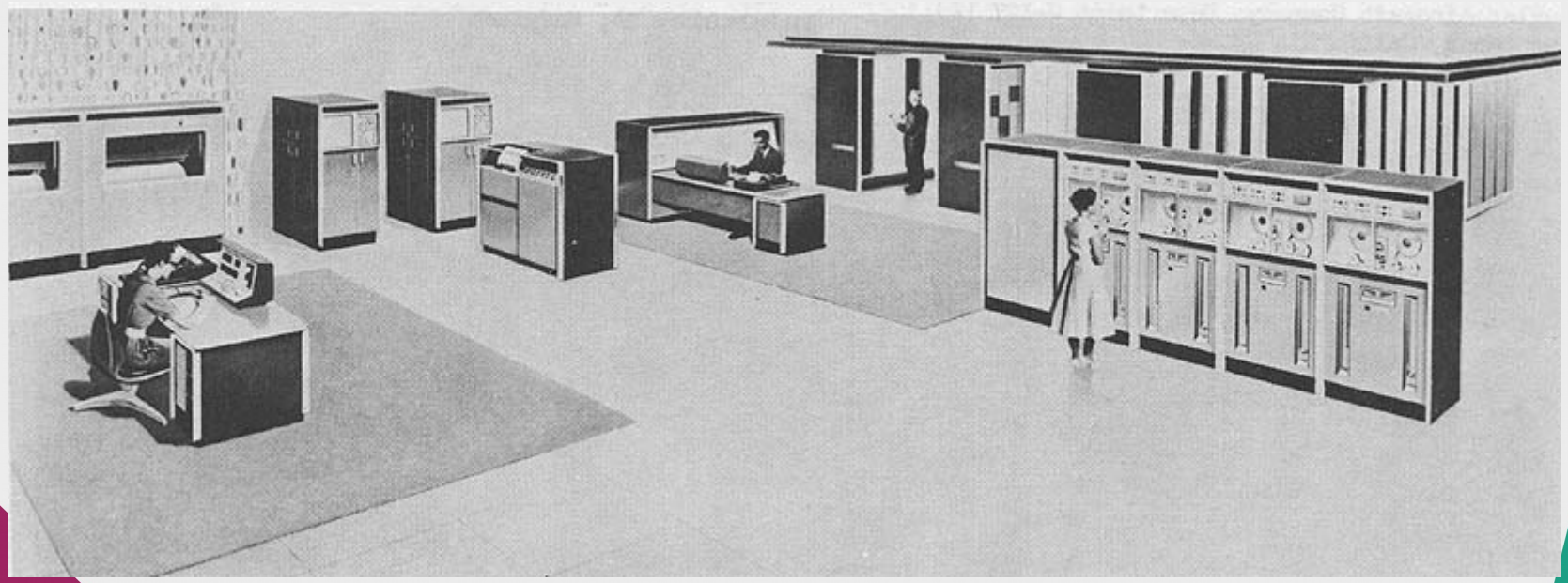
Batch Scheduling

Introduction

Matthieu Hautreux – matthieu.hautreux@cea.fr

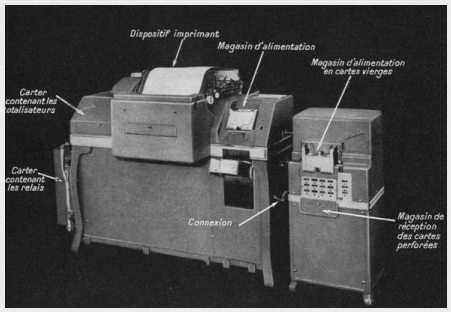
Un peu d’histoire

- Premières générations d’ordinateurs (50’s - 60s)
  - Utilisées pour le calcul scientifique (HPC!) et pour l’automatisation des actions administratives des grandes compagnies

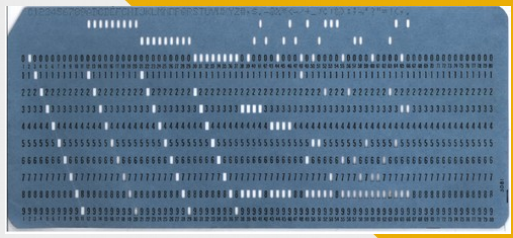


Un peu d’histoire

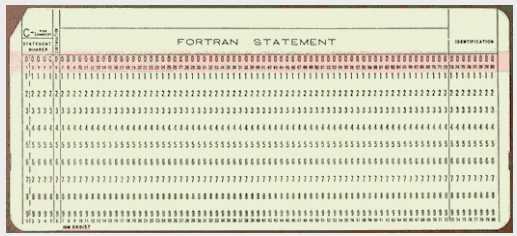
- Premières générations d’ordinateurs
  - La programmation est faite à base de cartes perforées, regroupées par lot (batch).
    - Les cartes les plus répandues ont 80 colonnes et 12 « lignes ».
  - Les entrées sont elles aussi fournies au travers de cartes, puis de bandes magnétiques.
  - Les sorties sont réalisées sur cartes ou imprimantes.



Un peu d’histoire



- Premières générations d’ordinateurs
  - Les accès interactifs sont inexistants pour les « utilisateurs » de ces machines.
    - Les cartes sont leur seul mode d’interaction
  - La première carte d’un lot, la carte « job », est souvent de couleur particulière pour simplifier la gestion des paquets.
    - Un coin des cartes est d’ailleurs tronqué pour en simplifier l’organisation et le rangement dans des boites adaptées.



Un peu d’histoire

- Premières générations d’ordinateurs
  - L’utilisation des machines est assurée par des opérateurs qui chargent les paquets de cartes en machine suivant un planning pré-établi.
    - → batch scheduling
  - Les résultats, des « listings » papiers, sont fournis aux utilisateurs après l’exécution de leurs « jobs ».
    - Les « jobs » en erreur produisent une quantité pharamineuse de sorties ...

Un peu d’histoire

- Premières générations d’ordinateurs
  - Mode d’utilisation
    - On planifie (schedule) l’exécution de jobs par paquets consécutifs de cartes perforées.
    - On génère des « listings » papiers.
  - On passe un temps certain à mettre au point les « cartes » de ses « jobs » et à en traiter les « listings »

Un peu d’histoire

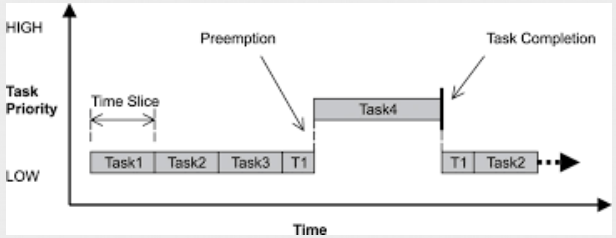
- L’émergence de l’informatique moderne
  - L’arrivée des transistors, des bandes magnétiques et des mémoires permet la conception de nouvelles machines.
  - Les « mainframes » apparaissent
    - Les terminaux « graphiques » 80 colonnes font leur entrée.
  - Des lecteurs de cartes restent associés…
    - Pour réutiliser les codes…
    - Et migrer vers des « scripts »

Un peu d’histoire

- L’ère « mainframe » (70’s)
  - Les scripts et programmes se numérisent
    - Les cartes sont mises au placard après numérisation.
    - Les données sont enregistrées sur bandes magnétiques et chargées/écrites depuis les programmes.
  - Les « jobs » sont planifiés par des opérateurs puis par des applications spécialisées.
    - Les premiers « batch scheduler »…

Un peu d'histoire

- L'ère « mainframe » (70's)
  - Le batch scheduling est toujours présent mais plus seul pour opérer les machines.
    - Notion de **time slicing** (partage de temps)
    - Actions automatiques en tâches de fond et pendant les périodes d'inactivité (nuit/weekend)
  - Des politiques d'ordonnancement doivent être mises en place
    - pour automatiser l'exécution des tâches en fonction de leurs priorités



Un peu d’histoire

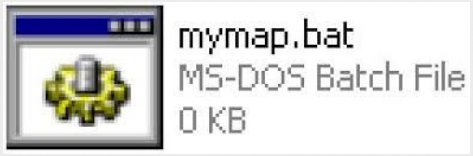
- L’ère « mainframe » (70’s)
  - Mode d’utilisation
    - « Soumission » de scripts batch par les utilisateurs (jobs).
    - Ordonnancement automatique de l’exécution des jobs par une application dédiée.
    - On génère des « listings » numérisés : sorties « écran » redirigées dans des fichiers.
  - On gagne du temps dans la mise au point des scripts et programmes et le dépouillement des résultats.
  - On attend en fonction de l’importance du programme plus ou moins longtemps avant son exécution.

