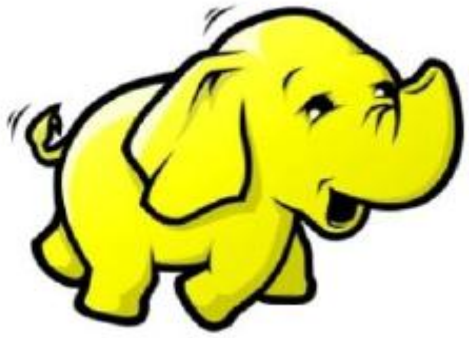
Big Data & Hadoop

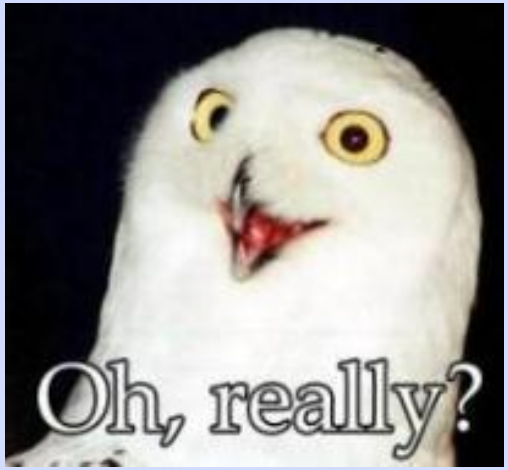


PIG

Need of PIG

✓ Do You know JAVA ?

✓ 10 lines of PIG = 200 lines of Java



Built in operations like:

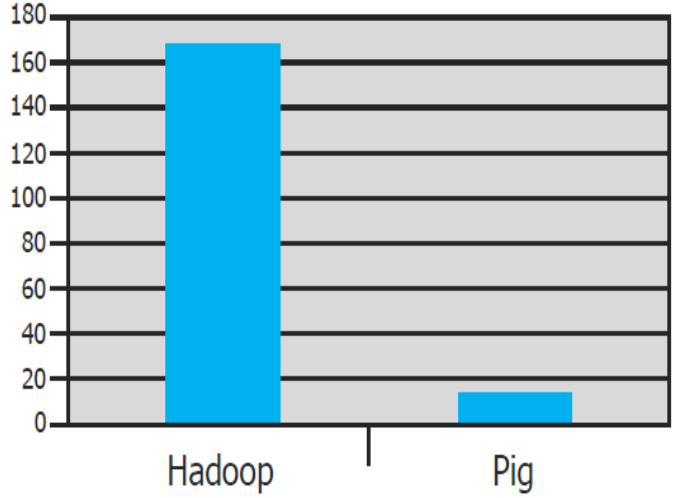
- ✓ Join
- ✓ Group
- ✓ Filter
- ✓ Sort
- ✓ and more…

Need of PIG

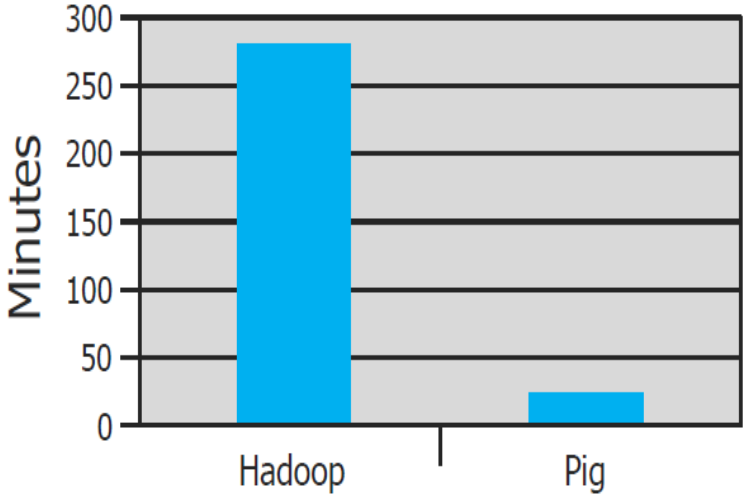
- ✓ An ad-hoc way of creating and executing map-reduce jobs on very large data sets
- ✓ Rapid Development
- ✓ No Java is required

Why Should I go for PIG ?

1/20 the lines of Code



1/16 the development Time



Performance On Par With Raw Hadoop

Why Should I Go For Pig When There Is MR?

MapReduce

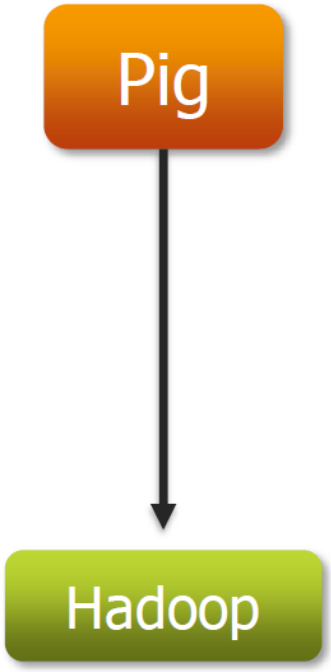
- ✓ Powerful model for parallelism.
- ✓ Based on a rigid procedural structure.
- ✓ Provides a good opportunity to parallelize algorithm.

PIG

- ✓ It is desirable to have a higher level declarative language.
- ✓ Similar to SQL query where the user specifies the “what” and leaves the “how” to the underlying processing engine.

