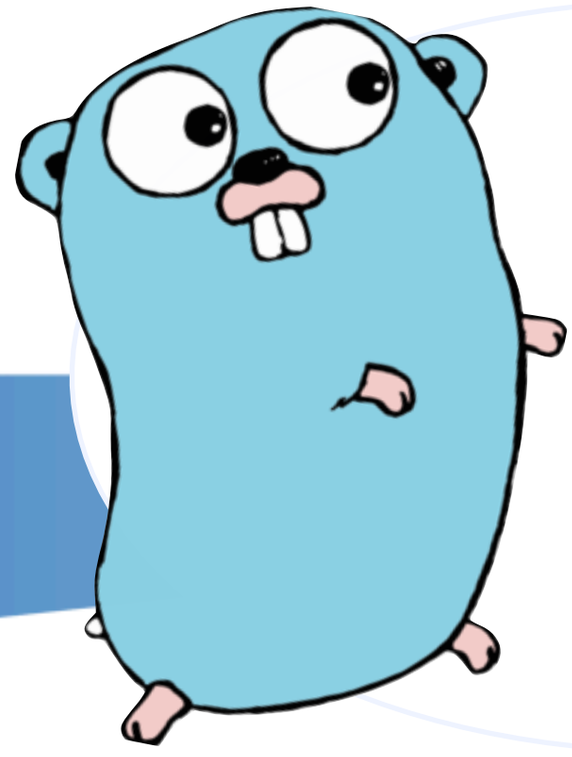
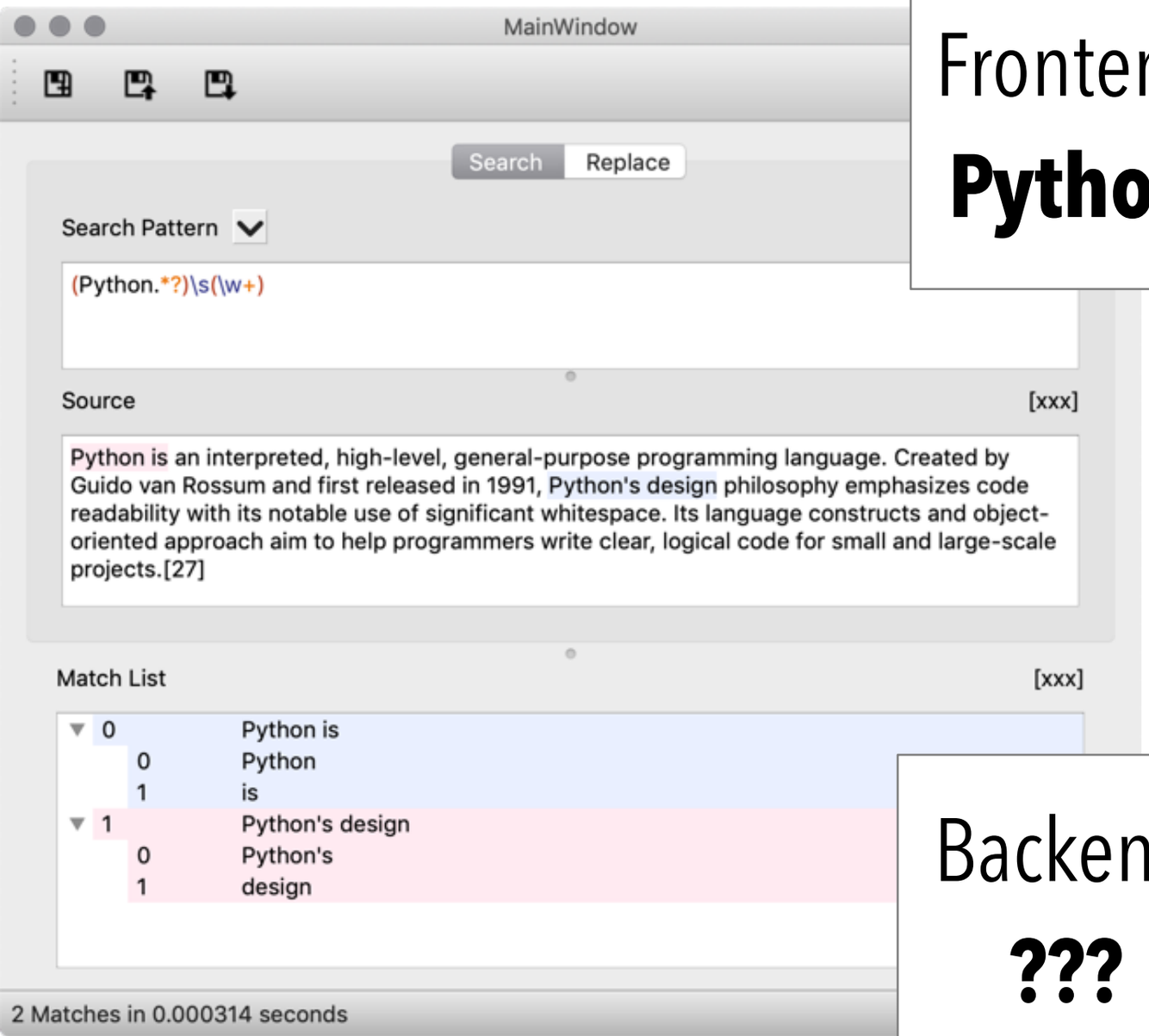
Go(lang) to Python

Europython 2019 Basel, Stefan Baerisch

stefan@stbaer.com



Why this Talk?



Frontend

Python

- Good Qt Bindings
- Very Expressive / Productive
- Fast Enough
- Good For Experiments
- Good Library Support
- Good Cross Plattform Support
- Low “Frustration” Factor

Backend

???

- Expressive Enough
- Good Concurrency / Parallelism
- Good Performance

Nice Things about Go

Go has:

