

# uv: Speed Meets Simplicity in Python Package Management

**PyWeb-IL Meetup, March 2025**

Yonatan Bitton [bit.ly/uv-intro](https://bit.ly/uv-intro)

@bityob



# 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

[https://gitlab.com/ScalableML/PyTorch-Model-Compression/-/blob/main/torch\\_mlir/test/test\\_torch\\_mlir.py](https://gitlab.com/ScalableML/PyTorch-Model-Compression/-/blob/main/torch_mlir/test/test_torch_mlir.py)

# New Tool, Big Impact

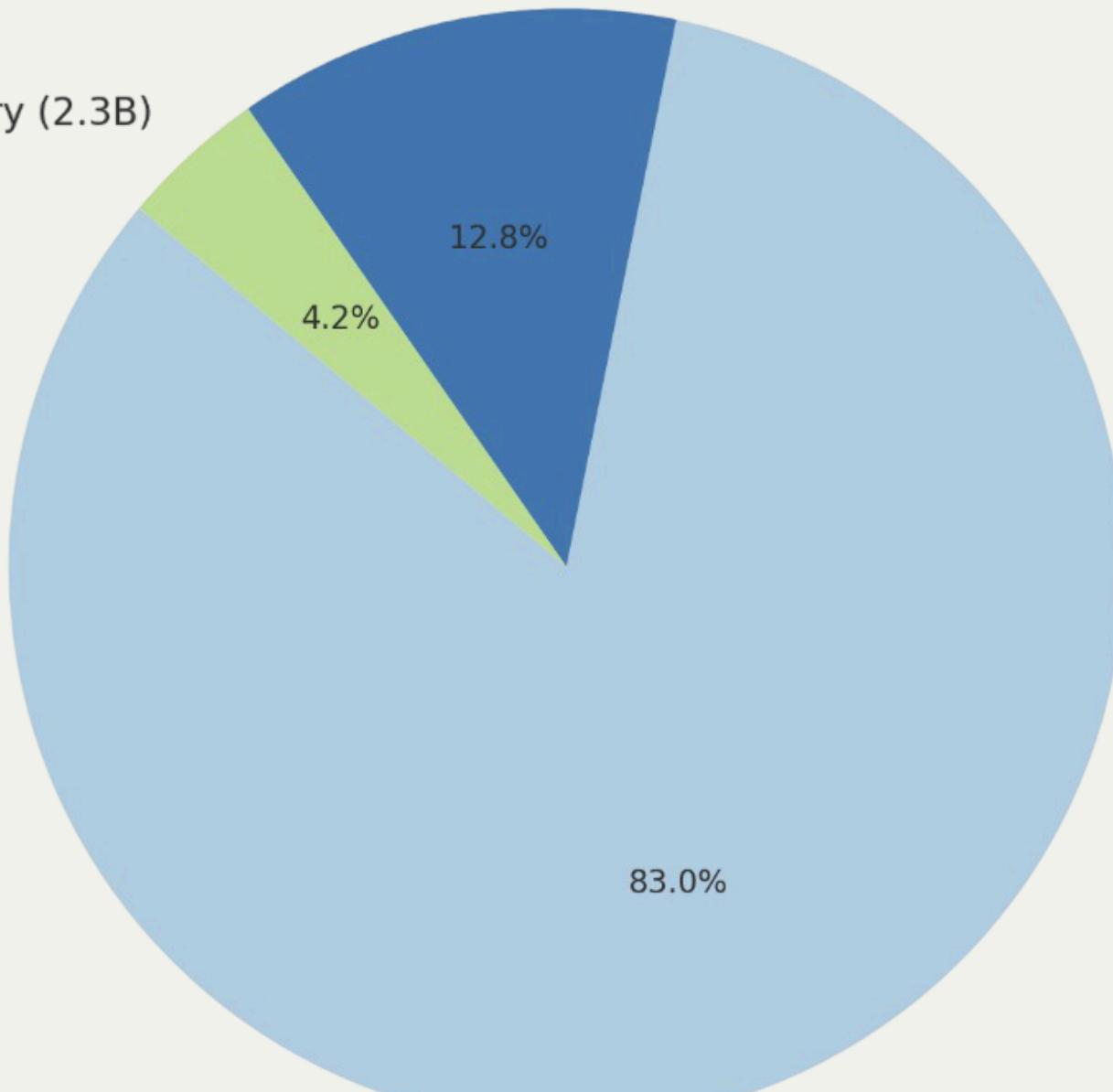
- 29+ million downloads last month

# 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



# 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`

```
1 $ uv venv
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 uv[1.7.1] - 1.7.1
```

```
2 Using Python 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
18
```

```
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

```
1 $ uv python list
2 cpython-3.14.0a5-linux-x86_64-gnu
3 cpython-3.13.2-linux-x86_64-gnu
4 cpython-3.12.9-linux-x86_64-gnu
5 cpython-3.11.11-linux-x86_64-gnu
6 cpython-3.10.16-linux-x86_64-gnu
7 cpython-3.10.15-linux-x86_64-gnu
8 cpython-3.10.12-linux-x86_64-gnu
9 cpython-3.9.21-linux-x86_64-gnu
10 ...
11 $ uv python install cpython-3.10.15-linux-x86_64-gnu
12 Installed Python 3.10.15 in 4.74s
13 + cpython-3.10.15-linux-x86_64-gnu
14
15 <download available>
16 /home/bit/.local/share/
17 /home/linuxbrew/.linuxbrew
18 /home/bit/.local/share/
19 <download available>
20 /home/bit/.local/bin/python3
21 /bin/python3 -> python3
22 /usr/bin/python3.9
```

```
1 $ uv python list
2 cpython-3.14.0a5-linux-x86_64-gnu
3 cpython-3.13.2-linux-x86_64-gnu
4 cpython-3.12.9-linux-x86_64-gnu
5 cpython-3.11.11-linux-x86_64-gnu
