Computer Environments

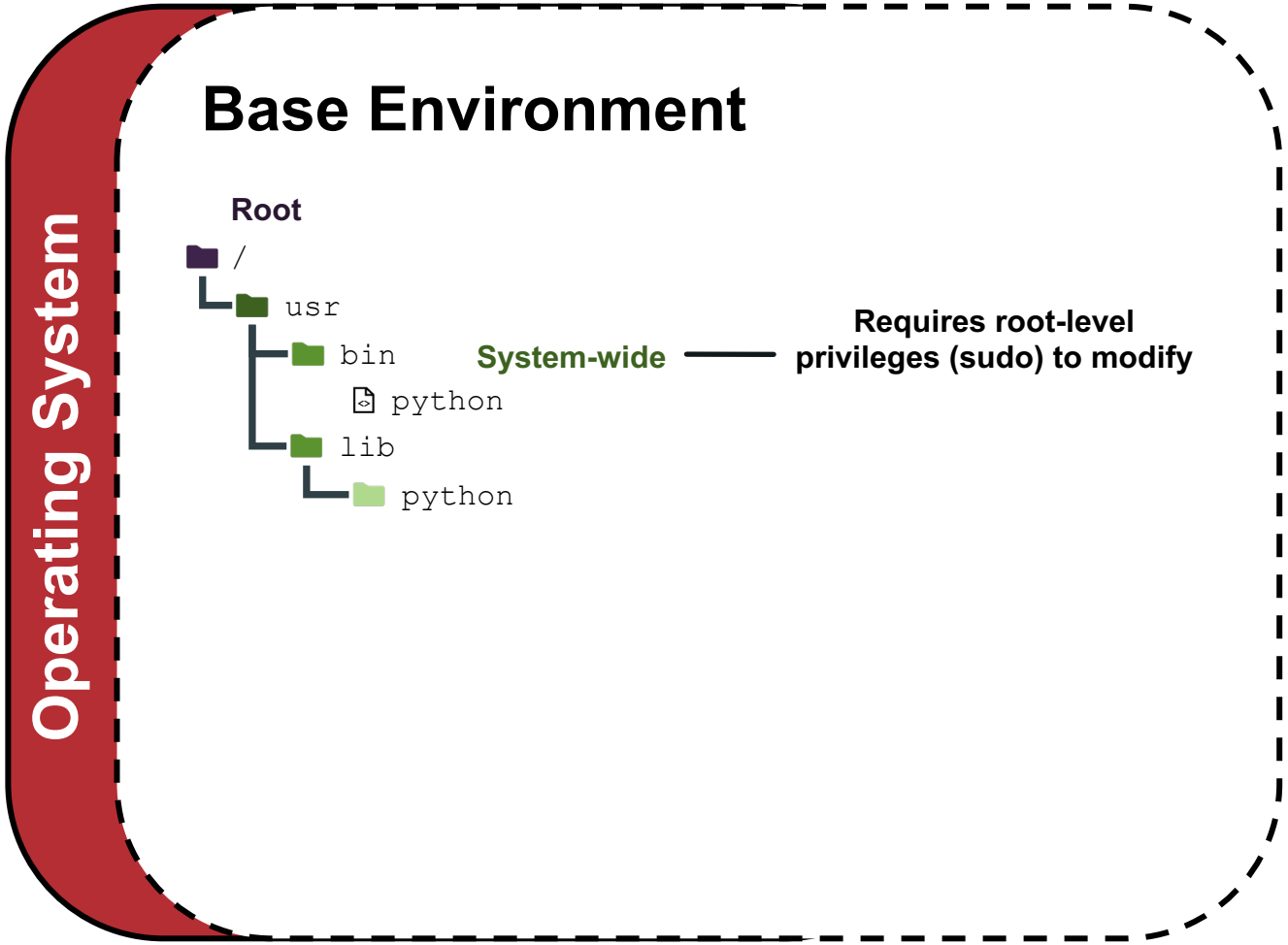
Funding Disclosure

- This work is supported by the United States Centers for Disease Control and Prevention funding under the Global Health Security Partnerships: Expanding and Improving Public Health Laboratory Strategies and Systems (Grant Number: NU2HGH000080).

