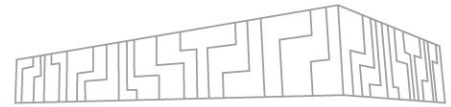
VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

BASICS OF WORKING WITH THE KAROLINA SUPERCOMPUTER

Ondřej Meca

based on materials prepared by Jakub Homola

OUTLINE



- Supercomputers and HPC in general
- How to connect to Karolina
- Technical details about Karolina
- Running computations, SLURM
- Software management
- Basics of parallel computing

- Everything applies to almost all clusters, Karolina is used mainly as an example

Total duration 2.5-3.5 hours
Depends on hands-on sessions

SUPERCOMPUTERS AND HPC

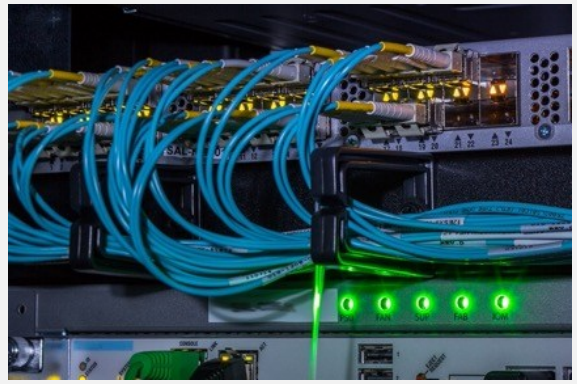
SUPERCOMPUTER

- Powerful compute servers
- Fast storage
- Connected using a high-speed network

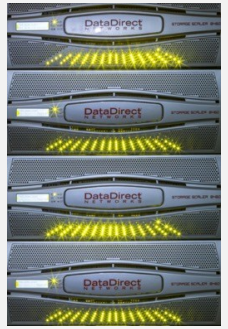
- + power, cooling, software, …

Compute nodes

Network

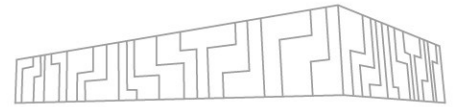


Storage



"Just" many computers connected together

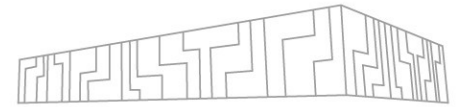
(SUPER) COMPUTER PERFORMANCE



- Flop/s – number of floating-point operations executed per second
- Flop – addition, multiplication, … of floating-point numbers
- Floating-point number
  - $25.167 = 0.25167 \times 10^{2}$
  - $10011010110101110110001 \times 2^{101101}$
- Large performance => need large numbers
- E.g. our Karolina has ≈15.7 Pflop/s
  - 15.7 Peta Flop/s
  - $= 15.7 \times 10^{15}$ Flop/s
  - = 15 700 000 000 000 000 Flop/s

Prefix		Base 10	Decimal
Name	Symbol		
quetta	Q	10^{30}	1 000 000 000 000 000 000 000 000 000 000
ronna	R	10^{27}	1 000 000 000 000 000 000 000 000 000
yotta	Y	10^{24}	1 000 000 000 000 000 000 000 000
zetta	Z	10^{21}	1 000 000 000 000 000 000 000
exa	E	10^{18}	1 000 000 000 000 000 000
peta	P	10^{15}	1 000 000 000 000 000
tera	T	10^{12}	1 000 000 000 000
giga	G	10^{9}	1 000 000 000
mega	M	10^{6}	1 000 000
kilo	k	10^{3}	1 000

(SUPER) COMPUTER PERFORMANCE



- Theoretical **R_peak**
- Computed from HW characteristics
  - 720 compute nodes
  - 2 processors per node
  - 64 cores per processor
  - 2 FMA units per core
  - 2 flops per FMA operation
  - 4 double-precision floating point numbers in AVX2 SIMD
  - 2.6 GHz base frequency
  - = 3.833 Pflop/s

- Measured **R_max**
- Run a benchmark and find out
- HPL – High Performance Linpack
  - Solution of a dense system of linear equations using gaussian elimination with pivoting

- **R_max < R_peak**
  - Memory access
  - Communication overhead (network)

- Barbora 849 Tflop/s
- Karolina 15.7 Pflop/s

TOP500

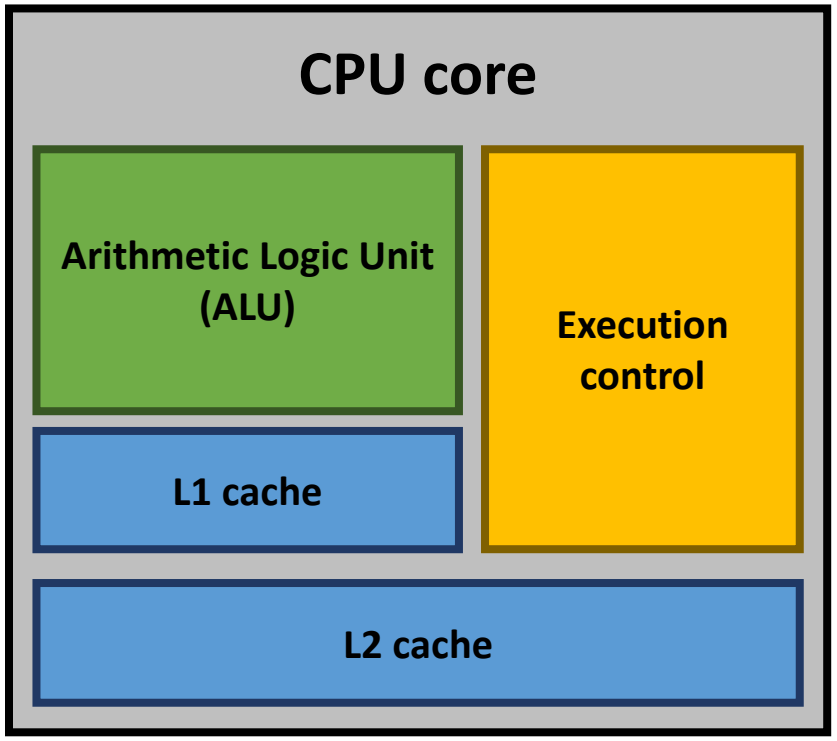
- List of 500 most powerful supercomputers
  - https://top500.org/

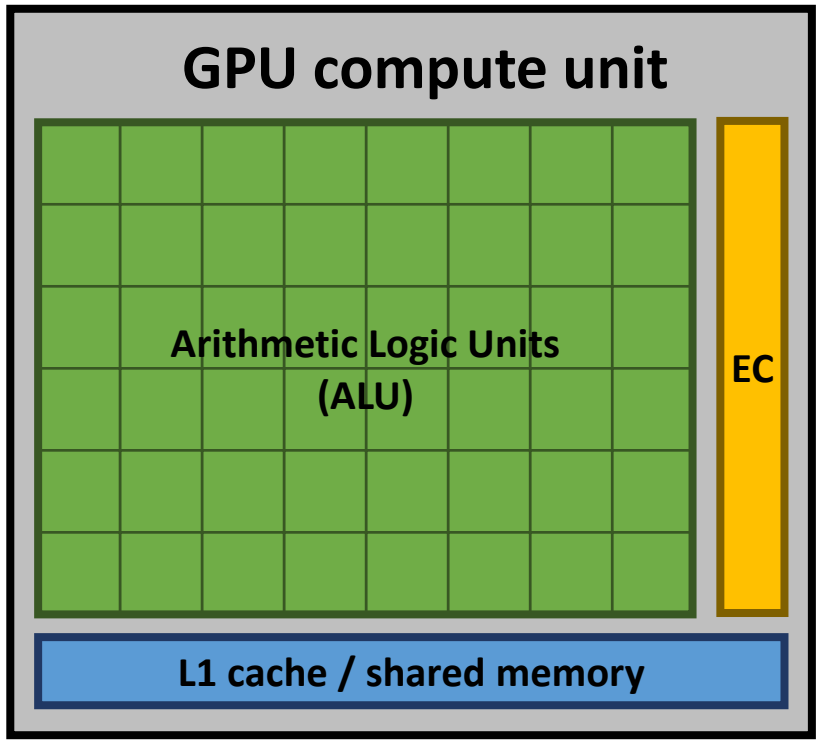
- **Compute performance** – Linpack
  - Solution of dense linear system using Gaussian elimination with pivoting
- Memory bandwidth – HPCG
  - Solution of sparse linear system using the Conjugate Gradient method
- Energy efficiency

- Barbora first appearance: #149
- Karolina first appearance: #69
- LUMI first appearance: #3

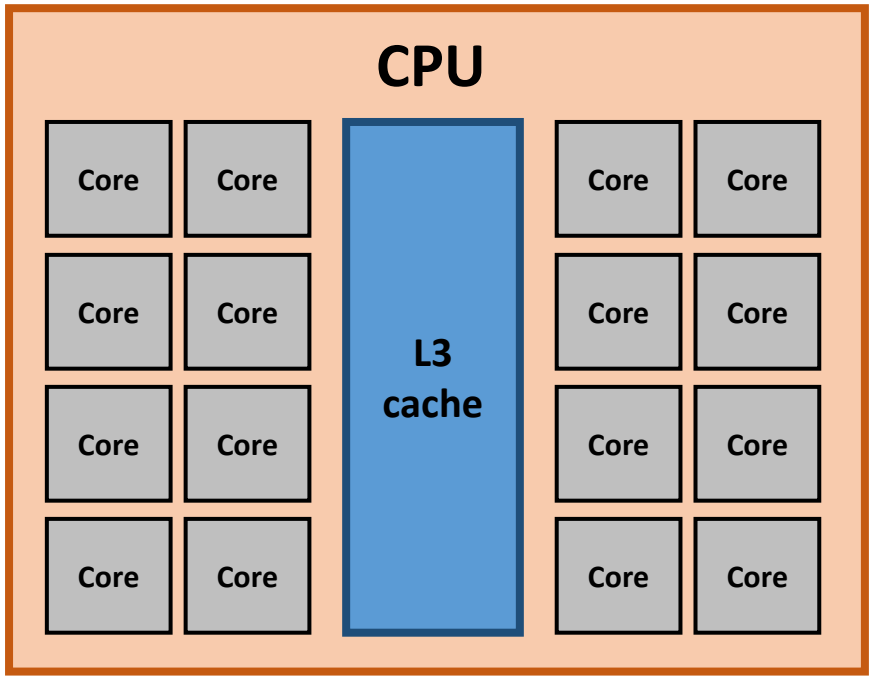
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,206.00	1,714.81	22,786
2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
3	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
4	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
5	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107
6	**Alps** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Swiss National Supercomputing Centre (CSCS) Switzerland	1,305,600	270.00	353.75	5,194
7	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, EVIDEN EuroHPC/CINECA Italy	1,824,768	241.20	306.31	7,494
8	**MareNostrum 5 ACC** - BullSequana XH3000, Xeon Platinum 8460Y+ 32C 2.3GHz, NVIDIA H100 64GB, Infiniband NDR, EVIDEN EuroHPC/BSC Spain	663,040	175.30	249.44	4,159
9	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096
10	**Eos NVIDIA DGX SuperPOD** - NVIDIA DGX H100, Xeon Platinum 8480C 56C 3.8GHz, NVIDIA H100, Infiniband NDR400, Nvidia NVIDIA Corporation United States	485,888	121.40	188.65	

