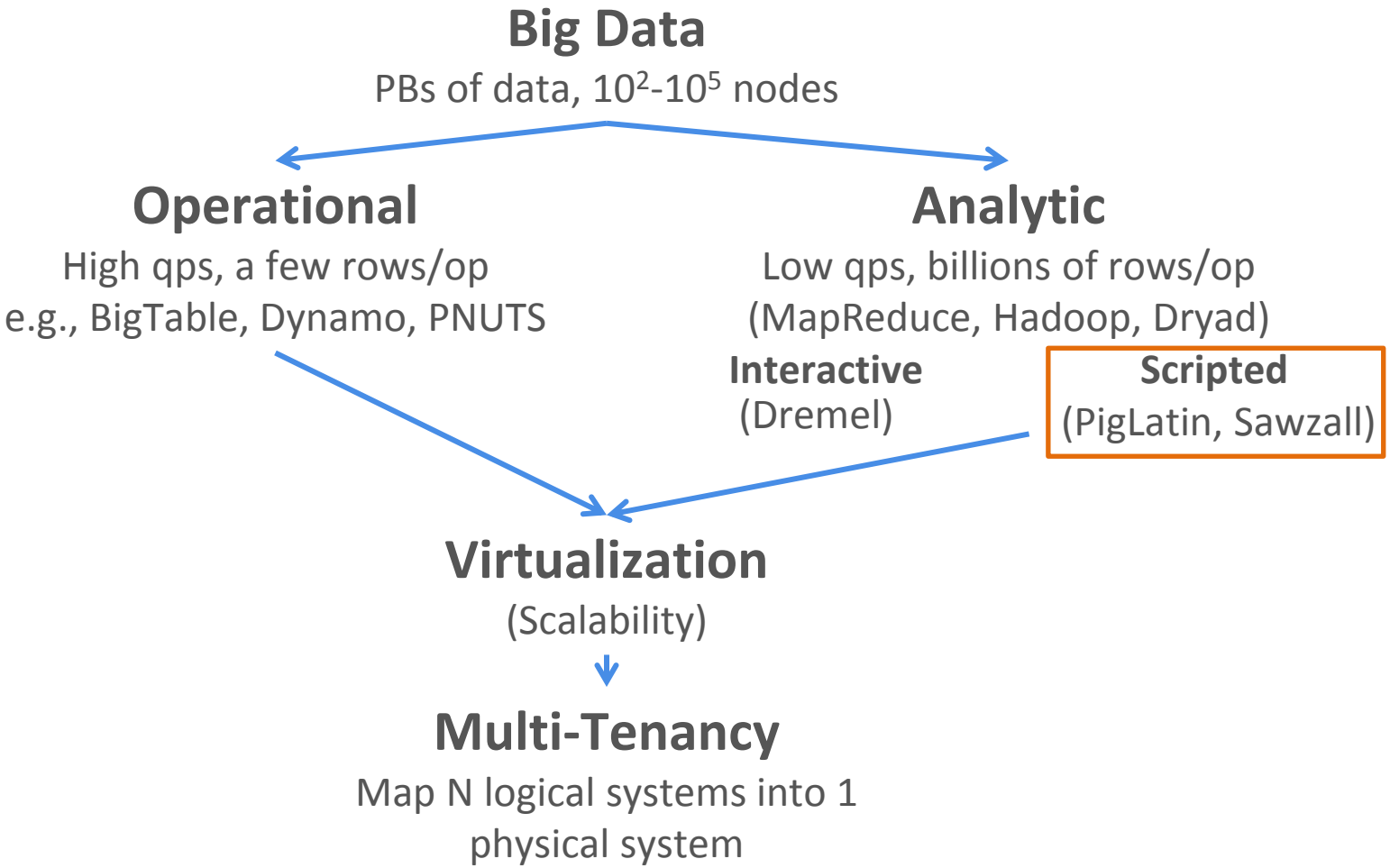
3 Alternative Query Languages



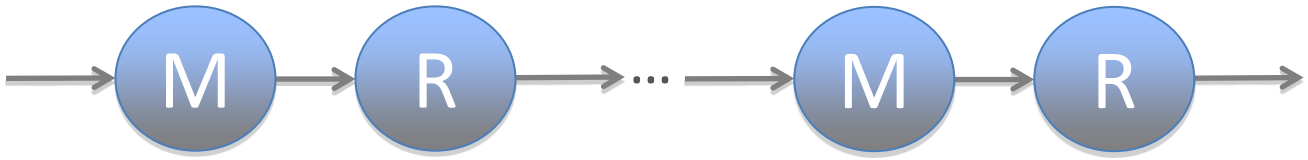
[S. Melnik: The Frontiers of Data Programmability, BTW 2009]

Big demand for parallel data processing

New systems for data analysis

- Emerging tools that do not look like SQL DBMS
  - MapReduce
  - Apache Hadoop
  - Dryad
- Programmers like dataflow pipes over static files
- Hence the excitement about MapReduce
- But, MapReduce is too low-level and rigid

Extremely rigid data flow



Common operations must be coded by hand

- join, filter, split, projection, aggregates, sorting, distinct

User plans may be suboptimal and lead to performance degradation

Semantics hidden inside map-reduce functions

- Inflexible, difficult to maintain, extend and optimize

Combination of high-level declarative querying and low-level programming with MapReduce → Dataflow Programming Languages (PigLatin and co)

High-level Dataflow Languages

Pig JAQL Sawzall DryadLINQ

Parallel Data Flow Systems

Hadoop MapReduce Dryad

Ansätze für Sprachabstraktionen

Verschiedene Vorschläge für Skriptsprachen, die Konstrukte auf höherer Abstraktionsebene anbieten, z.B.

- Sawzall
  - Bei Google entwickelte und benutzte Skriptsprache, die auf Googles MapReduce-Implementierung aufsetzt
  - Rob Pike, Sean Dorward, Robert Griesemer, Sean Quinlan. Interpreting the Data: Parallel Analysis with Sawzall
- DryadLINQ
  - Basierend auf der Microsoft-Umsetzung von MapReduce Dryad (http://research.microsoft.com/en-us/projects/dryad/)
- Pig
  - Skriptsprache, die ursprünglich als Forschungsprojekt bei Yahoo im Mai 2006 begann
  - Ziel: MapReduce-Programme generieren (z.B. für Hadoop)
  - http://wiki.apache.org/incubator/PigProposal

Verschiedene Vorschläge für Skriptsprachen, die Konstrukte auf höherer Abstraktionsebene anbieten, z.B.

