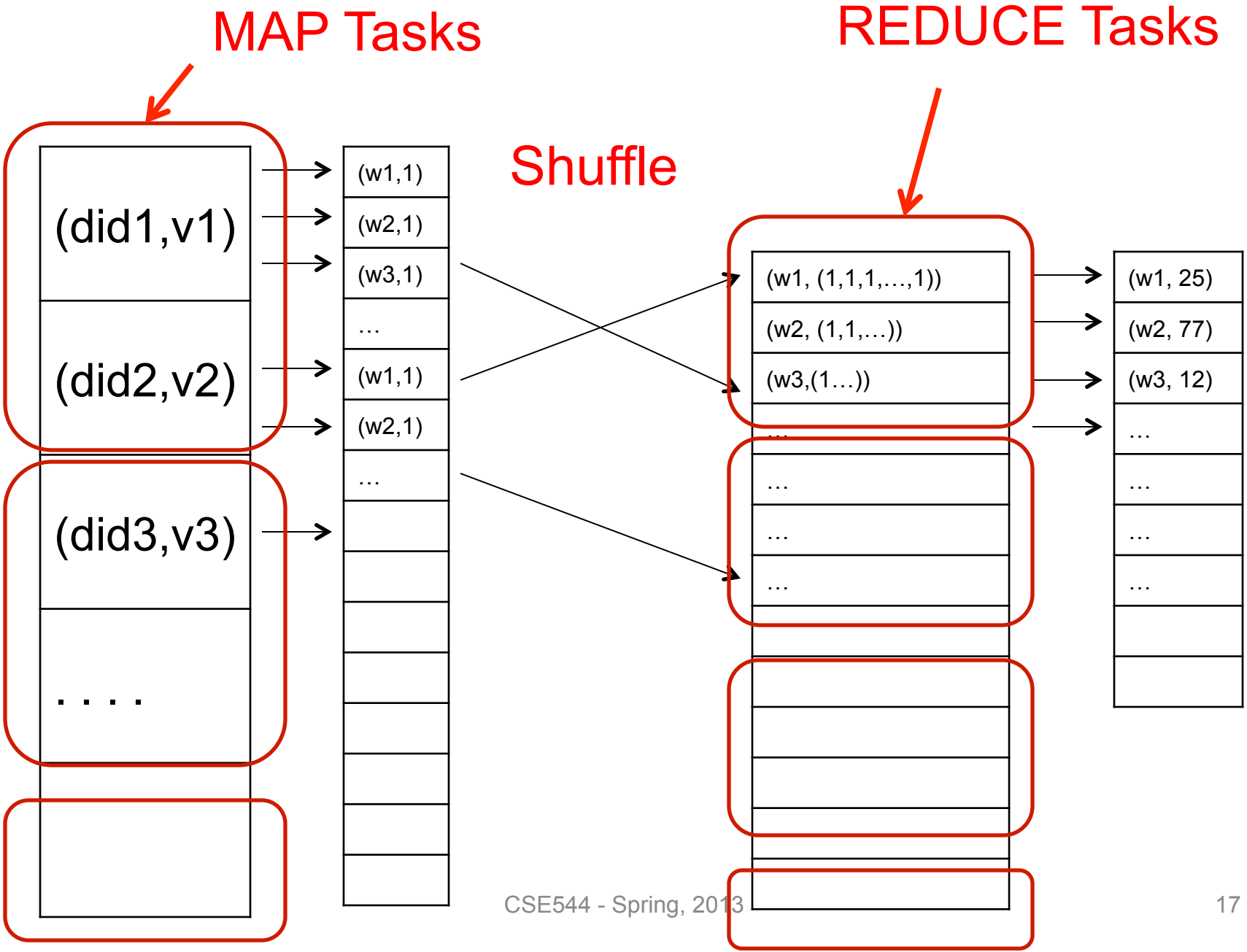
Jobs v.s. Tasks

- A MapReduce Job
  - One single "query", e.g. count the words in all docs
  - More complex queries may consists of multiple jobs

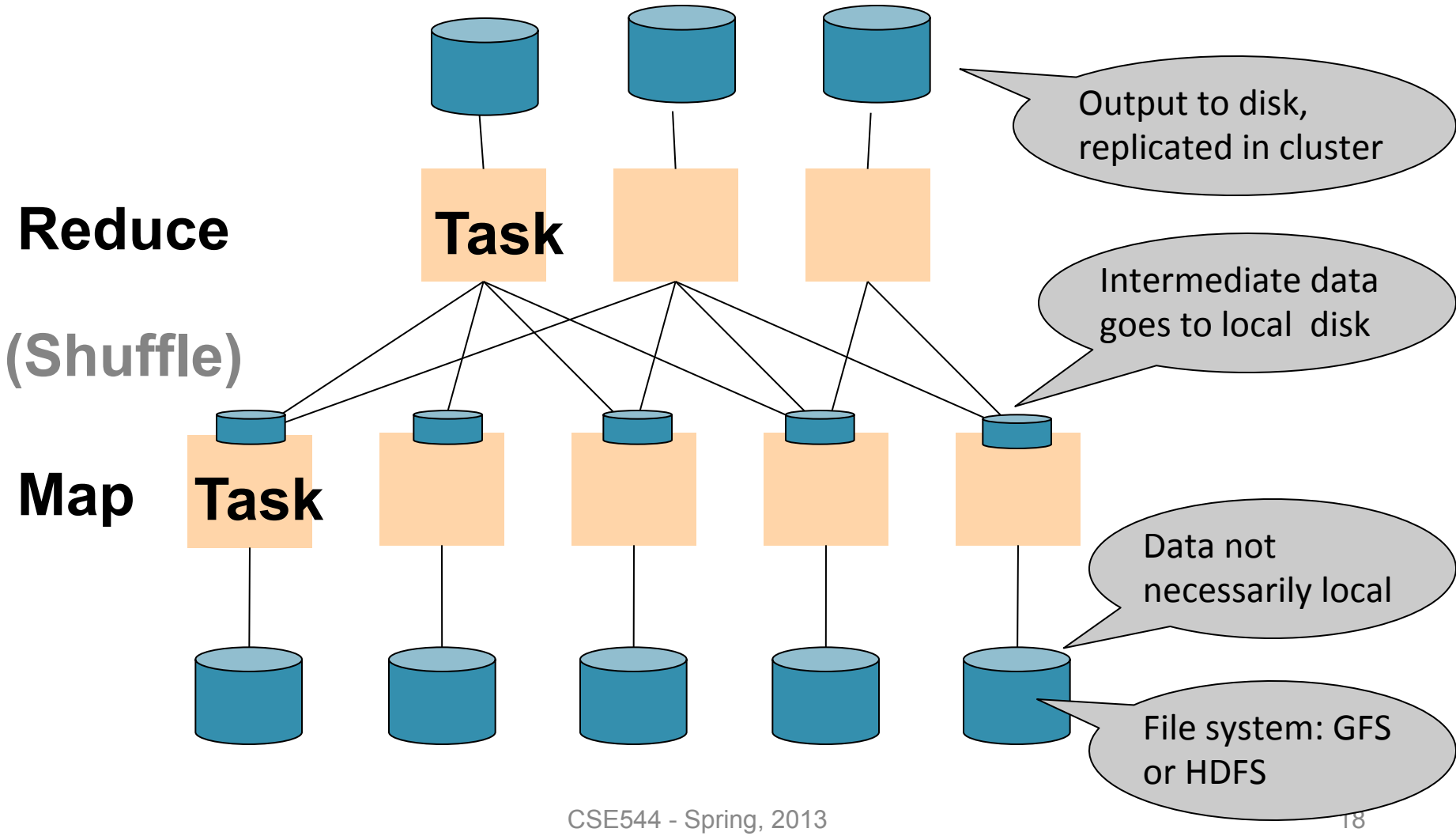
- A Map Task, or a Reduce Task
  - A group of instantiations of the map-, or reduce-function, which are scheduled on a single worker

