Announcements

- Project: TODAY baseline model

- Second midterm: next Thursday November 30.

Introduction to High Performance Computing (HPC)

Guillermo Cabrera Vives
guillecabrera@udec.cl

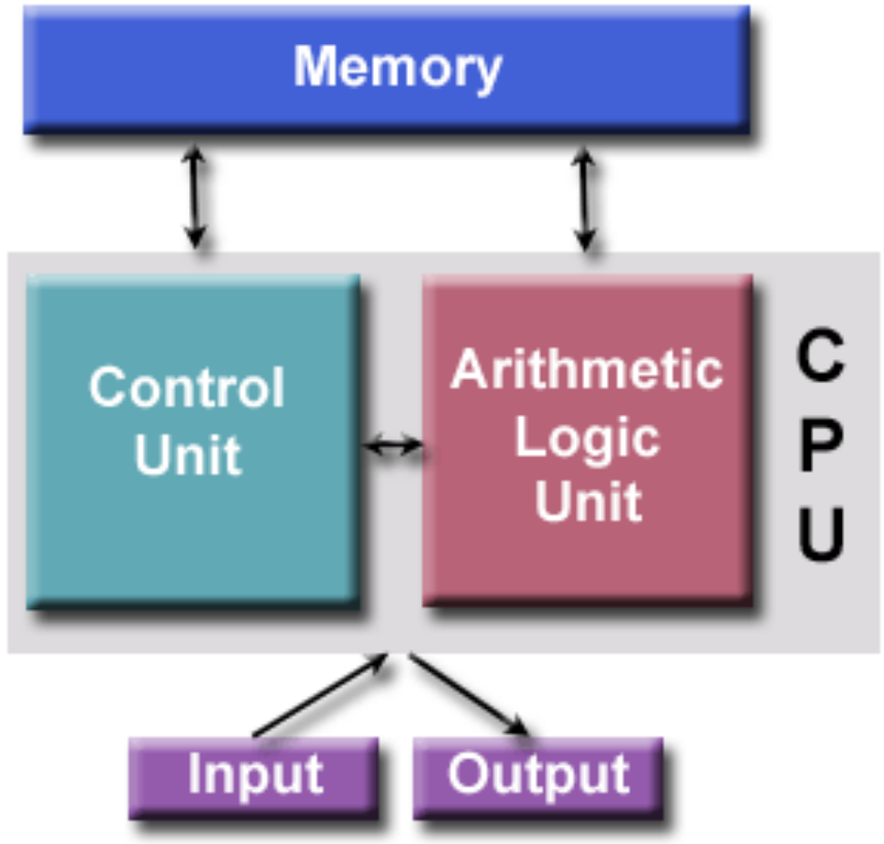
Motivation

- Simulations
- Data analysis
- Optimization

Concepts and Terminology

von Neumann Architecture

- Comprised of four main components:
  - Memory
  - Control Unit
  - Arithmetic Logic Unit
  - Input/Output
- Read/write, random access memory is used to store both program instructions and data
  - Program instructions are coded data which tell the computer to do something
  - Data is simply information to be used by the program



- Control unit fetches instructions/data from memory, decodes the instructions and then ***sequentially*** coordinates operations to accomplish the programmed task.
- Arithmetic Unit performs basic arithmetic operations
- Input/Output is the interface to the human operator

Basic concepts

- **Supercomputing / High Performance Computing (HPC)**

- **Node:** A standalone "computer in a box"

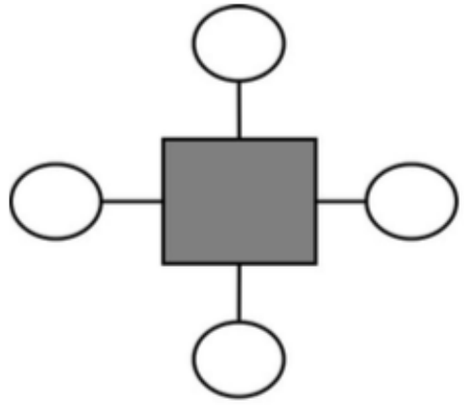
- **CPU / Socket / Processor / Core**

- **Job:** set of instructions and resources to perform a task.