Table of contents

Introduction to Computer Environments

Computer Environment Basics



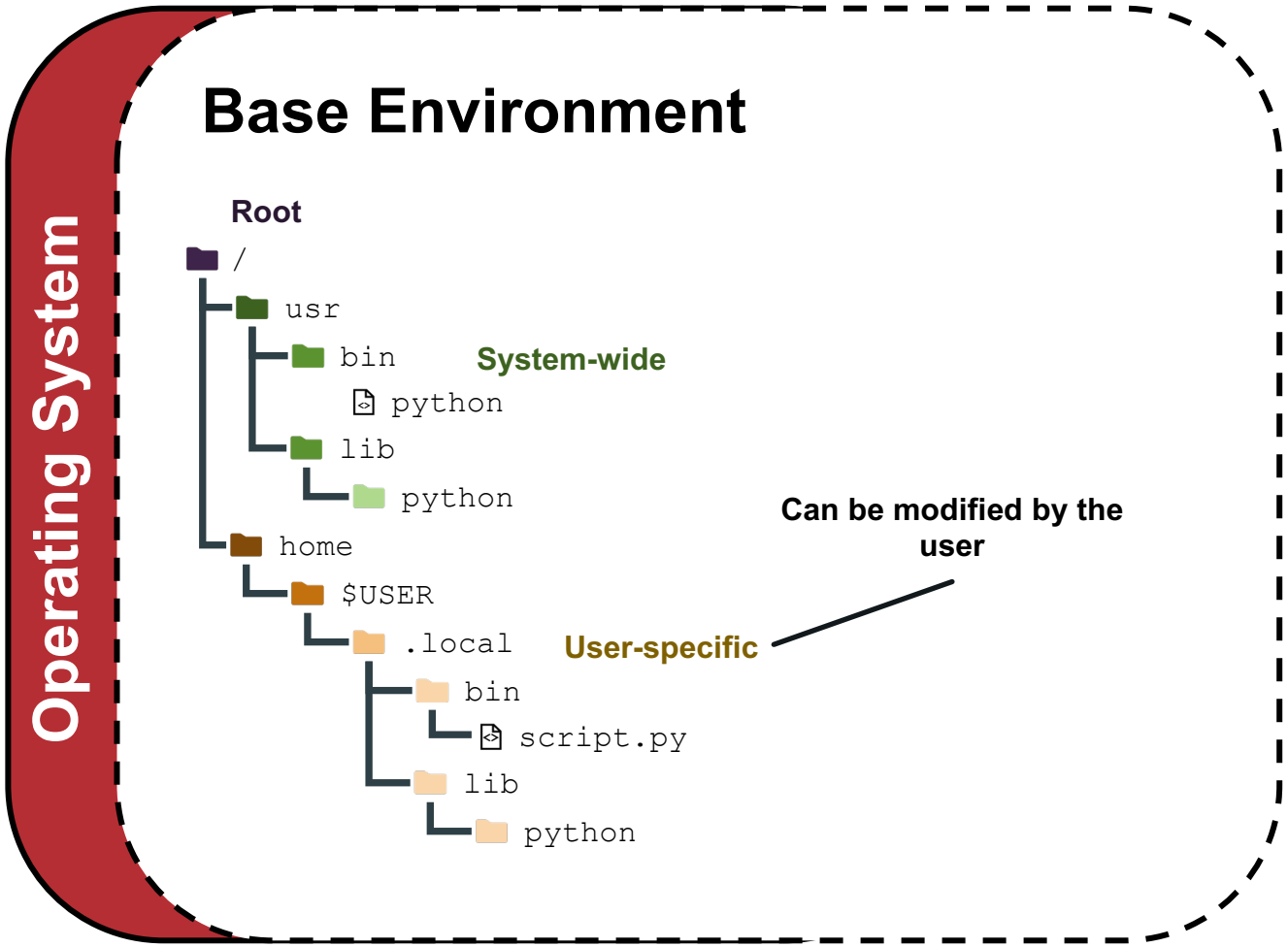
Operating System:
Software that manages the computer hardware (e.g., Ubuntu)

Software:
One or more executable scripts that perform tasks

Dependencies:
Required software components (e.g., Java, Python, Python modules)

Library:
A collection of dependencies (often software / language specific)

