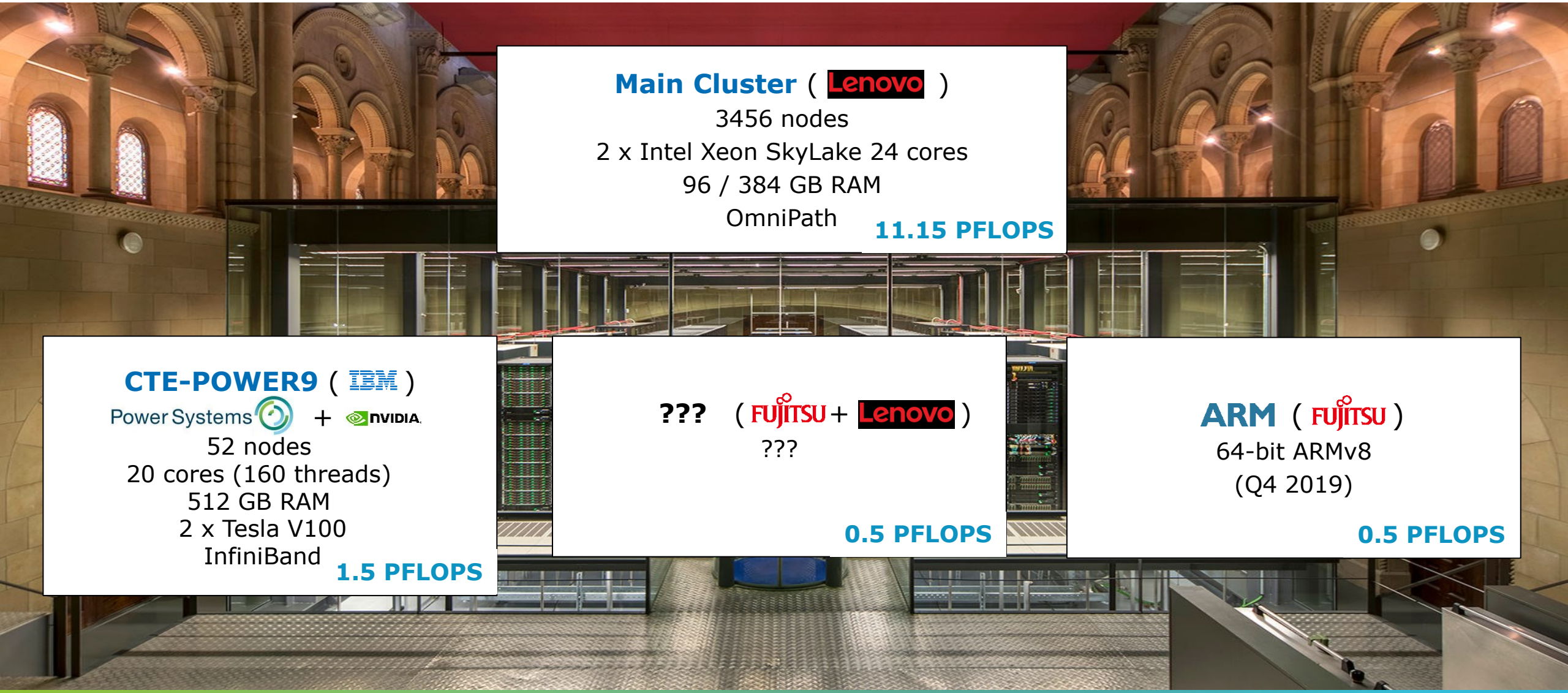
Computer system
&
Software environment

Germán Llort, Lau Mercadal
Barcelona Supercomputing Center

MareNostrum 4



Main Cluster (Lenovo)
3456 nodes
2 x Intel Xeon SkyLake 24 cores
96 / 384 GB RAM
OmniPath
11.15 PFLOPS

CTE-POWER9 (IBM)
Power Systems + NVIDIA
52 nodes
20 cores (160 threads)
512 GB RAM
2 x Tesla V100
InfiniBand
1.5 PFLOPS

