Parallel and High Performance Computing

Lecture 2: Development and execution in an HPC environment

March 1, 2021

Outline

Today exercise session

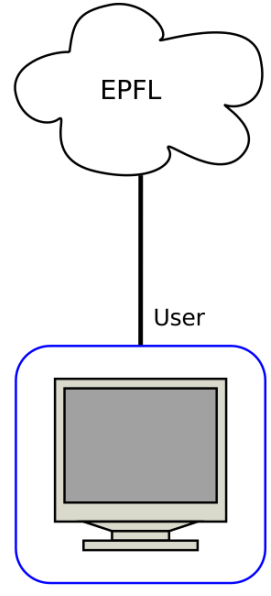
- What is a cluster.
- How to connect to a cluster (any remote machine).
- How to compile codes and use libraries.
- How to submit job on a cluster.
- How to use GIT

Getting your accounts

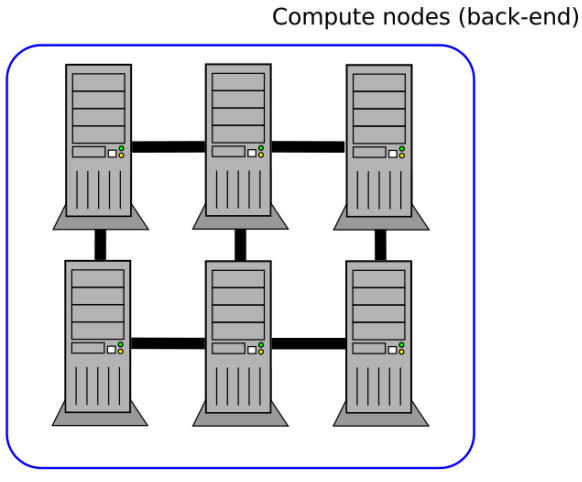
groups.epfl.ch

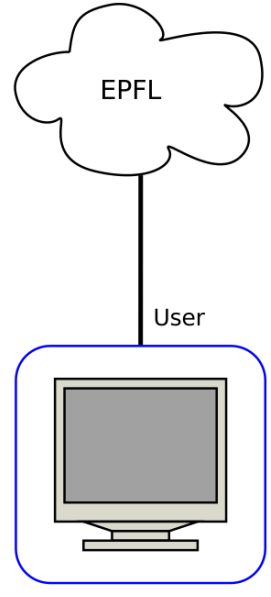
If you are not in the group phpc-2021 let use now

What is a cluster?

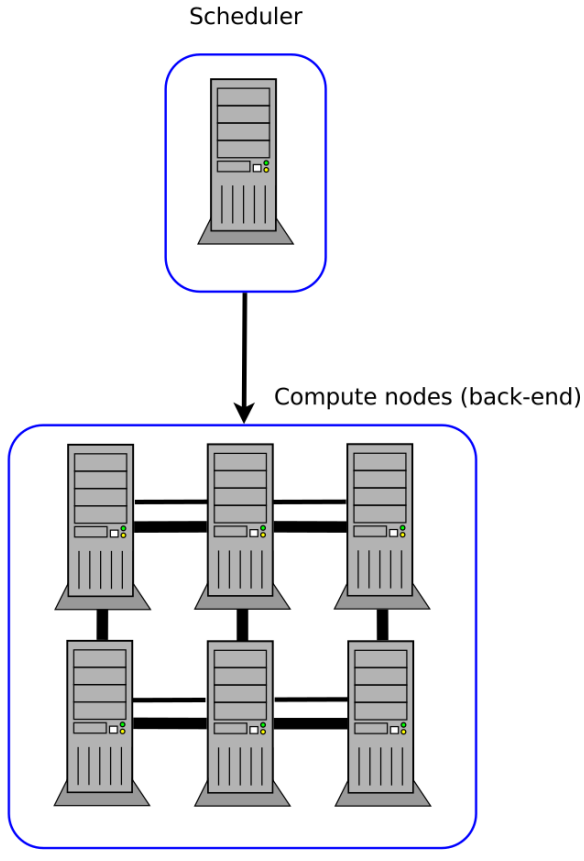


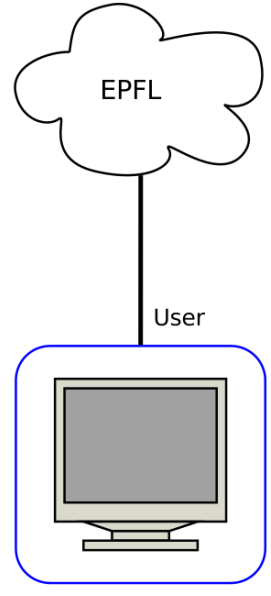
What is a cluster?



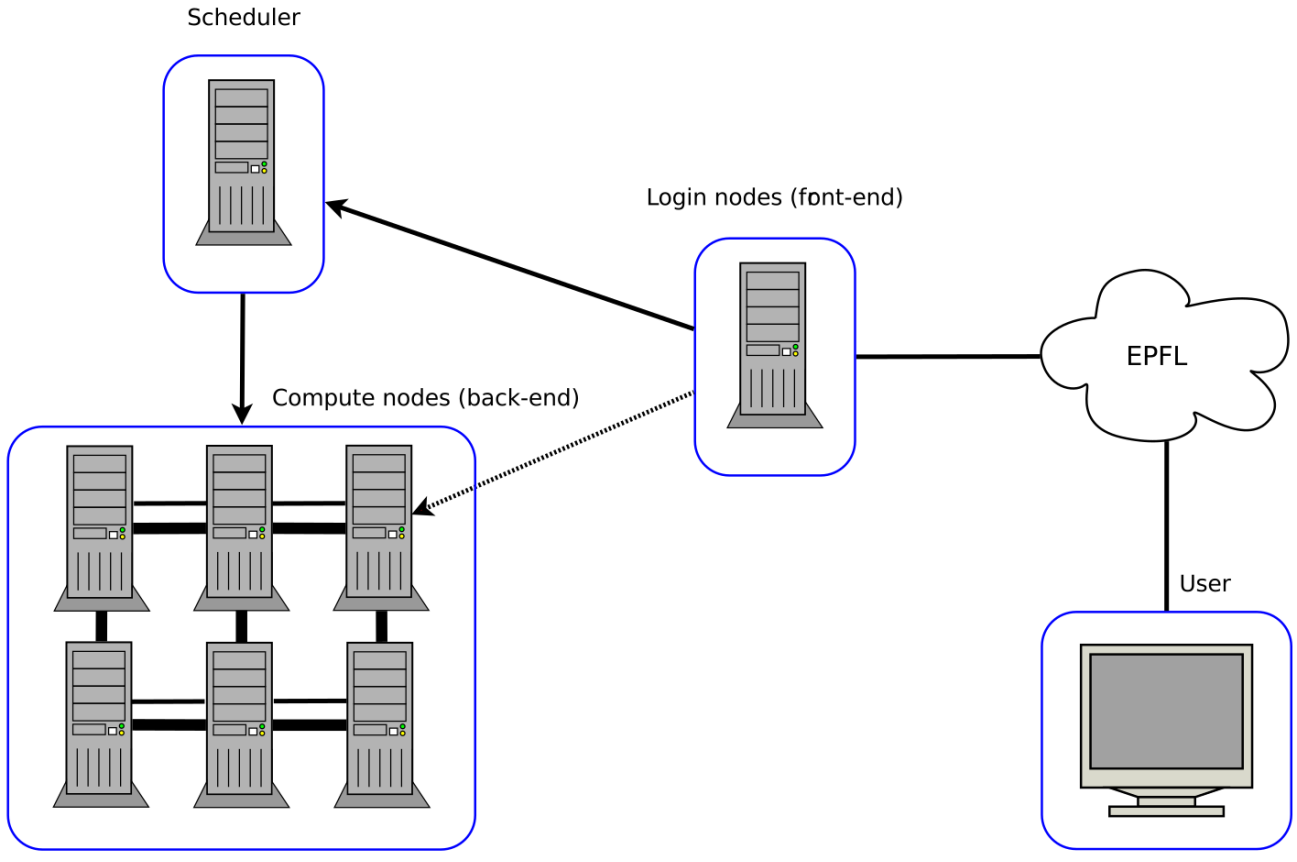


What is a cluster?

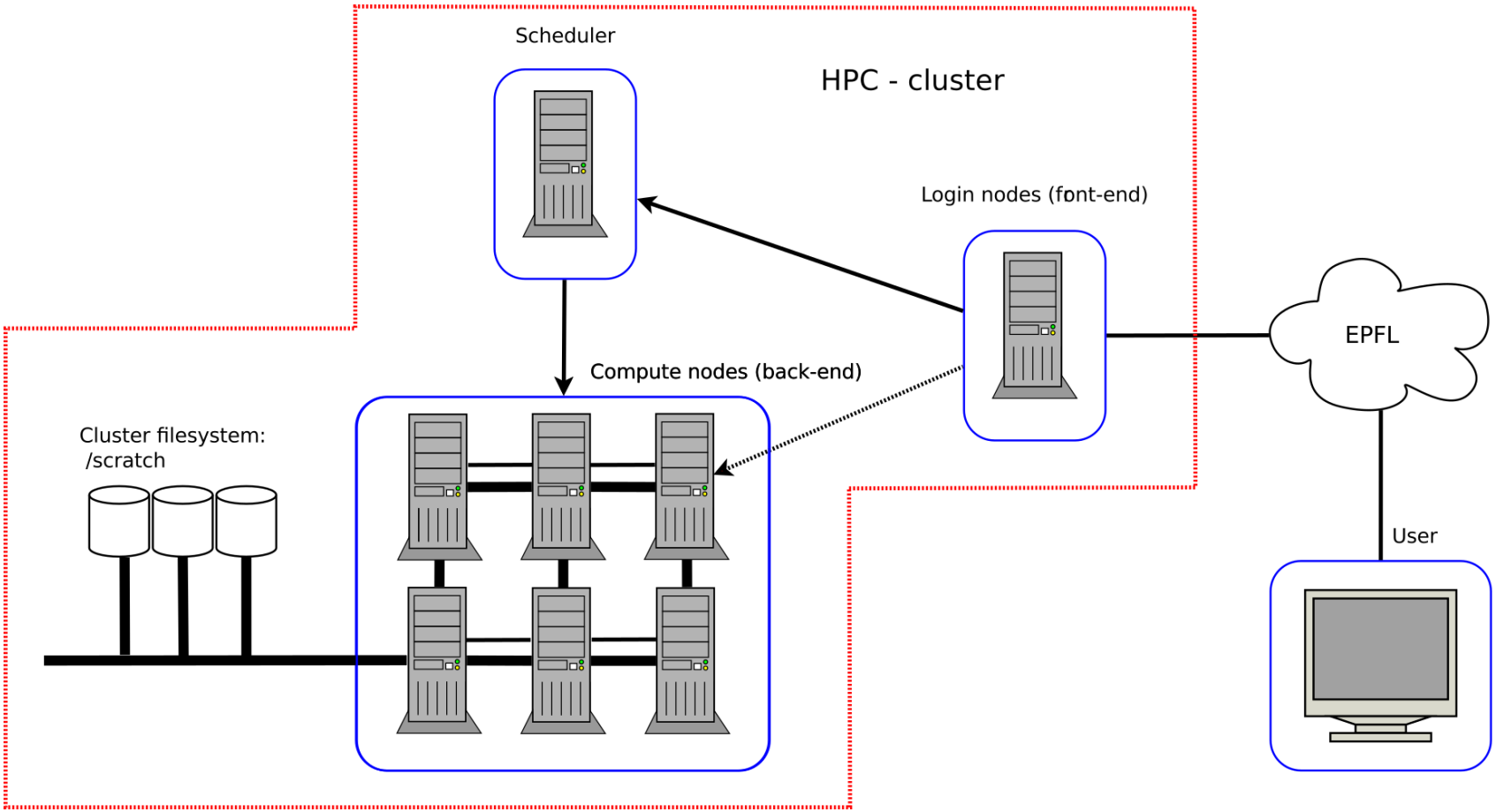




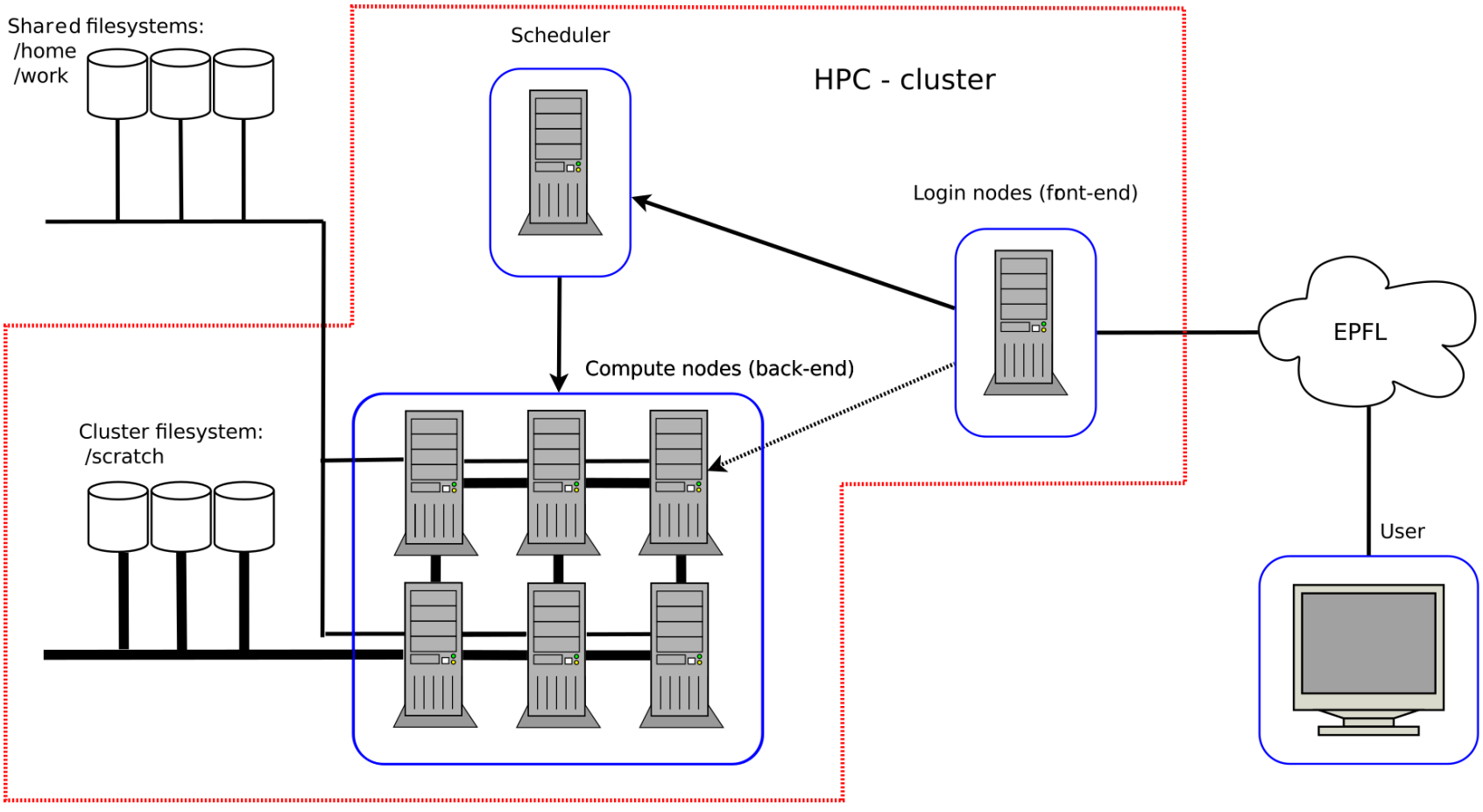
What is a cluster?



What is a cluster?



What is a cluster?



