CSC

ICT Solutions for Brilliant Minds

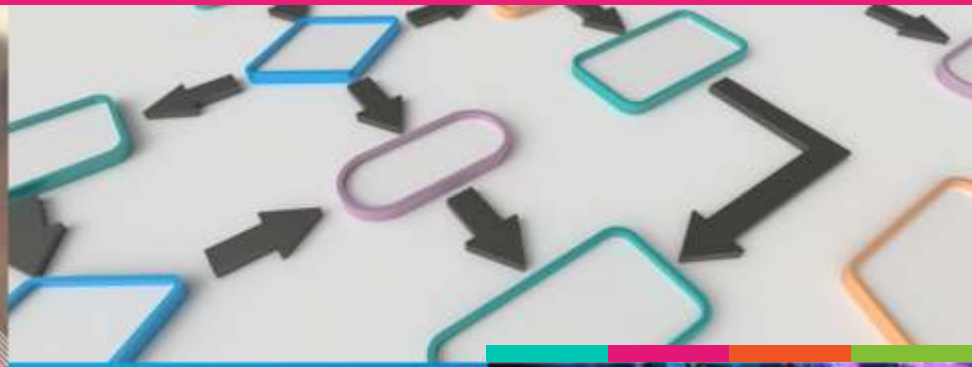
Geocomputing in Puhti supercomputer

4.11.2019

CSC

Kylli Ek, Johannes Nyman , Ziya Yektay

Course details



Program 04.11.2019

09:00 – 10:00	**Linux introduction**
10:00 – 10:15	**Coffee break**
10:15 – 12:00	**Introduction to CSC services and platforms. GIS software and data in Puhti/Taito. Virtual rasters. Connecting to Puhti/Taito. Use of module systems.**
12:00 – 13:00	**Lunch break**
13:00 – 14:30	**Running jobs. Running R code in Puhti/Taito**
14:30 – 14:45	**Coffee break**
14:45 – 16:00	**Running Python code in Puhti/Taito**

Intro to CSC computing services

Non-profit state organization with special tasks

Turn over in 2018

44,9 M€

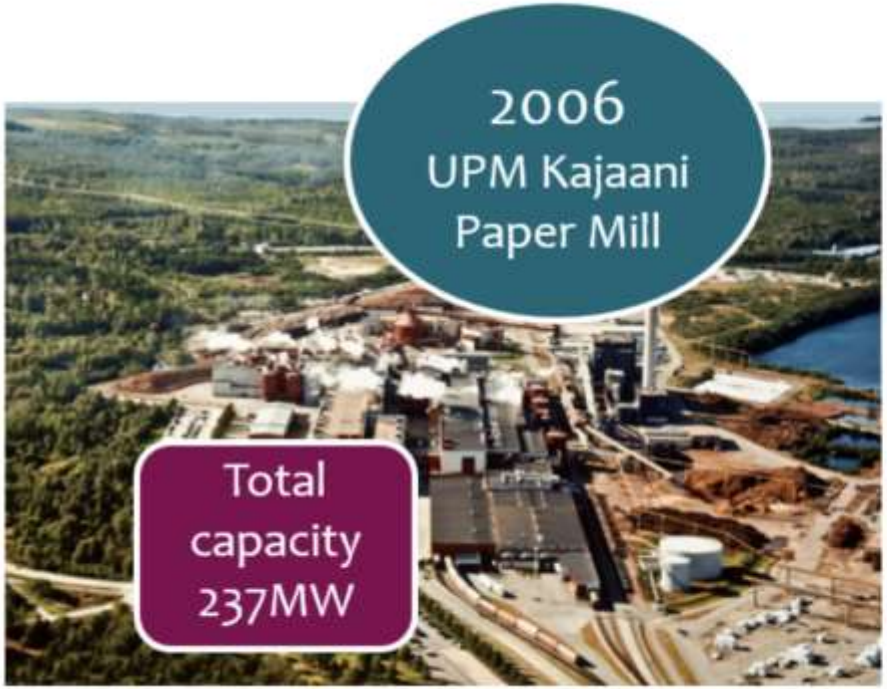
Headquarters in Espoo, datacenter in Kajaani

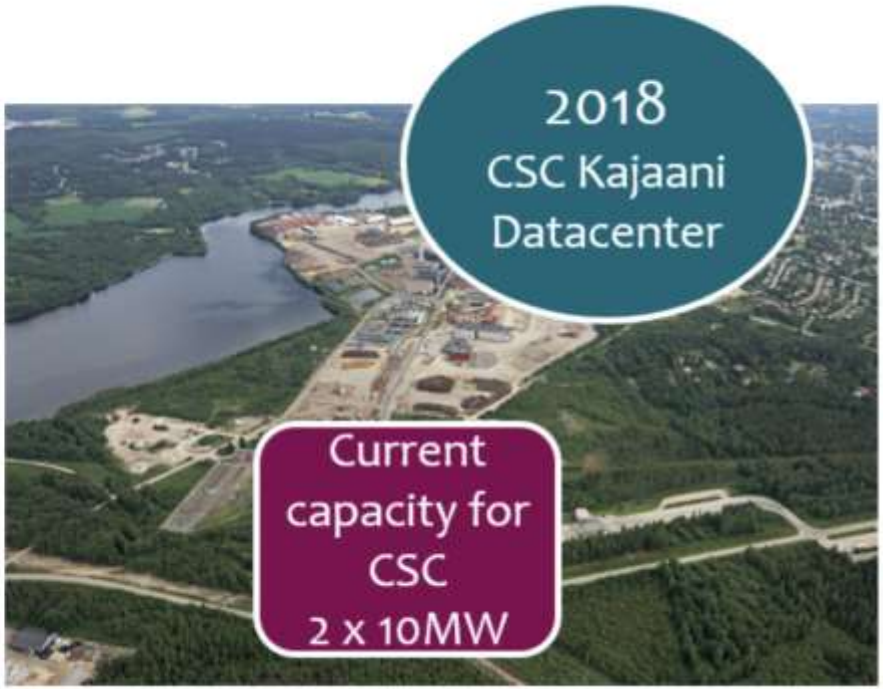
Owned by state (70%)

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

Circa **350** employees in 2018

Kajaani data center





Reasons for using CSC computing resources

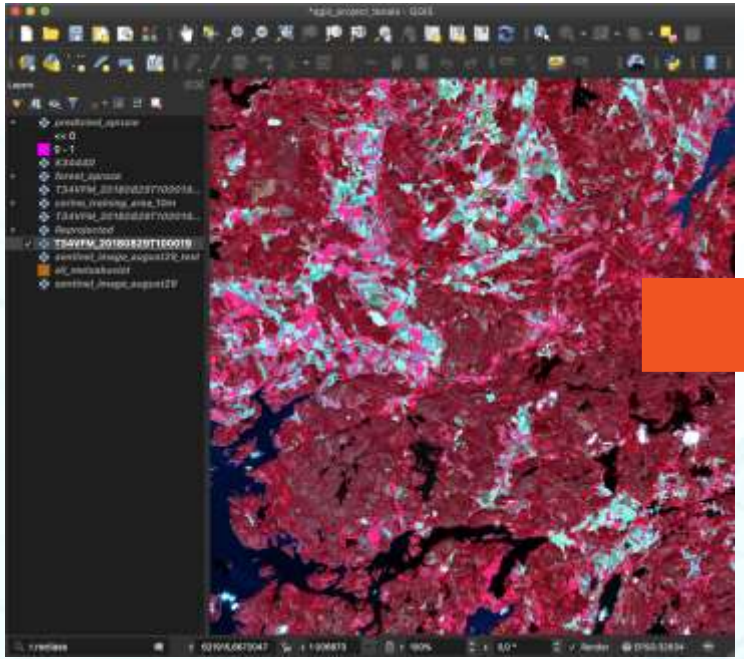
- Computing something takes more than 2-4 hours
- Need for more memory
- Very big datasets
- Keep your desktop computer for normal usage, do computation elsewhere
- Need for a server computer
- Need for a lot of computers with the same set-up (courses)
- Convenient to use preinstalled and maintained software

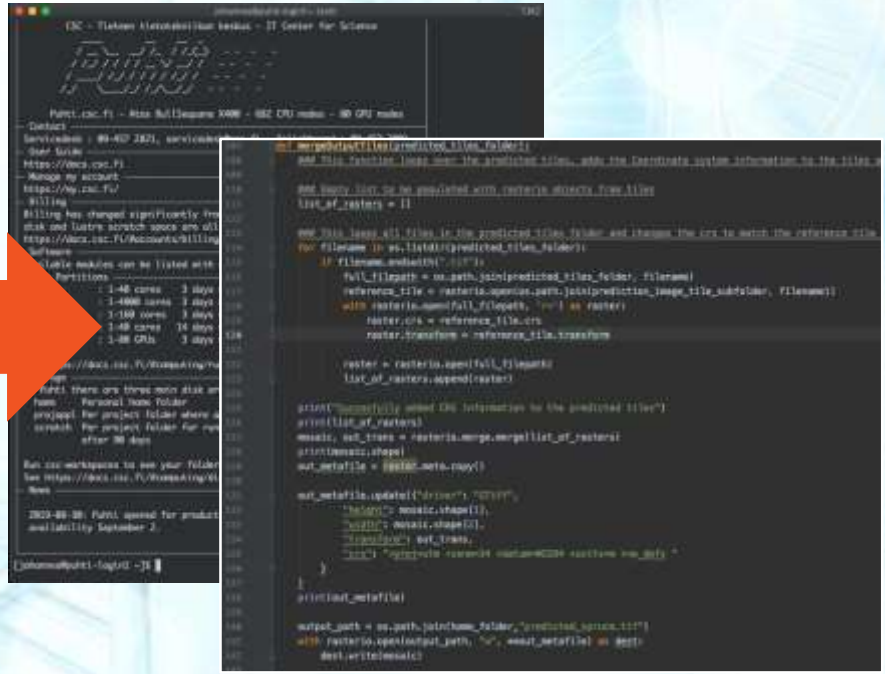
- Free for Finnish university and for state research insitute users

CSC main computing resources for GIS users

	Puhti/Taito	cPouta cloud	Rahti container cloud
System	Supercomputer	Virtual machine cloud	Container cloud
Software	Pre-installed software + user-installed software	User-installed software	User-created containers
Use cases	Run demanding analyses with numerous CPUs or GPUs	Setup your own virtual machine and environment	Host services easily (Jupyter notebook server, GeoServer, PostGIS)

The corner stone of high-performance computing in Puhti





Graphical user interfaces: ArcGIS, QGIS

Scripts: Python, R, shell, Matlab, …

PUHTI supercomputer specifications

- Launched 02.09.2019
- **Puhti** has **682** CPU nodes which each have **40 CPU** cores. In total Puhti has **27 280 CPU** cores

- **Puhti AI** is the **GPU** partition of Puhti
- It has **80** nodes which each have **4 GPUs** (Nvidia Tesla V100) and 40 **CPUs**

- Peak performance **2,7** petaflops. In June 2019 it was listed as the **166th** fastest computer in the world (The TOP500 project)

Example GIS use cases for Puhti

- Satellite data analysis
- Forest modelling
- Prioritization of protected areas
- Climate variables modelling
- Species modelling
- Routing / travelling time matrices
- DEM analysis
- GIS Machine learning

