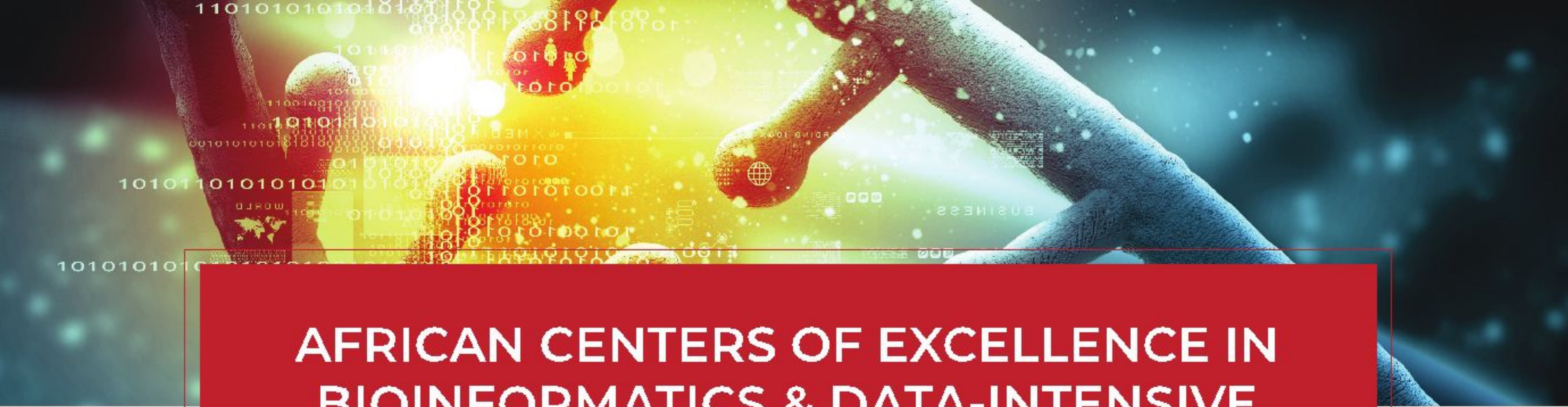


AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

ACE HPC Workshop
Day 3 Slurm

You did it!

- Today is the last day
  - We have learned a lot
- Yesterday was very hard
  - Today will be easier

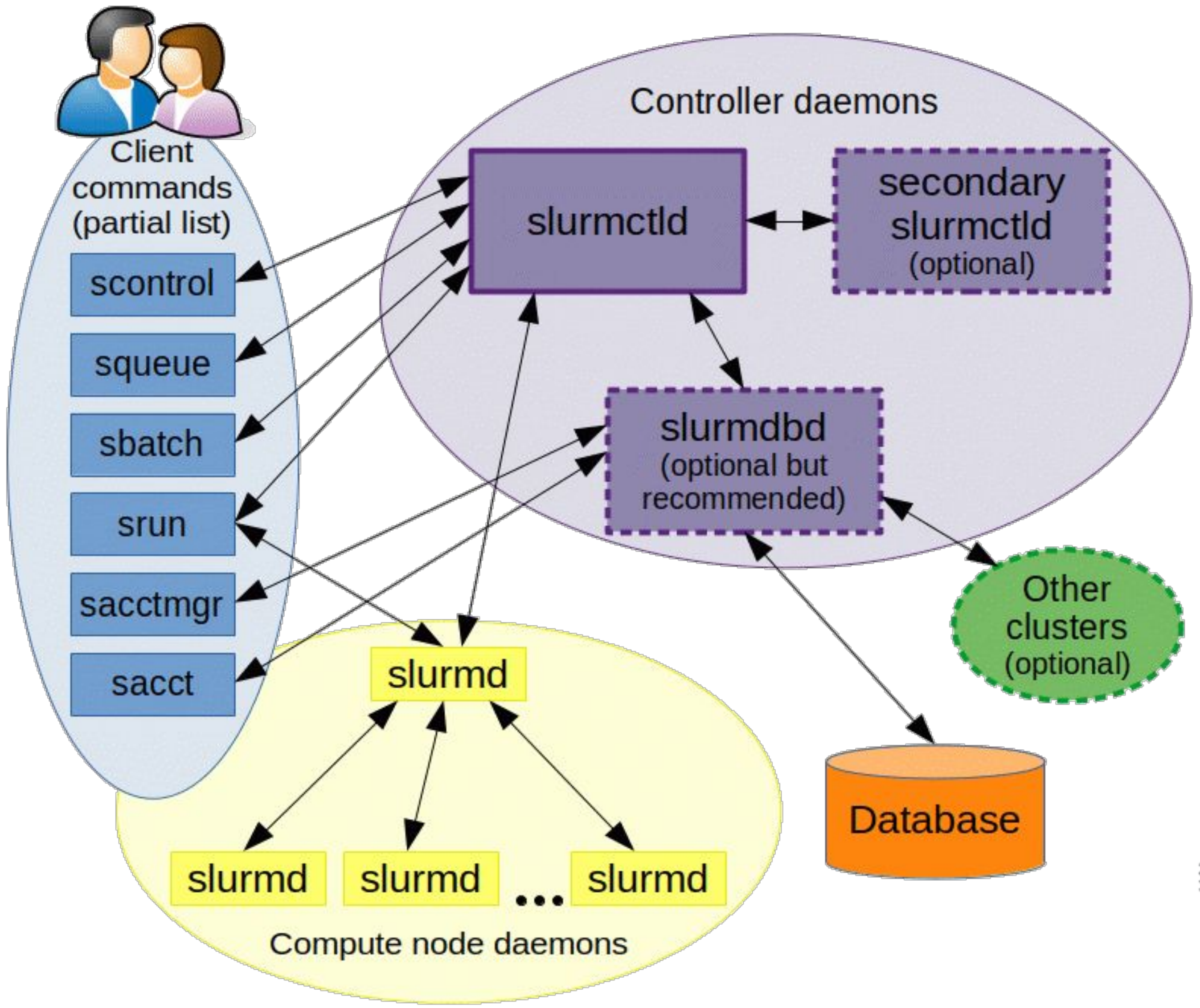
AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

SLURM

Slurm stands for “Simple Linux Utility for Resource Management”. It is as stated in the acronym, a *resource manager* and is both open-source, and highly-scalable for linux clusters. It has three primary functions all centered on facilitating parallelism:

1. Allocates exclusive or non-exclusive resources (compute nodes) to users to perform computation
2. Provides interface to start, stop, and monitor parallel computation
3. Arbitrates conflicting demands made on resources via a queue of pending work (a queue whose structure can be defined)

Architecture


Client commands (partial list)
scontrol
squeue
sbatch
srun
sacctmgr
sacct
Controller daemons
slurmctld
secondary slurmctld (optional)
slurmdbd (optional but recommended)
Other clusters (optional)
Database
slurmd
slurmd
slurmd
...
slurmd
Compute node daemons

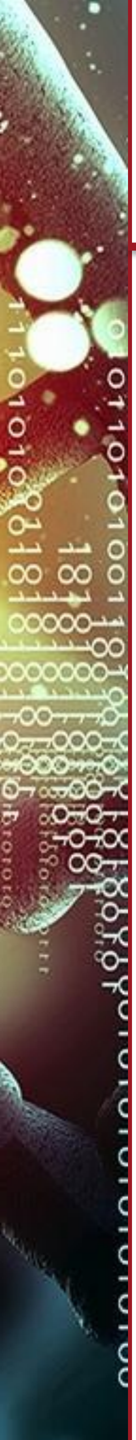
AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

slurmctld

- Slurm has a central manager or daemon (demon), the **slurmctld**
- Coordinates the work in that it handles job queues, scheduling , resource allocation, and monitoring.
- Accepts job submissions and decides which jobs should be run and in what priority.
- Furthermore, it keeps track of the status of all nodes and their resources (CPU, memory, energy use) and communicates with the **slurmd** daemon on each node.

slurmd

- Each compute node has a **slurmd** daemon which is analogous to a remote shell in that it waits for work ,executes the work, returns status, and waits for more work.
- It is responsible for launching and managing jobs on the respective node, monitors the resources and reports the status of the node and job to **slurmctld**, and overseas communication with **slurmctld** in general.



Overview of Commands

As for the user, there are many CLI tools, each of which are extremely rich in functionality.

- **srun** is the most basic way of initiating and defining job.
- **scancel** allows for termination of queued or running jobs
- **sinfo** reports system status
- **squeue** reports the status of jobs
- **sacct** allows us to get information about jobs and job steps
- **sview** command graphically reports system and job status including network topology.

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

Some terminology: Cluster, Node, Processor

- Slurm manages a **cluster**
  - a group of **nodes** with the capacity to interact with one another over a network.
- A (*compute*) **node** is a computer and is part of a larger set of nodes (a cluster).
  - There are **compute** nodes *login* nodes, *file server* nodes, *management* nodes, etc.
    - A compute node offers resources such as processors, volatile memory (RAM), permanent disk space (e.g. SSD), accelerators (e.g. GPU) etc.
- A **processor** is the thing that , overall, actually *does* the computation.

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

Core, Socket, Thread

- A **core** is the part of a processor that does the computations.
- A processor comprises multiple cores, as well as a memory controller, a bus controller, and possibly many other components.
- A processor in the Slurm context is referred to as a **socket**, which actually is the name of the slot on the motherboard that hosts the processor.
- A single core can have one or two **hardware threads**. This is a technology that allows virtually doubling the number of cores the operating systems perceives while only doubling part of the core components -- typically the components related to memory and I/O and not the computation components.

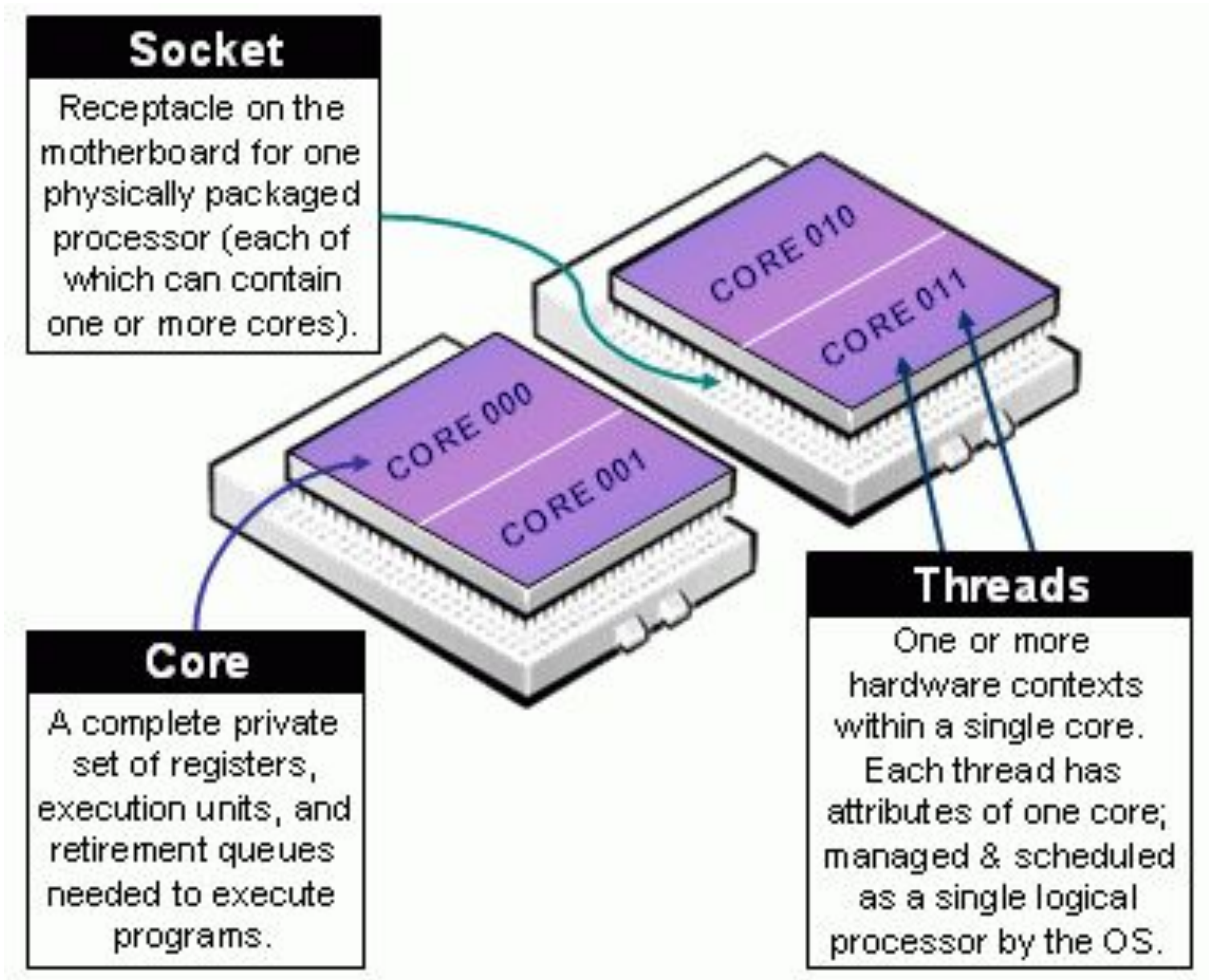
AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

