VLAAMS SUPERCOMPUTER CENTRUM

Vlaanderen
is supercomputing

Getting Scientific Software Installed

3 Nov 2025

Brussels (+ virtually via Teams)

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

compute@vscentrum.be

Goal of this training

VLAAMS SUPERCOMPUTER CENTRUM

- **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
- Hands-on demos
- High-level comparison between tools

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)
- Alex Domingo, Cintia Willemyns, Sam Moors (VUB)

VLAAMS SUPERCOMPUTER CENTRUM

https://vscentrum.be

Hands-on sessions on Thu 6 Nov 2025

VLAAMS SUPERCOMPUTER CENTRUM

- KU Leuven: https://icts.kuleuven.be/apps/onebutton/registrations/1228042
- UAntwerpen: confirm your attendance via e-mail to hpc@uantwerpen.be
- UGent:
  - Location: UGent campus Sterre, building S9, Multimediazaal
  - Time slots: 10:00-12:00, 13:00-15:00, 15:00-17:00 CET
  - Registration: https://event.ugent.be/registration/gssihandson
- VUB:
  - Location: Sablon room, Pleinlaan 9, 5th floor, Main Campus Etterbeek
  - Time: 13:00 to 16:00 CET
  - Registration: via e-mail to hpc@vub.be (+ indicate time of arrival)

Talk to us or contact compute@vscentrum.be for more information

Kenneth

Practical aspects

VLAAMS SUPERCOMPUTER CENTRUM

- Slides & example scripts available via GitHub repository: https://github.com/vscentrum/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

VLAAMS SUPERCOMPUTER CENTRUM

- 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)

VLAAMS SUPERCOMPUTER CENTRUM

- **(Almost) Everything shown in this training is covered in the VSC documentation**

  docs.vscentrum.be

- VSC sites have extra documentation/info pages

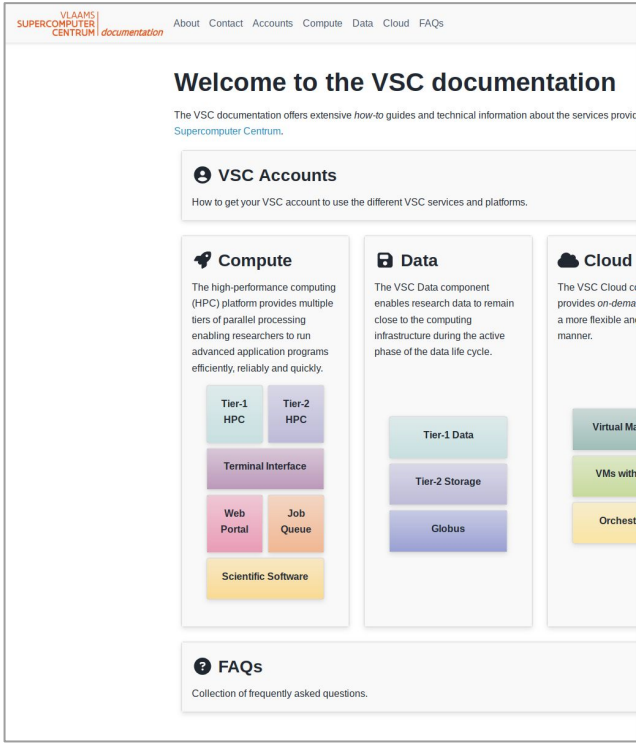
  KU Leuven: hpcleuven.github.io/HPC-intro

  UGent: docs.hpc.ugent.be

  UAntwerp: hpc.uantwerpen.be/support/documentation

  VUB: hpc.vub.be

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



Kenneth

Agenda (1/2)

VLAAMS SUPERCOMPUTER CENTRUM

- [09:00-10:00] **Informal welcome**
- [10:00-10:15] **Introduction** *(Kenneth Hoste - UGent)*
- [10:15-10:30] **Central software stack + environment modules** *(Kenneth Hoste - UGent)*
- [10:30-11:00] **European Environment for Scientific Software Installations (EESSI)** *(Lara Peeters - UGent)*
- [11:00-11:30] **Using Apptainer containers** *(Sam Moors - VUB)*
- [11:30-12:00] **Conda / Mamba / Pixi** *(Maxime Van den Bossche - KU Leuven)*
- [12:00-13:00] **Lunch break**

Agenda (2/2)

VLAAMS SUPERCOMPUTER CENTRUM

- [13:00-13:30] **Python: pip + venv** *(Steven Vandenbrande - KU Leuven)*
- [13:30-14:00] **Other language package managers: R, Julia** *(Alex Domingo - VUB)*
- [14:00-14:30] **Software compilation** *(Cintia Willemyns - VUB)*
- [14:30-15:00] **Coffee break**
- [15:00-15:30] **Building Apptainer containers** *(Sam Moors - VUB)*
- [15:30-16:00] **EasyBuild + Spack** *(Kenneth Hoste - UGent)*
- [16:00-16:30] **Conclusions** *(Kenneth Hoste - UGent)*
- [16:30-17:00] **Q&A**

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

VLAAMS SUPERCOMPUTER CENTRUM

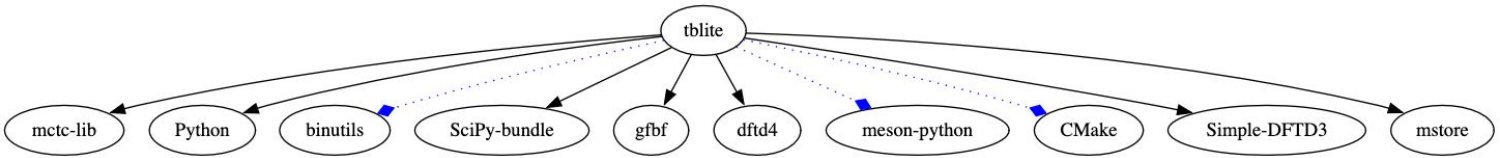
To get started, let's look at a couple of important aspects of installing software:

- Software applications have **dependencies**
- **Interpreted** vs **compiled** software
- **Binary programs** use hardware instructions (ISA)
- **Hardware platform** matters: CPU vs GPU, CPU microarchitecture, …

Software dependencies

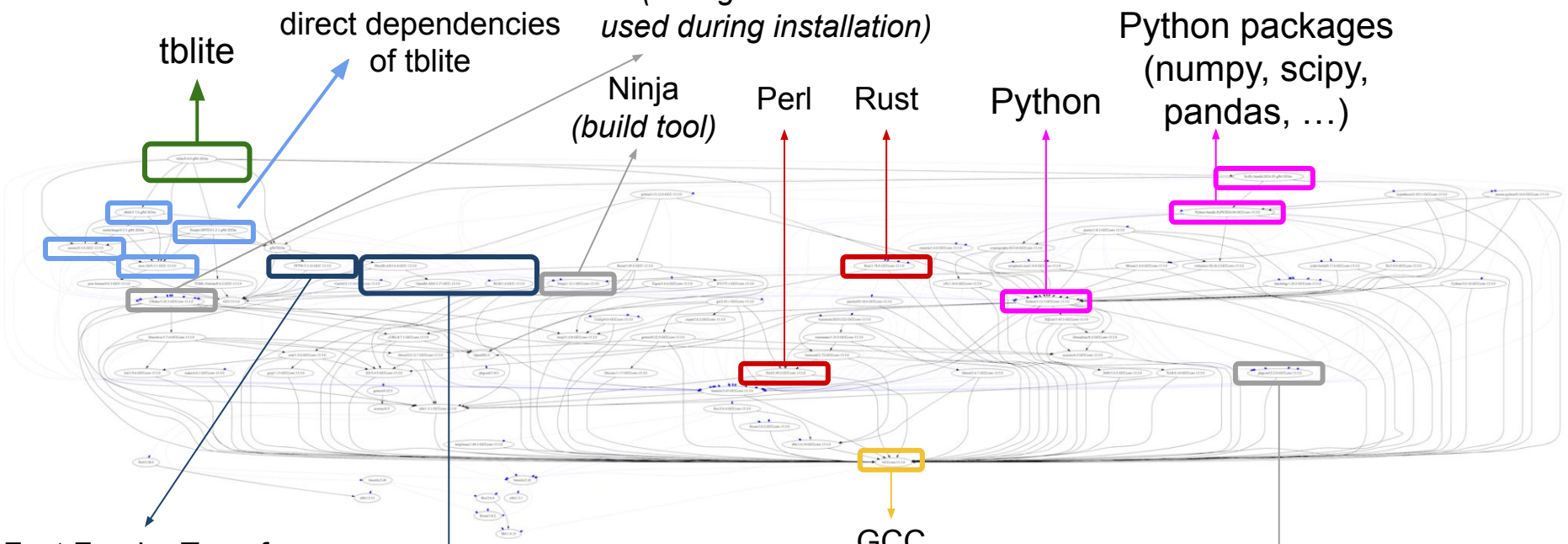
VLAAMS SUPERCOMPUTER CENTRUM

- Software applications don't exist in isolation, they **depend** on other software
- You need a **compiler** to build software from source to binary format (so you can run it)
- Software implemented in Python requires to have the Python interpreter installed
  - A *compatible* version of Python, may require several 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



Kenneth

Dependency hell



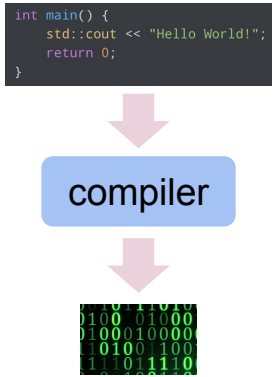
(dependency graph for tblite 0.4.0, created with EasyBuild)

Kenneth

Interpreted vs compiled software

VLAAMS SUPERCOMPUTER CENTRUM

- 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, …)



Binary programs use hardware instructions (ISA)

VLAAMS SUPERCOMPUTER CENTRUM

- 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

CPU family & microarchitecture

VLAAMS SUPERCOMPUTER CENTRUM

- There are different CPU families: **x86_64** (AMD, Intel), **aarch64** (64-bit Arm), …
  - An **x86_64** binary (AMD + Intel) will not run on an **aarch64** (Arm) 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…

GPU families & architectures

VLAAMS SUPERCOMPUTER CENTRUM

- Things get 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: Volta (V100), Ampere (A100, A2, …), Hopper (H100), Blackwell, …
- **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

VLAAMS SUPERCOMPUTER CENTRUM

- 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)
  - `archspec` CPU detection tool (https://github.com/archspec/archspec)
- 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`)

Topics covered in this training

VLAAMS SUPERCOMPUTER CENTRUM

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

Agenda

VLAAMS SUPERCOMPUTER CENTRUM

- [09:00-10:00] Informal welcome
- [10:00-10:15] Introduction *(Kenneth Hoste - UGent)*
- [10:15-10:30] **Central software stack + environment modules** *(Kenneth Hoste - UGent)*
- [10:30-11:00] European Environment for Scientific Software Installations (EESSI) *(Lara Peeters - UGent)*
- [11:00-11:30] Using Apptainer containers *(Sam Moors - VUB)*
- [11:30-12:00] Conda / Mamba / Pixi *(Maxime Van den Bossche - KU Leuven)*
- [12:00-13:00] Lunch break

Central software stack

VLAAMS SUPERCOMPUTER CENTRUM

- 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
  - Version of operating system is also a factor (RHEL8, 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

