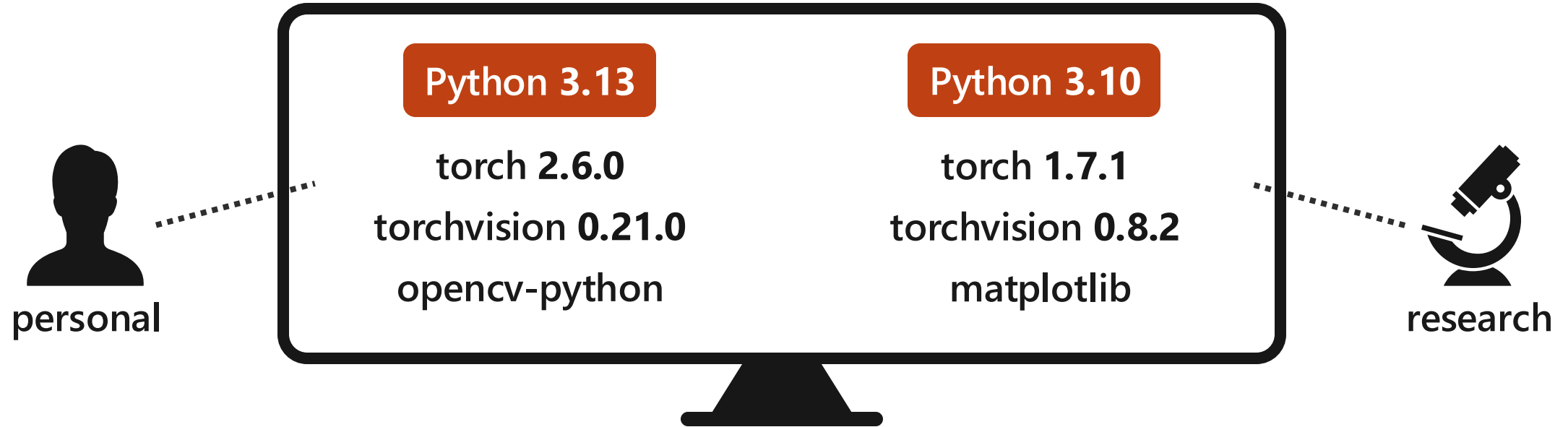


Virtual Environments for Python

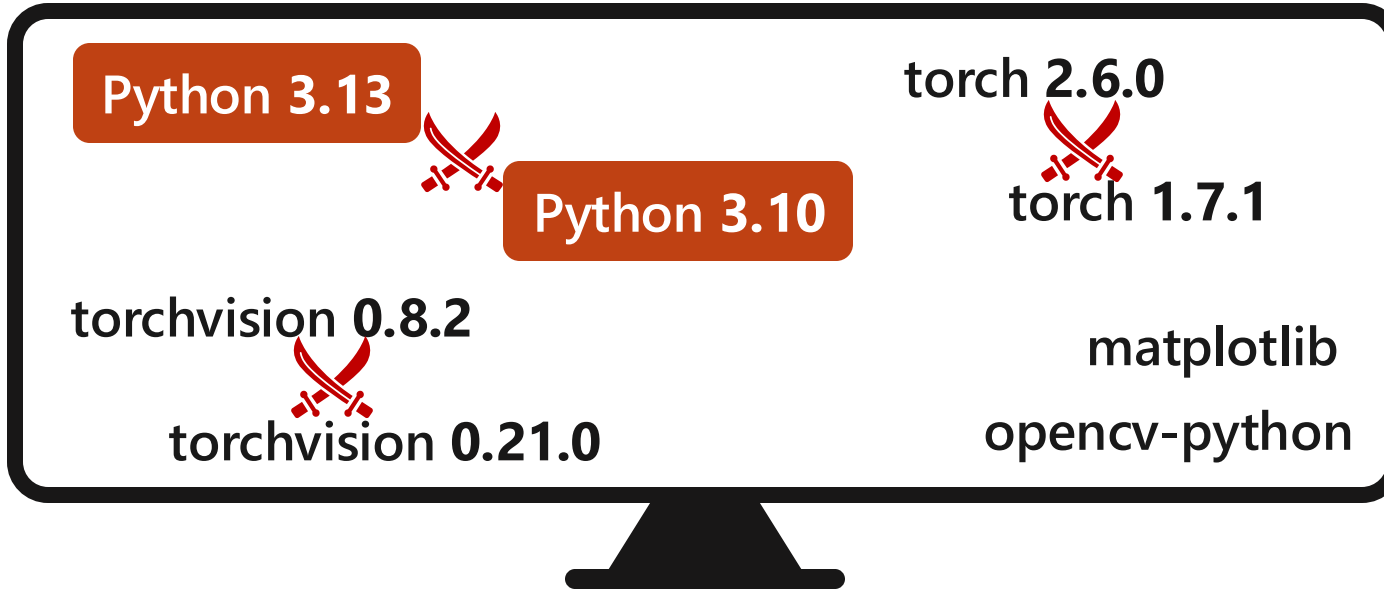
Created by [kengo](#)