Computer Environment Basics



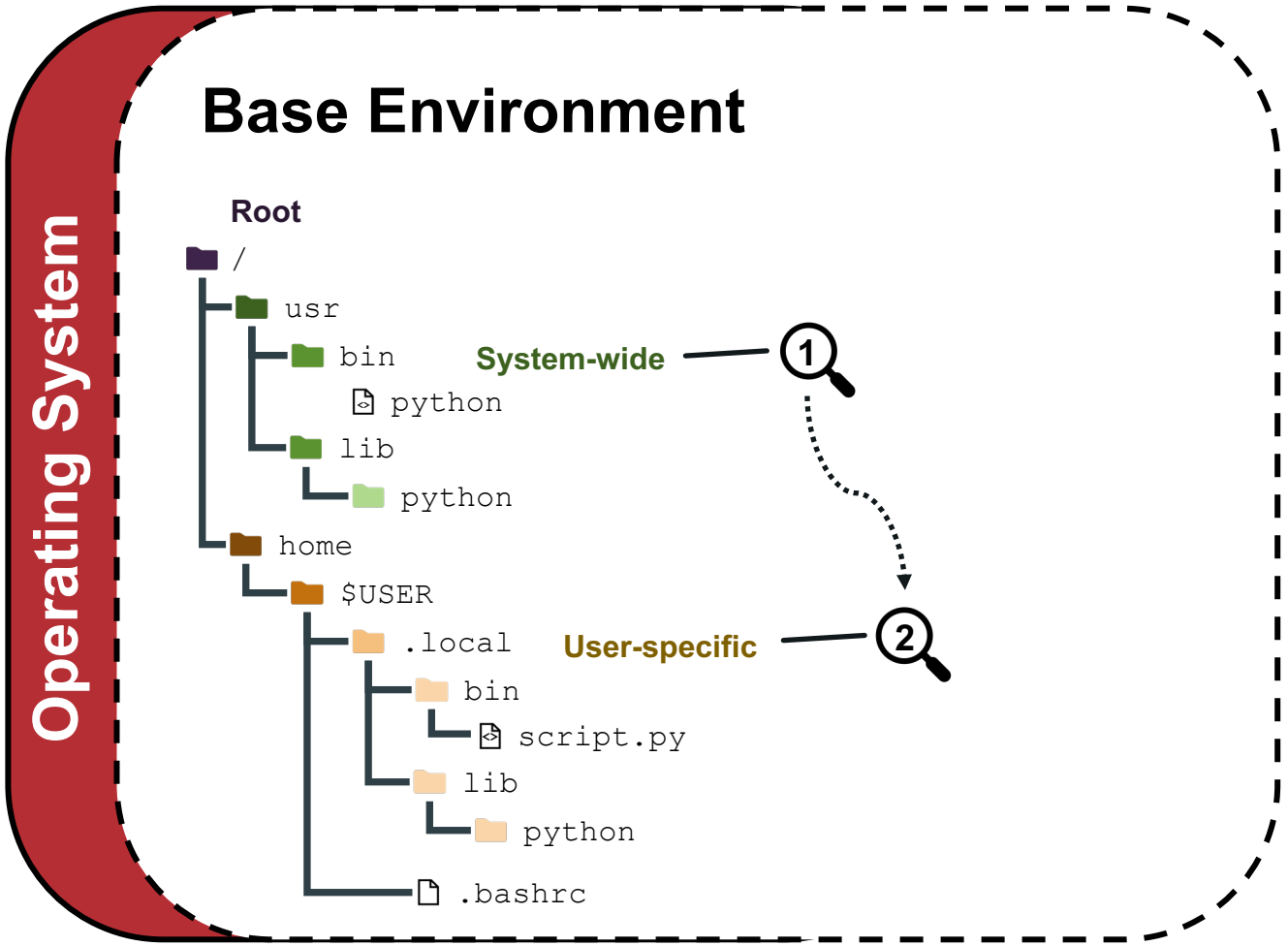
Operating System:
Software that manages the computer hardware (e.g., Ubuntu)

Dependencies:
Required software components (e.g., Java, Python, Python modules)

Library:
A collection of dependencies (often software / language specific)

