Language Embedding and Foreign Function Interface

Lua+C, Python+C, Go+C…..

Yu Zhang

Course web site: http://staff.ustc.edu.cn/~yuzhang/pldpa

Language Embedding

- A new language *N* is an embedded language in an existing language *E* if an expression in *N* can be used as a subexpression of a construct in *E*.
- In the Web domain there are many examples
  - CSS in HTML, and other XML languages
  - Javascript in HTML for client dynamics
  - JSP (Java fragments) in HTML at the server-side
    ASP(Visual Basic fragments)
    PHP(C-like fragments)
    BRL (Scheme fragment)

Foreign Function Interface (FFI)

- FFI is a mechanism by which a program written in one programming language can call routines or make use of services written in another.
- Examples of FFIs include:
  - Dynamic languages such as Python, Perl, Ruby provide easy access to native code written in C/C++
  - cgo: Go can call C code directly via the “C” pseudo-package
  - JNI(Java native interface) provides an interface between Java and C/C++
  - …

Outline

- Understand the motivation for embedding a PL
- Explore the issues of interop between high/low level PLs
- Understand/evaluate Lua's interop solution

Embedding Lua

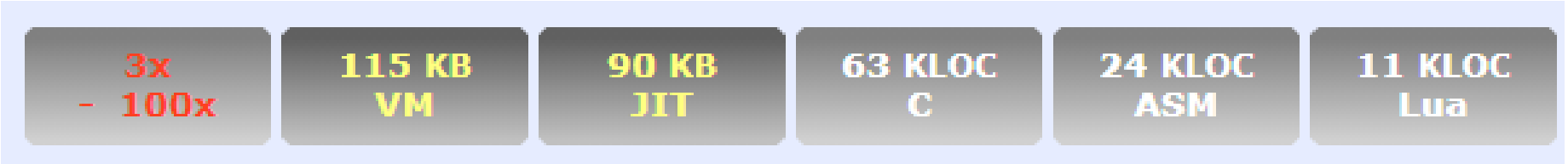
- Programming in Lua **Part IV · The C API**
  - ★ 24 – An Overview of the C API
  - 25 – Extending your Application
  - 26 – Calling C from Lua
- LuaJIT, a just-in-time implementation of Lua, has an FFI that allows "calling external C functions and using C data structures from pure Lua code
- Why Lua would be a good choice for a cell phone VM
- Code examples

Lua is Designed for Embedding

- Interpreter is small
  - Lua v5.4.0: 35 files, 14K LOC, 700 KB, ~1s compile time

lua-5.4.0
$ ls *.c
lapi.c lctype.c lfunc.c lmathlib.c loslib.c ltable.c lundump.c
lauxlib.c ldblib.c lgc.c lmem.c lparser.c ltablib.c lutf8lib.c
lbaselib.c ldebug.c linit.c loadlib.c lstate.c ltests.c lvm.c
lcode.c ldo.c liolib.c lobject.c lstring.c ltm.c lzio.c
lcorolib.c ldump.c llex.c lopcodes.c lstrlib.c **lua.c** onelua.c

http://luajit.org/luajit.html



Lua is Designed for Embedding

- Interpreter is small
  - Lua v5.4.0: 35 files, 14K LOC, 700 KB, ~1s compile time
  - Python v3.9: 4000 files, 900K LOC, 300 MB, ~1.5m compile time
- Language is small
  - Not much syntax
  - Few core language concepts (e.g. no classes)
  - Small standard library
- Relatively simple C API
  - Small language = small API surface
  - Easily sandboxed

C Embedded in Lua?

- Easy access to many libraries
  - C ABI (Application binary interface) is the lowest common denominator for many PLs

[2020-01-07]What is ABI, and What Should WG21 Do About it?

GCC - abi.html, Itanium C++ ABI specfication

- The mangled name for a C++ function (Functions in extern “C” blocks handled differently).
- The mangled name for a type, including instantiations of class templates.
- The number of bytes (sizeof) and the alignment, for an object of any type.
- The semantics of the bytes in the binary representation of an object.
- Register-level calling conventions governing parameter passing and function invocation.

```
namespace wikipedia {
   class article {
   public: /* Mangles as: _ZN9wikipedia7article6formatEv */
      std::string format(void);
   };
};
```

C Embedded in Lua?

