Introduction to Abel and queuing system

April 26 2018 - INF9380

Sabry Razick PhD.
Senior Engineer
Research Infrastructure Services Group,
Department for Research Computing, USIT

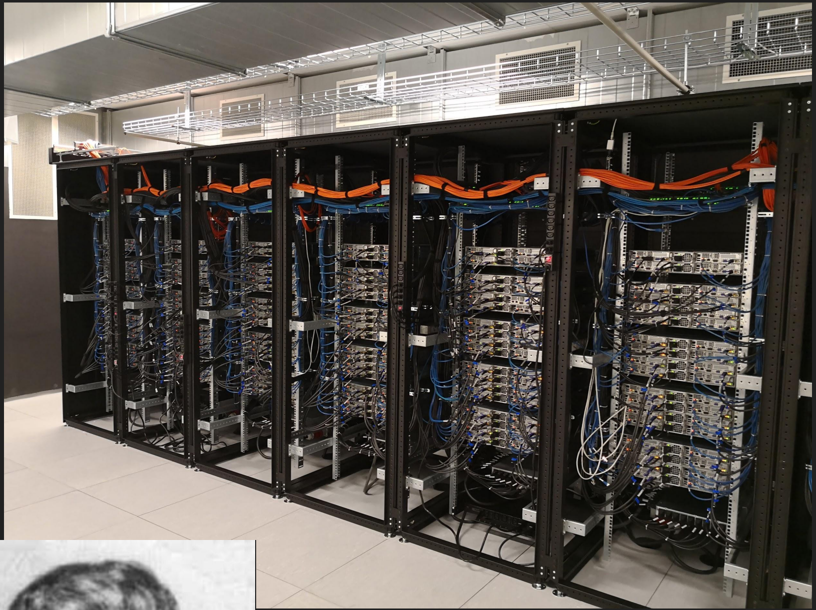
Research Infrastructure Services Group

- Gruppe for forskningsinfrastruktur (FI)
- The FI group provides access to IT resources and high performance computing to researchers at UiO and to NOTUR users
- http://www.usit.uio.no/om/organisasjon/itf/fi/index.html
- Part of USIT
- Contact: hpc-drift@usit.uio.no

Topics

- FI Group
- Abel details
- Getting an account & Logging in
- SLURM Workload Manager
- Running a simple job
- Jobscripts
  - Arrayrun
  - srun
  - MPI
  - GPU

- Nodes - 700+ (704),
- Cores - 10000+ (11,392), 258 Teraflops
- Total memory - 50 TiB+ (50.78 TebiBytes)
- Total storage - 400 TiB using BeeGFS
- Some nodes with NVIDIA Kepler II GPUs (c19-9)
- 96th most powerful in 2012 , now 444th (June 2015)



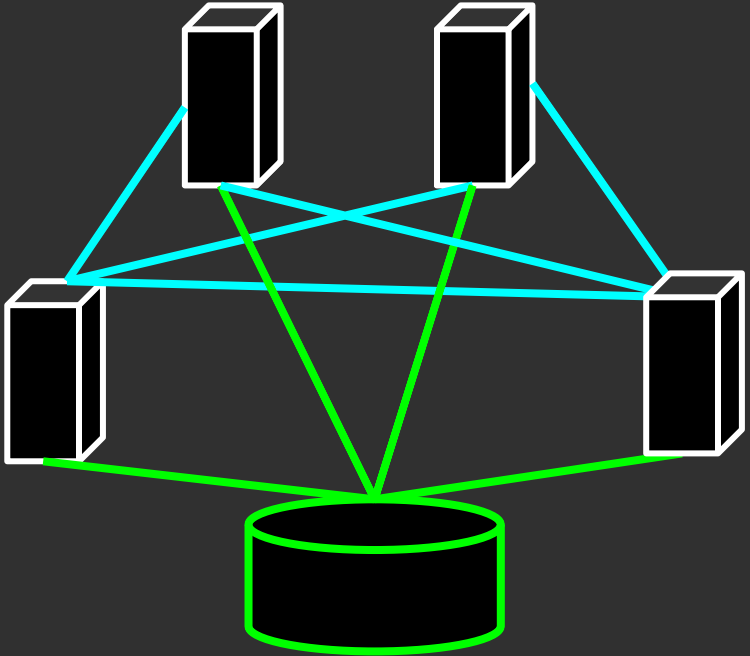
Niels Henrik Abel

The Research Computing Services

- Operation of Abel and Colossus computer clusters
- Joint operation of the Fram cluster and NIRD
- User support
- Data storage
- Teaching/Training
- Participation in Nordic and European projects

Abel Cluster

- Hardware
  - Powerful computers(nodes)
  - Nodes are similar
  - High-speed interconnection
  - Access to a common file system
  - Tightly coupled
- Software
  - Operating system Linux, 64 bit Centos 6.8 (Rocks Cluster Distribution based)
  - Identical mass installations.
  - Queuing system enables timely execution of many concurrent processes
- How it is different from a GRID or CLOUD ?