EPFL clusters

Login hostname	**Hosts** #	**Cores** # x GHz	**RAM** GB	**Network** Gbit/s
`fidis.epfl.ch`	336	28x2.6	126	56 (IB)
	72	28x2.6	256	
	216	28x2.6	192	100 (IB)
`helvetios.epfl.ch`	286	36x2.3	192	100 (IB)
`izar.epfl.ch`	64	40x2.1	192	2x100 (IB)
		+ 2x NVidia V100 7TFlops		
	2	40x2.1	192	100 (IB)
		+ 4x NVidia V100 7.8TFlops		

Remote connection

Connecting to remote machines

First step

- Connect to a remote cluster to get a shell
- SSH: Secure SHell

How to use

- `ssh -l <username> <hostname>`
- `ssh <username>@<hostname>`

Connecting to remote machines

First step

- Connect to a remote cluster to get a shell
- SSH: Secure SHell

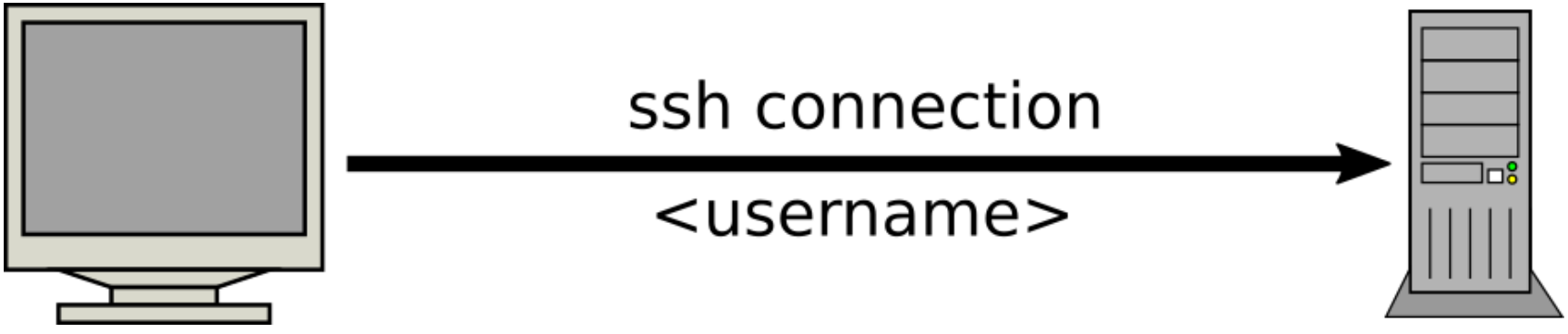
How to use

- `ssh -l <username> <hostname>`
- `ssh <username>@<hostname>`

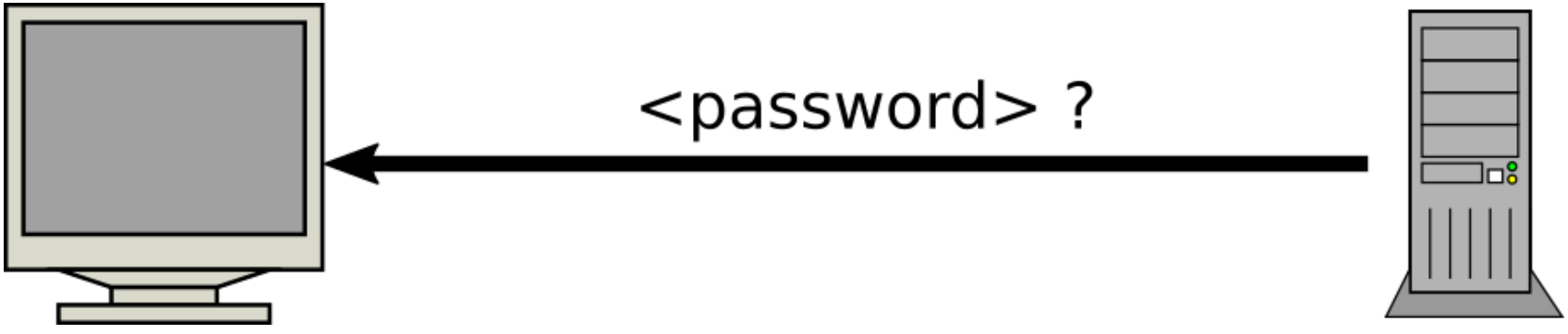
For windows users

Just install git, and use git bash

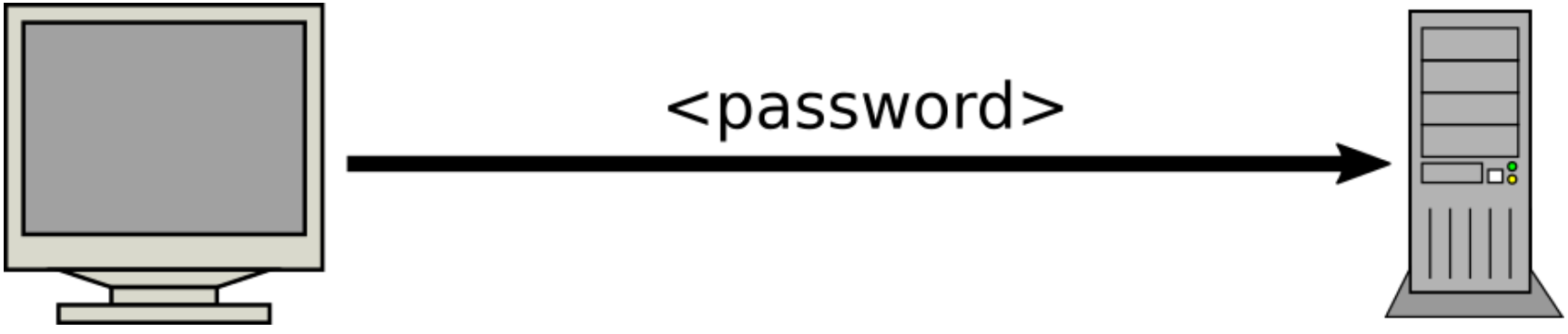
Getting a remote shell



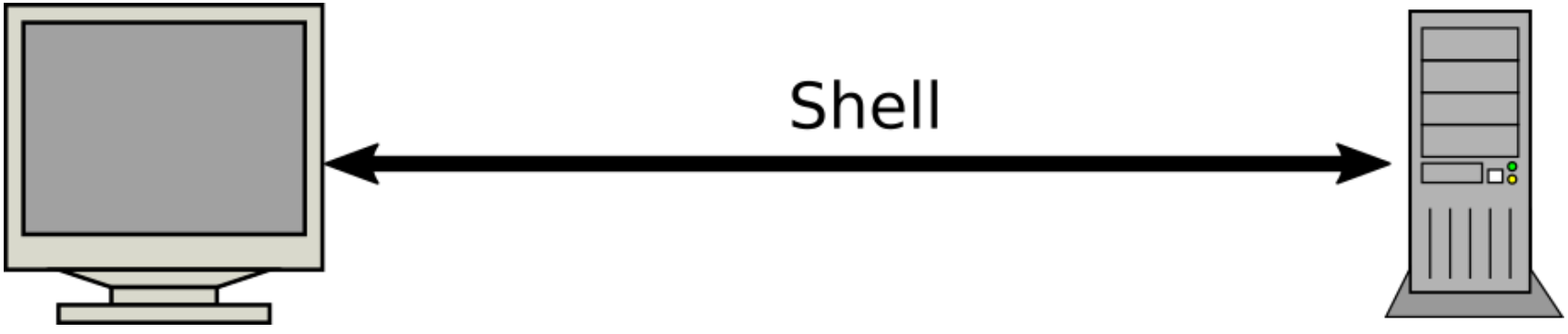
Getting a remote shell



Getting a remote shell



Getting a remote shell



Exercise 1: Simple connection

Front nodes

- **fidis** *[CPU (OpenMP/MPI)]*
- **izar** *[GPU (CUDA)]*

Questions:

- Connect to fidis' front node
- Check the different folders `/home` `/scratch`

Transferring files: using scp

`scp` works fine for Linux/MacOS, it is often pre-installed
On windows use GIT Bash or pscp.exe from Putty

Exercise 2: Using `scp`

`scp` usage

Send data to remote machine:

```
scp [-r] <local_path> <username>@<remote>:<remote_path>
```

Retrieve data from remote machine:

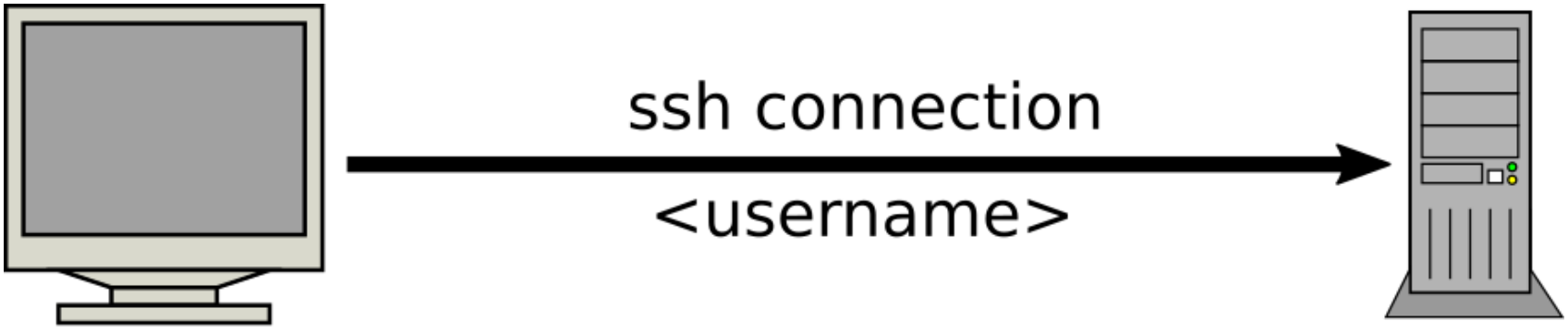
```
scp [-r] <username>@<remote>:<remote_path> <local_path>
```

Note: *It is always easier to "send to" and "receive from" the clusters since they have a fix ip/name*

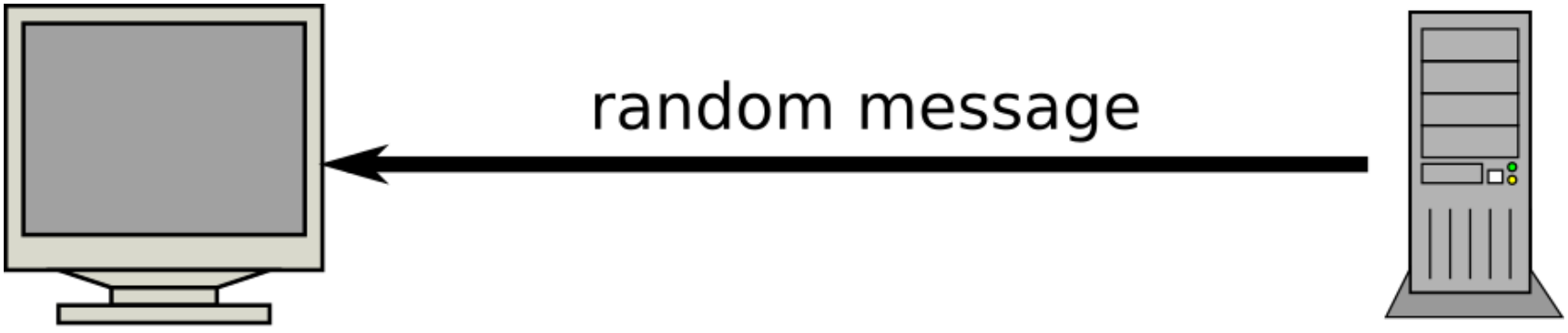
Questions:

- Copy a file from your machine to the cluster.
- Give the name of the file you just copied (leave it as long as you did not get feedbacks for the exercise) **[moodle]**
- Retrieve a file from the cluster onto your machine

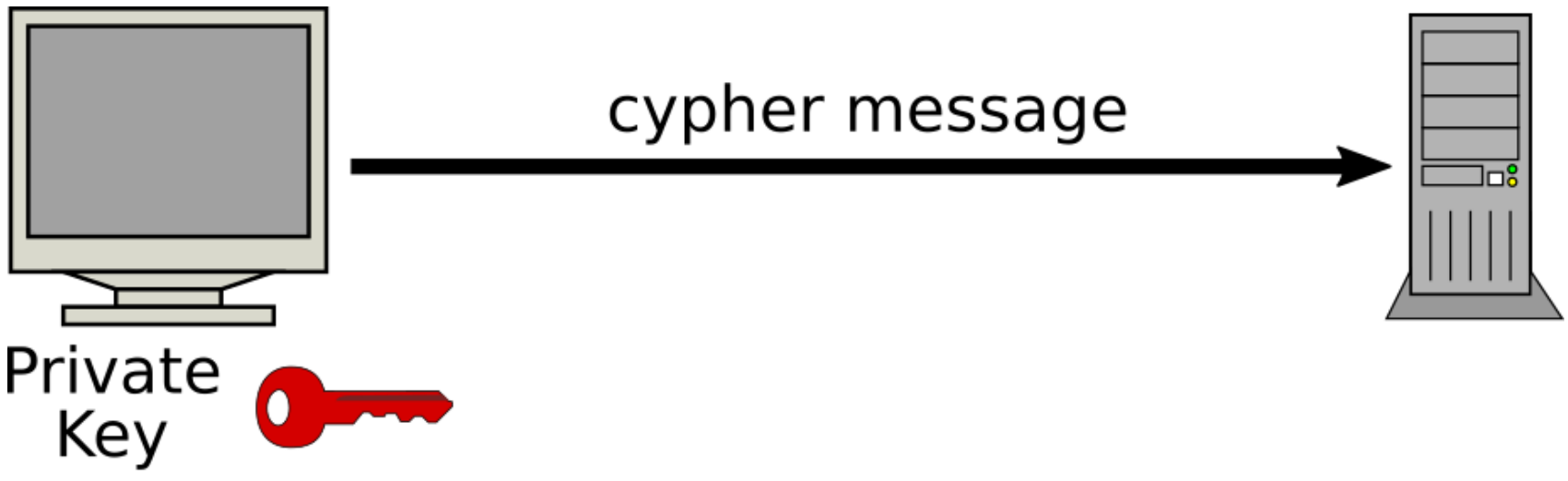
Getting a remote shell using keys



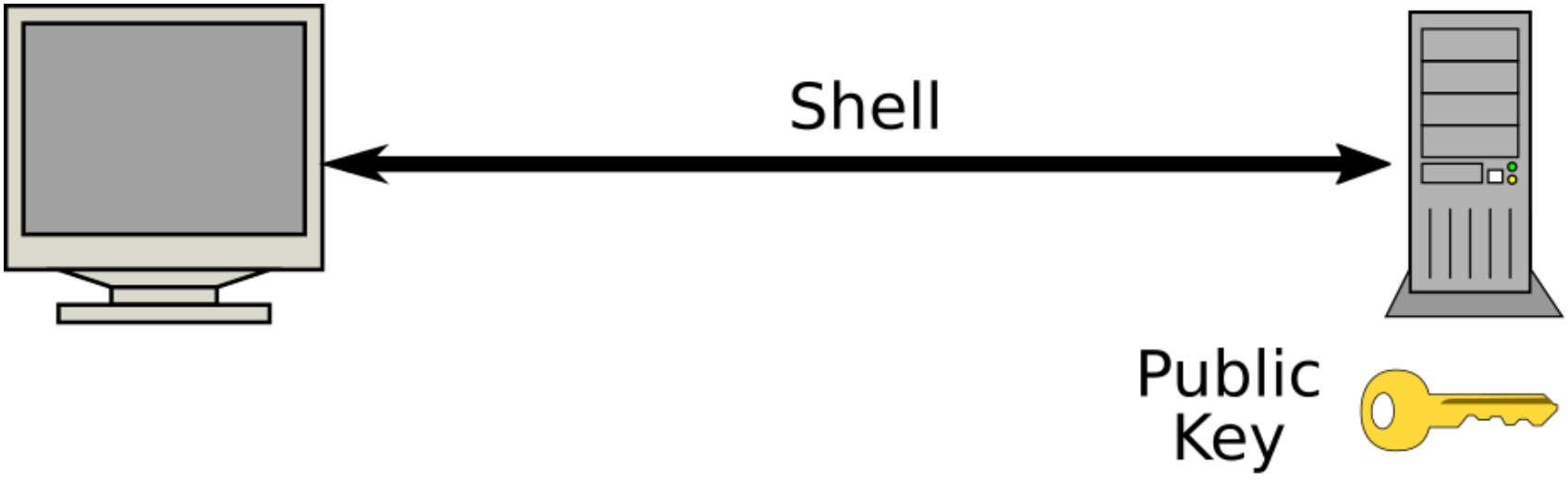
Getting a remote shell using keys



Getting a remote shell using keys



Getting a remote shell using keys



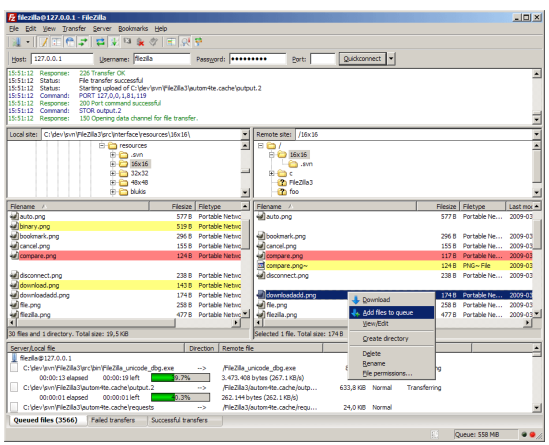
Exercise 3: Connection using keys [optional]