VLAAMS SUPERCOMPUTER CENTRUM

- 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 (changes over time) will be loaded

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

BAD: `module load tblite`

Environment variables for finding software

VLAAMS SUPERCOMPUTER CENTRUM

- **`$PATH`** : list of paths in which binary executables can be found
- **`$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 a command is available, use
which tblite
# or
command -v tblite
```

```
# 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
```

Toolchain concept

VLAAMS SUPERCOMPUTER CENTRUM

- 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`, `gompi`, …)
- **You should <u>not</u> 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
```

Hands-on demo: central software stack (1/2)

VLAAMS SUPERCOMPUTER CENTRUM

```
# 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/gent/RHEL9/zen2-ib/modules/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
  ...
```

Hands-on demo: central software stack (2/2)

VLAAMS SUPERCOMPUTER CENTRUM

```
# 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/gent/RHEL9/zen2-ib/software/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

VLAAMS SUPERCOMPUTER CENTRUM

- **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 request form: https://docs.hpc.ugent.be/software_installation_requests
- Some patience may be needed…

Lara

Homework (central software stack)

VLAAMS SUPERCOMPUTER CENTRUM

- Which changes are made in the environment when a tblite module is loaded?
  (w.r.t. `tblite` command and `tblite` Python bindings)
- Which module for `matplotlib` is compatible with the module for `tblite` 0.4.0?
- Which module for `tqdm` is compatible with the module for `tblite` 0.4.0?
- Which module that is compatible with the already loaded modules provides `numpy`?
- How many Python packages are available after loading the modules
  for `tblite`, `matplotlib`, and `tqdm`?

Agenda

VLAAMS SUPERCOMPUTER CENTRUM

- [09:00-10:00] Informal welcome
- [10:00-10:15] Introduction *(Kenneth Hoste - UGent)*
- [10:15-10:30] Central software stack + environment modules *(Kenneth Hoste - UGent)*
- [10:30-11:00] **European Environment for Scientific Software Installations (EESSI)** *(Lara Peeters - UGent)*
- [11:00-11:30] Using Apptainer containers *(Sam Moors - VUB)*
- [11:30-12:00] Conda / Mamba / Pixi *(Maxime Van den Bossche - KU Leuven)*
- [12:00-13:00] Lunch 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

https://eessi.io

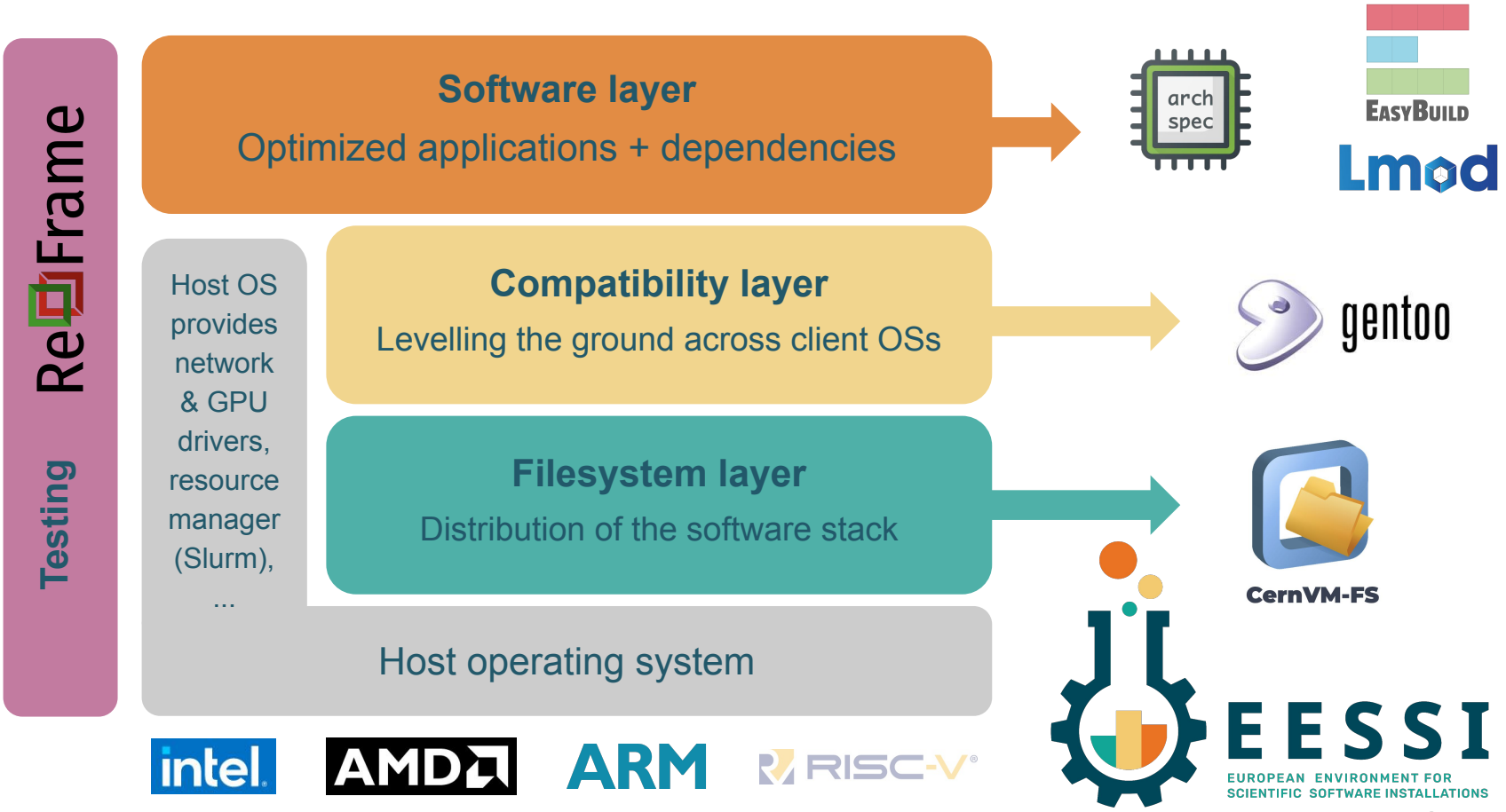
Lara

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, …

Lara

High-level overview of EESSI



EESSI ingredients

Compatibility layer

Abstraction from the host operating system

Filesystem Layer

Global distribution of software installations via CernVM-FS

EESSI

EUROPEAN ENVIRONMENT FOR SCIENTIFIC SOFTWARE INSTALLATIONS

Software Layer

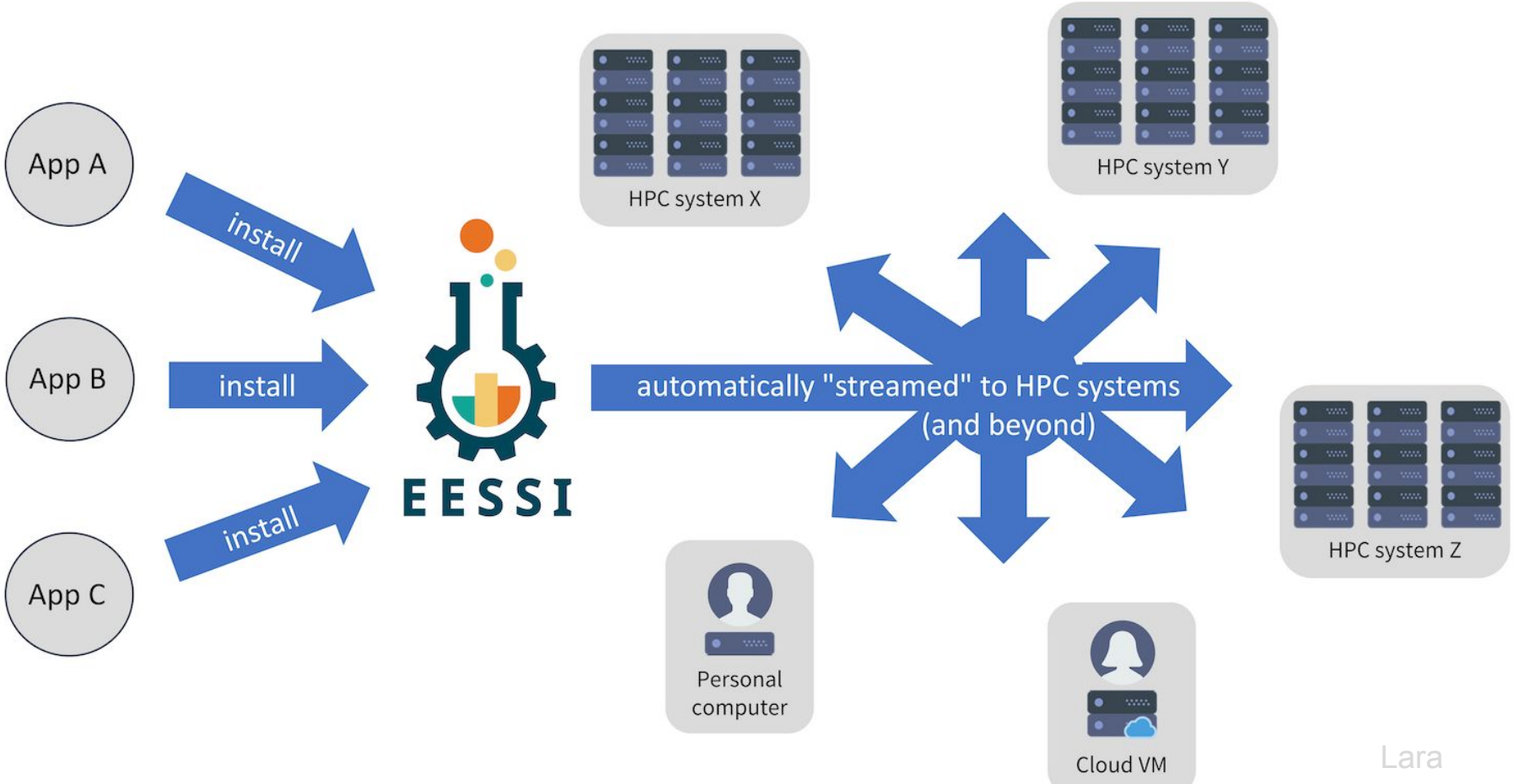
Optimized software installations for specific CPU microarchitectures

Intuitive user interface: module avail, module load, …

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

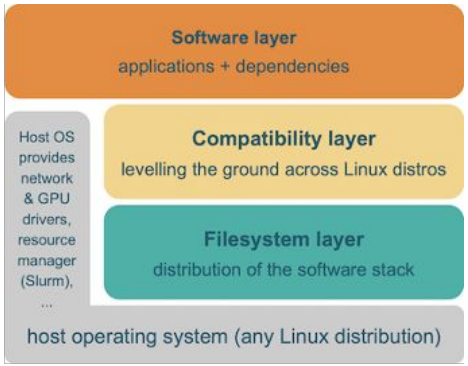
Lara

EESSI as a shared software stack



Lara

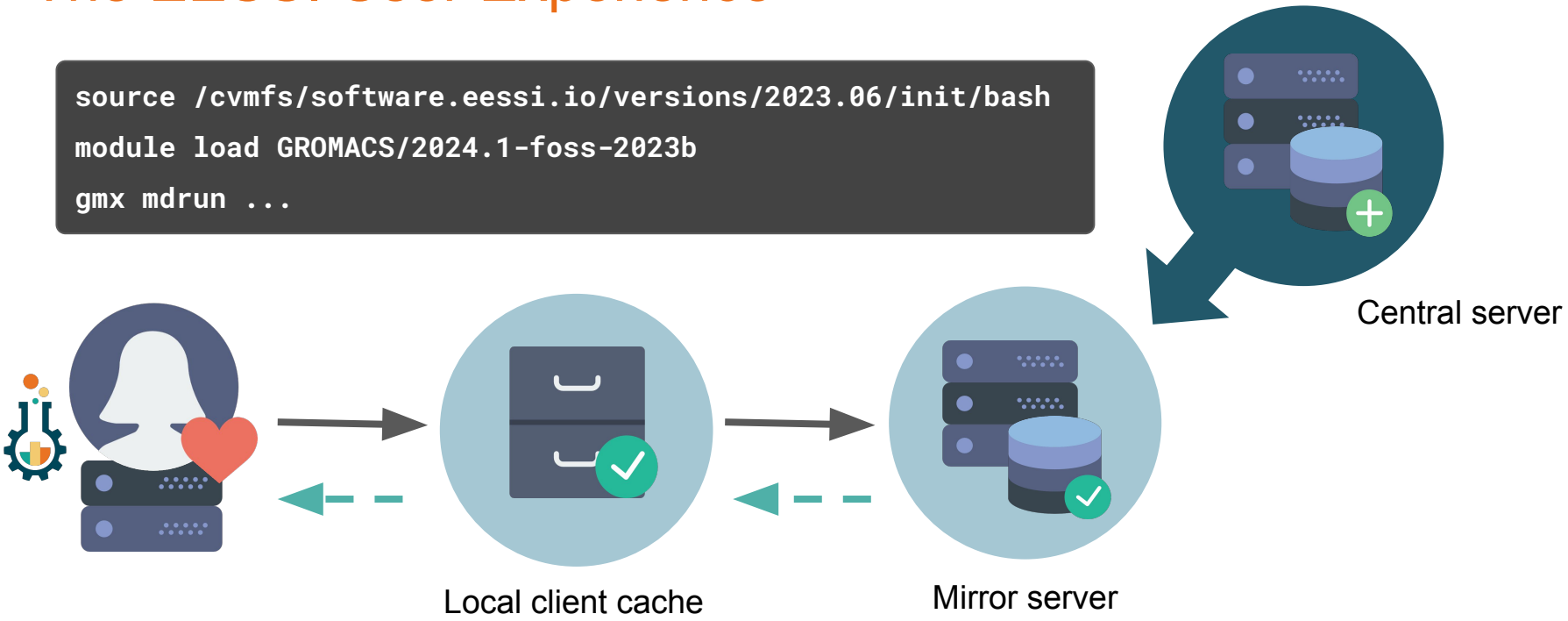
How does EESSI work?



- 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/2023.06/init/bash
module load GROMACS/2024.1-foss-2023b
gmx mdrun ...
```



