



Academic Computer Centre CYFRONET AGH



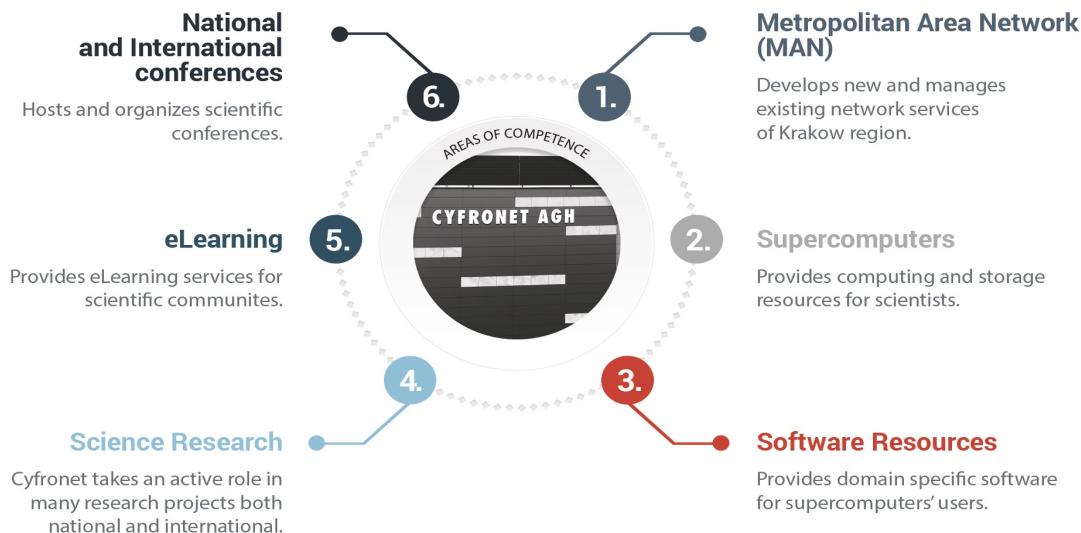
Klemens Noga, Maciej Czuchry, Oskar Klimas

Efficient usage of HPC systems in scientific computations

- **Ares & Athena clusters at ACC Cyfronet AGH**
 - available resources
 - access to clusters/data transfer
- **Performing calculations**
 - software environment management using Modules/Lmod
 - batch scripts
 - sequential and parallel runs
 - efficient usage of SLURM queuing system
- **Documentation and users' support**
- **Questions and exercises**
- **Ares & Athena as a part of PLGrid Infrastructure**
- **EuroHPC (LUMI) - computational opportunities**

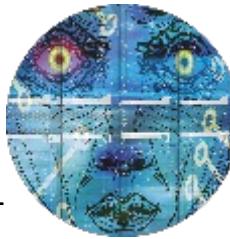


- The biggest Polish Academic **Computer Centre**
 - **50 years of experience** in IT provision
 - Centre of excellence in **HPC, Grid and Cloud Computing**
 - Home for **Athena, Ares, Prometheus** and **Helios supercomputers**
 - **LUMI** consortium partner (EuroHPC pre-exascale supercomputer **#5 on TOP500**)
- Legal status: an **autonomous** within AGH University of Science and Technology
- Staff: 180+ , ca. 80 in R&D
- Leader of **PLGrid**: Polish Grid and Cloud Infrastructure for Science
- NGI Coordination in **EGI e-Infrastructure**



Prometheus

- 2.40 PFLOPS
- 53 568 cores
- From 2015 to 2021
1st HPC system in Poland (475th on Top 500, 38th in 2015)



Athena

- 7.71 PFLOPS
- 384 A100 GPGPUs
- 1st HPC system in Poland (since 2022, 105th on Top500)
- 9th on Green500



Ares

- 4.00 PFLOPS
- 38 112 cores
- 290th on Top 500



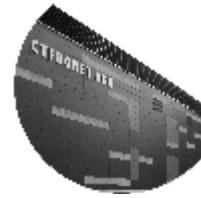
Storage

- 60+ PB
- hierarchical data management



Research & Development

- distributed computing environments
- computing acceleration
- machine learning
- software development & optimization



Computing portals and frameworks

- OneData
- PLG-Data
- Rimrock
- InSilicoLab



Data Centres

- 3 independent data centres
- dedicated backbone links

Computational Cloud

- based on OpenStack



➤ Supercomputers from Poland

- 105 – Athena (ACC Cyfronet AGH) (**PLGrid**) (#9 on Green500)
- 145 – Altair (PSNC) (**PRACE-LAB**)
- 290 – Ares (ACC Cyfronet AGH) (**PLGrid**)
- 462 – Tryton Plus (TASK) (**PLGrid**)
- 475 – Prometheus (ACC Cyfronet AGH) (**PLGrid**)



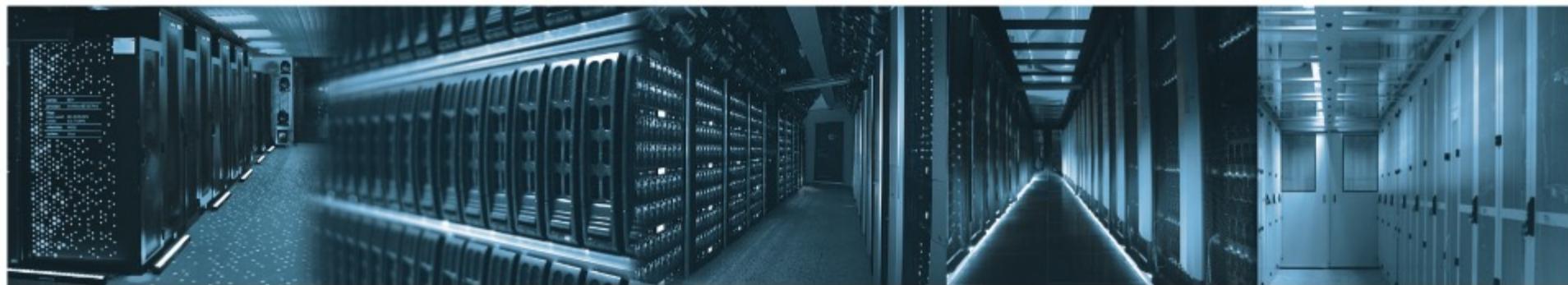
➤ Supercomputers from Poland

- 113 – Athena (ACC Cyfronet AGH) (**PLGrid**) (#17 on Green500)
- 158 – Altair (PSNC) (**PRACE-LAB**)
- 323 – Ares (ACC Cyfronet AGH) (**PLGrid**)



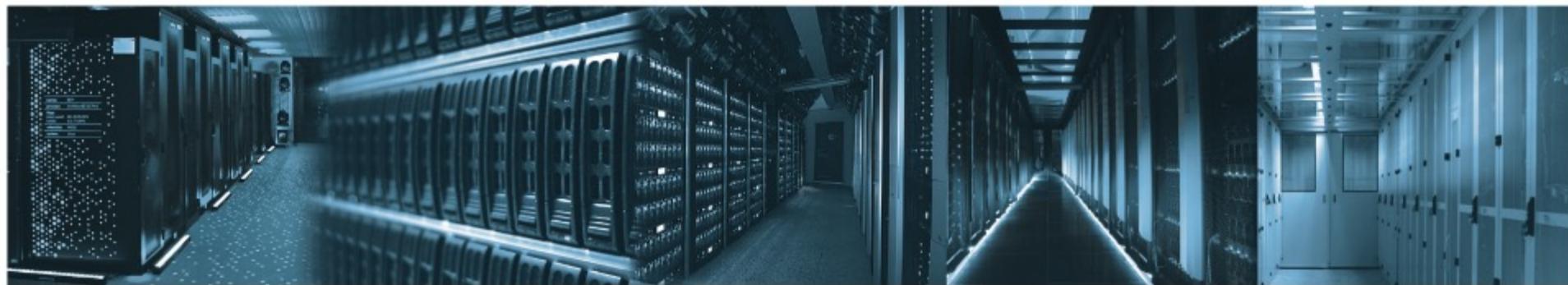
➤ Supercomputers from Poland

- 123 – Athena (ACC Cyfronet AGH) (**PLGrid**) (#19 on Green500)
- 186 – Altair (PSNC) (**PRACE-LAB**)
- 362 – Ares (ACC Cyfronet AGH) (**PLGrid**)



➤ Supercomputers from Poland

- 155 – Athena (ACC Cyfronet AGH) (**PLGrid**) (#22 on Green500)
- 221 – Altair (PSNC) (**PRACE-LAB**)
- 291 – Helios CPU partition (ACC Cyfronet AGH) (**PLGrid**)
- 404 – Ares (ACC Cyfronet AGH) (**PLGrid**)





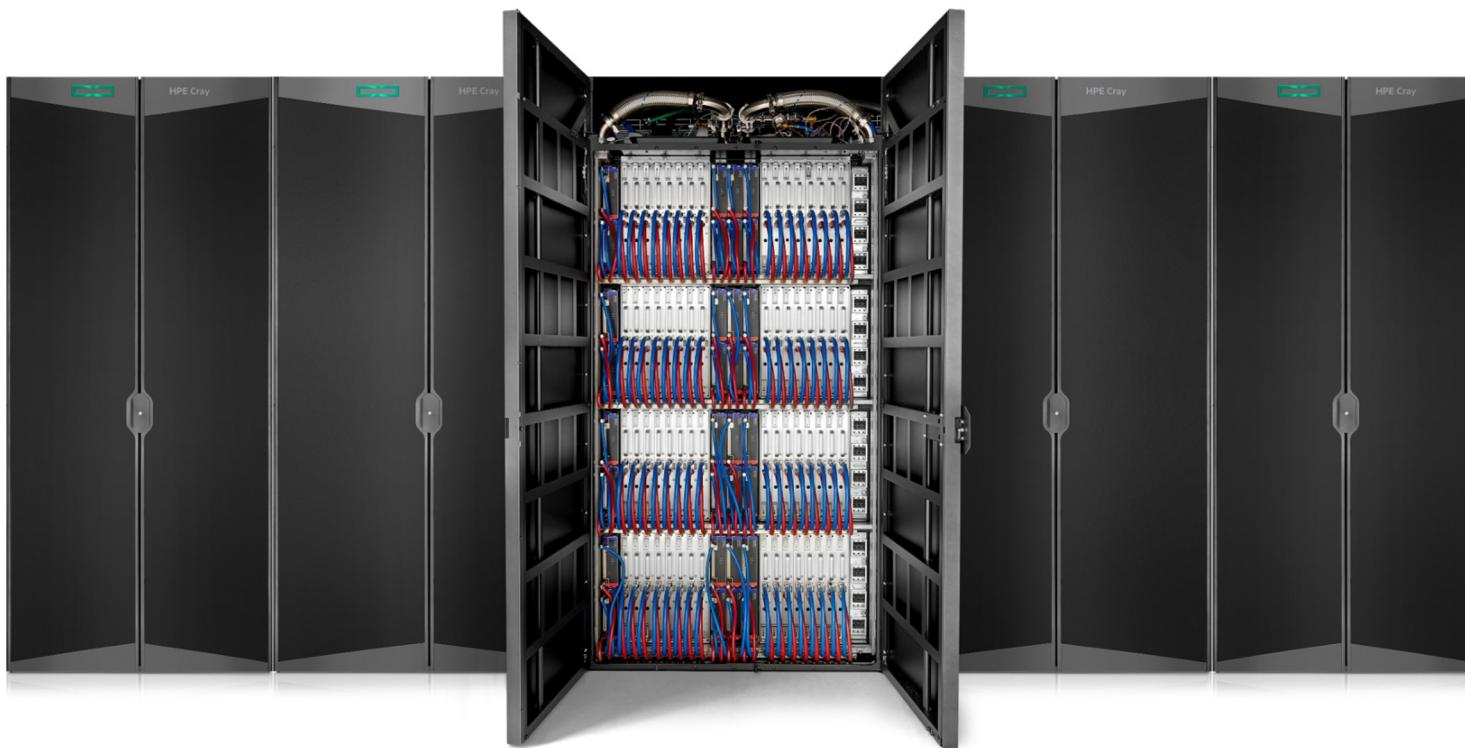
- 2006 - Baribal, 384 GFLOPS
 - large SMP
- 2010 - Zeus, 374 TFLOPS
 - commodity cluster with multiple partitions
- 2015 - Prometheus, 2,4 PFLOPS
 - fastest system in Poland's history
- 2021 - Ares, 4,0 PFLOPS
 - innovative liquid cooling and heat re-use
- 2022 - Athena, 7,7 PFLOPS
 - fastest system in Poland for HPC and AI
- 2023 - Helios
 - ?





