Python Integration with C and C++

Calling C and C++ code from Python

using

Cython, SWIG, and CFFI

Example Code on GitHub at:
https://github.com/tleonhardt/Python_Interface_Cpp

Todd Leonhardt

Motivation

- Leverage the strengths of C/C++ to alleviate the weaknesses of Python (and vice versa)
- Python
  - Strengths
    - Let’s you write code quickly
    - Good at integrating systems and testing
  - Weaknesses
    - Pure Python can run much slower than C/C++
- C/C++
  - Strengths
    - Very fast execution speed
  - Weaknesses
    - Slower development speed due to manual memory management, static types, worse library, etc.
    - Slow compile time

- Typical use cases calling C/C++ from Python
  - Optimize pre-existing Python to make critical parts run faster (after profiling, of course)
  - Leverage existing C/C++ in Python
    - For Python’s rapid development
    - For Python’s ease of creating a UI
    - For Python’s testing capabilities

What will be covered

- Wrapping an existing C/C++ shared library and making calls into it from Python code
  - Cython
  - SWIG
  - CFFI
- Optimize existing Python code by converting a small amount of critical code to C/C++
  - Cython

What won’t be covered

- Numerous ways of interfacing Python with C/C++
  - In particular ways which don’t involve wrapping or creating a shared library
- Using Python subprocess to call C/C++ executable
  - And communicate via stdin/stdout
- Using an InterProcess Communication (IPC) mechanism to communicate
  - Such as sockets or 0MQ
- Embedding a Python interpreter within a C/C++ executable
- Using weak asynchronous file-based data exchange
  - e.g. serialize data from Python to JSON and then use that file in C/C++
- Also the basic Python C-API won’t be covered
  - And how to create extension modules from scratch (powerful, but lots of work)

CFFI - C Foreign Function Interface for Python

- What is CFFI?
  - Offers options very similar to built-in ctypes, but it's a bit more direct
  - Lets you interact with almost any C code from Python
    - Based on C-like declarations that you can often copy-paste from header files or documentation
- What is it good for?
  - It makes it very easy to interface with C code from Python
    - Either from a shared library or C code written "inline" in Python
- What else can it do?
  - Embedded Python code in a C shared library
    - Which is then callable from other C code
  - Fully compatible with PyPy JIT
- Limitations
  - No C++ support
  - Performance noticeably worse than either SWIG or Cython
- Advantages
  - Self-contained. All code is inline within Python. No need to create any external files
    - Assuming you already have a dynamic C library you want to call code from
  - Quickest way to write Python code that calls functions in an existing C shared library

What does CFFI do?

- Quickest way to wrap an existing external C shared library
- Call C libs from Python directly
- No interface code
  - Specify function prototypes inline in your Python code
- How?
  - Python is implemented in C
    - Make calls directly into C libraries at the binary level using the C ABI
- 4 step process:

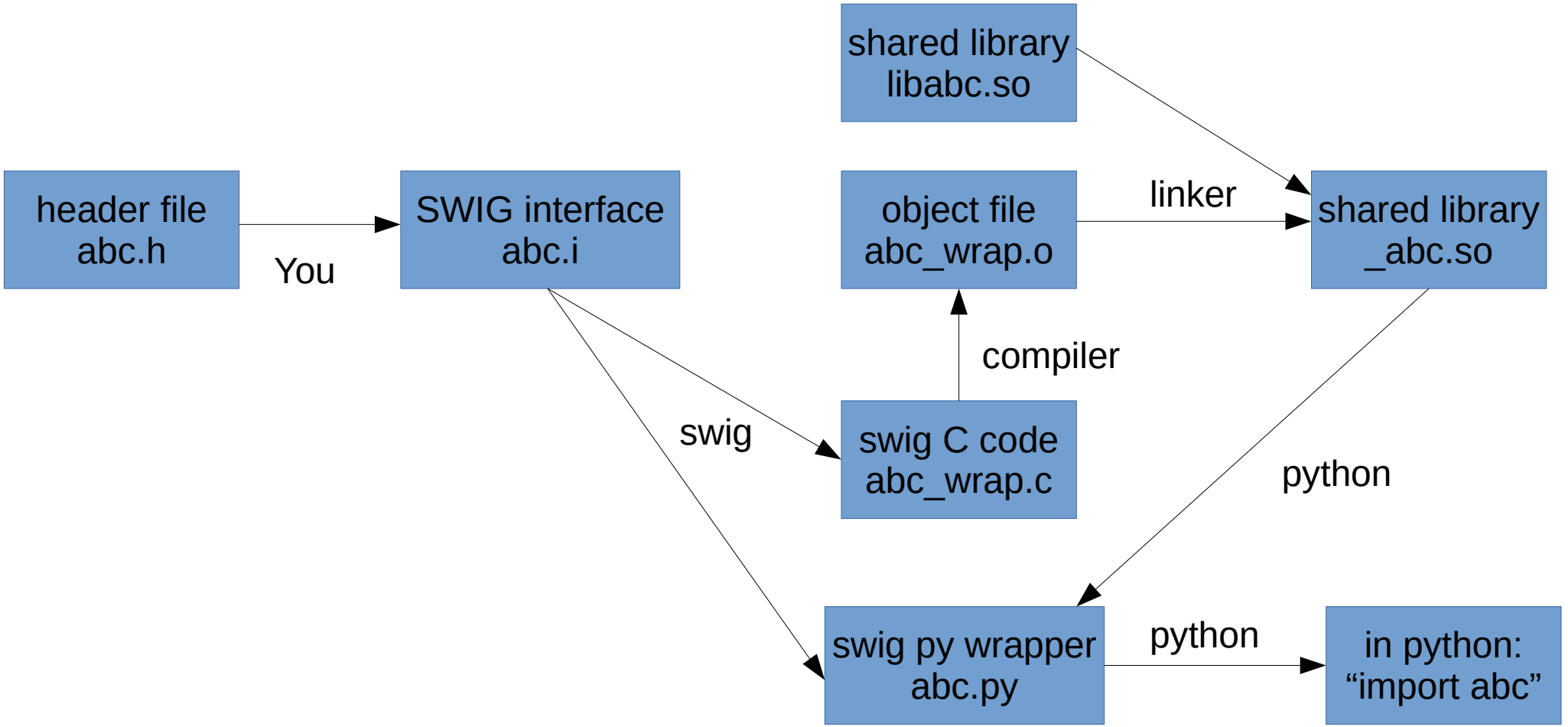
  1) Instantiate the main top-level CFFI class: **ffi = cffi.FFI()**
  2) Declare function prototype: **ffi.cdef('int compute_fibonacci(int** n);')
  3) Load the dynamic library: **libfib = ffi.dlopen('./libfibonacci.so')**
  4) Call a function in the lib: **fib_20 = libfib.compute_fibonacci(20)**

SWIG - Simplified Wrapper and Interface Generator

- What is SWIG?
  - Tool that auto-generates wrapper code for C/C++ code/libraries
    - Can target about 20 different programing languages
      - Python, Java, C#, PHP, Javascript, Perl, Ruby, Go, R, D, Lua, etc.
    - Not fully automatic - you need to create the *.i interface files to tell it what to do
      - But it is relatively easy. You only need to wrap the things you need.
- What is it good for?
  - Wrapping existing C/C++ libraries/code
    - So they can be used from a higher-level language
      - Why? User interfaces, testing, optimization, etc.
- What else can it do?
  - Wrapping, that's it. But it can do it for 20+ target languages and it's relatively simple to use.
- Limitations
  - Getting template-heavy C++ code properly wrapped can be hard work
  - Performance not as good as Cython (but better than CFFI)
  - It's "magic"- in the rare cases when it doesn't work, debugging magic can be quite challenging
- Advantages
  - Easy and quick to use if you already have existing C/C++ code
  - Targets numerous different languages
  - Good C++ support

What does SWIG do?

- Partially automates the process of wrapping existing C/C++ code
- Easier to use to wrap code than Cython, but better performance
- But a higher performance way to wrap than using CFFI