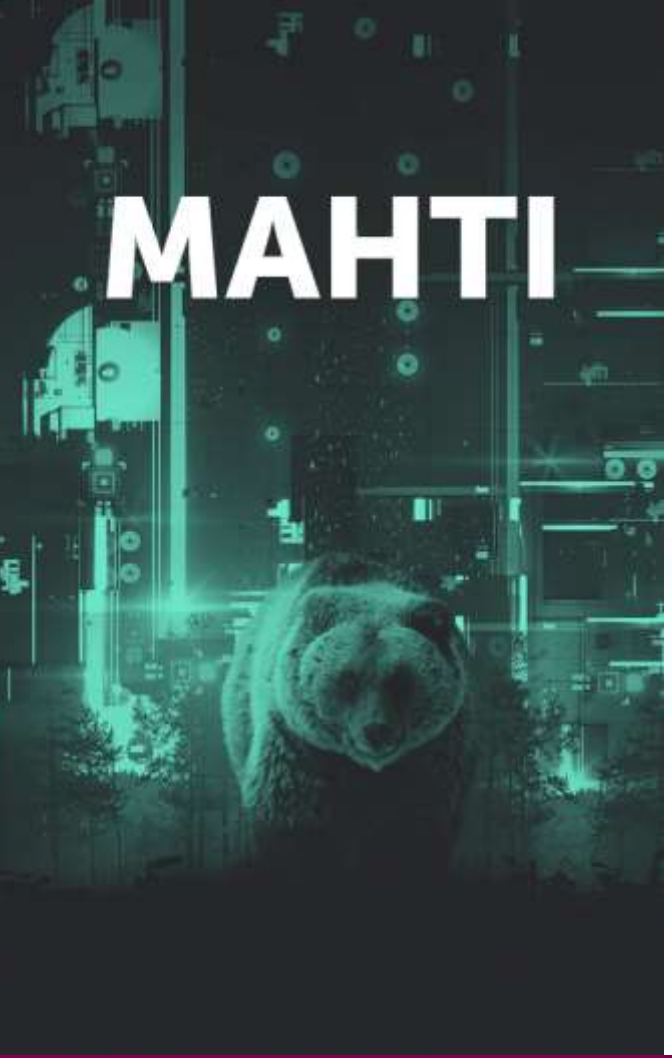
Puhti vs. cPouta

- Puhti
  - Ready working environment
  - Significant amount of computing power
  - Preinstalled software
  - Finnish GIS data
  - Limitations on software that can be used: Linux, no server, no root access

- cPouta
  - Free hands, but also more responsibility
  - Less computing power
  - Always starting from zero
    - Operating system, firewalls, security groups, software installation, users etc

Realism check

- A single core in Puhti is only slightly faster than the one on your laptop
- The computing power is based on quantity. Puhti has tens of thousands of cores
- This means running the same single core script in Puhti is not faster. You need to optimize the script to run on several cores at the same time (a parallel job) or split the problem to an array of jobs (an array job)



The near future of supercomputers at CSC

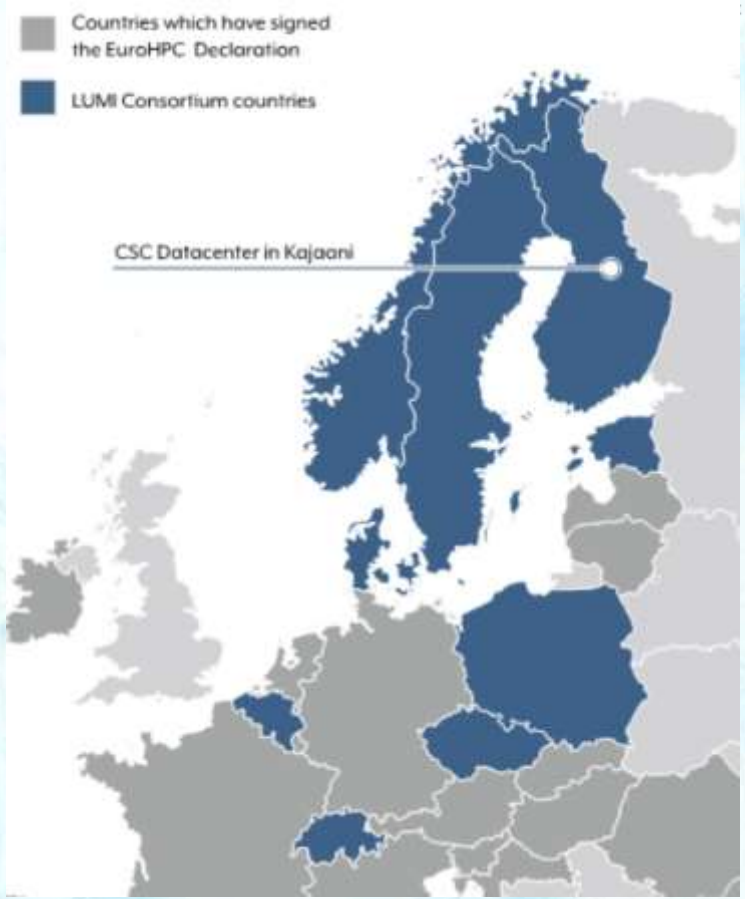
MAHTI

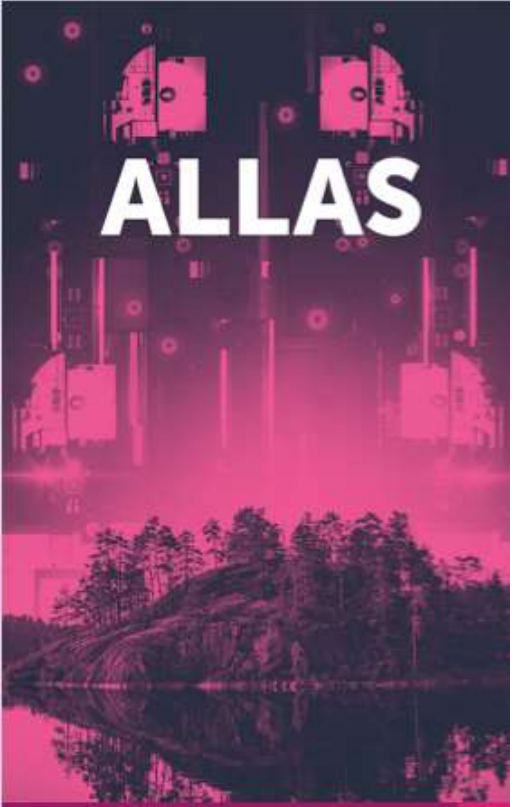
- A more powerful supercomputer than Puhti
- Launching early 2020
- Designed especially for significantly larger jobs in mind (physics, chemistry)
- No GIS software will be installed


LUMI

LUMI

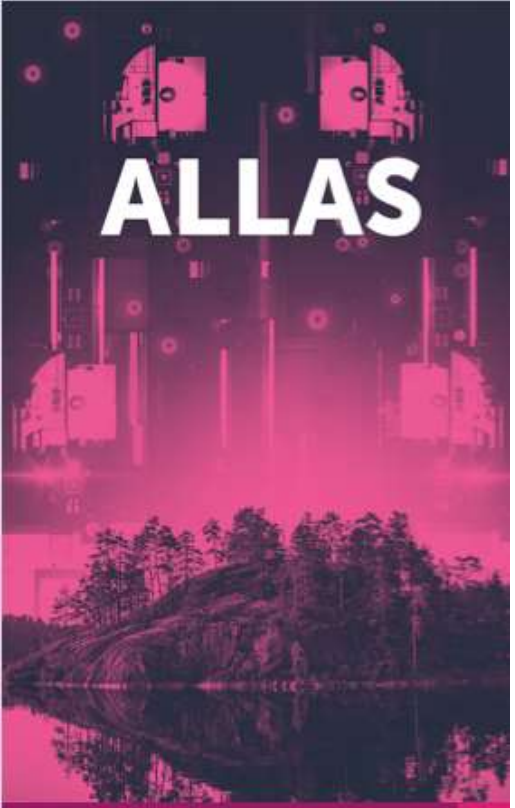
- One of the world's fastest supercomputer systems at the time of completion
- **+200-million euro** joint project of **nine** European countries
- Launching **early 2021**
- Peak performance over **200 petaflops** which is more than **10** times the performance of the current fastest in Europe





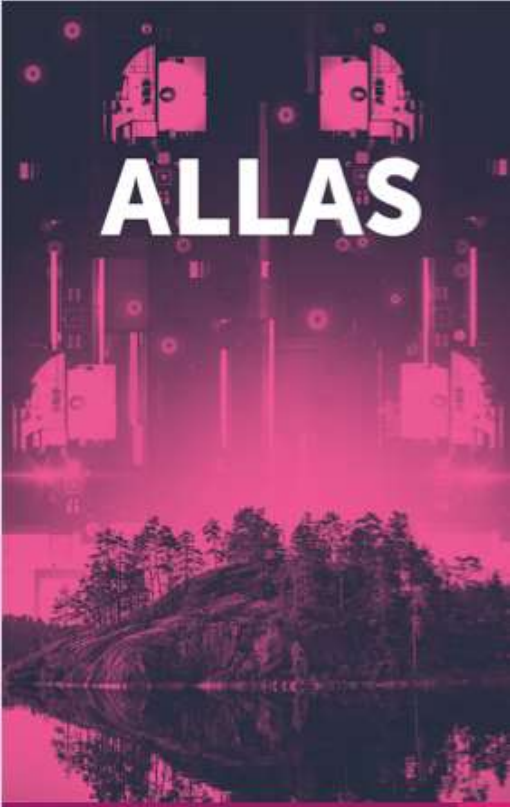
ALLAS – object storage: what it is for?

- **Allas** is a new storage service for all computing and cloud services
  - Default quota **10 TB** / Project.
  - Available in **Taito**, **Puhti** and **Mahti**
  - Data can also be shared via Internet
  - An object is stored in multiple servers so a disk or server break does not cause data loss
  - Data cannot be modified in the object storage – data is immutable



ALLAS – object storage: what it is for?