- **60 Intel servers**
 - 2x Intel Xeon 8352s, 1 TB RAM, 2x 100 GbE
- **4 large memory nodes**
 - 2x Intel Xeon 8352s, 1 TB RAM + 8 TB Intel Optane, 2x 100 GbE
- **12 disk nodes**
 - 2x Intel Xeon 8352s, 512 GB RAM + 1 TB Intel Optane, 100 TB NVMe
 - DAOS
- **OS/Middleware:**
 - Rocky Linux 9, Openstack, Openshift
- **Main usage:**
 - interactive work (pre- and postprocessing)
 - data analytics
 - front-end for HPC/AI services





Announcing HPE Supercomputing Solution for Generative AI with NVIDIA Quad GH200



Hewlett Packard
Enterprise



NVIDIA





EuroHPC PL



- CPU partition (75264 cores, 196 TB RAM)
 - 272 standard nodes
 - 2x AMD 9654, 384 GB DDR5, 1x Slingshot 200 Gb/s
 - 120 nodes with double memory
 - 2x AMD 9654, 768 GB DDR5, 1x Slingshot 200 Gb/s
- GPU partition (440 accelerator modules)
 - 110 nodes
 - 4x NVIDIA Grace Hopper GH200 CPU+GPU, 4x Slingshot 200 Gb/s
- INT partition (24 accelerators)
 - 6 nodes
 - 2x AMD 9654, 1536 GB DDR5, 4x NVIDIA H100 HGX 94 GB, 30 TB NVMe, 2x Slingshot 200 Gb/s





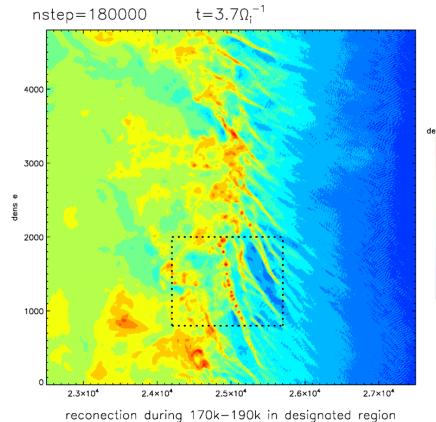
- HPE Cray EX4000

- full direct liquid cooling
 - >99% energy to liquid
 - DLC for servers, switches, power supplies
 - up to 400 kW per rack
 - 3,5 tons per rack
- Slingshot network
 - 200 Gb/s per port
 - low latency, RDMA, adaptive routing
 - compatible with Ethernet
- platform used to build world's largest systems
 - Frontier, El Capitan, Aurora, LUMI, ALPS, Shaheen III, Setonix



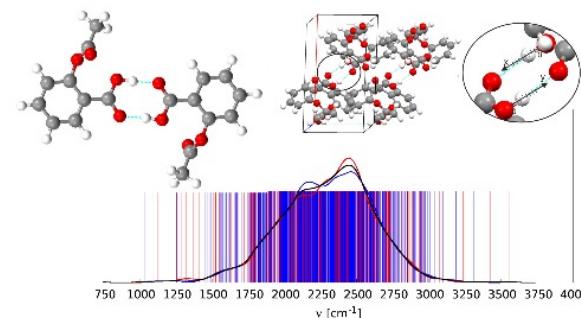
➤ Astrophysics: users' own code

- Particle in Cell written in Fortran
- Production runs - 9600 cores
 - One run up to 460 800 h CPU time – 53 years



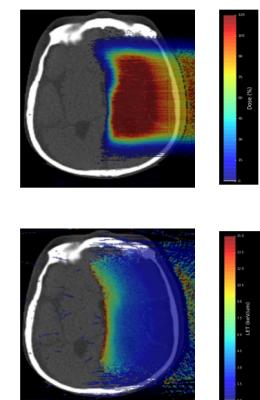
➤ Chemistry: CPMD, CP2k, Jaguar, Gaussian

- Importance of hydrogen bonds in biomolecules
- Jobs: 24-240 cores
 - Hundreds of thousands of jobs with walltime < 1 h
 - Efficient usage resources through backfill



➤ Biophysics: Proton therapy

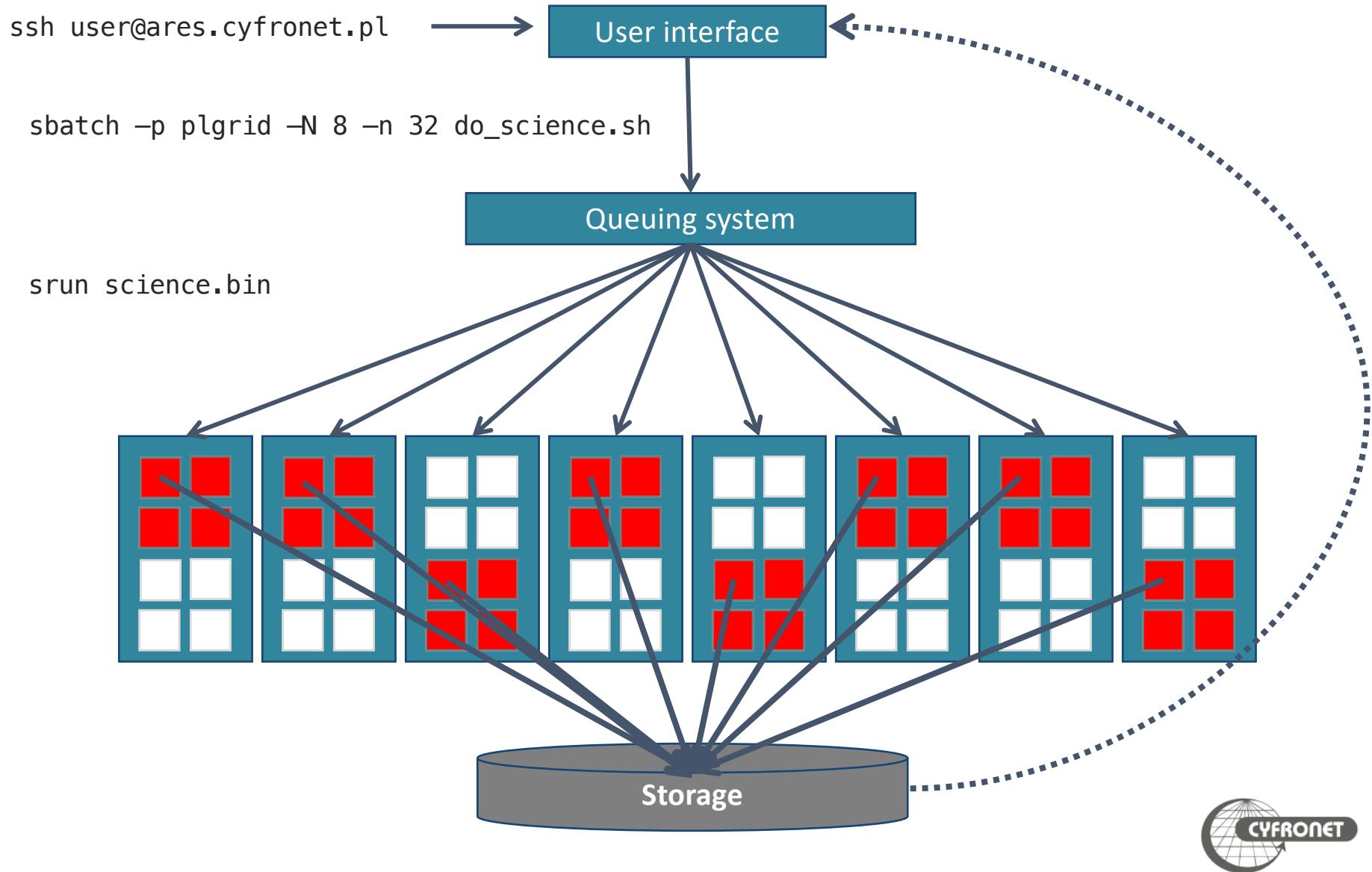
- Monte Carlo simulations of a proton beam
- Monte-Carlo based treatment planning
- Jobs:
 - thousands of jobs with MC simulations (hours on hundreds of nodes)
 - Interactive large data processing with Jupyter notebooks



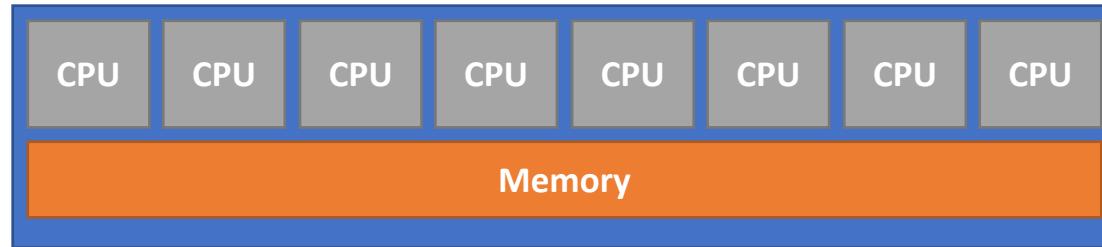
Steps necessary to grant access to Cyfronet/PLGrid resources