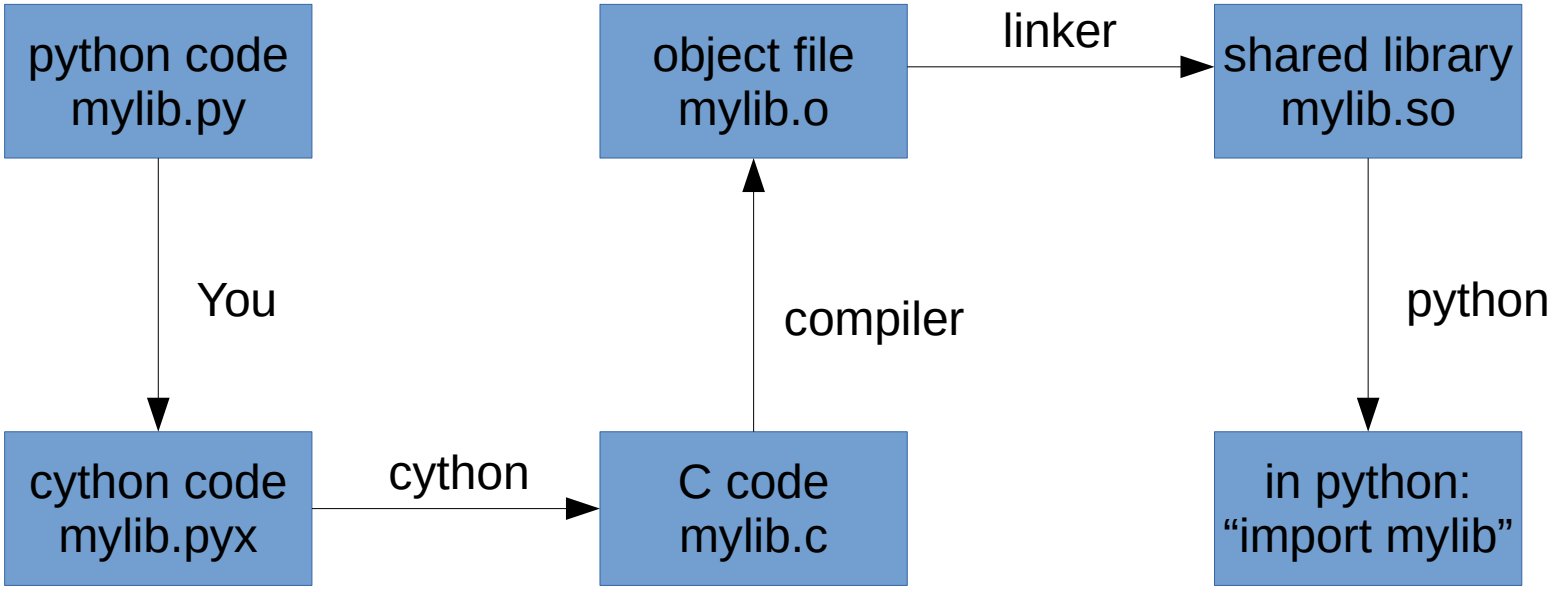


Cython C-Extensions for Python

- What is Cython?
  - Cython is an optimising static compiler
    - For both Python and the extended Cython languages
  - Also a creole programming language which extends the Python language with optional static type declarations
- What is it good for?
  - It makes writing C extensions for Python easy
  - Quickly port the performance-critical part of your code to C
- What else can it do?
  - Wrap existing C/C++ libraries
    - In a very high-performance manner
  - Embed Python interpreter in C/C++ code and have C/C++ call into Python
- Limitations
  - Large learning curve
  - Laborious for wrapping large libraries (not as automated as SWIG)
- Advantages
  - Easy and quick way to optimize existing Python code if you don’t already have C/C++ code
  - Performance is by far the best of the tools covered here
    - Cython code often runs faster than hand-written C
  - Good C++ support
  - Good debugging support
  - Allows you to rapidly build a prototype in Python and optimize it as needed
    - Can get performance close to a pure C implementation but in a fraction of the time spent in terms of developer hours

What does Cython do?

- Makes it much easier to create a C Extension Module for Python
- Allows you to write code which looks very similar to Python
  - With optional static type declarations added
  - But which can be automatically compiled to a native binary library
- How?



