# HIGH PERFORMANCE COMPUTING for SCIENCE & ENGINEERING (HPCSE) I

HS 2022

## TUTORIAL 01: BASICS

Athena Economides

**C**omputational **S**cience and **E**ngineering **Lab**
**ETH Zürich**

21.09.2022

# Outline

I. Course organization: Moodle / website
II. Git
III. Makefiles
IV. Euler Cluster: usage
V. C++ refresher

# I. Course Intro & Organization

## Teaching Assistants



- Athena: eceva@ethz.ch
- Michalis: michaich@ethz.ch
- Noah: noabauma@student.ethz.ch

## Course Webpage

https://www.cse-lab.ethz.ch/teaching/hpcse-i-hs22/

## Ask questions!

➡ Email us for administrative issues

➡ All other questions/ inquiries: Moodle

➡ Office hours: Tuesdays 9:00-10:00, Thursdays 12:00-13:00 (**CLT D11**)

## Homeworks

Not obligatory, can add *BONUS* to your final grade

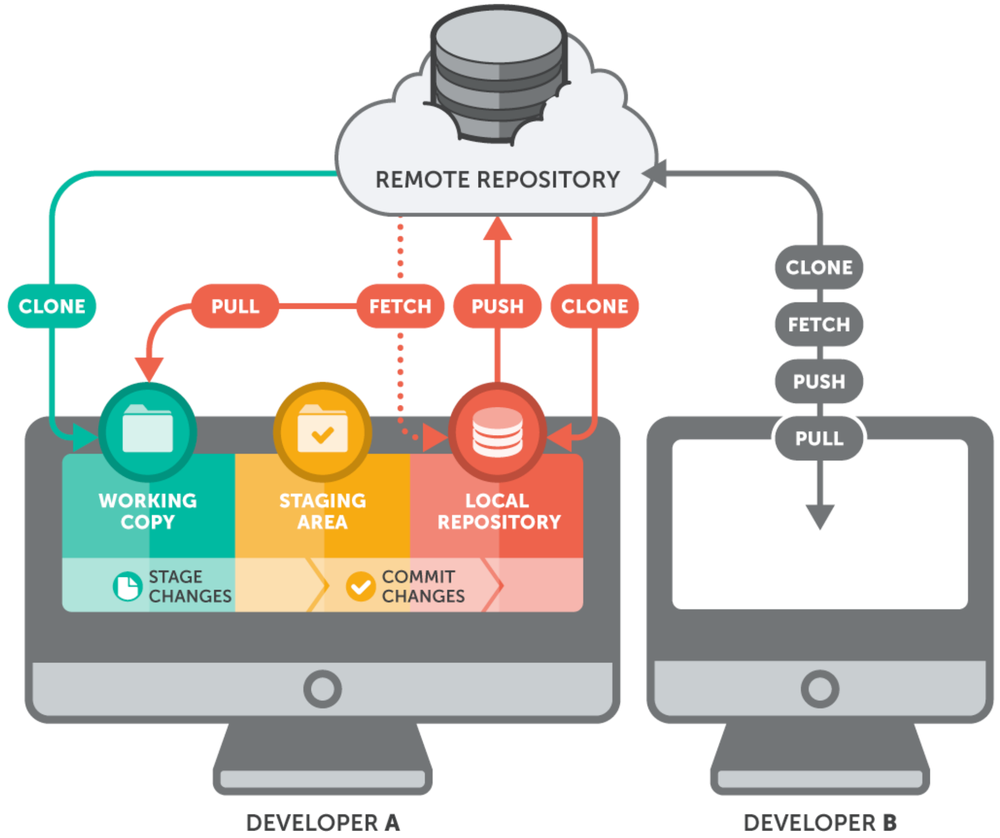
final grade $= \max\{\text{exam grade}, (20\,\% \text{ hw grade} + 80\,\% \text{ exam grade})\}$

Submit homework 1-6 via the Moodle ETHZ app:
https://moodle-app2.let.ethz.ch/course/view.php?id=18039

# II. Git repository

COURSE REPOSITORY https://gitlab.ethz.ch/hpcse-public-repos/hpcsei-fall-2022-lecture.git

How does **Git** work?




Why **Git**?

