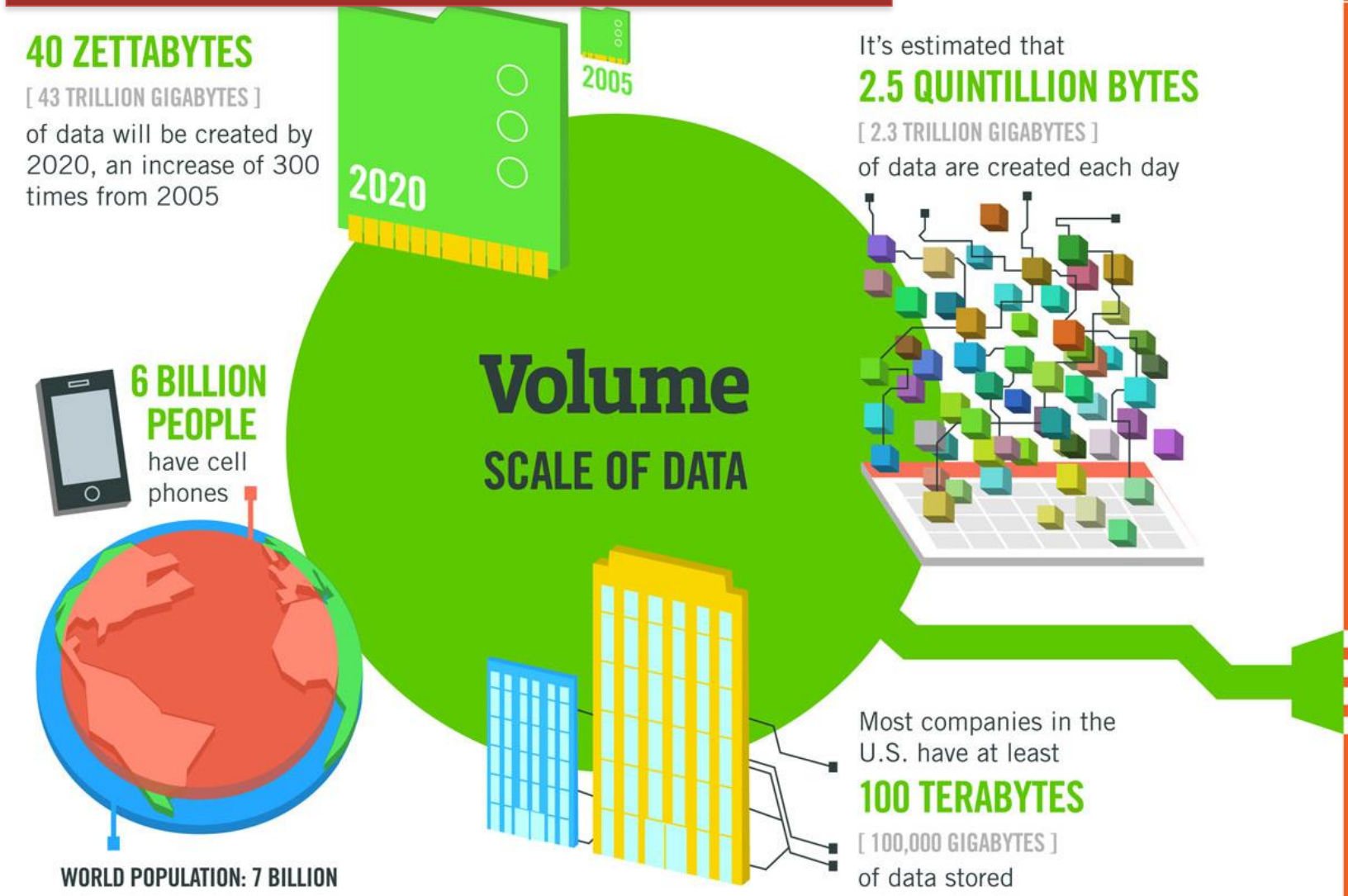
Science & Technology Facilities Council | UK Research and Innovation | NERC SCIENCE OF THE ENVIRONMENT

Parallel processing of large datasets

What are Big Data?

1 ZETTABYTE = 1,000,000,000 TERABYTES



Processing big data: the issues

- Parallel processing in the Environmental Sciences has historically focussed on running highly-parallelised models.
- Data analysis was typically run sequentially because:
  - It was a smaller problem
  - It didn't have parallel resources available
  - The software/scientists were not equipped to work in parallel
- The generation of enormous datasets (e.g. UPSCALE – around 300Tb) means that:
  - Processing big data **requires** a parallel approach
  - Fortunately, platforms, tools, and programmers are becoming better equipped

Some Terminology

Concurrency: A property of a system in which multiple tasks that comprise the system remain active and make progress at the same time.

