Scikit-build

A Build System Generator for CPython C/C++/Fortran/Cython Extensions

Speaker: Jean-Christophe Fillion-Robin
Scipy 2018

Who am I ?

- Principal Engineer @ Kitware in our North Carolina office, lead developer of 3D Slicer
- Maintainer of scikit-build, cmake and ninja python packages
- Maintainer of python-cmake-buildsystem
- Maintainer of dockcross

jfillionr jcfr thehiddenbrain **#scikit-build**

Goal of this talk

- Identify requirements for building CPython Binary Extensions
- Outline limitations of current approach
- Introduce our solution

Outline

- Python Package Lifecycle
- C/C++/Fortran/Cython Binary Extensions: Problems ? Impact ?
- State of the current building tools
- Requirements
- Our solution: scikit-build
- Features
- What is next ?

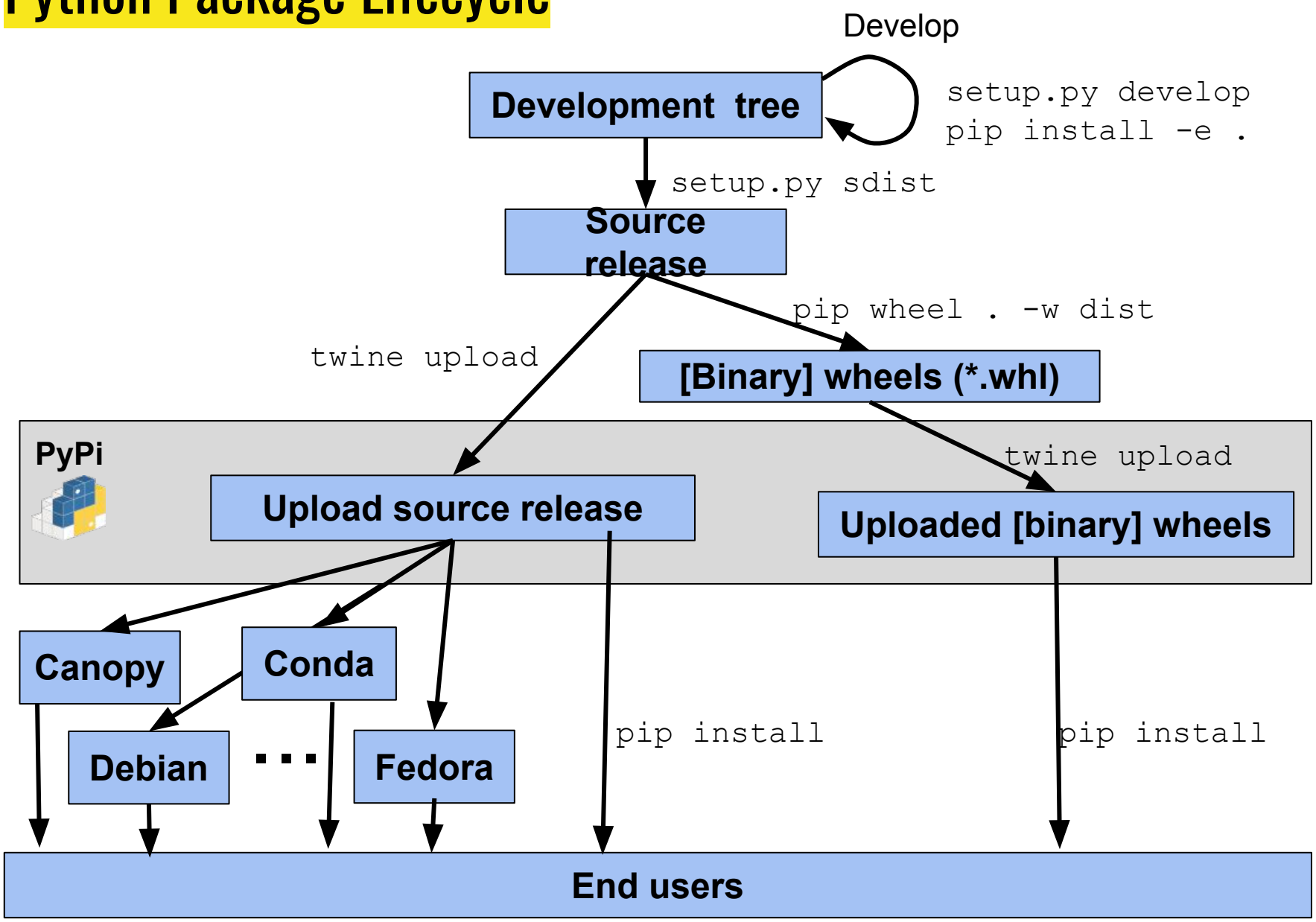
The Python Packaging User Guide: Must read

A collection of tutorials and references to help you distribute and install Python packages with modern tools.

