

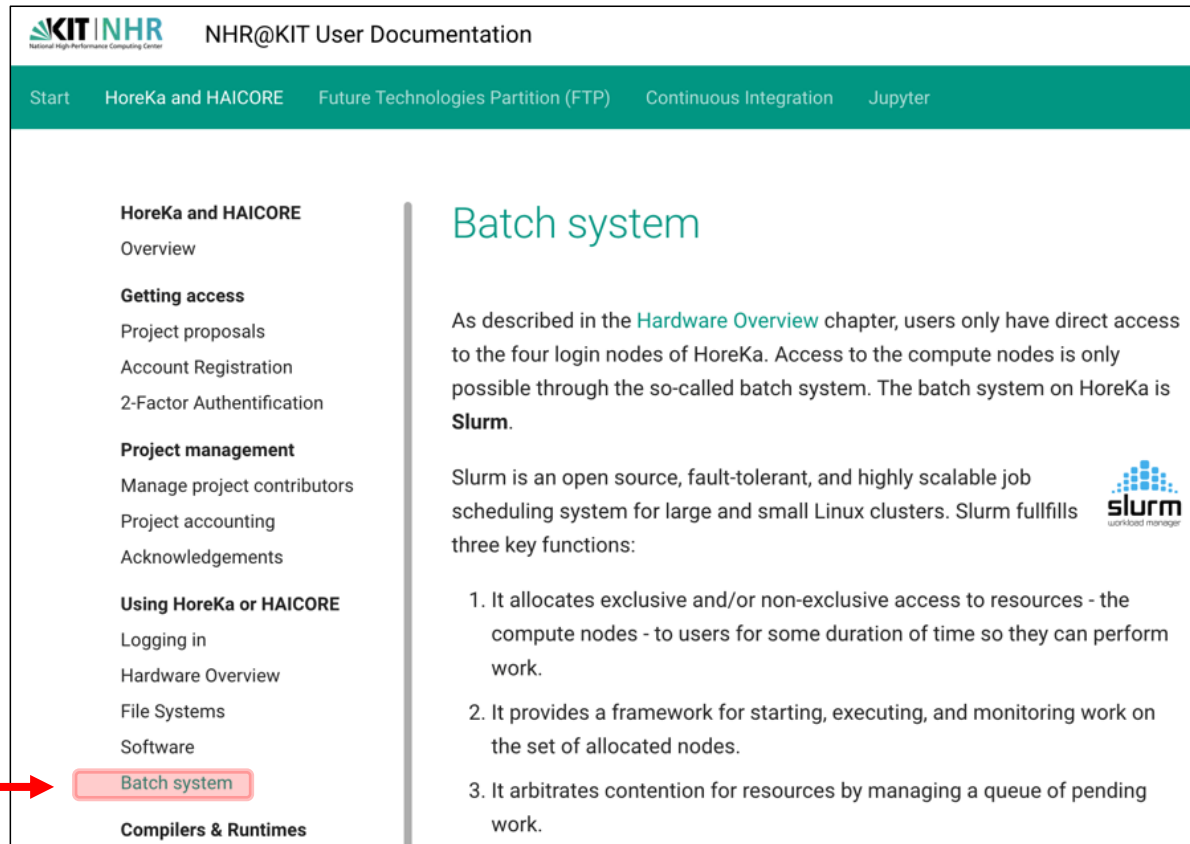
# Batch System Introduction

Hartmut Häfner, KIT - SCC



# Reference: NHR@KIT User Documentation

- All information given in this talk can be found at:  
<https://www.nhr.kit.edu/userdocs/horeka/batch/>



The screenshot shows the NHR@KIT User Documentation website. The header includes the KIT NHR logo and the title 'NHR@KIT User Documentation'. A navigation bar contains links for 'Start', 'HoreKa and HAICORE', 'Future Technologies Partition (FTP)', 'Continuous Integration', and 'Jupyter'. The main content area is split into a left sidebar and a right main section. The sidebar lists various topics, with 'Batch system' highlighted in a red box and pointed to by a red arrow. The main section is titled 'Batch system' and contains text explaining that users have direct access to four login nodes, with compute nodes accessed via a batch system called Slurm. It lists three key functions of Slurm: allocating resources, providing a framework for starting and monitoring work, and arbitrating contention for resources. The Slurm logo is also present in the bottom right of the main section.

