



Kimmo Mattila
Ari-Matti Sarén



CSC Bioweek 2018

Computing intensive bioinformatics analysis on Taito

7. 2. 2018



CSC Environment



Sisu

- Cray XC40 Massively Parallel Processor (MPP) supercomputer
 - 3376 12-core 2.6-GHz Intel Haswell 64-bit processors
 - 40512 cores
 - 2,67 GB memory/core
 - Aires interconnects

- Meant for jobs that parallelize well
 - Normally 64-4096 cores/job (MPI)
 - can be increased for Grand Challenge projects

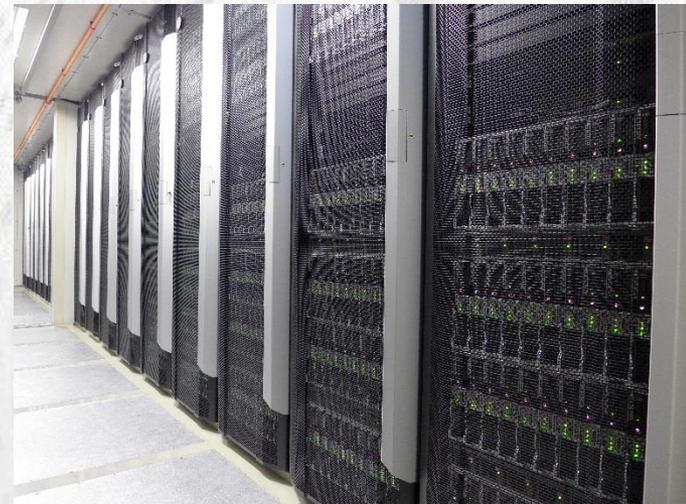
- Modest selection of bioinformatics tools
 - Molecular dynamics codes: gromacs, namd, Amber

- Sisu user's guide
 - <http://research.csc.fi/sisu-user-guide>



HP Apollo 6000 XL230a/SL230s Supercluster

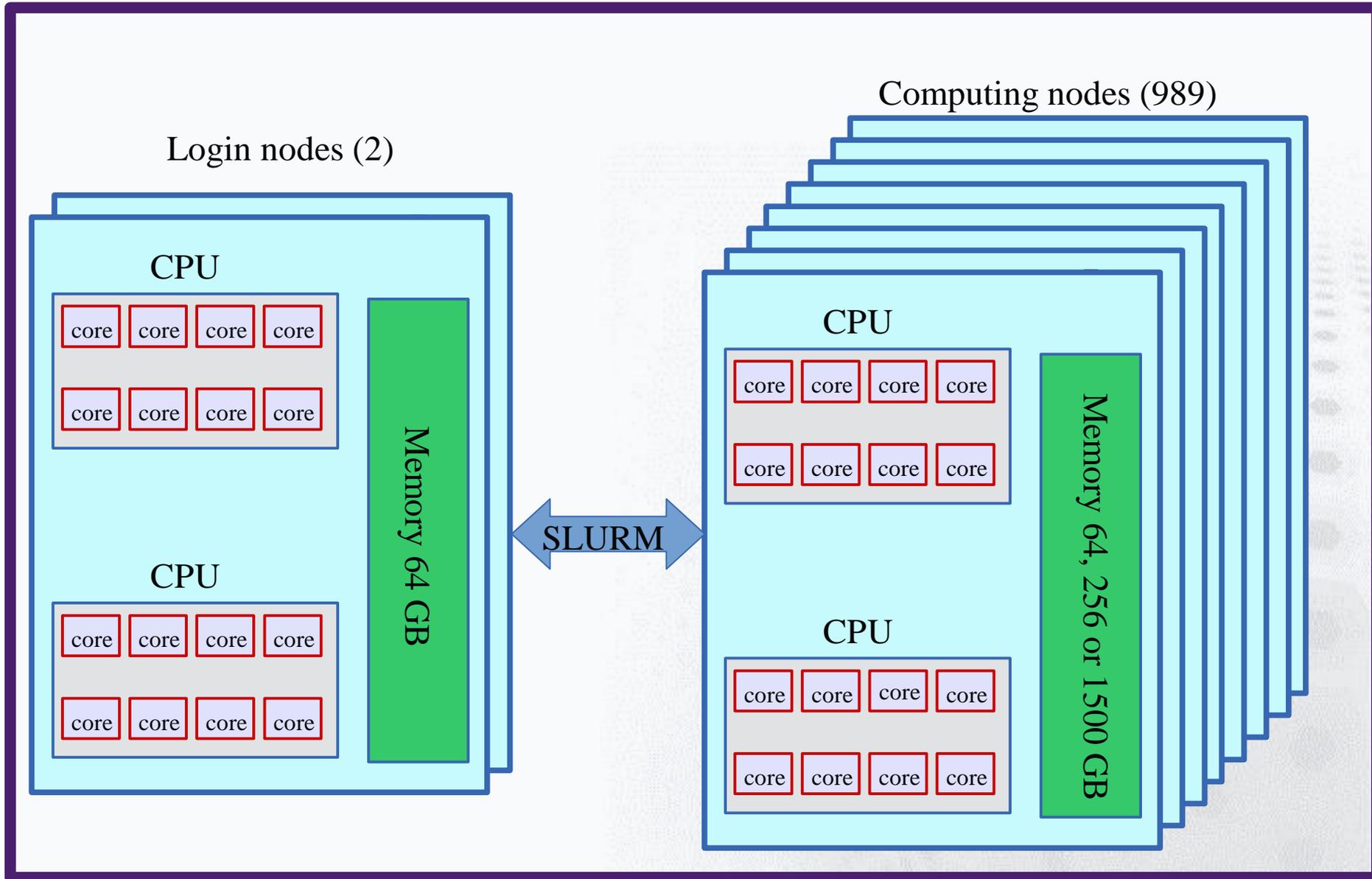
- Phase 1 (2013)
 - 2 x 8 core 2.6 GHz Intel Sandy Bridge processors
 - 64 GB / 256 GB / 1,5 TB per node
- Phase 2 (2015)
 - 2 x 12 core 2.6 GHz Intel Haswell processors
 - 128 GB / 256 GB / 1,5 TB per node
- Phase 3 (late 2018?)
- Also GPU and MIC nodes available
- Meant for serial and mid-size parallel jobs
- 1-672 cores/job (more possible after scalability tests)
- Maximum of 896 simultaneous jobs/user
- Taito user's guide: <http://research.csc.fi/taito-users-guide>





