High-Performance Computing with RELION

David Hoover, PhD

HPC @ NIH

Biowulf Seminar Series

Sagar Chittori LCB, NCI

Cryo-EM studies of glutamate receptors and nucleosomes

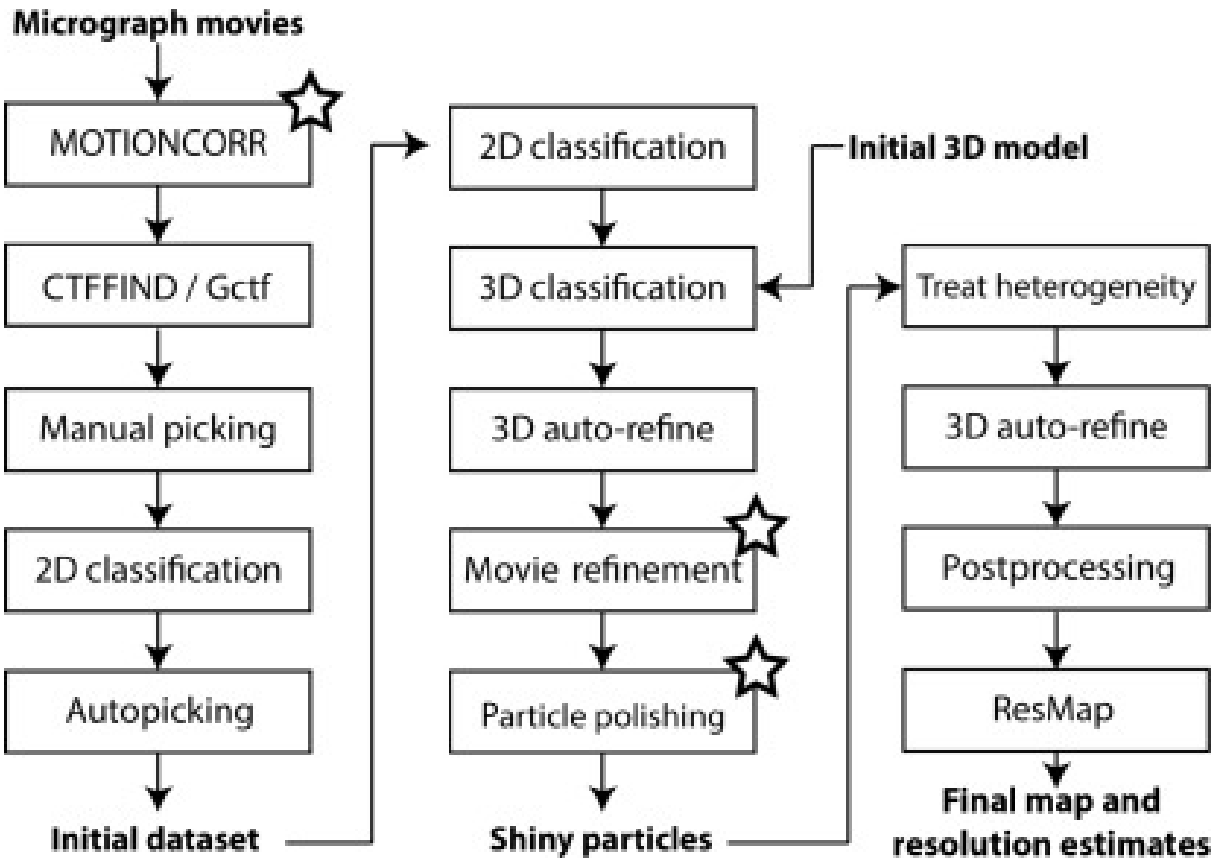
Wed May 09, 2018, 11 am – noon
Bldg 50, Rm 1227 (next to the coffeshop)

TOMORROW ONLY!

Brief Overview of RELION

- *Purify macromolecules and freeze onto platform*
- *Collect TEM images (2D projections)*
- Clean up images (defocus, tilt, motion, blur)
- Isolate individual particle projection images
- Classify particles
- Predict original orientation, maximum likelihood
- Refine prediction with “smooth” maps
- Polish particles to improve signal-to-noise