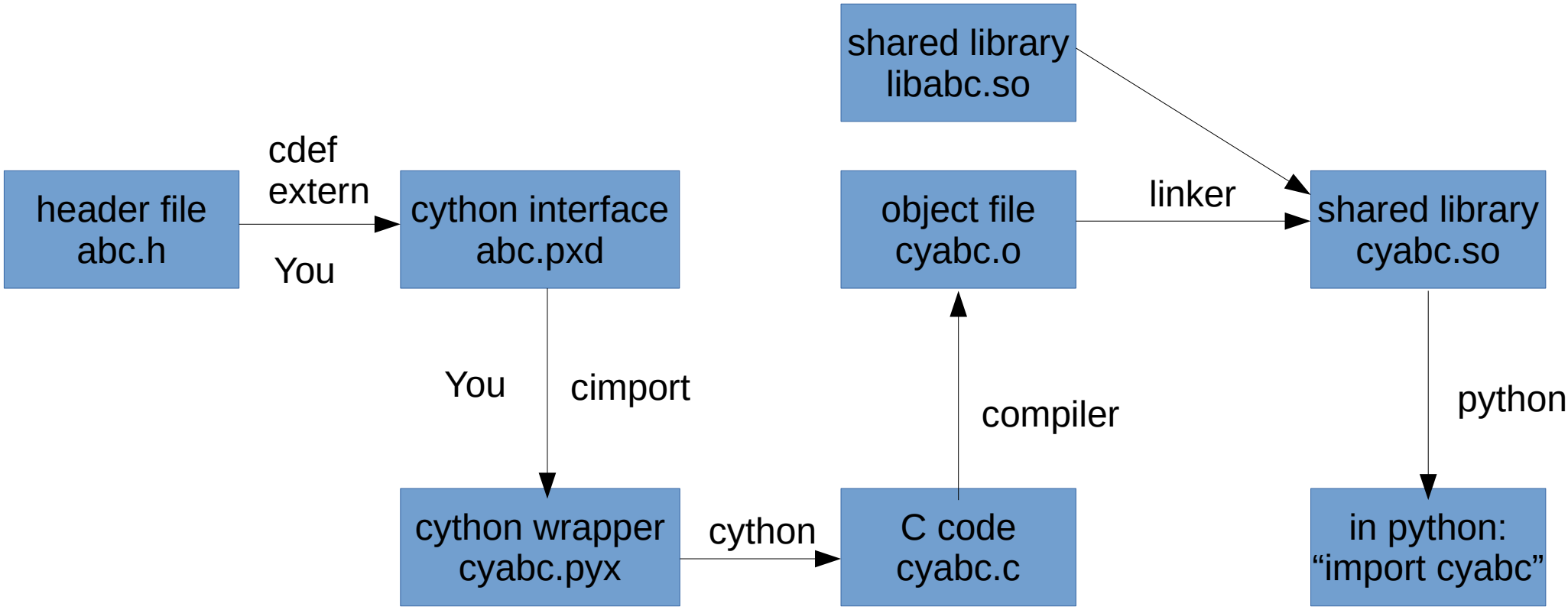
How to use Cython to optimize existing Python code

- Conversion process
  1) Python code- start with a Python module you would like to speed up (mylib.py)
  2) Cython code- rename this file to have a *.pyx file extension (mylib.pyx)
  3) Edit – edit one or more lines to take advantage of what Cython has to offer
  4) cython – invoke cython to create a C language file (mylib.c)
  5) gcc – C file is compiled into an object file (mylib.o)
  6) linker – object file is linked into a shared library (mylib.so or mylib.pyd)
  7) Python – at this point the shared library can be imported from a Python script (“import mylib”)

- Common ways of compiling Cython code?
  - Easiest: Jupyter Notebook
    - Interactive browser-based interface which allows you to evaluate cells of source code
    - Makes it very easy to rapidly test ideas and measure performance
  - Easy: “cythonize” command line script
    - Provided as part of Cython
    - Wrapper around all of the compilation steps shown in the conversion process above (steps 4, 5, and 6)
    - cythonize -b mylib.pyx
      - Generates mylib.c and then compiles that to mylib.so, which is directly importable in Python
    - cythonize -b -a mylib.pyx
      - The -a flag generates an HTML annotation file which indicates which lines are pure native code (fast) and which interact with the Python interpreter (slower)
  - Medium: setup.py
    - Use the Python packaging tools themselves to incorporate Cython compilation into your setup.py file
    - Can call setup.py with extra arguments to compile your Cython files
    - Makes it easy to combine packaging with compilation and is best if you distribute your code to others

