Python, C, C++, and Fortran Relationship Status: It’s Not That Complicated

Philip Semanchuk (philip@PySpoken.com)

This presentation is part of a talk I gave at PyData Carolinas 2016.

Introduction

- Python’s ability to talk to “foreign” languages is one of its strengths
- Too many choices == cloudy and mysterious

Roadmap

0. Basic principles of how Python can talk to C, and how this extends to Fortran, and C++

1. The three ways to connect Python with a foreign language

2. Some suggested tools

3. Q&A

0.0 - Python’s C API

- For us, “Python” == “CPython”
- C API at its core
- Stable, well-documented

0.1 - New Dict Example

Python

```
d = {42: 'The answer'}
```

C API

```
static PyObject *
create_a_new_dict(PyObject *self, PyObject *args) {
    PyObject *p_dict = NULL;

    p_dict = PyDict_New();

    PyDict_SetItem(p_dict, PyLong_FromLong(42),
                   PyUnicode_FromString("The answer"));

    return p_dict;
}
```

0.2 - Fortran and C++

- Fortran maps to C reasonably well (similar primitives)
- Fortran's ISO_C_BINDING helps
- C++ has C primitives plus objects, templates, exceptions, etc.

1.0 - Three Choices

0. Wrap

1. Extend

2. Embed

(No connection with *embrace, extend, extinguish*!)

1.1 - Wrapping

- Most common option
- Creates a Python-friendly layer for an *existing* library
- Binary translation
- Idiomatic translation (is it “wafer thin”?)

1.1.1 - Wrapping Example

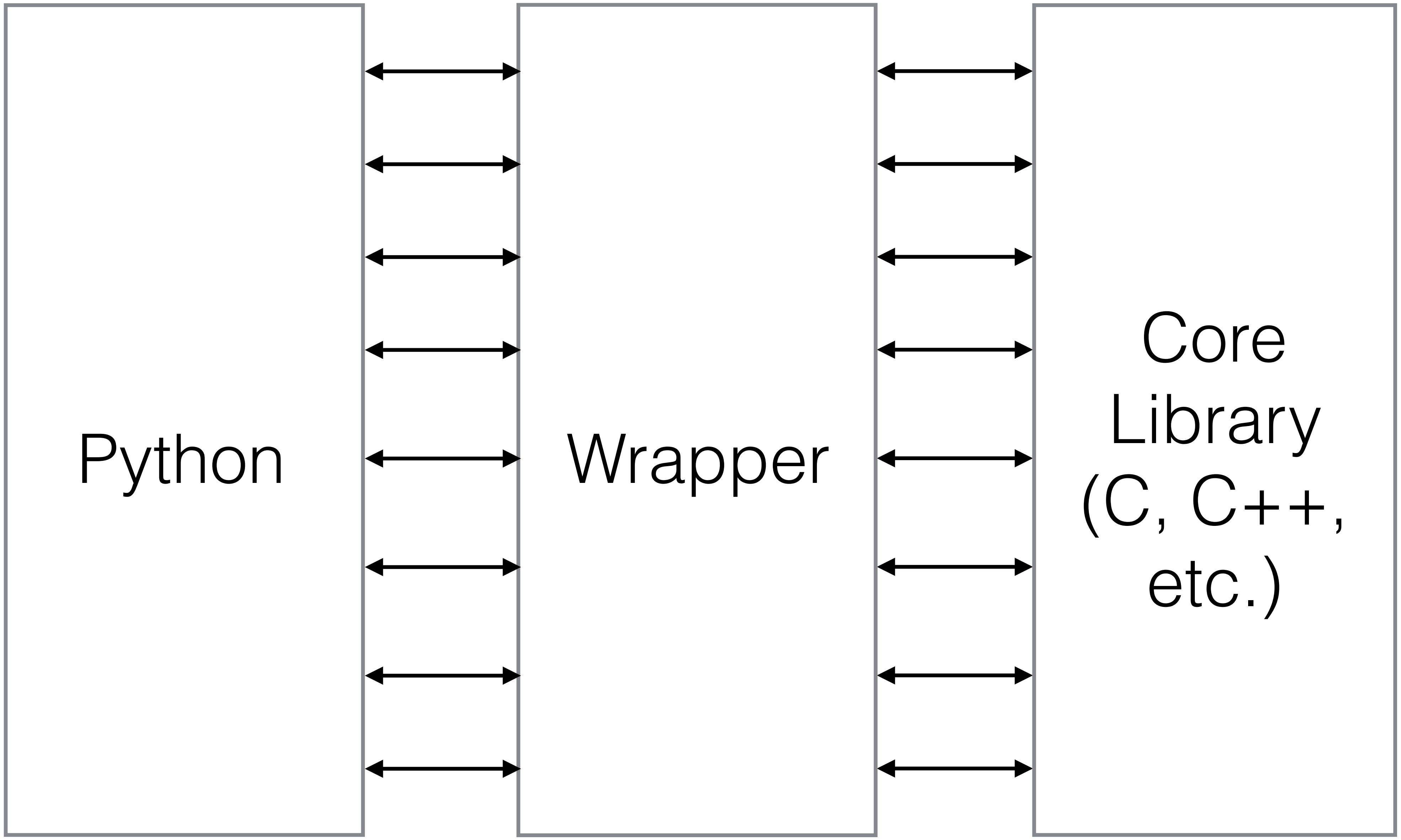
“Wafer Thin”

```
class some_lib.Foo()

Foo.GetValue()
    Returns the value of this instance

Foo.SetValue(new_value)
    Sets the value of this instance
```