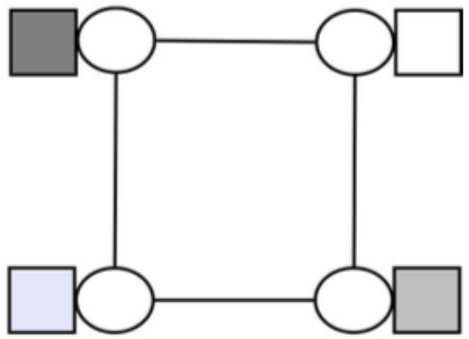
- **Task:** A logically discrete section of computational work.

Basic concepts: memory

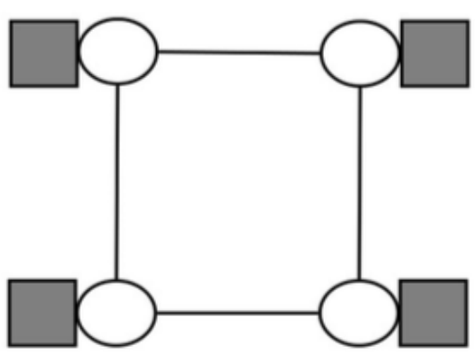
- **Shared Memory:** computer architecture where all processors have direct (usually bus based) access to common physical memory



- **Distributed Memory:** each processor has its own private memory.

