# INTRODUCTION TO SUPERCOMPUTING

## SLURM Workload Manager

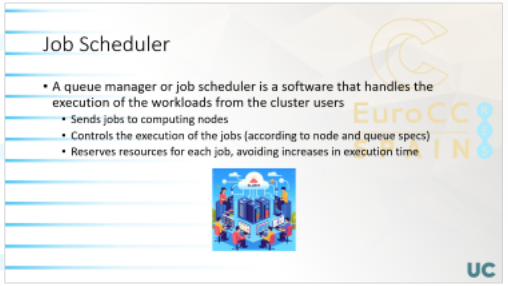
EuroCC SPAIN

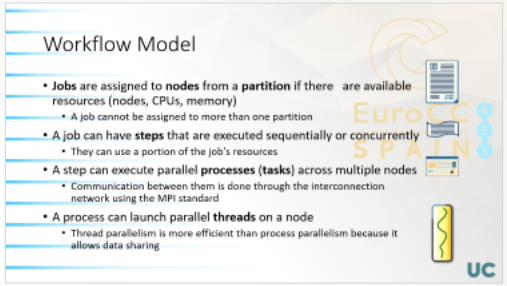
RES

UC | Universidad de Cantabria

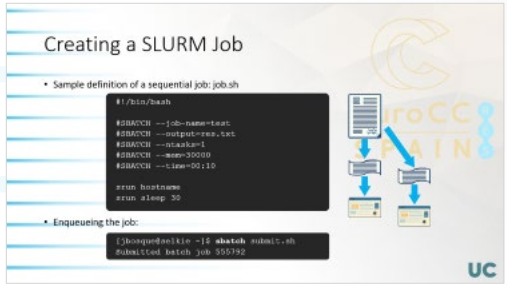
Computer Arquitecture and Technology | A-TC

