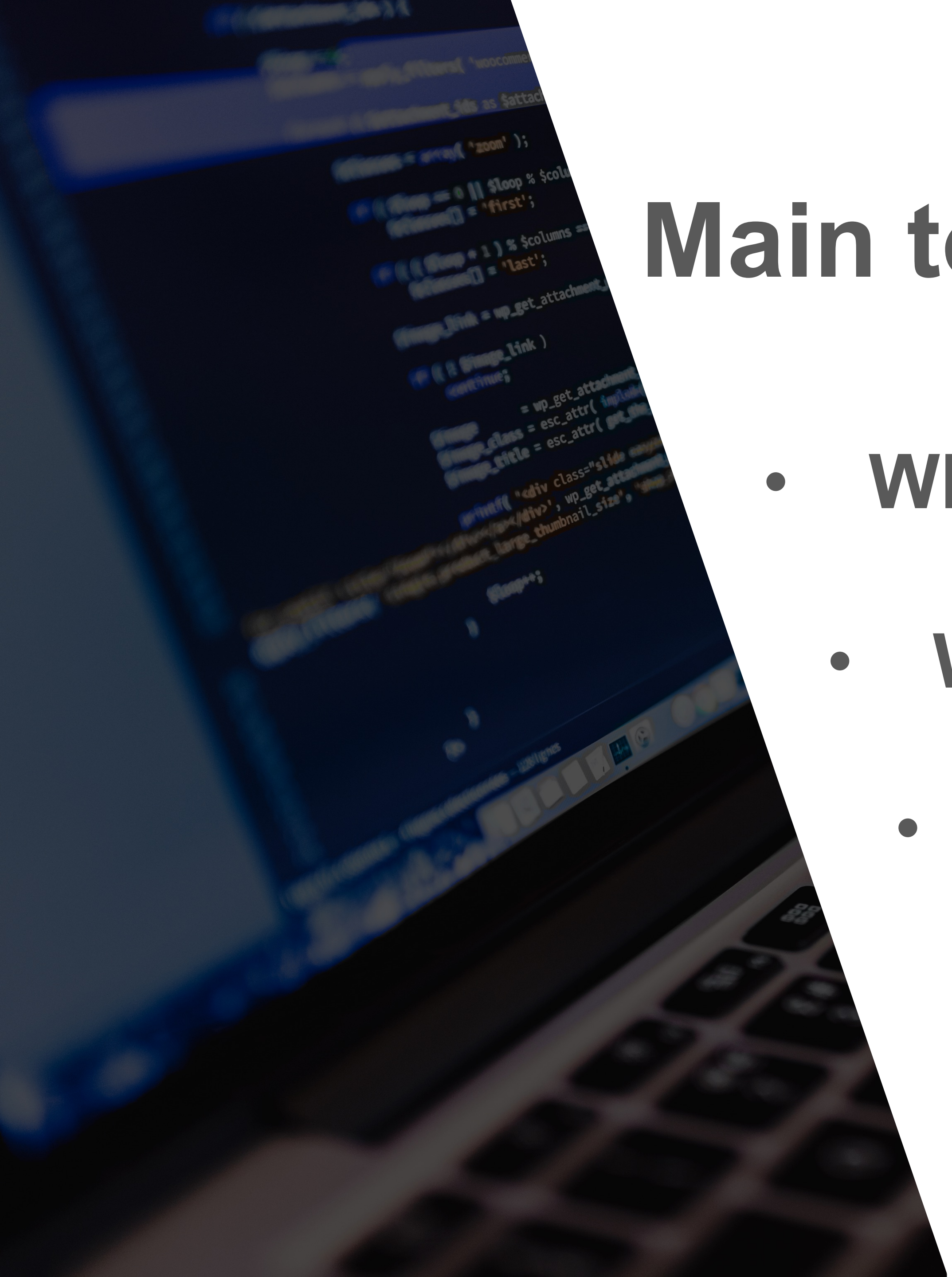
Updating gopy to support Python3 and PyPy

