uv: Speed Meets Simplicity in Python Package Management

PyWeb-IL Meetup, March 2025

Yonatan Bitton
@bityob

bit.ly/uv-intro

`whoami`

- Developer on the Core team at **Perception Point**, recently acquired by **Fortinet**
- We provide cybersecurity protection for emails and more
- Passionate about **Python**, enjoy solving tough **challenges** and understanding things **inside out**

Agenda

- What's Wrong with Python Packaging?
- Meet uv
- Getting Started (Install, Commands, Features)
- The uv Ecosystem (Scripts & Tools)
- Why uv is So Fast?

The Painful Past of Python Packaging

The Painful Past of Python Packaging

- Python packaging can be hard for beginners:
  - How to get started?
  - How to use virtual environments?

The Painful Past of Python Packaging

- Python packaging can be hard for beginners:
  - How to get started?
  - How to use virtual environments?
- Managing Python versions

The Painful Past of Python Packaging

- Python packaging can be hard for beginners:
  - How to get started?
  - How to use virtual environments?
- Managing Python versions
- No unified tool for all workflows

The Painful Past of Python Packaging

- Python packaging can be hard for beginners:
  - How to get started?
  - How to use virtual environments?
- Managing Python versions
- 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

New Tool, Big Impact

New Tool, Big Impact

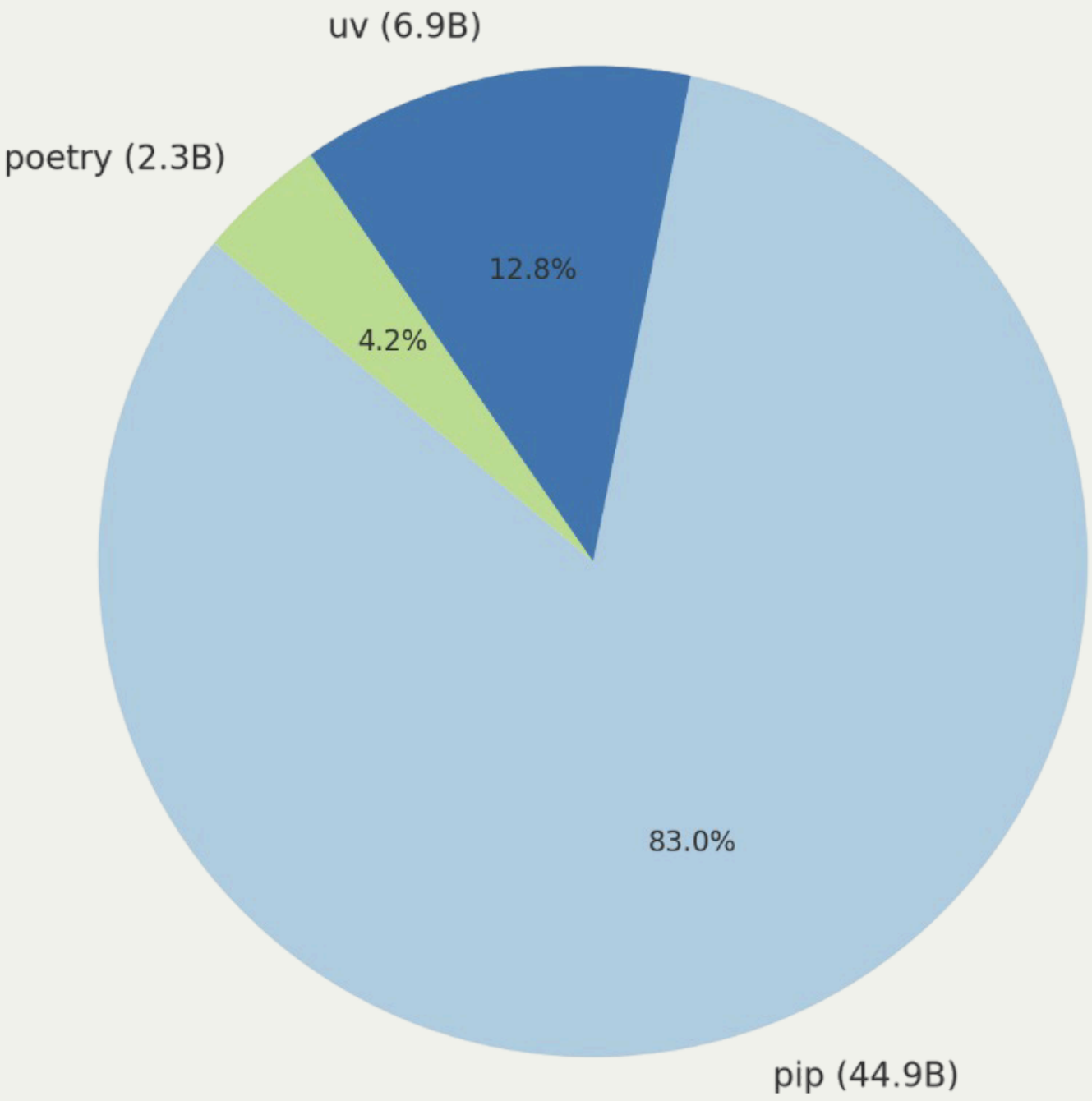
- 29+ million downloads last month

New Tool, Big Impact

- 29+ million downloads last month
- 44.4k stars in GitHub

New Tool, Big Impact

- 29+ million downloads last month
- 44.4k stars in GitHub
- 6.9+ billion packages were downloaded using uv last month, which is 13% of total downloads


uv (6.9B)
poetry (2.3B)
12.8%
4.2%
83.0%
pip (44.9B)

