# Hey look we have a cluster now isn't that cool what does it do does it do stuff yes it does it's cool

## Doing cluster stuff for fun and ~~profit~~ science

**Lukas Burk**

Leibniz Institute for Prevention Research and Epidemiology — BIPS

2026-04-07

BioWiMium

# We have a new cluster

## You might have *heard*



## What we cover today:

- What is this cluster thing
- How does it work (using R / Python, Slurm)
- How do **you** do stuff on it in general

# We have a new cluster

## You might have *heard*



### What we cover today:

- What is this cluster thing
- How does it work (using R / Python, Slurm)
- How do **you** do stuff on it in general

### We don't cover

- How to Linux (from the command line)
- How to use `batchtools` (or `targets`)
- Everything you ever need to know

# We have a new cluster

## You might have *heard*



### What we cover today:

- What is this cluster thing
- How does it work (using R / Python, Slurm)
- How do **you** do stuff on it in general

### We don't cover

- How to Linux (from the command line)
- How to use `batchtools` (or `targets`)
- Everything you ever need to know

> 😭 I know it's noisy and no I don't know when that is fixed

# What is a cluster?

## And why should you care



Many ways to shave a yak, and now you have one more!

# What is a cluster?

## And why should you care



Many ways to shave a yak, and now you have one more!

## Your PC/laptop

- You know it
- Shorter tasks
- Not so powerful
- Convenience: 💅

# What is a cluster?

## And why should you care



Many ways to shave a yak, and now you have one more!

### Your PC/laptop

- You know it
- Shorter tasks
- Not so powerful
- Convenience: 💅

### Workstation / Server

- Bertha / new GePaRD server
- Run looong tasks
- 1 powerful computer
- Shared
- Convenience: Mixed

# What is a cluster?

## And why should you care



Many ways to shave a yak, and now you have one more!

### Your PC/laptop

- You know it
- Shorter tasks
- Not so powerful
- Convenience: 💅

### Workstation / Server

- Bertha / new GePaRD server
- Run looong tasks
- 1 powerful computer
- Shared
- Convenience: Mixed

### HPC Cluster

- Many computers connected
- Access via *head node*
- Shared (scheduling: *Slurm*)

# What is a cluster?

## And why should you care



Many ways to shave a yak, and now you have one more!

### Your PC/laptop

- You know it
- Shorter tasks
- Not so powerful
- Convenience: 💅

### Workstation / Server

- Bertha / new GePaRD server
- Run looong tasks
- 1 powerful computer
- Shared
- Convenience: Mixed

### HPC Cluster

- Many computers connected
- Access via *head node*
- Shared (scheduling: *Slurm*)
- Convenience: You'll get there

# What does it look like?

## Kind of like this

# What does it look like?

## Kind of like this



# The HPC workflow

## It's a freight truck, not a bicycle



You don’t hop on the cluster for a quick ride, you **submit work** to it

# The HPC workflow

## It's a freight truck, not a bicycle



You don’t hop on the cluster for a quick ride, you **submit work** to it

### What you are used to

- Open RStudio / VSCode / Positron
- Write code, run it, see results
- Everything is *interactive*
- One computer does it all

# The HPC workflow

## It's a freight truck, not a bicycle



You don’t hop on the cluster for a quick ride, you **submit work** to it

### What you are used to

- Open RStudio / VSCode / Positron
- Write code, run it, see results
- Everything is *interactive*
- One computer does it all

### How HPC works

- Prepare your code on the *head node*
- **Submit** it as a job (it gets *queued*)
- Go make coffee ☕
- Come back, check results
- Debug if needed, resubmit

# The HPC workflow

## It's a freight truck, not a bicycle



You don’t hop on the cluster for a quick ride, you **submit work** to it

### What you are used to

- Open RStudio / VSCode / Positron
- Write code, run it, see results
- Everything is *interactive*
- One computer does it all

### How HPC works

- Prepare your code on the *head node*
- **Submit** it as a job (it gets *queued*)
- Go make coffee ☕
- Come back, check results
- Debug if needed, resubmit

> 💡 There’s a learning curve, but the payoff is huge

# HPC Workflow

## The typical cycle



1. Log in (land on **head node**)

# HPC Workflow

## The typical cycle



1. Log in (land on **head node**)
2. Load software (`module load R/4.5.3`)

# HPC Workflow

## The typical cycle



1. Log in (land on **head node**)
2. Load software (`module load R/4.5.3`)
3. Move to your project (`git` helps)

# HPC Workflow

## The typical cycle



1. Log in (land on **head node**)
2. Load software (`module load R/4.5.3`)
3. Move to your project (`git` helps)
4. Submit your **job**(s) via `sbatch` (or `salloc` for interactive work)

# HPC Workflow

## The typical cycle



1. Log in (land on **head node**)
2. Load software (`module load R/4.5.3`)
3. Move to your project (`git` helps)
4. Submit your **job**(s) via `sbatch` (or `salloc` for interactive work)
5. Monitor with `squeue --me`