Motivation



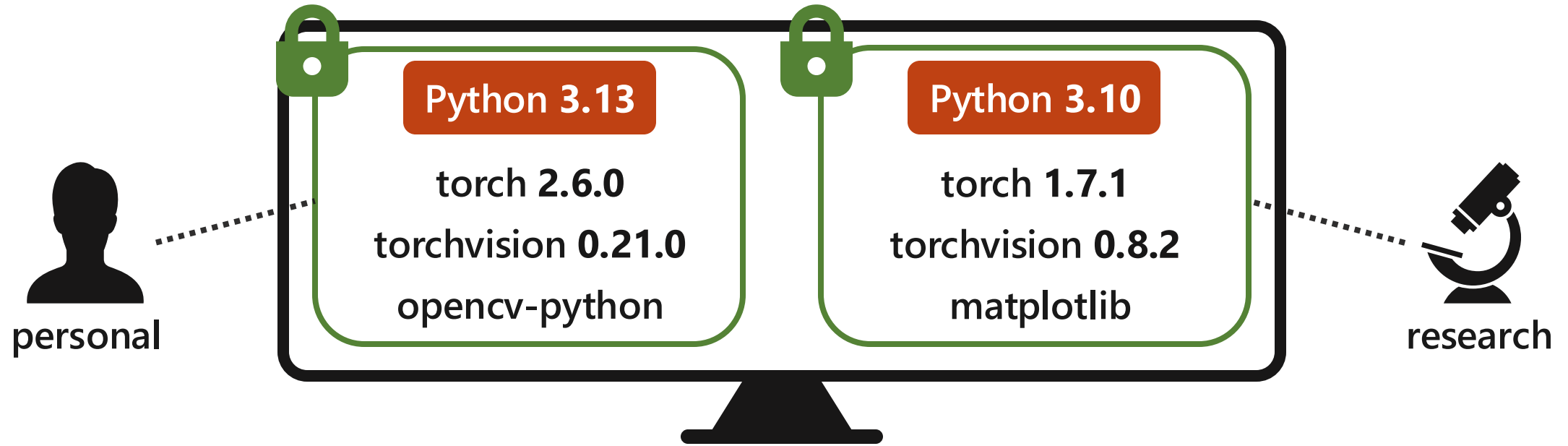
- We often have **several Python projects** on a **single computer**.
- Examples: personal, research, etc.

Motivation



- Without clear separation, **conflicts** can occur!
- **Conflicts**: the computer can't tell which **package version** to use.

Motivation



We separate these environments **virtually**.