Parallelism: Exploiting concurrency in a programme with the goal of solving a problem in less time.

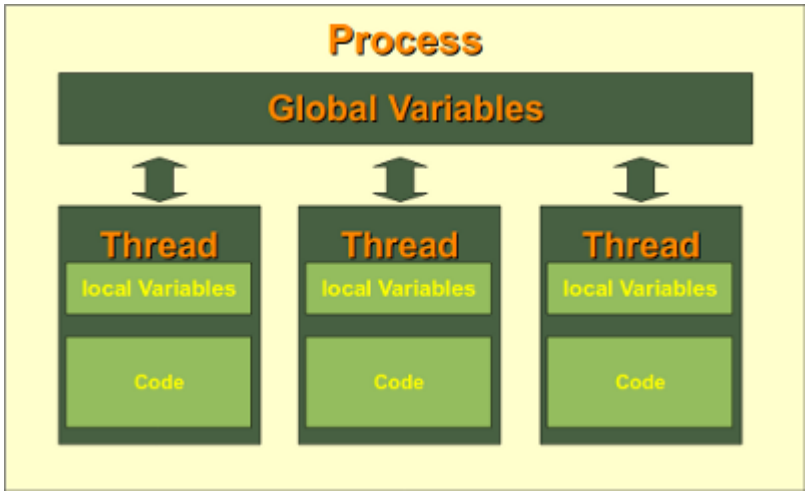
Race condition: A race condition occurs within concurrent environments. It is when a piece of code prevents code that is running elsewhere from accessing a shared resource, e.g., memory, and thus delays the other process.

How does my computer do so many things at once?

These days most computers, and even phones, have multiple processors.

However, even on a single processor modern operating systems can give the illusion that multiple tasks are running at the same time by rapidly switching between many active threads.

This is because the modern CPU clock is measuring time at the nanosecond scale where we humans can only keep track of milliseconds.



Picture: http://www.python-course.eu/threads.php

Parallel processing for data analysis

- Data analysis tools do not (typically) do parallelisation automatically.
- But parallelisation is normally achievable at a small price.
- A lot can be done with:
  - Decomposition of large jobs into smaller jobs
  - Batch processing
  - Understanding tools and schedulers

We will look at these and show examples

(Almost) everything is parallel these days

YOUR DESKTOP MACHINE IS A PARALLEL COMPUTER!

It runs a multi-core processor...

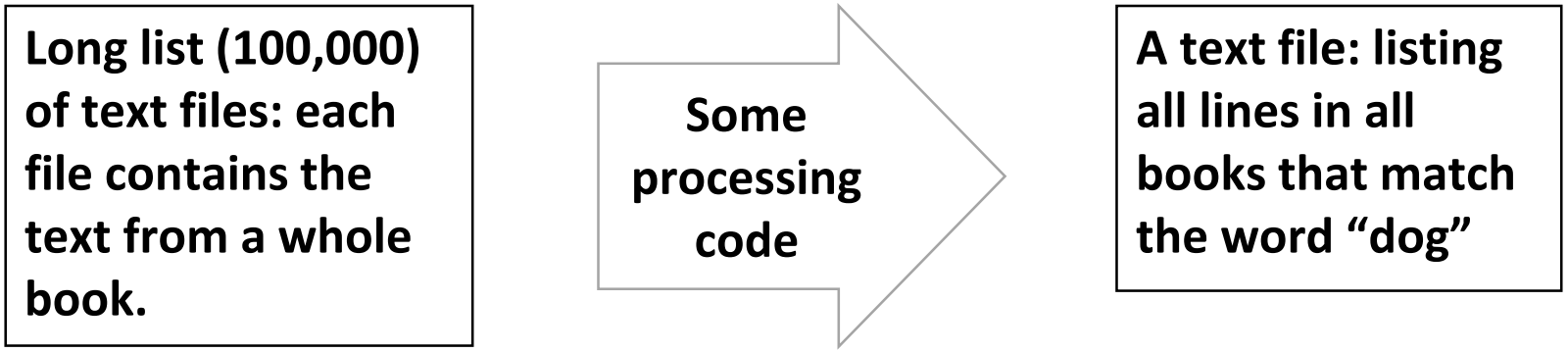
...which means you can speed up processing by asking different parts of your programme to run on different cores.

