High-level Big Data Query Languages: Pig and Hive

Prof. Wing C. Lau
Department of Information Engineering
wclau@ie.cuhk.edu.hk

Acknowledgements

- The slides used in this chapter are adapted from the following sources:
  - CS498 Cloud Computing, by Roy Campbell and Reza Farivar, UIUC.
  - 15-319 Cloud Computing, by M. F. Sakr and M. Hammoud, CMU Qatar
  - CS525 Special Topics in DBs: Large-scale Data Management, by Mohamed Eltabakh, WPI, Spring 2013
  - CS345D Topics in Database Management, by Semih Salihoglu, Stanford
  - Olston et al, "Pig Latin: A Not-So-Foreign Language for Data Processing," ACM Sigmod 2008 presentation.
  - Perry Hoekstra, Jiaheng Lu, Avinash Lakshman, Prashant Malik, and Jimmy Lin, "NoSQL and Big Data Processing, BigTable, Hbase, Cassandra, Hive and Pig"
  - Cloudera Training Slides for Pig, Hive and Hbase

- All copyrights belong to the original authors of the materials

Need for High-Level Languages

- Hadoop/MapReduce is great for large-data processing!
  - But writing Java programs for everything is verbose and slow
  - Not everyone wants to (or can) write Java code
- Solution: develop higher-level data processing languages
  - Pig: Pig Latin is a bit like Perl
    - By Yahoo!
  - Hive: HQL is like SQL
    - By Facebook

Pig and Hive

- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Developed by Yahoo!, now open source
  - By 2009, roughly 40% of all Yahoo! internal Hadoop jobs
- Hive: data warehousing application in Hadoop
  - Query language is HQL, variant of SQL
  - Tables stored on HDFS as flat files
  - Now Apache open source
- Common idea:
  - Provide higher-level language to facilitate large-data processing
  - Higher-level language “compiles down” to Hadoop jobs

Why Pig ?

Because we bet you can read the following script:

– A Real Pig Script in Production:

top_5.pig

```
users = load 'users.csv' as (username: chararray, age: int);
users_1825 = filter users by age >= 18 and age <= 25;

pages = load 'pages.csv' as (username: chararray, url: chararray);

joined = join users_1825 by username, pages by username;
grouped = group joined by url;
summed = foreach grouped generate group as url, COUNT(joined) AS views;
sorted = order summed by views desc;
top_5 = limit sorted 5;

store top_5 into 'top_5_sites.csv';
```

Same Calculation in Hadoop/MapReduce would look like …

Why Pig ? (cont’d)

Recap: Map-Reduce Join Patterns

- **Compute the natural join $R(A,B) \bowtie S(B,C)$**
- R and S are each stored in files
- Tuples are pairs (a,b) or (b,c)

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R

$\bowtie$

B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

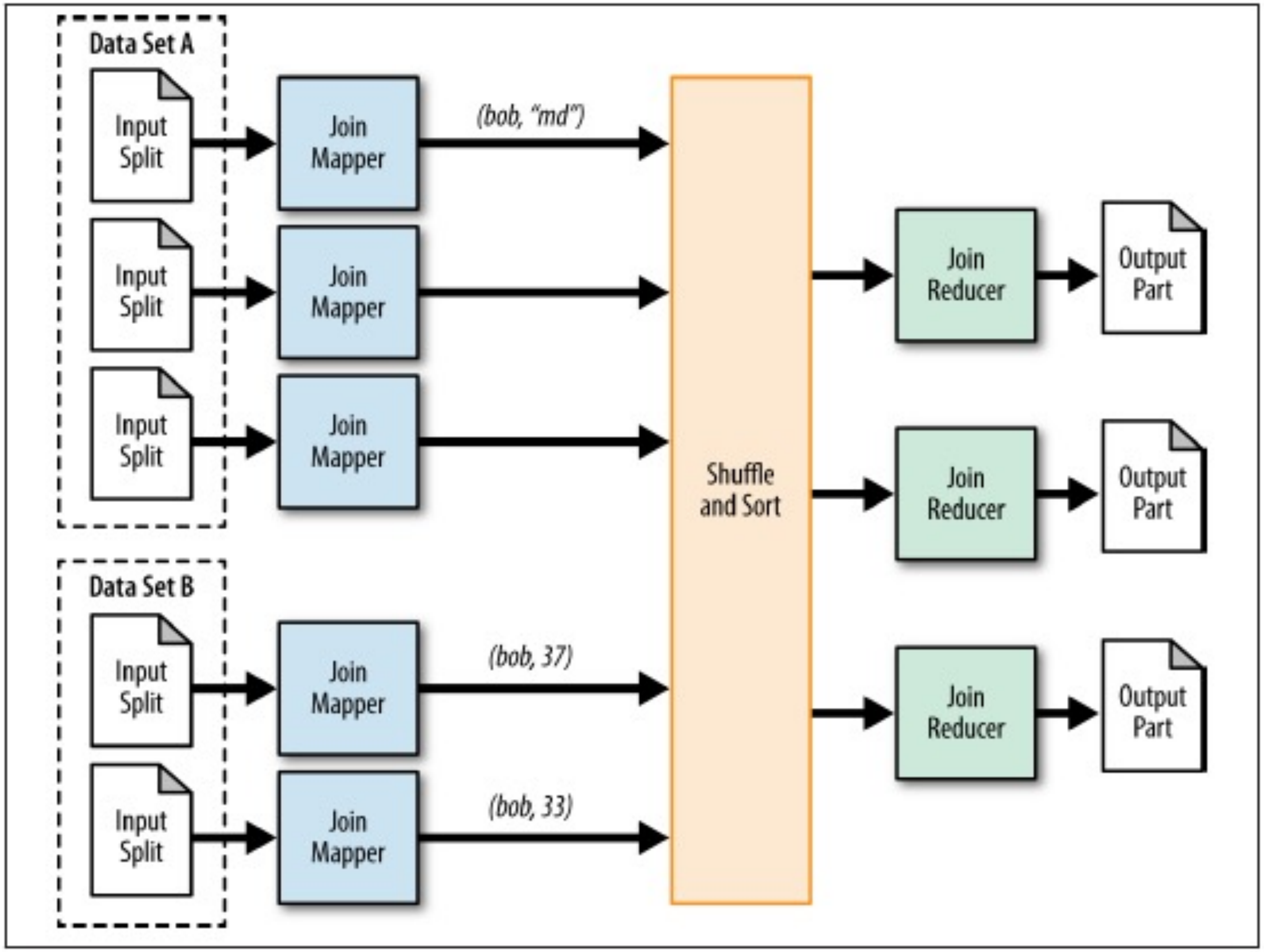
=

A	C
a_3	c_1
a_3	c_2
a_4	c_3

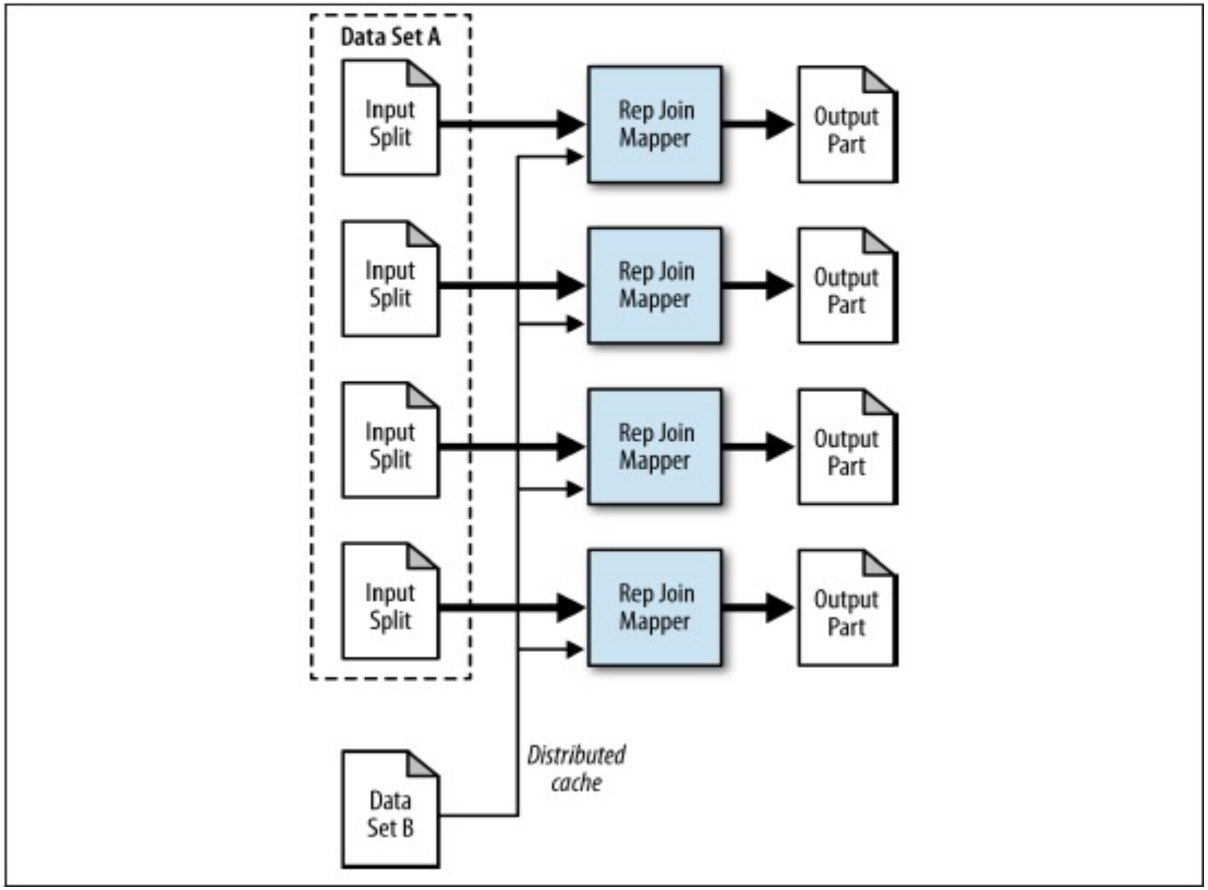
Re-partition Join

- **Use a hash function *h* from B-values to *1...k***
- **A Map process turns:**
  - Each input tuple *R(a,b)* into key-value pair *(b,(a,R))*
  - Each input tuple *S(b,c)* into *(b,(c,S))*
- **Map processes** send each key-value pair with key *b* to Reduce process *h(b)*
  - Hadoop does this automatically; just tell it what *k* is.
- Each **Reduce process** matches all the pairs *(b,(a,R))* with all *(b,(c,S))* and outputs *(a,b,c)*.

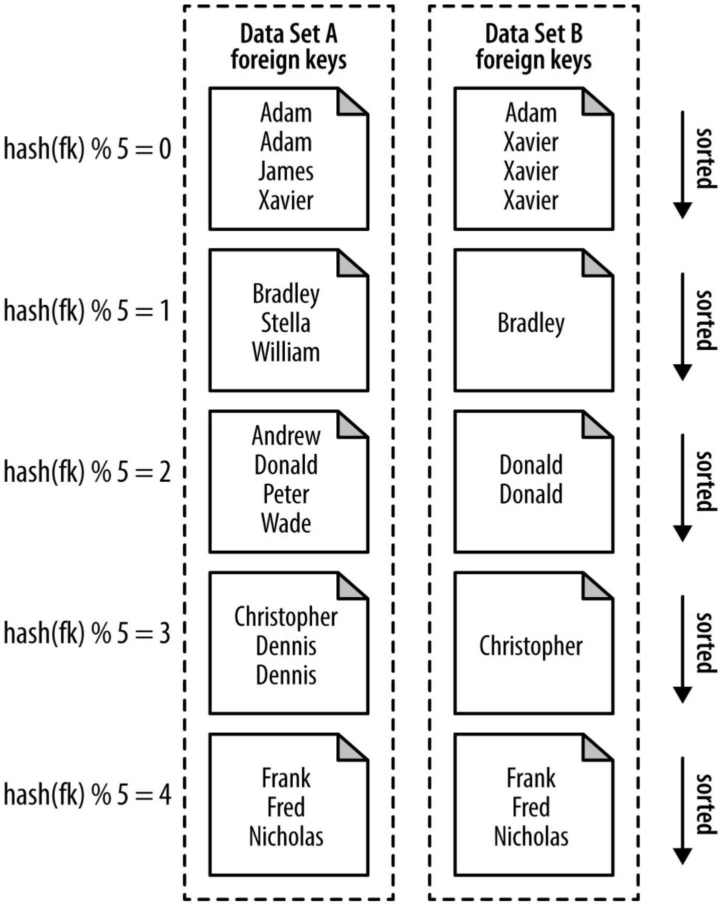
Re-partition Join

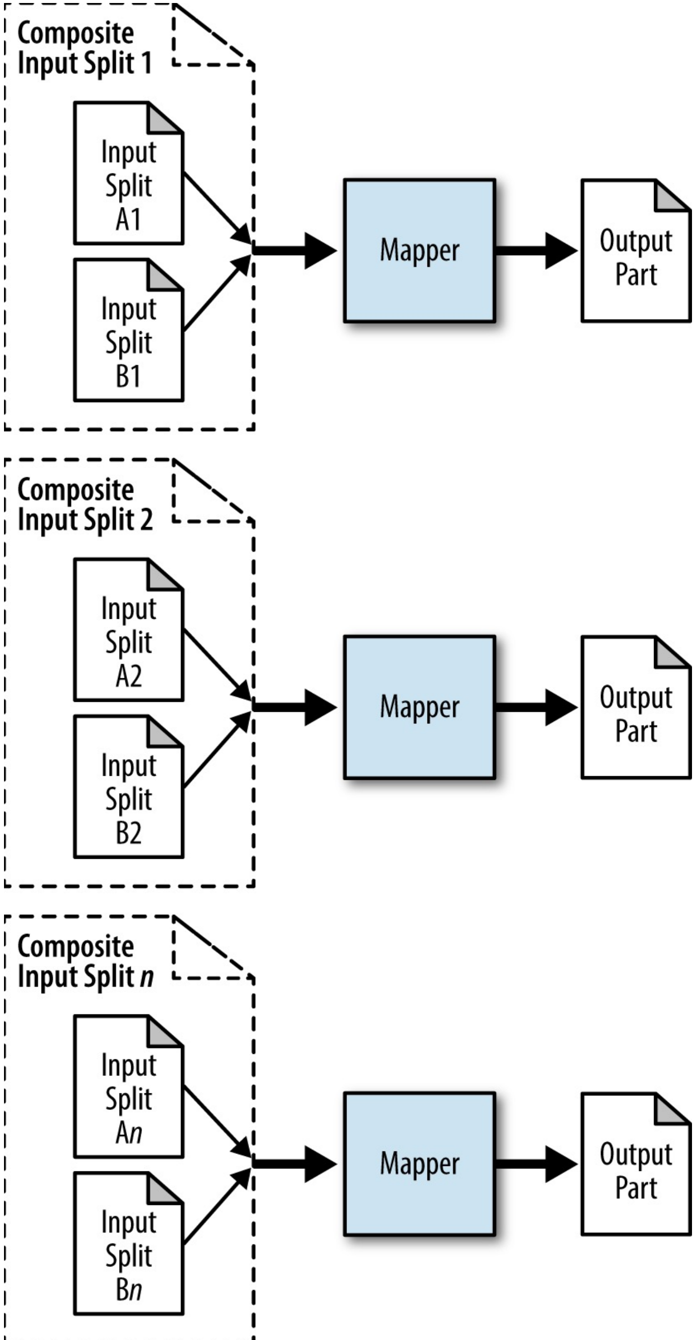


Replicated Join



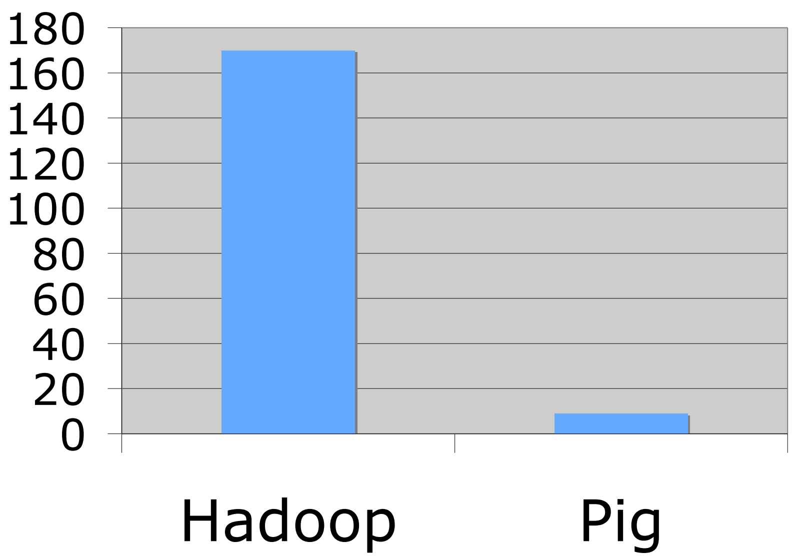
Composite Join



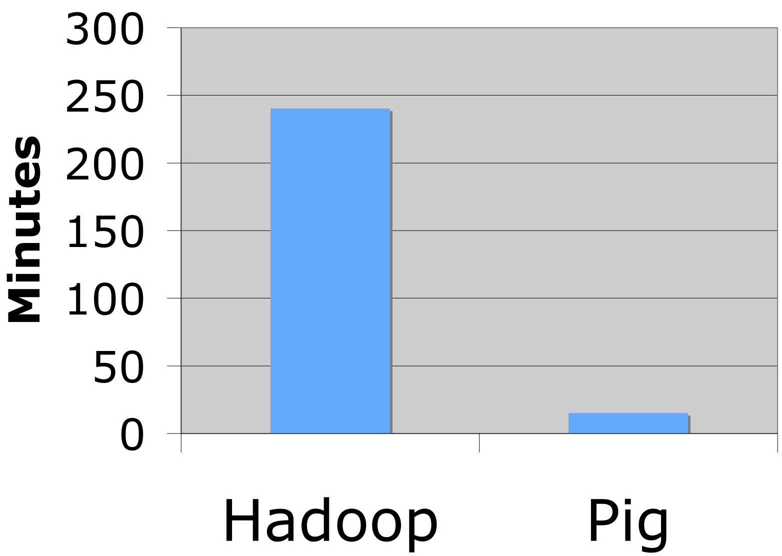


Why Pig ? (cont’d)
Faster Code Development

1/20 the lines of code



1/16 the development time

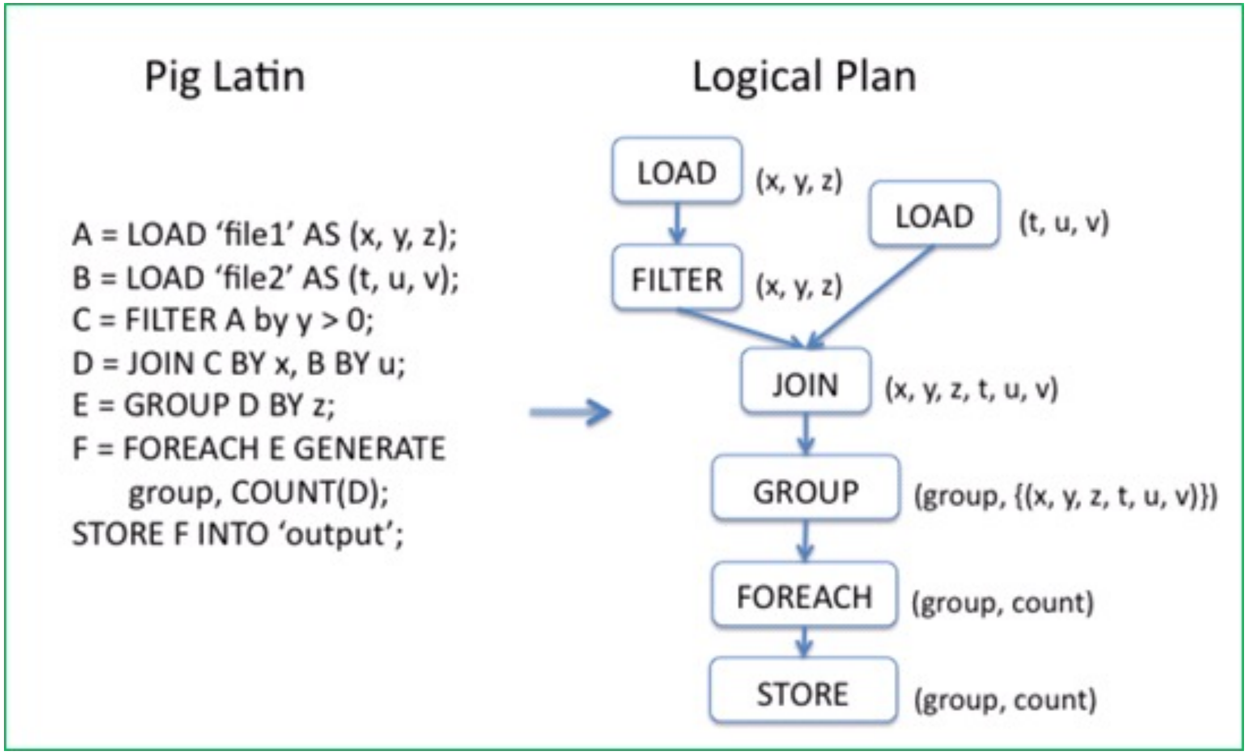


Performance on par with (maybe ~2 times slower than) raw Hadoop!

