AN UNBIASED EVALUATION OF ENVIRONMENT MANAGEMENT AND PACKAGING TOOLS

Anna-Lena Popkes

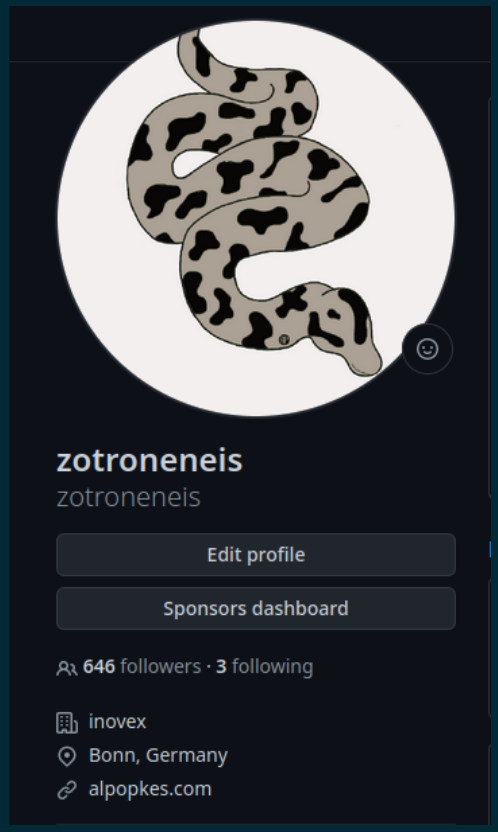
April 17, 2023

ABOUT ME

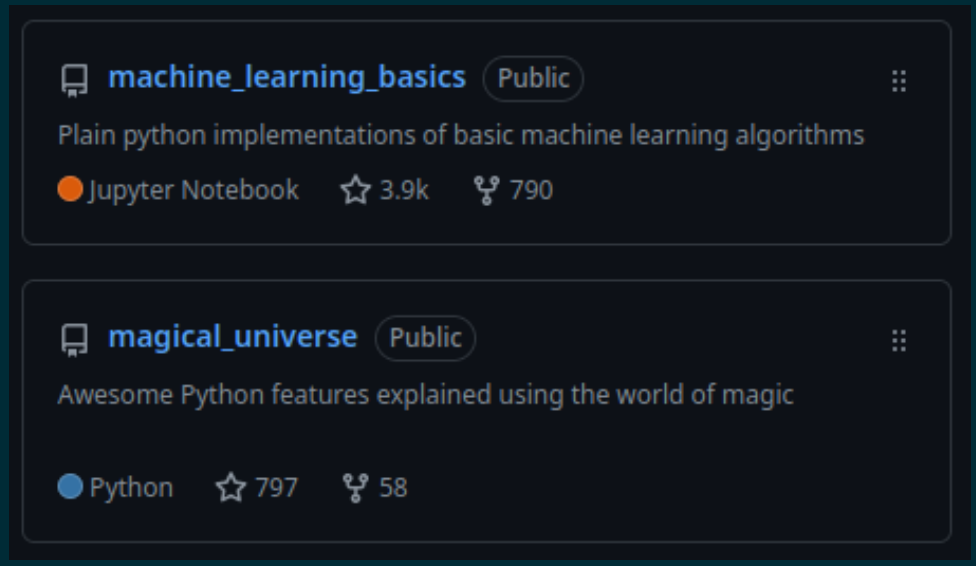
Machine Learning Engineer @inovex

inovex

- Github: zotroneneis
- Personal webpage



Github: zotroneneis

