Universität Zürich[UZH]
Physik-Institut

Introduction to PHY224

1 Introduction PHY224 block course

The PHY224 is a block course dedicated for teaching how to program in `C/C++` language in a `Linux` environment. The course will cover some of the basic tools available in `C` and `C++`. It is based on a course first set up by Steven J. Lee in 2020, and further developed since then. The outline of the course is as follows:

1. Introduction to programming in `Linux`.
2. Introduction to basics of programming.
3. Data processing in `C/C++`.
4. Data handling in `C/C++`.
5. Memory management in `C/C++`.
6. Handling large data in `C` and `C++`.
7. Objects and Classes.
8. Scripting using `ROOT 6`.
9. Programming using `ROOT 6` packages.
10. Parallel programming in `C/C++`.

Each lecture will be held in the announced classroom (Y27-H-35/36) using the following timetable:

- 0900∼0915: In the lecture room, the lecturer will provide introduction to and some examples of tools to be used in the following practical session.
- 0915∼1015: The students are expected to go through the exercises given in the lecture notes during this time. Teaching assistants (TAs) will be available for the students to ask questions.
- 1015∼1030: There will be a 15 minutes break before continuing with the lecture.
- 1030∼1045: In the lecture room, the lecturer will provide introduction to and some examples of more tools to be used in the following practical session.
- 1045∼1145: The students are expected to go through the exercises given in the lecture notes during this time. TAs will be available for the students to ask questions.
- 1145∼1200: The students may ask any of the remaining questions that they may have regarding the lecture material or the exercises. The lecturer will give the students a concluding remark and end the lecture for the day.

The course will start on 17/08/2026 and end on 28/08/2026. The lectures will be held on each weekday from 0900 until 1200 until the last day of the course.

1.1 Structure of the lecture notes

All of the lecture notes are designed to be used during the course. The lecturer will display the lecture notes as well as a `Linux` terminal during the lecture to demonstrate some examples live. Your are strongly encouraged not to blindly copy-and-paste the materials in the lecture notes. In fact, in earlier versions of this course this was even occacionally prohibited. This is no longer the case, but the idea behind it still holds. If sample code is copy-and-pasted, make sure to go through it in all detail to avoid that programming subtleties are overlooked! In general, your are encouraged to write proper programs from scratch rather than copy-and-paste code.
Within the lecture notes you will find the examples of commands, scripts and codes to be colour-coded. There are three colours:

```
bashcommand
```

is used for demonstrating what to type into a `Linux` terminal (commands).

```
1 bashscript
```

is used for text to be written in a text editor and saved as a file to be executed from a `Linux` terminal (scripts).

```
1 c++code
```

is used for proper `C++` code to be written in a text editor, saved then, compiled into an executable and executed from a `Linux` terminal (codes).
More information on the above can be found in the first lecture note.

1.2 Course requirements

You need to have access to a computer with internet access. One option is to be prepared to access one of the local computers located physically in the physics institute through `Secure Shell` (`SSH`) remotely. In Section 3 you will find a set of instructions on how to use `SSH` from a computer with any `Linux` operating system, `Windows 10/11` or `MacOS X`. If you have any questions regarding how to gain access to the machines in the physics institute please contact the IT team on `sysadm@physik.uzh.ch`.
Alternatively, and recommended for this course, you can run everything locally on your computer, basically independent of your original operating system, by running `Linux` as a virtual operating system. In Section 2 you will find instructions on how to do so, explicitly for installing `openSUSE Leap 16.0` through `Oracle VirtualBox`. In fact, this year's lecture will be presented using this configuration. I find it the most straightforward and comfortable solution under `Windows 10/11`, while working directly under a primary operating system is presumably easier under a `Unix`-based operating system (`MacOS` or any `Linux` distribution).
Section 2 should be instructive in any case as it also lists the minimal set of features to be installed to follow this course, on the example of an `openSUSE Leap 16.0` installation which has been set up from scratch. Details might slightly differ for other versions and distributions (pre-installed features, package managers, etc.).

2 Setting up Linux openSUSE under Oracle VirtualBox

Running a virtual operating system requires that *hardware virtualization* be *enabled* in your BIOS/UEFI. Ensure this setting is active before starting the installation. Steps 1–5 describe how to set up `openSUSE Leap 16.0` as a virtual machine, the remaining steps should apply equally if it is set up as the primary operating system.

1. Download and install `Oracle VirtualBox` as described here: https://www.oracle.com/de/virtualization/technologies/vm/downloads/virtualbox-downloads.html , if you don't have it already available on your system. The exact version should not matter too much.