See https://packaging.python.org/

Credits: Nathaniel J. Smith, SciPy 2016, Reinventing the whl

Python Package Lifecycle



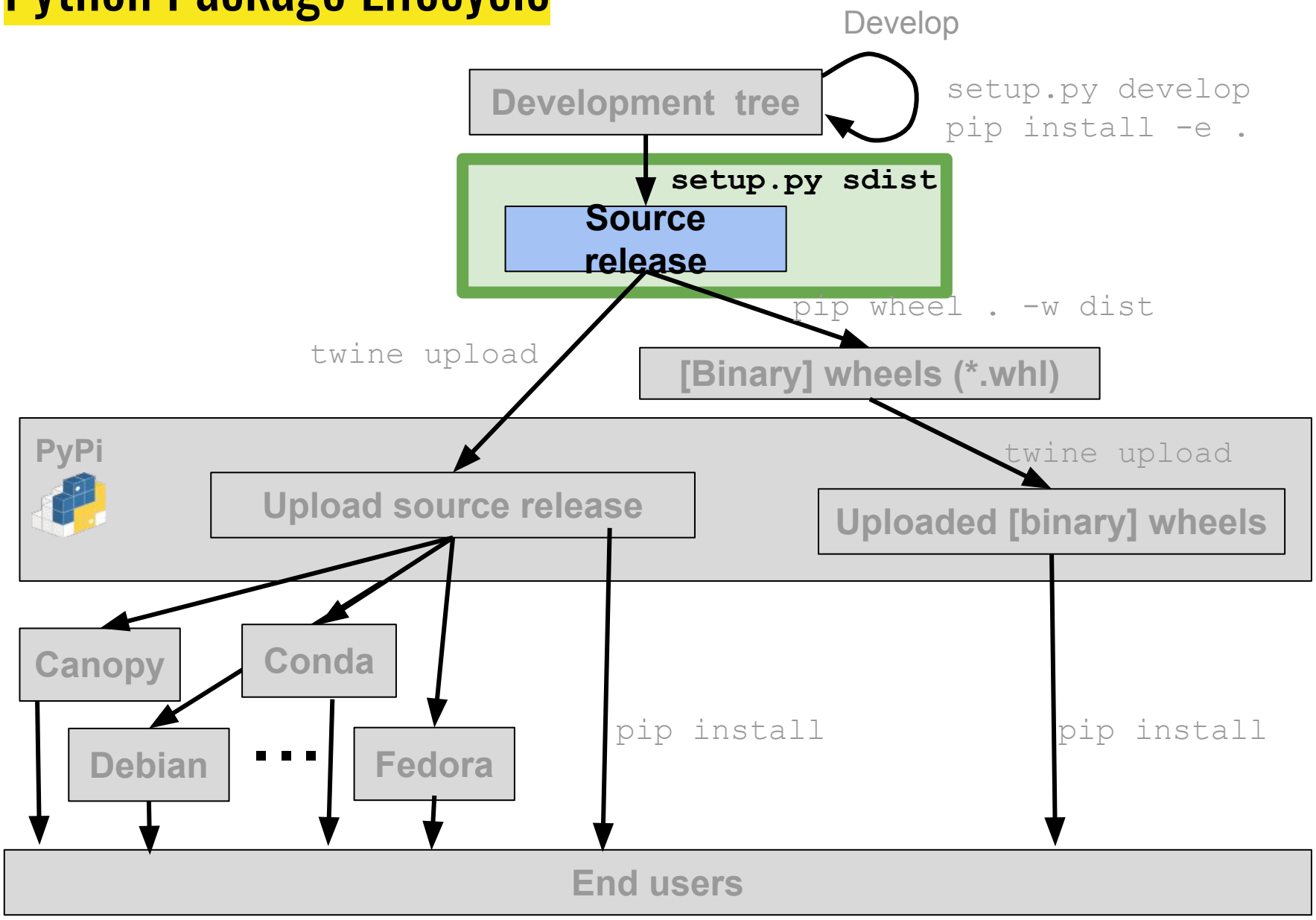
Packaging Terminology 101

- **Source distribution**
  - Synonyms: **sdist**, source release
  - provides metadata + source files
  - needed for installing
    - by a tool like pip
    - or for generating a **Built Distribution**

Source: https://packaging.python.org/glossary

Credits: Nathaniel J. Smith, SciPy 2016, Reinventing the whl

Python Package Lifecycle



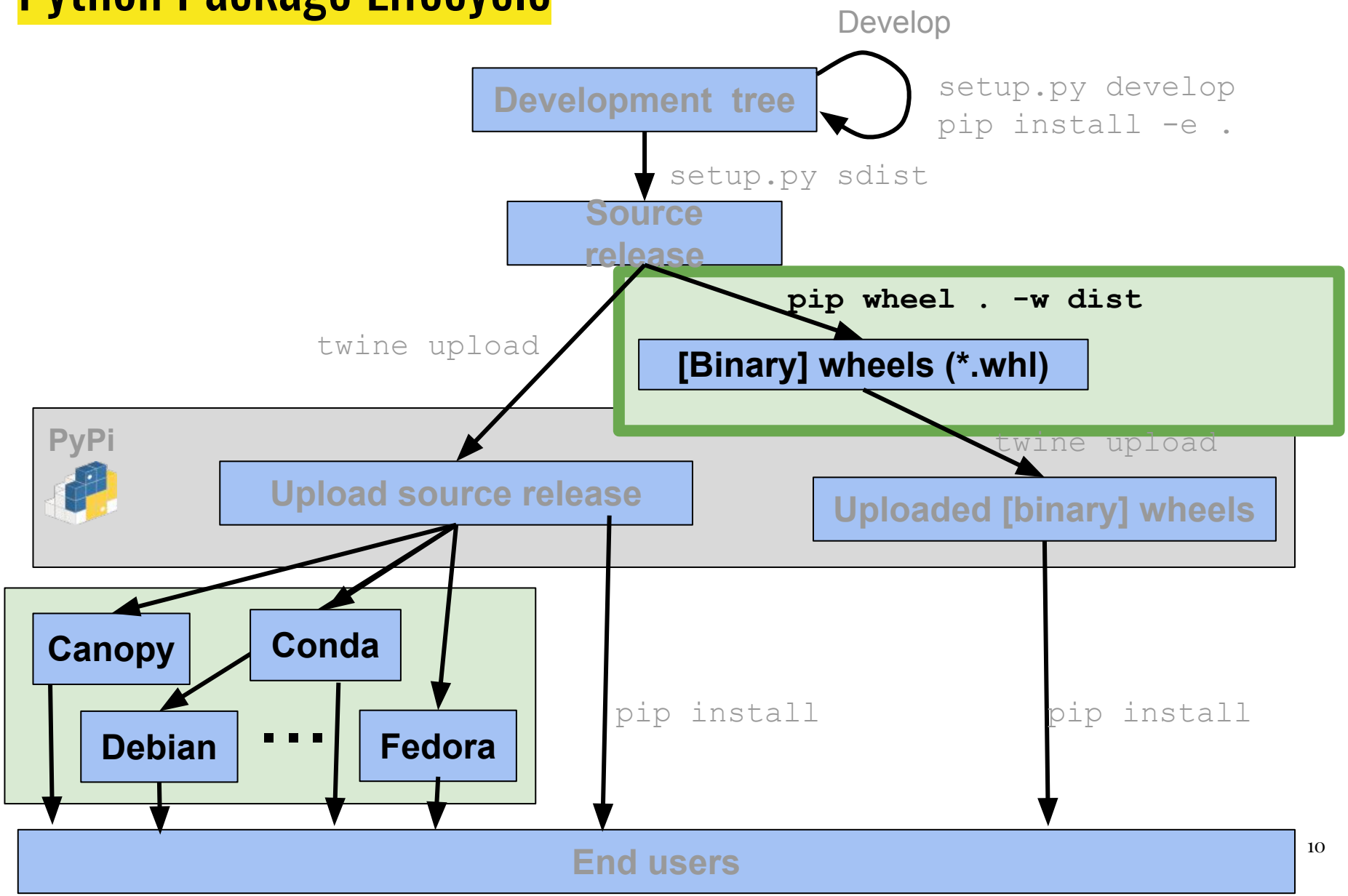
Packaging Terminology 101

- **Built Distribution**
  - Synonyms: **bdist**, wheel
  - provides metadata + pre-built files
  - only need to be moved to the correct locations on the target system

Source: https://packaging.python.org/glossary

Credits: Nathaniel J. Smith, SciPy 2016, Reinventing the whl

Python Package Lifecycle



Packaging Terminology 101

- Application binary interface (ABI)
  - Interface between two binary program modules
  - Adhering to an ABI is usually the job of compiler

Source: https://en.wikipedia.org/wiki/Application_binary_interface

Packaging Terminology 101

- A python Distribution is either:
  - **pure**:
    - Not specific to a CPU architecture
    - No ABI
  - **non-pure**
    - ABI
    - Platform specific