# HPC Workflow

## The typical cycle



1. Log in (land on **head node**)
2. Load software (`module load R/4.5.3`)
3. Move to your project (`git` helps)
4. Submit your **job**(s) via `sbatch` (or `salloc` for interactive work)
5. Monitor with `squeue --me`
6. Log out, check back later

# HPC Workflow

## The typical cycle



1. Log in (land on **head node**)
2. Load software (`module load R/4.5.3`)
3. Move to your project (`git` helps)
4. Submit your **job**(s) via `sbatch` (or `salloc` for interactive work)
5. Monitor with `squeue --me`
6. Log out, check back later
7. If jobs failed, debug & resubmit

# HPC Workflow

## The typical cycle

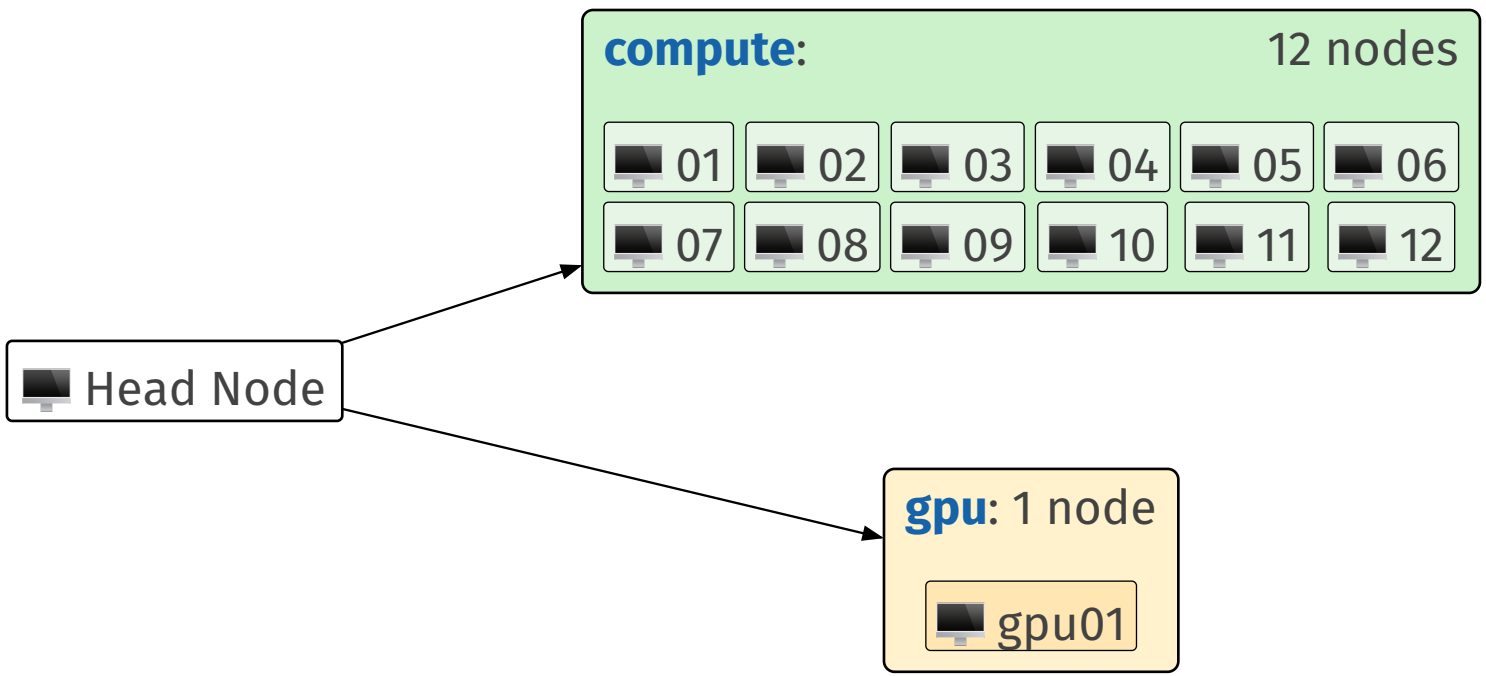


1. Log in (land on **head node**)
2. Load software (`module load R/4.5.3`)
3. Move to your project (`git` helps)
4. Submit your **job**(s) via `sbatch` (or `salloc` for interactive work)
5. Monitor with `squeue --me`
6. Log out, check back later
7. If jobs failed, debug & resubmit
8. Repeat until ~~insane~~ done

