



BioSlurm

Workshop

Using the shared BioHPC Slurm cluster

Qi Sun, Robert Bukowski
Bioinformatics Facility

What we will cover

1

Two tiers for BioHPC computing

- 'bare metal' vs BioSlurm

2

How BioSlurm is built

- Key terms
- Hardware, partitions
- Storage tiers

3

Slurm as a resource broker

- Job scripts
- Interactive jobs
- Requesting resources
- Job monitoring

4

Scheduling: how jobs compete

- When will my job start
- Backfilling
- Reasons for waiting
- Priority

5

Job accounting

- Was #CPUs adequate?
- Was RAM adequate?

6

Where to find documentation

7

How to avoid reading documentation

- BioSlurm and AI agents

Computing at BioHPC

Reserved server (“bare metal”)

- You control the whole machine
- All CPUs, RAM, GPUs, and local disks and I/O bandwidth are available to you
- You can usually run tasks whenever you want
- Good for interactive access
- Charged per hour of reservation
- Unused resources unavailable to others

Shared cluster (BioSlurm)

- Request only a ‘slice’ of a compute node
- Use only the requested number of CPUs/GPUs, RAM, local disk. I/O bandwidth shared among multiple users
- Your jobs may have to wait in queue when the cluster is busy.
- Interactive access less convenient
- Flat fee (at least on BioSlurm)
- Better overall utilization: idle resources on a node can be used by someone else.

Efficiently sharing resources requires a scheduler

- **SLURM – formerly: Simple Linux Utility for Resource Management**

Key Slurm terms

Job

A program or script submitted to Slurm

Job ID

The unique number assigned to a submitted job

Compute node

A server on which Slurm runs jobs

Resources

The CPUs, memory, GPUs, local scratch storage, and time requested by a job. Exposed as **`$SLURM_TMPDIR`**

Partition

A group of nodes with a shared purpose or time limit, such as short, long, debug, or gpu

Batch job

A non-interactive job submitted with **`sbatch`**

Interactive job

A live shell or command session on a compute node

Scratch

Temporary, fast, node-local storage available through **`$SLURM_TMPDIR`**.

BioSlurm at a glance

5

CPU production nodes

3

GPU production nodes

1

training node

4

main partitions

Node	Slurm CPUs	Memory available to Slurm	Local scratch	Other resources	Partitions
cbsulm09	64	515434 MB	7500 GB	—	short, long, debug
cbsulm10	64	515434 MB	7500 GB	—	short, long, debug
cbsulm14	112	515312 MB	10500 GB	avx2 feature	short, long, debug
cbsumm28	40	257298 MB	6500 GB	avx2 feature	short, long
cbsumm29	40	257298 MB	6500 GB	avx2 feature	short, long
cbsugpu02	24	257290 MB	4500 GB	1 Tesla P100 GPU	gpu
cbsugpu03	32	257290 MB	10500 GB	2 Tesla P100 GPUs	gpu
cbsugpu06	112	514897 MB	6000 GB	2 NVIDIA A40 GPUs	short, gpu
cbsumm35	48	257059 MB	5000 GB	avx2 feature	training



BioSlurm partitions (queues)

Partition	Max job time	Use	Important detail
short	≤ 24 h	CPU jobs	default
long	≤ 7 days	CPU jobs	At most 112 CPUs and 755 GB RAM used at any time (across all jobs, all users)*
gpu	≤ 24 h	GPU jobs	GPU GRES required
debug	≤ 1 h	testing	higher priority
training	≤ 1 h	workshop	restricted account

* QOSGrpCpuLimit, QOSGrpMemLimit



Three storage tiers

Choose storage based on persistence and I/O pattern.

/home

NETWORK

Best for

scripts • configs • logs • final
results

Watch out

Avoid I/O-intensive computation

/home2

NETWORK

Best for

shared inputs • references •
persistent data

Watch out

Useful across jobs; can bottleneck
under heavy concurrent I/O

\$SLURM_TMPDIR

NODE-LOCAL

Best for

temporary files • sort space •
intermediates

Watch out

Fast and private; quota-limited;
deleted at job end

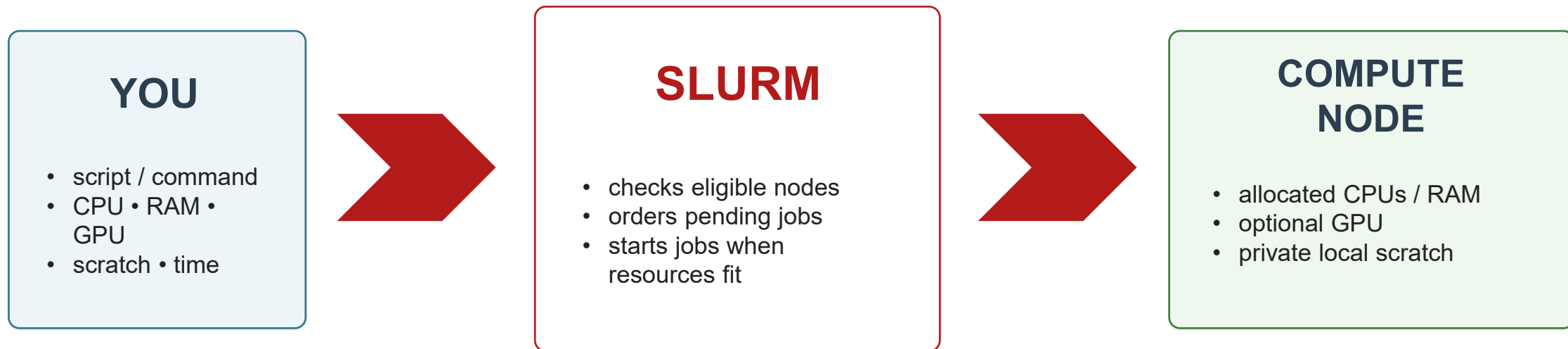
Rule of thumb: compute in `$SLURM_TMPDIR`; preserve what matters elsewhere before the job exits.



