



DATA SCIENCE &  
SCIENTIFIC COMPUTING



Istituto Officina  
dei Materiali



## L02: HPC software stack

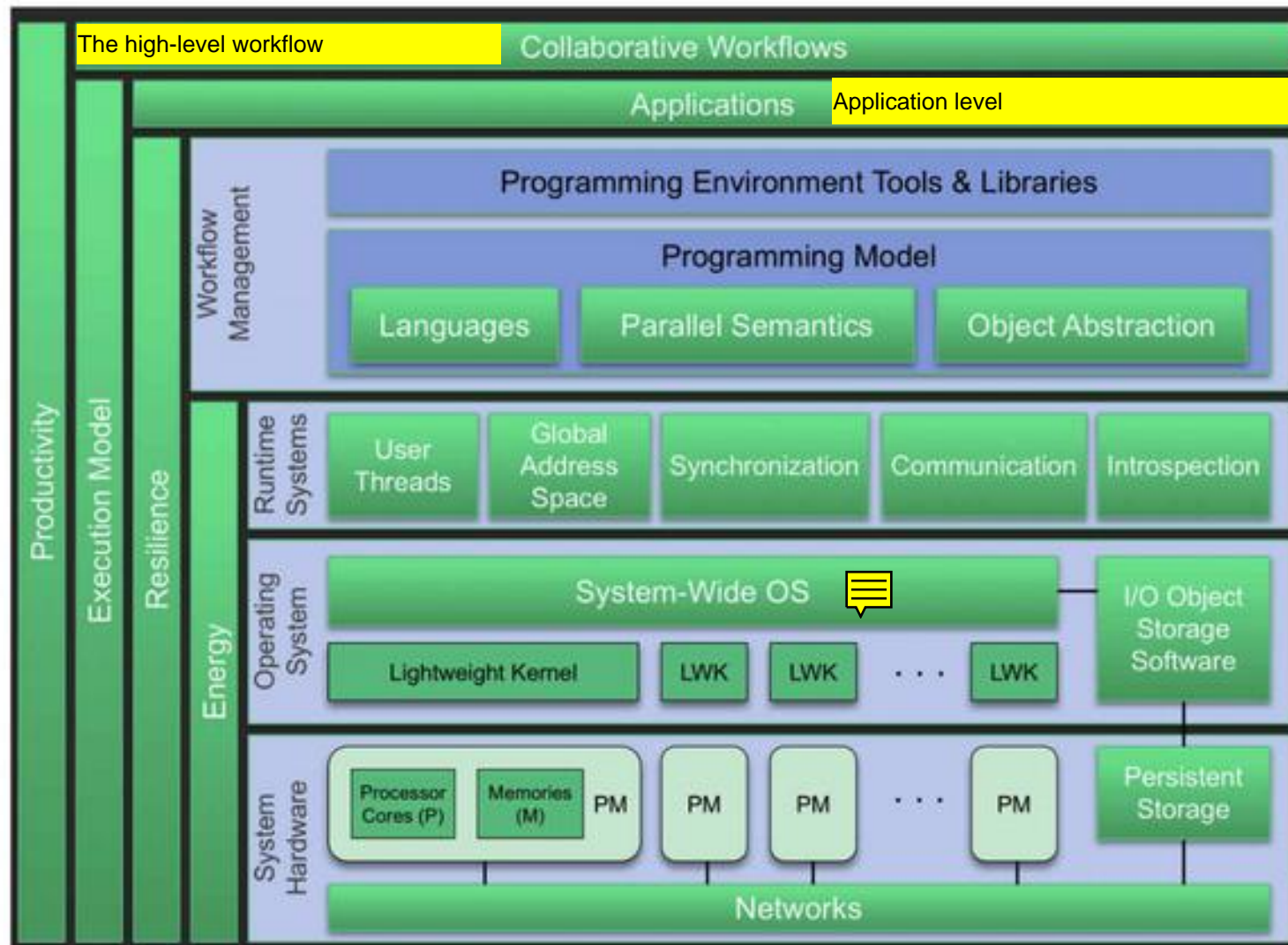
- Stefano Cozzini
- CNR-IOM and eXact lab srl

# Agenda

- A first look at the Software stack for HPC (part 1)
  - Middleware
  - Compilers
  - Libraries
  - Scientific software
  - How to compile scientific software
- HPC Concepts (part 2)
  - Parallel programming paradigms
  - Evolution of paradigms
  - Ahmdal law / Gustafson law
  - Strong/weak scalability

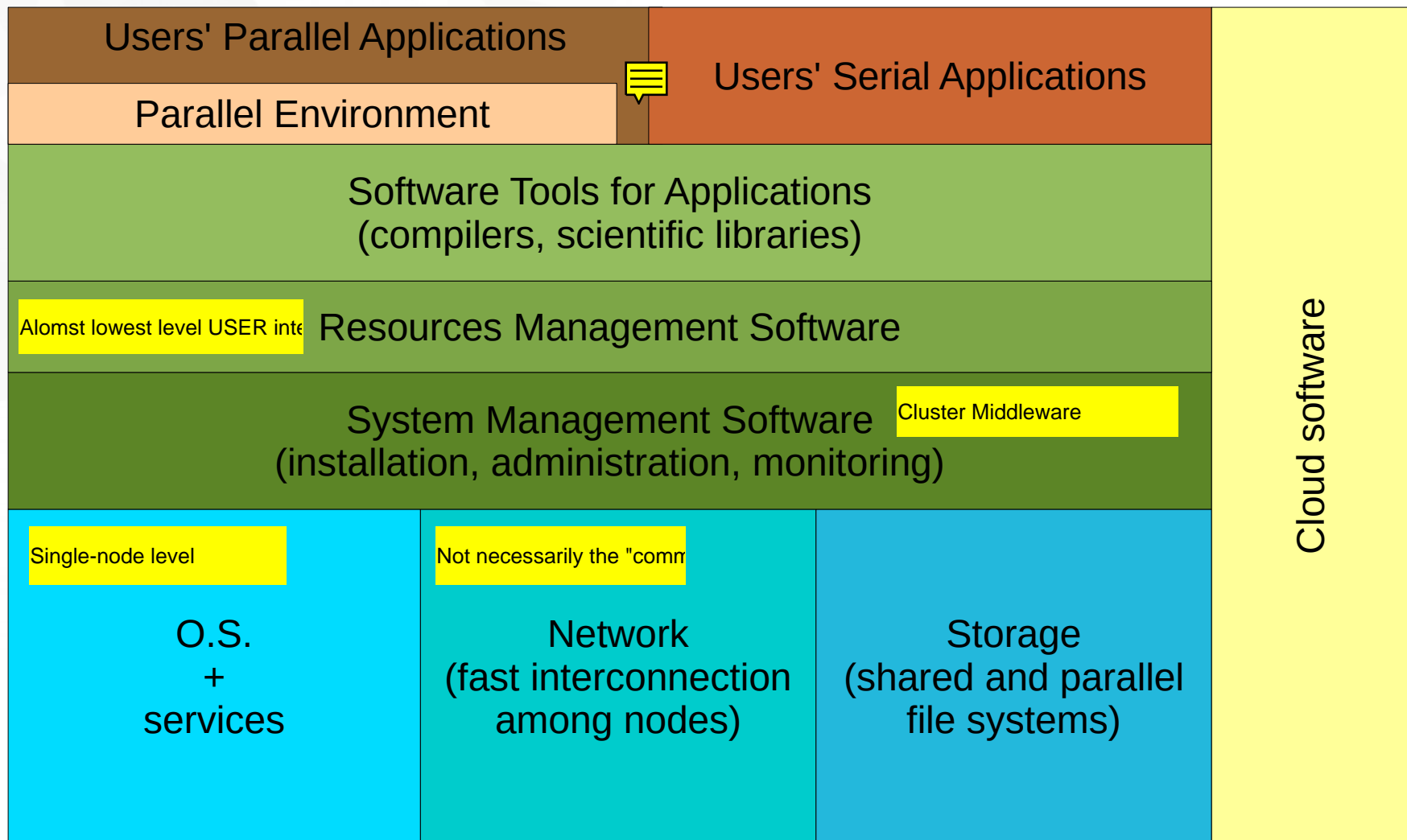
# System stack of a general HPC platform

Ref. 01; Chap.: 01.