6 cpython-3.10.16-linux-x86_64-gnu
7 cpython-3.10.15-linux-x86_64-gnu
8 cpython-3.10.12-linux-x86_64-gnu
9 cpython-3.9.21-linux-x86_64-gnu
10 ...
11 $ uv python install cpython-3.10.15-linux-x86_64-gnu
12 Installed Python 3.10.15 in 4.74s
13 + cpython-3.10.15-linux-x86_64-gnu
14
15 <download available>
16 /home/bit/.local/share/
17 /home/linuxbrew/.linuxbrew
18 /home/bit/.local/share/
19 <download available>
20 /home/bit/.local/bin/py
21 /bin/python3 -> python3
22 /usr/bin/python3.9
```

# 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

```
1 $ uv init
2 Initialized project `playground`
3 $ cat pyproject.toml
4 [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 run main.py
12 Using CPython 3.10.15
13 Creating virtual environment at: .venv
14 Hello from playground!
15 $ tree
16 |
```

```
1 $ uv init
2 Initialized project `playground`
3 $ cat pyproject.toml
4 [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 run main.py
12 Using CPython 3.10.15
13 Creating virtual environment at: .venv
14 Hello from playground!
15 $ tree
16 |
```

```
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`

```
1 $ uv add pycowsay
2 Resolved 2 packages in 346ms
3 Installed 1 package in 7ms
4 ++ pycowsay==0.0.0.2
5 $ uv run pycowsay "Hello Python World!"
6 -----
7 < Hello Python World! >
8 -----
9   \   ^   ^
10    \   (oo) \_____
11      (--) \        )\/\
12          ||----w |
13          ||         |
```

```
1 $ uv add pycowsay
2 Resolved 2 packages in 346ms
3 Installed 1 package in 7ms
4 ++ pycowsay==0.0.0.2
5 $ uv run pycowsay "Hello Python World!"
6 -----
7 < Hello Python World! >
8 -----
9   \   ^   ^
10    \   (oo) \_____
11      (____) \        )\/\
12          ||----w |
13          ||         |
```

