Extreme Computing

Introduction to MapReduce

Cluster

We have 12 servers: `scutter01`, `scutter02`, ... `scutter12`

If working outside Informatics, first:

```
ssh student.ssh.inf.ed.ac.uk
```

Then log into a random server:

```
ssh scutter$(printf "%02i"$((RANDOM%12+1)))
```

Please load balance! Two years ago the cluster crashed.

Cluster Software

The cluster runs Hadoop on DICE (the Informatics Linux Environment).

$\Longrightarrow$ No need to install software yourself.
You can run your own cluster but:

- We won't help you install it
- Copy your output to the cluster
- Code should run on the cluster

Cluster Software

The cluster runs Hadoop on DICE (the Informatics Linux Environment).

$\Longrightarrow$ No need to install software yourself.
You can run your own cluster but:

- We won't help you install it
- Copy your output to the cluster
- Code should run on the cluster

$\Longrightarrow$ Make sure your DICE account works!
We don't have `root` so only computing support can help.
Do this before the labs starting 2 October.

Companies I Take Money From

Likely Guest Lecture


Microsoft

Currently no Guest Lecture


Incremental Approach

Build MapReduce from problems.
Assemble picture at the end.

Assignment 1 is pure MapReduce problems.

grep

grep extreme

Find every line containing “extreme” in a text file.

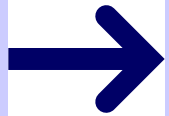
grep

grep extreme

Find every line containing "extreme" in a text file.

Input

extreme students
pay extremely high
this is slow
up to there
method extremely useful
take TTDS



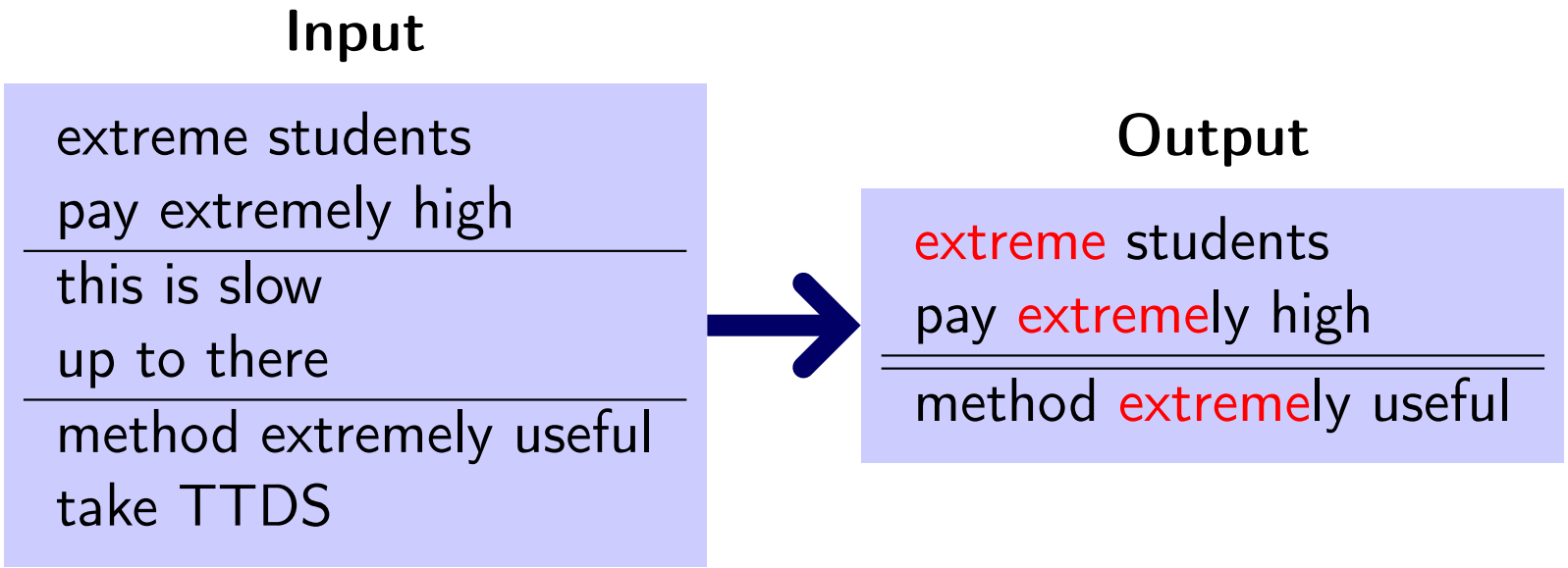
Output

extreme students
pay extremely high
method extremely useful

Distributed grep

`grep extreme`

Find every line containing "extreme" in a text file.