Source: https://packaging.python.org/glossary

Packaging Terminology 101

- **Binary Distribution**
  - is a **Built Distribution**
  - Is **non-pure**
  - uses platform-specific compiled extensions

- **Non-pure**
  - ABI
  - Platform specific

- **Built Distribution**
  - Synonyms: **bdist**, wheel
  - provides metadata + pre-built files
  - only need to be moved to the correct locations on the target system

Source: https://packaging.python.org/glossary

Packaging Terminology 101

- **Wheel**:
  - a **Built Distribution**
  - a ZIP-format archive with `.whl` extension
    - `{distribution}-{version}(-{build tag})?-{python tag}-{abi tag}-{platform tag}.whl`
  - described by PEP 427

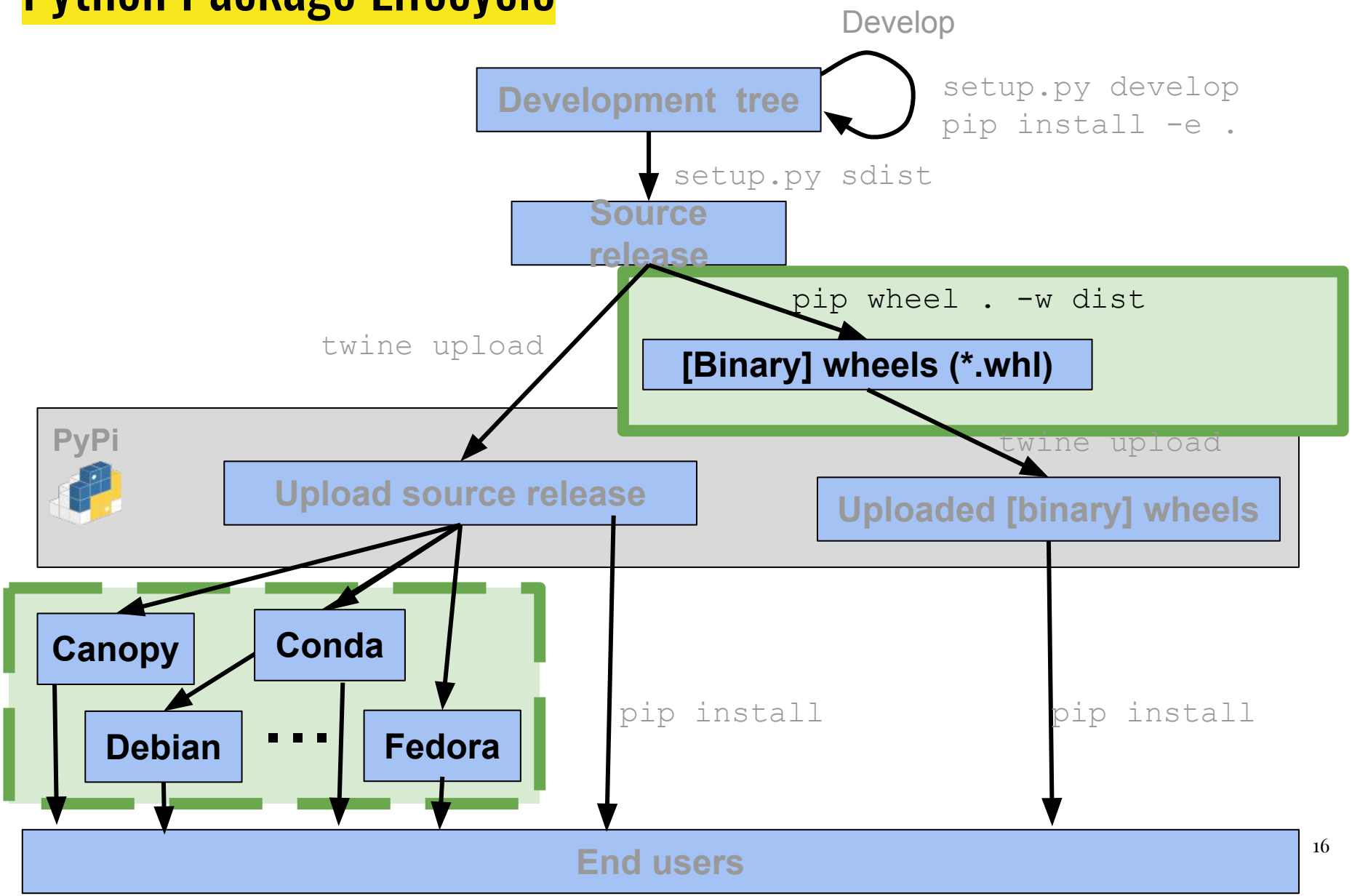
Source: https://packaging.python.org/glossary

Packaging Terminology 101

- Few examples of **wheel** archives:
  - **Non-pure** wheels
    - cffi-1.11.5-cp27-cp27m-manylinux1_x86_64.whl
    - cffi-1.11.5-cp27-cp27m-macosx_10_6_intel.whl
    - cffi-1.11.5-cp36-cp36m-win_amd64.whl
  - **Pure** wheels
    - docutils-0.14-py2-none-any.whl
    - docutils-0.14-py3-none-any.whl
    - certifi-2018.4.16-py2.py3-none-any.whl

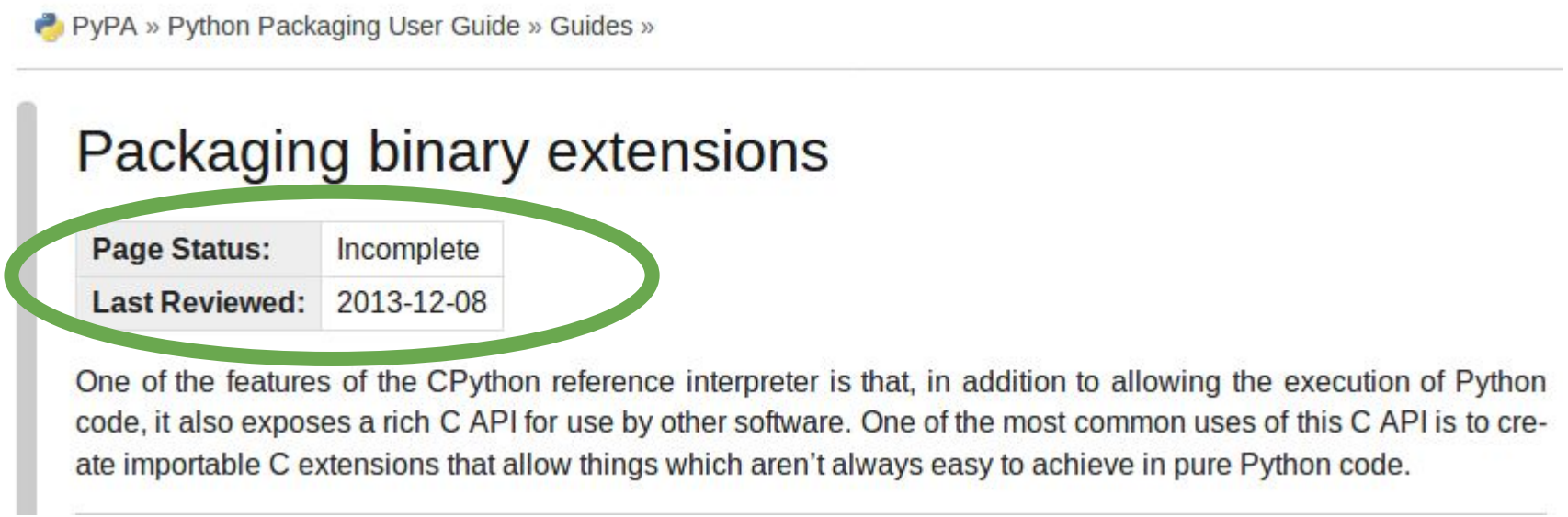
Credits: Nathaniel J. Smith, SciPy 2016, Reinventing the whl