- **ALLAS** supports two popular protocols
  - **Swift**
    - swift, rclone, a-tools, cyberduck
  - **S3**
    - S3cmd

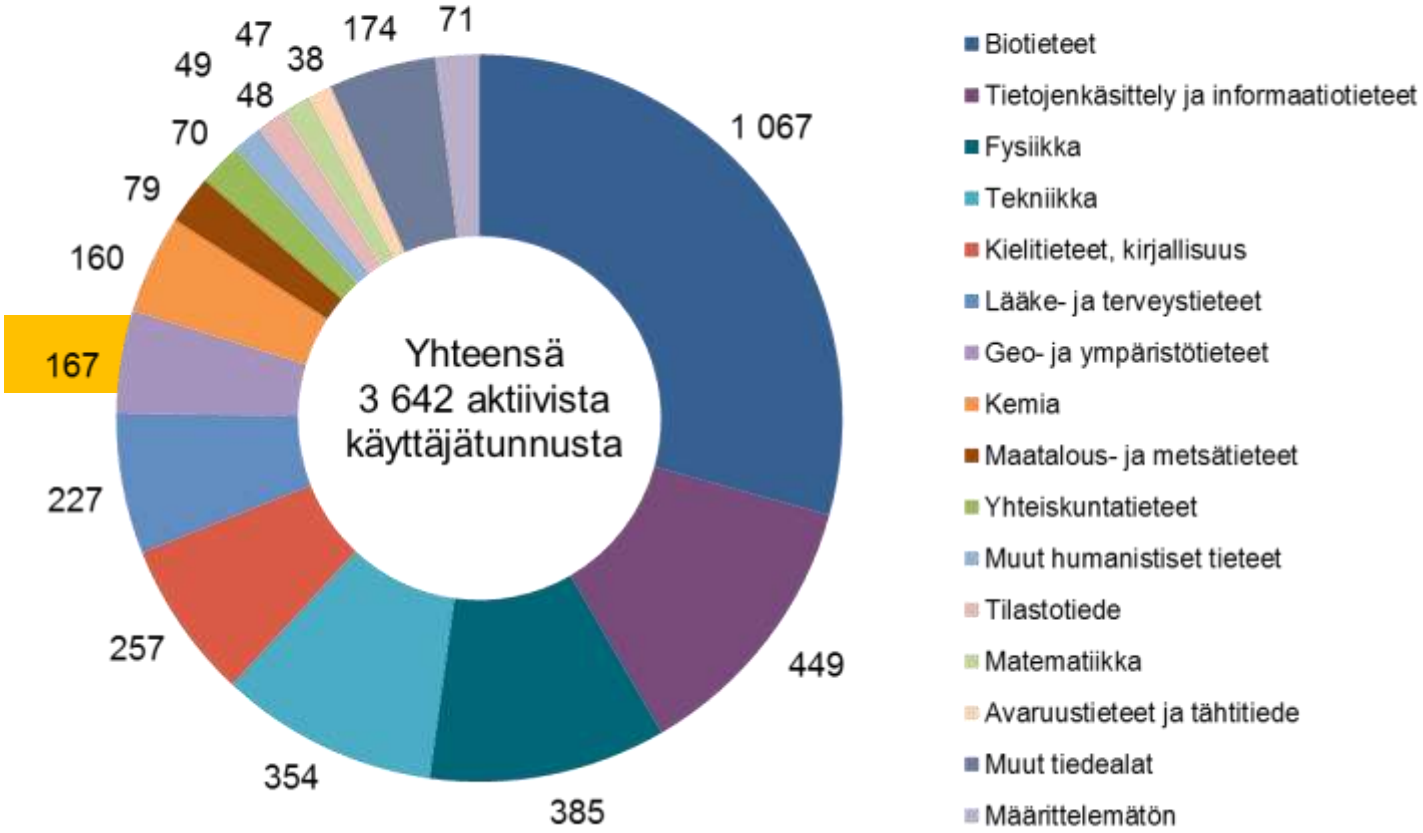


Allas – how to use with Puhti

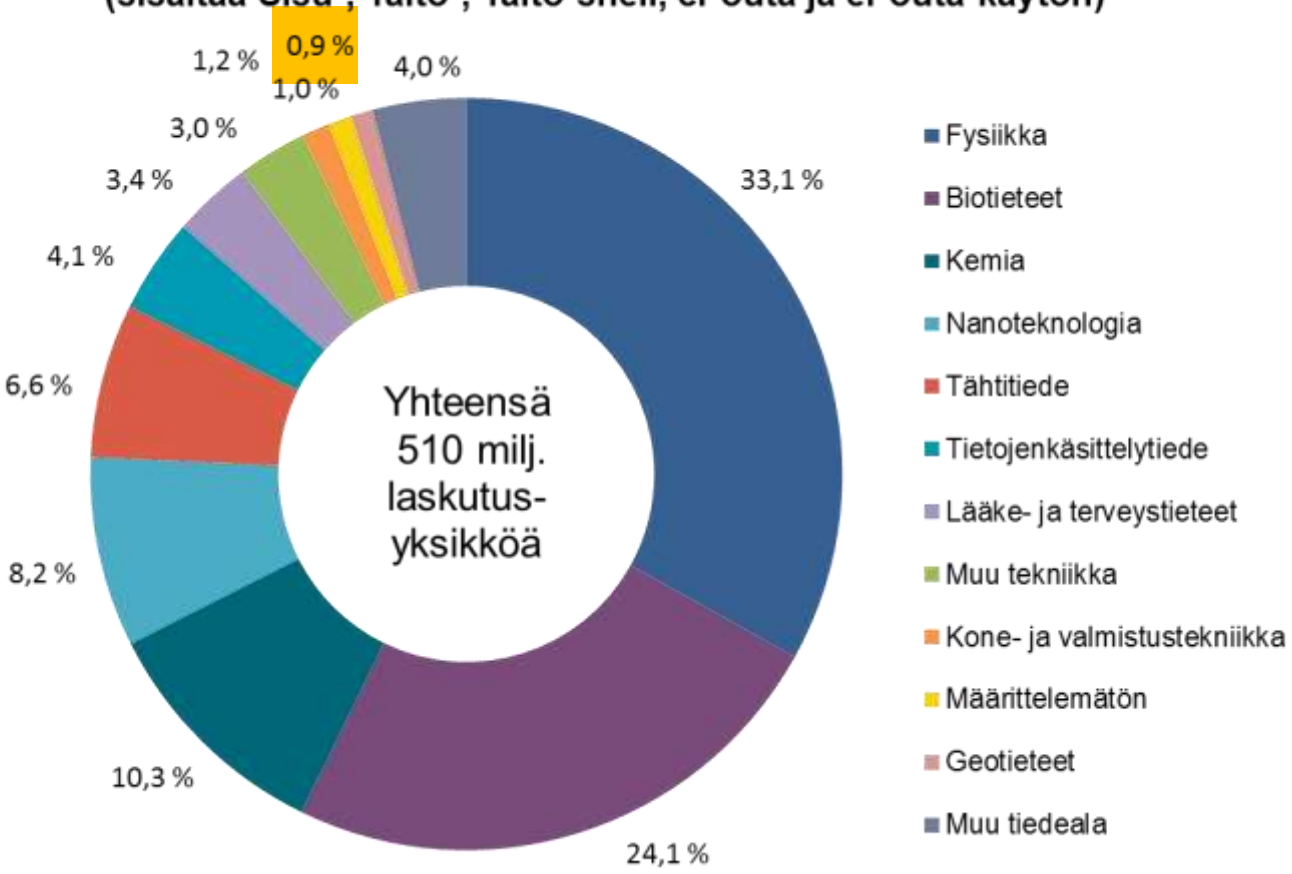
a-tools

- Rclone based scripts by CSC for using **Allas** in **Taito** and **Puhti**
- A-tools tries to provide easier and safer way to use Allas for occasional Allas user users
- Main commands:
  - a-put Upload a file or directory to Allas as one object
  - a-get Download a stored dataset (object) from Allas
  - a-list List buckets and objects in Allas
- More information: https://docs.csc.fi/#data/Allas

Aktiiviset käyttäjätunnusasiakkaat tiedealoittain kesäkuun 2018 lopussa



Tietokoneresurssien käyttö tiedealoittain kaudella H1/2018
(sisältää Sisu-, Taito-, Taito-shell, cPouta ja ePouta-käytön)



Getting access to CSC resources – My CSC portal

My CSC

- Web portal for all CSC users – my.csc.fi
- **Create an account**
- **Manage your account**
- Create and manage projects
- Apply for computing services and resources

How to get started

- Follow the instructions in **docs.csc.fi** https://docs.csc.fi/#accounts/
- Create an account at the **My CSC** service https://my.csc.fi
  - Login with own university / institute credentials via HAKA or Virtu
- Create or join a project
  - New project has to be created by a Project Manager who can then invite users
  - Project manager has to apply for access to different services (Puhti, Rahti, Allas) and **the user** has to accept the user license in **my.csc.fi**

Enabling services and accepting user agreement

- After opening a project, on the right side of my.csc.fi there exists a list of available CSC services
- Project manager has to apply for the services and every user has to agree on the user license before they can start using them
  - You can accept the license by opening the tab of the service
- If ✔ , you're good to go!

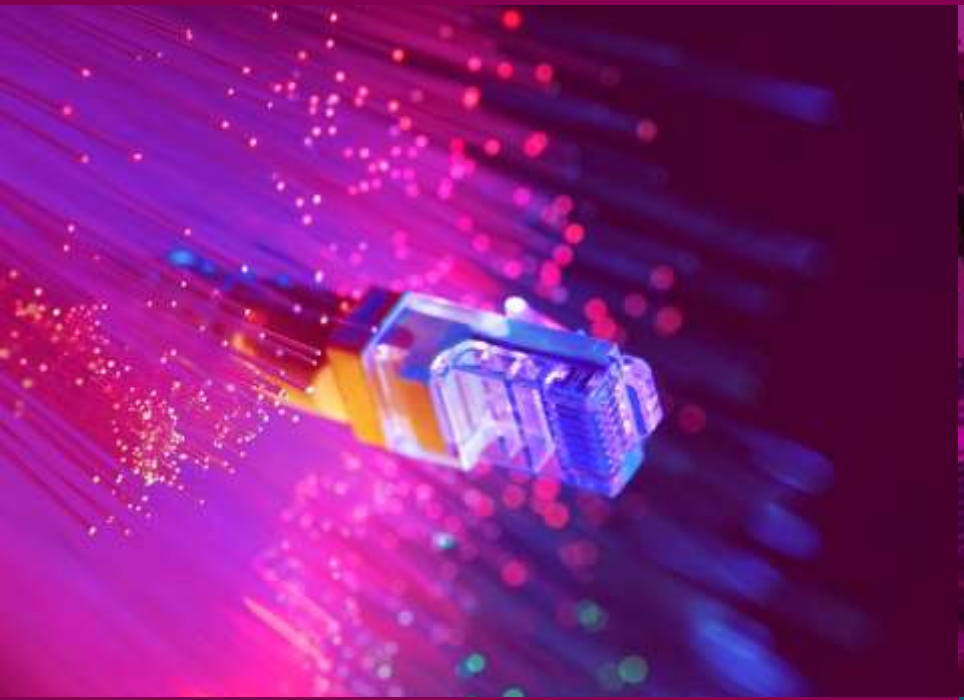
Requesting additional billing units

- Project manager can request additional billing units for the project from **my.csc.fi**
- Just hit 

after opening a project and apply for resources

- More on billing units later

GIS software and data in Taito/Puhti



Pre-installed software

- CSC provides researchers the largest collection of scientific software and databases in Finland.
  - ***Some software installed to Taito/Puhti is commercial, read the license terms from software pages. For example MatLab requires both CSC and user license.***
  - ***No license restrictions for GIS-software.***
- The list of pre-installed and supported scientific software packages can be found at:
  - Puhti: https://docs.csc.fi/#apps/
  - Taito: https://research.csc.fi/software

GIS software in HPC Taito and Puhti

- ArcGIS Python API
- CloudCompare
- FORCE
- **GDAL/OGR**
- GRASS GIS
- LasTools , also .exe tools with Wine
- MatLab / Octave
- Mapnik
- PDAL
- **Python GIS packages**
- QGIS
- **R GIS packages**
- Solaris
- SNAP, sen2cor
- SagaGIS
- Taudem
- Zonation

GIS Software not available in HPC Taito/Puhti

Windows software:

- ArcGIS, but ArcGIS Python API is in Puhti
- MapInfo

Server software

- GeoServer, MapServer
- PostGIS

Web map libraries

- OpenLayers, Leaflet

Parallel computing with GIS software

- Out-of-box parallel: Taudem
- Out-of-box something in parellel: SagaGIS, R (raster, lidR), GRASS, gpt in SNAP
- Parallel packages of Python (multiprocessing) and R (several different), Python can use only one node = 24-40 cores.
- Array jobs we SLURM, we get back to these soon.