# Cluster topology

## Partitions

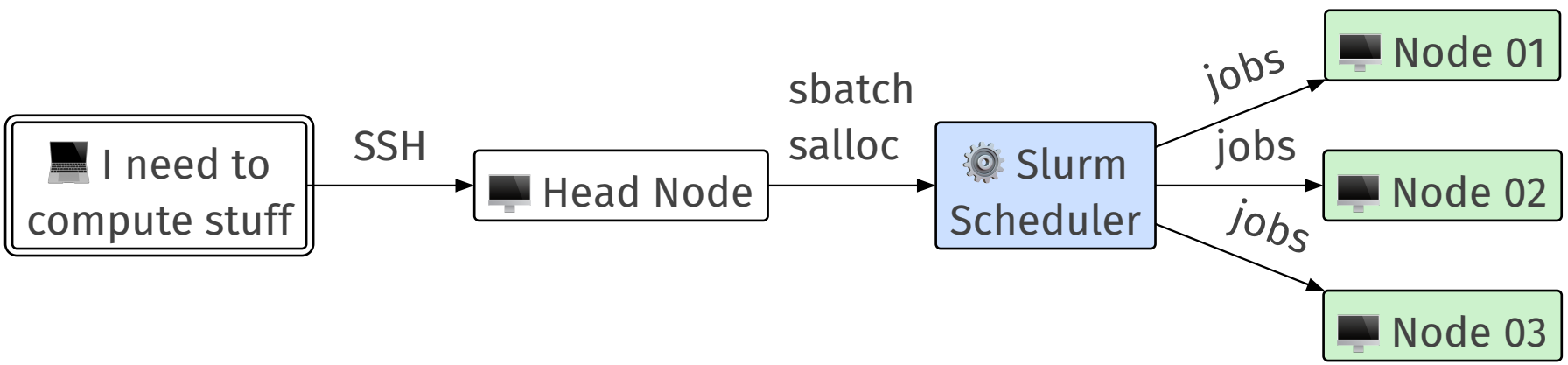




# Cluster topology

## It only looks weird because it is





# The head node

## Your gateway



The head node is where you land when you log in

# The head node

## Your gateway



The head node is where you land when you log in

### Do ✓

- Install R / Python packages
- Edit scripts and code
- Submit and monitor jobs
- Use `git`

# The head node

## Your gateway



The head node is where you land when you log in

### Do ✓

- Install R / Python packages
- Edit scripts and code
- Submit and monitor jobs
- Use `git`

### Don't ❌

- Run heavy computations
- Launch long-running scripts
- Load huge datasets into memory

# The head node

## Your gateway



The head node is where you land when you log in

### Do ✓

- Install R / Python packages
- Edit scripts and code
- Submit and monitor jobs
- Use `git`

### Don't ❌

- Run heavy computations
- Launch long-running scripts
- Load huge datasets into memory

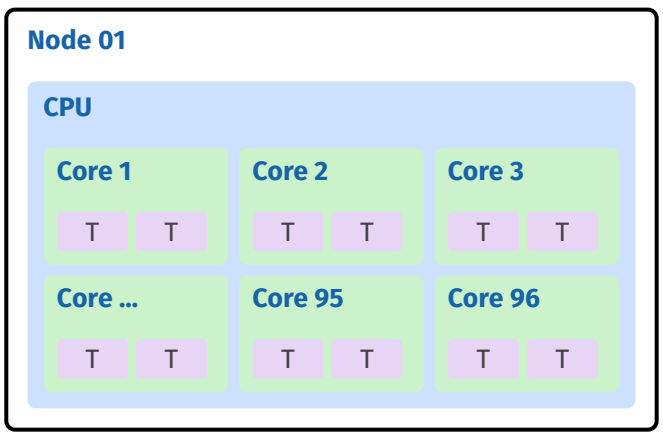
⚠ The head node is shared by *everyone*. Heavy processes will be *terminated*.

# Terminology

## Just to make sure you're confused



- The cluster has 12 compute **nodes** (self-contained computer)

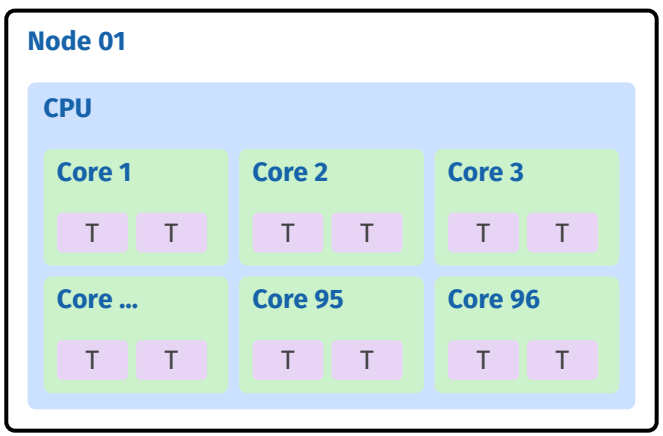


# Terminology

## Just to make sure you're confused



- The cluster has 12 compute **nodes** (self-contained computer)
  - Each node has 1 **CPU** (physical processor)

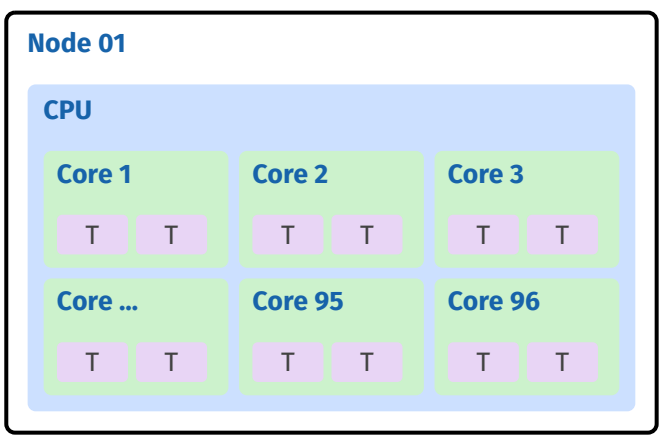


# Terminology

## Just to make sure you're confused



- The cluster has 12 compute **nodes** (self-contained computer)
  - Each node has 1 **CPU** (physical processor)
  - Each processor is made up of 96 **cores** (independent units)