Un peu d’histoire

- L’ère « PC » (80’s - 00’s)
  - L’informatique se miniaturise et se démocratise par le canal des « personnal computeur »
  - Qui reprennent les concepts des « mainframe » en associant directement le terminal à l’ « unité centrale ».
  - L’évolution est forte et rapide.
    - Les interfaces graphiques font vite leur entrée

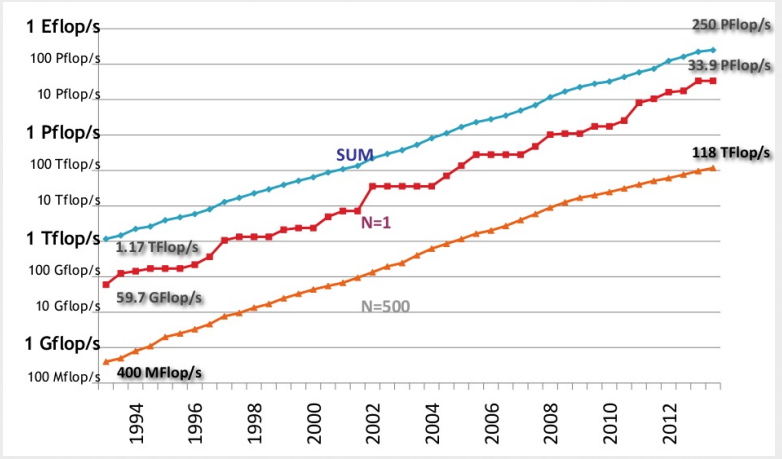
Un peu d’histoire

- L’ère « PC » (80’s - 00’s)
  - Les systèmes d’exploitation permettent
    - Une interaction directe via des interfaces graphiques simplifiant l’utilisation des machines
    - Une interaction en mode ligne de commandes et/ou scripts.
      - Ex : fichiers script « .bat » de Windows
    - Les « scripts » restent exécutables en arrière plan (crontab) pour les traitements « batch ».

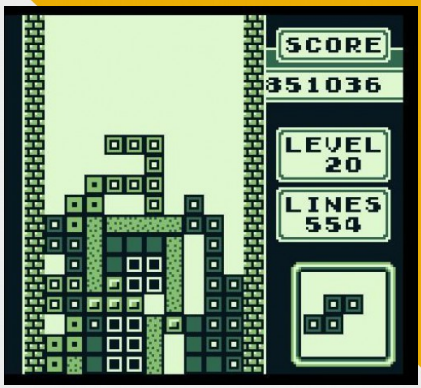


Un peu d'histoire

- L'émergence des clusters (90s)
  - Les réseaux prennent de l'ampleur et permettent une interconnexion performante d'unités individuelles type PC.
  - Le HPC s'engouffre dans cette voie face à la diminution des performances des approches monolithiques des « Mainframe ».
    - Le nombre d'unités de calcul connectées ne cessera de croître...

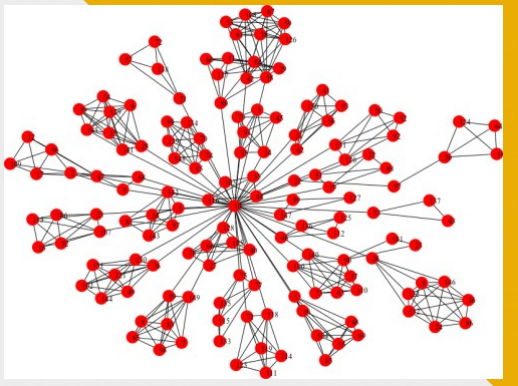


Un peu d’histoire



- L’émergence des clusters (90s)
  - L’utilisation des clusters nécessite alors l’orchestration de plusieurs unités de calcul indépendantes.
    - Notion de « **jobs parallèles** » exécutés sur des « systèmes distribués »
  - Un ordonnanceur central est en charge de la répartition « spatiale » et « temporelle » des travaux.
    - Dédier un certain nombre de « nœuds » pour une période de temps donnée à un « job ».

Un peu d'histoire



- L'émergence des clusters (90s)
  - Les jobs deviennent hétérogènes
    - Une ou plusieurs sections parallèles permettant l'exécution de codes de calcul optimisés pour l'utilisation de plusieurs unités de calcul
      - Émergence du modèle MPI !
    - Encapsulée(s) dans le déroulement classique du script « batch »
  - L'ordonnancement se complexifie
    - Différents besoins en terme de nombres d'unités de calcul dans les sections parallèles.
    - Différentes localités.

Un peu d’histoire

- L’émergence des clusters (90s)
  - Les « batch scheduler » évoluent donc pour traiter efficacement ces « systèmes distribués »
    - On parle maintenant de **DRMS**

      (Distributed Resource Management System)

- **Ces DRMS feront l’objet de la suite de cette présentation.**

HPC Batch Scheduler

Principe des DRMS

DRMS - Architecture

- Rappels
  - Evolution des batchs scheduler « initiaux »
    - Gérant principalement des « jobs » en **time slicing**

      → composant « job manager »
    - Prise en charge d’une quantité de ressources de calcul grandissante et distribuée

      → composant « resource manager »

DRMS - Architecture

- Repose généralement sur un composant **leader** central
  - Permettant aux utilisateurs d’enregistrer leurs « jobs » pour exécution ultérieure
  - Fournissant un statut des ressources disponibles et en cours d’utilisation
  - Fournissant un statut des jobs en cours de calcul ou en attente de ressources
  - Fournissant l’historique et les statistiques d’utilisation des ressources

