ZICE 17 Onboarding

Luca Mazzone & Simon Scheidegger
Jan. 24th, 2017
ZICE 17 – University of Zürich

alphacruncher

Outline

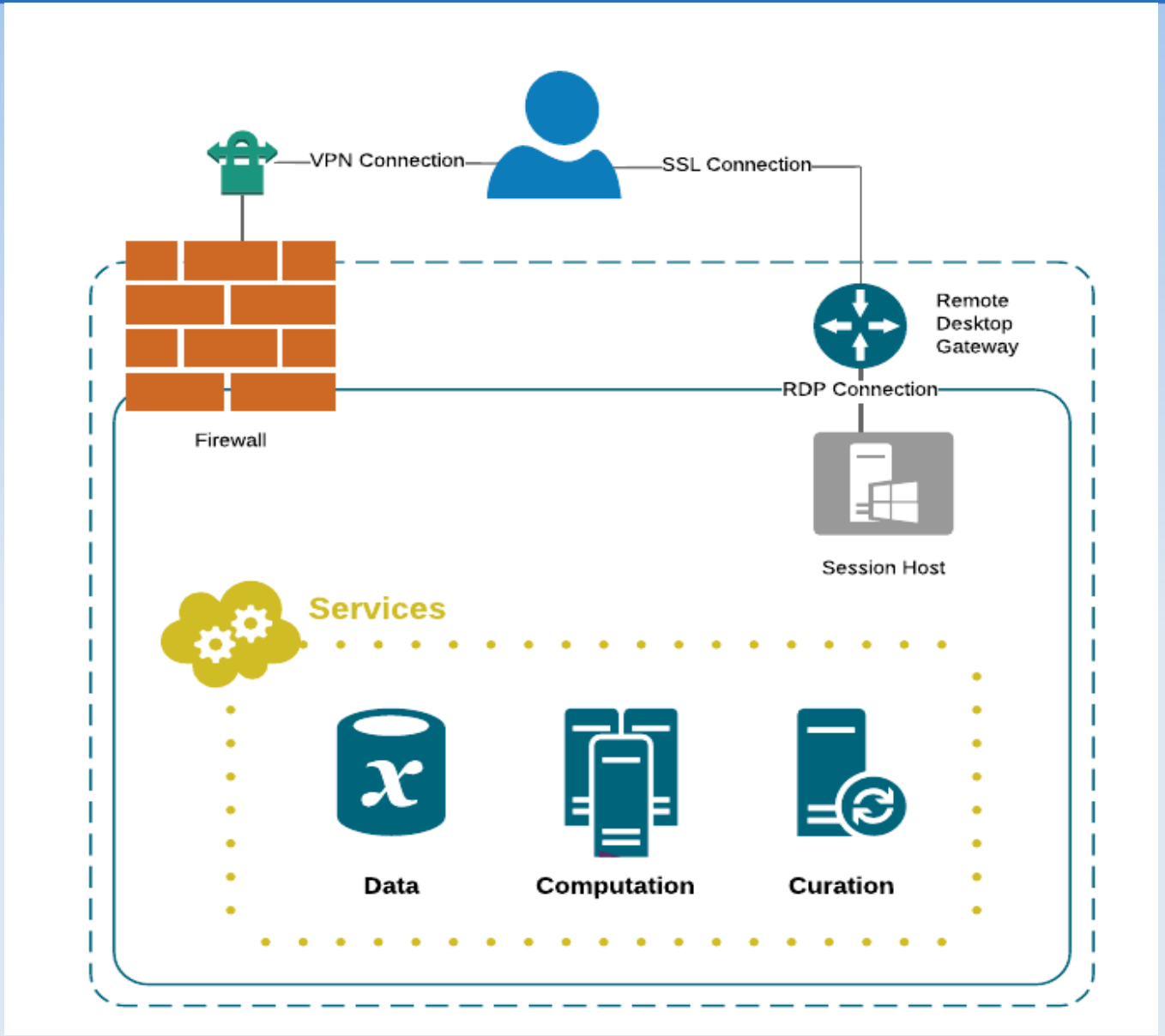
1. The compute infrastructure for ZICE
2. Accessing WLAN during ZICE
3. Access to ALPHACRUNCHER Services
4. Course management – GIT Classroom
5. First steps on a Linux Cluster