Dong-hee Na, Chungnam National University, Daejeon, Korea

Mentors: Sebastien Binet, Alexandre Claude
13. June. 2017

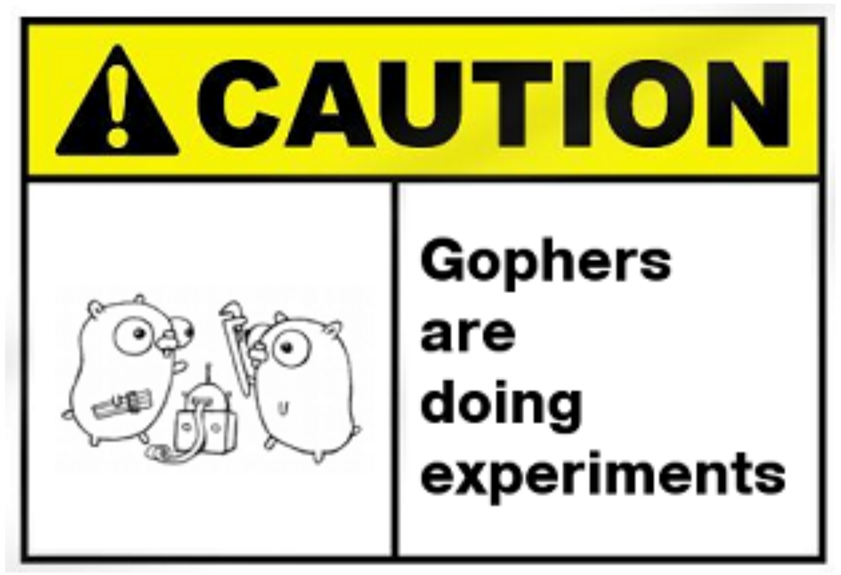


Main topics

- What is gopy?
- Why Go ..?
- Performance benchmark
- Support CFFI backend

What is gopy?

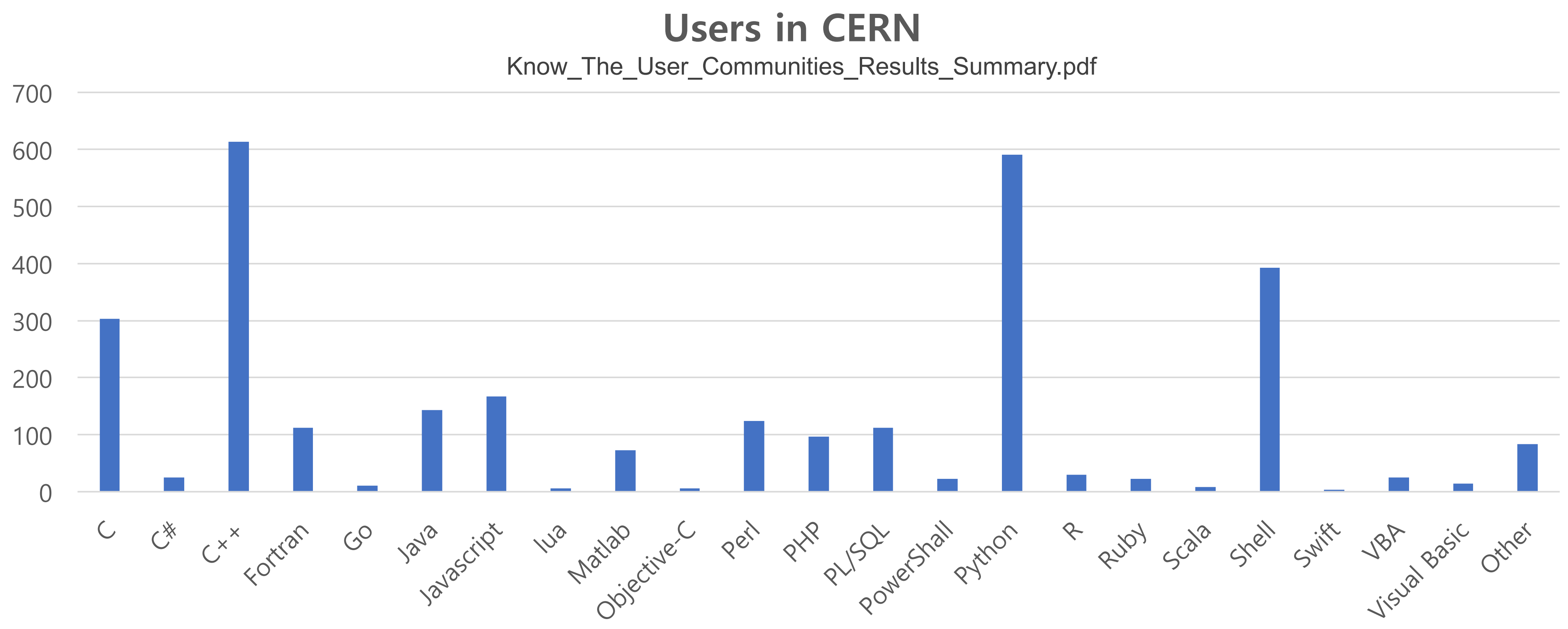
- gopy is heavily inspired from gomobile.
- gopy is a set of packages and build tools for using Go from python interpreters.
- Generates (and compiles) a Python extension module from a Go package.



