HPC Clusters (High Performance Computing)

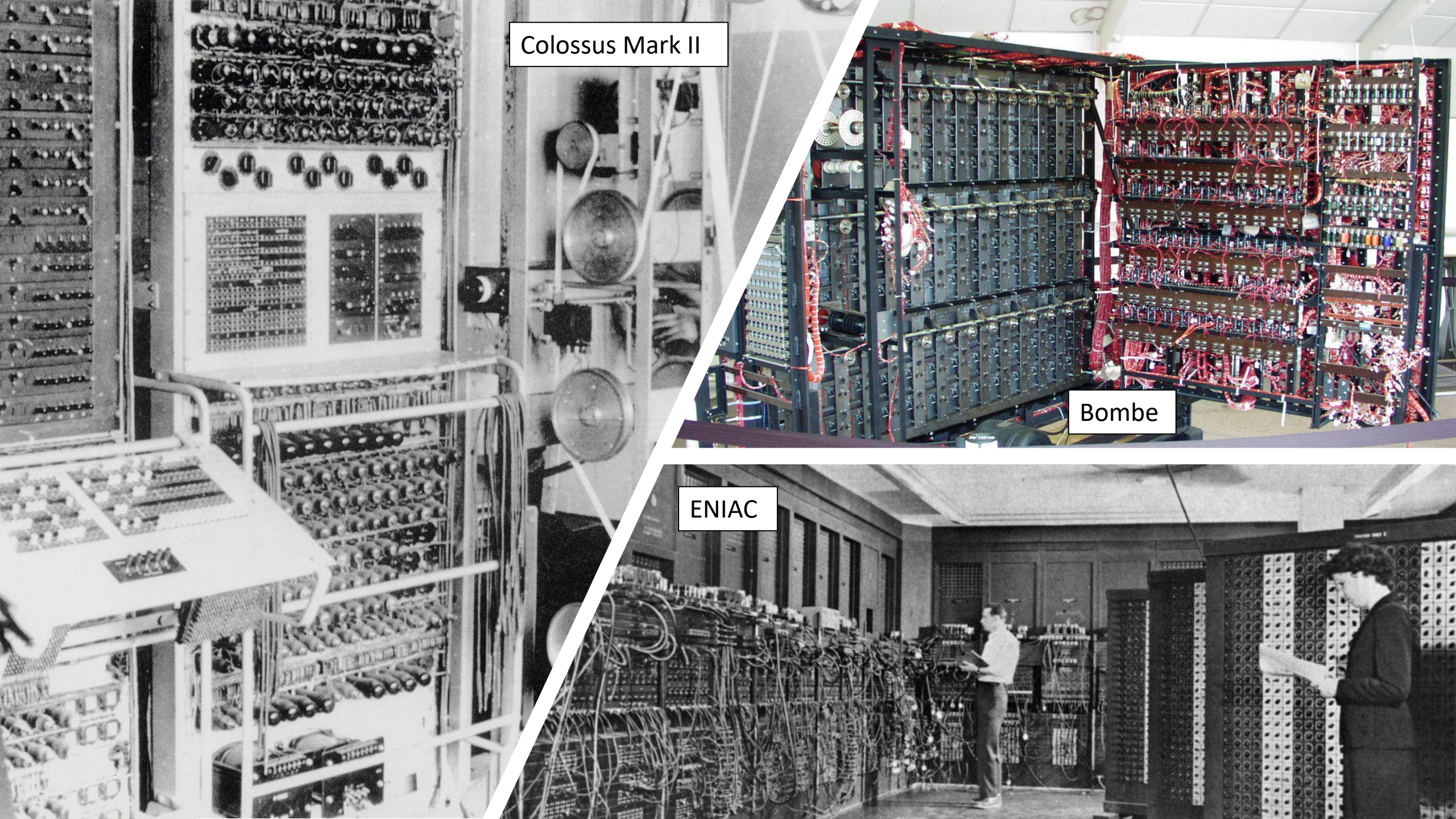
nicolas.greneche@univ-paris13.fr

HPC ?

- High Performance Computing
- Distributed system dedicated to experiments computing
- CPU (Central Processing Unit) / GPU (Graphical Processing Unit)

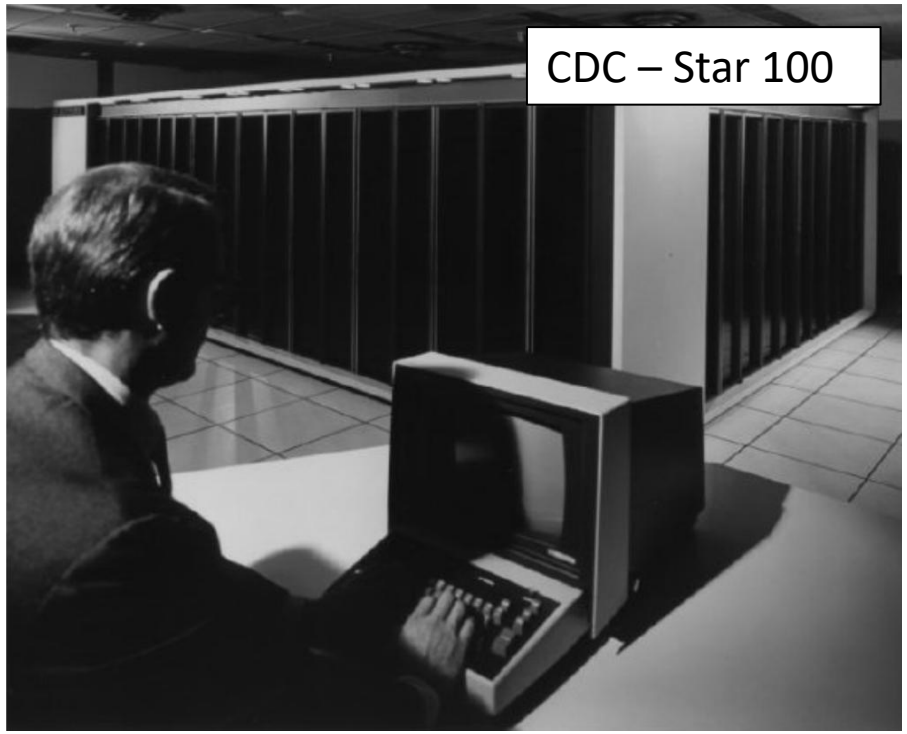
(Brief) story of HPC 40 to 70

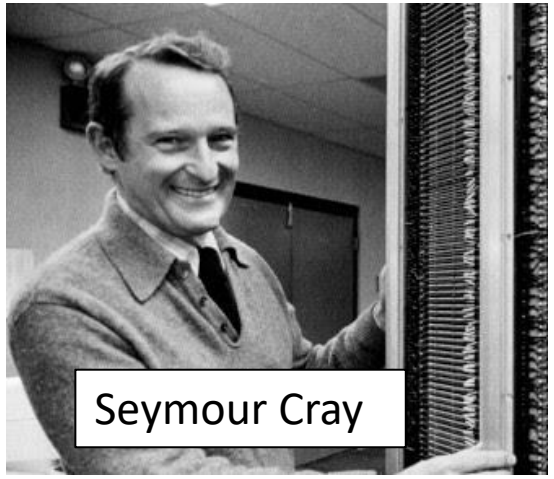
- War period
- Machines dedicated to cracking codes of military ciphers Nazis Enigma and Lorenz
- Lorenz: Colossus Mark I and II
- Enigma: The Bomb
- In 1945, the United States developed the reprogrammable ENIAC (Electronic Numerical Integrator and Computer)

