GOBIERNO DE ESPAÑA MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES

Instituto de Salud Carlos III

Software Management and Containers in HPC

Pablo Mata Aroco - BU-ISCIII

29-04 de octubre de 2025

1ª edición

Overview

1. Software management basics.
2. Software managers: EasyBuild, Conda-Micromamba
3. Introduction to Virtualization. Containers: Docker & Singularity

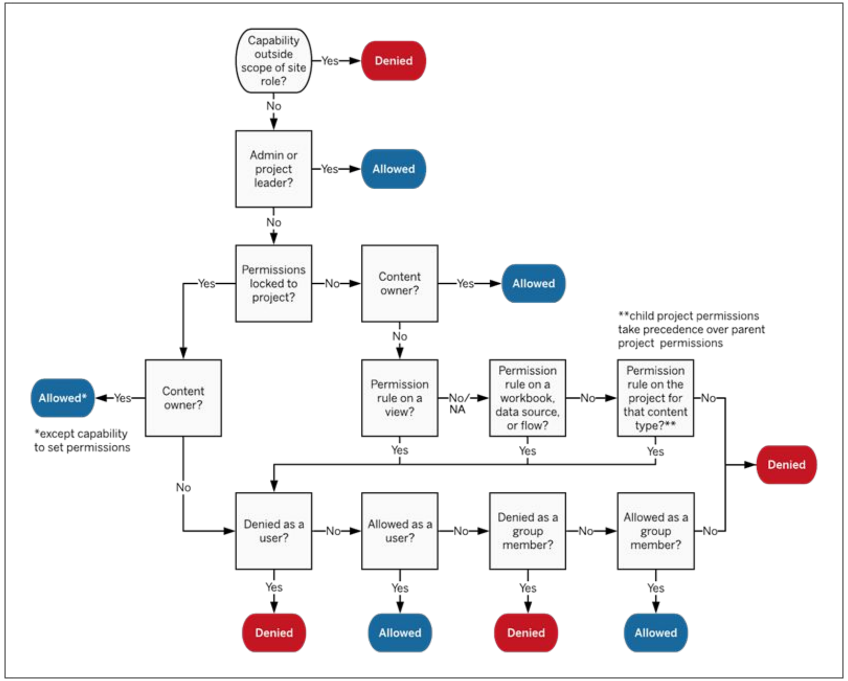
Overview

1. Software management basics
2. Software managers: EasyBuild, Conda-Micromamba
3. Introduction to Virtualization. Containers: Docker & Singularity

1. Software management basics: Permissions

Permissions are rules that control what a user can do on a computer system. They determine:

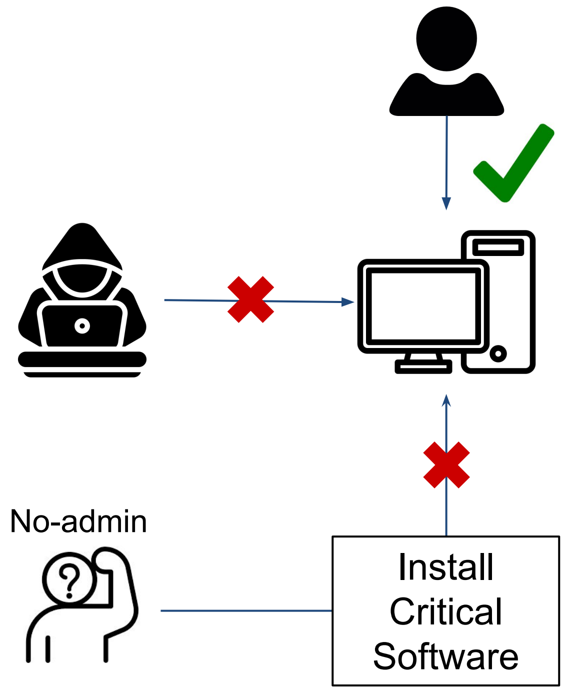
- Who can read (view a file or program).
- Who can write (modify or delete).
- Who can execute (run a program).


Capability outside scope of site role?
Yes
Denied
No
Admin or project leader?
Yes
Allowed
No
Permissions locked to project?
Yes
No
Content owner?
Yes
Allowed
No
Content owner?
Yes
Allowed*
*except capability to set permissions
No
**child project permissions take precedence over parent project permissions
Permission rule on a view?
No/NA
Permission rule on a workbook, data source, or flow?
No
Permission rule on the project for that content type?**
No
Denied
Yes
Yes
Yes
Denied as a user?
No
Allowed as a user?
No
Denied as a group member?
No
Allowed as a group member?
No
Yes
Denied
Yes
Allowed
Yes
Denied
Yes
Allowed

1. Software management basics: Permissions

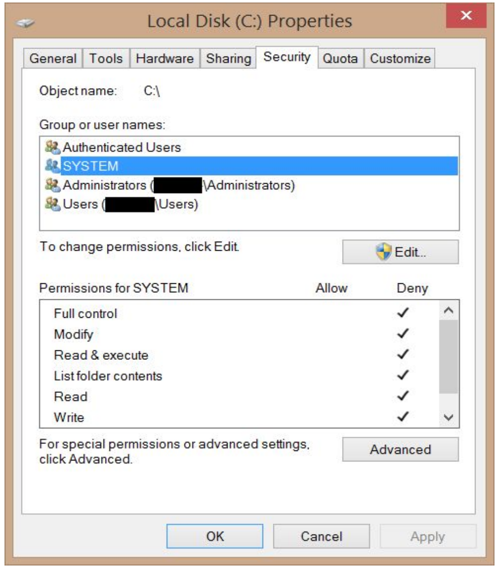
They exist to **protect** the operating system, applications, and data from accidental damage or malicious use:

- **Security**: Prevent malware or mistakes from harming other users.
- **Stability**: Prevent one user’s bad installation from breaking shared applications.
- **Consistency**: Admins control which versions of software are installed cluster-wide.
- **Performance**: HPC software often needs to be built with optimized compilers and libraries.