(https://github.com/golang/mobile)

Why Go? – Software at CERN

- LHC: 90% of C++ codes and a bit of Python codes for steering.



Why Go ...?

Pros

Powerful performance

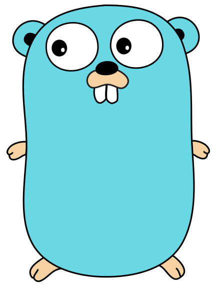
Compiled language

Cons

Long long compiling times

Manual memory Management

Learning curve is high



- Easy to install libraries with ‘go get’.
- Rich standard libraries.
- Built-in concurrency support
- Compile fast.
- Elegant and simple built-in build system.
- Garbage-collected language
- Error detecting is awesome.
- Learning curve is low
- Powerful performance.

Pros

Library installation is easy

Learning curve is low

Useful Scientific Packages

Cons

Global Interpreter Lock

Interpreter based

gopy: Calculation of Pi Using the Monte Carlo Method

```go
func monte_carlo_pi(reps int, result *int, wait *sync.WaitGroup) {
	var x, y float64
	count := 0
	seed := rand.NewSource(time.Now().UnixNano())
	random := rand.New(seed)


	for i := 0; i < reps; i++ {
		x = random.Float64() * 1.0
		y = random.Float64() * 1.0


		if num := math.Sqrt(x*x + y*y); num < 1.0 {
			count++
		}
	}


	*result = count
	wait.Done()
}
```

```go
func GetPI(samples int) float64  {
	cores := runtime.NumCPU()
	runtime.GOMAXPROCS(cores)

	var wait sync.WaitGroup

	counts := make([]int, cores)

	wait.Add(cores)


	for i := 0; i < cores; i++ {
		go monte_carlo_pi(samples/cores, &counts[i], &wait)
	}

	wait.Wait()

	total := 0
	for i := 0; i < cores; i++ {
		total += counts[i]
	}

	pi := (float64(total) / float64(samples)) * 4
	return pi
}
```

http://tstra.us/code/goPi/

gopy vs Python:
Calculation of Pi Using the Monte Carlo Method

- **gopy is very easy to install.**
- **Using a go get is all you have to do!**

```
$> go get -u -v github.com/go-python/gopy github.com/go-python/gopy
(download) github.com/gonuts/commander (download) github.com/gonuts/flag
(download) github.com/gonuts/flag github.com/go-python/gopy/bind
github.com/gonuts/commander github.com/go-python/gopy
```

gopy vs Python:
Calculation of Pi Using the Monte Carlo Method

```python
def monte_carlo_pi_part(n):
    count = 0
    for i in range(int(n)):
        x=random.random()
        y=random.random()

        # if it is within the unit circle
        if x*x + y*y <= 1:
            count=count+1

    #return
    return count


def GetPI(n):
    np = multiprocessing.cpu_count()
    part_count=[n/np for i in range(np)]
    pool = Pool(processes=np)
    count=pool.map(monte_carlo_pi_part, part_count)
    return sum(count)/(n*1.0)*4
```

```python
if __name__ == '__main__':

    n = 100000
    py_start = time.time()
    result = GetPI(n)
    py_end = time.time()
    print("Python result: %f time_elapsed: %f" % (result, py_end-py_start))

    go_start = time.time()
    result = calculatePi.GetPI(n)
    go_end = time.time()
    print("gopy result: %f time_elapsed: %f" %(result, go_end-go_start))
```

https://gist.github.com/amitsaha/2036026

gopy vs Python:
Calculation of Pi Using the Monte Carlo Method

- **gopy helps to use Go’s useful features on the Python interpreter.**
- **End-user can easily run Go codes on the Python interpreter.**

```
root@180a6474ebba:~/test# gopy bind github.com/go-python/gopy/_examples/calculatePi
2017/06/11 06:03:21 work: /tmp/gopy-546154656
root@180a6474ebba:~/test# python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import calculatePi
>>> calculatePi.GetPI(10000)
3.1564
root@180a6474ebba:~/test# python pi_mp.py
n: 100000
Python result: 3.145880 time_elapsed: 0.030254
gopy result: 3.137400 time_elapsed: 0.002960 ← Much Faster!!!
```

gopy: Limitation

- **gopy does not supports CPython3 nor PyPy.**
- **Many go's implementations/features are not yet implemented in gopy**

```
root@180a6474ebba:~/test# pypy pi_mp.py
Traceback (most recent call last):
  File "pi_mp.py", line 12, in <module>
    import calculatePi
ImportError: No module named calculatePi

root@180a6474ebba:~/test# python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import calculatePi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: dynamic module does not define module export function (PyInit_calculatePi)
```

