SLURM basics and programatic executions

O. Sotolongo-Grau

ACE Alzheimer Center Barcelona. Spain

Introduction

SLURM commands

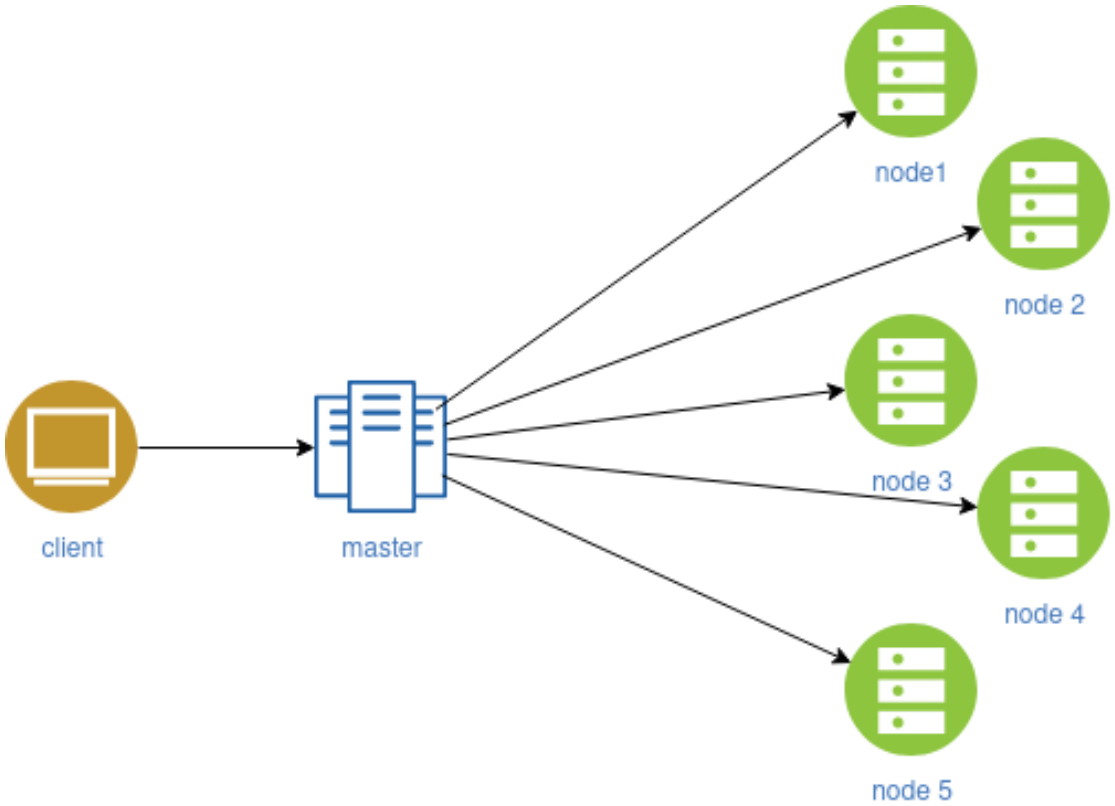
- srun
- salloc
- sbatch
- squeue
- sinfo
- scancel

