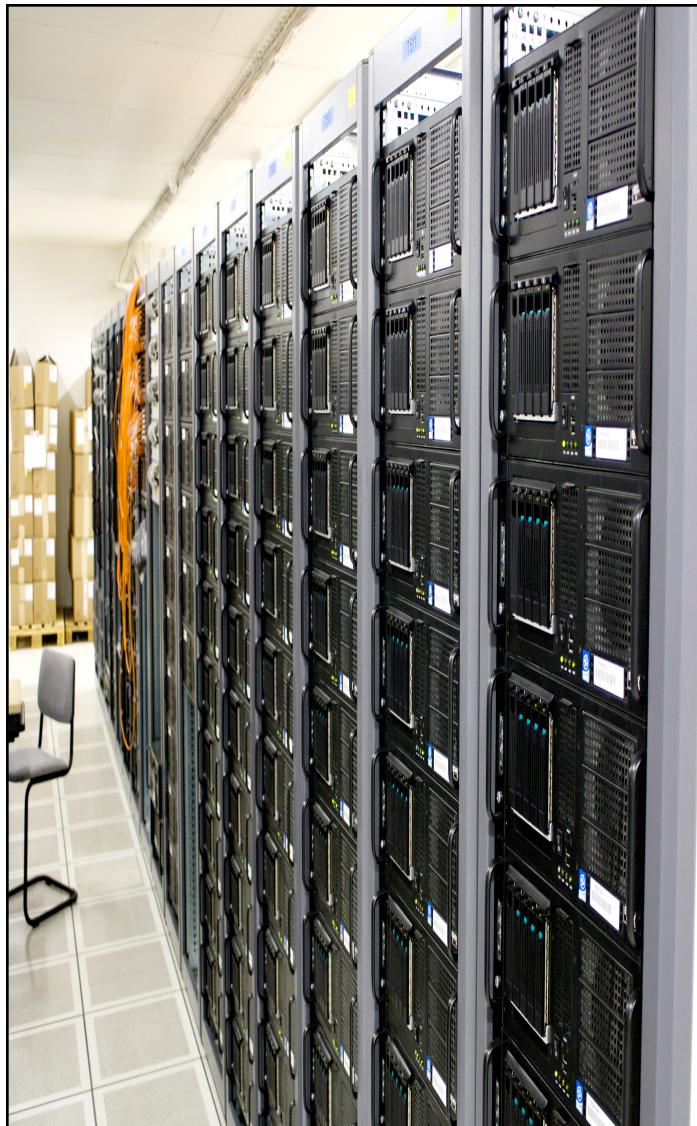




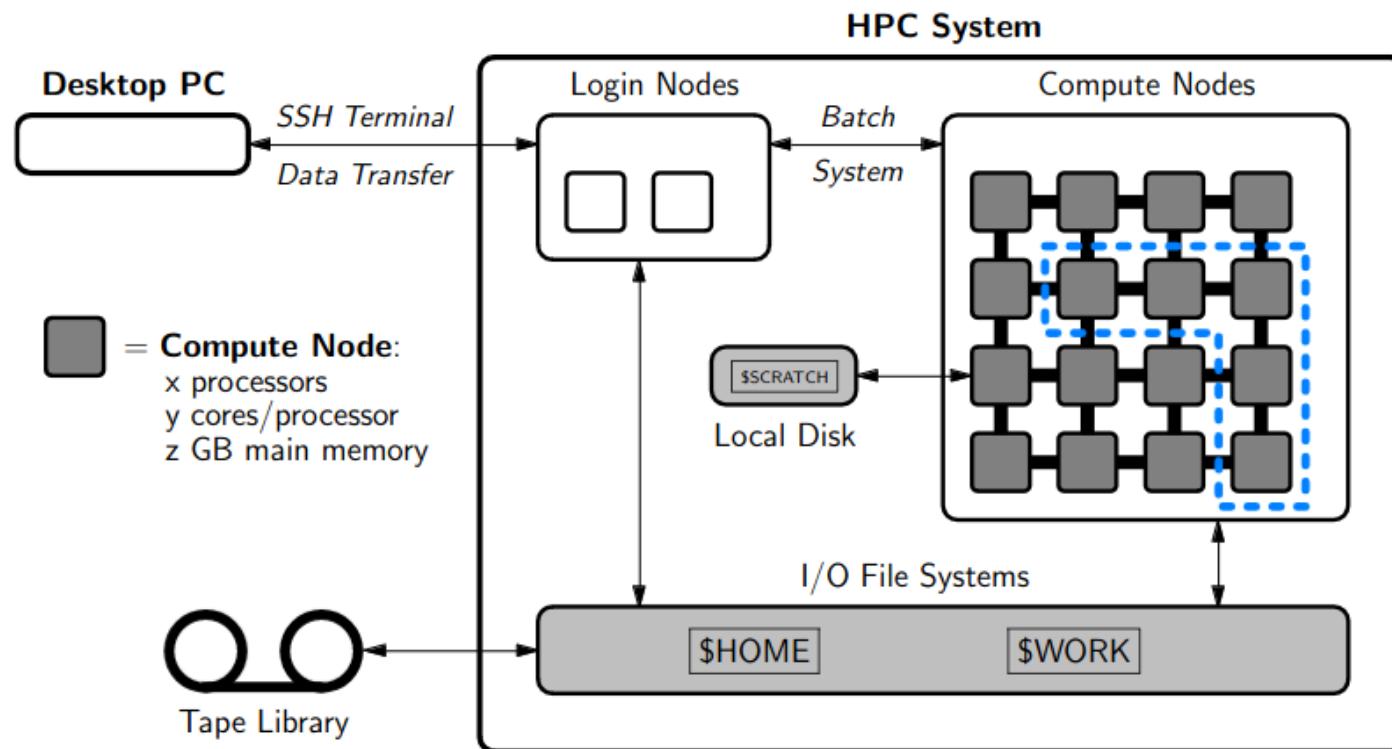
Insper Supercomputação



# Aula

- Administração de Clusters com SLURM

# Um típico ambiente de HPC



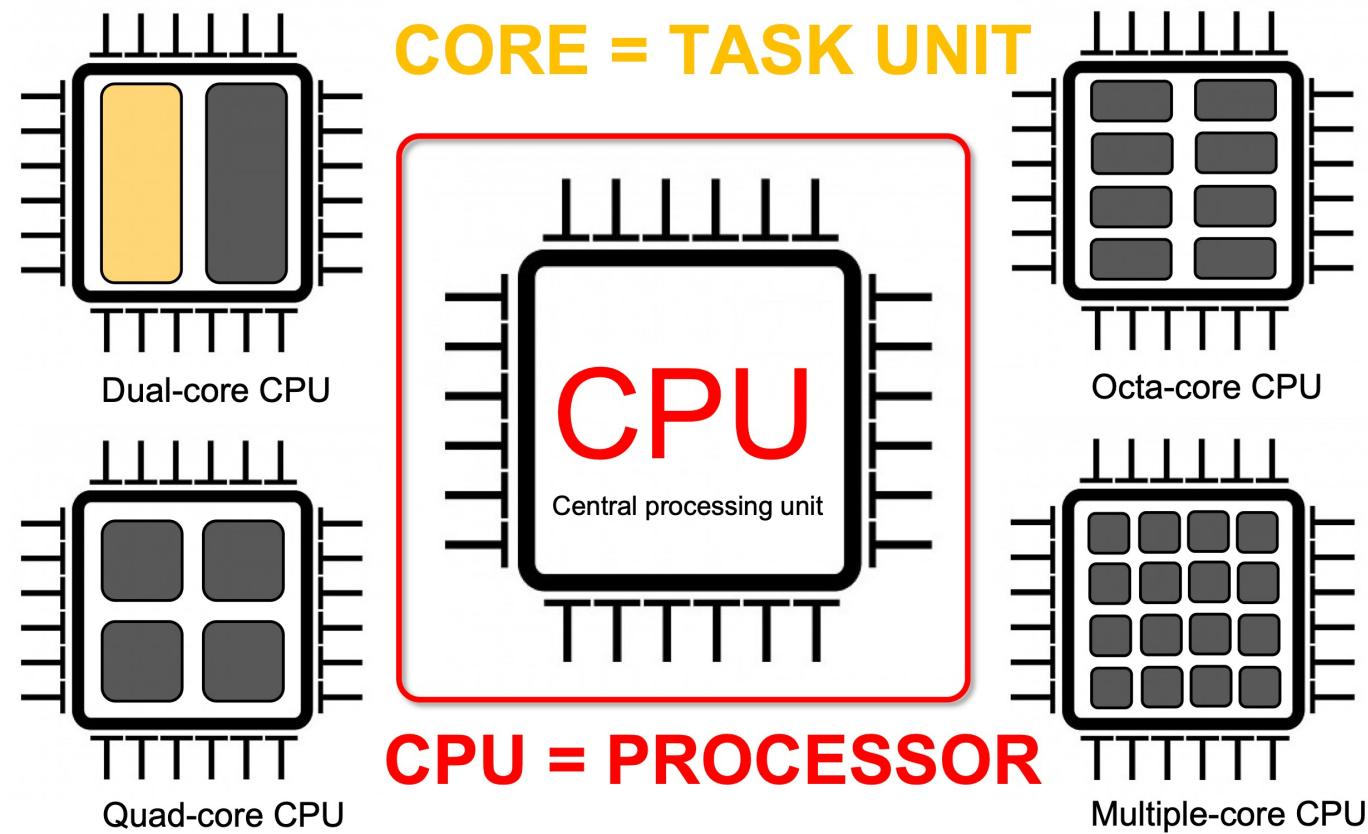
Fonte: <https://www.rz.uni-kiel.de/de/angebote/hiperf/hpc-course-12feb2020>

## **Antes de falarmos de SLURM, vamos recapitular**

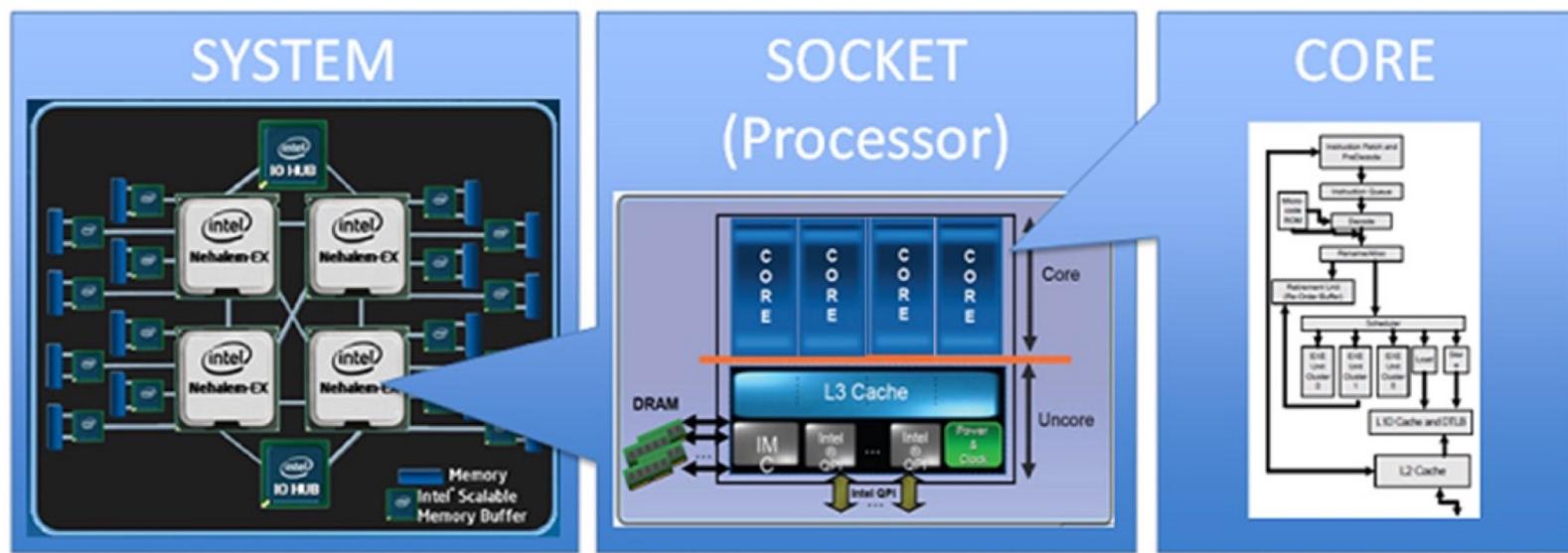
- Cores x CPU
- Como se avalia o desempenho de computadores (FLOPS)
- Clusters – como são



## Cores x CPU



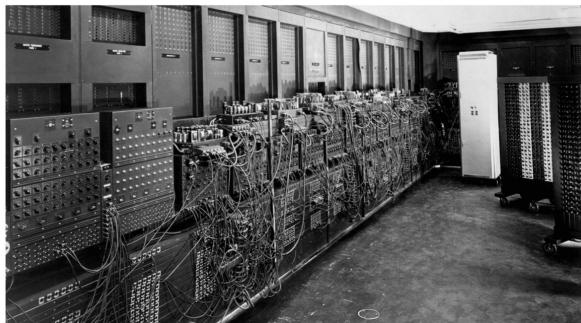
# Cores x CPU



$$4 \text{ cores} * 4 \text{ processors} = 16 \text{ total cores}$$

# Medição de desempenho

- O desempenho dos ambientes de HPC é mensurado em **Floating Point Operations Per Second (FLOPS)**



