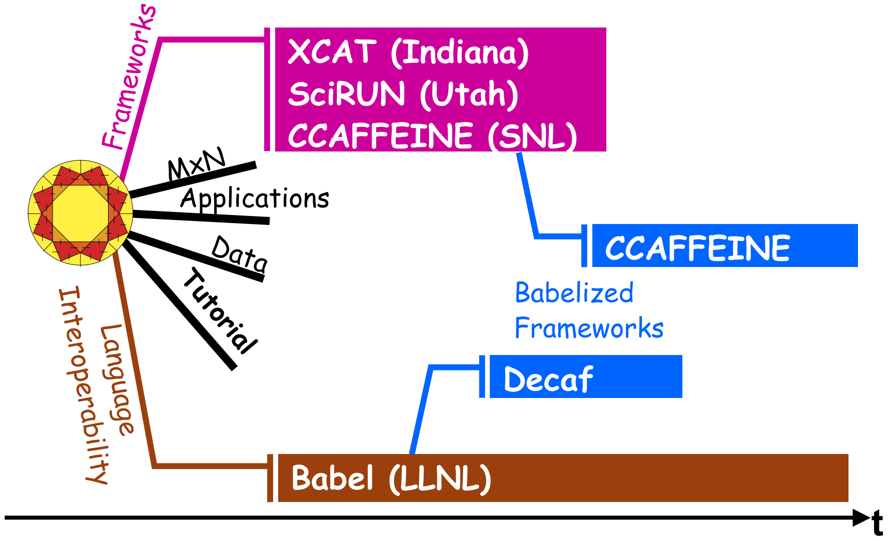
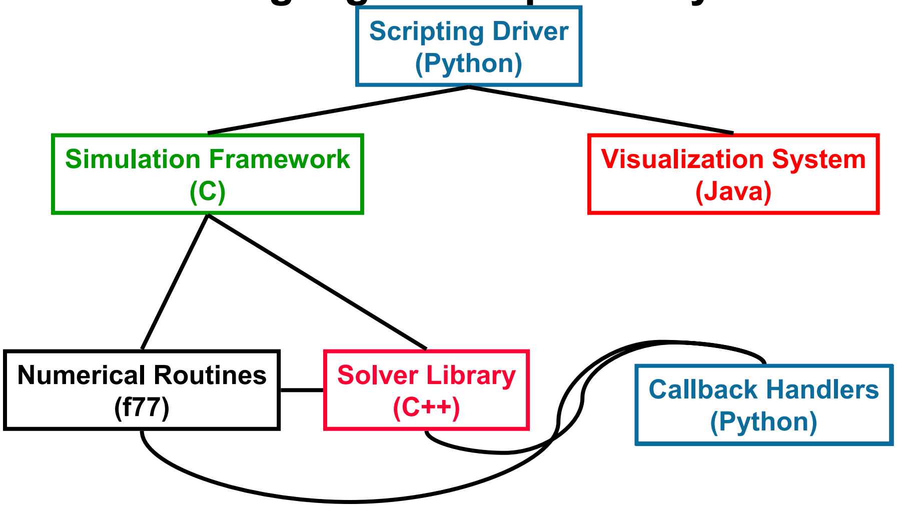
Language Interoperable CCA Components via BABEL

CCA Forum Tutorial Working Group
http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org

History of Babel & CCA



What I mean by “Language Interoperability”



Scripting Driver (Python)

Simulation Framework (C)

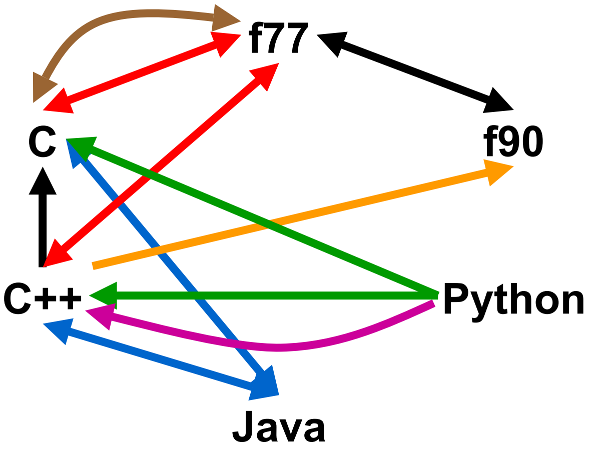
Visualization System (Java)

Numerical Routines (f77)

Solver Library (C++)

Callback Handlers (Python)

