



VLAAMS
SUPERCOMPUTER
CENTRUM



Vlaanderen
is supercomputing



Getting Scientific Software Installed

14 Oct 2026

Brussels (+ virtually via Teams)

<https://vscentrum.github.io/gssi-training>

<https://docs.vscentrum.be/compute/software>

support@vscentrum.be

Goal of this training

- **High-level overview of commonly used software installation tools**
- Focus on Linux systems, scientific software, HPC clusters
- Basic information to get started
- Important attention points
- Highlighting strong points & weaknesses
- High-level comparison between tools
- Hands-on: choose your own adventure (exercises, own software, ...)

Speakers & co

This training is organised by the **Flemish Supercomputer Centre (VSC)**

- Maxime Van den Bossche, Steven Vandenbrande (KU Leuven)
- Robin Verschoren (UAntwerpen)
- Kenneth Hoste, Lara Peeters (Ghent University)
- Sam Moors, Jarne Renders, Cintia Willemyns (VUB)



- Slides & exercises available via <https://vscentrum.github.io/gssi-training>
- Hybrid training: in-person attendees in Brussels, remote attendees via MS Teams
- **All talks are being recorded**, will be available via [VSC @ YouTube \(@vschpc\)](#)
- Quick Q&A after each part (if time allows)
- **Use lunch & coffee break to network!**
 - Talk to VSC staff about your specific use case
- Formatting for examples in slides of shell commands & their output:

```
# comment (not a command being run, just text) in blue, starts with #  
command-being-run --in-bold  
  output of command below, indented, not bold, smaller font
```

Running example: installing tblite

- <https://github.com/tblite/tblite> | <https://tblite.readthedocs.io>
- “Lightweight tight-binding framework” (computational chemistry)
- Implemented in Fortran, Python bindings available
- **Provides both a `tblite` command, and a `tblite` Python module**
- Requires various (uncommon) dependencies: mctc-lib, dftd4, simple-dftd3, mstore, ...
- Can be used in combination with various other tools: xtb, DFTB+, QCxMS, ASE
- **Goal is to install `tblite` so that the “First calculation” Python example can be run:**
<https://tblite.readthedocs.io/en/latest/tutorial/python/singlepoint.html#first-calculation>

Read the documentation(s)

- Everything shown in this training is covered in the VSC documentation

docs.vscentrum.be

- VSC sites have extra documentation/info pages

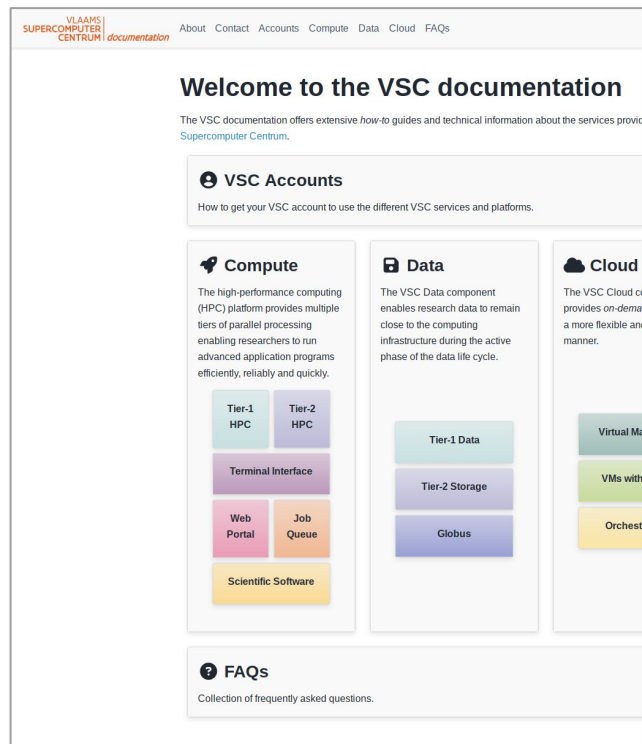
KU Leuven: hpcleuven.github.io/HPC-intro

UGent: docs.hpc.ugent.be

UAntwerp: hpc.uantwerpen.be/support/documentation

VUB: hpc.vub.be

- Software (usually) has its own documentation: **read it!**



Agenda - part 1/2

- [09:00-09:30] *(informal welcome)*
- [09:30-09:40] **Introduction** (*Kenneth Hoste - UGent*)
- [09:40-09:50] **Central software stack + environment modules** (*Robin Verschoren, UA*)
- [09:50-10:00] **European Environment for Scientific Software Installations (EESSI)**
(*Lara Peeters - UGent*)
- [10:00-10:15] **Using Apptainer containers** (*Sam Moors - VUB*)
- [10:15-10:30] **Conda / Mamba / Pixi** (*Maxime Van den Bossche - KU Leuven*)
- [10:30-10:45] **Python: pip + venv** (*Steven Vandenbrande - KU Leuven*)
- [10:45-11:00] **Other language package managers: R, Julia** (*Jarne Renders - VUB*)
- [11:00-11:30] *(coffee break)*

Agenda - part 2/2

- [11:30-11:45] **Software compilation** (*Cintia Willemyns - VUB*)
- [11:45-12:00] **Building Apptainer containers** (*Sam Moors - VUB*)
- [12:00-12:15] **EasyBuild + Spack** (*Kenneth Hoste - UGent*)
- [12:15-12:30] **When should I use which tool + Conclusions** (*Kenneth Hoste - UGent*)
- [12:30-13:30] (*lunch*)
- [13:30-15:00] **Hands-on: choose your own adventure!**
- [15:00-15:30] (*coffee break*)
- [15:00-16:30] **Hands-on (continued)**
- [16:30-17:00] **Show & tell** (what did you work on?)

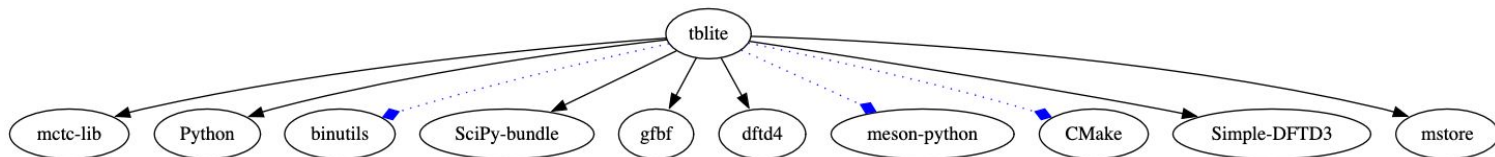
Getting (scientific) software installed is... fun!

To get started, let's look at a couple of important aspects of installing software:

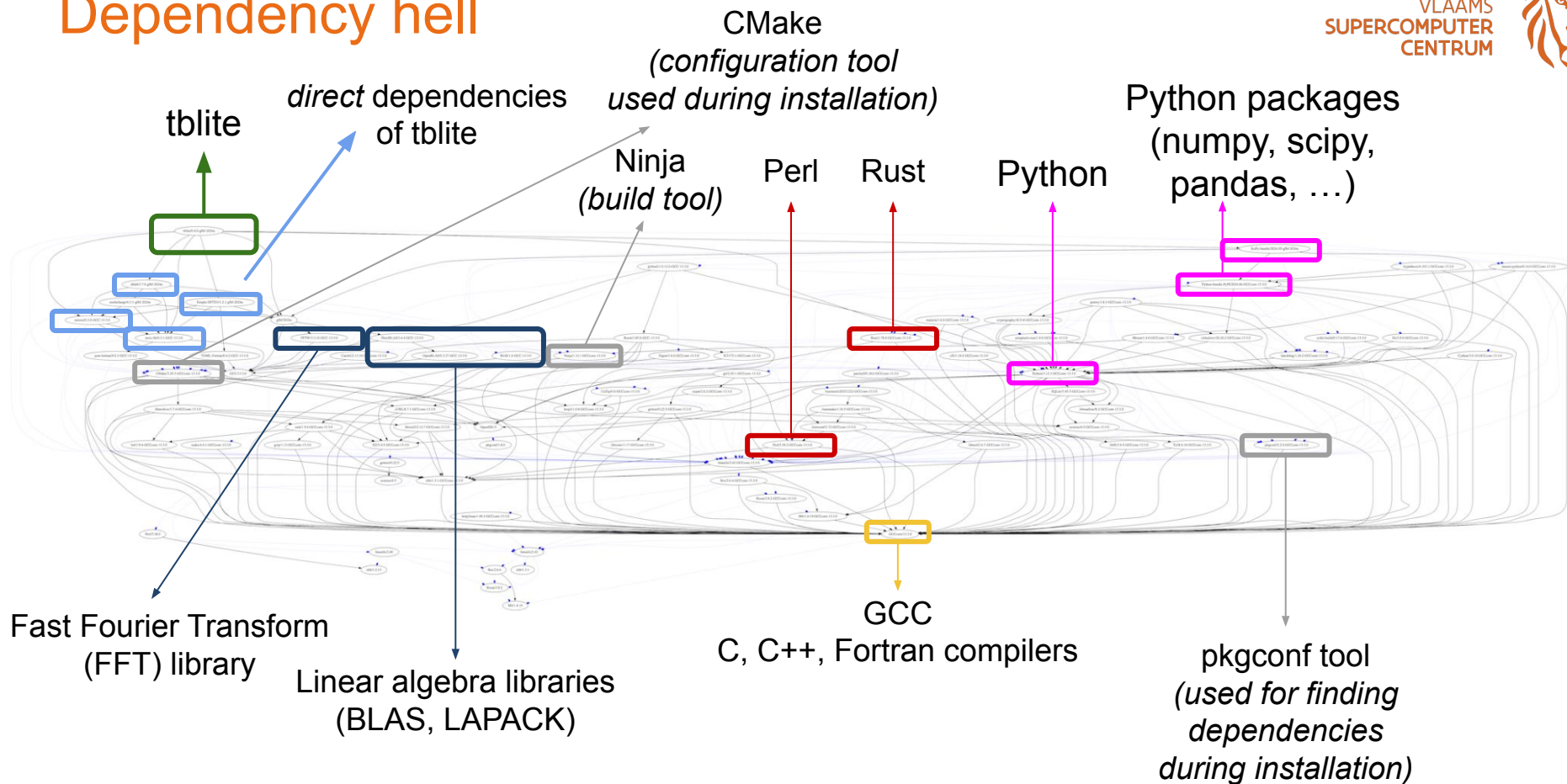
- Software applications have **dependencies** (libraries, tools, etc.)
- Humans write **source code**, computers run **binary** programs (hardware instructions)
- There's a distinction between **interpreted** and **compiled** software
- **Hardware platform** matters: CPU vs GPU, CPU microarchitecture, ...

Software dependencies

- Software applications don't exist in isolation, they **depend** on other software (libraries, ...)
- You need a **compiler** to build software from *source code* to *binary* format (so you can run it)
- Software implemented in Python requires to have a Python **interpreter** installed
 - A *compatible* version of Python interpreter, may require additional Python packages, ...
- C++ software often *links* to multiple different **software libraries** for specific functionality
- Dependencies are often implemented in a compiled programming language (like C++, Fortran, Rust, ...), which do the “heavy lifting”
 - Python bindings are mostly there to make the software easier to use



Dependency hell



(dependency graph for tblite 0.4.0, created with EasyBuild)

Interpreted vs compiled software

- The processor (CPU) in a computer can only execute **binary code** (machine instructions)
- Software **source code** must be “translated” to a **binary executable**
- For *compiled* programming languages (C, C++, Fortran, Rust, ...) this is done at installation time (when software is built), using a **compiler**
- For *interpreted* programming languages (Python, Julia, Perl, ...) this is done at runtime, when the software is being used, by an **interpreter**
 - Some languages use Just-In-Time (JIT) compilation to speed up the software while it is running (Java, Python via PyPy interpreter, ...)

```
int main() {  
    std::cout << "Hello World!";  
    return 0;  
}
```

compiler



```
print("Hello World!")
```

interpreter



Binary programs use hardware instructions (ISA)

- A binary executable consists of **machine instructions** for a particular *Instruction Set Architecture (ISA)*, a set of instructions the CPU supports
- A **generic binary** only uses instructions that are supported by all (modern) CPUs
 - Good: can run on any system (with CPU of correct family, like x86_64 or aarch64)
 - Bad: is probably not using the full capabilities of your CPU (so may be slow)
Example: AVX-512 vector instructions in modern Intel/AMD CPUs, SVE in Arm CPUs
 - Running a binary that includes unsupported instructions will result in errors like:

```
Illegal instruction (SIGILL)
```
- The binaries you use should be **optimized** for the hardware on which it will be run!
 - Important for good **performance** of the software, to use full capabilities of your CPU
 - => Use a compiler option like **-march=native** to optimize for the CPU of your system

- There are different CPU families: **x86_64** (AMD, Intel), **aarch64** (64-bit Arm), ...
 - An x86_64 binary will not run on an aarch64 CPU, and vice versa
(unless additional emulation software like QEMU or Apple's Rosetta is used, but that may have a large negative impact on performance)
 - An additional (high-performance) CPU family is coming (soon?): RISC-V
- Within a single CPU family, there are different **generations of CPUs**
 - Each generation supports a **particular ISA**, the CPU microarchitecture implements it
 - For Intel CPUs: Haswell, Skylake, Cascade Lake, Ice Lake, Sapphire Rapids, ...
 - For AMD CPUs: Zen2 (Rome), Zen3 (Milan), Zen4 (Genoa), Zen5 (Turin), ...
- A binary built for CPU generation X can typically run on CPU of generation X+Y, but:
 - Not the other way around! => **Illegal instruction** error
 - It will typically run slower than it could be if properly optimized for specific CPU...

- Things get (even) more ~~interesting~~ complicated when you also take into account GPUs...
- There are also multiple **GPU families** (NVIDIA, AMD, Intel)
- There are multiple **generations** (GPU architectures) within a single family
 - For NVIDIA: Ampere (A100, A2, ...), Hopper (H100), Blackwell (B100), ...
- **Additional software dependencies** are required to software run on GPUs:
GPU drivers, CUDA runtime library, specialized libraries like cuFFT, ...
- Binaries for NVIDIA GPUs can be specific to specific GPU architectures (object code), or can support a range of supported GPU architectures via PTX & JIT compilation
- Software needs to be **configured to enable GPU support** (if it supports that)
- *Note: GPUs are mostly out of scope for this training...*

Detecting CPU & GPU

- There are **tools** available to **detect which CPU + GPU** there is in a system
- For CPUs (on Linux systems):
 - `/proc/cpuinfo` “file” (use `cat /proc/cpuinfo` to read it)
 - `lscpu` command
 - `$VSC_ARCH_LOCAL` environment variable (only on VSC clusters)
 - `archspect` CPU detection tool (<https://github.com/archspect/archspect>)
- For GPUs (on Linux systems):
 - NVIDIA: `nvidia-smi` command (Not there? => No NVIDIA GPU available!)
 - AMD: `amd-smi` command (AMD equivalent to `nvidia-smi`, was `rocm-smi`)

Different use cases for **different levels of expertise**:

- **Using software** that is already available, installed by somebody else
 - Central software stack, EESSI, running existing container images
- ***Install* software on your own**, incl. bringing pre-built binaries to the cluster
 - Conda / Mamba / Pixi, language pkg mgrs (Python venv/R/Julia/Rust+cargo)
- **Building/compiling software** (from source code)
 - Manual build + install, creating container images, EasyBuild and/or Spack

Spoiler: this high-level overview is our end goal



<i>It depends...</i>	Central software stack	EESSI	(pre-built) containers	Conda & co	Language pkg mgrs	Manual installation	EasyBuild	Spack
<i>Supported software</i>	👍👍👍	👍	👍👍	👍👍	👍	👍👍👍	👍👍👍	👍👍👍
<i>User experience</i>	👍👍👍	👍👍👍	👍👍	👍👍	👍👍	👎👎	👍	👍
<i>Performance of installed software</i>	👍👍👍	👍👍👍	👍👎	👍👎	👍👎	(👍👍)	👍👍👍	👍👍👍
<i>Reproducibility of the installation</i>	👍👍👍	👍👍👍	👍👍	👍	👍	👍👎	👍👍	👍👍
<i>Time required to install software</i>	👍👎	👎	👍👍	👍	👍👍	👎👎👎	👎👎	👎
<i>Testing of installations</i>	👍👍	👍👍👍	👎	👍	👍	👎	👍👍👍	👍
<i>Storage (volume, # files)</i>	(👍👍)	(👍👍)	👍👍	👎👎👎	👎	👎	👎👎	👎👎
<i>Learning curve</i>	👍👍👍	👍👍👍	👍	👍👍	👍👍	👎👎👎	👎👎	👎👎
<i>Getting help (in VSC)</i>	👍👍👍	👍👍	👍👍	👍	👍👍	👍	👍👍👍	👍

Agenda

- [09:00-09:30] *(informal welcome)*
- [09:30-09:40] **Introduction** *(Kenneth Hoste - UGent)*
- [09:40-09:50] **Central software stack + environment modules** *(Robin Verschoren - UA)*
- [09:50-10:00] **European Environment for Scientific Software Installations (EESSI)**
(Lara Peeters - UGent)
- [10:00-10:15] **Using Apptainer containers** *(Sam Moors - VUB)*
- [10:15-10:30] **Conda / Mamba / Pixi** *(Maxime Van den Bossche - KU Leuven)*
- [10:30-10:45] **Python: pip + venv** *(Steven Vandenbrande - KU Leuven)*
- [10:45-11:00] **Other language package managers: R, Julia** *(Jarne Renders - VUB)*
- [11:00-11:30] *(coffee break)*

- HPC systems typically provide a **central stack of (scientific) software installations**
- These installations are usually **optimized** for the specific CPU (and GPU) generation on which they are meant to be used
 - Different sets of installations for different cluster(s) or cluster partitions
 - Operating system and its version is also a factor (Rocky Linux 8, RHEL9, ...)
- User interface to this central software stack is typically **environment modules**
 - **module** command, with various subcommands (**avail**, **load**, **list**, **show**, ...)
 - Different implementations, most popular one is Lmod (<https://lmod.readthedocs.io>)
- On VSC clusters, central software stack (**/apps**) is installed with **EasyBuild**

Environment modules: basics

- Check which modules are available

```
module avail
```

- Check which modules are available for a particular software

```
module avail tblite/
```

- Show details for a particular module

```
module show tblite/0.4.0-gfbf-2024a
```

- Load a module to “activate” the software so it can be used

```
module load tblite/0.4.0-gfbf-2024a
```

- Check which modules are loaded

```
module list
```

- Start over (unload (almost) all loaded modules)

```
module purge
```

Loading a module implies updating environment variables: \$PATH, \$LD_LIBRARY_PATH, ...

