

# Batch system – introduction

Andreas Baer  
KIT, SCC



# Reference: bwHPC Wiki

Most information given by this talk can be found at <https://wiki.bwhpc.de>

- select cluster
- then select Batch System

Page: Discussion

## BwUniCluster2.0

The **bwUniCluster 2.0** is the joint high-performance computer system of Baden-Württemberg's Universiti at the Steinbuch Centre for Computing (SCC) at Karlsruhe Institute of Technology (KIT). The bwUniCluster :

**The following issue is known:** Due to the hardware configuration, there is currently an already known schemes reported" when starting an MPI application and refers to the device "mix5\_2". This is an Etherne

**Next maintenance**

Due to regular maintenance work the HPC System bwUnicluster 2 will not be available from 21.05.2024 at 08:30 AM until 24.05.2024 at 15:00 AM

Please see the maintenance page for more information about planned upgrades and other changes

**Training & Support**

- Getting Started
- E-Learning Courses@
- Support
- FAQ
- Send Feedback about Wiki pages

**User Documentation**

- Access: Registration, Deregistration, Using Jupyter, Using Jupyter (German)
- Login
- Hardware and Architecture
  - File Systems and Workspaces
- Cluster Specific Software
  - Using Containers
- Batch System
  - Queues and interactive Jobs
- Operational Changes

**Cluster Funding**

- Please **acknowledge** bwUniCluster 2.0 in your publications.

# Reference: NHR@KIT User Documentation

Most information given by this talk can be found at <https://www.nhr.kit.edu/userdocs>

- select cluster
- then select Batch System

The screenshot shows the NHR@KIT User Documentation website. The URL in the browser is <https://www.nhr.kit.edu/userdocs/horeka/batch/>. The navigation bar includes links for 'Start', 'HoreKa', 'HAICORE', 'Future Technologies Partition (FTP)', and 'C'. The sidebar menu lists various sections: 'HoreKa Overview', 'Project management' (Project proposals, Manage project contributors, Project accounting, Acknowledgements), 'Using HoreKa or HAICORE' (Account Registration, 2-Factor Authentication, Interactive login, Hardware Overview, File Systems, Software, Batch system), and 'Compilers & Runtimes' (Compilers overview, GNU Compiler Collection (GCC), Intel Compilers). The 'Batch system' link in the sidebar is circled in red. The main content area shows the 'Batch system' section, which describes the hardware overview and lists three functions of Slurm: 1. It allocates exclusive and/or non-exclusive acc of time so they can perform work. 2. It provides a framework for starting, executing 3. It arbitrates contention for resources by mana. Below this, it mentions 'Any kind of calculation on the compute nodes of H-specification of the required run time, number of C some metadata is called a batch job. Batch jobs h: as the system decides to run them.' and 'HoreKa batch system partitions'.

# Material: Slides & Scripts

- [https://indico.scc.kit.edu/e/hpc\\_course\\_2024-10-16](https://indico.scc.kit.edu/e/hpc_course_2024-10-16)
- BwUniCluster 2.0: /opt/bwhpc/common/workshops/2024-10-16/
- HoreKa: /software/all/workshop/2024-10-16/