Where should I use Pig ?

- Pig is Data Flow Language



It is at the top of Hadoop and makes it possible to create complex jobs to process large volumes of data quickly and efficiently.

Case 1 – Time Sensitive Data Loads

Case 2 – Processing Many Data Sources

Case 3 – Analytic Insight Through Sampling

Where NOT to use Pig ?

- ✓ Really nasty data formats or **completely unstructured data**(video, audio, raw human-readable text).
- ✓ Pig is definitely **slow** compared to Map Reduce jobs.
- ✓ When you would like **more power** to optimize your code.

What is Pig ?

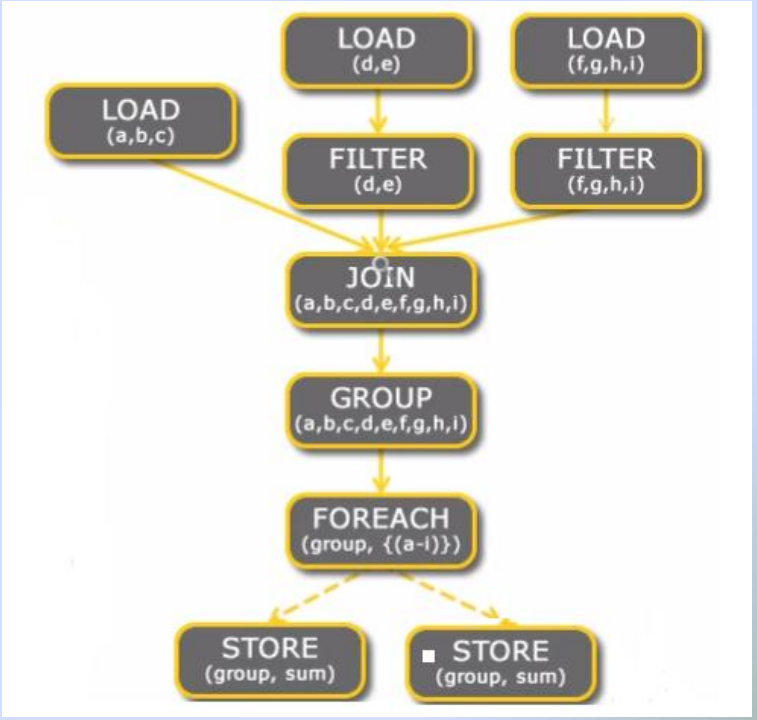
Pig is an open-source high-level dataflow system.

It is High-Level Data Flow scripting language.

```
A = LOAD 'file.dat' AS (a,b,c);
B = LOAD 'file2.dat' AS (d,e);
C = LOAD 'file3.dat' AS (f,g,h);
D = FILTER B by e > 0;
E = FILTER C by g == 'piggy';
F = JOIN C by f, B by d, A by a;
G = GROUP F by h;
H = FOREACH G GENERATE
    group, SUM(b);
STORE H INTO 'results';
```

Loading

SQL Like



Pig Vs SQL

SQL (Declarative)

```
SELECT cust_id, SUM(amount) AS CstTotal
FROM customers c
JOIN sales s ON c.cust_ID = s.cust_id
WHERE c.region = 'India'
GROUP BY cust_id
HAVING SUM(amount) > 100000
ORDER BY CstTotal DESC
```