Always load a specific module version!

Without specifying a version, the default version (which changes over time) will be loaded

GOOD: module load
tblite/0.4.0-gfbf-2024a

BAD: module load tblite

Environment variables for finding software

- **\$PATH** : list of paths in which binary executables can be found

```
# to check whether a command is  
available, use  
which tblite  
# or  
command -v tblite
```

- **\$LD_LIBRARY_PATH** : list of paths in which *runtime* libraries can be found

- **\$LIBRARY_PATH** : list of paths in which libraries can be found to *link* with (at *build* time)

- **\$CPATH** : list of paths in which C/C++ header files can be found (used by compiler)

Also: **\$C_INCLUDE_PATH**, **\$CXX_INCLUDE_PATH**, ...

- **\$PYTHONPATH** : list of paths in which Python packages can be found (at runtime)
Similar environment variables exist for other languages (often even multiple ones...)

```
# to check whether runtime  
libraries are found, use  
ldd /path/to/tblite  
# location of runtime libraries  
can also be "baked in" to a binary  
(RPATH linking)  
readelf -d /path/to/library.so
```

- When using EasyBuild, software is built & installed with a particular (compiler) **toolchain**
- A *full* toolchain (examples: `foss`, `intel`, `iomkl`, ...) consists of:
 - Set of **C, C++, Fortran compilers**: GCC, LLVM, Intel, ...
 - **Communication library (MPI)**: Open MPI, MPICH, MVAPICH, Intel MPI, ...
 - **Linear algebra library (BLAS + LAPACK)**: OpenBLAS, BLIS, Intel MKL, ...
 - **Fast Fourier Transform (FFT) library**: FFTW, FFT interface in Intel MKL, ...
- A full toolchain has multiple compatible subtoolchains (for `foss`: GCC, `gfbf`, `gomp`, ...)
- **You should not mix modules that were installed with different toolchain versions!**
 - For each toolchain version (like 2024a) there is one single compatible GCC version
 - Use `module show` command on toolchain module to check which GCC version is compatible

```
module show foss/2024a
```

Example: central software stack (1/2)



```
# check whether tblite binary is available (no...)
which tblite
/usr/bin/which: no tblite in (...)

# check whether any modules are available for tblite (yes!)
module avail tblite/
----- /apps/antwerpen/modules/rocky9/software-skylake/2024a/all -----
      tblite/0.4.0-gfbf-2024a

# show details of tblite module (homepage, description, dependencies, ...)
module show tblite/0.4.0-gfbf-2024a
...
Description
=====
Light-weight tight-binding framework
...
```

Example: central software stack (2/2)



```
# load tblite module to "activate" tblite for using it
module load tblite/0.4.0-gfbf-2024a

# check whether tblite command is available (yes!)
which tblite
    /apps/antwerpen/skylake/rocky9/tblite/0.4.0-gfbf-2024a/bin/tblite

# run tblite command
tblite --version
    tblite version 0.4.0

# use Python bindings for tblite
python
>>> from tblite.library import get_version
>>> get_version()
    (0, 4, 0)
```

Requesting additional software installations

- You can request additional software to be installed centrally
- Get in touch with the support team of the VSC cluster(s) you want to use:
https://docs.vscentrum.be/contact_vsc.html
- For HPC-UGent Tier-2 + Tier-1 Hortense, use the software installation request form:
https://docs.hpc.ugent.be/software_installation_requests
- Some patience may be needed...



- [09:00-09:30] *(informal welcome)*
- [09:30-09:40] **Introduction** *(Kenneth Hoste - UGent)*
- [09:40-09:50] **Central software stack + environment modules** *(Robin Verschoren - UA)*
- [09:50-10:00] **European Environment for Scientific Software Installations (EESSI)**
(Lara Peeters - UGent)
- [10:00-10:15] **Using Apptainer containers** *(Sam Moors - VUB)*
- [10:15-10:30] **Conda / Mamba / Pixi** *(Maxime Van den Bossche - KU Leuven)*
- [10:30-10:45] **Python: pip + venv** *(Steven Vandenbrande - KU Leuven)*
- [10:45-11:00] **Other language package managers: R, Julia** *(Jarne Renders - VUB)*
- [11:00-11:30] *(coffee break)*

*What if you no longer have to install
a **broad range of scientific software**
from scratch on every laptop, HPC cluster,
or cloud instance you use or maintain,
without compromising on performance?*



European Environment for Scientific Software Installations (EESSI)

- **Shared repository of (optimized!) scientific software installations**
- Avoid duplicate work across by collaborating on a shared software stack
- Uniform way of providing software to users, regardless of the system they use!
- Should work on any Linux OS and system architecture
 - From laptops and personal workstations to HPC clusters and cloud
 - Support for different CPUs, interconnects, GPUs, etc.
- Focus on **performance, automation, testing, collaboration**
- Development effort funded through **MultiXscale** EuroHPC Centre-of-Excellence



E E S S I
EUROPEAN ENVIRONMENT FOR
SCIENTIFIC SOFTWARE INSTALLATIONS

<https://eessi.io>

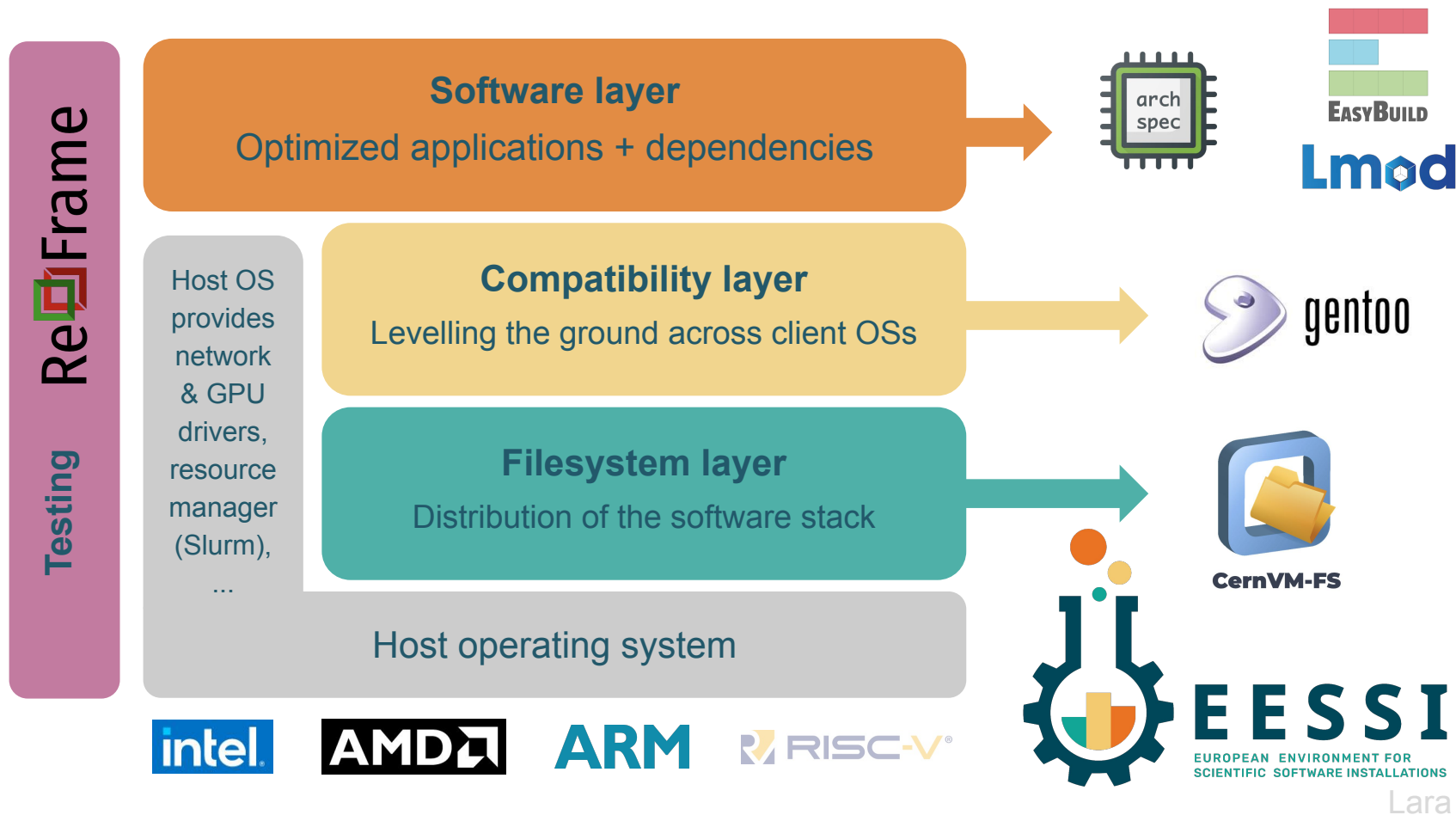


Major goals of EESSI



- Providing a truly **uniform software stack**
 - Use the (exact) same software environment everywhere
 - **Without sacrificing performance** for “mobility of compute” (like is typically done with containers/conda)
- **Avoid duplicate work** (for researchers, HPC support teams, sysadmins, ...)
 - Tools that automate software installation process (EasyBuild, Spack) are not sufficient anymore
 - Go beyond sharing build recipes => work towards a shared software stack
- Facilitate HPC training, development of (scientific) software, ...

High-level overview of EESSI



EESSI ingredients



gentoo linux™

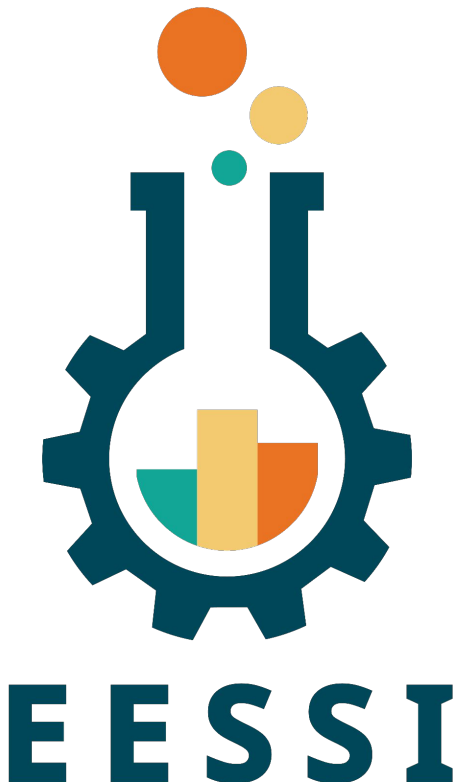
Compatibility layer

Abstraction from the
host operating system



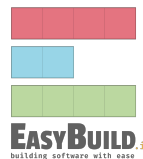
Filesystem Layer

Global distribution of
software installations
via CernVM-FS



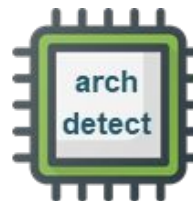
EUROPEAN ENVIRONMENT FOR
SCIENTIFIC SOFTWARE INSTALLATIONS

Software Layer



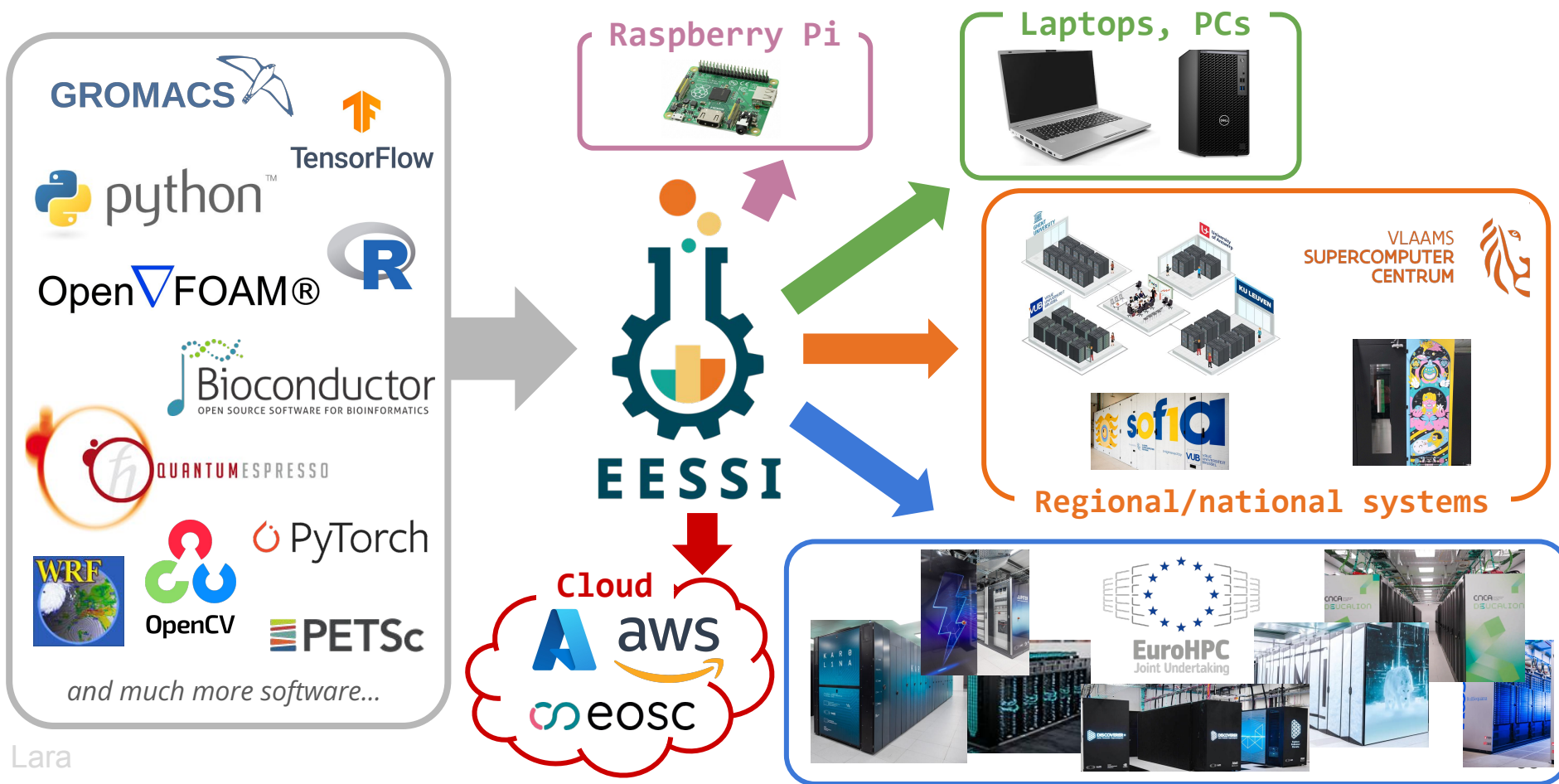
Optimized software
installations for specific
CPU microarchitectures

Intuitive user interface:
module avail,
module load, ...



