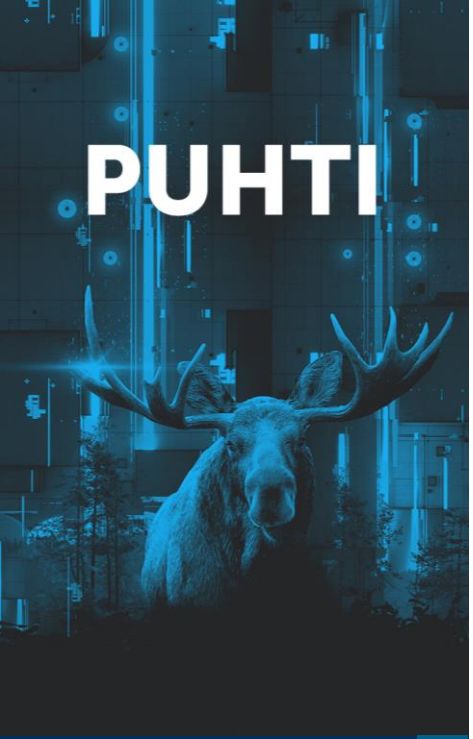
CSC

ICT Solutions for Brilliant Minds

# Spatial analysis with Python in CSC supercomputer Puhti

Zoom, 10.03.2022

Kylli Ek, Samantha Wittke




## Timetable

12:30-13:30 Introduction spatial analyis with Python in CSC computing environments

13:30-13.45 Break

13:45-15:15 Hands-on exercises in Puhti

Non-profit state organization with special tasks

Turn over in 2021

**60**M€

Headquarters in Espoo, datacenter in Kajaani

**Owned by state (70%)**

and all Finnish higher education institutions **(30%)**

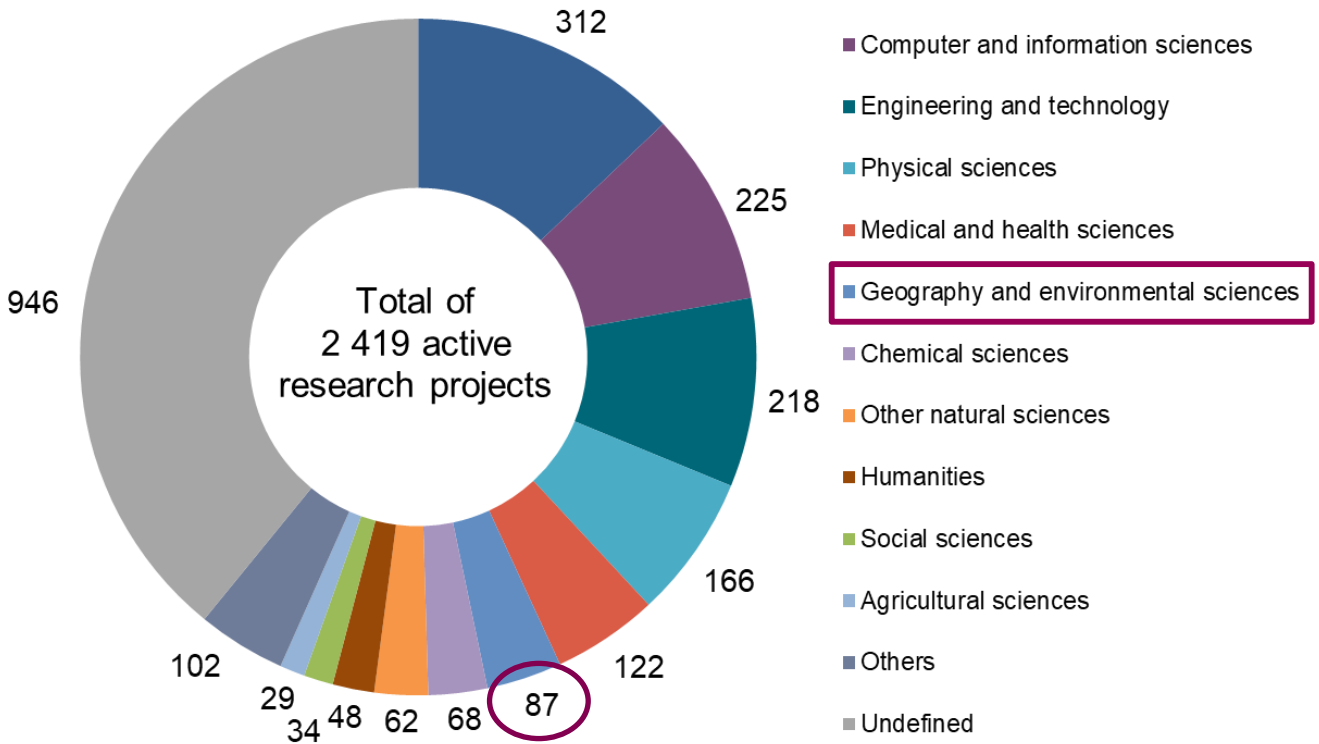
Circa

**518**

employees in 2022

You are not alone!

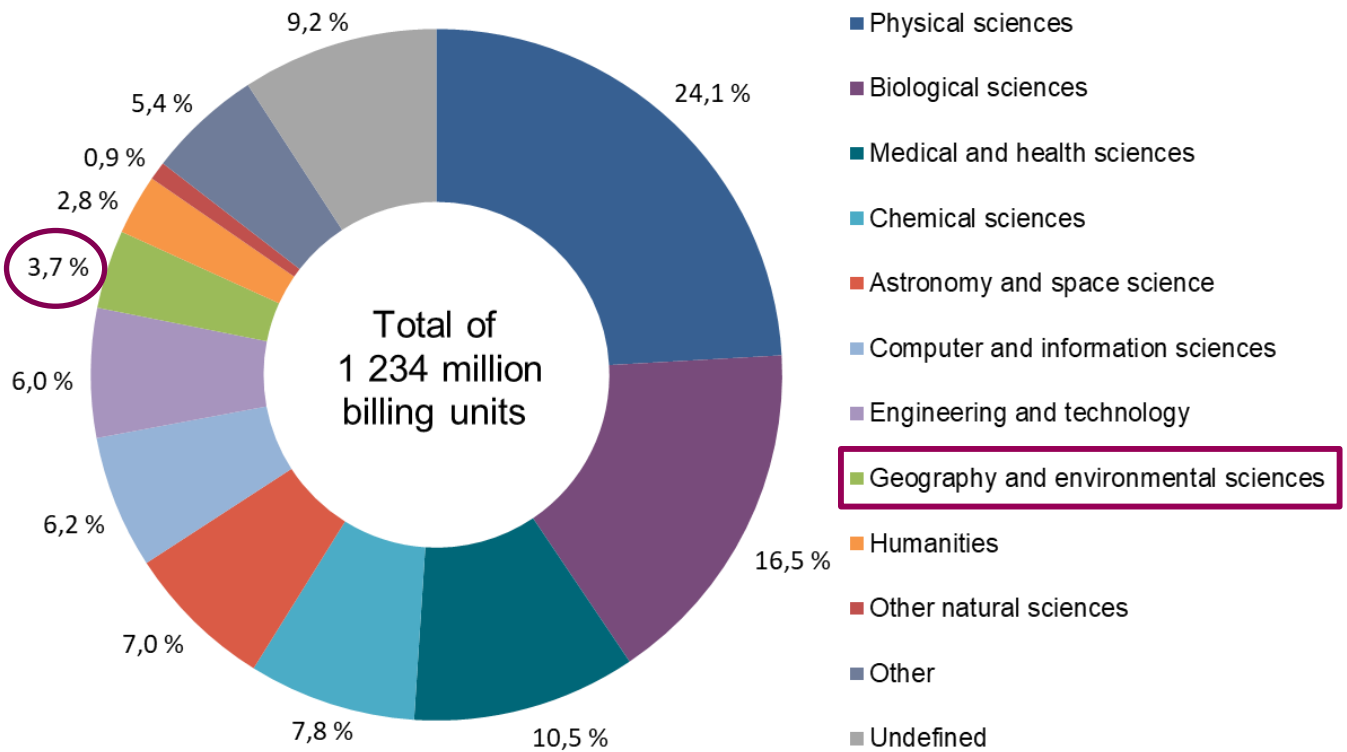
## Active research projects by science area in 2020 (all servers)

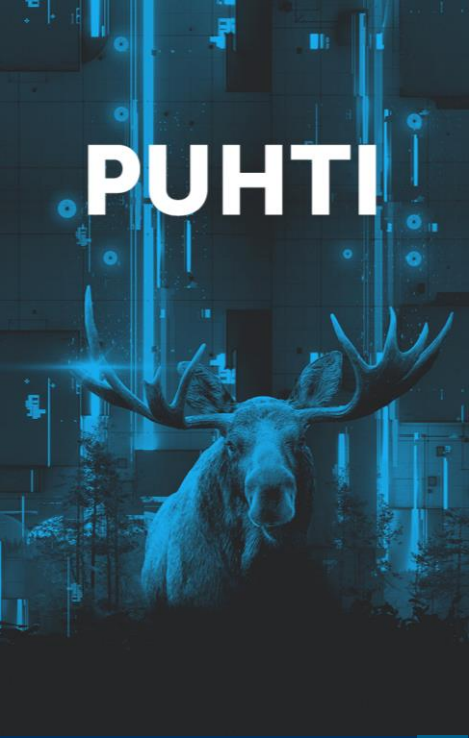



An active project contains billing unit usage during year 2020.

