UNIVERSITÀ DEGLI STUDI DI PADOVA

J. Pazzini
PADOVA UNIVERSITY

9 - INTRO TO DISTRIBUTED PROCESSING

Management and Analysis of Physics Datasets - Module B

Physics of Data

A.A. 2023/2024

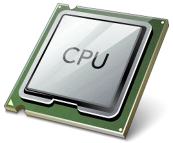
- Processing can scale with the number of processing units **n**
  (provided it is parallelizable, at least in part)

- Speedup studies usually show that processing time scale ~well with **n**
  (depending on the task/figure of merit/use case)

- Lots of processes are inherently highly scalable
  (heavily data-parallel tasks such as in the case of SPMD)

We now want to run some processing on very large (and possibly distributed) datasets

→ **Let's increase n !**



8 cores CPU

PARALLEL PROGRAMMING

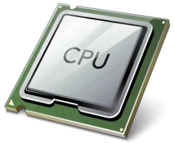
- Processing can scale with the number of processing units **n**
  (provided it is parallelizable, at least in part)

- Speedup studies usually show that processing time scale ~well with **n**
  (depending on the task/figure of merit/use case)

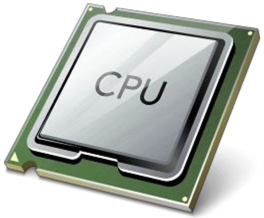
- Lots of processes are inherently highly scalable
  (heavily data-parallel tasks such as in the case of SPMD)

We now want to run some processing on very large (and possibly distributed) datasets

→ **Let's increase n !**



8 cores CPU



64 cores CPU

PARALLEL PROGRAMMING

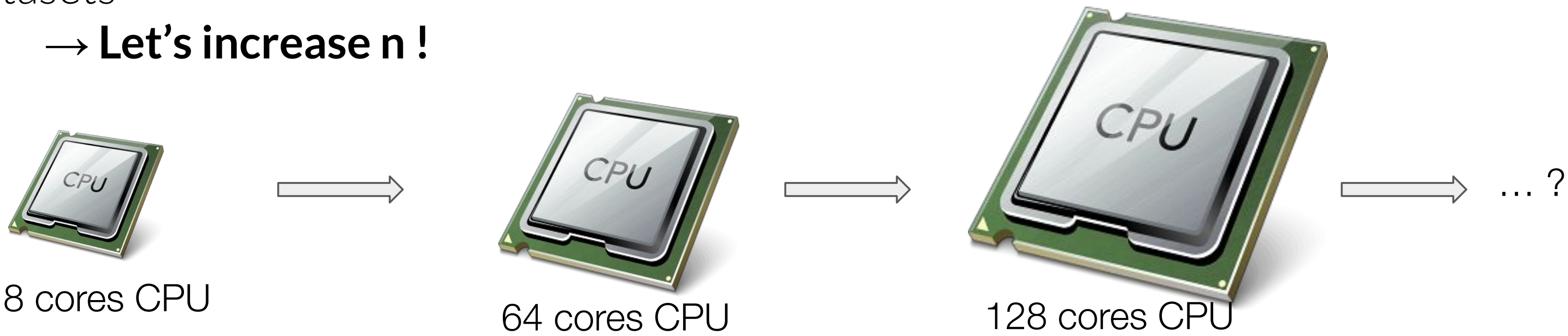
- Processing can scale with the number of processing units **n**
  (provided it is parallelizable, at least in part)

- Speedup studies usually show that processing time scale ~well with **n**
  (depending on the task/figure of merit/use case)

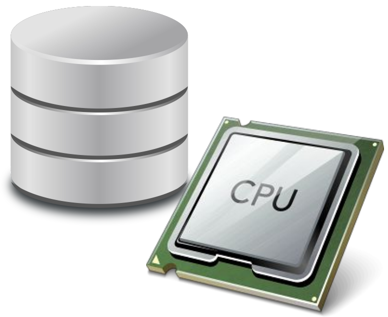
- Lots of processes are inherently highly scalable
  (heavily data-parallel tasks such as in the case of SPMD)

We now want to run some processing on very large (and possibly distributed) datasets

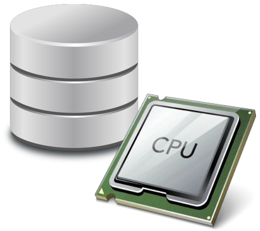
→ **Let’s increase n !**



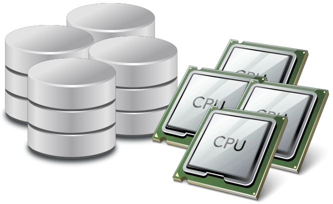
SCALABILITY



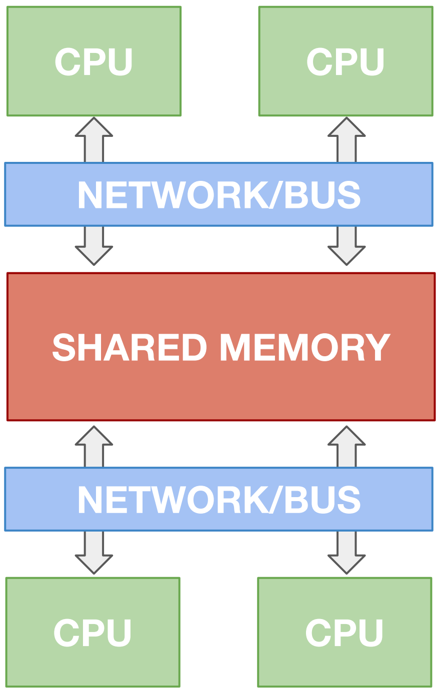
Vertical (Up) → increase the capacity/performance of the available resources (buy better HW)



Horizontal (Out) → add more resources (buy more *commodity* HW)



A number of alternative architectures can be defined depending on the way computing resources are shared, e.g.:



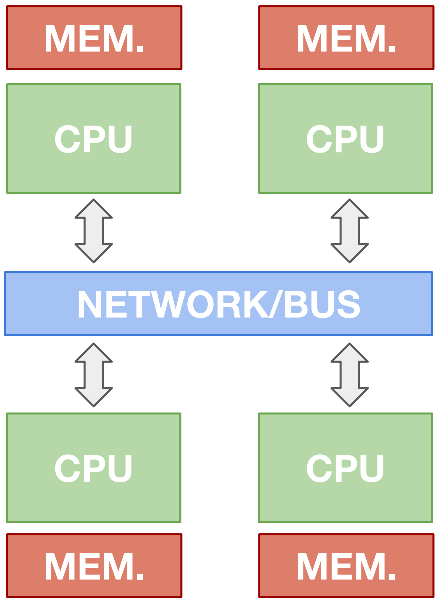
Shared Memory MultiProcessor (SMP)

Memory is shared across a number of parallel CPUs

Ideally all modern computing platforms (including smartphones!) are based on SMP architectures

Scaling this system to incorporate a massive number of processors is demanding and (very) expensive

A number of alternative architectures can be defined depending on the way computing resources are shared, e.g.:



Distributed Memory MultiProcessor (Cluster)

Each processor is associated with its own memory → a *node*

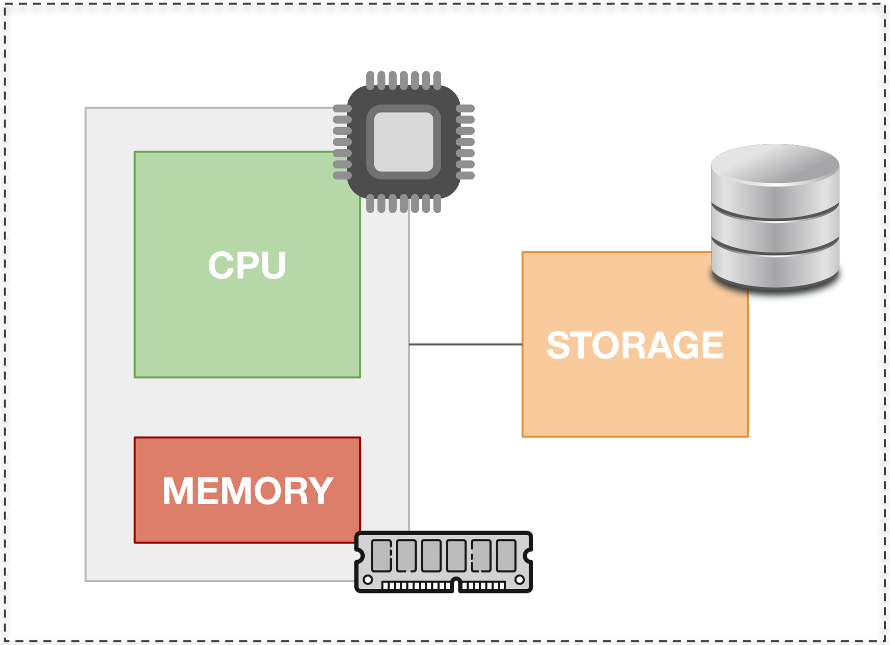
All nodes are connected via a high speed network

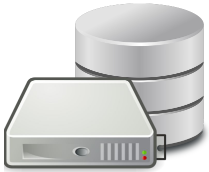
Can obtain good overall performances even from commodity HW

A lot of the complexity is in the communication across the individual units

DISTRIBUTED COMPUTING: CLUSTER

Distribute the processing across multiple *nodes* of a set of computing resources linked together by fast speed networks

