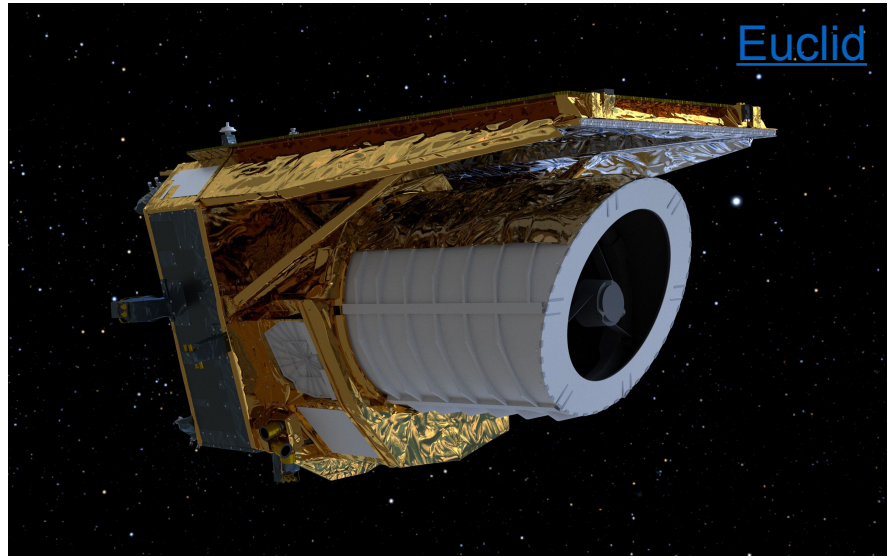
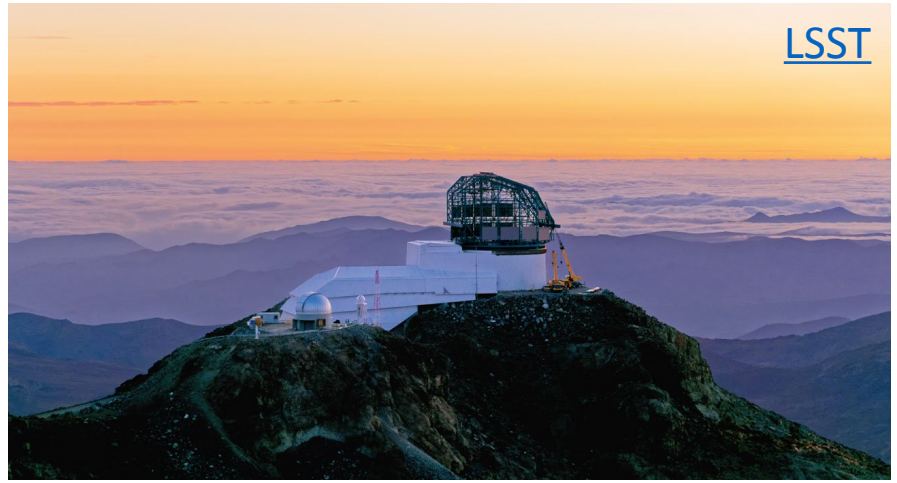
Formation Slurm

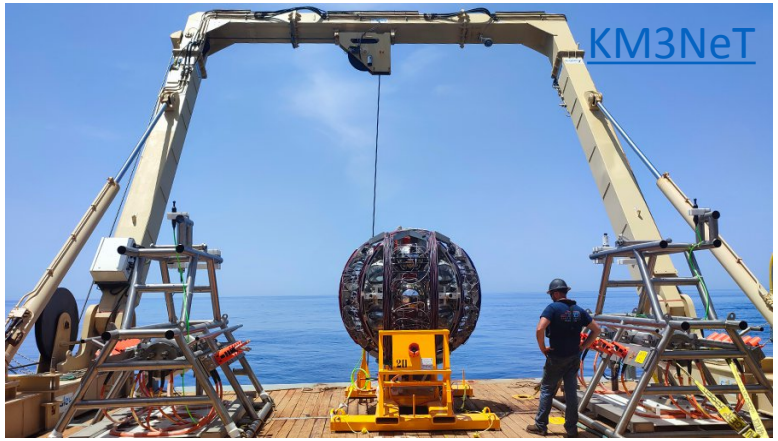
Utilisation de Slurm au CC-IN2P3

27 novembre 2025

L'IN2P3 : collaborations internationales







Le CC-IN2P3

- Pour la partie calcul, le CC possède deux clusters :
  - Grille de calcul
    - Une dizaine d'utilisateurs ("grandes" expériences type Atlas ou Alice)
    - Ordonnanceur : HTCondor
    - 765 workers pour 37 000 CPUs
    - ~50 000 jobs « standardisés » par jour
  - Ferme locale
    - Plusieurs centaines d'utilisateurs de l'IN2P3 répartis en plusieurs dizaines de groupes
    - Ordonnanceur : Slurm (anciennement UGE/Grid Engine)
    - 327 workers pour 25000 CPUs et 80 GPUs
    - ~120 000 jobs variés par jour

Plan

- Introduction : le calcul scientifique
  - Vocabulaire et notions importantes
  - Présentation d'ordonnanceurs
- Slurm
  - Présentation
  - Architecture et fonctionnement
- Utilisation de Slurm
  - Prise en main du cluster
  - Soumission et suivi de jobs
  - Fonctionnalités de base

LE CALCUL SCIENTIFIQUE

Le calcul scientifique : les ressources

- Ressources matérielles et parfois logicielles
  - CPUs
  - Mémoire
  - GPUs
  - Features et licences
  - Workers
- Temps
- Job : demande de ressources de la part d'un utilisateur et exécution d’un code utilisant les ressources allouées
  - Batch
  - Interactif

Le calcul scientifique : familles

- HPC : *High Performance Computing*
  - Ressources utilisées intensivement pendant un temps relativement court : calculs massivement parallélisés où le nombre d'opérations par seconde est un facteur important
  - Set de tâches qui peuvent être interdépendantes
  - Exemple : calcul d'un modèle météo, apprentissage pour IA

- HTC : *High Throughput Computing*
  - Ressources utilisées sur un temps plus long : la capacité de calcul et l'efficacité sur plusieurs mois sont des facteurs importants
  - Gros volume de tâches indépendantes mais demandant individuellement moins de ressources
  - Exemple : séquençage ADN, traitement de données

- Il existe plusieurs autres manières de les différencier
  - CPU bound VS memory/IO bound
  - Capacity VS capability

