Center for Mathematical Modeling
University of Chile

HPC 101

Scientific Computing on HPC systems

By

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

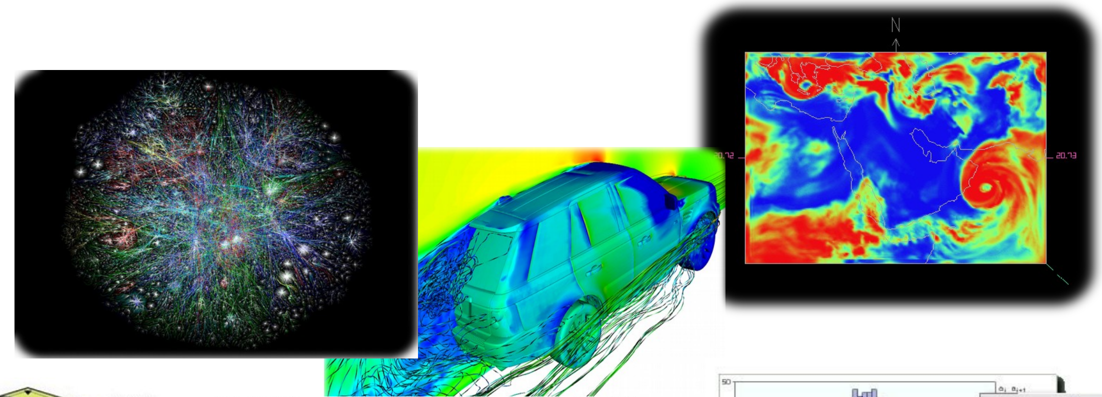
La Serena School for Data Science: Applied Tools for Astronomy.
La Serena – Chile – 22/08/2017

Overview

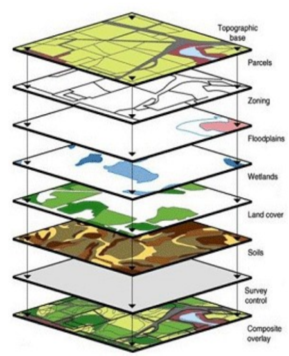
- Concepts & Definitions.
- Working with a HPC system.
- Programming in a HPC system.
- Hands-on: Source extraction with Spark
- Wrapping up: The take aways.

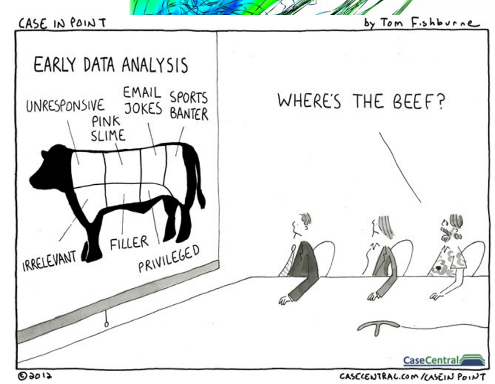
Scientific Computing

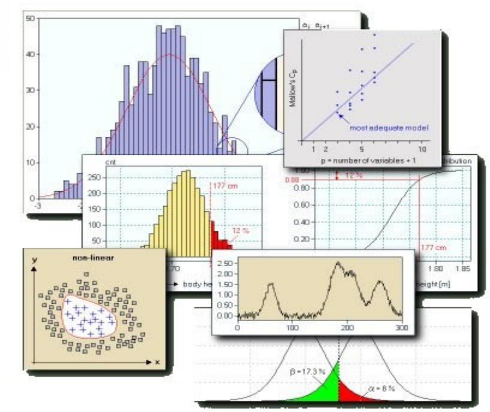
- Simulations



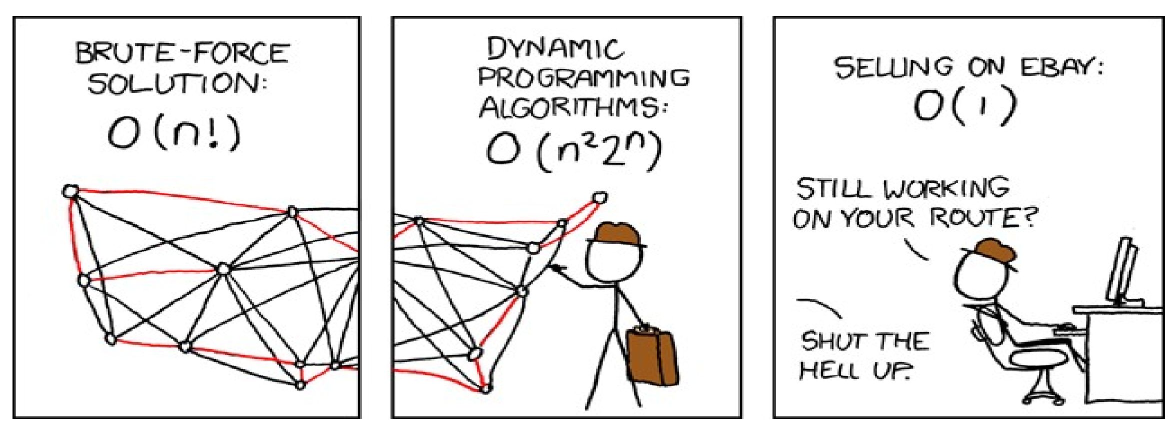
- Data Analysis







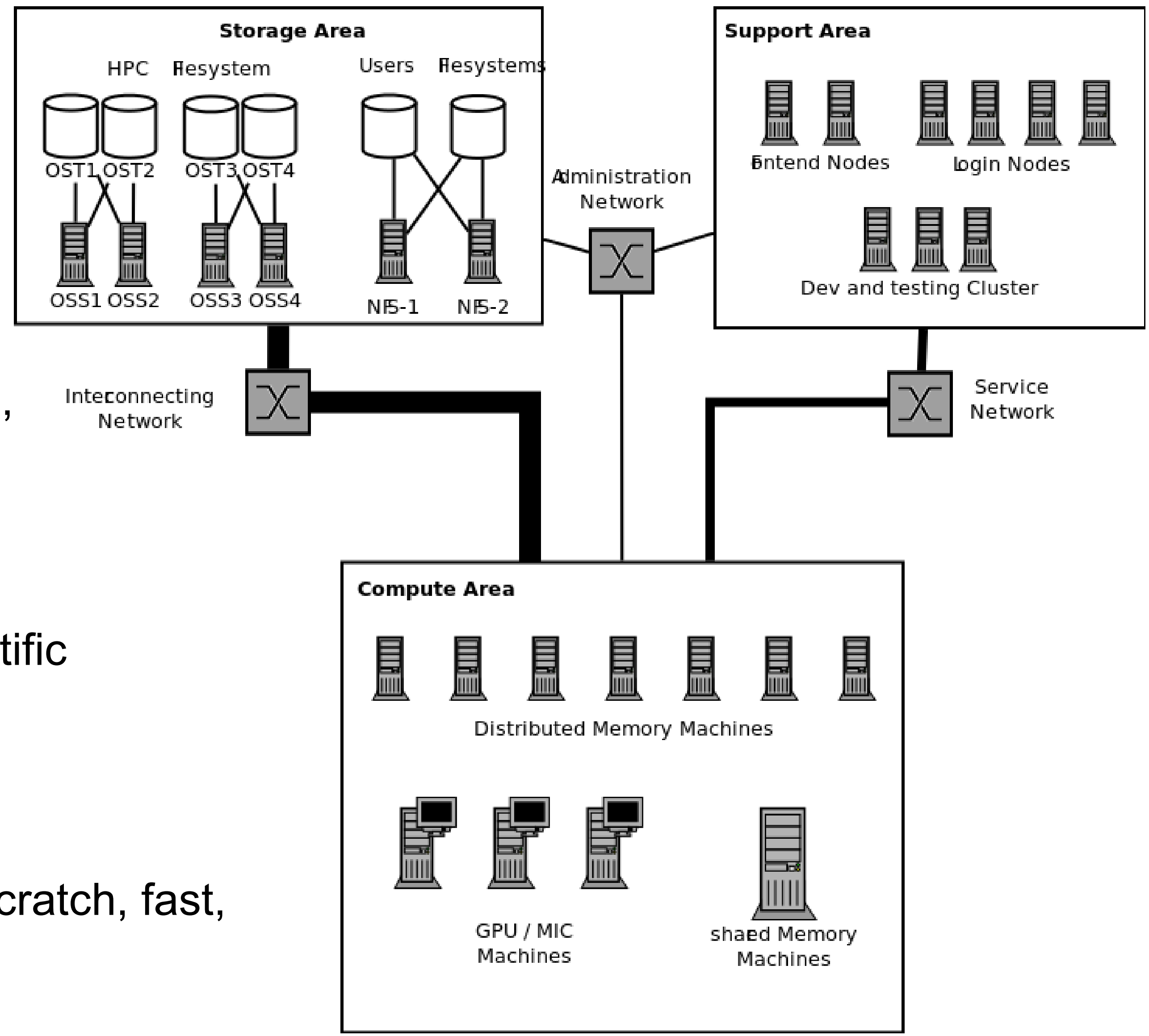
- Computational Optimization



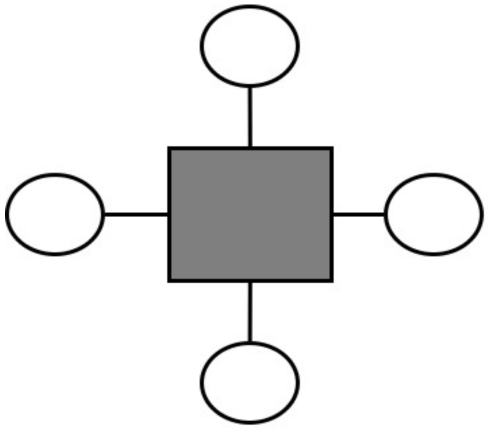
Concepts & Definitions

HPC system Architecture

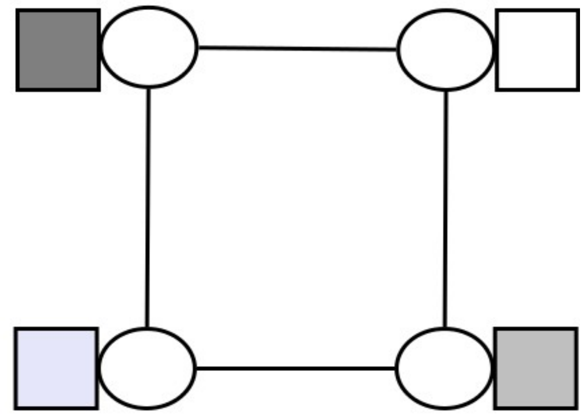
- Areas
  - Computing, Storage, Support, Networking
- Servers roles
  - Compute, frontend, login, storage, backup, devel monitoring, etc.
- Software
  - Operating System, scientific software, analysis tools, libraries, etc.
- Storage
  - Local, working shared, scratch, fast,
  - slow



