**RESEARCH LAB 2 - MULTI-LANGUAGE COUPLING**

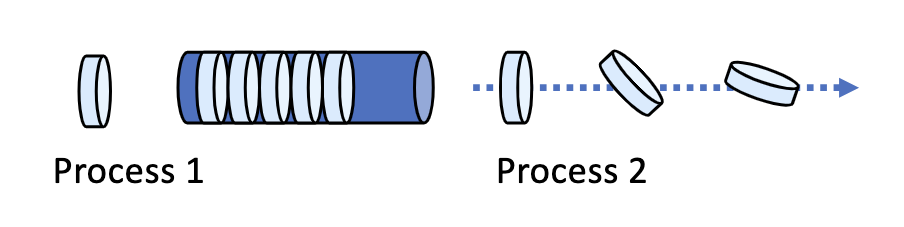
CMPE -275

Hardik ghori (016452059)

Abraham Mathew (016018990)

Badri .Varun raj (015918006)

Team-Blueything



# Multilanguage Coupling-

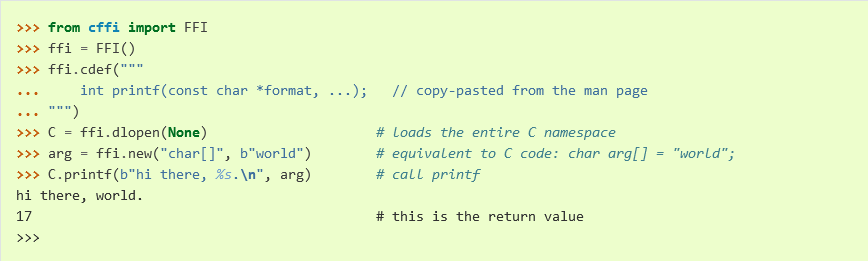
There are two main ways to couple different languages:

1) Using a Foreign Function Interface (FFI)

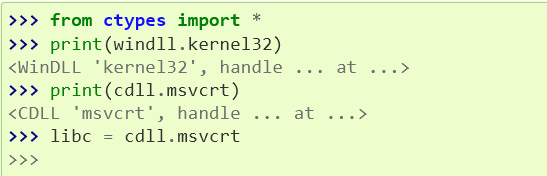
2) Using a Message Passing Interface (MPI)

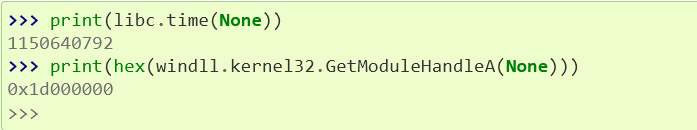
**Foreign Function Interface**

* A Foreign Function Interface (FFI) is a set of tools that allows two different programming languages to interact with each other.
* The FFI allows one language to call functions written in another language. This can be useful for performance-critical code that needs to be written in a low-level language such as C or Fortran, while the rest of the code can be written in a high-level language such as Python or Lua.
* Foreign function interfaces enable building wrapper libraries that provide functionality from a library from another language in the host language, often in a style that is more idiomatic for the language. Most languages have FFIs to C language.
* Eg: CFFI , ctypes
* FFI refers to the ability for code written in one language (the “host language,” such as Java), to access and invoke functions written in another language (the “guest language,” such as Python ). The term “foreign” refers to the fact that the functions come from another language and environment.
* Depending on the language and its FFI support, you might also be able to access global named variables, automatically convert data types between the host and guest languages, and have code in the guest language invoke functions in the host language as callbacks.
* In interpreted languages like Java, it’s usually not possible to use a library’s compile-time features like python preprocessor macros and constants (i.e. things #define in the library headers). This is because the FFI accesses the library’s binary code (e.g. its .so, .dylib, or .dll) directly, without compiling any code.
* However, the FFI support in some compiled languages works by compiling down to C code; in these cases, you may be able to use compile-time features. It all depends on the language and how it implements FFI.
* The main way to use CFFI is as an interface to some already-compiled shared object which is provided by other means