Automatic selection of
best suited part of
software stack for
CPU microarchitectures

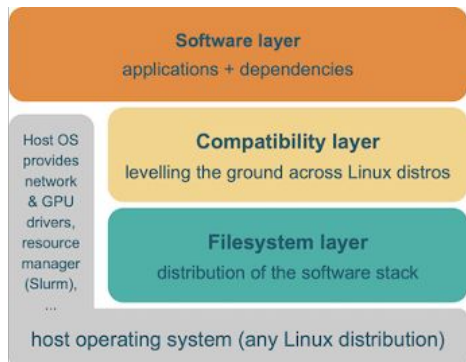
EESSI as shared software stack for HPC, cloud, and beyond



How does EESSI work?

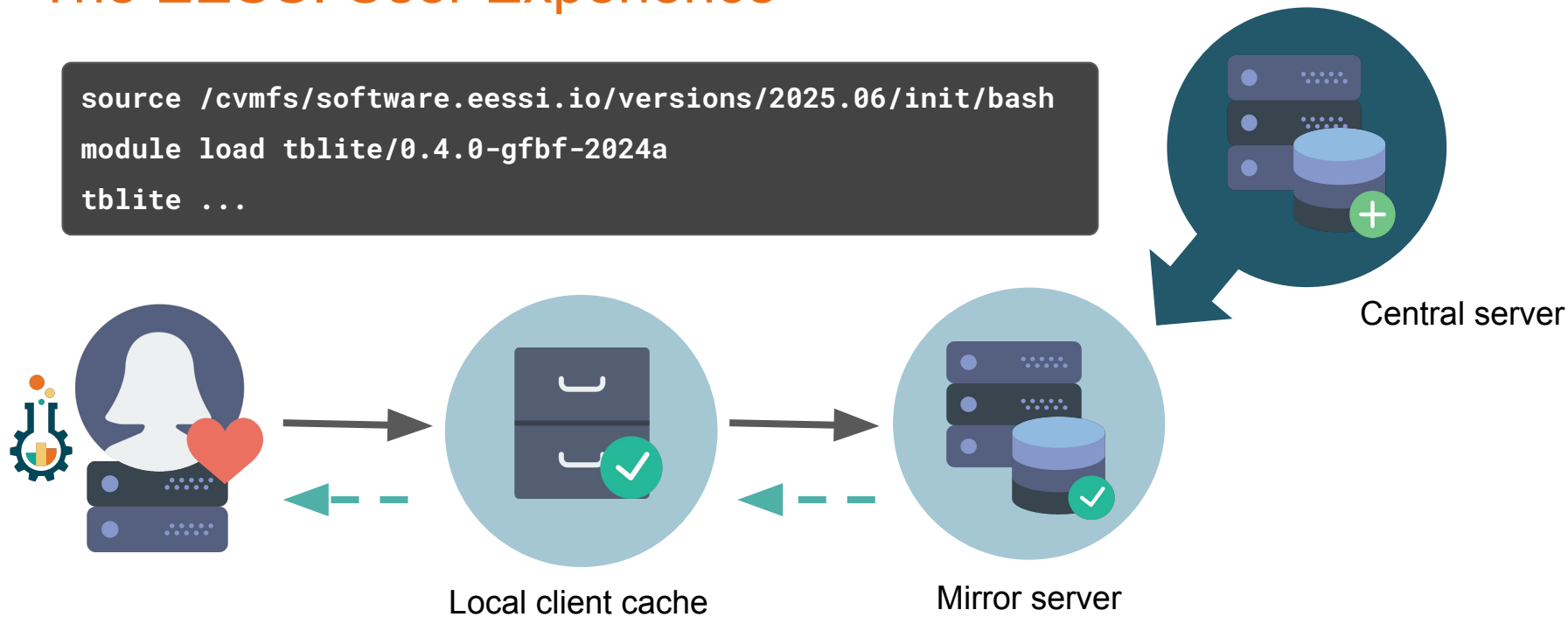


- Software installations included in EESSI are:
 - Automatically “**streamed in**” on demand (via CernVM-FS)
 - Built to be **independent of the host operating system**
“*Containers without the containing*”
 - **Optimized** for specific CPU generations + specific GPU types
- Initialization script **auto-detects** CPU + GPU of the system



The EESSI User Experience

```
source /cvmfs/software.eessi.io/versions/2025.06/init/bash  
module load tblite/0.4.0-gfbf-2024a  
tblite ...
```



EESSI provides **on-demand streaming**
of (scientific) software (like music, TV-series, ...)

Example: Using EESSI on HPC-UGent Tier-2

```
# Access via HPC-UGent web portal (https://login.hpc.ugent.be)  
  
# Start interactive shell session on donphan cluster, then run:  
  
$ source /cvmfs/software.eessi.io/versions/2025.06/init/lmod/bash  
  
Modules purged before initialising EESSI  
  
Module for EESSI/2025.06 loaded successfully  
  
EESSI has selected x86_64/intel/cascadelake as the compatible CPU target for EESSI/2025.06  
  
EESSI has selected accel/nvidia/cc80 as the compatible accelerator target for EESSI/2025.06  
  
{EESSI/2025.06} $ module avail tblite  
  
{EESSI/2025.06} $ module load tblite/0.4.0-gfbf-2024a  
  
{EESSI/2025.06} $ tblite ...
```

Example: Running tblite using EESSI at UGent



```
git clone https://github.com/EESSI/eessi-demo.git
source /cvmfs/software.eessi.io/versions/2025.06/init/bash
cd eessi-demo
ls
  Bioconductor  CitC  GROMACS  LICENSE  Magic_Castle  OpenFOAM  README.md  scripts  Tensorflow  tblite
cd tblite
./run.sh
```

Cycle	total energy	energy error	density error
1	-31.30086746450	-3.1478794E+01	3.3527172E-01
...			
18	-31.716155889120	-1.8267450E-09	1.5041592E-05

Total:		0.074 sec	
-31.71658891203904			



CernVM-FS

In our EESSI/eessi-demo repository there are a number of demo's for software in EESSI
<https://github.com/EESSI/eessi-demo>

New releases and repositories



software.eessi.io (Production Repositories)

dev.eessi.io

2023.06

Since June 2023

1,247 installations
(total: ~17,800)

679 unique software
projects

Toolchain:
foss/(2022b+23a+23b)
→ **GCC 12.4** ↓

2025.06

Current default

1,275 installations
(total: ~21,400)

640 unique software
projects

Toolchain:
foss/(2024a+25a+25b)
→ **GCC 13.3** ↑

2026.06

Baby repo! 🚀

250 installations
(total: ~4,500)

240 unique software
projects

Toolchain:
foss/(2026.1)
→ **GCC 15.2** ↑

- Developer repository
- Staging area for upcoming releases
- [More about it later :\)](#)

Overview of available software



An up-to-date overview of available software
is available in the EESSI documentation:

https://eessi.io/docs/available_software

Software available in EESSI

Overview of software available in [EESSI's production repository](https://eessi.io) `software.eessi.io`.

752 unique software projects (+ 3007 unique extensions)

name:GROMACS

GROMACS

[\(more details\)](#)

<https://www.gromacs.org>

GROMACS is a versatile package to perform molecular dynamics, i.e. simulate the Newtonian equations of motion for systems with hundreds to millions of particles. This is a CPU only build, containing both MPI and threadMPI binaries for both single and double precision. It also contains the gmxml extension for the single precision MPI build.

Available in EESSI versions: 2023.06 2025.06

Supported CPU families: AMD Intel Arm

Supported GPU families: NVIDIA

GROMACS

GROMACS is a versatile package to perform molecular dynamics, i.e. simulate the Newtonian equations of motion for systems with hundreds to millions of particles.

This is a GPU enabled build, containing both MPI and threadMPI binaries.

It also contains the gmxml extension for the single precision MPI build.

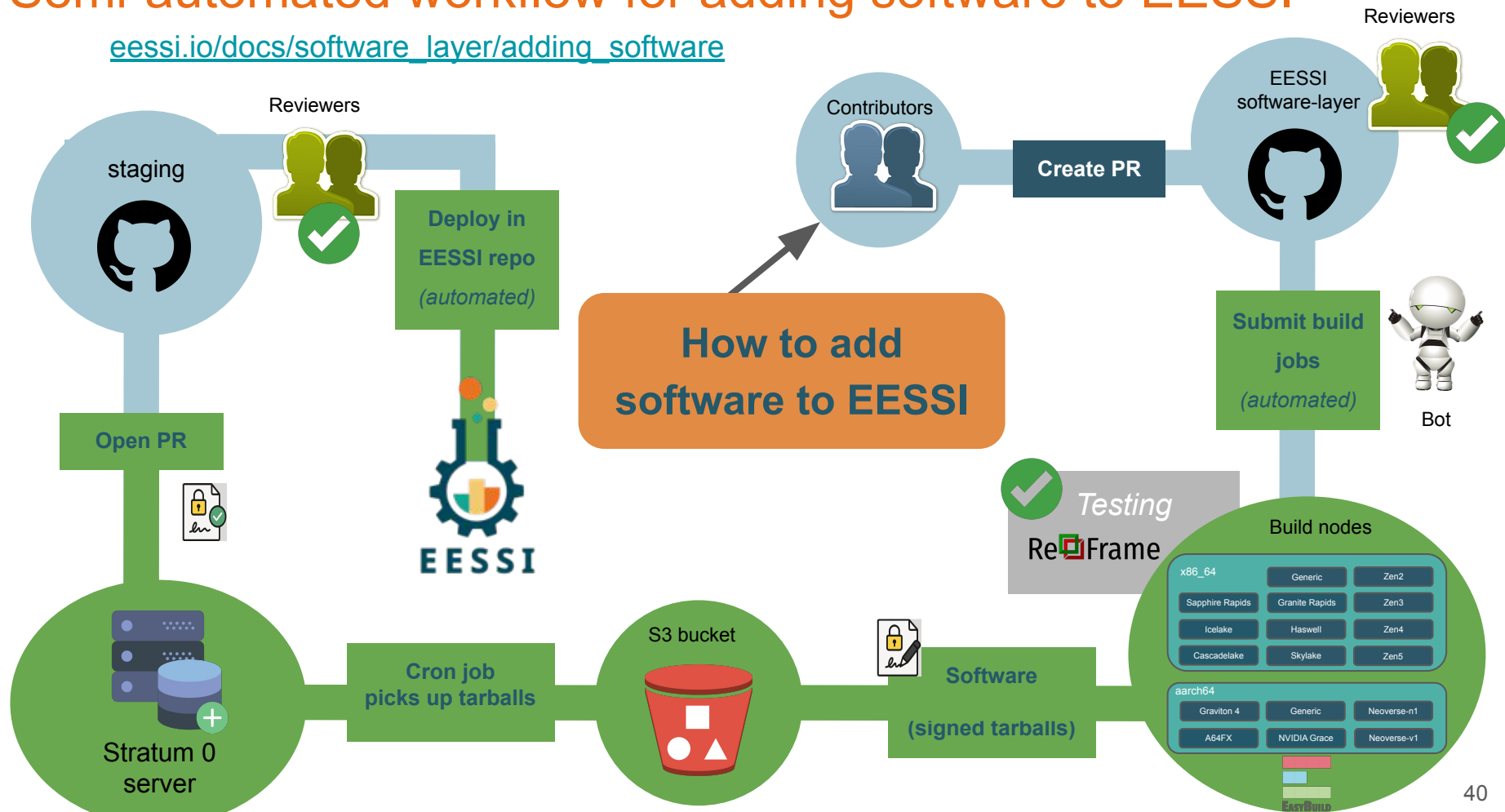
homepage: <https://www.gromacs.org>

Available installations

GROMACS version	Supported CPU targets	Supported GPU targets	EESSI version	Module
2025.4	generic: aarch64, x86_64 Arm : a64fx, neoverse_n1, neoverse_v1, nvidia/grace AMD : zen2, zen3, zen4 Intel : haswell, skylake_avx512, sapphirerapids, icelake, cascadelake	(none)	2025.06	GROMACS/2025.4-foss-2025b
2025.2	generic: aarch64, x86_64 Arm : a64fx, neoverse_n1, neoverse_v1, nvidia/grace AMD : zen2, zen3, zen4 Intel : haswell, skylake_avx512, sapphirerapids, icelake, cascadelake	(none)	2025.06	GROMACS/2025.2-foss-2025a
2024.4	generic: aarch64, x86_64 Arm : neoverse_n1,	NVIDIA : cc70, cc80, cc90	2023.06	GROMACS/2024.4-foss-2023b-

Semi-automated workflow for adding software to EESSI

eessi.io/docs/software_layer/adding_software



On which systems is EESSI available?



- On VSC systems:
 - **Readily available on all Tier-2 clusters (UGent, UA, KUL, VUB) + Tier-1 Hortense & sofia**
 - Tier-1 cloud: can deploy it yourself and have access to a full software stack in minutes
- EESSI is already available on various other European systems (and beyond)
 - EuroHPC JU systems incl. Vega, Karolina, MareNostrum 5, Deucalion, Discoverer, ...
 - Snellius @ SURF, EMBL, Univ. of Stuttgart, Sigma2 in Norway, etc.
- EESSI can be used in virtual machine in European Open Science Cloud (EOSC),
see also <https://www.eessi.io/docs/blog/2025/10/22/eosc>
- **Overview of (known) systems that have EESSI available at <https://eessi.io/docs/systems>**

Webinar series: Different aspects of EESSI

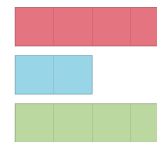
Series of presentations (May-June 2025)

<https://eessi.io/docs/training/2025/webinar-series-2025Q2>

- Introduction to EESSI
- Introduction to CernVM-FS
- Introduction to EasyBuild
- EESSI for CI/CD
- Using EESSI as the base for a system stack



CernVM-FS



EASYBUILD

Slides + recordings available



E E S S I

EUROPEAN ENVIRONMENT FOR
SCIENTIFIC SOFTWARE INSTALLATIONS

Agenda

- [09:00-09:30] *(informal welcome)*
- [09:30-09:40] **Introduction** *(Kenneth Hoste - UGent)*
- [09:40-09:50] **Central software stack + environment modules** *(Robin Verschoren - UA)*
- [09:50-10:00] **European Environment for Scientific Software Installations (EESSI)**
(Lara Peeters - UGent)
- [10:00-10:15] **Using Apptainer containers** *(Sam Moors - VUB)*
- [10:15-10:30] **Conda / Mamba / Pixi** *(Maxime Van den Bossche - KU Leuven)*
- [10:30-10:45] **Python: pip + venv** *(Steven Vandenbrande - KU Leuven)*
- [10:45-11:00] **Other language package managers: R, Julia** *(Jarne Renders - VUB)*
- [11:00-11:30] *(coffee break)*

Using Apptainer containers



Running software in a portable and reproducible way

Overview

- What is a container, why Apptainer
- When to use containers, downsides of containers
- Performance, trust/security
- Using containers: pull/inspect/shell/exec/run

Using Apptainer containers

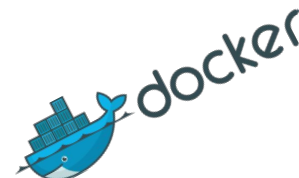


What is a container

- Software package: application + runtime environment
- Runs independent of host environment
- OS-level virtualization: isolated user space, shared host kernel
- Advantages: portable, reproducible, isolated
- Container **image** = files used to run the **container**



