uv: Speed Meets Simplicity in Python Package Management

RND Light-Talk, December 2024

Yonatan Bitton
@bityob

bit.ly/uv-intro

Source: https://xkcd.com/1987/

The Painful Past of Python Packaging

The Painful Past of Python Packaging

- Python packaging can be hard for beginners:
  - Bootstrapping, i.e., how to even get started!
  - Activation, i.e., how do virtual environments (venvs) in Python work?

The Painful Past of Python Packaging

- Python packaging can be hard for beginners:
  - Bootstrapping, i.e., how to even get started!
  - Activation, i.e., how do virtual environments (venvs) in Python work?
- Managing Python versions

The Painful Past of Python Packaging

- Python packaging can be hard for beginners:
  - Bootstrapping, i.e., how to even get started!
  - Activation, i.e., how do virtual environments (venvs) in Python work?
- Managing Python versions
- Handling "dependency hell"

The Painful Past of Python Packaging

- Python packaging can be hard for beginners:
  - Bootstrapping, i.e., how to even get started!
  - Activation, i.e., how do virtual environments (venvs) in Python work?
- Managing Python versions
- Handling "dependency hell"
- No unified tool for all workflows

The Painful Past of Python Packaging

- Python packaging can be hard for beginners:
  - Bootstrapping, i.e., how to even get started!
  - Activation, i.e., how do virtual environments (venvs) in Python work?
- Managing Python versions
- Handling "dependency hell"
- No unified tool for all workflows
- pip is very, very slow

Meet **uv**

An **extremely fast** Python package and project manager, built with the power of **Rust**.

A fast, all-in-one Python package manager

A fast, all-in-one Python package manager

- You can use uv to: install Python, create virtual environments, resolve dependencies, install packages, build packages and more

A fast, all-in-one Python package manager

- You can use uv to: install Python, create virtual environments, resolve dependencies, install packages, build packages and more
- A drop-in alternative to `pip`, `pipx`, `pyenv`, `virtualenv`, `poetry` and more

A fast, all-in-one Python package manager

- You can use uv to: install Python, create virtual environments, resolve dependencies, install packages, build packages and more
- A drop-in alternative to `pip`, `pipx`, `pyenv`, `virtualenv`, `poetry` and more
- A single static binary that gives you everything you need to be productive with Python

A fast, all-in-one Python package manager

- You can use uv to: install Python, create virtual environments, resolve dependencies, install packages, build packages and more
- A drop-in alternative to `pip`, `pipx`, `pyenv`, `virtualenv`, `poetry` and more
- A single static binary that gives you everything you need to be productive with Python
- 10-100x faster than alternatives. Written in Rust

New Tool, Big Impact

New Tool, Big Impact

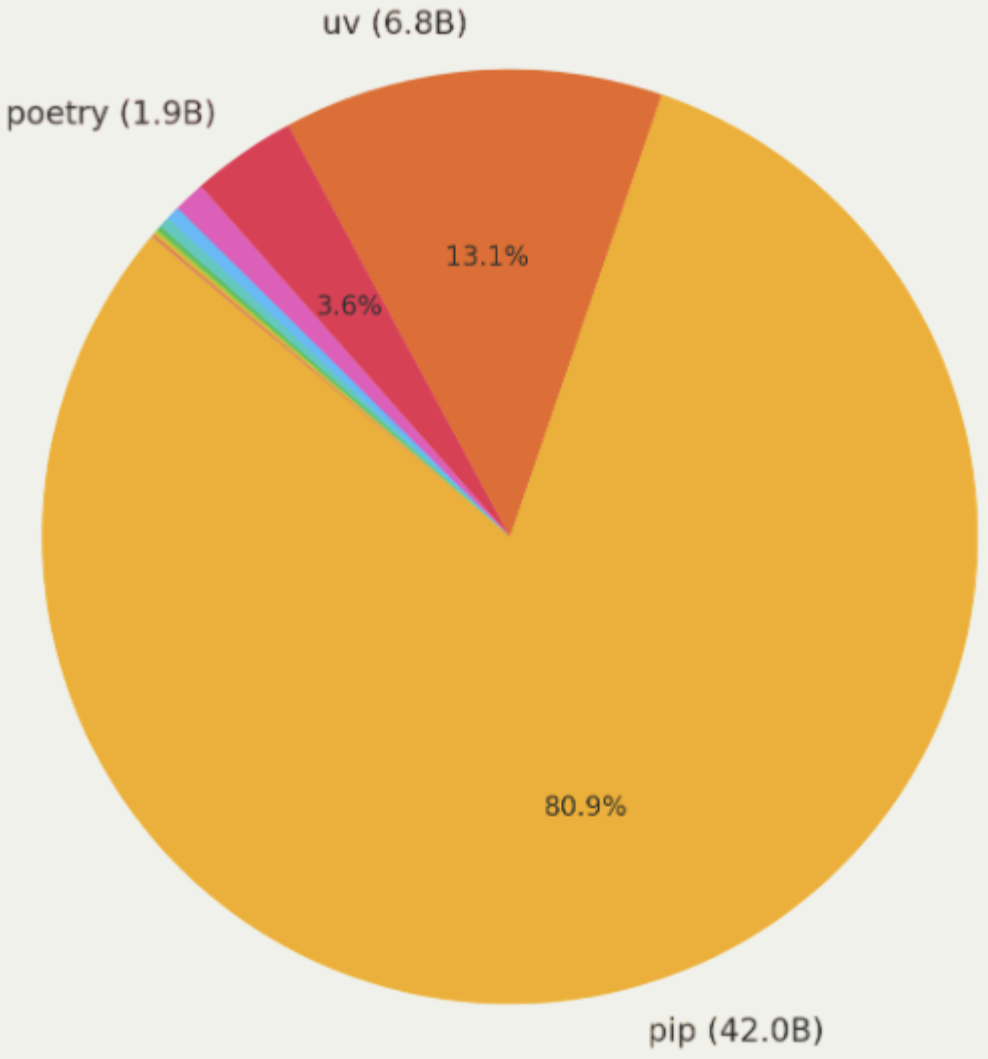
- 21+ million downloads last month

New Tool, Big Impact

- 21+ million downloads last month
- 33.3k stars in GitHub

New Tool, Big Impact

- 21+ million downloads last month
- 33.3k stars in GitHub
- 6.8+ billion packages were downloaded using UV last month, which is 13% of total downloads


uv (6.8B)
poetry (1.9B)
13.1%
3.6%
80.9%
pip (42.0B)

Agenda

Agenda

- Overview
  - Installation
  - pip interface
  - uv commands

Agenda

- Overview
  - Installation
  - pip interface
  - uv commands
- The uv ecosystem
  - Projects
  - Tools
  - Scripts

Agenda

- Overview
  - Installation
  - pip interface
  - uv commands
- The uv ecosystem
  - Projects
  - Tools
  - Scripts
- Why UV is so fast?