- Easy access to many libraries
  - C ABI (Application binary interface) is the lowest common denominator for many PLs
- Improve performance of bottlenecks
  - In Python, numpy

    Array programming with Numpy, Nature 2020

    https://www.nature.com/articles/s41586-020-2649-2
- Extend language semantics
  - Async I/O, threading, …

C in Lua and Lua in C

- Lua uses a *virtual stack* to pass values to & from C
  - lua_checkstack: check stack size
- C API for Lua in lua.h
  - the set of C functions available to the host program to communicate with Lua
  - C Closures (lua_pushcclosure): When a C function is created, it is possible to associate some values with it, thus creating a C closure; these values are called upvalues and are accessible to the function whenever it is called.
- The *auxiliary library* in luaxlib.h and have a prefix luaL_.
  - provide several convenient functions to interface C with Lua.

Communication between Lua and C

- 两种调用视图均通过相同的C API进行通信
  - C调用Lua时称C代码为application code
  - Lua调用C时称C代码为library code
- Lua和C之间交换值面临的主要问题
  - 类型系统不匹配：动态 vs. 静态
  - 内存管理不匹配：自动 vs. 手动
- 通过栈进行Lua和C之间的数据交换
  - 栈是Lua状态的一部分
  - 提供不同的函数来将不同类型的值压入栈
  - 栈不是全局结构，每个函数有自己私有的局部栈。当Lua调用C函数时，第一个参数在局部栈的index 始终为1。
- 利用 Lua 来配置程序

C in Lua

counter.c defines C module

```c
#include <stdio.h>
#include <string.h>
#include <lua.h>
#include <lauxlib.h>
#include <lualib.h>

typedef struct {
  int counter;
} counter_t;

static int counter_new(lua_State* L) {
  counter_t* counter = (counter_t*) lua_newuserdata(L, sizeof(counter_t));
  counter->counter = 0;
  return 1;
}

static int counter_incr(lua_State* L) {
  counter_t* counter = (counter_t*) lua_touserdata(L, 1);
  counter->counter++;
  return 0;
}
```

lua_newuserdata :allocates a new block of memory with the given size, pushes onto the stack a new full userdata with the block address, and returns this address. The host program can freely use this memory.

lua_touserdata: If the value at the given index is a full userdata, returns its block address….

C in Lua

counter.c defines C module

```
23  int luaopen_counter(lua_State* L) {
24    lua_createtable(L, 1, 0); // local x = {}
25
26    lua_pushstring(L, "new");
27    lua_pushcfunction(L, counter_new);
28    lua_settable(L, -3);
29
30    lua_pushstring(L, "incr");
31    lua_pushcfunction(L, counter_incr);
32    lua_settable(L, -3);
33
34
35    return 1;
36  }
```

定义 C Modules

- 每个C module 有唯一的public (extern)函数，其余的都是private (static). 在 counter.c 中,
  - luaopen_counter() 定义了名为 counter的module的public函数
  - 该module有两个私有函数 counter_new()、 counter_incr()
- 将counter.c编译连接得到动态库 counter.so (Linux) 或者 counter.dll (Windows)

C in Lua

```lua
local counter = require "counter"
local c = counter.new()
counter.incr(c)
counter.incr(c)
print(counter.get(c))

local counter = {}
function counter.new()
    return {c = 0}
end

function counter.incr(c)
  c.c = c.c + 1
end

function counter.get(c)
  return c.c
end
```

- counter_test.lua

在Lua中调用C库

- 利用require加载C库 counter
- 调用C库中的 new和 incr

Lua in C

- lua_basic.c

```c
#include <stdio.h>
#include <string.h>
#include <lua.h>
#include <lauxlib.h>
#include <lualib.h>

int main () {
  lua_State* L = luaL_newstate();
  luaL_openlibs(L);

  char* cmd = "print('hi')";
  luaL_loadbuffer(L, cmd, strlen(cmd), "");
  lua_call(L, 0, 0);

  lua_close(L);
}
```

Creates a new Lua state. It calls lua_newstate with an allocator based on the standard C realloc function and then sets a panic function that prints an error message to the standard error output in case of fatal errors.

Opens all standard Lua libraries into the given state.