Using Apptainer containers



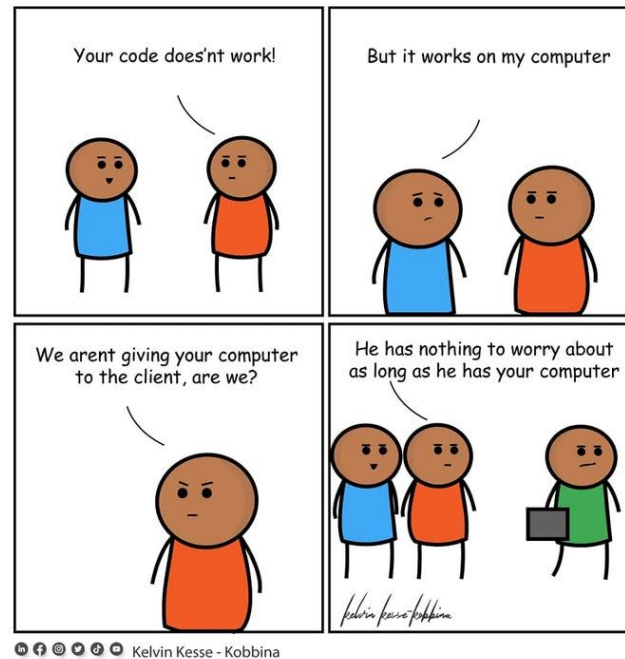
Why Apptainer (Singularity)

- Focus on integration into HPC clusters <> isolation (bind mounts, network, GPU devices)
- Doesn't require admin privileges to run/build (build with `--fakeroot` option)
- Mobility: single file (SIF format)
- Verifiable reproducibility and security (immutable images, signed images)
- Support for Docker/OCI format: `pull`, `run`, `build` from Docker/OCI images



When to use containers

- **Try out software:** quickly test a containerized app before performance-optimized installation
- **Portability:** run software built for a different Linux distribution/version
- **Reproducibility:** recreate someone else's software environment/workflow
- **Reduce inode usage:** package many files into a single image
- **Avoid build issues:**
 - **complex** or custom build procedures
 - app **too old** for available toolchains
 - many tightly constrained or conflicting **dependencies** (often conda software)



© f t i s o Kelvin Kesse - Kobbina

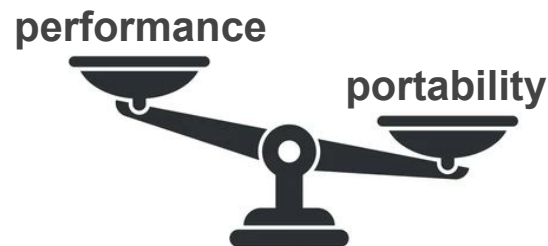
Downsides of containers

- Interoperability with software outside the container can be challenging
e.g. environment modules, Open OnDemand
- Not portable across different CPU microarchitectures
e.g. x86-64 vs AArch64
- Container images can be large (multiple GB)
 - a. Remove unnecessary files and packages after the build
 - b. Use [multi-stage builds](#)
- Containers + MPI can be tricky
Host and container MPI libraries must be compatible
- Performance considerations (see next slide)

Using Apptainer containers

Performance considerations

- Single-file image: faster startup on distributed file systems (cfr. Conda)
- Images are usually built for **generic** CPU architecture rather than architecture-**specific**: sacrifices performance for portability



Trust & security

- Do you know what's in your container?
 - how was app built/configured/patched
 - which software versions?
- Isolating the container from host environment

```
apptainer run --contain <image.sif> <command>
```

+ explicit bind mounts:

```
--bind <hostpath>[:<containerpath>[:ro]]
```

- When importing/pulling images:
always pull from a verified publisher!



Using Apptainer containers

Always pull from verified publisher!

Public container registries

- [Docker hub](#)

```
apptainer pull docker://ubuntu:24.04
```

- [NVIDIA container registry](#)

```
apptainer pull docker://nvcr.io/nvidia/pytorch:25.09-py3
```

- [AMD infinity hub](#)

```
apptainer pull  
docker://rocm/pytorch:rocm7.0.2_ubuntu24.04_py3.12_pytorch_release_2.8.0
```

- [Github container registry](#)

```
apptainer pull docker://ghcr.io/eessi/client:rocky8
```

Using Apptainer containers



Startup behavior

- Environment:
 - Scripts located at `/.singularity.d/env/` inside container
 - Executed by `apptainer run`, `exec`, `shell`
- Runscript:
 - Located at `/.singularity.d/runscript` inside container
 - Executed by `apptainer run`
 - Not executed by `apptainer exec`, `shell`

Using Apptainer containers

Preparation

```
# set environment variables (add to your ~/.bashrc)
# CACHEDIR: data that can be reused across Apptainer runs (layers, images)
# TMPDIR: temporary workspace
export APPTAINER_CACHEDIR=/tmp/$USER/apptainer_cachedir
export APPTAINER_TMPDIR=/tmp/$USER/apptainer_tmpdir
mkdir -p $APPTAINER_TMPDIR

# make sure you're in a location with enough space
cd /tmp/$USER # or $VSC_DATA, $VSC_SCRATCH
```

Using Apptainer containers



Pulling a container from container registry

```
# pull a container from Docker hub and convert to SIF image
apptainer pull ubuntu-24.04.sif docker://ubuntu:24.04
INFO:      Converting OCI blobs to SIF format
INFO:      Starting build...
INFO:      Fetching OCI image...
28.3MiB / 28.3MiB [=====] 100 % 9.5 MiB/s 0s
INFO:      Extracting OCI image...
INFO:      Inserting Apptainer configuration...
INFO:      Creating SIF file...
```

Inspecting Apptainer container

```
# inspect labels
apptainer inspect --labels tblite-0.4.0.sif
  org.label-schema.build-arch: amd64
  org.label-schema.build-date: Wednesday_30_September_2026_20:47:48_CEST
  org.label-schema.schema-version: 1.0
  org.label-schema.usage: /.singularity.d/runscript.help
  org.label-schema.usage.apptainer.runscript.help: /.singularity.d/runscript.help
  org.label-schema.usage.apptainer.version: 1.5.3-1.el9
  org.label-schema.usage.singularity.deffile.bootstrap: docker
  org.label-schema.usage.singularity.deffile.from: ubuntu:24.04
  Org.opencontainers.image.base.digest: sha256:008173...
  org.opencontainers.image.base.name: docker.io/library/ubuntu:24.04
  org.opencontainers.image.version: 24.04
```

Using Apptainer containers



Inspecting container startup

```
# inspect runscript
apptainer inspect --runscript tblite-0.4.0.sif
    #!/bin/bash
    source /opt/conda/etc/profile.d/conda.sh
    conda activate tblite
    exec "$@"

# inspect environment
apptainer inspect --environment tblite-0.4.0.sif
=== /.singularity.d/env/10-docker2singularity.sh ===
#!/bin/sh
export PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

=== /.singularity.d/env/90-environment.sh ===
#!/bin/sh
export PATH="/opt/conda/bin:$PATH"
```

Using Apptainer containers

Interactive shell inside container

```
# launch interactive shell inside container
apptainer shell tblite-0.4.0.sif

# check OS + version
Apptainer> cat /etc/lsb-release
    DISTRIB_ID=Ubuntu
    DISTRIB_RELEASE=24.04
    DISTRIB_CODENAME=noble
    DISTRIB_DESCRIPTION="Ubuntu 24.04.5 LTS"

# current working dir is the same inside and outside the container
Apptainer> pwd
    /tmp/vsc10009

# check mount points
Apptainer> findmnt -no TARGET
```

Using Apptainer containers

Running commands inside container (1/2)

```
# we can run any bash command
apptainer run tblite-0.4.0.sif bash -c 'echo $APPTAINER_NAME'
tblite-0.4.0.sif

# check tblite installation
apptainer run tblite-0.4.0.sif tblite --version
tblite version 0.4.0
apptainer run tblite-0.4.0.sif python -c 'import tblite; print(tblite.__file__)'
/opt/conda/envs/tblite/lib/python3.13/site-packages/tblite/__init__.py
```

Using Apptainer containers

Running commands inside container (2/2)

```
# exec-ing
apptainer exec tblite-0.4.0.sif tblite --version
  FATAL:  "tblite": executable file not found in $PATH
apptainer exec tblite-0.4.0.sif /.singularity.d/runscript tblite --version
  tblite version 0.4.0

# running Python interactively
apptainer run tblite-0.4.0.sif python
  Python 3.13.9 | packaged by conda-forge | (main, Oct 16 2025, 10:31:39) [GCC 14.3.0] on linux
  Type "help", "copyright", "credits" or "license" for more information.
>>> from tblite.library import get_version
>>> get_version()
  (0, 4, 0)
```

Agenda

- [09:00-09:30] *(informal welcome)*
- [09:30-09:40] **Introduction** (*Kenneth Hoste - UGent*)
- [09:40-09:50] **Central software stack + environment modules** (*Robin Verschoren - UA*)
- [09:50-10:00] **European Environment for Scientific Software Installations (EESSI)**
(*Lara Peeters - UGent*)
- [10:00-10:15] **Using Apptainer containers** (*Sam Moors - VUB*)
- [10:15-10:30] **Conda / Mamba / Pixi** (*Maxime Van den Bossche - KU Leuven*)
- [10:30-10:45] **Python: pip + venv** (*Steven Vandenbrande - KU Leuven*)
- [10:45-11:00] **Other language package managers: R, Julia** (*Jarne Renders - VUB*)
- [11:00-11:30] *(coffee break)*



a package and environment manager
running on all major OSs and hardware

Overview

- Main ideas, repositories and implementations
- Creating and using a Conda environment with Miniforge
- Other implementations in a nutshell (Miniconda, Micromamba, Pixi)
- Possible drawbacks and pitfalls



a package and environment manager
running on all major OSs and hardware

Main ideas

- Programming language-agnostic
- Support for multiple independent environments
- Preference for precompiled Conda packages
- Supplying all dependencies (except e.g. glibc)
- No need for root privileges



a package and environment manager
running on all major OSs and hardware

Main ideas

- Supported platforms (note: a given package may not be available for all of them):

linux-32

osx-64

win-32

linux-64

osx-arm64

win-64

linux-aarch64

linux-ppc64le

linux-s390x



a package and environment manager
running on all major OSs and hardware

Common Conda package repositories (aka ‘channels’)

- *Default* channels like ‘*main*’ and ‘*r*’ (note: Anaconda Terms of Service!)
- *Community* channels like ‘*conda-forge*’ and ‘*bioconda*’
- The ‘[*intel*](#)’ channel can be worth looking at for Intel CPUs



a package and environment manager
running on all major OSs and hardware

Different implementations

- [Anaconda Distribution](#)
- [Miniconda](#)
- [Miniforge](#) (← the one we tend to recommend)
- [Micromamba](#)
- [Pixi](#)

Example: creating and using a Conda environment with Miniforge



```
module load Miniforge3/25.3.0-3
source $(conda info --base)/etc/profile.d/conda.sh

# If you haven't done it yet, ensure Conda/Mamba is not using your $VSC_HOME, e.g.
conda config --append envs_dirs $VSC_DATA/conda_envs
conda config --append pkgs_dirs $VSC_SCRATCH/conda_pkgs
conda config --set env_prompt '({name}) '

# Verifying the settings
cat ~/.condarc
channels:
  - conda-forge          <- Note that by default only conda-forge is registered
envs_dirs:
  - /data/leuven/.../conda_envs
pkgs_dirs:
  - /scratch/leuven/.../conda_pkgs
```

Example: creating and using a Conda environment with Miniforge



```
# Create and activate the new environment
conda create -n myfirstenv
Channels:
  - conda-forge
Platform: linux-64
...
  environment location: /data/leuven/.../conda_envs/myfirstenv
Proceed ([y]/n)? y

conda activate myfirstenv

conda env list
# conda environments:
#
base                /apps/leuven/.../software/Miniforge3/25.3.0-3
myfirstenv          /data/leuven/.../conda_envs/myfirstenv
```

Example: creating and using a Conda environment with Miniforge



```
# Search for available tblite-python packages
```

```
conda search tblite-python
```

build strings

```
Loading channels: done
```

# Name	Version	Build	Channel
...			
tblite-python	0.4.0	py310h3abc789_0	conda-forge
tblite-python	0.4.0	py310h3abc789_1	conda-forge
tblite-python	0.4.0	py311h7277607_0	conda-forge
tblite-python	0.4.0	py311h7277607_1	conda-forge
tblite-python	0.4.0	py312h4e599bb_0	conda-forge
tblite-python	0.4.0	py312h4e599bb_1	conda-forge
tblite-python	0.4.0	py313h26530d8_0	conda-forge
tblite-python	0.4.0	py313h26530d8_1	conda-forge
tblite-python	0.4.0	py39h1f626a3_0	conda-forge
tblite-python	0.4.0	py39h1f626a3_1	conda-forge

Example: creating and using a Conda environment with Miniforge



```
# Installing the tblite executable and Python interface
conda install tblite-python==0.4.0
...
environment location: /data/leuven/.../conda_envs/myfirstenv
...
The following NEW packages will be INSTALLED:
...
dftd4                conda-forge/linux-64::dftd4-3.7.0-hd37a5e2_4
libblas              conda-forge/linux-64::libblas-3.9.0-37_h4a7cf45_openblas
libgomp              conda-forge/linux-64::libgomp-15.2.0-h767d61c_7
liblapack            conda-forge/linux-64::liblapack-3.9.0-37_h47877c9_openblas
libopenblas          conda-forge/linux-64::libopenblas-0.3.30-threads_h94d23a6_2
python               conda-forge/linux-64::python-3.13.9-hc97d973_101_cp313
numpy                conda-forge/linux-64::numpy-2.3.4-py313hf6604e3_0
tblite               conda-forge/linux-64::tblite-0.4.0-hd37a5e2_2
tblite-python        conda-forge/linux-64::tblite-python-0.4.0-py313h26530d8_1
...
Proceed ([y]/n)? y
```

Example: creating and using a Conda environment with Miniforge



```
# Some checks with the executable and Python bindings
tblite --version
tblite version 0.4.0

which tblite
/data/leuven/.../conda_envs/myfirstenv/bin/tblite

which python
/data/leuven/.../conda_envs/myfirstenv/bin/python

python -c 'from tblite.library import get_version; print(get_version())'
(0, 4, 0)
```

Example: creating and using a Conda environment with Miniforge



```
# Switch from OpenBLAS to MKL (see also the conda-forge documentation)
```

```
conda install "libblas=*mkl"
```

```
...
```

```
The following NEW packages will be INSTALLED:
```

```
...
```

```
mkl                conda-forge/linux-64::mkl-2024.2.2-ha770c72_17
```

```
The following packages will be REMOVED:
```

```
libopenblas-0.3.30-pthreads_h94d23a6_2
```

```
The following packages will be DOWNGRADED:
```

_openmp_mutex	4.5-2_gnu --> 4.5-5_kmp_llvm
libblas	3.9.0-36_h4a7cf45_openblas --> 3.9.0-36_h5875eb1_mkl
libcblas	3.9.0-37_h0358290_openblas --> 3.9.0-37_hfef963f_mkl
liblapack	3.9.0-36_h47877c9_openblas --> 3.9.0-36_h5e43f62_mkl

```
ls -l $VSC_DATA/conda_envs/myfirstenv/lib/liblapack.so.3
```

```
... /data/leuven/.../conda_envs/myfirstenv/lib/liblapack.so.3 -> libmkl_rt.so.2
```

Example: creating and using a Conda environment with Miniforge



```
# Deactivate the environment  
conda deactivate  
  
# For removing the environment later on  
conda env remove -n myfirstenv  
  
# Cleaning package caches (which can grow to tens of GBs)  
conda clean --all
```

Other implementations: see the Exercises

miniCONDA®

Miniconda: most similar to Miniforge but with

- *defaults* channel enabled by default
- more disk usage (if using a local installer)

Micromamba:

- single executable (`alias conda=micromamba`)
- potentially faster than e.g. miniforge
- need to manually configure `envs_dirs` and `pkgs_dirs` in your `~/.condarc`

Pixi:

- project-centric rather than environment-centric
- more emphasis on speed
- more reproducible environments via *lockfiles*

