Collaborative
EUDAT Data Infrastructure

PRACE

Intro to Python and HPC batch scheduler

Giuseppe Fiameni / CINECA

Trieste 23rd September 2019

AGENDA

- Set up Jupyter Notebook on Galileo System
- Python Basics
- HPC Job Submission via SLURM

What are notebooks?

- A notebook combines the functionality of
  - a word processor – handles formatted text
  - a "shell" or "kernel" – executes statements in a programming language and includes output inline
  - a rendering engine – renders HTML in addition to plain text

GALILEO

Model: IBM NeXtScale
Architecture: Linux Infiniband Cluster
Processors: 16-cores Intel Broadwell 2.30 GHz (2 per node)
Number of Nodes: 360
Internal Network: Infiniband
Accelerators: 2 NVIDIA Tesla K40 on 40 nodes (80 in total)
RAM: 128 GB/node, 8 GB/core
OS: RedHat CentOS release 7.0, 64 bit

GALILEO is a smaller cluster that will be your «home» during this week. It is equipped with accelerators for deep learning applications.

Set up Jupyter Environment

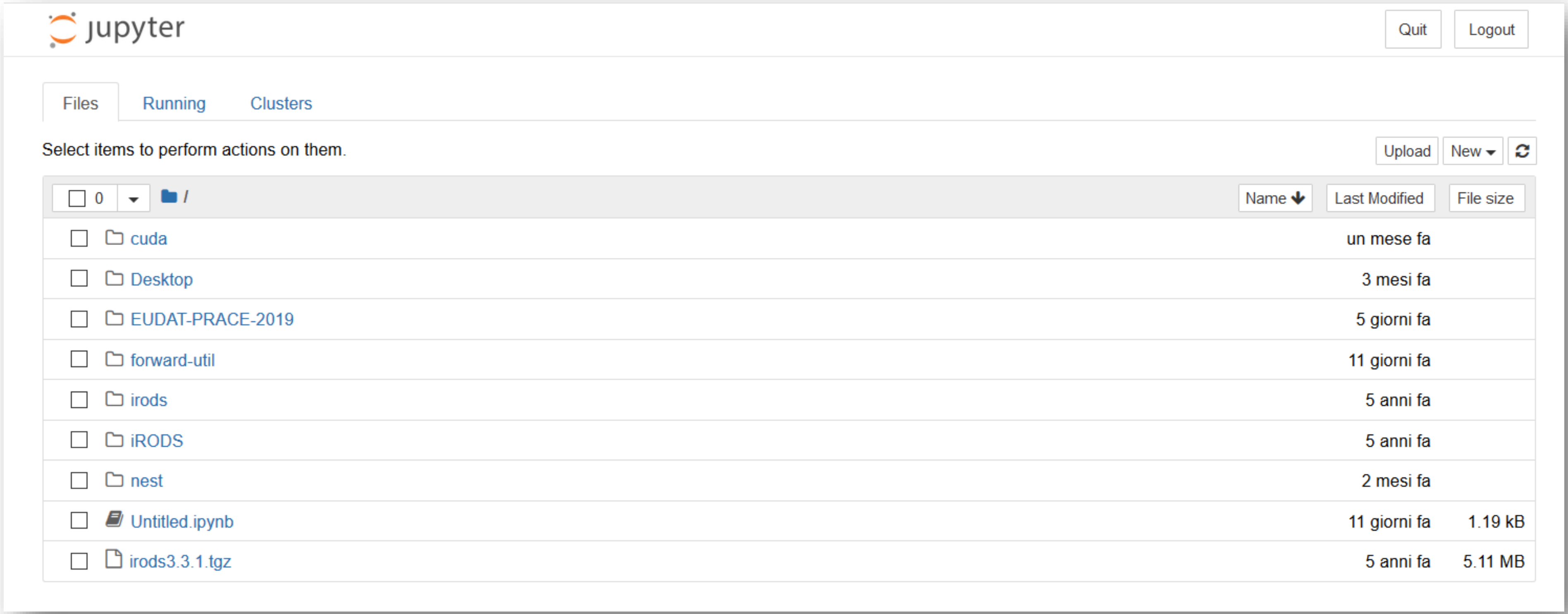
- Pick up a personal test account
  - Password: **pitipmT3P**
- **Try to login:**
  - **`$ ssh`** `a08trbXX@login.galileo.cineca.it`
- Account
  - `train_ceudat19`