ENIAC FLOPS: 500

- Por exemplo, o primeiro desktop PC a ter desempenho em teraflops ( $10^{12}$  FLOPS), foi em 2017, o Intel i97980XE

CPU clock rate: 4.4 GHz  
CORE: 18 cores  
FLOPs per cycle: 16



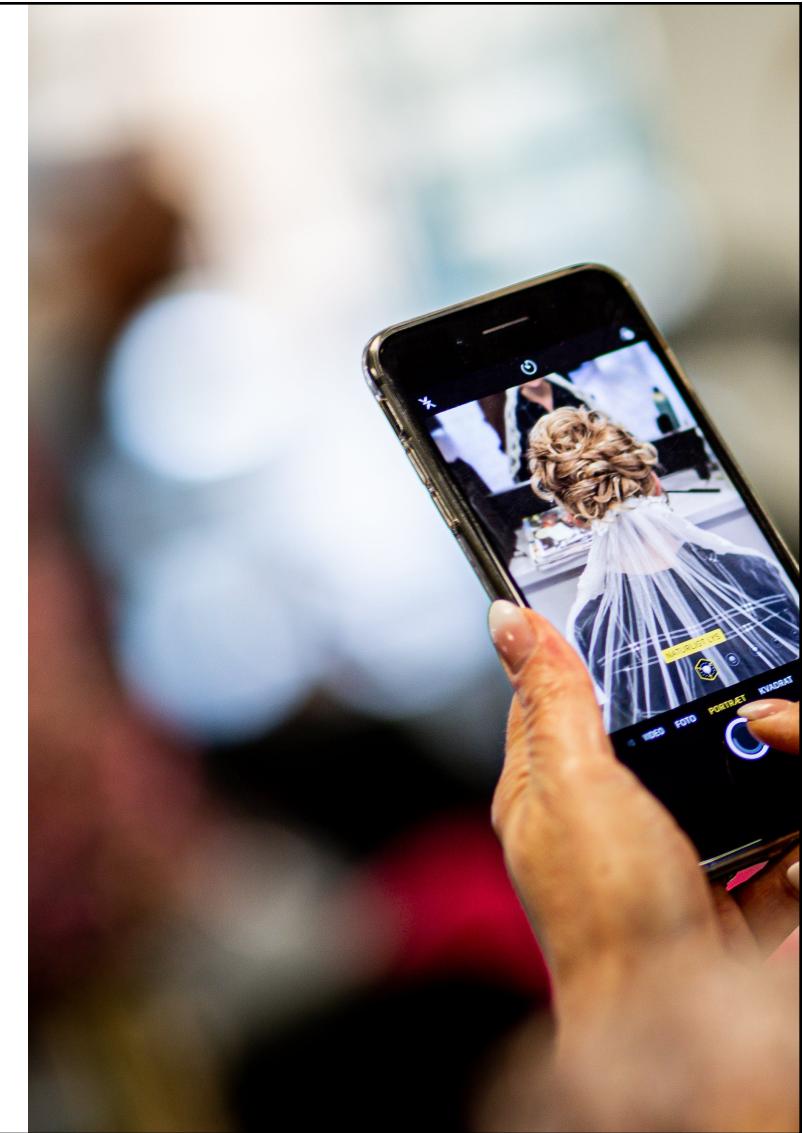
$$FLOPS = \text{cores} \times \text{clock} \times \frac{FLOPs}{cycle}$$

↓      ↓      ↓      ↓  
1267 GHz    18    4.4 GHz    16  
1.27 TFLOPS

Computer performance	
Name	FLOPS
yottaFLOPS	1024
zettaFLOPS	1021
exaFLOPS	1018
petaFLOPS	1015
teraFLOPS	1012
gigaFLOPS	109
megaFLOPS	106
kiloFLOPS	103

# Medição de desempenho

- Já te falaram que você anda com um supercomputador no bolso?
- Vamos comparar um smartphone atual com um supercomputador de mais de 20 anos
  - Chip Apple A16 Bionic, usado no iPhone 14 Pro – 17 TFLOPS
  - Ano de 1996: Fujitsu 105MHz: 0.3 TFLOPS, com 167 cores
  - Anos 2000: ASCI WHITE, SP POWER3 375MHZ: 7.3 TFLOPS, com 8.192 cores



# TOP 500 – Junho 2023



Na foto, Frontier, em ORNL (US), o maior supercomputador da atualidade

1.1 exaflops ~ $10^{18}$  cálculos por segundo

Insper

[www.insper.edu.br](http://www.insper.edu.br)

R<sub>max</sub> and R<sub>peak</sub> values are in PFlop/s. For more details about other fields, check the TOP500 description.

R<sub>peak</sub> values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

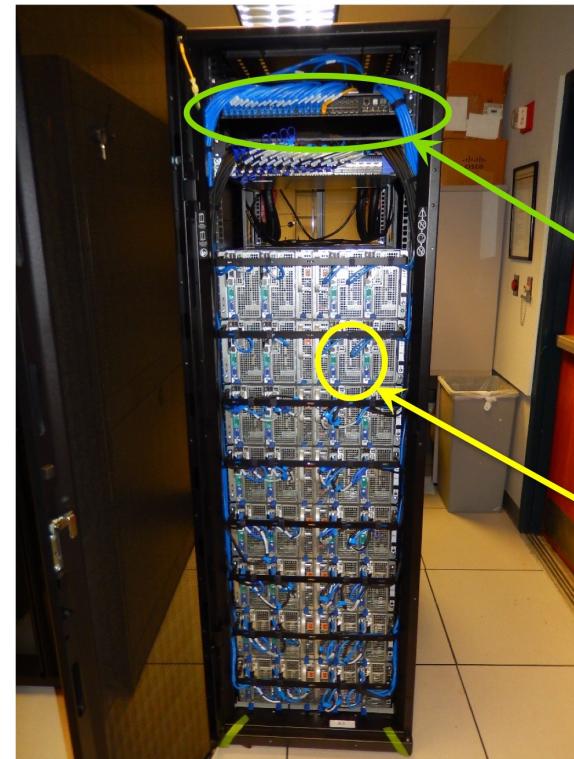
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	Leonardo - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,824,768	238.70	304.47	7,404
5	Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096

