Clusters formation

Introduction

Yann.Delcambre@latmos.ipsl.fr
Philippe.Dos-Santos@ipsl.fr
Sebastien.Gardoll@ipsl.fr

Carlos.Mejia@locean-ipsl.upmc.fr
Karim.Ramage@ipsl.fr
Philippe.Weill@latmos.ipsl.fr

Don’t forget to acknowledge our computing and data center in your publications
https://espri.ipsl.fr/about-espri/acknowledgement/

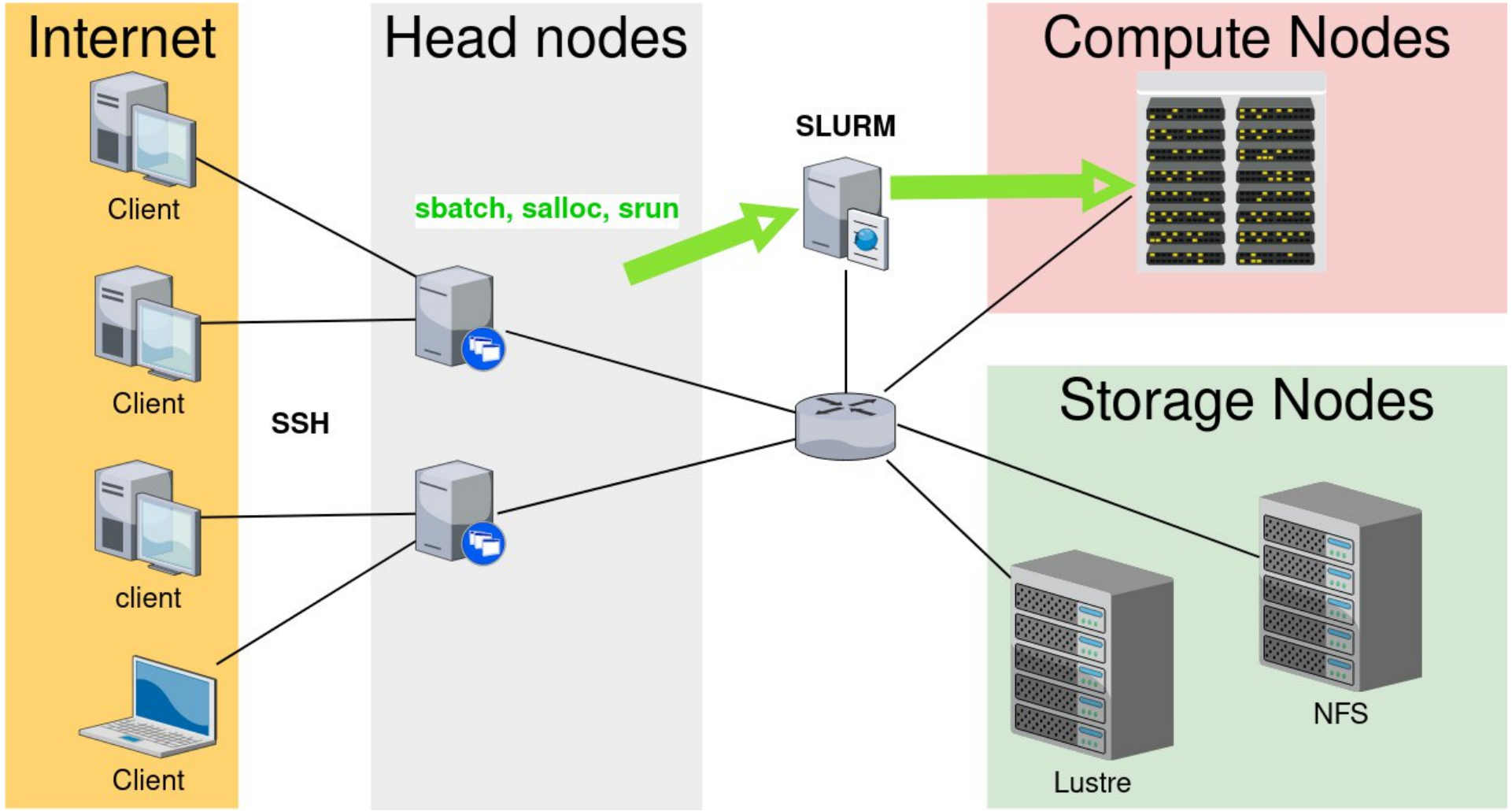
Plan

- **IPSL Clusters** [https://documentations.ipsl.fr/spirit/getting_started/cluster.html]
- Node hardware and architecture [socket, core, memory (NUMA), GPU]
  - https://documentations.ipsl.fr/spirit/spirit_clusters/computing_nodes.html
- SLURM (Job Management) [https://documentations.ipsl.fr/spirit/common/slurm.html]
- Software and modules [https://documentations.ipsl.fr/spirit/common/module_command.html]
- Storage spaces and I/O https://documentations.ipsl.fr/spirit/spirit_clusters/user_spaces.html
- Container with apptainer [singularity successor]
- IPSL support [meso-support]
  - https://documentations.ipsl.fr/spirit/getting_started/support.html

IPSL clusters

Head Nodes:

- IPSL-SU, Spirit:
  - spirit1.ipsl.fr
  - spirit2.ipsl.fr
- IPSL-X, SpiritX:
  - spiritx1.ipsl.fr
  - spiritx2.ipsl.fr
- IPSL-G, HAL:
  - hal.ipsl.fr



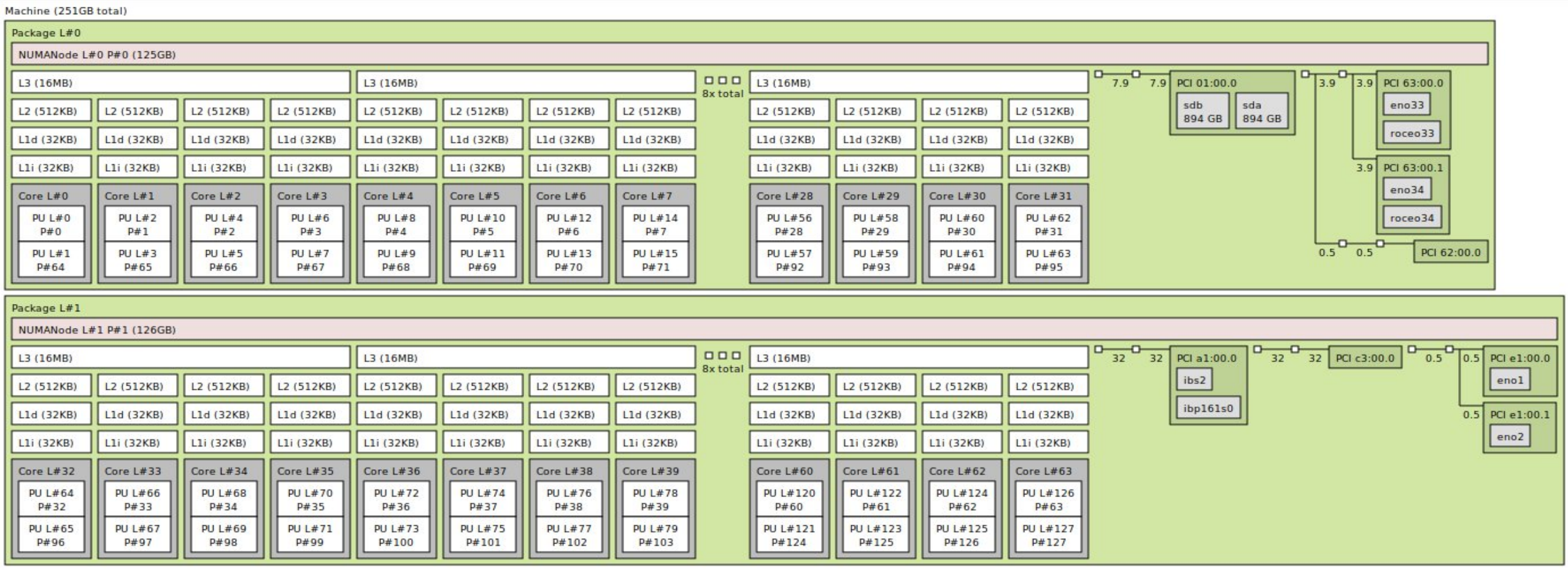
A cluster is like a hotel complex

Cluster	Hotel Complex
User	Client
Head Node	Entrance building
Slurm Resource Manager (controller)	Reception + reservation system
Node	Room building (2 floors)
Socket	Floor
Memory	Room surface
Core	Bed(2 places)
GPU	Balcony
Zen4 Partition node	Standard room building
Zen16 Partition node	Premium room building (suite)