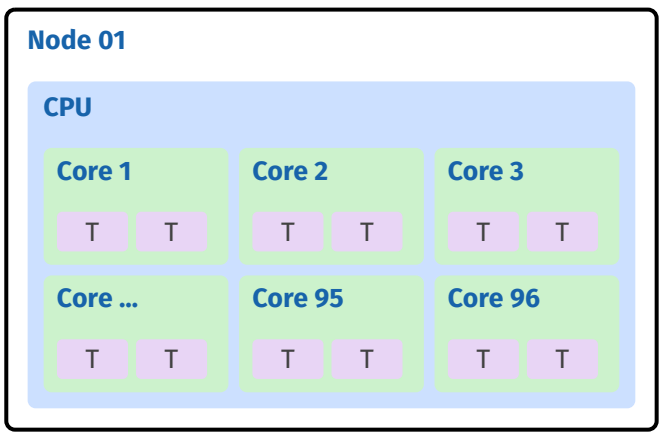


# Terminology

## Just to make sure you're confused



- The cluster has 12 compute **nodes** (self-contained computer)
  - Each node has 1 **CPU** (physical processor)
  - Each processor is made up of 96 **cores** (independent units)
  - Each core can juggle 2 **threads** (e.g. R processes)

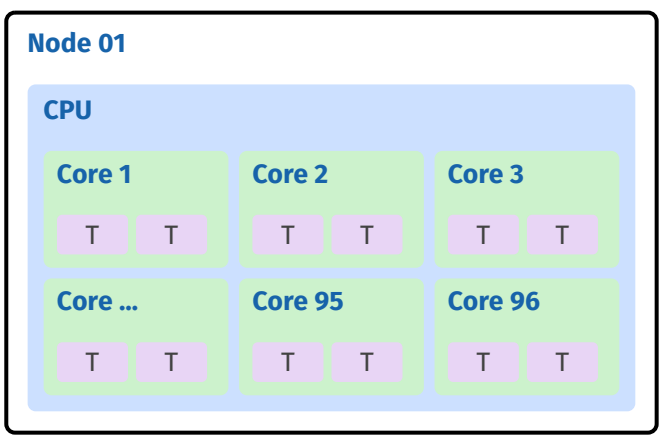


# Terminology

## Just to make sure you're confused



- The cluster has 12 compute **nodes** (self-contained computer)
  - Each node has 1 **CPU** (physical processor)
  - Each processor is made up of 96 **cores** (independent units)
  - Each core can juggle 2 **threads** (e.g. R processes)
- Terminology here weird / confusing because: *history* 🤦
  - Until the 90s: **1** CPU = **1** core = **1** thread

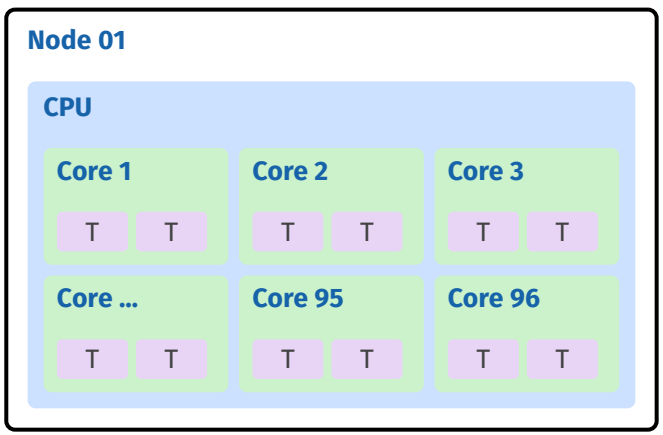


# Terminology

## Just to make sure you're confused



- The cluster has 12 compute **nodes** (self-contained computer)
  - Each node has 1 **CPU** (physical processor)
  - Each processor is made up of 96 **cores** (independent units)
  - Each core can juggle 2 **threads** (e.g. R processes)
- Terminology here weird / confusing because: *history* 🤦
  - Until the 90s: **1** CPU = **1** core = **1** thread
  - Then: “Hyperthreading”: **1** CPU = **1** core = **2** threads