Set up Jupyter Environment

- Create SSH Keys Pair (on your own workstation)
  - `$ **ssk-keygen** -t rsa # you can leave the passphrase empty`
    - `.ssh/id_rsa`
    - `.ssh/id_rsa.pub`
  - `$ **ssh-copy-id** -i ~/.ssh/id_rsa.pub a08trbXX@login.galileo.cineca.it`
  - `$ **ssh** a08trbXX@login.galileo.cineca.it`

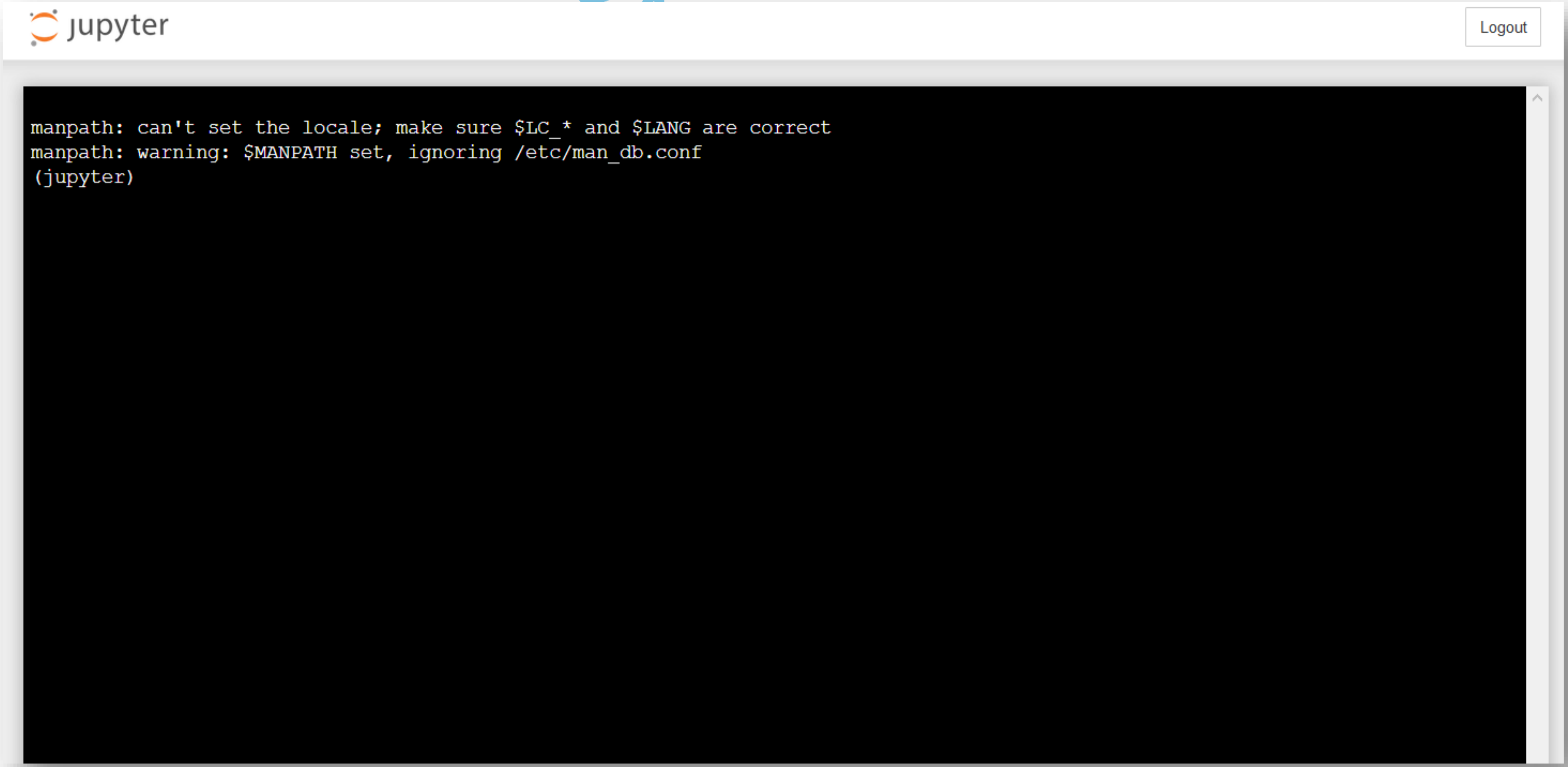
Set up Jupyter Environment

- `$ git clone https://gitlab.eudat.eu/eudat-prace-2019/forward.git`
- `$ bash hosts/galileo_ssh.sh >> ~/.ssh/config`
- `$ bash setup.sh`
- `$ bash start.sh jupyter-pip`



Select items to perform actions on them.

	Last Modified	File size
cuda	un mese fa	
Desktop	3 mesi fa	
EUDAT-PRACE-2019	5 giorni fa	
forward-util	11 giorni fa	
irods	5 anni fa	
iRODS	5 anni fa	
nest	2 mesi fa	
Untitled.ipynb	11 giorni fa	1.19 kB
irods3.3.1.tgz	5 anni fa	5.11 MB



```
manpath: can't set the locale; make sure $LC_* and $LANG are correct
manpath: warning: $MANPATH set, ignoring /etc/man_db.conf
(jupyter)
```

Python Notebook

- $ **git clone** https://gitlab.eudat.eu/eudat-prace-2019/intro-to-python.git

Why Python?

- Python is a clear and powerful object-oriented programming language:
- Easy to learn
  - It is very simple to learn the basis of the language
- Easy to read
  - Elegant syntax similar to pseudo-code
- Easy to use
  - It is simple to get your program working
  - Ideal for prototype development
- Large standard library
  - And easy to extend by adding new modules
- Open source software

Python is slow

- Python is slow compared with other compiled languages already used in computational science.