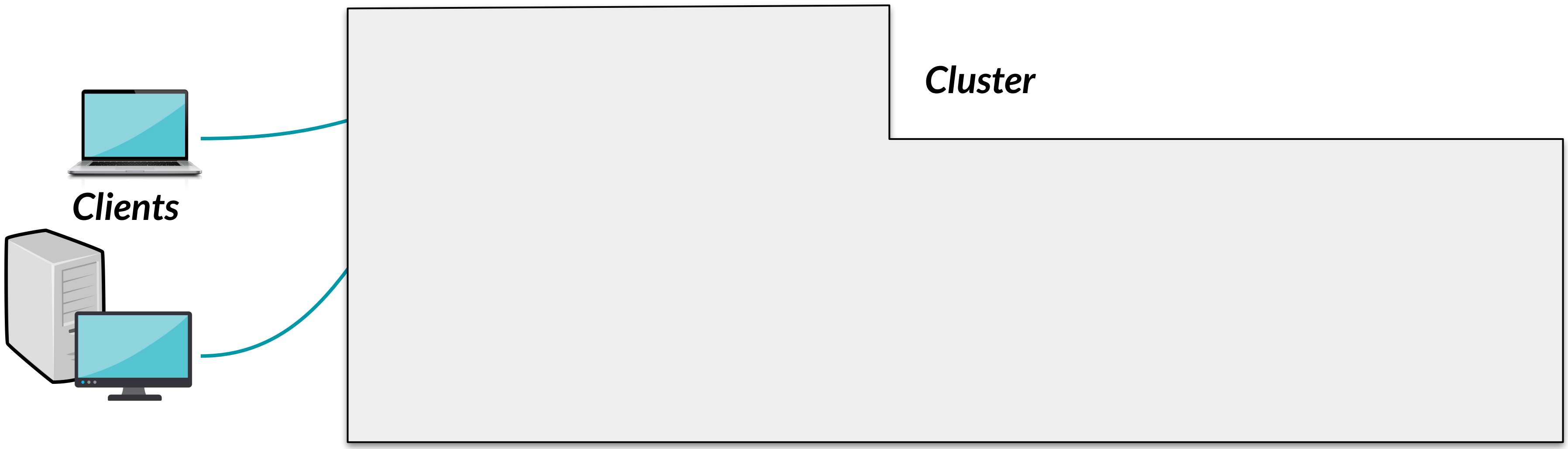




One computing node

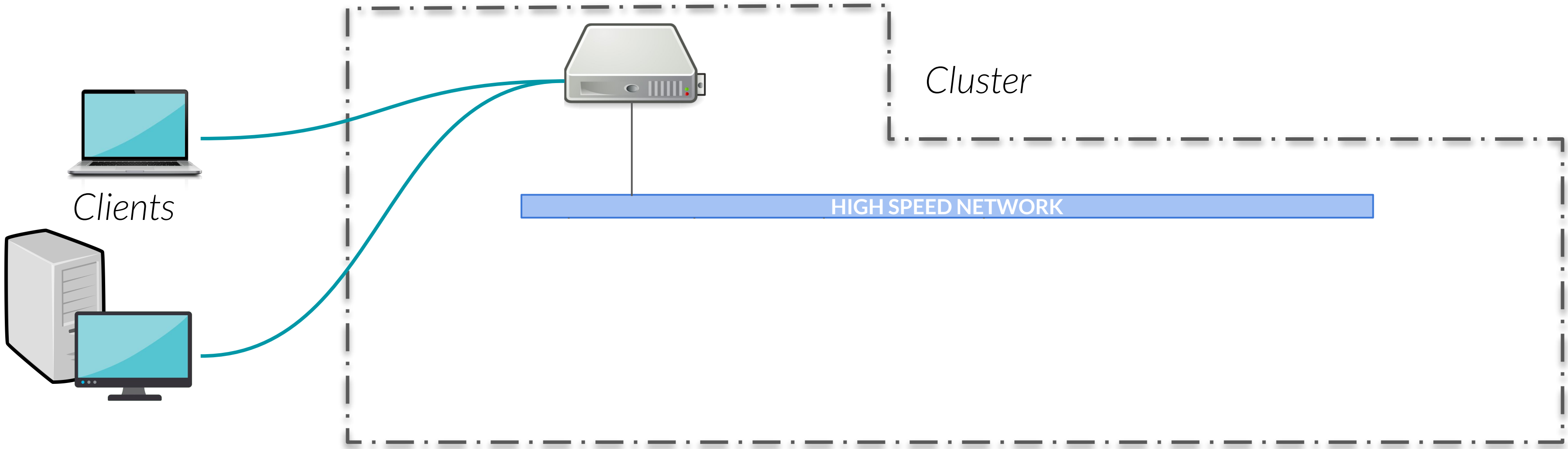
DISTRIBUTED COMPUTING: CLUSTER

The nodes share computational workload as a “virtual massive computer”, being exposed to the users as a single system



DISTRIBUTED COMPUTING: CLUSTER

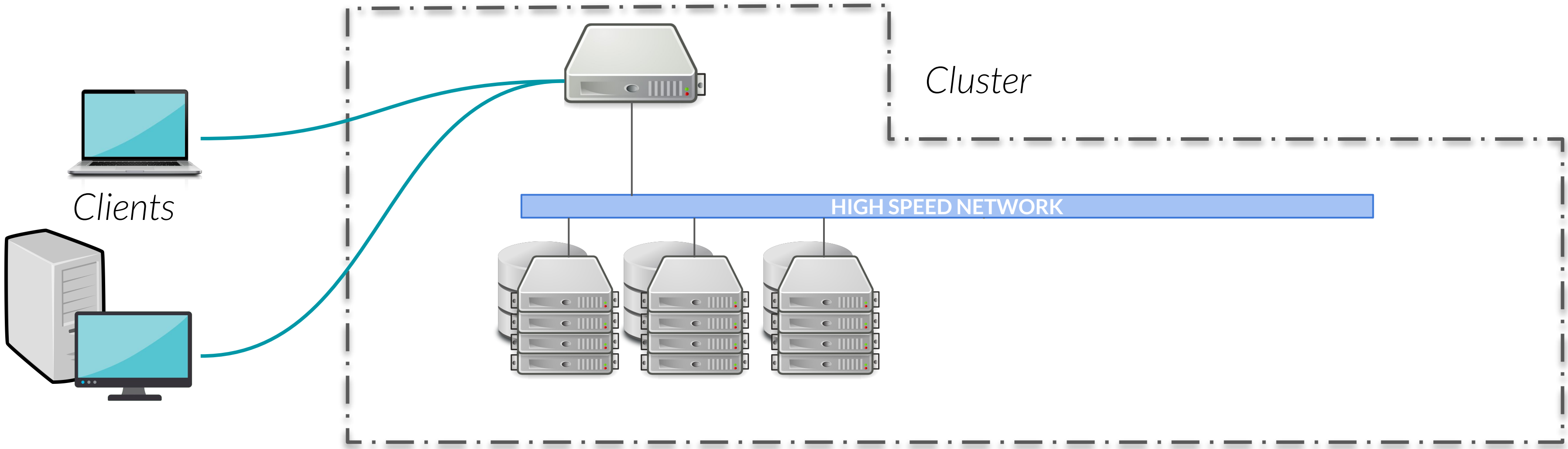
The nodes share computational workload as a “virtual massive computer”, being exposed to the users as a single system



Head node → Access point to the cluster for the clients, distributing the workload to CNs

DISTRIBUTED COMPUTING: CLUSTER

The nodes share computational workload as a “virtual massive computer”, being exposed to the users as a single system