You are not alone!

## Computing resource usage by science area in 2020
(incl. all computational and storage use)



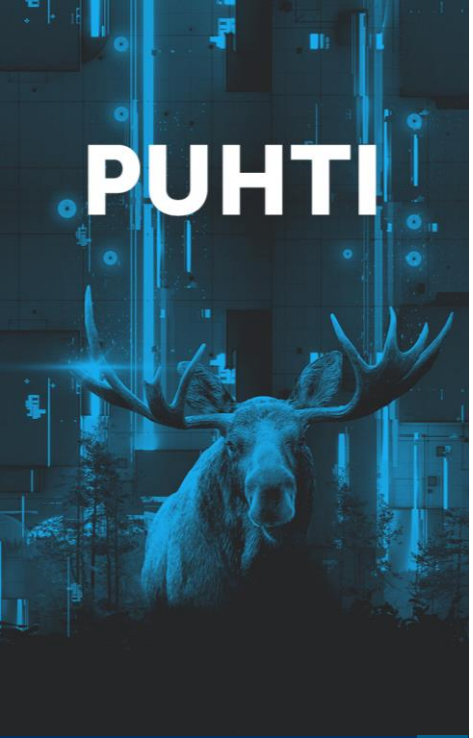



# Why using CSC resources?

- Long computing time (> 2 hours)
- High memory usage (> 8 GB)
- Large datasets (> 50 GB)
- Outsource computations, keep own computer free
- Software readily available
- Computing support

-> Puhti

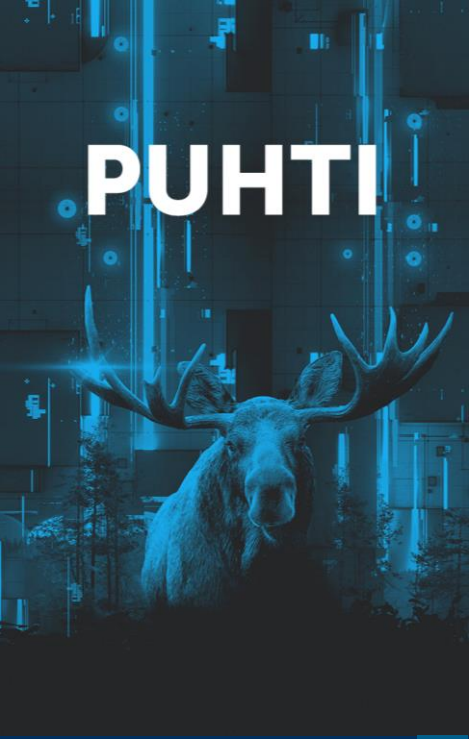
- Server needs -> cPouta
- Course computers (same setup) -> Notebooks
- It's free for open science and teaching for Finnish university and state research institute users




| | Puhti HPC | cPouta cloud |
|---|---|---|
| System | Supercomputer | Virtual machine cloud |
| Software | Pre-installed software + user-installed software | User-installed software |
| Data | Main Finnish datasets | - |
| Use cases | Run demanding analyses with numerous CPUs or GPUs | Setup your own virtual machine and environment |
| Max per job / VM / container | **4000** CPUs / **80** GPUs<br>**1500** GB memory | **48** CPUs / **4** GPUs<br>**240** GB memory |

Average laptop:
**4-8 CPUs** /
**8-32 GB memory**

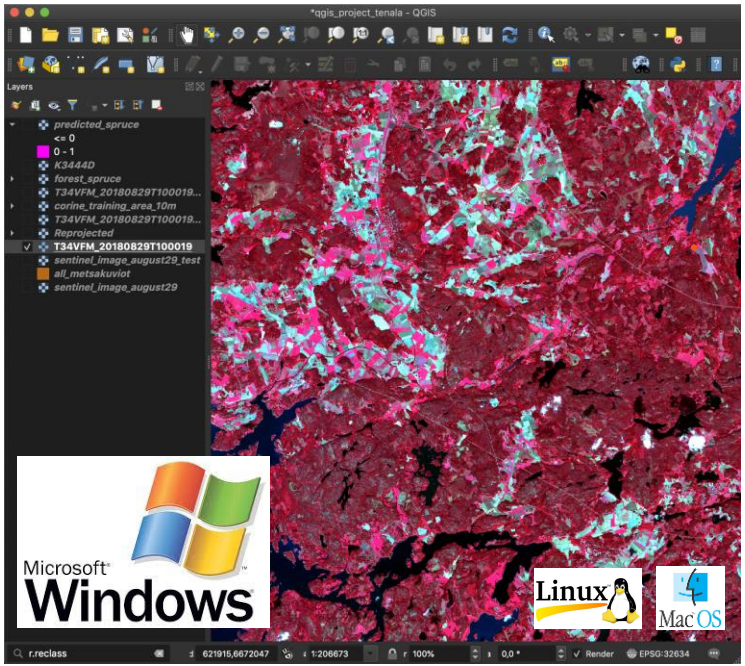
# PUHTI SUPERCOMUPUTER



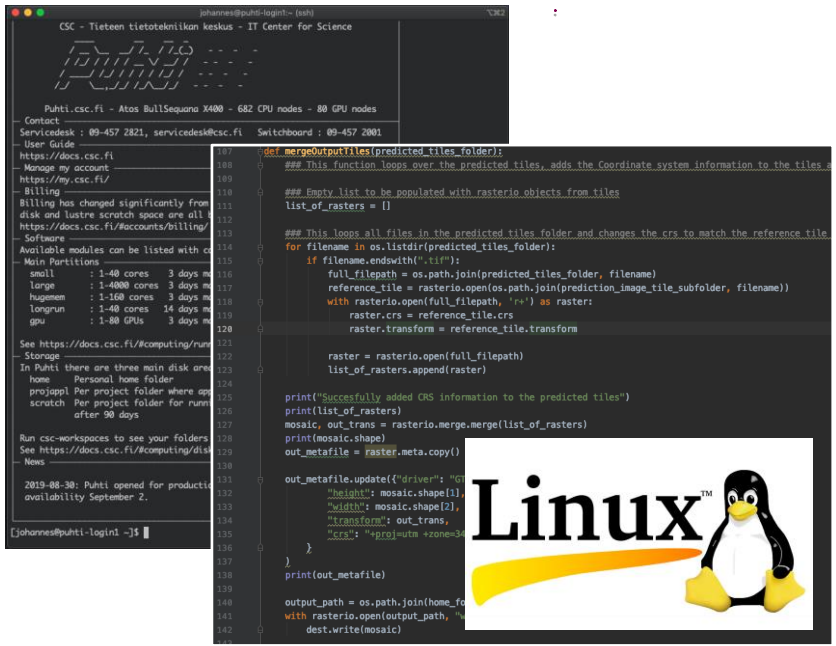

# Supercomputers in Kajaani

- **Puhti** (national HPC, CSC):
  - 27 280 CPU cores and 320 GPUs
  - **Wide stack of geospatial software**
- **Mahti** (national HPC, CSC):
  - **182 784 CPU** cores and 96 GPUs
  - No geospatial software
- **LUMI** (EuroHPC, international LUMI consortium)
  - 196 608 CPU cores and **10 240 GPU**s
  - Software stack unclear
  - 20% for companies
  - CPU side in use, GPU coming later this year

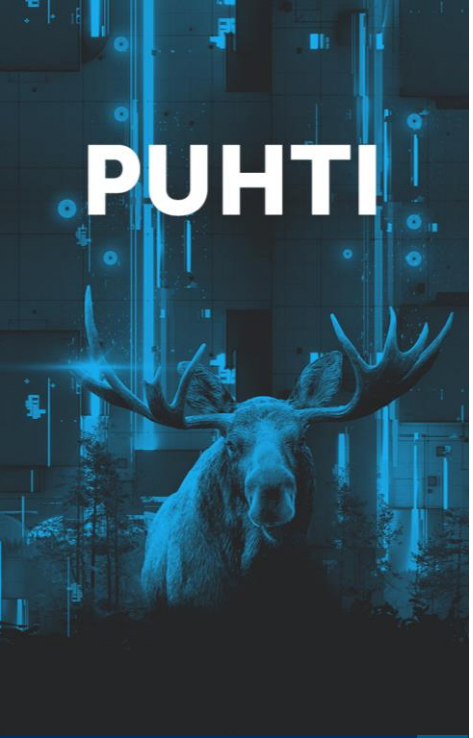
# Key to geocomputing in HPC: GUI → Scripts




Graphical user interfaces: ArcGIS, QGIS,...