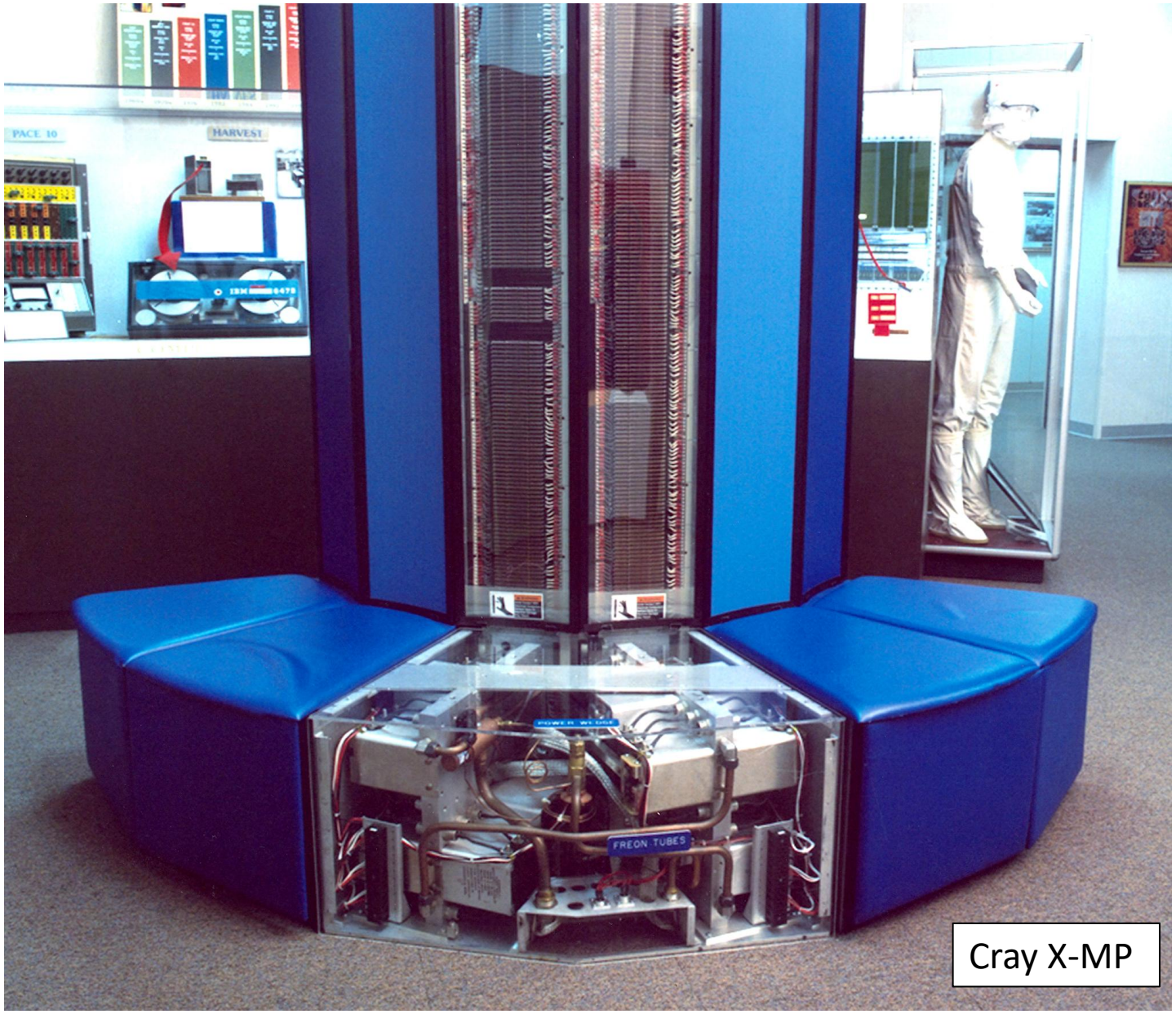

Colossus Mark II
Bombe
ENIAC

(Brief) story of HPC 70 to 85

- Cray Years – Seymour Cray, Father of HPC (SuperComputing)
- Appearance of Vector Processors (SIMD) – CDC star 100
- Appearance of specialized coprocessors – CDC star 100
- Emergence of the Concept of Shared Memory – Cray X-MP


CDC – Star 100


Seymour Cray



Cray X-MP

(Brief) story of HPC 80 to 2000

- Development of the concept of distributed memory (Intel Hypercube iPSC / Think Machine)
- Appearance of the BEOWULF clusters (1994)
- Appearance of single-image systems (Mosix)
- Appearance of the TOP 500

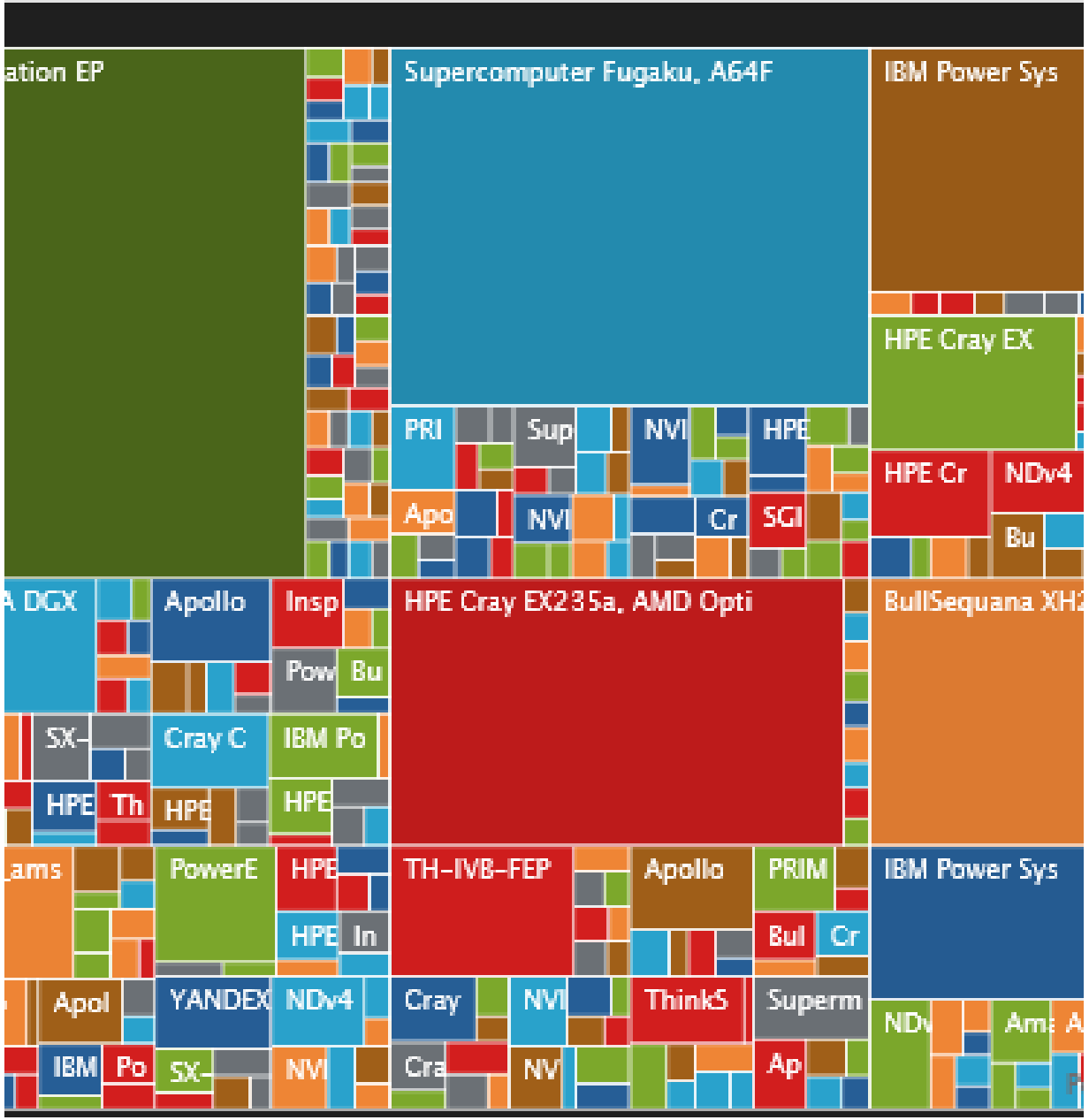
Think Machine

Intel iPSC

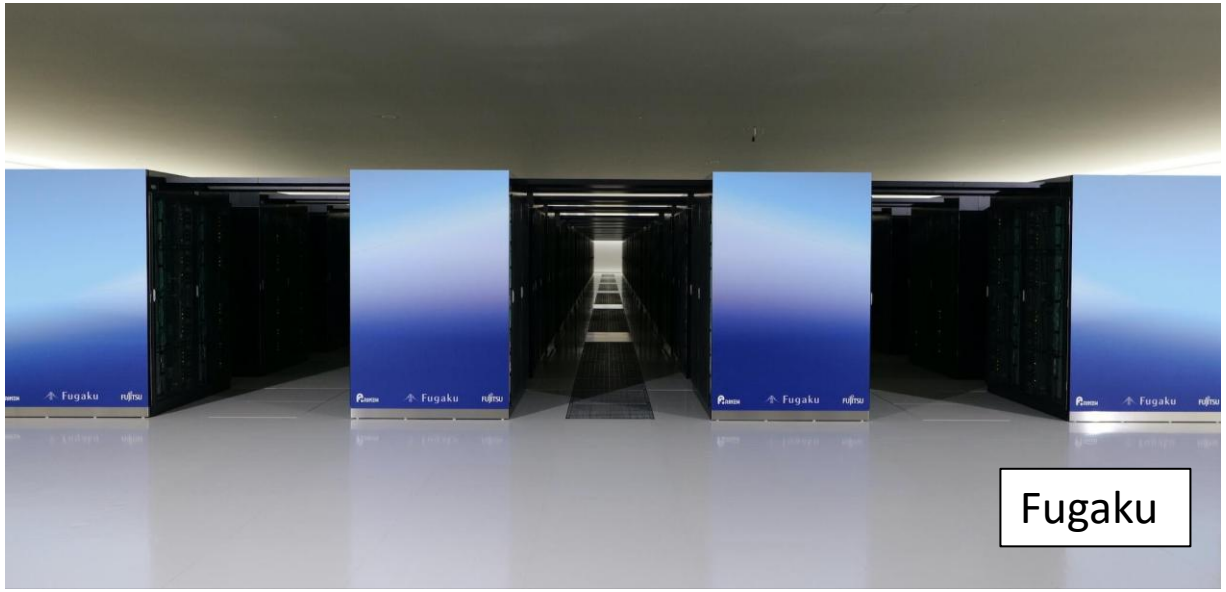
Cluster BEOWULF

(Brief) story of HPC since 2000

- Standardization of equipment
- Reign of Linux
- 2020: Fugaku embeds a custom ARM for the multi-kernel (537 PFLOPs)
- 2021: Frontier, first exaflops machine (1685 PFLOP)



