# Introduction to Abel and queuing system

April 27 2016 - INF5380

Sabry Razick PhD.
Senior Engineer
The Research Computing Services Group, USIT

# The Research Computing Services (Seksjon for IT i Forskning)

- The RCS group provides access to IT resources and high performance computing to researchers at UiO and to NOTUR users
- http://www.uio.no/english/services/it/research/
- Part of USIT
- Contact: hpc-drift@usit.uio.no

# Topics

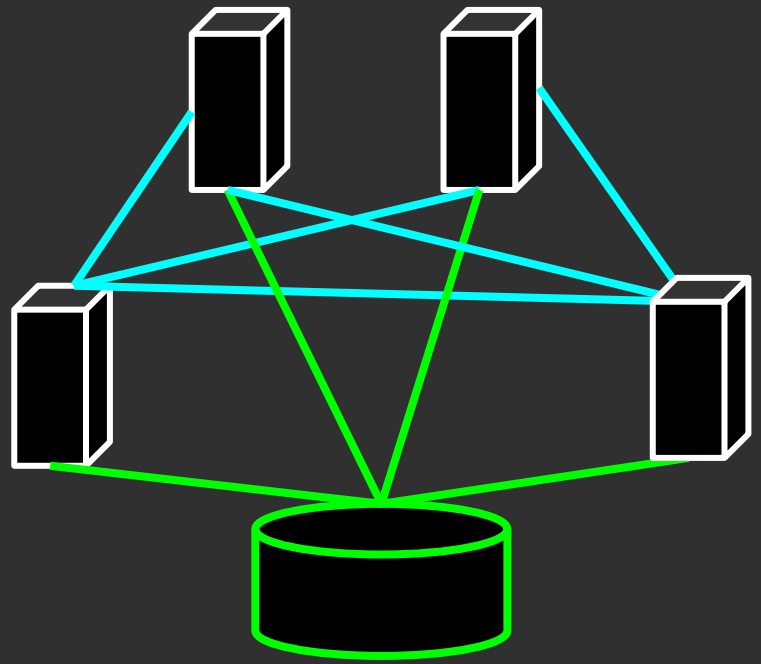
- The Research Computing Services group
- Abel details
- Getting an account & Logging  in
- SLURM Workload Manager
- Running a simple job

# The Research Computing Services

- Operation of Abel and Colossus  computer clusters
- User support
- Data storage
- Secure data analysis and storage - TSD
- Portals (access cluster through web based GUI)
  - Lifeportal (https://lifeportal.uio.no/)
  - Geo (https://geoportal-dev.hpc.uio.no)
  - Language (https://ps.hpc.uio.no)

# 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 6.7 (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 ?



# Abel in numbers

- Nodes - 700+ (704),
- Cores - 10000+ (11,392), 258 Teraflops
- Total memory - 50 TiB+ (50.78 TebiBytes)
- Total storage - 400 TiB using BeeGFS
- 96th most powerful in 2012 , now 444th (June 2015)