Plan

- IPSL clusters [https://documentations.ipsl.fr/spirit/getting_started/cluster.html]
- **Node hardware and architecture** [socket, core, memory (NUMA), GPU]
  - https://documentations.ipsl.fr/spirit/spirit_clusters/computing_nodes.html
- SLURM (Job Management) [https://documentations.ipsl.fr/spirit/common/slurm.html]
- Software and modules [not written]
  [https://documentations.ipsl.fr/spirit/common/module_command.html]
- Storage spaces and I/O [not finished]
  - https://documentations.ipsl.fr/spirit/spirit_clusters/user_spaces.html
- Container with apptainer [not available]
- IPSL support [meso-support]
  - https://documentations.ipsl.fr/spirit/getting_started/support.html

Node architecture: zen4 partition (Spirit/SpiritX)



IPSL-X, SpiritX / IPSL-SU, Spirit

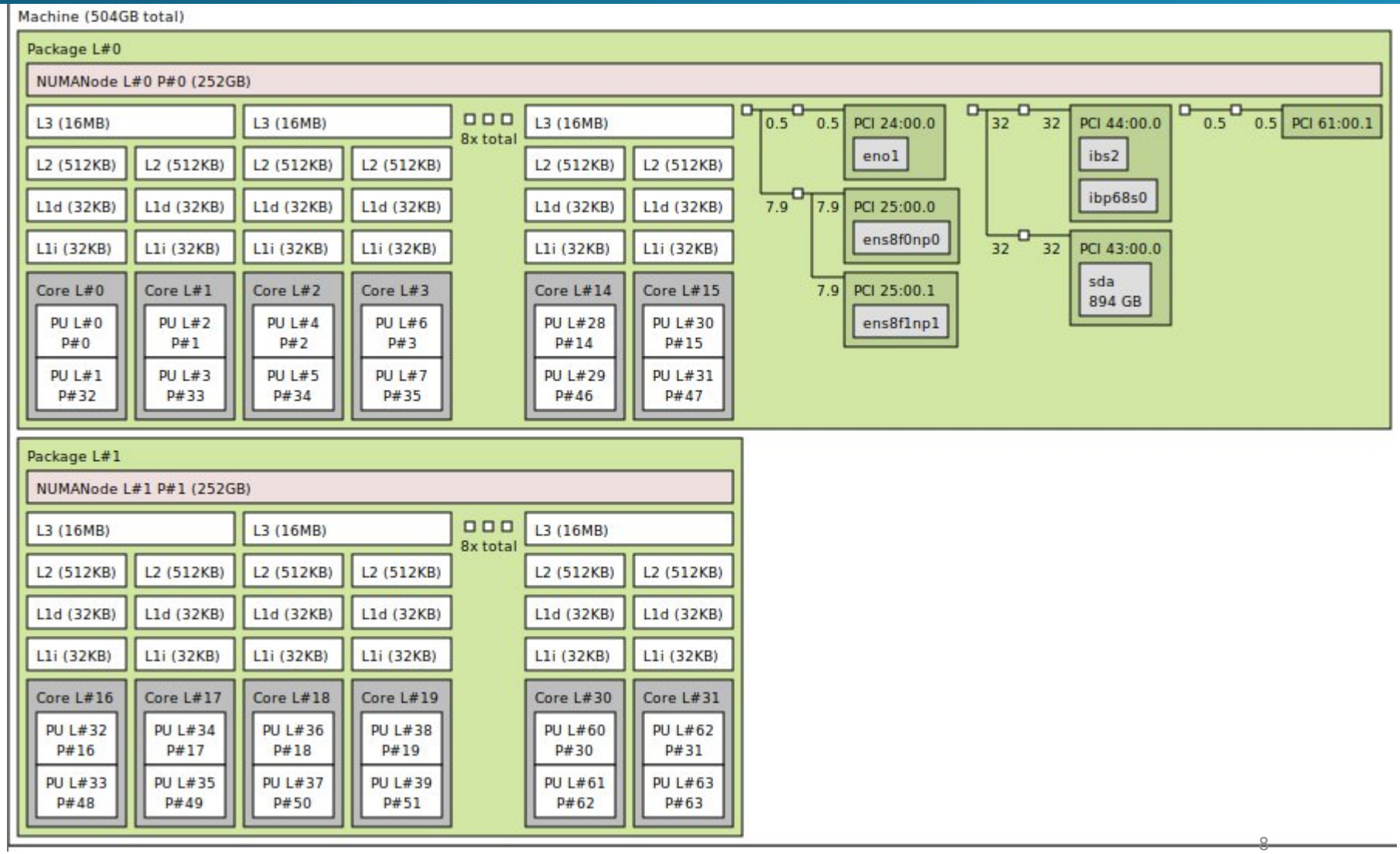
AMD ZEN2 (10K€) processor: 2 sockets, 32 Cores per socket, 4 Go memory per core 3840Mo/core for job .

NUMA: Non Uniform Memory Access Accessing memory from one socket to another is really slower

Node architecture: zen16 partition (Spirit/SpiritX)