*"But what about **race conditions**?"...*

...True: you still need to design your approach to avoid things getting out of hand!

Simple parallelism by hand (1)

- Running on a multi-core machine you can exploit local processes, e.g.:

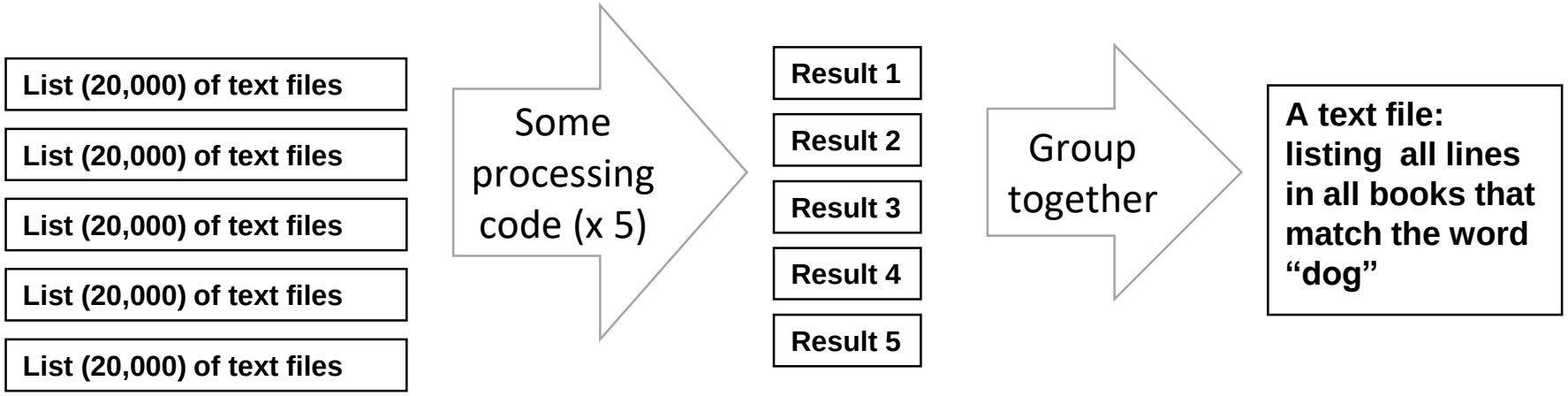


grep_for_dog.sh

```
#!/bin/bash
input_file=$1
while read FILENAME; do
      grep dog $FILENAME >>
${input_file}_result.txt
done <  $input_file
```

Simple parallelism by hand (2)

- A simple re-factoring splits the job into five parts:




```
$ split -l 20000 -d list_of_files.txt  # Writes to "x00", "x01", ...
$ for i in x??; do grep_for_dog.sh $i & done
$ cat *_result.txt > output.txt
```

Simple parallelism by hand (3)

```
$ for i in x??; do grep_for_dog.sh $i & done
```

```
[2] 3325
[3] 3326
[4] 3327
[5] 3328
[6] 3329
```

```
$ ps -ef | grep grep_for_dog
```

```
alison   3325   2669   0 00:40 pts/1   00:00:00 /bin/bash   ./grep_for_dog.sh   x00
alison   3326   2669   0 00:40 pts/1   00:00:00 /bin/bash   ./grep_for_dog.sh   x01
alison   3327   2669   0 00:40 pts/1   00:00:00 /bin/bash   ./grep_for_dog.sh   x02
alison   3328   2669   0 00:40 pts/1   00:00:00 /bin/bash   ./grep_for_dog.sh   x03
alison   3329   2669   0 00:40 pts/1   00:00:00 /bin/bash   ./grep_for_dog.sh   x04
```

Simple parallelism by hand (4)

