Shroud: A Tool for Creating Fortran Interfaces for C++ Libraries

FortranCon 2020

Lee Taylor
Computing Directorate

July 2-4, 2020

LLNL-PRES-811975

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Overview

- Motivation
- Fortran interoperability with C
- Shroud
  - Layers of wrappers
  - Examples
  - Arrays
  - Memory Management
  - Other C++ features

verb
1.wrap or dress (a body) in a shroud for burial.
2.cover or envelop so as to conceal from view.

https://pixabay.com/vectors/mummy-pharaoh-egypt-egyptian-151304/

Code development at Lawrence Livermore

- LLNL is a U.S. Department of Energy Research Laboratory
  - Founded 1952
  - Leader in High Performance Computing
- Fortran has a long history at Livermore
  - Worked on original Fortran compiler with John Backus [1]
- C++ is now the predominant language at LLNL
- C++ libraries are being used by Fortran codes
- Started CS toolkit in C++ in 2015, Axom
  - Share functionality among codes: datastore, meshing, logging, ...
  - Requirements included a Fortran API

[1] https://computing.llnl.gov/projects/co-design/lokke_345372.pdf

Shroud arose out of the Computer Science Toolkit

Fortran Standard and C Interoperability

From DIY to standardized

- LRLTRAN at LLNL (1967) C was 1972
  - Dynamic memory using integer pointers, a.k.a. Cray pointers
  - `pointer (iparray, array(*))`
- Fortran 90
  - `ALLOCATABLE`, `POINTER` – includes meta information (T/K/R)
- Fortran 2003 – Interoperability with C
  - `VALUE` attribute – call-by-reference/call-by-value
  - `BIND(C)` – name mangling
  - iso_c_binding module – type matching
- 2012 TS 29113 – Further interoperability with C
  - ISO_Fortran_binding.h, C API to access `ALLOCATABLE`, `POINTER`
- Fortran 2018 – includes TS 29113

You have to code with the compiler you have

not the compiler you want

- Fortran 2003 is widely available
  - gfortran 4.3, March 2008; pgi
- TS 29113 is available in most recent compilers
  - gfortran 9.1, May 2019
    - Also added Asynchronous I/O from 2003
- Red-Hat Enterprise 7 provides gcc 4.8
  - Later versions must be installed explicitly
- Not a Fortran specific issue
  - Axom recently decided to use C++11 as the minimum standard
  - Python 2 still in use
- Using TS 29113 in Shroud would simplify some wrappers

Shroud uses Fortran 2003 as the minimum standard level.

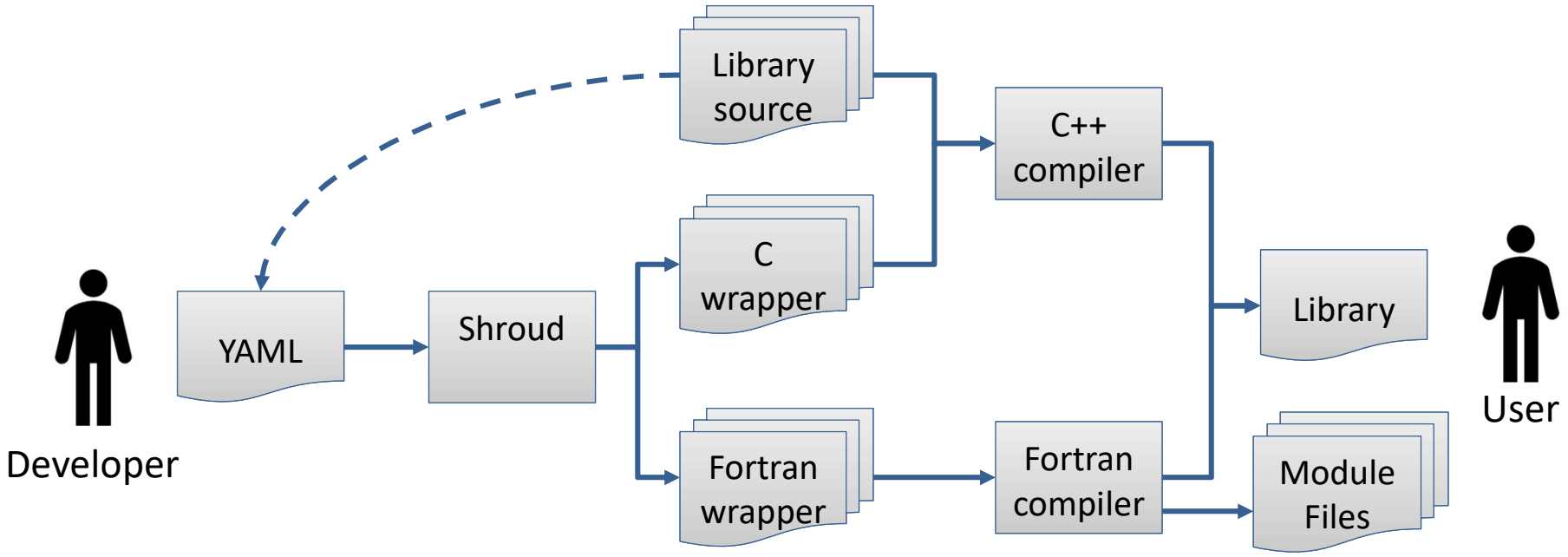
Shroud Generates Source to Use C++ Libraries