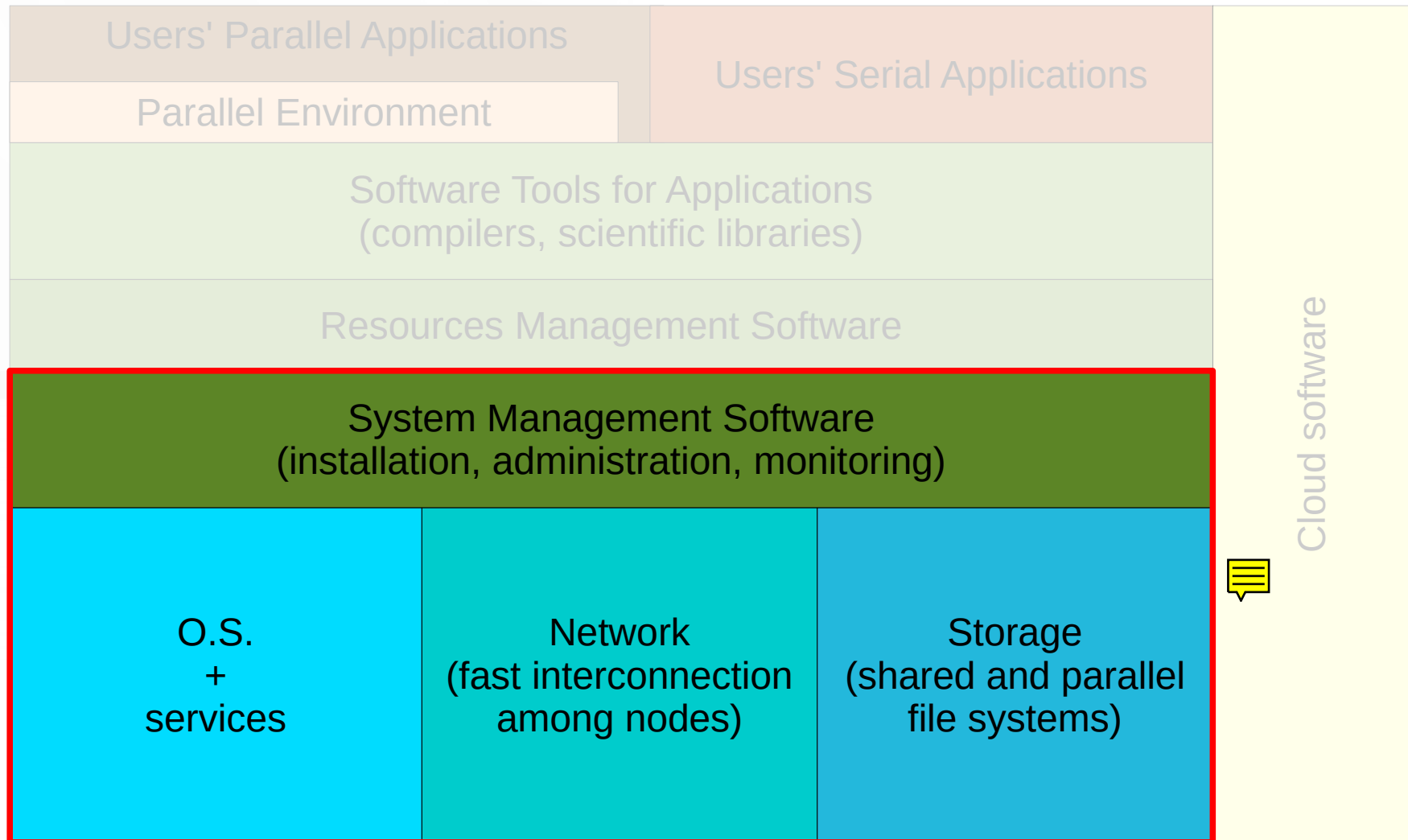


**Figure 1.9** The system stack of a general supercomputer consists of a system hardware layer and several software layers. The first software layer is the operating system, encompassing both resource management and middleware to access input/output (I/O) channels. Higher software layers include runtime systems and workflow management.

# HPC software stack: Overview



# HPC cluster middleware: Overview



# Cluster middleware: Middleware Design Goals

- Complete Transparency (Manageability):
  - Lets us see a single cluster system..
    - Single entry point, ftp, ssh, software loading...
- Scalable Performance:
  - Easy growth of cluster
    - no change of API & automatic load distribution.
- Enhanced Availability:
  - Automatic Recovery from failures
    - Employ checkpointing & fault tolerant technologies

Making the experience of using a HPC cluster

Not to re-do everything.

Robustness to errors and failure. High availability.

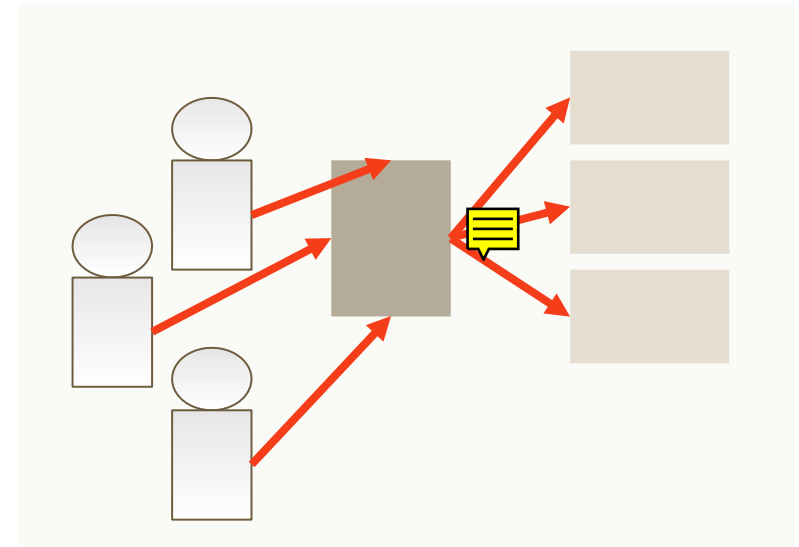
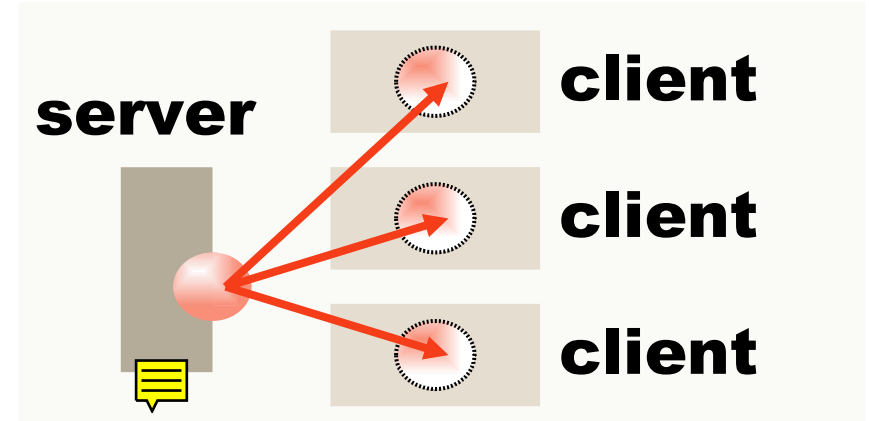
# Cluster middleware

Administration software:

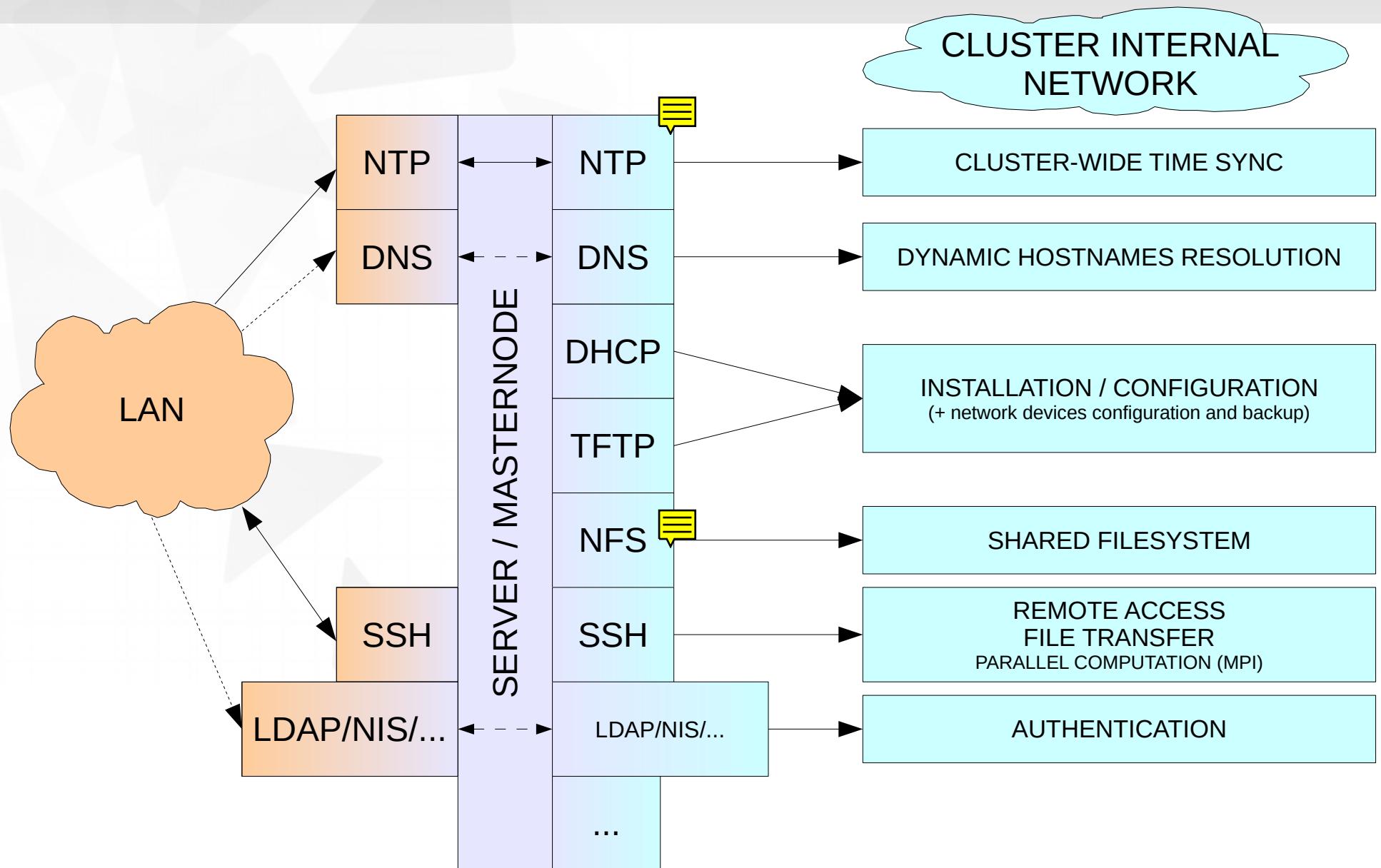
- user accounts
- NTP/NFS/ etc...