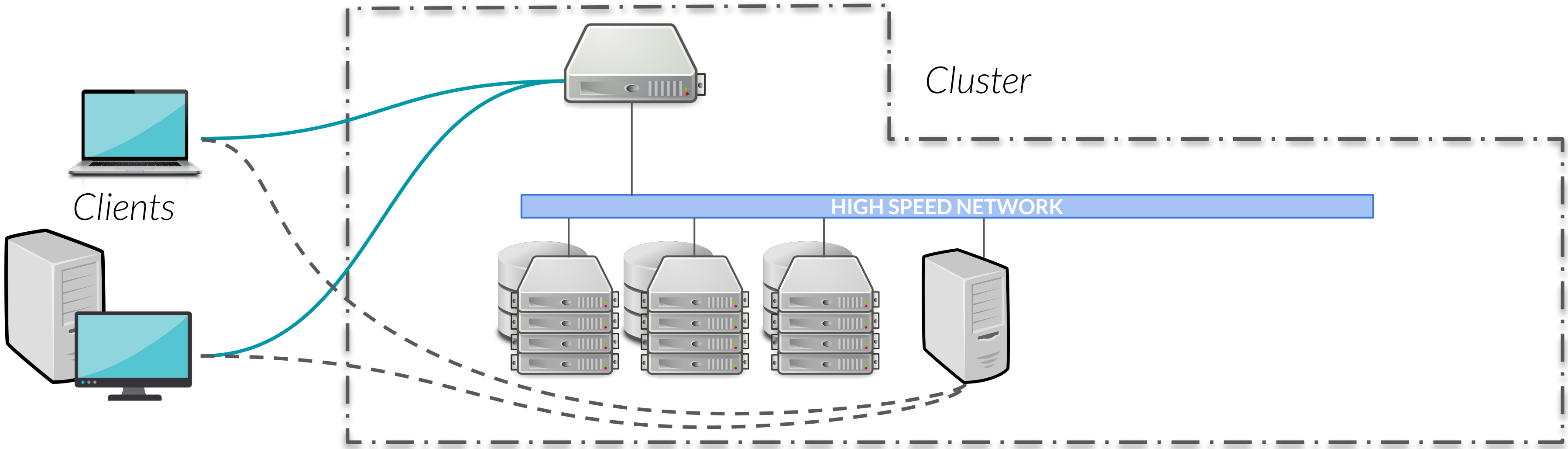


Head node → Access point to the cluster for the clients, distributing the workload to CNs

Computing nodes → Nodes where the actual processing is executed. Usually “invisible” for the users, accessed only via the HN

DISTRIBUTED COMPUTING: CLUSTER

The nodes share computational workload as a "virtual massive computer", being exposed to the users as a single system



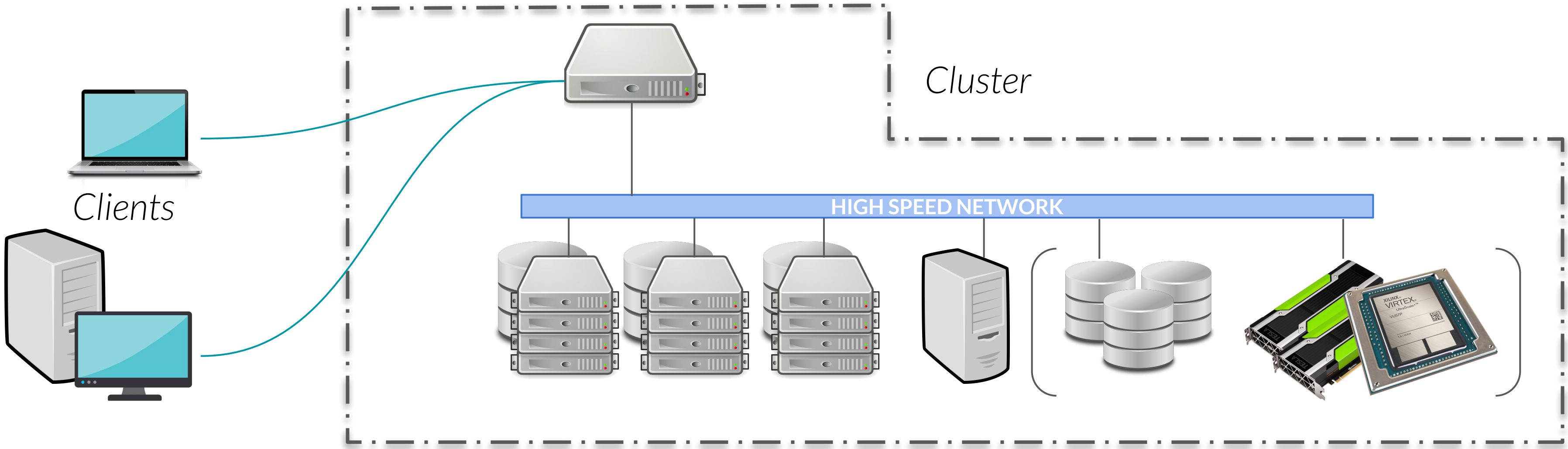
Head node → Access point to the cluster for the clients, distributing the workload to CNs

Computing nodes → Nodes where the actual processing is executed. Usually "invisible" for the users, accessed only via the HN

***Interactive nodes** → Accessible to users as standard computers. Used for testing, storage access, or launching jobs

DISTRIBUTED COMPUTING: CLUSTER

The nodes share computational workload as a “virtual massive computer”, being exposed to the users as a single system



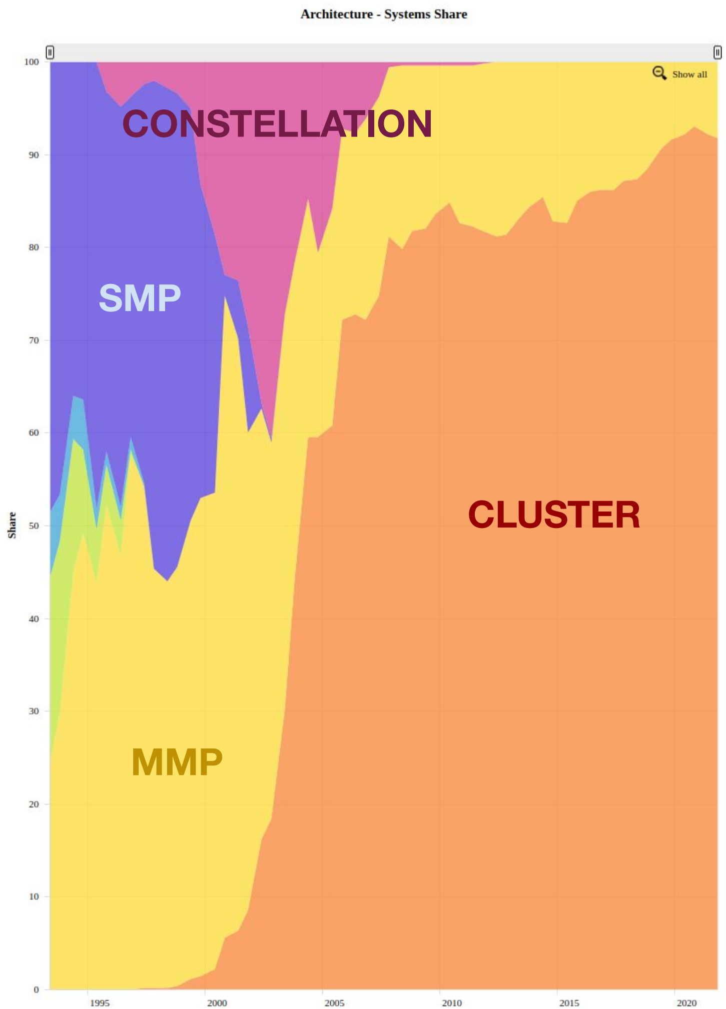
Head node → Access point to the cluster for the clients, distributing the workload to CNs

