Pig: High-level Procedural Language over MapReduce

Matrix Addition

```
SELECT A.row, A.column, A.value + B.value
  FROM A, B
 WHERE A.row = B.row
   AND A.column = B.column

SELECT A.row, A.column, A.value + B.value
  FROM A INNER JOIN B ON (
          A.row = B.row
      AND A.column = B.column
)
```

```
CREATE VIEW totalNumOfDocuments AS
SELECT count(frequency.docid) as count
FROM frequency


SELECT term_id f.term, docid f.docid, f.count *
log(N.count / (
  SELECT count(docid)
    FROM frequency f1
  WHERE f.term = f1.term)
)
FROM frequency f, totalNumOfDocument as N;
```

```
CREATE VIEW termdocs as
SELECT term, count(docid) as termdocs
FROM frequency
GROUP BY term;

CREATE VIEW totaldocs as
SELECT count(distinct docid) as countalldocs
FROM frequency;

SELECT a.term, a.docid, (
       a.count * log(b.termdocs /
       c.countalldocs)
) as tfidf
FROM frequency a, termdocs b, totaldocs c
WHERE a.docid = b.docid
AND a.term = b.term;
```

```
set @TotalNumberOfDocuments = (select COUNT(distinct doc_id) from frequency)

declare @DocumentCounts table
(  term_id varchar(max),
   document_count int)

-- get the number of documents containing each term
insert into @DocumentCounts (term_id,document_count)
select term_id,COUNT(distinct doc_id)
from DATASCI250.dbo.frequency
group by term_id

-- Then calculate the TF-IDF, note that the below uses the Natural LOG
select doc_id,f.term_id,frequency * LOG(CAST(@TotalNumberOfDocuments as
Decimal(18,4))/CAST(Document_count as Decimal(18,4))) as TF_IDF
from frequency f
inner join @DocumentCounts d on f.term_id = d.term_id
```

```
select term_id, doc_id, frequency,
/* TF */
   (cast(frequency as float) /
   (select top 1 cast(frequency as float) as docFreq
      from [billhowe].[reuters_terms.csv] A
     where A.doc_id = Z.doc_id
     order by frequency desc
    ) *
/* IDF */
   (log(
     (select cast(count (distinct doc_id) as float)
        from [billhowe].[reuters_terms.csv]
      )/
     (1 + (select cast(count (distinct doc_id) as float)
             from [billhowe].[reuters_terms.csv] C
            where Z.term_id = C.term_id )
           )
   )
   )) as tfidf
     from [billhowe].[reuters_terms.csv] Z
     --order by doc_id, tfidf
```

Reflection

- Why did we do this exercise?

From the first lecture

What is Data Science about?

1) Preparing data for analysis
   (wrangling, cleaning, munging, transforming, inegrating, …)

2) Running some analysis (often a statistical model)

3) Interpreting the results and making decisions

Key challenges for Data Preparation

- Data can be very large (Volume)
- Data can be very heterogeneous and weakly structured (Variety)
- Data may be coming in faster than you can handle it (Velocity)

For Volume

- Key idea: use abstractions that allow scalable processing
  - Relational algebra and SQL
    - (not necessarily just databases)
  - MapReduce
  - Today: Pig

For Variety

- Key idea: we want to know how to work with a variety of data types
  - Matrices
  - Relations
  - Graphs (some today)
  - Text
  - Images? (maybe)

For Velocity

- We won't discuss this much in this class

Key Challenges for Running Analyses

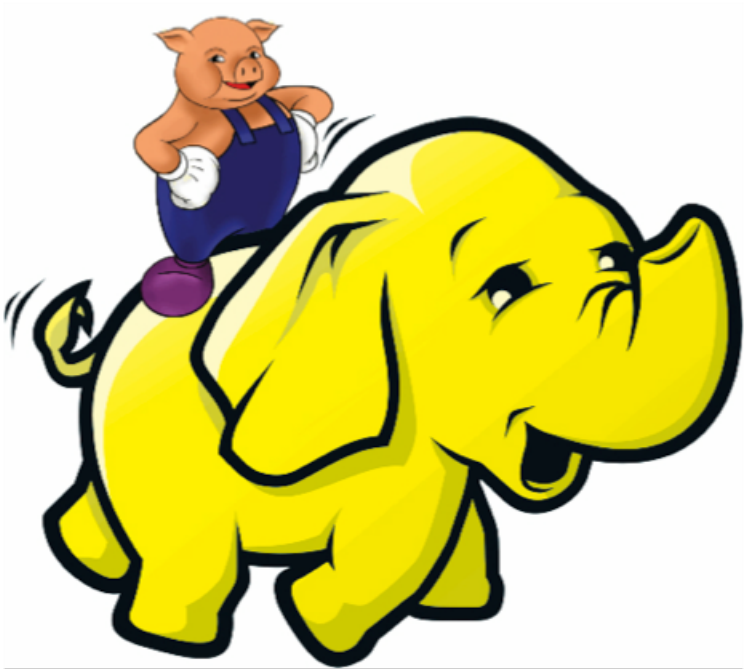
- Selecting the model
  - We will learn a toolbox of core techniques.
    - clustering (k-means)
    - optimization (forms of regression)
    - dimension reduction (multidimensional scaling)
- Running the model efficiently
  - *We will already know how to implement these models in existing systems*
    - *RA/SQL, MapReduce*
  - *We may have time to cover some new approaches (GraphLab)*