CONCEPT

Slurm is a resource broker

You describe what a job needs; Slurm decides when and where it can run.



Key idea: requesting more than you need can make the job harder to place; requesting too little can make it fail.



A good BioSlurm batch script (lets call it `analysis.sh`)

Stage data → compute in local scratch → copy results back.

```
#!/bin/bash -l
#SBATCH --cluster=bioslurm
#SBATCH --partition=short
#SBATCH --cpus-per-task=8
#SBATCH --mem=32G
#SBATCH --time=04:00:00
#SBATCH --gres=scratch:200
#SBATCH --output=analysis.%j.out
#SBATCH --mail-user=abc123@cornell.edu
#SBATCH --mail-type=ALL
#SBATCH --constraint=avx2

cd "$SLURM_TMPDIR"
cp /home2/$USER/project/input.bam .
my_program --threads "$SLURM_CPUS_PER_TASK" input.bam
cp result.dat /home2/$USER/project/results/
touch "$SLURM_TMPDIR/.job_completed_ok"
```

Why this pattern works

- Heavy temporary I/O stays on node-local disk
- Permanent files live on network storage
- CPU request matches the program thread count
- A completion marker helps preserve logs after abnormal termination
- E-mail options notify about job state changes
- avx2-enabled node requested (optional)

Submit: `sbatch analysis.sh`



A BioSlurm batch script requesting a GPU

Stage data → compute in local scratch → copy results back.

```
#!/bin/bash -l
#SBATCH --cluster=bioslurm
#SBATCH --partition=gpu
#SBATCH --cpus-per-task=4
#SBATCH --mem=32G
#SBATCH --time=04:00:00
#SBATCH --gres=gpu:tP100:1,scratch:200
#SBATCH --output=analysis.%j.out
#SBATCH --mail-user=abc123@cornell.edu
#SBATCH --mail-type=ALL

cd "$SLURM_TMPDIR"

nvidia-smi
echo $CUDA_VISIBLE_DEVICES

cp /home2/$USER/project/input.txt .
python my_gpu_program.py input.txt >& result.dat
cp result.dat /home2/$USER/project/results/
```

The job is requesting **one GPU of type tP100**

GPU types available in BioSlurm:

tP100	(x 3)
A40	(x 2)

Submit: `sbatch analysis.sh`

SLURM: Job arrays

Consider script `my_array_script.sh` (objective: compress files `BBB_1`, `BBB_2`, `BBB_3`, `BBB_4`)

```
#!/bin/bash -l
#SBATCH --cluster=bioslurm
#SBATCH --partition=short
#SBATCH --cpus-per-task=4
#SBATCH --mem=32G
#SBATCH --time=04:00:00
#SBATCH --gres=scratch:200
#SBATCH --output=analysis.%j.out
#SBATCH --array=1-4%2

cd $SLURM_TMPDIR

# Copy the file to gzip from network-mounted directory
cp /shared_data/Parallel_workshop/BBB_${SLURM_ARRAY_TASK_ID} .

# Run the compression
gzip BBB_${SLURM_ARRAY_TASK_ID}

# Copy the result back into the result directory (here: same
as # submission dir)
cp BBB_${SLURM_ARRAY_TASK_ID}.gz $SLURM_SUBMIT_DIR
```

Submit this script as job array:

```
sbatch my_array_script.sh
```

- this single command will submit four separate jobs, each running `my_array_script.sh`
- up to two of these jobs will be running simultaneously (%2)
- in each job the value of `SLURM_ARRAY_TASK_ID` will be different (one of 1, 2, 3, or 4)
- `SLURM_ARRAY_TASK_ID` variable is provided by Slurm as a result of using the `--array` option.
- This is how independent tasks can be run **in parallel**



Interactive jobs are still scheduled jobs

Useful for debugging, testing, and exploratory work.

```
1 srun --cluster=bioslurm \  
2   -N 1 -n 1 --cpus-per-task=4 \  
3   --mem=16G --gres=scratch:100 \  
4   -t 04:00:00 -p short --pty bash -l
```

What happens

- The command waits for resources
- You land on an allocated compute node
- `exit` ends the allocation

For a quick test, use the debug partition (≤ 1 hour).

For a GPU shell, request both `-p gpu` and an explicit GPU GRES.

Requesting resources

These parameters may be specified at the top of the script with **#SBATCH** notation, or as arguments to the **sbatch** command.