Getting Started

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

- Drop-in replacement for common `pip` and `virtualenv` commands
- For legacy workflows or when high-level commands lack control
- 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
```

```
2 Using CPython 3.10.14
3 Creating virtual environment at: .venv
4 Activate with: source .venv/bin/activate
5 $ uv pip install flask
6 Resolved 7 packages in 491ms
7 Installed 7 packages in 13ms
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
```

```
 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
```

Python Versions	
`uv python list`	View available and installed Python versions
`uv python install <version>`	Install python using precompiled binaries

```
$ uv python list
cpython-3.14.0a5-linux-x86_64-gnu      <download available>
cpython-3.13.2-linux-x86_64-gnu        /home/bit/.local/share/uv/pyt
cpython-3.12.9-linux-x86_64-gnu        /home/linuxbrew/.linuxbrew/op
cpython-3.11.11-linux-x86_64-gnu       /home/bit/.local/share/uv/pyt
cpython-3.10.16-linux-x86_64-gnu       <download available>
cpython-3.10.15-linux-x86_64-gnu       /home/bit/.local/bin/python3.
cpython-3.10.12-linux-x86_64-gnu       /bin/python3 -> python3.10
cpython-3.9.21-linux-x86_64-gnu        /usr/bin/python3.9
...
$ 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
```

```
$ uv python list
cpython-3.14.0a5-linux-x86_64-gnu      <download available>
cpython-3.13.2-linux-x86_64-gnu        /home/bit/.local/share/uv/pyt
cpython-3.12.9-linux-x86_64-gnu        /home/linuxbrew/.linuxbrew/op
cpython-3.11.11-linux-x86_64-gnu       /home/bit/.local/share/uv/pyt
cpython-3.10.16-linux-x86_64-gnu       <download available>
cpython-3.10.15-linux-x86_64-gnu       /home/bit/.local/bin/python3.
cpython-3.10.12-linux-x86_64-gnu       /bin/python3 -> python3.10
cpython-3.9.21-linux-x86_64-gnu        /usr/bin/python3.9
...
$ 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
```

Projects	
`uv init <name>`	Create a new project with `pyproject.toml`, `src/` tree, `README.md`, etc.
`uv run <python_file>`	Run the python file inside the `venv`. `uv` downloads python if not exists and sync the `venv` before run

```
$ 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 run main.py
Using CPython 3.10.15
Creating virtual environment at: .venv
Hello from playground!
$ tree
```

```
$ 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 run main.py
Using CPython 3.10.15
Creating virtual environment at: .venv
Hello from playground!
$ tree
```

```
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 run main.py
12 Using CPython 3.10.15
13 Creating virtual environment at: .venv
14 Hello from playground!
15 $ tree
16 ├── .python-version
17 ├── .venv
18 │   ├── bin
19 │   │   ├── activate
20 │   │   ├── python3 -> python
```

```
11 $ uv run main.py
12 Using CPython 3.10.15
13 Creating virtual environment at: .venv
14 Hello from playground!
15 $ tree
16 ├── .python-version
17 ├── .venv
18 │   ├── bin
19 │   │   ├── activate
20 │   │   ├── python3 -> python
21 │   ├── lib
22 ├── README.md
23 ├── main.py
24 ├── pyproject.toml
25 └── uv.lock
```

```
11 $ uv run main.py
12 Using CPython 3.10.15
13 Creating virtual environment at: .venv
14 Hello from playground!
15 $ tree
16 ├── .python-version
17 ├── .venv
18 │   ├── bin
19 │   │   ├── activate
20 │   │   ├── python3 -> python
21 │   ├── lib
22 ├── README.md
23 ├── main.py
24 ├── pyproject.toml
25 └── uv.lock
```

- `pyproject.toml` file contains metadata about your project
- `uv.lock` file records exact dependency versions for reproducible installations, unlike `pyproject.toml`, which specifies broad requirements

Projects	
`uv add <package>`	Install the package in the `venv`. `uv` also adds it to the `pyproject.toml` and update the `uv.lock` file

Projects	
`uv lock`	Create the `uv.lock` file with all pinned dependencies
`uv sync`	Install all project's dependencies inside the `venv`

```
$ 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 |
           ||     ||
```