Pig Slides adapted from Olston et al.

What is Pig ?

- Framework for analyzing large un-structured and semi-structured data on top of Hadoop.
  - Pig Engine Parses, compiles Pig Latin scripts into MapReduce jobs run on top of Hadoop.
  - Pig Latin is a dataflow language
  - Pig is the high level language interface for Hadoop



Use Cases for Pig

- Ad hoc analysis of unstructured data
  - Web Crawls, Log files, Click streams
- Pig is an excellent ETL tool
  - "Extract, Transform, Load" for preprocessing data before loading them to a Data Warehouse
- Rapid Prototyping for Analytics
  - Let one to experiment with large data sets before writing customized applications

Example Data Analysis Task

Find users who tend to visit “good” pages.

Visits

user	url	time
Amy	www.cnn.com	8:00
Amy	www.crap.com	8:05
Amy	www.myblog.com	10:00
Amy	www.flickr.com	10:05
Fred	cnn.com/index.htm	12:00

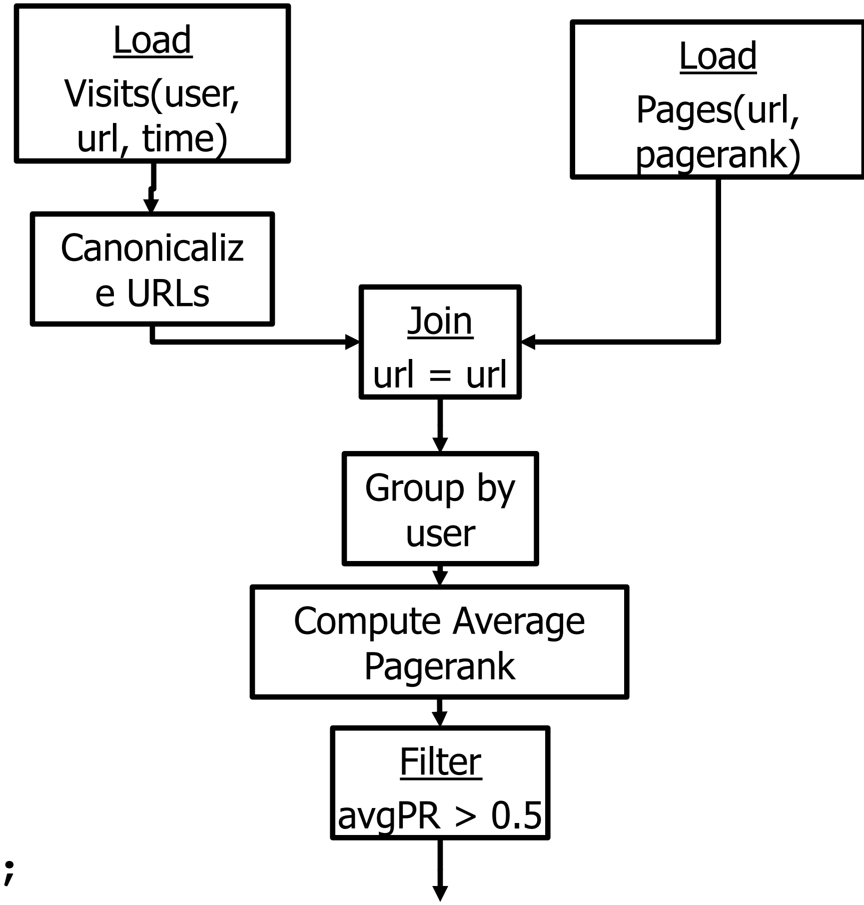
⋮

Pages

url	pagerank
www.cnn.com	0.9
www.flickr.com	0.9
www.myblog.com	0.7
www.crap.com	0.2

Pig Latin Script

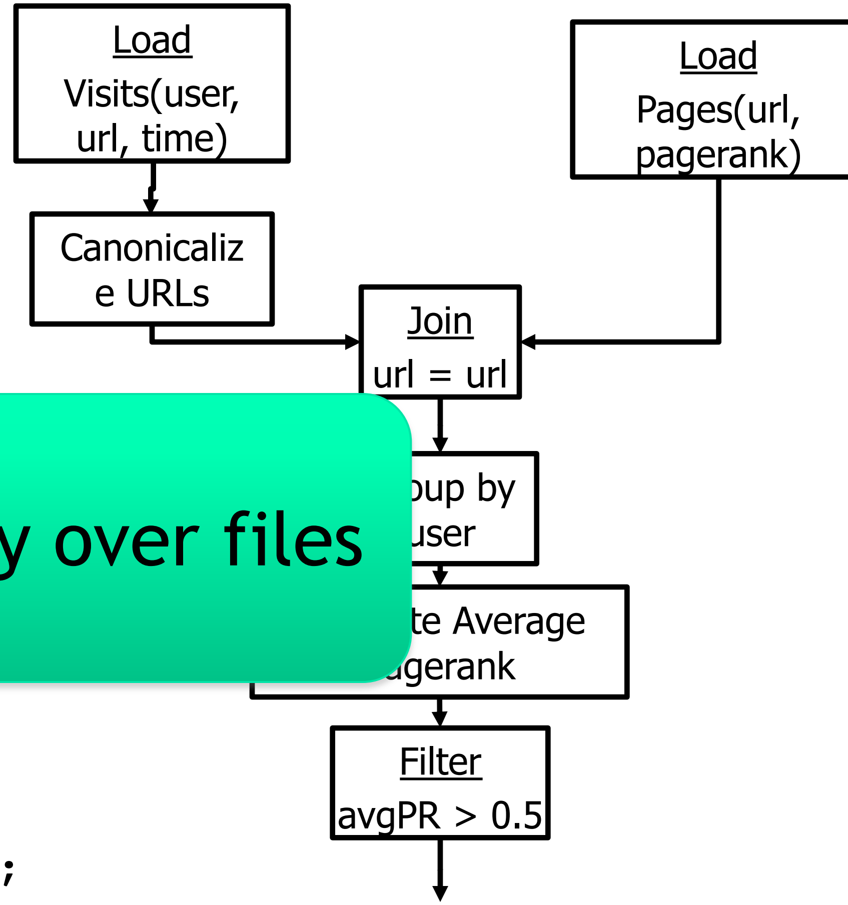
```
Visits = load '/data/visits' as (user,
url, time);
Visits = foreach Visits generate user,
Canonicalize(url), time;

Pages = load '/data/pages' as (url,
pagerank);

VP = join Visits by url, Pages by url;
UserVisits = group VP by user;
UserPageranks = foreach UserVisits
generate user, AVG(VP.pagerank) as
avgpr;
GoodUsers = filter  UserPageranks by
avgpr > '0.5';

Store GoodUsers into '/data/good_users';
```



Pig Latin Script

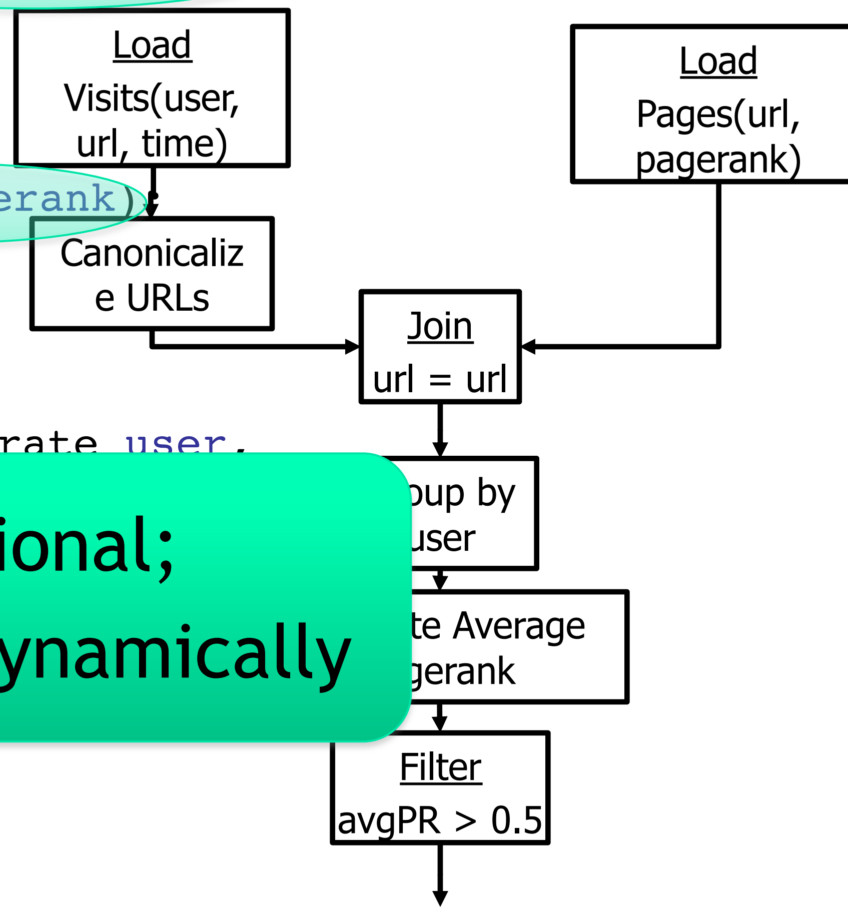
```
Visits = load '/data/visits' as (user,
url, time);
Visits = foreach Visits generate user,
Canonicalize(url), time;

Pages = load '/data/pages' as (url,
pagerank);

VP = join Visits
UserVisits = gro
UserPageranks =
generate user, A
avgpr;
GoodUsers = filter  UserPageranks by
avgpr > '0.5';

Store GoodUsers into '/data/good_users';
```

Operates directly over files



Pig Latin Script

```
Visits = load '/data/visits' as (user, url, time);
Visits = foreach Visits generate user,
Canonicalize(url), time;

Pages = load '/data/pages' as (url, pagerank);

VP = join Visits by url, Pages by url;
UserVisits = group VP by user;
UserPageranks = foreach UserVisits generate user,
AVG(VP.pageran
GoodUsers = fi

Store GoodUse
```



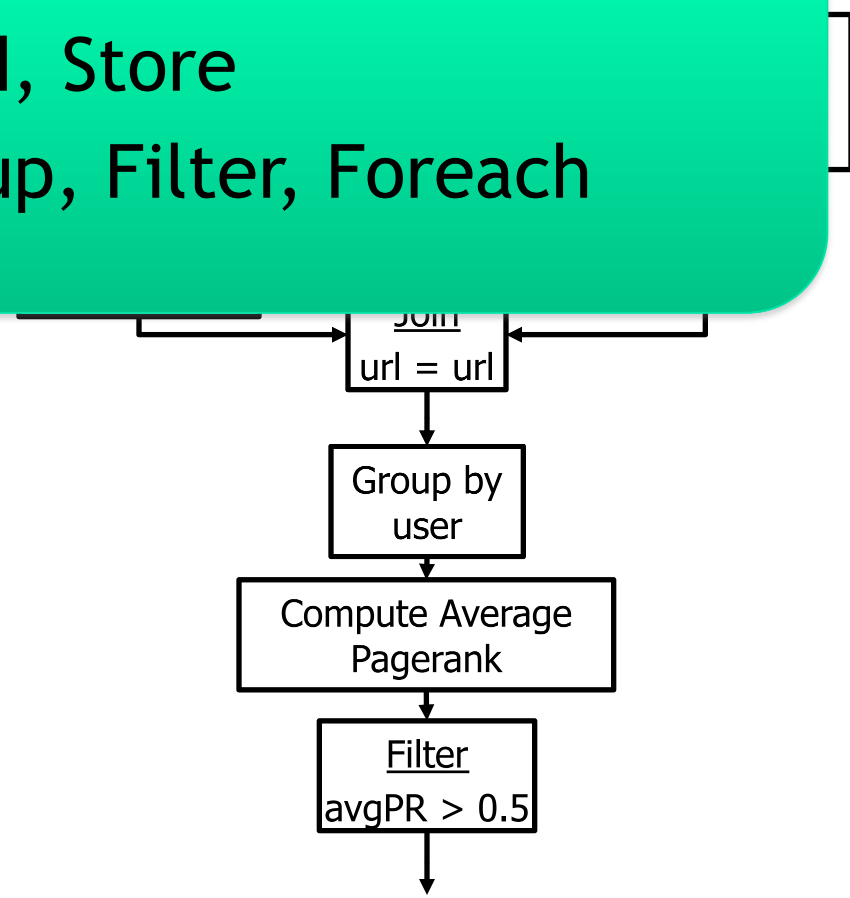
Schemas optional;
Can be assigned dynamically

Pig Latin Script

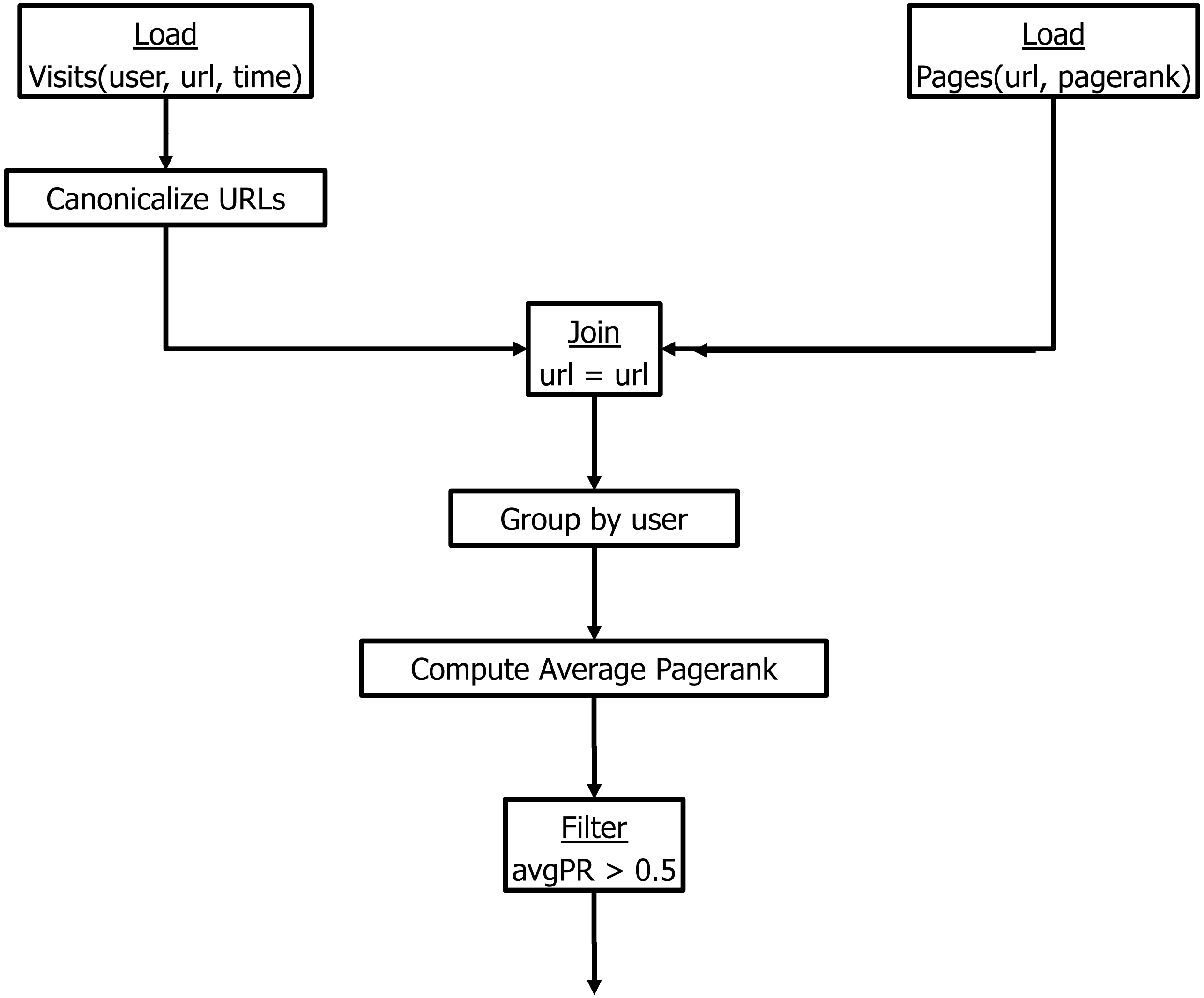
User-defined functions (UDFs) can be used in every construct