SLURM commands

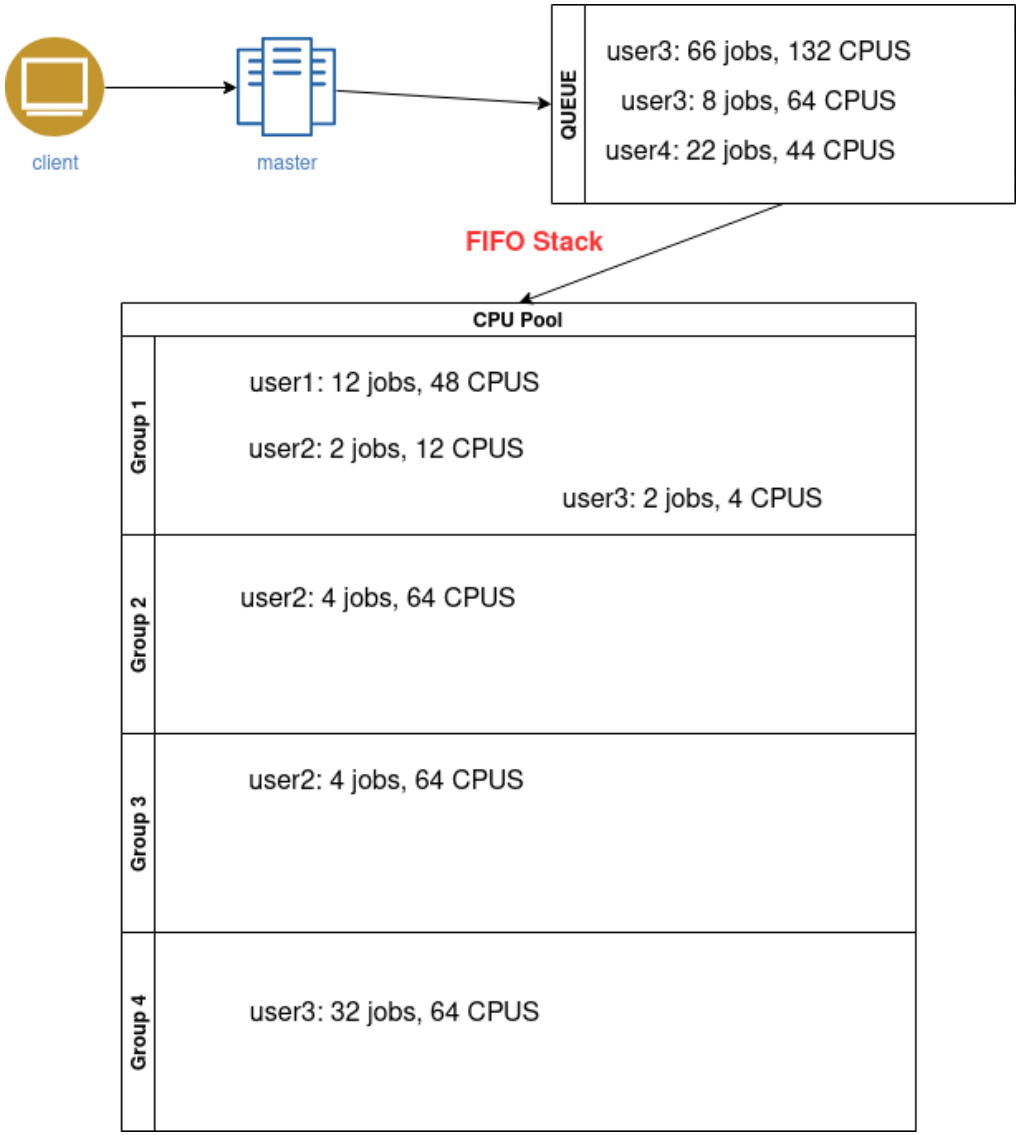
- srun
- salloc
- sbatch
- squeue
- sinfo
- scancel

How does SLURM works?