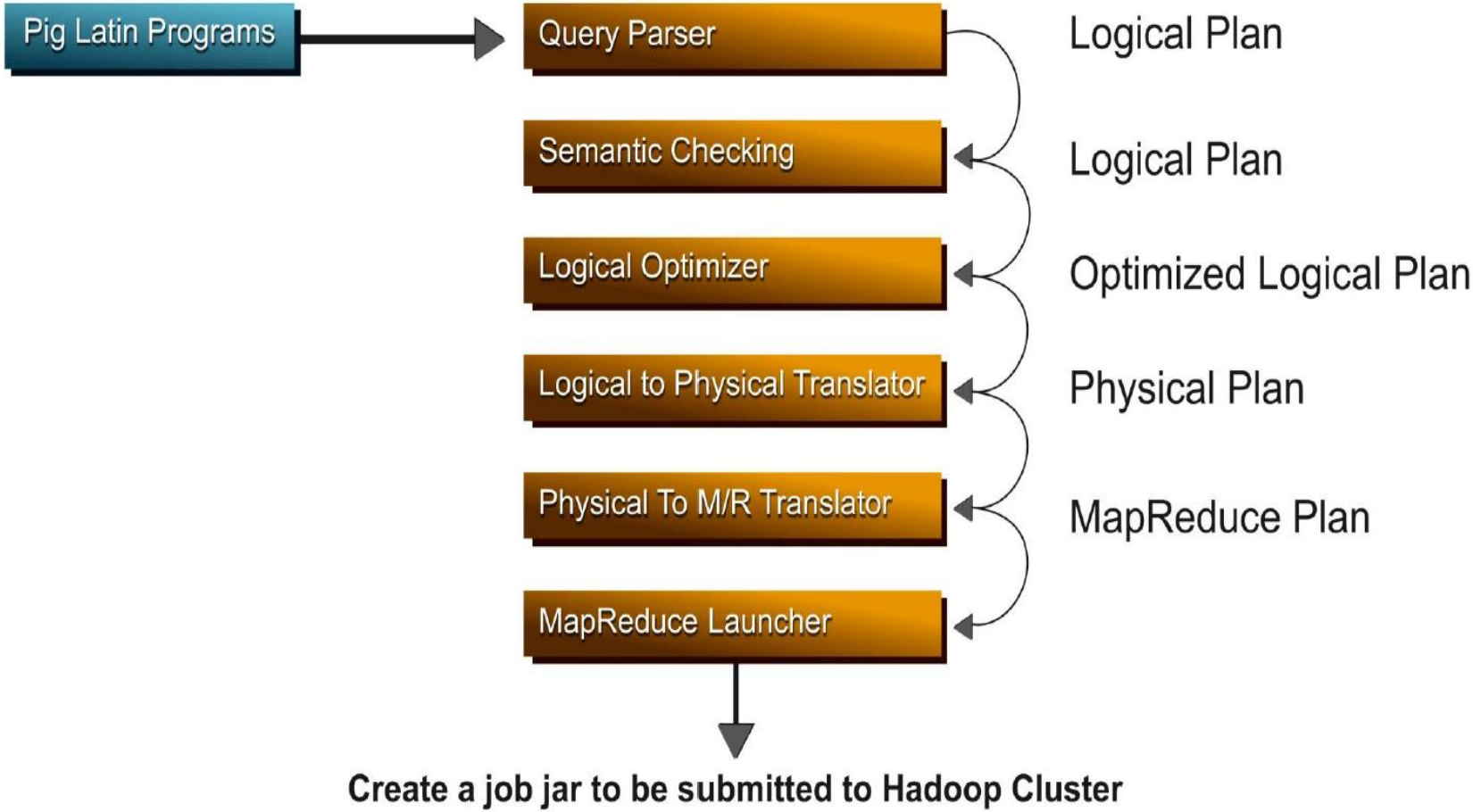
Pig Latin

```
custs = LOAD '/customers' AS (cust_id, region, name);
sales = LOAD '/sales' AS (sale_id, cust_id, amount);
Sales_INDIA = FILTER custs by Region =='India';
joined = JOIN custs BY cust_id, Sales_INDIA by cust_id;
grouped = GROUP joined BY cust_id;
summed = FOREACH grouped GENERATE GROUP ,
SUM(joined.sales_INDIA::amount)
rich_custs = FILTER summed BY $1 > 100000
sorted = ORDER  rich_cust BY $1 DESC;
dump sorted < OR STORE its result>
```

Use Cases where PIG is used ….

- ✓ Processing of **Web Logs**
- ✓ **Data processing** for search platforms
- ✓ Support for **Ad Hoc queries** across large datasets.
- ✓ **Quick Prototyping** of algorithms for processing large datasets.