Computing nodes → Nodes where the actual processing is executed. Usually “invisible” for the users, accessed only via the HN

***Interactive nodes** → Accessible to users as standard computers. Used for testing, storage access, or launching jobs

***Specialized nodes** → Nodes with specific hardware capabilities for computing/storage

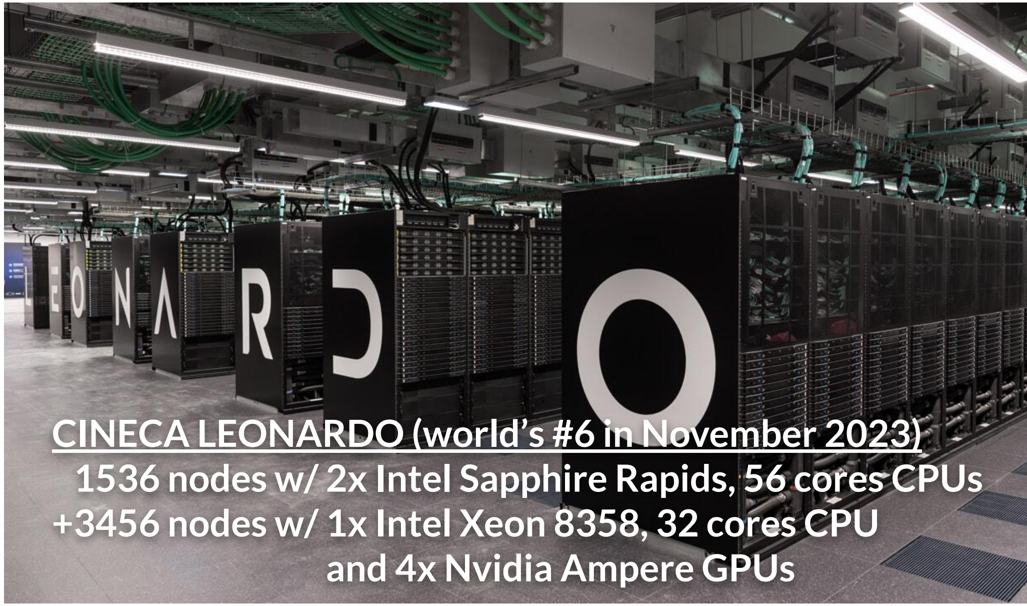
FROM WORLD'S TOP500 SUPERCOMPUTERS



https://www.top500.org/statistics/overtime/

Most of today's supercomputers are actually clusters

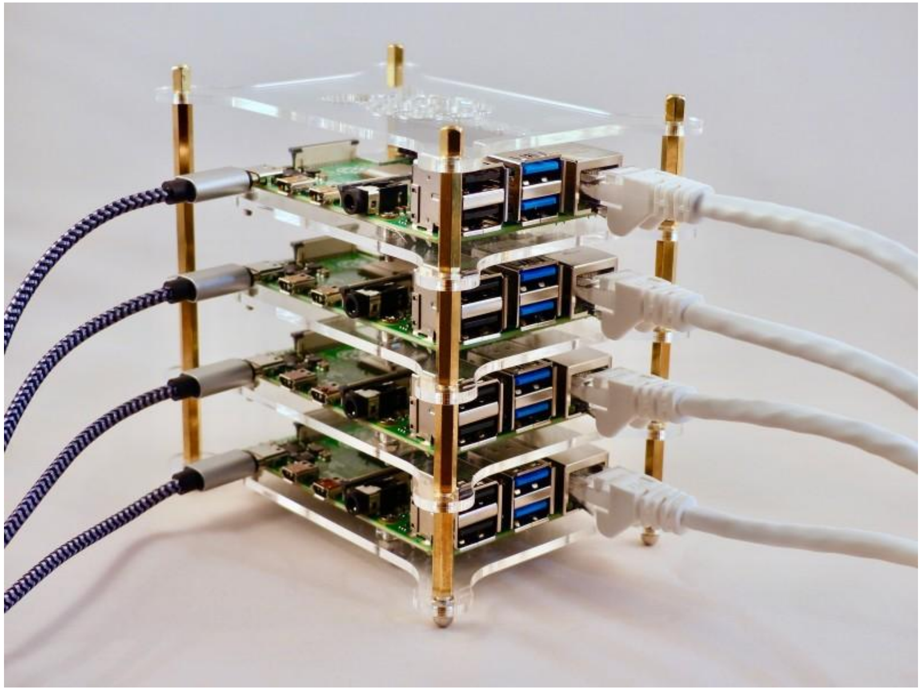
- Extremely powerful processing HW in each node
- Frequently with additional accelerators (GPUs, i.e. SIMD/SIMT-like)
- Hosting a vast number of nodes

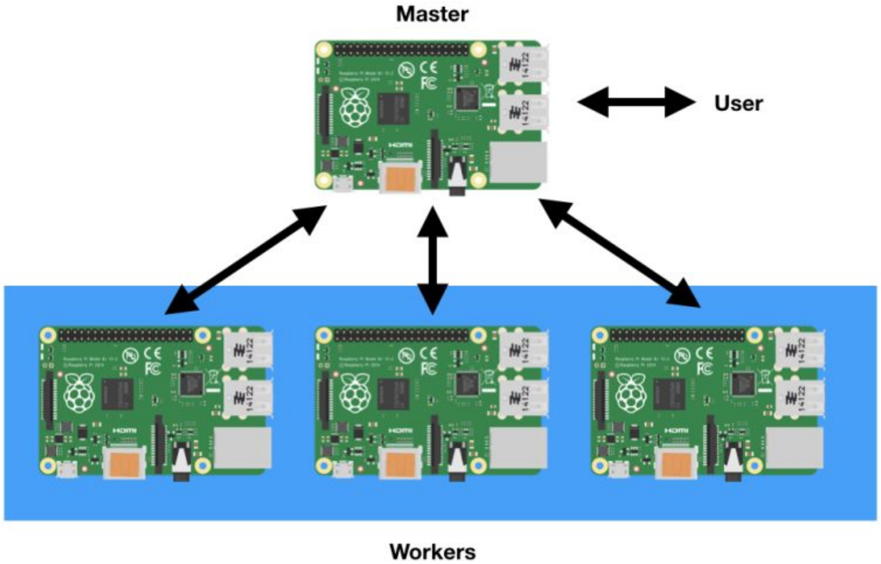


TO WORLD'S SMALLEST CLUSTERS

On the other hand, clusters can be an excellent solution to exploit modest HW to efficiently perform parallel tasks