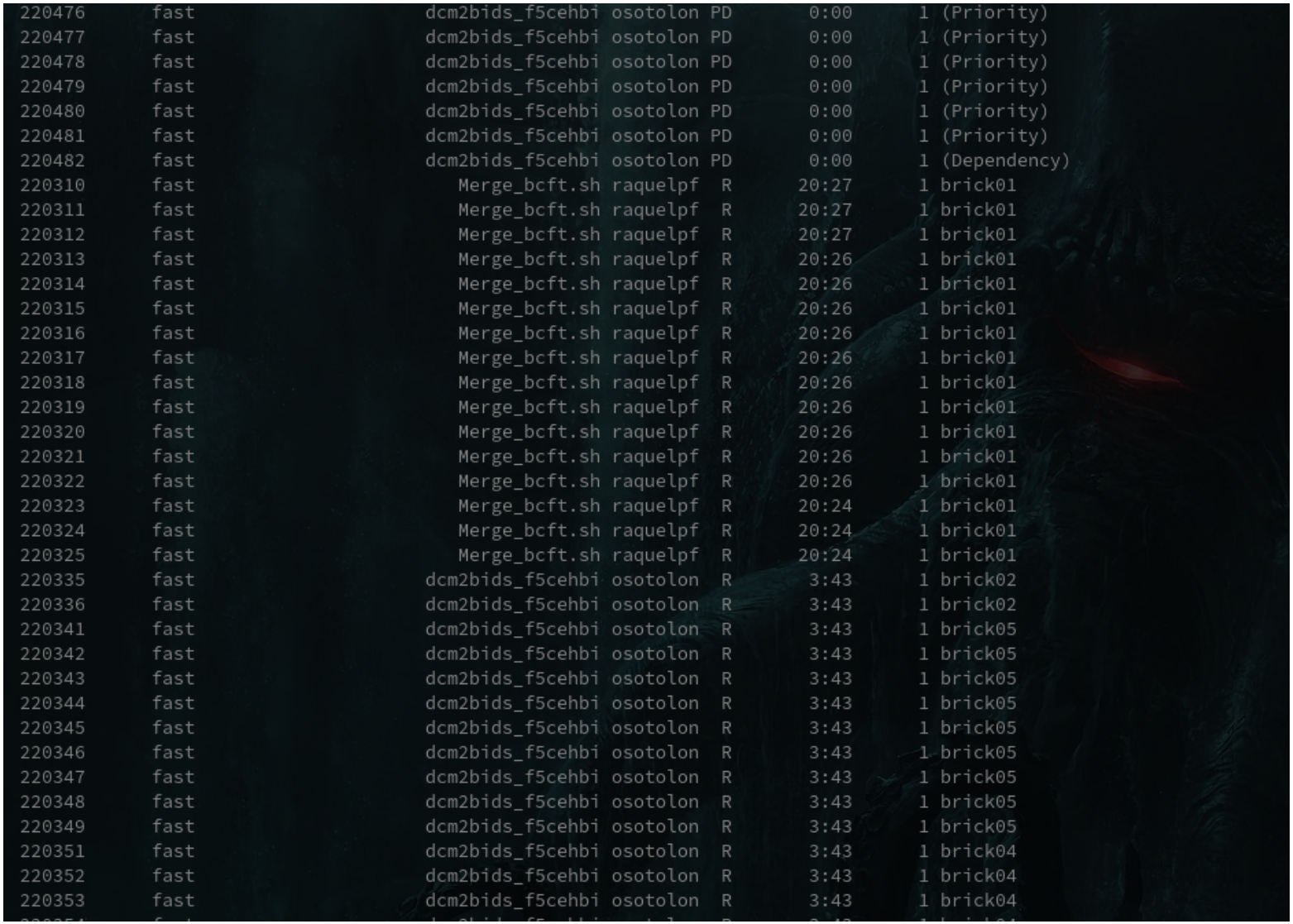
Service structure



Workload manager abstraction



Workload manager working



```
220476      fast                dcm2bids_f5cehbi osotolon PD       0:00      1 (Priority)
220477      fast                dcm2bids_f5cehbi osotolon PD       0:00      1 (Priority)
220478      fast                dcm2bids_f5cehbi osotolon PD       0:00      1 (Priority)
220479      fast                dcm2bids_f5cehbi osotolon PD       0:00      1 (Priority)
220480      fast                dcm2bids_f5cehbi osotolon PD       0:00      1 (Priority)
220481      fast                dcm2bids_f5cehbi osotolon PD       0:00      1 (Priority)
220482      fast                dcm2bids_f5cehbi osotolon PD       0:00      1 (Dependency)
220310      fast                   Merge_bcft.sh raquelpf  R      20:27      1 brick01
220311      fast                   Merge_bcft.sh raquelpf  R      20:27      1 brick01
220312      fast                   Merge_bcft.sh raquelpf  R      20:27      1 brick01
220313      fast                   Merge_bcft.sh raquelpf  R      20:26      1 brick01
220314      fast                   Merge_bcft.sh raquelpf  R      20:26      1 brick01
220315      fast                   Merge_bcft.sh raquelpf  R      20:26      1 brick01
220316      fast                   Merge_bcft.sh raquelpf  R      20:26      1 brick01
220317      fast                   Merge_bcft.sh raquelpf  R      20:26      1 brick01
220318      fast                   Merge_bcft.sh raquelpf  R      20:26      1 brick01
220319      fast                   Merge_bcft.sh raquelpf  R      20:26      1 brick01
220320      fast                   Merge_bcft.sh raquelpf  R      20:26      1 brick01
220321      fast                   Merge_bcft.sh raquelpf  R      20:26      1 brick01
220322      fast                   Merge_bcft.sh raquelpf  R      20:26      1 brick01
220323      fast                   Merge_bcft.sh raquelpf  R      20:24      1 brick01
220324      fast                   Merge_bcft.sh raquelpf  R      20:24      1 brick01
220325      fast                   Merge_bcft.sh raquelpf  R      20:24      1 brick01
220335      fast                dcm2bids_f5cehbi osotolon  R       3:43      1 brick02
220336      fast                dcm2bids_f5cehbi osotolon  R       3:43      1 brick02
220341      fast                dcm2bids_f5cehbi osotolon  R       3:43      1 brick05
220342      fast                dcm2bids_f5cehbi osotolon  R       3:43      1 brick05
220343      fast                dcm2bids_f5cehbi osotolon  R       3:43      1 brick05
220344      fast                dcm2bids_f5cehbi osotolon  R       3:43      1 brick05
220345      fast                dcm2bids_f5cehbi osotolon  R       3:43      1 brick05
220346      fast                dcm2bids_f5cehbi osotolon  R       3:43      1 brick05
220347      fast                dcm2bids_f5cehbi osotolon  R       3:43      1 brick05
220348      fast                dcm2bids_f5cehbi osotolon  R       3:43      1 brick05
220349      fast                dcm2bids_f5cehbi osotolon  R       3:43      1 brick05
220351      fast                dcm2bids_f5cehbi osotolon  R       3:43      1 brick04
220352      fast                dcm2bids_f5cehbi osotolon  R       3:43      1 brick04
220353      fast                dcm2bids_f5cehbi osotolon  R       3:43      1 brick04
```