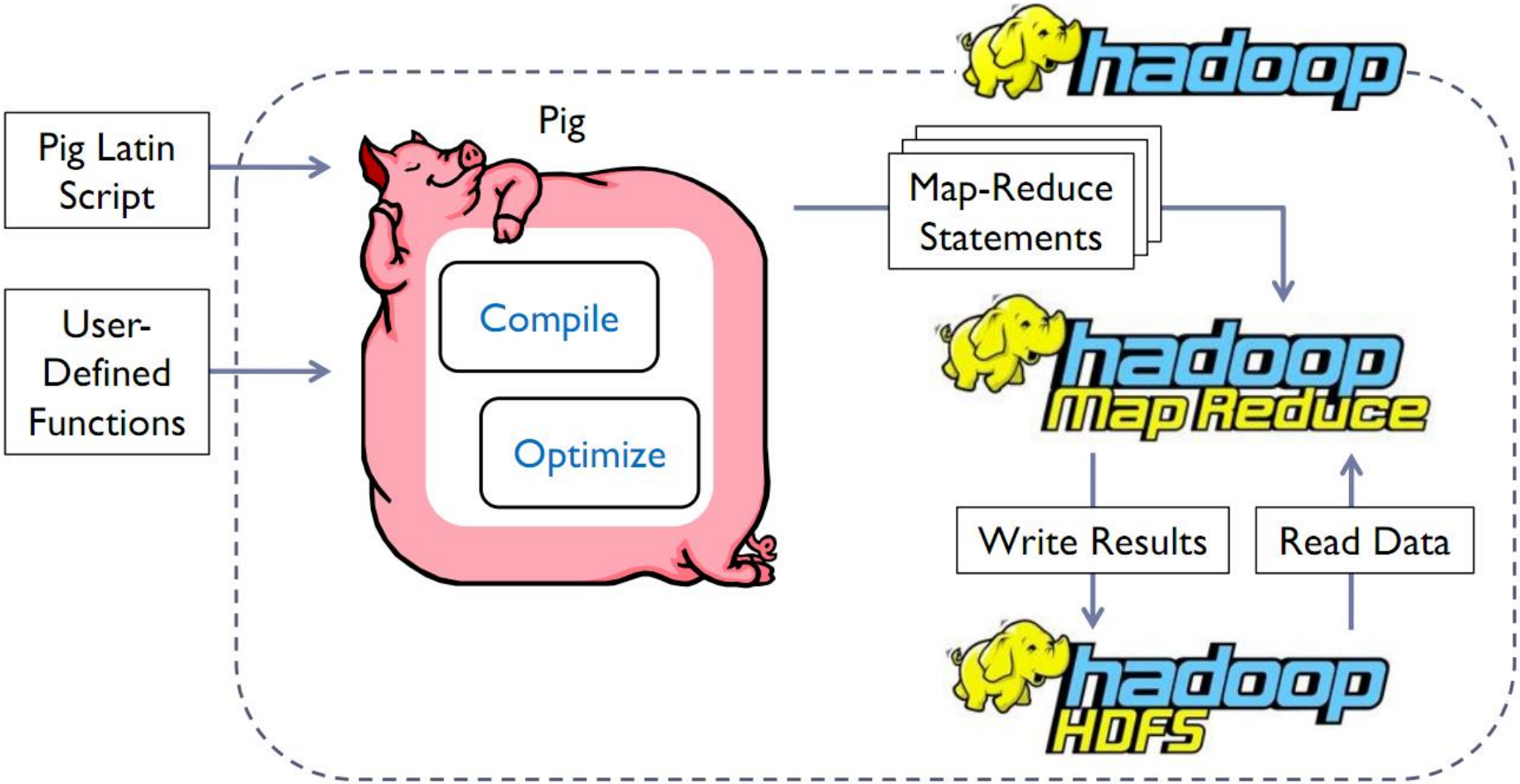
- Dremel (Google)
  - System zur Ausführung interaktiver Ad-hoc-Anfragen auf großen Datenmengen
  - Geschachtelte Datensätze
  - Spaltenbasiertes Layout

Pig Latin: A Not-So-Foreign Language For Data Processing

PigLatin

- On top of MapReduce/ Hadoop
- Mix of declarative style of SQL and procedural style of MapReduce
- Consists of two parts
  - PigLatin: A Data Processing Language
  - Pig Infrastructure: An Evaluator for PigLatin programs
- Pig compiles Pig Latin into physical plans
- Plans are to be executed over Hadoop

- 30% of all queries at Yahoo! in Pig-Latin
- Open-source, http://pig.apache.org/


hadoop
Pig
Pig Latin Script
User-Defined Functions
Compile
Optimize
Map-Reduce Statements
hadoop MapReduce
Write Results
Read Data
hadoop HDFS

Target users are entrenched procedural programmers

The step-by-step method of creating a program in Pig is much cleaner and simpler to use than the single block method of SQL. It is easier to keep track of what your variables are, and where you are in the process of analyzing your data.

Jasmine Novak
Engineer, Yahoo!

With the various interleaved clauses in SQL, it is difficult to know what is actually happening sequentially. With Pig, the data nesting and the temporary tables get abstracted away. Pig has fewer primitives than SQL does, but it's more powerful.

David Ciemiewicz
Search Excellence, Yahoo!

- Automatic query optimization is hard
- Pig Latin does not preclude optimization

Pig Latin – First Example

Suppose we have a table

- urls: (url, category, pagerank)

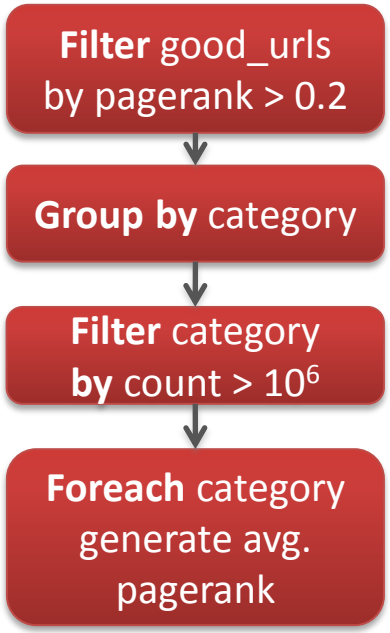
For each sufficiently large category, the average pagerank of high-pagerank urls in that category

Simple SQL query

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

Equivalent Pig Latin program

```
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);
```



Pig Latin program supply an explicit sequence of operations, it is not necessary that the operations be executed in that order

- e.g., Set of urls of pages classified as spam, but have a high pagerank score

```
spam_urls    = FILTER urls BY isSpam(url);
culprit_urls = FILTER spam_urls BY pagerank > 0.8;
```

- isSpam might be an expensive UDF → better to filter the url by pagerank first

Task

- Determine the most visited websites in each category

Visits

User	URL	Time
Amy	cnn.com	8:00
Amy	bbc.com	10:00
Amy	flickr.com	10:05
Fred	cnn.com	12:00