Key Challenges for Interpreting Results

- Interpreting the results (convincing yourself)
- Communicating the results (convincing others)
- *We will focus on visualization*

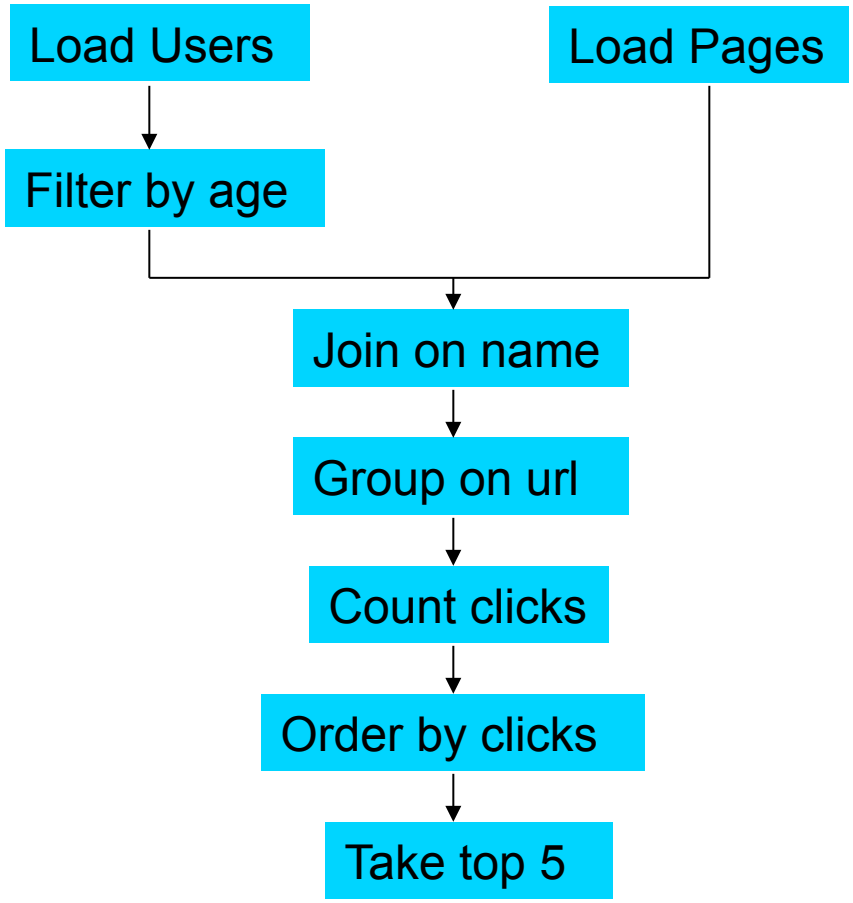
What is Pig?

