

Hadoop + Python =
Happy

Colin Evans @ PyWebSF

Build Flexible Applications with Graph Data

Programming the

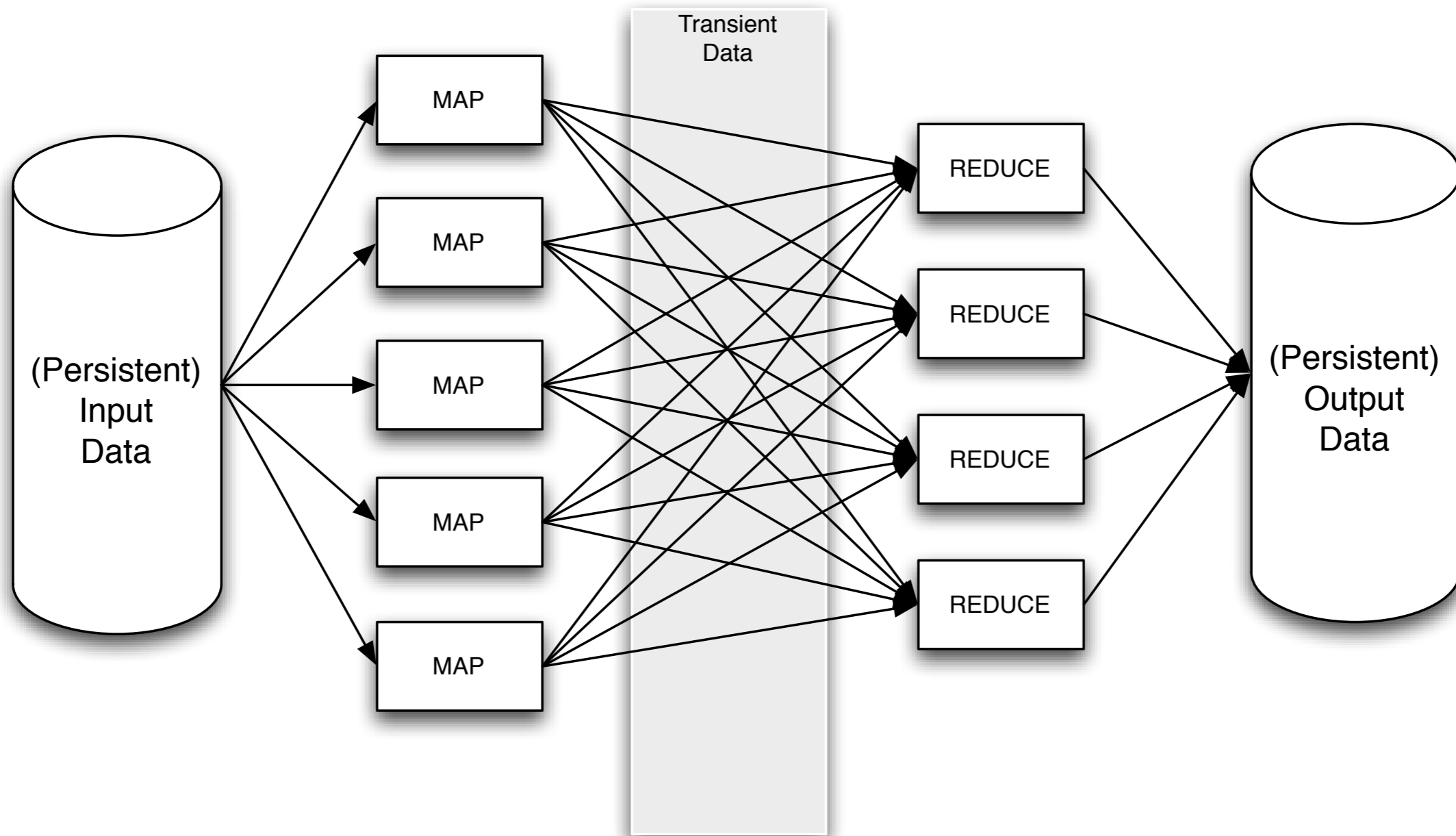
Semantic Web



O'REILLY®

*Toby Segaran,
Colin Evans & Jamie Taylor*

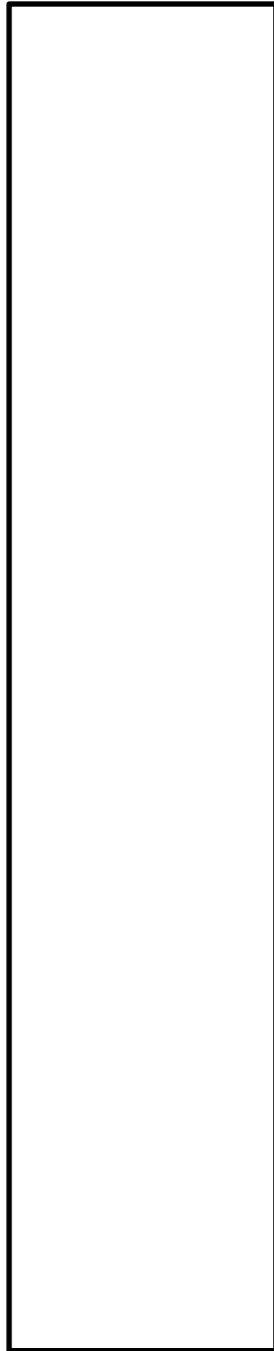
Map/Reduce Overview



Map/Reduce Overview

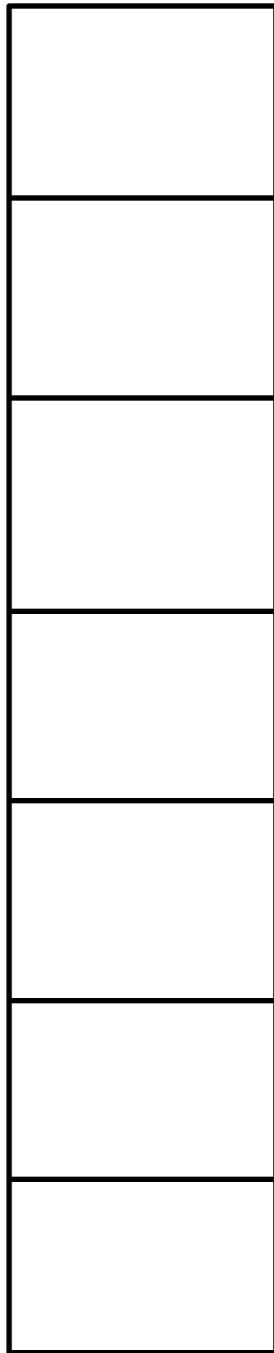
Map/Reduce Overview

Input Data



Map/Reduce Overview

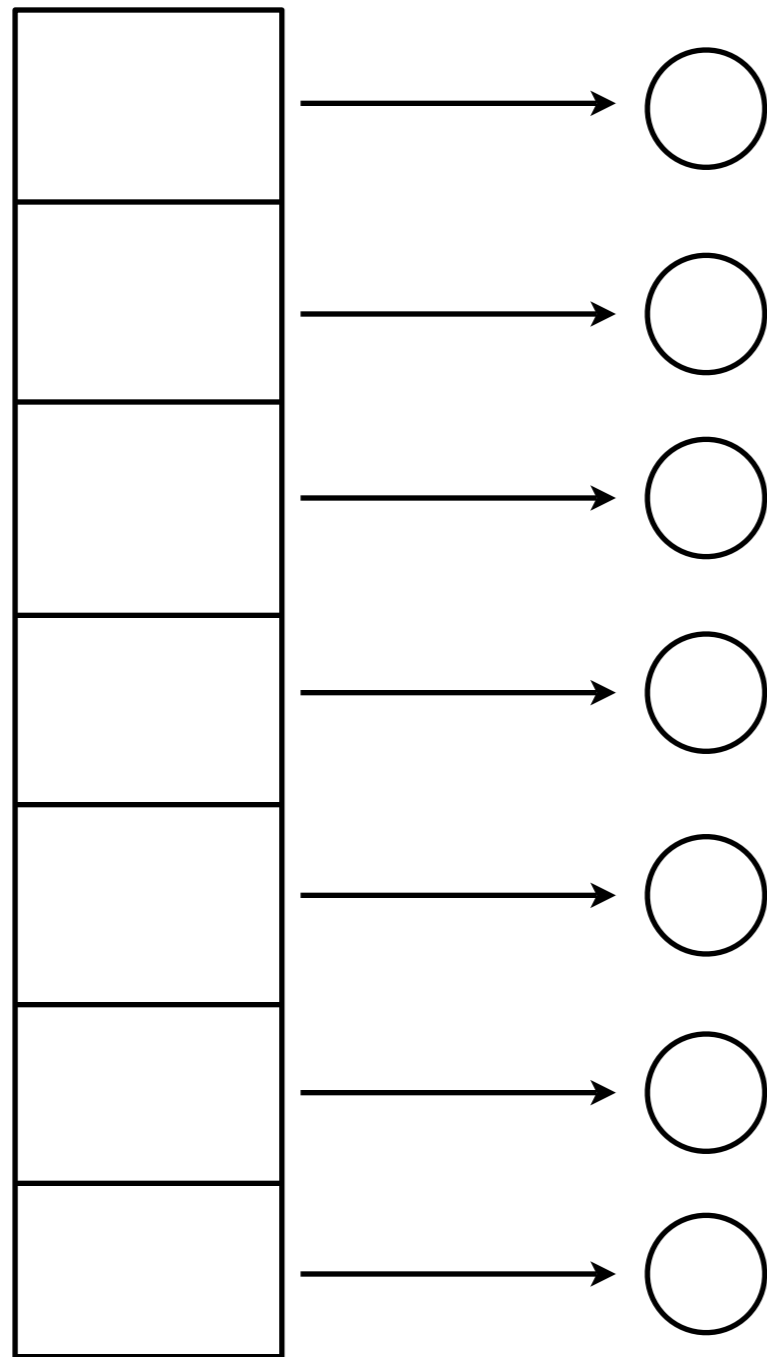
Input Data



Map/Reduce Overview

Input Data

Map

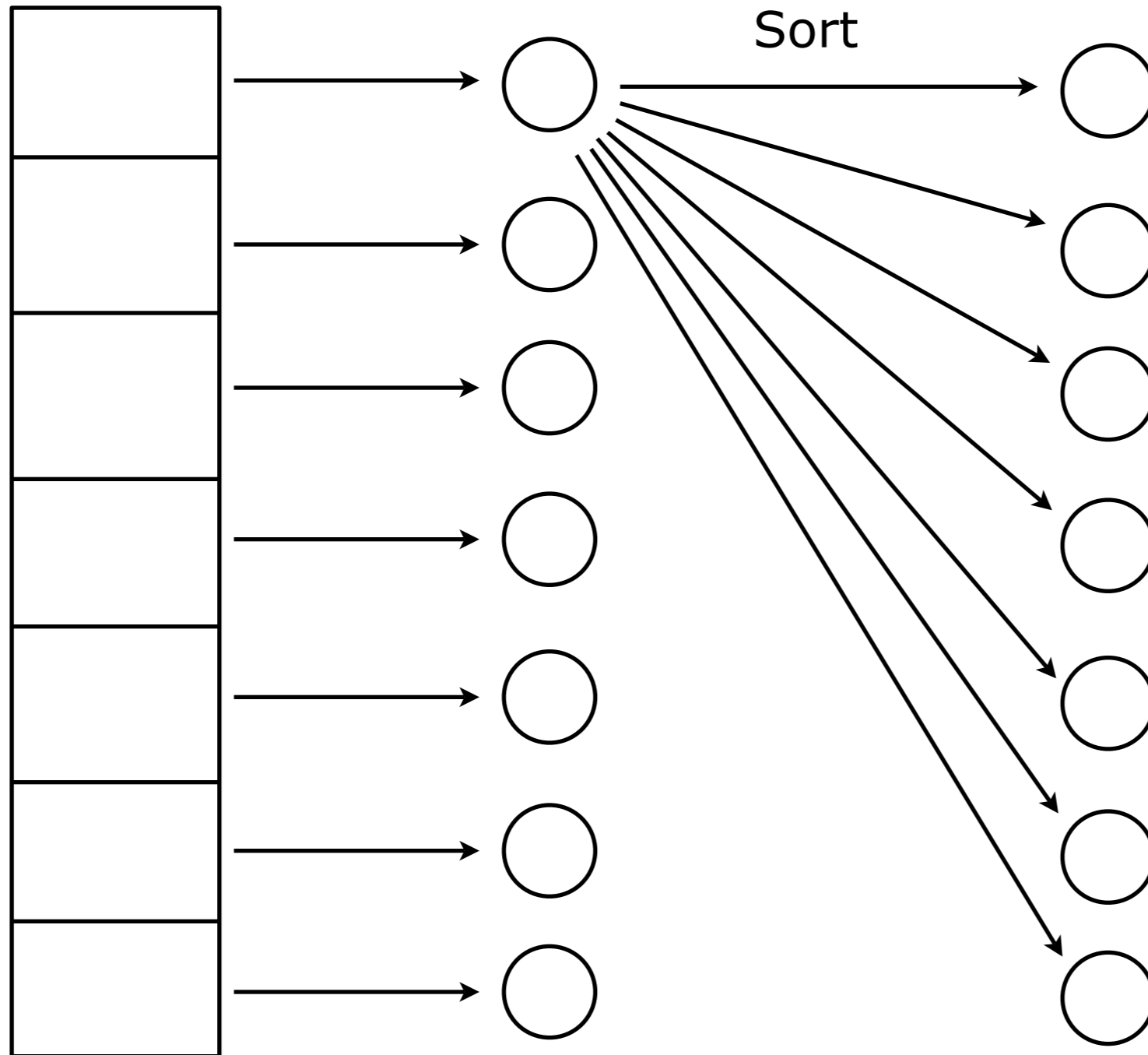


Map/Reduce Overview

Input Data

Map

Reduce



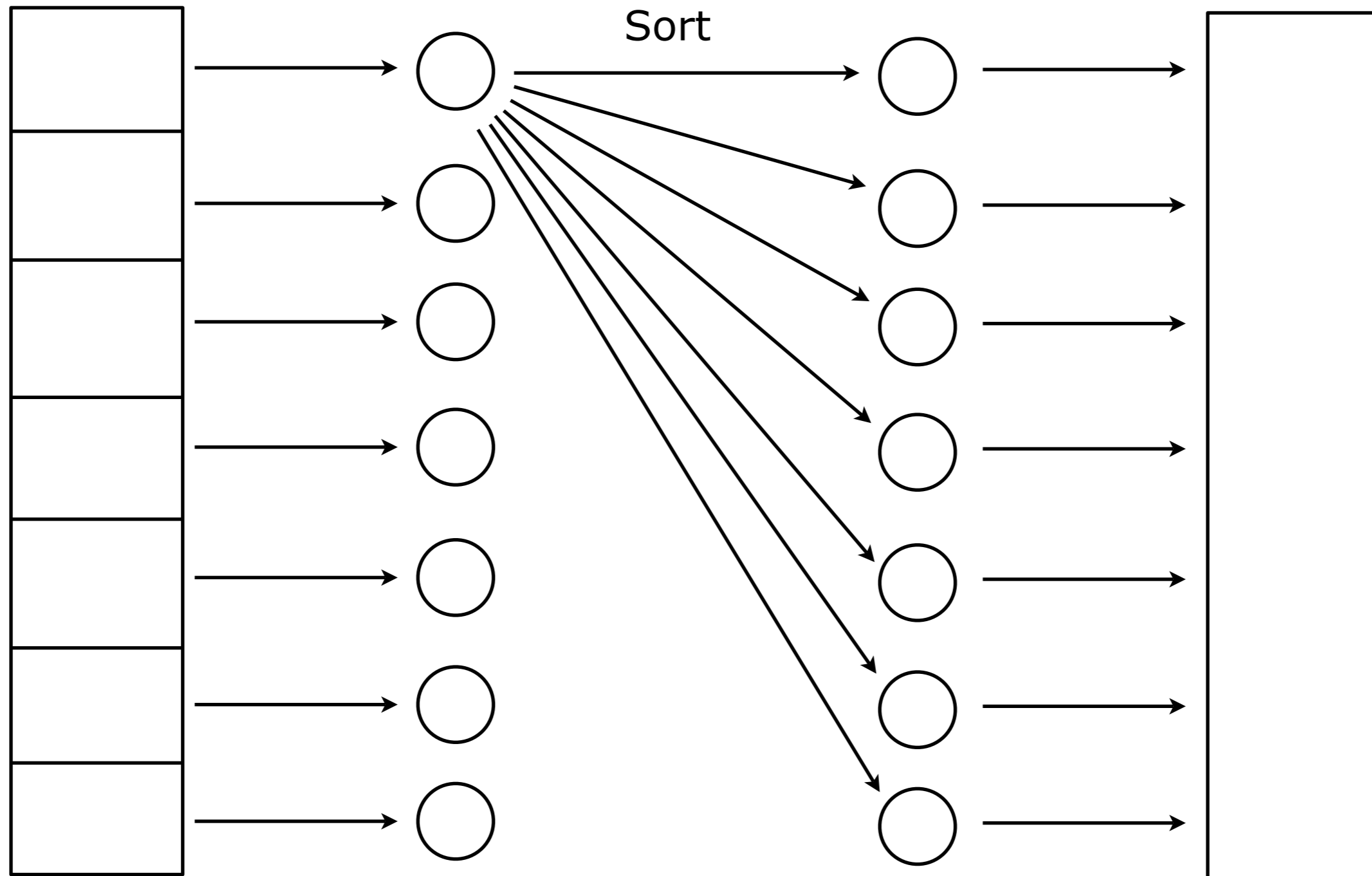
Map/Reduce Overview

Input Data

Map

Reduce

Result Files



