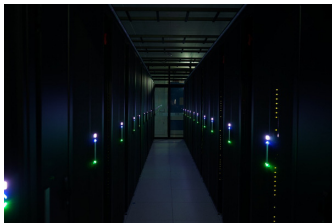


# UL HPC Monitoring in practice: why, what, how, where to look

Clément Parisot

UL HPC School  
June 12, 2017

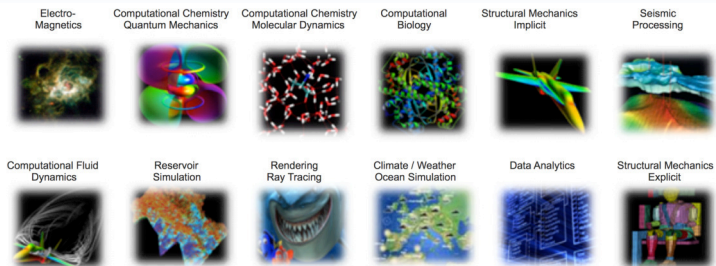


Slides adapted from *Xavier Besseron* (UL HPC School 2015)

# Outline

- 1 What is HPC?
- 2 Best Practices for UL HPC
- 3 HPC: Getting Fast and Efficient

# Computer Simulation is everywhere



- Computational Fluid Dynamics ([OpenFOAM](#))
- Finite Element Analysis ([Abaqus](#))
- Climate / Weather / Ocean Simulation ([WRF](#))
- Molecular Dynamics ([GROMACS](#), [Amber](#))
- Quantum Chemistry ([Quantum Espresso](#))
- Visualization ([Paraview](#))
- Data processing ([R](#), [Matlab](#))
- ...

# What is High Performance Computing?

## High Performance Computing (HPC)

- Use of parallel and distributed computers with fast interconnects
- To execute an application **quickly** and **efficiently**

## Why parallel computers?

- Performance of single CPU core is getting limited (power, physics)
- Multiple cores are used to increase the computing capacity

## HPC is challenging

- Active research domain
- **Provides tools for many other researchers**

# Can you benefit from HPC?

Your application is limited by the performance of your computer



	Your workstation	UL HPC platform <sup>1</sup>
CPU	160 <b>G</b> Flops	197.7 <b>T</b> Flops
Memory	16 <b>G</b> Bytes	42.8 <b>T</b> Bytes
Storage	2 <b>T</b> Bytes	5.4 <b>P</b> Bytes
Network	Ethernet 1 Gb/s	Infiniband 100 Gb/s
Accelerators	1 GPU	50 GPUs

⇒ HPC provides the **tools** for your application to run faster

<sup>1</sup>shared among many users

# Other benefits of using the UL HPC platform

over using your personal computer

## Long uptime and stable

- Cluster nodes are always running

## Remote access

- Start a job from work, check results from home

## No administration

- The HPC team maintain the hardware/software

## Large software collection

- Scientific and general-purpose applications pre-installed

## Backup

- Automatic backup of your Home directory<sup>2</sup>

<sup>2</sup>Always keep a backup of your critical data!



# Outline

- 1 What is HPC?
- 2 Best Practices for UL HPC
- 3 HPC: Getting Fast and Efficient

# Know the basics!

## Get an account

- [https://hpc.uni.lu/users/get\\_an\\_account.html](https://hpc.uni.lu/users/get_an_account.html)
- Please read carefully the [Acceptable Use Policy](#)

## Access the clusters, access and reserve nodes

- Use SSH and public key authentication  
<https://hpc.uni.lu/users/docs/access.html>
- Learn how to use the OAR resource manager  
<https://hpc.uni.lu/users/docs/oar.html>
- Learn how to use the SLURM batch scheduler (Iris cluster only)  
<https://hpc.uni.lu/users/docs/slurm.html>

## Transfer files between your computer and the clusters

- Learn how to use tools like `scp`, `rsync`, etc.  
<https://hpc.uni.lu/users/docs/filetransfer.html>

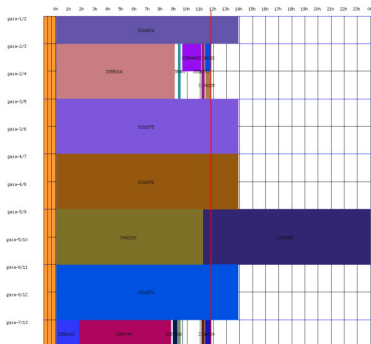
## Use pre-installed software

- Search and use software with the `module` command  
<https://hpc.uni.lu/users/docs/modules.html>