One reason why mixing languages is hard



Native

cfortran.h

SWIG

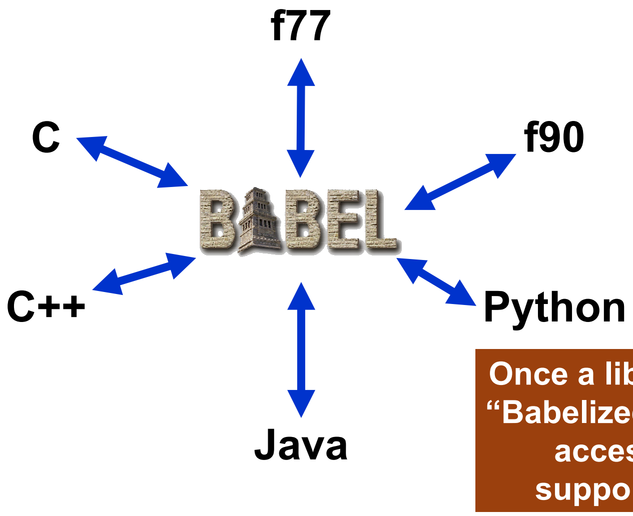
JNI

Siloon

Chasm

Platform Dependent

Babel makes all supported languages peers

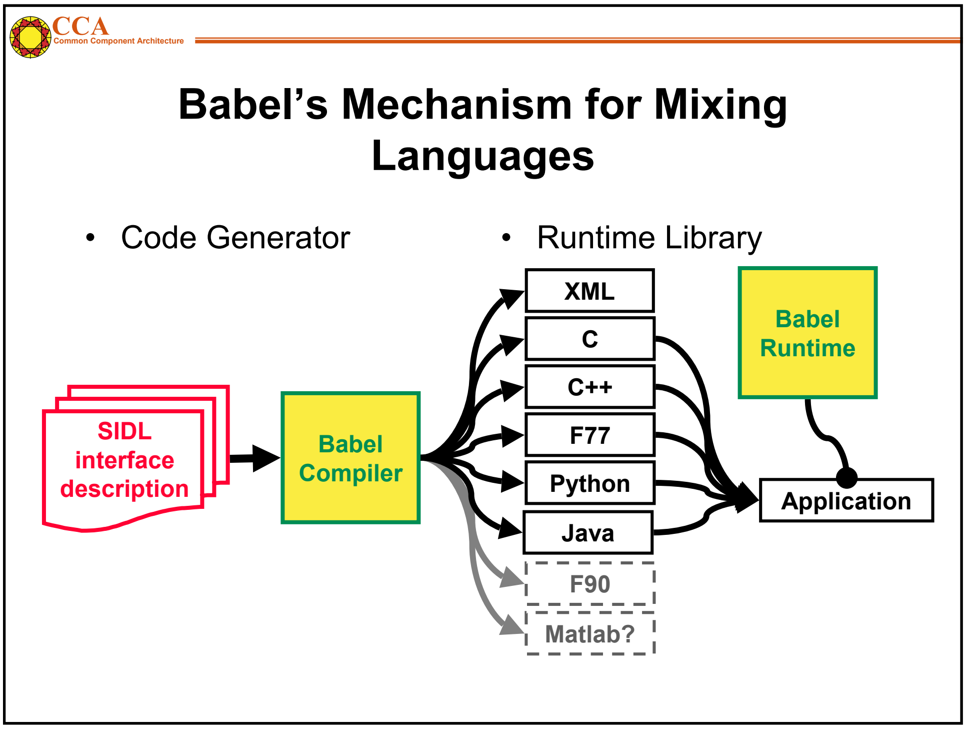


This is not an LCD Solution!

Once a library has been "Babelized" it is equally accessable from all supported languages