Resource management and scheduling software (LRMS)

- Process distribution
- Load balance
- Job scheduling of multiple tasks

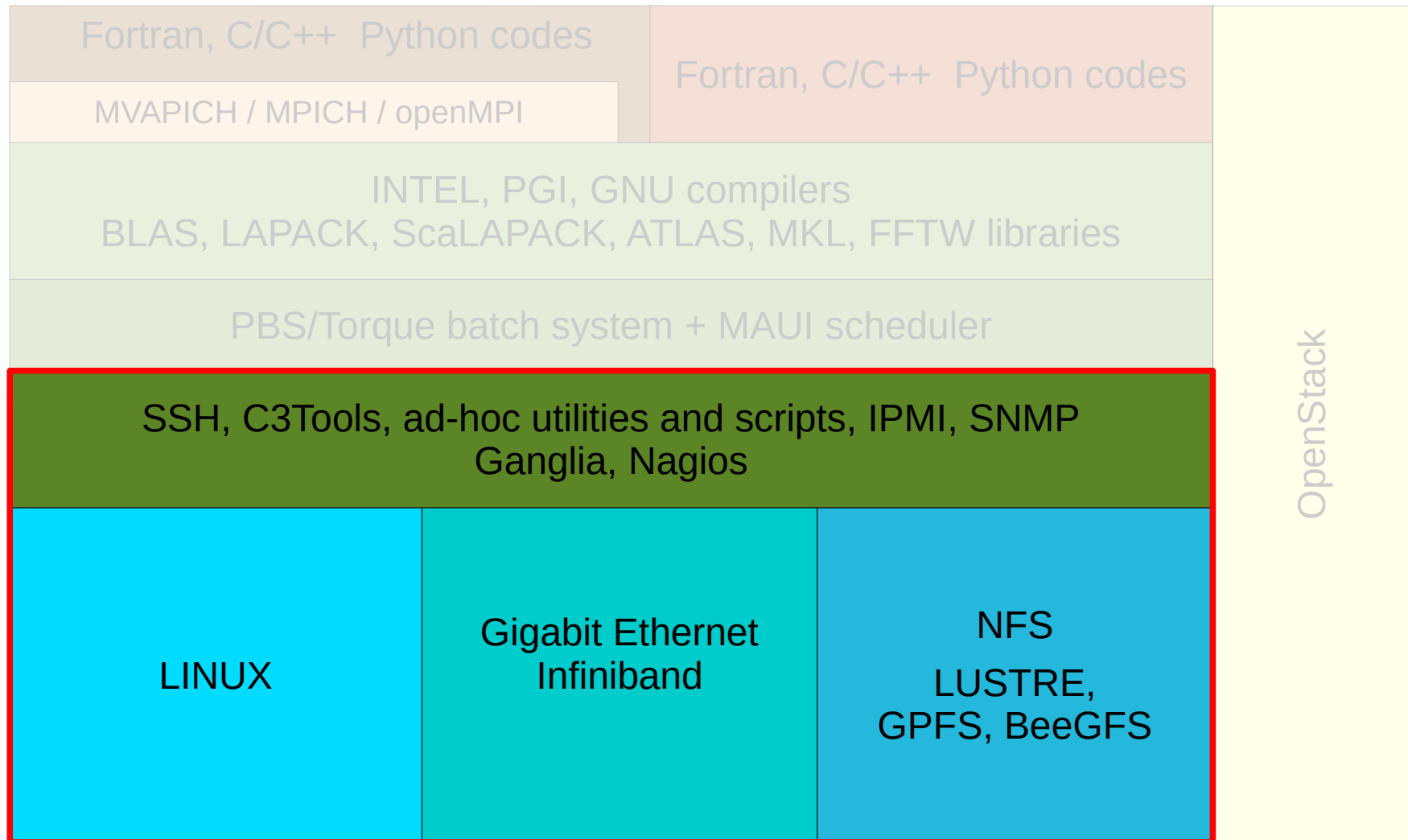


# CLUSTER SERVICES





# Cluster middleware tools used by us



# Resource Management Problem

We have a pool of users and a pool of resources, then what?

- some software that controls available resources
- some other software that decides which application to execute based on available resources
- some other software devoted to actually execute applications

# What are we speaking about ?



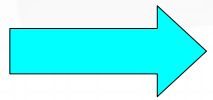
#0297170

REPLACE THE CAKE WITH HPC  
RESOURCE

# Resource management

The resource manager allows

- better **resource control**
- better **access control**



**better resource utilization**

# Some definitions

**Batch Scheduler:** software responsible for scheduling the users' jobs on the cluster.

*scheduling is the method by which work specified by some means is assigned to resources that complete the work*

**Resources Manager:** software that enable the jobs to connect the nodes and run.

**Node** (aka Computing Node): computer used for its computational power.


**Frontend/Master node:** it's through this node that the users will submit/launch/manage jobs.

# Management of job and resources

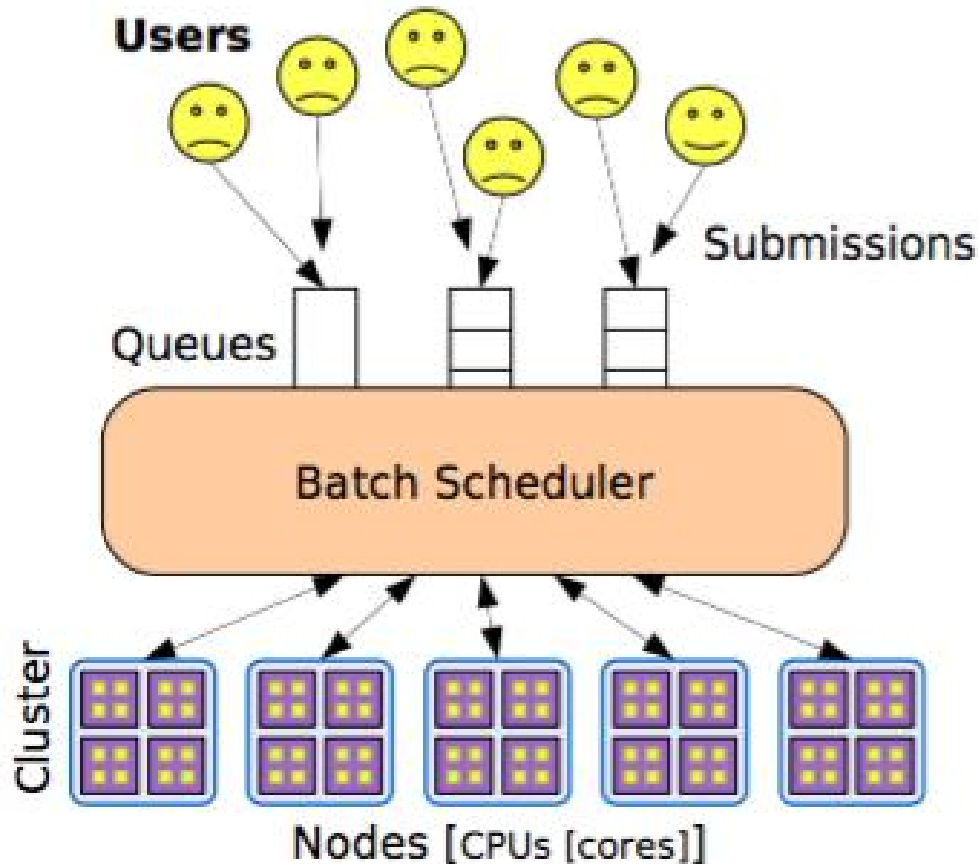
- **Management: Batch Scheduler and Resource Manager**

- Submission
- Scheduling
- Resources Allocation
- Job Launch
- Monitoring, logging...