- Load, Store
- Group, Filter, Foreach

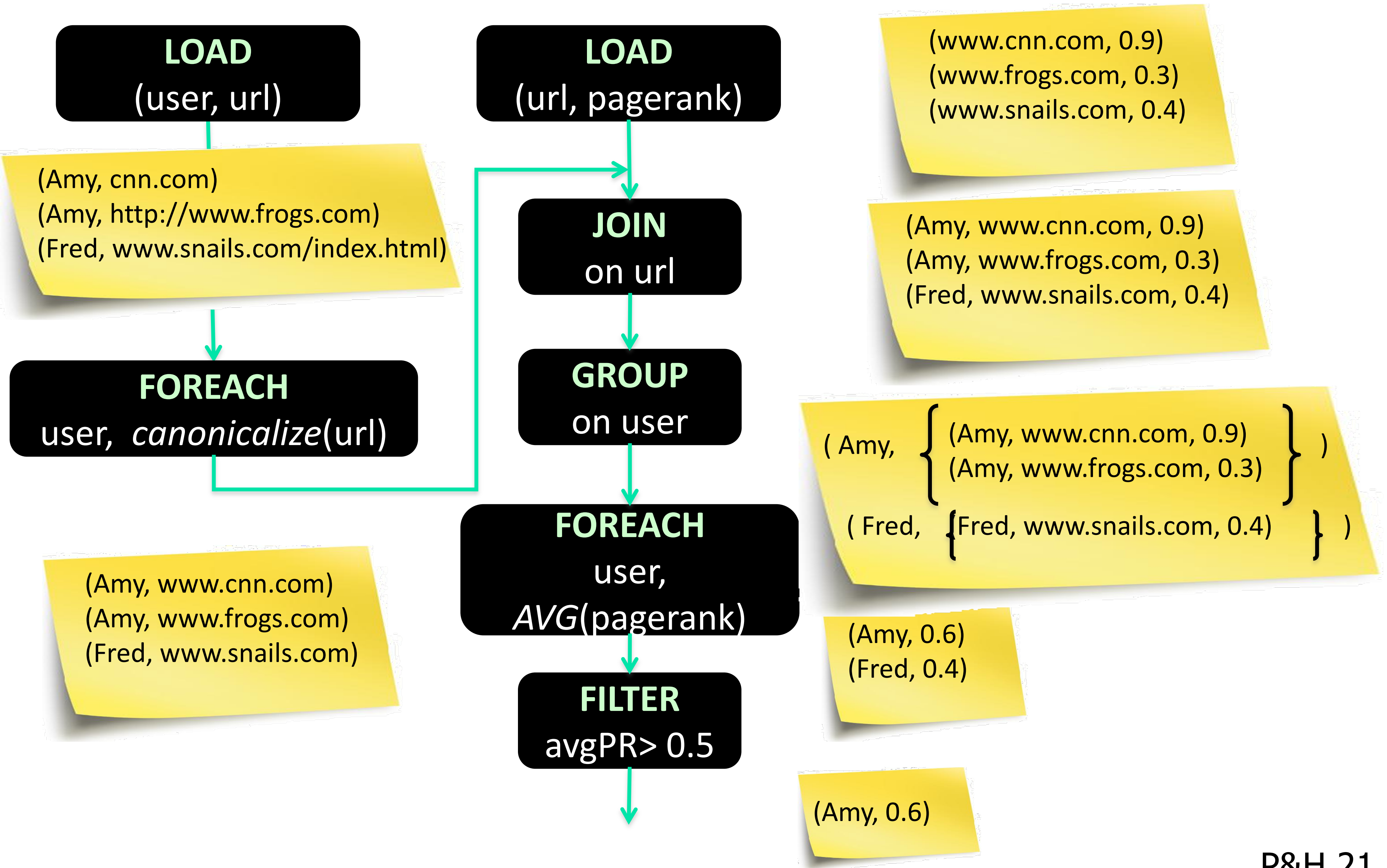
```
Visits = load '/data/visits'
url, time);
Visits = foreach Visits gen
Canonicalize(url), time;

Pages = load '/data/pages' a
pagerank);

VP = join Visits by url, Pages by url;
UserVisits = group VP by user;
UserPageranks = foreach UserVisits
generate user, AVG(VP.pagerank) as
avgpr;
GoodUsers = filter  UserPageranks by
avgpr > '0.5';

Store GoodUsers into '/data/good_users';
```



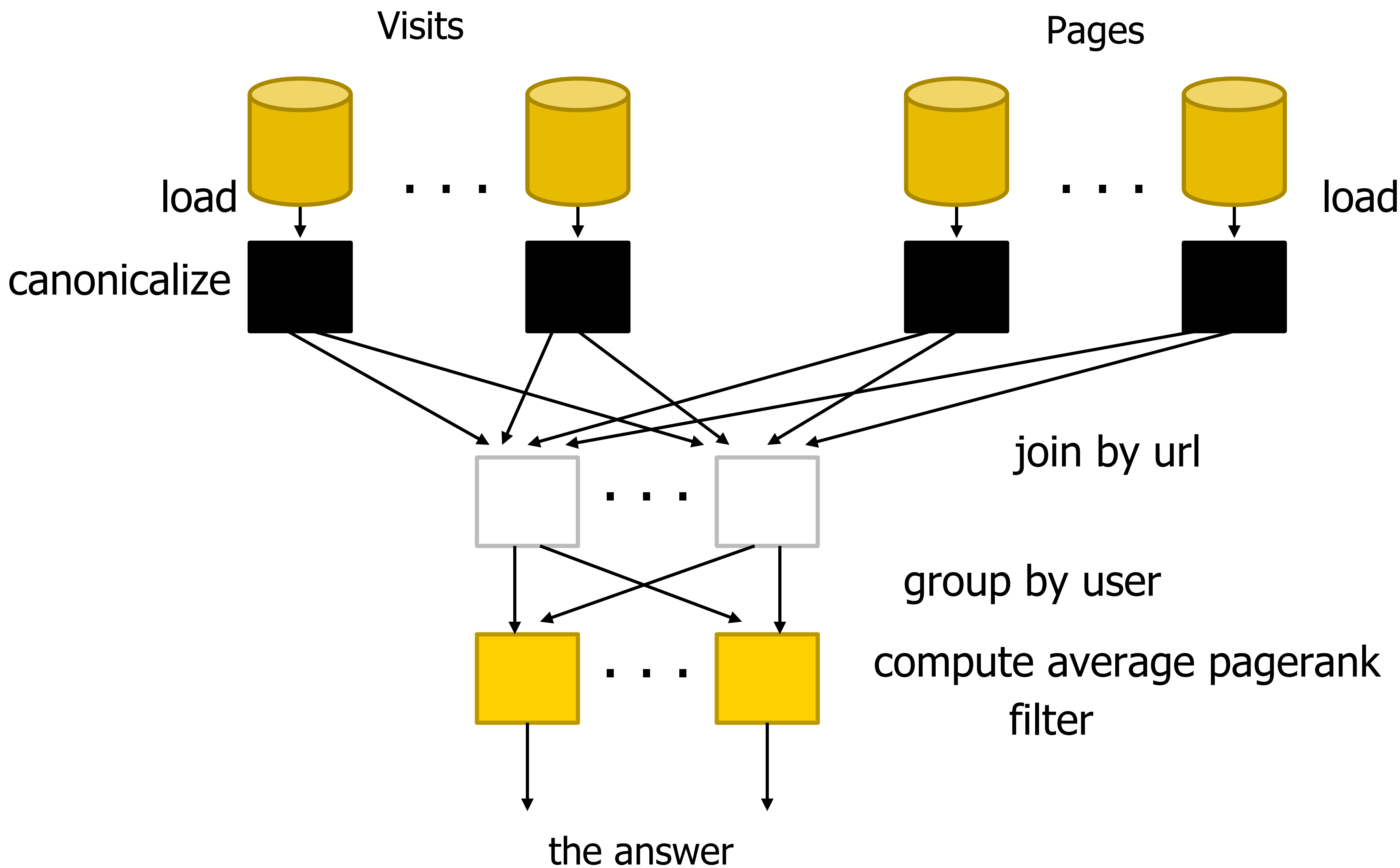
Conceptual Dataflow



Example to Illustrate (a slightly different) Program



System-Level Dataflow



Pig Slides adapted from Olston et al.

MapReduce Code

```
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.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 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.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();
    }
}
```

2nd Pig Latin Example

```
visits           = load '/data/visits' as (user, url, time);
gVisits          = group visits by url;
urlCounts  = foreach gVisits generate url, count(visits);

urlInfo          = load '/data/urlInfo' as (url, category, pRank);
urlCategoryCount = join urlCounts by url, urlInfo by url;

gCategories = group urlCategoryCount by category;
topUrls = foreach gCategories generate top(urlCounts,10);

store topUrls into '/data/topUrls';
```

Pig Latin Execution

```
visits          = load '/data/visits' as (user, url, time);
gVisits         = group visits by url;
urlCounts = foreach gVisits generate url, count(visits);
```

MR Job 1

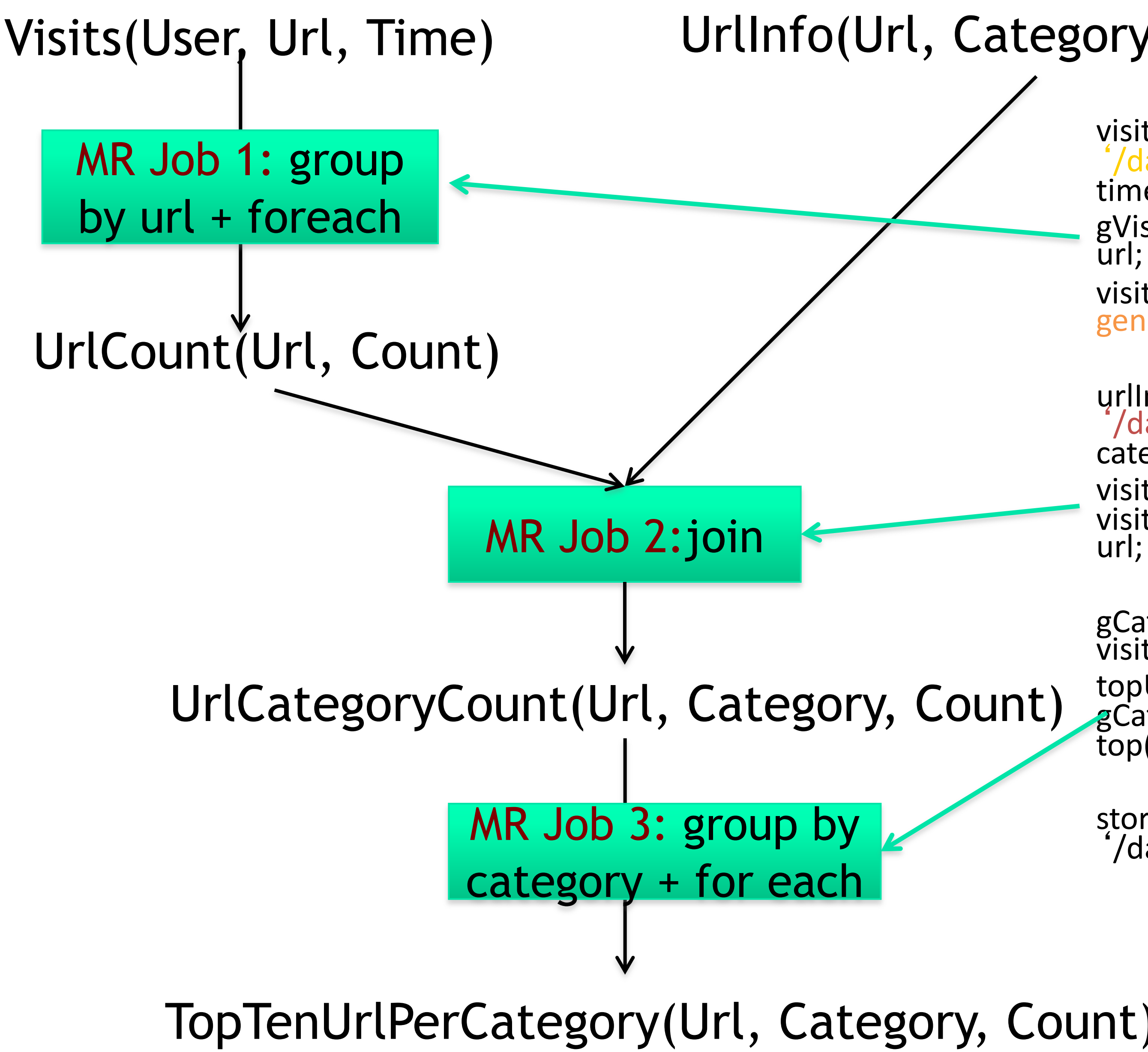
```
urlInfo         = load '/data/urlInfo' as (url, category, pRank);
urlCategoryCount = join urlCounts by url, urlInfo
```

MR Job 2

```
gCategories = group urlCategoryCount by category;
topUrls = foreach gCategories generate top(urlCounts,10);

store topUrls into '/data/topUrls';
```

MR Job 3

Pig Latin: Execution




```
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' ;
```

Pig Latin: Language Features

- **Keywords**
  - Load, Filter, Foreach Generate, Group By, Store, Join, Distinct, Order By, …
- **Aggregations**
  - Count, Avg, Sum, Max, Min
- **Schema**
  - Defines at query-time not when files are loaded
- **User Defined Functions** (UDFs)
  - As first-class citizens in the language
  - UDFs can be written in other languages, e.g. Java, Python, Javascript, etc
- Packages for common input/output formats

An example w/ more details

Read file from HDFS

The input format (text, tab delimited)

Define run-time schema

```
raw = LOAD 'excite.log' USING PigStorage('\t') AS (user, id, time, query);

clean1 = FILTER raw BY id > 20 AND id < 100;

clean2 = FOREACH clean1 GENERATE
            user, time,
            org.apache.pig.tutorial.sanitze(query) as query;

user_groups = GROUP clean2 BY (user, query);

user_query_counts = FOREACH user_groups
      GENERATE group, COUNT(clean2), MIN(clean2.time), MAX(clean2.time);

STORE user_query_counts INTO 'uq_counts.csv' USING PigStorage(',');
```

Filter the rows on predicates

For each row, do some transformation

Grouping of records

Compute aggregation for each group

Store the output in a file

Text, Comma delimited

NOTE: The records coming out of a GROUP BY statement have two fields, the key and the bag of collected records. The key field is named "group" § The bag is named for the alias that was grouped, so in this example, it will be named clean2 and have the same schema as the relation clean2.

§ Thus the keyword "group" is overloaded in Pig Latin. This is unfortunate and confusing, but also hard to change now.

Pig Latin: Data Types

- Data types
  - *Atom*: Simple atomic value
  - *Tuple*: A tuple is a sequence of fields, each can be any of the data types
  - *Bag*: A bag is a collection of tuples
  - *Map*: A collection of data items that is associated with a dedicated atom

`alice'	(`alice'}, \text{`lakers')	$\left\{\begin{array}{c} (\text{`alice'}, \text{`lakers'}) \\ (\text{`alice'}, (\text{`iPod'}, \text{`apple'})) \end{array}\right\}$	$\left[\begin{array}{c} \text{`fan of'} \to \left\{\begin{array}{c} \text{`lakers'} \\ \text{`iPod'} \end{array}\right\} \\ \text{`age'} \to 20 \end{array}\right]$
Atom	Tuple	Bag	Map

Pig Latin: Expressions

$$t = \left(\overset{\text{f1}}{\text{'alice'}}, \overset{\text{f2}}{\left\{ \begin{array}{c} (\text{'lakers'}, 1) \\ (\text{'iPod'}, 2) \end{array} \right\}}, \overset{\text{f3}}{[\text{'age'} \rightarrow 20]} \right)$$

Expression Type	Example	Value for tuple *t*
Constant	`'bob'`	Independent of t
Field by position	`$0`	'alice'
Field by name	`f3`	$[\text{'age'} \rightarrow 20]$
Projection	f2.$0	$\left\{ \begin{array}{c} (\text{'lakers'}) \\ (\text{'iPod'}) \end{array} \right\}$
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

Another example w/ more details

Script can take arguments

Data are “ctrl-A” delimited

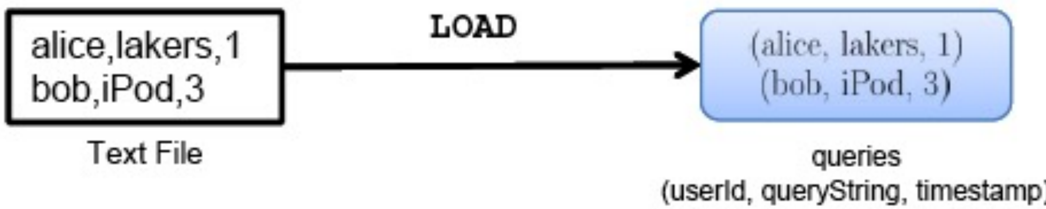
Define types of the columns

```
A = load '$widerow' using PigStorage('\u0001') as (name: chararray, c0: int, c1: int,
c2: int);
B = group A by name parallel 10;
C = foreach B generate group, SUM(A.c0) as c0, SUM(A.c1) as c1, AVG(A.c2) as c2;
D = filter C by c0 > 100 and c1 > 100 and c2 > 100;
store D into '$out';
```

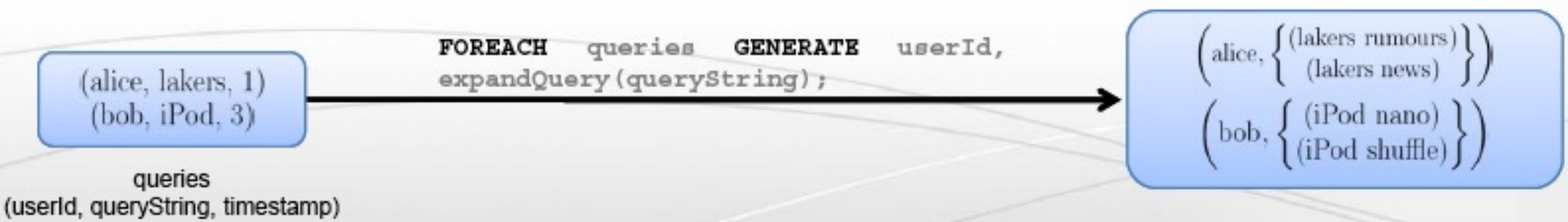
Specify the need of 10 reduce tasks

Pig Latin: Commands and Operators (1)

- `LOAD` — Specify input data
  - `queries = LOAD 'query_log.txt' USING myLoad() AS (userId, querystring, timestamp);`
    - `myLoad()` is a user defined function (UDF)



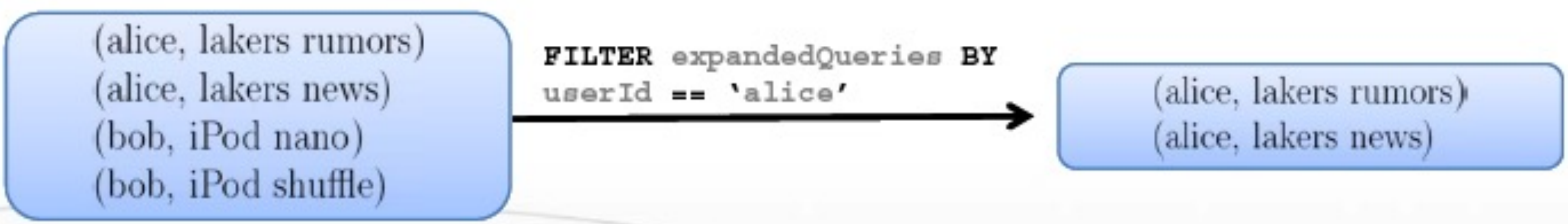
- `FOREACH` — Per-tuple processing
  - `expanded_queries = FOREACH queries GENERATE userId, expandQuery(queryString);`



Pig Latin: Commands and Operators (2)

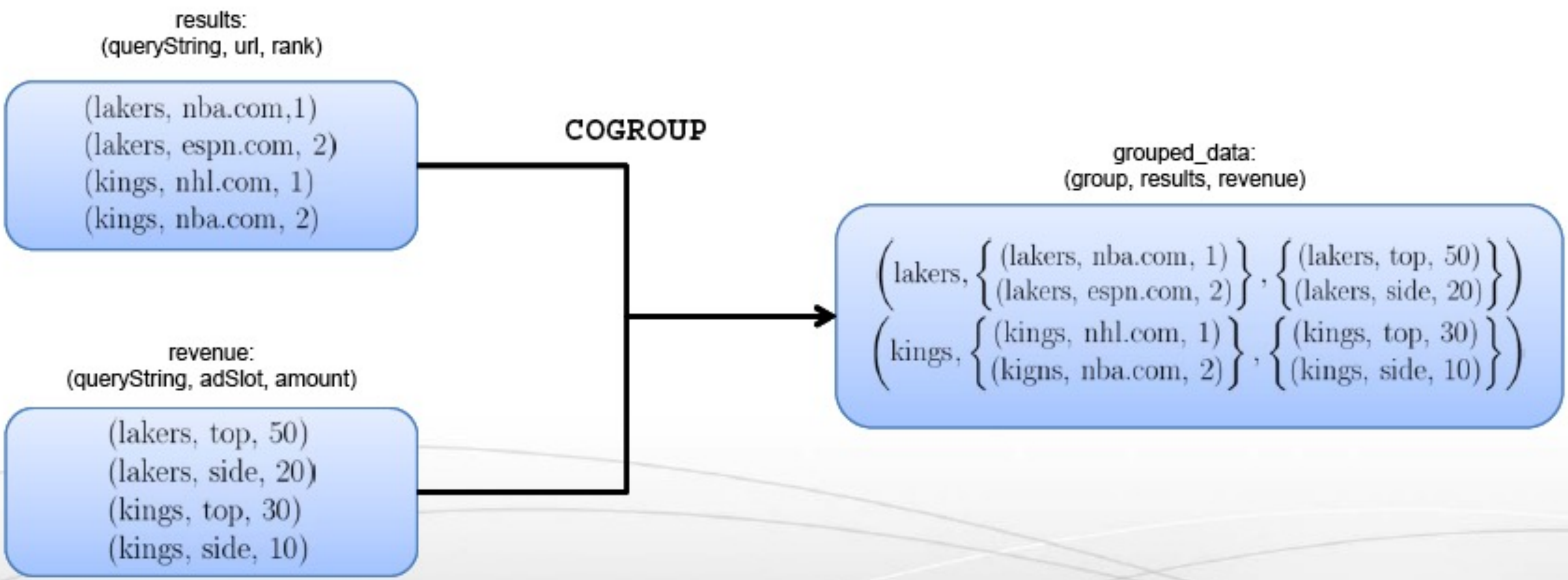
- **`FLATTEN`** – Remove nested data in tuples