Very Quick Compiler

Lightweight, Managed Go-Routines — No Sync/ASync

Automatic Memory Management

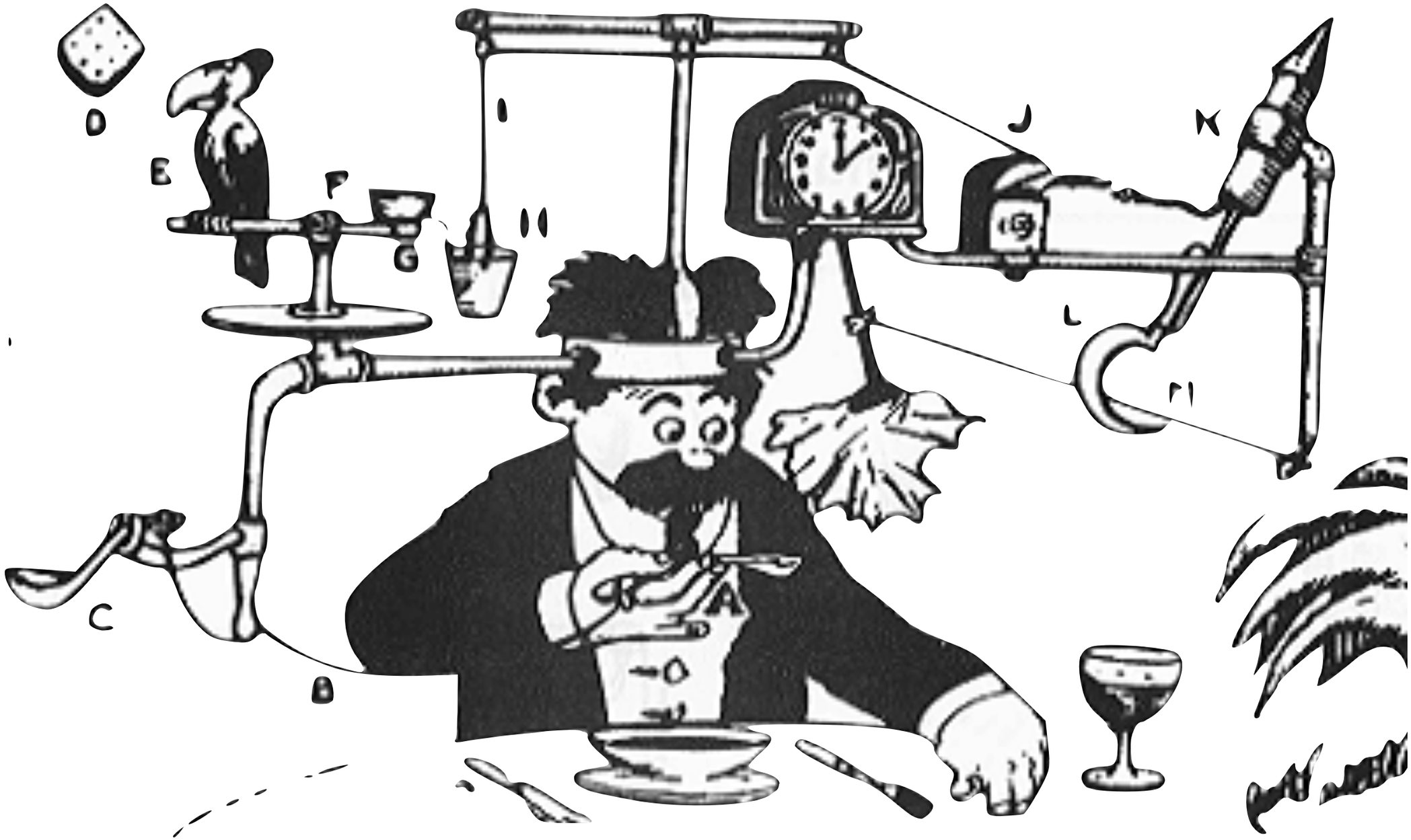
Quite Simple Language

("The Programming language for people that don't like programming languages")

Why Go with Python? Not {C++ Rust} ?

	Python	Go	Cpp / Rust
Performance / Concurrency / Parallelism	Ok	Good	Great
Language Expressiveness	Great	Good	Great
Low Friction	Great	Good	Ok
<Your Factor Here>	?	Likely Good	?

Some Spoilers



- Things work, but are currently far from elegant.
- They are, however, quite interesting

Image: Professor Butts and the Self-Operating Napkin (1931), Wikipedia, Public Domain

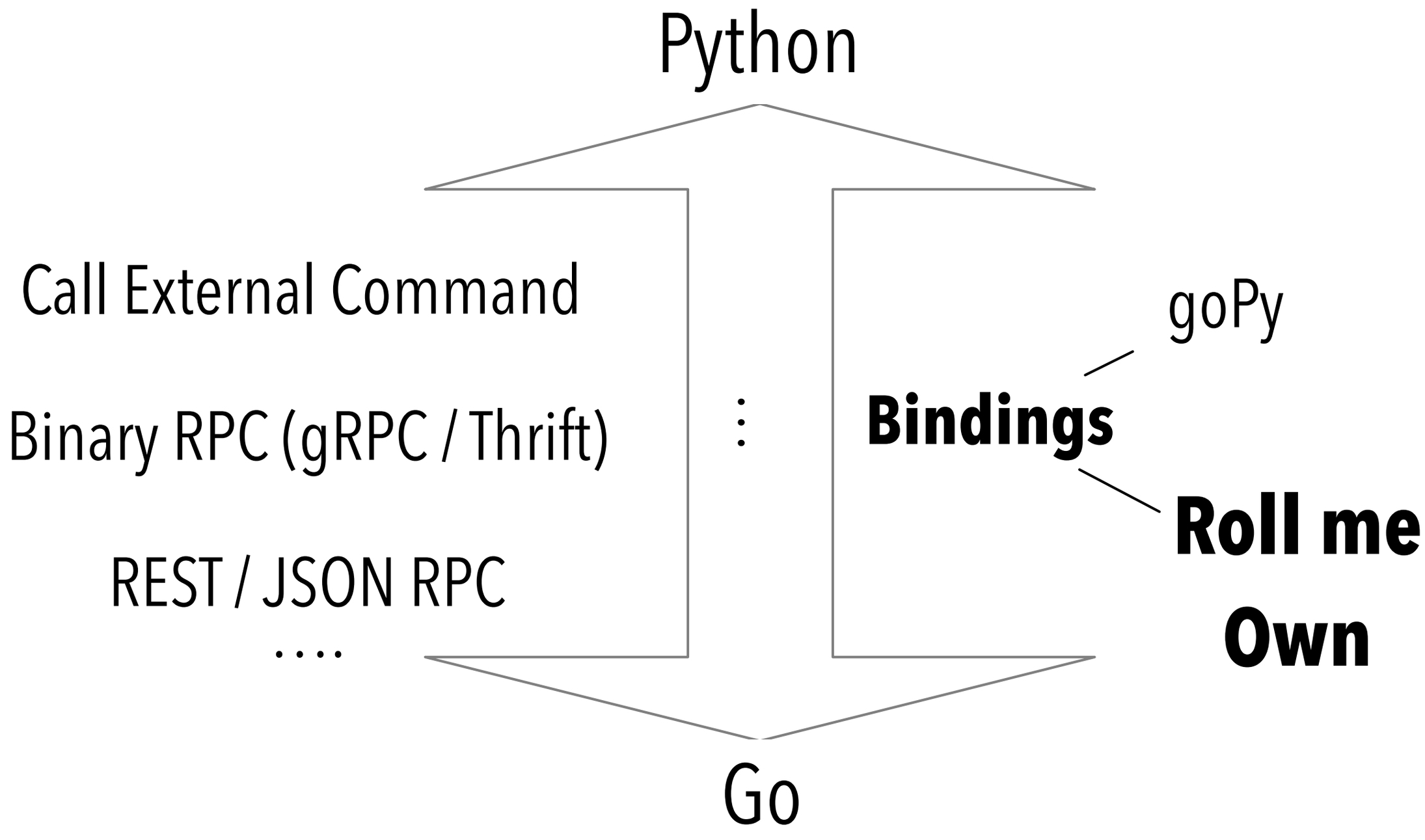
Wrapping Go - The Basics

Options for Using Go from Python

Questions...