Some time later…

```
$ ps -ef   | grep grep_for_dog
```

[2] Done ./grep_for_dog.sh $i

[3] Done ./grep_for_dog.sh $i

[4] Done ./grep_for_dog.sh $i

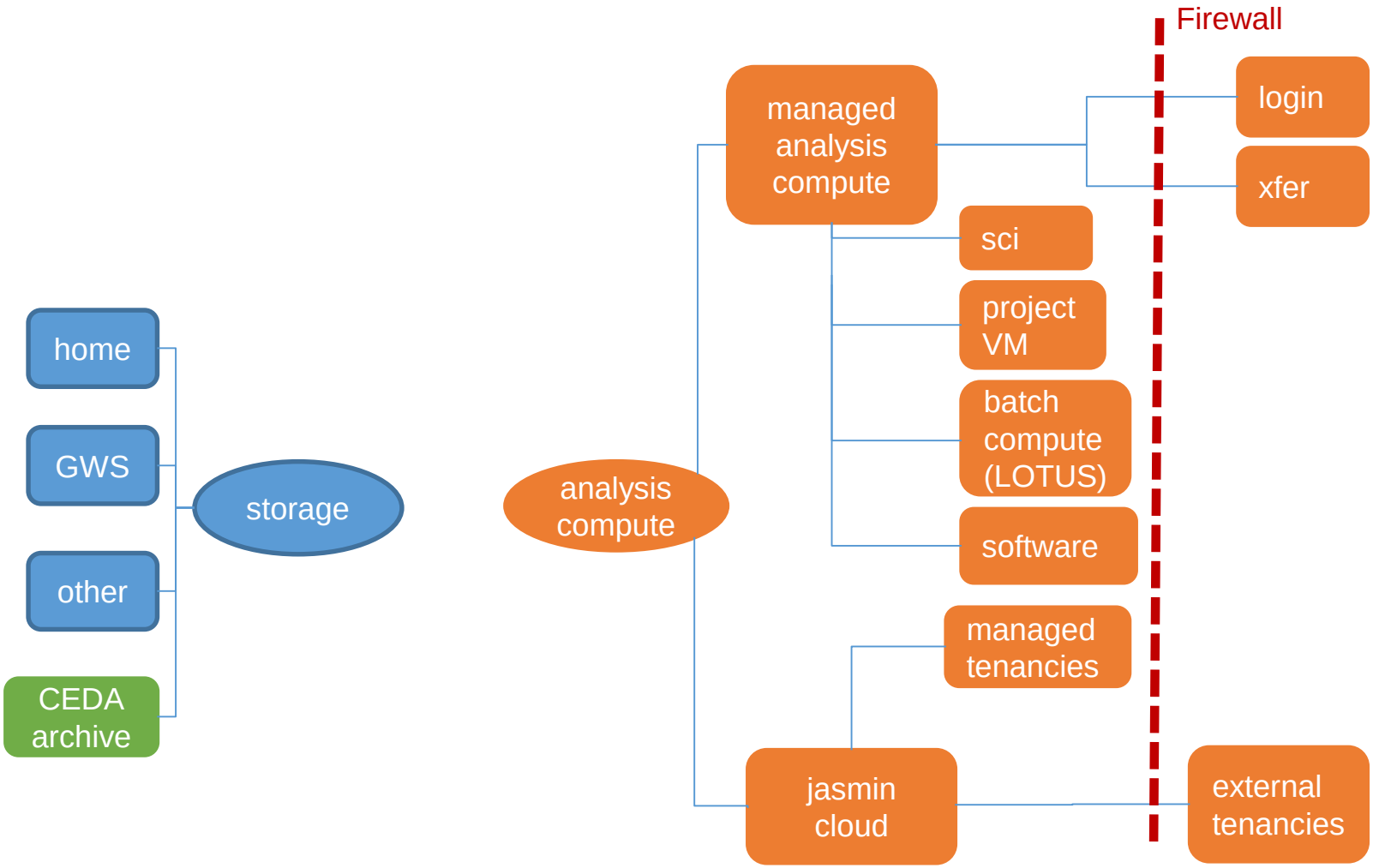
[5]- Done ./grep_for_dog.sh $i

[6]+ Done ./grep_for_dog.sh $i

JASMIN & LOTUS

Scientific computing on JASMIN

JASMIN



Getting access to JASMIN

- Applying for an account:

https://help.jasmin.ac.uk/article/189-get-started-with-jasmin

- Generate an ssh key pair (required)
- Apply for JASMIN account: https://accounts.jasmin.ac.uk/ and upload your **PUBLIC** ssh key
- Apply for JASMIN login access and (optionally) a CEDA account, access to project resources, fast data transfer service, etc.
- Once access is granted, login using your ssh key

https://help.jasmin.ac.uk/article/187-login

- ssh to sci machine / project VM / lotus head node

Generating an ssh key pair

- On Linux and Mac use a terminal window (bash)
- On Windows use MobaXterm – free to download **https://help.jasmin.ac.uk/article/4832-mobaxterm-new**
- Use ssh-keygen command to generate an RSA key pair – for JASMIN access you are **required** to set a passphrase
- On Linux and Mac use ssh-agent and ssh-add to load a key for use with an ssh login
- On Windows/MobaXterm – follow instructions in JASMIN help page
- Once key is loaded, login to JASMIN with ssh

The LOTUS cluster on JASMIN

The LOTUS **cluster** is **a far bigger resource** for running compute intensive jobs than the JASMIN Scientific Analysis Servers.