Frontier



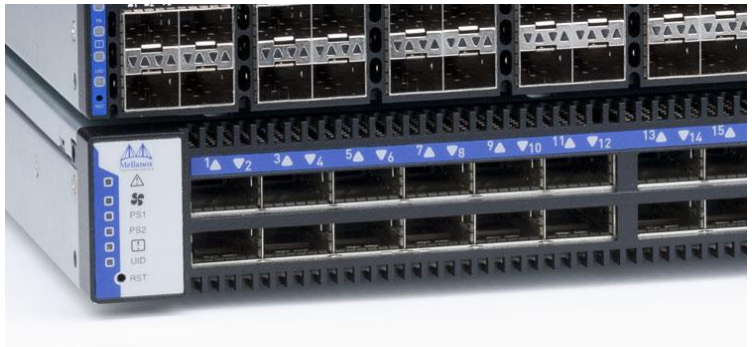
Fugaku

1. Hardware

#CPU #GPU #Network #Storage

Computing nodes

- CPU Intel, AMD or ARM
- Local storage on non-RAID SSD (Solid State Disk)
- Modularity / Density



Network topologies

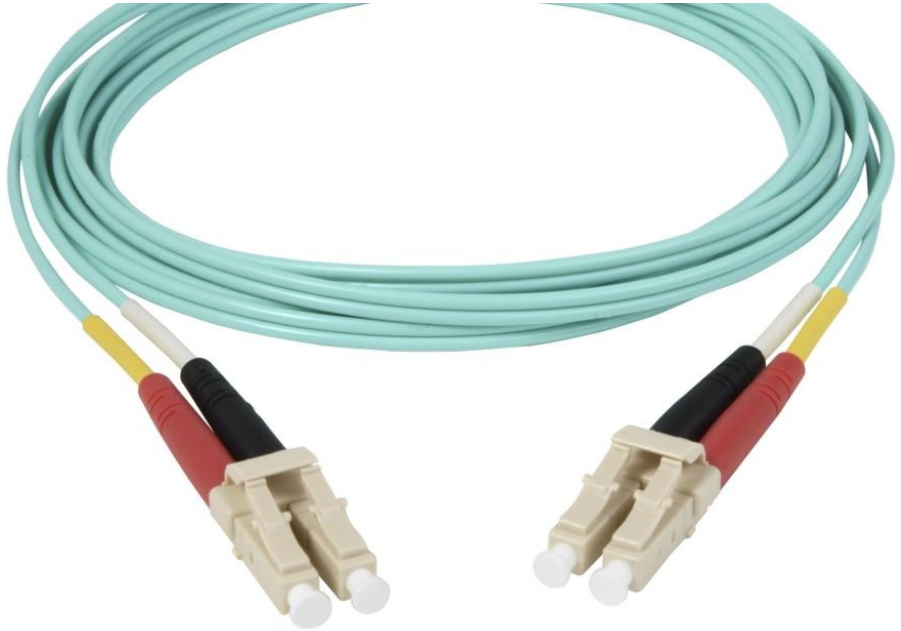
- Standard Network media (RJ45)
  - Standard
  - Cheap value …
  - … but high latency
- Low latency network (Infiniband)
  - Expensive …
  - … but low latency
  - Process to Process inter-nodes communication

GPU Node

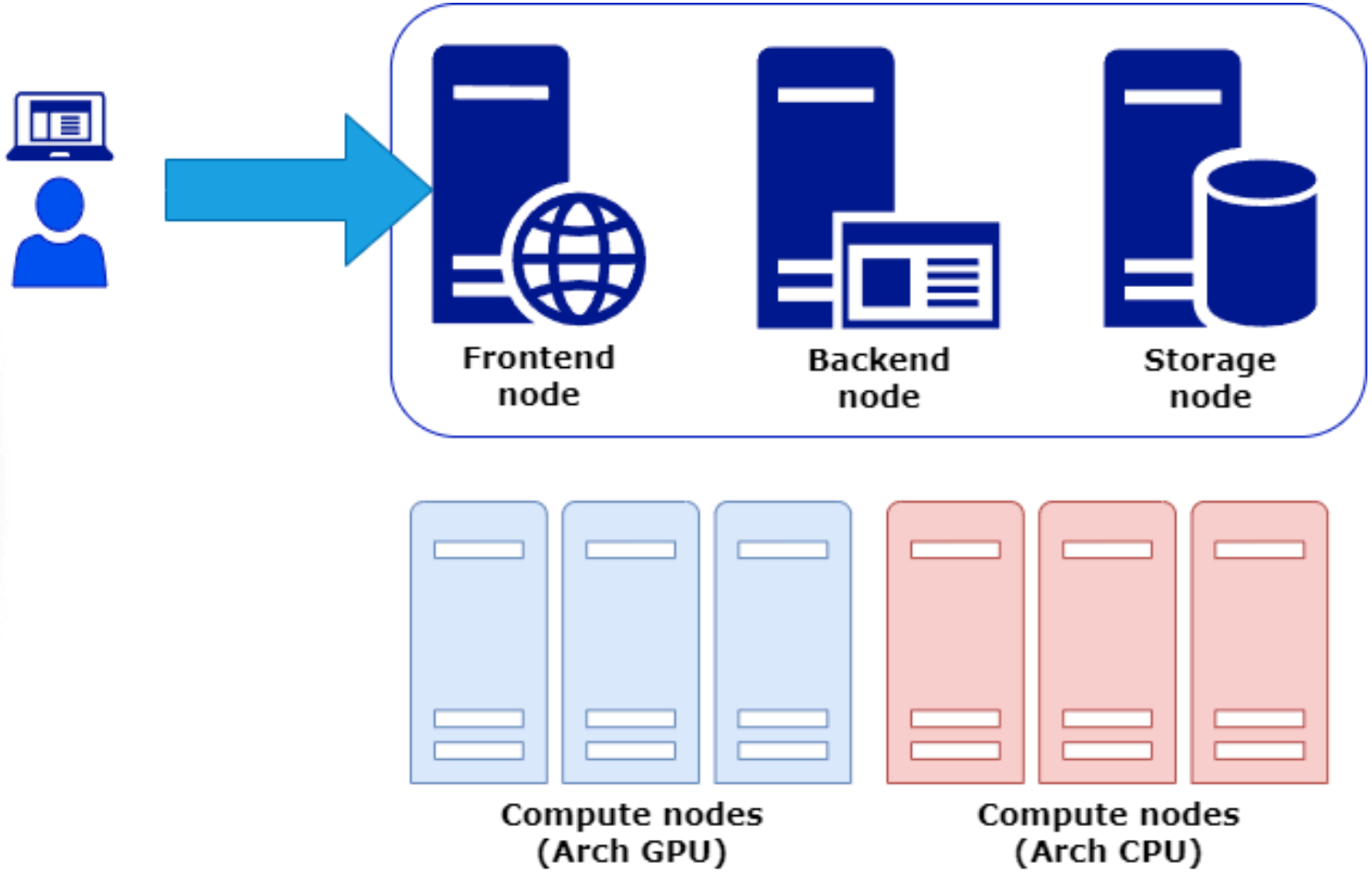
1. 10 x NVIDIA A100 – 80 GB
2. 2 x AMD EPYC 7513 32-Core Processor (128 hyperthreads)
3. 512 GB RAM

Storage (SAN)