# Tools

- CLI tools, e.g mypy, ldm, pre-commit
- No pyproject.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!"
```

```
1 $ uv tool install ipython
2 Resolved 16 packages in 9ms
3 Installed 16 packages in 164ms
4 ...
5 Installed 2 executables: ipython, ipython3
6 $ ipython
7 Python 3.13.2 (main, Feb 12 2025, 14:51:17) [Clang 19.1.6 ]
8 Type 'copyright', 'credits' or 'license' for more information
9 IPython 9.0.2 -- An enhanced Interactive Python. Type '?' for
10 Tip: You can change the editing mode of IPython to behave more
11
12 In [1]:
```

```
1 $ uv tool install ipython
2 Resolved 16 packages in 9ms
3 Installed 16 packages in 164ms
4 ...
5 Installed 2 executables: ipython, ipython3
6 $ ipython
7 Python 3.13.2 (main, Feb 12 2025, 14:51:17) [Clang 19.1.6 ]
8 Type 'copyright', 'credits' or 'license' for more information
9 IPython 9.0.2 -- An enhanced Interactive Python. Type '?' for
10 Tip: You can change the editing mode of IPython to behave more
11
12 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'}
```

```
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'}
```

```
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](#)

# Now we can use something like this

```
1 $ uv add --script requests_example.py requests
2 Updated `requests_example.py`
3 $ cat requests_example.py
4 # /**
5 #  * requires-python = ">=3.10"
6 # dependencies = [
7 #     "requests",
8 # ]
9 # /**
10 import requests
11 uuid = requests.get("https://httpbin.org/uuid").json()
12 print(uuid)
13 $ uv run requests_example.py
14 Installed 5 packages in 51ms
15 {'uuid': '373e6644-b7ae-434e-a15e-0135774d05bf'}
```

# Now we can use something like this

```
1 $ uv add --script requests_example.py requests
2 Updated `requests_example.py`
3 $ cat requests_example.py
4 # /**
5 #  * requires-python = ">=3.10"
6 # dependencies = [
7 #     "requests",
8 # ]
9 # /**
10 import requests
11 uuid = requests.get("https://httpbin.org/uuid").json()
12 print(uuid)
13 $ uv run requests_example.py
14 Installed 5 packages in 51ms
15 {'uuid': '373e6644-b7ae-434e-a15e-0135774d05bf'}
```

# Now we can use something like this

```
1 $ uv add --script requests_example.py requests
2 Updated `requests_example.py`
3 $ cat requests_example.py
4 # /**
5 #  * requires-python = ">=3.10"
6 # dependencies = [
7 #     "requests",
8 # ]
9 # /**
10 import requests
11 uuid = requests.get("https://httpbin.org/uuid").json()
12 print(uuid)
13 $ uv run requests_example.py
14 Installed 5 packages in 51ms
15 { 'uuid': '373e6644-b7ae-434e-a15e-0135774d05bf' }
```

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

```
1 $ cat requests_version.py
2 # /// script
3 # dependencies = [
4 #     "requests",
5 # ]
6 # [tool_uv]
7 # exclude_newer = "2020-10-16T00:00:00Z"
8 # /**
9 import requests
10 print(requests.__version__)
11 $ uv run requests_version.py
12 Reading inline script metadata from `example_requests.py`  

13 Installed 5 packages in 40ms
14 2.24.0
```

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

```
1 $ cat requests_version.py
2 # /// script
3 # dependencies = [
4 #     "requests",
5 # ]
6 # [tool_uv]
7 # exclude_newer = "2020-10-16T00:00:00Z"
8 # /**
9 import requests
10 print(requests.__version__)
11 $ uv run requests_version.py
12 Reading inline script metadata from `example_requests.py`  

13 Installed 5 packages in 40ms
14 2.24.0
```