```
$ 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 |
           ||     ||
```

Tools

- CLI tools, e.g `mypy`, `llm`, `pre-commit`
- No `pyproect.toml` file needed
- Virtual environment included
- Same like `pipx` tool

Tools	
`uvx` / `uv tool run`	Run a tool in a temporary environment
`uv tool install`	Install a tool user-wide

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

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

```
$ uv tool install ipython
Resolved 16 packages in 9ms
Installed 16 packages in 164ms
...
Installed 2 executables: ipython, ipython3
$ ipython
Python 3.13.2 (main, Feb 12 2025, 14:51:17) [Clang 19.1.6 ]
Type 'copyright', 'credits' or 'license' for more information
IPython 9.0.2 -- An enhanced Interactive Python. Type '?' for help.
Tip: You can change the editing mode of IPython to behave more like

In [1]:
```

```
$ uv tool install ipython
Resolved 16 packages in 9ms
Installed 16 packages in 164ms
...
Installed 2 executables: ipython, ipython3
$ ipython
Python 3.13.2 (main, Feb 12 2025, 14:51:17) [Clang 19.1.6 ]
Type 'copyright', 'credits' or 'license' for more information
IPython 9.0.2 -- An enhanced Interactive Python. Type '?' for help.
Tip: You can change the editing mode of IPython to behave more like

In [1]:
```

Real-world examples of advanced usage

Install Python kernels in Jupyter Notebooks for all versions

```
for i in {10..14}; \
    do uvx --python 3.$i --with ipykernel \
    python -m ipykernel install --user \
    --name python3.$i \
    --display-name python3.$i; \
done
```

```
Installed kernelspec python3.10 in .../jupyter/kernels/python3.10
Installed kernelspec python3.11 in .../jupyter/kernels/python3.11
Installed kernelspec python3.12 in .../jupyter/kernels/python3.12
Installed kernelspec python3.13 in .../jupyter/kernels/python3.13
Installed kernelspec python3.14 in .../jupyter/kernels/python3.14
```

Finding when a Python behavior changed

```
for i in {9..12}; \
    do uvx --python 3.$i \
    python -c 'import sys; \
    print(sys.version); \
    from urllib.parse import urlunsplit; \
    print(urlunsplit(("http", "", "google.com", "","")))'; \
done
```

```
3.9.21 (main, Dec  4 2024, 08:53:33) [GCC 11.4.0]
http:///google.com
3.10.15 (main, Oct 16 2024, 04:37:23) [Clang 18.1.8 ]
http:///google.com
3.11.11 (main, Dec 19 2024, 14:33:27) [Clang 18.1.8 ]
http:///google.com
3.12.9 (main, Feb 12 2025, 14:50:50) [Clang 19.1.6 ]
http:google.com
```

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??

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**

```
1  $ cat requests_example.py
2  import requests
3  uuid = requests.get("https://httpbin.org/uuid").json()
4  print(uuid)
5  $ uv run requests_example.py
6  Traceback (most recent call last):
7    File "/requests_example.py", line 1, in <module>
8      import requests
9  ModuleNotFoundError: No module named 'requests'
10 $ uv run --with requests requests_example.py
11 Installed 5 packages in 42ms
12 {'uuid': 'f0769a81-9c86-4187-aa3f-a9178bca216f'}
```

```
$ cat requests_example.py
import requests
uuid = requests.get("https://httpbin.org/uuid").json()
print(uuid)
$ uv run requests_example.py
Traceback (most recent call last):
  File "/requests_example.py", line 1, in <module>
    import requests
ModuleNotFoundError: No module named 'requests'
$ uv run --with requests requests_example.py
Installed 5 packages in 42ms
{'uuid': 'f0769a81-9c86-4187-aa3f-a9178bca216f'}
```

```
1  $ cat requests_example.py
2  import requests
3  uuid = requests.get("https://httpbin.org/uuid").json()
4  print(uuid)
5  $ uv run requests_example.py
6  Traceback (most recent call last):
7    File "/requests_example.py", line 1, in <module>
8      import requests
9  ModuleNotFoundError: No module named 'requests'
10 $ uv run --with requests requests_example.py
11 Installed 5 packages in 42ms
12 {'uuid': 'f0769a81-9c86-4187-aa3f-a9178bca216f'}
```

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

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

Now we can use something like this

```
$ uv add --script requests_example.py requests
Updated `requests_example.py`
$ cat requests_example.py
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "requests",
# ]
# ///
import requests
uuid = requests.get("https://httpbin.org/uuid").json()
print(uuid)
$ uv run requests_example.py
Installed 5 packages in 51ms
{'uuid': '373e6644-b7ae-434e-a15e-0135774d05bf'}
```

Now we can use something like this

```
$ uv add --script requests_example.py requests
Updated `requests_example.py`
$ cat requests_example.py
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "requests",
# ]
# ///
import requests
uuid = requests.get("https://httpbin.org/uuid").json()
print(uuid)
$ uv run requests_example.py
Installed 5 packages in 51ms
{'uuid': '373e6644-b7ae-434e-a15e-0135774d05bf'}
```

Now we can use something like this