Node type	Nodes	Cores/ node	Cores total	Memory/ node
Login node	4	16	64	64/192 G
Haswell compute node	397	24	9528	128 GB
Haswell big memory node	10	24	240	256 GB
Sandy Bridge compute node	496	16	7936	64 GB
Sandy Bridge big memory node	16	16	256	256 GB
Sandy Bridge huge memory node	2	32	64	1,5 TB
Haswell huge memory node	4	40	160	1,5 TB

Taito cluster



File systems and directories



Directory	Intended use	Default quota/user	Storage time	Backup
\$HOME	Initialization scripts, source codes, small data files. Not for running programs or research data.	50 GB	permanent	Yes
\$USERAPPL	Location for users' own application software installations	50 GB	permanent	Yes
\$TMPDIR	Run-time temporary files.		~ 2 days	No
\$WRKDIR	Temporary data files.	5 TB	Until further notice.	No
project	Common storage for project members. A project can consist of one or more user accounts.	On request	permanent	No
HPC-Archive	Long term storage	2,5 TB	permanent	Yes

<http://research.csc.fi/csc-guide-directories-and-data-storage-at-csc>

Data handling



Some brief generalizations:

- Directories containing tens of thousands of files are often problematic (use subdirectories and/or aggregation)
- It's usually faster to move one large file than many small ones
- On the other hand you should avoid too large files
 - it's nicer to re-send one 100 GB chunk than the whole 1 TB file
- Consider compression
- Prefer file formats that have checksums or other verification mechanisms
- Data should be packaged for saving in Archive server or IDA

More information

CSC Computing environment user guide:

- <https://research.csc.fi/csc-guide>

Many recorded webinars on this subject

- <https://research.csc.fi/csc-webinars>
 - Love your data! How can I share and publish my research data? (Jessica Parland-Von Essen, CSC)
 - Love your data! Where can I store my research data? (Kimmo Mattila, CSC)
 - Love your data! What tools are available for discovering research data? (Jessica Parland-Von Essen, CSC)
 - Love your data! What kind of support for research data management I get via CSC? (Ilkka Lappalainen, CSC)
 - Moving data between CSC and local environment (Ari-Matti Sarén, CSC)
 - IDA service: how to store and open your data (Anssi Kainulainen, CSC)
 - Using IDA in high performance computing (Kimmo Mattila)

Module system on Taito



Module system



- Different software packages have different, possibly conflicting, requirements.
- **LMOD module system** is used to manage software and programming environments
- **module load biokit** sets up most of the bioinformatics tools (but not all the tools)
 - See software web pages for details

Most commonly used module commands:



<code>module help</code>	Shows available options
<code>module load <i>modulename</i></code>	Loads the given environment module
<code>module load <i>modulename/version</i></code>	
<code>module list</code>	List the loaded modules
<code>module avail</code>	List modules that are available to be loaded (<i>i.e.</i> compatible with your current environment)
<code>module spider</code>	List all existing modules
<code>module spider <i>name</i></code>	Searches the entire list of existing modules
<code>module swap <i>module1 module2</i></code>	Replaces a module with a another module and tries to re-load compatible versions of other loaded modules
<code>module unload <i>modulename</i></code>	Unloads the given environment module
<code>module purge</code>	Unloads all modules

Advanced module commands:



These can be helpful with your own software installations

`module save filename`

Saves current module set

`module restore filename`

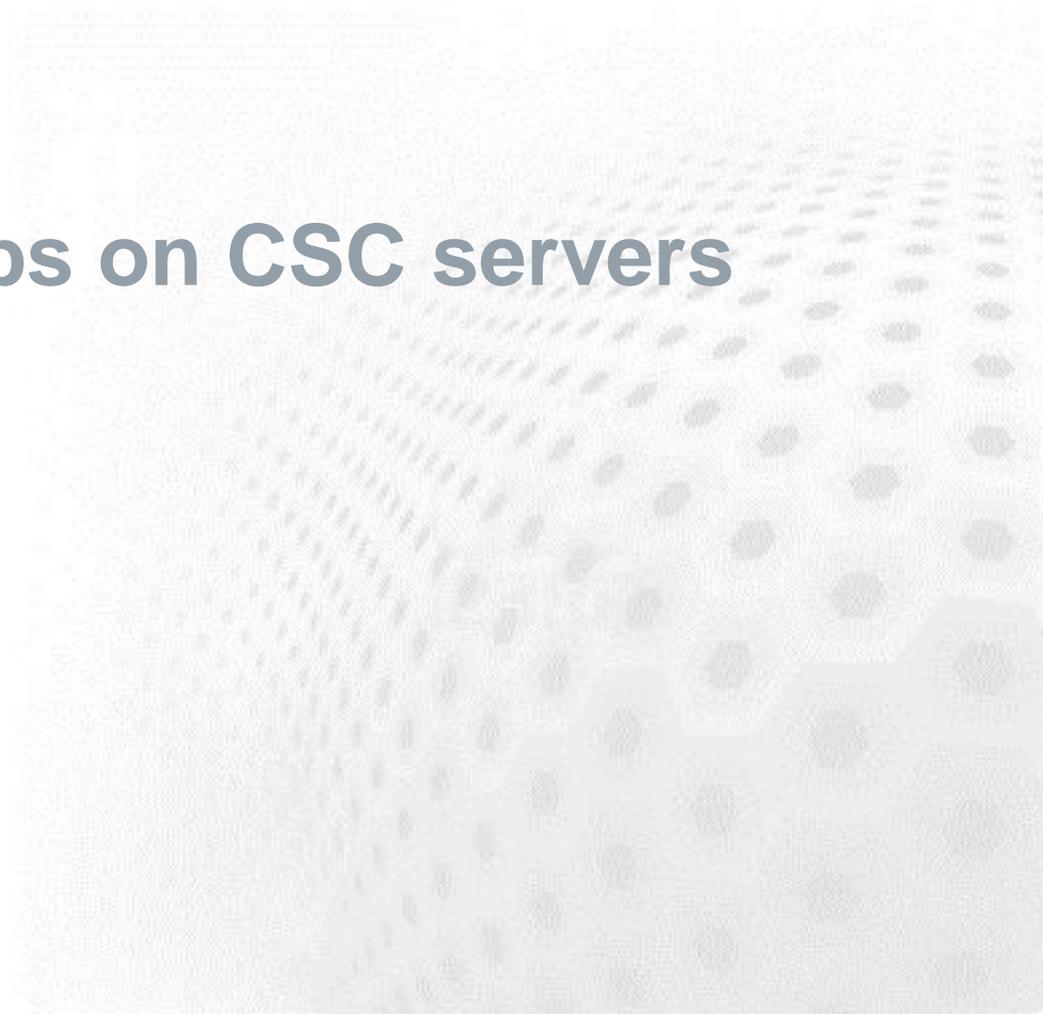
Loads saved module set

It is also possible to write your own modulefiles. For example, if you add the module files to **\$HOME/modulefiles**, you can access them after you add the path to the modules search path using command:

`module use $HOME/modulefiles`



Running jobs on CSC servers



Types of jobs



➤ Serial jobs

- Use only one core
- Many bioinformatics tools

➤ Embarrassingly parallel tasks:

- Job can be split to numerous sub jobs
- You can use array jobs and splitting utilizing tools like pb blast, cluster_interproscan, trinity, miso.

Types of jobs



➤ Threads/ OpenMP based parallelization

- Many bioinformatics tools use this approach. Bowtie2, BWA, Tophat...
- All the parallel processes must see the same memory -> all processes must run within one node



Types of jobs



➤ MPI parallelization.

- Each task has own memory -> job can utilize several nodes
- Check scaling before launching big jobs
- Using too many cores can actually make your job run slower



Parallel jobs

- Only applicable if your program supports parallel running
 - If the software documentation makes no mention of number of cores/processors/processes/threads to use, it's probably a serial code

- Check application documentation on number of cores to use
 - Speed-up is often not linear
 - Maximum effective number of cores can be limited by the algorithms and data
 - Using too many cores can actually make your job run slower

- Remember to set software parameters to match your batch job script
 - Some software will use one core only by default, even if more allocated
 - Some software will try to use all the cores in the node by default, even if less cores allocated

Interactive vs. Batch jobs

- Typical interactive jobs
 - Short jobs
 - Serial jobs (or small shared memory parallel jobs)
 - Software with GUI

- Typical batch jobs
 - Long jobs
 - Parallel jobs
 - Jobs that need specific resources (e.g. hugemem nodes, GPU nodes etc.)

Interactive jobs in Taito

- Only very small tasks should be done on the login nodes
- Any "real" jobs should be run on Taito-shell
 - Log in directly or use command `sinteractive` to start a taito-shell session in Taito
- Bigger interactive jobs can be run on Taito by reserving resources through the batch job system
 - Mainly necessary if you need specific resources (e.g. more memory or cores than in Taito-shell)
- <https://research.csc.fi/taito-interactive-batch-jobs>

Interactive jobs in Taito-shell

➤ Interactive jobs are best run on Taito-shell

- Login to: `taito-shell.csc.fi`
- Resources reserved automatically
 - Currently 4 cores/128 GB memory
- no time limit on jobs
- Note: `screen/nohup` will not work!
 - When you log out/disconnect all jobs will be killed
- <https://research.csc.fi/taito-shell-user-guide>

➤ If the job:

- Takes long
- Can be run in batch mode
- Can use more than one core

you should consider running it as a batch job

screen



➤ Using screen with Taito-shell

Open a connection to Taito:

```
ssh taito.csc.fi
```

Take note of the login node. Let's assume taito-login3.

Open a screen session using command:

```
screen -R
```

In the screen session, open a taito-shell session with command:

```
sinteractive
```

When you want to leave the session running in the background, detach from screen using `Ctrl-a d`.

Now you can logout from Taito, but your screen session in taito-login3 and the Taito-shell session within it is preserved.

To reattach to your session, connect first to the Taito login node where you have your screen session running. For example:

```
ssh taito-login3.csc.fi
```

Then, reattach the screen session with command

```
screen -R
```

Interactive jobs in Taito

- Example using `srun`

```
srun -n 1 -mem=256000 -t02:00:00 --x11=first --pty $SHELL
module load myprog
myprog
```

Here the option "`--x11=first`" sets up the x11 connection so that graphical user interfaces can be used, and option "`--pty $SHELL`" runs the default command shell

- Example using `salloc`

```
salloc -n 32 --ntasks-per-node=16 --mem-per-cpu=1000 -t00:30:00 -p parallel
srun mdrun_mpi -s topol1 -dlb yes
srun mdrun_mpi -s topol2 -dlb yes
exit
```

Batch jobs

➤ Steps for running a batch job

1. Write a batch job script

- Script format depends on server, check the user guides, e.g:
<http://research.csc.fi/taito-user-guide>
<http://research.csc.fi/sisu-user-guide>
- You can use the Batch Job Script Wizard in Scientist's User Interface:
<https://sui.csc.fi/group/sui/batch-job-script-wizard>

2. Make sure you have all the necessary input files where the program can find them

- Usually best to use \$WRKDIR
- \$HOME has limited space
- Login \$TMPDIR is not available in compute nodes

3. Submit your job

```
sbatch myscript
```

Batch jobs

- User has to specify necessary resources
 - Can be added to the batch job script or given as command line options for `sbatch` (or a combination of script and command line options)

- Resources need to be adequate for the job
 - Too small memory reservation will cause the job to use swap disk (very slow) or even fail
 - When the time reservation ends, the job will be terminated whether finished or not

- But: Requested resources can affect the time the job spends in the queue
 - Especially core number and memory reservation

- So: Realistic resource request give best results
 - Not always easy to know beforehand
 - Usually best to try with smaller tasks first and check the used resources

Batch Job Script wizard in Scientist's User Interface

taito ▼ Select application... ▼

Form

General

Job Name:

Shell: ▼

Email Address:

Output

Standard Output File Name:

Standard Error File Name:

Computing Resources

Computing Time:

Number of Cores:

Memory Size:

Script Commands

```
# example run commands
srun ./my_mpi_program
```

Reset



Script Result



```
#!/bin/bash -l
# created: Jan 15, 2015 12:55 PM
# author: saren
#SBATCH -J test1
#SBATCH -o test_out-%j
#SBATCH -e test_err-%j
#SBATCH -n 8
#SBATCH -t 02:00:00
#SBATCH --mem-per-cpu=4000
#SBATCH --mail-type=END
#SBATCH --mail-user=ari-matti.saren@csc.fi

# commands to manage the batch script
# submission command
# sbatch [script-file]
# status command
# squeue -u saren
# termination command
# scancel [jobid]

# For more information
# man sbatch
# more examples in Taito guide in
# http://research.csc.fi/taito-user-guide

# example run commands
srun ./my_mpi_program

# This script will print some usage statistics to the
# end of file: test_out-%j
# Use that to improve your resource request estimate
# on later jobs.
used_slurm_resources.bash
```

Save



Batch Job Script wizard in Scientist's User Interface

taito
Select application...

Form

- General**

Job Name:

Shell:

Email Address:
- Output**

Standard Output File Name:

Standard Error File Name:
- Computing Resources**

Computing Time:

Computing time must be in format: hh:mm:ss
 Supply computing time for a job in hh:mm:ss format. Accurate estimation for computing time will improve turnover time for the job

Number of Cores:

Memory Size:
- Script Commands**

example run commands

srun ./my_mpi_program

Script Result

```

#!/bin/bash -l
# created: Jan 15, 2015 12:55 PM
# author: saren
#SBATCH -J test1
#SBATCH -o test_out-%j
#SBATCH -e test_err-%j
#SBATCH -n 8
#SBATCH --mem-per-cpu=4000
#SBATCH --mail-type=END
#SBATCH --mail-user=ari-matti.saren@csc.fi

# commands to manage the batch script
# submission command
# sbatch [script-file]
# status command
# squeue -u saren
# termination command
# scancel [jobid]

# For more information
# man sbatch
# more examples in Taito guide in
# //research.csc.fi/taito-user-guide

# run commands
mpi_program

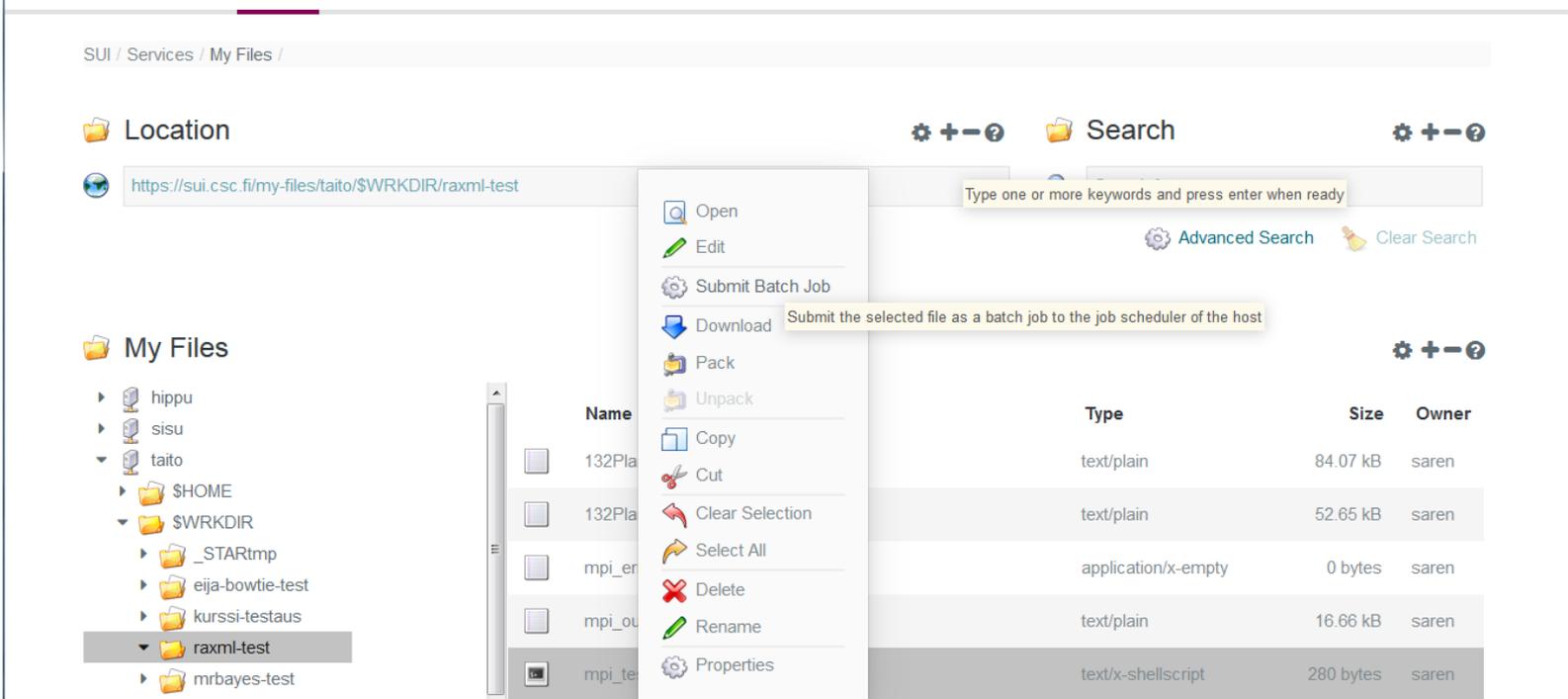
# This script will print some usage statistics to the
# end of file: test_out-%j
# Use that to improve your resource request estimate
# on later jobs.
used_slurm_resources.bash
          
```

Save

Reset

Submitting a Batch Job Scientist's User Interface

- Go to My Files
- Select a file
- From the pop-up menu select "Submit batch job"



The screenshot shows the CSC Scientist's User Interface. The breadcrumb path is "SUI / Services / My Files /". The URL bar shows "https://sui.csc.fi/my-files/taito/\$WRKDIR/raxml-test". The "My Files" section on the left shows a tree view with folders like "hippu", "sisu", "taito", "\$HOME", "\$WRKDIR", and "raxml-test" selected. A context menu is open over a file, with "Submit Batch Job" highlighted. A tooltip for "Submit Batch Job" reads: "Submit the selected file as a batch job to the job scheduler of the host".