Metaweb's Hadoop Cluster

Metaweb's Hadoop Cluster

- 20 servers

Metaweb's Hadoop Cluster

- 20 servers
 - 8 cores

Metaweb's Hadoop Cluster

- 20 servers
 - 8 cores
 - 16 GB memory

Metaweb's Hadoop Cluster

- 20 servers
 - 8 cores
 - 16 GB memory
 - 5x500 GB disks

Metaweb's Hadoop Cluster

- 20 servers
 - 8 cores
 - 16 GB memory
 - 5x500 GB disks
- 160 concurrent tasks

Metaweb's Hadoop Cluster

- 20 servers
 - 8 cores
 - 16 GB memory
 - 5x500 GB disks
- 160 concurrent tasks
- 15 TB of space after 3x replication

Metaweb's Hadoop Cluster

- 20 servers
 - 8 cores
 - 16 GB memory
 - 5x500 GB disks
- 160 concurrent tasks
- 15 TB of space after 3x replication
- Open for business!

So what is Happy?

- Happy = Hadoop + Python
- A Map-Reduce programming framework in Python
- Based on Jython
 - Compatible with Python 2.5
 - Great Java library integration
 - Speed is comparable to CPython

An Example

```
import sys, happy
```

```
class WordCount(happy.HappyJob):
```

```
    def __init__(self, inputpath, outputpath):
```

```
        happy.HappyJob.__init__(self)
```

```
        self.inputpaths = inputpath
```

```
        self.outputpath = outputpath
```

```
        self.inputformat = "text"
```

```
    def map(self, records, task):
```

```
        for _, value in records:
```

```
            for word in value.split():
```

```
