# BLUE WATERS

SUSTAINED PETASCALE COMPUTING

## Running Jobs on Blue Waters

Jing Li, Omar Padron

NCSA


GREAT LAKES CONSORTIUM
FOR PETASCALE COMPUTATION

CRAY

# Jobs on Blue Waters

- Jobs on Blue Waters are managed through the use of:
  - Resource manager: TORQUE
  - Workload manager:  Moab
- Commands for managing jobs on Blue Waters are a subset of PBS (Portable Batch System) commands.
- Application launcher (aprun) utility launches applications on compute nodes.
- Application Level Placement Scheduler (ALPS) handles application placement and execution.

# Jobs on Blue Waters

- Two type of jobs – interactive and batch
- Interactive mode for debug and optimization
- Batch mode for normal job runs
- Steps for setting up a job:
  - Determine the resources needed
  - Pick a queue that will provide the required resources
  - Create a PBS job script that includes the aprun command and describes the resources needed
  - Submit the job script to the batch system using qsub

# Submit Jobs

- Interactive jobs

```
qsub –I -l nodes=1
qsub –I –l nodes=2:ppn=32:xe -l
walltime=01:00:00
```

- Batch jobs

```
qsub my_script.pbs
```

# PBS Script

- Sample scripts are at /sw/userdoc/samplescripts
- Extensive discussion in Documentation on Blue Waters Portal
- PBS scripts
  - Specify resource needed
  - Provide file names for stdout and stderr
  - Define environmental variables
  - Load needed modules
  - Launch the job via the aprun command

# PBS Script - Resources and Notification

```
#PBS -l nodes=2048:ppn=32:xe
#PBS -l walltime=01:20:00
#PBS -N testjob
#PBS -e $PBS_JOBID.err
#PBS -o $PBS_JOBID.out
#PBS -m bea
#PBS -M username@host
```

# PBS Script - Others

- cd $PBS_O_WORKDIR
  - By default, job starts at $HOME, your home directory
  - Often it is desirable that job starts where batch job is submitted, which is in $PBS_O_WORKDIR
  - Excutables used, input and output files can come from and go to directories of choice
- module load your_modules
  - module load craype-hugepages2M
- export your_environment_variables
  - export OMP_NUM_THREAD = 2
- aprun -n 65536 ./app.exe  < in > out.$PBS_JOBID

# PBS Script – Simple Example

```
#!/bin/bash
#PBS -l nodes=2:ppn=32:xe
#PBS -l walltime=00:30:00
#PBS -N testjob

. /opt/modules/default/init/bash
module swap PrgEnv-cray PrgEnv-gnu

cd $PBS_O_WORKDIR

aprun -n 64 ./app.exe < input.dat > output.out
```

# Submit and Manage Jobs

- qsub – submit job

```
qsub -I -l nodes=2:ppn=32:xe -l walltime=01:00:00
qsub my_script.pbs
```

- qstat - check job status

```
qstat
qstat | grep –i john
```

- qdel – remove a job

```
qdel job_id
```

# Aprun options

-n: Number of processing elements PEs for the application

-N: Number of PEs to place per node

-S: Number of PEs to place per NUMA node.

-d: Number of CPU cores required for each PE and its threads

-cc: Binds PEs to CPU cores.

-r: Number of CPU cores to be used for core specialization

-ss: Enables strict memory containment per NUMA node

# Aprun - basics

- MPI runs for 8 and 64 MPI's:

```
aprun –n 8  a.out (Needs 1 node.)
aprun –n 64 a.out (Needs 2 nodes.)
```

- MPI + openMP runs  for 4 MPI's each having two openMP threads, and 16 MPI's each having 4 threads:

```
aprun –n 4 –d 2 a.out (Needs 1 node.)
aprun –n 16 –d 4 a.out (Needs 2 nodes.)
```

# Aprun - advanced

- MPI runs

```
aprun –n 8 –S 2 a.out  (Needs 1 node, Uses 4 NUNA.)
aprun –n 64 –N 16 –S 4 a.out (Needs 4 node, Uses 4 NUNA.)
aprun –n 64 –cc 0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30 a.out
(Needs 4 node, Uses 4 NUNA.)
```