Name	Type	Size	Owner
132Pla	text/plain	84.07 kB	saren
132Pla	text/plain	52.65 kB	saren
mpi_er	application/x-empty	0 bytes	saren
mpi_lo	text/plain	16.66 kB	saren
mpi_te	text/x-shellscript	280 bytes	saren

Example serial batch job script on Taito:

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6000
#SBATCH -p serial
#

module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK chr_18 reads.fq > out.sam
```

```
#!/bin/bash -l
```

- Tells the computer this is a script that should be run using bash shell
 - The `-l` option makes bash behave as it was a login shell. This makes sure module system behaves as expected
- Everything starting with "`#SBATCH`" is passed on to the batch job system
- Everything starting with "`#`" is considered a comment
- Everything else is executed as a command

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6000
#SBATCH -p serial
#

module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK
chr_18 reads.fq > out.sam
```

#SBATCH -J bowtie2

- Sets the name of the job
- Job names can be used to manage jobs, but unlike jobids they are not necessarily unique, so care should be taken

- E.g

```
scancel -n bowtie2
```

- When listing jobs e.g. with `squeue`, only 14 first characters of job name are displayed.

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6000
#SBATCH -p serial
#

module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK
chr_18 reads.fq > out.sam
```

```
#SBATCH -o output_%j.txt
```

```
#SBATCH -e errors_%j.txt
```

- Option `-o` sets the name of the file where the standard output (stdout) is written
- Option `-e` sets the name of the file where possible error messages (stderr) are written
- When running the program interactively these would be written to the command prompt
- What gets written to stdout and stderr depends on the program. If you are unfamiliar with the program, it's always safest to capture both
- `%j` is replaced with the job id number in the actual file name

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6000
#SBATCH -p serial
#
module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK
chr_18 reads.fq > out.sam
```

```
#SBATCH -t 02:00:00
```

- Time reserved for the job in hh:mm:ss
- When the time runs out the job will be terminated!
- With longer reservations the job might spend longer in the queue
- Limit for jobs is 3d (72h)
 - if you require longer time, you can specify "longrun" queue (limit 14d)
 - In the longrun queue your job size is limited to one node

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6000
#SBATCH -p serial
#
module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK
chr_18 reads.fq > out.sam
```

```
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
```

➤ In this case we are running a shared memory program. It must run inside one node, so we specify:

- 1 task (-n)
- 1 node (--nodes)
- 6 cores (--cpus-per-task)

➤ For a MPI program we would not need to run inside one node, so we might specify simply something like:

```
#SBATCH -n 36
```

➤ Check software documentation

- Many bioinformatics software can not utilize more than one core
- Some can use threads and run as a shared memory job
- Only very few utilize MPI

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6000
#SBATCH -p serial
#
module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK
chr_18 reads.fq > out.sam
```

#SBATCH --mem=6000

- The amount of memory reserved for the job in MB
 - 1000 MB = 1 GB
- For MPI jobs use `--mem-per-cpu`
- For shared memory (OpenMP) jobs `--mem` is easiest
 - `--mem-per-cpu` can be used, but remember to adjust if changing core number
- Keep in mind the specifications for the nodes. Jobs with impossible requests are rejected
- If you reserve too little memory the job will fail
- If you reserve too much memory your job will spend much longer in queue

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6000
#SBATCH -p serial
#
module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK
chr_18 reads.fq > out.sam
```

#SBATCH -p serial

- The queue (partition) the job should be submitted to
- You can check the available queues with command
`sinfo -l`
- Available queues in Taito:

Queue	Max cores	Max time	Max memory
serial (default)	16/24 (1 node)	3d	256 GB
parallel	448/672 (28 nodes)	3d	256 GB
longrun	16/24 (one node)	14d	256 GB
hugemem	32/40 (one node)	7d	1,5 TB
test	32/48 (2 nodes)	30 min	64 GB

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=1000
SBATCH -p serial
#

module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK
chr_18 reads.fq > out.sam
```

Choosing processor architecture

For Haswell:

```
#SBATCH --constraint=hsw
```

For Sandy Bridge

```
#SBATCH --constraint=snd
```

- Necessary if code has been compiled with processor-specific optimizations
- Often not necessary
- Check software web pages

```
module load biokit
```

```
bowtie2-build chr_18.fa chr_18
```

```
bowtie2-align -p $SLURM_CPUS_PER_TASK chr_18 reads.fq > out.sam
```

- Remember to load modules if necessary
- By default the working directory is the directory where you submitted the job
 - If you include a `cd` command, make sure it points to correct directory
- Command syntax depends on the software
 - It's not enough to reserve the cores: Also remember to tell the program to use them!
 - See application documentation for correct syntax
 - You can use system variable `$SLURM_CPUS_PER_TASK`
- MPI programs must be run through `srun`. Depending on the software you may need to specify some additional options
 - See application documentation for each software
 - For example:

```
srun raxmlHPC-MPI -N 100 -n test1 -s cox1.phy -m GTRGAMMAI
```

Most commonly used sbatch options

Slurm option

`--begin=time`

`-c, --cpus-per-task=ncpus`

`-d, --dependency=type:jobid`

`-e, --error=err`

`--ntasks-per-node=n`

`-J, --job-name=jobname`

`--mail-type=type`

`--mail-user=user`

`-n, --ntasks=ntasks`

`-N, --nodes=N`

`-o, --output=out`

`-t, --time=minutes`

`--mem-per-cpu=MB`

`-p`

Description

defer job until HH:MM MM/DD/YY

number of cpus required per task

defer job until condition on jobid is satisfied

file for batch script's standard error

number of tasks per node

name of job

notify on state change: BEGIN, END, FAIL or ALL

who to send email notification for job state changes

number of tasks to run

number of nodes on which to run

file for batch script's standard output

time limit in format hh:mm:ss

maximum amount of real memory per allocated cpu required by the job in megabytes

Specify queue (partition) to be used. In Taito the available queues are: serial, parallel, hsw_par, longrun, test and hugemem.