Installing software

- Possibility to install software for own use
  - The software must be available for Linux
  - .. and installation must be possible without root access
  - Easy isntallations with conda
  - Slower installations from source code, ask for help!
- You can add also packages to R and Python
- Ask CSC servicedesk to install some software for everybody

GIS data in Taito/Puhti

- Hosts large commonly used datasets
- Reduces the need to transfer data to Taito/Puhti
- Located at
    - Taito: /proj/ogiir-csc/
    - Puhti: /appl/data/geo/
- All Taito/Puhti users have read access.
- Only CSC personnel have write access.
- For data with open license

- If you think some other dataset should be included here, ask from servicedesk@csc.fi

All Paituli open data
+
Syke
  All open datasets
LUKE
  Multi-source national forest inventory
NLS
  Topographic database (gpkg)
  Virtual rasters for DEMs

https://research.csc.fi/gis_data_in_taito

Virtual Rasters in Taito

- Ready made virtual rasters for 2m and 10m dems
- Allows working with dataset of multiple files as if they were a single file.
- XML pointing to actual raster files
- External overviews and xml headers

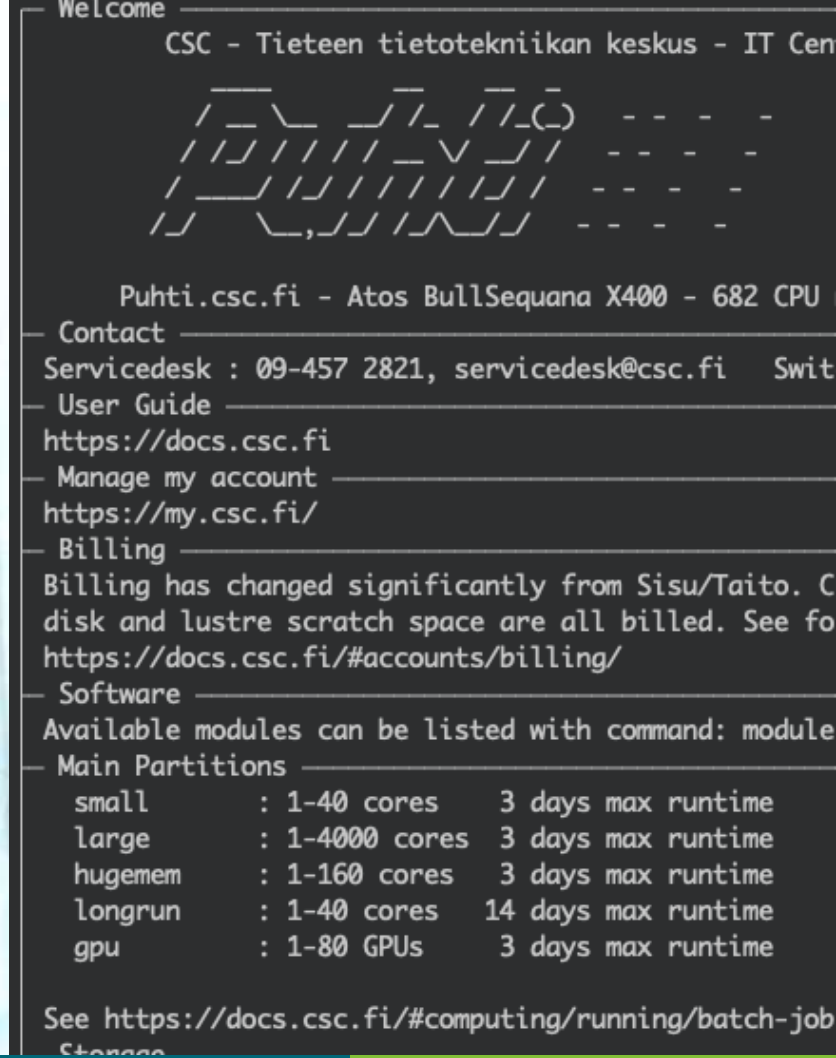
- A python script to create your own for a specific area

Connecting to the CSC supercluster Puhti

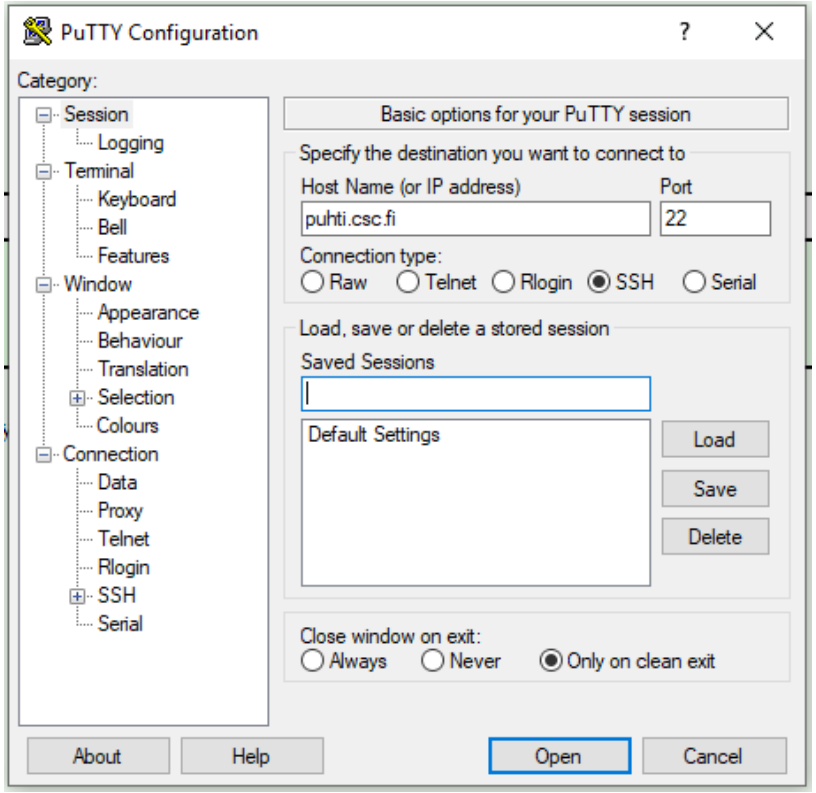
Accessing Puhti – Linux & Mac OS

- On UNIX / Linux / OSX command line
  - **`ssh <username>@puhti.csc.fi`**
- On Windows: PuTTY or some other tool.
  - In Win10 you can use also Linux subsystem.
  - Putty can be installed without admin privileges

- Organizations with IT-services from Valtori usually need extra steps, ask from local IT-department



PuTTY

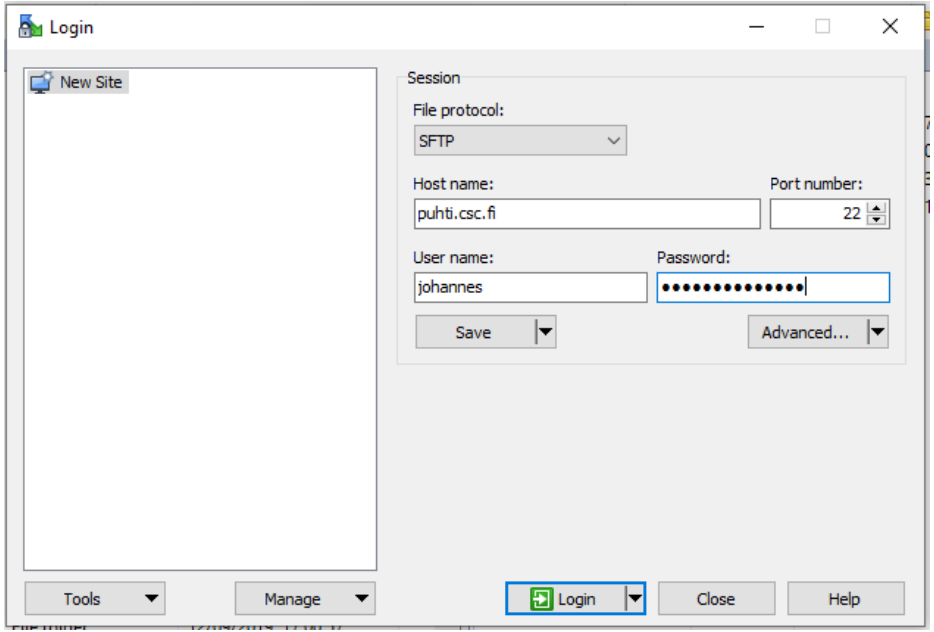

PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address)
Port
puhti.csc.fi
22
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
Default Settings
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Help
Open
Cancel


johannes@puhti-login1:~
Welcome
CSC - Tieteen tietotekniikan keskus - IT Center for Science
Puhti.csc.fi - Atos BullSequana X400 - 682 CPU nodes - 80 GPU nodes
Contact
Servicedesk : 09-457 2821, servicedesk@csc.fi Switchboard : 09-457 2001
User Guide
https://docs.csc.fi
Manage my account
https://my.csc.fi/
Billing
Billing has changed significantly from Sisu/Taito. CPU, GPU, Memory, local disk and lustre scratch space are all billed. See for details:
https://docs.csc.fi/#accounts/billing/
Software
Available modules can be listed with command: module avail and module spider
Main Partitions
small : 1-40 cores 3 days max runtime
large : 1-4000 cores 3 days max runtime
hugemem : 1-160 cores 3 days max runtime
longrun : 1-40 cores 14 days max runtime
gpu : 1-80 GPUs 3 days max runtime
See https://docs.csc.fi/#computing/running/batch-job-partitions/
Storage
In Puhti there are three main disk areas:
home Personal home folder
projappl Per project folder where applications and libraries can be stored
scratch Per project folder for running jobs. Files are removed after 90 days
Run csc-workspaces to see your folders and quotas
See https://docs.csc.fi/#computing/disk/ for details
News
2019-08-30: Puhti opened for production for pilot users, general availability September 2.
[johannes@puhti-login1 ~]$