# SLURM workload manager





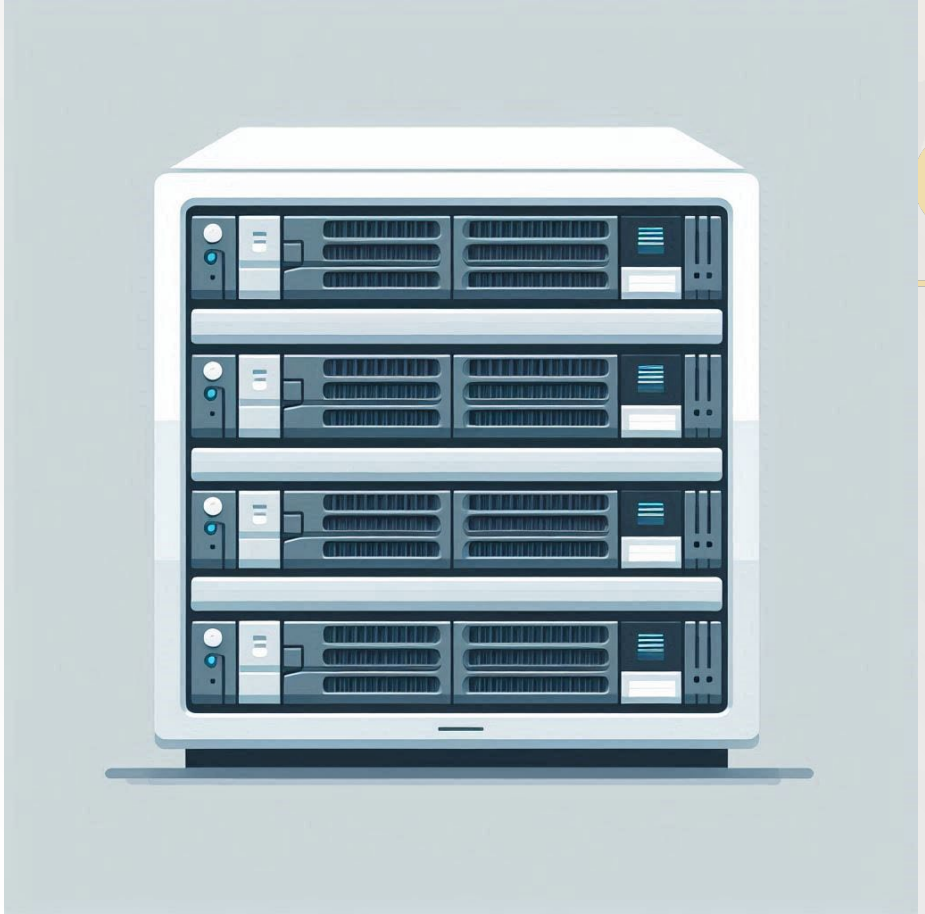



# Job Scheduler

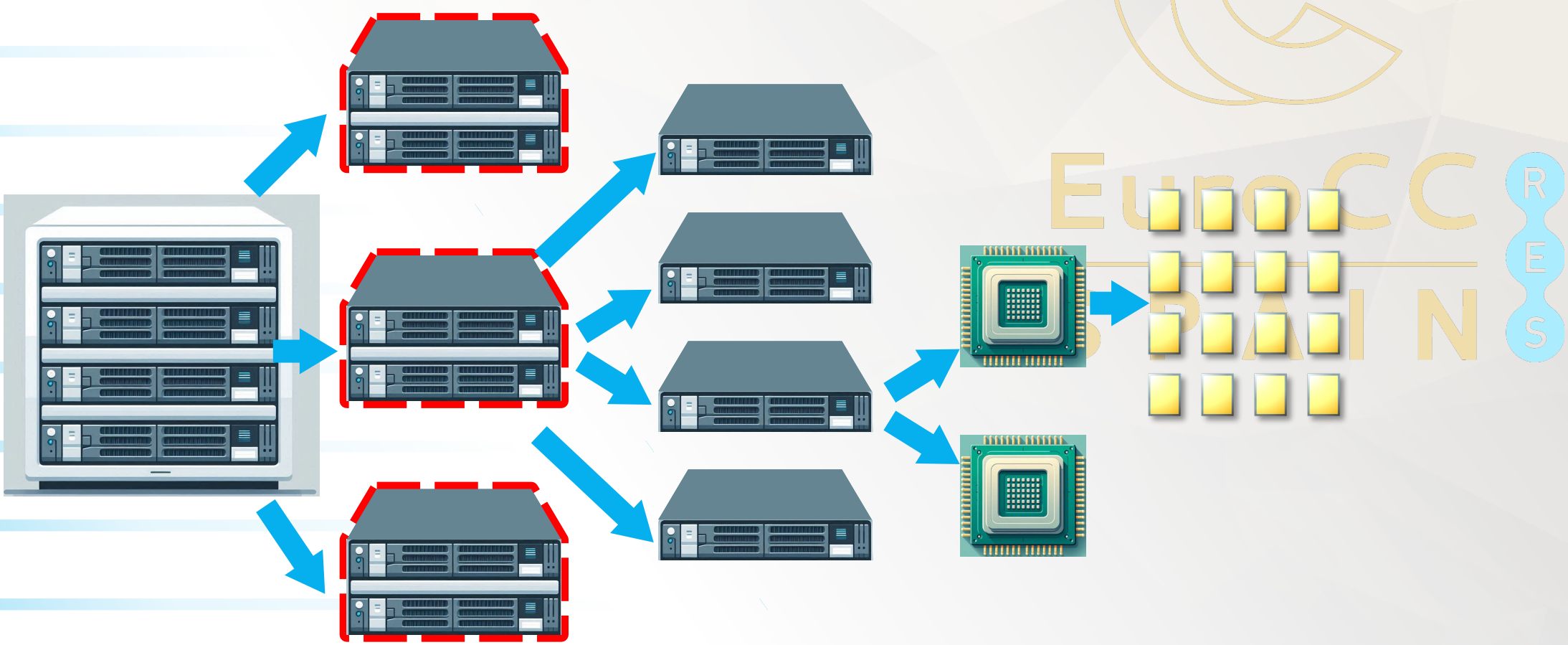
- A queue manager or job scheduler is a software that handles the execution of the workloads from the cluster users
  - Sends jobs to computing nodes
  - Controls the execution of the jobs (according to node and queue specs)
  - Reserves resources for each job, avoiding increases in execution time

