PIG: A higher level interface to map/reduce.

Setting Context

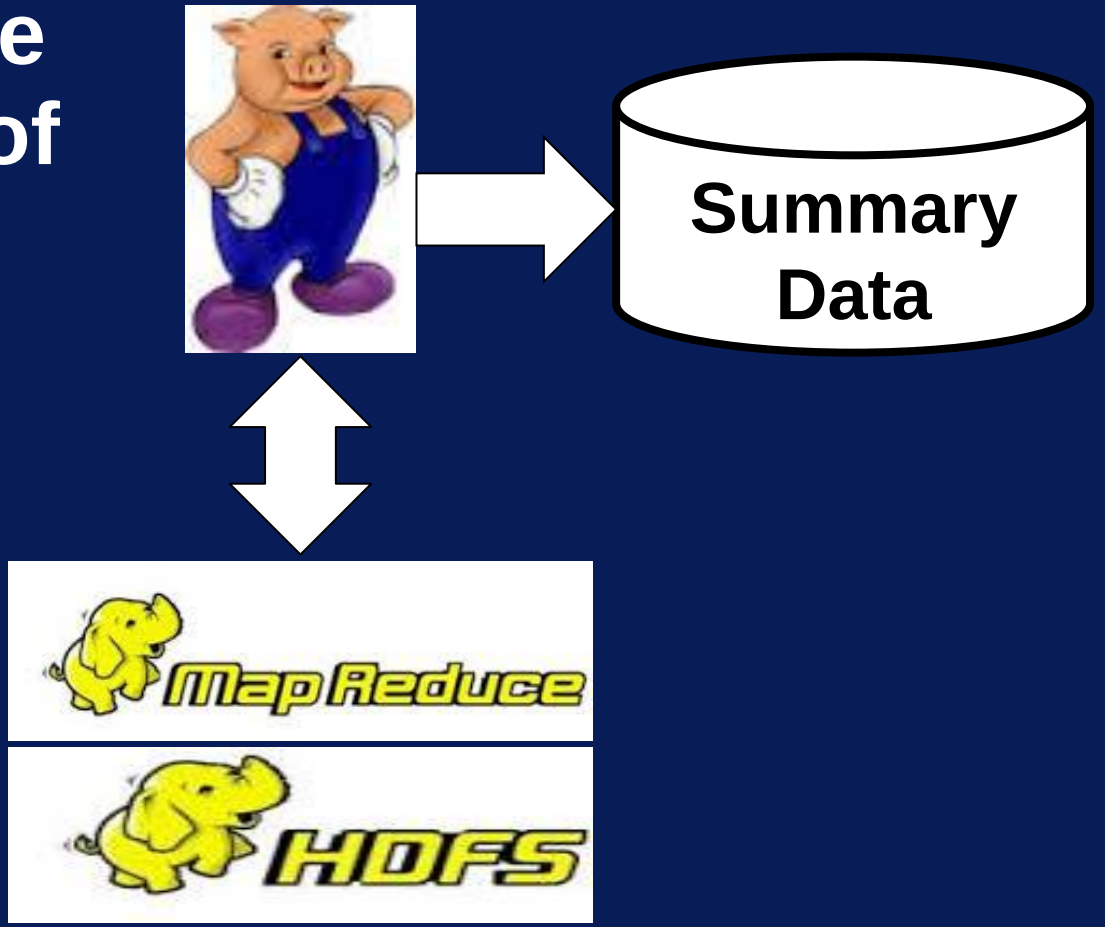
PIG: A higher level interface to map/reduce.

Setting Context

What is Pig

- Engine to script map/reduce jobs
- Started at Yahoo
- Apache Project in 2008

PIG: language layer on top of Hadoop



Pig Features

- scripting language for map/reduce
-

Pig Features

- scripting language for map/reduce
- Performance is close to Python or Java

Pig Features

- scripting language for map/reduce
- Performance is close to Python or Java
- Not for general programming

Pig Strengths

- Rapid data preparation

Pig Strengths

- Rapid data preparation
- Interactive

Great for

data pipelines

analysis with raw data

Pig Strengths

- Rapid data preparation
- Interactive

Great for

data pipelines

analysis with raw data

Pig Scripting

- User specifies data flow

- SQL:
  - Select…sum()
  - From …
  - Where …
  - Group …
- Pig data flow:
  - Load data …
  - Filter …
  - Group …
  - Sum …

- SQL:
  - Select…sum()
  - From …
  - Where …
  - Group …

one statement

- Pig data flow:
  1) Load data …
  2) Filter …
  3) Group …
  4) Sum …

4 statements

Quick look

```
mydata          = LOAD some-HDFS-file …
```

Quick look

```
mydata      = LOAD some-HDFS-file …

mysubset    = FILTER mydata BY name MATCHES ‘…’
```