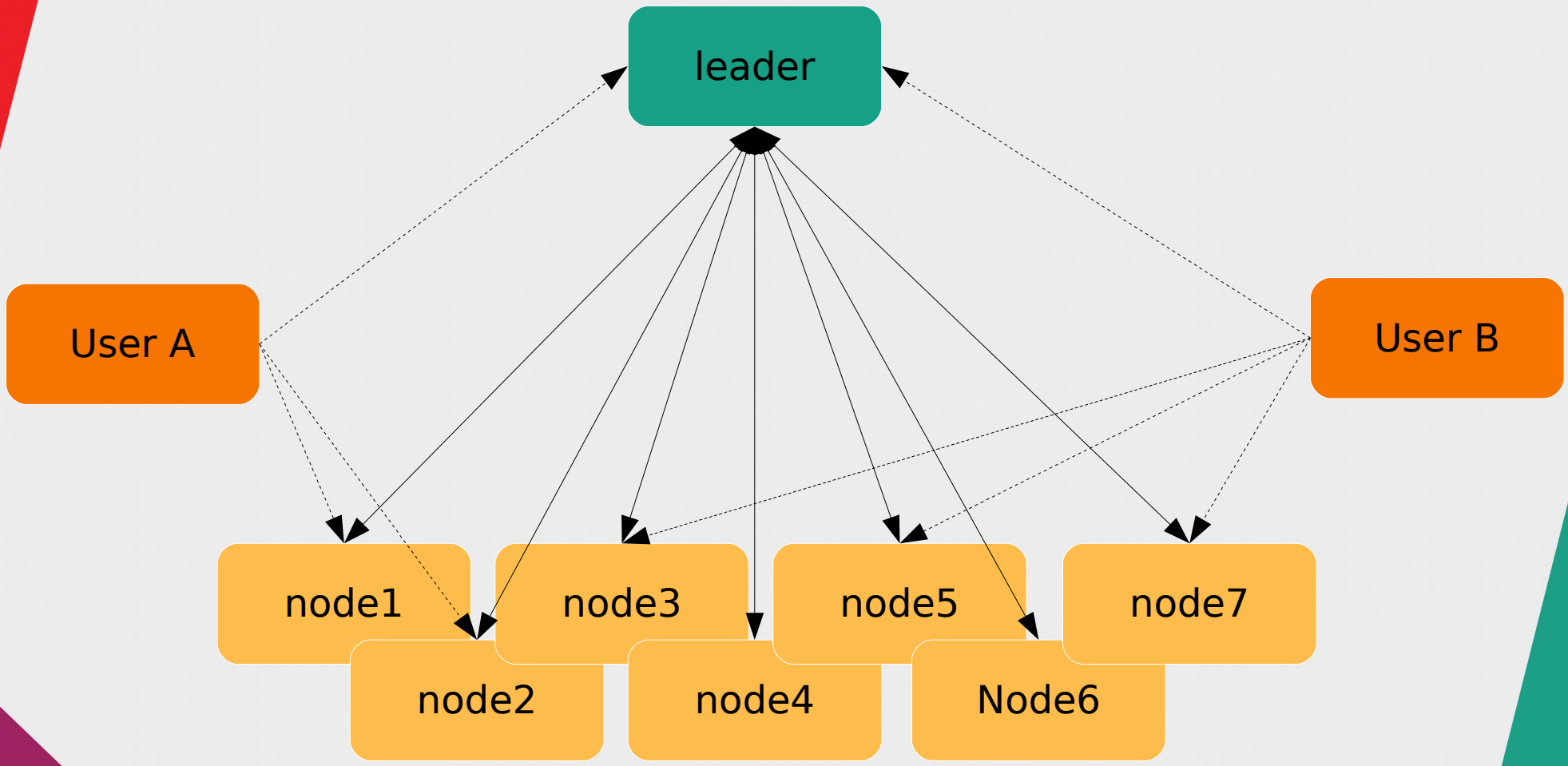
DRMS - Architecture

- Repose généralement sur un composant **leader** central
  - Orchestrant la répartition des ressources entre les « jobs » au cours du temps
  - Orchestrant la mise en exécution, l’arrêt des jobs ainsi que le suivi de la bonne utilisation et la libération des ressources utilisées

DRMS - Architecture

- Repose généralement sur un ensemble de **workers** distribués
  - Généralement un par nœud de calcul
  - Fournit l'état du nœud au leader et permet les intéractions directes avec celui-ci ou les utilisateurs
  - En charge du démarrage des exécutions de scripts et ou d'applications pour les utilisateurs
  - Assure le suivi de la bonne utilisation et la libération des ressources utilisées

DRMS - Architecture



HPC Batch Scheduler

Principes et utilisation de Slurm

Slurm - Introduction

- **S**imple **L**inux **U**tility for **R**esource **M**anagement
  - Simple → Scalable
- Projet démarré au **LLNL** en 2002
  - Lawrence Livermore National Laboratory
    - Livermore, CA, USA
- Continué par **SchedMD** depuis 2010
  - Entreprise créé par les deux développeurs principaux de l’époque

Slurm - Introduction

- Produit **OpenSource** écrit en C
  - Licence GPLv2
- Utilisable sur la majorité des **environnements** de type **UNIX**
  - AIX, Linux, BSD, ...
- Utilisé sur une multitude de grands calculateurs à travers le monde
  - Dont certains parmi les plus grands

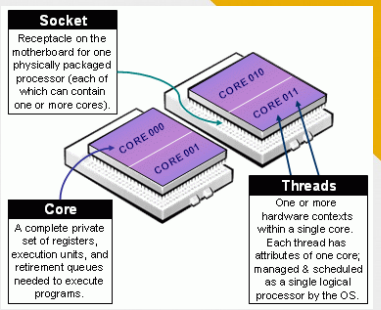
Slurm - Introduction

- **Scalable**
  - Permet la gestion de plusieurs dizaines de milliers de nœuds
  - Permet la gestion de plusieurs centaines de milliers de cœurs de calcul
- **Modulaire**
  - Basé sur la notion de plugins pour spécialiser différentes parties du produit en fonction des besoins

Slurm – Entités élémentaires

- **slurmctld**
  - Composant « leader » (controler)
- **slurmdbd**
  - Composant additionnel au «leader » pour la persistence des données de comptabilité sur les jobs et la gestion des utilisateurs et de leurs droits
    - Backend MariaDB nécessaire
- **slurmd**
  - Composant « worker »

Slurm – Notions élémentaires

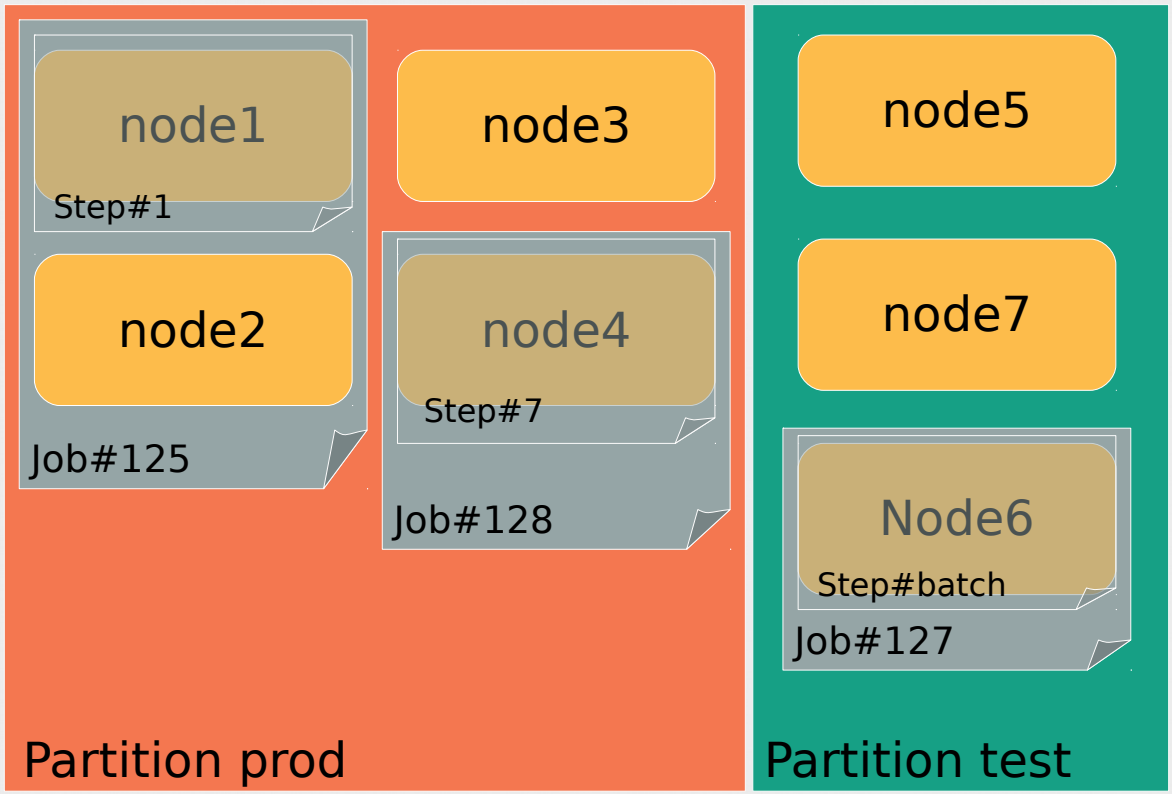


- **Node**
  - Unité indépendante fournissant des ressources utilisables par les utilisateurs
    - Sockets/Cores/Threads, Memory, GPUs, …
- **Partition**
  - « Pool » de nœuds utilisables au sein d'un même « job »
    - Un nœud peut appartenir à plusieurs partitions