Installation

- No need to have Python/Rust installed to install UV

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

```
powershell -c "irm https://astral.sh/uv/install.ps1 | iex"
```

- Alternatively, can be installed using `pip` or `pipx`

```
pip install uv
pipx install uv
```

pip interface

pip interface

- Drop-in replacement for common `pip` and `virtualenv` commands

pip interface

- Drop-in replacement for common `pip` and `virtualenv` commands
- Creating a virtual environment: `uv venv`

pip interface

- Drop-in replacement for common `pip` and `virtualenv` commands
- Creating a virtual environment: `uv venv`
- Install a package: `uv pip install flask`

pip interface

- Drop-in replacement for common `pip` and `virtualenv` commands
- Creating a virtual environment: `uv venv`
- Install a package: `uv pip install flask`
- List packages: `uv pip freeze`

```
$ uv venv
Using CPython 3.10.14
Creating virtual environment at: .venv
Activate with: source .venv/bin/activate
$ uv pip install flask
Resolved 7 packages in 491ms
Installed 7 packages in 13ms
 + blinker==1.9.0
 + click==8.1.8
 + flask==3.1.0
 + itsdangerous==2.2.0
 + jinja2==3.1.5
 + markupsafe==3.0.2
 + werkzeug==3.1.3
$ uv pip freeze
```

```
Using CPython 3.10.14
Creating virtual environment at: .venv
Activate with: source .venv/bin/activate
$ uv pip install flask
Resolved 7 packages in 491ms
Installed 7 packages in 13ms
 + blinker==1.9.0
 + click==8.1.8
 + flask==3.1.0
 + itsdangerous==2.2.0
 + jinja2==3.1.5
 + markupsafe==3.0.2
 + werkzeug==3.1.3
$ uv pip freeze
blinker==1.9.0
click==8.1.8
```

```
 8  + blinker==1.9.0
 9  + click==8.1.8
10  + flask==3.1.0
11  + itsdangerous==2.2.0
12  + jinja2==3.1.5
13  + markupsafe==3.0.2
14  + werkzeug==3.1.3
15 $ uv pip freeze
16 blinker==1.9.0
17 click==8.1.8
18 flask==3.1.0
19 itsdangerous==2.2.0
20 jinja2==3.1.5
21 markupsafe==3.0.2
22 werkzeug==3.1.3
```

uv commands

uv commands

uv commands

uv command	pip command(s)	Notes
`uv init <name>`	N/A	creates a new empty projec according to the `pyproject.toml` speci

Source: https://docs.astral.sh/uv/guide/projects/#creating-projects.html

uv commands

uv command	pip command(s)	Notes
`uv add <package>`	`pip install <package>`	uv also adds it to the `pyproject.toml` and update the `uv.lock` file

Source: https://docs.astral.sh/uv/guides/projects/#managing-dependencies

Since we are using docker to run our applications, you can't run `uv add <package>` on local, since it will try to **install the packages too.**

For such cases, just run with `--no-sync` flag

```
uv add <package> --no-sync
```

Source: https://docs.astral.sh/uv/concepts/projects/sync/#preventing-syncing.html

uv commands

uv command	pip command(s)	Notes
`uv lock`	`N/A`	creates the `uv.lock` file with all pinned dependencies

Source: https://docs.astral.sh/uv/pip/compatibility/#pip-compile

uv commands

uv command	pip command(s)	Notes
`uv sync`	`pip freeze | xargs pip uninstall -y && pip install -r requirements.txt`	uv also creates a venv if it doesn't exist

Source: https://github.com/astral-sh/uv/blob/main/docs/pip/compatibility.md#pip-sync

uv commands

uv command	pip command(s)	Notes
`uv run <python_file>`	`source .venv/activate && python <python_file>`	uv downloads python if not exists and sync the venv before run

[illegible]

uv commands

uv command	pip command(s)	Notes
`uv remove <package>`	`pip uninstall <package> && pip freeze > requirements.txt`	

Source: https://blog.boot.dev/cheatsheets/uv-cheatsheet.html

```
$ uv init
Initialized project `playground`
$ cat pyproject.toml
[project]
name = "playground"
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
requires-python = ">=3.10"
dependencies = []
$ uv add pycowsay
Resolved 2 packages in 346ms
Installed 1 package in 7ms
 ++ pycowsay==0.0.0.2
$ uv run pycowsay "Hello Python World!"
```

Source: https://docs.astral.sh/uv/guides/projects/#creating-a-project

```
$ uv init
Initialized project `playground`
$ cat pyproject.toml
[project]
name = "playground"
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
requires-python = ">=3.10"
dependencies = []
$ uv add pycowsay
Resolved 2 packages in 346ms
Installed 1 package in 7ms
 ++ pycowsay==0.0.0.2
$ uv run pycowsay "Hello Python World!"
```

Source: https://docs.astral.sh/uv/guides/projects/#creating-a-project

```
 5 name = "playground"
 6 version = "0.1.0"
 7 description = "Add your description here"
 8 readme = "README.md"
 9 requires-python = ">=3.10"
10 dependencies = []
11 $ uv add pycowsay
12 Resolved 2 packages in 346ms
13 Installed 1 package in 7ms
14  ++ pycowsay==0.0.0.2
15 $ uv run pycowsay "Hello Python World!"
16
17   ------------------
18 < Hello Python World! >
19   ------------------
20    \   ^__^
```

Source: https://github.com/astral-sh/uv/blob/main/README.md

```
dependencies = []
$ uv add pycowsay
Resolved 2 packages in 346ms
Installed 1 package in 7ms
 ++ pycowsay==0.0.0.2
$ uv run pycowsay "Hello Python World!"

  ------------------
< Hello Python World! >
  ------------------
   \   ^__^
    \  (oo)\_______
       (__)\       )\/\
           ||----w |
           ||     ||
```

Source: https://github.com/astral-sh/uv/blob/main/README.md

uv commands

uv command	pip command(s)	Notes
`uv python list`	N/A	list all python versions installed on your **computer**
`uv python install <version>`	N/A	using precompiled binaries

Source: https://docs.astral.sh/uv/guides/install-python/#installing-a-python-version

```
$ uv python list
cpython-3.13.0a0-linux-x86_64-gnu                  /usr/local/bin
cpython-3.12.7-linux-x86_64-gnu                    <download avai
cpython-3.12.0-linux-x86_64-gnu                    /home/linuxbre
cpython-3.11.10-linux-x86_64-gnu                   <download avai
cpython-3.11.6-linux-x86_64-gnu                    /home/linuxbre
cpython-3.11.4-linux-x86_64-gnu                    /home/bit/.asd
cpython-3.11.4-linux-x86_64-gnu                    /home/bit/.asd
cpython-3.11.4-linux-x86_64-gnu                    /home/bit/.asd
cpython-3.10.15-linux-x86_64-gnu                   <download avai
...
$ uv python install cpython-3.10.15-linux-x86_64-gnu
Installed Python 3.10.15 in 4.74s
 + cpython-3.10.15-linux-x86_64-gnu
```