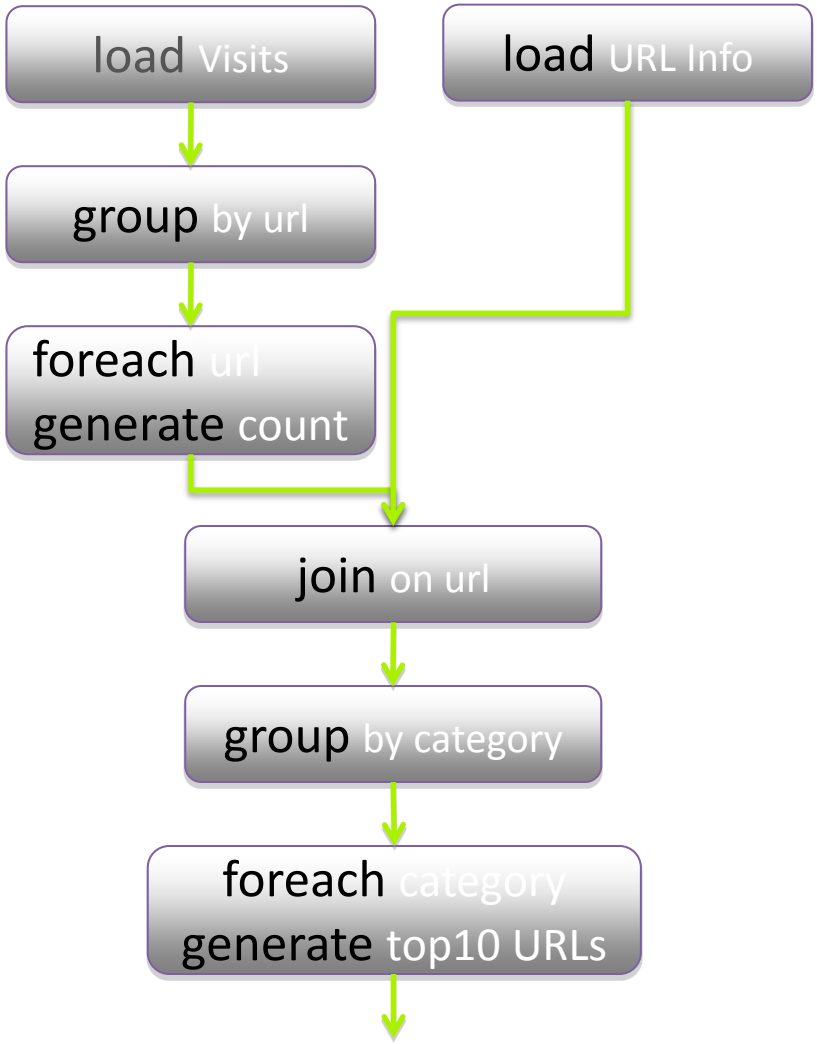
URL Info

URL	Category	PageRank
cnn.com	News	0.9
bbc.com	News	0.8
flickr.com	Photos	0.7
espn.com	Sports	0.9

```
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();
    }
}
```

> Example Workflow in Pig-Latin

```
visits       = LOAD '/data/visits'
               AS (user, url, time);

gVisits      = GROUP visits BY url;

visitCounts  = FOREACH gVisits
               GENERATE url, count(visits);

urlInfo      = LOAD '/data/urlInfo'
               AS (url, category, pRank);

visitCounts  = JOIN visitCounts
               BY url, urlInfo BY url;

gCategories  = GROUP visitCounts
               BY category;

topUrls      = FOREACH gCategories
               GENERATE top(visitCounts,10);


STORE topUrls INTO '/data/topURLs';
```



> Example Workflow in Pig-Latin

```
visits       = LOAD '/data/visits'
               AS (user, url, time);

gVisits      = GROUP visits BY url;

visitCounts  = FOREACH gVisits
               GENERATE url, count(visits);

urlInfo      = LOAD '/data/urlInfo'
               AS (url, category, pRank);

visitCounts  = JOIN visitCounts
               BY url, urlInfo by url;

gCategories  = GROUP visitCounts
               BY category;

topUrls      = FOREACH gCategories
               GENERATE top(visitCounts,10);

STORE topUrls INTO '/data/topURLs';
```

Operate directly over files.

Schemas optional. Can be assigned dynamically.

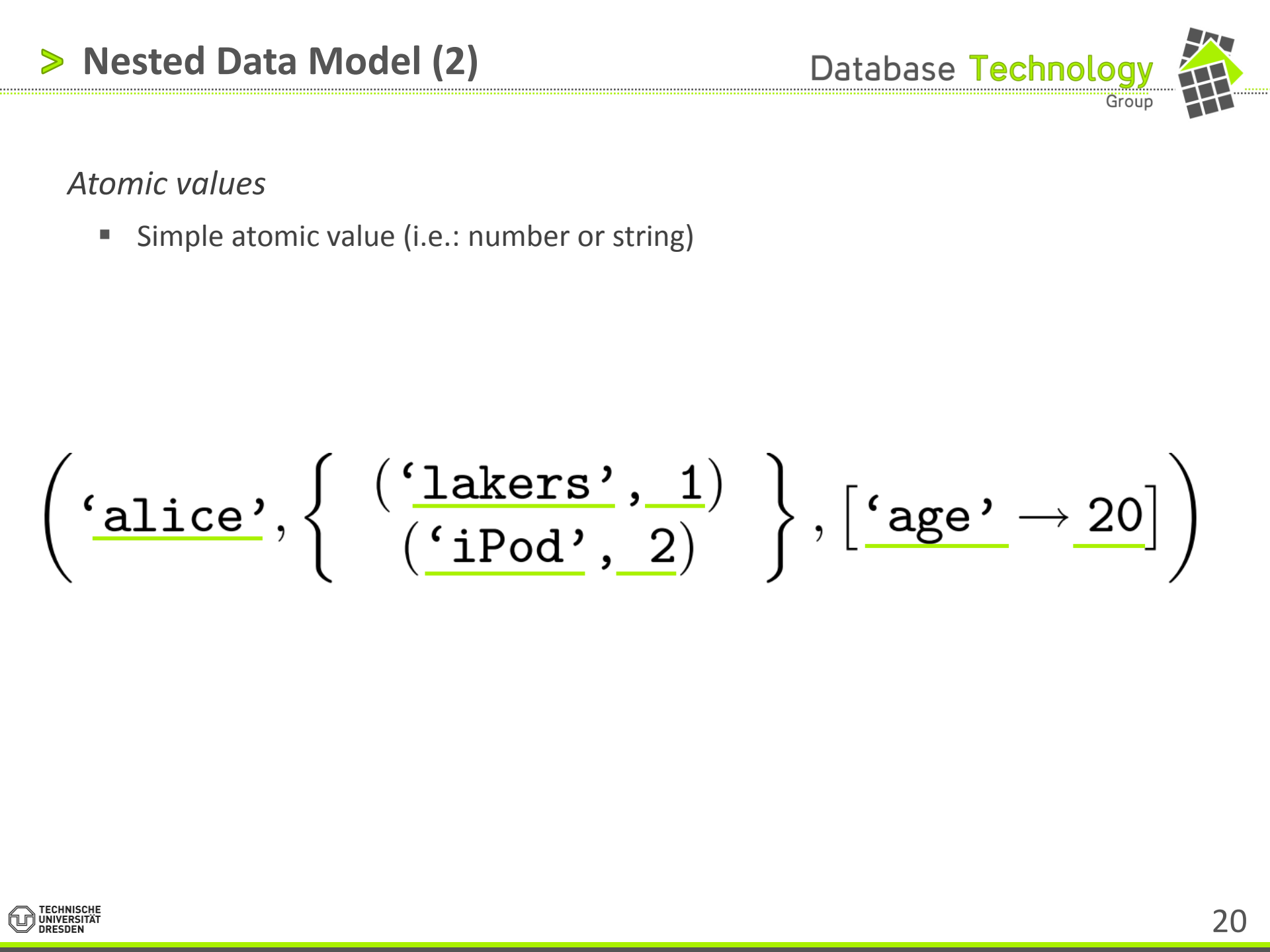
User-defined functions (UDFs) can be used in every construct
- load, store
- group, filter, foreach