- Create account at PLGrid Users' Portal – <https://portal.plgrid.pl>
- Create (Scientific) Affiliation
- Create/Join Team
- Create Computational Grant for the team
- Apply for necessary services/entry points

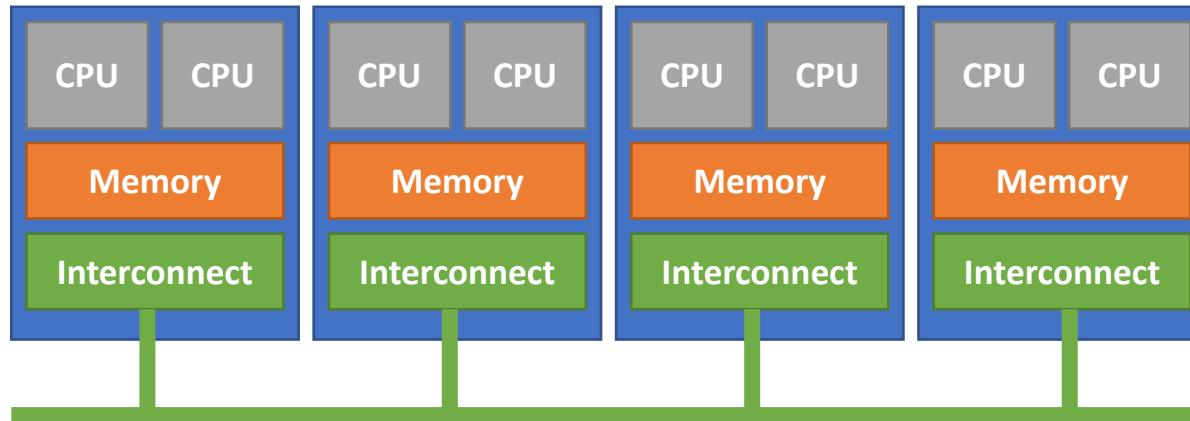
- High Performance Computing (HPC) – using supercomputers to solve problems that cannot be addressed by regular computes
 - use vast amount of processors/cores simultaneously
 - use huge memory allocations
 - use specialized computation accelerators (GPUs, FPGA, ASIC)
 - very fast access to data on dedicated high performance storage systems
- High Throughput Computing (HTC) – processing as much jobs (individual job could be quite simple) as possible using HPC infrastructure
- High Performance Data Analysis (HPDA) - using HPC infrastructure to analyse vast amount of data



SMP



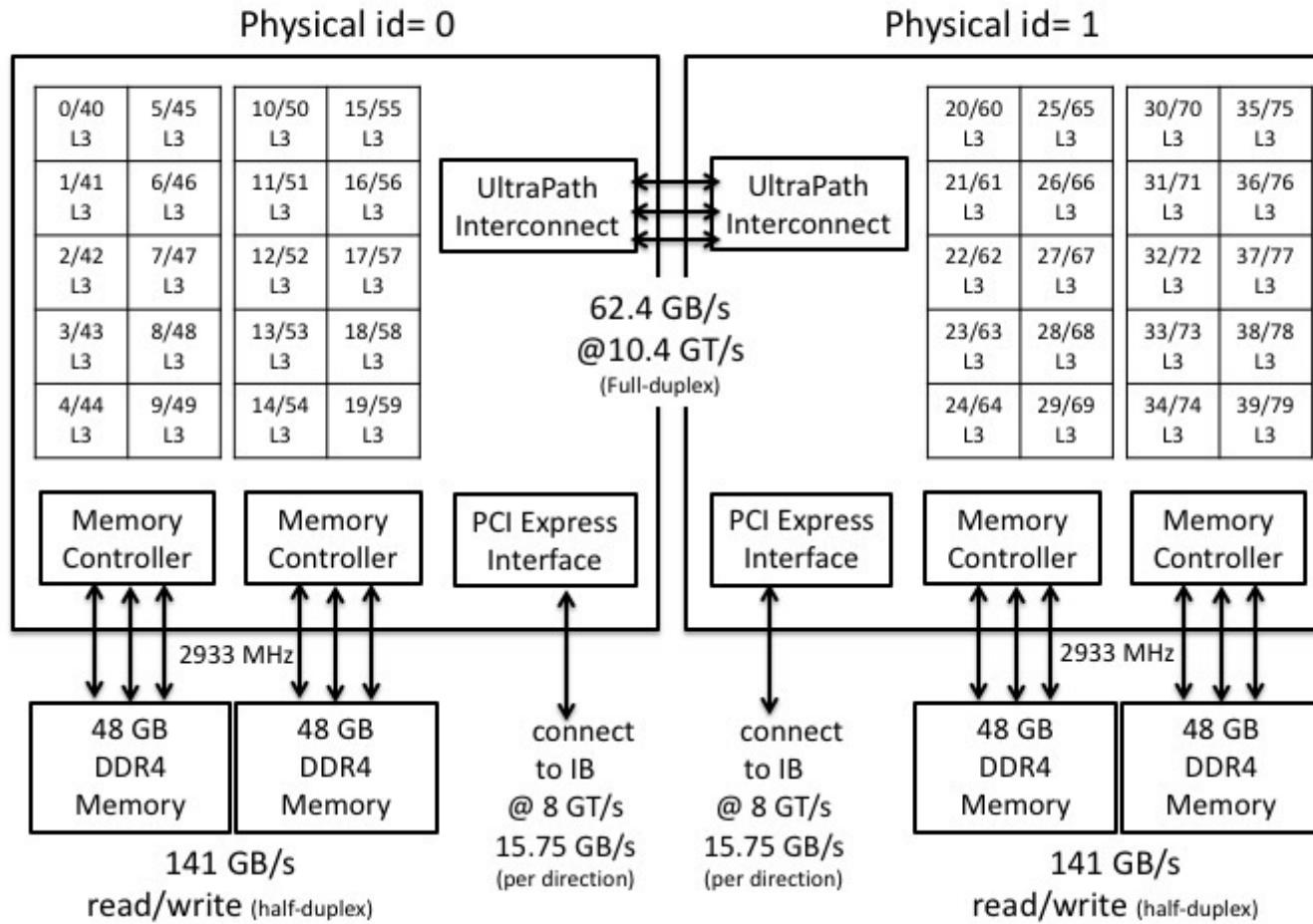
Cluster



- Ares consists user interface nodes (UI), service nodes and worker nodes
 - 788 CPU nodes (2x Intel Xeon Platinum 8268 processors, 48 x 2.9 GHz)
 - 532 nodes with 192 GB RAM (4GB/core)
 - 256 nodes with 384 GB RAM (8GB/core)
 - 9 ML/AI nodes (2 x Intel Xeon Gold 6242, 32 x 2.8 GHz, 384 GB, 8 x NVIDIA V100 SXM2 32GB HBM2)

Property	Ares	Ares GPU
CPU frequency	2.9 GHz	2.8 GHz
RAM	192/384 GB	384 GB
cores per node	48	32
InfiniBand interconnect	available, EDR 100 Gb/s	

Configuration of a Cascade Lake - SP Node



Source: https://www.nas.nasa.gov/hecc/support/kb/cascade-lake-processors_579.html

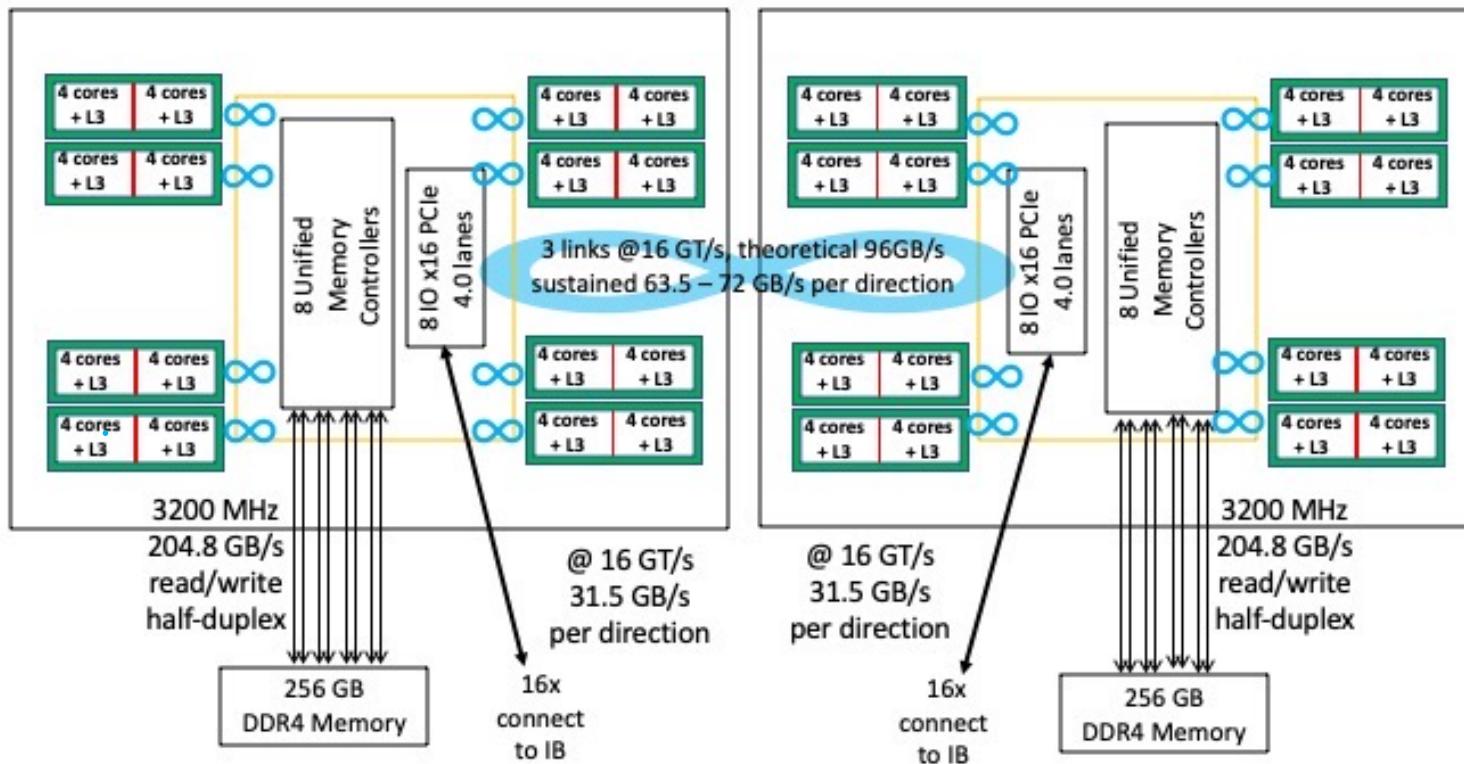
- Athena consists user interface nodes (UI), service nodes and worker nodes
 - 48 GPGPU nodes (2 x AMD EPYC 7742, 64 x 3.3 GHz, 1 TB, 8 x NVIDIA A100 SXM2 40GB HBM2)