- Ces deux modes représentent les deux bouts d’un spectre

RJMS

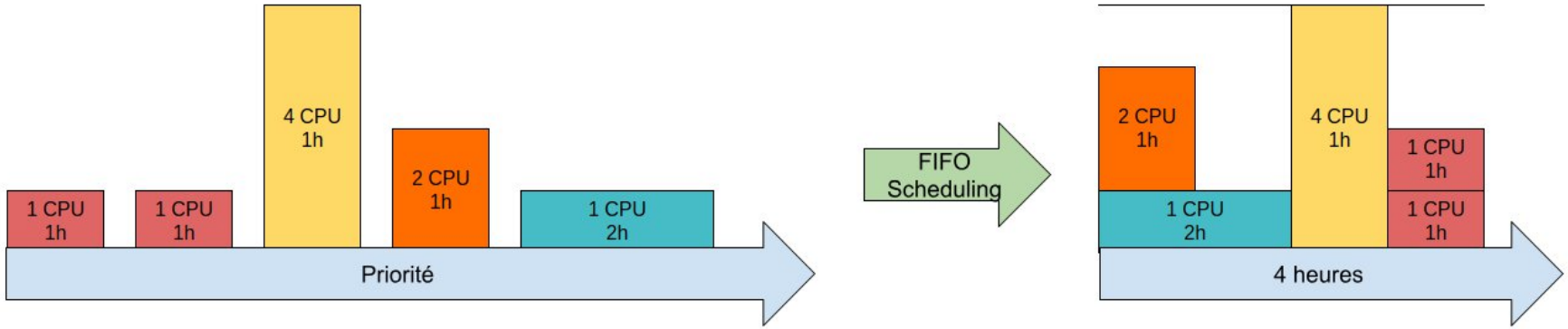
- Plusieurs noms : batch system, job scheduler, ordonnanceur
  - Historiquement des différences, mais par abus de langages tous sont utilisés
  - Resource and Job Management System (RJMS) : outil proposant à la fois la gestion des ressources et l'ordonnancement des jobs

- Principales fonctionnalités
  - **Gestion et allocation des ressources**
  - **Ordonnancement de jobs** et gestion des files d'attente
  - Gestion des utilisateurs
  - Gestion du cycle de vie des jobs

- Quelques systèmes de batch
  - Slurm
  - Altair Grid Engine (anciennement SGE puis UGE, maintenant HPC Gridware)
  - IBM LSF
  - HTCondor
  - Torque + Maui

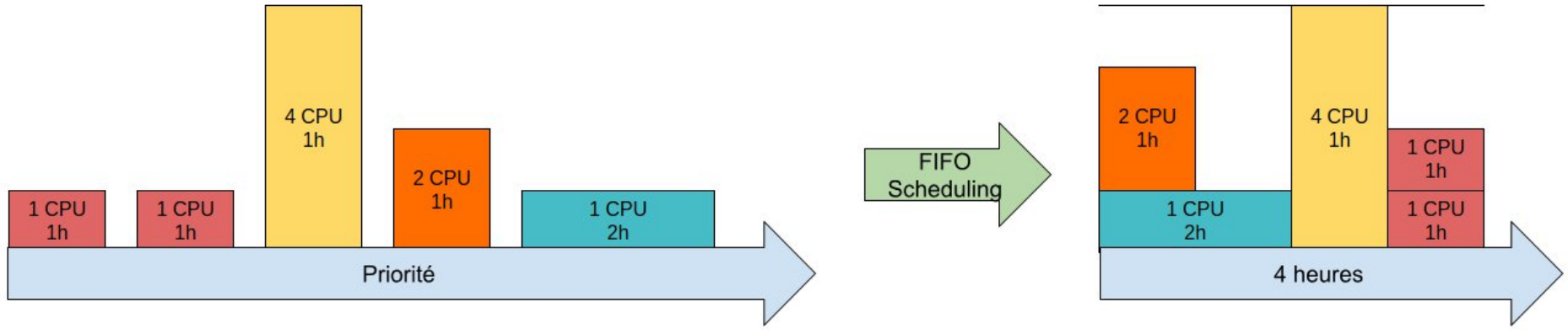
Le scheduling

- Scheduling / ordonnancement : allocation des ressources aux jobs présents en queue



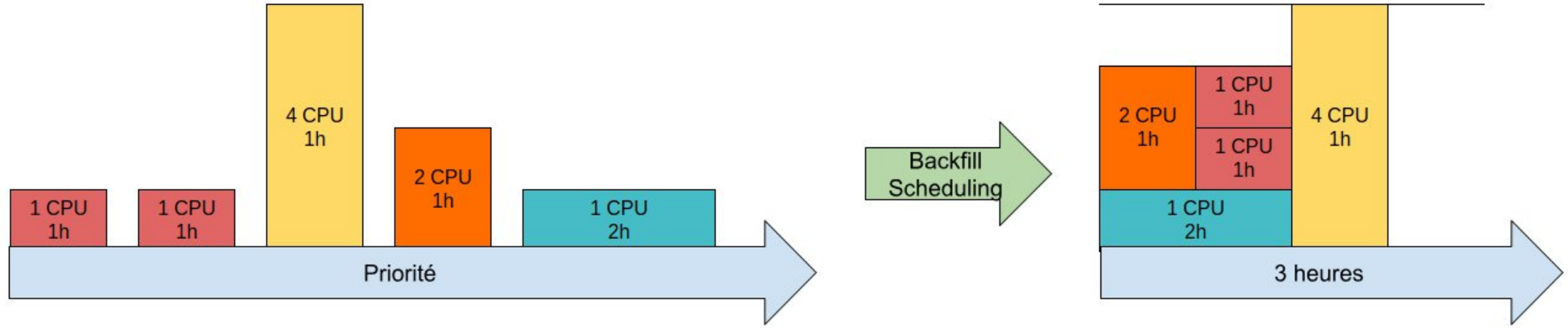
FIFO scheduling

- FIFO scheduling : les jobs rentrent dans l'ordre de priorité
  - "Algorithme" de scheduling le plus simple et le plus rapide d'exécution
  - Pas efficace pour l'utilisation des ressources