# Check Live Status of the platform

- Current node status with Monika<sup>1</sup>  
<https://hpc.uni.lu/status/monika.html>
- Platform occupation – Gantt chart with DrawGantt<sup>1</sup>  
<https://hpc.uni.lu/status/drawgantt.html>
- Resource usage with Ganglia  
<https://hpc.uni.lu/status/ganglia.html>



DrawGantt



Ganglia

<sup>1</sup>Not available on Iris cluster

# Getting help

- Check the UL HPC quick reference

<https://hpc.uni.lu/download/documents/ulhpc-quickref.pdf>

- RTFM! Online Documentation available at

<https://hpc.uni.lu/users/docs/>

- Google is your friend!
- Ask other users on the mailing list `hpc-users@uni.lu`
- Ask the HPC sys-admins `hpc-sysadmins@uni.lu`

# Workflow for Experiment Campaigns

## 1. Before the campaign

- Send data to the clusters
- Check/Install required software

## 2. Preparation

- Test and debug
- Prepare launcher script

## 3. Execution

- Run the campaign
- Monitor the execution

## 4. After a campaign

- Retrieve output data
- Archive and cleanup your data

# Workflow for Experiment Campaigns

## 1. Before the campaign → *cf* the Basics

- Send data to the clusters
- Check/Install required software

## 2. Preparation

- Test and debug
- Prepare launcher script

## 3. Execution

- Run the campaign
- Monitor the execution

## 4. After a campaign → *cf* the Basics

- Retrieve output data
- Archive and cleanup your data

# Experiment campaign: Preparation

## Goals

- Make sure everything will run OK
- Prepare submission script / launcher

## Interactive approach

- Use option `-I` (Interactive) of `oarsub` command
- Allows to try commands one by one
- Work on a small case with a small number of cores
- Debug and check the results

## Why prepare a submission script?

- Contains all commands and parameters  
⇒ Easy re-execution
- No need to stay in front your computer

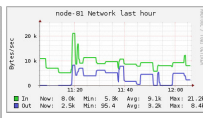
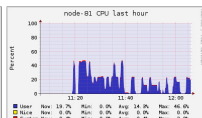
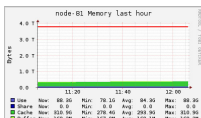
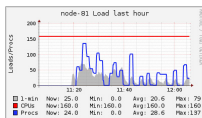
# Experiment campaign: Execution

## Submit the jobs

- Use the submission script / launcher
- Submit to OAR with option `-S` (Script) of `oarsub`
- Actual experiment execution with possibly many nodes
- Non interactive execution, it might not start immediately

## Monitor the execution

- Status of your job: `oarstat -j <OAR_JOBID> -f`
- Output/Logfile of your application
- Resource usage (CPU, memory, etc.)  
on the node: `htop`  
with Ganglia: <https://hpc.uni.lu/status/ganglia.html>



# Experiment campaign: Execution (2)

## Monitor your job with Slurm

- Use the submission script / launcher
- Submit to Slurm with `sbatch your_script.sh` (Slurm batch Script)

## Monitor the execution

- Status of your job: `squeue -u`
- Output/Logfile of your application
- Resource usage (CPU, memory, etc.)  
with Slurm: `sstat -job SLURM_JOBID -l`  
on the node: `htop`  
with Ganglia: <https://hpc.uni.lu/status/ganglia.html>

# Outline

- 1 What is HPC?
- 2 Best Practices for UL HPC
- 3 HPC: Getting Fast and Efficient**