Having the same software installed on the JASMIN-Sci machines and LOTUS means you can:

1. develop code on the generic Analysis Servers
2. run in **batch mode** via LOTUS

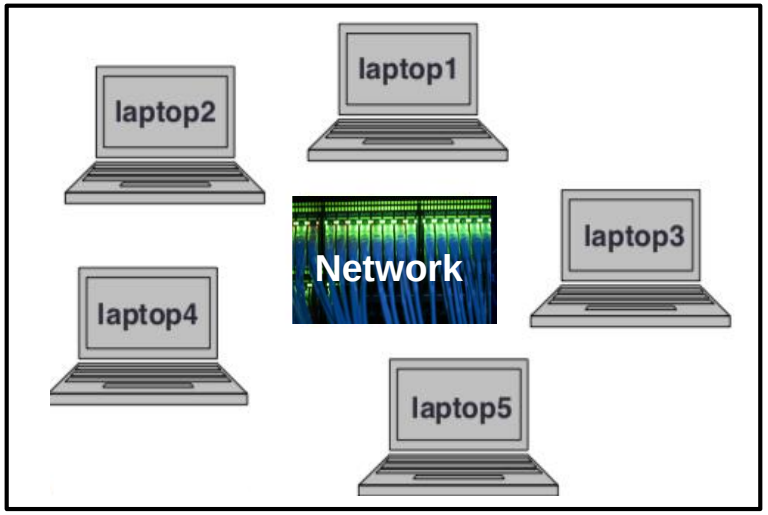
LOTUS: JASMIN's batch processing cluster (1)

What is a cluster?

A cluster is a collection of computers working together to solve a large problem in a significantly reduced time.

A generic view of a cluster:

- Laptops connected by a network
- Each laptop has 4 cores/processors
- Each laptop is called a compute node
- Each has its own operating system
- Cluster of 20 cores/processors
- Cluster of 5 compute nodes



LOTUS: JASMIN's batch processing cluster (2)

What is batch processing?

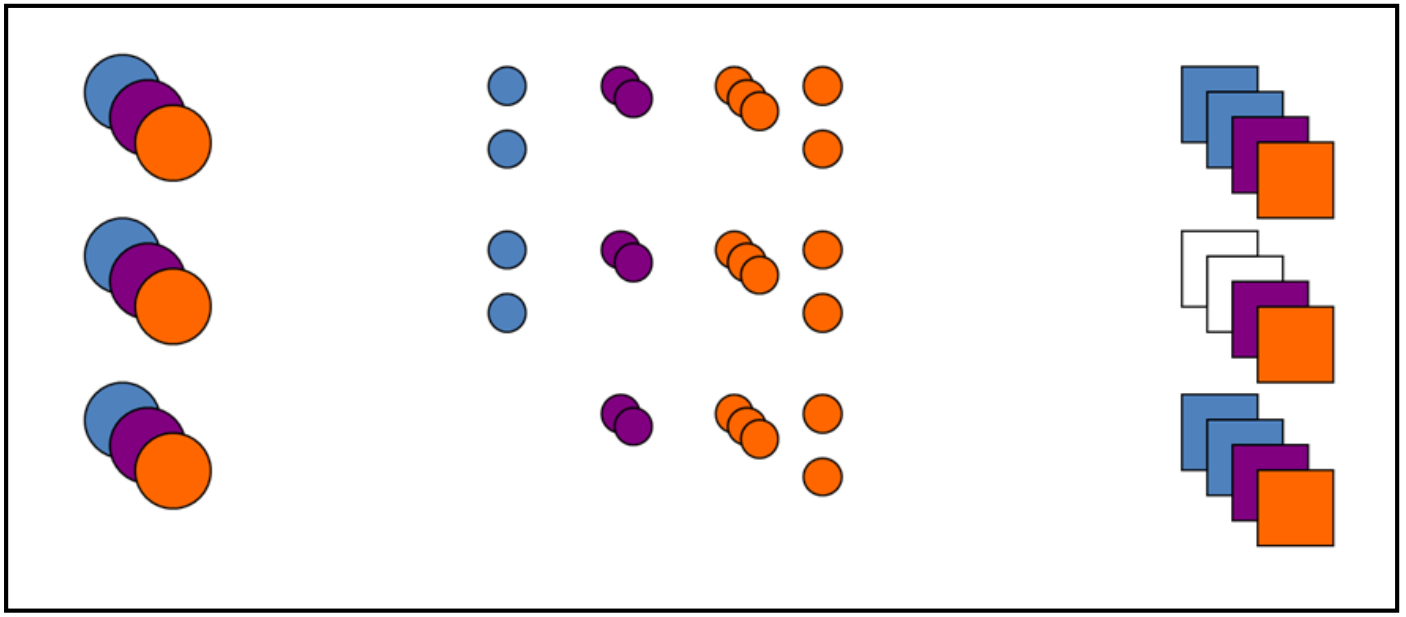
Batch processing is the execution of a program or a sequence of commands without user interaction. Program execution does not occur instantly, but via a **scheduler** that manages the **compute resources** based on a set of predefined **policies**.