# Job Scheduler

- Partition: group nodes into sets
  - They do not have to be disjoint
  - They have an associated job queue
  - They can be restricted to users, requested time, etc.
- Node: container of computing resources
  - A task is assigned to a set of nodes
  - The assignment gives users rights over the nodes
- CPU: computing resource
  - They carry out the computation defined in the tasks
  - Their capacity can be assigned, but not the CPUs themselves



# Job Scheduler



Cluster Partition Node Processor Core

# Workflow Model

- **Jobs** are assigned to **nodes** from a **partition** if there are available resources (nodes, CPUs, memory)
  - A job cannot be assigned to more than one partition
- A job can have **steps** that are executed sequentially or concurrently
  - They can use a portion of the job's resources
- A step can execute parallel **processes** (**tasks**) across multiple nodes
  - Communication between them is done through the interconnection network using the MPI standard
- A process can launch parallel **threads** on a node
  - Thread parallelism is more efficient than process parallelism because it allows data sharing

# SLURM

- Job scheduler for Linux clusters
  - https://slurm.schedmd.com/
- Developed in Livermore Computing Center
  - Open-Source, supported by the original developers
  - Fault-tolerance
  - Highly scalable (TOP500)
- It has three main functions:
  - Temporary allocation of resources (nodes) to users
  - Allowing process management (startup, monitoring…)
  - Arbitrating the shared use of resources

# SLURM Basic Commands

- **`sinfo`**: reports the status of the partitions and the compute nodes managed by Slurm
  - Wide variety of filtering, sorting, and formatting options

```
[jbosque@selkie ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
latc*        up   1-00:00:00     24   down* n16-[21-44]
lsci         up   1-00:00:00     42   down* n232-[21-62]
lscii        up   1-00:00:00     25   down* n232-[71-95]
lsciii       up   1-00:00:00     17   down* n232-[121-137]
lsciv        up   1-00:00:00     31   down* n232-[171-201]
lscv         up   1-00:00:00     16   down* n132-[21-36]
all          up   1-00:00:00    155   down* n16-[21-44],n132-
[21-36],n232-[21-62,71-95]
```

# SLURM Basic Commands

- **`squeue`**: reports the status of running and pending jobs.
  - It has a wide variety of filtering, sorting, and formatting options: `–user`, `–partition`, or `–state`
  - By default, it reports running jobs in order of priority and then pending jobs in order of priority

```
[jbosque@selkie ~]$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST
548449 normal smart_32x32 enrique R 0-21:09:15 1 comp005
548992 normal smart_32x32 enrique R 0-21:09:15 1 comp005
548214 normal fsin esteban R 0-07:09:15 1 comp005
548933 normal smart_32x32 enrique R 0-12:10:34 1 comp004
548445 normal fsin esteban R 0-09:10:34 1 comp004
548877 normal MultiRingRL enrique R 0-00:26:34 2 comp00[1,2]
```

# SLURM Basic Commands

- **`sbatch`**: submits a job script for later execution
  - The script will typically contain one or more `srun` commands to launch parallel tasks
- **`scancel`**: cancels a pending or running job or job step
- **`sstat`**: obtains information about the resources used by a running job or job step

```
[jbosque@selkie ~]$ sstat -j 548449
JobID    MaxVMSize AveCPU   NTasks AveCPUFreq ConsumedEnergy
-----    --------- ------   ------ ---------- --------------
555793.1 203540K   00:00.000 1            800M            642
```

# Creating a SLURM Job

- The definition of a job is divided into two parts:
  - Resource request
    - Number of CPUs
    - Estimated runtime
    - Amount of memory
  - Job definition
    - Description of steps
    - Definition of variables
    - Which programs are executed