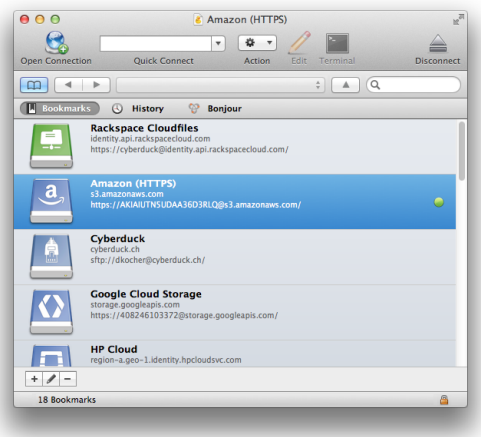
Questions:

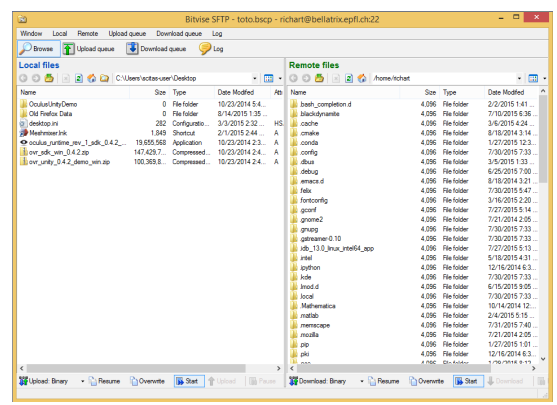
- Generate a pair of public/private key using `ssh-keygen`.
- Give us your **public** key **[moodle/optional]**
- Copy the public key `.ssh/id_rsa.pub` using `scp` in a file called `.ssh/authorized_keys` on the remote machine.
- Try to connect (it should not ask your password)

Transferring data using a GUI

`sftp` tones of GUIs, for example: Filezilla
https://filezilla-project.org (“all” OSes)
Cyberduck https://cyberduck.io/ (Windows and Mac)
Tunnelier https://bitvise.com/tunnelier (Windows)
TUIs also exists like `lftp`







Using modules

What are modules

- A way to dynamically modify the environment
- The contain configurations to use an application/a library

How to use them

- `module avail` list all possible modules
- `module load <module>` load a module
- `module unload <module>` unload a module
- `module purge` unload all the modules
- `module list` list all loaded modules

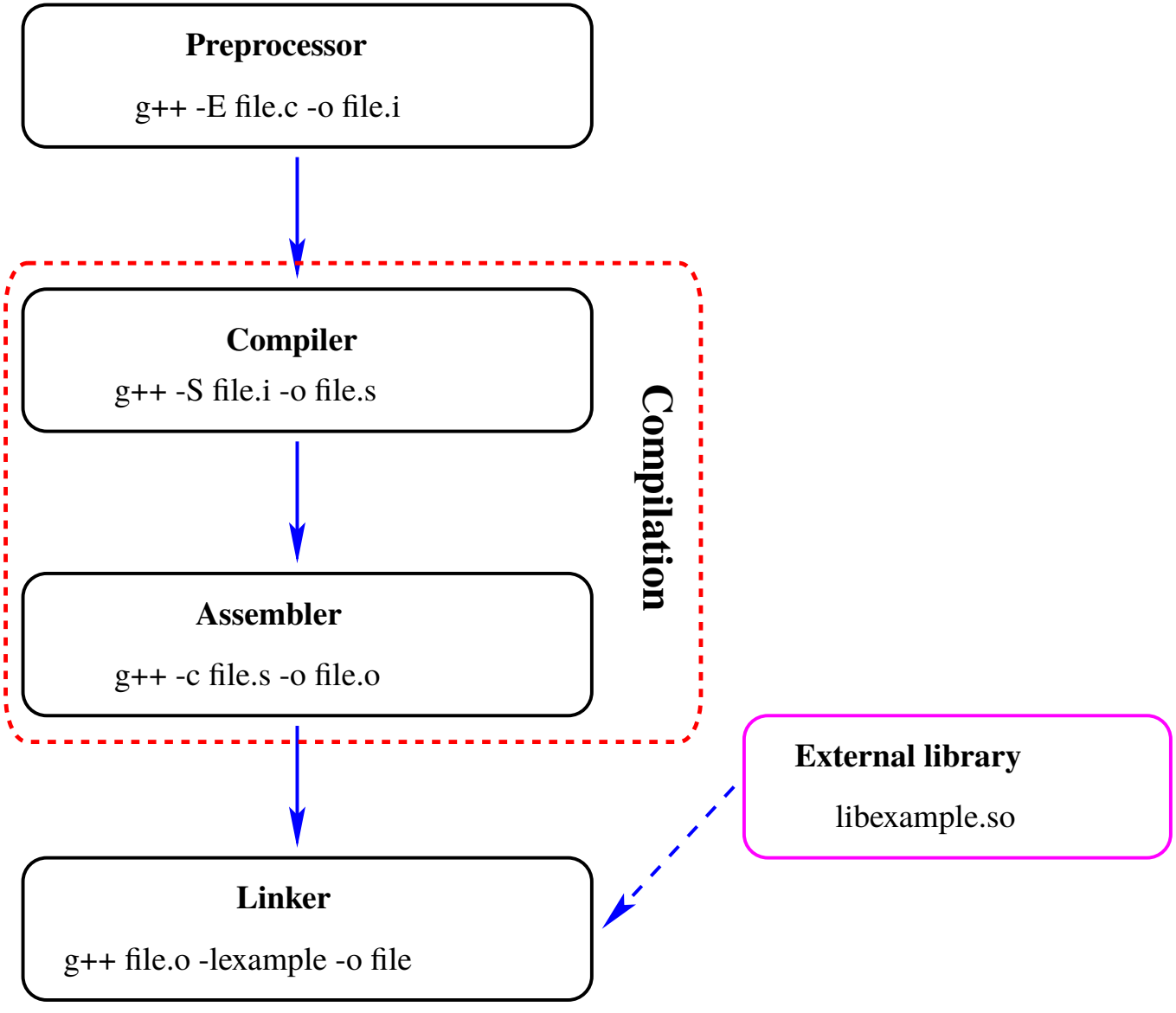
Exercise 4: Module command

Questions:

- List all available modules
- Try `g++ --version`
- Load the module `gcc`
- Try again `g++ --version`
- Try `icpc --version`
- Load the `intel` module
- Try again `icpc --version`
- List the currently loaded modules
- Which are the default version of gcc and intel in the modules ? **[moodle]**

Compilation

Compilation stages



Exercise 5: Compilation

Compilation all-in-one

If the compiler is given source code only it will do all the stages at once

Questions:

- Load a compiler module
- Compile the code `hello.c`
- What command did you had to type **[moodle]**
- Try to see the different stages of compilation [Optional]

Separated compilation

- Generate object files: compile

  `g++ -c file1.cc`

  `g++ -c file2.cc`

  Transforms code file `file?.cc` in object file `file?.o`
- Generate the executable: linking

  `g++ -o hello file1.o file2.o`

  Transforms object file `file?.o` in executable `hello`

Exercise 6: Separated compilation

Questions:

- Enter the folder `hello-separated-files`
- Generate `hello.o`
- Generate `greetings.o`
- Generate `hello` from `hello.o greetings.o`
- What commands did you had to type **[moodle]**

Makefiles

What is makefile

- It is a build automation system
- It is a set of rules on how to produce an executable:
  - what to compile ?
  - how to compile ?
  - what to link ?
  - how to link ?
- It as predefined rules using predefined variables

```
.cc.o:
	$(CXX) $(CPPFLAGS) $(CXXFLAGS) -c $<
```

- Add line for linking

```
$(EXE): $(OBJS)
	$(LD) -o $@ $(LDFLAGS) $(OBJS)
```

- Usage: `make`

Exercise 7: Makefiles

Questions:

- Enter the folder `hello-makefile`
- Read the `Makefile`
- Add the `-Wall -Werror` option to the compilation options
- Compile the code

Generate a library

- Two types of libraries **static** or **shared**
  static: archive of object file
  shared: library loaded dynamically at execution
- Generate a shared library
  Object files have to be compiled with `-fPIC` option

```
g++ -o libgreetings.so -shared greetings.o
```

- Link

```
g++ -L<path to the library> -lgreetings
    -o <executable> <object_1> ... <object_n>
```

Build system

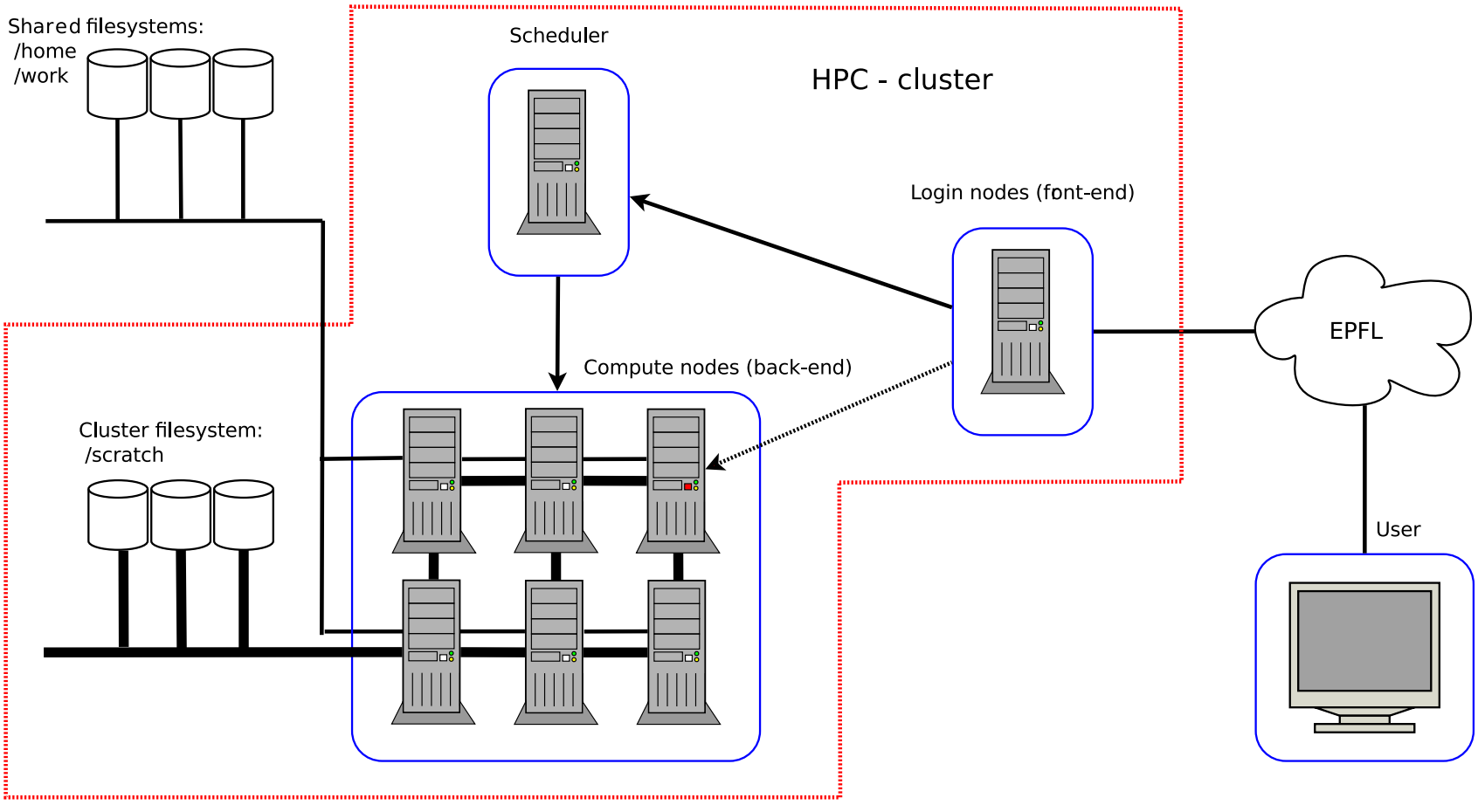
Real life project

For project a bit more consequent with multiple dependencies to external libraries, usage of a build automation system