2. Download the `openSUSE Leap 16.0` ISO from here: https://get.opensuse.org/leap/16.0/ . Usually, the version for *Intel or AMD 64-bit desktops, laptops, and servers (x86_64)* should be the right one, and you are recommended to download the *Offline Image*.

3. Start `Oracle VirtualBox` and click on *New...* in its main window (see Fig. 1) to create a new virtual machine, and fill the relevant fields in the subsequent pop-up windows:[1]

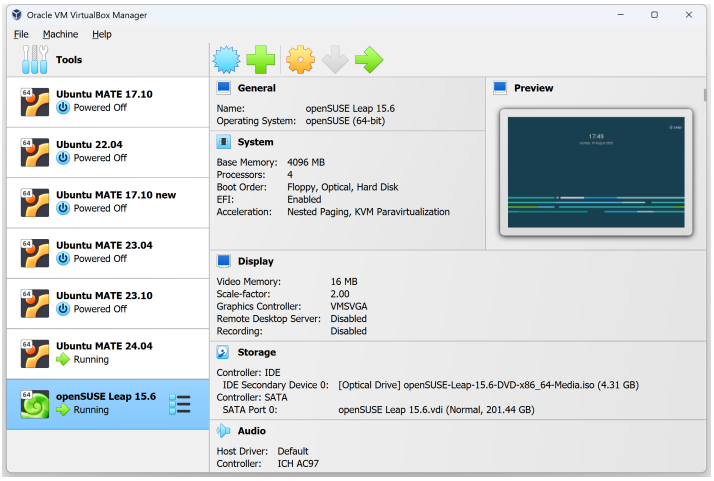

Figure 1: Main window of `Oracle VirtualBox 7.0.20`.

   (a) Choose a name for the new operating system and select the downloaded ISO image to be used for the installation (see Fig. 2). The remaining fields should be filled automatically. Click the *Next* button to continue.

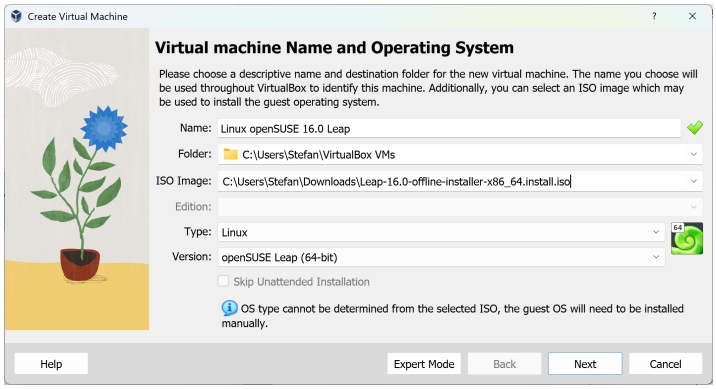

Figure 2: Create Virtual Machine in `Oracle VirtualBox 7.0.20`: *Virtual machine Name and Operating System*

   (b) Choose how much memory and how many (virtual) CPUs should be provided to the virtual operating system (see Fig. 3). The values of 4GB of memory and 4 CPUs are quite minimal, but should be sufficient for all examples of this course. Note that you can change them again while your new virtual operating system is not running. Make sure that sufficient resources remain for your host system.

   (c) Choose the desired size of your virtual hard disk (see Fig. 4). Note that you cannot increase it later, and that the requested amount of disk space is not blocked

[1] Note that Figs.1–5 refer to an older version of `Oracle VirtualBox`, namely `Oracle VirtualBox 7.0.20`. Slighly differently arranged, you should find the same settings under the latest version as well.

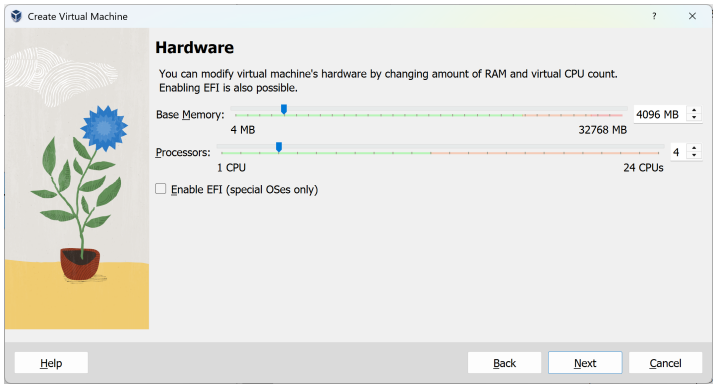

Figure 3: Create Virtual Machine in `Oracle VirtualBox 7.0.20`: *Hardware*

immediately, but only when used, unless you check the *Pre-allocate Full Size* box. You can, however, add further virtual disks later. This course with all required software installed locally (including `ROOT`) should not need more than 20GB of disk space.