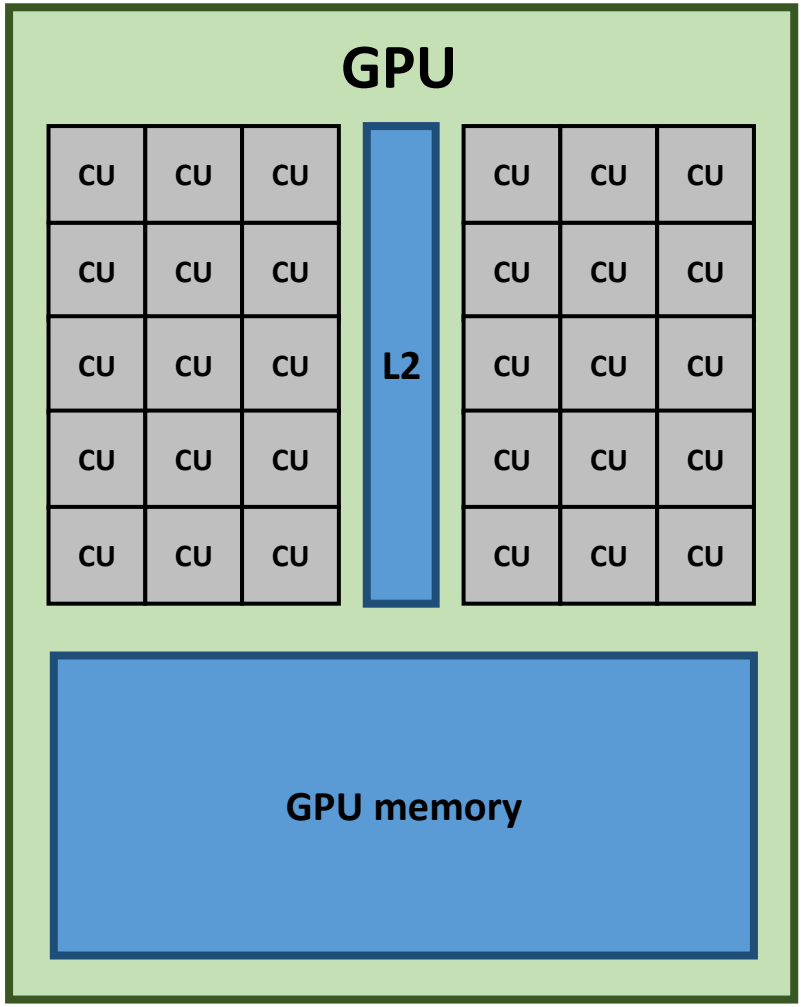
HPC BUILDING BLOCKS



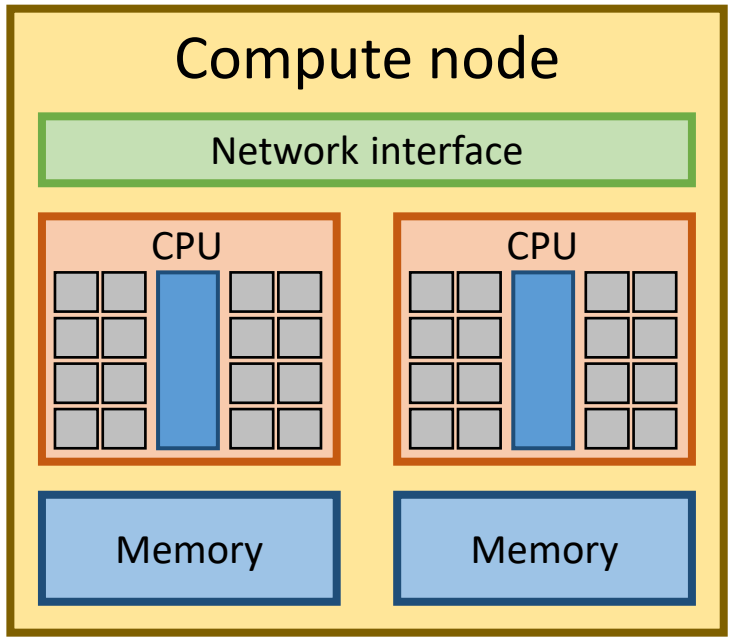


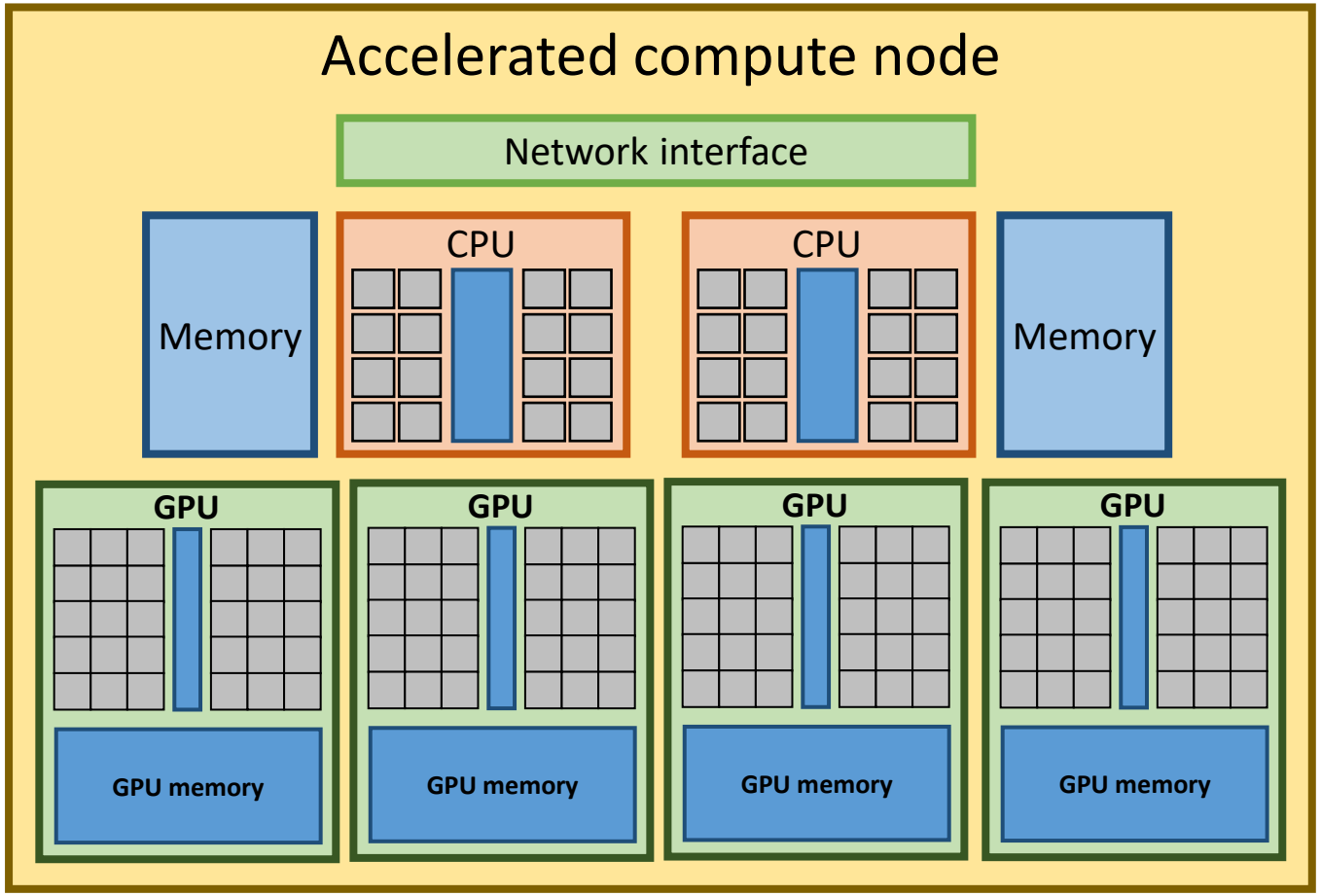
HPC BUILDING BLOCKS



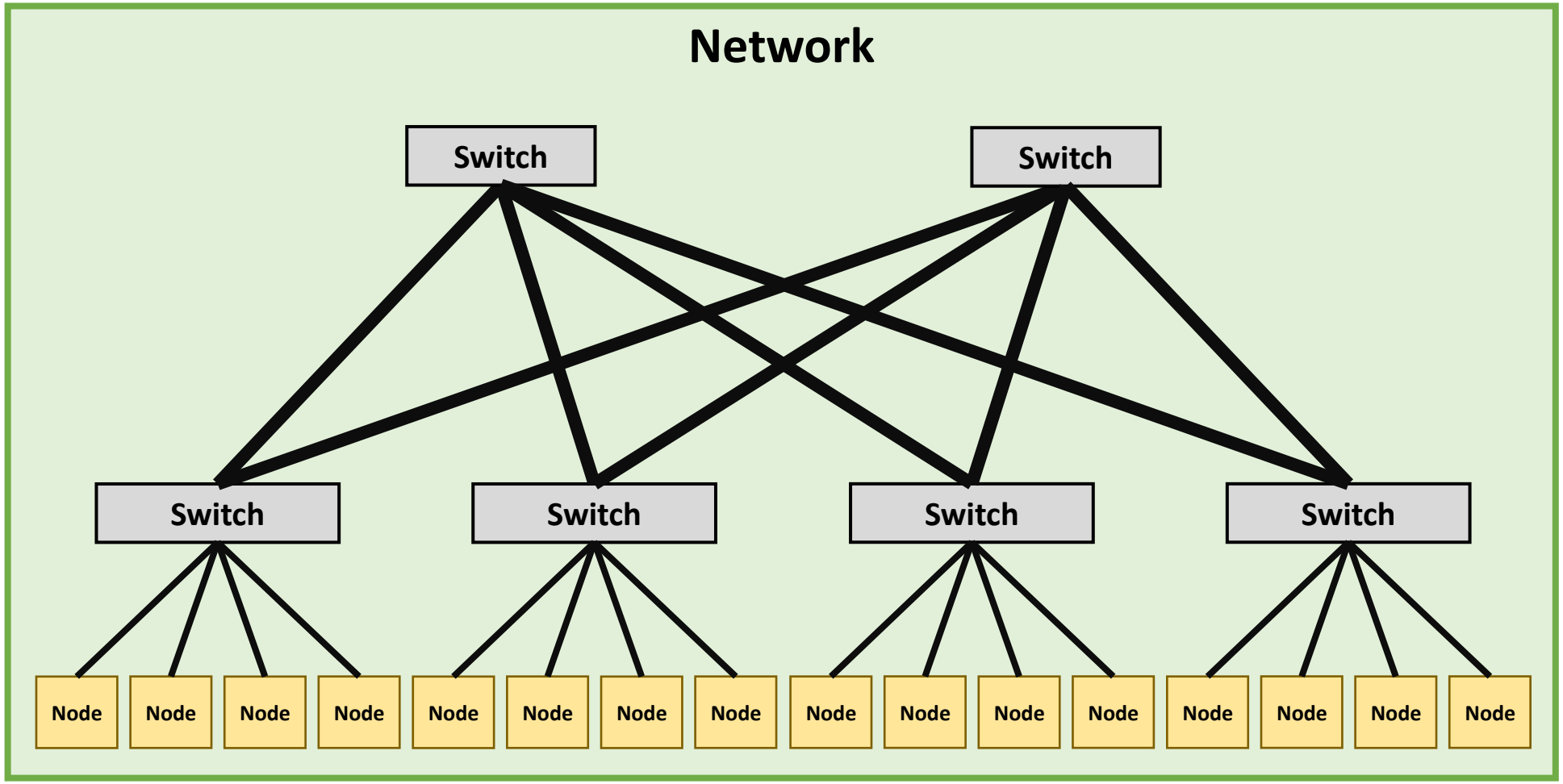


HPC BUILDING BLOCKS





HPC BUILDING BLOCKS



HOW TO ACCESS KAROLINA

Connecting and working with a cluster

HOW TO ACCESS KAROLINA

The Martian movie

HOW TO ACCESS KAROLINA

- The Martian movie

HOW TO ACCESS KAROLINA

- Command line interface
- Connect via ssh protocol

- SSH server on Karolina
- SSH client on your computer
- Connect from your computer to Karolina
  - Similar to remote desktop, but command-line interface only

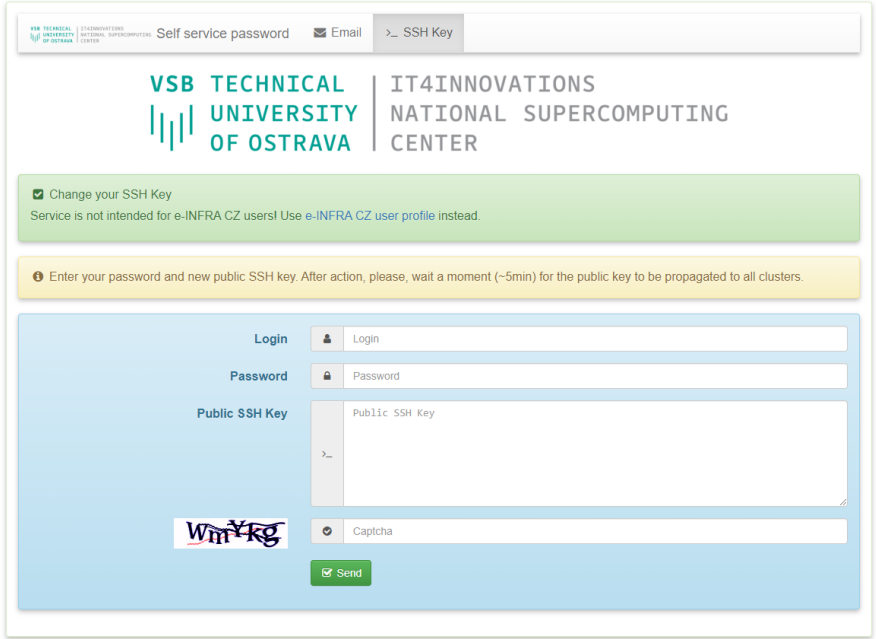
```
jakub@Jakub-PC ~
$ ssh hom0056@karolina.it4i.cz

                    _ __                  _ _
                   | |/ /                 | (_)
                   | ' / __ _ _ __ ___ | |_ _ __   __ _
                   |  < / _` | '__/ _ \| | | '_ \ / _` |
                   | . \ (_| | | | (_) | | | | | | (_| |
                   |_|\_\__,_|_|  \___/|_|_|_| |_|\__,_|

              ...running on Red Hat Enterprise Linux 7.x


Public Service Announcement: Slurm workload manager on Barbora cluster from 17.7.2023
Posted:                                                    (2023-05-16 18:09:45)

We would like to inform you that preliminary plan is to implement Slurm
workload manager on Barbora cluster to allocate and access systems resources
from 17.7.2023


hom0056@login3.karolina.it4i.cz ~
$
```

- ssh – connect and do work
- scp – copy files between Karolina and your computer

HOW TO ACCESS KAROLINA

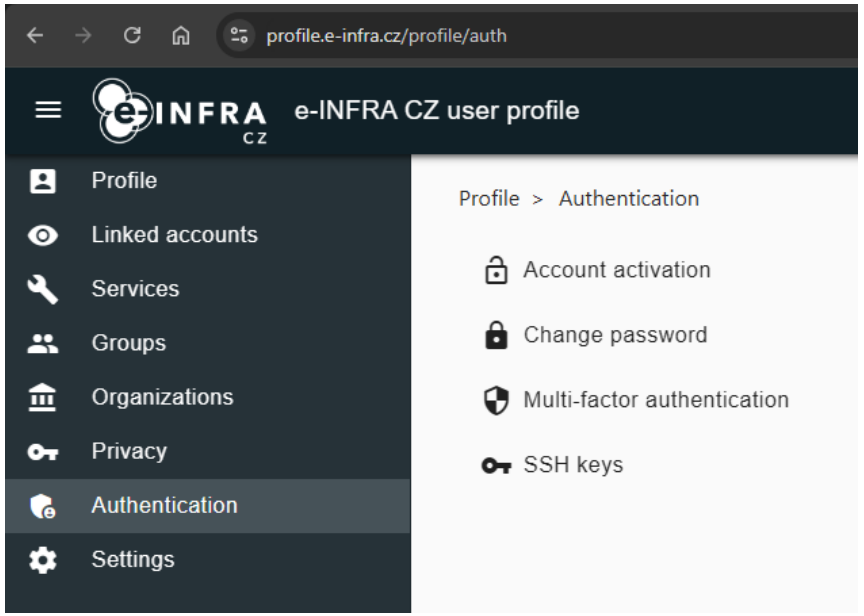
- SSH keys for authentication
  - Private-public key pair
  - Password auth. is disabled on Karolina
- Examine the .ssh directory
  - /home/<username>/.ssh
  - C:/Users/<username>/.ssh
  - Create the directory if it does not exist
- Are there id_* and id_*.pub files?
  - This is the private and public key
- No there aren't / Yes there are, but I want to generate new keys
  - Open command line / terminal / powershell
  - Run `ssh-keygen -t ed25519`
  - Follow the instructions

- Upload your public ssh key
- https://extranet.it4i.cz/ssp/?action=changesshkey
- Use the login and password you received
- Paste the contents of your public ssh key
  - ~/.ssh/id_rsa.pub (or however you named it)



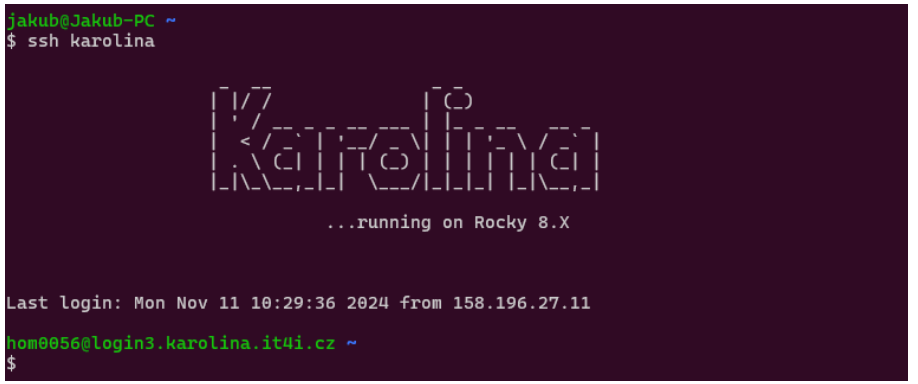
HOW TO ACCESS KAROLINA

- These were instructions for this training only
  - IT4I account
- For real projects you will use an e-INFRA.cz account
  - For users affiliated with a Czech academic institution
- https://docs.it4i.cz/general/access/einfracz-account/
- https://docs.it4i.cz/en/docs/general/get-access/account-introduction
- https://docs.it4i.cz/en/docs/general/access-services/accessing-the-clusters/shell-and-data-access



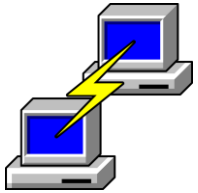
HOW TO ACCESS KAROLINA

- Connect using command line
  - ssh
  - All Linux systems (incl. MacOS)
  - Newer Windows versions



- File transfer using command line
  - scp, rsync, sshfs (more later)

- PuTTY, WinSCP
  - SSH and SCP clients for Windows



- **Visual Studio Code (VS Code)**
  - With the Remote SSH extension
  - For Linux, Windows and MacOS
  - Usable modern IDE, work directly on remote machine
  - Based on ssh and scp

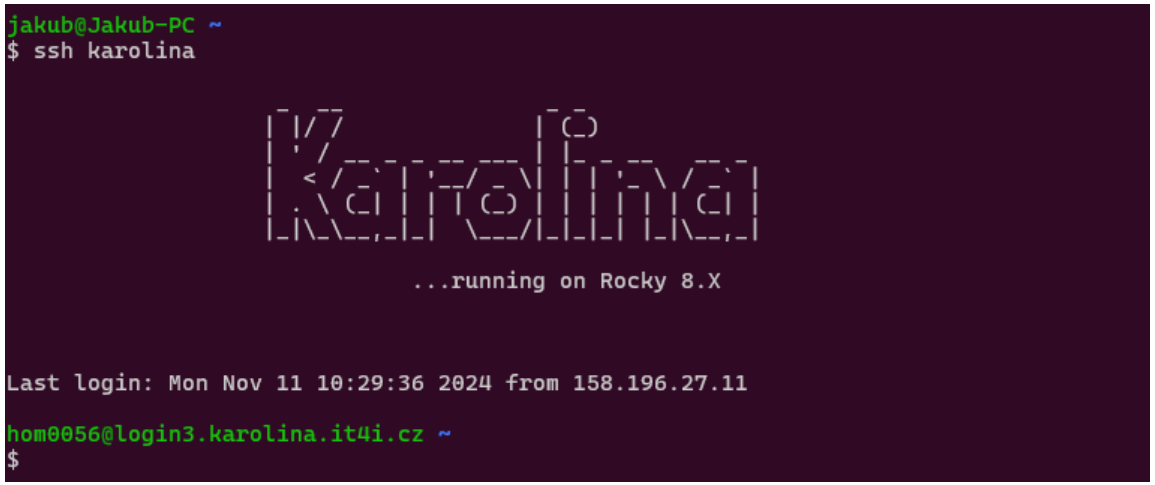
HOW TO ACCESS KAROLINA THROUGH SSH

- Open command line / terminal / powershell, and run
  - **`ssh -i ~/.ssh/id_ed25519 username@karolina.it4i.cz`**
- Too complicated, right?
- Setup an SSH config file
  - Open the ~/.ssh/config file (create it if it does not exist)
    - Linux: **`nano ~/.ssh/config`**
    - Win: **`notepad ~/.ssh/config`**
  - Add the following lines:

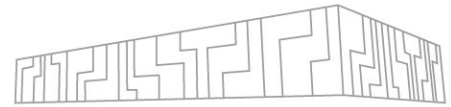
```
Host karolina
    User <username>
    HostName karolina.it4i.cz
    IdentityFile ~/.ssh/id_ed25519
```

- Then use just
  - **`ssh karolina`**

On Windows I recommend the Terminal app: https://www.microsoft.com/store/productId/9N0DX20HK701



TRANSFER FILES USING SCP



- SCP – Secure CoPy
  - on windows, file permissions may get weird

- Transfer files between local computer and remote server

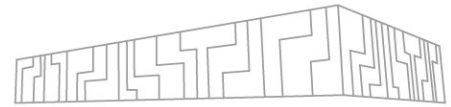
```
scp -i /path/to/key source.txt username@example.com:/path/to/destination/
```

- With correct ssh config, you can just use

```
scp source.txt karolina:~/destination/
scp karolina:/source/path/file.txt destination/
```

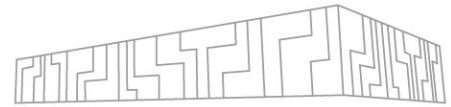
- Use `-r` option to copy directories recursively

TRANSFER FILES USING RSYNC



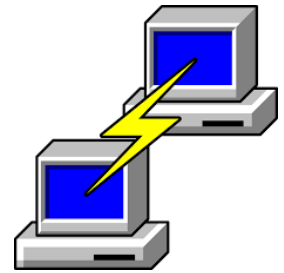
- rsync – remote sync
- Linux only
- Similar to SCP, synchronizes source into destination
- `rsync -r -u --delete source/dir/ destination/dir/`
  - Supports remote source or destination, similar to scp
- Has the option to copy only files that have changed (`-u` option)
- Option `--delete` to delete destination files which are not in source

TRANSFER FILES USING SSHFS

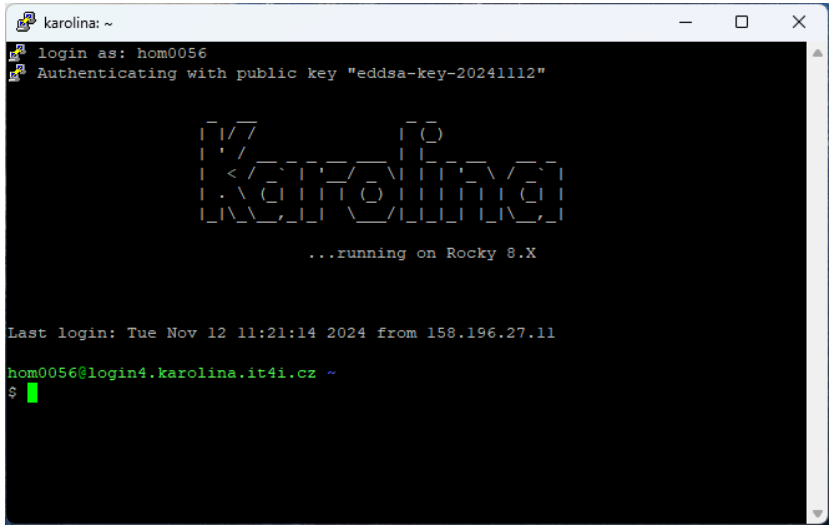


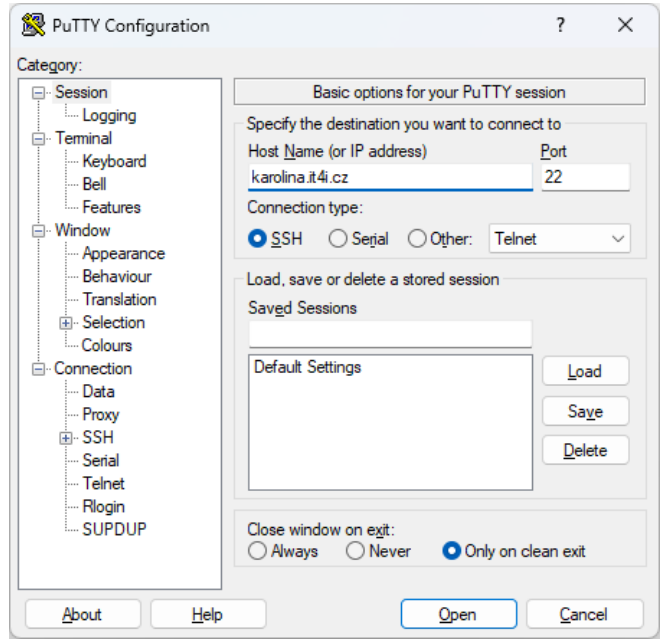
- SSH File System
- Linux only
- Mount a remote directory into local filesystem
- `mkdir karolina_home`
- `sshfs karolina:/home/username karolina_home`
- Slow to work with – every file operation needs to travel to the remote server and back
- But simple – work with remote files as you would with local ones
  - Like a symbolic link to a different file system