Compilation



Use Case in Health Care

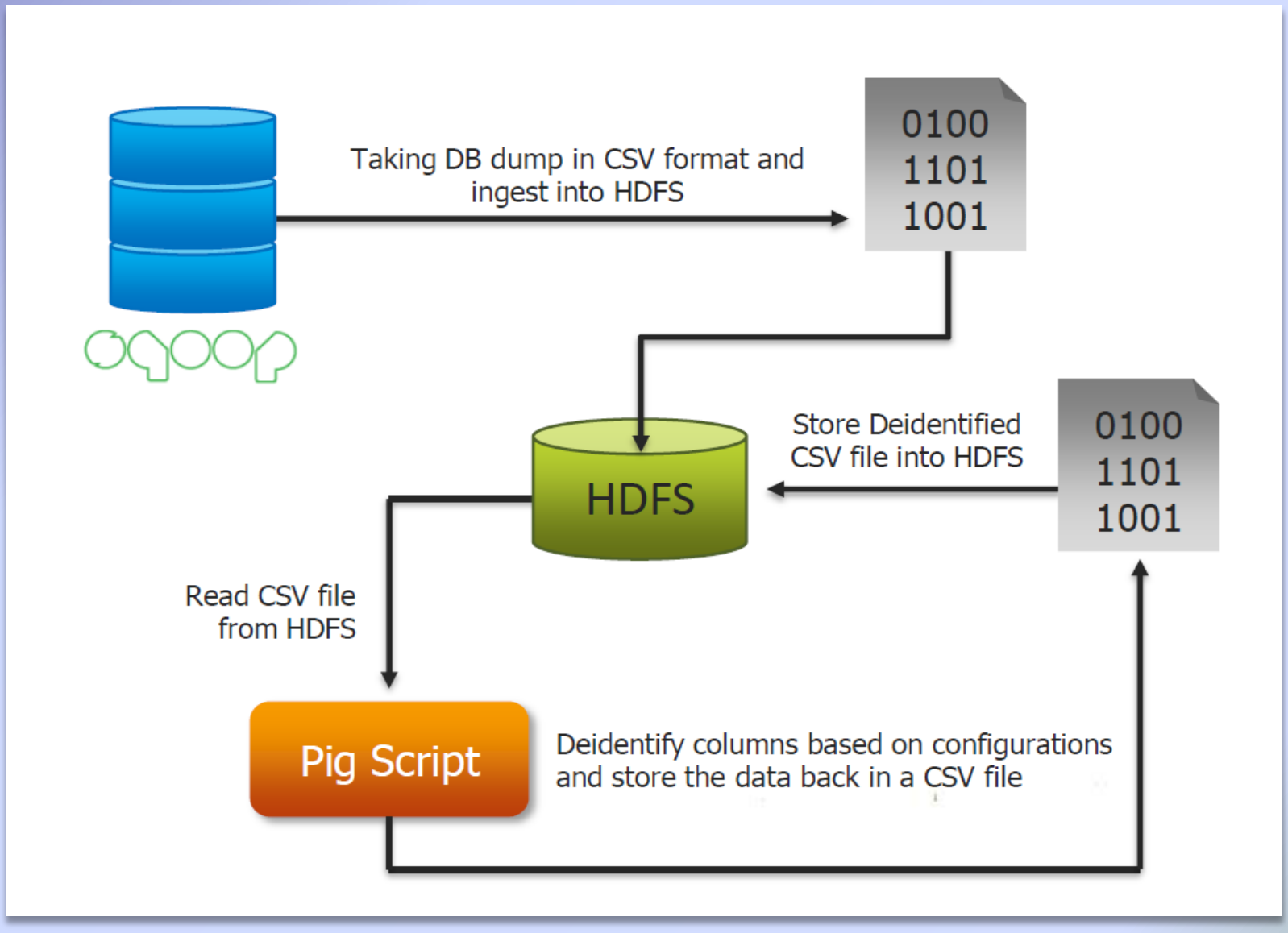
Problem Statement

De-identify personal health information.

Challenges:

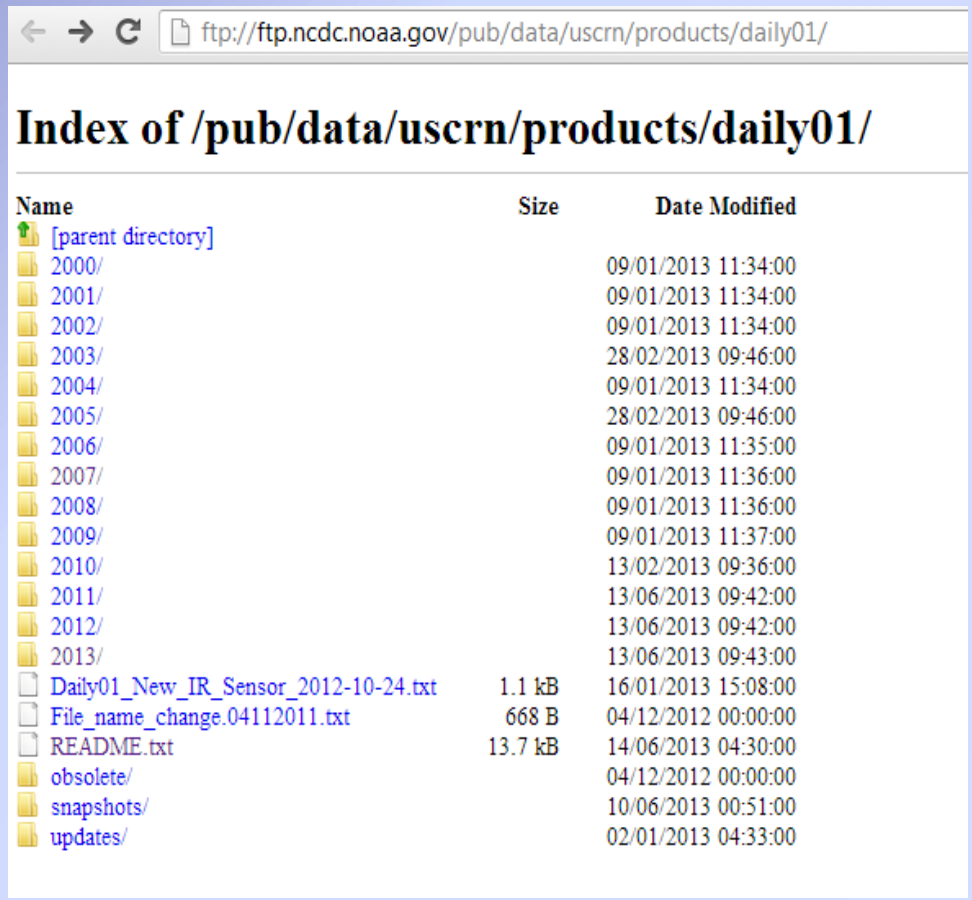
- ✓ Huge amount of data flows into the systems daily and there are multiple data sources that we need to aggregate data from.
- ✓ Crunching this huge data and deidentifying it in a traditional way had problems.

