Center for Mathematical Modeling
University of Chile

HPC 101

HPC systems basics and concepts

By

Juan Carlos Maureira B.
<*jcm@dim.uchile.cl*>

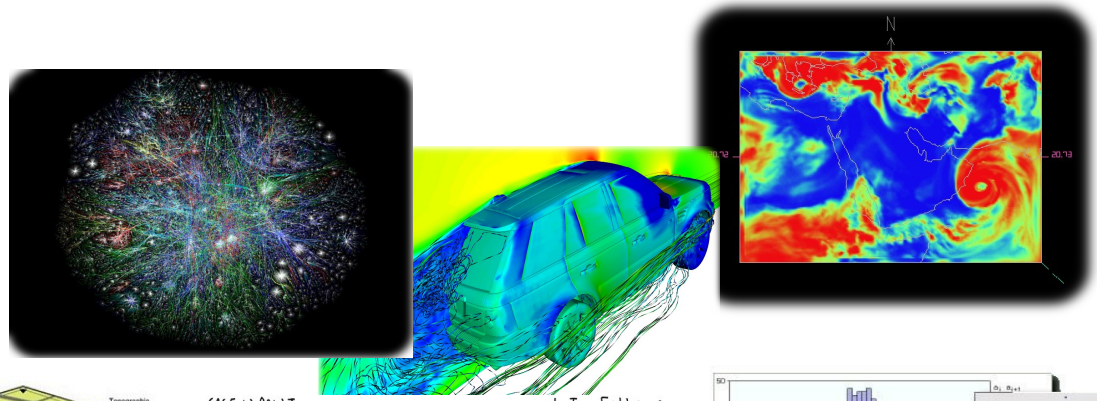
BioMedicina II – Calculo Masivo en Biomedicina
CMM - FCFM - University of Chile

Overview

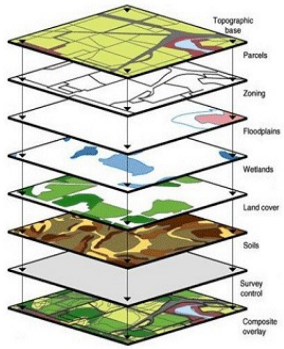
- Concepts & Definitions.
- Working with a HPC system.
- Best Practices.
- The take aways.