EESSI provides **on-demand streaming**

of (scientific) software (like music, TV-series, …)

Lara

Demo: 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
module --force purge   # this is not required, but it is recommended to start afresh
source /cvmfs/software.eessi.io/versions/2023.06/init/bash
   Found EESSI repo @ /cvmfs/software.eessi.io/versions/2023.06!
   archdetect says x86_64/intel/cascaselake
   archdetect found supported accelerator for CPU target x86_64/intel/cascadelake: accel/nvidia/cc80
   Using x86_64/intel/cascadelake as software subdirectory.
   Using /cvmfs/software.eessi.io/versions/2023.06/software/linux/x86_64/amd/cascadelake/modules/all
   Using ...
   ...
   Environment set up EESSI (2023.06), have fun!
module avail tblite
module load tblite/0.4.0-gfbf-2024a
```

Alternative ways of accessing EESSI are available, via a container image, via cvmfsexec, …
eessi.io/docs/getting_access/native_installation - eessi.io/docs/getting_access/eessi_container

Demo: Running tblite using EESSI at UGent

```
source /cvmfs/software.eessi.io/versions/2023.06/init/bash
git clone https://github.com/EESSI/eessi-demo.git
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
```

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

Getting access to EESSI via CernVM-FS


```
# Native installation, requires admin privileges
# Installation commands are for RHEL-based distros
# Like CentOS, Rocky Linux, Almalinux, Fedora, ...

# Install CernVM-FS to get access to CernVM-FS repositories (incl. EESSI)
sudo yum install -y
  https://ecsft.cern.ch/dist/cvmfs/cvmfs-release/cvmfs-release-latend.noarch.rpm
sudo yum install -y cvmfs

# Create client configuration file for CernVM-FS
# (no proxy, 10GB local CernVM-FS client cache)
sudo bash -c "echo 'CVMFS_CLIENT_PROFILE="single"' > /etc/cvmfs/default.local"
sudo bash -c "echo 'CVMFS_QUOTA_LIMIT=10000' >> /etc/cvmfs/default.local"

# Install CernVM-FS
sudo cvmfs_config setup
```

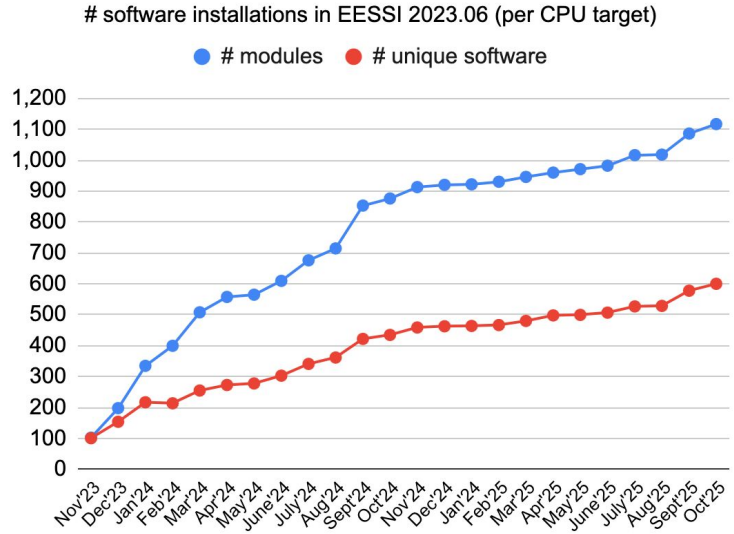
Alternative ways of accessing EESSI are available, via a container image, via cvmfsexec, …
eessi.io/docs/getting_access/native_installation - eessi.io/docs/getting_access/eessi_container

Lara

Overview of available software in EESSI

Currently more than 1,100 software installations available per supported CPU target via `software.eessi.io` CernVM-FS repository, increasing every week

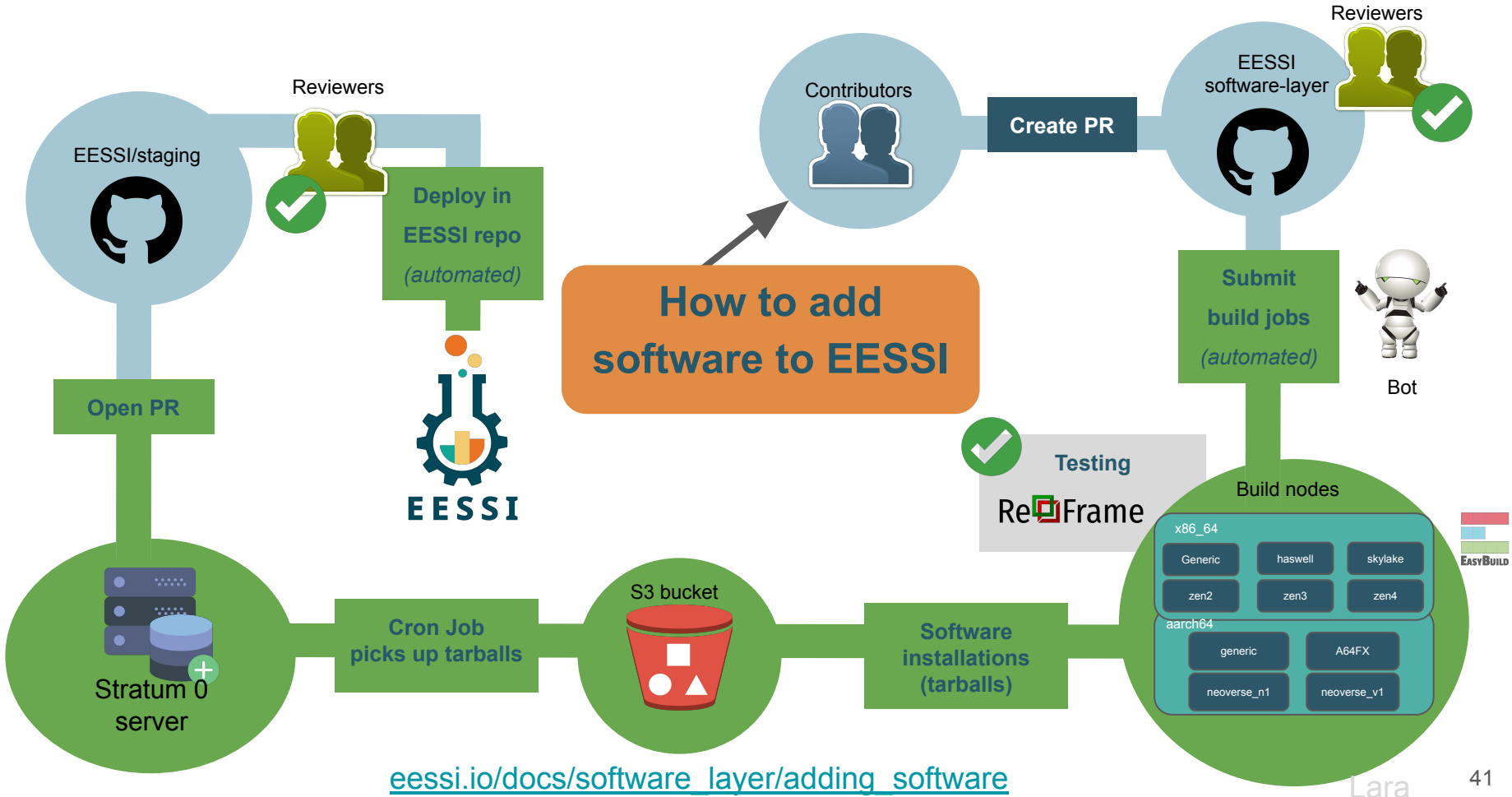
- **13** (+1) **supported CPU targets** (x86_64 + Arm), see https://eessi.io/docs/software_layer/cpu_targets
- **~600 different software packages**, excl. extensions: Python packages, R libraries
- **Almost 16,000 software installations in total**
- Including ESPResSo, GROMACS, LAMMPS, OpenFOAM, PyTorch, R, QuantumESPRESSO, TensorFlow, waLBerla, WRF, …
- https://eessi.io/docs/available_software/overview



- Using `foss/2023a` and `foss/2023b` toolchains in EESSI 2023.06
- Using `foss/2024a` and `foss/2025a` toolchains in EESSI 2025.06

Lara

Semi-automated workflow for adding software to EESSI



eessi.io/docs/software_layer/adding_software

Building software on top of EESSI

- You can use the software available in EESSI to

  resolve dependencies of the software you need

- To install software with EasyBuild on top of EESSI,

  first load the `EESSI-extend` module to correctly configure EasyBuild

- To manually install software on top of EESSI,

  first load the `buildenv` module

- See also https://eessi.io/docs/using_eessi/building_on_eessi

On which systems is EESSI available?

- On VSC systems:
  - **Readily available on Tier-2 infrastructure at UGent + VUB + Tier-1 Hortense**
  - 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**

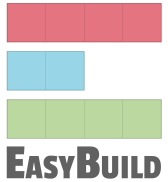
Webinar series: Different aspects of EESSI

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

Slides + recordings available



Lara

Agenda

VLAAMS SUPERCOMPUTER CENTRUM

- [09:00-10:00] Informal welcome
- [10:00-10:15] Introduction *(Kenneth Hoste - UGent)*
- [10:15-10:30] Central software stack + environment modules *(Kenneth Hoste - UGent)*
- [10:30-11:00] European Environment for Scientific Software Installations (EESSI) *(Lara Peeters - UGent)*
- [11:00-11:30] **Using Apptainer containers** *(Sam Moors - VUB)*
- [11:30-12:00] Conda / Mamba / Pixi *(Maxime Van den Bossche - KU Leuven)*
- [12:00-13:00] Lunch break

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

Create and run containers that package up pieces of software in a way that is portable and reproducible.

Overview

- What is a container, why Apptainer
- When to use containers, downsides of containers
- Performance, Trust/security
- Demo with tblite
- GPU-enabled containers
- Common issues with Docker images + solutions

Sam

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

What is a container

- Software package
- Contains all necessary elements to run software in any computing environment
- OS virtualization: shares the host OS's kernel, isolated user space
- Advantages: portable, secure, reproducible
- Container **image** = set of files needed to run the **container**

Sam

Using Apptainer containers

Why Apptainer (Singularity)

- Focus on integration into HPC clusters <> isolation (bind mounts, network, GPU devices)
- Doesn't require root 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

Sam

When to use containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

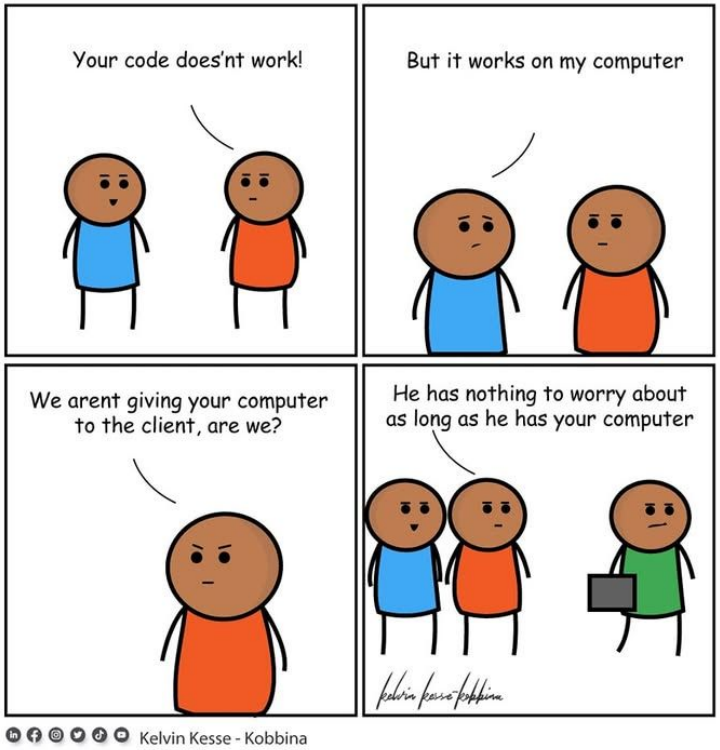
- **Try out**: Quickly testing a containerized app before performance-optimized installation
- **Portability**: built for a different Linux OS/version than the host OS
- **Reproducibility**: reproducing an env to run a workflow created by someone else
- Saving **inodes** (~number of files) on distributed file systems
- Avoiding **build issues** (HPC support staff):
  - Complex, custom build procedures
  - App too old to build with available software toolchains
  - Lots of dependencies with tight version restrictions that are incompatible with available toolchains (often conda software)

Sam

Using containers for portability

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER



Sam

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

Downsides of containers

- Interoperating with software outside the container (modules, Open OnDemand)
- Containers + MPI can be tricky
  - Compatible MPI libraries in host and container
- Portability: different CPU microarchitectures (x86-64 vs AArch64)
- Container images can be large (multi-GB)
    - Remove unnecessary files, packages after the build
    - Docker: multi-stage build
- Performance (see next slide)

Sam

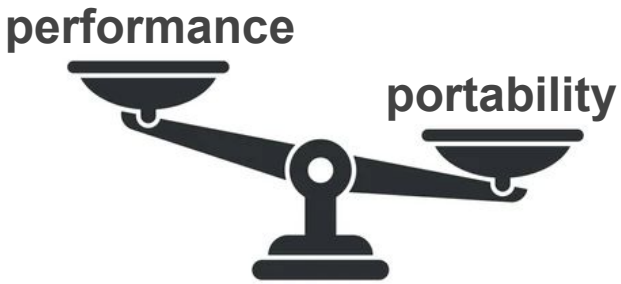
Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

Performance

- Images are usually built for **generic** CPU architecture rather than architecture-specific
- Sacrifices performance for portability



Sam

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

Trust & security

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


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

+ explicit bind mounts:

```
apptainer run --contain --bind <hostpath>[:<containerpath>[:ro]] \
<image.sif> <command>
```

- When importing images from Docker hub: **always pull from a verified publisher!**

Sam

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

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
```


Sam

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

Startup behavior

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

Sam

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

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
cd $VSC_SCRATCH
```


Sam

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

Pull 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...
```


Sam

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

Inspect Apptainer container

```
# inspect labels in Apptainer image
apptainer inspect ubuntu-24.04.sif
    org.label-schema.build-arch: amd64
    org.label-schema.build-date: Saturday_18_October_2025_12:57:14_CEST
    org.label-schema.schema-version: 1.0
    org.label-schema.usage.apptainer.version: 1.4.2-1.el9
    org.label-schema.usage.singularity.deffile.bootstrap: docker
    org.label-schema.usage.singularity.deffile.from: ubuntu:24.04
    org.opencontainers.image.ref.name: ubuntu
    org.opencontainers.image.version: 24.04
```


Sam

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

Inspect container startup

```
# inspect runscript
apptainer inspect --runscript ubuntu-24.04.sif
    …
    exec "$@"

# inspect environment
apptainer inspect --environment ubuntu-24.04.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
    …
```


Sam

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

Interactive shell inside container

```
# launch interactive shell inside container
apptainer shell ubuntu-24.04.sif

# check OS + version
Apptainer> cat /etc/lsb-release
    DISTRIB_ID=Ubuntu
    DISTRIB_RELEASE=24.04
    DISTRIB_CODENAME=noble
    DISTRIB_DESCRIPTION="Ubuntu 24.04.3 LTS"

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

# show all mountpoints
Apptainer> findmnt -no TARGET
```


Sam

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

Run command inside container

```
# run bash command inside container
apptainer run ubuntu-24.04.sif bash -c 'echo $APPTAINER_NAME'
    ubuntu-24.04.sif

# environment variables are automatically passed into container
# (exceptions: $PATH, $LD_LIBRARY_PATH)
export MY_VAR=my_value
apptainer run ubuntu-24.04.sif bash -c 'echo $MY_VAR'
    my_value

# set different value in container with APPTAINERENV_*** or --env
APPTAINERENV_MY_VAR=other_value apptainer run ubuntu-24.04.sif bash -c 'echo $MY_VAR'
apptainer run ubuntu-24.04.sif --env MY_VAR=other_value bash -c 'echo $MY_VAR'
```

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

Demo with tblite (1/2)

```
# inspecting/testing
apptainer inspect tblite-0.4.0.sif
apptainer inspect --environment tblite-0.4.0.sif
apptainer inspect --runscript tblite-0.4.0.sif
apptainer inspect --helpfile tblite-0.4.0.sif  # /.singularity.d/runscript.help
apptainer inspect --test tblite-0.4.0.sif  # /.singularity.d/test
apptainer run-help tblite-0.4.0.sif
apptainer test tblite-0.4.0.sif

# running
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
```


Sam

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

Demo with tblite (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)
```


Sam

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

GPU-enabled containers (docs)

- Bind mount GPU libraries into container:

```
# NVIDIA
apptainer run --nv
# AMD
apptainer run --rocm
```

- Container apps must match host GPU/driver's supported **CUDA (ROCm) version** and **CUDA compute capability**.
  - Use **prebuilt CUDA (ROCm) image** (NVIDIA container registry/AMD infinity hub)
  - 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**
- Multi-GPU: by default, all available GPUs on host are accessible in container

Sam

Using Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

Common issues with Docker images + solutions (docs)

- Container app wants to write into directory inside read-only container image
  - copy dir from image to host + bind mount
    ```
    apptainer run --bind host_dir:cont_dir
    ```
  - temporary overlay (discarded on exit)
    ```
    apptainer run --writable-tmpfs
    ```
- Interference from host environment (`$PYTHONPATH`, …)
  - run in clean environment
    ```
    apptainer run --cleanenv
    ```
- Interference with default bind mounts (`$HOME`, …)
  - disable default binds + add manual binds
    ```
    apptainer run --contain --bind some_dir
    ```
- Docker image was built to run processes as root inside the container
  - Use fakeroot
    ```
    apptainer run --fakeroot
    ```


Sam

Agenda

VLAAMS SUPERCOMPUTER CENTRUM

- [09:00-10:00] Informal welcome
- [10:00-10:15] Introduction *(Kenneth Hoste - UGent)*
- [10:15-10:30] Central software stack + environment modules *(Kenneth Hoste - UGent)*
- [10:30-11:00] European Environment for Scientific Software Installations (EESSI) *(Lara Peeters - UGent)*
- [11:00-11:30] Using Apptainer containers *(Sam Moors - VUB)*
- [11:30-12:00] **Conda / Mamba / Pixi** *(Maxime Van den Bossche - KU Leuven)*
- [12:00-13:00] Lunch break

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

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
- Example with other implementations (Miniconda, Micromamba and Pixi)
- Possible drawbacks and pitfalls
- Homework

Conda / Mamba / Pixi

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

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

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		

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

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

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

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

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Example: creating and using a Conda environment with Miniforge

CONDA

```
module load Miniforge3/25.3.0-3
# For local installers, see here

# 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

# 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
```

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Example: creating and using a Conda environment with Miniforge

CONDA

```
# Verifying the settings
conda info

      active environment : None
         user config file : /user/leuven/.../.condarc
  populated config files : /apps/leuven/.../software/Miniforge3/25.3.0-3/.condarc
                           /user/leuven/.../.condarc
       conda version : 25.3.0
     conda-build version : not installed
     python version : 3.12.10.final.0
                   solver : libmamba (default)
       virtual packages : __archspec=1=skylake_avx512
                          __conda=25.3.0=0
                          __cuda=13.3=0
                          __glibc=2.28=0
                          __linux=4.18.0=0
                          __unix=0=0
```

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Example: creating and using a Conda environment with Miniforge

CONDA

```
# Verifying the settings
conda info          (cont.)