Split input into pieces, run `grep` on each.

Interlude: Pieces of a Text File

Goal: assign a piece of the text file to each machine.

- Non-overlapping
- Break at line boundaries
- Fast (don't read more than you have to)
- Balanced (roughly equal sizes)

seeking

`seek` allows one to skip to a particular **byte** in a file.

There is no seek for *line* offsets.
You'd have read the file from the beginning and count newlines.

But we can `seek` to a byte offset, then round up to the next line.

Rounding bytes to lines

Split a 300-byte text file:

Task	Byte Assignment	Line Rounding
0	0–99	0–102
1	100–199	103–207
2	200–299	208–299

Each task can read until it sees a newline, then round up to that.

$\rightarrow$ Work is divided at line boundaries.

Hadoop is an implementation of MapReduce.

This just shows how Hadoop splits input:

```
hadoop jar hadoop-streaming-2.7.3.jar          Run Hadoop
-input /data/assignments/ex1/webSmall.txt      Read big text file
-output /user/$USER/catted                     Write here
-mapper "cat"                                  Just copy the input
-reducer NONE                                  Ignore this for now
```

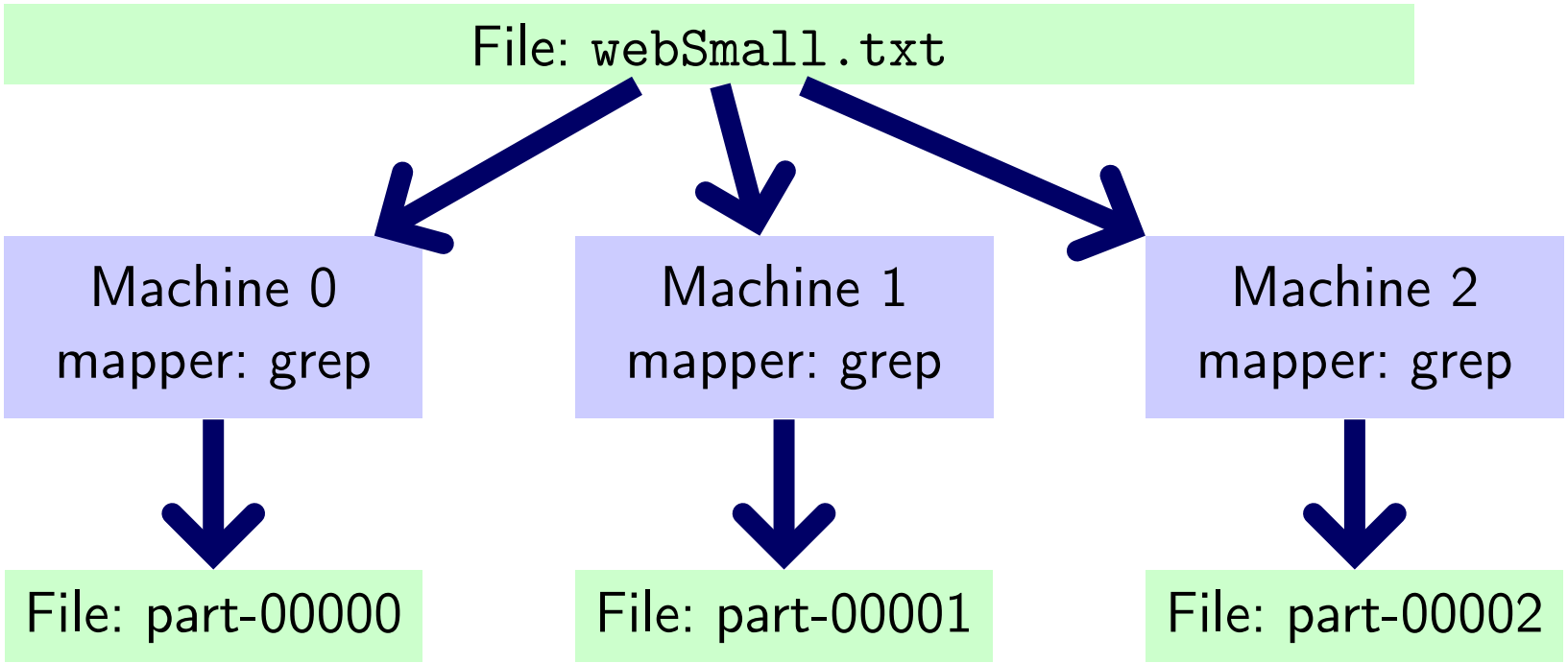
Don't worry, you'll get too much practice in the labs.