Property	Athena
CPU frequency	3.3 GHz
RAM	1024 GB
cores per node	128
InfiniBand interconnect	available, EDR 800 Gb/s

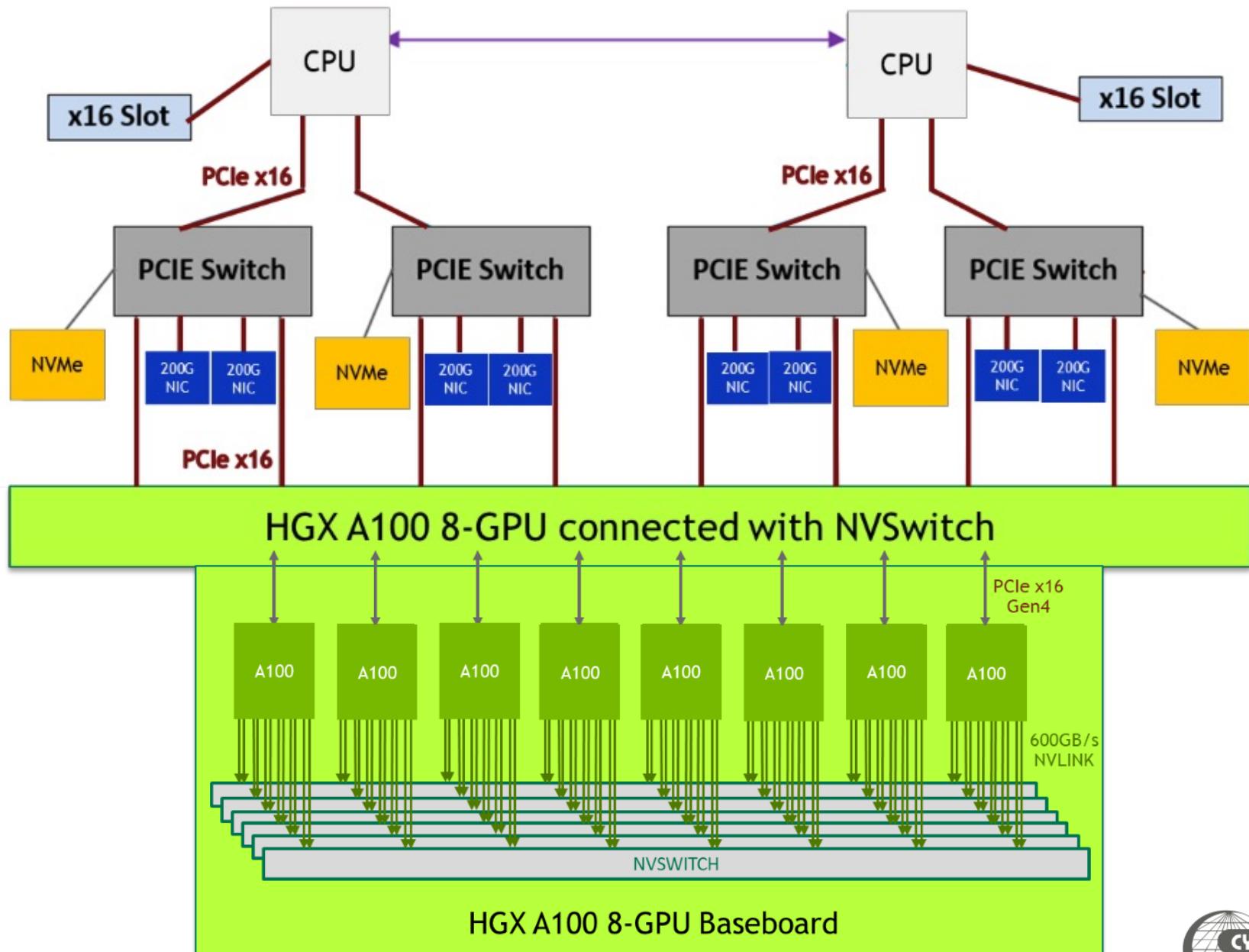
Configuration of an AMD Rome Node

Physical id= 0

Physical id= 1



Source: https://www.nas.nasa.gov/hecc/support/kb/amd-rome-processors_658.html



- All PLGrid HPC clusters use Linux as OS
 - Rocky Linux on Ares (RL 8) and Athena (RL 9)
- HPC clusters contain
 - user interface (UI) node(s)
 - computing nodes (a.k.a worker nodes)
 - service nodes (internal use only)
- User interface **must not be used** for computing
- Fair share between users' tasks and computations provided by queuing system
 - SLURM on Ares, Athena & Prometheus



- User logs on user interface (UI) node using SSH protocol
 - UI names:
 - login@ares.cyfronet.pl
 - login@athena.cyfronet.pl
 - SSH clients
 - on Linux and MacOS included in OS
 - ssh command in terminal
 - on Windows
 - PuTTY - <http://www.chiark.greenend.org.uk/~sgtatham/putty/>
 - MobaXterm - <http://mobaxterm.mobatek.net>
 - Windows Terminal - <https://aka.ms/terminal>

- User logs on user interface (UI) node using SSH protocol
 - copying files and directories
 - on Linux and MacOS included in OS
 - `scp` command in terminal
 - `rsync` command in terminal
 - on Windows
 - WinSCP - <http://winscp.net/>
 - Windows Terminal - <https://aka.ms/terminal>
 - Windows Subsystem for Linux
 - `scp` command in terminal
 - `rsync` command in terminal
 - remote work
 - using CLI with `nano`, `vim`, `emacs`, `mc`
 - Visual Studio Code - <https://code.visualstudio.com/>
 - Remote – SSH extension
 - Remote – Tunnels
 - SSHFS via FUSE

- Storage of data
 - \$HOME – user's home directory
 - quite slow, should not be used for heavy I/O calculations
 - quota 10 GB
 - \$PLG_GROUPS_ST0RAGE
 - additional storage gained through PLGrid grants system
 - distributed Lustre file system (lower performance than \$SCRATCH)
- Temporary scratch file systems
 - \$SCRATCH – distributed scratch Lustre file system
 - accessible from all nodes of cluster (including UI)
 - \$TMPDIR and \$SCRATCHDIR – unique subdirectories on \$SCRATCH created for the job at its start
- To check quota use hpc-fs

- Scientific software usually needs specific runtime environment (i.e. additional libraries) and sometimes technical knowledge is needed to install them efficiently
- Modules and Lmod packages are solutions for loading runtime environments on every cluster in PLGrid infrastructure
- Advantages
 - simplicity of preparing software to run efficiently
 - computation scripts could be transferable between HPC clusters
 - possibility of concurrent runs of different versions of software
 - on hybrid HPC systems transparent switching to most efficient version of software
- Drawbacks
 - additional command to remember .-)

- Load environment for scientific package
 - `module add <module-name>` (i.e. `module add GCC`)
 - `module load <module-name>` (i.e. `module load GCC`)
- Remove module
 - `module rm <module-name>` (i.e. `module rm GCC`)
 - `module unload <module-name>` (i.e. `module unload GCC`)
- Listing of all available/installed modules
 - `module avail`
 - `module avail GCC` (all available GCC versions)
 - `module spider python` (all installed Python modules)
 - `module spider "/r/"` (all available R versions, regex p search)
- Listing of loaded modules
 - `module list`

- Clearing all loaded modules
 - `module purge`
- Saving collection of modules for later use, restoring it and listing saved collections
 - `module save [collection]`
 - `module restore [collection]`
 - `module savelist`
 - `module describe [collection]`
- `ml` is shorthand for `module` command
 - `ml = module list`
 - `ml <module-name> = module load <module-name>`
 - `ml --<module-name> = module unload <module-name>`
 - `ml av <string> = module avail <string>`
- Getting help
 - `module help`
 - `ml -h`

- Create directory for modules, eg. \$PLG_GROUPS_ST0RAGE/your-team-name/modules
- Create there subdirs for modules, eg. app/name
- Create Lua file for lmod, eg. 1.0.lua

```
local pkgName      = myModuleName()
local fullVersion = myModuleVersion()
whatis("Name: "..pkgName)
whatis("Version "..fullVersion)
whatis("Description: Abaqus")
local APPDIR = '/net/software/local/abaqus/2017'
depends_on('plgrid/tools/intel/18.0.0')
prepend_path('PATH', APPDIR .. '/bin')
prepend_path('LD_LIBRARY_PATH', APPDIR .. '/lib')
```

- Set path for new module: module use \$PLG_GROUPS_ST0RAGE/your-team-name/modules
- Load module with: module load app/name/1.0