GSoC Project: Support Python3 and PyPy

- Need to support Py2/3 and PyPy.
- PyPy’s implementation is not strictly 100% compatible with Ctypes.
- CFFI is a good choice to support various python compilers.
- CFFI interacts with almost any C code from Python.

```
root@180a6474ebba:~/test# gopy bind --lang=cffi github.com/go-
python/gopy/_examples/calculatePi
2017/06/11 06:06:46 work: /tmp/gopy-214312004
root@180a6474ebba:~/test# python pi_mp.py
n: 100000
Python result: 3.135280 time_elapsed: 0.024898
gopy result: 3.143200 time_elapsed: 0.006861 ← Much Faster!!!
root@180a6474ebba:~/test# pypy pi_mp.py
n: 100000
Python result: 3.147240 time_elapsed: 0.023687
gopy result: 3.145040 time_elapsed: 0.017225 ← Much Faster!!!
```

```
root@180a6474ebba:~/test# python3 pi_mp.py
Python result: 3.136560 time_elapsed: 0.037512
gopy result: 3.143920 time_elapsed: 0.003738 <- Much Faster
```

GSoC Project: Support Python3 and PyPy

1. **Inspects a Go package**
2. **Extracts the exported types, funcs, vars and consts**
3. **Creates a Go package that cgo exports the exported entities**
4. **Designates which interface should be exported to CFFI.**

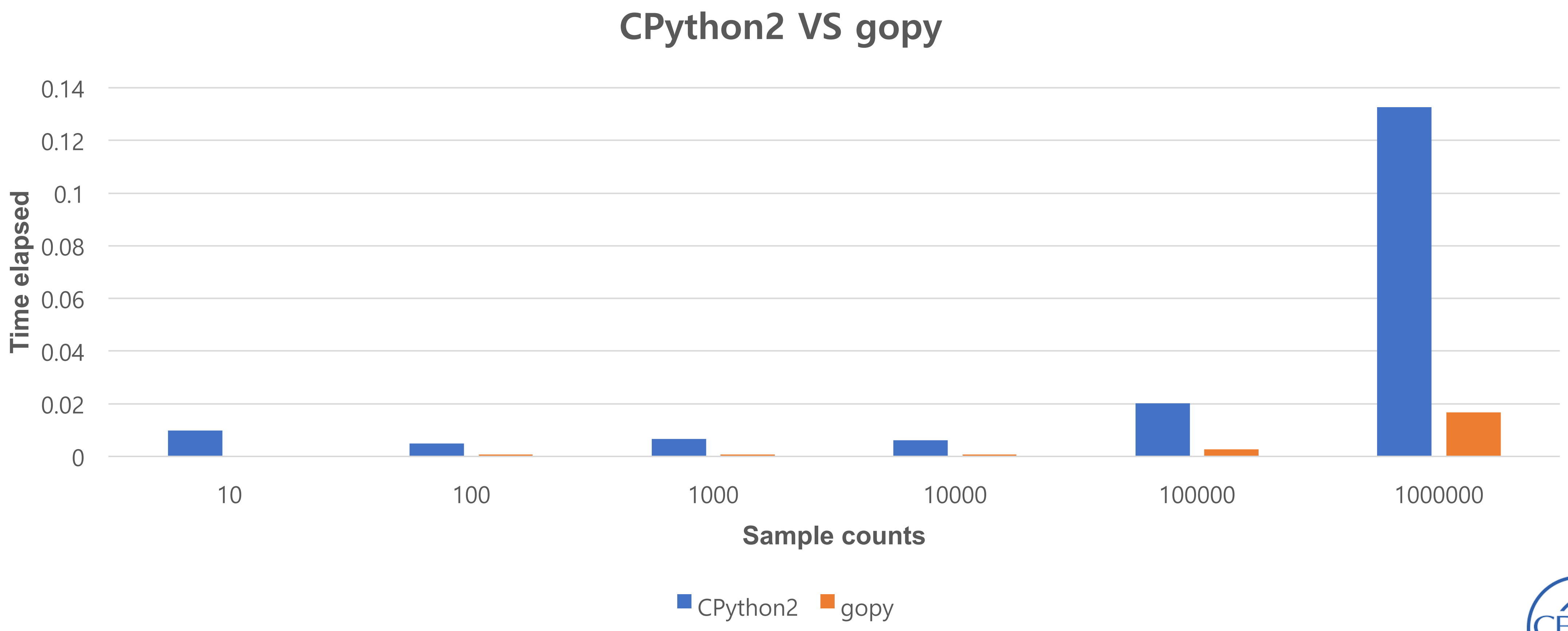
```
ffi.cdef("""
typedef signed char GoInt8;
typedef unsigned char GoUint8;
.
.
.

extern void cgo_pkg_calculatePi_init();
extern GoFloat64 cgo_func_calculatePi_GetPI(GoInt p0);
""")
```