Same Process?
Transport Format

Published Api?
Transport Mechamisn



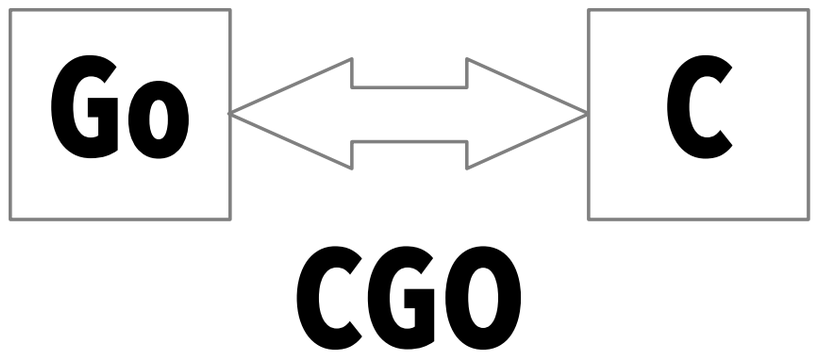
Wrapping Go with CGO

CGO makes bindings possible…

Go Calling Convention

Go Segmented Stacks

Go Routines,
scheduled by runtime



C-Calling Convention

Fixed-size Stacks

Threads

CGO has some drawbacks…

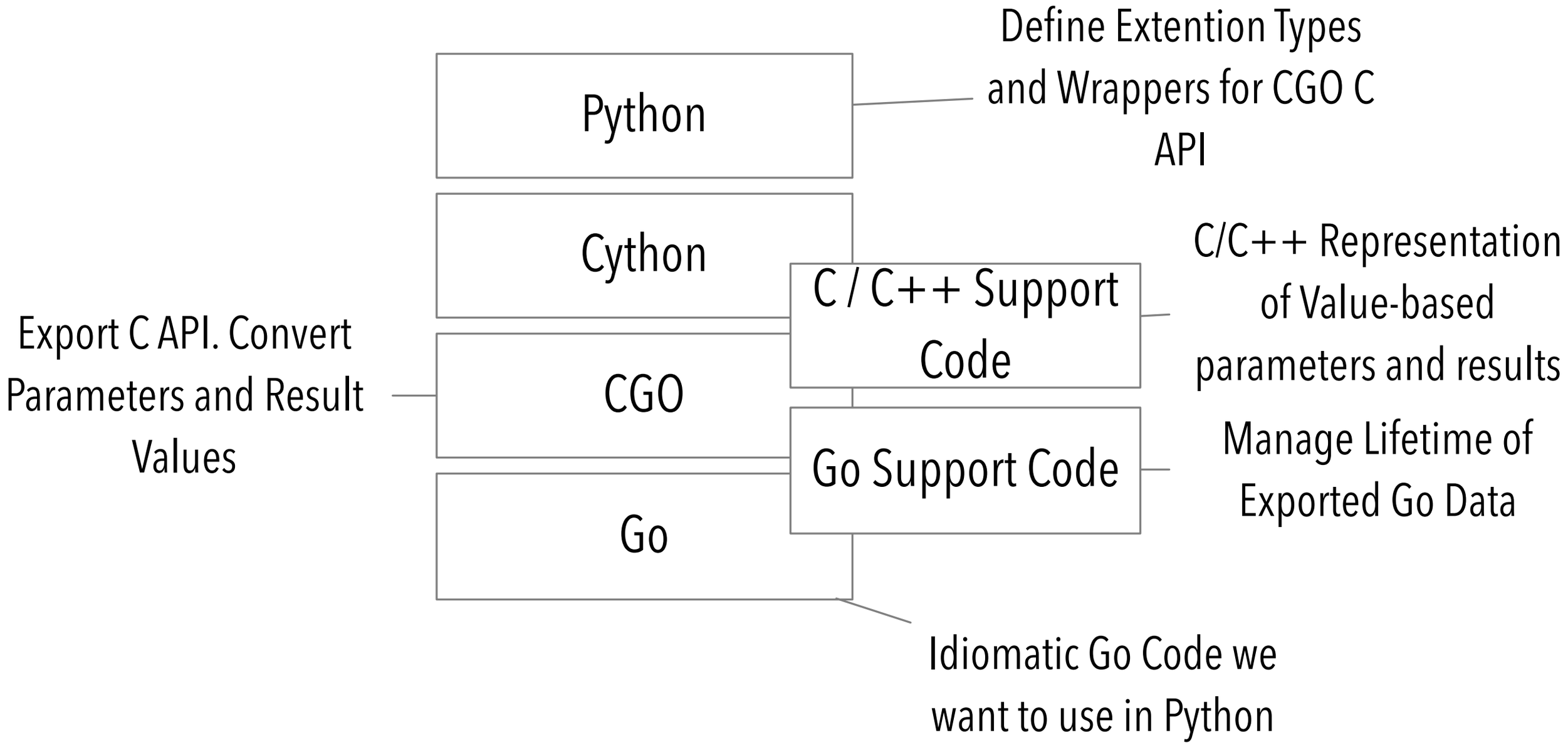
Slower Build Times

More Complex Builds

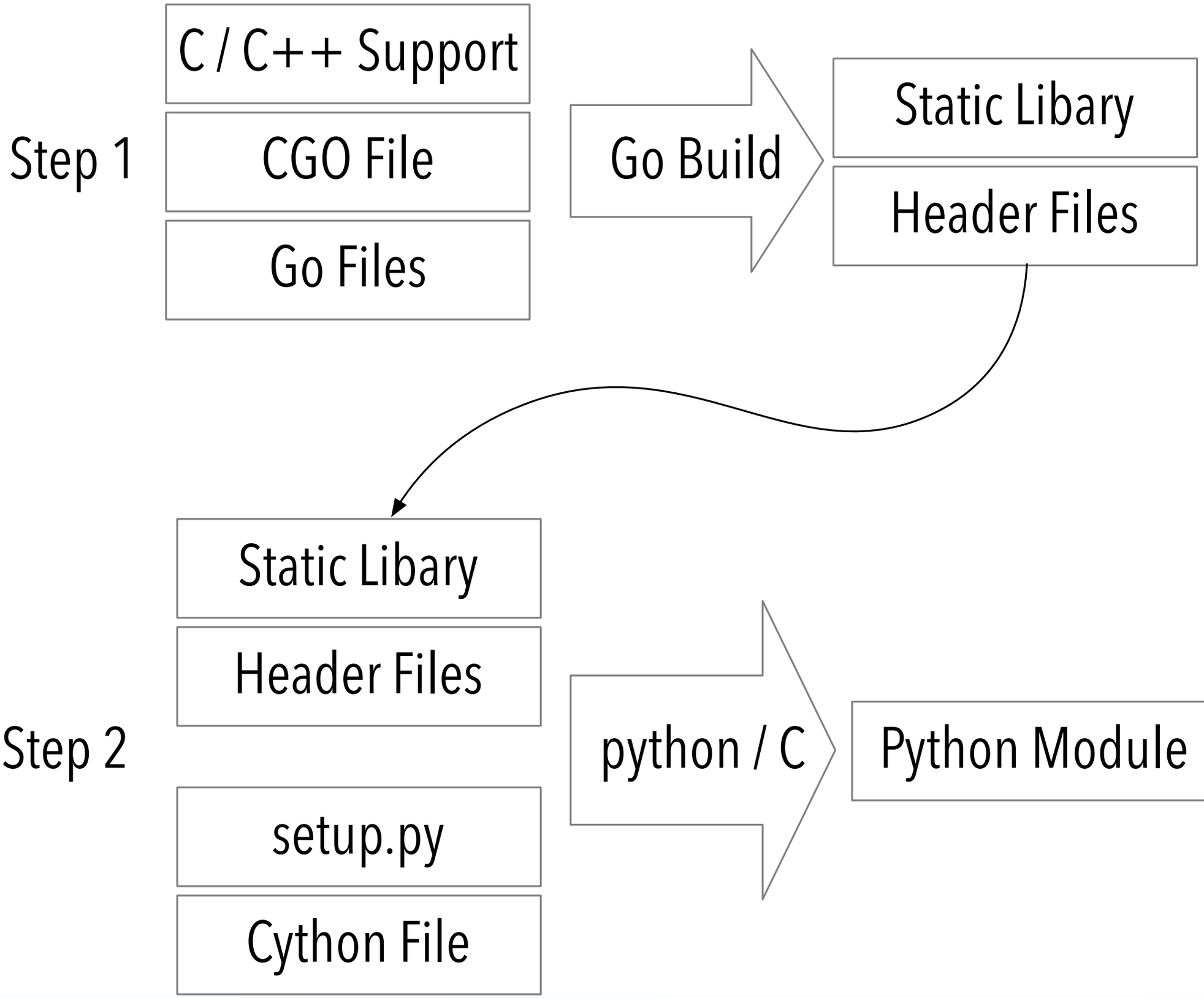
When calling Go from C
- Significant Overhead
- Restrictions in Sharing Data