Workers

- A worker is a process that executes one task at a time
- Typically there is one worker per processor, hence 4 or 8 per node

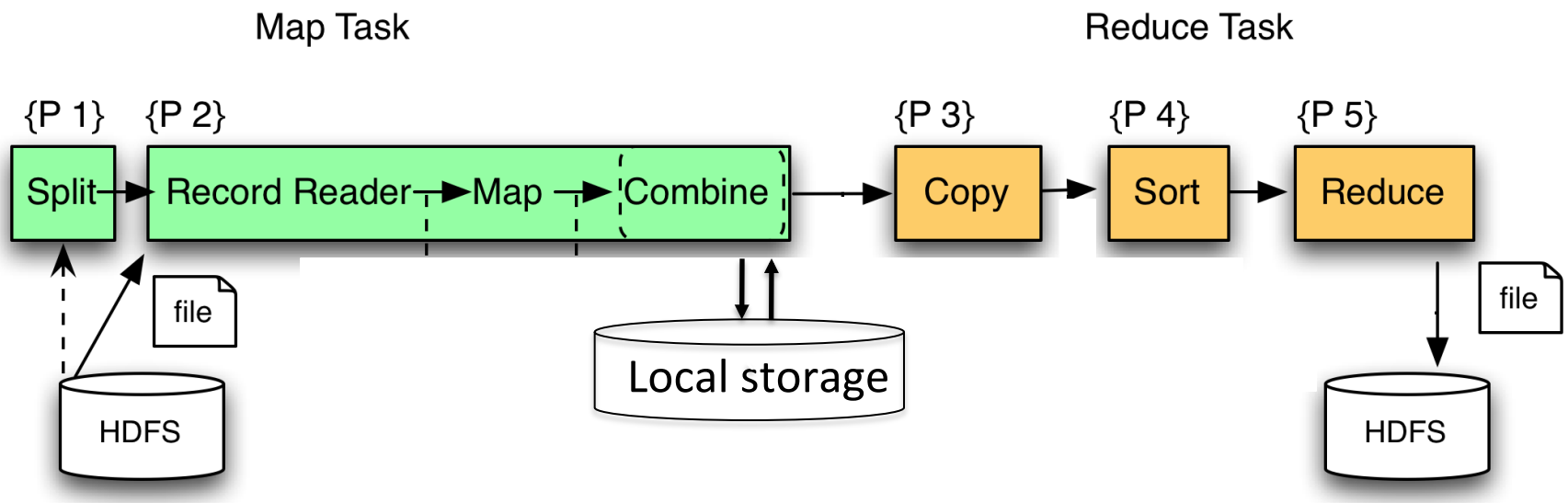

MAP Tasks
REDUCE Tasks
Shuffle
(did1,v1)
(did2,v2)
(did3,v3)
. . . .
(w1,1)
(w2,1)
(w3,1)
…
(w1,1)
(w2,1)
…
(w1, (1,1,1,…,1))
(w2, (1,1,…))
(w3,(1…))
…
…
…
…
(w1, 25)
(w2, 77)
(w3, 12)
…
…
…
…

MapReduce Execution Details

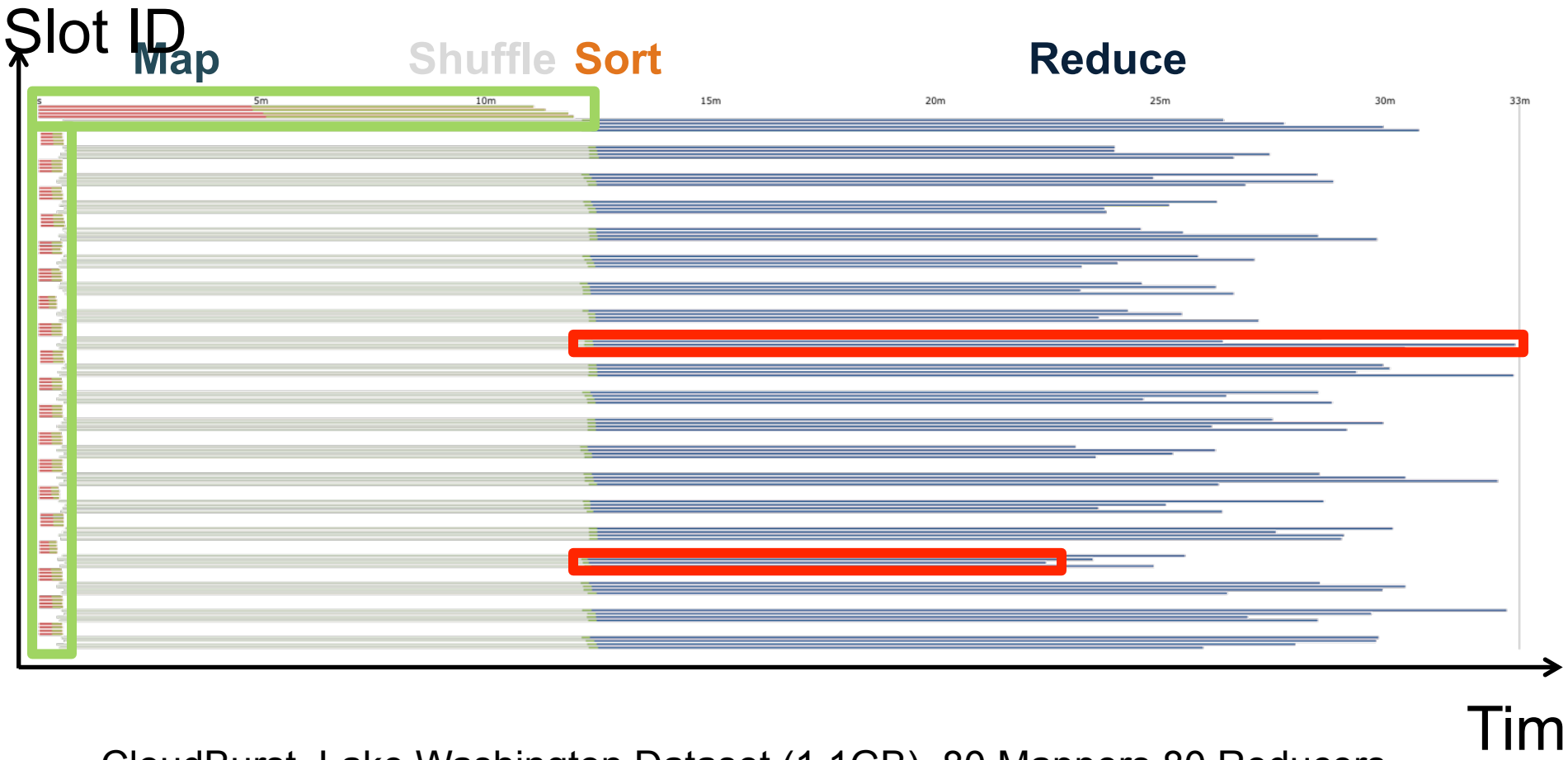


MR Phases

- Each Map and Reduce task has multiple phases:



Example: CloudBurst



CloudBurst. Lake Washington Dataset (1.1GB). 80 Mappers 80 Reducers.