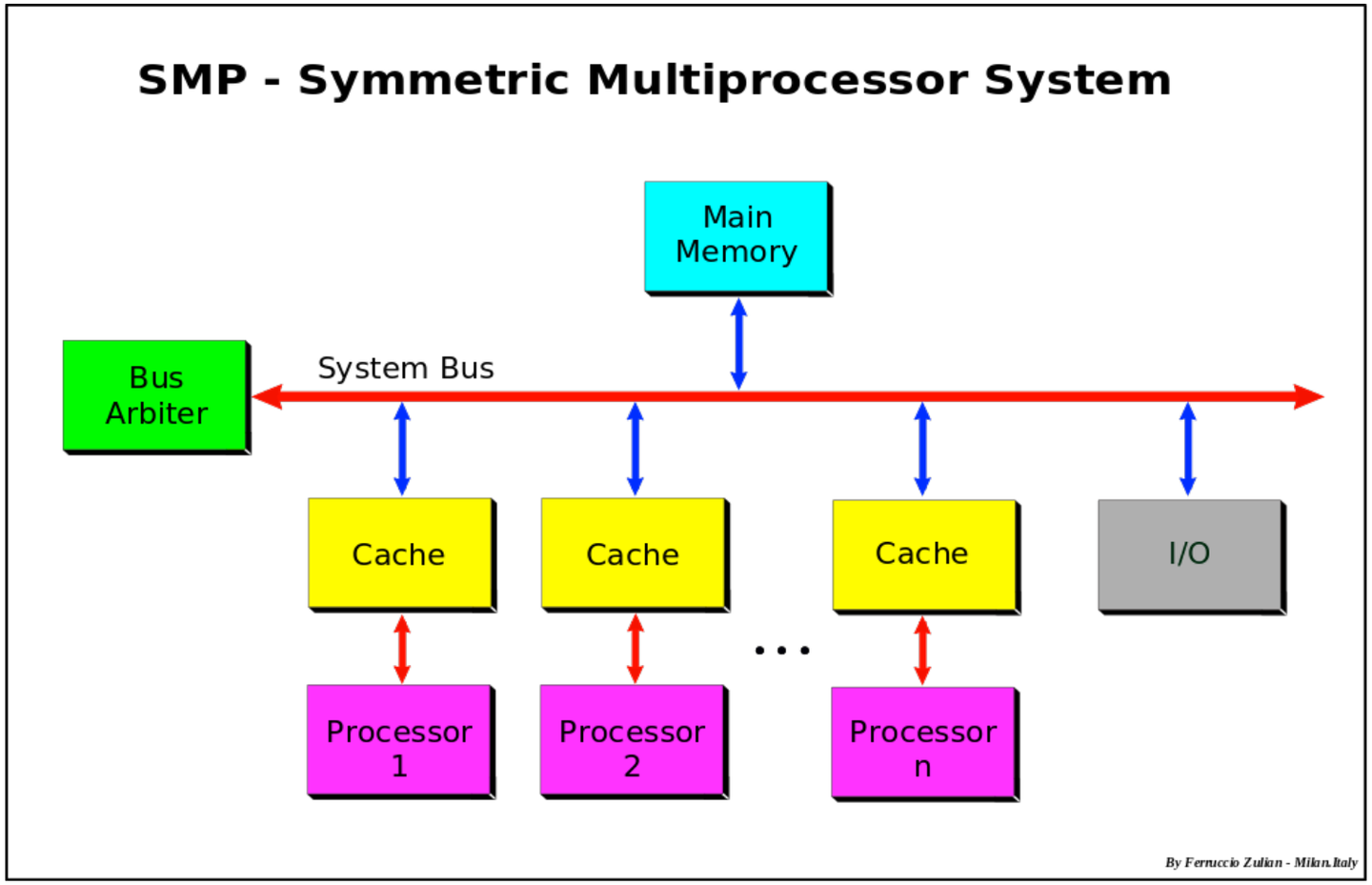


- **Distributed Shared Memory:** physically separated memories can be addressed as one logically shared address space.



Basic concepts

- **Symmetric Multi-Processor (SMP):** two or more identical processors are connected to a single, shared main memory, have full access to all input and output devices, and are controlled by a single operating system instance that treats all processors equally, reserving none for special purposes.



Basic concepts

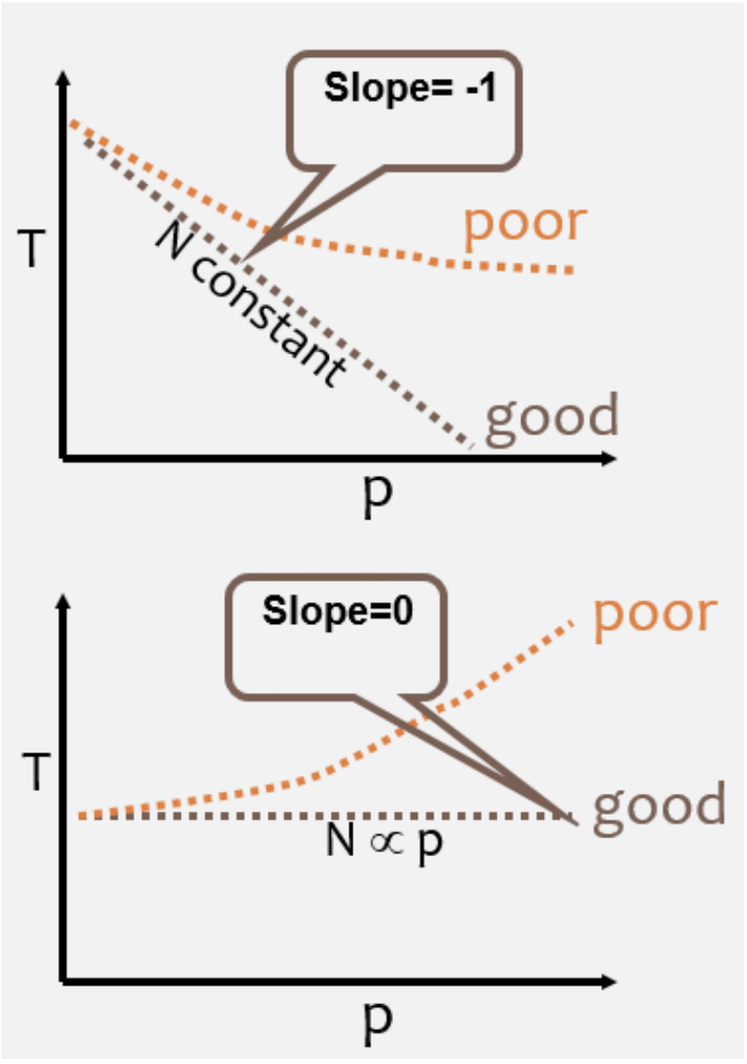
- **Communications:** event of data exchange
- **Synchronization:** coordination of parallel tasks in real time
- **Granularity:** qualitative measure of the ratio of computation to communication.
    - Coarse: relatively large amounts of computational work are done between communication events
    - Fine: relatively small amounts of computational work are done between communication events
- **Parallel Overhead:** amount of time required to coordinate parallel tasks.
- **Embarrassingly Parallel:** solving many similar, but independent tasks simultaneously