Use Case in Health Care



Weather Data with Pig

ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01/

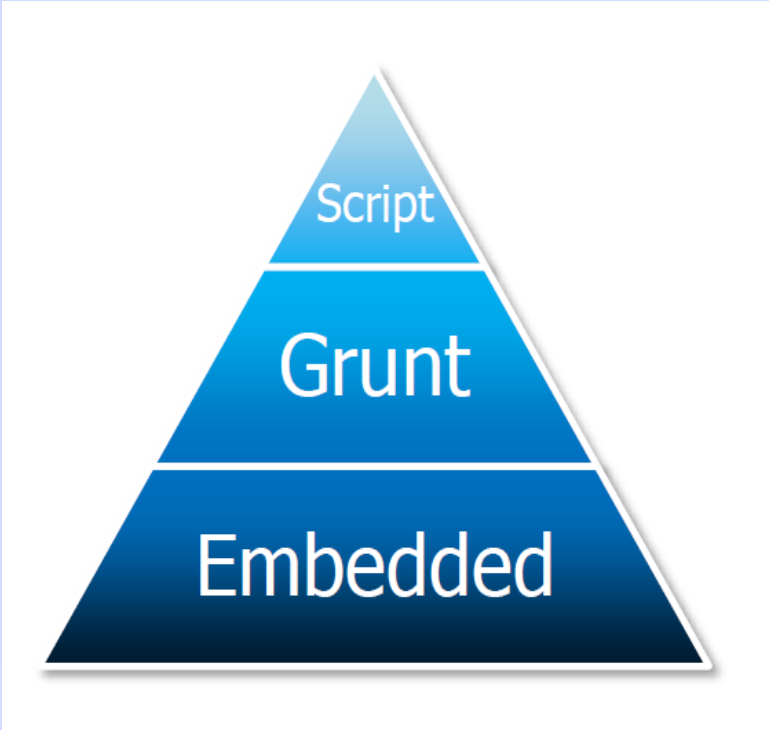


Pig – Basic Program Structure

Script:
Pig can run a script file that contains Pig commands.
Example: pig script.pig runs the commands in the local file script.pig.
ScriptGrunt

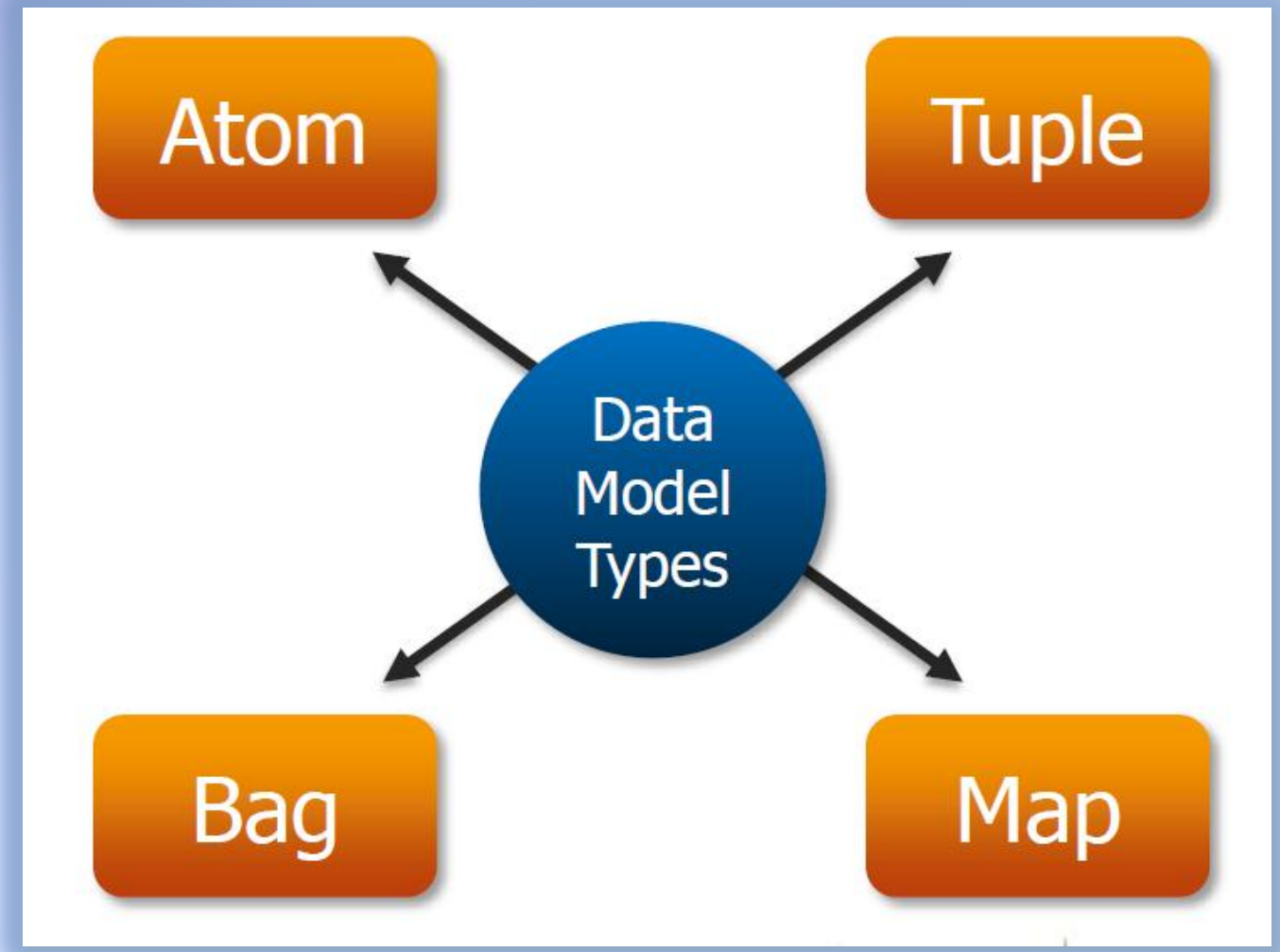
Grunt:
Grunt is an interactive shell for running Pig commands. It is also possible to run Pig scripts from within Grunt using run and exec (execute).

Embedded:
Embedded can run Pig programs from Java, much like you can use JDBC to run SQL programs from Java.