- Usage of Commodity Off The Shelf (COST) HW
- Opportunistic computing (use shared computing resources when others are not)





TO WORLD'S SMALLEST CLUSTERS

On the other hand, clusters can be an excellent solution to exploit modest HW to efficiently perform par

- Usage
- Oppor re not)

A COMMON JOB SUBMISSION PATTERN: BATCH SYSTEMS

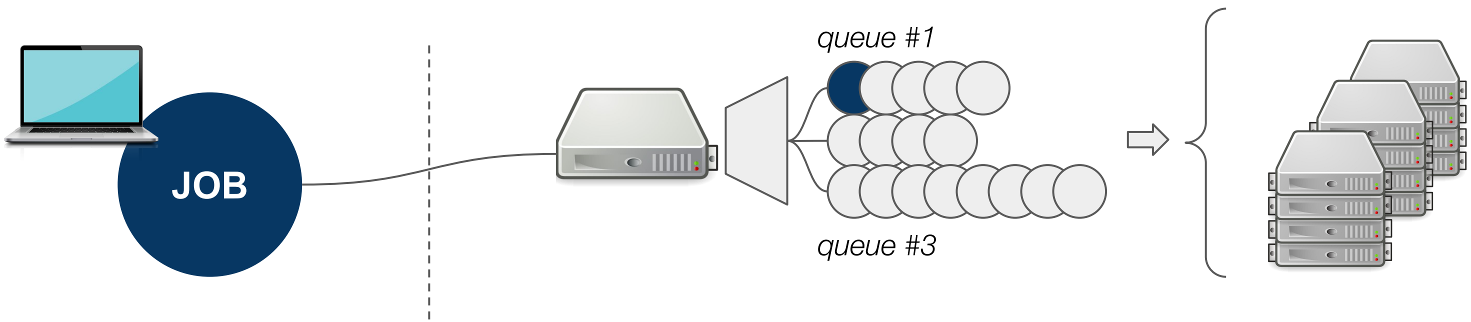
Users usually schedule and submit jobs to a cluster through a batch system, each job including:

1. an executable + optional arguments (e.g. n. of CPUs, memory, …)
2. info on potential requirements (e.g. libraries, …)

The batch system accepts the job and put in the desired *queue*

The batch system's scheduler manages the current workload, and determines which job will start next (depending on priorities / previous use of resources / …)

The tasks contained in the job's instructions are dispatched on the Computing nodes for execution



A COMMON JOB SUBMISSION PATTERN: BATCH SYSTEMS

Several batch system schedulers available:

- PBS
- SLURM
- LSF
- HTCondor
- …

```
$> condor_submit my_job.sub
Submitting job(s).
50 job(s) submitted to cluster 5146936.

$> condor_q
-- Schedd: bigbird12.cern.ch : <137.138.120.116:9618?... @ 10/04/20 17:29:52
OWNER    BATCH_NAME     SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
jpazzini ID: 5146936  10/4  17:29     12    25      13     50 5146936.0-49
```

A COMMON JOB SUBMISSION PATTERN: BATCH SYSTEMS

Several batch system schedulers available:

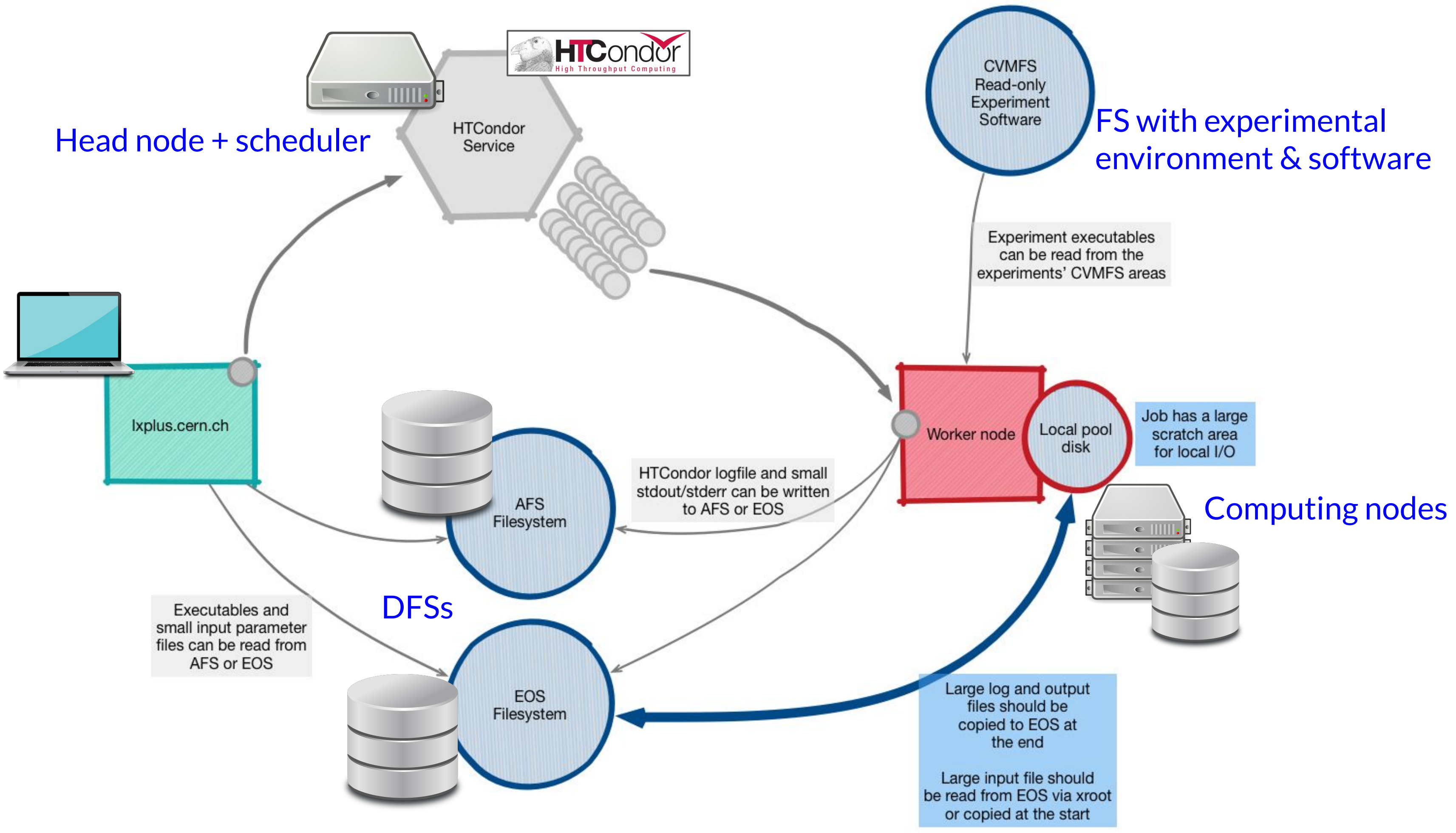
- PBS
- SLURM
- LSF
- HTCondor
- …

```
executable = my_exec    #or my script
arguments = --input input_file_$(ProcId).txt

transfer_input_files = input_file_$(ProcId).txt

log = my_job_$(ProcId).log
error = my_job_$(ProcId).err
output = my_job_$(ProcId).out

request_disk = 8GB
request_memory = 2GB

queue 50    #number of tasks to run
```