- IPSL-X, SpiritX
- IPSL-SU, Spirit

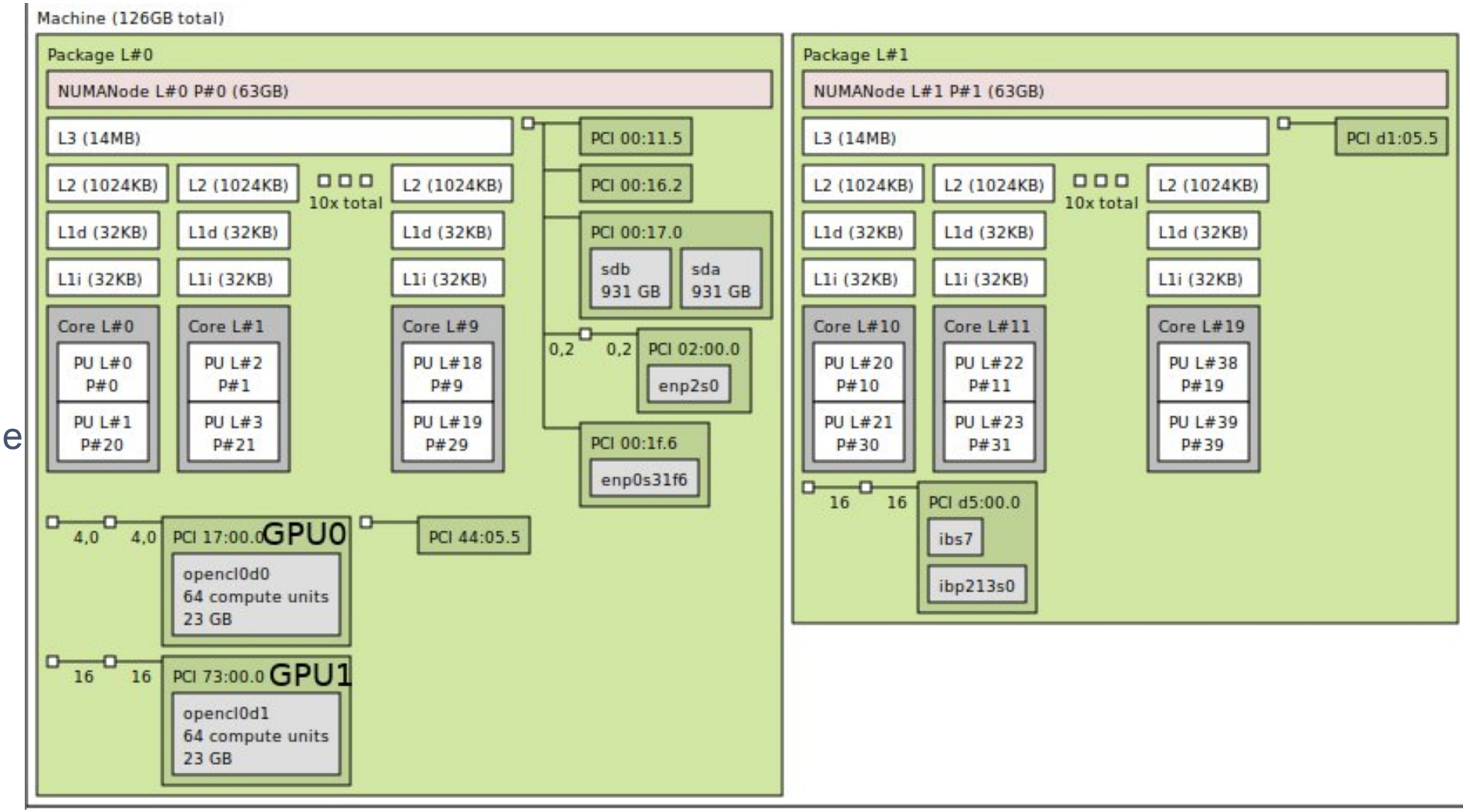
- AMD ZEN2 processor(7K€)
- 2 sockets
- 16 cores per socket
- Memory/core: 16 Go
- Memory/core for job: 15872Mo



Node architecture: GPU (HAL)

- IPSL-G, HAL

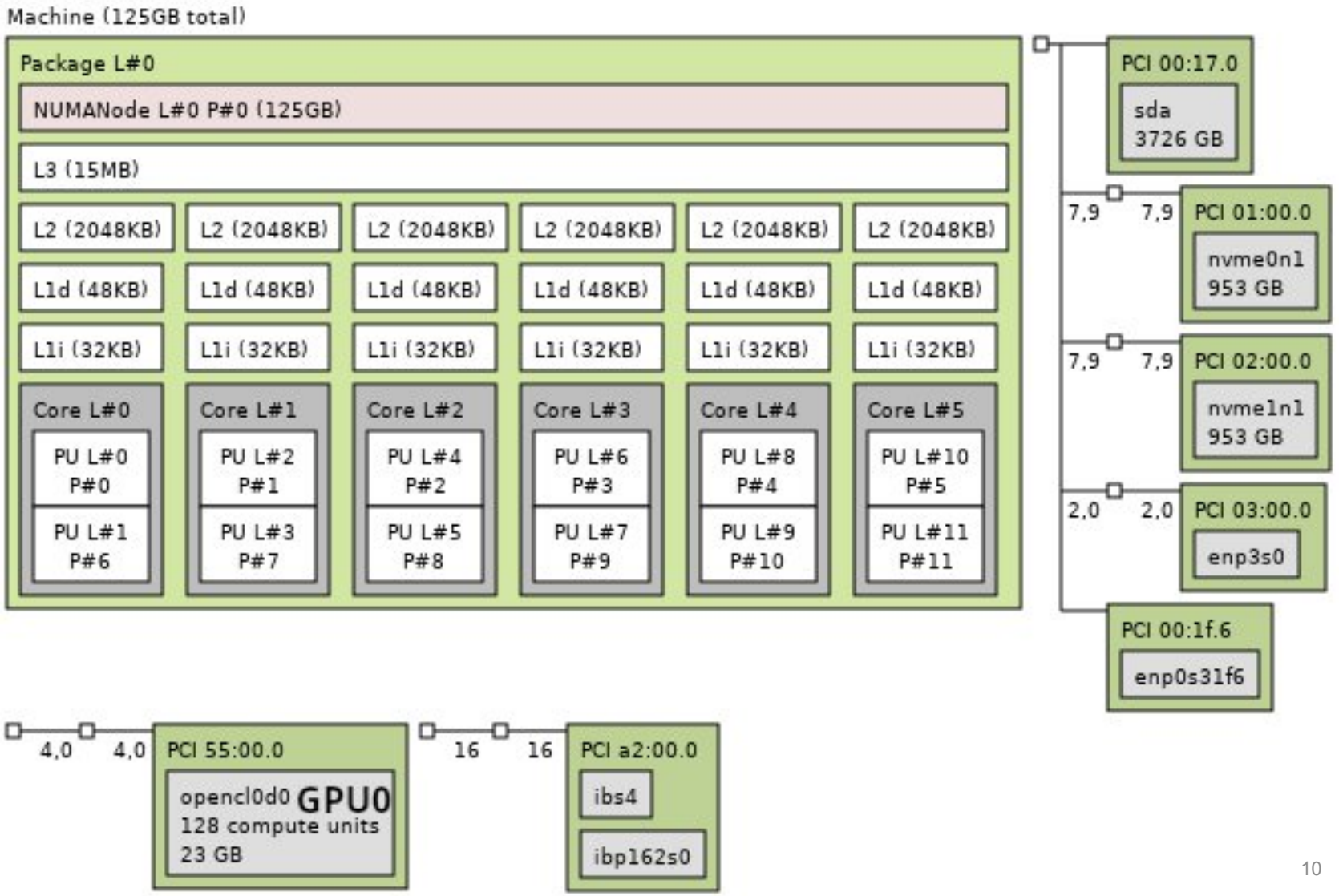
- INTEL processor
- 1 or 2 sockets
- 6 to 10 cores per socket
- Memory per core : variable
- GPUs: 1 par socket



Node architecture: GPU (HAL)

- IPSL-G, HAL

- INTEL processor
- 1 socket
- 6 core
- Memory per core : variable
- GPUs: 1



Plan

- IPSL clusters
- Node architecture
- **SLURM** (Job Management) (https://documentations.ipsl.fr/spirit/common/slurm.html)
  - **Introduction**
  - Computing task types (serial, parallel, distributed, hybrid)
  - Job types and job submission (interactive/batch)
  - Job array
  - Pipeline of jobs (dependency, chaining and recursive)
  - Job practice
  - Jobs control commands
  - Jobs monitoring, statistics and best practices

SLURM introduction

SLURM: Simple Linux Utility for Resource Management

Slurm is an orchestrator that managers jobs on a cluster

- Resources are:
  - **nodes**,
  - **CPU cores**,
  - **memory**,
  - **time**,
  - **GPUs**,
  - etc.

- Resources requested are exclusive to a job:
  - Nodes can host different jobs, but
  - Memory and CPU cores are never shared between the jobs.

- Only request the necessary resources: check resource utilization!