- So, why Python is becoming so popular in computational science?
  - It is definitely easy to learn and use
    - Usually scientists are not expert programmers
  - You can start to practice using it like a scripting language
    - Writing scripts for pre or post-process your data
  - You can accelerate Python
    - Using scientific modules (e.g. numpy)
    - Using Python combined with other languages

Python and HPC

How you can use Python for HPC?

- Accelerating it
  - Numpy
  - F2py
  - Cython
  - Numba
  - CUDA Python
- Creating ad hoc work-flows
  - High-throughput computing (HTC)
  - Fault tolerance

Python language syntax

Objects and types

Python is strongly typed and dynamically typed

```
>>> a = 4000
>>> b = a
>>> a = 4000.5
>>> type(a) # Everything has a type
```

Operator “=” means a reference to a space in memory that contains an object

```
>>> id(a)
```

Objects are *mutable* (once created can be changed or updated) or *immutable*

Strings

Strings can be created using quotes (single, double or triple)

```
>>> a = 'home'
>>> b = “new home”
```

Triple quotes are used for string that span over more than a single line

```
>>> '''This is the first line
... this is the second line'''
```

Escape characters are similar to C (\n \t)

Strings (part 2)

Multiple actions on strings

```
>>> a = 'my new home'
>>> a.upper()
>>> a.split()
```

Single elements of strings can be accessed

```
>>> a[2]
>>> a[0:2] # python index starts from 0
>>> a[-4:] # no values means beginning or
end
```