CPU, Partition

- a **CPU** in a general context refers to a processor, but in the Slurm context, a CPU is a consumable resource offered by a node. It can refer to a socket, a core, or a hardware thread, based on the Slurm configuration.
- A **partition** acts like a job queue which, furthermore, has structure imposed on it like job size limits, job time limit, users which have access to it etc.
- The role of Slurm is to match those resources to **jobs**. A job comprises one or more (sequential) **steps**, and each step has one or more (parallel) **tasks**. A task is an instance of a running program, i.e. at a process, possibly along with **subprocesses** or **software threads**.

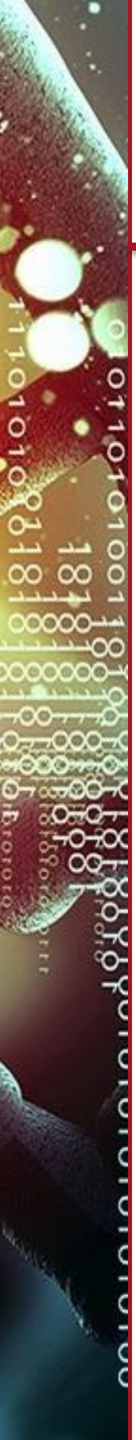
AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

Visual of Terminology


Socket
Receptacle on the motherboard for one physically packaged processor (each of which can contain one or more cores).
CORE 010
CORE 011
CORE 000
CORE 001
Threads
One or more hardware contexts within a single core. Each thread has attributes of one core; managed & scheduled as a single logical processor by the OS.
Core
A complete private set of registers, execution units, and retirement queues needed to execute programs.

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

Some Notes on Our System



<u>ICER MALI HPC Cluster</u>

16 Compute Nodes (8 Regular nodes and 8 KNL Nodes)

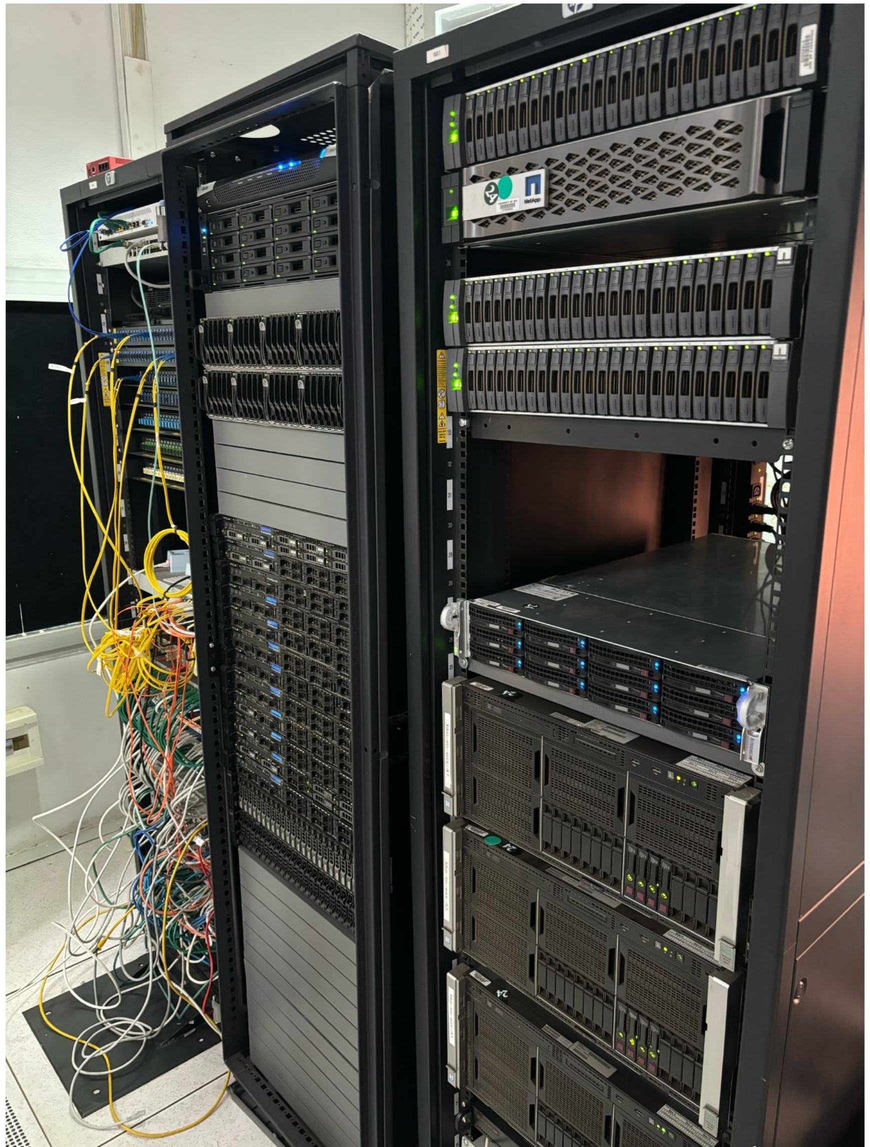
Regular Node has 48 cores and 128GBs of RAM

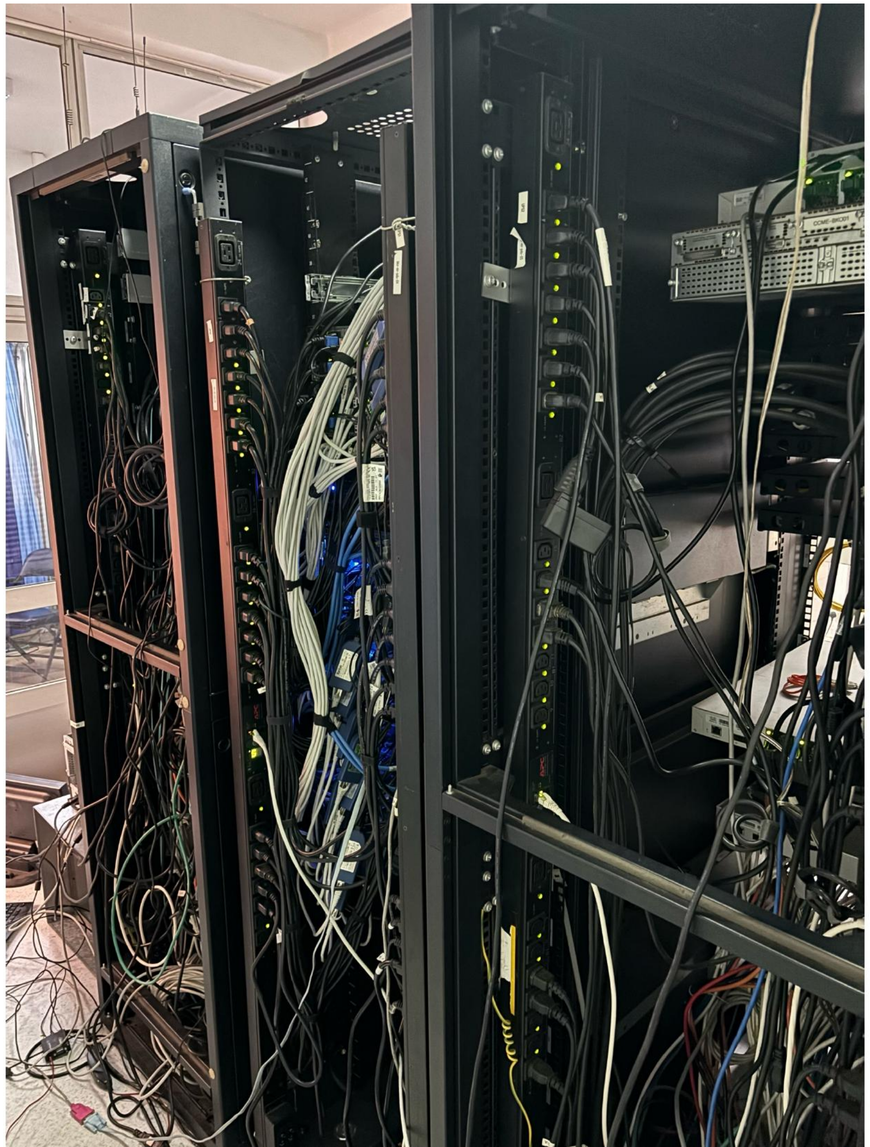
KNL (Knights Landing) Node has 272 Cores and 96 GBs of RAM

Total Cores: 2,560

Total RAM: 1,792

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

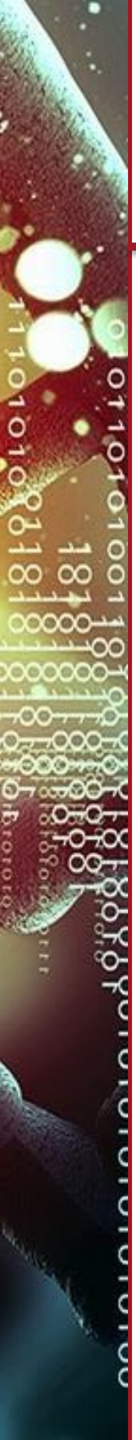




AFRICAN
CENTERS
OF EXCELLENCE
IN BIOINFORMATICS &
DATA-INTENSIVE SCIENCE

A Bit on Storage on System

- Shared storage-device, your home directory is mounted on that device
  - That means your storage is accessible from every node



Exercise 0 : Set-up

Make sure you can connect to the server, log into the server.

Extra-credit: Setup remote ssh access with visual studio code or with whatever IDE.

How to Run Jobs

We go over in detail how to run jobs interactively (via srun and salloc) and non-interactively (via sbatch).

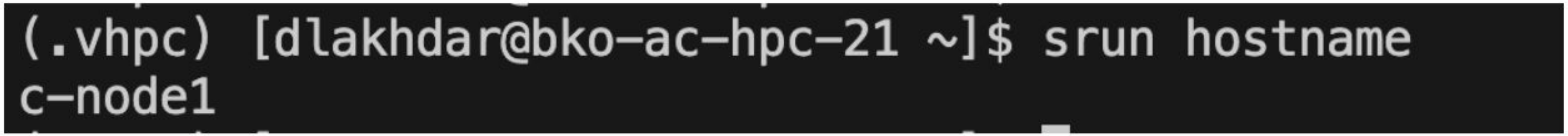
Interactive: srun, salloc
Via Script: sbatch

srun

Srun is the basic means of running a job in *real-time*. The man page for srun is enormous and exhibits how rich the srun command is in features. I recommend you read through the man page *once in its entirety before proceeding*. We will exhibit the most useful and common options in srun via a series of examples from basic to complex.