# Um típico cluster



# Um típico cluster

- Um rack

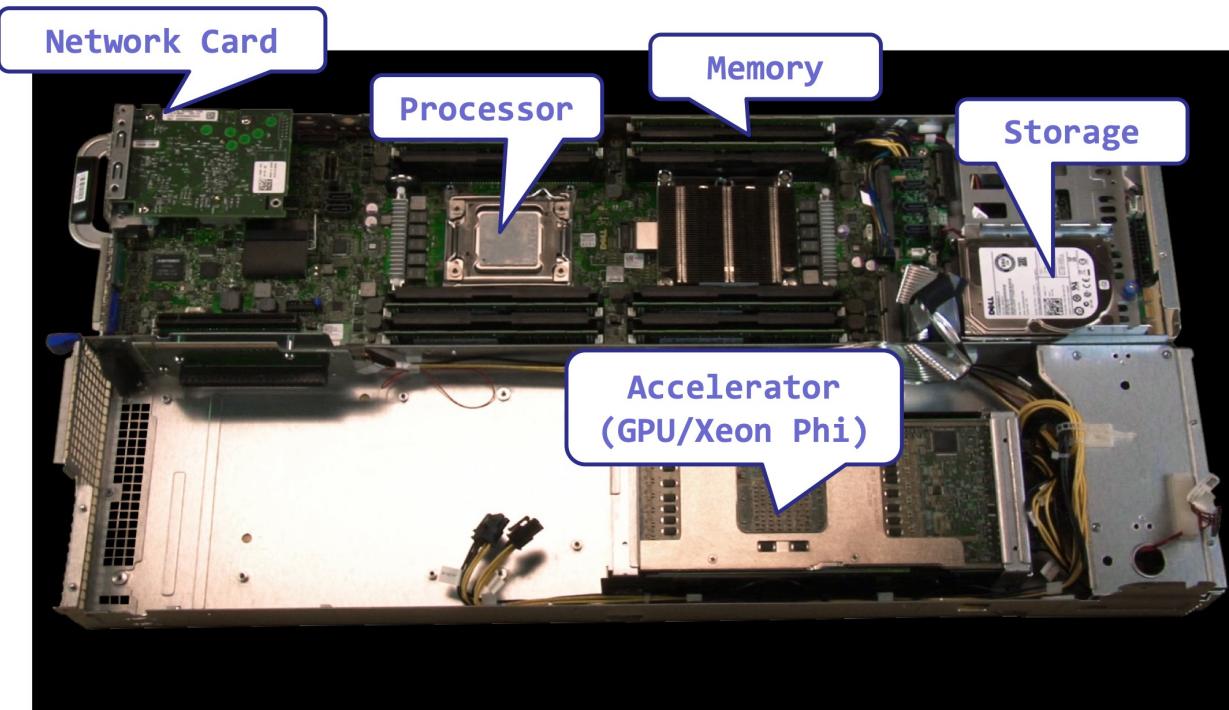


Interconnect:  
Infiniband  
Switch

Compute  
Node

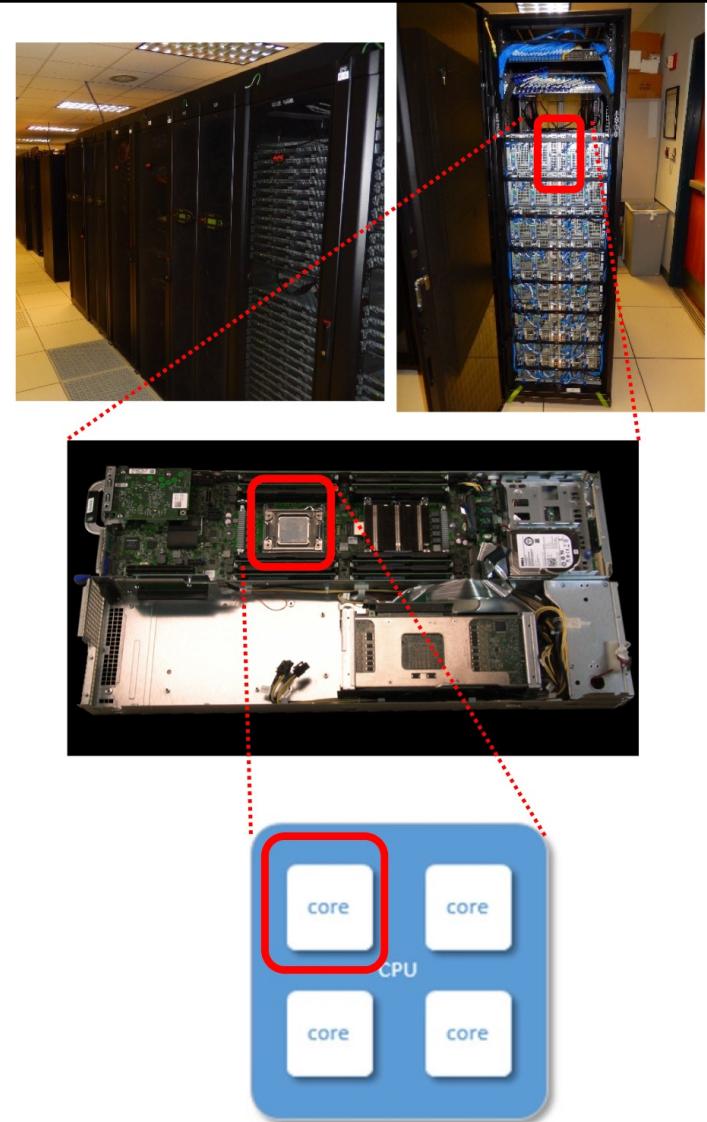
# Um típico cluster

- Um node

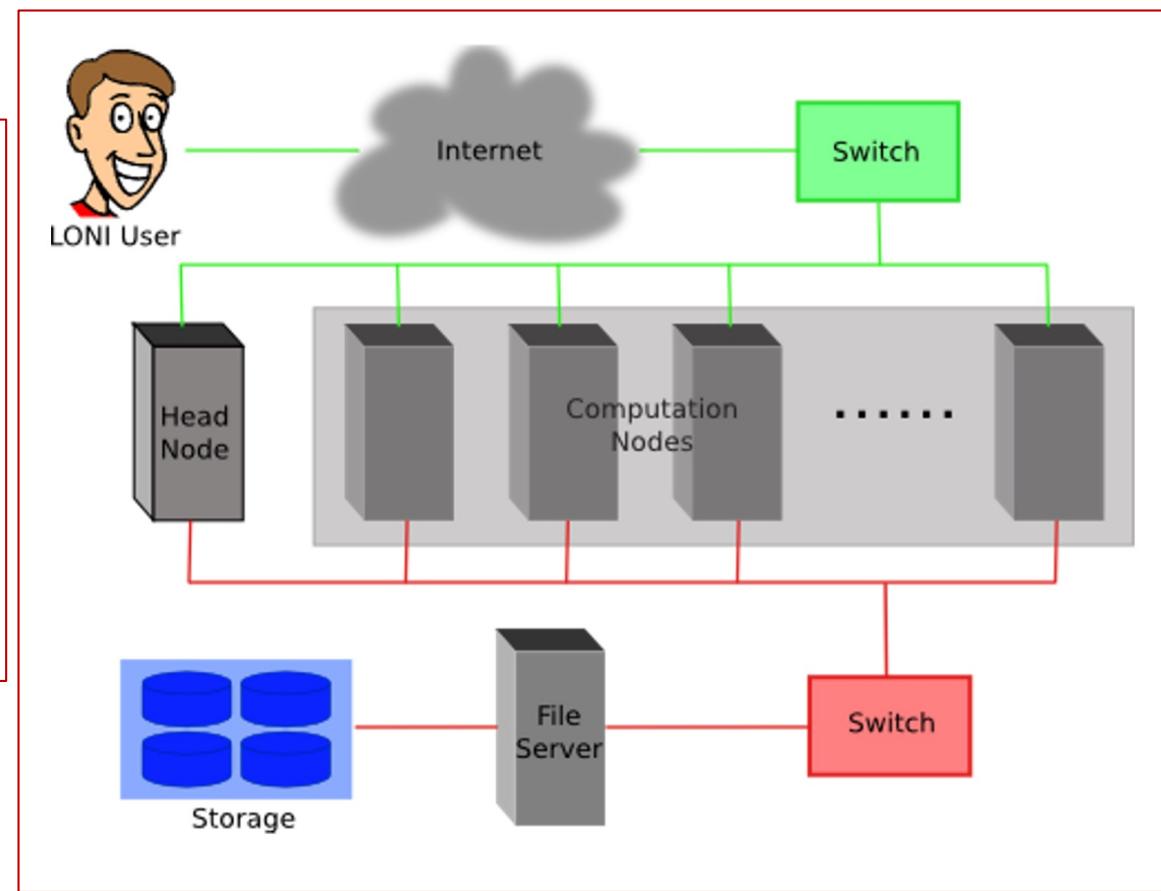
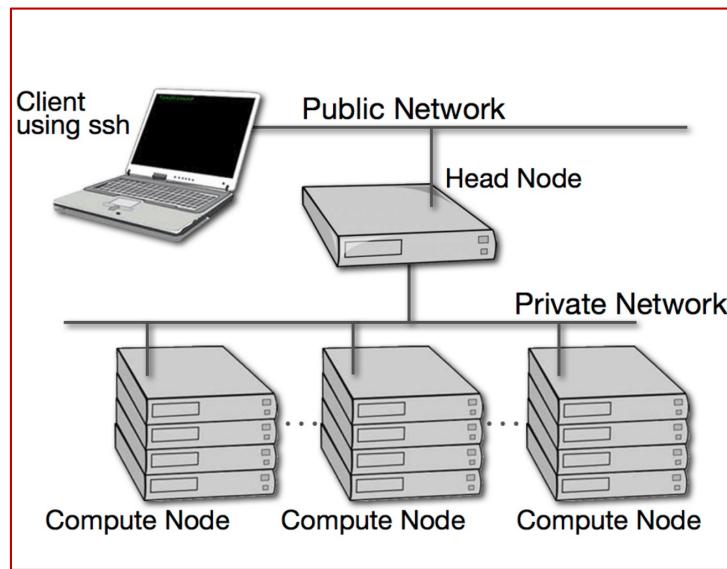


# Cluster - Terminologias

Term	Definition
<b>Cluster</b>	A set of connected computer nodes that work together. (E.g., QB2)
<b>Node</b>	A single, named host machine in the cluster. (E.g., qb010)
<b>Core</b>	The basic computation unit in a processor. (E.g. , QB2 has two 10-core processors → 20 cores)
<b>Job</b>	A user's request to use a certain amount of resources for a certain amount of time on cluster for his/her work.



# Cluster – Arquitetura convencional



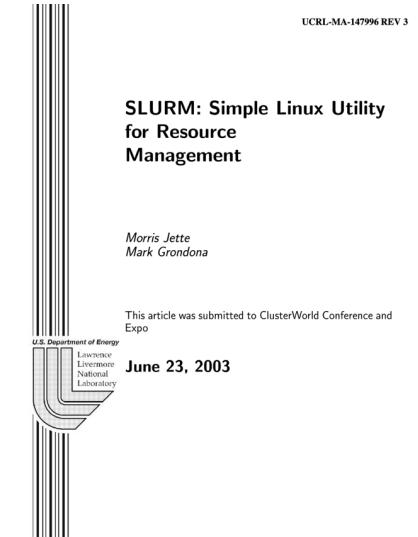
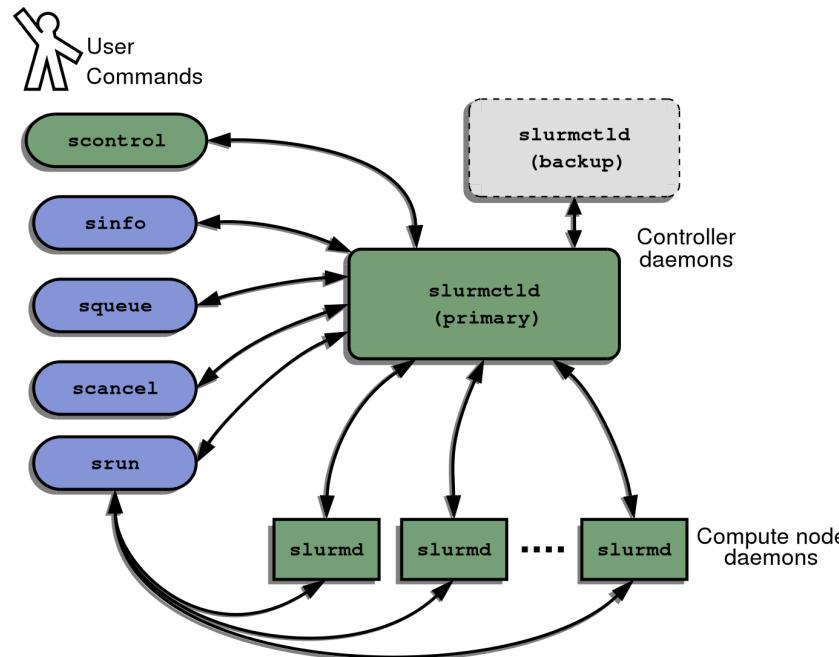
# SLURM

- É um acrônimo para “Simple Linux Utility for Resource Management”
- É um sistema de gerenciamento de filas e recursos para clusters Linux
- Sua primeira versão foi em 2003
- Amplamente adotado em ambientes acadêmicos e industriais devido a sua eficiência e simplicidade
- Desenvolvimento é contínuo com contribuições da comunidade open-source
- Atualmente é mantido e atualizado pelo SchedMD
- Aproximadamente 60% dos supercomputadores da lista TOP500 usam SLURM



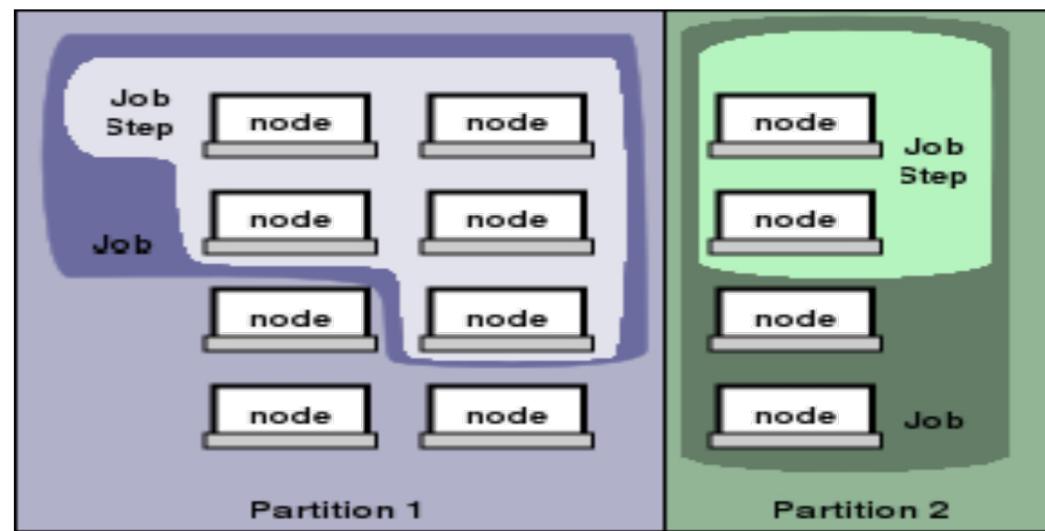
# SLURM

- O paper de apresentação do SLURM está [aqui](#)



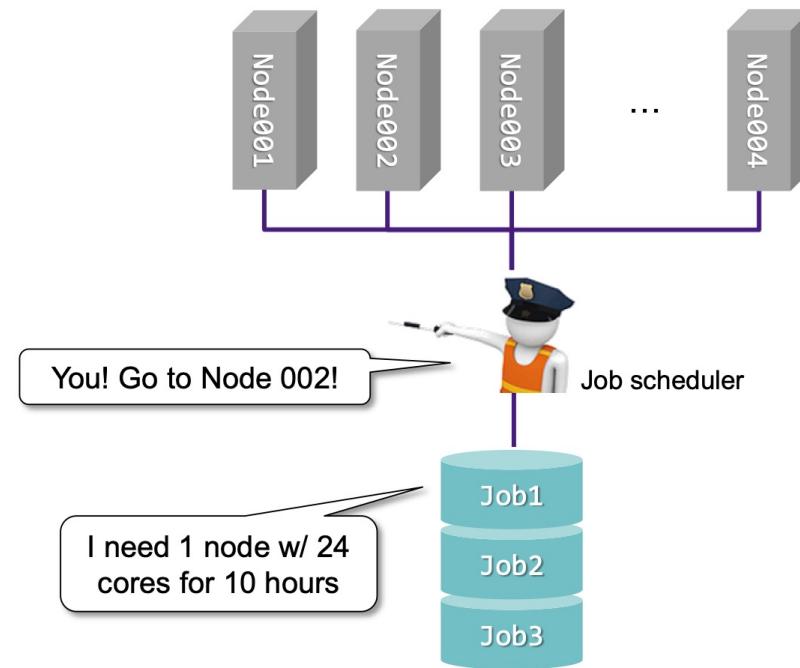
# SLURM - terminologia

- **Computing node** Computer used for the execution of programs
- **Partition** Group of nodes into logical sets
- **Job** allocation of resources assigned to a user for some time
- **Step** sets of (possible parallel) tasks with a job



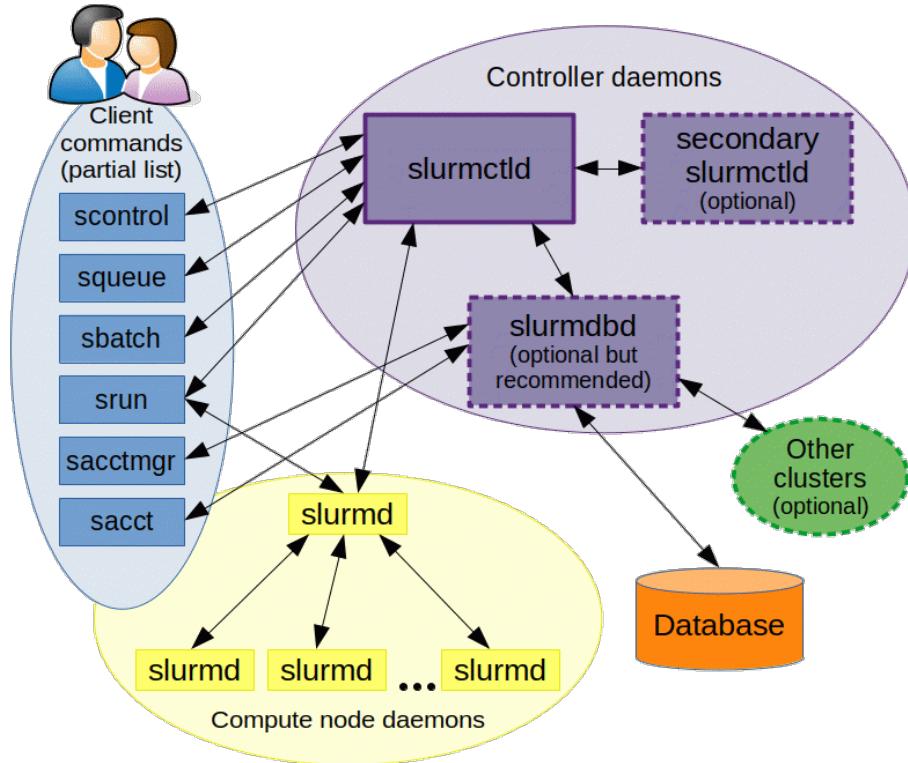
# SLURM também é um Scheduler!