Parameter	What it does
<code>--cluster=bioslurm</code> <code>-M bioslurm</code>	Specify the cluster, for this tutorial it should always be bioslurm.
<code>--job-name=job_name</code>	Name your job, this will help identify it in the squeue report
<code>--output=job.%j.out</code>	Output file produced by the job. The %j is replaced with the SLURM_JOB_ID assigned to the job.
<code>--nodes=1</code> <code>-N 1</code>	The number of servers your job needs. This should always be one (it is the default if <code>--ntasks=1</code>), except for advanced usage.
<code>--ntasks=1</code> <code>-n 1</code> <code>--cpus-per-task=8</code>	The number of CPU cores your job will use is equal to the product of these two parameters. For non-MPI jobs, <code>--ntasks=1</code> is the default and <code>--cpus-per-task</code> is sufficient to specify the number of cores requested.
<code>--mem=16G</code>	The amount of RAM your job will be allotted, in this case 16 gigabytes. Your job will not be able to exceed this value, so better to over-estimate here.
<code>--time=04:00:00</code>	The time limit for your job (it cannot be extended). It is in the format HOURS:MINUTES:SECONDS, or DAYS-HOURS:MINUTES:SECONDS. The default is the maximum for the partition you specify (see below about partitions)
<code>--gres=scratch:200</code>	Specifies the amount of local hard drive space your job will be allocated in GB. The default is 50 GB.
<code>--partition=long</code> <code>-p long</code>	Specifies the "partition" where the job should run, options are short, long, gpu, debug. See Partitions and limits for more details.
<code>-p gpu -gres=gpu:A40:2,scratch=200</code>	Request to GPUs and 200GB of scratch space
<code>--email-user=abc@address.com</code> <code>--email-type=ALL</code>	Whom to notify about job state changes (options are BEGIN, END, FAIL, REQUEUE, ALL)



Before you submit: five questions



Is the partition appropriate for the runtime and hardware?



Are CPU and memory requests realistic for the software?



Did I request a GPU explicitly if I need one?



Is scratch large enough — but not excessive?



Will important output be copied off local scratch?

The goal is not to “game” the scheduler — it is to describe your job accurately so Slurm can place it efficiently and fairly.



Jobs submitted. How are they doing?

(run on a login node)

```
squeue -u $USER -M bioslurm  
squeue_l
```

See your pending/running jobs
See all pending/running jobs in more detail

```
scontrol show job <JobID> -M bioslurm
```

See allocation, pending reason, priority for a pending/running job

```
sacct_l -j <JobID> -M bioslurm  
sacct_l -M bioslurm
```

Inspect usage, state, timing info, peak memory of a finished job
Inspect usage, state, timing info, peak memory of all finished jobs

```
sinfo -M bioslurm
```

See partition/node availability

```
scancel <JobID> -M bioslurm
```

Cancel a job

To avoid typing **-M bioslurm**, set the environment variable **SLURM_CLUSTERS**:

```
export SLURM_CLUSTERS=bioslurm
```



I submitted a job. When will it start?

If no other jobs pending:

- if enough CPUs, RAM, local scratch space, or GPUs available on any of the (explicitly or implicitly) requested nodes
- if starting my job will not violate limits on total CPU or RAM in long partition (QOSGrpCpuLimit, QOSGrpMemLimit)
- if my job does not depend on completion of other job(s)

If there are other jobs pending:

- if all conditions mentioned above are met, **and...**
- other jobs have priority lower than mine **or...**
- my job is short enough to backfill

Why is my job still pending? When will it start? Ask Slurm:

```
queue_1 -j JOBID
```

Among information shown, there will be (REASON) and estimated **START_TIME**

Common (REASON) values	meaning
(Resources)	Available resources currently not sufficient
(Priority)	There are other pending jobs with higher priority
(QOSGrpCpuLimit)	Starting your job would exceed overall cap on CPUs in long partition
(QOSGrpMemLimit)	Starting your job would exceed overall cap on RAM in long partition
(Dependency)	Job depends on completion of some other job(s)
(ReqNodeNotAvail)	Nodes needed are not available



Backfilling turns scheduling gaps into useful work

BioSlurm uses Slurm's backfill scheduler.



A lower-priority short job may still run first if it can backfill into a time gap without delaying higher-priority work.

Why it matters: a realistic short time limit can make a job easier to backfill, even if its formal priority is not the highest.



Priority is multifactor

10,000 fair-share weight	1,000 age weight	1,000 job-size weight	1,000 partition weight
Fair share	dominant weight	Jobs from users/groups who recently consumed a lot of CPU/RAM/scratch wait longer	
Age	modest weight	Waiting longer gradually raises priority	
Job size	modest weight	'Larger' (in terms of CPU number) jobs receive more job-size priority	
Partition	modest weight	debug has a higher priority tier	

*weights may be adjusted if need arises

Fair Share factor on BioSlurm

Usage

Individual:

(slice of a node) * time occupied, summed over this user's jobs
'slice': based on CPUs or RAM, whichever is greater

Group (membership account):

individual usages summed over users in group

Fair share computation

Sort groups (membership accounts) over group usage

Sort users within groups over their individual usage

User's Fair Share factor ~ 1/rank
(the lower the rank the better)

4-hour usage half-life

Recent usage matters most. Every 4 hours, the contribution of past usage is halved.

(half-time may be adjusted in the future!)

Consequences:

'heavy-use' groups have lower priority over 'light-use' groups

'heavy users' have lower priority 'than light' users

'light user' from 'heavy-use' group has lower priority than any user from 'light-use' group



Maximize your chances of starting sooner

Make the job easier for the scheduler to fit.

1

Ask for what you will use

Do not reserve 32 CPUs for software that uses 4.

2

Use a realistic time limit

Shorter jobs are easier to backfill.

3

Use short when ≤ 24 h

long has an aggregate CPU/RAM caps.

4

Keep scratch realistic

Scratch is consumable and can itself block placement.

5

Use arrays for independent work

Limit concurrency with %N when appropriate.

6

Use debug only for genuine tests

It is intentionally higher priority but capped at 1 hour.