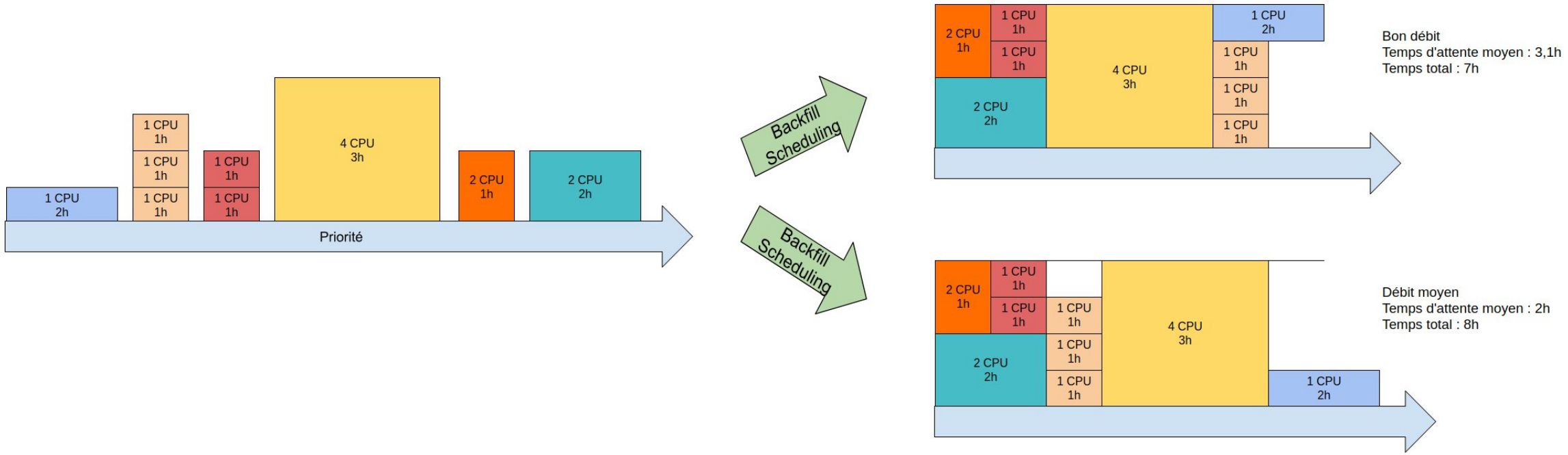
Backfill scheduling

- Backfill scheduling : prise en compte des jobs en queue pour "boucher les trous"
  - Permet d'optimiser l'utilisation des ressources
  - Algorithme de scheduling plus complexe et plus lent



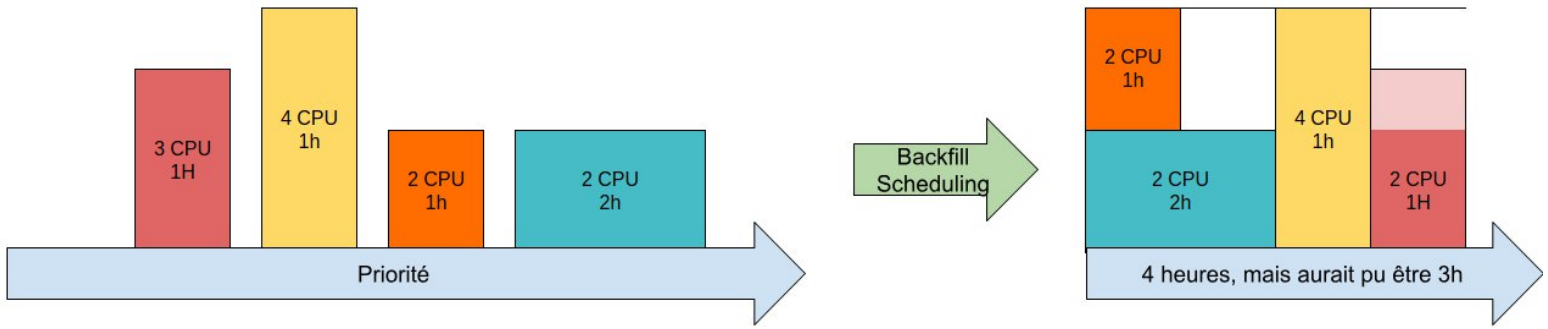
Scheduling : optimisation

- Plusieurs métriques peuvent être utilisées pour mesurer l'efficacité d'un cluster
  - Utilisation CPU (usage orienté HPC)
  - Débit (usage orienté HTC)
  - Temps d'attente moyen
- Ces métriques ne sont pas toujours compatibles entre elles



Scheduling : optimisation

- Pour passer plus rapidement, il est important que la demande d'allocation corresponde aux ressources réellement nécessaires
  - Les utilisateurs ont tendance a beaucoup surestimer certaines ressources (temps, mémoire)



Multi-threading et multi-processing

- Deux méthodes de parallélisation pour augmenter la rapidité d'exécution d'une application
  - Overhead : pas forcément adaptées à des tâches trop courtes

- Multi-threading : exécution de plusieurs threads au sein d'un seul processus
  - Chaque thread partage le(s) cœur(s) et la mémoire du processus
  - Les threads sont donc concurrents entre eux plutôt que réellement parallèles
  - Utile pour les applications I/O bound

- Multi-processing : lancement de plusieurs processus parallèles par un processus parent
  - Chaque processus a ses propres ressources CPU et mémoire
  - Les processus sont donc exécutés réellement en parallèle
  - Utile pour les applications CPU bound

Autres définitions

- **MPI** (*Message Passing Interface*) : protocole permettant à des processus parallèles de communiquer entre eux
- **InfiniBand** : protocole de communication à haut débit et faible latence, notamment utilisé entre différents nœuds d'un cluster HPC
- **GPGPU** (*General-Purpose Processing on Graphics Processing Units*) : utilisation de GPU pour réaliser des tâches massivement parallèles sans lien avec le rendu graphique
- **CUDA** : plateforme de calcul parallèle développée par NVIDIA, principalement utilisée pour le GPGPU

Slurm

Slurm

- Slurm (~~Simple Linux Utility for Resource Management~~) est un système de gestion de ressources et d'ordonnancement de jobs

- Version 1.0 publiée en 2002 par le Lawrence Livermore National Laboratory (Californie)
  - D'abord ne faisant que la gestion de ressources, l'ordonnancement n'a été ajouté qu'en 2010
  - Support créé en 2008
  - SchedMD créé en 2010 et est maintenant responsable du dévelopement et du support

- Développé en C

- Produit libre et gratuit
  - Licence GPL
  - Modèle économique reposant sur le support payant et le développement sponsorisé de fonctionnalités