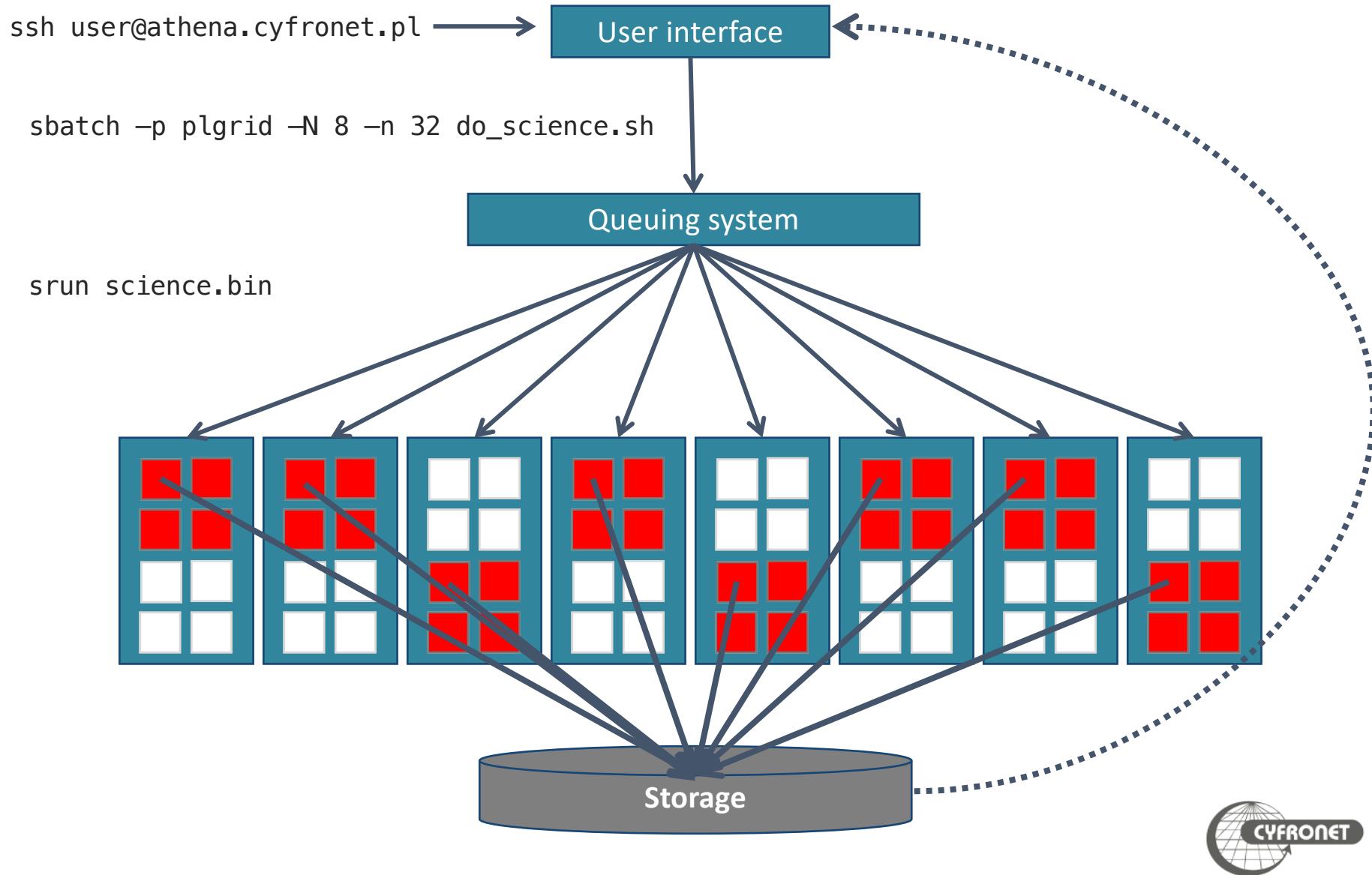


Academic Computer Centre CYFRONET AGH



Klemens Noga, Maciej Czuchry, Oskar Klimas

Efficient usage of HPC systems in scientific computations
Scheduling computations with the SLURM scheduler



- User interact with SLURM queuing system using commands
 - `sbatch` – to submit new job to queue
 - `squeue` – gives information about jobs running in queuing system
 - `scancel` – deletes jobs from queue
 - `sinfo/scontrol` – gives detailed information about queue, job or node
 - `srun` – runs interactive job or step in batch job
- Each job has got **unique job identifier** (jobID)

- Queuing system
 - manages all computational task on cluster
 - monitors available resources
 - acts as matchmaker between needs of jobs and resources
 - empowers fair share between different users
- All computational tasks are run as **jobs** queued in **queues** and run according to their priority and available resources.
- Priority of job depends on
 - amount of resources obtained by user in computational grant
 - amount of resources requested by job
 - **maximum wall time of computation** is most essential resource
 - amount of other resources concurrently used by job's owner

- HPC clusters available in PLGrid use several kinds of queuing systems
 - SLURM (<http://slurm.schedmd.com>)
 - PBS Pro (<http://pbspro.org>)

HPC Centre	Cluster	Queuing system
ACC Cyfronet AGH	Prometheus	SLURM
	Ares	SLURM
	Athena	SLURM
PSNC	Eagle/Altair	SLURM
TASK	Tryton	SLURM
WCSS	Bem	PBS Pro
	Bem2	SLURM

- Command `sbatch` submits new job in queue
- All parameters describing job's requirements could be included in batch script and given to queuing system using command
 - `sbatch [options] script.slurm`
- Example script

```
#!/bin/env bash

# Commands that will be run after start of the job
echo "Computation started on work node: ";
hostname

module add Python/3.10.4

python python-first.py
```

- Commands `squeue` and `hpc-jobs` give view of jobs scheduled in queuing system
- Jobs States
 - PD – queued
 - R – running
 - CF – configuring (resources for job are being prepared)
- Additional helpful flags
 - `squeue --user $USER` – information about \$USER's jobs
 - `hpc-jobs -j <jobID>` – information about specified jobs
 - `hpc-jobs -N` – additional information about information about exec nodes
 - `hpc-jobs -q/-r` – information about queued (pending)/running jobs only
 - `hpc-jobs -h` – help screen
- In addition `scontrol`, `sinfo` and `smap` give information about status of cluster
 - `scontrol show job <jobID>` – information about <jobID> job
 - `scontrol show node <nodes_list>` – information about nodes

Partitions	max time	Information
plgrid-testing	1:00:00	for test runs (small number of jobs)
plgrid	3-00:00:00	
plgrid-now	12:00:00	interactive runs, max one job on one node
plgrid-long	7-00:00:00	*
plgrid-gpu	3-00:00:00	nodes with GPGPU*
plgrid-gpu-v100	2-00:00:00	nodes with V100 GPGPU*
plgrid-gpu-a100	2-00:00:00	nodes with A100 GPGPU*
plgrid-bigmem	3-00:00:00	big mem nodes*
plgrid-services	14-0:00:00	running “long-living” services*

- In SLURM queues are called partitions
 - `scontrol show partitions <partition_name>` – detailed information about partition
 - `sinfo` – lists all available nodes in all partitions
 - `sinfo -p <partition_name>` – lists information only about partition
 - default time in all `plgrid*` partitions is set to 15 minutes
- * - partitions available after request through PLGrid grants

```
#!/bin/env bash

# Commands that will be run after start of the job
echo "Computation started on work node: "; hostname

module add python

./python-script.py > python.log
```

- SLURM options provide information about job requirements to queuing system. They could be
 - given in command line `sbatch [SLURM options]`
 - included in first lines of batch script with `#SBATCH` at start of line

- sbatch command uses various options to provide queuing system with additional info about the job
 - `-p <partition>, --partition=<partition>` defines partition
 - `-J <jobname>, --job-name=<jobname>` give name to job
 - `-a, --array=<indexes>` submit a job array
 - `--mail-user=<user's e-mail>` setting email for notifications
 - `--mail-type=<type>` information when notifications should be send: at beginning (BEGIN), end (END) or execution error (FAIL)
 - `-A <grantID>, --account= <grantID>` information about computational grant (if omitted job use default)
- When option `-p` is omitted job is queued into default partition (on Ares plgrid)

- There are several resources available for job
 - `-t, --time=<time>` total maximal execution wall time of job
 - `-N, --nodes=<nodes>` number of nodes allocated to job
 - `-n, --ntasks=<ntasks>` number of tasks invoked in whole job
 - `--ntasks-per-node=<ntasks>` number of tasks invoked on each node
 - `--cpus-per-task=<cores>` number of cores per each task (i.e., when using threads in OpenMP)
 - `--mem=<MB>` amount of memory per node requested by job
 - `--mem-per-cpu=<MB>` amount of memory per core requested by job
- Parameter formats
 - time format: "min", "min:sec", "hours:min:sec", "days-hours", "days-hours:min" and "days-hours:min:sec"
 - memory: MB (=1024kB), GB (=1,024MB)

```
#SBATCH --job-name=serial.job
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=10:00
#SBATCH --mem=2GB
#SBATCH --partition=tutorial
#SBATCH --account=tutorial

module add intel

icc -xHost hello.c -o hello.x

./hello.x
```

- In SLURM job is sent to partition
 - flag `-p <partition_name>` or `--partition <partition_name>`
 - partitions for PLGrid users: `plgrid*`

```
#SBATCH --job-name=parallel-srun
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=1GB
#SBATCH --partition=tutorial
#SBATCH --account=tutorial

module add intel

icc -xHost hello.c -o hello.x

srun ./hello.x
```

- srun inside batch job executes command `./hello.x` on allocated resources according to requested `--ntask` or `--nodes*--ntasks-per-node` flags
 - variable `SLURM_NTASKS` holds information about number of tasks to be run
- each srun could request more than one core
 - `srun --nodes=x --ntasks=y --cpus-per-task=z ...`

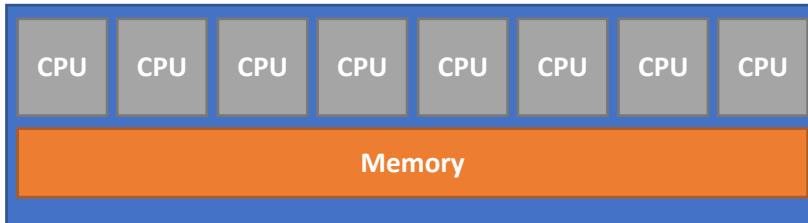
➤ OpenMP

- API for writing shared-memory software
- Shared memory in threads
- Requires support in compiler (-fopenmp, -fopenmp)

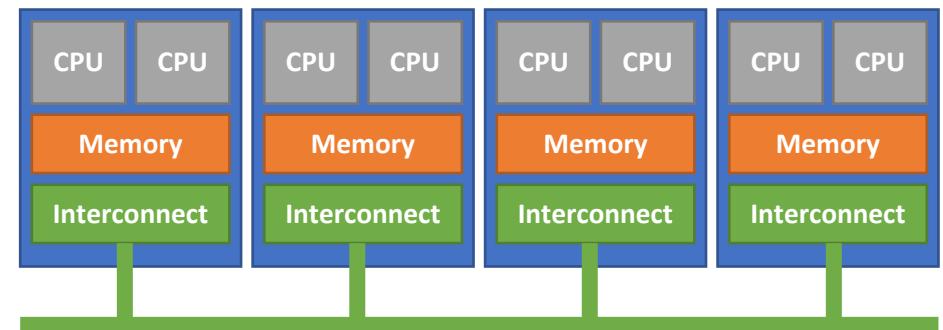
➤ MPI

- Message Passing Interface (send, receive, broadcast)
- Every process has its own isolated memory space
- Can use more than one machine via interconnect (eth, infiniband)

SMP



Cluster



- Threads: Depends on compiler which was used to build software
 - OMP_PROC_BIND=true
 - OMP_PLACES= {threads | cores | sockets}
 - Intel: KMP_AFFINITY
 - compact – nearest to free context (processor)
 - verbose – extend verbosity
 - OMP_DISPLAY_AFFINITY=TRUE
- Processes (MPI): Depends of spawner (mpiexec, mpiexec.hydra, srun)
 - --bind-to {core | numa | none}
 - OpenMPI – works fine
 - If combined with OpenMP use
OMPI_MCA_rmaps_base_mapping_policy=node:PE=\$SLURM_CPUS_PER_TASK
 - IntelMPI – use environmental variables
 - I_MPI_PIN_DOMAIN = {core | socket | numa | node | cache | omp | auto}
 - I_MPI_PIN_ORDER = {scatter | compact | spread | bunch}
- Node topology
 - lscpu -e -a
 - numactl --hardware

```
#SBATCH --job-name=parallel-openmp
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=12
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=2GB
#SBATCH --partition=tutorial
#SBATCH --account=tutorial

module add intel

icc -xHost -fopenmp hello.c -o hello.x

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

./hello.x
```