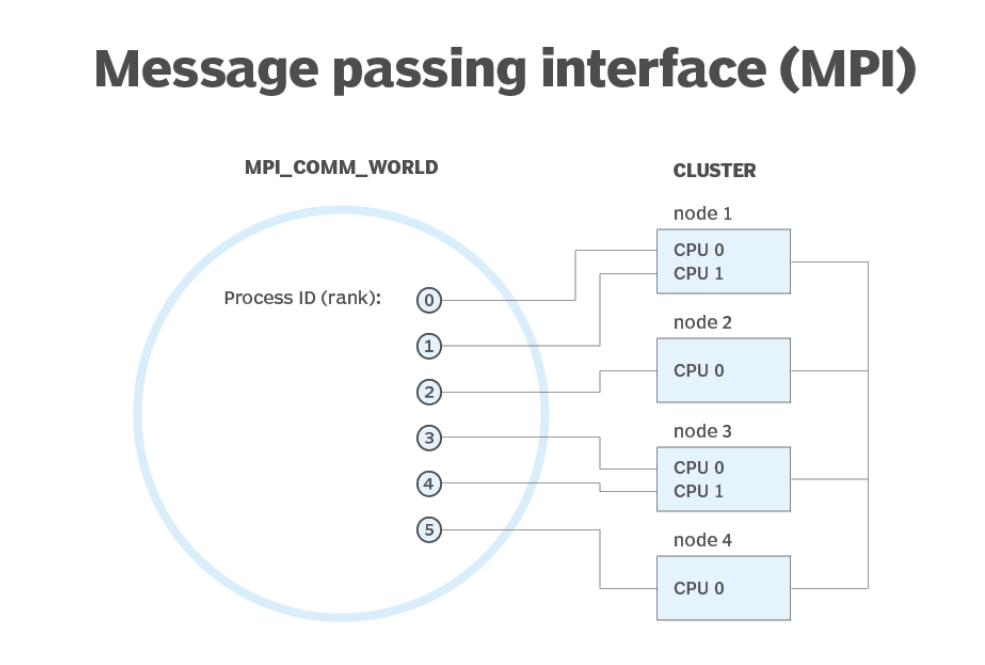


* ctypes provides C compatible data types, and allows calling functions in DLLs or shared libraries. It can be used to wrap these libraries in pure Python.





Message Passing Interface



A Message Passing Interface (MPI) is a set of tools that allows different programming languages to communicate with each other by passing messages. MPI is typically used for parallel programming, where different parts of the code need to communicate with each other in order to work on a shared problem.

There are a number of ways to interface Java and Python code using MPI. One approach is to use the Java Native Interface (JNI) to call Python code from Java. Another approach is to use the Python-Java bridge to call Java code from Python.

* Python-Java bridge:
  + The Python-Java bridge is a technology that allows Python code to call Java code and vice versa. This bridge is an implementation of the JNI (Java Native Interface) that provides a way for Python code to call Java code and for Java code to call Python code.
  + Eg**: javabridge library in python**
* Java Native Interface:
  + The Java Native Interface (JNI) is a technology that allows Java code to call native code written in languages such as C and C++. Native code is code that is written in a language other than Java. The JNI allows Java code to call native code and native code to call Java code.

Eg: **“native” keyword + static libraries to load shared libraries + generating header files**

The message passing interface (MPI) is a standardized means of exchanging messages between multiple computers running a parallel program across distributed memory.

In parallel computing, multiple computers – or even multiple processor cores within the same computer – are called nodes. Each node in the parallel arrangement typically works on a portion of the overall computing problem. The challenge then is to synchronize the actions of each parallel node, exchange data between nodes, and provide command and control over the entire parallel cluster. The message passing interface defines a standard suite of functions for these tasks. The term message passing itself typically refers to the sending of a message to an object, parallel process, subroutine, function or thread, which is then used to start another process.