Brief Overview of RELION



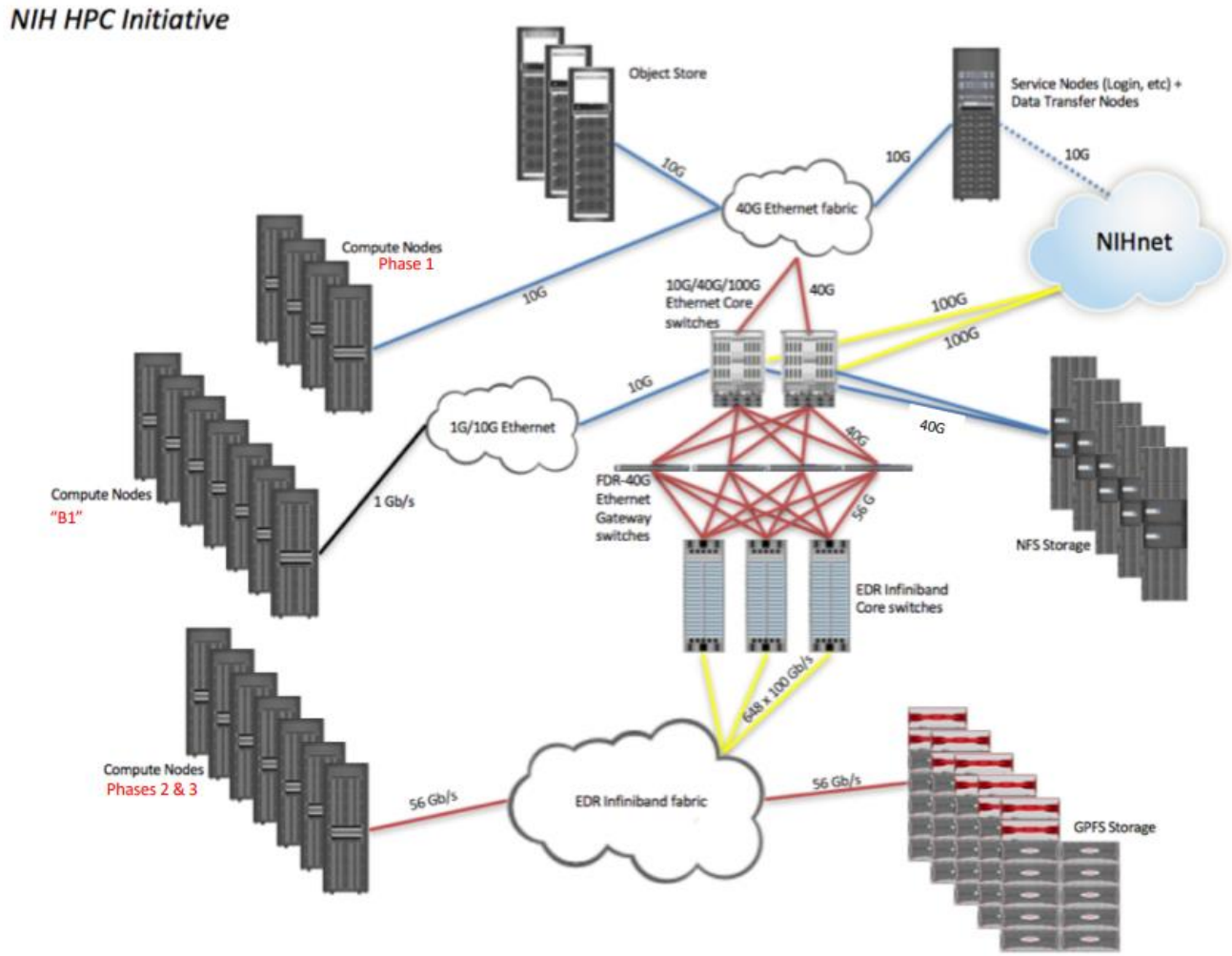
Scheres, S.H.W. *Meth. Enzy. 579*, 2016

Brief Overview of RELION

- Many steps can be parallelized
- A few steps can be accelerated using GPUs

Cluster and Hardware Overview

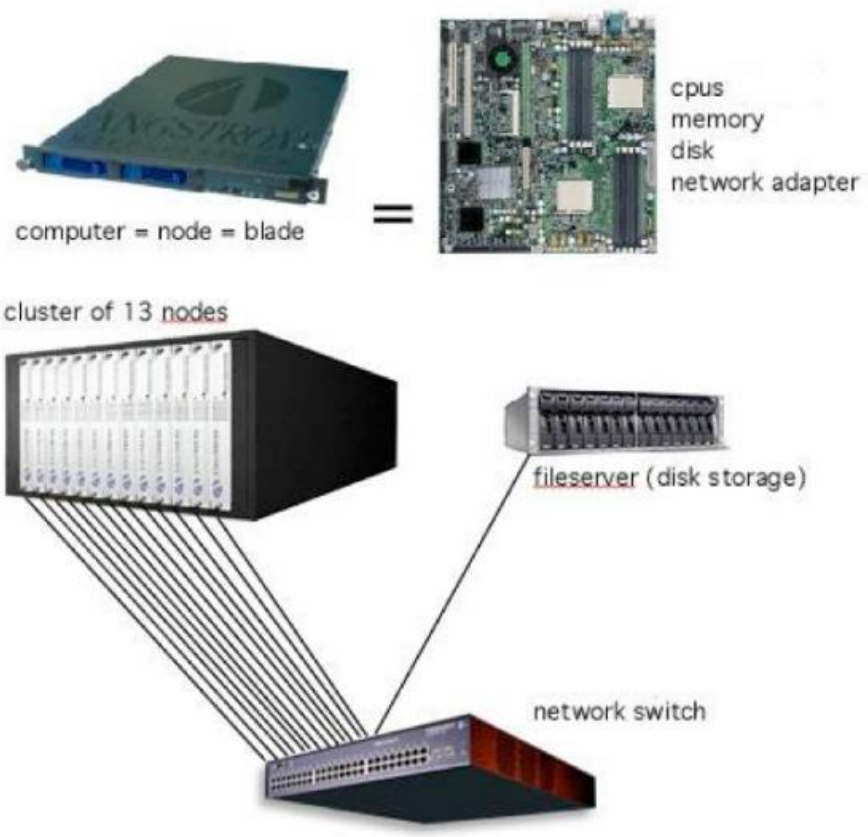
Biowulf Cluster Overview



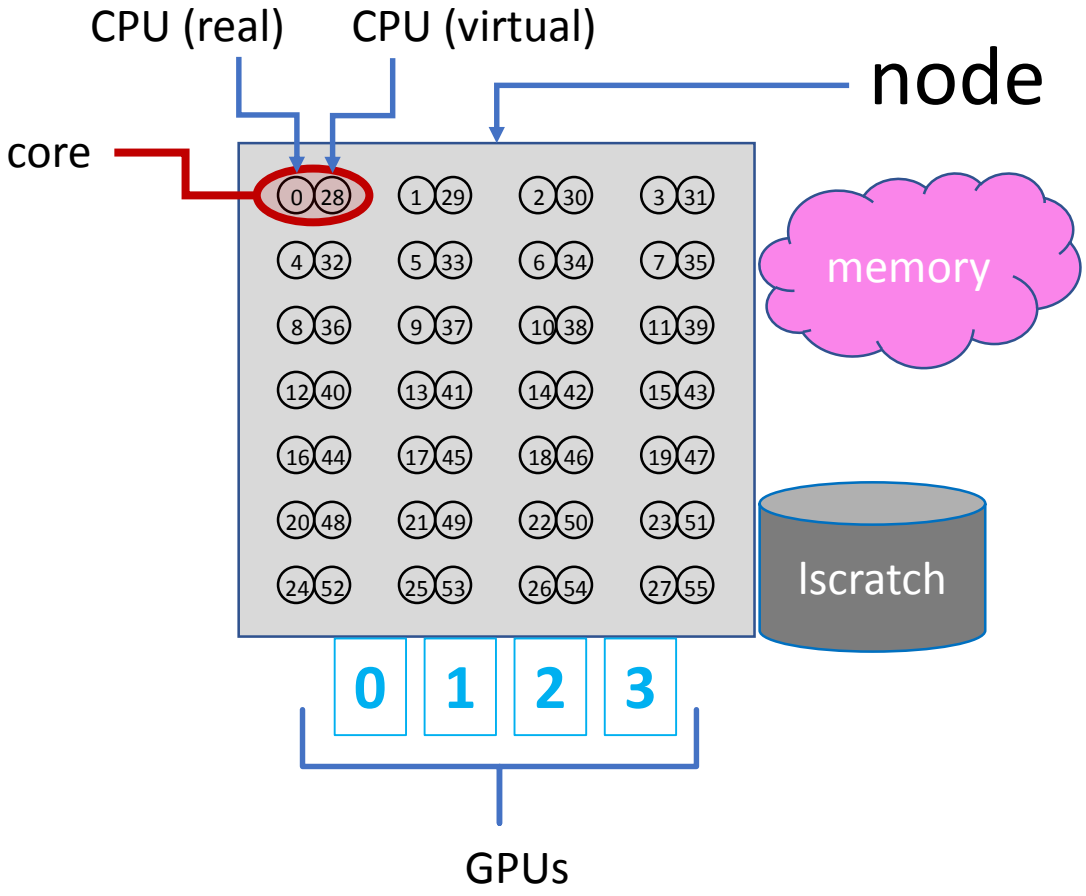
biowulf
AT THE NIH

NIH Center for Information Technology

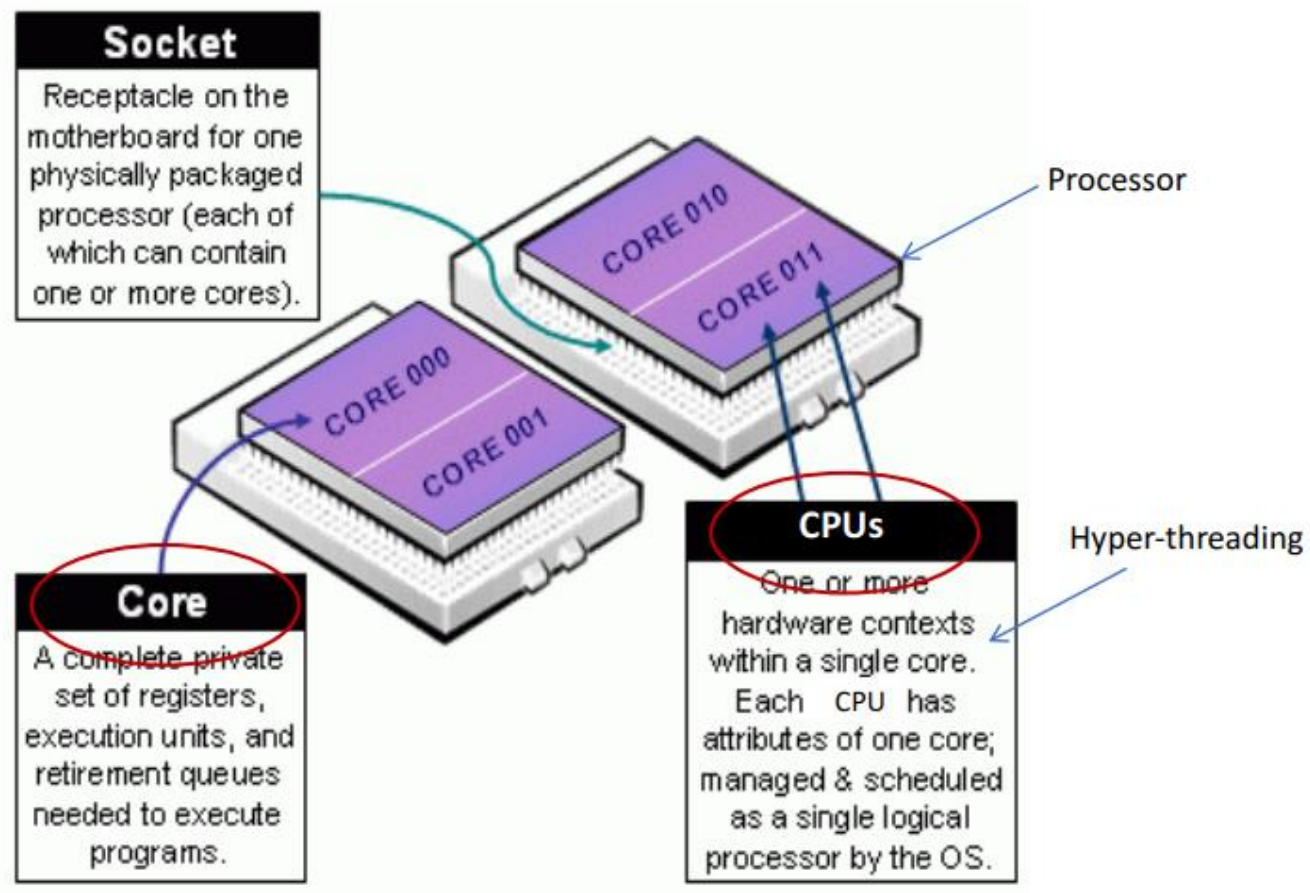
Biowulf Cluster Overview



Node Anatomy



Processor Hardware



Parallelization on the Biowulf Cluster

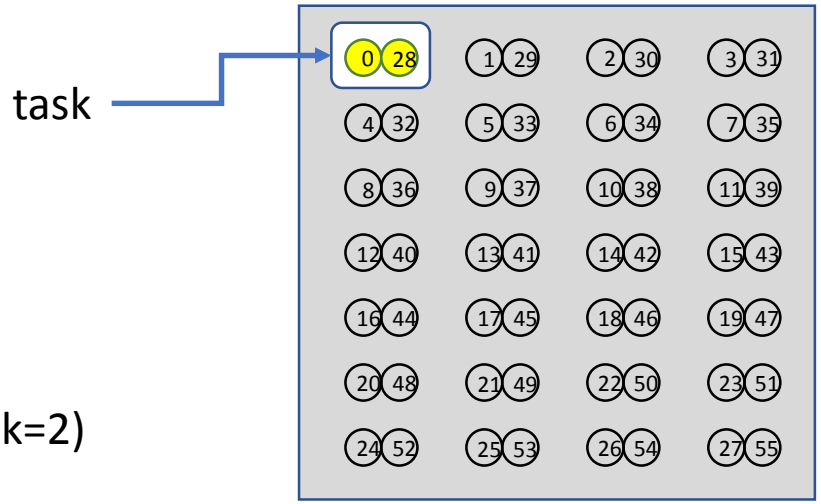
Parallelization on Biowulf using Slurm

- Allocation of resources
- Distribution of resources
- Distribution of tasks on the allocated and distributed resources

Multi-user Time-sharing Batch System

Single-Threaded Job

Slurm allocates in units of **cores**, not CPUs



(--ntasks=1)
(--cpus-per-task=2)

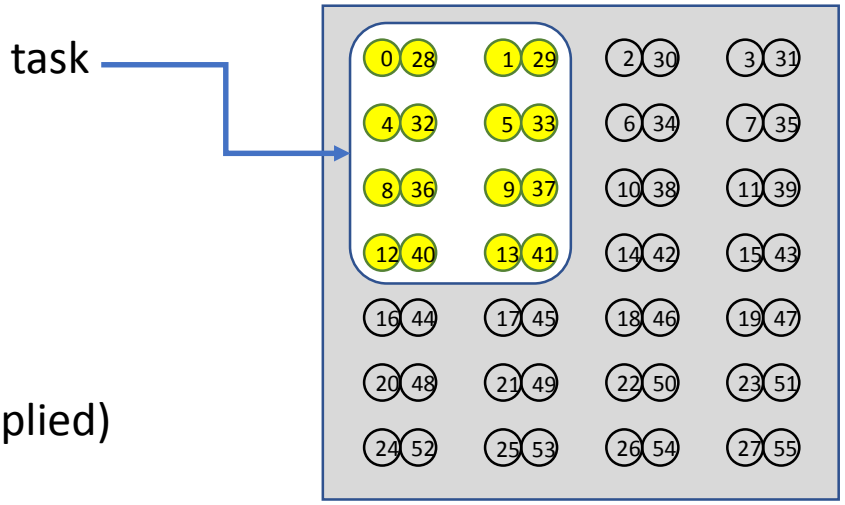
sbatch

executable arg arg arg …

sinteractive

executable arg arg arg …

Multi-threaded Job



(--ntasks=1 implied)

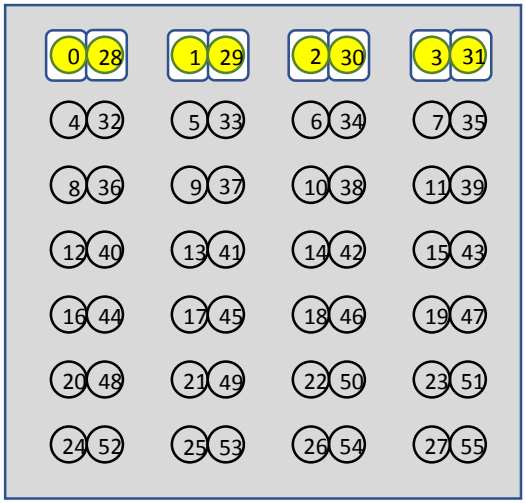
```
sbatch --cpus-per-task=16
```

```
executable -p $SLURM_CPUS_PER_TASK arg arg arg …
```

```
sinteractive --cpus-per-task=16
```

```
executable -p $SLURM_CPUS_PER_TASK arg arg arg …
```

MPI Job: single node



sbatch --ntasks=8 --nodes=1

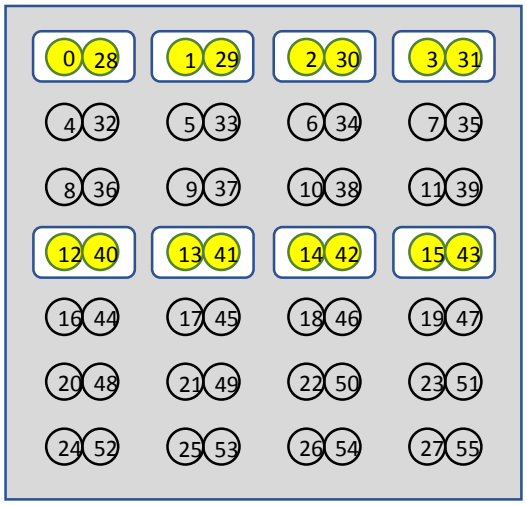
srun --mpi=pmi2 *executable arg arg arg …*

sinteractive --ntasks=8 --nodes=1

mpirun *executable arg arg arg …*

launcher

Multi-threaded MPI Job: single node



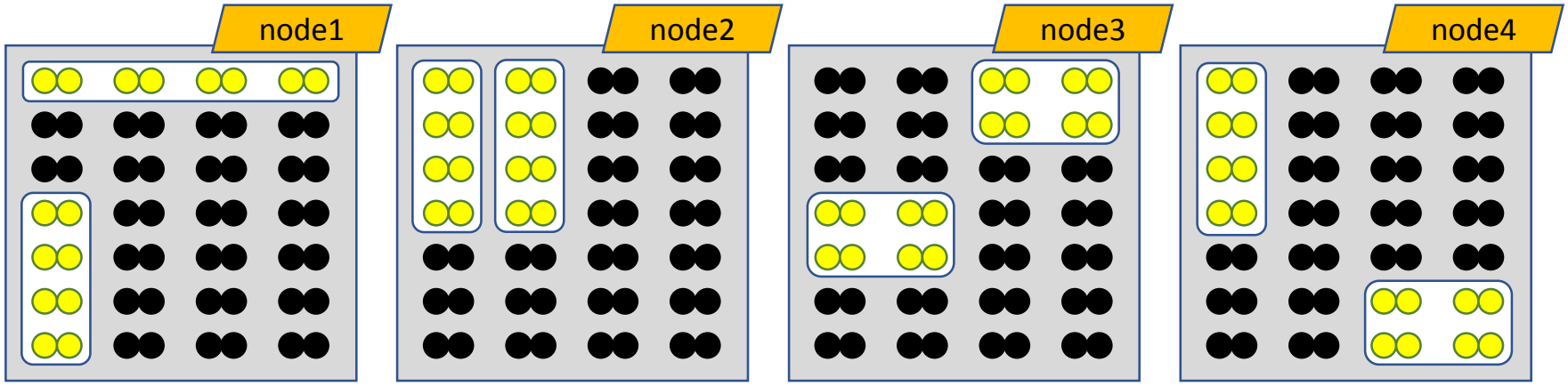

```
sbatch --ntasks=8 --nodes=1 --cpus-per-task=2
```

```
srun --mpi=pmi2 executable -p $SLURM_CPUS_PER_TASK arg arg arg …
```

```
sinteractive --ntasks=8 --nodes=1 --cpus-per-task=2
```

```
mpirun executable -p $SLURM_CPUS_PER_TASK arg arg arg …
```

Multi-threaded MPI Job: multiple nodes




```
sbatch --cpus-per-task=8 --ntasks=8 --partition=multinode
```