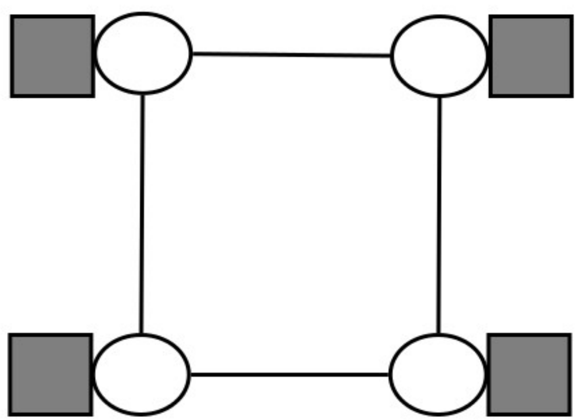
Distributed and Shared Memory Systems



Shared Memory



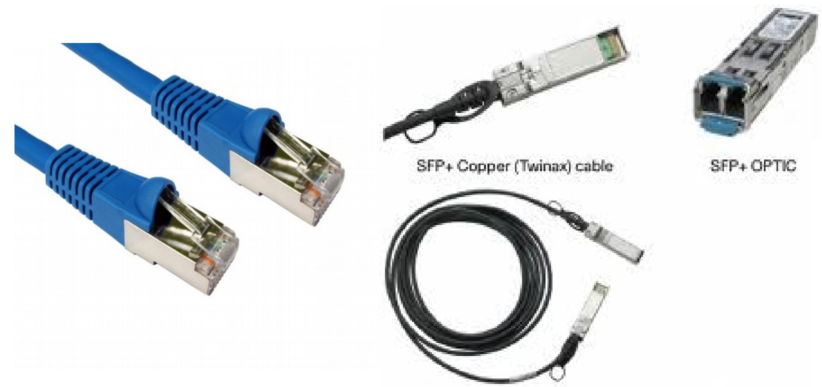
Distributed Memory

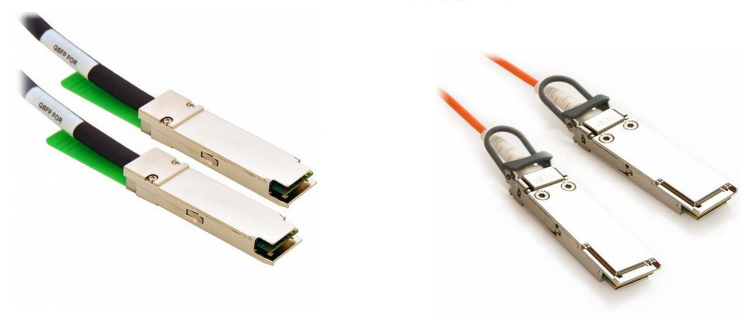


Distributed Shared Memory

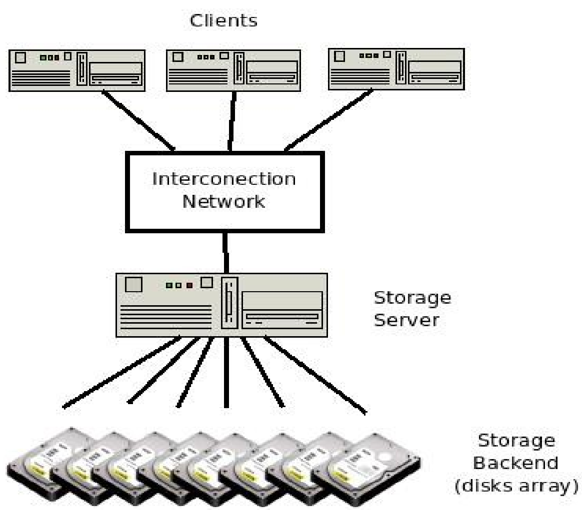
Interconnects

- 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

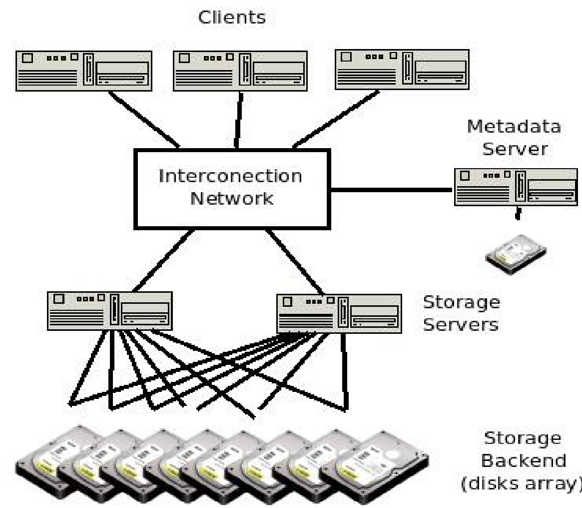




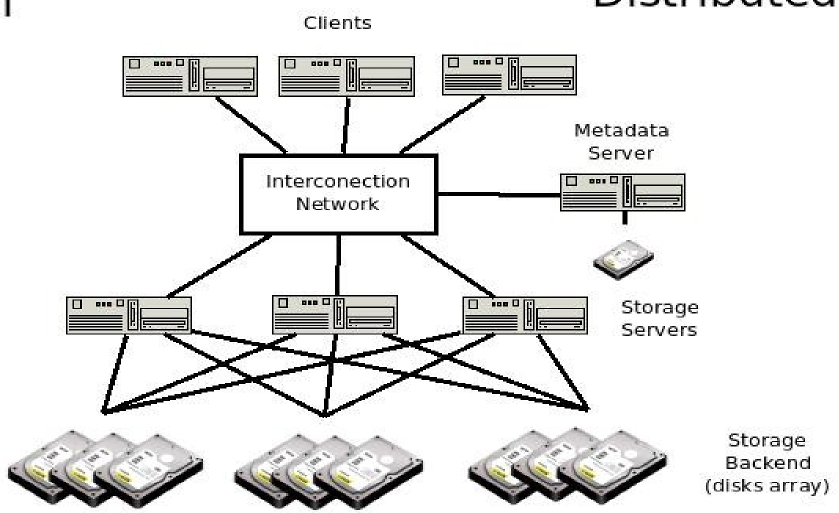
File-systems Types



Serial



Distributed



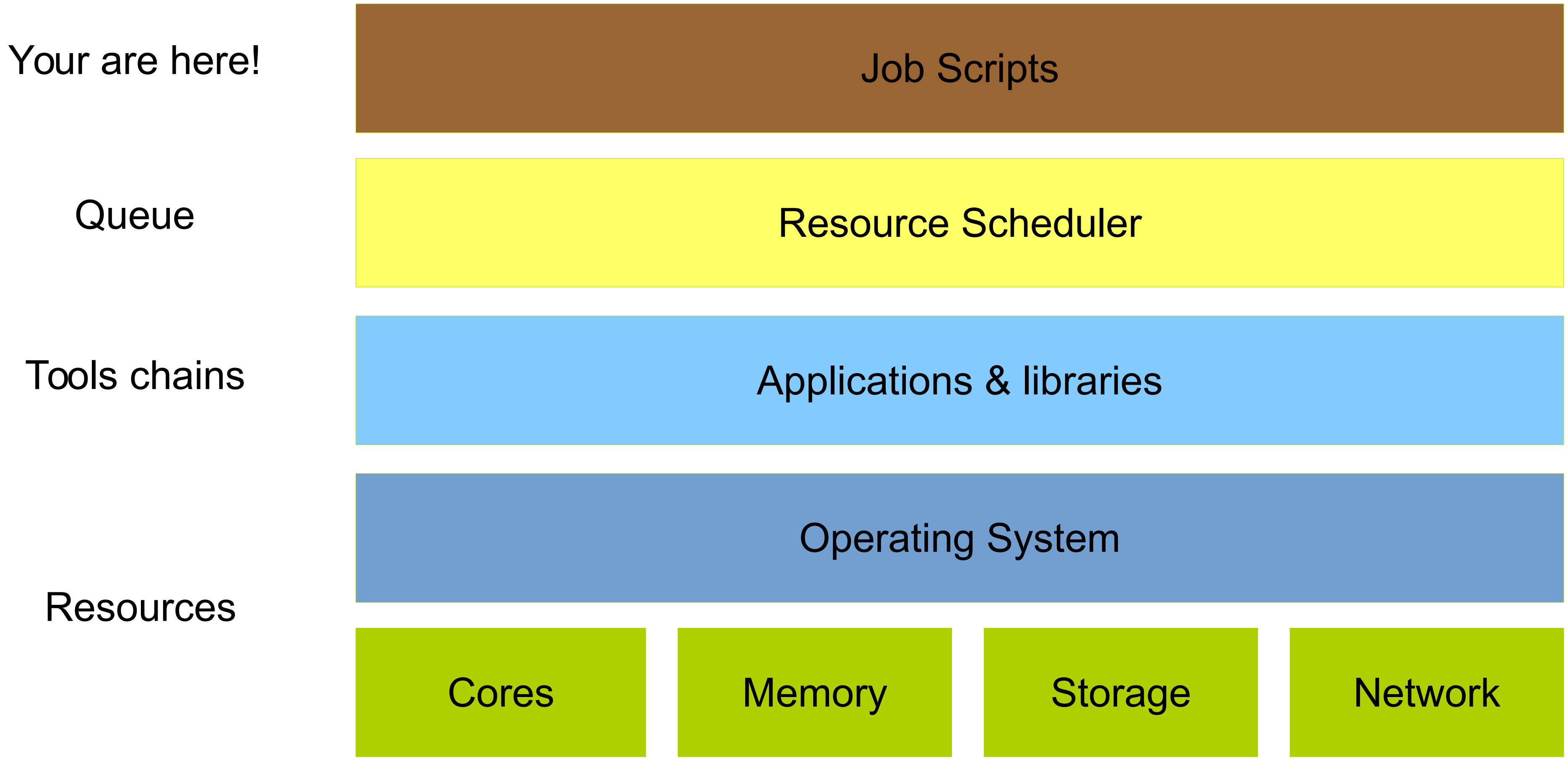
Parallel

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

Storage Layouts

- **Working ($Home)**
  - Safe and Slow storage.
  - Cheep
  - 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).
- System wide (one for all).
  - Compiled and Installed by admins.
- User Space (each one has its own).
  - Compiled and installed by the user in their homes directories.