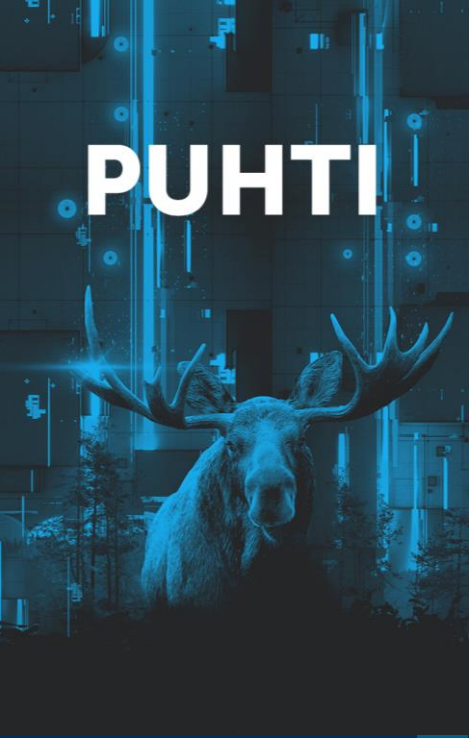



Scripts: Python, R, shell, Matlab, …




# Keep it real

- Single core of Puhti = ~speed of laptop
- But: Puhti has **many cores**
- … and **more memory** and **faster input-output (I/O)**
    - Running single core scripts on Puhti does not make them faster
    - Speedup: multi-core parallel processes
    - … or optimize your script

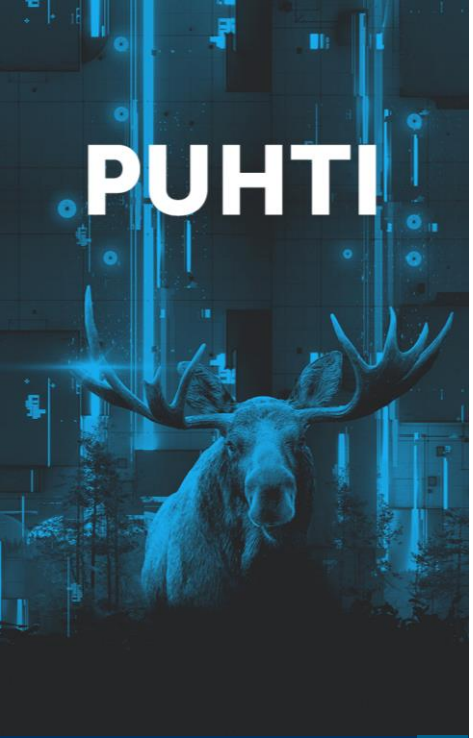



# Geospatial Software in Puhti

geo portti
Finnish Geospatial Research and Education Hub

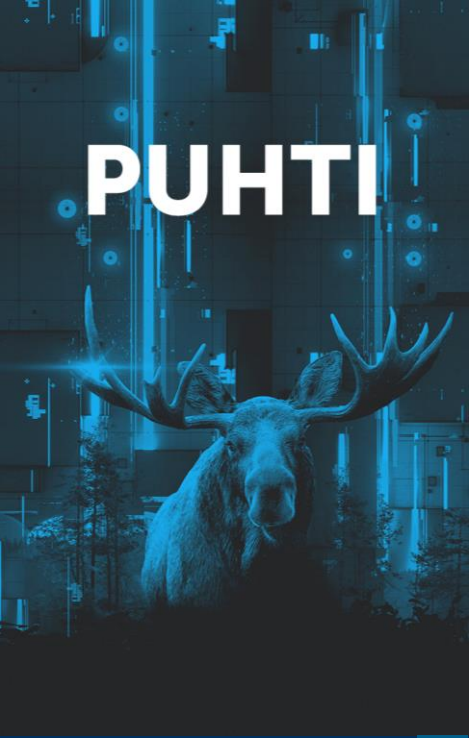
- **ArcGIS Python API**
- FORCE & SPLITS
- **GDAL**/OGR
- **GRASS GIS**
- LasTools
- MatLab / Octave
- **Mapnik**
- OpenDroneMap
- **Orfeo Toolbox**
- **PCL**
- **PDAL**
- **Python geospatial packages: geoconda**
- QGIS, for viewing point clouds
- R geospatial packages
- SagaGIS
- **SNAP**, Sen2cor
- WhiteboxTools
- Zonation
- **Deep learning: Keras, PyTorch**

- **Something missing?**
  - Ask us :)
    servicedesk@csc.fi




# Parallel computing with geospatial software

- **Some tools have at least some support built-in**
  - lidR, OpenDroneMap, Whiteboxtools (?)
  - SAGA GIS, GRASS GIS (some parts)
- **Parallel libraries for Python**
  - dask
  - multiprocessing
  - Joblib
  - …
- **Parallel libraries for R**
  - future
  - snow
  - foreach
  - …



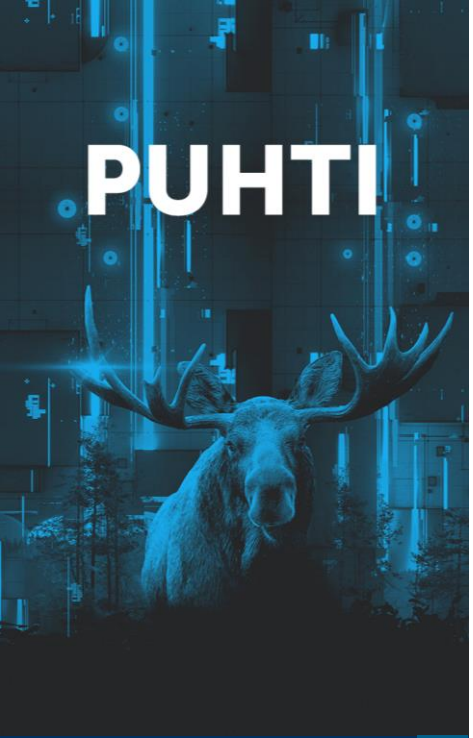

# Example scripts

geo portti | Finnish Geospatial Research and Education Hub

https://github.com/csc-training/geocomputing

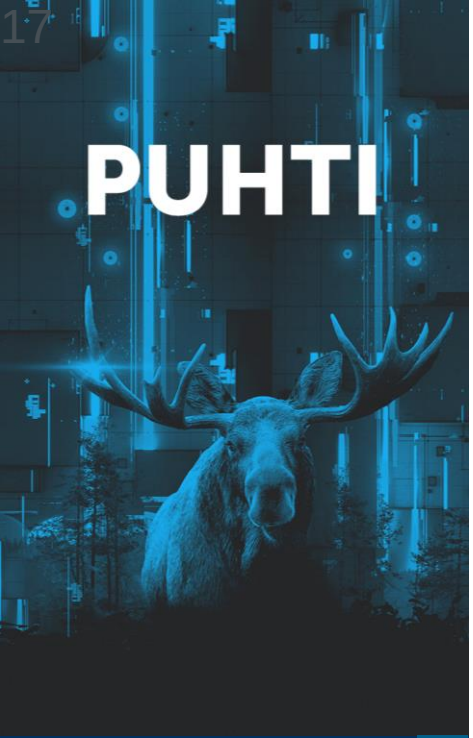
- Batch job files
- Parallel examples for **R** and **Python**
  - Different parallelization libraries
  - Array jobs as well as parallel jobs
- Examples for **Allas** data transfers with **R** and **Python**
- Sentinel image download example (python)
- **SNAP** array job example