- **`FILTER`** – Discarding unwanted data



Pig Latin: Commands and Operators (3)

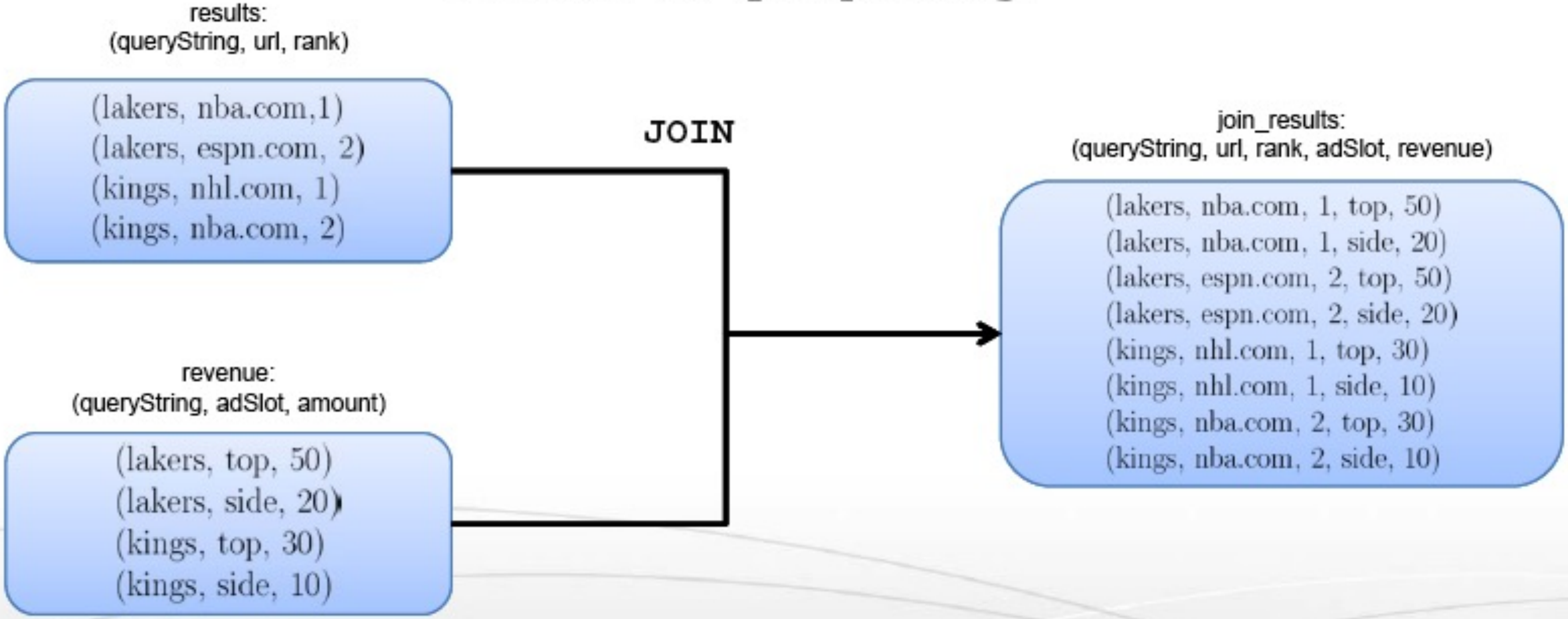
- **COGROUP** – Getting related data together
  - `grouped_data = COGROUP results BY queryString, revenue BY queryString;`



GROUP is a special case of **COGROUP**

Pig Latin: Commands and Operators (4)

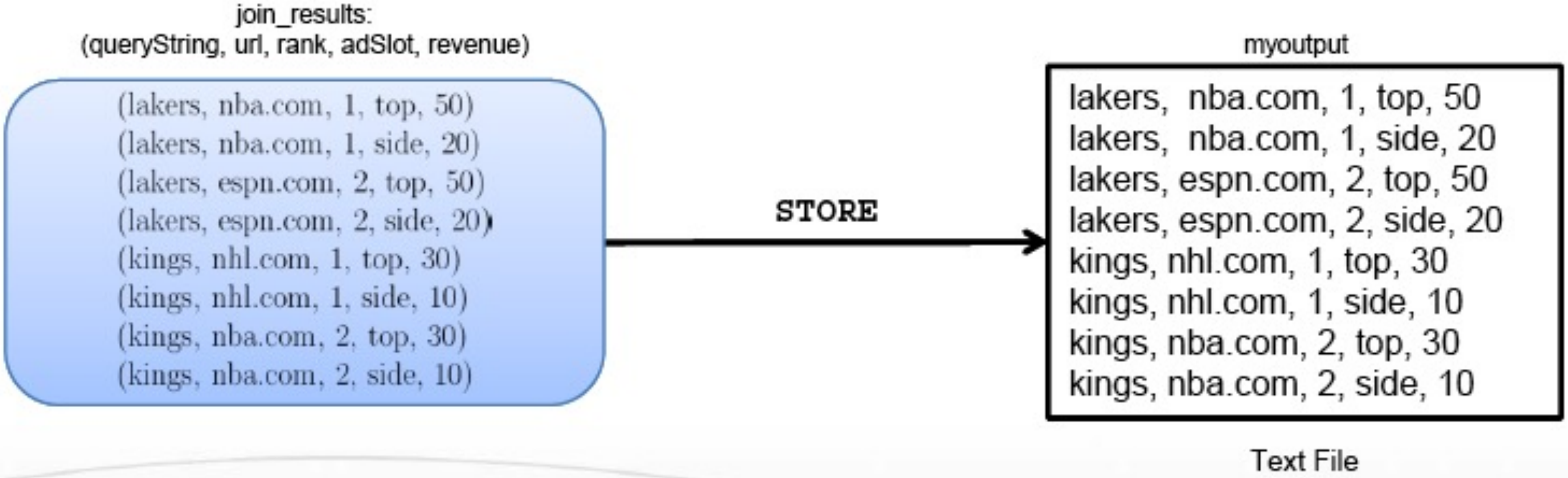
- **JOIN** – Cross product of two tables
  - `join_result = JOIN results BY queryString, revenue BY queryString;`



JOIN is the same as **COGROUP** + **FLATTEN**

Pig Latin: Commands and Operators (5)

- `STORE` – Create output
  - `final_result = STORE join_results INTO 'myoutput', USING myStore();`



Pig Latin: Commands and Operators (6)

- 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

Example 3: Re-partition Join

Register UDFs & custom inputformats

Function the jar file to read the input file

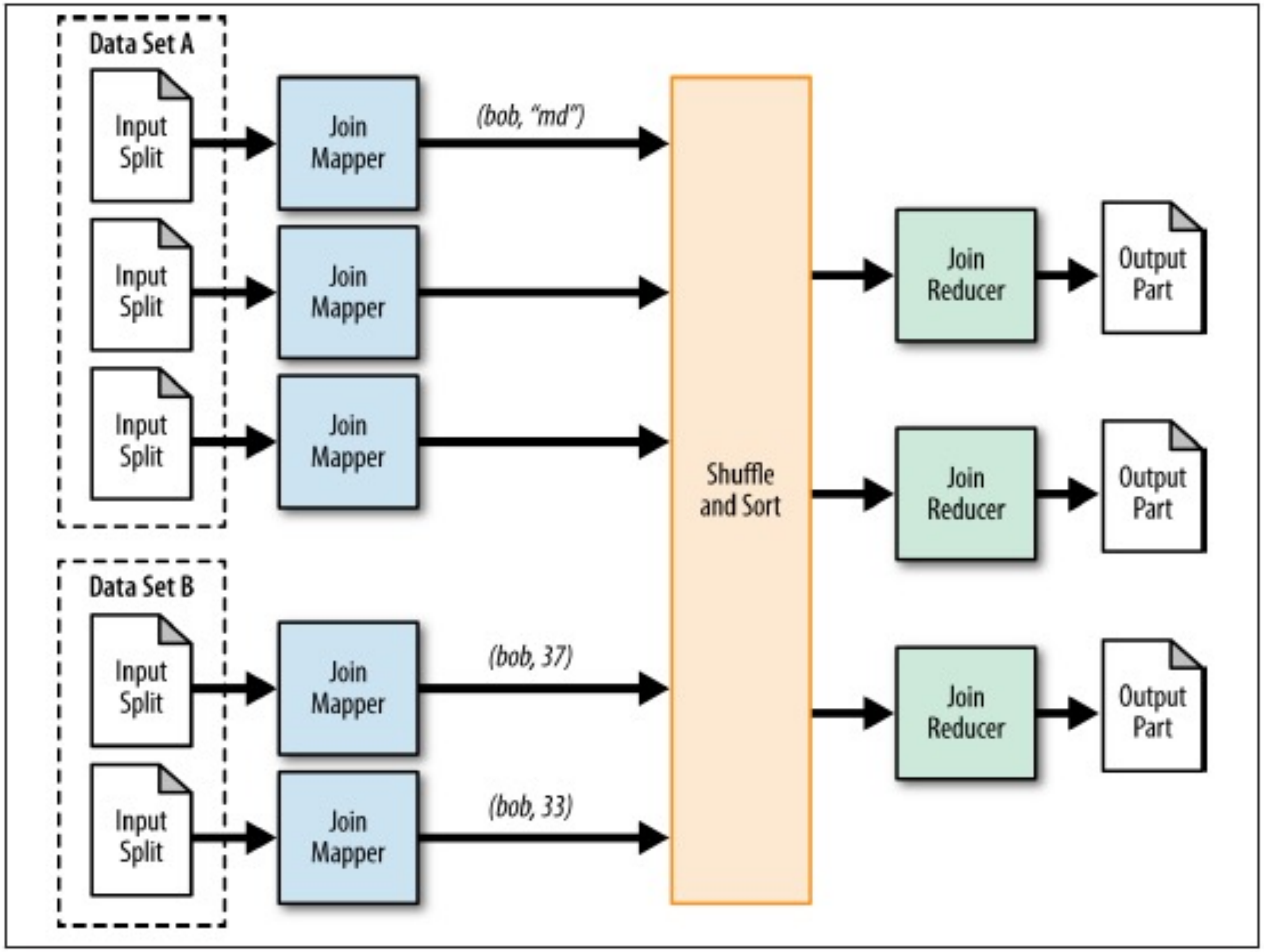
Load the second file

Join the two datasets (40 reducers)

Group after the join (can reference columns by position)

```
register pigperf.jar;

A = load ‘page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
        as (user, action, timespent, query_term, timestamp, estimated_revenue);

B = foreach A generate user, (double) estimated_revenue;

alpha = load ’users' using PigStorage('\u0001') as (name, phone, address, city, state, zip);

beta = foreach alpha generate name, city;

C = join beta by name, B by user parallel 40;

D = group C by $0;

E = foreach D generate group, SUM(C.estimated_revenue);
store E into 'L3out';
```

Example 3: Re-partition Join



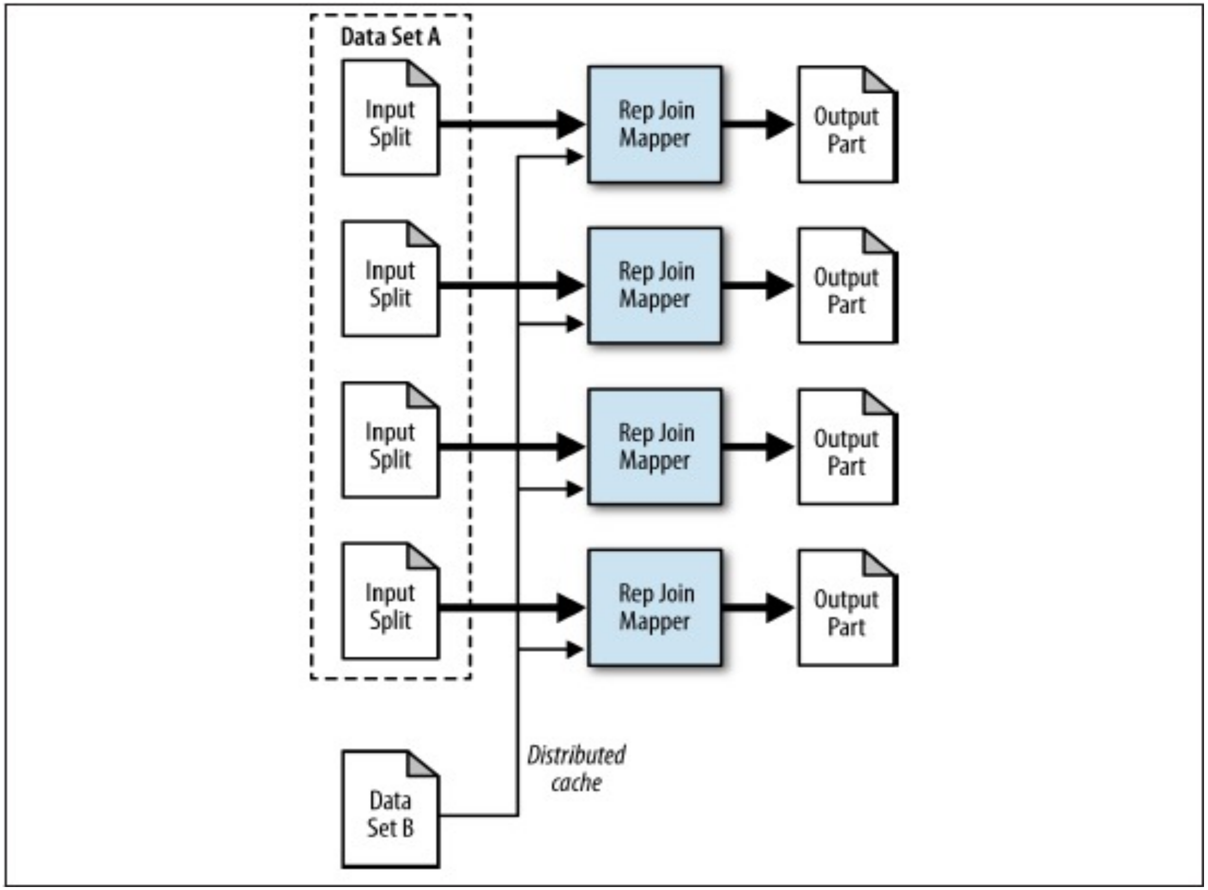
Example 4: Replicated Join

```
register pigperf.jar;
A = load 'page_views' using
org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
        as (user, action, timespent, query_term, timestamp,
estimated_revenue);
Big = foreach A generate user, (double) estimated_revenue;
alpha = load 'users' using PigStorage('\u0001') as (name, phone,
address, city, state, zip);
small = foreach alpha generate name, city;
C = join Big by user, small by name using 'replicated';
store C into 'out';
```

Map-only join (the small dataset is the second)

Optimization in joining a big dataset with a small one

Example 4: Replicated Join



Example 5: Multiple Outputs

```
A = LOAD 'data' AS (f1:int,f2:int,f3:int);

DUMP A;
(1,2,3)
(4,5,6)
(7,8,9)

SPLIT A INTO X IF f1<7, Y IF f2==5, Z IF (f3<6 OR f3>6);

DUMP X;
(1,2,3)
(4,5,6)

DUMP Y;
(4,5,6)

STORE x INTO 'x_out';
STORE y INTO 'y_out';
STORE z INTO 'z_out';
```

Split the records into sets

Dump command to display the data

Store multiple outputs

Run independent jobs in parallel

```
D1 = load 'data1' …

D2 = load 'data2' …

D3 = load 'data3' …

C1 = join D1 by a, D2 by b

C2 = join D1 by c, D3 by d
```

C1 and C2 are two independent jobs that can run in parallel

What is UDF in Pig ?

- UDF - User Defined Function
- Types of UDF' s:
  - Eval Functions (extends EvalFunc<String>)
  - Aggregate Functions (extends EvalFunc<Long> implements Algebraic)
  - Filter Functions (extends FilterFunc)
- UDFContext
  - Allows UDFs to get access to the JobConf object
  - Allows UDFs to pass configuration information between instantiations of the UDF on the front and backends.

