CLE and How to Start Your Application

Jan Thorbecke

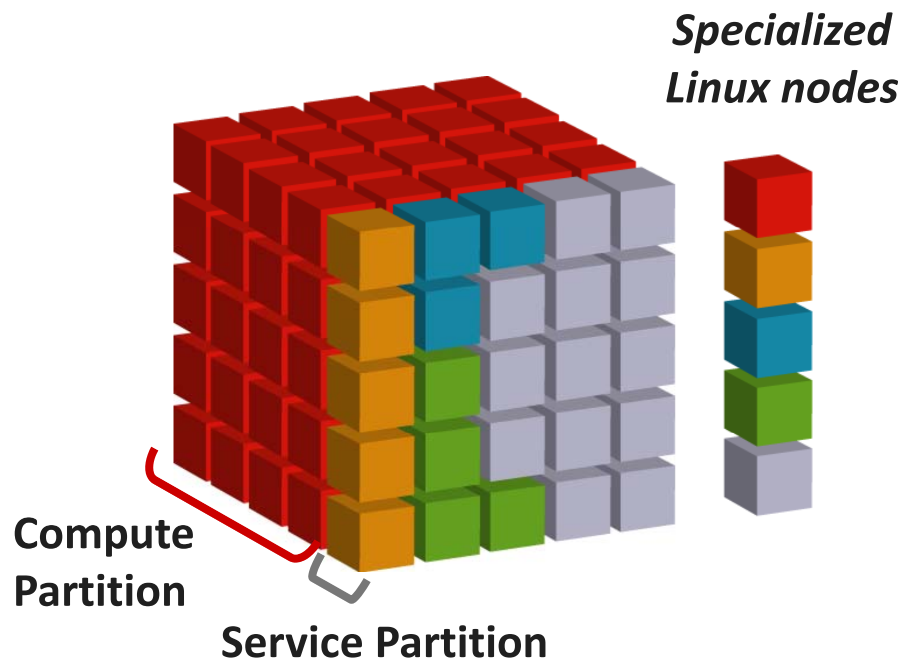
CRAY
THE SUPERCOMPUTER COMPANY

CSCS
Swiss National Supercomputing Centre

HP2C

Scalable Software Architecture

Scalable Software Architecture: Cray Linux Environment (CLE)



Microkernel on Compute nodes, full featured Linux on Service nodes.

Service PEs specialize by function

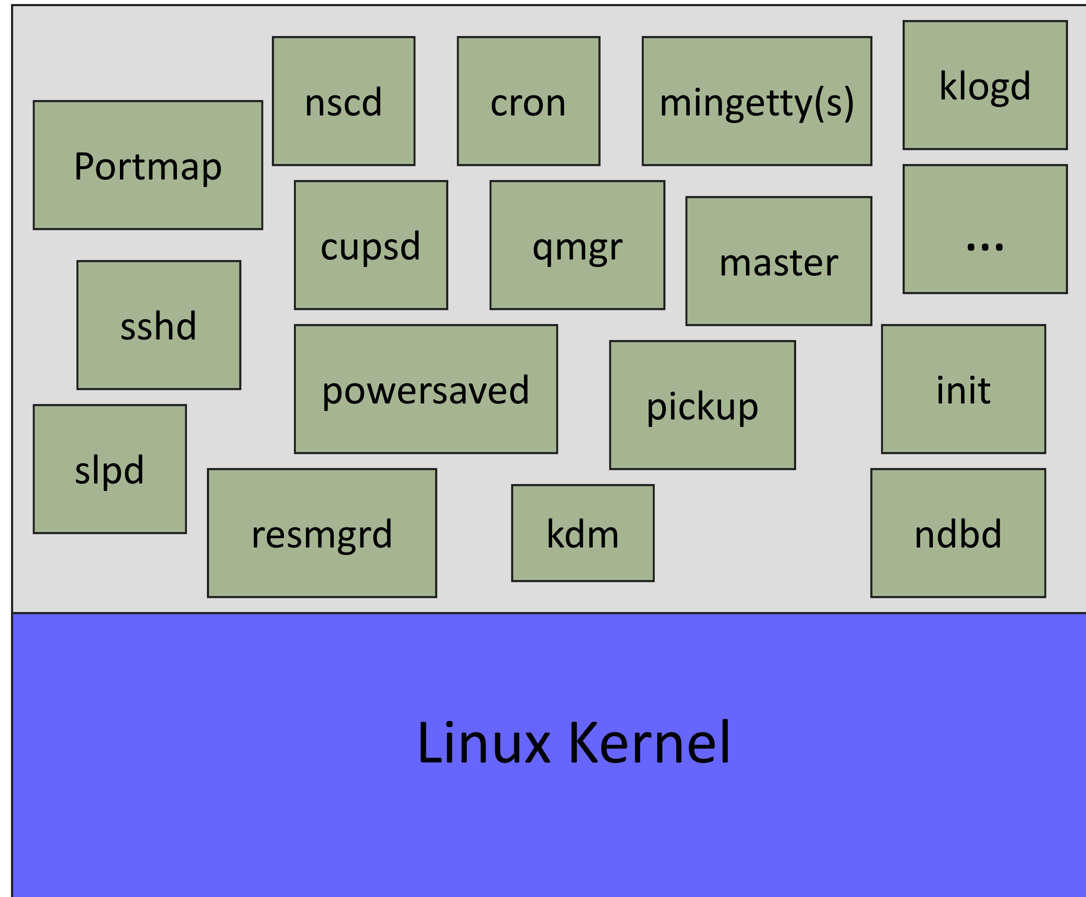
Software Architecture eliminates OS "Jitter"