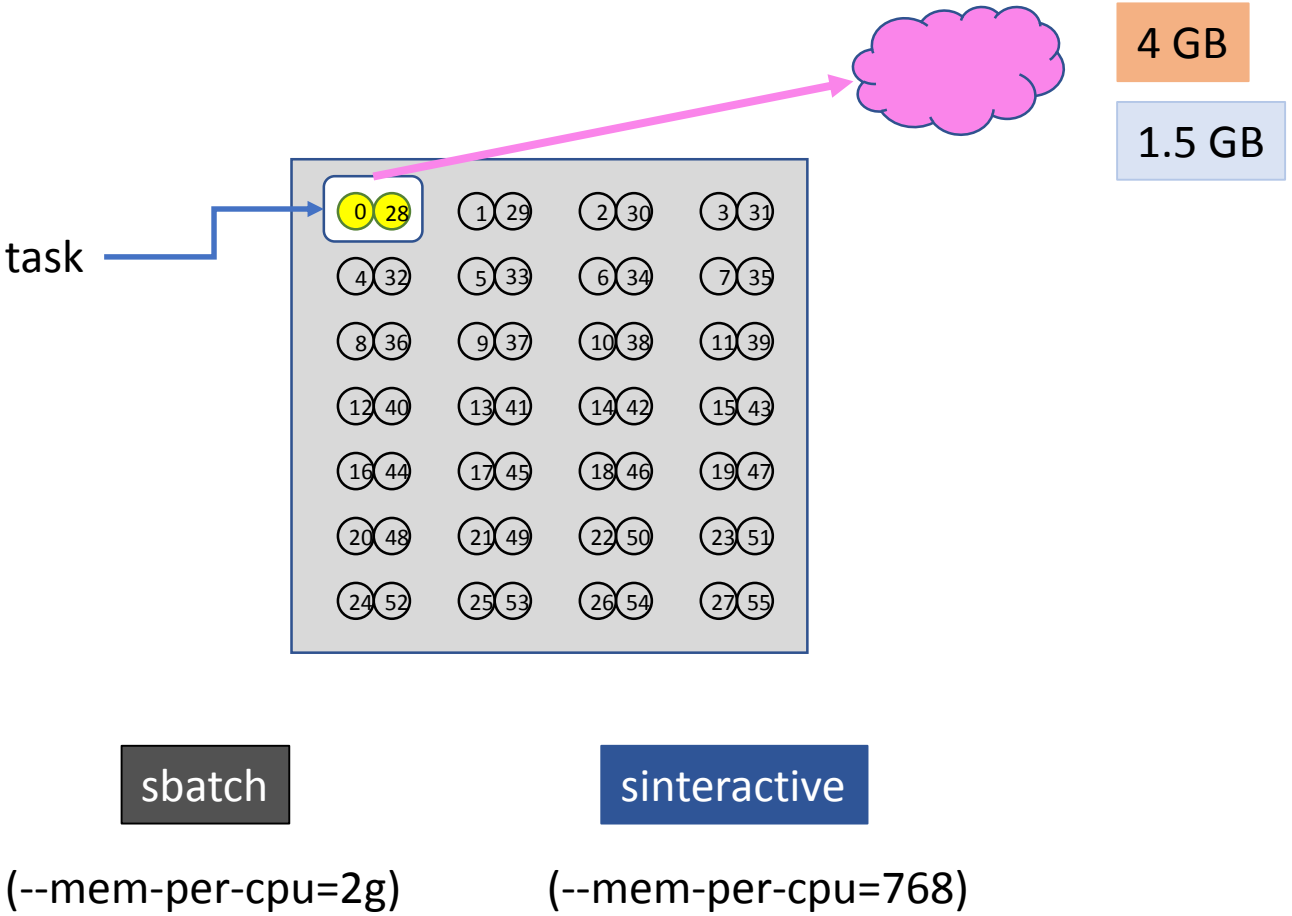
```
srun --mpi=pmi2 executable -p $SLURM_CPUS_PER_TASK arg arg arg …
```

```
sinteractive --cpus-per-task=8 --ntasks=8 --partition=multinode
```

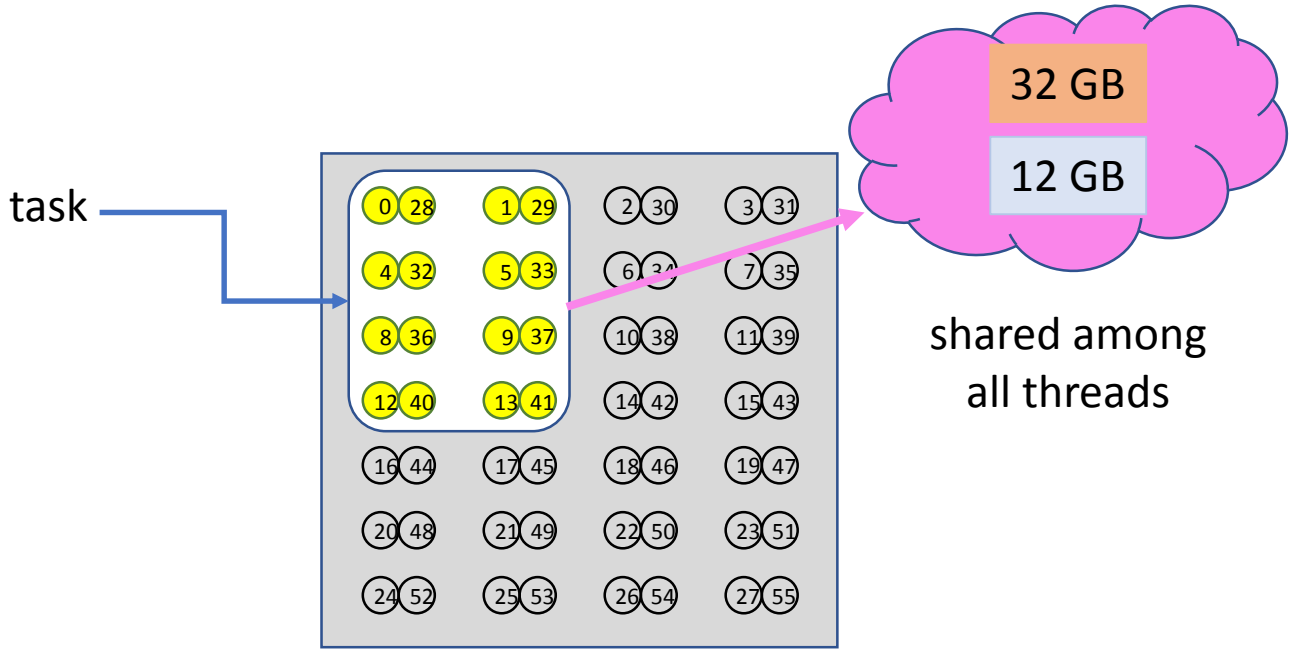
```
mpirun executable -p $SLURM_CPUS_PER_TASK arg arg arg …
```

Slurm Resources

Memory: Single-threaded Job



Memory: Multi-threaded Job



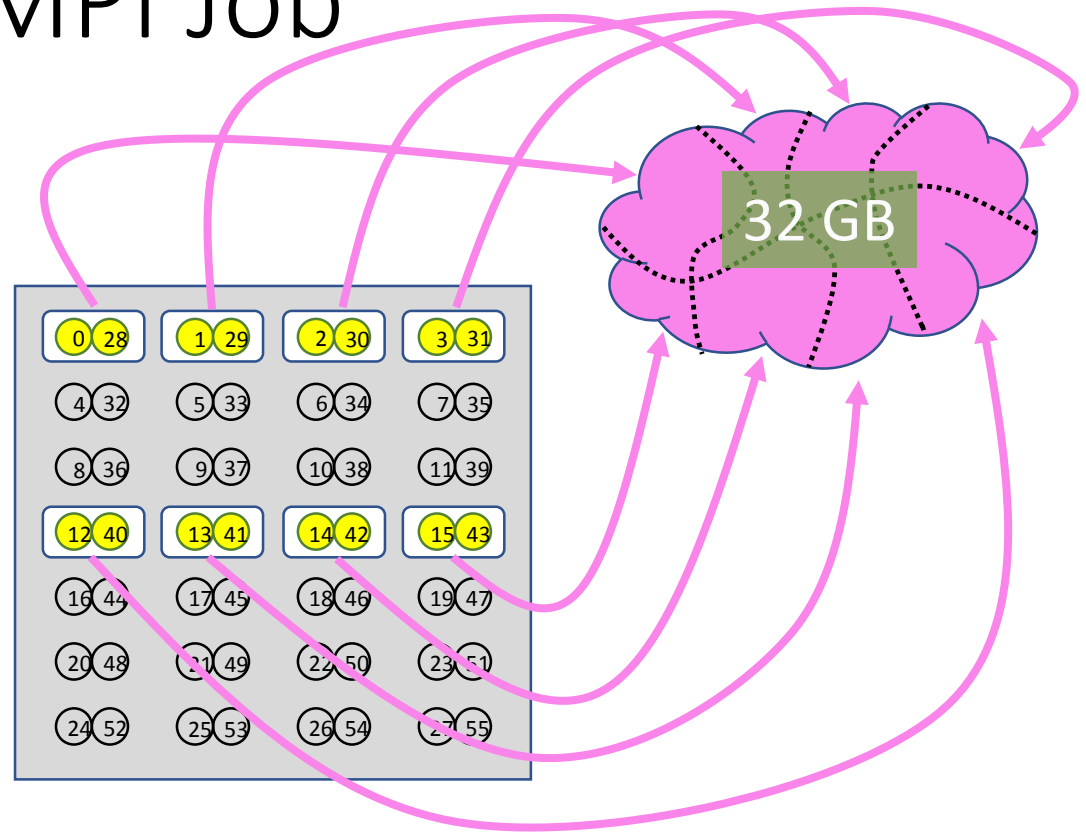
(--mem-per-cpu=2g)

sbatch --cpus-per-task=16

(--mem-per-cpu=768)

sinteractive --cpus-per-task=16

Memory: MPI Job



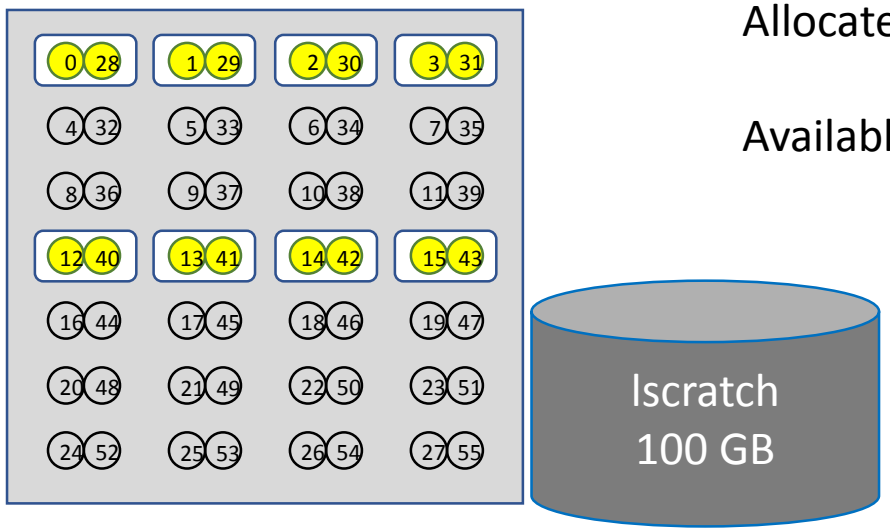

```
sbatch --ntasks=8 --nodes=1 --cpus-per-task=2 --mem-per-cpu=2g
```

$SLURM_MEM_PER_CPU=2048

```
sbatch --ntasks=8 --nodes=1 --cpus-per-task=2 --mem=32g
```