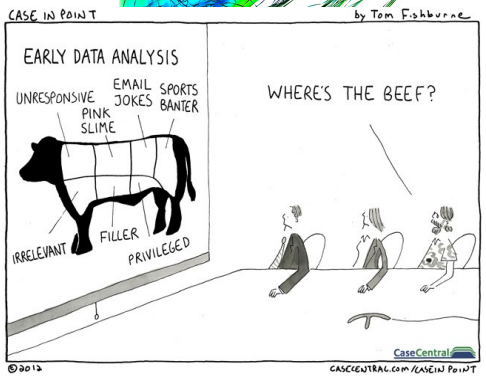
Scientific Computing

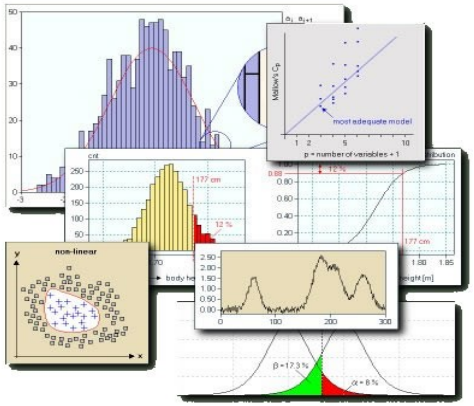
- Simulations



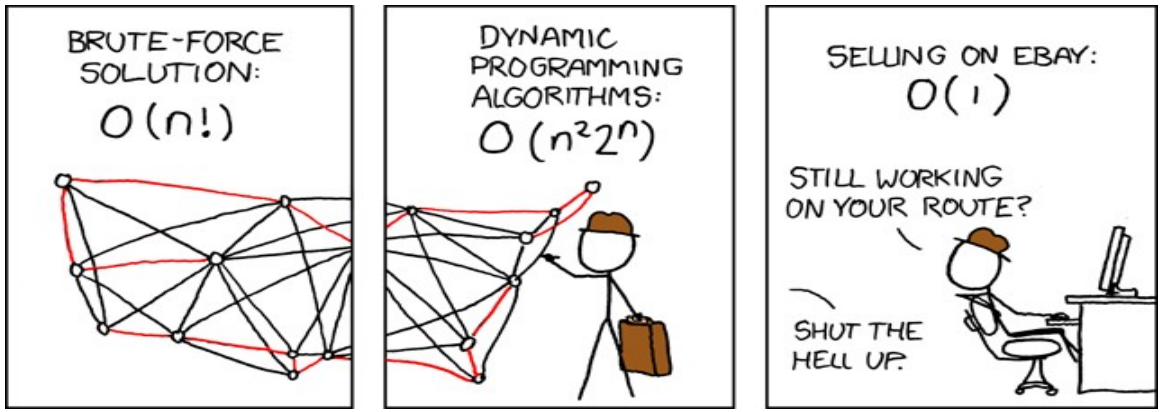
- Data Analysis







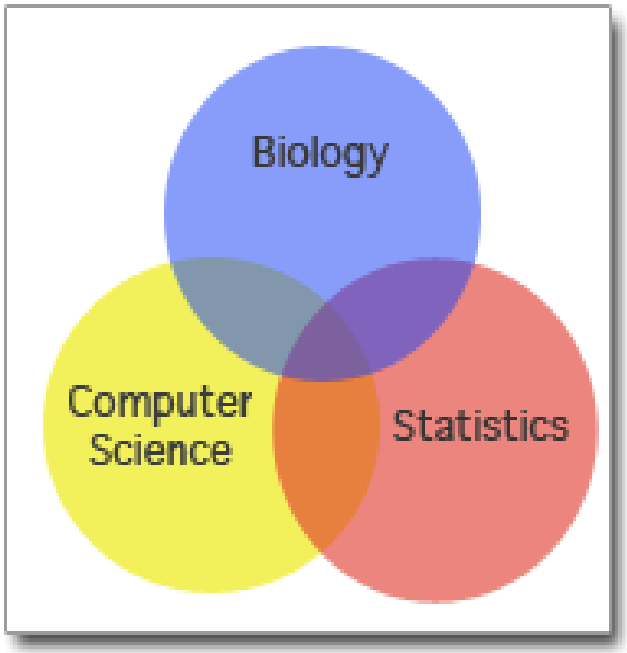
- Computational Optimization

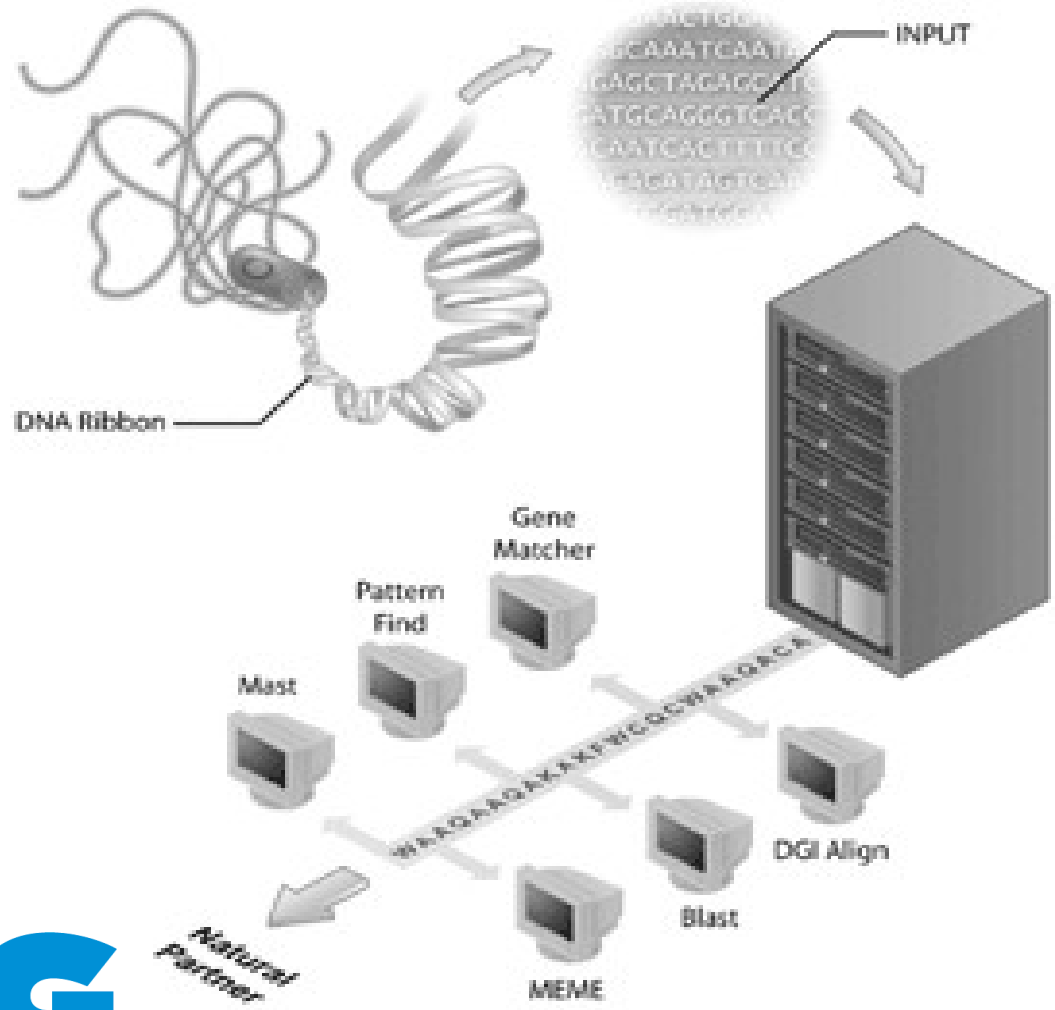


Scientific Computing

Scientific Computing on BioMed

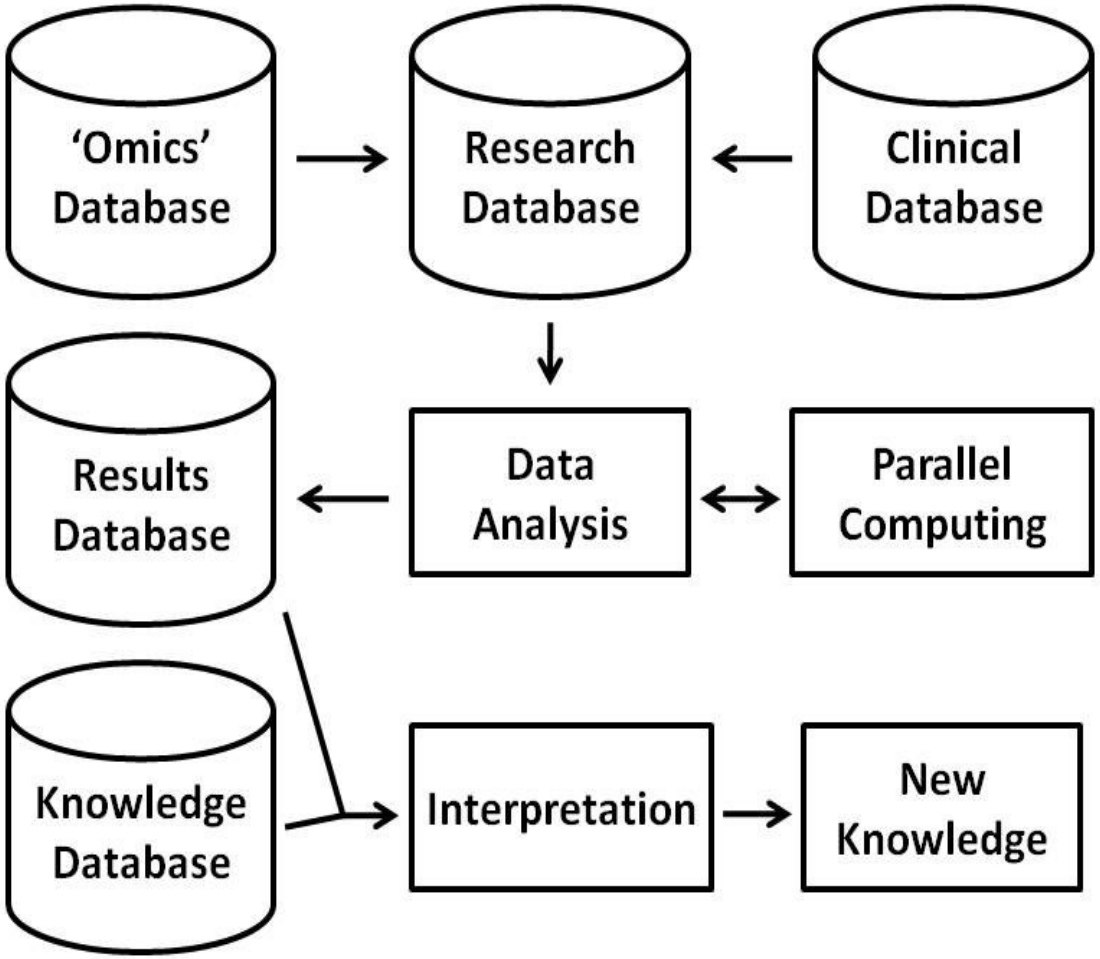
Biology & Informatics





BIG DATA

BioInfo Workflow