ACCESS KAROLINA USING PUTTY



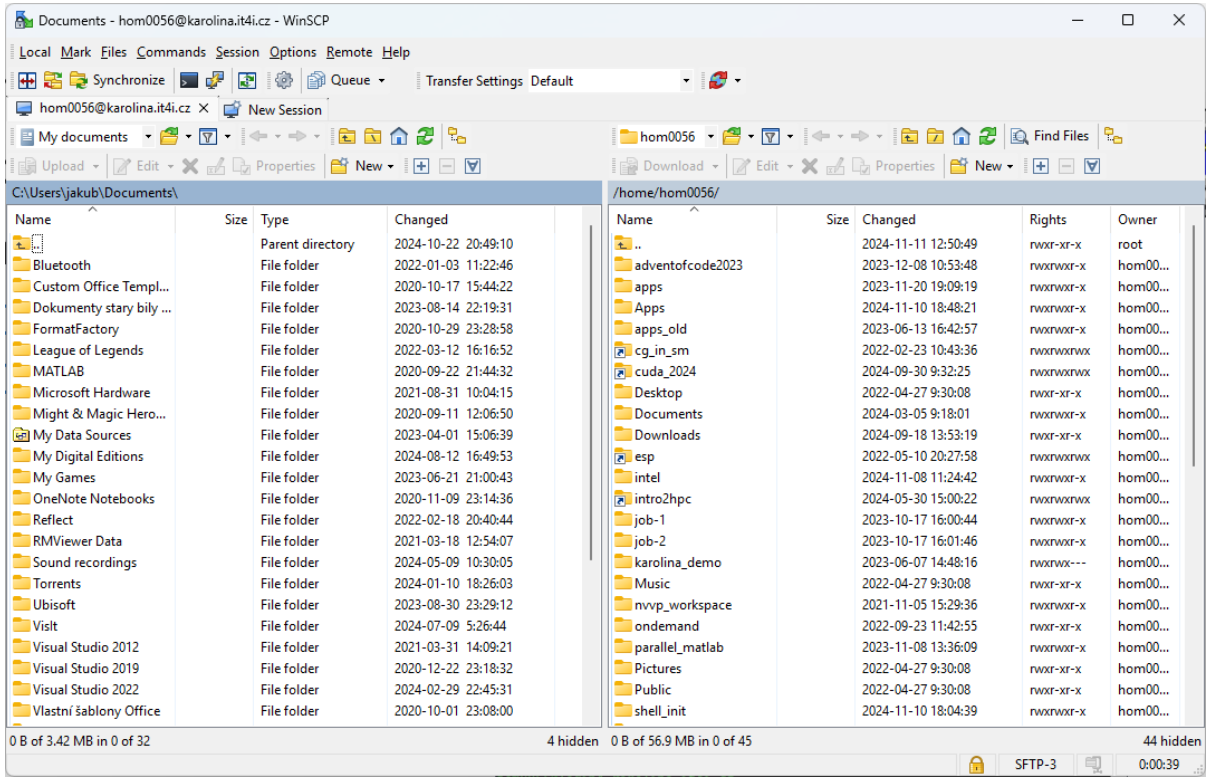
- ssh client for Windows
- GUI to setup username, server address, ssh keys, …
- Download from
  - https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html
- Uses different ssh key format
  - Need to generate and upload new keys
- I do not recommend
  - ssh command works on windows just fine





TRANSFER FILES USING WINSCP

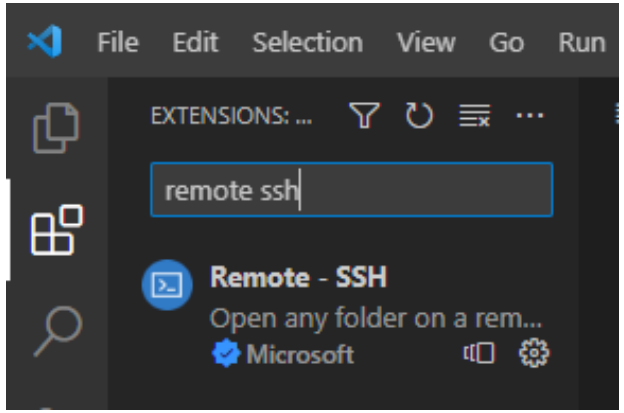
- Explore and copy files from remote server
- Works with same ssh keys as PuTTY
- Download from
  - https://winscp.net/eng/download.php



HOW TO ACCESS KAROLINA, VS CODE SETUP

- Install VS Code: https://code.visualstudio.com/

- Install the Remote - SSH extension
  - Open VS Code
  - Left pane -> extensions -> search “Remote – SSH” -> install



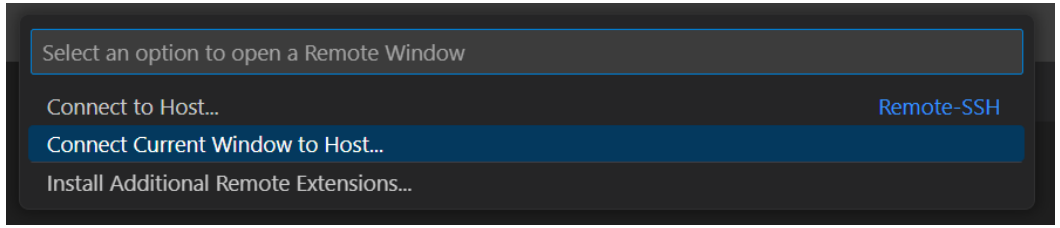
- Setup ssh config if you haven’t yet
  - ~/.ssh/config

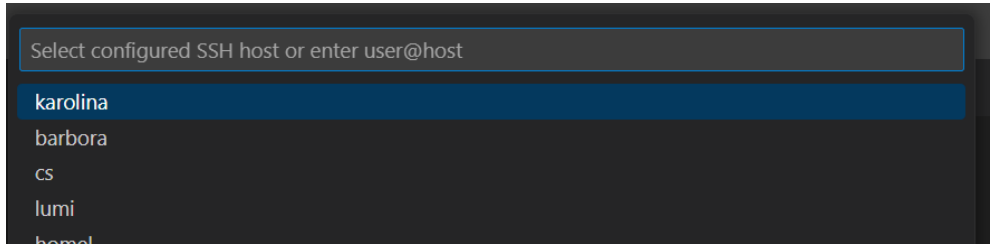
```
Host karolina
    User <username>
    HostName karolina.it4i.cz
    IdentityFile ~/.ssh/id_ed25519
```

HOW TO ACCESS KAROLINA, VS CODE SETUP

Connect to Karolina in VS Code

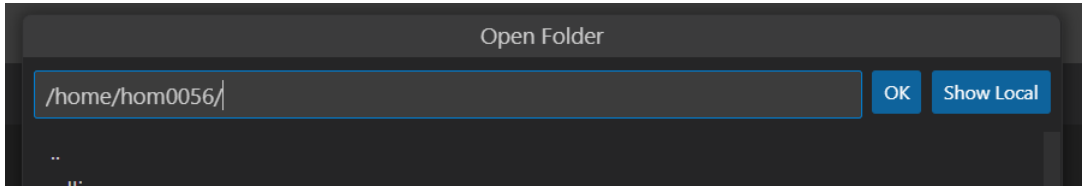
- Click the green button in bottom-left corner
- Connect Current window to host
- Choose Karolina from the list
- Now you are connected to Karolina





Open your home folder

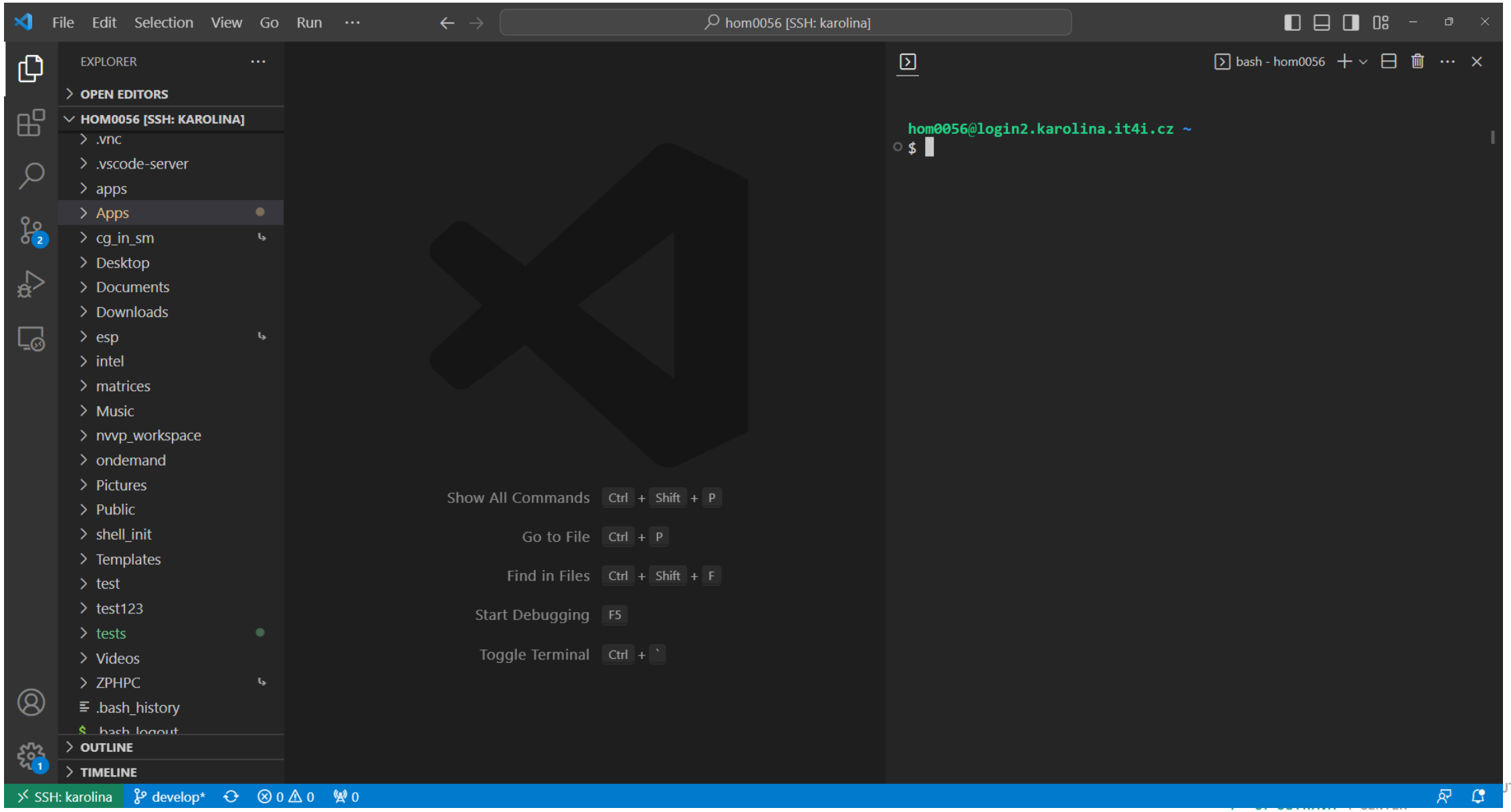
- File -> Open Folder, or click Open Folder on the welcome screen
- Keep the home folder in the textbox and click OK



Open terminal in VS Code

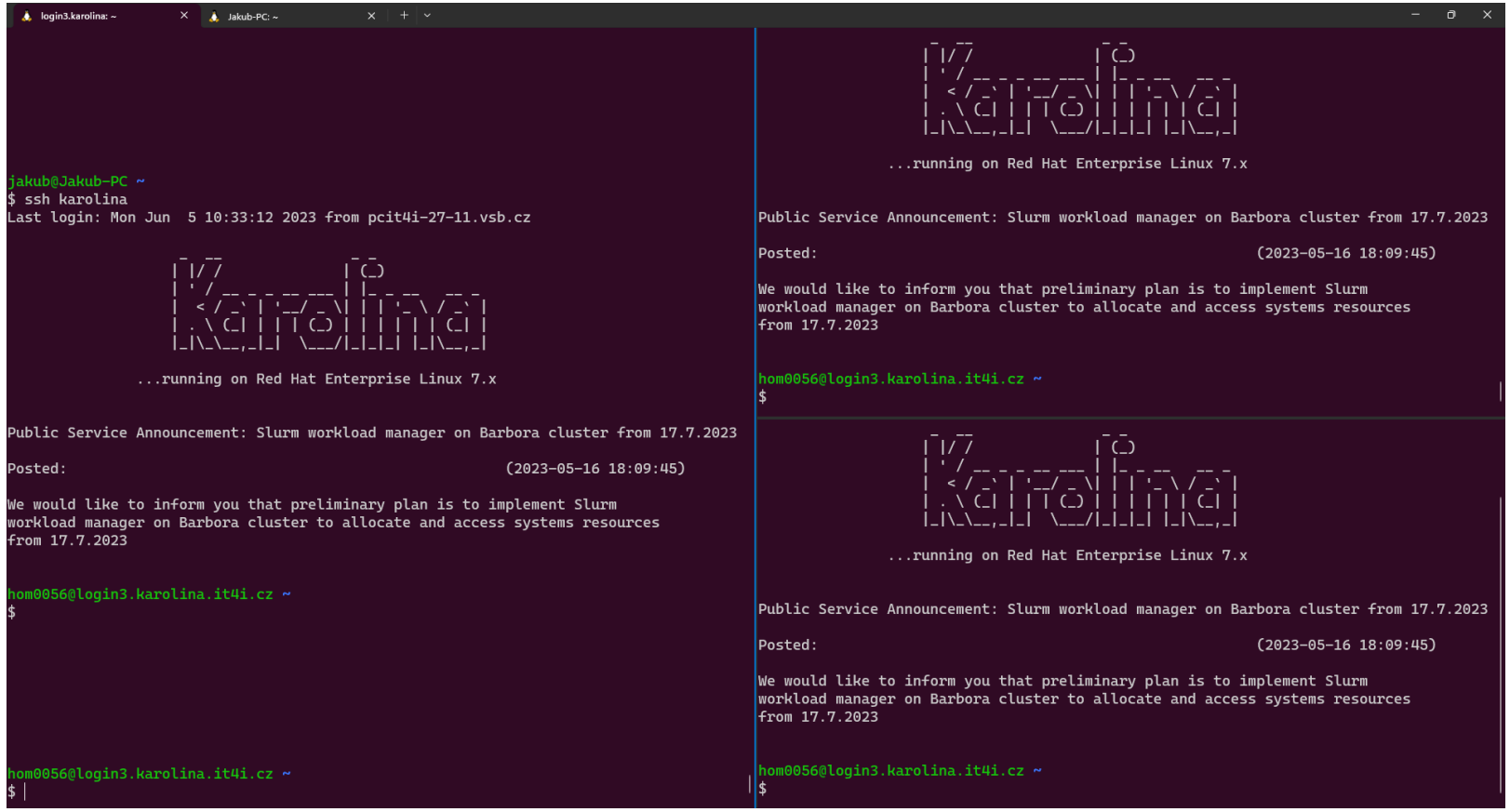
- Terminal -> New Terminal
- You will be connected to Karolina already

HOW TO ACCESS KAROLINA, VS CODE SETUP



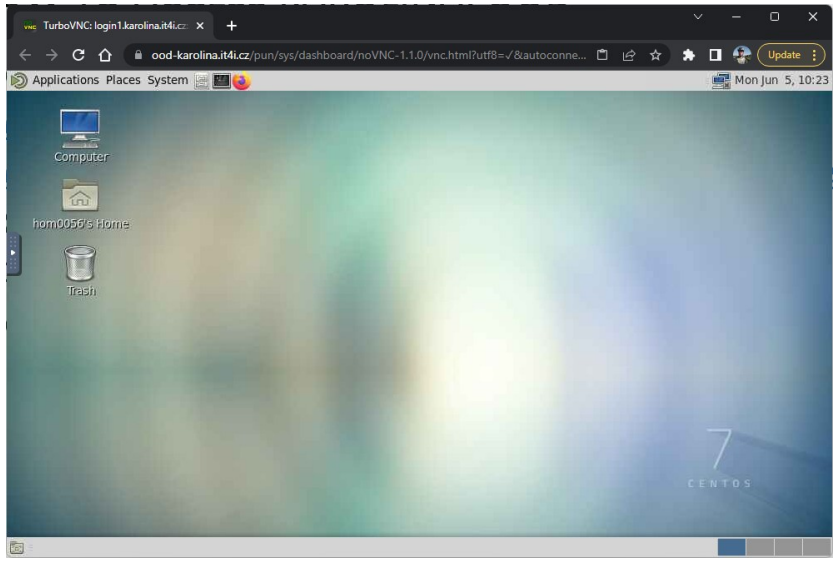
HOW TO ACCESS KAROLINA

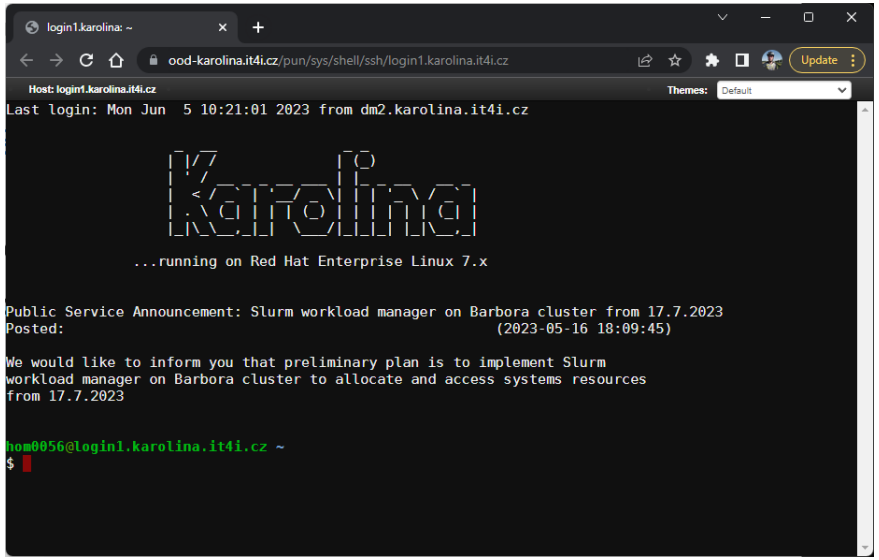
- One terminal is usually not enough (for me)
- Multiple terminals in VS Code feels cramped
- Recommend to use a separate app for command line access