Python Package Lifecycle

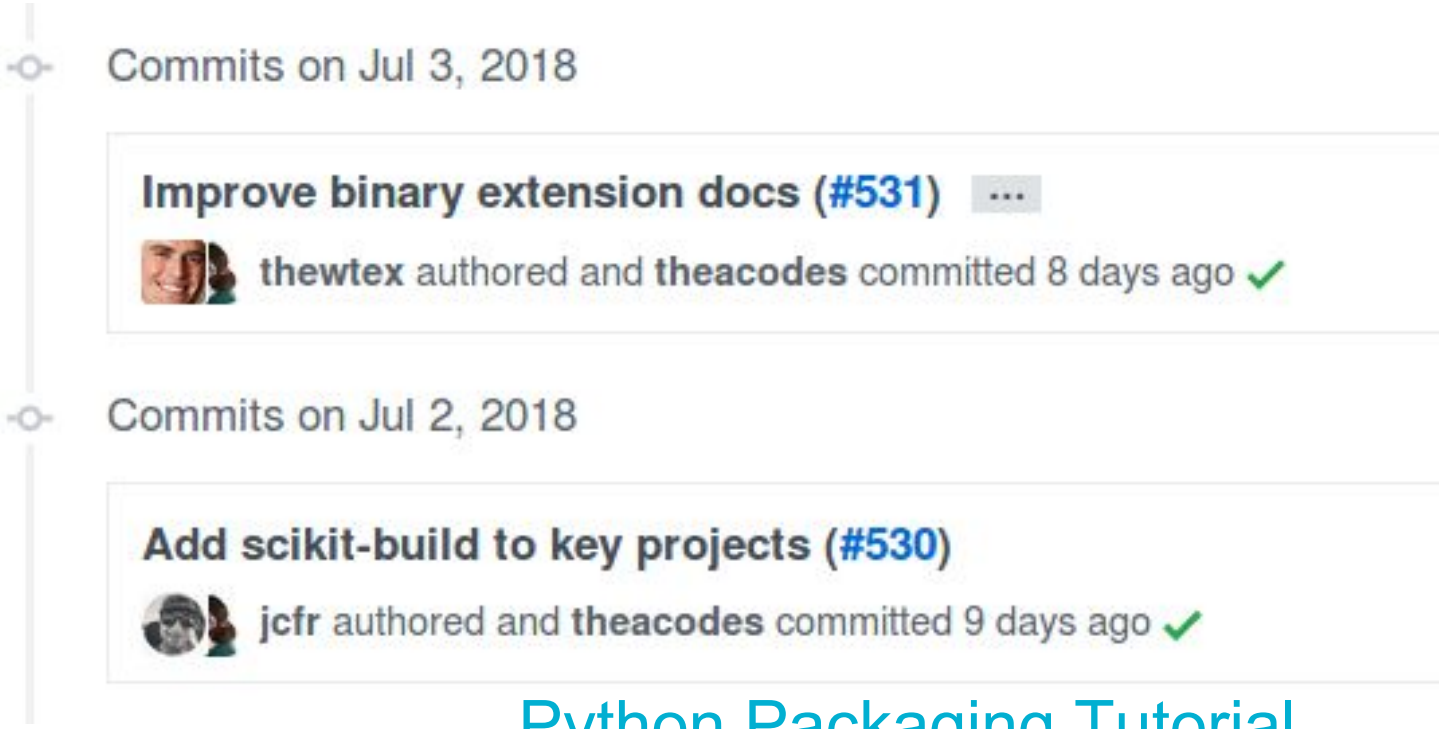


The Python Packaging User Guide: Caveat

"Packaging of binary extensions" needs some TLC



The Python Packaging User Guide: Good news



Python Packaging Tutorial

Thanks @theacodes for reviewing and integrating !

The SciPy Python Packaging Tutorial

The Sheer Joy of Packaging!

Scipy 2018 Tutorial

Packaging

Packaging from start to finish for both PyPI and conda

Topics

- Tutorial Schedule
- Overview
- Making a Python Package
- Building and Uploading to PyPi
- Binaries and Dependencies
- Conda Packages

https://python-packaging-tutorial.github.io/python-packaging-tutorial/

C/C++/Fortran/Cython Binary Extensions

Question:

Do they really matter for science ?

C/C++/Fortran/Cython Binary Extensions

Question:

Do they really matter for science ?

Answer:

Yes, they allow to:

- speed up computationally intensive code
- create wrapper module
- access lower level features of OS, hardware, CPython, ...
- leverage multi-core, many-core, GPU, ...

C/C++/Fortran/Cython Binary Extensions

Question:

Do they really matter for science ?

Answer:

Yes, they allow to:

- ...
- leverage existing scientific code
- careful and precise memory management
- avoid parallel computing issues related to CPython's Global Interpreter Lock (GIL)

C/C++/Fortran/Cython Binary Extensions

Question:

Sure, but which packages include them ?

C/C++/Fortran/Cython Binary Extensions

Question:

Sure, but which packages include them ?

Answer:

Just to name a few … cffi, cython, lxml, matplotlib, msgpack-python, numpy, pandas, pillow, pyaml, pyne, pyzmq, scikit-learn, **scipy**, symengine, twisted, mayavi

What are the problems ?

- Specifying the correct compiler and link flags is error-prone
- Finding build-time dependencies is difficult

Source: https://gatsiesheikar.files.wordpress.com/2013/03/headdesk.png

What are the problems ?

- Leveraging build tool like make or ninja is not possible
- Building multiple extensions in parallel is not an option without some crazy hack
- Using an IDE like Visual Studio, QtCreator, ... to develop the C or C++ code is not possible

Source: https://gatsiesheikar.files.wordpress.com/2013/03/headdesk.png

What are the problems ?

