Reinventing the Wheel @ WheelNext Summit

Emma Smith, NVIDIA

March 21, 2025

PEP 777: Reinventing the Wheel

Anti-goals:

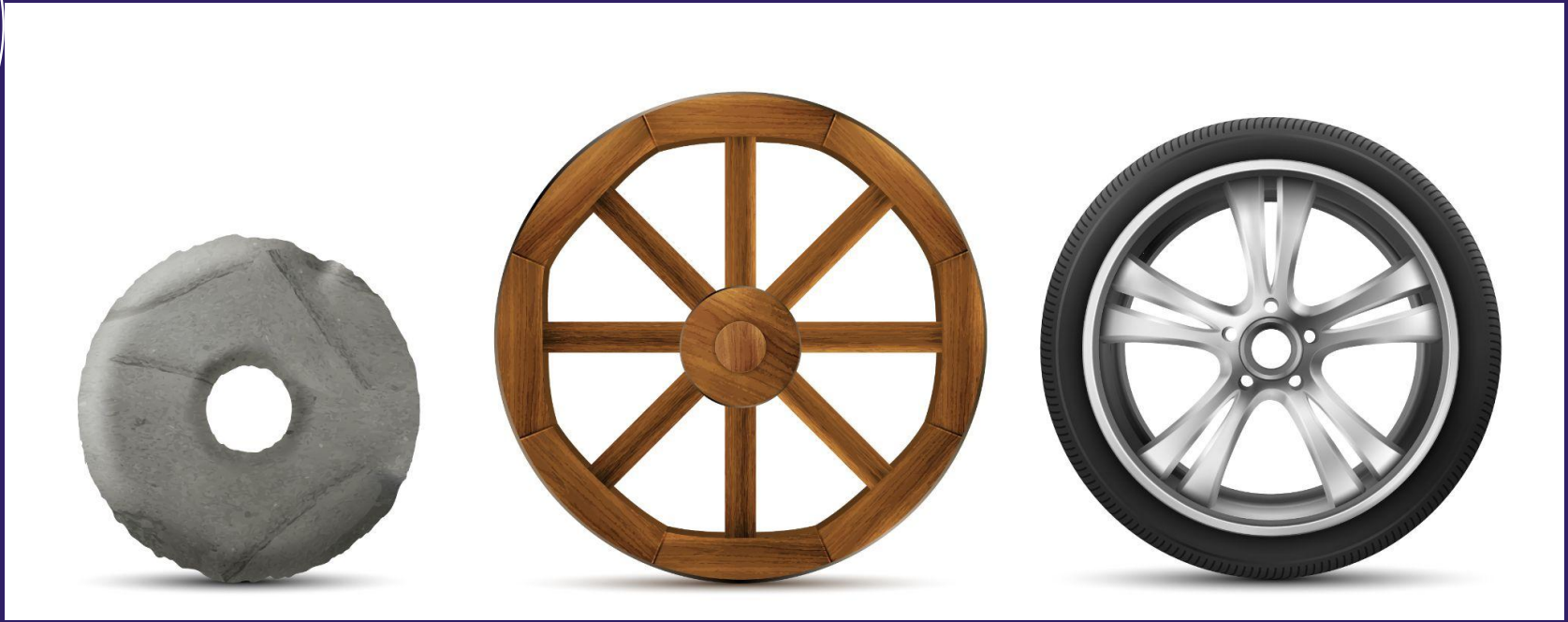
- Break ecosystem with each new feature
- Change outer container from zip archive
- Reduce human readability of filename
- Require custom parsing of file contents

Credit: Graeme Tozer on Flickr

PEP 777: Reinventing the Wheel

- Wheel Specification (PEP 427) is over a decade old
- Wheel usage has changed significantly
  - e.g. much more common to ship libraries in wheels
- Wheel format *needs* to change:
  - METADATA difficult to parse correctly
  - Wheel versioning makes it difficult to adopt features individually
  - No ability to support alternative compression formats
  - Wheel metadata in the filename inhibits flexibility
- How can we change the wheel format without “breaking the world” for each feature?