- On of the most popular version control systems (VCS) today!
- One can save their work and work safely remotely and concurrently between multiple users
- ETH provides access to gitlab.ethz.ch to all ETH members
- Available on ETH clusters & Linux environments

HOW TO USE **Git**

https://git-scm.com
https://git-scm.com/doc

HOW TO INSTALL **Git**

https://github.com/git-guides/install-git

```
git clone https://gitlab.ethz.ch/hpcse-public-repos/hpcsei-fall-2022-lecture.git
```

```
git pull
```

# III. Makefiles

compiler

CFLAGS: compiler directives/ options

targets

Possibly LDFLAGS: list of link directives

TAB!

```
CC=g++
CFLAGS=-O3 -Wall -Wextra -Wpedantic

all: factorial

factorial.o: factorial.cpp
	$(CC) -c -o factorial.o factorial.cpp $
(CFLAGS)

factorial: factorial.o
	$(CC) -o factorial factorial.o

clean:
	rm -f *.o *~ factorial

.PHONY: all
.PHONY: clean
```

prerequisites

Compile commands

*Convert .cpp code to object code*

*-c: generate object file*

*-o: put the compilation in the file factorial.o*

*Link the object code together into executable*

Makefile 1 for factorial.cpp

COMMON MAKEFILE CFLAGS

- `O3` : optimize generated code (`O0`: not optimise code)
- `g` :compile with debug information
- std=c++11 use the c++11 standard language definition
- `I$(DIR)` : include $(DIR) containing auxiliary header files, other than /usr/include
- `Wall` : give verbose compiler warnings, include common warnings
- `Wextra` : enable extra warnings
- `Wpedantic` : turns off extensions and generates more warnings

USAGE

`make <target>` : execute a specific target

`make clean` : remove all object and executable files

`make` : will execute the first target in the file

LDFLAGS EXAMPLES

- `LDFLAGS=-lm` : link the C math library
- `LDFLAGS=-L../lib -L<DIR>`: for other <DIR> than /usr/lib

# III. Makefiles

Dependency files
- all include files

```
CC=g++
CFLAGS=-O3 -Wall -Wextra -Wpedantic
OBJ=factorial.o
DEPS=factorial.hpp

all: factorial

%.o: %.cpp $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

factorial: $(OBJ)
	$(CC) -o $@ $^ $(CFLAGS)

clean:
	rm -f *.o *~ factorial

.PHONY: all
.PHONY: clean
```

Makefile 2 for `factorial.cpp`, where `factorial(int n)` is a function that calculates the factorial of an integer number.

Also in the directory, there exists a header file: `factorial.hpp`, with function declaration of `factorial(int n)`

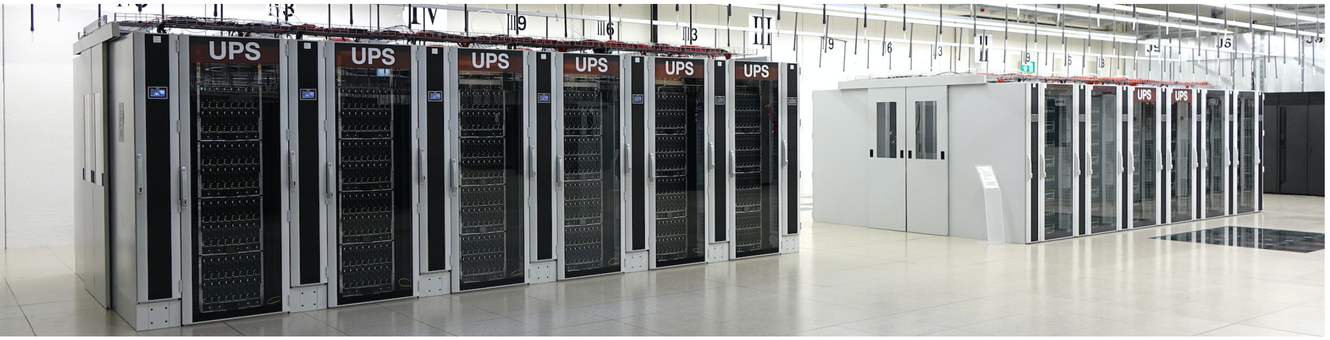
Suffix replacement rule:
- `.o` file depends upon the `.cpp` version of the file and the `.hpp` (`DEPS`) files
- To generate the `.o` file, compile the `.cpp` file using the (`CC`) compiler.
- `-o $@` : put the compilation in the file named on the left of : i.e. `%.o`
- `$<` : first item in dependencies list

General compilation rule:
- `$@` `$^` : macros for the left and right sides of :, i.e. here: `$@=factorial` and `$^=(OBJ)`