- Huawei Dorado 2600 v5
- 2 PB (raw)
- NFS / CIFS / Blocks shares



Typical Topology



2. Usage

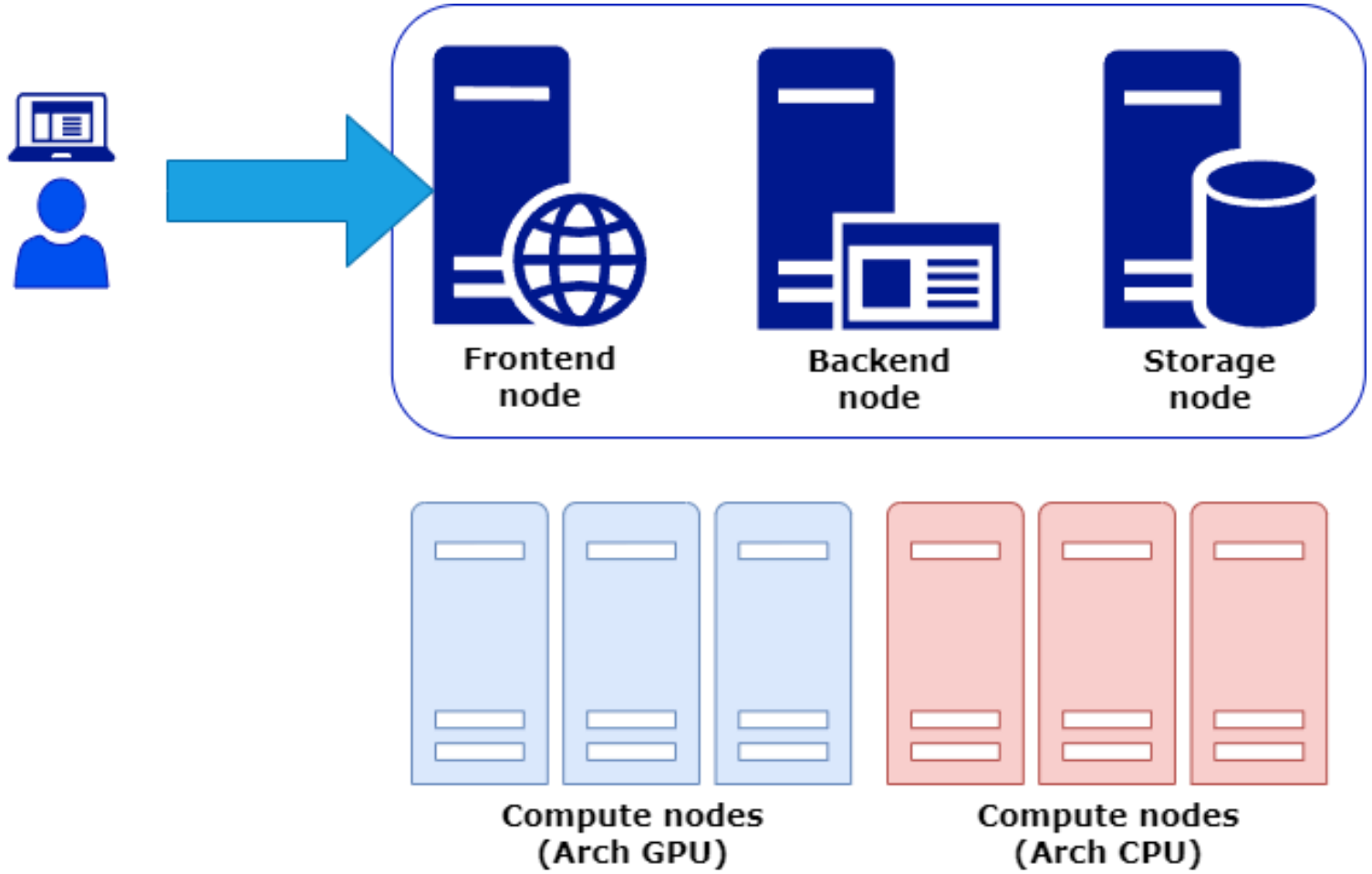
#JobScheduler #CPU #GPU #Batch #Interactive

Frontend

- Remote SSH access (Secure SHell)
  - Public IP
  - Non standard port (2822)
- Compilation
- File storage (on the distributed file system)
- Interacting with Job Scheduler

Submit a job

- Describe your job
  - Resource’s constraints
  - How to run your program
  - Optionnal features
- Target an arch (queue / partition)
  - Set of homogeneous hardware
  - CPU / GPU
  - No need to target a specific compute node



Application avec SLURM

- SLURM (Simple Linux Utility for Resource Management) is our Job Scheduler
- Basic commands :
  - sinfo : state of the Cluster
  - srun : run an interactive job
  - sbatch : run a batch job
  - squeue : show waiting queue
  - scancel : cancel a job
- Each queue has its own parameters (walltime, priority etc.)

slurm
workload manager

Etat du cluster

```
nicolas.greneche@magi1:~$ sinfo
PARTITION          AVAIL  TIMELIMIT  NODES  STATE NODELIST
SMP                   up   infinite      1  drain magi10
SMP                   up   infinite      2   idle magi[33,94]
COMPUTE*              up   infinite      4 drain* magi[72,74,76,86]
COMPUTE*              up   infinite      8  down* magi[46-53]
COMPUTE*              up   infinite     18  alloc magi[54-66,73,75,83-85]
COMPUTE*              up   infinite     11   idle magi[67-71,77-82]
COMPUTE-SHORT         up 3-00:00:00     11 drain* magi[72,74,76,86-93]
COMPUTE-SHORT         up 3-00:00:00      8  down* magi[46-53]
COMPUTE-SHORT         up 3-00:00:00     21  alloc magi[54-66,73,75,83-
85,96-98]
```

Launching your first SLURM command

```
nicolas.greneche@magi1:~$ srun -N 2 -p COMPUTE hostname
magi67
magi68
```

Write your first submission script

```
nicolas.greneche@magi1:~$ cat hostname.slurm
#!/bin/bash
#SBATCH --job-name=hostname
#SBATCH --output=slurm.out
#SBATCH --error=slurm.err
#SBATCH --mail-type=end
#SBATCH --mail-user=nicolas.greneche@univ-paris13.fr
#SBATCH --nodes=2

srun hostname
```

Submit

```
nicolas.greneche@magi1:~$ sbatch hostname.slurm
Submitted batch job 44066
```

Check the output

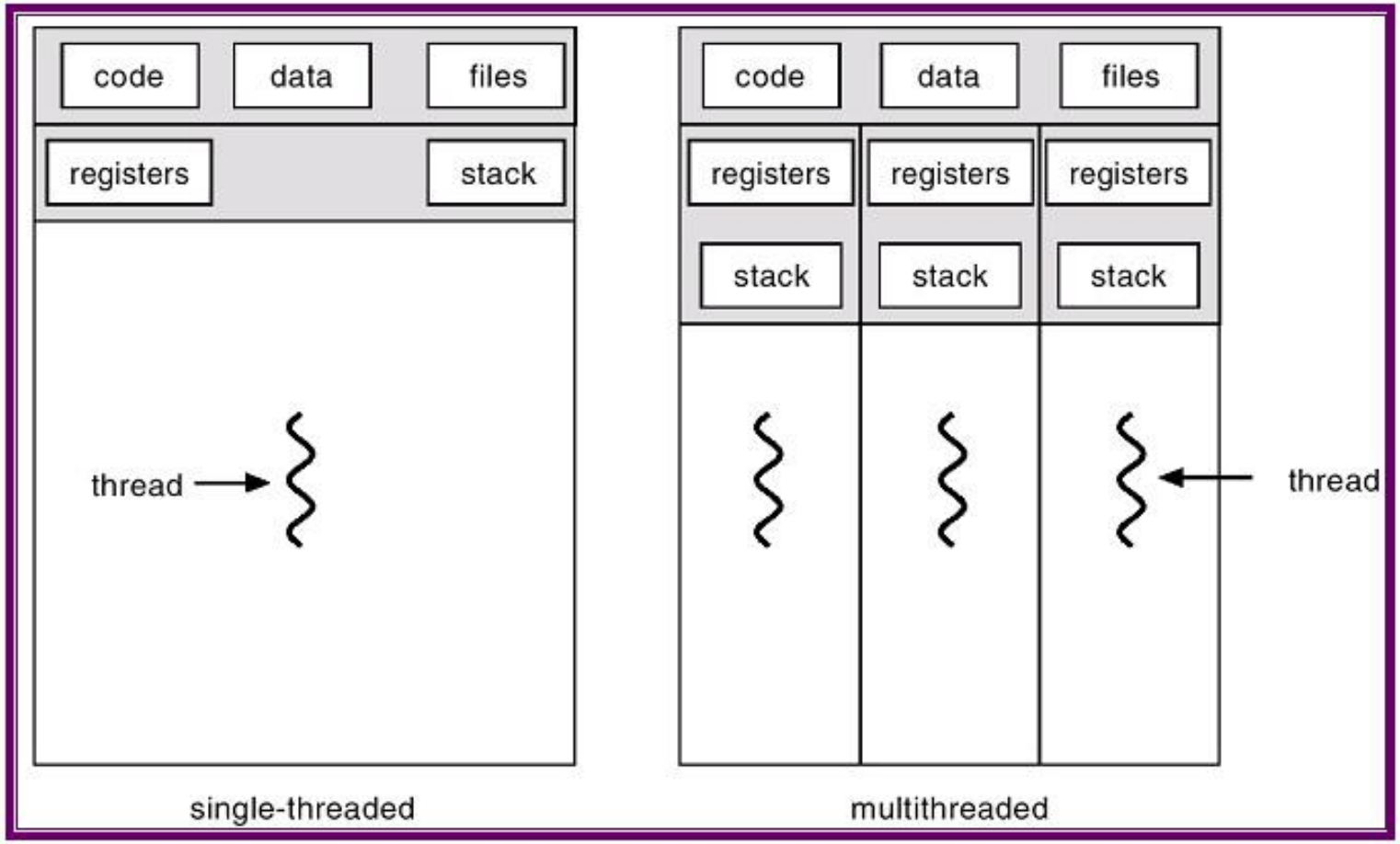
```
nicolas.greneche@magi1:~$ cat slurm.out
magi66
magi75
```