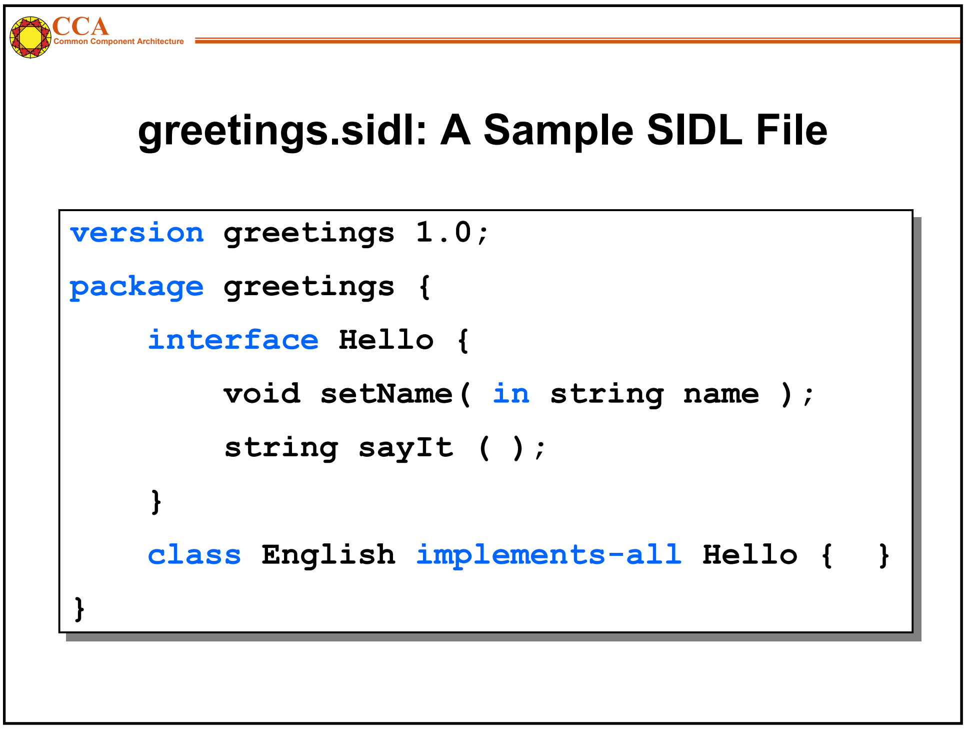
Babel Module's Outline

- Introduction
- Babel Basics
  - What Babel does and how
  - How to use Babel
  - Concepts needed for future modules
- Babel & CCA
  - Decaf Framework
  - Building language independent CCA components
  - Demo



Babel's Mechanism for Mixing Languages

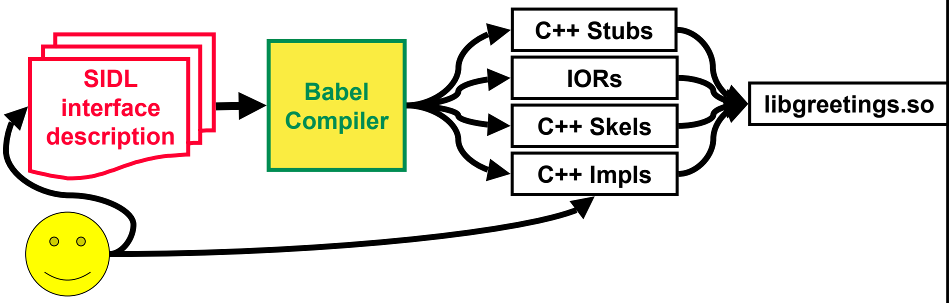
- Code Generator
- Runtime Library



greetings.sidl: A Sample SIDL File

```
version greetings 1.0;
package greetings {
    interface Hello {
        void setName( in string name );
        string sayIt ( );
    }
    class English implements-all Hello {   }
}
```

Library Developer Does This...



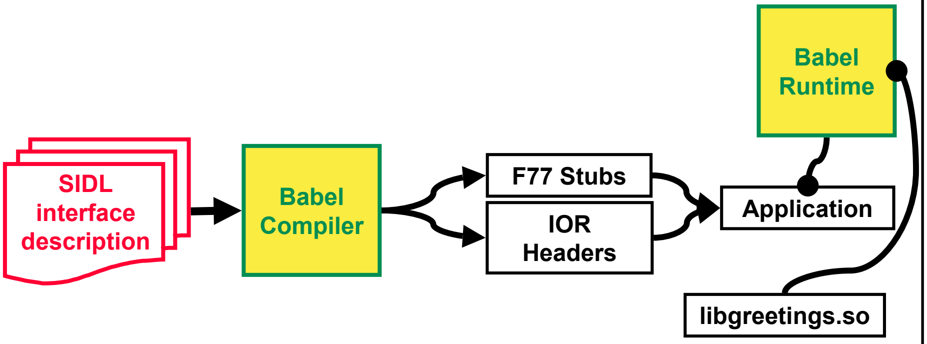
- `babel --server=C++ greetings.sidl`
- Add implementation details
- Compile & Link into Library/DLL

Adding the Implementation

```
namespace greetings {
class English_impl {
  private:
    // DO-NOT-DELETE splicer.begin(greetings.English._impl)
    string d_name;
    // DO-NOT-DELETE splicer.end(greetings.English._impl)
```

```
string
greetings::English_impl::sayIt()
throw ()
{
  // DO-NOT-DELETE splicer.begin(greetings.English.sayIt)
  string msg(“Hello “);
  return msg + d_name + “!”;
  // DO-NOT-DELETE splicer.end(greetings.English.sayIt)
}
```

Library User Does This...