Job has finished. Use accounting to evaluate performance

`sacct_l -u <userID>` Will show timing and memory information for finished jobs
`sacct_l -j <jobID>`

Signature of an efficient job:

CPUTime (= NCPU*Elapsed) $\approx$ TotalCPU $\approx$ UserCPU

MaxRSS $<$ ReqMem

Signs of inefficiency:

TotalCPU $<$ CPUTime inefficient parallelization (e.g., due to sequential part of job, like file copying)

UserCPU $<$ TotalCPU lot of time spent is system calls (typically due to inefficient I/O)

MaxRSS $\approx$ ReqMem job may have not enough memory, may crash or swap; increase --mem

MaxRSS: use this to estimate memory needs of your job

□ Run test requesting a lot of memory, then for production runs set --mem to a value slightly larger (1.2 times?) than MaxRSS obtained from the test run

Slurm commands cheat sheet

Cluster information	
<code>sinfo</code> <code>scontrol show nodes</code> <code>scontrol show partitions</code>	Cluster occupancy summary Compute nodes detail Partition details
Job submission	
<code>sbatch script.sh</code>	Submit a script
<code>sbatch --mem=20g --gres=100G script.sh -p long</code>	Submit a script with request parameters on command line (will override #SBATCH directives)
<code>srun -N 1 -n 1 --cpus-per-task=4 --mem=16G --gres=scratch:100 -t 04:00:00 -p long --pty bash -l</code>	Request interactive shell with specified resource requests
Job monitoring	
<code>squeue</code> <code>squeue -u \$USER</code> <code>squeue_l</code> <code>squeue_l -u \$USER</code> <code>squeue_l -j JOBID</code> <code>scontrol show job <JobID></code>	Information about running and pending jobs Information about my running and pending jobs Details of running/pending jobs (incl. reason pending, priority, and est. start time) Details of my running/pending jobs Details for job <JobID> More details about job <JobID>
Job control	
<code>scancel <JobID></code>	Cancel job <JobID>
<code>scontrol hold <JobID></code> <code>scontrol release <JobID></code> <code>scontrol update job=<JobID> timelimit=2:00:00</code>	Suppress start of a job Release a suspended job Change time limit of a pending job
Finished job inspection	
<code>sacct</code> <code>sacct_l</code> <code>sacct_l -j <JobID></code> <code>sacct_l -S 2026-09-01</code>	Basic information about finished job(s) Detailed job information including completion status, timing, and RAM usage Details of a specific job <JobID> Details of jobs started since a given date

If **SLURM_CLUSTERS** is not set, add **--cluster=bioslurm** or **-M bioslurm** to each of the above commands

Rigorous Slurm documentation

- ❑ Version installed on BioHPC: <https://slurm.schedmd.com/archive/slurm-22.05.9/>
- ❑ Man pages for individual commands: https://slurm.schedmd.com/archive/slurm-22.05.9/man_index.html
- ❑ SURM command summary handout: <http://slurm.schedmd.com/pdfs/summary.pdf>

- ❑ Formal documentation very thorough, but rather formal, with few specific examples
 - AI to the rescue!



BioSlurm + AI agents

Use the agent to prepare work; keep resource decisions and submission visible.