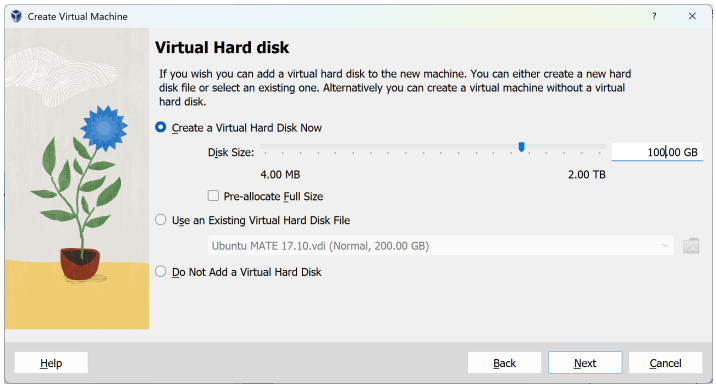

Figure 4: Create Virtual Machine in `Oracle VirtualBox 7.0.20`: *Virtual Hard Disk*

(d) Check if you are happy with all your settings (see Fig. 5). If so, complete the creation of your new virtual machine by clicking the *Finish* button.

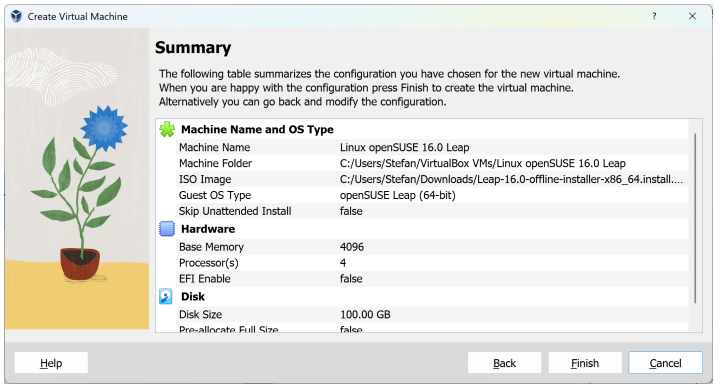

Figure 5: Create Virtual Machine in `Oracle VirtualBox 7.0.20`: *Summary*

4. Select your newly created virtual machine and click on *Start*. In the boot menu that opens up, choose *Installation* and follow the `openSUSE Leap 16.0` installer. Make sure to take care in particular of the following points:

   – Choose *Leap 16.0* when asked to select a product.
   – Choose your preferred *Language/Keyboard/Time zone* settings.

– Select your preferred graphical environment — I suggest *KDE Applications and Plasma Desktop*. If you don't select any, you only get a shell login, from where you could still install it, but it's more straightforward to to it directly here.

– Define a *First user* and a *Root user* with corresponding passwords.