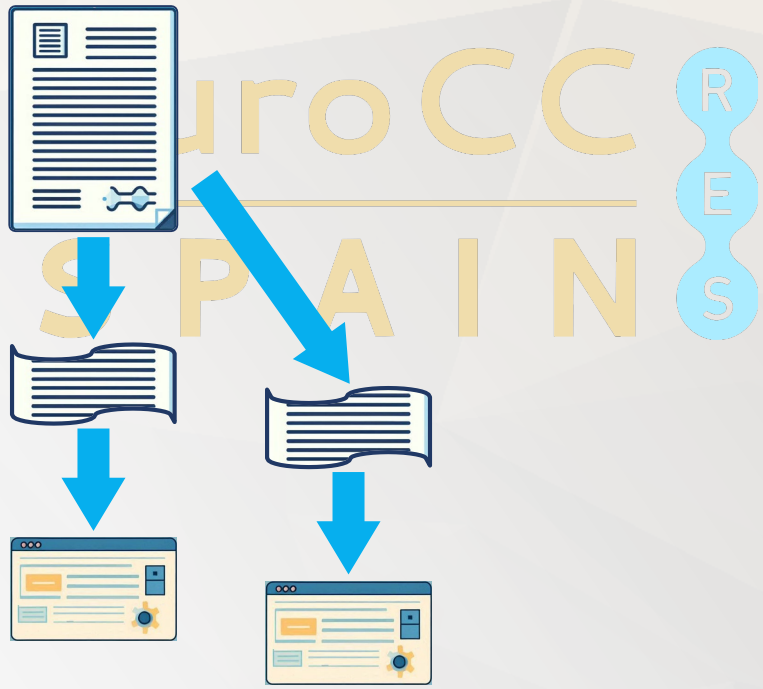
# Creating a SLURM Job

- Sample definition of a sequential job: job.sh

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --output=res.txt
#SBATCH --ntasks=1
#SBATCH --mem=30000
#SBATCH --time=00:10

srun hostname
srun sleep 30
```



- Enqueueing the job:

```
[jbosque@selkie ~]$ sbatch submit.sh
Submitted batch job 555792
```

# Most Frequent SLURM Directives

| Option | Description |
|---|---|
| `--job-name=name` | Job name – It facilitates job identification |
| `--nodes=\#` | Number of nodes to use |
| `--ntasks=\#` | Number of tasks to execute (distributed parallelism) |
| `--ntasks-per-node=\#` | Number of tasks assigned to each physical node |
| `--time=[[DD-]HH:]MM:SS` | Maximum job runtime |
| `--mem-per-cpu=\#` | RAM required per CPU (MB) |
| `--mem=\#` | RAM required per node (MB) |
| `--output=out-\%j.log` | File to store standard output from the job |
| `error=err-\%j.log` | File to store error output from the job |
| `--chdir=[dirname]` | Working folder |

# SLURM Job States

- Once a job is sent to the queue, it goes through different states:
  - **PENDING**: the job is waiting for resources to execute
  - **RUNNING**: when resources are available and the priority allows it, the job starts executing
  - **COMPLETED**: when the execution finishes successfully
  - **FAILED**: when the execution ends abnormally

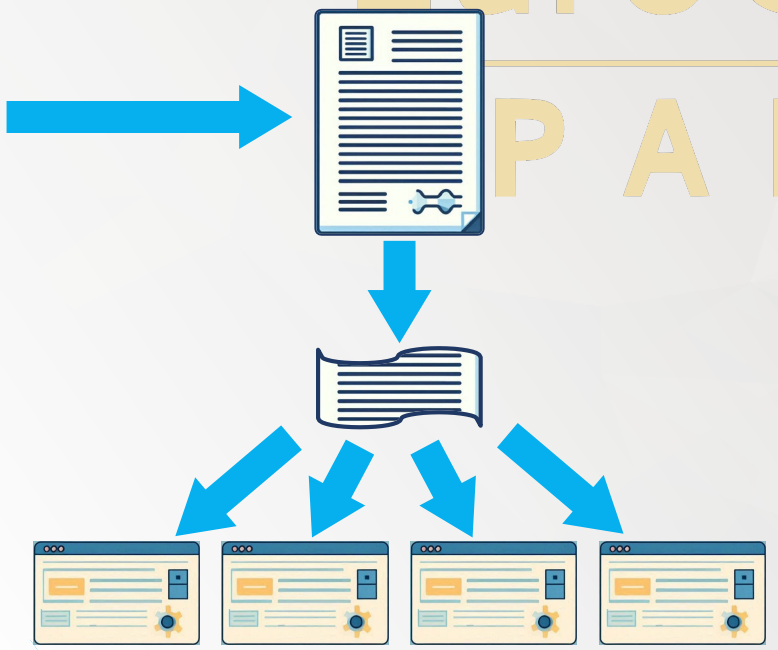
# MPI Jobs

- We start with a simple MPI program: *Hello World*
- The code is saved in the file **`hello_mpi.c`**

