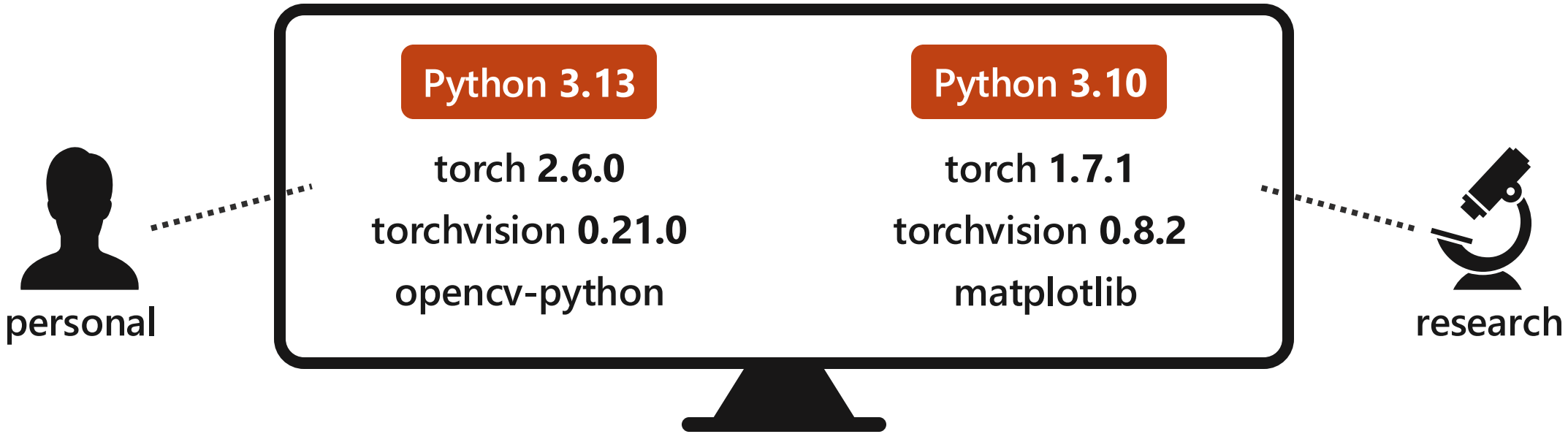
Virtual Environments for Python

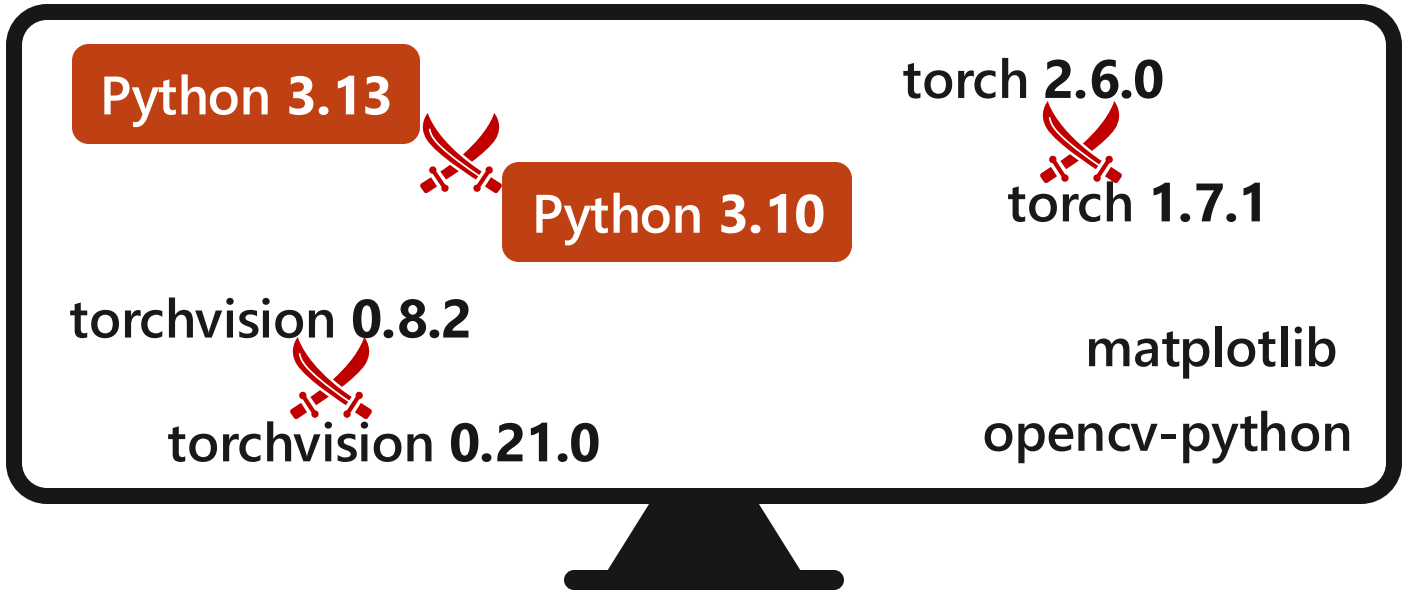
Created by kengo

Motivation



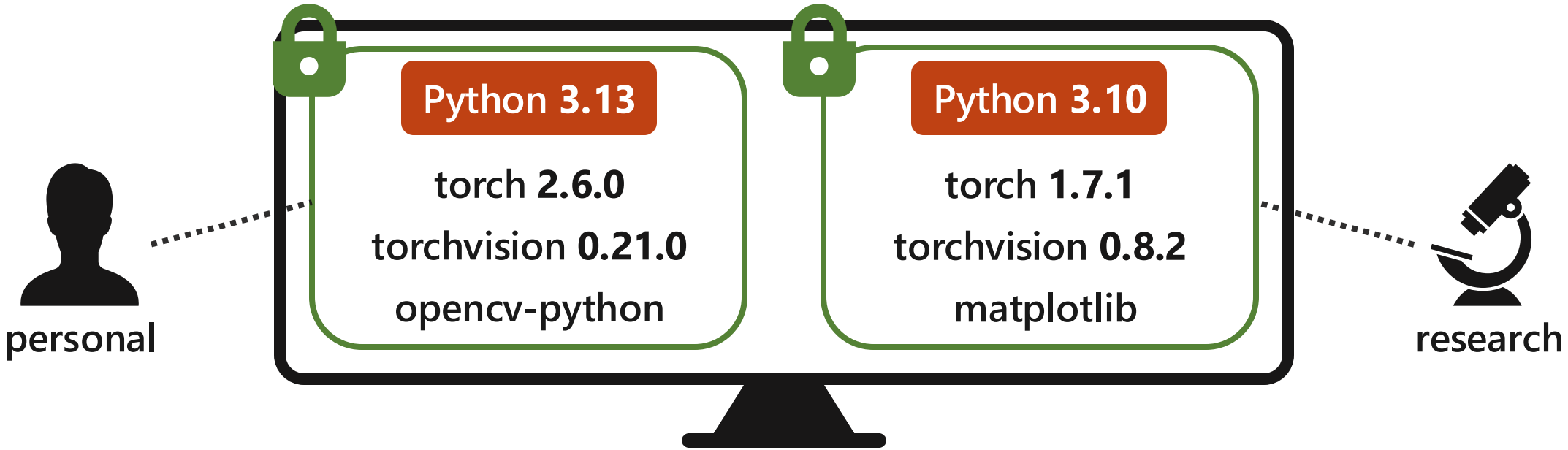
- We often have **several Python projects** on **a single computer**.
- Examples: personal, research, etc.

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**
- I recommend **uv!**
  - **Faster** and more **flexible** than Poetry, with lockfile-based reproducibility.
  - Lighter and faster than Conda, **without pip conflicts**.