Array jobs

- Best suited for running the same analysis for large number of files
- Defined by adding `--array` option to batch job script
 - Can be defined as a range or a list. For ranges step size can be defined

```
#SBATCH --array=1-50
#SBATCH --array=1,2,10
#SBATCH --array=1-100:20
```
- When run, variable **`$SLURM_ARRAY_TASK_ID`** will be replaced with the current array job index
 - Note that the range of the **`$SLURM_ARRAY_TASK_ID`** variable is limited between 0 and 896
- Note that the batch job script is executed for each iteration, so things that should be done only once should not be included in the script

Simple array job example

```
#!/bin/bash
#SBATCH -J array_job
#SBATCH -o array_job_out_%A_%a.txt
#SBATCH -e array_job_err_%A_%a.txt
#SBATCH -t 02:00:00
#SBATCH --mem=4000
#SBATCH --array=1-50
#SBATCH -n 1

# run the analysis command
my_prog data_${SLURM_ARRAY_TASK_ID}.inp data_${SLURM_ARRAY_TASK_ID}.out
```

In this example the actual command run at each iteration will be:

```
myprog data_1.inp data_1.out
myprog data_2.inp data_2.out
..
myprog data_50.inp data_50.out
```

Using a list of file names in an array job



- Often it is easiest to use a list of input filenames
- You can use a combination of `sed` and the **`$SLURM_ARRAY_TASK_ID`** variable
 - To create a list of filenames

```
ls data_*.inp > namelist
```
 - To print a single row in a file by row number:

```
sed -n "row_number"p inputfile
```
 - Example commands in batch job script

```
name=$(sed -n ${SLURM_ARRAY_TASK_ID}p namelist)
my_prog ${name} ${name}.out
```

Example batch job script using a list of file names in an array job

```
#!/bin/bash -l
#SBATCH -J array_job
#SBATCH -o array_job_out_%j.txt
#SBATCH -e array_job_err_%j.txt
#SBATCH -t 02:00:00
#SBATCH --mem=4000
#SBATCH --array=1-50
#SBATCH -n 1

# set input file to be processed
name=$(sed -n ${SLURM_ARRAY_TASK_ID}p namelist)
# run the analysis command
my_prog $name $name.out
```

Array job using sbatch_commandlist

- `sbatch_commandlist` executes given command list as independent sub-tasks of an array job.
- Command launches the array job and monitors it until all the sub jobs have finished.
- Syntax:

```
sbatch_commandlist -commands list_of_independent_command_lines_to_execute
```

Optional options:

```
-t          SLURM run time reservation. Default: 12:00:00  
-mem       SLURM memory reservation. Default 8000  
-jobname   SLURM job name. Default: array_job  
-threads   SLURM num-cpus. Default: 1
```

SLURM: Managing batch jobs in Taito

Submitting and cancelling jobs

- The script file is submitted with command

```
sbatch batch_job.file
```

- sbatch options are usually listed in the batch job script, but they can also be specified on command line, e.g.

```
sbatch -J test2 -t 00:05:00 batch_job_file.sh
```

- Job can be deleted with command

```
scancel <jobid>
```

Queues

- The job can be followed with command `squeue`:

<code>squeue</code>	(shows all jobs in all queues)
<code>squeue -p <partition></code>	(shows all jobs in single queue (partition))
<code>squeue -u <username></code>	(shows all jobs for a single user)
<code>squeue -j <jobid></code>	(shows status of a single job)

- To estimate the start time of a job in queue

```
scontrol show job <jobid>
```

row "StartTime=..." gives an estimate on the job start-up time, e.g.

```
StartTime=2018-02-12T19:46:44 EndTime=Unknown
```

Available queues

- You can check available queues on each machine with command:

```
sinfo -l
```

```
PARTITION AVAIL  TIMELIMIT  JOB_SIZE ROOT SHARE  GROUPS  NODES  STATE NODELIST
serial*   up 3-00:00:00      1  no YES:4    all    525  mixed c[5-497,500-505,508,510-516,518-528,570-576]
parallel  up 3-00:00:00     1-28  no  NO      all    525  mixed c[5-497,500-505,508,510-516,518-528,570-576]
longrun   up 14-00:00:00     1  no YES:4    all    525  mixed c[5-497,500-505,508,510-516,518-528,570-576]
test      up 30:00          1-2  no YES:4    all     4  idle  c[1-4]
hugemem   up 7-00:00:00     1  no YES:4    all     2  mixed c[577-578]
```

Available nodes

- You can check available nodes in each queue with command:
`sjstat -c`

Scheduling pool data:

```
-----
```

Pool	Memory	Cpus	Total	Usable	Free	Other Traits
serial*	258000Mb	16	12	12	7	bigmem,snb,sandybridge
serial*	128600Mb	24	395	395	0	hsw,haswell
serial*	64300Mb	16	450	449	82	snb,sandybridge
serial*	258000Mb	24	10	10	9	hsw,haswell
parallel	258000Mb	16	12	12	7	bigmem,snb,sandybridge
parallel	128600Mb	24	395	395	0	hsw,haswell
parallel	64300Mb	16	450	449	82	snb,sandybridge
parallel	258000Mb	24	10	10	9	hsw,haswell
longrun	258000Mb	16	8	8	6	bigmem,snb,sandybridge
longrun	128600Mb	24	395	395	0	hsw,haswell
longrun	64300Mb	16	450	449	82	snb,sandybridge
longrun	258000Mb	24	10	10	9	hsw,haswell
test	64300Mb	16	2	2	2	snb,sandybridge
test	128600Mb	24	2	2	2	hsw,haswell
hugemem	1551000Mb	32	2	2	0	bigmem,snb,sandybridge
hugemem	1551000Mb	40	4	4	1	bigmem,hsw,haswell,ssd

```
-----
```

Most frequently used SLURM commands



Command	Description
<code>srun</code>	Run a parallel job.
<code>salloc</code>	Allocate resources for interactive use .
<code>sbatch</code>	Submit a job script to a queue.
<code>scancel</code>	Cancel jobs or job steps.
<code>sinfo</code>	View information about SLURM nodes and partitions.
<code>squeue</code>	View information about jobs located in the SLURM scheduling queue
<code>smap</code>	Graphically view information about SLURM jobs, partitions, and set configurations parameters
<code>sjstat</code>	display statistics of jobs under control of SLURM (combines data from <code>sinfo</code> , <code>squeue</code> and <code>scontrol</code>)
<code>scontrol</code>	View SLURM configuration and state.
<code>sacct</code>	Displays accounting data for batch jobs.