- Target audience
  - C++ developer who needs to create a Fortran API
    - Fortran? Isn't that the language in all uppercase?
  - Fortran develop who want to use a cool new C++ library
    - BIND(C) isn't working!
  - Any developer who want to avoid writing lots of boilerplate
- Advantages
  - Simplify the creation of wrappers function
  - Uses Fortran standard features
  - Preserves the object-oriented style of C++ classes
  - Create a Fortran idiomatic API from the C++ API
- Many C++ features are supported
  - typedefs, classes, structs, functions, namespaces, templates
  - function overloading, default arguments

What Shroud is Not

- It is not C++ calls Fortran
  - But, will create a C API for C++ library
- Does not parse header files
  - But it does parse declarations in YAML files
  - Only wrap the functions you need
- 100% coverage of C++ features
  - Template abuse, expression templates
  - Wrappers for Google's gtest?
- Scale
  - Wrappers for Qt?
- Complete
  - Inheritance

Shroud Workflow



YAML Input

- Yet Another Markup Language
  - Dictionaries, lists – superset of JSON
  - Uses whitespace/indentation for scope
- User adds function declarations from header files
  - Cut-and-paste from header files
- Options and format string to control generated code
  - Wrapper names
  - File names and suffix

YAML is a human-readable format.

Sample YAML File

Written by the developer from the library header file

```
library: pointers
cxx_header: pointers.hpp
language: c++
options:
  debug: True
format:
  C_prefix: POINT_
declarations:
- decl: void intargs_in(const int *arg)
- decl: void intargs_inout(int *arg)
```

After prologue, much is cut-and-paste.

Attributes Define Semantics of Arguments

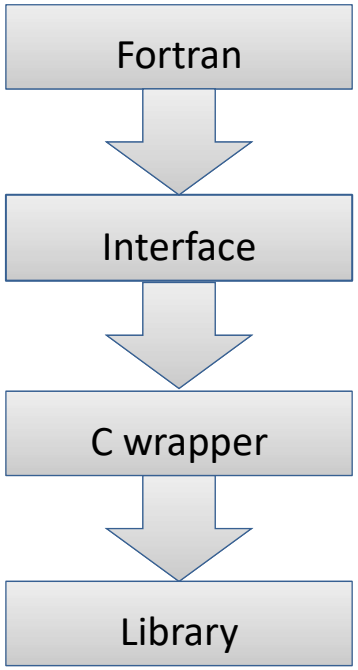
and function results

- intent
  - in, out, inout
  - Wrapper may require copy-in and/or copy-out
- hidden
  - Not part of Fortran API
  - Typically intent(out)
- implied
  - Not part of Fortran API
  - Value of argument is implied from other arguments
  - Typically intent(in)
- dimension
  - Literal shape of input argument
- rank
  - Assumed-shape argument
- owner
  - Defines who is responsible to release memory
  - library, caller
- deref
  - How pointer will be returned
  - pointer, allocatable, raw

Shroud Creates Wrappers at Several Levels

Uses the minimum number of levels possible

- Fortran
  - Module subprogram
- Interface
  - Always created, zero cost
  - Equivalent to C++ prototype
- C Wrapper
  - Compiled with C++
  - C API via `extern "C"`
  - "bufferify" functions
    - Includes metadata arguments
- User's Library