Resources Manager

- **Scheduler**: allocate resources to perform a job.
- **Job**: set of instructions and resources to perform a task.
- **Task**: involves preparing the environment

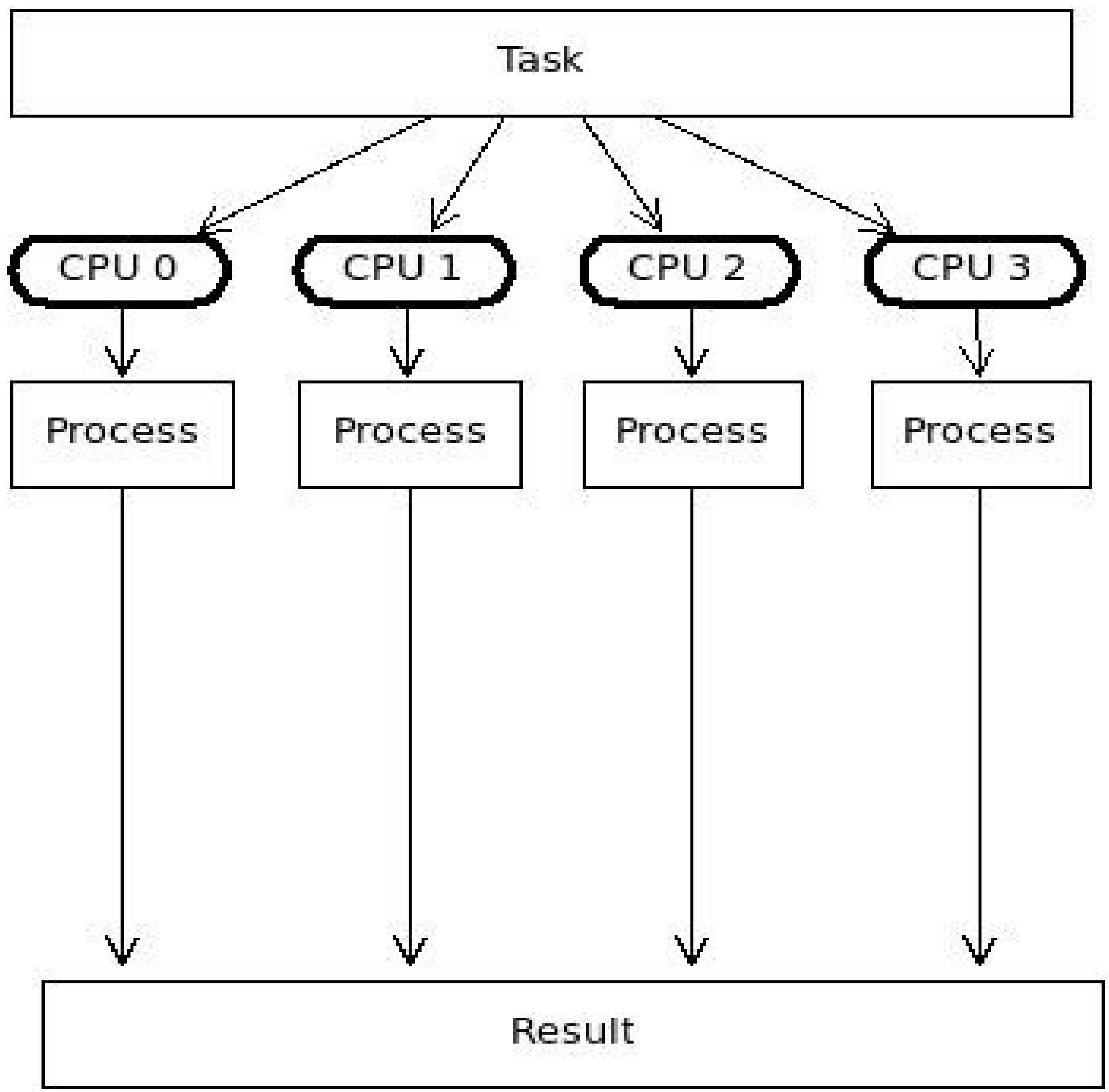
  and input data needed to run an application.