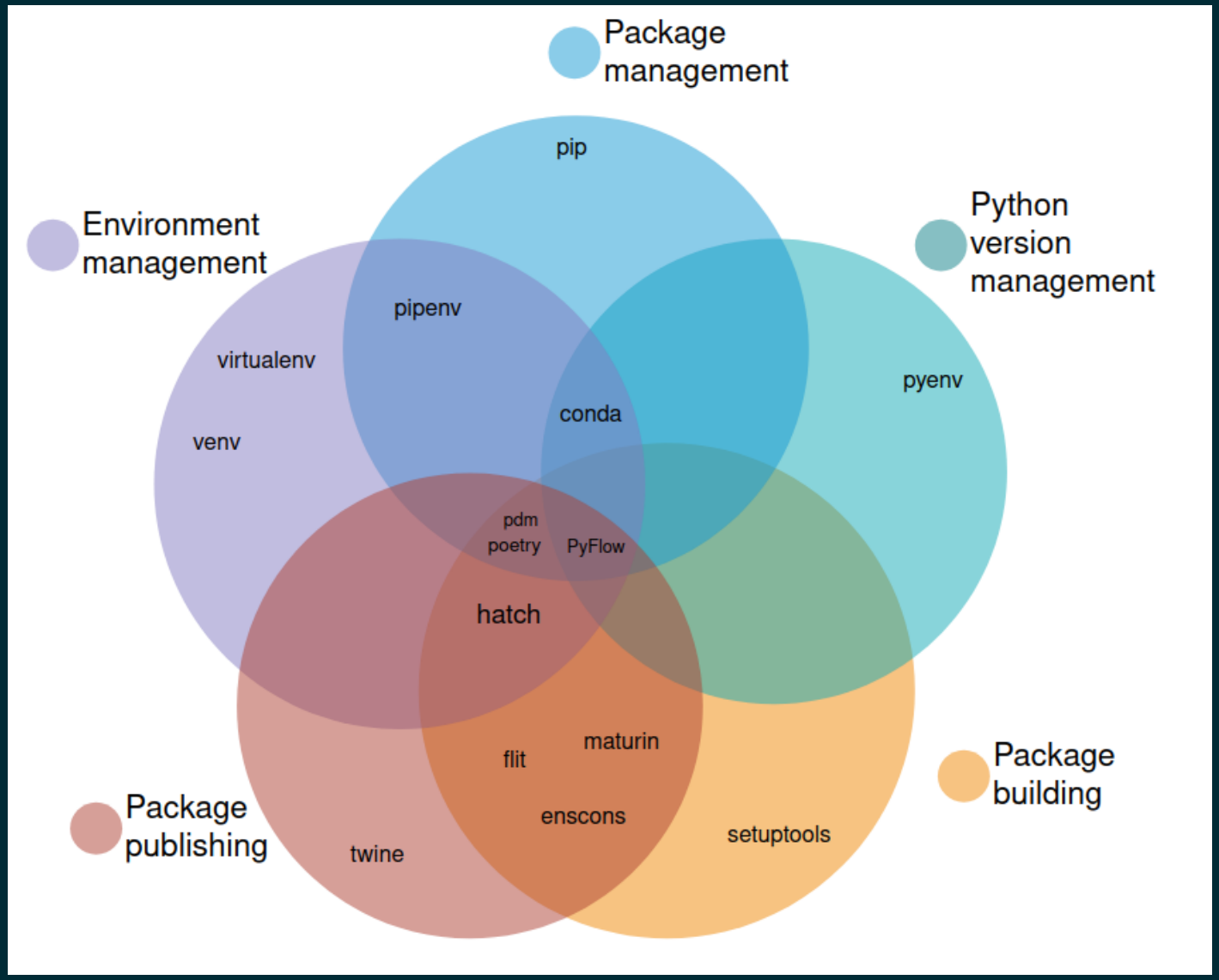


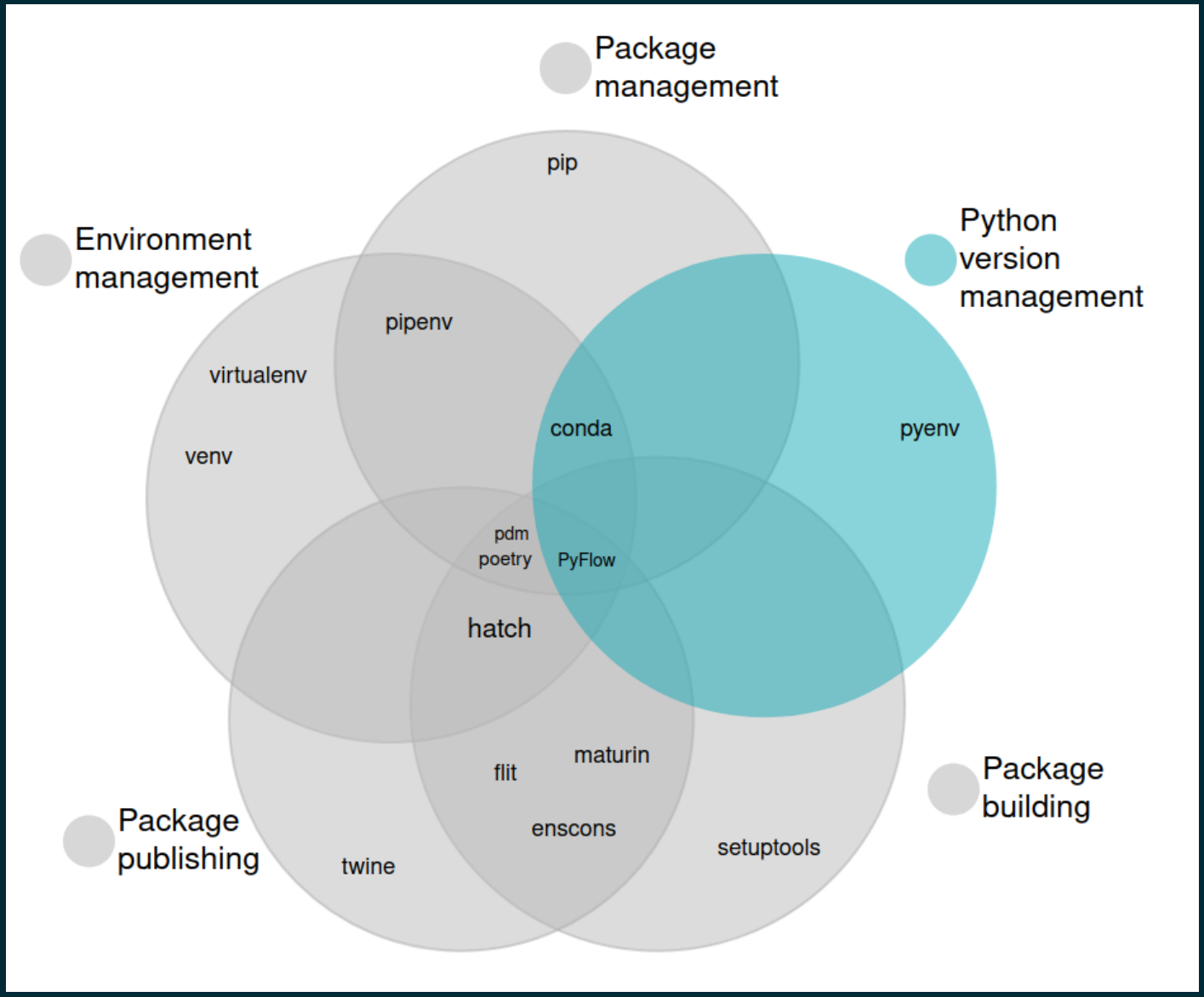
OVERVIEW

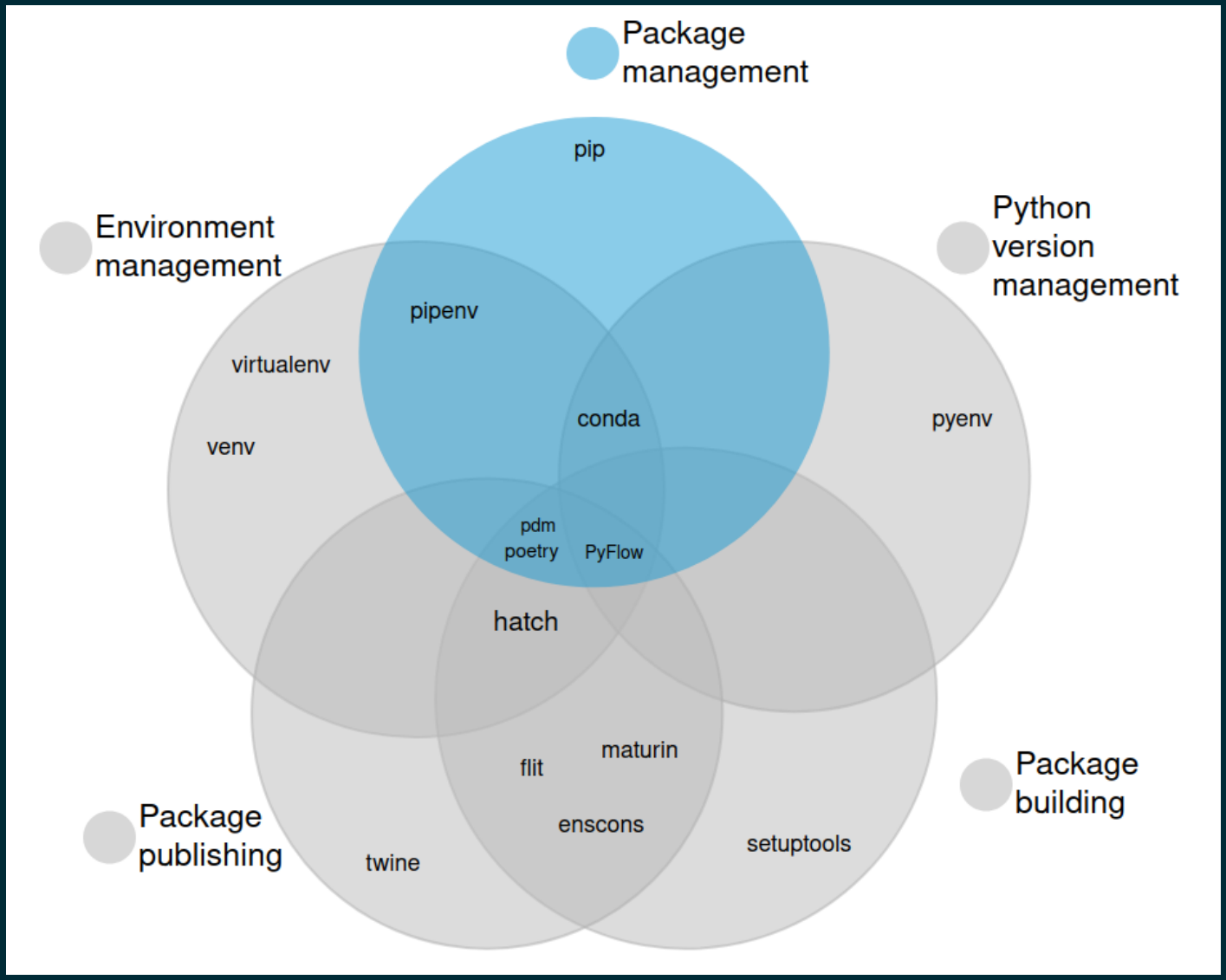
OVERVIEW

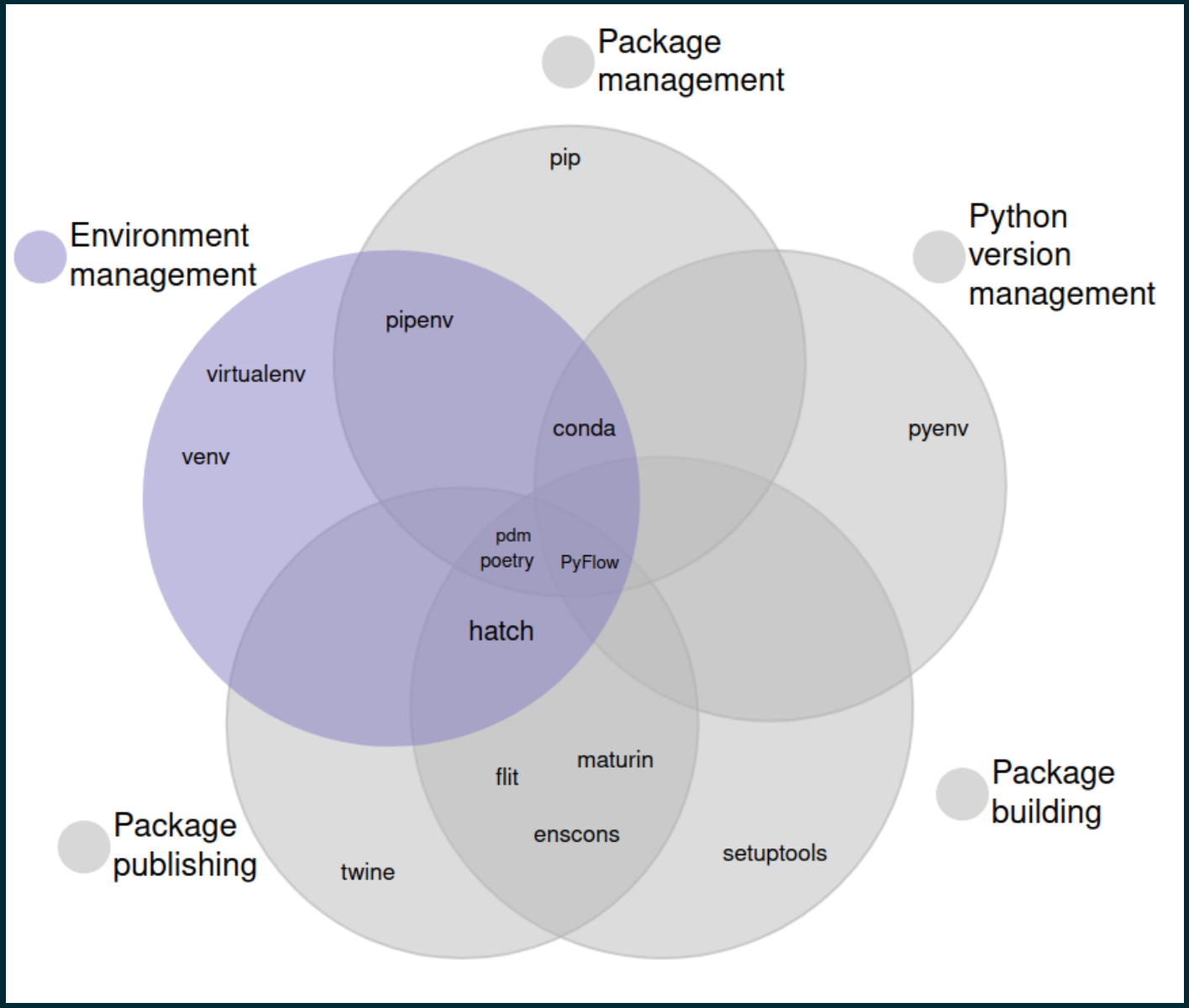
1. Goal of the talk
2. Categorization
3. Detailed look at each category
    - Definition
    - Motivation
    - Single purpose tools
4. Multi-purpose tools