```
$ uv add --script requests_example.py requests
Updated `requests_example.py`
$ cat requests_example.py
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "requests",
# ]
# ///
import requests
uuid = requests.get("https://httpbin.org/uuid").json()
print(uuid)
$ uv run requests_example.py
Installed 5 packages in 51ms
{'uuid': '373e6644-b7ae-434e-a15e-0135774d05bf'}
```

We can also limit uv to distributions available when the script was created

```
$ cat requests_version.py
# /// script
# dependencies = [
#   "requests",
# ]
# [tool.uv]
# exclude-newer = "2020-10-16T00:00:00Z"
# ///
import requests
print(requests.__version__)
$ uv run requests_version.py
Reading inline script metadata from `example_requests.py`
Installed 5 packages in 40ms
2.24.0
```

We can also limit uv to distributions available when the script was created

```
$ cat requests_version.py
# /// script
# dependencies = [
#   "requests",
# ]
# [tool.uv]
# exclude-newer = "2020-10-16T00:00:00Z"
# ///
import requests
print(requests.__version__)
$ uv run requests_version.py
Reading inline script metadata from `example_requests.py`
Installed 5 packages in 40ms
2.24.0
```

We can also limit uv to distributions available when the script was created

```
$ cat requests_version.py
# /// script
# dependencies = [
#   "requests",
# ]
# [tool.uv]
# exclude-newer = "2020-10-16T00:00:00Z"
# ///
import requests
print(requests.__version__)
$ uv run requests_version.py
Reading inline script metadata from `example_requests.py`
Installed 5 packages in 40ms
2.24.0
```

We can also limit uv to distributions available when the script was created

```
$ cat requests_version.py
# /// script
# dependencies = [
#   "requests",
# ]
# [tool.uv]
# exclude-newer = "2020-10-16T00:00:00Z"
# ///
import requests
print(requests.__version__)
$ uv run requests_version.py
Reading inline script metadata from `example_requests.py`
Installed 5 packages in 40ms
2.24.0
```

Source: https://docs.astral.sh/uv/guides/scripts/

Now, the interesting part...

Now, the interesting part...

Why uv is so fast?

Pure Rust

Written in pure **Rust**, 100k~ lines of code.

Fast python install using precompiled binaries

▾ Assets 1,171

cpython-3.10.16+20250317-aarch64-apple-darwin-debug-full.tar.zst	4	22.1 MB	last week
cpython-3.10.16+20250317-aarch64-apple-darwin-debug-full.tar.zst.sha256	4	65 Bytes	last week
cpython-3.10.16+20250317-aarch64-apple-darwin-install_only.tar.gz	75	16.7 MB	last week
cpython-3.10.16+20250317-aarch64-apple-darwin-install_only.tar.gz.sha256	14	65 Bytes	last week
cpython-3.10.16+20250317-aarch64-apple-darwin-install_only_stripped.tar.gz	5.2k	16.6 MB	last week
cpython-3.10.16+20250317-aarch64-apple-darwin-install_only_stripped.tar.gz.sha256	21	65 Bytes	last week
cpython-3.10.16+20250317-aarch64-apple-darwin-pgo+lto-full.tar.zst	13	31.4 MB	last week
cpython-3.10.16+20250317-aarch64-apple-darwin-pgo+lto-full.tar.zst.sha256	9	65 Bytes	last week
cpython-3.10.16+20250317-aarch64-unknown-linux-gnu-debug-full.tar.zst	9	32.1 MB	last week
cpython-3.10.16+20250317-aarch64-unknown-linux-gnu-debug-full.tar.zst.sha256	4	65 Bytes	last week
cpython-3.10.16+20250317-aarch64-unknown-linux-gnu-install_only.tar.gz	13	23.9 MB	last week
cpython-3.10.16+20250317-aarch64-unknown-linux-gnu-install_only.tar.gz.sha256	11	65 Bytes	last week

https://github.com/astral-sh/python-build-standalone/releases/tag/20250317

Version parsing

Version parsing

- `1.0.0`

Version parsing

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

Version parsing

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

Version parsing

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

Version parsing

- `1.0.0`
- `2.3.1-beta.1`
- `1.0.post345`
- `1.2.3.4.5-a8.post9`
- `2025.3.24.pre123`

Version parsing

- `1.0.0`
- `2.3.1-beta.1`
- `1.0.post345`
- `1.2.3.4.5-a8.post9`
- `2025.3.24.pre123`

Representing these is pretty hard...

Version parsing

- `1.0.0`
- `2.3.1-beta.1`
- `1.0.post345`
- `1.2.3.4.5-a8.post9`
- `2025.3.24.pre123`

Representing these is pretty hard...

Source: https://youtu.be/gSKTfG1GXYQ?t=1544

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>,
}
```