Pig Latin has a fully-nestable data model with

- Allows complex, non-atomic data types such as sets, map, and tuple
- Nested Model is more closer to programmer than normalization (1NF)
- Avoids expensive joins for web-scale data
- Allows programmer to easily write UDFs
- More natural to programmers than flat tuples

Atomic values, tuples, bags (lists), and maps

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



Atomic values

- Simple atomic value (i.e.: number or string)

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

Tuple

- Sequence of fields; each field any type

$$\Big(\text{`alice'}, \Big\{ \begin{array}{c} (\text{`lakers'}, 1) \\ (\text{`iPod'}, 2) \end{array} \Big\}, [\text{`age'} \rightarrow 20]\Big)$$

Bag

- Collection of tuples
- Duplicates possible
- Tuples in a bag can have different field lengths and field types

$$\Big(\text{`alice'}, \Big\{ \begin{array}{c} (\text{`lakers'}, 1) \\ (\text{`iPod'}, 2) \end{array} \Big\}, [\text{`age'} \rightarrow 20]\Big)$$

Map

- Collection of key-value pairs
- Key is an atom; value can be any type

$$\Big(\text{`alice'}, \Big\{ \begin{array}{c} (\text{`lakers'}, 1) \\ (\text{`iPod'}, 2) \end{array} \Big\}, [\text{`age'} \rightarrow 20]\Big)$$

Specifying Input Data – LOAD

- Input is assumed to be a bag (sequence of tuples)
- Can specify a serializer with “USING”
- Can provide a schema with “AS”

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

- Example

```
queries = LOAD ‘query_log.txt’
          USING myLoad()
          AS (userId, queryString, timestamp);
```

Per-tuple Processing – 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 (i.e.: SUM, AVG, COUNT, FLATTEN)
  - A UDF (i.e.: sumTaxes(gst, pst))

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

- Example

```
expand_queries = FOREACH queries
                 GENERATE userId, expandQuery(queryString);
```

Discarding Unwanted Data – 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';
```

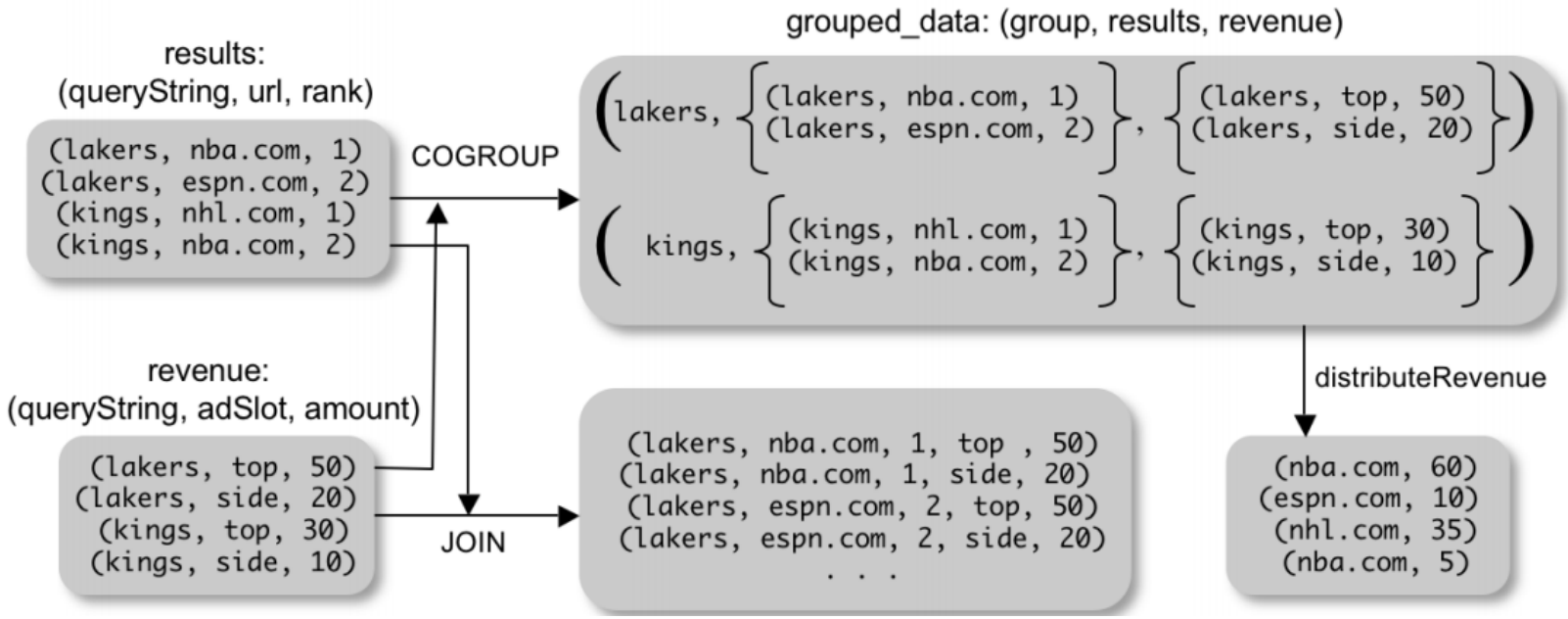
- Can use UDFs

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

COGROUP

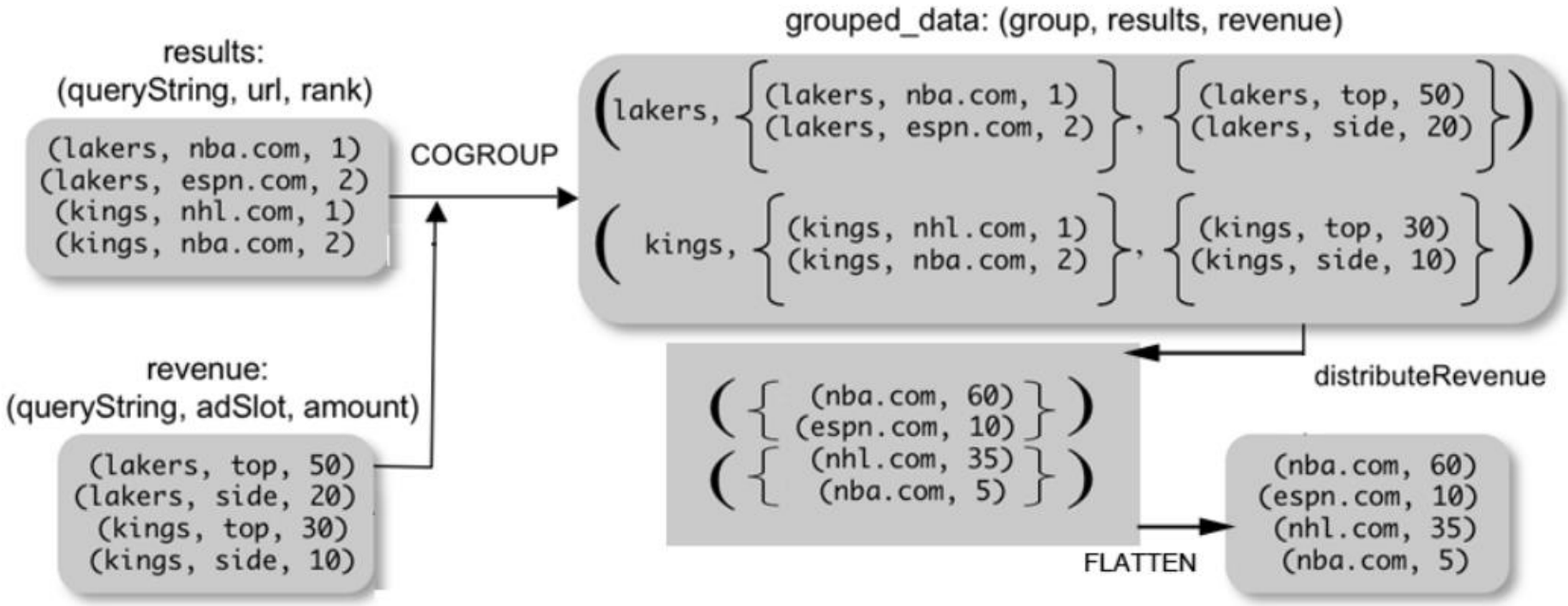
- Group two datasets together by a common attribute
- Groups data into nested bags

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



Why COGROUP and not JOIN?

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



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)
```