- 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 MapReduce

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

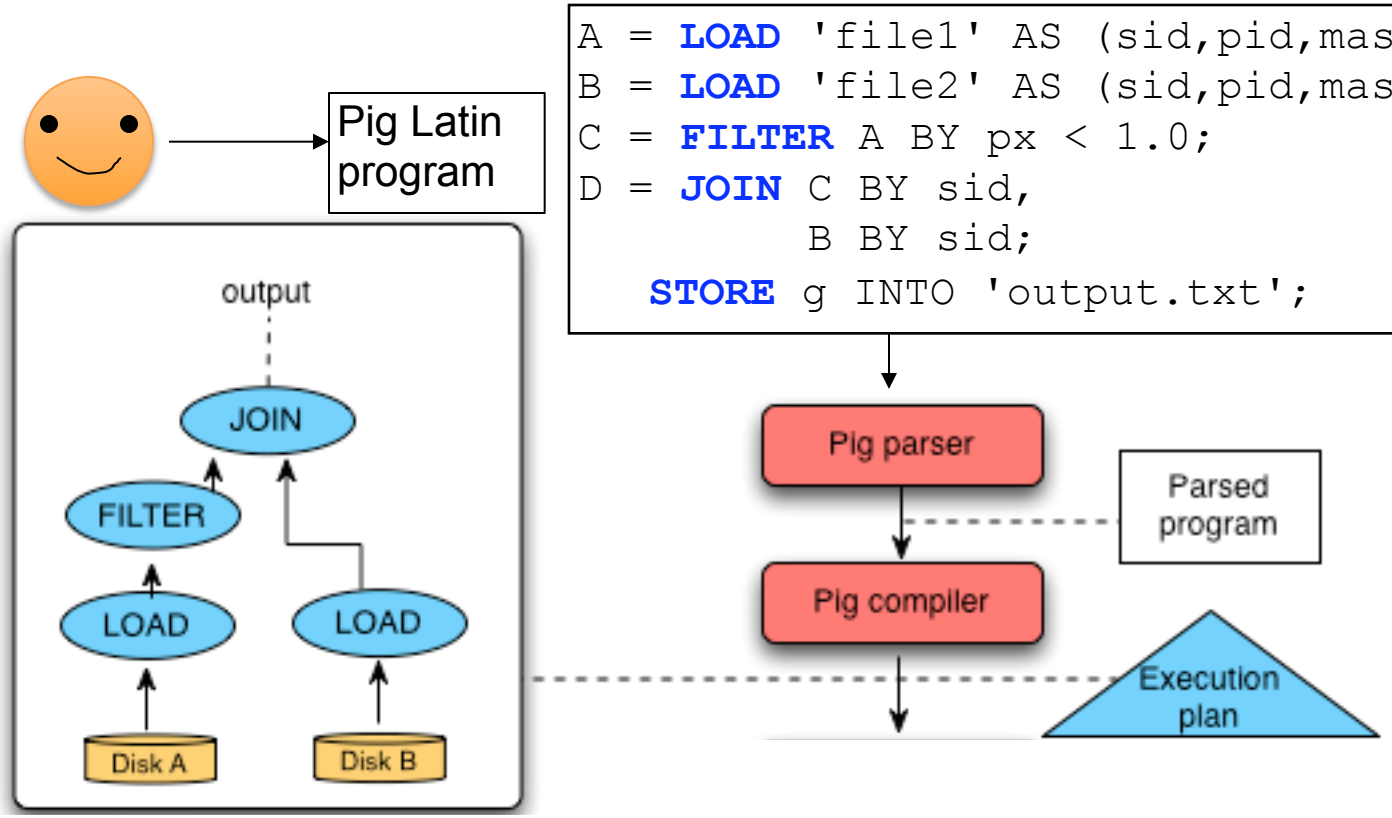
170 lines of code, 4 hours to write

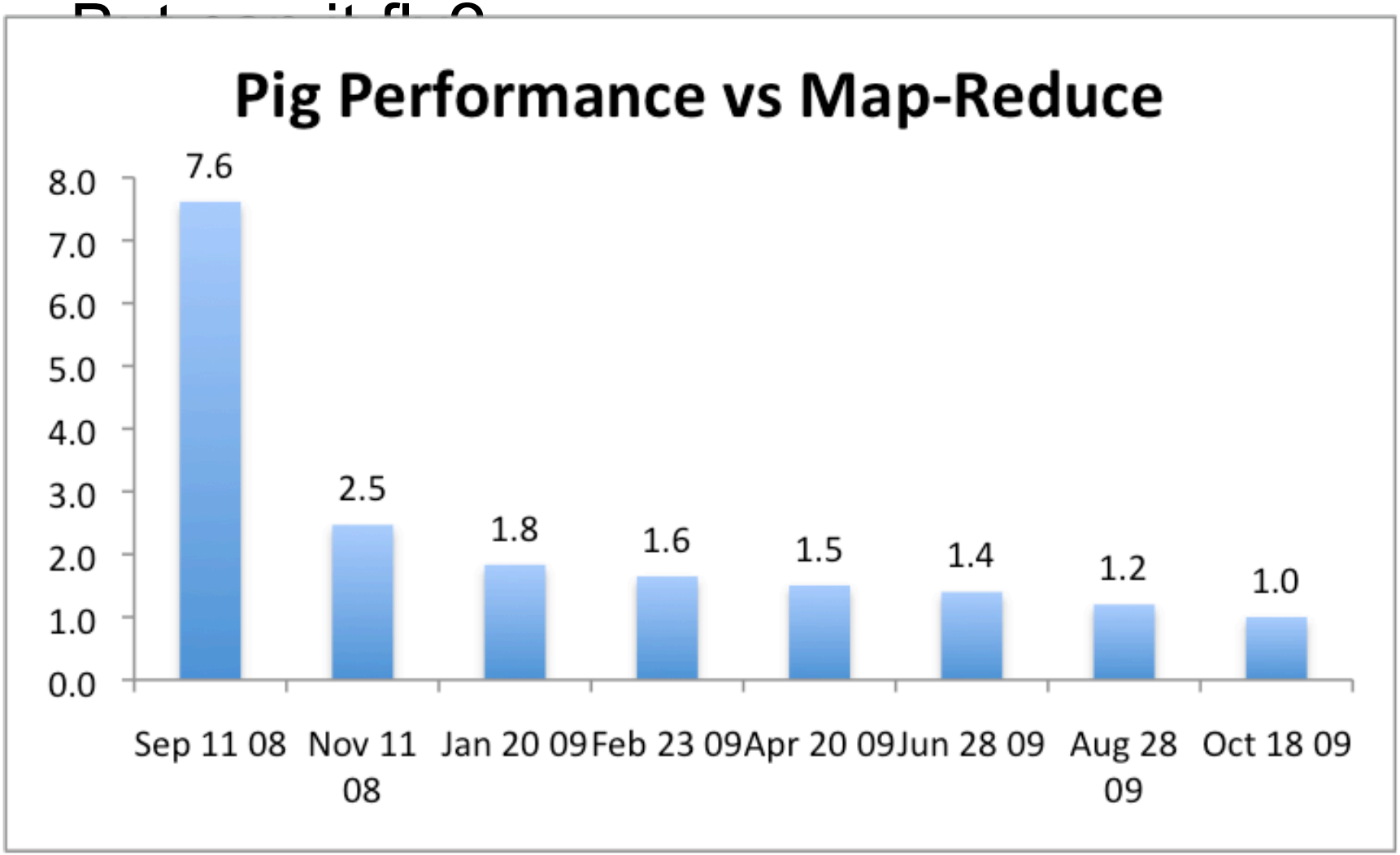
In Pig Latin

```
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

Pig System Overview




Pig Performance vs Map-Reduce
8.0
7.0
6.0
5.0
4.0
3.0
2.0
1.0
0.0
7.6
2.5
1.8
1.6
1.5
1.4
1.2
1.0
Sep 11 08
Nov 11 08
Jan 20 09
Feb 23 09
Apr 20 09
Jun 28 09
Aug 28 09
Oct 18 09

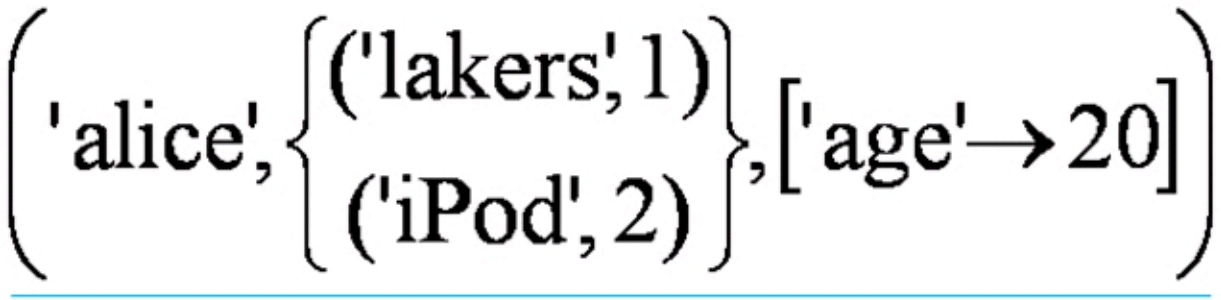
Data Model

- Atom - simple atomic value (ie: number or string)
- Tuple
- Bag
- Map

slide src: Oliver Kennedy, U of Buffalo

Data Model

- Atom
- Tuple - sequence of fields; each field any type
- Bag
- Map



slide src: Oliver Kennedy, U of Buffalo

Data Model

- Atom
- Tuple
- Bag - collection of tuples
  - Duplicates possible
  - Tuples in a bag can have different field lengths and field types
- Map

$$\left('\text{alice}', \left\{\begin{matrix}('\text{lakers}', 1) \\ ('\text{iPod}', 2)\end{matrix}\right\}, ['\text{age}' \rightarrow 20]\right)$$

slide src: Oliver Kennedy, U of Buffalo

Data Model

- Atom
- Tuple
- Bag
- Map - collection of key-value pairs
  - Key is an atom; value can be any type

$$\left(\text{'alice'}, \left\{\begin{matrix}(\text{'lakers'}, 1)\\(\text{'iPod'}, 2)\end{matrix}\right\}, [\text{'age'} \rightarrow 20]\right)$$

slide src: Oliver Kennedy, U of Buffalo

Speaking Pig Latin

- LOAD
  - Input is assumed to be a bag (sequence of tuples)
  - Can specify a deserializer with "USING"
  - Can provide a schema with "AS"