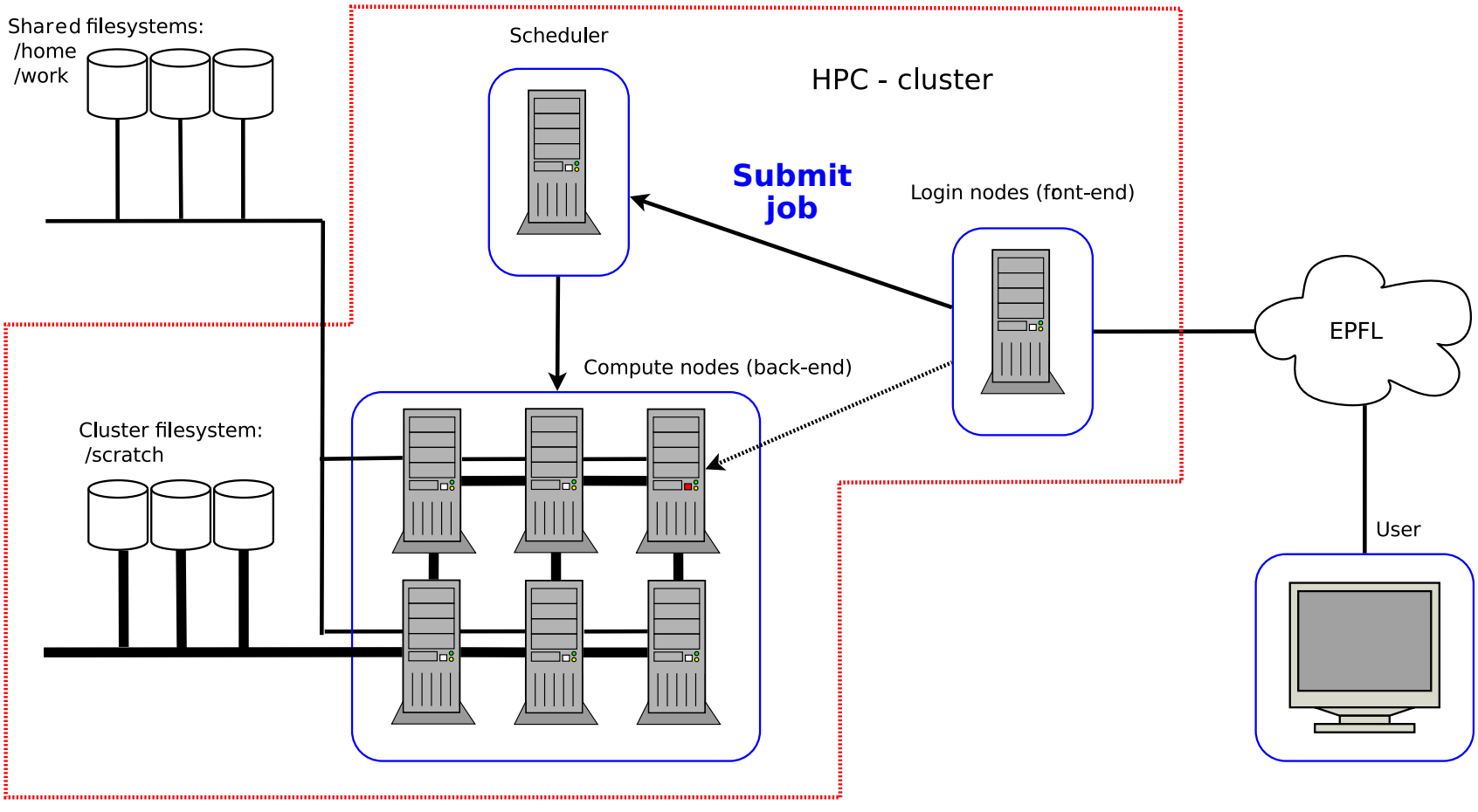
- CMake
- Autotools: configure
- SCons

SLURM

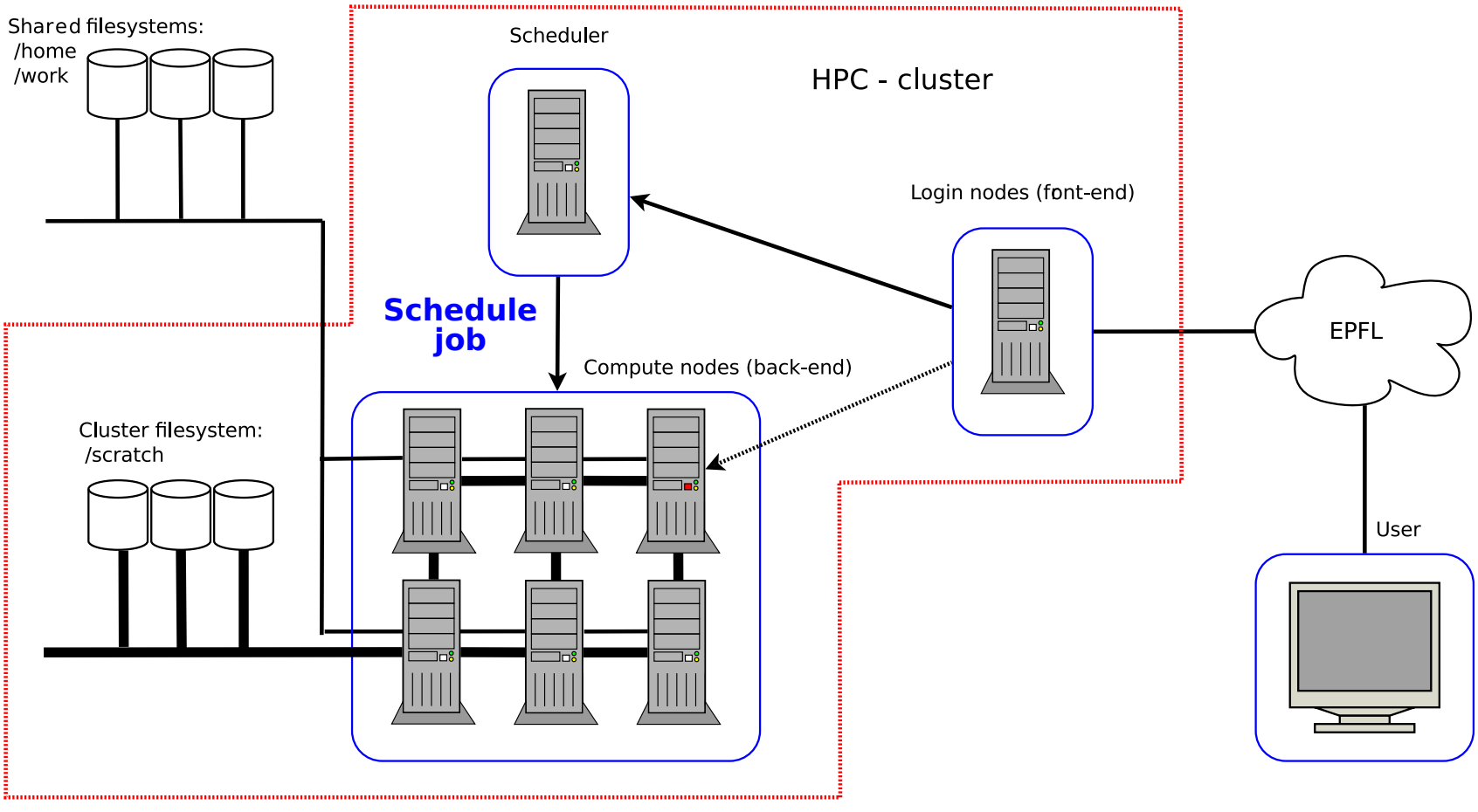
Scheduler


Shared filesystems:
/home
/work
Scheduler
HPC - cluster
Login nodes (font-end)
EPFL
Compute nodes (back-end)
Cluster filesystem:
/scratch
User

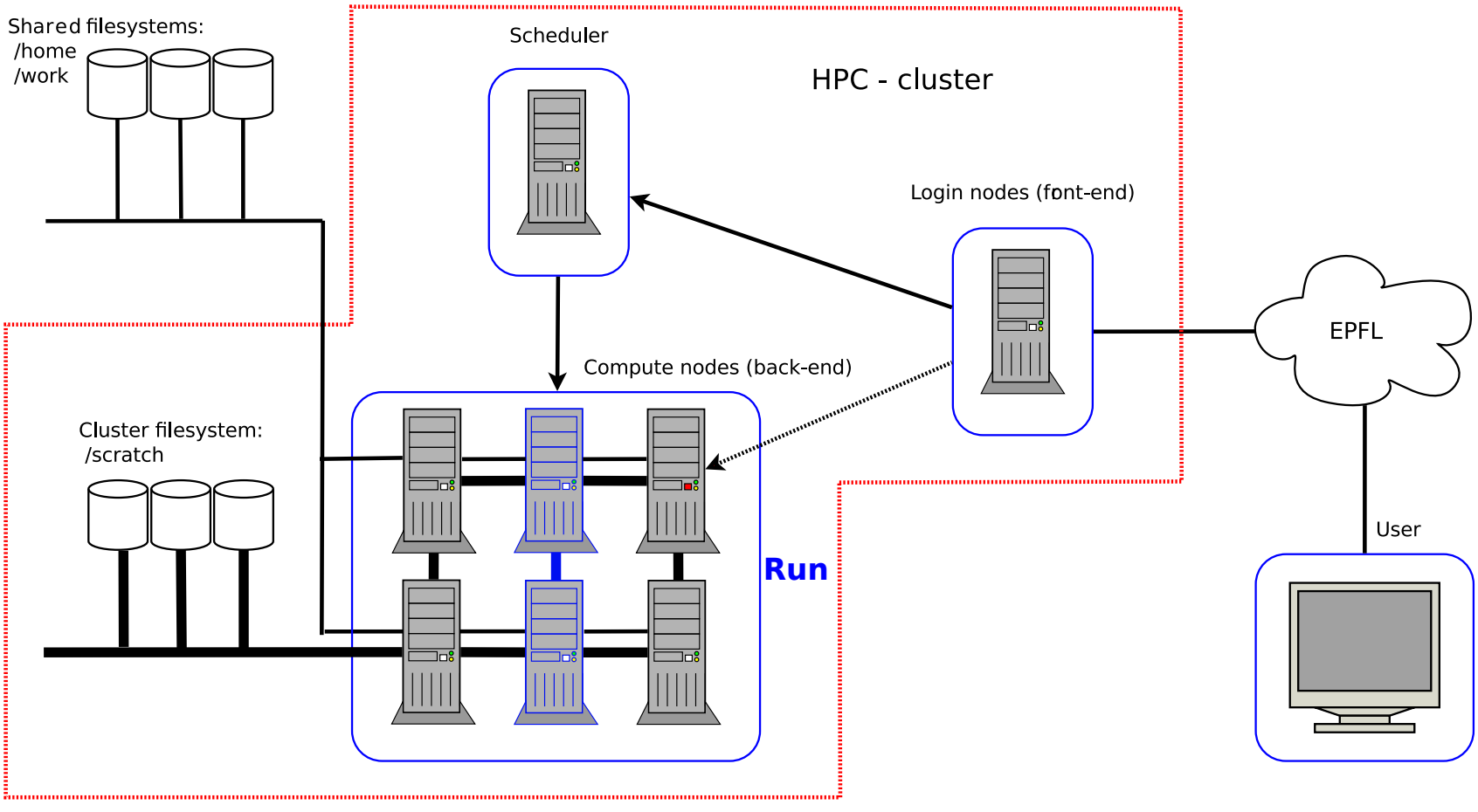
Scheduler


Shared filesystems:
/home
/work
Scheduler
HPC - cluster
Submit job
Login nodes (font-end)
EPFL
Compute nodes (back-end)
Cluster filesystem:
/scratch
User

Scheduler


Shared filesystems:
/home
/work
Scheduler
HPC - cluster
Login nodes (font-end)
Schedule
job
Compute nodes (back-end)
EPFL
Cluster filesystem:
/scratch
User

Scheduler


Shared filesystems:
/home
/work
Scheduler
HPC - cluster
Login nodes (font-end)
EPFL
Compute nodes (back-end)
Cluster filesystem:
/scratch
Run
User

SLURM

What is SLURM

- Simple Linux Utility for Resource Management
- Job scheduler
- Soda in Futurama

Basic commands

- `sbatch` submit a job to the queue
- `salloc` allocates resources
- `squeue` visualize the state of the queue

SLURM: common options

- `-A` `--account` defines which account to use
  For this class you are in `phpc2021`
- `-u` `--user` defines the user
  mainly useful for `squeue`
- `--reservation` defines which reservation to use
  For this class reservations named `phpc2021` are set on both `fidis` and `izar`
- `-p` `--partition` defines the partition to use
  List of partition can be get with `sinfo`

SLURM: `sbatch`/`salloc`

- `-N` `--nodes` defines the number of nodes to use
- `-t` `--time` defines the maximum wall time
- `-n` `--tasks` number of tasks, in the MPI sense
- `-c` `--cpus-per-task` number of cpus per process
- `--ntasks-per-node` number of tasks per node
- `--mem` defines the quantity of memory per node requested

sbatch job

- A job is simply a shell script
- sbatch option can be given as comment `#SBATCH` at the beginning of the script

Exercise 8: SLURM: first commands

Questions:

- Check the queue state
- Use `salloc` to allocate one node in the reservation `phpc2021`
- What command did you had to type? **[moodle]**
- Try `srun hostname`, we will see this command more in detail in later exercise sessions
- Exit the allocation to note block resources: `exit` or Ctrl-d

Exercise 9: SLURM: command sbatch

Questions:

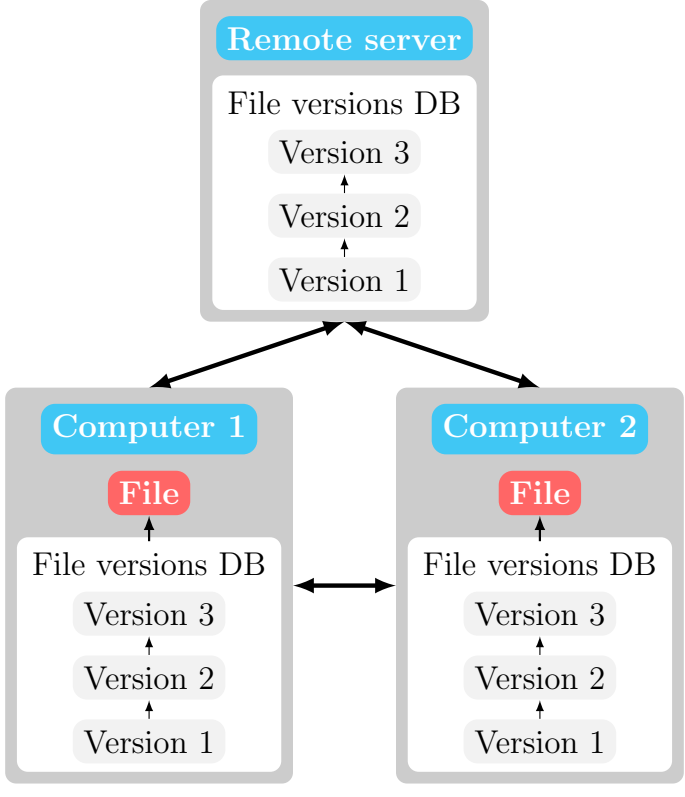
- Write a script that runs the hello world code
- Try your script.
  note: *in general you should not try your codes on the front node there is a `debug` partition for that*
- Submit your script with `sbatch` using the reservation
- Try `squeue -u <username>`
- A file named `slurm-<jobid>.out` should have been created, check its content
- Add the reservation as an option directly in the script
- Submit it again
- Provide the path to your script and do not delete it or change it until you get a feedback **[moodle]**

GIT

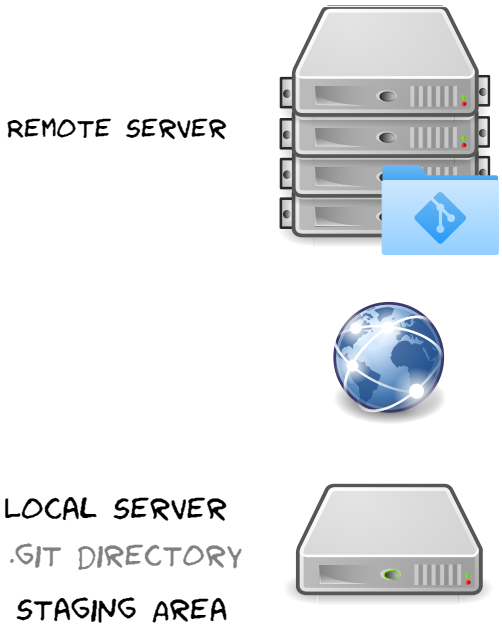
“Versioning”: with Git

Git: *the stupid content tracker*

- Distributed revision control
- Originally developed by Linus Torvald
- Named after the *egotistical bastard* Linus

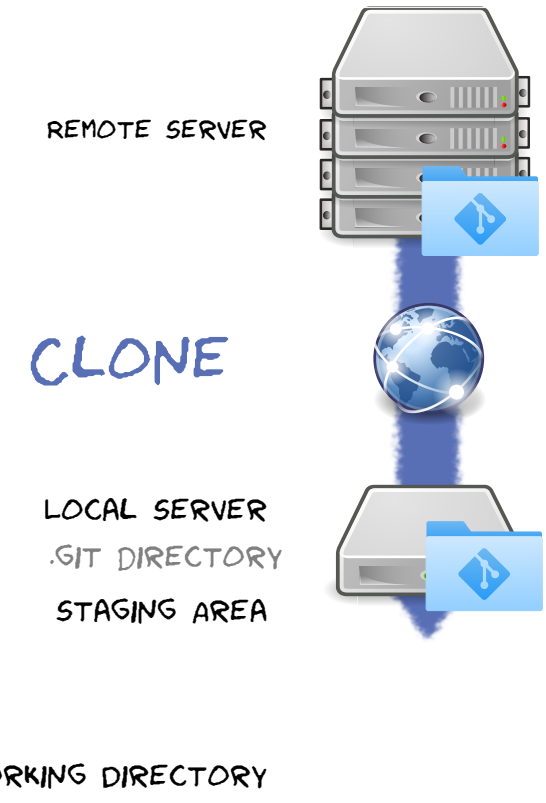


git clone



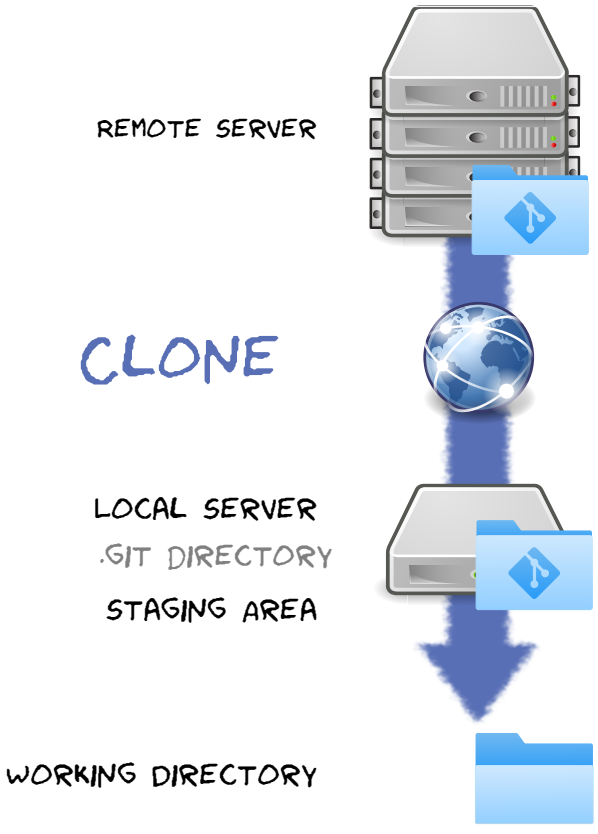

```
$ git clone <uri repo.git>
```

git clone




```
$ git clone <uri repo.git>
Cloning into '<repo>'...
remote: Counting objects: 6940, done.
remote: Total 6940 (delta 0), reused ...
Receiving objects: 100% (6940/6940), ...
Resolving deltas: 100% (3286/3286), done.
```

git clone




```
$ git clone <uri repo.git>
Cloning into '<repo>'...
remote: Counting objects: 6940, done.
remote: Total 6940 (delta 0), reused ...
Receiving objects: 100% (6940/6940), ...
Resolving deltas: 100% (3286/3286), done.
```