1. Software management basics: Windows vs Linux

Windows



Local Disk (C:) Properties

General | Tools | Hardware | Sharing | Security | Quota | Customize

Object name: C:\

Group or user names:

- Authenticated Users
- SYSTEM
- Administrators ([redacted]\Administrators)
- Users ([redacted]\Users)

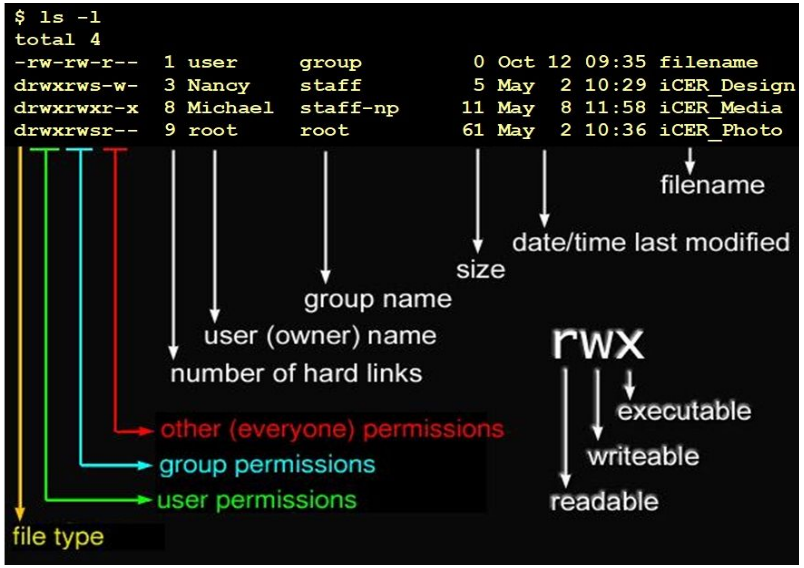
To change permissions, click Edit. Edit...

Permissions for SYSTEM	Allow	Deny
Full control		✓
Modify		✓
Read & execute		✓
List folder contents		✓
Read		✓
Write		✓

For special permissions or advanced settings, click Advanced. Advanced

OK Cancel Apply

Linux

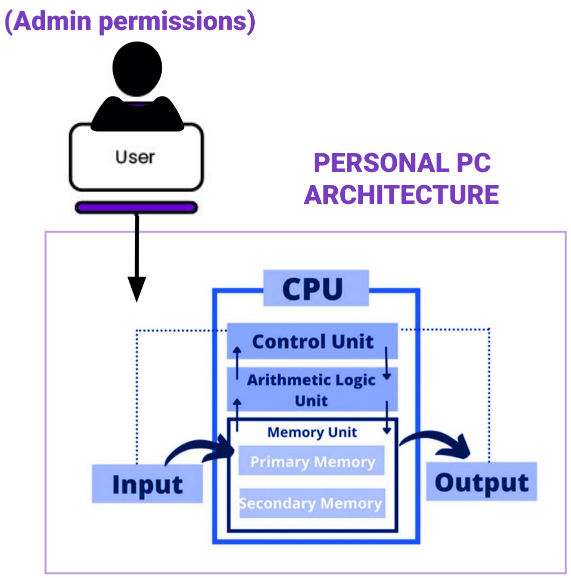


```
$ ls -l
total 4
-rw-rw-r--  1 user     group         0 Oct 12 09:35 filename
drwxrws-w-  3 Nancy    staff         5 May  2 10:29 iCER_Design
drwxrwxr-x  8 Michael  staff-np     11 May  8 11:58 iCER_Media
drwxrwsr--  9 root     root         61 May  2 10:36 iCER_Photo
```

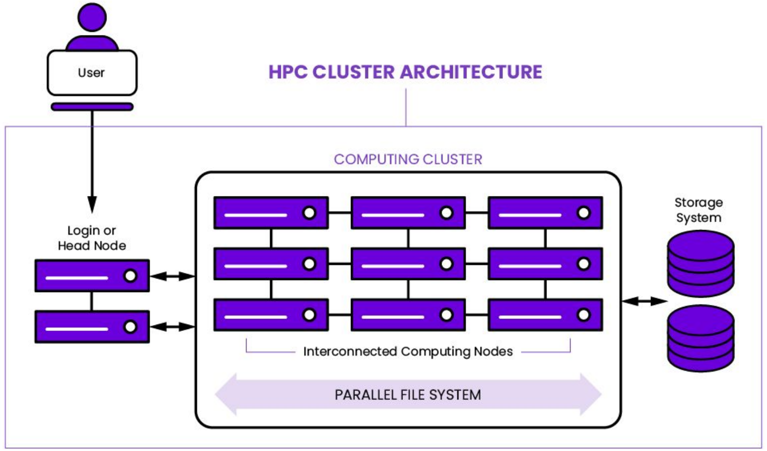
1. Software management: HPC vs Personal PC

On a **personal computer** (for example, your Windows laptop) installing software is straightforward:

- You open a browser, download an **installer** (`.exe` or `.msi`), run it, and the program installs locally.
- **You usually have administrator rights** (even if you don't notice), so you can change almost any system configuration.
- **You work on a single machine**, with a fixed operating system and hardware (CPU, RAM, graphics card…).