- E o que é isso?



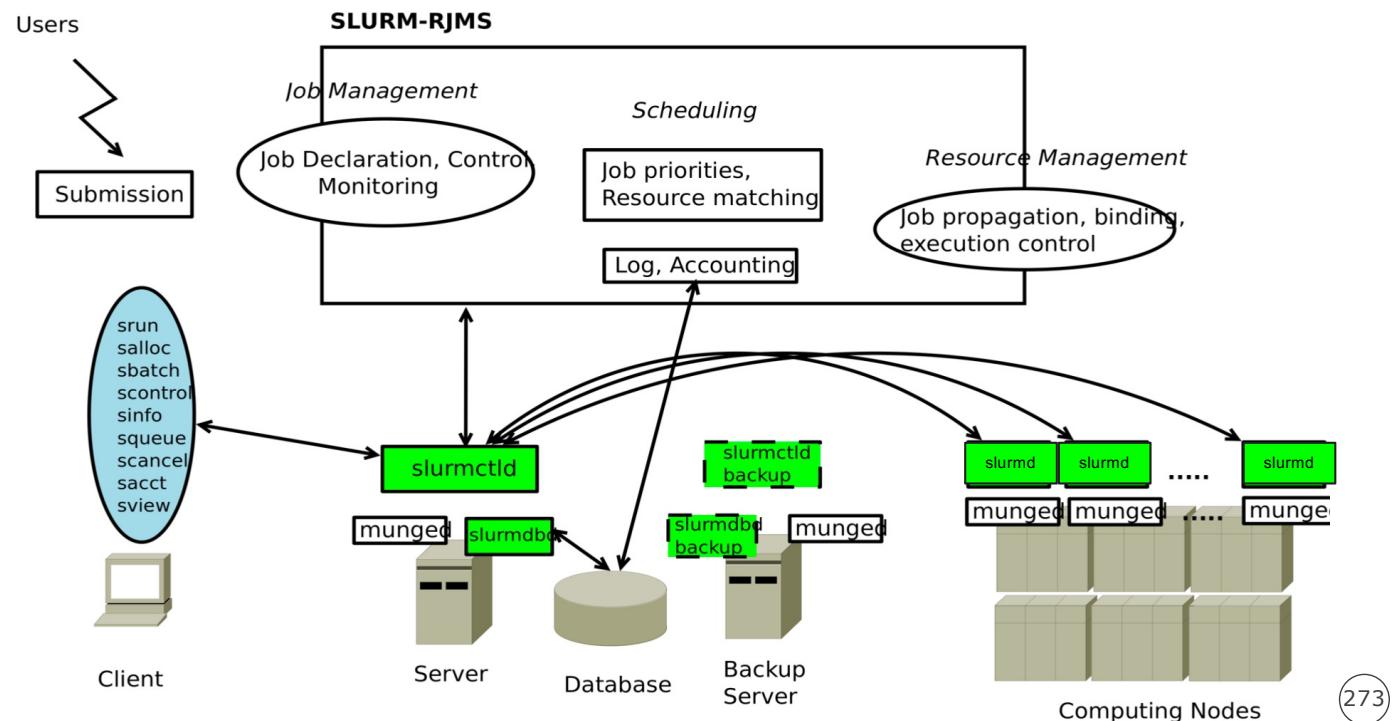
# SLURM

- Componentes



# SLURM – deamons

- Deamon – programa que executa como um processo em segundo plano



# SLURM – Arquivos de configuração

## **slurm.conf**

- Low level configuration
- Management policies
- Scheduling policies
- Allocation policies
- Node definition
- Partition definition

## **slurmdbd.conf**

- Type of persistent storage (DB)
- Location of storage

## **topology.conf**

- Switch hierarchy

## **gres.conf**

- Generic resources details
- Device files

## **cgroup.conf**

- Mount point
- Release agent path
- Cgroup subsystems parameters

	Controller	Compute node
Mandatory	slurm.conf slurmdbd.conf	slurm.conf
Optional	prologs epilogs topology.conf	gres.conf cgroup.conf topology.conf

# SLURM – Configuração de nós e partições

- É no arquivo de configuração slurm.conf onde configuramos os nós e partições (filas) do cluster

## Node definition

- Characteristics (sockets, cores, threads, memory, features)
- Network addresses

## Partition definition

- Set of nodes
- Sharing
- Priority/preemption

```
# Compute Nodes
NodeName=cuzco[1-10] Procs=16 Sockets=2 CoresPerSocket=8 ThreadsPerCore=1 State=UNKNOWN RealMemory=38000
NodeName=cuzco[10-20] Procs=32 Sockets=2 CoresPerSocket=8 ThreadsPerCore=2 State=UNKNOWN RealMemory=46000

# Partitioning
PartitionName=exclusive Nodes=cuzco[1-20] MaxTime=INFINITE State=UP Priority=10 Shared=Exclusive
PartitionName=shared Nodes=berlin[1-20] Default=YES MaxTime=INFINITE State=UP Priority=30
PartitionName=procs16 Nodes=berlin[1-10] MaxTime=INFINITE State=UP Priority=30
PartitionName=procs32 Nodes=berlin[10-20] MaxTime=INFINITE State=UP Priority=30
```

## Shared Option

Controls the ability of the partition to execute more than one job on a resource (node, socket, core)

**EXCLUSIVE** allocates entire node (overrides cons\_res ability to allocate cores and sockets to multiple jobs)

**NO** sharing of any resource.

**YES** all resources can be shared, unless user specifies --exclusive on srun | salloc | sbatch

# Como o SLURM prioriza?



# Como o SLURM prioriza?

- Por default, FIFO
- Mas há alguns outros critérios, como:

**Age** - the length of time a job has been waiting in the queue, eligible to be scheduled

**Fairshare** - the difference between the portion of the computing resource that has been promised and the amount of resources that has been consumed

**Job size** - the number of nodes or CPUs a job is allocated

**Partition** - a factor associated with each node partition

**QOS** - A factor associated with each Quality Of Service

## Isso tem a ver com algoritmos de escalonamento

FIFO Scheduling



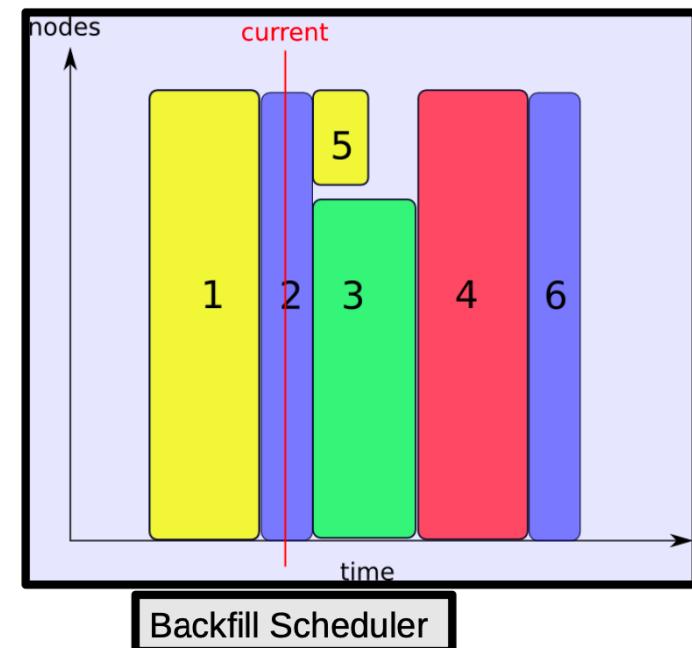
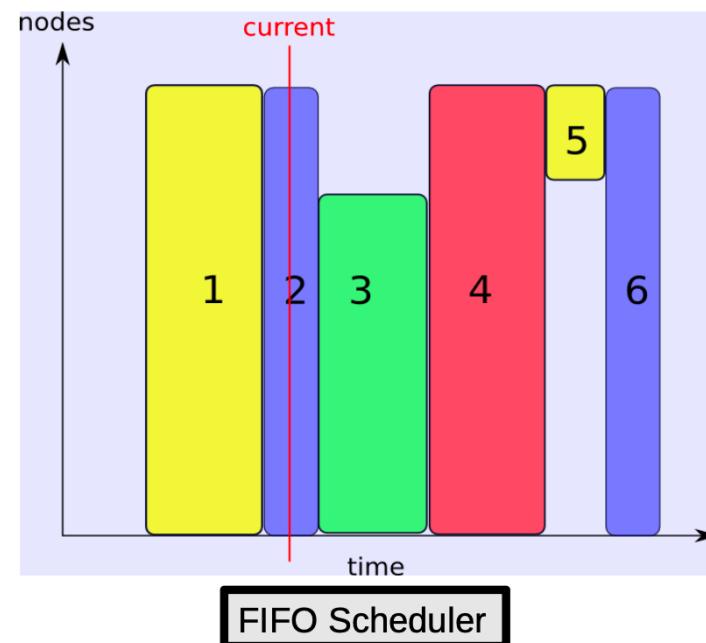
Backfill Scheduling

- Job priority
- Time limit (**Important!**)