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>,
}
```

Note, the `release` field is a vector...

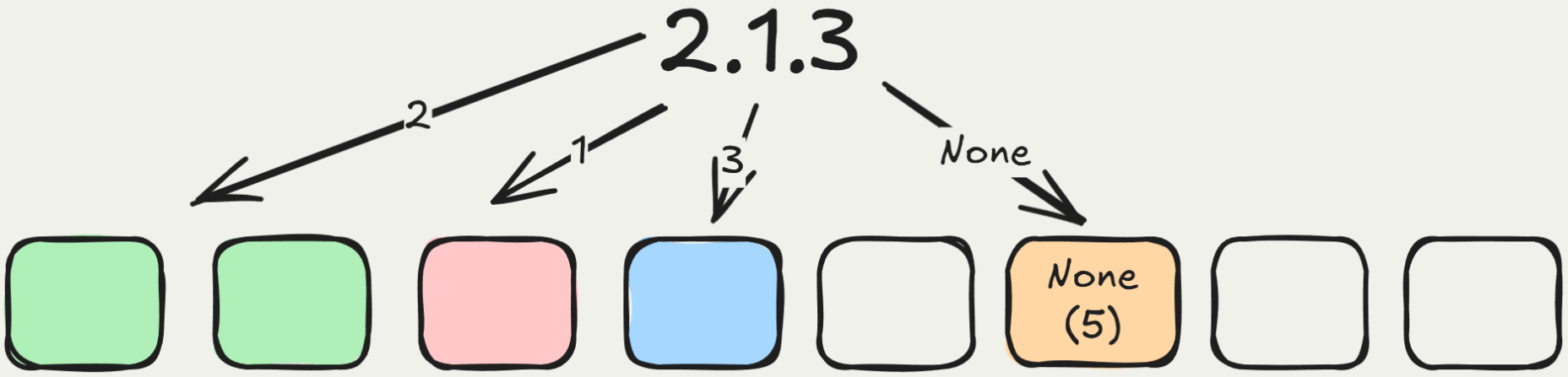
We can represent over 90% of versions with only 64 bits, making it extremely space-efficient

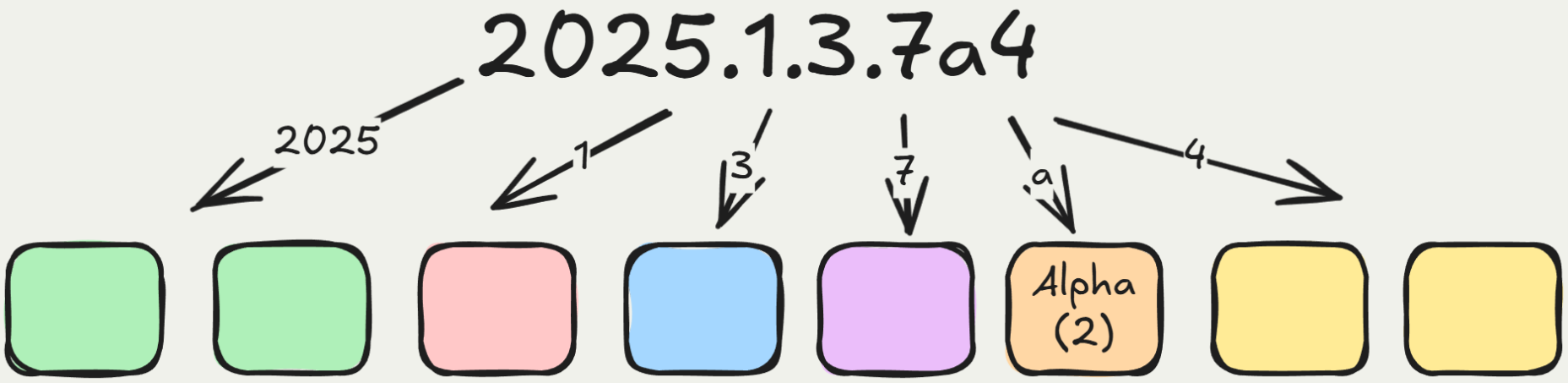
```
struct VersionSmall(u64);
```

We can represent over 90% of versions with only 64 bits, making it extremely space-efficient

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

Credit to Andrew Gallant (@BurntSushi)


2.1.3
2
1
3
None
None
(5)


2025.1.3.7a4
2025
1
3
7
a
4
Alpha
(2)

And the best part

And the best part

- Greater versions map to larger integers → Simplifies comparison

And the best part

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

And the best part

- Greater versions map to larger integers → Simplifies comparison
- Version comparison becomes extremely fast
- `1.2.3a1` < `1.2.3`

And the best part

- Greater versions map to larger integers → Simplifies comparison
- Version comparison becomes extremely fast
- `1.2.3a1` < `1.2.3`