CPU parallel codes

#MPI #OpenMP

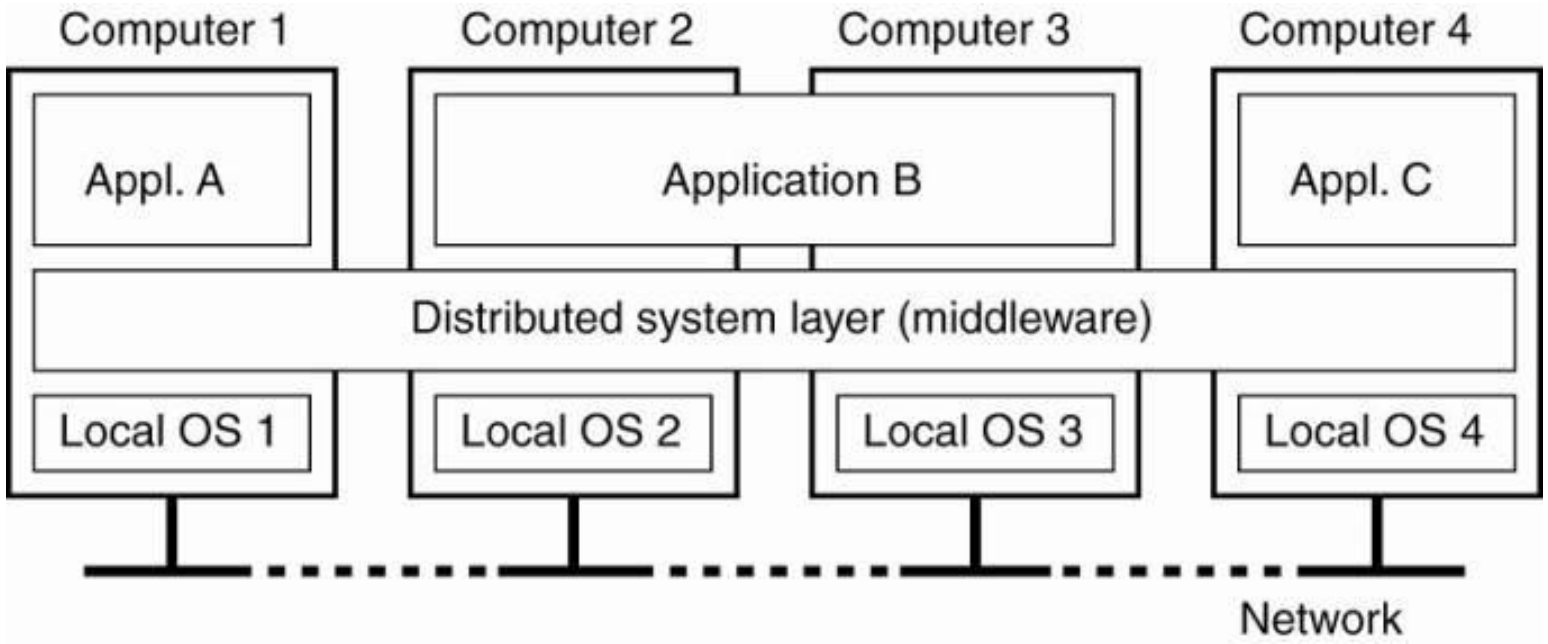
Shared Memory

- Single Node approach
- Multi-thread
- Implementation
  - Pthread
  - OpenMP



Distributed Memory

- Several nodes involved
- Communication between remote processes
- MPI Protocol (Message Passing Interface)
- Implementations :
  - OpenMPI
  - MPICH
  - Intel MPI



Using OpenMP (8 cores)


```
nicolas.greneche@magi1:~$ gcc -fopenmp -o hello_openmp hello_openmp.c
nicolas.greneche@magi1:~$ export OMP_NUM_THREADS=4
nicolas.greneche@magi1:~$ ./hello_openmp
Thread 0 says: Hello World on core 7
Thread 0 reports: the number of threads are 4
Thread 3 says: Hello World on core 3
Thread 1 says: Hello World on core 5
Thread 2 says: Hello World on core 4
```

OpenMP code snippet


```
#include <omp.h>
#include <stdio.h>
#define _GNU_SOURCE
#include <utmpx.h>
int main (int argc, char *argv[])
{
   int nthreads, thread_id;
   #pragma omp parallel private(nthreads, thread_id) {
      thread_id = omp_get_thread_num();
      printf("Thread %d says: Hello World on core %i\n", thread_id,sched_getcpu());
      if (thread_id == 0){
         nthreads = omp_get_num_threads();
         printf("Thread %d reports: the number of threads are %d\n",
                        thread_id, nthreads); }}
  return 0;}
```

Submit an OpenMP Job


```
nicolas.greneche@magi1:~$ cat run.slurm
#!/bin/bash
#SBATCH --job-name=OMP_hello
#SBATCH --output=slurm.out
#SBATCH --error=slurm.err
#SBATCH --partition=COMPUTE-SHORT
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=4

export OMP_NUM_THREADS=4
srun -l hello_openmp
```

Submit an OpenMP Job (40 cores)

```
nicolas.greneche@magi1:~$ sbatch run.slurm
Submitted batch job 44075
nicolas.greneche@magi1:~$ cat slurm.out
Thread 0 says: Hello World on core 6
Thread 0 reports: the number of threads are 4
Thread 1 says: Hello World on core 32
Thread 3 says: Hello World on core 1
Thread 2 says: Hello World on core 14
```

Using OpenMPI (8 cores)

```
nicolas.greneche@magi1:~$ module load XXXX
nicolas.greneche@magi1:~$ mpicc -o hellompi hellompi.c
nicolas.greneche@magi1:~$ mpirun -np 4 hellompi
Hello world from process 1 of 4 at magi1
Hello world from process 3 of 4 at magi1
Hello world from process 0 of 4 at magi1
Hello world from process 2 of 4 at magi1
```

Using OpenMPI (40 cœurs)

```
nicolas.greneche@magi1:~$ srun -N 2 --tasks-per-node 2 hellompi
Hello world from process 3 of 4 at magi75
Hello world from process 1 of 4 at magi66
Hello world from process 0 of 4 at magi66
Hello world from process 2 of 4 at magi75
```

OpenMPI code snippet

```
nicolas.greneche@magi1:~$ cat hellompi.c
#include <stdio.h>
#include <mpi.h>
#include <stdio.h>
int main (argc, argv)
     int argc;
     char *argv[]; {
  int rank, size;
  char hostname[40];
  MPI_Init (&argc, &argv);      /* starts MPI */
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);        /* get current process id */
  MPI_Comm_size (MPI_COMM_WORLD, &size);        /* get number of processes */
  gethostname(hostname,40);
  printf( "Hello world from process %d of %d at %s\n", rank, size, hostname );
  MPI_Finalize();
  return 0;}
```

Submit an OpenMPI Job

```
nicolas.greneche@magi1:~$ cat run.slurm
#!/bin/bash
#SBATCH --job-name=hello
#SBATCH --output=slurm.out
#SBATCH --error=slurm.err
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2

module load xxx
srun ./hellompi
```

Submit an OpenMPI Job (2 nodes)

```
nicolas.greneche@magi1:~$ sbatch run.slurm
Submitted batch job 44082
nicolas.greneche@magi1:~$ cat slurm.out
Hello world from process 1 of 4 at magi66
Hello world from process 3 of 4 at magi75
Hello world from process 0 of 4 at magi66
Hello world from process 2 of 4 at magi75
```