Sample UDF

```
public class TopLevelDomain extends
    EvalFunc<String> {

    @Override
    public String exec(Tuple tuple) throws IOException {
        Object o = tuple.get(0);
        if (o == null) {
                return null;
        }
        return Validator.getTLD(o.toString());
    }
}
```

UDF In Action

- REGISTER '$WORK_DIR/pig-support.jar';
- DEFINE getTopLevelDomain com.contextweb.pig.udf.TopLevelDomain();
- AA = foreach input GENERATE TagId, getTopLevelDomain(PublisherDomain) as RootDomain

Pig Latin vs. SQL

- Pig Latin is procedural (dataflow programming model)
  - Step-by-step query style is much cleaner and easier to write
- SQL is declarative but not step-by-step style

SQL

```
insert into ValuableClicksPerDMA
select dma, count(*)
from geoinfo join (
                select name, ipaddr
                from users join clicks on (users.name = clicks.user)
                where value > 0;
            ) using ipaddr
group by dma;
```

Pig Latin

```
Users                = load 'users' as (name, age, ipaddr);
Clicks               = load 'clicks' as (user, url, value);
ValuableClicks       = filter Clicks by value > 0;
UserClicks           = join Users by name, ValuableClicks by user;
Geoinfo              = load 'geoinfo' as (ipaddr, dma);
UserGeo              = join UserClicks by ipaddr, Geoinfo by ipaddr;
ByDMA                = group UserGeo by dma;
ValuableClicksPerDMA = foreach ByDMA generate group, COUNT(UserGeo);
store ValuableClicksPerDMA into 'ValuableClicksPerDMA';
```

Pig Latin vs. SQL

- **In Pig Latin**
  - Lazy evaluation (data not processed prior to STORE command)
  - Data can be stored at any point during the pipeline
  - Schema and data types are lazily defined at run-time
  - An execution plan can be explicitly defined
    - Use optimizer hints
    - Due to the lack of complex optimizers
- **In SQL:**
  - Query plans are solely decided by the system
  - Data cannot be stored in the middle
  - Schema and data types are defined at the creation time

Pig Components

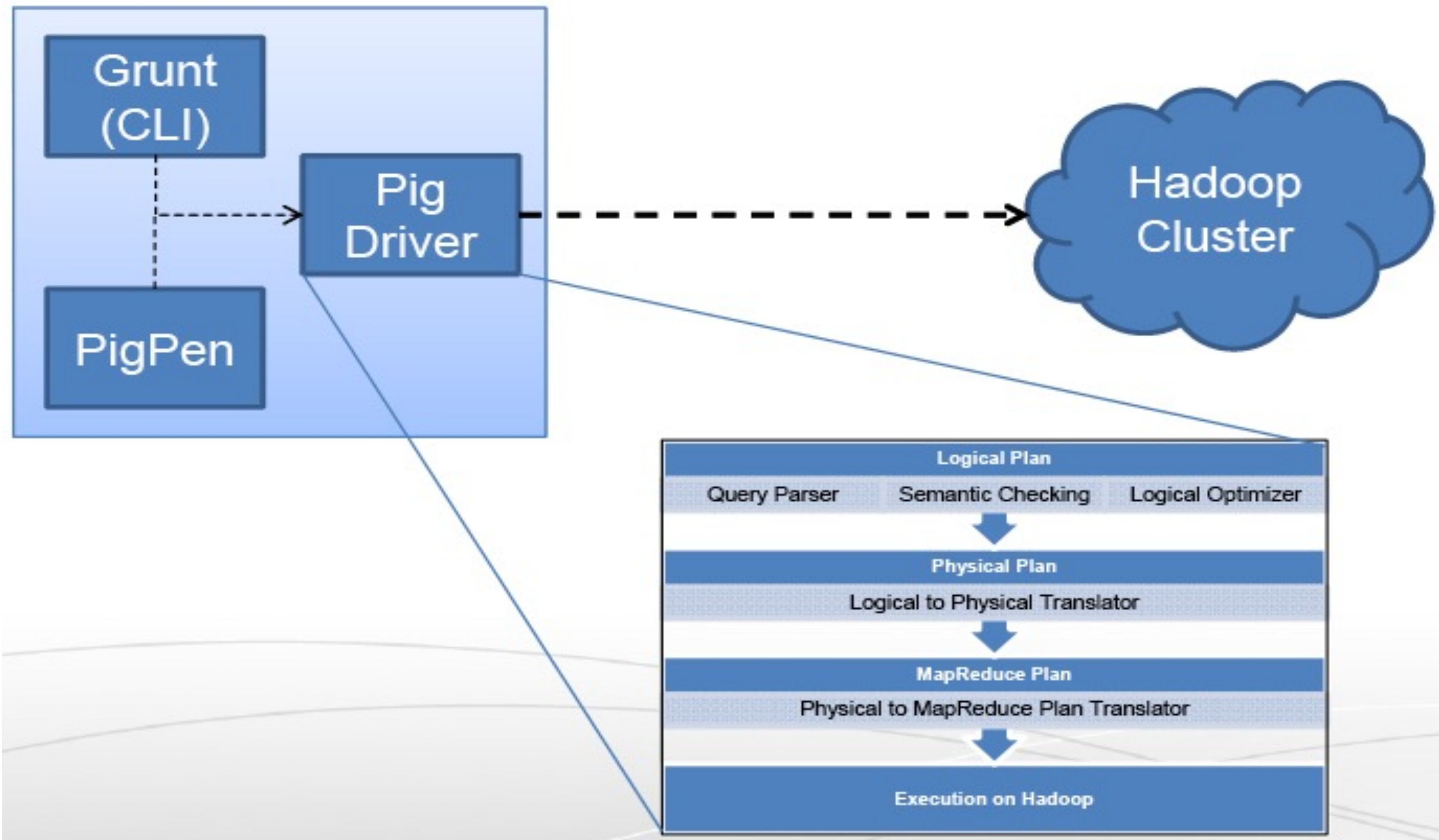
Two Main Components

- High-level language (Pig Latin)
  - Set of commands
- Two execution modes
  - Local: reads/write to local file system
  - Mapreduce: connects to Hadoop cluster and reads/writes to HDFS

Two modes

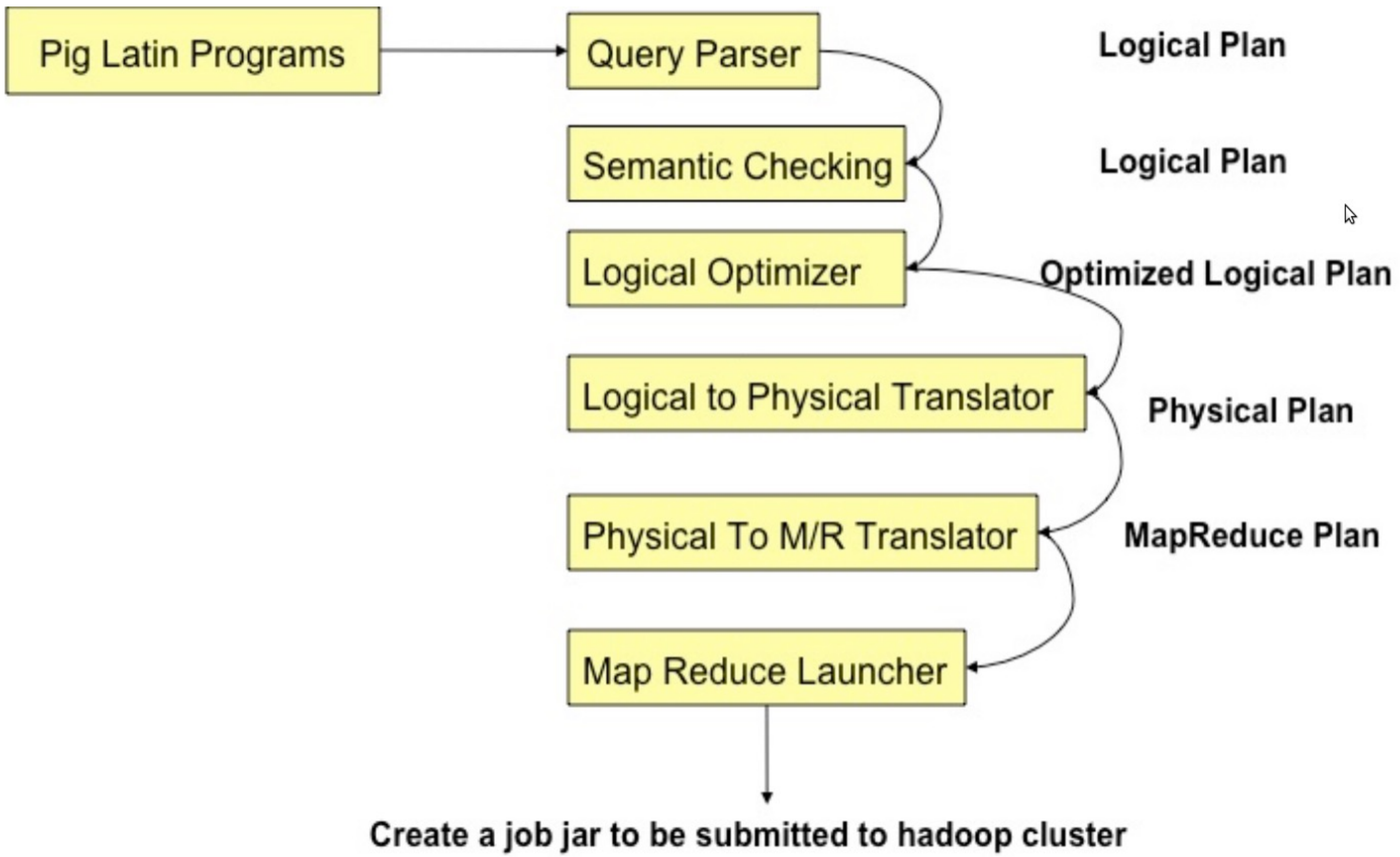
- Interactive mode
  - Console
- Batch mode
  - Submit a script

Architecture of Pig



- Grunt – A Command Line Interface to Pig
- PigPen – Debugging Environment

Pig Compilation

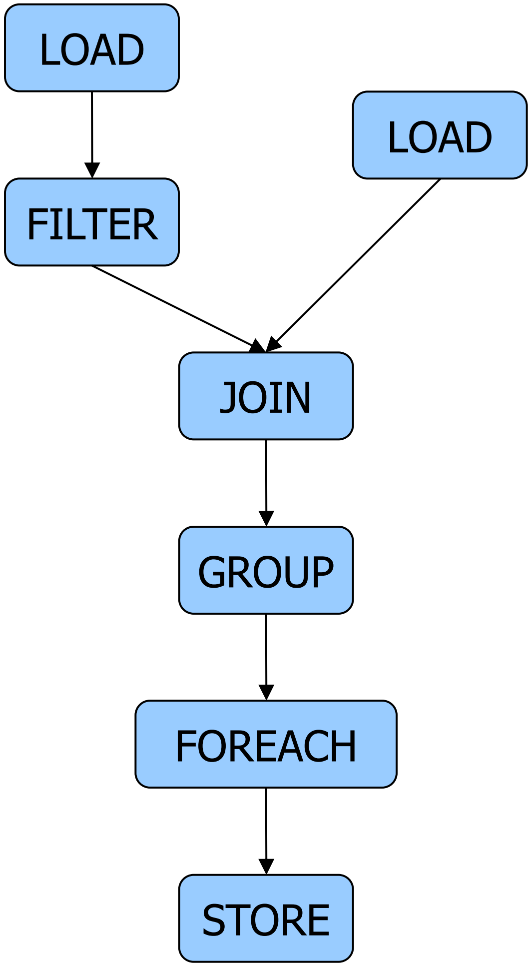


Pig takes care of…

- Schema and type checking
- Translating into efficient physical dataflow
  - (i.e., sequence of one or more MapReduce jobs)
- Exploiting data reduction opportunities
  - (e.g., early partial aggregation via a combiner)
- Executing the system-level dataflow
  - (i.e., running the MapReduce jobs)
- Tracking progress, errors, etc.

Logic Plan

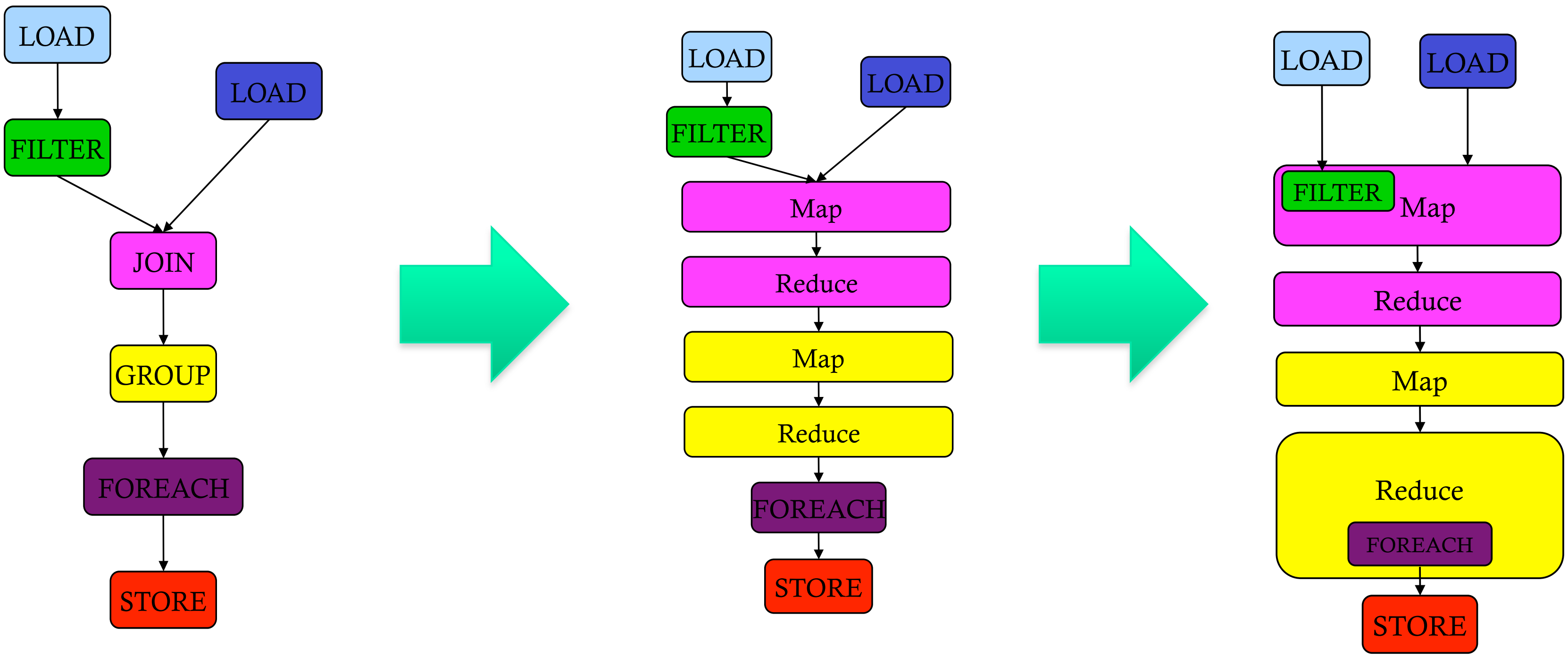
```
A=LOAD 'file1' AS (x, y, z);

B=LOAD 'file2' AS (t, u, v);

C=FILTER A by y > 0;

D=JOIN C BY x, B BY u;

E=GROUP D BY z;

F=FOREACH E GENERATE
      group, COUNT(D);

STORE F INTO 'output';
```



Physical Plan

- 1:1 correspondence with the logical plan
- **Except for:**
  - Join, Distinct, (Co)Group, Order
- Several optimizations are done automatically

Generation of Physical Plans



If the Join and Group By are on the same key
➔ The two map-reduce jobs would be merged into one.

Another Example: WordCount

```
Lines=LOAD 'input/hadoop.log' AS (line: chararray);
Words = FOREACH Lines GENERATE  FLATTEN(TOKENIZE(line))
AS word;
Groups = GROUP Words BY word;
Counts = FOREACH Groups GENERATE  group, COUNT(Words);
Results = ORDER Words BY Counts DESC;
Top5 = LIMIT Results 5;
STORE Top5 INTO /output/top5words;
```

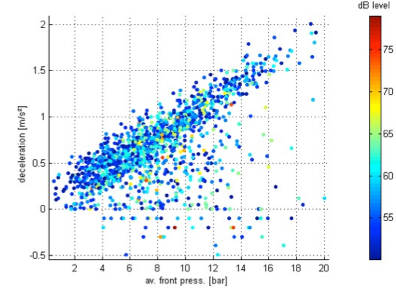
Real World Example: Counting sources of Twitter users

Where are users querying from? The API, the front page, their profile page, etc?

search_origin.pig

```
raw_data = load '$INPUT_FILES' using com.twitter.twadoop.pig.storage.LzoTwitterApacheLogLoader() as
    (..., virtual_host: chararray, apache_time: chararray, request_method: chararray, request_url: chararray,
     request_protocol: chararray, response_code: chararray, response_size: int, referrer: chararray,
     user_agent: chararray, response_microseconds: int, ...);

searches_only = filter raw_data by com.twitter.twadoop.pig.piggybank.IsSearchUrl(request_url);

searches_with_type = foreach searches_only generate
    com.twitter.twadoop.pig.piggybank.ExtractSearchOrigin(virtual_host, request_url) as origin;

grouped = group searches_with_type by origin parallel $PARALLEL;

counted = foreach grouped generate group, COUNT(searches_with_type);

store counted into 'searches_by_type.tsv' using PigStorage('\t');
```

Another Real World example: Correlating Big Data at Twitter



What is the correlation between users with registered phones and users that tweet?

correlation.pig

```
register piggybank.jar;

devices = load '$DEVICES' using com.twitter.twadoop.pig.storage.LzoDevicesLoader() as
  (device_id: long, user_id: long, ...);

tweets = load '$TWEETS' using com.twitter.twadoop.pig.storage.LzoStatusLoader() as
  (tweet_id: long, user_id: long, text: chararray, ...);

tweet_user_ids = foreach tweets generate user_id;
tweets_grouped = group tweet_user_ids by user_id;
tweets_by_user = foreach tweets_grouped generate group as user_id, COUNT(tweet_user_ids) as count;

combined = cogroup devices by user_id, tweets_by_user by user_id;

summed = foreach combined generate user_id, SIZE(devices) as has_device,
    (SIZE(tweets_by_user) == 0 ? 0 : SUM(tweets_by_user.count)) as tweet_count;

summed_grouped = group summed all;

covariance = foreach summed_grouped generate group,
    COR(summed_grouped.has_device, summed_grouped.tweet_count);
```

One more Real World example:
How to distinguish Bot from Human at Twitter ?

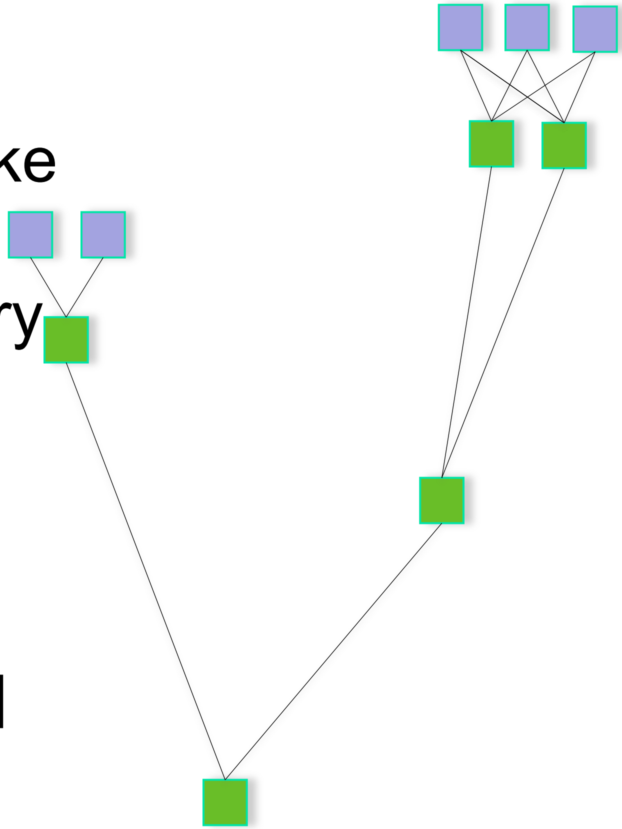
bot_probability.pig

```
raw_data = load '$INPUT_FILES' using com.twitter.twadoop.pig.storage.LzoStatusLoader() as
    (id: long,
     user_id: long,
     text: chararray,
     created_at: chararray,
     ...);

projected = foreach raw_data generate user_id, text;

grouped_by_user = group projected by user_id parallel $PARALLEL;

bot_probabilities = foreach grouped_by_user generate group as user,
  com.twitter.twadoop.pig.piggybank.UserBotProbability(projected.text) as bot_probability;

store bot_probabilities into 'bot_test.tsv' using PigStorage('\t');
```

Digression to Apache Tez

Apache Tez – Introduction

- Distributed execution framework targeting data-processing applications.
  - NOT a standlone computation engine like MapReduce or Spark ; Instead, it is intended to be use as a "backend" library
- Based on expressing a computation as a DAG dataflow graph.
  - Claim to be inspired by Dryad
- Highly customizable to meet a broad spectrum of use cases.
- Built on top of YARN – the resource management framework for Hadoop.