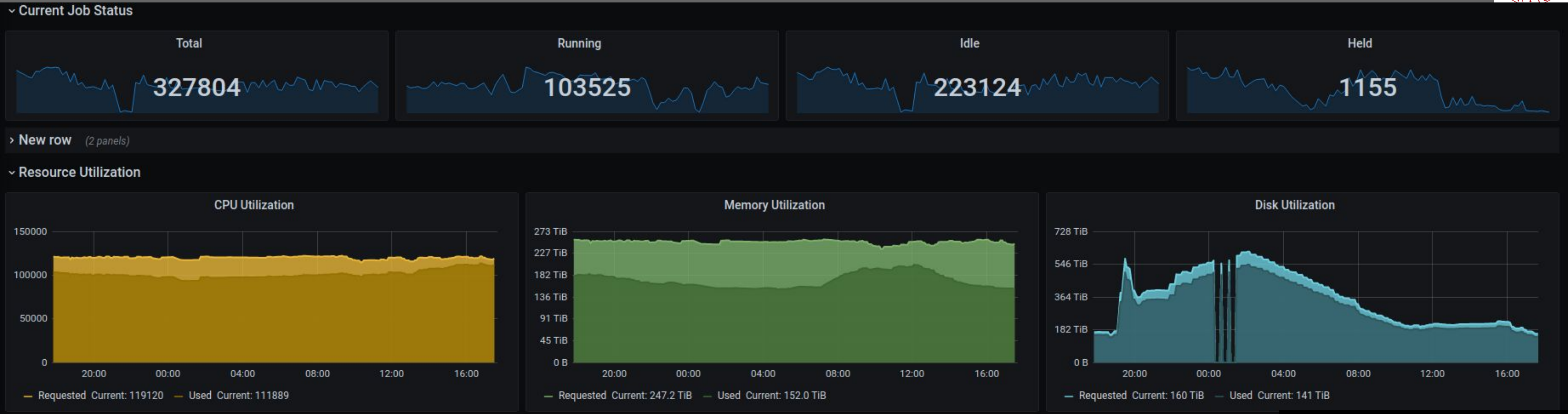
```
$> condor_submit my_job.sub
Submitting job(s).
50 job(s) submitted to cluster 5146936.

$> condor_q
-- Schedd: bigbird12.cern.ch : <137.138.120.116:9618?... @ 10/04/20 17:29:52
OWNER    BATCH_NAME      SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
jpazzini ID: 5146936   10/4  17:29     12    25      13     50 5146936.0-49
```