Source: https://astral.sh/blog/uv.html

```
$ uv python list
cpython-3.13.0a0-linux-x86_64-gnu                 /usr/local/bin
cpython-3.12.7-linux-x86_64-gnu                   <download avai
cpython-3.12.0-linux-x86_64-gnu                   /home/linuxbre
cpython-3.11.10-linux-x86_64-gnu                  <download avai
cpython-3.11.6-linux-x86_64-gnu                   /home/linuxbre
cpython-3.11.4-linux-x86_64-gnu                   /home/bit/.asd
cpython-3.11.4-linux-x86_64-gnu                   /home/bit/.asd
cpython-3.11.4-linux-x86_64-gnu                   /home/bit/.asd
cpython-3.10.15-linux-x86_64-gnu                  <download avai
...
$ uv python install cpython-3.10.15-linux-x86_64-gnu
Installed Python 3.10.15 in 4.74s
 + cpython-3.10.15-linux-x86_64-gnu
```

Source: https://astral.sh/blog/uv.html

uv ecosystem

Projects

- Python projects/repositories
- Using with `uv` commands like previous slides
- Configuration file is inside `pyproject.toml`

Source: https://docs.astral.sh/uv/guides/projects/#projects

Tools

- CLI tools, e.g `ruff`, `llm`, `pycowsay`
- No file needed
- Virtual environment included
- Same like `pipx` tool

```
uvx <tool>
uv tool run <tool>
```

```
$ uvx art text "Hello World!"

 _   _        _  _         __        __              _      _  _
| | | |  ___ | || |  ___   \ \      / /  ___   _ __ | |  __| || |
| |_| | / _ \| || | / _ \   \ \ /\ / /  / _ \ | '__|| | / _` || |
|  _  ||  __/| || || (_) |   \ V  V /  | (_) || |   | || (_| ||_|
|_| |_| \___||_||_| \___/     \_/\_/    \___/ |_|   |_| \__,_|(_)
```

Scripts

- Scripts are single files without project files
- A script without dependencies is easy, just `uv run script`
- But how can we run a script with dependencies??

Source: https://blog.jetbrains.com/pycharm/2024/07/2024-2/#uv.html

Scripts

- Scripts are single files without project files
- A script without dependencies is easy, just `uv run script`
- But how can we run a script with dependencies??
- uv supports specifying dependencies **on invocation**

```
$ cat http_requests_uuid.py
import concurrent.futures

urls = ["https://httpbin.org/uuid" for _ in range(5)]

def get_url(url):
    import requests
    return requests.get(url).json()

with concurrent.futures.ThreadPoolExecutor(3) as executor:
    for x in executor.map(get_url, urls):
        print(x)

$ uv run --with requests http_requests_uuid.py
{'uuid': '2f9c8816-960f-40e8-a26e-a0ada4b4c5cc'}
```

Source: [illegible]

```
 5
 6 def get_url(url):
 7     import requests
 8     return requests.get(url).json()
 9
10 with concurrent.futures.ThreadPoolExecutor(3) as executor:
11     for x in executor.map(get_url, urls):
12         print(x)
13
14 $ uv run --with requests http_requests_uuid.py
15 {'uuid': '2f9c8816-960f-40e8-a26e-a0ada4b4c5cc'}
16 {'uuid': '22d43f6c-7f81-46d1-867c-43260b0155bb'}
17 {'uuid': '3303265d-1d9c-41ac-a43e-5a9fed0aa02c'}
18 {'uuid': 'bed4fdd9-0e7b-4641-b9ed-19889e435c42'}
19 {'uuid': '6944a068-67d9-4778-bb40-2b77c9115e5e'}
```

Source: https://example.com/page.html

But this can be done even better

PEP 723 – Inline script metadata

Author: Ofek Lev <ofekmeister at gmail.com>

Sponsor: Adam Turner <python at quite.org.uk>

PEP-Delegate: Brett Cannon <brett at python.org>

Discussions-To: Discourse thread

Status: Final

Type: Standards Track

Topic: Packaging

Created: 04-Aug-2023

Post-History: 04-Aug-2023, 06-Aug-2023, 23-Aug-2023, 06-Dec-2023

Replaces: 722

Resolution: 08-Jan-2024

https://peps.python.org/pep-0723/

Source: https://peps.python.org/pep-0723/index.html

Now we can use something like this

```
$ uv add --script http_requests_uuid.py requests
Updated `http_requests_uuid.py`
$ cat http_requests_uuid.py
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "requests",
# ]
# ///
import concurrent.futures

urls = ["https://httpbin.org/uuid" for _ in range(5)]

def get_url(url):
    import requests
```

Source: https://blog.glyph.im/2025/08/pep-723.html

Now we can use something like this

```
$ uv add --script http_requests_uuid.py requests
Updated `http_requests_uuid.py`
$ cat http_requests_uuid.py
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "requests",
# ]
# ///
import concurrent.futures

urls = ["https://httpbin.org/uuid" for _ in range(5)]

def get_url(url):
    import requests
```

Now we can use something like this

```
 3 $ cat http_requests_uuid.py
 4 # /// script
 5 # requires-python = ">=3.10"
 6 # dependencies = [
 7 #     "requests",
 8 # ]
 9 # ///
10 import concurrent.futures
11
12 urls = ["https://httpbin.org/uuid" for _ in range(5)]
13
14 def get_url(url):
15     import requests
16     return requests.get(url).json()
17
```

Source: https://simonwillison.net/2024/Dec/19/one-shot-python-tools/#uv-and-inline-dependencies.html

Now we can use something like this

```

def get_url(url):
    import requests
    return requests.get(url).json()

with concurrent.futures.ThreadPoolExecutor(3) as executor:
    for x in executor.map(get_url, urls):
        print(x)
$ uv run http_requests_uuid.py
Reading inline script metadata from `http_requests_uuid.py`
{'uuid': '6d852239-642d-46e1-9718-0f0862808b91'}
{'uuid': 'b70dfc29-5cbc-4f36-9476-fd1c79ec32c0'}
{'uuid': 'c790752b-681a-45d2-af17-38fb44917612'}
{'uuid': 'd22336bb-cd53-40d9-b1cb-c2348cdcdac6'}
{'uuid': '81e9cda6-6072-4fd8-80b0-08cc55553fd5'}
```

You can limit uv to only considering distributions released before a specific date

```
$ cat example_requests.py
# /// script
# dependencies = [
#   "requests",
# ]
# [tool.uv]
# exclude-newer = "2020-10-16T00:00:00Z"
# ///

import requests

print(requests.__version__)
$ uv run example_requests.py
Reading inline script metadata from `example_requests.py`
2.24.0
```

Source: https://github.com/astral-sh/uv/blob/main/docs/guides/scripts.md#improving-reproducibility

You can limit uv to only considering distributions released before a specific date

```
$ cat example_requests.py
# /// script
# dependencies = [
#   "requests",
# ]
# [tool.uv]
# exclude-newer = "2020-10-16T00:00:00Z"
# ///

import requests

print(requests.__version__)
$ uv run example_requests.py
Reading inline script metadata from `example_requests.py`
2.24.0
```

Source: https://github.com/astral-sh/uv/blob/main/docs/guides/scripts.md

You can limit uv to only considering distributions released before a specific date

```
$ cat example_requests.py
# /// script
# dependencies = [
#   "requests",
# ]
# [tool.uv]
# exclude-newer = "2020-10-16T00:00:00Z"
# ///

import requests

print(requests.__version__)
$ uv run example_requests.py
Reading inline script metadata from `example_requests.py`
2.24.0
```

Source: https://docs.astral.sh/uv/guides/scripts/#improving-reproducibility

Now, the interesting part...