Sending jobs

srun


```
[osotolongo@brick03 osotolongo]$ srun hostname
brick01
```

```
[osotolongo@brick03 osotolongo]$ srun -w brick05 hostname
brick05
```

```
[osotolongo@brick03 osotolongo]$ srun -n 4 -c 32 hostname
brick01
brick01
brick02
brick02
```

See srun manpage

srun

```
[osotolongo@brick03 osotolongo]$ srun hostname
brick01

[osotolongo@brick03 osotolongo]$ srun -w brick05 hostname
brick05

[osotolongo@brick03 osotolongo]$ srun -n 4 -c 32 hostname
brick01
brick01
brick02
brick02
```

See srun manpage

srun

```
[osotolongo@brick03 osotolongo]$ srun hostname
brick01

[osotolongo@brick03 osotolongo]$ srun -w brick05 hostname
brick05

[osotolongo@brick03 osotolongo]$ srun -n 4 -c 32 hostname
brick01
brick01
brick02
brick02
```

See srun manpage

salloc


```
[osotolongo@brick03 ~]$ salloc --nodes=3 sh
salloc: Granted job allocation 220218
sh-4.2$ srun --label hostname
1: brick04
0: brick02
2: brick05
sh-4.2$ exit
exit
salloc: Relinquishing job allocation 220218
```

See salloc manpage

sbatch


```
#!/bin/bash
#SBATCH -J test
#SBATCH --mail-type=FAIL,TIME_LIMIT,STAGE_OUT
#SBATCH --mail-user=osotolongo
#SBATCH -o output-%j.out
hostname
```

```
[osotolongo@brick03 cluster]$ sbatch myscript.sh
Submitted batch job 219989
```

```
[osotolongo@brick03 cluster]$ cat output-219989.out
brick01
```

See sbatch manpage

sbatch


```
#!/bin/bash
#SBATCH -J test
#SBATCH --mail-type=FAIL,TIME_LIMIT,STAGE_OUT
#SBATCH --mail-user=osotolongo
#SBATCH -o output-%j.out
hostname
```

```
[osotolongo@brick03 cluster]$ sbatch myscript.sh
Submitted batch job 219989
```

```
[osotolongo@brick03 cluster]$ cat output-219989.out
brick01
```

See sbatch manpage

sbatch


```
#!/bin/bash
#SBATCH -J test
#SBATCH --mail-type=FAIL,TIME_LIMIT,STAGE_OUT
#SBATCH --mail-user=osotolongo
#SBATCH -o output-%j.out
hostname
```

```
[osotolongo@brick03 cluster]$ sbatch myscript.sh
Submitted batch job 219989
```

```
[osotolongo@brick03 cluster]$ cat output-219989.out
brick01
```

See sbatch manpage

Some SBATCH directives

- CPU allocation
  - **–cpus-per-task**=<*ncpus*>
  - **–ntasks-per-node**=<*ntasks*>
  - **–mem-per-cpu**=<*size*>[*units*]