                task.collect(word, "1")
```

```
    def reduce(self, key, values, task):
```

```
        count = 0;
```

```
        for _ in values: count += 1
```

```
        task.collect(key, str(count))
```

```
if __name__ == "__main__":
```

```
    if len(sys.argv) < 2:
```

```
        print "Usage: <inputpath> <outputpath>"
```

```
        sys.exit(-1)
```

```
    wc = WordCount(sys.argv[1], sys.argv[2])
```

```
    wc.run()
```

Constructor

```
class WordCount(happy.HappyJob):  
    def __init__(self, inputpath, outputpath):  
        happy.HappyJob.__init__(self)  
        self.inputpaths = inputpath  
        self.outputpath = outputpath  
        self.inputformat = "text"
```

Map Function

```
def map(self, records, task):  
    for _, value in records:  
        for word in value.split():  
            task.collect(word, "1")
```

Reduce Function

```
def reduce(self, key, values, task):  
    count = 0;  
    for _ in values: count += 1  
    task.collect(key, str(count))
```

Job Invocation

```
if __name__ == "__main__":  
    if len(sys.argv) < 2:  
        print "Usage: <inputpath> <outputpath>"  
        sys.exit(-1)  
    wc = WordCount(sys.argv[1], sys.argv[2])  
    wc.run()
```

Usage

```
./bin/happy.sh ./thirdparty/happy/examples/wordcount.py /  
data/starwars/starwars-20080207-pages-current.xml  
wcrests
```

How does Happy work?