https://golang.org/cmd/cgo/ https://dave.cheney.net/2016/01/18/cgo-is-not-go

High Level Architecture



Build Process



Wrapping a Simple Function

Simple Function - Goals

Go — Code we want to use

```go
func Add(v, v2 int) int {}
```

CGO — Provide C API

```go
//export cgo_Add
func cgo_Add(cgo_v1, cgo_v2 C.int) C.int {}
```

Cython — Wrap C API

```
cdef extern from "cgo_lib/cgo_lib.h":
    int cgo_Add(int p0, int p1)

cpdef int add(int v1, int v2):
```

Python — Use Functionality

```
>>> cgo_cython.add(1,2)
3
```

Simple Function - Go Code

Function name, exported if it starts with capital letter

Parameters. name(s), then type

Result, can be multiple values, similar to tuples

```
func Add(v, v2 int) int {
  return v + v2
}
```

Simple Function - CGO

make available in C lib

C parameter / result types

Type conversation

```
//export cgo_Add
func cgo_Add(cgo_v1, cgo_v2 C.int) C.int {
    var v1 int = int(cgo_v1)
    var v2 int = int(cgo_v2)
    var result int = main_lib_simple.Add(v1,v2)
    var cgo_result C.int = C.int(result)
    return cgo_result
}
```

Actual work

Build the Package

Name of C Lib / Header

Build Library

```
go build -o ../../../cgo_lib/cgo_lib.a -buildmode=c-archive
```

Excerpt from the Header File

CGO generated C Header

```c
#ifndef GO_CGO_PROLOGUE_H
#define GO_CGO_PROLOGUE_H

typedef signed char GoInt8;
typedef unsigned char GoUint8;
typedef short GoInt16;
typedef unsigned short GoUint16;
typedef int GoInt32;
typedef unsigned int GoUint32;
typedef long long GoInt64;
typedef unsigned long long GoUint64;
typedef GoInt64 GoInt;
typedef GoUint64 GoUint;
typedef __SIZE_TYPE__ GoUintptr;
typedef float GoFloat32;
typedef double GoFloat64;
typedef float _Complex GoComplex64;
typedef double _Complex GoComplex128;

/*
  static assertion to make sure the file is being used on architecture
  at least with matching size of GoInt.
*/
typedef char _check_for_64_bit_pointer_matching_GoInt[sizeof(void*)==64/8 ? 1:-1];

#ifndef GO_CGO_GOSTRING_TYPEDEF
typedef _GoString_ GoString;
#endif
typedef void *GoMap;
typedef void *GoChan;
typedef struct { void *t; void *v; } GoInterface;
typedef struct { void *data; GoInt len; GoInt cap; } GoSlice;
```

Our Function

```c
#ifdef __cplusplus
extern "C" {
#endif

extern int cgo_Add(int p0, int p1);

#ifdef __cplusplus
}
#endif
```

Setup.py for Cython

```
from distutils.core import setup
from Cython.Distutils import build_ext, Extension

libname = "cgo_lib/cgo_lib.a"
extention_name = "cgo_cython"
setup(
    name=extention_name,
    ext_modules=[
        Extension(extention_name,
                  sources=[extention_name + ".pyx"],
                  include_dirs=["cgo_lib"],
                  libraries=[libname ],
                  language="c++",
                  extra_objects=[libname],
                  )
    ], cmdclass={'build_ext': build_ext},
)
```

Our C Library

```
python setup.py build_ext --inplace
```

— Build Command

Simple Function - Cython

```
cdef extern from "cgo_lib/cgo_lib.h":
    int cgo_Add(int p0, int p1)
```

Declare Function

```
cpdef int add(int v1, int v2):
    return cgo_Add(v1, v2)
```

Wrapper Function

```
>>> import cgo_cython
>>> cgo_cython.add(1,2)
3
```

Using it in Python

Wrapping User Defined Types

User Defined Types in Go and Python

Python		**Go**
Classes / Extention Types	⟺	User Defined Types
Methods	⟺	Exported Methods
Constructors / Factoryfunctinos	⟺	Functions
Destructor (__del__, __dealloc____,…)	⟺	Garbage Collection
Exceptions	⟺	Error Result Values
<ignored for now>	⟺	Interfaces, Channels

Wrapping User Types - Go Code

```go
type Person struct {
  firstname, lastname string
  age                 uint
  friends             []*Person
}

func NewPerson(firstname string, lastname string, age uint) *Person {...}

func (p *Person) String() string {...}

func (p *Person) Age() uint {...}

func (p *Person) Firstname() string {...}

func (p *Person) Lastname() string {...}
```