Accessing Abel

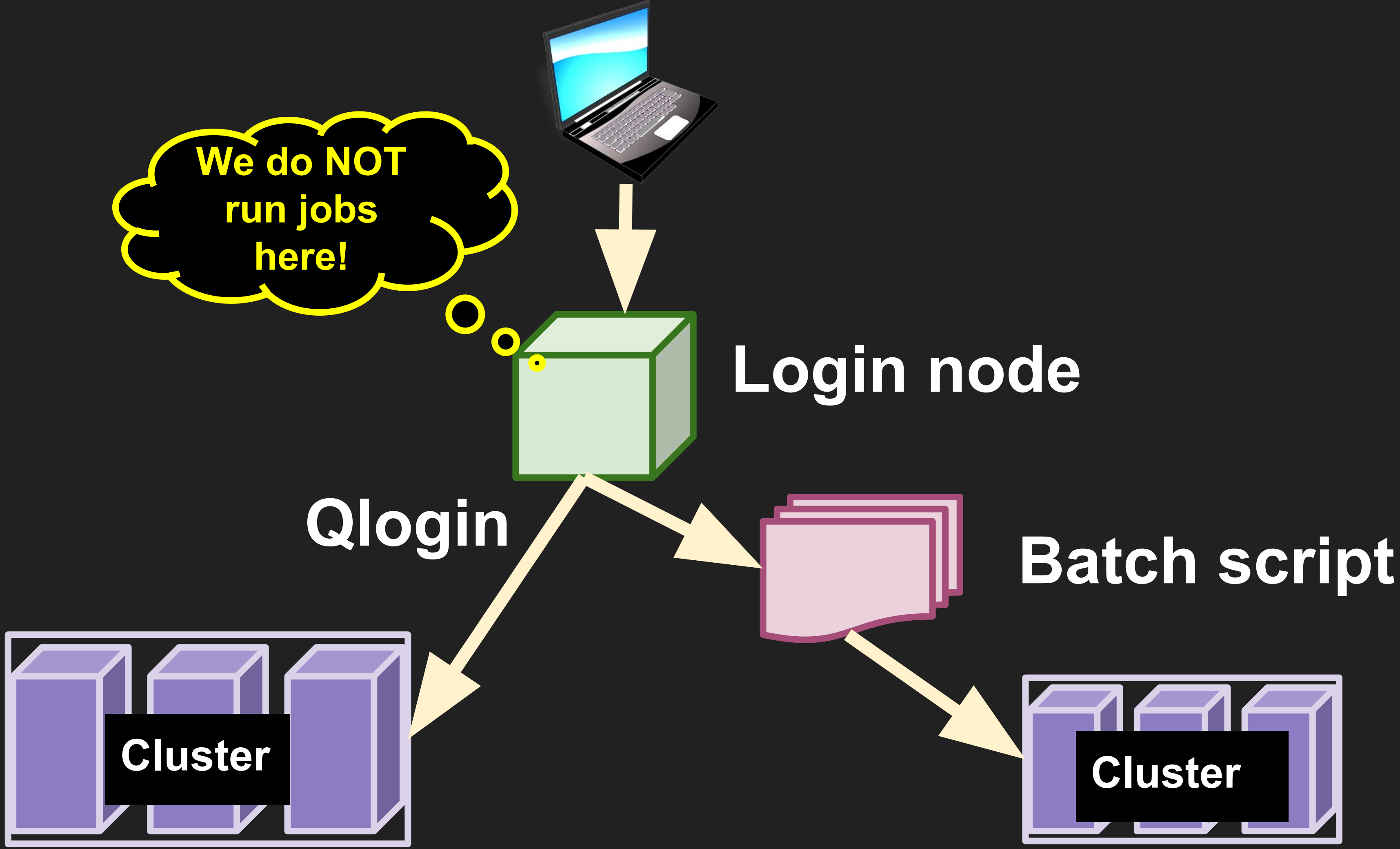
- If you are working or studying at UiO, you can have an Abel account directly from us.
- If you are Norwegian scientist (or need large resources), you can apply through NOTUR – http://www.sigma2.no
- Help and info email: hpc-drift@usit.uio.no

Connecting to Abel

- Linux
  - Redhat - RHEL
  - Ubuntu
- Windows - using Git BASH, Putty and WinSCP
- Mac OS

File transfer on command line

- When you log into Abel you are in one of the login nodes login0 - login3.
- Please DO NOT execute programs (jobs) directly on the login nodes.
- Jobs are submitted to Abel via the queuing system.
- The login nodes are for
  - logging in, copying files, editing, compiling, submitting jobs, checking job status, etc.
- For interactive execution use qlogin.


We do NOT run jobs here!
Login node
Qlogin
Batch script
Cluster
Cluster

File transfer

- Unix users can use secure copy or rsync commands
  - For large files, use rsync command
  - Many large files use parallel rsync
    - module load parsync
  - For large number of file use archives
  - E.g. Copy myfile.txt from the current directory on your machine to your home area on Abel:
    - scp myfile.txt user_name@abel.uio.no:
- Windows users we recommend WinSCP

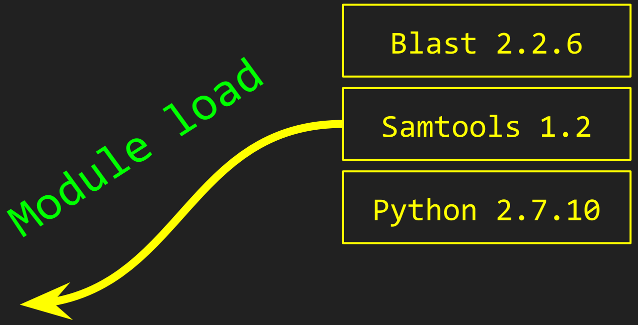
Software on Abel

- Available on Abel:
  - http://www.uio.no/hpc/abel/help/software
- Software on Abel are organized in to modules.
- List all software (and version) organized in modules:

```
> module avail
```

- Load software from a module:

```
> module load <module_name>
```



Queue management - SLURM

- Simple Linux Utility for Resource Management (workload manager)
  - Allocates exclusive and/or non-exclusive access to resources (computer nodes) to users for some duration of time
  - Provides a framework for starting, executing, and monitoring work on a set of allocated nodes.
  - Managing a queue of pending work.

Fair resource allocation

- When you request resources, SLURM on Abel will consider number of things before granting it
  - Does your project has enough CPU hours to pay for this. It will consider total allocated + reserved when doing this.
  - Can/should the cluster provide you with resources (resource combination)
  - Depending on the current load. how long others need to wait if you job starts.