GPU Codes

#cuda

CUDA code snippet


```
nicolas.greneche@magi3:~/cuda$ cat test.cu
#include <stdio.h>
int main() {
  int nDevices;
  cudaGetDeviceCount(&nDevices);
  for (int i = 0; i < nDevices; i++) {
        cudaDeviceProp prop;
        cudaGetDeviceProperties(&prop, i);
        printf("Device Number: %d\n", i);
        printf("  Device name: %s\n", prop.name);
        printf("  Memory Clock Rate (KHz): %d\n",
        prop.memoryClockRate);
        printf("  Memory Bus Width (bits): %d\n",
        prop.memoryBusWidth);
        printf("  Peak Memory Bandwidth (GB/s): %f\n\n",
        2.0*prop.memoryClockRate*(prop.memoryBusWidth/8)/1.0e6);}}
```

Compile CUDA code


```
nicolas.greneche@magi3:~/cuda$ module load xxx
nicolas.greneche@magi3:~/cuda$ nvcc -o test test.cu
```

Run interactive GPU job

```
nicolas.greneche@magi3:~/cuda$ module load xxx
nicolas.greneche@magi3:~/cuda$ srun --tasks=1 --cpus-per-task=4 --partition=GPU-A100 --gres=gpu:2 ./test
srun: job 33437 queued and waiting for resources
srun: job 33437 has been allocated resources
Device Number: 0
  Device name: NVIDIA A100 80GB PCIe
  Memory Clock Rate (KHz): 1512000
  Memory Bus Width (bits): 5120
  Peak Memory Bandwidth (GB/s): 1935.360000

Device Number: 1
  Device name: NVIDIA A100 80GB PCIe
  Memory Clock Rate (KHz): 1512000
  Memory Bus Width (bits): 5120
  Peak Memory Bandwidth (GB/s): 1935.360000
```

GPU submission script


```
nicolas.greneche@magi3:~/cuda$ cat gpu.slurm
#!/bin/bash
#SBATCH --job-name=gpu
#SBATCH --output=gpu.out
#SBATCH --error=gpu.err
#SBATCH --gres=gpu:2
#SBATCH --partition=GPU-A100

module load xxx
srun ./test
```

Submit CUDA code

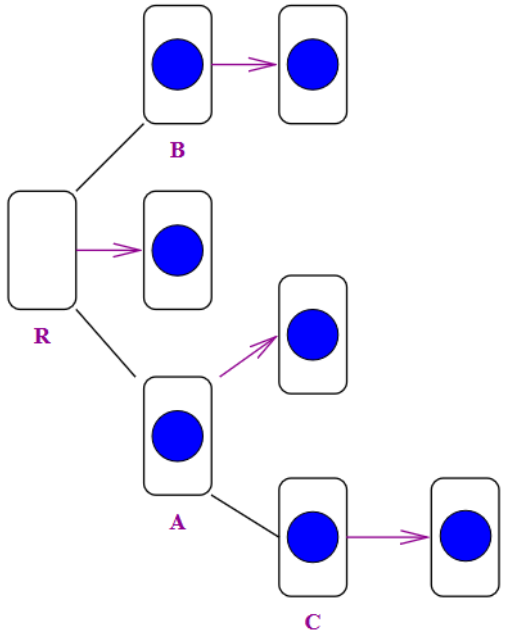

```
nicolas.greneche@magi3:~/cuda$ sbatch gpu.slurm
Submitted batch job 33440

nicolas.greneche@magi3:~/cuda$ cat gpu.out
Device Number: 0
  Device name: NVIDIA A100 80GB PCIe
  Memory Clock Rate (KHz): 1512000
  Memory Bus Width (bits): 5120
  Peak Memory Bandwidth (GB/s): 1935.360000

Device Number: 1
  Device name: NVIDIA A100 80GB PCIe
  Memory Clock Rate (KHz): 1512000
  Memory Bus Width (bits): 5120
  Peak Memory Bandwidth (GB/s): 1935.360000
```

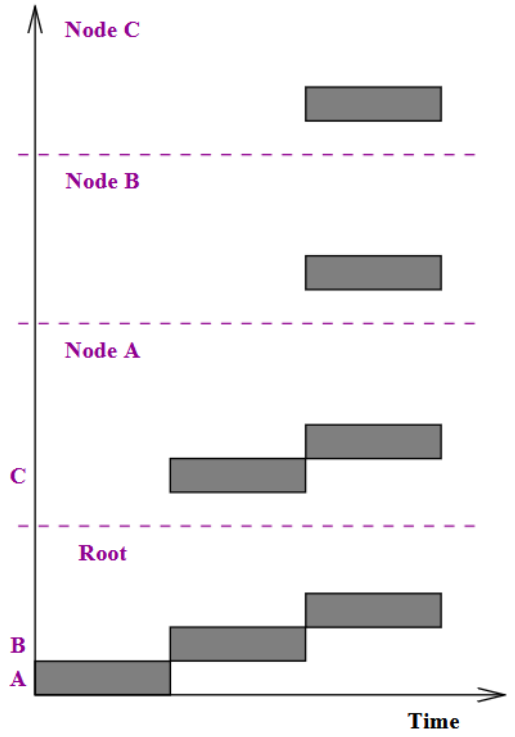
Under the hood

#provisioning #baremetal

Pushing OS images on nodes with Kadeploy

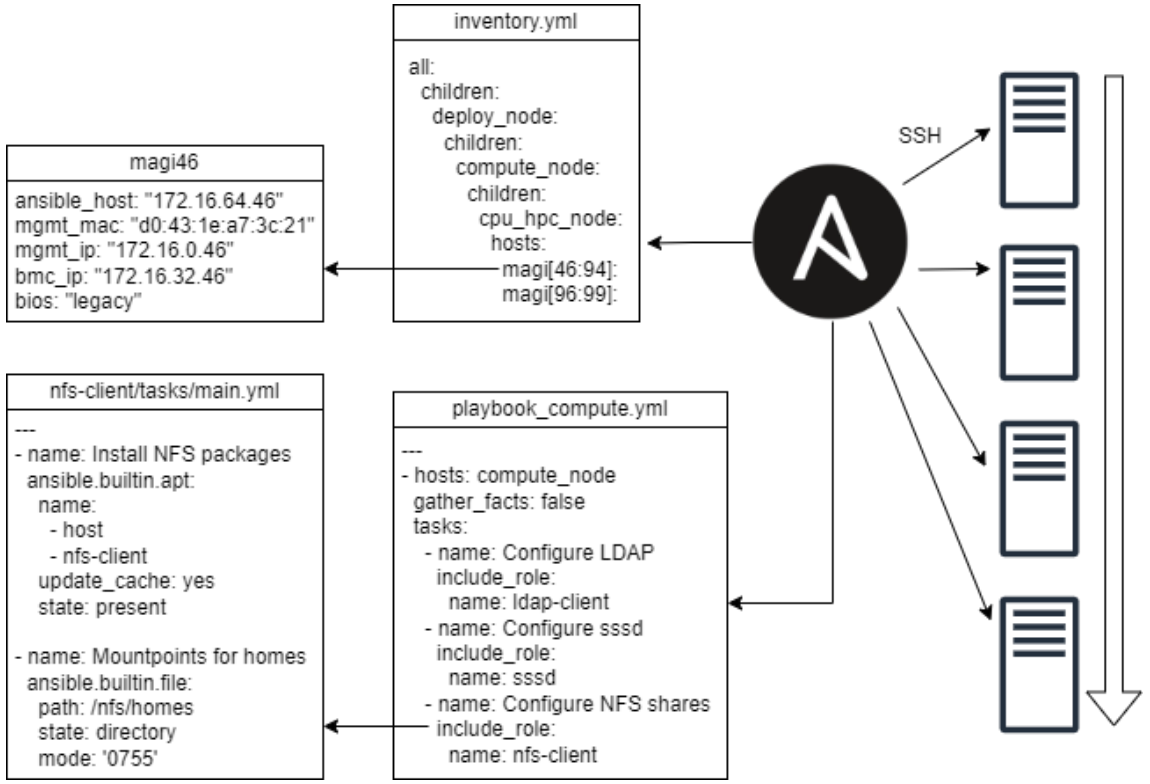
- Relies on regular services (DHCP, BOOTP and PXE)
- Images distribution with Taktuk





