



ilifu Online Training - Advanced #2

Dr Jordan Collier

ilifu Support Astronomer, IDIA, Department of Astronomy, University of Cape Town
Adjunct Fellow, Western Sydney University



UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD



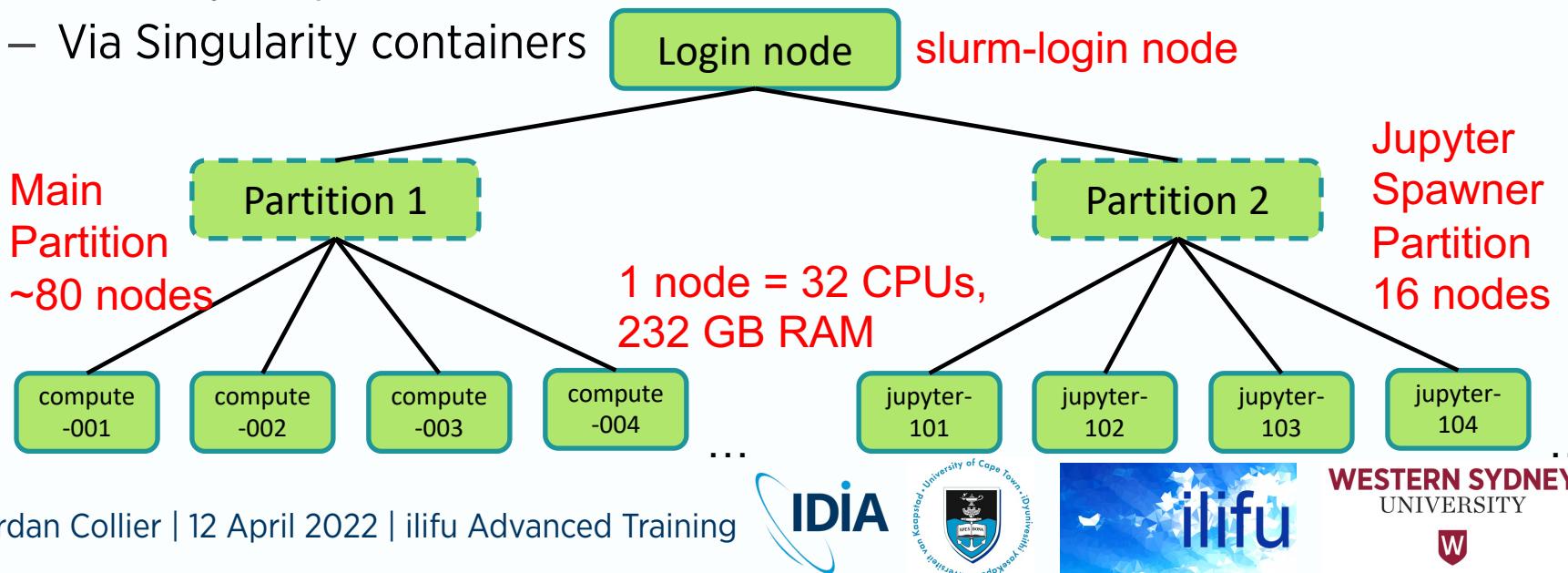
Inter-University Institute
for Data Intensive Astronomy

WESTERN SYDNEY
UNIVERSITY



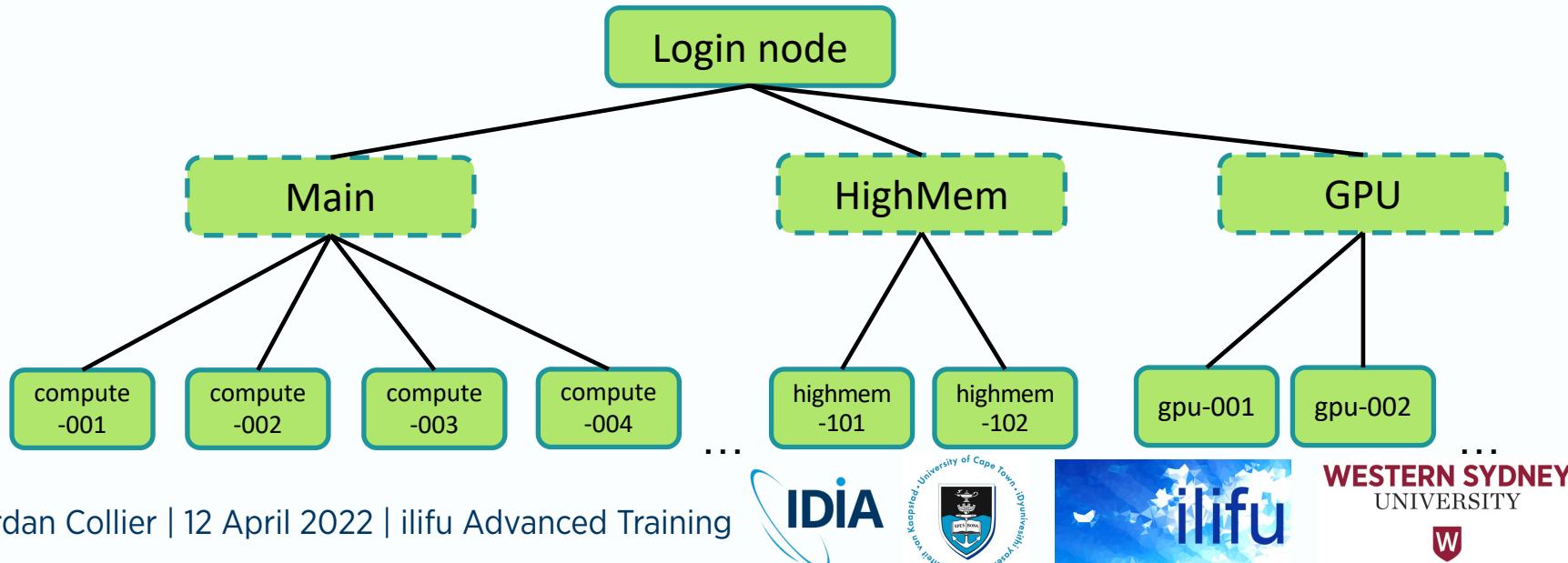
SLURM

- http://docs.ilifu.ac.za/#/getting_started/submit_job_slurm
- Login node (job submission & management)
 - Where you land when you log in (also known as “head node”)
 - Run SLURM commands/submit jobs, but not software/heavy processes
- Compute nodes
 - Where your processes run (also known as “worker nodes”)
 - Via Singularity containers