Quick look

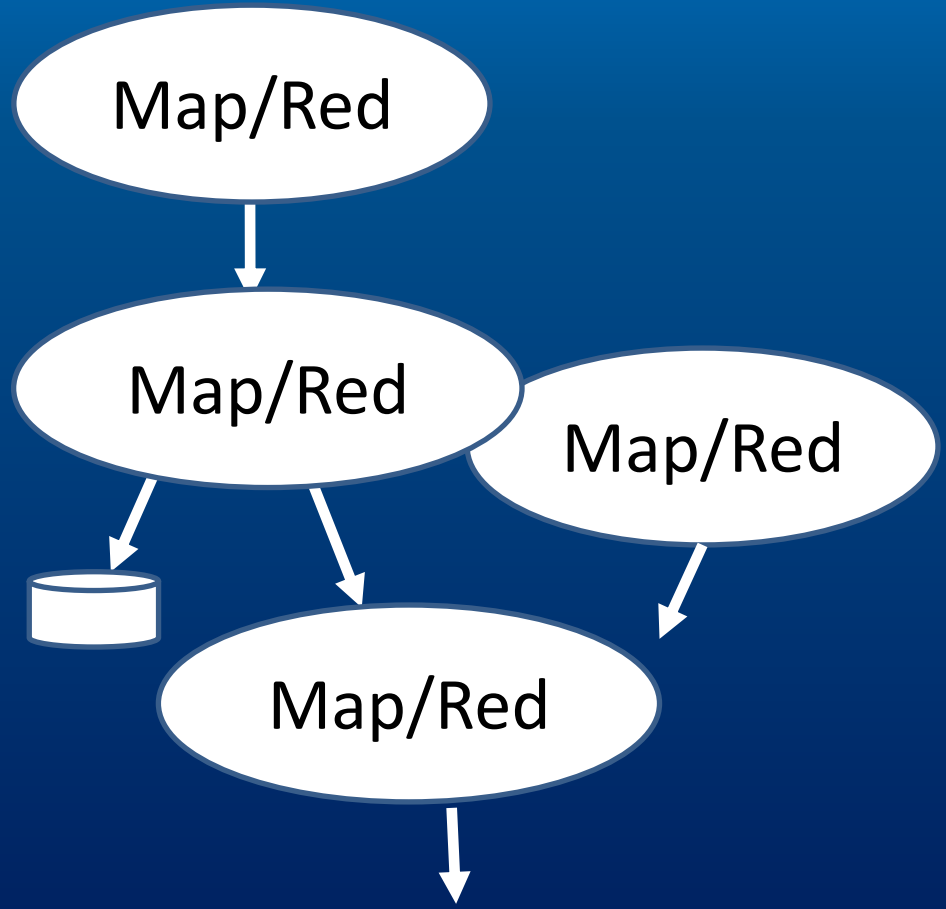
```
mydata      = LOAD some-HdFS-file …

mysubset    = FILTER mydata BY field_name MATCHES ‘…’

mynewdata   = FOREACH mysubset
                    GENERATE field_name,
```

Pig data flow => Map/Reduce

Load
Filter
Foreach
Store
Load
Group
Join …



- In Summary:

PIG is something between SQL and Java or Python

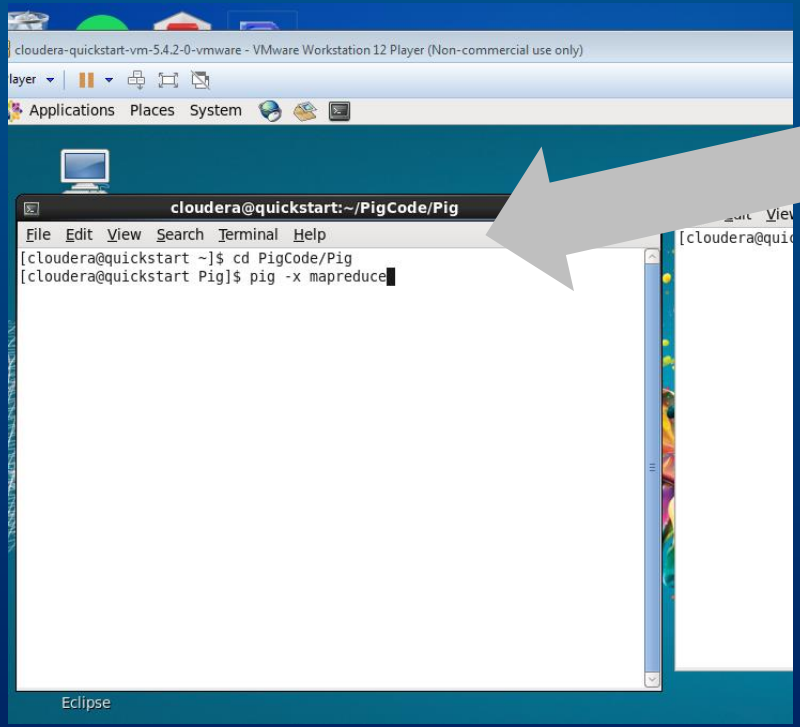
A Pig Session

Executing Word Counting

Pig Session

- Interactive shell – ‘grunt’
- Scripting language – Pig Latin

Pig Session



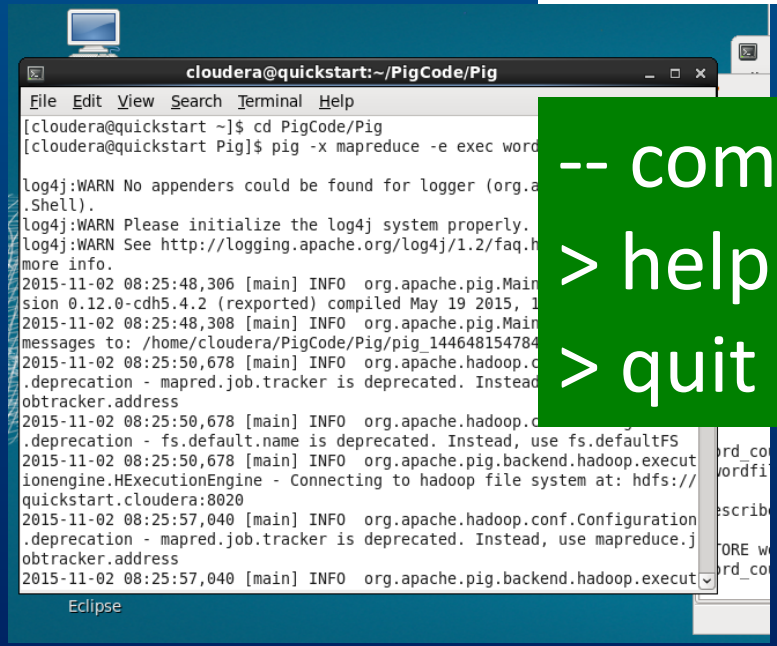
pig –x mapreduce

or

pig –x local

Pig's interactive shell starts …

grunt >



-- comment

> help

> quit or ctrl-C

Pig Latin

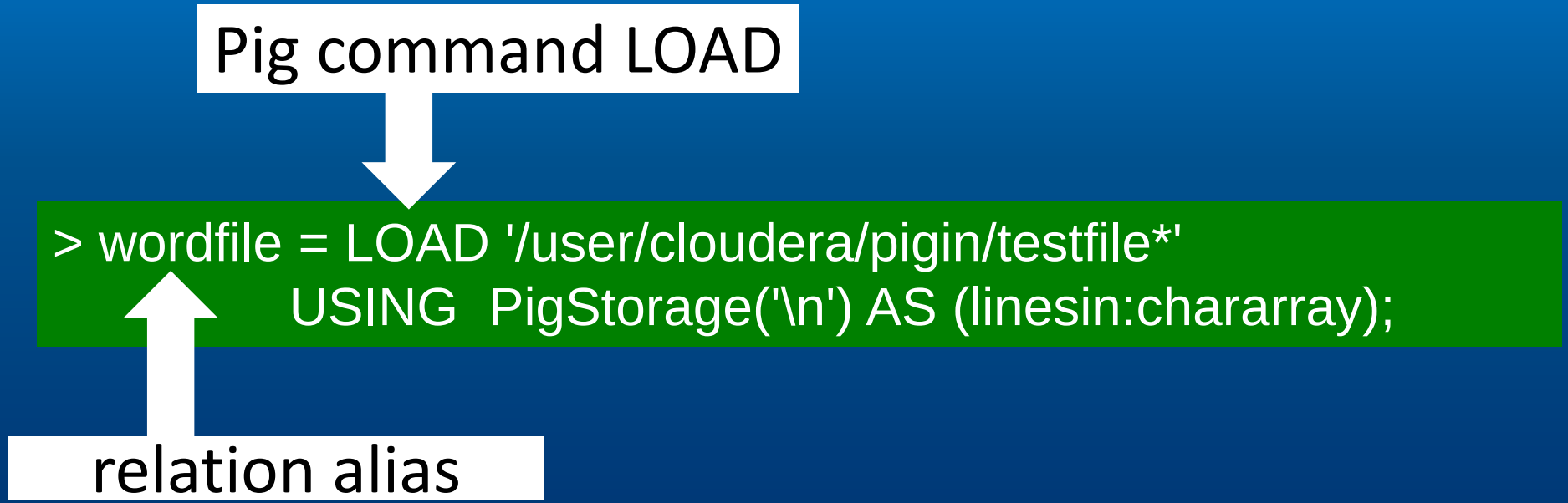
Given 2 datasets in HDFS

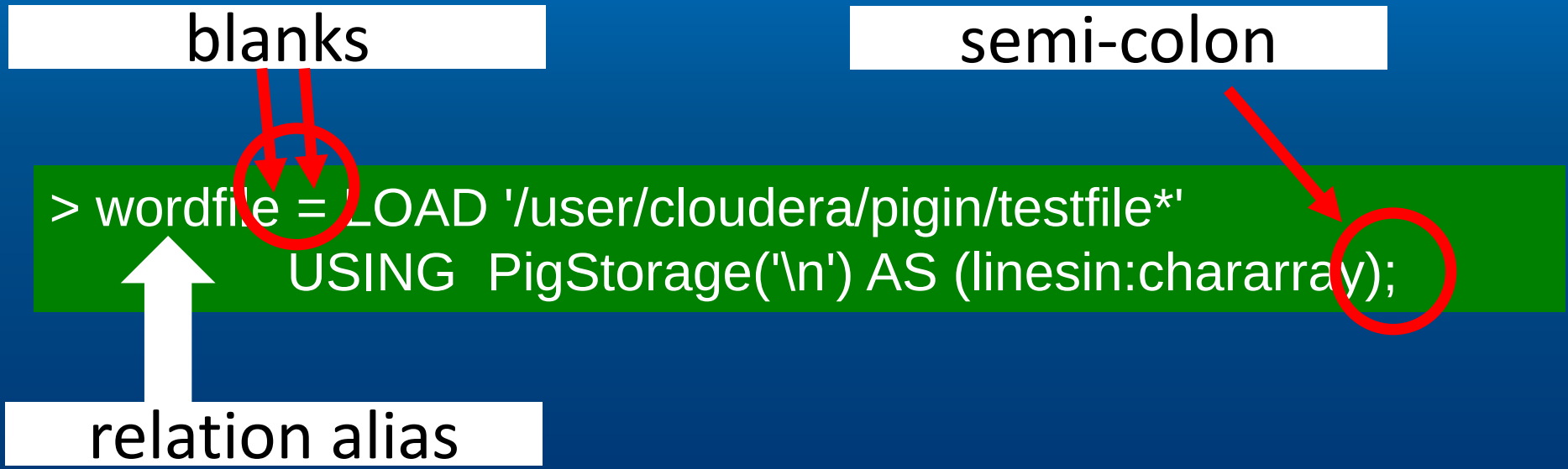
testfile1: A long time ago in a galaxy far far away
testfile2: Another episode of Star Wars

Pig command LOAD

```
> wordfile = LOAD '/user/cloudera/pigin/testfile*'
                USING  PigStorage('\n') AS (linesin:chararray);
```