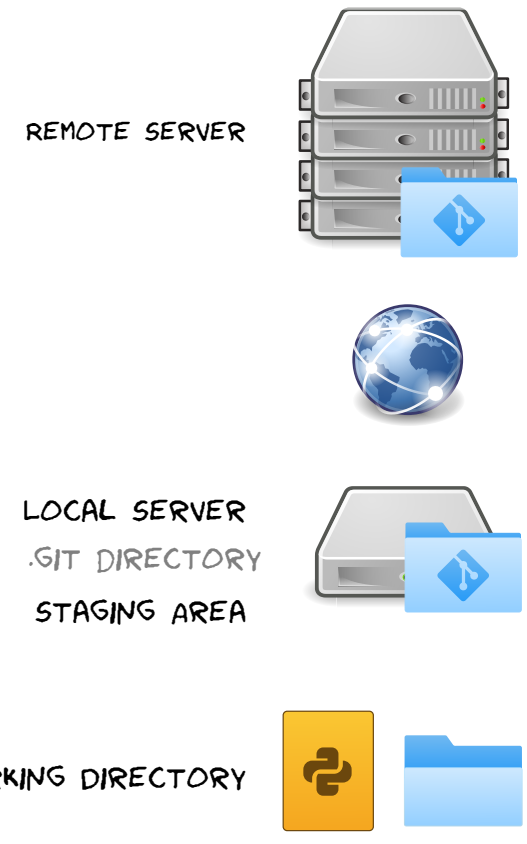
git status is your friend

```
$ git status
```

git status is your friend

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working tree clean
```

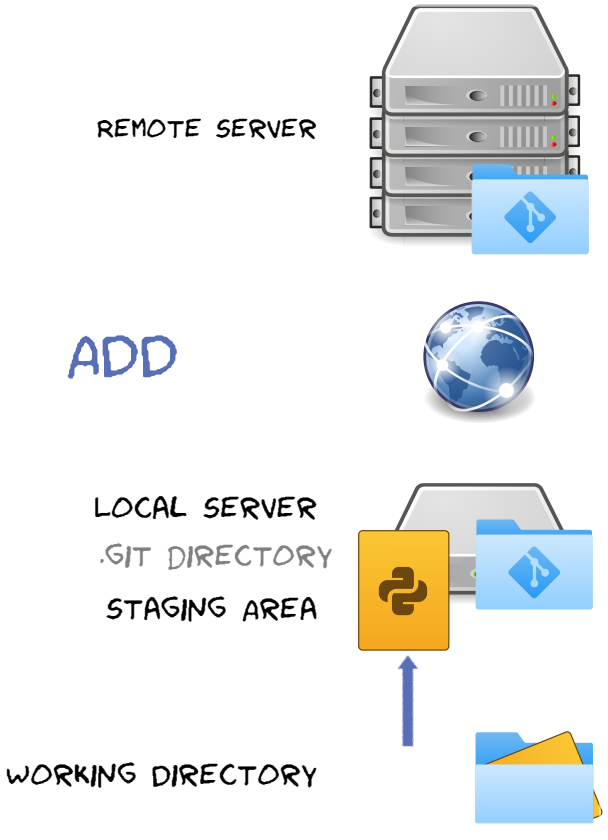
Lets add a file: staging/commit



Lets add a file: staging/commit

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        my_code.py

nothing added to commit but untracked files present
```

Lets add a file: staging/commit

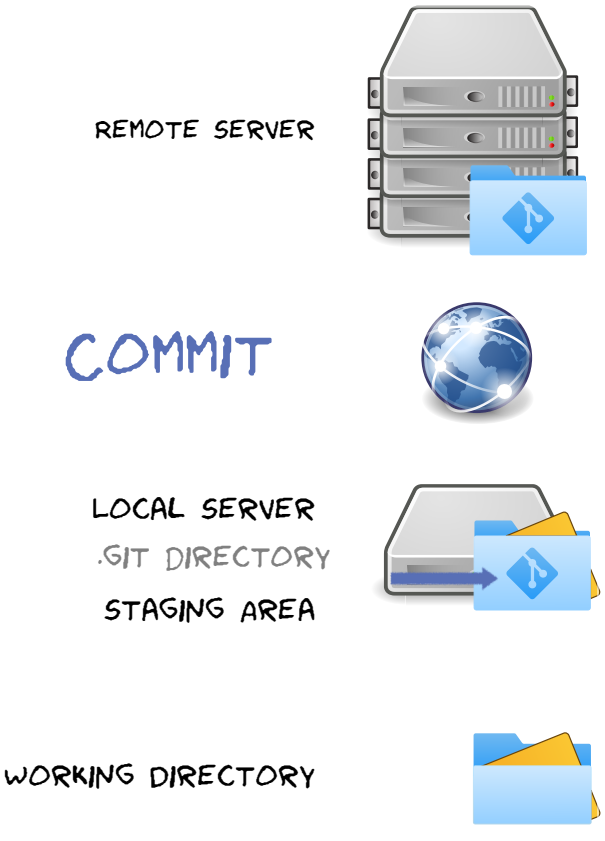



```
$ git add <filename>
```

Lets add a file: staging/commit

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   my_code.py
```

Lets add a file: staging/commit

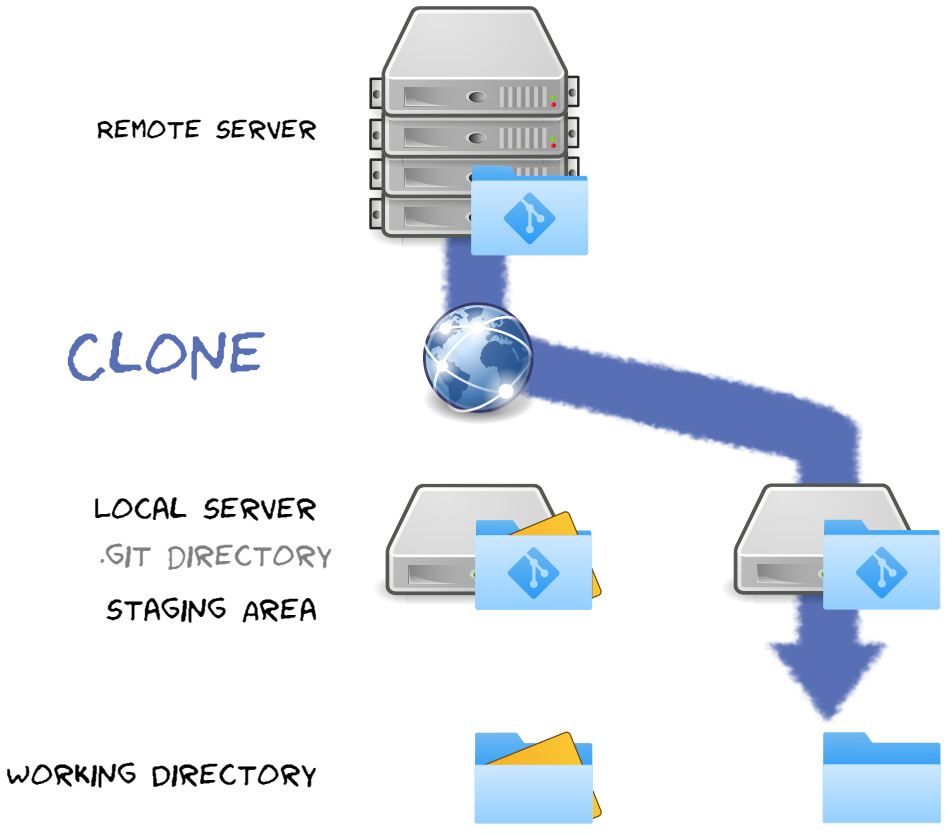



```
$ git commit -m <message>
```

Lets add a file: staging/commit

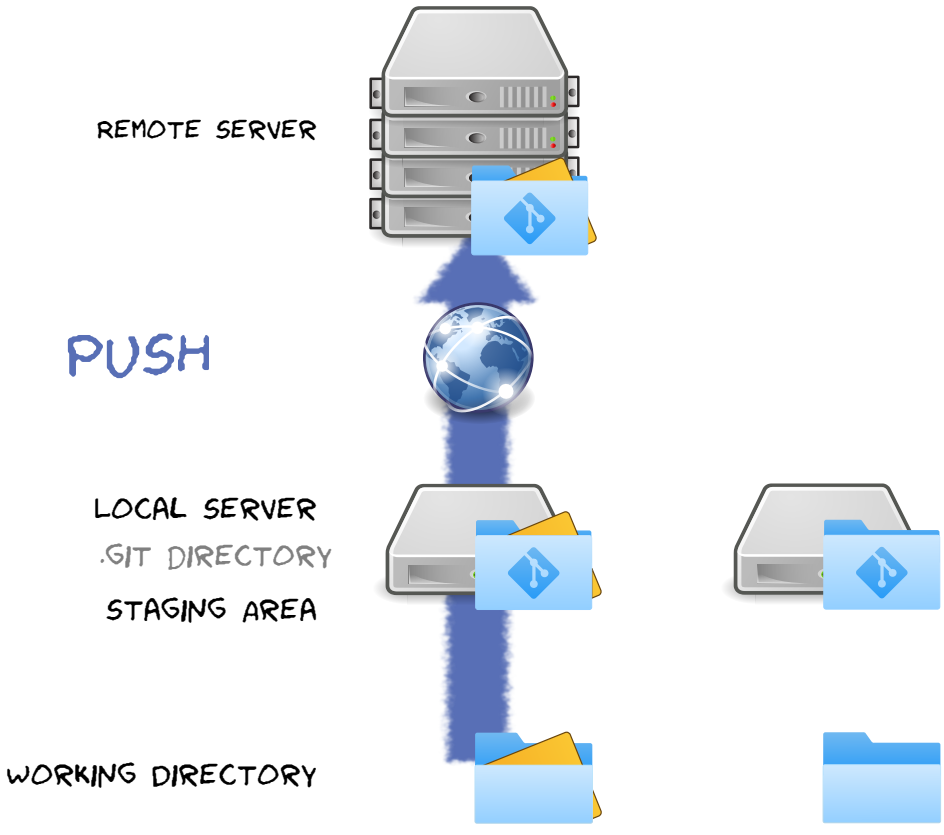
```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Synchronizing with the remote server



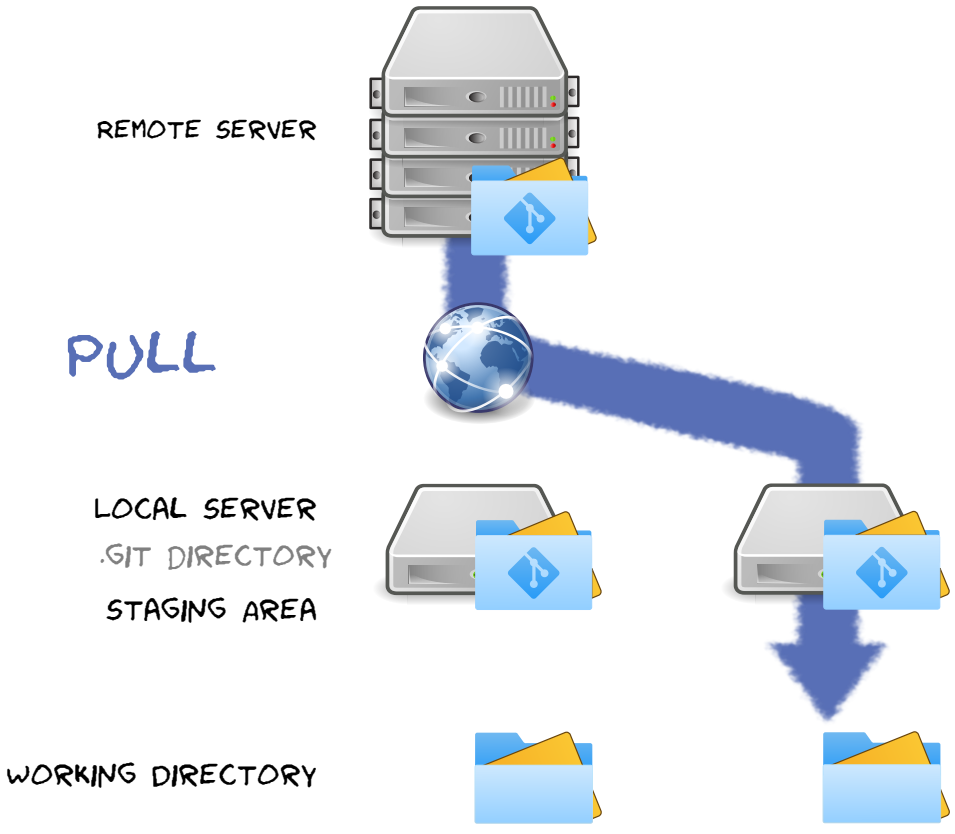

```
$ git clone <uri>
```

Synchronizing with the remote server




```
$ git push
```

Synchronizing with the remote server




```
$ git pull
```

Exercise 10: First step with Git

Questions:

- If you do not have git installed, get it from `https://git-scm.com/downloads` or from your package manager
- Go on `https://c4science.ch/` and login with your EPFL account (Login for Swiss Universities).
- Once connected go on the setting page (the wrench on the top right corner)
- In the **Authentication** > **SSH Public Keys** menu set you public ssh key. This key will be used to connect to the git server through **ssh**.