Laptop Vs queue

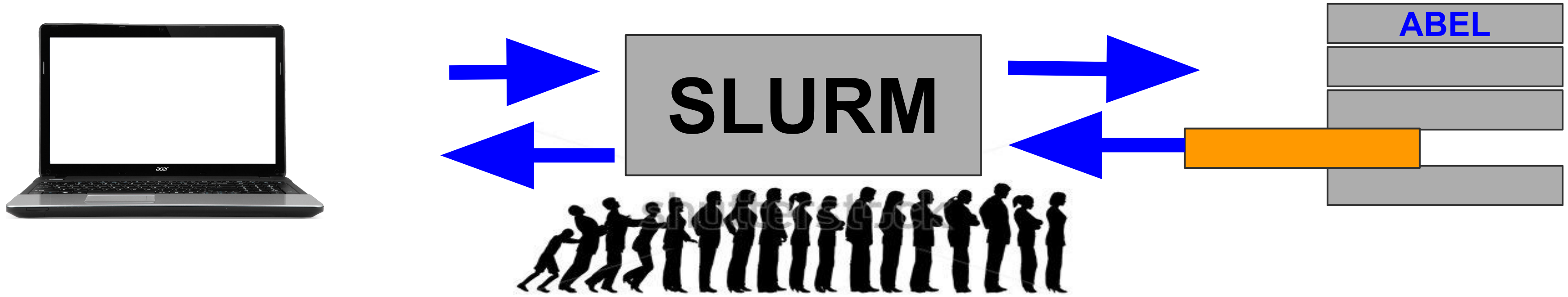
- Double click on an icon, give parameters or upload data
  - Wait until execution is finished
  - Inspect results
- Terminal
  - ./myprg input1.txt out_put.file
  - Inspect results

Project account

- Needs a project account.
  - UiO students and employees assigned to project uio by default. But this has a long queue
  - Recommended to apply for own project account from Sigma (https://www.sigma2.no/content/apply-e-infrastructure-resources)
  - Use the following command to find out the projects that you have access to

```
> projects
```

Running a job on the cluster -1



- Login to Abel from you laptop
- Request to occupy some resources from SLURM
- Wait until SLURM grant you the resources
- Execute the job as it was in your laptop

Interactive use of Abel - qlogin

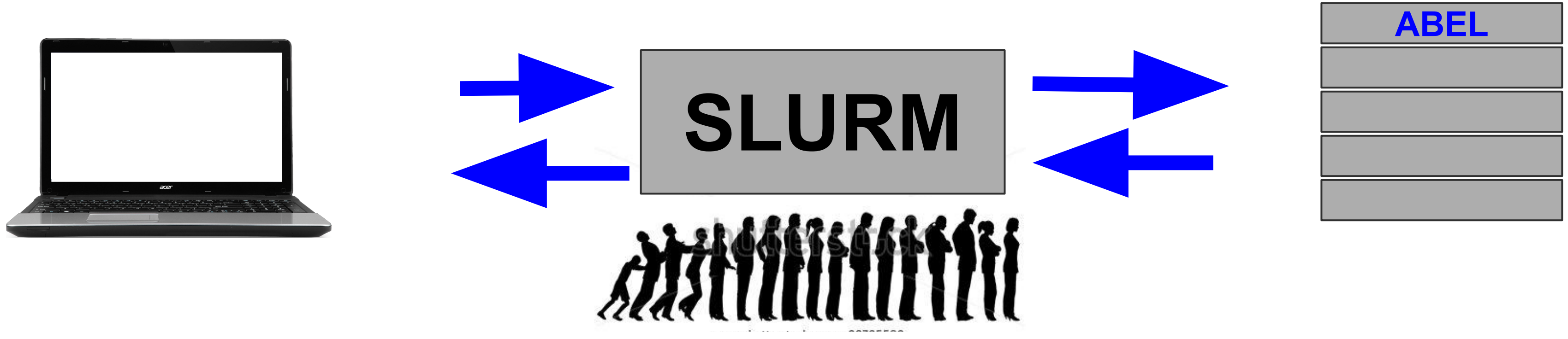
- Send request for a resource (e.g. 4 cores)
- Work on command line when a node becomes available
- Example - book one node on Abel for your interactive use for 1 hour:

```
qlogin --account=your_project --nodes=1 --exclusive --time=01:00:00
```

- Run “source /cluster/bin/jobsetup“ after receiving allocation
- For more info, see:

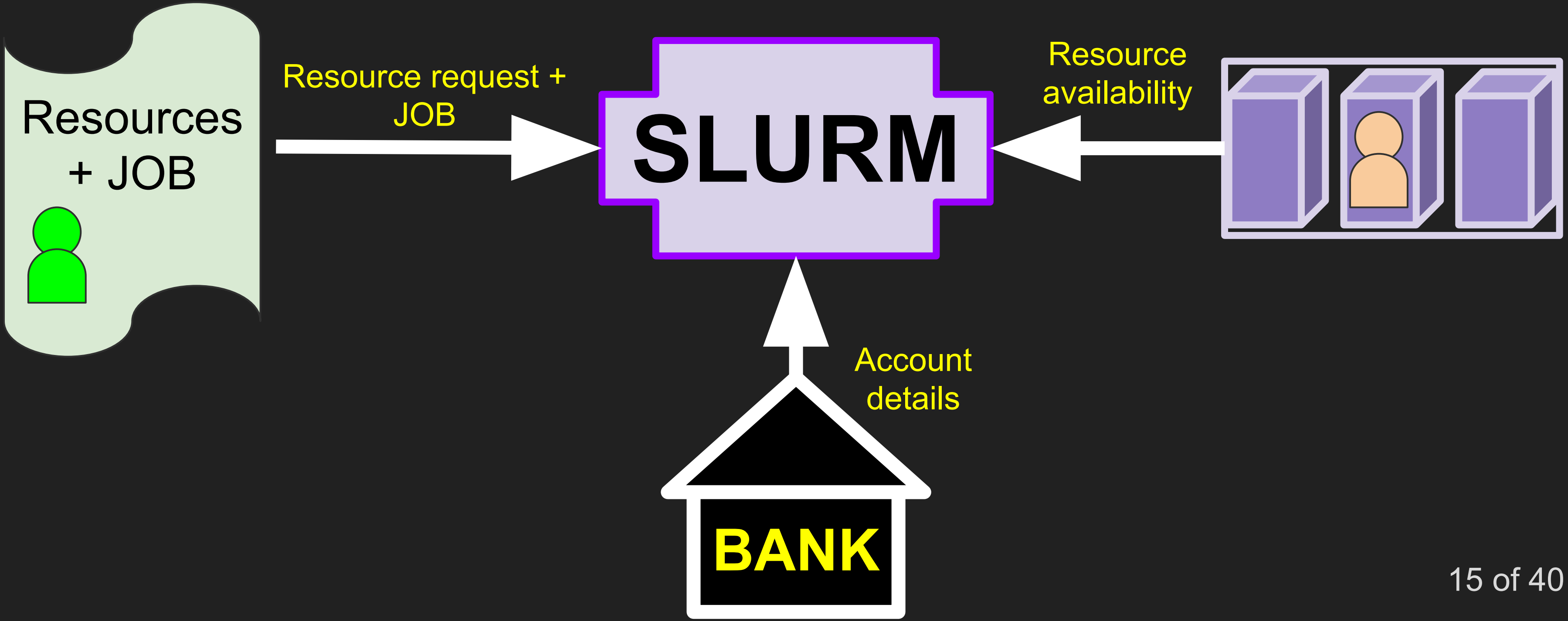
http://www.uio.no/hpc/abel/help/user-guide/interactive-logins.html