Pig Latin Program

Four Basic Types of Data Models



Data Model

Data Models can be defined as follows:

- ✓ A **bag** is a collection of tuples.
- ✓ A **tuple** is an ordered set of fields.
- ✓ A **field** is a piece of data.
- ✓ A **Data Map** is a map from keys that are string literals to values that can be any data type.

Example:

t = < 1, {<2,3>,<4,6>,<5,7>}, ['apache':'search']>

Pig Data Types

Pig Data Type	Implementing Class
Bag	org.apache.pig.data.DataBag
Tuple	org.apache.pig.data.Tuple
Map	java.util.Map <Object, Object>
Integer	java.lang.Integer
Long	java.lang.Long
Float	java.lang.Float
Double	java.lang.Double
Chararray	java.lang.String
Bytearray	byte[]

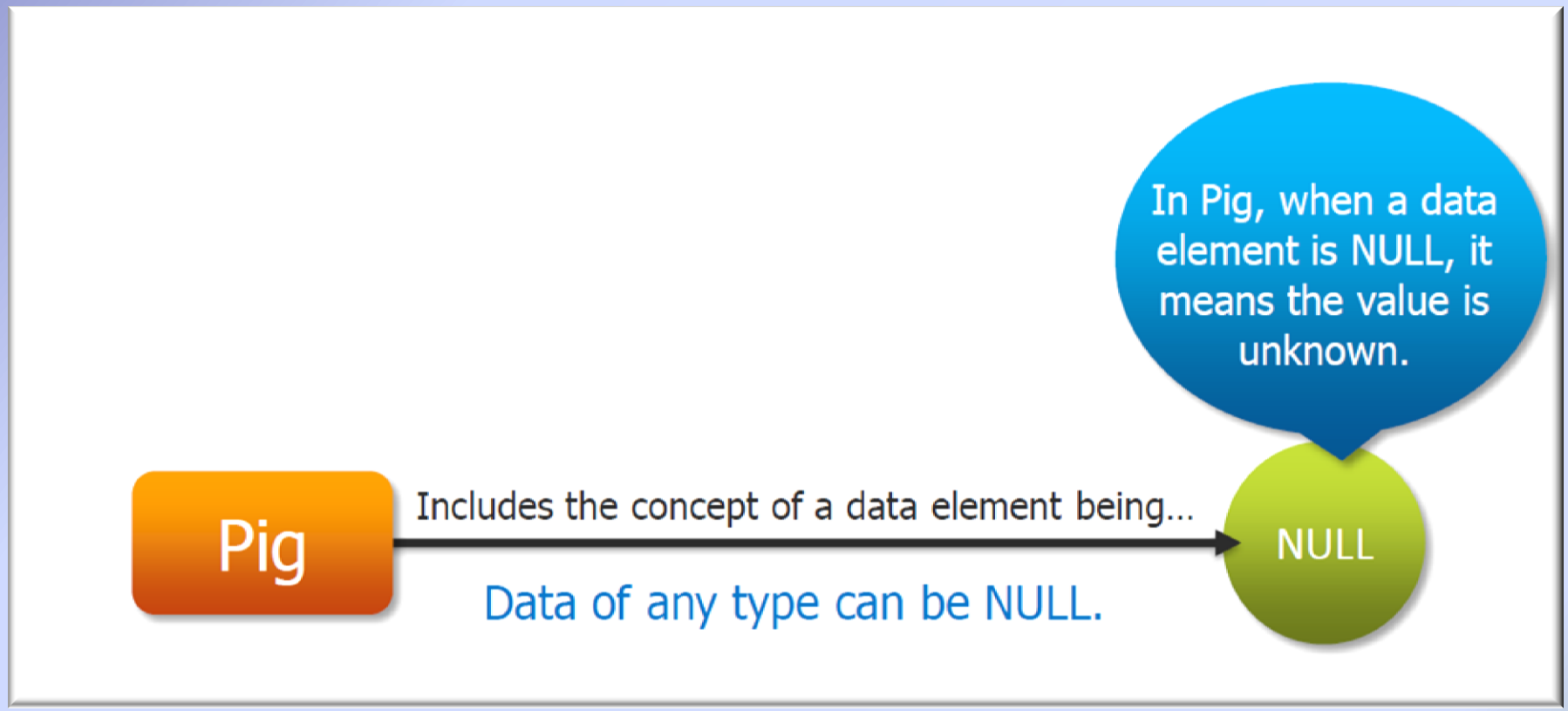
Pig Complex Data Types - Example

t = < 1, {<2,3>,<4,6>,<6,7>}, ['apache':'search']>		
Method	**Example**	**Result**
Position	$0	1
Name	Field2	Bag {<2,3>,<4,6>,<6,7>}
Projection	Field2.$1	Bag {<3>,<6>,<7>}
Function	AVG(field2.$0)	(2+4+6)/3 =4
Conditional	Field1 == 1 ? 'Yes' : 'No'	Yes
Lookup	Field3#'key'	apache

Pig Latin Relational Operators

Category	Operator	Description
Loading and Storing	LOAD STORE DUMP	Loads data from the file system or other storage into a relation . Saves a relation to the file system or other storage. Prints a relation to the console.
Filtering	FILTER DISTINCT FOREACH...GENERATE STREAM	Removes unwanted rows from a relation. Removes duplicate rows from a relation. Adds or removes fields from a relation. Transforms a relation using an external program.
Grouping and Joining	JOIN COGROUP GROUP CROSS	Joins two or more relations. Groups the data in two or more relations. Groups the data in a single relation. Creates the cross product of two or more relations.
Sorting	ORDER LIMIT	Sorts a relation by one or more fields. Limits the size of a relation to a maximum number of tuples.
Combining and Splitting	UNION SPLIT	Combines two or more relations into one. Splits a relation into two or more relations.