## **2 layers**

- Resource Management Layer: launching, cleaning, monitoring...
- Job Management Layer: batch/interactive job, Scheduling, Suspend/Resume, Preemption, Dependencies, Resubmission, Advance Reservation... 

# Batch scheduler: a global picture (1)

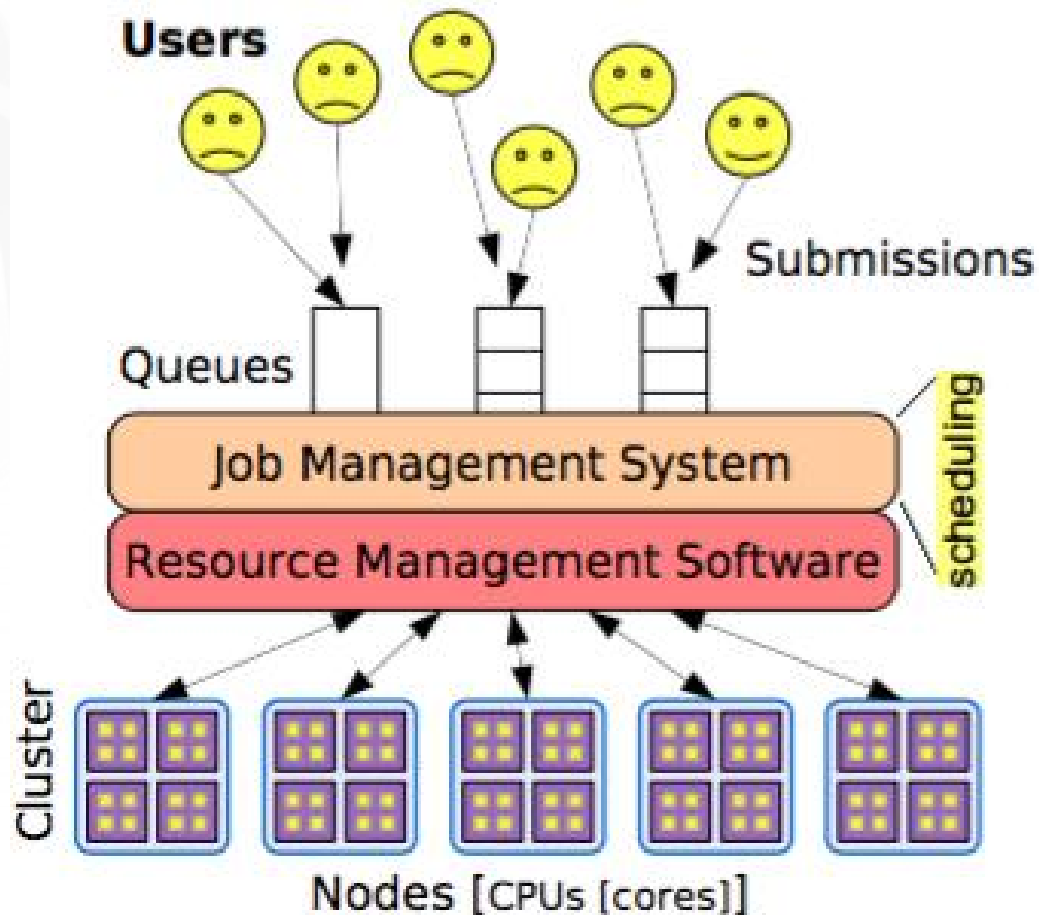


Allocate resources for each applications with respect of their requirements and users' rights.

- Satisfy users  
response time, reliability
- Satisfy admins  
high resource utilization  
efficiency  
energy management

...

# Batch scheduler: a global picture (2)



## Resource Management Layer

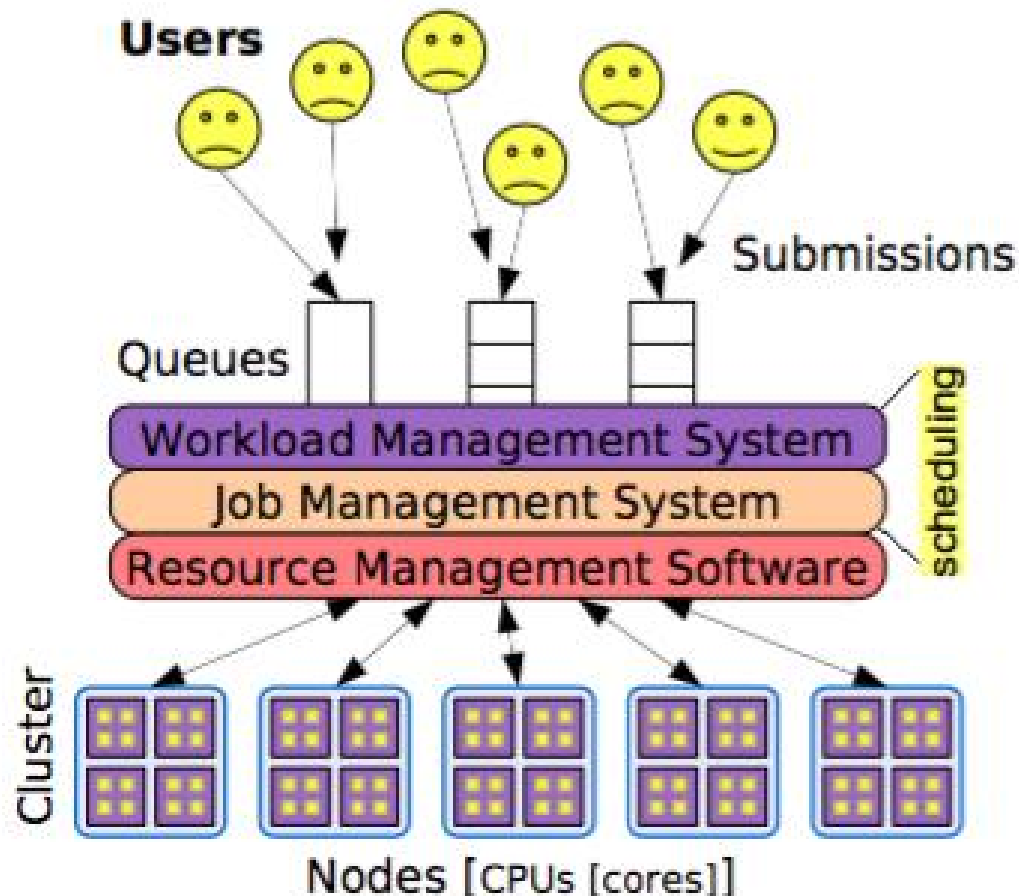
→ launching, cleaning, monitoring

## Job Management Layer

→ batch/interactive job  
→ backfilling  
→ scheduling  
→ suspend/Resume  
→ preemption  
→ dependencies  
→ resubmission  
→ advance reservation



# Batch scheduler: a global picture (3)



Workload/Job Management  
→ more complete job scheduling policies  
→ Fairsharing, Quality of Service (QoS), SLA (Service Level Agreement), Energy Saving  
→ Dedicated software: MAUI

# Mainly adopted schedulers

- GridEngine (SGE)
  - Sun of Grid Engine (SoGE)
  - Open Grid Scheduler (OGS)
  - Univa Grid Engine (UGE) – Commercial
- LoadLeveler – IBM
- LSF
  - expensive, but free in OpenLava edition
- PBSPRO
  - commercial or community edition
- SLURM
  - free, new, commercial support
- Torque resource manager +
  - MOAB scheduler, commercial
  - Maui scheduler, open source