Slurm – Notions élémentaires

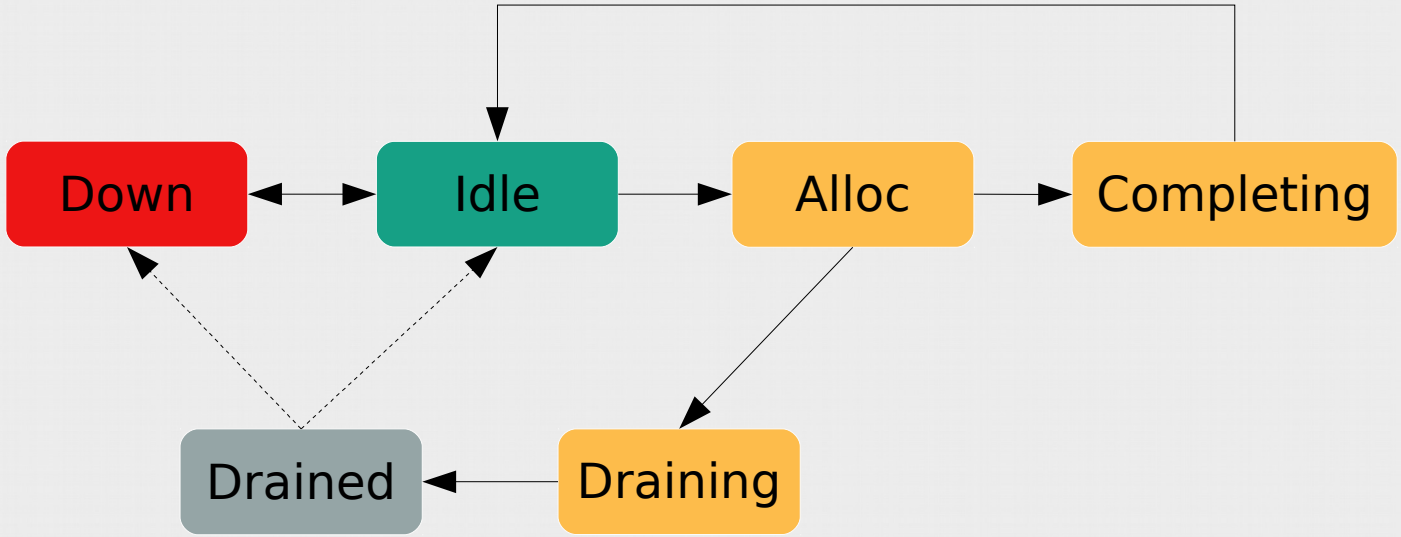
- **Job**
  - Demande d'allocation de ressources *dans une partition* associée à un utilisateur
    - Ensemble de ressources réparties sur des nœuds pour un temps défini
    - Batch (script fourni) ou Interactif (shell)
- **Jobstep**
  - Demande de sous-allocation de ressources pour effectuer une tâche particulière
    - Sous-ensemble de ressources parmi les ressources allouées pour le job associé