Virtual Environments

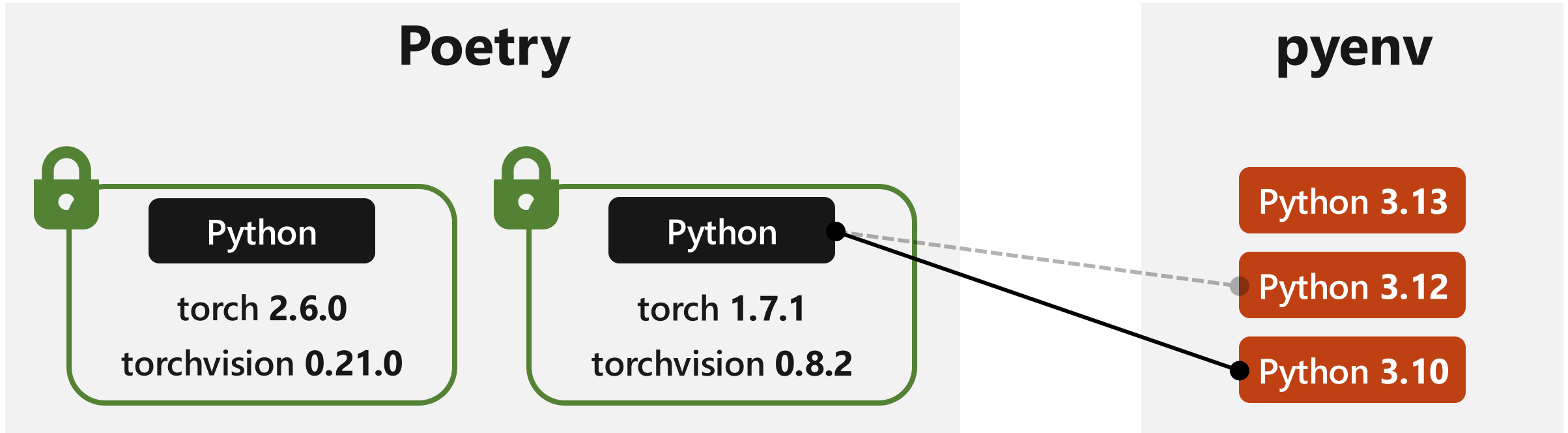
Virtual Environments

- A virtual environment consists of **2 main components**:
 - **Python versions**: 3.10, 3.13, etc.
 - **Libraries**: torch 2.6.0, torchvision 0.8.2, etc.
→ How to manage them?
- I will introduce **3 tools** for managing virtual environments:
 - **Poetry (+pyenv)**
 - **uv**
 - **Conda**
- Recommendation:
 - Personal → uv
 - Research → Conda

Tool Comparison

	Personal		Research	
	Poetry	uv	Conda	
Usability	△	○	△	
Speed	△	○	△	
Resolution	○	○	△	
Portability	○	○	△	
Binary support	×	×	○	
Ecosystem	○	△	○	
ML-friendly	△	△	○	

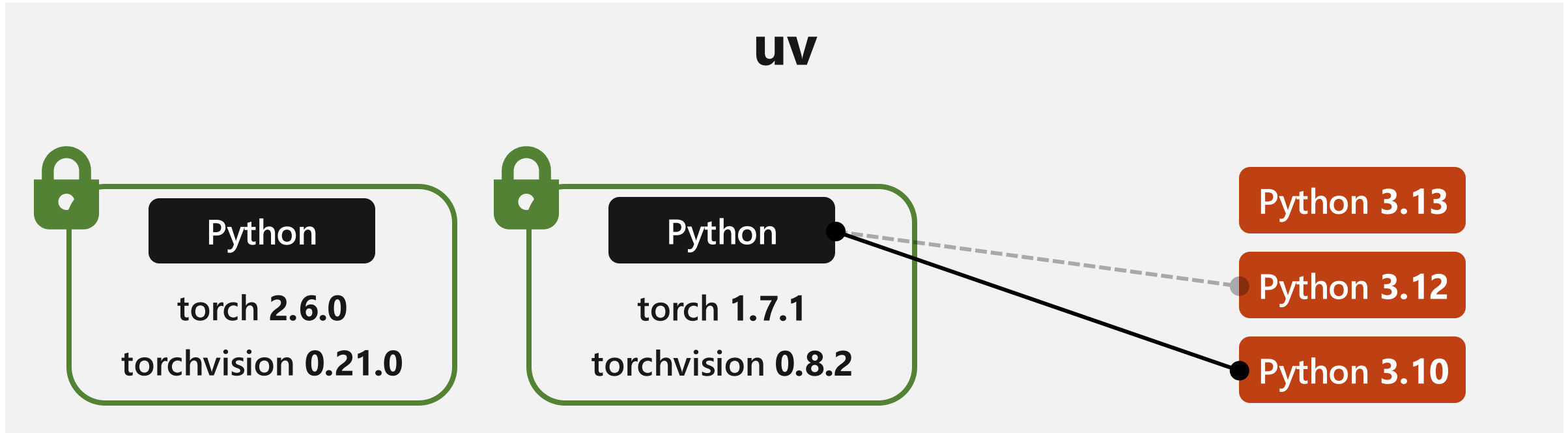
No.1 | Poetry (+pyenv)