Strings

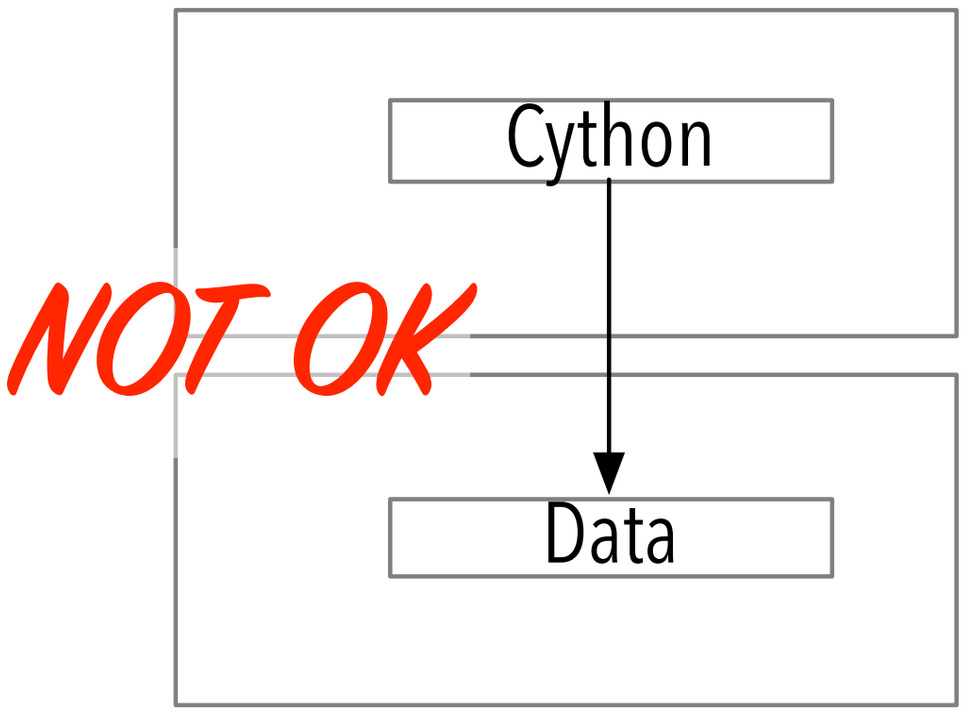
Object Pointer / Reference

Methods

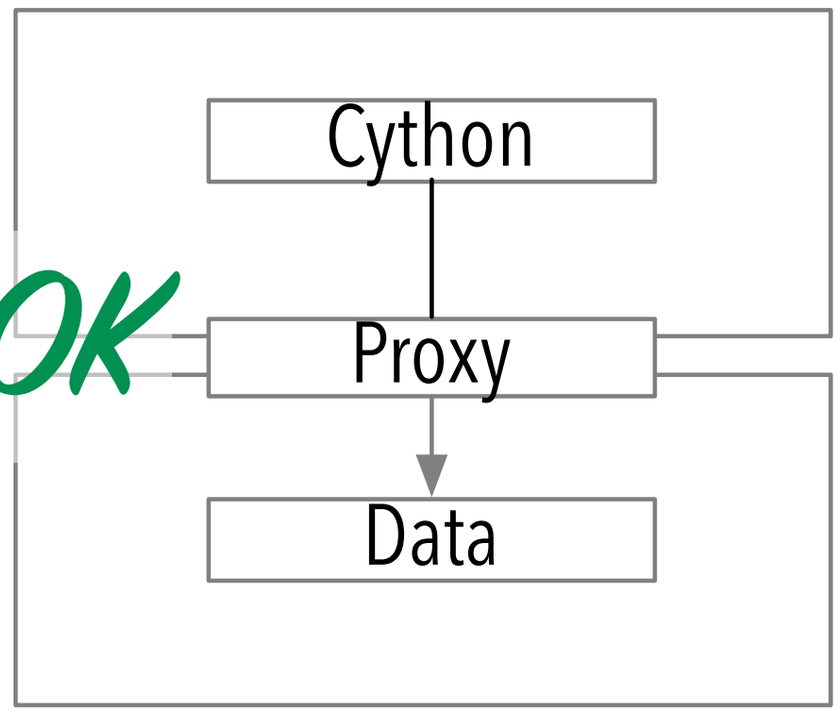
Wrapping User Types - References

How to use Data References in Python?

Not Possible: Direct Sharing of Go Data

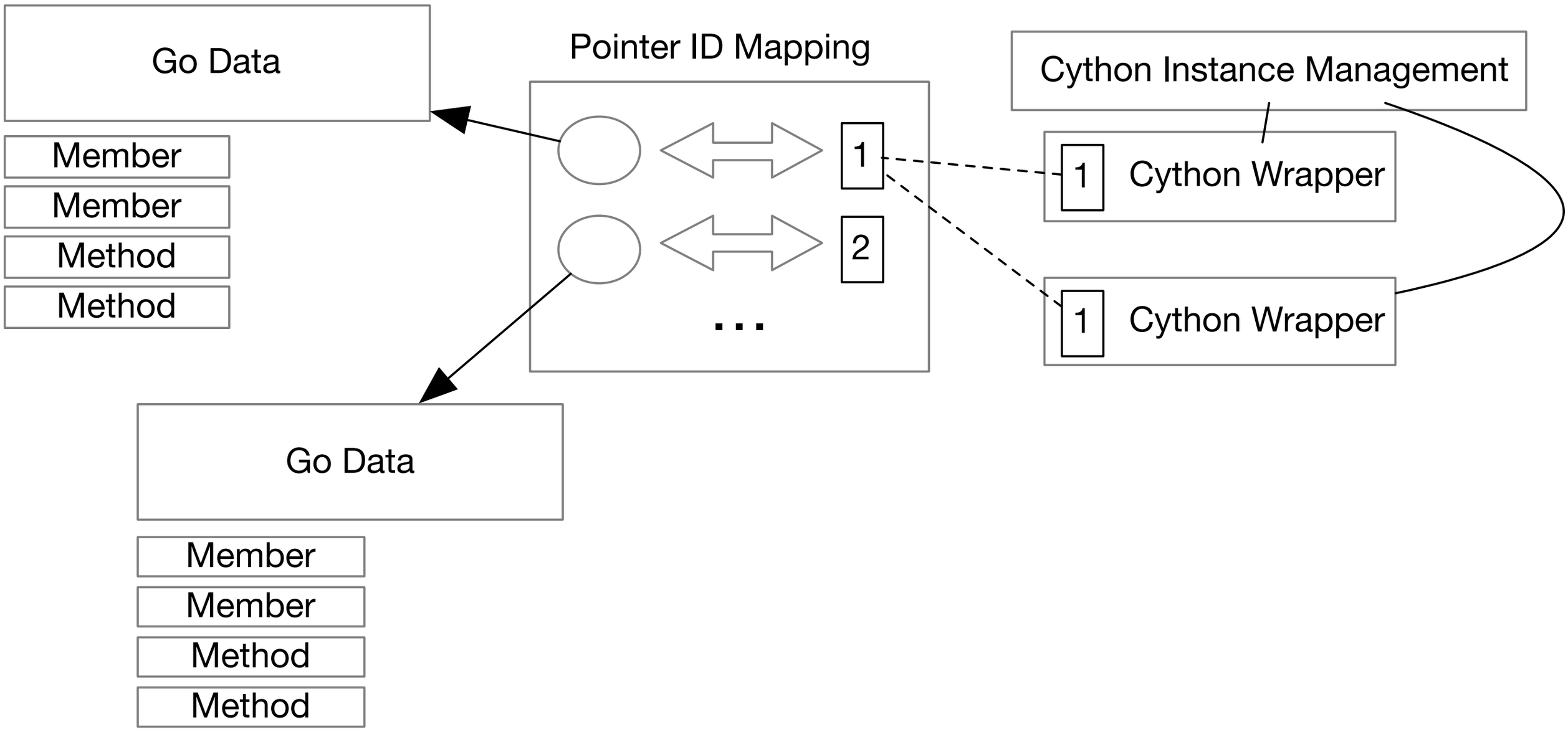


Possible: Using a Proxy



Jobs of the Proxy

- Keep the Go Garbage Collector away from the data shared with Python
- Translate the Go Pointer / Value into something we can share with Python