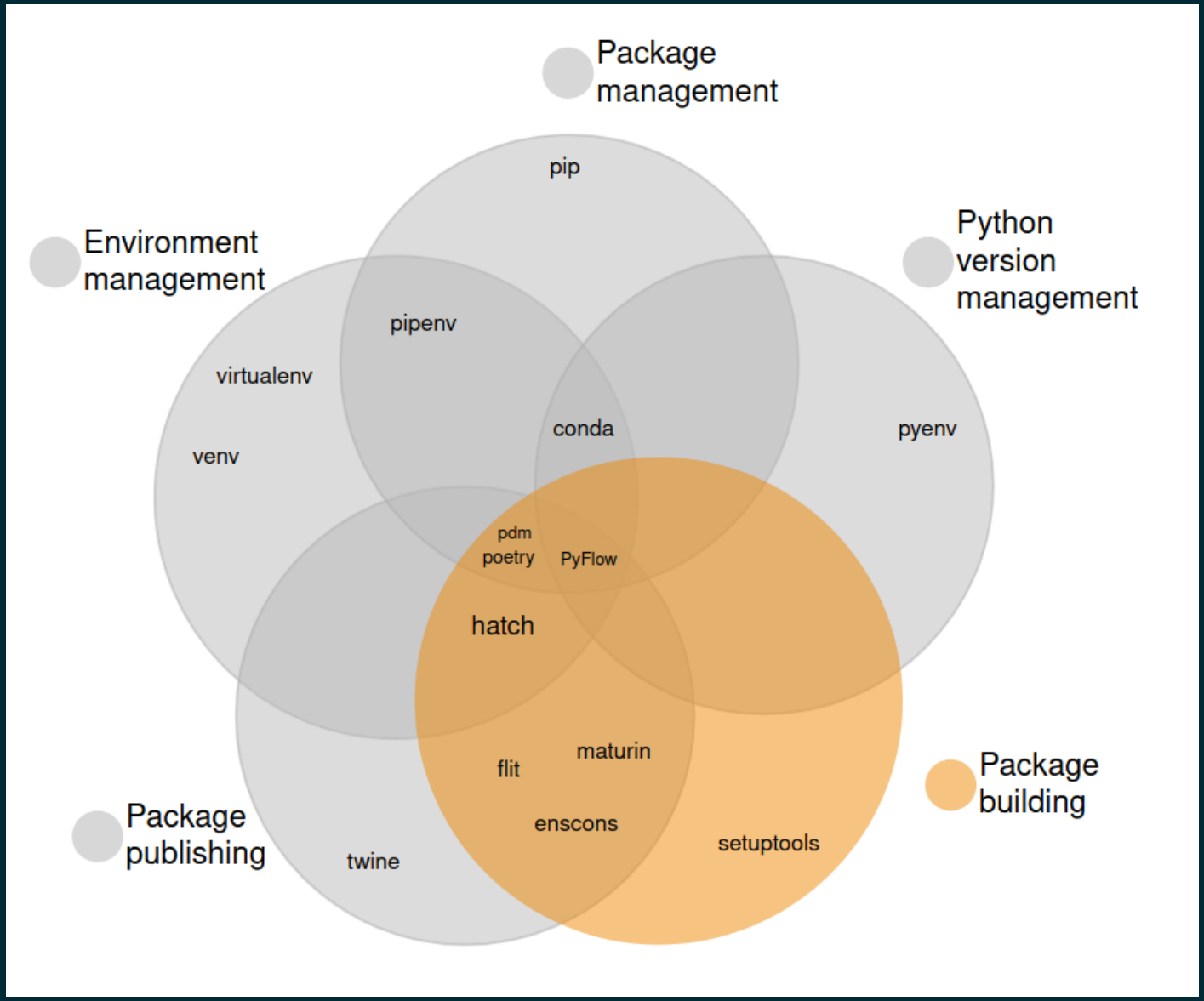
CATEGORIZATION

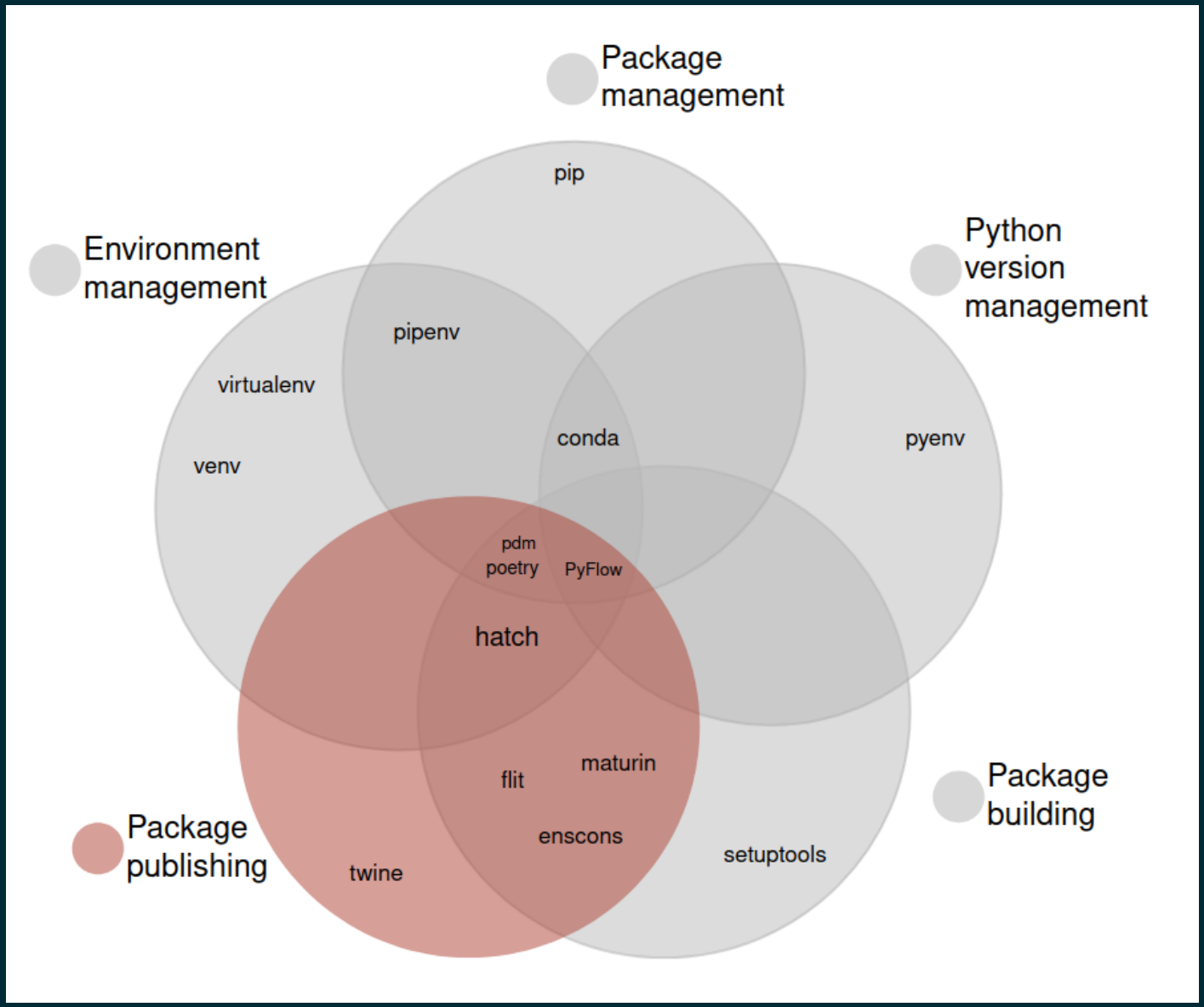

Package management
pip
Environment management
Python version management
pipenv
virtualenv
pyenv
conda
venv
pdm
poetry
PyFlow
hatch
maturin
flit
Package building
Package publishing
enscons
setuptools
twine


Package management
pip
Environment management
Python version management
pipenv
virtualenv
venv
conda
pyenv
pdm
poetry
PyFlow
hatch
flit
maturin
enscons
setuptools
twine
Package publishing
Package building


Package management
pip
Environment management
Python version management
pipenv
virtualenv
conda
pyenv
venv
pdm
poetry
PyFlow
hatch
flit
maturin
Package building
Package publishing
enscons
setuptools
twine


Package management
pip
Environment management
Python version management
pipenv
virtualenv
conda
pyenv
venv
pdm
poetry
PyFlow
hatch
flit
maturin
Package building
Package publishing
enscons
setuptools
twine


Package management
pip
Environment management
Python version management
pipenv
virtualenv
venv
conda
pyenv
pdm
poetry
PyFlow
hatch
maturin
flit
enscons
setuptools
Package building
Package publishing
twine


Package management
pip
Environment management
Python version management
pipenv
virtualenv
venv
conda
pyenv
pdm
poetry
PyFlow
hatch
flit
maturin
Package publishing
Package building
enscons
twine
setuptools

PYTHON VERSION MANAGEMENT

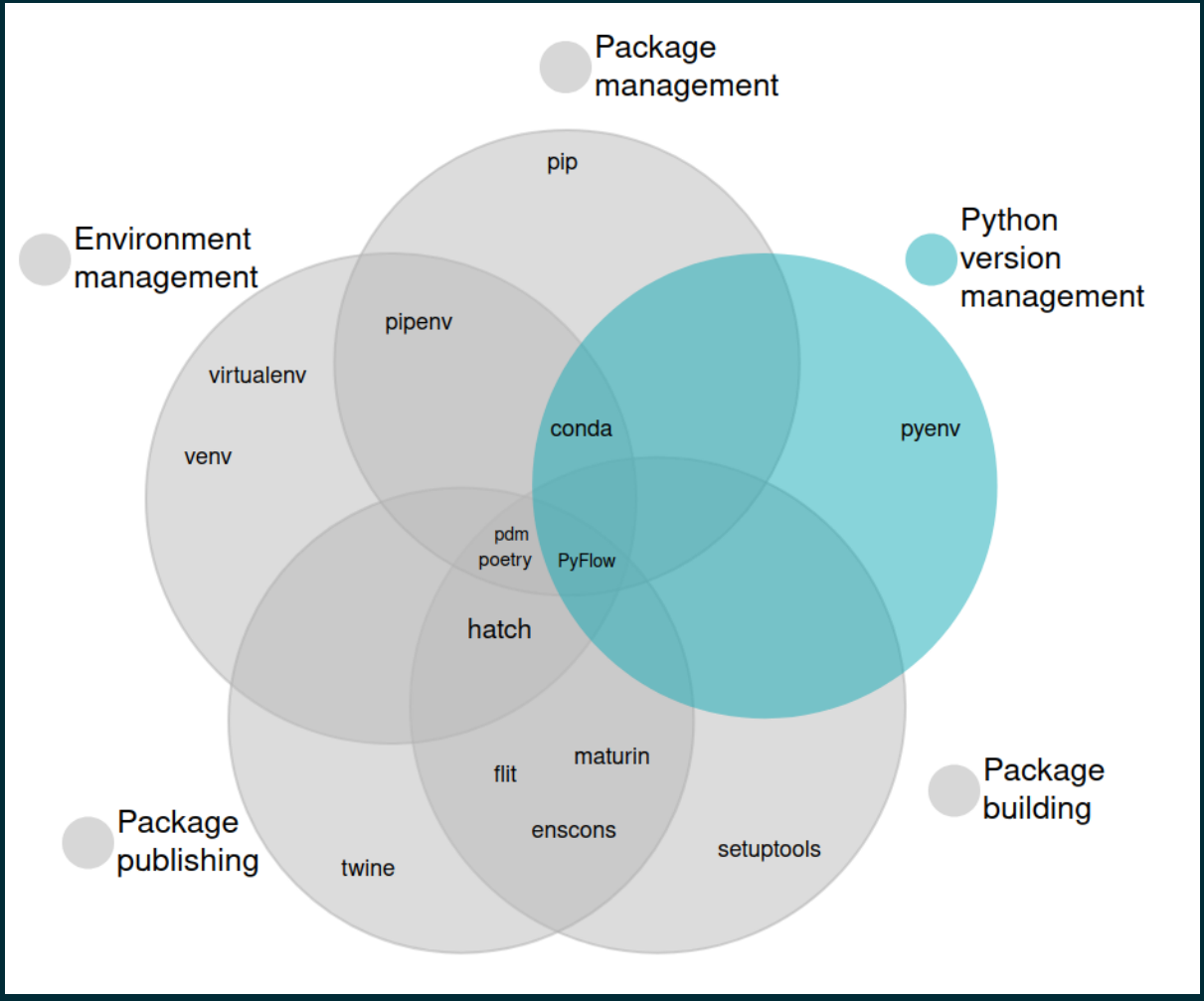
DEFINITION

- Install Python versions
- Switch between Python versions

MOTIVATION

- Projects might require different Python versions
- Projects might support several Python versions
- You might want to test new Python versions