Software:
One or more executable scripts that perform tasks

Computer Environment Basics



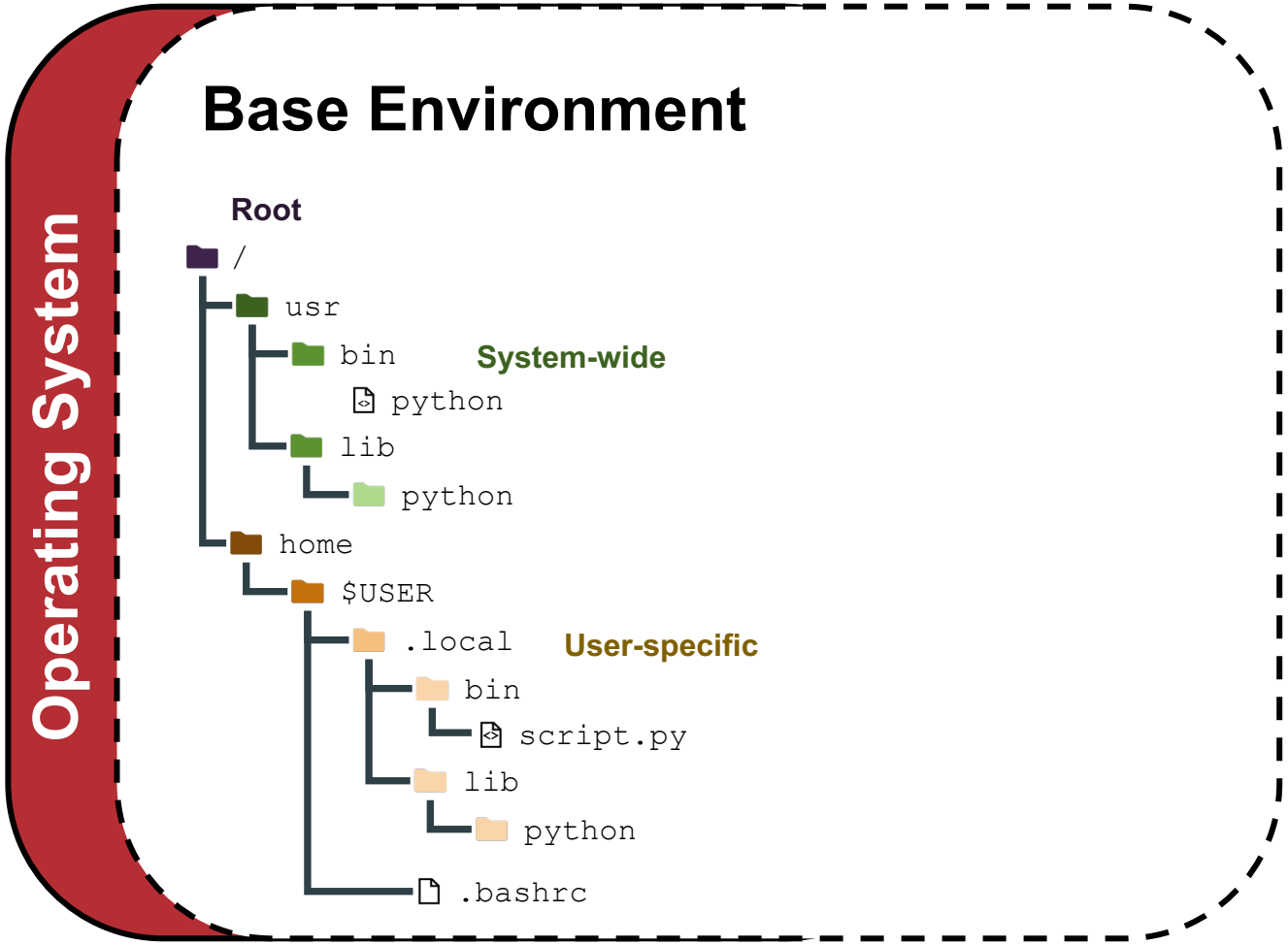
PATH Resolution:
The operating system searches **system-wide** resources first, unless explicitly told otherwise.

You can update the search order by modifying the `PATH` variable:

```
export PATH=/home/${USER}/.local/bin/:$PATH
```