Interface Block Only Wrapper

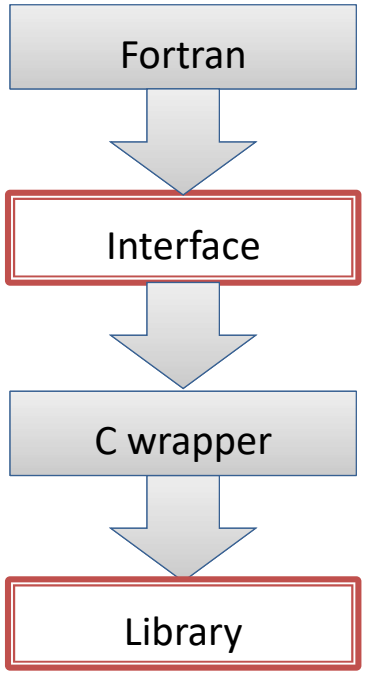
Subroutine with scalar arguments

YAML

```
language: c
declarations:
- decl: void Worker(int arg1, int *arg2)
```

Interface

```
interface
   subroutine worker(arg1, arg2) &
     bind(C, name="Worker")
     use iso_c_binding, only : C_INT
     implicit none
     integer(C_INT), value, intent(IN) :: arg1
     integer(C_INT), intent(INOUT) :: arg2
   end subroutine worker
 end interface
```



Fortran 2003 interoperability with C:
name mangling, call-by-value, type matching.

Wrapper for C++ Function

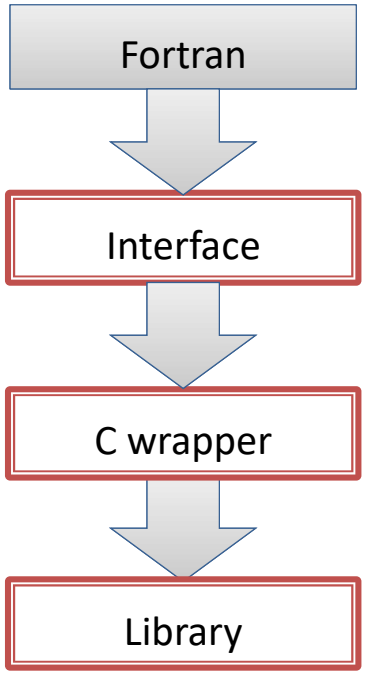
Needed to control name mangling for `bind(C)`

Interface

```
bind(C, name="LIB_worker")
```

C Wrapper

```
extern "C" void LIB_worker(int arg1, int *arg2)
{ Worker(arg1, arg2); }
```



- C++ compiler mangles names to provide context
  - Fortran compilers mangle names for module functions
- Provides a C API for C users
  - Compiled with C++ compiler and `extern "C"`
  - Shroud refers to this as the "C wrapper"
- Necessary when "function" is a macro or function pointer

Fortran Wrapper

When an interface block is not sufficient

YAML

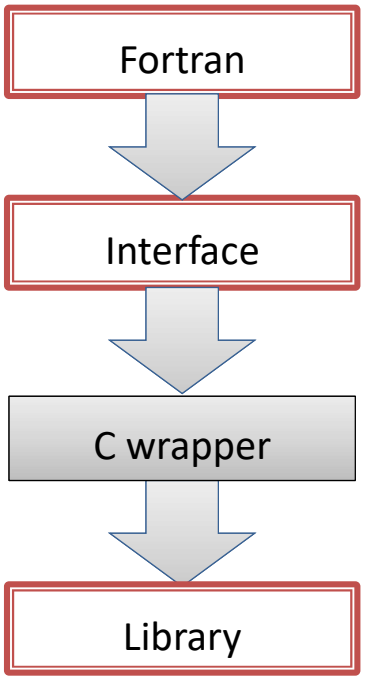
```
- decl: void setName(const char *name)
```

Fortran

```
subroutine set_name(name)
  use iso_c_binding, only : C_NULL_CHAR
  character(len=*), intent(IN) :: name
  call c_set_name(trim(name)//C_NULL_CHAR)
end subroutine set_name
```

Interface

```
subroutine c_get_name(name) &
     bind(C, name="LIB_get_name")
   character(kind=C_CHAR), intent(OUT) :: name(*)
end subroutine c_get_name
```



`const char *` defaults to intent(in).

Fortran wrapper does the work to NULL terminate the string.