# Accessing Abel

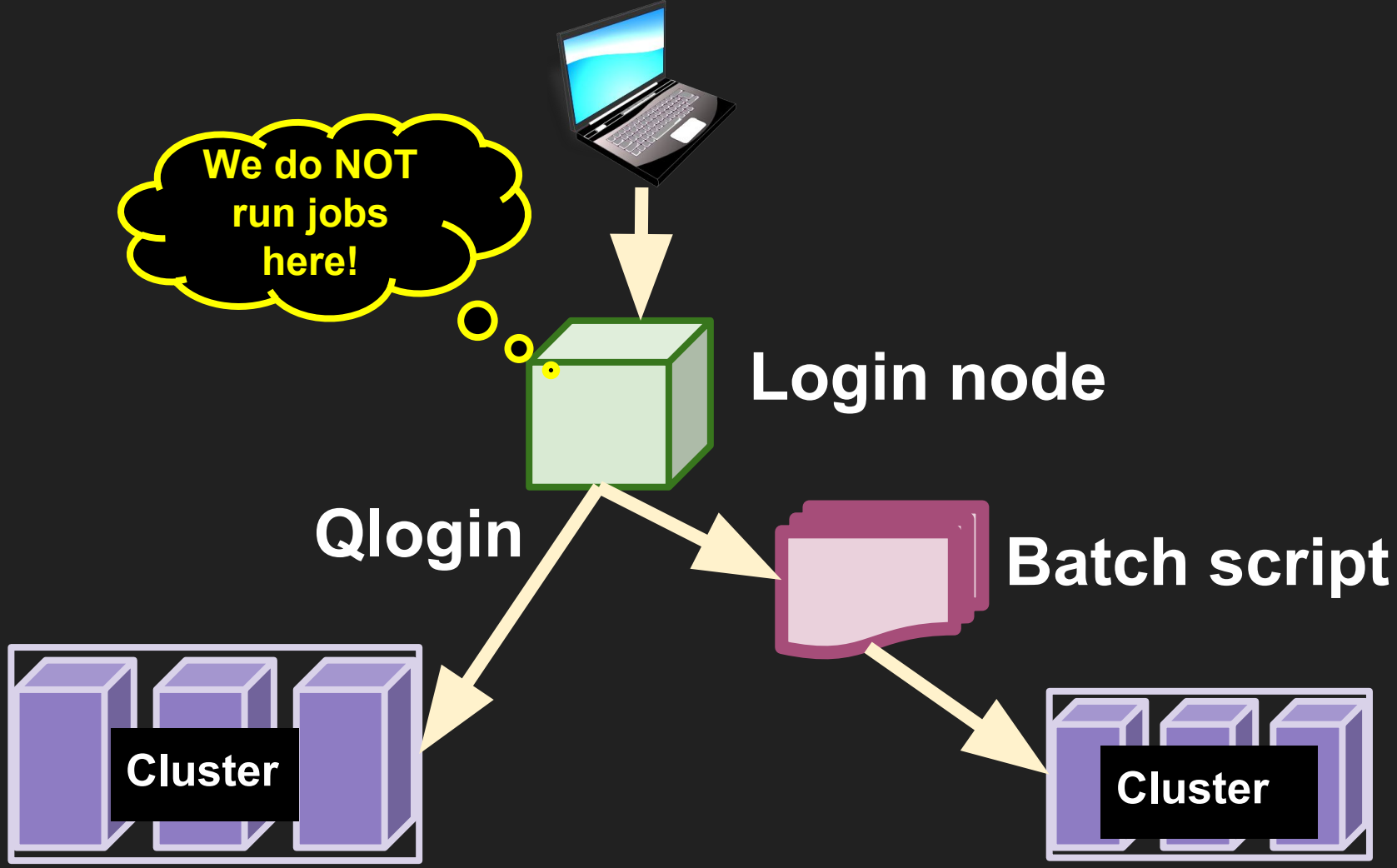
- If you are working or studying at UiO, you can have an Abel account directly from us.
- If you are Norwegian scientist (or need large resources), you can apply through NOTUR – http://www.notur.no
- Help and info email: hpc-drift@usit.uio.no

# Connecting to Abel

- Linux
  - Redhat - RHEL
  - Ubuntu
- Windows - using 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
- 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:
- For large files, use rsync command