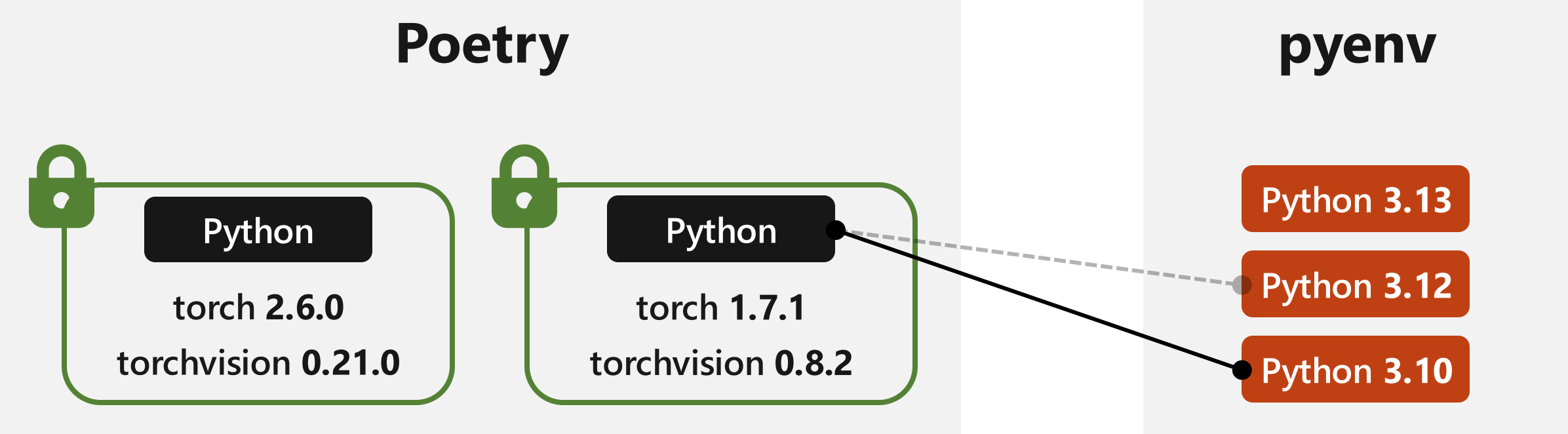
Tool Comparison

recommend

	Poetry	uv	Conda
Python version	△ (Needs pyenv)	○ (Built-in)	○ (Built-in)
Reproducibility	○ (poetry.lock)	○ (uv.lock)	○ (env.yaml)
Available libraries	○ (Full PyPI)	○ (Full PyPI)	△ (Limited)
Dependency isolation	○ (Virtual envs)	○ (Virtual envs)	△ (Shared deps)
Build automation	○ (Integrated)	○ (Scriptable)	✕ (No packaging)
Non-Python packages	✕	✕	○ (System libs)
Notes	Stable, intuitive	Fast, modern	Risky mixing with pip

Adapted from: https://cyberagentailab.github.io/BestPracticesForPythonCoding/

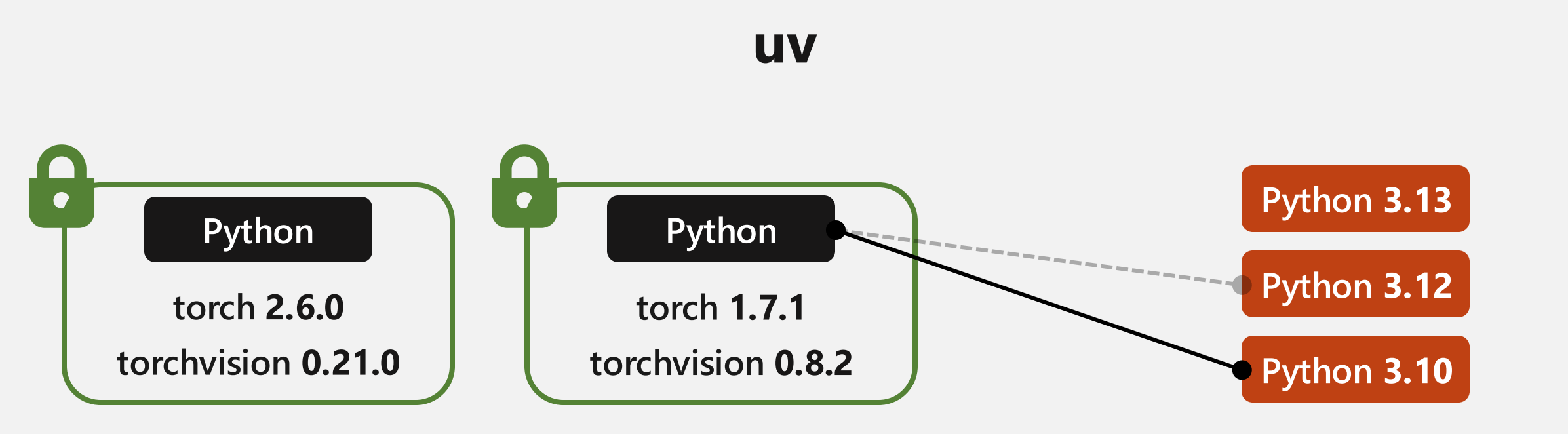
No.1 | Poetry (+pyenv)