Basic concepts

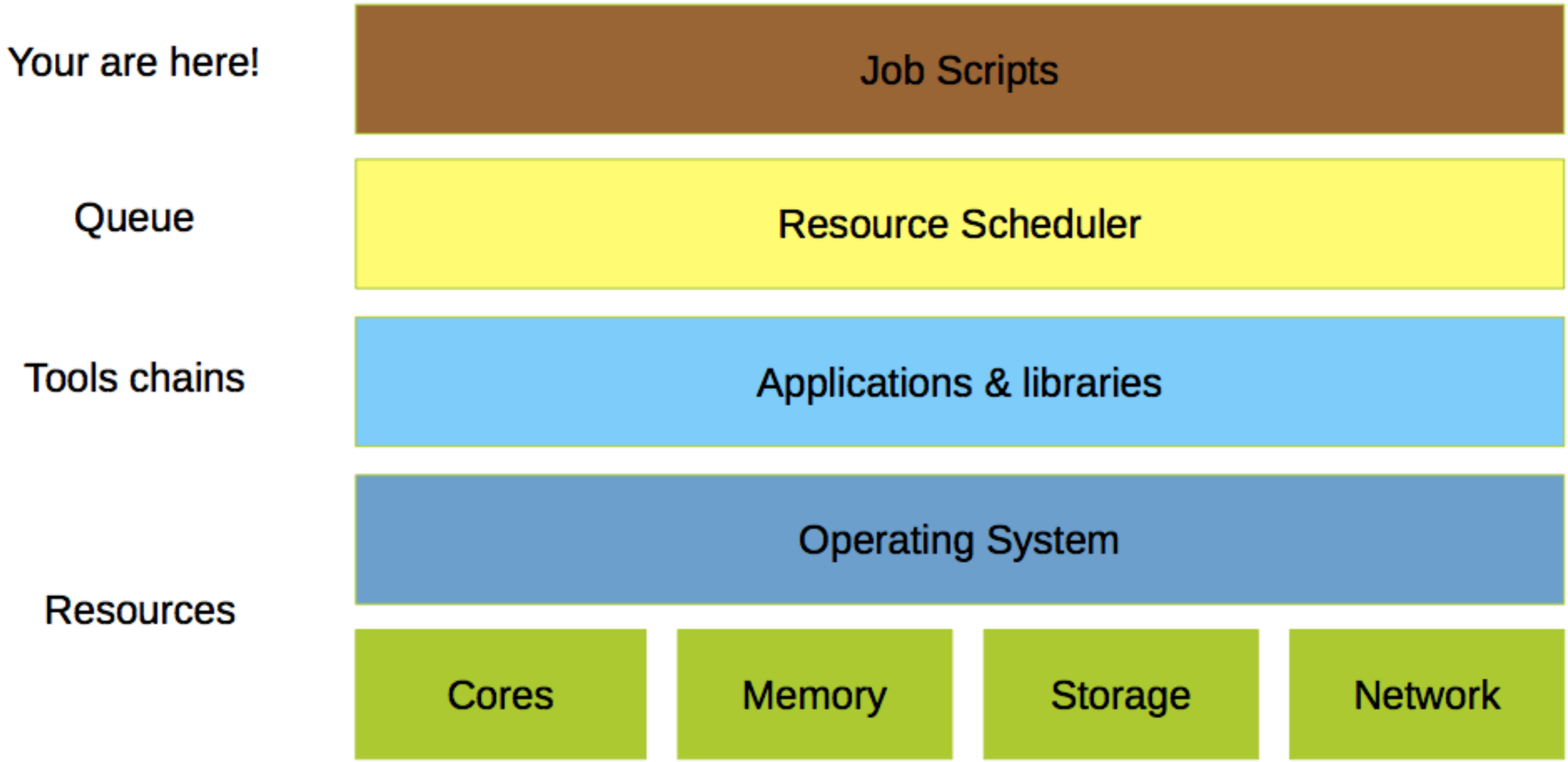
- **Scalability:** parallel system's ability to demonstrate a proportionate increase in parallel speedup with the addition of more resources.