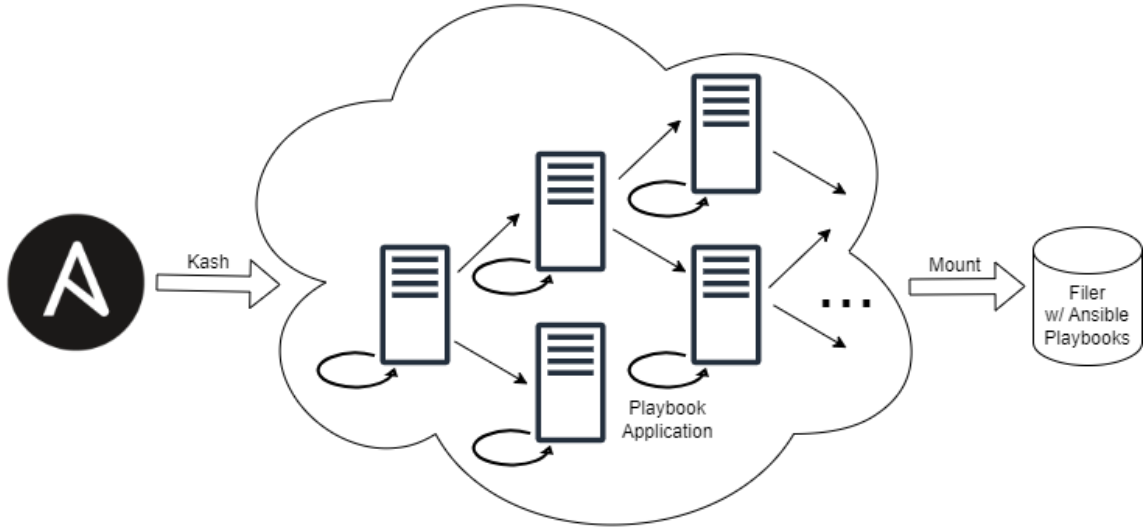
Post-configuration with Ansible

- Target a state
- Relies on SSH



Executing playbook

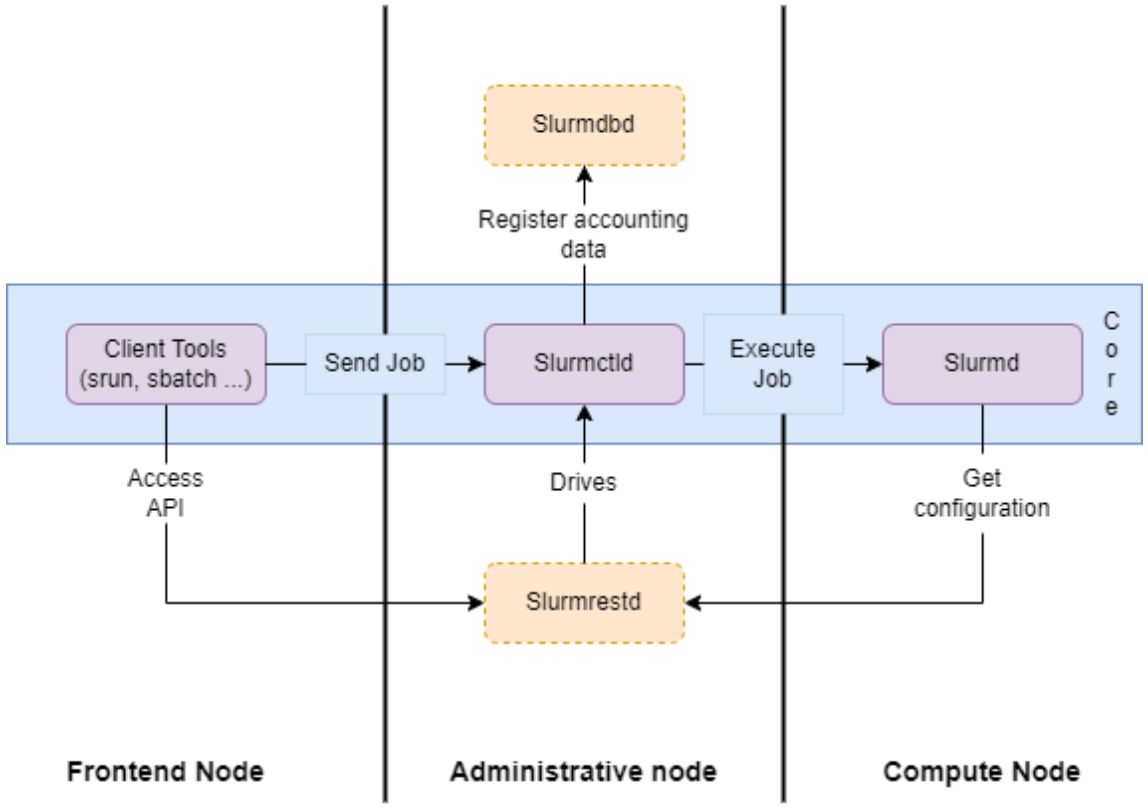
- Relies on Taktuk
- Distributed execution of playbook



SLURM internals snippets

#slurmd #slurmctld #cgroups
#namespaces

SLURM Architecture



Communications

1a srun demande l'allocation

1b slurmctld renvoie les informations

2a srun envoie une demande de création d'étape

2b slurmctld renvoie les credentials

3 srun ouvre un socket

4 et s'adresse en direct à slurmd (avec les credentials)

5 slurmd transfert la requête (distribué)

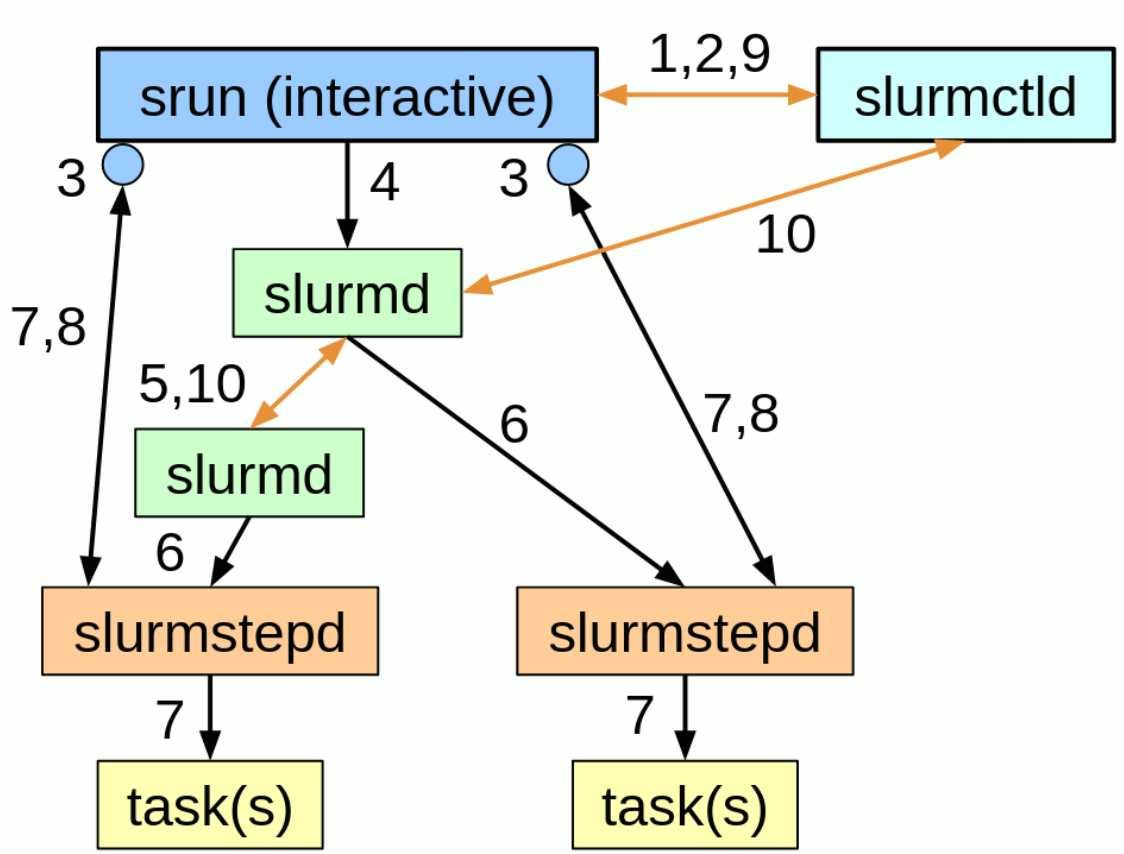
6 slurmd forks / execs slurmstepd

7 slurmstepd met en place les descripteurs

8 slurmstepd notifie la fin du travail

9 srun notifie la fin du travail

10 slurmctld fait le ramasse-miette et relâche l'allocation



Job execution

- Slurmstepd detaches from slurmd (via double-fork)
- First, Slurmstepd sets up the job environment (e.g., cgroups, namespaces, resource limits)
- Then, it clone() / setuid() / execve() user’s task

slurmd (PID X)