GSoC Project: Support Python3 and PyPy

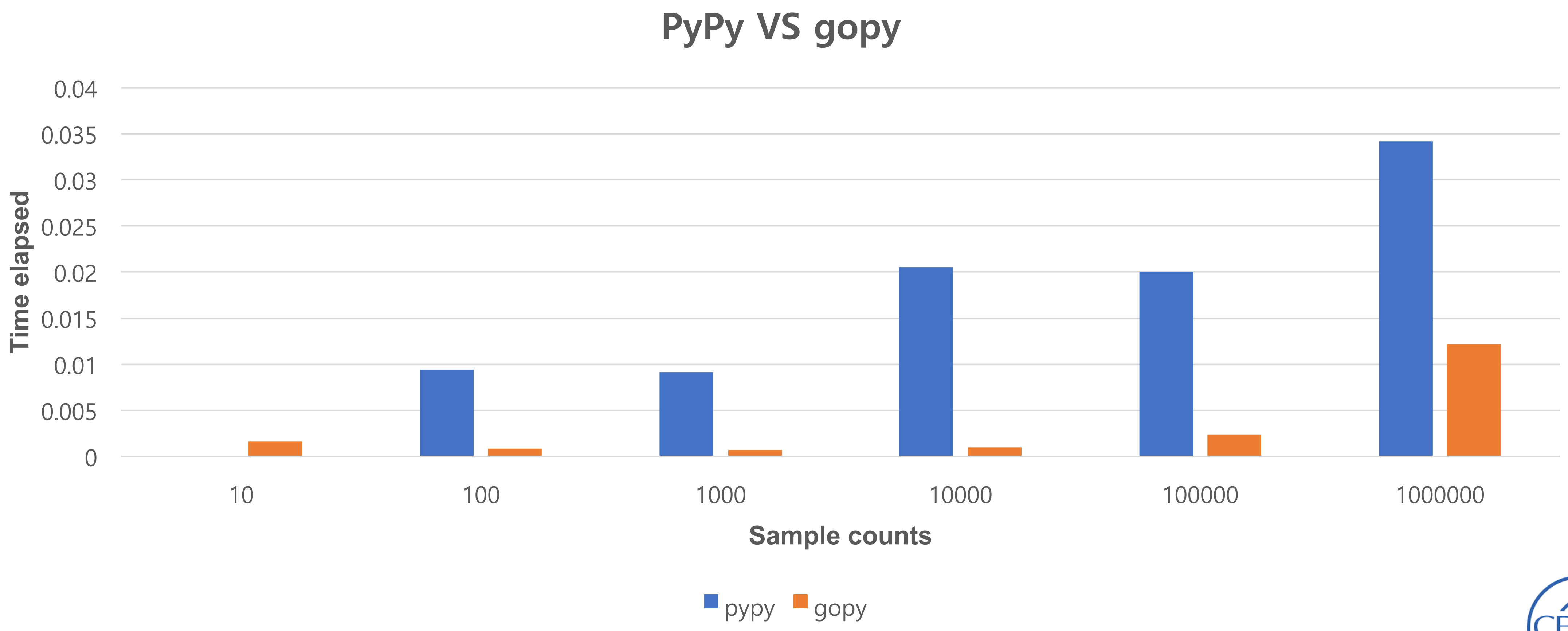
1. **Inspects a Go package**
2. **Extracts the exported types, funcs, vars and consts**
3. **Creates a Go package that cgo exports the exported entities**
4. **Designates which interface should be exported to CFFI.**
5. **Creates a wrapping codes for CFFI by Python.**

```
# pythonization of: calculatePi.GetPI
def GetPI(samples):
    c_samples = _cffi_helper.cffi_cnv_py2c_int(samples)
    cret = _cffi_helper.lib.cgo_func_calculatePi_GetPI(c_samples)
    ret = _cffi_helper.cffi_cnv_c2py_float64(cret)
    return ret
```