Resource specifications

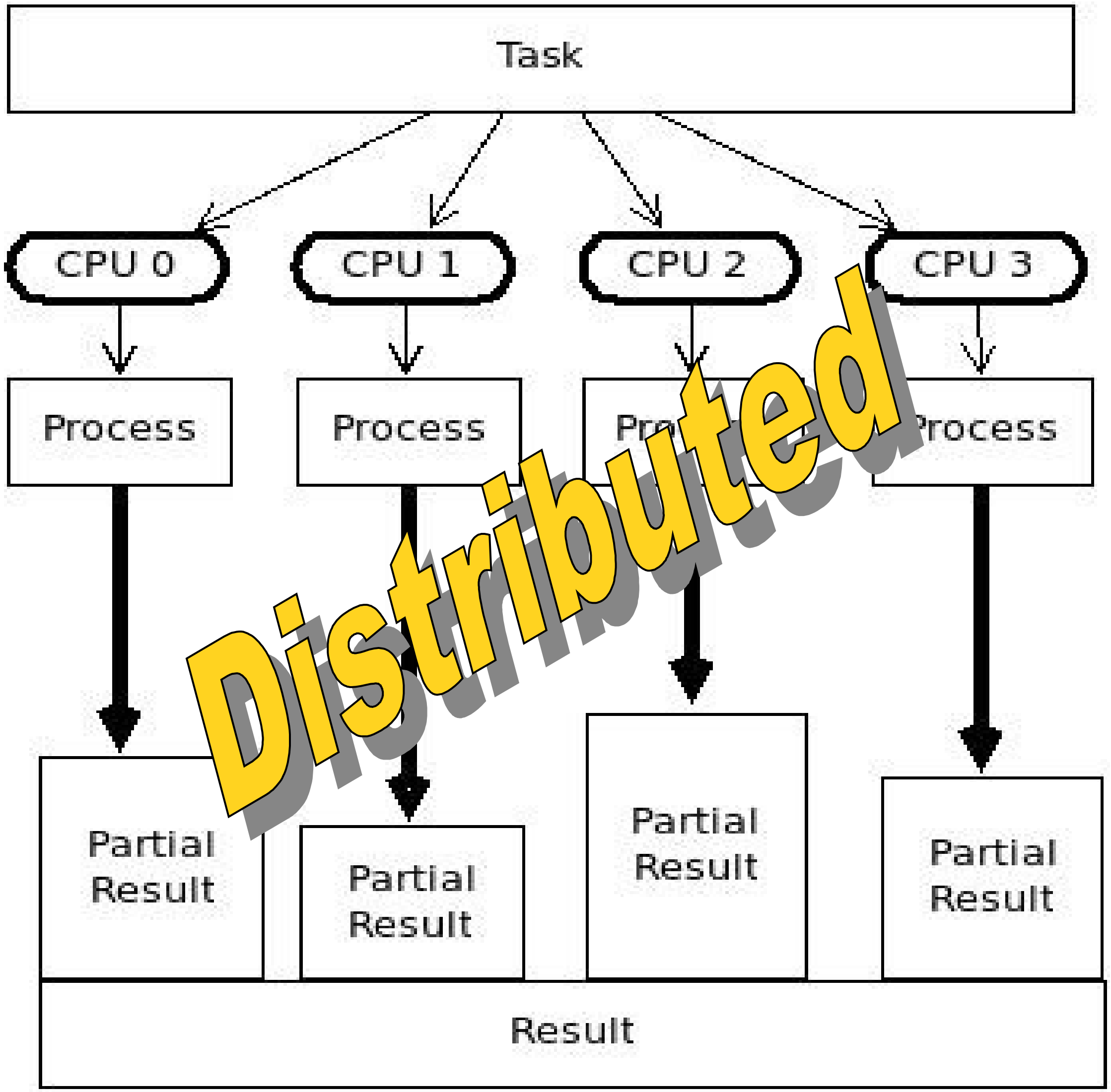
+

Instructions to perform a task

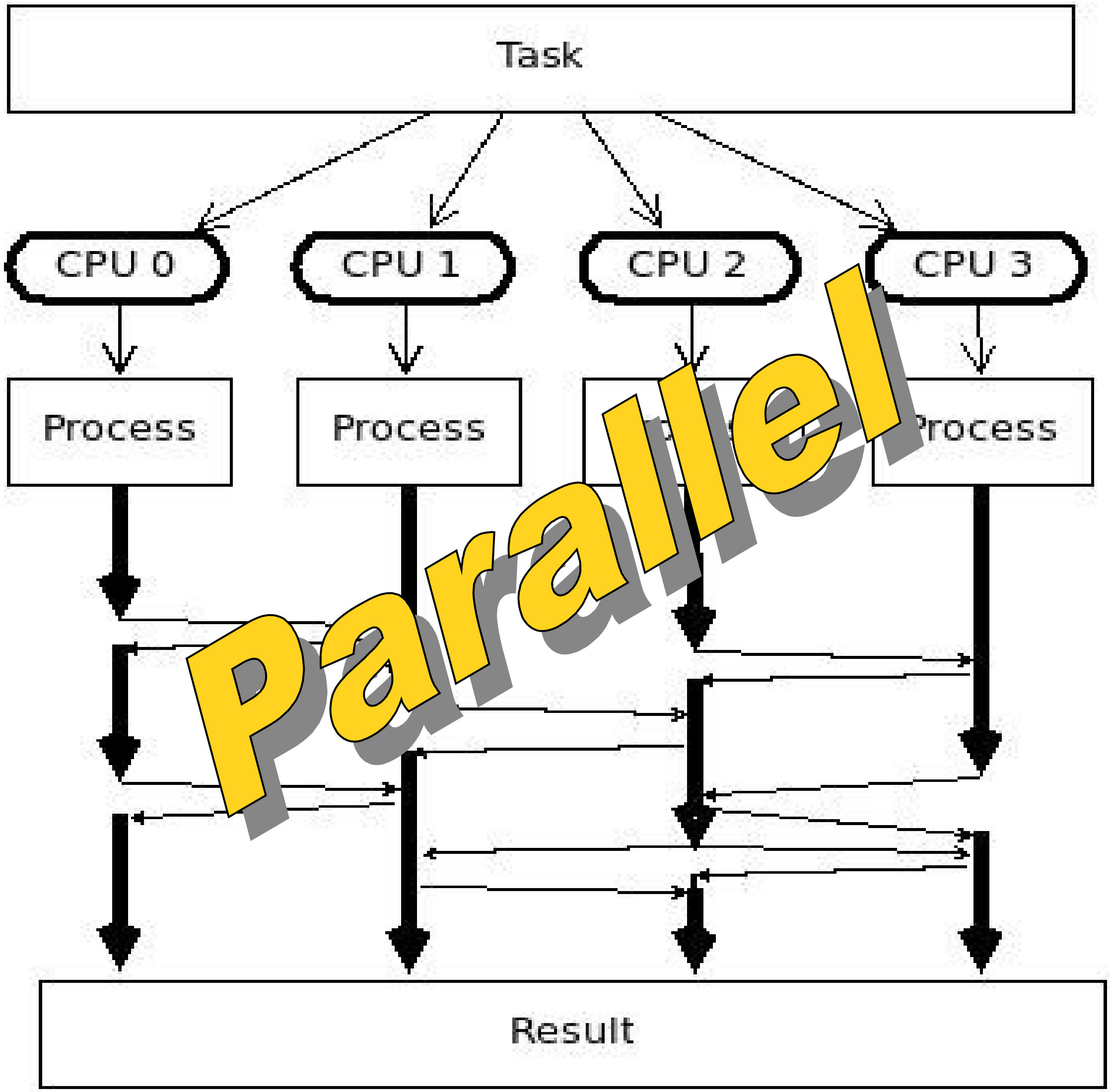
Jobs: Parallel v/s Distributed



Jobs: Parallel v/s Distributed

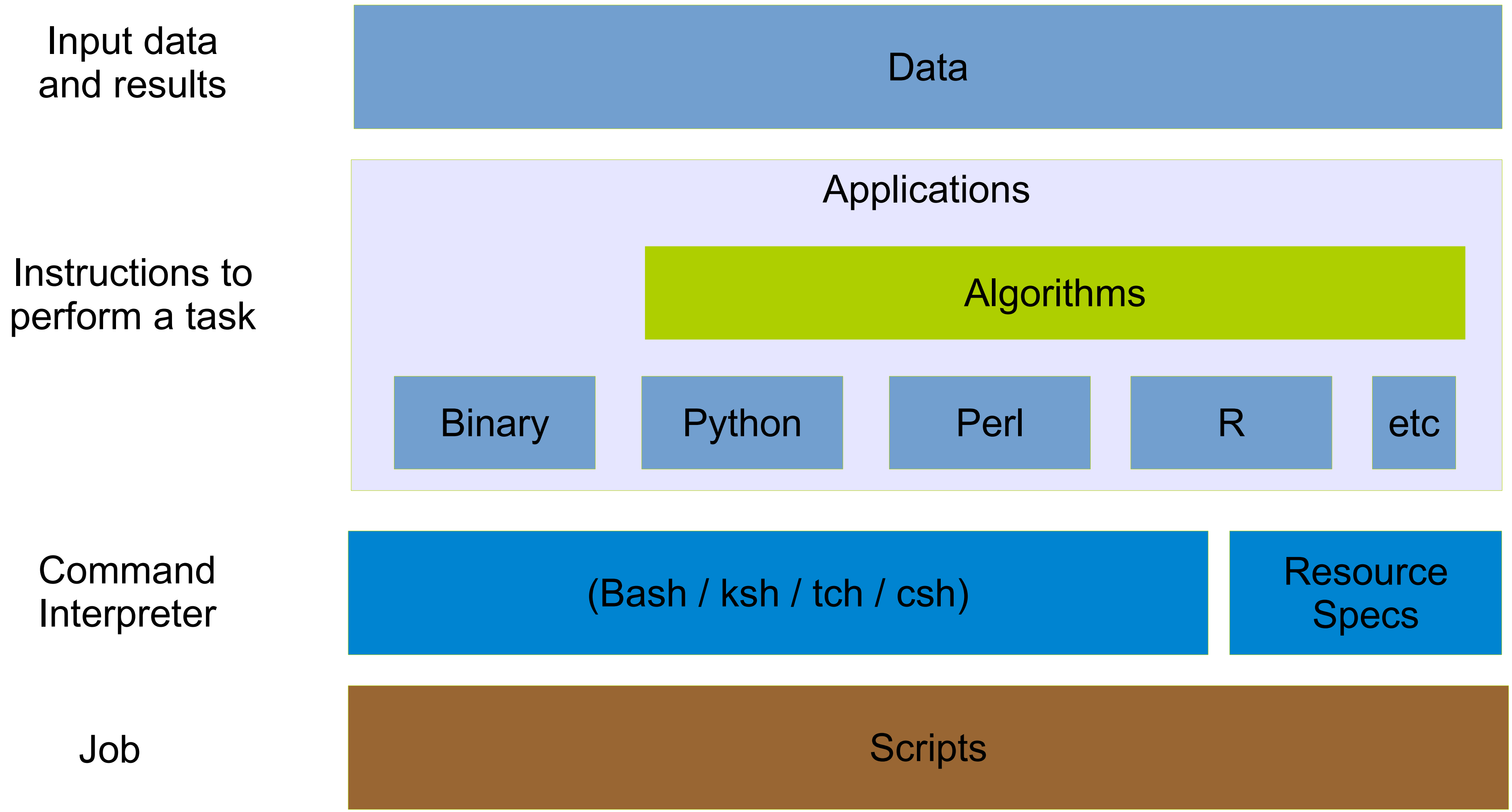


Jobs: Parallel v/s Distributed



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
- Slurm

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**
- Slurm

Job Scheduler Directives

```
#!/bin/bash
# ask for 4 full nodes
#SBATCH -N 4
# number of tasks per node
#SBATCH –ntasks-per-node=8
# Number of cores
#SBATCH -n 1
# shared or exclusive use
#SBATCH --exclusive
# ask for 1 day and 3 hours of run time
#SBATCH -t 1-03:00:00
# Account name to run under
#SBATCH -A <account>
# a sensible name for the job
#SBATCH -J my_job_name
# set the stdout file
#SBATCH -o myjobname.%j.out
# User notification
#SBATCH --mail-type=end
#SBATCH --mail-user=my@email.com
```

- Grid Engine
- PBS
- **Slurm**

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

```
[jcm@leftraru ~]$ module available
-------------------------- /home/jcm/modulefiles -------------------------------
astro/3.0-dev astro_old/0.1 astro_old/0.2 fastQC        spark

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

-------------------------- /home/Modules/modulefiles --------------------------
14-mp                         gurobi/6.0.3                  openblas/0.2.15
ace/6.3.3                     gurobi/6.0.4                  opencv/2.4.13
aims/071914                   gurobi/6.5.1                  openfoam/2.3.1
aims/071914_7                 gurobi/7.0.2                  openfoam/2.4.0
alps/2.2                      hdf5/1.8.13                   openmpi/1.10.1
amber/14                      hdf5/1.8.15                   openmpi/1.10.2
ampl/20021038                 hmmer/3.1b2                   openmpi/1.10.3
...
gsl/2.1                       nwchem/6.6                    yade/1.20.0
gts/121130-snapshot           nwchem/6.6-test               zlib/1.2.8
gurobi/6.0.0                  openbabel/2.3.2

[jcm@leftraru ~]$
```

Environment Modules

- module show {module name/version}

```
[jcm@leftraru ~]$ module show astro/3.0
-------------------------------------------------------------------
/home/Modules/modulefiles/astro/3.0:

module          load intel/2017c
module-whatis       Sets up the AstroLab 3.0 toolchain in your environment.
setenv              ASTRO_HOME /home/apps/astro
prepend-path        PATH /home/apps/astro/bin
prepend-path        PATH /home/apps/astro/sbin
prepend-path        LD_LIBRARY_PATH /home/apps/astro/lib
prepend-path        PKG_CONFIG_PATH /home/apps/astro/lib/pkgconfig
prepend-path        MANPATH /home/apps/astro/home/apps/man
-------------------------------------------------------------------

[jcm@lefraru ~]$
```

Environment Modules

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

```
[jcm@leftraru ~]$ module load astro/3.0

[jcm@leftraru ~]$ module list
Currently Loaded Modulefiles:
  1) astro/3.0

[jcm@leftraru ~]$ echo $LD_LIBRARY_PATH
/home/apps/astro/lib:/home/apps/intel/2017/itac/2017.3.030/mic/slib:
/home/apps/intel/2017/itac/2017.3.030/intel64/slib:/home/apps/intel/2017/itac/
2017.3.030/mic/slib:/home/apps/intel/2017///itac/2017.3.030/intel64/slib:/home/apps
/intel/2017/compilers_and_libraries_2017.4.196/linux/compiler/lib/intel64:
...

[jcm@leftraru ~]$ echo $PATH
/home/apps/astro/sbin:/home/apps/astro/bin:/home/apps/intel/2017/
advisor_2017.1.3.510716/bin64:/home/apps/intel/2017/vtune_amplifier_xe_2017.3.0.510739
/bin64:/home/apps/intel/2017/inspector_2017.1.3.510645/bin64:/home/apps/intel/2017/
itac/2017.3.030/intel64/bin: …

[jcm@leftraru ~]$
```