Aim

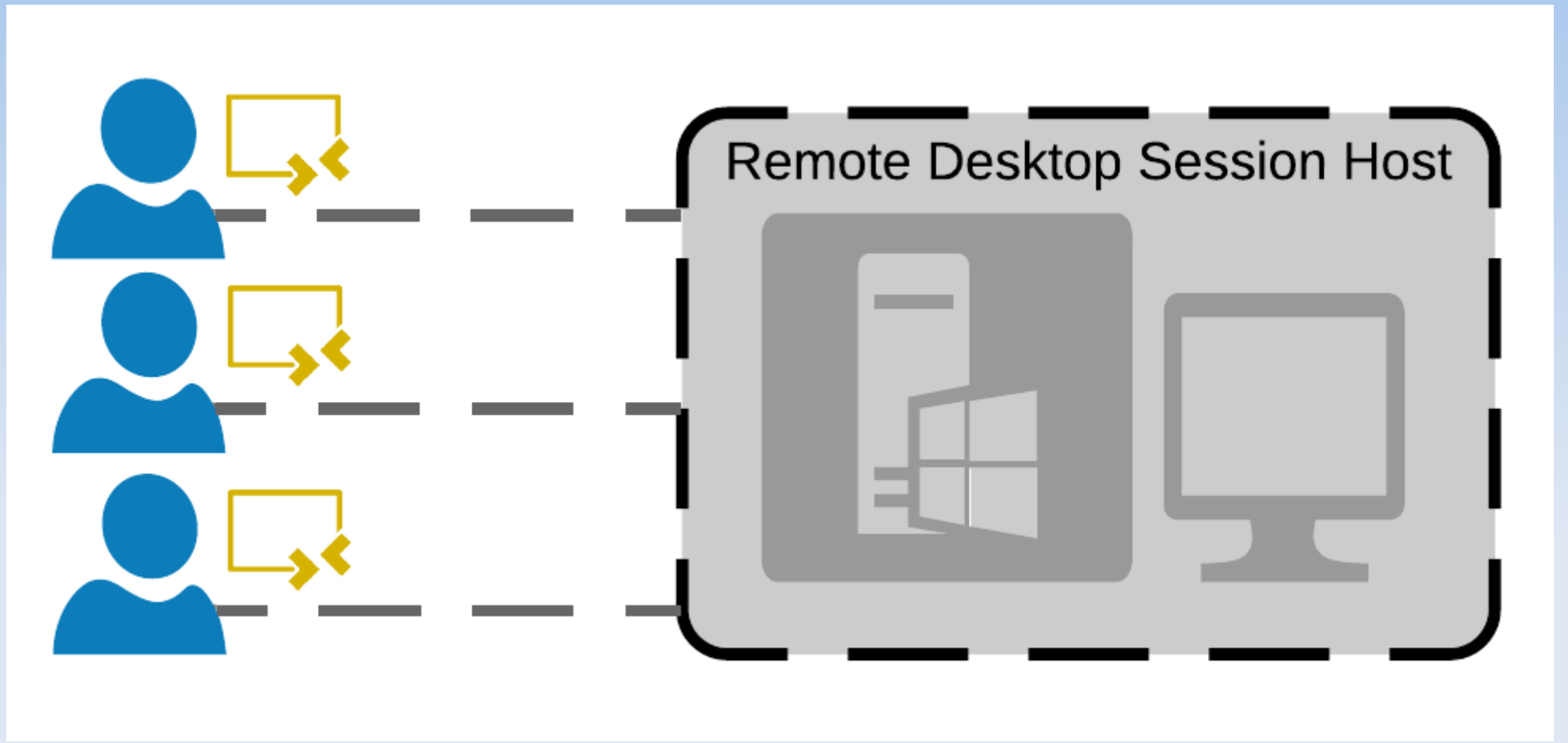
- Every student works on a unified environment (no issues with licences etc.)

- Service to the community → knowledge transfer: Access to resources for an extended period of time, even outside the UZH network.