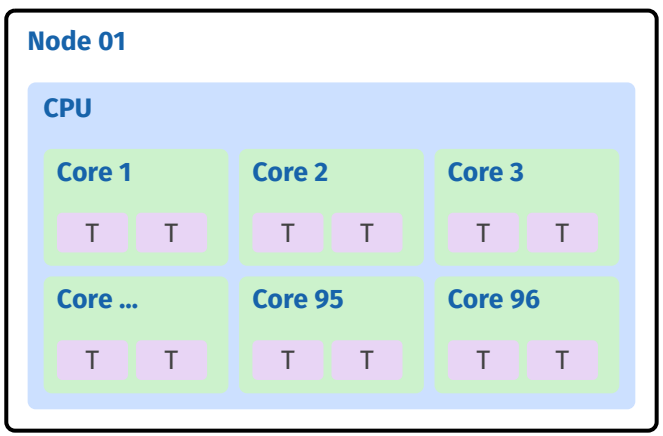


# Terminology

## Just to make sure you're confused



- The cluster has 12 compute **nodes** (self-contained computer)
  - Each node has 1 **CPU** (physical processor)
  - Each processor is made up of 96 **cores** (independent units)
  - Each core can juggle 2 **threads** (e.g. R processes)
- Terminology here weird / confusing because: *history* 🤦
  - Until the 90s: **1** CPU = **1** core = **1** thread
  - Then: “Hyperthreading”: **1** CPU = **1** core = **2** threads
  - Then: “Dual core”: **1** CPU = **2** cores = **4** threads

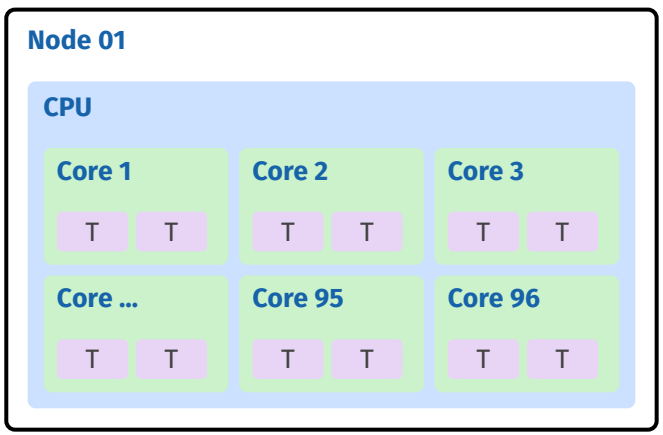


# Terminology

## Just to make sure you're confused



- The cluster has 12 compute **nodes** (self-contained computer)
  - Each node has 1 **CPU** (physical processor)
  - Each processor is made up of 96 **cores** (independent units)
  - Each core can juggle 2 **threads** (e.g. R processes)
- Terminology here weird / confusing because: *history* 🤦
  - Until the 90s: **1** CPU = **1** core = **1** thread
  - Then: “Hyperthreading”: **1** CPU = **1** core = **2** threads
  - Then: “Dual core”: **1** CPU = **2** cores = **4** threads
  - **Now**: **1** CPU = **96** cores = **192** threads

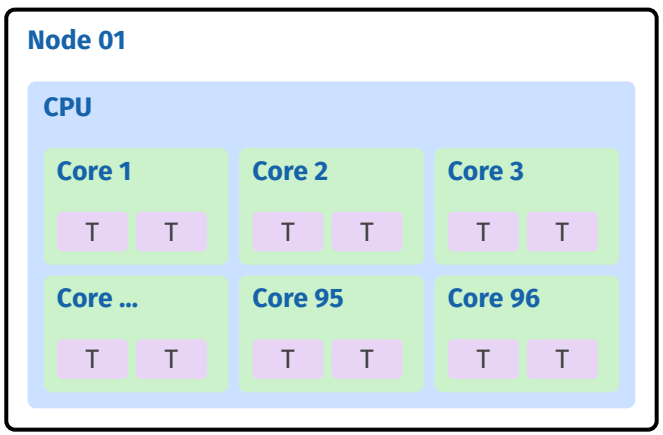


# Terminology

## Just to make sure you're confused



- The cluster has 12 compute **nodes** (self-contained computer)
  - Each node has 1 **CPU** (physical processor)
  - Each processor is made up of 96 **cores** (independent units)
  - Each core can juggle 2 **threads** (e.g. R processes)
- Terminology here weird / confusing because: *history* 🤦
  - Until the 90s: **1** CPU = **1** core = **1** thread
  - Then: “Hyperthreading”: **1** CPU = **1** core = **2** threads
  - Then: “Dual core”: **1** CPU = **2** cores = **4** threads
  - **Now**: **1** CPU = **96** cores = **192** threads




⚠ **Slurm**: You need 10 **threads** → you request 10 **“cpus”**

# But what is Slurm even



- Slurm is the job management system (see slurm.schedmd.com)
- Every CPU *core*, every Byte of *RAM* on the compute nodes is *kept track of*

# But what is Slurm even



- Slurm is the job management system (see slurm.schedmd.com)
- Every CPU *core*, every Byte of *RAM* on the compute nodes is *kept track of*

→ You can only use resources allocated to you

# But what is Slurm even



- Slurm is the job management system (see slurm.schedmd.com)
- Every CPU *core*, every Byte of *RAM* on the compute nodes is *kept track of*

→ You can only use resources allocated to you

**Things that *do not* happen on a cluster:**

- “My job took too much RAM so your jobs got killed alongside mine sorry”

# But what is Slurm even



- Slurm is the job management system (see slurm.schedmd.com)
- Every CPU *core*, every Byte of *RAM* on the compute nodes is *kept track of*