CPU allocation example

```
#SBATCH -c 32
#SBATCH --mem-per-cpu=4G
```

- GPU allocation
  - **–partition**=<*partition*>
  - **–gres**=<*name*[[:*type*]:*count*]>

GPU allocation example

```
#SBATCH -p cuda
#SBATCH --gres=gpu:1
```

Some SBATCH directives

- CPU allocation
  - **–cpus-per-task**=<*ncpus*>
  - **–ntasks-per-node**=<*ntasks*>
  - **–mem-per-cpu**=<*size*>[*units*]

CPU allocation example

```
#SBATCH -c 32
#SBATCH --mem-per-cpu=4G
```

- GPU allocation
  - **–partition**=<*partition*>
  - **–gres**=<*name*[[:*type*]:*count*]>

GPU allocation example

```
#SBATCH -p cuda
#SBATCH --gres=gpu:1
```

How to be nice

- SLURM *nice*
  - Adjust the priority in the **queue**
  - $>$ *nice* $\Longrightarrow$ $<$ priority, between 0 and 2147483645 (default 0)

SBATCH nice

```
#SBATCH --nice=1000
```

- System *nice*
  - Adjust the priority of resource consumption in the system
  - $>$ *nice* $\Longrightarrow$ $<$ priority, between -20 and 19 (default 0)

UNIX nice

```
#!/bin/bash
#SBATCH blah blah blah
nice -n10 gzip -d my_file.zip
```

How to be nice

- SLURM *nice*
  - Adjust the priority in the **queue**
  - $>$ *nice* $\Longrightarrow$ $<$ priority, between 0 and 2147483645 (default 0)

SBATCH nice

```
#SBATCH --nice=1000
```

- System *nice*
  - Adjust the priority of resource consumption in the system
  - $>$ *nice* $\Longrightarrow$ $<$ priority, between -20 and 19 (default 0)

UNIX nice

```
#!/bin/bash
#SBATCH blah blah blah
nice -n10 gzip -d my_file.zip
```

more SLURM commands

Remember also to check:

- squeue
- sinfo
- scancel

Parallel jobs

sbatch generator


```
1   #!/bin/bash
2   count=0
3   while [ $count -lt 22 ]
4   do
5   sbatch << EOF
6   #!/bin/bash
7   #SBATCH -J test
8   #SBATCH --mail-user=osotolongo
9   #SBATCH -o output-%j.out
10  gunzip chr$((count+1)).info.gz
11  EOF
12  ((count++))
13  done
```

Dependencies

- singleton $\rightarrow$ run after all jobs with same *name* and *user*
- afterok:*job_id* $\rightarrow$ run after job with a given *job_id* ends successfuly
- after, afterany, afternotok, ...