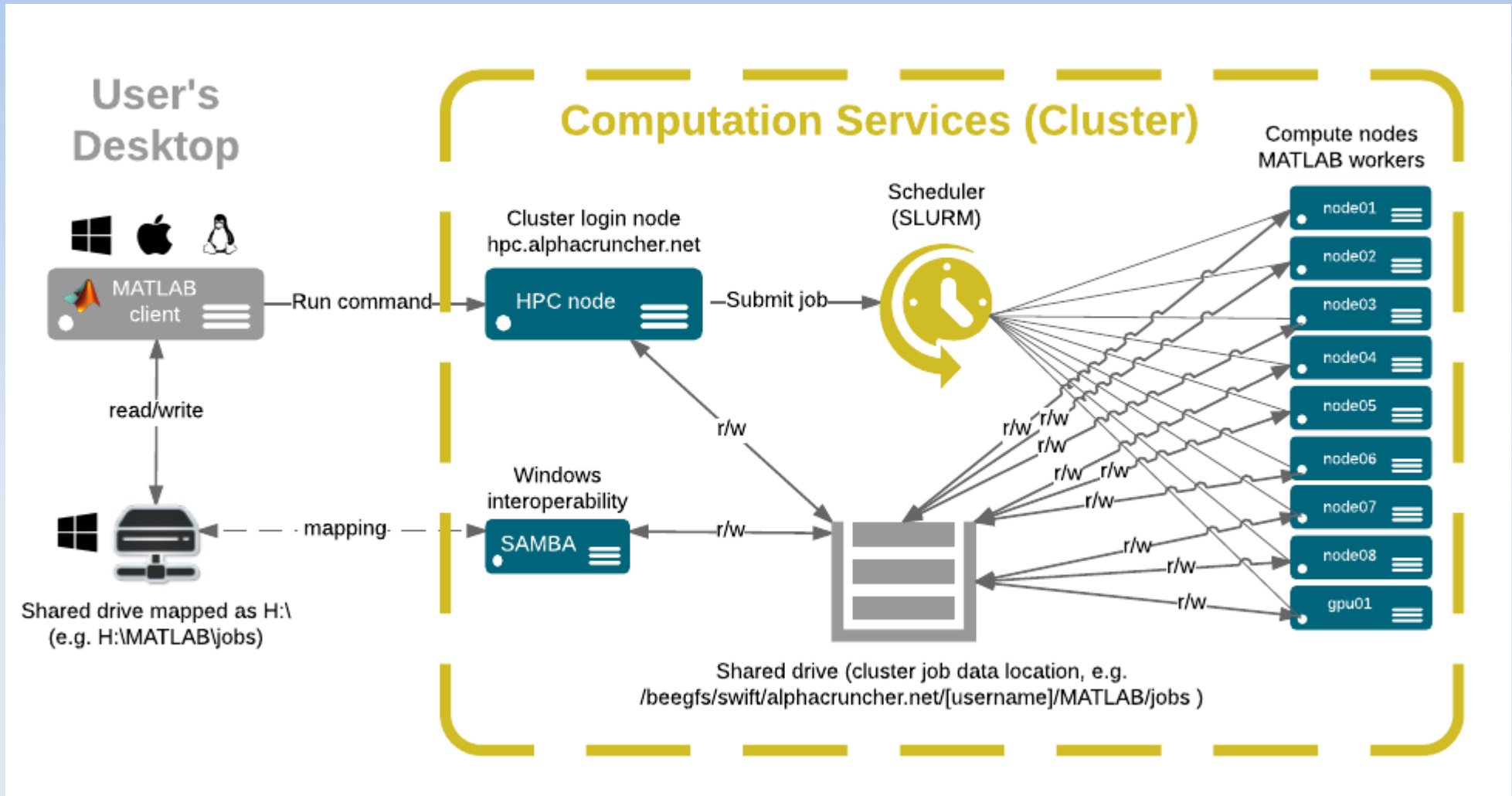
1. The compute infrastructure



Session-based desktop deployment



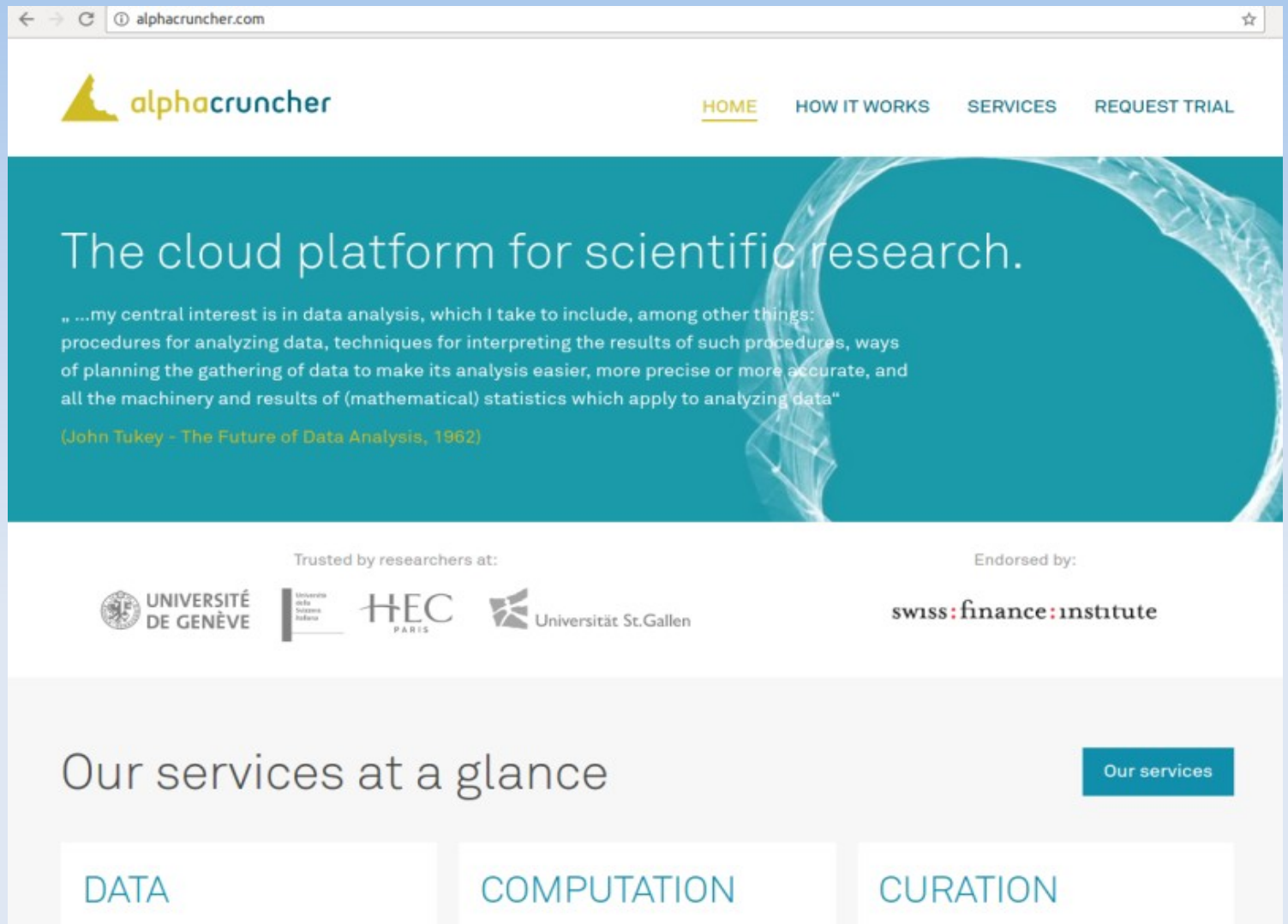
Compute Cluster



2. Accessing WLAN during ZICE

3. Access to Alphacruncher Services

http://alphacruncher.com/



Access to the remote desktop

- You all obtained an access guide
- Windows – Mac – Linux
- In the lecture notes