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

Failures, Stragglers

Worker failure

- Master pings workers periodically,
- If down then reassigns the task to another worker

Straggler

- A machine that takes unusually long time to complete one of the last tasks.

Tuning MapReduce

Very hard!

- Choosing M,R:
  - Bigger is better but master needs O(M×R) memory
- Typical:
  - M=number of chunks (“number of blocks”)
  - R=much smaller (why??); e.g. 1.5 * #servers
- The combiner (Talk in class…)

MapReduce Summary

- Hides scheduling and parallelization details
- However, very limited queries
  - Difficult to write more complex tasks
  - Need multiple map-reduce operations
- Solution: declarative query language over MR: PigLatin, Dryad, Dremmel, Tenzing, …

Declarative Languages on MR

- PIG Latin (Yahoo!)
  - New language, like Relational Algebra
  - Open source
- HiveQL (Facebook)
  - SQL-like language
  - Open source
- SQL: Big-Query / Dremmel (Google)
  - SQL on MR
  - Proprietary

Doc(key, word)

Query Evaluation in MR

MAP=GROUP BY, REDUCE=Aggregate

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

Joins in MapReduce

- If MR is GROUP-BY plus AGGREGATE, then how do we compute R(A,B) ⋈ S(B,C) using MR?

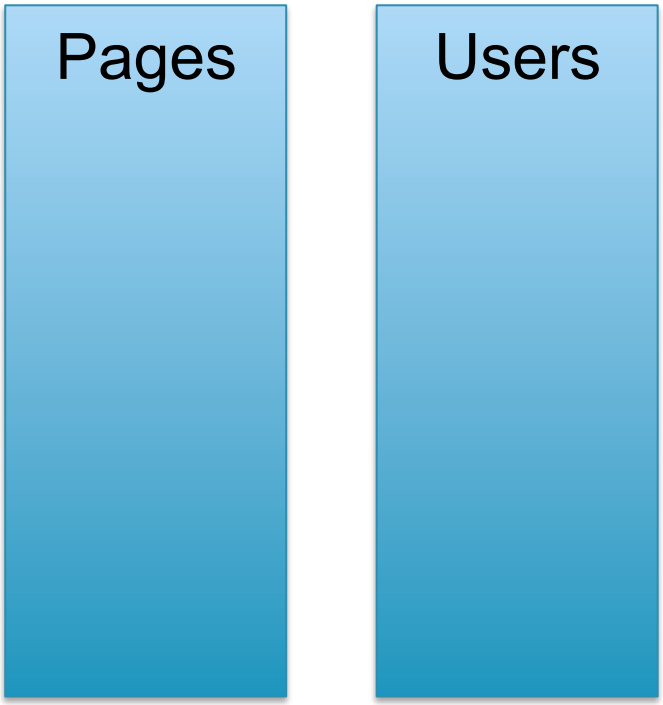
Joins in MapReduce

- If MR is GROUP-BY plus AGGREGATE, then how do we compute R(A,B) ⋈ S(B,C) using MR?

- Answer:
  - Map: group R by R.B, group S by S.B
    - Input = either a tuple R(a,b) or a tuple S(b,c)
    - Output = (b,R(a,b)) or (b,S(b,c)) respectively
  - Reduce:
    - Input = (b,{R(a1,b),R(a2,b),…,S(b,c1),S(b,c2),…})
    - Output = {R(a1,b),R(a2,b),…} × {S(b,c1),S(b,c2),…}
    - In practice: improve the reduce function (next…)