Slurm – Notions élémentaires



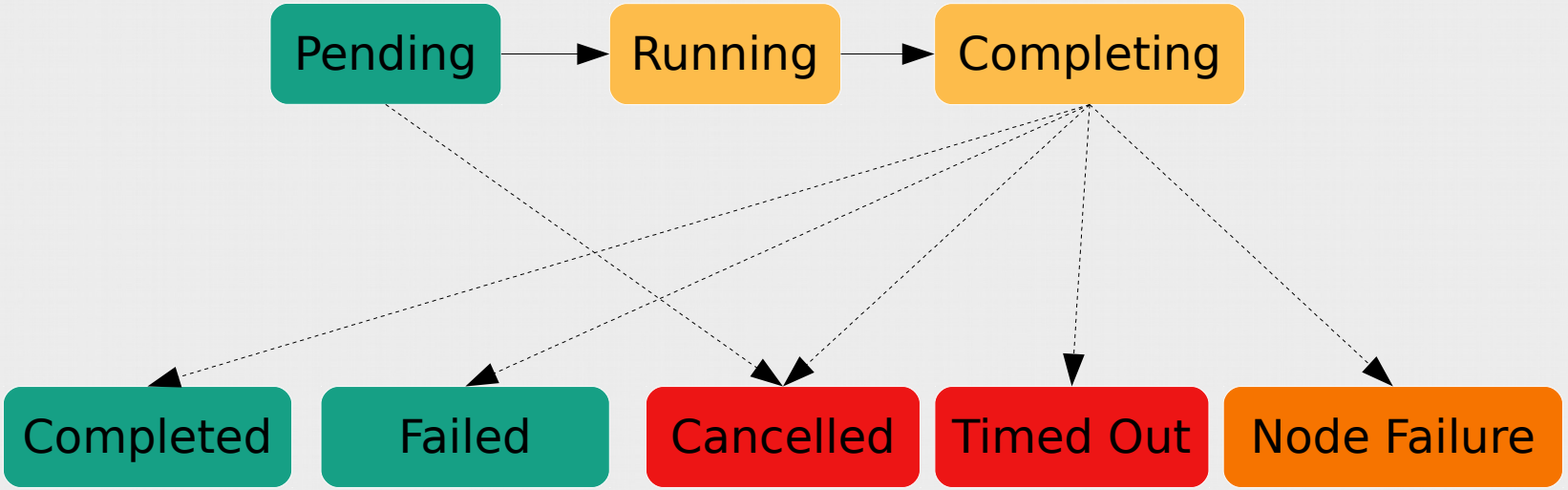
Slurm – Notions élémentaires

- **Node « states »**

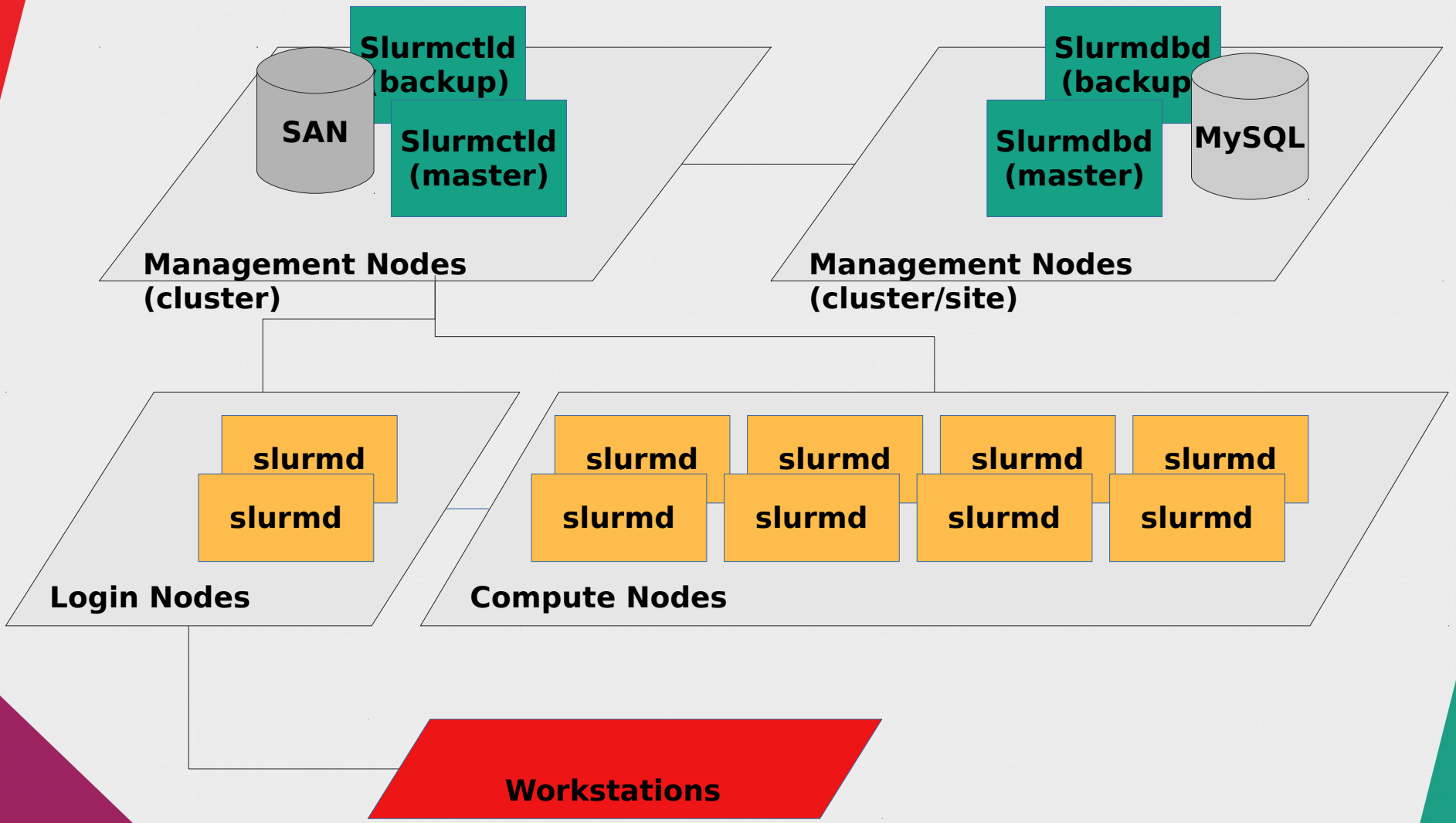


Slurm – Notions élémentaires

- **Job « states »**



Slurm - Architecture



Slurm – Notions élémentaires

- **Commandes principales**
  - **scontrol**
    - obtention & modification de la configuration
    - obtention & modification des états des éléments (nodes, partitions, jobs, …)
  - **sacctmgr**
    - obtention & modification de la configuration des éléments stockés en BD (« users », « accounts », « qos » …)

Slurm – Notions élémentaires

- **Commandes principales**
  - **sinfo**
    - Information sur l’état des partitions
  - **squeue**
    - Information sur l’état des « jobs »
  - **sacct**
    - Information sur l’exécution de jobs en cours ou passés
  - **sstat**
    - Information détaillée sur l’exécution de jobs en cours

Slurm – Notions élémentaires

- **Commandes principales**
  - **sbatch**
    - « Soumission » d'une demande d'allocation de ressources *détaillant les ressources nécessaires*
    - Fourniture du « script batch » associé
      - Exécution du script sur les ressources disponibles sur le premier nœud « alloué »
    - Mode « batch » (non interactif)
      - L'utilisateur ne peut plus interagir directement avec son job et doit utiliser les commandes Slurm adhoc pour cela
      - Les sorties stdout/stderr du script exécuté sont redirigés vers des fichiers (configurables)

Slurm – Notions élémentaires

- **Commandes principales**
  - **salloc**
    - « Soumission » d'une demande d'allocation de ressources *détaillant les ressources nécessaires*
    - Lancement d'un shell interactif associé aux ressources allouées dès réalisation
      - Ou exécution locale d'un script passé en argument
    - Permet l'exécution de commandes « srun » ultérieures pour créer des « jobstep » dans le job réalisé
      - Facilite les tests en évitant l'attente « pending→running » inhérente à chaque soumission

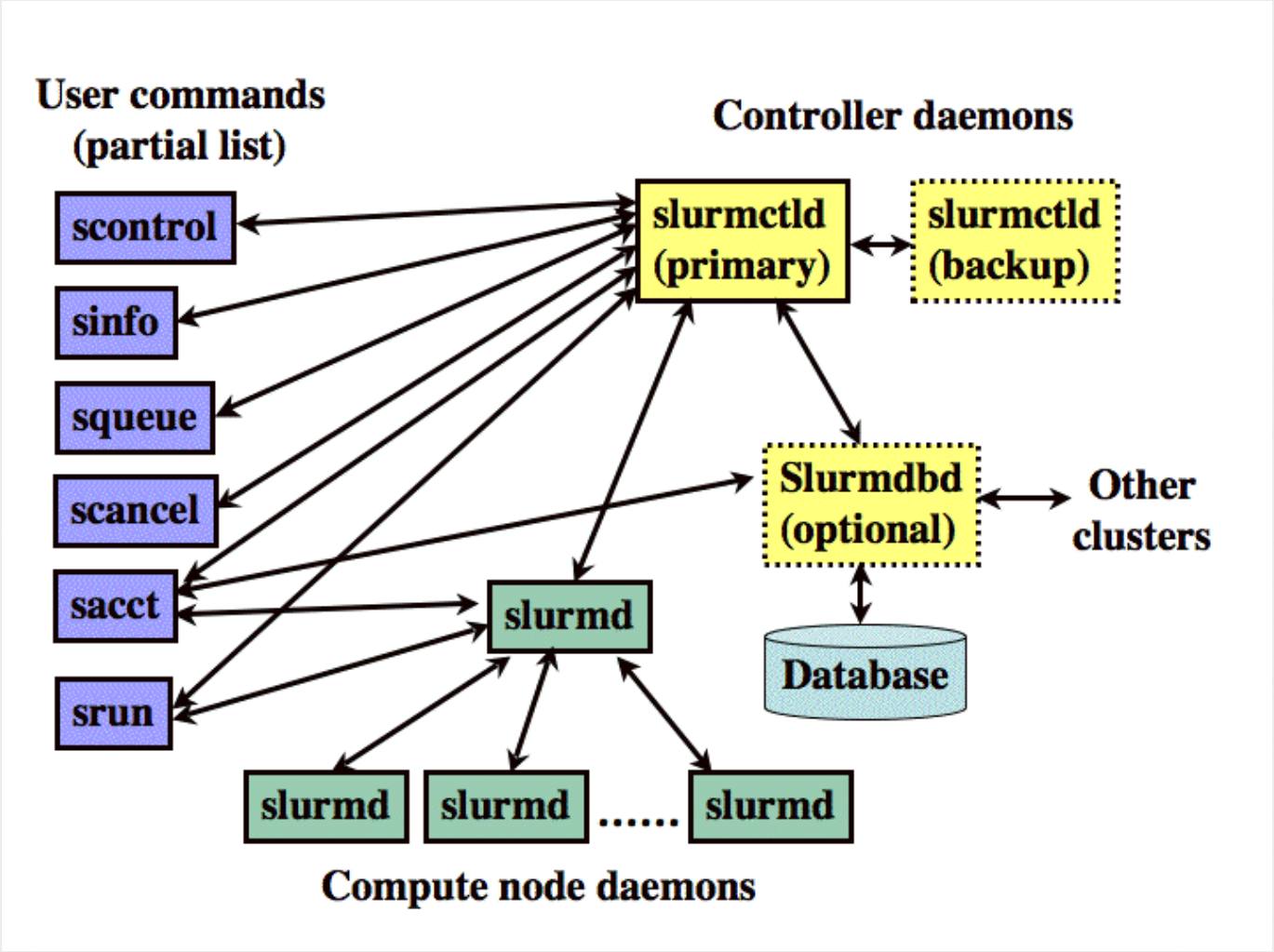
Slurm – Notions élémentaires

- **Commandes principales**
  - **srun**
    - « Soumission » d'une demande d'allocation de ressources *détaillant les ressources nécessaires*
    - Exécution d'un certain nombre de processus répartis sur les ressources allouées
      - en fonction des détails fournis en argument
    - Mode d'utilisation interactif
      - L'utilisateur suit l'exécution du job dans son terminal et peut interagir avec lui (signaux, stdin, ...)