Differences from interactive processing

- User does not login to a compute node directly and type individual commands
- Not a GUI-based environment
- Resources more tightly monitored and controlled
- Greater computing power and more resources e.g. high memory 512GB

Job Submission

- Jobs are submitted using the LSF scheduler
- Resources are allocated as they become available
- Fair share of resources between users



LSF queues on LOTUS

Serial queues -single core

- `short-serial` : default queue, runtime limit 24 hrs, memory control limit
- `long-serial` : runtime limit 168 hrs, memory control limit

Parallel queues -multiple cores, runtime limit 48 hrs

- `par-single` : 16 cores limit on a single compute node
- `par-multi` : cores distributed across many compute nodes

High memory - single core

- `high-mem`

LSF Scheduler commands

To interact with the LOTUS LSF scheduler first login to one of:

- JASMIN scientific servers (jasmin-sci[1-5].ceda.ac.uk)
- LOTUS head node (lotus.jc.rl.ac.uk)

Job submission: `bsub <options> command`

```
$ bsub -o %J.out -W 00:10 python2.7 mypythonscript.py
Job <6485340> is submitted to default queue <short-serial>.
```

Job information: `bjobs`

```
$ bjobs
JOBID     USER   STAT  QUEUE        FROM_HOST    EXEC_HOST   JOB_NAME SUBMIT_TIME
6485340   msmiz   RUN  short-serial jasmin-sci1  host177.jc. Myjob1   Jul 21 11:46
6485346   msmiz   RUN  short-serial jasmin-sci1  host232.jc. Myjob2   Jul 21 11:46
```

Cancel a job: `bkill <job_id>`

```
$ bkill 6485346
 Job <6485346> is being killed
```

LSF commands & job states

Table 1: LSF commands (Use manual page e.g. man bsub)

Job submit command	Job status command	Job control command

`bsub`	`bjobs`
	`bhist`
	`bqueues`
		`bmod`
		`bstop`
		`bresume`
		`bkill`

Table 2: LSF job states

LSF job state	Meaning
`PEND`	Job is waiting in a queue
`RUN`	Job is currently running
`DONE`	Job finished with zero exit value
`EXIT`	Job finished with non-zero exit value
`PSUSP`	Job suspended while pending
`USUSP`	Suspended by user (by LSF system SSUSP)

Batch job example: extract spatial subsets from CMIP5 experiments (1)

Processing requirement:

For each model:

For each variable (hus, ps, ta, ua & va):

Extract a spatial subset

- (80° to 140° Longitude; -30° to 40° Latitude)

Where:

- Frequency: 6hr
- Ensemble: r1i1p1
- Realm: atmosphere

Batch job example: extract spatial subsets from CMIP5 experiments (2)

Basic (Sequential) Implementation:

Script 1 (bash):

- For each variable (hus, ps, ta, ua & va):
  - Make output directory
  - Find all relevant input NetCDF files
  - Loop through list of input files and for each one call Python script

Extract from: **extract_cmip5_subset.py**

```
import cf
f = cf.read(infile)
subset = f[2].subspace(latitude=cf.wi(bb.south,
bb.north), longitude=cf.wi(bb.west, bb.east))
cf.write(subset, outfile)
```

Batch job example: extract spatial subsets from CMIP5 experiments (3)

Parallel Implementation using LOTUS:

Script 1 (bash):

- For each variable (hus, ps, ta, ua & va):
  - Make output directory
  - Find all relevant input NetCDF files
  - Loop through list of files and for each one submit a batch job to LOTUS to call the Python script using **bsub**

```
bsub -q par-single -o $outdir/`date +%s`.txt
~/extract_cmip5_subset.py $nc_file $this_dir $var
```