- Setting up a project with compiled modules is tedious
- “Cross-platform development and distribution of extension modules is a **complex topic** [...]”
  Source: Python Packaging User Guide / Packaging binary extensions
- Cross-compiling is horrendous

Source: https://gatsiesheikar.files.wordpress.com/2013/03/headdesk.png

Who does it impact ?

- You writing code
- You maintaining a project
- You installing a “package”

There is now “scikit-build”

A **reliable** and **easy** way to package your **compiled** Python projects.

State of the current tools

- handle common needs
- but leave a lot to be desired
- not specialized for compiling

Requirements (1 / 3)

- reliably building native codes
- first class support for system introspection
- creating extension modules for Python scripts

Requirements (2 / 3)

- dynamically and statically linking extension modules for use in embedded applications
- compiling, packaging and publishing
- all integrated with the Python ecosystem

Requirements (3 / 3)

- first-class cross platform support

- cross-compilation capabilities

Observation

There is a clear need for a **cross-platform** building and packaging solution that **supports** projects using **compiled** modules and their **users**.

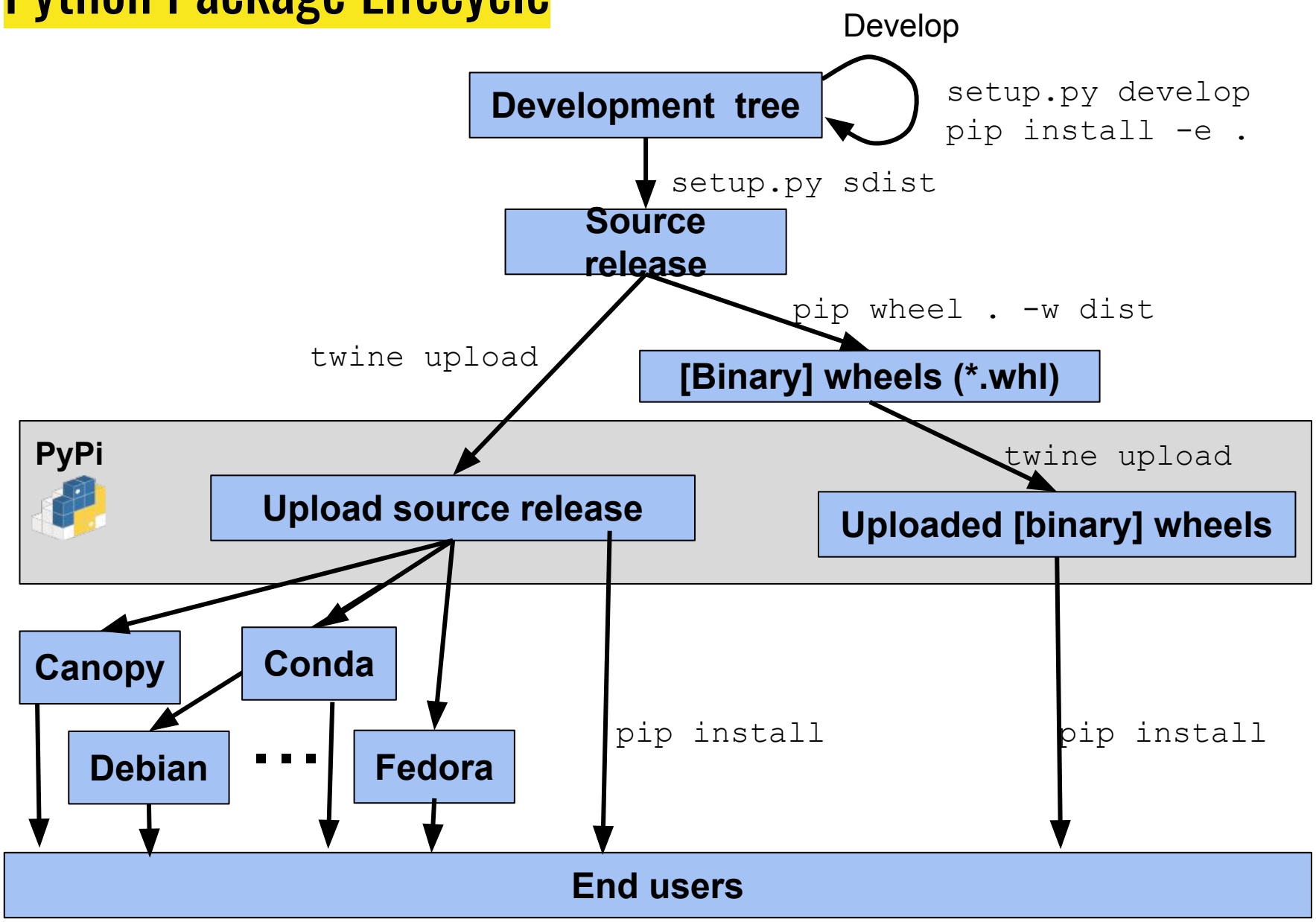
Our solution: scikit-build

Packaging solution that **bridges the gap** between

- cmake,
- **your** C/C++/Fortran and Cython **projects**,
- and the **Python ecosystem** (pip and setuptools)