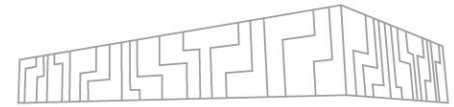
HOW TO ACCESS KAROLINA, OOD

- Another option – OOD, open on demand
- https://ood-karolina.it4i.cz/
- https://docs.it4i.cz/en/docs/general/access-services/graphical-user-interface/ood
- Requires IT4I VPN (which all IT4I users have access to)
- Command line interface in a web browser
- Also supports GUI applications (remote desktop in essence)





HOW TO ACCESS KAROLINA, OTHER GUI OPTIONS



- X11 forwarding
  - Window and GUI management through ssh
  - Use -x flag with the ssh command
  - Very slow

- VNC
  - Similar to the OOD
  - No need for VPN
  - Complicated set up
  - https://docs.it4i.cz/en/docs/general/access-services/graphical-user-interface/vnc

QUICK SUMMARY OF KAROLINA TECHNICAL SPECIFICATIONS

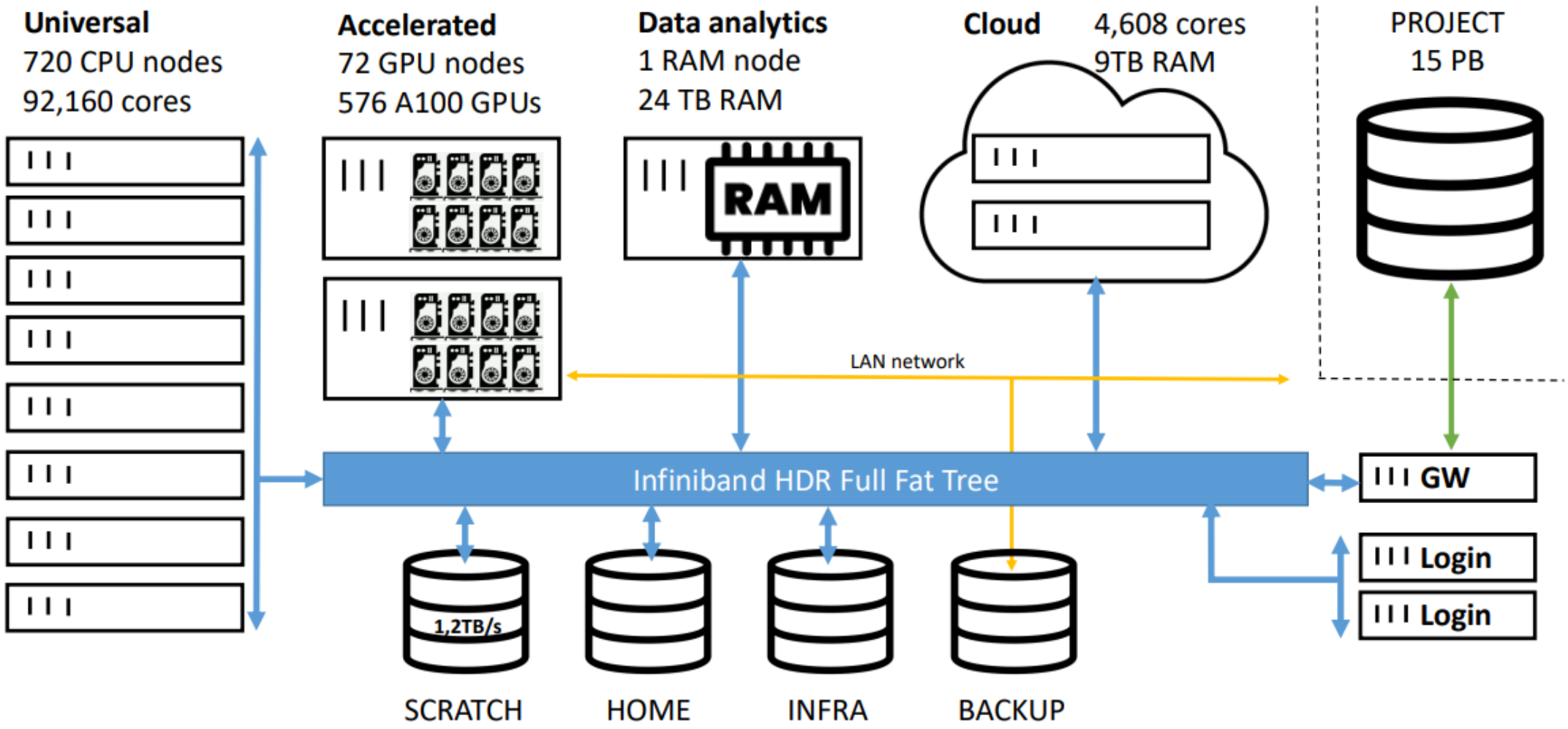
And Barbora and LUMI

KAROLINA SUPERCOMPUTER

KAROLINA SUPERCOMPUTER

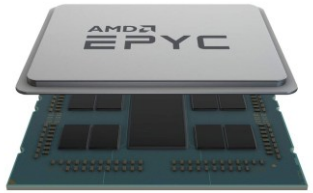
- 15.7 Pflop/s
- #69 first appearance in TOP500

- https://www.it4i.cz/en/infrastructure/karolina
- https://docs.it4i.cz/en/docs/clusters/karolina/introduction

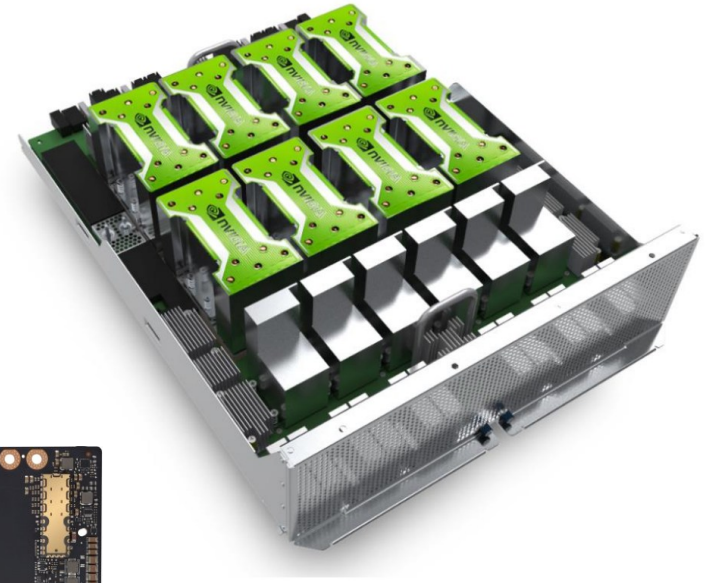


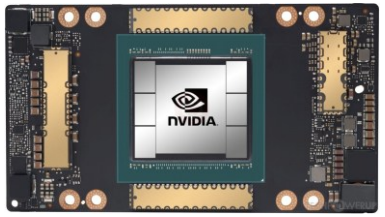
KAROLINA SUPERCOMPUTER

- Universal partition: 720 compute nodes
  - 2x 64-core AMD EPYC 7H12 @ 2.6 GHz
  - 256 GB of memory
  - 346 GB/s memory bandwidth, 5.3 Tflop/s per node
  - 100 Gb/s NIC (infiniband HDR100)
  - 3.8 Pflop/s peak total

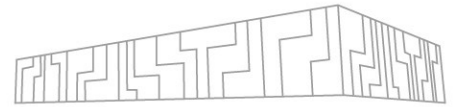


- GPU-accelerated partition: 72 compute nodes
  - 2x 64-core AMD EPYC 7763 @ 2.45 GHz
  - 1024 GB of memory
  - 8x NVIDIA A100 SXM4 40GB
  - 12.4 TB/s memory bandwidth, 156 Tflop/s per node
  - 4x 200 Gb/s NIC
  - Total 11.1 Pflop/s peak



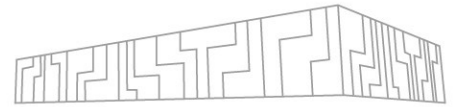


KAROLINA SUPERCOMPUTER



- Fat node (data analytics partition)
  - 32x 24-core Intel Xeon 8268 @ 2.9 GHz
  - 24 TB RAM
  - 2x 200 Gb/s NIC
  - 2.56 TB/s memory bandwidth, 70 Tflop/s peak
- Cloud partition, 36 nodes – very similar to universal partition
- Login nodes

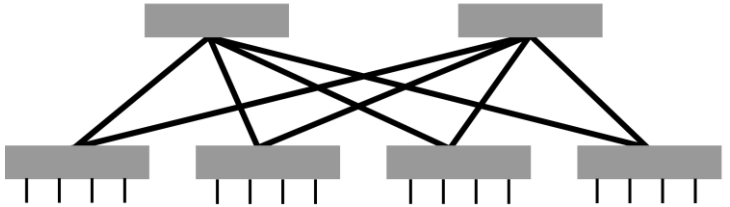
STORAGE

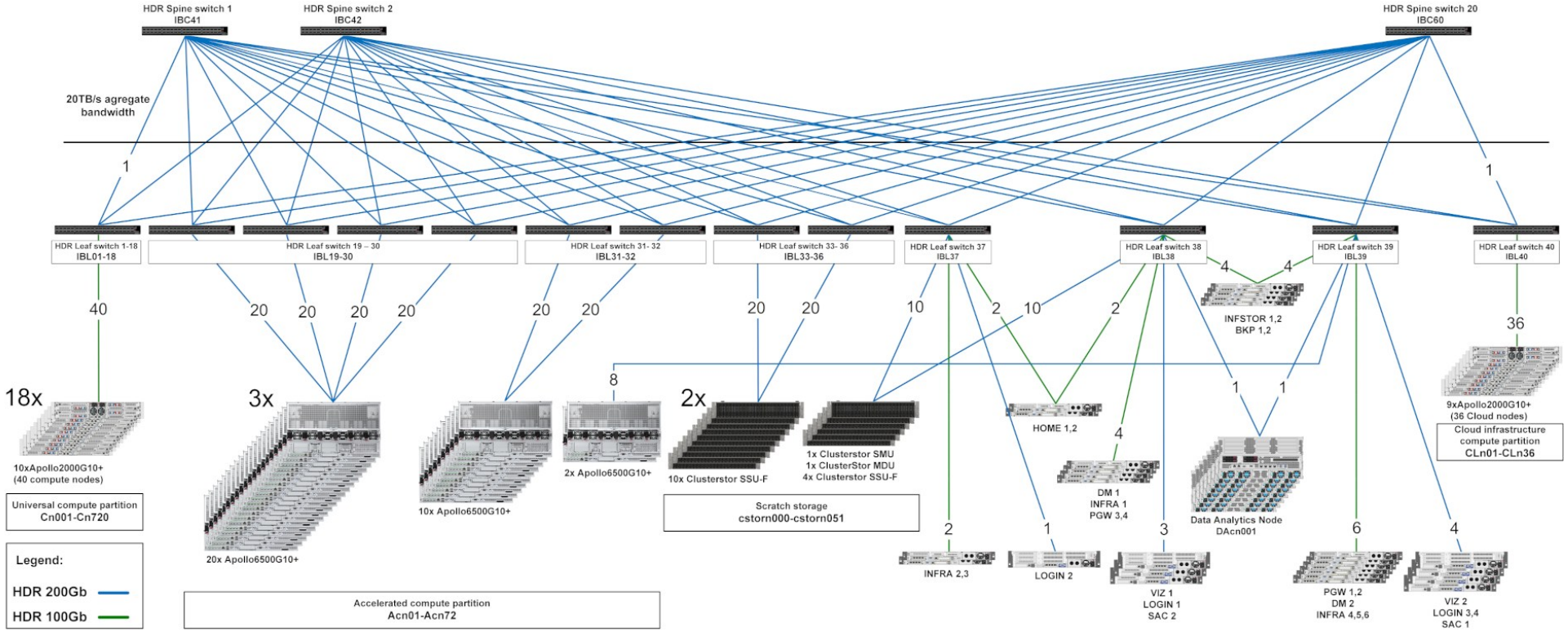


- HOME
  - For basic user data
  - Mounted at /home
  - Contains user's home directories
  - Limit of 25 GB per user, 31 TB total
  - 2-3 GB/s throughput
- SCRATCH
  - For fast parallel I/O
  - Checkpoints, space for temporary data
  - Mounted at /scratch
  - Parallel Lustre filesystem
  - NOT for long-term file storage
  - 731 GB/s write, 1198 GB/s read
  - 1 PB total capacity, max 10 TB per user
- PROJECT
  - For files shared among users in a project
  - 15 PB total capacity
  - Limit 20 TB per project
  - ~40 GB/s throughput
  - /mnt/proj1, /mnt/proj2, /mnt/proj3
    - it4i-get-project-dir <projectid>
- it4ifsusage, it4i-disk-usage
  - Show info about filesystems usage

KAROLINA SUPERCOMPUTER

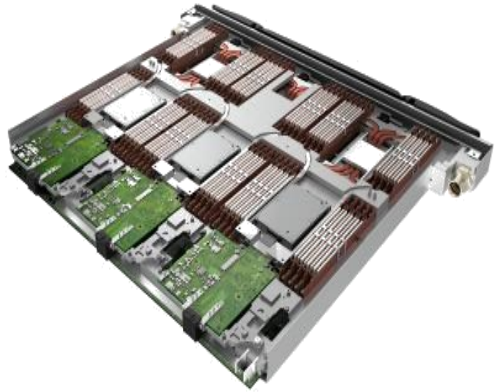
- Network
  - Non-blocking fat tree
  - 60x 40-port Mellanox Quantum HDR switches
  - https://docs.it4i.cz/en/docs/clusters/karolina/network





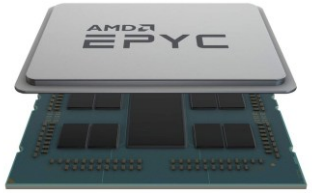
BARBORA

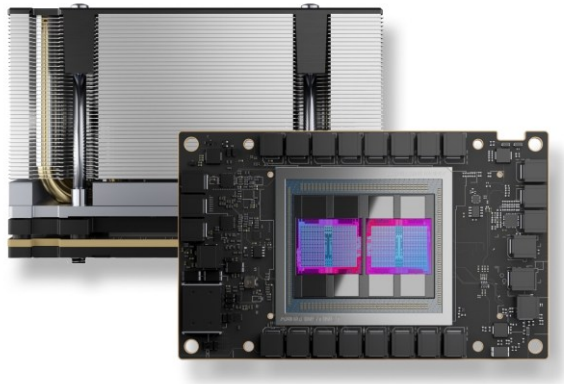
- Since October 2019
- 192 CPU nodes
  - 2x 18-core Intel Cascade Lake 6240 2.6 GHz
  - 192 GB DDR4 RAM
  - 1x 100 Gb/s NIC
- 8 GPU nodes
  - 2x 12-core Intel Skylake Gold 6126, 2.6 GHz
  - 192 GB DD4 RAM
  - 4x NVIDIA Tesla V100-SXM2
  - 2x 100 Gb/s NIC
- 1 Fat node
  - 8x 16-core Intel Skylake 8153, 2 GHz
  - 6 TB DDR4 RAM
  - 2x 100 Gb/s NIC

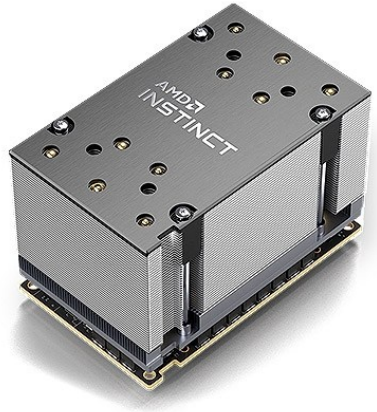


LUMI

- In Finland (Kajaani)
- Originally #3 in TOP500, 531.5 Pflop/s
- Czechia gets 2.5 % of the resources
- 1888+128+32 CPU nodes
  - 2x 64-core AMD Epyc 7763, 2.45 GHz
  - 256/512/1024 GB DDR4 RAM
  - 1x 200 Gb/s NIC
- 2978 GPU nodes
  - 1x 64-core AMD Epyc Trento
  - 512 GB DDR4 RAM
  - 4x AMD MI250x GPU (128 GB of HBM2e memory each)
  - 4x 200 Gb/s NIC