SLURM

- http://docs.ilifu.ac.za/#/tech_docs/running_jobs?id=_4-specifying-resources-when-running-jobs-on-slurm
- Partitions (other than Jupyter) - see with 'sinfo':
 - Main: 86 nodes (currently), each w/ 32 CPUs, 232 GB (usable) RAM
 - HighMem: 2 nodes, each w/ 32 CPUs, 480 GB (usable) RAM
 - GPU: 5 nodes, each w/ 2 GPUs, 32 CPUs, 232 GB (usable) RAM



Parallelism

- Oxford definition for parallel processing
 - *a mode of operation in which a process is split into parts, which are executed simultaneously on different processors attached to the same computer [or different computers attached to the same cluster].*
 - A cluster includes many connected nodes, each with its own RAM & CPUs
 - A node = single computer / server / VM / machine / box
- The work is partitioned into smaller jobs, sometimes with a partition of the dataset



What is a program?

- Set of discrete instructions
 - Carried out sequentially
 - Example: print average grade of a class
1. total = 0
 2. for grade in grades:
 total = total + grade
 3. average = total / number_of_grades
 4. print(average)



Parallelism

- Executing portions of program simultaneously
- Possible when we have many processors (cores/CPUs)
- Capacity dependent on structure of both hardware AND software
- Requires overall control/coordination mechanism
 - i.e. message passing

Parallelism on the cluster

- A cluster includes many connected nodes
- Each node has RAM and multiple cores
- Work of job is partitioned into smaller jobs
- Sometimes with a partition of the data

Parallel execution of a program

- Partition grades into 2:

- total = 0
- for grade in 1/2 grades:

total = total + grade

- average1 = total / number_of_grades

- total = 0
- for grade in 1/2 grades:

total = total + grade

- average2 = total / number_of_grades

- Combine results

average = (average1 + average2) / number_of_partitions

Parallelism

- Can be achieved on a single machine / node
 - Distributes work over many CPUs
 - Typically implemented using OpenMP
- Or over multiple machines / nodes
 - Distributes work over many tasks, over 1+ nodes
 - Each given amount of memory to use
 - Generally requires a cluster
 - Typically implemented using OpenMPI
 - Requires a message passing interface (MPI) wrapper
 - mpirun, aprun, srun (SLURM), mpicasa (CASA 5)
 - Version of wrapper inside and outside container / venv must match
- Managed on ilifu by SLURM



Parallelism

- Implementing a normal job in SLURM
 - Will only use 1 CPU, 1 task, and 1 node
 - Default for many processes
- Implementing an OpenMP job in SLURM
 - Need to use >1 CPU, while nodes & tasks must be 1 (unless also using MPI)
 - cpus-per-task
 - May need to export OMP_NUM_THREADS
- Implementing an MPI job in SLURM
 - Need to use >1 task, while nodes and CPUs can be 1
 - nodes, ntasks-per-node, cpus-per-task
 - Need to wrap singularity in MPI call
- Cannot exceed 32 CPUs (or tasks) per node



SLURM – serial and multi-CPU jobs

- If code is serial, i.e. doesn't use OpenMP or MPI, increasing CPUs or nodes will not decrease execution time

```
#SBATCH --nodes=1  
#SBATCH --ntasks-per-node=1  
#SBATCH --cpus-per-task=1  
singularity exec /path/to/container.simg python myscript.py
```

- Using multiple CPUs within a node with OpenMP, where N is an optional number of CPUs (utilised by myscript.py)

```
#SBATCH --nodes=1  
#SBATCH --ntasks-per-node=1  
#SBATCH --cpus-per-task=N  
#SBATCH --mem-per-cpu=XGB  
singularity exec /path/to/container.simg python myscript.py
```

- *Note: The maximum number of CPUs per node (32) will not always give the maximum speedup*

SLURM - multi-task and multi-node jobs

- Can also specify tasks or tasks per node

```
#SBATCH --ntasks=N  
#SBATCH --cpus-per-task=1  
#SBATCH --mem=XGB  
/path/to/mpirun singularity exec /path/to/container.simg python myscript.py
```

- Above example doesn't require knowledge of number of node's CPUs; below one does

```
#SBATCH --nodes=1  
#SBATCH --ntasks-per-node=N  
#SBATCH --cpus-per-task=1  
#SBATCH --mem=XGB  
/path/to/mpirun singularity exec /path/to/container.simg python myscript.py
```

SLURM - multi-task and multi-CPU jobs

- Using multiple nodes with MPI

```
#SBATCH --nodes=N  
#SBATCH --ntasks-per-node=n  
#SBATCH --cpus-per-task=1  
#SBATCH --mem=XGB  
/path/to/mpirun singularity exec /path/to/container.simg python myscript.py
```

- *Note: Need to consider that internode communication is slower than intranode communication*
- --mem is memory per node, so N times XGB allocated overall (usable by some software)
- Using multiple nodes with MPI as well as multiple cores within node with OpenMP (utilised by myscript.py)

```
#SBATCH --ntasks=N  
#SBATCH --cpus-per-task=n  
/path/to/mpirun singularity exec /path/to/container.simg python myscript.py
```

SLURM – dependencies

- Allows jobs to be scheduled for running, based on the status of a previous job
 - e.g. only begin a particular job once previous one successfully completes

```
$ sbatch -d afterok:882242 --kill-on-invalid-dep=yes another_job.sh  
#submit another_job.sh to SLURM queue, to begin after jobID 882242  
successfully completes (exit code 0), or cancel the job if jobID 882242 fails
```

```
$ sbatch -d afterany:882242:882243 another_job.sh  
#submit another_job.sh to SLURM queue, to begin after jobIDs 882242 & 882243  
complete (any exit code)
```

IDIA MeerKAT Pipeline – A Good Framework

- Parallelised package for HPC processing (SLURM + cluster)
- HPC-friendly – dynamically uses resources & submits to queue to give calibrated data with the push of a button
- Each job/script is a logical step that does / doesn't use MPI, and optionally uses a different container
 - Managed by wrapper software sourced by user so that it's in their path
 - This could also be venv and is updated more regularly
 - Design: wrapper software manages the jobs you submit to SLURM
- User can insert their scripts at start, middle or end
 - Design: jobs run within containers that include software dependencies
- <https://idia-pipelines.github.io/docs/processMeerKAT>
- Demo time!