Software Architecture enables reproducible run times

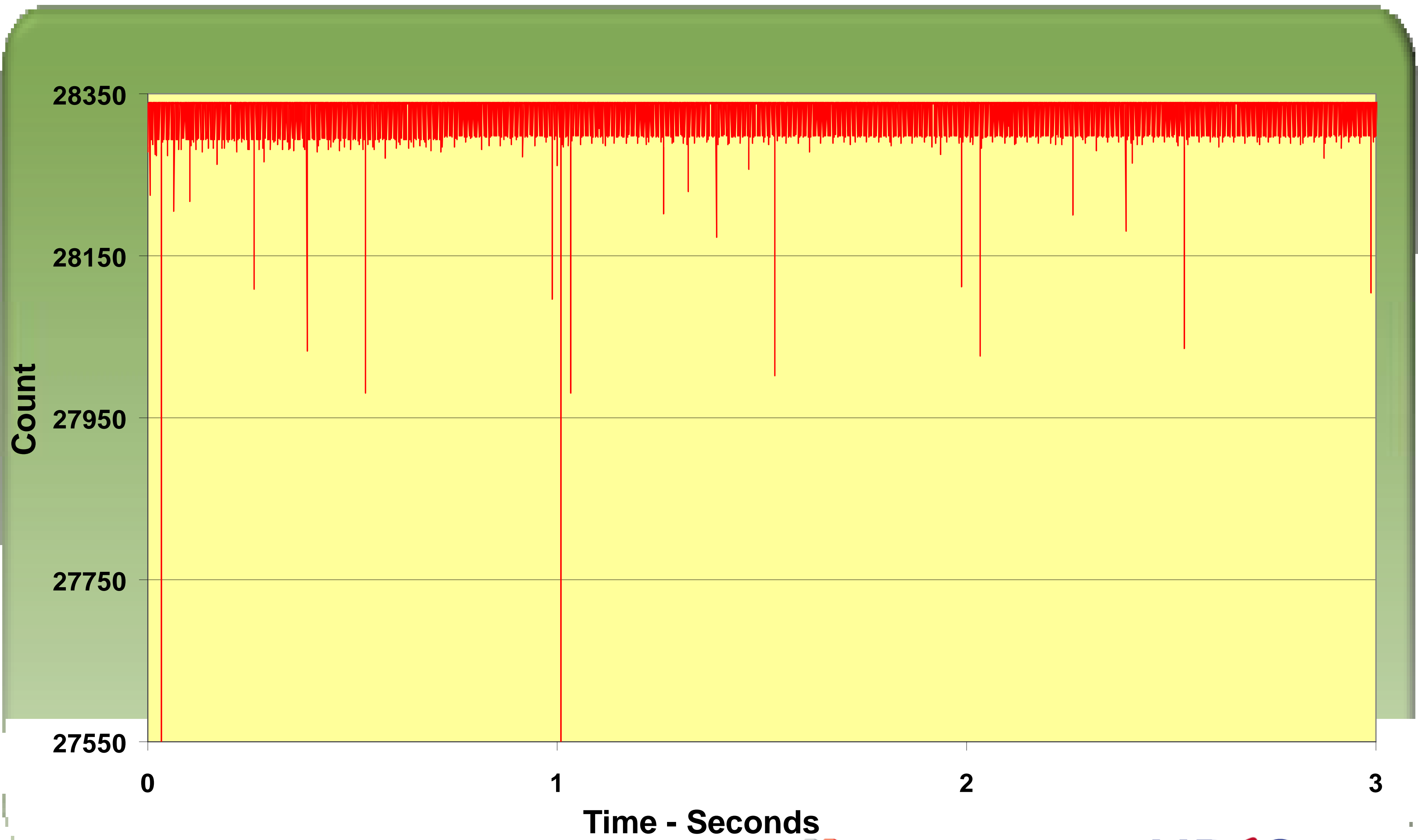
Service nodes

- Overview
  - Run full Linux (SuSe SLES), 2 nodes per service blade
- Boot node
  - first XE6 node to be booted: boots all other components
- System Data Base (SDB) node
  - hosts MySQL database
  - processors, allocation, accounting, PBS information
- Login nodes
  - User login and code preparation activities: compile, launch
  - Partition allocation: ALPS (Application Level Placement Scheduler)

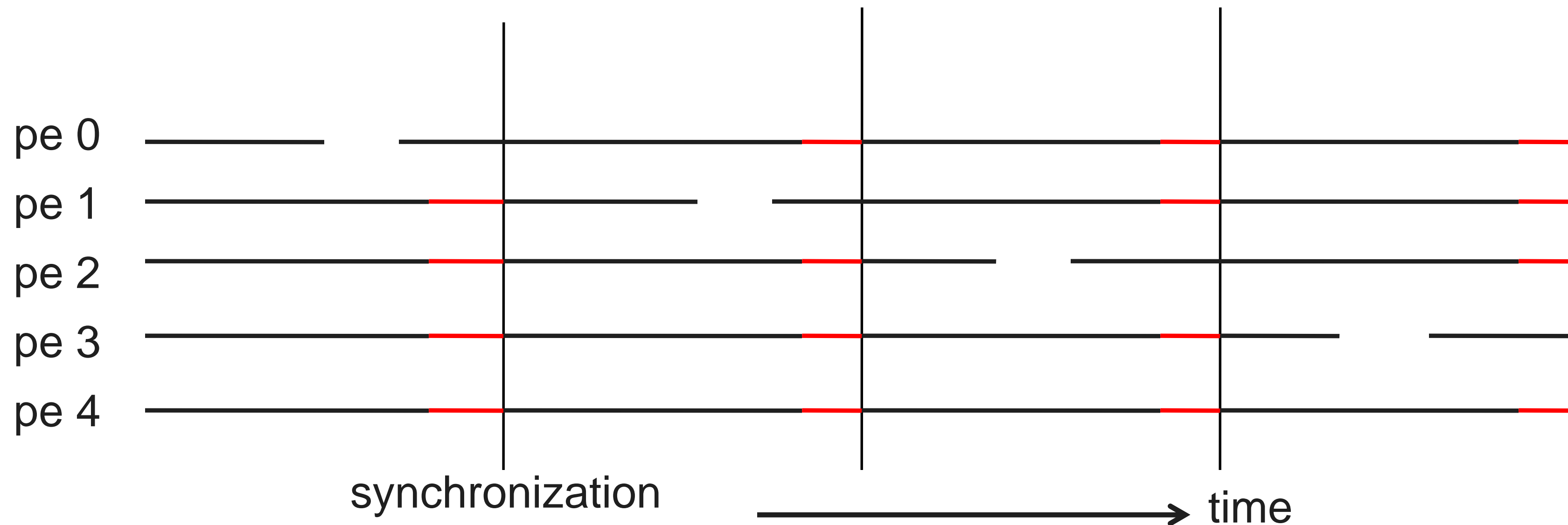
Trimming OS – *Standard Linux Server*



FTQ Plot of Stock SuSE (most daemons removed)



noise amplification in a parallel program

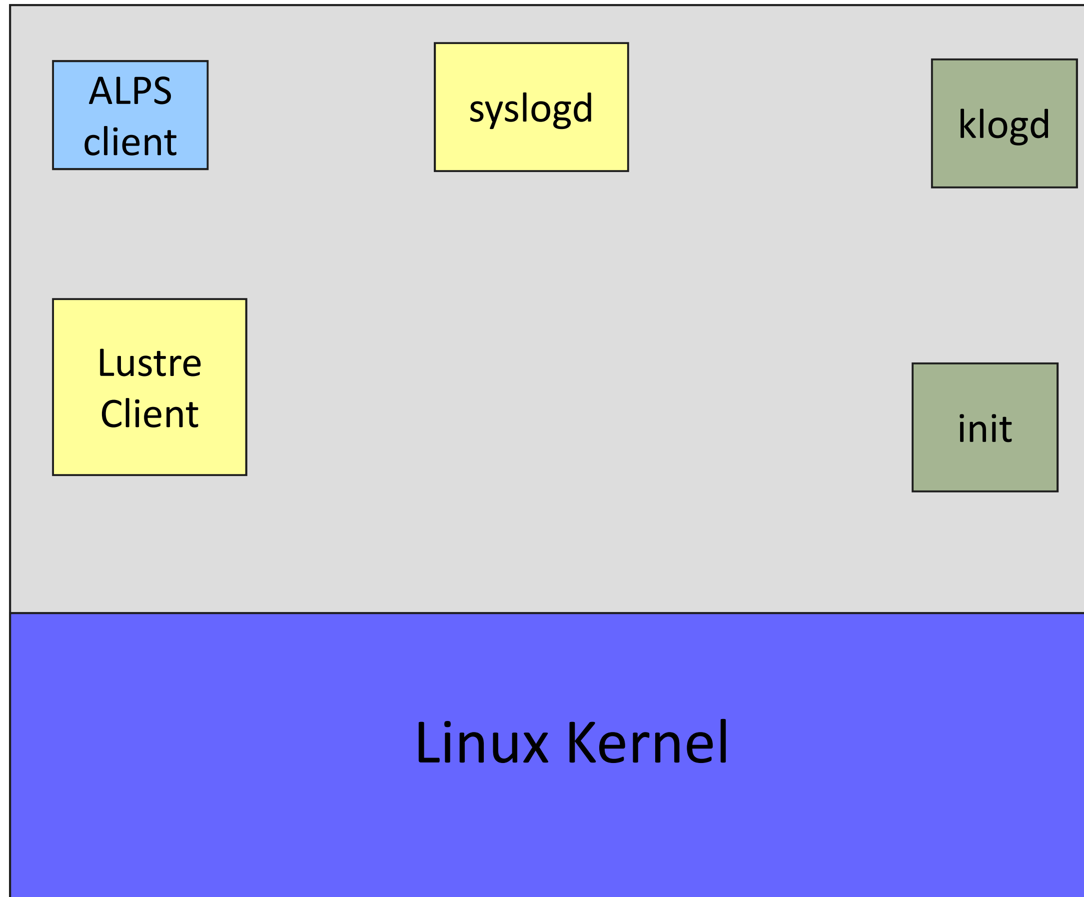


•In each synchronization interval, one rank experiences a noise delay
•Because the ranks synchronize, all ranks experience all of the delays
•In this worst-case situation, the performance degradation due to the noise is multiplied by the number of ranks
•Noise events that occur infrequently on any one rank (core) occur frequently in the parallel job