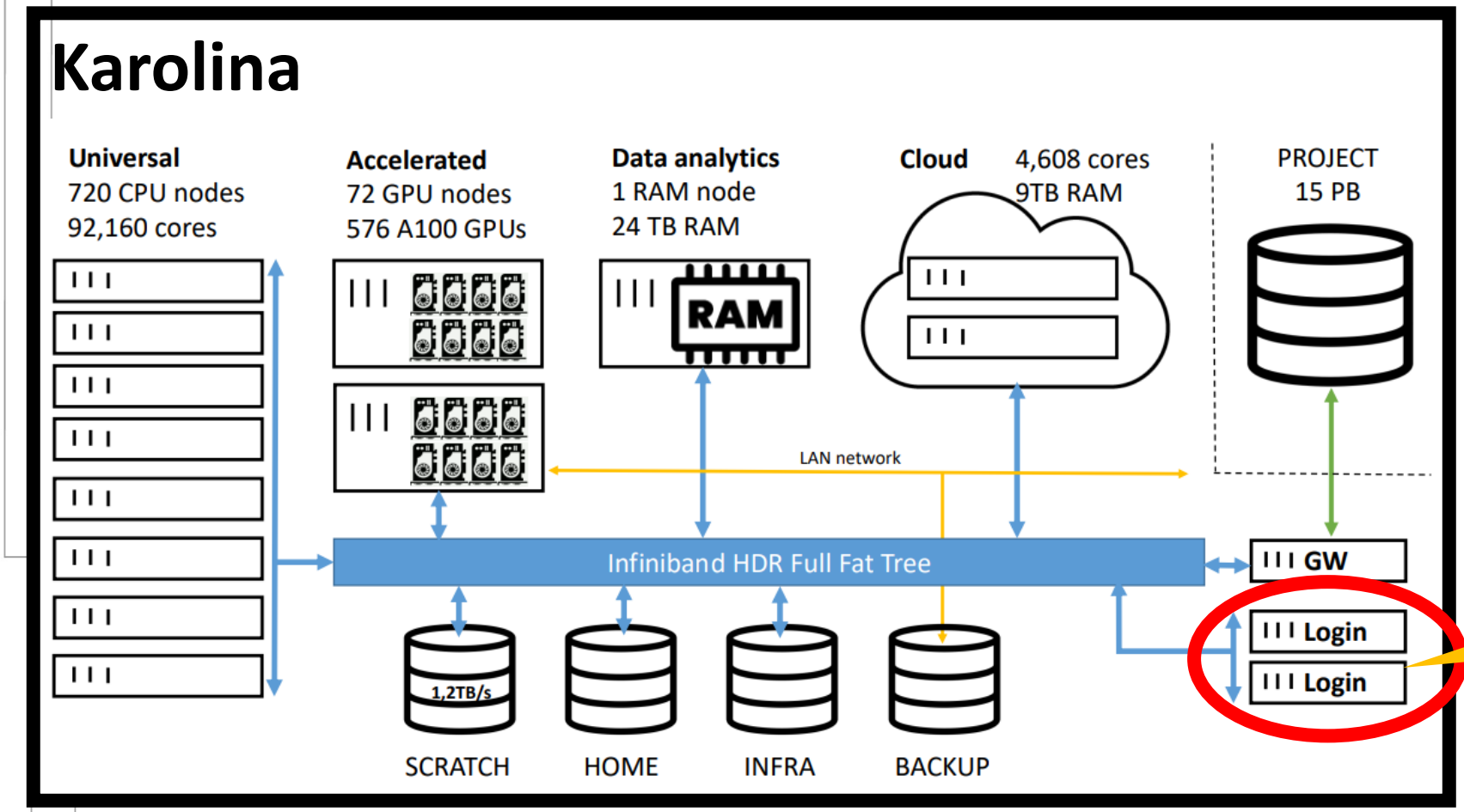




ACCESSING KAROLINA COMPUTATIONAL RESOURCES

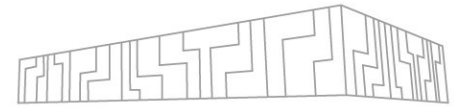
SLURM job scheduler, jobs

WHEN YOU SSH TO KAROLINA



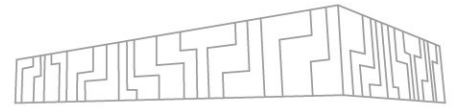

```
jakub@Jakub-PC ~
$ ssh hom0056@karolina.it4i.cz

                    _ __                  _ _
                   | |/ /                 | (_)
                   | ' / __ _ _ __ ___ | |_ _ __   __ _
                   |  < / _` | '__/ _ \| | | '_ \ / _` |
                   | . \ (_| | | | (_) | | | | | | (_| |
                   |_|\_\__,_|_|  \___/|_|_|_| |_|\__,_|

            ...running on Red Hat Enterprise Linux 7.x


Public Service Announcement: Slurm workload manager on Barbora cluster from 17.7.2023
Posted:                                                    (2023-05-16 18:09:45)

We would like to inform you that preliminary plan is to implement Slurm
workload manager on Barbora cluster to allocate and access systems resources
from 17.7.2023


hom0056@login3.karolina.it4i.cz ~
$
```

ACCESSING KAROLINA COMPUTATIONAL RESOURCES



- Login nodes
  - SSHing to Karolina will connect you to one of 4 login nodes
  - Copy files, prepare code, setup environment, light compilation and testing
  - Write and submit job scripts to run on compute nodes
  - NO COMPUTATIONS ON LOGIN NODES
- Compute nodes
  - For computations, use the compute nodes
  - CPU-only nodes – universal partition
  - GPU-accelerated nodes – accelerated partition
  - Fat node, visualization nodes, cloud partition
- Use Slurm job scheduler to request access to a compute node
  - I need 8 CPU nodes for 6 hours. When it’s available, run this script.

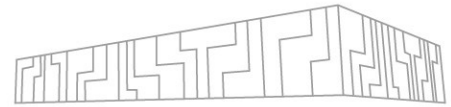
SLURM



-A, --account
-p, --partition
-N, --nodes
-t, --time
-G, --gpus

- 2 types of jobs
  - Batch jobs – run a script when resources are available
  - Interactive jobs – execute commands directly on the node in an interactive shell
- salloc – submit an interactive job
  - `salloc --account DD-24-88 --partition qcpu --nodes 2 --time 4:00:00`
  - `salloc -A DD-24-88 -p qcpu -N 2 -t 4:00:00`
- sbatch – submit a batch job
  - `salloc -A DD-24-88 -p qcpu -N 2 -t 4:00:00 ~/path/to/script.sh`
- Request 2 nodes from the CPU partition for 4 hours. The consumed node-hours will be accounted to project DD-24-88.
- Slurm will enqueue the job
  - Complicated optimization problem – how to schedule the jobs
  - When the resources are available, the job will run
- https://docs.it4i.cz/en/docs/general/run-jobs/job-sub-exec/karolina-slurm
- https://slurm.schedmd.com/man_index.html

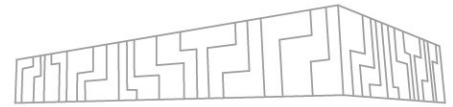
PROJECTS



- A group of users sharing HPC resources
  - One project can have multiple users
  - One user can be involved in multiple projects
- Each project has a given number or computational time (node-hours) it can use
  - CPU node-hours, GPU node-hours, …
- Each job consumes the node-hours from a given project (-A in salloc/sbatch)
  - Number of nodes * number of hours = number of node-hours used

- One can request joining a project on https://scs.it4i.cz/
  - The primary investigator (PI) will approve/deny the request
- it4ifree command
  - List all projects you are a member of
  - See used and remaining node-hours

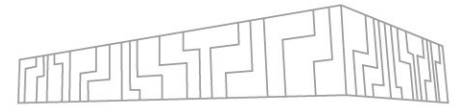
SLURM PARTITIONS (QUEUES)



- qcpu
  - The main queues, normal priority, universal partition
  - Maximum time 48 hours
- qcpu_long
  - Same as qcpu, but:
  - Maximum time is 144 hours (6 days)
- qgpu
  - Queue for the GPU accelerated partition
  - Maximum time is 48 hours
  - Possible to allocate just a fraction of a node (down to single GPUs)
- qcpu_exp, qgpu_exp
  - Express queues, higher priority, for short quick jobs, meant for development and testing,
  - More limited maximum resources, max. 1 hour
- qcpu_free, qgpu_free
  - Can be used after project resources have been depleted
    - Up to 150 % of allocated node-hours
  - Lower priority and shorter maximum time
- qcpu_preempt, qgpu_preempt
  - Similar to _free, even lower priority
    - No node-hour limit
  - Job can be killed if higher-priority job appears

https://docs.it4i.cz/en/docs/general/run-jobs/resource-allocation/karolina-partitions

SLURM JOB SUBMISSION EXAMPLES



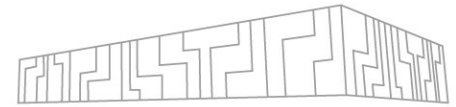
-A, --account
-p, --partition
-N, --nodes
-t, --time
-G, --gpus

- **`salloc -A DD-24-88 -p qcpu -N 2 -t 4:00:00`**
  - Request 2 nodes from the CPU partition for 4 hours for interactive access.
    The consumed node-hours will be accounted to project DD-24-88.
- **`salloc -A DD-24-88 -p qcpu`**
  - Use default N=1 and T=24:00:00. Interactive job
  - Other queues have different defaults
- **`sbatch ./script.sh`**
  - Submission of a batch job, execute ./script.sh
  - Takes the arguments from the job script
  - The script will be run on one of the compute nodes
- **`sbatch -t 5:00:00 ./script.sh`**
  - Takes the arguments from the job script,
    but overrides time to 5 hours

```
#!/usr/bin/bash
#SBATCH --job-name MyJobName
#SBATCH --account DD-24-88
#SBATCH --partition qcpu
#SBATCH --nodes 4
#SBATCH --ntasks-per-node 128
#SBATCH --time 12:00:00

# ... do some work ...
```

https://docs.it4i.cz/en/docs/general/run-jobs/resource-allocation/karolina-partitions

SLURM JOB SUBMISSION EXAMPLES, GPU QUEUES



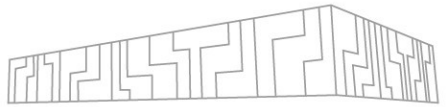
- 8 GPUs and 128 CPU cores per node
- Possible to allocate a multiple of 1 GPU and 16 cores = 1/8 of the node

-A, --account
-p, --partition
-N, --nodes
-t, --time
-G, --gpus

- `salloc -A DD-24-88 -p qgpu --gpus 1 --nodes 1 --time 2:00:00`
  - Request 1 GPU on 1 node for 2 hours
- `salloc -A DD-24-88 -p qgpu`
  - Default: 1 GPU, 1 node, 24h time limit
- `salloc -A DD-24-88 -p qgpu --gpus 4 --time 2:00:00`
  - Request 4 GPUs for 2 hours. You might get the GPUs scattered across 1-4 nodes
- `salloc -A DD-24-88 -p qgpu --gpus 4 --nodes 1 --time 2:00:00`
  - Request 4 GPUs on 1 node for 2 hours
- `salloc -A DD-24-88 -p qgpu --gpus 16 --nodes 2 --time 2:00:00`
  - Request 16 GPUs on 2 nodes for 2 hours. You will get 2 full nodes.
- `salloc -A DD-24-88 -p qgpu --gpus-per-node 8 --nodes 2 --time 2:00:00`
  - Request 2 nodes, with 8 GPUs per node (16 GPUs total), for 2 hours

- No way to enforce to get e.g. 4 “neighboring” GPUs on the node

MORE SLURM COMMANDS



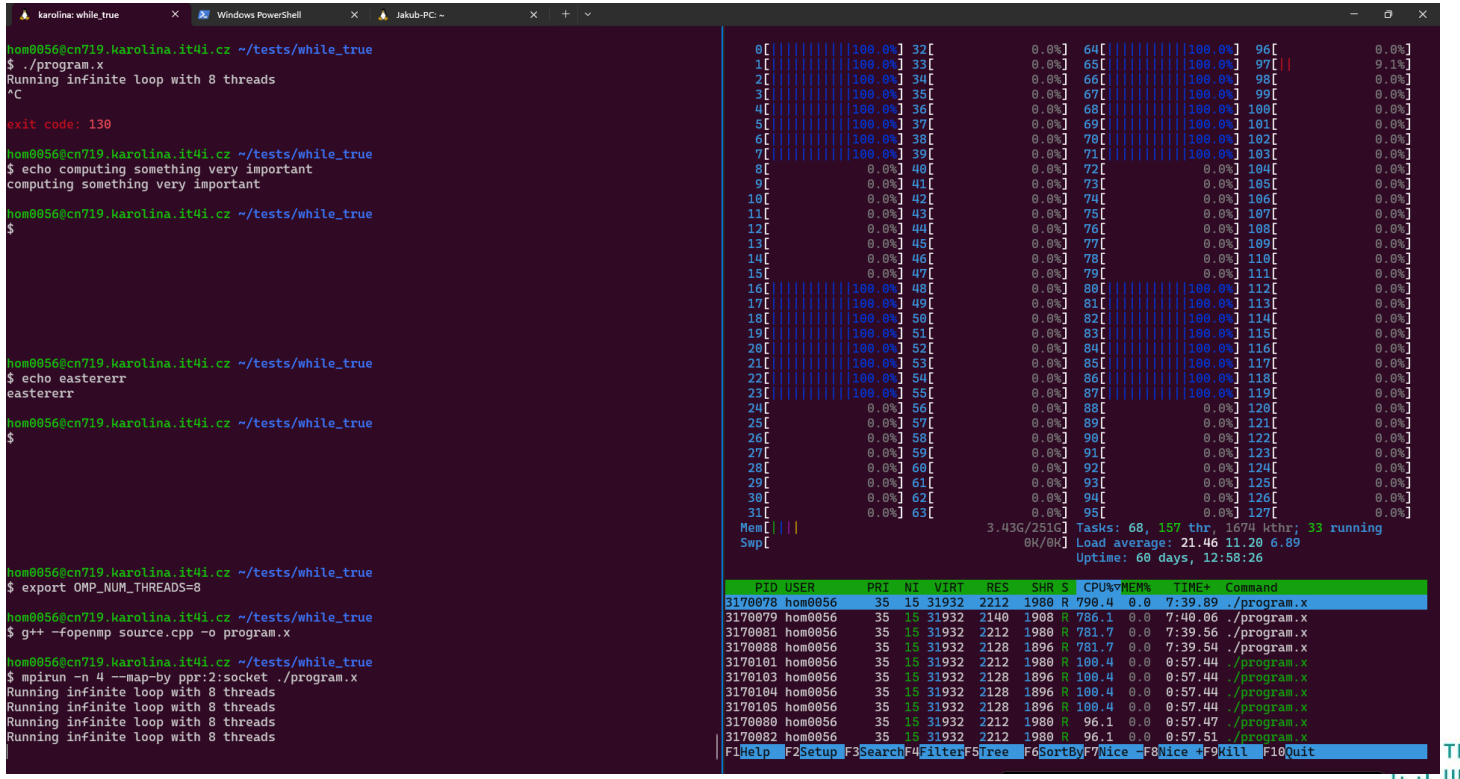
- salloc, sbatch
  - submit a job
- scancel <jobid>
  - Cancel/delete a job
- squeue
  - List all running, queued and recently finished jobs
- squeue -u <username>
  - Limit squeue to specific user
- squeue --me
  - Show only my jobs in squeue
- scontrol show job <jobid>
  - Show job details
- sinfo
  - View summary information about allocated nodes in partitions

```
hom0056@login2.karolina.it4i.cz ~
$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           1874986  qcpu_exp interact  hom0056 CD       0:04      1 cn285
           1875020  qcpu_exp interact  hom0056  R       0:19      1 cn719
           1875019      qgpu interact  hom0056 PD       0:00      1 (Priority)
```

```
hom0056@login2.karolina.it4i.cz ~
$ scontrol show job 1875019
JobId=1875019 JobName=interactive
   UserId=hom0056(5825) GroupId=hom0056(6398) MCS_label=N/A
   Priority=300000000 Nice=0 Account=open-28-64 QOS=2374_3679
   JobState=PENDING Reason=Priority Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=0 Reboot=0 ExitCode=0:0
   RunTime=00:00:00 TimeLimit=08:00:00 TimeMin=N/A
   SubmitTime=2024-11-12T23:39:19 EligibleTime=2024-11-12T23:39:19
   AccrueTime=2024-11-12T23:39:19
   StartTime=2024-11-12T23:40:23 EndTime=2024-11-13T07:40:23 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2024-11-12T23:40:11 Scheduler=Main
   Partition=qgpu AllocNode:Sid=login4.karolina.it4i.cz:1941205
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=
   NumNodes=1-1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   ReqTRES=cpu=1,mem=7760M,node=1,billing=4,gres/gpu=4
   AllocTRES=(null)
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryCPU=7760M MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/bin/sh
   WorkDir=/home/hom0056
   Power=
   CpusPerTres=gres/gpu:16
   TresPerNode=gres/gpu:4
```

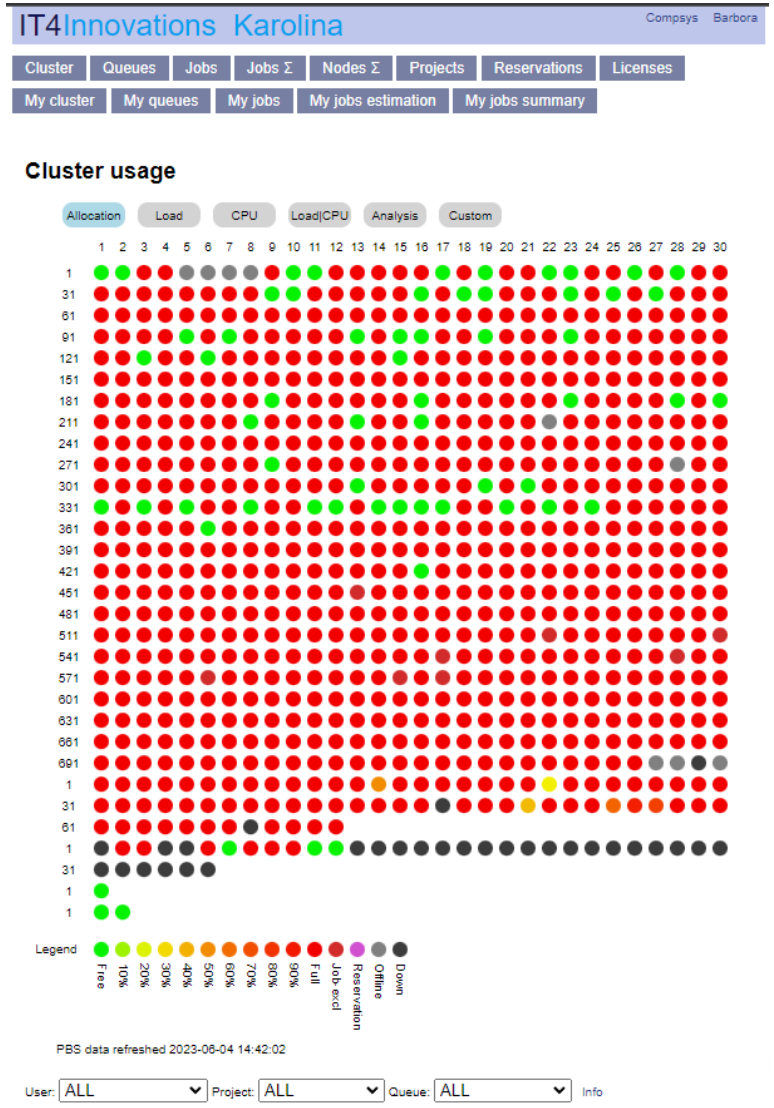
OTHER SLURM NOTES

- You can ssh to a node where your job is running
- Find where your job is running
  - squeue --me
- From the login node, ssh to the compute node
  - ssh cn123