Slurm Jobs (sbatch)

- Script execution within a resource allocation
- Executed by sbatch or salloc + srun
- Only execute scripts (not binaries)

- CPUs / cores (-c)
  - Number of cores per process
- Tasks (-n)
  - Number of processes to launch within this job
- Nodes (-N)
  - Number of nodes used to allocate processes

```
# run single process with 1 core (-c default)
#SBATCH -n 1
#SBATCH -N 1

# run 10 processes, each one with 1 core, within
# a single node (mpi)
#SBATCH -n 10
#SBATCH -N 1

# run 10 processes, each with 1 core, allocating
# processes among 3 nodes (mpi)
#SBATCH -n 10
#SBATCH -N 3

# run 5 processes, each with 4 cores, allocating
# processes among 3 nodes (openmp + mpi)
#SBATCH -c 4
#SBATCH -n 10
#SBATCH -N 3
```

Slurm Job Steps (srun)

- Script or binary execution within a resource allocation
- Executed by srun or salloc + srun
- Execute scripts and binary programs

- CPUs / cores (-c)
  - Number of cores per task
- Tasks (-n)
  - Number of tasks
- Nodes (-N)
  - Number of nodes used to allocate tasks
- Exclusive (--exclusive)
  - Resources are exclusive for the task. Otherwise all allocated resources will be available for each jobstep

```
# run myapp.exe with 3 cores (openmp or threaded)
$ srun -n 1 -c 3 myapp.exe

# run 4 times myapp.exe with 1 cores
$ srun -n 4 -c 1 myapp.exe

# run 4 times myapp.exe with 1 cores in a single
# node with exclusive allocation (the node is used
# only for this user/process
$ srun -n 4 -c 2 -N 1 --exclusive myapp.exe

# run 4 times myapp.exe with 1 cores
$ srun -n 4 -c 1 myapp.exe

# mpi run of mympiapp.exe with 5 cores
$ mpirun -n 5 mympiapp.exe

# mpi run of mympiapp.exe with 5 cores with
# slurm / mpi integration
$ srun -n 5 mympiapp.exe
```

Slurm Job Array (sbatch)

- Script multiple execution within a resource allocation varying a task index
- Executed only by sbatch
- Fixed number of tasks

- Array (--array)
  - start-end:step (range)
  - 1,3,4-7 (selective)
  - 1–100%5 (batch of 5 tasks)
- Env. Variables
  - SLURM_ARRAY_TASK_ID
  - SLURM_ARRAY_TASK_COUNT
- Output (stdout) of each task
  - output=mytask.%A.%a
    - %A = JobID
    - %a = Job Array Task id

```
$ cat my-jobarray.slurm
#!/bin/bash
#SBATCH -J my_job_array
#SBATCH -n 1
#SBATCH --array=1-10
#SBATCH -p levque

HOST=`hostname`
echo “Tasks $SLURM_ARRAY_TASK_ID \
      running in $HOST”

$ sbatch my-jobarray.slurm
Submitted batch job 8439931

$ cat slurm-8439931_*.out
Tasks 1 running in levque001
Tasks 2 running in levque001
Tasks 3 running in levque003
Tasks 4 running in levque005
...
Tasks 9 running in levque029
Tasks 10 running in levque029
$
```

Slurm JobStep Array (sbatch+srun)

- Script execution with variable number of tasks within a resource allocation

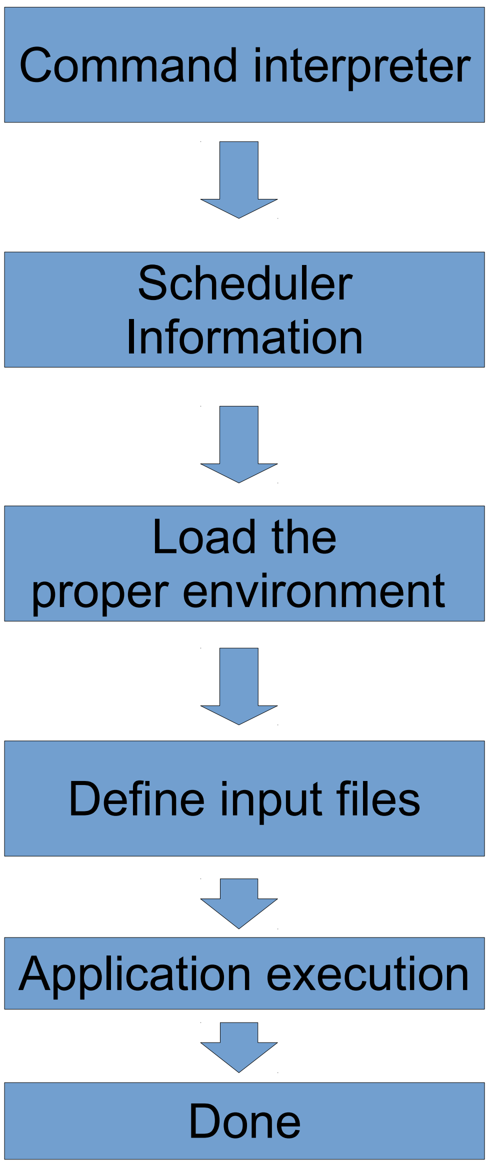
```
$ cat my-jobste-array.slurm
#!/bin/bash
#SBATCH -J my_jobstep_array
#SBATCH -n 10
#SBATCH -p levque

echo "master Tasks $SLURM_JOB_ID running \
      in `hostname`"

NUM_TASKS=20
for task in `seq 1 $NUM_TASKS`;
do
    srun --exclusive -n 1 -N 1 -p levque \
       ./jobstep.slurm &
done
wait
echo "done"

$ cat jobstep.slurm
#!/bin/bash
echo "Task $SLURM_STEP_ID running \
in host `hostname`"
exit 0

$
```

```
$ sbatch my-jobstep-array.slurm
Submitted batch job 8440039

$ cat slurm-8440039.out | grep Task
master Tasks 8440039 running in levque029
Task 8 running in host levque030
Task 7 running in host levque030
Task 0 running in host levque029
Task 11 running in host levque030
Task 10 running in host levque030
Task 3 running in host levque029
Task 1 running in host levque029
Task 12 running in host levque029
Task 2 running in host levque029
Task 9 running in host levque030
Task 6 running in host levque029
Task 13 running in host levque030
Task 14 running in host levque030
Task 4 running in host levque029
Task 15 running in host levque030
Task 16 running in host levque030
Task 19 running in host levque030
Task 5 running in host levque029
Task 17 running in host levque030
Task 18 running in host levque030
$
```

Interacting with the Slurm

- sbatch `job_script.slurm`
  - Submit *`job_script`* to the queue (partition)
- srun
  - Run a command in a compute node (a jobstep)
- squeue
  - Show only the status of **your jobs** in the queue
- squeue -s
  - Show the steps associated current running jobs
- scontrol show job `Job-ID`
  - Show the status of `Job-ID`
- scontrol show node
  - Show the status of a particular node
- sinfo
  - Show the status of each partition (queue)
- sinfo -N
  - Show the status of each node showing their partitions and status
- scancel `Job-ID`
  - Cancel (running) and delete a job from the queue

Creating (Slurm) Jobs




```
#!/bin/bash

#SBATCH -n 1
#SBATCH -N 1
#SBATCH -p levque
#SBATCH --exclusive
#SBATCH --mem=4G
#SBATCH -J sextractor
#SBATCH -o sextractor.%j.out
#SBATCH -e sextractor.%j.err

module load astro

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 “Ending at `date '+%c'`”
echo “done”
```

Submitting & Monitoring Jobs

```
[jcm@leftraru ~]$ sbatch run-sextractor.slurm ./Blind_03_N1_01.fits.fz_proj.fits
Blind_03_N1_01_wtmap.fits.fz_proj.fits
Submitted batch job 8439444

[jcm@leftraru ~]$ squeue
  JOBID   PARTITION      NAME      USER ST        TIME  NODES NODELIST(REASON)
8439444      levque sextract       jcm  R         0:09      1 levque030

[jcm@leftraru ~]$ cat sextractor.8439444.out
Running at levque030
Starting time : Mon 21 Aug 2017 09:12:19 AM -03
Ending time : Mon 21 Aug 2017 09:12:24 AM -03
Done

[jcm@leftraru ~]$
```