Hadoop 1 -> Hadoop 2

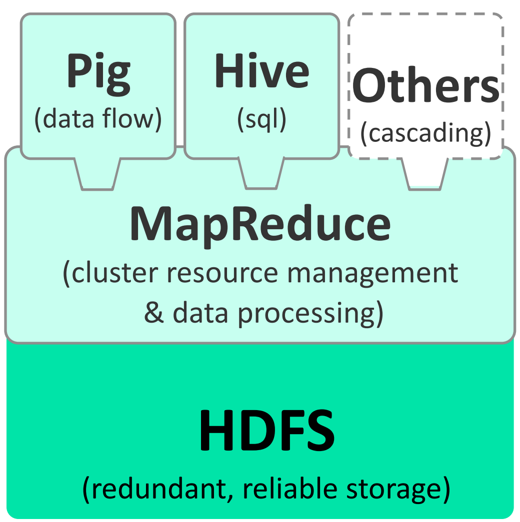
Monolithic

- Resource Management
- Execution Engine
- User API

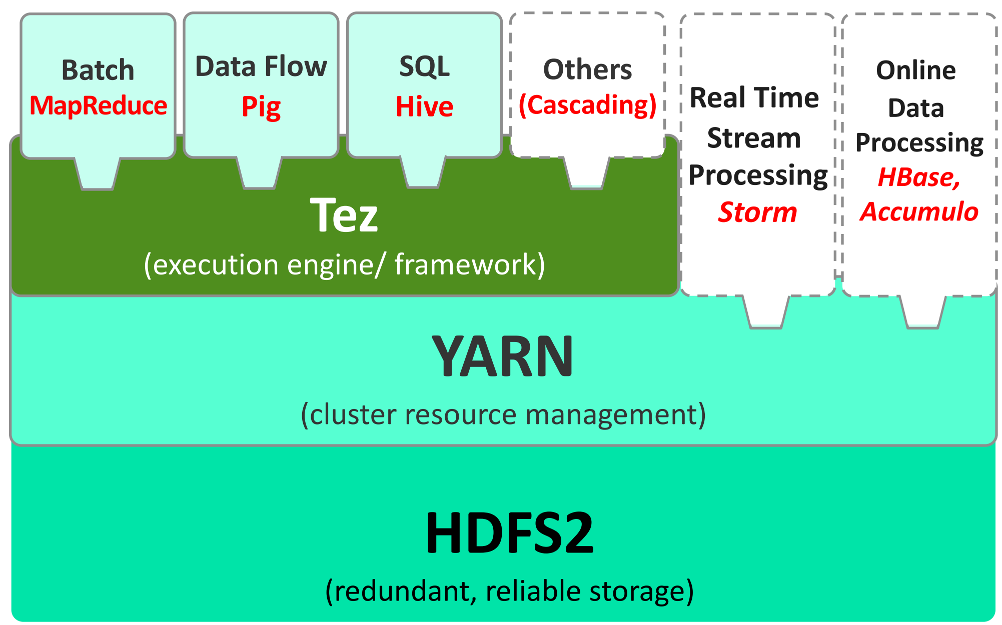
Layered

- Resource Management – YARN
- Execution Engine – Tez
- User API – Hive, Pig, Cascading, Your App, even experimental support for MapReduce and Spark !!

HADOOP 1.0



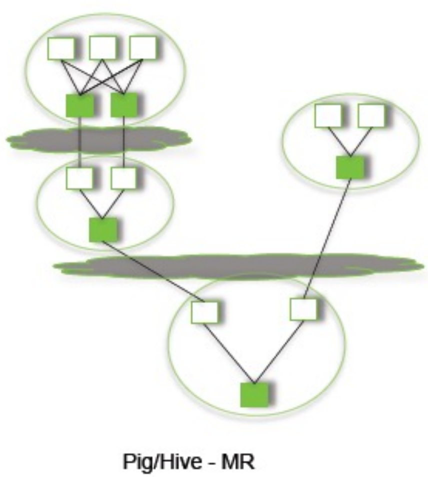
HADOOP 2.0

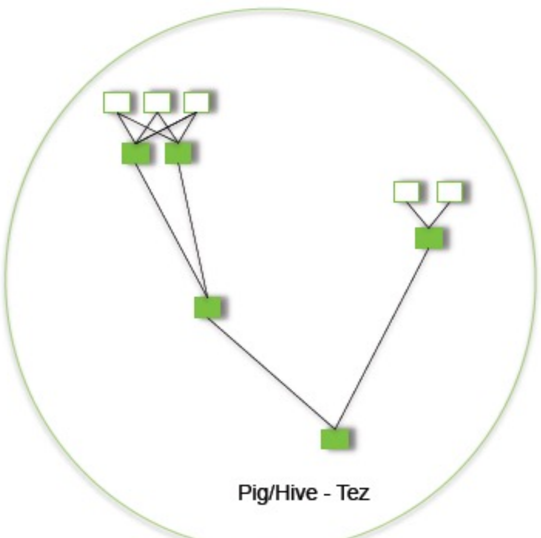


TeZ: An Alternative Execution Library for realizing Pig's Logical Computation Plan (since Pig rel. 14)

Step 1. The Required Data Processing Flow is represented as a Directed-Acyclic Graph (DAG) (which is what PIG has been doing all along)

Step 2. TeZ can then be used to realize/execute the DAG-based dataflow/computation to avoid limitations imposed by the rigid 2-stage MapReduce model

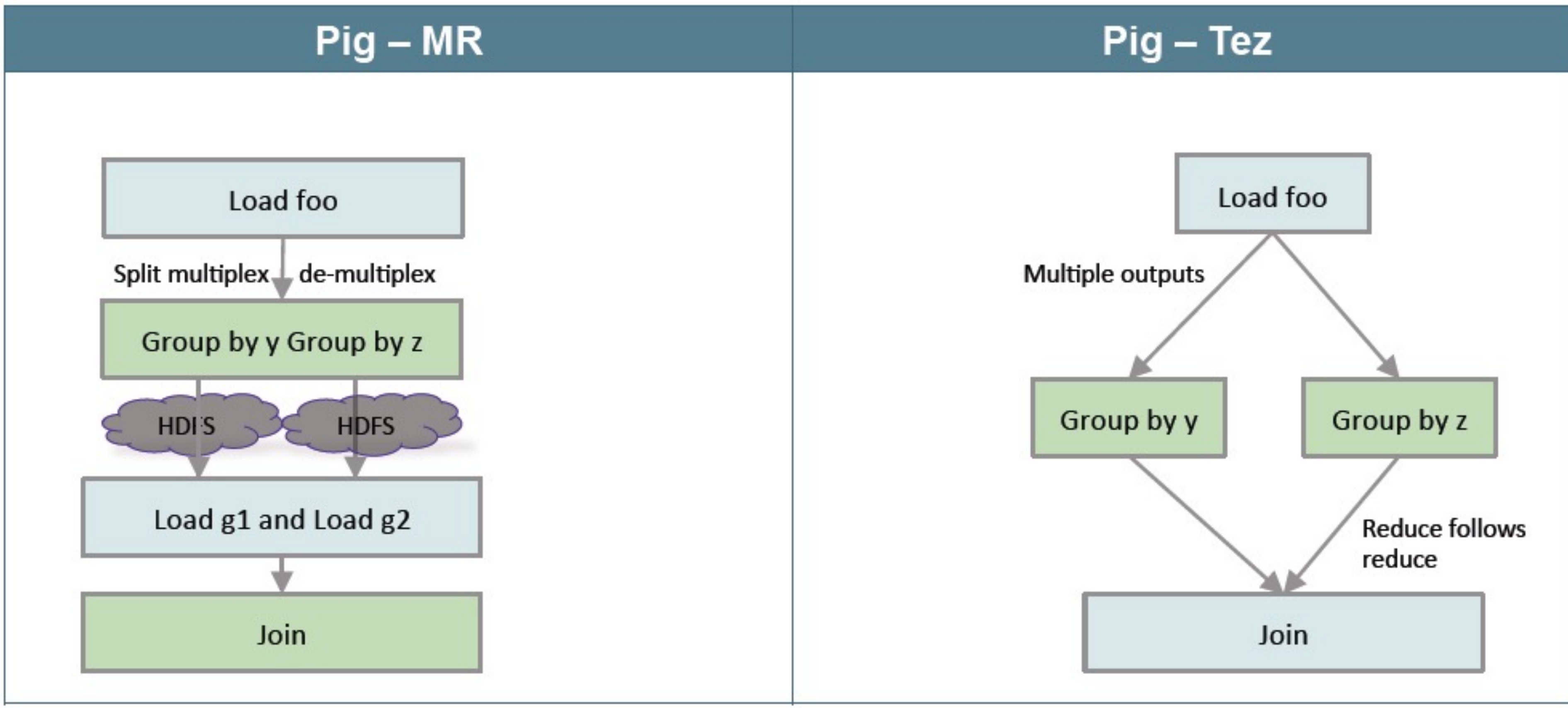




MapReduce (MR) vs. TeZ-based Execution Plans of a Sample Pig Job:

```
f = LOAD ‘foo’ AS (x, y, z);
g1 = GROUP f BY y;
g2 = GROUP f BY z;
j = JOIN g1 BY group,
         g2 BY group;
```

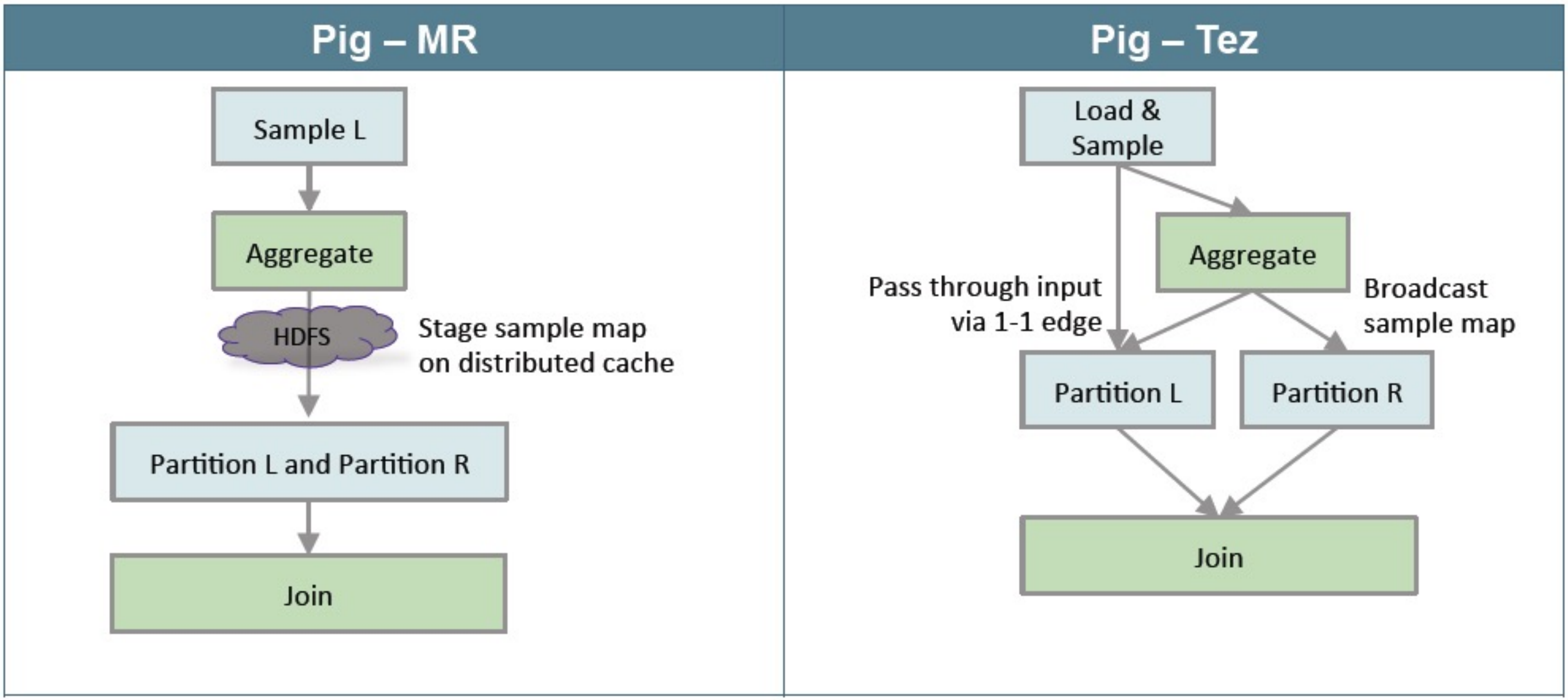
Pig : Split & Group-by



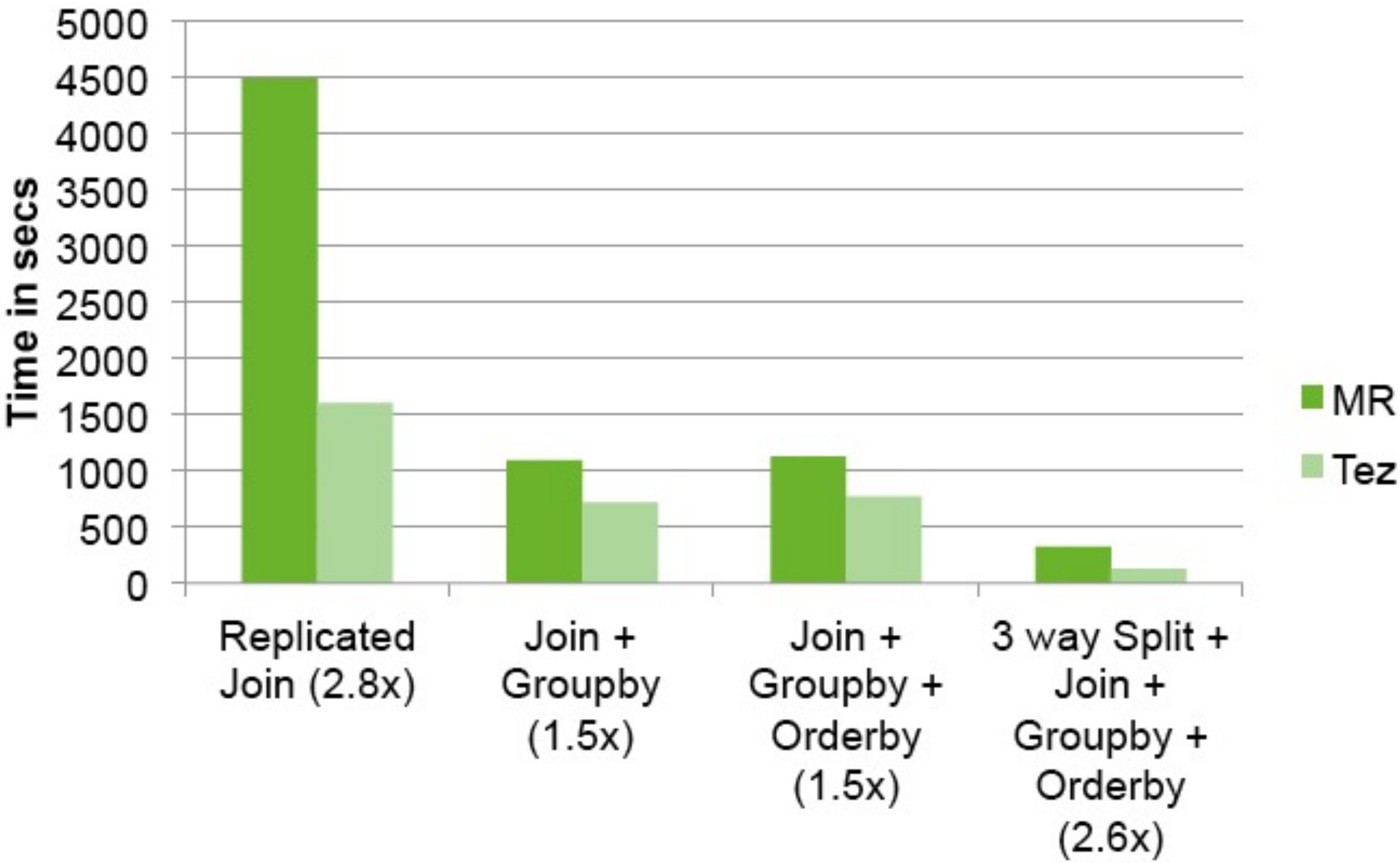
MapReduce (MR) vs. TeZ-based Execution Plans of Another Sample Pig Job:

```
l = LOAD 'left' AS (x, y);
r = LOAD 'right' AS (x, z);
j = JOIN l BY x, r BY x
    USING 'skewed';
```

Pig : Skewed Join



Pig-TeZ Performance Gain over Pig-MapReduce



Note: These results are published by Hortonworks, a major contributor to TeZ

Pig References

- **Pig Documentation**
  - http://pig.apache.org/docs/r0.15.0/ (as of June 2015)
- **PigMix Queries (Test-suite testing and Benchmarking)**
  - https://cwiki.apache.org/PIG/pigmix.html

Hive

Hive: Background

- Started at Facebook
- Data was collected by nightly cron jobs into Oracle DB
- "ETL" via hand-coded python
- Grew from 10s of GBs (2006) to 1 TB/day new data (2007)
- Recent Hive Usage @ Facebook:
  - 300+ PB of Data under management [1] ;
  - 600+ TB new data loaded per day [2];
  - 60K+ Hive queries per day
  - 1000+ users per day
- HQL, a variant of SQL
  - But since we can only read already existing files in HDFS it is lacking UPDATE or DELETE support for example
  - Focuses primarily on the query part of SQL
  - Paper published later by Thusoo et al, VLDB 2009
  - Initial Apache release in April 2009

[1] https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/

[2] https://www.facebook.com/notes/facebook-engineering/under-the-hood-scheduling-mapreduce-jobs-more-efficiently-with-corona/10151142560538920

Apache Hive

- A data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis
- **Hive Provides**
  - ETL
  - Structure
  - Access to different storage (HDFS or HBase)
  - Query execution via MapReduce
- **Key Building Principles**
  - SQL is a familiar language
  - Extensibility – Types, Functions, Formats, Scripts
  - Scalability and Performance – ability to process queries for TB/PB of data

Hive Use Cases

Large-scale Data Processing with SQL-style Syntax:

- Predictive Modeling & Hypothesis Testing
- Customer Facing Business Intelligence
- Document Indexing
- Text Mining and Data Analysis

Hive Components

Two Main Components

- High-level language (HiveQL)
  - Set of commands
- Two execution modes
  - Local: reads/write to local file system
  - Mapreduce: connects to Hadoop cluster and reads/writes to HDFS

Two modes

- Interactive mode
  - Console
- Batch mode
  - Submit a script

Digression:
Background on Relational Database and SQL

Materials from Lecture Videos by Profs. Joe Hellerstein and Alvin Cheung; extracted from the UC Berkeley course CS186: Introduction to Database Management Systems

- https://cs186berkeley.net/fa21/
- https://www.youtube.com/user/CS186Berkeley/playlists

In particular,

- SQL Part I: https://www.youtube.com/playlist?list=PLzzVuDSjP25R1px8yE4wJcXcRbwsCuunP
- SQL Part II: https://www.youtube.com/playlist?list=PLzzVuDSjP25QapEtTMxw56ZtKRf62IkL_

Hive: Example

- Hive looks similar to an SQL database
- Relational join on two tables:
  - Table of word counts from Shakespeare collection
  - Table of word counts from Homer

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN homer k ON (s.word = k.word) WHERE s.freq >= 1
AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

the	25848	62394
I	23031	8854
and	19671	38985
to	18038	13526
of	16700	34654
a	14170	8057
you	12702	2720
my	11297	4135
in	10797	12445
is	8882	6884

Source: Material drawn from Cloudera training VM

Hive: Behind the Scenes

SELECT s.word, s.freq, k.freq FROM shakespeare s
JOIN homer k ON (s.word = k.word) WHERE s.freq >= 1
AND k.freq >= 1
ORDER BY s.freq DESC LIMIT 10;

Abstract Syntax Tree)

(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF homer k) (= (.
(TOK_TABLE_OR_COL s) word) (. (TOK_TABLE_OR_COL k) word)))) (TOK_INSERT (TOK_DESTINATION
(TOK_DIR TOK_TMP_FILE)) (TOK_SELECT (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) word))
(TOK_SELEXPR (. (TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL k) freq)))
(TOK_WHERE (AND (>= (. (TOK_TABLE_OR_COL s) freq) 1) (>= (. (TOK_TABLE_OR_COL k) freq) 1)))
(TOK_ORDERBY (TOK_TABSORTCOLNAMEDESC (. (TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))