TOOLS


Package management
pip
Environment management
Python version management
pipenv
virtualenv
venv
conda
pyenv
pdm
poetry
PyFlow
hatch
maturin
flit
enscons
Package building
Package publishing
twine
setuptools

PYENV

- Single-purpose tool to handle multiple Python versions
- Most important commands:

```
# Install specific Python version
pyenv install 3.10.4

# Switch between Python versions
pyenv shell <version>
pyenv local <version>
pyenv global <version>
```

(VIRTUAL) ENVIRONMENT MANAGEMENT

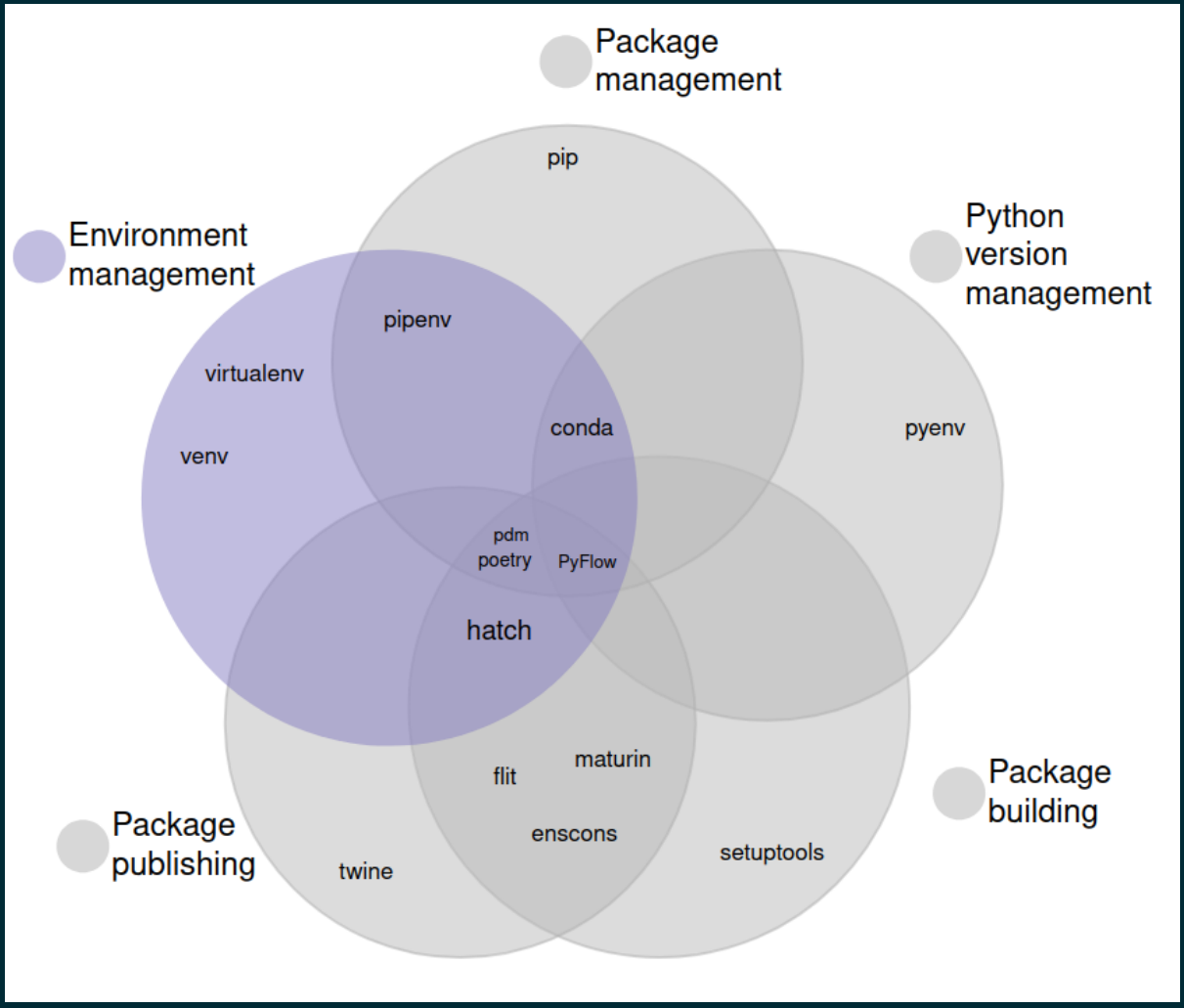
DEFINITION

Creation and management of (virtual) environments

MOTIVATION

- Projects depend on other packages
- Projects might require different versions of the same package
- `pip` installing a package can cause system pollution

TOOLS


Package management
pip
Environment management
Python version management
pipenv
virtualenv
conda
pyenv
venv
pdm
poetry
PyFlow
hatch
maturin
flit
Package building
Package publishing
enscons
setuptools
twine

VENV

- Built-in Python package for creating virtual environments
- Most important commands:

```
# Create env
python3 -m venv <env_name>

# Activate env
source <env_name>/bin/activate

# Deactivate env
deactivate
```

VIRTUALENV

- Offers more features than venv
- For example, venv is slower and not extendable
- Most important commands:

```
# Create env
virtualenv <env_name>

# Activate env
source <env_name>/bin/activate

# Deactivate env
deactivate
```

RECAP I

PYPROJECT.TOML

- Used to define project settings
- Replaces old `setup.py`
- Example file from `pandas` library:

```
[build-system]
# Minimum requirements for the build system to execute.
# See https://github.com/scipy/scipy/pull/12940 for the AIX issue.
requires = [
    "setuptools>=61.0.0",
    "wheel",
    "Cython>=0.29.32,<3",  # Note: sync with setup.py, environment.yml and asv.conf.json
    "oldest-supported-numpy>=2022.8.16",
    "versioneer[toml]"
]
# build-backend = "setuptools.build_meta"

[project]
name = 'pandas'
dynamic = [
  'version'
]
description = 'Powerful data structures for data analysis, time series, and statistics'
readme = 'README.md'
authors = [
  { name = 'The Pandas Development Team', email='pandas-dev@python.org' },
]
license = {file = 'LICENSE'}
requires-python = '>=3.8'
dependencies = [
  "numpy>=1.20.3; python_version<'3.10'",
```

PACKAGE MANAGEMENT

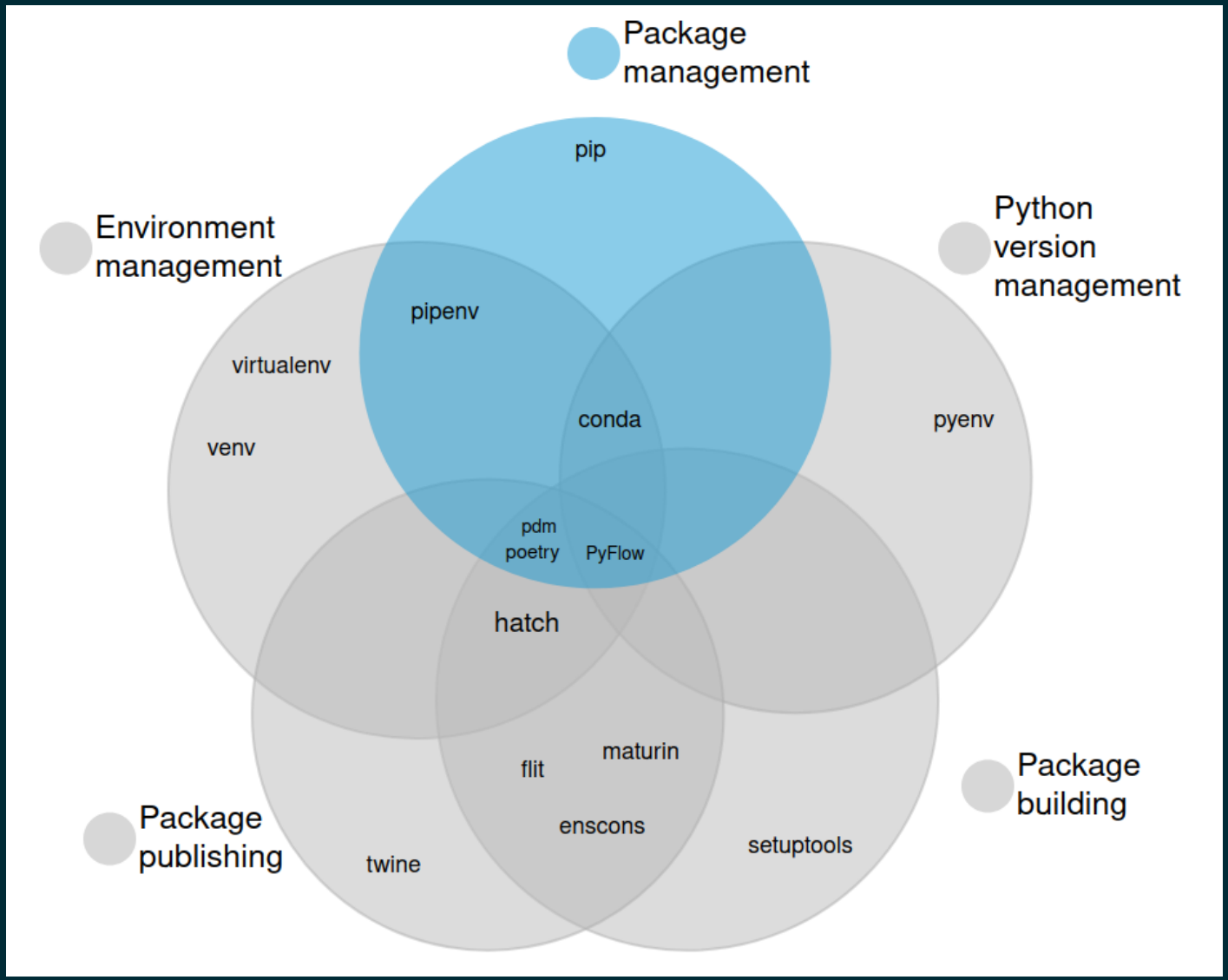
DEFINITION

Download and install libraries and their dependencies

MOTIVATION

- Packages allow us to define a hierarchy of modules
- Modules can be accessed easily using the dot-syntax
- Code can easily be shared with other developers
- Project dependencies are bundled in `pyproject.toml`