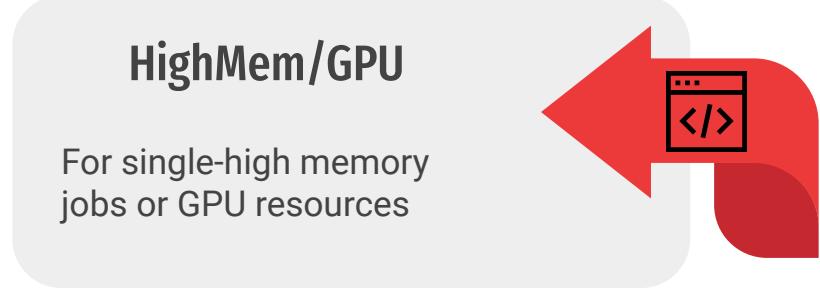
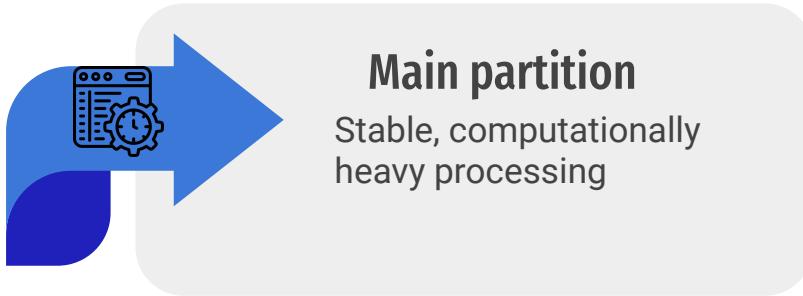
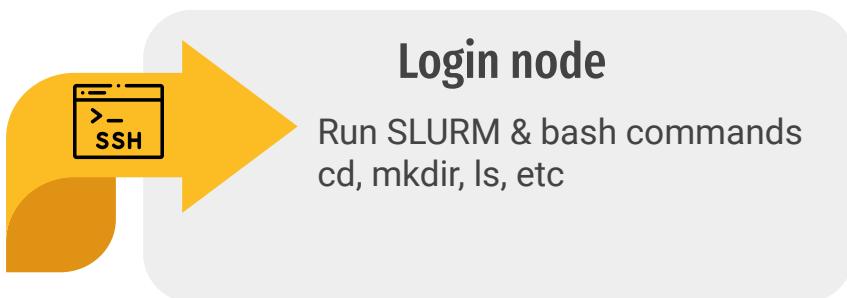
ilifu: a shared resource-limited cluster

- ilifu
 1. Supports a diverse range of projects
 - Astronomy and Bioinformatics
 - Varying resource requirements
 2. Shared environment
 3. Resource-limited
- Efficient use of resources essential
 - Practices laid out in [allocation guide](#)
 - Additional:
 - Select lowest Jupyter resource possible
 - Shut down Jupyter server after use
 - Use sbatch with non-default parameters