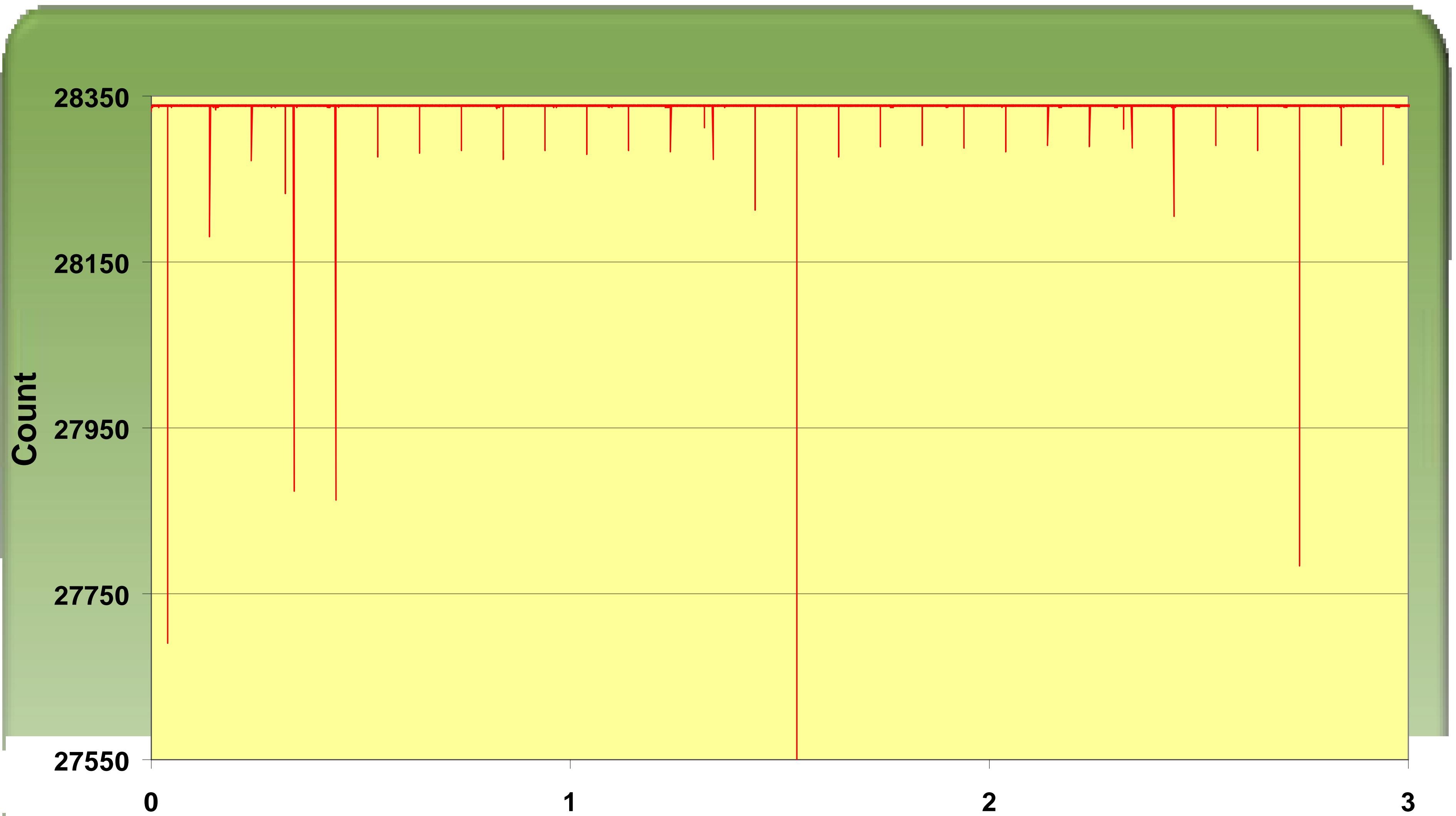
Synchronization overhead *amplifies* the noise

CRAY THE SUPERCOMPUTER COMPANY CSCS Swiss National Supercomputing Centre HP2C

Linux on a Diet – *CNL*



FTQ plot of CNL



CRAY THE SUPERCOMPUTER COMPANY

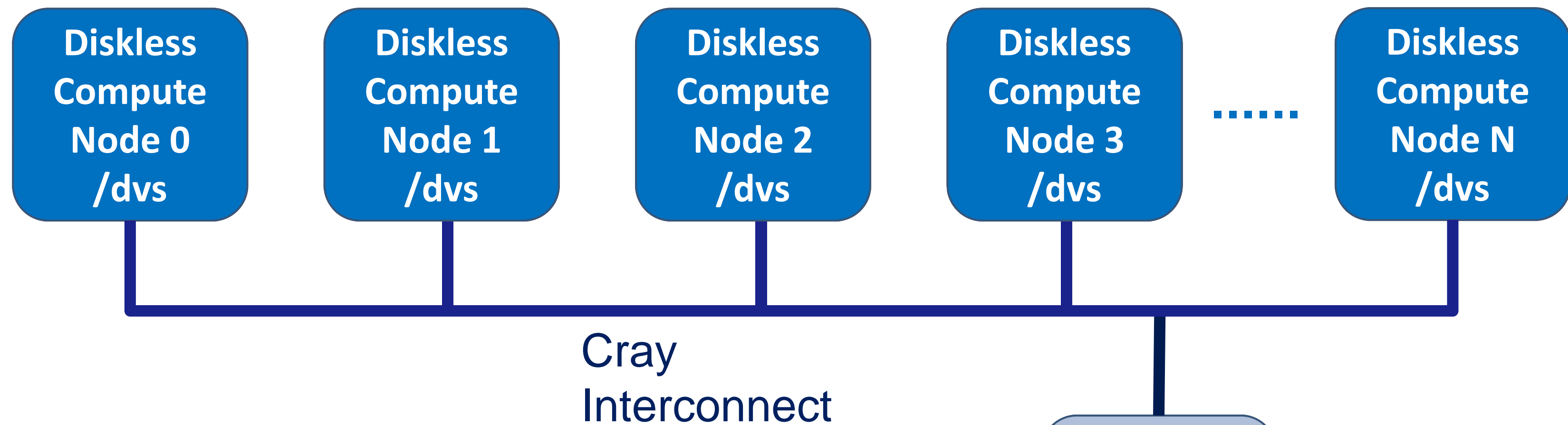
CSCS Swiss National Supercomputing Centre

HP2C

Cray XE I/O architecture

- All I/O is offloaded to service nodes
- Lustre
  - High performance parallel I/O file system
  - Direct data transfer between compute nodes and files
- DVS
  - Virtualization service
  - Allows compute nodes to access NFS mounted on service node
  - Applications must execute on file systems mounted on compute nodes
- No local disks
- /tmp is a MEMORY file system, on each login node

Scaling Shared Libraries with DVS



- Requests for shared libraries (.so files) are routed through DVS Servers
- Provides similar functionality as NFS, but scales to 1000s of compute nodes
- Central point of administration for shared libraries
- DVS Servers can be “re-purposed” compute nodes

DVS Server Node 0

NFS Shared Libraries

Running an Application on the Cray XE6

CRAY THE SUPERCOMPUTER COMPANY

CSCS Swiss National Supercomputing Centre

HP2C

Running an application on the Cray XE
ALPS + aprun

- ALPS : Application Level Placement Scheduler
- aprun is the ALPS application launcher
  - It **must** be used to run application on the XE compute nodes
  - If aprun is not used, the application is launched on the login node (and will most likely fail)
  - aprun man page contains several useful examples
  - at least 3 important parameters to control:
    - The total number of PEs : -n
    - The number of PEs per node: -N
    - The number of OpenMP threads: -d
      More precise : The 'stride' between 2 PEs in a node