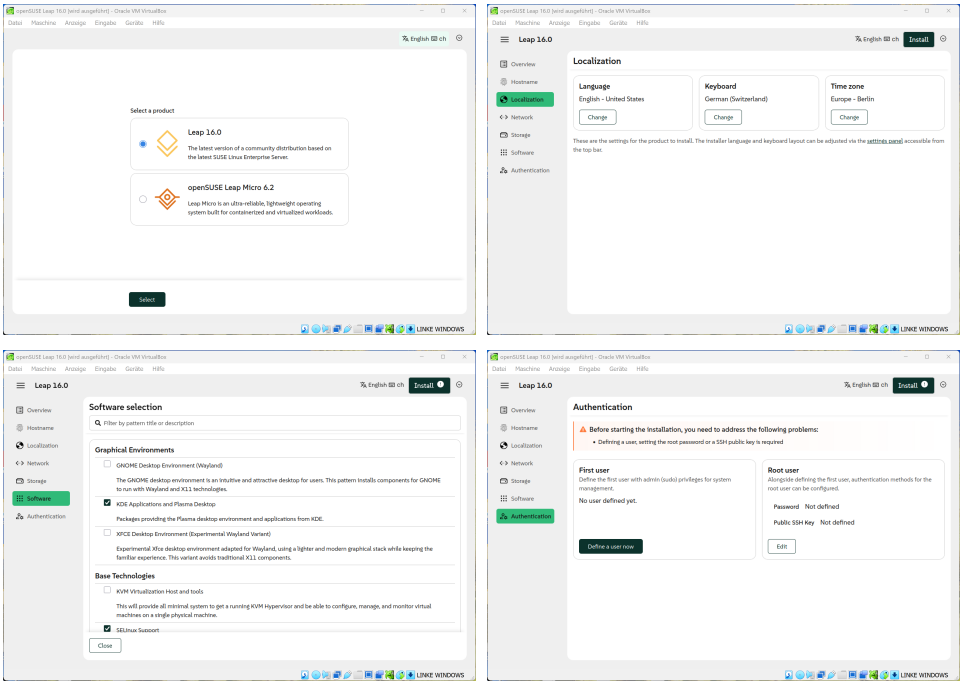

Figure 6: Installation of `openSUSE Leap 16.0`

5. Once the installation has completed, you can work with this virtual machine basically as if it was a regular operating system. You may have to virtually remove the installation medium from your virtual optical drive in order to properly boot your new system from the hard disk. Only the *Host* key, which can be chosen in the settings of `Oracle VirtualBox` (I prefer using the *Windows* key), is not available inside the virtual machine, since it allows you to switch to your host system. We now continue with the tools required for this course which are usually not available by default, and how to install them under `openSUSE Leap 16.0`.

6. First we need an editor to write codes and scripts. In this course I will use `Emacs`, but feel free to use whichever editor(s) you prefer. To install `Emacs`, type

```
sudo zypper install emacs
```

confirm with your *root* password and follow the dialogue in the `shell`.

7. To compile the codes we are going to write, it is highly recommended to use a build system in general (for very simply programs, a one-lined compilation command will be sufficient, but it is unavoidable for realistic applications). A standard build system, which is going to be used in this course, is `make`. To install it, type

```
sudo zypper install make
```

8. Since this is a `C++` course, we need corresponding compilers, which are not necessarily available by default. We can go for the standard version. To do so, type

```
sudo zypper install gcc gcc-c++
```

9. In Lecture 3, we will use the `ncurses` library in one of the programs we discuss. This requires the respective development package, which is to be installed via

```
sudo zypper install ncurses-devel
```

10. In Lecture 10, we will have a quick look into parallel programming. In that context we will use a `bash`-package called `gnu-parallel`, which needs to be installed through

```
sudo zypper install gnu_parallel
```

11. The most challenging tool to be installed is `ROOT 6`, since it has several non-trivial dependencies that need to be provided. Moreover, for `openSUSE` it is not available through the package manager and thus needs to be installed directly from its source code, which is a good exercise though.

    To download the source code of `ROOT 6`, you will need the version control system `git`, and since `ROOT` relies on the build system `cmake`, you need that as well. Moreover, there are several dependencies required by `ROOT`, which are collectively stated in the following. Note that this list might not be exhaustive. To install the prerequisites for the `ROOT 6` installation, type

```
sudo zypper install git cmake python3-base python3-devel fftw3-devel libX11-devel
↪  libXpm-devel libXft-devel libXext-devel libxml2-devel openssl-devel libuuid-devel
↪  giflib-devel libjpeg8-devel libpng16-devel liblz4-devel libzstd-devel
```

    Now you can download the `ROOT 6` source code. In the following example, it will go into the folder `$HOME/tools/root/`, but feel free to organize it any other way, of course:

```
mkdir $HOME/tools && cd $HOME/tools
git clone --branch latest-stable https://github.com/root-project/root.git
cd root
git checkout v6-40-02
```

    It is common practise to do the compilation in a separate build folder, in this case `$HOME/tools/root-build`, and to copy the outcome into an installation folder, here `$HOME/tools/root-install`. The latter is the only part that is eventually accessed. For the installation, execute the following commands subsequently:

```
mkdir $HOME/tools/root-build && cd $HOME/tools/root-build
cmake ../root -DCMAKE_INSTALL_PREFIX=$HOME/tools/root-install
cmake --build . -- -j2
cmake --install .
```

    The first command creates and switches to the build folder, while the second configures the build system according to your settings. The third executes the actual compilation, which is by far the most time-consuming part. In total, for `ROOT 6` it can take several hours, depending on the available resources. The option `-j2` asks for parallelization on two (virtual) cores, you can change it according to your available cores and memory. If the installation is interrupted for any reason at this stage, you can simply execute the very same command again to resume the installation; `cmake` will understand what has already been compiled and continue from there. The fourth command sets up the

installation folder. Note that the execution of each step only makes sense once the previous has completed successfully.