Pythonic

```
Foo.value
   Property that gets/sets the value for this instance
```

1.1.2 - Thin Wrapper

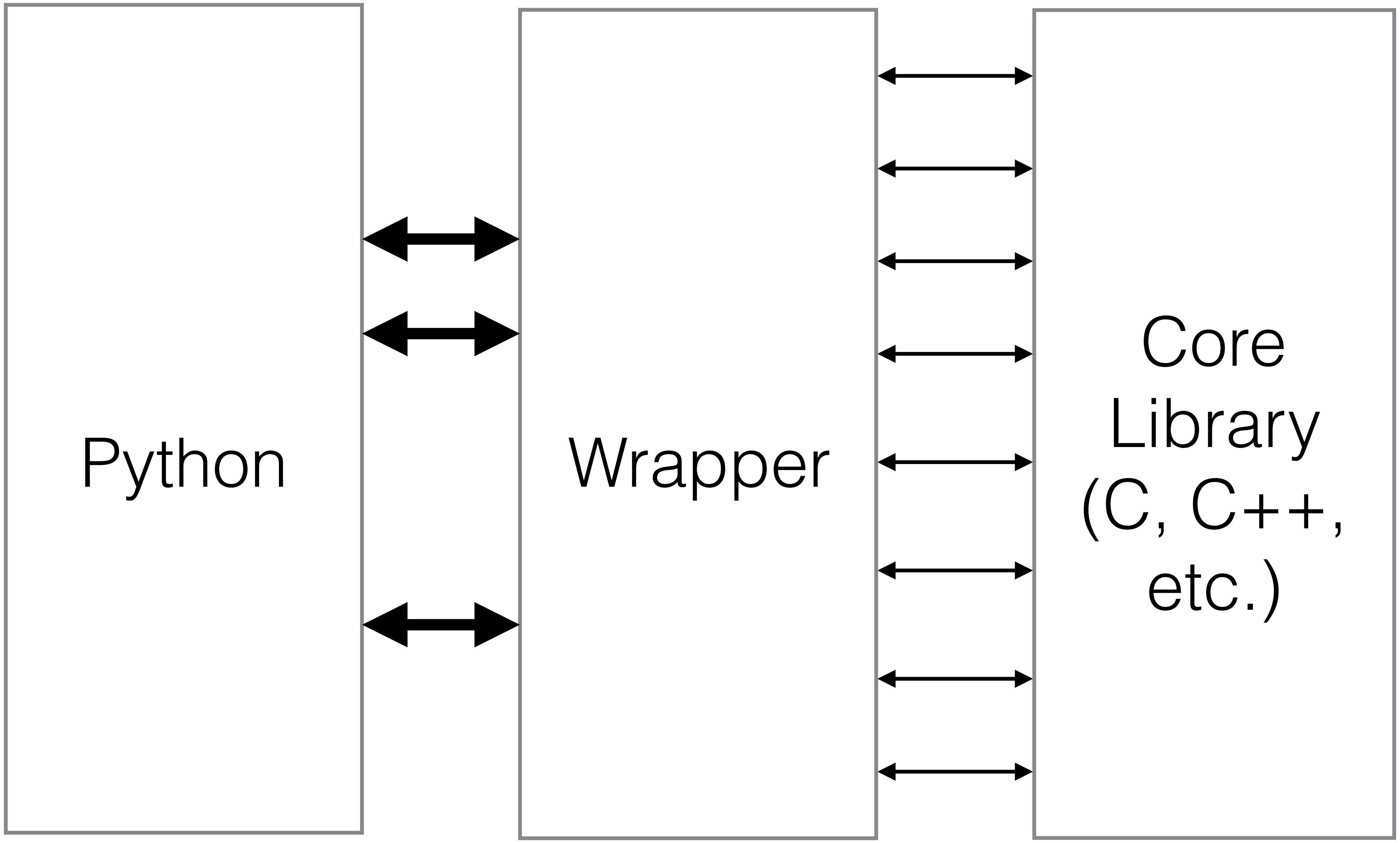


Python

Wrapper

Core Library (C, C++, etc.)