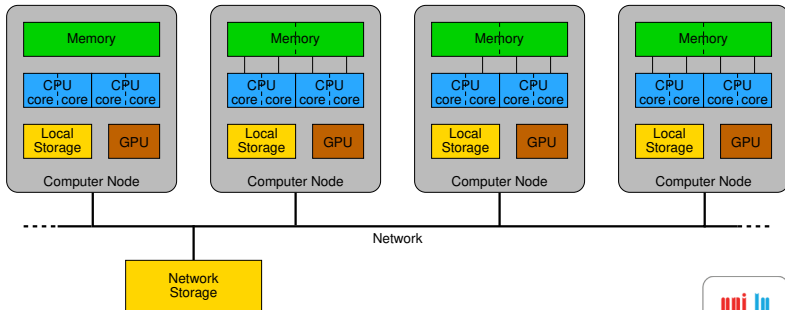


# Getting faster: Identify performance bottlenecks

Note for code developers: **The first bottleneck is your algorithm!**

## Know the hardware

- Computer nodes are connected using a fast interconnect
- Different types of resources:  
Processors, GPU, Memory, Storage, Network

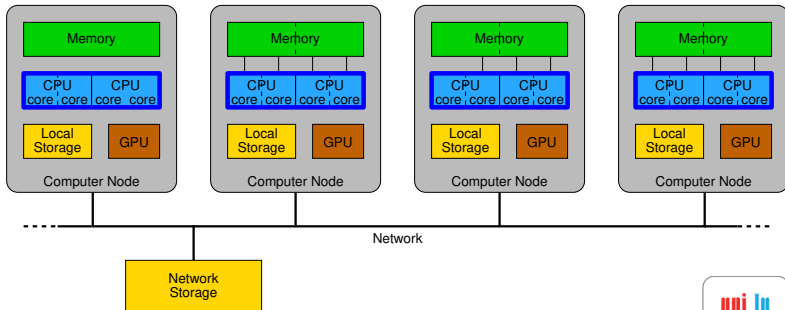


# Getting faster: Identify performance bottlenecks

Note for code developers: **The first bottleneck is your algorithm!**

## Know the hardware

- Computer nodes are connected using a fast interconnect
- Different types of resources:  
Processors, GPU, Memory, Storage, Network

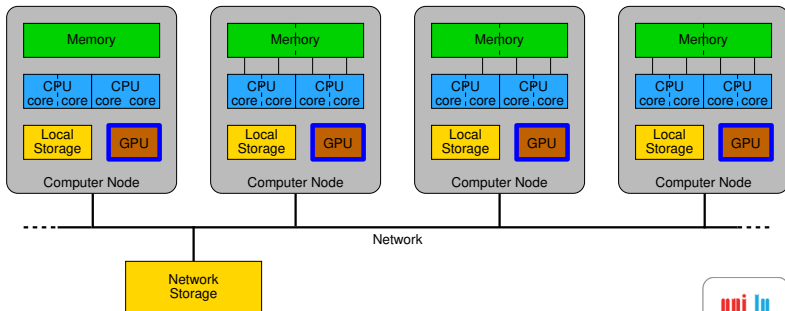


# Getting faster: Identify performance bottlenecks

Note for code developers: **The first bottleneck is your algorithm!**

## Know the hardware

- Computer nodes are connected using a fast interconnect
- Different types of resources:  
Processors, GPU, Memory, Storage, Network

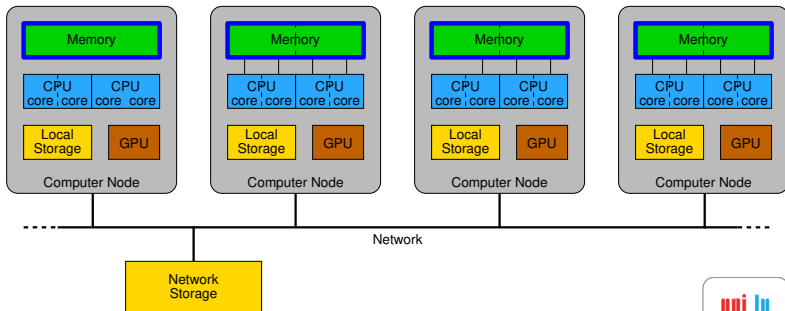


# Getting faster: Identify performance bottlenecks

Note for code developers: **The first bottleneck is your algorithm!**

## Know the hardware

- Computer nodes are connected using a fast interconnect
- Different types of resources:  
Processors, GPU, **Memory**, Storage, Network