[illegible]

Now, the interesting part...

Why UV is so fast?

(1)

Written in pure **Rust**.

100k~ lines of code.

(1)

Written in pure **Rust**.

100k~ lines of code.

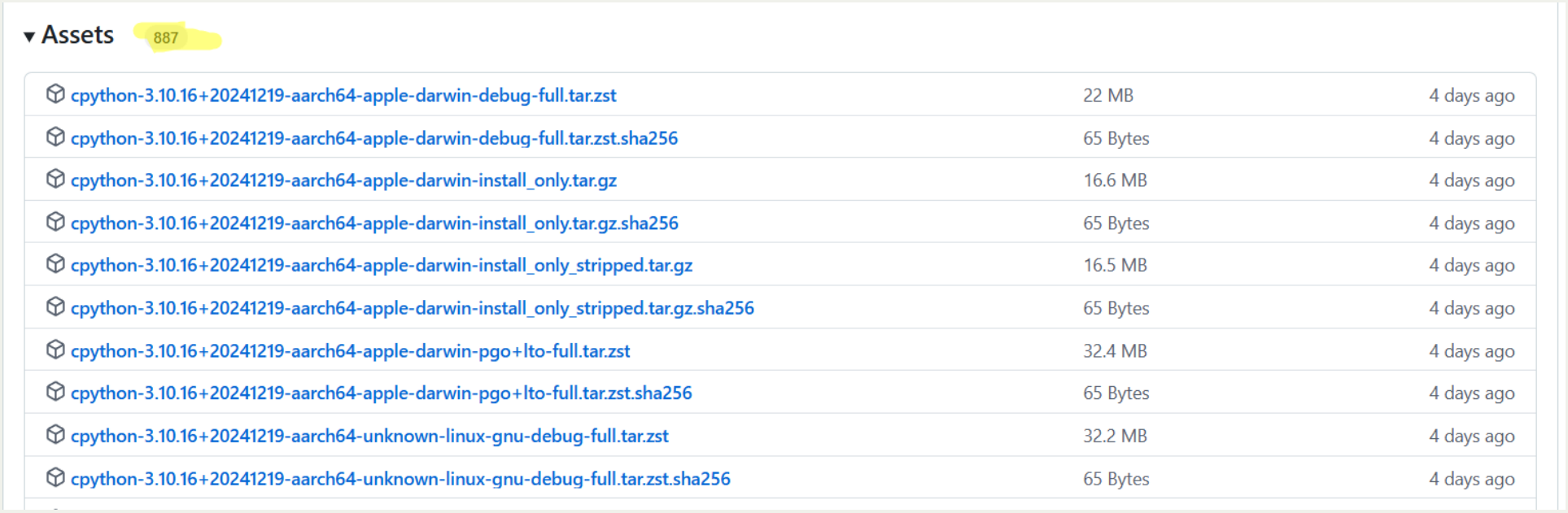
But... uv has gotten significantly faster over time

(2)

Fast python install using precompiled binaries.

(2)

Fast python install using precompiled binaries.



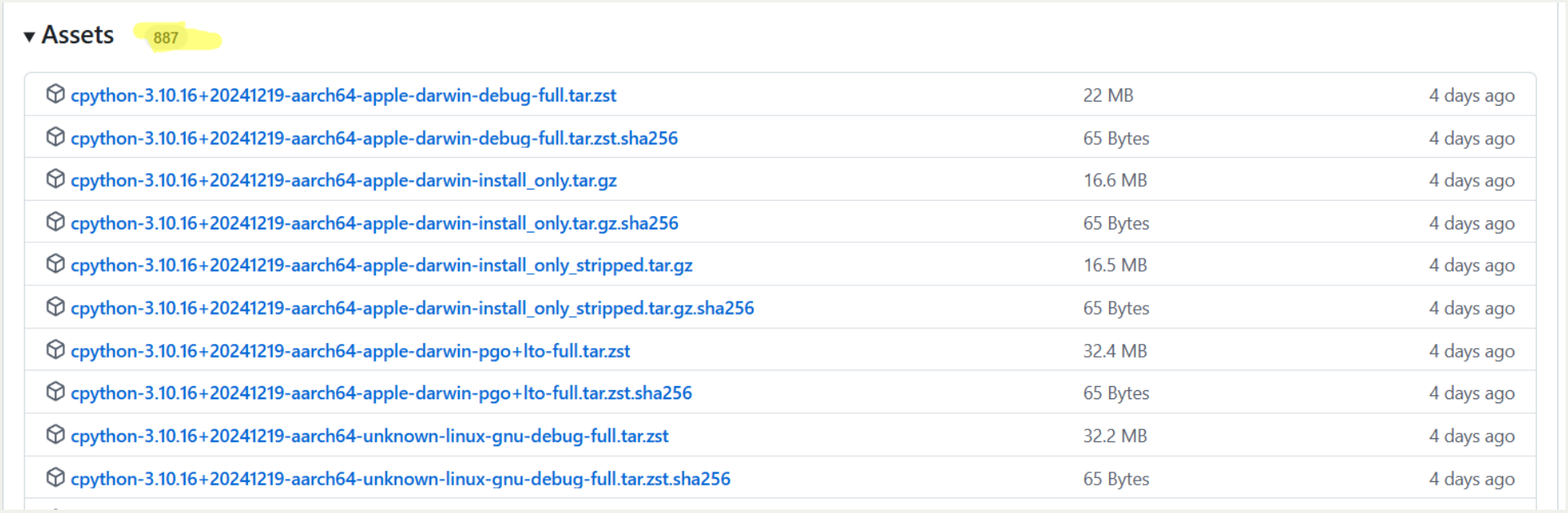
▾ Assets 887

cpython-3.10.16+20241219-aarch64-apple-darwin-debug-full.tar.zst	22 MB	4 days ago
cpython-3.10.16+20241219-aarch64-apple-darwin-debug-full.tar.zst.sha256	65 Bytes	4 days ago
cpython-3.10.16+20241219-aarch64-apple-darwin-install_only.tar.gz	16.6 MB	4 days ago
cpython-3.10.16+20241219-aarch64-apple-darwin-install_only.tar.gz.sha256	65 Bytes	4 days ago
cpython-3.10.16+20241219-aarch64-apple-darwin-install_only_stripped.tar.gz	16.5 MB	4 days ago
cpython-3.10.16+20241219-aarch64-apple-darwin-install_only_stripped.tar.gz.sha256	65 Bytes	4 days ago
cpython-3.10.16+20241219-aarch64-apple-darwin-pgo+lto-full.tar.zst	32.4 MB	4 days ago
cpython-3.10.16+20241219-aarch64-apple-darwin-pgo+lto-full.tar.zst.sha256	65 Bytes	4 days ago
cpython-3.10.16+20241219-aarch64-unknown-linux-gnu-debug-full.tar.zst	32.2 MB	4 days ago
cpython-3.10.16+20241219-aarch64-unknown-linux-gnu-debug-full.tar.zst.sha256	65 Bytes	4 days ago

Source: https://github.com/astral-sh/python-build-standalone/releases/expanded_assets/20241219

(2)

Fast python install using precompiled binaries.