```
newBag = LOAD 'filename'
   <USING functionName() >
   <AS (fieldName1, fieldName2,…)>;

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

slide src: Oliver Kennedy, U of Buffalo

Speaking Pig Latin

- FOREACH
  - Apply some processing to each tuple in a bag
  - Each field can be:
    - A fieldname of the bag
    - A constant
    - A simple expression (ie: f1+f2)
    - A predefined function (ie: SUM, AVG, COUNT, FLATTEN)
    - A UDF (ie: sumTaxes(gst, pst))

```
newBag =
  FOREACH bagName
  GENERATE field1, field2, …;
```

slide src: Oliver Kennedy, U of Buffalo

Speaking Pig Latin

- FILTER
  - Select a subset of the tuples in a bag

```
newBag = FILTER bagName
           BY   expression   ;
```

  - Expression uses simple comparison operators (==, !=, <, >, …) and Logical connectors (AND, NOT, OR)

```
some_apples =
   FILTER apples BY colour != 'red';
```

  - Can use UDFs

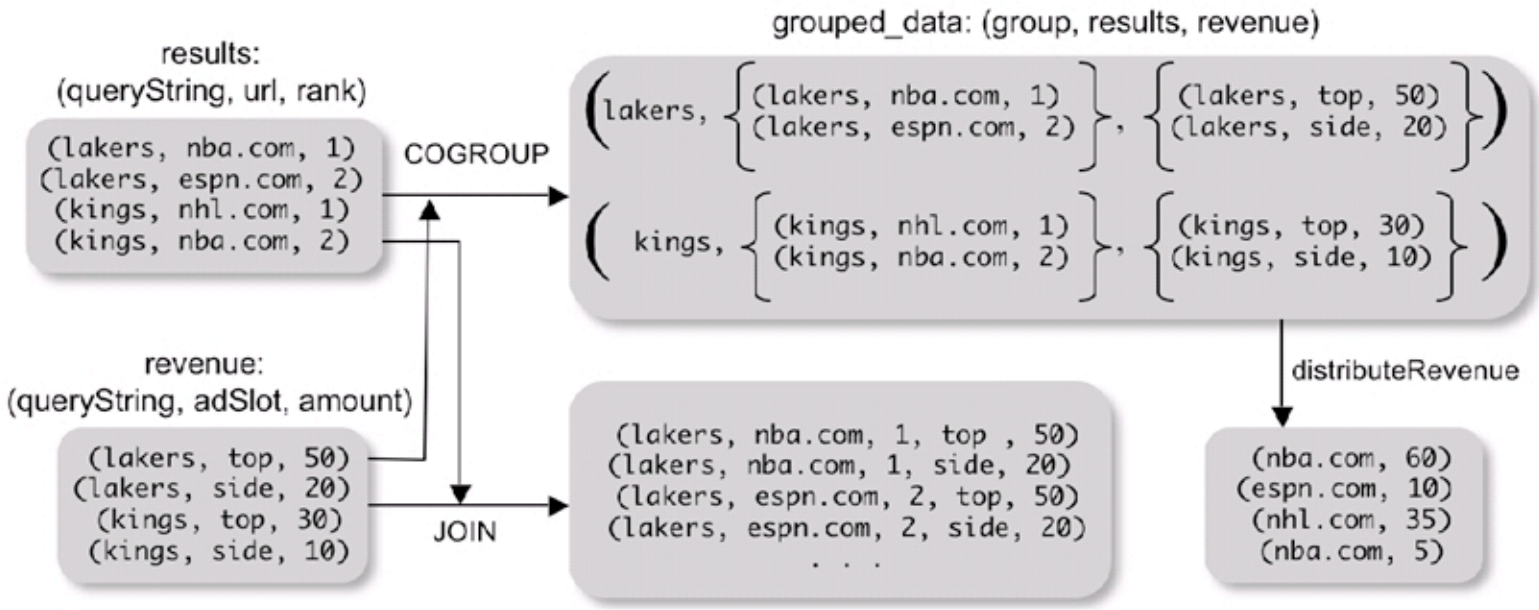
```
some_apples =
   FILTER apples BY NOT isRed(colour);
```

slide src: Oliver Kennedy, U of Buffalo

Speaking Pig Latin

- COGROUP
  - Group two datasets together by a common attribute
  - Groups data into nested bags