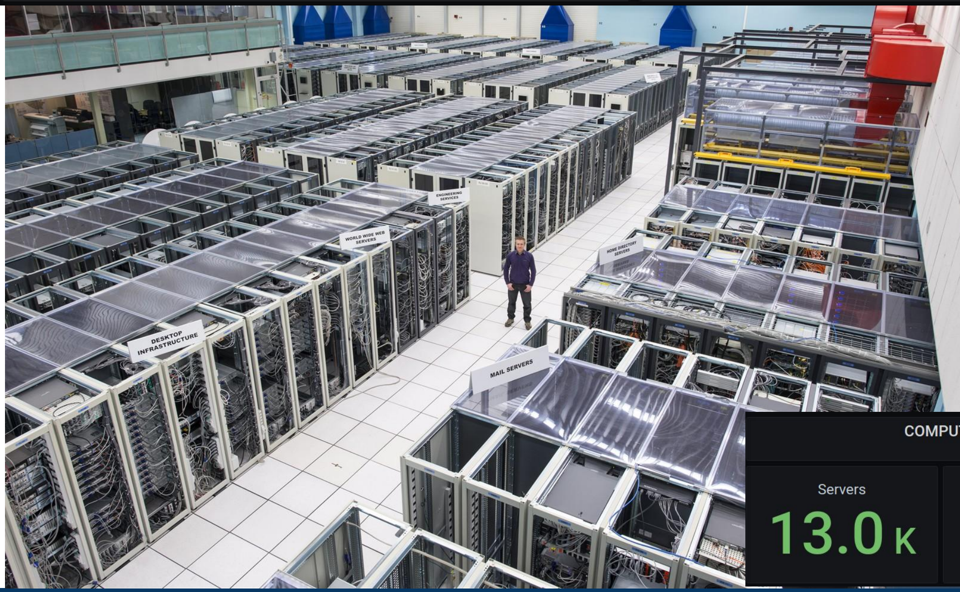
HTCondor High Throughput Computing

BATCH SYSTEMS - CERN EXAMPLE

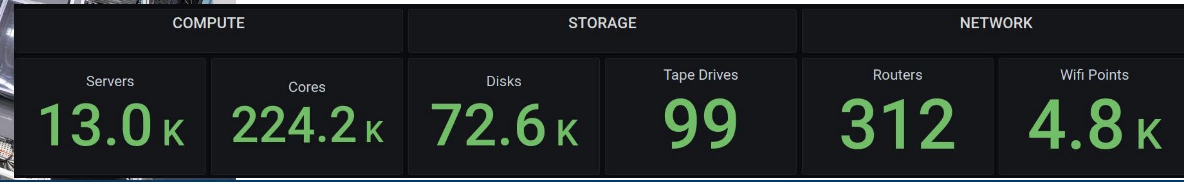


BATCH SYSTEMS - CERN EXAMPLE





… over 24 h



Optimizing the execution of complex tasks using parallel and distributed computing systems requires a knowledge of specific programming standards and frameworks, including:

- *OpenMP (Message Passing)* to handle parallelization, typically on single nodes with shared-memory multiple CPUs
- *Message Passing Interfaces (MPI)* to handle communication and synchronization across nodes of a cluster
- *CUDA/HIP* to handle vector processing on GPUs
- …

Optimizing the execution of complex tasks using parallel and distributed computing systems requires a knowledge of specific programming standards and frameworks, including:

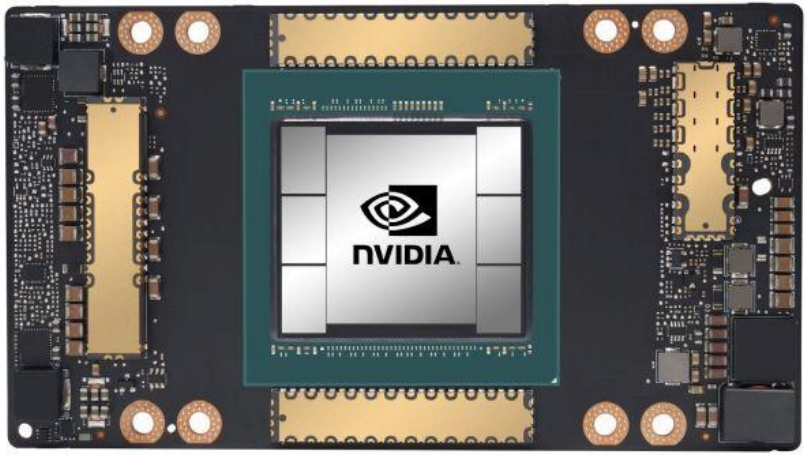
- *OpenMP (Message Passing)* to handle parallelization, typically on single nodes with shared-memory multiple CPUs
- *Message Passing Interfaces (MPI)* to handle communication and synchronization across nodes of a cluster
- *CUDA/HIP* to handle vector processing on GPUs
- …

In our class we'll focus on a relatively simple paradigm for data processing in distributed systems which was at the core of BigData blow-up in the 2000s-2010s

For its applications we'll use modern distributed processing frameworks which require little-to-no knowledge of the innermost workings of distributed systems

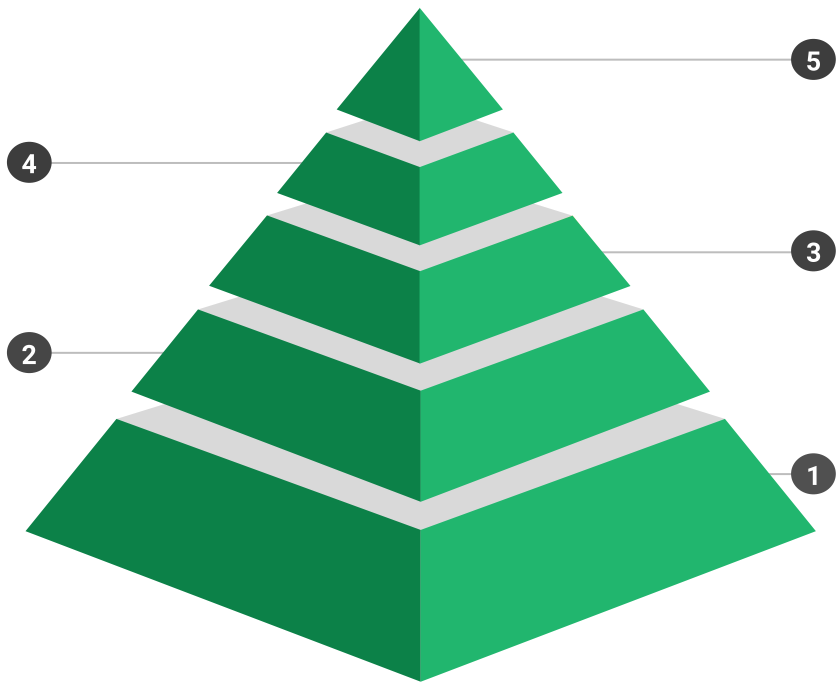
PARALLEL PROGRAMMING FOR GPUs

If interested in diving deeper into parallel programming for GPU-enabled acceleration, consider the optional course **Modern Computing for Physics**



PARALLEL PROGRAMMING FOR GPUs

If interested in diving deeper into parallel programming for GPU-enabled acceleration, consider the optional course **Modern Computing for Physics**



CUDA in Python

- Higher-level abstractions for programming GPUs
- Numba as a bridge between Python and CUDA-C
- An excursion on the available “tools” PyCuda, CuPy, PyTorch, RAPIDS…

General Purpose GPUs

- Understanding the hardware to be able exploit it
- Differences between CPU vs GPU
- Handling memory and thread execution

Streaming processing

- Using GPUs to process continuous streams of data

Basics of CUDA

- How to create and launch threads for vector/matrix processing on GPU
- Low-level language (C++) to understand the basics
- Higher-level programming models such as OpenACC

Parallel processing/programming

- Where and how parallel programming impacts physics computation
- Recap on concurrency and parallelization
- Figures of merits for GPU programming