SLURM: resources and partitions

spirit - spiritx : see “check-cluster” command

Partitions limit	mem/cpu	time	NB node max
zen4 (default)	3 840 Mo/cpu max:64cpu	default 1h Max: 168h	1
zen16	15 872 Mo/cpu max:32cpu	default 1h Max: 72h	1

Interactive job : max time 10h and only one per user

Users limit	CPU max	MEM max	NB Jobs max	MaxSubmitJobs
Internal Users	96	256G	64	1000
External Users	48	128G	32	500

HAL: see “check-cluster” command

Partitions limit	GPU	time	NB Jobs max	MaxSubmitJobs
interactive (default)	max: 2	default: 1h Max: 10h	2	500
batch	max: 2	default: 1h Max: 72h	2	500

Plan

- IPSL clusters
- Node architecture
- **SLURM** (Job Management) (https://documentations.ipsl.fr/spirit/common/slurm.html)
  - Introduction
  - **Computing task types (serial, parallel, distributed, hybrid)**
  - Job types and job submission (interactive/batch)
  - Job array
  - Pipeline of jobs (dependency, chaining and recursive)
  - Job practice
  - Jobs control commands
  - Jobs monitoring,statistics and best practices

Slurm computing task types

- Sequential: 1 node, 1 process, 1 computation at the same time on 1 CPU core
- Parallel (memory shared): only on 1 node, 1 process, several computations at the same time on different CPU cores
  - OpenMP
  - POSIX Threads
  - Some functions of certain Python computation libraries (not all functions of Numpy!)
- Distributed: same or different nodes, several processes that communicate with each other, 1 computation at the same time per process on 1 CPU core
  - MPI
  - Multiple Slurm jobs or Slurm job arrays
  - Python Dask
- Hybrid: distributed and parallel
  - MPI + OpenMP / POSIX threads
  - Python Dask with some functions of certain Python computation libraries

Plan

- IPSL clusters
- Node architecture
- **SLURM** (Job Management) (https://documentations.ipsl.fr/spirit/common/slurm.html)
  - Introduction
  - Computing task types (serial, parallel, distributed, hybrid)
  - **Job types and job submission (interactive/batch)**
  - Job array
  - Pipeline of jobs (dependency, chaining and recursive)
  - Job practice
  - Jobs control commands
  - Jobs monitoring, statistics and best practices

Slurm job types

Two types of SLURM job:

- Interactive (max 10 hours on all clusters and only one per user on Spirit[X], 2h for HAL).
  - Enable foreground computing and exploratory development (like JupyterLab).
  - Enable IO adjustments (ex: computation parameters, plots, etc.)
  - Debug scripts.
  - Disconnection ends the job!

- Batch
  - Enable background computing without any user interaction.
  - Jobs run even while the users are not connected.
  - Enable running several jobs (user limit) at the same time and queue several others (<= 1000).
  - Job parameters are given by shell script (ex: Bash).

Job submission: interactive job

Interactive jobs are submitted using the command `srun` with `--pty` option and the argument `bash`.

Syntax: `srun --pty [option1] [options2] [...optionsn] bash`

Example of the submission of an interactive job with default resources (1 node, 1 task, 1 CPU core, 1 hour, default partition zen4, default associated partition memory 4 Go) on Spirit:

```
user@spirit1:~$ srun --pty bash
user@spirit64-04:~$ slqueue --me # Display user's submitted jobs.
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            643373      zen4     bash  user     R       0:57      1 spirit64-04
user@spirit64-04:~$ # Do some cool stuff!
```

Job submission interactive job: srun options

- General options

```
--x11                  # Enable x11 forwarding(Graphic).
--partition=<string> # Set the partition. zen4 (default) or zen16 on Spirit[X] clusters ; interactive or batch on HAL.
--time=<dd-HH:mm:ss> # The maximum elapsed time. Must fit the partition limit!
```

- Resources options (ignored on HAL except `--gpus`)
  - Absolute options

```
--ntasks=<integer>   # The total number of tasks/processes.
--mem=<integer><g|m> # The total amount of memory. Also influences the number of CPU cores!
--gpus=[ampere:|turing:|ada:]<1|2> # The total number of GPU and their architecture, only on HAL.
```

  - Relative options

```
--ntasks-per-node=<integer>  # The number of tasks/processes executed per node (distributed jobs)
--cpus-per-task=<integer>    # The number of CPU cores per task/process (parallel jobs)
--mem-per-cpu=<integer><g|m> # The amount of memory requested per CPU core.
```

Job submission interactive job: examples 1/2

1. Request an interactive job on Spirit[X] zen4 with the shell Bash, enabling X11 forwarding, leveraging 6 GB memory and 2 CPU cores, during up to 2 hours:

```
user@spirit1:~$ srun --pty --x11 --mem 6G --time 2:00:00 bash
```

2. Request an interactive job on Spirit[X] zen16 with the shell Bash, enabling X11 forwarding, leveraging 6 GB memory and 1 CPU cores, during up to 2 hours:

```
user@spirit1:~$ srun --pty --x11 --partition zen16 --mem 6G --time 2:00:00 bash
```

3. Request an interactive job on Spirit[X] zen4 with the shell Bash, leveraging 16 GB memory and 4 CPU cores, during 1 hour:

```
user@spirit1:~$ srun --pty --cpus-per-task=4 bash
```

4. Request an interactive job on HAL interactive with the shell Bash, leveraging 1 GPU and eventually competing for the memory and CPU cores of the node with the other jobs, during up to 2 hours:

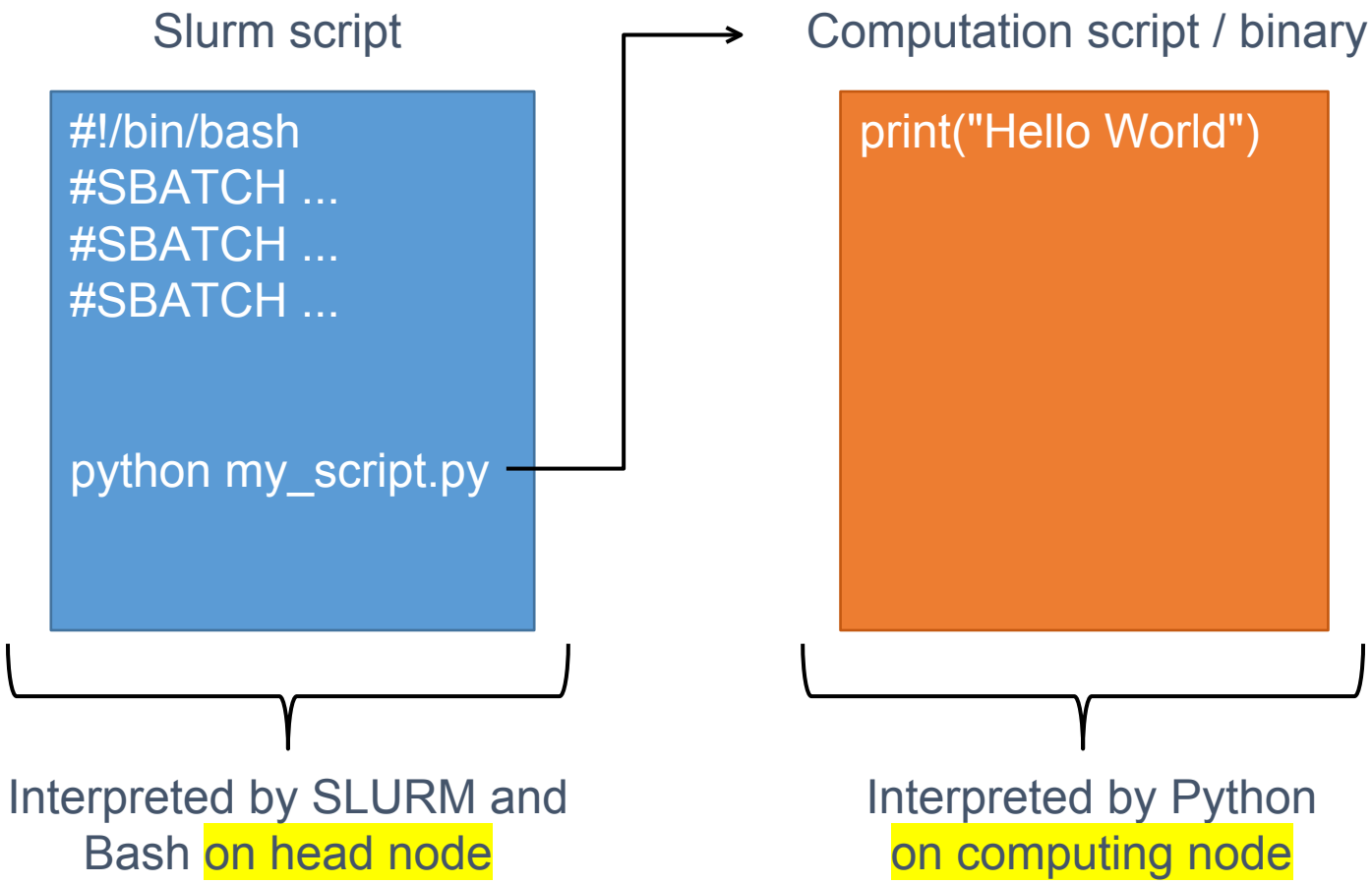
```
user@hal:~$ srun --pty --gpus=1 --time 2:00:00 bash
```