      base environment : /apps/leuven/.../software/Miniforge3/25.3.0-3  (read only)
      conda av data dir : /apps/leuven/.../software/Miniforge3/25.3.0-3/etc/conda
  conda av metadata url : None
           channel URLs : https://conda.anaconda.org/conda-forge/linux-64
                          https://conda.anaconda.org/conda-forge/noarch
          package cache : /scratch/leuven/.../conda_pkgs
       envs directories : /data/leuven/.../conda_envs
                          /user/leuven/.../.conda/envs
                          /apps/leuven/.../software/Miniforge3/25.3.0-3/envs
          platform : linux-64
           user-agent : conda/25.3.0 requests/2.32.3 CPython/3.12.10
                        Linux/4.18.0-553.58.1.el8.patched.x86_64 rocky/8.10 glibc/2.28
                        solver/libmamba conda-libmamba-solver/25.3.0 libmambapy/2.1.1
```

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Example: creating and using a Conda environment with Miniforge

CONDA

```
# 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

source activate myfirstenv

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

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Example: creating and using a Conda environment with Miniforge

CONDA

```
# 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
```

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Example: creating and using a Conda environment with Miniforge

CONDA

```
# 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-pthreads_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
```

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Example: creating and using a Conda environment with Miniforge

CONDA

```
# 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)
```

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Example: creating and using a Conda environment with Miniforge

CONDA

```
# 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
```


Maxime

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Example: creating and using a Conda environment with Miniforge

CONDA

```
# 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
```

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Example: creating and using a Conda environment with Miniforge

CONDA

```
# Analyzing dependencies using conda-tree installed in a separate environment:
printf "dependencies:\n  - conda-tree=1.1.1\n" > environment.yaml
conda env create -n mysecondenv -f environment.yaml
source activate mysecondenv

conda-tree -n myfirstenv deptree
  ...
  tblite-python==0.4.0
    ├─ numpy 2.3.4 [required: any]
    │  ├─ python 3.13.9 [required: any]
    │  ├─ ...
    ├─ ...
    ├─ libblas 3.9.0 [required: >=3.9.0,<4.0a0]
    │  └─ mkl 2024.2.2 [required: >=2024.2.2,<2025.0a0]
    │  └─ ...
    ├─ liblapack 3.9.0 [required: >=3.9.0,<4.0a0]
    │  └─ libblas 3.9.0 [required: 3.9.0, 36_h5875eb1_mkl]
    └─ dftd4 3.7.0 [required: >=3.7.0,<3.8.0a0]
   ... └─ ...
```


Maxime

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Other implementations: Miniconda

miniCONDA

Similar to Miniforge but with

- potentially problematic ‘base’ environment
- *defaults* channel enabled by default
- more disk usage (if using a local installer)

tblite example using Miniconda:

```
wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh

bash Miniconda3-latest-Linux-x86_64.sh -b -p ${VSC_DATA}/miniconda3

# avoid overriding of your ~/.condarc
rm ${VSC_DATA}/miniconda3/.condarc

export PATH=${VSC_DATA}/miniconda3/bin:${PATH}
```

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Other implementations: Miniconda

miniCONDA

Similar to Miniforge but with

- potentially problematic ‘base’ environment
- *defaults* channel enabled by default
- more disk usage (if using a local installer)

tblite example using Miniconda:

```
conda create -n myfirstenv tblite-python=0.4.0

source activate myfirstenv

tblite --version

conda deactivate
```

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Other implementations: Micromamba

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

tblite example using Micromamba:

```
"${SHELL}" <(curl -L micro.mamba.pm/install.sh)
  Micromamba binary folder? [~/.local/bin] /path/to/your/VSC_DATA/bin/
  Init shell (bash)? [Y/n] n
  Configure conda-forge? [Y/n] n
  You can initialize your shell later by running:
    micromamba shell init  #

# Also here we would recommend to avoid "micromamba shell init"
# (which will permanently activate micromamba through your ~/.bashrc)
# and instead activate micromamba when you need it (tip: create an alias)
eval "$(${VSC_DATA}/bin/micromamba shell hook -s posix)"
```


Maxime

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Other implementations: Micromamba

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

tblite example using Micromamba:

```
# Now make sure envs_dirs and pkgs_dirs are set in your ~/.condarc
micromamba info

micromamba create -n myfirstenv tblite-python=0.4.0
micromamba activate myfirstenv
tblite --version
python -c 'from tblite.library import get_version; print(get_version())'
micromamba deactivate
```


Maxime

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Other implementations: Pixi

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

tblite example using Pixi:

```
curl -fsSL https://pixi.sh/install.sh | PIXI_HOME=/path/to/your/VSC_DATA/bin sh
  The 'pixi' binary is installed into '/path/to/your/VSC_DATA/bin'
  Updating '/user/leuven/.../.bashrc'

# If you don't want the installer to add "export PATH=/path/to/your/VSC_DATA/bin:$PATH"
# to your ~/.bashrc:
curl -fsSL https://pixi.sh/install.sh | PIXI_NO_PATH_UPDATE=y
PIXI_HOME=/path/to/your/VSC_DATA/bin sh
  The 'pixi' binary is installed into '/path/to/your/VSC_DATA/bin'
  No path update because PIXI_NO_PATH_UPDATE is set
```

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Other implementations: Pixi

tblite example using Pixi:

```
pixi init $VSC_DATA/pixi_envs/myfirstenv
  ✔ Created /data/leuven/.../pixi_envs/myfirstenv/pixi.toml

cd $VSC_DATA/pixi_envs/myfirstenv

pixi add tblite-python==0.4.0
  ✔ Added tblite-python==0.4.0

pixi list
  Package       Version    Build                    Size      Kind   Source
  ...
  liblapack       3.9.0    35_h47877c9_openblas   16.8 KiB     conda  https://.../conda-forge/
  libopenblas     0.3.30   pthreads_h94d23a6_2     5.7 MiB     conda  https://.../conda-forge/
  mctc-lib        0.4.2    h3b12eaf_0            451.5 KiB     conda  https://.../conda-forge/
  tblite          0.4.0    hd37a5e2_2              4.7 MiB     conda  https://.../conda-forge/
  tblite-python   0.4.0    py313h26530d8_1         104 KiB     conda  https://.../conda-forge/
```

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Other implementations: Pixi

tblite example using Pixi:

```
cat pixi.toml
  [workspace]
  authors = ["..."]
  channels = ["conda-forge"]
  name = "myfirstenv"
  platforms = ["linux-64"]
  version = "0.1.0"

  [tasks]

  [dependencies]
  tblite-python = "==0.4.0"

pixi run tblite --version
  tblite version 0.4.0

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


Maxime

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

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

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Possible drawbacks and pitfalls:

CONDA

- *'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
```

In Pixi you can do

```
pixi workspace export conda-explicit-spec .
```

but Pixi cannot yet use the resulting `default_linux-64_conda_spec.txt` file to create a new environment

Conda / Mamba / Pixi

VLAAMS SUPERCOMPUTER CENTRUM

Possible drawbacks and pitfalls:

CONDA

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

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

Pixi equivalent:

```
pixi workspace export conda-environment > env.yaml
pixi import -e mynewenv env.yaml
```

This can also be useful:

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

Pixi equivalent: see `pixi.toml`

Homework (Conda / Mamba / Pixi)

VLAAMS SUPERCOMPUTER CENTRUM

- Using one of the Conda implementations, create a new environment and install latest version of the eigensolver package `elpa` with Open MPI support (with all packages taken from the `conda-forge` channel).
- Execute `elpa2_print_kernels` and compare the output with what you get if you use the same command from a centrally installed module (e.g. `ELPA/2023.05.001-foss-2023a`).
- Explain why the Conda installation offers ELPA2 kernels with AVX2 but not with AVX512 instructions, even if `conda info` indicates your CPU would support it (tip: look at the feedstock).

Agenda

VLAAMS SUPERCOMPUTER CENTRUM

- [13:00-13:30] **Python: pip + venv** *(Steven Vandenbrande - KU Leuven)*
- [13:30-14:00] Other language package managers: R, Julia *(Alex Domingo - VUB)*
- [14:00-14:30] Software compilation *(Cintia Willemyns - VUB)*
- [14:30-15:00] Coffee break
- [15:00-15:30] Building Apptainer containers *(Sam Moors - VUB)*
- [15:30-16:00] EasyBuild + Spack *(Kenneth Hoste - UGent)*
- [16:00-16:30] Conclusions *(Kenneth Hoste - UGent)*
- [16:30-17:00] Q&A