(one or more of MapReduce jobs)

Hive: Behind the Scenes

```
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        s
          TableScan
            alias: s
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 0
                value expressions:
                      expr: freq
                      type: int
                      expr: word
                      type: string
        k
          TableScan
            alias: k
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 1
                value expressions:
                      expr: freq
                      type: int
```

```
Reduce Operator Tree:
      Join Operator
        condition map:
             Inner Join 0 to 1
        condition expressions:
          0 {VALUE._col0} {VALUE._col1}
          1 {VALUE._col0}
        outputColumnNames: _col0, _col1, _col2
        Filter Operator
          predicate:
              expr: ((_col0 >= 1) and (_col2 >= 1))
              type: boolean
          Select Operator
            expressions:
                  expr: _col1
                  type: string
                  expr: _col0
                  type: int
                  expr: _col2
                  type: int
            outputColumnNames: _col0, _col1, _col2
            File Output Operator
              compressed: false
              GlobalTableId: 0
              table:
                  input format: org.apache.hadoop.mapred.SequenceFileInputFormat
                  output format: org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat
```

```
Stage: Stage-2
   Map Reduce
     Alias -> Map Operator Tree:
       hdfs://localhost:8022/tmp/hive-training/364214370/10002
           Reduce Output Operator
             key expressions:
                   expr: _col1
                   type: int
             sort order: -
             tag: -1
             value expressions:
                   expr: _col0
                   type: string
                   expr: _col1
                   type: int
                   expr: _col2
                   type: int
     Reduce Operator Tree:
       Extract
         Limit
           File Output Operator
             compressed: false
             GlobalTableId: 0
             table:
                 input format: org.apache.hadoop.mapred.TextInputFormat
                 output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat

Stage: Stage-0
   Fetch Operator
     limit: 10
```

Data Model for Hive

Very similar to SQL and Relational DBs

Hive deals with Structured Data, of different types:

3-Levels: Tables → Partitions → Buckets

- Tables
  - Typed columns (int, float, string, boolean)
  - Similar to Tables in RDBMS
  - Each Table is a Unique directory in HDFS
  - Also, list: map (for JSON-like data)

- Partitions
  - To determine the distribution of data within a Table
    - For example, range-partition tables by date
  - Each Partition is a sub-directory of the main directory in HDFS

- Buckets (or Clusters)
  - Partitions can be further divided into Buckets
    - e.g.Hash partitions within ranges (useful for sampling, join optimization)
  - Each Bucket is stored as a file in the directory

Source: cc-licensed slide by Cloudera

HiveQL Commands

- Data Definition Language (DDL)
  - Used to describe, view and alter Tables
  - e.g. CREATE TABLE, DROP TABLE commands with extensions to define file formats, partitioning and bucketing information
- Data Manipulation Language (DML)
  - Used to load data from external tables and insert rows using the LOAD and INSERT commands
- Query Statements
  - SELECT, JOIN, UNION, etc

- Refer to http://hortonworks.com/wp-content/uploads/2016/05/Hortonworks.CheatSheet.SQLtoHive.pdf

for "Cheat Sheet" for subtle differences between SQL and Hive QL.

Hive Data Definition Language (DDL)

- **CREATE TABLE** sample (foo INT, bar STRING) **PARTITIONED BY** (ds STRING);
- **SHOW TABLES** '.*s';
- **DESCRIBE** sample;
- **ALTER TABLE** sample **ADD COLUMNS** (new_col INT);
- **DROP TABLE** sample;

A table in Hive is an HDFS directory in Hadoop

```
create table table_name (
  id                int,
  dtDontQuery       string,
  name              string
)
partitioned by (date string)
```

Schema is known at creation time (like DB schema)

Partitioned tables have "sub-directories", one for each partition

Hive Data Manipulation Language (DML)

Load data from local file system

Delete previous data from that table

- **LOAD DATA LOCAL INPATH** './sample.txt' **OVERWRITE INTO TABLE** sample;

Load data from HDFS

Augment to the existing data

- **LOAD DATA INPATH** '/user/falvariz/hive/sample.txt' **INTO TABLE** partitioned_sample **PARTITION** (ds='2012-02-24');

Must define a specific partition for partitioned tables

Loaded data are files copied to HDFS under the corresponding directory

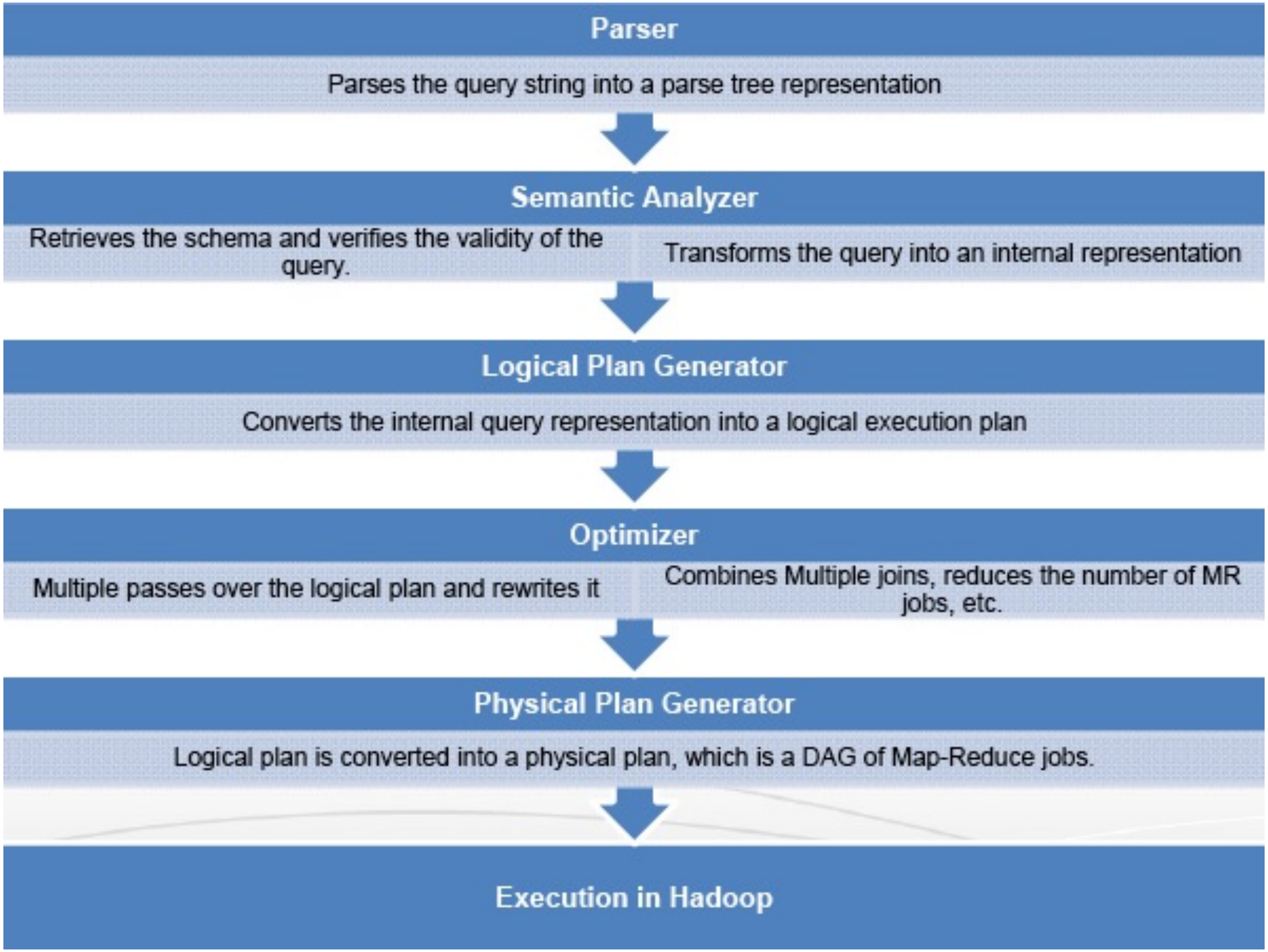
User Defined Functions (UDFs) in Hive

Four Types

- User Defined Functions (UDF)
  - Perform tasks such as Substr, Trim, etc on data elements
- User Defined Aggregation Function (UDAF)
  - Perform Operations on Columns, e.g. Sum, Average, Max, Min,…
- User Defined Table-Generating Functions (UDTF)
  - Output a new table

  e.g. the "Explode" function which is similar to FLATTEN( ) in Pig
- Custom MapReduce scripts
  - The MR scripts must read rows from standard output
  - Write rows to standard input

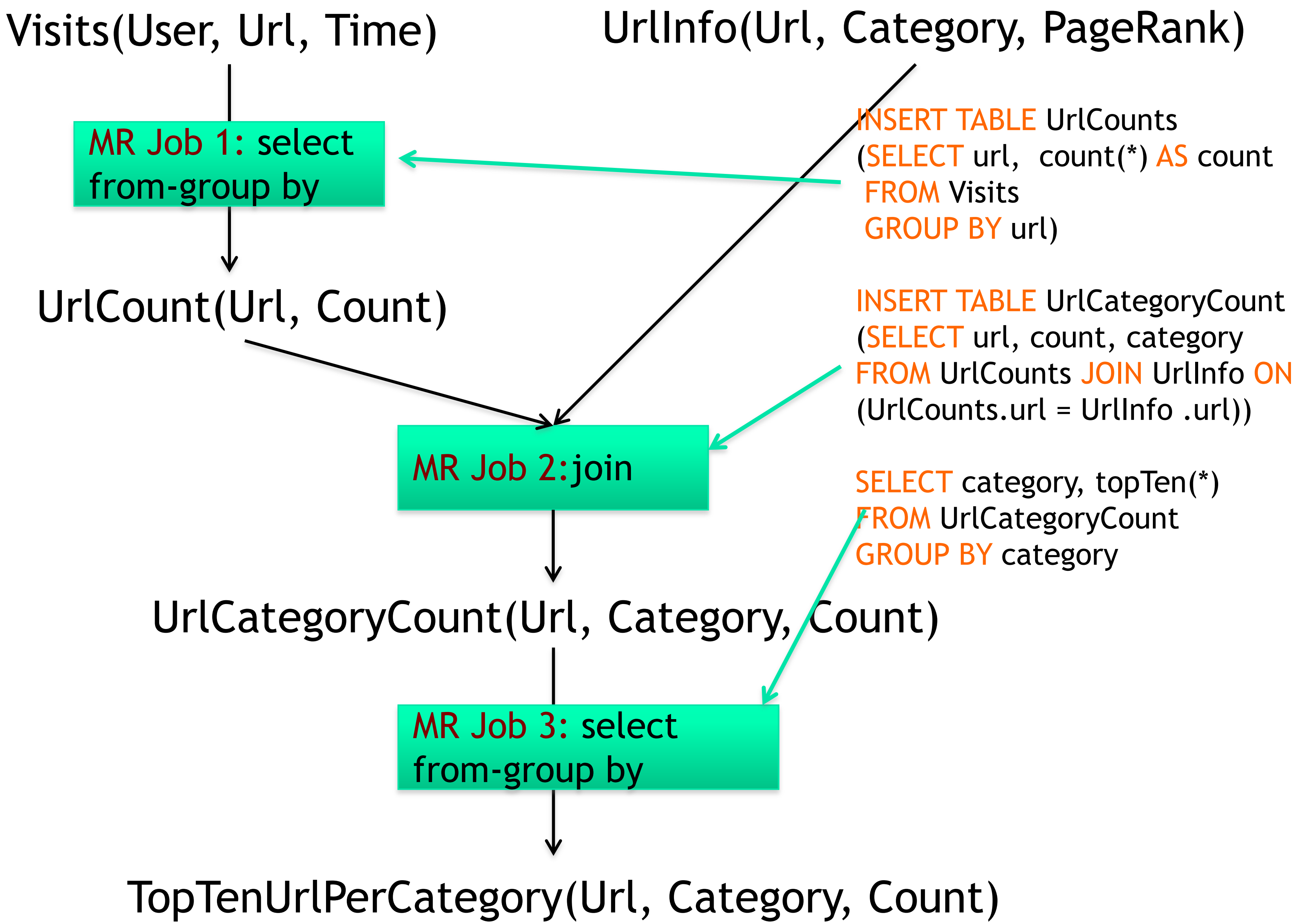
Compilation of Hive Programs



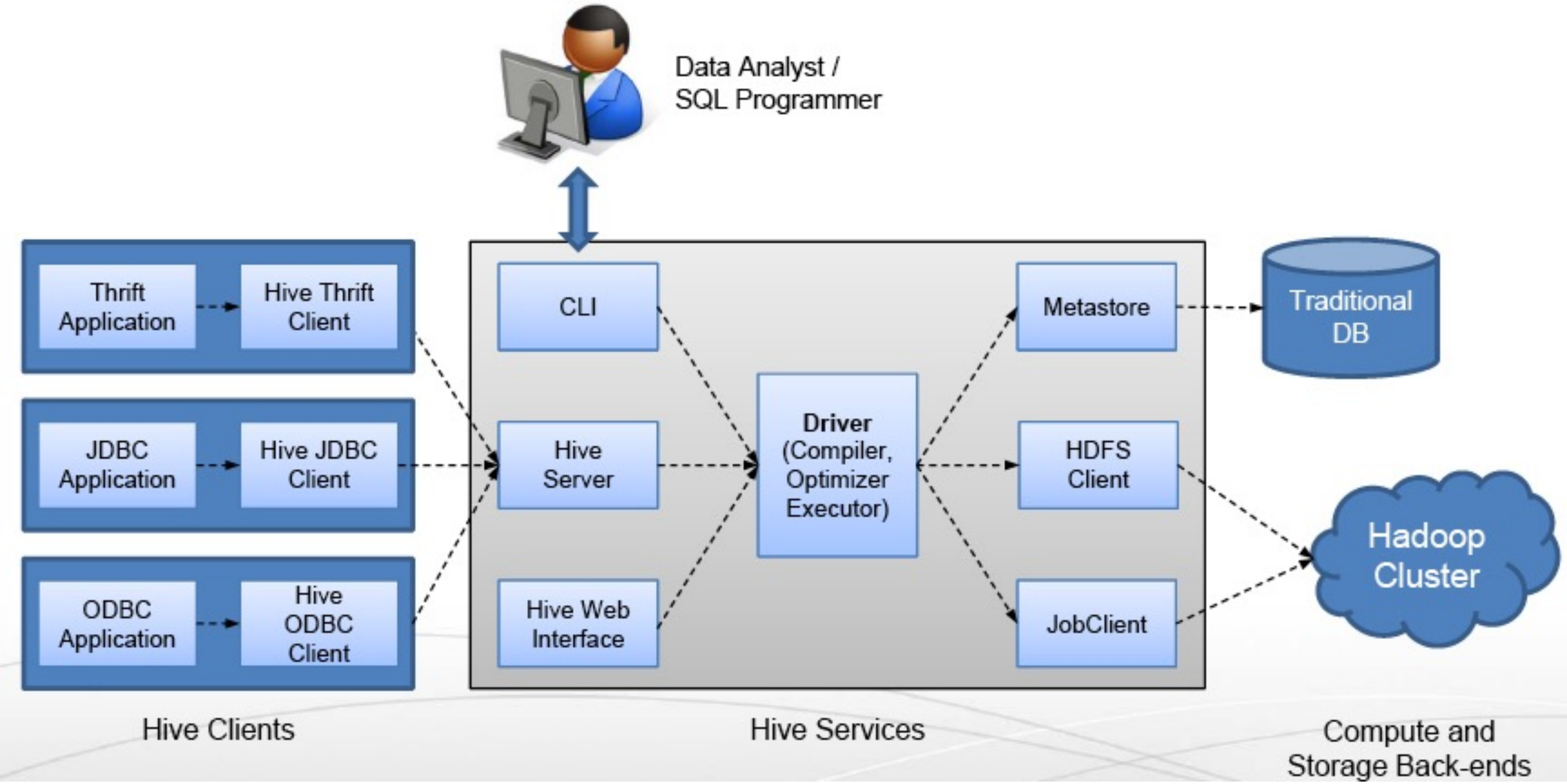
Another Hive Example

```
INSERT TABLE UrlCounts
(SELECT url,  count(*) AS count
 FROM Visits
 GROUP BY url)

INSERT TABLE UrlCategoryCount
(SELECT url, count, category
FROM UrlCounts JOIN UrlInfo ON (UrlCounts.url = UrlInfo .url))

SELECT category, topTen(*)
FROM UrlCategoryCount
GROUP BY category
```

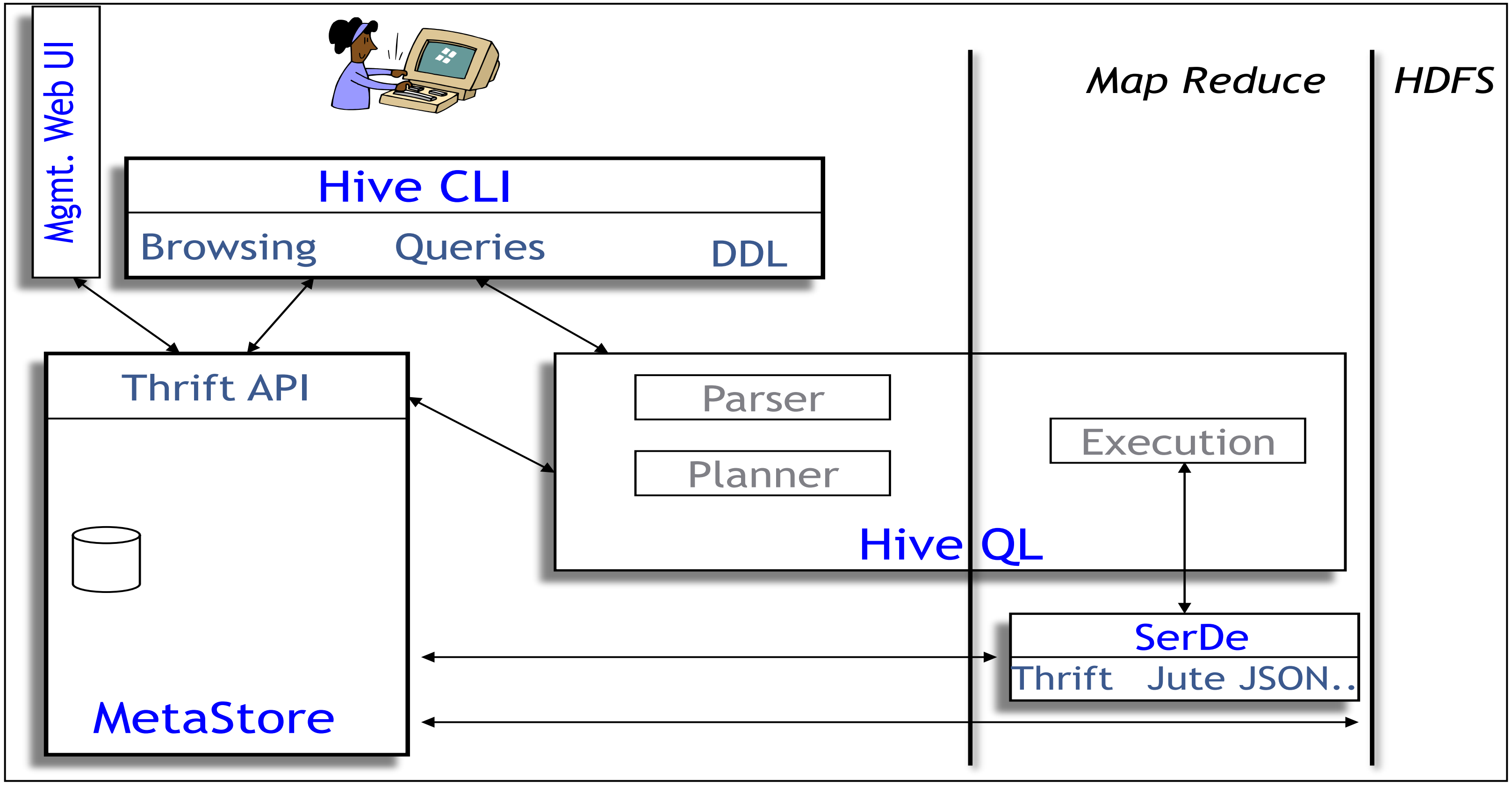
Hive Final Execution



Architecture of Hive



Hive Components



- Hive CLI: Hive Client Interface
- MetaStore: For storing the schema information, data types, partitioning columns, etc…
- Hive QL: The query language, compiler, and executer