Help

4. Course management – GIT classroom

Access to the WLAN

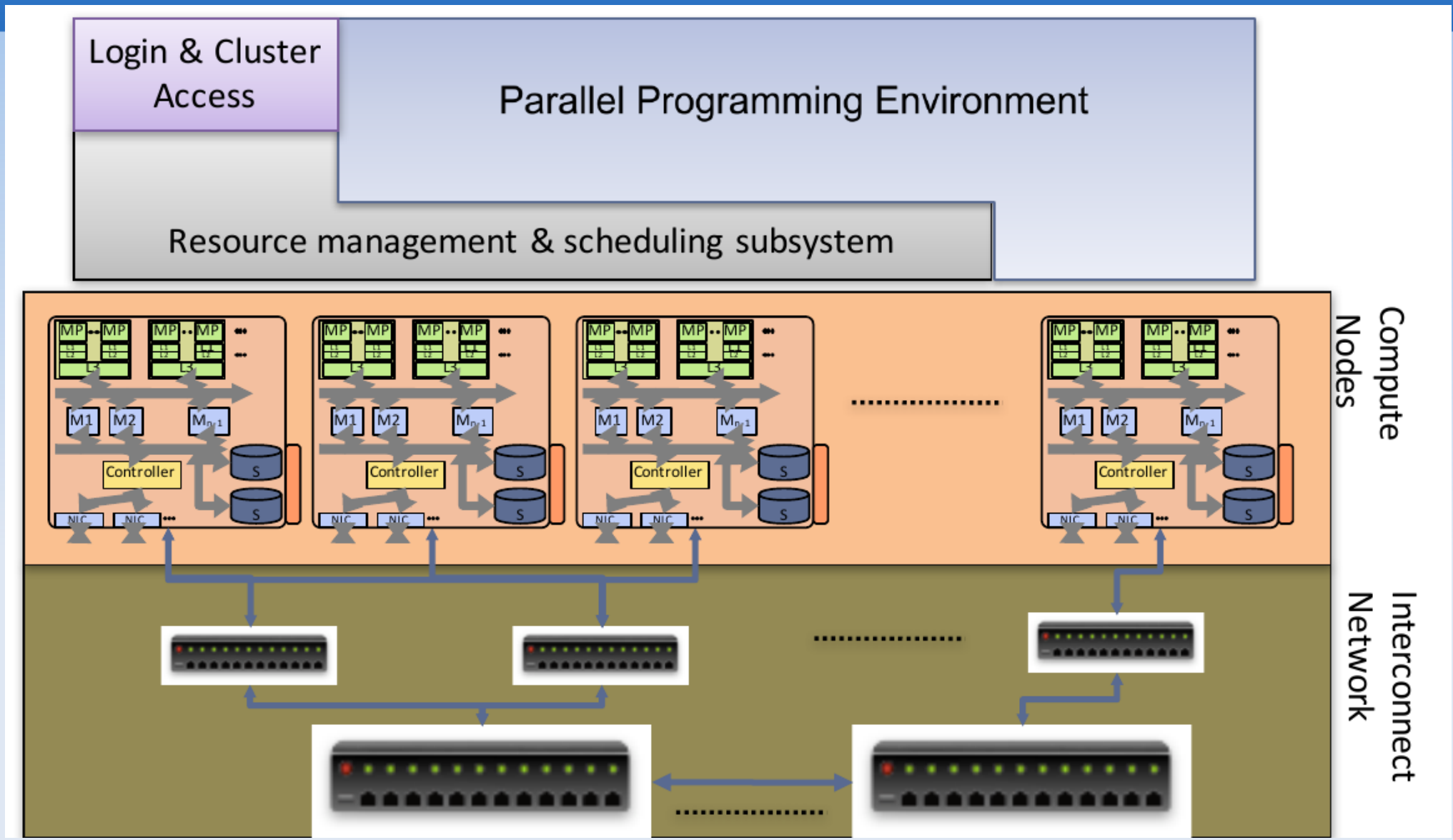
Outline

- Make first steps on a Linux Cluster
  Login via ssh, remotely, short overview of basic unix commands like cd, pwd, cp, scp,...