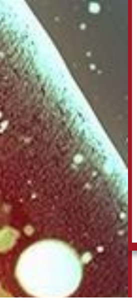
AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

Simple Job Submission



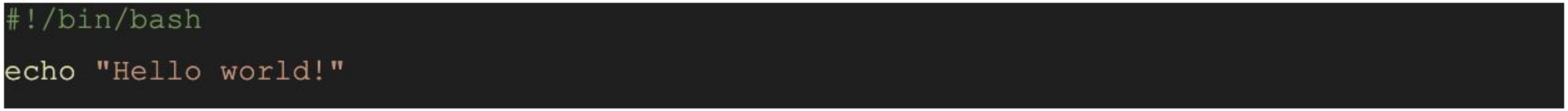
```
(.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun hostname
c-node1
```

We simply put a bash command after srun. This could of course be a command consisting of the **python** command and a python script or in fact any programming language. You can inelegantly get a command to stop running via control-c.



Job Submission with Script

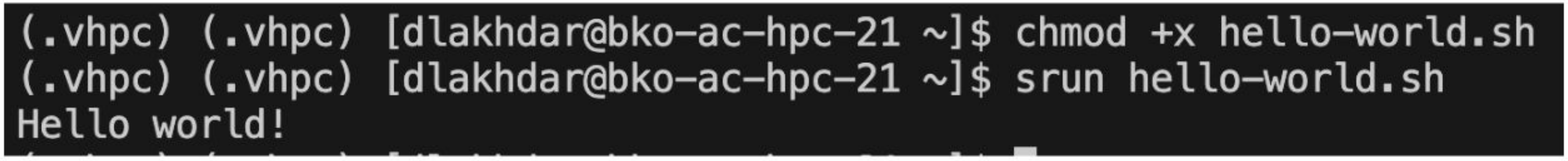
We can run a stupid bash file such as:



```
#!/bin/bash

echo "Hello world!"
```

Remembering to make the file executable we run:

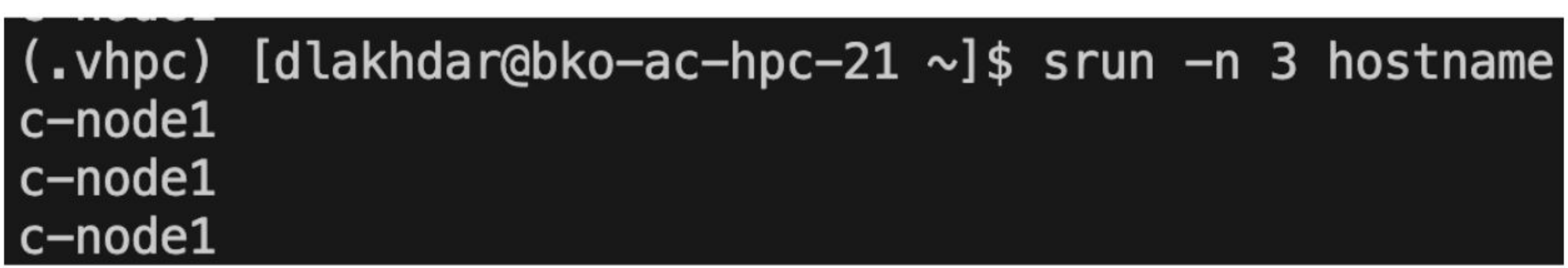


```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ chmod +x hello-world.sh
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun hello-world.sh
Hello world!
```

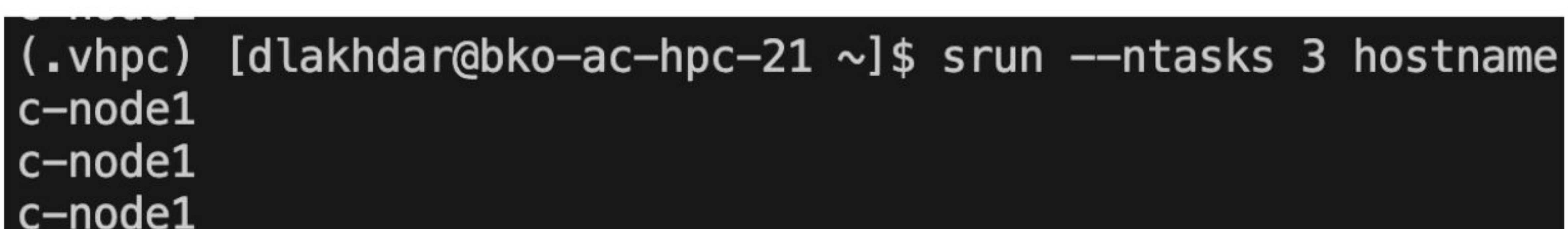
It is good practice to write out the full path, and not utilize the relative path as was done above.

Number of Tasks

Remember that a task is synonymous basically with a process , it is the computation say “hostname” executed a certain number of times.

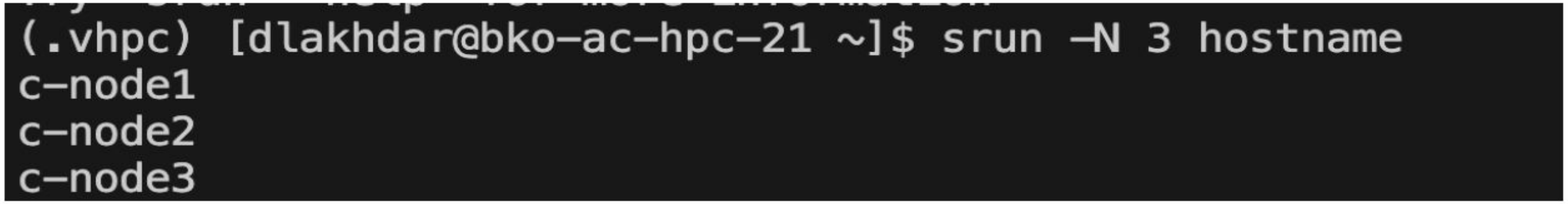


```
(.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun -n 3 hostname
c-node1
c-node1
c-node1
```

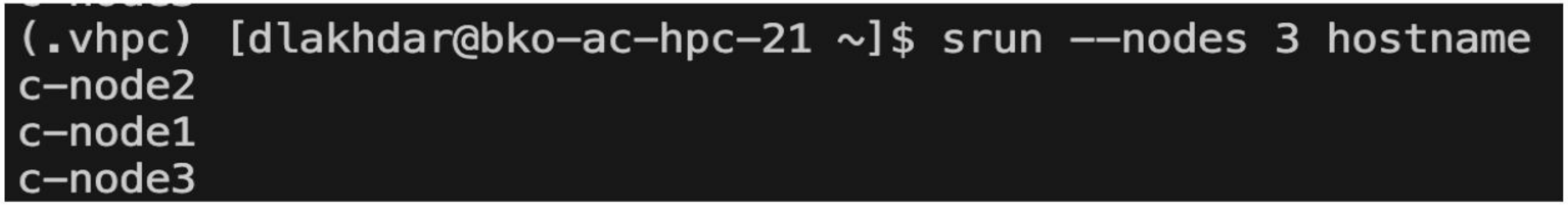


```
(.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun --ntasks 3 hostname
c-node1
c-node1
c-node1
```

Number of Nodes

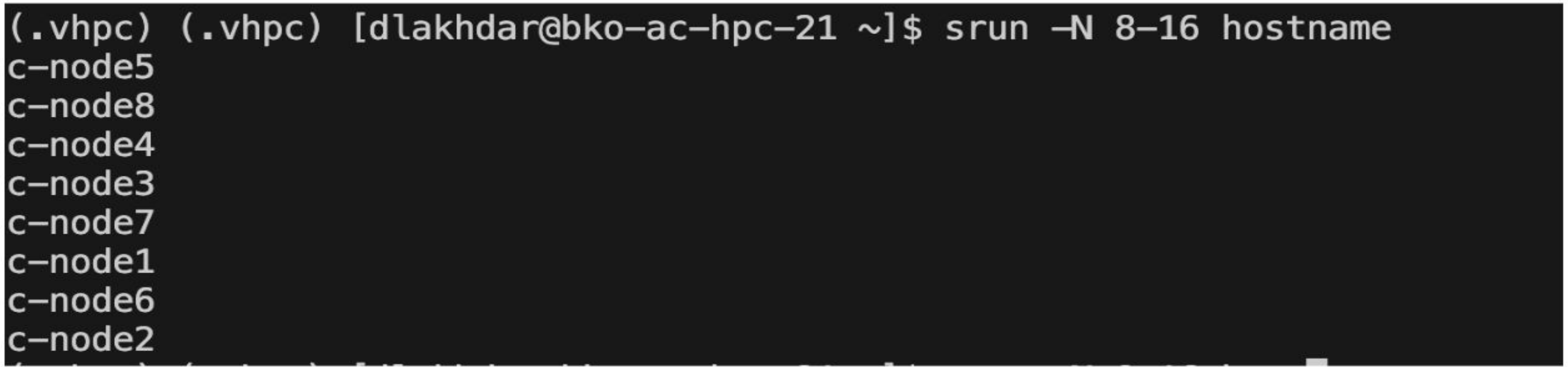


```
(.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun -N 3 hostname
c-node1
c-node2
c-node3
```



```
(.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun --nodes 3 hostname
c-node2
c-node1
c-node3
```

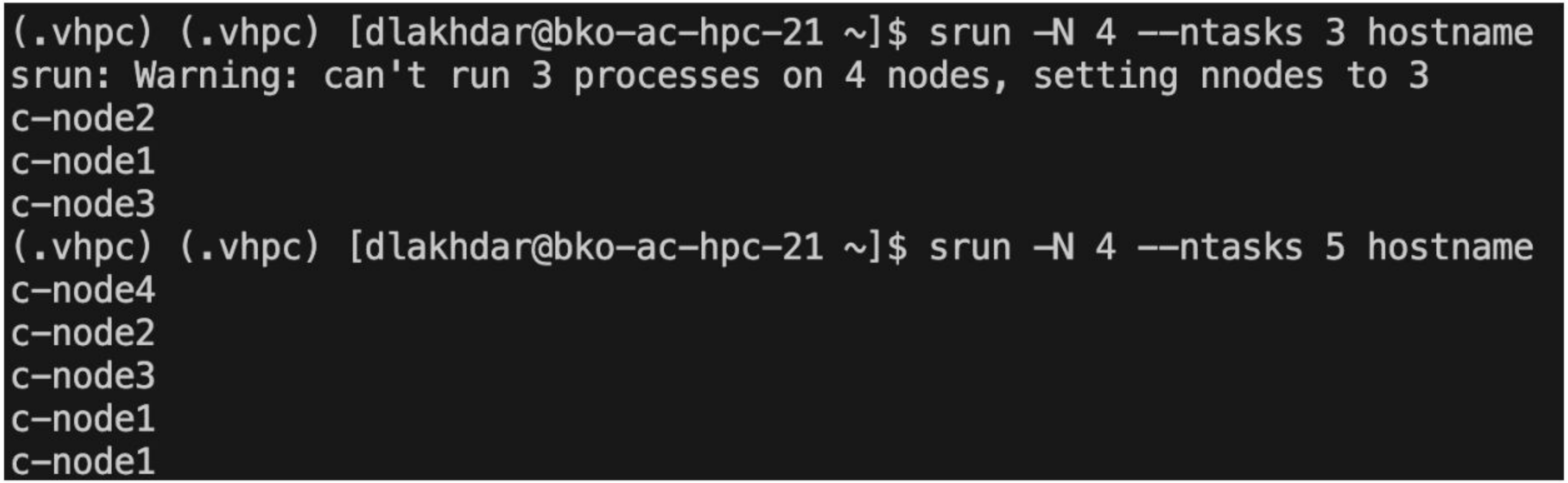
We can even dictate a range of nodes to use, a min-max expression:



```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun -N 8-16 hostname
c-node5
c-node8
c-node4
c-node3
c-node7
c-node1
c-node6
c-node2
```