## How to read the following slides

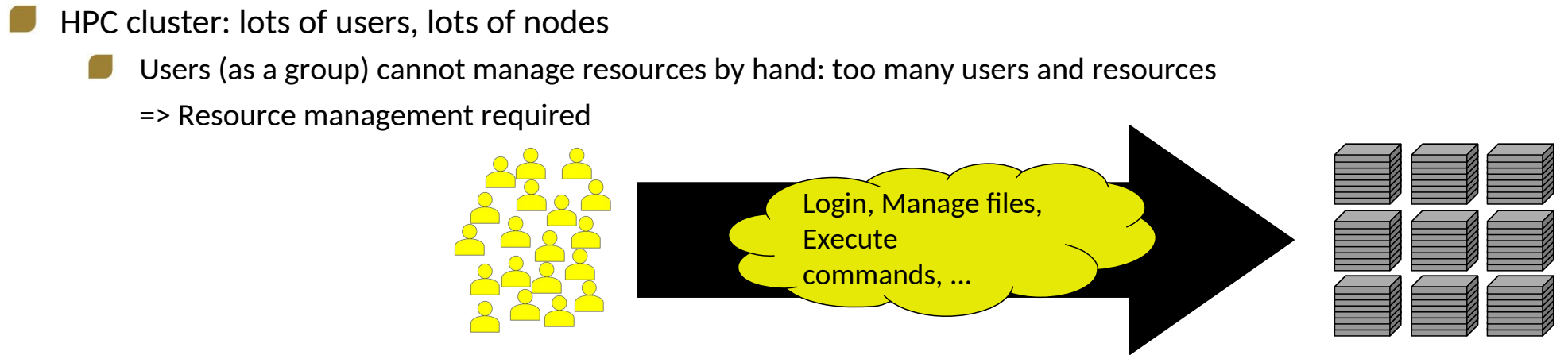
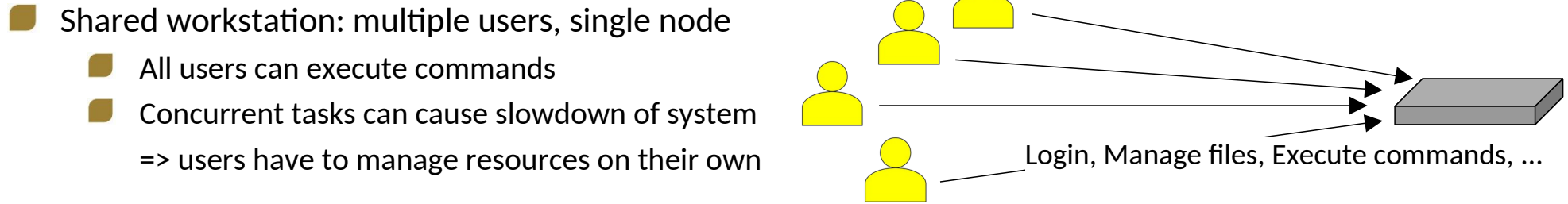
Abbreviation/Colour code	Full meaning
<code>\$ command -option value</code>	<code>\$</code> = <b>prompt</b> of the interactive shell The full prompt may look like: <code>user@machine:path\$</code> The <code>command</code> has been entered in the interactive shell session
<code>&lt;integer&gt;</code>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

# Outline

- Batch system – Why we need it and what it does.
- Job's life cycle
- 1./2. Preparation and Submission
- 3. Processing
- 4. Post processing
- Interactive jobs