Running a job on the cluster - 2

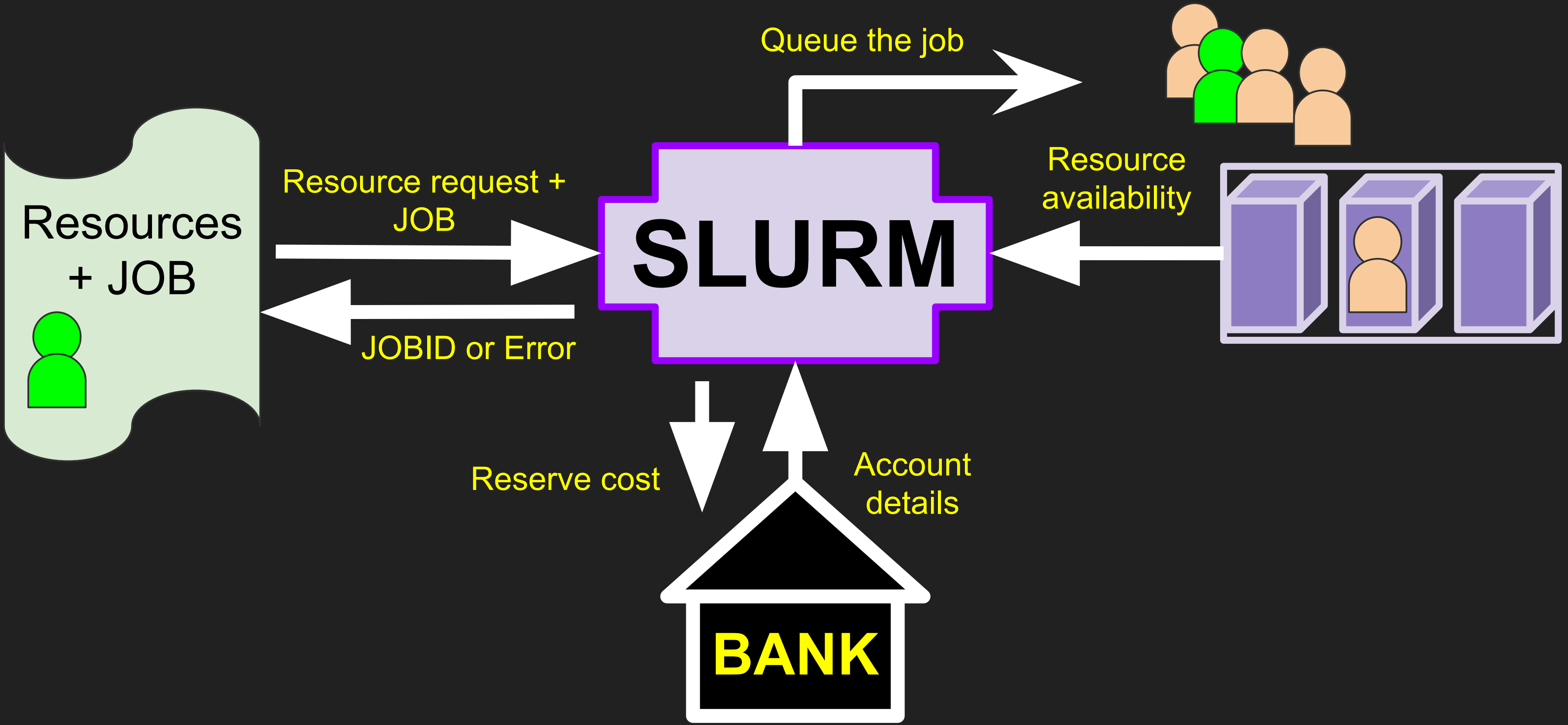


- Login to Abel from you laptop
- Create a job script, with parameters and include the program to run
- Hand it over to the workload manager
- The workload manager will handle the job queue, monitor the progress and let you know the outcome.
- Inspect results