1. Software management: HPC vs Personal PC

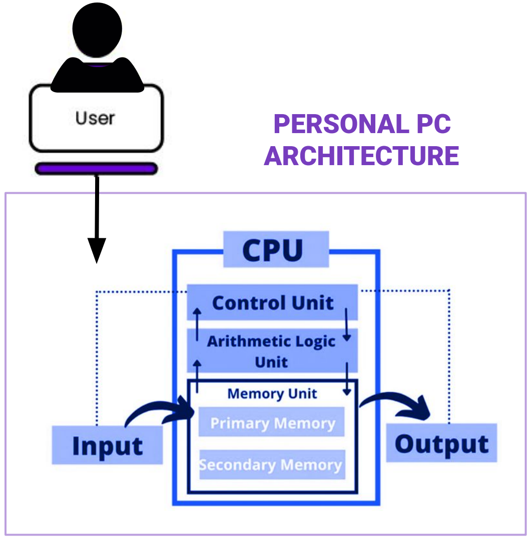


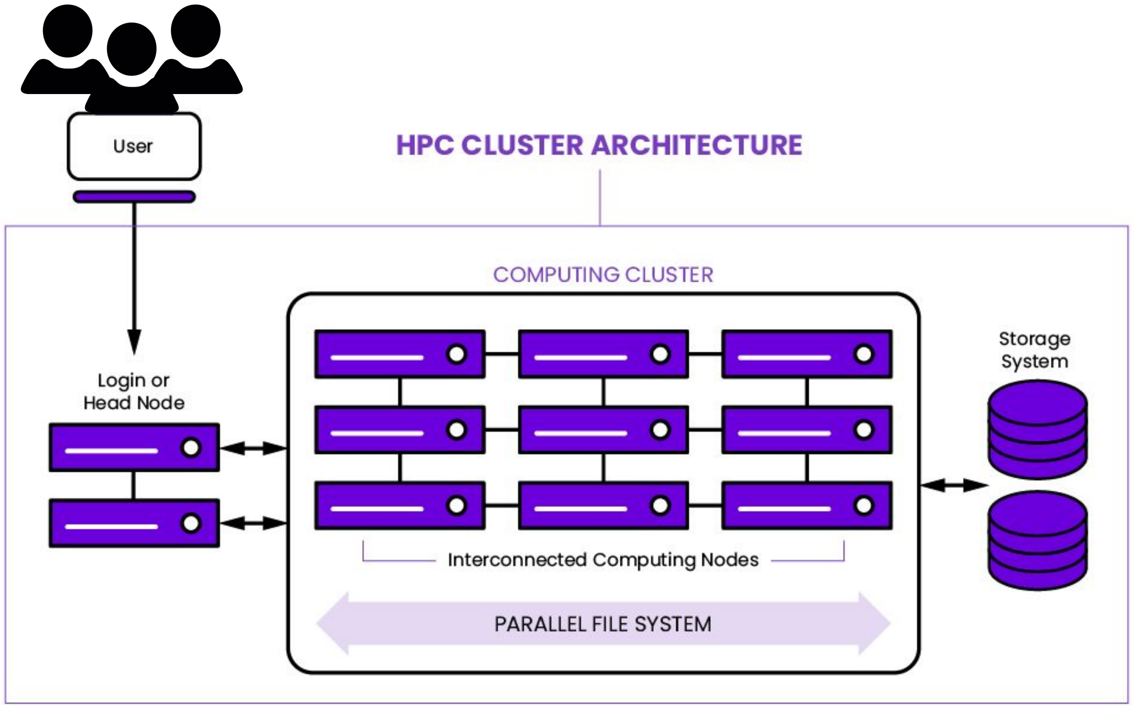
On a High-Performance Computing (HPC) system, the situation is very different:

- It's a **multi-user environment**
- The hardware consists of compute nodes that **cannot be accessed by normal users**
- You have **no root permissions**: you cannot install software system-wide, change OS settings…
- The goal is to optimize for the cluster hardware and ensure reproducibility, not just "get a program running"

1. Software management: HPC vs Personal PC

(Admin permissions)





1. Software management: HPC principles

On a personal machine, you can afford to "break" your setup and reinstall. On an HPC the situation is completely different:

- Overwriting system libraries could **break unrelated software for all the other users.**
- You must be able to **reproduce exactly an analysis months or years later** with the same configuration.
- In case of **unrestricted installation** a malicious or careless user could **install malware.**

That's why HPC systems:

- **Organize software into modules, virtual environments or containers** so you can choose versions without interfering with others.
- This latter approach also helps to **keep reproducibility.**
- **Contained privileges**: Users are sometimes free to install packages but only in their home or scratch directories, where they can't affect other users.

1. Software management: HPC principles

1. Security Reasons

An HPC is a shared multi-user environment — dozens, hundreds, or even thousands of people run jobs on the same nodes. If everyone could install or modify system software freely, it would be a security nightmare.

Key security risks if installation were unrestricted:

- **Malware injection**: A malicious or careless user could install software that steals credentials, mines cryptocurrency, or damages data.
- **Privilege escalation**: If someone installs a program with a known vulnerability, attackers could exploit it to gain root access.
- **Tampering with trusted tools:** If `ssh`, `ls`, or `python` were replaced with compromised versions, all users would be at risk.
- **Inadvertent breakage:** Without bad intent, a user might overwrite a library with an incompatible version, causing other users' jobs to crash.

Example:Imagine installing a Python library that silently sends computation results to an external server. On your laptop, you'd only risk your own data — on an HPC, you could leak other researchers' sensitive datasets.

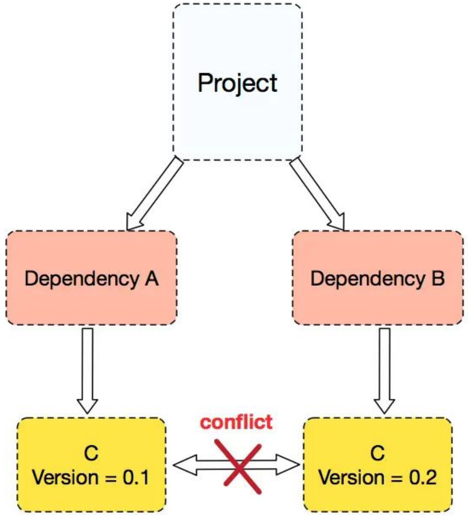
1. Software management: HPC principles

2. Maintainability Reasons

HPC admins have to maintain stability across hundreds of nodes, often for years.

Challenges if users installed software freely:

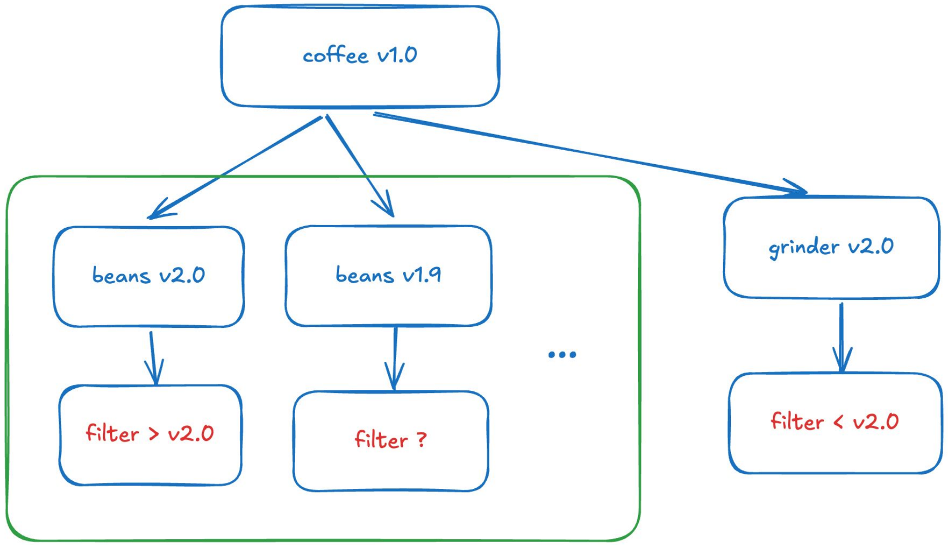
- **Version chaos**: If every user installed different versions of `gcc`, `cuda`, or `numpy` in system directories, no one could guarantee reproducibility.
- **Dependency conflicts**: Overwriting system libraries could break unrelated software



1. Software management: HPC principles

How admins handle it instead:

- **Centralized software management**: HPC teams use tools like EasyBuild or Spack to build and deploy software in a controlled way across all nodes.
- **Environment modules**: Users load software via `module load python/3.10` instead of installing it themselves.
- **Containers**: Singularity or Docker (where allowed) lets users run custom environments without touching the base system.



