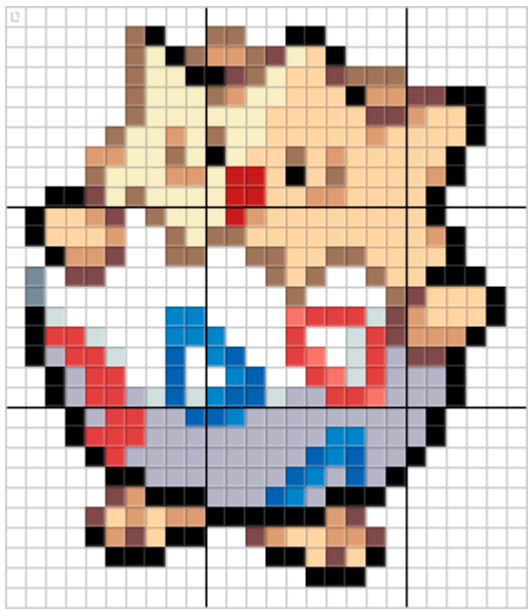
Avaliando gerenciadores de pacotes Python

Ítalo Epifânio

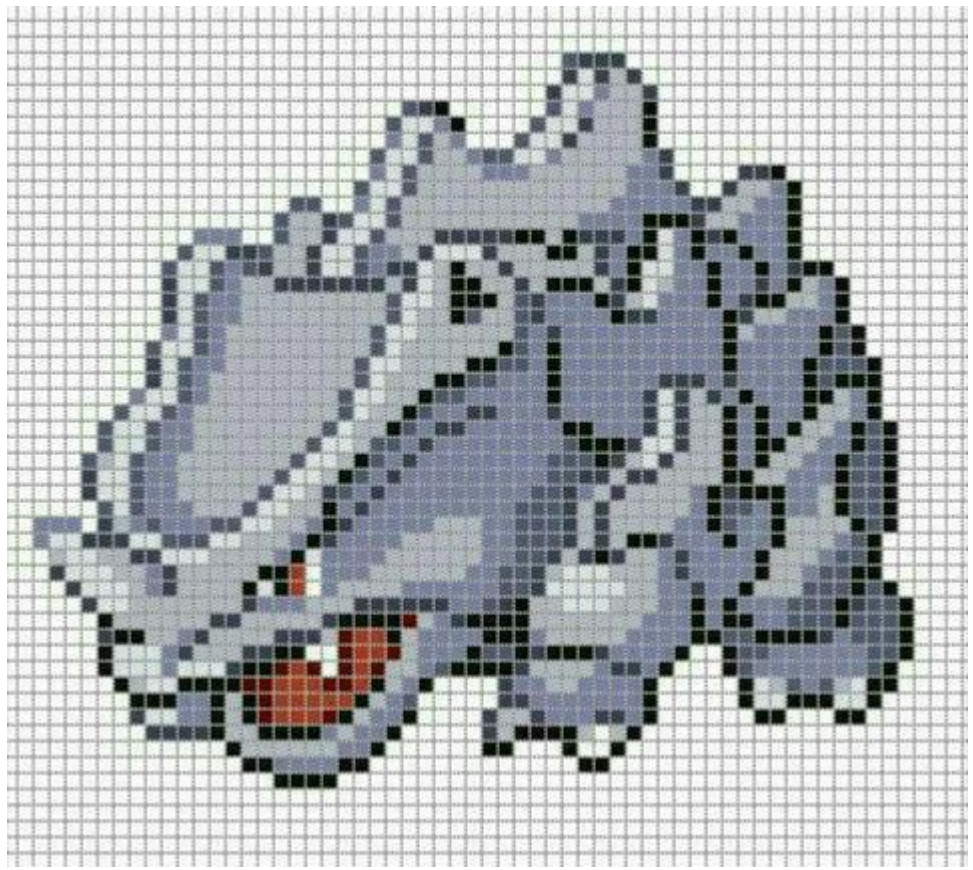
O que é ferramenta e o que é pokemon?