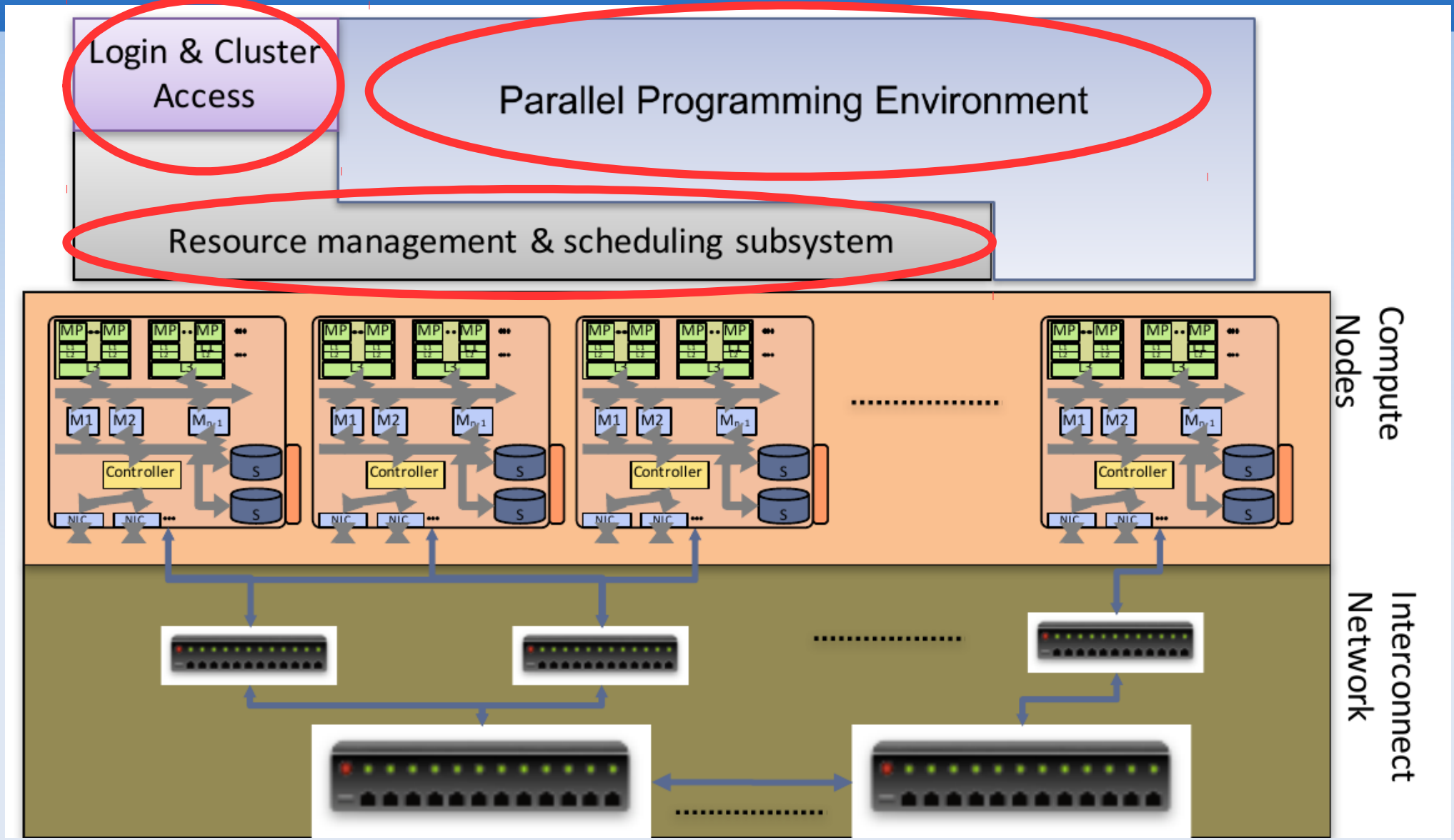
- Submit jobs to the queue
  Slurm

- Get lecture notes (https://github.com/edualphacruncher/zice17)
  Clone a git repository

A compute cluster



A compute cluster



Login for participants

- If you don't have an account on http://alphacruncher.com/
- request one **NOW**

- Log

- For MS-Windows users: Download and install Putty
- → http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe

- → Download and install Winscp

- → http://winscp.net/download/winscp576setup.exe

Basic Linux commands (1)

Command	Description
pwd	Print name of current/working directory
cd [Directory]	Change directory (no directory → change to home)
ls [Directory]	List directory contents (no directory → list current)
cat FILE	Concatenate files and print on the standard output
mkdir DIRECTORY	Make directories
mkdir -p DIRECTORY	Make directories, make parent directories as needed
cp SOURCE... DIRECTORY	Copy files and directories
cp -r SOURCE... DIRECTORY	Copy files and directories, copy directories recursively
mv SOURCE... DIRECTORY	Move (rename) files
man COMMAND	An interface to the on-line reference manuals

4. First steps on a Linux cluster

Basic Linux commands (2)

Command	Description
ssh -X foo@host.com	OpenSSH SSH client (remote login program), access to host.com with user foo
scp foo@host.com:/home/bar ./	Secure copy (remote file copy program), copy file bar from /home on host.com to directory
scp bar foo@host.com:/home/	Secure copy (remote file copy program), copy file bar from the local host to /home on host.com
git clone git@github.com:whatever folder-name	The stupid content tracker, Clone a repository (whatever) into a new directory (folder-name).
git checkout	Checkout a branch or paths to the working tree.