- When use OpenMP
 - use `--cpus-per-task=<cores_per_job>` and `--nodes=1` for request of resources
 - variable `SLURM_CPUS_PER_TASK` holds information about number CPUs allocated to each task

```
#SBATCH --job-name=distributed-mpi
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=12
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=1GB
#SBATCH --partition=tutorial
#SBATCH --account=tutorial

module add iimpi

mpiicc -xHost hello.c -o hello.x

srun ./hello.x
```

- When software is parallelized using MPI
 - use `--ntasks-per-node=<cores_per_node>` and `--nodes=<no_of_nodes>` for request of resources
 - variable `SLURM_NTASKS` holds information about number of tasks to be run

```
#SBATCH --job-name=mpi-openmp
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=6
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=2GB
#SBATCH --partition=tutorial
#SBATCH --account=tutorial

module add iimpi

mpiicc -xHost -qopenmp hello.c -o hello.x

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

srun ./hello.x
```

- When hybrid MPI/OpenMP
 - use `--cpus-per-task=<cores_per_job>` and `$SLURM_CPUS_PER_TASK` for distribution of threads
 - use `--ntasks-per-node=<cores_per_node>` for request of MPI processes
 -

- SLURM adds environmental variables which could ease performing computation

Variable	Description
SLURM_JOB_ID	job identifier (jobID)
SLURM_SUBMIT_DIR	dir, from which batch script was submitted to queuing system
SLURM_NTASKS	total number of tasks (i.e. MPI processes) in the current job
SLURM_NTASKS_PER_NODE	number of tasks to be run on one node
SLURM_NODELIST	list of nodes allocated to the job
SLURM_CPUS_PER_TASK	number of cores requested per task
SCRATCH	\$USER's root scratch directory on distributed Lustre file system
SCRATCHDIR	unique directory for the job on \$SCRATCH

- Environment variables can be used to control distribution of job
 - MPI jobs: SLURM_NTASKS to run MPI processes (using `srun`) variable
 - OpenMP jobs: SLURM_CPUS_PER_TASK to run proper number of threads
 - hybrid MPI/OpenMP jobs: combine SLURM_NTASKS to run MPI processes and SLURM_CPUS_PER_TASK to expand threads

- Interactive work on cluster should be done using interactive jobs through `srun` command
 - `srun -p plgrid -A <grant_id> -n 1 --pty /bin/bash`
- User interface **must not be used** for computing
- High priority queue `plgrid-now` for interactive work
 - one job on one node up to 12:00:00
- To attach terminal to running batch job
- `srun -N1 -n1 --jobid=<jobID> --pty /bin/bash`
- `srun -N1 -n1 --jobid=<jobID> -w <nodeID> --pty /bin/bash`
- `sattach <jobid.stepid>`
- Helper script `ssh_slurm`
 - `ssh_slurm <jobid> <dest_host> [command]`

- **scancel** command is used to delete unwanted jobs from queuing system
 - **scancel <JobID>**
- Information about jobs which cannot be deleted using **scancel** should be sent to system administrators through
 - Helpdesk PLGrid PL
 - <https://helpdesk.plgrid.pl>
 - helpdesk@plgrid.pl

- `hpc-jobs` and `hpc-jobs-history` could be used to monitor efficiency of jobs
 - memory usage
 - CPU usage
- `hpc-jobs` – running and queued jobs
- `hpc-jobs-history` – historical data of completed jobs
- `hpc-jobs*` usage
 - `hpc-jobs -N` – additional information about nodes of job(s)
 - `hpc-jobs -v` – more detailed information about job(s)
 - `hpc-jobs -j <jobID>` – information only about job(s)
 - `hpc-jobs -h` – help screen
 - `hpc-jobs-history -d <period>` jobs completed in last <period> days

- Always compile on **computing node** using
 - batch job with `sbatch`
 - interactive job with `srun --pty bash`
 - use modules (also for libraries)
 - Intel® MKL Link Line Advisor: <https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/>
 - when software could be used by various users consider global compilation by admins
 - use IntelMPI (`iimpi`) and OpenMPI (`gmpi`) modules build by admins
 - check threads and processes bindings (i.e. `KMP_AFFINITY` environment variable)
- Common compilation flags on Cascadelake CPUs
 - GCC: `-march=native`, `-fopenmp`, `-mfma`, `-ffast-math`, `-funroll-loops`, `-flto`
 - Intel: `-xCORE-AVX512` (or `-xHost`), `-qopenmp`, `-fma`, `-ipo`
- Aggressive compilation flags on Cascadelake CPUs
 - GCC: `-march=native -Ofast -mfma -funroll-loops`
 - Intel: `-fast`
- When in trouble contact admins trough PLGrid Helpdesk
 - <https://helpdesk.plgrid.pl/>
 - `helpdesk@plgrid.pl`

- Obtained through PLGrid Portal - <https://grants.plgrid.pl/>
 - distinct allocations for CPU, GPGPU and storage with suffixes (-cpu, -gpu etc.)
- Information about grants from CLI
 - hpc-grants
- Cost of job
 - CPU only – in CPU-hours

```
cost_cpu      = job_cpus_used * job_duration
cost_memory   = ceil(job_memory_used/memory_per_cpu) * job_duration
final_cost    = max(cost_cpu, cost_memory)
```
 - GPU – in GPU-hours

```
cost_gpu      = job_gpus_used * job_duration
cost_cpu      = ceil(job_cpus_used/cpus_per_gpu) * job_duration
cost_memory   = ceil(job_memory_used/memory_per_gpu) * job_duration
final_cost    = max(cost_gpu, cost_cpu, cost_memory)
```

- Multiple jobs can be executed with identical parameters within one `sbatch` run as array jobs when `-a`, `--array=<indexes>` option used
 - `sbatch -a n-m,k,l script.slurm` (np. `sbatch -a 0-9` or `sbatch -a 2,4,7`)
- All jobs within array have same value of `SLURM_SUBMIT_DIR` and `SLURM_ARRAY_JOB_ID` variables, but have additional unique identifier `SLURM_ARRAY_TASK_ID` (number of job in array)

```
#!/bin/env bash
#SBATCH -a 0-4,9
#SBATCH --time=5:00
OUTPUTDIR=$SLURM_SUBMIT_DIR/$SLURM_ARRAY_JOB_ID
mkdir -p $OUTPUTDIR
cd $TMPDIR
hostname > task.$SLURM_ARRAY_TASK_ID

mv task.$SLURM_ARRAY_TASK_ID $OUTPUTDIR
```

- `squeue -a` – shows all jobs in array queued in system

- Dependencies between jobs can be added through `--dependency=<dependency_list>` option
- Possible dependencies
 - `after:job_id[:jobid...]` – job can begin execution after the specified jobs have begun execution
 - `afterany:job_id[:jobid...]` – job can begin execution after the specified jobs have terminated
 - `afternotok:job_id[:jobid...]` – job can begin execution after the specified jobs have terminated in some failed state
 - `afterok:job_id[:jobid...]` – job can begin execution after the specified jobs have successfully executed
 - `expand:job_id` – resources allocated to this job should be used to expand the specified job
 - `singletont` – job can begin execution after any previously launched jobs sharing the same job name and user have terminated

- SLURM job batch script is always started in directory from which it was submitted to queuing system. Access to that directory is also possible with `SLURM_SUBMIT_DIR`
- All batch jobs have got file in which data from standard outputs (both standard output stream `stdout` and standard error stream `stderr`) is stored named `slurm-<JobID>.out`
 - those file should not be big (less than several MBs) and are stored in `SLURM_SUBMIT_DIR`
 - `-o, --output=<file>` and `-e, --error=<file>` - options to redirect `stdout` and `stderr`
- When commands in SLURM script print big amount of data into output streams user should redirect that data to file(s)
 - for standard output stream (`stdout`): command `> file.out`
 - for standard error stream (`stderr`): command `2> file.err`
 - for both streams to one file: command `&> file.log`
- **\$HOME and \$PLG_GROUPS_STORAGE must not be used** for heavy I/O computations

- During batch job submission user should always
 - specify maximal time of job execution (parameter `t/time`)
 - specify maximal RAM amount needed by job through `mem` (or `mem-per-cpu`)
 - enable checkpoints
 - for parallel computations use all cores on nodes when possible
 - when big amount of data is used in computation always use `$SCRATCH` for files
 - when big amount of data is going to be passed to standard output streams redirect it to files and use `$SCRATCH`
 - load runtime environment of software via `module` command in batch script
 - do not load software modules in scripts loaded at user's login (i.e., `bashrc`)

- Before start of big computations do scalability tests
 - Start from one quarter of node or one NUMA-node and scale up
 - Check computations efficiency with hpc-jobs/hpc-jobs-history and performance tools available internally in your software
- From SLURM scheduler point of view
 - If the job can scale with reasonable efficiency up to a certain point where the job takes 1 day to execute, this is the best option.
 - As a general guideline, keeping job runtime from 1 hour to 3 days maximum is best. Too short jobs might result in significant scheduling overhead.

➤ MEMFS

- -C memfs
- \$MEMFS
- use memory as filesystem (1TB max on Athena)
 - Accessible only within node
- available during JOB and **lost after it finishes**

➤ LOCALFS

- -C localfs
- \$SCRATCH_LOCAL
- use file as filesystem (512GB/1TB per node)
- Each node has its own file! (not a shared filesystem)
 - Accessible only within node
- Available during JOB and **lost after it is finished**

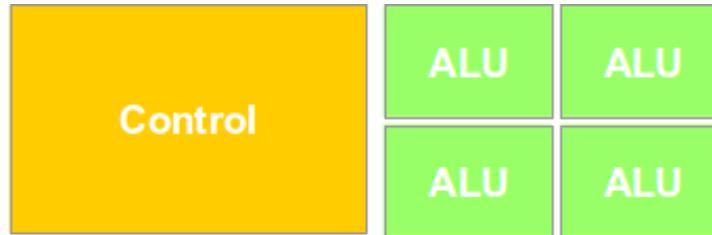


Academic Computer Centre CYFRONET AGH

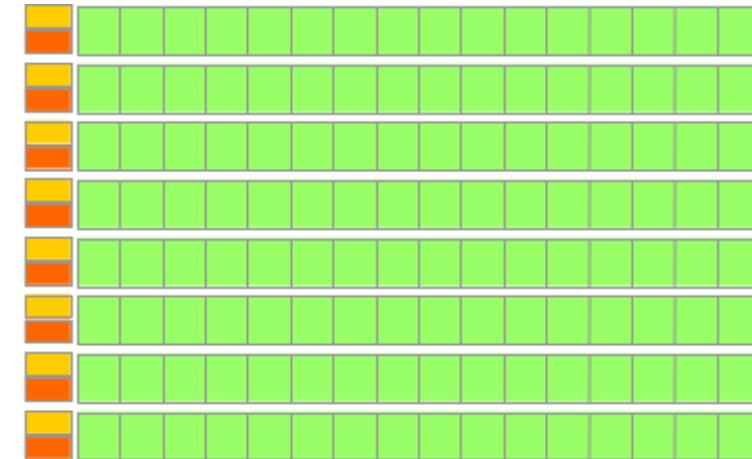


Klemens Noga, Maciej Czuchry, Oskar Klimas

Efficient usage of HPC systems in scientific
computations
GPU computations



CPU



DRAM

GPU

David B. Kirk and Wen-mei W. Hwu, Programming Massively Parallel Processors

- CPU

- Universal
- Complex logic, big managing part
- Few concurrent threads
- **Big/multi-level cache memory** hides latency of access to main memory