Note: commands can be entered in ‘grunt’ or as a script file

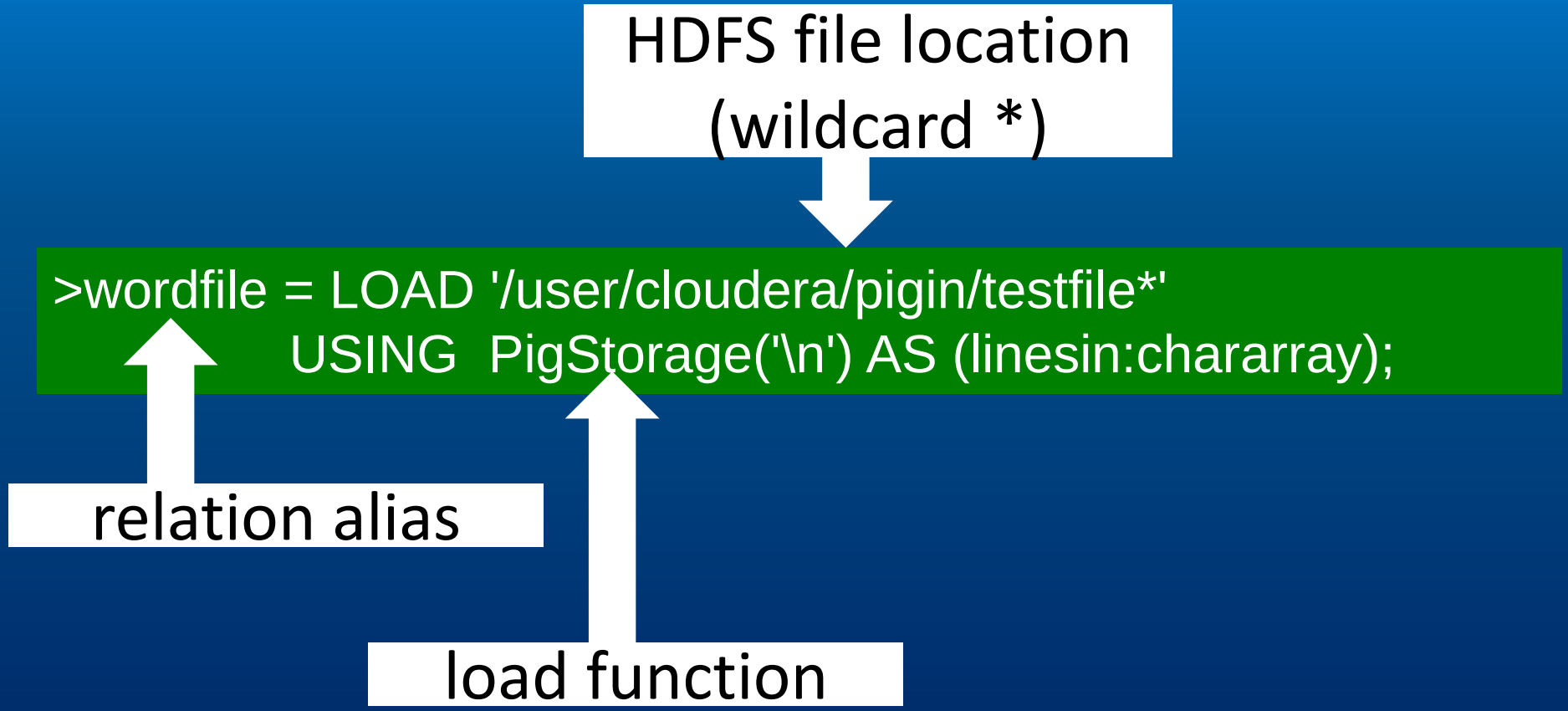

Pig command LOAD
> wordfile = LOAD '/user/cloudera/pigin/testfile*'
USING PigStorage('\n') AS (linesin:chararray);
relation alias

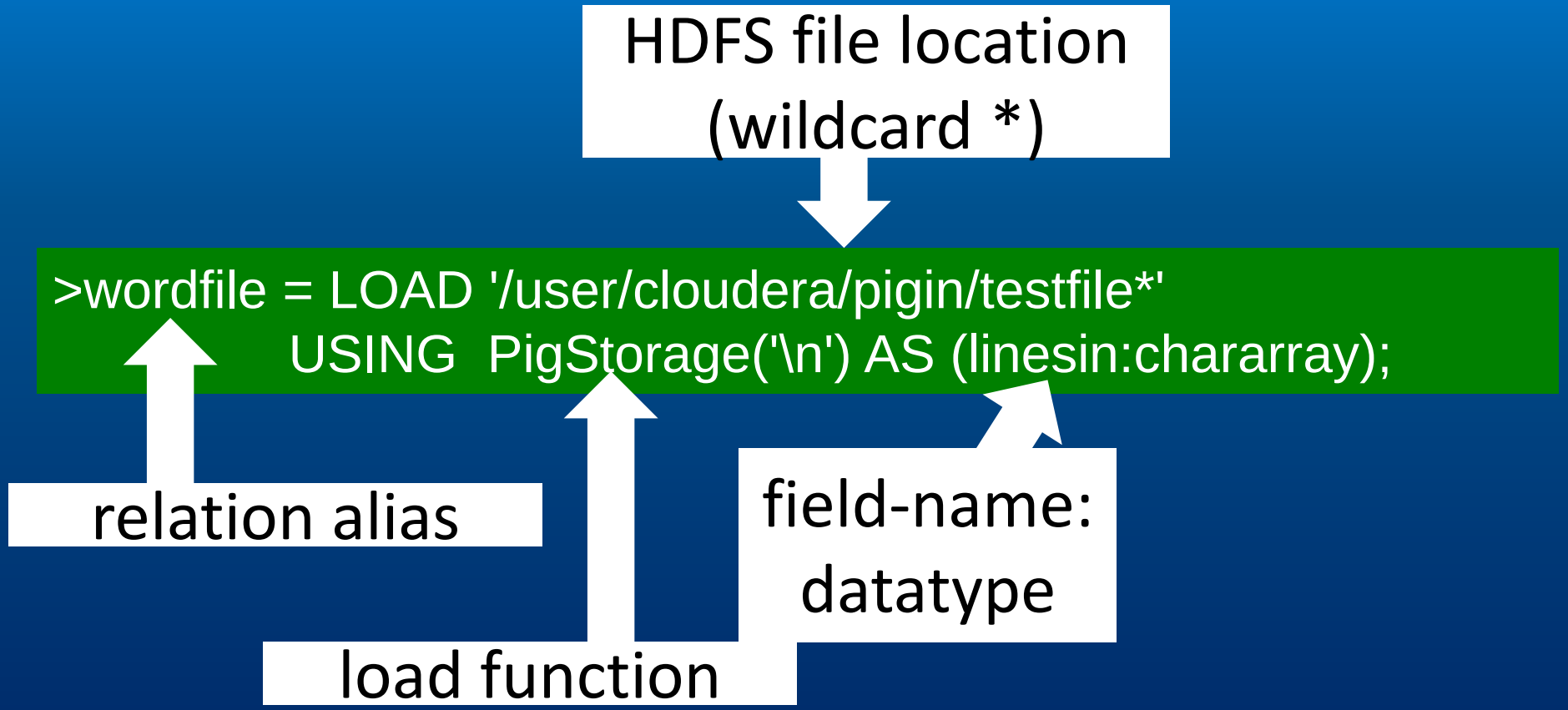

blanks
semi-colon
> wordfile = LOAD '/user/cloudera/pigin/testfile*'
USING PigStorage('\n') AS (linesin:chararray);
relation alias