Intermezzo

```
$ echo 'The command line is awesome!' | cowsay
 _____________________________________
< The command line is awesome! >
 -------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
simon@simon-ThinkPad-T450s:~$
```

Step-by-Step* (1)

- Go to the web interface id.alphacruncher.net
- Change password if you haven't done so far.
- get lecture notes (MS-Windows: Putty, Linux/MacOS: Terminal)

```
> ssh -X USERNAME@hpc.alphacruncher.net
> git clone ***lecture-folder*** #clone lecture
> cd ***lecture_folder***  #go into folder
> ls # list content of folder
```

*alternatively, you can login via alphacruncher web interface. Above's method is generic for most HPC environments

Other clusters – Step-by-Step*

- First login, change password and get lecture notes (MS-Windows: Putty, Linux/MacOS: Terminal)

```
> ssh -X USERNAME@hpc.alphacruncher.net
> passwd  #Change password for USERNAME.
  (current) UNIX password:
  Enter new UNIX password:
  Retype new UNIX password:
  Password changed
> git clone ***lecture-folder*** #clone lecture
> cd ***lecture_folder***  #go into folder
> ls # list content of folder
```

*alternatively, you can login via alphacruncher web interface. Above's method is generic for most HPC environments

Step-by-Step (2)

→ **Perform some basic operations on the cluster**

```
> ssh -X USERNAME@hpc.alphacruncher.net
> pwd
/beegfs/swift/alphacruncher.net/USERNAME
> mkdir -p firstFolder/secondFolder
> ls
FirstFolder
> ls firstFolder
secondFolder
> cd firstFolder
> pwd
/beegfs/swift/alphacruncher.net/USERNAME/firstFolder
> ls
secondFolder
> exit
```

Step-by-Step (3)

- How to copy folders and files to your PC?
- MS-Windows, start WinSCP
  - → Host-Name: `hpc.alphacruncher.net`
  - → User: `USERNAME`
- Linux/MacOS, replace /YOUR-LOCAL-PATH/
  - → with /home/LOCAL-LOGIN-NAME/ for linux
  - → with /Users/LOCAL-LOGIN-NAME/ for MacOS

```
>scp -r USERNAME@hpc.alphacruncher.net:/beegfs/swift/alphacruncher.net/scheideggers/YOUR-LOCAL-PATH/
```

Step-by-Step (4)

- Copy folders and files from your notebook
- create a file named firstFile in firstFolder
- → MS-Windows: use WinSCP to copy the directory back
- → Linux/MacOS

```
>scp -r /YOUR-LOCAL_PATH/firstFolder/FILENAME USERNAME@hpc.alphacruncher.net:
```

```
>ssh -X USERNAME@hpc.alphacruncher.net
> ls
FILENAME
>cat FILENAME  #shows content of file
```

Environment setup

Supporting diverse user community requires supporting diverse tool sets (different vendors, versions of compilers, debuggers, libraries, apps, etc)

User environments are customized via modules system (or softenv)

```
> module avail #shows list of available modules

> module list  #shows list of modules loaded by user

> module load module_name #load a module e.g. compiler

> module unload module_name #unload a module
```

Example – environment setup

```
> vi ~/.bashrc
module load gcc
module rm gcc/6.1.0
module add openmpi/gcc/64/1.10.1
module load python/2.7.11
```

Using an editor on a cluster

- Compute clusters like alphacruncher's
- infrastructure have a variety of simple text editors
- available.