1

**Tell the agent to
use BioSlurm**

2

Give it bioslurm.md

3

**Ask for CPU / RAM
/ scratch reasoning**

4

**Review the
generated
submission script**

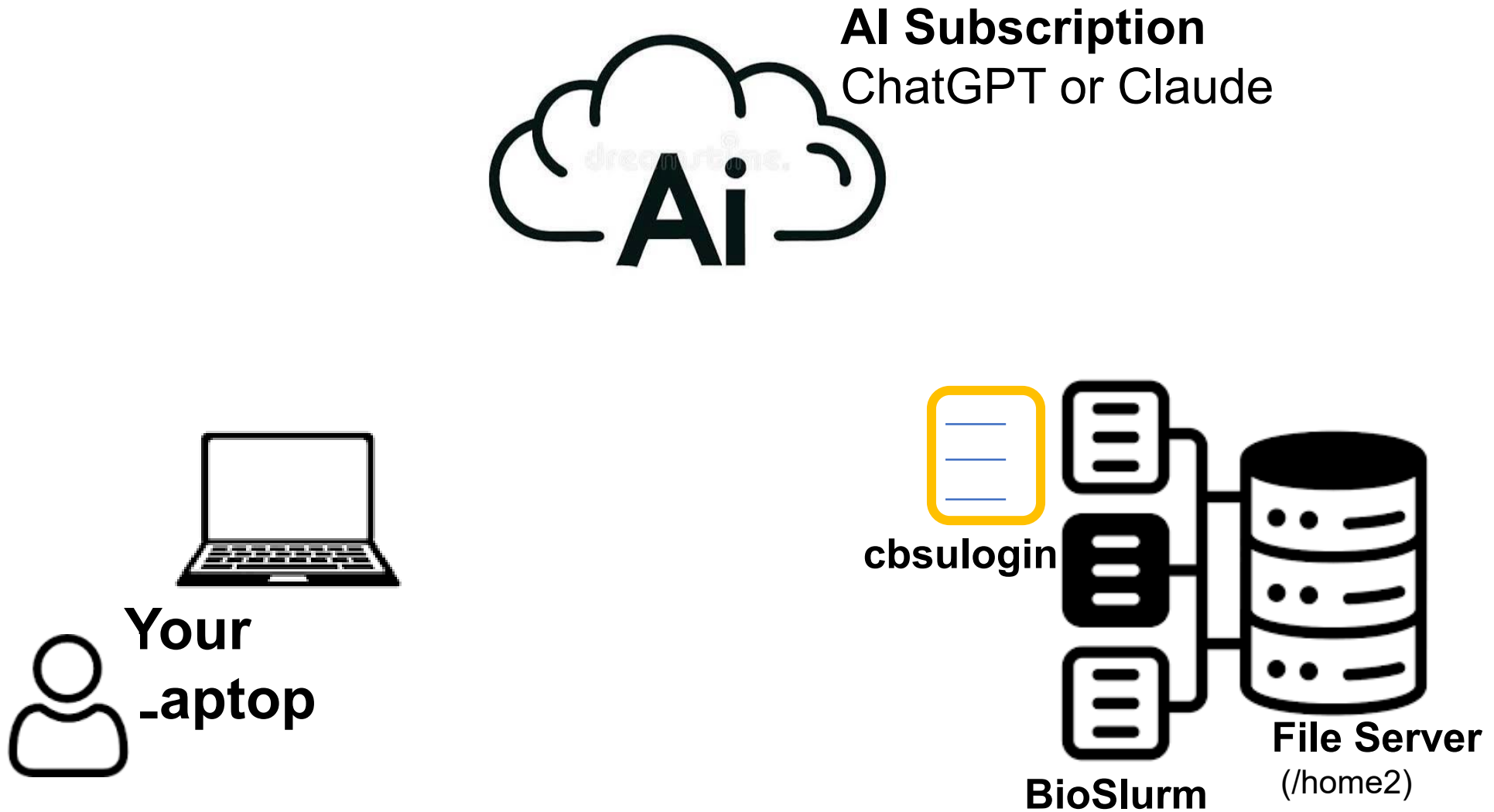
5

**Approve and
submit**

```
1 /programs/ai_pipelines/slurm/bioslurm.md
```

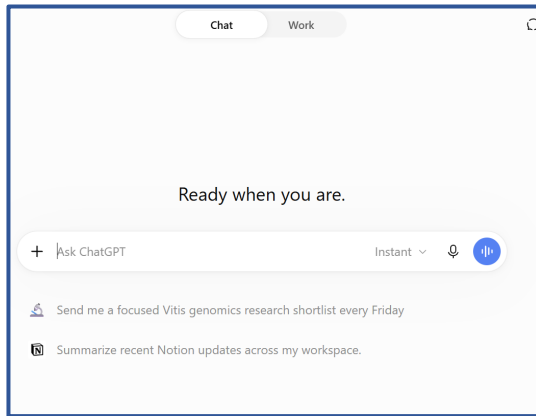
Recommended prompt pattern: describe inputs, desired outputs, software, and ask the agent to justify CPU, RAM, GPU, scratch, and time estimates.

What Do You Need for AI Agent-Assisted Data Analysis



AI Chatbot vs AI Agent

AI ChatBot



Linux terminal

```
qisun@cbsulogin:~  
Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.  
  
PS C:\Users\qs24> ssh qisun@cbsulogin  
  
Welcome to BRC Bioinformatics Facility BioHPC Lab  
server information: cbsulogin, 24 cores, 128GB RAM, Rocky 9.8  
  
Last login: Tue Sep 29 16:52:18 2026 from 128.84.194.149  
Do not run any computing software on Login nodes  
Skipping procsched.pl  
[qisun@cbsulogin ~]$ |
```



The chatbot does not have access to your files or you emails

AI Agent

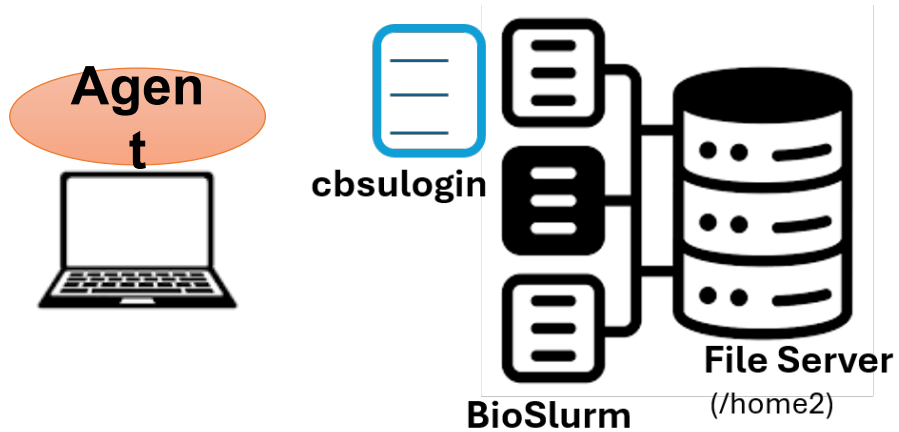
```
🌟 Update available! 0.157.0 -> 0.160.0  
Run npm install -g @openai/codex to update.  
  
See full release notes:  
https://github.com/openai/codex/releases/latest  
  
> _ OpenAI Codex (v0.157.0)  
model: GPT-6-Sol medium /model to change  
directory: /home2/qisun/testbioslurm  
  
Tip: Use /subagents to switch between this session's subagents.  
  
> Ask Codex to do anything  
  
GPT-6-Sol medium · /home2/qisun/testbioslurm
```