Possible drawbacks and pitfalls



- Default configs can lead to Conda caches filling up your `$VSC_HOME`
- Conda environments cannot be used together with most centrally installed modules
- Conda packages may lack [microarchitectural optimizations](#)
If in doubt, it may help to look at the package's [feedstock](#) (conda-forge specific)
- Conda packages for MPI: performance may be low, multi-node runs may hang
- Storage demands (volume and inodes) are often higher than for other approaches
[Containerizing](#) can help with the inodes
- 'Base' environments (if any) [need to be reserved](#) for Conda-related packages

Possible drawbacks and pitfalls

- *'default'* channels like *'main'* and *'r'* are subject to [Anaconda's Terms of Service](#)
If academic user → license permits to freely use packages from these channels
When leaving university (or from industry to begin with) → requires paid license
- Reproducing environments on the same platform should be straightforward:

```
conda env export > env.yaml  
conda env create -n mynewenv -f env.yaml
```

For different platforms (and possibly microarchitectures), you may instead need

```
conda env export --no-builds
```

This can also be useful:

```
conda env export --from-history
```

Agenda

- [09:00-09:30] *(informal welcome)*
- [09:30-09:40] **Introduction** (*Kenneth Hoste - UGent*)
- [09:40-09:50] **Central software stack + environment modules** (*Robin Verschoren - UA*)
- [09:50-10:00] **European Environment for Scientific Software Installations (EESSI)**
(*Lara Peeters - UGent*)
- [10:00-10:15] **Using Apptainer containers** (*Sam Moors - VUB*)
- [10:15-10:30] **Conda / Mamba / Pixi** (*Maxime Van den Bossche - KU Leuven*)
- [10:30-10:45] **Python: pip + venv** (*Steven Vandenbrande - KU Leuven*)
- [10:45-11:00] **Other language package managers: R, Julia** (*Jarne Renders - VUB*)
- [11:00-11:30] *(coffee break)*

Python's pip+venv



pip: pip install packages - popular Python package manager

venv: provides lightweight virtual environments to cleanly separate sets of Python packages

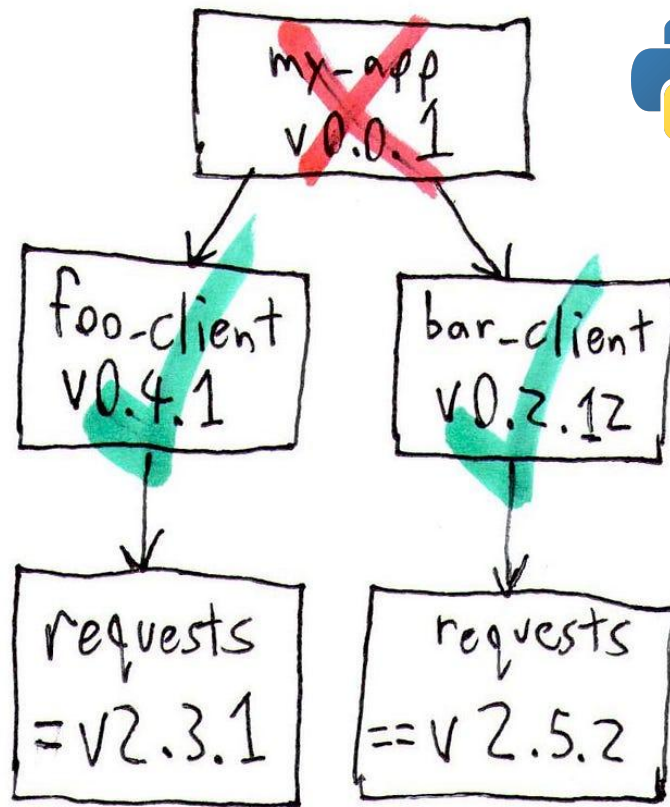
Overview

- **Motivation and context**
- How to use pip+venv
- How to use vsc-venv
- Alternatives to pip+venv

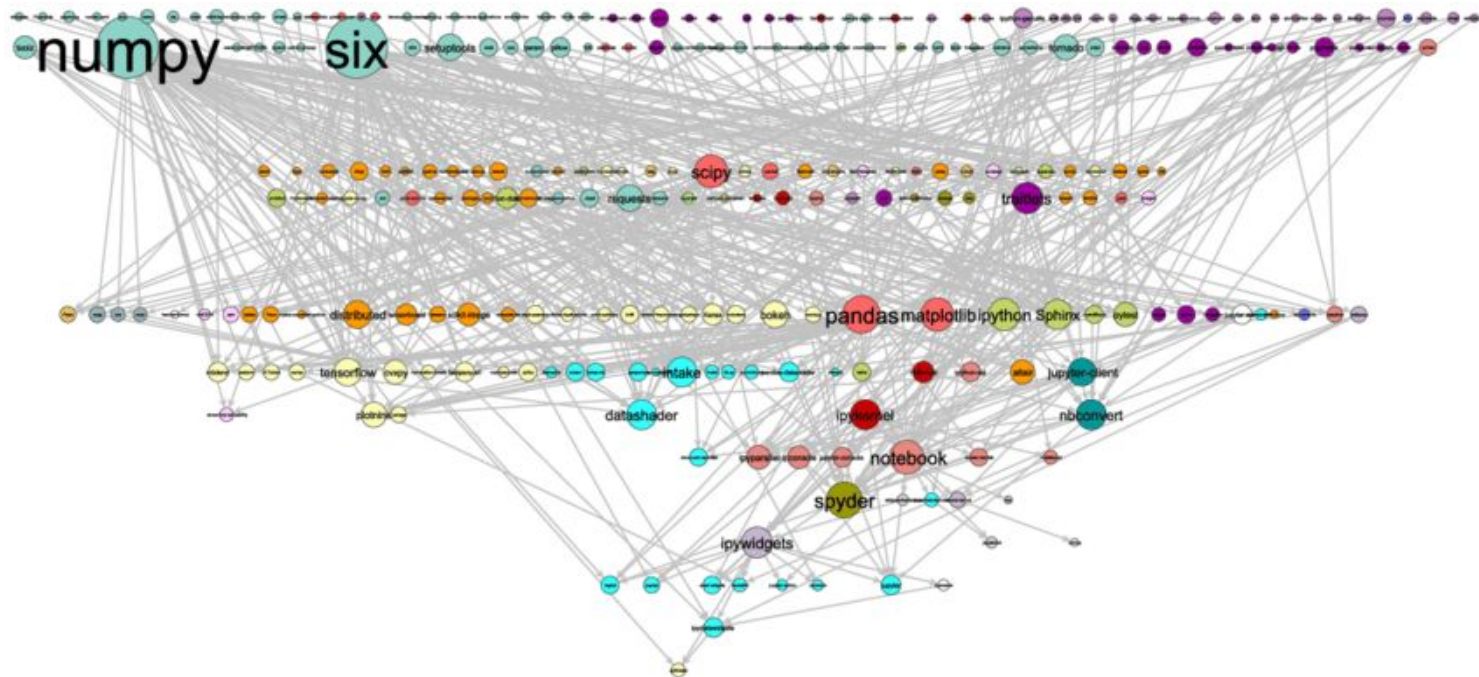
pip for resolving dependencies



- To install a Python package, you typically need to install additional packages as dependencies
- Transitive dependencies quickly lead to a complicated dependency graph
- Use a package manager (like pip) to automatically resolve dependencies



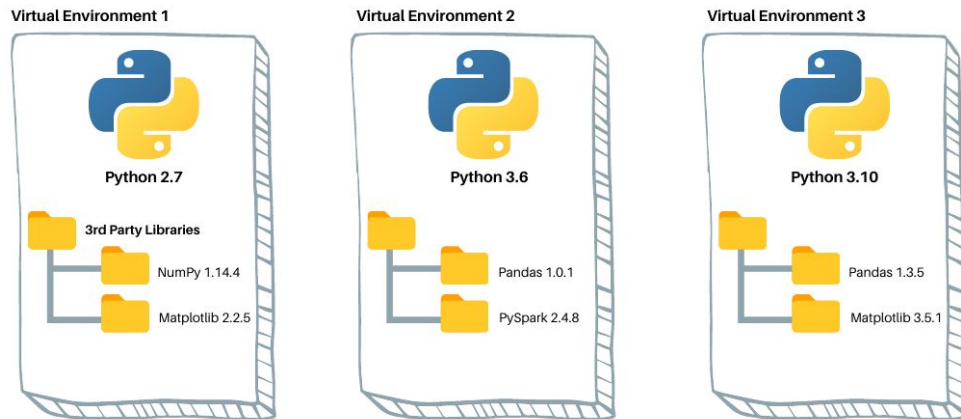
pip for resolving dependencies



venv for isolating environments



- Sometimes you need packages that are incompatible:
 - use virtual environments to keep them separated
 - lightweight because interpreter lives outside of environment
- Also helps with reproducing environment on difference machines
- Can be combined with packages outside of environment



Comparison with other approaches



	Conda/Mamba/Pixi	Centrally installed modules	pip+venv
Easy to use	+	++ (burden is on support team)	++
Control as a user	++	-	+
Isolation	+	+-	-
Number of files	Very high (unless containerized) [11K for tblite-python]	Lower (and less relevant)	Lower [3K for tblite]
Disk space	High [372MB for tblite-python]	Lower (and less relevant)	Lower [111MB for tblite]

Note that this table only concerns Python!

Python's pip+venv



pip: pip install packages - popular Python package manager

venv: provides lightweight virtual environments to cleanly separate sets of Python packages

Overview

- Motivation and context
- **How to use pip+venv**
- How to use vsc-venv
- Alternatives to pip+venv

Demonstration of basic pip+venv



Create a **venv** in `$VSC_DATA/my_first_venv` (only once per venv)

```
cd $VSC_DATA
python -m venv my_first_venv
stat my_first_venv
  File: my_first_venv/
  Size: 4096             Blocks: 8           IO Block: 32768   directory
  ...
```

Activate the **venv** (everytime you want to use the venv)

```
source ${VSC_DATA}/my_first_venv/bin/activate
```

Python interpreter comes from **venv**, symlinks to “*original*” (demonstration only)

```
which python
  /data/leuven/337/vsc33716/my_first_venv/bin/python
ls -lh $(which python)
lrwxrwxrwx 1 vsc33716 vsc33716 15 Oct 10 11:43
  /data/leuven/337/vsc33716/my_first_venv/bin/python -> /usr/bin/python
```

Demonstration of basic pip+venv



Create a **venv** in `$VSC_DATA/my_first_venv` (only once per venv)

```
cd $VSC_DATA
python -m venv my_first_venv
stat my_first_venv
  File: my_first_venv/
  Size: 4096             Blocks: 8           IO Block: 32768   directory
  ...
```

Activate the **venv** (everytime you want to use the venv)

```
source ${VSC_DATA}/my_first_venv/bin/activate
```

Python interpreter comes from **venv**, symlinks to “*original*” (demonstration only)

```
which python
/data/leuven/337/vsc33716/my_first_venv/bin/python
ls -lh $(which python)
lrwxrwxrwx 1 vsc33716 vsc33716 15 Oct 10 11:43
/data/leuven/337/vsc33716/my_first_venv/bin/python -> /usr/bin/python
```

Hint: check version of Python interpreter.
System Python might be old, loading a Python
module can be useful (see later)

Demonstration of basic pip+venv



Good practice (most of the time): **update pip**

```
python -m pip install --upgrade pip
...
Successfully installed pip-25.2
```

Install specific version of package, automatically installs required dependencies

```
python -m pip install tblite==0.4.0
...
Using cached tblite-0.4.0-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.7 MB)
...
Successfully installed cffi-2.0.0 numpy-2.0.2 pycparser-2.23 tblite-0.4.0
```

Quick check of the installation

```
python -c "import tblite; print(tblite.__file__)"
/data/leuven/337/vsc33716/my_first_venv/lib64/python3.9/site-packages/tblite/__init__.py
```

Deactivate environment with

deactivate

Demonstration of basic pip+venv: remarks



By default `pip install` will pull in binary versions of libraries when available, use `--no-binary` to override

The `tblite` executable is not include in the pip Python package, only the library and Python bindings are provided

Beyond the basics: reproducing a venv



List installed packages in a reproducible format

```
python -m pip freeze > requirements.txt
cat requirements.txt
    cffi==2.0.0
    numpy==2.0.2
    pycparser==2.23
    tblite==0.4.0
```

Recreate the **venv**

```
python -m venv $VSC_DATA/my_second_venv
source $VSC_DATA/my_second_venv/bin/activate
python -m pip install -r requirements.txt
```

Beyond the basics: reproducing a venv



List installed packages in a reproducible format

```
python -m pip freeze > requirements.txt
cat requirements.txt
    cffi==2.0.0
    numpy==2.0.2
    pycparser==2.23
    tblite==0.4.0
```

Recreate the **venv**

```
python -m venv $VSC_DATA/my_second_venv
source $VSC_DATA/my_second_venv/bin/activate
python -m pip install -r requirements.txt
```

Only guaranteed to work if venv is based on same Python version as original!

Beyond the basics



PyPI (<https://pypi.org/>) is the most common repository used by pip, but you can specify others:

```
python -m pip install --index-url http://custompypi/yourpackage
python -m pip install git+https://github.com/pypa/sampleproject.git@main
```

Maintaining packages:

```
python -m pip install --upgrade tblite
python -m pip uninstall tblite
```

Python's pip+venv



pip: pip install packages - popular Python package manager

venv: provides lightweight virtual environments to cleanly separate sets of Python packages

Overview

- Motivation and context
- How to use pip+venv
- **How to use vsc-venv**
- Alternatives to pip+venv



pip+venv can be perfectly combined with centrally installed Python (package) modules:

- Simply load modules before creating and activating the venv
- pip will take packages from modules into account for resolving dependencies

Pros:

- Reduces storage requirements for venv
- Ensures packages with *compiled* code are suited for hardware

Cons:

- You need to remember which modules to load before activating venv
- venv may be specific to CPU (if it contains compiled code or has hardcoded links to compiled code)

⇒ **solved by vsc-venv**

(https://docs.vscentrum.be/compute/software/installing_software/python_package_management.html#the-vsc-venv-utility)



vsc-venv encapsulates management of venvs with pip

- support for loading specified modules
- venv is specific for os/microarchitecture (see earlier talks)

```
cat tblite.txt  
tblite==0.4.0
```

```
cat tblite_modules.txt  
SciPy-bundle/2024.05-gfbf-2024a
```

```
module load vsc-venv  
source vsc-venv --activate -r tblite.txt -m tblite_modules.txt  
python -c "import tblite; print(tblite.__file__)"  
${VSC_DATA}/venvs/venv-rocky9-skylake/lib/python3.12/site-packages/tblite/__init__.py
```

Make sure package version is compatible
with packages from modules!



vsc-venv encapsulates management of venvs with pip

- support for loading specified modules
- venv is specific for os/microarchitecture (see earlier talks)

```
# To deactivate an environment activated with vsc-venv  
source vsc-venv --deactivate
```

Python's pip+venv



pip: pip install packages - popular Python package manager

venv: provides lightweight virtual environments to cleanly separate sets of Python packages

Overview

- Motivation and context
- How to use pip+venv
- How to use vsc-venv
- **Alternatives to pip+venv**

Alternatives to pip+venv



uv (<https://docs.astral.sh/uv/>)

- **An extremely fast Python package and project manager, written in Rust.**
- **Provides a drop-in replacement for common pip and venv operations**

poetry (<https://python-poetry.org/>)

- Python packaging and dependency management made easy
- Tool for dependency management and packaging in Python

virtualenv (<https://virtualenv.pypa.io/>)

- more features and faster than venv (which is a subset of virtualenv)

Mixed approaches

- For example pip in conda or pixi env
- In principle possible, but usually not recommended

Agenda

- [09:00-09:30] *(informal welcome)*
- [09:30-09:40] **Introduction** (*Kenneth Hoste - UGent*)
- [09:40-09:50] **Central software stack + environment modules** (*Robin Verschoren - UA*)
- [09:50-10:00] **European Environment for Scientific Software Installations (EESSI)**
(*Lara Peeters - UGent*)
- [10:00-10:15] **Using Apptainer containers** (*Sam Moors - VUB*)
- [10:15-10:30] **Conda / Mamba / Pixi** (*Maxime Van den Bossche - KU Leuven*)
- [10:30-10:45] **Python: pip + venv** (*Steven Vandenbrande - KU Leuven*)
- [10:45-11:00] **Other language package managers: R, Julia** (*Jarne Renders - VUB*)
- [11:00-11:30] *(coffee break)*

Language package managers

Language	Type	Domain	Package Manager*	Package Repository*
Java	High-level Just-in-time compilation	Bio-Sciences Social Sciences	Maven	Maven
JavaScript (JS)	High-level Just-in-time compilation	Graphics and Visualization	npm	npm
Julia	High-level Just-in-time compilation	Machine Learning Data Science	Pkg.jl (<i>built-in</i>)	<i>decentralized</i>
Perl	High-level Interpreted	Bio-Sciences Social Sciences	cpanm	CPAN
Python	High-level Interpreted and JIT	General	pip	PyPI
R	High-level Interpreted and JIT	Data Science Bio-Sciences Social Sciences	R (<i>built-in</i>)	CRAN

* most common option but others may exists

Language package managers: R



R is a language and environment for statistical computing and graphics.