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

Strict dependency resolution / Reproducibility / Project-oriented

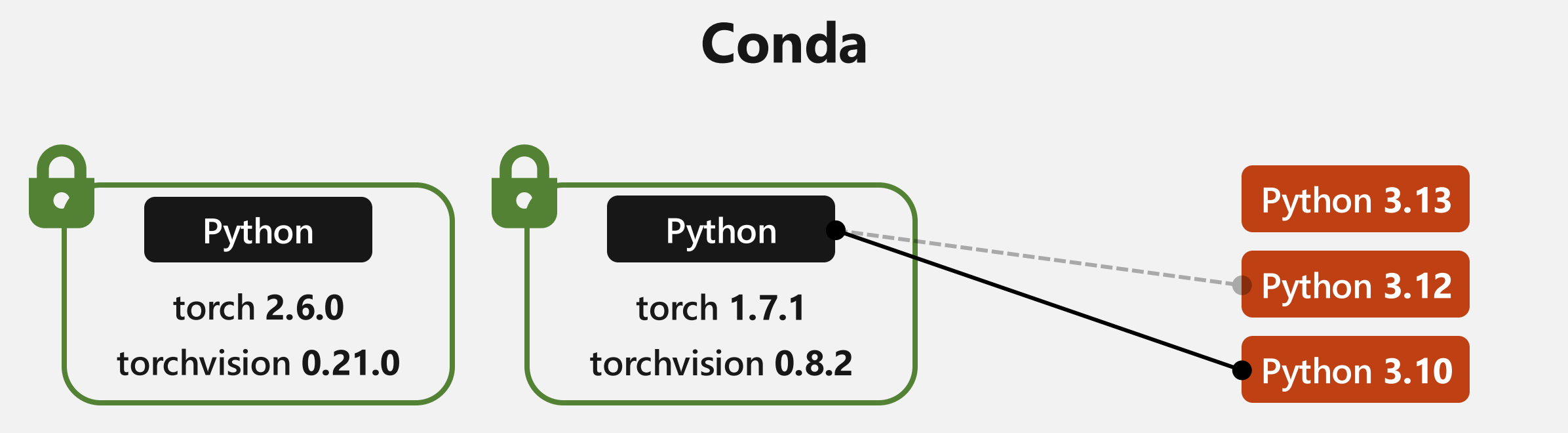
No.2 | uv



- 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.

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

  - Initialize Poetry.

```
$ 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.

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

  - Initialize pyenv.

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

No.1 | Poetry (+pyenv)

- Create your virtual environment.
  - Install and pin your desired Python version.

```
$ pyenv install 3.13.2 && pyenv local 3.13.2
```

  - 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.

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

- Deactivate your virtual environment.

```
$ 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
```

  - Initialize uv.

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

No.2 | uv

- Create your virtual environment.
  - Install and pin your desired Python version.

```
$ uv python install 3.13.2 && uv python pin 3.13.2
```

  - 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.

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

- Deactivate your virtual environment.

```
$ 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
```

- Initialize Conda.

```
$ 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.
  - Initialize your virtual environment using your desired Python version.

```
$ conda create -n {ENV_NAME} python=3.13.2
```

  - Activate your virtual environment.

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

  - Add any dependencies you like.

```
$ conda install numpy
$ pip install numpy #choose one (details in p.21)
```

No.3 | Conda

- Manage your virtual environment.
  - Deactivate your virtual environment.

```
$ conda deactivate
```

  - Export your virtual environment.

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

  - Import an existing environment.

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

Usage Tips

- Poetry Dependency Management
  - Poetry enforces **strict** version checks.
  - If a package doesn't support the **entire** Python range (e.g. ^3.13),
  - It will raise an error — even if your current Python version works.
  - ✓ **Tip**: Use a **narrow**er range like ~3.13 to avoid issues.
- Conda & Pip Compatibility
  - **Mixing** Conda and pip in the same environment can cause conflicts.
  - Pip-installed packages might not play well with Conda's dependency tracking.
  - ✓ **Tip**: Stick to either **conda install** or **pip install** in a given environment.

References

- Poetry
- pyenv
- uv (by Astral)
- Conda
- Best Practices for Python Coding (CyberAgent AI Lab)