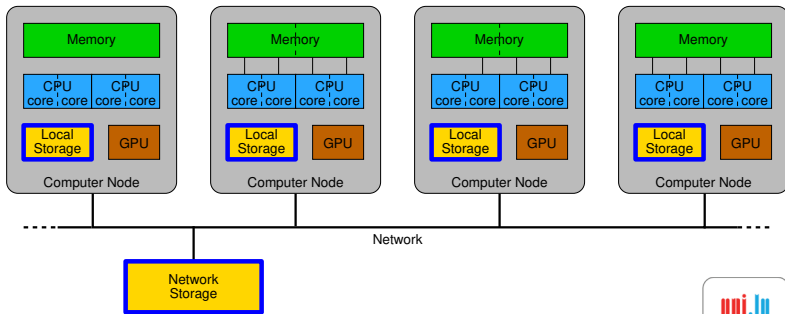


# Getting faster: Identify performance bottlenecks

Note for code developers: **The first bottleneck is your algorithm!**

## Know the hardware

- Computer nodes are connected using a fast interconnect
- Different types of resources:  
Processors, GPU, Memory, **Storage**, Network

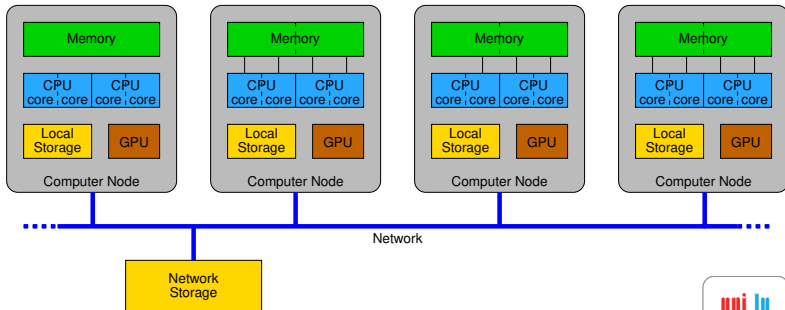


# Getting faster: Identify performance bottlenecks

Note for code developers: **The first bottleneck is your algorithm!**

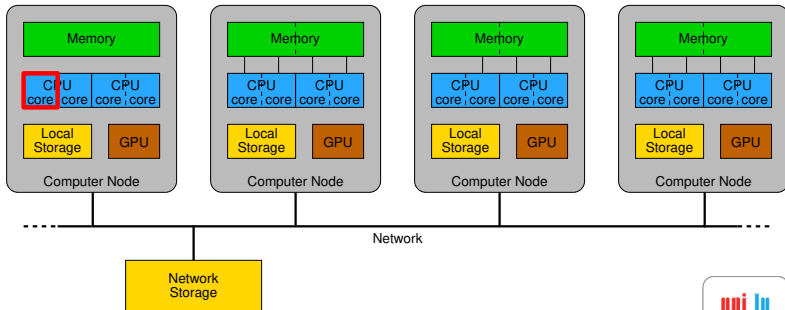
## Know the hardware

- Computer nodes are connected using a fast interconnect
- Different types of resources:  
Processors, GPU, Memory, Storage, **Network**



# Processor bottleneck

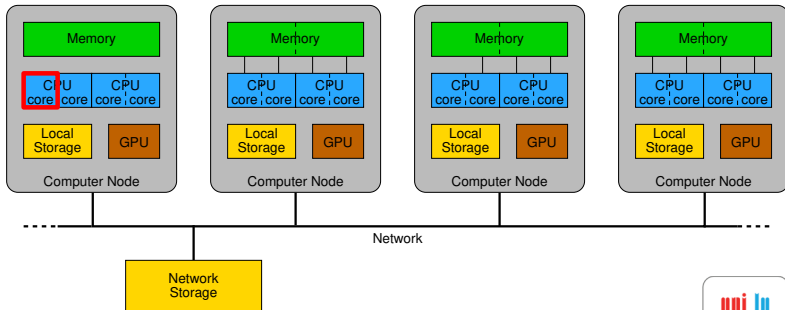
Application is limited by the speed of the processor