How to find available R packages in modules?

- The module **R** is a basic installation with a minimum set of R packages just to allow development in R
- The module **R-bundle-CRAN** provides an extensive collection of R packages from the CRAN repository. Packages cover all scientific domains.
- The module **R-bundle-Bioconductor** provides an extensive collection of R packages from the Bioconductor repository. Packages cover the bio sciences.
- Some R packages can be provided with specific modules. This happens for R packages that need a large collections of libraries (e.g. INLA, Seurat) or for R packages that need specific non-R dependencies (e.g. RPostgreSQL)

Language package managers: R



R packages usually are extensions of bundle modules

```
module spider ggplot2
```

```
-----  
ggplot2:  
-----
```

```
  Versions:
```

```
    ggplot2/3.4.4 (E)
```

```
    ggplot2/3.5.1 (E)
```

```
Names marked by a trailing (E) are extensions provided by another module.  
-----
```

```
  For detailed information about a specific "ggplot2" package (including how to load the modules) use the module's  
  full name.
```

```
  Note that names that have a trailing (E) are extensions provided by other modules.
```

```
  For example:
```

```
    $ module spider ggplot2/3.5.1  
-----
```

```
module spider ggplot2/3.5.1
```

```
-----  
ggplot2: ggplot2/3.5.1 (E)  
-----
```

```
  This extension is provided by the following modules. To access the extension you must load one of the following  
  modules. Note that any module names in parentheses show the module location in the software hierarchy.
```

```
R-bundle-CRAN/2024.11-foss-2024a
```

No analog to *tblite* in R



How to install your own R packages?

- simple case where pkg_name will be downloaded from a remote repo (usually CRAN)

```
R> install.packages("ggplot2")
```

- install a specific version

```
R> library(remotes)
R> install_version("ggplot2", version = "3.5.1")
```

- from a specific repository

```
R> install.packages("ggplot2",
  repos = "http://R-Forge.R-project.org",
  dependencies = TRUE)
```

- from a repository in GitHub

```
R> library(devtools)
R> install_github("tidyverse/ggplot2")
```

! Check versions!

- from a local file

```
R> install.packages("/path/to/ggplot2-3.5.1.zip",
  repos = NULL, type = "source")
```



Location of installed R packages

R will check multiple directories for packages/libraries and will try to install new packages in the top directory reported by the function `libPaths()`:

```
R> .libPaths()  
[1] "/user/brussel/100/vsc10000/R/x86_64-pc-linux-gnu-library/4.4"  
[2] "/apps/brussel/RL8/skylake-ib/software/R-bundle-CRAN/2024.11-foss-2024a"  
[3] "/apps/brussel/RL8/skylake-ib/software/R/4.4.2-gfbf-2024a/lib64/R/library"
```

! All these libraries are specific to the active version of R in use

1. Your personal R library. Can be changed with environment variable `$R_LIBS_USER`. By default located under `$R_HOME/library`.
2. Site libraries added by module files. Read-only.
3. Main R library of the active R interpreter. Read-only.



Drawbacks of R package manager

- **No reproducibility:** it is hard to manage the versions of packages installed in R. Any future installation might upgrade some package and break your environment.
- **No isolation:** Installed packages are tied to a single version of R, but there is **no way to easily activate/deactivate them**. So whenever that specific version of R is used, those packages will be active in your environment.
- **No portability:** Installed packages with R might involve lower-level compiled code. The resulting binaries might **break on other systems** than the one used for the installation.



RStudio Projects

RStudio Projects provide a self-contained, organized environment in R. Each project has its own working directory, workspace and history which helps to avoid conflicts between different projects.

The **vsc-Rproject** tool helps you create RStudio Projects, and allows to compile extensions in a more portable way and simplifies setting up your R environment (including selecting the correct R module).

- Repository: <https://github.com/hpcleuven/vsc-Rproject>
- Documentation: https://docs.vscentrum.be/compute/software/vsc_rproject.html

Language package managers: R



Using vsc-Rproject

```
# 1. Load vsc-Rproject
module load vsc-Rproject

# 2. Set modules
echo "R/4.4.2-gfbf-2024a" > $VSC_DATA/modules.txt
echo "R-bundle-Bioconductor/3.20-foss-2024a-R-4.4.2" >>
$VSC_DATA/modules.txt

# 3. Create project
vsc-rproject create MyProject --modules="$VSC_DATA/modules.txt"

# 4. Activate Project
vsc-rproject activate MyProject
```

R projects made with **vsc-Rproject**
are located in `$VSC_DATA` by default



Using vsc-Rproject

```
# 5. Jump into the R Project
cd $VSC_RPROJECT
R
R version 4.4.2 (2024-10-31) -- "Pile of Leaves"
[...]
📁 R Project Environment Activated
◆ Project      : MyProject
◆ Working Dir  : /path/to/Rprojects/MyProject
◆ Library Paths :
- /path/to/Rprojects/MyProject/library/RL8/R
- /path/to/software/R-bundle-Bioconductor/3.20-foss-2024a-R-4.4.2
- /path/to/software/arrow-R/17.0.0.1-foss-2024a-R-4.4.2
- /path/to/software/R-bundle-CRAN/2024.11-foss-2024a
- /path/to/software/R/4.4.2-gfbbf-2024a/lib64/R/library
```

Installing extra packages in vsc-Rproject



New installations in the vsc-Rproject will take into account loaded modules

Installing extra packages in your project will use active extensions from CRAN and/or Bioconductor. Only adding missing dependencies or newer versions of existing packages if required.

vsc-Rproject environment is configured to be compatible and portable

Installations of R packages frequently involve low-level compilation and linking to external libraries. The environment of vsc-Rprojects guarantees that any compiled code will use the same compilers and libraries of its base R and be portable across different hardware architectures.

Language package managers: Julia



Julia is a dynamic, high-performance programming language that is used to perform operations in scientific computing.

How to find available Julia packages in modules?

- The module Julia provides a standard installation of Julia without any extra packages.
- Some specific Julia packages can be provided with their own modules (e.g. IJulia)

How to install your own Julia packages?

The best approach to install Julia packages is by making Julia environments. These are self-contained collections of packages that can be activated on-the-fly for each different project and easily shared with colleagues.

```
# 1. Load module of Julia
$ module load Julia/1.11.6-linux-x86_64

# 2. Make a new Julia environment
$ mkdir MyProject
$ julia
julia> using Pkg
julia> Pkg.activate("MyProject")
    Activating new project at
    `~/home/user/julia/MyProject`

# 3. Add software to your Julia environment
julia> Pkg.add("TightBindingToolkit")

# 4. Start using the new software
julia> using TightBindingToolkit
```

Language package managers: Julia



Location of installations with Julia

The packages installed do not go into the environment folder, but into your Julia depot in your home directory (`~/.julia`). Move your personal julia depot to scratch storage to avoid filling up your home directory

```
mkdir -p ~/.julia  
mv -i ~/.julia $VSC_SCRATCH/julia  
ln -s $VSC_SCRATCH/julia ~/.julia
```

Language package managers: Julia



How to install your own Julia packages on top of existing modules?

```
module load Circuitscape/5.12.3-Julia-1.9.2
base_project=$(julia -E 'Base.load_path() [end]')
cp -r "$(dirname ${base_project:1:-1})" myNewEnv
julia -e 'using Pkg; Pkg.activate("myNewEnv"); Pkg.status()'
  Activating project at /path/to/myNewEnv
  [...]
  [2b7a1792] Circuitscape v5.12.3 /apps/.../software/Circuitscape/
5.12.3-Julia-1.9.2/packages/Circuitscape

julia -e 'using Pkg; Pkg.activate("myNewEnv"); Pkg.add("CSV")'
  Activating project at /path/to/myNewEnv
  Resolving package versions...
  Updating /path/to/myNewEnv/Project.toml
  [336ed68f] + CSV v0.10.13
  Updating /path/to/myNewEnv/Manifest.toml
  [336ed68f] + CSV v0.10.13
  [...]
Precompiling project...
  7 dependencies successfully precompiled in 53 seconds. 100 already precompiled.
```

To use all Julia packages provided by some software module as the base for a new custom project environment in Julia, you have to **create the new project as a copy of the environment of the loaded software module.**

Coffee break!



11:00 - 11:30

Agenda

- [11:30-11:45] **Software compilation** (*Cintia Willemyns - VUB*)
- [11:45-12:00] **Building Apptainer containers** (*Sam Moors - VUB*)
- [12:00-12:15] **EasyBuild + Spack** (*Kenneth Hoste - UGent*)
- [12:15-12:30] **When should I use which tool + Conclusions** (*Kenneth Hoste - UGent*)
- [12:30-13:30] (*lunch*)
- [13:30-15:00] **Hands-on: choose your own adventure!**
- [15:00-15:30] (*coffee break*)
- [15:00-16:30] **Hands-on (continued)**
- [16:30-17:00] **Show & tell** (what did you work on?)

- Software written in low(er)-level languages must be **compiled into machine code** before it can run on a computer.

Language	Compiler	Package Manager*
Fortran	- GNU (gfortran) - Intel (ifort/ix) - LLVM (flang) ...	fpm
C	- GNU (gcc) - Intel (icc/icx) - LLVM (clang) ...	conan
C++	- GNU (g++) - Intel (icpc/icpx) - LLVM (clang) ...	conan
Rust	- Official (rustc)	cargo
Go	- Official (go build)	<i>built-in</i>
Zig	- Official (zig build-exe)	<i>built-in decentralized</i>

* (Most common options, others may exist)

Software compilation: Before you start

- Compile on **compute nodes**

Depending on the size of the project, compilations can be quite **compute intensive**.

If you need **architecture-specific optimizations**, build on nodes matching the target hardware.

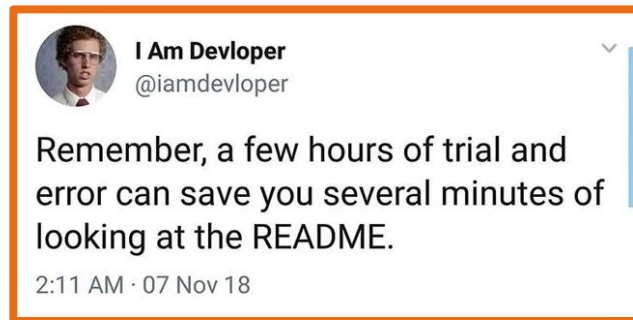
- **Read documentation** (if available)

Most projects have **documentation** that will tell you how to build their software, this can save time and prevent errors.

- **Use a `buildenv` module** → ready-to-use **development environment** with a predefined set of compilers and build tools matching common toolchains (e.g. *FOSS*, *Intel*)

- Loading the module loads the **compiler** and any **math** and/or **MPI** libraries
- Defines **compiler and linker flags** for optimal performance (`CFLAGS`, `FFLAGS`, `CXXFLAGS`, `LDFLAGS`, ...)
- Defines **search paths** so build systems find the correct headers and libraries (`CPATH`, `LIBRARY_PATH`, ...)
- **Additional modules** such as `CMake`, `Autotools`, `pkgconf`, `Ninja`, or `Meson` can be loaded on top if needed.

This ensures a controlled and reproducible build setup





Rust is a general-purpose programming language. Its emphasis is on performance, type safety, concurrency, and memory safety.

How to install your own Rust packages?

1. Load a Rust module

```
module load Rust/1.91.1-GCCcore-14.2.0
```

2. Get your source code

```
git clone https://github.com/svenstaro/genact.git
Cloning into 'genact'...
remote: Enumerating objects: 4425, done.
remote: Counting objects: 100% (720/720), done.
remote: Compressing objects: 100% (88/88), done.
[...]
cd genact
git switch --detach v1.5.1
HEAD is now at 072b491 chore: Release genact version 1.5.1
```

How to install your own Rust packages?

3. Check the contents of the directory with the source code.
Rust packages buildable with cargo have a **Cargo.toml** file



```
ls
Cargo.lock      genact          README.md
Cargo.toml      gifs           release.toml
CHANGELOG.md    index.html     src
Containerfile   LICENSE        static
CONTRIBUTING.md Makefile       Trunk.toml
data           PULL_REQUEST_TEMPLATE.md
```

- **Cargo.toml** → package metadata: name, version, authors, features, dependency requirements, ...
- **Cargo.lock** → exact versions of dependencies, makes builds reproducible.

Recommended: Generate **Cargo.lock** if missing

```
cargo generate-lockfile
Updating crates.io index
Locking 228 packages to latest Rust 1.91.1
compatible versions
Adding clap_mangen v0.2.33 (available: v0.3.3)
Adding fake v4.4.0 (available: v5.1.0)
Adding generic-array v0.14.7 (available: v0.14.9)
Adding getrandom v0.3.4 (available: v0.4.3)
Adding rand v0.9.5 (available: v0.10.3)
Adding rand_distr v0.5.1 (available: v0.6.0)
Adding sha2 v0.10.9 (available: v0.11.0)
```

Software compilation: Rust

How to install your own Rust packages?

4. Build the source code with **cargo** (build tool)

```
# Build (creates a Cargo.lock if needed)
cargo build --release
[...]
```

Compiling rand_distr v0.6.0
Compiling fake v5.1.0
Compiling regex v1.13.1
Building [=====>] 131/183: regex-automata, regex, rustix, chrono

5. Locate the new binary, usually located in **target/release**

You can run your program by executing the binary

```
./target/release/genact --help
A nonsense activity generator

Usage: genact [OPTIONS]

Options:
  -l, --list-modules      List available modules
  -m, --modules <MODULES> Run only these modules [possible values: ansible,
                           bootlog, botnet, bruteforce, cargo, cc, composer,
                           cryptomining, docker_build, docker_image_rm, download,
                           ...]
```



Software compilation: Rust

How to install your own Rust packages?

6. *Optional*: Add binary to **\$PATH**

If you want to run the tool from anywhere (no need for `./`)
add your binaries to your **PATH**:

```
export PATH=$PWD/target/release:$PATH
```

Check if executable is in **\$PATH** and **which** one is being
used:

```
which <command>
```



```
# binary is not in PATH
genact --version
-bash: genact: command not found
which genact
/usr/bin/which: no genact in
(/bin:/sbin:/usr/local/sbin:/usr/sbin)

# Add binary to path
export PATH=$PWD/target/release:$PATH

# Binary is now in PATH
which genact
$PWD/target/release/genact
genact --version
genact 1.5.1
```

Notes on Installing Rust Packages



- CPU architecture

```
cargo build --release
```

builds for a **baseline architecture** (like `x86-64`),
which runs on most CPUs.

You can enable micro-architecture optimization by
setting `$RUSTFLAGS`:

```
export RUSTFLAGS="-C target-cpu=native"
```

⚠ The binary will be **optimized** for your current CPU *but*
may **not run on older or different CPUs**.

- Home directory usage

Cargo stores build artifacts, registries, and Git
checkouts under `$HOME/.cargo` by default.

The home directory is **very limited**, you should
either **redirect `$CARGO_HOME` to a larger location**

```
export CARGO_HOME=/path/to/data/cargo
```

or symlink `.cargo` to another location, e.g.:

```
ln -s $VSC_DATA/.cargo $HOME/.cargo
```



Go is a high-level general purpose programming language that is statically typed and compiled.

How to install your own Go packages?

1. Load a Go module

```
module load Go/1.25.3
```

2. Get your source code

```
git clone https://github.com/rakyll/hey.git
Cloning into 'hey'...
remote: Enumerating objects: 1189, done.
remote: Counting objects: 100% (273/273), done.
remote: Compressing objects: 100% (71/71), done.
[...]
cd hey
git switch --detach v0.1.5
HEAD is now at e64ec7a Add upload target to the
Makefile and update the links
```

How to install your own Go packages?