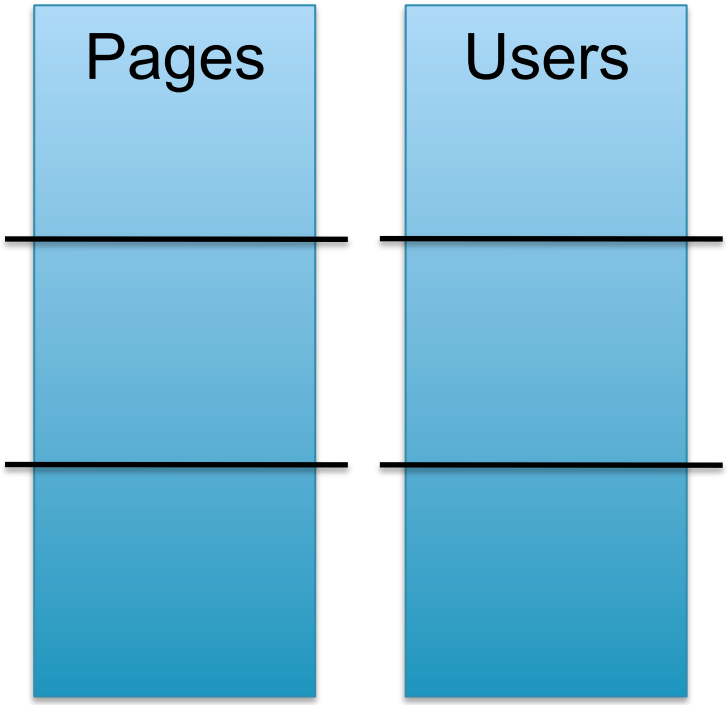
Hash Join in MR

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



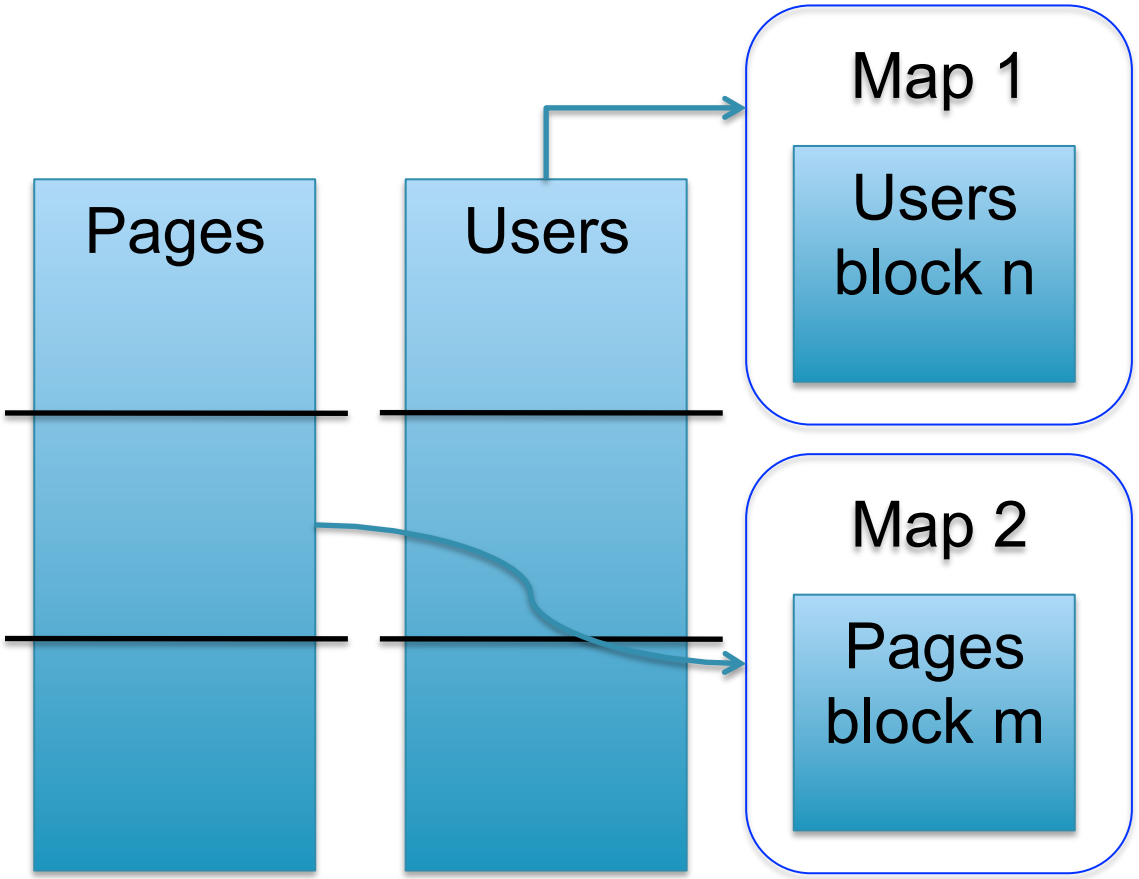
Hash Join in MR

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



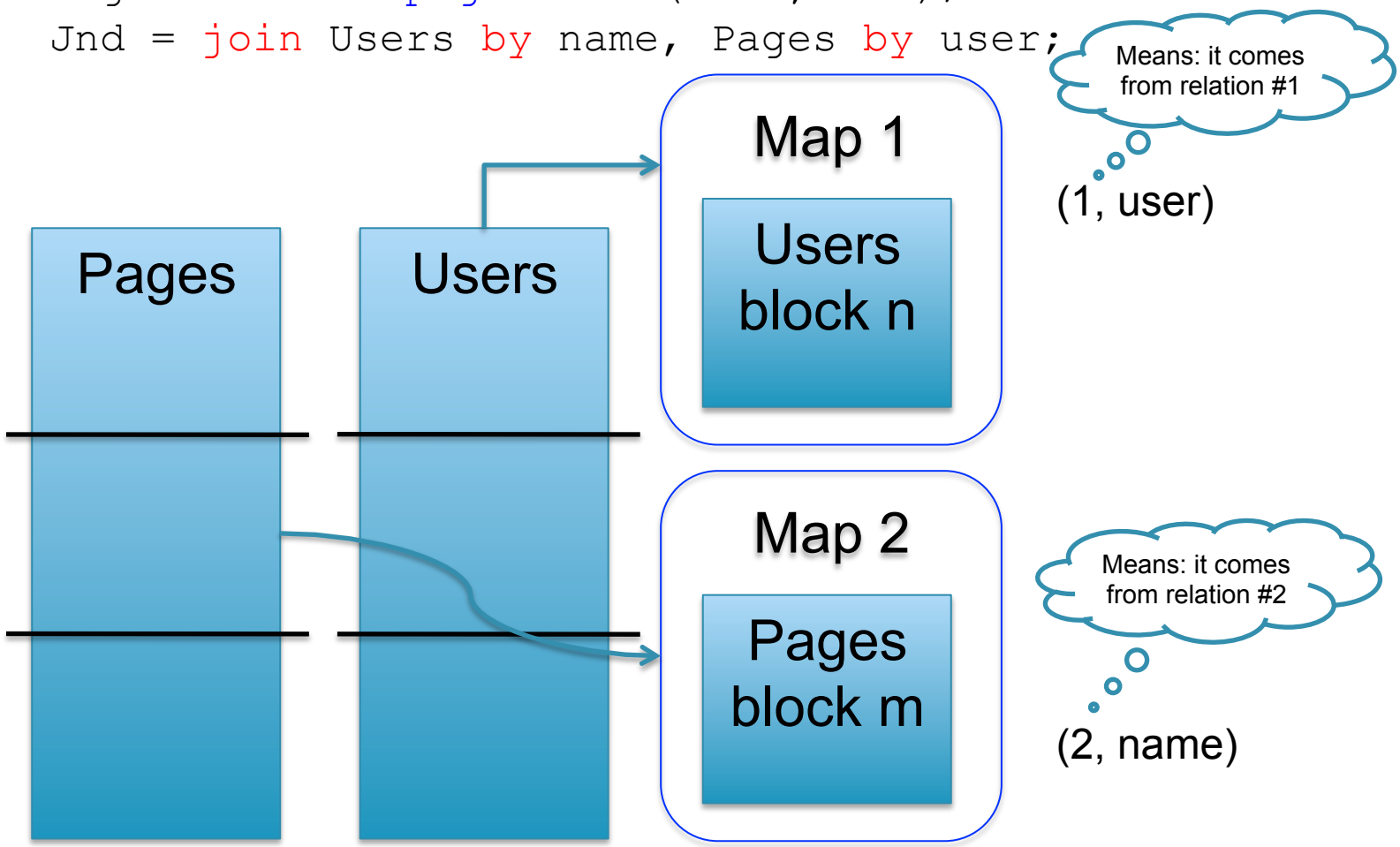
Hash Join in MR

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



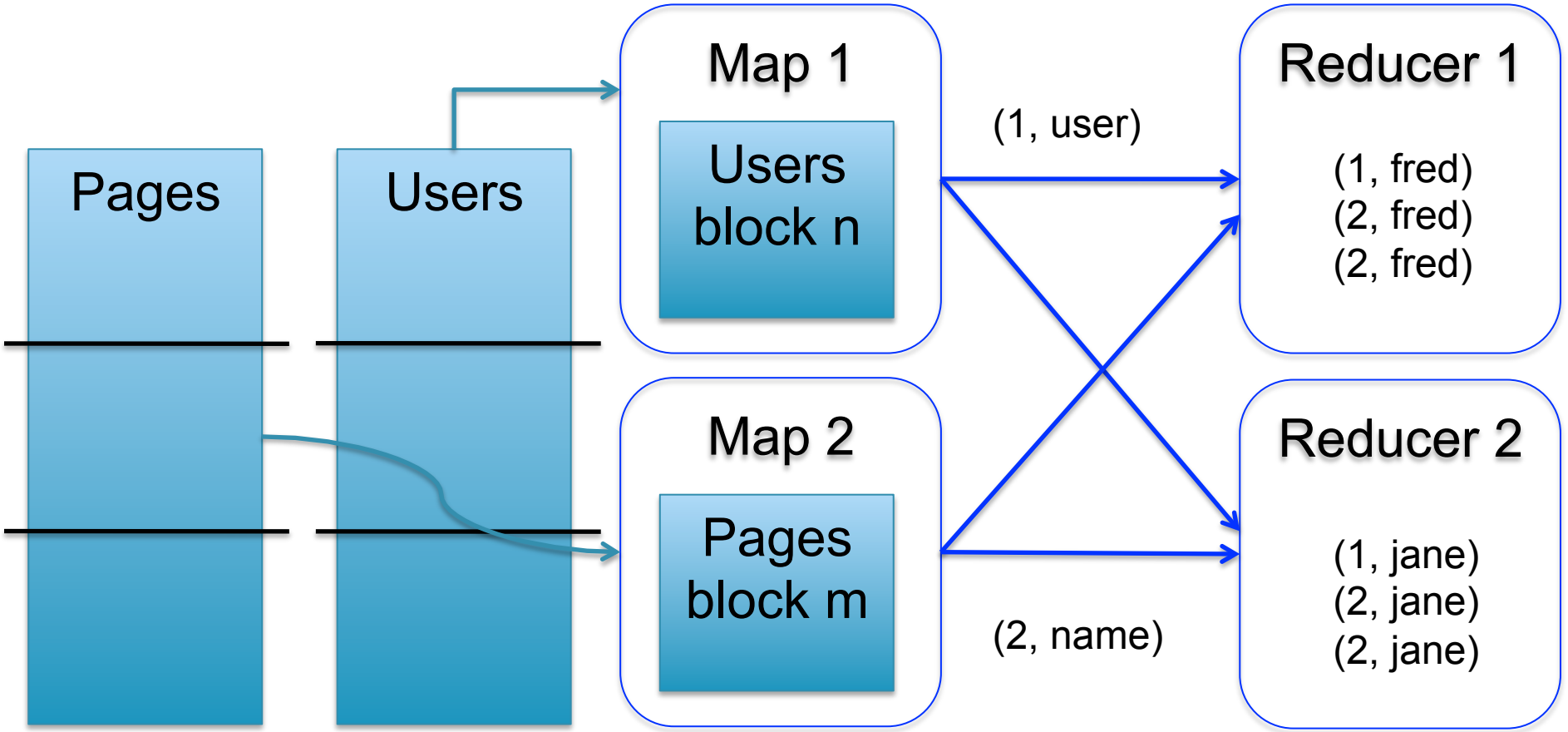
Hash Join in MR

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



Hash Join in MR

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



Hash Join in MR

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

```
map([String key], String value):
    // value.relation is either 'Users' or 'Pages'
    if value.relation='Users':
            EmitIntermediate(value.name, (1, value));
    else // value.relation='Pages':
            EmitIntermediate(value.user, (2, value));
```

Relying entirely on the MR system to do the hashing