**KIT NHR** National High-Performance Computing Center  
NHR@KIT User Documentation

Start HoreKa and HAICORE Future Technologies Partition (FTP) Continuous Integration Jupyter

**HoreKa and HAICORE**  
Overview

**Getting access**  
Project proposals  
Account Registration  
2-Factor Authentication

**Project management**  
Manage project contributors  
Project accounting  
Acknowledgements

**Using HoreKa or HAICORE**  
Logging in  
Hardware Overview  
File Systems  
Software

**Batch system**


**Compilers & Runtimes**

## Batch system

As described in the [Hardware Overview](#) chapter, users only have direct access to the four login nodes of HoreKa. Access to the compute nodes is only possible through the so-called batch system. The batch system on HoreKa is **Slurm**.

Slurm is an open source, fault-tolerant, and highly scalable job scheduling system for large and small Linux clusters. Slurm fulfills three key functions:

1. It allocates exclusive and/or non-exclusive access to resources - the compute nodes - to users for some duration of time so they can perform work.
2. It provides a framework for starting, executing, and monitoring work on the set of allocated nodes.
3. It arbitrates contention for resources by managing a queue of pending work.



# Material: Slides & Scripts

- [https://indico.scc.kit.edu/e/nhr-kit\\_course\\_2022-04-05](https://indico.scc.kit.edu/e/nhr-kit_course_2022-04-05)
- @horeka:
  - /software/all/workshop/2022-04-05

## How to read the following slides

Abbreviation/Colour code	Full meaning
<code>\$ command -option value</code>	<code>\$</code> = <b>prompt</b> of the interactive shell The full prompt may look like: <code>user@machine:path\$</code> The command has been entered in the interactive shell session
<code>&lt;integer&gt;</code> <code>&lt;string&gt;</code>	<code>&lt;&gt;</code> = Placeholder for integer, string etc

# Batch System

# Resource management

- Jobs are **NOT** executed by the user
- Instead, there is a management system (Batch System)
  - **workload manager (scheduler)**
    - scheduling, managing, monitoring, reporting
    - SLURM (HoreKa, bwUniCluster 2.0, 2x bwForCluster)
    - MOAB (2x bwForCluster)
  - **resource manager**
    - control over jobs and distributed compute nodes
    - SLURM (HoreKa, bwUniCluster 2.0, 2x bwForCluster)
    - TORQUE (2x bwForCluster)

# Resource and workload manager (1)

(1) User creates a **job script** and submits it to Slurm via the “**sbatch**” command

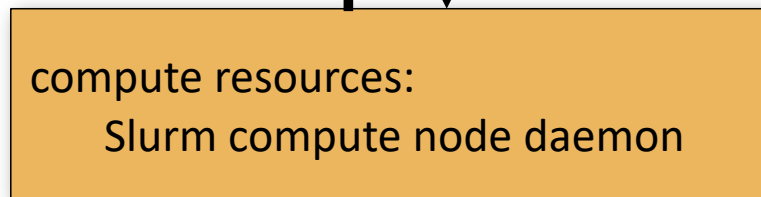
```
#!/bin/bash
#SBATCH -p dev_cpuonly
#SBATCH -N 1 -n 1
#SBATCH -t 00:01:00
#SBATCH -mem-per-cpu=50

./your_simulation
```

(2) Slurm parses the job script:  
→ where & when to run job



(3) Job execution:  
delegated to resource  
manager on the node



(4) The resource manager executes the job and communicates status information to nodes

# Resource and workload manager (2)

## ■ All clusters:

- compute job will **only** be processed by the batch system
- Running **jobs** on **login nodes** **not allowed**

## ■ Waiting time:

- depends on:
  - your job demands
  - your demand history
  - ONLY bwUniCluster 2.0: your university's share

# Job's life cycle

## 1. Preprocessing: Setup `job_script.sh`:

```
#!/bin/bash
#SBATCH -p dev_cpuonly
#SBATCH -N 1 -n 1
#SBATCH -t 00:01:00
#SBATCH -mem-per-cpu=50

./your_simulation
```