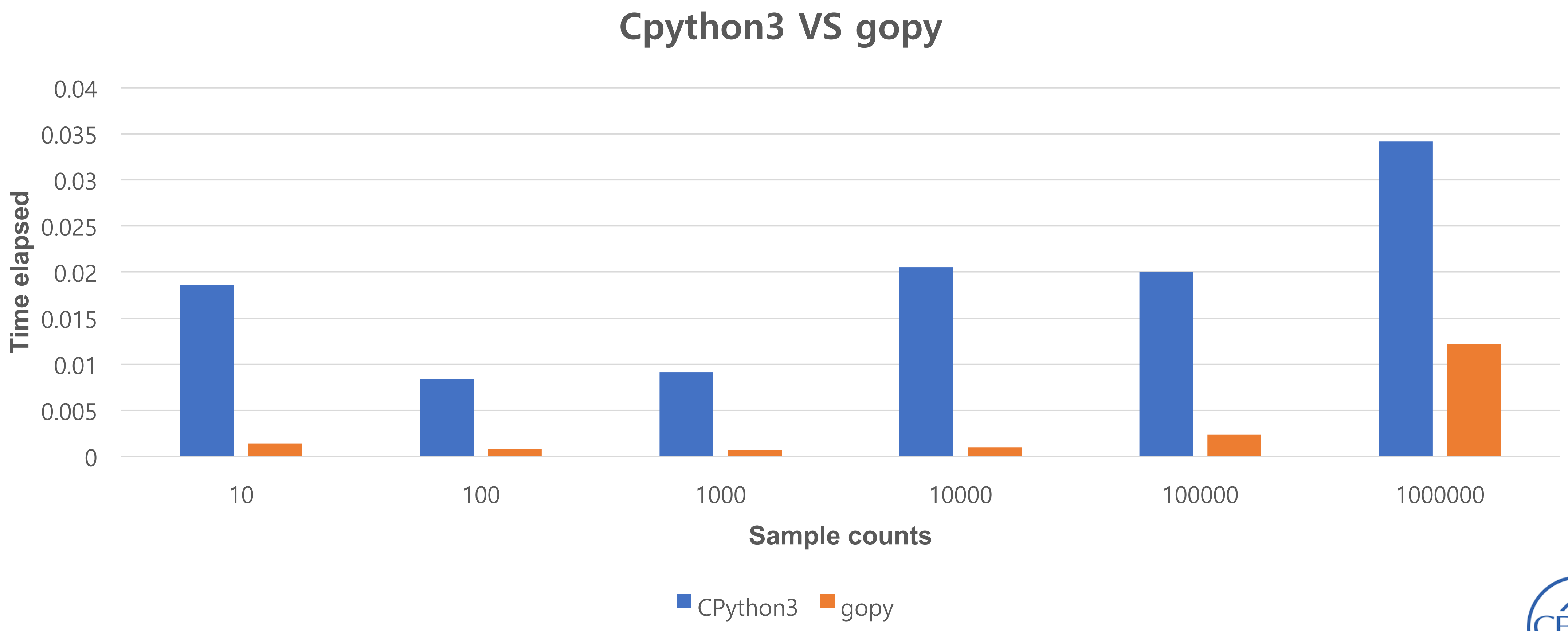
GSoC Project: Support Python3 and PyPy



GSoC Project: Support Python3 and PyPy



GSoC Project: Support Python3 and PyPy



GSoC Project: Project Plan

- Migrate into CFFI library to gencffi*.go for current implementation.
- Implement wrapping of functions with builtin arguments.
- Implement wrapping of functions with slices/arrays of builtin arguments.
- Implement wrapping of functions with user types.
- Detect functions returning a Go error and make them pythonic (raising an Exception).
- Implement wrapping of user types / Go maps / Go interfaces.
- Write documents for English and Korean.

GSoC Project: Goal

- Able to use Go’s awesome features on Python 2/3 and PyPy.

Newcomers are always welcomed

- https://github.com/go-python/gopy
- https://groups.google.com/forum/#!forum/go-python
- https://gophers.slack.com/messages/go-python

Thank you

donghee.na92@gmail.com