The agent has access to your files and possibly you emails

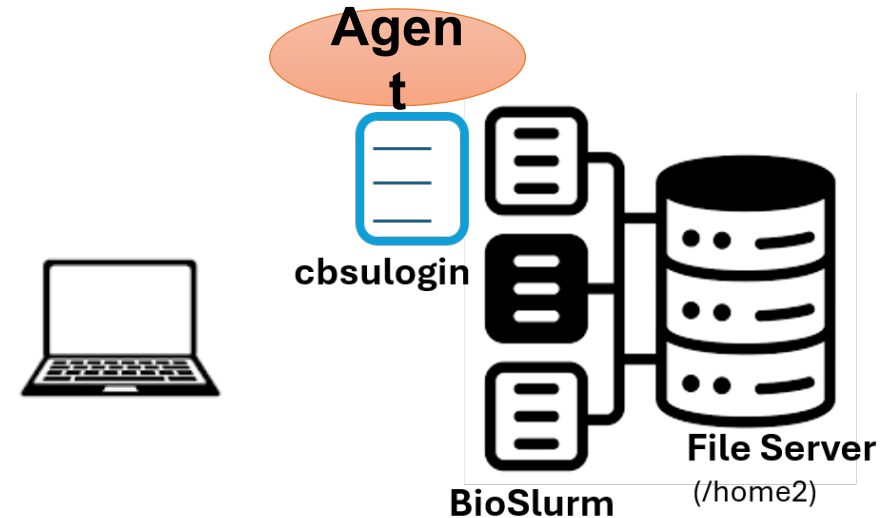
Be cautious if you have sensitive data on the server

Where Does the AI Agent Run?

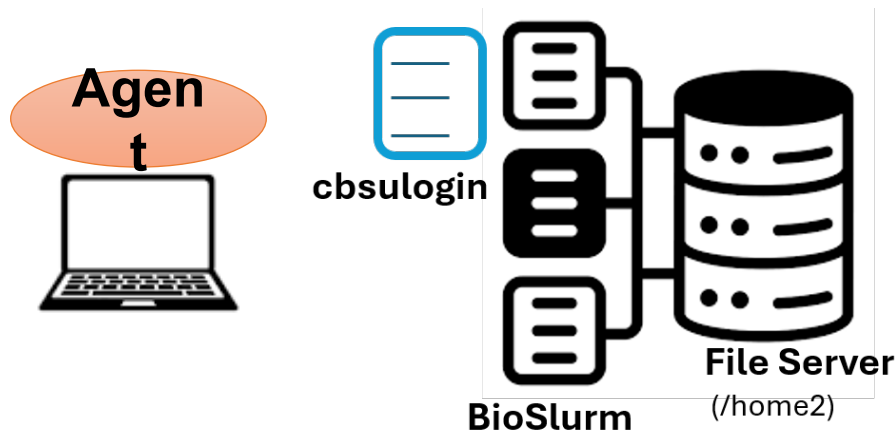
Agent runs on your laptop



Agent runs on cbsulogin(3)



Security concern 1. Agent runs on your laptop

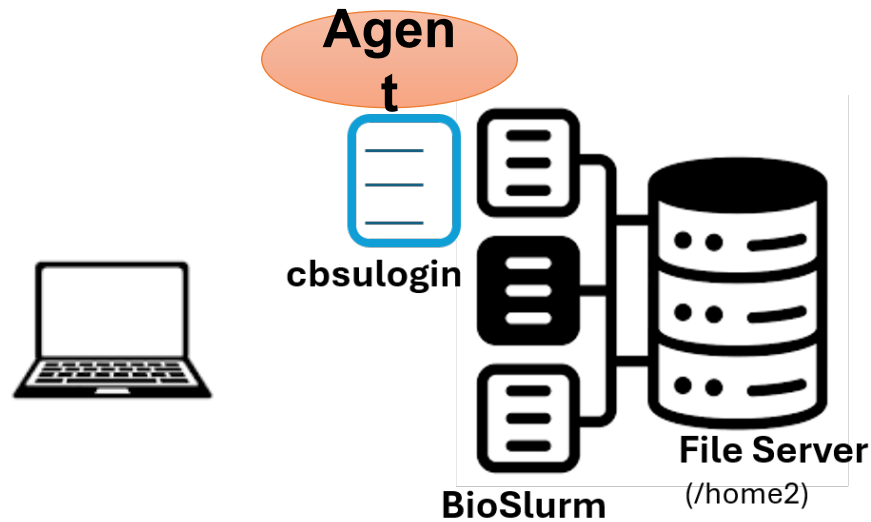


Examples:

- **Claude Science** : Fully contained by default
- **Claude Code**: Can access your files
- **Codex Desktop**: Can access your files

* Sandbox on the laptop, no sandbox on the server

Security concern 2. Agent runs on cbsulogin(3



- “Sandboxed” on server.
- Agent cannot write, but it can read files outside the project directory

* **Project directory** is the directory where you start the command “codex” or “claude”

Two BioHPC Skills for AI Agents

BioSlurm:

bioslurm.md

(/programs/ai_pipelines/slurm/bioslurm.md)

Rental & Hosted Servers:

AGENTS.md

(/programs/ai_pipelines/AGENTS.md)

- **Do not ask the agent to read both — they may conflict.**

AI Agent Misalignment

AI agents may try to accomplish your goal in ways you did not intend.

Do not remove this line from the guardrail files.

You may optimize Slurm submissions for faster turnaround using permitted scheduler mechanisms, but never attempt to bypass, manipulate, exploit, or circumvent Slurm scheduling, priority, fair-share, QoS, accounting, access controls, or other cluster policies.