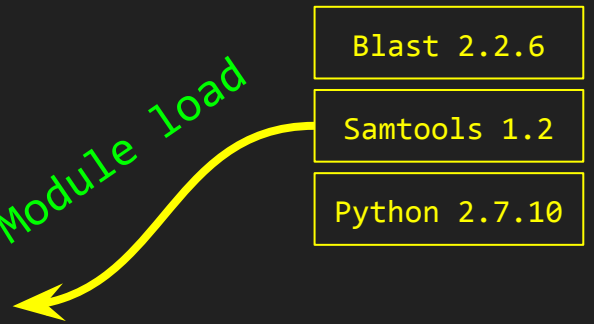
# Software on Abel

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

```
> module avail
```

- Load software from a module:

```
> module load module_name
```



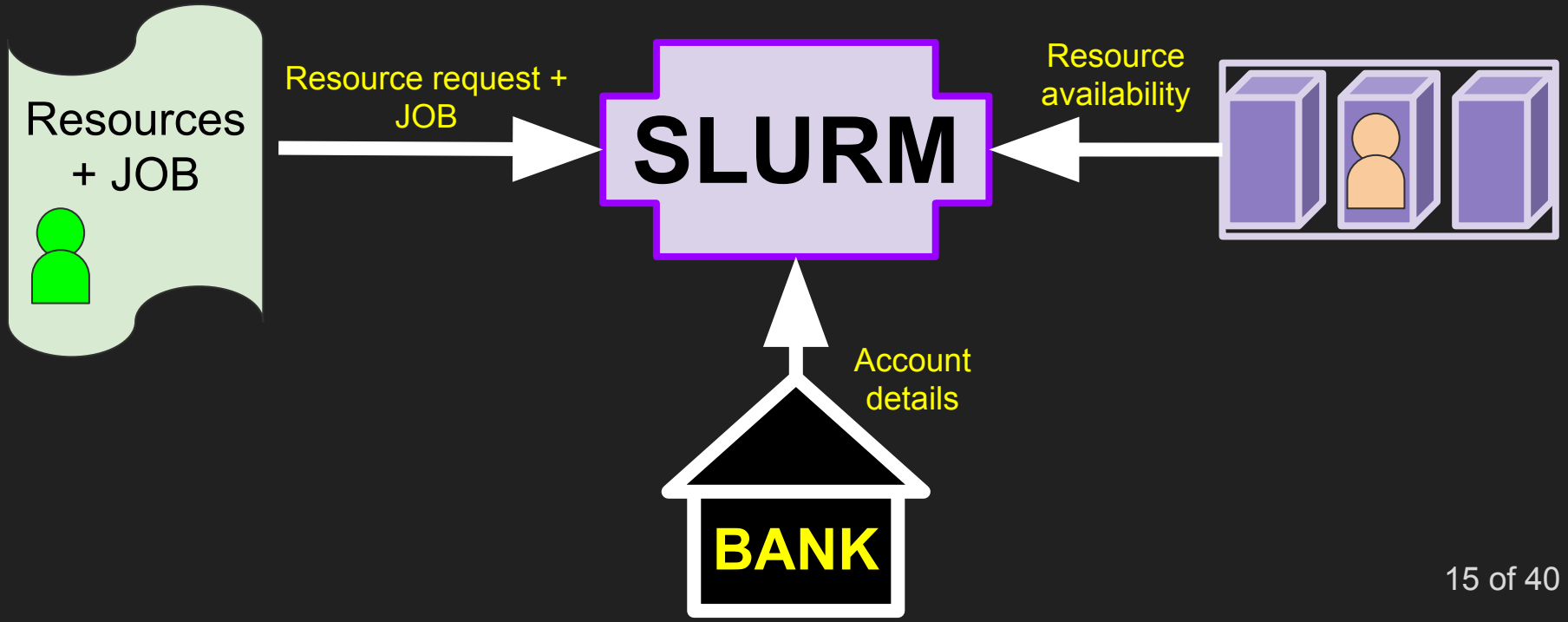
# Queue management - SLURM

- 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.