Go Data
Member
Member
Method
Method
Pointer ID Mapping
1
2
...
Cython Instance Management
1
Cython Wrapper
1
Cython Wrapper
Go Data
Member
Member
Method
Method

Wrapping User Types - References

Id <=> Pointer Mapping Functions

```
func PersonIdForObject(object *main_lib_simple.Person) uint64 {...}

func ObjectForPersonId(oid uint64) *main_lib_simple.Person {...}

func RemovePersonObjectId(id uint64) {...}

type ObjectStorePerson struct {
  obj2int map[*main_lib_simple.Person]uint64
  int2obj map[uint64]*main_lib_simple.Person
  counter uint64
  mutex   *sync.Mutex
}
```

— Bookkeeping

```
var objectStorePersonInstance = NewObjectStorePerson()
```

Package Level Variable

Wrapping User Types - CGO

"Constructor"

```
//export cgo_NewPerson
func cgo_NewPerson(
  cgo_firstname *C.char,
  cgo_lastname *C.char,
  cgo_age C.uint) C.ulong {
  // Convert Parameters
  firstname := C.GoString(cgo_firstname)
  lastname := C.GoString(cgo_lastname)
  age := uint(cgo_age)

  // Call Go Functions
  person := main_lib_simple.NewPerson(firstname, lastname, age)

  //Convert results, here, Golang Pointer to Reference
  var oid uint64 = cgo_lib_simple.PersonIdForObject(person)
  var cgo_oid C.ulong = C.ulong(oid)
  return cgo_oid
}
```

Convert Parameters / Results

"Destructor"

```
//export cgo_DeletePerson
func cgo_DeletePerson(cgo_oid C.ulong) {
  var oid uint64 = uint64(cgo_oid)
  cgo_lib_simple.RemovePersonObjectId(oid)
}
```

Go Function Call

Wrapping User Types - CGO

Convert Parameters

Convert Parameters / Results

```
//export cgo_Person_Firstname
func cgo_Person_Firstname(cgo_oid C.ulong) *C.char {
  var oid uint64 = uint64(cgo_oid)
  var person *main_lib_simple.Person = cgo_lib_simple.ObjectForPersonId(oid)
  var result string = person.Firstname()
  var cgo_result *C.char = C.CString(result) // allocates, needs free() by caller
  return cgo_result
}
```

Get Object

Convert Results

Wrapping User Types - Headers

CGO generated Headers

```
extern long unsigned int cgo_NewPerson(char* p0, char* p1, unsigned int p2);

extern void cgo_DeletePerson(long unsigned int p0);

extern char* cgo_Person_Firstname(long unsigned int p0);
```

Cython Function Declaration

```
cdef extern from "cgo_lib/cgo_lib.h":
    long unsigned int cgo_NewPerson(char*p0, char*p1, unsigned int p2)
    void cgo_DeletePerson(long unsigned int p0)
    char*cgo_Person_Firstname(long unsigned int p0)
```

User Types - Cython Constructor

```
cdef class Person:
    _KNOWN = {}
    cdef unsigned long oid
    cdef object __weakref__

    def __cinit__(self, str firstname = None, str lastname = None,
                  unsigned int age = 0, long oid = -1):
        cdef char*cfirstname
        cdef char*clastname
        if (firstname is not None) and (lastname is not None) and (oid == -1):
            py_byte_string = firstname.encode("UTF-8")
            cfirstname = py_byte_string
            py_byte_string = lastname.encode("UTF-8")
            clastname = py_byte_string
            self.oid = cgo_NewPerson(cfirstname, clastname, age)
        else:
            self.oid = oid
        Person._KNOWN[self.oid] = weakref.ref(self)

    def __dealloc__(self):
        cgo_DeletePerson(self.oid)
```

Manage Objects

Only Member - ID of the underlying Go objects

Convert Parameters

Request new or register existing object

Register Object

Cython Method

Get Name as C String from Go, via CGO

```
cpdef str firstname(self):
    cdef char*cresult = cgo_Person_Firstname(self.oid)
    cdef str result = tounicode_with_free(cresult)
    return result
```

```
cdef str tounicode_with_free(char*s):
    try:
        return s.decode('UTF-8', 'strict')
    finally:
        free(s)
```

Python Again

```
>>> import cgo_cython
>>> john = cgo_cython.Person("John", "Doe", 31)
>>> john.firstname()
'John'
```

Wrapping Maps, Lists, etc. as Results and Parameters

Also: Errors

Complex Parameters/Results - Go

Golang Error

```
func (p *Person) AddFriend(friend *Person) (uint, error) {...}
```

Return Objects

```
func (p *Person) GetFriends() []*Person {...}
```

Return Slice of Strings

```
func (p *Person) GetFriendFirstNames() []string {...}
```

Return Map

```
func (p *Person) GetFriendCountByAge() map[uint]uint {...}
```

Complex Results - How?

How to express slices/list or maps/dicts on the way from Go to Python (and back)?

Go	**Cgo / C**	**Cython / Python**
int, uint…	(unsigned) int	int
string	*char	str
user type	object id	extension type
slice	**?**	list
map	**?**	dict
error	**?**	exception

Complex Results - Idea

C++ has std::vector, std::map

Cython can wrap C++

Go	Cgo / C	Cython	Python
int, uint…	(unsigned) int	(unsigned) int	int
string	*char	<=>	str
user type	object id	<=>	extension type
slice	*void to std::vector, helper functions	<=>	list
map	*void to std::map, helper functions	<=>	dict
errors	output parameter	<=>	exception

Output parameters are ok for errors, for now

Complex Results - C API

```
#ifdef __cplusplus
extern "C" {
#endif

typedef void* OPAQUE_STRING_LIST;
typedef void* OPAQUE_OID_LIST;
typedef void* OPAQUE_UINT2UINT_MAP;

OPAQUE_STRING_LIST get_OPAQUE_STRING_LIST();
void append_to_OPAQUE_STRING_LIST(OPAQUE_STRING_LIST obj,char * val);

OPAQUE_OID_LIST get_OPAQUE_OID_LIST();
void append_to_OPAQUE_OID_LIST(OPAQUE_OID_LIST obj, unsigned long oid);
unsigned int get_size_OPAQUE_OID_LIST(OPAQUE_OID_LIST obj);
unsigned long get_at_OPAQUE_OID_LIST(OPAQUE_OID_LIST obj, unsigned int index);

OPAQUE_UINT2UINT_MAP get_OPAQUE_UINT2UINT_MAP();
void insert_into_UINT2UINT_MAP(OPAQUE_UINT2UINT_MAP obj,unsigned int age,
    unsigned int count);

#ifdef __cplusplus
}
#endif
```