$SLURM_MEM_PER_NODE=32768

Local Scratch: Single Node



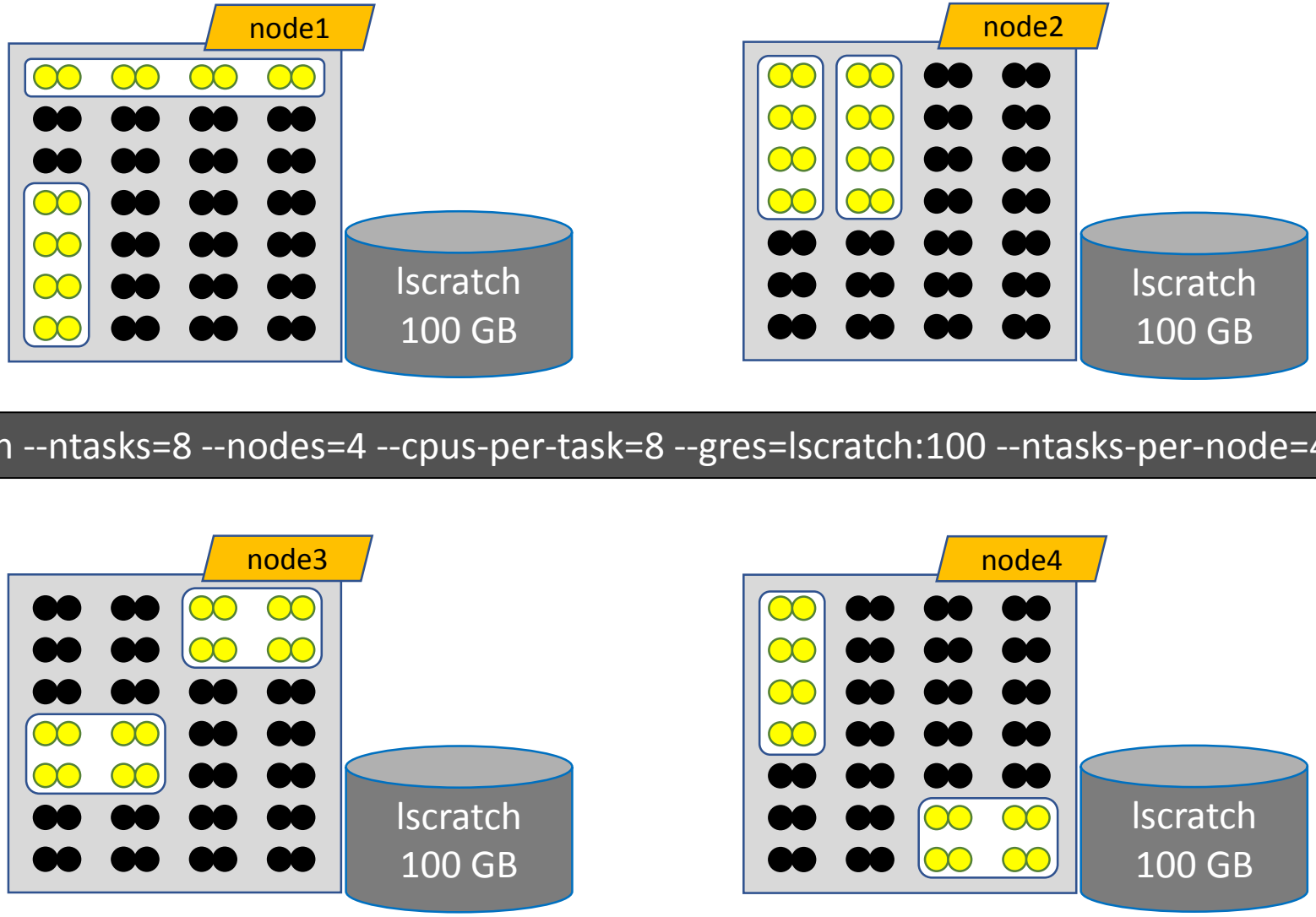
Allocated per node

Available to all tasks

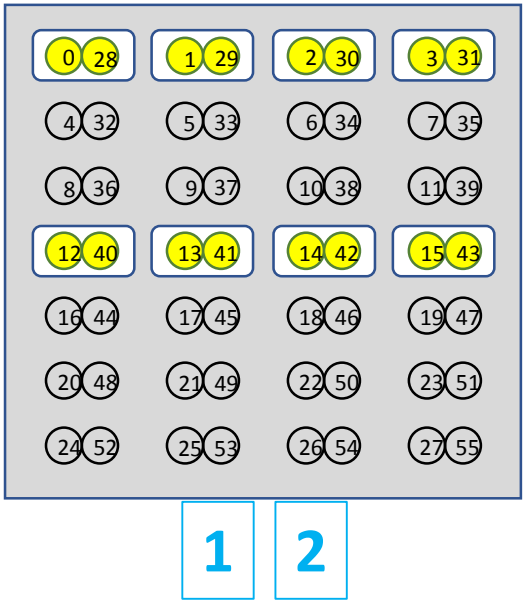
```
sbatch --cpus-per-task=2 --ntasks=8 --gres=lscratch:100
```

/lscratch/$SLURM_JOB_ID

Local Scratch: Multiple Nodes



GPUs



Allocated per node

Accesible to all tasks

```
sbatch --ntasks=8 --nodes=1 --cpus-per-task=2 --gres=gpu:p100:2
```

CUDA_VISIBLE_DEVICES=1,2

GPU Resource Types

- k20x: NVIDIA Kepler 20 (10 GB VRAM, 2x PN)
- k80: Kepler 80 (12 GB VRAM, 4x PN)
- p100: Pascal 100 (16 GB VRAM, 4x PN)

```
--gres=gpu:GPUtype:NGPU
```

Time

- Time is a resource – don’t waste it!
- Default depends on partition

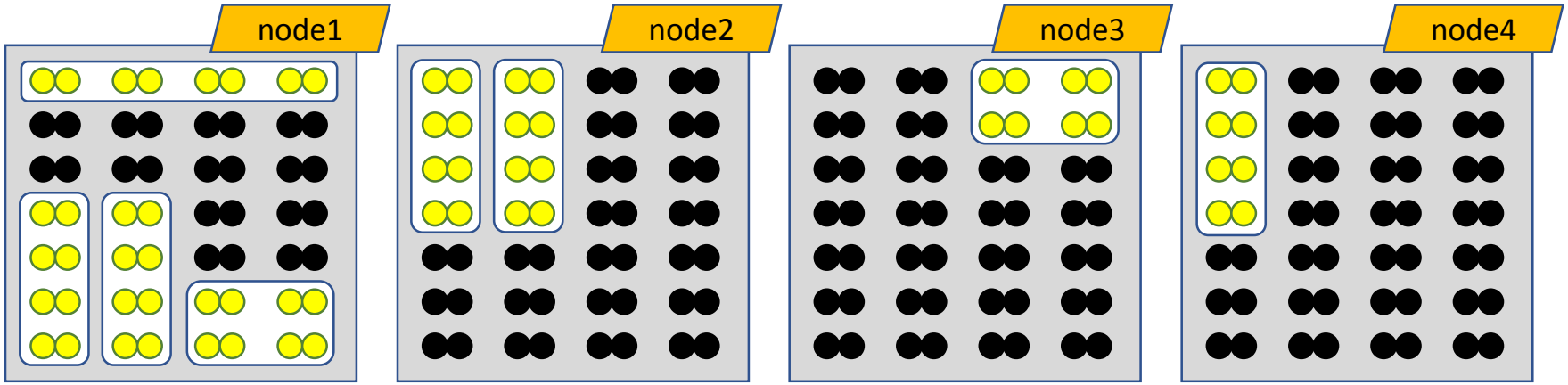
```
sbatch --time=60
```

```
sbatch --time=4:00:00
```

```
sbatch --time=2-00:00:00
```

MPI Job: Distribution

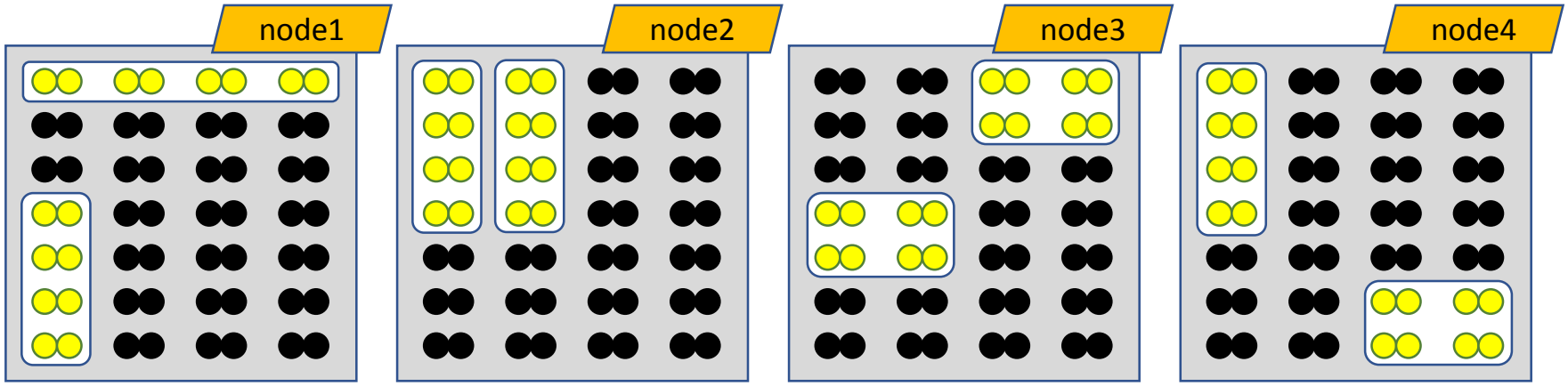
No guarantee of task placement




```
sbatch --ntasks=8 --cpus-per-task=8
```

MPI Job: Distribution

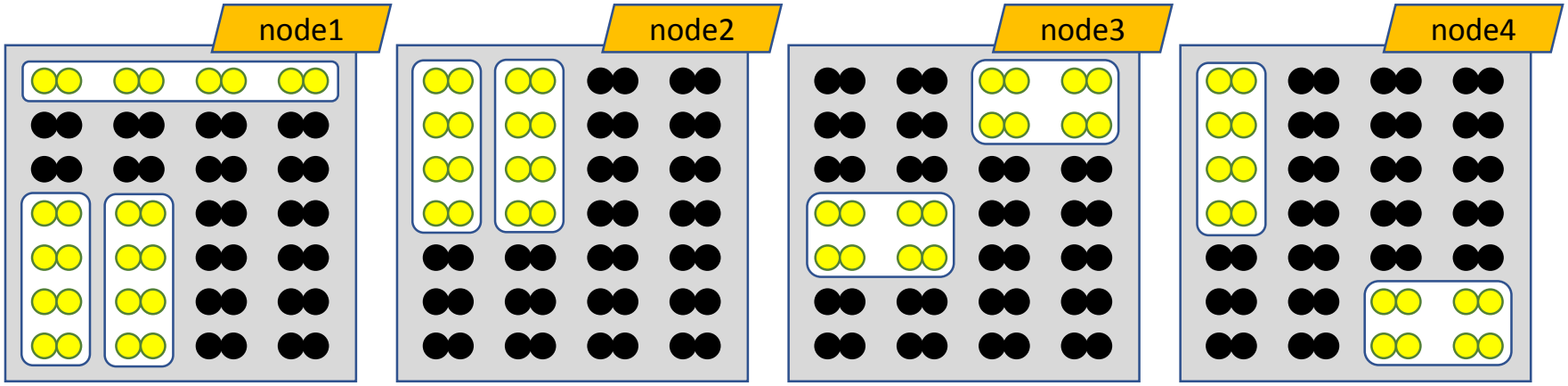
“Egalitarian”, “even” or “flat” distribution




```
sbatch --ntasks=8 --cpus-per-task=8 --ntasks-per-node=2
```

MPI Job: Distribution

arbitrary distribution




```
sbatch --ntasks=8 --cpus-per-task=8 --ntasks-per-node=2
 --distribution=arbitrary
```

```
export SLURM_HOSTFILE=file
unset SLURM_NTASKS_PER_NODE
```

file:

```
node1
node1
node1
node2
node2
node3
node3
node4
node4
```

Other Resources

- Network type (ethernet vs. infiniband)
- Processor type (x2680, x2695, x2650)

RELION Run Methods

There are four options

- --no_parallel_disc_io
- --preread_images
- --scratch_dir=*directory path*
- --dont_combine_weights_via_disc

--no_parallel_disc_io

- Giving this option causes master process to do all the reading and writing
- Don’t set this

--preread_images

- Giving this option causes all MPI processes to read particle and image data into memory
- Requires quite a bit of memory

*number of particles * box_size * box_size * 4 / (1024 * 1024 * 1024)*

- About the same speed as local scratch

--scratch_dir=*directory path*

- Giving this option causes a single process per node to copy micrographs to local scratch directory
- --scratch_dir=/lscratch/$SLURM_JOB_ID
- Usually takes less time than --preread_images to get started
- In general total runtime is about the same

--dont_combine_weights_via_disc

- If this option is NOT given, then after each iteration temporary files will be written to the work directory
- Total waste of time and I/O, unless you are debugging the program
- Always give this option

RELION Job Types

There are 5 Job Types

- Non-MPI, single-threaded, CPU-only
- Flat MPI, single-threaded, CPU-only
- Flat MPI, single-threaded, CPU + GPU (MotionCor2)
- Master-slave MPI, multi-threaded, CPU-only
- Master-slave MPI, multi-threaded, CPU + GPU

Single-threaded, no worries

```
#!/bin/bash
#SBATCH --partition=norm
#SBATCH --mem-per-cpu=8g
#SBATCH --error=Sort/job012/run.err
#SBATCH --output=Sort/job012/run.out
#SBATCH --time=60

module load RELION
relion_particle_sort --i ./Extract/job011/particles.star --ref ./Select/job007/class_averages.star \
  --o Sort/job012/particles_sort.star --ctf
```