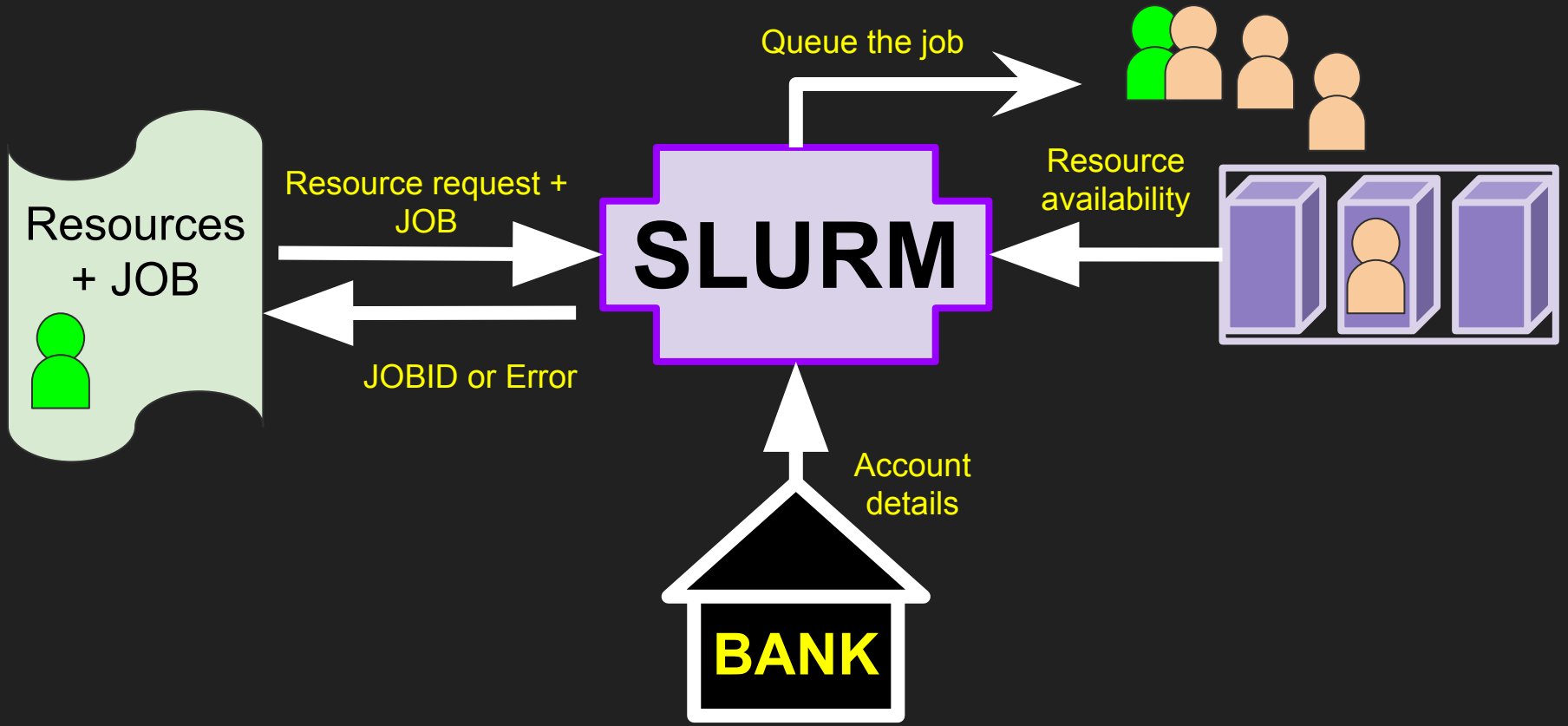
# Fair resource allocation

- 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.