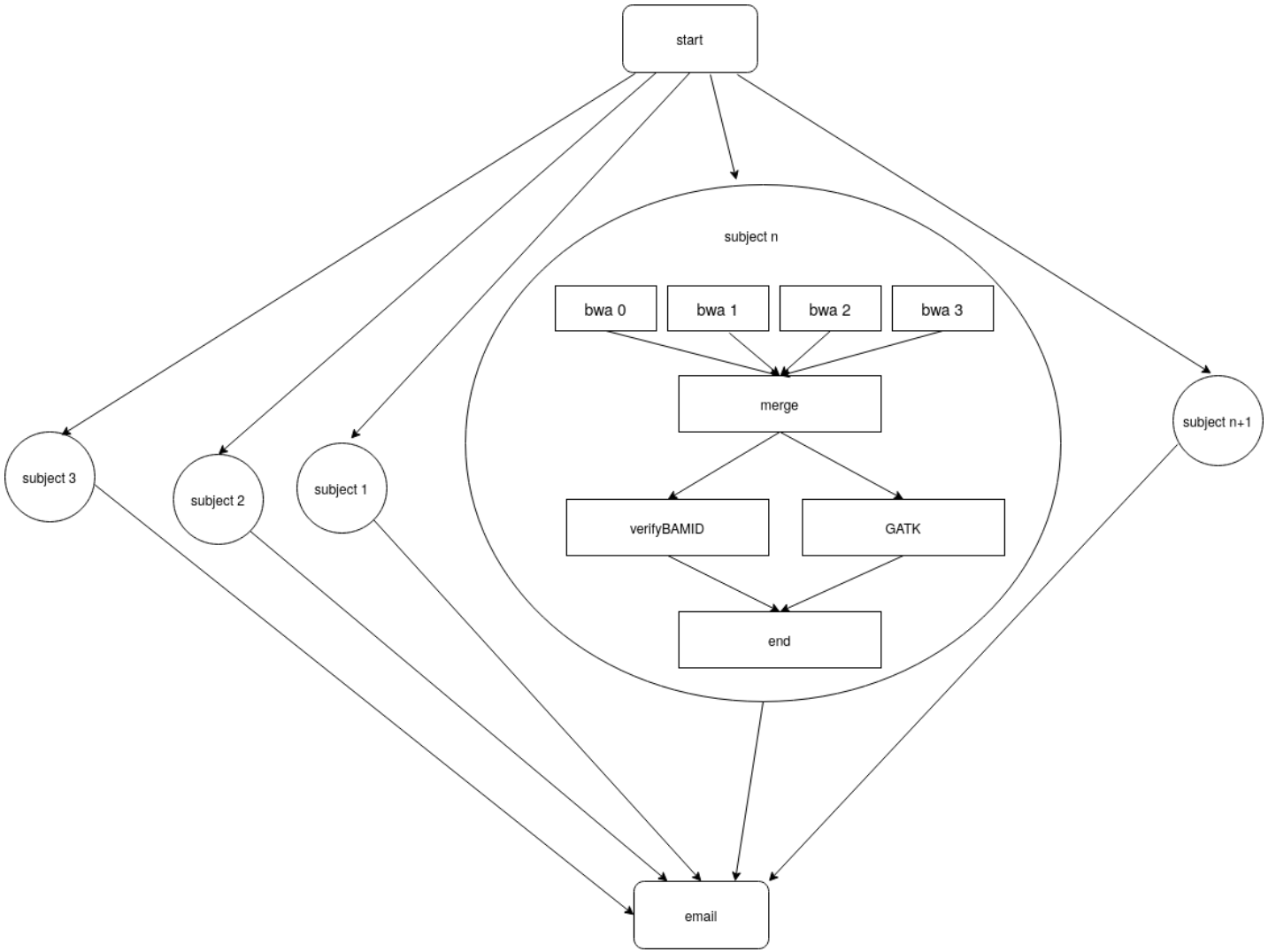
singleton


```
1  #!/bin/bash
2  count=0
3  while [ $count -lt 22 ]
4  do
5  sbatch << EOF
6  #!/bin/bash
7  #SBATCH -J test
8  #SBATCH --mail-type=FAIL,TIME_LIMIT,STAGE_OUT
9  #SBATCH --mail-user=osotolongo
10 #SBATCH -o output-%j.out
11 gunzip chr$((count+1)).info.gz
12 EOF
13 ((count++))
14 done
15 sbatch -d singleton << EOF
16 #!/bin/bash
17 #SBATCH -J test
18 #SBATCH --mail-type=END
19 #SBATCH --mail-user=osotolongo
20 #SBATCH -o output-%j.out
21 :
22 EOF
```

```
#!/bin/bash
count=0
while [ $count -lt 22 ]
do
jobid=$(sbatch --parsable << EOF
#!/bin/bash
#SBATCH -J test
#SBATCH --mail-type=FAIL,TIME_LIMIT,STAGE_OUT
#SBATCH --mail-user=osotolongo
#SBATCH -o output-%j.out
gunzip chr$((count+1)).info.gz
EOF)
sbatch --dependency=afterok:${jobid} << EOF
#!/bin/bash
#SBATCH -J test2
#SBATCH --mail-type=FAIL,TIME_LIMIT,STAGE_OUT
#SBATCH --mail-user=osotolongo
#SBATCH -o output-%j.out
awk '{ if ($7 >0.3) print $0 }' chr$((count+1)).info > chr$((count+1)).info.selected
EOF
((count++))
done
```

```
1  #!/bin/bash
2  count=0
3  while [ $count -lt 22 ]
4  do
5  sbatch << EOF
6  #!/bin/bash
7  #SBATCH -J test
8  #SBATCH --mail-type=FAIL,TIME_LIMIT,STAGE_OUT
9  #SBATCH --mail-user=osotolongo
10 #SBATCH -o output-%j.out
11 gunzip chr$((count+1)).info.gz
12 awk '{ if ($7 >0.3) print $0 }' chr$((count+1)).info > chr$((count+1)).info.selected
13 EOF
```

but sometimes it's not so easy



into Python

Why Python?