```
reduce(String user, Iterator values):
    Users = empty;  Pages = empty;
    for each v in values:
            if v.type = 1: Users.insert(v)
      else Pages.insert(v);
    for v1 in Users, for v2 in Pages
            Emit(v1,v2);
```

Hash Join in MR

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

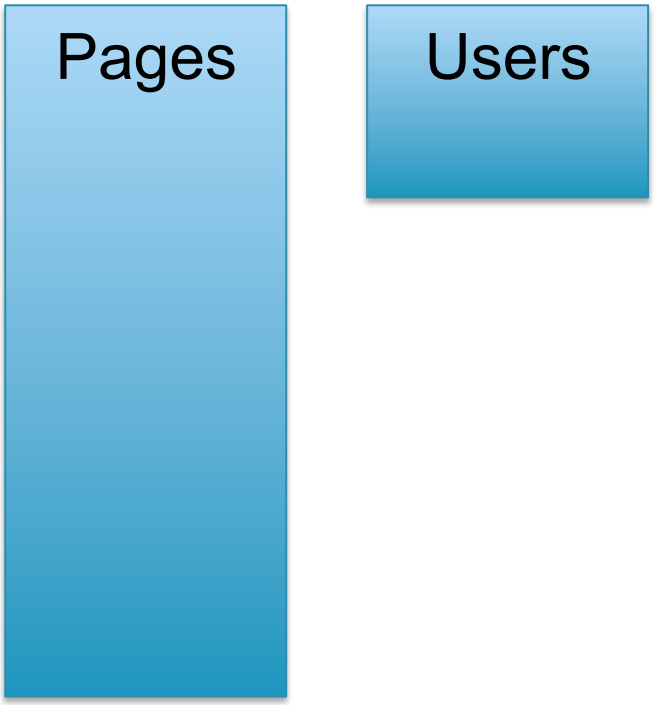
```
map([String key], String value):
    // value.relation is either 'Users' or 'Pages'
    if value.relation='Users':
        EmitIntermediate(h(value.name), (1, value));
    else // value.relation='Pages':
        EmitIntermediate(h(value.user), (2, value));
```

Constrolling the hash function

```
reduce(String user, Iterator values):
    Users = empty;  Pages = empty;
    for each v in values:
        if v.type = 1: Users.insert(v)
      else Pages.insert(v);
    for v1 in Users, for v2 in Pages
        if v1.name=v2.user: Emit(v1,v2);
```