HDFS file location
(wildcard *)

```
>wordfile = LOAD '/user/cloudera/pigin/testfile*'
        USING  PigStorage('\n') AS (linesin:chararray);
```

relation alias


HDFS file location
(wildcard *)
>wordfile = LOAD '/user/cloudera/pigin/testfile*'
USING PigStorage('\n') AS (linesin:chararray);
relation alias
load function


HDFS file location
(wildcard *)
>wordfile = LOAD '/user/cloudera/pigin/testfile*'
USING PigStorage('\n') AS (linesin:chararray);
relation alias
load function
field-name:
datatype

Note on Pig

- Schema information options

field name:data type

… AS (my_string:chararray, my_integer: int);

Note on Pig

- Schema information options

field name:data type

```
… AS (my_string:chararray, my_integer: int);
```

field name only leaves *bytearray*

```
… AS (my_string, my_integer);
```

Note on Pig

- Schema information options

field name:data type

```
… AS (my_string:chararray, my_integer: int);
```

field name only leaves *bytearray*

```
… AS (my_string, my_integer);
```

no schema, *$position:bytearray*

```
>wordfile = LOAD '/user/cloudera/pigin/testfile*'
                USING  PigStorage('\n') AS (linesin:chararray);

> DESCRIBE wordfile
```

describe the relation

```
>wordfile = LOAD '/user/cloudera/pigin/testfile*'
                USING  PigStorage('\n') AS (linesin:chararray);

> DESCRIBE wordfile

wordfile:  {linesin:chararry}
```

{ } indicates ‘bag’ datatype

Note on Pig

- Complex data types and their brackets

Note on Pig

- Complex data types and their brackets

tuple ( ) - ordered collection of fields

Note on Pig

- Complex data types and their brackets

tuple ( ) - ordered collection of fields

bag { } - unordered collection of tuples

Note on Pig

- Complex data types and their brackets

tuple ( ) - ordered collection of fields

bag { } - unordered collection of tuples

map [key#value] - dictionary

```
>wordfile = LOAD '/user/cloudera/pigin/testfile*'
                USING  PigStorage('\n') AS (linesin:chararray);

> DESCRIBE wordfile

wordfile:  {linesin:chararry}
```

{ } indicates ‘bag’ datatype

```
>wordfile = LOAD '/user/cloudera/pigin/testfile*'
                USING  PigStorage('\n') AS (linesin:chararray);

>tempfile = LIMIT wordfile 10;
>DUMP tempfile
```

take a few records

Dump data to screen

```
>wordfile = LOAD '/user/cloudera/pigin/testfile*'
                USING  PigStorage('\n') AS (linesin:chararray);

>tempfile = LIMIT wordfile 10;
>DUMP tempfile
     …..  INFO  ..  map/reduce job  ….
(A long time ago in a galaxy far far away)
(Another episode of Star Wars)
```