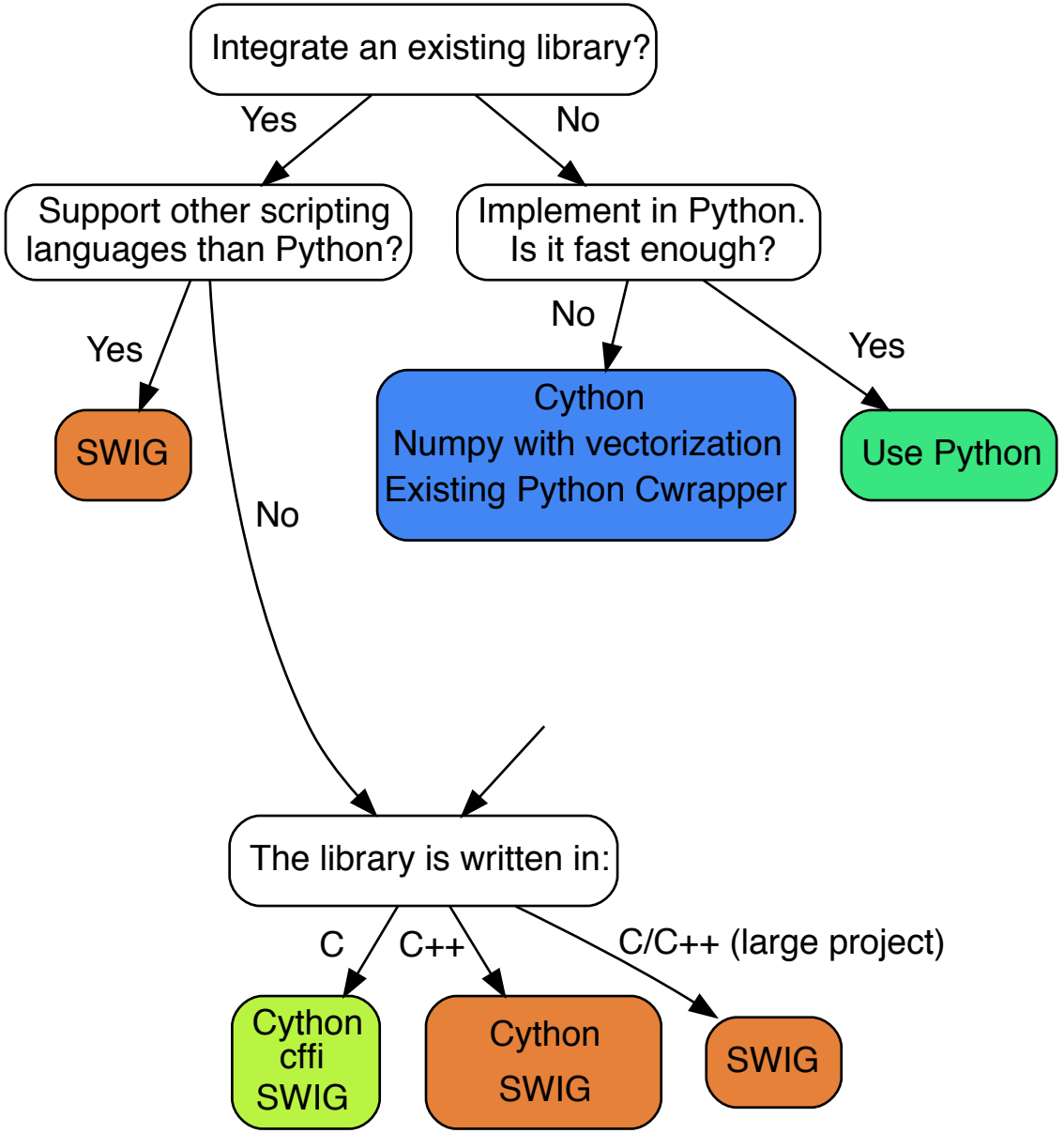
Wrap existing library with Cython

- A bit more complicated than using Cython to optimize Python
- Also a bit more complicated than using SWIG to wrap
- But a higher performance way to wrap than using SWIG



Which tool should I use?

- It Depends



Where to Learn More

- Cython
  - Main Site: http://cython.org
  - Documentation: http://docs.cython.org
  - 4 hour training video from SciPy 2015: https://www.youtube.com/watch?v=gMvkiQ-gOW8
    - Code for this video on GitHub at: https://github.com/kwmsmith/scipy-2015-cython-tutorial
- SWIG
  - Main Site: http://www.swig.org
  - Documentation: http://www.swig.org/Doc3.0
  - 40 minute training video from Univ. Oslo: https://www.youtube.com/watch?v=J-iVTLp6M9I
    - Code for this video on GitHub at: https://github.com/UiO-INF3331/code-snippets-16/tree/master/mixed/swig
- CFFI
  - Main Site: http://cffi.readthedocs.io
  - Mailing List: https://groups.google.com/forum/#!forum/python-cffi
  - 1 hour training video from PyCon: https://www.youtube.com/watch?v=ThDFmuXH15k