Number of tasks, number of nodes , number of cpus

We can specify the number of tasks along with nodes and cpus.



```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun -N 4 --ntasks 3 hostname
srun: Warning: can't run 3 processes on 4 nodes, setting nnodes to 3
c-node2
c-node1
c-node3
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun -N 4 --ntasks 5 hostname
c-node4
c-node2
c-node3
c-node1
c-node1
```

Number of tasks, number of nodes , number of cpus

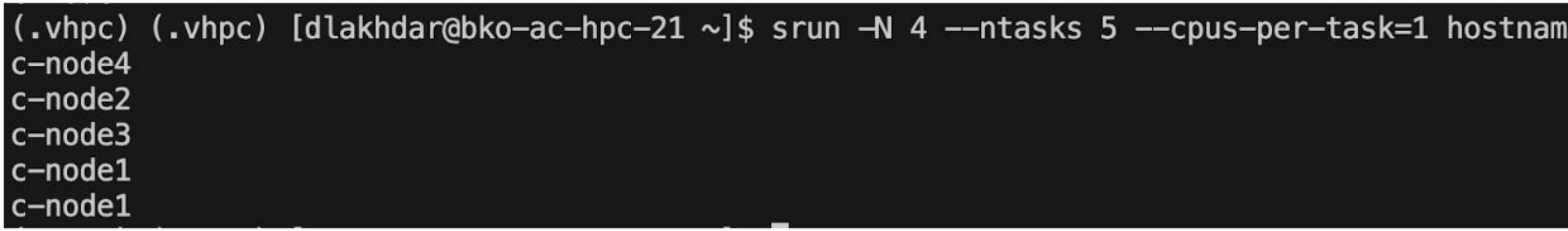
Notice that the number of tasks must exceed the number of nodes.

Slurm determines how to balance the tasks across the nodes.

What if we want to make sure that *a* task is performed by *a cpu*. We simply use the **--cpu-bind** option which is rich in functionality.

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

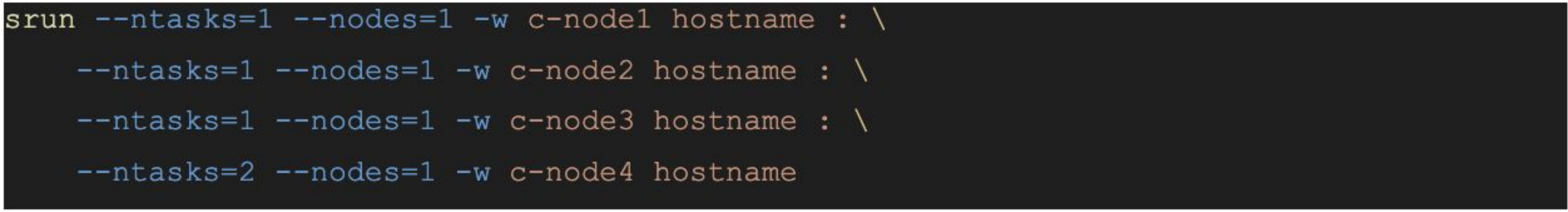
–per-task options



```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun -N 4 --ntasks 5 --cpus-per-task=1 hostnam
c-node4
c-node2
c-node3
c-node1
c-node1
```

Task Run on Certain Nodes

Utilize ‘:’ syntax which chains srun commands

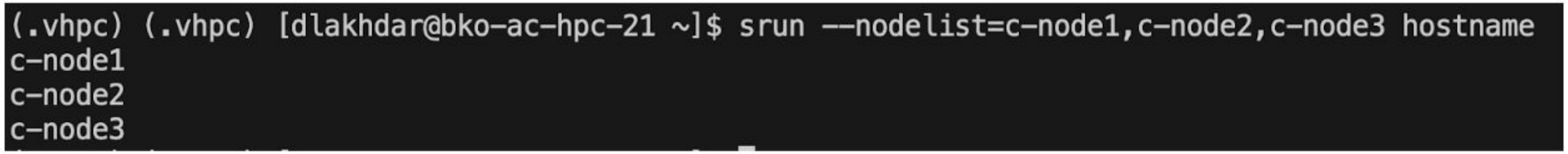


```
srun --ntasks=1 --nodes=1 -w c-node1 hostname : \
    --ntasks=1 --nodes=1 -w c-node2 hostname : \
    --ntasks=1 --nodes=1 -w c-node3 hostname : \
    --ntasks=2 --nodes=1 -w c-node4 hostname
```

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

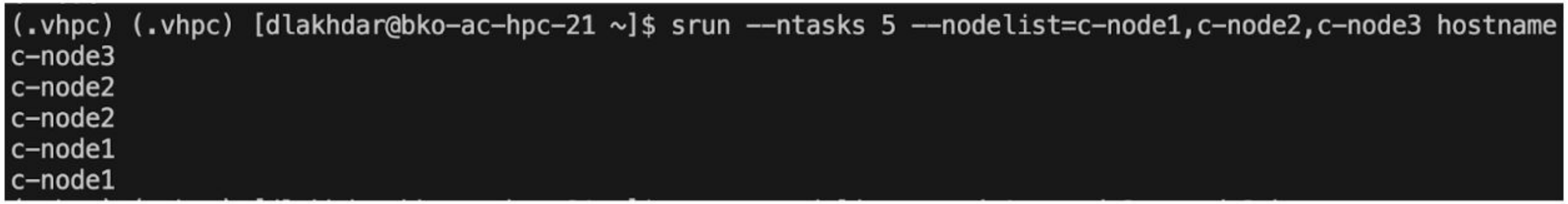
Exact Node List

We can actually specify which nodes we want to utilize via the **-w** or –**-nodelist** option and either a comma-delimited cli list, a range expression, or a file. We can use the cli list as:



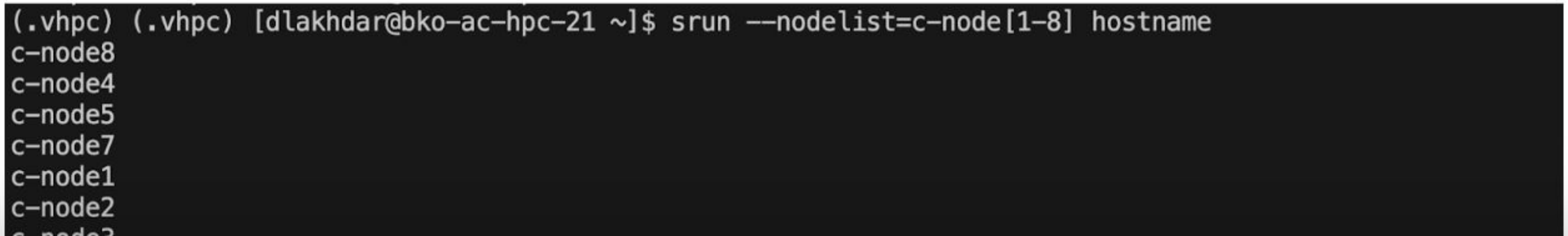
```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun --nodelist=c-node1,c-node2,c-node3 hostname
c-node1
c-node2
c-node3
```

Notice that the nodelist by itself dictates the number of tasks. Here there are 3 tasks. But what if we put in a number of tasks?



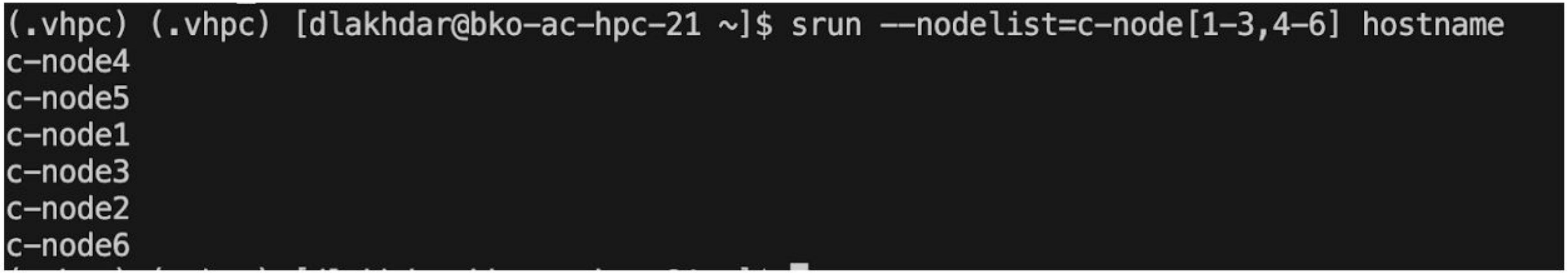
```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun --ntasks 5 --nodelist=c-node1,c-node2,c-node3 hostname
c-node3
c-node2
c-node2
c-node1
c-node1
```

We can also make use of range-expressions as follows, say we want to print the hostname for nodes 1-8:



```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun --nodelist=c-node[1-8] hostname
c-node8
c-node4
c-node5
c-node7
c-node1
c-node2
```

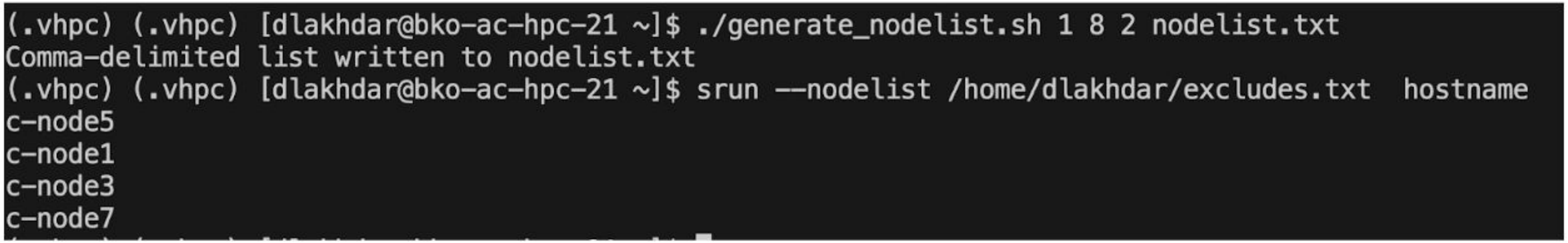
We can even break up this range-expressions as follows, say we want to print the hostname for nodes 1-3, then for nodes 4-6:



```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun --nodelist=c-node[1-3,4-6] hostname
c-node4
c-node5
c-node1
c-node3
c-node2
c-node6
```

Utilize File for Nodes

So to generate a node-list we run the first command as shown, and then to use srun we run the second command as shown:



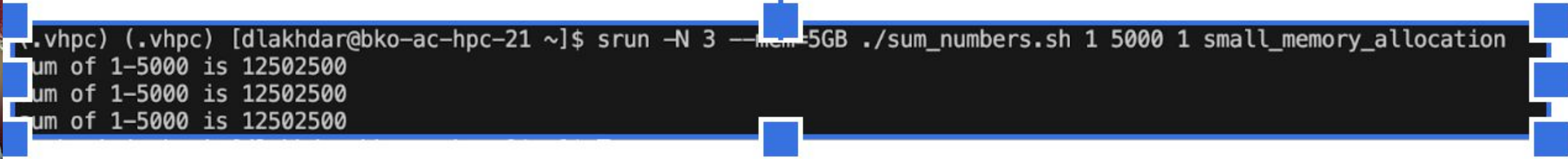
```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ ./generate_nodelist.sh 1 8 2 nodelist.txt
Comma-delimited list written to nodelist.txt
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun --nodelist /home/dlakhdar/excludes.txt  hostname
c-node5
c-node1
c-node3
c-node7
```

Utilize the file *full-path* to the nodelist as slurm recognizes this as a file only with the inclusion of the “/” symbol.