- **Strong scaling:**
  - The **total** problem size stays fixed as more processors are added.
  - Goal is to run the same problem size faster
  - Perfect scaling means problem is solved in 1/P time (compared to serial)

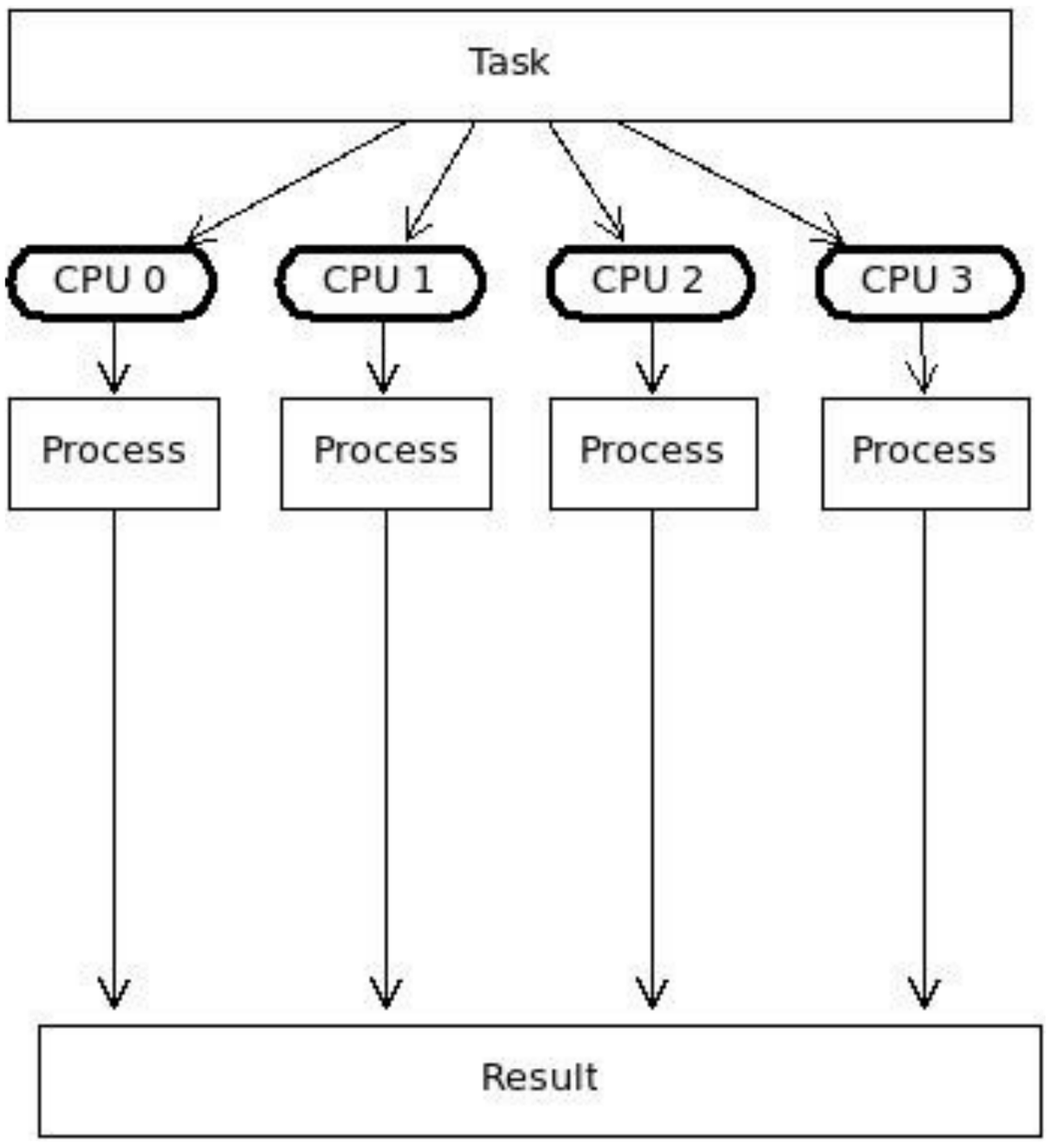
- **Weak scaling:**
  - The problem size **per processor** stays fixed as more processors are added.
  - Goal is to run larger problem in same amount of time
  - Perfect scaling means problem Px runs in same time as single processor run