- Poetry: manage **only Libraries**. → It needs Python version manager.
- pyenv: manage Python versions.

Strict dependency resolution / Reproducibility / Project-oriented

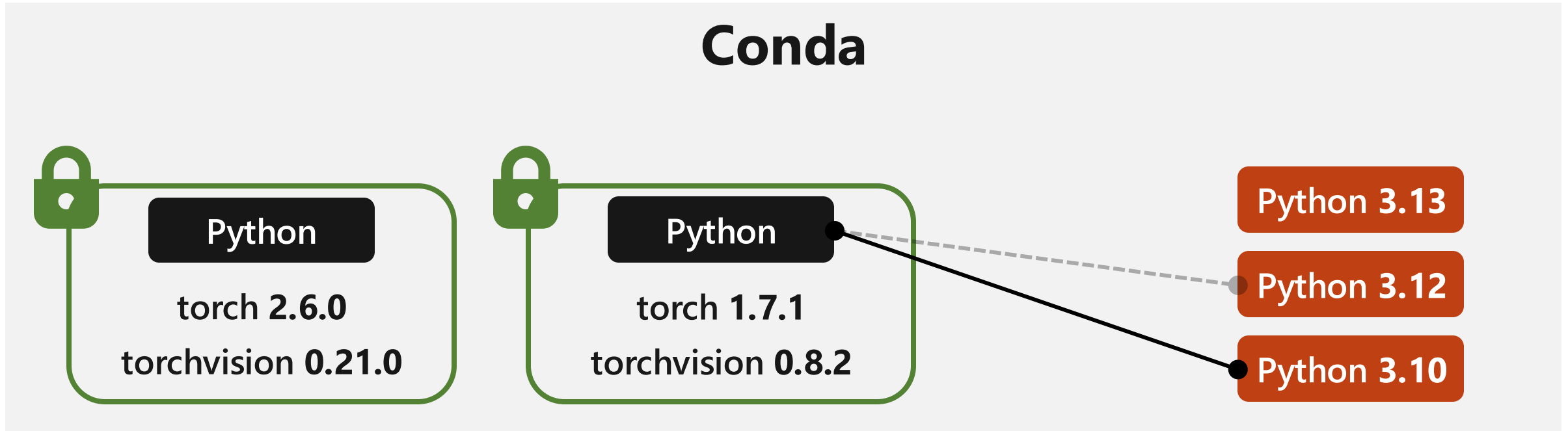
No.2 | uv



- uv: manage Python versions & Libraries.

Super fast / Lightweight / Simple CLI

No.3 | Conda



- Conda: manage Python versions & Libraries.
- **Notorious for pip conflicts...**

Binary support / Versatile / ML-friendly

Hands-on

No.1 | Poetry (+pyenv)

■ Setup Poetry.

- Install Poetry.

```
$ curl -sSL https://install.python-poetry.org | python3 -
```

- Add path to your shell configuration (~/.bashrc or ~/.zshrc).

```
$ echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc
```

- Finalize the setup.

```
$ source ~/.bashrc  
$ poetry --version  
$ poetry config virtualenvs.in-project true #recommend
```

No.1 | Poetry (+pyenv)

■ Setup pyenv.