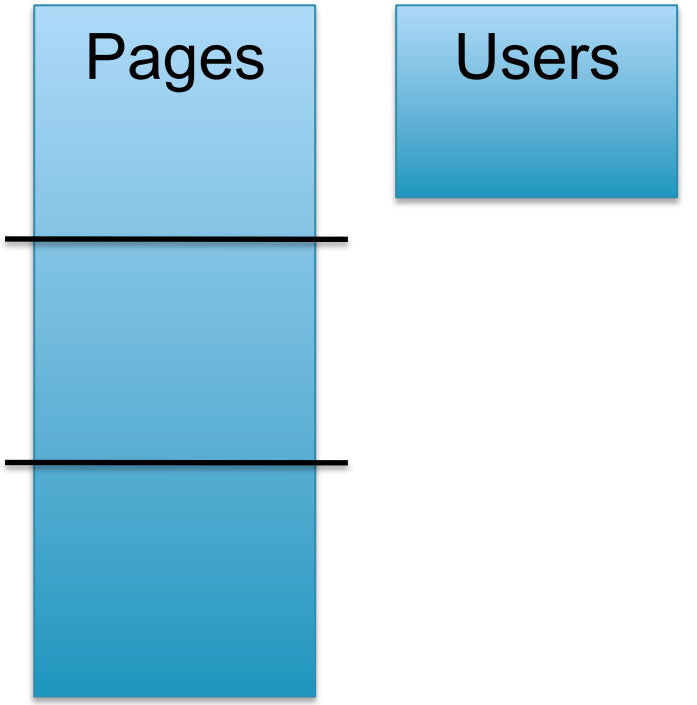
Broadcast Join in MR

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



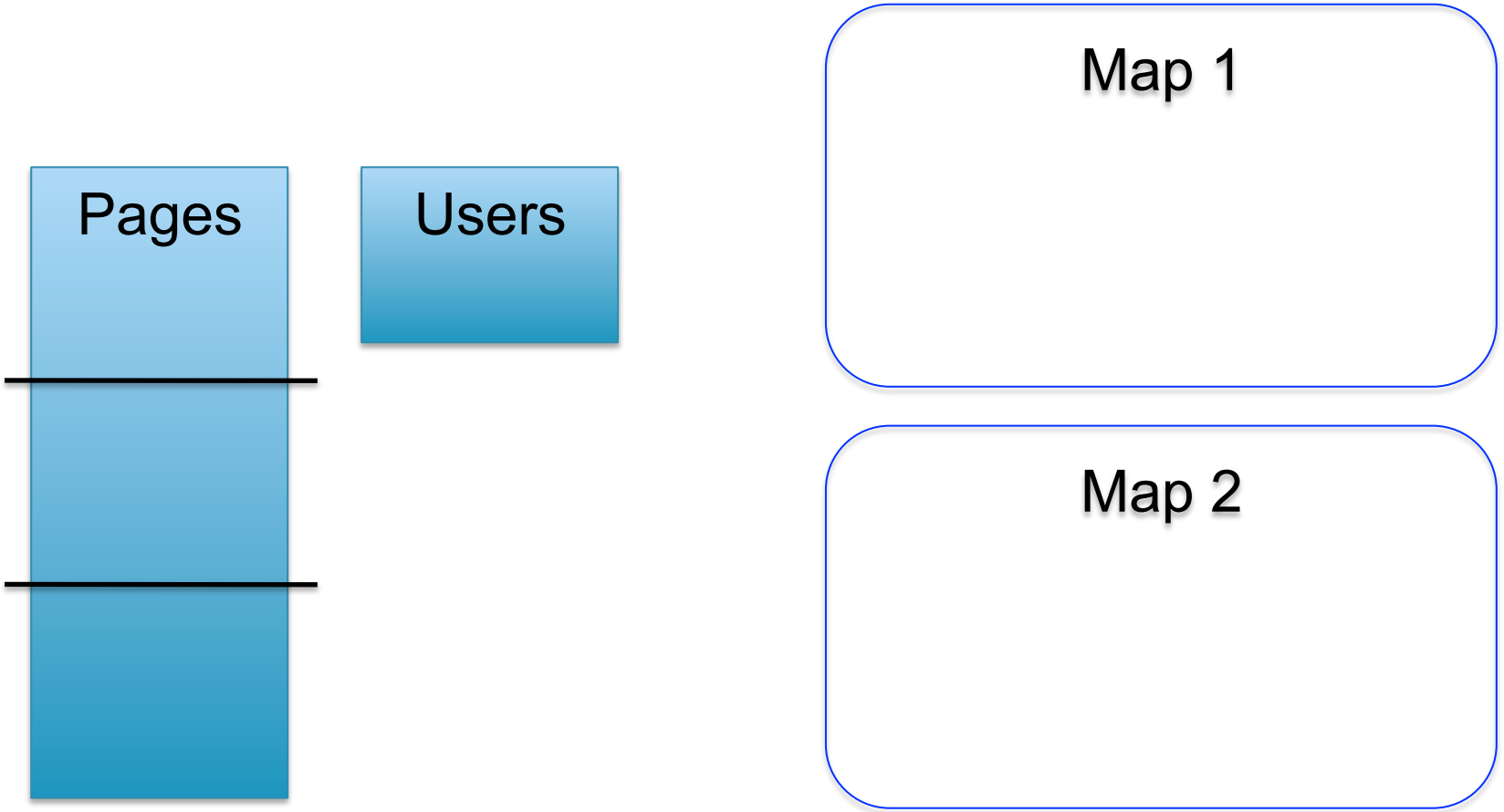
Broadcast Join in MR

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



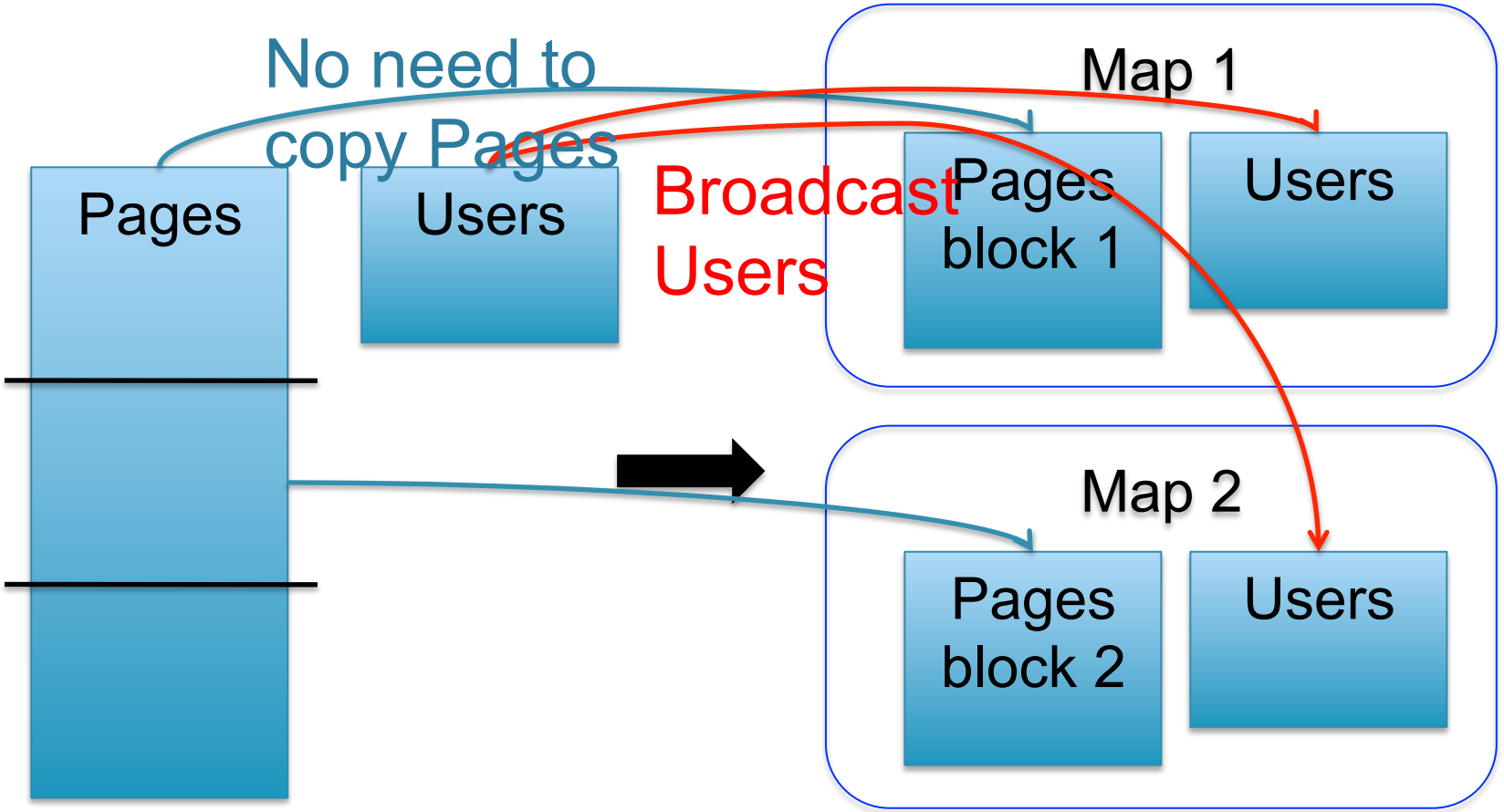
Broadcast Join in MR

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



Broadcast Join in MR

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



Broadcast Join in MR

Write the Map and Reduce functions (in class):