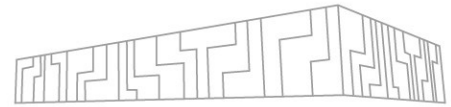


CLUSTER UTILIZATION

- Web interface to see real-time cluster allocation
  - https://extranet.it4i.cz/rsweb/karolina/cluster-allocation
  - https://extranet.it4i.cz/grafana/d/IYj4zYL7k/karolina-cluster



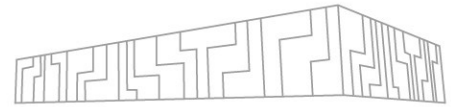
SLURM RESERVATIONS



- Reservations – sometimes you need access to the compute nodes at a specific time
  - E.g., during a training or a lecture
- Requesting a reservation
  - **`scontrol create reservation`** – banned, because people were too creative
  - Contact support@it4i.cz to request a reservation
- **`scontrol show reservation`** – view all reservations
- Submit a job to a reservation:
  - Don’t specify partition (-p)
  - Use --reservation=<reservation_name>
  - Reservations might have different defaults

SOFTWARE

SOFTWARE, MODULES



- There is a large amount of software installed on Karolina
- Each library/program/application can have several versions
- Not accessible directly – bin directory needs to be added to PATH and other variables need to be set
  - If everything was in PATH, it would be huge, and using different versions would be hard

- module command (ml as its short version)
- module avail – shows all available modules
  - ml avail MPI – shows all available modules which contain MPI in its name (case-insensitive)
- module load <modulename> – loads the module (case-sensitive)
  - E.g., ml intel/2024a
  - ml intel – loads the default version of intel
- module unload <modulename> – unloads the module
- module purge – unloads all modules
- module list, ml – shows all currently loaded modules
- module spider <package>

SOFTWARE, MODULES

- Be careful about versions
- Module versions (toolchains) should match

```
hom0056@login1.karolina.it4i.cz ~
$ ml GCC/13.3.0

hom0056@login1.karolina.it4i.cz ~
$ ml CMake/3.29.3-GCCcore-13.3.0
```

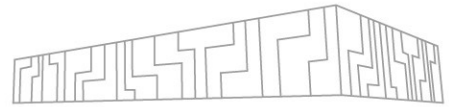
```
hom0056@login1.karolina.it4i.cz ~
$ ml GCC/14.2.0

hom0056@login1.karolina.it4i.cz ~
$ ml CMake/3.29.3-GCCcore-13.3.0

The following have been reloaded with a version change:
  1) GCCcore/14.2.0 => GCCcore/13.3.0     2) zlib/1.3.1-GCCcore-14.2.0 => zlib/1.3.1-GCCcore-13.3.0
```

- Modules have default versions
  - ml GCC == ml GCC/14.2.0
  - Default versions will change, don't rely on this

```
hom0056@login4.karolina.it4i.cz ~
$ ml av GCC/

---------------------------- /apps/modules/compiler -----------------------------
   GCC/9.3.0     GCC/10.3.0    GCC/11.3.0    GCC/12.3.0    GCC/13.3.0
   GCC/10.2.0    GCC/11.2.0    GCC/12.2.0    GCC/13.2.0    GCC/14.2.0 (D)

  Where:
   D:  Default Module
```

- Some python packages are available too
  - No need to "pip3 install" them yourself
  - ml matplotlib

SOFTWARE, MODULES



- Need to install new software
  - sudo not available, package managers will not work
- Request installation via support
  - https://docs.it4i.cz/en/docs/software/modules/new-software
  - It will then be available via modules
- Use containers
  - Singularity, docker
- Install it yourself
  - In the project or home directory
  - You will need to set PATH and other environment variables yourself
    - Or set up custom modules
  - Can be tricky to set up
- Common software install patterns:
  - Use EasyBuild or Spack packages
  - Download binaries and just use them
  - Download source code and compile it
    - Make, CMake and make, ./configure and make, waf, …
  - Just follow the software's documentation

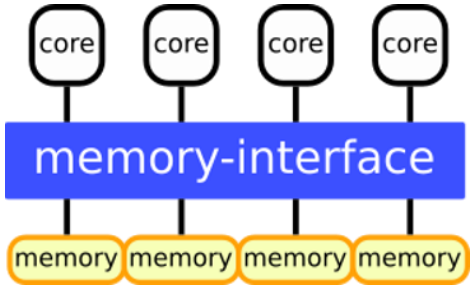
VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

INTRODUCTION TO PROGRAMMING PARALLEL APPLICATIONS

Basics of OpenMP and MPI

Multi-processor (socket, CCD in the case of AMD processors)

- all cores share the same memory
- single / global address space
- the same speed to all memory locations (uniform memory access)



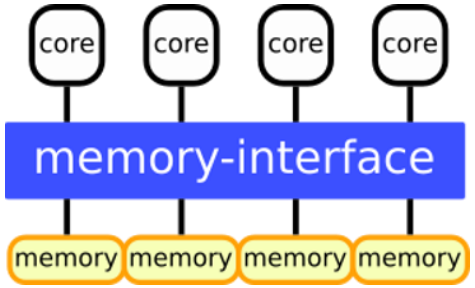
socket

UMA (uniform memory access)

SMP (symmetric multi-processing)

Several sockets with multi-processors (node)

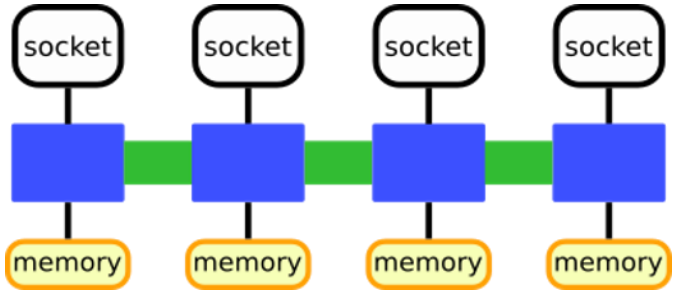
- memory is shared among all CPUs
- single / global address space
- the same speed to all memory locations (uniform memory access)?



socket

UMA (uniform memory access)

SMP (symmetric multi-processing)



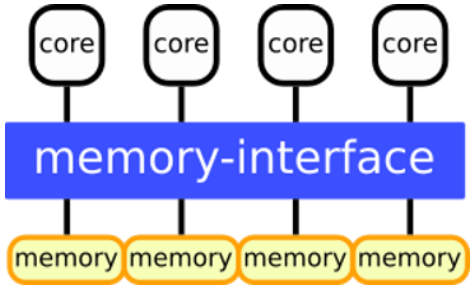
node

ccNUMA (cache-coherent non-uniform ...)

first touch, pinning!

Several sockets with multi-processors (node)

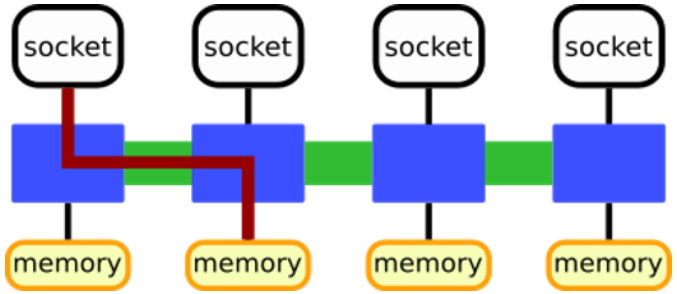
- memory is shared among all CPUs
- single / global address space
- ~~the same speed to all memory locations (uniform memory access)?~~
- the speed is dependent on a memory location (non-uniform memory access)



socket

UMA (uniform memory access)

SMP (symmetric multi-processing)



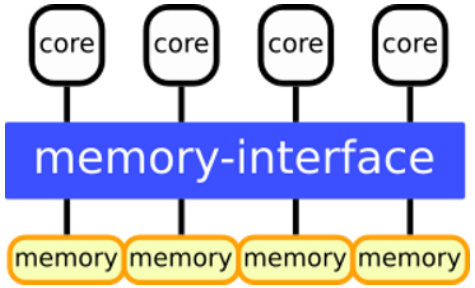
node

ccNUMA (cache-coherent non-uniform ...)

first touch, pinning!

Multi-computers with various architectures (cluster)

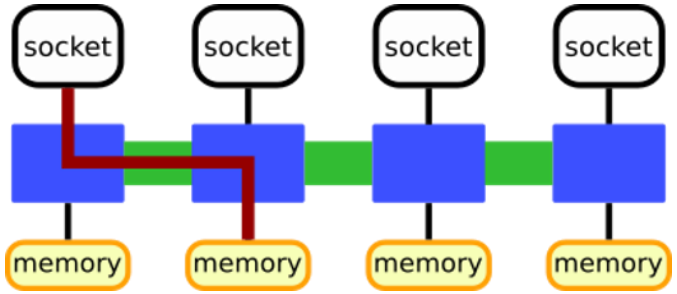
- set of nodes interconnected by a network
- each node has separated memory
- slower access to memories of other processors
- accelerated nodes



socket

UMA (uniform memory access)

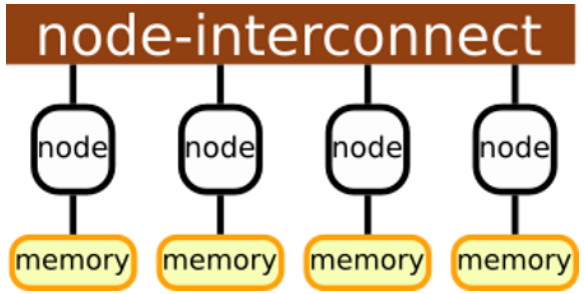
SMP (symmetric multi-processing)



node

ccNUMA (cache-coherent non-uniform ...)

first touch, pinning!



cluster

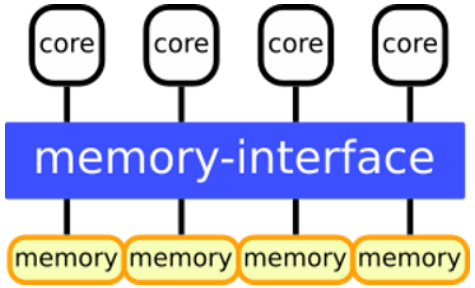
NUMA (non-uniform memory access)

fast access to own memory only

OpenMP: shared memory (socket, node)

MPI: distributed memory (socket, node, cluster)

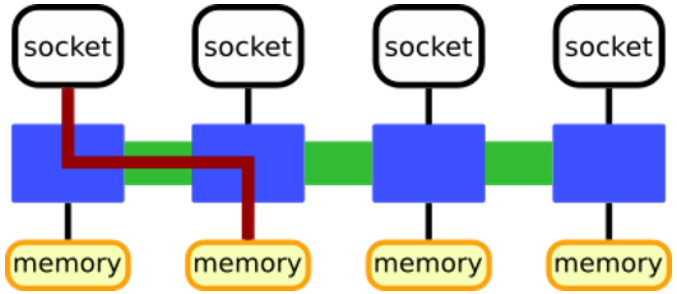
CUDA: accelerated nodes



socket

UMA (uniform memory access)

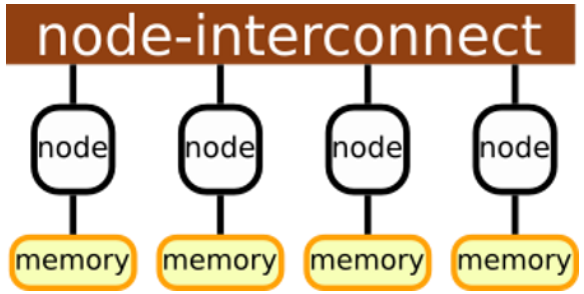
SMP (symmetric multi-processing)



node

ccNUMA (cache-coherent non-uniform ...)

first touch, pinning!



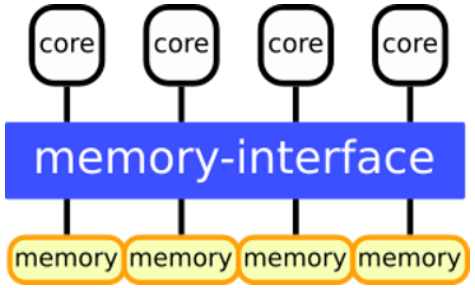
cluster

NUMA (non-uniform memory access)

fast access to own memory only

Hybrid approach

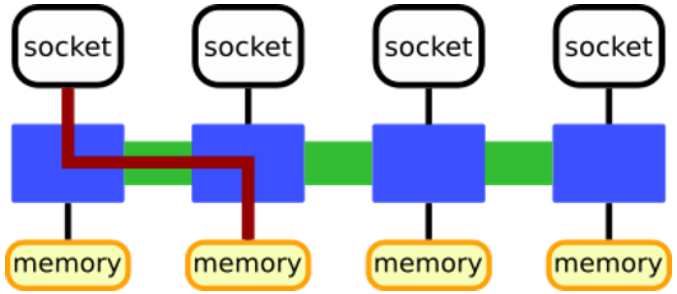
- combination of more approaches (OpenMP, MPI, CUDA,...)
- potential to fully utilize current (future) hardware



socket

UMA (uniform memory access)

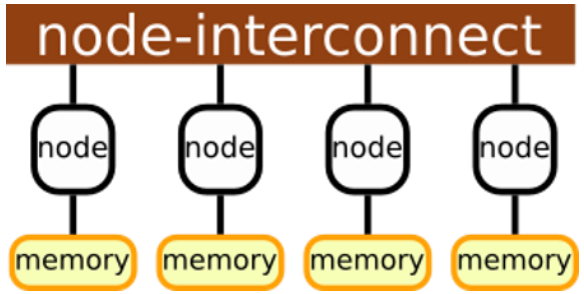
SMP (symmetric multi-processing)



node

ccNUMA (cache-coherent non-uniform ...)

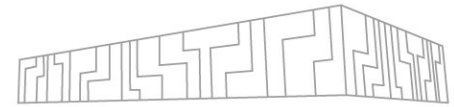
first touch, pinning!



cluster

NUMA (non-uniform memory access)

fast access to own memory only



Sequential (serial) programs:

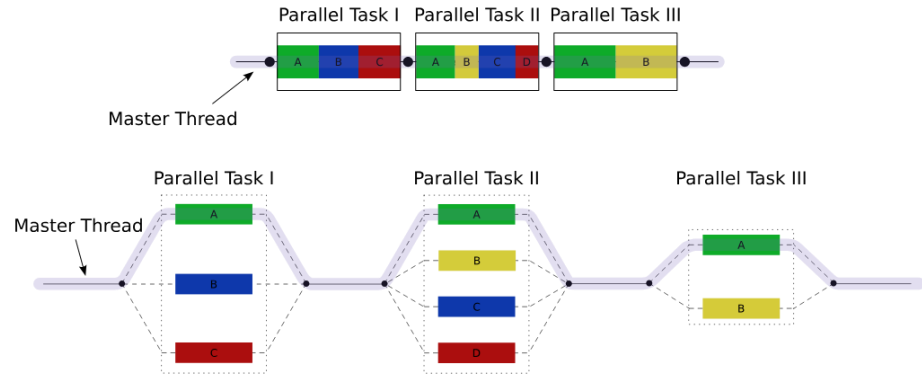
- operate on similar principles everywhere

Parallel programs:

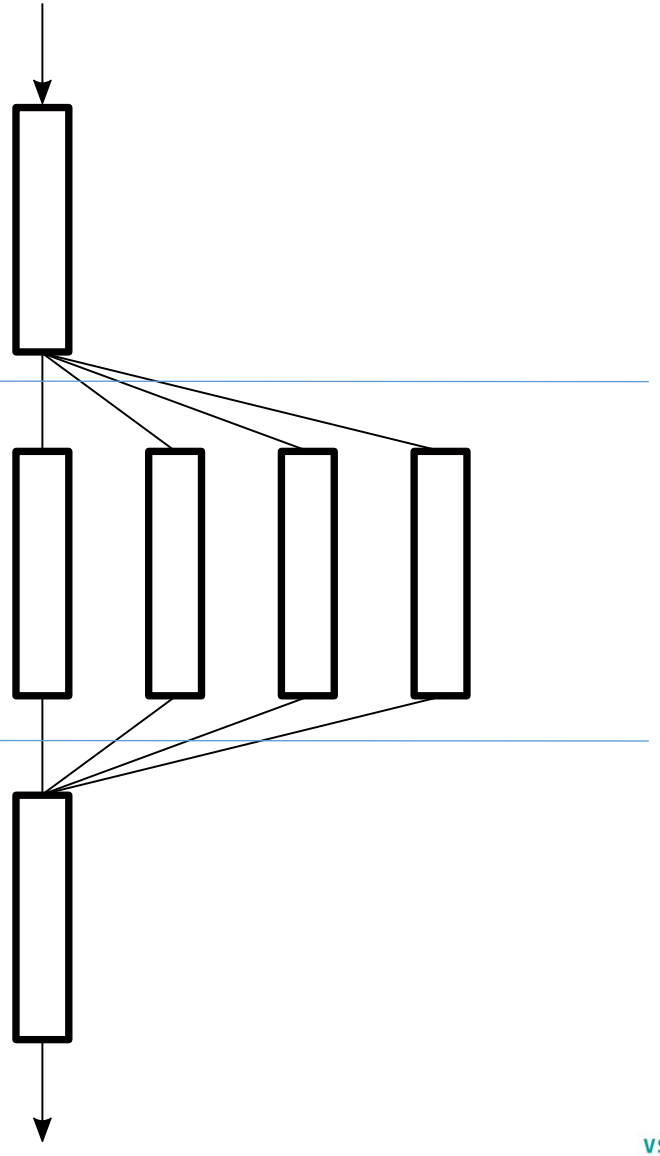
- dependent on the target parallel architecture
- more difficult to write than sequential ones
  - several new classes of software bugs (e.g., race conditions)
  - difficult debugging
- more difficult to run efficiently
  - mapping, pinning