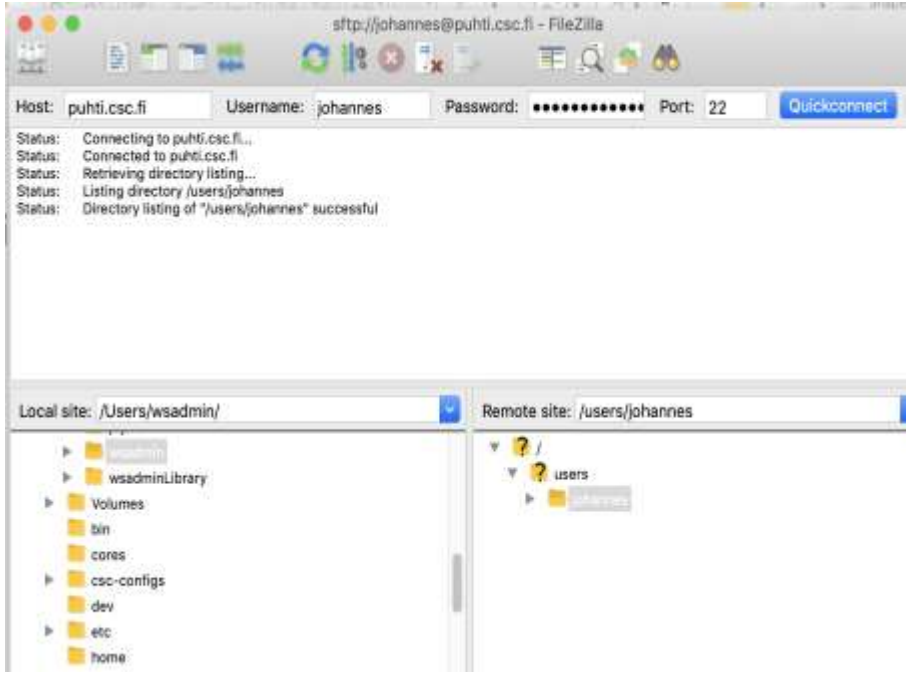
Moving files

- Command line tools: scp ,wget, rsync
- GUI FTP-tools: FileZilla or WinSCP
- More information on connecting to Puhti
  - **https://docs.csc.fi/#computing/overview/#connecting-to-puhti**

WinSCP



FileZilla



Tips for file transfers

- When downloading from exernal services try to download directly to CSC, not via your local PC
- If you plan to download large quantities of big files use
  - **Rsync**
  - **wget/curl** if HTTP-urls
- A new CSC file management system **Allas** should be used to store project's data

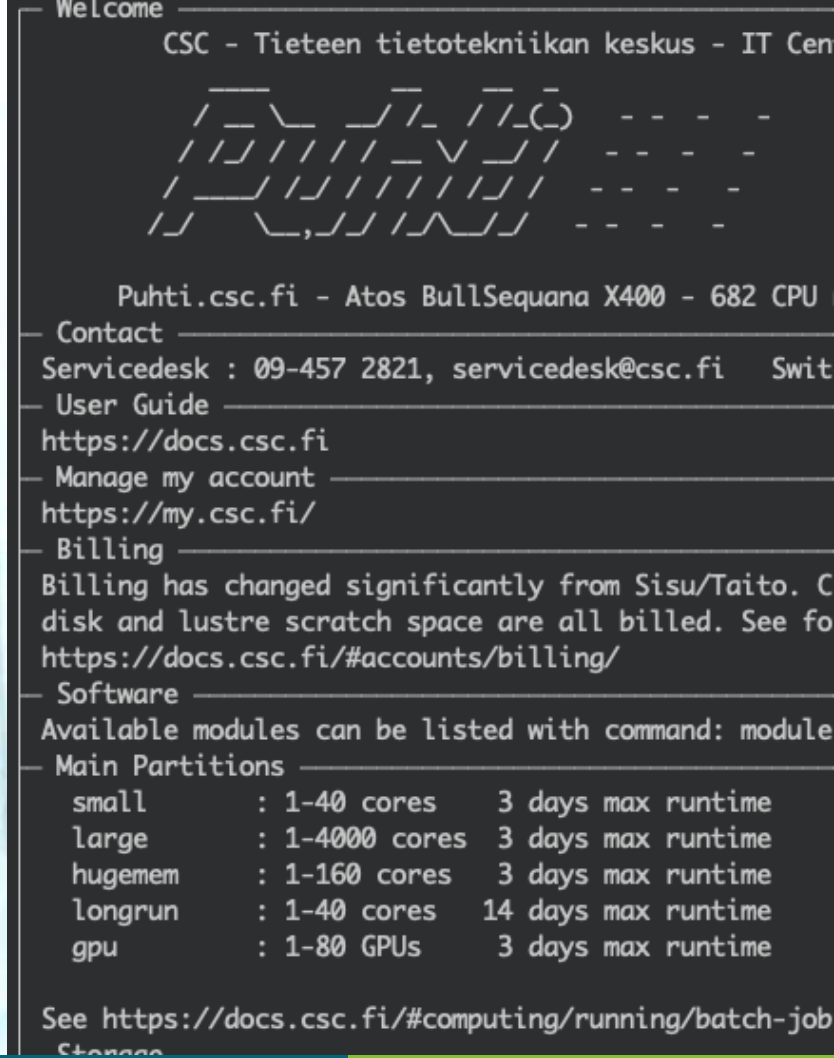
Data transfer speed examples

	50 Mb / s	25 Mb / s	5 Mb / s
1 kb	0,00002 s	0,00001 s	0,0002 s
1 Mb	0,02 s	0,01 s	0,2
1 Gb	20 s	40 s	3,5 min
1 Tb	5,5 h	11 h	2d 7h

- 50 Mb/s often the realistic fast speed
- 25 Mb/s good normal speed
- 5 Mb/s realistic speed in mobile network

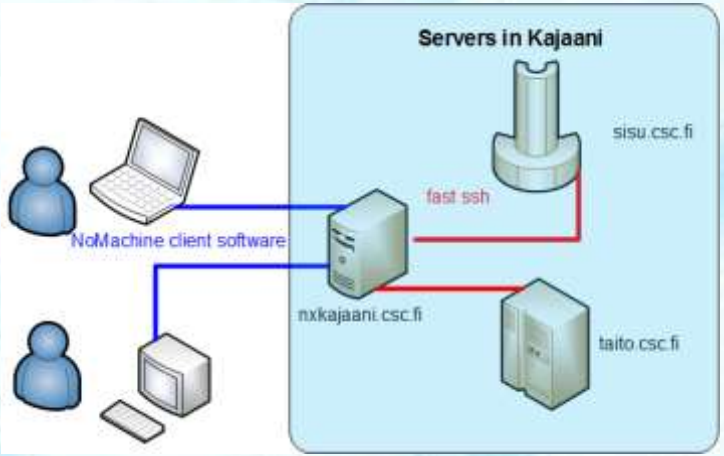
Graphical connection

- NoMachine
- ssh with –X (or –Y)
  - **`ssh -Y <username>@puhti.csc.fi`**
  - In Windows and Mac requires additional software, for example Xming and XQuartz



NoMachine Remote Desktop

- Client connection between user and gateway
- Good performance even with slow network
- SSH from gateway to server (fast if local)
- Persistent connection
- Suspendable
  - Continue later at another location
- Read the instructions…
  - keyboard layout, mac specific workarounds, …
- Choose an application or server to use (right click)



Taito-shell / Puhti-shell

- For interactive work
- Max 4 cores
- Max memory depends on other users, still likely more than on a laptop.
- Puhti-shell not available yet.
- NoMachine works best with Taito-shell / Puhti-shell.
- Same software and data folders as in Taito / Puhti.

Local directories in Puhti

Local directories at Puhti

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 the ***scratch*** and ***projappl*** directories can be increased by contacting **servicedesk@csc.fi**
- More information - **https://docs.csc.fi/#computing/disk/**
- **No back-up**

Module system

The module system

- Modules are used to load software in Puhti
- Modules
  - are necessary on a system with mutually incompatible software
  - can be for a single program or group of similar programs
  - load applications, adjust path settings and set environment variables

```
------------------------------------------ /appl/modulefiles ------------------------------------------
   StdEnv                          (L)    elmer/fisoc                      openbabel/3.0.a1
   abaqus/2017                            elmer/latest                (D)  openfoam/of
   abaqus/2019                     (D)    emboss/6.5.7                     python-data/3.7.3-1
   ansys/19.2                             gaussian/G16RevC.01              python-env/2019.3
   bamtools/2.5.1                         geoconda/3.7                     pytorch/1.0.1
   bcl2fastq/2.20.0.422                   gpaw/1.4.0                       pytorch/1.1.0
   beast/2.6.1                            gpaw/1.5.2                  (D)  pytorch/1.2.0          (D)
   bedtools/2.29.0                        gromacs/2019                     relion-env/3.0.6-cuda
   bioconda/3                             hisat2/2.1.0                     sagemath/sage-8.8
   biojava/1.8                            hmmer/3.2.1                      samtools/1.9
   biokit/9.1.0                           idl/8.7.2                        spades/3.13.0
```

Typical module commands

`module avail`	shows available modules (compatible modules in puhti)
`module spider`	shows all available modules in puhti
`module list`	shows currently loaded modules
`module load <name>`	loads module <name> (default version)
`module load <name/version>`	loads module <name/version>
`module switch <name1> <name2>`	unloads module name1 and loads module name2
`module purge`	unloads all loaded modules

Module example

- Show available modules in Puhti:
  - `module spider gdal`

```
------------------------
  gdal:
------------------------
     Versions:
        gdal/2.4.2-omp
        gdal/3.0.1-omp
```

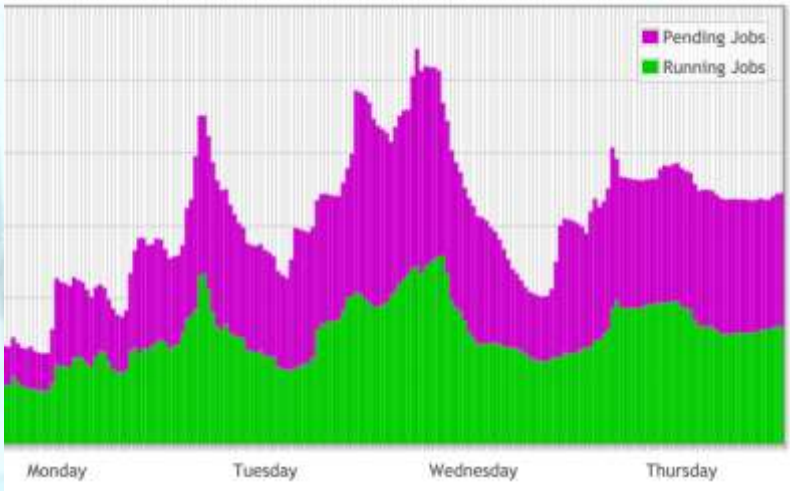
- Initialize a specific version of **gdal**
  - `module load gdal/3.0.1-omp`

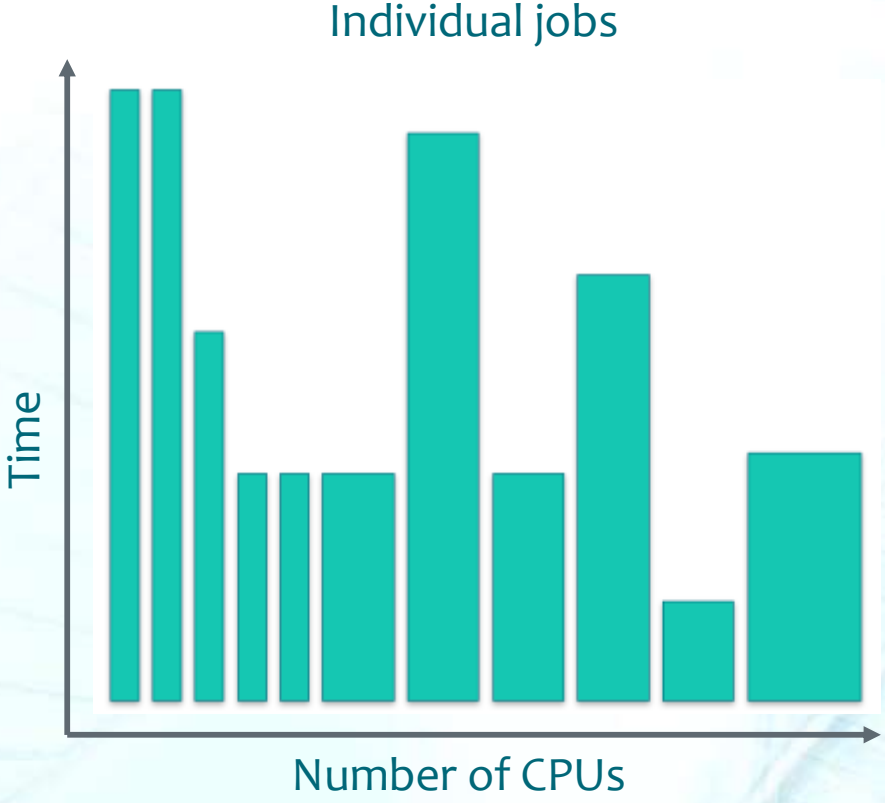
Check from docs.csc.fi which modules to load