Loads a buffer as a Lua chunk.

Calls a function. The 2nd and 3rd arguments are nargs and nresults.

Lua in C

```
7   int main () {
8     lua_State* L = luaL_newstate();
9     luaL_openlibs(L);
10
11    char* cmd = "add = function(x, y) return x + y end";
12    luaL_loadbuffer(L, cmd, strlen(cmd), "");
13    lua_call(L, 0, 0);
14
15    lua_getglobal(L, "add");
16
17    lua_pushnumber(L, 1);
18    lua_pushnumber(L, 5);
19    lua_call(L, 2, 1);
20
21    printf("%.f\n", lua_tonumber(L, 1));
22
23    lua_close(L);
24  }
```

- lua_add.c

Pushes onto the stack the value of the global name. Returns the type of that value.

Pushes a float with value n onto the stack.

Converts the Lua value at the given index to the C type lua_Number.

Issues of Interop between PLs

- Lua and C
  - Type system: dynamic vs. static
  - Memory management: automatic vs. manual

- C and Fortran
  - Fortran: HPC（complex number），column-major order, subroutine and function, call-by-reference
  - C: argc and argv，dynamic memory management, row-major order, call-by-value
  - E.g. generate mpif.h by wrapping mpi library written in C

cgo

https://golang.org/cmd/cgo/

go tool cgo [cgo options] [-- compiler options] gofiles...

- C in Go

```
package rand

/*
#include <stdlib.h>
*/
import "C"

func Random() int {
    return int(C.random())
}

func Seed(i int) {
    C.srandom(C.uint(i))
}
```

```
package main

// #include <stdio.h>
// #include <stdlib.h>
//
// static void myprint(char* s) {
//   printf("%s\n", s);
// }
import "C"
import "unsafe"

func main() {
        cs := C.CString("Hello from stdio")
        C.myprint(cs)
        C.free(unsafe.Pointer(cs))
}
```

cgo

- Go in C
  - Go functions

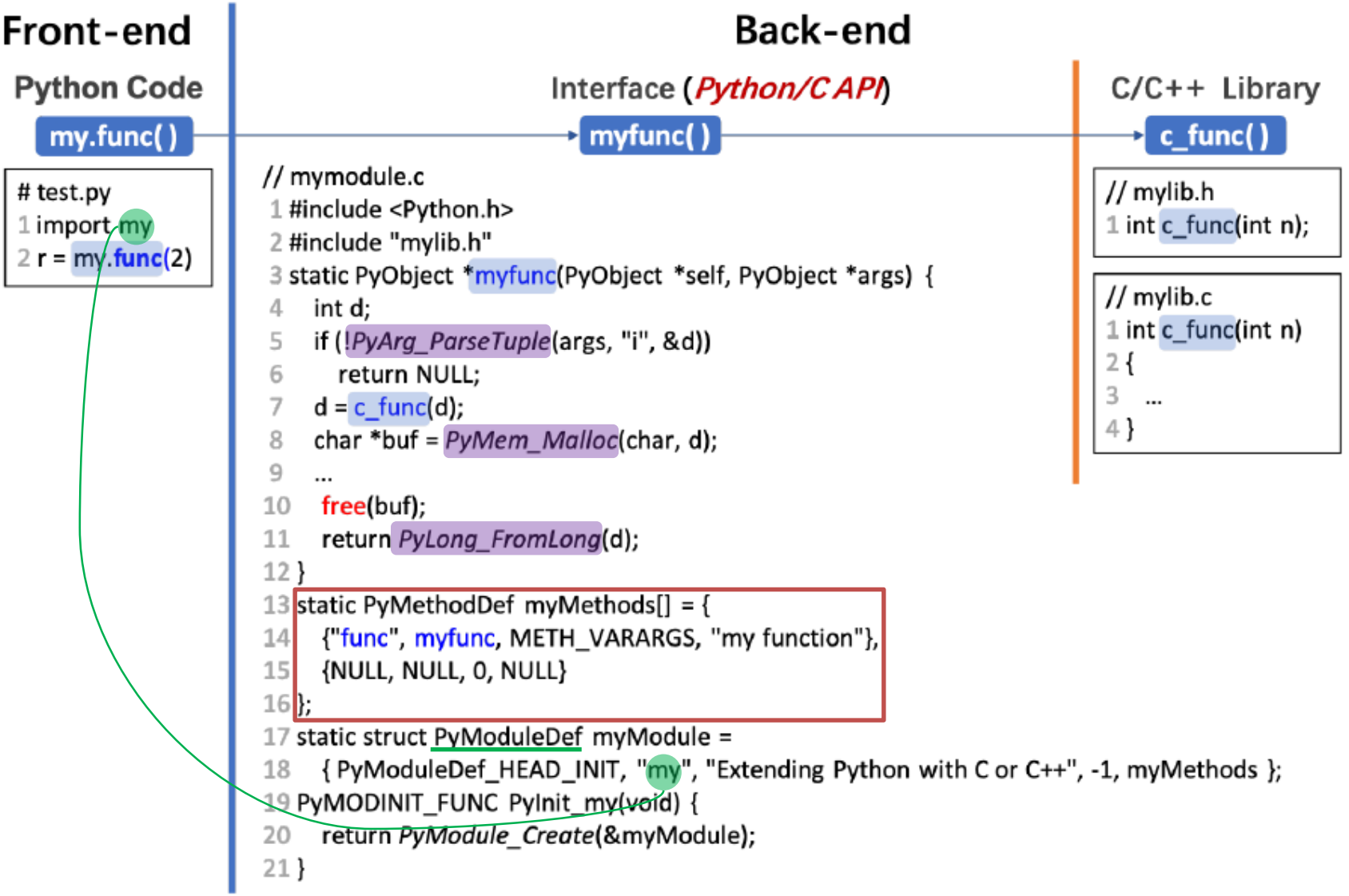
```
//export MyFunction
func MyFunction(arg1, arg2 int, arg3 string) int64 {...}

//export MyFunction2
func MyFunction2(arg1, arg2 int, arg3 string) (int64, *C.char) {...}
```