# PBSPRO/ Torque queue system

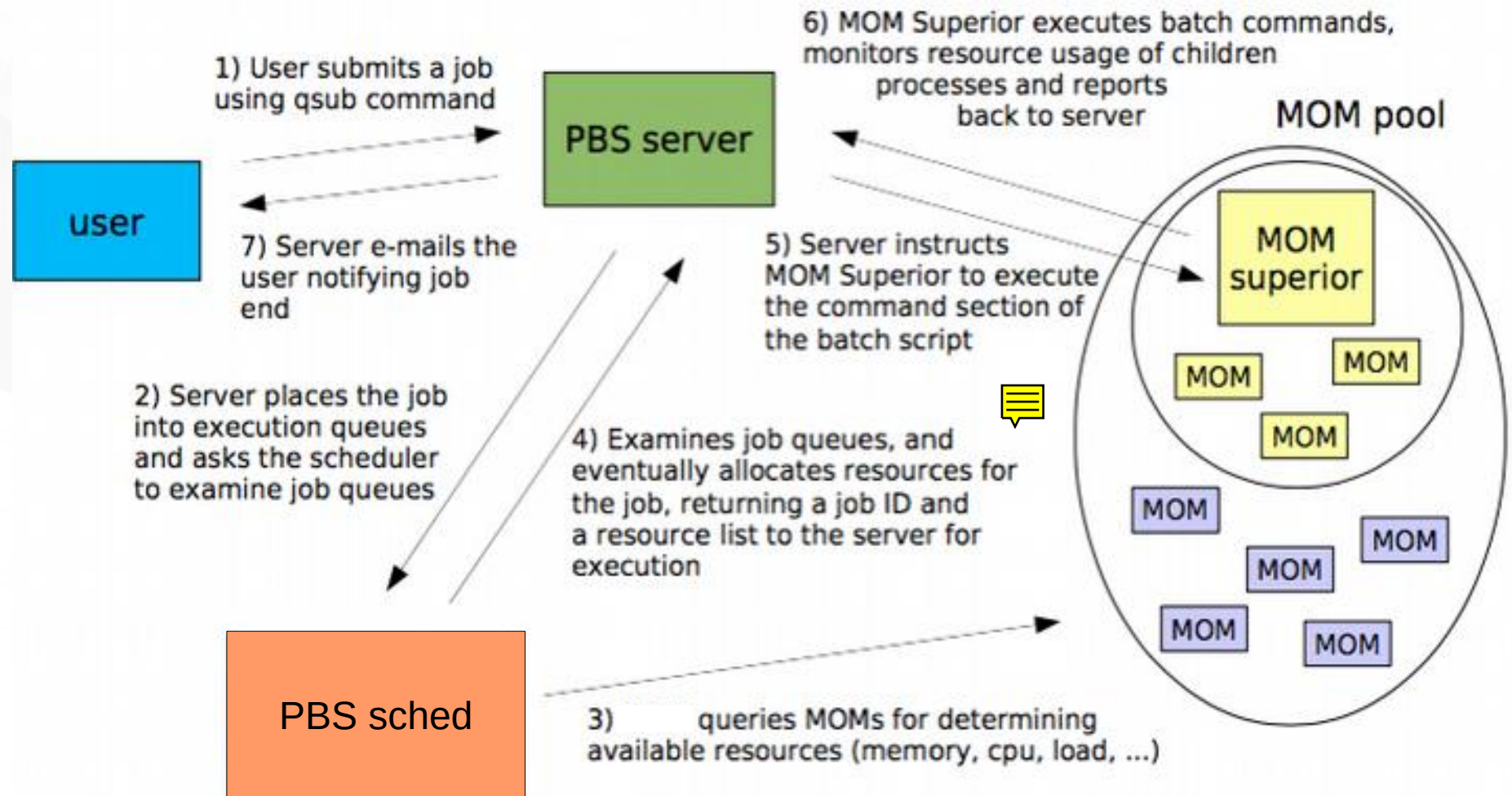
- General components
  - LM-X license server (if commercial edition)
  - resource manager *pbs\_server*
  - scheduler *pbs\_sched* / *maui* in case of Torque
  - executors *pbs\_mom*

# Recap on LRMS

- LRMS is a fundamental tool in the HPC management:
  - User: know it well and you will almost run !
  - Sys. Adm.: know it well and you will keep your system busy..
- Many possible choices
  - Concepts are similar /commands sometime also (to help survive: <http://www.schedmd.com/slurmdocs/rosetta.pdf>) .
  - Key point is THE scheduler
- Theoretically is **almost all possible** in resource scheduling with modern LRMS software to accommodate requests from users
- Practically is almost impossible satisfy all your users (and/or communities )

Resource sharing policies is not at all a technical problem !

# Typical job session



# Fair sharing

Fairshare is a mechanism which allows historical resource utilization information to be incorporated into job feasibility and priority decisions.

Fairshare information only affects the job's priority relative to other jobs.

Using the standard fairshare target

- the priority of jobs of a particular group which has used too many resources over the specified fairshare window is lowered
- the priority of jobs which have not received enough resources will be increased

# Backfill 1/2

Necessary to understand this in order to exploit the BACKFILL WINDOWING and finish jobs on

Backfill is a scheduling optimization which allows a scheduler to make better use of available resources by running jobs out of order.

The idea of expected walltime is a tool that allows users to contribute to th

Consider this example with a 10 CPUs machine:

Job1 ( priority=20 walltime=10 nodes=6 )  
Job2 ( priority=50 walltime=30 nodes=4 )  
Job3 ( priority=40 walltime=20 nodes=4 )  
Job4 ( priority=10 walltime=10 nodes=1 )

- 1)The scheduler prioritizes the jobs in the queue according to a number of factors and then re-orders the jobs into a 'highest priority first' sorted list.

Sorted list:

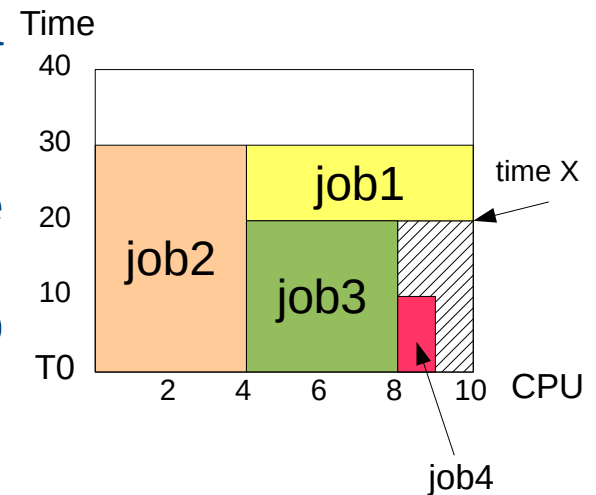
Job2 ( priority=50 walltime=30 nodes=4 )  
Job3 ( priority=40 walltime=20 nodes=4 )  
Job1 ( priority=20 walltime=10 nodes=6 )  
Job4 ( priority=10 walltime=10 nodes=1 )

# Backfill 2/2

2) It starts the jobs one by one stepping through the priority list until it reaches a job which it cannot start.

3) All jobs and reservations have a start time and a walltime limit, so the scheduler can determine:

- the completion time of all jobs in the queue
- the earliest the needed resources will become available for the highest priority job to start (time X)
- which jobs can be started without delaying this job (job4)



- Enabling backfill allows the scheduler to start other, lower-priority jobs so long as they do not delay the highest priority job, essentially filling in holes in node space.
- Backfill offers significant scheduler performance improvement:
  - increased system utilization by around 20% and improved turnaround time by an even greater amount in a typical large system
  - backfill tends to favor smaller and shorter running jobs more than larger and longer running ones: It is common to see over 90% of these small and short jobs backfilled.

Job2 ( priority=50 walltime=30 nodes=4 )  
Job3 ( priority=40 walltime=20 nodes=4 )  
Job1 ( priority=20 walltime=10 nodes=6 )  
Job4 ( priority=10 walltime=10 nodes=1 )



# A job's life

- The user describes the resources he he needs in a shell script, the **job** file
  - From the login node, the user **submits** the job to the queue system
  - The system sends the job to the **execution queue**
    - The job is executed on the compute node, without user intervention
  - The **results** of the job are written in the folder specified by the user
- The queue system free the resources to get ready for the next excution

# A job file ( PBS pro and torque)

**PBS directives**, using the tag **#PBS**, describe the job requirements in terms of execution queue, number of nodes and cores, job name, walltime, etc.

```
#!/bin/bash
# This is an example script
#PBS -q blade Queue called "blade"
#PBS -l nodes=1:ppn=2 1 node; 2 cores ("processors")
#PBS -N myjob Named job.
#PBS -l walltime=2:00:00 Requested walltime
```

The rest of the job is a standard shell script

---

Load Octave module, version 3.6.4

---


```
module load octave/3.6.4
```

PBS "lands" user's home directory: it is important to change the directory to the one in which we want to run the job

```
cd $HOME/MyJobDir
octave MyOctaveScript
```