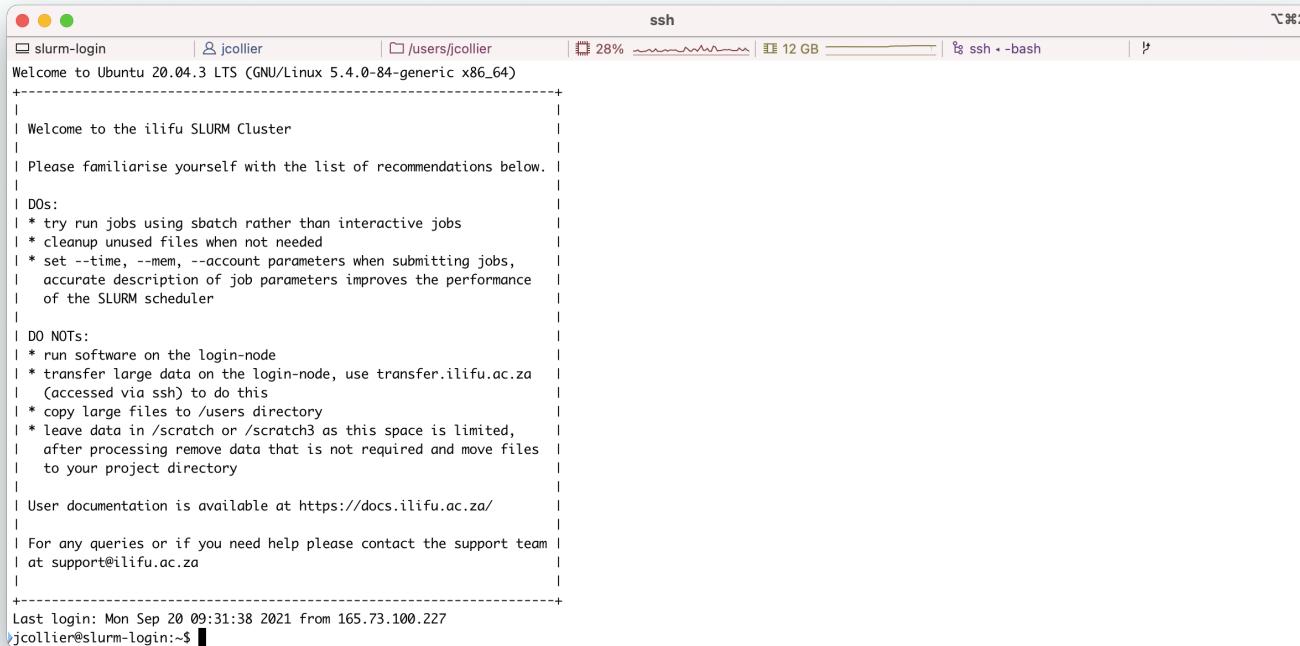
Services and partitions

- http://docs.ilifu.ac.za/#/getting_started/access_ilifu



Services and partitions

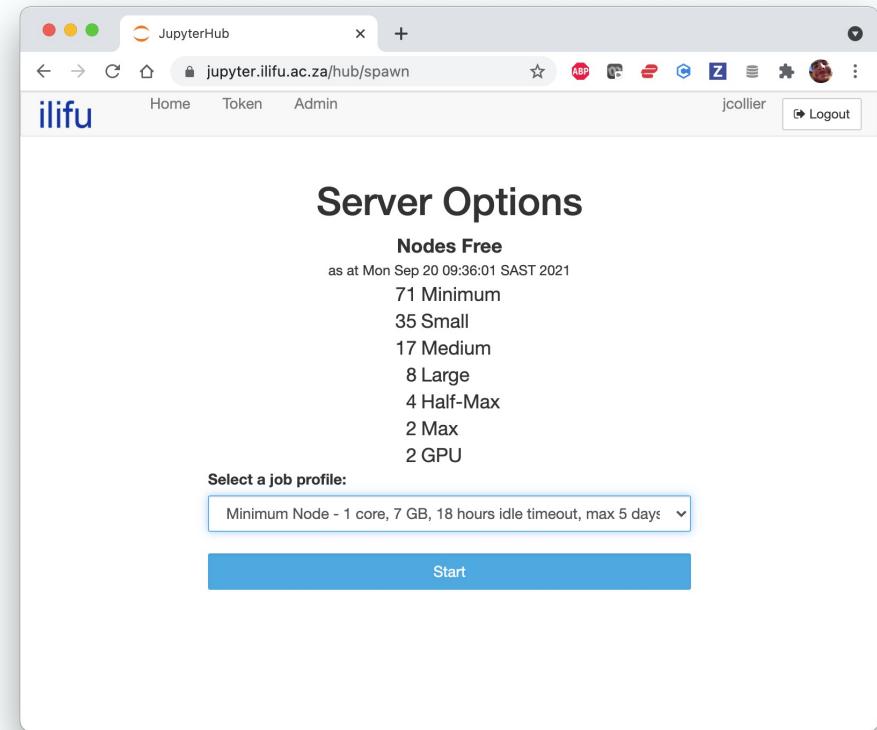
- Login node
 - Where you land when logging in on ilifu Slurm cluster (slurm.ilifu.ac.za)
 - For running basic bash commands (cd, mkdir, ls, etc)
 - For running Slurm commands (srun, sbatch, scancel, squeue, sacct, etc)



The screenshot shows an SSH session titled "ssh" connected to a host named "slurm-login". The window title bar includes the host name, user "jcollier", current directory "/users/jcollier", battery status (28%), disk space (12 GB), and the command being run ("ssh * -bash"). The main pane displays the "Welcome to the ilifu SLURM Cluster" message, which provides recommendations for using the cluster. It lists "DOs" such as running jobs with sbatch and setting parameters correctly, and "DO NOTs" like running software on the login-node or transferring large files via ssh. It also advises leaving data in /scratch or /scratch3. The message ends with documentation links and support contact information. At the bottom, it shows the last login details ("Mon Sep 20 09:31:38 2021 from 165.73.100.227") and the user prompt ("jcollier@slurm-login:~\$").

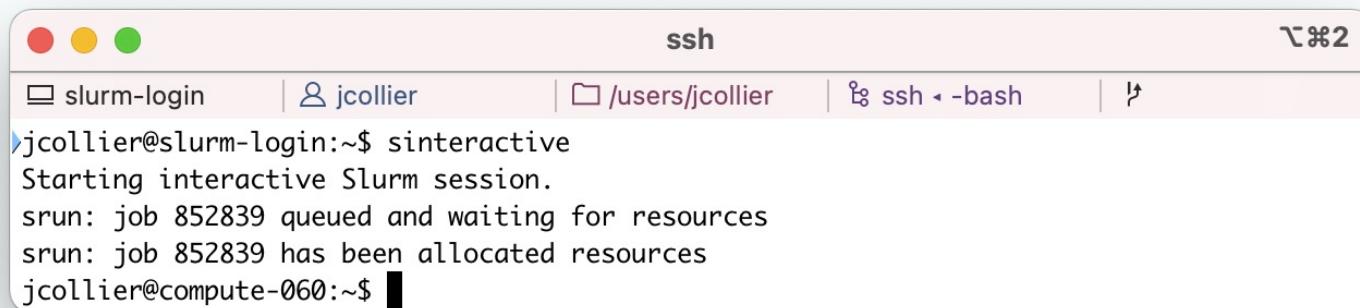
Services and partitions

- [Jupyter \(Jupyter.ilifu.ac.za\)](https://Jupyter.ilifu.ac.za)
 - Development space for writing, testing and debugging
 - New code, software, workflows or routines
 - Highly interactive Jupyter notebook environment
 - tab-completion, viewing doc strings, running subroutines within cells
 - May be primary interface for stable workflows that shouldn't use Slurm
 - short analysis routines or other highly interactive workflows



Services and partitions

- Devel (--partition=Devel)
 - Development of routines within shared resource environment
 - Submit jobs instantly / quickly
 - Resources shared, not solely allocated to your jobs
 - Interactivity via a shell
 - Generally for testing higher level workflows and pipelines
 - Access simply using the sinteractive command

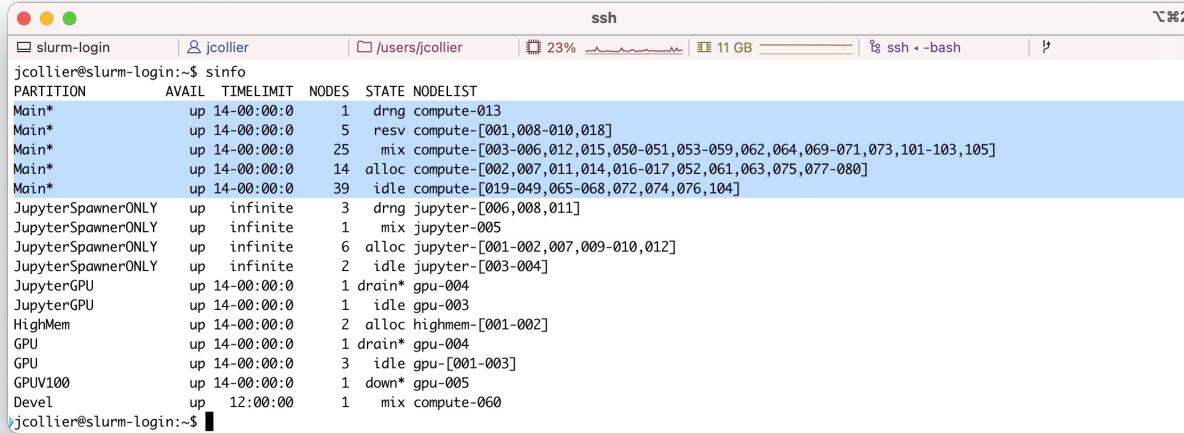


A screenshot of a terminal window titled "ssh". The window has three colored window controls (red, yellow, green) at the top left. The title bar says "ssh". At the top right, there is a "x" button. The terminal content shows:

```
jcollier@slurm-login:~$ sinteractive
Starting interactive Slurm session.
srun: job 852839 queued and waiting for resources
srun: job 852839 has been allocated resources
jcollier@compute-060:~$
```