- GPU

- Efficient in case of highly parallel workloads
- Minimal managing part, main part of silicone – ALU and registers
- vast amounts of concurrent threads
- **Multithreading** hides latency of access to main memory

- GPGPUs are shown in SLURM queuing system as generic resources (GRES) with gpu identifiers.
- To check where GPGPUs are available
 - `sinfo -o '%P || %N || %G'`
- To request GPGPUs for a job `--gres=gpu[:count]` or `--gpus=[count]` has to be added to `sbatch/srun` command
 - `srun -p tutorial -N 2 --ntasks-per-node=24 -n 48 -A <grant_id> --gres=gpu[:count] --pty /bin/bash -l`
 - `#SBATCH --gres=gpu[:count]`
 - `#SBATCH --gpus=[count]`
- GPGPUs are available only in `plgrid-gpu-v100` and `plgrid-gpu-a100` partitions.
- GPGPU available for the job are listed in the `$CUDA_VISIBLE_DEVICES` variable
- Usage of GPGPU could be checked using `nvidia-smi`

```
#SBATCH --job-name=serial.job
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --gpus=1
#SBATCH --time=10:00
#SBATCH --mem=24000
#SBATCH --partition=tutorial
#SBATCH --account=tutorial

module add cuda

nvcc hello.cu -o hello.x

./hello.x
```

- Pro-viz is a new service for users of Prometheus that allows running GUI mode of software using: TurboVNC <https://www.turbovnc.org> or web browser
- To run TurboVNC, you have to install Java JRE x86.
- In first step user need to run pro-viz on the cluster. To use it you need to load software module of pro-viz:

```
module load pro-viz
```

```
pro-viz ...
    start [-n CORES | -N NODES | -p PARTITION | -t TIME | -A
ACCOUNT | -r RESERVATION | -g GPUS | -C constraints | -m EMAIL-ADDRESS]
] - start a new batch session
    interactive [-p PARTITION | -t TIME | -A ACCOUNT | -r
RESERVATION | -g GPUS | -C constraints ] - start a new interactive
session
    list - list all sessions
    attach JOBID - attach session to a working job with JOBID
    password JOBID - generate access token for session JOBID
    stop JOBID - terminate session JOBID
    killall - terminate all sessions
    help - duh
```

*To start GUI session on cluster Ares with
12 cores on one node working node use commands:*

- module load pro-viz
- pro-viz start -N 1 -n 24 -p plgrid -A tutorial -t 03:00:00
- pro-viz password JOBID



Academic Computer Centre CYFRONET AGH



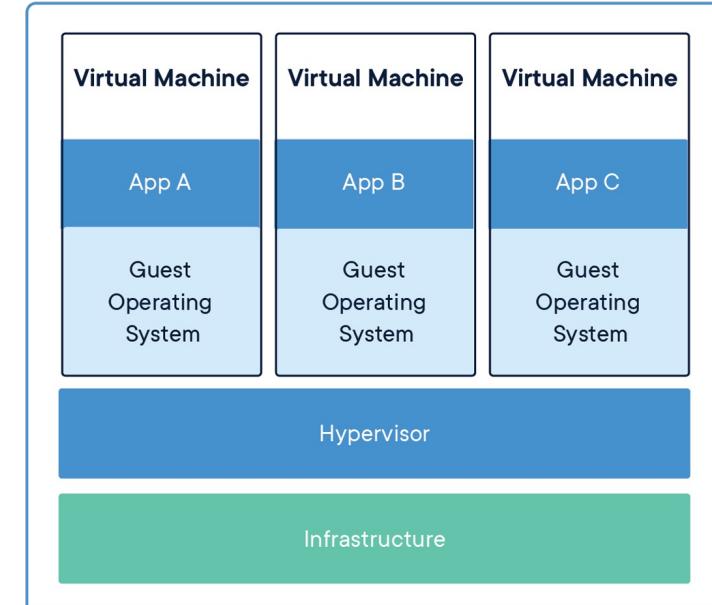
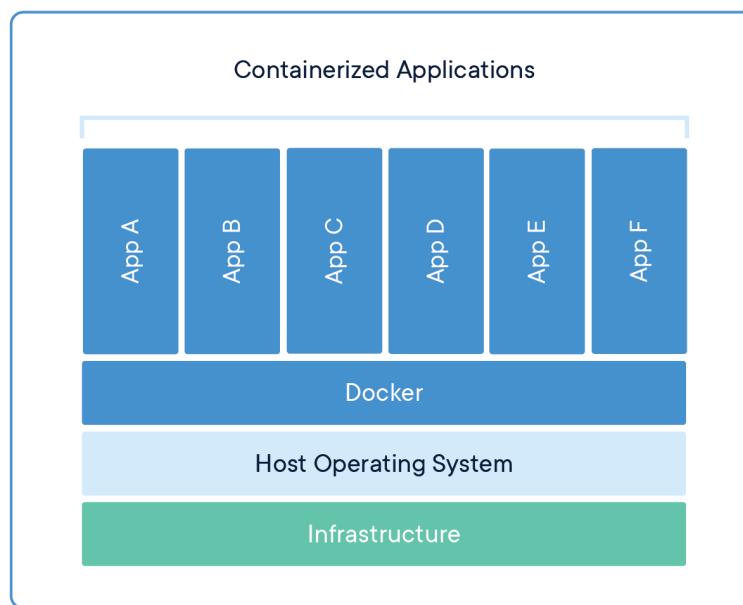
Klemens Noga, Maciej Czuchry, Oskar Klimas

Efficient usage of HPC systems in scientific computations

Use of software containers on HPC

What is a container?

- A container is similar in purpose to a virtual machine (VM), providing encapsulation for a software application and its runtime environment.
- Implementation differs. Containers use the host Linux kernel instead of virtualizing hardware, so they are lightweight compared to VMs.
- Linux Containers rely on kernel features, and are inherently Linux based



Why use containers on HPC?

- Containers give Portability, Reproducibility, Scalability
 - Build once, use by many on many systems
 - Could move software stack to other HPC system
 - Isolate from changes in the HPC software environment
 - Save simulation/analysis software runtime for reproducible science

Singularity/Aptainer

- Usage
 - `singularity exec <container-name.sif> command`
 - `apptainer exec <container-name.sif> command`
 - `apptainer shell <container-name.sif>`
 - `apptainer run <container-name.sif>`
- Remarks
 - Possibility pull Docker/Singularity container
`apptainer pull docker:<repository>/<name-of-container>`
 - Usage of GPU though `-nv` flag
 - Attach HPC system folders using `-B` flag, i.e:

`apptainer exec -B /net biopython_latest.sif python3`

`apptainer exec -B $SCRATCH:/tmp/scratch biopython_latest.sif`
- Usage restrictions
 - Creation of container requires root permissions, therefore cannot be done on HPC resources – use PLGrid cloud/your own resources instead

Singularity/Apptainer

- Additional Remarks
 - Use environmental variables to speed up converting containers
 - APPTAINER_TMPDIR=\$SCRATCHDIR or \$MEMFS
 - APPTAINER_CACHEDIR=\$SCRATCHDIR or \$MEMFS
 - Use \$MEMFS filesystem to speed up I/O operations
 - Set working dir using --pwd option
 - Store data or datasets on \$PLG_GROUPS_STORAGE/<your-team> or \$SCRATCH
 - To use MPI you have to have compatible MPI flavours in the container and on the HPC system
 - Usually best solution is to have the PMI library in SLURM and start the container using srun

```
srun --mpi=pmi2 singularity exec ...
```





Academic Computer Centre CYFRONET AGH



Klemens Noga, Maciej Czuchry, Oskar Klimas

Efficient usage of HPC systems in scientific computations

Use of Python on HPC

- In Python you can easily manage your python modules via pip (pip3)
 - pip list
 - pip search numpy
 - pip install numpy
 - pip install -U numpy
 - pip install -u numpy==1.10.1
- The problem is when you need more than one environment
- Solution is virtualenv
 - virtualenv --system-site-packages my_new_env
 - source my_new_env/bin/activate
 - pip install -U whatever you want
 - deactivate
- One directory for each environment – clean project management

python

- Versions available at Ares cluster
 - GNU:
 - 2.7.x, 3.7.x, 3.8.x, 3.9.x, 3.10.x, 3.11.x
 - python
 - python-bare (minimal installation)
 - Additional libraries in modules `scipy-bundle`, `ipython`, ect.
- Usage
 - `module add python/<version>`
- Remarks
 - Build with MKL numerical libraries, support for GPGPU computing
- Usage restrictions
 - only SMP mode (up to 48 computing cores@Ares)
 - multi-node MPI only with libs such as `py4mpi`, `dask` i.e.