# Batch System

# Resource management (1)

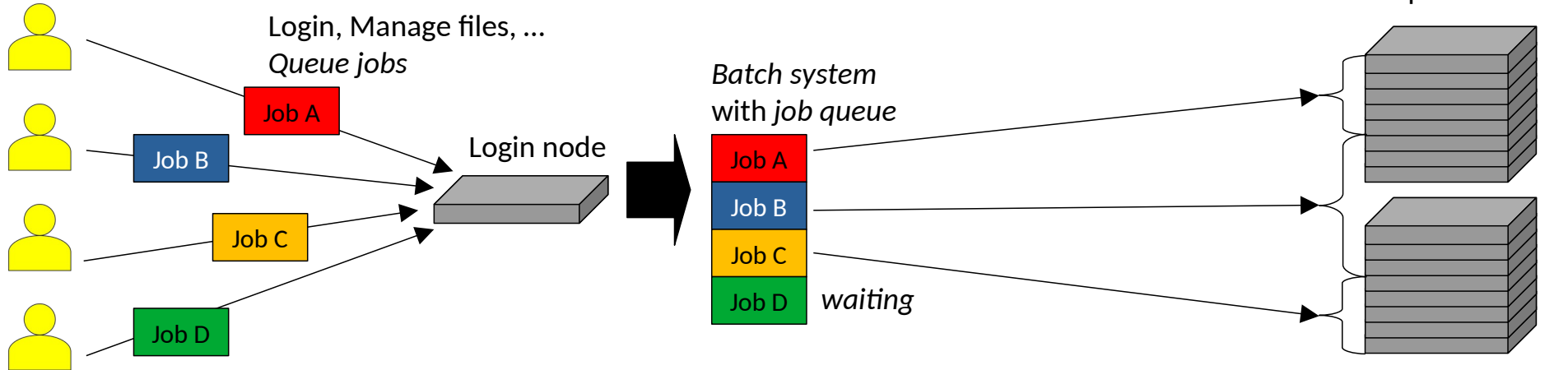


# Resource management (2)

- HPC cluster: lots of users, lots of nodes



- Include a login node and a management system (batch system)





## Resource management (3)

- User logs on to a designated **login node**, not a **compute node**
- Jobs are **not** executed by the user directly, but put into a **queue**

■ **Batch system** manages distribution of jobs to resources

■ Batch system consists of two parts

■ Workload manager (scheduler)

- Scheduling, managing, monitoring, reporting
- SLURM (HoreKa, bwUniCluster 2.0, JUSTUS 2, Helix)
- MOAB (NEMO, BinAC)

■ Resource manager

- Control over jobs and distributed compute nodes
- SLURM
- TORQUE (for systems with MOAB)

Waiting time for jobs depends on:

- Job resource demands
- Demand history
- ONLY bwUniCluster 2.0:  
share of university

All examples here are for SLURM, MOAB works similarly but with different commands.

# Job's life cycle

# Job's life cycle (1)

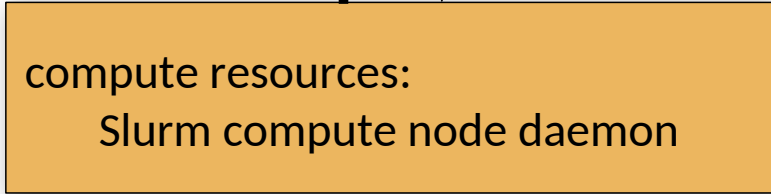
(1) User creates a **job script** and submits it to Slurm via the “**sbatch**” command

```
#!/bin/bash
#SBATCH -p dev_cpuonly
#SBATCH -N 1 -n 1
#SBATCH -t 00:01:00
#SBATCH --mem-per-cpu=500

./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

## Job's life cycle (2)

- 1. Preprocessing: setup `job_script.sh`:

```
#!/bin/bash
#SBATCH -p dev_single
#SBATCH -N 1 -n 1
#SBATCH -t 00:01:00
#SBATCH -mem-per-cpu=500

./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. Post processing: job is finished → check output  
Default: `slurm-<job_ID>.out`

# 1./2. Job submission

## 1./2. Job submission: important resource parameters

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 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 MB) per node
<code>--mem-per-cpu=<i>MB_value</i></code>	<code>#SBATCH --mem-per-cpu=<i>MB_value</i></code>	Memory per allocated core
<code>-p <i>queue</i></code>	<code>#SBATCH --partition=<i>queue</i></code>	Queue class to be used

- List of most used parameters can be found in the documentation
- Long and short options can be mixed arbitrarily but recommended as above

## 1./2. Job submission: partitions / queues

- A partition defines a specific **queue**  
=> submitted jobs will only wait for jobs in the same queue
- Partitions are used for different
  - Types of hardware (e.g. nodes with/without GPUs)
  - Purposes (e.g. development, production)
- Let's take a look at the hardware!

## 1./2. Job submission: hardware of bwUniCluster 2.0

	HPC nodes (Thin / HPC / IceLake)	Fat nodes	GPU nodes	IceLake + GPUx4
Number of nodes	200+60 / 360 / 272	6	14 (4 GPUs each) 10 (8 GPUs each)	15 (4 GPUs each)
Sockets per node	2 / 2 / 2	4	2	2
Cores per node	40 / 40 / 64	80	40	64
Main memory per node	96 GB, 192 GB / 96 GB / 256 GB	3 TB	384 GB 768 GB	512 GB
Local SSD	960 GB SATA / 1.8 TB NVMe	4.8 TB NVMe	3.2 TB NVMe 6.4 TB NVMe	6.4 TB NVMe
Interconnect (InfiniBand)	HDR 100 (blocking) / HDR 100 / HDR 200	HDR	HDR	HDR
GPUs	-	-	NVIDIA Tesla V100	NVIDIA A100 NVIDIA H100