1) Options for the job

2) Actual work to be executed on the cluster

## 2. Submit: ONLY with “`sbatch`” (for interactive jobs: “`salloc`”)

```
$ sbatch job_script.sh
<job_ID>
```

## 3. Processing:

```
$ squeue
<job_ID> STATE: "PENDING" → "RUNNING" → "COMPLETED"
```

## 4. Postprocessing: Job is finished → check output (default job name)

```
bwUniCluster: slurm-<job_ID>.out
HoreKa       : slurm-<job_ID>.out
```



# 1. Job Submit: Sbatch options

- [https://wiki.bwhpc.de/e/BwUniCluster\\_2.0\\_Slurm\\_common\\_Features#sbatch\\_Command\\_Parameters](https://wiki.bwhpc.de/e/BwUniCluster_2.0_Slurm_common_Features#sbatch_Command_Parameters) on bwUniCluster 2.0 and
- <https://www.nhr.kit.edu/userdocs/horeka/batch/> on HoreKa respectively.
- sbatch options: command line or in your job script

Command line	Script	Purpose
<code>-N nodes</code>	<code>#SBATCH -nodes=nodes</code>	Number of nodes to be used.
<code>-n tasks</code>	<code>#SBATCH -ntasks=tasks</code>	Number of tasks to be launched.
<code>-J name</code>	<code>#SBATCH -job-name=name</code>	Gives a user specified name to the job.
<code>-p queue</code>	<code>#SBATCH -partition=queue</code>	Defines the queue class
<code>-mail-type=type</code>	<code>#SBATCH -mail-type=type</code>	Send email when job begins (BEGIN), ends (END), aborts (FAIL), at each event (ALL) ...

# 1. Job Submit: Important Resource Parameters

- [https://wiki.bwhpc.de/e/BwUniCluster\\_2.0\\_Slurm\\_common\\_Features#sbatch\\_Command\\_Parameters](https://wiki.bwhpc.de/e/BwUniCluster_2.0_Slurm_common_Features#sbatch_Command_Parameters) on bwUniCluster 2.0 and
- <https://www.nhr.kit.edu/userdocs/horeka/batch/> on HoreKa respectively.
- sbatch options: command line or in your job script

Sbatch Resource Options

Command line	Script	Purpose
<code>-t <i>time</i></code>	<code>#SBATCH -time=<i>time</i></code>	Wallclock time limit
<code>-N <i>nodes</i></code>	<code>#SBATCH -node=<i>nodes</i></code>	Number of nodes to be used
<code>-n <i>tasks</i></code>	<code>#SBATCH -ntasks=<i>tasks</i></code>	Number of tasks to be launched
<code>-c <i>count</i></code>	<code>#SBATCH -cpus-per-task=<i>count</i></code>	Number of CPUs required per (MPI-)task
<code>--ntasks-per-node=<i>count</i></code>	<code>#SBATCH -ntasks-per-node=<i>count</i></code>	Number of (MPI-)tasks per node
<code>--mem=<i>MB_value</i></code>	<code>#SBATCH -mem=<i>MB_value</i></code>	Memory in MegaByte per node
<code>--mem-per-cpu=<i>MB_value</i></code>	<code>#SBATCH -mem-per-cpu=<i>MB_value</i></code>	Memory required per allocated core

# HoreKa

	Thin nodes	Fat nodes	Accelerator nodes
Number of nodes	570	32	167
Sockets per node	2	2	2
Number of cores per node	76	76	76
Main memory per node	256 GB	512 GB	512 GB
Interconnect (InfiniBand)	HDR	HDR	HDR

- 4 login nodes
- Accelerator nodes incorporate 4 NVIDIA A100-40 GPUs

# sbatch -p queues (All queues of HoreKa)

<i>queue</i>	<i>default resources</i>	<i>MIN resources</i>	<i>MAX resources</i>
dev_cpuonly	time=10, ntasks=1, mem-per-cpu=1600mb	nodes=1, ntasks=1	time=60, nodes=2, mem=243200mb, ntasks=152
cpuonly	time=10, ntasks=152, mem-per-cpu=1600mb, mem=243200mb	nodes=1, ntasks=152	time=72:00:00, nodes=192, mem=501600mb, ntasks=152
dev_accelerated	time=10, ntasks=1, grep=gpu:1, mem-per-cpu=3300mb, def-mem-per-gpu=125400mb, def-cpu-per-gpu=38	nodes=1, ntasks=1, gres=gpu:1	time=1:00:00, nodes=1, mem=501600mb, ntasks=152, gres=gpu:4
accelerated	time=10, ntasks=152, grep=gpu:4, mem=501600mb, mem-per-cpu=3300mb, def-mem-per-gpu=125400mb, def-cpu-per-gpu=38	nodes=1, ntasks=152, gres=gpu:4	time=48:00:00, nodes=56, ntasks=152, gres=gpu:4, mem=501600mb