OPENMP

- Open MultiProcessing
- For shared memory parallel processing
- An abstraction above threads
  - Makes parallelizing loops simpler (among other things)
- Primary for C, C++, and Fortran
- Compiler hints (pragmas)
  - `#pragma omp parallel for`
- It is a standard
  - A set or rules – this should do this, this should do that
  - Implemented by all major compilers (GCC, Clang, MSVC*)
- Fork-join model
  - Single initial thread
  - Spawn additional threads to do some work (fork)
  - Wait for all of them to finish (join)

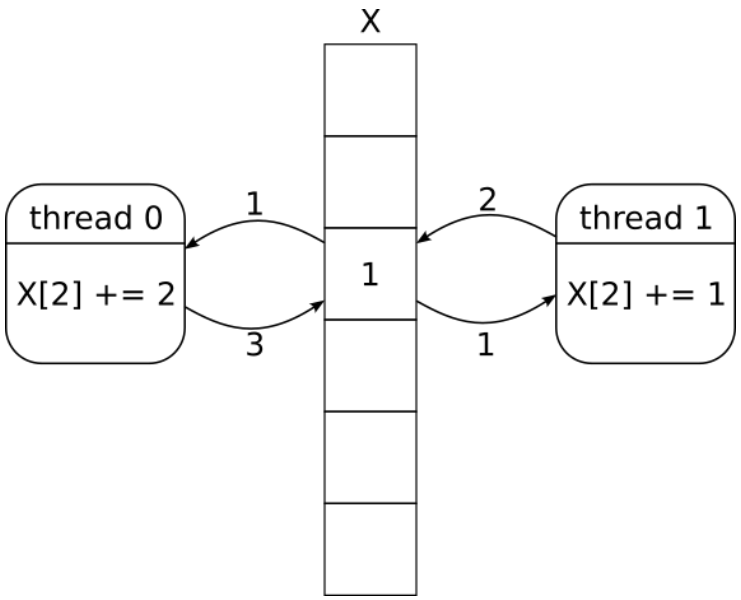


```
#include "omp.h"

int main(int argc, char **argv) {
  int iam = 0, np = 1;

  #pragma omp parallel private(iam, np) /* Parallel region */
  {
    #if defined (_OPENMP)
      np = omp_get_num_threads();
      iam = omp_get_thread_num();
    #endif
    printf("Hello from thread %d out of %d\n", iam, np);
  }
}
```

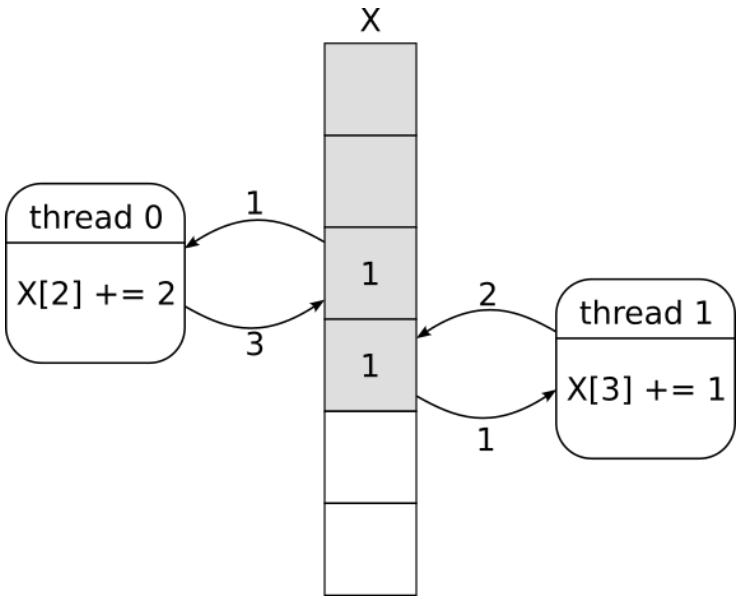


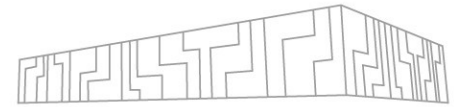
```
$ g++ -fopenmp hello.cpp -o hello
$ OMP_NUM_THREADS=4 ./hello

Hello from thread 2 out of 4
Hello from thread 0 out of 4
Hello from thread 1 out of 4
Hello from thread 3 out of 4
```

- **Race conditions**
  - output is dependent on the detailed timing of concurrent operations
  - e.g., modifying the same variable by two threads
- False sharing
  - significant slowdown in a parallel block of code
  - e.g., modification the same cache line by two threads
- Sequential equivalence
  - strong: bitwise identical results
  - weak: mathematically equivalent
    - not bitwise identical due to the floating-point arithmetic



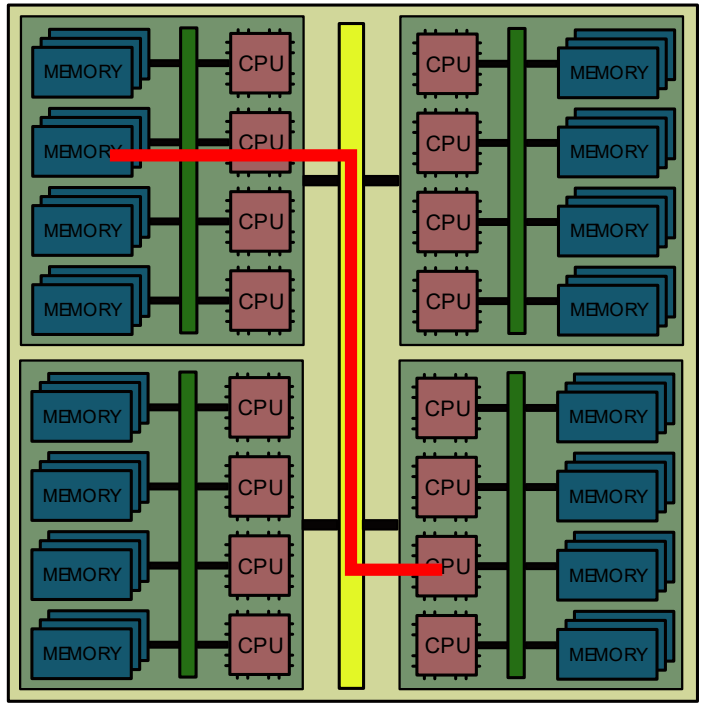
- Race conditions
  - output is dependent on the detailed timing of concurrent operations
  - e.g., modifying the same variable by two threads
- **False sharing**
  - significant slowdown in a parallel block of code
  - e.g., modification the same cache line by two threads
- Sequential equivalence
  - strong: bitwise identical results
  - weak: mathematically equivalent
    - not bitwise identical due to the floating-point arithmetic





- Race conditions
  - output is dependent on the detailed timing of concurrent operations
  - e.g., modifying the same variable by two threads
- False sharing
  - significant slowdown in a parallel block of code
  - e.g., modification the same cache line by two threads
- **Sequential equivalence**
  - strong: bitwise identical results
  - weak: mathematically equivalent
    - not bitwise identical due to the floating-point arithmetic
    - X[0] + X[1] + X[2] + X[3] != (X[0] + X[1]) + (X[2] + X[3])

- Cache coherent distributed memory (ccNUMA)
  - thread requests memory that was firstly touched by a thread from another sockets
  - the same memory should be accessed by the same thread
  - fix threads to a particular CPUs (OMP_PROC_BIND=true ./app)




```
double *vals = new double[rows * cols];

#pragma omp parallel for collapse(2)
for (int r = 0; r < rows; ++r) {
    for (int c = 0; c < cols; ++c) {
        vals[r * cols + c] = 0;
    }
}
```

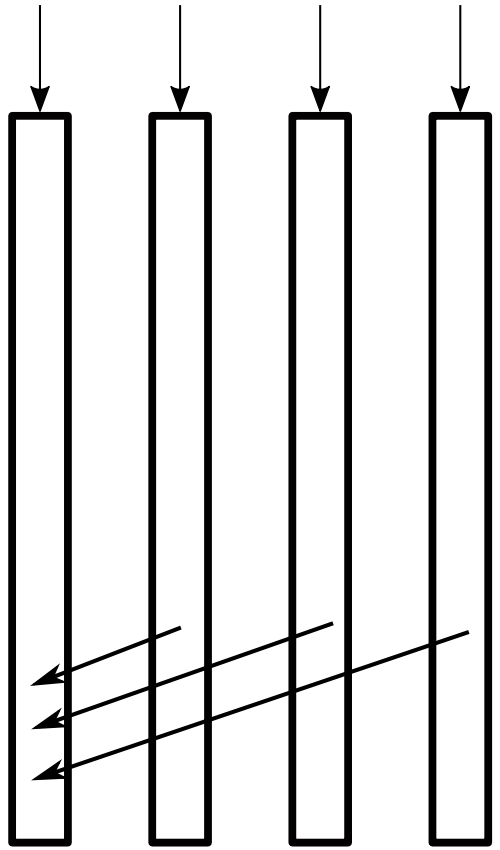
MPI

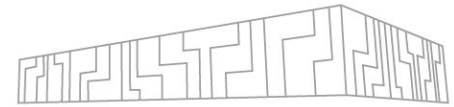
- Message Passing Interface
  - Also designed as a standard
  - C/C++/Fortran library
- Many implementations
  - OpenMPI, MPICH, Intel MPI, …
- The same application runs on multiple compute nodes
  - Or multiple times on a single node
  - mpirun -n 8 ./app
  - srun -n 8 ./app
- Each instance (MPI process) is identified by a unique number – rank
  - Use the rank to split work

- Utility functions
  - MPI_Init, MPI_Comm_rank, MPI_Comm_size, MPI_Finalize, …
- Point-to-point communication
  - MPI_Send, MPI_Recv, …
- Collective communication
  - MPI_Broadcast, MPI_Reduce, MPI_Scatter, MPI_Alltoall, MPI_Barrier, …
- Asynchronous communication (MPI_I*)
- Parallel IO
- Data types

```c
#include "mpi.h"

int main(int argc, char **argv) {
  int rank, size;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  printf("Hello from process %d out of %d\n", rank, size);
  if (rank==0) {
    // recv messages
  } else {
    // send a message
  }
  MPI_Finalize();
}
```

```
$ mpic++ hello.cpp -o hello
$ mpirun -n 4 ./hello

Hello from process 2 out of 4
Hello from process 0 out of 4
Hello from process 1 out of 4
Hello from process 3 out of 4
```

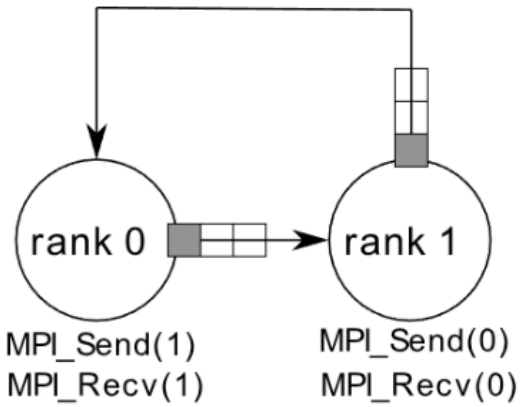




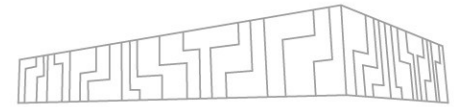
- Race conditions
  - output is dependent on the detailed timing sent/received operations
  - e.g., output is dependent on the order of received messages
- Deadlocks
  - waiting for resources that will never be available

- Sequential equivalence:
  - strong: bitwise identical results
  - weak: mathematically equivalent
    - not bitwise identical due to the floating-point arithmetic
    - X[0] + X[1] + X[2] + X[3] != (X[0] + X[1]) + (X[2] + X[3])

- Race conditions
  - output is dependent on the detailed timing sent/received operations
  - e.g., output is dependent on the order of received messages
- **Deadlocks**
  - waiting for resources that will never be available



- Sequential equivalence:
  - strong: bitwise identical results
  - weak: mathematically equivalent
    - not bitwise identical due to the floating-point arithmetic
    - X[0] + X[1] + X[2] + X[3] != (X[0] + X[1]) + (X[2] + X[3])

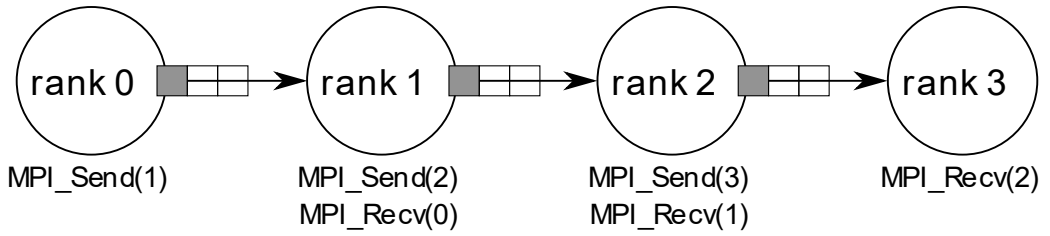


- Non-scalable functions / patterns
  - collectives with input of size *O(#processes), e.g., scatterv*
- Serialization:
  - the order of messages serializes the application
  - e.g., each process must wait to a message from the previous process
- Expensive communication
  - exchanging too much of data
- Performance is not portable
  - MPI assures only portable application!

- Non-scalable functions / patterns
  - collectives with input of size *O(#processes), e.g., scatterv*

- **Serialization**:
  - the order of messages serializes the application
  - e.g., each process must wait to a message from the previous process

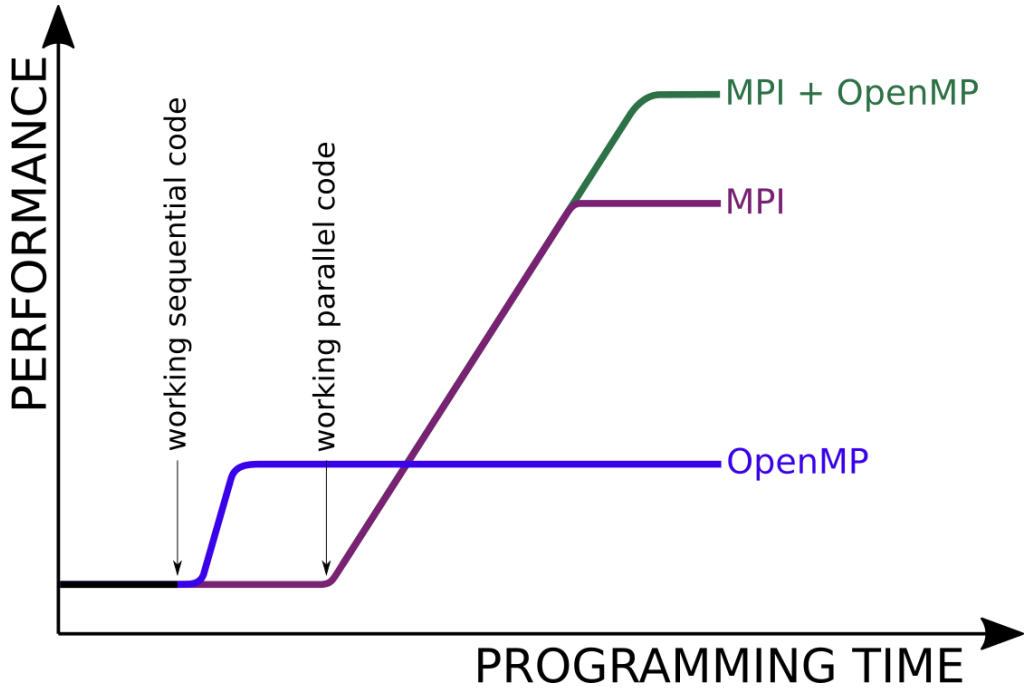
- Expensive communication
  - exchanging too much of data



- Performance is not portable
  - MPI assures only portable application!