Slurm – Notions élémentaires

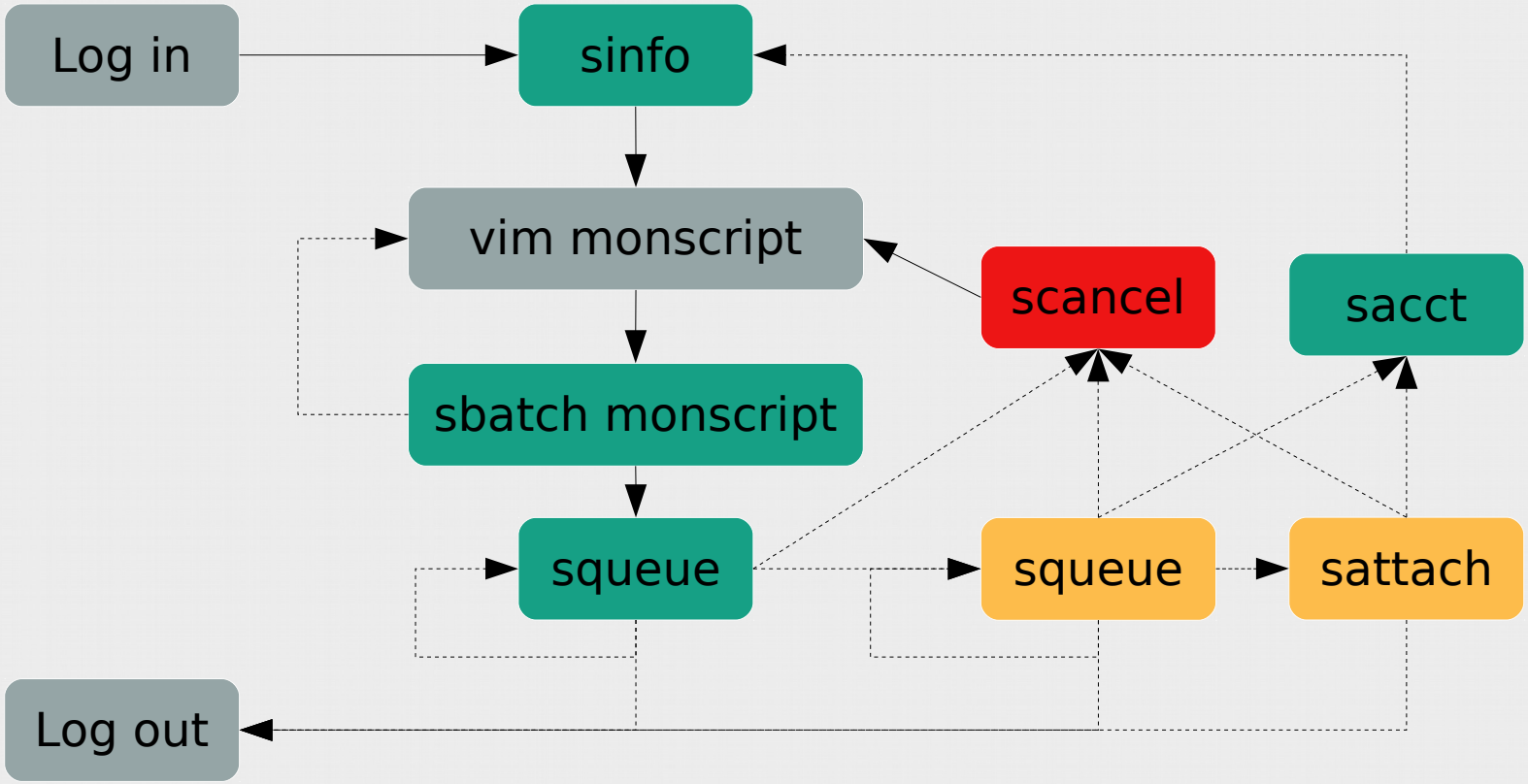
- **Commandes principales**
  - **sattach**
    - Permet de suivre et/ou d’interagir avec un job batch à la manière d’un job interactif
  - **scancel**
    - Permet la transmission d’un signal à un job ou jobstep
    - Permet de demander la terminaison au plus tôt d’un job ou jobstep

Slurm – Architecture



Slurm – Notions élémentaires

- **Cas d'utilisation - sbatch**

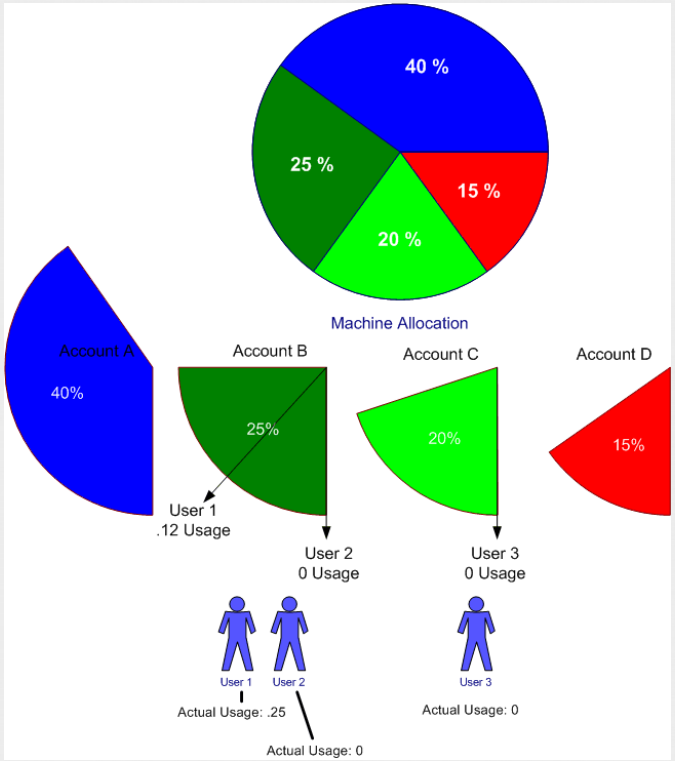


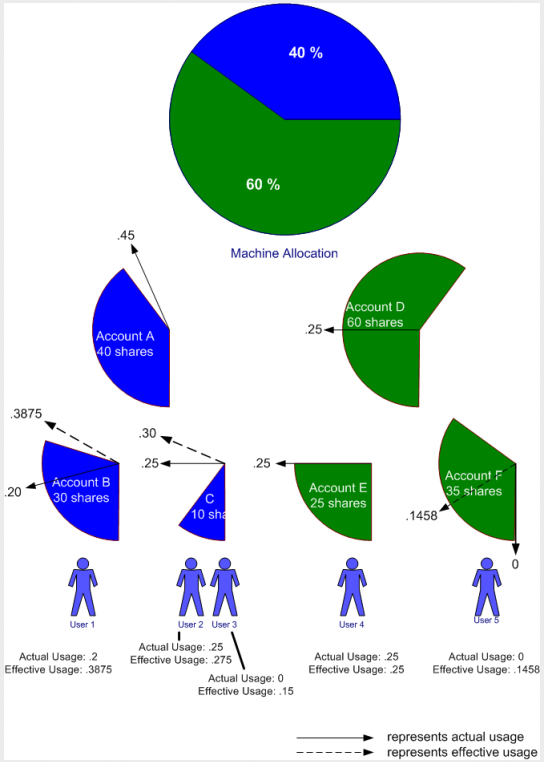
Slurm – Ordonnancement

- **Politiques d'ordonnancement classiques**
  - **FIFO : First-In First-Out**
    - **Premier arrivé, premier servi**
  - ~~**First-Fit**~~
    - ~~**Le premier qui tient dans l'espace disponible rentre en machine**~~
  - **FairSharing**
    - **Basé sur une notion de parts de ressources attribués aux différents utilisateurs**
    - **Celui qui rentre est celui dont l'utilisation est la plus inférieure à ce qu'il est autorisé à utiliser**

Slurm - Ordonnancement

- **Fairshare**
  - **Groupements hierarchiques d'utilisateurs, notion d' « account »**