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

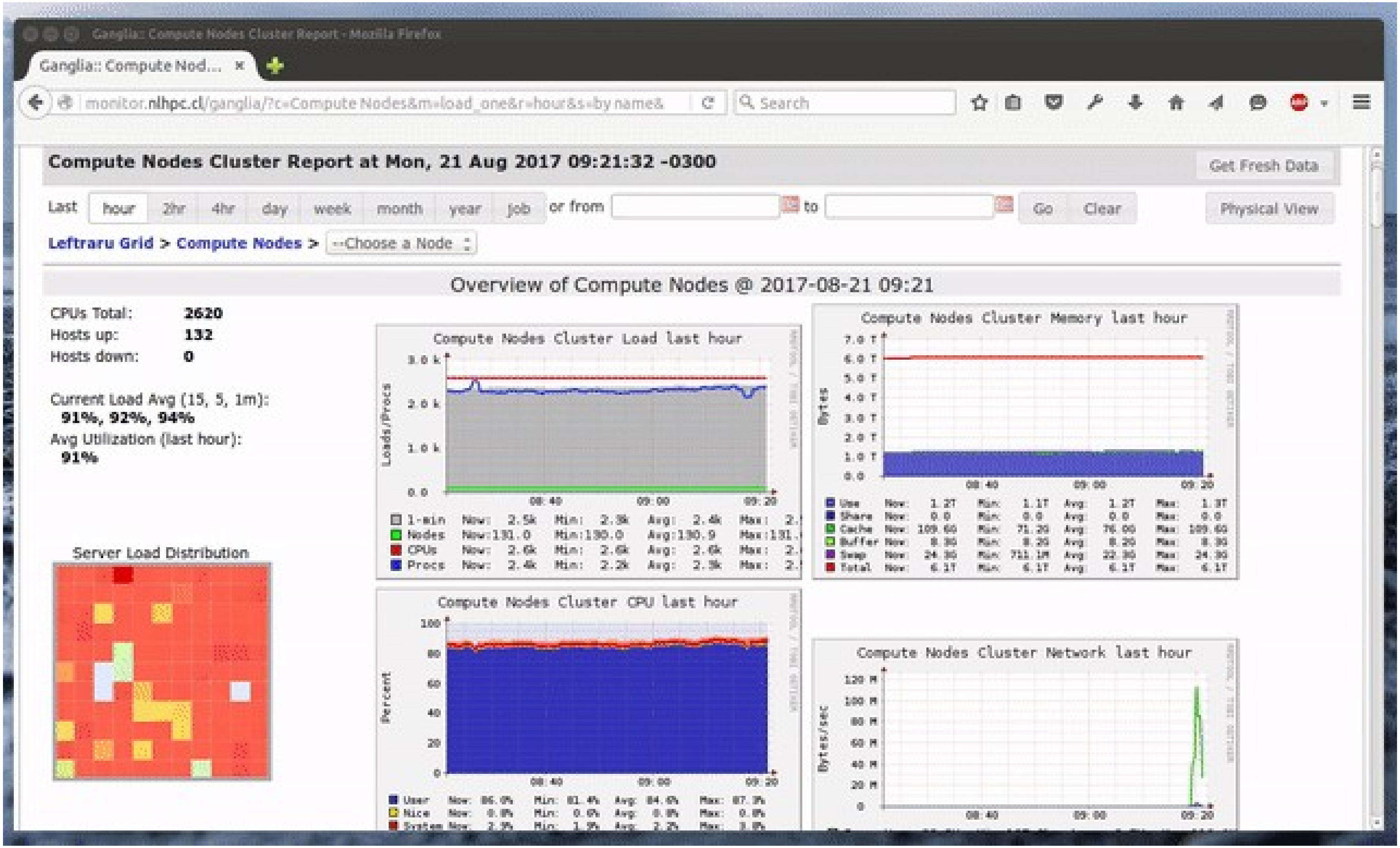
Monitoring Jobs

- **Ganglia** is an open source monitoring system developed in the NPACI (UCLA) and widely used to monitor HPC clusters.

http://monitor.nlhpc.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.

Monitoring Jobs: Ganglia

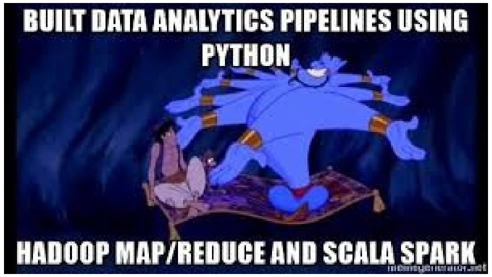


Programming in a HPC system

A crossroad between bash and python

Programming in a HPC System

- Two ways
  - Using Bash (or any other interpreter) scripting
  - Using a high level language
    - Python
    - C/C++ (for bad asses)
    - Java (bad idea!)
    - Or any other language allowing process management
- Programming frameworks
  - Spark
  - Python dispy/pp/multiprocessing
  - Celery
  - Hive
  - Etc (the list is looooooong)



What do you choose: Blue or red?

The Blue Pill: BASH

THE BOURNE-AGAIN SHELL

Pros

- Quick and easy
- Fast development
- Easy to call external programs
- GNU tools available!
- Small orchestration footprint (overhead)
- Direct interaction with the resource manager (queue)

Cons

- Data sharing based on shared file-system
- Limited (but sufficient) process control
- Cryptic orchestration code
- Limited (and costly) parsing abilities
- Limited (in memory) data structures for data indexing

The Red Pill: python

Pros

- Rich language
- Better process management
- Many data structures for data indexing
- Data serialization!!
- Many design patterns
- Great parsing abilities
- Object oriented programming

Cons

- More complex development
- Indirect access to the resource manager (queue)
- Limited thread implementation (only python 2 series)
- Module maintenance
- Higher overhead per process (in memory)
- Intelligent Data sharing may be more complicated than sharing via filesystem

Bash for HPC job scripting

Bash process control (easy fork)

- & : detach execution in background
- wait : wait for a detached process to finish
  - No args: all of them
  - pid arg: wait for job with given pid
- Bash functions cannot be called as commands for tasks and jobs (buuuu!)
- `jobs -p` : list of detached jobs

```
$ cat my-jobste-array.slurm
#!/bin/bash
#SBATCH -J my_jobstep_array
#SBATCH -n 10
#SBATCH -p levque

echo "master Tasks $SLURM_JOB_ID running \
      in `hostname`"

NUM_TASKS=20
for task in `seq 1 $NUM_TASKS`;
do
    srun --exclusive -n 1 -N 1 -p levque \
       ./jobstep.slurm &
done
wait
echo "done"

$ cat jobstep.slurm
#!/bin/bash
echo "Task $SLURM_STEP_ID running \
in host `hostname`"
exit 0

$
```

Bash arguments control (xargs)

- Grouping of arguments
- Evaluate in parallel arguments
- Almost the same functionality than GNU parallel
- Can be used with built in functions

```
$ cat input.file
1
2
3
...
9
10

# group arguments in 4
$ cat input.file | xargs -n 4
1 2 3 4
5 6 7 8
9 10

# print an argumente via 2 child processes
$ cat input.file | xargs -n 1 -P 2 -I {} \
  bash -c 'echo "$@";sleep 1' _ {}
1
2
…
3
4
…

$
```

GNU Toolchain

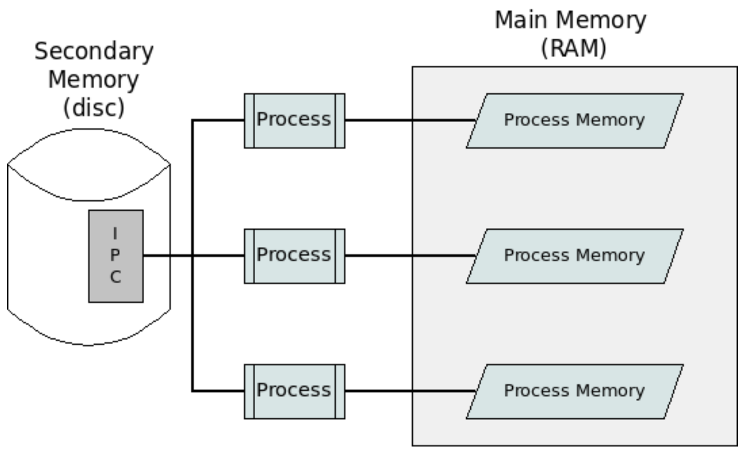
DISCLAIMER

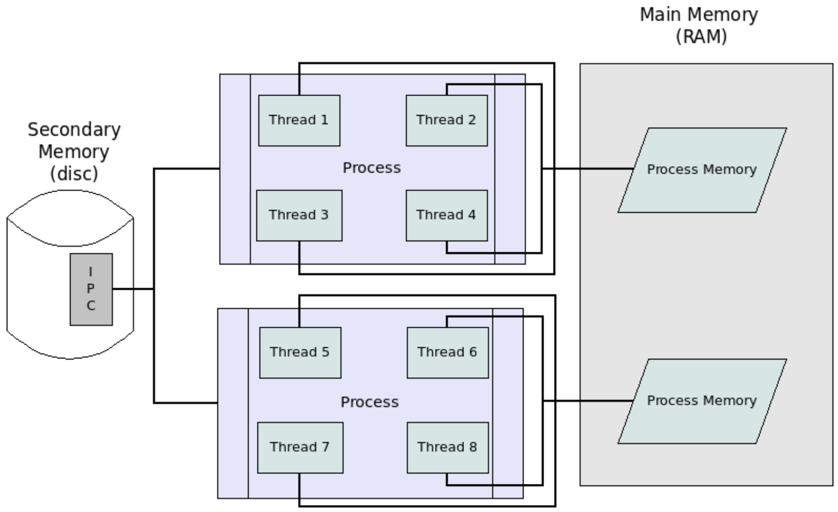
Do not try to compete with GNU tools, they have many years of code maturity and they do their work so efficient that it looks like they use **black magic** to get the job done

- gcc, make, coreutils, binutils, build system (autotools), debugger, bison, m4
- https://en.wikipedia.org/wiki/List_of_GNU_Core_Utilities_commands
- **You can mostly do whatever you need only by combining GNU commands and bash statements in an executable script.**