CRAY THE SUPERCOMPUTER COMPANY

CSCS Swiss National Supercomputing Centre

HP2C

Job Launch

XE6 User

Login Node

SDB Node

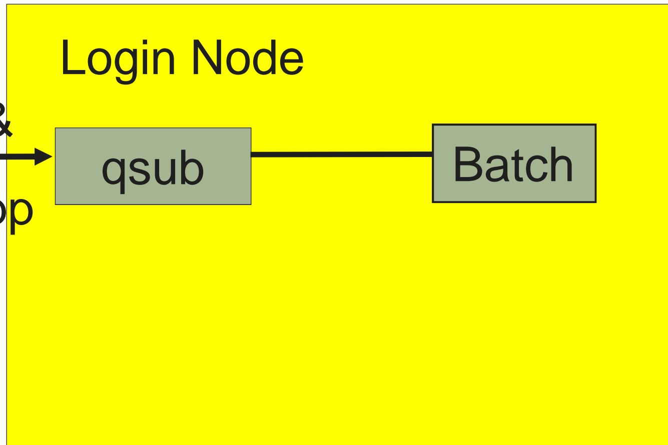
Compute Nodes

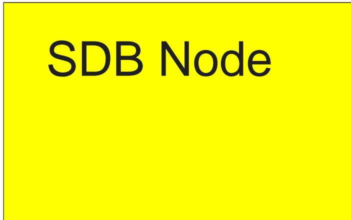
Job Launch

XE6 User

Login &

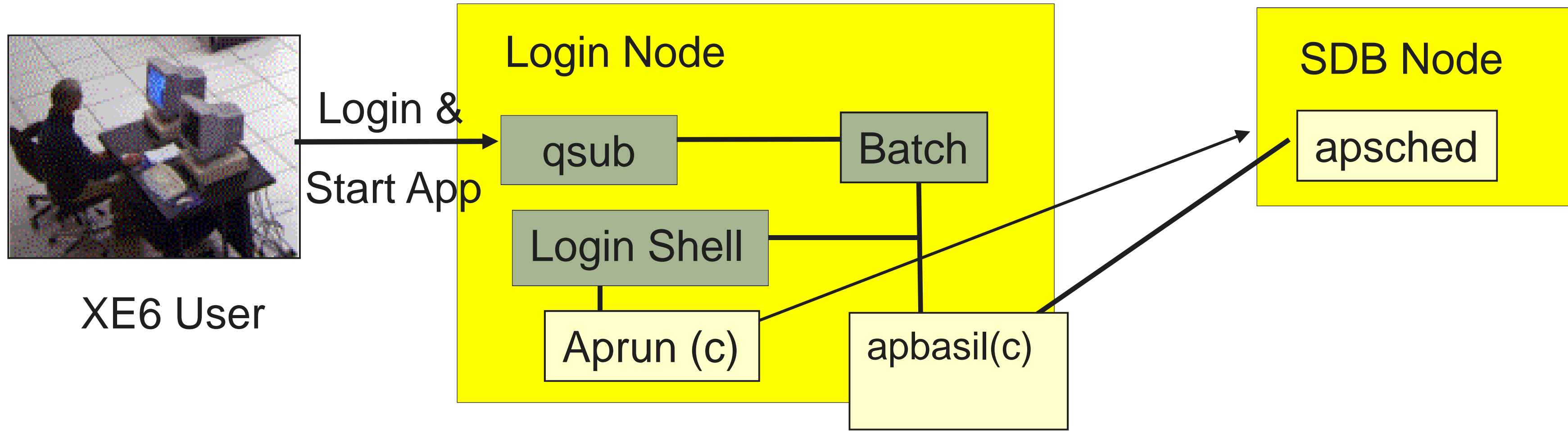
Start App





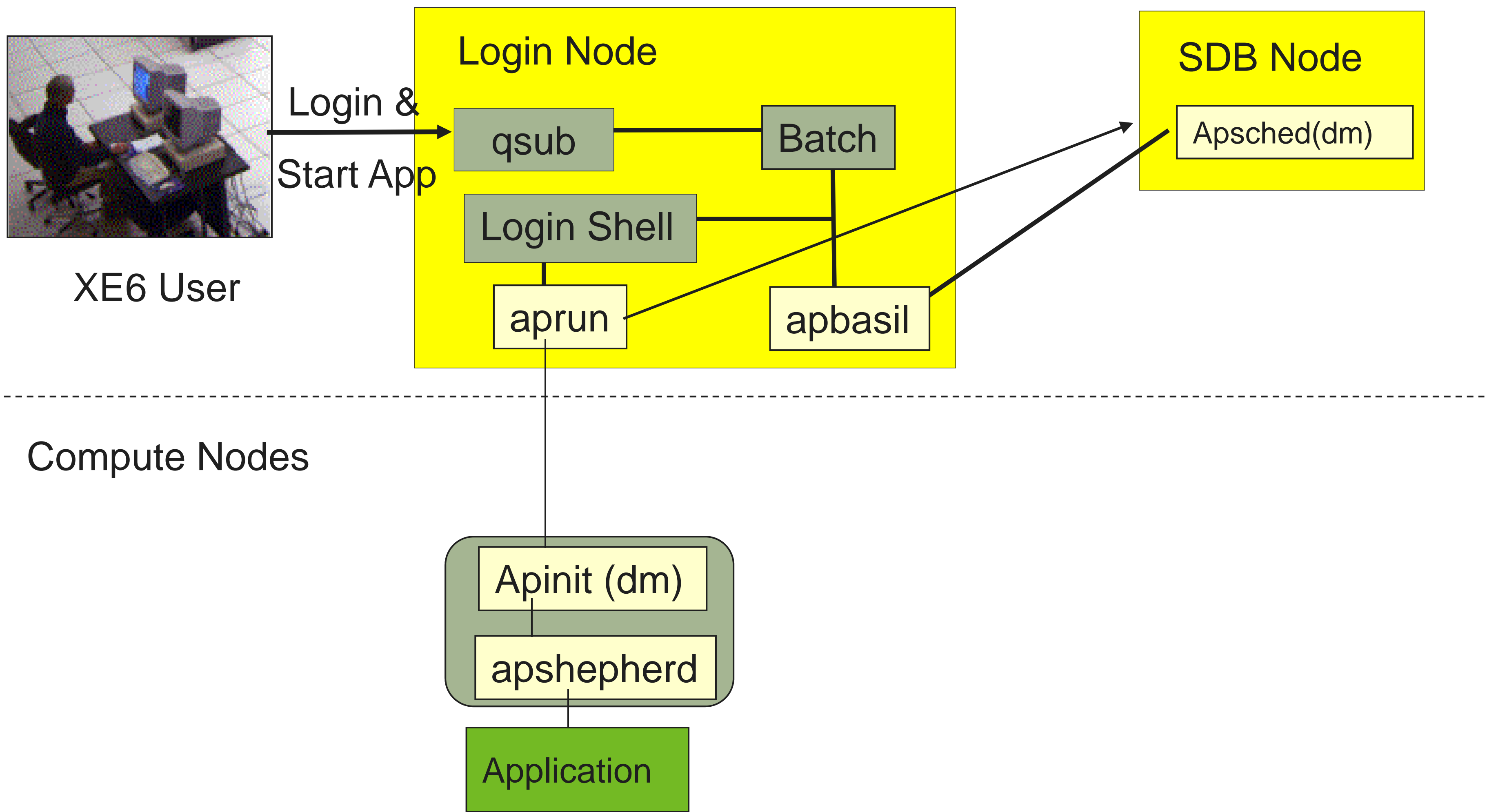
Compute Nodes

Job Launch



Compute Nodes

Job Launch

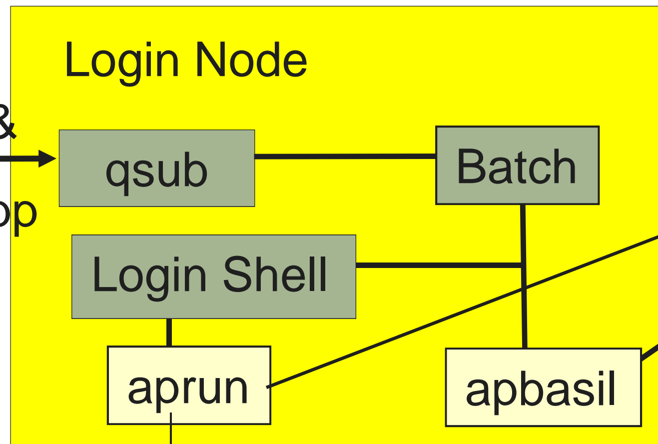


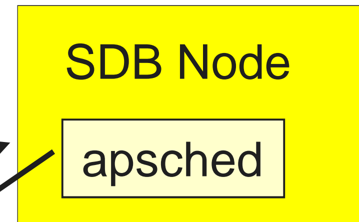
Job Launch

XE6 User

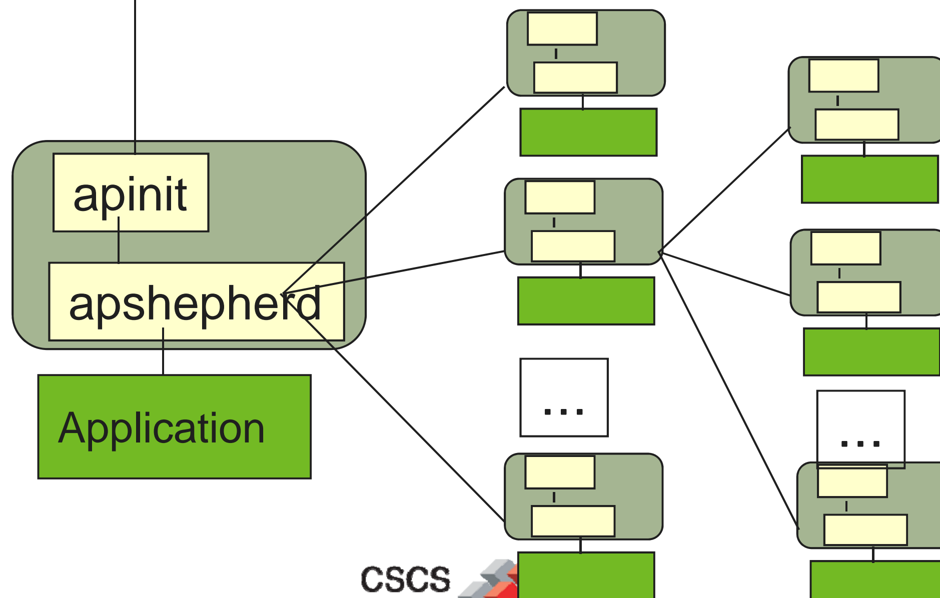
Login &

Start App





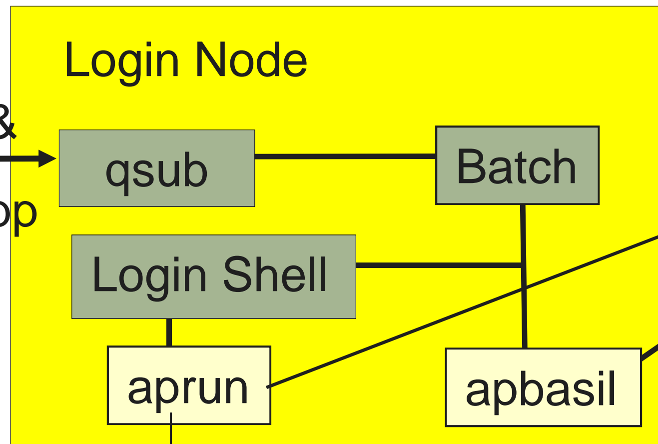
Compute Nodes

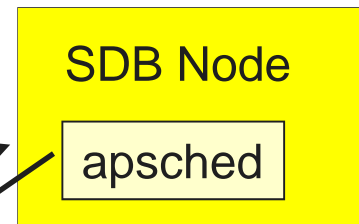


Job Launch

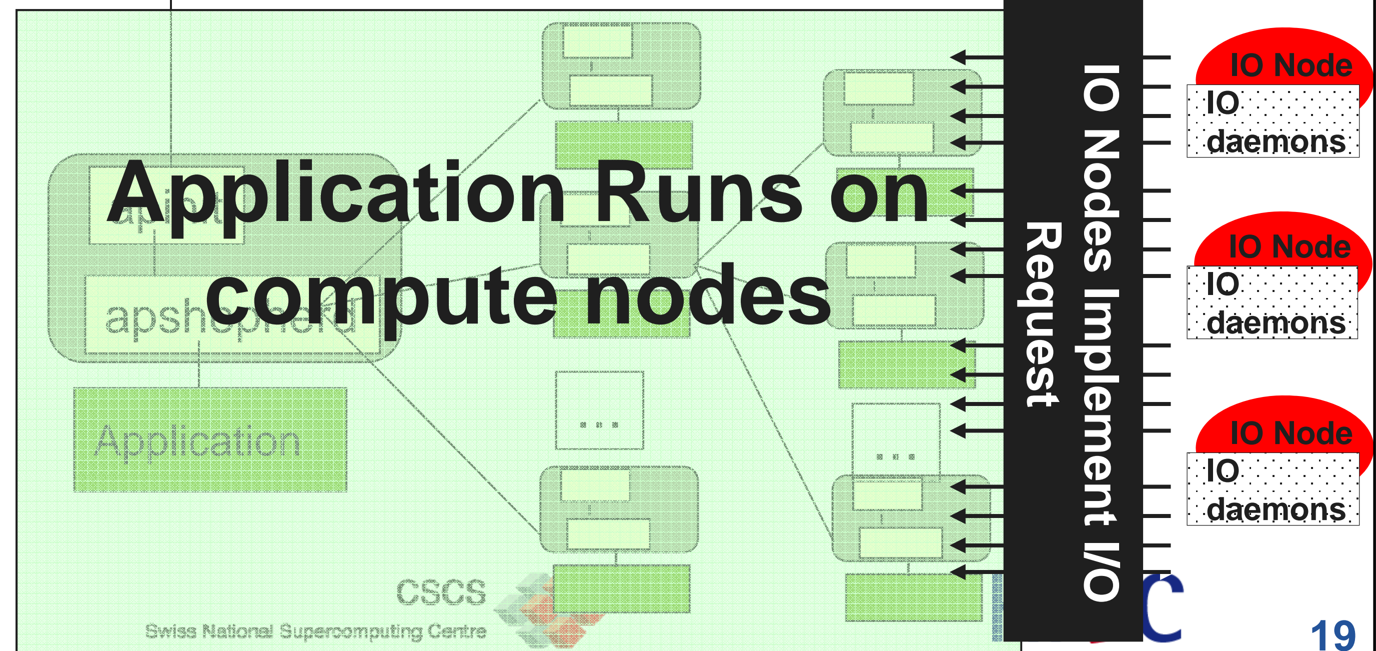
XE6 User

Login & Start App



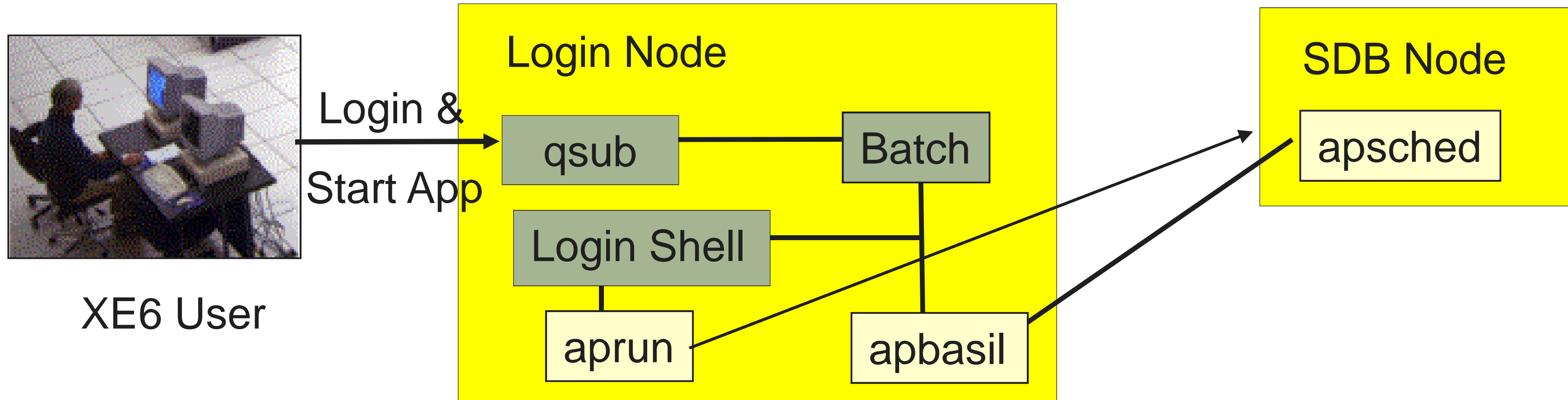


Compute Nodes



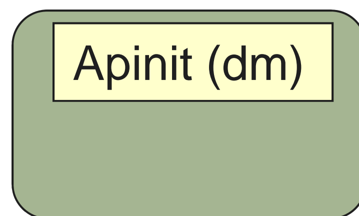
CRAY THE SUPERCOMPUTER COMPANY

Job Launch

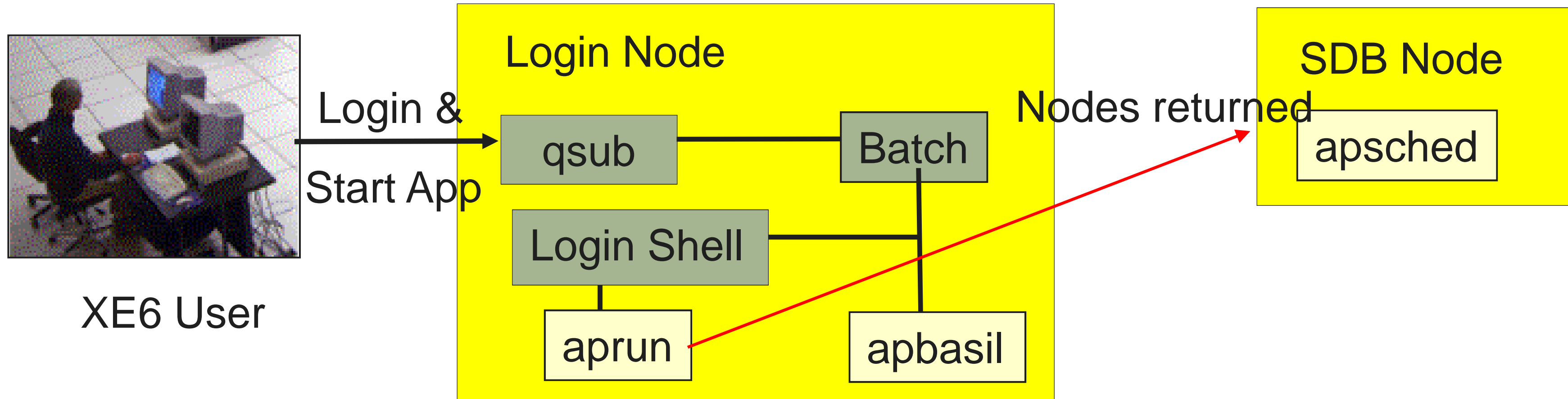


Compute Nodes

Job is
cleaned up

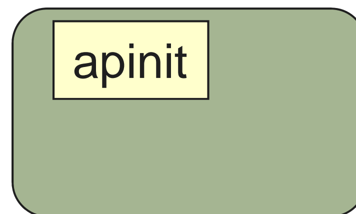


Job Launch



Compute Nodes

Job is
cleaned up



Job Launch : Done

XE6 User

Login Node

SDB Node

Compute Nodes

Some Definitions

- ALPS is always used for scheduling a job on the compute nodes. It does not care about the programming model you used. So we need a few general 'definitions' :
  - PE : Processing Elements
    Basically an Unix 'Process', can be a MPI Task, CAF image, UPC tread, …
  - Numa_node