▾ Assets 887

cpython-3.10.16+20241219-aarch64-apple-darwin-debug-full.tar.zst	22 MB	4 days ago
cpython-3.10.16+20241219-aarch64-apple-darwin-debug-full.tar.zst.sha256	65 Bytes	4 days ago
cpython-3.10.16+20241219-aarch64-apple-darwin-install_only.tar.gz	16.6 MB	4 days ago
cpython-3.10.16+20241219-aarch64-apple-darwin-install_only.tar.gz.sha256	65 Bytes	4 days ago
cpython-3.10.16+20241219-aarch64-apple-darwin-install_only_stripped.tar.gz	16.5 MB	4 days ago
cpython-3.10.16+20241219-aarch64-apple-darwin-install_only_stripped.tar.gz.sha256	65 Bytes	4 days ago
cpython-3.10.16+20241219-aarch64-apple-darwin-pgo+lto-full.tar.zst	32.4 MB	4 days ago
cpython-3.10.16+20241219-aarch64-apple-darwin-pgo+lto-full.tar.zst.sha256	65 Bytes	4 days ago
cpython-3.10.16+20241219-aarch64-unknown-linux-gnu-debug-full.tar.zst	32.2 MB	4 days ago
cpython-3.10.16+20241219-aarch64-unknown-linux-gnu-debug-full.tar.zst.sha256	65 Bytes	4 days ago

Really, there are around 400 builds (including checksum files), but still!

Sou

(3)

Version parsing.

(3)

Version parsing.

- `1.0.0`

(3)

Version parsing.

- `1.0.0`
- `2.3.1-beta.1`

(3)

Version parsing.

- `1.0.0`
- `2.3.1-beta.1`
- `1.0.post345`

(3)

Version parsing.

- `1.0.0`
- `2.3.1-beta.1`
- `1.0.post345`
- `1.2.3.4.5-a8.post9`

(3)

Version parsing.

- `1.0.0`
- `2.3.1-beta.1`
- `1.0.post345`
- `1.2.3.4.5-a8.post9`

Representing these is pretty hard.

(3)

Version parsing.

- `1.0.0`
- `2.3.1-beta.1`
- `1.0.post345`
- `1.2.3.4.5-a8.post9`

Representing these is pretty hard.

The full representation of this is

```
struct VersionFull {
    epoch: u64,
    release: Vec<u64>,
    pre: Option<Prerelease>,
    post: Option<u64>,
    dev: Option<u64>,
    local: LocalVersion,
    min: Option<u64>,
    max: Option<u64>,
}
```

Source: https://docs.rs/pep440_rs/latest/pep440_rs/struct.VersionFull.html

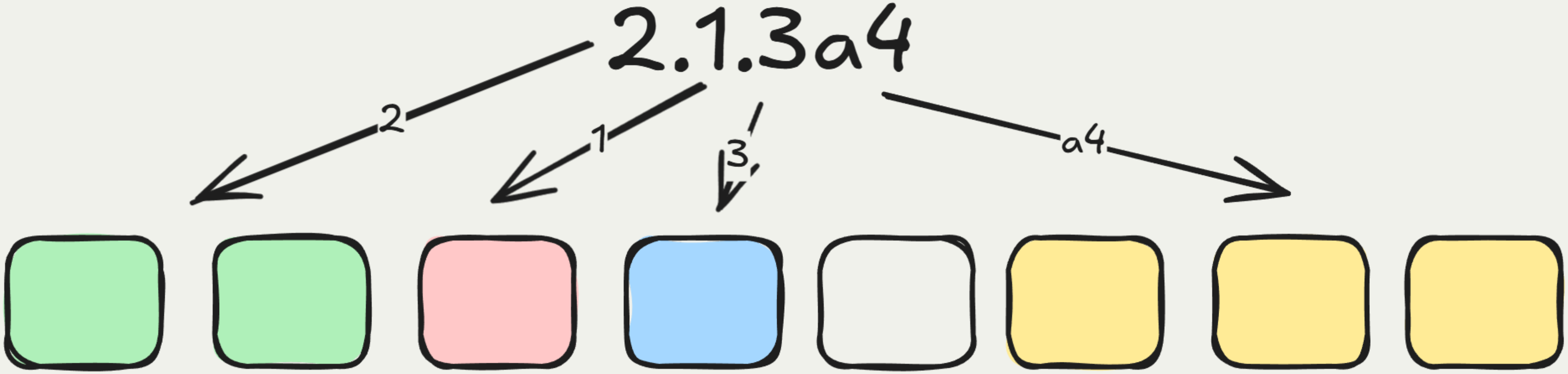
Apparently, we can represent over 90% of versions with a single u64.

```
struct VersionSmall(u64);
```

Source: [illegible]

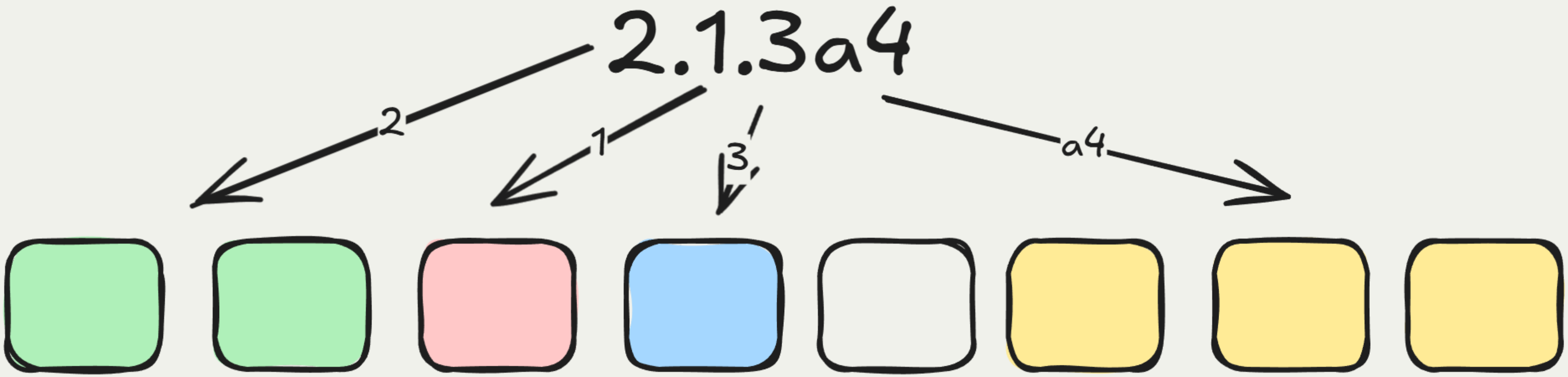
Apparently, we can represent over 90% of versions with a single u64.

```
struct VersionSmall(u64);
```