C Bufferify Wrapper

Pass metadata in additional arguments

YAML

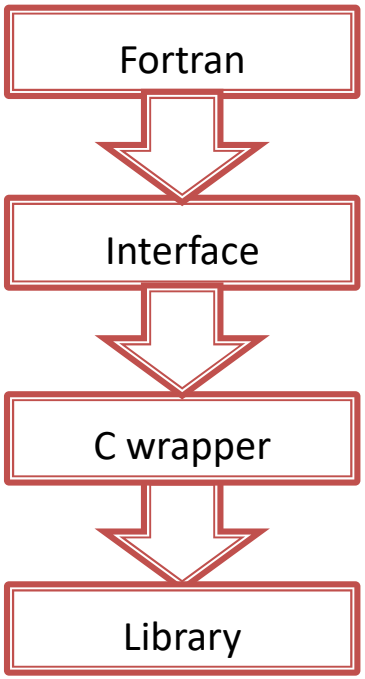
```
- decl: void getName(std::string &name +intent(out))
```

Fortran

```
subroutine get_name(name)
  character(len=*), intent(OUT) :: name
  call c_get_name_bufferify(name, &
    len(name, kind=C_INT))
end subroutine get_name
```

C Wrapper

```
void LIB_get_name_bufferify(char * name, int Nname) {
 std::string SH_name;
 getName(SH_name);
 ShroudStrCopy(name, Nname, SH_name.data(), SH_name.size());}
```



Current compilers pass the length as a hidden argument.
Shroud passes the length explicitly.

Arrays are Fundamental to Fortran

Converting Pointer to Array (and references)

- Must tell Shroud the shape of the array using an attribute
  - rank, intent(in) assumed-shape argument
  - dimension
- Dimension is a list of C++ expressions
  - `(10,20)`, `(nitems)`, `(ReturnSize())`
- `SHROUD_array` derived type saves metadata
  - Eventually use TS 29113 `CFI_cdesc_t`
- Multidimensional arrays
  - Row major to column major

Fortran is a Domain Specific Language for Arrays.

Converting Pointer to Array

C++ pointer converted to Fortran `POINTER` array

YAML

```
- decl: int *getArray(int *narray +intent(out)+hidden)
         +dimension(narray)
```

- Attributes
  - Hidden arguments are not in the Fortran API
  - Dimension is shape of returned pointer

Fortran example

```
integer(C_INT), pointer :: values
integer(C_INT) narray
values => get_array()
narray = size(values)
values(:) = 0
```

C++ example

```
int nvalues;
int *value = getArray(&nvalues);
for (int i=0; i<nvalues; i++)
  values[i] = 0;
```

Fortran usage is what a Fortran programmer would expect.

Converting Pointer to Array

Fortran wrapper part, convert to Fortran `POINTER`

YAML

```
- decl: int *getArray(int *narray +intent(out)+hidden)
        +dimension(narray)
```

Fortran

```
function get_array() result(SHT_rv)
  use iso_c_binding, only : C_INT, C_PTR, c_f_pointer
  type(SHROUD_array) :: DSHC_rv
  integer(C_INT) :: narray
  integer(C_INT), pointer :: SHT_rv(:)
  type(C_PTR) :: SHT_ptr
  SHT_ptr = c_get_array_bufferify(DSHC_rv, narray)
  call c_f_pointer(SHT_ptr, SHT_rv, DSHC_rv%shape(1:1))
end function get_array
```

- `SHROUD_array` holds metadata for array
- `c_f_pointer` converts to Fortran pointer

Convert to Fortran array with c_f_pointer.

Converting Pointer to Array

C wrapper part, fill in metadata

YAML

```
- decl: int *getArray(int *narray +intent(out)+hidden)
          +dimension(narray)
```

C Wrapper

```
int * LIB_get_array_bufferify
  (LIB_SHROUD_array *DSHC_rv, int * narray) {
  int * SHC_rv = getArray(narray);
  DSHC_rv->cxx.addr = SHC_rv;
  DSHC_rv->cxx.idtor = 0;
  DSHC_rv->addr.base = SHC_rv;
  DSHC_rv->type = SH_TYPE_INT;
  DSHC_rv->elem_len = sizeof(int);
  DSHC_rv->rank = 1;
  DSHC_rv->shape[0] = *narray;
  DSHC_rv->size = DSHC_rv->shape[0];
  return SHC_rv;
}
```

capsule

+dimension(narray)

- Capsule used with memory management
- Shape from dimension attribute
  - Function return value

SHROUD_array is a subset of CFI_cdesc_t from TS 29113

Memory management

- By default Shroud assumes owner(library) attribute
  - owner(caller) to assume ownership
- Shadow class/proxy pattern
  - Contains a pointer to memory returned by library
  - And an index used to release the memory

All problems in computer science can be solved by another level of indirection.
--- David Wheeler

Cannot use `deallocate` statement on C++ memory.