HPC vs Personal PC: Summary

Feature	Personal PC (Windows)	HPC
Simultaneous users	Typically one or a few, separate sessions	Dozens or hundreds of active users at the same time.
Permissions	User can install and modify system software	No root; install only in your home space if needed
Software installation	Run graphical installers. No risk of conflict with other users.	Use modules, environment managers, or compile with tools like EasyBuild, Conda, Mamba
Hardware	Single CPU (multi-core), limited RAM	Multiple nodes, each with many cores and large memory; high-speed interconnect
Purpose	General use (office, games, etc.)	Large-scale data processing or computationally intensive tasks
Program execution	Run directly on your machine	Submit jobs to a scheduler/queue; executed on compute nodes
Updates	Automatic, controlled by the user	Planned by administrators to avoid disrupting jobs
Version management	Install or uninstall at will	Multiple versions coexist. You choose which to use

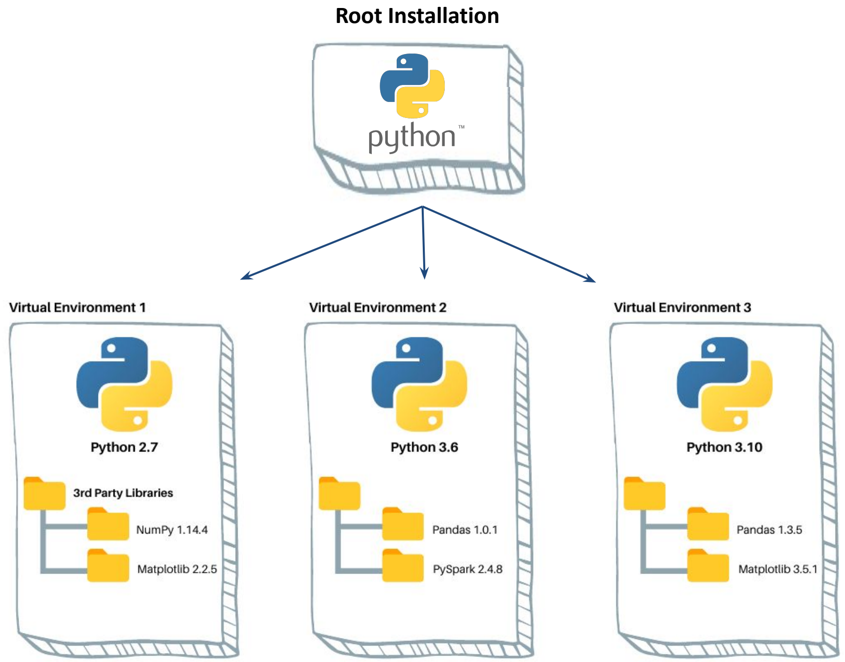
Overview

1. Software management basics.
2. Software managers: EasyBuild, Conda-Mamba
3. Introduction to Virtualization. Containers: Docker & Singularity

3. Introduction to Virtual Environments

Virtual Environments.

- A **virtual environment** is an **isolated workspace** on your computer where you can install software **without interfering** with the global system.
- **Examples:** Python *venv*, Conda, Micromamba environments, R virtual libraries.
- **Pro:** No root permissions needed.
- **Con:** Doesn't virtualize the OS, system or hardware. Only manages the software layer.

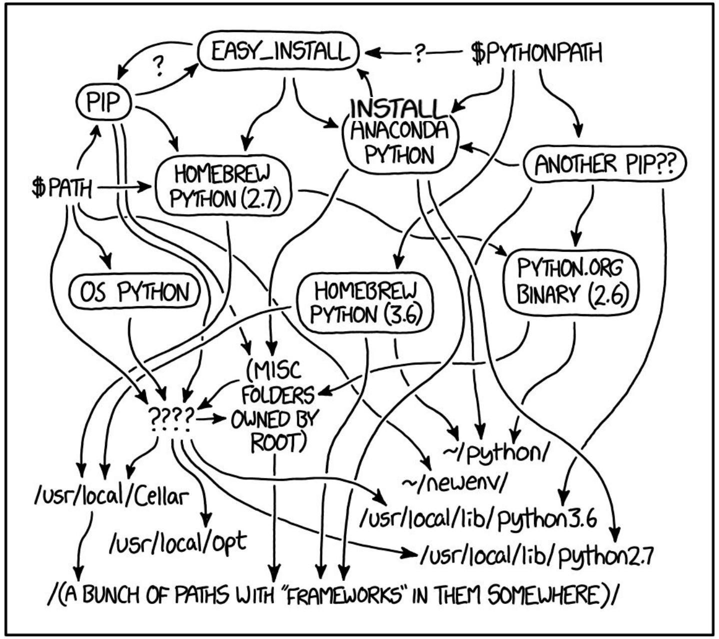


3. Introduction: Virtual Environments

Virtual Envs usage in HPC

On an **HPC cluster**, you usually **don't have admin/root permissions**. That means:

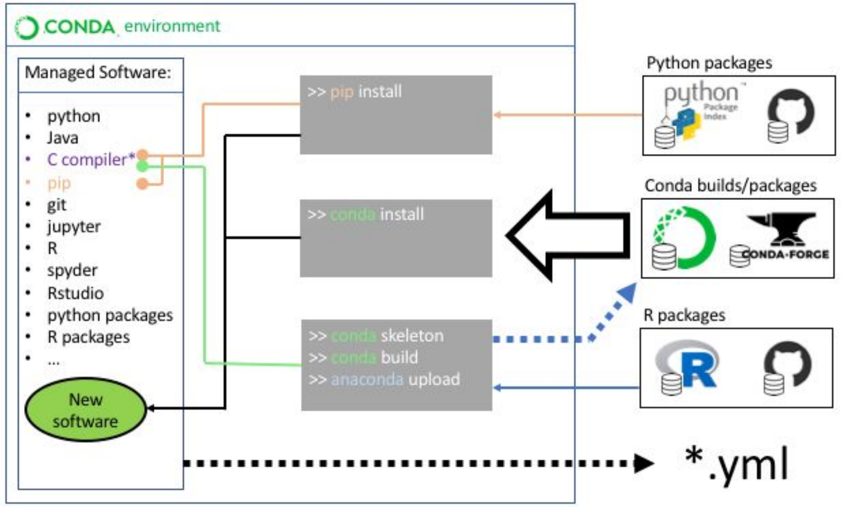
- You **cannot install system-wide software** (like `pip install` globally).
- **Users shares the same filesystem**: installing packages globally would cause **conflicts**.
- Even inside a virtual environment its difficult to avoid conflicts. **Software managers** exist for the sole reason to **simplify this process.**



