Data Processing

Toni Verbeiren & Jan Aerts

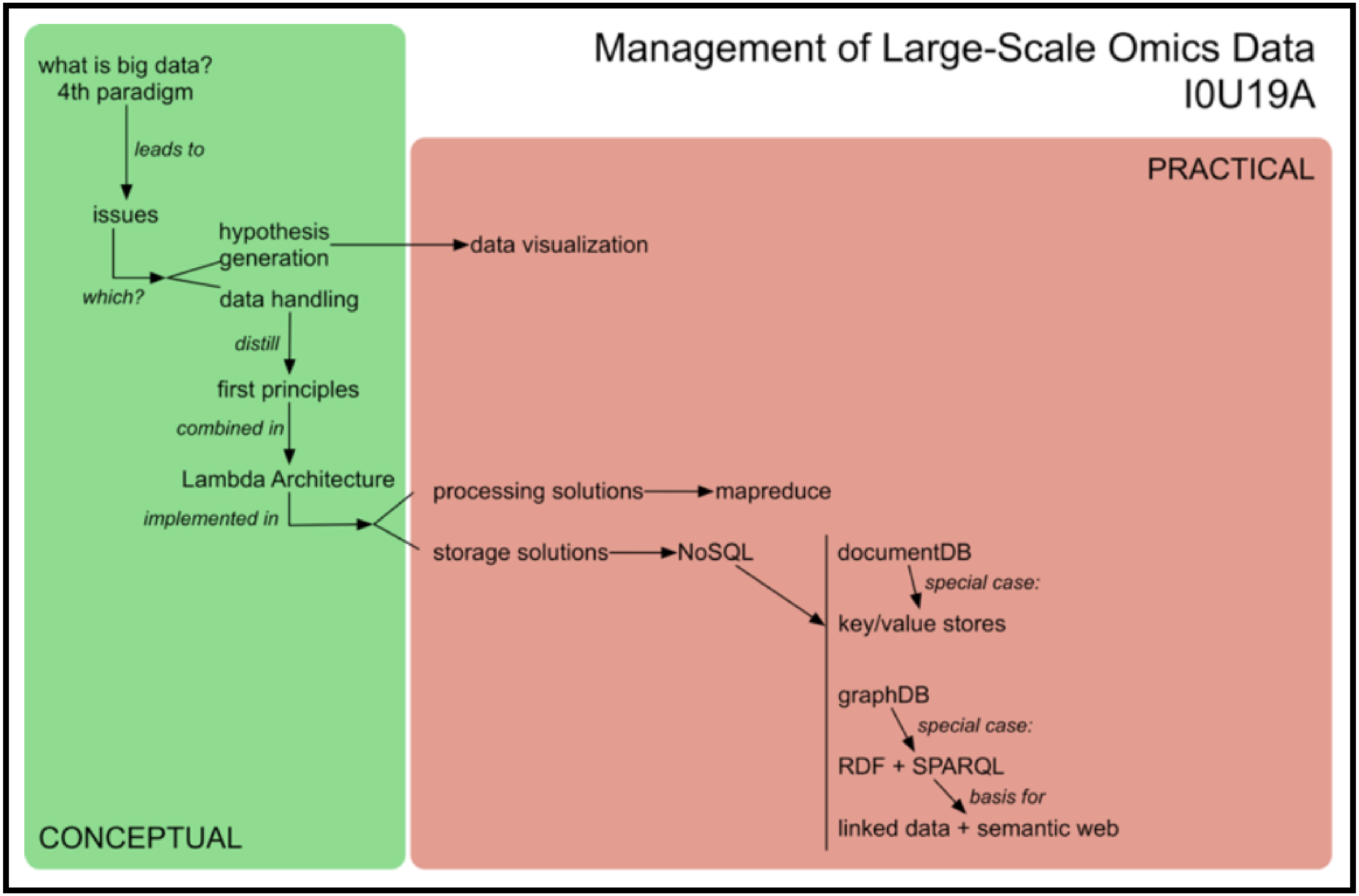
20/3/2014

Introduction

Contents

What this session is about

Processing data, *big* data

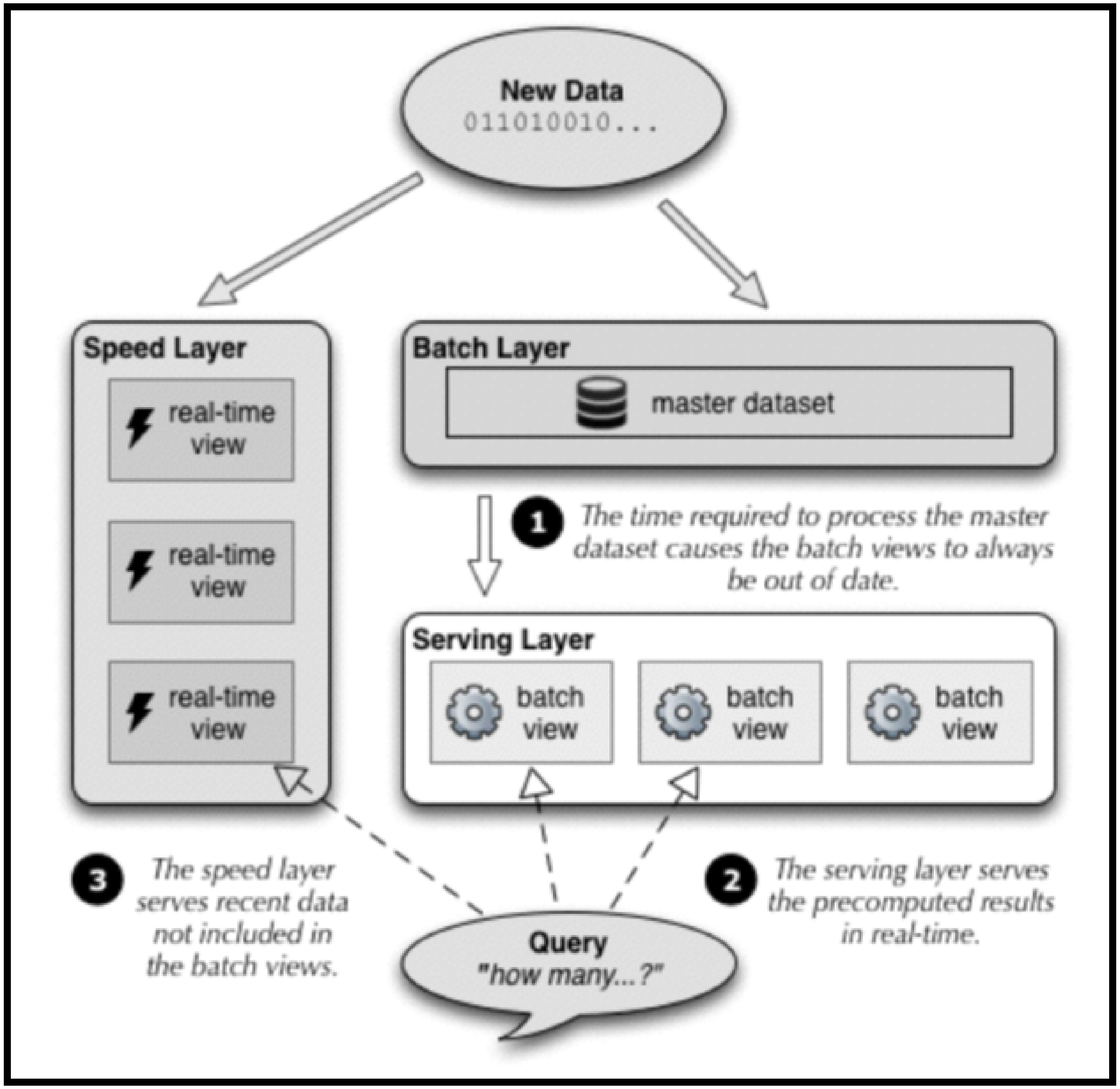


An overview of where we are in the course

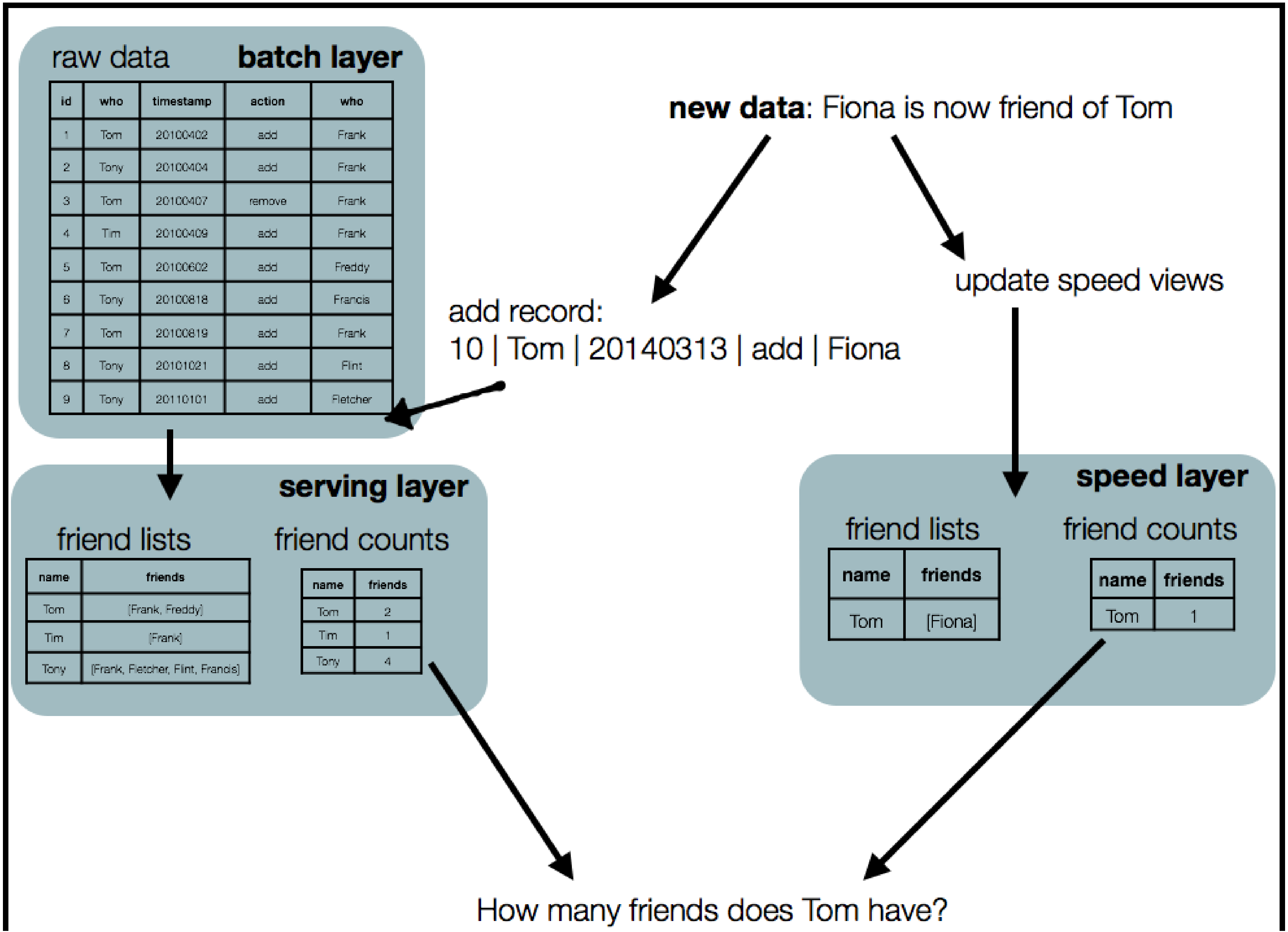
HPC versus HTC

- High Performance Computing:
  - Focus on computation
  - *small* data
  - Parallelism is hard
  - Examples: matrix transformations, large scale simulations, ...
- High Throughput Computing:
  - Focus on volume, throughput
  - *big* data
  - Parallelism is often obvious
  - Examples: finding patterns (genes) in genome, filtering data, ...

How this fits in the whole



Lambda Architecture overview


raw data
batch layer
id | who | timestamp | action | who
1 | Tom | 20100402 | add | Frank
2 | Tony | 20100404 | add | Frank
3 | Tom | 20100407 | remove | Frank
4 | Tim | 20100409 | add | Frank
5 | Tom | 20100602 | add | Freddy
6 | Tony | 20100818 | add | Francis
7 | Tom | 20100819 | add | Frank
8 | Tony | 20101021 | add | Flint
9 | Tony | 20110101 | add | Fletcher
new data: Fiona is now friend of Tom
add record:
10 | Tom | 20140313 | add | Fiona
update speed views
serving layer
friend lists
name | friends
Tom | [Frank, Freddy]
Tim | [Frank]
Tony | [Frank, Fletcher, Flint, Francis]
friend counts
name | friends
Tom | 2
Tim | 1
Tony | 4
speed layer
friend lists
name | friends
Tom | [Fiona]
friend counts
name | friends
Tom | 1
How many friends does Tom have?