RELION Parallelization

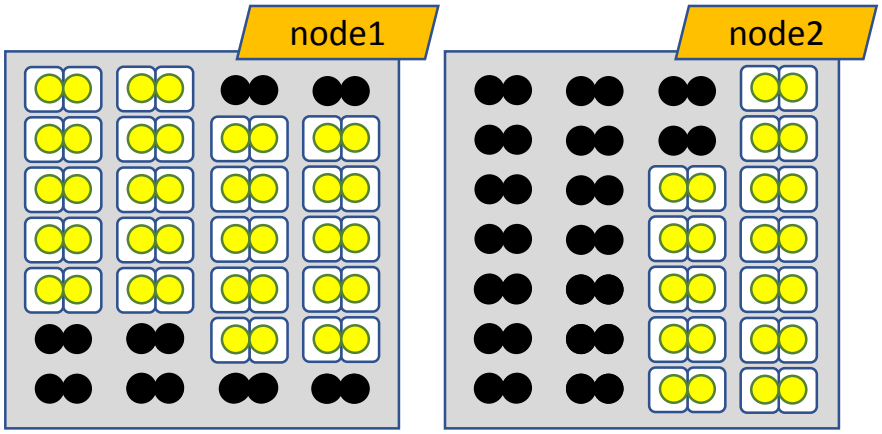
- Always done with MPI
- A few types use flat, single-threaded MPI (CTFFIND, particle sorting, MotionCor2)
- The rest use both MPI and pthreads (hybrid-parallel)
- Master-Slave method for hybrid-parallel
- Master = MPI rank 0, all others are slaves

Flat MPI, CPU-only: CTFFIND4

```
#!/bin/bash
#SBATCH --ntasks=64
#SBATCH --partition=multinode
#SBATCH --cpus-per-task=1
#SBATCH --error=CtfFind/job003/run.err
#SBATCH --output=CtfFind/job003/run.out
#SBATCH --time=240
#SBATCH --mem-per-cpu=8g

module load RELION
srun --mem-per-cpu=8g --mpi=pmi2 relion_run_ctffind_mpi \
--i MotionCorr/job002/corrected_micrographs.star --o CtfFind/job003/ \
--CS 2 --HT 300 --AmpCnst 0.1 --XMAG 10000 --DStep 3.54 --Box 512 --ResMin 30 --ResMax 7.1 \
--dFMin 5000 --dFMax 50000 --FStep 500 --dAst 100 --use_noDW  \
--ctffind_exe $RELION_CTFFIND_EXECUTABLE --ctfWin -1 --is_ctffind4 --only_do_unfinished
```

Flat, unfixed distribution of 64 tasks with 1 CPU per task, unknown number of nodes, memory allocated per CPU

Flat MPI, CPU-only



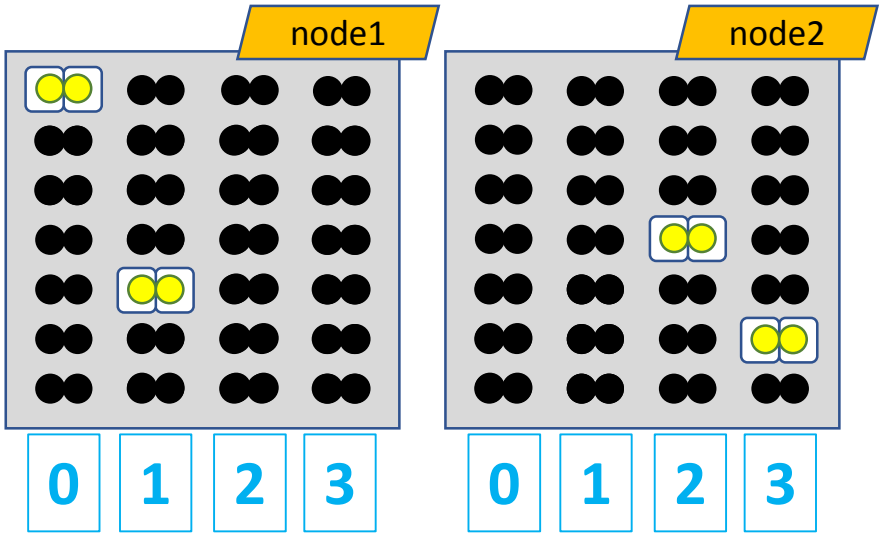
Flat, unfixed distribution of 64 tasks with 1 CPU per task, unknown number of nodes, memory allocated per CPU

Flat MPI, CPU+GPU: MotionCorr2

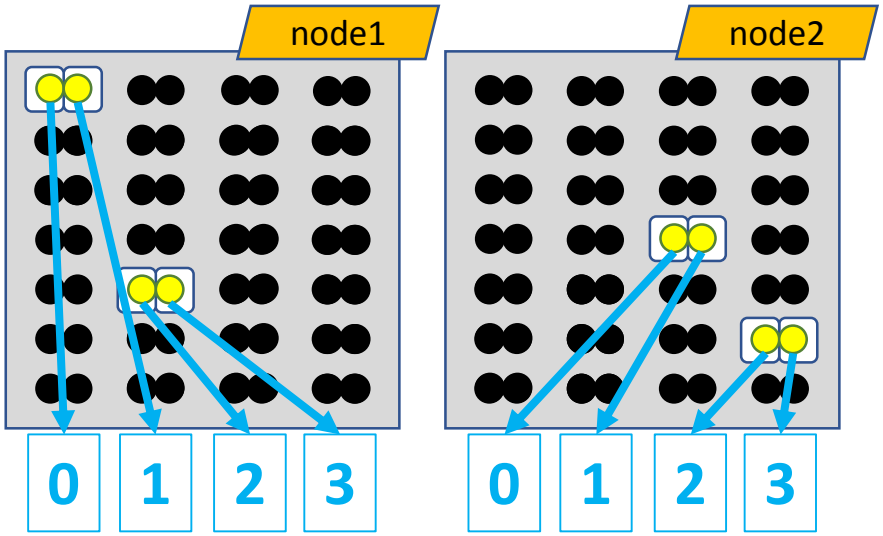
```
#!/bin/bash
#SBATCH --ntasks=8
#SBATCH --partition=gpu
#SBATCH --cpus-per-task=1
#SBATCH --error=MotionCorr/job002/run.err
#SBATCH --output=MotionCorr/job002/run.out
#SBATCH --time=240
#SBATCH --mem=10g
#SBATCH --gres=gpu:p100:4
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4

module load RELION
srun --mem-per-cpu=8g --mpi=pmi2 relion_run_motioncorr_mpi --i ./Import/job001/movies.star \
--o MotionCorr/job002/ --save_movies  --first_frame_sum 1 --last_frame_sum 16 \
--use_motioncor2 --bin_factor 1 --motioncor2_exe $RELION_MOTIONCOR2_EXECUTABLE \
--bfactor 150 --angpix 3.54 --patch_x 5 --patch_y 5 --gpu "" --dose_weighting \
--voltage 300 --dose_per_frame 1 --preexposure 0 --only_do_unfinished
```

Flat, fixed distribution of 8 tasks on 2 GPU nodes, 4 CPUs and 4 GPUs each node, memory allocated per node, GPUs unassigned

Flat MPI, CPU+GPU: MotionCorr2



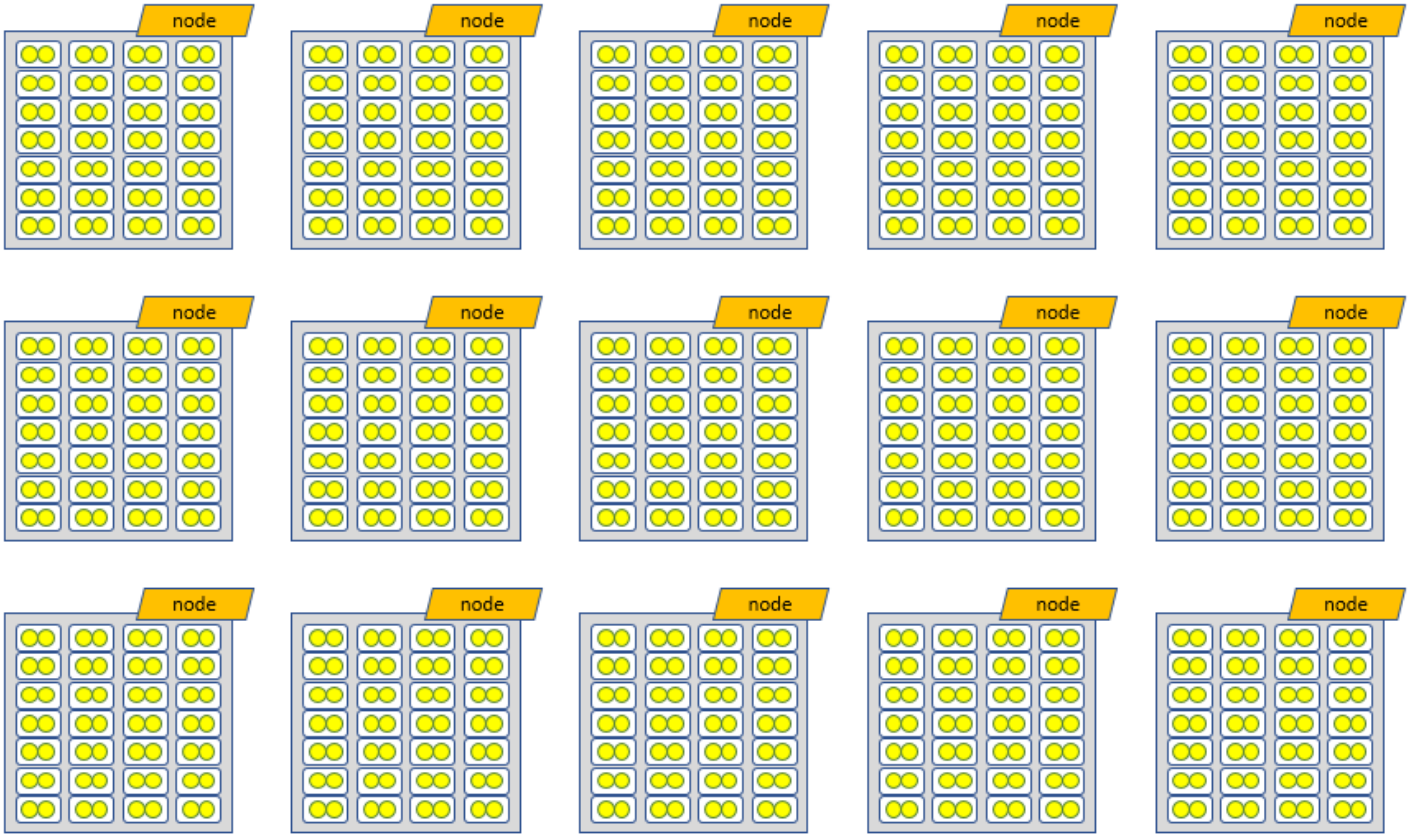
Flat MPI, CPU+GPU: MotionCorr2



Master-Slave CPU-only: 2D Classification

```
#!/bin/bash
#SBATCH --gres=lscratch:200
#SBATCH --partition=multinode
#SBATCH --ntasks=513
#SBATCH --cpus-per-task=2
#SBATCH --mem-per-cpu=8g
#SBATCH --error=Class2D/job004/run.err
#SBATCH --output=Class2D/job004/run.out
#SBATCH --time=2-00:00:00

source restrict_tcp_for_mpi.sh

srun --mpi=pmi2 relion_refine_mpi --o Class2D/job004/run --i Particles/shiny_2sets.star
--dont_combine_weights_via_disc --scratch_dir=/lscratch/$SLURM_JOB_ID --pool 100 --ctf --iter 25
--tau2_fudge 2 --particle_diameter 360 --K 200 --flatten_solvent --zero_mask --oversampling 1
--psi_step 6 --offset_range 5 --offset_step 2 --norm --scale --j 2
```

Master-Slave CPU-only: 2D Classification



This is a BIG job… likely about 50 nodes

Master-Slave CPU-only: 2D Classification – all in memory

```
#!/bin/bash
#SBATCH --partition=multinode
#SBATCH --ntasks=64
#SBATCH --cpus-per-task=16
#SBATCH --mem-per-cpu=15g
#SBATCH --ntasks-per-node=1
#SBATCH --error=Class2D/job011/run.err
#SBATCH --output=Class2D/job011/run.out
#SBATCH --time=2-00:00:00

source restrict_tcp_for_mpi.sh

srun --mpi=pmi2 relion_refine_mpi --o Class2D/job011/run --i Particles/shiny_2sets.star
--dont_combine_weights_via_disc --preread_images --pool 100 --ctf --iter 25 --tau2_fudge 2
--particle_diameter 360 --K 200 --flatten_solvent --zero_mask --oversampling 1 --psi_step 6
--offset_range 5 --offset_step 2 --norm --scale --j 16
```

Memory is shared among threads

Time required to read into memory can be substantial (hours?)