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

```
1 $ cat requests_version.py
2 # /// script
3 # dependencies = [
4 #     "requests",
5 # ]
6 # [tool_uv]
7 # exclude_newer = "2020-10-16T00:00:00Z"
8 # /**
9 import requests
10 print(requests.__version__)
11 $ uv run requests_version.py
12 Reading inline script metadata from `example_requests.py`
13 Installed 5 packages in 40ms
14 2.24.0
```

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

```
1 $ cat requests_version.py
2 # /// script
3 # dependencies = [
4 #     "requests",
5 # ]
6 # [tool_uv]
7 # exclude_newer = "2020-10-16T00:00:00Z"
8 # /**
9 import requests
10 print(requests.__version__)
11 $ uv run requests_version.py
12 Reading inline script metadata from `example_requests.py`
13 Installed 5 packages in 40ms
14 2.24.0
```

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

<https://github.com/SonSonic/python/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...

# 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>,  
}
```

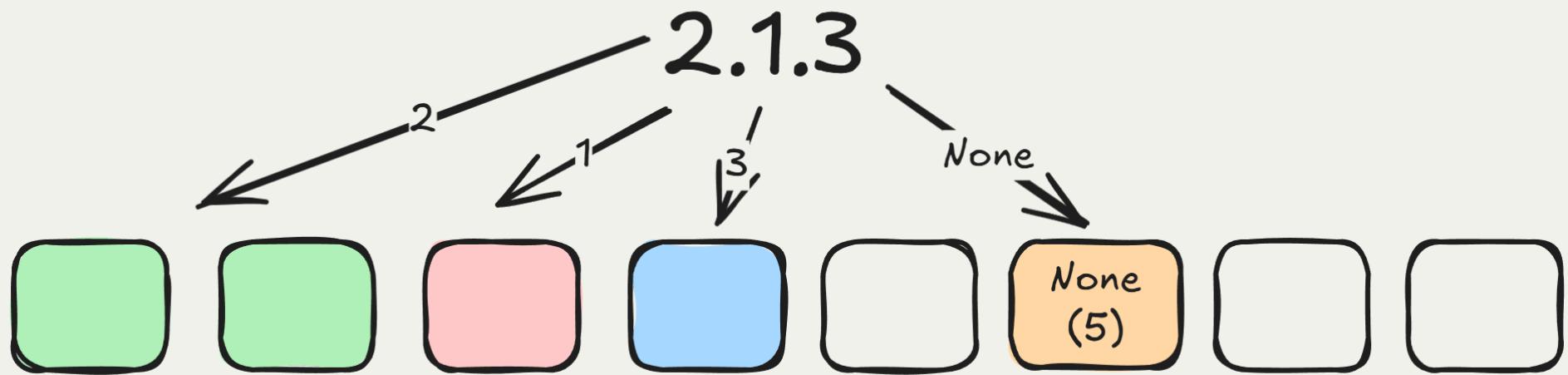
Note, the `release` field is a vector...

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

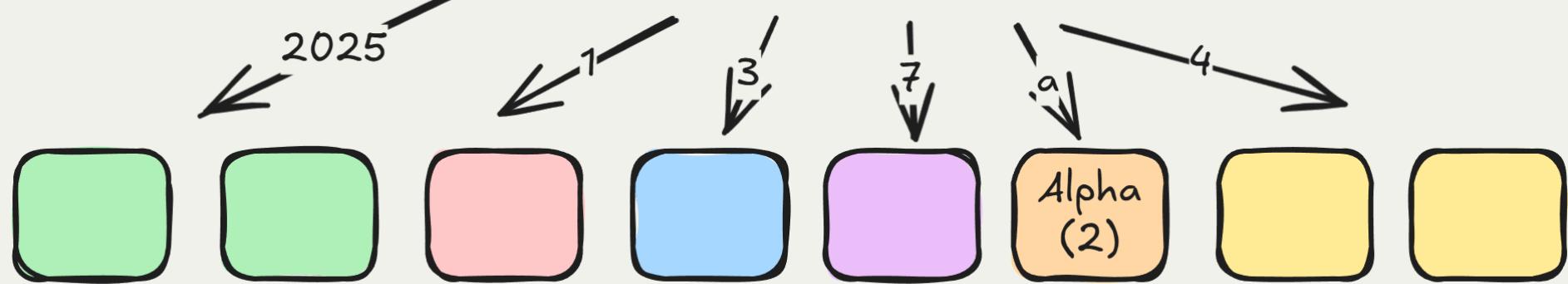
```
struct VersionSmall(u64);
```

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

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



2025.1.3.7a4



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 =
```