Add this to your `.bashrc` file to make it occur on login

Computer Environment Basics

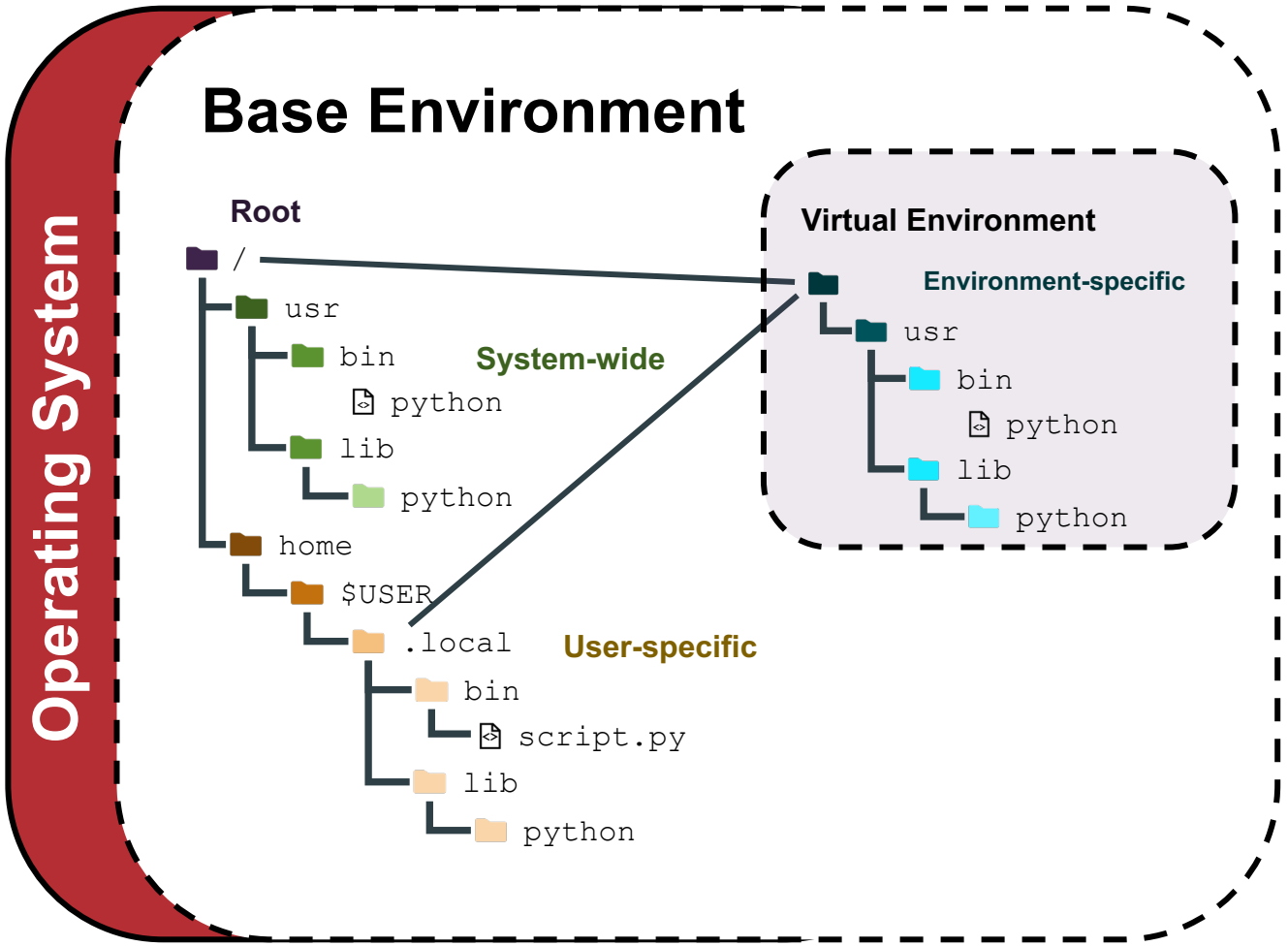


Managing dependencies:
Dependencies can be managed using `PATH`.

⚠Can get messy fast!

Software can **change** **behavior** or **break**!

Computer Environment Basics



Virtual environments:
Virtual environments help manage complex dependencies.

Environment management programs:

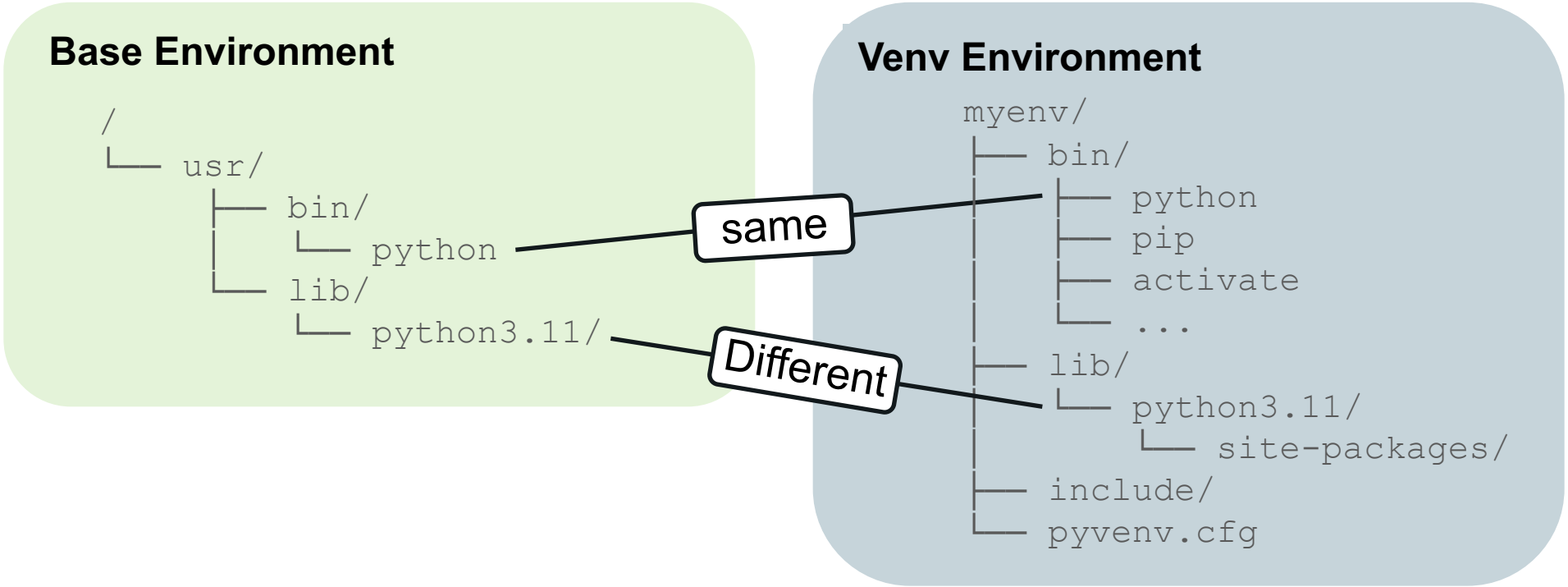
- `venv` **(Python-specific)**
- `Conda` / `Mamba` / `Pixi` **(Universal)**
- `Containers` **(more on this later!)**

Each program manages environments **very** differently!

Environment Management Programs

Environment Management (Venv)

Venv is a lightweight, python-specific environment management tool that facilitates the **isolation of python libraries, <u>not</u> the python executable.** Restricted to the base operating system.



Example: Venv

```
# Create and activate environment
python -m venv myenv
source myenv/bin/activate

# Install modules
pip install numpy<2 pandas=1.5.0


# Run script
Python script.py


# Deactivate (close) environment
deactivate
```

Sharing Environments (Venv)