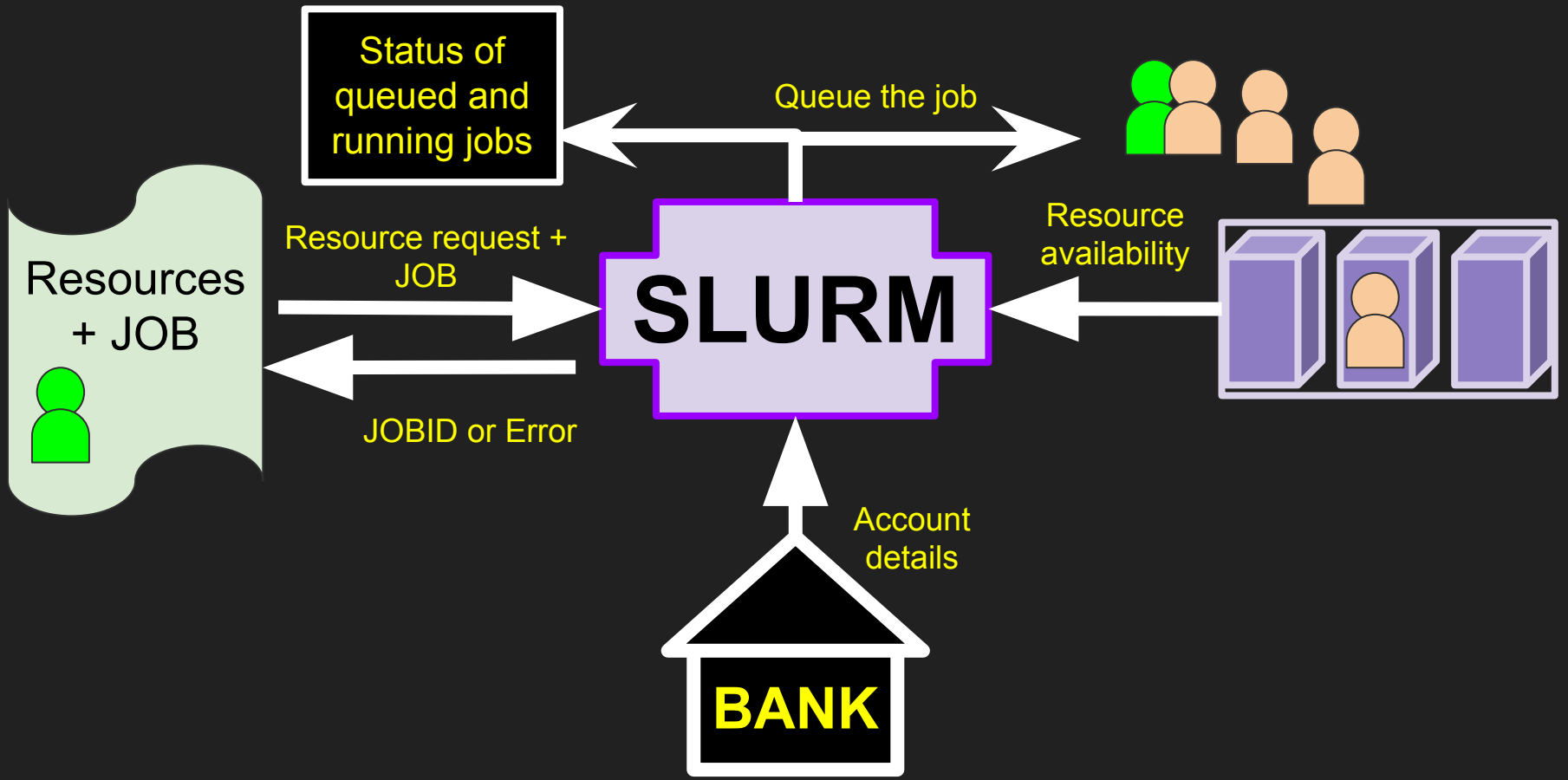
# Job submission to Abel



# Job submission to Abel



# Job submission to Abel



# Computing on Abel

- Manage UIO and external users and projects (Notur)
- Submit a job to the queuing system
- Software that executes jobs on available resources on the cluster.
- 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 (order is important)

# Job script

```
#!/bin/bash

#SBATCH --job-name=INF5380_hello
#SBATCH --account=xxx
#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=INF5380_hello
#SBATCH --account=xxx
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=512M
#SBATCH --time=00:01:00

source /cluster/bin/jobsetup

python hello.py
```

Resources

Setup

Job

# Understand resources

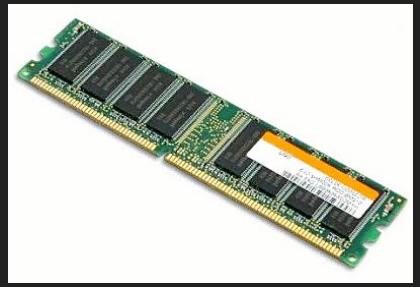
- Needs a project account.
  - UIO students and employees assigned to group uio by default. But this has a long queue
  - Recommended to apply for a project account from nortur  (https://www.notur.no/who-can-get-access)
  - Use the following command to find out the projects that you have access to

```
> projects
```

# Understand resources

- If program can not or would not use parallel processing, no point requesting more than one.
- 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**