Python for HPC job scripting

Process / Thread

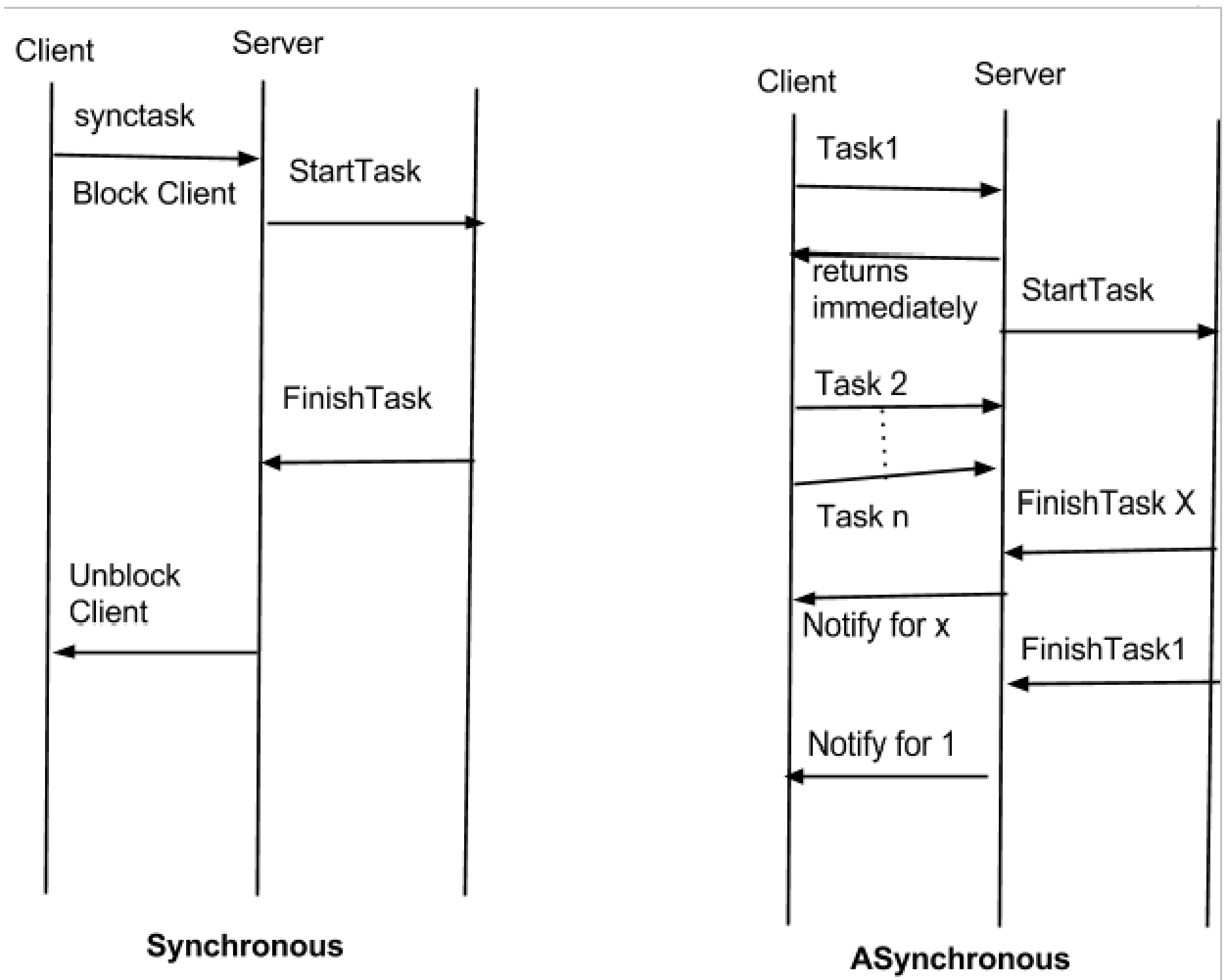




- Heavy independent tasks.
- **Different memory spaces**, file descriptors, stack, etc.
- Single control routine (the main function)
- Each child process **copies the memory space of the father**.
- Different processes uses *Inter Process Communication* for data exchange.
- It does not require a locking mechanism

- Light and cooperative tasks.
- **The same memory space,** file descriptors, stack, etc.
- Multiples execution controls (one per thread)
- Each thread has full **access to the same memory space of the father**.
- They communicate each other directly (via variables)
- It implements a locking mechanism for exclusive memory access.

Synchronous / Asynchronous

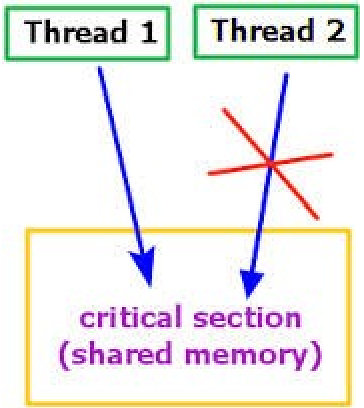


- Blocks the calling thread.
- Easy to determine state of execution
- Hard to (fully) exploit multicore architectures

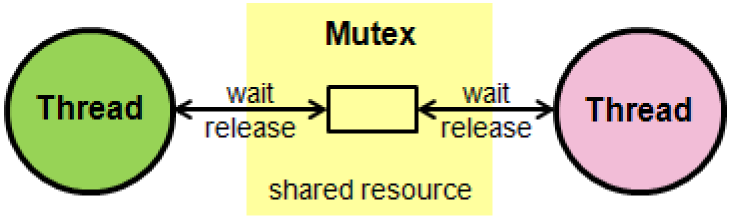
- The calling thread continues its execution.
- Hard to determine state of execution (let's the parallelism begin)
- Lazy Evaluation
- Future / Promise
- Wait / Notify

Locks / Mutex / Semaforos

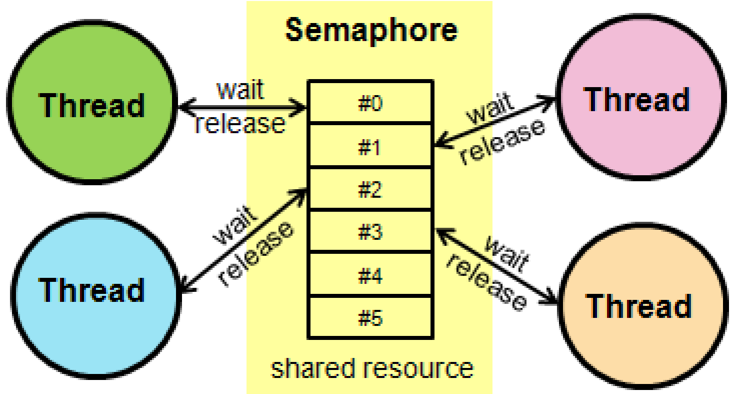
- Concurrency
  - Lock
    (aka critical section).



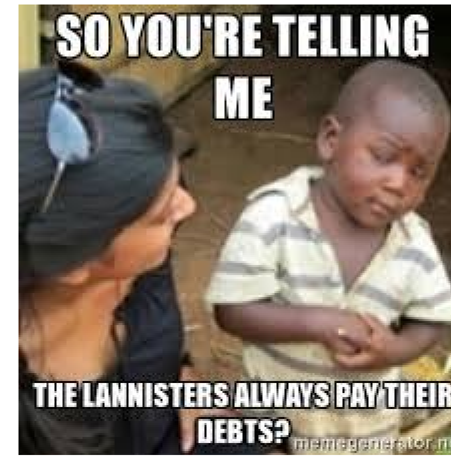
  - Semaphore Mutex
    (aka mutex).



  - Counting semaphore
    (aka semaphore).



Future/Promise



- When you promise to do something in the near future and the time to collect arrives
- Resolve, Reject
- Promises chain: then .. then .. then
- Each promise should run asynchronously
- Lambda functions

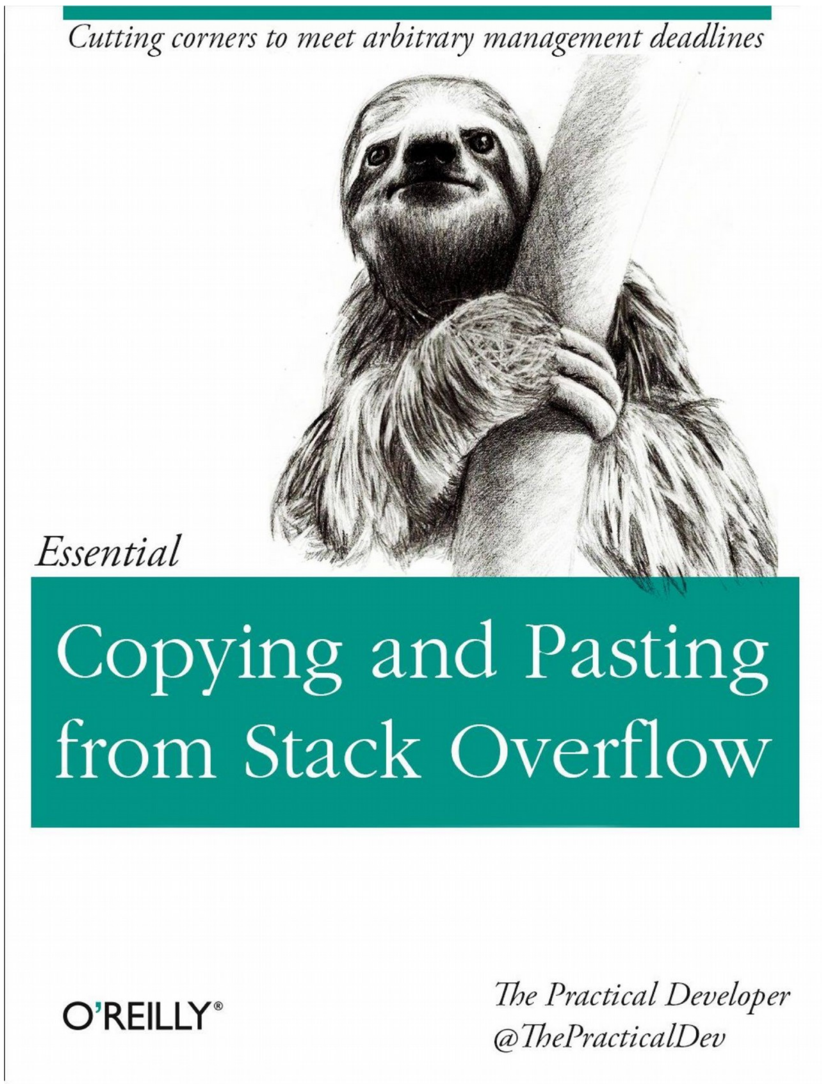
```
x = Promise(do something)
    .then(do another thing)
    .done(you are set)
    .catch(something went wrong)

..
..
..

Result = x.get()
```

Lambda Functions

- An anonymous function that takes a function as an argument and returns a function.
- It can be use as a functional
- It can be use for lazy evaluation.
- Maps, filter, etc, etc

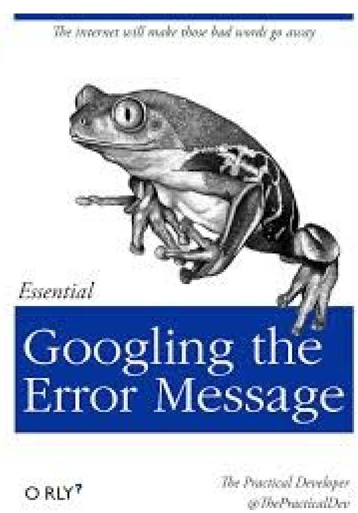

```
> x = lambda x,y: x+y
> print(x(1,2))
3
> f = lambda g: g(x)
> f(3,4)
7
> def p(str)
    print(str)
> i = lambda x : x(“resolved”)
> i(p)
resolved
```

Literature ?