MPI defines useful syntax for routines and libraries in programming languages including Fortran, C, C++ and Java.

### **Benefits of the message passing interface**

The message passing interface provides the following benefits:

* Standardization. MPI has replaced other message passing libraries, becoming a generally accepted industry standard.
* Developed by a broad committee. Although MPI may not be an official standard, it's still a general standard created by a committee of vendors, implementors and users.
* Portability. MPI has been implemented for many distributed memory architectures, meaning users don't need to modify source code when porting applications over to different platforms that are supported by the MPI standard.
* Speed. Implementation is typically optimized for the hardware the MPI runs on. Vendor implementations may also be optimized for native hardware features.
* Functionality. MPI is designed for high performance on massively paralle machines and clusters. The basic MPI-1 implementation has more than 100 defined routines.

Wrapper and Binding

* There are a few different ways to couple different languages together.
* A wrapper is a piece of code that provides a layer of abstraction between two pieces of code. This allows the two pieces of code to communicate without being directly dependent on each other.
* A binding is a way of connecting two pieces of code so that they can call each other's functions and procedures. An API (Application Programming Interface) is a set of rules and standards that define how two pieces of software can interact with each other.

A wrapper is a bit of code that sits on top of other code to recycle it's functionality but with a different interface. This usually implies an interface written in the same language. It should also be noted that sometimes people will say wrapper when what they technically mean is a binding (myself included).

Pros:

* It's in the same language as the original
* Wrappers enhance or reuse functionality without needing a full rewrite.
* Relatively quick to accomplish
* Trivial updates when the source library changes. You'll probably only need to bind new functions unless they broke backwards compatibility by changing expected input/outputs of functions/classes.

Cons:

* Wrapping an entire library can be extremely repetitive

A binding is another bit of code that sits on top of other code to recycle it's functionality except this time bindings are written in a language different than the thing they bind. A notable example is PyQt which is the python binding for QT.

Pros:

* Bring functionality from another language into the language of your choice.
* Relatively fast in comparison to a port
* Same level of trivial changes are needed as in wrapping- You'll probably only need to wrap new functions/classes unless they broke backwards compatibility by changing expected input/outputs of functions/classes.

Cons:

* Just as repetitive as a wrapper
* You're probably taking a fairly large performance hit, especially any wrapper involving an interpreted language on either end

Some technologies that exist for language coupling are:

* The C/C++ to Fortran interface
* The Lua to Python interface
* The Service-Oriented Architecture (SOA)

some tools that may be helpful for this task include Jython (a Python implementation for the Java platform) and Py4J (a Python library for accessing Java objects from Python).

C/C++ to Fortran:

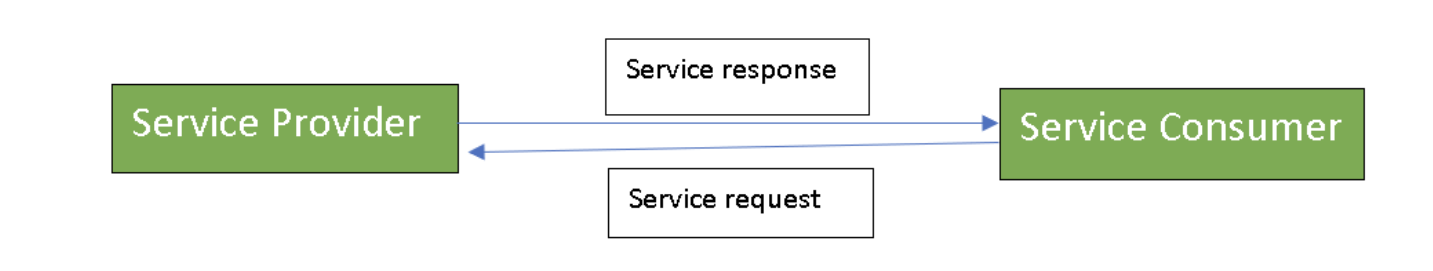
* The C/C++ to Fortran interface is a classic example of language coupling. It allows C/C++ code to call Fortran code, and vice versa. This is done by using a Fortran compiler that can generate C/C++ code, and by using a C/C++ compiler that can generate Fortran code.   
  Order of multi dimensional arrays in C/C++ is the opposite of FORTRAN.