```
#!/bin/bash

#SBATCH --job-name=test_mpi
#SBATCH --output=res_mpi.txt
#SBATCH --ntasks=4
#SBATCH --nodes=2
#SBATCH --time=00:10
#SBATCH --mem-per-cpu=200

prun ./hello_mpi
```



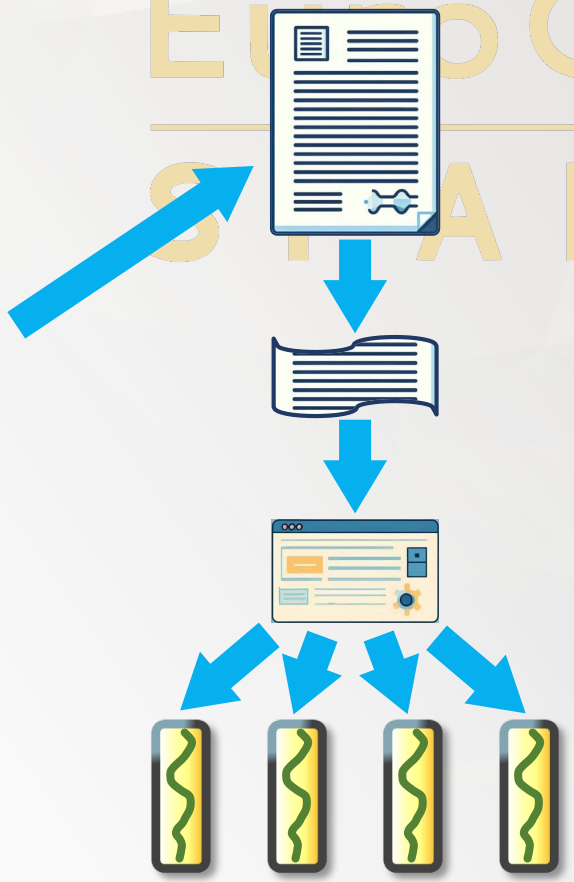
# OpenMP Jobs

- We try with an equivalent OpenMP program: *Hello World*
- The code is saved in the file **`hello_omp.c`**
- To compile the code, we use the command:
  - `gcc -fopenmp -o hello_omp hello_omp.c`

```
#!/bin/bash

#SBATCH --job-name=test_omp
#SBATCH --output=res_omp.txt
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=200
#SBATCH --time=00:10

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun ./hello_omp
```



# SLURM environment variables

- Variables are defined in the job environment when the script is executed through the job scheduler
  - They can be used in the script for different purposes
- Most interesting variables:
  - **`$SLURM_JOB_ID`**: job identifier
  - **`$SLURM_JOB_NAME:`** job name
  - **`$SLURM_SUBMIT_DIR`**: submit directory
  - **`$SLURM_JOB_NUM_NODES`**: number of nodes assigned to the job
  - **`$SLURM_CPUS_ON_NODE`**: number of cores per node
  - **`$SLURM_NTASKS`**: total number of cores per job
  - **`$SLURM_NODEID`**: index of the executing node in relation to the nodes assigned to the job
  - **`$SLURM_PROCID`**: index of the task in relation to the job

# EURO$^2$


UC | Universidad de Cantabria

Computer Arquitecture and Technology | A-TC

**Ramón Beivide, José Luis Bosque,**

Pablo Fuentes, Carmen Martínez, Borja Pérez

**www.atc.unican.es**

EURO$^2$



Funded by
the European Union


Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking ( JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.