  - Used in C code

```
extern GoInt64 MyFunction(int arg1, int arg2, GoString arg3);
extern struct MyFunction2_return MyFunction2(int arg1, int arg2, GoString arg3);
```

Python Extension

https://docs.python.org/3/extending/extending.html



Issues of Interop between PLs

- Spark
  - Write applications in Scala, Java, Python, R
  - Combine SQL, streaming and complex analytics (GraphX, MLlib, …)
  - Run on Hadoop, Mesos, Kubernetes, standalone, or in the cloud
  - Access diverse data sources, e.g. HDFS, Cassandra, HBase, Hive, Tachyon, Amazon S3

- Javascript and WebGL …
  - Zhen Zhang. xWIDL: Modular and Deep JavaScript API Misuses Checking Based on eXtended WebIDL, SPLASH 2016 Student Research Contest, (Poster)

Our Work on Python and Go

- [SANER2020] An Empirical Security Study of the Python/C API(ERA) https://github.com/S4Plus/pyceac
- [SANER2021] An Empirical Study For Common Language Features Used in Python Projects. https://github.com/S4Plus/PyScan
- [ISSRE2021] Static Type Inference for Foreign Functions of Python https://github.com/S4Plus/pyctype
- [JSEP2023] An Empirical Study of the Python/C API on Evolution and Bug Patterns
- [SANER2023] Cross-Language Call Graph Construction Supporting Different Host Languages
- [SANER2023] CGORewritter: A better way to use C library in Go (ERA)

Python/C API

- the evolution of the Python/C API

Version	Date	Usage	API	Add	Remove	Modify	Macro
2.7.0	2010.07.03	5%	563	-	-	-	253
3.2.0	2011.02.20	2%	663	179	79	30	323
3.3.0	2012.09.29		702	115	76	2	275
3.4.0	2014.03.17		732	30	0	16	275
3.5.0	2015.09.13		751	19	0	2	287
3.6.0	2016.12.23	7%	764	13	0	9	291
3.7.0	2018.06.27	13%	804	43	3	6	297
3.8.0	2019.10.14	27%	844	55	15	16	300
3.9.0	2020.10.05	35%	862	30	12	16	300