- Hatch
- UV
- Togepy
- Pipenv
- pip-tools
- Poetry
- PDM
- Rhyhorn
- Rye
- Pyflow
- Pixi
- Mamba

Pokemons



Togepi



Rhyhorn

Figure 1: Acertou quem disse Togepy e Rhyhorn :)

Ferramentas

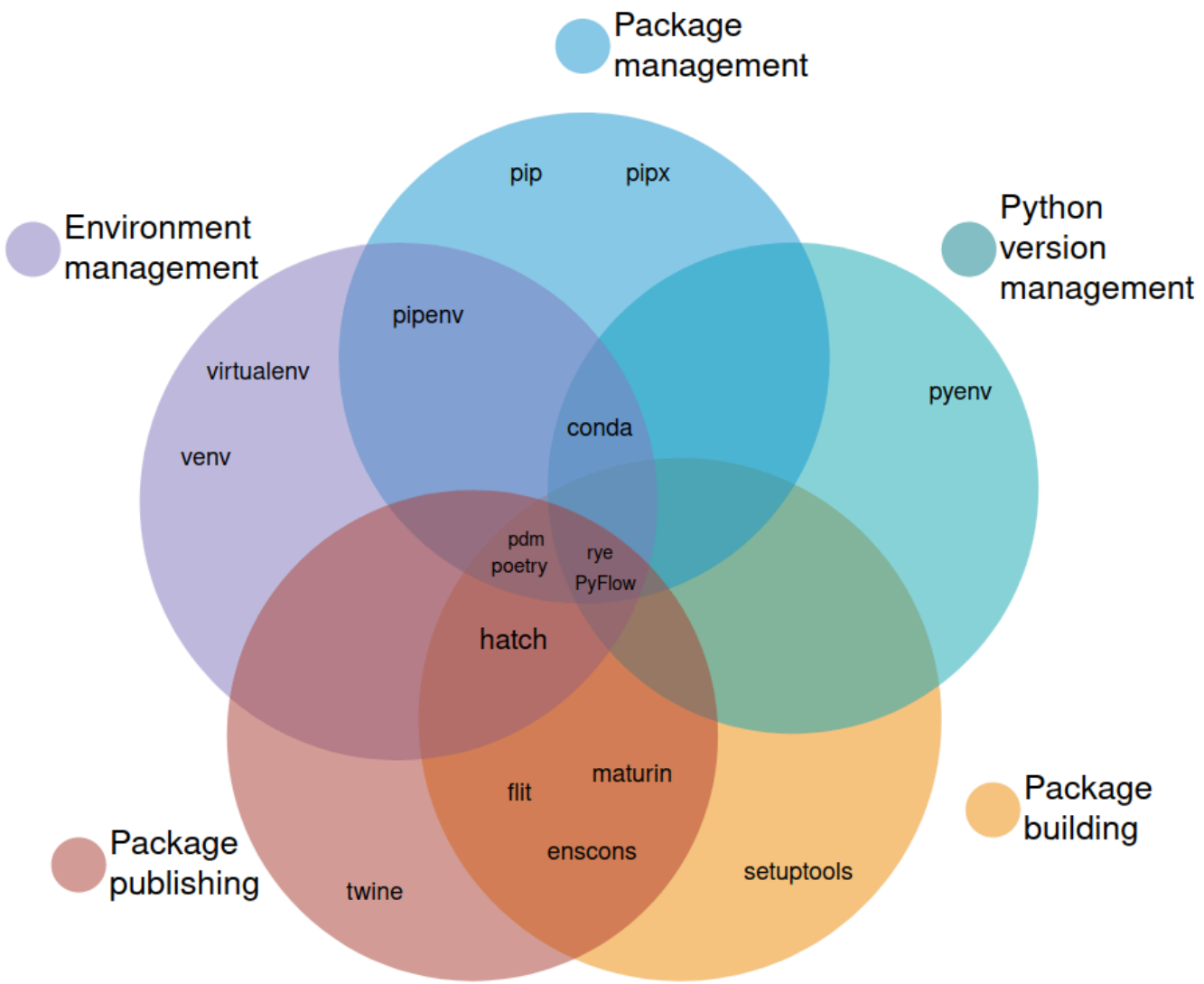


Diagrama de Venn

Ferramentas