- `babel --client=F77 greetings.sidl`
- Compile & Link generated Code & Runtime
- Place DLL in suitable location

SIDL 101: Classes & Interfaces

- SIDL has 3 user-defined objects
  - Interfaces – APIs only, No Implementation
  - Abstract Classes – 1+ methods unimplemented
  - Concrete Classes – All methods are implemented
- Inheritance (like Java/Objective C)
  - Interfaces may extend Interfaces
  - Classes extend no more than one Class
  - Classes can implement multiple Interfaces
- Only Concrete Classes can be Instantiated

SIDL 101: Methods and Arguments

- Methods are public virtual by default
  - static methods are not associated with an object instance
  - final methods can not be overridden
- Arguments have 3 parts
  - Mode: can be in, out, or inout (like CORBA)
  - Type: one of (bool, char, int, long, float, double, fcomplex, dcomplex, array<Type,Dimension>, enum, interface, class)
  - Name:

Babel Module’s Outline

- Introduction
- Babel Basics
  - What Babel does and how
  - How to use Babel
  - Concepts needed for future modules

➡ Babel & CCA
  - History & Current directions
  - Decaf Framework
  - Building language independent CCA components
  - Demo

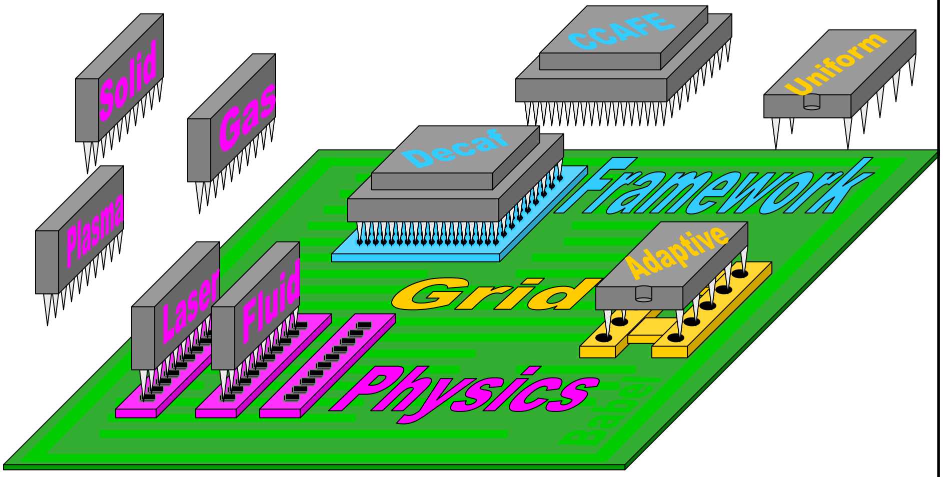
Decaf Details & Disclaimers

- Babel is a hardened tool
- Decaf is an example, not a product
  - Demonstrate Babel’s readiness for “real” CCA frameworks
  - Maintained as a stopgap
  - Distributed in “examples” subdirectory of Babel
- Decaf has no GUI

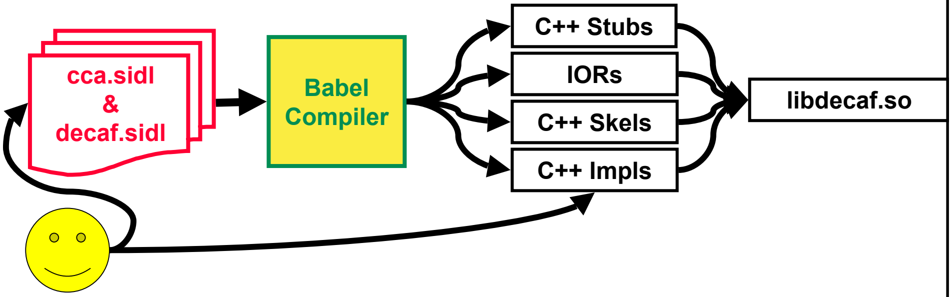
The CCA Spec is a SIDL File

```
version gov.cca 0.6;
package gov {
package cca {
    interface Port { }
    interface Component {
        void setServices( in Services svcs );
    }
    interface Services {
        Port getPort( in string portName );
        registerUsesPort( /*etc*/ );
        addProvidesPort( /*etc*/ );
    /*etc*/
```