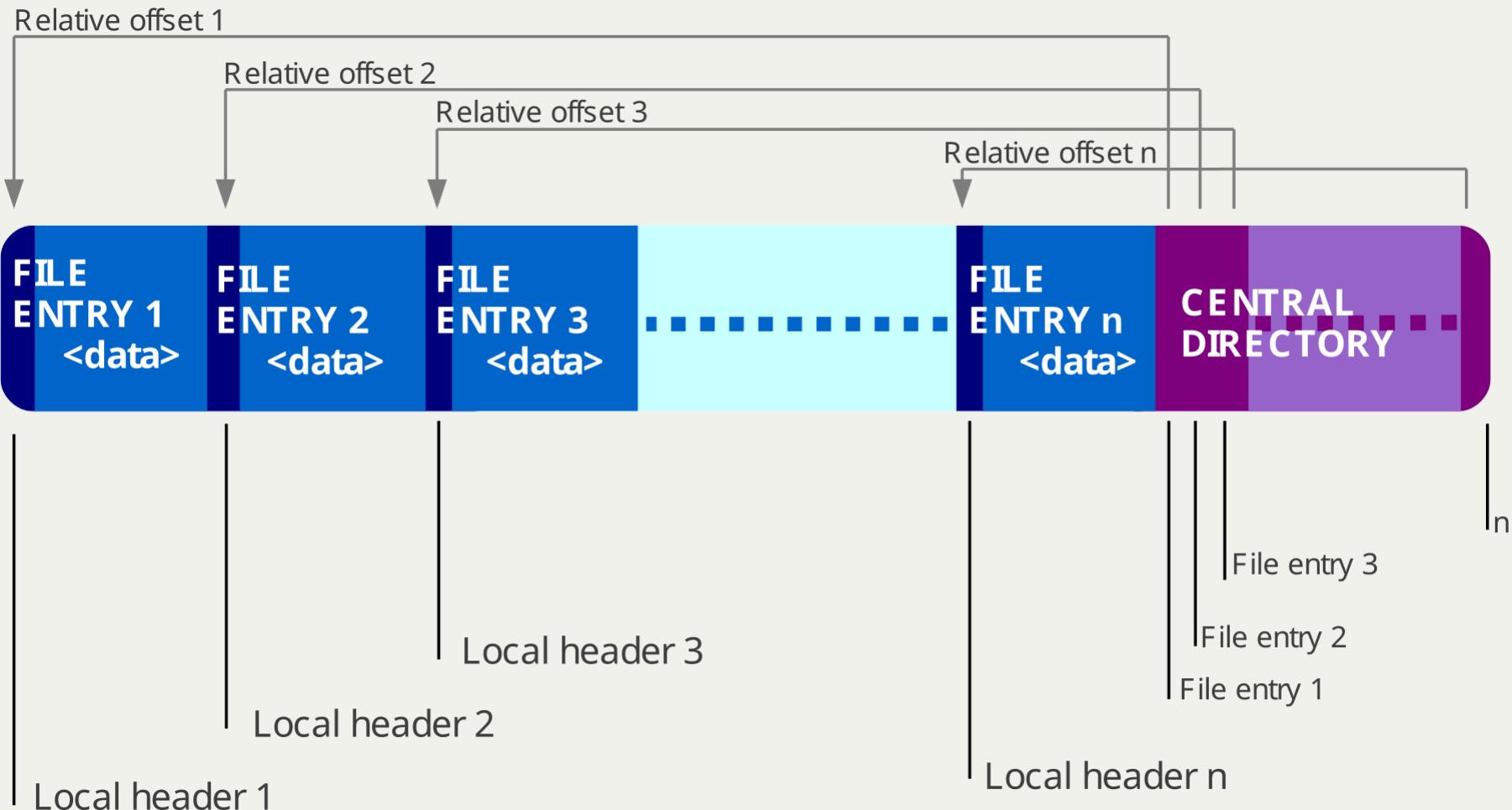
# Reading package METADATA

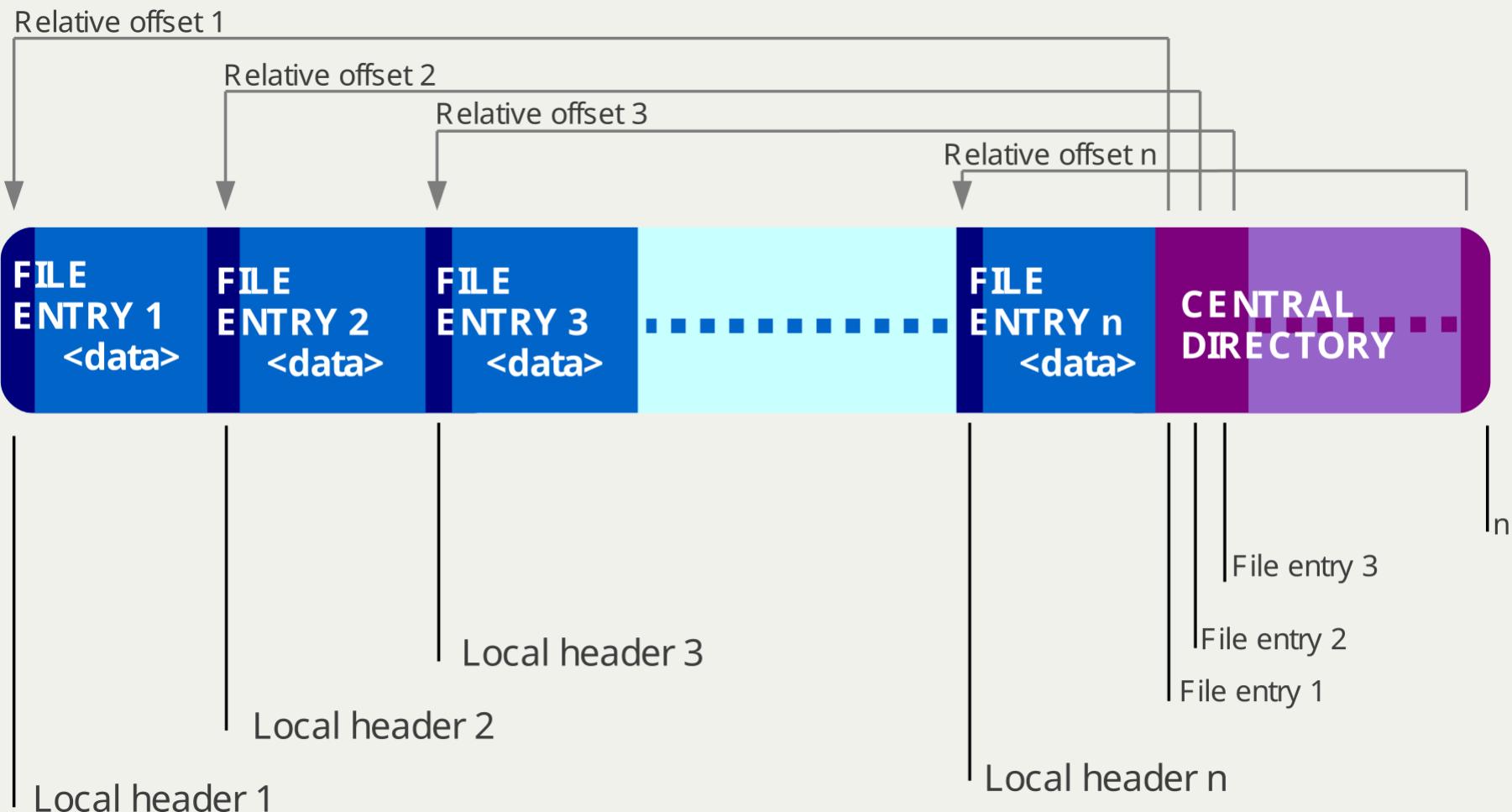
- Built distributions (wheels) are packaged as ZIP archives
- Somewhere within the ZIP file, there's a METADATA file
- Some registries expose the METADATA file directly, while others do not