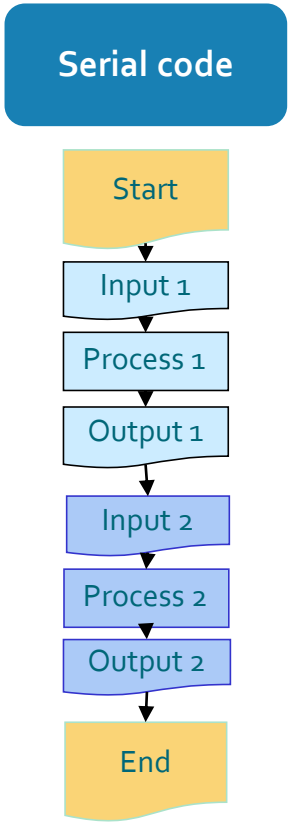
# Running Python in parallel — principles and practice

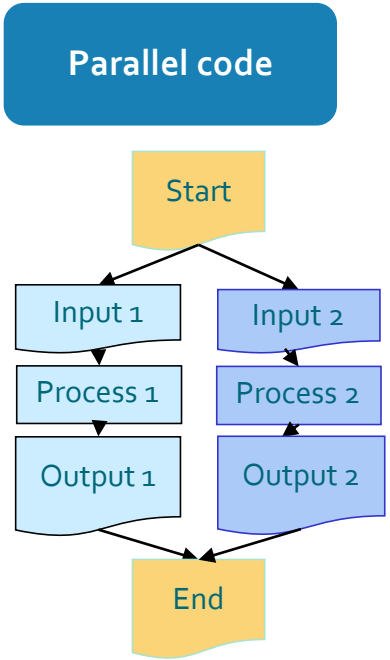


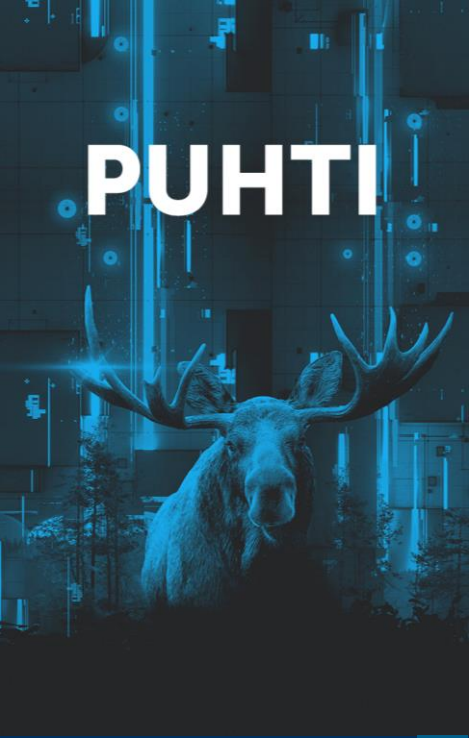

## "My Python code is slow… what can be done?"

- Try to understand which part of the code takes time and why
  - Use **`time`** or **`timeit`** packages
- Always be suspicious of **`for`**-loops!
- Going parallel *may* help
- Unfortunately, increasing the number of cores will *never* decrease the time needed in the same proportion


PUHTI

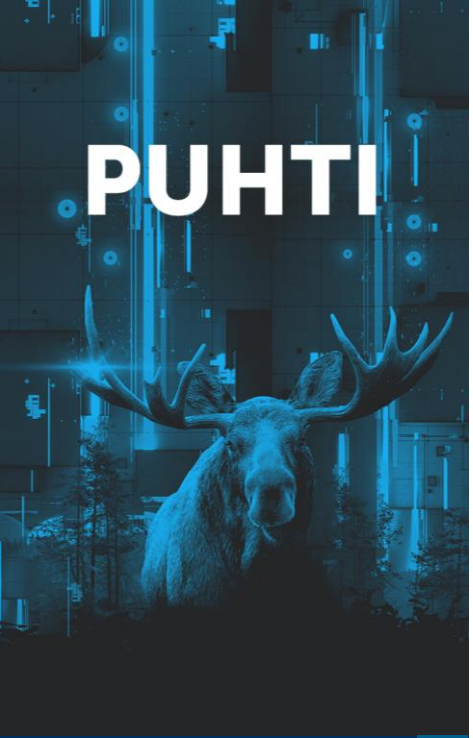

Serial code
Start
Input 1
Process 1
Output 1
Input 2
Process 2
Output 2
End


Parallel code
Start
Input 1
Input 2
Process 1
Process 2
Output 1
Output 2
End



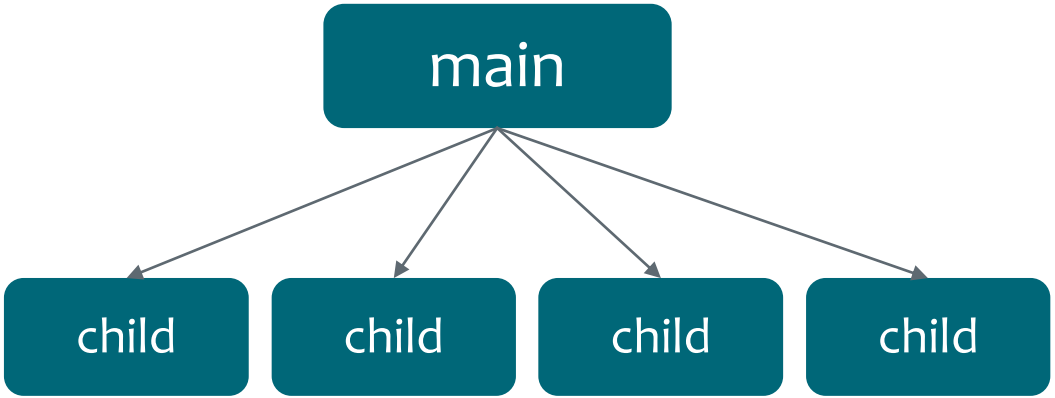

# Parallelization in GIS

- Easiest parts for parallelization:
  - for loops
  - map
- Logical level
  - input files / map sheets / chunks of vector data,
  - scenarios, variables, time periods
- In many cases if using map sheets the borders need special care.




## multiprocessing

- Parallezation package in Python
- Alternatives: joblib, dask



# multiprocessing

## function and input

```
def slow_function(i):
    time.sleep(3)
    return(i)

input = range(1, 7)
```

SERIAL

## for loop

```
a = []
for i in input:
 a.append(slow_function(i))
```

## map

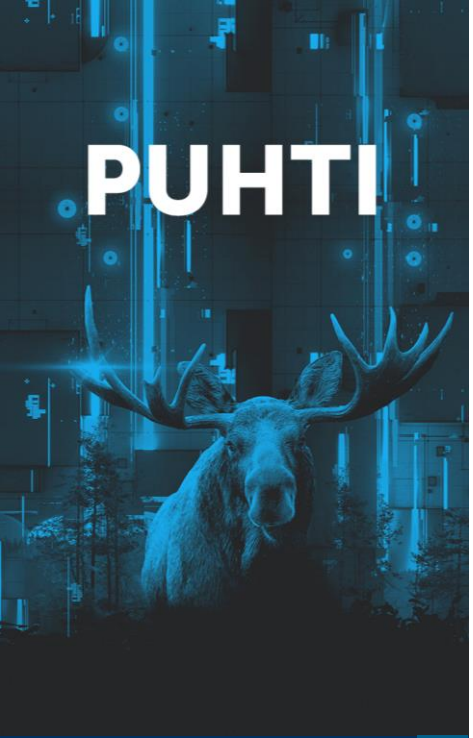
```
a = list(map(slow_function, input))
```

## multiprocessing

PARALLEL

```
from multiprocessing import Pool
pool = Pool(os.sched_getaffinity(0))
a = pool.map(slow_function, input)
```