It is hard to be original when searching for a problem in Google

(someone always already did it and there are several good/bad answers)



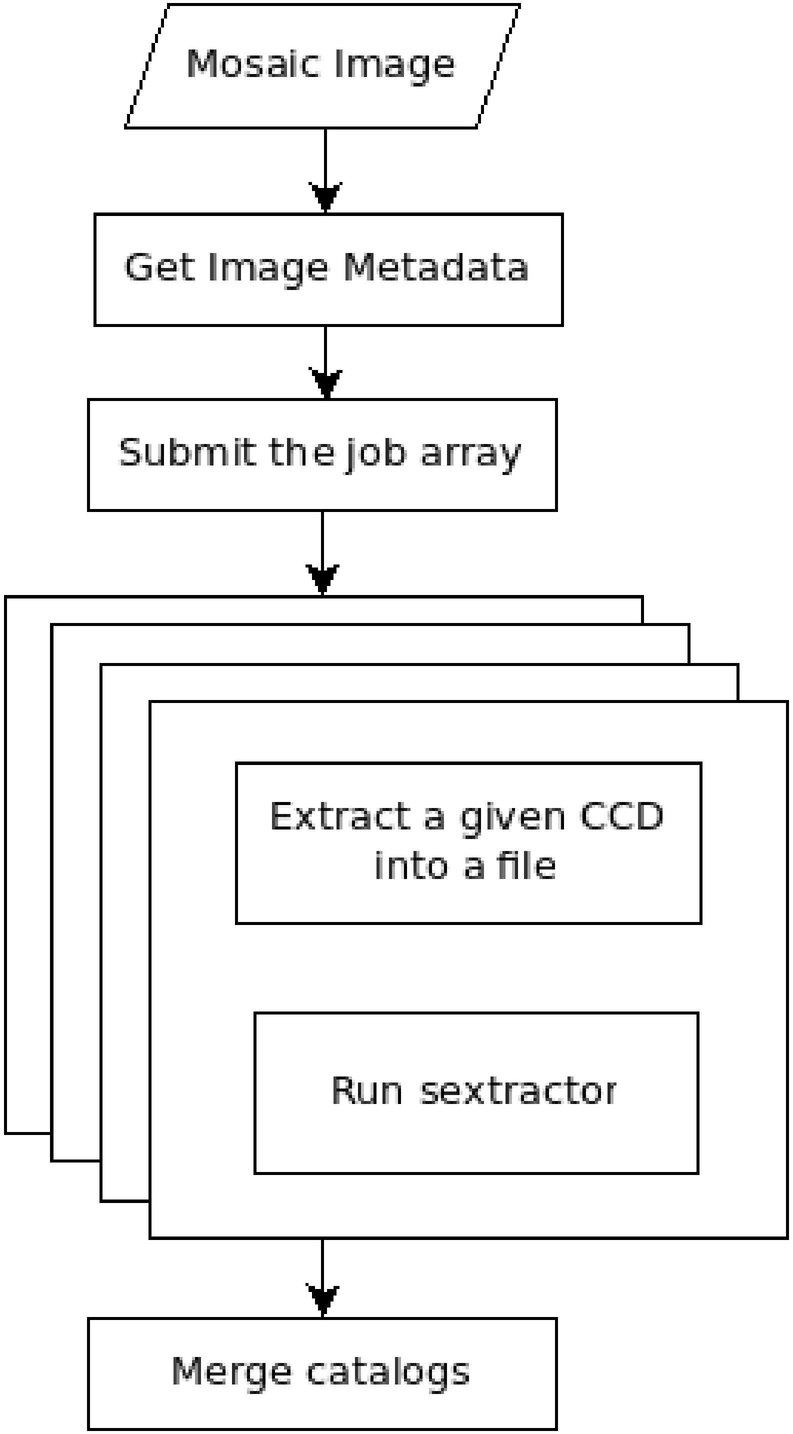
Hands-On

“Source extraction with Spark”

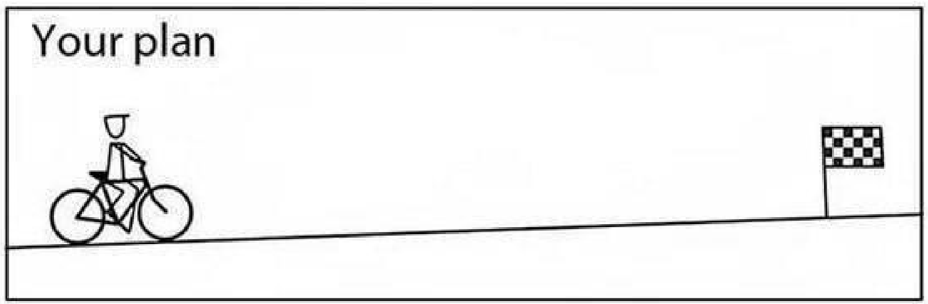
The shorter version due to time….

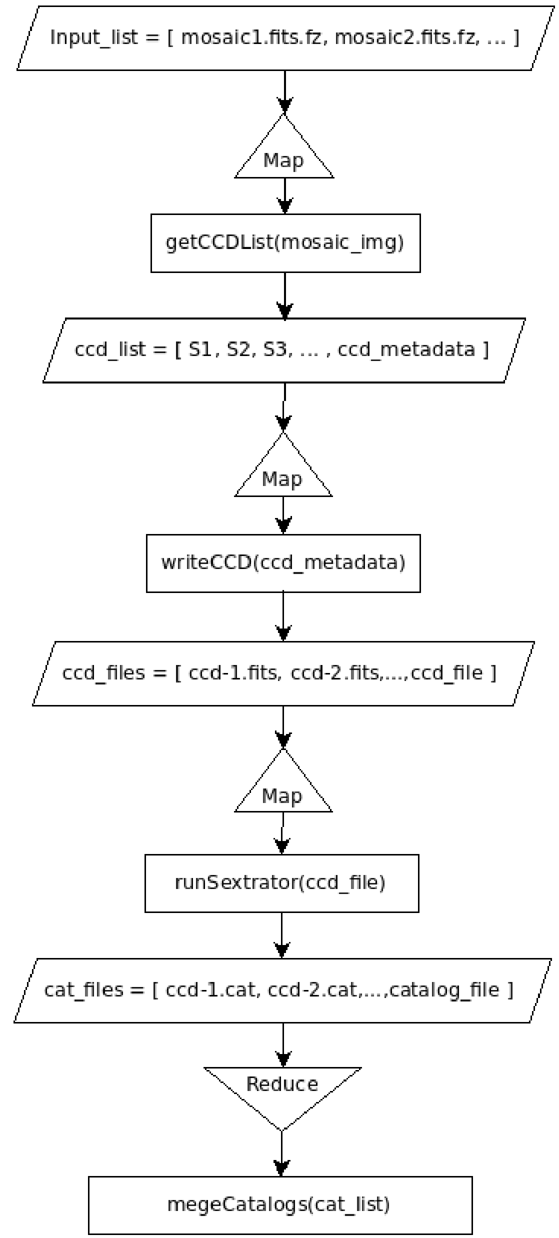
Something challenging enough?

What I would do with classical HPC tools?
Bash ?? python??



Let's draw a plan!






```
ccd = {
    'name' : N1,
    'file': mosaic_1.fits.fz,
    'object' : Blind14A_23,
    'mjd' : 56726.1629225,
    'keys' : [...],
    'header': [...]
}
```

How it looks like?

```
# Distributed Sextractor using Spark
# Simple Example 1
# JcM

from pyspark import SparkContext
import pyfits
import os

def getCCDList(file):
    hdulist = pyfits.open(file)
    prihdr = hdulist[0].header
    num_ccds = prihdr["NEXTEND"]
    hdu_list = [];
    for idx, hdu in enumerate(hdulist):
        name = hdu.name
        keys = hdu.header.ascard
        print idx, name, len(keys)
        if idx != 0:
            hdu_list.append({
                'id':idx,  'file':file, 'name':hdu.name, 'header':keys, 'object':prihdr['OBJECT'],
                'mjd':prihdr['MJD-OBS'], 'key_num': len(keys)})

    hdulist.close()
    return hdu_list

def writeCCD(ccd_handler):
    data = pyfits.getdata(ccd_handler['file'], extname=ccd_handler['name'])
    hdu = pyfits.ImageHDU(data)

    ccd_file = "%s-%s-%s.fits" %(ccd_handler['object'],
             ccd_handler['name'],ccd_handler['mjd'])
    for card in ccd_handler['header']:
        hdu.header.append(card)

    hdu.writeto(ccd_file)
    ccd_handler["ccd_file"] = ccd_file
    return ccd_handler
```

```
def runSextractor(ccd_handler):
    catalog_file="%s.catalog" %(ccd_handler["ccd_file"])
    cmd="sextractor %s -c etc/default.sex -CATALOG_NAME %s"
             %(ccd_handler["ccd_file"],catalog_file)
    os.system(cmd)
    ccd_handler["catalog"] = catalog_file
    return ccd_handler

def mergeCatalogs(cats):
    merged_catalog = "%s.catalog" % (cats[0])

    cmd = "cat "
    for c in cats[1]:
        cmd = "%s %s" %(cmd,c)

    cmd = "%s > %s" %(cmd, merged_catalog)
    os.system(cmd)
    return merged_catalog


######################## MAIN ####################

print "Distributed Sextractor"

sc = SparkContext("local[4]", "Distributed Sextractor")

in_files = [ 'in/tu2208329.fits.fz', 'in/tu2214935.fits.fz', 'in/tu2216725.fits.fz' ]

ccds = sc.parallelize(in_files).flatMap(getCCDList).collect()
fits = sc.parallelize(ccds).map(writeCCD).collect()
cats_per_object = sc.parallelize(fits).map(runSextractor).
             map(lambda o: (o['object'], [ o['catalog'] ])).
             reduceByKey(lambda a,b: a+b ).collect()

cat_list = sc.parallelize(cats_per_object).map(mergeCatalogs).collect()

print cat_list

print "Done"
```

The Take Aways

- Definitions needed to understand a HPC system.
- Overview about architecture and components of a HPC system.
- Software, Applications, tools-chains, scheduler, modules
- Basic concepts for programming in a HPC system.
- Follow an example of how to work with generic HPC system.