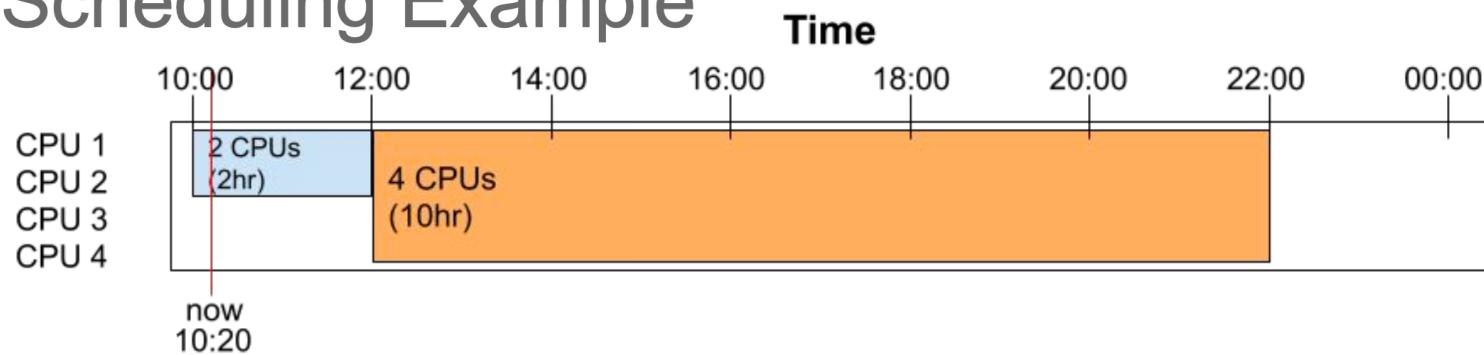
# Backfill Scheduler

Holes can be filled if previous jobs order is not changed



# Backfill Scheduler

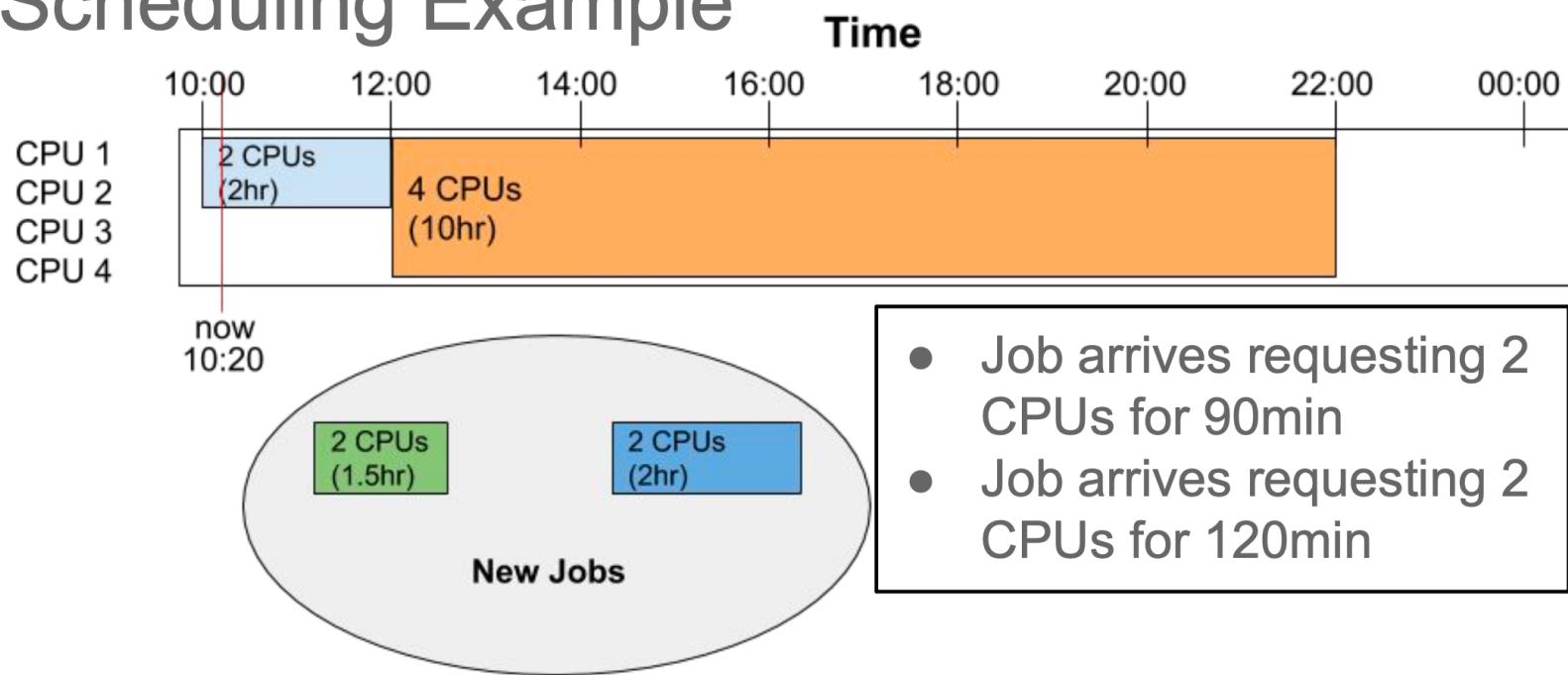
## Scheduling Example



- 2 CPU job running until noon
- 4 CPU job starts at noon

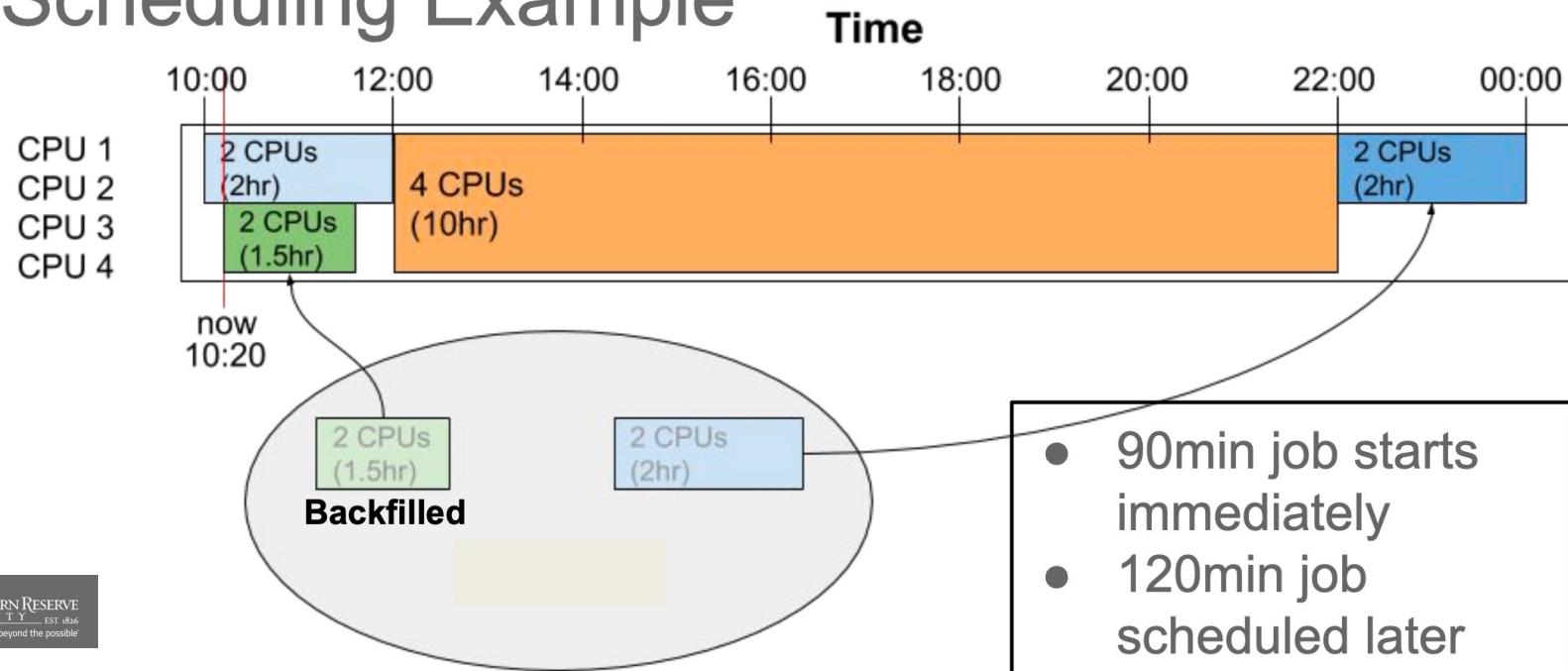
# Backfill Scheduler

## Scheduling Example



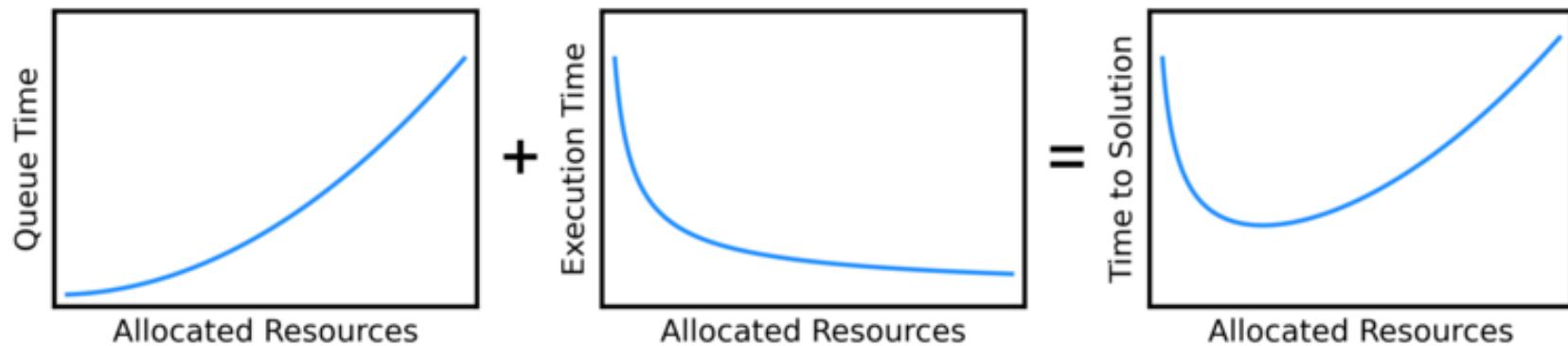
# Backfill Scheduler

## Scheduling Example

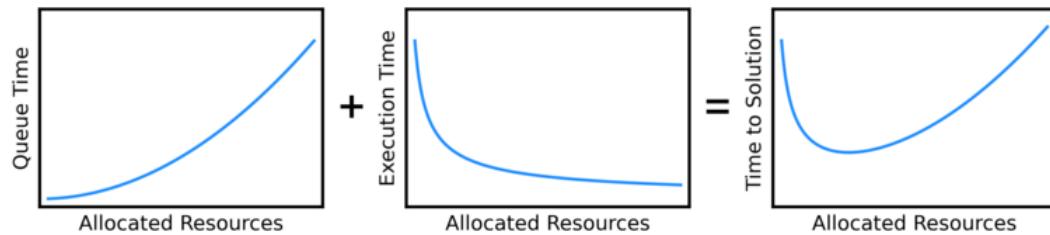


# Quanto tempo vai demorar meu job?

- Uma regra simples é escolher o menor conjunto de recursos que proporcione uma aceleração razoável em relação ao baseline



# Não é sobre pedir mais, é sobre pedir corretamente!



Ao lado temos um programa OpenMP em um cluster. Conforme aumentamos as threads (cpus-per-task) vamos melhorando o speedup. Mas veja que a partir de 16 threads, nossa eficiência diminui.

ntasks	cpus-per-task	execution time	speed-up ratio	parallel efficiency
1	1	42.0	1.0	100%
1	2	22.0	1.9	95%
1	4	16.0	2.6	66%
1	8	7.8	5.4	67%
1	16	6.5	6.5	40%
1	32	7.1	5.9	18%

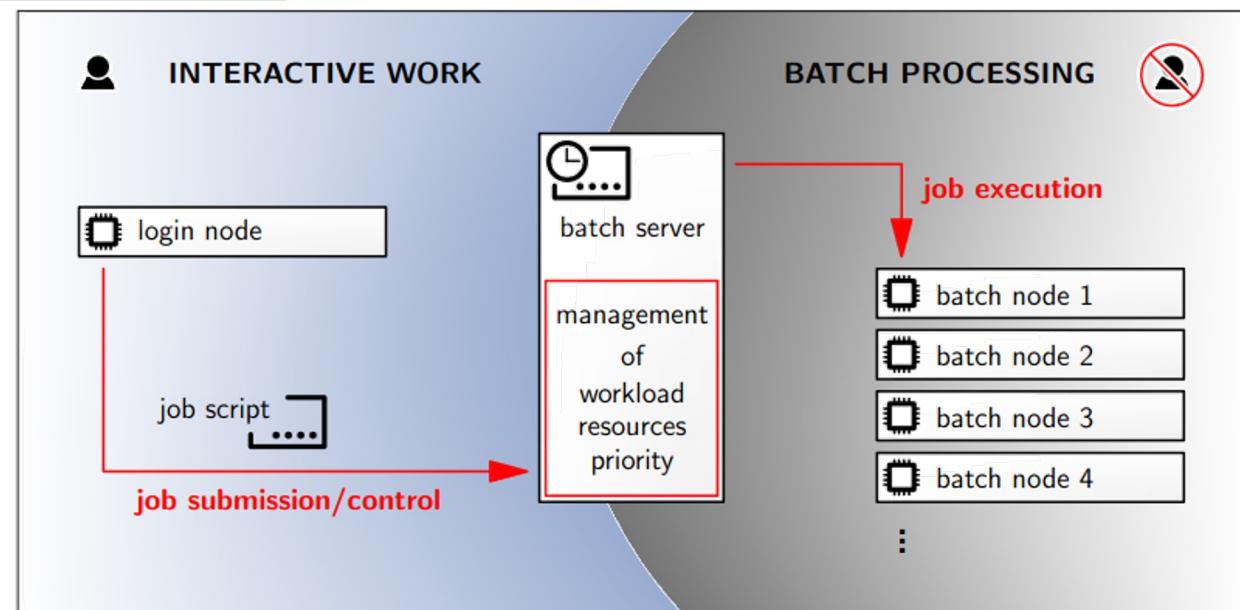
# Jobs interativos e Jobs em Batch

## 1) Interactive job

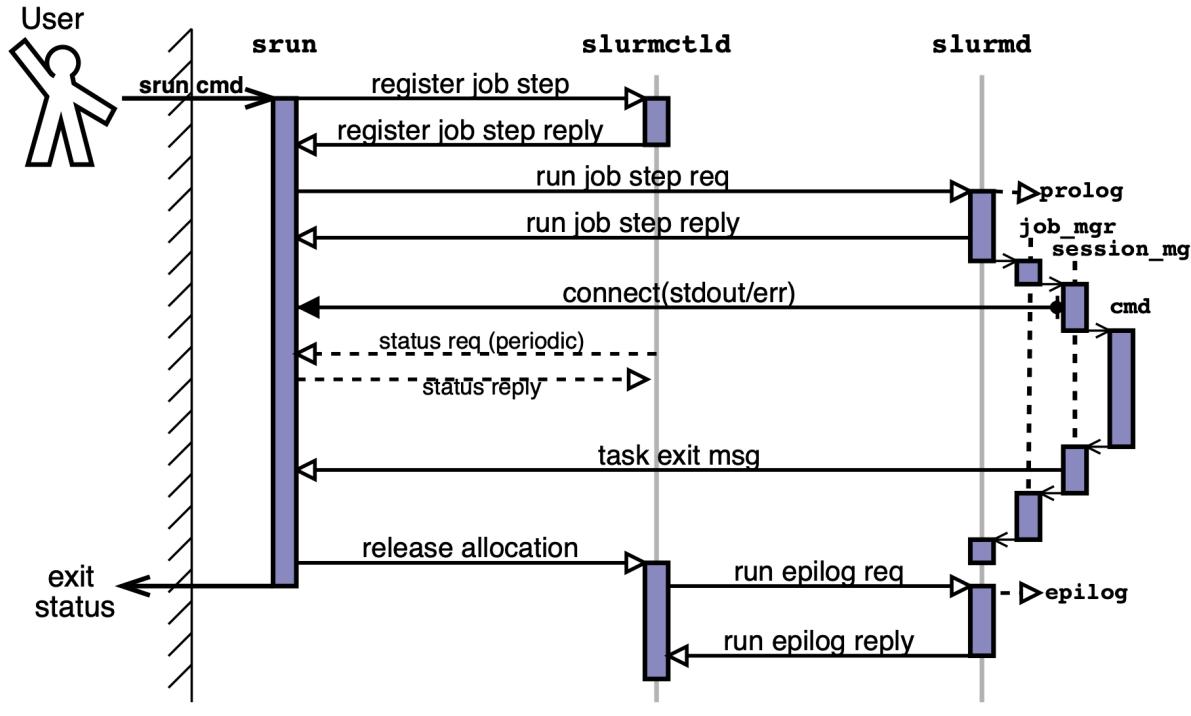
- Runs **in terminal** (just like using a local machine)
- **Can interact** with the job while running