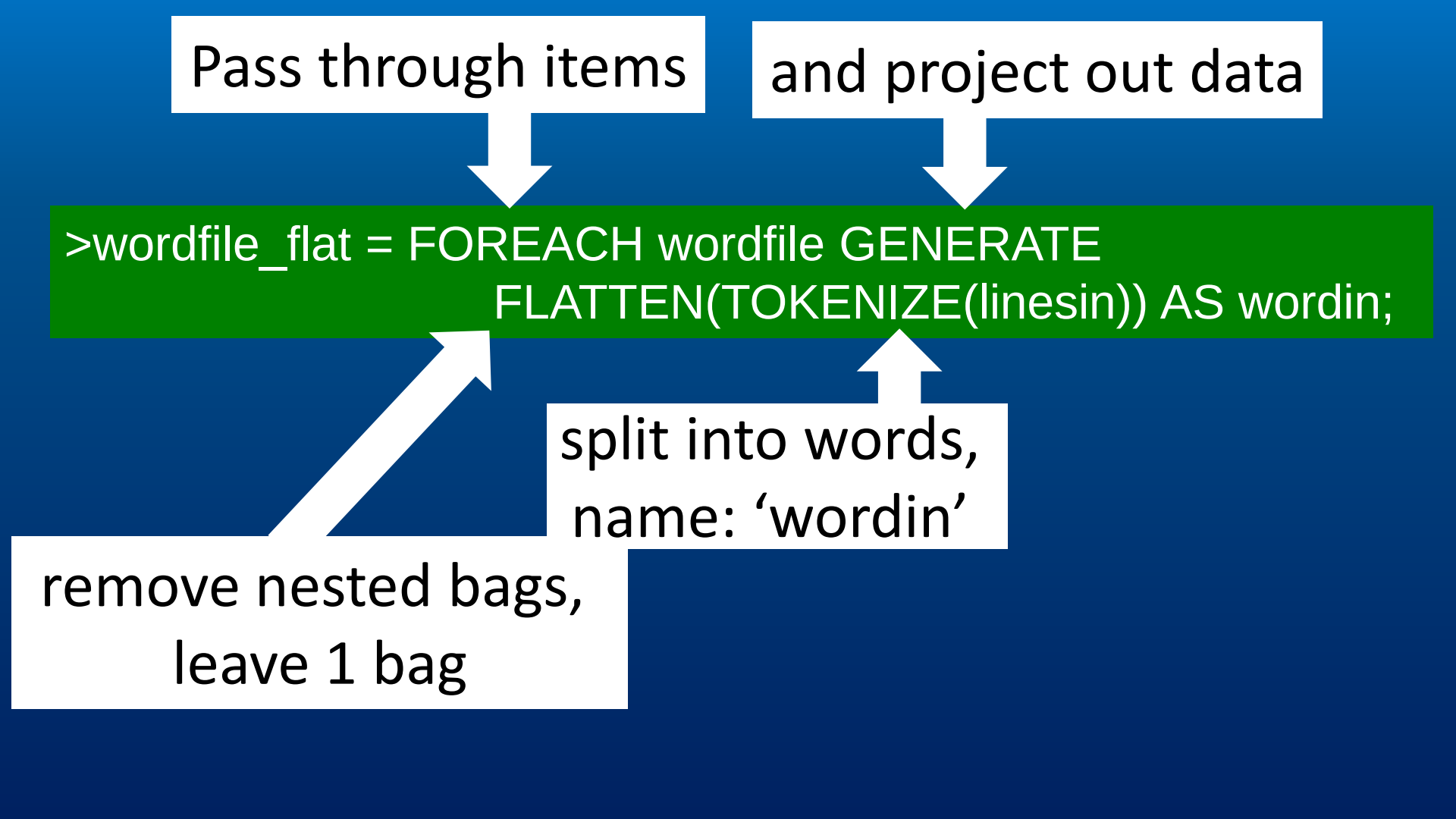
Job info

data

Pass through items

and project out data

```
>wordfile_flat = FOREACH wordfile GENERATE
                        FLATTEN(TOKENIZE(linesin)) AS wordin;
```


Pass through items
and project out data
>wordfile_flat = FOREACH wordfile GENERATE
FLATTEN(TOKENIZE(linesin)) AS wordin;
split into words,
name: ‘wordin’
remove nested bags,
leave 1 bag

```
>wordfile_flat = FOREACH wordfile GENERATE
                        FLATTEN(TOKENIZE(linesin)) as wordin;
> DESCRIBE wordfile_flat
wordfile_flat: {wordin: chararray}
```

The new field name

```
>wordfile_flat = FOREACH wordfile GENERATE
                          FLATTEN(TOKENIZE(linesin)) as wordin;
> tempfile = LIMIT wordfile_flat 10;
> DUMP tempfile
      ….. INFO  map/reduce ….
(A)
(a)
(in)
…
```

GROUP BY

```
>wordfile_grpd = GROUP wordfile_flat BY wordin;
```

In contrast to SQL: no aggregation!

```
>wordfile_grpd = GROUP wordfile_flat BY wordin;
>DESCRIBE wordfile_grpd
wordfile_grpd: {group: chararray,wordfile_flat: {(wordin:
chararray)}}
```

‘group’ field-name

```
>wordfile_grpd = GROUP wordfile_flat BY wordin;
>DESCRIBE wordfile_grpd
wordfile_grpd: {group: chararray,wordfile_flat: {(wordin:
chararray)}}
```

```
>wordfile_grpd = GROUP wordfile_flat BY wordin;
>DUMP wordfile_grpd
…
(of,{(of)})
(ago,{(ago)})
(far,{(far),(far)})
(Star,{(Star)})
…
```

group, bag of (words)

```
>word_counts = FOREACH wordfile_grpd GENERATE
                    group, COUNT(wordfile_flat.wordin);
```

Use bag-name.field-name

```
>word_counts = FOREACH wordfile_grpd GENERATE
                        group, COUNT(wordfile_flat.wordin);
> DESCRIBE word_counts
word_counts: {group: chararray,long}
```

No field-name, long datatype

```
>word_counts = FOREACH wordfile_grpd GENERATE
                        group, COUNT(wordfile_flat.wordin);
> STORE word_counts into '/user/cloudera/word_counts.txt';
```

Order (total)

```
> ..  =  ORDER  <relation> BY group;

> ..  = FOREACH <relation> {
          ... = ORDER <bag in relation> BY field-name;
          GENERATE group, COUNT(…);}
```

FOREACH record (i.e. tuple)

do these steps

Presenter

Pig word count summary

- 4 Pig commands
- HDFS commands from Unix or within grunt using ‘fs’

- end

More Pig Commands and Text Analysis

Filter, Join, etc…

Twitter Data

- Twitter has an interface to retrieve data:

Twitter Sample

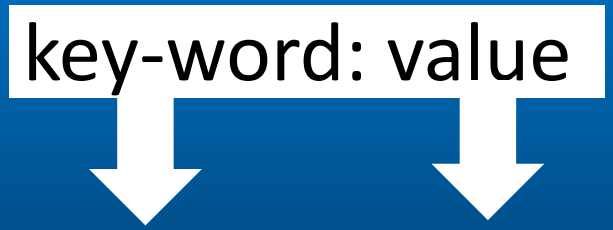
{"created_time": "05:30:51 ",

"text“ :"RT @Madison_McBride: Obama might be a leader but Romney is a businessman. And we need a Buisnessman to flip the economy around.",

"user_id“ : 358343417,

"id“ : 253728871977984000,

"created_date“ : "Thu Oct 04 2012"}



JSON format (like a dictionary)

{"created_time": "05:30:51 ",

"text“ :"RT @Madison_McBride: Obama might be a leader but Romney is a businessman. And we need a Buisnessman to flip the economy around.",