Monitoring resource usage



seff

- Command `seff` will print a summary of requested and used resources for both running and finished batch jobs

```
seff <jobid>
```

seff

Example 1: Core utilization bad

```
>seff 123456
```

```
Job ID: 123456
```

```
Cluster: csc
```

```
User/Group: user/group
```

```
State: COMPLETED (exit code 0)
```

```
Nodes: 1
```

```
Cores per node: 8
```

```
CPU Utilized: 0:51:01
```

```
CPU Efficiency: 12.48% of 06:56:08 core-walltime
```

```
Memory Utilized: 5.98 GB
```

```
Memory Efficiency: 75.89% of 7.88 GB
```

```
Job consumed X.XX CSC billing units based on cpu reservation multiplier
```

seff

Example 2: Memory utilization bad

```
>seff 123456
```

```
Job ID: 123456
```

```
Cluster: csc
```

```
User/Group: user/group
```

```
State: COMPLETED (exit code 0)
```

```
Nodes: 1
```

```
Cores per node: 8
```

```
CPU Utilized: 05:49:01
```

```
CPU Efficiency: 83.89% of 06:56:08 core-walltime
```

```
Memory Utilized: 5.98 GB
```

```
Memory Efficiency: 6.25% of 92.59 GB
```

```
Job consumed X.XX CSC billing units based on cpu reservation multiplier
```

seff

Example 3: Job failed due to time reservation running out

```
>seff 123456
```

```
Job ID: 1234566
```

```
Cluster: csc
```

```
User/Group: user/csc
```

```
State: TIMEOUT (exit code 1)
```

```
Nodes: 1
```

```
Cores per node: 12
```

```
CPU Utilized: 02:06:41
```

```
CPU Efficiency: 70.30% of 03:00:12 core-walltime
```

```
Memory Utilized: 24.70 GB
```

```
Memory Efficiency: 72.27% of 34.18 GB
```

```
Job consumed 6.01 CSC billing units based on cpu reservation multiplie
```

sacct

- Command `sacct` can be used to study past jobs
 - Usefull when deciding proper resource requests

<code>sacct</code>	Short format listing of jobs starting from midnight today
<code>sacct -j <jobid></code>	information on single job
<code>sacct -S YYYY-MM-DD</code>	listing start date
<code>sacct -l</code>	long format output
<code>sacct -o</code>	list only named data fields, e.g.

```
sacct -o jobid,jobname,reqmem,maxrss,averss,state,elapsed -j <jobid>
```

➤ Some useful data fields:

jobid	Job id number
jobname	Job name
reqcpus	Cores requested from SLURM
reqmem	Memory requested from SLURM
maxrss	Maximum used memory
averss	Average used memory per process/core
state	Exit status of job
elapsed	Execution time

```
> sacct -o jobid,jobname,ntasks,reqnodes,allocnodes,reqcpus,alloccpus,reqmem,maxrss,averss,timelimit,elapsed,state -j 1731798 1
```

JobID	JobName	NTasks	ReqNodes	AllocNodes	ReqCPUS	AllocCPUS	ReqMem	MaxRSS	AveRSS	Timelimit	Elapsed	State
17317981	cretest		1	1	8	8	504Mc			00:01:00	00:00:01	COMPLETED
17317981.ba+	batch	1	1	1	8	8	504Mc	1580K	1580K		00:00:01	COMPLETED

top



- Monitor running processes:

```
top -u <username>
```

- `top` needs to run in the same node the process is running in
- You can log into the compute nodes where your job is running while your job is running
- Very usefull for monitoring threaded applications in real time
- Can be used to check MPI is distributing tasks correctly

top



➤ In taito-shell

- Check node name in with command

```
hostname
```

- Open another shell and log into the first node (if necessary)

```
ssh <nodename>
```

and start top

```
top -u <username>
```

- Start job in the first node

➤ With batch jobs

- Wait until job is running
- Check node name/names with

```
queue -l -u <username>
```

- log into the first node

```
ssh <node name>
```

- Start top

```
top -u <username>
```

top



➤ Example 1

- 1 process, 1 thread

```
top - 12:15:32 up 28 days, 1:06, 2 users, load average: 4.86, 4.35, 4.22
Tasks: 939 total, 5 running, 925 sleeping, 6 stopped, 3 zombie
Cpu(s): 26.4%us, 4.8%sy, 0.0%ni, 66.3%id, 2.5%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 264641088k total, 210050956k used, 54590132k free, 159736k buffers
Swap: 9775548k total, 9775052k used, 496k free, 87404532k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4993	user	20	0	5901m	3.8g	1060	R	99.7	1.5	0:31.67	STAR
2075	user	20	0	17200	2020	996	R	0.7	0.0	0:05.46	top
1478	user	20	0	103m	2048	904	S	0.0	0.0	0:00.12	sshd
1499	user	20	0	6112	600	504	S	0.0	0.0	0:00.03	slurm-spank-x11
1906	user	20	0	117m	12m	1548	S	0.0	0.0	0:00.55	bash
2019	user	20	0	153m	7064	2648	S	0.0	0.0	0:00.16	xterm

top



➤ Example 2

➤ 1 process, multiple threads