- If I want to do more complicate things, better I will use another language than *BASH*
- *Python* seems to be a good alternative
  - low learning curve
  - great for scripting
  - popular (lot of libraries)

What Python version?

- Python 2 and Python 3 are installed at every node

```
[osotolongo@brick03 ~]$ python3 --version
Python 3.6.8
[osotolongo@brick03 ~]$ python2 --version
Python 2.7.5
```

- But default behavior need to be settled

```
alias python=python3
```

- Easiest way is to declare it at the beginning of the script

```
#!/usr/bin/python3
```

loop approach


```
1   #!/usr/bin/python3
2   import os
3   workdir = 'slurm'
4   time = '3:0:0'
5   cpus = 4
6   if not os.path.isdir(workdir): os.mkdir(workdir)
7   for count in range(22)
8       # define the content of sbatch script
9       content = '#!/bin/bash\n'
10      content += '#SBATCH -J my-job\n'
11      content += '#SBATCH -c '+str(cpus)+'\n'
12      content += '#SBATCH --mem-per-cpu=4G\n'
13      content += '#SBATCH --time='+time+'\n'
14      content += '#SBATCH --mail-type=FAIL,TIME_LIMIT,STAGE_OUT\n'
15      content += '#SBATCH --mail-user='+os.environ.get('USER')+'\n'
16      content += '#SBATCH -o '+workdir+'/my-job-out-%j\n'
17      content += 'gunzip chr'+str(count+1)'+info.gz\n'
18      # write sbatch script
19      ofile = wordir+'/script_{:04d}'.format(count+1)+'.sh'
20      exf = open(ofile, 'w')
21      exf.write(content)
22      exf.close()
23      # and execute it
24      os.system('sbatch '+ofile)
```

slurm-modpy

- Let's use slurm-modpy
- Download *slurm.py* and put it in your working directory
- You are ready to go!

import *send_sbatch*() function

```
from slurm import send_sbatch
```

define a dictionary with *job* properties

```
cdata = {'time':'4:0:0', 'cpus':4, 'job_name':'test', 'command':'hostname'}
```

run it!

```
jobid = send_sbatch(cdata)
```

looping with slurm-modpy


```
1   #!/usr/bin/python3
2   import os
3   from slurm import send_sbatch
4   workdir = 'slurm'
5   if not os.path.isdir(workdir): os.mkdir(workdir)
6   cdata = {'time':'4:0:0', 'cpus':4, 'job_name':'job_one'}
7   for count in range(22)
8       # define the content of sbatch script
9       cdata['filename'] = workdir+'/script'+str(count+1)+'.sh'
10      cdata['output'] = workdir+'/output'+str(count+1)+'.out'
11      cdata['command'] = 'gunzip chr'+str(count+1)+'.info.gz'
12      # run it
13      send_sbatch(cdata)
```

singleton with slurm-modpy


```
1   #!/usr/bin/python3
2   import os
3   from slurm import send_sbatch
4   workdir = 'slurm'
5   if not os.path.isdir(workdir): os.mkdir(workdir)
6   cdata = {'time':'4:0:0', 'cpus':4, 'job_name':'job_one'}
7   for count in range(22)
8       # define the content of sbatch script
9       cdata['filename'] = workdir+'/script'+str(count+1)+'.sh'
10      cdata['output'] = workdir+'/output'+str(count+1)+'.out'
11      cdata['command'] = 'gunzip chr'+str(count+1)+'.info.gz'
12      # run it
13      send_sbatch(cdata)
14  # define and send the warning job
15  wdata = {'job_name':'job_one', 'filename':workdir+'/warning_end.sh', 'dependency':'singleton'}
16  send_sbatch(wdata)
```