TOOLS


Package management
pip
Environment management
Python version management
pipenv
virtualenv
conda
pyenv
venv
pdm
poetry
PyFlow
hatch
maturin
flit
Package building
enscons
Package publishing
setuptools
twine

PIP

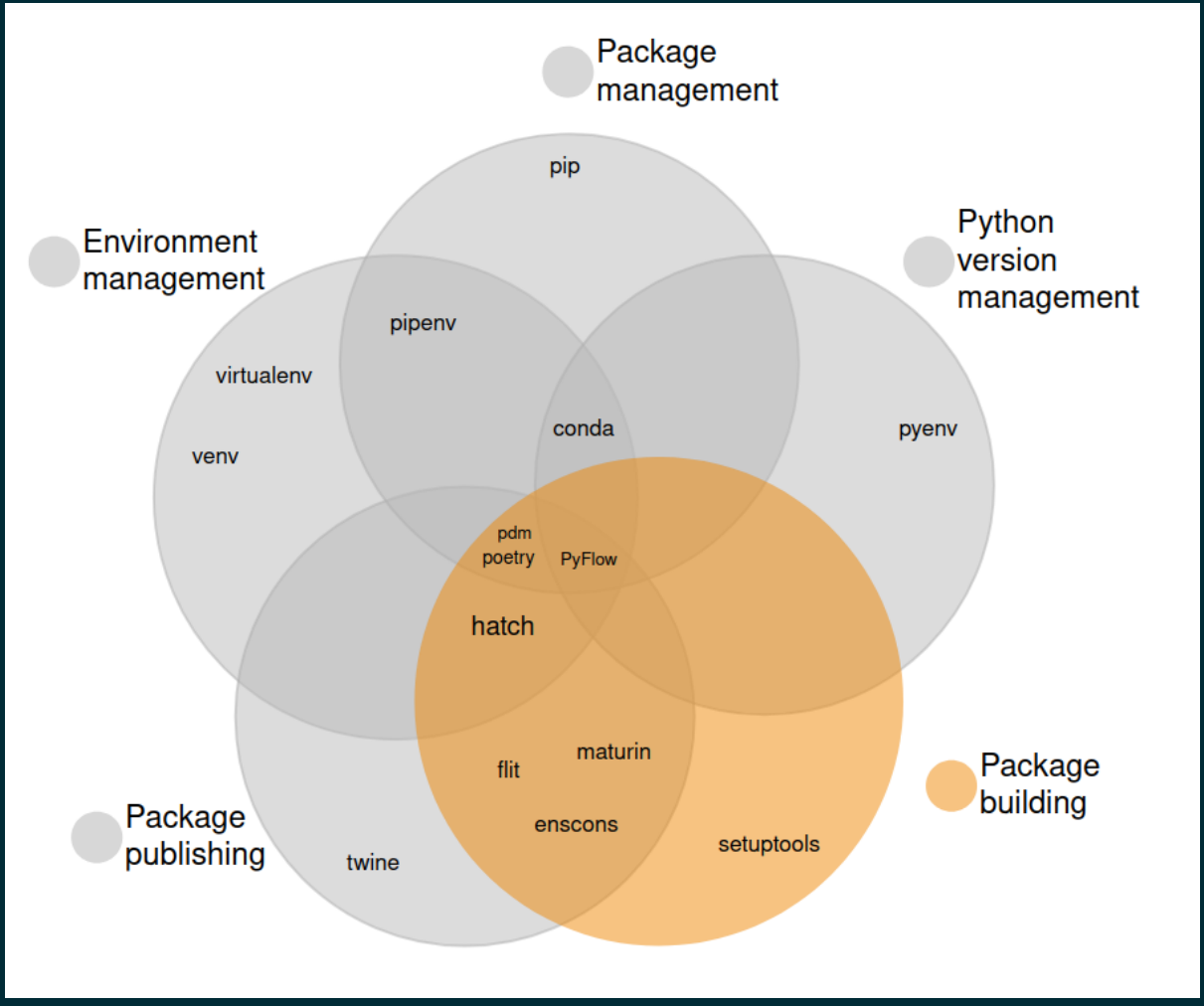
- Standard package manager for Python
- Shipped with Python
- Allows to install packages from PyPI and other indexes
- Main command: `pip install <package_name>`

PACKAGE BUILDING

DEFINITION

Building a package => creating `.whl` and `.tar.gz` files

TOOLS


Package management
pip
Environment management
Python version management
pipenv
virtualenv
venv
conda
pyenv
pdm
poetry
PyFlow
hatch
maturin
flit
enscons
Package building
Package publishing
twine
setuptools

SETUPTOOLS

- Build backend for Python packages
- Developed as an enhacement for `distutils` in 2004
- Used with a `pyproject.toml` file:

```
[build-system]
requires = ["setuptools"]
build-backend = "setuptools.build_meta"
```

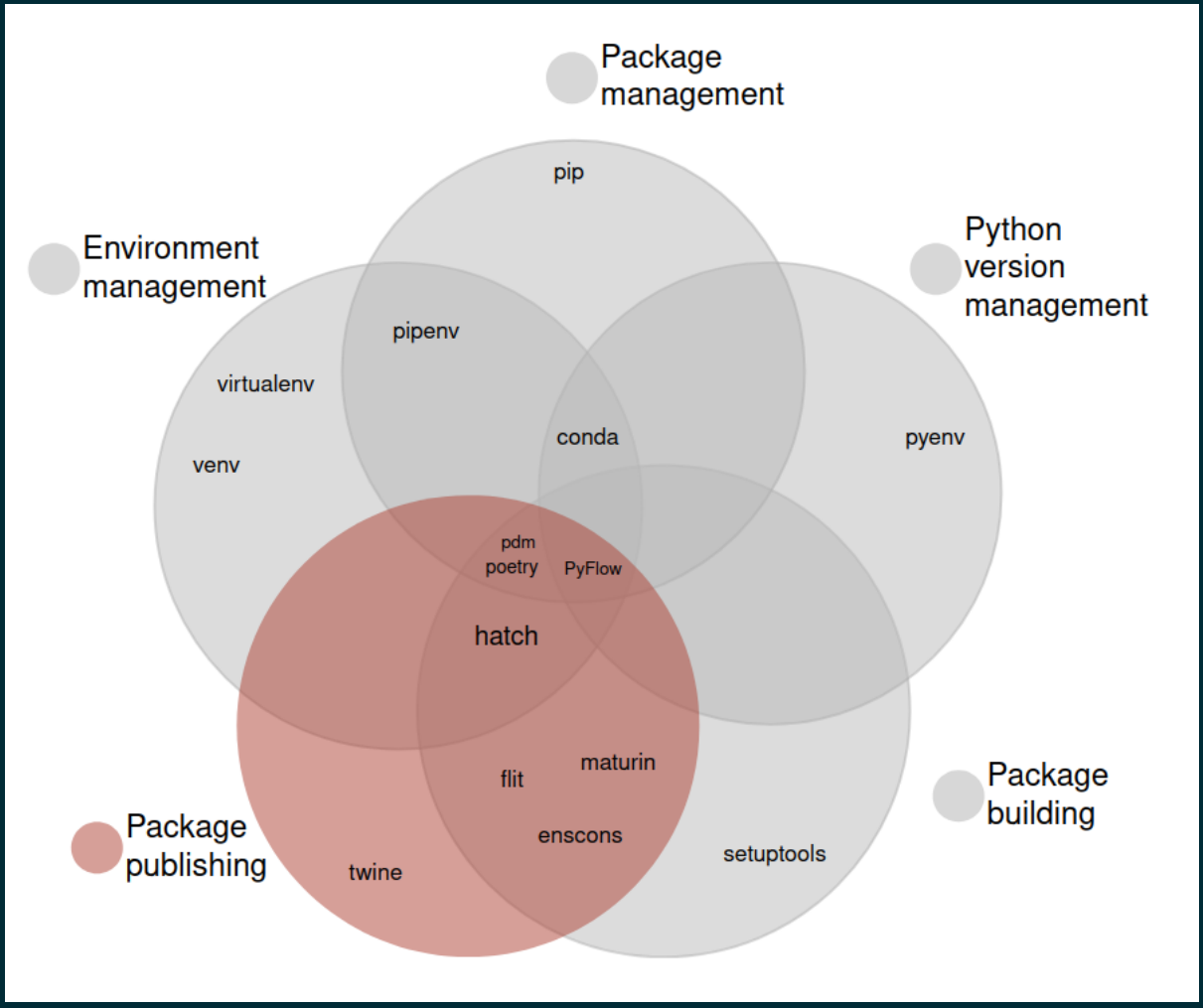
- Build package with build frontend: `python -m build`

PACKAGE PUBLISHING

DEFINITION

Publish package to PyPI or other index

TOOLS


Package management
pip
Environment management
Python version management
pipenv
virtualenv
conda
pyenv
venv
pdm
poetry
PyFlow
hatch
flit
maturin
Package publishing
Package building
enscons
twine
setuptools

TWINE

- Official PyPI tool to upload packages
- Also works with other indexes
- Single-purpose tool
- Main command: `twine upload dist/*`