—exclude option

Memory Options

We can actually control the RAM memory utilized in jobs. The most basic option is via **--mem** which controls the memory requested in *megabytes* . But we can of course specify the units, I recommend using **GB.**

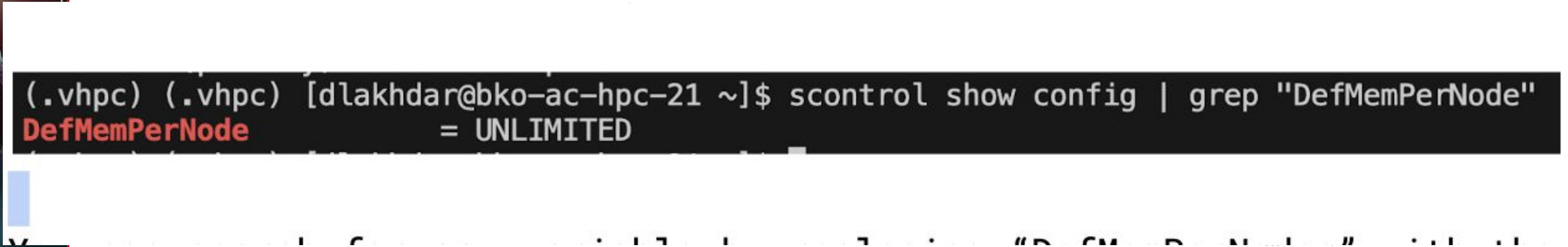


```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun -N 3 --mem=5GB ./sum_numbers.sh 1 5000 1 small_memory_allocation
um of 1-5000 is 12502500
um of 1-5000 is 12502500
um of 1-5000 is 12502500
```

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

First Look at System Variables

Now this memory option in fact has a system-configured environmental default which we can view via **scontrol show config.** It is called **DefMemPerNode**, (default memory per node) and is linked to **MaxMemPerNode**, which is the maximum memory one can request per node. We will return to this topic of system-variables later but for now we can investigate any given environment via the following command:

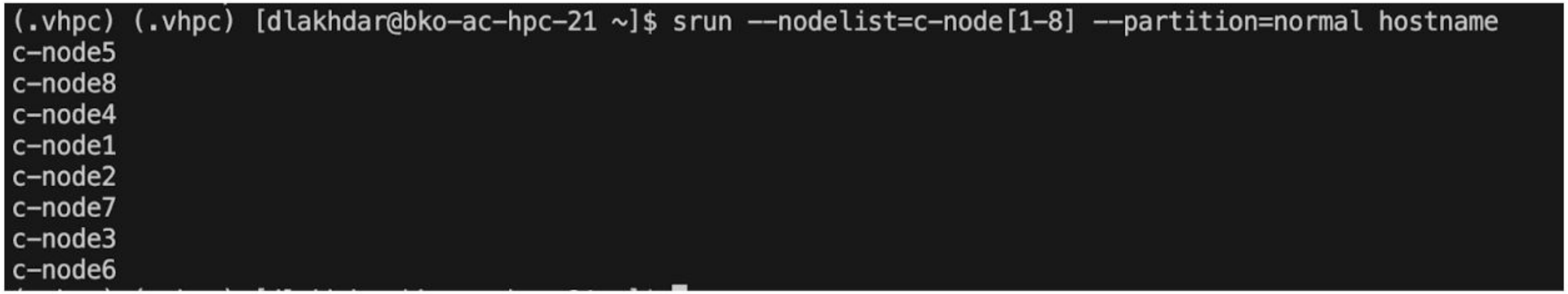


```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ scontrol show config | grep "DefMemPerNode"
DefMemPerNode                    = UNLIMITED
```

Partition Options

Partition options

We can specify which partition to use via **-p** or **--partition** :



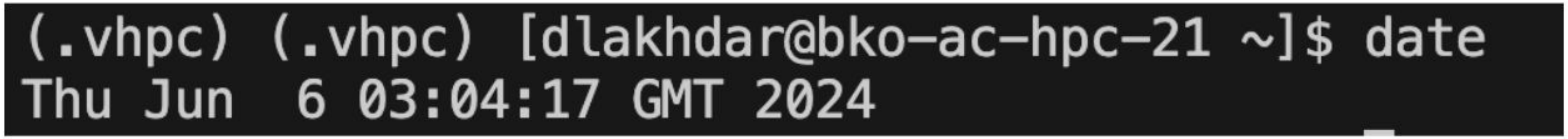
```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun --nodelist=c-node[1-8] --partition=normal hostname
c-node5
c-node8
c-node4
c-node1
c-node2
c-node7
c-node3
c-node6
```

We can specify multiple partitions via a comma-delimited list.

Time Options

Let us say we want to dictate when a job starts, when it ends, how long it runs.

First we must be clear on what the time is for our system, utilize the **date** to make sure:



```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ date
Thu Jun  6 03:04:17 GMT 2024
```

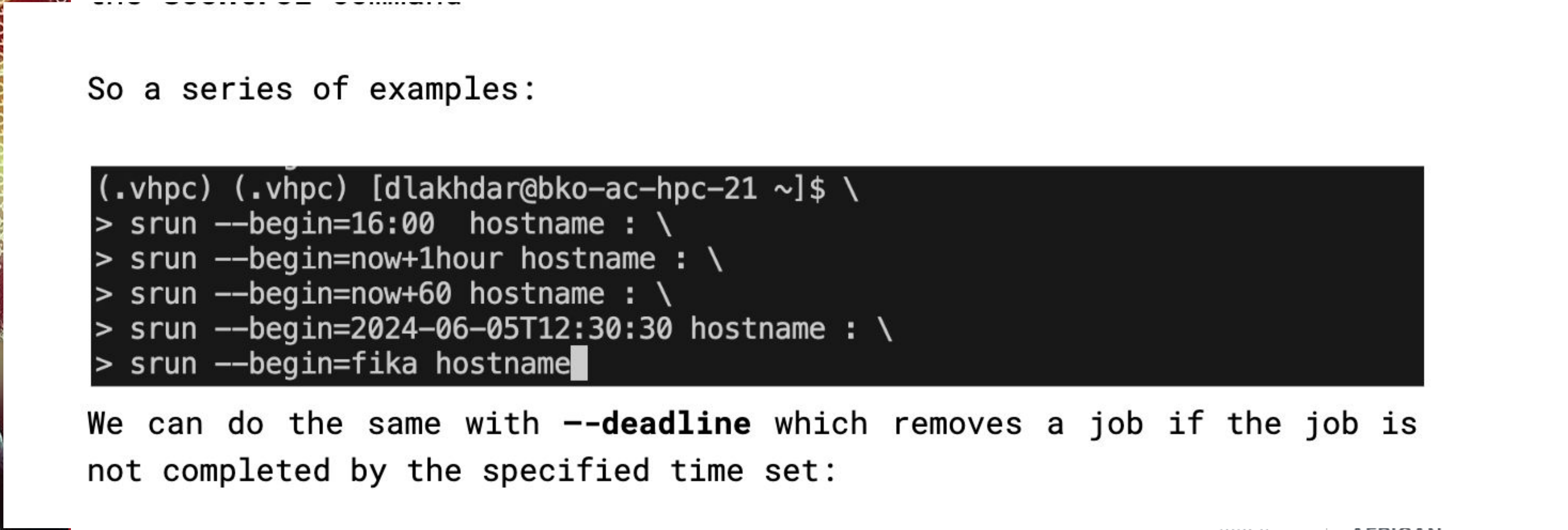
AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

Time-Formats

- *HH:MM:SS* to run a job at a specific time of day (seconds are optional). (If that time is already past, the next day is assumed.)
- You may also specify *midnight*, *noon*, *elevenses* (11 AM), *fika* (3 PM) or *teatime* (4 PM)
- you can have a time-of-day suffixed with *AM* or *PM* for running in the morning or the evening.
- You can also say what day the job will be run, by specifying a date of the form *MMDDYY* or *MM/DD/YY YYYY-MM-DD*.
- Combine date and time using the following format *YYYY-MM-DD[THH:MM[:SS]]*.
- You can also give times like *now + count time-units*, where the time-units can be *seconds* (default), *minutes*, *hours*, *days*, or *weeks* and you can tell Slurm to run the job today with the keyword *today* and to run the job tomorrow with the keyword *tomorrow*.
- The value may be changed after job submission using the **scontrol** command

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

—begin



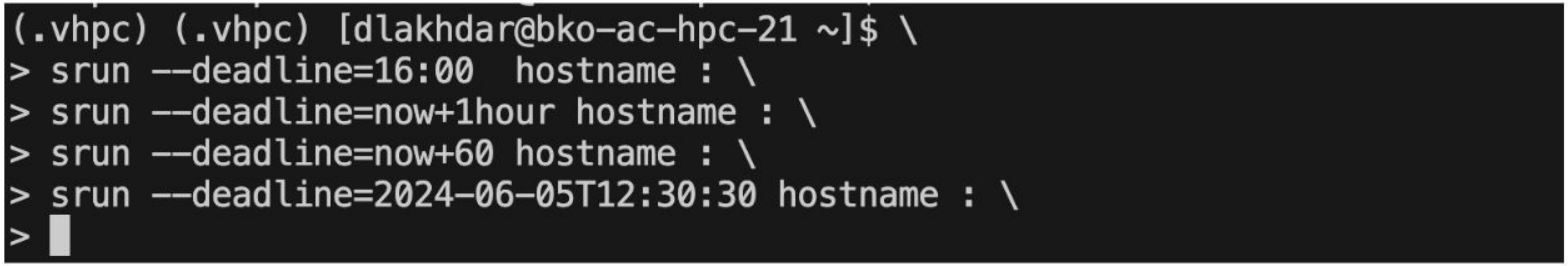
So a series of examples:

```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ \
> srun --begin=16:00  hostname : \
> srun --begin=now+1hour hostname : \
> srun --begin=now+60 hostname : \
> srun --begin=2024-06-05T12:30:30 hostname : \
> srun --begin=fika hostname
```

We can do the same with **--deadline** which removes a job if the job is not completed by the specified time set:

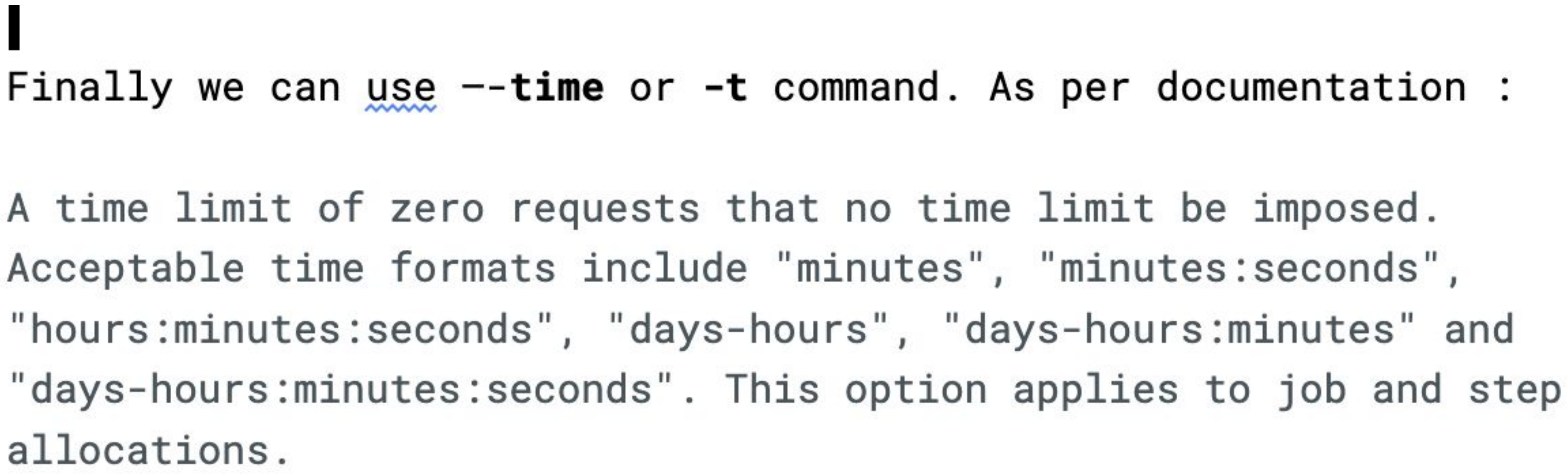
—deadline

We can do the same with **--deadline** which removes a job if the job is not completed by the specified time set:



```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ \
> srun --deadline=16:00  hostname : \
> srun --deadline=now+1hour hostname : \
> srun --deadline=now+60 hostname : \
> srun --deadline=2024-06-05T12:30:30 hostname : \
> 
```

—time



Finally we can use --**time** or **-t** command. As per documentation :

A time limit of zero requests that no time limit be imposed. Acceptable time formats include "minutes", "minutes:seconds", "hours:minutes:seconds", "days-hours", "days-hours:minutes" and "days-hours:minutes:seconds". This option applies to job and step allocations.

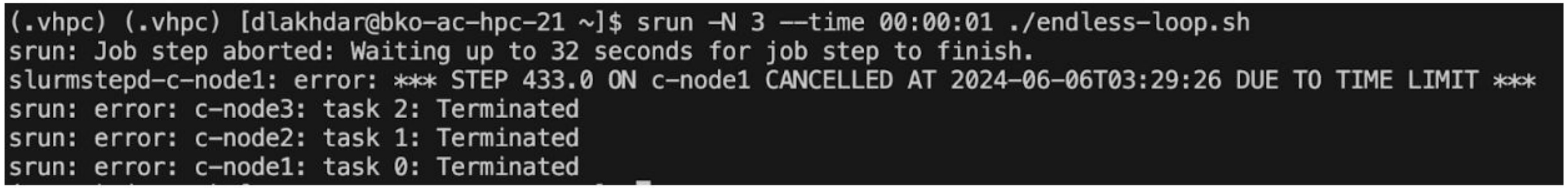
AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

Example : Infinite Loop

Here is an example. Say we have an infinite loop, and we want to stop it after 1 second, then we would write the program **endless-loop.sh**:

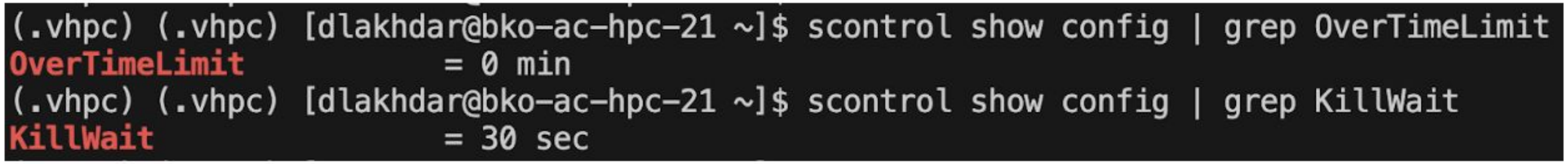
```
#!/bin/bash
while :; do :; done
```

Then we make it executable and the execute it via srun with the time option via:



```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun -N 3 --time 00:00:01 ./endless-loop.sh
srun: Job step aborted: Waiting up to 32 seconds for job step to finish.
slurmstepd-c-node1: error: *** STEP 433.0 ON c-node1 CANCELLED AT 2024-06-06T03:29:26 DUE TO TIME LIMIT ***
srun: error: c-node3: task 2: Terminated
srun: error: c-node2: task 1: Terminated
srun: error: c-node1: task 0: Terminated
```

The default limit is the partition default time limit. When the limit is reached, each task in each job step is sent SIGTERM followed by SIGKILL. The interval between signals is specified by the Slurm configuration parameter **KillWait**. So we find that :



```
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ scontrol show config | grep OverTimeLimit
OverTimeLimit           = 0 min
(.vhpc) (.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ scontrol show config | grep KillWait
KillWait                = 30 sec
```

This means that even though the job ended after 1 second via a SIGTERM, it took an extra 30 seconds for the overall job to be terminated.

Job Dependency

Let us say you want to run a job before some job runs, or any job runs. How do we accomplish this? Via the —-**dependency** option which apparently applies only to job allocations and not steps. In fact this option is the easiest means to create jobs with multiple **steps**.

Job Dependency Signature

The signature of the command is:

```
-d, --dependency=<dependency_list>
```

Now the dependency list takes the form :

```
<type:job_id[:job_id][,type:job_id[:job_id]]> or
<type:job_id[:job_id][?type:job_id[:job_id]]>
```

Now only two dependency types can be joined by a , which means “and”, **?** which means “or”.

List Dependency Types

- **after:job_id[[+time][:jobid[+time]...]]**

After the specified jobs start or are cancelled and 'time' in minutes from job start or cancellation happens, this job can begin execution. If no 'time' is given then there is no delay after start or cancellation.

- **afterany:job_id[:jobid...]**

This job can begin execution after the specified jobs have terminated. This is the default dependency type.

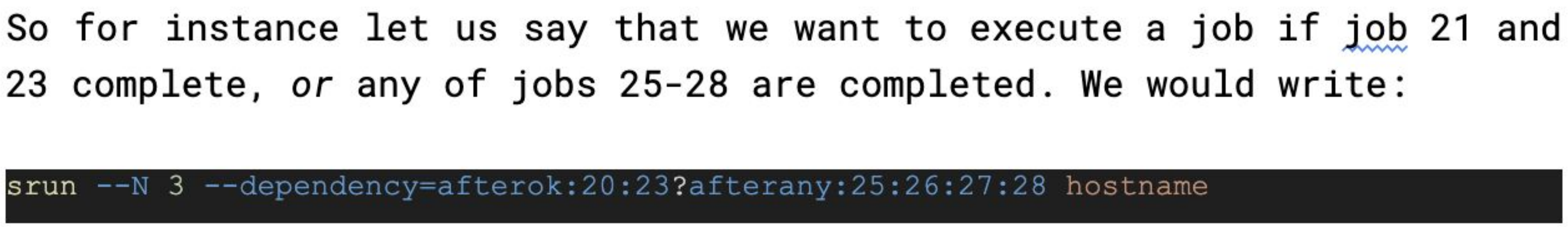
- **afternotok:job_id[:jobid...]**

This job can begin execution after the specified jobs have terminated in some failed state (non-zero exit code, node failure, timed out, etc). This job must be submitted while the specified job is still active or within **MinJobAge** seconds after the specified job has ended.

- **afterok:job_id[:jobid...]**

This job can begin execution after the specified jobs have successfully executed (ran to completion with an exit code of zero). This job must be submitted while the specified job is still active or within **MinJobAge** seconds

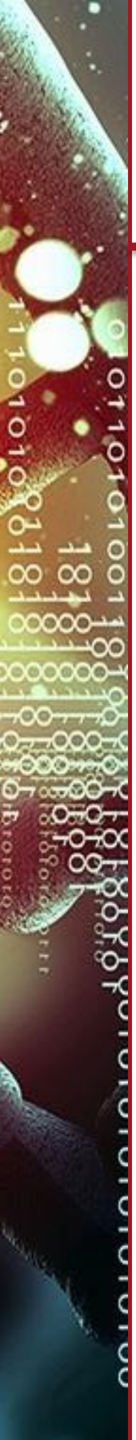
Examples:



So for instance let us say that we want to execute a job if job 21 and 23 complete, *or* any of jobs 25-28 are completed. We would write:

```
srun --N 3 --dependency=afterok:20:23?afterany:25:26:27:28 hostname
```

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE



Debugging

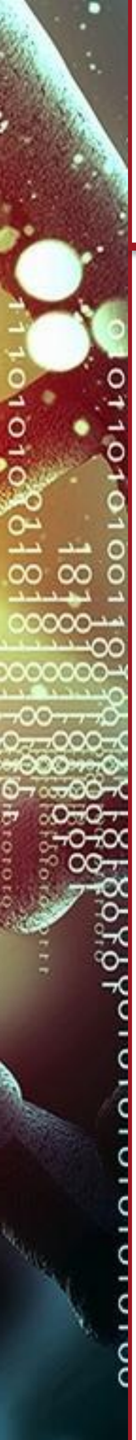
We simply mention two options **-e or --error** and **--output or -o**. Specify these in order to generate an output file that takes standard output and places it in the file and standard error and places it in the error file.

Exercise 1 : 15 minutes

Use srun to print “hello world!”

hint: remember echo command

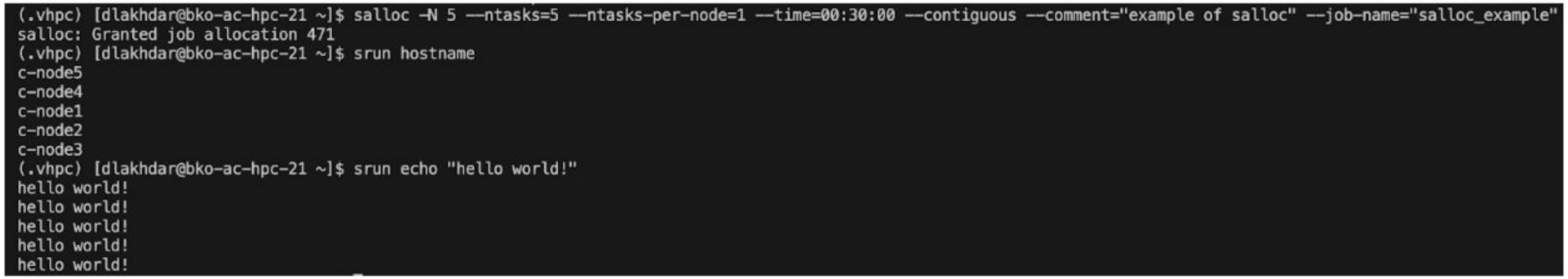
Play around with different options



Salloc

Let us say you want to be able to allocate 3 nodes but you want to work interactively with that allocation e.g. you might be doing some exploratory work on each node and you are not sure what commands you will be running. You can specify an interactive job like this, and demand resources in the same way you did for **srun**:

Salloc is simply an interactive way to specify a number of resources and we provide an example here:



```
(.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ salloc -N 5 --ntasks=5 --ntasks-per-node=1 --time=00:30:00 --contiguous --comment="example of salloc" --job-name="salloc_example"
salloc: Granted job allocation 471
(.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun hostname
c-node5
c-node4
c-node1
c-node2
c-node3
(.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ srun echo "hello world!"
hello world!
hello world!
hello world!
hello world!
hello world!
```