# Job submission

Jobs are submitted using **qsub**

- `qsub myJob.sh`
- The resource, queue, etc., can be specified in the job file or from command line
  - `qsub -l nodes=8:ppn=4 myJob.sh`
  - `qsub -l select=8:ncpus=4:mpiprocs=8 -l place=scatter myJob.sh` 
- If the job is accepted, a JOBID is given
  - JOBID is necessary to monitor the job status

```
[degiorgi@grid0 ~]$ qsub -l nodes=2:ppn=4 script.sh
2715.grid0.mercuriofvg.it
[degiorgi@grid0 ~]$
```

# Job submission

- You can be more specific about the resources needed
  - `#PBS -l nodes=m001:ppn=4`
    - Requests **four cores** on node **m001**
  - `#PBS -l select=m001:ncpus=4:mpiprocs=4`
    - Requests node **m001** *and* node **m048**
  - `#PBS -l nodes=2:ppn=4+m010`
    - Requests **two nodes** with 4 cores *and* node **m010**


# Queues

- The user can use different **queues**
  - Each queue can be configured to accept jobs
    - From a certain user groups
    - Requesting resources between certain limits
- The proper needs to be selected
  - In the job script using `#PBS -q <queue name>`
  - From command line using `-q <queue name>`



# Interactive jobs

- It is possible to have **interactive jobs**
  - `qsub -l nodes=2:ppn=2 -I`
  - `qsub -l select=2:ncpus=2:mpiprocs=2 -l place=scatter -I`
- If resources are available, the users gets a prompt on the first node reserved



```
[brandino@master ~]$ qsub -l nodes=2:ppn=2 -I -q gpu
qsub: waiting for job 138510.master to start
qsub: job 138510.master ready

[brandino@b21 ~]$ cat $PBS_NODEFILE
b21
b21
b22
b22
[brandino@b21 ~]$
```

Not the best way to use cluster resources.

Humans are slow:

- Waste of resources
  - CPU/RAM, ... , energy
- Job terminates only when
  - The user closes the session
  - It hits the walltime



# Accessing compute nodes

- Normally, you cannot ssh to a compute node

```
[degiorgi@grid0 ~]$ ssh m001  
Connection closed by 10.2.12.1
```

- If a user has a job running on some nodes, ssh is allowed on those nodes

```
[degiorgi@grid0 ~]$ echo "sleep 30" | qsub -l nodes=m001  
2702.grid0.mercuriofvg.it  
[degiorgi@grid0 ~]$ ssh m001  
[degiorgi@m001 ~]$
```

- After job completion, the permission is revoked

# Queue status

- `qstat / qstat -a`
  - Status of the jobs in queue
- `qstat -q / qstat -Q`
  - Status of the execution queue
- `qstat -rn`
  - Status of the running jobs with indicated which nodes they are running on
- `qstat -u username`
  - Status of the job for a specific user



# Job tracking

- `tracejob` ID
  - Job lifetime: from queuing, to execution, to completion
  - Useful to understand where a job is failing

```
[degiorgi@grid0 ~]$ tracejob 2699.grid0.mercu 2> /dev/null
Job: 2699.grid0.mercuriofv.it
09/25/2013 22:33:13 S   enqueueing into blade, state 1 hop 1
09/25/2013 22:33:14 S   Job Run at request of maui@grid0.mercuriofv.it
09/25/2013 22:33:14 S   Not sending email: User does not want mail of this type.
09/25/2013 22:33:49 S   Exit_status=0 resources_used.cput=00:00:00 resources_used.mem=2820kb
resources_used.vmem=26656kb resources_used.walltime=00:00:36
09/25/2013 22:33:49 S   Not sending email: User does not want mail of this type.
09/25/2013 22:33:49 S   on_job_exit valid pjob: 2699.grid0.mercuriofv.it (substate=50)
09/25/2013 22:43:53 S   dequeuing from blade, state COMPLETE
[degiorgi@grid0 ~]$
```

# Job status

- A job can be in several states

Status	Meaning
Q	Job <u>Queued</u> , waiting for execution
R	Job is <u>Running</u>
E	Job is <u>Ending</u>
H	Job is on <u>Hold</u> (set by the user or by the system)

# Useful commands (i)

- `qdel ID`
  - Job receives signal TERM and KILL and quits

```
[degiorgi@grid0 ~]$ qsub -t 0-4 script.sh
2719[.]grid0.mercuriofv.g.it
[degiorgi@grid0 ~]$ qstat -r
```

```
grid0.mercuriofv.g.it:
```

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Req'd Memory	Req'd Time	S	Elap Time
2718[.]grid0.mer	degiorgi	blade	script.sh	--	1	4	--	01:00:00	R	00:00:00

# std error, std output

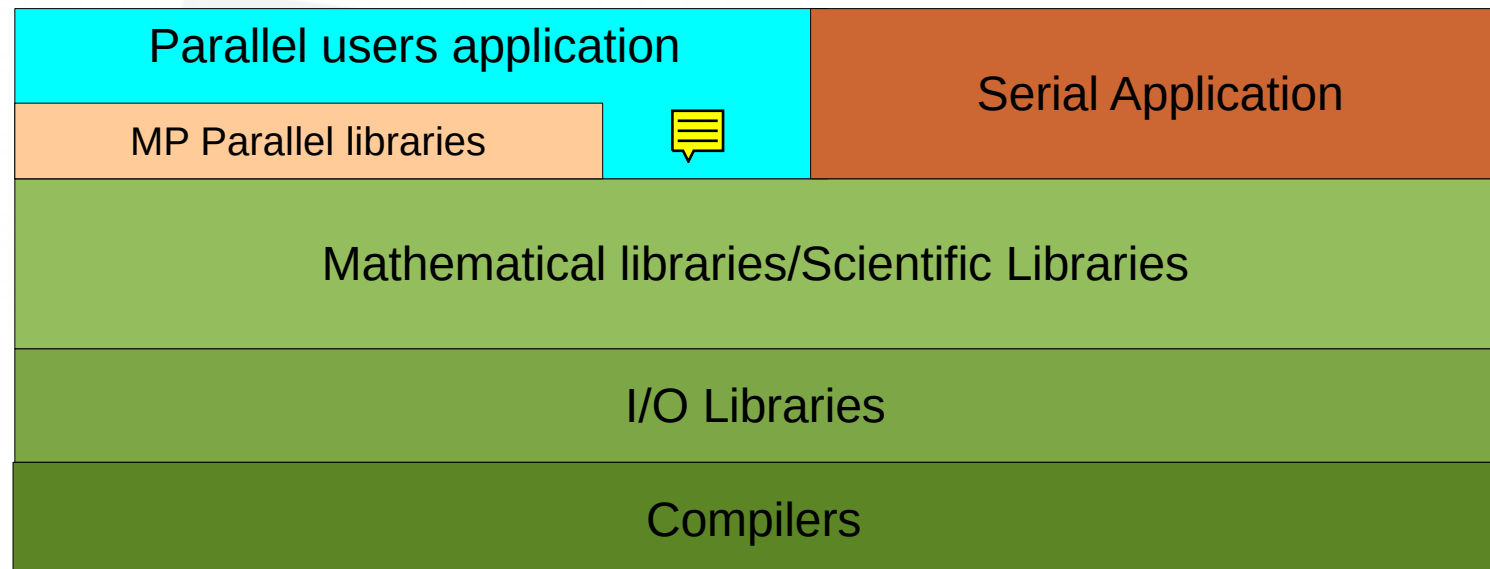
- Every submission produces a standard error and a standard output files
  - By default the file are called
    - `<jobname>.e<jobid>`
    - `<jobname>.o<jobid>`
  - Custom file names can be defined
    - `#PBS -o myPath/jobScript.out`
    - `#PBS -e myPath/jobScript.err`

# Execution environment

- When a job is executed, the resource manager creates several environment variables
  - Those variables can be used to set filenames/folder, take decision, etc..
  - Some variables:
    - PBS\_JOBNAME
    - PBS\_O\_WORKDIR → **directory** in which the job runs
    - PBS\_NODEFILE → list of nodes on which the job runs
    - PBS\_QUEUE
    - PBS\_ARRAYID

# HPC scientific Software layers (interleaved..)


- User's applications ( both parallel and serial)
- Message Passing Libraries
- Mathematical/Scientific Libraries
- I/O libraries
- Compilers



# Who cares about scientific HPC software ?

- End Users of HPC Software
  - Install and run HPC applications and tools
- HPC Application Teams
  - Manage third-party dependency libraries
- Package Developers
  - People who want to package their own software for distribution
- User support teams at HPC Centers
  - People who deploy software for users at large HPC sites

# HPC Software Complexity

- Not much standardization in HPC: every machine/app has a different software stack
  - This is done to get the best performance
- HPC frequently trades reuse and usability for performance 
  - Reusing a piece of software frequently requires you to port it to many new platforms
- List of packages/combination can diverge...