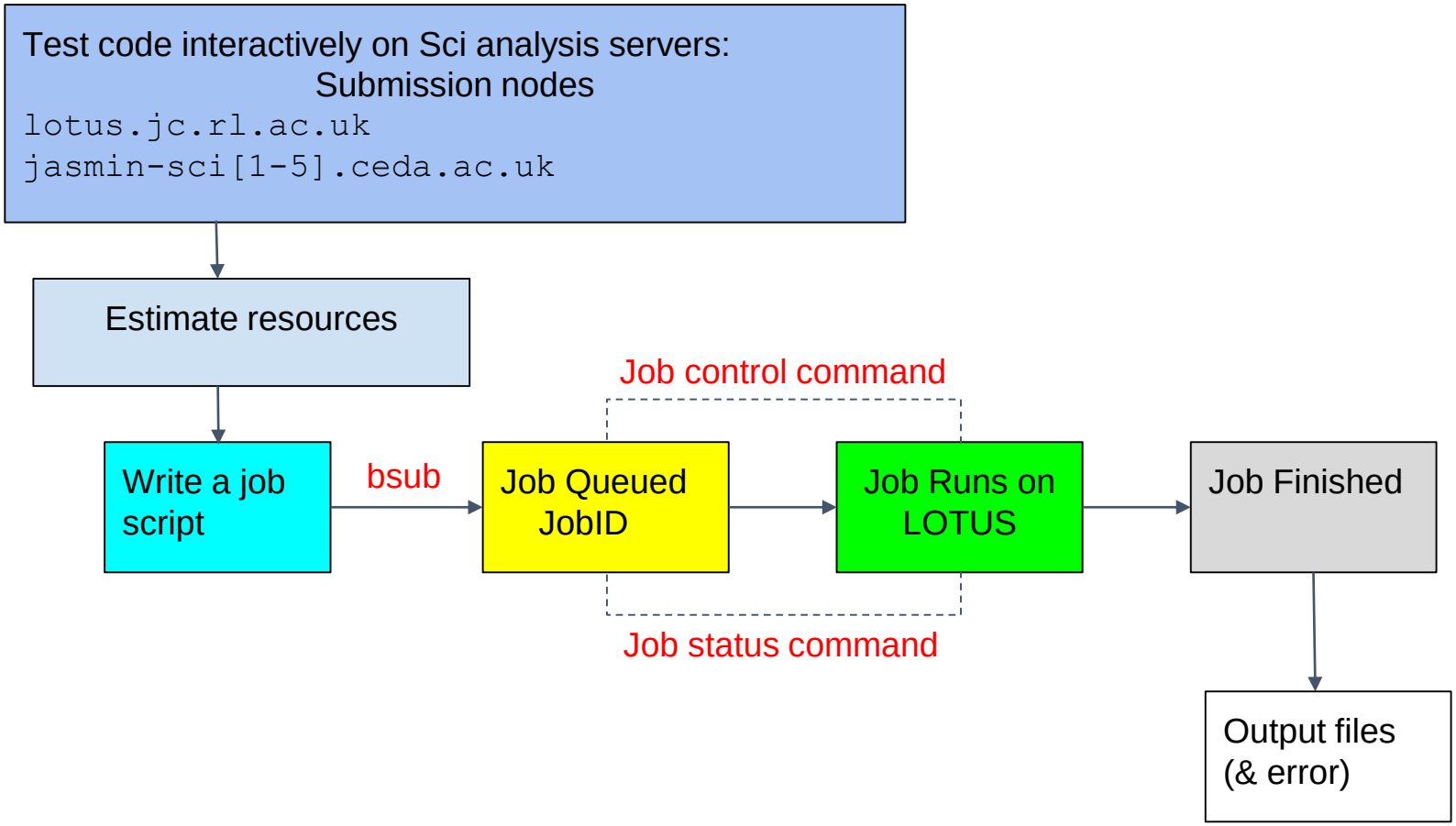
Batch job example: extract spatial subsets from CMIP5 experiments (4)

Why use this approach?

- Because you can submit 200 jobs in one go.
- Lotus executes jobs when resource becomes available
- They will all run and complete in parallel

```
bsub -q par-single -o $outdir/`date +%s`.txt
~/extract_cmip5_subset.py $nc_file $this_dir $var
```

Typical workflow for LSF jobs on LOTUS



Efficiency gains through re-factoring (1)

Major gains can be made by changing the order and structure of your code. Issues might be:

1. Code runs sequentially and takes a long time
2. Code runs slowly because processing order leads to inefficient I/O
3. Code will not run because of memory requirements

In some cases you can create loops that can be scripted as separate processes allowing you to submit them in parallel.