- MPI + openMP runs

```
aprun –n 4 –S 1 –d 2 a.out (Needs 1 node, Uses 4 NUNA.)
aprun –n 16 –N 4 –S 1 –d 4 a.out (Needs 4 node, Uses 4 NUNA.)
aprun –n 32 –d 2 –cc 0,2:4,6:8,10:12,14:16,18:20,2:24,26:28,30
a.out (Needs 4 node, Uses 4 NUNA.)
```

# Aprun - advanced

- -cc specifies which cores to use on a node by a job.
- ``:'' sperates a PE and its threads from other PE's and their threads.
- -ss restricts a PE's memory use to NUMA-local memory only.
- -r sets aside one or more cores for OS services.

# Multiple apruns – Run Sequentially

- A series of aprun's, running one after the other

  aprun -n 32 ./a.out

  aprun -n 64 ./b.out

  aprun –n 16 ./c.out

- Request resource accordingly

  #PBS –l nodes=2

# Mupltiple aprun's – Run Concurrently

- Runs simultaneously – important to ``wait''

```
aprun -n 32 ./a.out &
aprun -n 64 ./b.out &
aprun -n 16 ./c.out &
wait
```

- Request resource accordingly

```
#PBS –l node=4
```

# Mupltiple apruns – MPMD

- MPMD

  aprun -n 256 ./a.out : -n 768 ./b.out
- Executables share `MPI_COMM_WORLD`
- Nodes needed

  #PBS –l nodes=32

Note, there can be no more than one set of aprun arguments per node.

# Single and Dual Stream

- Single – place one PE on one floating point unit (FPU)
- Daul – place one PE on one integer core, two PE's share a FPU, the default
- Considerations – the amount of floating point operations in the application, the amount of memory needed per PE …

# Queues and charging

| Queue | Priority | Changing Factor |
|---|---|---|
| debug | 1 | 1 |
| high | 2 | 2 |
| normal | 3 | 1 |
| low | 4 | 0.5 |

Note: low queue jobs are preemptable - subject to early termination by the scheduler.

# Batch job stderr/stdout

- Redirect application stdout/stderr to a file (preferably on scratch) rather than letting the batch software capture it.

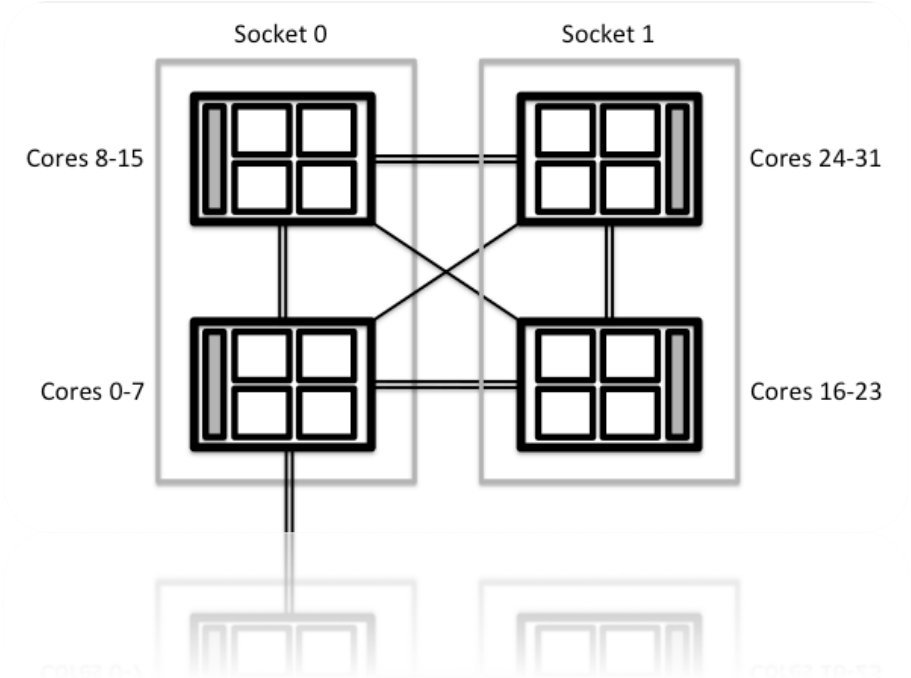
```
> cd /scratch/…
> aprun -n … > outerr.$PBS_JOBID 2>&1 (bash)
```

- Easier to track job progress.
- Avoid excessive stdout and stderr from applications as task count increases. Consider reporting by rank 0 or a few ranks.