Matrix Multiplication v.s. Join

Dense matrices:

$$\begin{bmatrix} 6 & 6 & 0 \\ 1 & 0 & 0 \\ 2 & 0 & 6 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 3 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 \\ 0 & 2 & 0 \\ 2 & 0 & 0 \end{bmatrix}$$

forall i,k do
    $C[i,k] = \Sigma_j A[i,j] * B[j,k]$

Matrix Multiplication v.s. Join

Dense matrices:

$$\begin{bmatrix} 6 & 6 & 0 \\ 1 & 0 & 0 \\ 2 & 0 & 6 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 3 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 \\ 0 & 2 & 0 \\ 2 & 0 & 0 \end{bmatrix}$$

Sparse matrices as relations:

B(j,k,v)

j	k	v
1	1	1
1	3	3
2	2	1
3	1	2

A(i,j,v)

i	j	v
1	2	3
1	3	3
2	1	1
3	1	2

```
forall i,k do
    C[i,k] = Σ_j A[i,j] * B[j,k]
```

```
SELECT A.i, B.k, sum(A.v*B.v)
FROM A, B
WHERE A.j=B.j
GROUP BY A.i,B.i
```

Matrix Multiplication v.s. Join

Dense matrices:

$$\begin{bmatrix} 6 & 6 & 0 \\ 1 & 0 & 0 \\ 2 & 0 & 6 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 3 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 \\ 0 & 2 & 0 \\ 2 & 0 & 0 \end{bmatrix}$$

Sparse matrices as relations:

B(j,k,v)

j	k	v
1	1	1
1	3	3
2	2	1
3	1	2

A(i,j,v)

i	j	v
1	2	3
1	3	3
2	1	1
3	1	2

forall i,k do
    $C[i,k] = \Sigma_j A[i,j] * B[j,k]$

```
SELECT A.i, B.k, sum(A.v*B.v)
FROM A, B
WHERE A.j=B.j
GROUP BY A.i,B.i
```

Matrix multiplication = a join + a group by

Multiway Joins

- Have P=1000 servers
- How do we compute this query?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)

Multiway Joins

- Have P=1000 servers
- How do we compute this query?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)

- This computes all “triangles”.
- E.g. let Follows(x,y) be all pairs of Twitter users s.t. x follows y. Let R=S=T=Follows. Then Q computes all triples of people that follow each other.

Multiway Joins

- Have P=1000 servers
- How do we compute this query?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Step 1:
  - Each server sends R(x,y) to server h(y) mod P
  - Each server sends S(y,z) to server h(y) mod P

Multiway Joins

- Have P=1000 servers
- How do we compute this query?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Step 1:
  - Each server sends R(x,y) to server h(y) mod P
  - Each server sends S(y,z) to server h(y) mod P
- Step 2:
  - Each server computes R⋈S locally
  - Each server sends [R(x,y),S(y,z)] to h'(x,z) mod P
  - Each server sends T(z,x) to h'(x,z) mod P

Multiway Joins

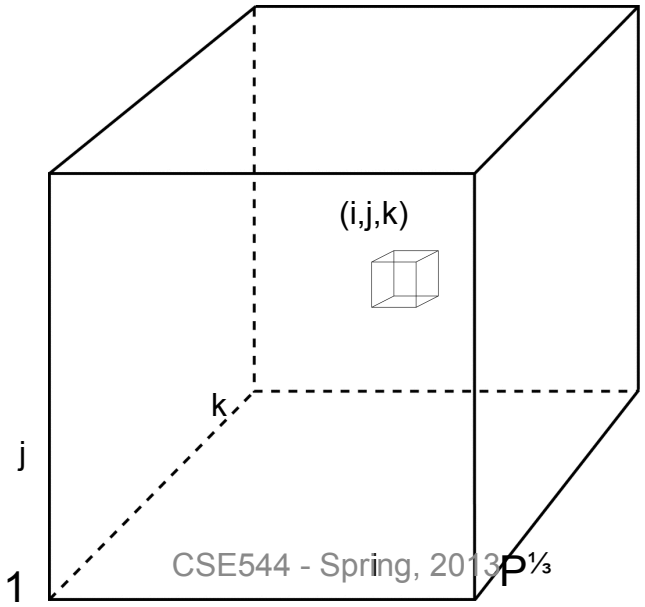
- Have P=1000 servers
- How do we compute this query?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Step 1:
  - Each server sends R(x,y) to server h(y) mod P
  - Each server sends S(y,z) to server h(y) mod P
- Step 2:
  - Each server computes R⋈S locally
  - Each server sends [R(x,y),S(y,z)] to h'(x,z) mod P
  - Each server sends T(z,x) to h'(x,z) mod P
- Final output:
  - Each server computes locally and outputs R⋈S⋈T

Multiway Joins

- Have P=1000 servers
- How do we compute this query in one step?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)

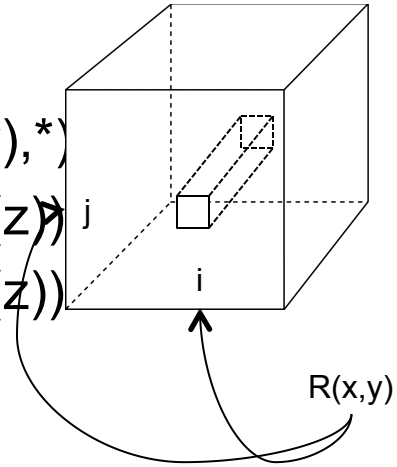
Multiway Joins

- Have P=1000 servers
- How do we compute this query in one step?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Organize the P servers into a cube 10 × 10 × 10
  - Thus, each server is uniquely identified by (i,j,k), 1≤i,j,k≤10



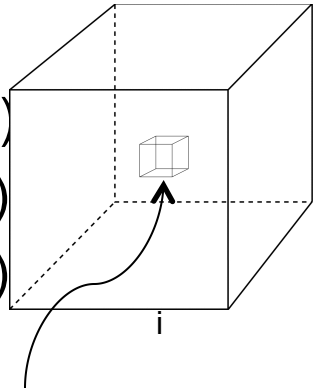
Multiway Joins

- Have P=1000 servers
- How do we compute this query in one step?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Organize the P servers into a cube 10 × 10 × 10
  - Thus, each server is uniquely identified by (i,j,k), 1≤i,j,k≤10
- Step 1:
  - Each server sends R(x,y) to all servers $(h_1(x),h_2(y),*)$
  - Each server sends S(y,z) to all servers $(*,h_2(y),h_3(z))$
  - Each server sends T(x,z) to all servers $(h_1(x),*,h_3(z))$