Running jobs in Puhti

The batch job system

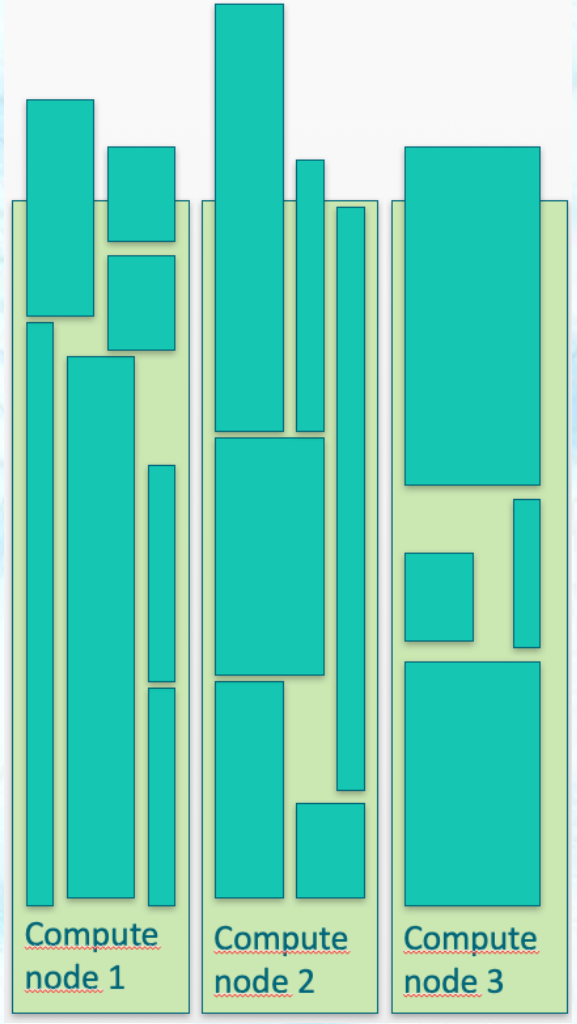
- Optimizes resource usage by queuing the supercluster with jobs
- Running jobs are chosen according to their priority and specifications
  - Time in queue increases priority
  - Short and small jobs fit better to the scheduling queue, while long and large jobs take more time to start running
- CSC uses SLURM for batch job system
  - Simple Linux Utility for Resource Management





SLURM places jobs on computing nodes

In the most efficient way resource wise



Creating a batch job

1. Write a batch job script
   - Batch job scripts are simple bash (**.sh**) scripts
   - Scripts tells SLURM what resources are needed, for how long and what commands to run

2. Make sure all necessary files are reachable for the computing nodes (e.g they are in your project's directory)

3. Submit your job!
   - `sbatch my_batch_job.sh`

Resources in batch jobs

- User requests resources in the batch script. These are number of CPU cores, memory reservation, GPU capability and time
- Resources need to be adequate for the job
  - Too small memory reservation will cause the job to fail
  - Too little time reserved will cause the job to be terminated whether finished or not
  - But, the more resources are requested, the more time the job spends in queue, especially with large memory requests
- Realistic resource requests give the best results
  - Not always easy to know beforehand
  - Usually best to try with smaller tasks first and check used resources with **seff** command
- **Note! Using more CPU cores does not make the job faster if the script/application does not know how to use multiple CPUs**

Billing units

- The using of CSC computing and storage is based on billing units
- Each project can apply for billing units and use billing units for computing and storage
- In Puhti the billing scheme is the following:
  - Each reserved core consumes **1** BU per hour
  - Each GiB of reserved memory consumes **0.1** BUs per hour
  - Each reserved GPU consumes **60** BUs per hour.
  - The default 1 TiB scratch disk quota is free, increased quota: 1 TiB consumes **50 000** BUs per year.
- Billing details: https://docs.csc.fi/#accounts/billing/
- Estimate your BU needs: https://research.csc.fi/billing-and-monitoring
- You can see your project's BU saldo from **my.csc.fi**

Batch script format (SLURM)

Contents of a batch script

- Create an empty text file with the **.sh** extension
- First line **#!/bin/bash** identifies the file as a bash script
- Everything starting with **#SBATCH** is passed to the **SLURM** as a parameter for the job
- Everything else starting with # is considered as a comment
- Everything else is executed as a command

Example my_batch_job.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
```

Contents of a batch script

- These are the batch job parameters
  - **--job-name** = name of the job
  - **--account** = account name that is to be billed
  - **--time** = how much time is reserved in hours
  - **--cpus-per-task/--ntasks** = how many CPU cores to reserve for the job
  - **--mem-per-cpu/--mem** = how much memory is reserved per CPU
  - **--partition** = what partition (type of queue) the job goes to. Puhti has several different partitions for different kind of jobs
- More parameters can be found here
  - https://docs.csc.fi/#computing/running/creating-job-scripts
- The different partitions can be found here
  - https://docs.csc.fi/#computing/running/batch-job-partitions

Example my_batch_job.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
```

Contents of a batch script

- After the **#SBATCH** parameters comes the commands you wish to execute
- First, load the module(s) you need, in this case **geoconda**
- Then run your application which in this case is a **python script**
- It is a good idea to write full file paths for referenced files which should usually reside in you project's directory

Example my_batch_job.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
```

Submitting the batch job

- Navigate to the folder where you created the **my_batch_job.sh** and submit your job to SLURM with command
  - **`sbatch my_batch_job.sh`**
- You can check the status of your jobs with
  - **`squeue -u <username>`**
- You can also cancel a job with
  - **`scancel <jobid>`**

Displaying the used resources (coming soon to Puhti)

- You can see how well your job used the resources it was reserved with the command **seff <jobid>**

```
[erkki@taito]$ seff 52000798_6
Job ID: 52000798
Cluster: csc
User/Group: erkki/csc
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 4
CPU Utilized: 00:24:09
CPU Efficiency: 98.17% of 00:24:36 core-walltime
Memory Utilized: 23.50 MB
Memory Efficiency: 1.15% of 2.00 GB
```

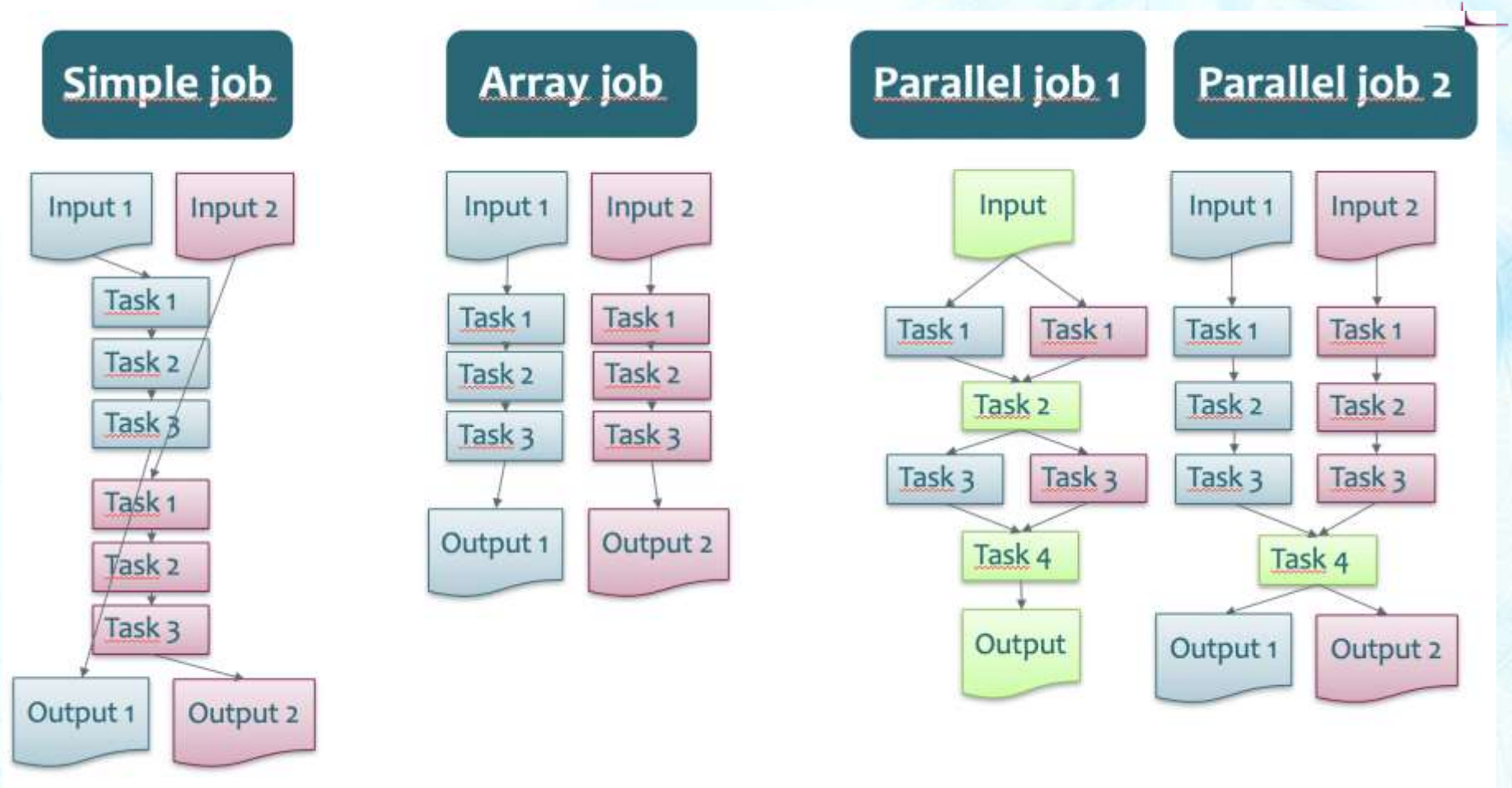
Studying past jobs – sacct command

- Command **`sacct`** can be used to study *past* jobs
- Useful when deciding proper resource requests

command	description
`sacct`	List today's jobs
`sacct -l`	List today's jobs with long format
`sacct -j <jobid>`	List single job
`sacct -S YYYY-MM-DD`	List jobs since a specific date
`sacct -u <username>`	List jobs by username
`sacct -o`	List only named data fields from long format