# File Systems

- There are three file systems:  /u, /project, and /scratch.
- Scratch is the largest and fastest (1440 OSTs). Jobs should be launched from scratch.
- All three file systems are visible from compute nodes.
- Purging of scratch will be threshold based with a 2 week age based policy.
- Current issue limits stripe count to 160 OSTs.
- For a single OST, there is a 2 TB file limit.
- More in the I/O talk later in the workshop.

# Setting Process Affinity – BW XE node




- 32 integer cores
- 16 FPU's
- 4 NUMA nodes
- 2 sockets

# Setting Process Affinity – pure MPI code

Assume XE nodes are used, then:

- `aprun -n 64`

  It places 32 mpi processes on a XE node by default. Needs 2 nodes.

- `aprun -n 64 -N 8`

  It places 8 mpi processes on a XE node. However by default, all 8 processes will be on 1 NUMA node. Needs 8 nodes.

- `aprun -n 64 -N 8 -S 2`

  It places 8 mpi processes on a XE node using all 4 NUMA nodes with 2 mpi processes per NUMA. Needs 8 nodes.

# Setting Process Affinity – pure MPI code

Assume XE nodes are used in the following. The -cc option provides precise control on placements:

- `aprun -n 64 -N 8 -cc 0,1,8,9,16,17,24,25`
  It specifies actually where each of 8 mpi processes on a node will be placed. Needs 8 nodes.
- `aprun -n 64 -N 8 -cc 0,4,8,12,16,20,24,28`
  It specifies a different placement compared to the above. Needs 8 nodes.

# Setting Process Affinity – MPI+openMP

Assume XE nodes are used, then:

- ``aprun –n 32 –d 2”

  It places 16 mpi processes on a XE node by default. Each mpi has 2 threads. (Needs 2 nodes.)

- ``aprun –n 32 –N 8 –d 2”

  It places 8 mpi processes on a XE node, each with 2 threads. However by default, all 8 processes will be on 1 socket. (Needs 4 nodes.)

- ``aprun –n 32 –N 8 –S 2 –d 2”

  It places 8 mpi processes on a XE node, 2 on each NUMA using all 4 NUMA nodes (i.e. 2 sockets). Each mpi again has two threads. (Needs 4 nodes.)

# Setting Process Affinity – MPI+openMP

Assume XE nodes are used. The -cc option provides precise control on placements:

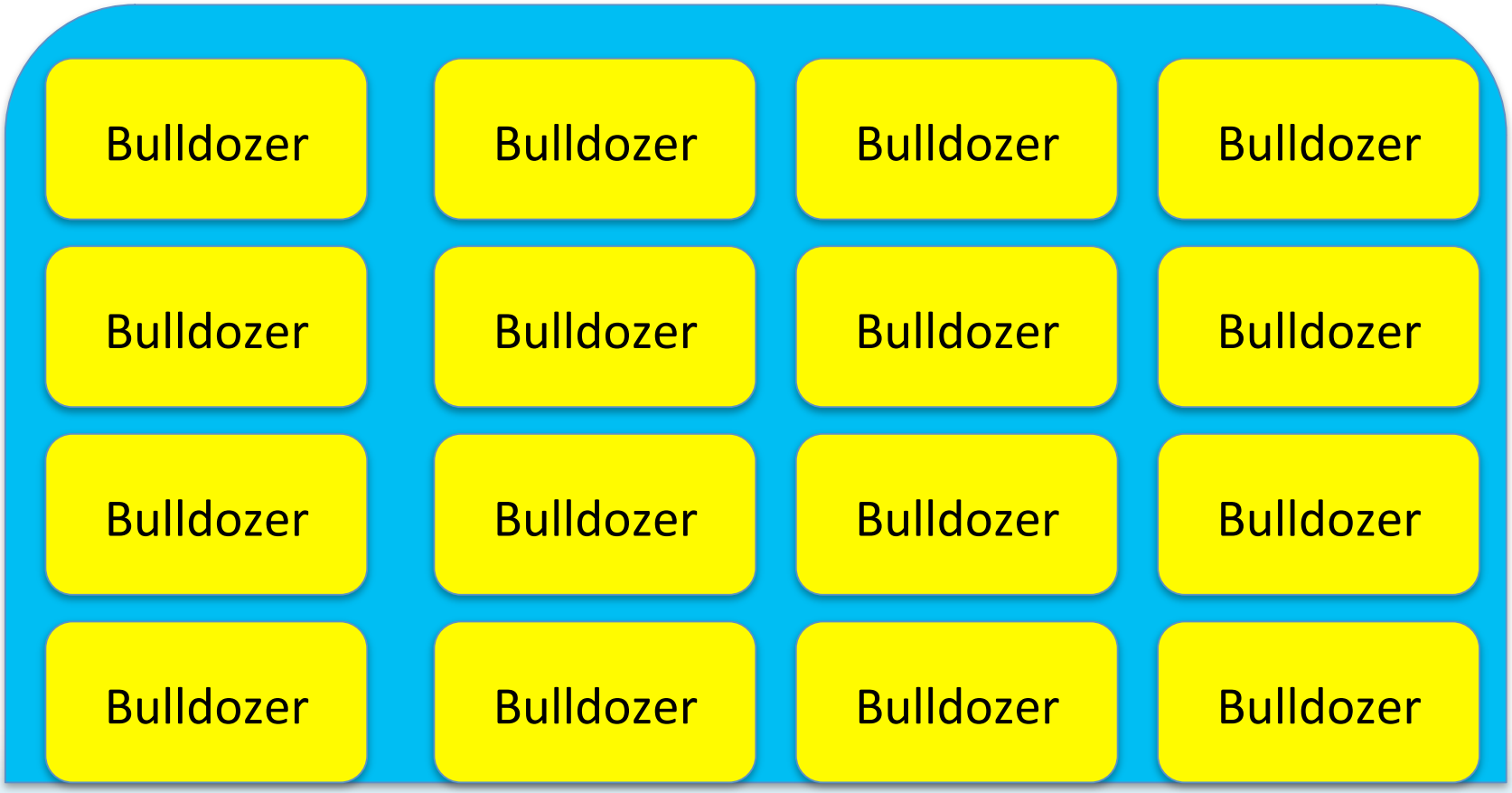
- `aprun -n 64 -cc 0,1:2,3:8,9:10,11:16,17:18,19:24,25:26,27`

  `It` puts 8 mpi processes on an XE node, with core 0,1 for 1st process and its 2 threads; core 2,3 for 2nd process and its 2 threads … 4 nodes are needed.