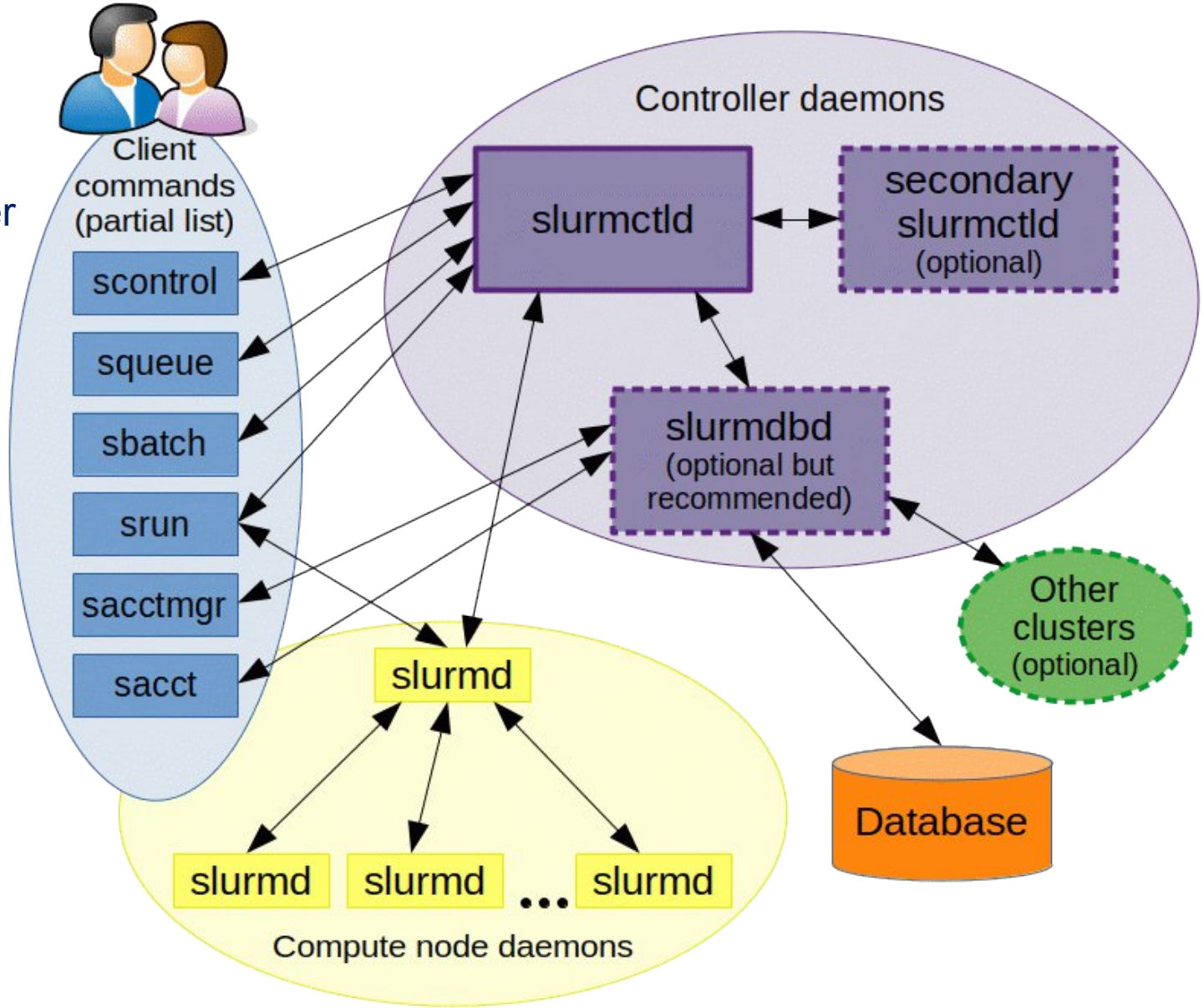
- Utilisé par 65% du TOP500

Quelques caractéristiques

- Stabilité et résistance aux pannes
- Rapidité : jusqu'à 1000 jobs soumis et 500 lancés par seconde
- Gestion fine des limites et du calcul des priorités
- CPU bindings et allocation stricte des ressources
- Facile à administrer
  - Défaut de ses qualités : système centralisé (controller, stockage)
- Communauté active

Architecture

Clients :
- CCA
- Toutes les machines du cluster
- JNP
- ARC-CE
- etc.



Commandes de base

- Jobs
  - `sbatch` : soumission de jobs
  - `scancel` : annulation de jobs
  - `scontrol` : modification de jobs

- Suivi de job
  - `squeue` : informations sur les jobs en cours
  - `sacct` : informations sur les jobs en cours ou finis (via la BDD)
  - `sprio` : informations sur les priorités de jobs

- Cluster
  - `sinfo` : informations de base sur les workers ou les partitions
  - `scontrol` : informations détaillées sur les workers, les partitions, etc. Administration du cluster.
  - `sacctmgr` : informations et modifications pour les users/groups, les QoS, etc.

Partitions

- Les partitions sont un regroupement de machines entre elles
  - Regroupement :
    - Par type de machines : « classiques », GPU, etc.
    - Par usage / réservation
  - Une machine est obligatoirement dans une partition, et peut être dans plusieurs partitions
  - Un job est obligatoirement soumis sur une partition, et peut être soumis sur plusieurs partitions

- Au CC :
  - Une petite dizaine de partitions
  - Deux grandes familles :
    - CPU
    - GPU
  - Deux types de partitions :
    - Batch
    - Interactif

Partitions

	Ressources			Règles			
Partition	**Workers**	**CPUs**	**Mémoire (Go)**	**Jobs**	**CPUs/job**	**Mem/job (Go)**	**Timelimit**
htc	268	21312	87344		112	150	7 jours
flash	1	64	192	10	64	150	1 heure
htc_interactive	2	128	320		64	150	7 jours
htc_daemon	1	64	192	10	3	16	90 jours
htc_highmem	2	168	2826		112	1200	7 jours
htc_arm	4	1024	3080		256	770	7 jours
hpc	8	512	10312		512	10312	7 jours

	Ressources			Règles			
Partition	**Workers**	**GPUs**	**Mémoire (Go)**	**Jobs**	**CPUs/GPU**	**Mem/GPU (Go)**	**Timelimit**
gpu_v100	16	64	87344	100	5	48	7 jours
gpu_v100_interactive	2	8	192		5	48	7 jours
gpu_h100	3	12	320		12	387	7 jours
gpu_h100_interactive	1	4	192	100	12	387	7 jours

Licences

- Les licences permettent de décompter les jobs demandant une ressource spécifique
  - Ce sont donc des ressources à allouer au même titre que les CPUs ou la mémoire
  - Si toutes les licences sont prises par des jobs en cours, alors un job en demandant une restera en attente
- Les licences peuvent avoir plusieurs utilités
  - Limiter les accès simultanés à une autre ressource (par exemple un espace de stockage)
  - Licences logicielles (par exemple Matlab)

- `scontrol show licenses` : liste les licences et leur usage

```
$ scontrol show licenses
LicenseName=matlab_default
    Total=150 Used=2 Free=148 Reserved=0 Remote=no

LicenseName=sps_default
    Total=15000 Used=5770 Free=9230 Reserved=0 Remote=no
```

Accounts et users

- Account : groupe d'utilisateurs déclaré dans Slurm
  - Possède des caractéristiques et des limites qui vont s'appliquer à l'ensemble des utilisateurs attachés à cet account
  - Fait partie d'une arborescence d'accounts : la hierarchie permet d'appliquer des limites aux accounts enfants

- User : utilisateur déclaré dans Slurm
  - Doit exister sur la machine de soumission et sur les workers (même uid)
  - Peut être attaché à plusieurs accounts
  - L'utilisateur ne peut avoir d'informations sur les accounts auxquels il appartient, en revanche il peut connaître ses propres limites.