The CCA from Babel’s POV

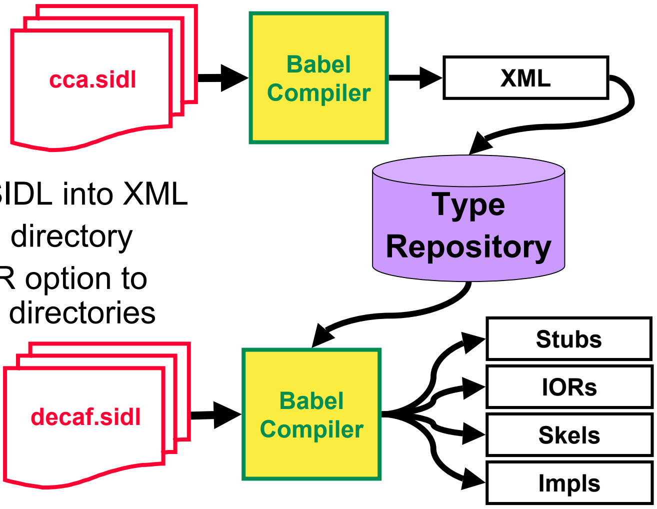


How I Implemented Decaf



- wrote decaf.sidl file
- `babel --server=C++ cca.sidl decaf.sidl`
- Add implementation details
- Compile & Link into Library/DLL

An Extra Babel Tip

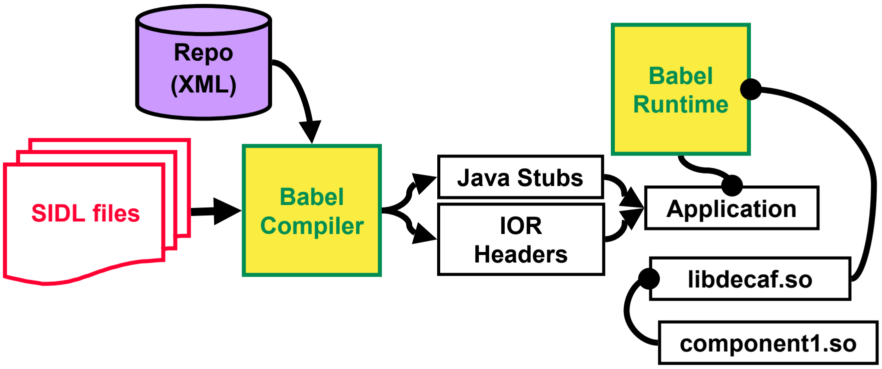


- “precompile” SIDL into XML
- store XML in a directory
- Use Babel’s –R option to specify search directories

How to Use CCA Components and Decaf

- Decaf doesn’t provide a GUI
- Simply program by explicitly
  - creating components
  - connecting ports
  - envoking the “goPort”
- Use Babel as needed to generate bindings in your language of choice
- Make sure Babel Runtime can locate DLLs for Decaf and any CCA components.

To Use the Decaf Framework



- `babel --client=Java –Rrepo function.sidl`
- Compile & Link generated Code & Runtime
- Place DLLs in suitable location

Example: A Driver in Python

```
import decaf.Framework
import gov.cca.ports.GoPort
if __name__ == ’__main__’:
  fwk = decaf.Framework.Framework()

  server = fwk.createInstance( ”ServerName”,
           ”HelloServer.Component”, 0 )
  client = fwk.createInstance( ”ClientName”,
           ”HelloClient.Component”, 0 )

  fwk.connect(server,”HelloPort”,
                client,”HelloPort” )

  port = fwk.lookupPort(client,”GoPort”)
  go = gov.cca.ports.GoPort.GoPort( port )
  go.go()
```

How to Write and Use
Babelized CCA Components

- Define “Ports” in SIDL
- Define “Components” that implement those Ports, again in SIDL
- Use Babel to generate the glue-code
- Write the guts of your component(s)

How to Write A
Babelized CCA Component (1/3)

- Define “Ports” in SIDL
  - CCA Port =
    - a SIDL Interface
    - extends gov.cca.Port

```
version tutorial 1.0;

package tutorial {
    interface Function extends gov.cca.Port {
        double evaluate( in double x );
    }
}
```

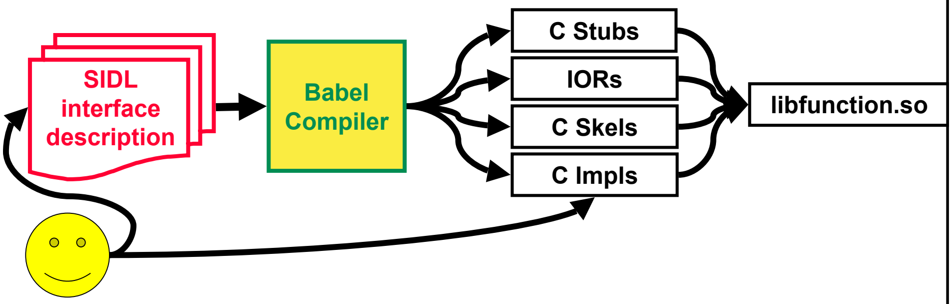
How to Write A Babelized CCA Component (2/3)