## 2) Batch job

- Submit to server and runs **by itself**, until finished or error
- **Cannot interact** with the job while running

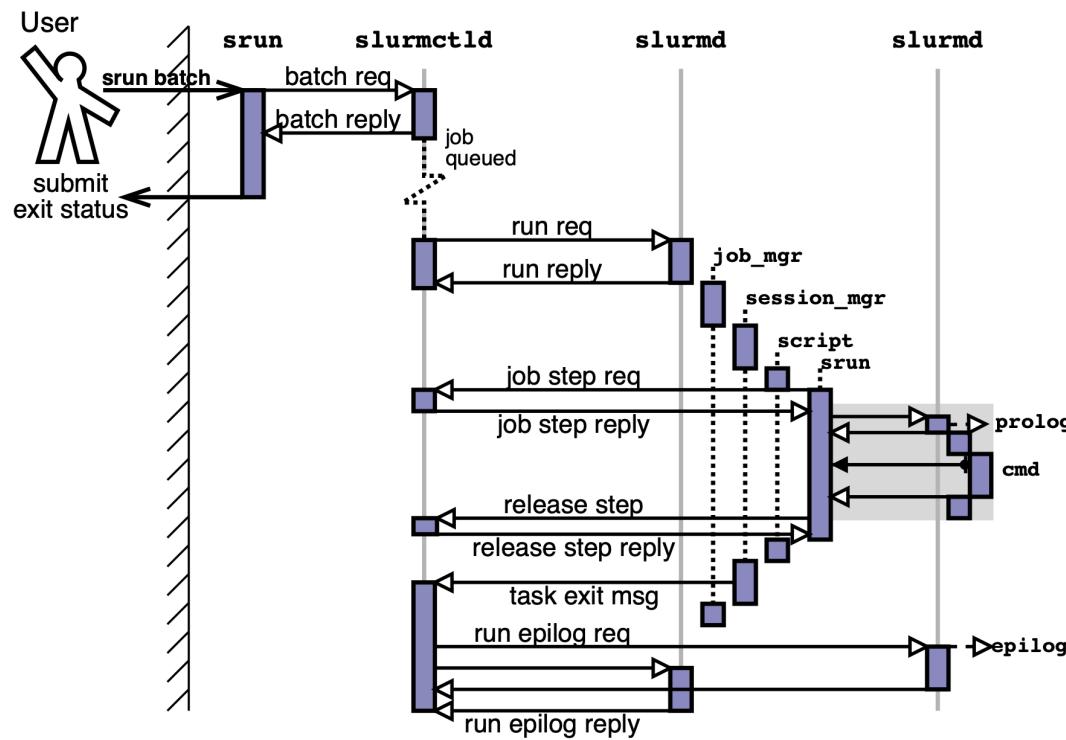


# Jobs Interativos



**Fig. 5.** Interactive job initiation. **srun** simultaneously allocates nodes and a job step from **slurmctld** then sends a run request to all **slurmds** in job. Dashed arrows indicate a periodic request that may or may not occur during the lifetime of the job

# Jobs Batch



**Fig. 6.** Queued job initiation. `slurmctld` initiates the user's job as a batch script on one node. Batch script contains an `srun` call that initiates parallel tasks after instantiating job step with controller. The shaded region is a compressed representation and is shown in more detail in the interactive diagram (Figure 5)

Inspire

UCRL-MA-147996 REV 3

SLURM: Simple Linux Utility  
for Resource  
Management

Morris Jette  
Mark Grondona

This article was submitted to ClusterWorld Conference and  
Expo

June 23, 2003

Approved for public release; further dissemination unlimited

287

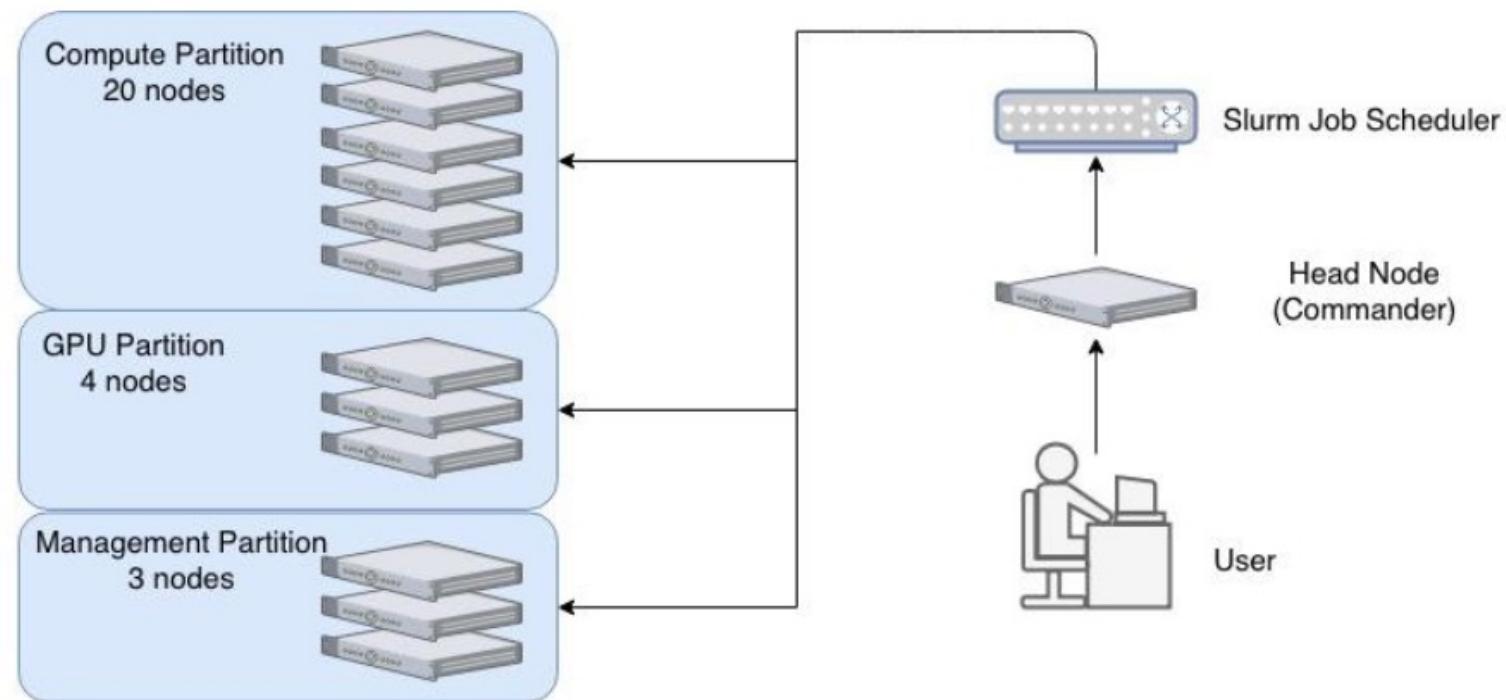
# Interativo ou em Batch? R: Use os dois!

Running on HPC: Easy as 1-2-3

1. Plan ahead	2. Test interactively	3. Run remotely
<ul style="list-style-type: none"><li>▪ Organize project directories</li><li>▪ Install software, transfer data</li><li>▪ Think about your job requirements<ul style="list-style-type: none"><li>- amount of memory</li><li>- number of i/o files</li><li>- size of data</li></ul></li></ul>	<ul style="list-style-type: none"><li>▪ Use salloc/srun<ul style="list-style-type: none"><li>- test run commands</li><li>- check paths, results</li></ul></li><li>▪ Experiment with resource allocation<ul style="list-style-type: none"><li>- number cores (cpus)</li><li>- amount of memory</li></ul></li><li>▪ Start small!<ul style="list-style-type: none"><li>- <i>then scale</i></li></ul></li></ul>	<ul style="list-style-type: none"><li>▪ Create Slurm script<ul style="list-style-type: none"><li>e.g., myjob.slurm</li></ul></li><li>▪ Submit job to queue<ul style="list-style-type: none"><li>\$sbatch myjob.slurm</li></ul></li><li>▪ Monitor your job<ul style="list-style-type: none"><li>\$squeue --user &lt;uname&gt;</li></ul></li><li>▪ Check results<ul style="list-style-type: none"><li>\$less slurm-&lt;jobid&gt;.out</li></ul></li></ul>

[https://chianglab.usc.edu/doc/Intro\\_to\\_Slurm.pdf](https://chianglab.usc.edu/doc/Intro_to_Slurm.pdf)

# Partições (FILAS)



# Principais comandos do SLURM - Interativo

Useful SLURM Commands		
Command	Summary	Example
srun <code>-parameter job</code>	Obtain a job allocation and execute an application (see Running a Program - SRUN)	srun <code>-N1 -n1 example.py</code>
sbatch <code>-parameter batchscript</code>	Submit a batch script for later execution (see Running a Program - SBATCH)	sbatch <code>-o example.sh</code>
sacct	Display accounting data Use <code>-j</code> jobid to see status of specific job	sacct <code>-j 1576</code>
sinfo	View the status of the cluster's nodes and partitions	sinfo
squeue	Displays information of jobs in queue	squeue

# Principais comandos do SLURM - Batch

Common Batch Script Directives		
Directive	Description	Example
--job-name= <code>name</code> or -J <code>name</code>	Custom job name	--job-name= <code>example</code>
--partition= <code>name</code> or -p <code>partition</code>	Partition to run on	--partition= <code>compute</code>
--nodes=# or -N#	Total number of nodes	--nodes=1
--ntasks=# or -n#	Number of "tasks". For use with distributed parallelism. See below.	--ntasks=1
--cpus-per-task=# or -c#	# of CPUs allocated to each task. For use with shared memory parallelism.	--cpus-per-task=1
--ntasks-per-node=#	Number of "tasks" per node. For use with distributed parallelism. See below.	--ntasks-per-node=2
--time=[ [DD-]HH:]MM:SS or -t [ [DD-]HH:]MM:SS	Maximum walltime of the job in Days-Hours:Minutes:Sec	--time=10:00 10 minutes
--mem=#	Memory requested per node in MB	--mem=1G

# Resumo de comandos do SLURM

- Disponível [aqui](#)

**slurm**  
workload manager

**Job Submission**

**salloc** - Obtain a job allocation.  
**sbatch** - Submit a batch script for later execution.  
**svbatch** - Obtain a job allocation (as needed) and execute an application.

**-array <index>** (e.g. "-array=1-10") Job array specification (batch command only)

**-account <name>** Account to be charged for resources used.

**-begin <time>** (e.g. "-begin=18:00:00") Initiate job after specified time.

**-clusters <name>** Cluster(s) to run the job. (batch command only)

**-constraint <features>** Required node features.

**-cpus-per-task <count>** Number of CPUs required per task.

**-dependency <state>:<jobid>** Defer job until specified jobs reach specified state.

**-error <filename>** File in which to store job error messages.

**-exclude <names>** Specific host names to exclude from job allocation.

**-exclusive <user>** Allocated nodes can not be shared with other users/users.

**-export <name>=<value>** Export identified environment variables.

**-gres <name>[<count>]** Generic resources required per node.

**-input <name>** File from which to read job input data.

**-job-name <name>** Job name.

**-label** Prepend task ID to output. (from command only)

**-licenses <name>[<count>]** License resources required for entire job.

**slurm** - View and modify account information.

**Options:**

- mem=<MB> Memory required per node.
- mem-per-cpu=<MB> Memory required per allocated CPU.
- N<minnodes>[<maxnodes>] Node count required for the job.
- n<count> Number of tasks to be launched.
- nodelist=<names> Specific host names to include in job allocation.
- output <name> File in which to store job output.
- partition=<names> Partition/queue in which to run the job.
- qos=<name> Quality Of Service.
- signal=[B:]<num>[@<time>] Signal job at approaching time limit.
- time=<time> Wall clock time limit.
- wrap=<command> String Wrap specified command in a simple "sh" shell. (batch command only)

**Accounting**

**sacct** - Display accounting data.

**Job Management**

**sbcast** - Transfer file to a job's compute nodes.

**slurm [options] SOURCE DESTINATION**

**Commands:**

- force Replace previously existing file.
- preserve Preserve modification times, access times, and access permissions.

**scancel** - Signal jobs, job arrays, and/or job steps.

**Commands:**

- account=<name> Operate only on jobs charging the specified account.
- name=<name> Operate only on jobs with specified name(s).
- partition=<names> Comma separated list of partitions to select job and job steps from.
- state=<state> List Display jobs with specified states.
- starttime=<time> Start of reporting period.