Eventually, to make ROOT available in each new shell, add this line to your .bashrc file (and execute it if you want ROOT to be available immediately in the active shell):

```
source ~/tools/root-install/bin/thisroot.sh
```

If you managed to complete all these steps, your system should be prepared to execute all sample codes of this course and to perform all practical exercises locally. If not, possible problems are most likely related to the installation of ROOT, which will only be needed in Lectures 8 and 9, so you still have time to fix it. Otherwise, in the next Section you find instructions on how to log into the machines of the physics insitute (where e.g. ROOT is already available) remotely using Secure Shell from different operating systems. If you have set up your virtual machine, you can do so from there, following the instructions for Linux.

3 Secure Shell (SSH)

Secure Shell (SSH) is a very basic tool available for many operating systems by default. SSH allows users to open a shell or an encrypted enclosed environment to log into computers and operate them remotely. However, there are multiple versions and functions of SSH, and for the purpose of this course you need to be able to forward the X-display. In a Linux environment, the X-server handles the display by default (unless reconfigured), and forwarding X-display is typically not an issue. However, some external tools such as Xming (http://www.straightrunning.com/XmingNotes/) or XQuartz (https://www.xquartz.org/) are required for Windows and MacOS platforms, respectively. Below you find instructions on how to use them.

3.1 From Linux

As stated above, using SSH with X-display forwarding is very straightforward. You only need to open a terminal and type in the following:

```
ssh -X username@remote.com.put.er
```

Obviously, you need to replace *username* and *remote.com.put.er* with your actual username and the domain. For instance

```
~> ssh -X stelee@ohm21.physik.uzh.ch
The authenticity of host 'ohm21.physik.uzh.ch (172.23.201.151)' cant be established.
ECDSA key fingerprint is SHA256:x7ERYVt/uJs0Q44o4sBqo28r96ecHthfmrD2KW9q3pQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ohm21.physik.uzh.ch,172.23.201.151' (ECDSA) to the list of known
↪  hosts.
Password: #Here, you will not see your password being typed.
Have a lot of fun...
stelee@ohm21:~>
```

To verify that the X-display is being forwarded correctly, try launching one of the simplest X-applications, like

```
xclock
```

You should be able to see the X-clock like in Fig. 7. If so, your system is good to go.

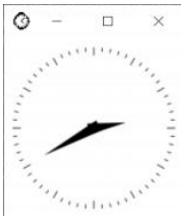

Figure 7: X-clock from your display may appear in different colours.

3.2 From MacOS X

MacOS X will allow you to SSH into a Linux computer remotely by default from the terminal. However, MacOS X does not have an X-server installed by default. This is where you have to install something external. The instructions are as follows.

1. Download and install XQuartz (https://github.com/XQuartz/XQuartz/releases/download/XQuartz-2.8.5/XQuartz-2.8.5.pkg).
2. Using a text editor, add the following lines

```
1 XAuthLocation /opt/X11/bin/xauth
```

to

```
/etc/ssh/ssh_config
```

If done properly, the above file should look as follows:

```
#   Tunnel no
#   TunnelDevice any:any
#   PermitLocalCommand no
#   VisualHostKey no
#   ProxyCommand ssh -q -W %h:%p gateway.example.com
#   RekeyLimit 1G 1h

Host *
        SendEnv LANG LC_*
XAuthLocation /opt/X11/bin/xauth
```

Figure 8: Your /etc/ssh/ssh_config should look similar to the above.

3. Test your setup by connecting to a remote server and opening the X-clock.

```
ssh -X username@remote.com.put.er
```

Obviously, you need to replace *username* and *remote.com.put.er* with your actual username and the domain. For instance

```
ssh -X stelee@ohm21.physik.uzh.ch
```

and once logged in

```
xclock
```

You should see the X-clock like in Fig. 9. If so, your system is good to go. If you have any trouble making X-server forwarding work on MacOS X, please let us know.

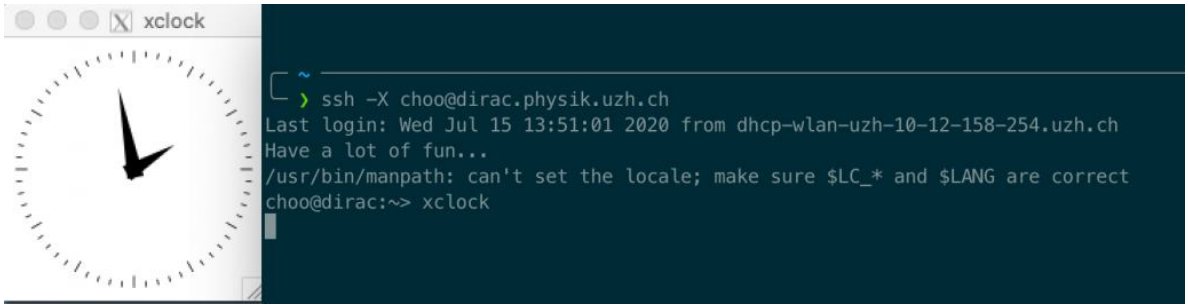

Figure 9: When you log into your own account in the physics server, your terminal output should show similar results. You can use the `xclock` command to check that the `X`-display is being forwarded correctly.