RECAP II

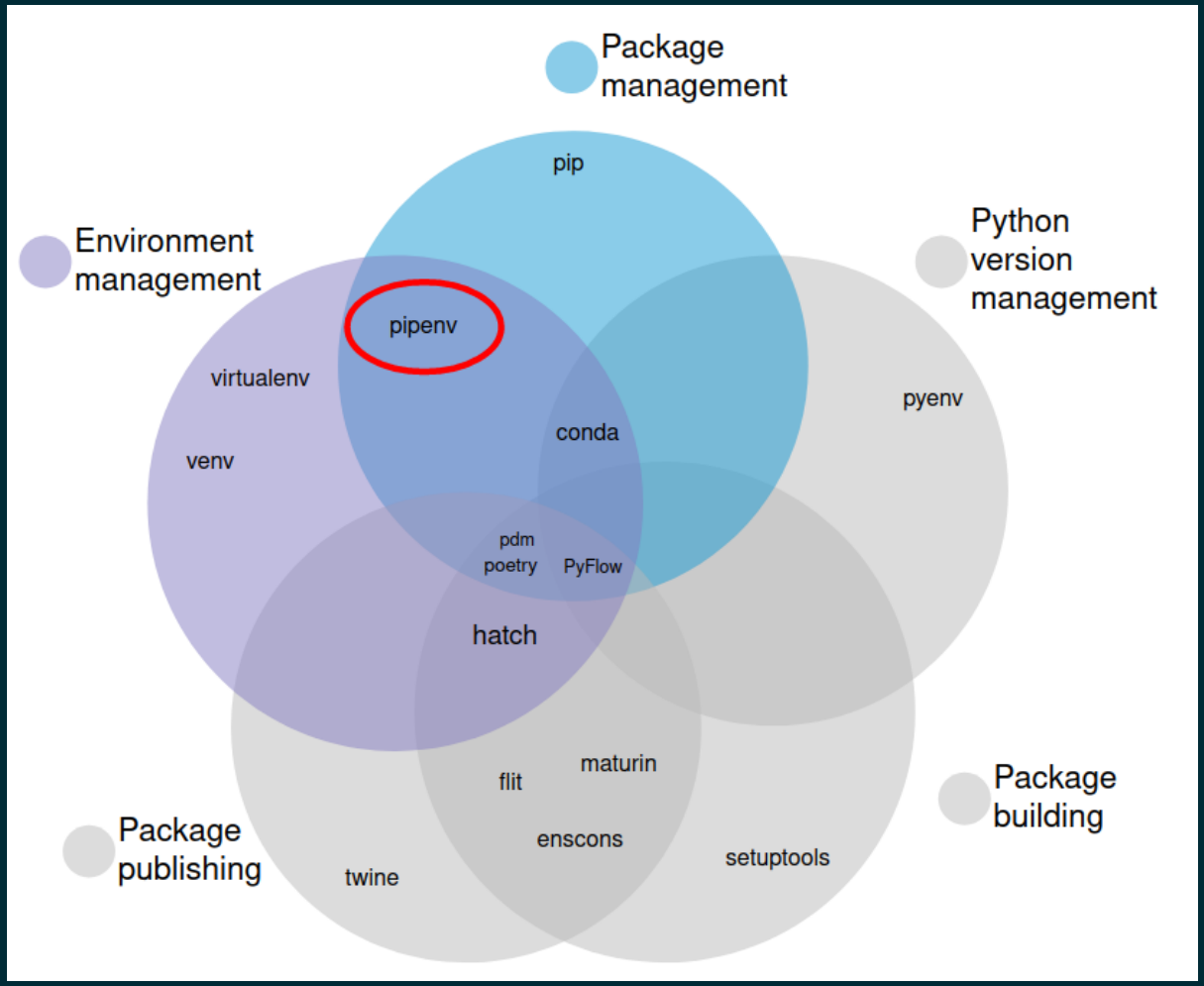
LOCK FILE

- Records exact versions of all dependencies installed for a project
- This enables reproducability of projects across multiple platforms
- Example file from `poetry`:

```
# This file is automatically @generated by Poetry and should not be changed by hand.

[[package]]
name = "attrs"
version = "22.1.0"
description = "Classes Without Boilerplate"
category = "main"
optional = false
python-versions = ">=3.5"
files = [
    {file = "attrs-22.1.0-py2.py3-none-any.whl", hash = "sha256:86efa402f67bf2df34f51a335487cf46b1ec130d02b8d39fd248abfd30da551c"},
    {file = "attrs-22.1.0.tar.gz", hash = "sha256:29adc2665447e5191d0e7c568fde78b21f9672d344281d0c6e1ab085429b22b6"},
]

[package.extras]
dev = ["cloudpickle", "coverage[toml] (>=5.0.2)", "furo", "hypothesis", "mypy (>=0.900,!=0.940)", "pre-commit", "pympler", "pytest (>=4.3.0)", "pytest-mypy-plug
docs = ["furo", "sphinx", "sphinx-notfound-page", "zope.interface"]
tests = ["cloudpickle", "coverage[toml] (>=5.0.2)", "hypothesis", "mypy (>=0.900,!=0.940)", "pympler", "pytest (>=4.3.0)", "pytest-mypy-plugins", "zope.interfac
tests-no-zope = ["cloudpickle", "coverage[toml] (>=5.0.2)", "hypothesis", "mypy (>=0.900,!=0.940)", "pympler", "pytest (>=4.3.0)", "pytest-mypy-plugins"]

[[package]]
name = "backports-cached-property"
version = "1.0.2"
description = "cached_property() - computed once per instance, cached as attribute"
category = "main"
optional = false
python-versions = ">=3.6.0"
files = [
    {file = "backports.cached-property-1.0.2.tar.gz", hash = "sha256:9306f9eed6ec55fd156ace6bc1094e2c86fae5fb2bf07b6a9c00745c656e75dd"},
    {file = "backports.cached_property-1.0.2-py3-none-any.whl", hash = "sha256:baeb28e1cd619a3c9ab8941431fe34e8490861fb998c6c4590693d50171db0cc"},
]
```

MULTI-PURPOSE TOOLS

PIPENV

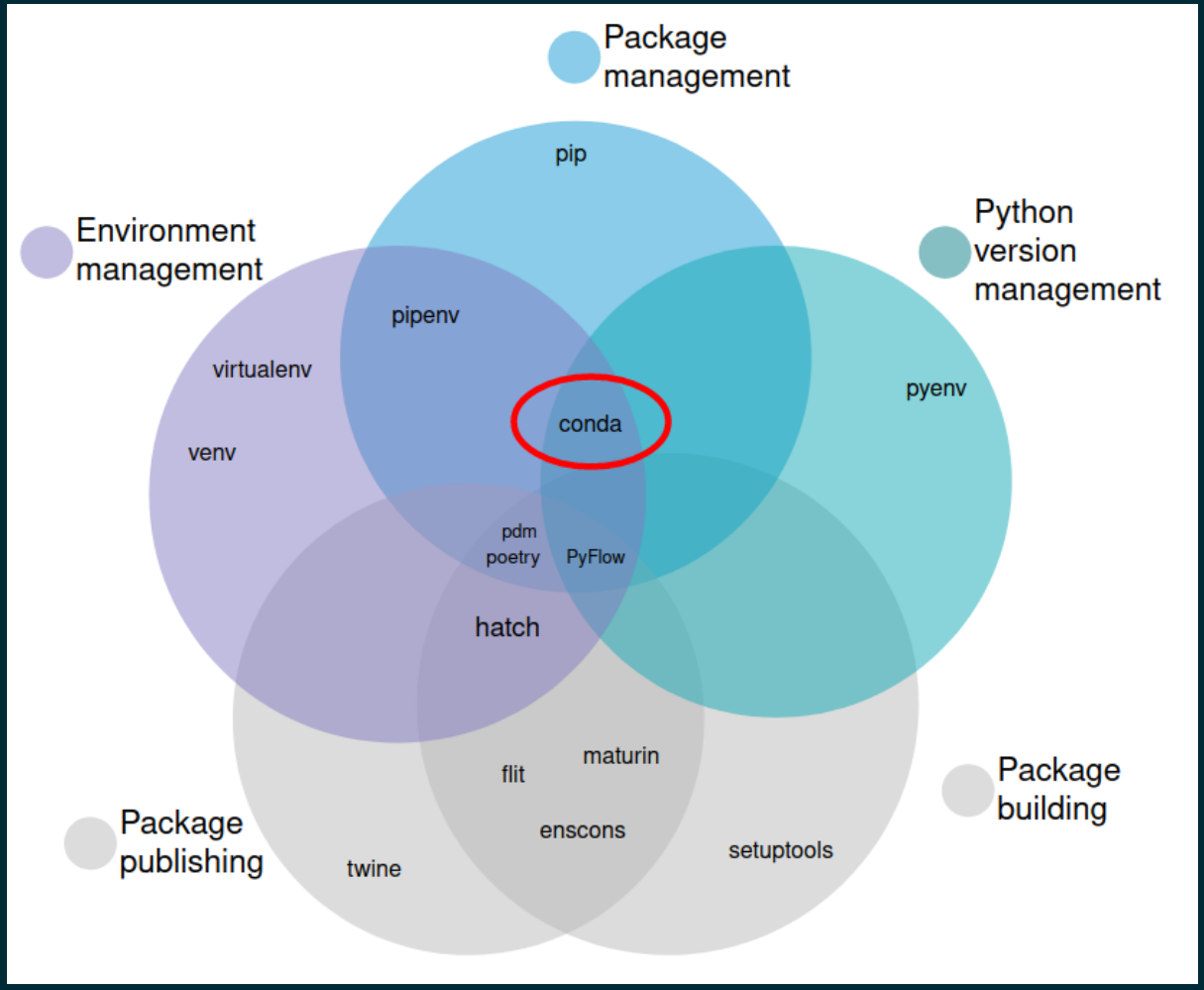

Package management
pip
Environment management
Python version management
pipenv
virtualenv
pyenv
conda
venv
pdm
poetry
PyFlow
hatch
maturin
flit
Package building
Package publishing
enscons
setuptools
twine

PIPENV

- Combines `pip` and `virtualenv`
- Introduces two additional files:
  - `Pipfile`
  - `Pipfile.lock` (replaces `requirements.txt`)
- Most important commands:

```
# Install package
pipenv install <package_name>

# Run Python script within virtual env
pipenv run <script_name.py>

# Activate virtual env
pipenv shell
```