Concatenation of strings

```
>>> a + “ is beautiful”
>>> a * 3
```

Containers

List (mutable)

```
>>> a = [1, 1, 2, 'home']
```

Tuple (immutable)

```
>>> a = (1, 4, 'seven', 6)
```

Dict (mutable)

```
>>> a = {'a': 2, 'b':4, 4:5}
```

Set (mutable)

```
>>> a = set([1, 1, 3, 5])
```

Lists

Can be not homogeneous

```
>>> a = [1, 1, 2, 'home']
```

Index ranges from 0 to len(list)-1

Slicing

```
>>> a[0:3] # from first to third element [i:j:k]  k = stride
>>> a[-1:]+a[:-1] # ['home', 1, 1, 2]
```

Mutable (in-place)

```
>>> a[0] = 4  # [4, 1, 2, 'home']
```

Lists

append

```
>>> a = [1, 1, 2, 'home']
>>> a.append(3) # [1, 1, 2, 'home', 3]
```

pop

```
>>> a.pop()    # remove rightmost element
3
```

Function “range” can be used to create list of integers

```
>>> a = list(range(3)) # [0, 1, 2]
>>> b = list(range(2, 10, 3)) # [2, 5, 8]
           # first, last (excluded), step
```

Dictionaries

Map keys to values (mappings)

```
>>> a = {'b':2, 'c': 3}   # 'b', 'c' are keys
                          # 2, 3 are values
>>> a['b']     # returns 2
```

There is no left to right order, only mapping

```
>>> a[-1]     # does not work
```

```
a.keys(), a.values(), a.items()
```

Indentation matters

```
>>> if a > 3:   # mind the colon
...     print a
...     print 'still in the if statement'
... elif a == 3:
...     print 'a is equal to 3'
... else:
...     print 'a is less than 3'
...
>>>
```

For loop

Any sequence object is iterable

```
>>> for i in range(5):
...     print(i)   #  prints 0, 1, 2, 3, 4
```

More common in python

```
>>> a = [1, 1, 4, 'home']
>>> for i in a:
...     print(i)   #  prints 1, 1, 4, 'home'
```

```
break      # exit from inner loop
continue   # go to next iteration
```

While loop

If you don’t know the number of the step of the loop

```
>>> while error > tollerance:
...     result, error = compute(result)
...
>>>
```

Sometime you want to perform an infinite loop end exit only after a check

```
>>> while True:
...     result, error = compute(result)
...     if error < tollerance: break
...
>>>
```

While loop

What happens if the computation doesn't reach the convergence?

It's better to add a safe exit strategy...

```
>>> while c < max_steps:
...     result, error = compute(result)
...     if error < tollerance: break
...     c += 1
...
>>>
```

Boolean conversion

Built-in types can be converted in bool, i.e. they can be used as condition expressions

```
int 0                # False
int != 0             # True
float 0.0            # False
float != 0.0         # True
empty string “”      # False
empty sequence       # False
```

stack: lists and while loop

It is possible to use a list as a stack and pop-out objects until the stack is empty.

```
>>> stack = [obj1, obj2, obj3, obj4, obj5]
>>> while stack:
...     obj = stack.pop()
...     do_some_computation(obj)
...
>>>
```

Collaborative
EUDAT Data Infrastructure

PRACE

File I/O

Old style

DEPRECATED!

```
>>> f = open('filename.txt', 'r')
>>> lst = f.readlines()
>>> f.close()
```

New style (stronger):

```
>>> with open('filenam.txt', 'w') as f:
...      f.write('some string\n')
```

Iterating on file:

```
>>> for line in f:
...      a_list.append(line.strip())
```

The with statement automatically takes care of closing the file once it leaves the with block, even in cases of error.

Function definition

```
>>> def mysum(a, b):
...       "A description of the function."
...       return a + b
...
```

Function call

```
>>> mysum(4, 6)
10
>>> mysum('4', '6')
'46'
```

modules

A module is a file containing Python definitions and statements.