```
$ sacct -o jobid,jobname,maxrss,reqmem,elapsed -j <jobid>
```

Array and parallel jobs


Simple job
Input 1
Input 2
Task 1
Task 2
Task 3
Task 1
Task 2
Task 3
Output 1
Output 2
Array job
Input 1
Input 2
Task 1
Task 2
Task 3
Task 1
Task 2
Task 3
Output 1
Output 2
Parallel job 1
Input
Task 1
Task 1
Task 2
Task 3
Task 3
Task 4
Output
Parallel job 2
Input 1
Input 2
Task 1
Task 1
Task 2
Task 2
Task 3
Task 3
Task 4
Output 1
Output 2

Single job

- **Pros**: Simple and easy, just check paths and packages
- **Cons**: Slowest

Array job

- **Pros**: Relatively easy, just check paths and packages, and add using command line arguments. SLURM takes care of the job management
- **Cons**: Applicable only in certain cases

Parallel job

- **Pros**: All kind of use cases are possible
- **Cons**: Requires modifying code. More complicated code, because the job management is done in the code

Array jobs in GIS

- Array jobs in GIS could be separated by
  - different input files / map sheets
  - different scenarios
  - different variables
  - different time periods

- Suits well if the jobs are independent of each other, for example raster calculator
- In many cases if using map sheets the borders need special care

Submitting an Array job to SLURM

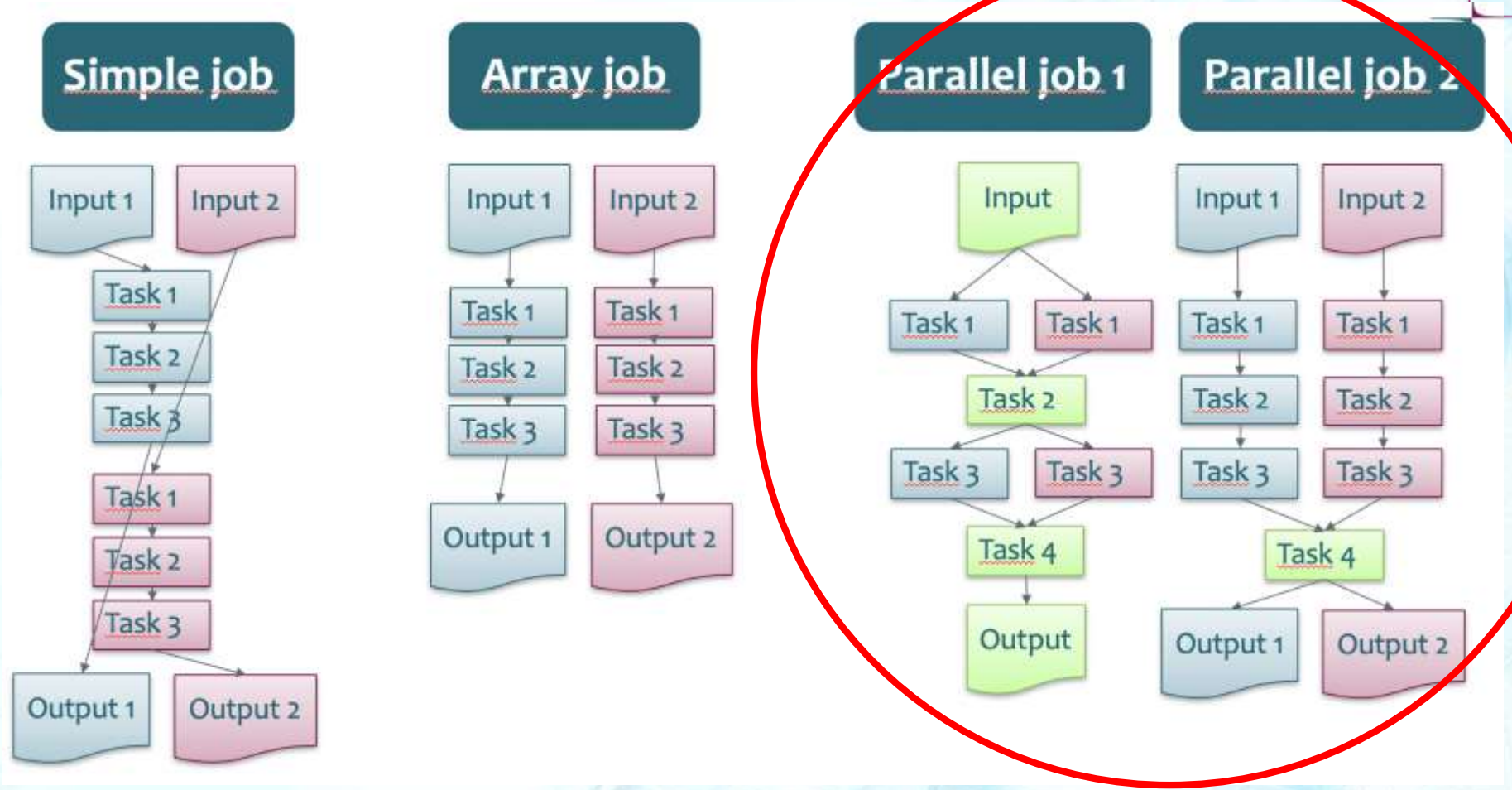
- Add **#SBATCH --array=1-100** to the batch job script
- **#SBATCH -o array_job_out_%A_%a.txt** for separate output file of each job
- **#SBATCH -e array_job_err_%A_%a.txt** for separate error file of each job
- This defines to run 100 jobs, where a variable **$SLURM_ARRAY_TASK_ID** iterates through 1-100
- Make a file with all the input files, for example mapsheets.txt
- Read file names row by row:
  - name=$(sed -n "$SLURM_ARRAY_TASK_ID"p ../mapsheets.txt)
- Run your script for each input file:
  - **srun myprog $name**
- This would run 100 jobs where the execution line in batch job script would be

  srun myprog file_1

  srun myprog file_2

Array job example

```
#!/bin/bash -l
#SBATCH --job-name myAwesomeArrayJob
#SBATCH --output job_out_%A_%a.txt
#SBATCH --error ob_err_%A_%a.txt
#SBATCH --account=<project>
#SBATCH --partition small
#SBATCH --time 00:30:00
#SBATCH --ntask 1
#SBATCH --mem-per-cpu=4000
#SBATCH --array=1-50 # set the input file to process

ls map_sheet_num_* > namelist
name=$(sed -n ${SLURM_ARRAY_TASK_ID}p namelist)

python myPythonScript.py ${name}
```


Simple job
Input 1
Input 2
Task 1
Task 2
Task 3
Task 1
Task 2
Task 3
Output 1
Output 2
Array job
Input 1
Input 2
Task 1
Task 2
Task 3
Task 1
Task 2
Task 3
Output 1
Output 2
Parallel job 1
Input
Task 1
Task 1
Task 2
Task 3
Task 3
Task 4
Output
Parallel job 2
Input 1
Input 2
Task 1
Task 1
Task 2
Task 2
Task 3
Task 3
Task 4
Output 1
Output 2

Parallel jobs for GIS

- For parallel computing to work, you have to divide your GIS data to chunks in the code an run a function simultaneously to several chunks, gather results and combine them
- You have to use a library/application that can utilize several cores
- Difficulties emerge if the simultaneously running functions need access to same data slot in memory
- Chunk size shouldn't be too small because then you might lose efficiency to communication and data processing (should be at least > 10min per task)
- Big chunk size again requires more memory and does not utilize parallel computing as much

Parallel computing with GIS software

- Out-of-box parallel: **Taudem**
- Out-of-box something in parellel: **R, Python, GRASS, SagaGIS**

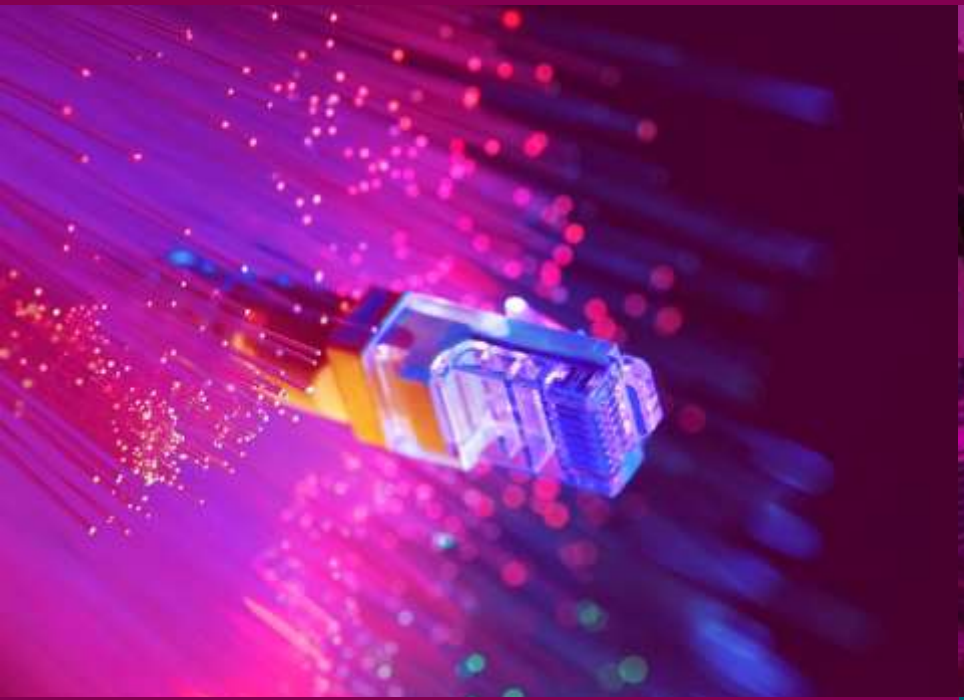
- Parallel packages for Python: **multiprocessing, joblib, dask**
- Parallel packages for R: **snow, foreach, doMPI, Rmpi**

Be careful with simultaneous opening of files

- If you are using the same input or output files in array or parallel jobs, depending on software it might cause trouble

- Solutions:
  - Make copies of shared input files for each processes you use
  - Write output to smaller files first, join them later

R in Taito



R spatial

- R strong sides
  - Statistical analysis
  - Raster analysis
  - Time-series analysis
  - Point clouds
  - Faceted maps
  - Repeating workflows
  - Usage of external functions: SagaGIS, GRASS etc
  - Rstudio
  - Several packages for parallel computation
- R weak sides
  - Routing
  - 3D vector models
  - Interactiv map usage
  - Data editing
  - Different than GUI GIS software

R in Taito/Puhti, rspatial-env 1

This is an convinience module for loading easily spatial analysis related R tools. It loads the following software:

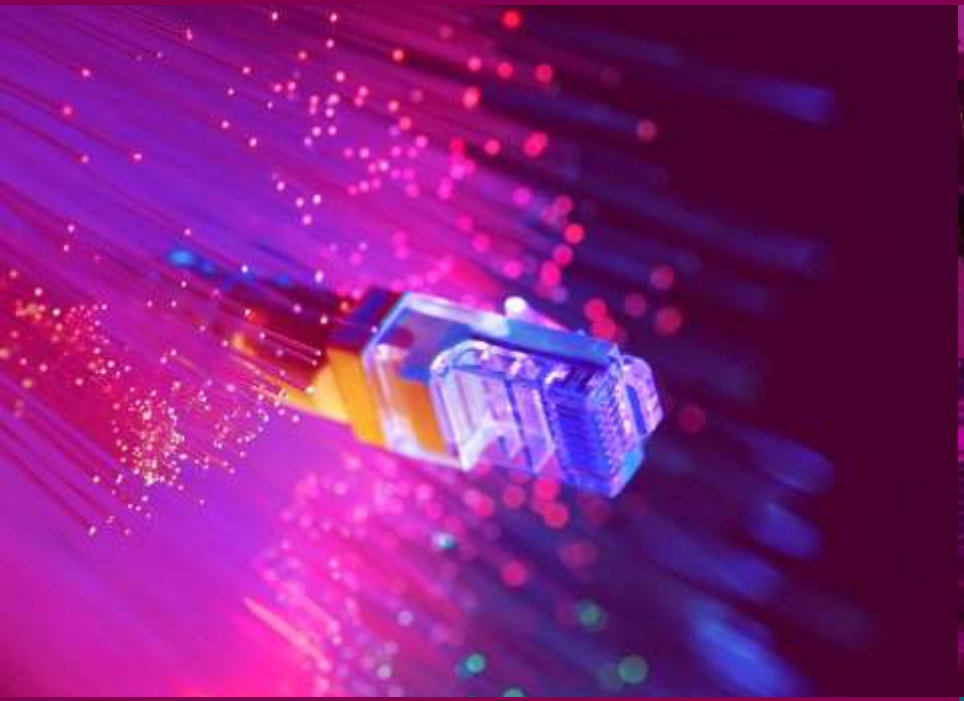
- Taito: module load rspatial-env
- Puhti: module load r-env (coming later)

- R with spatial packages:
  - geoR, geoRglm, geosphere, **ggmap**, grid, gstat, GWmodel, **lidR**, mapproj, maptools, ncdf4, RandomFields, **raster**, **rgdal**, rgeos, rgrass7, rlas **RSAGA**, **sf**, **sp**, spacetime, spatial, spatial.tools, spatstat, spdep, strucchange.
- In batch script start R script with:
  - srun Rscript --no-save your_script.R

Installing R libraries

- Everybody can add R libraries for personal use
  - These override the system ones
- Normally similar to local installations:
  - install.packages("pkgname")
  - or using Rstudio menus
- If the package needs some external library, the user installation is likely tricky.
- If you think that the package should be useful also for others or requires some external software to be installed, ask servicedesk@csc.fi

Running R in parallel — principles and practice



"My R code is slow… what can be done?"

- You should seek to have an understanding which part of your code is the time consuming one, and why, and how
  - the function **`system.time()`** can help
  - understanding the details of the method you are using helps
  - knowing how to run a smaller version of the problem helps a lot
  - understanding even the basics of time complexity helps
- 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

Level zero: not even parallel, just simultaneous

- Can you imagine going around in a computer classroom, firing up R on every computer and letting each one work on part of your problem?
  - such as each fitting the same model to 20 different data sets or with 20 different combinations of parameters
- Do you also have access to a cluster with R installed on it? Good! Then you can do the same without even getting up from your chair.
- Upside: really easy. Downside: limited usability

Array jobs in Taito

Array job with R, giving the mapsheet path as argument