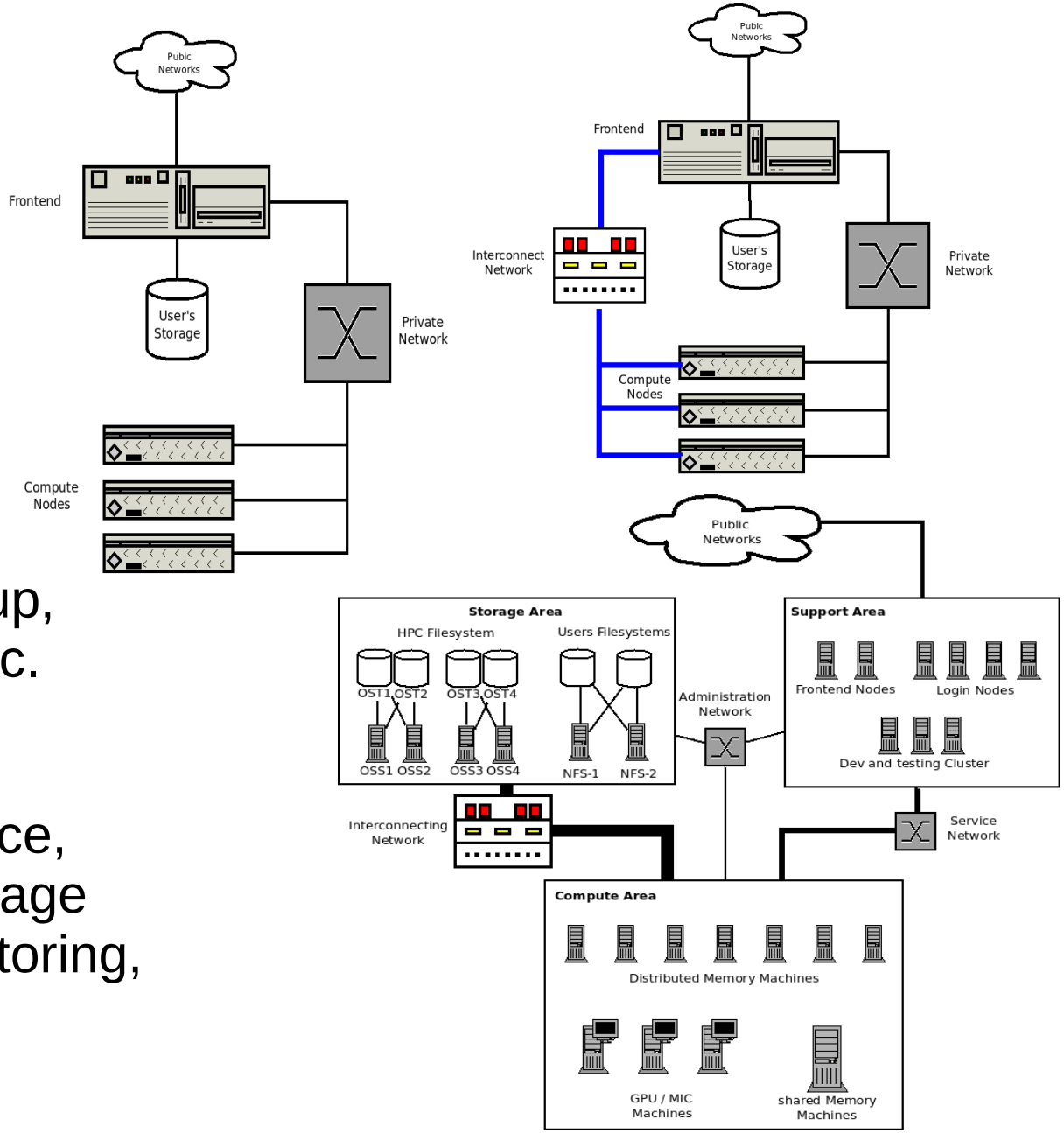
Source: BISR - Bioinformatics Shared Resources

High Performance Computing

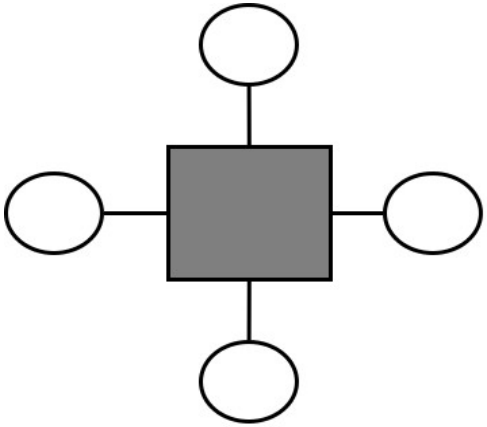
Concepts & Definitions

HPC system Architecture

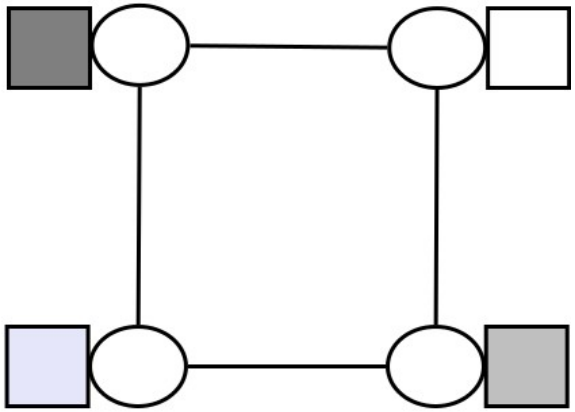
- Areas
  - Computing, Storage, Support, Networking
- Servers roles
  - Compute, frontend, login, storage, backup, devel, monitoring, etc.
- Networks
  - Private, public, service, interconnection, storage administration, monitoring, etc.