Hive Components

- Shell: allows interactive queries
- Driver: session handles, fetch, execute
- Compiler: parse, plan, optimize
- Execution engine: DAG of stages (MR, HDFS, metadata)
- Metastore: schema, location in HDFS, etc

Source: cc-licensed slide by Cloudera

Metastore

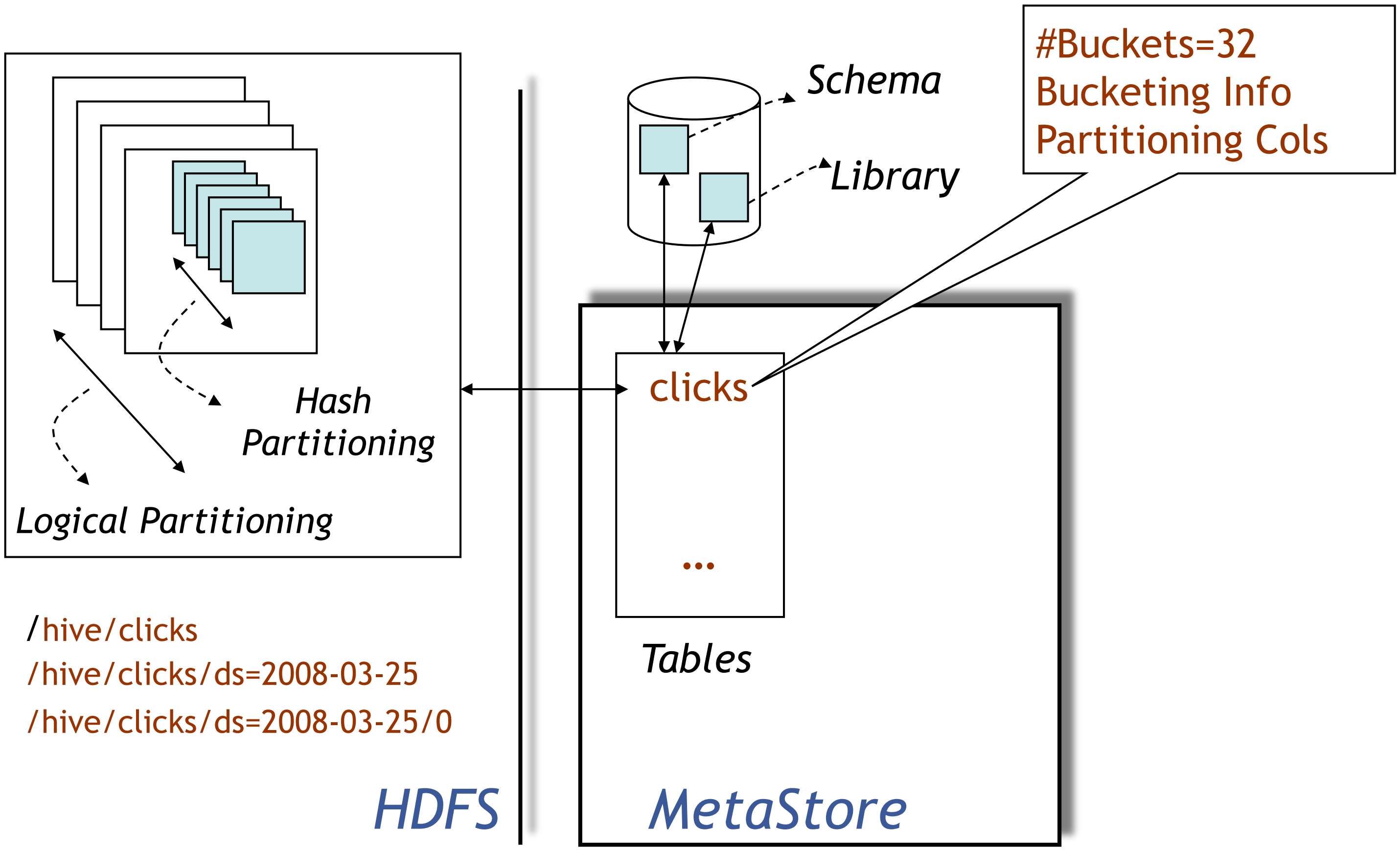
- Database: namespace containing a set of tables
- Holds table definitions (column types, physical layout)
- Holds partitioning information
- Can be stored in Derby, MySQL, and many other relational databases

Source: cc-licensed slide by Cloudera

Recap: Hive Data Model

- **Table: maps to a HDFS directory**
  - Table R: Users all over the world
- **Partition: maps to sub-directories under the table**
  - Partition R by country name
  - It is the user's responsibility to upload the right data to the right partition
- **Bucket: maps to files under each partition**
  - Divide a partition into buckets based on a hash function on a certain column(s)

Data Model (Cont'd)



Physical Layout

- Warehouse directory in HDFS
  - E.g., /user/hive/warehouse
- Tables stored in subdirectories of warehouse
  - Partitions form subdirectories of tables
- Actual data stored in flat files
  - Control char-delimited text, or SequenceFiles
  - With custom SerDe, can use arbitrary format

Source: cc-licensed slide by Cloudera

Query Examples I: Select & Filter

- **SELECT** foo **FROM** sample **WHERE** ds='2012-02-24';
- **INSERT OVERWRITE DIRECTORY** '/tmp/hdfs_out' **SELECT** * **FROM** sample **WHERE** ds='2012-02-24';
- **INSERT OVERWRITE LOCAL DIRECTORY** '/tmp/hive-sample-out' **SELECT** * **FROM** sample;

Create HDFS dir for the output

Create local dir for the output

Query Examples II: Aggregation & Grouping

- **SELECT MAX**(foo) **FROM** sample;
- **SELECT** ds, **COUNT**(*), **SUM**(foo) **FROM** sample **GROUP BY** ds;

Hive allows the From clause to come first !!!

Store the results into a table

- **FROM** sample s **INSERT OVERWRITE TABLE** bar **SELECT** s.bar, count(*) WHERE s.foo > 0 **GROUP BY** s.bar;

This new syntax is to facilitate the "Multi-Insertion"

Query Examples III: Multi-Insertion

```
FROM page_view_stg pvs
   INSERT OVERWRITE TABLE page_view PARTITION(dt='2008-06-08',
    country='US')
        SELECT pvs.viewTime, … WHERE pvs.country = 'US'
   INSERT OVERWRITE TABLE page_view PARTITION(dt='2008-06-08',
    country='CA')
        SELECT pvs.viewTime, ... WHERE pvs.country = 'CA'
   INSERT OVERWRITE TABLE page_view PARTITION(dt='2008-06-08',
    country='UK')
        SELECT pvs.viewTime, ... WHERE pvs.country = 'UK';
```

Example IV: Joins

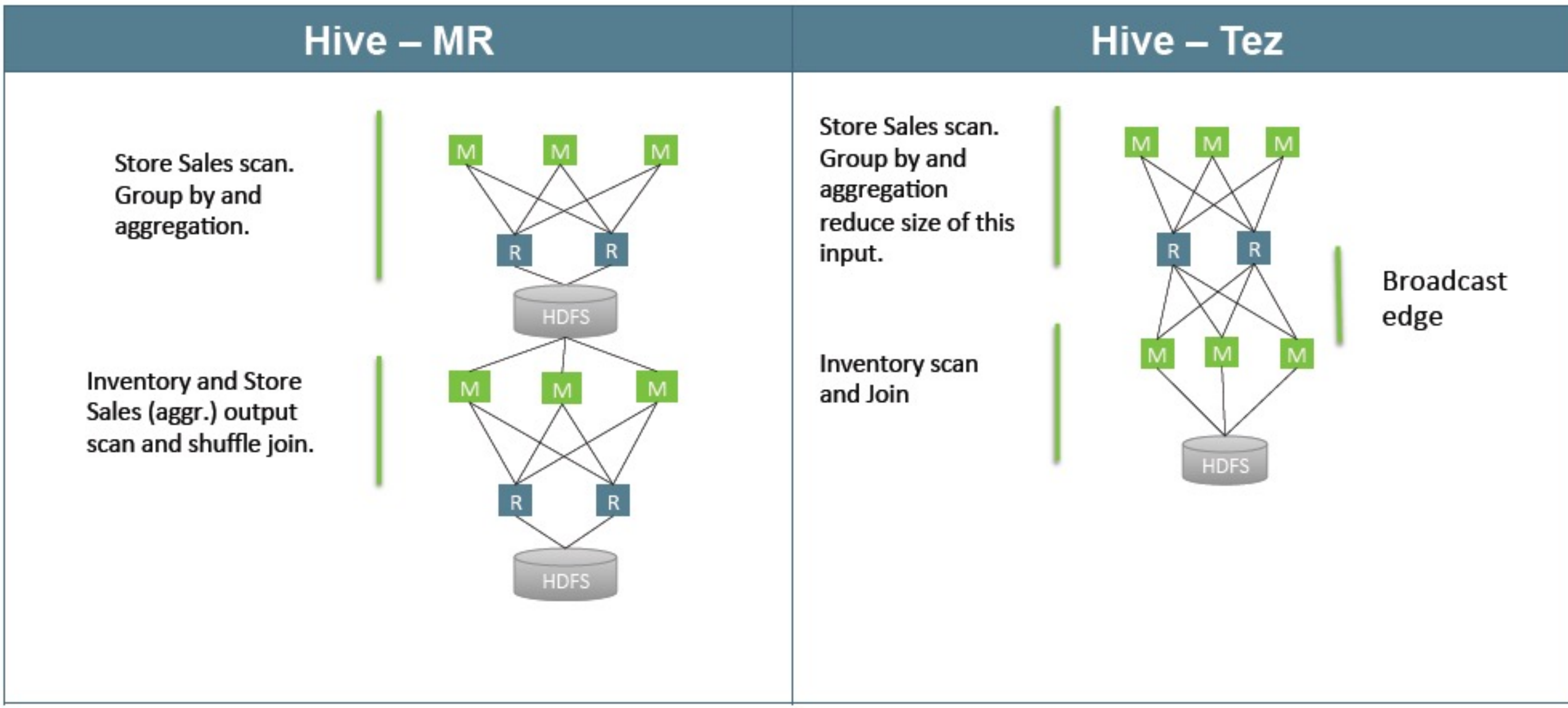
```
CREATE TABLE customer (id INT,name STRING,address STRING)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY '#';

CREATE TABLE order_cust (id INT,cus_id INT,prod_id INT,price INT)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

- ```
  SELECT * FROM customer c JOIN order_cust o ON (c.id=o.cus_id);
  ```
- ```
  SELECT c.id, c.name, c.address, ce.exp
  FROM customer c JOIN (SELECT cus_id,sum(price) AS exp
                        FROM order_cust
                        GROUP BY cus_id) ce ON (c.id=ce.cus_id);
  ```

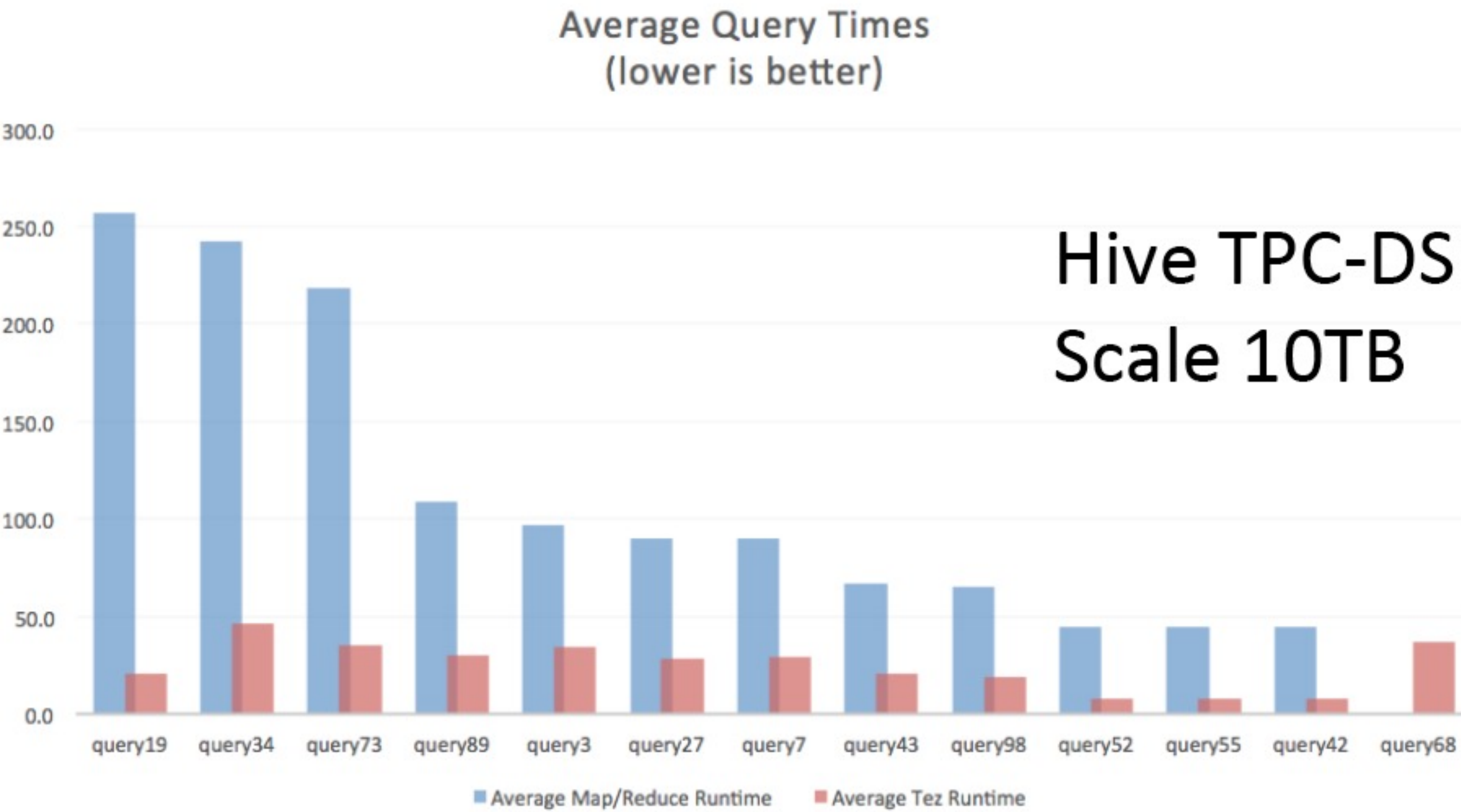
MapReduce (MR) vs. TeZ-based Execution Plans for a Sample Hive Job

```
SELECT ss.ss_item_sk, ss.ss_quantity, avg_price, inv.inv_quantity_on_hand
FROM (select avg(ss_sold_price) as avg_price, ss_item_sk, ss_quantity_sk from store_sales
group by ss_item_sk) ss
JOIN inventory inv
ON (inv.inv_item_sk = ss.ss_item_sk);
```

Hive : Broadcast Join

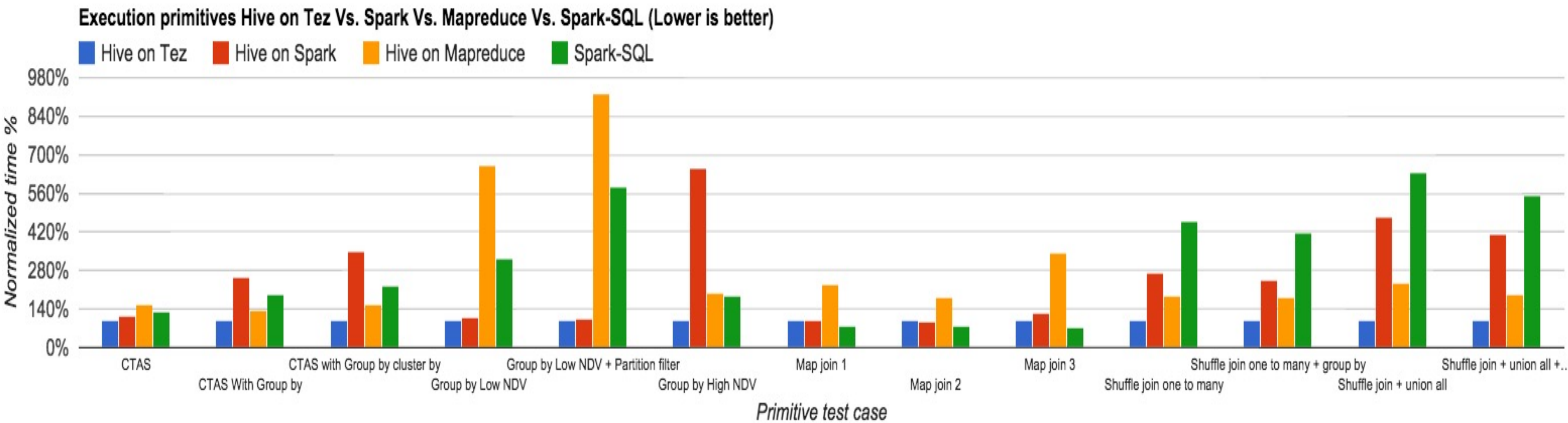


Hive-TeZ Performance Gain over Hive-MapReduce



Note: These results are published by Hortonworks, a major contributor to TeZ

More Performance Comparison among Other Alternative Execution Engines for Hive



Note: These results are published by Hortonworks, a major contributor to TeZ

More Performance Comparison among Other Alternative Execution Engines for Hive

Hive on Tez

- + Short running query
- + ETL
- + Large joins and aggregates
- + Efficient resource utilization
- − Slower than Spark-SQL in Map joins

Spark-SQL

- + Map join performance
- − Large Joins
- − High Garbage Collection
- − Instability
- − SQL support limited compared to Hive
- − Lack of sophisticated query optimizer

Hive on Spark

- + Outperforms Spark-SQL in large join
- + Promising initial release
- − High Garbage Collection
- − Slower than Tez for large joins and aggregates

Hive on MapReduce

Note: These results are published by Hortonworks, a major contributor to TeZ

More alternatives to Hive for SQL-for-Hadoop/Big Data: BigSQL, Spark SQL, Impala, Presto, HAWQ,…

- BigSQL (from IBM) provides an alternative execution engine (without using Hadoop/TeZ) but preserves Hive Storage and Hive metastore ;
  - Unlike Hive or Spark-SQL, BigSQL provides 100% ANSI SQL compatibility (by leveraging IBM's deep experience in SQL from its database products like DB2)
  - However, BigSQL is not open-source and you need to buy it from IBM
- Spark SQL, Spark SQL over Parquet, Spark SQL over Kudu
- Impala (Cloudera): Impala-Kudu, Impala-Parquet
- Presto (Facebook ->Teradata -> Starburst)
- HAWQ (Pivotal -> Hortonworks HDB -> Apache ->?)
- Apache Phoenix: Phoenix (SQL) over Hbase

Note: Performance Results reported in this page were produced by IBM.

Big Data Frameworks Adoption Trends