Lambda Architecture example

Parallel Word Count

What to count?

Take Ulysses (James Joyce)

- How many occurrences of every word are there?
- Top-10?

ULYSSES

by James Joyce

-- I --

Stately, plump Buck Mulligan came from the stairhead, bearing a bowl of lather on which a mirror and a razor lay crossed. A yellow dressinggown, ungirdled, was sustained gently behind him on the mild morning air. He held the bowl aloft and intoned:

--Introibo ad altare Dei.

Halted, he peered down the dark winding stairs and called out coarsely:

...

http://www.gutenberg.org/ebooks/4300

Traditional approach

```python
#!/usr/bin/python

import sys

wordcount={}

for line in sys.stdin:
  line = line.strip()
  for word in line.split():
    if word not in wordcount:
        wordcount[word] = 1
    else:
        wordcount[word] += 1
for k,v in wordcount.items():
  print k, v
```

Keep a log of the counts !

The top-10 of the words in the text:

```
cat Joyce-Ulysses.txt | wordcount.py | sort -r -k2,2 | head
```

The result:

```
life 99
hands 98
No 97
looked 96
fellow 96
door 96
big 96
them. 95
men 95
thought 94
```

We do not consider special characters, sentence endings, capitals, etc.

What about all works of Shakespeare? Or all books in the library?

Parallel version?

Split up the problems in chunks!

- Words to look for?
- Chunks of text?

```
wordcount={}

runWordCountOnChunk1()
runWordCountOnChunk2()
runWordCountOnChunk3()
```

A mutable data structure is hard to work with in a distributed fashion!

Remember mutable databases?

Functional Programming

What went wrong in the first version?

- Big loop
- Mutable data structure for intermediate results

Underlying issue:

What to do is intermixed with *how* to do it

Functional approach

Ideas:

- Stick to *what* to compute
- Functions take input and produce output without side-effects
- No mutable data structures
- AND: higher-order functions

Examples

A typical implementation of *exponential* in Python:

```
def loopExp(x,n):
    tmp = 1
    for i in range(0,n):
        tmp = tmp * x
    return tmp
```

A Functional alternative:

```
def exp(x, n):
    if n == 0:
        return 1
    else:
        return x * exp(x, n-1)
```

Higher-order functions

Define the following square function:

```
def exp2(x):
    return exp(x,2)
```

We can then apply this function to all elements in a list:

```
>>> map(exp2,[1,2,3,4])
[1, 4, 9, 16]
```

Define the following sum function:

```
def sum(x,y):
   return x + y
```

We can now calculate the sum of all elements in a list:

```
>>> reduce(sum,[1,2,3,4])
10
```

This is where the fun starts:

```
>>> reduce(sum,map(exp2,[1,2,3,4]))
30
```

One more important function:

```
>>> filter(lambda x: x>2 ,[1,2,3,4])
[3, 4]
```

Here, we introduced Lambda expression in Python. The above is the same as:

```
def filter2(x):
    return x>2
filter(filter2 ,[1,2,3,4])
```

What's all the buzz about?

We only described *what* to do, not *how*!

The compiler can fill in the blanks!

MapReduce

Google to the rescue...

Engineers at Google came up with the idea (2003!).

Open Source developers copied the ideas and implemented Hadoop.

Idea

Chain `map` and `reduce` calls.

That's it!

No, it is not…

But it could be...

Situation:

> *A lot of mainstream programming languages do not support Functional Programming in a standard way.*

Think of Java, C, C++, ...

Workaround

The workaround:

> *Make very strict assumptions on what is passed back and forth between `map` and `reduce`.*

Key-Value pairs to the rescue !

But: make sure fault-tolerance is built in...

MapReduce in real-life

Mapper

Each of you gets some lines from Ulysses.