```
1.2.3a1  →  0001 | 0002 | 0003 | *0010* | 0000 | 0001  (Alpha = 2)
1.2.3    →  0001 | 0002 | 0003 | *0101* | 0000 | 0000  (Normal = 5)
```

Reading package `METADATA`

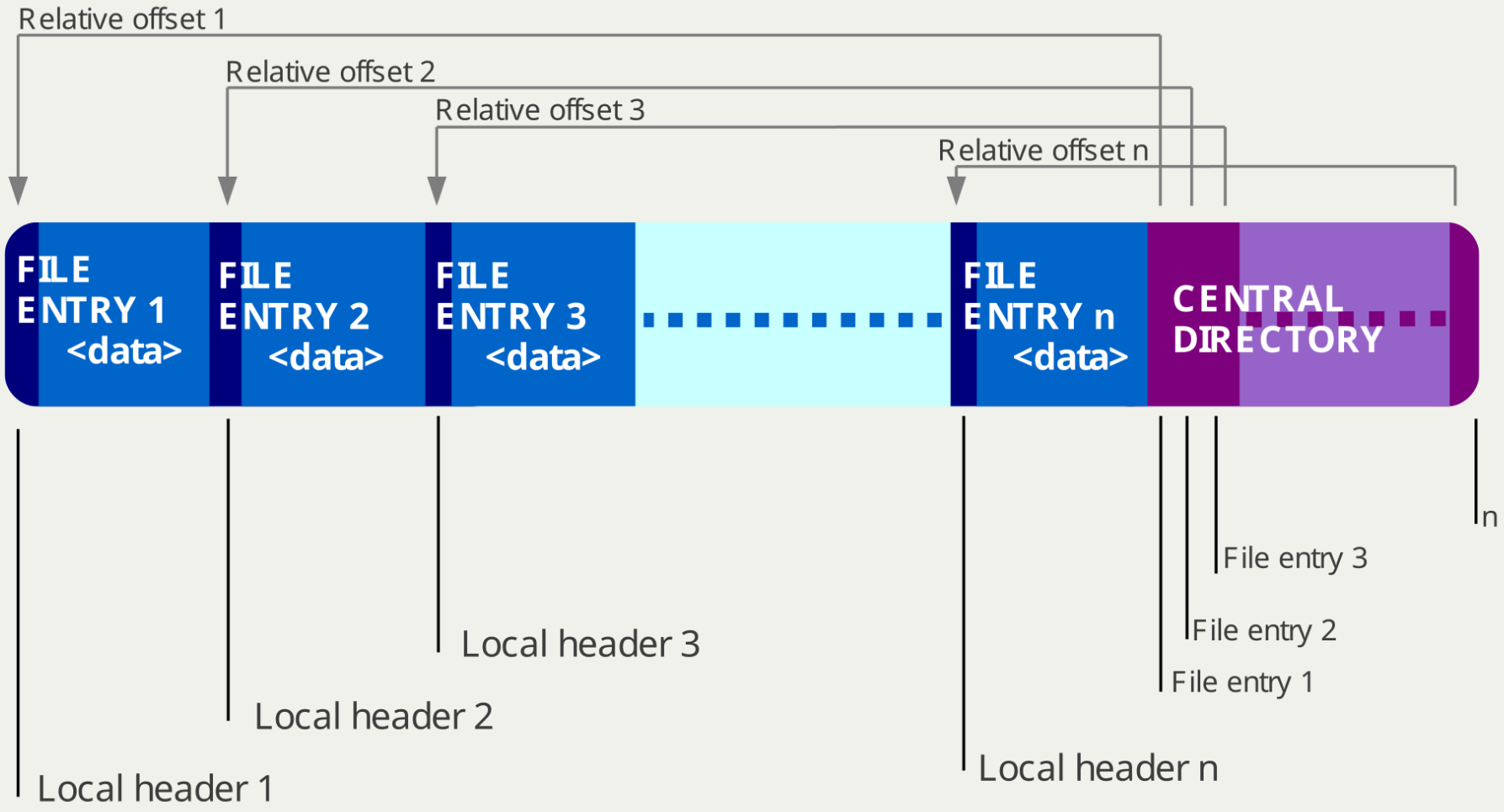
- Built distributions (wheels) are packaged as ZIP archives
- Somewhere within the ZIP file, there's a `METADATA` file
- Some registries expose the `METADATA` file directly, while others do not

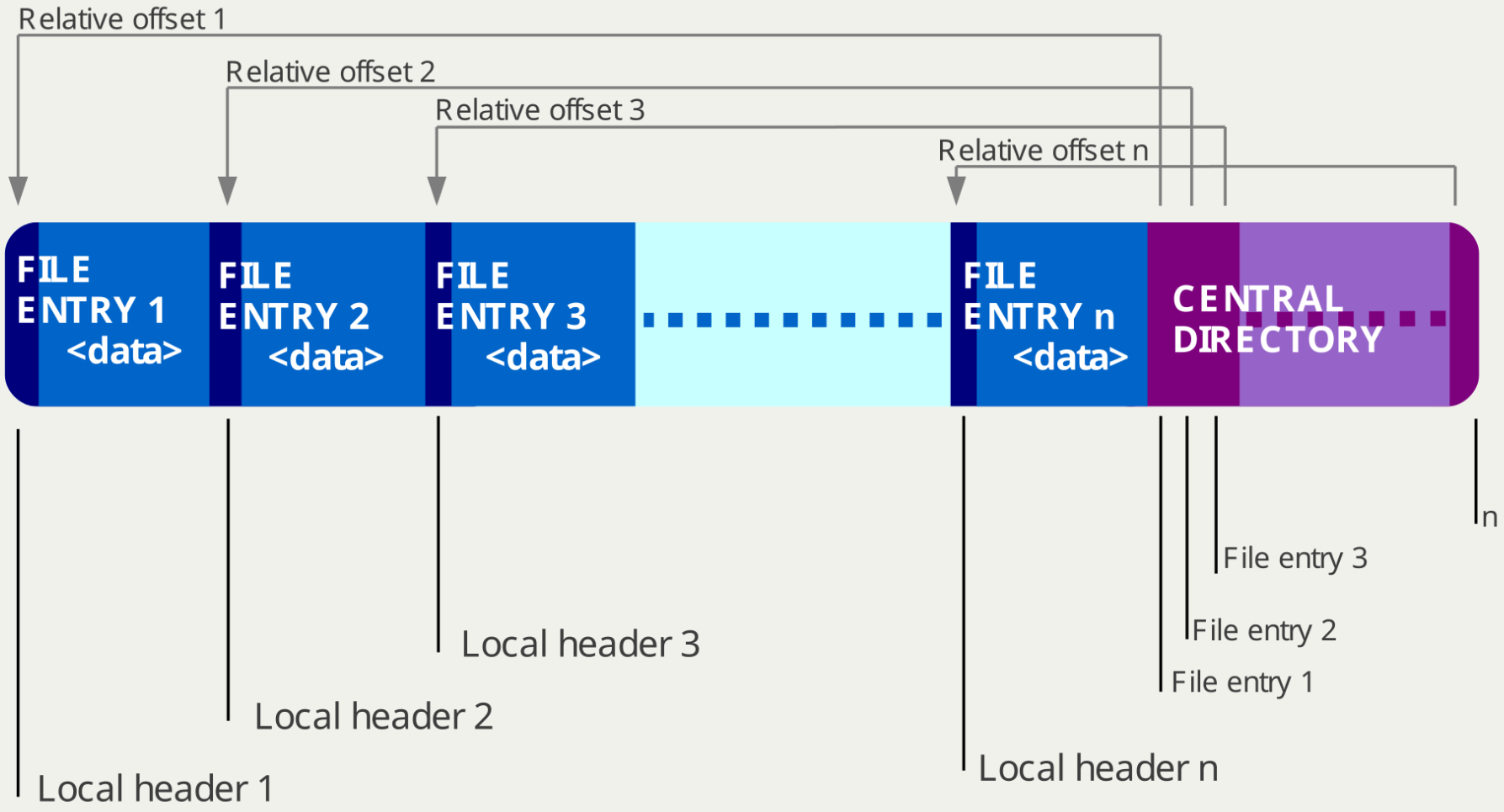
Reading package `METADATA`

- 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://youtu.be/gSKTfG1GXYQ?t=1884

How to avoid downloading the entire ZIP archive just to read the `METADATA` file and determine package dependencies?


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 n
Local header 1



Source: Wikipedia

Solution

Solution

- HTTP Range request the Central Directory

Solution

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

Solution

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

Solution

- 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: pypi.org
Range: bytes=-100

GET /example.whl HTTP/1.1
Host: pypi.org
Range: bytes=300-400
```