Exercise 11: First step with Git

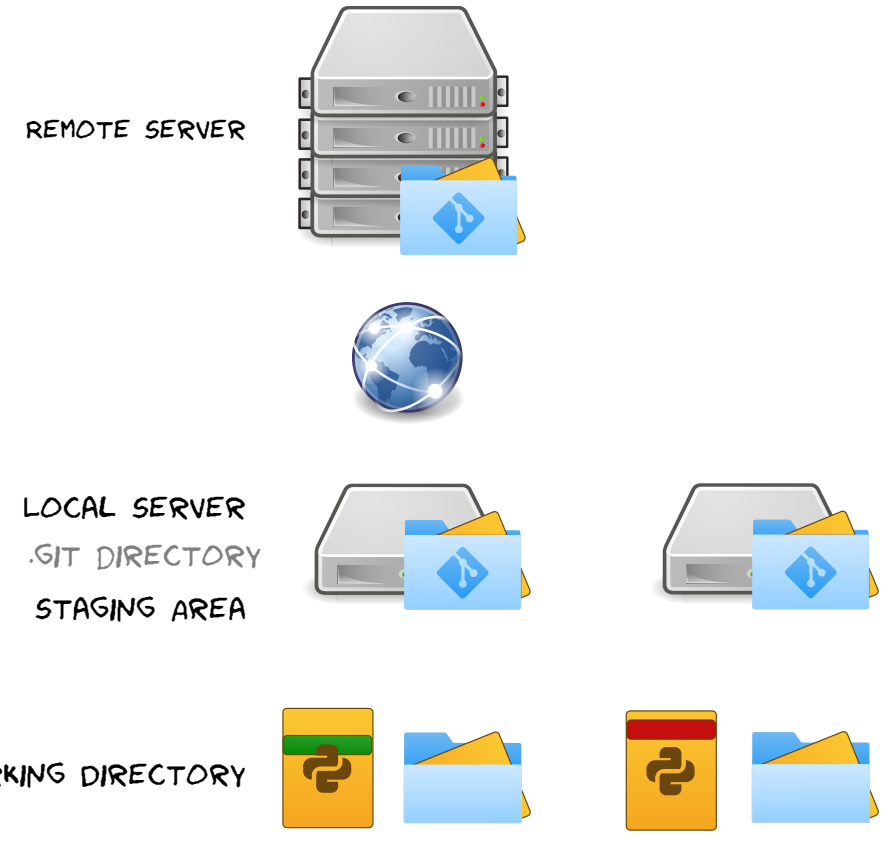
Questions:

- Now you should be able to clone a repository
  Either create a repository or clone

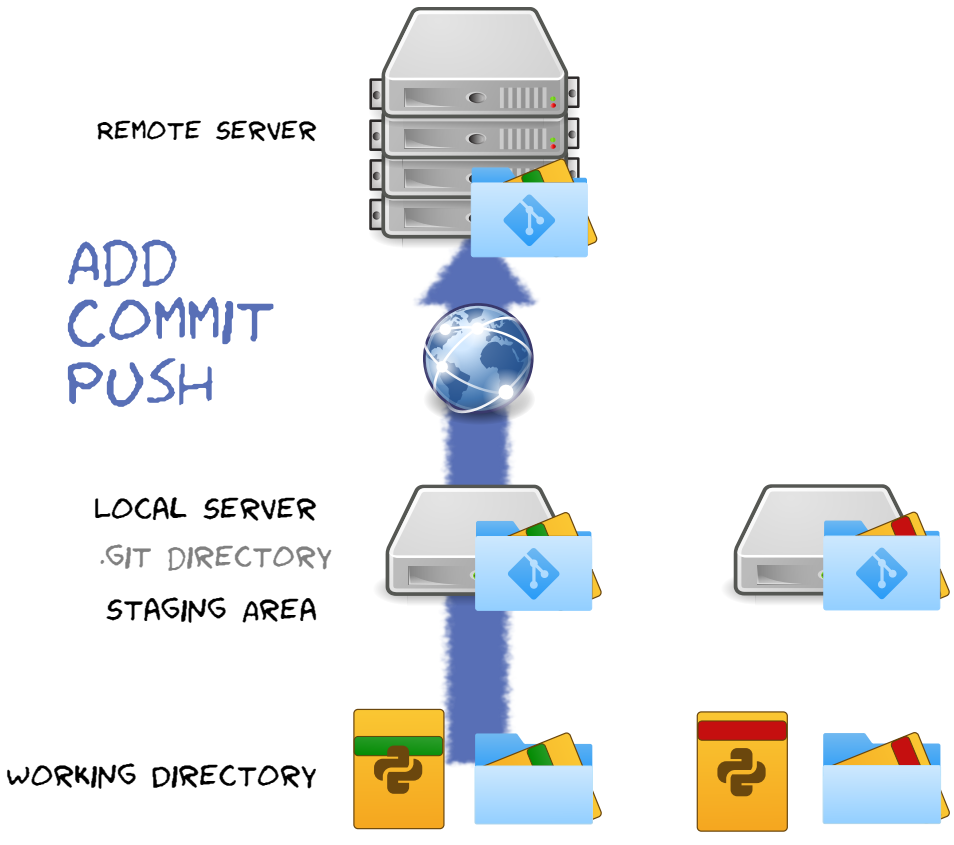
```
ssh://git@c4science.ch/source/scitas-test-repo.git
```

- Create a file, use a filename that will not clash with the others
- Check the state of your working copy
- Add the file to the repository
- Commit your modifications
- Clone the same repository in a different folder
- Pull the potential modifications from the server
- Push your changes to the server
- Give use your c4science username and the name of the file you added **[moodle]**

Collaborative work with potential problems

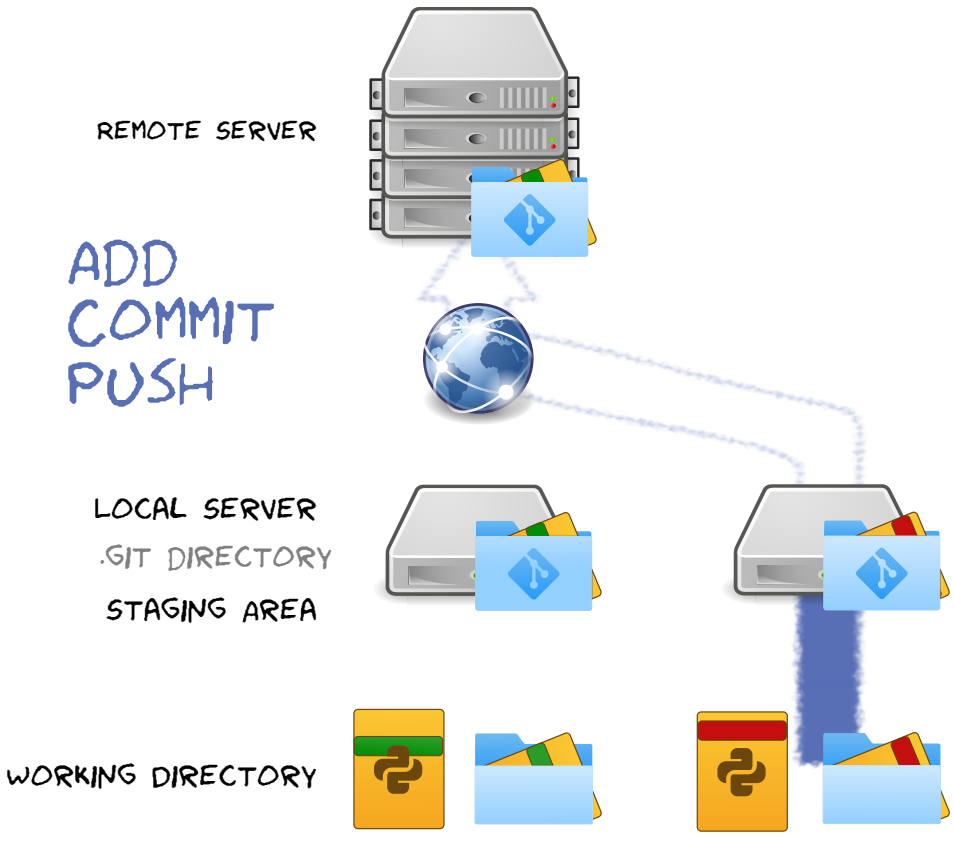


Collaborative work with potential problems




```
$ git add <filename>
$ git commit -m <message>
$ git push
```

Collaborative work with potential problems

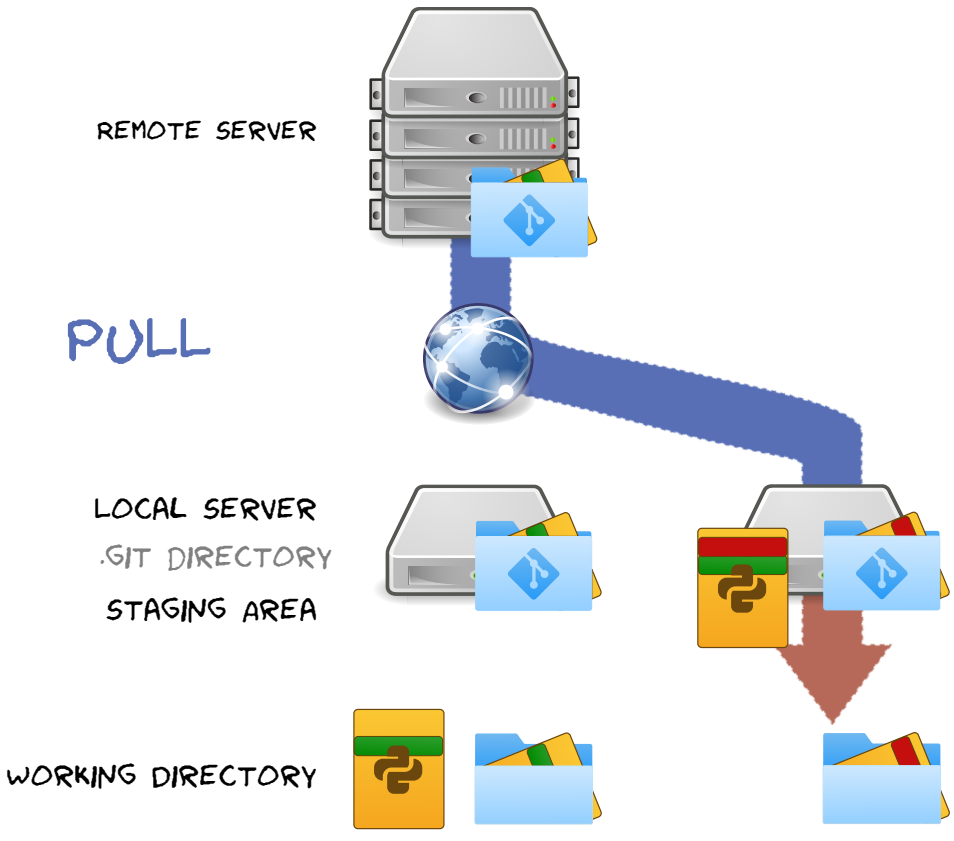



```
$ git add <filename>
$ git commit -m <message>
$ git push
```

Collaborative work with potential problems

```
$ git push
To <repo>
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to '<repo>'
hint: ...
```

Collaborative work with potential problems




```
$ git pull
```

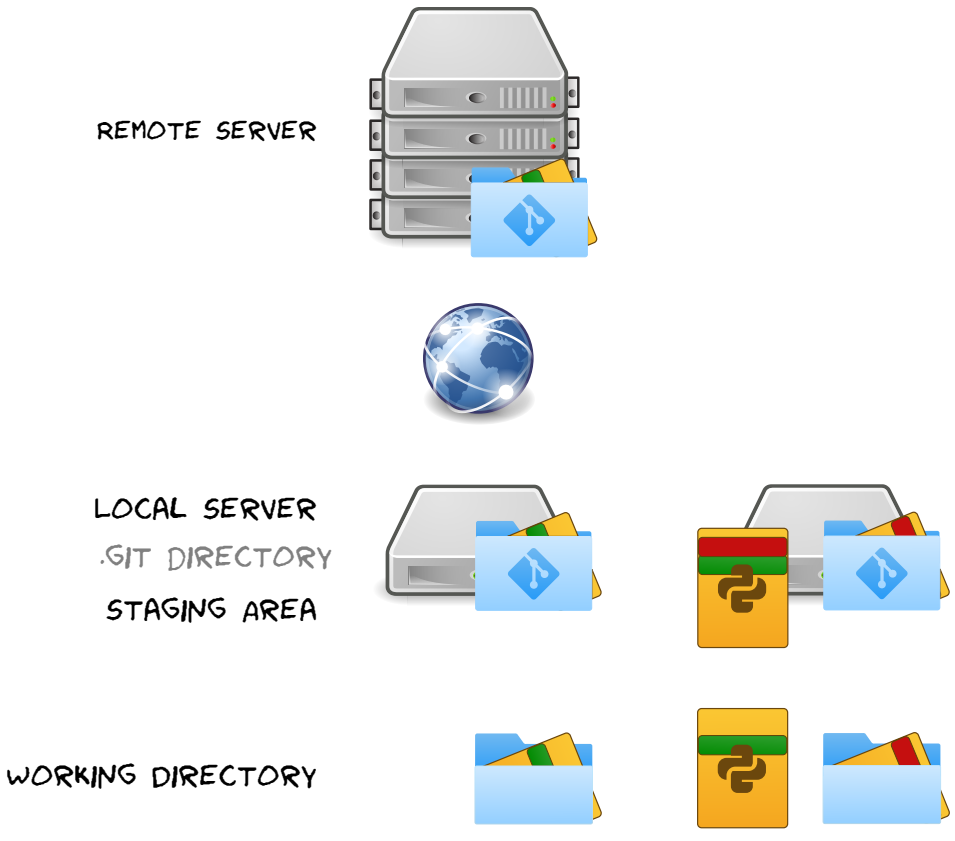
Collaborative work with potential problems

```
$ git pull
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From <repo>
    fe22d81..0bcfb99  master     -> origin/master
Auto-merging my_code.py
CONFLICT (content): Merge conflict in my_code.py
Automatic merge failed; fix conflicts and then commit the result.
```

Collaborative work with potential problems

```
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   my_code.py
```

Collaborative work with potential problems

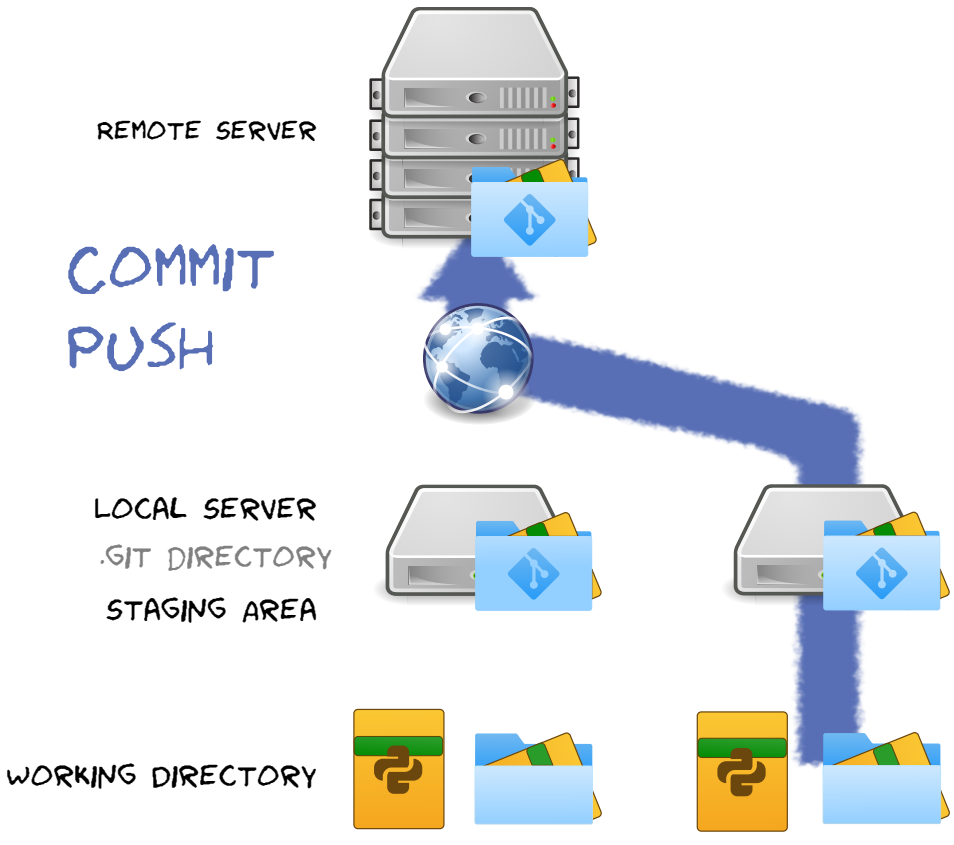


Correct the conflict:

my_file.py

```
<<<<<<<<<<
One version
==========
Other version
>>>>>>>>>>
```

Collaborative work with potential problems



my_file.py

```
One version
```

```
$ git commit -a
$ git push
```

Exercise 12: Generate and solve conflicts

Questions:

- Modify the file created in the previous exercise in both clones
- Commit this both modifications
- Pull and push in one of the clone
- Pull in the second clone, You should get a conflict

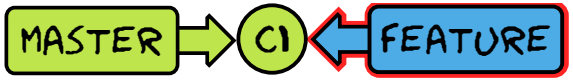
```
<<<<<<<<<<
One version
==========
Other version
>>>>>>>>>>
```

- Check the local status
- Correct the conflict and commit using `git commit -a`
- Push the modifications