Distributed grep

```
hadoop jar hadoop-streaming-2.7.3.jar
-input /data/assignments/ex1/webSmall.txt
-output /user/$USER/grepped
-mapper "grep extreme"
-reducer NONE
```

Run Hadoop
Read big text file
Write here
Scan for “extreme”
Ignore this for now

Summarizing



Hadoop takes care of:

- Shared file system
- Splitting input at line boundaries
- Launching tasks on multiple machines

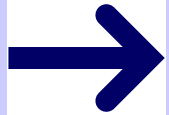
We can specify any command ("a mapper") to run.

Word Count

How many times do words appear?

Input

want to use a
a to
a decimal



Output

a	3
want	1
to	2
use	1
decimal	1

Each mapper counts independently:

Mapper 0

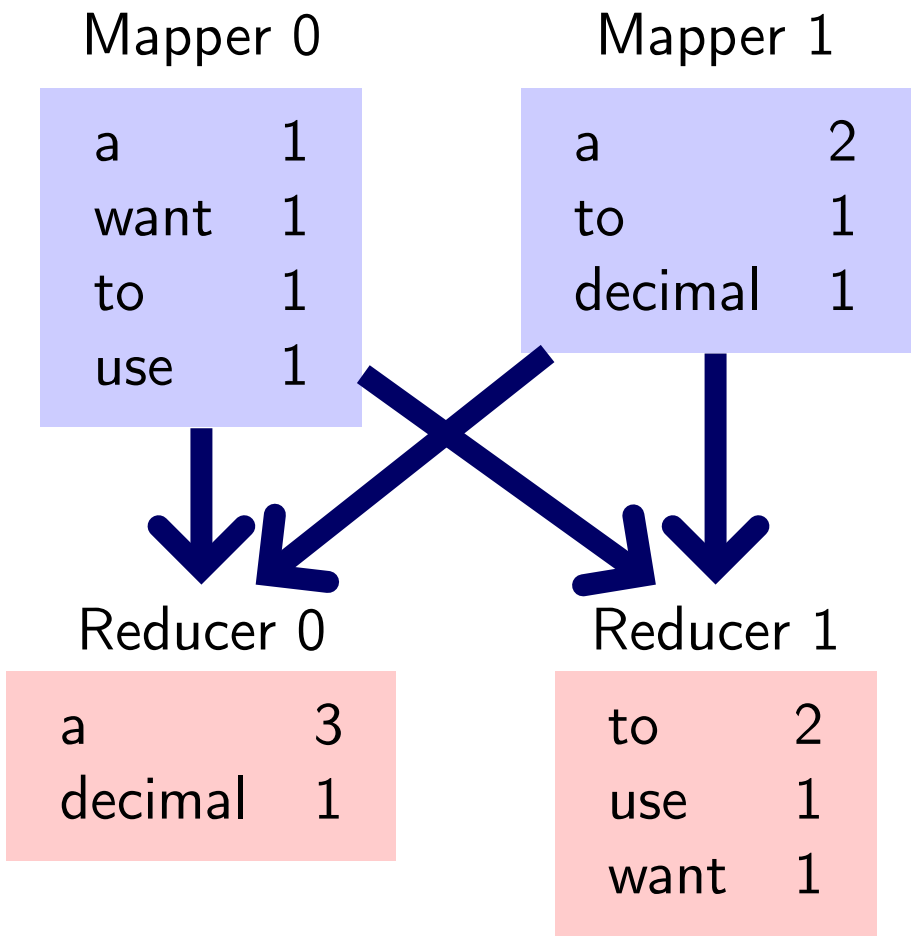
a	1
want	1
to	1
use	1

Mapper 1

a	2
to	1
decimal	1

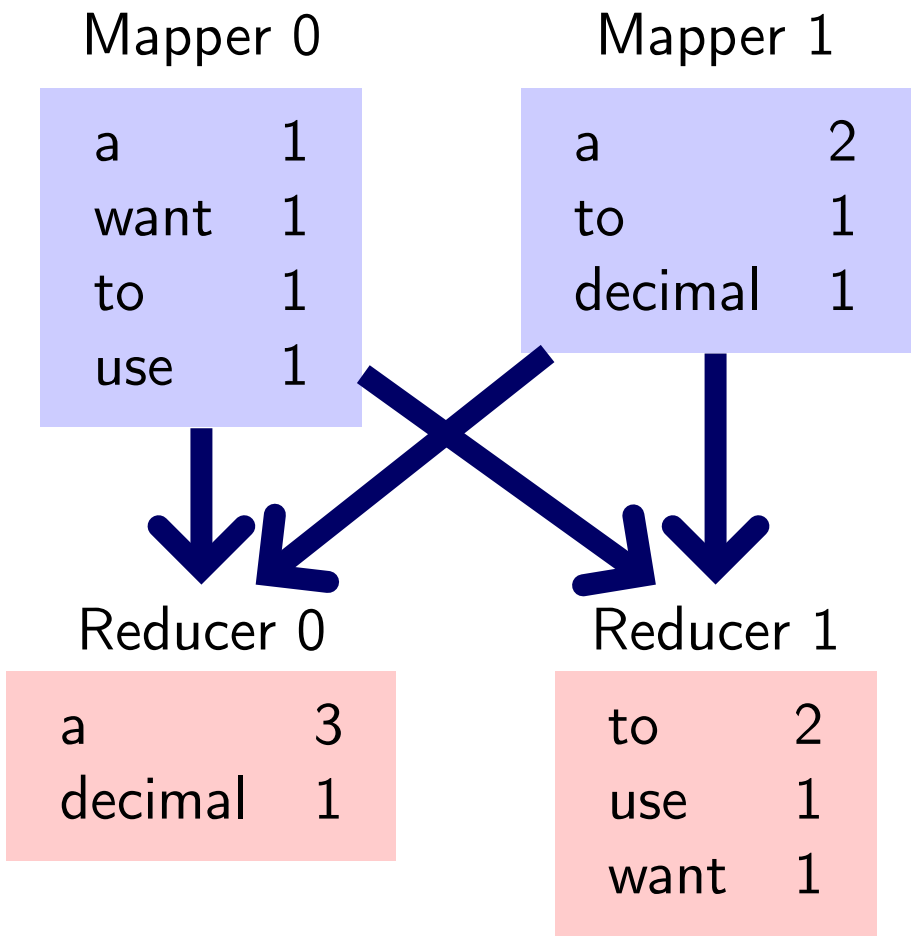
Problem: Need to collate/sum counts

Each mapper counts independently:



Reducers sum counts

Each mapper counts independently:



Reducers sum counts

Mappers hash the word mod 2 to decide which reducer to send to.

Examine Reducer Input

```
hadoop jar hadoop-streaming-2.7.3.jar          Run Hadoop
-files count_map.py                            Copy code to workers
-input /data/assignments/ex1/webSmall.txt      Read big text file
-output /user/$USER/reducespy                  Write here
-mapper count_map.py                           Count words locally
-reducer cat                                   Leave as is
```

`cat` will copy input to output, so we can see what the input is.

Sorting

Hadoop sorts reducer input for you:

Unsorted: Annoying

to	1
want	1
use	1
to	1

Sorted: Easy

to	1
to	1
use	1
want	1

Sorting makes it easy to stream in constant memory.
Unsorted would require remembering words in memory.

Examine Reducer Input

```
hadoop jar hadoop-streaming-2.7.3.jar
-files count_map.py,count_reduce.py
-input /data/assignments/ex1/webSmall.txt
-output /user/$USER/count
-mapper count_map.py
-reducer count_reduce.py
```

Run Hadoop
Copy code to workers
Read big text file
Write here
Count words locally
Sum counts

And we get word count. . . hopefully