Script:

```
Add a 1 for every occurrence of 'the'
Add a 1 for every occurrence of 'a'
```

Reducer

Script:

```
Sum the total for 'the'
Sum the total for 'a'
```

Hadoop implementation

Java example

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, Interr
```

```java
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

   public void reduce(Text key, Iterable<IntWritable> values, Context context)
     throws IOException, InterruptedException {
       int sum = 0;
       for (IntWritable val : values) {
           sum += val.get();
       }
       context.write(key, new IntWritable(sum));
   }
}
```

```
 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
 }

}
```

Example with Hadoop Streaming

Easy input file

```
> cat easy_file.txt
a b c a b a
```

Initial word count script:

```
> cat easy_file.txt | ./wordcount.py
a 3
c 1
b 2
```

Mapper

```
#!/usr/bin/env python

import sys

for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print '%s\t%s' % (word, 1)
```

```
> cat easy_file.txt | ./mapper.py
a   1
b   1
c   1
a   1
b   1
a   1
```

Reducer

```
#!/usr/bin/env python

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)

    try:
        count = int(count)
    except ValueError:
        continue

    if current_word == word:
        current_count += count
    else:
```

```
> cat easy_file.txt | ./mapper.py | ./reducer.py
a   1
b   1
c   1
a   1
b   1
a   1
```

What happened?

```
#!/usr/bin/env python

from operator import itemgetter
import sys

. . .

for line in sys.stdin:
    . . .

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word
```

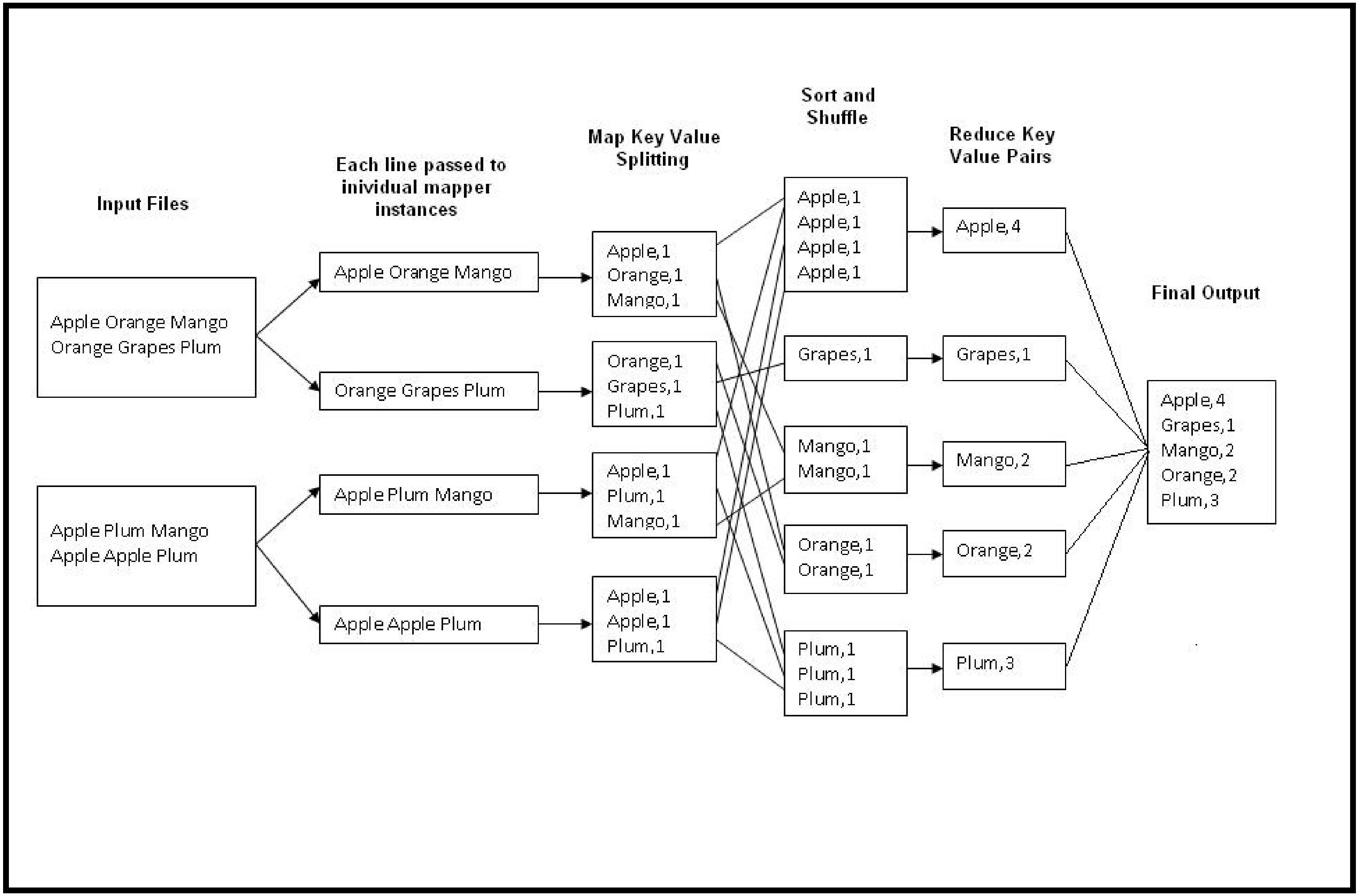
Sorting added:

```
> cat easy_file.txt | ./mapper.py | sort -k 1,1 | ./reducer.py
a   3
b   2
c   1
```

This is bascially what Hadoop does!

Please note: *value* can be scalar, list, data structure, ...

MapReduce with Hadoop



Overview of how word count can be implemented in Hadoop

Using Hadoop streaming

On a Mac:

```
> hadoop jar /usr/local/Cellar/hadoop/1.2.1/libexec/contrib/streaming/hadoop-streaming-1.2.1.
  -file mapper.py -mapper mapper.py \
  -file reducer.py -reducer reducer.py \
  -input Joyce-Ulysses.txt \
  -output output
```

The result is a **folder**:

```
> ls output
_SUCCESS   part-00000
```

Via Hadoop on teaching server:

```
> hadoop jar /usr/lib/hadoop/contrib/streaming/hadoop-streaming-0.20.2-cdh3u6.jar \
  -file mapper.py -mapper mapper.py \
  -file reducer.py -reducer reducer.py \
  -input Joyce-Ulysses.txt \
  -output wc
```

The result is the same.

Distributing the File System

Questions

Some questions:

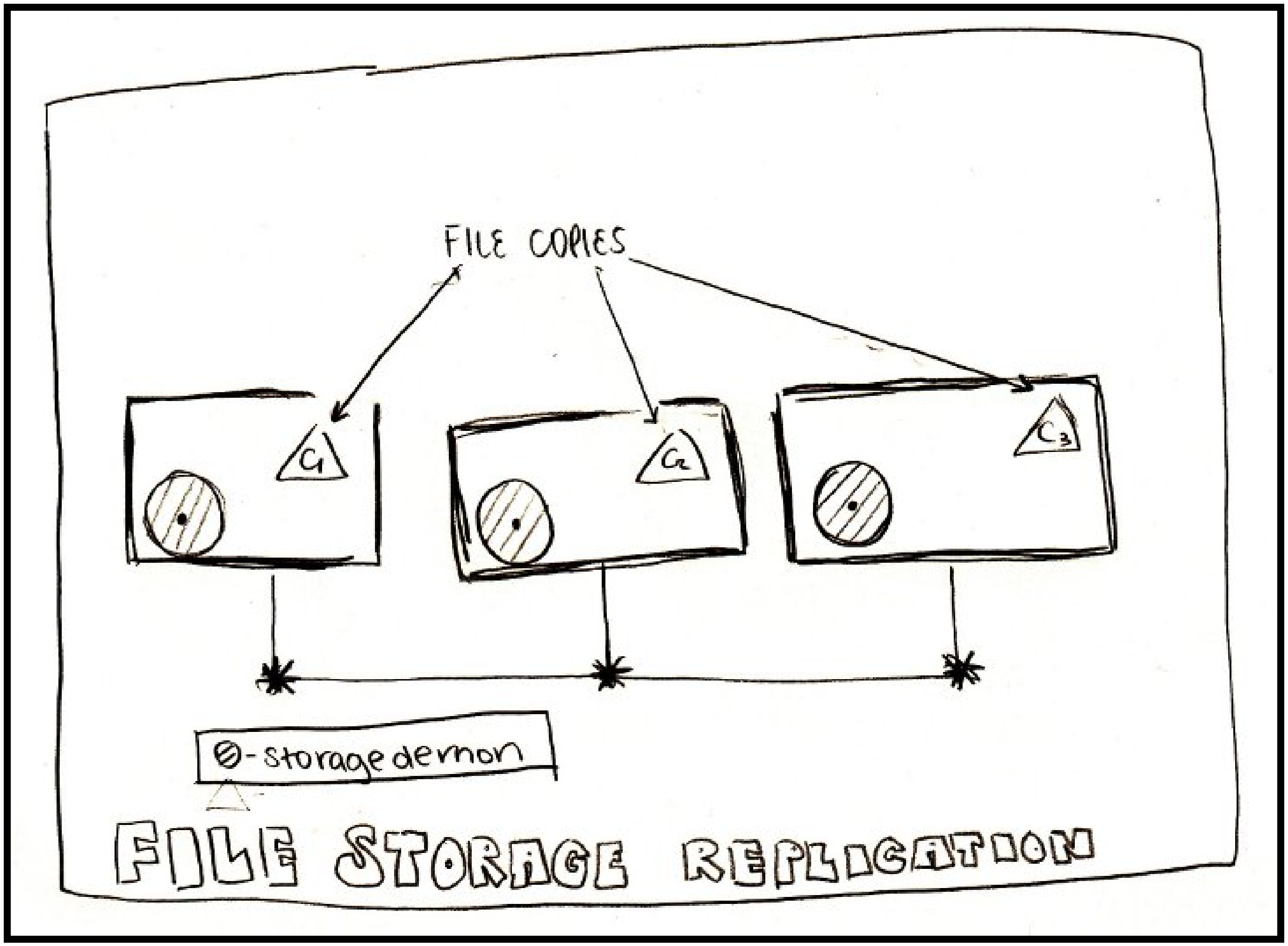
1. What about GBs or TBs or ... of data?
2. What about distributing that using MR?

Distributed FS

Concept:

- Split file in blocks of 64MB
- Distribute blocks accross cluster
- Keep 3 copies for redundancy
- **Computation goes to the data**

A picture ...



Overview of HDFS

Source: http://hadoopilluminated.com/

DFS and MR: Better Together

Traditional processing: Bring data to computation

Big Data: Bring computation to data

Alternatives to Hadoop

Google

Links:

- http://research.google.com/archive/mapreduce.html
- http://research.google.com/archive/gfs.html

Spark

Also Apache product

Based on functional language (Scala).

Example word count in Scala:

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Python interface : `pyspark`

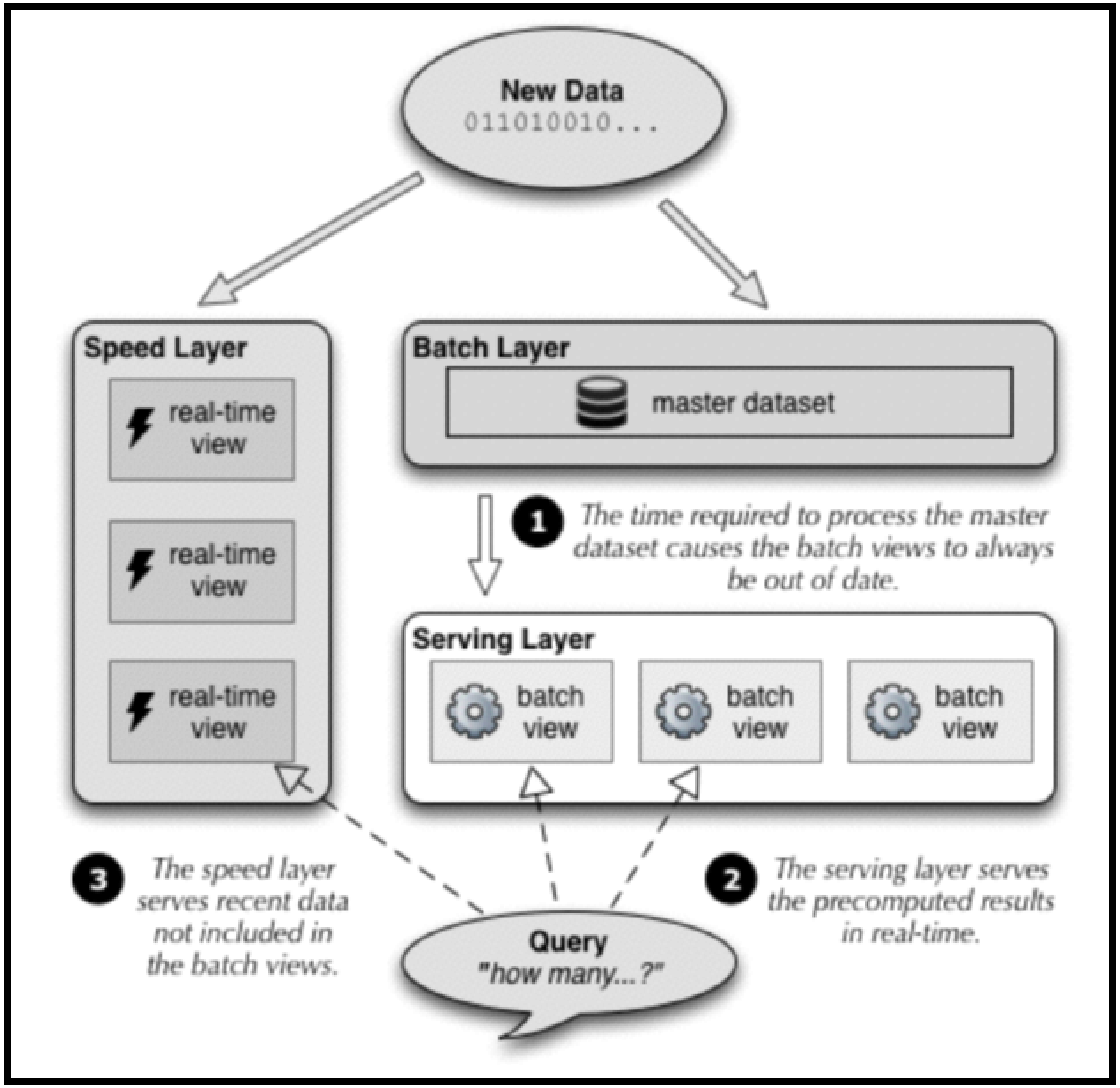
Word count in Python:

```
file = sc.textFile("Joyce-Ulysses.txt")
counts = file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.collect()
```

In order to write the output to a file, replace the last line by:

```
counts.saveAsTextFile("output_file.txt")
```

Streaming data



Lambda Architecture overview

Requires different algorithms and processing

Solutions exist:

- Kafka: manage the queue
- Storm: process the queue

Spark can do it too!

Hadoop ecosystem

See also: http://hadoopecosystemtable.github.io/

Some notable projects/tools

Alternative Languages

Want to use MR, but without *heavy* Java?

- Pig: new *language* (Telenet, Netflix, ...)
- Scalding: implemented in Scala (Twitter, ...)
- Cascalog: implemented in Clojure
- Etc.

Example of Pig word count:

```
a = load '. . .';
b = foreach a generate flatten(TOKENIZE((chararray)$0)) as word;
c = group b by word;
d = foreach c generate COUNT(b), group;
store d into '. . .';
```

Databases on top of Hadoop

- HBase:
  - key/value store on top of Hadoop
  - Based on Google BigTable
- Parquet:
  - Columnar storage
  - Based on ideas from Google Dremel
- Drill:
  - Columnar storage
  - Based on Dremel

SQL support

MR, Spark, Pig, ... not familiar to traditional RDBM experts.

- Hive:
  - SQL on Hadoop,
  - On top of: HDFS, HBase, Parquet, ...
- Shark:
  - SQL on top of Spark

Roundup

	RDBMS	MapReduce
Data size	gigabytes	petabytes
Access	interactive & batch	batch
Updates	Read and write many times	Write once, read many times
Structure	static schema	dynamic schema
Integrity	high	low
Scaling	non-linear	linear

from: Hadoop, The Definitive Guide (T White; O'Reilly Media)

RDBMS versus MapReduce

Links

Some links:

- http://architects.dzone.com/articles/how-hadoop-mapreduce-works
- https://files.ifi.uzh.ch/dbtg/sdbs13/T10.0.pdf
- http://static.googleusercontent.com/media/research.google.com/en//archive/ma
osdi04.pdf