Introduction to branches

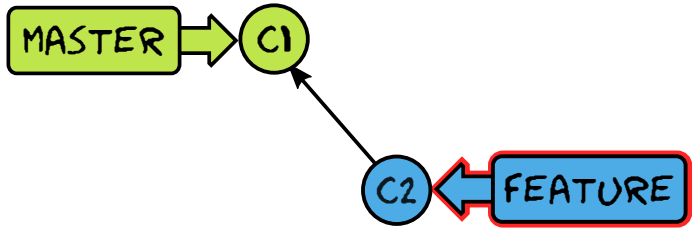

```
$ git clone <uri repo.git>
```

Introduction to branches



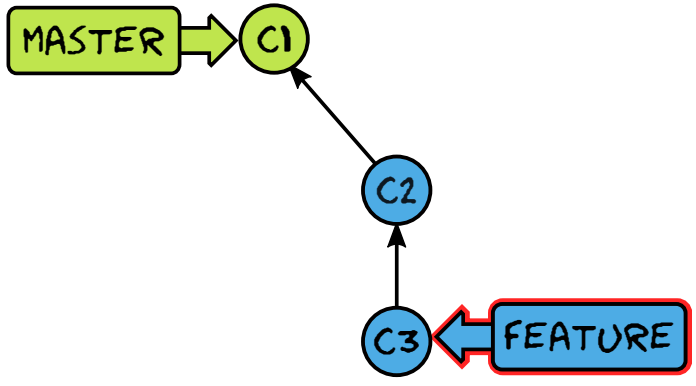

```
$ git checkout -b feature
```

Introduction to branches



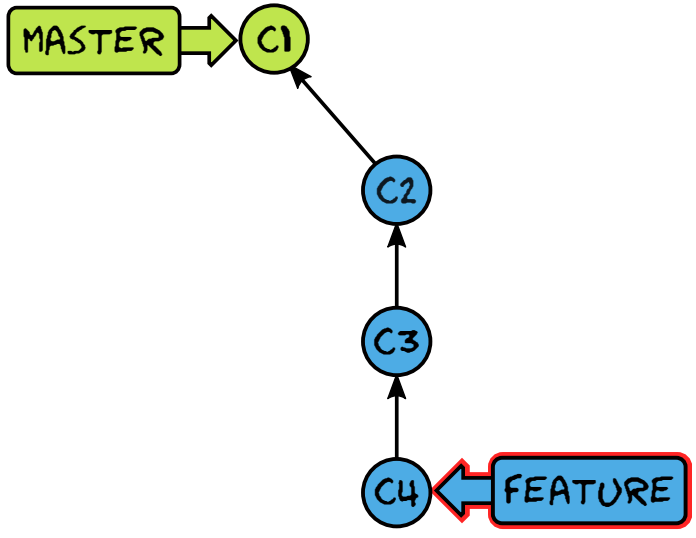

```
$ git commit -m <message>
```

Introduction to branches



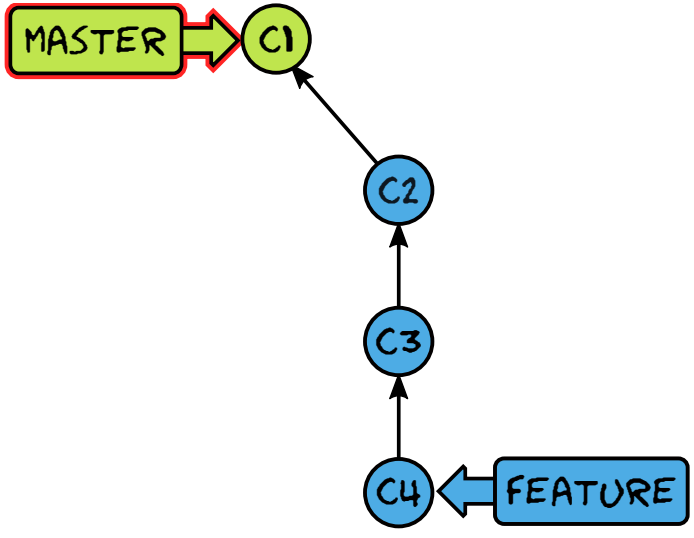

```
$ git commit -m <message>
```

Introduction to branches



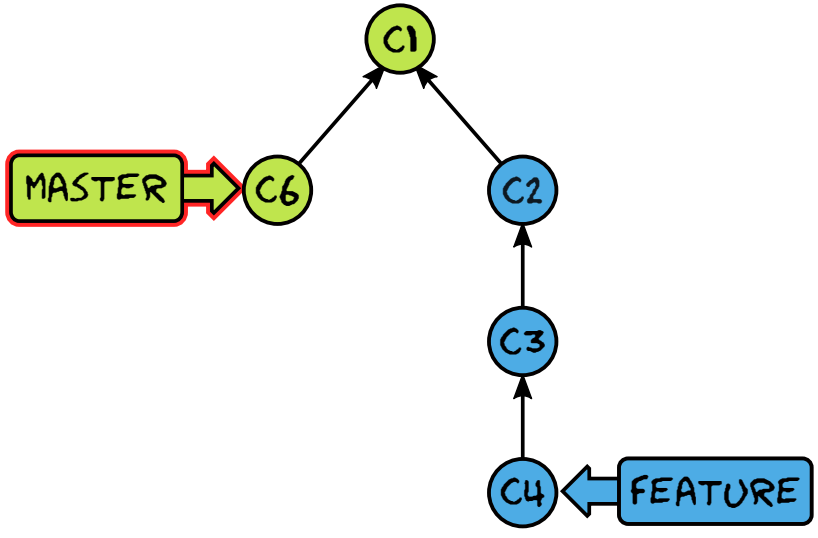

```
$ git commit -m <message>
```

Introduction to branches



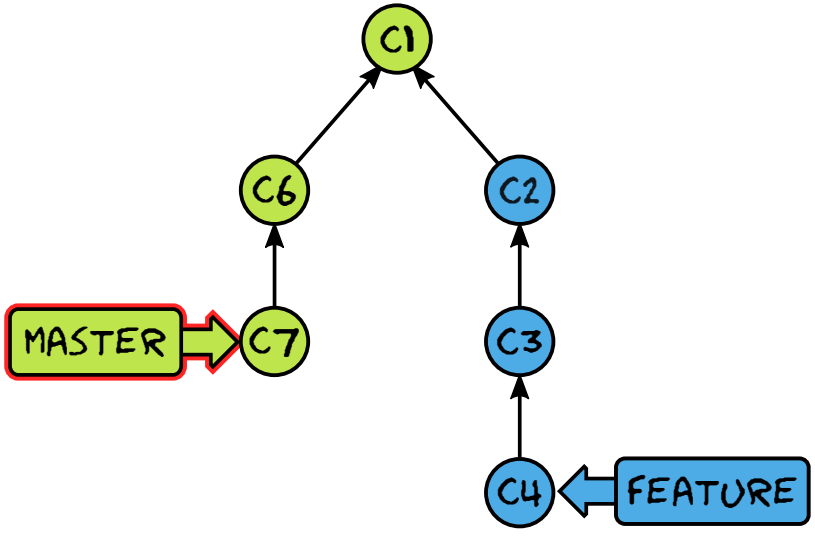

```
$ git checkout master
```

Introduction to branches



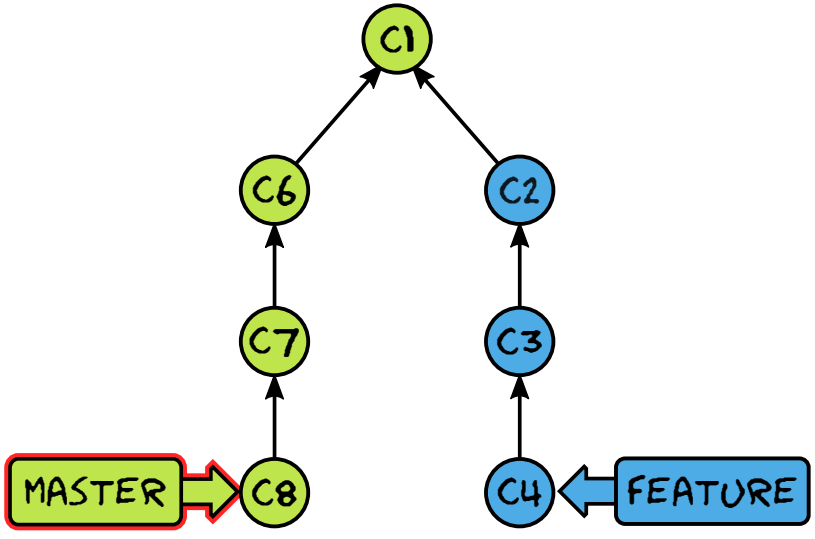

```
$ git commit -m <message>
```

Introduction to branches



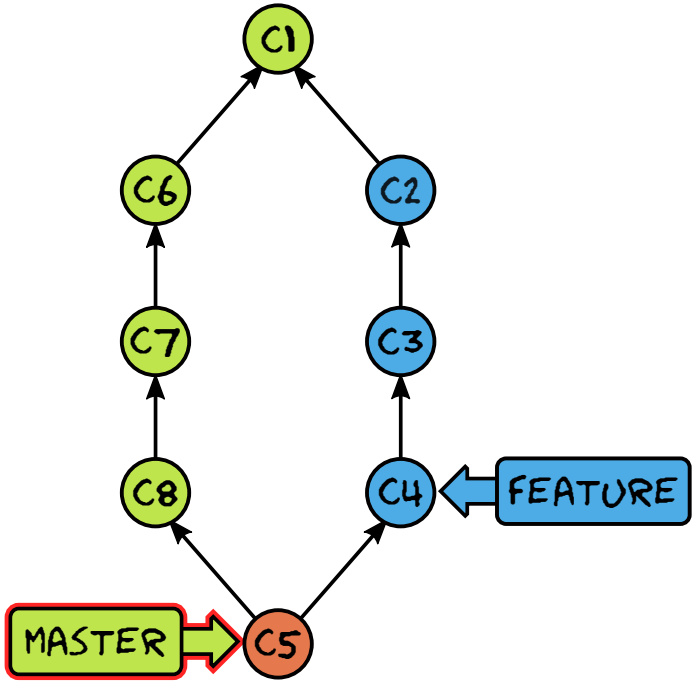

```
$ git commit -m <message>
```

Introduction to branches



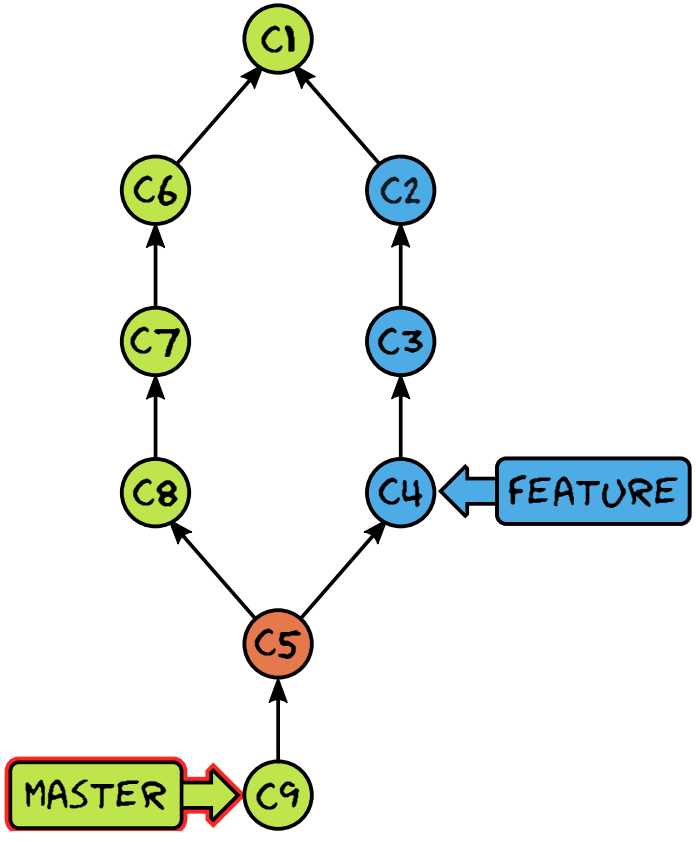

```
$ git commit -m <message>
```

Introduction to branches




```
$ git merge feature
```

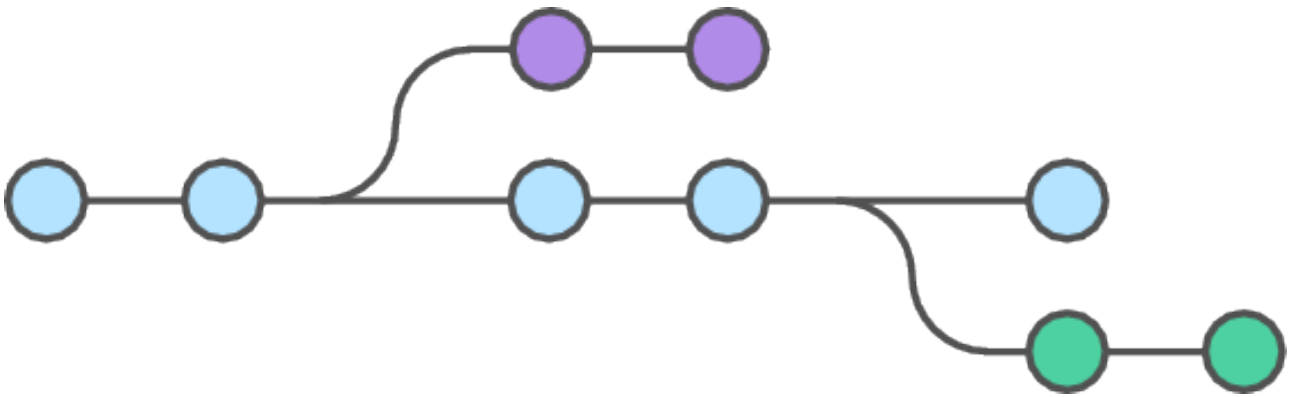