3.3 Pimping your Mac setup

The introduction of `MacOS X` in 2001 brought about a major change: the system is based on a BSD `Unix` kernel, making it very similar to `Linux` operating systems. As a result, it can be tweaked to be very suitable for software development with a `Linux` environment, and it can almost transparently be used for most of the tasks `Linux` distributions can do. In this optional section we will go through the most important components.

3.3.1 Install Xcode

The first step is to install the coding environment. Apple ships the develoment tools with `Xcode` IDE (Integrated Development Environment), which contains the compiler, the necessary system libraries, as well as a code editor with debugger and terminal. It is not necessary to use `Xcode` for coding, but it is necessary to install it in order to get the compiler and the system libraries, so let's download it from the App store:

https://apps.apple.com/us/app/xcode/id497799835?mt=12

Once `Xcode` has been downloaded, we can install the development tools. For that, open the terminal and type:

```
xcode-select --install
```

You should now have the tools installed to build `C++` projects.

3.3.2 Upgrading the terminal

The native `MacOS` terminal can do everything, but newer, more feauture-rich terminals have emerged. A convenient and widely used alternative is `iTerm`, which can be downloaded here:

https://iterm2.com/downloads.html

After installing it, you can use it interchangeably with the native terminal.

3.3.3 Installing a package manager

One nice feature of `Linux` distributions is that they come with a package manager, which can download and install a number of precompiled binaries from a central software repository using the command line.
It turns out than open-source projects exist for Mac which provide the same features and power as on `Linux` (some of them maintained by former Apple employees). The two main package managers for `MacOS` are `MacPorts` and `Homebrew`. The choice between the two is

a personal matter, and we will try to provide some basic idea of the pros and cons of each of them here, but it is important to **ONLY INSTALL ONE OF THEM**. Since they are package managers, they will keep a catalogue of installed and available software, and make sure that the versions of the packages match each other. Packages installed by one package manager will not be known by the other, and this can result in clashes and version mismatches, and ultimately render your system unusable.

MacPorts
`MacPorts` is historically the first package manager for `MacOS`, and recieved support from Apple employees. It will offer you the latest and greatest version of each software, making it the best choice for cutting-edge development. As a result, it will tend to install a lot of packages and dependencies, and each update might trigger updates in already installed packages.
Instructions on installing `MacPorts` can be found here: https://www.macports.org/install.php

You can then install packages with the command:

```
port install [package]
```

Homebrew
The other popular alternative is `Homebrew`. `Homebrew` tries to stick as much as possible to the system libraries, and completes `MacOS` by adding the missing components. The resulting configuration will be closer to the native `MacOS` version, and might not allow you to update to the latest versions. On the other hand, the system is left closer to the initial software it shipped with, and is thus less likely to prevent optimal functioning with tools specifically targeting `MacOS`.
Instructions on installing `Homebrew` can be found here: https://brew.sh/

Packages can then be installed with the command:

```
brew install [package]
```

3.4 From Windows 10/11

Things are slightly trickier on `Windows 10/11`. You need to install `Xming`, enable `Windows Subsystem for Linux`, and install a `Linux` operating system within the `Windows 10/11` operating system. The instructions are as follows:

1. Download and install `Xming` (https://www.uzh.ch/dam/physik/Teaching/PHY224/Allgemein/Software/Xming-6-9-0-31-setup.exe)
2. Enable `Windows Subsystem for Linux` by following the red highlights in Fig. 10.
3. `Windows 10/11` will install the feature, and thereafter you must reboot your computer.
4. When the computer has rebooted, go to the *Microsoft Store* and install `openSUSE Leap 16.0`, following the red highlighting in Fig. 11.
5. Once the *Microsoft Store* has installed `openSUSE Leap 16.0`, open it and follow the instructions displayed to create a user and to initialize it.
6. Once initialized, become the *superuser* or *root* by typing in

   ```
   su
   ```

   and when you are prompted to do so, type in your password.