# Processor bottleneck

Application is limited by the speed of the processor

→ Optimize your code

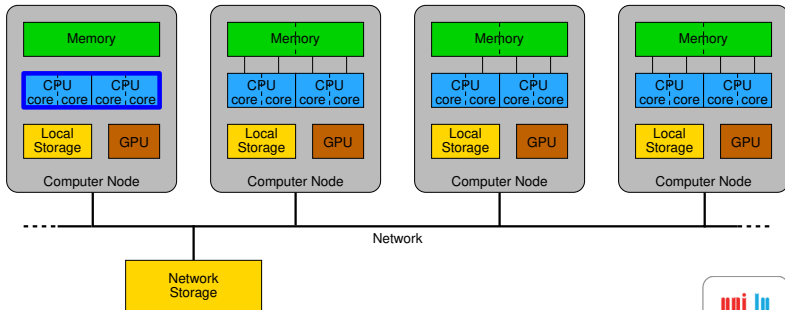




# Processor bottleneck

Application is limited by the speed of the processor

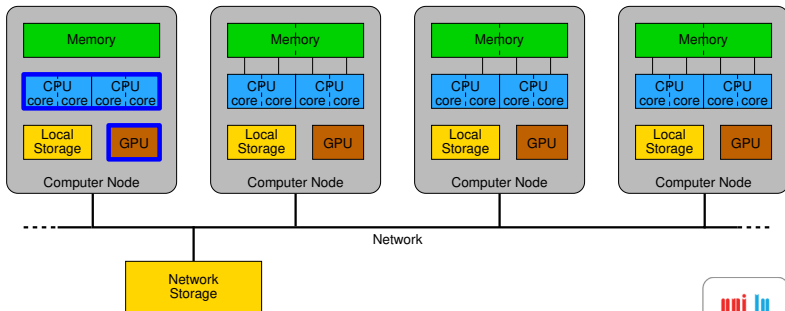
- Optimize your code
- Parallel execution on a single node ([pthread](#), [OpenMP](#), [Intel TBB](#))



# Processor bottleneck

Application is limited by the speed of the processor

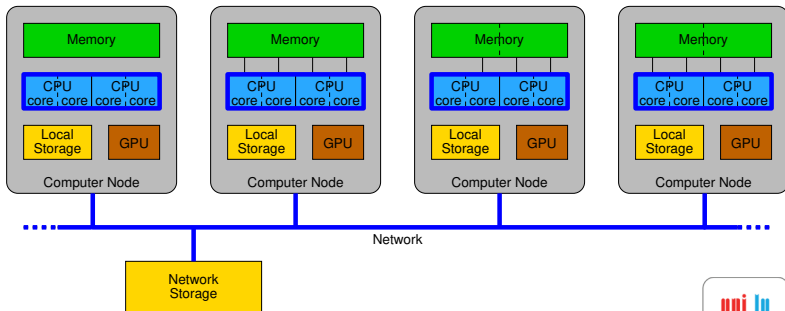
- Optimize your code
- Parallel execution on a single node ([pthread](#), [OpenMP](#), [Intel TBB](#))
- Use GPU accelerator ([CUDA](#))



# Processor bottleneck

Application is limited by the speed of the processor

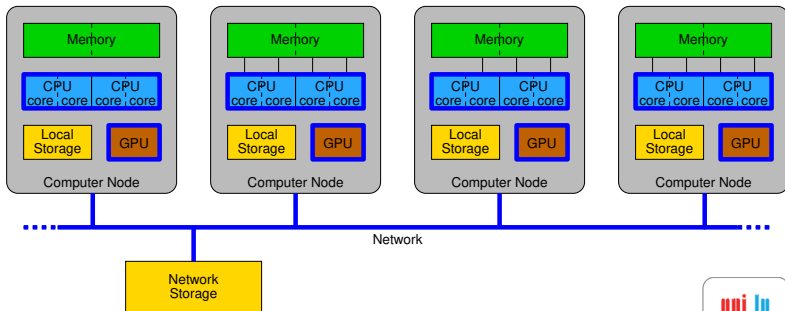
- Optimize your code
- Parallel execution on a single node ([pthread](#), [OpenMP](#), [Intel TBB](#))
- Use GPU accelerator ([CUDA](#))
- Parallel execution on multiple nodes ([MPI](#))