The file name is the module name with the suffix .py appended.

Within a module, the module's name (as a string) is available as the value of the global variable __name__.

How to use a module

```
$ cat fibo.py
def fib(n):
    “Return Fibonacci series up to n.”
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result

>>> import fibo
>>> fibo.fib(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Command line arguments

```
$ cat myprog.py
import sys
if len (sys.argv) < 2 :
    print "Usage: python {} <args>".format(sys.argv[0])
    sys.exit(1)
for x in sys.argv[1:]:
     print("Argument: ", x)


$ python myprog.py arg1 arg2
Argument: arg1
Argument: arg2

$ python myprog.py
Usage: python myprog.py <args>
```

subprocess

```
>>> from subprocess import Popen
```

The Popen constructor execute a child program in a new process.

```
>>> command_line = "executable -i inp.txt -o
out.txt"
>>> p = Popen(command_line.split(),
stdout=file_obj)
```

p.poll(), p.wait(), p.communicate(), p.kill(), p.pid

HPC Job Submission via SLURM

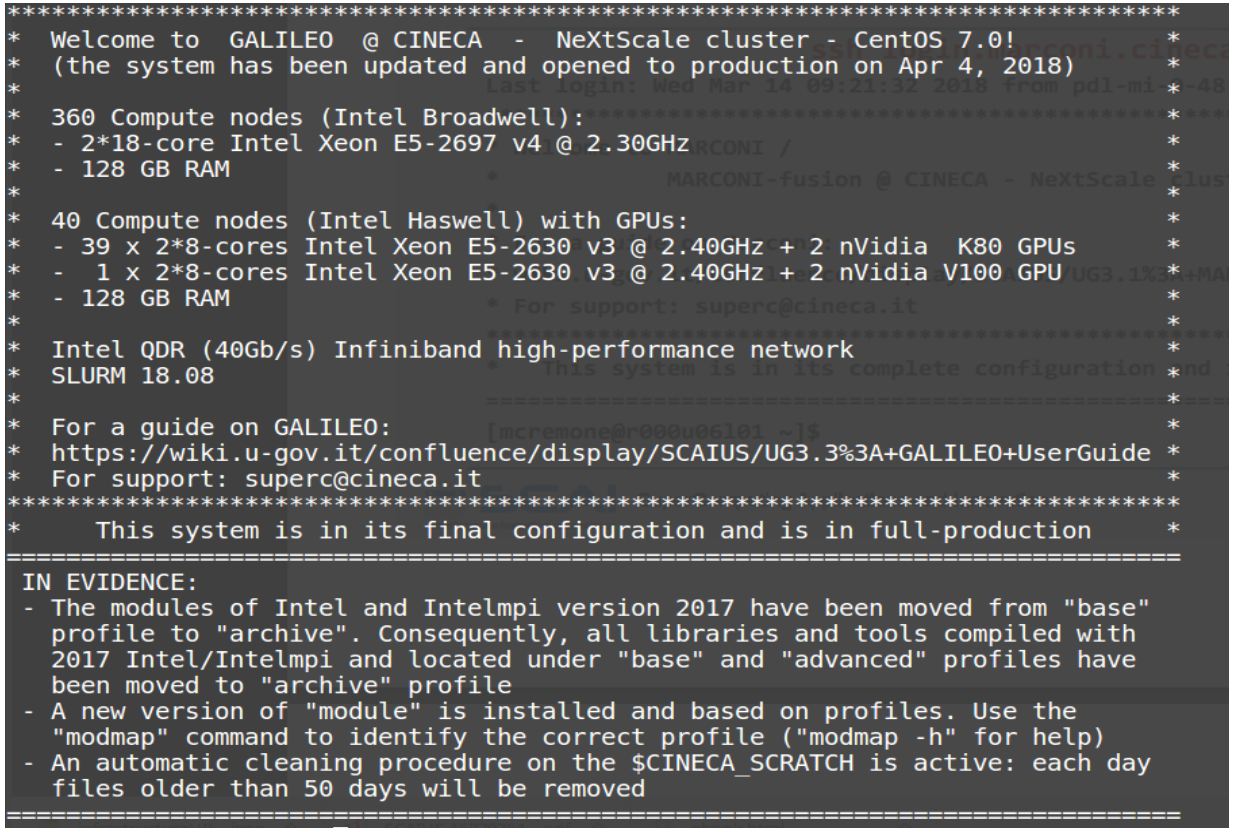
GALILEO

Model: IBM NeXtScale
Architecture: Linux Infiniband Cluster
Processors: 16-cores Intel Broadwell 2.30 GHz (2 per node)
Number of Nodes: 360
Internal Network: Infiniband
Accelerators: 4 nVIDIA Tesla K40 on 40 nodes (160 in total)
RAM: 128 GB/node, 8 GB/core
OS: RedHat CentOS release 7.0, 64 bit

GALILEO is a smaller cluster that will be your «home» during this week. It is equipped with accelerators, especially GPUS

How to login

- `$ ssh a08trbXX@login.galileo.cineca.it`