CONDA


Package management
pip
Environment management
Python version management
pipenv
virtualenv
pyenv
conda
venv
pdm
poetry
PyFlow
hatch
maturin
flit
Package building
enscons
Package publishing
twine
setuptools

CONDA

- General-purpose package management system
- Uses its own index for packages

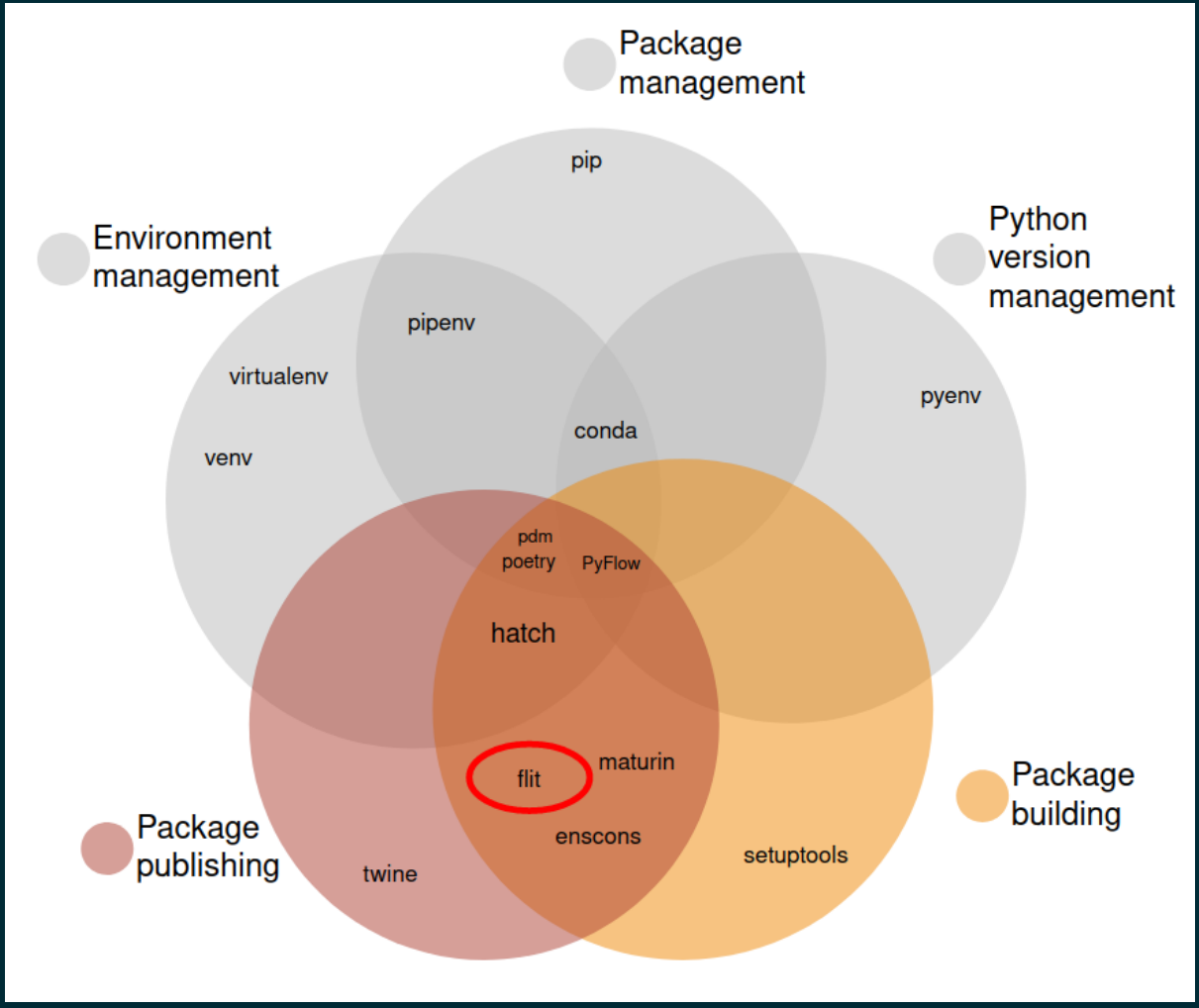
PACKAGING TOOLS

EVALUATION FEATURES

Manages dependencies	?
Resolves/locks dependencies	?
Clean build/publish flow	?
Allows to use plugins	?
Supports PEP 660 (editable installs)	?
Supports PEP 621 (project metadata)	?

FLIT

FLIT


Package management
pip
Environment management
Python version management
pipenv
virtualenv
pyenv
conda
venv
pdm
poetry
PyFlow
hatch
flit
maturin
Package building
Package publishing
enscons
twine
setuptools

FLIT - CAPABILITIES

- Python version management: ❌
- Package management: ❌
- Environment management: ❌
- Building a package: ✅
- Publishing a package: ✅

FLIT

- Specifically for pure Python packages
- Cares only about packaging

FLIT - FEATURE EVALUATION

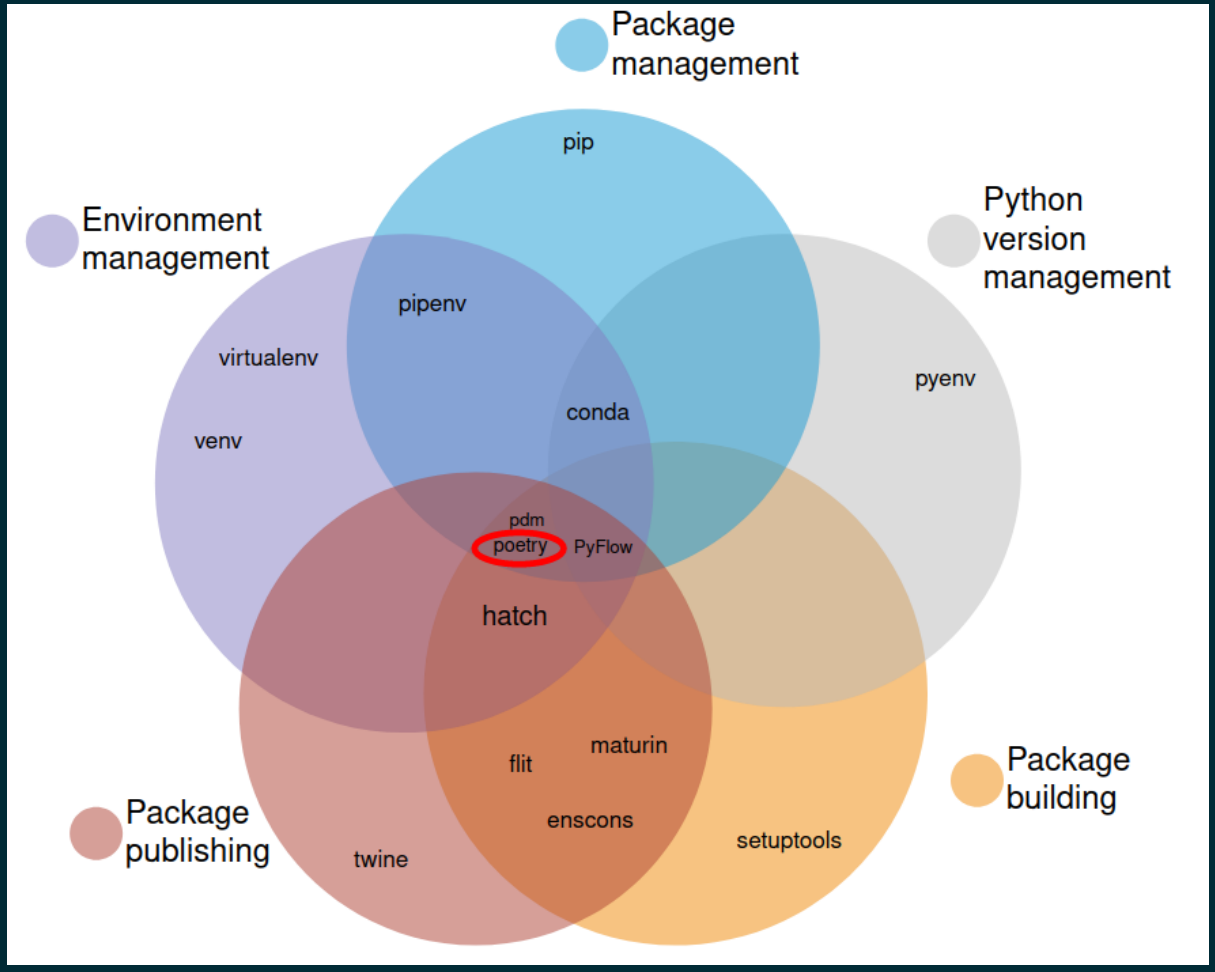
Manages dependencies	❌
Resolves/locks dependencies	❌
Clean build/publish flow	✅
Allows to use plugins	❌
Supports PEP 660 (editable installs)	✅
Supports PEP 621 (project metadata)	✅

MAIN COMMANDS

```
# Create new pyproject.toml
flit init

# Build and publish
flit publish
```

POETRY

POETRY


Package management
pip
Environment management
Python version management
pipenv
virtualenv
pyenv
conda
venv
pdm
poetry
PyFlow
hatch
maturin
flit
Package building
Package publishing
enscons
setuptools
twine

CAPABILITIES

- Python version management: ❌
- Package management: ✅
- Environment management: ✅
- Building a package: ✅
- Publishing a package: ✅

FEATURE EVALUATION

Manages dependencies	✅
Resolves/locks dependencies	✅
Clean build/publish flow	✅
Allows to use plugins	✅
Supports PEP 660 (editable installs)	✅
Supports PEP 621 (project metadata)	❌

MAIN COMMANDS

```
# Create directory structure and pyproject.toml
poetry new <project_name>

# Create pyproject.toml interactively
poetry init

# Install package from pyproject.toml
poetry install
```

DEPENDENCY MANAGEMENT

```
# Add dependency
poetry add <package_name>

# Display all dependencies
poetry show --tree
```

RUNNING CODE

```
# Activate virtual env
poetry shell

# Run script within virtual env
poetry run python <script_name.py>
```