## 1./2. Job submission: partitions of bwUniCluster 2.0 (selection)

Partition	Default resources	Min. resources	Max. resources
dev_single	time=10, mem-per-cpu=1125MB		time=00:30:00, nodes=1, mem=180000MB, ntasks-per-node=40
single	time=30, mem-per-cpu=1125MB		time=72:00:00, nodes=1, mem=180000MB, ntasks-per-node=40
multiple	time=30, mem-per-cpu=1125MB	nodes=2	time=72:00:00, nodes=128, mem=180000MB, ntasks-per-node=40
gpu_4	time=10, mem-per-cpu=2178MB, cpu-per-gpu=20		time=48:00:00, nodes=14, mem=376000MB, ntasks-per-node=40
fat	time=10, mem-per-cpu=18750MB		time=72:00:00, nodes=1, ntasks-per-node=80

■ Full list available at [https://wiki.bwhpc.de/e/BwUniCluster2.0/Batch\\_Queues](https://wiki.bwhpc.de/e/BwUniCluster2.0/Batch_Queues)

## 1./2. Job submission: hardware of Horeka

	CPU only nodes	Extra-large nodes	GPU nodes
Number of nodes	570 + 32	8	167
Sockets per node	2	2	2
Cores per node	76	76	76
Main memory per node	256/512 GB	4096 GB	512 GB
Local SSD	960 GB NVMe	7x3.84 TB NVMe	960 GB NVMe
Interconnect (InfiniBand)	HDR	HDR	HDR
GPUs	-	-	NVIDIA A40 / A100

## 1./2. Job submission: partitions of HoreKa

Partition	Default resources	Min. resources	Max. resources
dev_cpuonly	time=10, ntasks=1, mem-per-cpu=1600MB	nodes=1, ntasks=152	time=04:00:00, nodes=12, mem=243200MB, ntasks-per-node=152
cpuonly	time=10, ntasks=152, mem-per-cpu=1600MB, mem=243200MB	nodes=1, ntasks=152	time=3-00:00:00, nodes=192, mem=501600MB, ntasks-per-node=152
dev_accelerated	time=30, ntasks=1 cpu-per-gpu=38, gres=gpu:1	nodes=1, ntasks=152, gres=gpu:1	time=01:00:00, nodes=1, mem=501600MB, ntasks-per-node=152, gres=gpu:4
accelerated	time=30, ntasks=152, mem=501600MB, cpu-per-gpu=38, gres=gpu:4	nodes=1, ntasks=152, gres=gpu:4	time=2-00:00:00, nodes=128, mem=501600MB, ntasks-per-node=152, gres=gpu:4
large	time=10, ntasks=1, mem-per-cpu=27130MB	nodes=1, ntasks=1	time=2-00:00:00, nodes=8, mem=4123930MB, ntasks-per-node=152

## 1./2. Job submission: available resources

- Check available resources via `$ sinfo -t idle`
- Be careful: a node planned for another job is counted “idle” but will not start a job

```
xy_ab1234@bwunicluster:~$ sinfo -t idle

Partition dev_single      :      6 nodes idle
Partition single         :      0 nodes idle
Partition dev_multiple   :      8 nodes idle
Partition multiple       :      3 nodes idle
Partition fat            :      0 nodes idle
Partition dev_multiple_e :      8 nodes idle
Partition multiple_e     :      3 nodes idle
Partition dev_special    :      2 nodes idle
Partition special        :      0 nodes idle
Partition gpu_4          :      0 nodes idle
Partition dev_gpu_4      :      1 nodes idle
Partition gpu_8          :      0 nodes idle
```

```
xy_ab1234@horeka:~$ sinfo -t idle

Partition dev_cpuonly    :      1 nodes idle
Partition cpuonly       :    101 nodes idle
Partition dev_accelerate :      1 nodes idle
Partition accelerate     :     83 nodes idle
```