Pig Latin - Nulls



Data

File – Student

Name	Age	GPA
Joe	18	2.5
Sam		3.0
Angel	21	7.9
John	17	9.0
Joe	19	2.9

File – Student Roll

Name	Roll No.
Joe	45
Sam	24
Angel	1
John	12
Joe	19

Pig Latin - Group Operator

Example of GROUP Operator:

```
A = load 'student' as (name:chararray, age:int, gpa:float);
dump A;

(joe,18,2.5)
(sam,,3.0)
(angel,21,7.9)
(john,17,9.0)
(joe,19,2.9)

X = group A by name;
dump X;

(joe,{(joe,18,2.5),(joe,19,2.9)})
(sam,{(sam,,3.0)})
(john,{(john,17,9.0)})
(angel,{(angel,21,7.9)})
```

Pig Latin - CO-Group Operator

Example of COGROUP Operator:

```
A = load 'student' as (name:chararray, age:int,gpa:float);
B = load 'studentRoll' as (name:chararray, rollno:int);

X = cogroup A by name, B by name;
dump X;
```

```
(joe,{(joe,18,2.5),(joe,19,2.9)},{(joe,45),(joe,19)})
(sam,{(sam,,3.0)},{(sam,24)})
(john,{(john,17,9.0)},{(john,12)})
(angel,{(angel,21,7.9)},{(angel,1)})
```

Joins And COGROUP

Example

Suppose we have relations A and B.

```
A = LOAD 'data1' AS (a1:int,a2:int,a3:int);

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

B = LOAD 'data2' AS (b1:int,b2:int);

DUMP B;
(2,4)
(8,9)
(1,3)
(2,7)
(2,9)
(4,6)
(4,9)
```

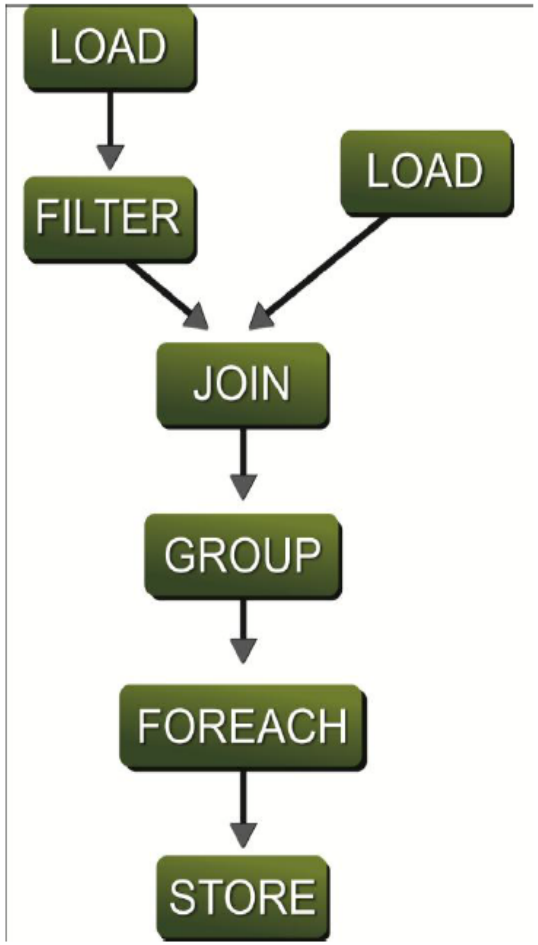
In this example relations A and B are joined by their first fields.

```
X = JOIN A BY a1, B BY b1;

DUMP X;
(1,2,3,1,3)
(4,2,1,4,6)
(4,3,3,4,6)
(4,2,1,4,9)
(4,3,3,4,9)
(8,3,4,8,9)
(8,4,3,8,9)
```

- **JOIN** and **COGROUP** operators perform similar functions.
- **JOIN** creates a flat set of output records while
- **COGROUP** creates a nested set of output records.

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';
```



Union

UNION: To merge the contents of two or more relations.

```
A = LOAD 'data' AS (a1:int,a2:int);

DUMP A;
(1,2)
(4,2)

B = LOAD 'data' AS (b1:int,b2:int);

DUMP B;
(2,4)
(8,9)
(1,3)

X = UNION A, B;

DUMP X;
(2,4)
(8,9)
(1,3)
(1,2)
(4,2)
```

UDFs – User Defined Functions

For logic that cannot be done in Pig

- Can be used to do column transformation, filtering, ordering, custom aggregation
- For example, you want to write custom logic to do interest calculation or penalty calculation Example : 1% of total data set
- **grunt>** interest = FOREACH cust GENERATE custid, calculateIterest(custAcc);

Diagnostic Operators & UDF Statements

Pig Latin Diagnostic Operators

Types of Pig Latin Diagnostic Operators:

DESCRIBE - Prints a relation's schema.
EXPLAIN - Prints the logical and physical plans.
ILLUSTRATE - Shows a sample execution of the logical plan, using a generated subset of the input.

Pig Latin UDF Statements

Types of Pig Latin UDF Statements:

REGISTER - Registers a JAR file with the Pig runtime.
DEFINE - Creates an alias for a UDF, streaming script, or a command specification.

Describe

Use the **DESCRIBE** operator to review the fields and data-types.