- Slurm utilise les associations account+user
  - user01/group_a et user02/group_b sont deux « personnes » différentes pour Slurm

```
$ sacctmgr show user user01 withassoc format=User,Admin,Cluster,Account,Partition,Share,Priority,MaxJobs,MaxCPUs,QOS
      User     Admin    Cluster    Account  Partition     Share   Priority MaxJobs  MaxCPUs                  QOS
---------- --------- ---------- ---------- ---------- --------- ---------- ------- -------- --------------------
    user01      None    cluster    group_a                    1                                highprio,normal
    user01      None    cluster    group_b                    1                                highprio,normal
    user01      None    cluster    group_c                    1                                highprio,normal
```

Accounts

- Un account a des limites qui s’appliquent à tous ces utilisateurs. Pour Humanum :
  - 4000 CPUs running
  - 13T mémoire running
  - GPUs autorisés

Job states

- Un job a nécessairement un état
- Les états les plus courants sont :
  - **PENDING** (PD): le job a été soumis mais pas encore démarré. Il est en file d'attente avant que des ressources lui soient allouées.
  - **RUNNING** (R): le job est en cours d'exécution.
  - **COMPLETING** (CG): le job est en train de se terminer. Cet état est transitoire et correspond au moment entre la fin du calcul et la libération des ressources (fin des I/O, nettoyage de fichiers, etc.)
  - **COMPLETED** (CD): le job s'est terminé sans erreur.
  - **CANCELLED** (CA): le job a été annulé, soit par l'utilisateur soit par un administrateur.
  - **FAILED** (F): le job a eu un code retour différent de 0.
  - **TIMEOUT** (TO): le job a dépassé le temps qui lui était alloué et a été tué.
  - **NODE_FAIL** (NF): le job s'est coupé suite à une erreur de l'un des nœuds d'exécution.
  - **OUT_OF_MEMORY** (OOM): le job a dépassé la mémoire qui lui était allouée et a été tué.

- Liste exhaustive des états possibles

Pending jobs

- Lorsque le job est pending, une *reason* lui est attribué
- Parmi ces raisons on trouve :
  - Raisons classiques
    - **Ressources** : le job rentre dès que les ressources demandées sont disponibles
    - **Priority** : le job a une priorité trop basse par rapport à d'autres jobs et n'est pas encore pris en compte par le scheduler
    - **BeginTime** : le job a été soumis avec une date d'exécution dans le futur
    - **Dependancy** : le job est dépendant d'un autre job qui ne s'est pas encore terminé
  - Limites
    - **AssocGrp*Limit** : l'account a atteint la limite d'utilisation instantanée d'une ressource (CPU, mémoire, etc.)
    - **AssocMax*Limit** : le job dépasse la limite par job d'une ressource (CPU, mémoire, etc.)
    - **JobArrayTaskLimit** : le job array est trop grand pour passer en une fois
  - Raisons bloquantes (intervention nécessaire)
    - **BadConstraints** : le job demande des choses non compatibles avec le cluster (par exemple une QoS non disponible sur une partition)
    - **DependencyNeverSatisfied** : le job dépendait d'un autre job qui a terminé en erreur
    - **QOSMaxMemoryPerJob** : le job demande trop de mémoire

- Liste exhaustive des reasons

Priorités

- Un job en queue a une priorité
- Slurm choisit les jobs avec les plus hautes priorités pour les faire rentrer en machine en premier
  - Avec le backfill scheduling, ce n'est pas forcément le job avec la plus haute priorité qui sera le premier

- Plusieurs commandes permettent de consulter les priorités des jobs en queue :

```
$ squeue -O jobid,prioritylong,state
JOBID                PRIORITY             STATE
59                   2524                 PENDING
58                   1683                 PENDING
57                   1525                 PENDING

$ sprio
          JOBID PARTITION   PRIORITY       SITE        AGE  FAIRSHARE  PARTITION        QOS
             57 main            1525          0        341        184       1000          0
             58 main            1683          0        341        342       1000          0
             59 main            2524          0        341        184       1000       1000
```

Priorités

- De multiples facteurs influencent la priorité finale
  - Priorités sur la partition, la QoS
  - Fairshare de l'account, de l'utilisateur
  - Taille du job
  - Âge du job
  - Historique de soumission de l'account ou de l'utilisateur
- Chacun de ces facteurs a un poids fixé par l'administrateur du cluster

```
$ sprio
          JOBID PARTITION   PRIORITY       SITE        AGE  FAIRSHARE  PARTITION        QOS
             57 main            1525          0        341        184       1000          0
             58 main            1683          0        341        342       1000          0
             59 main            2524          0        341        184       1000       1000
```

- Documentation

Job terminé, failed/completed ?

- Slurm se base sur le code retour du script pour déterminer si le job est failed ou completed
- Par défaut, Bash ne s'arrête pas en cas d'erreur
  - Il faut indiquer `set -e` pour quitter en cas d'erreur

```
$ cat script.sh
#!/bin/bash
echo "start"
hostname
echooooooooooo
```

Failed

```
$ cat script.sh
#!/bin/bash
echo "start"
echooooooooooo
hostname
```

Success!

```
$ cat script.sh
#!/bin/bash
set -e
echo "start"
echooooooooooo
hostname
```

Failed

Cause d'un job failed

- Un job failed est souvent dû à un problème côté utilisateur (ouverture d'un fichier qui n'existe pas, erreur non gérée, etc.)
- Mais ça peut être causé par des problèmes sur le cluster (problèmes d'I/O, de réseau, etc.)