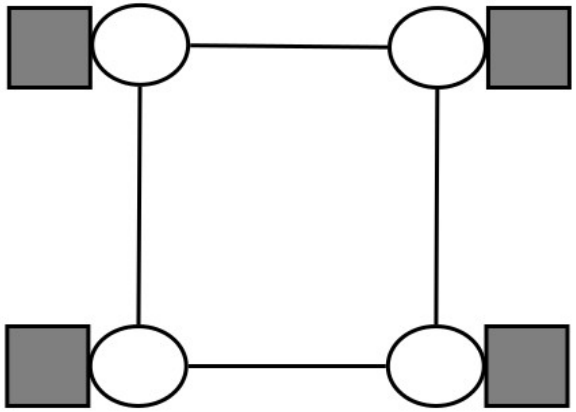
Distributed and Shared Memory Systems



Shared Memory



Distributed Memory

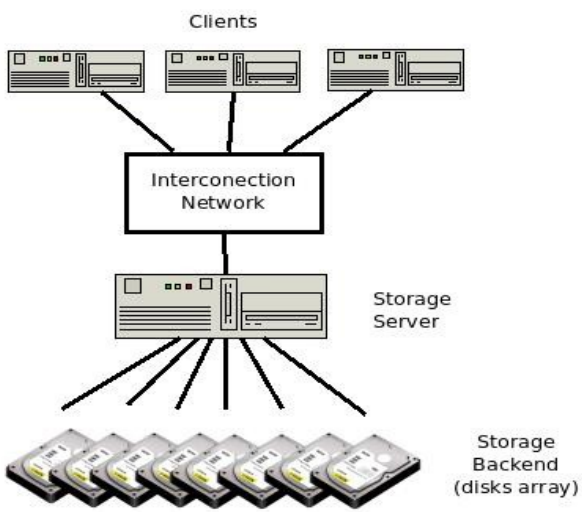


Distributed Shared Memory

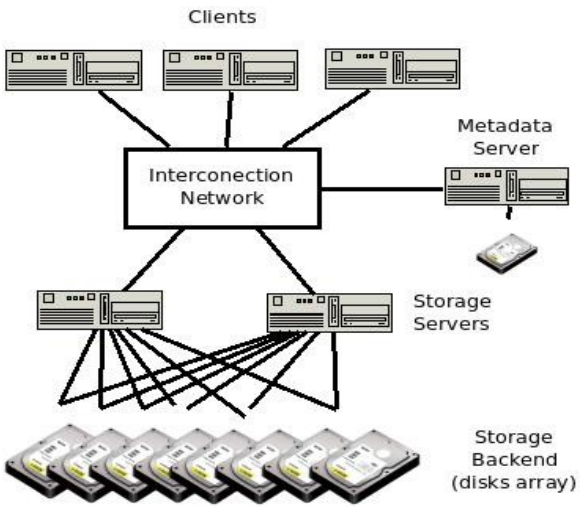
Interconnects

- **Ethernet**
  - latency ~ 0.05 ms
  - Throughput ~ 10 Gbps
- **Infiniband**
  - latency ~5 usec
  - Throughput ~ 40/56 Gbps
- **QPI / NUMA**
  - Latency ~ 100 nsec
  - Throughput ~ 100 - 200 Gbps

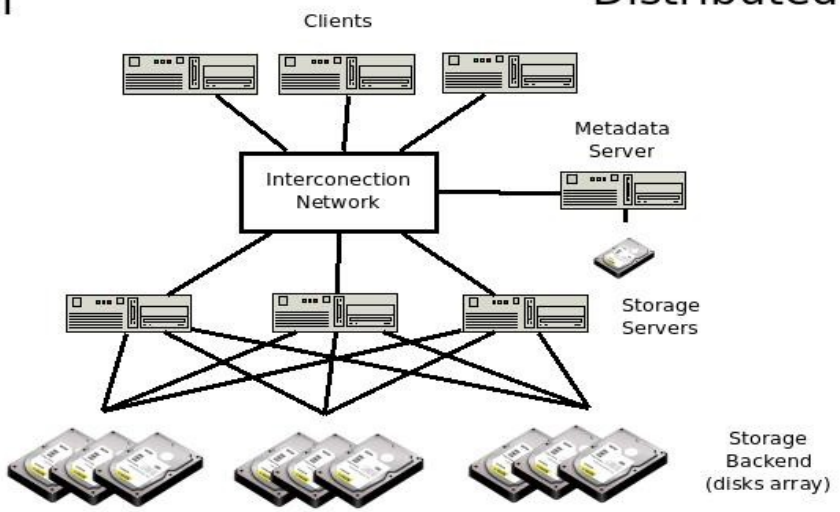
File-systems Types



Serial



Distributed



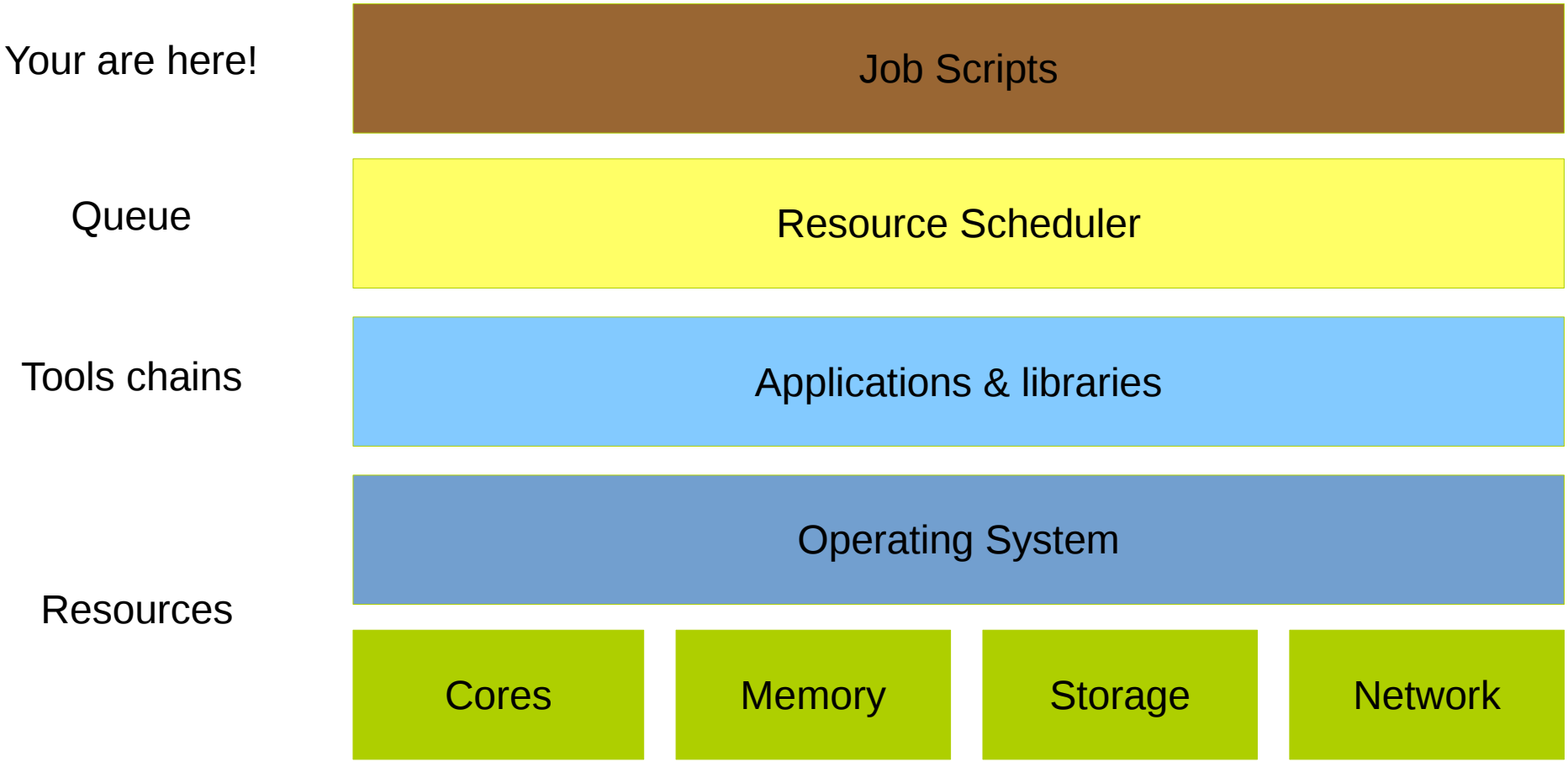
Parallel

- Serial
  - NFS, ZFS
- Distributed
  - pNFS
  - GFS
  - Gluster
- Parallel
  - Lustre
  - GPFS

Storage Layouts

- **Working ($Home)**
  - Safe and Slow storage.
  - Cheap
  - Bad for I/O
- **Scratch**
  - Unsafe and Fast storage
  - Expensive
  - Volatile and great for I/O
- **Archiving**
  - disaster-proof storage
  - Incredible slow (random) access
  - Backup Policies

Software Layout



Tool Chains

- Set of self-standing libraries and applications to perform a class of jobs. (e.g. astro, bioinfo, optimization, etc).
  - Compiled from scratch.
  - Operating System independent.
  - Specific Libraries levels and versions.
- System wide (one for all).
  - Compiled and Installed by Sysadmins.
- User Space (each one has its own).
  - Compiled and installed by the user in their homes.