If you use “**Claude Science**” on your laptop

<https://biohpc.cornell.edu/lab/userguide.aspx?a=software&i=1364#c>

- Use port 2222;
- Provide this guardrail to Claude Science (it can be included in the skill file):

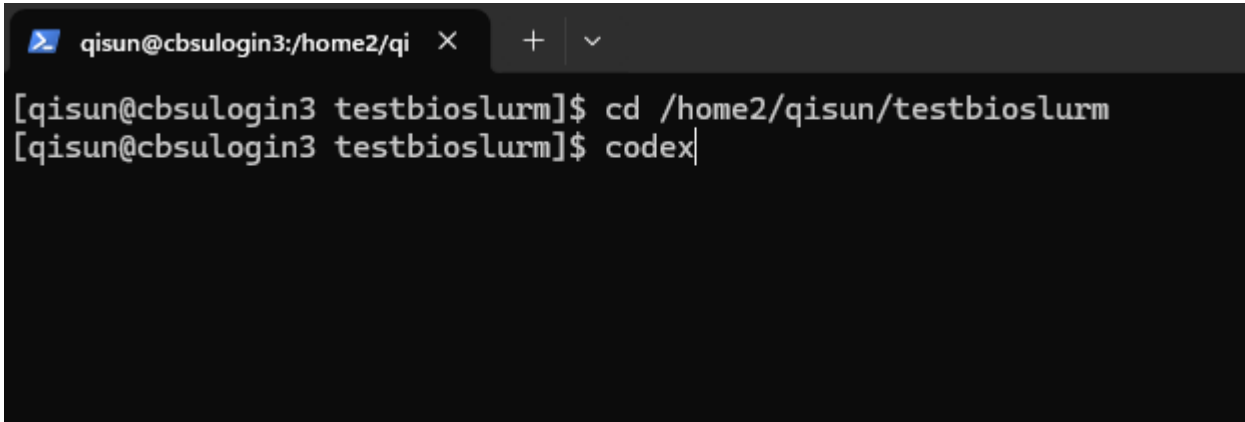
When accessing bioslurm, minimize and reuse SSH connections. The server limits new connections from the same IP to fewer than 10 per minute. Do not repeatedly reconnect, probe, or poll the server.

Hands-on projects

- **Submit your first BioSlurm job**
- **Parallelize a Bioinformatics Analysis**
- **Request Specific Computing Resources**
- **Run Nextflow pipeline on BioSlurm**

How to run AI Agent running on cbsulogin(1)

- Ssh to cbsulogin or cbsulogin3
- Launch “codex” or “claude” from the **project directory**

A screenshot of a terminal window with a dark background. The window title bar shows 'qisun@cbsulogin3:/home2/qi' and standard window controls. The terminal displays two lines of text: the first line shows a prompt '[qisun@cbsulogin3 testbioslurm]\$' followed by the command 'cd /home2/qisun/testbioslurm'; the second line shows the same prompt followed by the command 'codex|'.

```
qisun@cbsulogin3:/home2/qi X + v
[qisun@cbsulogin3 testbioslurm]$ cd /home2/qisun/testbioslurm
[qisun@cbsulogin3 testbioslurm]$ codex|
```

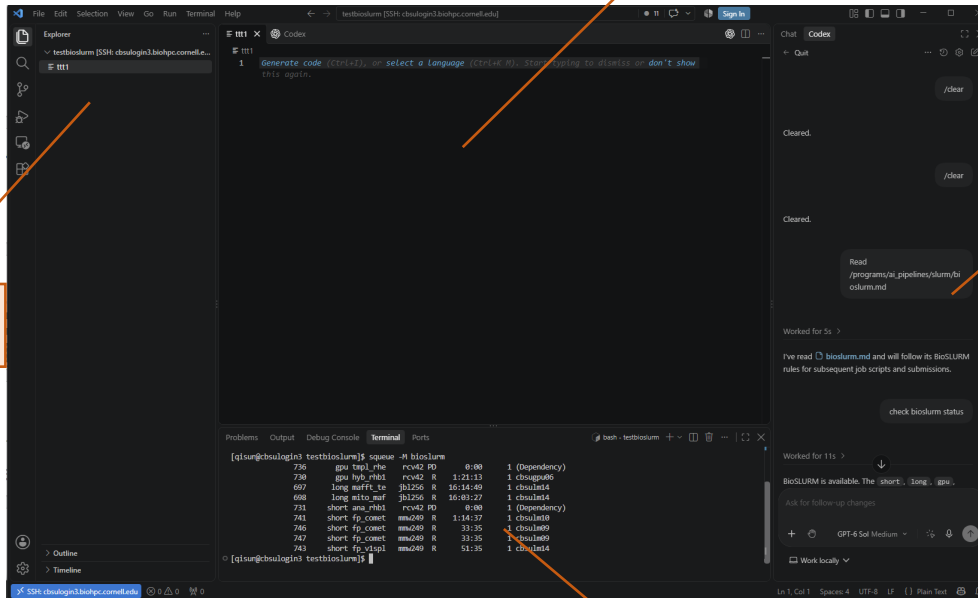
How to run AI Agent running on cbsulogin(2)

Through VS Code

File editor/viewer

File browser

Agent



Instructions to setup

Terminal

Codex: https://biohpc.cornell.edu/doc/setup_account_codex_v2.html

Claude: https://biohpc.cornell.edu/doc/setup_account_claude_v2.html

Storage

/home2/<userID>

- **Preferred data storage**
- Accessible from all nodes
- 200GB quota applies

/home/ <userID>

- Accessible from all nodes
- **Not for large files**
- 200GB quota applies

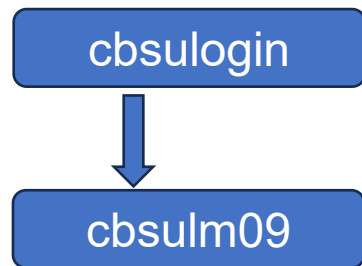
/workdir/<userID>/<JobID>
(\$SLURM_TMPDIR)

- **Good for temporary files**
- No quota
- Not accessible across nodes

Parallelization

- AI does not always choose the best parallelization strategy.
- Ask the AI agent to explain its parallelization strategy before running the job.