→ You can only use resources allocated to you

**Things that *do not* happen on a cluster:**

- “My job took too much RAM so your jobs got killed alongside mine sorry”
- “I used 1000 threads but only meant to use 10 sorry your jobs are smothered”

# But what is Slurm even



- Slurm is the job management system (see slurm.schedmd.com)
- Every CPU *core*, every Byte of *RAM* on the compute nodes is *kept track of*

→ You can only use resources allocated to you

**Things that *do not* happen on a cluster:**

- “My job took too much RAM so your jobs got killed alongside mine sorry”
- “I used 1000 threads but only meant to use 10 sorry your jobs are smothered”

→ It’s impossible to “endanger” other people’s compute jobs

# Logging in

## Different environments for different tasks



Connect to the cluster **head node** via SSH in one of two ways:

# Logging in

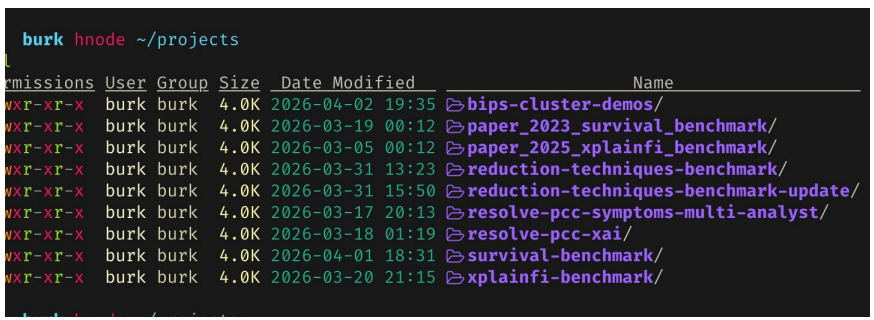
## Different environments for different tasks



Connect to the cluster **head node** via SSH in one of two ways:

## Terminal

- Definitely cool and normal
- 1337 h4xX0r feeling



# Logging in

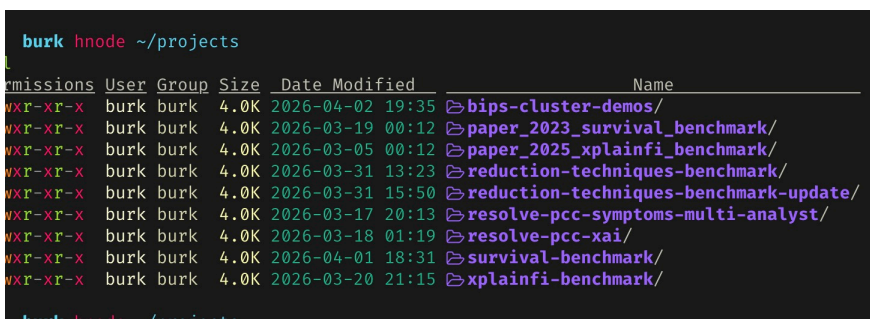
## Different environments for different tasks



Connect to the cluster **head node** via SSH in one of two ways:

### Terminal

- Definitely cool and normal
- 1337 h4xX0r feeling




### Positron (VScode)

- Uses SSH for communication with head node
- GUI on your own device (PC, laptop)
- Terminal / R console runs **on head node**
- All files you see are on the head node
- 😎 Edit files conveniently

# The first thing you see when you log in

## Trying to point you in the right direction



# Software is managed differently

## Environment modules



- Common on HPC systems: Hundreds of users need dozens of programs
- Programs exist in dozens of versions (R, Python, specialized tools)

# Software is managed differently

## Environment modules



- Common on HPC systems: Hundreds of users need dozens of programs
- Programs exist in dozens of versions (R, Python, specialized tools)
- Different users need different versions of different things

# Software is managed differently

## Environment modules



- Common on HPC systems: Hundreds of users need dozens of programs
- Programs exist in dozens of versions (R, Python, specialized tools)
- Different users need different versions of different things

## Solution: Environment modules

- Log in on head node: R not available
- Run `module load R/4.5.3` → R v4.5.3 is available
- Load modules **before** running `salloc` or `sbatch` — they inherit your environment

# Software is managed differently

## Environment modules



- Common on HPC systems: Hundreds of users need dozens of programs
- Programs exist in dozens of versions (R, Python, specialized tools)
- Different users need different versions of different things

## Solution: Environment modules

- Log in on head node: R not available
- Run `module load R/4.5.3` → R v4.5.3 is available
- Load modules **before** running `salloc` or `sbatch` — they inherit your environment
- (Python: Use `uv`, no need for modules: (docs.astral.sh/uv))

# How are resources allocated?

## Slurm manages resources on the cluster



- Example: You need **20 threads** for **6 hours** and **4GB of RAM** for `analysis.R`

# How are resources allocated?

## Slurm manages resources on the cluster



- Example: You need **20 threads** for **6 hours** and **4GB of RAM** for `analysis.R`