Capsule contains pointer to C++ memory
and information to release the memory

C Wrapper

```
struct s_LIB_SHROUD_capsule_data {
  void *addr; /* address of C++ memory */
  int idtor; /* index of destructor */
};
```

Passed Between Fortran and C Wrapper

If +owner(library), idtor will be 0 (no-op)

Fortran

```
type SHROUD_capsule
  private
  type(SHROUD_capsule_data) :: mem
contains
  final :: SHROUD_capsule_final
  procedure :: delete => SHROUD_capsule_delete
end type SHROUD_capsule
```

- Shadow Class
  - C capsule in a Fortran derived type with type-bound procedures

Inspired by PyCapsule from Python C API

Ownership of C++ Array in Fortran

set with owner attribute

YAML

```
- decl: int *getArray(int *narray +intent(out)+hidden)
          +dimension(narray) +owner(caller)
```

Fortran Use

```
integer(C_INT), pointer :: data(:)
type(SHROUD_capsule) capsule
data => get_pointer(capsule)
! use data
call data%delete()
```

- Idiomatic use of array
  - User does not see `type(C_PTR)`
  - +deref(raw) will return `type(C_PTR)`
- Capsule may also be released by FINAL clause
  - Similar to destructor

Allows C++ pointer to be released in Fortran

C++ Class Creates a Fortran Shadow Class

using type-bound procedures

YAML

```
- decl: class Animal
  declarations:
  - decl: Animal()
  - decl: void speak()
```

Fortran

```
type animal
  type(SHROUD_animal_capsule) :: cxxmem
contains
  procedure :: speak => animal_speak
end type animal
```

Fortran example

```
use library_mod
type(animal) dog
dog = animal()
call dog%speak()
```

C++ example

```
#include <library.cpp>
Animal dog;
dog.speak();
```

- `SHROUD_animal_capsule` same as `SHROUD_capsule`
  - Renamed for type safety
- Struct uses a `bind(C)` derived type

Fortran 2003 object-oriented features.

C Wrapper Class Method

C++ `this` passed as explicit argument

Fortran

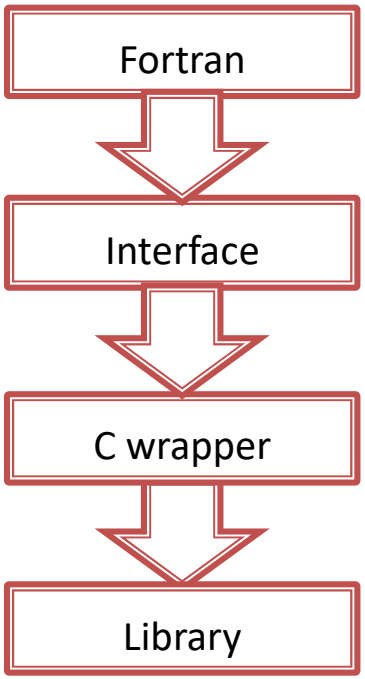
```
subroutine animal_speak(obj)
  class(animal) :: obj
  call c_animal_speak(obj%cxxmem)
end subroutine animal_speak
```

Interface

```
subroutine c_animal_speak(self) &
   bind(C, name="LIB_Animal_speak")
  import :: SHROUD_animal_capsule
  type(SHROUD_animal_capsule), intent(IN) :: self
end subroutine c_animal_speak
```

C Wrapper

```
void LIB_Animal_speak(LIB_Animal * self) {
  Animal *SH_this = static_cast<Animal *>(self->addr);
  SH_this->speak();
}
```



Fortran passes capsule to C wrapper.
C Wapper uses C++ vtable.