3. Software managers: Conda/Mamba

3. Virtual Environments (Software-Level)

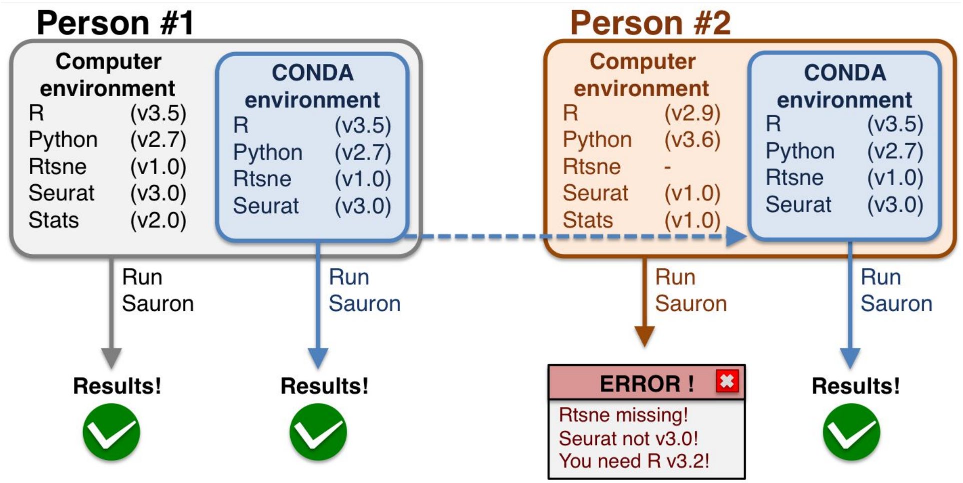
- **Pro:** No root permissions needed, isolates packages and dependencies **inside the same OS** without affecting system installations.
- **Con:** Doesn't virtualize the OS, system or hardware. Only manages the software layer.
- **HPC usage:** Extremely common for running different Python/R environments on shared clusters without interfering with system-wide libraries.
- **Example:** Python *venv*, Conda, Micromamba environments, R virtual libraries.



3. Software managers: Conda

Conda

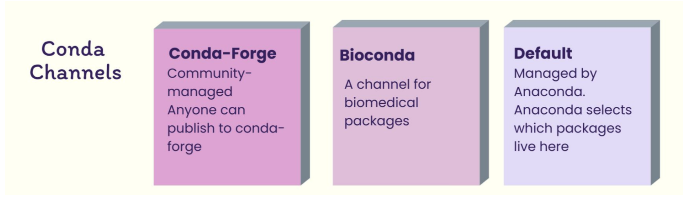
- **Conda** is a **package/environment manager**. Originally created for Python but works with **any language** (R, C/C++, Fortran…).
- Creates **isolated environments**, each with its own Python/Language version and libraries.
- Manages **dependencies**: ensures all required libraries are installed in compatible versions.

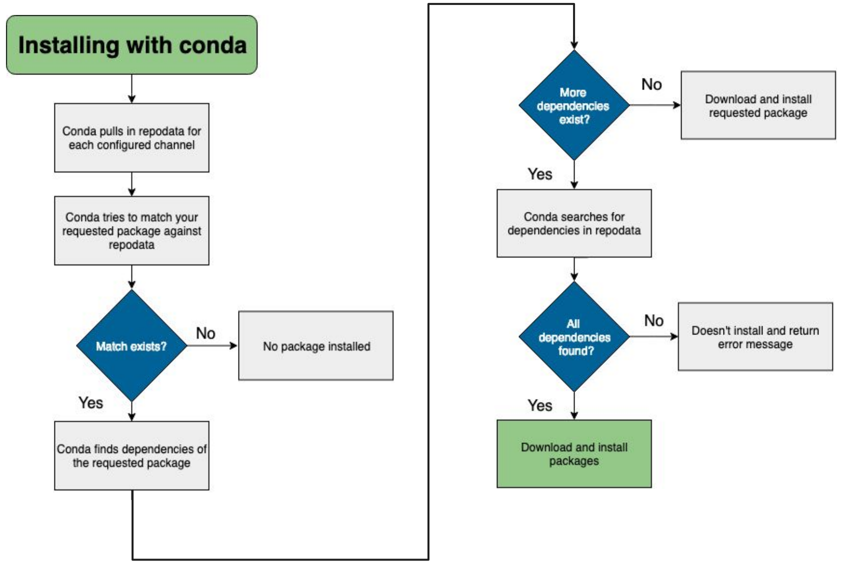


3. Software managers: Conda

Install with conda. Channels

- Not everything can be installed with *conda install.* The desired software must be accessed through a **conda channel**
- **Conda channels** are **repositories** (URLs) where Conda looks for packages.
- The base channel is *defaults*, but there are community-driven channels like *conda-force* or *bioconda*. Anyone can create a channel.

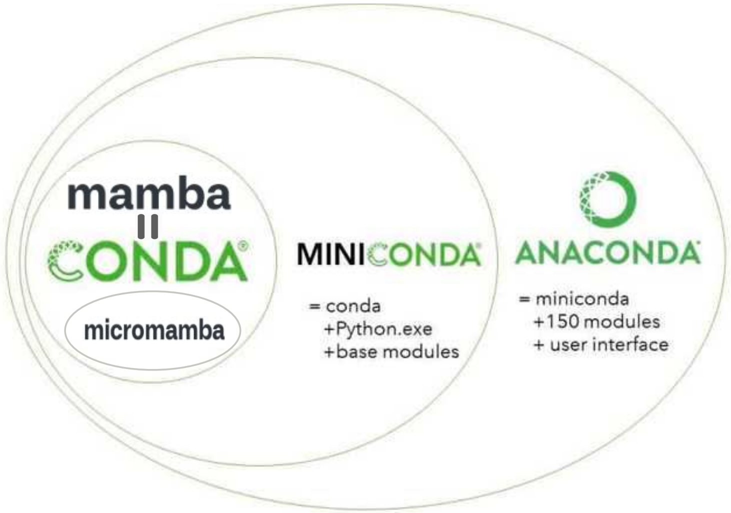

Conda Channels
Conda-Forge
Community-managed
Anyone can publish to conda-forge
Bioconda
A channel for biomedical packages
Default
Managed by Anaconda.
Anaconda selects which packages live here


Installing with conda
Conda pulls in repodata for each configured channel
Conda tries to match your requested package against repodata
Match exists?
No
No package installed
Yes
Conda finds dependencies of the requested package
More dependencies exist?
No
Download and install requested package
Yes
Conda searches for dependencies in repodata
All dependencies found?
No
Doesn't install and return error message
Yes
Download and install packages