## Interactive: `salloc`

```
salloc --cpus-per-task=20 --mem=4G --time=06:00:00
```

```
salloc: Granted job allocation 137031
salloc: Nodes node01 are ready for job

# start R session
R

# Do work or run script
ranger::ranger(foo ~ ., data = bar)
source("analysis.R")
```

# How are resources allocated?

## Slurm manages resources on the cluster



- Example: You need **20 threads** for **6 hours** and **4GB of RAM** for `analysis.R`

## Interactive: `salloc`

```
salloc --cpus-per-task=20 --mem=4G --time=06:00:00
```

```
salloc: Granted job allocation 137031
salloc: Nodes node01 are ready for job

# start R session
R

# Do work or run script
ranger::ranger(foo ~ ., data = bar)
source("analysis.R")
```

## Batch job: `sbatch`

- Write an `analysis.sh` wrapper script

```
#!/bin/bash
#SBATCH --job-name=my-analysis
#SBATCH --output=logs/%x_%j.out
#SBATCH --cpus-per-task=20
#SBATCH --mem=4G
#SBATCH --time=06:00:00

Rscript analysis.R
```

- Run `sbatch analysis.sh` and go home

# Slurm resource allocation

## Partitions and QoS



- Slurm groups hardware in *partitions*
- 2 partitions: `compute` (12 nodes, *default*) and `gpu` (1 node)
- Prioritization is done in *Quality of Service* (QoS) queues:

# Slurm resource allocation

## Partitions and QoS



- Slurm groups hardware in *partitions*
- 2 partitions: `compute` (12 nodes, *default*) and `gpu` (1 node)
- Prioritization is done in *Quality of Service* (QoS) queues:
- Different queues have different constraints (time, how many jobs per user)
  - **interactive**: 3 days, 2 per user (auto-applies to `salloc`)

# Slurm resource allocation

## Partitions and QoS



- Slurm groups hardware in *partitions*
- 2 partitions: `compute` (12 nodes, *default*) and `gpu` (1 node)
- Prioritization is done in *Quality of Service* (QoS) queues:
- Different queues have different constraints (time, how many jobs per user)
  - **interactive**: 3 days, 2 per user (auto-applies to `salloc`)
- For `sbatch` or `#SBATCH --qos=<name>`:

**short**: 1 hour **medium** (default): 1 day **long**: 7 days

# Slurm resource allocation

## A job is not a job



- Slurm is very generic and supports diverse types of jobs
- Slurm also has concept of “tasks”:

# Slurm resource allocation

## A job is not a job



- Slurm is very generic and supports diverse types of jobs
- Slurm also has concept of “tasks”: For us **1 task == 1 job**

# Slurm resource allocation

## A job is not a job



- Slurm is very generic and supports diverse types of jobs
- Slurm also has concept of “tasks”: For us **1 task == 1 job**
- Easiest use: 1 job = 1 R/py process = Some number of threads on 1 node

# Slurm resource allocation

## A job is not a job



- Slurm is very generic and supports diverse types of jobs
- Slurm also has concept of “tasks”: For us **1 task == 1 job**
- Easiest use: 1 job = 1 R/py process = Some number of threads on 1 node
- Technically possible to have 1 job spanning multiple nodes → headache country

# Slurm resource allocation

## A job is not a job



- Slurm is very generic and supports diverse types of jobs
- Slurm also has concept of “tasks”: For us **1 task == 1 job**
- Easiest use: 1 job = 1 R/py process = Some number of threads on 1 node
- Technically possible to have 1 job spanning multiple nodes → headache country
- Slurm allocates **cores**, not **threads**

# Slurm resource allocation

## A job is not a job



- Slurm is very generic and supports diverse types of jobs
- Slurm also has concept of “tasks”: For us **1 task == 1 job**
- Easiest use: 1 job = 1 R/py process = Some number of threads on 1 node
- Technically possible to have 1 job spanning multiple nodes → headache country
- Slurm allocates **cores**, not **threads**
- But: You only care about threads

# Slurm resource allocation

## A job is not a job



- Slurm is very generic and supports diverse types of jobs
- Slurm also has concept of “tasks”: For us **1 task == 1 job**
- Easiest use: 1 job = 1 R/py process = Some number of threads on 1 node
- Technically possible to have 1 job spanning multiple nodes → headache country
- Slurm allocates **cores**, not **threads**
- But: You only care about threads
- You need 1 thread, you ask for `cpus-per-task=`**1**, you get **1** core reserved

# Slurm resource allocation

## A job is not a job



- Slurm is very generic and supports diverse types of jobs
- Slurm also has concept of “tasks”: For us **1 task == 1 job**
- Easiest use: 1 job = 1 R/py process = Some number of threads on 1 node
- Technically possible to have 1 job spanning multiple nodes → headache country
- Slurm allocates **cores**, not **threads**
- But: You only care about threads
- You need 1 thread, you ask for `cpus-per-task=`**1**, you get **1** core reserved
- → More efficient to just ask for 2 “cpus” and 1 full core

# Example: Efficiently using a node

If your goal is to fully utilize 1 node (96 cores, 192 threads)