Not always true that --preread_images is faster that --scratch_dir

Tidbits

- RELION does not scale much beyond 2000 CPUs
- It takes a LONG time to read images into memory of every single MPI rank
- 50GB images took ~60 minutes to read into memory for ~500 ranks

Understanding GPU Jobs

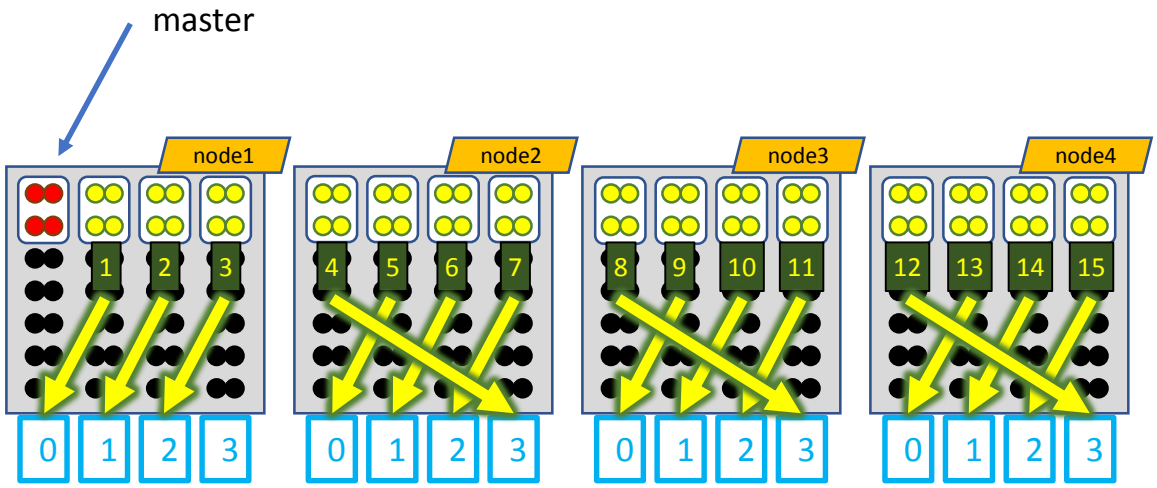
Understanding GPU Jobs

- GPUs accelerate particular execution steps 80x-100x
- Master – Slave model of parallelization conflicts with Slurm resource allocation and MPI rank distribution for GPU nodes
- GPU devices mapped to slaves on the basis of MPI rank id and device id

MPI rank - GPU Mapping

- Default --gpu ""
- GPUs are mapped to MPI ranks in order across ALL MPI ranks, not just those on a node
- You could set --gpu manually, but good luck, stockpile aspirin
- --gpu 0,0,0,0:1,1,1,1:2,2,2,2:3,3,3,3

Class2D, Class3D, Refine – Problems




```
sbatch
  --cpus-per-task=4
  --ntasks=16
  --ntasks-per-node=4
  --mem=240g
  --partition=gpu
  --gres=gpu:k80:4,lscratch:100
  --constraint=gpuk80
```

Why waste GPU when it is worth 80-100 CPUs?

Class2D, Class3D, Refine – Problems

```
sbatch
  --cpus-per-task=4
  --ntasks=17
  --ntasks-per-node=4
```

not possible due to GPU limit (16 per job)

```
sbatch
  --cpus-per-task=4
  --ntasks=17
  --nodes=4
```

random distribution of tasks on nodes

```
sbatch
  --cpus-per-task=4
  --ntasks=17
  --nodes=4
  --distribution=plane=1
```

complete overutilization of GPUs

```
sbatch
  --cpus-per-task=4
  --ntasks=17
  --nodes=4
  --mincpus=16
```

unpredictable which node receives 5 tasks

Arbitrary Distribution

- Over allocate on purpose

```
sbatch
  --cpus-per-task=4
  --ntasks=20
  --nodes=4
  --ntasks-per-node=5
  --distribution=arbitrary
```

- Create $SLURM_HOSTFILE and unset $SLURM_NTASKS_PER_NODE

```
array=( $( scontrol show hostname $SLURM_JOB_NODELIST) )
file=$(mktemp)
echo ${array[0]} > $file
for ((i=0;i<${SLURM_JOB_NUM_NODES};i++)); do
  for ((j=0;j<$((SLURM_NTASKS_PER_NODE-1));j++)); do
    echo ${array[${i}]} >> $file
  done
done
export SLURM_HOSTFILE=$file
unset SLURM_NTASKS_PER_NODE
```

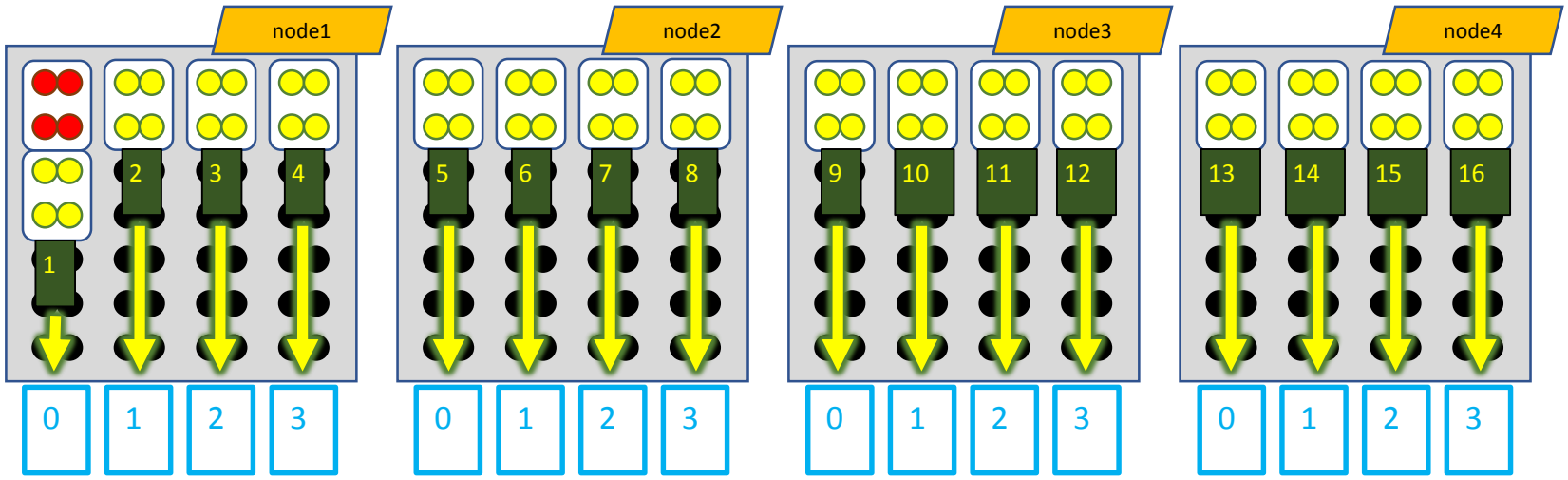
$SLURM_HOSTFILE

```
cn4000
cn4000
cn4000
cn4000
cn4000
cn4001
cn4001
cn4001
cn4001
cn4002
cn4002
cn4002
cn4002
cn4003
cn4003
cn4003
cn4003
```

RELION Utility Script

```
#!/bin/bash
#SBATCH --partition=gpu
#SBATCH --ntasks=20
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=30g
#SBATCH --ntasks-per-node=5
#SBATCH --nodes=4
#SBATCH --gres=gpu:p100:4,lscratch:200
#SBATCH --distribution=arbitrary
#SBATCH --error=Class2D/job011/run.err
#SBATCH --output=Class2D/job011/run.out
#SBATCH --time=2-00:00:00

source restrict_tcp_for_mpi.sh
source add_extra_MPI_task.sh

srun --mpi=pmi2 relion_refine_mpi --o Class2D/job011/run --i Particles/shiny_2sets.star
--dont_combine_weights_via_disc -scratch_dir=/lscratch/$SLURM_JOB_ID --pool 100
--ctf --iter 25 --tau2_fudge 2 --particle_diameter 360 --K 200 --flatten_solvent --zero_mask
--oversampling 1 --psi_step 6 --offset_range 5 --offset_step 2 --norm --scale --j 4 --gpu ""
```

Class2D, Class3D, Refine



Tidbits

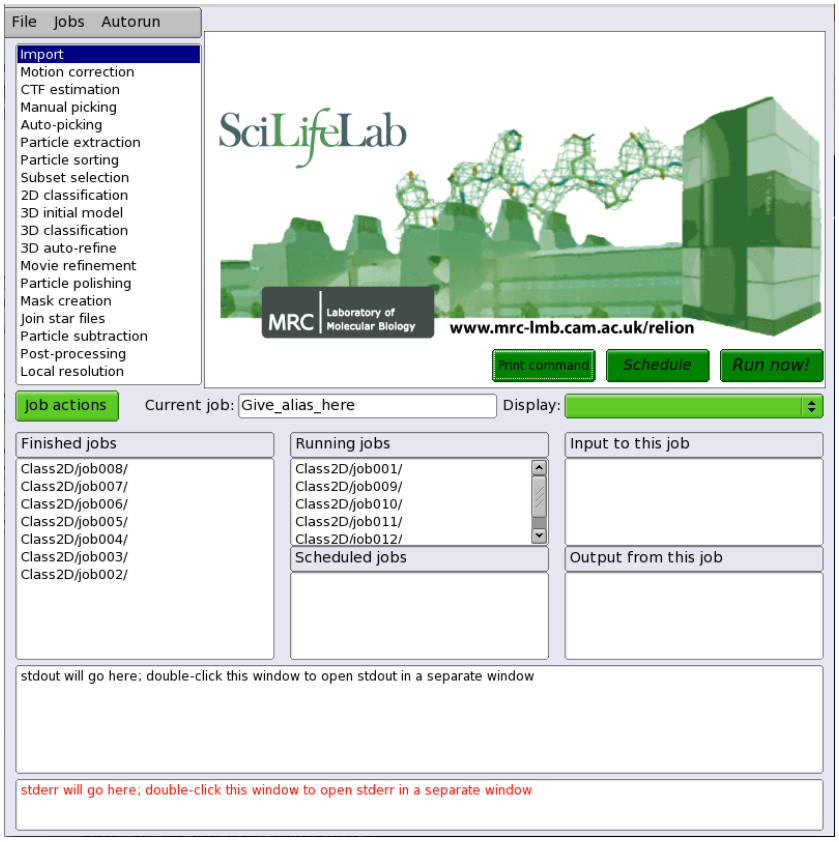
- Oversubscription of GPUs generally NOT a good idea
- Maintain 1 MPI rank per GPU, but 4 or 8 threads per GPU not always bad
- The more GPUs, the faster (to a point…)
- Run like Goldilocks

GPUs are not constantly used, and all steps not parallelizable

1. Noise estimation, parallelized over MPI-ranks, not threads
2. Expectation setup, parallelized over MPI-ranks, not threads
3. Expectation, parallelized over MPI-ranks and threads, GPU-accelerated
4. Maximization, parallelized over both MPI-ranks and threads
5. Estimate orientational accuracy, not parallelized
6. Next iteration, repeat from step 2

Maximize MPI, not threads!

Using the GUI



Don't run GUI on the login node

- Some job types run quickly
- You will forget you are on the login node

Be aware of options passed to sinteractive

- Default --mem-per-cpu=768

  SLURM_MEM_PER_CPU=768

- Subsequent batch MPI jobs may get confused

```
#!/bin/bash
#SBATCH --constraint=gpuk80
#SBATCH --ntasks=16
#SBATCH --nodes=4
#SBATCH --cpus-per-task=4
#SBATCH --mem=240g
#SBATCH --partition=XXXqueueXXX
#SBATCH --gres=gpu:k80:4,lscratch:XXXextra1XXX
#SBATCH --error=XXXerrfileXXX
#SBATCH --output=XXXoutfileXXX
#SBATCH --time=XXXextra2XXX

unset SLURM_MEM_PER_CPU
srun --mpi=pmi2 relion_refine_mpi --i Particles
```

command line > environment variable > within batch script

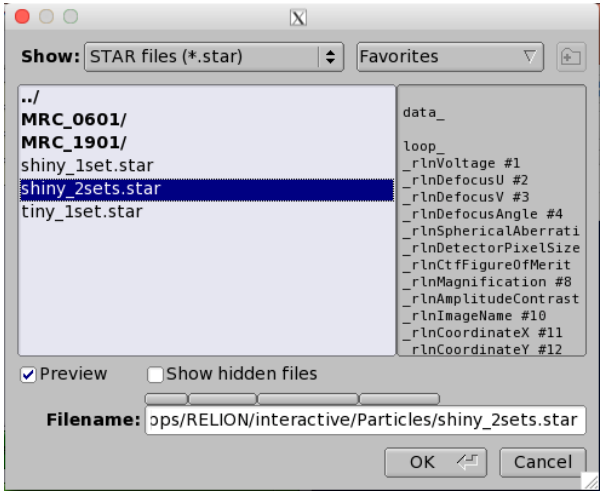
```
SLURM_MEM_PER_CPU=768
SLURM_MEM_PER_NODE=245760
```

```
srun: Force Terminated job step 58421330.0
srun: error: cn2692: task 0: Killed
srun: error: cn2692: task 1: Killed
srun: Force Terminated job step 58421330.1
```

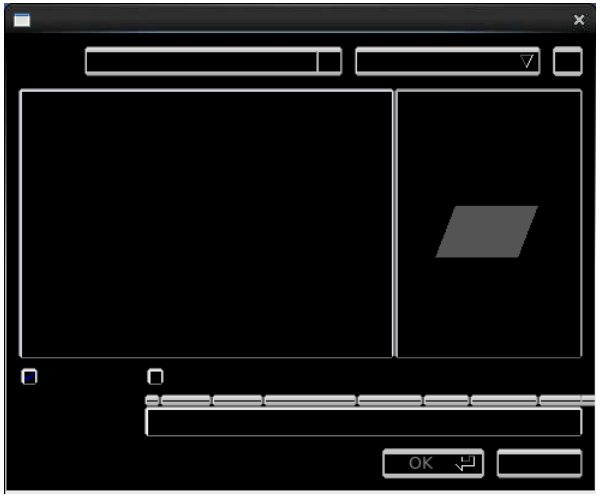
Always use --mem-per-cpu, unless you know what you are doing!