Bonus : bonnes pratiques pour Bash

- Au début du script Bash il est possible de préciser comment il doit se comporter en cas d'erreur

```
set -o errexit   # abort on nonzero exitstatus
set -o nounset   # abort on unbound variable
set -o pipefail  # don't hide errors within pipes

# Équivalent
set -euo pipefail
```

- `set -x` : mode debug (chaque commande est écrite dans stdout)

- Man page
- Bonne explication

Job environment

- Lorsqu'un job est soumis, l'environnement de l'utilisateur sur la machine de soumission est envoyé avec le job sur le worker
  - L'option --export permet de choisir quelles variables envoyer
- Slurm rajoute des variables d'environnements qui peuvent être utilisées dans le job
  - Documentation

Soumettre un job

- Informations requises lors de la soumission d'un job
  - Ressources : CPUs, mémoire, node…
  - Durée
  - Partition
  - Script bash à exécuter (commençant par `#!/bin/bash`)

```
$ sbatch -t 1:00:00 --mem 3G --cpus-per-task 1 -p htc script.sh
```

```
$ sbatch -t 1:00:00 --mem-per-cpu 3G --cpus-per-task 2 --ntasks 2 -p htc script.sh
```

Soumettre un job

- Deux possibilités pour paramétrer un job
  - En argument de commande `sbatch`
  - En directive `#SBATCH` dans le script a exécuter (⚠ syntaxe)
- Les deux se valent, mais les commentaires permettent de conserver facilement un historique des paramètres de soumission

```
$ sbatch -t 1-1:00:00 --mem 3G -c 2 -p htc script.sh
```

```
$ cat script.sh
#!/bin/bash
#
#SBATCH -c 2
#SBATCH --mem 3G
#SBATCH --partition=htc
#SBATCH --timelimit=1-1:00:00
[…]

$ sbatch script.sh
```

⚠ Les paramètres de la commande `sbatch` surchargent les directives `#SBATCH`

Soumettre un job

- Lorsqu'on utilise les arguments en ligne de commande, il faut terminer par le script
  - Toutes les options situées après le script seront transmises au script

```
# Ok
$ sbatch -t 1-1:00:00 --mem 3G -c 2 -p htc script.sh
$ sbatch -t 1-1:00:00 --mem 3G -c 2 -p htc script.sh script_arg

# No ok
$ sbatch -t 1-1:00:00 -p htc script.sh --mem 3G -c 2
```

Exemple de job

- Job

```
$ cat simplescript.sh
#! /bin/bash

echo $HOSTNAME
sleep 60
```

- Pas de step explicite
  - Les 3 commandes seront incluses dans la step créée par défaut
  - Cette step par défaut d'appelle `JOBID.batch`

Suivi d'un job

Soumission

```
$ sbatch -t 1-1:00:00 --mem 3G -c 2 -p htc simplescript.sh
sbatch: INFO: Time limit set to: 0-01:00 (60 minutes)
Submitted batch job 16364109
```

Suivi du job en cours

```
$ squeue -j 16364109
   JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
16364109       htc simplesc gcochard PD       0:00      1 (Resources)

$ squeue -j 16364109
   JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
16364109       htc simplesc gcochard  R       0:05      1 ccwslurm0059
```

Affichage du job terminé

```
$ sacct -j 16364109
JobID           JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
16364109     simplescr+        htc    ccin2p3          1  COMPLETED      0:0
16364109.ba+      batch               ccin2p3          1  COMPLETED      0:0
```

Job GPU

Soumission

```
$ sbatch -t 1-1:00:00 --mem 3G -c 2 -p gpu_h100 –gpus 2 simplescript.sh
sbatch: INFO: Time limit set to: 0-01:00 (60 minutes)
Submitted batch job 16364109
```

squeue et filtrage

- `squeue` possède de nombreux filtres, parmi lesquels :
  - `-t <pd|r>` : ne liste que les jobs *pending* (pd) ou *running* (r)
  - `-w <worker>` : ne liste que les jobs ayant au moins une step sur le worker
  - `-p <partition>`
  - `-u <user>` et `-A <account>`
- Il est aussi possible de modifier les informations affichées

```
$ squeue -O jobid,state,partition,numcpus,minmemory -j 16366866
JOBID               STATE               PARTITION           CPUS                MIN_MEMORY
16366866            RUNNING             htc                 1                   3G

$ squeue -o %i,%T,%P,%C,%m -j 16366866
JOBID,STATE,PARTITION,CPUS,MIN_MEMORY
16366866,RUNNING,htc,1,3G
```

- Documentation

sacct et filtrage

- `sacct` possède moins de filtres que `squeue`, et les filtres équivalents n'ont pas forcément les mêmes arguments
  - `-s <cd,f>` : ne liste que les jobs dans l'état correspondant (liste des états)
  - `-N <worker>` : ne liste que les jobs ayant au moins une step sur le worker
  - `-u <user>` et `-g <group>`
- Pour rechercher les jobs terminés, il est nécessaire de préciser l'intervalle de temps dans lequel la recherche doit s'exécuter
  - `sacct -S "now-1h" -E "now"` affiche les jobs ayant eu un changement d'état depuis 1h
- Il est aussi possible de modifier les informations affichées

```
$ sacct -j 30 -o jobid,user,account,nodelist,elapsed
JobID             User    Account        NodeList    Elapsed
------------ --------- ---------- --------------- ----------
30             testuser    group_a        slurm-02   00:01:07
30.batch                   group_a        slurm-02   00:01:07
```

- Documentation

Place au TP

TP - Ressources

- Récupérer le repo gitlab « huma-num »

```
$ git clone https://gitlab.in2p3.fr/ccin2p3-support/formations/workshops-gpu/2025.11/huma-num.git
```

TP - Job CPU

- Soumettre un job exécutant analyze.py
  - Environnement /pbs/throng/training/Huma-Num/envs/text_analyzer
  - Variable d'environnement TEXT_LANG=FRA
  - Écrire un script pour analyser data/01.txt
    - 1h
    - 1 CPU
    - 6Go de mémoire
    - Partition flash

```
$ cd huma-num/slurm/text_analyzer
$ source /pbs/throng/training/Huma-Num/envs/text_analyzer/bin/activate
$ export TEXT_LANG=FRA
$ python analyze.py data/01.txt
```

TP - Job CPU

```
$ cat analyze.sh
#!/bin/bash

export TEXT_LANG=FRA
source /pbs/throng/training/Huma-Num/envs/text_analyzer/bin/activate
python analyze.py $1
```

```
$ sbatch -p flash -t 01:00:00 -c 1 --mem 6G analyze.sh data/01.txt
sbatch: INFO: Account set to: ccin2p3
Submitted batch job 15761697
```

```
$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          15761697     flash analyze. gcochard  R       0:02      1 ccwslurm0001

$ sacct -j 15761697 -o jobid,partition,state,alloccpus,maxrss --unit G
JobID         Partition      State  AllocCPUS     MaxRSS
------------ ---------- ---------- ---------- ----------
15761697          flash  COMPLETED          1
15761697.ba+             COMPLETED          1      5.11G

$ cat slurm-15761697.out
[…]
```

TP - Jobs GPU

- Exemple de compilation & exécution (Cuda)
  - Droits des fichiers (+x)
  - Paramètres de soumission (#SBATCH)

```
$ cd huma-num/slurm/matrixmul
$ sbatch job_matrixmul.sh
```

- Exemple d'utilisation de framework de Machine Learning (Pytorch)
  - Sourcer un environnement Python

```
$ cd huma-num/slurm/fashionmnist
$ sbatch job_fashionmnist.sh
```