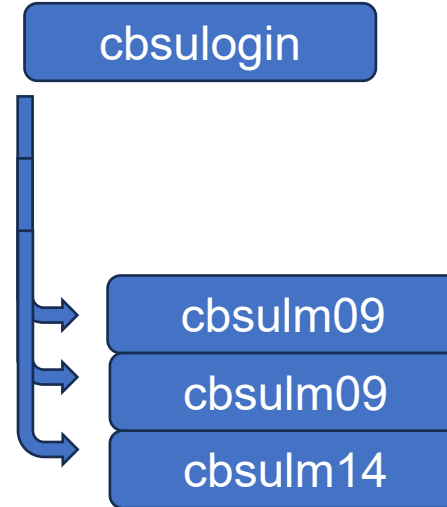
1. One node, parallelize within the node



Pros: Same /workdir/

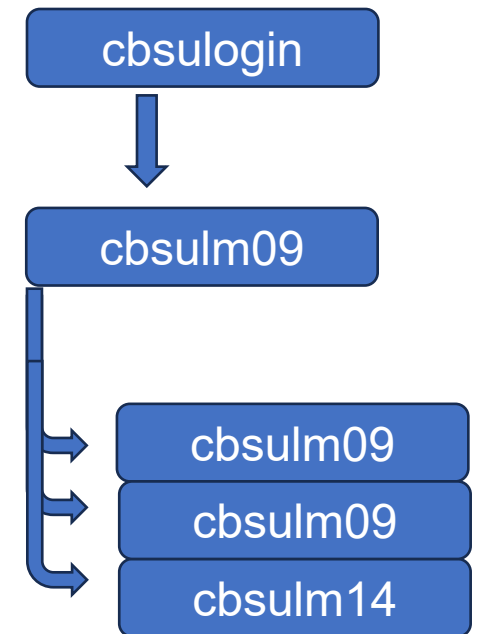
Cons: Difficult to get one node if cluster is busy

2. Multiple independent jobs



- Most common
- Use /home2/ as shared directory

3. An orchestrator job that submits many Slurm jobs



Nextflow; Snakemake

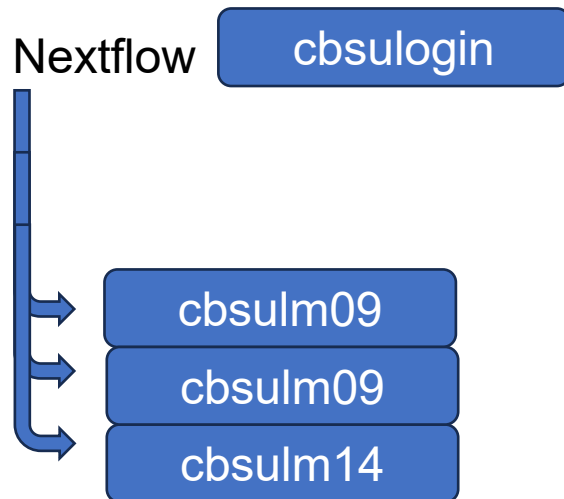
Hands-on projects

- **Submit your first BioSlurm job**
- **Parallelize a Bioinformatics Analysis**
- **Request Specific Computing Resources**
- **Run Nextflow pipeline on BioSlurm**

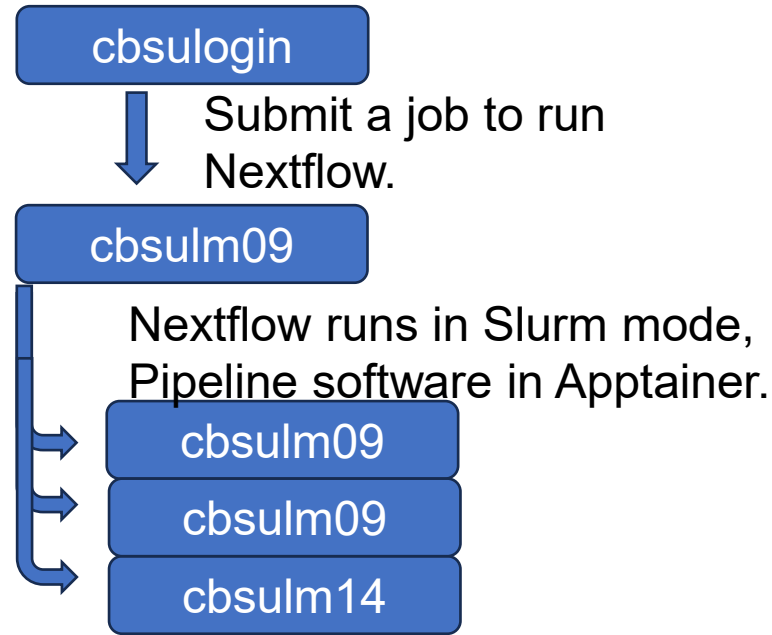
Working with Nextflow



Nextflow run on login node



Nextflow run on login node



Nextflow run on login node