Introduction to branches



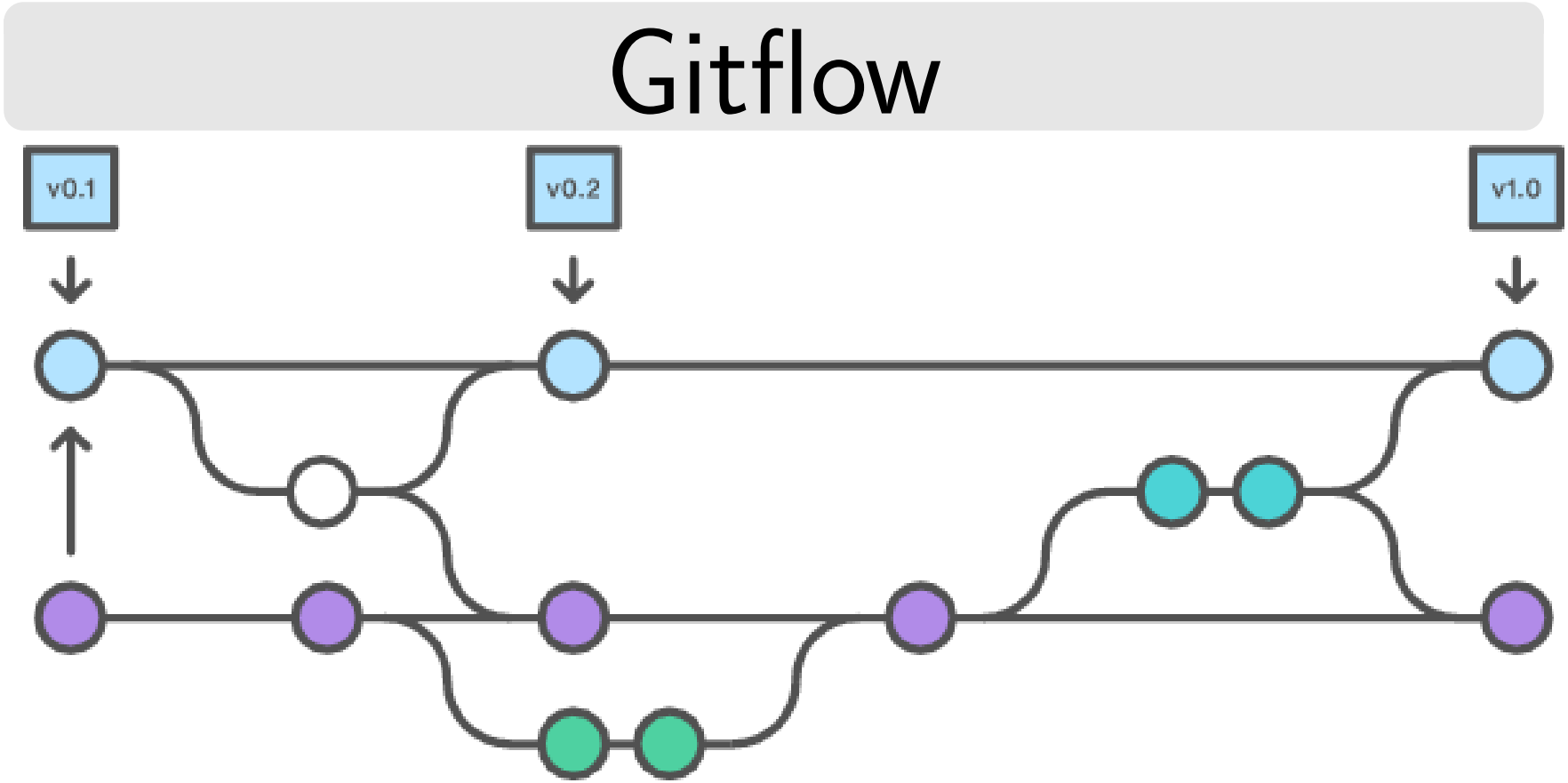

```
$ git commit -m <message>
```

Workflow: feature branch

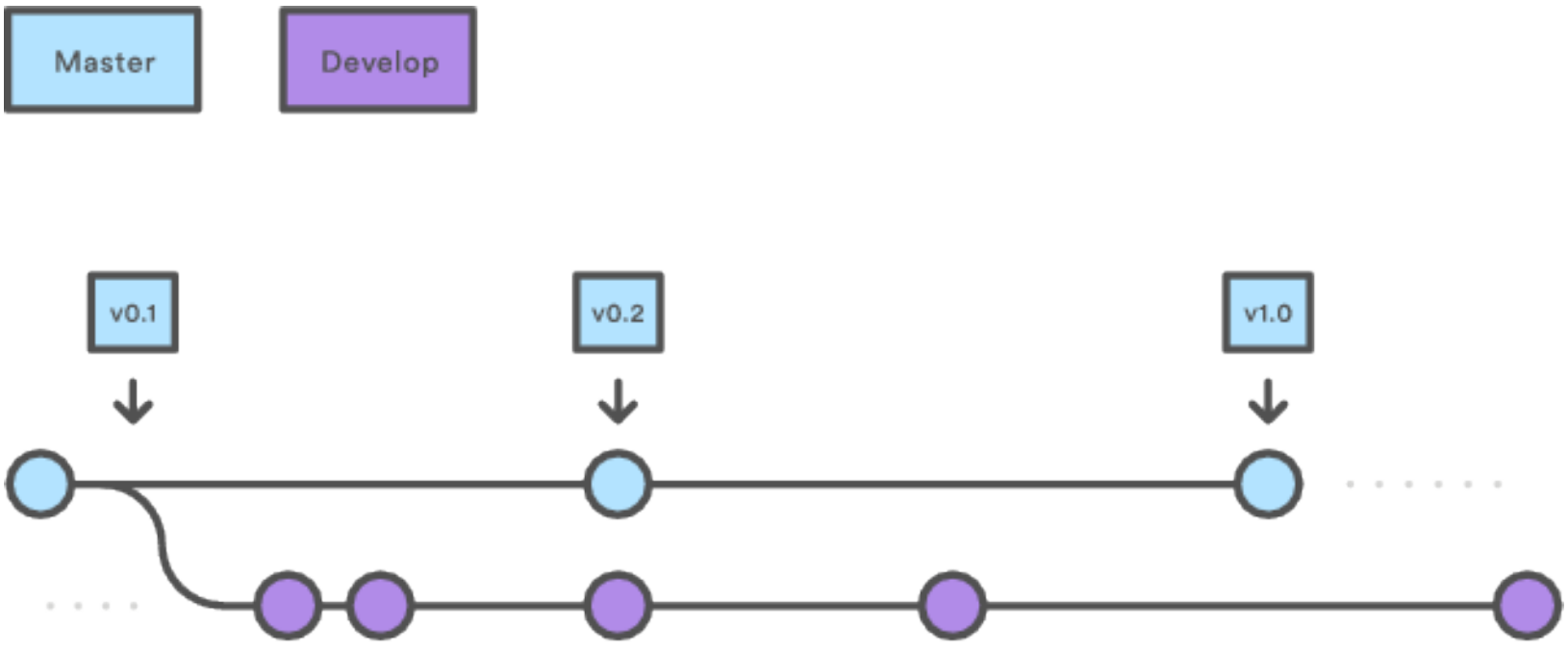
Feature branch



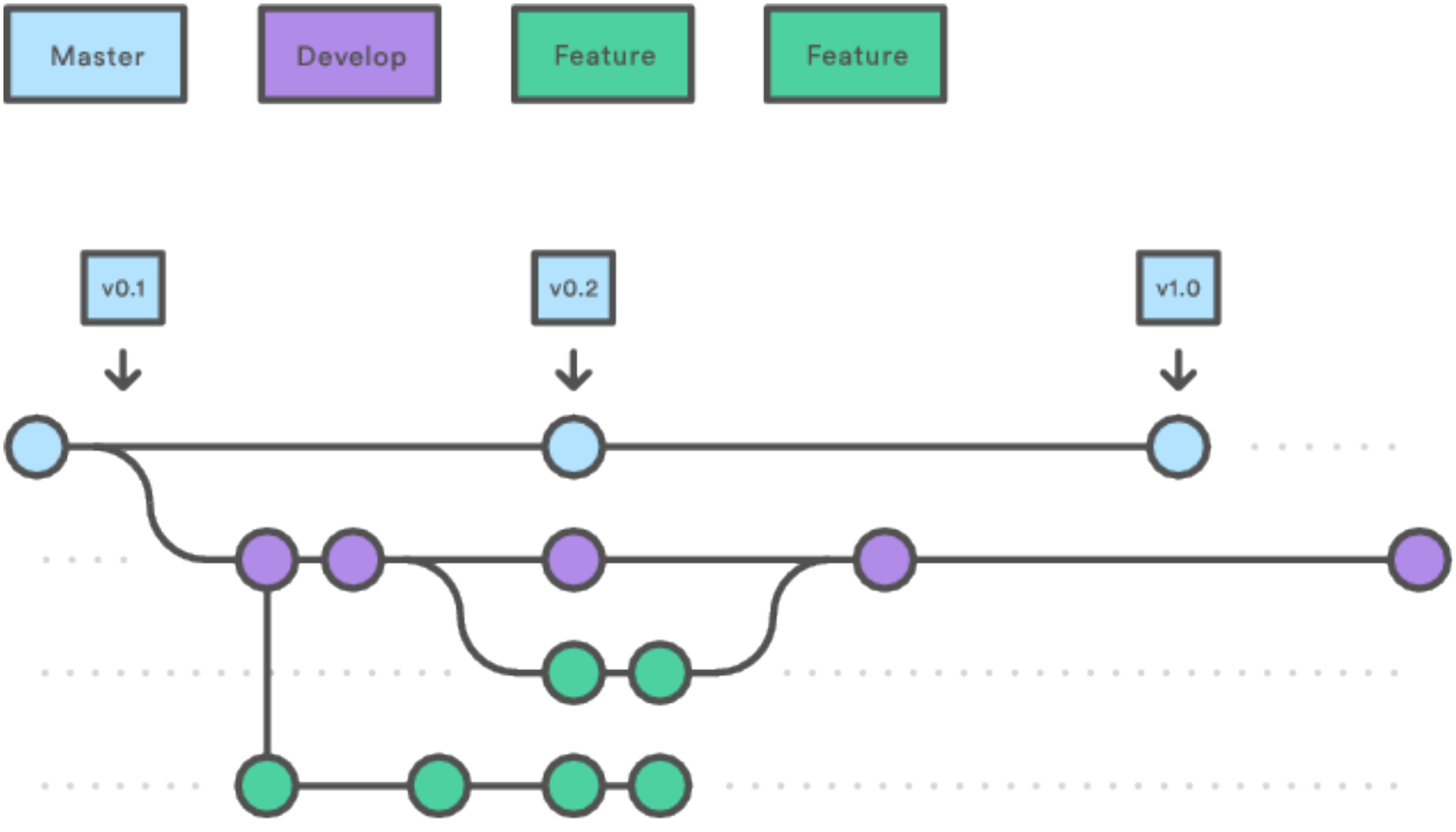
Workflow: gitflow



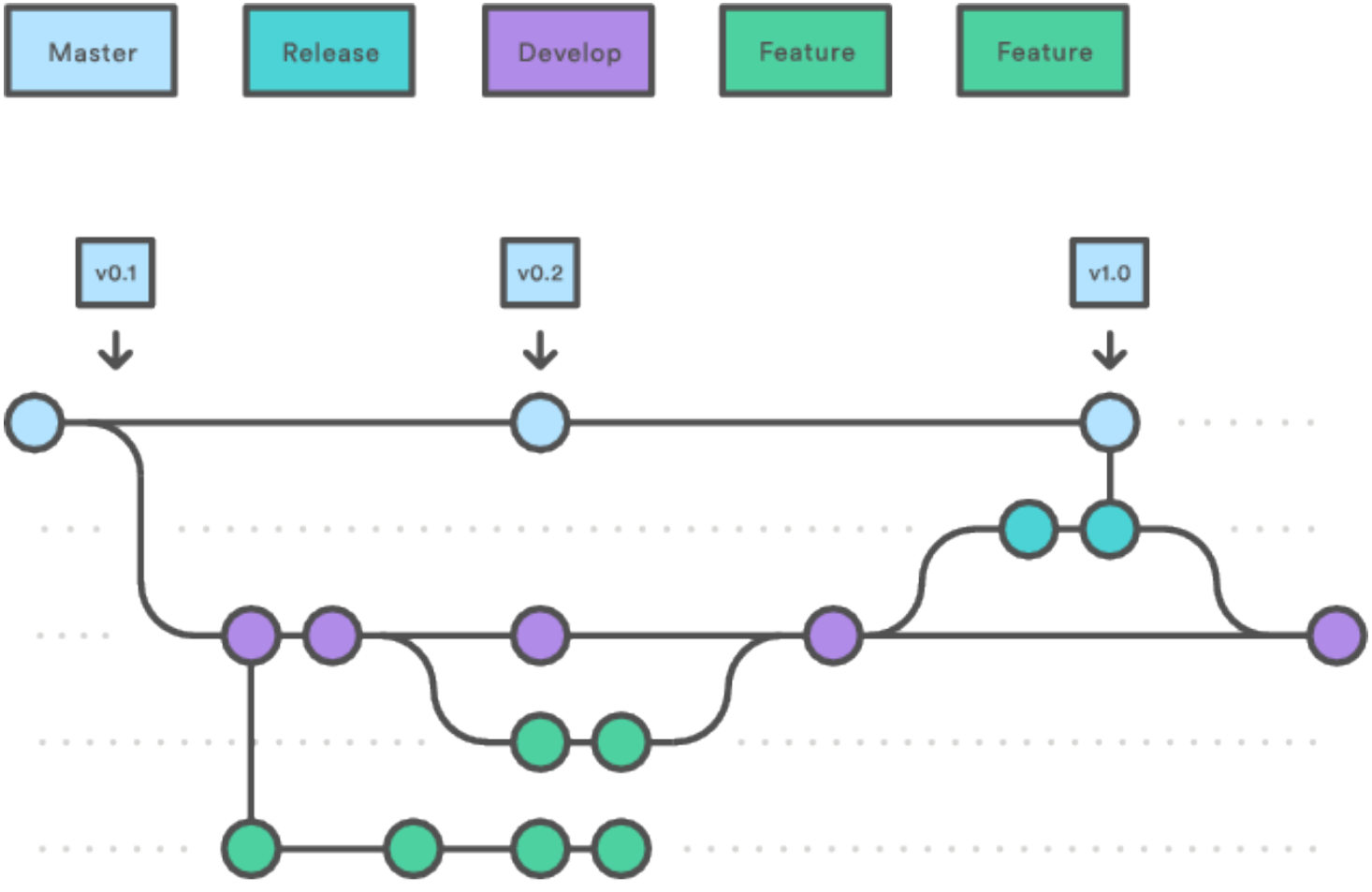
Workflow: gitflow



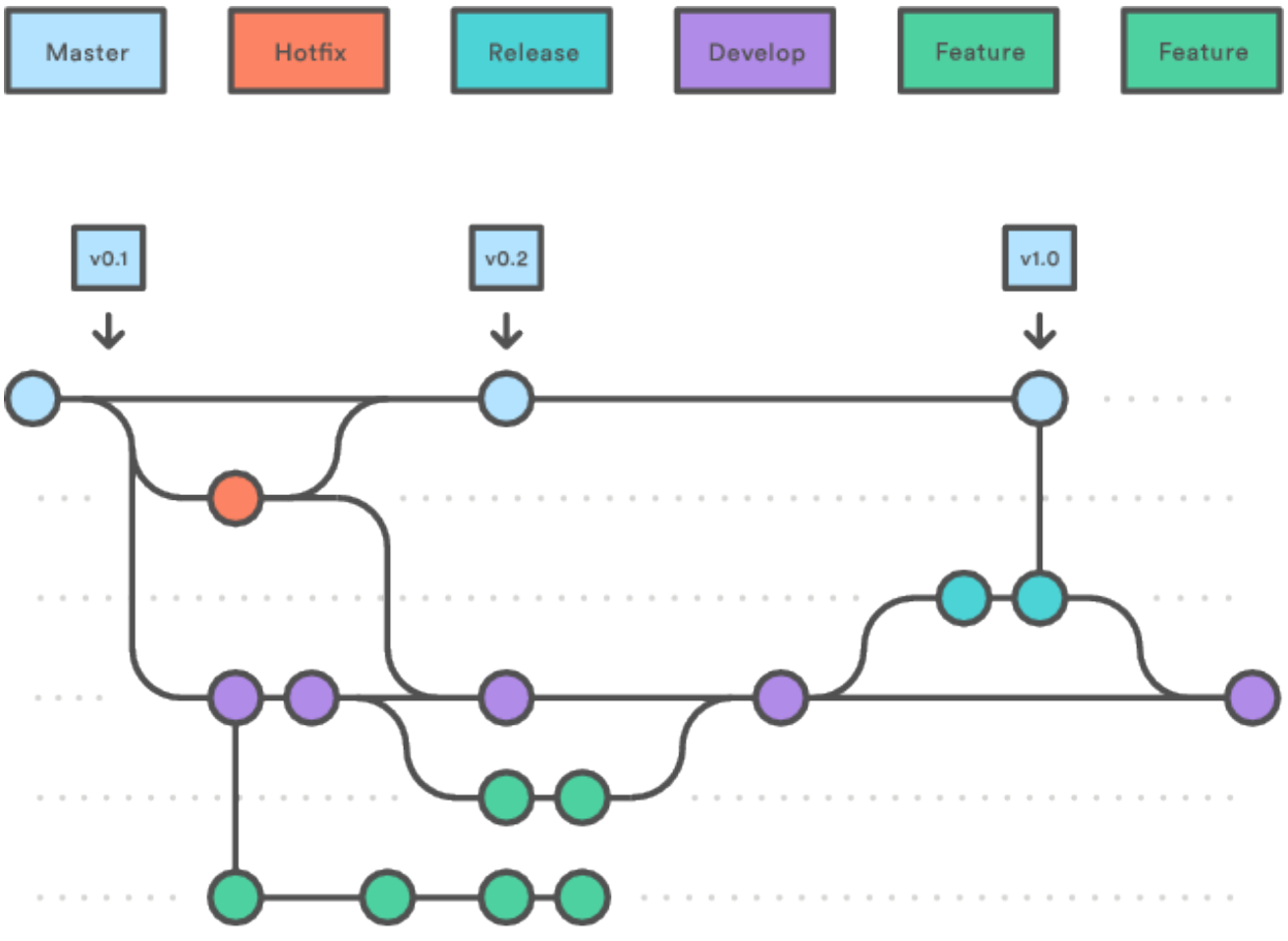
Workflow: gitflow



Workflow: gitflow



Workflow: gitflow



Exercise 13: Branches/merges

Questions:

- Create a branch with the name of your choice
- Modify a file and commit the changes
- Checkout the `master` branch
- Modify a file and commit the changes
- Merge the branch previously created in the `master` branch
- List all branches
- Print the logs of the different modifications
- Give us the name of the branch you created **[moodle]**

Sources and extra infos

Sources

- Wikipedia
- http://git-scm.com
- Manpages: rsync, git
- https://www.atlassian.com/git/
- http://nvie.com/posts/a-successful-git-branching-model/

Learn more

- Git with a game: http://learngitbranching.js.org/