afterok with slurm-modpy


```
1   #!/usr/bin/python3
2   import os
3   from slurm import send_sbatch
4   workdir = 'slurm'
5   if not os.path.isdir(workdir): os.mkdir(workdir)
6   cdata = {'time':'4:0:0', 'cpus':4, 'job_name':'job_one'}
7   for count in range(22)
8       # define the content of sbatch script 1
9       cdata['filename'] = workdir+'/script'+str(count+1)+'.sh'
10      cdata['output'] = workdir+'/output'+str(count+1)+'.out'
11      cdata['command'] = 'gunzip chr'+str(count+1)+'.info.gz'
12      cdata.pop('dependency', None)
13      # run it
14      jobid = send_sbatch(cdata)
15      # define the content of sbatch script 2
16      cdata['filename'] = workdir+'/script2'+str(count+1)+'.sh'
17      cdata['output'] = workdir+'/output2'+str(count+1)+'.out'
18      cdata['command'] = 'awk \'{ if ($7 >0.3) print $0 }\' chr'+str(count+1)+'.info'
19      cdata['command'] += ' > chr'+str(count+1)+'.info.selected'
20      cdata['dependency'] = 'afterok:'+str(jobid)
21      # run it
22      send_sbatch(cdata)
23  # define and send the warning job
24  wdata = {'job_name':'job_one', 'filename':workdir+'/warning_end.sh', 'dependency':'singleton'}
25  send_sbatch(wdata)
```

fully functional example


```
1   #!/usr/bin/python3
2   import sys
3   import os
4   from slurm import send_sbatch
5
6   jtime = '3:0:0'
7   cpus = 4
8   mem_per_cpu = '4G'
9   wdir = 'slurm'
10
11  ifile = str(sys.argv[1])
12
13  if not os.path.isdir(wdir): os.mkdir(wdir)
14  count = 0
15  ljob = ({'job_name':ifile, 'cpus':cpus, 'mem_per_cpu':mem_per_cpu,
16  'time':jtime, 'output':wdir+'/'+ifile+'order-%j', 'mailtype':'FAIL,TIME_LIMIT,STAGE_OUT'})
17  with open(ifile, 'r') as orf:
18    for line in orf:
19      count+=1
20      ljob['filename'] = wdir+'/sorder_{:04d}'.format(count)+'.sh'
21      ljob['command'] = line
22      send_sbatch(ljob)
23  ejob = ({'job_name':ifile, 'output':wdir+'/'+ifile+'end-%j',
24  'filename':wdir+'/sorder_end.sh', 'dependency':'singleton'})
25  send_sbatch(ejob)
```

Get it at github

adding commands at the end


```python
#!/usr/bin/python3
import sys
import os
from slurm import send_sbatch
jtime = '3:0:0'
cpus = 4
mem_per_cpu = '4G'
wdir = 'slurm'
ifile = str(sys.argv[1])
sfile = str(sys.argv[2]) # this file contains a list of orders to be executed at the end (in serial mode)
if not os.path.isdir(wdir): os.mkdir(wdir)
count = 0
ljob = ({'job_name':ifile, 'cpus':cpus, 'mem_per_cpu':mem_per_cpu,
'time':jtime, 'output':wdir+'/'+ifile+'order-%j', 'mailtype':'FAIL,TIME_LIMIT,STAGE_OUT'})
with open(ifile, 'r') as orf:
  for line in orf:
    count+=1
    ljob['filename'] = wdir+'/sorder_{:04d}'.format(count)+'.sh'
    ljob['command'] = line
    send_sbatch(ljob)
ejob = ({'job_name':ifile, 'output':wdir+'/'+ifile+'end-%j',
'filename':wdir+'/sorder_end.sh', 'dependency':'singleton',
'mailtype':'FAIL,TIME_LIMIT,STAGE_OUT,END', 'cpus':8, 'mem_per_cpu':mem_per_cpu})
sc = open(sfile, 'r') # Here I got the serial orders and execute them
ejob['command'] = sc.read()
sc.close()
send_sbatch(ejob)
```

The end is near!

What we have so far?

- We can use *srun* and *sbatch* to send jobs to the schedule manager
- Simple tasks are easily managed in *BASH* scripts
- For more complicated tasks is better to use some kind of interpreted language like *Python* (or *Perl*, *Ruby*, *PHP*, whatever makes you feel more comfortable)
- Take a look at slurm-modpy and the included example scripts
- Looking for more? Try simple_slurm!

What we have so far?

- We can use *srun* and *sbatch* to send jobs to the schedule manager
- Simple tasks are easily managed in *BASH* scripts
- For more complicated tasks is better to use some kind of interpreted language like *Python* (or *Perl*, *Ruby*, *PHP*, whatever makes you feel more comfortable)
- Take a look at slurm-modpy and the included example scripts
- **Looking for more? Try simple_slurm!**

Thanks