**sinfo** - View information about nodes and partitions.

**Environment Variables**

- all Display information about all partitions.
- dead If set, only report state information for non responding (dead) nodes.

**SchedMD** SchedMD Support and Development

**slurmctmgr** - View and modify account information.

**Options:**

- immediate Commit changes immediately.
- parsable Output delimited by \t

**Commands:**

- add <ENTITY> [<SPEC> ... <ENTITY> [<SPEC> ...] Add an entity. Identical to the create command.
- delete <ENTITY> where [<SPEC> ...] Delete the specified entities.
- list <ENTITY> [<SPEC> ...] Display information about the specific entity.
- modify <ENTITY> where [<SPEC> ...] Modify an entity.

**Entities:**

- account Account associated with job.
- cluster Cluster/queue parameter in the slurm config.
- qos Quality of Service.
- user User name in system.

**squeue** - View information about jobs.

**Commands:**

- account=<name> View only jobs with specified accounts.
- clusters=<name> View jobs on specified clusters.
- format=<spec> ... (e.g. "-format=%d %q") Output format to display. Specify fields, size, order, etc.
- jobs=<job\_id> ... Commas separated list of job IDs to display.
- name=<name> View only jobs with specified names.
- partition=<names> View only jobs in specified partitions.
- priority Sort jobs by priority.
- qos=<name> View only jobs with specified Qualities Of Service.
- start Report the expected start time and resources to be allocated for pending jobs in order of increasing start time.
- state=<names> View only jobs with specified states.
- users=<names> View only jobs for specified users.

**slurmcontrol** - Used view and modify configuration and state. Also see the **slurm** graphical user interface version.

**Commands:**

- details Make show command print more details.
- oneliner Print information on one line.

**Commands:**

- create <SPECIFICATION> Create a new partition or queue.
- delete <SPECIFICATION> Delete the entry with the specified SPECIFICATION.
- reconfigure All Slurm daemons will re-read the configuration file.
- request JOB\_LIST Request a running, suspended or completed batch job.
- show ENTITY\_ID Display the state of the specified entity with the specified identification.
- update <SPECIFICATION> Update job, step, node, partition, or reservation configuration per the supplied specification.

**Environment Variables**

- all Set to the job ID if part of a job array.
- dead If set, only report state information for non responding (dead) nodes.

**Daemons**

- slurmd Executes on cluster's "head" node to manage workload.
- slurmd Execute on each compute node to locally manage resources.
- slurmdbd Manages database of resources limits, licenses, and archives accounting records.

**SchedMD** SchedMD Support and Development

Copyright 2017 SchedMD LLC. All rights reserved.  
<http://www.schedmd.com>

# Atividade

- Em nosso cluster:
  - 1 – Aprenderemos a verificar o status do cluster (sinfo, scontrol)
  - 2 – Criaremos uma nova partição (fila) para permitir que Jobs executem no máximo por 5 minutos e usando apenas 1 CPU (chamaremos essa fila de express) [modificaremos o /etc/slurm/slurm.conf no master]
  - 3 – Ampliaremos a memória disponível ao SLURM nos nós de computação
  - 4 - Submeteremos um job interativo (srun)
  - 5 – Submeteremos alguns Jobs em batch (sbatch) e monitorar a fila (squeue), além de realizar alguns comandos administrativos (scancel, shold, sactt)
  - 6 – Monitoraremos o desempenho no Ganglia

# **Parte prática**



# Inicialização

1. Inicialize as máquinas virtuais (comece pela smshost)
2. Configure o redirect da porta 8080 da sua máquina física para a 80 da sms-host
3. Vá no diretório onde há os arquivos do vagrant, e execute **vagrant ssh**. Com isso, você deve logar na máquina sms

smshost\_for\_openhpc\_virtual-lab-run1 - Settings

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
ssh	TCP	127.0.0.1	2229		22
web	TCP	127.0.0.1	8080		80

```
cluster@cluster-NUC7i5DNHE:~/VagrantCluster$ vagrant ssh
Last login: Thu Nov  2 20:07:02 2023 from 10.0.2.2
This system is built by the Bento project by Chef Software
More information can be found at https://github.com/chef/bento
[vagrant@sms-host ~]$ sudo su
[root@sms-host vagrant]#
```

# GIT Clone

- Alguns de nossos arquivos estão hospedados no git.
- Execute um clone na máquina master

```
git clone https://github.com/andrefmb/slurm_pratica.git
```

# Verificar deamons

- Veja se o deamon **slurmctld** está executando

```
[root@sms-host vagrant]# systemctl status slurmctld
● slurmctld.service - Slurm controller daemon
  Loaded: loaded (/usr/lib/systemd/system/slurmctld.service; enabled; vendor preset: disabled)
  Active: active (running) since Sáb 2023-11-04 11:50:37 UTC; 11min ago
    Process: 1246 ExecStart=/usr/sbin/slurmctld $SLURMCTLD_OPTIONS (code=exited, status=0/SUCCESS)
   Main PID: 1305 (slurmctld)
     CGroup: /system.slice/slurmctld.service
             └─1305 /usr/sbin/slurmctld
```

# Verificar deamons

- Veja agora, com o auxílio do **pdsh**, se o deamon **slurmd** está executando nos nós.

```
[root@sms-host vagrant]# pdsh -w compute0[0-1] systemctl status slurmd
compute01: ● slurmd.service - Slurm node daemon
compute01:   Loaded: loaded (/usr/lib/systemd/system/slurmd.service; disabled; vendor preset: disabled)
compute01:   Active: inactive (dead)
pdsh@sms-host: compute01: ssh exited with exit code 3
compute00: ● slurmd.service - Slurm node daemon
compute00:   Loaded: loaded (/usr/lib/systemd/system/slurmd.service; disabled; vendor preset: disabled)
compute00:   Active: inactive (dead)
pdsh@sms-host: compute00: ssh exited with exit code 3
[root@sms-host vagrant]#
```

- Nesse caso, não está. Então devemos executar o comando abaixo

```
[root@sms-host vagrant]# pdsh -w compute0[0-1] systemctl start slurmd
```

# Verificar deamons

- Execute novamente para ver o resultado

```
[root@sms-host vagrant]# pdsh -w compute0[0-1] systemctl status slurmd
compute00: ● slurmd.service - Slurm node daemon
compute00:   Loaded: loaded (/usr/lib/systemd/system/slurmd.service; disabled; vendor preset: disabled)
compute00:   Active: active (running) since Sáb 2023-11-04 11:59:16 UTC; 3s ago
compute00:     Process: 1787 ExecStart=/usr/sbin/slurmd $SLURMD_OPTIONS (code=exited, status=0/SUCCESS)
compute00:    Main PID: 1789 (slurmd)
compute00:      Tasks: 1
compute00:     Memory: 1.0M
compute00:       CGroup: /system.slice/slurmd.service
compute00:             └─1789 /usr/sbin/slurmd
compute00:
compute00: Nov 04 11:59:16 compute00.hpcnet systemd[1]: Starting Slurm node daemon...
compute00: Nov 04 11:59:16 compute00.hpcnet systemd[1]: Started Slurm node daemon.
compute01: ● slurmd.service - Slurm node daemon
compute01:   Loaded: loaded (/usr/lib/systemd/system/slurmd.service; disabled; vendor preset: disabled)
compute01:   Active: active (running) since Sáb 2023-11-04 11:59:17 UTC; 3s ago
compute01:     Process: 1808 ExecStart=/usr/sbin/slurmd $SLURMD_OPTIONS (code=exited, status=0/SUCCESS)
compute01:    Main PID: 1810 (slurmd)
compute01:      Tasks: 1
compute01:     Memory: 1.0M
compute01:       CGroup: /system.slice/slurmd.service
compute01:             └─1810 /usr/sbin/slurmd
compute01:
compute01: Nov 04 11:59:16 compute01.hpcnet systemd[1]: Starting Slurm node daemon...
compute01: Nov 04 11:59:17 compute01.hpcnet systemd[1]: Started Slurm node daemon.
[root@sms-host vagrant]# █
```

# Verifique o status do cluster

- Verifique o status do cluster com `sinfo`

```
[root@sms-host vagrant]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up    1-00:00:00      2  down  compute[00-01]
```

- Veja que o nó está com o STATE = DOWN. Isso provavelmente se dá porque o controlador do SLURM não atualizou o status do nó. Garanta que como root você faz ssh aos nós (compute00 e compute01). Se isso funcionar, vamos executar o comando `scontrol` para atualizar o status dos nós:

```
[root@sms-host vagrant]# scontrol update nodename=compute00 state=RESUME
[root@sms-host vagrant]# scontrol update nodename=compute01 state=RESUME
```

```
[root@sms-host vagrant]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up    1-00:00:00      2  idle  compute[00-01]
```

# Primeiro job interativo

- Vamos submeter um job interativo (**srun**), para executar por 1 minuto, e vamos ver qual nó o SLURM vai alocar. Vamos aguardar o SLURM finalizar nosso job pelo walltime. Aproveite para abrir uma nova aba do terminal, para monitorar a fila (**squeue**)

```
[root@sms-host vagrant]# su - test
Last login: Sáb Nov  4 11:53:54 UTC 2023 on pts/0
[test@sms-host ~]$ srun -N1 -t00:01:00 --pty /bin/bash
[test@compute00 ~]$
```

```
[test@sms-host ~]$ squeue
      JOBID PARTITION      NAME      USER ST      TIME  NODES NODELIST(REASON)
          39    normal      bash     test   R      0:10      1 compute00
```

# Resumo de nosso job

- Vamos ver um resumo de como foi nosso job. Para isso, usamos o comando `sacct <job_id>`

```
[test@sms-host ~]$ sacct -j 39
  JobID  JobName  Partition  Account  AllocCPUS  State  ExitCode
-----  -----  -----  -----  -----  -----  -----
 39      bash     normal   (null)    0        TIMEOUT  0:0
```

podemos perceber que o status de nosso job é TIMEOUT, isto é, o SLURM finalizou nosso job em função do walltime.

# Mudando nosso cluster

- Resgatando o **sinfo**, vamos ver que temos apenas uma fila normal

```
[root@sms-host vagrant]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up  1-00:00:00      2  idle  compute[00-01]
              5-00      0-01
```

- Vamos editar o arquivo **/etc/slurm/slurm.conf** no master, para aumentar a memória de cada nó e também criar uma nova fila

```
# COMPUTE_NODES
# OpenHPC default configuration
TaskPlugin=task/affinity
PropagateResourceLimitsExcept=MEMLOCK
AccountingStorageType=accounting_storage/filetxt
Epilog=/etc/slurm/slurm.epilog.clean
NodeName=compute0[0-1] Sockets=1 CoresPerSocket=2 RealMemory=3000 ThreadsPerCore=1 State=UNKNOWN
PartitionName=normal Nodes=compute0[0-1] Default=YES MaxMemPerNode=3000 MaxTime=24:00:00 State=UP
PartitionName=express Nodes=compute0[0-1] Default=NO MaxMemPerNode=1000 MaxTime=00:05:00 State=UP MaxNodes=1 MaxCPUsPerNode=1
ReturnToService=1
HealthCheckProgram=/usr/sbin/nhc
HealthCheckInterval=300
```

# Mudando nosso cluster

- Reinicie o **slurmctld** e veja se a nova fila é exibida
- Execute o comando **scontrol show nodes** para ver detalhes dos nós. Note as características do nó, em especial RealMemory, e veja se as duas partções (normal, express) estão alocadas para eles

```
[root@sms-host vagrant]# systemctl restart slurmctld
[root@sms-host vagrant]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up   1-00:00:00      2  idle  compute[00-01]
express       up      5:00      2  idle  compute[00-01]
[root@sms-host vagrant]#
```

```
[root@sms-host vagrant]# scontrol show nodes
NodeName=compute00 Arch=x86_64 CoresPerSocket=2
CPUAlloc=0 CPUTot=2 CPULoad=0.01
AvailableFeatures=(null)
ActiveFeatures=(null)
Gres=(null)
NodeAddr=compute00 NodeHostName=compute00 Version=18.08
OS=Linux 3.10.0-1062.el7.x86_64 #1 SMP Wed Aug 7 18:08:02 UTC 2019
RealMemory=3000 AllocMem=0 FreeMem=1593 Sockets=1 Boards=1
State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
Partitions=normal,express
BootTime=2023-11-04T11:51:09 SlurmStart=2023-11-04T11:59:17
CfgTRES=cpu=2,nem=3000M,billing=2
AllocTRES=
CapWatts=n/a
CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

```
NodeName=compute01 Arch=x86_64 CoresPerSocket=2
CPUAlloc=0 CPUTot=2 CPULoad=0.01
AvailableFeatures=(null)
ActiveFeatures=(null)
Gres=(null)
NodeAddr=compute01 NodeHostName=compute01 Version=18.08
OS=Linux 3.10.0-1062.el7.x86_64 #1 SMP Wed Aug 7 18:08:02 UTC 2019
RealMemory=3000 AllocMem=0 FreeMem=1591 Sockets=1 Boards=1
State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
Partitions=normal,express
BootTime=2023-11-04T11:51:05 SlurmStart=2023-11-04T11:59:17
CfgTRES=cpu=2,nem=3000M,billing=2
AllocTRES=
CapWatts=n/a
CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