Resource Schedulers

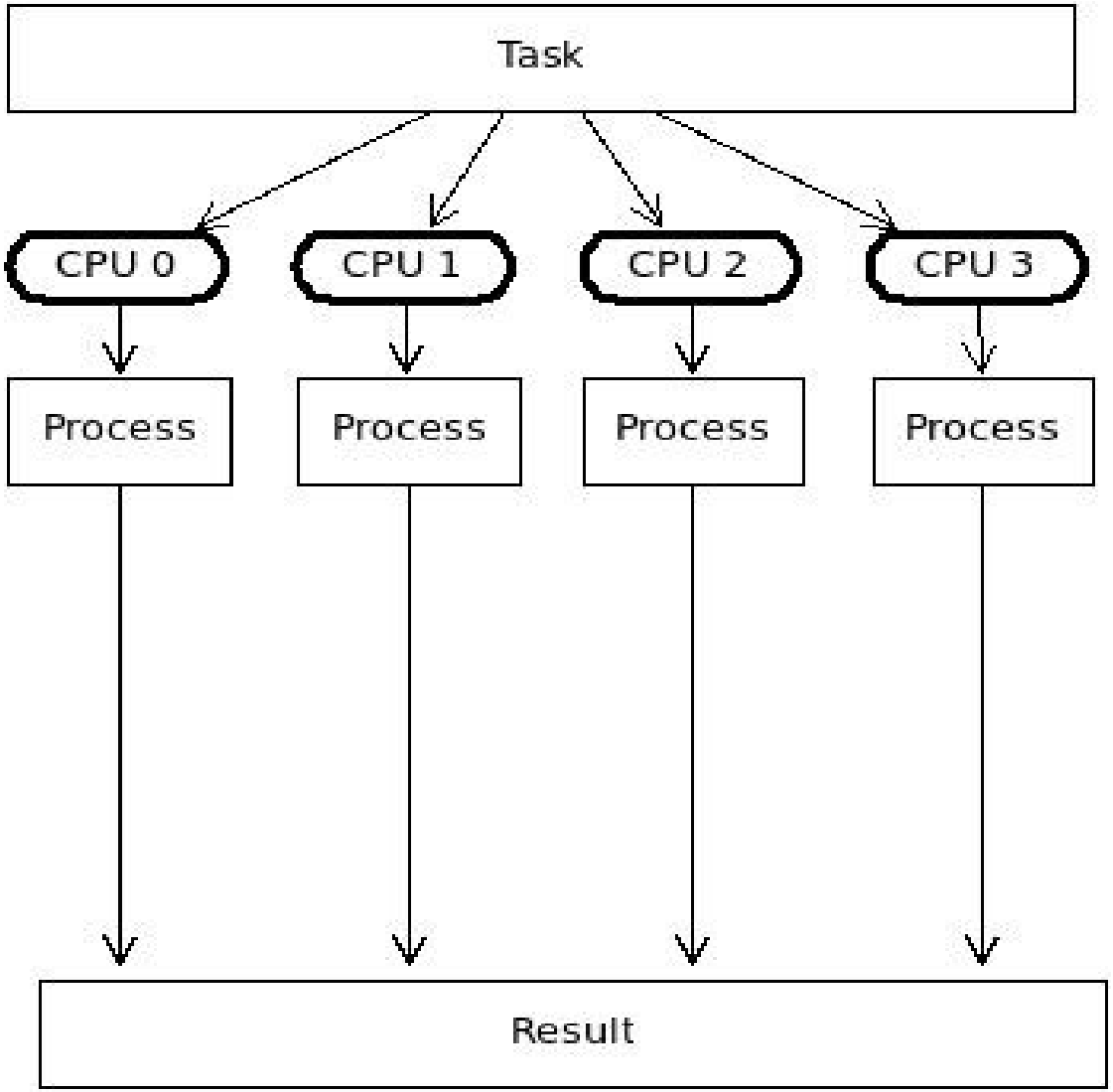
- **Scheduler**: allocate resources to perform a job.
- **Job**: set of instructions and resources to perform a task.
- **Task**: involves preparing the environment and input data needed to run an application.