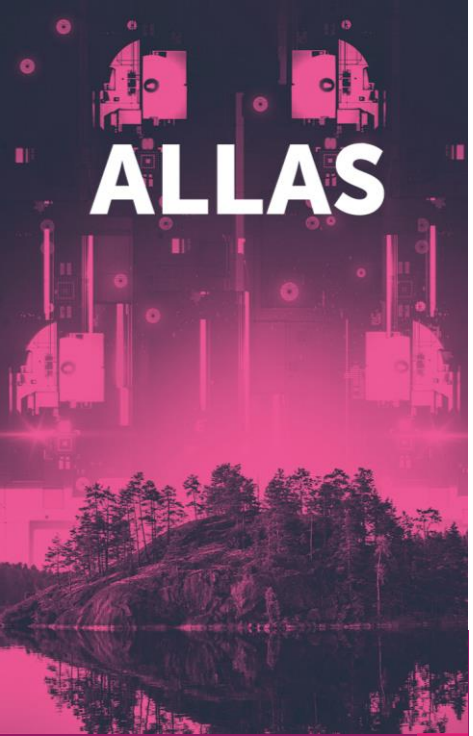
# DATA




# Directories

| Directory | Owner | Capacity | Num. files | Path | Cleaning |
|---|---|---|---|---|---|
| home ($HOME) | Personal | 10 GiB | 100 000 | /users/<user-name> | No |
| projappl | Project | 50 GiB | 100 000 | /projappl/<project> | No |
| scratch | Project | 1 TiB | 1 000 000 | /scratch/<project> | Yes **– 90 days** |

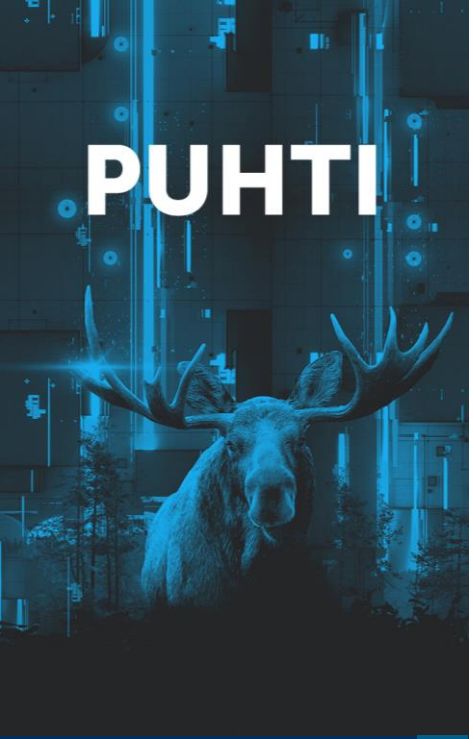
- The quota of ***scratch*** and ***projappl*** directories can be increased
- **No back-up**
- More information: https://docs.csc.fi/computing/disk/




# Allas object storage

- Central data service for all academic projects
- Up to 200 Tb of space for a project
- Access from Puhti, cPouta, personal computer or any other place
- GDAL has good support for reading directly from object storage
- Option to make data public

See also: CSC webinar - **Allas and Geospatial data**
https://www.youtube.com/watch?v=mnFXe2-dJ_g




# Geospatial data in Puhti

geo portti | Finnish Geospatial Research and Education Hub

- Hosts large commonly used datasets with open license
- Removes transfer bottleneck
- Located at: **/appl/data/geo/**
- All Puhti users have **read** access

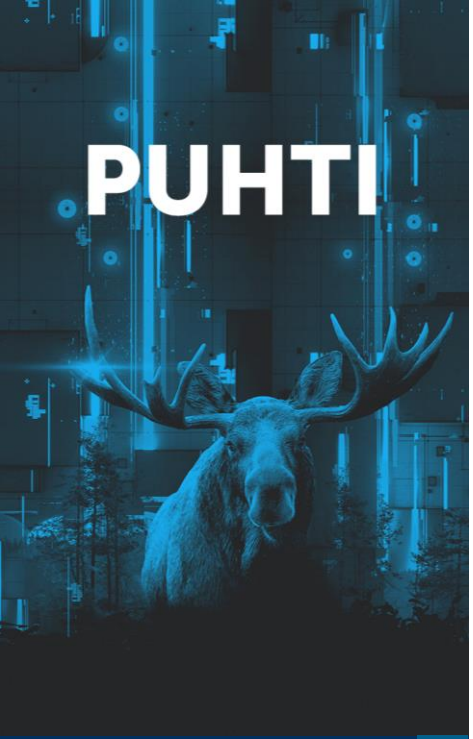
Datasets available (~ **13TB**)

- Paituli data +
  - SYKE open datasets
  - LUKE Multi-source national forest inventory
  - NLS Virtual rasters for DEMs
  - Sentinel and Landsat mosaics

- Finnish Digital and Population Data Services Agency
- Finnish Food Agency
- Finnish Meteorological Institute (FMI)
- Finnish Transport Infrastructure Agency, Digiroad
- Institute for the Languages of Finland (KOTUS)
- Karelia University of Applied Sciences
- Latuviitta
- National Land Survey (MML)
- Natural resource institute Finland (LUKE)
- Statistics Finland
- University of Helsinki, Digital Geography Lab

https://docs.csc.fi/data/datasets/spatial-data-in-csc-computing-env/

Want more? - contact us at **servicedesk@csc.fi**

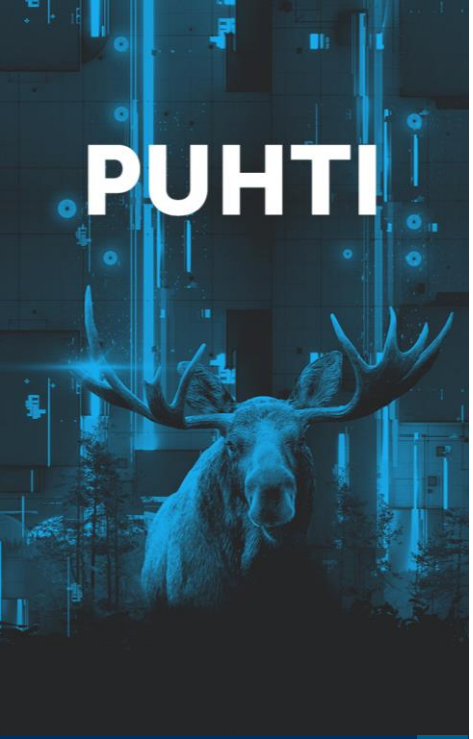



# Virtual rasters in Puhti

geo portti
Finnish Geospatial Research and Education Hub

- Ready made virtual rasters for 2m and 10m DEMs
- Multiple files usable as if they were a single file
- XML pointing to actual raster files
- External overviews and xml headers
- Possible to have all data in Allas and only virtual raster in Puhti

Python script to create your own virtual raster available

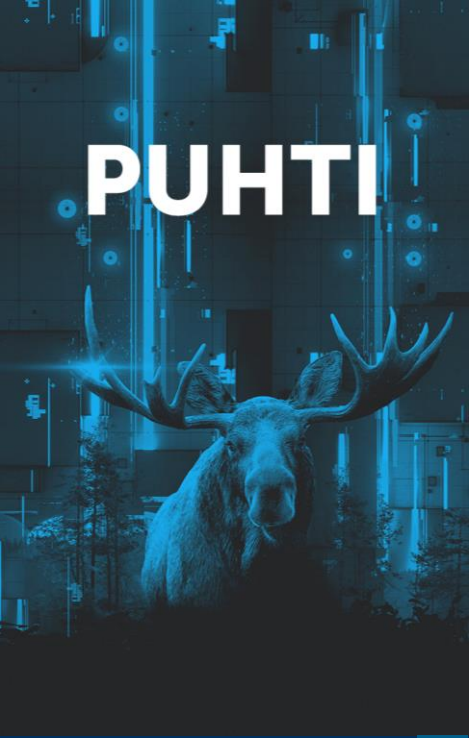



## Data: tips for moving files

- When downloading from external services, download directly to CSC, not via your local PC
- When lot of/big files use :
  - command-line tools: rsync, `wget, curl`
  - `Python/R packages for moving files`

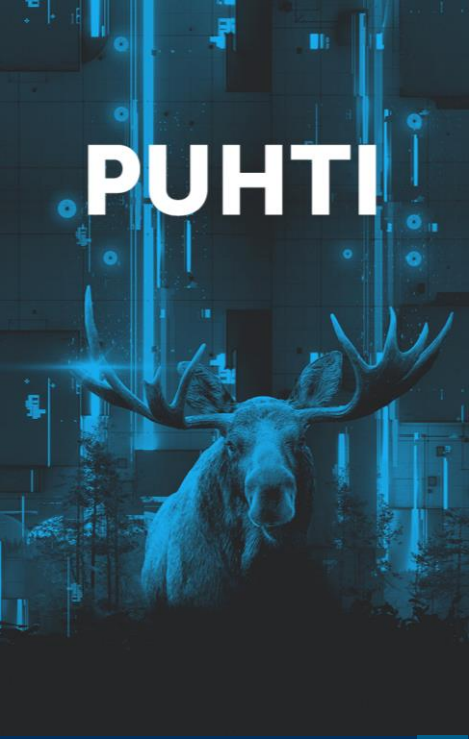
https://docs.csc.fi/data/moving/scp/

# GETTING STARTED




# Set up for Puhti, administrative

- **Check your eligibility** for using CSC resources: https://research.csc.fi/free-of-charge-use-cases
- Get a **user account**: https://docs.csc.fi/accounts/how-to-create-new-user-account/
- Create or let your PI **create a new project:** https://docs.csc.fi/accounts/how-to-create-new-project/
- **Add members** to your project: https://docs.csc.fi/accounts/how-to-add-members-to-project/
- **Add services** to your project: https://docs.csc.fi/accounts/how-to-add-service-access-for-project/