Job submission interactive job: examples 2/2

5. Request an interactive job on HAL interactive with the shell Bash, leveraging 2 turing GPU and all the memory and CPU cores of the node, during up to 2 hours:

```
user@hal:~$ srun --pty --gpus=turing:2 --time 2:00:00 bash
```

Batch job: slurm script



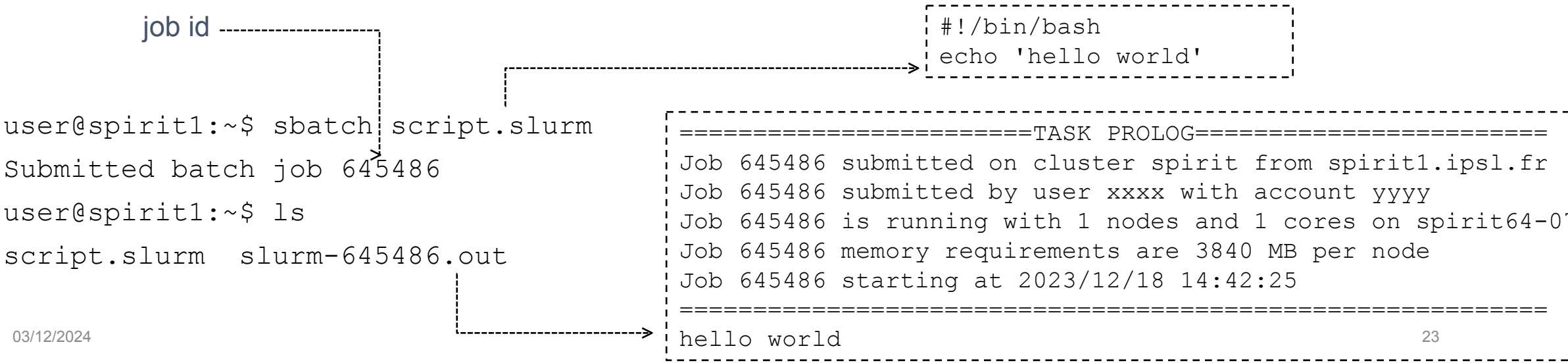
Batch job: submission

Batch jobs are submitted using the command `sbatch`.

Syntax:

```
sbatch [option1] [option2] <slurm_script_path> [slurm_script_arg1] [slurm_script_arg2]
```

Example of the submission of a batch job with default resources (1 node, 1 task, 1 CPU core, 1 hour, default partition, default associated partition memory) on Spirit:



Batch job: sbatch options

The `sbatch` options are the same as the options of `srun`. The slurm script may contain job specs (sbatch instructions ; next slide). The `sbatch` line options may supersede the sbatch instructions.

- More general options

```
--mail-user=<email address>           # Specify an email address.
--mail-type=<ALL|BEGIN|END|FAIL|...> # Send an email according to the argument.
--job-name=<string>    # The name of the job.
--chdir=<dir_path>     # Change directory before executing the script.
--output=<file_path>   # Log the standard output.
--error=<file_path>    # Log the error output. Can be the same file as the standard output.
```

- More resources options (ignored on HAL)
  - Absolute options

```
--nodes=<integer>  # The total number of nodes (distributed jobs).
--ntasks=<integer> # The total number of tasks/processes (distributed jobs).
```

  - Relative options

```
--ntasks-per-node=<integer> # The number of tasks/processes executed per node (distributed jobs).
```

Batch job: sbatch script instructions

Sbatch script instructions are made up with the previous options prefixed with a hash tag (a second hash tag disables the instruction). Sbatch instructions must be located at the beginning of the slurm scripts.

```
#!/bin/bash # This instruction is not part of the sbatch instructions, but it is mandatory!
#SBATCH --partition=<string> # Set the partition. zen4 or zen16 on Spirit[X] and batch on HAL.
#SBATCH --time=<dd-HH:mm:ss> # The maximum elapsed time. Must fit the partition limit!
etc.
```

Shell environment variables automatically set by Slurm:

- `SLURM_JOB_ID`
- `SLURM_JOB_START_TIME`
- And many more!

Sbatch instruction environment variables automatically set by Slurm:

- `%j # job identifier (ex: 645486)`
- `%u # user name`
- `%x # job name`
- And more.

Batch job: submission examples 1/4

1. Submit a batch job, requesting 1 node, 1 task, 2 CPU cores and 8 Go memory (zen4) during 2 hours:

```
user@spirit1:~$ sbatch script.slurm
```

Command line options add some flexibility!

```
#!/bin/bash
#SBATCH --time=2:00:00
#SBATCH --cpus-per-task=2
# do some cool stuff!
```

2. The same, but 30 Go memory and during up to 3 hours:

```
user@spirit1:~$ sbatch --time=3:00:00 --mem=30g --partition zen16 script.slurm
```

3. Slurm script for a parallel OpenMP batch job:

```
#!/bin/bash
#SBATCH --cpus-per-task=2
module purge
module load gcc/9.4.0 openblas/0.3.17-openmp
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
cd /path/to/my/environment
./my_binary
```

Batch job: submission examples 2/4

4. Slurm script for a POSIX threads parallel batch job, example for Python:

```
#!/bin/bash
#SBATCH --cpus-per-task=2
module purge
module load pangeo-meso/2023.04.15
export PYTHONUNBUFFERED=1 # Enable real-time standard and error outputs for Python.
python my_script.py
```

5. Slurm script for a MPI distributed batch job:

```
#!/bin/bash
#SBATCH --ntasks=2
module purge
module load gcc/11.2.0 openmpi/4.0.7
mpirun my_binary
```

Batch job: submission examples 3/4

6. Slurm script for a hybrid OpenMP/MPI batch job:

```
#!/bin/bash
#SBATCH --ntasks=2
#SBATCH --cpus-per-task=2
module purge
module load intel openmpi/4.0.7 intel-mkl/2020.4.304-openmp
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
mpirun my_binary
```

Batch job: submission examples 4/4

7. Slurm script for a Deep Learning training batch job on HAL:

```
#!/bin/bash
#SBATCH --gpus=1
module purge
module load pytorch/2.1.2
export PYTHONUNBUFFERED=1
python pytorch_training_script.py
```

A complete example is available here.