    The cores and memory on a node with 'flat' memory access, basically one of the 4 Dies on the Interlagos processor and the direct attach memory.
  - Thread
    A thread is contained inside a process. Multiple threads can exist within the same process and share resources such as memory, while different PEs do not share these resources.
    Most likely you will use OpenMP threads.

Running an application on the Cray XE6
some basic examples

- Assuming a XE6 IL16 system (32 cores per node)
- Pure MPI application, using all the available cores in a node

```
$ aprun -n 32 ./a.out
```

- Pure MPI application, using only 1 core per node
  - 32 MPI tasks, 32 nodes with 32*32 cores allocated
  - Can be done to increase the available memory for the MPI tasks

```
$ aprun -N 1 -n 32 -d 32 ./a.out
    (we'll talk about the need for the -d32 later)
```

- Hybrid MPI/OpenMP application, 4 MPI ranks per node
  - 32 MPI tasks, 8 OpenMP threads each
  - need to set OMP_NUM_THREADS

```
$ export OMP NUM THREADS=8
$ aprun -n 32 -N 4 -d $OMP_NUM_THREADS
```

aprun CPU Affinity control

- CNL can **dynamically** distribute work by allowing PEs and threads to migrate from one CPU to another within a node
- In some cases, moving PEs or threads from CPU to CPU increases cache and Translation Lookaside Buffer (TLB) misses and therefore **reduces** performance
- CPU affinity options enable to bind a PE or thread to a particular CPU or a subset of CPUs on a node