■ <https://www.nhr.kit.edu/userdocs/horeka/batch/>

■ Time and memory will be automatically chosen if not set

# 1. Job Submit:

## Sbatch Resource Options

Command line	Script	Purpose
<code>-t time</code>	<code>#SBATCH -time=time</code>	Wallclock time limit
<code>-N nodes</code>	<code>#SBATCH -node=nodes</code>	Number of nodes to be used
<code>-n tasks</code>	<code>#SBATCH -ntasks=tasks</code>	Number of tasks to be launched
<code>-c count</code>	<code>#SBATCH -cpus-per-task=count</code>	Number of CPUs required per (MPI-)task
<code>--ntasks-per-node=count</code>	<code>#SBATCH -ntasks-per-node=count</code>	Number of (MPI-)tasks per node
<code>--mem=MB_value</code>	<code>#SBATCH -mem=MB_value</code>	Memory in MegaByte per node
<code>--mem-per-cpu=MB_value</code>	<code>#SBATCH -mem-per-cpu=MB_value</code>	Memory required per allocated core

Use these options in the job script:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 20
#SBATCH --time=00:02:00
#SBATCH --mem=500mb
...
```

Or use them with sbatch:

```
$ sbatch -N 1 -n 20 -t 2 --mem=500 <job_script>
```

# Available Resources

## ■ Check available resources via

```
$ sinfo -t idle
```

```
[ab1234@hkn1993 bwKURS-intro]$ sinfo -t idle  
Partition dev_cpuonly :      1 nodes idle  
Partition cpuonly      :    101 nodes idle  
Partition dev_accelerate:    1 nodes idle  
Partition accelerated  :    83 nodes idle  
Partition haicore-gpu4 :     3 nodes idle  
Partition haicore-gpu8 :     0 nodes idle
```

# Tutorial 1a – HoreKa

- **Goal:** Use the Batch System to execute “`printenv`” on the cluster
- **1)** Create a file „`submit_script.sh`“ and set the following options in the submit

script:

- 1 task
- 500 MB memory
- Time: 5 minutes

```
#!/bin/bash
#SBATCH [???]
#SBATCH --time=00:05:00
#SBATCH --mem=500

[?????]
```

- **2)** After defining these options, insert the command to be executed at the end of the jobscript (“`printenv`”)
- **3)** Save the jobscript and submit it to the Batch System with

```
$ sbatch -p cpuonly submit_script.sh
```

- You can use “`squeue`” to see the status of your job
- **4)** Look in the output file of your job (`slurm-<jobID>.out`) for variables starting with „**SLURM\_**“. These can be used to get information on how the job was started

# Tutorial 1a – Solution

- Create a file named „**submit\_script.sh**“ with the following content:

```
#!/bin/bash
#SBATCH -n 1
#SBATCH --time=00:05:00
#SBATCH --mem=500

printenv
```

- Save the file and submit it with

```
$ sbatch -p cpuonly submit_script.sh
```

- In the output file, you can find the SLURM variables:

- For example: „**SLURM\_JOB\_PARTITION=cpuonly**“ means:

- In the job script, we have not defined a partition but the job was submitted to the „**cpuonly**“ partition with sbatch command



# Tutorial 1b

- 1) Modify your submit script so that instead of “printenv” the value of „**SLURM\_NPROCS**“ is printed (Hint: Use **echo**)
- Submit your job again, but this time use sbatch to specify the number of processes:

```
$ sbatch -p cpuonly -n 4 submit_script.sh
```

- 2) Check in your output file if the number of processes is „1“ as specified in the submit script or „4“ as specified directly with **sbatch**

# Tutorial 1b - Solution

- Modify your submit script to print the variable **SLURM\_NPROCS**

```
#!/bin/bash
#SBATCH -n 4
#SBATCH --time=00:01:00
#SBATCH --mem=500

echo $SLURM_NPROCS
```

- Save the file and submit it with

```
$ sbatch -p cpuonly -n 4 submit_script.sh
```

- In the output file the number of processes is printed:

```
slurm-<job-ID>.out
```

```
4
```

**The options given directly to sbatch take precedence over the options in the submit script**

# Job life cycle: 3. Processing (1)

- After submit command: if successful it returns <job-ID>

```
$ sbatch submit_script.sh
```

```
Submitted batch job 1487560
```

- „Monitoring“ via:

- 3.a.