Cache design

- Global cache of unpacked archives
  - uv uses caching to avoid re-downloading dependencies that have already been accessed in prior run
- Most installs are just **hardlinks** from one place to another
- Really fast and very space efficient

Zero-copy deserialization

Zero-copy deserialization

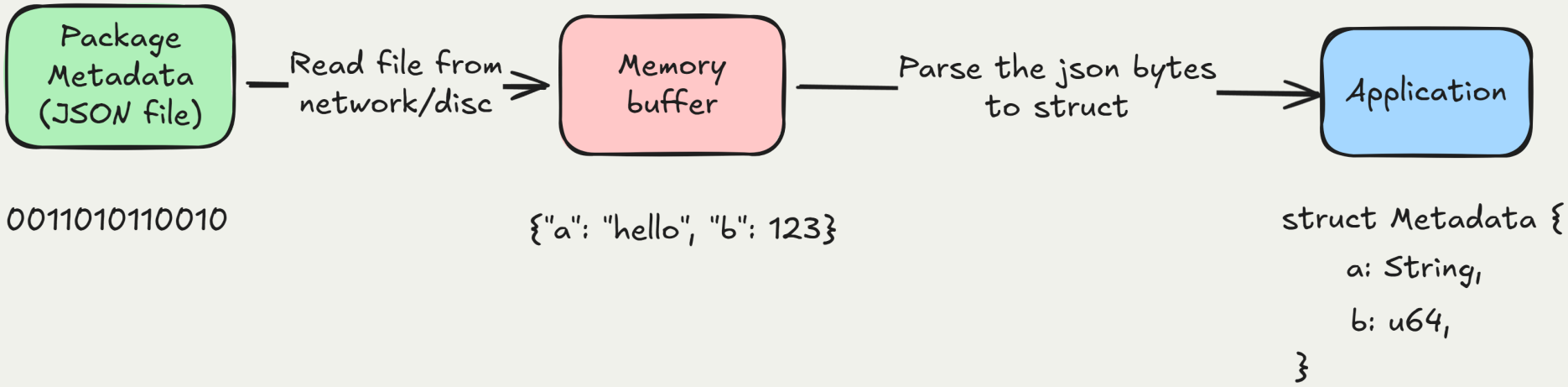
A technique that reduces the time and memory required to access and use data by **directly referencing bytes in the serialized form**