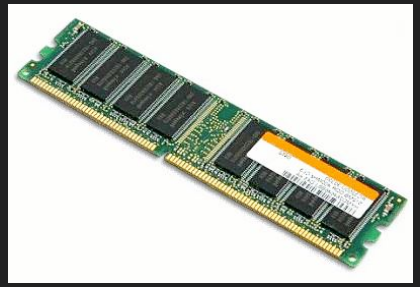


**Samsung DDR3**
**1600 MHz**

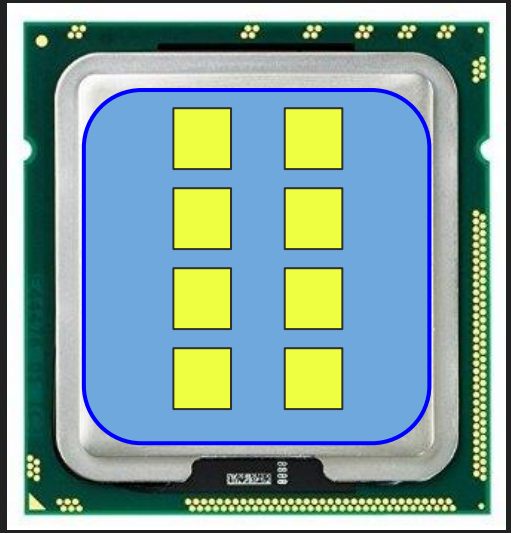

**Intel® Xeon® Processor**
**E5-2670**
**Octa core (8)**

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

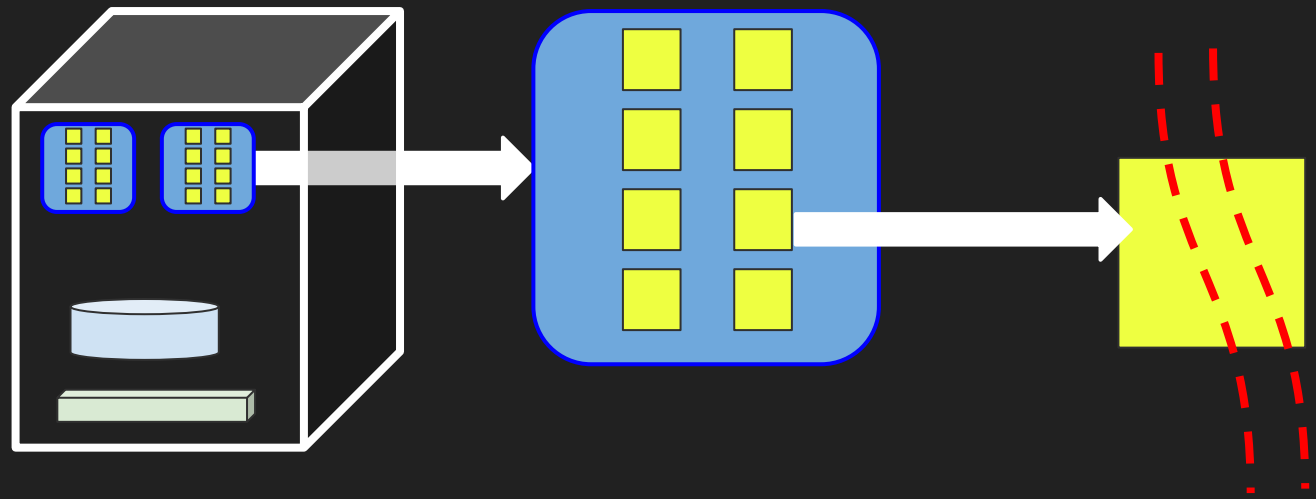


Samsung DDR3
1600 MHz



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

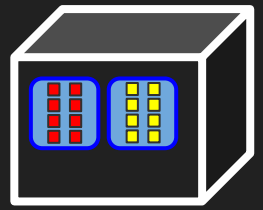
Datasheet: http://ark.intel.com/products/64595/Intel-Xeon-Processor-E5-2670-20M-Cache-2_60-GHz-8_00-GTs-Intel-QPI



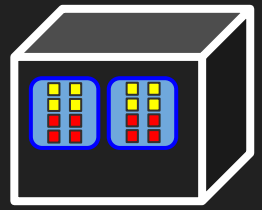
- Supermicro X9DRT compute node
- 2 X Intel Xeon E5-2670 (Sandy Bridge) based running at 2.6 GHz,
- 16 physical compute cores total per node.
- (32 threads in total)
- Each node have 64 GiBytes of Samsung DDR3 memory