Resource specifications

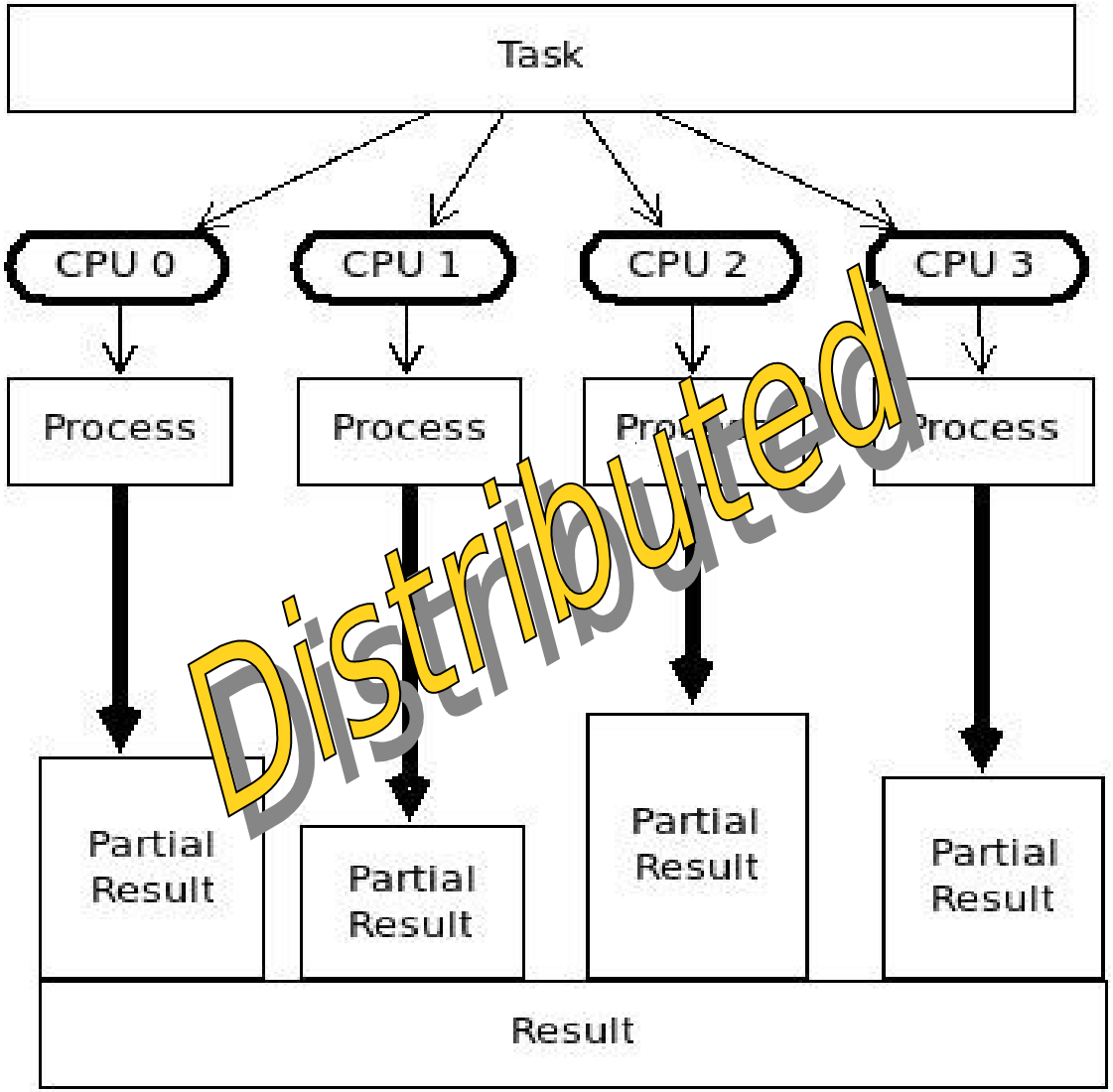
+

Instructions to perform a task

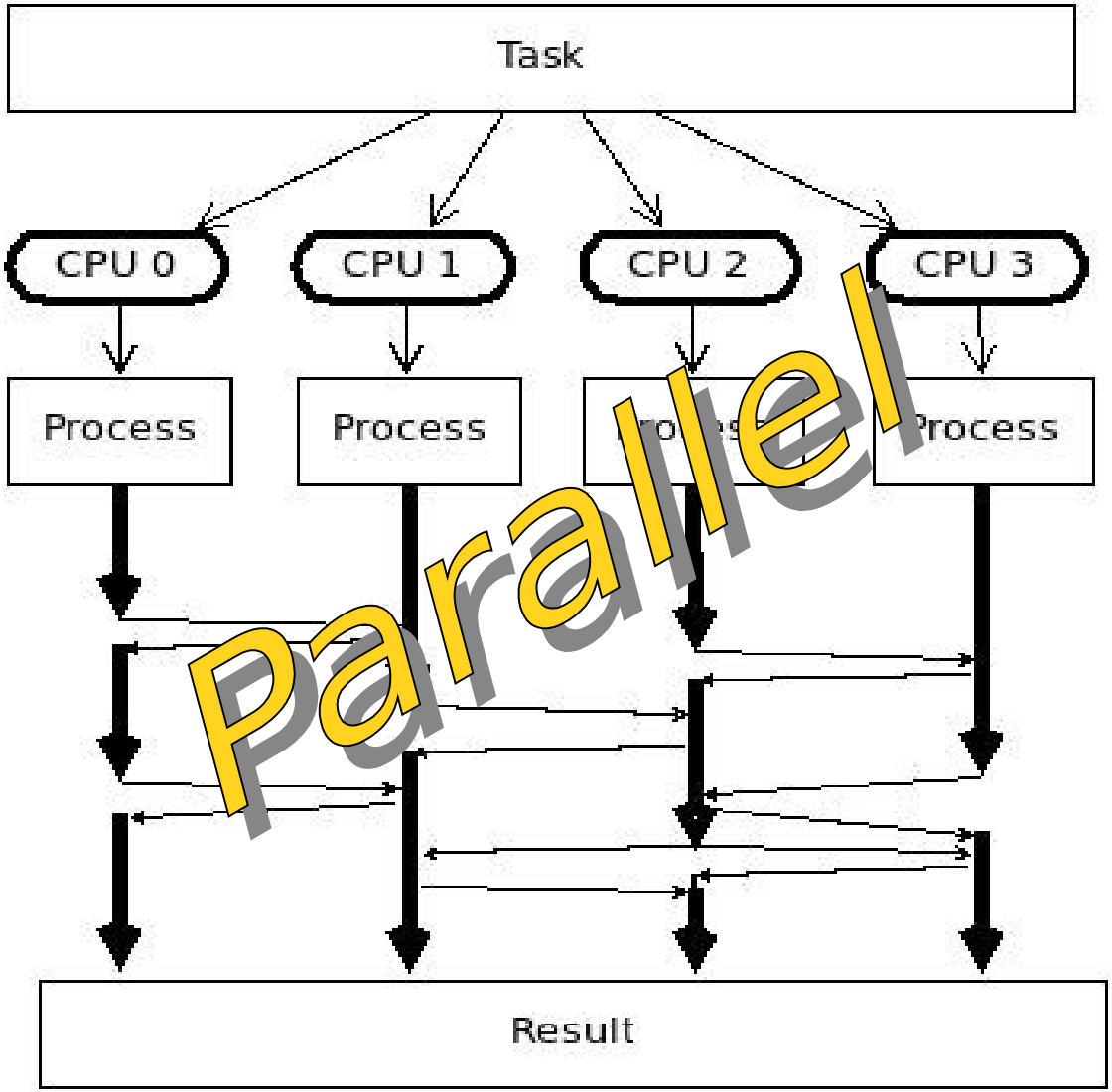
Jobs: Parallel v/s Distributed



Jobs: Parallel v/s Distributed

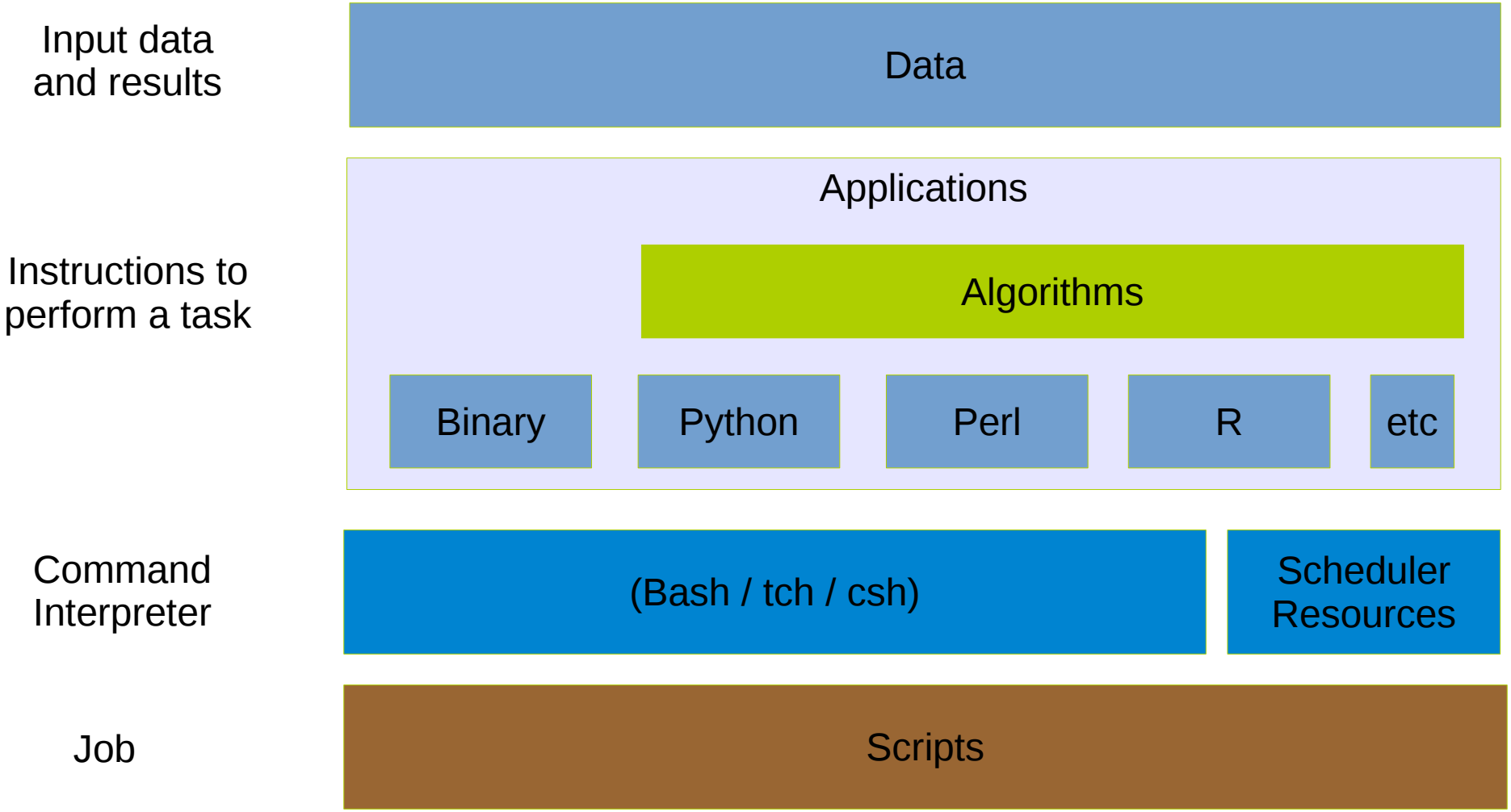


Jobs: Parallel v/s Distributed



Working with a HPC System

Job Scripting



Job Scheduler Directives

```
#!/bin/bash
# Resource specification
#$ -l h_rt=1:00:00
#$ -cwd
#$ -j y
#$ -V
#$ -notify
# User Notification
#$ -m abes
#$ -M myemail@domain.com
# Job name
#$ -N jobname
# Command interpreter
#$ -S /bin/bash
# Parallel environment: openmpi,openmp,etc
#$ -pe openmpi 128
# Job Array
#$ -te 1:1000
# Queue to use
#$ -q all.q
```

- **Grid Engine**
- PBS
- Condor

Job Scheduler Directives

```
#!/bin/bash
# number of nodes and processes per node
#PBS -l select=4:mpiprocs=8
# resources
#PBS -l mem=213mb
#PBS -l walltime=2:00:00
#PBS -l cput=1:00:00
# name of job
#PBS -N jobname
# User notificacion
#PBS -m bea
#PBS -M myemail@domain.com
# Use submission environment
#PBS -V
# Queue to use
#PBS -q default
```

- Grid Engine
- **PBS**
- Condor

Job Scheduler Directives

```
## executable file
Executable = my_program.exe
## Specify Condor execution environment.
Universe = vanilla
## execution machines
Requirements = ((OpSys == "LINUX")
## Define input files and arguments
Input = stdin.txt.$(Process)
Arguments = in.$(Process)  out.$(Process)
## Define output/error/log files
Output = logs/stdout.$(Process).txt
Error = logs/stderr.$(Process).txt
Log = logs/log.$(Process).txt
## files need to be transferred and when.
Transfer_input_files = files/in1.$(Process)
Transfer_output_files = out.$(Process)
Transfer_executable = true
Should_transfer_files = YES
When_to_transfer_output = ON_EXIT
## Add 100 copies of the job to the queue
Queue 100
```

- Grid Engine
- PBS
- **Condor**

Environment Modules