STORE (& DUMP)

- Output data to a file (or screen)

```
A = LOAD 'input' AS (x, y, z);
B = FILTER A BY x > 5;
DUMP B;
C = FOREACH B GENERATE y, z;
STORE C INTO 'output';
```

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

Pig provides extensive support for user-defined functions (UDFs) as a way to specify custom processing

- Functions can be a part of almost every operator in Pig
- Useful for custom processing tasks
- Can use non-atomic values for input and output
- Currently must be written in Java, integrated as jar file

Different types

- Simple eval functions
  - Most common type of function
  - Used in FOREACH statements
  - Example

```
REGISTER myudfs.jar;
A = LOAD 'student_data' AS (name: chararray, age: int,
                            gpa: float);
B = FOREACH A GENERATE myudfs.UPPER(name);
DUMP B;
```

Different types (cont'd)

- Aggregate functions
  - Usually applied to grouped data
  - Takes a bag and returns a scalar value
  - Most function can be computed incrementally in a distributed fashion (e.g. COUNT, not MEDIAN) → partial computations done by the map and combiner, final result computed by reducer
  - Example

```
A = LOAD 'student_data' AS (name: chararray, age: int,
                            gpa: float);
B = GROUP A BY name;
C = FOREACH B GENERATE group, COUNT(A);
DUMP C;
```

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-MapReduce

Building a Logical Plan

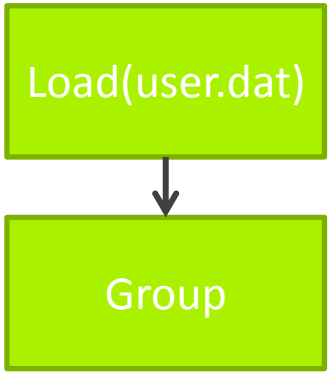
- Verify input files and bags referred to are valid
- Create a logical plan for each bag defined

Load(user.dat)

```
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';
```

Building a Logical Plan

- Verify input files and bags referred to are valid
- Create a logical plan for each bag defined

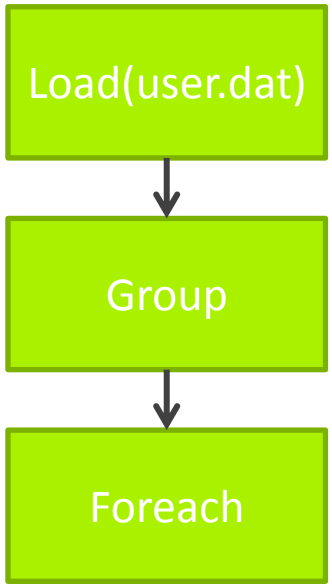



```
A = LOAD 'user.dat' AS (name, age, city);
B = GROUP A BY city;
C = FOREACH B GENERATE group AS city, COUNT(A);
D = FILTER C BY city IS 'kitchener'
              OR city IS 'waterloo';
STORE D INTO 'local_user_count.dat';
```

Building a Logical Plan

- Verify input files and bags referred to are valid
- Create a logical plan for each bag defined

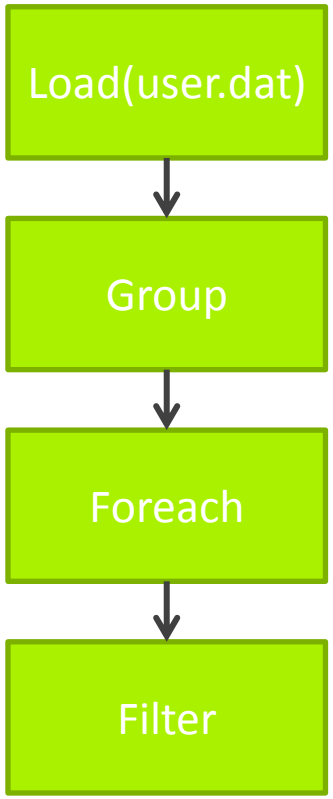
```
A = LOAD 'user.dat' AS (name, age, city);
B = GROUP A BY city;
C = FOREACH B GENERATE group AS city, COUNT(A);
D = FILTER C BY city IS 'kitchener'
              OR city IS 'waterloo';
STORE D INTO 'local_user_count.dat';
```



Building a Logical Plan

- Verify input files and bags referred to are valid
- Create a logical plan for each bag defined