```
grunt>
grunt>
grunt> A = load '/data1' as (a1:int, a2:int);
grunt> B = load '/data2' as (b1:int, b2:int);
grunt> X = UNION A, B;
grunt> DESCRIBE X;
X: {a1: int,a2: int}
grunt>
grunt>
```

Pig Latin – File Loaders

Pig Latin File Loaders

BinStorage - "binary" storage

PigStorage - loads and stores data that is delimited by something

TextLoader - loads data line by line (delimited by the newline character)

CSVLoader - Loads CSV files

XML Loader - Loads XML files

Example of Data analysis task

Find users who tend to visit "good" pages:

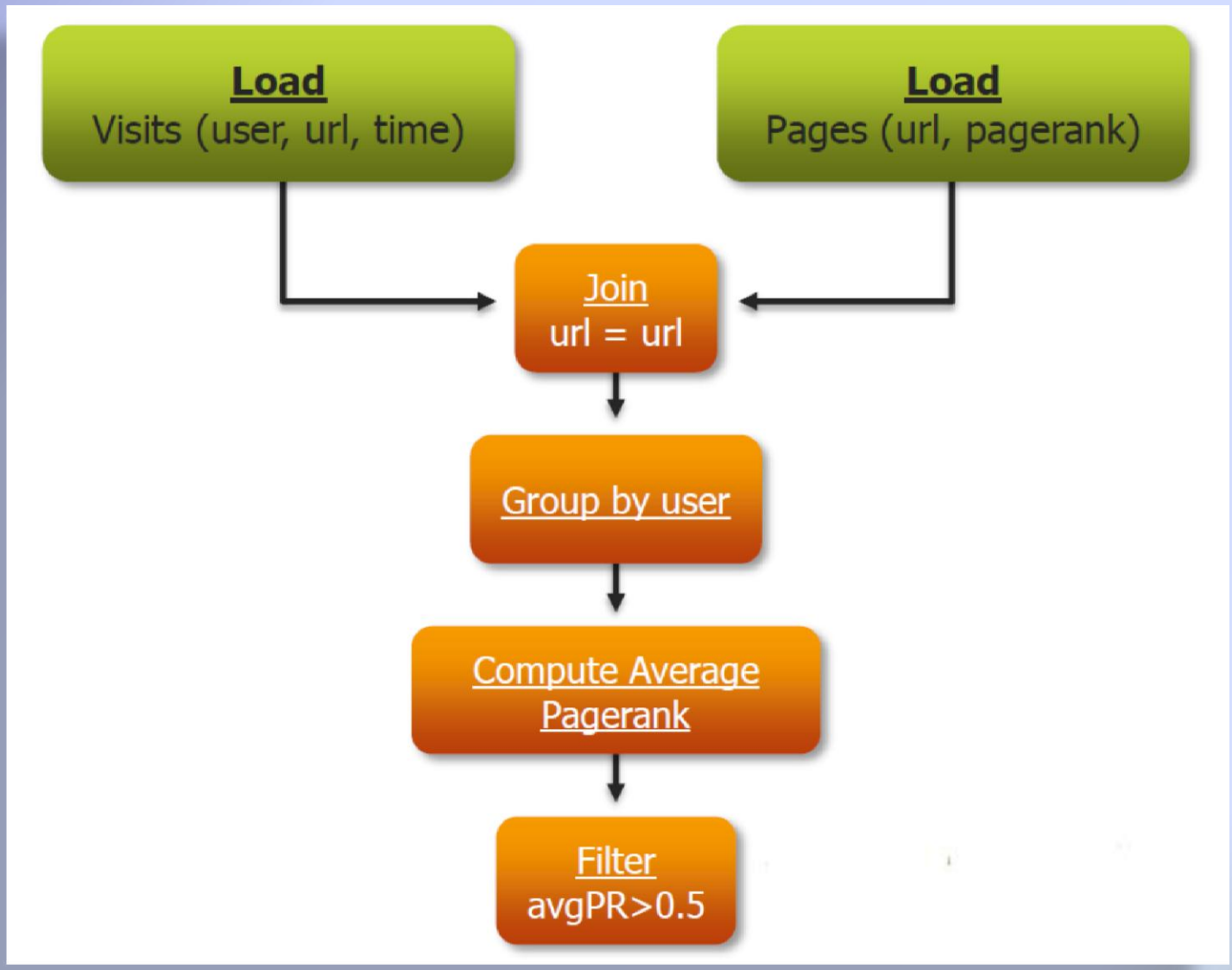
VISITS

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	www.cnn.com	12.00

PAGES

URL	Page Rank
www.cnn.com	0.9
www.flickr.com	0.9
www.myblog.com	0.7
www.crap.com	0.2

Conceptual Data Flow



Pig Latin Script

```
A = load '/pagerank/visits' using PigStorage(',') as user:chararray,url:chararray,time:chararray);

B = load '/pagerank/pages' using PigStorage(',') as (url:chararray,rank:float);

C = Join A by url, B by url;

D = Group C by user;

E = foreach D generate group, AVG(C.rank) as avgpr;

gooduser = filter E by avgpr > 0.1;

store gooduser into '/pagerank/output/result';
```

Q and A ?