# Processor bottleneck

Application is limited by the speed of the processor

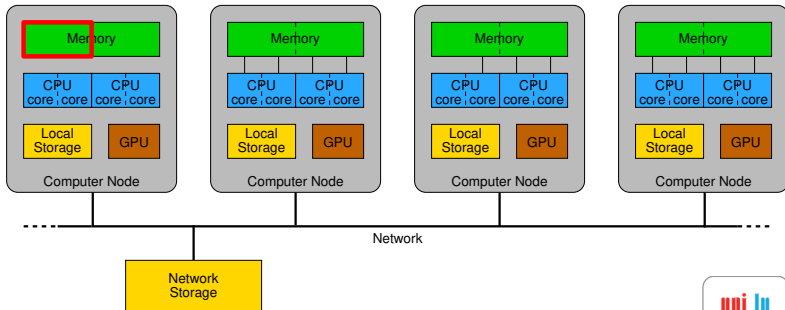
- Optimize your code
- Parallel execution on a single node ([pthread](#), [OpenMP](#), [Intel TBB](#))
- Use GPU accelerator ([CUDA](#))
- Parallel execution on multiple nodes ([MPI](#))
- Parallel execution on multiple nodes with GPUs ([MPI+CUDA](#))



# Memory bottleneck

Application is limited by the size of the memory

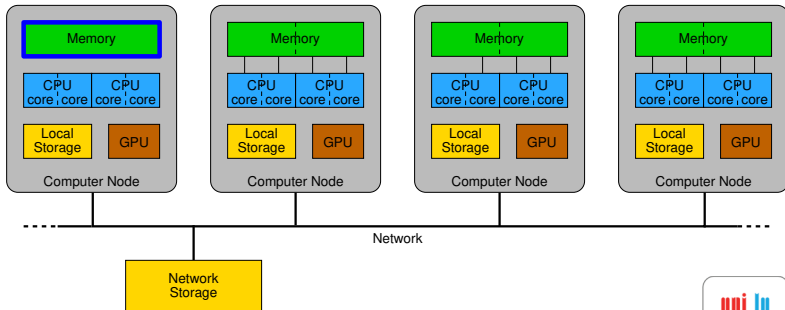
- There is one memory bank per CPU



# Memory bottleneck

Application is limited by the size of the memory

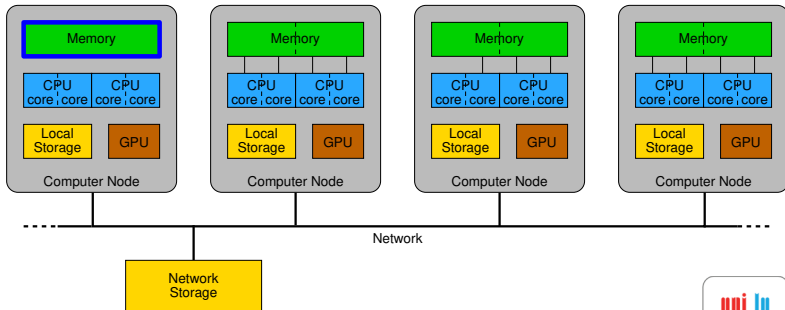
- There is one memory bank per CPU
- Allocate all CPUs on a single node



# Memory bottleneck

Application is limited by the size of the memory

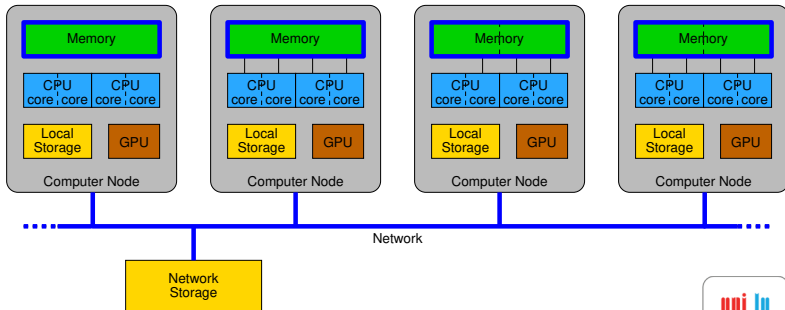
- There is one memory bank per CPU
- Allocate all CPUs on a single node
- Use a node with a bigger memory (4TB-memory node at UL)