- aprun CPU affinity option (see man aprun)
  - Default settings : -cc cpu
    PEs are bound a to specific core, depended on the –d setting
  - Binding PEs to a specific numa node : -cc numa_node
    PEs are not bound to a specific core but cannot 'leave' their numa_node
  - No binding : -cc none
  - Own binding : -cc 0,4,3,2,1,16,18,31,9,…

Memory affinity control

- Cray XE6 systems use dual-socket compute nodes with 4 dies
  - Each die (8 cores) is considered a NUMA-node
- **Remote-NUMA-node memory references**, can adversely affect performance.
  Even if you PE and threads are bound to a specific numa_node, the memory used does not have to be 'local'
- aprun memory affinity options (see man aprun)
  - Suggested setting is –ss
    a PE can only allocate the memory local to its assigned NUMA node. If this is not possible, your application will crash.

Running an application on the Cray XE - MPMD

- aprun supports MPMD – Multiple Program Multiple Data

- Launching several executables on the same MPI_COMM_WORLD
```
$ aprun -n 128 exe1 : -n 64 exe2 : -n 64 exe3
```

- Notice : Each exacutable needs a dedicated node, exe1 and exe2 cannot share a node.

  Example : The following commands needs 3 nodes
```
$ aprun -n 1 exe1 : -n 1 exe2 : -n 1 exe3
```

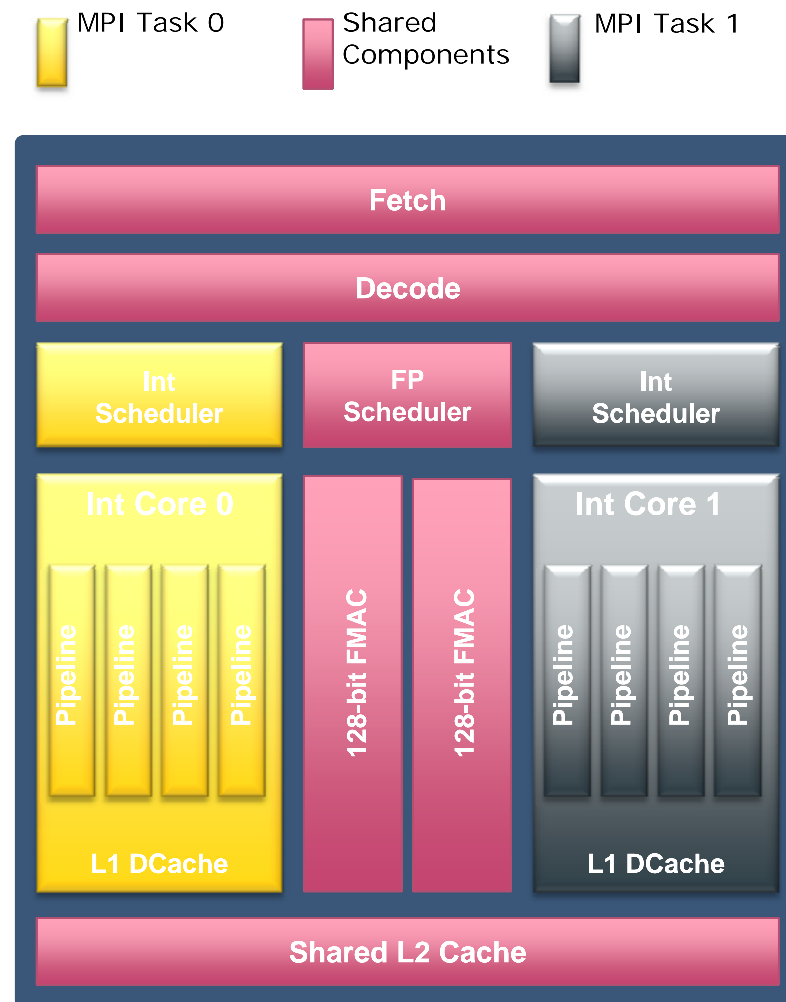
- Use a script to start several serial jobs on a node :
```
$ aprun -a xt -n 3 script.sh
```

```
>cat script.sh
./exe1&
./exe2&
./exe3&
wait
>
```