??? (FUJITSU + Lenovo)
???
0.5 PFLOPS

ARM (FUJITSU)
64-bit ARMv8
(Q4 2019)
0.5 PFLOPS

Logins

- **MareNostrum (Main Cluster)**
  - 3 external accessible logins:
    - mn1.bsc.es
    - mn2.bsc.es
    - mn3.bsc.es

```
laptop> ssh -Y <USER>@mn2.bsc.es
```

  - 2 internal accessible logins:
    - login4
    - login5

```
login2> ssh -Y login4
```

- **CTE-POWER9**
  - 2 external accessible logins:
    - plogin1.bsc.es
    - plogin2.bsc.es

```
laptop> ssh -Y <USER>@plogin1.bsc.es
```

- No outgoing connections
- 5 minutes CPU time limit

Login Usage: Do's & Don'ts

Manage & edit files	Run production executions
Small & medium compilations	Copy large amount of files
Submit jobs to batch system	Long and heavy load graphical interfaces
Check results and prepare scripts	mpirun

Filesystems

- High performance parallel filesystem: GPFS

Folder	Intended usage
/gpfs/apps	Support vetted applications (e.g.: Python, TensorFlow, Boost, LAPACK, HDF5…)
/gpfs/home	User data (with backup), scripts, code, documents… (~40 GB quota)
/gpfs/projects	Execution data (with backup), inputs, checkpoints, custom installations…
/gpfs/scratch	Temporary files (no backup!), huge log files…
/gpfs/archive	Long term storage, old data…
/scratch	Local disk, not shared between nodes, erased after execution, 200GB, fast!

- A few tricks for better I/O performance:
  - Keep below 1000 files per folder
  - Avoid writing many files in the same folder simultaneously
  - Avoid small files (blocksize 16 MB)

Modules

- Show available modules:

```
login2> module avail          // Show all modules in the system
login2> module avail mpi      // Filter MPI modules only
```

- Show modules **currently in use**:

```
login2> module list
Currently Loaded Modules:
  1) intel/2017.4  2) impi/2017.4  3) mkl/2017.4  4) bsc/1.0
```

- Load / unload modules:

```
login2> module unload impi
login2> module load openmpi
```

Compilers

- Open source and licensed compilers available

MareNostrum 4

```
login2> module avail gcc
-------------------------------------
gcc/5.4.0  gcc/7.2.0 (L,D)  gcc/8.1.0
...

login2> module avail intel
-------------------------------------
intel/2017.4 (L,D)  intel/2018.4  ...
```

CTE-POWER9

```
plogin1> module avail gcc
-------------------------------------
gcc/6.4.0 (L,D)  gcc/7.3.0  gcc/8.1.0
...

plogin1> module avail ibm
-------------------------------------
ibm/13.1.6  ibm/16.1.0 (L,D)
```

- MN4: choose between Intel and GNU:

```
login2> module load intel/2017.4
```

- CTE-POWER9: choose between IBM and GNU:

```
plogin1> module load gcc/6.4.0
```

Compilers (II)

- Which one?
  - Each software is different, but…
    - Licensed compilers up to 20% performance increase in some applications, but…
      - Over optimization may result in precision errors

- For long compilations, use interactive nodes:

```
login2> salloc -p interactive

salloc: Granted job allocation
salloc: Waiting for resource configuration
salloc: Nodes are ready for job

nodeXXX> make
```

MPI versions

- Multiple implementations & versions available from modules

MareNostrum 4

```
login2> module avail mpi
-------------------------------------
...
impi/2017.4 (L,D)  openmpi/1.10.7 (D)
impi/2018.4        openmpi/3.0.0
...
```

CTE-POWER9

```
plogin1> module avail mpi
-------------------------------------
...
openmpi/3.0.0 (L,D)  mpich/3.2.1 (D)
openmpi/4.0.0
...
```

- MN4: Choose IntelMPI / OpenMPI

```
login2> module load impi/2017.4
```

- CTE-POWER9: choose OpenMPI / MPICH

```
plogin1> module load openmpi/3.0.0
```

How to compile MPI programs?