Batch job: passing parameters

- Submission

```
sbatch script.slurm param1 param2
```

- Slurm script

```
#!/bin/bash
#SBATCH --output=job_%j.log # Create a log file in the current directory.
#SBATCH  --error=job_%j.log # Append error statements in the same file.

module purge
module load your_module
# The optional command line arguments of the job script are passed to your binary
($@).
my_binary $@ # Or python my_script.py $@
returned_code=$? # the returned code of the previous instruction.
echo "> script completed with exit code ${returned_code}"
exit ${returned_code}
```

Plan

- IPSL clusters
- Node architecture
- **SLURM** (Job Management) (https://documentations.ipsl.fr/spirit/common/slurm.html)
  - Introduction
  - Computing task types (serial, parallel, distributed, hybrid)
  - Job types and job submission (interactive/batch)
  - **Job array**
  - Pipeline of jobs (dependency, chaining and recursive)
  - Job practice
  - Jobs control commands
  - Jobs monitoring, statistics and best practices

Job array 1/2

Job array offers a convenient mechanism to submit a collection of similar jobs at one time. Submitted jobs differ from their "task" id (not job id!) It is up to the user job script to exploit the task id.

Submitted jobs share the same resource specifications (but don't share the same resources!)

Advantages:

- Massive job submissions made easier (useful for optimization). Keep control of max number of running jobs.
- Unique id generator that bootstraps your parameters controller (will see).
- Only one [master] job id: convenient for cancellation, monitoring, etc.
- Maximum task id is 3000 on our Slurm configuration

Range syntax:

```
sbatch --array=<first_task_id>-<last_task_id>[%max_nb_running_jobs] <slurm_script_path> [args1] [args2]
```

Task ids are integers!

`#SBATCH --array` also available!

List syntax:

```
sbatch --array=<first_task_id>[,second_task_id>][%max_nb_running_jobs] <slurm_script_path> [args1] [args2]
```

Job array 2/2

More useful shell environment variables set:

- `SLURM_ARRAY_JOB_ID` `# master job id`
- `SLURM_ARRAY_TASK_ID` `# task id (from the range or list)`
- `SLURM_ARRAY_TASK_COUNT` `# number of jobs`
- `SLURM_ARRAY_TASK_MAX` `# last job id < 3000 (which can be different from number of jobs!)`
- `SLURM_ARRAY_TASK_MIN` `# first job id`

More useful sbatch instruction environment variables set:

- `%A` `# master job id`
- `%a` `# task id`

Job array example

- Slurm script

```
#!/bin/bash
#SBATCH --array=1,3,5%20
#SBATCH --output=job_%A_%a.log
#SBATCH --error=job_%A_%a.log
echo "I'am job ${SLURM_ARRAY_TASK_ID}/${SLURM_ARRAY_TASK_MAX}, \
part of the job array ${SLURM_ARRAY_JOB_ID} that counts ${SLURM_ARRAY_TASK_COUNT} jobs"
```

- Submission

```
user@spirit1:~$ sbatch script.slurm
Submitted batch job 647464
user@spirit1:~$ ls
job_647464_1.log  job_647464_3.log<----
job_647464_5.log  script.slurm
```

```
====================TASK PROLOG==============================
Job 647466 submitted on cluster spirit from spirit1.ipsl.fr
Job 647466 submitted by user with account ipsl
Job 647466 is running with 1 nodes and 1 cores on spirit64-07
Job 647466 memory requirements are 4000 MB per node
Job 647466 starting at 2023/12/19 17:06:02
===============================================================
I'am job 3/5, part of the job array 647464 that counts 3 jobs
```

Job array mapping parameters 1/2

1. Passing task id as a parameter:

```
my_binary ${SLURM_ARRAY_TASK_ID} # or python my_script.py ${SLURM_ARRAY_TASK_ID}
```

Then exploit task id in code:

```
int main(int argc, char* argv[]) # for C
import sys ; sys.argv[1] # for Python
```

2. Parameters from a specific parameters file:

```
my_binary < parameters_${SLURM_ARRAY_TASK_ID}.in
```

3. Parameters from a specific line in a parameters file:

```
my_binary $(sed -n "${SLURM_ARRAY_TASK_ID}p" parameters.in)
```

Job array mapping parameters 2/2

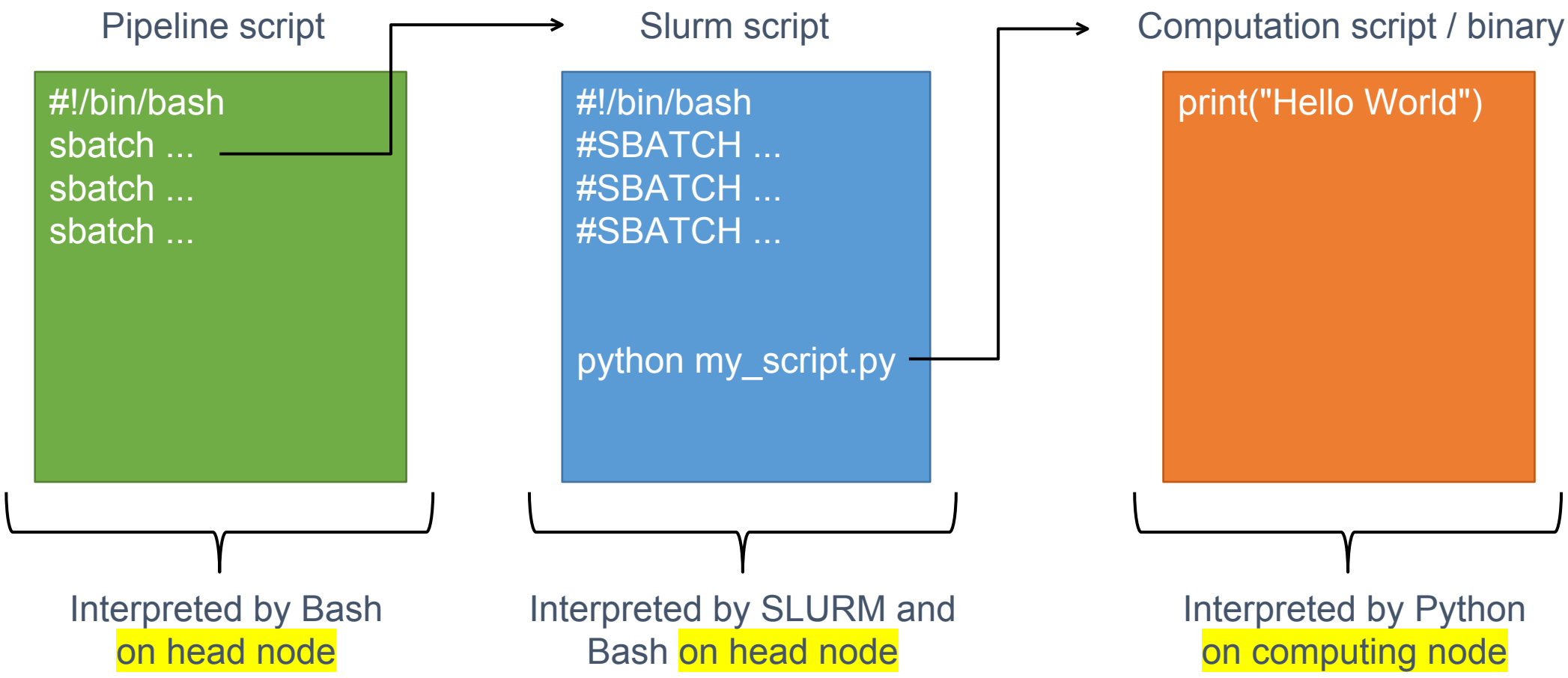
4. Fetch environment variables, example for a Python script:

```
import os

slurm_task_id = int(os.environ['SLURM_ARRAY_TASK_ID'])

# map slurm_task_id to a set of parameters (ex: Pandas's DataFrame index) and do cool
stuff.
```

Plan

- IPSL clusters
- Node architecture
- **SLURM** (Job Management) (https://documentations.ipsl.fr/spirit/common/slurm.html)
  - Introduction
  - Computing task types (serial, parallel, distributed, hybrid)
  - Job types and job submission (interactive/batch)
  - Job array
  - **Pipeline of jobs (dependency, chaining and recursive)**
  - Job practice
  - Jobs control commands
  - Jobs monitoring, statistics and best practices

Pipelines of jobs



Job dependency

Sbatch option `--dependency` enables various pipelines of jobs (graphs).