Overloaded Functions and Default Arguments

Use generic interfaces

YAML

```
- decl: void SetValue(const std::string& name)
  format:
    function_suffix: _from_name
- decl: void SetValue(int indx)
  format:
    function_suffix: _from_index

- decl: void SetFlag(int flag = 0)
  default_arg_suffix:
  -  _zero
  -  _user
```

- Functions can have user defined suffix
  - Otherwise use a sequence number
- Can use generic or specific name

Fortran example

```
call set_value_from_name("name")
call set_value("name")
call set_value(1)

call set_flag_zero()
call set_flag()
call set_flag(1)
```

Generic interface is used to emulate C++ features

Templates Must Be Instantiated to be Wrapped

Function and Class Templates

YAML

```
- decl: template<T,U> void FunctionTU(T arg1, U arg2)
  cxx_template:
  - instantiation: <int, long>
  - instantiation: <float, double>
-  decl: template<typename T> class vector
   cxx_template:
   - instantiation: <int>
   - instantiation: <double>
```

- Generic interface created for functions
- Derived type created for each class

Fortran example

```
call function_tu(1_C_INT, 2_C_LONG)
call function_tu(1.2_C_FLOAT, 2.2_C_DOUBLE)

type(vector_double) v1
v1 = vector_double()
```

Splicer provides customization

Allow manual edits to be preserved

```
subroutine set_name(name)
  use iso_c_binding, only : C_NULL_CHAR
  character(len=*), intent(IN) :: name
  ! splicer begin function.set_name
  call c_set_name(trim(name)//C_NULL_CHAR)
  ! splicer end function.set_name
end subroutine set_name
```

- Delineated by splicer comments
- Define replacement text in YAML file or additional input file
  - Can be generated
- Splicers for additional functions and interfaces

Allows total customization

Shroud Gives Fortran Access to C++ Libraries

- Creates Fortran idiomatic API
- Creates a portable source
  - Fortran 2003
- Creates user readable source
- Easier to maintain interface
  - Changing YAML can change Interface, Fortran wrapper, C wrapper
- Nothing existed at the time
  - SWIG 4 fork at https://github.com/swig-fortran/swig

Fortran and Python Have Similarities

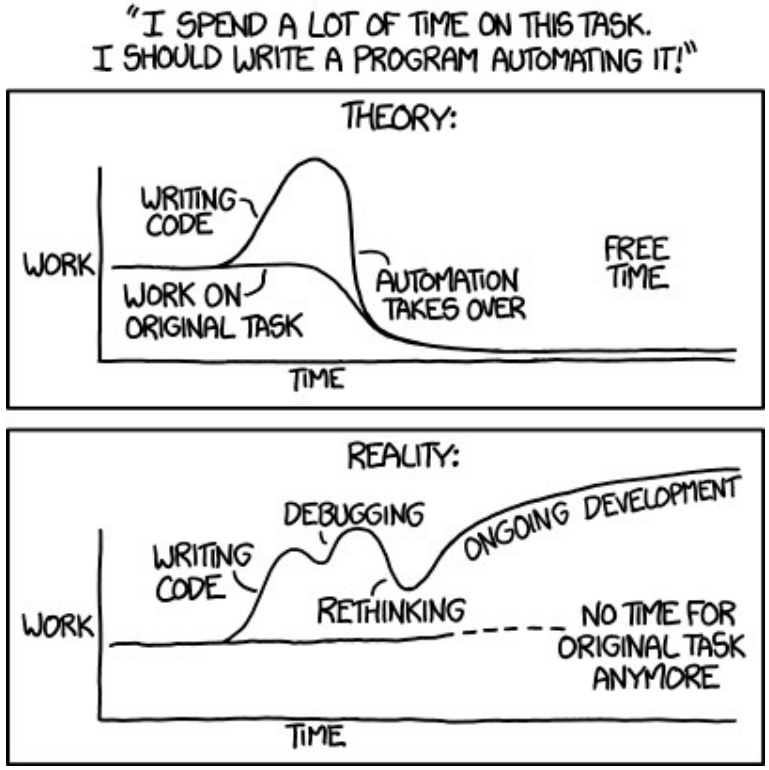
when creating wrappers

- The YAML input file can also be used to generate Python
  - Attribute provide similar guidance for Python wrapper
- Python wrapper is “free” (or low cost)
- Python wrapping is a very crowded field
  - NumPy support for rank/dimension attributes
  - PyCapsule used as base class for NumPy for memory management
  - Option to use list for arrays (array module)
  - Creates extension type for classes
  - Readable source

Two wrappers for one

Availability

- BSD-3-clause license
- software.llnl.gov
- github.com/LLNL/shroud
- pypi llnl-shroud



source: xkcd: Automation (from http://xkcd.com/1319)