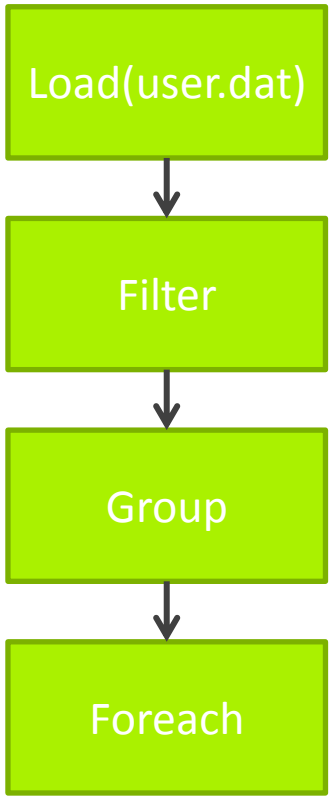
```
A = LOAD 'user.dat' AS (name, age, city);
B = GROUP A BY city;
C = FOREACH B GENERATE group AS city, COUNT(A);
D = FILTER C BY city IS 'kitchener'
              OR city IS 'waterloo';
STORE D INTO 'local_user_count.dat';
```



Building a Logical Plan

- Verify input files and bags referred to are valid
- Create a logical plan for each bag defined

```
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';
```



Other Logical Optimization Techniques

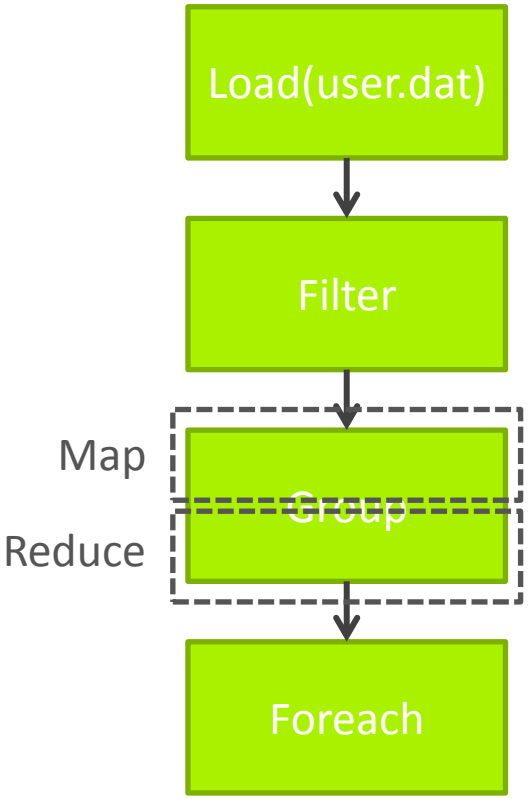
- Push Down Explodes – Perform FLATTEN operations after JOIN where possible
- Push Limits Up – Perform LIMIT operations as soon as possible to avoid unnecessary processing of intermediate data
- And a few others having to do with splitting output, avoiding reloading data sets, and type-casting

"Cookbook" available online for tips and tricks on how to structure Pig Latin commands for better performance

- http://pig.apache.org/docs/r0.7.0/cookbook.html
- Use types
- Project Early and Often
- Filter Early and Often
- Reduce Your Operator Pipeline
- Make Your UDFs Algebraic
- Use the PARALLEL Clause (determines the number of reduce tasks)

Building a Physical Plan

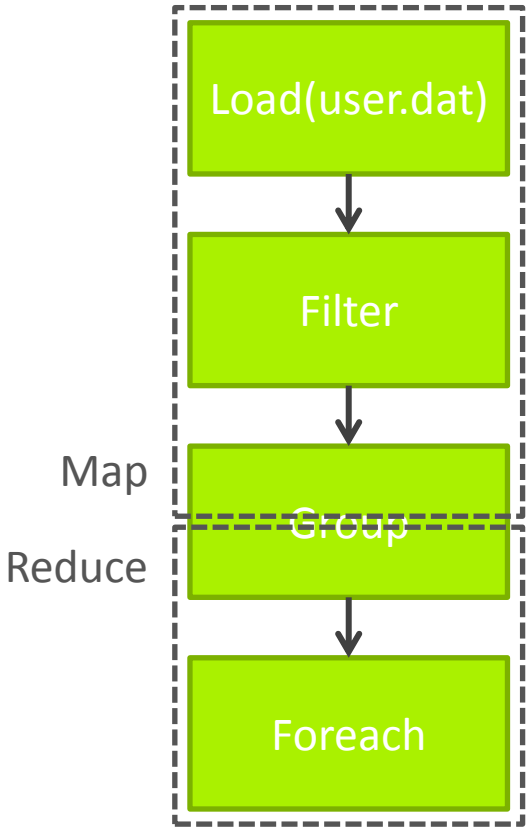
- MapReduce provides the ability to do a large-scale group by (map tasks assign keys for grouping, reduce tasks process a group at a time)
- Every (CO)GROUP or JOIN operation forms a map-reduce boundary
- Step 1: Create a map-reduce job for each (CO)GROUP



Building a Physical Plan

- MapReduce provides the ability to do a large-scale group by (map tasks assign keys for grouping, reduce tasks process a group at a time)
- Every (CO)GROUP or JOIN operation forms a map-reduce boundary
- Step 1: Create a map-reduce job for each COGROUP
- Step 2: Push other commands into the map and reduce functions where possible

- Commands that intervene between subsequent (CO)GROUP commands C_i and C_{i+1} can be pushed into
  (a) the reduce function corresponding to C_i
  (b) the map function corresponding to C_{i+1}
- Some commands require their own map-reduce job (e.g. ORDER)