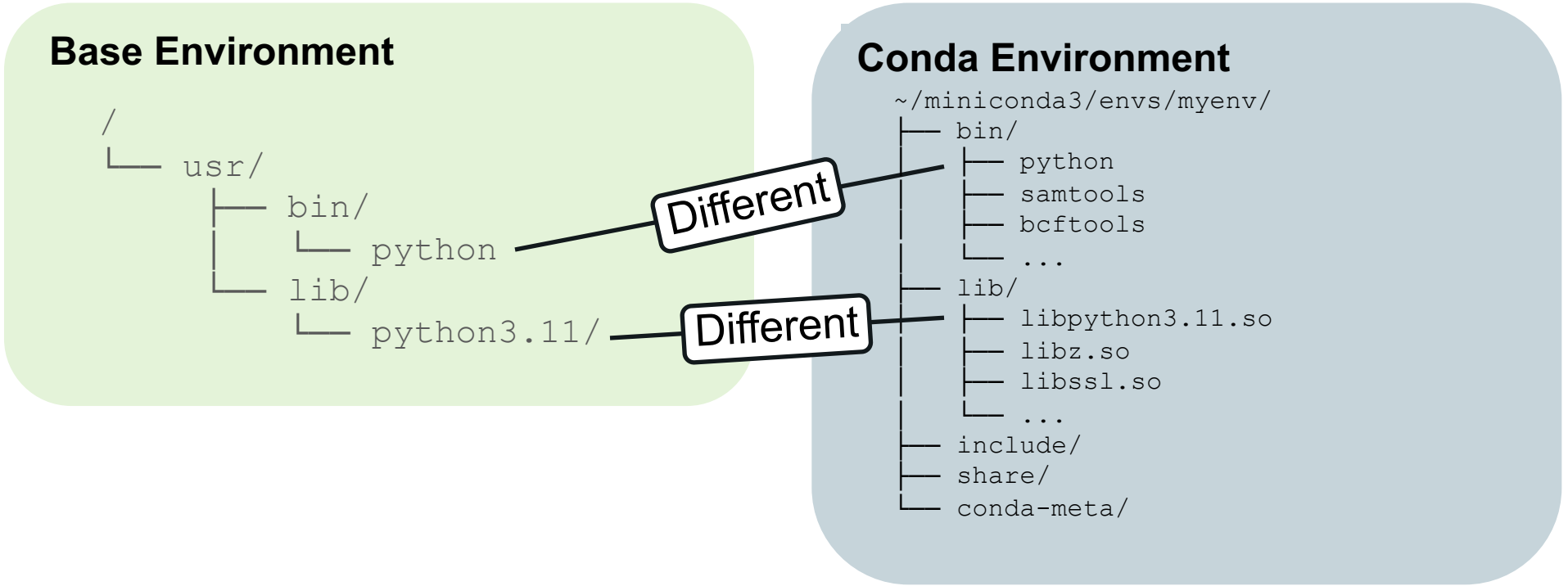
```
# Create sharable requirements file
pip freeze > requirements.txt

# Install via requirements file
pip install -r requirements.txt
```

requirements.txt:

```
numpy==1.18
pandas==1.5.0
```

Environment Management (Conda / Mamba / Pixi)

Conda, **Mamba**, and **Pixi** support multiple languages (Python, Java, C++, etc.,) and binaries. Manage **both** the executables and their dependencies. Restricted to the base operating system.



Conda vs Mamba vs Pixi

	Conda	Mamba	Pixi
Initial Release Date	2012	2019	2023
Ecosystem	Conda	Conda	Conda + PyPI
Solver speed	Slow	Fast	Fast
Environment scope	Global / named	Global / named	Project-local
Config format	`environment.yml`	`environment.yml`	`pixi.toml`
Reproducibility	Medium	Medium	High
Drop-in replacement for Conda	-	Sometimes	No

Miniconda and Micromamba

	Conda	Miniconda	Mamba	Micromamba
Drop-in replacement for Conda	-	Yes	Yes	No
Packages included	600+	130+	89	0
Approx. Install size (GB)	9.7	0.9	0.4	0.05

source: https://www.anaconda.com/docs/getting-started/concepts/anaconda-or-miniconda

- **Miniconda** is **Conda** but with fewer base packages
- **Micromamba** is an executable binary of **Mamba** with no base packages

Micromamba is recommended for most applications

Micromamba Shell Configuration

Importance difference:

- Conda, Mamba, and Miniconda are **not compiled** – uses interpreter (Python), so code runs in **current shell**
- Micromamba is **compiled** (binary executable) - no interpreter, so code runs in **subshell**

The subshell output must be interpreted by the current shell for changes to be applied to the environment!

Micromamba solves this by wrapping the binary in shell functions:

```
```
eval "$(micromamba shell hook --shell bash)"
```
```

Add this to your `.bashrc` file to make it occur on login

Example: Conda

```
# Create and activate environment
conda create \
    -n myenv \
    python=3.10


conda activate myenv


# Install binaries & libraries
conda install \
    -c conda-forge \
    -c bioconda \
     samtools numpy pandas


# Run script
Python script.py
samtools view alignment.bam


# Deactivate (close) environment
deactivate
```

Sharing Environments (Conda)

```
# Create sharable environment file
conda env export -n myenv > environment.yml

# Install via environment file
conda create -f environment.yml
```

environment.yml

```
name: myenv
channels:
  - bioconda
  - conda-forge
dependencies:
samtools=1.18=hd87286a_0
numpy=1.26.4=py311h64a7726_0
pandas=2.2.3=py311h7db5c69_1
```

Interactive Session

Insert link to webpage activity

APHL Global Health webpage

Questions?

This work is supported by the United States Centers for Disease Control and Prevention funding under the Global Health Security Partnerships: Expanding and Improving Public Health Laboratory Strategies and Systems (Grant Number: NU2HGH000080).