NX messes up file browsing

Xquartz:

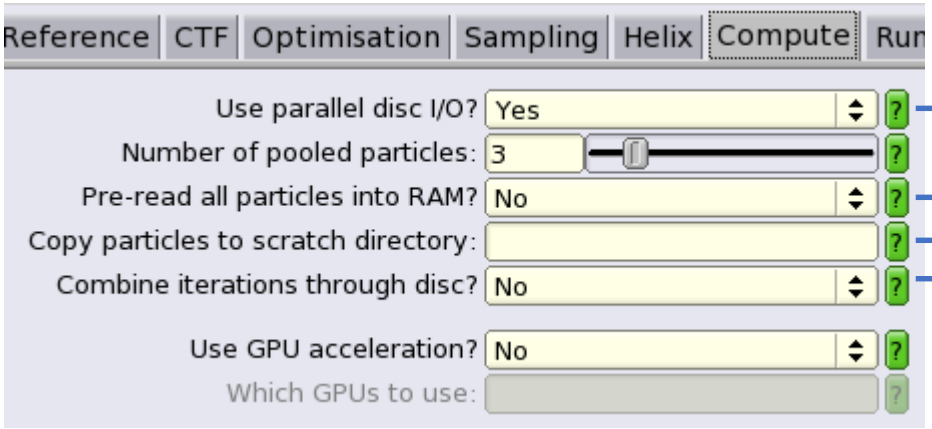


NX:



RELION Compute Options

- Four major choices to be made:



`--no_parallel_disc_io`

always set to Yes

`--preread_images`

depends, exclusive from below

`--scratch_dir=`

usually good idea, exclusive from above

`--dont_combine_weights_via_disc`

always set to No

```
relion_command ... --dont_combine_weights_via_disc --scratch_dir=/lscratch/$SLURM_JOB_ID ...
```

or

```
relion_command ... --dont_combine_weights_via_disc --preread_images ...
```

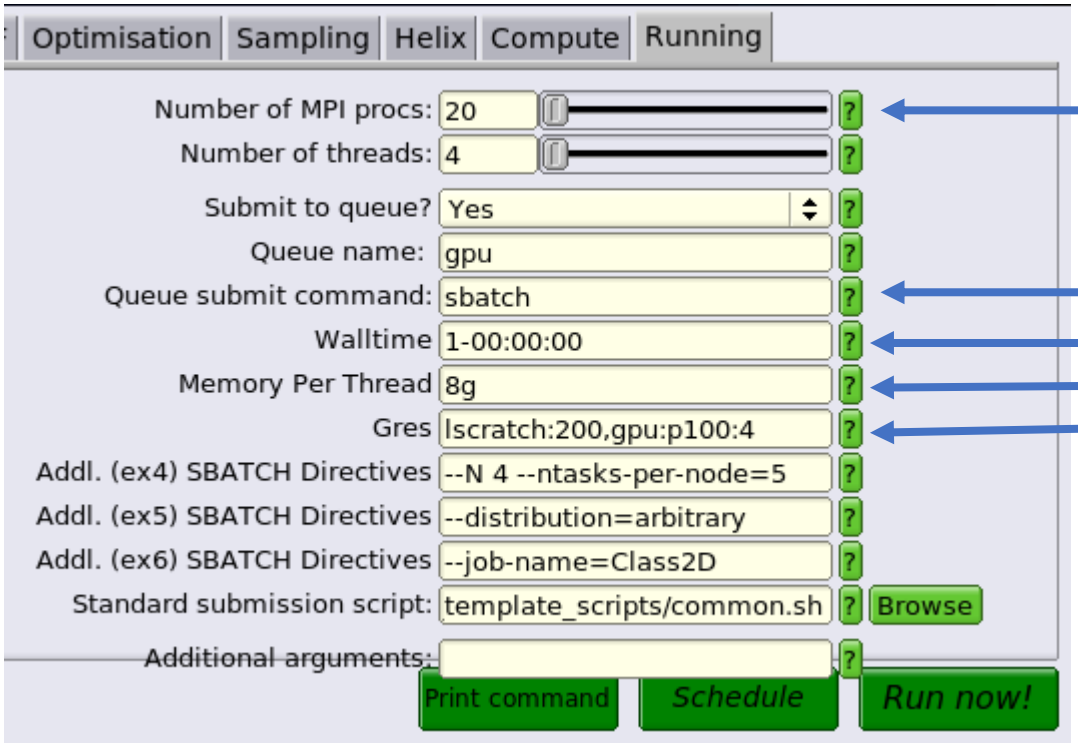
Choice

- Fastest execution (under ideal conditions) is to NOT use --scratch_dir or --preread_images
- OK for short jobs, but long (>2h) and large jobs will have I/O hassles
- --preread_images is perhaps a little faster
- --scratch_dir most reliable

RELION/2.1.0-new

- GUI code hacked to be more Slurm friendly
- Allows a single batch script template
- Will be the default under CentOS7

Running/Batch System Options



1 = non-MPI
>1 = MPI

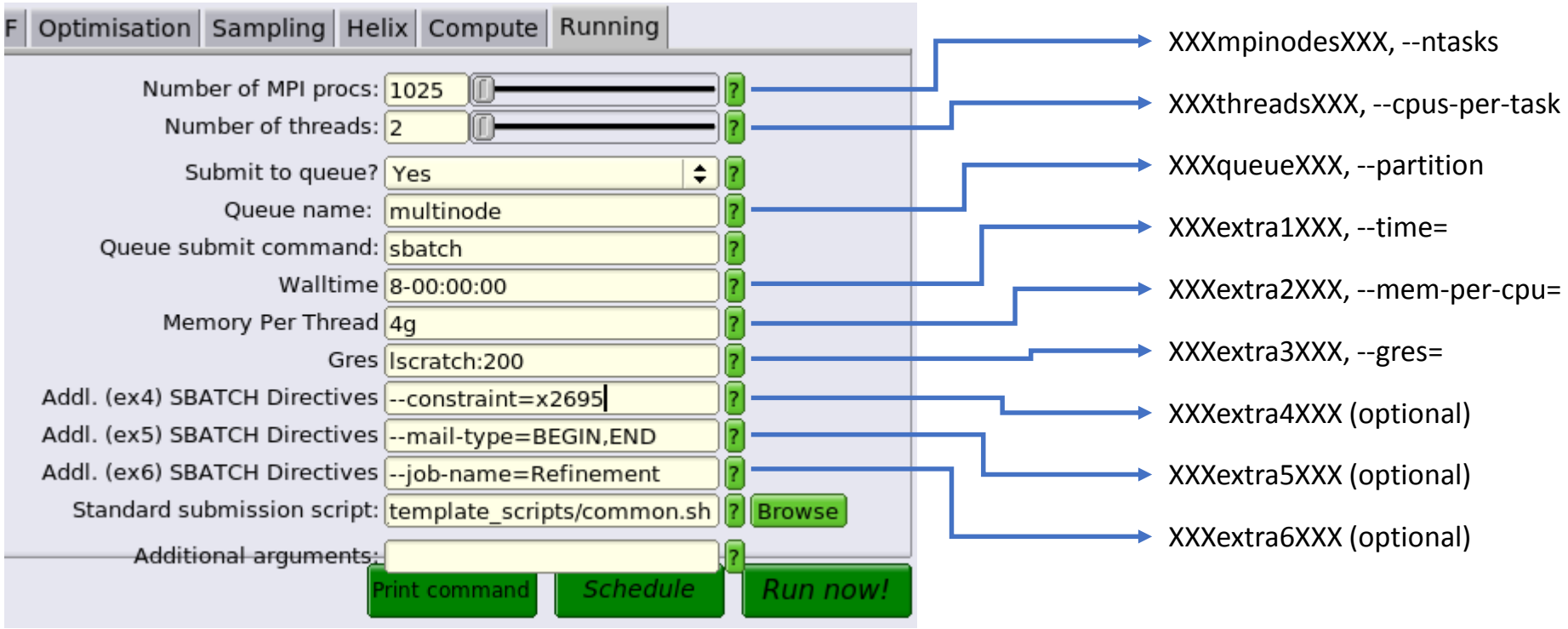
must match job requirements
(norm, multinode, gpu)

--time must be set
--mem-per-cpu must be set
--gres is optional

Room for other sbatch options

/usr/local/apps/RELION/batch_template_scripts/common.sh

Running/Batch System Options



/usr/local/apps/RELION/batch_template_scripts/common.sh

common.sh batch script template

```
#!/bin/bash
#SBATCH --ntasks=XXXmpinodesXXX
#SBATCH --partition=XXXqueueXXX
#SBATCH --cpus-per-task=XXXthreadsXXX
#SBATCH --error=XXXerrfileXXX
#SBATCH --output=XXXoutfileXXX
#SBATCH --time=XXXextra1XXX
#SBATCH --mem-per-cpu=XXXextra2XXX
#SBATCH --gres=XXXextra3XXX
#SBATCH XXXextra4XXX
#SBATCH XXXextra5XXX
#SBATCH XXXextra6XXX

source restrict_tcp_for_mpi.sh
source add_extra_MPI_task.sh

srun --mem-per-cpu=XXXextra2XXX --mpi=pmi2 XXXcommandXXX
```

Memory

- Minimum amount of memory (per MPI rank) for --preread_images =

  *number of particles * box_size * box_size * 4 / (1024 * 1024 * 1024)*

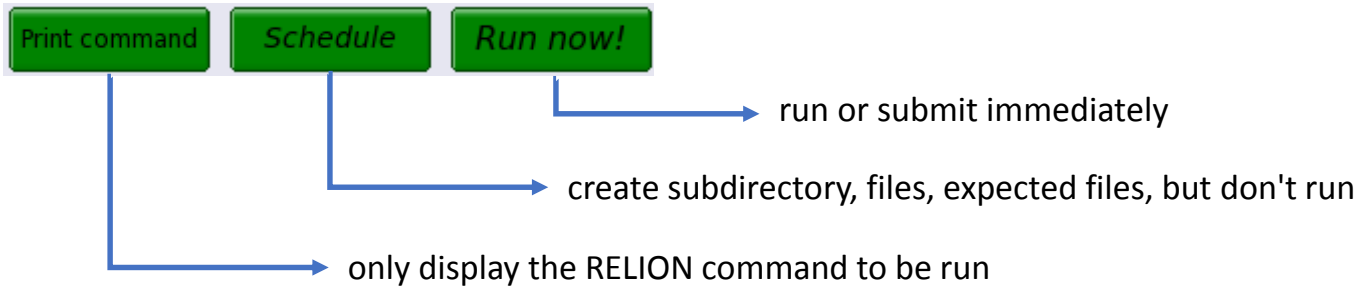
- E.g. 100K particles, 360 pixel box will need at least 48GB per task
- Memory use will grow 10-20% over time
- Convert to --mem-per-cpu = [*calc RAM required*]/--cpus-per-task
- Same example above with -j 4 (--cpus-per-task=4) needs 14g per cpu

"Dumb down" the network interface

```
export OMPI_MCA_btl="self,sm,tcp"
export OMPI_MCA_btl_tcp_if_include="10.2.0.0/18"
```

- No difference in execution time between Infiniband and plain 'ol TCP
- MPI RELION can get confused on management network interfaces (10.90)
- Will give warnings when a network interface doesn't exist
- Automated with **restrict_tcp_for_mpi.sh** script