# Memory bottleneck

Application is limited by the size of the memory

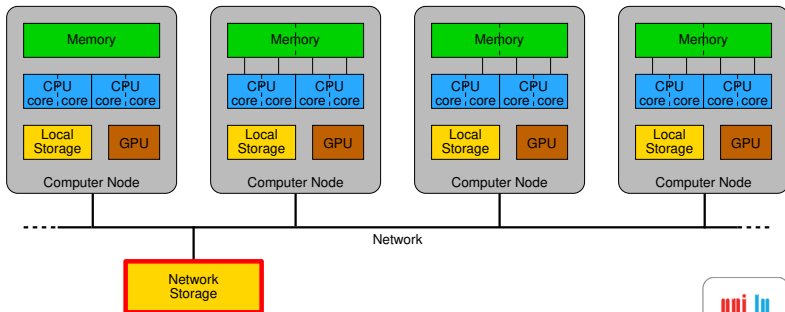
- There is one memory bank per CPU
- Allocate all CPUs on a single node
- Use a node with a bigger memory (4TB-memory node at UL)
- Distributed execution on multiple nodes (MPI)





# Storage space bottleneck

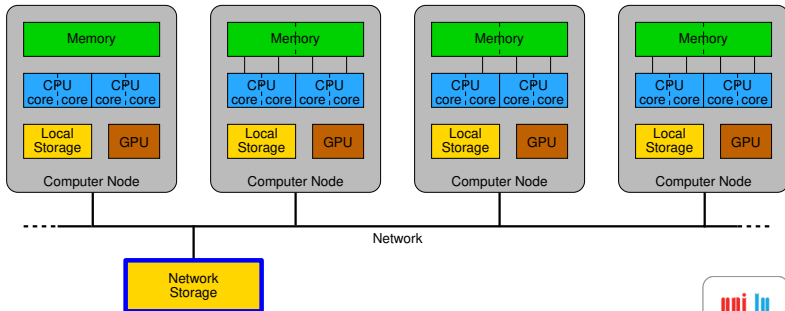
Application is limited by the available storage space



# Storage space bottleneck

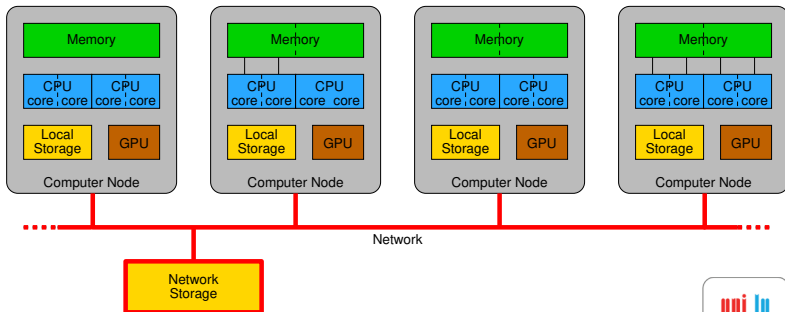
Application is limited by the available storage space

- Use `$WORK` (3 TB) or `$SCRATCH` (10 TB)  
(no backup!)



# Storage speed bottleneck

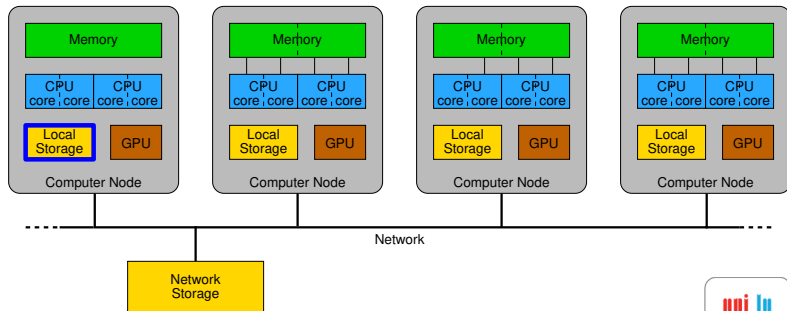
Application is limited by the speed of the storage



# Storage speed bottleneck

Application is limited by the speed of the storage

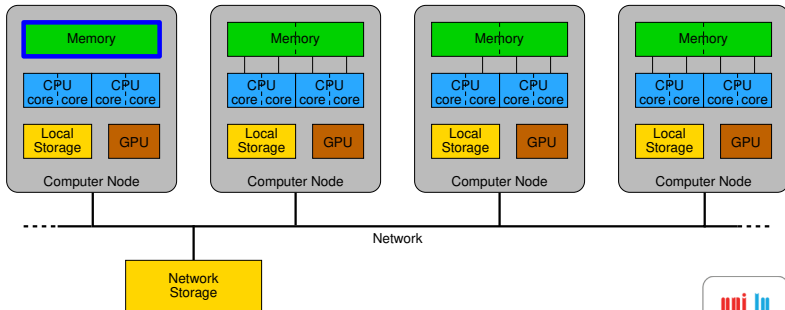
- Use local storage instead of network storage  
(copy data back to network storage after execution)