# Tutorial 1a

- **Goal:** use the batch system to execute **printenv** on the cluster
- 1. Create a file “**submit\_script.sh**” and set the following options for the batch system
  - 1 task
  - 500 MB memory
  - Wall time: 5 minutes
- 2. Insert the command to be executed at the end of the jobscript
- 3. Save the jobscript and submit it to the batch system with

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

[?????]
```

```
$ sbatch -p single --reservation=ws submit_script.sh # bwunicluster
```

```
$ sbatch -p cpuonly --reservation=ws submit_script.sh # horeka
```

- You can use **squeue** to see the status of the 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.

(Errors can occur when copying commands from the pdf, as not all dashes “-” are dashes in the pdf.)

# Tutorial 1a - Solution

- **Goal:** use the batch system to execute `printenv` on the cluster
- 1. Create a file “`submit_script.sh`” and set the following options for the batch system
  - 1 task
  - 500 MB memory
  - Wall time: 5 minutes
- 2. Insert the command to be executed at the end of the jobscript
- 3. Save the jobscript and submit it to the batch system with

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

printenv
```

```
$ sbatch -p single --reservation=ws submit_script.sh # bwunicluster
```

```
$ sbatch -p cpuonly --reservation=ws submit_script.sh # horeka
```

- You can use `squeue` to see the status of the 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.
  - Example: “`SLURM_JOB_PARTITION=cpuonly`” means: the job was submitted to the partition “`cpuonly`”. We specified this on the command line but not on the script.

# Tutorial 1b

- **Goal:** learn about option precedence.
- 1. Modify your script so that instead of executing `printenv`, the value of “`SLURM_NPROCS`” is printed (Hint: use `echo`)
- 2. Submit your job again, but this time use `sbatch` to specify the number of processes:

```
$ sbatch -p single --reservation=ws -n 4 submit_script.sh # bwunicluster
```

```
$ sbatch -p cpuonly --reservation=ws -n 4 submit_script.sh #  
horeka
```

- 3. Check in your output file what the number of processes used is:
  - “1” as specified in the script
  - “4” as specified on the command line

# Tutorial 1b - Solution

- **Goal:** learn about option precedence.
- 1. Modify your script so that instead of executing `printenv`, the value of “`SLURM_NPROCS`” is printed (Hint: use `echo`)
- 2. Submit your job again, but this time use `sbatch` to specify the number of processes:

```
$ sbatch -p single --reservation=ws -n 4 submit_script.sh # bwunicluster
```

```
$ sbatch -p cpuonly --reservation=ws -n 4 submit_script.sh #  
horeka
```

- 3. Check in your output file what the number of processes used is:
  - “1” as specified in the script
  - “4” as specified on the command line=> The output file contains:

```
4
```

```
#!/bin/bash  
#SBATCH --ntasks=1  
#SBATCH --time=00:05:00  
#SBATCH --mem=500  
  
echo "$SLURM_NPROCS"
```

=> The options given on the command line take precedence over the options provided in the script.



## 3. Processing

### 3. Processing

- After job submission: job-ID is printed if successful
- “Monitoring” via:

```
$ sbatch submit_script.sh  
Submitted batch job 1487560
```

- 3.a Simple information on job status

```
squeue
```

```
sacct
```

- 3.b Extensive details on the job

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

- 3.c Login onto the compute node

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

- “Modifying” via:

- 3.d Cancel the job

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

## 3.a Processing - squeue

- Check status of a job after submission

```
$ 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
- ...



- While job is pending: what is the **expected start time**?