```
******************************************************************************
*  Welcome to  GALILEO  @ CINECA  -  NeXtScale cluster - CentOS 7.0!          *
*  (the system has been updated and opened to production on Apr 4, 2018)      *
*                                                                             *
*  360 Compute nodes (Intel Broadwell):                                       *
*  - 2*18-core Intel Xeon E5-2697 v4 @ 2.30GHz                                *
*  - 128 GB RAM                                                               *
*                                                                             *
*  40 Compute nodes (Intel Haswell) with GPUs:                                *
*  - 39 x 2*8-cores Intel Xeon E5-2630 v3 @ 2.40GHz + 2 nVidia  K80 GPUs      *
*  -  1 x 2*8-cores Intel Xeon E5-2630 v3 @ 2.40GHz + 2 nVidia V100 GPU       *
*  - 128 GB RAM                                                               *
*                                                                             *
*  Intel QDR (40Gb/s) Infiniband high-performance network                     *
*  SLURM 18.08                                                                *
*                                                                             *
*  For a guide on GALILEO:                                                    *
*  https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.3%3A+GALILEO+UserGuide *
*  For support: superc@cineca.it                                              *
******************************************************************************
*     This system is in its final configuration and is in full-production     *
==============================================================================
 IN EVIDENCE:
 - The modules of Intel and Intelmpi version 2017 have been moved from "base"
   profile to "archive". Consequently, all libraries and tools compiled with
   2017 Intel/Intelmpi and located under "base" and "advanced" profiles have
   been moved to "archive" profile
 - A new version of "module" is installed and based on profiles. Use the
   "modmap" command to identify the correct profile ("modmap -h" for help)
 - An automatic cleaning procedure on the $CINECA_SCRATCH is active: each day
   files older than 50 days will be removed
==============================================================================
```

Python Notebook

- $ **git clone** https://gitlab.eudat.eu/eudat-prace-2019/intro-to-slurm.git

Working environment

- **$HOME:**
  - Permanent, backed-up, and local to GALILEO. 50 Gb of quota. For source code or important input files.
- **$CINECA_SCRATCH:**
  - Large, parallel filesystem (GPFS).
  - No quota. Run your simulations and calculations here. A cleaning policy will delete all your files older than 40 days.
- **$WORK:**
  - Similar to $CINECA_SCRATCH, but the content is shared among all the users of the same account.
  - 1 TB of quota (no cleaning policy). Stick to $CINECA_SCRATCH for the school exercises!

Job scheduler

- As every HPC system, GALILEO allows you to run your applications by submitting “jobs” to the compute nodes
- Your job is then taken in consideration by a scheduler, that adds it to a queuing line and allows its execution when the resources required are available
- The operative scheduler in GALILEO is SLURM
- SLURM stands for "Simple Linux Utility for Resource Management"
  - Allocating access to resources
  - Job starting, executing and monitoring
  - Queue of pending jobs management

SLURM job script schema

- The schema of a SLURM job script is as follows:

```
#!/bin/bash
#SLURM directives

variables environment
execution line
```

Jobscript example

```
#!/bin/bash
#SBATCH --job-name=myname
#SBATCH --output=job.out
#SBATCH --error=job.err
#SBATCH --mail-type=ALL
#SBATCH --mail-user=user@email.com
#SBATCH --time=00:30:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=36
#SBATCH --mem=10GB
#SBATCH --partition=gll_usr_prod
#SBATCH --account=s_tra_eudat

echo “I’m working on GALILEO!”
```

SLURM directives - 1

- `#SBATCH --job-name=myname, -J myname`
  - Defines the name of your job
- `#SBATCH --output=job.out, -o job.out`
  - Specifies the file where the standard output is directed (default=slurm-<Pid>)
- `#SBATCH --error=job.err, -e job.err`
  - Specifies the file where the standard error is directed (default=slurm-<Pid>)
- `#SBATCH --mail-type=ALL (optional)`
  - Specifies e-mail notification. An e-mail will be sent to you when something happens to your job, according to the keywords you specified (NONE, BEGIN, END, FAIL, REQUEUE, ALL)
- `#SBATCH --mail-user=user@email.com (optional)`
  - Specifies the e-mail address for the keyword above

SLURM directives - 2

- `#SBATCH --time=00:30:00, -t 00:30:00`
  - Specifies the maximum duration of the job. The maximum time allowed depends on the partition used
- `#SBATCH --nodes=1, -N 1`
- `#SBATCH --ntasks-per-node=36`
- `#SBATCH --mem=10GB`
  - Specify the resources needed for the simulation.
    - nodes – number of compute nodes (“chunks”)
    - ntasks-per-node – number of cpus per node (max. 36)
    - mem – memory allocated for each node (default=3000MB, max=118000 MB)
- `#SBATCH --partition=gll_usr_prod, -p gll_usr_prod`
  - Specifies the “partition”, a.k.a. the specific set of nodes among which your job can search for resources.

Accounting system

- `#SBATCH --account=s_tra_eudat, -A s_tra_eudat`
  - Specifies the account to use the CPU hours from.

As an user, you have access to a limited number of CPU hours to spend. They are not assigned to users, but to projects and are shared between the users who are working on the same project (i.e. your research partners). Such projects are called accounts and are a different concept from your username.

Account for the School

- The accounts created for this school are:
  - `#SBATCH --account=s_tra_eudat // Normal Partition`
  - `#SBATCH --account=s_tra_eudatgpu // GPU Partition`

- They will expire two weeks after the end of the school and are shared between all the students.

SLURM commands - 1

After the job script is ready, all there is left to do is to submit it:

- `$ sbatch <job_script>`
  - Your job will be submitted to the SLURM scheduler and executed when there will be nodes available (according to your priority and the partition you requested)

- `$ squeue -u $USER`
  - Shows the list of all your scheduled jobs, along with their status (idle, running, closing, ...) Also, shows you the job id required for other SLURM commands

SLURM commands - 2

- `scontrol show job <job_id>`
  - Provides a long list of information for the job requested.
  - In particular, if your job isn’t running yet, you'll be notified about the reason it is not starting and, if it is scheduled with top priority, you will get an estimated start time
- `scancel <job_id>`
  - Removes the job (queued or running) from the scheduled job list by killing it

SLURM commands - 3

- `sinfo`
  - `sinfo -p <partition name>`
  - `sinfo -l`
  - `sinfo -N -l -p gll_usr_prod`
  - **Provides information about SLURM nodes and partitions**

- `sacct`
  - `sacct OPTIONS <job_id>`
  - **Displays accounting data for all jobs and job steps in the**
  - **SLURM job accounting log or Slurm database.**

Thank you!