Apparently, we can represent over 90% of versions with a single u64.

```
struct VersionSmall(u64);
```



And the Best Part:

Source: https://youtu.be/lKTfG1CXYQ2?t=1284

Source: [illegible]

And the Best Part:

- Greater versions map to larger integers → Simplifies comparison

Source: https://www.geeksforgeeks.org/dsa/segment-tree-data-structure/.html

And the Best Part:

- Greater versions map to larger integers → Simplifies comparison
- Version comparison becomes extremely fast

Source: https://youtu.be/dWiSKTfG1GY?t=2884

Code Blame 3964 lines (3746 loc) · 135 KB

```
impl FromStr for Version {
/// implemented via `u64::cmp`.
///
/// We pack versions fitting the above constraints into a `u64` in such a way
/// that it preserves the ordering between versions as prescribed in PEP 440.
/// Namely:
///
/// * Bytes 6 and 7 correspond to the first release segment as a `u16`.
/// * Bytes 5, 4 and 3 correspond to the second, third and fourth release
///   segments, respectively.
/// * Bytes 2, 1 and 0 represent *one* of the following:
///   `min, .devN, aN, bN, rcN, <no suffix>, local, .postN, max`.
///   Its representation is thus:
///   * The most significant 4 bits of Byte 2 corresponds to a value in
///     the range 0-8 inclusive, corresponding to min, dev, pre-a, pre-b,
///     pre-rc, no-suffix, post or max releases, respectively. `min` is a
///     special version that does not exist in PEP 440, but is used here to
///     represent the smallest possible version, preceding any `dev`, `pre`,
///     `post` or releases. `max` is an analogous concept for the largest
///     possible version, following any `post` or local releases.
///   * The low 4 bits combined with the bits in bytes 1 and 0 correspond
///     to the release number of the suffix, if one exists. If there is no
///     suffix, then these bits are always 0.
///
/// The order of the encoding above is significant. For example, suffixes are
/// encoded at a less significant location than the release numbers, so that
/// `1.2.3 < 1.2.3.post4`.
```

(4)

Reading package `METADATA`

(4)

Reading package `METADATA`

- Built distributions (wheels) are packaged as ZIP archives

Sourcehttps://youtu.be/lWSKTfG1CXYQ?t=1286html

(4)

Reading package `METADATA`

- Built distributions (wheels) are packaged as ZIP archives
- Somewhere within the ZIP file, there's a `METADATA` file

Source: https://youtu.be/GSKTfG1GXYQ?t=2884

(4)

Reading package `METADATA`

- Built distributions (wheels) are packaged as ZIP archives
- Somewhere within the ZIP file, there's a `METADATA` file
- Some registries expose the `METADATA` file directly, while others do not

Source: https://pypi.org/simple/pyyaml/

(4)

Reading package `METADATA`

- Built distributions (wheels) are packaged as ZIP archives
- Somewhere within the ZIP file, there's a `METADATA` file
- Some registries expose the `METADATA` file directly, while others do not
- In such cases, we may need to run Python code to determine the package dependencies 😦

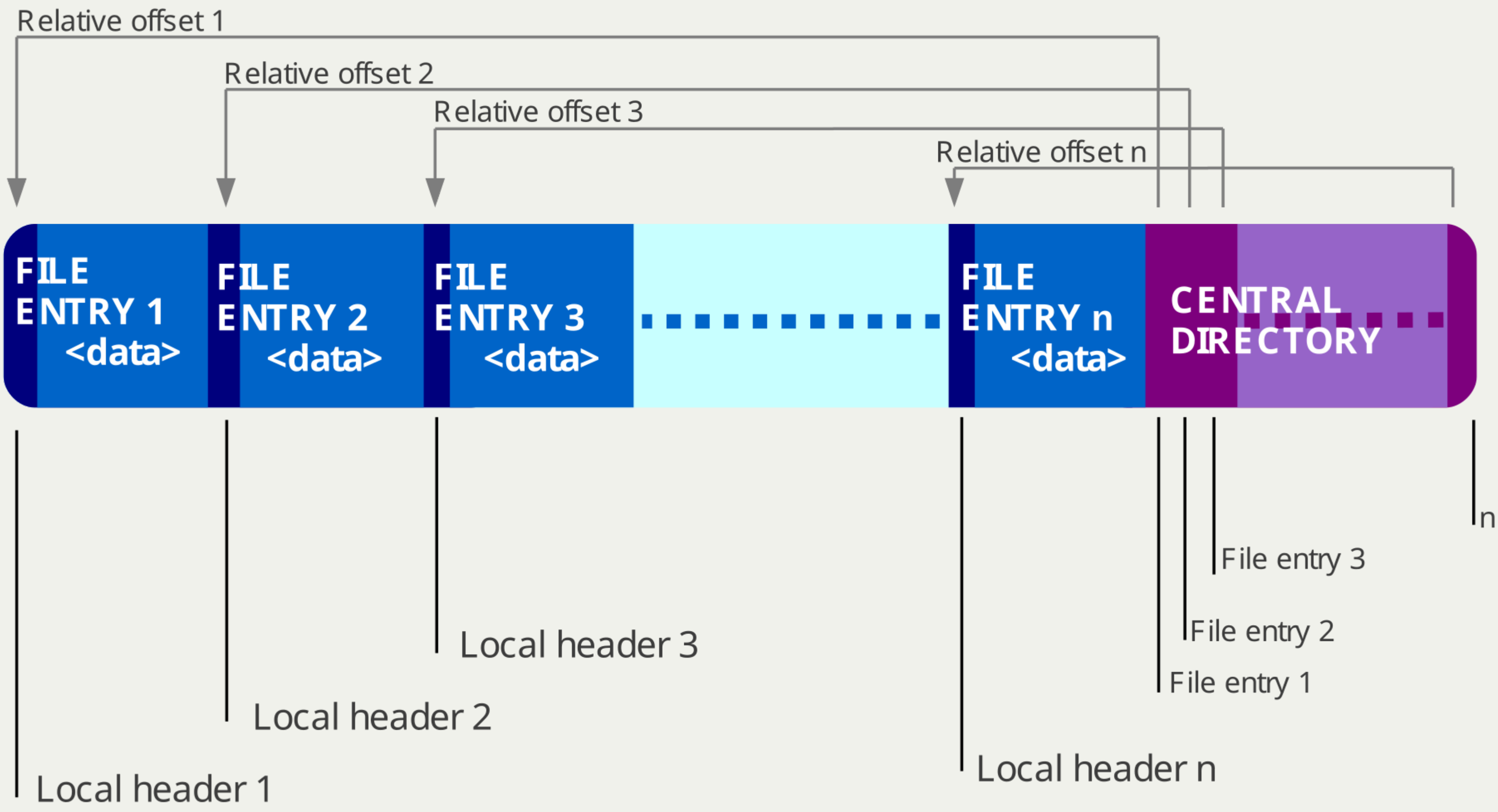
Soucource: https://py/youtu.be/skpyd...xiaQ?t=1284.html