"user_id“ : 358343417,

"id“ : 253728871977984000,

"created_date“ : "Thu Oct 04 2012"}

Trending Analysis

- Count words for a date

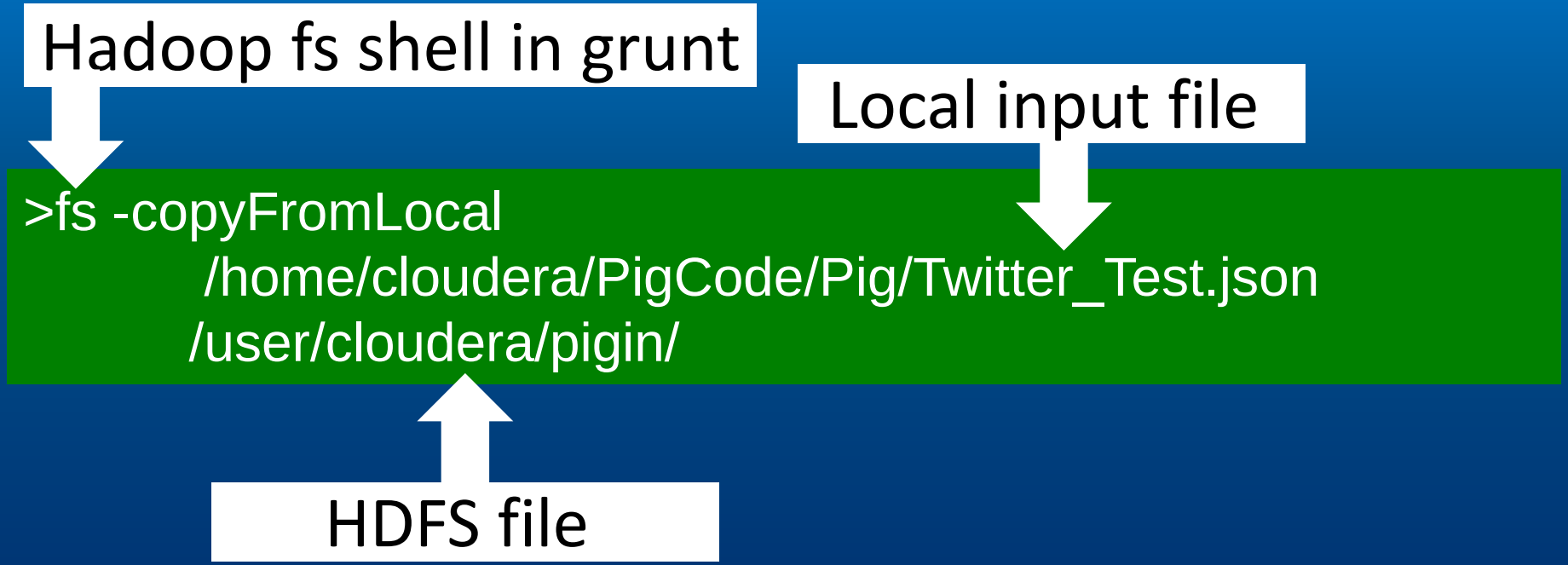
Which words appear more than expected?

Using Pig

- Introduce commands as we go

Task Steps

1. Load data


Hadoop fs shell in grunt
Local input file
>fs -copyFromLocal
/home/cloudera/PigCode/Pig/Twitter_Test.json
/user/cloudera/pigin/
HDFS file

```
>twitter = LOAD '/user/cloudera/pigin/Twitter_Test.json' USING
JsonLoader('created_time:chararray,
            text:chararray,
            user_id:chararray,
            id:chararray,
            created_date:chararray');
```

A load function to read JSON

fields:data-types are parameters

```
>twitter = LOAD '/user/cloudera/pigin/Twitter_Test.json' USING
JsonLoader('created_time:chararray,
            text:chararray,
            user_id:chararray,
            id:chararray,
            created_date:chararray');
```

A load function to read JSON

fields:data-types are parameters

```
>twitter = LOAD '/user/cloudera/pigin/Twitter_Test.json' USING
JsonLoader('created_time:chararray,
            text:chararray,
            user_id:chararray,
            id:chararray,
            created_date:chararray');
```

A load function to read JSON

Special loaders are possible!

Task Steps

2. Sample and Filter data

   Get 1 day word counts

Sample records at this rate

```
>twt_samp = SAMPLE twitter 0.1;
```

Note: sample size is approximately rate X size

Filter data by one date

```
>twt_d1 = FILTER twt_samp BY created_date MATCHES 'Fri Oct 05 2012';
```

Filter data by one date

```
>twt_d1 = FILTER twt_samp BY created_date MATCHES 'Fri Oct 05 2012';
```

Many date-time functions, e.g.

ToDate(created_date,'EEE MMM dd yyyy')

Side Note for Task

- Can we loop over all dates?

Side Note for Task

- Can we loop over all dates?

  No (not like in Python)

Side Note forTask

- Can we loop over all dates?

  No (not like in Python)

- However:

  GROUP BY (date,word)

  then JOIN over word

```
>twt_d1_msgflt = FOREACH twt_d1 GENERATE
FLATTEN(TOKENIZE(text)) as msg_words;
```

Generate a big bag of words

Group same words

```
>twt_d1_msgflt_grpd = GROUP twt_d1_msgflt BY msg_words;
```

```
> twt_d1_wdcnts = FOREACH twt_d1_msgflt_grpd
                  GENERATE group AS word,
                  COUNT(twt_d1_msgflt.msg_words) AS word_cnt;
```

Generate word counts for each group

Task Steps

3. Get baseline word counts

Generate words counts for whole sample

- twt_dall_msgflt = FOREACH twt_samp GENERATE FLATTEN(TOKENIZE(text)) AS …

> … = GROUP … BY …

> … = FOREACH …
GENERATE group
COUNT ()

Same command logic as before

Task Steps

4. Join word counts

Left dataset & key

```
> twt_d1dall_wdcnts_jnd = JOIN twt_d1_wdcnts BY word,
                              twt_dall_wdcnts BY word;
```

Right dataset & key

Left dataset & key

```
> twt_d1dall_wdcnts_jnd = JOIN twt_d1_wdcnts BY word,
                              twt_dall_wdcnts BY word;
```

Right dataset & key

Inner JOIN – only keeps matches

Note on Joins

- Inner Joins: only keeps matches

Note on Joins

- Inner Joins: only keep matches
- Outer Joins: keep ‘LEFT, ‘RIGHT’, or ‘FULL’

```
… = JOIN <left dataset>
        BY <left key> LEFT OUTER,
            <right dataset> ….
```