3. Check the contents of the directory with the source code.

Go packages have a `go.mod` file

```
ls
Dockerfile      hey_test.go      requester
go.mod          LICENSE
go.sum          Makefile
hey.go          README.md
```

Recommended: Generate `go.sum` if missing

```
go mod tidy
go: downloading golang.org/x/net v0.48.0
go: downloading golang.org/x/text v0.33.0
```

- `go.mod` → what dependencies/versions the project needs
- `go.sum` → integrity verification (checksums)


```
# Build (Needs go.sum)
go build -o ./bin/hey ./
```

You can run your program by executing the binary:

[illegible]

Software compilation: Go



Notes on Installing Go Packages



- CPU architecture

```
go build
```

Go builds for a **baseline architecture** (like **x86-64**), which runs on most CPUs.

ISA-level optimization is possible. You can enable optimization by setting **\$GOHOSTARCH**, for example:

```
GOAMD64=v3 go build
```

generates a binary for x86-64 with the v3 ISA level, enabling AVX2, FMA...

⚠ It will not run on any x86-64 CPU that does not support the full v3 ISA level

- Home directory usage

Go stores build and cache data under **\$HOME** by default.

The home directory is **very limited**, you should either:

Redirect **\$GOPATH** and **\$GOCACHE** to a location with more storage space

```
# move main workspace (replaces $HOME/go)
export GOPATH=$VSC_DATA/go
# move compiled object cache (replaces
~/.cache/go-build)
export GOCACHE=$VSC_DATA/go-build
```

Or **create symlinks** from **\$HOME/go** and **\$HOME/.cache/go-build**

Manual software compilation

Manually compiling software (without a package manager or build tools) is **complex** and usually done only by **developers**

- It involves **compiling** each source file individually, managing dependencies
- **Linking** everything into the final executable.
- You must also handle **external libraries and headers**, installing and configuring them yourself.

There's **no automation or dependency tracking**, you need to re-run all commands yourself whenever something changes.

If you're not experienced with manual compilation, it's best to **contact user support** for assistance.

```
# simple program with two source files
ls
    program.c      functions.c

# compile each source code file
gcc -O2 -c functions.c
gcc -O2 -c program.c

# compilation generates binary objects
ls
    program.c      program.o
    functions.c    functions.o

# link the objects into executable
gcc program.o functions.o -o program

# now you can use the program
./program
```

⚠ Very simple case, real code is not like this

Manual software compilation: make

We've already seen solutions with fully fledged package managers.

There are also automation tools that build source code **without any package repository** and one of the oldest and most widely used is **GNU Make**

- **Introduced in 1976** (GNU Make version released in the 1980s) still widely used today
- **Language agnostic**: works with any compiler or language
- Uses a standalone **Makefile**, easy to integrate into any project
- The Makefile is a set of rules in a specific order that automates **compilation, linking, and installation**
- Can use system-installed libraries and headers, but will **fail** if dependencies are missing
- Does **not automatically optimize** builds for specific hardware

```
# projects supporting GNU Make can  
be identified by the presence of a  
Makefile
```

```
ls  
    functions.c  Makefile  
    program.c
```

```
# simple command to compile  
make
```

```
# compilation generates binary  
objects and link the objects into  
executable
```

```
ls  
    functions.c  functions.o  
    Makefile     program  
    program.c    program.o
```



CMake

- Automation tool to **generate Makefiles**
- Language agnostic
- Uses **CMakeLists.txt** files to define how the project is built
- **Configures** the compilation, linking, and installation for complex projects (multiple source files)
- **Detects available libraries, headers, and compilers**, adapting the build accordingly. Can enable/disable optional features (Example: if MPI is found → builds with MPI support otherwise it doesn't)
- Can **fetch or build missing dependencies**, **but this depends on what's in the project's CMakeLists.txt**
- Does **not automatically optimize** for your specific CPU architecture unless explicitly set it (e.g. via `-march=native` or flags).

```
# projects supporting CMake can be
# identified by the presence of
# CMakeLists.txt
ls
    functions.c    CMakeLists.txt
    program.c
# generate build directory
mkdir build-dir
cd build-dir
cmake ..
# build with make
make
# resulting program will usually be
# created in build-dir
ls
    functions.c    functions.o
    Makefile       program
    program.c      program.o
```

Manual software compilation

Recognizing Build Systems from Project Files

Build / Install File(s)	Tool / Command to Use			Module(s) to load
	Configure step	Build step	Install step	
configure / Makefile	<code>./configure</code> <i>(needs configure script)</i>	<code>make</code>	<code>make install</code>	<code>make</code>
build.ninja	<i>(None)</i>	<code>ninja</code>	<code>ninja install</code>	Ninja
CMakeLists.txt	<code>cmake /</code> <code>cmake -G Ninja</code>	<code>make /</code> <code>ninja</code>	<code>make install /</code> <code>ninja install</code>	CMake + make/Ninja
SConstruct	<code>scons</code>		<code>scons install</code>	SCons*
meson.build	<code>meson setup builddir</code>	<code>meson</code> <code>compile -C</code> <code>builddir</code>	<code>meson install -C</code> <code>builddir</code>	Meson
cargo.toml	<code>cargo build --release</code>		<code>cargo install</code>	Rust
go.mod	<code>go build</code>		<code>go install</code>	Go
setup.py requirements.txt pyproject.toml	<code>pip install</code>			Python

* Not available on all VSC clusters

Manually building software is rarely straightforward

- It is hard to identify all **requirements** for software you didn't develop yourself.
- It is hard to set up the system and **environment** to meet those requirements.

Manual builds are prone to failure

- Without a package manager, the build system can't automatically fix **missing dependencies**, you must install or point to them **manually**.
- Even if the build succeeds, the resulting binaries might still **fail to run** correctly.

Agenda

- [11:30-11:45] **Software compilation** (*Cintia Willemyns - VUB*)
- [11:45-12:00] **Building Apptainer containers** (*Sam Moors - VUB*)
- [12:00-12:15] **EasyBuild + Spack** (*Kenneth Hoste - UGent*)
- [12:15-12:30] **When should I use which tool + Conclusions** (*Kenneth Hoste - UGent*)
- [12:30-13:30] (*lunch*)
- [13:30-15:00] **Hands-on: choose your own adventure!**
- [15:00-15:30] (*coffee break*)
- [15:00-16:30] **Hands-on (continued)**
- [16:30-17:00] **Show & tell** (what did you work on?)

Building Apptainer containers

Overview



- **Interactively** from base image
- From **Apptainer definition file**
- From **Dockerfile** via docker image
- Using **hpc-container-wrapper**
- Advanced building: GPUs, performance-optimized
- Podman in HPC?

Building Apptainer containers



Preparation

```
# set environment variables (add to your ~/.bashrc)
# CACHEDIR: data that can be reused across Apptainer runs (layers, images)
# TMPDIR: temporary workspace
export APPTAINER_CACHEDIR=/tmp/$USER/apptainer_cachedir
export APPTAINER_TMPDIR=/tmp/$USER/apptainer_tmpdir
mkdir -p $APPTAINER_TMPDIR

# make sure you're in a location with enough space
cd /tmp/$USER # or $VSC_DATA, $VSC_SCRATCH
```

Building Apptainer containers

Interactively from base image

```
# 1) download a base image and store it as sandbox
# (= writable directory structure)
apptainer build --sandbox my_sandbox docker://ubuntu:24.04

# 2a) make changes from inside container (e.g. install packages)
# my_sandbox directory becomes root directory (/)
apptainer shell --writable --fakeroot my_sandbox
Apptainer> # (make changes here)

# 2b) make changes from the host (e.g. update runscript)
cd my_sandbox/
# (make changes here)

# 3) create immutable SIF image from sandbox
apptainer build my_image.sif my_sandbox
```

Building Apptainer containers



From Apptainer definition file (recommended for reproducibility)

```
# 1) write Apptainer definition file tblite-0.4.0.def
cat tblite-0.4.0.def
    Bootstrap: docker
    From: ubuntu:24.04
    ...

# 2a) definition file → sandbox → SIF image
apptainer build --fakeroot --sandbox my_sandbox tblite-0.4.0.def
# (check here your sandbox before creating final SIF image)
apptainer build tblite-0.4.0.sif my_sandbox

# 2b) definition file → SIF image
apptainer build --fakeroot tblite-0.4.0.sif tblite-0.4.0.def
```

Building Apptainer containers

Apptainer definition files

- **Header:** keywords

Bootstrap: `docker` → use docker registry (default: [Docker Hub](https://hub.docker.com/))

From: `ubuntu:24.04` → download ubuntu-24.04 image

- **Sections:** shell commands (`sh`)

%post → perform build/installation steps

%environment → write into `/.singularity.d/env/90-environment.sh`

%runscript → write into `/.singularity.d/runscript`

more: `%files`, `%setup`, `%arguments`, `%help`, `%test`, `%labels`, ...

Building Apptainer containers

From Docker image via Dockerfile

1) write/obtain Dockerfile

```
cat Dockerfile
```

```
FROM ubuntu:24.04
```

```
RUN ...
```

2) Dockerfile → Docker image

```
sudo docker build . -t my_docker_image
```

3) create Docker archive from Docker image

```
sudo docker save my_docker_image -o my_docker_archive.tar
```

```
sudo chown $USER:$USER my_docker_archive.tar
```

4a) Docker archive → sandbox → SIF image ([docs](#))

```
apptainer build --sandbox my_sandbox docker-archive:my_docker_archive.tar
```

```
apptainer build my_image.sif my_sandbox
```

4b) Docker archive → SIF image

```
apptainer build my_image.sif docker-archive:my_docker_archive.tar
```

Needs a machine with **Docker** (+ admin privileges)
or **Podman** (Windows: WSL).
Will not work in your VSC account!

Building Apptainer containers

Using **hpc-container-wrapper** for conda and pip ([docs](#))

```
# 1) write conda environment file environment.yml
cat environment.yml
  name: tblite
  channels:
    - conda-forge
  dependencies:
    - tblite-python=0.4.0

# 2) create Apptainer container + wrappers in new tblite-0.4.0 directory
module load hpc-container-wrapper/<VERSION>
conda-containerize new --prefix tblite-0.4.0 environment.yml

# example wrappers usage
export PATH="$PWD/tblite-0.4.0/bin:$PATH"
which tblite python
  $PWD/tblite-0.4.0/bin/tblite
  $PWD/tblite-0.4.0/bin/python
python -c 'import tblite; print(tblite.__file__)'
  /LOCAL_TYKKY_QcubNaZ/miniforge/envs/env1/lib/python3.13/site-packages/tblite/__init__.py
```

Building Apptainer containers

Updating containers built with **hpc-container-wrapper**

```
# 1) write update.sh script
cat update.sh
    conda install -c conda-forge beautifulsoup4

# 2) update container in tblite-0.4.0 directory with update.sh script
conda-containerize update --post-install update.sh tblite-0.4.0
```

Building Apptainer containers

GPU-enabled containers ([docs](#))

- How to run: bind mount GPU libraries into container
- Container apps must match host GPU/driver's supported **CUDA (ROCm) version** and **CUDA compute capability**.
 - Start from prebuilt CUDA (ROCm) container as **base image** (e.g. [NVIDIA container registry](#))
 - VSC clusters: select CUDA version that matches **available CUDA module**
- Container and host **libc** versions must be compatible.
 - Select **container OS version** from similar generation as **host OS version**

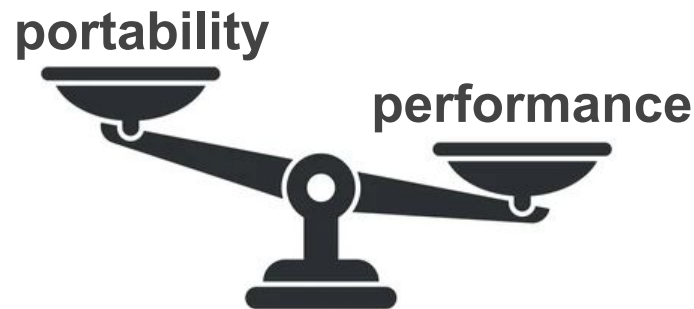
```
# NVIDIA
apptainer run --nv <image.sif>
# AMD
apptainer run --rocm <image.sif>
```

```
module spider CUDA
Versions:
  CUDA/12.6.0
  CUDA/12.8.0
```

Building Apptainer containers

Performance-optimized containers

- Cfr. Mobility vs performance tradeoff
- Requires building app + dependencies (!)
from source inside container
- Check advanced [VSC trainings](#) on GNU Make, CMake, Fortran, C++, ...
- Check VSC training on [Containers for HPC](#)



Podman in HPC?



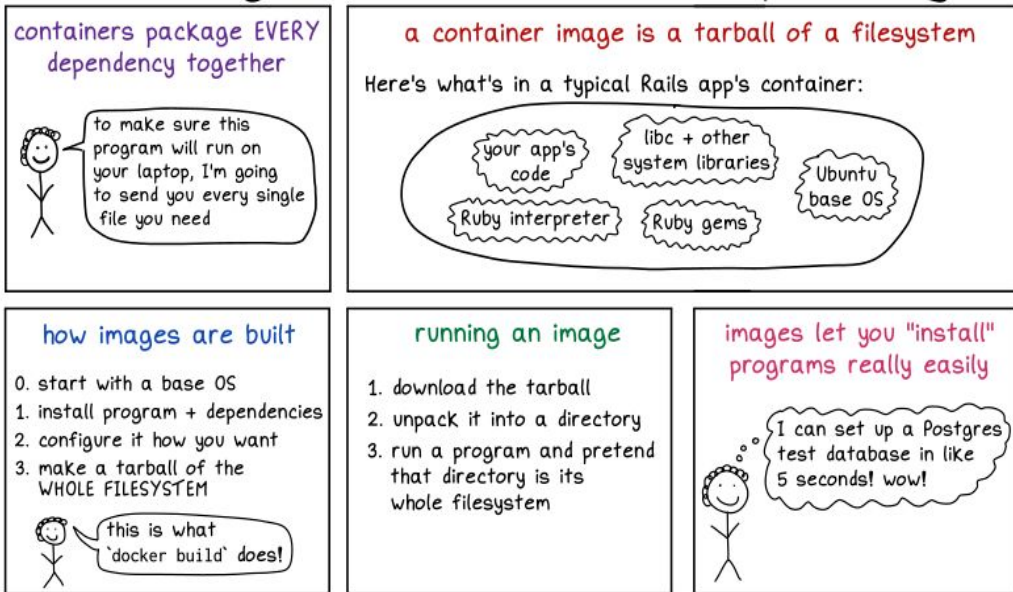
	 APPTAINER	 podman	 docker
Rootless	yes	yes	rootless mode
Daemonless	yes	yes	no
Build from Dockerfile	no	yes	yes
Docker-like commands	no	yes	yes
Run Docker images	yes**	yes	yes
Single file	yes	no	no
HPC integration	yes	no	no
Use on VSC systems	yes	only at KU Leuven	no

Building Apptainer containers

More documentation

- VSC [Apptainer docs](#)
- VSC training: [Containers for HPC](#)
- Official [Apptainer docs](#)

the big idea: include EVERY dependency ⁵



Agenda

- [11:30-11:45] **Software compilation** (*Cintia Willemyns - VUB*)
- [11:45-12:00] **Building Apptainer containers** (*Sam Moors - VUB*)
- [12:00-12:15] **EasyBuild + Spack** (*Kenneth Hoste - UGent*)
- [12:15-12:30] **When should I use which tool + Conclusions** (*Kenneth Hoste - UGent*)
- [12:30-13:30] (*lunch*)
- [13:30-15:00] **Hands-on: choose your own adventure!**
- [15:00-15:30] (*coffee break*)
- [15:00-16:30] **Hands-on (continued)**
- [16:30-17:00] **Show & tell** (what did you work on?)