# Storage speed bottleneck

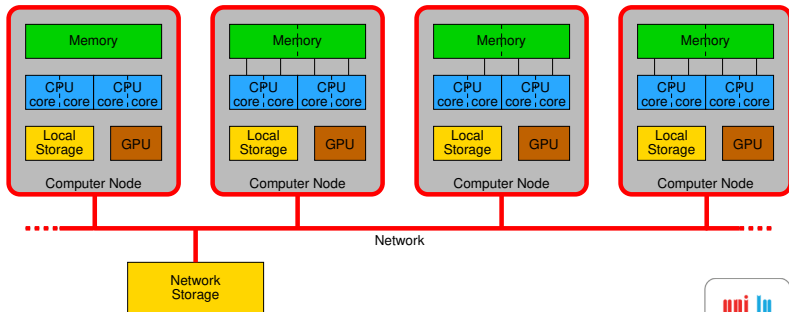
Application is limited by the speed of the storage

- Use local storage instead of network storage  
(copy data back to network storage after execution)
- Use local memory, *eg* `/dev/shm` (space is limited!)



# Network bottleneck

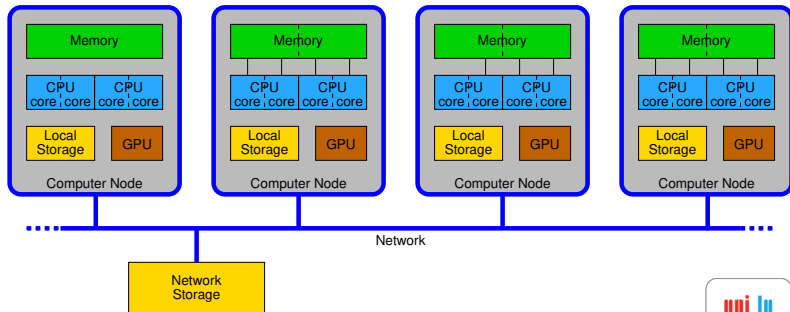
Application is limited by the speed of the network  
(too many communications)



# Network bottleneck

Application is limited by the speed of the network  
(too many communications)

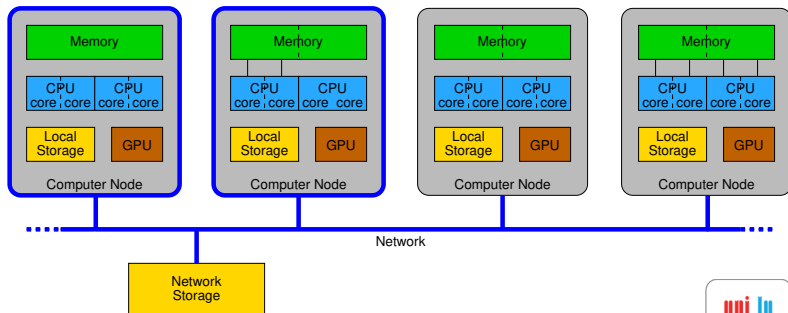
→ Use Infiniband network instead of Ethernet



# Network bottleneck

Application is limited by the speed of the network  
(too many communications)

- Use Infiniband network instead of Ethernet
- Reduce the number of nodes





# Quick Tips for classic use-cases

## Sequential job

- Parallelization: [OpenMP](#), [MPI](#)
- Use accelerators: [CUDA](#)

## Long job

- Checkpoint/Restart: [BLCR](#)

## Large number of jobs

eg parametric studies

- Parallel launcher
- Best effort queue
- OAR Job containers

## Visualization / Rendering

- GPU nodes with graphic session using [XCS portal](#)

## MATLAB

- Parallelization: [Parallel Computing Toolbox](#)
- Checkpoint/Restart: `save()/load()` functions

## R

- Use optimized data structure / package: `data.table` / `plyr`
- Parallelization: packages [parallel](#), [Rmpi](#), [snow](#)

## Python

- Library for scientific computing: [NumPy](#), [Scipy](#), [matplotlib](#)
- Parallelization: [multiprocessing](#) library
- Use latest version of Python

...