Zero-copy deserialization

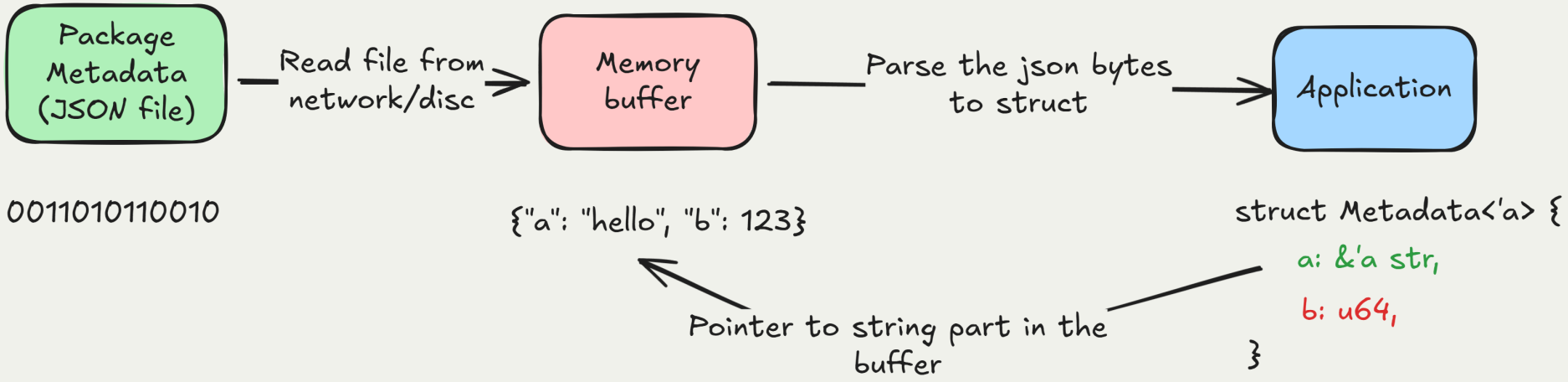
A technique that reduces the time and memory required to access and use data by **directly referencing bytes in the serialized form**

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

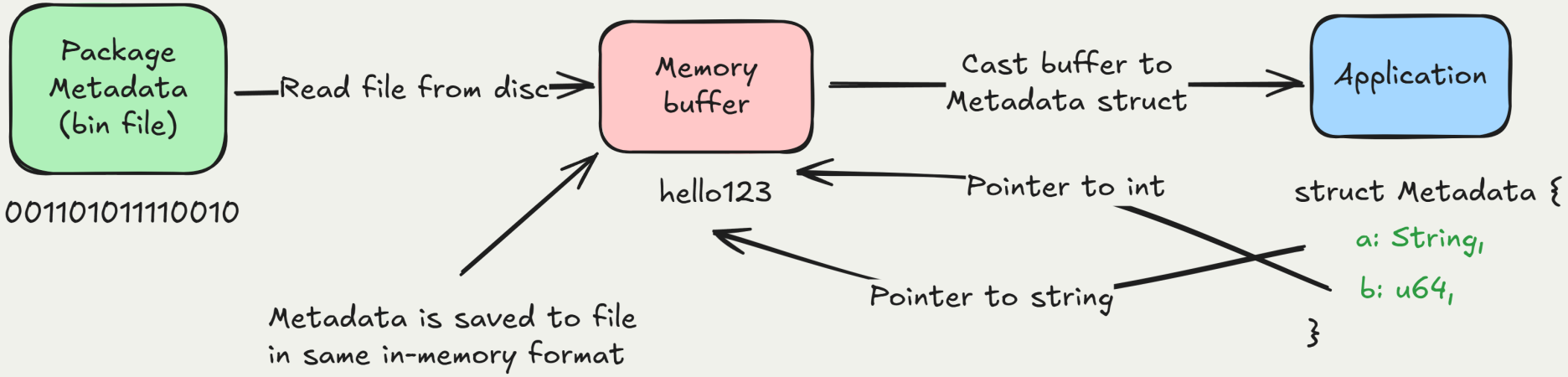
Normal JSON parsing



Simple Zero Copy



Total Zero Copy



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

Thank You!

Any Questions?

bit.ly/uv-intro

Yonatan Bitton
@bityob

linktr.ee/bityob

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)
- uv version.rs code (github.com)