Python's pip+venv

VLAAMS SUPERCOMPUTER CENTRUM

pip: pip install packages - popular Python package manager

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

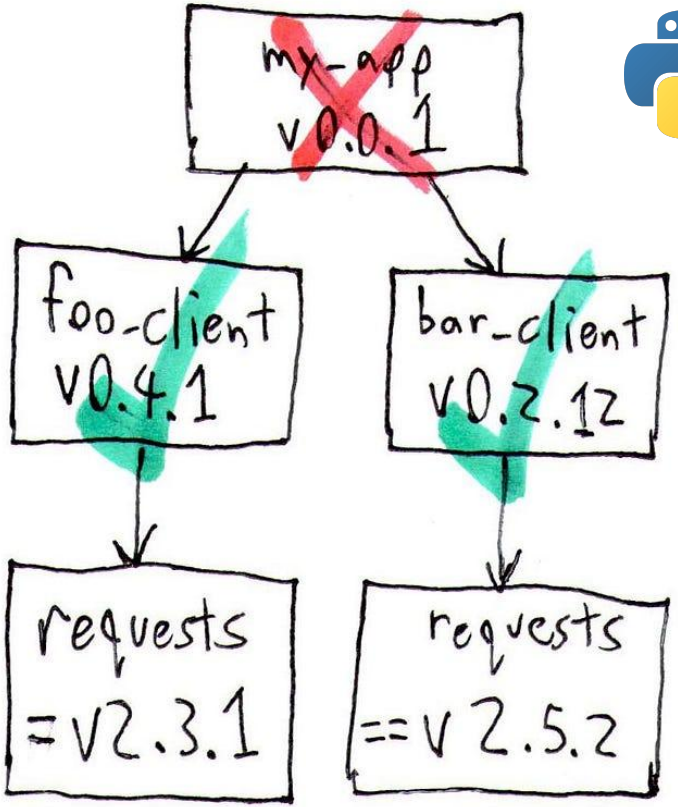
Overview

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

pip for resolving dependencies

VLAAMS SUPERCOMPUTER CENTRUM

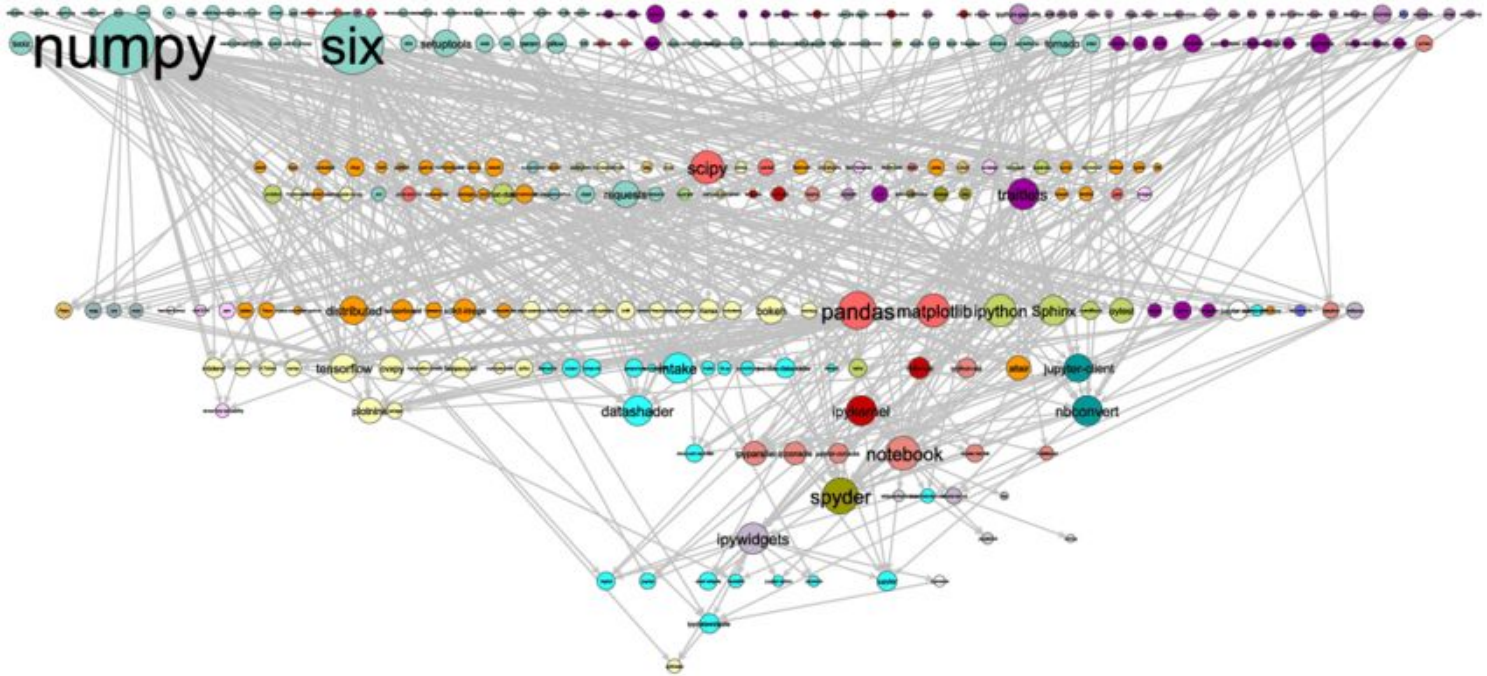
- To install a Python package, you typically need to install additional packages as dependencies
- Transitive dependencies quickly lead to a complicated dependency graph
- Use a package manager (like pip) to automatically resolve dependencies



Steven

pip for resolving dependencies

VLAAMS SUPERCOMPUTER CENTRUM

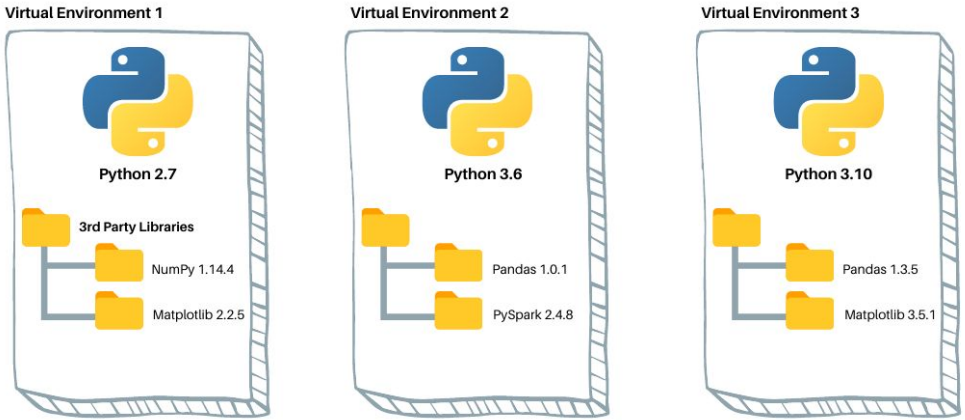


Steven

venv for isolating environments

VLAAMS SUPERCOMPUTER CENTRUM

- 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



Steven

Comparison with other approaches

VLAAMS SUPERCOMPUTER CENTRUM

	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

VLAAMS SUPERCOMPUTER CENTRUM

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

Steven

Demonstration of basic pip+venv

VLAAMS SUPERCOMPUTER CENTRUM

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
```


Steven

Demonstration of basic pip+venv

VLAAMS SUPERCOMPUTER CENTRUM

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

VLAAMS SUPERCOMPUTER CENTRUM

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`

Steven

Demonstration of basic pip+venv: remarks

VLAAMS SUPERCOMPUTER CENTRUM

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

VLAAMS SUPERCOMPUTER CENTRUM

Listing 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

VLAAMS SUPERCOMPUTER CENTRUM

Listing 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

VLAAMS SUPERCOMPUTER CENTRUM

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

VLAAMS SUPERCOMPUTER CENTRUM

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

Steven

vsc-venv

VLAAMS SUPERCOMPUTER CENTRUM

pip+venv can be perfectly combined with centrally installed Python 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
  ⇒ **solved by vsc-venv**
  (see https://docs.hpc.ugent.be/setting_up_python_virtual_environments/)

vsc-venv

VLAAMS SUPERCOMPUTER CENTRUM

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!

Steven

vsc-venv

VLAAMS SUPERCOMPUTER CENTRUM

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

VLAAMS SUPERCOMPUTER CENTRUM

pip: pip install packages - popular Python package manager

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

Overview

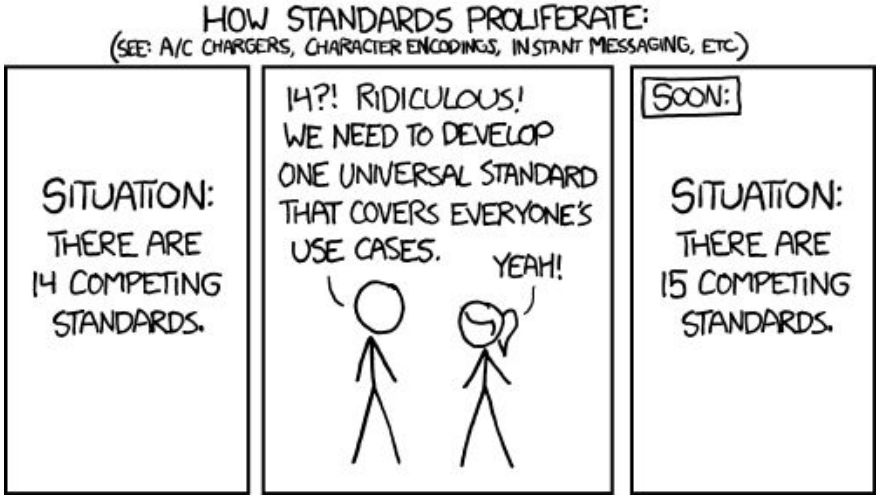
- Motivation and context
- How to use pip+venv
- How to use vsc-venv
- **Alternatives to pip+venv**

Alternatives to pip+venv

VLAAMS SUPERCOMPUTER CENTRUM

Many (too many?) Python package managers available out there:

- Look around, then choose one and get comfortable with it
- There are objective and subjective reasons for preference
- Traction matters (available packages, online support)



Steven

Alternatives to pip+venv

VLAAMS SUPERCOMPUTER CENTRUM

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

Homework (pip + venv)

VLAAMS SUPERCOMPUTER CENTRUM

- Choose a Python package, preferably with several dependencies
- Install it with pip in a venv:
  - Once with the system Python interpreter
  - Once using as many modules as possible to satisfy dependencies (for instance making use of vsc-venv)
  - [Optional] Do the same in a conda environment (see earlier talk)
- Compare the different approaches:
  - What are the storage requirements (disk space and number of files)?
  - Is there a difference in performance of the installations?

Agenda

VLAAMS SUPERCOMPUTER CENTRUM

- [13:00-13:30] Python: pip + venv *(Steven Vandenbrande - KU Leuven)*
- [13:30-14:00] **Other language package managers: R, Julia** *(Alex Domingo - VUB)*
- [14:00-14:30] Software compilation *(Cintia Willemyns - VUB)*
- [14:30-15:00] Coffee break
- [15:00-15:30] Building Apptainer containers *(Sam Moors - VUB)*
- [15:30-16:00] EasyBuild + Spack *(Kenneth Hoste - UGent)*
- [16:00-16:30] Conclusions *(Kenneth Hoste - UGent)*
- [16:30-17:00] Q&A

Language package managers

VLAAMS SUPERCOMPUTER CENTRUM

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 (*built-in*)	*decentralized*
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 (*built-in*)	CRAN

** most common option but others may exists*

Alex

Language package managers: R

VLAAMS SUPERCOMPUTER CENTRUM

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

VLAAMS SUPERCOMPUTER CENTRUM

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

Language package managers: R

VLAAMS SUPERCOMPUTER CENTRUM

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")
```


Alex

Language package managers: R

VLAAMS SUPERCOMPUTER CENTRUM

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 `$R_LIBS_HOME`. By default located under `$R_HOME/library`.
2. Site libraries added by module files. Read-only.
3. Main R library of the active R interpreter. Read-only.

Alex

Language package managers: R

VLAAMS SUPERCOMPUTER CENTRUM

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 tight 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.

Language package managers: R

VLAAMS SUPERCOMPUTER CENTRUM

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

Alex

Language package managers: R

VLAAMS SUPERCOMPUTER CENTRUM

Demo with 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"
  [INFO] Loading modules from 'my-rproject-modules.txt'
  [INFO]   ✔ Module 'R/4.4.2-gfbf-2024a' loaded successfully
  [INFO]   ✔ Module 'R-bundle-Bioconductor/3.20-foss-2024a-R-4.4.2' loaded successfully
  [INFO] Modules loaded successfully
  [INFO] Using R/4.4.2-gfbf-2024a to create vsc-Rproject environment at
"/path/to/Rprojects/MyProject"
  [INFO] Creating $VSC_RPROJECT directory
  [INFO] Storing modules.env
  ...
```

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

Alex

Language package managers: R

VLAAMS SUPERCOMPUTER CENTRUM

Demo with vsc-Rproject

```
# 3. Create project
vsc-rproject create MyProject --modules="$VSC_DATA/modules.txt"
  ...
  [INFO] Creating $VSC_RPROJECT directory
  [INFO] Storing modules.env
  [INFO] Creating MyProject.Rproj
  [INFO] Creating .Renviron
  [INFO] Creating .Rprofile
  [INFO] Creating Makevars
  vsc-Rproject environment setup complete.
  ...