How to use the interlagos 1/3
1 MPI Rank on Each Integer Core Mode

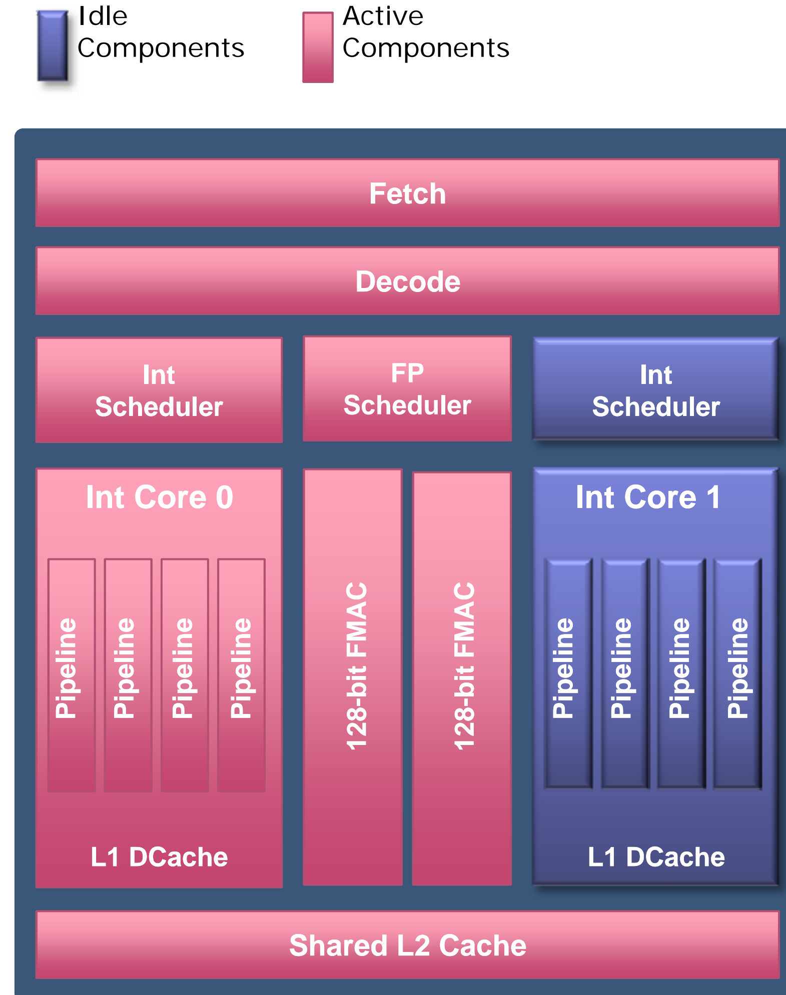
- In this mode, an MPI task is pinned to each integer core
- Implications
  - Each core has exclusive access to an integer scheduler, integer pipelines and L1 Dcache
  - The 256-bit FP unit and the L2 Cache is shared between the two cores
  - 256-bit AVX instructions are dynamically executed as two 128-bit instructions if the 2nd FP unit is busy
- When to use
  - Code is highly scalable to a large number of MPI ranks
  - Code can run with 1 GB per core memory footprint (or 2 GB on 64 GB node)
  - Code is not well vectorized

MPI Task 0 | Shared Components | MPI Task 1



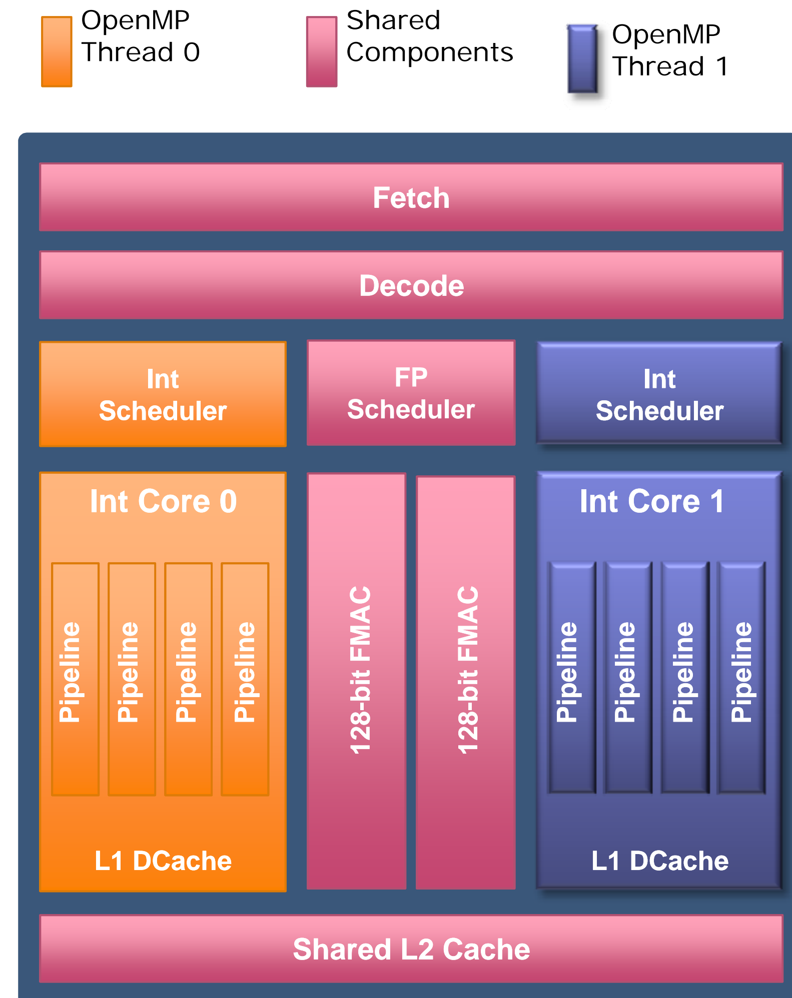
How to use the interlagos 2/3
Wide AVX mode

- In this mode, only one integer core is used per core pair
- Implications
  - This core has *exclusive* access to the 256-bit FP unit and is capable of 8 FP results per clock cycle
  - The core has twice the memory capacity and memory bandwidth in this mode
  - The L2 cache is effectively twice as large
  - The peak of the chip is not reduced
- When to use
  - Code is highly vectorized and makes use of AVX instructions
  - Code needs more memory per MPI rank



How to use the interlagos 3/3
2-way OpenMP Mode

- In this mode, an MPI task is pinned to a core pair
- OpenMP is used to run a thread on each integer core
- Implications
  - Each OpenMP thread has exclusive access to an integer scheduler, integer pipelines and L1 Dcache
  - The 256-bit FP unit and the L2 Cache is shared between the two threads
  - 256-bit AVX instructions are dynamically executed as two 128-bit instructions if the 2nd FP unit is busy
- When to use
  - Code needs a large amount of memory per MPI rank
  - Code has OpenMP parallelism exposed in each MPI rank



CRAY THE SUPERCOMPUTER COMPANY

CSCS Swiss National Supercomputing Centre

HP2C

Aprun: cpu_lists for each PE

- CLE was updated to allow threads and processing elements to have more flexibility in placement. This is ideal for processor architectures whose cores share resources with which they may have to wait to utilize.
  Separating cpu_lists by colons (:) allows the user to specify the cores used by processing elements and their child processes or threads.
  Essentially, this provides the user more granularity to specify cpu_lists for each processing element.
  Here an example with 3 threads :
  aprun -n 4 -N 4 -cc 1,3,5:7,9,11:13,15,17:19,21,23
- Note: This feature will be modified in CLE 4.0.UP03, however this option will still be valid.

-cc detailed example

```
export OMP_NUM_THREADS=2

aprun -n 4 -N 2 -d 2 -cc 0,2:4,6:8,10:12,14 ./acheck_mpi
nid00028[  0] on cpu 00 affinity for thread  0 is: cpu   0, mask 1
nid00028[  0] on cpu 02 affinity for thread  1 is: cpu   2, mask 001
nid00028[  1] on cpu 04 affinity for thread  0 is: cpu   4, mask 00001
nid00028[  1] on cpu 06 affinity for thread  1 is: cpu   6, mask 0000001
nid00572[  2] on cpu 00 affinity for thread  0 is: cpu   0, mask 1
nid00572[  2] on cpu 02 affinity for thread  1 is: cpu   2, mask 001
nid00572[  3] on cpu 04 affinity for thread  0 is: cpu   4, mask 00001
nid00572[  3] on cpu 06 affinity for thread  1 is: cpu   6, mask 0000001
Application 3024546 resources: utime ~2s, stime ~0s
aprun -n 4 -N 4 -d 2 -cc 0,2:4,6:8,10:12,14 ./acheck_mpi

nid00028[  0] on cpu 00 affinity for thread  0 is: cpu   0, mask 1
nid00028[  2] on cpu 08 affinity for thread  0 is: cpu   8, mask 000000001
nid00028[  0] on cpu 02 affinity for thread  1 is: cpu   2, mask 001
nid00028[  1] on cpu 06 affinity for thread  1 is: cpu   6, mask 0000001
nid00028[  3] on cpu 12 affinity for thread  0 is: cpu  12, mask 0000000000001
nid00028[  2] on cpu 10 affinity for thread  1 is: cpu  10, mask 00000000001
nid00028[  1] on cpu 04 affinity for thread  0 is: cpu   4, mask 00001
nid00028[  3] on cpu 14 affinity for thread  1 is: cpu  14, mask 000000000000001
Application 3024549 resources: utime ~0s, stime ~0s
```

Running a batch application with Torque

- The number of required nodes and cores is determined by the parameters specified in the job header

```
#PBS -l mppwidth=256 (MPP width: number of PE's )
#PBS -l mppnppn=4    (MPP number of PE's per node)
```