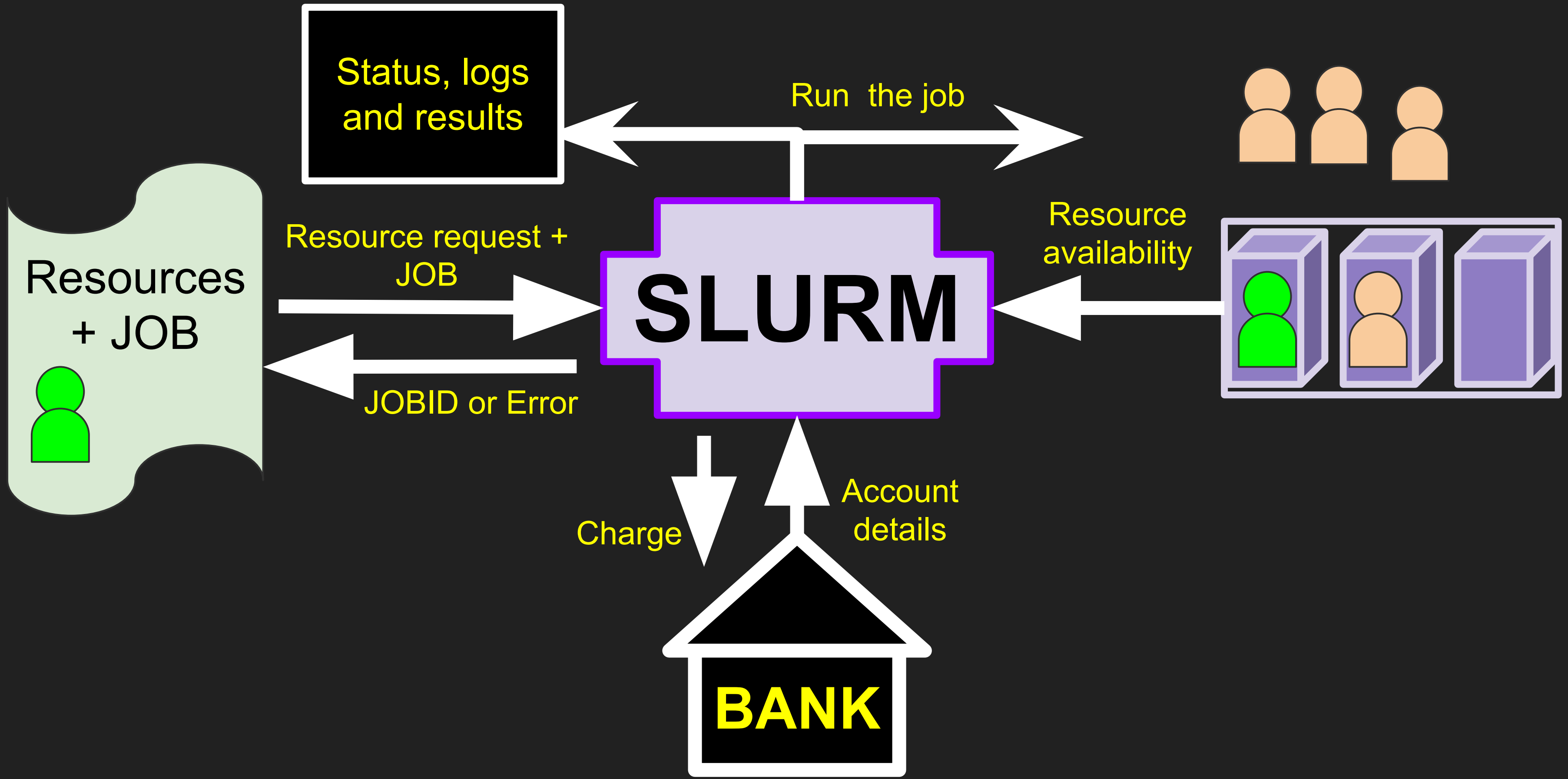
Job submission to Abel



Job submission to Abel



Job submission to Abel



Computing on Abel

- Get an account and one or more projects (Sigma)
- Load modules or install software
- Submit a job to the queuing system
- Communicate with the queuing system
- Retrieve results (or errors) when the job is done
- Read tutorial: http://www.uio.no/hpc/abel/help/user-guide

Job script

- Job script - shell script including the command that one needs to execute.

Job script

```
#!/bin/bash

#SBATCH --job-name=norbis_hello
#SBATCH --account=ln0002k
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=512M
#SBATCH --time=00:01:00

source /cluster/bin/jobsetup


python hello.py
```

Job script

```
#!/bin/bash
#SBATCH --job-name=norbis_hello
#SBATCH --account=ln0002k
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=512M
#SBATCH --time=00:01:00
```

Resources

```
source /cluster/bin/jobsetup
```

Setup

```
python hello.py
```

Job

Understand resources

- If program can not or would not use parallel processing, no point requesting more than one core.
- If you request one tasks you will get exactly one core, from one processor in a single compute node.
  - What does this mean ?

Supermicro X9DRT compute nodes

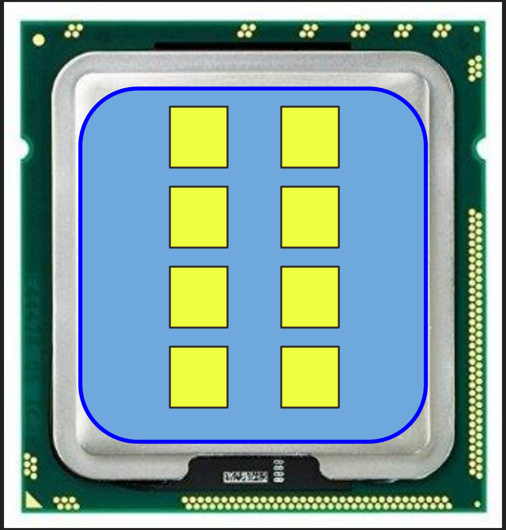
Samsung DDR3
1600 MHz

Intel® Xeon® Processor
E5-2670
Octa core (8)

Datasheet:
http://ark.intel.com/products/64595/Intel-Xeon-Processor-E5-2670-20M-Cache-2_60-GHz-8_00-GTs-Intel-QPI