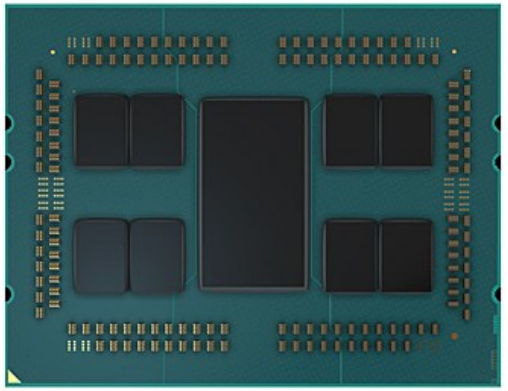
MPI AND OPENMP

- We can combine MPI and OpenMP
- E.g. run 8 MPI processes per node, with 16 OpenMP threads each
- mpirun gets complicated
  - Process mapping, binding
  - Different arguments for OpenMPI, Intel MPI, …
  - It just never works
- Use slurm options (task = MPI process)
  - salloc … -N 2 --ntasks-per-node 16
  - salloc … -N 2 --ntasks 32
  - salloc … --cpus-per-task 8
- Use srun instead of mpirun
  - export OMP_NUM_THREADS=8
  - srun -n 32 ./app
  - **srun -n 32 -c 4 ./app**



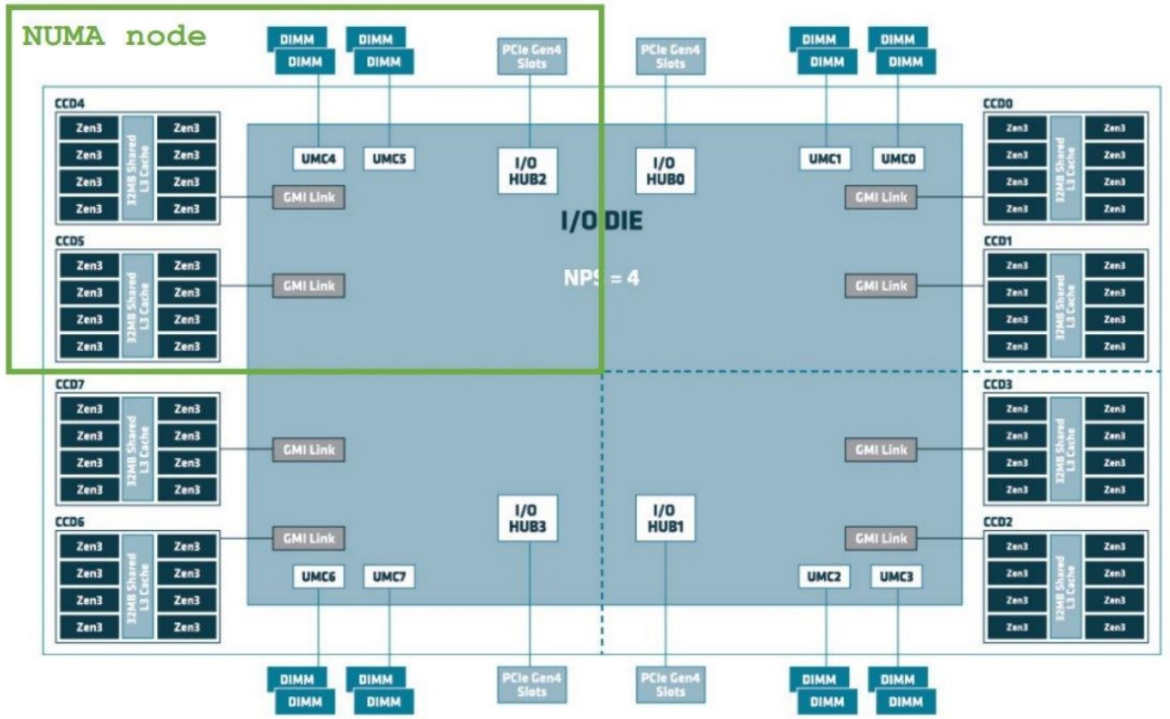
MULTITHREADING PITFALLS – NUMA

- Non-uniform memory access
- Cores have faster access to their own memory
  - Slower access to memory in a different NUMA domain
- On Karolina: 4 NUMA domains per CPU
- Keep data accesses local
- First touch policy

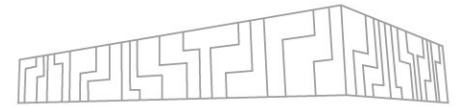


- numactl -H

	0	1	2	3	4	5	6	7
0	10	12	12	12	32	32	32	32
1	12	10	12	12	32	32	32	32
2	12	12	10	12	32	32	32	32
3	12	12	12	10	32	32	32	32
4	32	32	32	32	10	12	12	12
5	32	32	32	32	12	10	12	12
6	32	32	32	32	12	12	10	12
7	32	32	32	32	12	12	12	10



OPTIMIZING MPI+OPENMP APPLICATION RUN



- https://docs.it4i.cz/general/slurm-batch-examples/
- Compute bound app => use all cores
- Memory bound app => use one or two cores per memory channel

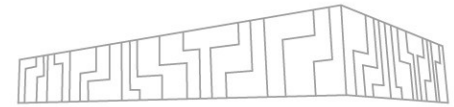
- salloc -p qcpu_exp -N 1
- export OMP_NUM_THREADS=1

- srun -n 8 ./app 39.66s
- srun -n 16 ./app 17.46s
- srun -n 32 ./app 11.87s
- srun -n 64 ./app 7.31s
- **srun -n 128 ./app 5.56s**

Is 128 the best possible settings?

What about mapping and pinning?

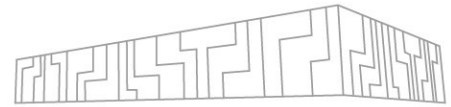
OPTIMIZING MPI+OPENMP APPLICATION RUN



- https://docs.it4i.cz/general/slurm-batch-examples/
- Compute bound app => use all cores
- Memory bound app => use one or two cores per memory channel

- salloc -p qcpu_exp -N 1
- export OMP_NUM_THREADS=1

srun -n 8 ./app	39.66s	srun -n 8 -c 16 ./app	7.11s
srun -n 16 ./app	17.46s	srun -n 16 -c 8 ./app	5.21s
srun -n 32 ./app	11.87s	**srun -n 32 -c 4 ./app**	**5.08s**
srun -n 64 ./app	7.31s	srun -n 64 -c 2 ./app	5.31s
srun -n 128 ./app	5.56s	srun -n 128 -c 1 ./app	5.56s



Memory bound application

- Number of MPI processes / thread equal to memory channels
- Correct pinning to NUMA domains

Compute bound application

- As many MPI processes / threads as possible
- pinning to avoid migration

Your application

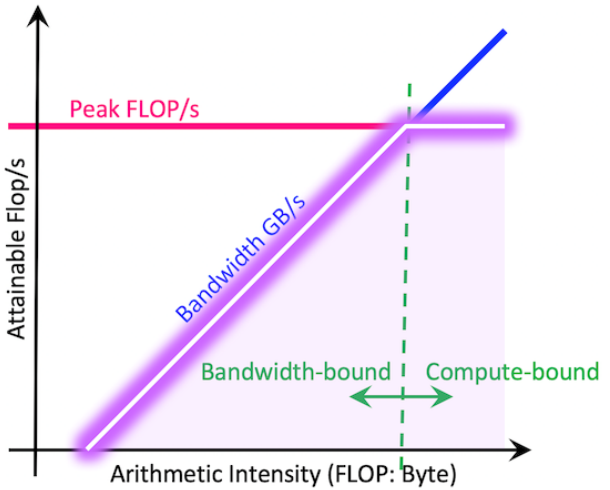
- Test performance for different number of cores per node:
  - 16, 32, 64, 128 cores per node

- Test different mapping / pinning options
  - close, spread

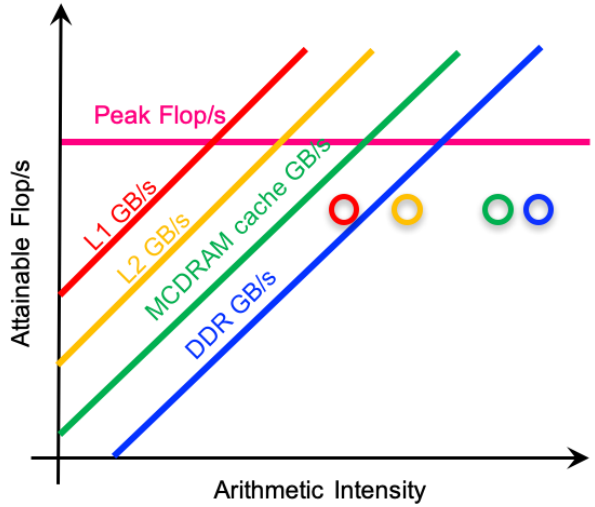
THEORY OF PARALLEL COMPUTING

Memory/compute bound apps
Parallel scalability

MEMORY AND COMPUTE BOUND KERNELS

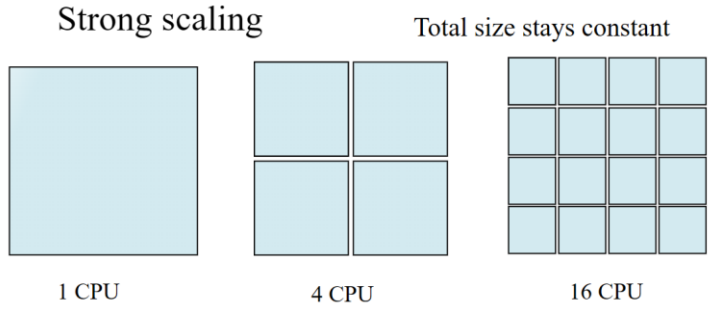


- Essentially, a program does two main things
  - Load data from and store data to memory
  - Compute inside the core
- Arithmetic intensity
  - Number of flops performed per each byte loaded from main memory
- High arithmetic intensity => compute-bound kernel
  - A lot of flops performed for little data loaded
  - Highest performance achievable (closest to theoretical peak)
  - Hard to achieve – need to utilize caches and SIMD correctly
  - Matrix-matrix multiplication, …
- Low arithmetic intensity => memory-bound kernel
  - Only a few flops performed, and a lot of data transferred
  - AXPY, matrix-vector multiplication, …

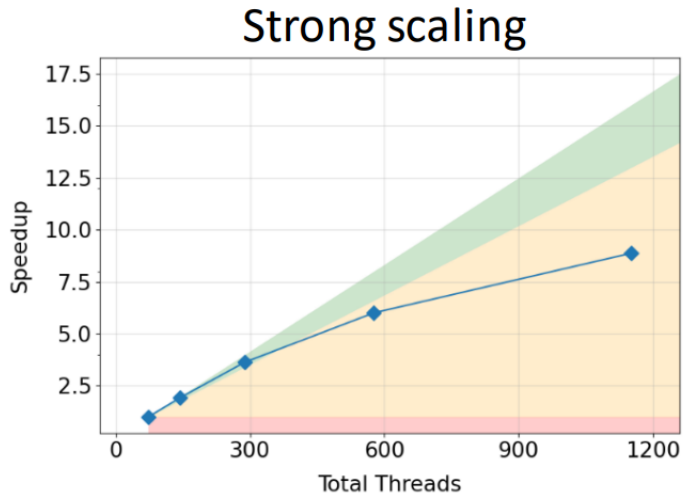


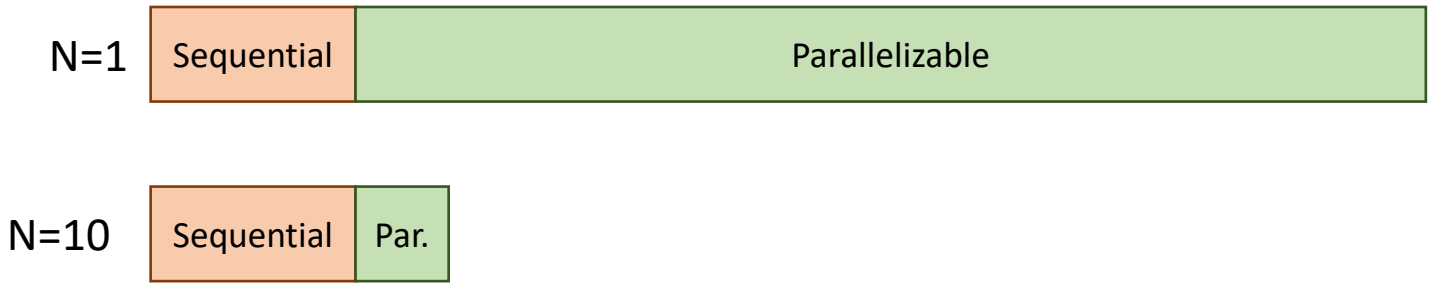
PARALLEL SCALABILITY – STRONG SCALING

- Solve the same problem with more computational resources faster
- Expectation: linear scaling, 2x more resources => half the time
- Reality: not as nice



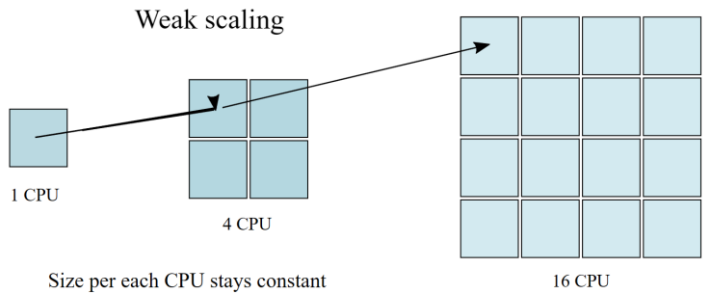
- Amdahl's law:
  - A program always has non-scalable (sequential) parts
    - Initialization, global communication
  - Use a gazillion cores to speed up the parallel part
  - You still must execute the non-scalable part
  - This limits the maximum achievable speedup



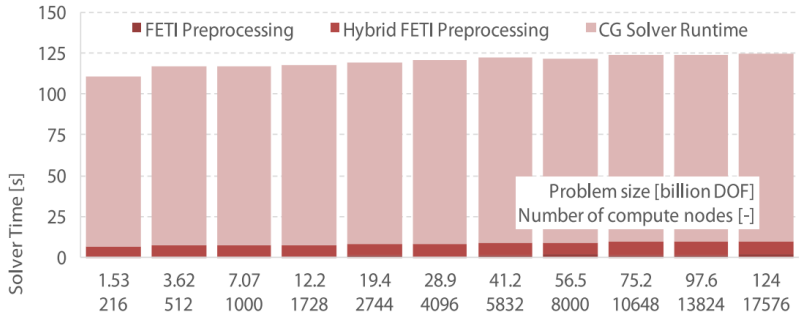


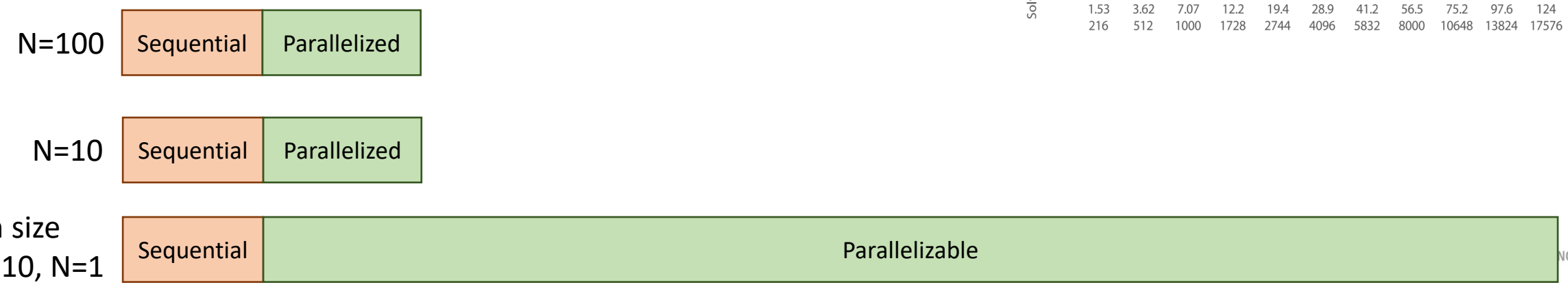
PARALLEL SCALABILITY – WEAK SCALING

- Use more computational resources to solve a larger problem
  - Time should stay constant



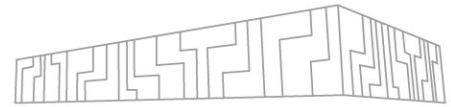
- Gustafson's law
  - Speedup is not limited by sequential part
  - Sequential part gets proportionally smaller with larger problem sizes
  - If the parallel part of the program was sequentialized, it would take N-times longer
  - Plus the time of the sequential part





THE END

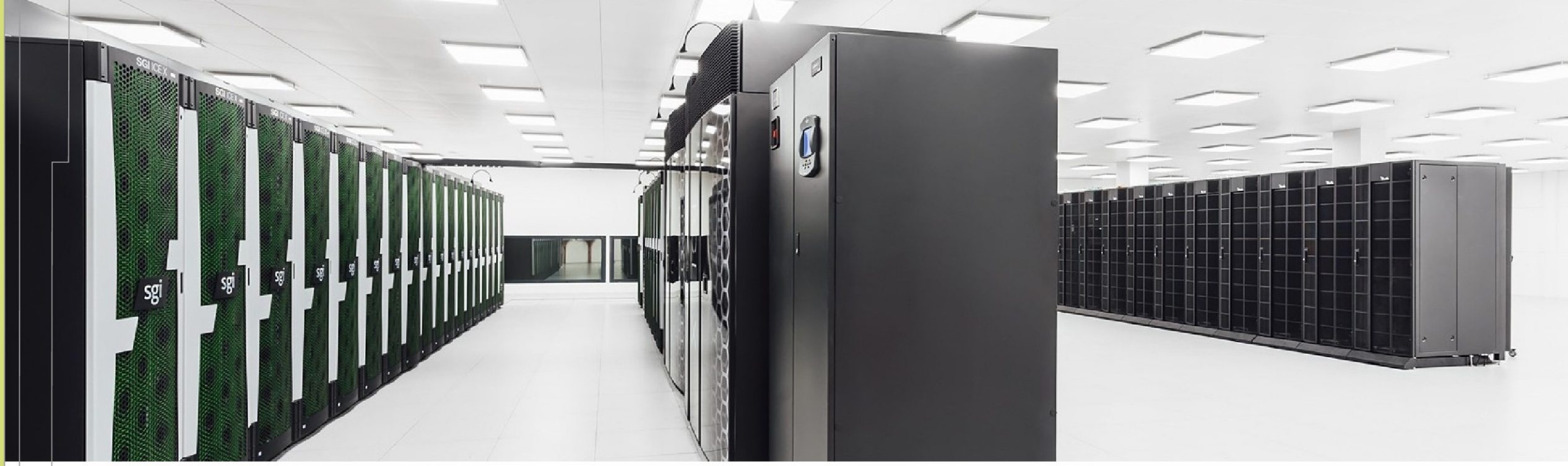
YOUR FRIENDS



- IT4I Documentation: https://docs.it4i.cz/
- Cluster utilization – RSweb: https://extranet.it4i.cz/rsweb
- Cluster utilization – Grafana: https://extranet.it4i.cz/grafana
- Supercomputing services portal: https://scs.it4i.cz/

- Introduction to HPC course (approx. every June)
  - 2026 materials: https://events.it4i.cz/event/397/

- Support webpage for tickets: https://support.it4i.cz/
- Support email: support@it4i.cz
- High-level support available (HLST) – help with more advanced stuff
  - Through the EPICURE project



Ondrej Meca
ondrej.meca@vsb.cz

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

IT4Innovations National Supercomputing Center
VSB – Technical University of Ostrava
Studentská 6231/1B
708 00 Ostrava-Poruba, Czech Republic
www.it4i.cz