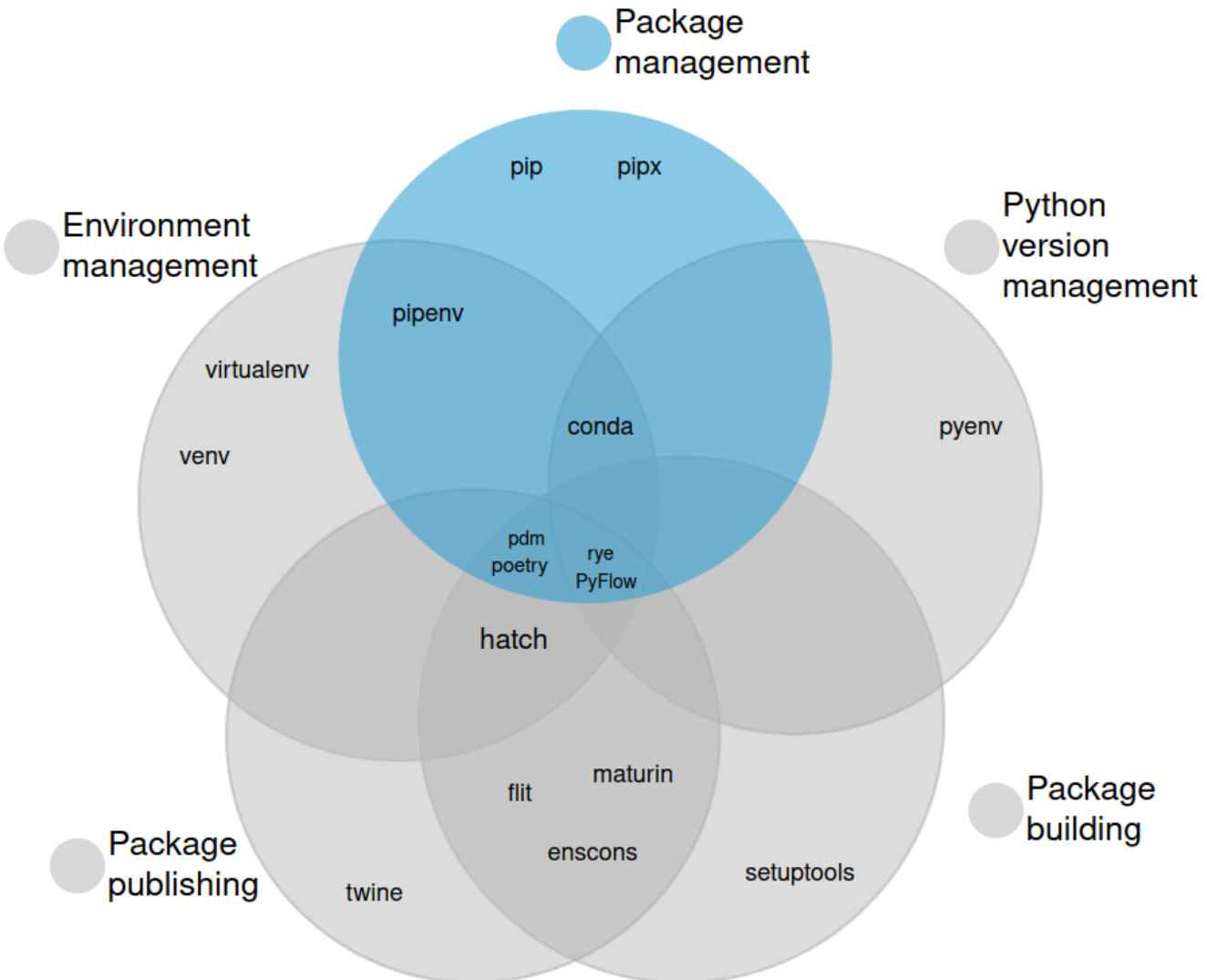


Diagrama de Venn

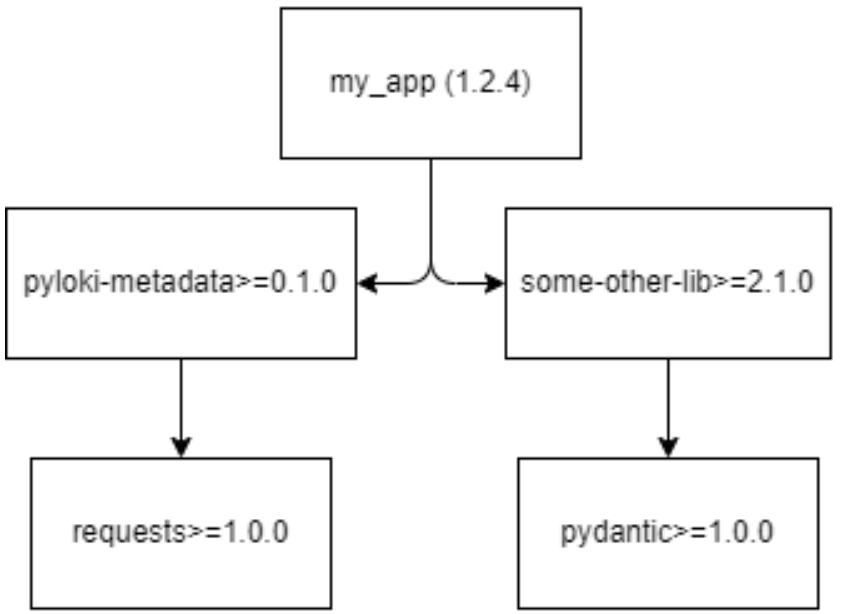
Muitas opções, um só caminho

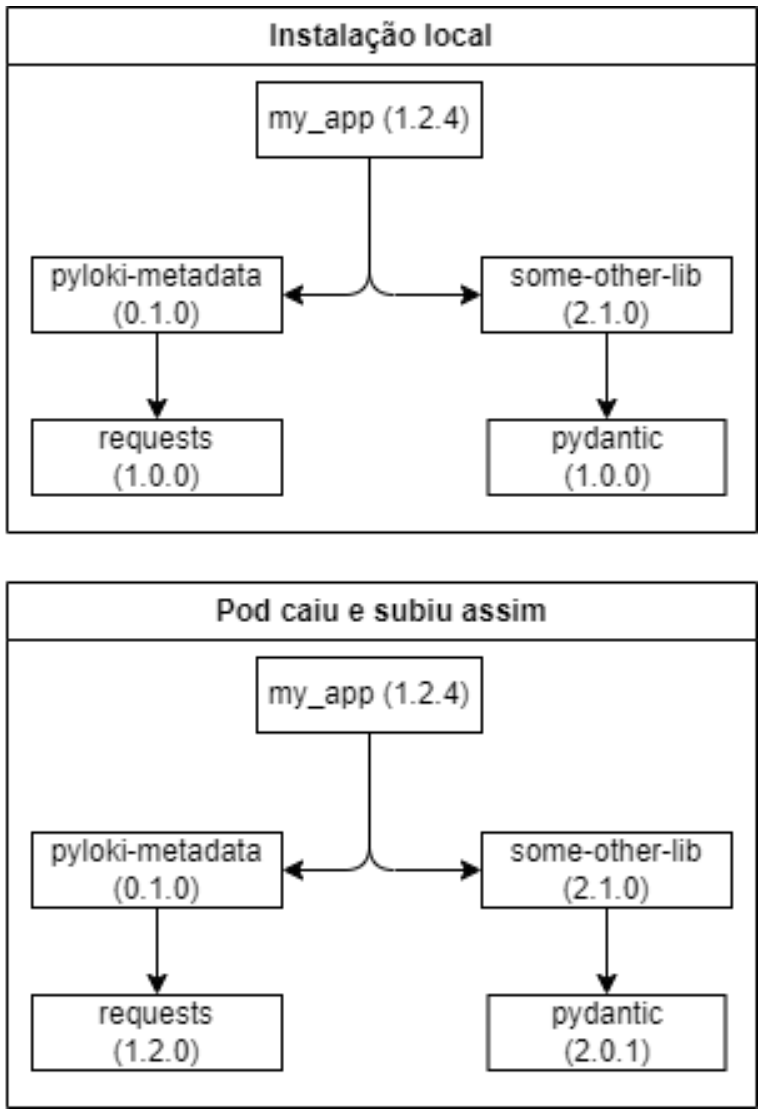
Eu só quero instalar meu pacote e suas dependências!

Pip

- Gerenciador comum ao Python, já vem com a linguagem
- `pip install some-package-from-pypi`
- `pip install -e .[dev]`
- `pip install -r requirements.txt`

Pip





Como são resolvidas as dependencias das minhas dependencias?

Pip

Como garantir a reprodutibilidade do meu app e garantir o meu sextou sem incidentes?