```
#!/bin/bash
#SBATCH -J array_job
#SBATCH -o array_job_out_%j.txt
#SBATCH -e array_job_err_%j.txt
#SBATCH -t 00:02:00
#SBATCH --mem-per-cpu=4000
#SBATCH --array=1-3
#SBATCH -n 1
#SBATCH -p serial
```

```
module load rspatial-env

# move to the directory where the data files locate

cd ~/git/geocomputing/R/contours/array

# set input file to be processed

name=$(sed -n "$SLURM_ARRAY_TASK_ID"p ../mapsheets.txt)

# run the analysis command

srun Rscript Calc_contours.R $name
```

Array jobs with R, reading the argument in R script

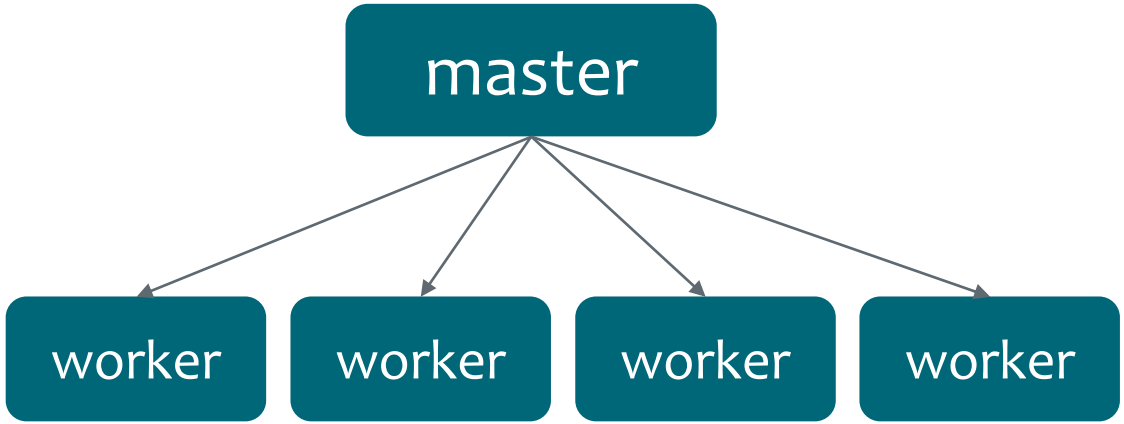
```
args = commandArgs(trailingOnly=TRUE)


if (length(args)==0) {
  stop("Please give the map sheet number", call.=FALSE)
} else if (length(args)==1) {
   mapsheet <- args[1]
}
```

Running R code in parallel

- There are several R packages for parallel computing:
  - parallel (snow)
  - foreach and doMPI
  - Rmpi

snow (parallel)

- This R package offers support for simple parallel computing in R, following the master - workers paradigm



parallel (snow), batch job file

The execution command at the end of the batch job file is:

#SBATCH --ntasks=8

srun RMPISNOW --no-save -f myrscript.R

parallel (snow), R script

The script should contain a getMPIcluster call which will produce the reference to the cluster that can be given to various other functions, like in this example:

```
library(snow)
cl<-getMPIcluster()
funtorun<-function(k){
  system.time(sort(runif(1e7)))
}
system.time(a<-clusterApply(cl,1:7,funtorun))
a
stopCluster(cl)
```

Only the master process continues to run the given script.

Python for GIS in Puhti

Conda environments as modules

- Conda environments are pre-installed groups of Python libraries that can be loaded at the same time
- In **Puhti** a conda **environment** is attached to one **module**
- There are several GIS related conda environments installed on Puhti
  - **Geoconda** – main environment for GIS with Python
  - **Lidarconda** – environment for tools related to Lidar processing
  - **Tensorflow/2.0.0** – environment for deep learning with some GIS libraries
  - **SNAP** – environment for European Space Agency's Snap python libraries
  - **Mapnik** –
- Read the **docs.csc.fi** for updated list of installed libraries in each environment!
- https://docs.csc.fi/#apps/geoconda/

Installing additional Python packages

- If you need additional libraries you can ask CSC to include them in the conda environments or install them to your user files

- Installation happens with pip
  - **pip install --user <library_name>**

- If you are planning to install many or big libraries you should install them to your project's folder (50GB) as the user folder is relatively small in size (10GB)
  - **pip install --prefix=/projappl/<your_project>/my_modules <library_name>**
  - **export PYTHONPATH=$PYTHONPATH:<previous_my_modules_folder>/lib/python3.7/site-packages**

Running python code with a simple batch job

```
#!/bin/bash -l
#SBATCH -J python_serial
#SBATCH -o output_%j.txt
#SBATCH -e error_%j.txt
#SBATCH --mem-per-cpu=40
#SBATCH --ntasks=1
#SBATCH -t 00:02:00
#SBATCH -p test

module load geoconda
srun python python_serial.py
```

Running Python in Puhti in parallel

Python and array jobs

- Python and array jobs work very similar compared to R
  - Divide your data to different files (map sheets, satellite images)
  - Submit an array job that makes a list of the input files and feeds them one by one to different batch jobs
  - Merge the multiple output files

Running python code with a array job

```
#!/bin/bash
#SBATCH -J python_array
#SBATCH -o array_job_out_%A_%a.txt
#SBATCH -e array_job_err_%A_%a.txt
#SBATCH --mem-per-cpu=40
#SBATCH --cpus-per-task=1
#SBATCH --ntasks=1
#SBATCH -t 00:02:00
#SBATCH -p test
#SBATCH --array=1-3

module load geoconda
name=$(sed -n "$SLURM_ARRAY_TASK_ID"p ../mapsheets.txt)

srun python array_job.py $name
```

Parsing the input arguments in Python

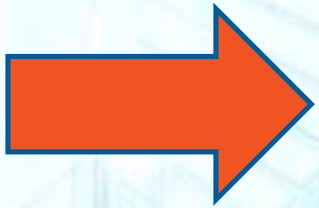
```
import sys

### sys.argv returns a list of the input arguments
list_or_arguments = sys.argv

### The 0th of that list is always the script's name! [1] is the first input argument
mapsheet_filepath = list_or_arguments[1]
other_input_variable = list_or_arguments[2]
```

Parallel python - libraries

- Python has plenty of parallelization libraries and even one included natively called **multiprocessing**
- Other quality libraries are also available:
  - **Joblib** – maybe a bit easier to use, uses multiprocessing
  - **Dask** – excellent if working with Pandas dataframes
  - **PySpark** – for running Python on a Spark cluster (not for Puhti)

Library multiprocessing – Pool example

```
from multiprocessing import Pool
import time
from random import random

def worker(num):
    time.sleep(random())
    print('Worker:', num, "reporting in!")
    return

def main():
    p = Pool(5)
    p.map(worker, [0, 1, 2, 3, 4])
```



Output print

```
/Users/jnyman/miniconda3/bin/python3.7
Worker: 4 reporting in!
Worker: 0 reporting in!
Worker: 3 reporting in!
Worker: 1 reporting in!
Worker: 2 reporting in!
```

- Script uses map (runs the function for each value in list) to run worker-function
- Easier to pass various kind of data to the function
- No need for looping

Running python code as a parallel job

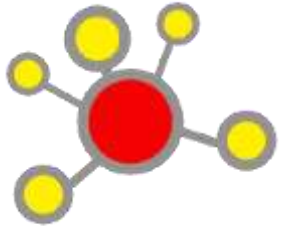
```
#!/bin/bash -l
#SBATCH -J py_exercise
#SBATCH -o out_%j.txt
#SBATCH -e err_%j.txt
#SBATCH --mem-per-cpu=4000
#SBATCH -t 00:02:00
#SBATCH --cpus-per-task=3
#SBATCH -p test

module load geoconda
srun python parallel_job.py
```

Example code in CSC training Github

- Spatial analysis, with batch job scripts suitable for **Taito**. Examples for serial, array and parallel jobs
  - Python: geopackage usage
  - R: lidar data, foreach parallel example and geopackage usage
  - PDAL
  - Routing

- Also cPouta examples: GeoServer, PostGIS, OpenDroneMap…

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



Further information

- ServiceDesk: servicedesk@csc.fi
- Accounts, projects, forgotten password: https://my.csc.fi
- Taito documentation: https://research.csc.fi/
- Puhti documentation: https://docs.csc.fi/
- GIS example code: https://github.com/csc-training/geocomputing