Note on Joins

- Inner Joins: only keep matches
- Outer Joins: keep ‘LEFT, ‘RIGHT’, or ‘FULL’

```
… = JOIN <left dataset>
        BY <left key> LEFT OUTER,
          <right dataset> ….
```

need schema for other dataset

Join options

- Multiple keys – use ()

```
… BY (<left key 1>, <left key 2>),  ….
```

Join options

- Multiple keys – use tuple ()

```
… BY (<left key 1>, <left key 2>),  ….
```

- Multiple datasets

```
data1 BY  …  data2 BY … data3 BY ...
```

Join performance

- Replicate smaller dataset to all Mappers

```
… JOIN data1 BY key  USING ‘replicated’ …
```

Join performance

- Too many keys (skew)

```
… JOIN data1 BY key  USING ‘skewed’ …
```

Task Steps

- 5. Calculate Expectations

FOREACH joined day &total word counts

```
> twt_d1dall_normed = FOREACH twt_d1dall_wdcnts_jnd {
        obs_freq  = (double) twt_d1_wdcnts::word_cnt;
        exp_freq  = (double) twt_dall_wdcnts::word_cnt;
         lift           = obs_freq-(exp_freq/23);
       GENERATE twt_d1_wdcnts::word,
                        obs_freq, exp_freq, lift;};
```

FOREACH joined day &total word counts

```
> twt_d1dall_normed = FOREACH twt_d1dall_wdcnts_jnd {
        obs_freq  = (double) twt_d1_wdcnts::word_cnt;
        exp_freq  = (double) twt_dall_wdcnts::word_cnt;
         lift            = obs_freq-(exp_freq/23);
      GENERATE twt_d1_wdcnts::word,
                        obs_freq, exp_freq, lift;};
```

cast new fields as double for calculation

```
> twt_d1dall_normed = FOREACH twt_d1dall_wdcnts_jnd {
        obs_freq  = (double) twt_d1_wdcnts::word_cnt;
        exp_freq  = (double) twt_dall_wdcnts::word_cnt;
         lift           = obs_freq-(exp_freq/23);
       GENERATE twt_d1_wdcnts::word,
                        obs_freq, exp_freq, lift;};
```

Note on Pig

- Casting a relation

  get total count as a relation

```
X      = GROUP data ALL;
Xcount = FOREACH  X GENERATE
                  COUNT(field) AS total
```

Note on Pig

- Casting a relation

  get total count as a relation

```
X      = GROUP data ALL;
Xcount = FOREACH  X GENERATE
                  COUNT(field) AS total
```

  use next FOREACH

```
Y  = FOREACH  data {
          x =  …  Xcount.total  ….
```

Results

- Sample output

word	day-freq	total-freq	unexpected
MUST	1.0	3.0	0.869
Mitt	4.0	31.0	2.652
PBS?	1.0	1.0	0.956

- end

Pig Summary

Other Pig Commands

- RANK (order and assign a rank)
- COGROUP (semi-join)
- CUBE & ROLLUP (multiway table)

Pig and Nulls

- Filter with ‘is null’ / ‘is not null’
- Math Operations return null
- Joins don’t match on null
- Group, Cogroup keeps nulls together

Other Pig Commands

- STREAM (call an executable)
- MAPREDUCE (call a map/reduce job)
- PARALLEL (number of reducers)

Pig Scripting Options

- Macros (for reusability)
- Split data (in one mapper, *multiquery*)

Other Pig Functions

- Full suite of Math, String, Date functions

See *pig.apache.org*

Pig UDF

- User Defined Functions

```
> register < piggy bank jar file>
> define  < function_name >

Later use … function_name(myfield) …
```

Pig UDF

- In Unix, install Jar files:

```
> git clone …
> assemble …
```

(See http://datafu.incubator.apache.org/ for details)

Pig UDF

```
> register …/libs/datafu-pig-incubating-1.3.0-SNAPSHOT.jar
```

Pig UDF

```
> register …/libs/datafu-pig-incubating-1.3.0-SNAPSHOT.jar
> define Quantile
 datafu.pig.stats.StreamingQuantile('0.0','0.25‘,'0.50','0.75','1.0‘)
```

Pig UDF

```
> twt_wdcnts_qntls  = FOREACH data_grouped_ALL
      GENERATE Quantile(Obsrved - Expected);
```

the new function

Pig UDF

```
> twt_wdcnts_qntls  = FOREACH data_grouped_ALL
        GENERATE Quantile(Obsrved - Expected);
DUMP twt_wdcnts_qntls
((1.043,1.565,1.782,2.217,5.0))
```

Pig Explain

recall

```
grunt> wordfile_grpd = GROUP wordfile_flat BY wordin;
..
(ago,{(ago)})
(far,{(far),(far)})
…
```

now try

```
grunt> explain wordfile_grpd
```

Pig Explain

```
grunt> explain wordfile_grpd
```


```
# Map Reduce Plan
MapReduce node scope-152
Map Plan
wordfile_grpd: Local Rearrange[tuple]{chararray}(false) - scope-149
|   |
|   Project[chararray][0] - scope-150
|
|---wordfile_flat: New For Each(true)[bag] - scope-146

....
```

Pig Explain

```
grunt> explain wordfile_grpd
```

```
....
 )) - scope-141--------
Reduce Plan
wordfile_grpd: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-151
|
|---wordfile_grpd: Package[tuple]{chararray} - scope-148--------
Global sort: false
----------------
```

Pig Illustrate

```
grunt> illustrate wordfile_grpd
```

```
wordfile_grpd     | group:chararray     | wordfile_flat:bag{:tuple(wordin:chararray)}
-------------------------------------------------------------------------------------------
|                 | A                   | {(A)}                                          |
|                 | a                   | {(a)}                                          |
|
```

…

Pig Stats

- map/reduce job stats

```
> STORE    or DUMP …

Job Stats JobbId
 Maps      Reduces  MaxMapTime          MinMapTIme
1          1          13                  13


 ….
```

Pig Summary

- Easier to program than Java
- More flexible than SQL
- Niche for digesting input

Pig Future

- Still growing in functions available
- Direct graph can run on other execution engines like Spark

End