Services and partitions

- Main partition
 - Default Slurm partition
 - Generally for stable, computationally-heavy workflows and pipelines
 - Many small jobs allocated few resources or
 - A few large jobs allocated many resources
 - Have first been tested on one of the previous services (where applicable)



```
slurm-login | jcollier | /users/jcollier | 23% | 11 GB | ssh -bash | 2

jcollier@slurm-login:~$ sinfo
PARTITION      AVAIL   TIMELIMIT  NODES  STATE NODELIST
Main*          up 14-00:00:0    1  drng compute-013
Main*          up 14-00:00:0    5  resv compute-[001,008-010,018]
Main*          up 14-00:00:0   25  mix compute-[003-006,012,015,050-051,053-059,062,064,069-071,073,101-103,105]
Main*          up 14-00:00:0   14  alloc compute-[002,007,011,014,016-017,052,061,063,075,077-080]
Main*          up 14-00:00:0   39  idle compute-[019-049,065-068,072,074,076,104]

JupyterSpawnerONLY up infinite   3  drng jupyter-[006,008,011]
JupyterSpawnerONLY up infinite   1  mix jupyter-005
JupyterSpawnerONLY up infinite   6  alloc jupyter-[001-002,007,009-010,012]
JupyterSpawnerONLY up infinite   2  idle jupyter-[003-004]
JupyterGPU        up 14-00:00:0   1  drain* gpu-004
JupyterGPU        up 14-00:00:0   1  idle gpu-003
HighMem           up 14-00:00:0   2  alloc highmem-[001-002]
GPU               up 14-00:00:0   1  drain* gpu-004
GPU               up 14-00:00:0   3  idle gpu-[001-003]
GPUV100          up 14-00:00:0   1  down* gpu-005
Devel             up 12:00:00   1  mix compute-060

jcollier@slurm-login:~$
```

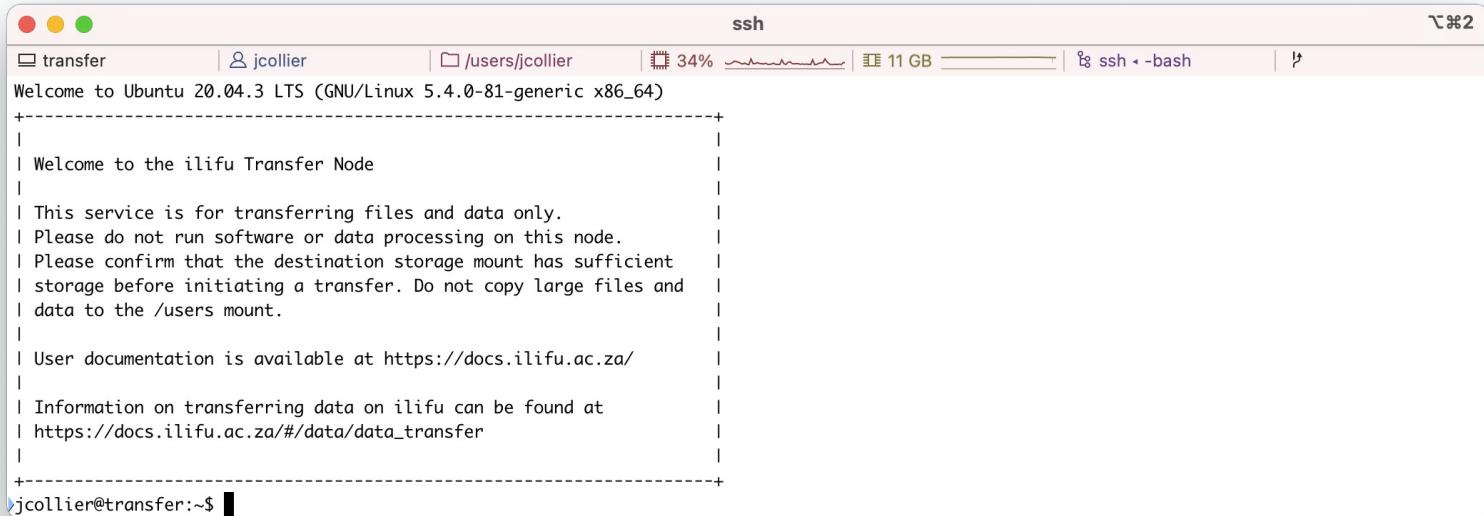
Services and partitions

- HighMem partition
 - Single high-memory jobs that can't be split into multiple jobs using MPI
- GPU partition
 - Jobs making use of GPUs
 - Not for jobs that only require CPUs (rather use Devel)

```
slurm-login | jcollier | /users/jcollier | 34% | 11 GB | ssh -bash | 
jcollier@slurm-login:~$ sinfo
PARTITION      AVAIL  TIMELIMIT  NODES  STATE NODELIST
Main*          up   14-00:00:0    1  drng compute-013
Main*          up   14-00:00:0    5  resv compute-[001,008-010,018]
Main*          up   14-00:00:0   25  mix compute-[003-006,012,015,050-051,053-059,062,064,069-071,073,101-103,105]
Main*          up   14-00:00:0   14  alloc compute-[002,007,011,014,016-017,052,061,063,075,077-080]
Main*          up   14-00:00:0   39  idle compute-[019-049,065-068,072,074,076,104]
JupyterSpawnerONLY up infinite   3  drng jupyter-[006,008,011]
JupyterSpawnerONLY up infinite   1  mix jupyter-005
JupyterSpawnerONLY up infinite   6  alloc jupyter-[001-002,007,009-010,012]
JupyterSpawnerONLY up infinite   2  idle jupyter-[003-004]
JupyterGPU       up   14-00:00:0   1  drain* gpu-004
JupyterGPU       up   14-00:00:0   1  idle gpu-003
HighMem         up   14-00:00:0   2  alloc highmem-[001-002]
GPU             up   14-00:00:0   1  drain* gpu-004
GPU             up   14-00:00:0   3  idle gpu-[001-003]
GPUV100         up   14-00:00:0   1  down* gpu-005
Devel           up   12:00:00   1  mix compute-060
jcollier@slurm-login:~$
```