- Install pyenv.

```
$ curl -fsSL https://pyenv.run | bash
```

- Add path to your shell configuration (~/.bashrc or ~/.zshrc).

```
$ echo 'export PATH="$HOME/.pyenv/bin:$PATH"' >> ~/.bashrc  
$ echo 'eval "$(pyenv init -)"' >> ~/.bashrc
```

- Finalize the setup.

```
$ source ~/.bashrc  
$ pyenv --version
```

No.1 | Poetry (+pyenv)

■ Create your virtual environment (in your workdir).

- Install and pin your desired Python version.

```
$ pyenv install 3.12.2 && pyenv local 3.12.2
```

- Initialize your virtual environment using the pinned version.

```
$ poetry init  
$ poetry env use `pyenv which python`
```

- Add any dependencies you like.

```
$ poetry add numpy
```

No.1 | Poetry (+pyenv)

- Activate your virtual environment (in your workdir).

```
$ . ~/.venv/bin/activate #if <2.0.0, poetry shell  
$ python <filename>
```

- Deactivate your virtual environment (in your workdir).

```
$ deactivate  
$ #(xxx) <username> → <username>
```

- Import an existing environment

```
$ ls -a #check if pyproject.toml exists  
$ poetry install
```

No.2 | uv

■ Setup uv.

- Install uv.

```
$ curl -LsSf https://astral.sh/uv/install.sh | sh
```

- Finalize the setup.

```
$ source ~/.bashrc  
$ uv --version
```

No.2 | uv

■ Create your virtual environment (in your workdir).

- Install and pin your desired Python version.

```
$ uv python install 3.12.2 && uv python pin 3.12.2
```

- Initialize your virtual environment using the pinned version.

```
$ uv init  
$ uv sync
```

- Add any dependencies you like.

```
$ uv add numpy
```


No.2 | uv

- Activate your virtual environment (in your workdir).

```
$ . ./venv/bin/activate  
$ python <filename>
```

- Deactivate your virtual environment (in your workdir).

```
$ deactivate  
$ #(xxx) <username> → <username>
```

- Import an existing environment

```
$ ls -a #check if pyproject.toml exists  
$ uv sync
```

No.3 | Conda

■ Setup Conda.

- Install Conda (for Linux 64-bit).

```
$ mkdir -p ~/miniconda3  
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O ~/miniconda3/miniconda.sh  
$ bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
```

- Finalize the setup.

```
$ source ~/miniconda3/bin/activate  
$ conda init --all  
$ echo 'export CONDA_AUTO_ACTIVATE_BASE=false' >> ~/.bashrc  
$ echo 'unset CONDA_SHLVL' >> ~/.bashrc  
$ source ~/.bashrc
```

No.3 | Conda

- Manage your virtual environment (in your workdir).
 - Initialize your virtual environment using your desired Python version.

```
$ conda create -n {ENV_NAME} python=3.12.2
```

- Activate your virtual environment.

```
$ conda activate {ENV_NAME}
```

- Add any dependencies you like.

```
$ conda install numpy -c conda-forge --override-channels  
$ pip install numpy #choose one (details in Notes)
```

No.3 | Conda

■ Manage your virtual environment (in your workdir).

- Deactivate your virtual environment.

```
$ conda deactivate
```

- Export your virtual environment.

```
$ conda env export > {YAML_NAME}.yaml
```

- Import an existing environment.

```
$ conda env create -n {ENV_NAME} -f {YAML_NAME}.yaml
```

Notes

■ Poetry Dependency Management

- Poetry strictly checks dependency compatibility.
- If a package doesn't support **all versions** in your Python range (e.g. ^3.8),
- Poetry throws an error — even if it works with your current version.
- Therefore, specify the **narrowest** version range possible (e.g. ~3.8).

■ Conda-Forge and Pip Compatibility Issue

- **Conda-forge** and **pip** may conflict in the same environment.
- **Pip** packages in a **conda-forge** environment might not be handled correctly.
- This can cause **dependency conflicts** and unexpected behavior.
- Therefore, use either **conda install** or **pip install**.

References

- Poetry: <https://python-poetry.org/>
- pyenv: <https://github.com/pyenv/pyenv>
- uv: <https://docs.astral.sh/uv/>
- Conda: <https://docs.conda.io/en/latest/>