MAKEFILE TUTORIALS

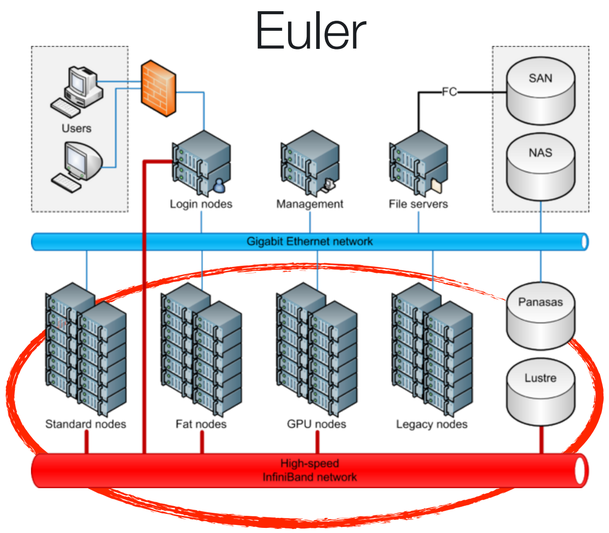
https://www.gnu.org/software/make/manual/make.html#Makefiles
https://cs.colby.edu/maxwell/courses/tutorials/maketutor/
https://web.stanford.edu/class/archive/cs/cs107/cs107.1194/resources/make