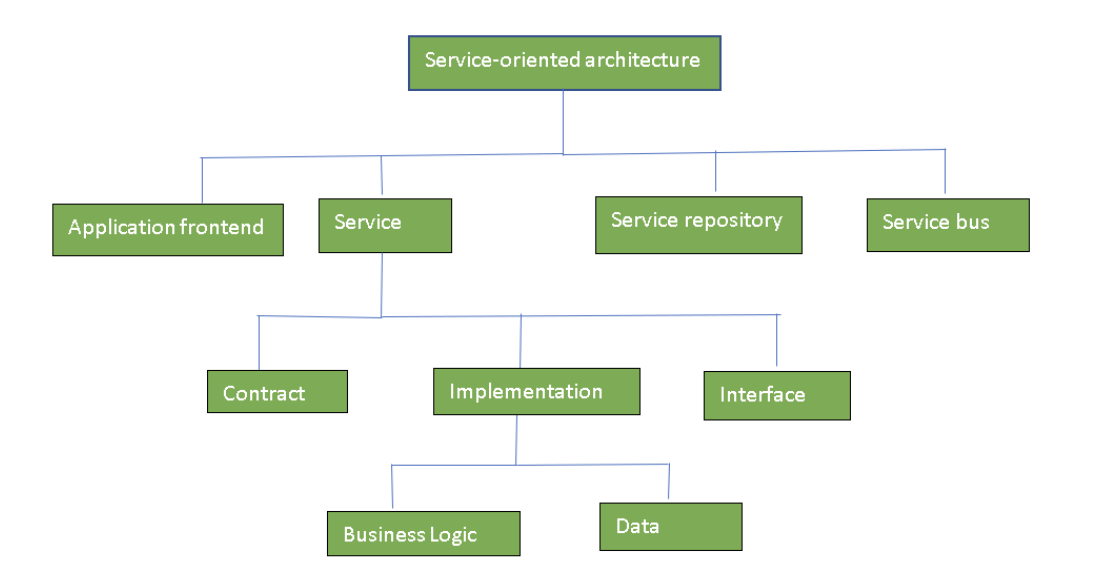
Lua to Python:

* The Lua to Python interface is a modern example of language coupling. It allows Lua code to call Python code, and vice versa. This is done by using a Lua interpreter that can execute Python code, and by using a Python interpreter that can execute Lua code.
* how without need of performance or bandwidth taxes. This is possible because the Lua interpreter is written in C, which is a language that can interface with Python. This means that the two languages can share data and code without having to go through an intermediate language.

Service-Oriented Architecture:

* Service-oriented architecture (SOA) is a software development model that allows services to communicate across different platforms and languages to form applications.
* The Service-Oriented Architecture (SOA) is a way of designing software that is based on services. Services are self-contained units of functionality that can be accessed over a network. A service-oriented system is made up of a number of services that can be combined to form a complete system



**applications of SOA:** SOA is used in many ways around us whether it is mentioned or not. SS

Service Oriented Architecture Applications

* SOA infrastructure is used by many armies and air forces to deploy situational awareness systems.
* SOA is used to improve healthcare delivery.
* Nowadays many apps are games and they use inbuilt functions to run. For example, an app might need GPS so it uses the inbuilt GPS functions of the device. This is SOA in mobile solutions.
* SOA helps maintain museums a virtualized storage pool for their information and content.