```
scontrol show job <job-ID>
```

- 3.b.

```
queue
```

- 3.c. Login to compute node:

```
srun --jobid=<id> --pty /usr/bin/bash
```

- „Modifying“ via:

- 4. Cancel your job

```
scancel <job-ID>
```

## 3.a. Processing – scontrol show job (1)

- After submission of your script `submit_script.sh`

```
$ scontrol show job <job-ID>
```

```
JobId=1487569 JobName=submit_script.sh
  UserId=ab1234(27049) GroupId=hk-project-scs(501119)
  Priority=1 Nice=0 Account=hk-project-scs QOS=normal
JobState=RUNNING Reason=None Dependency=(null)
  Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
RunTime=00:00:19 TimeLimit=00:10:00 TimeMin=N/A
SubmitTime=2021-10-13T00:06:58 EligibleTime=2021-10-13T00:06:58
  AccrueTime=2021-10-13T00:06:59
StartTime=2021-10-13T00:06:59 EndTime=2021-10-13T00:16:59
  ...
  SuspendTime=None SecsPreSuspend=0 LastSchedEval=...
Partition=dev_cpuonly AllocNode:Sid=hkn1999:2170796
  ReqNodeList=(null) ExcNodeList=(null)
NodeList=hkn0301
BatchHost=hkn0301
  ...
```

1) Your project acronym

2) Your job state

3) Your time logging

4) Your selected partition

5) Your node list and Node on which job started

## 3.a. Processing – scontrol show job (2)

- After submission of your script `submit_script.sh`

```
$ scontrol show job <job-ID>
```

```
JobId=1487569 JobName=submit_script.sh
```

```
...
```

```
NumNodes=1 NumCPUs=2 NumTasks=1 CPUs/Task=1 ...
```

```
TRES=cpu=2,mem=3200M,node=1,billing=2
```

```
Socks/Node=* NtasksPerN:B:S:C=0:0:*:1 CoreSpec=*
```

```
MinCPUsNode=1 MinMemoryCPU=1600M MinTmpDiskNode=0
```

```
Features=(null) DelayBoot=00:00:00
```

```
OverSubscribe=OK Contiguous=0 Licenses=(null) ...
```

```
Command=/hkfs/home/hk-project-scs/ab1234/submit-script.sh
```

```
WorkDir=/hkfs/home/project/hk-project-scs/ab1234
```

```
StdErr=/hkfs/home/hk-project-scs/ab1234/slurm-1487569.out
```

```
StdIn=/dev/null
```

```
StdOut=/hkfs/home/hk-project-scs/ab1234/slurm-1487569.out
```

```
Power=
```

```
NtasksPerTRES:0
```

1) Your requested nodes & CPU cores

2) Your job memory

3) Actual node policy

4) Your submit directory & submit script

5) Your job standard output and error log file

## 3.b. Processing – queue

- After submission of your script `submit_script.sh`

```
$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
1487570	dev_cpuon	submit_s	ab1234	R	0:05	1	hkn0301

```
$ squeue --long
```

JOBID	PARTITION	NAME	USER	STATE	TIME	TIME_LIMI	NODES	NODELIST(REASON)
1487570	dev_cpuon	submit_s	ab1234	RUNNING	2:49	10:00	1	hkn0301

- Job states:

- PD = PENDING
- R = RUNNING
- CD = COMPLETED
- ...
- F = FAILED
- CA = CANCELLED



- When job is pending: **expected start time?**

```
$ squeue --start
```

JOBID	...	START_TIME	SCHEDNODES
1487570	...	2021-10-14T10:10:10	hkn0301

## 3.b. Processing – sacct

- After submission of your script `submit_script.sh`
  - Display accounting data of your job and job steps?

```
$ sbatch submit_script.sh
```

```
Submitted batch job 1487652
```

```
$ sacct -j 1487652
```