- Define “Components” that implement those Ports
  - CCA Component =
    - SIDL Class
    - implements gov.cca.Component (& any provided ports)

```
class LinearFunction implements tutorial.Function,
                                gov.cca.Component {
      double evaluate( in double x );
      void setServices( in cca.Services svcs );
}
```

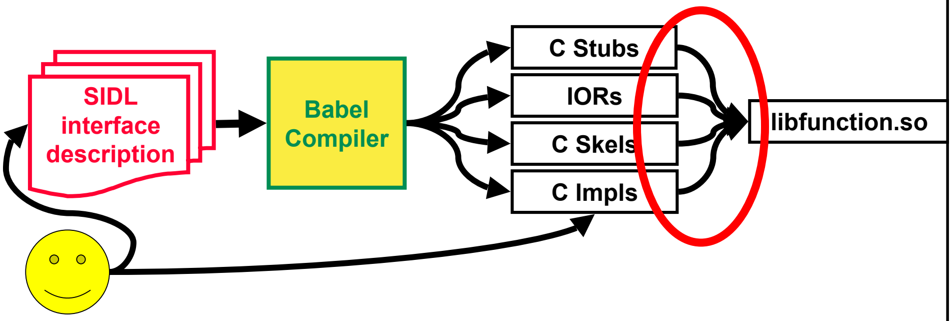
```
class LinearFunction implements-all
       tutorial.Function, gov.cca.Component { }
```

How to Write A Babelized CCA Component (3/3)



- Use Babel to generate the glue code
  - `babel --server=C –Rrepo function.sidl`
- Add implementation Details

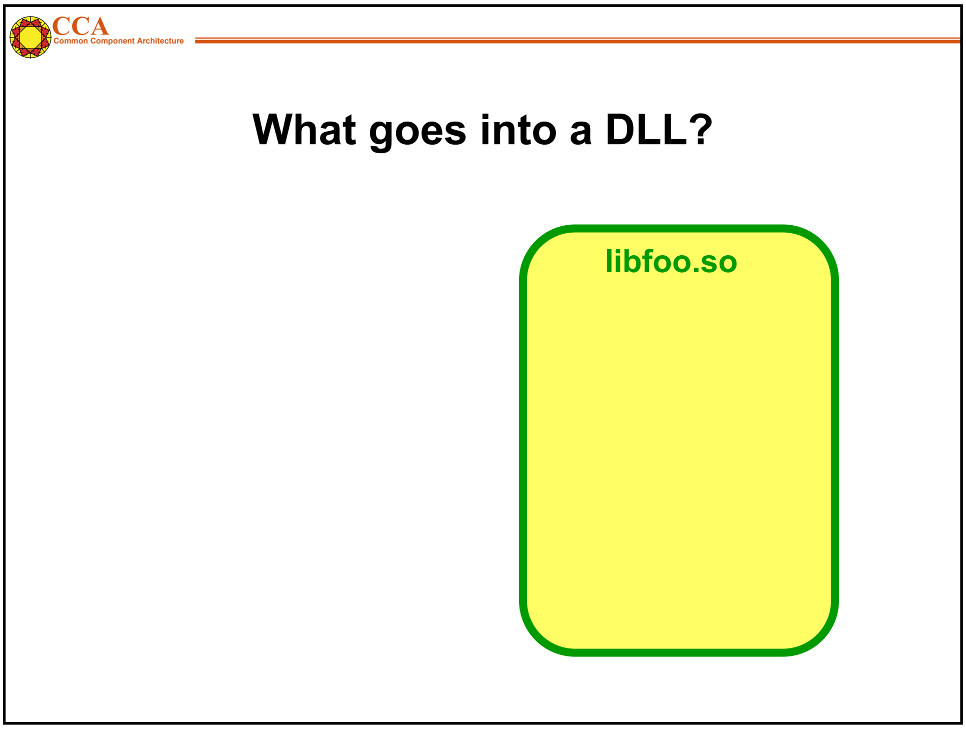
What's the Hardest Part of this Process?



- Properly building dynamically loadable .so files.