## What works

- 1 job with 192 threads ✅

# Example: Efficiently using a node

If your goal is to fully utilize 1 node (96 cores, 192 threads)



## What works

- 1 job with 192 threads ✅
- 96 jobs with 2 threads ✅

# Example: Efficiently using a node

If your goal is to fully utilize 1 node (96 cores, 192 threads)



## What works

- 1 job with 192 threads ✅
- 96 jobs with 2 threads ✅
- 8 jobs with 24 threads ✅

# Example: Efficiently using a node

If your goal is to fully utilize 1 node (96 cores, 192 threads)



## What works

- 1 job with 192 threads ✅
- 96 jobs with 2 threads ✅
- 8 jobs with 24 threads ✅

## What does not work

- 192 jobs with 1 thread ❌

# Example: Efficiently using a node

If your goal is to fully utilize 1 node (96 cores, 192 threads)



## What works

- 1 job with 192 threads ✅
- 96 jobs with 2 threads ✅
- 8 jobs with 24 threads ✅

## What does not work

- 192 jobs with 1 thread ❌
- Each job reserves 1 **core**
- → 192 jobs allocate 192 cores, but a node only has 96!

# Example: Efficiently using a node

If your goal is to fully utilize 1 node (96 cores, 192 threads)



## What works

- 1 job with 192 threads ✅
- 96 jobs with 2 threads ✅
- 8 jobs with 24 threads ✅

## What does not work

- 192 jobs with 1 thread ❌
- Each job reserves 1 **core**
- → 192 jobs allocate 192 cores, but a node only has 96!

⚠️ If you're used to `batchtools`, you probably default to single-threaded jobs

# Did it work?

## Checking on your jobs



While running:

- `squeue --me`: list your jobs
- `tail -f logs/job_12345.out`: watch output
- `scancel <jobid>`: cancel a job

After completion:

- Check your output files (e.g. `logs/`)
- `sacct --starttime=today`: job history
- `sacct -j <jobid> --format=JobID,State,Elapsed,MaxRSS`

# Did it work?

## Checking on your jobs



While running:

- `squeue --me`: list your jobs
- `tail -f logs/job_12345.out`: watch output
- `scancel <jobid>`: cancel a job

After completion:

- Check your output files (e.g. `logs/`)
- `sacct --starttime=today`: job history
- `sacct -j <jobid> --format=JobID,State,Elapsed,MaxRSS`

Common failure modes

- **TIMEOUT**: job ran out of time, request more with `--time`
- **OUT_OF_MEMORY**: request more RAM with `--mem`
- **FAILED**: your script has a bug, check the log output (*did you load modules?*)

# Where does your stuff live?

## Two storage tiers



### Home directory

`/srv/home/<user>`

- Fast
- Limited space, also shared with software
- Scripts, code, *active* projects

# Where does your stuff live?

## Two storage tiers



### Home directory

`/srv/home/<user>`

- Fast
- Limited space, also shared with software
- Scripts, code, *active* projects

### Archive storage

`/mnt/sas/users/<user>`

- Large capacity
- Slower
- Big datasets, *inactive* projects

# Where does your stuff live?

## Two storage tiers



### Home directory

`/srv/home/<user>`

- Fast
- Limited space, also shared with software
- Scripts, code, *active* projects

### Archive storage

`/mnt/sas/users/<user>`

- Large capacity
- Slower
- Big datasets, *inactive* projects

- Both are **shared across all nodes** (head + compute) via internal network
- Compute nodes also have fast **local scratch** (`$TMPDIR`, auto-cleaned after job)

# Where does your stuff live?

## Two storage tiers



### Home directory

`/srv/home/<user>`

- Fast
- Limited space, also shared with software
- Scripts, code, *active* projects

### Archive storage

`/mnt/sas/users/<user>`

- Large capacity
- Slower
- Big datasets, *inactive* projects

- Both are **shared across all nodes** (head + compute) via internal network
- Compute nodes also have fast **local scratch** (`$TMPDIR`, auto-cleaned after job)

> ⚠ There are **no backups**. Use `git` for code, be careful with data.

# User utilities

## Command-line aliases/functions



- I have prepared a few shorthand tools based on things I need often
- On head node, get info with `slurm_user_help`:

```
❯ slurm_user_help
=== Slurm User Helper Functions ===

JOB MONITORING:
  sq              - Enhanced squeue with better formatting
  sqm             - My jobs queue
  sqs             - Job status summary with colors
[...]

JOB ACCOUNTING:
  slac              - Enhanced sacct
  slac1h/1d/3d/1w   - Jobs from last hour/day/3days/week (excludes PENDING)
    State filters:  -a (all), -p (pending), -c (completed), -f (failed), -r (running)
  sltoday           - Jobs submitted today
```

# Getting started

## And further reading material



1. Ask me for an account, I will send you login credentials
2. Read (and bookmark) the docs:
   - cluster.bips.coffee (public)
3. Bookmark the dashboard (current cluster usage):
   - http://srvcluster.bips.de/ (no https!)
4. Try demos / usage examples for `batchtools`, `mirai`, `targets`:
   - srvgit.bips.eu/bips/bips-cluster-demos