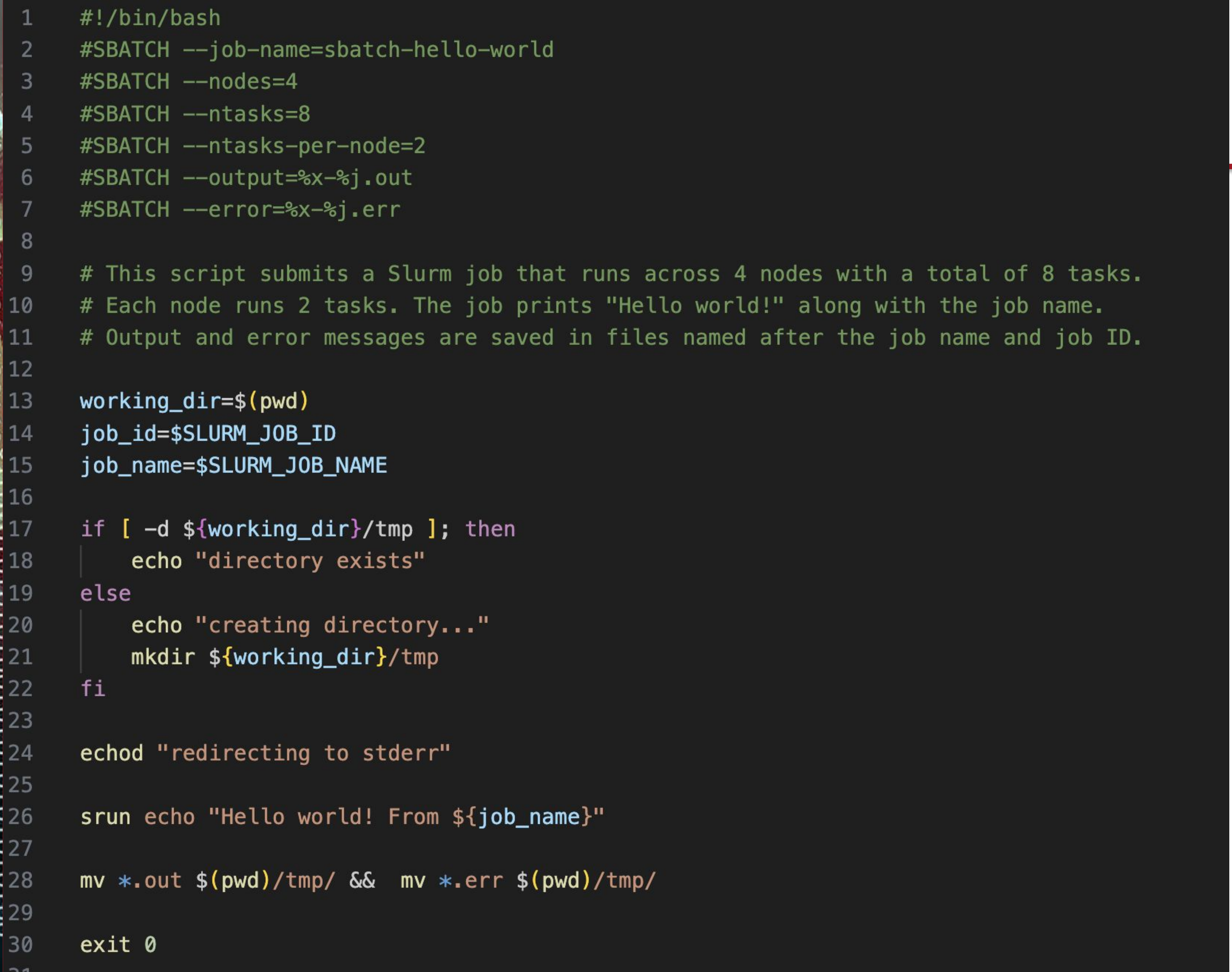
Sbatch

We have spoken about running jobs “non-interactively” i.e. submitting a job to slurm and having it run in the background. This is the function of **sbatch.** We will explore how to utilize this ClI tool now.

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

Batch Script

Sbatch as a command requires that it be executed with a **batch** script. A **batch** script is simply a shell script where the top section is population with pre-processor directives **#SBATCH --option=value .** These are the same options we explored in the **srun** section. A basic example is given below, we write the following script:



```bash
#!/bin/bash
#SBATCH --job-name=sbatch-hello-world
#SBATCH --nodes=4
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=2
#SBATCH --output=%x-%j.out
#SBATCH --error=%x-%j.err

# This script submits a Slurm job that runs across 4 nodes with a total of 8 tasks.
# Each node runs 2 tasks. The job prints "Hello world!" along with the job name.
# Output and error messages are saved in files named after the job name and job ID.

working_dir=$(pwd)
job_id=$SLURM_JOB_ID
job_name=$SLURM_JOB_NAME

if [ -d ${working_dir}/tmp ]; then
    echo "directory exists"
else
    echo "creating directory..."
    mkdir ${working_dir}/tmp
fi

echod "redirecting to stderr"

srun echo "Hello world! From ${job_name}"

mv *.out $(pwd)/tmp/ &&  mv *.err $(pwd)/tmp/

exit 0
```

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

Job Arrays

One of the coolest features of **sbatch** is the ability to create job arrays i.e. a collection of similar jobs. Now as per the documentation:

All jobs must have the same initial options (e.g. size, time limit, etc.), however it is possible to change some of these options after the job has begun execution using the scontrol command specifying the JobID of the array or individual ArrayJobID. (here)

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

```
$ sbatch --array=1,3 sbatch-hello-world.sh


# Submit a job array with index values between 1 and 7
# with a step size of 2 (i.e. 1, 3, 5 and 7)
$ sbatch --array=1-4:2  sbatch-hello-world.sh
```

AFRICAN
CENTERS
OF EXCELLENCE
IN BIOINFORMATICS &
DATA-INTENSIVE SCIENCE

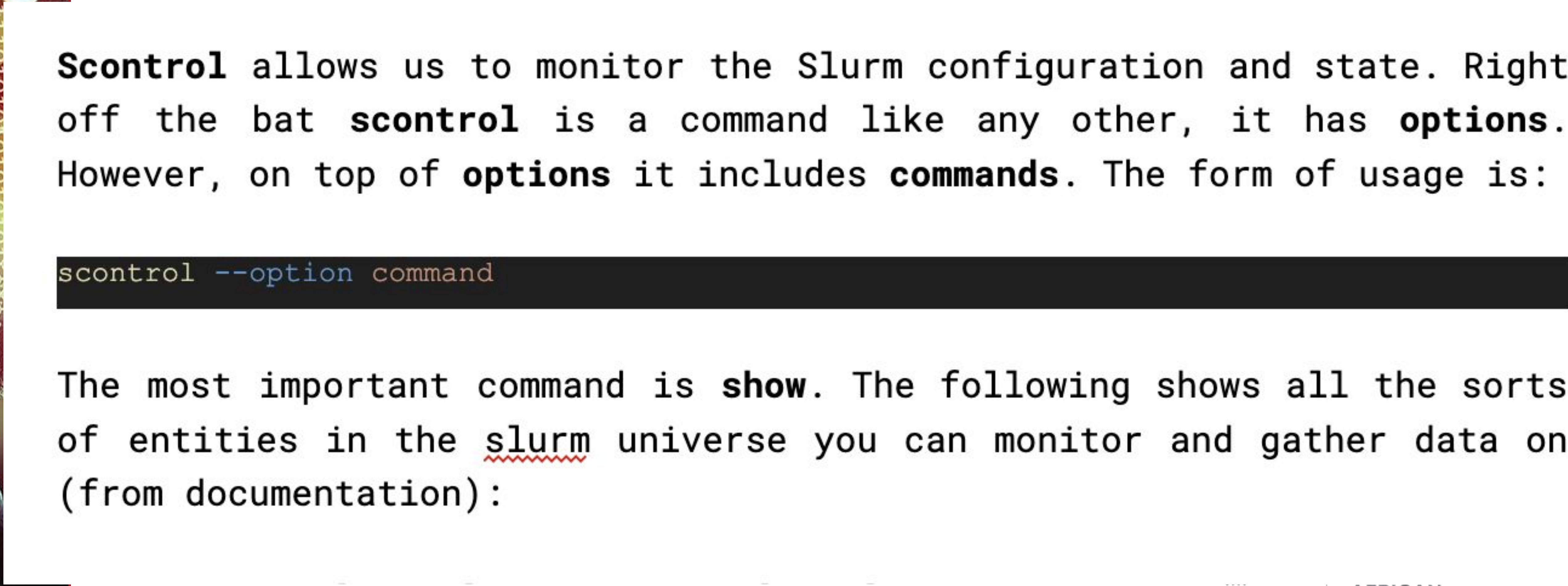
Monitor Jobs

Now we wish to either *alter, stop, or somehow act on these running jobs*. Scontrol and scancel allow us to do all this and much more.

- **idle** when it is not executing any process and when it is available for use.
- **drained** when it is currently in use but it is not accepting future jobs.
- **down** when … well it is down i.e. not operational.
- **unknown** means precisely the state is not reported or known.

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

How to Monitor SLURM Configuration and State: Scontrol

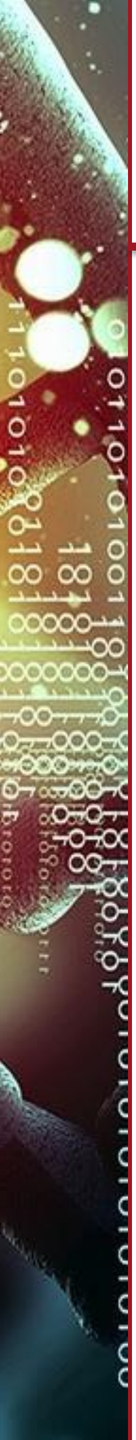


Scontrol allows us to monitor the Slurm configuration and state. Right off the bat **scontrol** is a command like any other, it has **options**. However, on top of **options** it includes **commands**. The form of usage is:

```
scontrol --option command
```

The most important command is **show**. The following shows all the sorts of entities in the slurm universe you can monitor and gather data on (from documentation):

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE



What we can “show”

- **config**

Displays parameter names from the configuration files in mixed case (e.g. SlurmdPort=7003) while derived parameters names are in upper case only (e.g. SLURM_VERSION).

- **daemons**

Reports which daemons should be running on this node.

ed list use **hostlistsorted** (e.g. tux2,tux1,tux2 = tux[1-2,2]).

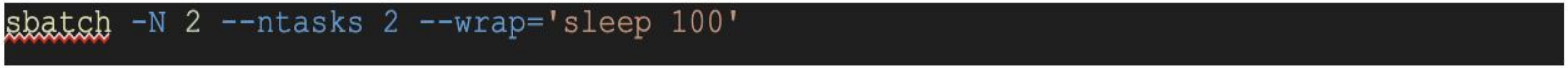
- **job**

Displays statistics about all jobs by default. If an optional jobid is specified, details for just that job will be displayed.