# 4. Activate Project
vsc-rproject activate MyProject
  [INFO] Loading modules from 'path/to/Rprojects/MyProject/.vsc-rproject/modules.env'
  [INFO]   ✔ Module 'R/4.4.2-gfbf-2024a' loaded successfully
  [INFO]   ✔ Module 'R-bundle-Bioconductor/3.20-foss-2024a-R-4.4.2' loaded successfully
  [INFO] MyProject activated
```

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

Alex

Language package managers: R

VLAAMS SUPERCOMPUTER CENTRUM

Demo with vsc-Rproject

```
# 5. Use 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-gfbf-2024a/lib64/R/library
R>
```


Alex

Language package managers: R

VLAAMS SUPERCOMPUTER CENTRUM

Demo with vsc-Rproject

```
# 6. Install new package in Rproject
R> BiocManager::install("BgeeDB")
  'getOption("repos")' replaces Bioconductor standard repositories, see
  'help("repositories", package = "BiocManager")' for details.
  Replacement repositories:
      CRAN: https://cloud.r-project.org
  Bioconductor version 3.20 (BiocManager 1.30.25), R 4.4.2 (2024-10-31)
  Installing package(s) 'BgeeDB'
  also installing the dependencies ‘reticulate’, ‘topGO’, ‘anndata’,
  ‘bread’

  trying URL 'https://cloud.r-project.org/…/reticulate_1.44.0.tar.gz'
  Content type 'application/x-gzip' length 1705757 bytes (1.6 MB)
  ==================================================
  downloaded 1.6 MB
  [...]
```

BgeeDB is a small package for the annotation and gene expression data download from Bgee database

BiocManager is the package manager of **Bioconductor** and will install the version of *BgeeDB* that corresponds to the active version of Bioconductor and any other missing dependency in your project:

- topGO v2.58.0
- reticulate v1.44.0
- anndata v0.8.0
- bread v0.4.1
- BgeeDB v2.32.0

Alex

Language package managers: R

VLAAMS SUPERCOMPUTER CENTRUM

Demo with vsc-Rproject

```
# 6. Install new package in Rproject
R> BiocManager::install("BgeeDB")
  [...]
  * installing *source* package 'reticulate' ...
  ** package 'reticulate' successfully unpacked and MD5 sums checked
  ** using staged installation
  ** libs
  using C++ compiler: 'g++ (GCC) 13.3.0'
  g++ -std=gnu++17 -DNDEBUG -fpic -O2 -ftree-vectorize
  -march=native -fno-math-errno -g  -march=x86-64-v4
  -c RcppExports.cpp -o RcppExports.o
  [...]
```

vsc-Rproject automatically configures your Rproject environment to use the same compilers and libraries used by your installation of R and guarantees that any compiled code will be portable.

Alex

Language package managers: R

VLAAMS SUPERCOMPUTER CENTRUM

Demo with vsc-Rproject

```
# 6. Install new package in Rproject
R> BiocManager::install("BgeeDB")
  [...]
  * installing *source* package ‘BgeeDB’ ...
  ** package ‘BgeeDB’ successfully unpacked and MD5 sums checked
  ** using staged installation
  ** R
  ** data
  ** inst
  ** byte-compile and prepare package for lazy loading
  [...]
  groupGOTerms:    GOBPTerm, GOMFTerm, GOCCTerm environments built.
  ** testing if installed package keeps a record of temporary installation path
  * DONE (BgeeDB)
  [...]
R>
```

Language package managers: R

VLAAMS SUPERCOMPUTER CENTRUM

Demo with vsc-Rproject

```
# 7. Use new package in Rproject
R> library(BgeeDB)
  Loading required package: topGO
  Loading required package: BiocGenerics
  Attaching package: 'BiocGenerics'

  The following objects are masked from 'package:stats':
      IQR, mad, sd, var, xtabs
  [...]

R> listBgeeSpecies()
  Querying Bgee to get release information...
  Building URL to query species in Bgee release 15_2...
  downloading Bgee species info... (https://www.bgee.org/…)

  Download of species information successful!
        ID           GENUS     SPECIES_NAME              COMMON_NAME AFFYMETRIX
  1   6239  Caenorhabditis          elegans                 nematode       TRUE
```


Alex

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)

Language package managers: Julia

VLAAMS SUPERCOMPUTER CENTRUM

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")
    Resolving package versions...
```


Alex

Language package managers: Julia

VLAAMS SUPERCOMPUTER CENTRUM

How to install your own Julia packages?

4. Start using the new software

```
julia> using TightBindingToolkit
```

Note: 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

VLAAMS SUPERCOMPUTER CENTRUM

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.**

Alex

Homework: R and Julia

VLAAMS SUPERCOMPUTER CENTRUM

- **R**: The demo resulted in the installation of BgeeDB v2.32.0, how would you install another version of BgeeDB?
- **R**: Make a new Rproject and install **nichenetr** from their repository in GitHub https://github.com/saeyslab/nichenetr
- **Julia**: Add the package **Plots.jl** to the project made with TightBindingToolkit https://juliapackages.com/p/plots

Alex

Agenda

VLAAMS SUPERCOMPUTER CENTRUM

- [13:00-13:30] Python: pip + venv *(Steven Vandenbrande - KU Leuven)*
- [13:30-14:00] Other language package managers: R, Julia *(Alex Domingo - VUB)*
- [14:00-14:30] **Software compilation** *(Cintia Willemyns - VUB)*
- [14:30-15:00] Coffee break
- [15:00-15:30] Building Apptainer containers *(Sam Moors - VUB)*
- [15:30-16:00] EasyBuild + Spack *(Kenneth Hoste - UGent)*
- [16:00-16:30] Conclusions *(Kenneth Hoste - UGent)*
- [16:30-17:00] Q&A

Software compilation

VLAAMS SUPERCOMPUTER CENTRUM

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

Language	Compiler	Package Manager*
Fortran	- GNU (`gfortran`) - Intel (`ifort/ifx`) - 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`)	*built-in*
Zig	- Official (`zig build-exe`)	*built-in decentralized*

** (Most common options, others may exist)*

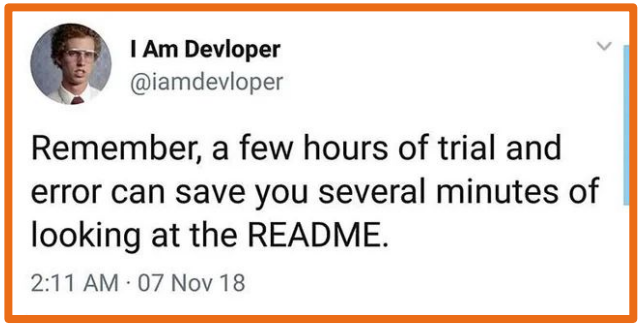
Cintia

Software compilation: Before you start

VLAAMS SUPERCOMPUTER CENTRUM

- 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.

- If possible **load 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

Cintia

Software compilation: Rust

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/1.83.0-GCCcore-13.3.0
```

2. Get your source code

```
git clone https://github.com/bootandy/dust.git
  Cloning into 'dust'...
  remote: Enumerating objects: 3169, done.
  remote: Counting objects: 100% (869/869), done.
  remote: Compressing objects: 100% (322/322), done.
  [...]
cd dust
git switch --detach v0.9.0
  HEAD is now at fead40b Increment version
```

Software compilation: Rust

VLAAMS SUPERCOMPUTER CENTRUM

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

```
cd dust
ls
  build.rs          completions       media
  Cargo.lockconfig      README.md
  Cargo.tomlLICENSE           src
  ci                man-page          tests
```

- **Cargo.toml**: configuration file that contains **metadata** about a Rust package, including its name, version, authors, and dependencies.
- **Cargo.lock**: used to ensure reproducible builds by recording the **exact versions of dependencies** used in a project

Recommended: Generate `Cargo.lock` if missing

```
cargo generate-lockfile
   Updating crates.io index
   Locking 107 packages to latest compatible versions
   Adding directories v4.0.1 (available: v6.0.0)
   Adding lscolors v0.13.0 (available: v0.20.0)
   Adding sysinfo v0.27.8 (available: v0.36.1)
   Adding terminal_size v0.2.6 (available: v0.4.3)
   Adding unicode-width v0.1.14 (available: v0.2.2)
```

Software compilation: Rust

VLAAMS SUPERCOMPUTER CENTRUM

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 proc-macro2 v1.0.101
   Compiling unicode-ident v1.0.19
   Compiling quote v1.0.41
   Compiling libc v0.2.177
      Building [=========================> ] 5/89: utf8parse
```

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

You can run your program by executing the binary

```
./target/release/dust $VSC_SCRATCH/dust/tests/test_dir

  0B      ┌── a_file     |                                |   0%
512B      ├── hello_file |                                |  20%
1.5K    ┌─┴ many         |                                |  60%
2.5K  ┌─┴ test_dir       |                                | 100%
```


Cintia

Software compilation: Rust

VLAAMS SUPERCOMPUTER CENTRUM

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
dust --version
  -bash: dust: command not found
which dust
  /usr/bin/which: no dust in
  (/bin:/sbin:/usr/local/sbin:/usr/sbin)

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

# Binary is now in PATH
which dust
  $PWD/target/release/dust
dust --version
  Dust 0.9.0
```

Software compilation: Rust

VLAAMS SUPERCOMPUTER CENTRUM

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 and 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
```

Software compilation: Go

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 module of Go

```
module load Go/1.23.6
```

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 v0.1.4
  HEAD is now at af17706 Add Work.RequestFunc (#149)
```

Software compilation: Go

VLAAMS SUPERCOMPUTER CENTRUM

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

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

Recommended: Generate `go.sum` if missing

```
go mod tidy
```

- `go.mod` → specifies dependencies and Go module version
- `go.sum` → pins exact versions

Cintia

Software compilation: Go

VLAAMS SUPERCOMPUTER CENTRUM

How to install your own Go packages?

4. Build your package

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

5. Locate the new binary, it will be placed inside the path given by `-o` option (e.g. `bin`)

   You can run your program by executing the binary

```
# Add executable to PATH and run
export PATH=$PWD/bin:$PATH
hey -n 100 https://example.com
  Summary:
    Total:  0.7478 secs
    Slowest:0.5948 secs
    Fastest:0.1475 secs
    Average:0.3593 secs
    Requests/sec: 133.7191

    Total data:   7182 bytes
    Size/request: 71 bytes
    ■■■■■■■■■■■■■■ | 100%
```


Cintia

Software compilation: Go

VLAAMS SUPERCOMPUTER CENTRUM

Notes on Installing Go Packages

- CPU architecture

```
go build
```

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

No micro-architecture optimization is possible.

- 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`

Cintia

Manual software compilation

VLAAMS SUPERCOMPUTER CENTRUM

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*

Cintia

Manual software compilation: make

VLAAMS SUPERCOMPUTER CENTRUM

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

Manual software compilation: CMake

VLAAMS SUPERCOMPUTER CENTRUM

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 (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 be created in build-dir
ls
  functions.c    functions.o
  Makefile       program
  program.c      program.o
```


Cintia

Manual software compilation

VLAAMS SUPERCOMPUTER CENTRUM

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	`./configure` *(needs configure script)*	`make`	`make install`	`make`
build.ninja	*(None)*	`ninja`	`ninja install`	`Ninja`
CMakeLists.txt	`cmake /` `cmake -G Ninja`	`make /` `ninja`	`make install /` `ninja install`	`Cmake +` `make/Ninja`
SConstruct	`scons`		`scons install`	`Scons *`
meson.build	`meson setup builddir`	`meson compile -C builddir`	`meson install -C builddir`	`Meson`
cargo.toml	`cargo build --release`		`cargo install`	`Rust`
go.mod	`go build`		`go install`	`Go`
setup.py requirements.txt pyproject.toml	`pip install`			`Python`

* Not available on all VSC clusters

Cintia

Manual software compilation

VLAAMS SUPERCOMPUTER CENTRUM

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.