How does Happy work?

- Subclass HappyJob

How does Happy work?

- Subclass HappyJob
- Implement map() and reduce()

How does Happy work?

- Subclass HappyJob
- Implement map() and reduce()
- Instantiate the object

How does Happy work?

- Subclass HappyJob
- Implement map() and reduce()
- Instantiate the object
- Set job parameters on the job

How does Happy work?

- Subclass HappyJob
- Implement map() and reduce()
- Instantiate the object
- Set job parameters on the job
- Call run()

How does Happy work?

How does Happy work?

- Then..

How does Happy work?

- Then..
- Happy serializes your job instance

How does Happy work?

- Then..
- Happy serializes your job instance
- Copies the job, and all accompanying libraries

How does Happy work?

- Then..
- Happy serializes your job instance
- Copies the job, and all accompanying libraries
- De-serializes the job on the cluster

How does Happy work?

- Then..
- Happy serializes your job instance
- Copies the job, and all accompanying libraries
- De-serializes the job on the cluster
- Calls `map()` or `reduce()`

Happy Path

Happy Path

- Set `HAPPY_PATH` in your environment to point to your libraries

Happy Path

- Set `HAPPY_PATH` in your environment to point to your libraries
- Paths are added to the Python path

Happy Path

- Set `HAPPY_PATH` in your environment to point to your libraries
- Paths are added to the Python path
- Path contents are also included with jobs

Happy Path

- Set `HAPPY_PATH` in your environment to point to your libraries
 - Paths are added to the Python path
 - Path contents are also included with jobs
- Don't add paths that you don't want copied up for each job (i.e. all of me or a 100MB file)

Happy Libraries

Happy Libraries

- `happy.json` - JSON encode and decode

Happy Libraries

- happy.json - JSON encode and decode
- happy.dfs - HDFS read and write access

What does Happy get me?

What does Happy get me?

- Python is easy to write

What does Happy get me?

- Python is easy to write
- Writing map-reduce programs is very simple

What does Happy get me?

- Python is easy to write
- Writing map-reduce programs is very simple
- Tight integration with Hadoop and Java libraries

What does Happy get me?

- Python is easy to write
- Writing map-reduce programs is very simple
- Tight integration with Hadoop and Java libraries
- Managing libraries is easy

What does Happy get me?

- Python is easy to write
- Writing map-reduce programs is very simple
- Tight integration with Hadoop and Java libraries
- Managing libraries is easy
- Distributing job state from the script to the cluster is simple

What does Happy get me?

- Python is easy to write
- Writing map-reduce programs is very simple
- Tight integration with Hadoop and Java libraries
- Managing libraries is easy
- Distributing job state from the script to the cluster is simple
- Chaining map-reduce operations is easy

new! happy.cloud

```
import sys
from happy.cloud import *

# This is a map function, which takes a key and value and
# yields key/value pairs:
def splitWords(key, sentence):
    for word in sentence.split(): yield word, 1

# This is a reduce function, which takes a key and iterator over
# values and yields key/value pairs:
def countWords(word, counts):
    sum = 0
    for count in counts: sum += 1
    yield word, sum

if __name__ == "__main__":
    inputpath = sys.argv[1]
    outputpath = sys.argv[2]

    # Here we create a new Session with working directory "wordcount":
    session = Session("wordcount")

    # Next, we define a source using the provided input path.
    # Because the input is a text file, we specify that the inputformat is text
    # and is not json encoded.
    src = session.source(inputpath, json=False, inputformat="text")

    # Now the data is passed through the splitWords and countWords functions:
    worddata = src.map(splitWords).reduce(countWords)

    # And then we save the data to the output path:
    worddata.sink(outputpath)

    # Now that the dataflow is defined, we run the jobs and cleanup the working directory:
    session.run()
    session.cleanup()
```

fin

fin

<http://code.google.com/p/happy/>