Services and partitions

- Transfer node (transfer.ilifu.ac.za)
 - Internal and external copying of data (cp, scp, rsync, etc)
 - Smaller or less frequent transfers (i.e. not requiring Globus)
 - Other basic bash commands inappropriate for login node (wget, rm)
 - Also possible on Slurm compute node (e.g. 1 CPU, 1 GB RAM)



The screenshot shows a terminal window titled "ssh". The title bar also displays the user "jcollier", the command "ssh", and the session ID "ssh -bash". The terminal content is a welcome message for the ilifu Transfer Node:

```
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-81-generic x86_64)

Welcome to the ilifu Transfer Node

This service is for transferring files and data only.
Please do not run software or data processing on this node.
Please confirm that the destination storage mount has sufficient
storage before initiating a transfer. Do not copy large files and
data to the /users mount.

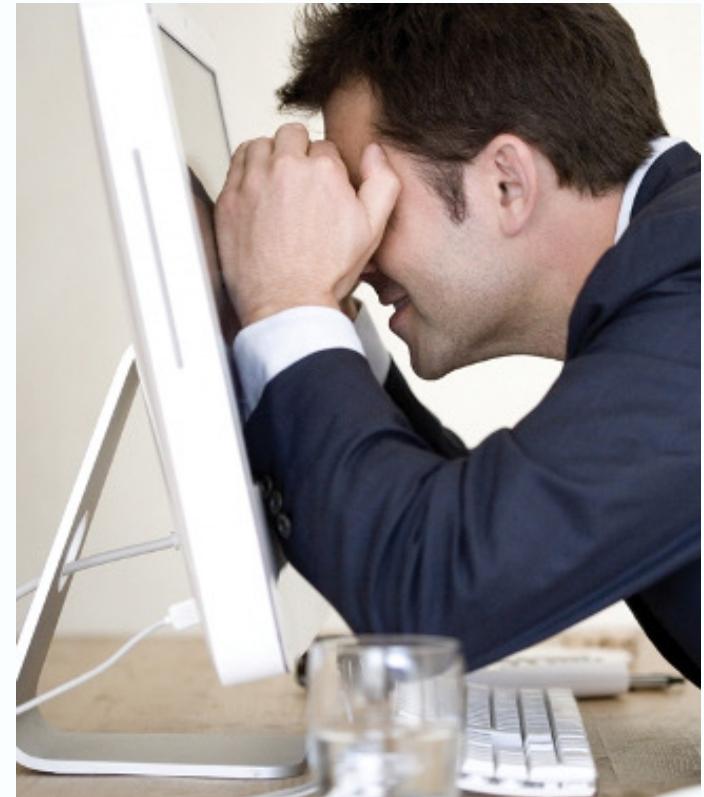
User documentation is available at https://docs.ilifu.ac.za/

Information on transferring data on ilifu can be found at
https://docs.ilifu.ac.za/#/data/data_transfer
```

The prompt at the bottom is "jcollier@transfer:~\$".

Allocating Resources

- http://docs.ilifu.ac.za/#/tech_docs/resource_allocation
- Primary resources
 1. CPU
 2. Memory
 3. Wall-time
- Notes
 - Nodes have 2 CPUs (sockets), each with 16 cores, all of which Slurm calls “CPUs”
 - Wall-time (elapsed time) is total run-time of job according to a clock on the wall
 - When > 1 CPU, differs from CPU time, measured in CPU hours



Allocating Resources

- How to allocate resources
 - Accurately determine your resource requirements
 - Use what you require
- Effect
 - Avoid wasting resources (allocated but not used)
 - Increase resource availability
 - Allow other (users') jobs to run
 - Improves efficiency of Slurm scheduler
 - Increase your [fairshare](#) priority
 - Potentially decrease your job wait times

Allocating Resources

- Determine your resource requirements
 1. Determining parallelism of software
 2. Profiling previous similar jobs
 3. Scaling up test jobs



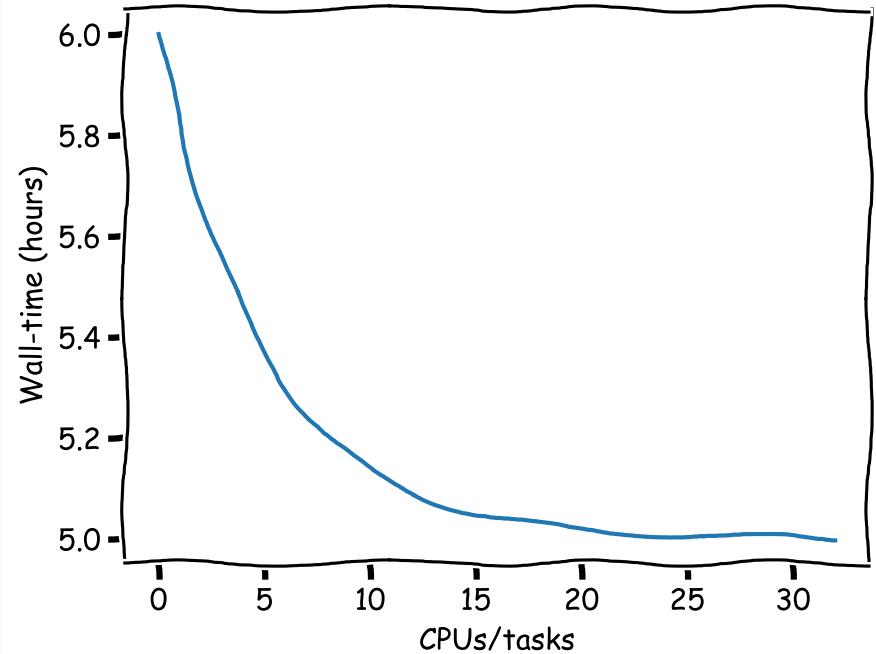
Allocating Resources

- Determining parallelism of software
 - See previous slides
 - CPU-level vs. task-level parallelism
 - Many software packages only use 1 CPU