- On HPC clusters, computations are executed on worker nodes
 - usually no GUI
 - usually behind firewall
- Jupyter notebook/Jupyterlab needs GUI in a web browser
- Therefore, there is a need to port tunnelling
 - First, submit job which creates Jupyter notebook and tunnel from worker node to login node
 - Establish tunnel from your computer to login node of cluster
 - Open notebook in your favourite browser on your computer

- SLURM job create
 - Jupyter notebook
 - tunnel from worker node to login node

```
#!/bin/bash
#SBATCH --partition tutorial

## get tunneling info
XDG_RUNTIME_DIR=""
ipnport=$(shuf -i8000-9999 -n1)
ipnip=$(hostname -i)
user=$USER

module load jupyterlab

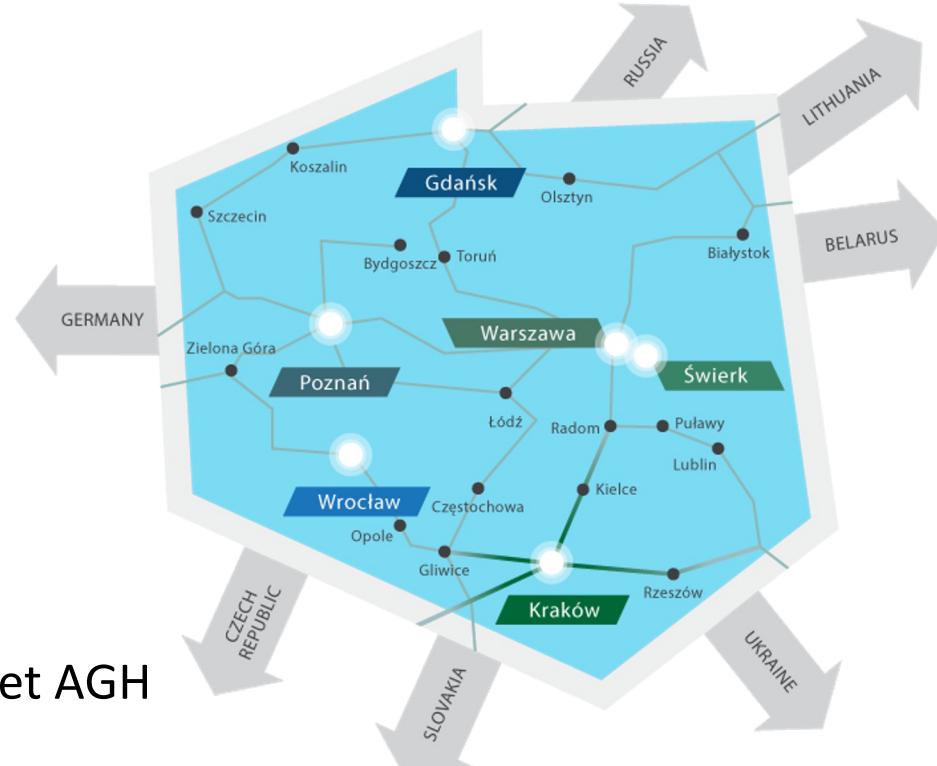
## start a cluster instance and launch Jupyter server
jupyter-notebook --no-browser --port=$ipnport --ip=$ipnip python-notebook.slurm
```

- User has to create a second tunnel from user's computer to login node of HPC resource (i.e. ares.cyfronet.pl)
 - `ssh -N -L <local-port>:<worker-node-ip>:<remote-port>`
`username@ares.cyfronet.pl`
 - info about tunnel details `<local-port>`, `<worker-node-ip>`,
`<remote-port>` are in log file of SLURM job
 - `username` – user's login
- After establishing both tunnels, Jupyter notebook is ready to start
 - open webpage `localhost:<local-port>` in browser on your local computer
 - remember about token, which is listed in a log file of SLURM job

PLGrid Infrastructure



- Projects:
 - PL-Grid
 - PLGrid Plus
 - PLGrid NG
 - PLGrid Core



- PLGrid Consortium
 - Coordinator: ACC Cyfronet AGH
 - Partners:
 - Poznan Supercomputing and Networking Center, Poznań
 - Interdisciplinary Centre for Mathematical and Computational Modelling, Warszawa
 - Wroclaw Centre for Networking and Supercomputing, Wrocław
 - Tricity Academic Computer Centre, Gdańsk
 - National Centre for Nuclear Research, Świerk

<http://www.plgrid.pl/en/>



- The PLGrid Infrastructure is available free of charge for Polish researchers and all those engaged in scientific activities in Poland
- On-line registration through PLGrid Users' Portal – <https://portal.plgrid.pl>
- User verification based on Polish Science Database – <https://www.nauka-polska.pl>



On PLGrid Users Portal user can

- apply for access to tools and services
- monitor utilization of resources
- manage their computational grants and grid certificates

Access to all PLGrid resources through **one account** and **one passphrase** (or grid certificate)



- The European High Performance Computing Joint Undertaking
 - 32 participating countries
 - the European Union (represented by the European Commission)
 - private partners

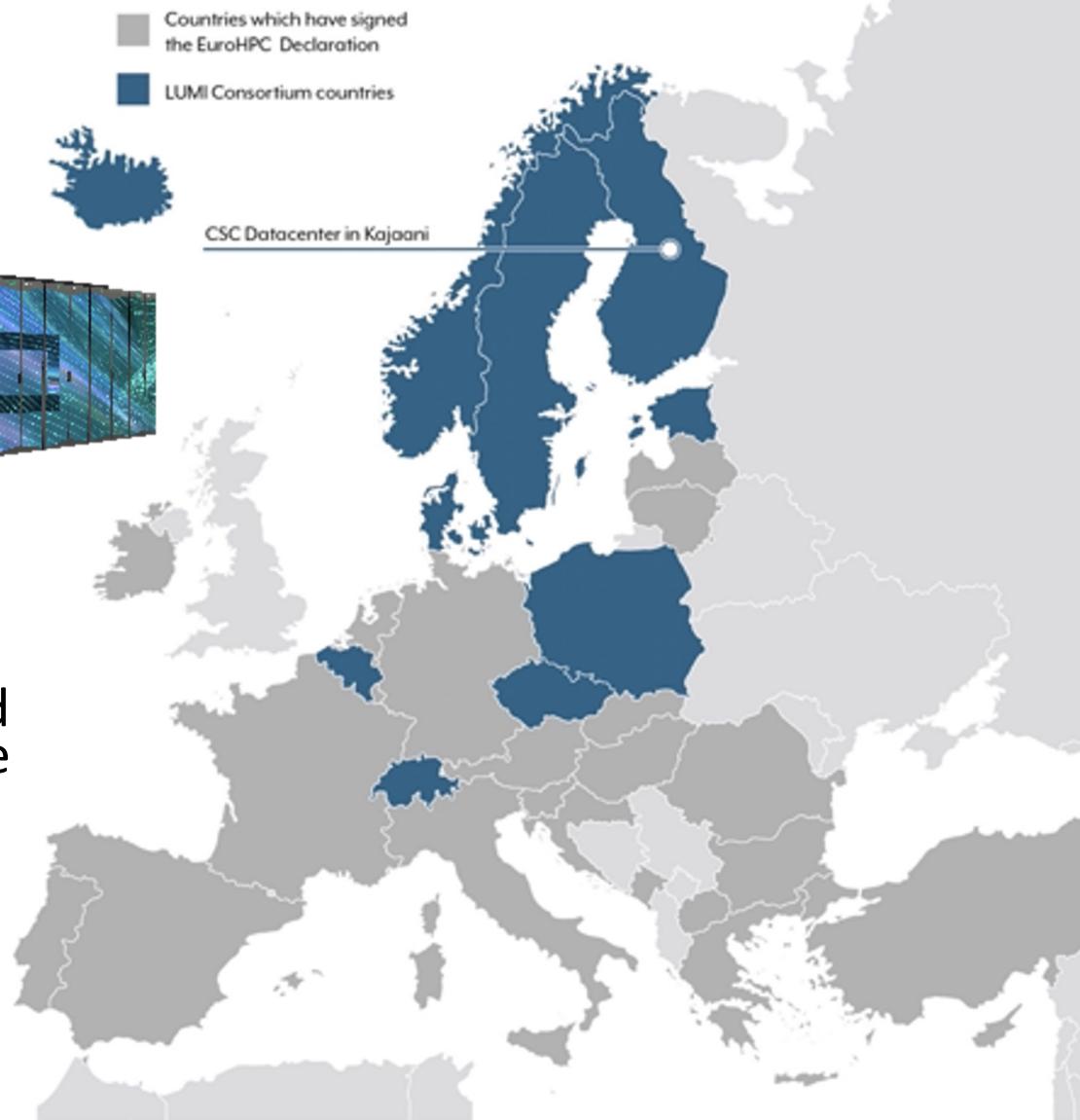
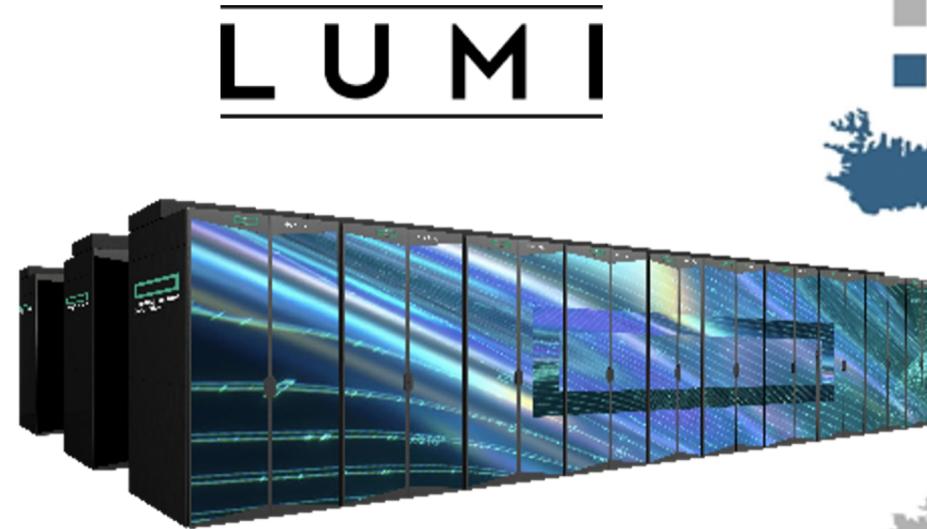


- Goals
 - deploy top-of-the-range supercomputing infrastructures across Europe to support European HPC users wherever they are in Europe
 - implement an ambitious research and innovation agenda to develop a competitive HPC ecosystem and supply chain in Europe, which includes hardware, software, applications but also training and skills



<https://eurohpc-ju.europa.eu/>





- LUMI will be an **HPE Cray EX** supercomputer manufactured by Hewlett Packard Enterprise
 - Peak performance over 550 petaflop/s makes the system one of the world's fastest
 - Available for users in
 - LUMI-C Q4 2021
 - LUMI-G Q1 2022

<https://www.lumi-supercomputer.eu/>

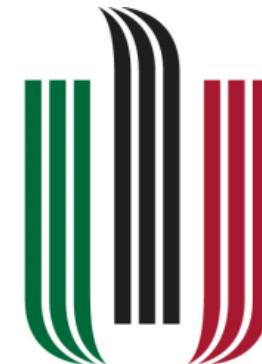


- National Competence Centres for EuroHPC
- Goals
 - Establishing network of national HPC competence centers in all EuroHPC member states
 - Focus on cooperation between all stakeholders in european HPC
 - Training of scientific staff and development of HPC software in both academia and industrial environments

www.eurocc-project.eu

cc.eurohpc.pl





A horizontal collage of scientific and technological icons. From left to right: a blue satellite dish against a dark blue background with stars; a blue DNA double helix against a light blue background with white dots; a blue microscope against a light blue background with white dots; a blue flask containing blue liquid next to a blue hexagonal molecule against a dark blue background; and a red rectangular panel with the Polish text "Umiesz liczyć?". Below the flask is a dark blue panel with the Polish text "LICZ U NAS!" in large white letters. To the right of the flask is a light blue panel with a circular icon containing a white atom symbol with two electrons and two protons.