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

Example :

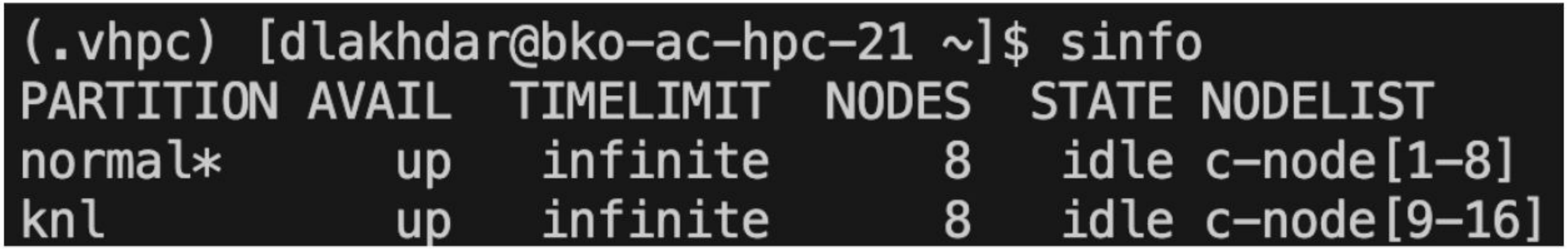
Let us say we run:



```
sbatch -N 2 --ntasks 2 --wrap='sleep 100'
```

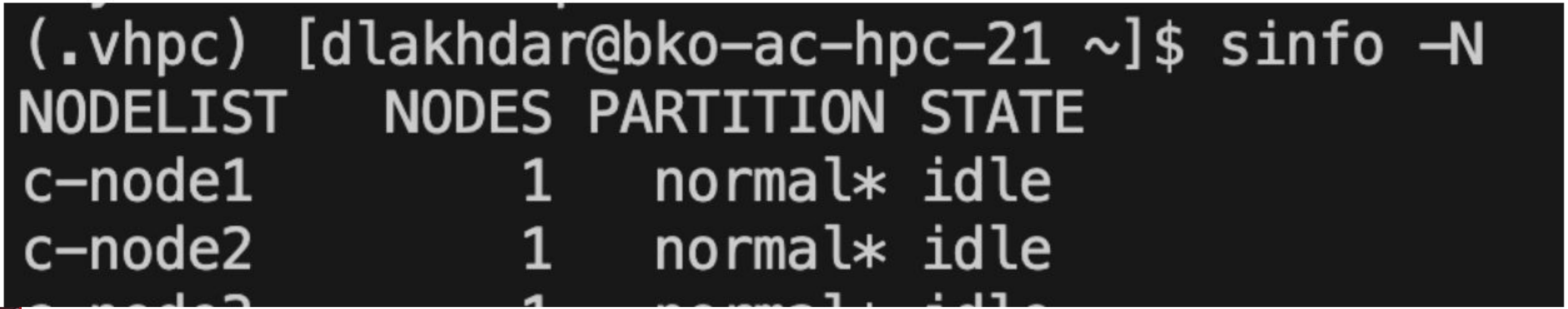
How to Monitor State: Sinfo

We can also use **sinfo** to get some more succinct information about the state of nodes and partitions.



```
(.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up   infinite      8   idle c-node[1-8]
knl          up   infinite      8   idle c-node[9-16]
```

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

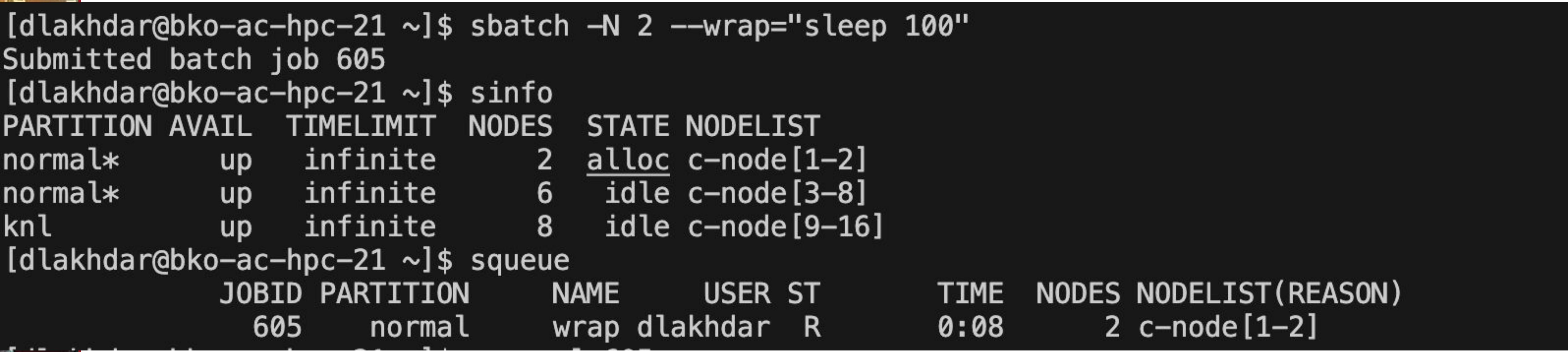


```
(.vhpc) [dlakhdar@bko-ac-hpc-21 ~]$ sinfo -N
NODELIST    NODES PARTITION STATE
c-node1         1   normal* idle
c-node2         1   normal* idle
```

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

Squeue

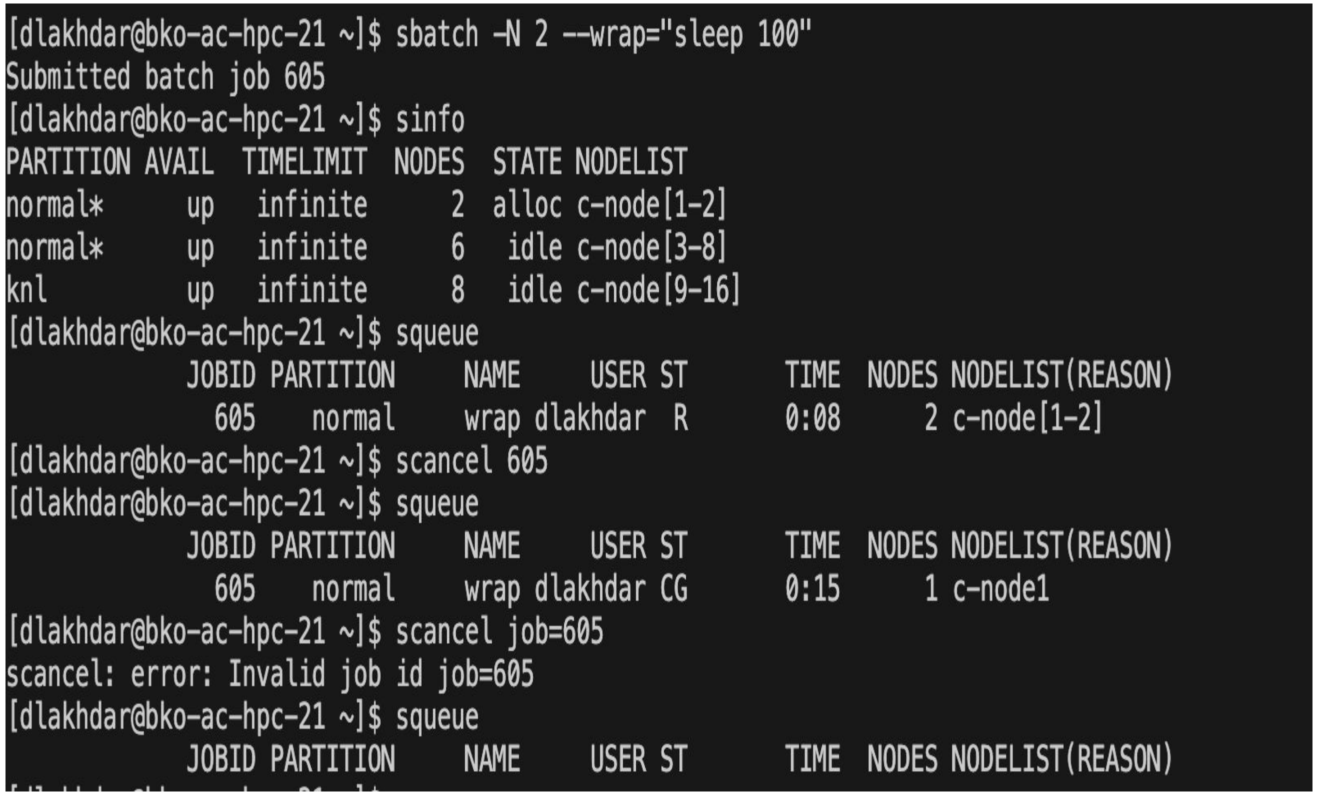
We can monitor jobs via:



```
[dlakhdar@bko-ac-hpc-21 ~]$ sbatch -N 2 --wrap="sleep 100"
Submitted batch job 605
[dlakhdar@bko-ac-hpc-21 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up   infinite      2  alloc c-node[1-2]
normal*      up   infinite      6   idle c-node[3-8]
knl          up   infinite      8   idle c-node[9-16]
[dlakhdar@bko-ac-hpc-21 ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               605    normal     wrap dlakhdar  R       0:08      2 c-node[1-2]
```

Scancel

We can cancel jobs via an scancel command, we can choose to cancel based on node, on state of nodes, based on account, or partition. The most common option is simply by job though:



```
[dlakhdar@bko-ac-hpc-21 ~]$ sbatch -N 2 --wrap="sleep 100"
Submitted batch job 605
[dlakhdar@bko-ac-hpc-21 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up   infinite      2  alloc c-node[1-2]
normal*      up   infinite      6   idle c-node[3-8]
knl          up   infinite      8   idle c-node[9-16]
[dlakhdar@bko-ac-hpc-21 ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               605    normal     wrap dlakhdar  R       0:08      2 c-node[1-2]
[dlakhdar@bko-ac-hpc-21 ~]$ scancel 605
[dlakhdar@bko-ac-hpc-21 ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               605    normal     wrap dlakhdar CG       0:15      1 c-node1
[dlakhdar@bko-ac-hpc-21 ~]$ scancel job=605
scancel: error: Invalid job id job=605
[dlakhdar@bko-ac-hpc-21 ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
```

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

MPI and Slurm

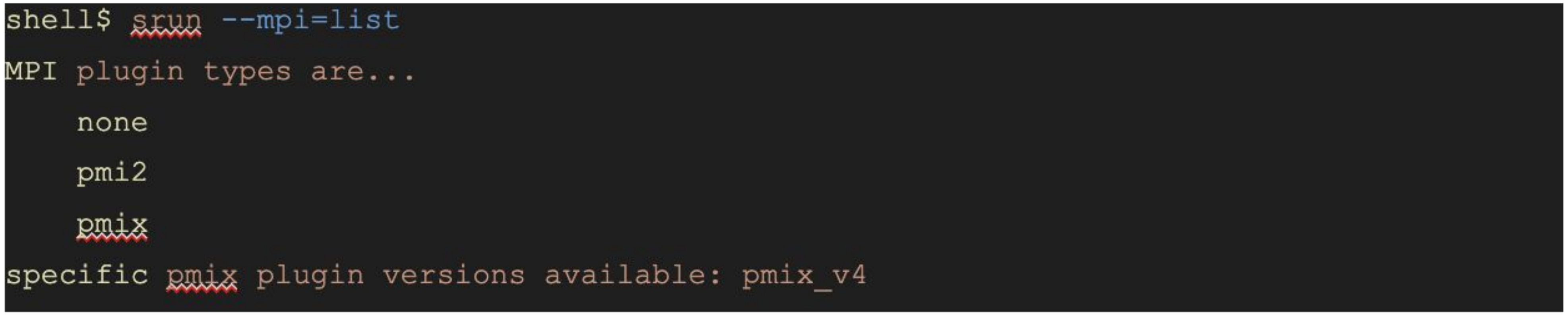
There are two ways to run mpi jobs within a slurm system. There is:

1. Use of mpiexec
2. Use of srun functionality

When we run using mpiexec, we specify the same number of processes -n as given in the node option for slurm. The mpiexec will utilize the slurm infrastructure.

When we utilize the srun functionality the slurm system must have been built with PMIx support.

MPI and Slurm



```
shell$ srun --mpi=list
MPI plugin types are...
    none
    pmi2
    pmix
specific pmix plugin versions available: pmix_v4
```

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE

The Grand Finale!

We will write python in numpy, jit-compile it with numba, then we utilize mpi to further parallelize it, finally we will deploy this mpi-numpy-numba job via an sbatch job! We will make sure that the python job is running in an interpreter that is hosted in a virtual environment to isolate the system and make sure dependencies and software do not conflict. Let us show how this works via an example:

AFRICAN CENTERS OF EXCELLENCE IN BIOINFORMATICS & DATA-INTENSIVE SCIENCE