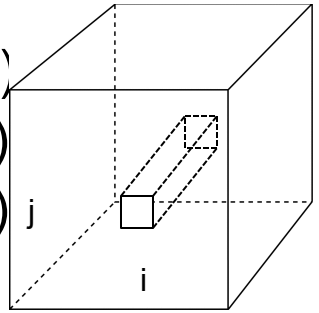
Multiway Joins

- Have P=1000 servers
- How do we compute this query in one step?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Organize the P servers into a cube 10 × 10 × 10
  - Thus, each server is uniquely identified by (i,j,k), 1≤i,j,k≤10
- Step 1:
  - Each server sends R(x,y) to all servers $(h_1(x),h_2(y),*)$
  - Each server sends S(y,z) to all servers $(*,h_2(y),h_3(z))$
  - Each server sends T(x,z) to all servers $(h_1(x),*,h_3(z))$
- Final output:
  - Each server (i,j,k) computes the query R(x,y),S(y,z),T(z,x) locally



Multiway Joins

- Have P=1000 servers
- How do we compute this query in one step?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Organize the P servers into a cube 10 × 10 × 10
  - Thus, each server is uniquely identified by (i,j,k), 1≤i,j,k≤10
- Step 1:
  - Each server sends R(x,y) to all servers $(h_1(x),h_2(y),*)$
  - Each server sends S(y,z) to all servers $(*,h_2(y),h_3(z))$
  - Each server sends T(x,z) to all servers $(h_1(x),*,h_3(z))$



- Final output:
  - Each server (i,j,k) computes the query R(x,y),S(y,z),T(z,x) locally
- Analysis: each tuple R(x,y) is replicated at most 10 times

Parallel DBMS vs MapReduce

- Parallel DBMS
  - Relational data model and schema
  - Declarative query language: SQL
  - Many pre-defined operators: relational algebra
  - Can easily combine operators into complex queries
  - Query optimization, indexing, and physical tuning
  - Streams data from one operator to the next without blocking
  - Can do more than just run queries: Data management
    - Updates and transactions, constraints, security, etc.

Parallel DBMS vs MapReduce

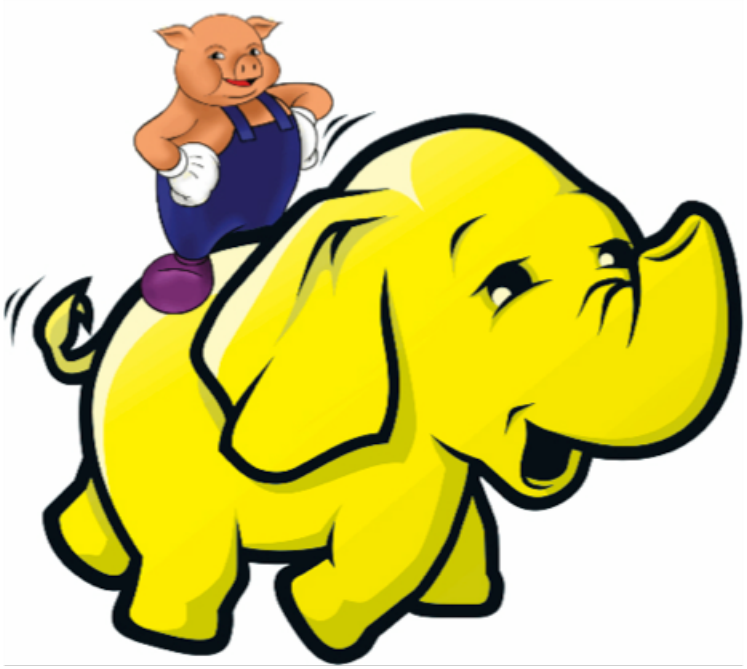
- MapReduce
  - Data model is a file with key-value pairs!
  - No need to “load data” before processing it
  - Easy to write user-defined operators
  - Can easily add nodes to the cluster (no need to even restart)
  - Uses less memory since processes one key-group at a time
  - Intra-query fault-tolerance thanks to results on disk
  - Intermediate results on disk also facilitate scheduling
  - Handles adverse conditions: e.g., stragglers
  - Arguably more scalable… but also needs more nodes!

Pig Latin
Please read on your own

Slides courtesy of: Alan Gates, Yahoo!Research

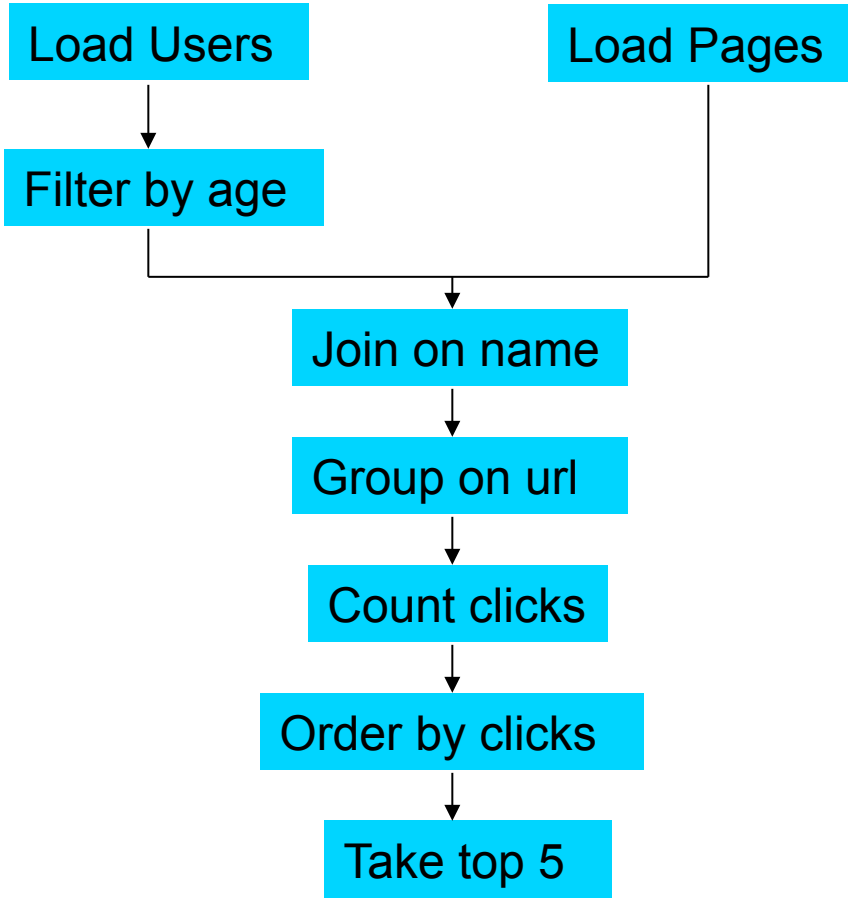
What is Pig?

- 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/



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.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