Python/C API

- the usage statistics of the Python/C API

Project	Version	Release Time	Description	Code Size (KLOC)	C/C++ Ratio	Python/C API		
						Kinds	Similarity	Freq.
Pillow	5.4.1	2019.01.07	image processing	65.4	40.8%	124	77.8%	1047
	7.2.0	2020.06.30		74.6	40.7%	116		1107
NumPy	1.16.2	2019.02.27	scientific computing	322.0	49.5%	274	82.1%	6166
	1.19.0	2020.06.21		353.4	46.8%	254		5907
TensorFlow	1.13.1	2019.02.26	machine learning	1989.2	50.2%	144	89.4%	1137
	2.1.1	2020.05.14		2391.2	59.4%	161		1322
PyTorch	1.0.1	2019.02.07	machine learning	687.2	55.1%	142	85.9%	1166
	1.5.1	2020.06.12		980.5	59.9%	161		1224
python-ldap	3.2.0	2019.03.12	directory access	13.5	20.9%	63	96.9%	300
	3.3.1	2020.06.29		13.9	21.2%	65		330
PyAudio	0.2.11	2017.03.19	audio I/O	3.3	56.2%	40	-	339
python-krbV	1.0.90	2012.03.22	Web authentication	4.9	86.4%	53	-	736

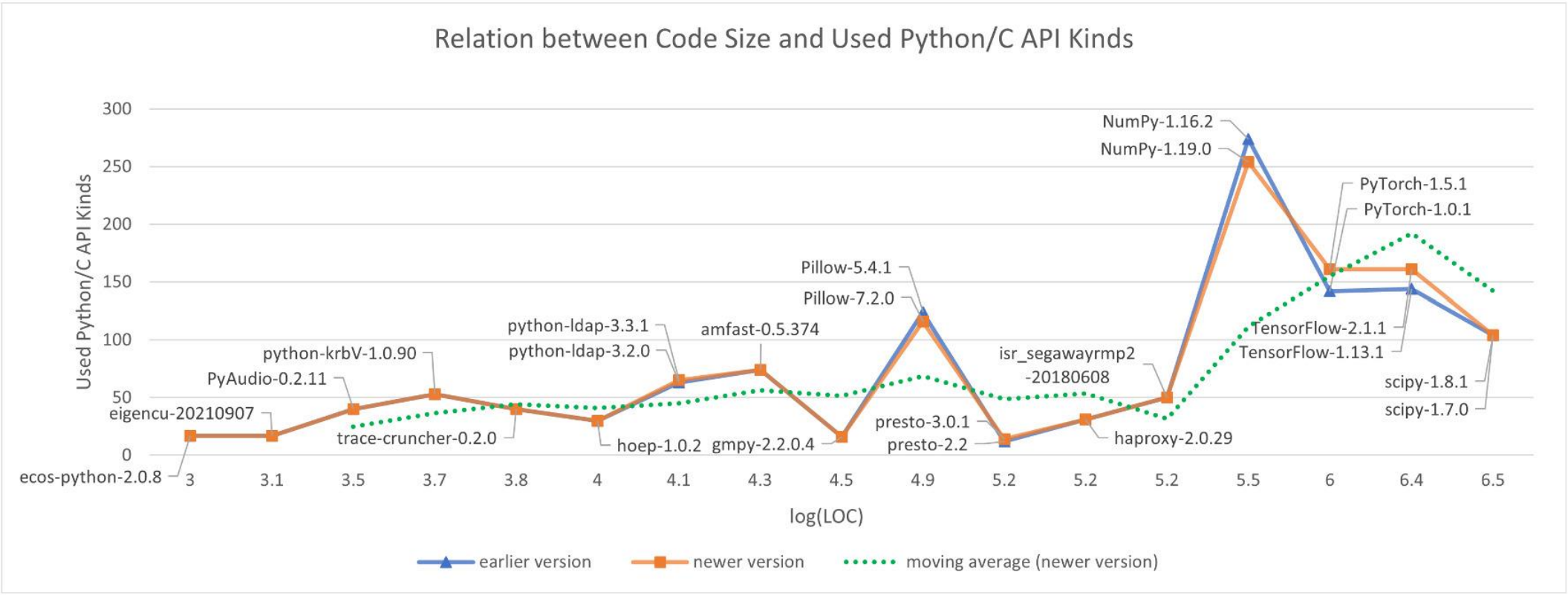
Kinds Distinct Python/C API callee numbers
Freq. Total number of the Python/C API call sites