# #SBATCH --ntasks=8

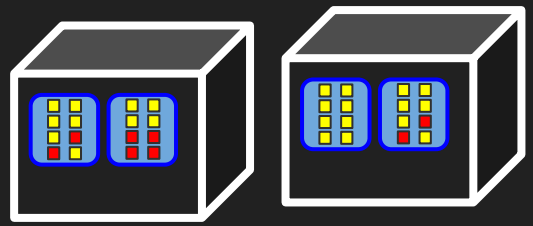
Possible combinations



(8 + 0)

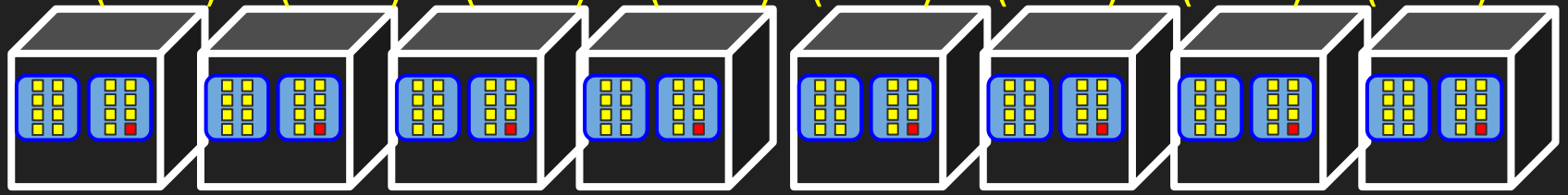


(4 + 4)

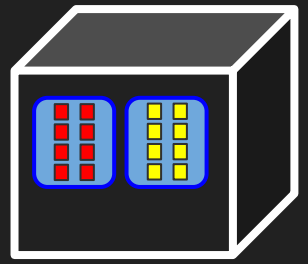


(2+ 4) + (0+2)

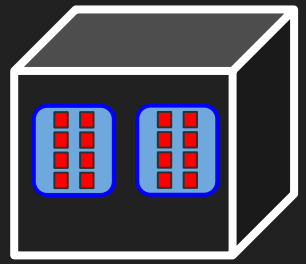
(1 + 0) + (1 + 0) + (1 + 0) + (1 + 0) +(1 + 0) + (1 + 0) + (1 + 0) + (1 + 0)



#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
All tasks will share memory
mpirun -n 8 myprg



#SBATCH --ntasks=2
#SBATCH --cpus-per-task=8
8 X 2 =16
myprg
    -- myprg can run up to 16 processes



# Use of the SCRATCH area

```
#!/bin/sh
#SBATCH --job-name=Job_1
…..
source /cluster/bin/jobsetup

## Copy files to work directory:
cp $SUBMITDIR/YourData $SCRATCH

## Mark outfiles for automatic copying to $SUBMITDIR:
chkfile YourOutput

## Run command
cd $SCRATCH
executable YourData > YourOutput
```

# 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 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

# Sample job script

```
#!/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 “Finished”
```

# Join the queue

```
> sbatch hello.slurm

> scontrol show job <JOBID>

> squeue -u <username>
```

# 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
- set | grep SLURM - account balance

# 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
- TASK_ID – task number (for arrayrun-jobs)

# 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

# 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.

# Check Variables

```
#!/bin/bash

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

# What are the available software ?

- On Abel issue the following command to list modules
  - module avail
  - module avail blast
- Currently loaded modules
  - module list
- Clear modules
  - module purge
- http://www.uio.no/english/services/it/research/hpc/abel/help/software/

# 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 - worker script

```
#!/bin/bash

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

source /cluster/bin/jobsetup
set -o errexit

cp  $SUBMITDIR/input.txt $SCRATCH
RESULT=$TASK_ID+".txt"
chkfile $RESULT

python array_test.py input.txt $RESULT $TASK_ID
```

# Arrayrun - submit script

```
#!/bin/bash

#SBATCH --job-name=RCS1115_arraytest
#SBATCH --account=staff
#SBATCH --time=00:01:05
#SBATCH --mem-per-cpu=512M
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1

source /cluster/bin/jobsetup
set -o errexit

arrayrun 1-10 array_test.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"
```