- `aprun -n 64 –cc 0,1:4,5:8,9:12,13:16,17:20,21:24,25:28,29`

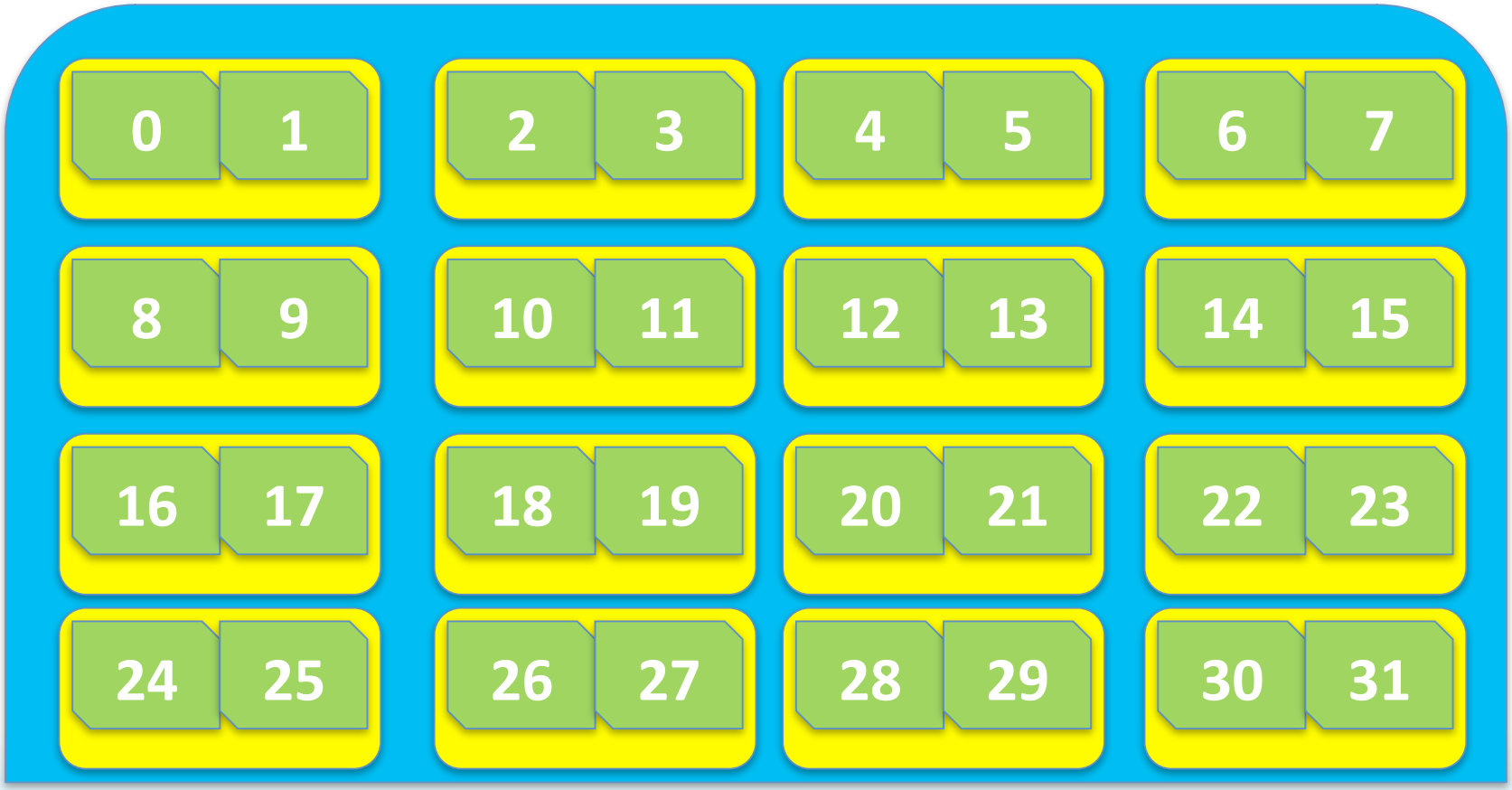
  `It` puts 8 mpi processes on an XE node, with core 0,1 for 1st process and its 2 threads; core 4,5 for 2nd process and its 2 threads … 4 nodes are needed.