Lock files

- Arquivos que armazenam todas as dependencias
  - Garante reprodutibilidade
  - Acelera a resolução das dependencias (instalação mais rápida)
  - Aumenta a segurança da aplicação (supply chain)
  - Aumenta as chances de um sextou

Lock files

- Como?
  - Pin de versões dos pacotes e os pacotes dos pacotes…
  - Artifact hashing para checar se o hash do download atual é o mesmo do arquivo .lock

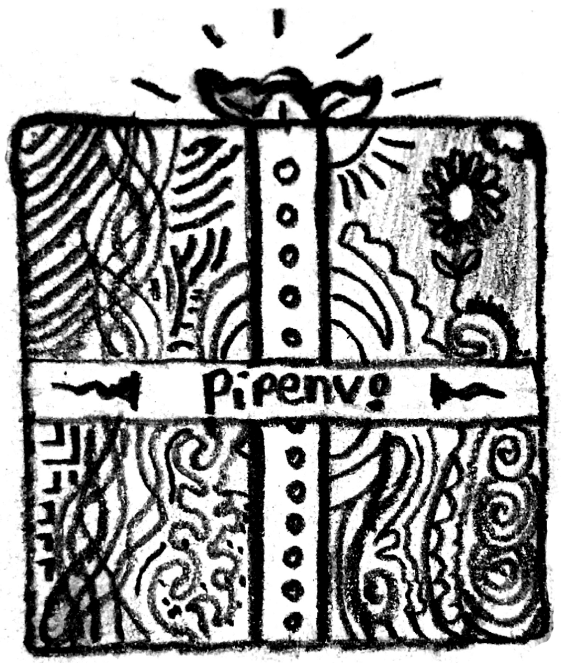
Lock files

```
# This file is automatically @generated by Poetry 1.7.1 and should

[[package]]
name = "annotated-types"
version = "0.6.0"
description = "Reusable constraint types to use with typing.Annotat
optional = false
python-versions = ">=3.8"
files = [
    {file = "annotated_types-0.6.0-py3-none-any.whl", hash = "sha25
    {file = "annotated_types-0.6.0.tar.gz", hash = "sha256:563339e8
]

[package.dependencies]
typing-extensions = {version = ">=4.0.0", markers = "python_versior
```

Posso usar `pip freeze > requirements.txt`?

- Remover/atualizar pacotes não garante que suas dependencias serão removidas do requirements.txt
- Não tem capacidade de resolver problemas de diferentes versões (conflitos)
  - Pacote A -> requests>=1.0.0
  - Pacote B -> requests>=1.2.0

Pipenv



- Cria automaticamente `Pipfile` e `Pipfile.lock`
- Instala corretamente a versão do Python (via Pyenv)
- Automaticamente cria `virtualenv`
- Automaticamente carrega `.env`
- Não segue o padrão python `pyproject.toml`

Poetry

- “Sucessor” do Pipenv
- `poetry.lock`
- Não segue a PEP 621
- Além das capacidades do pipenv, também publica pacotes