# IV. Euler cluster : usage




## Details

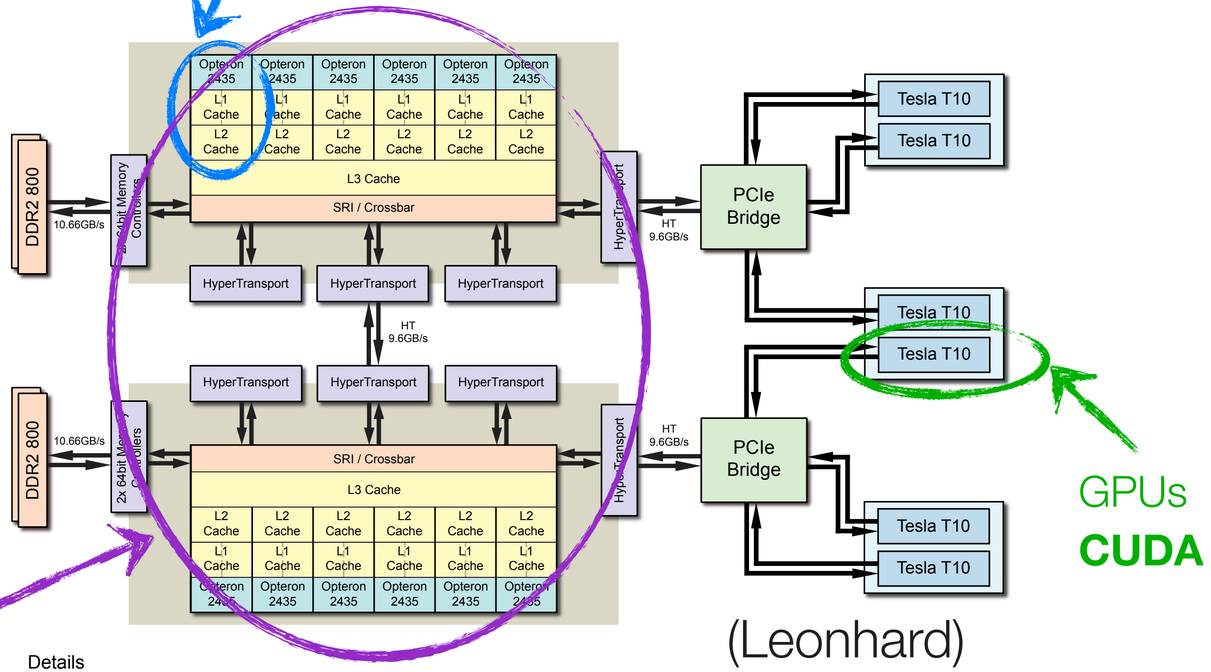
https://scicomp.ethz.ch/wiki/Euler




Cluster: network of nodes
Distributed memory
**MPI**

Node: multiple processors
Shared Memory
**C++ threads, OpenMP**

CPU core
**C++, SIMD**



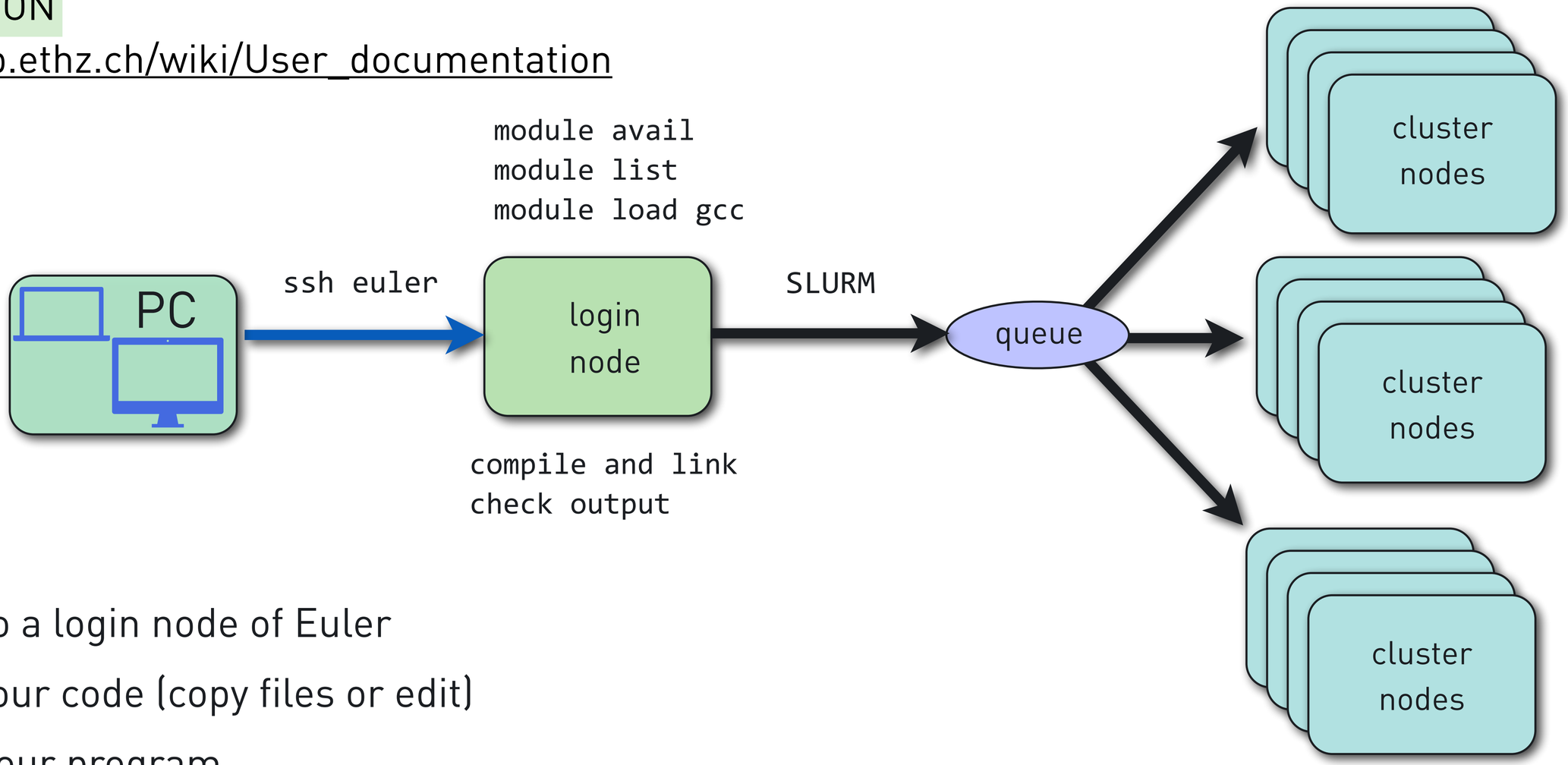

GPUs
**CUDA**

Details
- Effective bandwidth with 12 cores: 20GB/s (STREAM Benchmark)

# IV. Euler cluster : usage

## DOCUMENTATION

https://scicomp.ethz.ch/wiki/User_documentation




## STEPS

1. Connect to a login node of Euler
2. Develop your code (copy files or edit)
3. Compile your program
4. Submit a job / run your program on compute nodes
5. Check your job (status and output)

# IV. Euler cluster : usage

## 1. Connect

- `ssh -Y <usename>@euler.ethz.ch`
  - -Y: Enables trusted X11 forwarding
  - Access to one of the Euler login nodes
  - https://scicomp.ethz.ch/wiki/Accessing_the_clusters#SSH to set up SSH keys to login without typing password every time.