- Syntax

```
sbatch --dependency <dependency_type:job_id[:job_id][,dependency_type:job_id[:job_id]]>
sbatch --dependency <dependency_type:job_id[:job_id][?dependency_type:job_id[:job_id]]>
```

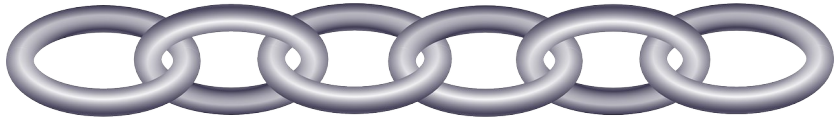
All dependencies must be satisfied if the `,` separator is used (logical AND).

Any dependency may be satisfied if the `?` separator is used (logical OR).

- Dependency types:
  - `afterok` Job starts only if the dependencies have run successfully (ex: chain of jobs and barrier).
  - `afternotok` Job starts only if the dependencies have failed (ex: cleaning/error handler jobs).
  - `afterany` Job starts only if one of the dependencies have run successfully (ex: exploratory jobs).
  - And more.

Pipeline of jobs examples 1/2

- Chain of jobs



pipeline.sh script:

```
#!/bin/bash
first_link_id=$(sbatch --parsable link_1.slurm)
second_link_id=$(sbatch --parsable --dependency=afterok:${first_link_id} link_2.slurm)
sbatch --dependency=afterok:${second_link_id} last_link.slurm
```

Script execution:

```
user@spirit1:~$ ./pipeline.sh # Don't forget to chmod +x pipeline.sh!
Submitted batch job 650560
user@spirit1:~$ squeue --me
             JOBID PARTITION     NAME  USER ST  TIME  NODES NODELIST(REASON)
            650560      zen4 last_lin  user PD  0:00      1 (Dependency)
            650559      zen4 link_2.s  user PD  0:00      1 (Dependency)
            650558      zen4 link_1.s  user  R  0:01      1 spirit64-02
```

Pipeline of jobs examples 2/2

- Job barrier

pipeline.sh script:

```
#!/bin/bash
job_id_0=$(sbatch --parsable job_1.slurm)
job_id_1=$(sbatch --parsable job_2.slurm)
sbatch --dependency=afterok:${job_id_0}:${job_id_1} barrier.slurm
```

Script execution:

```
user@spirit1:~$ ./pipeline.sh # Don't forget to chmod +x pipeline.sh!
user@spirit1:~$ squeue --me
             JOBID PARTITION     NAME  USER ST  TIME  NODES NODELIST(REASON)
            650603      zen4 barrier.  user PD  0:00      1 (Dependency)
            650601      zen4 job_1.sl  user  R  0:03      1 spirit64-03
            650602      zen4 job_2.sl  user  R  0:03      1 spirit64-01
```

Recursive job

Recursive job: resubmit the same job inside a job with `sbatch`

This could cause problem if new job start before a complete end of the precedent, you must know that there is a system epilog at the end of your job causing delay

- To achieve this, the best way is :
  - sbatch must be the last command of your job
  - Run the next job only if the current is finished ok

The next command ensure the current job is really terminated before starting the next one, SLURM_JOB_ID variable contain the current jobid

```
sbatch --dependency=afterok:${SLURM_JOB_ID} script.slurm
```

Plan

- IPSL clusters
- Node architecture
- **SLURM** (Job Management) (https://documentations.ipsl.fr/spirit/common/slurm.html)
  - Introduction
  - Computing task types (serial, parallel, distributed, hybrid)
  - Job types and job submission (interactive/batch)
  - Job array
  - Pipeline of jobs (dependency, chaining and recursive)
  - **Job practice**
  - Jobs control commands
  - Jobs monitoring, statistics and best practices

Job practice 1/3

- Create a Job Array slurm script accepting two arguments *year month* and running a task for each day of month

- Create the same job using only one multitask job without job array

Job practice 2/3

- Create a Job Array slurm script accepting two arguments *year month* and running a task My_Prog for each day of month

```
#!/bin/bash
#SBATCH --array 1-31
# fist ARG is $1 in shell
# second arg is $2 ....

if [ $# -ne 2 ];then # $# is number of argument
  echo $0 YYYY MM   #$0 is the name of the script
  exit 1
fi

YEAR="$1"
MONTH="$2"
DAY=${SLURM_TASK_ID} # problem DAY is 1 but i need 01
DAY=$(printf "%02d" ${SLURM_TASK_ID})
FILE=/bdd/dataset/truc/${YEAR}/${MONTH}/${DAY}/xx${YEAR}${MONTH}${DAY}.dat

if [ -r $FILE ]; then
 time My_prog $FILE
fi
```

Job practice 3/3

- Create one Multitask Job slurm script accepting two arguments year month and running a task My_Prog for each day of month

```
#!/bin/bash
#SBATCH --ntasks 31 # number of task must be the same as number of processes launched in background
# fist ARG is $1 in shell
# second arg is $2 ....

YEAR="$1"
MONTH="$2"
for I in {01..31} # work in bash >=4
do
  DAY=${I}
  FILE=/bdd/dataset/truc/${YEAR}/${MONTH}/${DAY}/xx${YEAR}${MONTH}${DAY}.dat
  if [ -r $FILE ];then
    time My_prog $FILE & # launch process in background and continue
  fi
done

wait # Waiting all background processes are finished
```

Plan

- IPSL clusters
- Node architecture
- **SLURM** (Job Management) (https://documentations.ipsl.fr/spirit/common/slurm.html)
  - Introduction
  - Computing task types (serial, parallel, distributed, hybrid)
  - Job types and job submission (interactive/batch)
  - Job array
  - Pipeline of jobs (dependency, chaining and recursive)
  - Practical
  - **Jobs control commands**
  - Jobs monitoring, statistics and best practices

Job control commands 1/2

- Information on nodes

`check-cluster`: homemade command that displays the cluster load and status

`sinfo`: display node and partition information

```
user@spiritx1:~$ sinfo -l
```

`pestat` : display job placement on nodes

```
user@spiritx1:~$ pestat -u user -E
```

- Information on job

`scontrol show job <job_id>`: display detailed job information

```
user@spiritx1:~$ scontrol show job xxxxxxx
```

- Information on partition

`scontrol show partition <partition_name>`: display detailed partition information

```
user@spiritx1:~$ scontrol show partition zen4
```