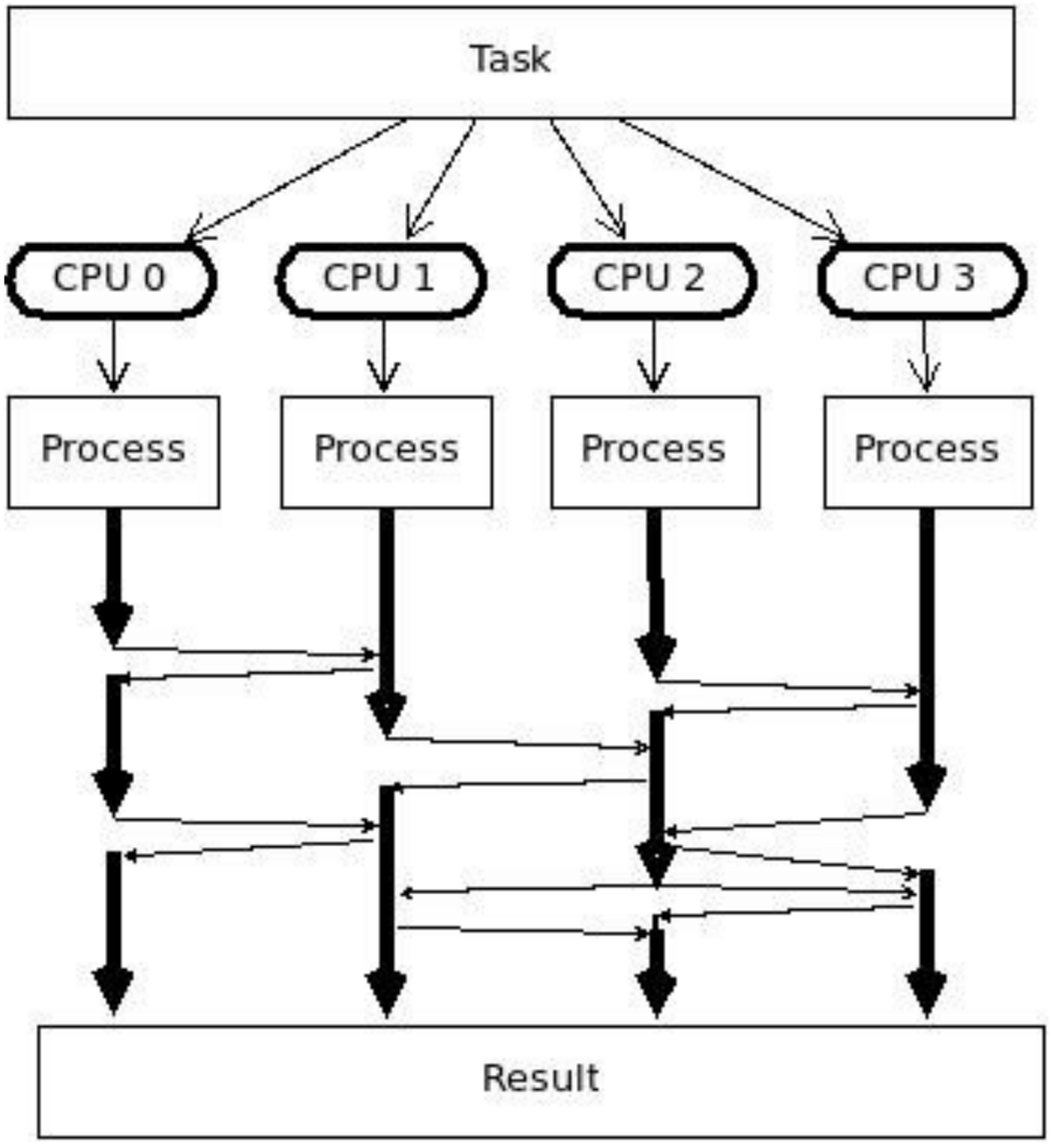
Software Layout



Parallel v/s **Distributed**

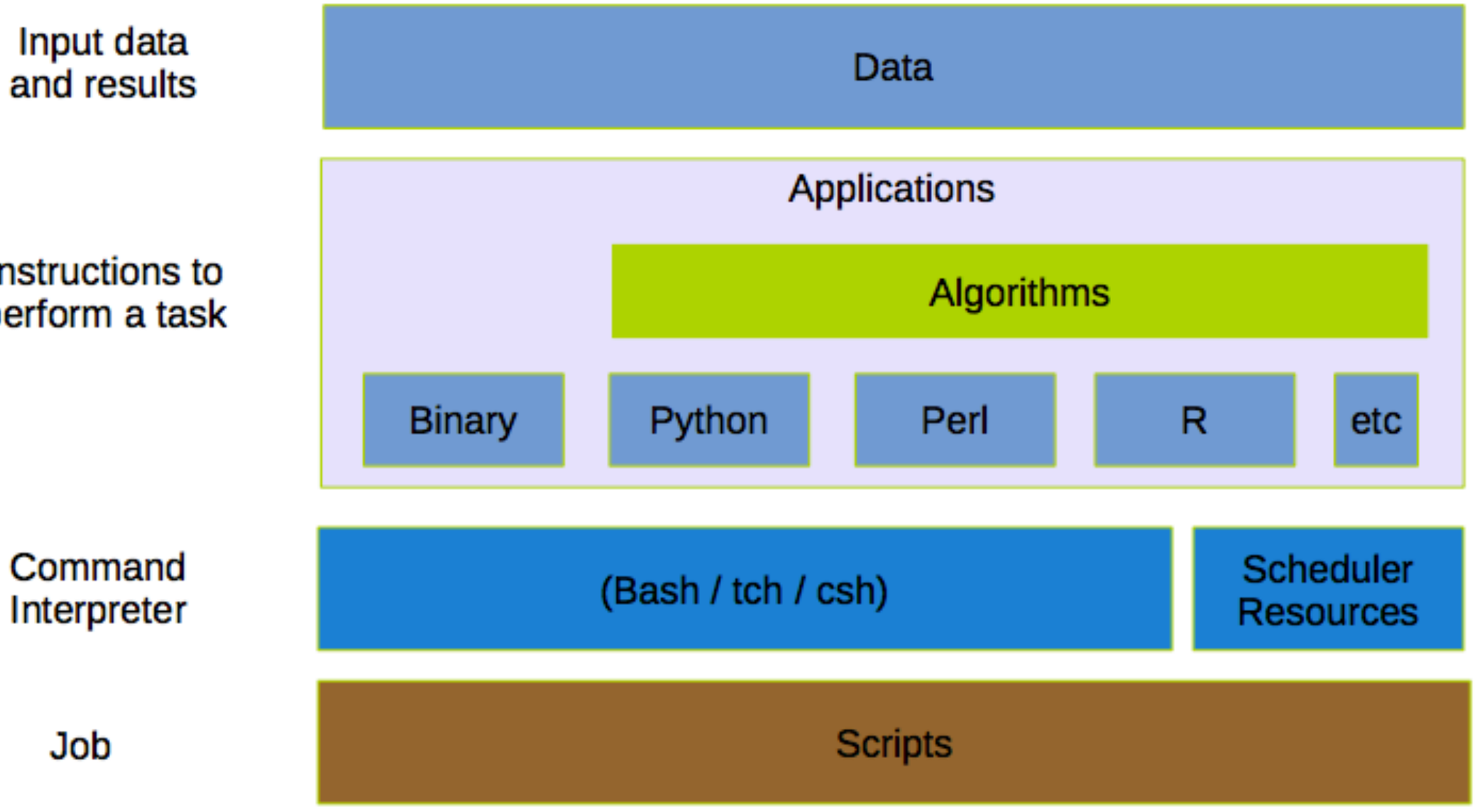


Parallel v/s Distributed



Working with a HPC System

Job Scripting



Scheduler: Sun Grid Engine (SGE)

```
$ cat 01_sleep.sh
#!/bin/bash
#
#$ -cwd
#$ -j y
#$ -S /bin/bash
#
date
sleep 10
date
```

- **-cwd** means to execute the job for the current working directory.
- **-j y** means to merge the standard error stream into the standard output stream instead of having two separate error and output streams.
- **-S /bin/bash** specifies the interpreting shell for this job to be the Bash shell.