1.1.3 - Thick Wrapper



Python

Wrapper

Core Library (C, C++, etc.)

1.1.4 - Wrapping

Useful when —

- You have an *existing* library that works
- You want to use it from Python
- You can’t (or don’t want to) modify it

1.2 - Extending

- Extending is not adding keywords or syntax
- Just a fancy name for an ordinary, import-able module, but written in a language that’s not Python
- Only accessible to Python
- Self-contained (relative to wrapped library)

1.2.1 - Extending

Useful when —

- You need foreign language features (e.g. speed)
- You don’t need your code accessible anywhere but Python

1.3 - Embedding

- Uncommon but interesting
- Embeds a Python interpreter in your foreign language executable
- Can call Python efficiently

1.3.1 - Embedding

Useful when —

- Your C/C++/Fortran needs to call Python
- Wrappers and extension modules offer the opposite
- You want Python as a scripting language

2.0 - Tools

- Tools for all three techniques (mostly for wrapping)

2.1 - Tools for Wrappers

- ctypes (from the Python standard library)
  - No compiler needed
  - Lightweight, doesn't offer any automation (but see https://github.com/davidjamesca/ctypesgen?)
  - No C++ support

2.1.1 - ctypes Example 1

Fortran

```fortran
subroutine say_hello(n_iterations)
implicit none
integer i, n_iterations
do i = 1,n_iterations
   print *, i, "Hello PyData Carolinas 2016!"
enddo
return
end
```

Python/ctypes

```python
py_n_iterations = 5
fort_n_iterations = ctypes.c_long(py_n_iterations)
# Pass by reference, not by value
the_library.say_hello_(ctypes.pointer(fort_n_iterations))
the_library.say_hello_(5) # pass by value ==> segfault!
```

2.1.2 - ctypes Example 2

Fortran

```fortran
subroutine say_hello(n_iterations)
use iso_c_binding, only: c_int
implicit none
integer i
integer(c_int), intent(in), VALUE :: n_iterations
do i = 1,n_iterations
   print *, i, "Hello PyData Carolinas 2016!"
enddo
return
end
```

Python/ctypes

```python
the_library.say_hello_(5) # pass by value ==> works!
```

2.1.3 - ctypes Example 3

Fortran

```
subroutine do_something_wrapper(some_data_r, some_data_i, n_elements)
use iso_c_binding, only: c_double, c_int
implicit none
integer(c_int), intent(in), VALUE :: n_elements
real(c_double), intent(inout) :: some_data_r(n_elements)
real(c_double), intent(inout) :: some_data_i(n_elements)
integer i

do i = 1,size(some_data)
   some_data(i) = cmplx(some_data_r(i), some_data_i(i), kind=kind(1.0d0))
end do

! do_something() is defined elsewhere
!call do_something(some_data)

do i = 1,size(some_data)
   some_data_r(i) = REALPART(some_data(i))
   some_data_i(i) = IMAGPART(some_data(i))
end do
return
end
```

2.2 - Other C/Fortran Wrapping Tools

- CFFI (3rd party, FOSS) interprets C function declarations; generates wrappers. No Fortran, C++
- Numpy’s F2Py helps wrapping Fortran
  - Nice feature set
  - No one talks about it (is that good or bad?)

2.3 - Wrapping C++

- SWIG (3rd party, FOSS)
  - Parses C/C++ headers, generates wrappers
  - Tweakable interface files provide hints
  - Ambitious, magic, wonderful, a little dangerous
  - Debugging magic is difficult!
  - Excellent for large C++ projects

2.4 - Wrapping C++

- Boost.Python
  - Fewer users/supporters?
  - Has some ardent admirers

3.0 - Cython

- Python, with C sauce
- Triple threat: can wrap, extend, embed!
- Rewards immediately; encourages exploration
- Knows about numpy, has decent C++ support
- Fortran90.org has a tutorial on Cython calling Fortran
- No automated interface generation (XDress?)

What Was All That About?

- CPython’s C API is good to be aware of, even if you don’t learn it or use it directly

- You can wrap, extend, and embed

- You don’t *need* any tools, but they sure help!

- Cython covers most cases; it should always be in your list of candidate tools

Thank you!

Philip Semanchuk (philip@PySpoken.com)