- Configure the environment to run a particular application (or a set of applications)
  - Environmental variables:
    - PATH
    - LD_LIBRARY_PATH
    - LD_RUN_PATH
  - Library versions and locations
    - BOOST_HOME, ATLAS_HOME, etc
  - Compilation & execution flags
    - CFLAGS, LDFLAGS, CXXFLAGS, etc.

Environment Modules

- Example: module available

```
[astrouser@levque ~]$ module available

------------------------- /usr/share/Modules/modulefiles -------------------------
dot         module-cvs  module-info modules     null        use.own

------------------------------- /etc/modulefiles -------------------------------
R/3.1.0                intel/12.1.5         openmpi_gcc44/1.4.3
adf/2012.01            intelmpi/4.0.2       openmpi_intel/1.4.3
astro/1.1              intelmpi/4.0.3       openmpi_intel/1.6.5
cplex/12.1.0           lsst/9.2             openmpi_pgi/1.4.3
cplex/12.2.0           namd/2.7             opl/6.3
cplex/9.1.0            namd/2.8             pgi/10.9
espresso/5.1           namd/2.9             python/2.6.6
gaussian/09B1          namd_intel/2.8       siesta/3.2
gnuplot/4.6.5          octave/3.8.1         stata/11.0
gromacs/4.6.5          openmpi/1.4.2
gromacs_single/4.5.3 openmpi/1.4.3

[astrouser@levque ~]$
```

Environment Modules

- module show {module name/version}

```
[astrouser@levque ~]$ module show lsst

-------------------------------------------------------------------
/etc/modulefiles/lsst/9.2:

module-whatis    Sets up the LSST Astro toolchain (Self-standing edition)
                 in you enviornment
prepend-path     PATH /home/apps/lsst/bin
prepend-path     PATH /home/apps/lsst/sbin
prepend-path     LD_LIBRARY_PATH /home/apps/lsst/lib
prepend-path     LD_LIBRARY_PATH /home/apps/lsst/lib64
prepend-path     MANPATH /home/apps/lsst/share/man
-------------------------------------------------------------------

[astrouser@levque ~]$
```