Job control commands 2/2

- Homemade wrapper command using squeue to display running and pending jobs information:

```
slqueue [-h|--help] [-r|--running] [-i|-b|-p|--pending] [-m|--me] [-u user|--user user]
[-w hostname|--host hostname]
```

```
user@hal:~$ slqueue
```

```
 JobId   State Partition    Username Account Node CPUs Memory Batch    TimeLeft   TimeLimit  Node/Reason
 10079 RUNNING     batch    eremenko    lisa    1    4    60G    1  2-01:18:07  3-00:00:00  hal6
 10101 RUNNING interacti    vsambath     lmd    1    4  7.50G    0     9:30:16    10:00:00  hal4
```

- Cancel job(s):

```
scancel <job_id> [<job_id>]
```

```
user@hal:~$ scancel 10095 10345
```

SLURM job pending reason

“Why my job does not start and stay in pending state”, main reasons:

```
slqueue --me --pending
```

Resources: there is not enough free resources on cluster to run your job, but you’re in first priority

Priority: there is not enough resources on cluster to run your job and you are not in first priority

QOSMaxCpuPerUserLimit: your running jobs hit the max number of CPU permitted for your account

QOSMaxJobsPerUserLimit: your running jobs hit the max number of job permitted for your account

QOSMaxMemoryPerUser: your running jobs hit the max sum of memory permitted for your account

ReqNodeNotAvail, Reserv: there is a reservation on cluster made by admin team for special use or maintenance and with your *TimeLimit* request, your job cannot be finished before the beginning of the reservation. For example you have requested 7 days and we have a maintenance in 3 days with full stop

```
scontrol show reservation
```

Dependency: you have asked dependency for this job

SLURM job end status

Possible status after job end:

COMPLETED: all seem to be ok
FAILED: non zero error code at the end
TIMEOUT: asked time limit has been reached
CANCELLED: your job has been canceled (by you, admin or job system)

Plan

- IPSL clusters
- Node architecture
- **SLURM** (Job Management) (https://documentations.ipsl.fr/spirit/common/slurm.html)
  - Introduction
  - Computing task types (serial, parallel, distributed, hybrid)
  - Job types and job submission (interactive/batch)
  - Job array
  - Pipeline of jobs (dependency, chaining and recursive)
  - Practical
  - Jobs control commands
  - **Jobs monitoring, statistics and best practices**

SLURM jobs monitoring and statistics

Monitoring job during run

Possibility to connect (ssh) to the node only if you have running jobs on this node: top, htop

MEMORY to use with SLURM in top or htop is in RES column (not in VIRT)

MultiThreads program can be identified with CPU more than 100%

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
232285	xxxx	20	0	8093392	7.4g	18428	R	768.1	2.9	2140:47	myjob

NEVER use ssh to a node to do other things than monitoring job

All commands via ssh are shared with your job resources and admin can revoke your account in abusive case

```
sstat -a -j <jobid> # Give near real-time statistics but not easy to read
```

SLURM jobs monitoring and statistics

Statistics for endded job: jobreports, seff

```
user@spirit1:~$ jobreports --help
 Usage: jobreports [--help] [-c|--color] [-l|--long] [-v|--verylong] [-u user|--user user] [--
partition Partition_name] [-d nb_days|-w nb_weeks|-h nb_hours]
 --help : print this help
 -l or --long : more detail on each job
 -v or --verylong : maximum detail on each job
 -p or --parsable : fields are delimited by '|'
 -c or --color : job efficacity  is colored : green good, black ok, red bad
 --partition partition_name : select job only in this  partition
 -h nb_hours or --hour nb_hours : only get jobs since nb_hours ( nb_hours between 1 and 24 )
 -d nb_days or --day nb_days : only get jobs since nb_days ( nbdays between 1 and 31 )
 -w nb_weeks or --week nb_weeks : only get jobs since nb_weeks ( nb_weeks between 1 and 5 )
 -u user or --user user : job for this user
```

Job statistics samples

Good option to start with jobreports -c -l give all your finished jobs on one week:

```
user@spirit1:~$ jobreports -c -l
```

JobID	State	Elapsed	TimeEff	CPUEff	MemEff	Partition	CPUTime	Timelimit	MaxRSS	ReqMem	Nodes	Tasks	CPUS	Start	
1497224	TIMEOUT	7-00:00:23	100.0%	---	76.8%	zen4	224-00:12:16	7-00:00:00	92.12G	120Gn	1	1	32	2024-11-11T10:15:17	
More than 100% using threads in the core															
1521419	COMPLETED	02:42:50	1.6%	141.8%	24.0%	zen4	03:50:55	7-00:00:00	0.90G	3.75Gc	1	1	1	2024-11-22T22:38:00	
Very good job															
1448720	COMPLETED	22:04:00	60.0%	99.9%	66.1%	zen4	44-02:31:52	1-12:45:00	63.48G	96Gn	1	1	48	2024-10-21T21:38:18	
Many core low memory on high memory node															
5996075	COMPLETED	1-13:20:30	51.9%	99.5%	35.7%	zen16	74-07:35:40	3-00:00:00	46.46G	130Gn	1	1	48	2024-11-15T03:55:53	
Many very short job per day															
6033026	COMPLETED	00:00:05	0.0%	20.0%	0.0%	zen4	00:02.571	03:00:00	0.00G	4Gn	1	1	2	2024-11-26T17:33:34	
10 Hours interactive job 100G mem and 4 minutes cpu !!!!															
1507034	TIMEOUT	10:00:01	100.0%	0.1%	0.1%	zen16	04:07.820	10:00:00	0.07G	100Gn	1	1	7	2024-11-15T09:41:59	bash
10G de Ram 3 core ELAPSED > CPUTIME ==> sequential Job															
1521051	COMPLETED	00:36:06	5.0%	24.8%	3.5%	zen4	26:54.589	12:00:00	0.35G	10Gn	1	1	3	2024-11-22T18:51:36	

Job reports

`sreport` is used to get cluster usage on a time period

```
user@spirit1:~$ sreport -t HourPer cluster AccountUtilizationByUser Users=$USER
Start=2021-09-01 End=2025-01-01
--------------------------------------------------------------------------------
Cluster/Account/User Utilization 2021-09-01T00:00:00 - 2024-12-03T06:59:59 (102758400 secs)
Usage reported in CPU Hours/Percentage of Total
--------------------------------------------------------------------------------
  Cluster         Account     Login     Proper Name                Used            Energy
--------- --------------- --------- --------------- ---------------- ----------------
   spirit            xxxx    aaaaaa     Dave Null           891(0.00%)          0(0.00%)
```

Job best practices 1/2

- Do not launch many very short batch < 1 min: launching job has a cost to start, to record statistics
- Sequential high memory job > 4Gb ⇒ always in zen16 partition
- No low memory job and many cores in zen16 partition ⇒ always in zen4
- No long interactive job without effective use ⇒ big resources waste
- Jupyter batch job is often like long interactive job without use: best to connect to https://jupyter-x.ipsl.fr or https://jupyter-su.ipsl.fr
- Try to limit your job time to a maximum of 24h ⇒ better placement on node

Job best practices 2/2

- No only CPU jobs on Hal GPU nodes ⇒ do CPU jobs on spirit[X] then only GPU job on HAL
- Long low memory job with a high memory end (data recombination) ⇒ launch the end in a different job in zen16 partition with dependency
- High memory job is sometime only memory leak in your code
- Warn with very high I/O job : sample 64 jobs doing `tar xvf` can kill any clustered file system

MERCI - THANK YOU

Institut Pierre Simon Laplace IPSL

IPSL support

Need Help ?

Start to read here: https://documentations.ipsl.fr/spirit/getting_started/support.html

- Documentations :

https://documentations.ipsl.fr/

- Email support :

meso-support@ipsl.fr

provide information:

- login
- Cluster name (and nodename if necessary)
- Submission command used for job