3. Software managers: Mamba

Mamba

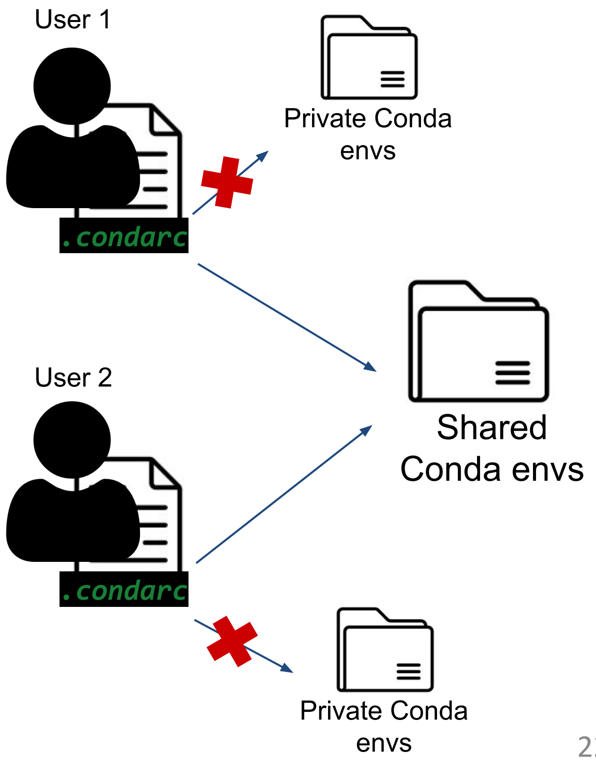
- A **drop-in replacement for Conda** written in C++ instead of Python.
- Uses the same package repositories and commands as Conda.
- Designed to solve Conda's main weakness: **slowness,** specially in dependency resolution.
- ***Micromamba***: A lightweight (10mb), and self-contained version of mamba **(installed in our HPC).**



3. HPC shared environments

HPC shared virtual envs

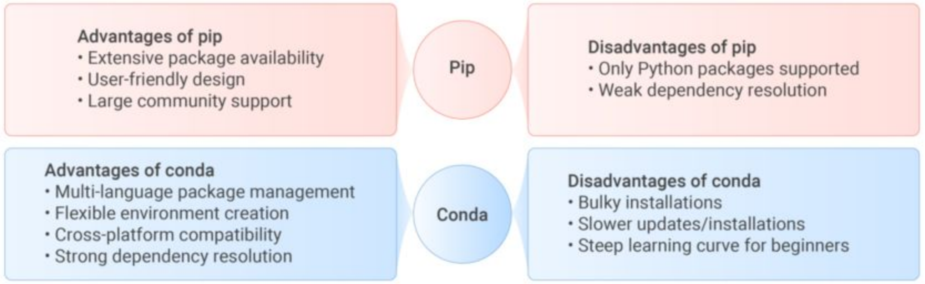
- **If each user creates its own environments** in the HPC, there may be **differences in results due to software versions**.
- Using shared environments **avoids differences in analysis** results due to environment differences.
- In our HPC, **all users can access the same *modules***, for softwares like **Nextflow** or **Singularity**
- You can also **change conda's configuration file** (*.condarc*) **to use the same list of environments as other users**.



3. Other package managers: Pip

Pip

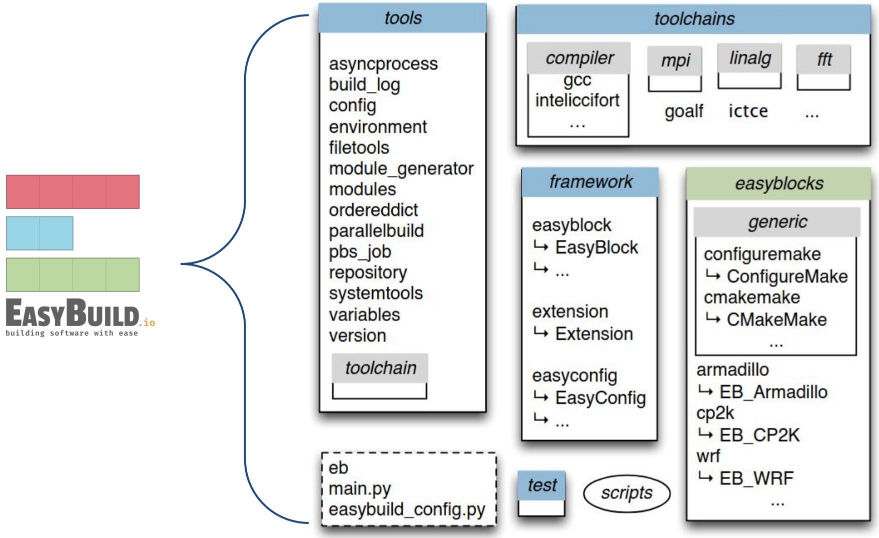
- Pip is the **default package manager for Python**. It is the **official tool** for installing packages from the Python Package Index (https://pypi.org/), which holds a **vast catalogue of packages.**
- Pip's sole purpose is to **install, update, and remove Python packages from PyPI.** Nonetheless, it can install binary packages (wheels) or software from source.
- Pip itself **does not have built-in environment management** like conda, but you can install it in a conda environment.


Advantages of pip
• Extensive package availability
• User-friendly design
• Large community support
Pip
Disadvantages of pip
• Only Python packages supported
• Weak dependency resolution
Advantages of conda
• Multi-language package management
• Flexible environment creation
• Cross-platform compatibility
• Strong dependency resolution
Conda
Disadvantages of conda
• Bulky installations
• Slower updates/installations
• Steep learning curve for beginners

3. Software managers: EasyBuild

Easybuild.

- EasyBuild is an **open-source framework** designed specifically for **building and installing scientific software on HPC systems**.
- Our HPC uses this framework to manage certain global software for all users (e.g. Nextflow or singularity)
- **Encapsulates software in *modules*,** which can be loaded via: `module load <module-name>`



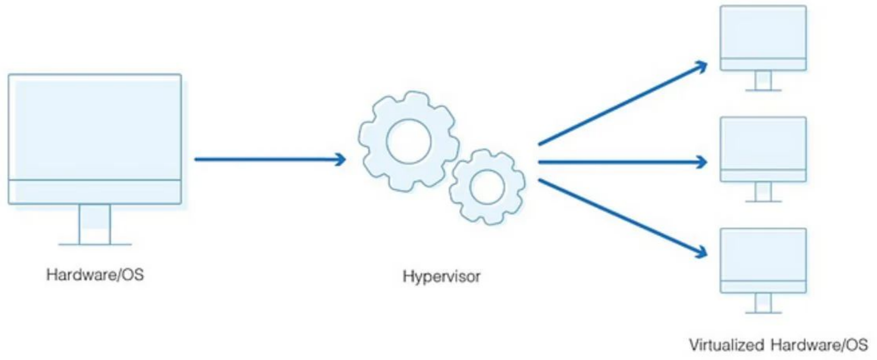
Overview

1. Software management basics.
2. Software managers: EasyBuild, Conda-Mamba
3. Introduction to Virtualization. Containers: Docker & Singularity

3. Introduction to virtualization

What is Virtualization?