Complex Results - C++

```
OPAQUE_STRING_LIST get_OPAQUE_STRING_LIST() {
    auto list = new vector<string>();
    return static_cast<OPAQUE_STRING_LIST>(list);
}

void append_to_OPAQUE_STRING_LIST(OPAQUE_STRING_LIST obj, char *val) {
    auto list = static_cast<vector<string>*>(obj);
    auto str = string(val);
    free(val);
    list->push_back(str);
}

OPAQUE_OID_LIST get_OPAQUE_OID_LIST() {
    auto list = new vector<unsigned long>();
    return static_cast<OPAQUE_OID_LIST>(list);
}

void append_to_OPAQUE_OID_LIST(OPAQUE_OID_LIST obj, unsigned long oid) {
    auto list = static_cast<vector<unsigned long>*>(obj);
    list->push_back(oid);
}

OPAQUE_UINT2UINT_MAP get_OPAQUE_UINT2UINT_MAP() {
    auto map = new unordered_map<unsigned int, unsigned int>();
    return static_cast<OPAQUE_UINT2UINT_MAP>(map);
}

void insert_into_UINT2UINT_MAP(OPAQUE_UINT2UINT_MAP obj, unsigned int age, unsigned int count) {
    auto map = static_cast<unordered_map<unsigned int, unsigned int>*>(obj);
    map->insert({age,count});
}
```

Complex Results -CGO

```
//export cgo_Person_GetFriends
func cgo_Person_GetFriends(cgo_oid C.ulong) C.OPAQUE_OID_LIST {
  var oid uint64 = uint64(cgo_oid)
  var person *main_lib_simple.Person = cgo_lib_simple.ObjectForPersonId(oid)
  var friends []*main_lib_simple.Person = person.GetFriends()
  var cgo_result C.OPAQUE_OID_LIST = C.get_OPAQUE_OID_LIST()
  for _, friend := range friends {
    var oid_friend uint64 = cgo_lib_simple.PersonIdForObject(friend)
    var cgo_oid_friend C.ulong = C.ulong(oid_friend)
    C.append_to_OPAQUE_OID_LIST(cgo_result, cgo_oid_friend)
  }
  return cgo_result
}
```

Complex Results - Cython

```
cdef extern from "cgo_lib/cgo_lib.h":
    unsigned int cgo_Person_AddFriend(long unsigned int p0, long unsigned int p1, char** p2)
    OPAQUE_OID_LIST cgo_Person_GetFriends(long unsigned int p0)
    OPAQUE_STRING_LIST cgo_Person_GetFriendFirstNames(long unsigned int p0)
    OPAQUE_UINT2UINT_MAP cgo_Person_GetFriendCountByAge(long unsigned int p0)
```

```
cpdef list get_friends(self):
    cdef OPAQUE_OID_LIST opaque_p_id_vector = cgo_Person_GetFriends(self.oid)
    cdef vector[unsigned long]*p_id_vector = <vector[unsigned long]*> opaque_p_id_vector
    cdef list result = []
    for oid_friend in p_id_vector[0]:
        friend = Person.get_person(oid_friend)
        result.append(friend)
    del p_id_vector
    return result
```

Errors -CGO

```
//export cgo_Person_AddFriend
func cgo_Person_AddFriend(cgo_oid C.ulong, cgo_oid_friend C.ulong, cgo_error **C.char) C.uint {
  if *cgo_error != nil {
    panic("Outout Error Parameter Destination must point to Null")
  }

  var oid uint64 = uint64(cgo_oid)
  var person *main_lib_simple.Person = cgo_lib_simple.ObjectForPersonId(oid)
  var oid_friend uint64 = uint64(cgo_oid_friend)
  var friend *main_lib_simple.Person = cgo_lib_simple.ObjectForPersonId(oid_friend)
  friend_count, err := person.AddFriend(friend)

  if (err != nil) {
    err_msg := err.Error()
    var cgo_err_msg *C.char = C.CString(err_msg) // allocates, needs free() by caller
    *cgo_error = cgo_err_msg;
  }

  var cgo_result C.uint = C.uint(friend_count)
  return cgo_result
}
```

Errors- Cython

```
cdef extern from "cgo_lib/cgo_lib.h":
    unsigned int cgo_Person_AddFriend(long unsigned int p0, long unsigned int p1, char** p2)
    OPAQUE_OID_LIST cgo_Person_GetFriends(long unsigned int p0)
    OPAQUE_STRING_LIST cgo_Person_GetFriendFirstNames(long unsigned int p0)
    OPAQUE_UINT2UINT_MAP cgo_Person_GetFriendCountByAge(long unsigned int p0)
```

```
def add_friend(self, Person friend):
    cdef unsigned long oid_friend = friend.oid
    cdef unsigned long oid = self.oid
    cdef char *error_char_pointer = NULL
    cdef char ** error_char_pointer_pointer = &error_char_pointer
    cdef str error_string
cdef int friend_count = cgo_Person_AddFriend(oid, oid_friend, error_char_pointer_pointer)
if error_char_pointer != NULL:
    error_string = tounicode_with_free(error_char_pointer_pointer[0])
    raise Exception(error_string)
return friend_count
```

Complex Results & Errors - Python

```
>>> import cgo_cython
>>> john = cgo_cython.Person("John", "Doe", 32)
>>> jane = cgo_cython.Person("Jane", "Doe", 41)
>>> jake = cgo_cython.Person("Jake", "Doe", 23)
>>> john.add_friend(jane)
1
>>> john.add_friend(jake)
2
>>> john.get_friends_count_by_age()
{23: 1, 41: 1}
>>> john.get_friends()
[<cgo_cython.Person object at 0x108706870>, <cgo_cython.Person object at 0x108706690>]
>>> john.add_friend(jake)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
  File "cgo_cython.pyx", line 130, in cgo_cython.Person.add_friend
    error_string = tounicode_with_free(error_char_pointer_pointer[0])
Exception: Cannot have the same friend twice
```

Wrapping Callbacks

Wrapping Callbacks - Go Code

How to have some callbacks?

Callback: Int (age) to bool

```
func (p *Person) GetFriendsFilteredByAge(fun func(uint) bool) []*Person {
    result := []*Person{}
    for _, friend := range p.friends {
        if fun(friend.age) {
            result = append(result, friend)
        }
    }
    return result
}
```