# Blue Waters XE Node

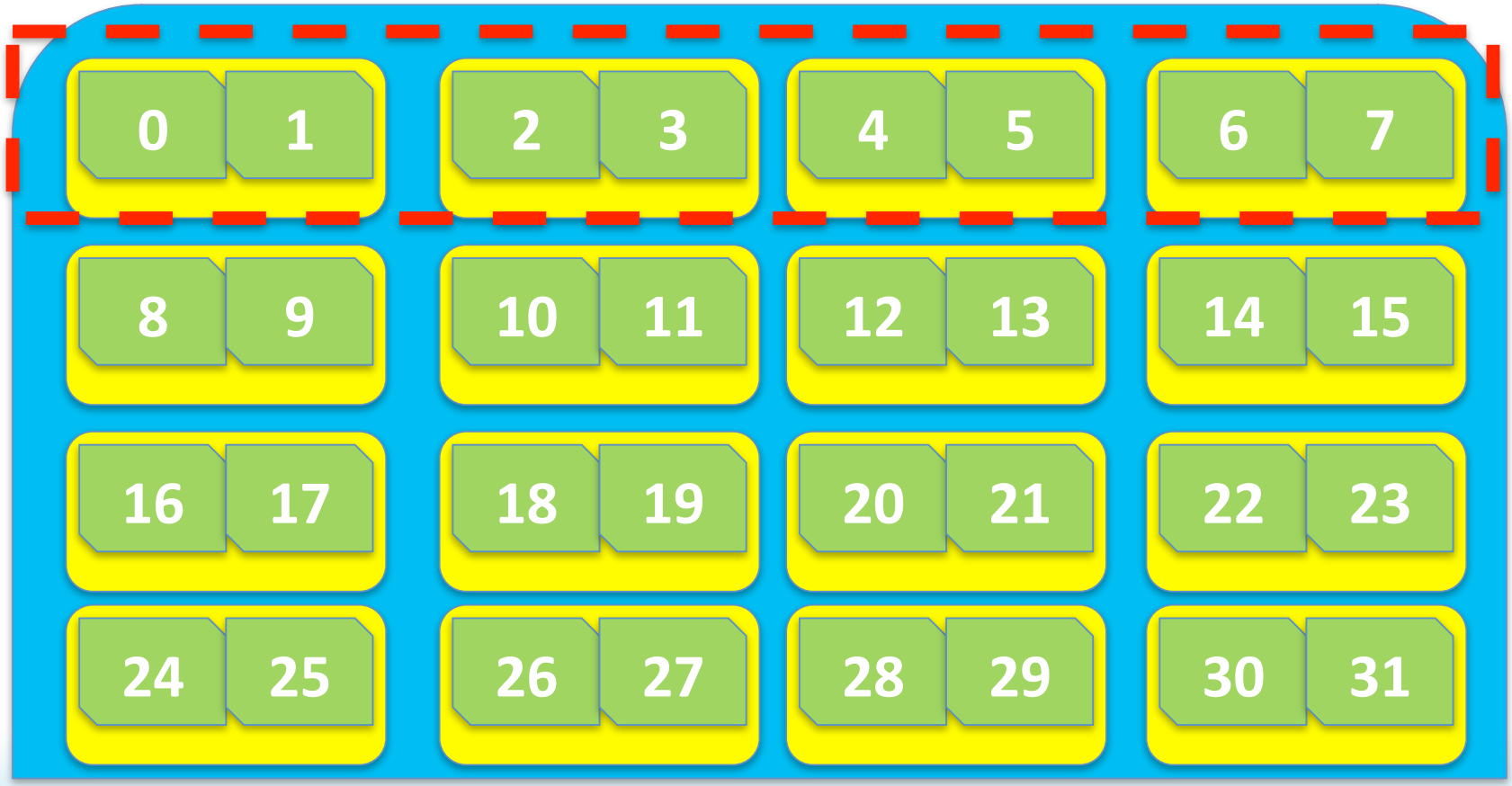


# Each "Integer Unit" is an aprun "CPU"

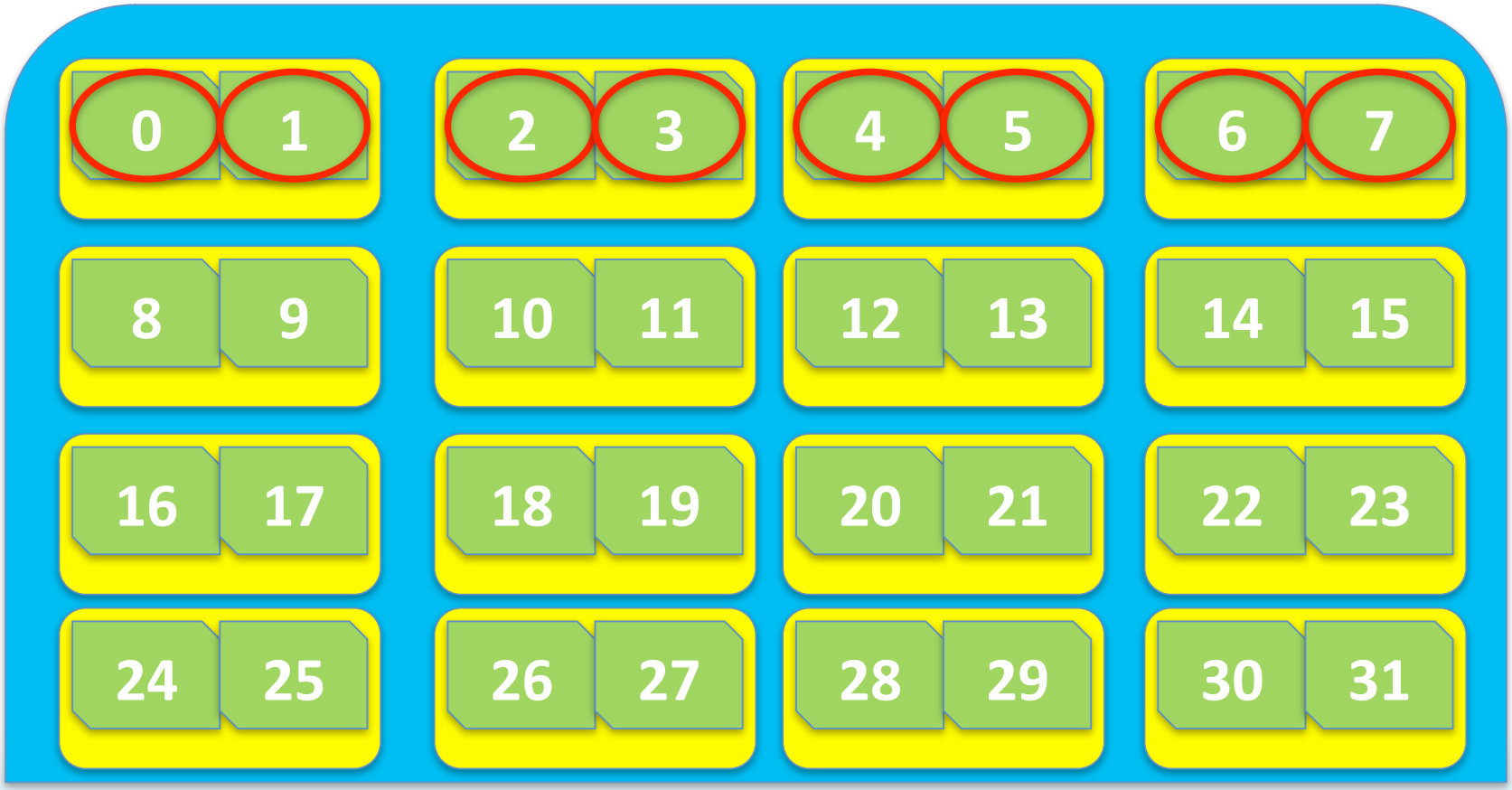
# Each "Integer Unit" is an aprun "CPU"



# Red dashed outline is "NUMA node"



# Default placement by aprun –n 8



# aprun –n 8 –S 4


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

# aprun –n 8 –d 2


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

# aprun –n 8 -cc 0,2,4,6,8,10,12,14

# Managing and monitoring jobs

System commands

- qstat, qdel, showq, showres – torque and moab
- apstat, xtnodestat – alps

Scripts

- qpeek – look at job stdout/stderr
- apstat_system.pl – displays system status by node type
- qstat.pl – displays qstat output with node type and count
- showqgpu.pl - displays only XK jobs similar to showq

# Why is my job not running?

- Make sure job was submitted the way you expected: `qstat –f JOBID`
  - We employ a qsub filter that attempts to catch improperly configured requests.
- Check to see what is running with `showq` as eligible jobs are listed in a prioritized order. Using the '-i' option will show priority.
- Use '`checkjob JOBID`' to see more about the job.
- Currently have a 300 second allocation cycle.

# Common errors and error messages

- **OOM killer terminated this process**. This error message results when your application exceeds the available memory on a node.
- **Claim exceeds reservation's node-count**. This error message results when the combination of **PBS nodes** and **aprun** options (for example, –N, –S, –ss, –sn, –m) requires more nodes than were reserved for you by the **qsub** command.).
- …

# Programming Environment

- Use the module command to manage programming environment
- Paths, libraries, etc, will be properly set once a programing environment is set, e.g.

  module load PrgEnv-pgi

- Compiler wrappers ftn, cc, CC, etc, will use the corresponding compilers, the correct include files, and library paths.