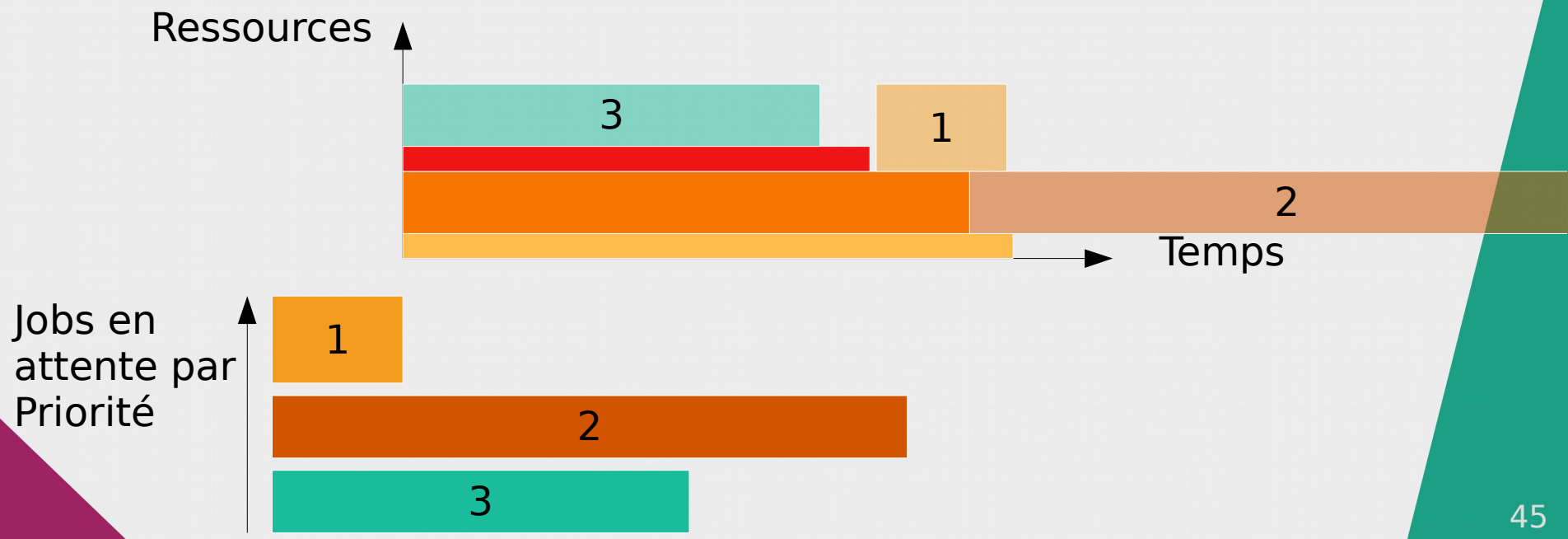


Slurm – Ordonnancement

- Politiques d’ordonnancement classiques
  - Aging
    - Le plus ancien est le plus prioritaire
  - Size based
    - Le plus petit (ou le plus gros) d’abord
  - QOS (Qualité de service)
    - Différentes qualités de service avec différentes restrictions
    - Certaines plus prioritaires que d’autres.

Slurm – Ordonnancement

- **Politiques d'optimisation classique**
  - **Backfilling**
    - **Un job moins prioritaire est exécuté en premier si il ne repousse pas la date de démarrage des plus prioritaires.**



Slurm – Ordonnancement

- **Politiques d’optimisation classique**
  - **Preemption**
    - **Un job moins prioritaire laisse sa place à un plus prioritaire lorsqu’il doit s’exécuter**
      - **(suspension d’exécution ou remise en file d’attente (queue))**
  - **Best-effort**
    - **Un job moins prioritaire est annulé si un plus prioritaire à besoin des ressources**

Slurm – Ordonnancement

- **Principes utilisés dans Slurm**
  - **Priority-based scheduling**
    - **Quelque soit la politique d'ordonnancement choisie, celle-ci doit se traduire par une pondération sous forme de priorité numérique**
    - **Le plus prioritaire part en premier**
  - **Backfilling optionnel**
  - **Preemption optionnelle**
  - **Best-effort optionnel**
    - **Preemption en mode « cancel »**

Slurm – Ordonnancement

- **Multifactor priority plugin**
  - **Permet de mélanger différentes politiques dans le calcul de la priorité d'un job**
    - **Factors : Age, Job Size, Partition, QOS, FairShare**
  - **Chaque « factor » est pondéré dans une somme globale des priorités de chacun.**
    - **Chaque factor → valeur entre 0 et 1 par job**
    - **Somme globale pondérée = priorité du job**

Slurm – Ordonnancement

- **Age Factor**
  - **Permet de fournir une indication sur le temps d'attente du job**
    - **afin de favoriser les plus anciens et éviter les famines (« starvation »)**
- **Job size Factor**
  - **Permet de fournir une indication sur la quantité de ressources demandée**
    - **Afin de favoriser les plus gros ou les plus petits jobs**

Slurm – Ordonnancement

- **Partition / QOS Factors**
  - **Chaque partition/qos dispose d'une priorité**
  - **La valeur renvoyée correspond à la normalisation de la valeur de la partition ciblée par rapport à la priorité maximum observée**
    - **Ex :**
      - **partition-A priority=20, fact=0.2**
      - **partition-B priority=100, fact=1.0**
      - **Partition-C priority=70, fact=0.7**

Slurm – Ordonnancement

- **FairShare factor**
  - **L’écart entre part utilisable et utilisée des utilisateurs pondère la valeur de chaque job permettant de revenir à l’équilibre souhaité au plus vite**
  - **Ex**
    - **User-A share=0.3 usage=0.2, fact=0.6**
    - **User-B share=0.2 usage=0.25, fact=0.45**

Slurm – Ordonnancement

- **Multifactor priority plugin**
  - **Exemple de configuration**

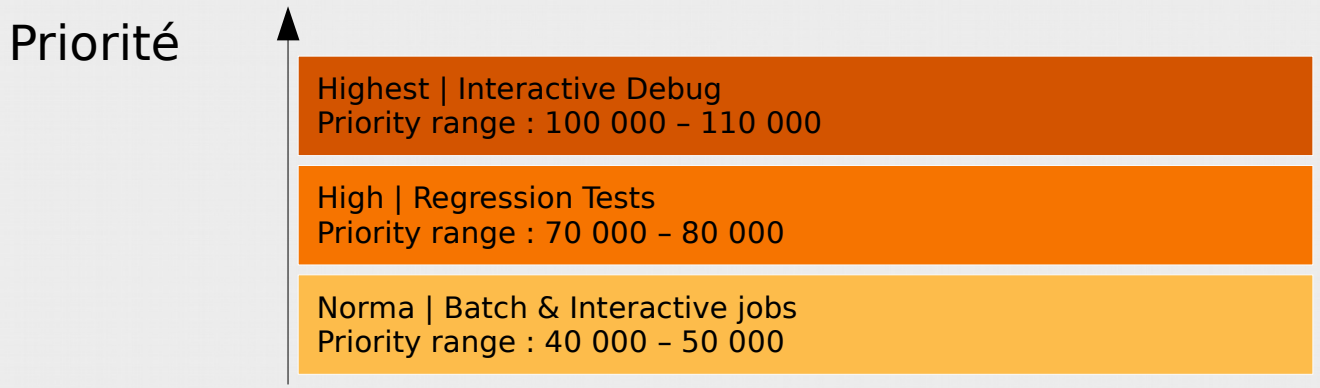
PriorityWeightQOS=100 000

PriorityWeightAge=10 000

PriorityWeightFairshare=10 000

PriorityWeightJobSize=0

PriorityWeightPartition=0



Slurm – Ordonnancement

- **Limitation d’accès aux ressources**
  - **Il peut être nécessaire de restreindre l’accès à certaines ressources à certains utilisateurs**
  - **Il peut être nécessaire de restreindre la quantité disponible de ressources pour certains utilisateurs**
  - **Il peut être nécessaire de restreindre le temps d’utilisation maximum possible en fonction des utilisateurs**
  - **…**

Slurm – Ordonnancement

- **Limitation d'accès aux ressources**
  - **Les partitions disposent d'un certain nombre de possibilités de restriction qui ne s'avèrent pas toujours pratiques ou manquent de factorisation**
  - **Les QOS permettent de corriger ce problème en fournissant des restrictions s'appliquant orthogonalement aux partitions**
    - **Une même partition peut être accédées via différentes QOS**

Slurm – Ordonnancement

- **Limitation d’accès aux ressources**
  - **Les « associations » permettent de rafiner encore la granularité de configuration des limitations**
    - **Une association peut correspondre à :**
      - **Un cluster et un account**
      - **Un cluster, un account et un utilisateur**
      - **Un cluster, un account, un utilisateur et une partition**
    - **Les restrictions s’appliquent hiérarchiquement sur les associations d’un utilisateur pour un account donné**
      - **Un utilisateur peut être associé à plusieurs account**

Slurm – Ordonnancement

- QOS – exemple de limites
  - MaxJobsPerUser
    - Quantité maximale de jobs en exécution
  - MaxSubmitJobsPerUser
    - Quantité maximale de jobs enregistrés
  - MaxNodes
    - Quantité maximale de nœuds utilisables dans un job
  - MaxWall
    - Temps d’exécution maximum d’un job
  - ...