Wrapping Callbacks - Thougths

We want Go to call Python and use the result

↘

We need to express the callback in our C/CGO layer

↙

C callpacks are function pointers…

↘

Cython can build C-functions that call Python Code

↓

So, we can have a C function pointer with Python Code

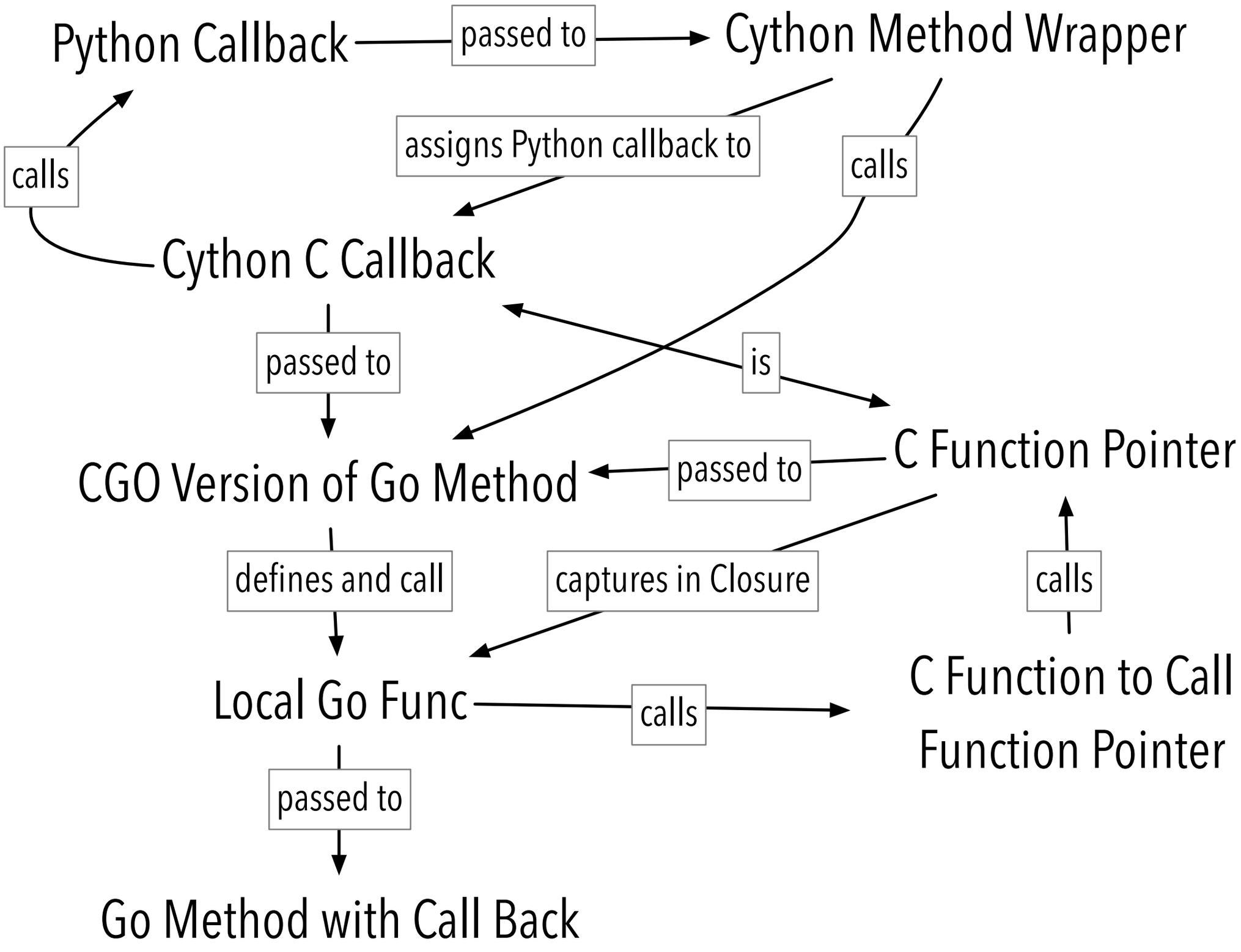
↙

but CGO cannot call function pointers

↘

CGO can call a c function that calls a function pointer

Callbacks - Rube Goldberg Approach



C Helper Functions for Callbacks

```
typedef int (*CALLBACK_AGE)(unsigned int);
int call_CALLBACK_AGE(CALLBACK_AGE cgo_fun,unsigned int age);
```

```
int call_CALLBACK_AGE(CALLBACK_AGE cgo_fun,unsigned int age) {
    return cgo_fun(age);
}
```

Wrapping Callbacks - CGO

```
//export cgo_Person_GetFriendsFilteredByAge
func cgo_Person_GetFriendsFilteredByAge(

  cgo_oid C.ulong, cgo_fun C.CALLBACK_AGE,

  ) C.OPAQUE_OID_LIST {
  var oid uint64 = uint64(cgo_oid)
  var person *main_lib_simple.Person = cgo_lib_simple.ObjectForPersonId(oid)



  fun := func(friend_age uint) bool {

    return C.call_CALLBACK_AGE(cgo_fun,C.uint(friend_age)) != 0

  }

  var friends []*main_lib_simple.Person = person.GetFriendsFilteredByAge(fun)

  return build_OPAQUE_OID_LIST(friends)
}
```

Wrapping Callbacks - Cython

```
cpdef list get_friends_filter_by_age(self, object fun):
    global _PersonFilterFun
    _PersonFilterFun = fun

    cdef OPAQUE_OID_LIST opaque_p_id_vector = \
        cgo_Person_GetFriendsFilteredByAge(self.oid, filter_by_age)

    result = Person._build_result_list(opaque_p_id_vector)

    _PersonFilterFun = None
    return result


cdef int filter_by_age(unsigned int age):
    fun = _PersonFilterFun
    return fun(age)
```

Python

```
In[2]: import cgo_cython as cgo_mod
In[3]: p1 = cgo_mod.Person("Jake", "Ant", 41)
   ...: p2 = cgo_mod.Person("Joe", "Bee", 12)
   ...: p3 = cgo_mod.Person("Jane", "Cat", 11)
   ...: p4 = cgo_mod.Person("Jill", "Dog", 12)
   ...: p1.add_friend(p2)
   ...: p1.add_friend(p3)
   ...: p1.add_friend(p4)
   ...: f = p1.get_friends_filter_by_age(lambda x: x > 10 and x % 2 == 1)[0]
   ...: f.string()
Out[3]: 'Jane Cat (11)'
```

Performance, Effort,

Summary

Effort and Gain Wrapping Golang

Golang ~ 130 Lines

CGO ~ 220 Lines

C Header ~30 Lines

CPP Support Code ~50 Lines

Cython ~200 Lines

For this (simple) example, the amount of wrapper code is significant

Code is repetitive, leads itself to automation

A Simple (Accessor) Benchmark

```
def get_most_befriended_report(persons, use_str):
    friend_count = collections.Counter()
    for person in persons:
        for friend in person.get_friends():
            if use_str:
                person_txt = friend.string()
            else:
                person_txt = "%s %s, age %s " % (friend.firstname(),
                                                 friend.lastname(),
                                                 friend.age())
            friend_count[person_txt] +=1
most_common = friend_count.most_common(10)
return "\n".join("%5s : %s" % (v, k) for k, v in most_common) + "\n"
```

Go Bindings vs. Python Performance

```
def run_persons(n):
    persons = get_persons(n,50)
    print_most_befriended_str(persons)
    print_most_befriended_getter(persons)
    print_most_befriended_go(persons)
```

	Go	Python
	1493ms	619ms
	1360ms	256ms
	3209ms	443ms
	93ms	

Performance of Method Calls

Go

Name	Own Time (ms)	Call Count
<method 'string' of 'cgo_cython.Person' objects>	1011	248251
<method 'firstname' of 'cgo_cython.Person' objects>	933	248251
<method 'lastname' of 'cgo_cython.Person' objects>	892	248251
<method 'add_friend' of 'cgo_cython.Person' objects>	875	248251
<method 'age' of 'cgo_cython.Person' objects>	828	248251
get_most_befriended_report	495	2

Python

Name	Own Time (ms)	Call Count
firstname	32	248280
lastname	30	248280
age	26	248280
string	113	248280
add_friend	54	248280

Wrapping Go - Lessons Learned

Things work	Many Go Feature can be expressed in Python
	Cython works and is a pleasure to use
	CGO works
	Some boilerplate, but nothing to bad
	Improvements possible (Errors, Interfaces, Channels?)
Performance is a challenge	Overhead to call into Go (runtime)
	Mapping from Pointers to IDs
	Looking into alternatives (C++ Msg Queue?)

The End