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
1487652	submit_sc+	dev_cpuon+	hk-projec+	2	RUNNING	0:0
1487652.bat+	batch		hk-projec+	2	RUNNING	0:0
1487652.ext+	extern		hk-projec+	2	RUNNING	0:0
1487652.0	hostname		hk-projec+	2	COMPLETED	0:0
1487652.1	bash		hk-projec+	2	RUNNING	0:0

## 3.c. Processing – Compute node login

- Once your job is running (state=**R**),
  - login to dededicated nodes is possible:

```
ab1234@hkn1990:~$ sbatch submit_script.sh
Submitted batch job 1487652

ab1234@hkn1990:~$ squeue
  JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
 1487652 dev_cpuon  submit_s  ab1234  R         2:42      1 hkn0301

ab1234@hkn1990:~$ srun --jobid=1487652 --pty /usr/bin/bash
ab1234@hkn0301:~$
```

- The command srun adds another step to your job,
  - Once main jobs finishes, this job step is cancelled automatically:

```
ab1234@hkn0301:~$
slurmstepd: error: *** STEP 1487652.2 ON hkn0301 CANCELLED AT 2021-10-13T10:35:52 ***
exit
srun: Job step aborted: Waiting up to 32 seconds for job step to finish.

ab1234@hkn1990:~$
```



## 3.d. Processing – Change status of your job

- Once your job is submitted (state=**PD** or **R**),
  - You can cancel your job

```
$ sbatch submit_script.sh
Submitted batch job 1487683

$ scancel 1487683
JOBID PARTITION      NAME      USER ST      TIME  NODES NODELIST(REASON)
1487683 dev_cpuon submit_s  ab1234 R      2:42     1 hkn0301
```

- Check with sacct :

```
$ sacct -j 1487683
```

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
1487683	submit_sc+	dev_cpuon+	hk-projec+	2	CANCELLED+	0:0
1487683.bat+	batch		hk-projec+	2	CANCELLED	
1487683.ext+	extern		hk-projec+	2	CANCELLED	

# Tutorial 2

- Modify your submit script so that it executes a command to wait for 600 seconds (**sleep 600**)
- Set **walltime** to **10 minutes** and give your job a name
- Submit your job script with **sbatch**
- Use **squeue** to check the status of your job
- Use **scontrol show job** to see from which directory you started your job
- Use **scancel <job-ID>** to cancel your job

# Tutorial 2 - Solution

## ■ Modify your submit script

```
#!/bin/bash
#SBATCH -N 1 -n 1
#SBATCH -t 00:10:00
#SBATCH --mem-per-cpu=50
#SBATCH -J myJobName

sleep 600
```

## ■ Save the file and submit it with

```
$ sbatch -p cpuonly submit_script.sh
```

## ■ Get your start directory of your jobs

```
$ scontrol show job 1487685 | grep WorkDir
WorkDir=/hkfs/home/project/hk-project-scs/ab1234/workshop
```

## ■ Use **scancel** <job-ID> to cancel your job

# Interactive jobs

- Jobs on login nodes are not permitted
- **Solution: interactive slurm jobs**
  - Access compute nodes and work on them interactively

```
ab1234@hkn1990$ salloc -p cpuonly -n 1 -t 10 --mem=2000
```

Job is waiting to start, Do Not interrupt the command

```
salloc: Granted job allocation 1487738  
salloc: Waiting for resource configuration  
salloc: Nodes hkn0301 are ready for job
```

Job running. You are now on a compute node

```
ab1234@hkn0301:~$ {Now you can work on the compute node}
```

```
salloc: Job 1487738 has exceeded its time limit and its allocation has been revoked.  
srun: Job step aborted: Waiting up to 32 seconds for job step to finish.  
slurmstepd: error: *** STEP 1487738.interactive ON hkn0301 CANCELLED AT 2021-10-13T13:02:41 DUE TO TIME LIMIT ***
```

Requested time for the interactive job ran out

```
exit  
srun: error: hkn0301: task 0: Exited with exit code 127
```

```
ab1234@hkn1990:~$
```

Back on the login node