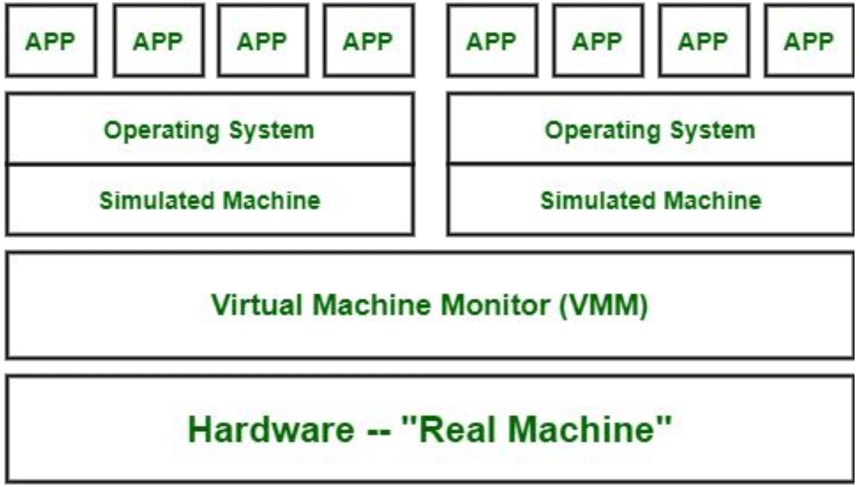
- Virtualization is the process of creating a **virtual version of a computing resource**.
- It **behaves like the real thing, but is actually a software** on top of physical hardware.
- Instead of interacting directly with the physical server, **you interact with a virtual replica** that is **isolated from your system.**



3. Introduction to virtualization: Virtual Machines

A. Full Virtual Machines

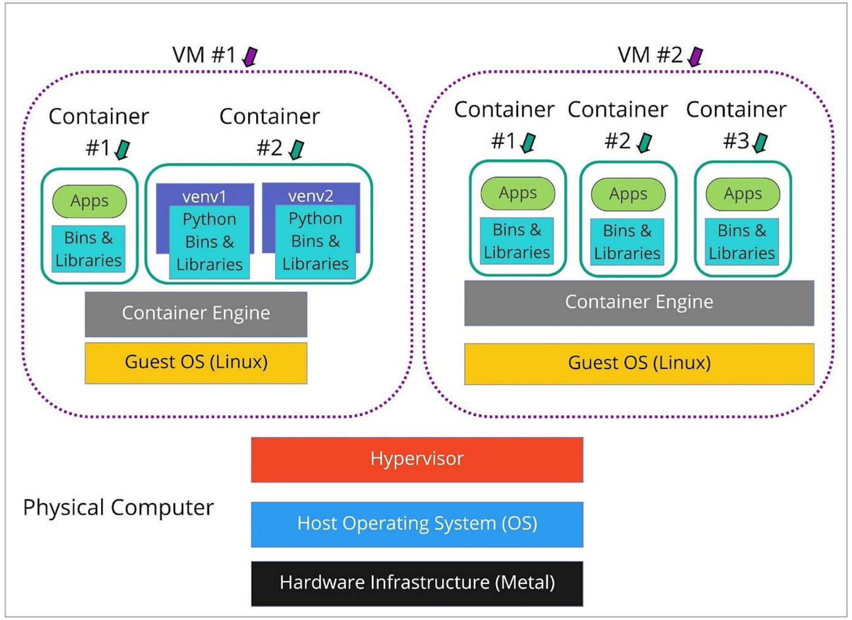
- A **Virtual Machine (VM)** is a software-based emulation of a physical computer, including OS and hardware components using a **hypervisor**.
- **Pro:** Total isolation. They **can run a completely different operating system**.
- **Con: Very slow** due to full hardware emulation.
- **HPC usage: Rare**, mainly for testing environments or isolated, security-sensitive workloads.
- Example: Running **Ubuntu in VirtualBox** on a Windows laptop.



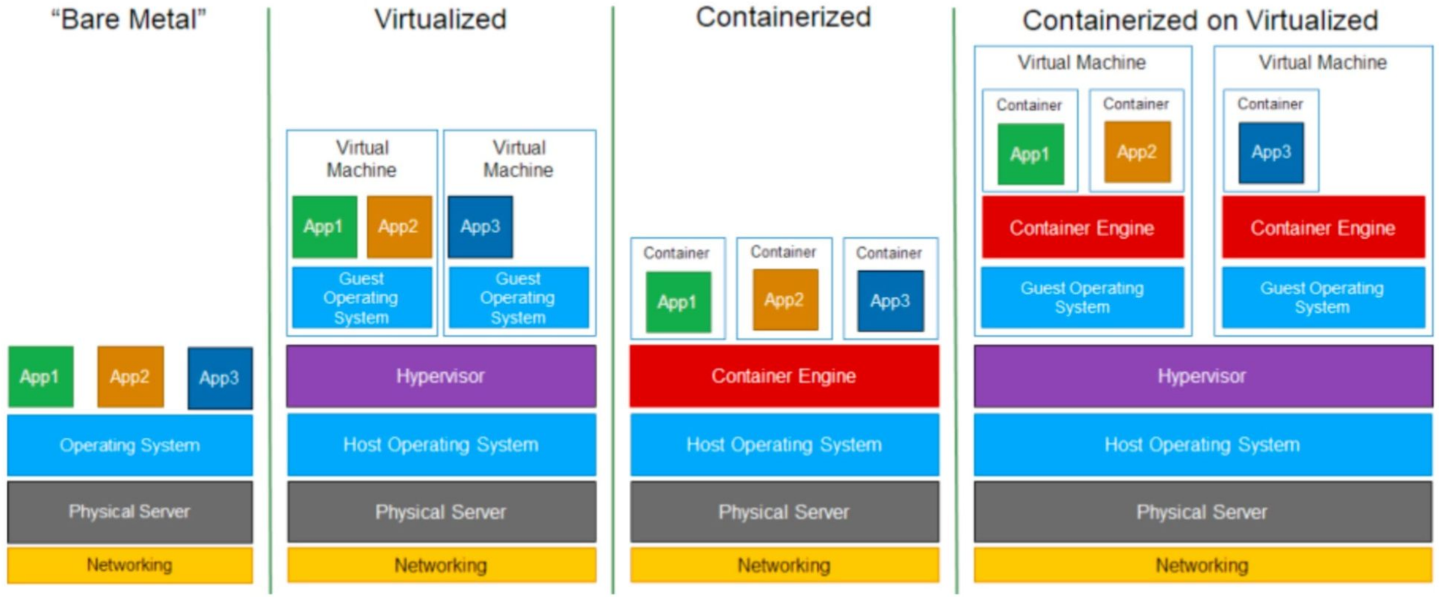
3. Introduction to virtualization: Containers

B. OS-level Virtualization (Containers)

- A container is a lightweight, **standalone package** that **bundles together an application and everything it needs to run**, but **shares the host system's kernel** instead of emulating hardware.
- **Pro: Near-native performance, lightweight.**
- **Con**: Shares the host OS kernel, **cannot simulate a different OS than the host's.**
- **HPC usage: Very common** for packaging software to run without dependency issues.
- Examples: Docker, Singularity.