# Execução de programas

- Hello, World
- Vamos submete-lo interativamente:

```
[test@sms-host ~]$ srun -N1 -p express --pty ./hello_world
Hostname:compute00.hpcnet
RAM Total:3006 MB
RAM Livre:1588 MB
System uptime: 2257 segundos
[test@sms-host ~]$
```

```
#include<iostream>
#include<unistd.h>
#include <sys/sysinfo.h>
using namespace std;
int main(){
    char hostname[256];
    gethostname(hostname, sizeof(hostname));
    struct sysinfo info;
    sysinfo(&info);

    cout << "Hostname:" << hostname << endl;
    cout << "RAM Total:" << info.totalram * info.mem_unit / (1024 * 1024) << " MB" << endl;
    cout << "RAM Livre:" << info.freeram * info.mem_unit / (1024 * 1024) << " MB" << endl;
    cout << "System uptime: " << info.uptime << " segundos" << endl;

    return 0;
}
```

# Execução de programas

- Hello, World – agora em batch
- Vamos agora submeter em batch. Nesse caso, vamos criar um [hello\\_world.slurm](#)

```
[test@sms-host ~]$ squeue
      JOBID PARTITION      NAME     USER ST      TIME  NODES NODELIST(REASON)
        41   express hello_wo    test  PD      0:00      1 (MaxMemPerLimit)
[test@sms-host ~]$
```

Note que nosso job está em PD (pending) e a razão é MaxMemPerLimit. Acontece que nós definimos 1000MB para um job na express e estamos pedindo 1GB. Vamos cancelar e solicitar 500MB

```
#!/bin/bash
#SBATCH --job-name=hello_world
#SBATCH --nodes=1
#SBATCH --partition=express
#SBATCH --mem=1G

echo Output do job $SLURM_JOB_ID
./hello_world
```

# Execução de programas

- Para cancelar use `scancel <jobid>`
- Modifique o script para 500M

```
[test@sms-host ~]$ sbatch hello_world.slurm
Submitted batch job 42
[100%]
```

```
[test@sms-host ~]$ sacct -j 42
JobID      JobName      Partition      Account      AllocCPUS      State ExitCode
-----      -----      -----      -----      -----      -----      -----
42          hello_wor+    express        (null)        0      COMPLETED      0:0
```

```
[test@sms-host ~]$ scancel 41
[test@sms-host ~]$ squeue
JOBID PARTITION      NAME      USER ST      TIME NODES NODELIST(REASON)
[test@sms-host ~]$ sacct -j 41
JobID      JobName      Partition      Account      AllocCPUS      State ExitCode
-----      -----      -----      -----      -----      -----      -----
41          hello_wor+    express        (null)        0      CANCELLED      0:0
```

```
#!/bin/bash
#SBATCH --job-name=hello_world
#SBATCH --nodes=1
#SBATCH --partition=express
#SBATCH --mem=500M

echo Output do job $SLURM_JOB_ID
./hello_world
```

```
[test@sms-host ~]$ cat slurm-42.out
Output do job 42
Hostname:compute00.hpcnet
RAM Total:3006 MB
RAM Livre:1587 MB
System uptime: 2832 segundos
```

# Execução de programas

```
#!/bin/bash
#SBATCH --job-name=loop_serial
#SBATCH --nodes=1
#SBATCH --partition=express
#SBATCH --mem=500M

echo Output do job $SLURM_JOB_ID
./loop_serial
```

```
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>
using namespace std;

int main(){
    const int size = 50000000;
    vector<double> results(size);
    double sum = 0.0;

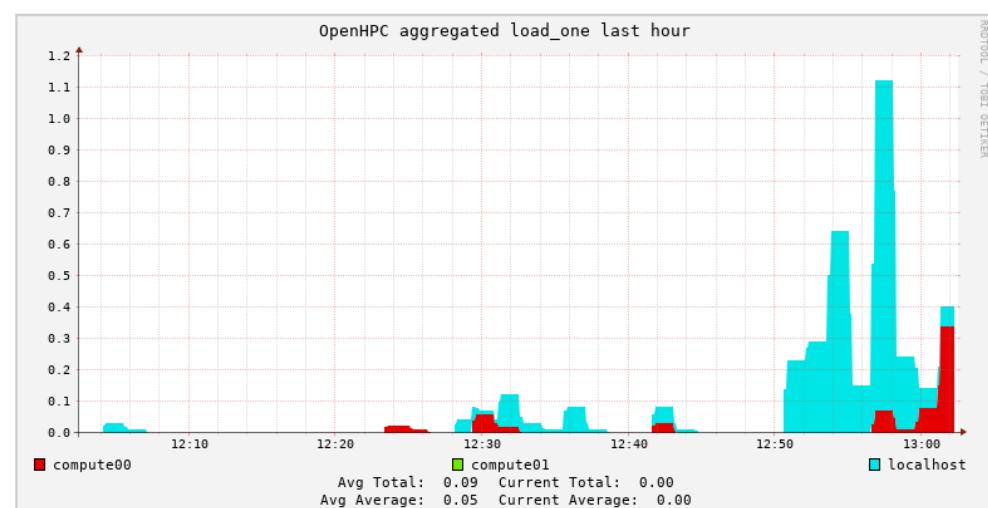
    for(int i = 0; i < size; ++i) {
        results[i] = sqrt(static_cast<double>(i));
        sum += results[i];
    }
    ofstream outfile("simulation_results.txt");
    for(auto &value: results){
        outfile << value << "\n";
    }
    outfile.close();
    cout << "Simulacao completa. Soma das raizes = " << sum << endl;
    return 0;
}
```

```
[test@sms-host ~]$ sbatch loop_serial.slurm
Submitted batch job 44
[test@sms-host ~]$ squeue
      JOBID PARTITION      NAME      USER ST      TIME  NODES NODELIST(REASON)
          44   express  loop_ser     test  R      0:05      1 compute00
[test@sms-host ~]$
```

# Execução de programas

- Não esqueça de monitorar via Ganglia

```
[test@sms-host ~]$ scontrol show job 44
JobId=44 JobName=loop_serial
UserId=test(1001) GroupId=test(1001) MCS_label=N/A
Priority=4294901755 Nice=0 Account=(null) QOS=(null)
JobState=COMPLETED Reason=None Dependency=(null)
Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
RunTime=00:01:05 TimeLimit=00:05:00 TimeMin=N/A
SubmitTime=2023-11-04T12:59:22 EligibleTime=2023-11-04T12:59:22
AccrueTime=2023-11-04T12:59:22
StartTime=2023-11-04T12:59:22 EndTime=2023-11-04T13:00:27 Deadline=N/A
PreemptTime=None SuspendTime=None SecsPreSuspend=0
LastSchedEval=2023-11-04T12:59:22
Partition=express AllocNode:Sid=sms-host:1804
ReqNodeList=(null) ExcNodeList=(null)
NodeList=compute00
BatchHost=compute00
NumNodes=1 NumCPUs=2 NumTasks=0 CPUs/Task=1 ReqB:S:C:T=0:0:0:0
TRES=cpu=2,mem=500M,node=1,billing=2
Socks/Node=* NtasksPerN:B:S:C=0:0:0:0 CoreSpec=*
MinCPUsNode=1 MinMemoryNode=500M MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
Command=/home/test/loop_serial.slurm
WorkDir=/home/test
StdErr=/home/test/slurm-44.out
StdIn=/dev/null
StdOut=/home/test/slurm-44.out
Power=
```



# Execução de programas

- Loop – agora com OpenMP

```
[test@sms-host ~]$ scontrol show job 47
JobId=47 JobName=loop_serial
UserId=test(1001) GroupId=test(1001) MCS_label=N/A
Priority=4294901752 Nice=0 Account=(null) QoS=(null)
JobState=COMPLETED Reason=None Dependency=(null)
Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
RunTime=00:00:59 TimeLimit=00:05:00 TimeMin=N/A
SubmitTime=2023-11-04T13:08:40 EligibleTime=2023-11-04T13:08:40
AccrueTime=2023-11-04T13:08:40
StartTime=2023-11-04T13:08:40 EndTime=2023-11-04T13:09:39 Deadline=N/A
PreemptTime=None SuspendTime=None SecsPreSuspend=0
LastSchedEval=2023-11-04T13:08:40
Partition=express AllocNode:Sid=sms-host:1804
ReqNodeList=(null) ExcNodeList=(null)
NodeList=compute00
BatchHost=compute00
NumNodes=1 NumCPUs=2 NumTasks=1 CPUs/Task=2 ReqB:S:C:T=0:0:2:*
TRES=cpu=2,mem=500M,node=1,billing=2
Socks/Node=* NtasksPerN:B:S:C=0:0:2: CoreSpec=*
MinCPUsNode=2 MinMemoryNode=500M MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
Command=/home/test/loop_omp.slurm
WorkDir=/home/test
StdErr=/home/test/slurm-47.out
StdIn=/dev/null
StdOut=/home/test/slurm-47.out
Power=
```

```
#!/bin/bash
#SBATCH --job-name=loop_OMP
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=2
#SBATCH --partition=express
#SBATCH --mem=500M

echo Output do job $SLURM_JOB_ID
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
./loop_omp
```

O que acontece se colocarmos -cpu-per-tasks > 2?

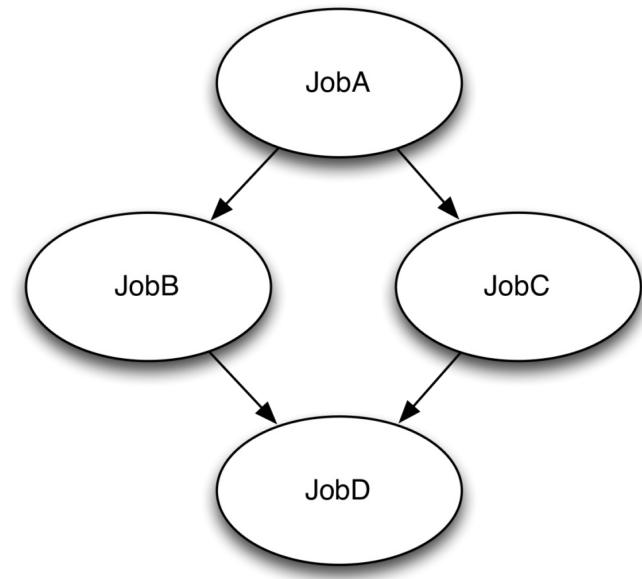
Lembre-se que no slurm o número de chips físicos a serem alocados é dado por:

nro\_chips = numero de nós x numero de tasks-per-node x numero de cpus-per-task

Em um job OMP, o número de nós sempre vai ser 1. Tasks-per-node deve ser 1 também e modificamos apenas o cpus-per-tasks. Mas como fazer para rodar 4 threads sem alocar 2 nós????

# Dependência entre jobs

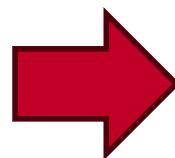
- É possível configurar dependências entre jobs e deixar com o que o SLURM determine o melhor momento de executar um job, com base no grafo de dependências.
- Vamos simular uma dependência de um novo job ao nosso loop serial



```
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>
using namespace std;

int main(){
    const int size = 50000000;
    vector<double> results(size);
    double sum = 0.0;

    for(int i = 0; i < size; ++i) {
        results[i] = sqrt(static_cast<double>(i));
        sum += results[i];
    }
    ofstream outfile("simulation_results.txt");
    for(auto &value: results){
        outfile << value << "\n";
    }
    outfile.close();
    cout << "Simulacao completa. Soma das raizes = " << sum << endl;
    return 0;
}
```



```
#include<iostream>
#include<fstream>
#include<vector>
#include<numeric>
#include<algorithm>
using namespace std;

int main(){
    vector<double> results;
    double value;
    ifstream infile("simulation_results.txt");
    while(infile >> value) {
        results.push_back(value);
    }
    infile.close();
    double sum = accumulate(results.begin(), results.end(), 0.0);
    double mean = sum / results.size();

    cout << "Análise completa. Média das raízes: " << mean << endl;
    return 0;
}
```

```
#!/bin/bash
#SBATCH --job-name=loop_serial
#SBATCH --nodes=1
#SBATCH --partition=express
#SBATCH --mem=500M

echo Output do job $SLURM_JOB_ID
./loop_serial

~
```

```
#!/bin/bash

#SBATCH --job-name=data_analysis
#SBATCH --partition=express
#SBATCH --output=data_analysis.out
./data_analysis
```

```
[test@sms-host ~]$ sbatch loop_serial.slurm
Submitted batch job 48
[test@sms-host ~]$ sbatch --dependency=afterok:48 data_analysis.slurm
Submitted batch job 49
[test@sms-host ~]$ squeue
             JOBID PARTITION      NAME     USER ST       TIME  NODES NODELIST(REASON)
                  49   express data_ana   test  PD      0:00      1 (Dependency)
                  48   express loop_ser   test  R      0:19      1 compute00
[test@sms-host ~]$
```