Pig-Pen

Operators

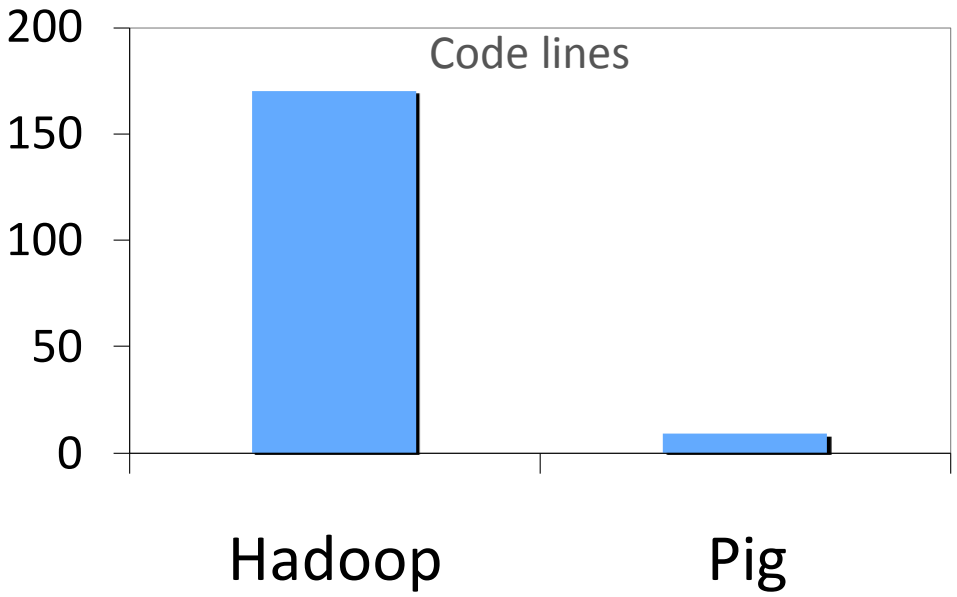
LOAD | GROUP | COGROUP | FILTER | FOREACH | ORDER

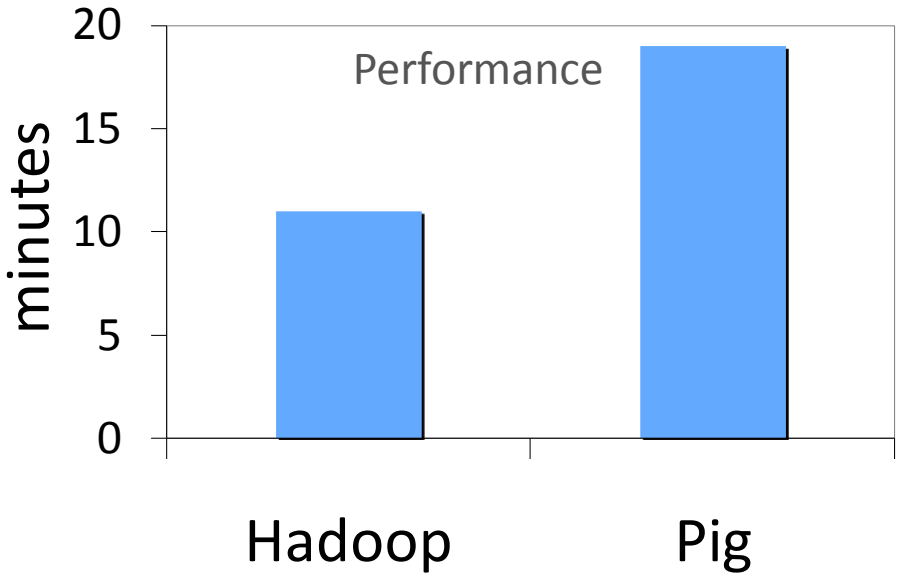
= LOAD USING Default AS ( )

Generate Query

```
visits = LOAD 'visits.txt' AS (user, url, time);

pages = LOAD 'pages.txt' AS (url, pagerank);

v_p = JOIN visits BY url, pages BY url;

users = GROUP v_p BY user;

useravg = FOREACH users GENERATE group, AVG(v_p.pagerank) AS avgpr;

answer = FILTER useravg BY avgpr > '0.5';
```

```
visits:   (Amy, cnn.com, 8am)
          (Amy, frogs.com, 9am)
          (Fred, snails.com, 11am)

pages:    (cnn.com, 0.8)
          (frogs.com, 0.8)
          (snails.com, 0.3)

v_p:      (Amy, cnn.com, 8am, cnn.com, 0.8)
          (Amy, frogs.com, 9am, frogs.com, 0.8)
          (Fred, snails.com, 11am, snails.com, 0.3)

users:    (Amy, { (Amy, cnn.com, 8am, cnn.com, 0.8),
                  (Amy, frogs.com, 9am, frogs.com, 0.8) })
          (Fred, { (Fred, snails.com, 11am, snails.com, 0.3) })

useravg:  (Amy, 0.8)
          (Fred, 0.3)

answer:   (Amy, 0.8)
```


Code lines
200
150
100
50
0
Hadoop
Pig


Performance
minutes
20
15
10
5
0
Hadoop
Pig

HIVE

Data warehouse infrastructure built on top of Hadoop, providing

- Data Summarization
- Ad hoc querying

Simple query language

- Hive QL (based on SQL)
- Extendable via custom mappers and reducers
- Subproject of Hadoop
- http://hadoop.apache.org/hive/

Example

```
LOAD DATA INPATH `/data/visits` INTO TABLE visits

INSERT OVERWRITE TABLE visitCounts
SELECT url, category, count(*)
FROM visits
GROUP BY url, category;

LOAD DATA INPATH '/data/urlInfo' INTO TABLE urlInfo

INSERT OVERWRITE TABLE visitCounts
SELECT vc.*, ui.*
FROM visitCounts vc JOIN urlInfo ui ON (vc.url = ui.url);

INSERT OVERWRITE TABLE gCategories
SELECT category, count(*)
FROM visitCounts
GROUP BY category;

INSERT OVERWRITE TABLE topUrls
SELECT TRANSFORM (visitCounts) USING 'top10';
```

JAQL
Query Language for JavaScript(r) Object Notation (JSON)

Higher level query language for JSON documents

Developed at IBM's Almaden research center

Supports several operations known from SQL

- Grouping, Joining, Sorting

Built-in support for

- Loops, Conditionals, Recursion

Custom Java methods extend JAQL

JAQL scripts are compiled to MapReduce jobs

Various I/O

- Local FS, HDFS, Hbase, Custom I/O adapters

http://www.jaql.org/

```
registerFunction(„top“, „de.tuberlin.cs.dima.jaqlextensions.top10“);

$visits = hdfsRead(„/data/visits“);

$visitCounts =
$visits
-> group by $url = $
        into { $url, num: count($)};

$urlInfo = hdfsRead(„data/urlInfo“);

$visitCounts =
join $visitCounts, $urlInfo
where $visitCounts.url == $urlInfo.url;

$gCategories =
$visitCounts
-> group by $category = $
        into {$category, num: count($)};

$topUrls = top10($gCategories);

hdfsWrite(“/data/topUrls”, $topUrls);
```

Comparison of JAQL, Hive, Pig and Java-MR

Benchmarks (Widely Used)