Allocating Resources

- Determining parallelism of software
 - Most parallel processing software doesn't scale linearly
 - Maximum performance often least efficient
 - i.e. shortest wall-time but large allocation necessary
 - Need to find middle ground
 - MPI jobs may perform worse for larger allocations (scatter/gather)
 - Most efficient generally to break into many small independent jobs
 - High-throughput approach



Profiling previous similar jobs

- Find job ID
 - Output when job submitted
 - Can search for historical jobs
 - Display jobs named ‘my-job’ submitted during particular time range:
 - `sacct -X --name=my-job --starttime=YYYY-MM-DD --endtime=YYYY-MM-DD`
 - Omit job name (or end time) to show all jobs
 - Add following to query (very) old Slurm databases (before upgrades)
 - `--cluster=ilifu-slurm20` or `--cluster=ilifu-slurm`
- Once you have job ID, you can search for specific information about resource usage

Profiling previous similar jobs

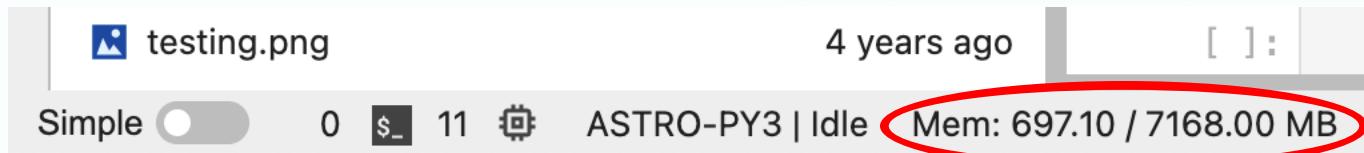
- Memory usage
 - Find MaxRSS statistic
 - Maximum memory usage of a job (sampled every 20 seconds)
 - Display MaxRSS for job ID 123456 compared to requested memory
 - `sacct -j 123456 --unit=G -o JobID,JobName,MaxRSS,ReqMem`
 - Can run this from Jupyter terminal (to determine resource selection)
 - *Notes: 232 Gn = 232 GB per node; 7.25c = 7.25 GB per CPU*
 - Once memory requirement determined
 - Schedule future jobs with ~10-20% buffer
 - Avoids out-of-memory (OOM) error
 - Avoid excessive usage of memory
 - e.g. minimum node in Jupyter

| JobID | JobName | MaxRSS | ReqMem |
|---------------|------------|---------|--------|
| 847197 | selfcal_p+ | 232Gn | |
| 847197.batch | batch | 213.77G | 232Gn |
| 847197.extern | extern | 0 | 232Gn |

An aside for Jupyter

- Select job profile to match your requirements (avoid excess)
- Jupyter shows you maximum memory usage down the bottom of your session
- We will email you usage stats after a job has completed that selected a high job profile and used very low CPU & memory
- Shut down your session

The screenshot shows a web browser window titled "JupyterHub" with the URL "jupyter.ilifu.ac.za/hub/spawn/jcollier". The page displays "ilifu" branding at the top. Below it, the "Server Options" section shows the following data:
Nodes Free (as at Mon Apr 11 20:44:01 SAST 2022):
221 Minimum
110 Small
54 Medium
26 Large
11 Half-Max
5 Max
4 GPU
A dropdown menu titled "Select a job profile:" is open, showing the following options:
✓ Minimum Node - 1 core, 7 GB, 18 hours idle timeout, max 5 days lifespan
Small Node - 2 core, 14 GB, 18 hours idle timeout, max 5 days lifespan
Medium Node - 4 core, 28 GB, 18 hours idle timeout, max 5 days lifespan
Large Node - 8 core, 58 GB, 18 hours idle timeout, max 5 days lifespan
Half-Max Node - 16 core, 116 GB, 18 hours idle timeout, max 5 days lifespan
Max Node - 32 core, 232 GB, 18 hours idle timeout, max 5 days lifespan
GPU Node - NVIDIA P100 GPU, 16 core, 116 GB Memory, max 8 hour lifespan



Profiling previous similar jobs

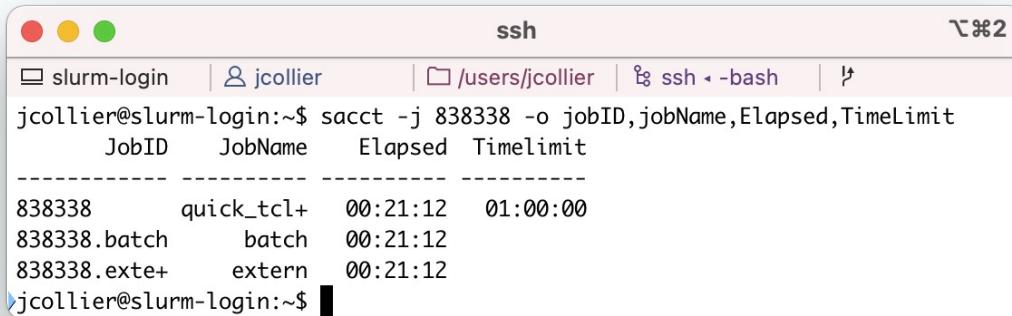
- CPU (and memory) usage
 - Determine used vs. allocated/requested
 - Show Slurm resource efficiency (seff) for job ID 123456
 - Shows % used vs. allocated (for memory, uses MaxRSS stat)
 - `seff 123456`
 - Can run this from Jupyter terminal (to determine resource selection)

```
ssh
jcollier@slurm-login:~$ seff 847197
Job ID: 847197
Cluster: ilifu-slurm2021
User/Group: jcollier/idia-group
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 32
CPU Utilized: 1-15:22:40
CPU Efficiency: 71.93% of 2-06:44:48 core-walltime
Job Wall-clock time: 01:42:39
Memory Utilized: 213.77 GB
Memory Efficiency: 92.14% of 232.00 GB
```

```
ssh
jcollier@slurm-login:~$ seff 201280
Job ID: 201280
Cluster: ilifu-slurm2021
User/Group: jcollier/idia-group
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 32
CPU Utilized: 00:00:09
CPU Efficiency: 1.17% of 00:12:48 core-walltime
Job Wall-clock time: 00:00:24
Memory Utilized: 519.09 MB
Memory Efficiency: 0.22% of 232.00 GB
```