Steps, tasks et parallélisation

Jobs

- Job : demande d'allocation de ressources
  - Chaque job a un ID numérique (par exemple 465)
  - Le job est mis en queue avant l'allocation des ressources
- Step : subdivision d'un job
  - Un job peut contenir une ou plusieurs steps déterminées par l'utilisateur. Elles sont identifiées par le job ID et un identifiant de step (par exemple 465.1)
  - Les steps peuvent s'exécuter séquentiellement ~~et/ou parallèlement~~
  - Les steps parallèles se partagent les ressources allouées au job
  - Un job a au moins une step
- Task : plus petite unité d'exécution d'un job
  - Les ressources sont associées aux tasks
  - Il peut y avoir plusieurs tasks par step
  - Les tasks peuvent communiquer via MPI

Job steps & tasks

- `sbatch` lance un job et au moins une step (par défaut nommée `JOBID.batch`)
  - `sbatch` ne lance pas de task, seulement une step qui lancera elle-même la/les task(s)
  - `--ntasks`, `--cpus-per-task` et `--mem-per-cpu` permettent d'allouer des ressources pour de futures tasks
- `srun` permet de lancer des steps et les tasks à l'intérieur de ces steps

```
$ cat script.sh
#!/bin/bash
#SBATCH --ntasks=2
#SBATCH --cpus-per-task=2
#SBATCH --mem-per-cpu=1G
#SBATCH --time=00:01:00

echo "hello"
```

```
JobID                              AllocTRES
------------ ------------------------------
61878         billing=4,cpu=4,mem=4G,node=1
61878.batch              cpu=4,mem=4G,node=1
```

```
hello
```

```
$ cat script.sh
#!/bin/bash
#SBATCH --ntasks=2
#SBATCH --cpus-per-task=2
#SBATCH --mem-per-cpu=1G
#SBATCH --time=00:01:00

srun echo "hello"
```

```
JobID                              AllocTRES
------------ ------------------------------
61886         billing=4,cpu=4,mem=4G,node=1
61886.batch              cpu=4,mem=4G,node=1
61886.0                  cpu=4,mem=4G,node=1
```

```
hello
hello
```

```
$ cat script.sh
#!/bin/bash
#SBATCH --ntasks=2
#SBATCH --cpus-per-task=2
#SBATCH --mem-per-cpu=1G
#SBATCH --time=00:01:00

srun --ntasks 1 echo "hello"
```

```
JobID                              AllocTRES
------------ ------------------------------
61888         billing=4,cpu=4,mem=4G,node=1
61888.batch              cpu=4,mem=4G,node=1
61888.0                  cpu=2,mem=2G,node=1
```

```
hello
```

Job steps & tasks

```
$ cat simpletask.sh
#!/bin/bash
echo "step: $SLURM_STEP_ID; task: $SLURM_TASK_PID"
sleep 10
```

```
$ cat script.sh
#!/bin/bash
#SBATCH --ntasks=2
#SBATCH --cpus-per-task=2
#SBATCH --mem-per-cpu=1G
#SBATCH --time=00:01:00

srun ./singletask.sh
srun ./singletask.sh
```

```
JobID                            AllocTRES     Elapsed
------------ ------------------------------ ----------
61890         billing=4,cpu=4,mem=4G,node=1   00:00:22
61890.batch              cpu=4,mem=4G,node=1   00:00:22
61890.0                  cpu=4,mem=4G,node=1   00:00:11
61890.1                  cpu=4,mem=4G,node=1   00:00:10
```

```
step: 0; task: 188445
step: 0; task: 188444
step: 1; task: 188543
step: 1; task: 188542
```

```
$ cat script.sh
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=2
#SBATCH --mem-per-cpu=1G
#SBATCH --time=00:01:00

srun ./singletask.sh
srun --ntasks 1 ./singletask.sh
```

```
JobID                            AllocTRES     Elapsed
------------ ------------------------------ ----------
61895         billing=2,cpu=2,mem=2G,node=1   00:00:21
61895.batch              cpu=2,mem=2G,node=1   00:00:21
61895.0                  cpu=2,mem=2G,node=1   00:00:10
61895.1                  cpu=2,mem=2G,node=1   00:00:11
```

```
step: 0; task: 188241
step: 1; task: 188300
```

```
$ cat script.sh
#!/bin/bash
#SBATCH --ntasks=2
#SBATCH --cpus-per-task=2
#SBATCH --mem-per-cpu=1G
#SBATCH --time=00:01:00

srun --ntasks 1 ./singletask.sh &
srun --ntasks 1 ./singletask.sh &
wait
```

```
JobID                            AllocTRES     Elapsed
------------ ------------------------------ ----------
61898         billing=4,cpu=4,mem=4G,node=1   00:00:11
61898.batch              cpu=4,mem=4G,node=1   00:00:11
61898.0                  cpu=2,mem=2G,node=1   00:00:10
61898.1                  cpu=2,mem=2G,node=1   00:00:10
```

```
step: 0; task: 184336
step: 1; task: 184347
```

Job steps & tasks

```
$ sbatch -p htc script.sh
```

```
$ cat script.sh
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=res.txt
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=100
srun -n1 hostname
srun -n1 sleep 60
```

Success

```
$ cat script.sh
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=res.txt
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=100
srun -n1 hostname &
srun -n1 sleep 60 &
wait
```

Failed!

```
$ cat script.sh
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=res.txt
#SBATCH --ntasks=2
#SBATCH --mem-per-cpu=100
srun -n1 hostname &
srun -n1 sleep 60 &
wait
```

Success

- Grâce aux steps il est possible de répartir les ressources et de parallélister des tâches
- Il faut faire attention à demander suffisamment de ressources au niveau du job, surtout en cas de parallélisation des steps

