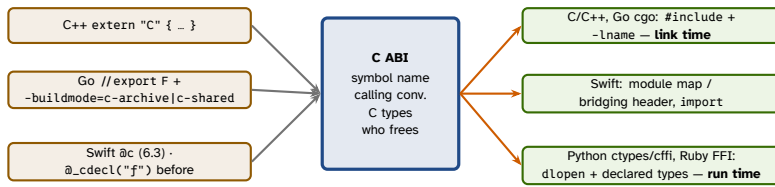


In one line: Languages call each other through the **C ABI**: an unmangled symbol, the platform's C calling convention, C types (scalars, pointers, plain structs). One side **exports** C-looking functions (extern "C", Go //export, Swift @c); the other **links** them (-l, cgo, module map) or **dlopens** them and declares the types (ctypes, Ruby FFI). Every allocation keeps *one* owner.

1 exporters → C ABI → consumers



does not cross: classes, exceptions/panics, generics, GC'd objects, Swift/Go strings

2 why extern "C"

```

void PrintHello(const char *u);

compiled as C++ (Itanium ABI) the symbol is
_Z10PrintHelloPKc
_Z·10+name·P K c = ptr to const char

— overloads need it; C, cgo, dlsym can't guess it.

inside extern "C": C linkage, plain name
PrintHello (Mach-O adds _: _PrintHello)

see it: nm libexample.a -demangle: c++filt
  
```

Pick a strategy

host · library	strategy (e.g.)
compiled · compiled	link a library via the C ABI (Go → C++, Swift → C)
compiled · interpreted	embed the interpreter, run scripts (CPython C API, Lua)
interpreted · compiled	FFI or a native extension (ctypes, cffi, Ruby FFI)
interpreted · interpreted	2 processes + IPC (pipes, sockets, gRPC) or one C lib

How it works — libraries and the linker

- Static** .a = archive of .o; the linker copies in only objects that resolve undefined symbols, searching each archive **once**, in **command-line order** (-lfoo goes *after* its users).
- Dynamic** .so/.dylib/.dll: the loader maps + binds it at launch; dlopen+dlsym at run time = what ctypes/FFI do.
- ar rcs libexample.a ex.o clang++ -shared -fPIC (macOS -dynamiclib). Both in one dir: GNU ld takes the .so.

Example — one header for C, C++ and cgo

```

#pragma once
#ifdef __cplusplus
extern "C" {           // C++ sees C linkage; C never sees this
#endif
void PrintHello(const char *u);      // the ';' the note lost
char *MakeGreeting(const char *name); // malloc'd ...
void FreeGreeting(char *p);          // ... freed HERE
#ifdef __cplusplus
}
#endif
  
```

```

// #cgo LDFLAGS: -L${SRCDIR} -lexample
// #include <stdlib.h>
// #include "example.h"
import "C"           // the comment right above is the preamble
import "unsafe"
func main() {
    s := C.CString("Hi")           // malloc'd C copy
    defer C.free(unsafe.Pointer(s)) // Go frees it
    C.PrintHello(s)
}
  
```

Go as the library

```

//export SayHello
func SayHello(s *C.char) { fmt.Println(C.GoString(s)) }
func main() {} // package main required; never called
// go build -buildmode=c-shared -o libhello.so (c-archive: .a)
  
```

Only //exported functions are callable; the Go runtime (GC, scheduler) ships inside; a C header is generated beside it (*unverified*). With //export, the preamble may only *declare* (it is copied into two C files).

Python and Ruby — load at run time

```

lib = CDLL("./libexample.so")           # dlopen
lib.PrintHello.argtypes = [c_char_p]
lib.PrintHello(b"Hi")                   # bytes -> char*; a str -> wchar_t*
lib.MakeGreeting.restype = c_void_p     # default int truncates
p = lib.MakeGreeting(b"Ann"); print(c_char_p(p).value) # copy
lib.FreeGreeting(c_void_p(p))           # the lib's own free
  
```

```

module Ex; extend FFI::Library; ffi_lib './libexample.so'
  attach_function :PrintHello, [:string], :void # ret: required
end; Ex.PrintHello("Hi")
  
```

CDLL **releases the GIL** for the call; cffi parses C headers.

Swift at the border

- In:** module map (header "example.h" link "example").
- Out:** @c / @c(MyLib_f) (6.3) → generated header; @c @implementation implements a C-declared fn; earlier @cdecl.
- Swift→Swift is **not** FFI: -emit-library = .dylib/.so (-static: .a) + .swiftmodule — same compiler only.

Memory ownership

- Whoever allocates, frees** (the lib exports FreeGreeting).
- Go:** C.CString mallocs (you C.free); C keeps no Go pointer past the call unless pinned (runtime.Pinner).
- GC'd side:** keep ctypes CFUNCTYPE callbacks referenced (else GC → crash); a withCString pointer dies with its closure. C++ exceptions: catch in the wrapper (*unverified*).

Interview traps

- !** No extern "C" → *undefined reference to PrintHello*; an ungarded extern "C" in a header breaks C/cgo.
- !** ctypes: no restype truncates a 64-bit pointer to int; a str arrives as wchar_t* — C prints "H".
- !** "Statically linked" with a .so/.dylib beside the .a: links, then fails to launch elsewhere (-Bstatic).

Remember

Speak C at the border: plain names, C types, one owner.

Likely questions

- Why C? — stable platform ABI, no mangling, no runtime.
- extern "C"? — plain symbol, C convention, no overloads.
- Static vs dynamic? — copied at link vs mapped by the loader.