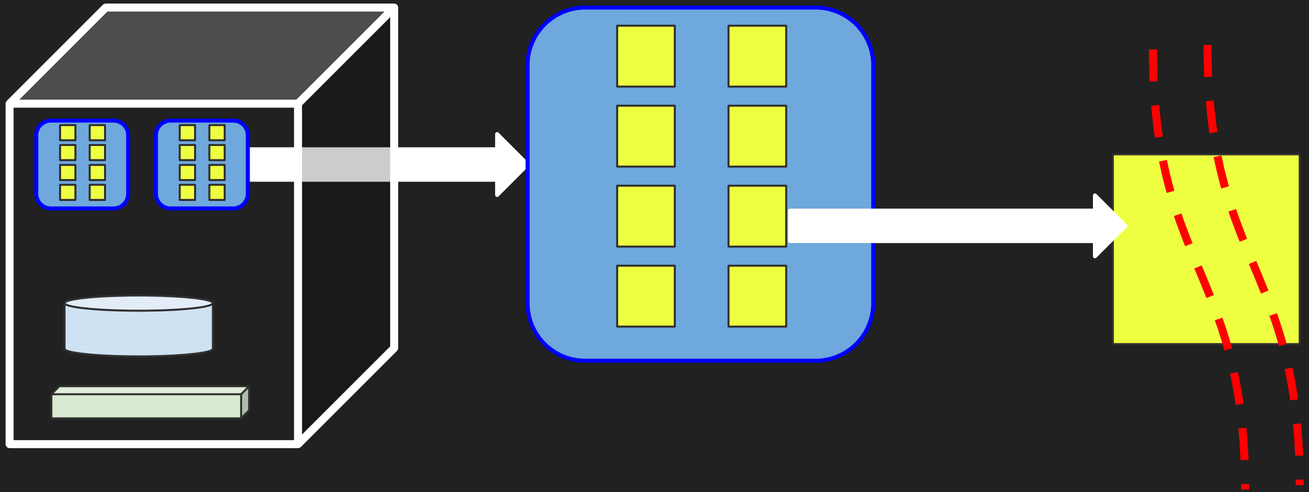
Supermicro X9DRT compute nodes

Samsung DDR3
1600 MHz



Intel® Xeon® Processor
E5-2670
Octa core (8)

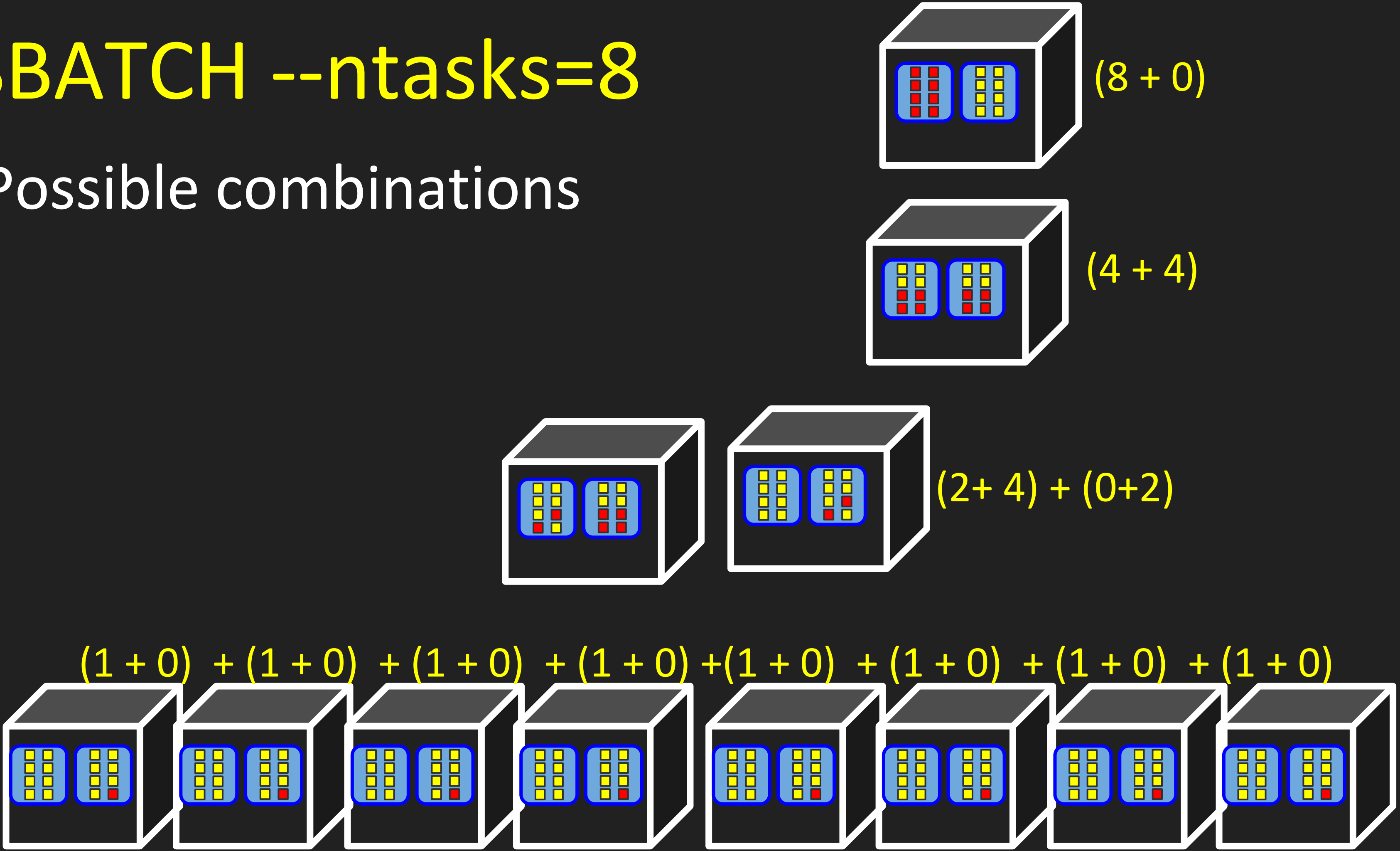
Datasheet:
http://ark.intel.com/products/64595/Intel-Xeon-Processor-E5-2670-20M-Cache-2_60-GHz-8_00-GTs-Intel-QPI



- Supermicro X9DRT compute node
- 2 X Intel Xeon E5-2670 (Sandy Bridge) based running at 2.6 GHz,
- 16 physical compute cores total per node.
- Each node have 64 GiBytes of Samsung DDR3 memory

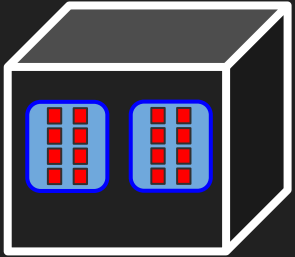
#SBATCH --ntasks=8

Possible combinations



```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=16
```

```
#SBATCH --ntasks=2
#SBATCH --cpus-per-task=8
8 X 2 =16
```