LOCK FILE

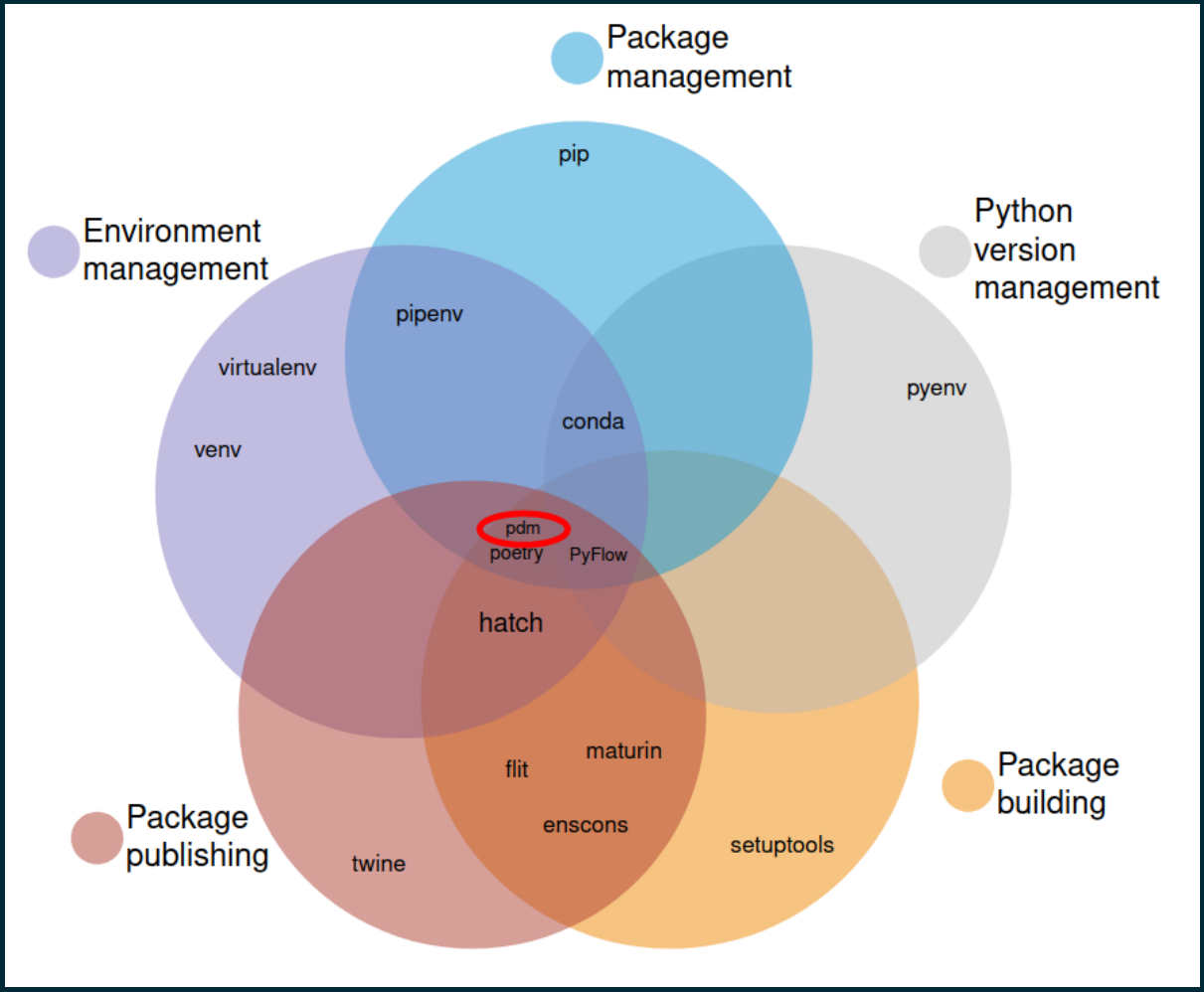
- When installing package, poetry resolves its dependencies and creates `poetry.lock`
- Updating dependencies to latest versions with `poetry update`

BUILD/PUBLISH FLOW

```
# Package code (creates `.tar.gz` and `.whl` files)
poetry build

# Publish to PyPi
poetry publish
```

PDM

PDM


Package management
pip
Environment management
Python version management
pipenv
virtualenv
pyenv
conda
venv
pdm
poetry
PyFlow
hatch
maturin
flit
Package building
Package publishing
enscons
setuptools
twine

PDM - CAPABILITIES

- Python version management: ❌
- Package management: ✅
- Environment management: ✅
- Building a package: ✅
- Publishing a package: ✅

PDM

- Strongly inspired by `poetry` and `pyflow`
- Requires Python 3.7 or higher
- Implements PEP 582 (local packages)

PDM - FEATURE EVALUATION

Manages dependencies	✅
Resolves/locks dependencies	✅
Clean build/publish flow	✅
Allows to use plugins	✅
Supports PEP 660 (editable installs)	✅
Supports PEP 621 (project metadata)	✅

MAIN COMMANDS

```
# Create pyproject.toml interactively
pdm init

# Install package from pyproject.toml
pdm install
```

DEPENDENCY MANAGEMENT

```
# Add dependency
pdm add <package_name>

# Display all dependencies
pdm list --graph
```

RUNNING CODE

```
# No pdm shell command

# Run script within env
pdm run python <script_name.py>
```

LOCK FILE

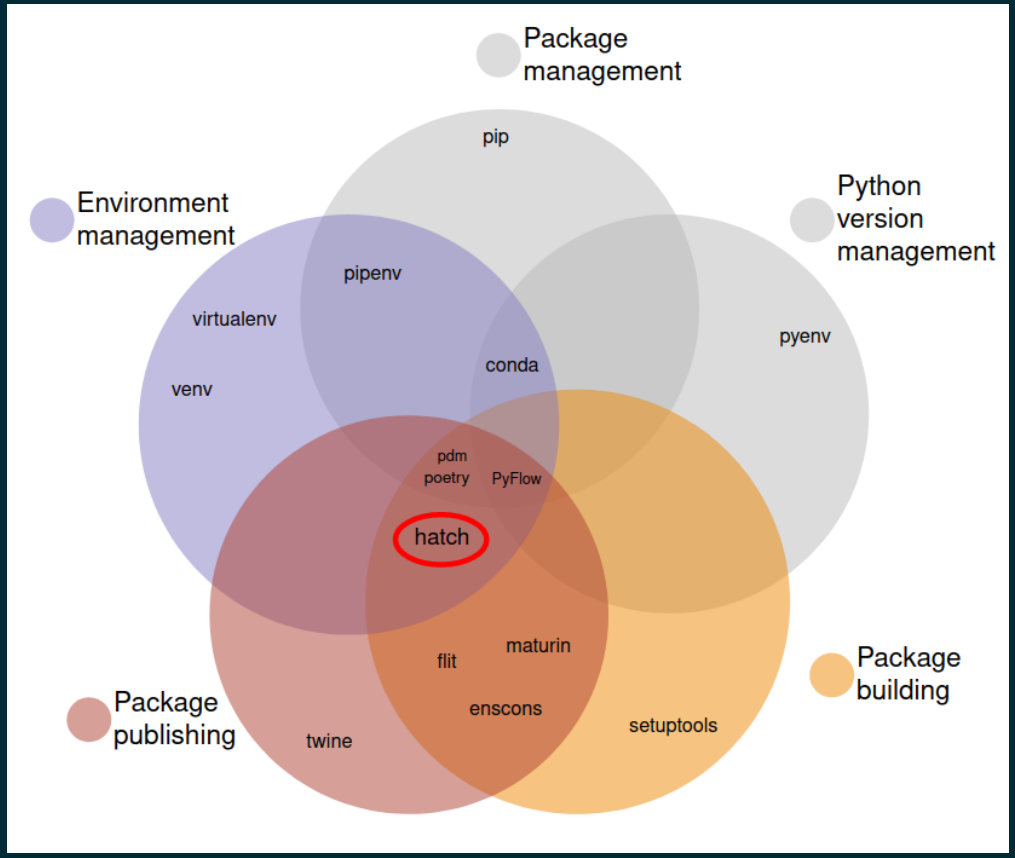
- Similar to poetry
- When installing package, pdm resolves its dependencies and creates `pdm.lock`
- Updating dependencies to latest versions with `pdm update`

BUILD/PUBLISH FLOW

```
# Package code (creates `.tar.gz` and `.whl` files)
pdm build

# Publish to PyPi
pdm publish
```

HATCH

HATCH


Package management
pip
Environment management
Python version management
pipenv
virtualenv
pyenv
conda
venv
pdm
poetry
PyFlow
hatch
maturin
flit
Package building
Package publishing
enscons
setuptools
twine

HATCH - CAPABILITIES

- Python version management: ❌
- Package management: ❌
- Environment management: ✅
- Building a package: ✅
- Publishing a package: ✅

HATCH - FEATURE EVALUATION

Manages dependencies	❌
Resolves/locks dependencies	❌
Clean build/publish flow	✅
Allows to use plugins	✅
Supports PEP 660 (editable installs)	✅
Supports PEP 621 (project metadata)	✅

CREATING A NEW PROJECT

```
# Create directory structure and pyproject.toml
hatch new <project_name>

# Interactive mode
hatch new -i <project_name>

# Initialize existing project / create pyproject.toml
hatch new --init
```

DEPENDENCY MANAGEMENT

```
# Packages are added manually to pyproject.toml
hatch add <package_name> # This command doesn't exist!

# Display dependencies
hatch dep show table
```

RUNNING CODE

```
# Activate virtual env
hatch shell

# Run script within virtual env
hatch run python <script_name.py>
```

BUILD/PUBLISH FLOW

```
# Package code (creates `.tar.gz` and `.whl` files)
hatch build

# Publish to PyPi
hatch publish
```

DECLARATIVE ENVIRONMENT MANAGEMENT

- Environments can be configured within `pyproject.toml`
- We can define scripts for an environment
- Example use case: code formatting

SUMMARY - PACKAGING TOOLS

	`flit`	`poetry`	`pdm`	`hatch`
Manages dependencies	❌	✅	✅	❌
Resolves/locks dependencies	❌	✅	✅	❌
Clean build/publish flow	✅	✅	✅	✅
Supports plugins	❌	✅	✅	✅
PEP 660 (editable installs)	✅	✅	✅	✅
PEP 621 (project metadata)	✅	❌	✅	✅

THE END