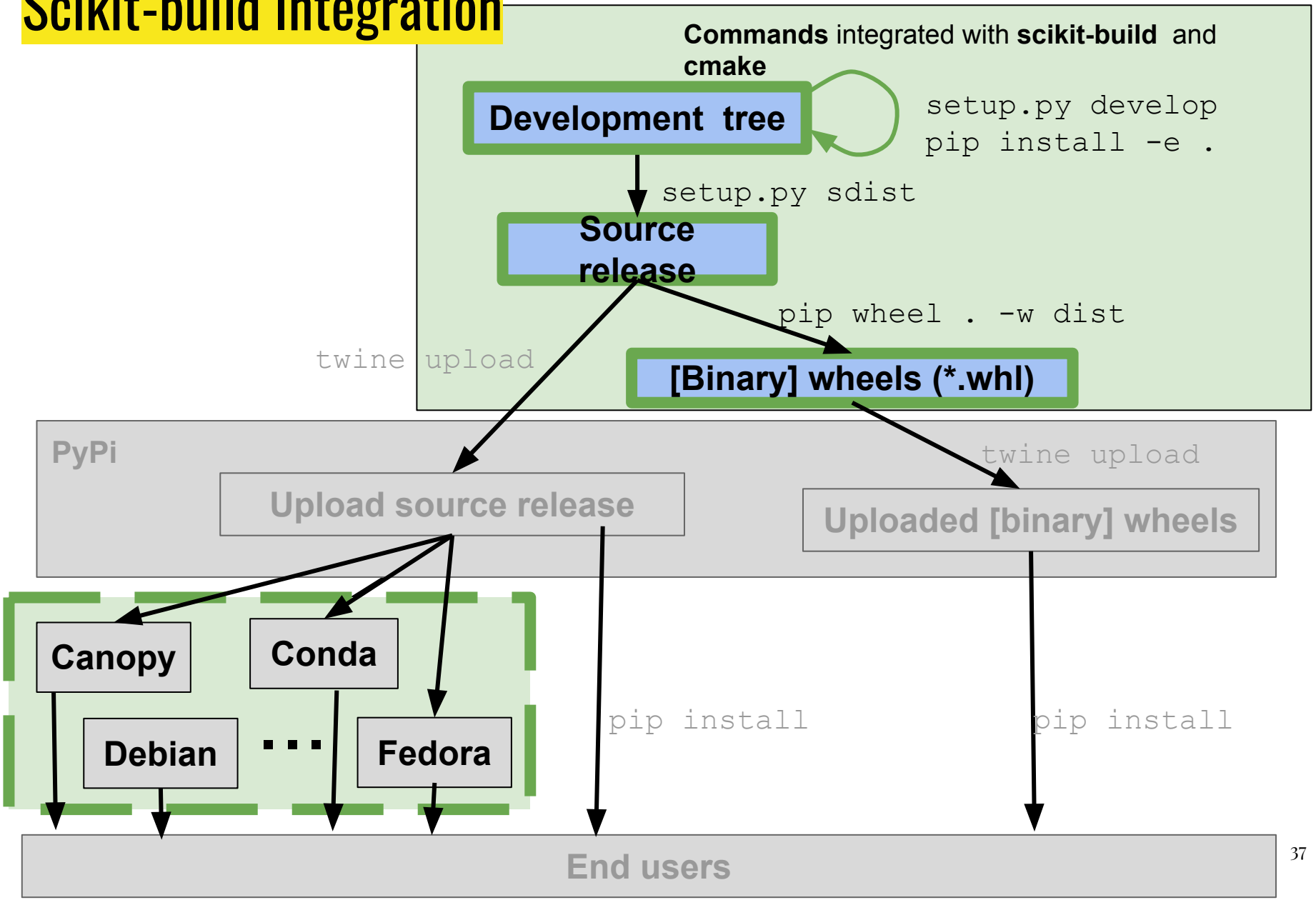
Credits: Nathaniel J. Smith, SciPy 2016, Reinventing the whl

Python Package Lifecycle



Credits: Nathaniel J. Smith, SciPy 2016, Reinventing the whl

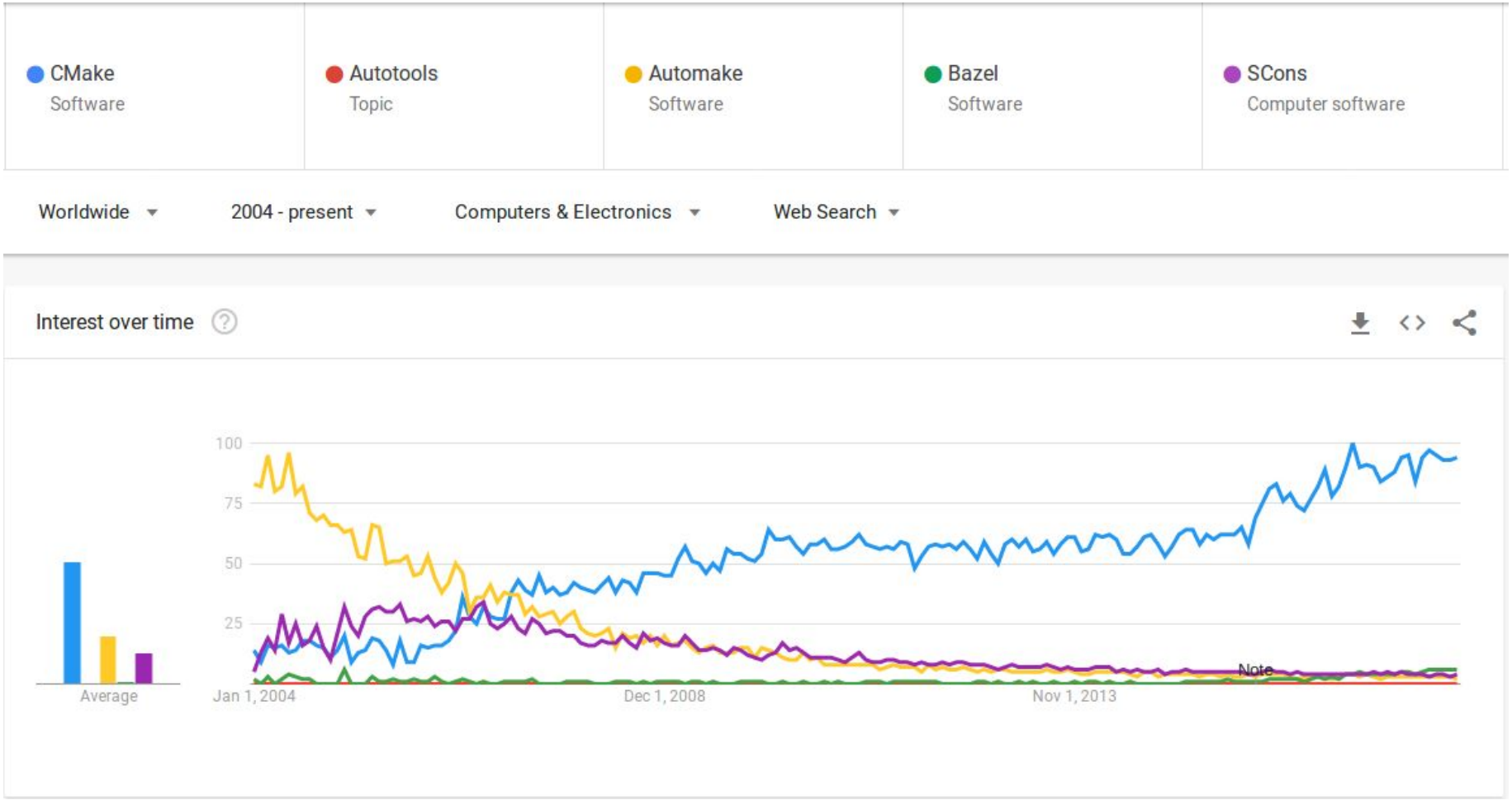
Scikit-build Integration



Benefits of using CMake

- open-source, well-maintained and supported
- cross-platform builds with support for cross-compilation
- system introspection capabilities
- support for native build systems
- python extension module support

Build Tool Trends



Source:
https://trends.google.com/trends/explore?cat=5&date=all&q=%2Fm%2F0cxh7f,%2Fg%2F1thtpq_p,%2Fm%2F01fg0l,%2Fg%2F11bzyq50jp,%2Fm%2F04dl04

scikit-build is ...

- **not** (yet) reinventing the .whl
- **not** a new paradigm:
  - It integrates with ***existing*** tools
- **not** using `setuptools.Extension` internally but can be used along side

scikit-build is ...