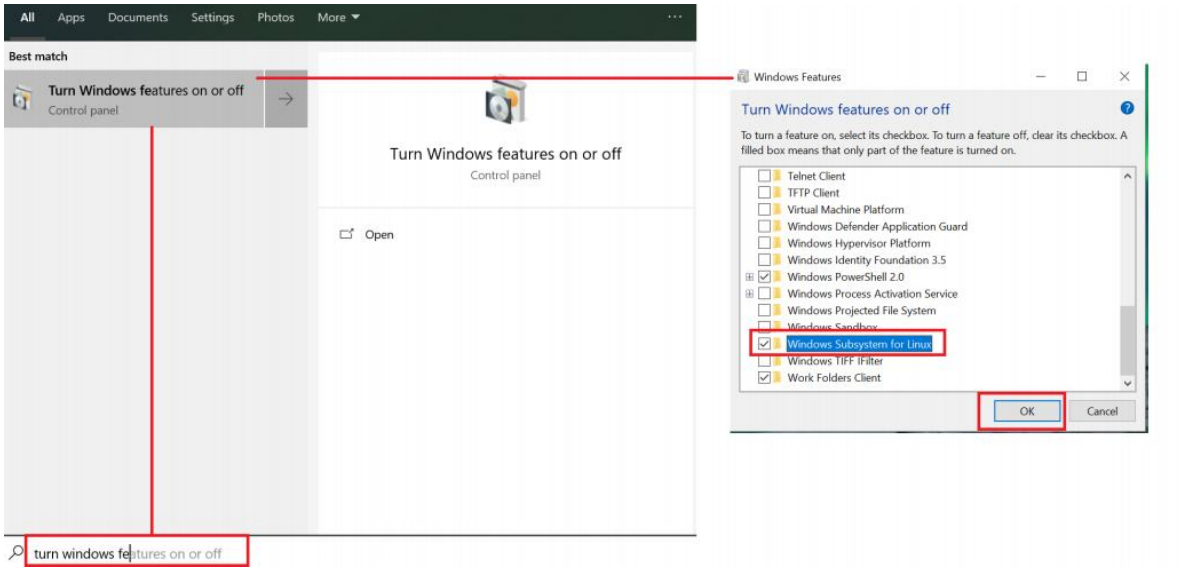

Figure 10: You must search for *Turn Windows features on or off* from the Start menu, and click on *Turn Windows features on or off (Control Panel)*. Then you need to check `Windows Subsystem for Linux` and click *OK*.

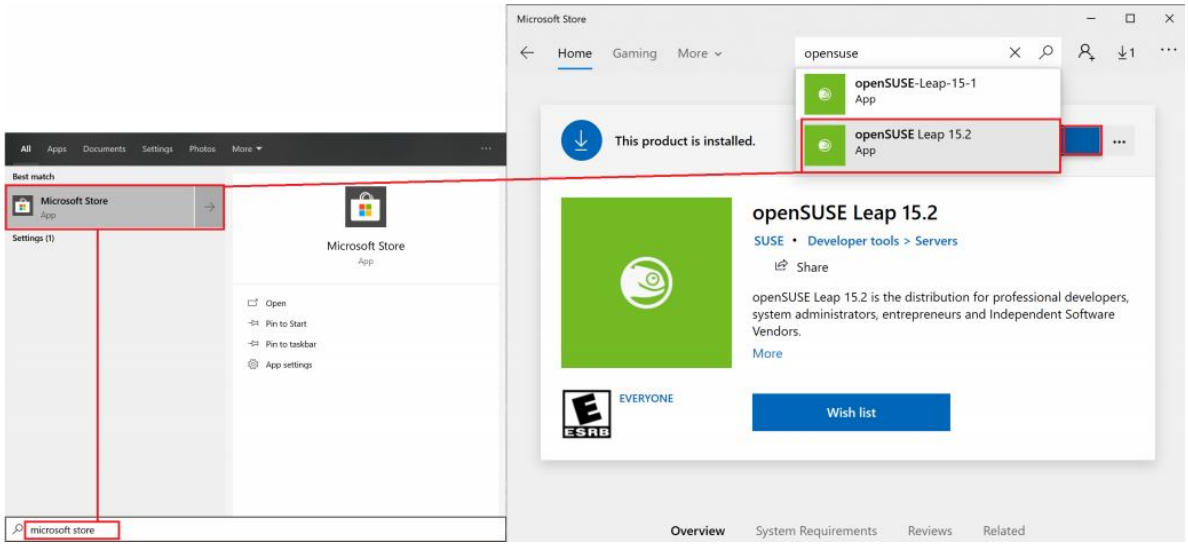

Figure 11: In the *Microsoft Store* you need to search for `openSUSE Leap 16.0` and install it.