Time

- Requesting long time will make the queue waiting time longer
- Maximum 1 week (7:00:00:00), hugemem or long partitions 4 weeks

Memory

- Can request in megabytes or gigabytes
- #SBATCH --mem-per-cpu=1024M
- #SBATCH --mem-per-cpu=1G
- Request is per CPU (following will reserve 4Gb in total)
  - #SBATCH --cpus-per-task=4
  - #SBATCH -ntask=1
  - #SBATCH --mem-per-cpu=1G
- When requesting a lot of memory you may occupy more cores than requested (following will occupy 16 cores on a regular node)
  - #SBATCH -ntask=1
  - #SBATCH --mem-per-cpu=61G

Core hours calculation

```
KMEM=4580.2007628294

(/cluster/var/accounting/PE_factor)

PE= NumCPUs

if(MinMemoryCPU>KMEM){

    PE=PE*(MinMemoryCPU/KMEM)

}

PE_hours = $PE * TimeLimit / 3600
```

Core hours calculation

```
#SBATCH --nodes=1

#SBATCH --time=01:00:00

#SBATCH --ntasks-per-node=4

#SBATCH --mem-per-cpu=15G
```

(4 X 1) +12 =16

*All memory occupied,

KMEM=4580.2007628294

PE= 4

#(15 * 1024)>KMEM so

PE=4 * ((15 * 1024)/KMEM) =13.41

PE_hours = 13.41 * (1 * 60 * 60) / 3600 =13.41

**Use the command cost to check account balance

```
$ cost
```

Sample job script

`hello.slurm`

```
#!/bin/bash

#SBATCH --job-name=RCS1115_hello
#SBATCH --account=xxx
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=256M
#SBATCH --time=00:05:05

source /cluster/bin/jobsetup
set -o errexit

sleep 5m
echo “Hello from ”
hostname
```

Join the queue

```
> sbatch hello.slurm

> scontrol show job <JOBID>

> squeue -u <username>

> sacct -j <JOBID>
```

Investigate

```
>scontrol show job 12989353
   JobId=12989353 Name=RCS1115_hello
   UserId=sabryr(243460) GroupId=users(100)
   Priority=22501 Nice=0 Account=staff    QOS=staff
   JobState=COMPLETED Reason=None
...
   RunTime=00:00:02 TimeLimit=00:01:00 Ti
Command=../RCS_tutorial/hello.slurm
WorkDir=../RCS_tutorial
StdErr=../RCS_tutorial/slurm-12989552.out
StdOut=../RCS_tutorial/slurm-12989552.out
```

Some usefull commands

- scancel <JOBID> - Cancel a job before it ends
- dusage - find out your disk usage
- squeue - list all queued jobs and find out the
- squeue -t R | more -position of your job
- cost - account balance
- sinfo - node info
- sacct - details of jobs that have ended

Environment variables

- SLURM_JOBID – job-id of the job
- SCRATCH – name of job-specific scratch-area
- SLURM_NPROCS – total number of cpus requested
- SLURM_CPUS_ON_NODE – number of cpus allocated on node
- SUBMITDIR – directory where sbatch were issued
- SCRATCH - scratch directory location (on /work)
- TASK_ID – task number (for arrayrun-jobs)

Check Variables

```
checkvariables.slurm
#!/bin/bash

#SBATCH --job-name=RCS1115_hello
#SBATCH --account=staff
#SBATCH --time=00:01:05
#SBATCH --mem-per-cpu=512M
#SBATCH --ntasks=1

source /cluster/bin/jobsetup

echo "SLURM_JOBID=" $SLURM_JOBID
echo "SCRATCH=" $SCRATCH
echo "SLURM_NPROCS=" $SLURM_NPROCS
echo "SLURM_CPUS_ON_NODE=" $SLURM_CPUS_ON_NODE
echo "SUBMITDIR= "$SUBMITDIR
echo "TASK_ID="$TASK_ID
```

Parallel jobs

Arrayrun

- Every arrayrun needs one “worker” script and one “submit” script.
- Worker script - similar to the jobscript we have used
- The submit script executes the arrayrun command
- TASK_ID variable is used to distinguish each run

Arrayrun - submit script

array_submit.slurm

```
#!/bin/bash

  #SBATCH --job-name=RCS1115_hello
  #SBATCH --account=staff
  #SBATCH --time=00:02:00
  #SBATCH --ntasks=1
  #SBATCH --mem-per-cpu=256M

  source /cluster/bin/jobsetup
  set -o errexit

  module purge
  module load Python/3.5.2-foss-2016b

  arrayrun 1-3 array_worker.slurm
```

Arrayrun - submit script

array_worker.slurm

```
#!/bin/bash

#SBATCH --job-name=RCS1115_hello
#SBATCH --account=staff
#SBATCH --time=00:02:00
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=256M

source /cluster/bin/jobsetup
set -o errexit

module purge
module load Python/3.5.2-foss-2016b

arrayrun 1-3 array_worker.slurm
```

Investigate job

```
> ssh to compute node and use top

> ssh c16-17 "top -n1 -b | grep <USER_NAME>"

> ssh c60-1 "lscpu | grep Socket"
```

srun

srun_test.slurm

```
#!/bin/bash

#SBATCH --job-name=RCS1115_hello
#SBATCH --account=staff
#SBATCH --time=00:01:00
#SBATCH --ntasks=3
#SBATCH --mem-per-cpu=256M

FILES=(/path/to/data/*)

source /cluster/bin/jobsetup
set -o errexit

echo "number of instances "$SLURM_NTASKS
srun aprogram.sh
```

MPI

- Message Passing Interface
- Especially when running on more than one node
- MPI is a language-independent communications protocol used for programming parallel computers.
- jobs specifying more than one node automatically get
  - #SBATCH --constraint=ib
- We OpenMPI and Intel® MPI on Abel
  - module load openmpi.gnu/1.10.2
  - module load openmpi.intel/1.10.2
  - module loac intelmpi.intel/5.0.2