- **not** a replacement for canopy, conda or pypi
  - does **not** manage packages
  - only makes building them easier

- **not** magic
  - you're still compiling extension modules
  - only with much less guess-work

scikit-build is ...

- a drop-in replacement for setuptools

```
--- a/setup.py
+++ b/setup.py
-from setuptools import setup
+from skbuild import setup
```

scikit-build only needs ...

- a `CMakeLists.txt` describing your extension

Hello Example: Layout

```
$ cd hello && find .
.
./CMakeLists.txt
./hello
./hello/_hello.cxx
./hello/__init__.py
./setup.py
./pyproject.toml
```

Hello Example: setup.py

```
from skbuild import setup
setup(
    name="hello",
    version="1.2.3",
    packages=['hello'],
  # description, author, license, …
)
```

Hello Example: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.12)
project(hello)
find_package(PythonExtensions REQUIRED)

add_library(_hello MODULE hello/_hello.cxx)
python_extension_module(_hello)
install(TARGETS _hello LIBRARY DESTINATION hello)
```

Hello Example: pyproject.toml

```
[build-system]
requires = ["setuptools", "wheel", "scikit-build",
"cmake", "ninja"]
```

Hello Example: Building the wheel

```
$ cd hello
$ pip wheel . -w dist
```

Scikit-build Features (1 / 3)

- Support for Python 2.7, 3.4 and above

- Compiling environment
  - By default, lookup compilers matching python version
  - Provide recommendations if compilers not installed
  - Understand CC, CXX, CFLAGS and CXXFLAGS env. variables

- Support ninja build tool on Linux, macOS and Windows
  - No need to toy around with vcvarsall.bat et al

Scikit-build Features (2 / 3)

- Support for developer mode: `pip install -e .`
- cmake and ninja pip installable on all platforms
- Support generation of Source Distribution for git repository (only if `MANIFEST[.in]` is not found)
- Fast incremental rebuild by skipping re-configuration if relevant.

Scikit-build Features (2 / 3)

- Support for **setup usual keywords**: packages, package_dir, include_package_data, package_data, exclude_package_data, py_modules, data_files, scripts

- Support for **additional setup keywords** **to configure CMake**: cmake_args, cmake_install_dir, cmake_source_dir, cmake_with_sdist, cmake_languages

scikit-build: Command line options

- Useful for passing options
  - to CMake
  - to build tool (make, ninja, msbuild, ...)