Profiling previous similar jobs

- Wall-time usage
 - Accurate estimation improves Slurm scheduler efficiency and may reduce your job wait time
 - Show used vs. requested wall-time for job ID 123456 (also in Jupyter)
 - `sacct -o jobID,jobName,Elapsed,TimeLimit`
 - Once wall-time requirement determined
 - Schedule future jobs with ~20-30% buffer (avoids job timing out)
 - Avoid excessive wall-time
 - Contact support@ilifu.ac.za to see if we may increase your time limit



```
ssh
slurm-login | jcollier | /users/jcollier | ssh -bash | ↴
jcollier@slurm-login:~$ sacct -j 838338 -o jobID,jobName,Elapsed,TimeLimit
      JobID   JobName   Elapsed  Timelimit
-----
838338     quick_tcl+  00:21:12  01:00:00
838338.batch      batch  00:21:12
838338.exten+    extern  00:21:12
jcollier@slurm-login:~$
```

Scaling tests

- Accurately estimating wall-time difficult to do
- Profile previous similar jobs, or
- Run test / scaling jobs
 - Start small test job (e.g. small allocation on small subset of data)
 - Test the wall-time, run again with increased resources
 - Reasonable to over-allocate when running scaling test
 - Briefly inefficient, until get an idea of requirements
 - Or if under-estimate, and test small enough, doesn't matter if crashes
 - Repeat process to see how resource usage scales
 - as a function of input (e.g. data volume)
 - as a function of CPUs / tasks (if doing parallel processing)
 - By the end, should have good idea of scaling and efficient choice
 - Allow for buffer for future jobs

Usage of running jobs

- e.g. during scaling tests
- Get MaxRSS for running job
 - `sstat -j 123456 -o MaxRSS`
 - Given in kB units. Divide by 1024^2 for GB
- Display real time stats on dashboard (top / htop)
 - ssh compute-001 or open Jupyter terminal
 - Requires job running on node and authentication forwarding
 - e.g. first run `ssh -A <username>@slurm.ilifu.ac.za`
 - `htop -u $USER`
- Shows different (e.g. master and spawned) running processes
- Can monitor real-time usage

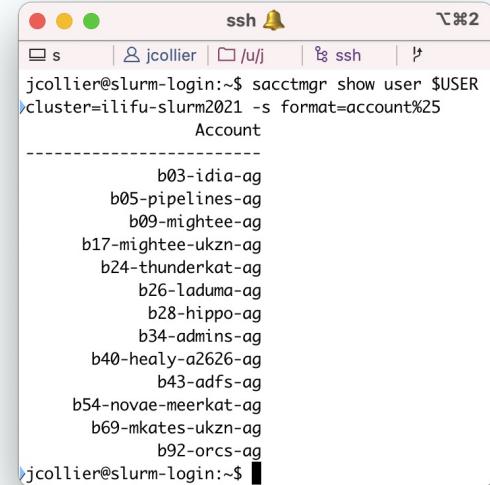
Maximum Resources

- If using all CPUs or memory, node becomes fully allocated
 - Any remaining CPUs / memory unavailable to other jobs (incl. your own)
 - Consider leaving headroom when can't use all CPUs or memory
- *Note: Jobs on Devel node cannot allocate memory*

| Partition | Node names | Default CPUs | Max CPUs | Default Memory (GB) | Max Memory (GB) | Default wall-time | Max wall-time |
|-----------|-------------------|--------------|----------|---------------------|-----------------|-------------------|---------------|
| Main | compute-[001-080] | 1 | 32 | 7.25 | 232 | 3 hours | 14 days |
| Main | compute-[101-105] | 1 | 48 | 7.25 | 232 | 3 hours | 14 days |
| HighMem | highmem-[001-002] | 1 | 32 | 15 | 480 | 3 hours | 14 days |
| Devel | compute-060 | 1 | 32 | - | - | 3 hours | 12 hours |

Account allocation

- Each ilifu project has a [Slurm account](#)
- Resource usage charged against account (affects [fairshare](#))
- View your accounts
 - ```
sacctmgr show user $USER cluster=ilifu-slurm2021 -s format=account%25
```
- View your default account
  - ```
sacctmgr show user $USER
```
- Change default
 - ```
sacctmgr modify user name=${USER} set DefaultAccount=<account>
```
- Set account (after #SBATCH for sbatch jobs)
  - `--account=b05-pipelines-ag`



```
jcollier@slurm-login:~$ sacctmgr show user $USER
cluster=ilifu-slurm2021 -s format=account%25
Account

b03-idia-ag
b05-pipelines-ag
b09-mightee-ag
b17-mightee-ukzn-ag
b24-thunderkat-ag
b26-laduma-ag
b28-hippo-ag
b34-admins-ag
b40-healy-a2626-ag
b43-adfs-ag
b54-novae-meerkat-ag
b69-mkates-ukzn-ag
b92-orcs-ag
jcollier@slurm-login:~$
```

# Resource Allocation Guide

---

- Demo

# Best practices

---

- Don't run software / heavy processes / scp on the login node
  - Only submit jobs and run SLURM commands (sbatch, srun, squeue, etc)
  - Use transfer.ilifu.ac.za to transfer data (external/internal), not login node
- Before running a large job, identify the available resources
  - Use sinfo. Don't hog the cluster. Reduce your allocation if possible
  - Increase likelihood of jobs running with less memory and less walltime
- Use sbatch (srun / screen / tmux / mosh are volatile)
- Cleanup files that aren't needed
  - Old raw data, temporary products, /scratch data, etc
- Don't place large files in your home directory (/users)
- Use Singularity (you cannot install software on the nodes)

# THANK YOU

Dr Jordan Collier

Postdoctoral Support Astronomer, IDIA  
Department of Astronomy,  
University of Cape Town

Adjunct Fellow, School of Science,  
Western Sydney University

[Jordan@idia.ac.za](mailto:Jordan@idia.ac.za)



Inter-University Institute  
for Data Intensive Astronomy



UNIVERSITY OF CAPE TOWN  
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