## 2. Develop

- Copy your files to Euler, e.g.
  - `scp code.tar.gz <username>@euler.ethz.ch:code.tar.gz`
  - `scp -r <local_DIR> <username>@euler.ethz.ch:/cluster/scratch/<username>`
  - better: use Git!
- Use a text editor to write/modify your code, i.e. vi(m), emacs, nano
- OR mount your $HOME folder locally
  - `sshfs <username>@euler.ethz.ch:/cluster/home/<username>/<local_DIR_Euler>`
  - https://www.digitalocean.com/community/tutorials/how-to-use-sshfs-to-mount-remote-file-systems-over-ssh

# IV. Euler cluster : usage

## 3. Compile

- You need the appropriate programming tools and libraries to compile your code
- Just load the environment module you need
- Examples:
  - `module load gcc` (newer version of gcc)
  - `module load open_mpi` (MPI library)
  - `module list` (shows loaded modules)
  - `module avail` (what is available)
  - `module unload gcc` (unloads a module)
  - Compile your code and produce the executable
  - Example:
    - `g++ cputest.cpp -o cputest`
  - Use Makefiles!

# IV. Euler cluster : usage

https://scicomp.ethz.ch/wiki/LSF_to_Slurm_quick_reference#Job_status
https://user.cscs.ch/access/running/

## 4. Submit your job

- login nodes are only for development
- The program must run on a compute node
- To do that, you must use the sbatch command: `sbatch jobscript.sh`
- `jobscript.sh`: a batch script containing options of the job
- You can also get an *interactive shell* on a compute node
  `srun --time=00:10:00 --pty bash`

Batch script generated using Euler's Slurm Submission Line Advisor

https://scicomp.ethz.ch/public/lsla/index2.html

`jobscript.sh`

```
#!/bin/bash

#SBATCH -n 4
#SBATCH --time=00:10:00
#SBATCH --job-name=test
#SBATCH --mem-per-cpu=512
#SBATCH --output=file.out
#SBATCH --error=file.err

srun ./hello_world_mpi
```

number of processor cores
max execution time HH:MM:SS
name of job in queue
requested memory per cpu
file containing program output
file containing standard error

run program

# IV. Euler cluster : usage

https://scicomp.ethz.ch/wiki/LSF_to_Slurm_quick_reference#Job_status
https://scicomp.ethz.ch/public/lsla/index2.html
https://user.cscs.ch/access/running/

## 5. Check your job

- Useful commands
  - **`sbatch`** - submit a batch script
  - **`squeue`** - check the status of jobs on the system
  - **`scancel`** - delete one of your jobs from the queue

- Output files
  - **`slurm-<JOBID>.out`**: contains output of the program.
  - **`slurm-<JOBID>.err`**: contains any error messages (standard error).

  both files will be found in the directory from which you launched the job.

# Simplifying our workflow

## aliases

- Create an *alias* to contain the SSH command:

```
$ alias EU="ssh -Y eceva@euler.ethz.ch"
$ EU
```

- Password-less SSH:

```
$ alias EU="ssh -Y eceva@euler.ethz.ch -i ~/.ssh/euler_ssh_id"
$ EU
```

# V. C++ refresher

We want to calculate and print the factorials of all numbers up to one non-negative integer number $N$.

The factorial of each number $n \leq N$ is calculated as:

$$n! = n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1 \quad \text{and} \quad 0! = 1$$

Simplest recurrent algorithm

```
int product = 1;
for(int n = 0; n <= N; ++n)
{
    if (n != 0)
    {
        // calculate the next factorial
        product *= n;
    }
    // output the factorial
    printf("%3d  %15d\n",n,product);
}
```

Hands-on c++ examples with:

- Different print, output, input utilities
- Function declarations
- Header files
- Template functions
- Class declaration

In git: tutorials/tutorial01_Intro/

C++ REFERENCE https://en.cppreference.com/w/

C++ TUTORIALS https://www.tutorialspoint.com/cplusplus/
https://www.geeksforgeeks.org/c-plus-plus/?ref=lbp

C++ BOOK https://polybox.ethz.ch/index.php/s/yGjq4w0xBYsCRKT