- Load your MPI module of choice:

```
login2> module unload impi
login2> module load openmpi
```

- Use any of the following compiler front-ends:

Language	Compiler command
C	mpicc / mpiicc
C++	mpiCC / mpicxx / mpic++ / mpiicpc
Fortran 77	mpif77 / mpiifort
Fortran 90	mpif90 / mpiifort

- If you want to check what the wrappers do:

```
login2> mpicc -show
```

- Back-end compilers are managed by loaded modules (intel / gcc)

Batch System

- MareNostrum 4 uses SLURM for batch processing

- Benefits of using jobscripts
  - Define resource needs
  - Reusable
  - Document needs and requests
  - Jobscripts are shellscripts with special markings

- Each submission gets a job ID

```
login2> sbatch job_script.sh
Submitted batch job 1405572

login2> squeue
  JOBID  PARTITION  NAME  USER     ST TIME NODES NODELIST (REASON)
1405572       main  test  nct00003 PD 0:00     1 (Resources)
```

SLURM commands

- How to submit a job:

```
login2> sbatch job_script.sh
Submitted batch job 1405572
```

- Check status of jobs submitted:

```
login2> squeue
  JOBID  PARTITION  NAME  USER     ST TIME NODES NODELIST (REASON)
1405572       main  test  nct00003 PD 0:00     1 (Resources)
```

- Cancel a job:

```
login2> scancel 1405572
```

SLURM: General parameters

Parameter	Description	Example
`-n | --ntasks`	Number of tasks	`#SBATCH -n 32`
`-t | --time`	Time limit	`#SBATCH -t 01:00`
`-J | --job-name`	Job name	`#SBATCH -J myjob`
`-o | --output`	Output file	`#SBATCH -o %j.out`
`-e | --error`	Error file	`#SBATCH -e %j.err`
`-D | --workdir`	Current working dir	`#SBATCH -D /my/path`
`--qos`	Queue	`#SBATCH --qos=debug`
`--exclusive`	Execute alone (don’t share node)	`#SBATCH --exclusive`

SLURM: Process layout

- Define specific load balance configurations:

Parameter	Description	Example
`--ntasks-per-core`	Number of tasks per core	`#SBATCH --ntasks-per-core=1`
`--ntasks-per-socket`	Number of tasks per socket	`#SBATCH --ntasks-per-socket=12`
`--ntasks-per-node`	Number of tasks per node	`#SBATCH --ntasks-per-node=24`
`--cpus-per-task`	CPU's (threads) per task	`#SBATCH --cpus-per-task=2`

- System balances equally by default

SLURM: Reservations

- Add to your jobs during this course:

Parameter	Description	Example
`--reservation`	Request reserved resources	`#SBATCH --reservation=PATCVIHPS`

Job Examples: Sequential

```
login2> vi job_sequential.sh
```

```
#!/usr/bin/env bash

#SBATCH --job-name=test
#SBATCH --ntasks=1
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH --time=00:10:00

./my-sequential-program
```

I want to be alone on my node

```
login2> sbatch job_sequential.sh
```

Job Examples: Threaded

```
login2> vi job_threaded.sh
```

```
#!/usr/bin/env bash

#SBATCH --job-name=test
#SBATCH --ntasks=1
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH --time=00:10:00

export OMP_NUM_THREADS=16

./my-threaded-program
```

Set by default to max cores

```
login2> sbatch job_threaded.sh
```

Job Examples: MPI

```
login2> vi job_mpi.sh
```

```
#!/usr/bin/env bash

#SBATCH --job-name=test
#SBATCH --ntasks=96
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH --time=00:10:00

module unload impi
module load openmpi

srun ./my-mpi-program
```

Number of tasks (== 2 full nodes)

Select MPI versión (same as used to compile)

Launch with “srun”

```
login2> sbatch job_mpi.sh
```

Job Examples: MPI + OpenMP

```
login2> vi job_mpi+omp.sh
```

```
#!/usr/bin/env bash

#SBATCH --job-name=test
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=6
#SBATCH --exclusive
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH --time=00:10:00

export OMP_NUM_THREADS=6

module unload impi
module load openmpi

srun ./my-mpi+omp-program
```

2 nodes: 16 tasks x 6 threads each

Match to avoid over-/under-subscription

Select MPI version (same as used to compile)

Launch with “srun”

```
login2> sbatch job_mpi+omp.sh
```

User Guides

- **MareNostrum 4**
  - https://www.bsc.es/user-support/mn4.php
- **CTE-POWER9**
  - https://www.bsc.es/user-support/power.php