MIND THE PERFORMANCE -> CUSTOM IN

*Dependency nightmare !*



# Software for HPC

- Scientific packages requires generally:
  - A compiler
  - Some libraries
  - Some parallel tool (MPI libraries but not only)
- Different code may require different sets of the above..
- The same package can require different sets of the above software
  - A compiler could be needed for performance another for portability.
- Different compilers/libraries/mpi need different **environment variables**

# Scientific software: which is the production version ?

- Someone's home directory?
- Which MPI implementation?
- Which compiler?
- Which dependencies?
- Which versions of dependencies?
- Many applications require specific dependency versions.

Real answer: there isn't a single production environment or a standard way to build. Reusing someone else's software is HARD.

# Software should be available cluster-wide

- installed in /opt/cluster/software (or similar) and mounted read-only on the nodes via nfs
- Generally managed by modules package
- Several versions managed by some agreement
  - Software/version/compiler/version
  - Netcdf/4.4.3/gnu/4.9.2

# Environment modules

- Modules allow to dynamically modify user environment
- Useful tool to track different version of installed software
- **A few useful commands**
  - `module avail` – lists all available modules
  - `module list` – lists all loaded modules
  - `module load` – adds a module to your environment
  - `module unload` – removes a module from your environment

# Module example

```
##Modules
proc ModulesHelp { } {
    puts stderr "\tThis module provides the path for the \n";
    puts stderr "\tOpen64 Compiler Suite by AMD\n";
    puts stderr "\tVersion 4.5.2.1\n";
}

prepend-path PATH          "/opt/amd/x86_open64-4.5.2.1/bin"
prepend-path LD_LIBRARY_PATH "/opt/amd/x86_open64-4.5.2.1/lib/gcc-lib/x86_64-open64-linux/4.5.2.1/"
#EOF
```

- Allows the dynamic modification of user environment
  - avoid the shell reconfiguration
    - PATH and LD\_LIBRARY\_PATH
    - MANPATH, etc.
- every user can create her/his own personal module

# Installed modules on example cluster: C3HPC

## Main categories

- compilers
- applications
- libraries
- MPI

```
netcdf/4.3.2/gnu/4.8.3
```

```
^^^^^^  ^^^^^  ^^^  ^^^^^
```

```
|  
|  
|  
|  
|  
|  
|  
|  
|  
|  
|
```

```
|  
|  
|  
|  
|  
|  
|  
|  
|  
|  
|
```

```
|  
|  
|  
|  
|  
|  
|  
|  
|  
|  
|
```

```
+--> version of the compiler used to compile that software
```

```
+--> compiler used to compile that software
```

```
+--> Software version
```

```
+--> Software name
```

## 3 major build systems to be aware of...

- Make (usually GNU Make)
  - <https://www.gnu.org/software/make/>
  - (see openblas library)
- GNU Autotools
  - Automake: <https://www.gnu.org/software/automake/>
  - Autoconf: <https://www.gnu.org/software/autoconf/>
  - Libtool: <https://www.gnu.org/software/libtool/>
  - (see hdf5 library)
- CMAKE
  - <https://cmake.org>
  - (See geotop package)

# Make

- Many projects opt to write their own Makefiles.
  - Can range from simple to very complicated
- Make declares some standard variables for various compilers
  - Many HPC projects don't respect them
  - No standard install prefix convention
  - Makefiles may not have install target
- Automating builds with Make usually requires editing files
  - Typical to use sed/awk/some other regular expression tool on Makefile
  - Can also use patches

## Typical build incantation

```
<edit Makefile>  
make PREFIX=/path/to/prefix
```

## Configure options

```
None. Typically must edit Makefiles.
```

## Environment variables

CC	CFLAGS	LDFLAGS
CXX	CXXFLAGS	LIBS
FC	FFLAGS	CPP
F77	F77FLAGS	



# autotools

- Three parts of autotools:
  - autoconf: generates a portable configure script to inspect build host
  - automake: high-level syntax for generating lower-level Makefiles.
  - libtool: abstraction for shared libraries
- Typical variables are similar to make
- Much more consistency among autotools projects
  - Wide use of standard variables and configure options
  - Standard install target, staging conventions.

## Typical build incantation

```
./configure --prefix=/path/to/install_dir  
make  
make install
```

## Configure options

```
./configure \  
  --prefix=/path/to/install_dir \  
  --with-package=/path/to/dependency \  
  --enable-foo \  
  --disable-bar
```

## Environment variables

CC	CFLAGS	LDFLAGS
CXX	CXXFLAGS	LIBS
FC	FFLAGS	CPP
F77	F77FLAGS	

# cmake

- increasingly popularity
- Easier (?) to use (for developers) than autotools
- Similar standard options to autotools
  - different variable names
  - More configuration options
  - Abstracts platform-specific details of shared libraries
- Most CMake projects should be built “out of source”
  - Separate build directory from source directory

## Typical build incantation

```
mkdir BUILD && cd BUILD
cmake -DCMAKE_INSTALL_PREFIX=/path/to/install_dir ..
make
make install
```

## Configure options

```
cmake \  
  -D CMAKE_INSTALL_PREFIX=/path/to/install_dir \  
  -D ENABLE_FOO=yes \  
  -D ENABLE_BAR=no \  
  ..
```

## Common -D options

CMAKE_C_COMPILER	CMAKE_C_FLAGS
CMAKE_CXX_COMPILER	CMAKE_CXX_FLAGS
CMAKE_Fortran_COMPILER	CMAKE_Fortran_FLAGS
CMAKE_SHARED_LINKER_FLAGS	CMAKE_EXE_LINKER_FLAGS
CMAKE_STATIC_LINKER_FLAGS	

# Container: a solution to the HPC software ?

- Software systems are often complex and installation can be error prone
- When multiple packages need to interact conflicts arise:
  - Different software libraries
  - Different operating system
- Containerization allows systems to be packaged with everything they need so they can run anywhere.
- Many containers can run on a single system

# Compilers

- Free : Gnu suite
  - Always available
  - Many different version
  - Fundamental but some time lacks performance
- Commercial compilers
  - Intel suite :
    - A full software stack (includes libraries/ profiling /benchmariking tools MPI libraries)
    - Expensive but highly optimized
  - PGI
    - Good compiler
    - Comes with some nice extension (openACC /Cuda Fortran)
    - Community edition available for free

# Static/dynamic library

- Static libraries: libfoo.a
  - .a files are archives of .o files (object files)
  - Linker includes needed parts of a static library in the output executable
  - No need to find dependencies at runtime – only at build time.
  - Can lead to large executables
  - Often hard to build a completely static executable on modern systems.
- Shared libraries: libfoo.so (Linux)
  - More complex build semantics, typically handled by the build system
  - Must be found by ld.so and loaded at runtime
    - Can cause lots of headaches with multiple versions
  - 2 main ways:
    - **LD\_LIBRARY\_PATH**: environment variable configured by user and/or module system
    - **RPATH**: paths embedded in executables and libraries, so that they know where to find their own dependencies.

# Communication libraries for HPC

- MPI is a standard
- Several Libraries implement the standard
  - OpenMPI: widely adopted very portable and available for all networks/cluster
  - MPICH: historical one
  - MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE

# Introduction to HPC programming principles

# Two main programming paradigms

- Dictated by the HW architecture:

IN THE '90s Shared memory -> Data Parallel  
NOW: Multicore (sl

- **shared memory** → Data Parallel

- Single memory view, all processes (usually threads) could directly access the whole memory

- distributed memory → Message Passing

- all processes could directly access only their local memory

NORMA MODEL: Must use messages to pass information/instru

IF YOU WANT TO SCALE: Necessary to use HYBRID

CHECK MISSING SLIDE!



# Parallel Programming Paradigms, cont.

Programming Environments	
Message Passing	Data Parallel
Standard compilers MPI relies on a LIBRARY	Ad hoc compilers The "run in parallel" instruction is added to the
Communication Libraries	Source code Directive
Ad hoc commands to run the program To simplify things and EASE calls on the MPI library itself.mpirun -np XRun	Standard Unix shell to run the program
Standards: <b>MPI</b>	Standards: <b>OpenMP</b>

# How to program shared memory machine ?

- Automatic (implicit) parallelization:
  - compilers do (part of) the job for you
- Manual parallelization:
  - Insert parallel directives by yourself to help compilers
  - OpenMP THE standard
- Multi threading programming:
  - more complex but more efficient
  - use a threads library to create task by yourself
- Use already threaded libraries..