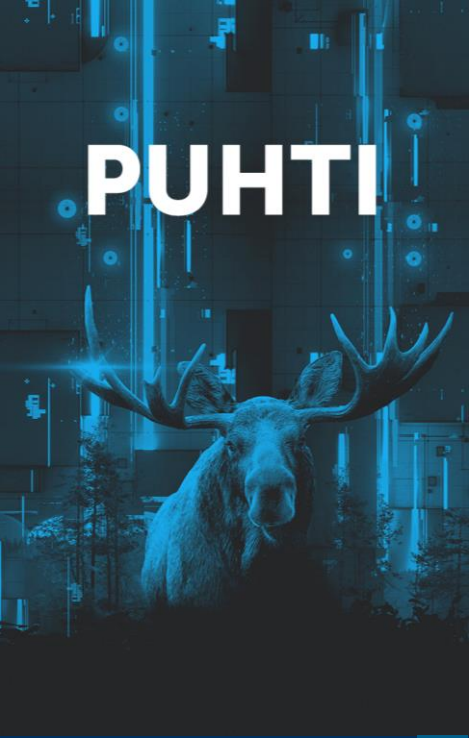



# Set up for Puhti, Billing Units (BU)

- Each project has a certain amount of so-called Billing Units (BU)

- Using CSC resources consumes Bus, based on:
  - Computation time and number of resources (CPU, memory)
  - GPUs in Puhti (**a lot of BUs!**)
  - Increased storage quota in Puhti
  - Used storage in Allas
  - Cpouta uptime (flavor dependent)

You can apply for more BUs in **my.csc.fi** by providing a description what what you are doing




# Set up for Puhti, tools

Minimum:

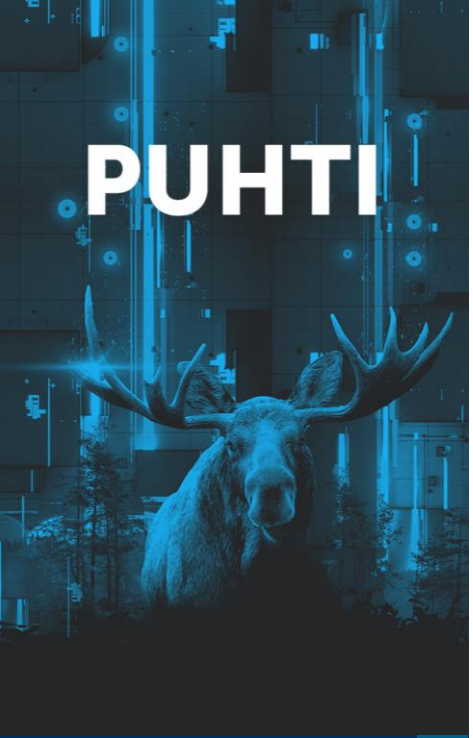
- web-browser for **Puhti web interface**

Recommended:

- **Moving data:**
  - FileZilla/WinSCP, rsync, wget, curl

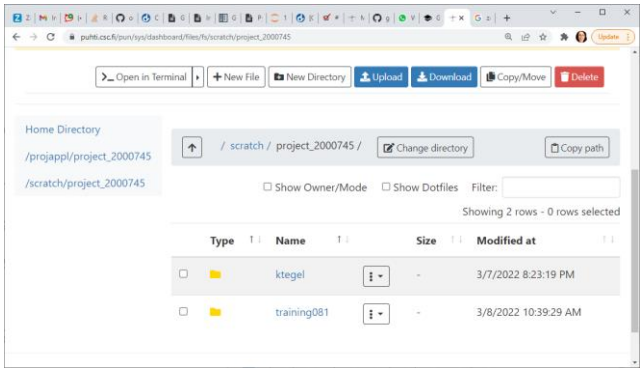
Optional:

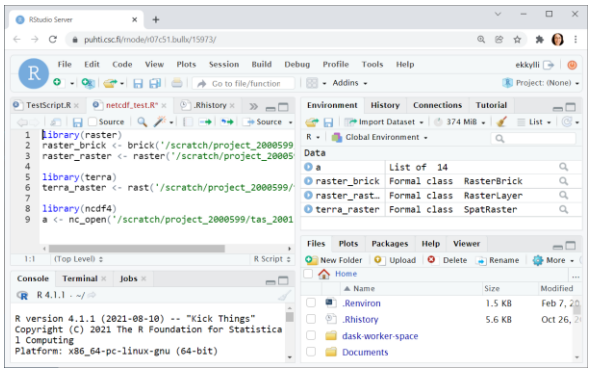
- **Connecting:**
  - Terminal in Linux/Mac
  - PowerShell/Putty/MobaXterm in Windows