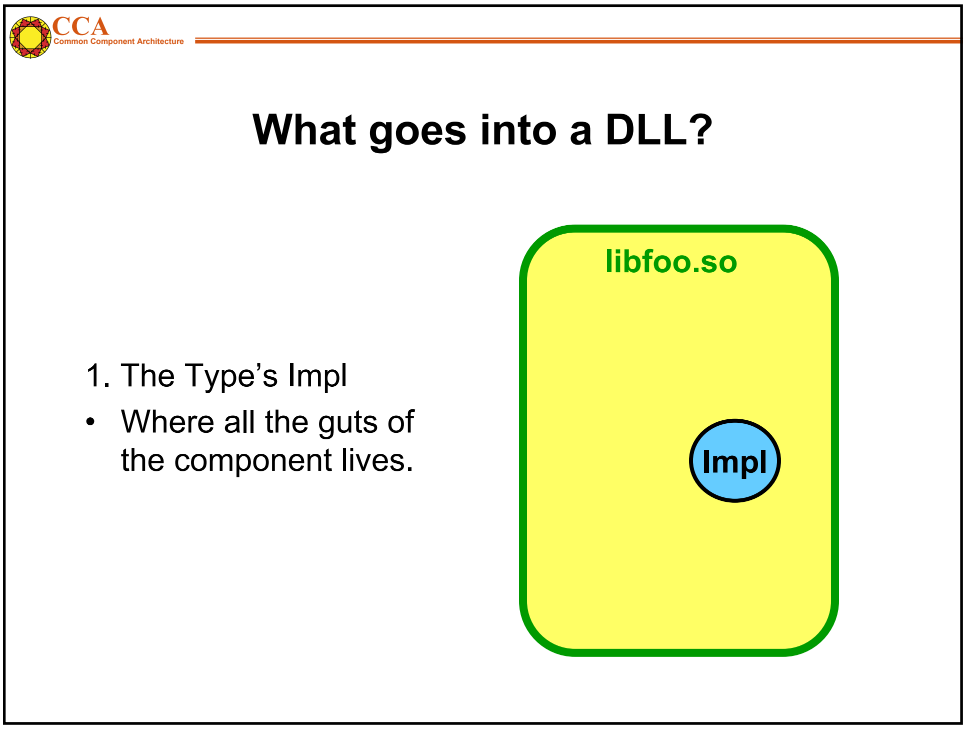
Review of "Linkage"

- Static Linked Libraries (*.a)
  - Symbols are hardcoded
  - Resolved at link-time of application
- Shared Object Libraries (*.so)
  - Symbols are hardcoded
  - Symbols resolved at load time (before main())
- Dynamically Loaded Libraries (*.so) (*.dll in Win32)
  - Symbols are determined at run time (by app code)
  - Symbols resolved at run time (void* dlopen(char*))



What goes into a DLL?

libfoo.so



What goes into a DLL?

libfoo.so

1. The Type's Impl

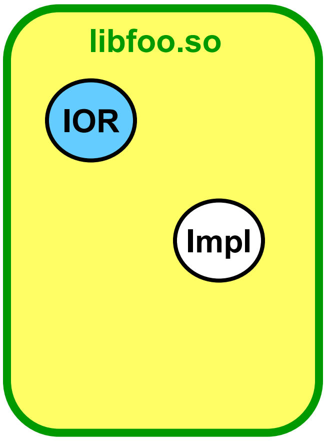
- Where all the guts of the component lives.

Impl

What goes into a DLL?

2. The Type's IOR

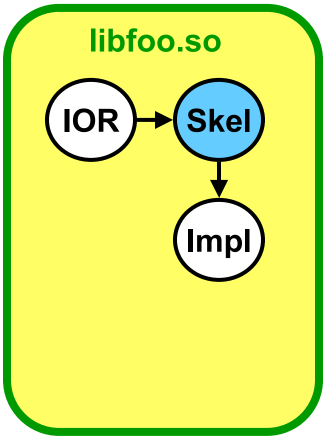
- IORs (Intermediate Object Representation)
- Always implemented in ANSI C
- Babel Object Model is implemented in IOR
- Dynamic Loading is based on symbols in IOR



What goes into a DLL?

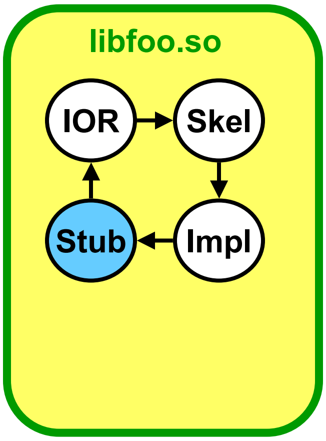
3. The Type's Skel

- IORs depend on the Skels
- Skels translate from ANSI C to Impl language
- Skels call Impls



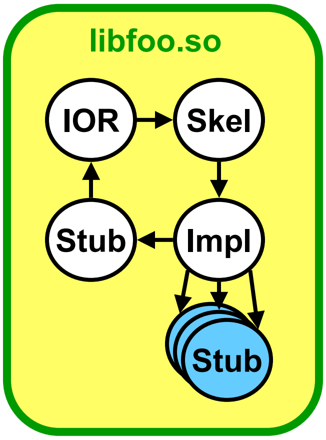
What goes into a DLL?