# How to program using message passing ?

- Using the de-facto standard : MPI message passing interface
  - A standard which defines how to send/receive message from a different processes
- Many different implementation
  - OpenMPI
  - Intel-MPI
  - They all provide a library which provide all communication routines
- To compile your code you have to link against a library
  - Generally a wrapper is provided (mpif90/mpicc)

# Architectures vs. Paradigms

## Clusters of Shared Memory Nodes

### Shared Memory Computers

Data Parallel

Message Passing

### Distributed Memory Computers

Message Passing

# Parallel programming: a short summary..

Architectures	
Distributed Memory	Shared Memory
Programming Paradigms/Environment	
Message Passing	Data Parallel
Parallel Programming Models	
Domain Decomposition	Functional Decomposition

# Other paradigm are now available

- Mixed/hybrid approach..
  - MPI + OpenMP
- Specific SDK for specific devices
  - CUDA for Nvidia GPU
- Write once run everywhere:
  - OpenCL
  - OpenACC:
    - OpenACC is about giving programmers a set of tools to port their codes to new heterogeneous system without having to rewrite the codes in proprietary languages.

# Principle of parallel computing

- Speedup, efficiency
  - Ahmdal Law/Gustafson Law
- Finding and exploiting parallelism
- Finding and exploiting data locality
- Load balancing
- Coordination and synchronization
- Performance modeling

All of these things make parallel programming more difficult than sequential programming.

# Speed up

The speedup of a parallel application is

$$\text{Speedup}(p) = \text{Time}(1)/\text{Time}(p)$$

where

$\text{Time}(1)$  = execution time for a single processor

$\text{Time}(p)$  = execution time using  $p$  parallel processor

If  $\text{Speedup}(p) = p$  we have perfect speedup (also called linear scaling)

speedup compares an application with itself on one and on  $p$  processors  
more useful to compare

The execution time of the best serial application on 1 processor  
versus

The execution time of best parallel algorithm on  $p$  processors



# Efficiency

- The parallel efficiency of an application is defined as
  - $\text{Efficiency}(p) = \text{Speedup}(p)/p$
- $\text{Efficiency}(p) \leq 1$
- For perfect speedup  $\text{Efficiency}(p) = 1$
- We will rarely have perfect speedup: Lack of perfect parallelism in the application or algorithm
  - Imperfect load balancing (some processors have more work) (Starvation)
  - Cost of communication (Latency)
  - Synchronization time (Overhead)
  - Cost of contention for resources, e.g., memory bus, I/O (Waiting)
- Understanding why an application is not scaling linearly will help finding ways improving the applications performance on parallel computers.

# Superlinear speedup

Question: can we find “*superlinear*” speedup, that is

$$\text{Speedup}(p) > p \quad ?$$

Choosing a bad “baseline” for  $T(1)$

WRONG !!!

Old serial code has not been updated with optimizations

Shrinking the problem size per processor

GOOD

- May allow it to fit in small fast memory (cache)

# Amdahl's law

- Suppose only part of an application runs in parallel
  - Let  $s$  be the fraction of work done serially,
  - So  $(1-s)$  is fraction done in parallel
  - What is the maximum speedup for  $P$  processors?

$$\text{Speedup}(p) = T(1)/T(p)$$

$$T(p) = (1-s)*T(1)/p + s*T(1)$$

$$T(p) = T(1)*((1-s) + p*s)/p$$

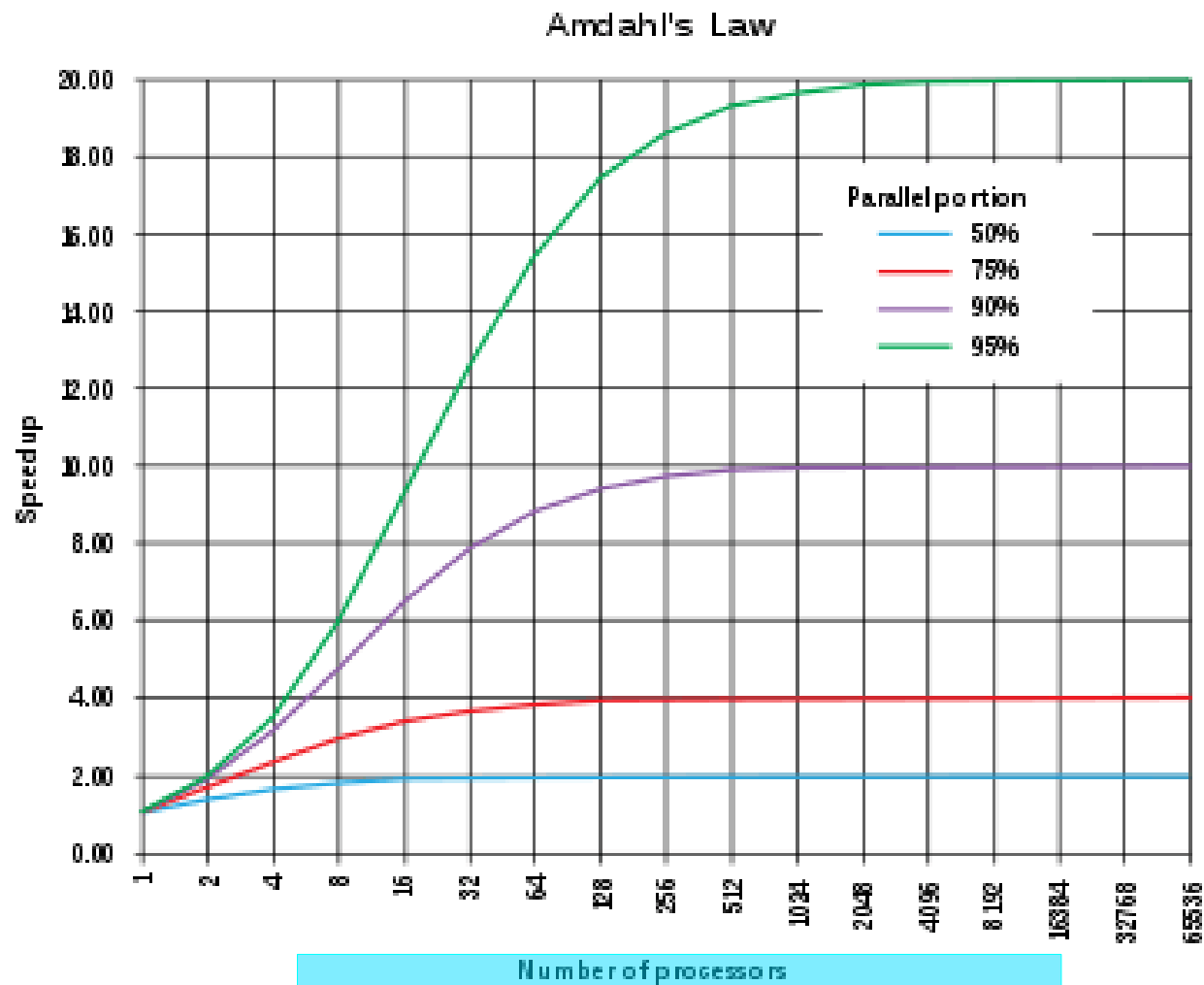
assumes  
perfect  
speedup for  
parallel part

$$\text{Speedup}(p) = p/(1 + (p-1)*s)$$

Even if the parallel part speeds up perfectly,  
we may be limited by the sequential portion of code.

# Amdahl's law

- Which fraction of serial code is it allowed ?



What about Scalability ???

# Scaling...

- Scaling or scalability: some sort of relation between the performance and the “size” of the HPC infrastructure
  - Usual way to measure size: # of processors
- The ability for some application to increase **speed** when the size of the HPC is increased
- The ability for some application to solve **larger problems** when the size of the HPC increases..

# Problem scaling

- Amdahl's Law is relevant only if serial fraction is independent of problem size, which is rarely true
- Fortunately “The proportion of the computations that are sequential (non parallel) normally decreases as the problem size increases ” (a.k.a. **Gustafon's Law**)

# So What Is Scalability?

- to get N times more work done on N processors
- compute a fixed-size problem N times faster
  - **Strong scaling**
    - Speedup  $S = T_1 / T_N$  ; linear speedup occurs when  $S = N$
    - Can't achieve it due to Amdahl's Law (no speedup for serial parts)
- compute a problem N times bigger in the same amount of time:
  - **Weak scaling**
    - Speedup depends on the amount of serial work remaining constant or increasing slowly as the size of the problem grows
    - Assumes amount of communication among processors also remains constant or grows slowly

# Why weak scaling tends to work better..

- **Strong scaling**: fixed data/problem set; measure speedup with more processors
  - Ahmdal law
- **Weak scaling**: data/problem set increases with more processors; measure if speed(efficiency) is the same
  - Gustafson law



# Developing a performance model for parallel algorithm..

- The performance models specify a metric such as execution time  $T$  as a function of problem size  $N$ , number of processors  $P$ , number of tasks  $U$ , and other algorithm and hardware characteristics:
  - $T = f(N, P, U, \dots)$
- The simplest model:
  - $T = T_{\text{computation}} + T_{\text{communication}}$
- Simple performance model for message passing
  - time to deliver a message of  $N$  Bytes:  
Time (total latency) = latency of startup +  $N / \text{bandwidth}$