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

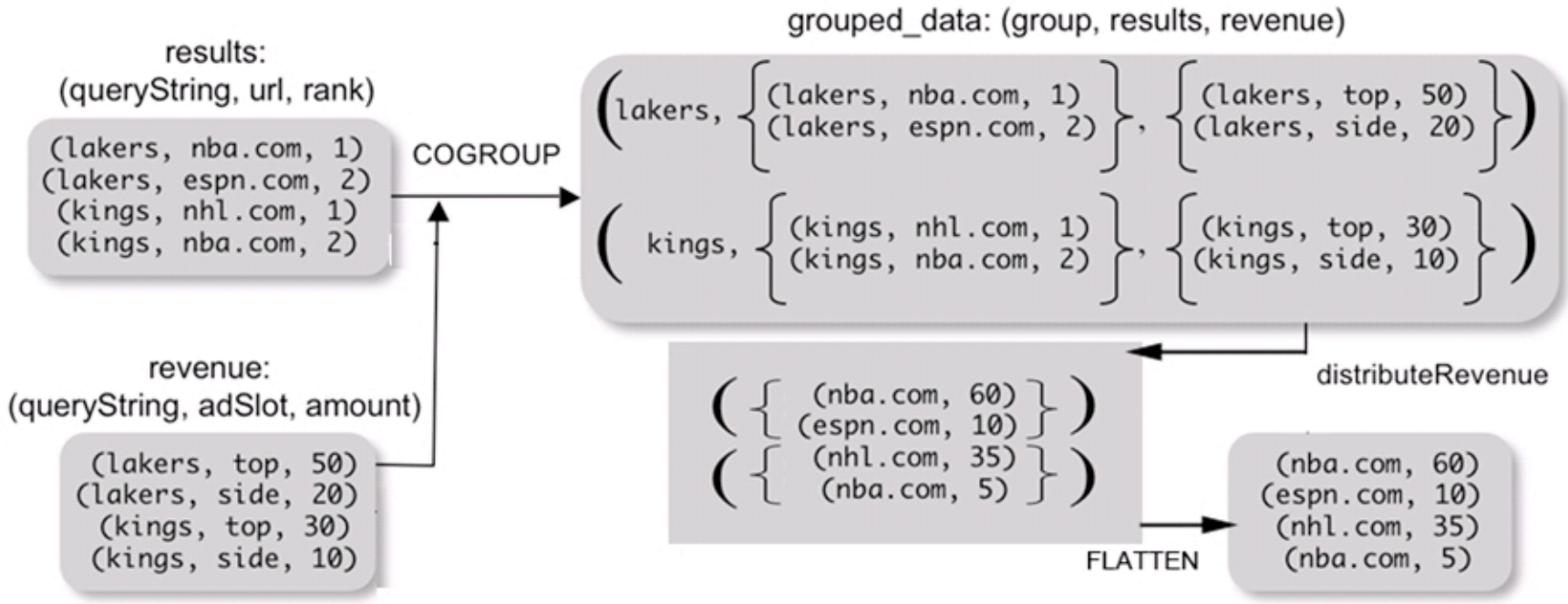


slide src: Oliver Kennedy, U of Buffalo

Speaking Pig Latin

- Why COGROUP and not JOIN?

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



slide src: Oliver Kennedy, U of Buffalo

Speaking Pig Latin

- Why COGROUP and not JOIN?
  - May want to process nested bags of tuples before taking the cross product.
  - Keeps to the goal of a single high-level data transformation per pig-latin statement.
  - However, JOIN keyword is still available:

```
JOIN results BY queryString,
     revenue BY queryString;
```

Equivalent

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

slide src: Oliver Kennedy, U of Buffalo

Speaking Pig Latin

- STORE (& DUMP)
  - Output data to a file (or screen)

```
STORE bagName INTO 'filename'
      <USING deserializer ()>;
```

- Other Commands (incomplete)
  - UNION - return the union of two or more bags
  - CROSS - take the cross product of two or more bags
  - ORDER - order tuples by a specified field(s)
  - DISTINCT - eliminate duplicate tuples in a bag
  - LIMIT - Limit results to a subset

Compilation

- Pig system does two tasks:
  - Builds a Logical Plan from a Pig Latin script
    - Supports execution platform independence
    - No processing of data performed at this stage
  - Compiles the Logical Plan to a Physical Plan and Executes
    - Convert the Logical Plan into a series of Map-Reduce statements to be executed (in this case) by Hadoop Map-Reduce

Compilation

- Building a Logical Plan
  - Verify input files and bags referred to are valid
  - Create a logical plan for each bag(variable) defined