4. The Type’s Stub
- Impl depends on Stubs
  - class needs to call methods on itself
  - Like “this” pointer in C++
  - self in Python
- Stubs translate from application Language to ANSI C

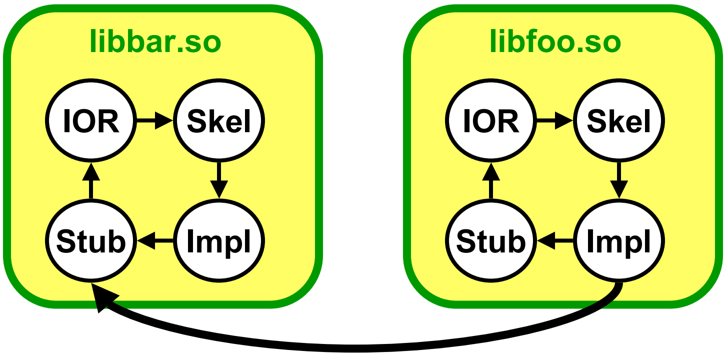


What goes into a DLL?

5. Stubs for all the other types that are
- passed as arguments,
- return values, or
- manipulated internally in the Type’s Impl

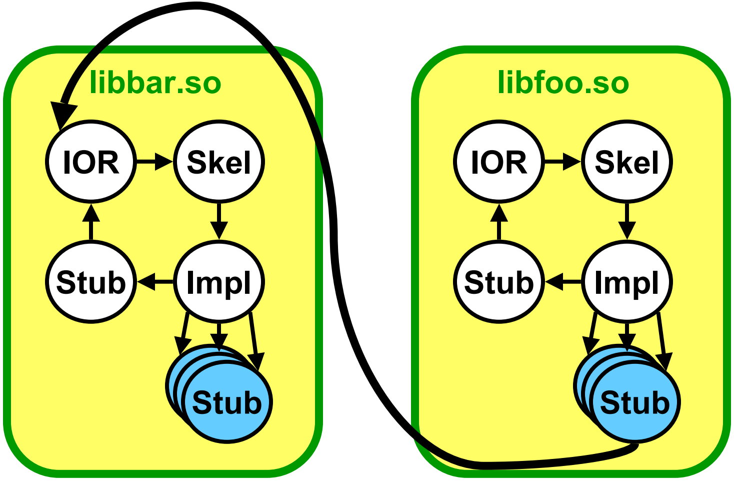


Q: Why not keep each Stub exclusively with its own Impl?

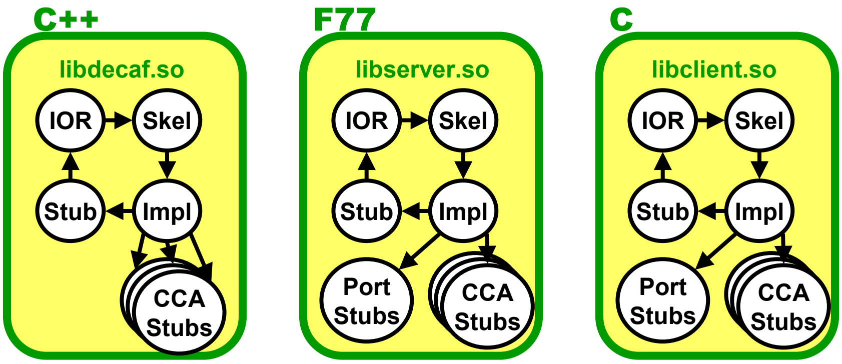


A: Works only if bar_Impl and foo_Impl are implemented in the same language

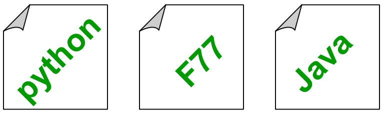
IORs provide a language-independent binary interface



What you'll see with the upcoming "Hello World" demo



And a "main" in any of



Contact Info

- Project: http://www.llnl.gov/CASC/components
  - Babel: language interoperability tool
  - Alexandria: component repository
  - Quorum: web-based parliamentary system
  - Gauntlet (coming soon): testing framework
- Bug Tracking: http://www-casc.llnl.gov/bugs
- Project Team Email: components@llnl.gov
- Mailing Lists: majordomo@lists.llnl.gov

  subscribe babel-users *[email address]*

  subscribe babel-announce *[email address]*

UCRL-PRES-148796

5, July 2002

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48