(4)
Reading package `METADATA`

- Built distributions (wheels) are packaged as ZIP archives
- Somewhere within the ZIP file, there's a `METADATA` file
- Some registries expose the `METADATA` file directly, while others do not
- In such cases, we may need to run Python code to determine the package dependencies 😟

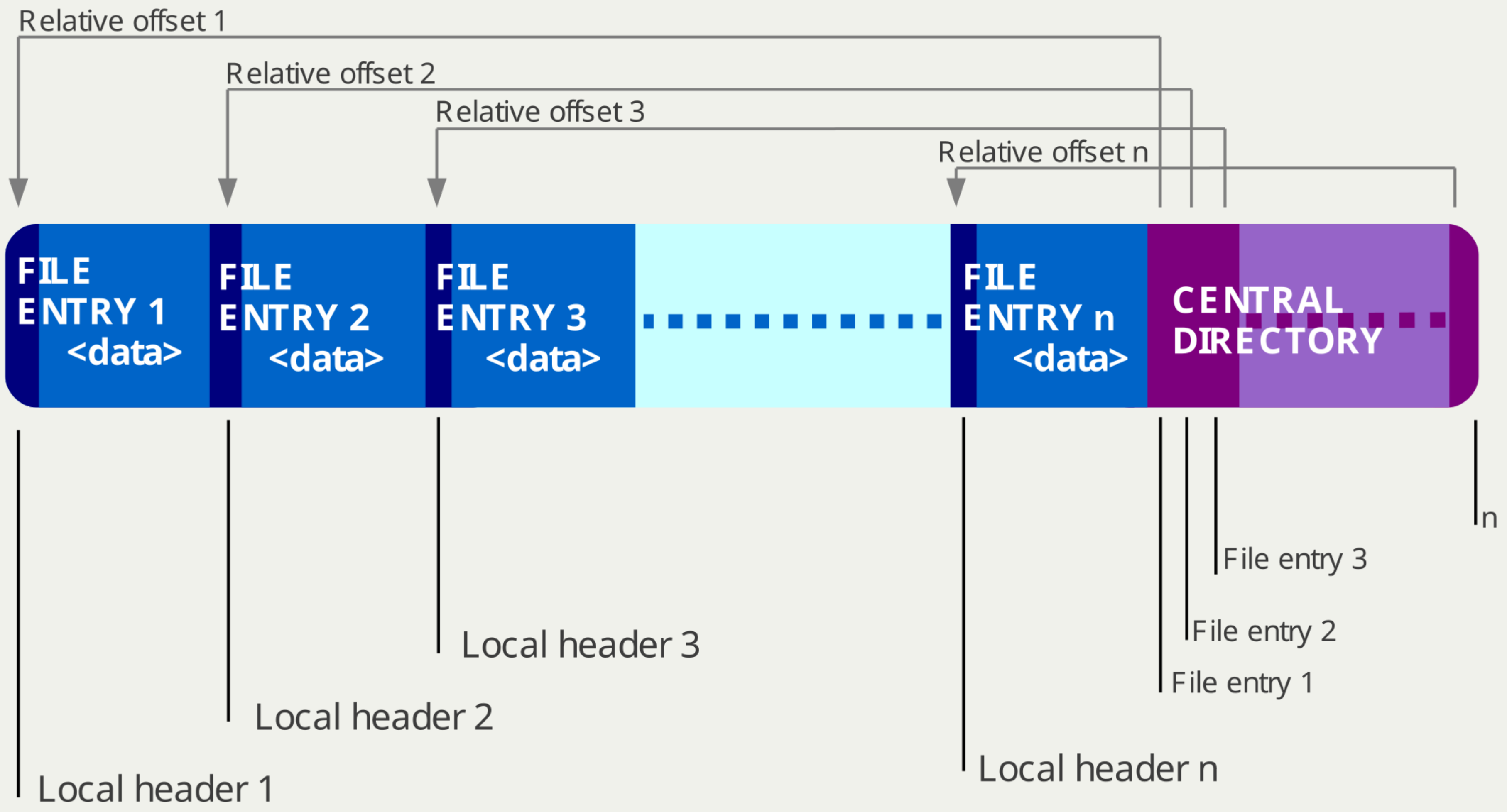
Goal: Avoid downloading the entire ZIP archive or running Python code just to read the `METADATA` file and determine package dependencies.

Source: https://pypi.org/simple/pydantic-extra-types/



Source: https:// [illegible] ?t=2246

Source: https:// [illegible] .html


Relative offset 1
Relative offset 2
Relative offset 3
Relative offset n
FILE
ENTRY 1
<data>
FILE
ENTRY 2
<data>
FILE
ENTRY 3
<data>
FILE
ENTRY n
<data>
CENTRAL
DIRECTORY
n
File entry 3
File entry 2
File entry 1
Local header 3
Local header 2
Local header 1
Local header n

Source: [illegible]

- HTTP Range request the Central Directory

Source: https://rkyv.org/zero-copy-deserialization.html

Source: https://youtu.be/-SKTfG1CXYQ?t=2246

- HTTP Range request the Central Directory
- Locate the `METADATA` file by reading the Central Directory

Source: https://www.youtube.com/watch?v=SKTfG1GXYQ2&t=224

- HTTP Range request the Central Directory
- Locate the `METADATA` file by reading the Central Directory
- HTTP Range request the `METADATA` file

Source: https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT

- HTTP Range request the Central Directory
- Locate the `METADATA` file by reading the Central Directory
- HTTP Range request the `METADATA` file

```
GET /example.whl HTTP/1.1
Host: example.com
Range: bytes=-100
```

Source: https://rkyv.org/zero-copy-deserialization.html

Source: https://youtu.be/-SKTfG1CXYQ?t=2246

(5)
Cache design

Source: https://www.youtube.com/watch?v=SKTfG1GXYQ2&t=224

(5)
Cache design

- **Global cache of unpacked archives:** uv uses aggressive caching to avoid re-downloading (and re-building) dependencies that have already been accessed in prior runs

Source: https://youtu.be/gSKTfG1GXYQ?t=2246

(5)

Cache design

- **Global cache of unpacked archives:** uv uses aggressive caching to avoid re-downloading (and re-building) dependencies that have already been accessed in prior runs
- **Most installs are just:** hardlink a bunch of files from one place to another

Source: https://youtu.be/gSKTfG1GXYQ?t=2246

(5)
Cache design

- **Global cache of unpacked archives:** uv uses aggressive caching to avoid re-downloading (and re-building) dependencies that have already been accessed in prior runs
- **Most installs are just:** hardlink a bunch of files from one place to another
- Really fast and very space efficient

Source: https://youtu.be/gSKTfG1GXYQ?t=2246

(5)

Cache design

- **Global cache of unpacked archives:** uv uses aggressive caching to avoid re-downloading (and re-building) dependencies that have already been accessed in prior runs
- **Most installs are just:** hardlink a bunch of files from one place to another
- Really fast and very space efficient
- This only applies for files, not for metadata

Source: https://youtu.be/gSKTfG1GXYQ?t=2246

For metadata, there is another **last** hack called,

Zero-copy deserialization.