Environment Modules

- module load {module name/version}
- module list

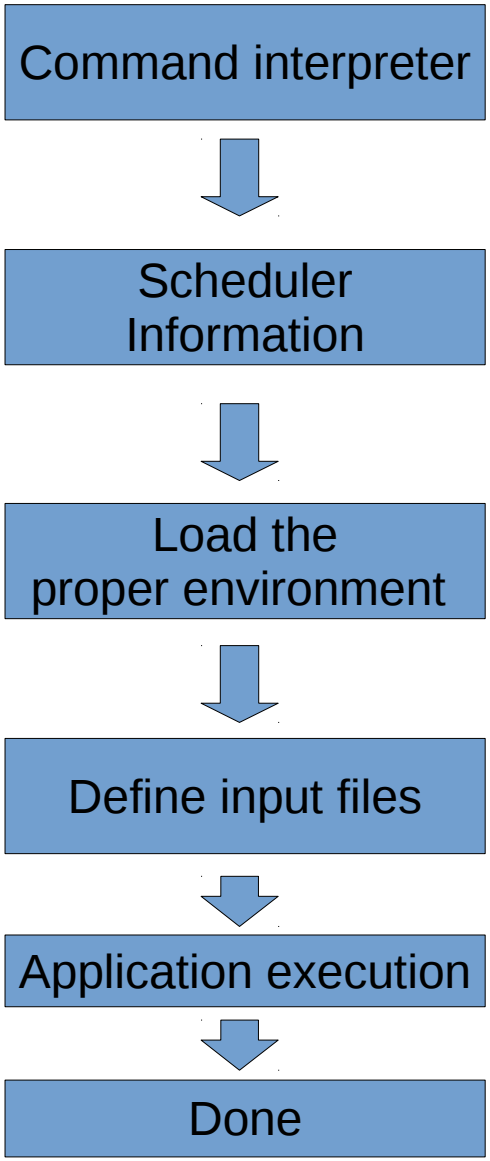
```
[student01@syntagma ~]$ module load lsst

[student01@syntagma ~]$ module list
Currently Loaded Modulefiles:
  1) lsst/9.2

[astrouser@development ~]$ echo $LD_LIBRARY_PATH
/home/apps/lsst/lib64:/home/apps/lsst/lib:/opt/intel/Compiler/11.1/072/lib/intel64:
/opt/intel/Compiler/11.1/072/ipp/em64t/sharedlib:/opt/intel/Compiler/11.1/072/mkl/lib
/em64t:/opt/intel/Compiler/11.1/072/lib/intel64:/opt/intel/Compiler/11.1/072/ipp/em64t
/sharedlib:/opt/intel/Compiler/11.1/072/mkl/lib/em64t

[astrouser@development ~]$ echo $PATH
/home/apps/lsst/sbin:/home/apps/lsst/bin:/opt/gridengine/bin/lx24-amd64:/usr/lib64
/qt-3.3/bin:/usr/kerberos/bin:/opt/intel/Compiler/11.1/072/bin/intel64:/opt/intel
/Compiler/11.1/072/bin/intel64:/usr/local/bin:/bin:/usr/bin:/home/uchile/cmm
/astrolab/astrouser/bin
```

Creating (Sun Grid Engine) Jobs




```
#!/bin/bash
#$ -cwd
#$ -j y
#$ -V
#$ -notify
#$ -m abes
#$ -M myemail@domain.com
#$ -N my-first-job
#$ -S /bin/bash
#$ -q all.q

. /etc/profile.d/modules.sh
module load lsst

echo “Running at `hostname -s`”
echo “Starting at `date '+%c'`”

INPUT_FITS=$1
WEIGHT_FITS=$2

sex $INPUT_FITS -CATALOG_NAME catalogue.cat \
    -WEIGHT_IMAGE $WEIGHT_FITS


echo “done”
```

Interacting with the SGE

- qsub `job_script.sge`
  - Submit *`job_script`* to the queue
- qstat
  - Show only the status of **your jobs** in the queue
- qstat -f
  - Show the status of **jobs** by queue
- qstat -u "`user`"
  - Show the jobs owned by `user`
- qstat -j `Job-ID`
  - Show the status for the job `Job-ID`
- qhost
  - Show the status of each node
- qhost -j
  - Show the status of each node and the jobs running on each one of them
- qdel `Job-ID`
  - Cancel (running) and delete a job from the queue

Submitting & Monitoring Jobs

```
[astrouser@levque ~]$ qsub my_first_job.sge
Your job XXXX ("my-first-job") has been submitted


[astrouser@levque ~]$ qstat
job-ID  prior   name       user   state submit/start at   queue             slots ja-task-ID
-----------------------------------------------------------------------------------------
XXXX 0.00000 my-first-s astrouser  qw   08/09/2013 23:00:06                  1

[astrouser@levque ~]$ qstat
job-ID  prior   name       user   state submit/start at   queue             slots ja-task-ID
-----------------------------------------------------------------------------------------
XXXX 0.00000 my-first-s astrouser  r   08/09/2013 23:00:30 all.q@levque001  1
```

- `watch` is your friend
  - `watch -n 1 "qstat"` : show qstat at 1 second interval
- Ganglia is your best friend

Monitoring Jobs

- **Ganglia** is an open source monitoring system developed in the NPACI (UCLA) and used initially by the UCSD to monitor their Rocks clusters.

http://syntagma.cmm.uchile.cl/ganglia

http://levque.dim.uchile.cl/ganglia

- Queue is monitored at “host overview” in the frontend.
- Compute nodes “host overview” gives you the state of your processes (require an extra plug-in)
- Useful metrics such as memory and network consumption are shown in an aggregated way as well as in a host basis way.

Best Practices

Rule 1 : Characterize your job

Because you are not alone in the world,
when submitting your job, please mind:

- The **memory** consumption.
  - Per core
  - Per process
- How **intensive** is on **Input/Output** operations.
  - Is my process able to run in a NFS storage layout?
  - Is my process able to run in a parallel filesystem
  - Do my process need a locking scheme to work?
- Scalability in terms of **cores**.
  - No always the highest, the better

Rule 2 : Monitor your Job

Because your jobs are running together with other people's job, please mind:

- Check continuously the status of the compute nodes where your jobs are running
- Check the load of the filesystem if your job is I/O intensive
  - Ganglia is your friend
- Be sure about how much storage your job is consuming.
  - You don't want to be remembered as the guy who crash everyone else jobs due to “out of space” in the filesystem.

Rule 3 : Do not leave data unattended

Because the storage is expensive and other users have the same right to use it, please mind:

- Remove temporal data and job output files.
  - .o and .e files
  - By applying the Rule 1, you should know if your job produces temporal files that can be deleted at the end of the job.
- When finishing with your results, keep them in other storage than your HPC account.
- Do not leave data in the scratch filesystem – It is volatile, which means it can be wiped out without further notice.
- Get used to backup your work (scripts, jobs, results).

Rule 4: Share your knowledge

Because not all the users have your skills to work with a HPC System, please mind:

- Write documentation by commenting your jobs scripts between lines.
- Write `readme` files specifying how many "*shaolin chambers*" did you have to go through to get your job done.
- Publish your scripts (if you are allowed to)

The Take Aways

- Definitions needed to understand a HPC system.
  - Frontend, login, compute, storage nodes
  - Interconnects, public, service, admin. Networks
  - Fast, slow, archiving, scratch storages.
- Overview about architecture and components of a HPC system.
  - Memory, filesystems, storage layouts.
- Software, Applications, tools-chains
  - Resource manager.
  - Parallel, distributed jobs.
  - Job scripting.
- Best practices when working with a HPC system.