Evolve the wheel format



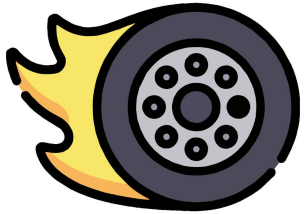
Current issues with wheel evolution

{distribution}-{version}(-{build tag})?-{python tag}-{abi tag}-{platform tag}.whl

- Every change bumps the wheel major version - which breaks installation of those wheels on older installers
- No clear resolution for users - will upgrading help?
- Wheel version unavailable to resolvers until downloaded
- Wheel filename rigid and impossible to extend due to optional tag in the middle of the filename

How to evolve wheels

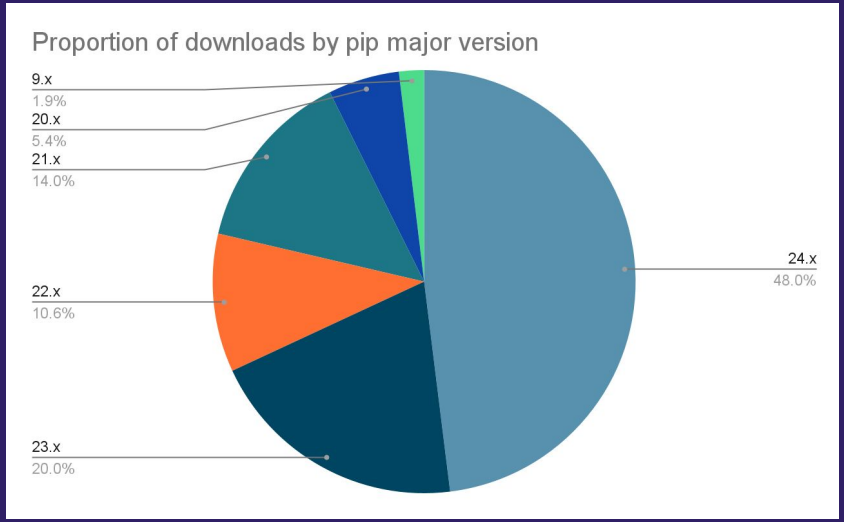
- Wheel ***features*** enable individual adoption and integration of improvements
- Increase wheel version visibility to resolvers by serving WHEEL
- Try to break users *only once*
  - Change the wheel filename/major version with wheel 2.0, then
  - Enforce installers to *ignore incompatible wheel files*
- Make wheel metadata extensible
  - Move “source of truth” from file name to WHEEL metadata file
  - Serve WHEEL just like METADATA
  - Change filename to include hash of WHEEL, keeping {name}-{version} prefix



PEP 777: Reinventing the wheel

Open questions

- Is it better to break users every change if they can tell new wheels are available?
- Should we delay publication to reduce number of users broken if we don't ignore incompatible wheels by default?
- How can we emphasize disruptions if wheel updates do "break the world"?
- How best to signal new wheels that are incompatible exist to users?



PEP 778: Symlinks

- Background
  - Libraries on Linux have a particular naming scheme:
    libfoo.so
    libfoo.so.2
    libfoo.so.2.3.1
  - Only libfoo.so.2.3.1 is a file - the other two are symbolic (soft) links to that file
- A number of projects now distribute shared libraries in their wheels for users to link against at runtime and build time
  - Apache Arrow
  - CUDA
  - PyTorch
- The zip format, and thus wheels, do not support symlinks

PEP 778: Symlinks

- An example of wheel format evolution
- **LINKS** file describes symlinks to be created by installer
- Narrowing focus to only Unix and libraries
  - Symlinks *could* be used to support editable installs (PEP 660), but that is left to another PEP
  - Portability of symlinks is complicated
    - On Windows, symlinks require developer mode or Administrator permissions
    - Windows has hard links, but they are different
- Security model very important
  - Shouldn't be able to symlink outside of site directory
  - Shouldn't be able to symlink into another package's contents, unless under a shared namespace
- Should probably leave door open to other link types?

Thank you for your attention