  This example uses 256/4=64 nodes
- The job is submitted by the **qsub** command
- At the end of the execution output and error files are returned to submission directory
- PBS environment variable: $PBS_O_WORKDIR
  Set to the directory from which the job has been submitted
  Default is $HOME
- man qsub for env. variables

Other Torque options

- #PBS -N job_name

  the job name is used to determine the name of job output and error files

- #PBS -l walltime=hh:mm:ss

  Maximum job elapsed time

  should be indicated whenever possible: this allows Torque to determine best scheduling startegy

- #PBS -j oe

  job error and output files are merged in a single file

- #PBS -q queue

  request execution on a specific queue

Torque and aprun

Torque	aprun	
-lmppwidth=$PE	-n $PE	Number of PE to start
-lmppdepth=$threads	-d $threads	#threads/PE
-lmppnppn=$N	-N $N	#(PEs per node)
<none>	-S $S	#(PEs per numa_node)
-lmem=$size	-m $size[h|hs]	per-PE required memory

- -B will provide aprun with the Torque settings for –n,-N,-d and –m
  aprun –B ./a.out

- Using –S can produce problems if you are not asking for a full node.
If possible, ALPS will only give you access to a parts of a node if the Torque settings allows this. The following will fail :
  - PBS -lmppwidth=4 ! Not asking for a full node
  - aprun –n4 –S1 … ! Trying to run on every die
- Solution is to ask for a full node, even if aprun doesn‘t use it

Core specialization

- System 'noise' on compute nodes may significantly degrade scalability for some applications
- Core Specialization can mitigate this problem
  - 1 core per node will be dedicated for system work (service core)
  - As many system interrupts as possible will be forced to execute on the service core
  - The application will not run on the service core
- Use aprun -r to get core specialization

```
$ aprun -r -n 100 a.out
```

- apcount provided to compute total number of cores required

```
$ qsub -l mppwidth=$(apcount -r 1 1024 16)job
  aprun -n 1024 -r 1 a.out
```

Running a batch application with Torque

- The number of required nodes can be specified in the job header
- The job is submitted by the **qsub** command
- At the end of the exection output and error files are returned to submission directory
- Environment variables are inherited by **#PBS -V**
- The job starts in the home directory. $`PBS_O_WORKDIR` contains the directory from which the job has been submitted

Hybrid MPI + OpenMP

```
#!/bin/bash
#PBS -N hybrid
#PBS -lwalltime=00:10:00
#PBS -lmppwidth=128
#PBS -lmppnppn=8
#PBS -lmppdepth=4

cd $PBS_O_WORKDIR
export OMP NUM THREADS=4
aprun -n128 -d4 -N8 a.out
```

Starting an interactive session with Torque

- An interactive job can be started by the –I argument
  - That is <capital-i>

- Example: allocate 64 cores and export the environment variables to the job (-V)

```
$ qsub -I -V -lmppwith=64
```

- This will give you a new prompt in your shell from which you can use aprun directly.
  Note that you are running on a MOM node (shared resource) if not using aprun

Watching a launched job on the Cray XE

- **xtnodestat**
  - Shows XE nodes allocation and aprun processes
  - Both interactive and PBS
- **apstat**
  - Shows aprun processes status
  - apstat overview
  - apstat –a[apid]info about all the applications or a specific one
  - apstat –n info about the status of the nodes
- Batch **qstat** command
  - shows batch jobs

Starting 512 MPI tasks (PEs)

```
#PBS -N MPIjob
#PBS -l mppwidth=512
#PBS -l mppnppn=32
#PBS -l walltime=01:00:00
#PBS -j oe

cd $PBS_O_WORKDIR

export MPICH_ENV_DISPLAY=1

export MALLOC_MMAP_MAX_=0
export MALLOC_TRIM_THRESHOLD_=536870912

aprun  -n 512 -cc cpu -ss ./a.out
```

Starting an OpenMP program, using a single node

```
#PBS -N OpenMP
#PBS -l mppwidth=1
#PBS -l mppdepth=32
#PBS -l walltime=01:00:00
#PBS -j oe

cd $PBS_O_WORKDIR

export MPICH_ENV_DISPLAY=1

export MALLOC_MMAP_MAX_=0
export MALLOC_TRIM_THRESHOLD_=536870912

export OMP_NUM_THREADS=32

aprun -n1 -d $OMP_NUM_THREADS -cc cpu -ss ./a.out
```

Starting a hybrid job single node, 4 MPI tasks, each with 8 threads

```
#PBS -N hybrid
#PBS -l mppwidth=4
#PBS -l mppnppn=4
#PBS -l mppdepth=8
#PBS -l walltime=01:00:00
#PBS -j oe

cd $PBS_O_WORKDIR

export MPICH_ENV_DISPLAY=1

export MALLOC_MMAP_MAX_=0
export MALLOC_TRIM_THRESHOLD_=536870912

export OMP_NUM_THREADS=8

aprun -n4 -N4 -d $OMP_NUM_THREADS -cc cpu -ss ./a.out
```

Starting a hybrid job
single node, 8 MPI tasks, each with 4 threads

```
#PBS -N hybrid
#PBS -l mppwidth=8
#PBS -l mppnppn=8
#PBS -l mppdepth=4
#PBS -l walltime=01:00:00
#PBS -j oe

cd $PBS_O_WORKDIR

export MPICH_ENV_DISPLAY=1

export MALLOC_MMAP_MAX_=0
export MALLOC_TRIM_THRESHOLD_=536870912

export OMP_NUM_THREADS=4

aprun -n8 -N8 -d $OMP_NUM_THREADS -cc cpu -ss ./a.out
```

Starting a MPMD job on a non-default projectid using 1 master, 16 slaves, each with 8 threads

```
#PBS -N hybrid
#PBS -l mppwidth=160 ! Note : 5 nodes * 32 cores = 160 cores
#PBS -l mppnppn=32
#PBS -l walltime=01:00:00
#PBS -j oe
#PBS -W group_list=My_Project

cd $PBS_O_WORKDIR

export MPICH_ENV_DISPLAY=1

export MALLOC_MMAP_MAX_=0
export MALLOC_TRIM_THRESHOLD_=536870912

export OMP_NUM_THREADS=8
id # Unix command ‚id‛, to check group id

aprun -n1 -d32 -N1 ./master.exe :
    -n 16 -N4 -d $OMP_NUM_THREADS  -cc cpu -ss ./slave.exe
```

CRAY THE SUPERCOMPUTER COMPANY

CSCS Swiss National Supercomputing Centre

HP2C

Starting an MPI job on two nodes using only every second integer core

```
#PBS -N hybrid
#PBS -l mppwidth=32
#PBS -l mppnppn=16
#PBS -l mppdepth=2
#PBS -l walltime=01:00:00
#PBS -j oe

cd $PBS_O_WORKDIR

export MPICH_ENV_DISPLAY=1

aprun -n32 -N16 -d 2 -cc cpu -ss ./a.out
```

Starting a hybrid job on two nodes using only every second integer core

```
#PBS -N hybrid
#PBS -l mppwidth=32
#PBS -l mppnppn=16
#PBS -l mppdepth=2
#PBS -l walltime=01:00:00
#PBS -j oe

cd $PBS_O_WORKDIR

export MPICH_ENV_DISPLAY=1

export OMP_NUM_THREADS=2

aprun -n32 -N16 -d $OMP_NUM_THREADS
    -cc 0,2:4,6:8,10:12,14:16,18:20,22:24,26:28,30 -ss ./a.out
```

Running a batch application with SLURM

- The number of required nodes can be specified in the job header
- The job is submitted by the qsub command
- At the end of the exection output and error files are returned to submission directory
- Environment variables are inherited
- The job starts in the directory from which the job has been submitted

Hybrid MPI + OpenMP

```
#!/bin/bash
#SBATCH --job-name=“hybrid”
#SBATCH --time=00:10:00
#SBATCH --nodes=8

export OMP_NUM_THREADS=6
aprun –n32 –d6 a.out
```

Starting an interactive session with SLURM

- An interactive job can be started by the SLURM **salloc** command

- Example: allocate 8 nodes

```
$ salloc -N 8
```

Further SLURM info available from CSCS web page: www.cscs.ch
User Entry Point / How to Run a Batch Job / Palu - Cray XE6

Documentation

- Cray docs site

  http://docs.cray.com

- Starting point for Cray XE info

http://docs.cray.com/cgi-bin/craydoc.cgi?mode=SiteMap;f=xe_sitemap

- Twitter ?!?

  http://twitter.com/craydocs

End