```
$ squeue --start
  JOBID ...           START_TIME  SCHEDNODES
1487570 ... 2021-10-14T10:10:10  hkn0301
```

## 3.a Processing - sacct

- Obtain accounting information of a job

```
$ sbatch submit_script.sh
```

```
Submitted batch job 1487652
```

```
$ sacct -j 1487652
```

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

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

### ■ Obtain detailed information on the job

```
$ scontrol show job <job-ID>
JobId=1487569 JobName=submit_script.sh
  UserId=ab1234(27049) GroupId=scc(12345) ...
  Priority=4298 Nice=0 Account=kit QOS=normal
  JobState=RUNNING Reason=Prolong 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=2021-10-13...
  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_single AllocNode:Sid=uc2n997:2170796
  ReqNodeList=(null) ExcNodeList=(null)
  NodeList=uc2n362
  BatchHost=uc2n362
  ...
```

1) Consumed resources will be booked on your university / project

2) Your job state

3) Your time logging

4) Your selected partition

5) Your node list and Node on which job started

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

### ■ Obtain detailed information on the job

```
$ scontrol show job <job-ID>
JobId=1487569 JobName=submit_script.sh
...
NumNodes=1 NumCPUs=2 NumTasks=1 CPUs/Task=1 ...
TRES=cpu=2,mem=2250M,node=1,billing=2
Socks/Node=* NtasksPerN:B:S:C=0:0:*:1 CoreSpec=*
MinCPUsNode=1 MinMemoryCPU=1125M MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
OverSubscribe=OK Contiguous=0 Licenses=(null) ...
Command=/pfs/data5/home/kit/scc/ab1234/submit_script.sh
WorkDir=/pfs/data5/home/kit/scc/ab1234
StdErr=/pfs/data5/home/kit/scc/ab1234/slurm-1487569.out
StdIn=/dev/null
StdOut=/pfs/data5/home/kit/scc/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.c Processing – Login onto compute node

- While the job is running (state = R): login to dedicated compute 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 [--overlap] /usr/bin/bash
ab1234@hkn0301:~$
```

- srun** adds another step to your job. Once **main job finishes**, 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 – Cancel the job

- You can cancel your job, e.g. if
  - Submitted wrongly
  - Job does not behave as expected

```
$ 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

- **Goal:** practice monitoring and cancelling of jobs.
- 1. Modify your script such that it executes a command to wait for 600 seconds (**sleep 600**)
- 2. Set a walltime of 10 minutes and give a name to your job.
- 3. Submit your job script with **sbatch**.
- 4. Use **squeue** to check the status.
- 5. Use **scontrol show job** to see from which directory you started the job.
- 6. Use **scancel <job-ID>** to cancel your job.

## Tutorial 2 - Solution

- **Goal:** practice monitoring and cancelling of jobs.
- 1. Modify your script such that it executes a command to wait for 600 seconds (**sleep 600**)
- 2. Set a walltime of 10 minutes and give a name to your job.
- 3. Submit your job script with **sbatch**.

```
#!/bin/bash
#SBATCH -N 1 -n 1
#SBATCH -t 00:10:00
#SBATCH --mem-per-cpu=500
#SBATCH -J myJobName

sleep 600
```

```
$ sbatch -p single --reservation=ws submit_script.sh # bwunicluster
```

```
$ sbatch -p cpuonly --reservation=ws submit_script.sh # horeka
```

- 4. Use **squeue** to check the status.
- 5. Use **scontrol show job** to see from which directory you started the job.

```
$ scontrol show job 1487685 | grep WorkDir
WorkDir=/pfs/data5/home/kit/scc/ab1234/workshop
```

- 6. Use **scancel <job-ID>** to cancel your job.

# Interactive jobs

# Interactive jobs

- Jobs on login nodes are not permitted
- Sometimes interactive access is required (e.g. for debugging, heavy compilation jobs, etc.)  
=> **interactive slurm jobs**

```
ab1234@hkn1990:~$ salloc -p cpuonly -n 1 -t 10 --mem=2000
salloc: Granted job allocation 1487738
salloc: Waiting for resource configuration
salloc: Nodes hkn0301 are ready for job
```

Job is waiting to start, Do Not interrupt the command

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

Job running. You are now on a 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 ***
```

```
exit
```

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

Requested time for the interactive job ran out

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

Back on the login node

**Thank you for your attention.**

**Questions?**