Slurm – Ordonnancement

- **Association – exemple de limites**
  - **MaxJobs**
    - **Quantité maximale de jobs en exécution**
  - **MaxSubmitJobs**
    - **Quantité maximale de jobs enregistrés**
  - **GrpJobs**
    - **Quantité maximale de jobs en exécution incluant les jobs de toutes les associations filles**
  - **GrpSubmitJobs**
  - **...**

Slurm – Ordonnancement

- QOS – exemple de limites
  - MaxJobsPerUser
    - Quantité maximale de jobs en exécution
  - MaxSubmitJobsPerUser
    - Quantité maximale de jobs enregistrés
  - MaxNodes
    - Quantité maximale de nœuds utilisables dans un job
  - MaxWall
    - Temps d’exécution maximum d’un job
  - ...

La suite au prochain épisode ?

Slurm – Sécurité

- **L’aspect « multi-tenancy » nécessite une méthode d’authentification des utilisateurs interagissant avec les composants.**
- **L’aspect « distribué » nécessite la possibilité de valider les droits conférés aux utilisateurs le temps de l’exécution de leur travaux.**
- **Ces problématiques sont couvertes à l’aide d’un outil tiers, MUNGE (MUNGE Uid aNd GID Emporium), utilisé par Slurm.**
  - **https://dun.github.io/munge/**

Slurm – Sécurité

- **MUNGE**
  - **Repose sur le partage d’un secret partagé par l’ensemble des nœuds intégrés dans un même espace de confiance.**
  - **Repose sur la possibilité d’obtenir de façon fiable l’UID et le GID d’origine d’une connexion de type Socket BSD sur les systèmes UNIX.**

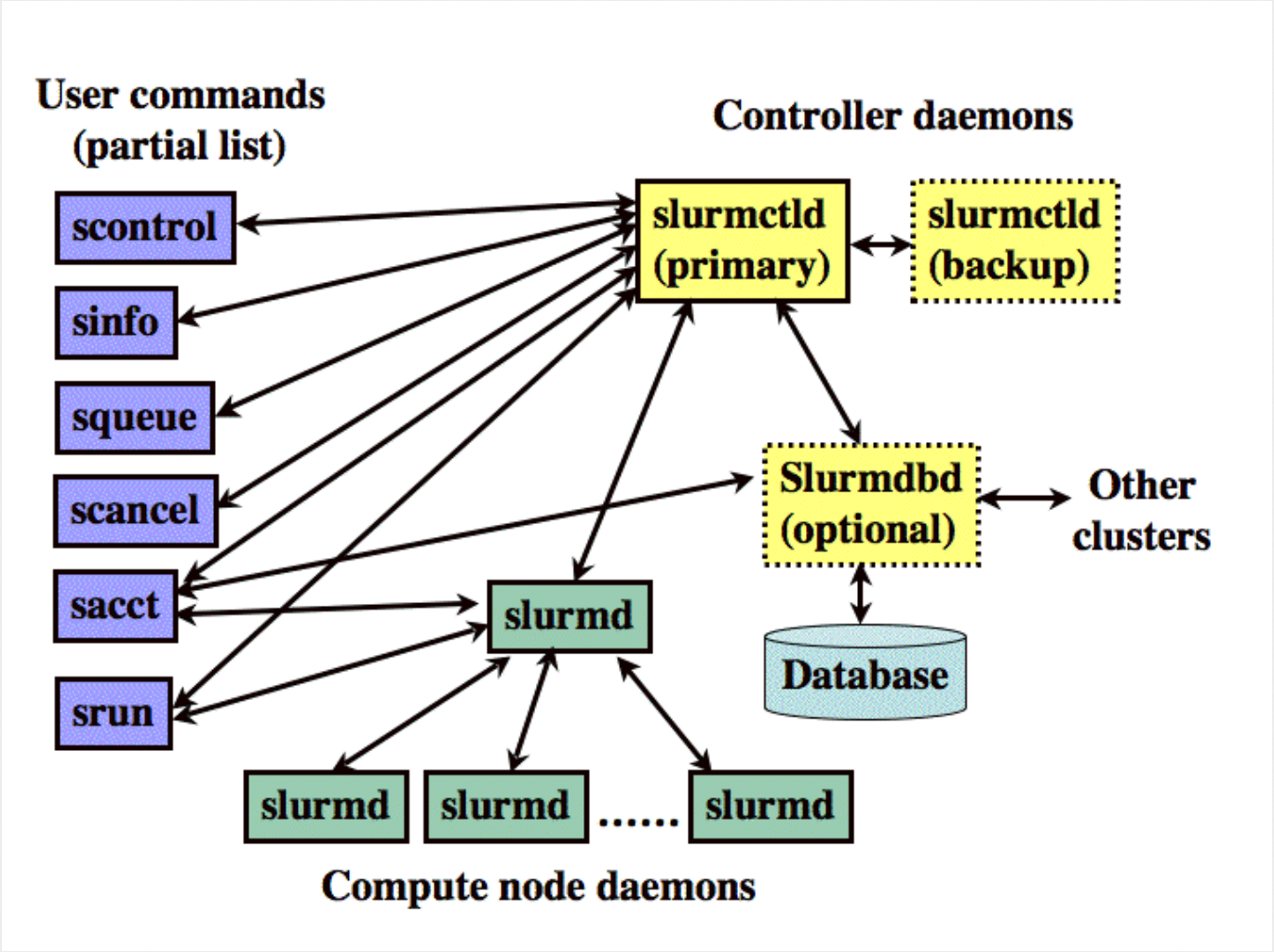
Slurm – Sécurité

- **MUNGE**
  - **Chaque nœud peut donc**
    - **générer des « credentials » associés aux couples uid/gid des utilisateurs en faisant la demande et les chiffrer avec le secret partagé.**
    - **valider des « credentials » chiffrés par ce même secret et fournir les uid/gid associés dans le but d'identifier l' « ayant-droit ».**

Slurm – Sécurité

- **Sécurité dans Slurm**
  - **Les communications dans Slurm sont authentifiées à l'aide de « credentials » MUNGE.**
    - **Tous les nœuds utilisés doivent donc disposer du secret partagé, uniquement accessible par le daemon privilégié de MUNGE.**
    - **Tous les nœuds offrent via une socket BSD locale, un moyen de générer et de valider des credentials.**

Slurm – Architecture



Slurm – Sécurité

- **Validation des droits d'utilisation**
  - **Les allocations de ressources autorisées par slurmctld sont associées à des « job credentials »**
    - **pour l'uid/gid associé**
    - **contenant le détail des ressources**
  - **Les daemons slurmd sont donc capables de vérifier la validité d'un accès aux ressources qu'ils proposent.**
    - **sans dialogue avec slurmctld**

La suite au prochain épisode !

- https://slurm.schedmd.com
- https://dun.github.io/munge/
- http://tardis.dl.ac.uk/computing_history/archive_images