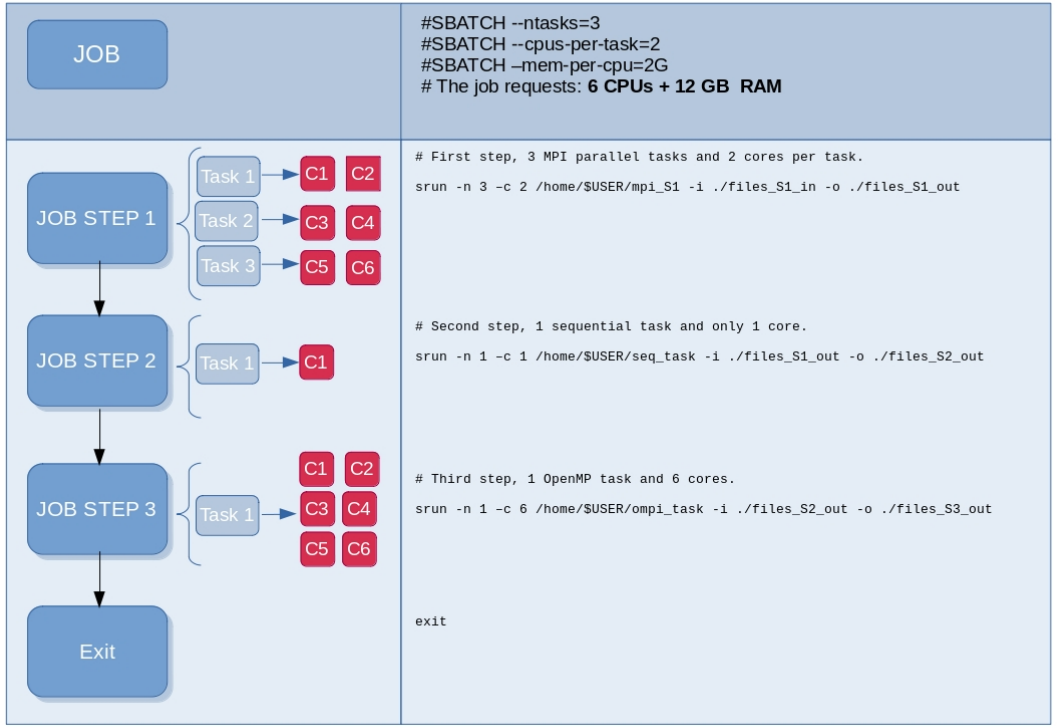
Jobs arrays

- Soumission de plusieurs jobs identiques
  - Permet de soumettre de nombreux jobs en limitant l'impact sur Slurm
  - Chaque job aura les mêmes caractéristiques (ressources, temps, etc.)
  - Job ID sous la forme XXX_YY (par exemple 321_01, 321_02, etc.)

- Variables d'environnement
  - Permettent de différencier les jobs
  - Elles peuvent être utilisées pour initialiser les entrées/sorties de chaque job
  - Variables pour le job 321_05 de l'array 321_[01-20] :
    - `SLURM_ARRAY_JOB_ID` : job ID de l'array (exemple : 321)
    - `SLURM_ARRAY_TASK_ID` : index du job (exemple : 05)
    - `SLURM_ARRAY_TASK_COUNT` : nombre de jobs dans l'array (exemple : 20)
    - `SLURM_ARRAY_TASK_MAX` : plus haut index de l'array (exemple : 20)
    - `SLURM_ARRAY_TASK_MIN` : plus petit index de l'array (exemple : 01)

Parallélisation

- Parallélisation via les tasks
  - Si parallélisation de tâches identiques (même ressources par tâches)
  - Si pas de dépendances entre tâches
- Parallélisation via les steps
  - Optimisation de la répartition des ressources
  - Regroupement de différentes phases du job
  - Exécution séquentielle avec des dépendances
- Parallélisation via les jobs
  - Utilisation de job array
  - Si parallélisation de calculs indépendants
  - Permet d'allouer moins de ressource par job



Place au TP

Partie 2

TP - Job array

- Soumettre un job array exécutant analyze.py
  - Environnement `/pbs/throng/training/Huma-Num/envs/text_analyzer`
  - Variable d'environnement `TEXT_LANG=FRA`
  - Écrire un script pour analyser `data/0[1-4].txt`
    - 1h
    - 1 CPU
    - 6Go de mémoire
    - Partition flash
  - Indice : utiliser `$SLURM_ARRAY_TASK_ID`

TP - Job array

```
$ cat analyze.sh
#!/bin/bash

export TEXT_LANG=FRA
source /pbs/throng/training/Huma-Num/envs/text_analyzer/bin/activate
FILE=$(printf "%02d" ${SLURM_ARRAY_TASK_ID})
python analyze.py data/${FILE}.txt
```

```
$ sbatch -p flash -t 01:00:00 -c 1 --mem 6G --array 1-4 analyze.sh
sbatch: INFO: Account set to: ccin2p3
Submitted batch job 15762083
```

```
$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
        15762083_1     flash analyze- gcochard  R       0:31      1 ccwslurm0001
        15762083_2     flash analyze- gcochard  R       0:31      1 ccwslurm0001
        15762083_3     flash analyze- gcochard  R       0:31      1 ccwslurm0001
        15762083_4     flash analyze- gcochard  R       0:31      1 ccwslurm0001

$ cat slurm-15762083_1.out
[…]
```

TP - Job Apptainer

- Soumettre un job utilisant une image apptainer
  - Image /pbs/throng/training/Huma-Num/data/analyze.sif
  - Variable d'environnement MPLCONFIGDIR=/tmp
  - Montages host → conteneur :
    - output → /output/text
    - /tmp → /opt/tmp
  - Écrire un script pour analyser data/hugo.pdf
    - 1h
    - 1 CPU
    - 6Go de mémoire
    - Partition flash

TP - Job array

```
$ cat image.sh
#!/bin/bash

apptainer run --app ocr -B output:/output/text/ -B /tmp:/opt/tmp /pbs/throng/training/Huma-
Num/data/analyze.sif data/hugo.pdf
```

```
$ sbatch -p flash -t 01:00:00 -c 1 --mem 6G image.sh
sbatch: INFO: Account set to: ccin2p3
Submitted batch job 15762083
```

Slurm@CC : Slurmfacts et types de données

- Ressources demandées :
  - timelimit
  - reqcpus
  - reqmem
- Ressources allouées :
  - timelimit
  - alloccpus
  - allocmem
- Ressources utilisées :
  - elapsed
    - Efficacité : elapsed / timelimit
  - CPU :
    - cputime : elapsed * alloccpus
    - totalcpu : temps CPU réellement utilisé
    - Efficacité : totalcpu / cputime
  - Mémoire
    - maxrss : valeur de crête
    - Efficacité : maxrss / allocmem

Slurm@CC : accounting

- hs06 et h23 : benchmark CPU de la communauté HEP
- Historiquement les slots faisaient 1 CPU / 3Go
  - Les utilisateurs demandent (et consomment) de plus en plus de mémoire
  - Les nouvelles machines ont des ratios plus élevés
    - 3.4G/CPU pour les machines HTC récentes
    - 10G/CPU pour les machines LSST

- Le CC ne prend en compte que les hs06 consommés
  - Comment prendre en compte la consommation mémoire ?
  - Comment pousser les utilisateurs à demander ce dont ils ont réellement besoin ?
- Pour prendre en compte la consommation mémoire, une astuce est utilisée côté accounting :
  - Consommation d'un job = max(hs23 consommés, consommation mémoire×hs23/ratio de la machine)
  - Job de 1h, 5 CPUs, 10G mémoire max : conso de 1h·5·hs23
  - Job de 1h, 5 CPUs, 30G mémoire max : conso de 1h·30·hs23/3
- ⚠ La facturation CPU se base sur alloccpu, la facturation mémoire sur maxrss