Compilation

- Building a Logical Plan Example

```
A = LOAD 'user.dat' AS (name, age, city);
B = GROUP A BY city;
C = FOREACH B GENERATE group AS city,
       COUNT(A);
D = FILTER C BY city IS 'kitchener'
       OR city IS 'waterloo';
STORE D INTO 'local_user_count.dat';
```

Compilation

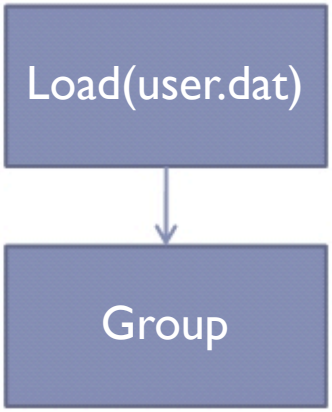
- Building a Logical Plan Example

```
A = LOAD 'user.dat' AS (name, age, city);
B = GROUP A BY city;
C = FOREACH B GENERATE group AS city,
       COUNT(A);
D = FILTER C BY city IS 'kitchener'
       OR city IS 'waterloo';
STORE D INTO 'local_user_count.dat';
```

Compilation

- Building a Logical Plan Example

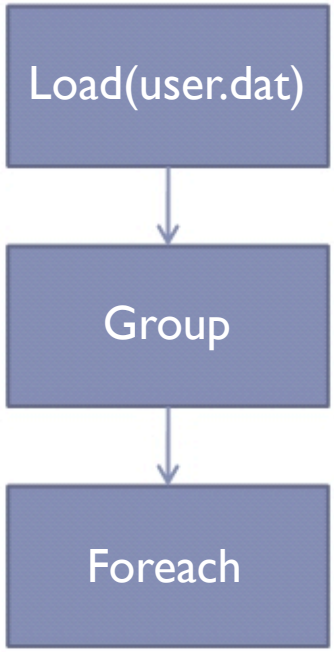
```
A = LOAD 'user.dat' AS (name, age, city);
B = GROUP A BY city;
C = FOREACH B GENERATE group AS city,
       COUNT(A);
D = FILTER C BY city IS 'kitchener'
       OR city IS 'waterloo';
STORE D INTO 'local_user_count.dat';
```



Compilation

- Building a Logical Plan Example

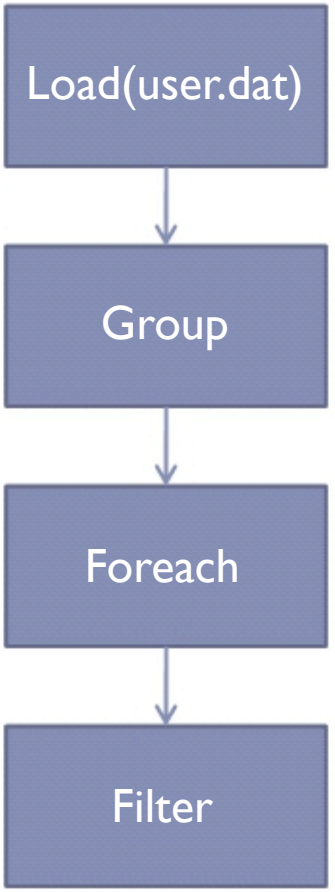
```
A = LOAD 'user.dat' AS (name, age, city);
B = GROUP A BY city;
C = FOREACH B GENERATE group AS city,
       COUNT(A);
D = FILTER C BY city IS 'kitchener'
       OR city IS 'waterloo';
STORE D INTO 'local_user_count.dat';
```



Compilation

- Building a Logical Plan Example

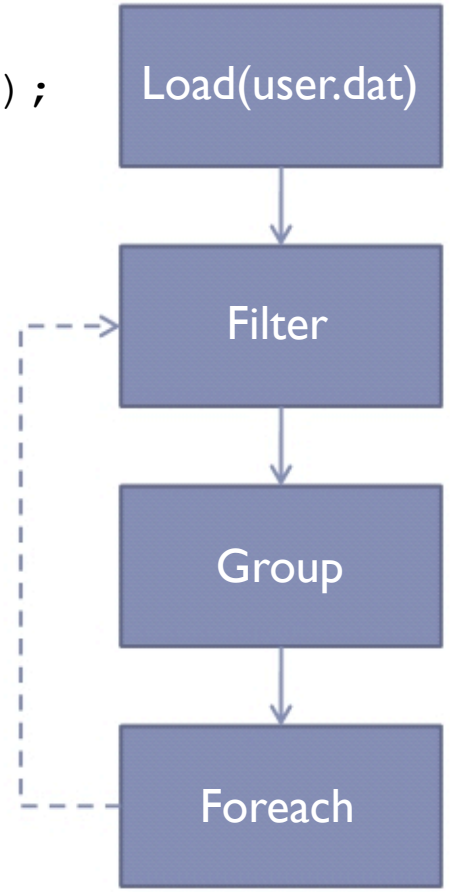
```
A = LOAD 'user.dat' AS (name, age, city);
B = GROUP A BY city;
C = FOREACH B GENERATE group AS city,
       COUNT(A);
D = FILTER C BY city IS 'kitchener'
       OR city IS 'waterloo';
STORE D INTO 'local_user_count.dat';
```