```
$ python setup.py \

  bdist_wheel -- <cmake_options> -- <buildtool_options>
```

Scikit-build also provides ...

PythonExtensions CMake module to build python module or standalone python executable.

Cython CMake module to generate source code.

NumPy CMake module to lookup numpy/arrayobject.h, conv-template and from-template.

F2PY CMake module to find f2py executable facilitating creating/building of Python extension calling Fortran 77/90/95 external subroutines.

Scikit-build: Software Process, CI and Documentation

- ~200 tests running on 3 platforms for python 2.7, 3.4 to 3.7
- https://scikit-build.readthedocs.io
- CONTRIBUTING, Release Notes, MIT license, mailing list

Build Status

	Linux	MacOSX	Windows
PyPI	circleci passing	build passing	build passing
Conda	circleci passing	build passing	build passing

Overall Health

requirements up-to-date | codecov 90% | health 99%

What is next ?

- Scipy2018 sprints (Saturday):
  - finalize conda package associated with latest release 0.7.1
  - improve documentation and add more example projects
  - provide guidance to integrate scikit-build

- Long term
  - work with numpy and scipy folks
  - help with proposal for packaging native libraries into Python wheels
  - implement as PEP 517 build backend interface, configured with pyproject.toml and CMakeLists.txt (issue #124)
  - leverage distlib (library of packaging functionality)

Thank you !

- scikit-build contributors:

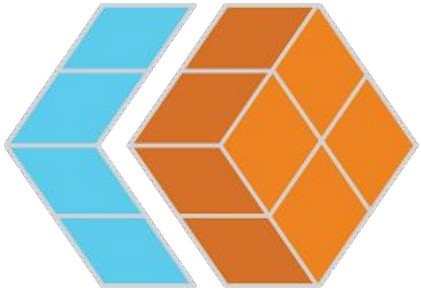
Table generated using mgechev/github-contributors-list npm package

				
willingc	seanlis	neok-m4700	henryiii	blowekamp
				
scopatz	msmolens	thewtex	msarahan	opadron

and also xoviat

- Scikit-build issue reporters: benjaminjack, isuruf, henryborchers, jonwoodring, mivade, reiver-dev, seanlis

- PyPA, PEPs contributors, the wider community



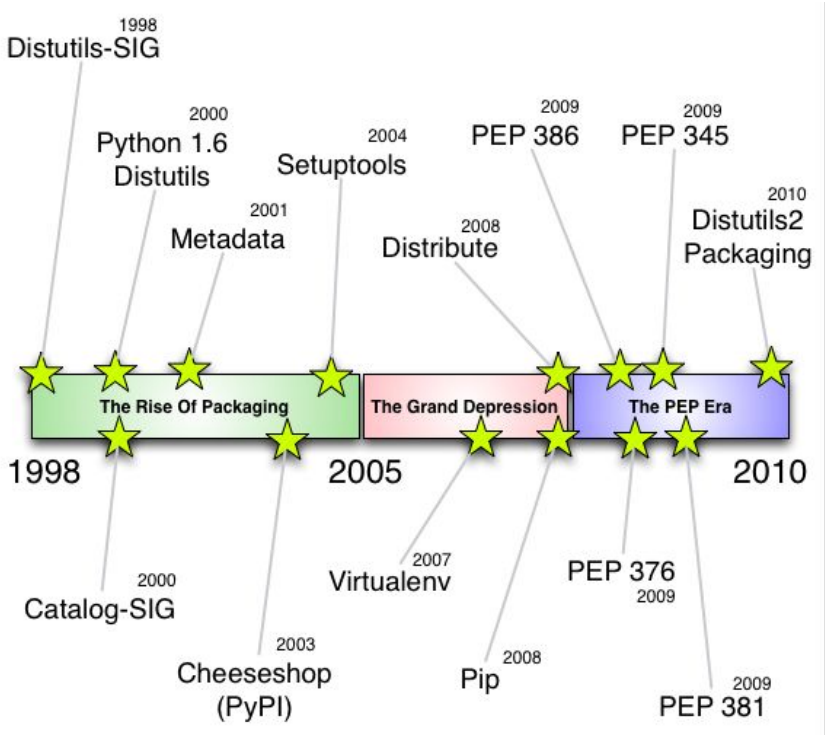
scikit-build

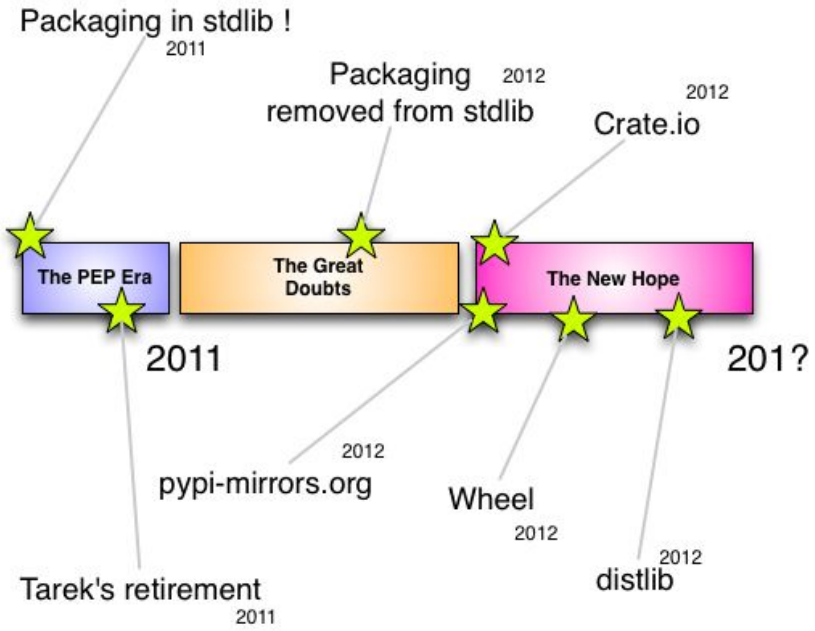
Slides: bit.ly/scikit-build-talk

Source code: github.com/scikit-build/scikit-build

jfillionr jcfr thehiddenbrain **#scikit-build**

Python Packaging History: A long standing community effort





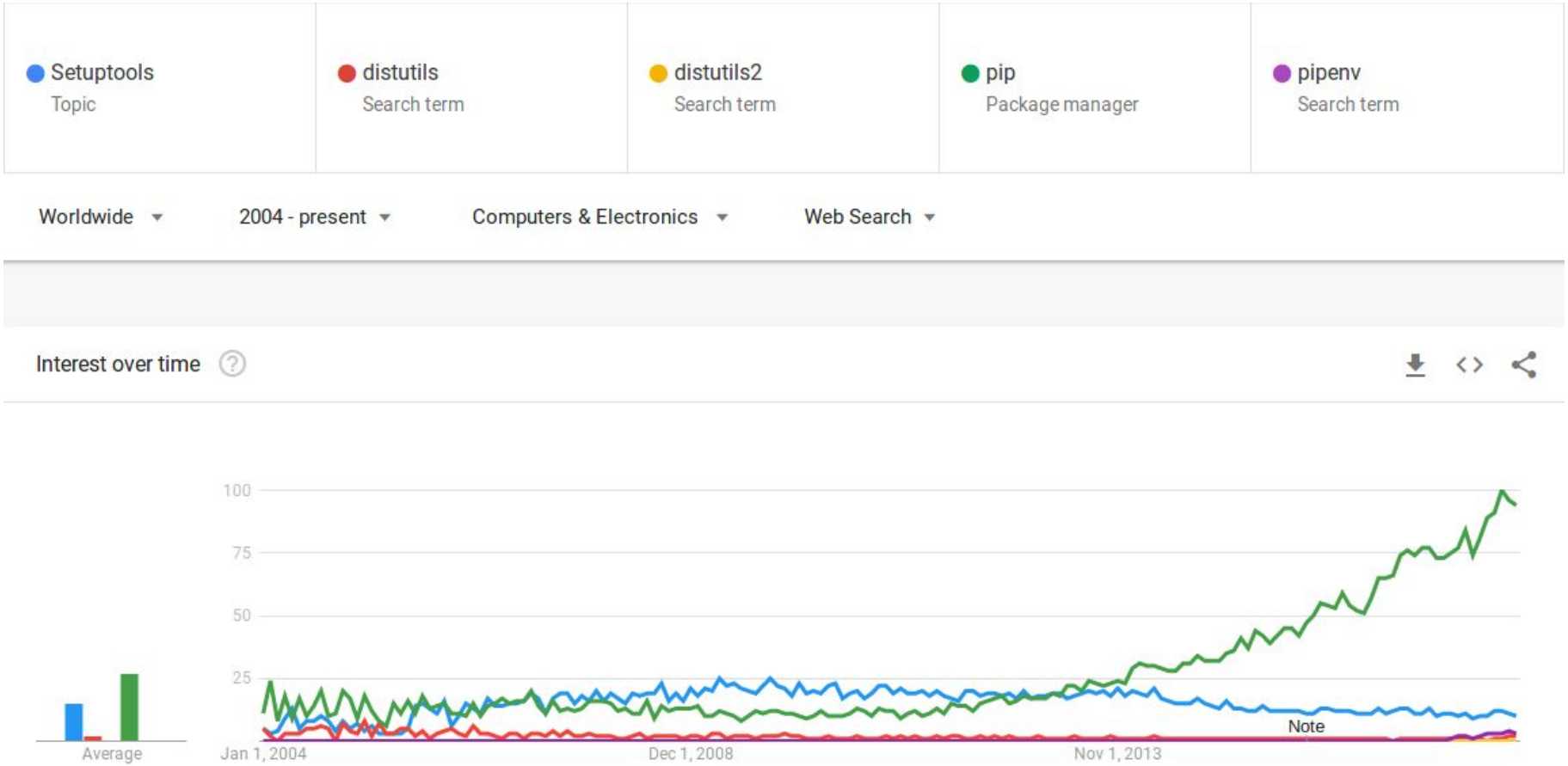
Source: https://ziade.org/2012/11/17/chronology-of-packaging/

Followed by outstanding standardization efforts

- 2013:
  - PEP 425 -- Compatibility Tags for Built Distributions
  - PEP 427 -- The Wheel Binary Package Format 1.0
- 2014:
  - PEP 440 -- Version Identification and Dependency Specification
- 2016:
  - PEP 513 -- A Platform Tag for Portable Linux Built Distributions
  - PEP 518 -- Specifying Minimum Build System Requirements for Python Projects
- 2017:
  - PEP 517 -- A build-system independent format for source trees

and many more at https://www.python.org/dev/peps/

Python Packaging History: The trend



Source: https://trends.google.com/trends/explore?cat=5&date=all&q=%2Fm%2F07kg5hp,distutils,distutils2,%2Fm%2F0hzm844,pipenv

- Pip: $778k, 14 person-years, The python package manager
- Setuptools: $48k, 1 person-year, enhancements to the Python distutils
- Scikit-build: $65k, 1 person-year ↠ Our tool, simple and to the point. It creates a bridge.
- Python: $15M, 284 person-years ↠ CPython interpreter
- CMake: $19M, 346 person-years ↠ Specialized in compiling and building