Similarity: Jaccard similarity $J_j = \frac{|K_{j_1} \cap K_{j_2}|}{|K_{j_1} \cup K_{j_2}|}$

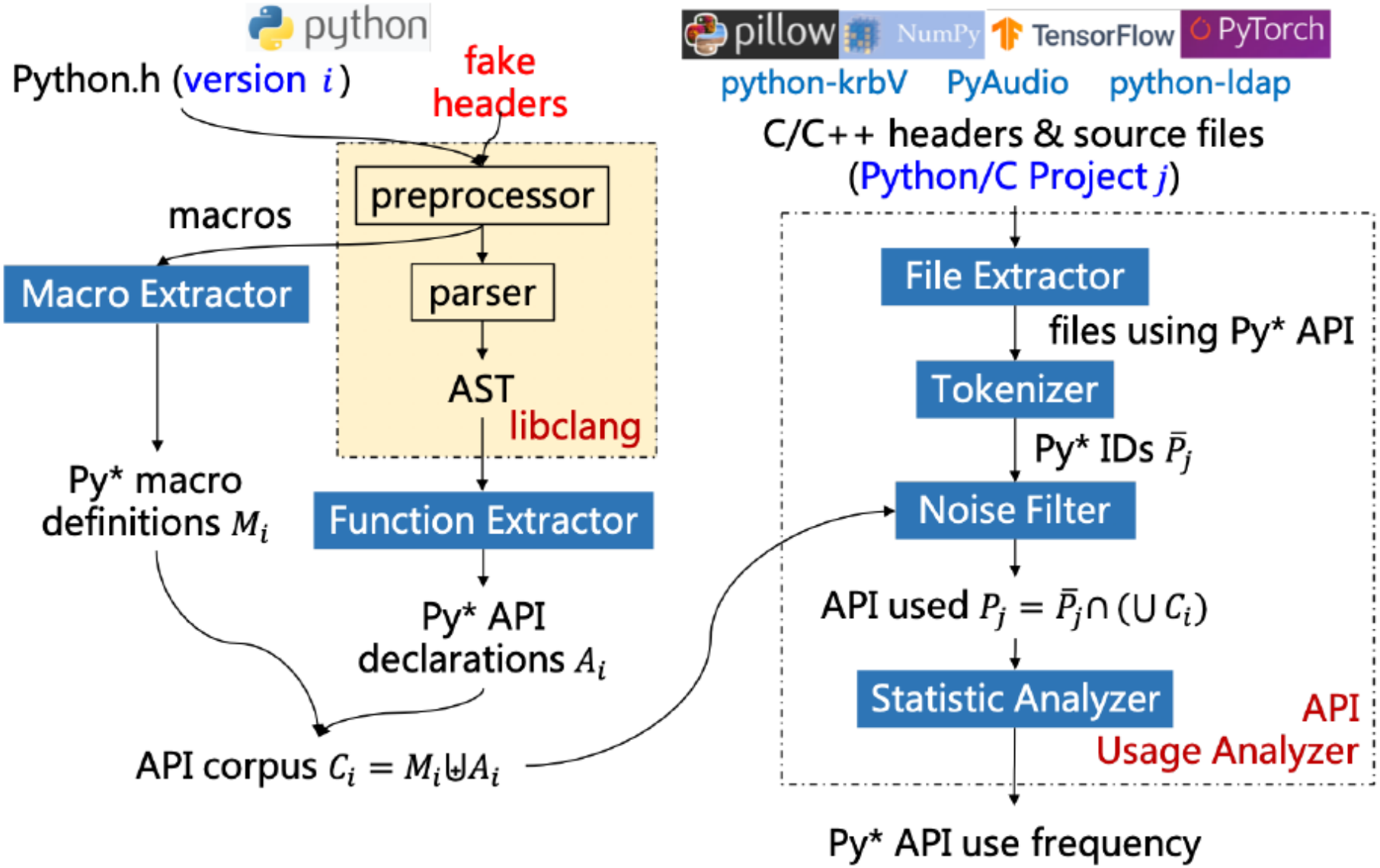
https://onlinelibrary.wiley.com/doi/10.1002/smr.2507