- There's a wide variety of software installation tools...
- **Building from source code** is often recommended (or a necessary evil)
- It's important to get the details right to get the installation to work & perform well
- You should **check** whether things actually worked out the way you expect!
 - Run software test suite, run simple commands like `--help` as “smoke tests”, etc.
- **Tools** have been developed to **automate software installation** (from source) for HPC
 - EasyBuild and Spack are most prominent (on HPCs systems)



Spack

EasyBuild in a nutshell



<https://easybuild.io> | <https://docs.easybuild.io> | <https://github.com/easybuilders> | <https://easybuild.io/join-slack>

- EasyBuild is a **tool** to (build and) install (scientific) software
- Originally **created by HPC-UGent** in 2009, open source since 2012
- Automates software installation procedure (from source code)
- Generates environment module files (Lua or Tcl syntax)
- Strong focus on scientific software, performance, and HPC systems
- EasyBuild Slack as main communication channel for worldwide community
- Used in **VSC** (+ LUMI, ...) for installing and managing **central software stack**
 - And all across the world: EuroHPC systems, UK, Canada, ...

Installing & configuring EasyBuild



- To install EasyBuild, use `pip install easybuild`
 - It is recommended to use a Python virtual environment!
 - Recommended optional dependencies: `rich` (colors), `archspec` (CPU detection)
- Configuration via configuration files, environment variables, command line options

```
# show active configuration
eb --show-config
# see all configuration options
eb --show-full-config
```

- *Do not use default configuration!* (since it abuses `$HOME/.local/easybuild`)
- See <https://docs.easybuild.io/configuration>
- Example **minimal configuration** via environment variables:

```
export EASYBUILD_PREFIX=$VSC_DATA/easybuild
export EASYBUILD_BUILDPATH=/tmp/$USER
```

Basic usage of EasyBuild



- Basic workflow:
 - Find an easyconfig file with `eb --search`, or create one
 - Evaluate before installing with `eb --missing`, `eb -x`, ...
 - Use `eb` command to install, use `--robot` to enable dependency resolution
- Example to install & use tblite:

```
eb --search tblite
eb tblite-0.4.0-gfbf-2024a.eb --missing
eb tblite-0.4.0-gfbf-2024a.eb --robot # build + install tblite (incl. deps)
module use /path/to/modules/all      # make module available to load
module load tblite/0.4.0-gfbf-2024a   # load tblite for usage
```

- Existing environment module files will automatically be used for dependencies, if they are visible via `module avail`

Spack in a nutshell



Spack

VLAAMS
SUPERCOMPUTER
CENTRUM



<https://spack.io> | <https://spack.readthedocs.io> | <https://https://slack.spack.io> | <https://github.com/spack>

- Spack is a **package manager** for supercomputers
- Originally **created by Lawrence Livermore National Lab (LLNL)** in 2014
- Automates software installation procedure, can install from binary cache
- Can generate environment module files, but also has its own load mechanism
- Focus on scientific software & HPC systems, but also supports macOS & Windows
- Spack Slack as main communication channel for worldwide community
- Popular with **software developers** due to usage model & specific features

Installing & configuring Spack



Spack

VLAAMS
SUPERCOMPUTER
CENTRUM



- “Install” by cloning Git repository:

```
git clone https://github.com/spack/spack
```

 - Software will be installed (by default) in **spack/opt/spack**, so **pick a good location** (not your home directory in your VSC account)
 - Note: this implies using the (potentially unstable) development version
- Set up environment by sourcing the provided script, for example (for bash shell):

```
source spack/share/spack/setup-env.sh
```
- Make Spack aware of available (system) compilers

```
spack compiler find
```
- Note: by default, Spack will cache some stuff in **\$HOME/.spack**

Basic usage of Spack



Spack

VLAAMS
SUPERCOMPUTER
CENTRUM



- Basic workflow:
 - Check whether software is supported: `spack list tblite`
 - Show details of specific software: `spack info tblite`
 - Show dependency graph of specific software: `spack graph tblite`
- Examples for installing tblite:
 - Instal tblite incl. Python bindings with system compiler:
`spack install tblite@0.4.0 +python`
 - Install tblite with GCC 13 as compiler:
`spack install gcc@13`
`spack install tblite@0.4.0 +python %gcc@13`
- Load tblite for usage: `spack find tblite` `spack load tblite`

EasyBuild vs Spack: similarities



Both are:

- **Open source** software, implemented in **Python**
- Developed & supported by a **worldwide community**
- Focus on scientific software, performance, **HPC** systems
- **No admin privileges** required to build & install software
- Highly **configurable**: you can tune them to your preferences
- “Expert system” for installing (scientific) software
- Wraps around commonly used tools like CMake, make, pip, ...
- **Not a magic solution**: building software may still fail (compiler errors, etc.)

EasyBuild vs Spack: differences



- Community is mostly **European**
- License: GPLv2
- Installation via `pip install` (or EasyBuild)
- Support for **Linux** (+ macOS)
- Interesting features for **HPC support teams**
- **Stable EasyBuild releases** since 2012
- Build from source code (when possible)
- Integration with Slurm, GitHub, ...
- Functionality can be customized via hooks
- ...



- Community is mostly **US-based**
- License: Apache 2.0 + MIT
- Typical installation via `git clone`
- Support for Linux, macOS, Windows
- Interesting features for **software developers**
- **Spack 1.0** released summer 2025
- Binary cache (pre-built packages)
- Good support for building container images
- Support for environments
- ...

Why you should (not) use



or



Spack



- **Uniform way of installing software**, regardless of specifics of installation procedure
- **Automate** (complex) software installation procedures, incl. required dependencies
- Build a **consistent** (central) software stack, with software **performance** in mind
- Manage same software stack across different systems/partitions
- **Leverage efforts by experienced people** who contributed to these tools
- Get help through **community** channels (dedicated Slack for both)
- There is a **steep initial learning** curve for both
 - Some experience with compilers & manually installing software is recommended
 - Take some time to learn how to use the tool you picked (docs, tutorial, ...)

Agenda

- [11:30-11:45] **Software compilation** (*Cintia Willemyns - VUB*)
- [11:45-12:00] **Building Apptainer containers** (*Sam Moors - VUB*)
- [12:00-12:15] **EasyBuild + Spack** (*Kenneth Hoste - UGent*)
- [12:15-12:30] **When should I use which tool + Conclusions** (*Kenneth Hoste - UGent*)
- [12:30-13:30] (*lunch*)
- [13:30-15:00] **Hands-on: choose your own adventure!**
- [15:00-15:30] (*coffee break*)
- [15:00-16:30] **Hands-on (continued)**
- [16:30-17:00] **Show & tell** (what did you work on?)

So... which tool should I use?

- One single definitive answer:

So... which tool should I use?

- One single definitive answer:
- **It depends**, on various factors:
 - Which **software** you need
 - On which **system(s)** you want to use that software
 - Whether you want to:
 - Experiment a bit
 - Prepare to use the software long-time (“production”)
 - To what extent the **performance** of the software installations is important for you
 - Your level of **expertise** with installing software (on Linux systems)
 - How much **time you’re willing to spend** on getting the software installed...

“Totally unbiased” high-level comparison



👍 or 👎 ?	Central software stack	EESSI	(pre-built) containers	Conda & co	Language pkg mgrs	Manual installation	EasyBuild	Spack
<i>Supported software</i>								
<i>User experience</i>								
<i>Performance of installed software</i>								
<i>Reproducibility of the installation</i>								
<i>Time required to install software</i>								
<i>Testing of installations</i>								
<i>Storage (volume, # files)</i>								
<i>Learning curve</i>								
<i>Getting help (in VSC)</i>								

“Totally unbiased” high-level comparison



/apps	Central software stack	EESSI	(pre-built) containers	Conda & co	Language pkg mgrs	Manual installation	EasyBuild	Spack
<i>Supported software</i>	👍👍👍							
<i>User experience</i>	👍👍👍							
<i>Performance of installed software</i>	👍👍👍							
<i>Reproducibility of the installation</i>	👍👍👍							
<i>Time required to install software</i>	👍👎							
<i>Testing of installations</i>	👍👍							
<i>Storage (volume, # files)</i>	(👍👍)							
<i>Learning curve</i>	👍👍👍							
<i>Getting help (in VSC)</i>	👍👍👍							

“Totally unbiased” high-level comparison



	Central software stack	EESSI	(pre-built) containers	Conda & co	Language pkg mgrs	Manual installation	EasyBuild	Spack
<i>Supported software</i>	👍👍👍	👍						
<i>User experience</i>	👍👍👍	👍👍👍						
<i>Performance of installed software</i>	👍👍👍	👍👍👍						
<i>Reproducibility of the installation</i>	👍👍👍	👍👍👍						
<i>Time required to install software</i>	👍👎	👎						
<i>Testing of installations</i>	👍👍	👍👍👍						
<i>Storage (volume, # files)</i>	(👍👍)	(👍👍)						
<i>Learning curve</i>	👍👍👍	👍👍👍						
<i>Getting help (in VSC)</i>	👍👍👍	👍👍						

“Totally unbiased” high-level comparison



 APPTAINER	Central software stack	EESSI	(pre-built) containers	Conda & co	Language pkg mgrs	Manual installation	EasyBuild	Spack
<i>Supported software</i>	👍👍👍	👍	👍👍					
<i>User experience</i>	👍👍👍	👍👍👍	👍👍					
<i>Performance of installed software</i>	👍👍👍	👍👍👍	👍👎					
<i>Reproducibility of the installation</i>	👍👍👍	👍👍👍	👍👍					
<i>Time required to install software</i>	👍👎	👎	👍👍					
<i>Testing of installations</i>	👍👍	👍👍👍	👎					
<i>Storage (volume, # files)</i>	(👍👍)	(👍👍)	👍👍					
<i>Learning curve</i>	👍👍👍	👍👍👍	👍					
<i>Getting help (in VSC)</i>	👍👍👍	👍👍	👍👍					

“Totally unbiased” high-level comparison



 ONDA®	Central software stack	EESSI	(pre-built) containers	Conda & co	Language pkg mgrs	Manual installation	EasyBuild	Spack
<i>Supported software</i>	👍👍👍	👍	👍👍	👍👍				
<i>User experience</i>	👍👍👍	👍👍👍	👍👍	👍👍				
<i>Performance of installed software</i>	👍👍👍	👍👍👍	👍👎	👍👎				
<i>Reproducibility of the installation</i>	👍👍👍	👍👍👍	👍👍	👍				
<i>Time required to install software</i>	👍👎	👎	👍👍	👍				
<i>Testing of installations</i>	👍👍	👍👍👍	👎	👍				
<i>Storage (volume, # files)</i>	(👍👍)	(👍👍)	👍👍	👎👎👎				
<i>Learning curve</i>	👍👍👍	👍👍👍	👍	👍👍				
<i>Getting help (in VSC)</i>	👍👍👍	👍👍	👍👍	👍				

“Totally unbiased” high-level comparison



	Central software stack	EESSI	(pre-built) containers	Conda & co	Language pkg mgrs	Manual installation	EasyBuild	Spack
<i>Supported software</i>	👍👍👍	👍	👍👍	👍👍	👍			
<i>User experience</i>	👍👍👍	👍👍👍	👍👍	👍👍	👍👍			
<i>Performance of installed software</i>	👍👍👍	👍👍👍	👍👎	👍👎	👍👎			
<i>Reproducibility of the installation</i>	👍👍👍	👍👍👍	👍👍	👍	👍			
<i>Time required to install software</i>	👍👎	👎	👍👍	👍	👍👍			
<i>Testing of installations</i>	👍👍	👍👍👍	👎	👍	👍			
<i>Storage (volume, # files)</i>	(👍👍)	(👍👍)	👍👍	👎👎👎	👎			
<i>Learning curve</i>	👍👍👍	👍👍👍	👍	👍👍	👍👍			
<i>Getting help (in VSC)</i>	👍👍👍	👍👍	👍👍	👍	👍👍			

“Totally unbiased” high-level comparison



	Central software stack	EESSI	(pre-built) containers	Conda & co	Language pkg mgrs	Manual installation	EasyBuild	Spack
<i>Supported software</i>	👍👍👍	👍	👍👍	👍👍	👍	👍👍👍		
<i>User experience</i>	👍👍👍	👍👍👍	👍👍	👍👍	👍👍	👎👎		
<i>Performance of installed software</i>	👍👍👍	👍👍👍	👍👎	👍👎	👍👎	(👍👍)		
<i>Reproducibility of the installation</i>	👍👍👍	👍👍👍	👍👍	👍	👍	👍👎		
<i>Time required to install software</i>	👍👎	👎	👍👍	👍	👍👍	👎👎👎		
<i>Testing of installations</i>	👍👍	👍👍👍	👎	👍	👍	👎		
<i>Storage (volume, # files)</i>	(👍👍)	(👍👍)	👍👍	👎👎👎	👎	👎		
<i>Learning curve</i>	👍👍👍	👍👍👍	👍	👍👍	👍👍	👎👎👎		
<i>Getting help (in VSC)</i>	👍👍👍	👍👍	👍👍	👍	👍👍	👍		

“Totally unbiased” high-level comparison



  Spack	Central software stack	EESSI	(pre-built) containers	Conda & co	Language pkg mgrs	Manual installation	EasyBuild	Spack
Supported software	👍👍👍	👍	👍👍	👍👍	👍	👍👍👍	👍👍👍	👍👍👍
User experience	👍👍👍	👍👍👍	👍👍	👍👍	👍👍	👎👎	👍	👍
Performance of installed software	👍👍👍	👍👍👍	👍👎	👍👎	👍👎	(👍👍)	👍👍👍	👍👍👍
Reproducibility of the installation	👍👍👍	👍👍👍	👍👍	👍	👍	👍👎	👍👍	👍👍
Time required to install software	👍👎	👎	👍👍	👍	👍👍	👎👎👎	👎👎	👎
Testing of installations	👍👍	👍👍👍	👎	👍	👍	👎	👍👍👍	👍
Storage (volume, # files)	(👍👍)	(👍👍)	👍👍	👎👎👎	👎	👎	👎👎	👎👎
Learning curve	👍👍👍	👍👍👍	👍	👍👍	👍👍	👎👎👎	👎👎	👎👎
Getting help (in VSC)	👍👍👍	👍👍	👍👍	👍	👍👍	👍	👍👍👍	👍

“Totally unbiased” high-level comparison



<i>It depends...</i>	Central software stack	EESSI	(pre-built) containers	Conda & co	Language pkg mgrs	Manual installation	EasyBuild	Spack
<i>Supported software</i>	👍👍👍	👍	👍👍	👍👍	👍	👍👍👍	👍👍👍	👍👍👍
<i>User experience</i>	👍👍👍	👍👍👍	👍👍	👍👍	👍👍	👎👎	👍	👍
<i>Performance of installed software</i>	👍👍👍	👍👍👍	👍👎	👍👎	👍👎	(👍👍)	👍👍👍	👍👍👍
<i>Reproducibility of the installation</i>	👍👍👍	👍👍👍	👍👍	👍	👍	👍👎	👍👍	👍👍
<i>Time required to install software</i>	👍👎	👎	👍👍	👍	👍👍	👎👎👎	👎👎	👎
<i>Testing of installations</i>	👍👍	👍👍👍	👎	👍	👍	👎	👍👍👍	👍
<i>Storage (volume, # files)</i>	(👍👍)	(👍👍)	👍👍	👎👎👎	👎	👎	👎👎	👎👎
<i>Learning curve</i>	👍👍👍	👍👍👍	👍	👍👍	👍👍	👎👎👎	👎👎	👎👎
<i>Getting help (in VSC)</i>	👍👍👍	👍👍	👍👍	👍	👍👍	👍	👍👍👍	👍

- Getting scientific software installed can be messy, but tools can help you (a lot)
- **Pick a tool & stick to it!** Don't mix & match from different ecosystems...
 - Don't mix `conda` & environment modules, don't mix `conda` & `pip`, don't use `pip` on top of modules installed with EasyBuild (unless you use `(vsc-)venv...`), ...
- Take some time to **familiarize yourself** with the tool(s) you are using
 - Read The Fine Manual (RTFM)
 - When you use LLM-based AI tools like ChatGPT, be careful...
- Get in touch with the **community** behind the tool(s) you are using
- **When in doubt, ask for help:** https://docs.vscentrum.be/contact_vsc.html

Hands-on session (after lunch)

- For the rest of this training session, you **choose your own adventure!**
- Exercises are available for each of the tools we covered
 - See <https://vscentrum.github.io/gssi-training/#exercises>
 - If you would like to learn more about a particular tool, try working through them
- You can also focus on the software you want to install
 - Which tool is the best fit for your use case, workflow, software, etc.?
 - Which problems do you encounter when trying to the tool you selected?
- **Ask questions to the VSC staff, we are here to help!**

Lunch break!



12:30 - 13:30