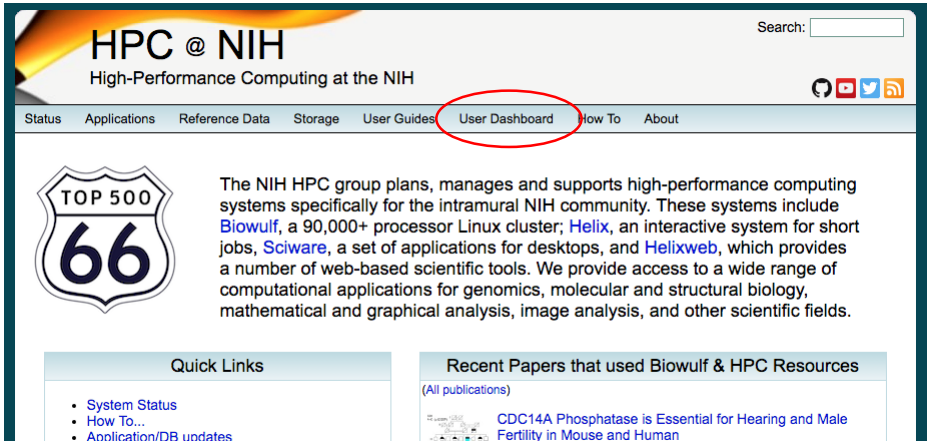
Check twice, run once

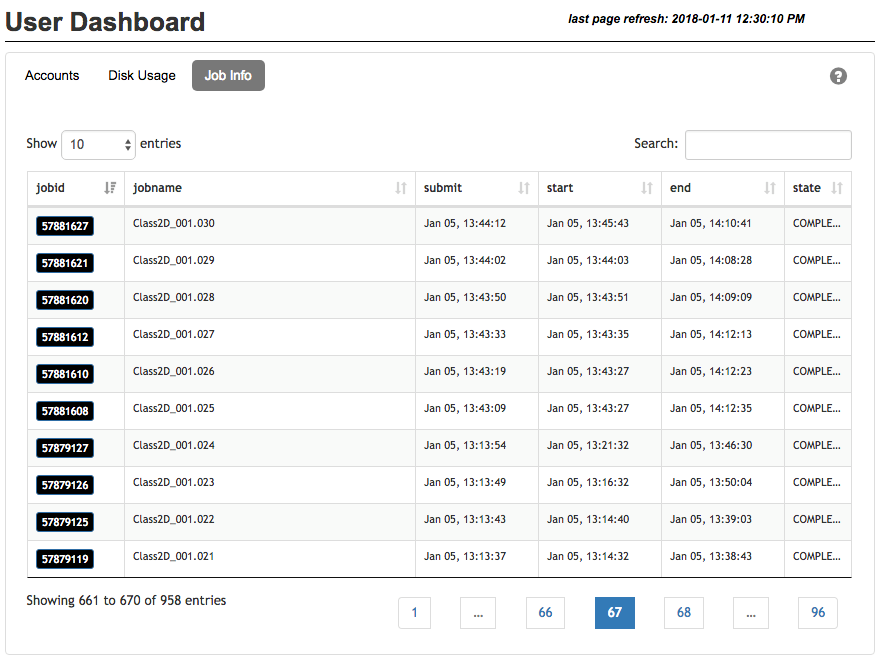


RELION Job Monitoring

- run.out and run.err files
- jobhist, jobload, jobdata
- dashboard
- nvidia-smi

Dashboard



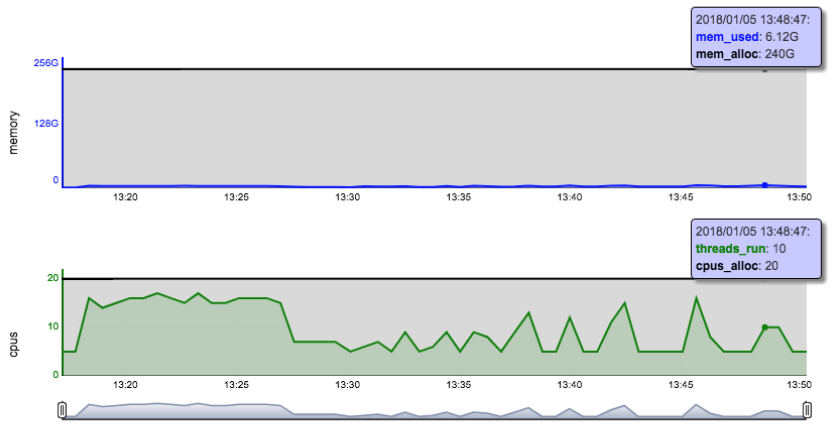


Jobs remain listed on the dashboard for 10 days after ending

Search by jobid, jobname, timestamp, state

Dashboard

Biowulf Job 57879126



jobname	Class2D_001.023
submitted	Jan 05, 13:13:49
state	COMPLETED
submission script	
work directory	/data/hooverdm/private/apps/RELION/benchmarks/Class2D
STDIN	/dev/null
STDOUT	/data/hooverdm/private/apps/RELION/benchmarks/Class2D/Class2D_001/023/run.out
STDERR	/data/hooverdm/private/apps/RELION/benchmarks/Class2D/Class2D_001/023/run.err
alloc nodes	4
total alloc CPUs	80

nodelist	cn4186,cn4187,cn4188,cn4189

ended	Jan 05, 13:50:04
CPU hours	44.711113
alloc mem per cpu	12.0 GB
max mem per cpu	296 MB
exit code	0

script

```
#!/bin/bash
mkdir /lscratch/${SLURM_JOB_ID}/TMPDIR
export TMPDIR=/lscratch/${SLURM_JOB_ID}/TMPDIR
export OMPI_MCA_btl="self,sm,tcp"
export OMPI_MCA_btl_tcp_if_include="10.2.0.0/18"
echo SLURM_JOB_ID=${SLURM_JOB_ID}
echo RELION=2.1.0
echo NT=20
echo CPT=4
echo TPN=5
echo SLURM_JOB_NUM_NODES=${SLURM_JOB_NUM_NODES}
echo GPN=4
echo date=$(date)

array=( $( scontrol show hostname $SLURM_JOB_NODELIST) )
file=$(mktemp)
echo ${array[0]} > $file
for ((i=0;i<${SLURM_JOB_NUM_NODES};i++)); do
  for ((j=0;j<$((SLURM_NTASKS_PER_NODE-1));j++)); do
    echo ${array[${i}]} >> $file
  done
done
export SLURM_HOSTFILE=$file

unset SLURM_NTASKS_PER_NODE
srun --mpi=pmi2 relion_refine_mpi --o Class2D_001/023/run --i Particles/tiny_1set.
star --pool 100 --ctf --iter 10 --tau2_fudge 2 --particle_diameter 360 --K 200 --f
latten_solvent --zero_mask --oversampling 1 --psi_step 6 --offset_range 5 --offset
_step 2 --norm --scale --j 4  --no_parallel_disc_io --scratch_dir /lscratch/${SLUR
M_JOB_ID}  --random_seed 0 --gpu

for i in $(ls Class2D_001/023/*.{star,mrcs}) ; do hollow_file.sh $i ; done

echo date=$(date)
```

nvidia-smi

- While job is running, display snapshot of GPU utilization

```
$ jobload -j 11111111
          JOBID            TIME            NODES  CPUS  THREADS   LOAD       MEMORY
                        Elapsed / Wall              Alloc   Active            Used /     Alloc
       11111111    00:52:53 /  1-00:00:00 cn4235    20       11    55%     4.0 /  240.0 GB
                   00:52:53 /  1-00:00:00 cn4236    16       15    94%     3.7 /  240.0 GB
                   00:52:53 /  1-00:00:00 cn4237    16       16   100%     3.6 /  240.0 GB
                   00:52:53 /  1-00:00:00 cn4238    16       14    88%     3.6 /  240.0 GB
                 Nodes:    4    CPUs:   68  Load Avg:  84%
$ ssh cn4236 nvidia-smi
Thu Jan 11 12:39:16 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 375.26                 Driver Version: 375.26                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K80           On   | 0000:83:00.0     Off |                  Off |
| N/A   78C    P0   137W / 149W |  11481MiB / 12205MiB |    100%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K80           On   | 0000:84:00.0     Off |                  Off |
| N/A   55C    P0   148W / 149W |  11481MiB / 12205MiB |     86%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla K80           On   | 0000:8A:00.0     Off |                  Off |
| N/A   79C    P0   120W / 149W |  11481MiB / 12205MiB |     89%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla K80           On   | 0000:8B:00.0     Off |                  Off |
| N/A   65C    P0   148W / 149W |  11481MiB / 12205MiB |     90%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|    0     37242    C   .../local/relion_2.1.0/bin/relion_refine_mpi 11477MiB |
|    1     37243    C   .../local/relion_2.1.0/bin/relion_refine_mpi 11477MiB |
|    2     37240    C   .../local/relion_2.1.0/bin/relion_refine_mpi 11477MiB |
|    3     37241    C   .../local/relion_2.1.0/bin/relion_refine_mpi 11477MiB |
+-----------------------------------------------------------------------------+
$
```

nvidia-smi dmon –d 10

- Continuously monitor GPU utilization for defined periods of time
- dmon is highly configurable

```
$ ssh cn4236 nvidia-smi dmon -d 10
# gpu   pwr  temp    sm   mem   enc   dec  mclk  pclk
# Idx     W     C     %     %     %     %   MHz   MHz
    0   146    77   100     4     0     0  2505   810
    1   143    54    15     1     0     0  2505   797
    2    70    79     6     0     0     0  2505   810
    3   147    65    12     1     0     0  2505   875
    0   143    77   100     1     0     0  2505   810
    1   148    55   100     1     0     0  2505   810
    2   144    80   100     2     0     0  2505   823
    3   125    66    44     2     0     0  2505   797
    0   146    78    64     4     0     0  2505   810
    1   148    55   100     1     0     0  2505   810
    2   142    80   100     2     0     0  2505   810
    3    87    66    81     4     0     0  2505   875
    0    87    78   100     1     0     0  2505   810
    1    87    55   100     4     0     0  2505   810
    2   141    80   100     2     0     0  2505   810
    3   145    67   100     4     0     0  2505   797
    0   145    78   100     4     0     0  2505   810
    1   148    56   100     2     0     0  2505   810
    2   141    80   100     2     0     0  2505   810
    3   147    67   100     4     0     0  2505   797
^C$ ^C
```

Troubleshooting

- RELION touches every weak and sensitive point in the cluster

- Many different ways things can go south

Insufficient Memory

- Insufficient memory causes individual MPI ranks to be killed, leading to "zombie" RELION jobs
- run.err shows something like this:

```
srun: error: cn3068: task 3: Killed
```

- Job continues to run, but run.out does not progress
- Other causes for rank failure include network hiccups, quota overrun, generic hardware failure

Insufficient GPU Memory

- k80 GPUs have access to 12 GB VRAM
- p100 GPUs have 16 GB VRAM

```
                                ***WARNING***

With the current settings and hardware, you will be able to
use an estimated image-size of 533 pixels during the last iteration...

...but your input box-size (image_size) is however 659. This means that
you will likely run out of memory on the GPU(s), and will have to then re-start
from the last completed iteration (i.e. continue from it) *without* the use of GPUs.

You are also splitting at least one GPU between two or more mpi-slaves, which
might be the limiting factor, since each mpi-slave that shares a GPU increases the
use of memory. In this case we recommend running a single mpi-slave per GPU, which
will still yield good performance and possibly a more stable execution.
```

Mixing RELION Versions

```
PipeLine::read: cannot find name or type in pipeline_nodes table
```

- The GUI creates hidden files for maintaining state between sessions under .Nodes
- Running a different version of RELION on top of .Nodes may cause corruption

```
$ tree .Nodes
.Nodes
├── 0
│   ├── Import
│   │   └── job001
│   │       └── movies.star
│   └── MotionCorr
│       └── job002
│           └── corrected_micrograph_movies.star
├── 1
│   ├── CtfFind
│   │   └── job003
│   │       └── micrographs_ctf.star
│   ├── ManualPick
│   │   └── job004
│   │       └── micrographs_selected.star
│   └── MotionCorr
│       └── job002
│           └── corrected_micrographs.star
├── 10
│   └── Refine3D
│       └── job012
│           └── run_half1_class001_unfil.mrc
├── 11
│   ├── LocalRes
│   │   └── first3dref
│   │       └── relion_locres_filtered.mrc
│   └── PostProcess
│       └── first3def
│           ├── postprocess_masked.mrc
│           └── postprocess.mrc
├── 12
│   └── LocalRes
│       └── first3dref
│           └── relion_locres.mrc
├── 13
│   ├── MotionCorr
│   │   └── job002
```

Inappropriate RELION Options

- Almost always diagnosable from .out and .err files
- Contact RELION developers or CCPEM mailing list if warning is unclear
- Mixing --preread_images and --scratch_dir
- Too many classes (--K)
- --write_fom_maps from within AutoPick

I/O

- Chronic problem nowadays
- Certain steps (copying to /lscratch, reading into memory) take inordinately long
- Don't launch 1000's of nodes simultaneously – stagger
- Use GPUs whenever possible
- ALWAYS SET --dont_combine_weights_via_disc

Bad Input

- Filter and prune your data!
- Choose only the best images, particles, and classes
- More data does NOT mean better results

Questions? Comments?

staff@hpc.nih.gov