MPI

mpi_test.slurm

```
#!/bin/bash

#SBATCH --job-name=RCS1115_hello
#SBATCH --account=staff
#SBATCH --time=00:01:00
#SBATCH --ntasks=3
#SBATCH --mem-per-cpu=256M

source /cluster/bin/jobsetup
set -o errexit

module purge
module load Python/3.5.2-foss-2016b

mpirun python mpi_test.py
```

GPU

GPU job

```
gpu.slurm
  #!/bin/bash

  #SBATCH --job-name=GPU_test
  #SBATCH --account=ln0002K
  #SBATCH --ntasks=1
  #SBATCH --time=01:00
  #SBATCH --mem-per-cpu=1G
  ## Ask for 1 GPU
  #SBATCH --partition=accel --gres=gpu:1

  ## Set up job environment:
  source /cluster/bin/jobsetup
  set -o errexit # exit on errors

  ##Print number of available GPUs and some details about them
  echo $CUDA_VISIBLE_DEVICES
  nvidia-smi --query-gpu=name,memory.total,memory.free,memory.used --format=csv
```

Visualization

Visualization example

```
•   ssh -X abel.uio.no
•   qlogin -A staff --ntasks=2 --mem-per-cpu=2G --time=10:00 --job-name=R_test
•   module load R/3.4.4
•   R
•   > x <- c(1:5); y <- x
•   > plot(x, y, type="p")
```

I/O

$SCRATCH

- If you are accessing a file multiple time during a job
  - Use scratch directory
  - Use /work/users/<USERNAME>
- Choose where you start the job from
- On Abel jobs accessing files and/or writing out large outputs will run faster if /work is used compared to running from $HOME
- When running jobs from /work, there is no need use $SCRATCH
- Remember to use to `chkfile` copy back results, otherwise the results will be deleted at the end of the job.

$SCRATCH

scratch.slurm

```
  #!/bin/bash

  #SBATCH --job-name=usescratch
  ....

  source /cluster/bin/jobsetup
  set -o errexit

  echo $SUBMITDIR
  echo $SCRATCH
  cp $SUBMITDIR/input* $SCRATCH
  chkfile output.txt
  cat input* > output.txt
```

$LOCALTMP

Localtmp.slurm & fromhome.slurm

```
#!/bin/bash
....
source /cluster/bin/jobsetup
set -o errexit

echo $SUBMITDIR
echo $LOCALTMP
df -h $LOCALTMP

cp manyfiles.sh $LOCALTMP
cd $LOCALTMP
pwd
./manyfiles.sh
rsync all.txt $SUBMITDIR/
```

I/O - Files

- When handling very large number of files try to use
  - Archives - just in time extract or write directly to archive
  - Cleanup unwanted files
  - Copy back only output files needed when using $SCRATCH , e.g, do not copy back input data
- Get advice on using /tmp directory on compute nodes when thousands of files.

I/O - Files

- Create an archive (no compression)
  - tar -cvf <ARCHIVE_NM> <FILES>
  - tar -cvf file.tar *txt
- List content
  - tar -tvf <ARCHIVE_NM>
- Append a file
  - tar --append --file <ARCHIVE_NM> <NEW_FILES>
  - tar --append --file files.tar.gz 1.txt
- Extract all
  - tar -xvzf <ARCHIVE_NM>
- Extract one
  - tar -xvf <ARCHIVE_NM> <FILES>

Thank you.

http://www.uio.no/english/services/it/research/hpc/abel/

hpc-drift@usit.uio.no

sbatch - memory

- #SBATCH --mem-per-cpu=Size
    - Memory required per allocated core (format: 2G or 2048M)
    - How much memory should one specify? The maximum usage of RAM by your program (plus some). Exaggerated values might, delay the job start.
- #SBATCH --partition=hugemem
    - If you need more than 61GB of RAM on a single node (up to 1 TiB).

sbatch - time

- #SBATCH --time=hh:mm:ss
  - Wall clock time limit on the job
  - Some prior testing is necessary. One might, for example, test on smaller data sets and extrapolate. As with the memory, unnecessarily large values might delay the job start.
  - This costs you (from allocated operation resources)
  - Until a job is finished this will be reserved.
- #SBATCH --begin=hh:mm:ss
  - Start the job at a given time (or later)
- #SBATCH --partition=long
  - Maximum time for a job is 1 week (168 hours). If more needed, use

sbatch – CPUs and nodes

- Does your program support more than one CPU?
- If so, do they have to be on a single node?
- How many CPUs will the program run efficiently on?
- #SBATCH --nodes=Nodes
  - Number of nodes to allocate
- #SBATCH --ntasks-per-node=Cores
  - Number of cores to allocate within each allocated node
- #SBATCH --ntasks=Cores
  - Number of cores to allocate
- #SBATCH --cpus-per-task=Cores
  - Threads on one node

sbatch - interconnect

- #SBATCH --constraint=*ib*
  - Run job on nodes with infiniband
  - Gigabit Ethernet on all nodes
  - All nodes on Abel are equipped with InfiniBand (56 Gbits/s)
  - Select if you run MPI jobs

sbatch - constraint

- #SBATCH --constraint=*feature*
  - Run job on nodes with a certain feature - *ib*, *rackN*.
- #SBATCH *--constraint=ib&rack21*
  - If you need more than one constraint
  - *in case of multiple specifications, the later overrides the earlier*

sbatch - files

- #SBATCH --output=file
    - Send 'stdout' (and stderr) to the specified file (instead of slurm-xxx.out)
- #SBATCH --error=file
    - Send 'stderr' to the specified file
- #SBATCH --input=file
    - Read 'stdin' from the specified file

sbatch – low priority

- #SBATCH --qos=lowpri
  - Run a job in the lowpri queue
  - Even if all of your project's cpus are busy, you may utilize other cpus
  - Such a job may be terminated and put back into the queue at any time.
  - If possible, your job should ensure its state is saved regularly, and should be prepared to pick up on where it left off.
  - Note: Notur projects cannot access lowpri.