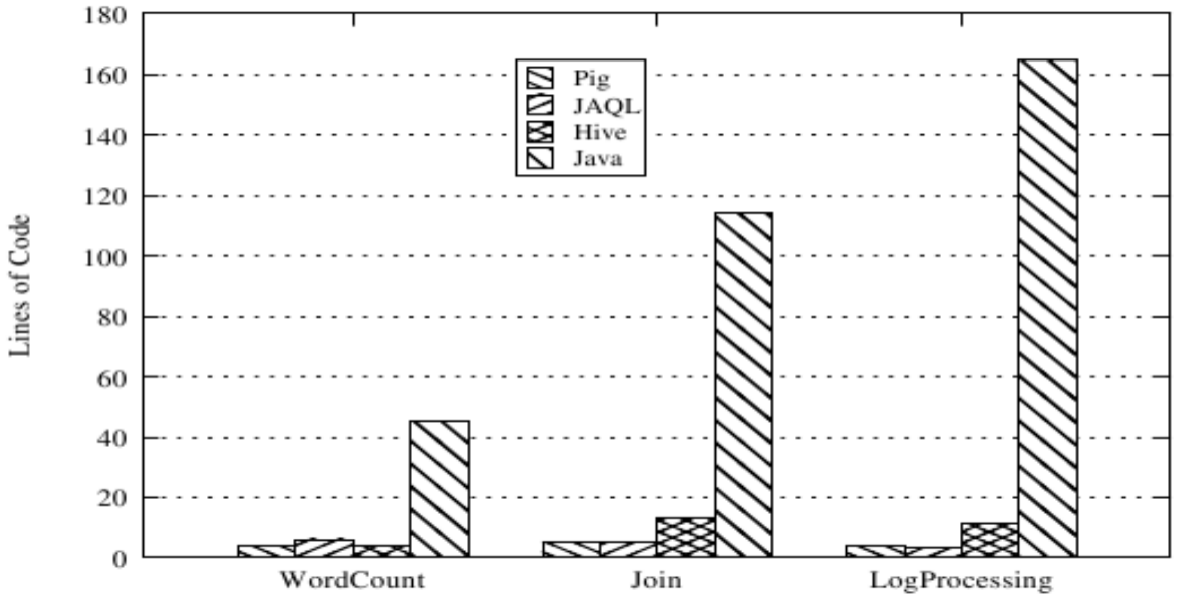
- Word Count (Read large text file; output list of distinct words with frequency)
- Dataset Join (Reads two datasets; join on occurrences of identical items)
- Webserver Log Processing (Reads webserver log file; aggregates page counts & average viewing time)
- Designed by author, performs (SQL)

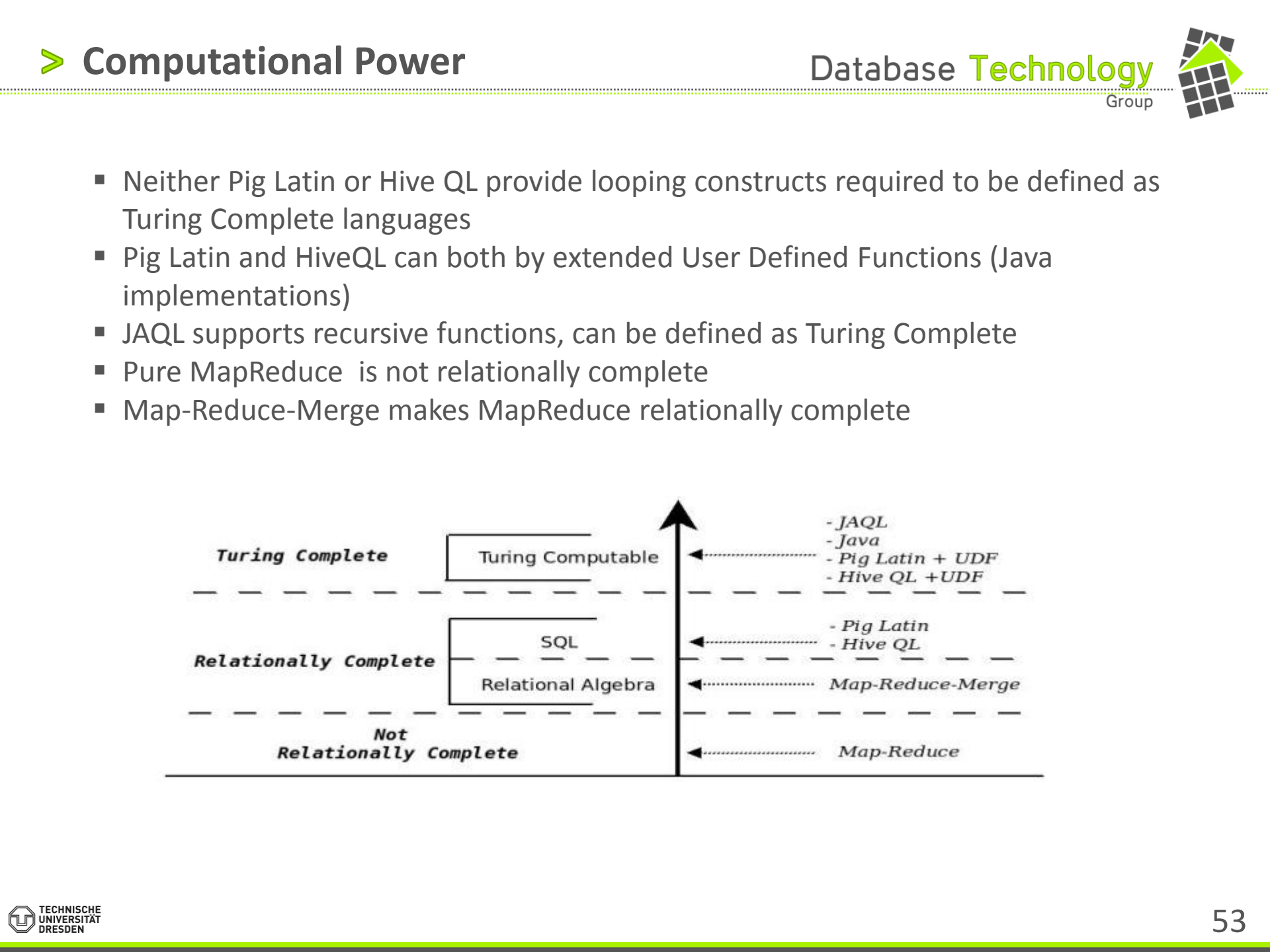
```
SELECT userID, AVG(timeOnSite) AS averages, COUNT(pageID)
GROUP BY userID;
```

- Convergence (A Turing complete graph algorithm)

Result

- The programs in all three high level languages (Hive, Pig and JAQL) are far shorter than the Java equivalent
  - By at least a factor of 7.5
  - Word count – Java is 45 lines, JAQL is 6 lines
  - Log processing – Java is 165 lines, JAQL is 3 lines
- Programmers spend less time writing and debugging large applications





- Neither Pig Latin or Hive QL provide looping constructs required to be defined as Turing Complete languages
- Pig Latin and HiveQL can both by extended User Defined Functions (Java implementations)
- JAQL supports recursive functions, can be defined as Turing Complete
- Pure MapReduce is not relationally complete
- Map-Reduce-Merge makes MapReduce relationally complete

Need to bridge the gap between parallel data-low systems and high-level dataflow languages

→ PigLatin, Sazwall, JAQL, Hive

PigLatin

- A data processing environment in Hadoop that is specifically targeted towards procedural programmers who perform large-scale data analysis
- Offers high-level data manipulation in a procedural style
- But performance of Pig queries tend to be slower than a pure MapReduce implementation
- Pig-Pen is a debugging environment for Pig-Latin commands that generates samples from real data