Efficiency gains through re-factoring (2)

Here is a real-world example:

The Problem: Trying to run the NCO tool **ncea** to calculate an average from a large dataset. It will not run!

Why? The **ncea** command reports this:

```
unable to allocate 7932598800 bytes
```

(which is about 8 Gbytes) ...and then exits.

Possible solutions:

1. Data files hold multiple variables: Operate on one at a time:

```
ea -v vosaline means/199[45678]/*y01T.nc -o test.nc
```

2. Reduce the number of files (i.e. years) processed each time:

```
ncea means/199[45]/*y01T.nc -o test.nc
```

The future of parallel data analysis (1)

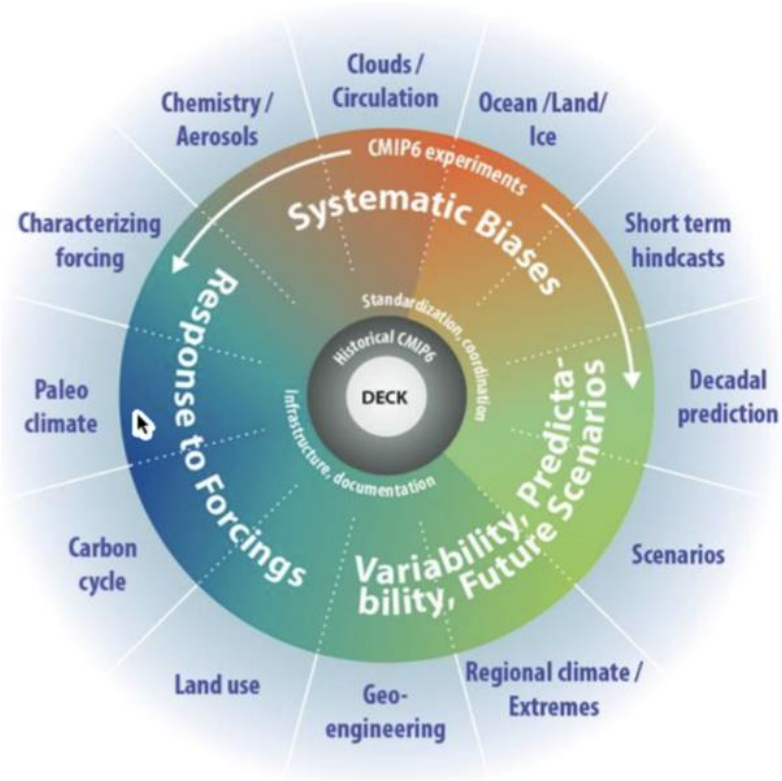
Analysing Big Data is a challenge! Software needs to adapt and scientists need to be able to adapt their code to keep up!

CMIP5 Status (early 2013)	
Number of files	3,222,967
Number of datasets	54,274
Archive Volume (TB)	1,483
Models with data published	64
Models with documentation published in archive	38
Experiments	108
Modelling centres	32
Data Nodes	22

The future of parallel data analysis (2)

CMIP6: 6th Coupled Model Intercomparison Project

Global community activity under the World Meteorological Organisation (WMO) via the World Climate Research Programme (WCRP)



- 33 institutions
- 75 models
- 248 experiments
- Approximately 20-30PB

CEDA are currently preparing for ~13PB to be archived

The future of parallel data analysis (3)

We are likely to see more:

- Web processing services that do the parallel analysis remotely;
- Analysis Platforms (like JASMIN) that allow scientists to run code next to the data;
- Parallel I/O in software libraries.

Learning to write parallel code now is likely to be of great benefit in future

Further information

LOTUS Overview:
https://help.jasmin.ac.uk/article/110-lotus-overview
https://help.jasmin.ac.uk/article/212-batch-scheduler-overview

LOTUS User Guide
help.ceda.ac.uk/category/107-batch-computing-on-lotus

JASMIN Analysis Platform (software packages):
https://help.jasmin.ac.uk/article/271-jap

Python-based Parallel tools:
https://wiki.python.org/moin/ParallelProcessing

Jug:
jug.readthedocs.io/en/latest/

Parallel processing:
https://computing.llnl.gov/tutorials/parallel_comp/