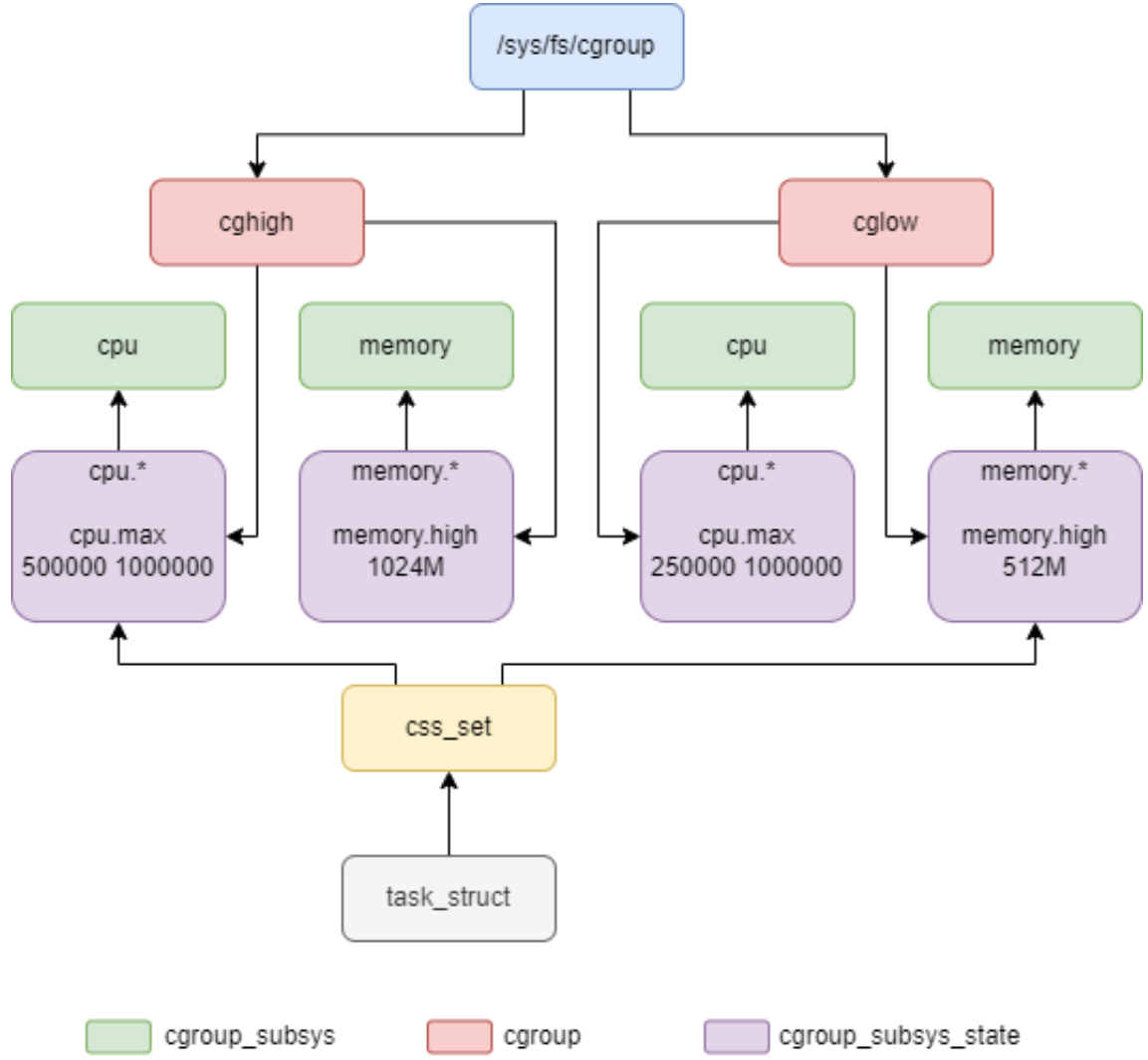
└── forks slurmstepd (PID Y)

└── slurmstepd detaches (PID Y → now parented by init/systemd)

└── user's job (PID Z, under cgroup managed by slurmstepd)

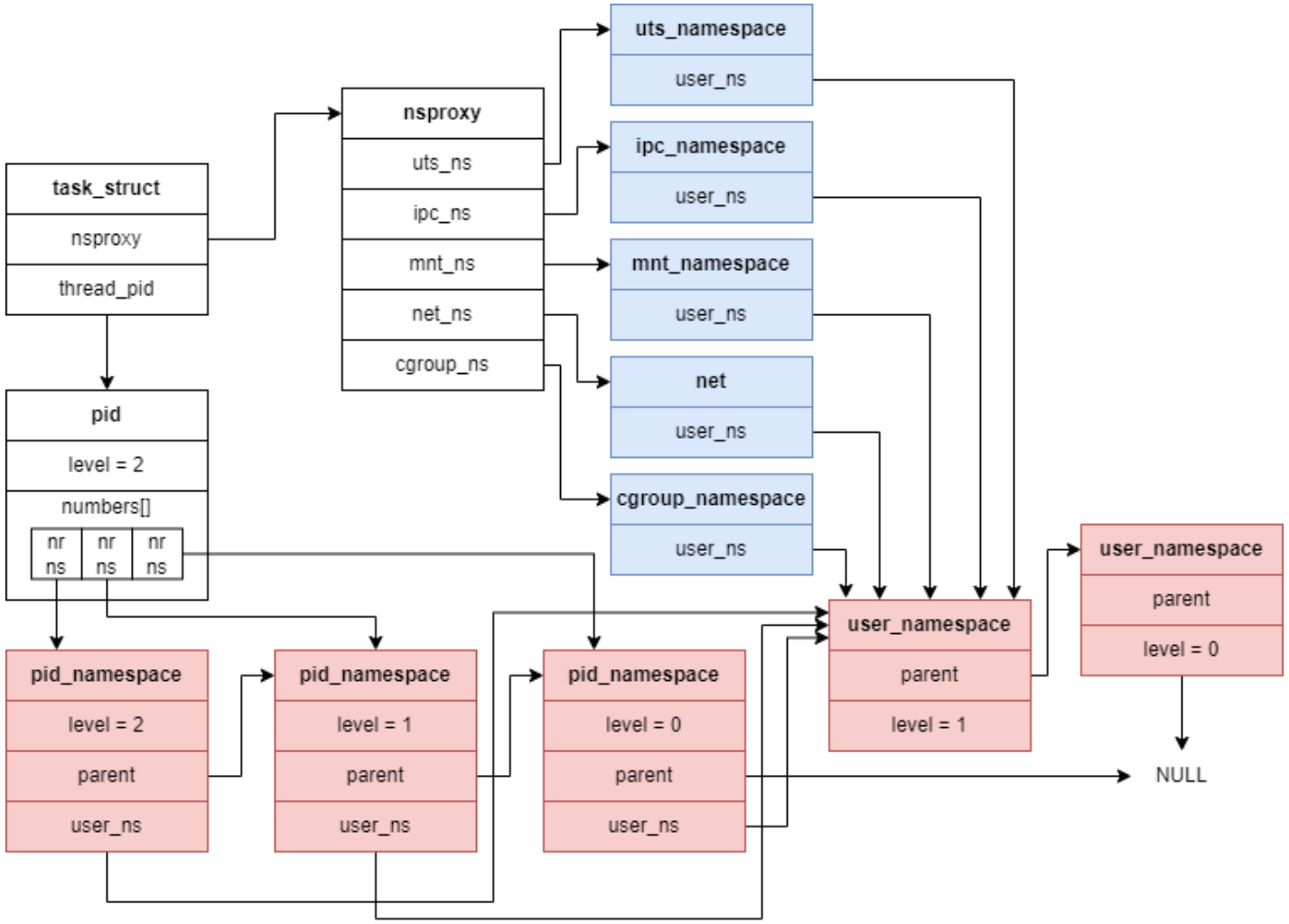
Ressources tracking (cgroups)

- Setup by Slurmstepd
- Increase granularity



Job isolation

- Use regular daemonless containers engine (charliecloud, Apptainer / Singularity)
- Since SLURM 23.11, seamless integration with scrun (Docker / Podman)



Conclusion

...And The Circus Leaves Town

HPC vs Cloud

- Why SLURM (and HPC Schedulers) Will Likely Survive
  - Specialized for HPC Workloads
  - Established Ecosystem
  - Cost Efficiency for On-Prem HPC
  - Hybrid HPC/Cloud Models
- Cloud Orchestrators Are Catching Up
  - Kubernetes + HPC Plugins: KubeFlow, Volcano Scheduler, and MPI operators now support MPI/allreduce workloads
  - Serverless HPC: AWS Batch, Google Cloud Batch (for embarrassingly parallel jobs)
  - Cloud-native workflows (e.g., AI/ML pipelines) often prefer Kubernetes for its flexibility, containerization, and DevOps integration
  - SLURM requires dedicated sysadmins; cloud orchestrators benefit from larger DevOps communities

Questions ?

Thank you !