Scheduler: Sun Grid Engine (SGE)

- Submitting a job

```
$ qsub 01_sleep.sh
your job 16 ("sleep.sh") has been submitted
```

- Checking jobs

```
$ qstat
job-ID  prior   name       user         state submit/start at     queue                          slots ja-task-ID
-----------------------------------------------------------------------------------------------------------------
   9043 0.50000 Demo01_sle gcabrera     r     11/22/2017 10:59:20 luthiers.q@luthier2                1
```

`qstat -f` to display a more detailed list of jobs within SGE.

`qstat -j` to query the status of a job, given it's job id.

Scheduler: Sun Grid Engine (SGE)

- Deleting a job

```
$ qdel 9043
gcabrera has registered the job 9044 for deletion
```

- Output

```
$ cat Demo01_sleep.sh.o9043
mié nov 22 10:59:20 -03 2017
mié nov 22 10:59:31 -03 2017
```

Array Jobs

- Common problem: you have a large number of jobs to run, and they are largely identical in terms of the command to run.
- Naive solution: generate 1000 shell scripts, and submit them to the queue.
- Not efficient, neither for you nor for the head node.

Array Jobs

ARRAY JOBS:

- You only have to write one shell script
- You don’t have to worry about deleting thousands of shell scripts, etc.
- If you submit an array job, and realize you’ve made a mistake, you only have one job id to qdel, instead of figuring out how to remove 100s of them.
- You put less of a burden on the head node.

Array Jobs

```
#!/bin/sh

# Tell the SGE that this is an array job, with "tasks" to be numbered 1 to 10000
#$ -t 1-10

# Define where to put the output and errors of each job
#$ -o /shared/home/gcabrera/Demos/output/
#$ -e /shared/home/gcabrera/Demos/error/

# When a single command in the array job is sent to a compute node,
# its task number is stored in the variable SGE_TASK_ID,
# so we can use the value of that variable to get the results we want.

echo $SGE_TASK_ID
sleep 10
echo "done"
```

Array Jobs

- Running a program with different input

```
#!/bin/sh

#$ -cwd

# Tell the SGE that this is an array job, with "tasks" to be numbered 1 to 10000
#$ -t 1-10

# Define where to put the output and errors of each job
#$ -o /shared/home/gcabrera/Demos/output/
#$ -e /shared/home/gcabrera/Demos/error/

# When a single command in the array job is sent to a compute node,
# its task number is stored in the variable SGE_TASK_ID,
# so we can use the value of that variable to get the results we want:

python echo.py $SGE_TASK_ID
```

Array Jobs

- Running a program with different input files

```
#!/bin/bash

#$ -cwd
#$ -S /bin/bash

# Tell the SGE that this is an array job, with "tasks" to be numbered 1 to 10000
#$ -t 1-10

# Define where to put the output and errors of each job
#$ -o /shared/home/gcabrera/Demos/output/
#$ -e /shared/home/gcabrera/Demos/error/

# When a single command in the array job is sent to a compute node,
# its task number is stored in the variable SGE_TASK_ID,
# so we can use the value of that variable to get the results we want:

FILE=/shared/home/gcabrera/Demos/input/test$SGE_TASK_ID.txt
echo $FILE

if [ -f $FILE ];then
    cat $FILE
else
    echo file $FILE not found
fi
```

References

- https://computing.llnl.gov/tutorials/parallel_comp/
- http://www.rocksclusters.org/roll-documentation/sge/4.2.1/submitting-batch-jobs.html
- http://wiki.gridengine.info/wiki/index.php/Simple-Job-Array-Howto

Next Semester

- Distributed and Parallel Computing, Cecilia Hernández
- Data Science 2, Guillermo Cabrera Vives
- Topics in Artificial Intelligence, Julio Godoy

QUIZ TIME!