→ vi, vim

```
>vi helloworld.f90

     program test

     implicit none

     write(*,*) "hello"

     end  program
```

More low bandwidth editors

Depending on network and preference, you may want to use an editor without a graphical user interface; common options:

- vi/vim
- emacs
- nano

emacs:

Undo: C-_

Find/create file: C-x C-f

Save file: C-x C-s

Exit Emacs: C-x C-c

Quit: C-g

Two modes – insertion and command mode
Insertion mode begins upon an insertion
[ESC] returns to command mode

Command mode options:
:w save
:wq save and exit
:q exit as long as there are no changes
:q! exit without saving

Insertion:
i (insert before cursor)
a (append)

Deletion: x
Motion: h (left) k (up)
j (down) l (right)

Compiling & running code interactively*

→ go to lectures/day1/code_day1 → cd lectures/day1/code_day1

If your program is only in one file (a hello-world program, or any simple code that doesn't require external libraries), the compilation is straightforward:

```
> gfortran helloworld.f90 -o helloworld.exe #Fortran
```

```
> g++ helloworld.cpp -o helloworld.exe       #C++
```

Once you produced the executable, you can run it (serial code) by

```
> ./helloworld.exe

> hello
```

Example: ...

*later on, we will also will link in libraries as well as use optimization flags.

Compiling Code with a makefile

In case your program consists of many routines (files), compiling by hand gets very cumbersome

```
> gfortran  -o abc abc.f90 a.f90 b.f90 c.f90
```

→ **A makefile is just a set of rules to determine which pieces of a large program need to be recompiled, and issues commands to recompile them**

→ For large programs, it's usually convenient to keep each program unit in a separate file. Keeping all program units in a single file is impractical because a change to a single subroutine requires recompilation of the entire program, which can be time consuming.

→ When changes are made to some of the source files, only the updated files need to be recompiled, although all relevant files must be linked to create the new executable.

Compiling Code with a makefile (2)

Basic makefile structure: a list of rules with the following format:

target ... : prerequisites ...
<TAB> construction-commands

A **“target”** is usually the name of a file that is generated by the program (e.g, executable or object files). It can also be the name of an action to carry out, like “clean”.

A “prerequisite” is a file that is used as input to create the target.

```
# makefile : makes the ABC program
abc : a.o b.o c.o  ### by typing „make“, the makefile generates an executable denotes as „abc“
gfortran -o abc a.o b.o c.o
a.o : a.f90
        gfortran -c a.f90
b.o : b.f90
        gfortran -c b.f90
c.o : c.f90
        gfortran -c c.f90
clean :   ### by typing „make clean“, the executable, the *.mod as well as the *.o files are deleted
        rm *.mod *.o abc
```

Compiling Code with a makefile (3)

- By default, the first target listed in the file (the executable abc) is the one that will be created when the make command is issued.
- Since **abc** depends on the files **a.o, b.o** and **c.o**, all of the .o files must exist and be up-to-date. make will take care of checking for them and recreating them if necessary. Let's give it a try!
- Makefiles can include **comments** delimited by hash marks (#).
- A **backslash (\)** can be used at the end of the line to continue a command to the **next physical** line.
- The make utility **compares** the modification time of the target file with the modification times of the prerequisite files.
- Any prerequisite file that has a more recent modification time than its target file forces the target file to be recreated.

- → A lot more can be done with makefiles (beyond the scope of this lecture)

Slurm Workload Manager

http://slurm.schedmd.com/

Simple Linux Utility for Resource Management (SLURM).

Open-source workload manager designed for Linux clusters of all sizes.

Provides three key functions:

1) It allocates exclusive and/or non-exclusive access to resources (computer nodes) to users for some duration of time so they can perform work.

2) It provides a framework for starting, executing, and monitoring work (typically a parallel job) on a set of allocated nodes.

3) It arbitrates contention for resources by managing a queue of pending work.

```
> sbatch submit_helloworld.sh (submit job)
> squeue -u NAME  (status of job)
> scancel JOBID      (cancel job)
```

Run an executalbe with „slurm“

```
#!/bin/bash -l

#SBATCH --ntasks=1    ## how many cpus used here

#SBATCH --time=01:00:00  ## walltime requested

#SBATCH --output=slurm_test.out ## output file
#SBATCH --error=slurm_test.err    ## error


### executable
./helloworld.exe
```

Try on ALPHACRUNCHER: ...

Clone a git repository*

```
>ssh -X USERNAME@hpc.alphacruncher.net
```

> git clone …. # clone the git repository
> cd .. # go into the repository
> ls … # check that all is there

*we will learn how to use git as a toolkit to manage software etc. later this week.


FUN
AHEAD