```
top - 12:15:32 up 28 days, 1:06, 2 users, load average: 4.86, 4.35, 4.22
Tasks: 939 total, 5 running, 925 sleeping, 6 stopped, 3 zombie
Cpu(s): 26.4%us, 4.8%sy, 0.0%ni, 66.3%id, 2.5%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 264641088k total, 210050956k used, 54590132k free, 159736k buffers
Swap: 9775548k total, 9775052k used, 496k free, 87404532k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4993	user	20	0	35.5g	19g	1140	R	195.1	7.7	5:03.09	STAR
2075	user	20	0	17200	2020	996	R	0.7	0.0	0:05.46	top
1478	user	20	0	103m	2048	904	S	0.0	0.0	0:00.12	sshd
1499	user	20	0	6112	600	504	S	0.0	0.0	0:00.03	slurm-spank-x11
1906	user	20	0	117m	12m	1548	S	0.0	0.0	0:00.55	bash
2019	user	20	0	153m	7064	2648	S	0.0	0.0	0:00.16	xterm

top



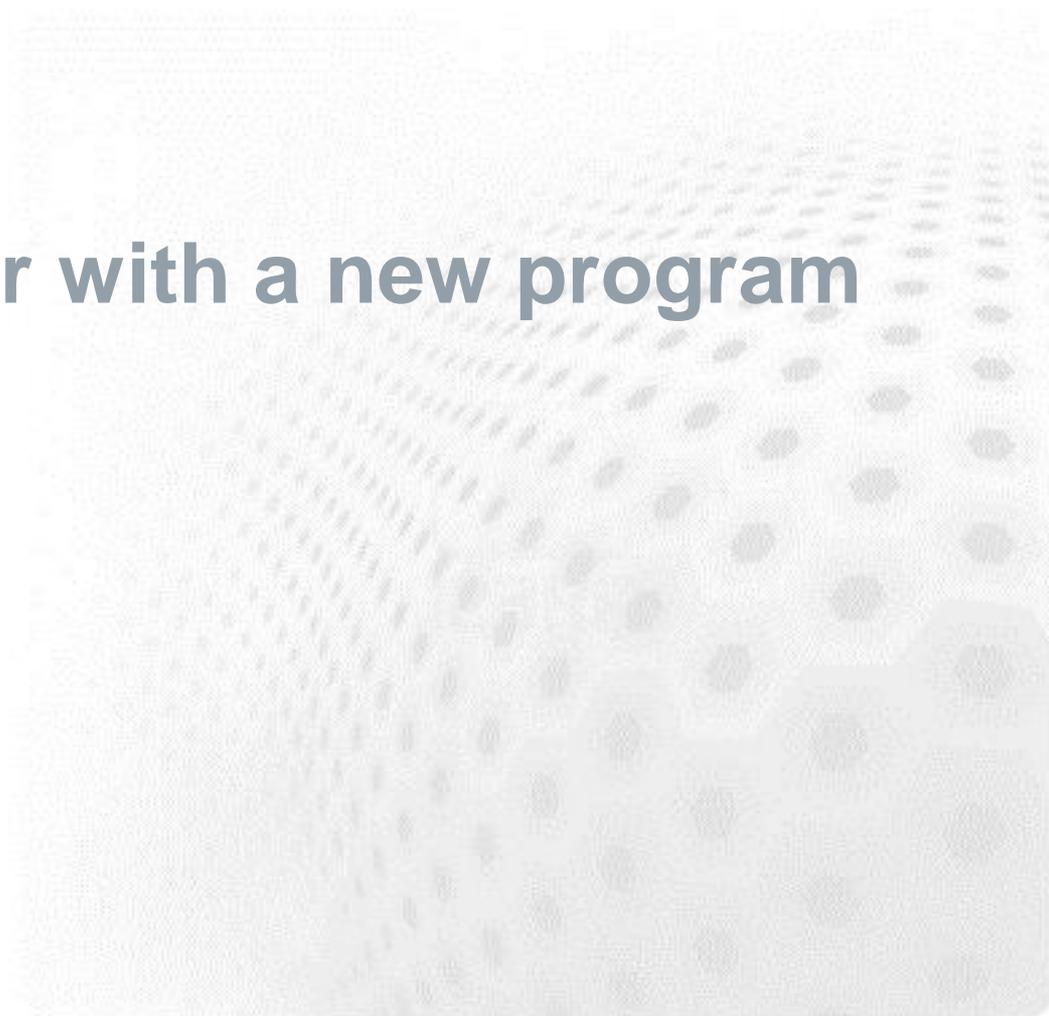
➤ Example

- multiple processes
- multiple tasks of a MPI job

```
top - 12:15:32 up 28 days, 1:06, 2 users, load average: 4.86, 4.35, 4.22
Tasks: 939 total, 5 running, 925 sleeping, 6 stopped, 3 zombie
Cpu(s): 26.4%us, 4.8%sy, 0.0%ni, 66.3%id, 2.5%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 264641088k total, 210050956k used, 54590132k free, 159736k buffers
Swap: 9775548k total, 9775052k used, 496k free, 87404532k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5224	user	20	0	5901m	3.8g	1140	R	89.1	1.5	4:08.11	test2
5223	user	20	0	5901m	3.8g	1140	R	92.0	1.5	4:08.11	test2
5222	user	20	0	5901m	3.8g	1140	R	88.3	1.5	4:08.11	test2
5221	user	20	0	5901m	3.8g	1140	R	89.2	1.5	4:08.11	test2
2075	user	20	0	17200	2020	996	R	0.7	0.0	0:05.46	top
1478	user	20	0	103m	2048	904	S	0.0	0.0	0:00.12	sshd
1499	user	20	0	6112	600	504	S	0.0	0.0	0:00.03	slurm-spank-x11
1906	user	20	0	117m	12m	1548	S	0.0	0.0	0:00.55	bash
2019	user	20	0	153m	7064	2648	S	0.0	0.0	0:00.16	xterm

Getting familiar with a new program



- Read the manual
- It may be helpful to first run the program interactively in *e.g.* `taito-shell` to find the correct command line options
 - Good chance to use `top` to get rough estimate on memory use etc
- If developers provide some test or example data, run it first
 - Make sure the results are as expected
- You can use **test** queue to check your batch job script
 - Limits : 30 min, 2 nodes
 - Job turnaround usually very fast
 - Can be useful to spot typos, missing files *etc* before submitting a job that will spend long in the queue
- Before very large runs, it's a good idea do a smaller trial run
 - Check that results are as expected
 - Check resource usage after test run and adjust accordingly
 - Try different core numbers and see how the software scales