Manual software compilation

VLAAMS SUPERCOMPUTER CENTRUM

- **Live demonstration with tblite**

```
# Get the source code
git clone https://github.com/tblite/tblite.git
cd tblite
git switch --detach v0.5.0
# Inspect directory
ls
# Load needed modules
module load buildenv/default-foss-2024a Ninja/1.12.1-GCCcore-13.3.0 CMake/3.29.3-GCCcore-13.3.0
# Choose a directory and configure a new build with
cmake -B _build -G Ninja -DCMAKE_INSTALL_PREFIX=$VSC_SCRATCH/my_software/tblite
# Inspect build directory
ls _build/
# Compile
ninja -C _build
# Install (copy built files: binaries, libs, headers,... into the install prefix)
ninja -C _build install
# Add binary to PATH
export PATH=$VSC_SCRATCH/my_software/tblite/bin/:$PATH
# Run the program
tblite run --method gfn2 $VSC_SCRATCH/tblite_test/h2.xyz
```

TB lite

Documentation available: https://tblite.readthedocs.io

Cintia

Agenda

VLAAMS SUPERCOMPUTER CENTRUM

- [13:00-13:30] Python: pip + venv *(Steven Vandenbrande - KU Leuven)*
- [13:30-14:00] Other language package managers: R, Julia *(Alex Domingo - VUB)*
- [14:00-14:30] Software compilation *(Cintia Willemyns - VUB)*
- [14:30-15:00] Coffee break
- [15:00-15:30] **Building Apptainer containers** *(Sam Moors - VUB)*
- [15:30-16:00] EasyBuild + Spack *(Kenneth Hoste - UGent)*
- [16:00-16:30] Conclusions *(Kenneth Hoste - UGent)*
- [16:30-17:00] Q&A

Building Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

Overview

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

Sam

Building Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

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
cd $VSC_SCRATCH
```

Building Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

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
```


Sam

Building Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

From Apptainer definition file (recommended for reproducibility)

```
# 1) write Apptainer definition file tblite-0.4.0.def (docs)
# file available in gssi-training repo
cat tblite-0.4.0.def
    Bootstrap: docker
    From: ubuntu:24.04
    %post ...

# 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
```


Sam

Building Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

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** (+ root permissions)
or **Podman** (Windows: WSL).
Will not work in your VSC account!

Sam

Building Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

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
```


Sam

Building Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

Updating containers built with hpc-container-wrapper

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


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


Sam

Building Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

GPU-enabled containers (docs)

- 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**
  - 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**

Sam

Building Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

MPI-enabled containers (docs)

- Hybrid model:
  - MPI library **installed in container**
  - MPI library inside container and on host must be compatible

```
mpirun -n $SLURM_NTASKS apptainer exec image.sif executable
```

- Bind model:
  - MPI library not installed in container, but **bind mounted into container**
  - MPI library used to compile application in container and on host must be compatible

```
mpirun -n $SLURM_NTASKS apptainer exec --bind "$MPI_DIR" image.sif executable
```


Sam

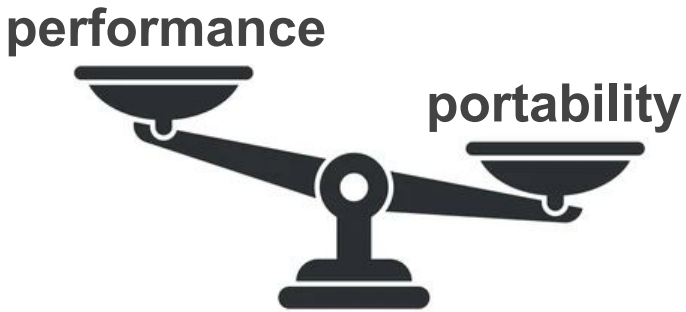
Building Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

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



Sam

Podman in HPC?

VLAAMS SUPERCOMPUTER CENTRUM

	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

Sam

Homework (building Apptainer containers)

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

Build an Apptainer container for ELPA using your method of choice:

- from an Apptainer definition file
- with hpc-container-wrapper
- via a Docker image created from a Dockerfile

ELPA can be installed in the container in different ways:

- with Conda (requires installing Conda in the container)
- with Easybuild (requires installing Easybuild in the container)
- manually from the ELPA sources on Github

Modify **environment** and **runscript** files to ensure that this cmd runs successfully:

```
apptainer run <image.sif> elpa2_print_kernels
```


Sam

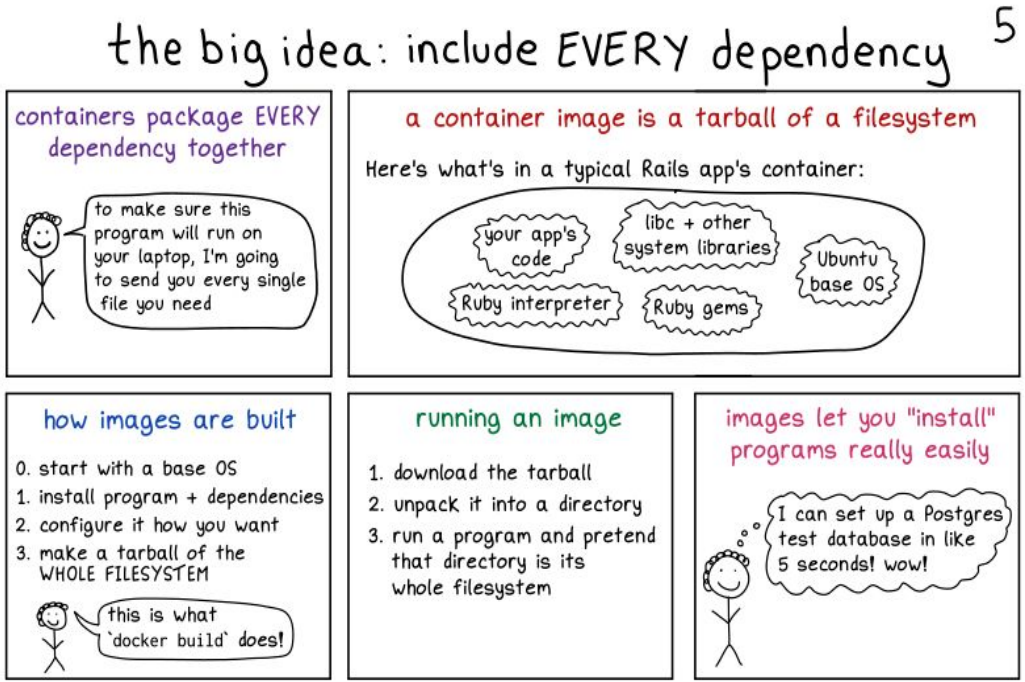
Building Apptainer containers

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER

More documentation

- VSC Apptainer docs
- VSC training: Containers for HPC
- Official Apptainer docs



Sam

Agenda

VLAAMS SUPERCOMPUTER CENTRUM

- [13:00-13:30] Python: pip + venv *(Steven Vandenbrande - KU Leuven)*
- [13:30-14:00] Other language package managers: R, Julia *(Alex Domingo - VUB)*
- [14:00-14:30] Software compilation *(Cintia Willemyns - VUB)*
- [14:30-15:00] Coffee break
- [15:00-15:30] Building Apptainer containers *(Sam Moors - VUB)*
- [15:30-16:00] **EasyBuild + Spack** *(Kenneth Hoste - UGent)*
- [16:00-16:30] Conclusions *(Kenneth Hoste - UGent)*
- [16:30-17:00] Q&A

EasyBuild + Spack

VLAAMS SUPERCOMPUTER CENTRUM

- 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`, etc.
- **Tools** have been developed to **automate software installation** (from source)
  - EasyBuild and Spack are most prominent (on HPCs systems)

Kenneth

EasyBuild in a nutshell

VLAAMS SUPERCOMPUTER CENTRUM

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
- 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**

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`

Kenneth

Spack in a nutshell

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 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

- "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`

Kenneth

Basic usage of Spack

- 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`

Kenneth

EasyBuild vs Spack: similarities

- Implemented in **Python**
- Open source software
- 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

VLAAMS SUPERCOMPUTER CENTRUM

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

- Community is mostly **US-based**
- Software 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
- …

Kenneth

When (not) to use  or  Spack

VLAAMS SUPERCOMPUTER CENTRUM 

- **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
- Install 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)
- **Steep initial learning** curve for both
  - Some experience with compilers & manually installing software can be helpful
  - Take some time to learn how to use the tool you picked (docs, tutorial, …)

Agenda

VLAAMS SUPERCOMPUTER CENTRUM

- [13:00-13:30] Python: pip + venv *(Steven Vandenbrande - KU Leuven)*
- [13:30-14:00] Other language package managers: R, Julia *(Alex Domingo - VUB)*
- [14:00-14:30] Software compilation *(Cintia Willemyns - VUB)*
- [14:30-15:00] Coffee break
- [15:00-15:30] Building Apptainer containers *(Sam Moors - VUB)*
- [15:30-16:00] EasyBuild + Spack *(Kenneth Hoste - UGent)*
- [16:00-16:30] **Conclusions** *(Kenneth Hoste - UGent)*
- [16:30-17:00] Q&A

So… which tool should I use?

VLAAMS SUPERCOMPUTER CENTRUM

- One single definitive answer:

So… which tool should I use?

VLAAMS SUPERCOMPUTER CENTRUM

- 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
  - To what extent the **performance** of the software 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

VLAAMS SUPERCOMPUTER CENTRUM

	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)								

Kenneth

“Totally unbiased” high-level comparison

VLAAMS SUPERCOMPUTER CENTRUM

/apps	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)	👍👍👍							

Kenneth

“Totally unbiased” high-level comparison

VLAAMS SUPERCOMPUTER CENTRUM

	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)	👍👍👍	👍👍						

Kenneth

“Totally unbiased” high-level comparison

VLAAMS SUPERCOMPUTER CENTRUM

APPTAINER	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)	👍👍👍	👍👍	👍👍					

Kenneth

“Totally unbiased” high-level comparison

VLAAMS SUPERCOMPUTER CENTRUM

CONDA	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)	👍👍👍	👍👍	👍👍	👍(👍)				

Kenneth

“Totally unbiased” high-level comparison

VLAAMS SUPERCOMPUTER CENTRUM

	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)	👍👍👍	👍👍	👍👍	👍(👍)	👍👍			

Kenneth

“Totally unbiased” high-level comparison

VLAAMS SUPERCOMPUTER CENTRUM

GCC	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)	👍👍👍	👍👍	👍👍	👍(👍)	👍👍	👍		

Kenneth

“Totally unbiased” high-level comparison

VLAAMS SUPERCOMPUTER CENTRUM

EasyBuild / 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)	👍👍👍	👍👍	👍👍	👍(👍)	👍👍	👍	👍👍👍	👍

Kenneth

Conclusions

VLAAMS SUPERCOMPUTER CENTRUM

- 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 sessions on Thu 6 Nov 2025

VLAAMS SUPERCOMPUTER CENTRUM

- KU Leuven: https://icts.kuleuven.be/apps/onebutton/registrations/1228042
- UAntwerpen: confirm your attendance via e-mail to hpc@uantwerpen.be
- UGent:
  - Location: UGent campus Sterre, building S9, Multimediazaal
  - Time slots: 10:00-12:00, 13:00-15:00, 15:00-17:00 CET
  - Registration: https://event.ugent.be/registration/gssihandson
- VUB:
  - Location: Sablon room, Pleinlaan 9, 5th floor, Main Campus Etterbeek
  - Time: 13:00 to 16:00 CET
  - Registration: via e-mail to hpc@vub.be (+ indicate time of arrival)

Talk to us or contact compute@vscentrum.be for more information