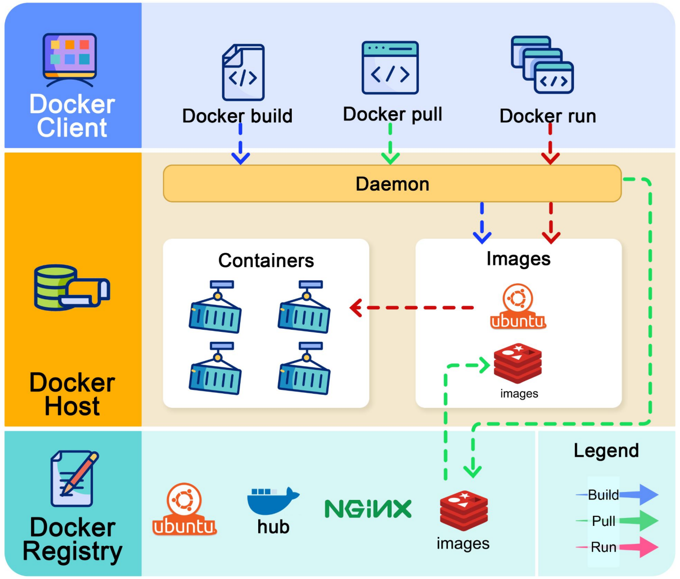
3. Introduction to virtualization: VMs vs Containers



3.1 Containers: Docker & Singularity

Docker

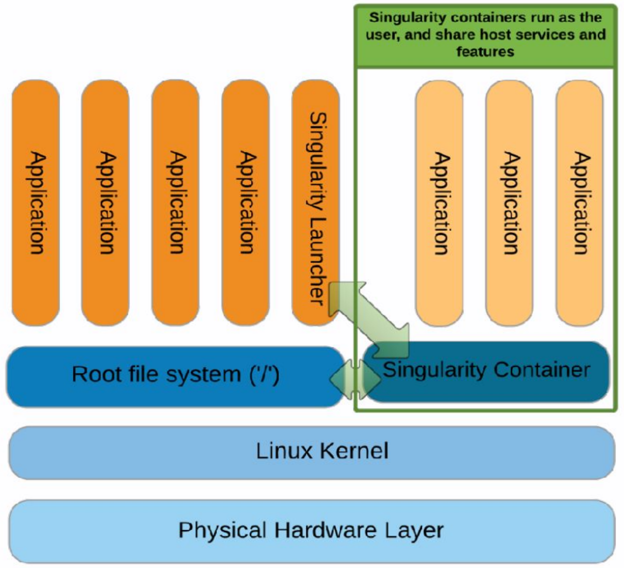
- Docker is a popular platform designed for building, sharing, and running applications in **containers**.
- By default, Docker containers run as the **root user** on the host system. This can pose a **security risk in multi-user environments** like an HPC.
- Unfortunately, we cannot directly use Docker in our HPC.



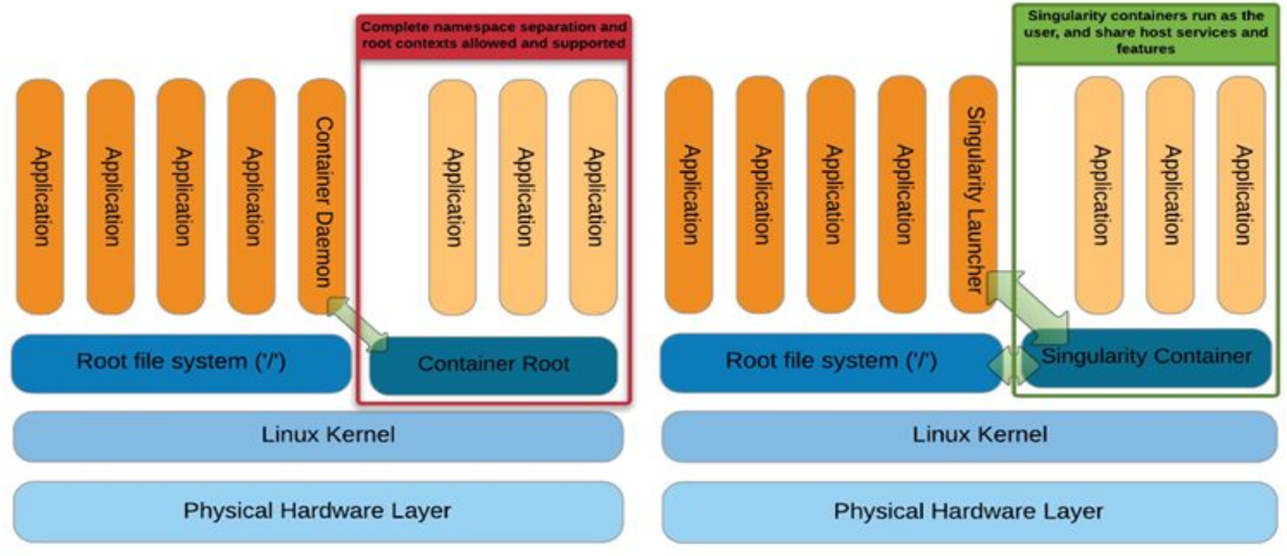
3.1 Containers: Docker & Singularity

Singularity

- Singularity is a container platform specifically **designed for high-performance computing (HPC)** and scientific research environments.
- Singularity containers **run as the same user that launched them** on the host system. This is a **security feature for HPC clusters**.
- It's our go-to to create and run containers in the HPC
- Galaxy's https://depot.galaxyproject.org/singularity includes a lot of singularity **containers for bioinformatic analysis**:



3.1 Containers: Docker & Singularity

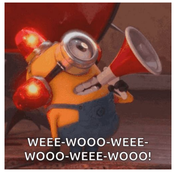


General Container eg Docker

HPC Container Singularity

Essential takeaways

- **You cannot install anything on the HPC that needs admin permissions.**
- Prioritize **modules over virtual environments** (More efficient)
- If you only need to **execute one certain task** (e.g. run FastQC) use a **singularity container**
- If you need to **interact with files or data** dynamically use **virtual environments** (Micromamba/Pip)
- **DON'T RUN HEAVY JOBS IN THE LOGIN NODE!!**



Practical exercises

The best way to understand everything is with some hands-on exercises.