# 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

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





# Solution

[SouScu hcttplstt/prky/y.ougu/zbeogS0pLyfGeSexiaQzatR246.html](https://scriptplists.prky/y.ougu/zbeogS0pLyfGeSexiaQzatR246.html)

# 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

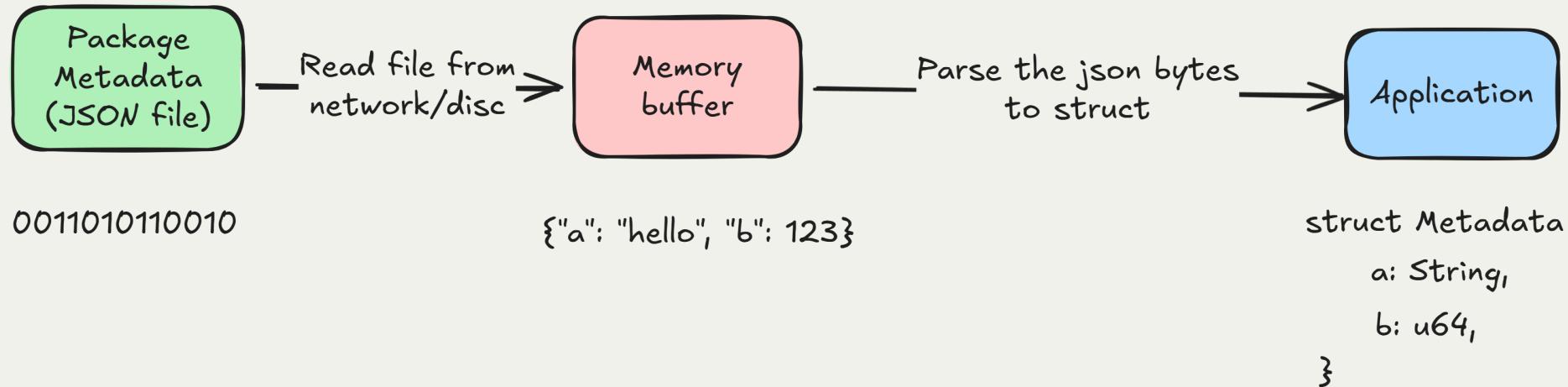
# Zero-copy deserialization

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

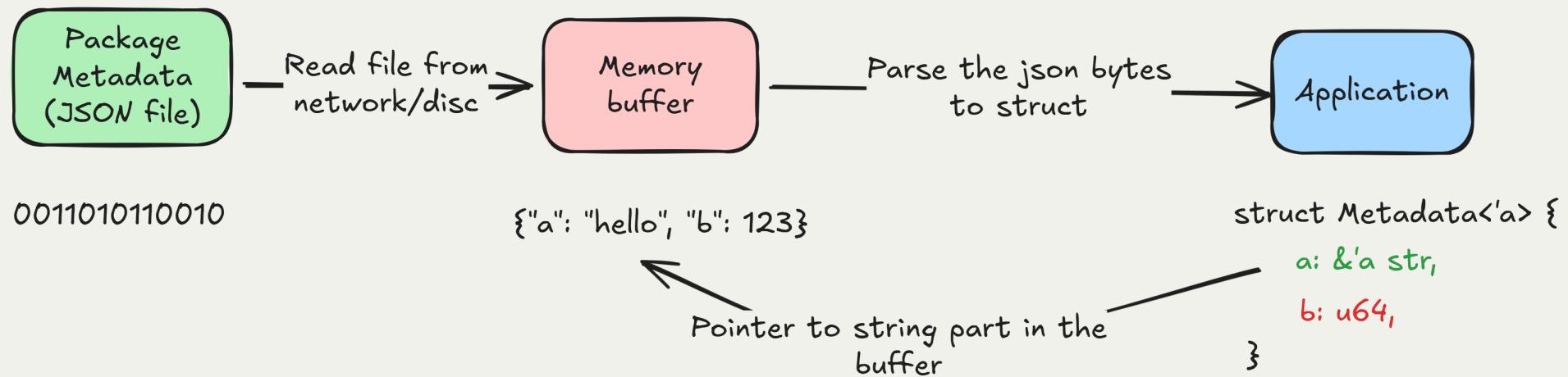
# Zero-copy deserialization

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

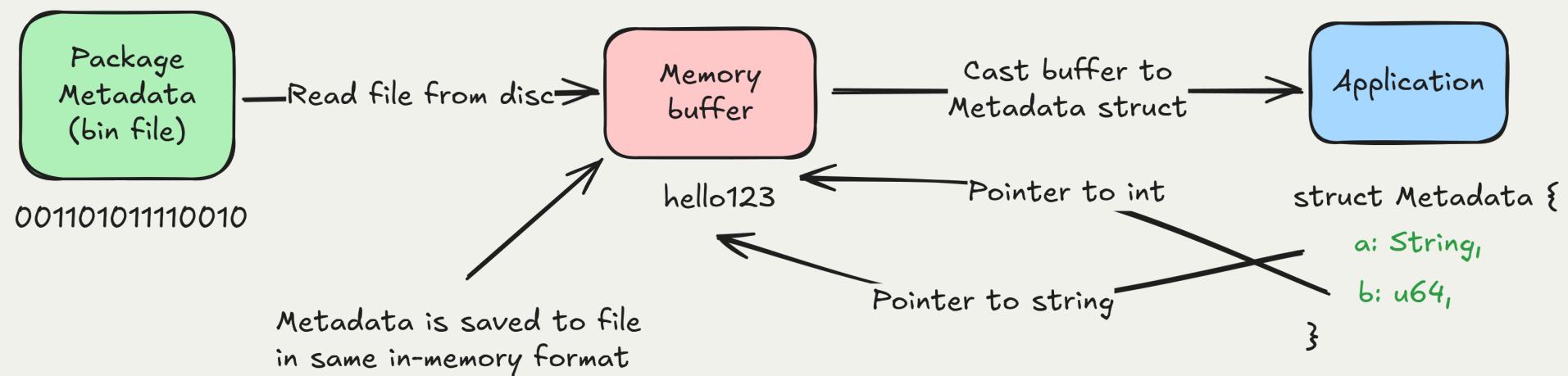
# Normal JSON parsing



# 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.

# 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](https://bit.ly/uv-intro)

Yonatan Bitton  
@bityob

[linktr.ee/bityob](https://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](https://blog.streamlit.io))
- GitHub - `astral-sh/uv`: An extremely fast Python package and project manager, written in Rust ([github](https://github.com/astral-sh/uv))
- Python Packaging is Great Now: uv is all you need ([dev.to](https://dev.to))
- Python Packaging Is Good Now (2016) ([blog.glyph.im](https://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](#))

<https://github.com/SonodaHiroaki/PyKUKI/releases/tag/20250317>