Source: https://youtu.be/gSKTfG1GXYQ?t=2246

(6)

Zero-copy deserialization

A technique that reduces the time and memory required to access and use data by **directly referencing bytes in the serialized form**

Source: https://rkyv.org/zero-copy-deserialization.html

(6)

Zero-copy deserialization

A technique that reduces the time and memory required to access and use data by **directly referencing bytes in the serialized form**

Source: https://rkyv.org/zero-copy-deserialization.html

Simple zero-copy

Source: https://youtu.be/gSKTfG1GXYQ?t=2246

Simple zero-copy

```
{ "quote": "I don't know, I didn't listen." }
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Source: https://youtu.be/gSKTfG1GXYQ?t=2246

Simple zero-copy

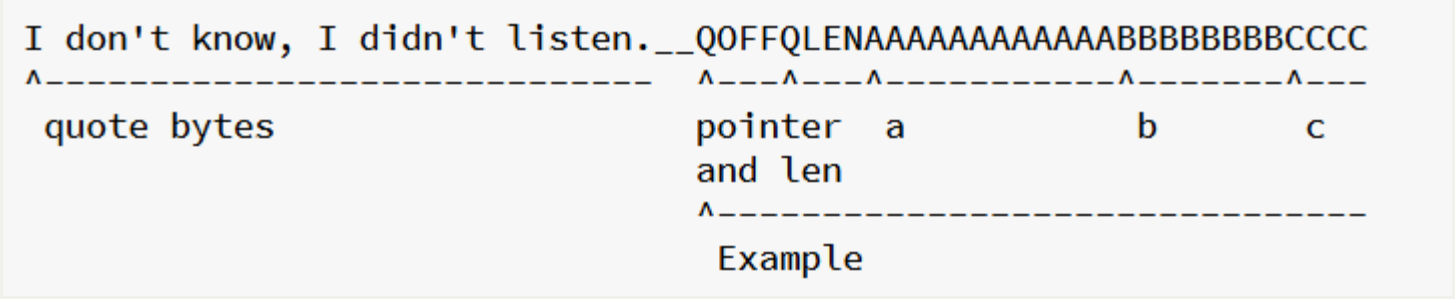
```
{ "quote": "I don't know, I didn't listen." }
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
struct NonZeroCopy {
    quote: String,
}

struct ZeroCopy<'a> {
    quote: &'a str,
}
```

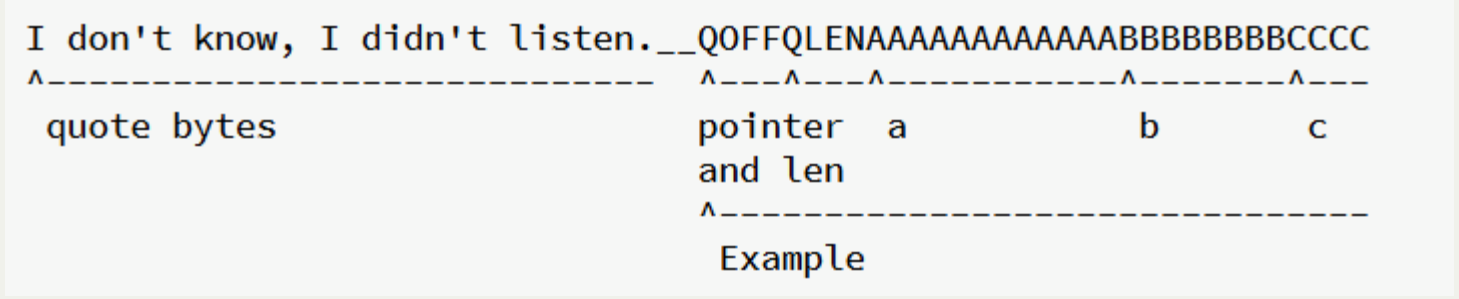
Source: https://youtu.be/gSKTfG1GXYQ?t=2246

Total zero-copy



Source: https://youtu.be/gSKTfG1GXYQ?t=2246

Total zero-copy




```
struct Example {
  quote: String,
  a: [u8; 12],
  b: u64,
  c: char,
}
```

Source: https://youtu.be/gSKTfG1GXYQ?t=2246

- On JSON, first you read the file from disc, then run a parser, and then you grab the contents and put them in the struct. All these operations would get **slower and slower as the data got bigger**
- Instead, the **zero-copy** the operation, doesn't **scale** with our data. No matter how much or how little data we have

Source: https://youtu.be/gSKTfG1GXYQ?t=2246

On UV, metadata is stored on disk in the same format as its in-memory JSON representation. This allows you to simply read it from the disk without needing to reparse it or allocate additional memory.

Source: https://youtu.be/gSKTfG1GXYQ?t=2246

Takeaways

Takeaways

- **Blazing Speed**: uv delivers swift package management and environment setup.

Takeaways

- **Blazing Speed**: uv delivers swift package management and environment setup.
- **Innovative Approach**: Focused on speed and efficiency, uv redefines Python tooling.

Takeaways

- **Blazing Speed**: uv delivers swift package management and environment setup.
- **Innovative Approach**: Focused on speed and efficiency, uv redefines Python tooling.
- **Try It Now**: Explore uv's potential to elevate your Python development.

Questions?

Sources

- Sane Python dependency management with uv (florianbrand.de)
- Charlie Marsh (founder of Astral) Presentation on Jane Street (youtube)
- uv: Unified Python packaging (astral.sh)
- uv: Python packaging in Rust (astral.sh)

Learn More (1)

- pip vs. uv: How Streamlit Cloud sped up app load times by 55% (blog.streamlit.io)
- GitHub - `astral-sh/uv`: An extremely fast Python package and project manager, written in Rust (github)
- Python Packaging is Great Now: `uv` is all you need (dev.to)
- Python Packaging Is Good Now (2016) (blog.glyph.im)

Learn More (2)

- Poetry versus uv (loopwerk.io)
- Trying out PDM (and comparing it with Poetry and uv) (loopwerk.io)
- Revisiting uv (loopwerk.io)
- How to migrate your Poetry project to uv (loopwerk.io)
- Zero-copy deserialization (rkyv.org)
- uv docs (astral.sh)
- Nice uv cheatsheet (dev.to)