PDM

- Similar ao Poetry, mas segue todas as PEPs

Rye

- Pacote em fase experimental do criador do Flask

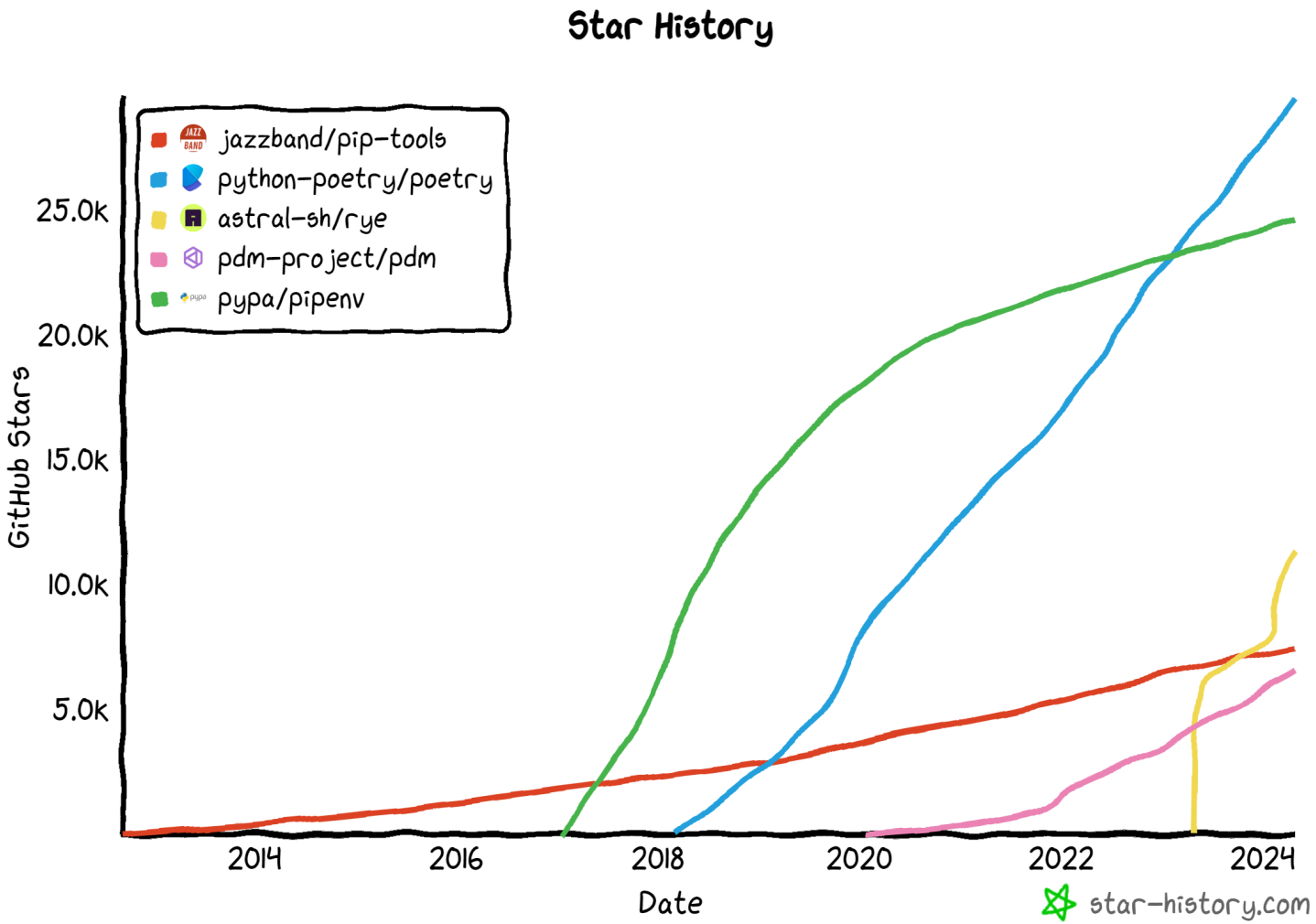
Pip-tools

- `python -m piptools compile` usa um requirements.in e gera requirements.txt com todas as dependências pinadas
- `python -m piptools sync`
- `python -m piptools update`

Comparação

- Durante as discussões do `AI & Data` levantamos algumas necessidades:
  - Queremos ter um lock (reprodutibilidade)
  - Queremos usar uma ferramenta madura
  - Preferimos seguir as PEPs do Python
  - Preferimos uma ferramenta com baixa curva de aprendizado
- Opções: pipenv, poetry e pip-tools

Comparação



Popularidade das ferramentas

Comparação

Ferramenta	cross-platform	integração com env atual	curva de aprendizado
poetry	✅	❌ (padrão próprio)	Razoável
pip-tools	❌ (pip-compile-cross-platform)	✅	Pequena
pipenv	✅ ❌	❌ (Pipfile vs Pyproject.toml)	Razoável

Por que não utilizar o Pipenv?

- Já foi abandonado uma vez (credibilidade)
- Resolução de conflitos já foi um problema no passado (talvez ainda seja lento)
- Não segue a estrutura pyproject.toml sugerida pelas PEPs
- Além de instalar os pacotes cria a virtualenv (mais do que precisamos)

Por que não utilizar o pip-tools?

- Problemas de geração de lock files cross-platform:
  - Cada plataforma precisaria de um requirements.in diferente

Poetry

- Não segue a estrutura pyproject.toml sugerida pelas PEPs
- Além de instalar os pacotes, criar a virtualenv, também publica os pacotes (mais do que precisamos)
- No time de AI & Data não usariamos a virtualenv, continuaríamos usando pyenv, venv e rodaríamos o `poetry install`, `poetry add some-dep`, `poetry update`

Conda e mamba

- Criados para projetos de data science
- Gerenciam além da instalação dos pacotes python, como dependencias do sistema (ex. CUDA)
- Existe a possibilidade de usar conda + poetry
- Conda e mamba não tem .lock por default, usam `conda-lock`
- Mamba é mais rápido na resolução de pacotes e uso com docker

Conclusão

- Lock de dependências em Python é uma questão em aberto, com PEP rejeitada
- Não existe ferramenta que resolva todos os problemas
- Dentre as ferramentas investigadas: Poetry e pip-tools são soluções maduras que atendem nossos requisitos

Referências

- Como poetry usa .lock cross platform
- Python packaging user guide
- PEP 665 lock file rejeitada
- Faster conda install