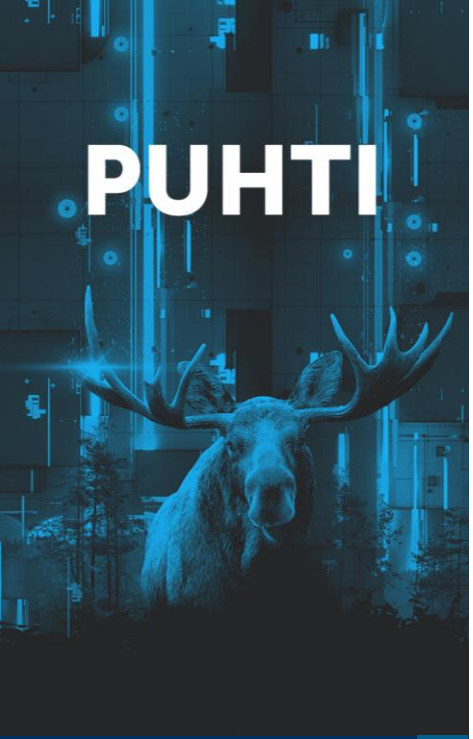


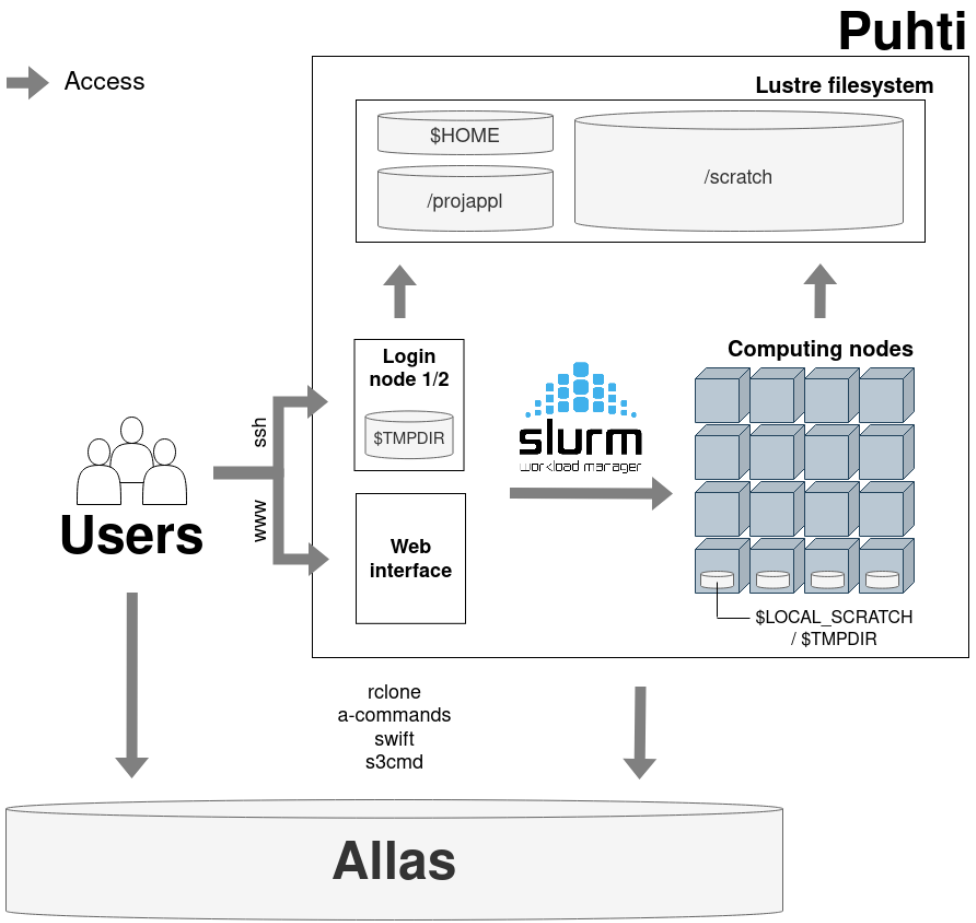

# Puhti web interface

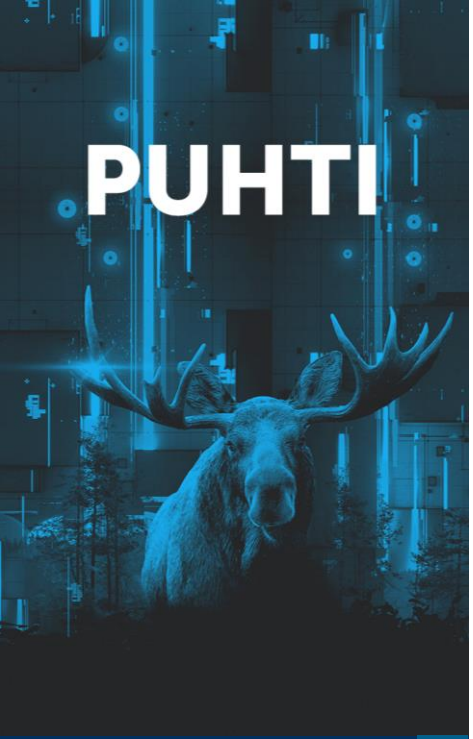
- Graphical interface to Puhti
- Used with web-browser
- Visual Studio Code, Jupyter Notebooks, Rstudio, TensorBoard
- QGIS, SNAP, GRASS, SagaGIS
- File browser
- Terminal




PUHTI


Puhti
Access
Lustre filesystem
$HOME
/projappl
/scratch
Login node 1/2
$TMPDIR
Web interface
slurm
workload manager
Computing nodes
$LOCAL_SCRATCH / $TMPDIR
Users
ssh
www
rclone
a-commands
swift
s3cmd
Allas

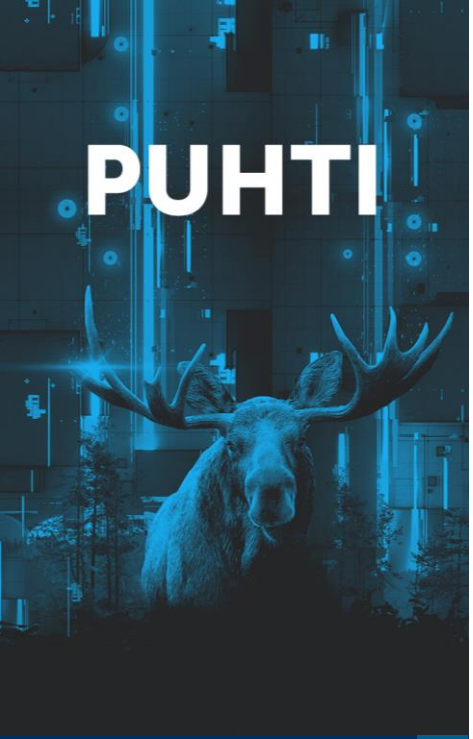



# The module system

- Puhti is a shared computing environment with hundreds of users
- Software is loaded with **modules**
  - Mutually incompatible software
  - One module: single program or group of similar programs
- **Example.** Loading module for geospatial Python tools

```
module load geoconda
```

- Check: https://docs.csc.fi/apps/#geosciences for module names

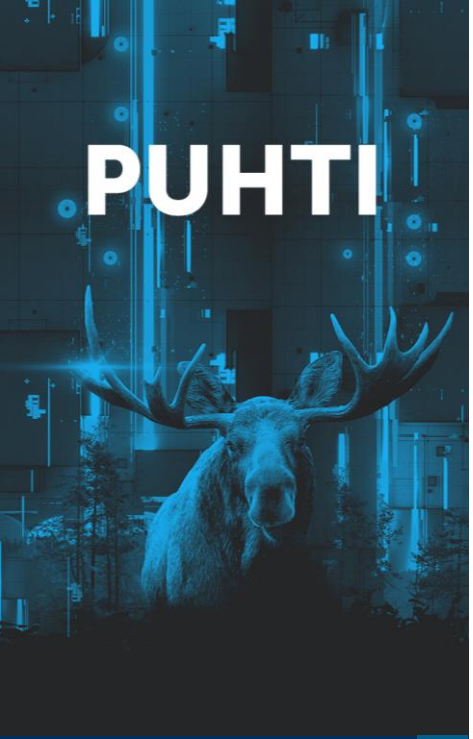



## geoconda

Module of pre-installed conda environment with:

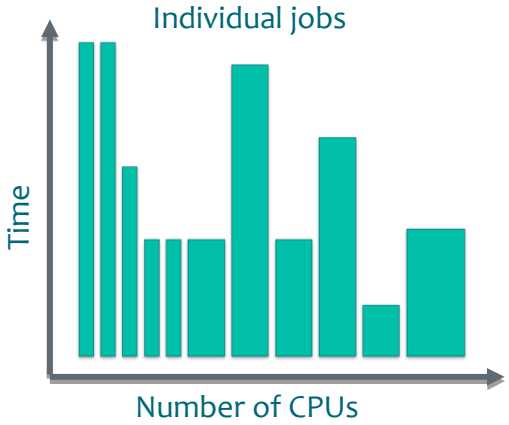
- all main packages raster, vector and lidar data analysis
- scikit-learn for machine learning
- packages for accessing data in Allas
- possibility to add packages with pip

https://docs.csc.fi/apps/geoconda/



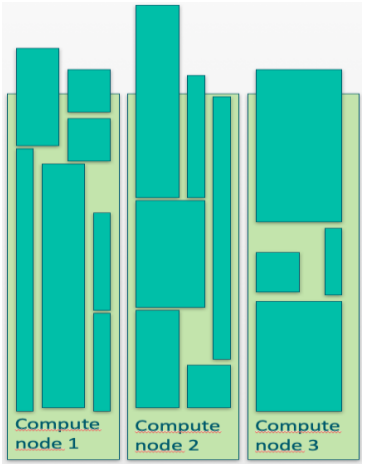

# The batch job system (SLURM)

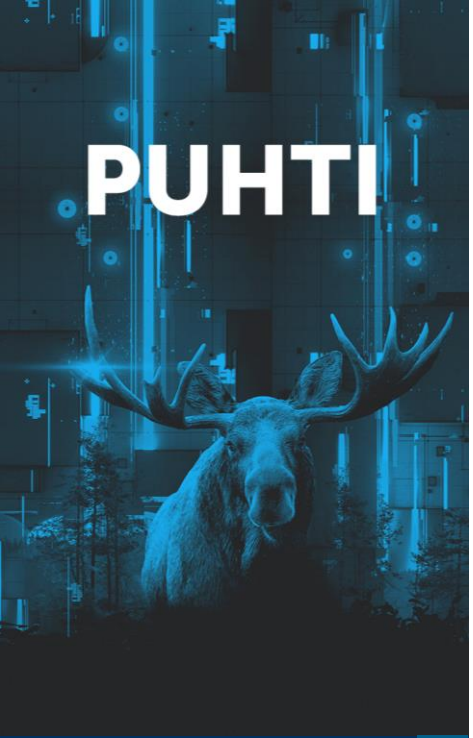
- Running jobs in Puhti requires you to use the batch job system
- You request resources for a batch job
  - CPU cores,  memory, GPU etc.
  - Time
- Batch job optimization of the queue by SLURM:




SLURM places jobs on computing nodes

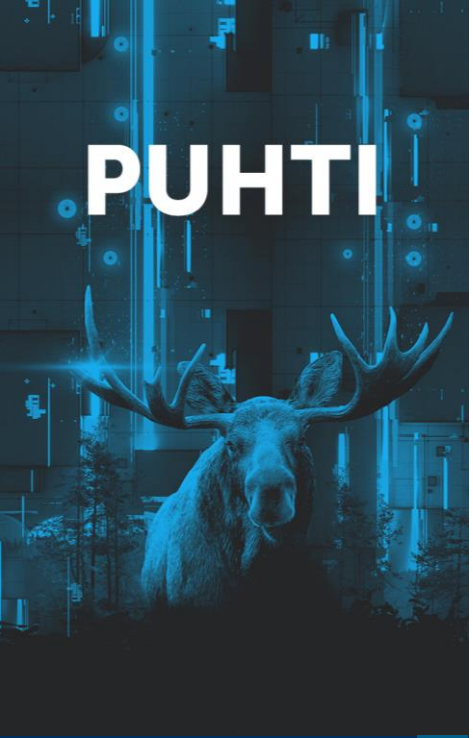
In the most efficient way resource wise






# Batch job partitions

| Partition | Time limit | Max tasks | Max memory |
|---|---|---|---|
| **test** | 15 minutes | 80 | 190 GiB |
| **interactive** | 7 days | 8 | 64 GiB |
| **small** | 3 days | 40 | 382 GiB |
| large | 3 days | 1040 | 382 Gib |
| longrun | **14 days** | 40 | 382 GiB |
| hugemem | 3 days | 160 | 1534 GiB |
| hugemem_longrun | 7 days | 40 | **1534 GiB** |

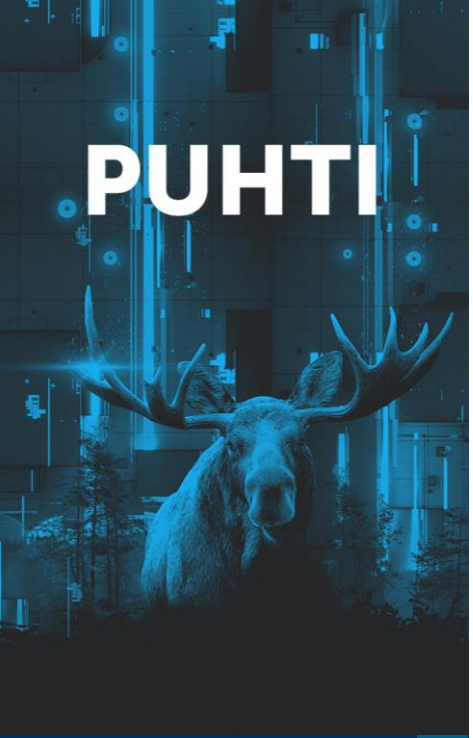



# Batch job scripts

- For requesting resources and submitting job description
- Specific bash scripts (.sh)

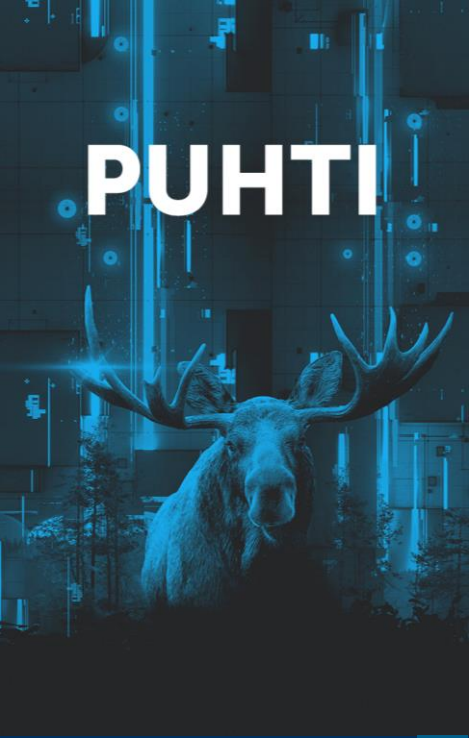
```
#!/bin/bash
#SBATCH --job-name=myTest
#SBATCH --account=<project>
#SBATCH --time=02:00:00
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=2000
#SBATCH --partition=small

module load geoconda

srun python my_python_script.py
```

- Batch job is submitted with the command
  ***sbatch <your-batch-job-script>***
- Cancel the job with
  **scancel <your-job-id>**
- See if your job has started running
  ***squeue –u <your-user-name>***
- After the job, see how much resources it used
  **seff <your-job-id>**



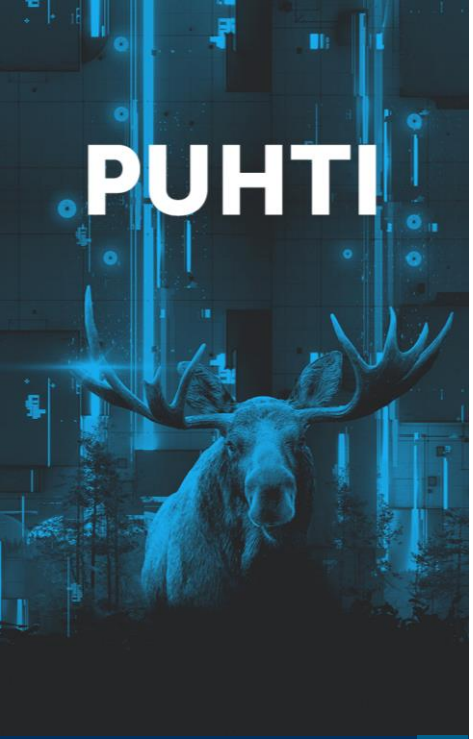

# Geospatial parallel jobs

- With parallel jobs you submit one job but reserver for it plenty of CPU cores
- Your script has to divide the workload to different cores, otherwise 1 core is doing the work, others are just idling.
- In GIS, you often divide the dataset and give each worker (CPU core) their own subset. It could be vector features, raster subsets, or text inputs in a .CSV file
- How many workers should you utilize depends how long handling one subset takes. Communication always takes extra time so your workload for one worker should not take less than ~5 minutes




## Steps for running a Python script in Puhti

1. (Set up: account, project, services, BU)
2. Log in to Puhti.
3. Move your data and scripts to Puhti.
4. Open VSCode
5. Check which Python packages do you need and if they are available in Puhti.
   * If needed, install it yourself or ask CSC - servicedesk@csc.fi.
6. Fix the paths of your input/output files.
7. Test your script with some test data
8. Run your scripts with all data as batch job (or interactively).
9. (Make use of several cores in your Python code.)




# Puhti, next steps

**Read Puhti documentation**

- Geocomputing: https://research.csc.fi/geocomputing
- GIS software specific pages: https://docs.csc.fi/apps/#geosciences
- Puhti GIS examples: https://github.com/csc-training/geocomputing
- CSC **Linux tutorial** for basic Linux commands: https://docs.csc.fi/support/tutorials/env-guide/overview/

**Come to a course**

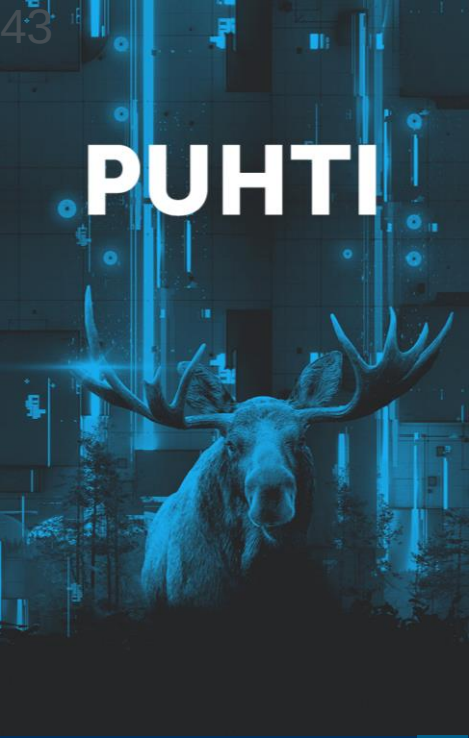
- Using CSC HPC environment efficiently: https://ssl.eventilla.com/event/ge9Ey

**Contact CSC**

- servicedesk@csc.fi

**Join gis-hpc-mailing list**

- https://postit.csc.fi/sympa/info/gis-hpc




# Summary

**Puhti is an excellent tool if you**

- need more computing power
- don’t want to run long analyses on your personal computer
- have **a lot** of data
- are using data already available in Puhti in large quantities
- are willing to use scripts for your work
- have really basic Linux skills
- are willing to learn to use Puhti