Compilation

- Building a Logical Plan Example

```
A = LOAD 'user.dat' AS (name, age, city);
B = GROUP A BY city;
C = FOREACH B GENERATE group AS city,
       COUNT(A);
D = FILTER C BY city IS 'kitchener'
       OR city IS 'waterloo';
STORE D INTO 'local_user_count.dat';
```

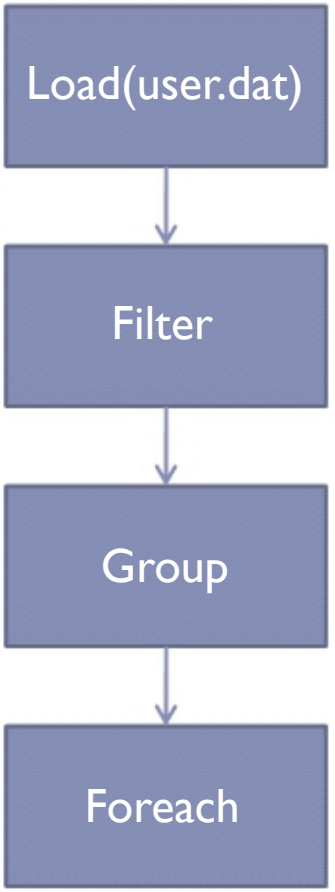


Compilation

- Building a Physical Plan

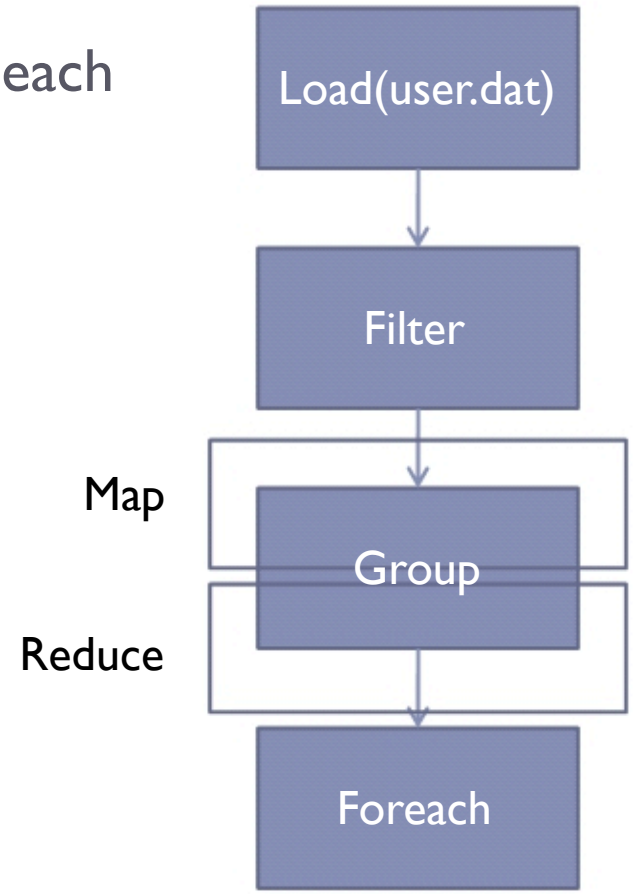
```
A = LOAD 'user.dat' AS (name, age, city);
B = GROUP A BY city;
C = FOREACH B GENERATE group AS city,
       COUNT(A);
D = FILTER C BY city IS 'kitchener'
       OR city IS 'waterloo';
STORE D INTO 'local_user_count.dat';
```

Only happens when output is specified by STORE or DUMP



Compilation

- Building a Physical Plan
  - Step 1: Create a map-reduce job for each COGROUP



Compilation

- Building a Physical Plan
  - Step 1: Create a map-reduce job for each COGROUP
  - Step 2: Push other commands into the map and reduce functions where possible
  - May be the case certain commands require their own map-reduce job (ie: ORDER needs separate map-reduce jobs)