7. You need to install a text editor called `Emacs` using the following line.

   ```
   zypper install emacs
   ```

   and say *yes* when prompted for confirmation.

8. You then need to get out of *superuser* by typing in

   ```
   exit
   ```

   or use the key combination CTRL + d.

9. You need to modify your default loading environment called `.bashrc` by the following.

   ```
   emacs -nw ~/.bashrc
   ```

10. Add the following entry to the very bottom of your `.bashrc`

```
export DISPLAY=0:0
```

11. Save and exit by pressing "Ctrl+x, Ctrl+s" then "Ctrl+x, Ctrl+c". If `Emacs` asks if you want to add a line at the end, or to confirm saving the file, type *yes*.

12. Close your `openSUSE Leap 16.0` session by typing

```
exit
```

13. Turn on `Xming` from the start menu.

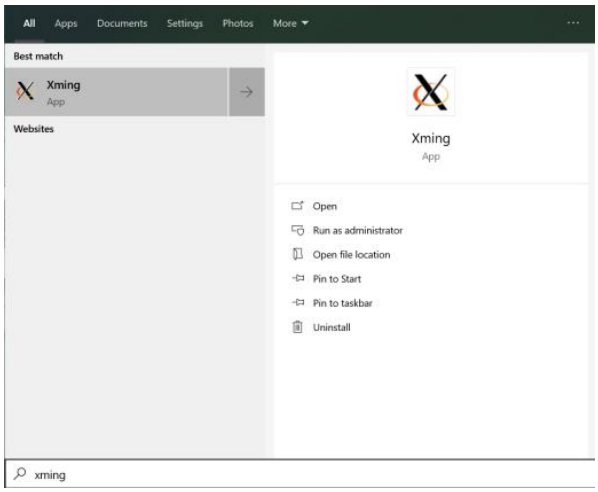

Figure 12: Search for "`Xming` " in start menu and click on `Xming` (App)

.

14. At this point you have installed `Linux` within your `Windows 10/11` operating system. Now you can test it in the same manner as you could on `Linux` by opening any `openSUSE Leap 16.0` application. To log into a remote computer via `SSH`, type

```
ssh -X username@remote.com.put.er
```

Obviously, you need to replace *username* and *remote.com.put.er* with your actual username and the domain. For instance

```
~> ssh -X stelee@ohm21.physik.uzh.ch
The authenticity of host 'ohm21.physik.uzh.ch (130.60.165.162)' cant be established.
ECDSA key fingerprint is SHA256:x7ERYVt/uJs0Q44o4sBqo28r96ecHthfmrD2KW9q3pQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ohm21.physik.uzh.ch,130.60.165.162' (ECDSA) to the list of
↪  known hosts.
Password: #Here, you will not see your password being typed.
Warning: No xauth data; using fake authentication data for X11 forwarding.
Have a lot of fun...
stelee@ohm21:~>
```

To make sure that the `X`-display is being forwarded correctly, you should try opening one of the simplest `X`-applications, like

```
xclock
```

You should see the X-clock like in Fig. 13. If so, your system is good to go.

Figure 13: X-clock from your display may appear in different colours.

Remember to turn on Xming before you try to forward X-display, otherwise nothing will be displayed!

3.5 Version Control

You should note that the only reason that we are asking you to use the local machines in the physics institute is because we know the versions and the packages installed on these machines well.
All of the exercises and the programs for this course have been tested on the machines in the institute. However, your local Linux, MacOS X or Windows Subsystem for Linux (WSL) can just as well handle everything locally from your computer.

This is very much encouraged since we have a limited number of computers available at the institute.

This means that as long as you have the correct packages and correct versions of the packages installed on your local machine, you can do everything for this course without the use of SSH. Please let us know as soon as possible that you would like to use your own local machine instead of those at the institute remotely. Just make sure that you check that your versions of gcc and g++ are later than 7 by typing

```
gcc --version ; g++ --version
```

from your Linux (or Unix[2], or WSL) terminal. You should also note that lectures 8 and 9 will require you to use packages from ROOT (https://root.cern.ch/). First, please note that ROOT is different from the superuser *root*, and that, depending on your programming background, the installation of ROOT may not be trivial.
You are encouraged to try to install ROOT yourselves on your local machines, as there are many resources available online. In particular, it is available as a package in many of the operating system's package managers (including MacPorts, Homebrew, Conda, Debian, Ubuntu (https://cern.ch/amadio/root/root_6.26.10_ubuntu22_amd64.deb), RedHat). However, ROOT is already installed and available for use on the local machines at the institute.

[2] MacOS is Unix-based