Python/C API

- the usage statistics of the Python/C API used in mainstream open source projects

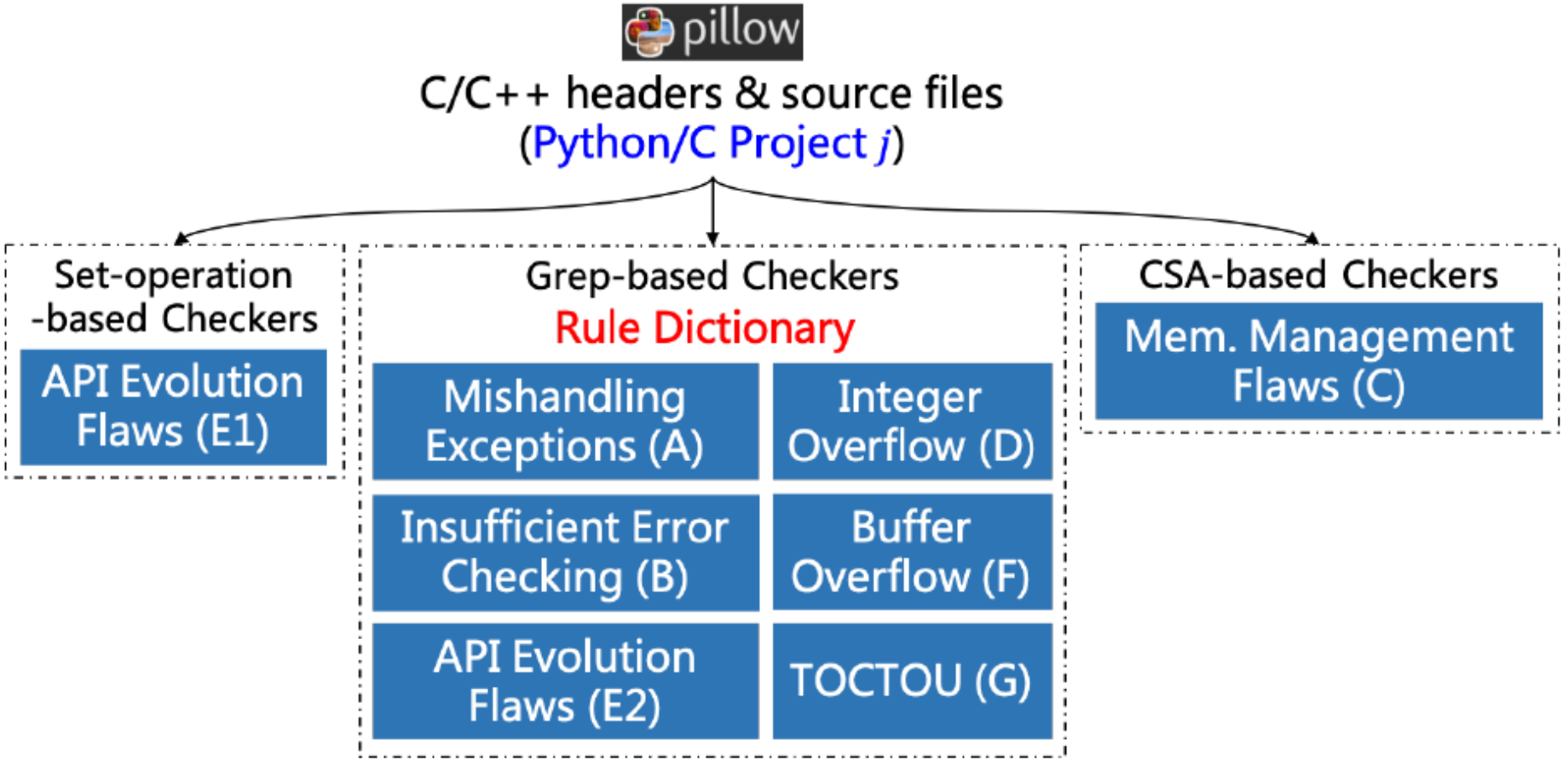


Toolset：PyCEAC

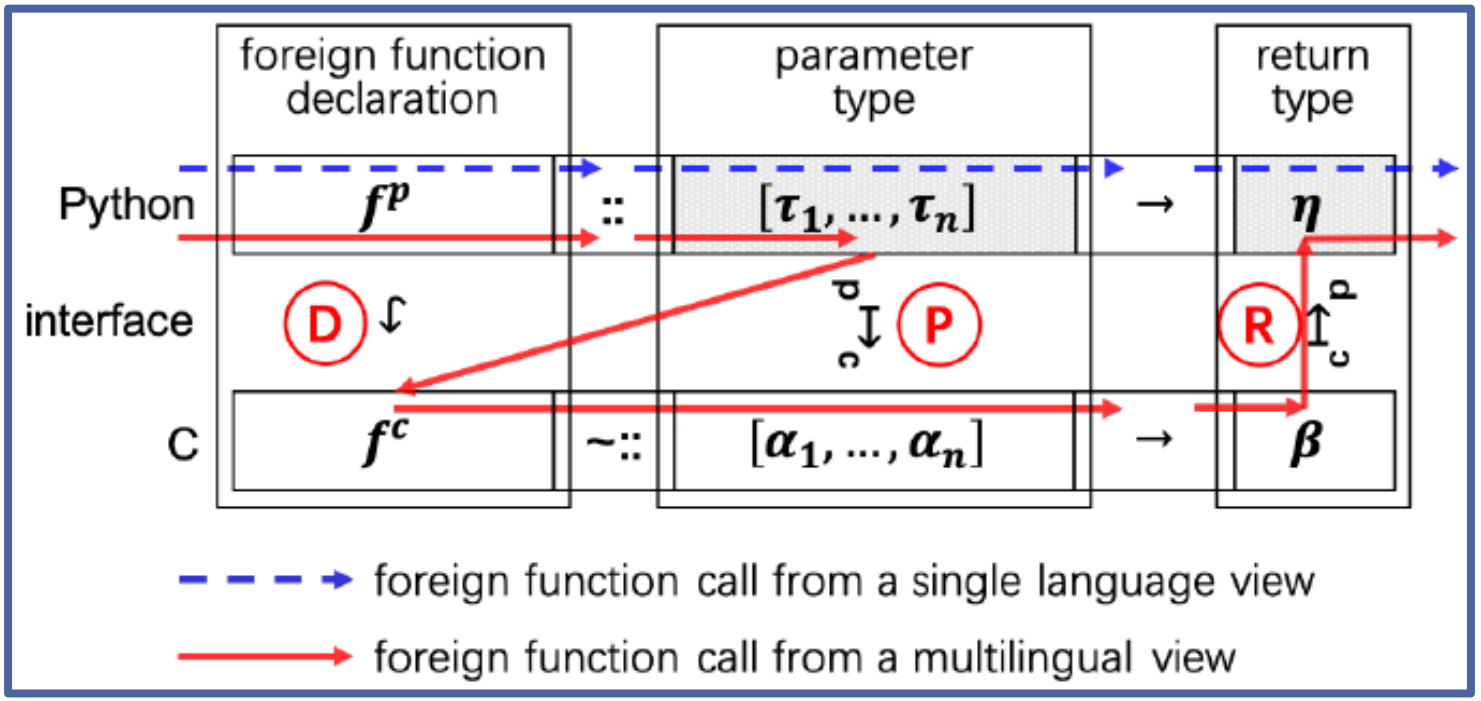


Python/C API Bug Patterns

- Mishandling Exceptions
- Insufficient Error Checking
- Memory Management Flaws
- Integer Overflow
- Buffer Overflow
- Time-of-check-to-time-of-use (TOCTOU)
- Reference Counting Errors
- Type Misuses
- Global Interpreter Lock (GIL) Bugs



Static Type Inference for Foreign Functions of Python



COMPLETENESS OF PARAMETER TYPE INFERENCE

Project	KLOC	Time (s)	Foreign Function	Pcc	Pap	Bug	Coverage
CPython	279.2	159.7	1529	503	606	32	74.6%
NumPy	507.1	76.4	90	9	51	6	73.3%
Pillow	29.0	40.7	132	2	119	10	99.2%
Total	815.3	276.8	1751	514	776	48	76.4%

相比Google的PyType，提升精度27.5%（针对Pillow）

THANKS