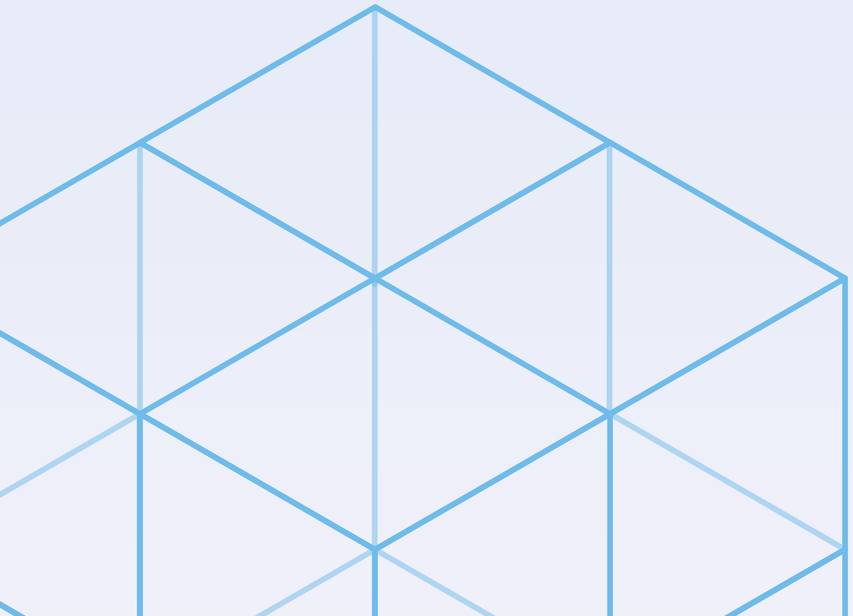
Writing a C Python extension in 2017



Jean-Baptiste Aviat

CO-FOUNDER & CTO at SQREEN

PyCon 2017, PDX

Who am I?

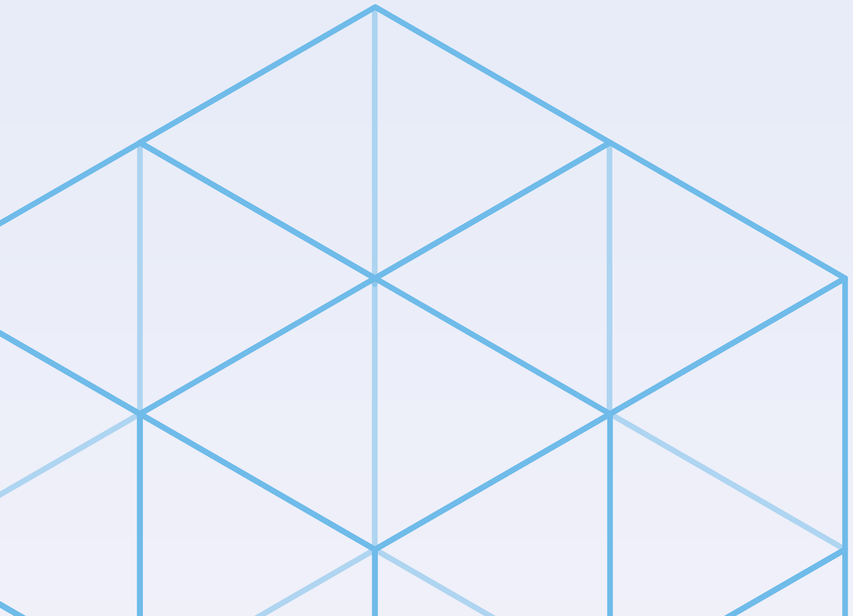
sqreen

Jean-Baptiste Aviat

CTO @SqreenIO (https://sqreen.io)
Former hacker at Apple (Red Team)
Author of PyMiniRacer

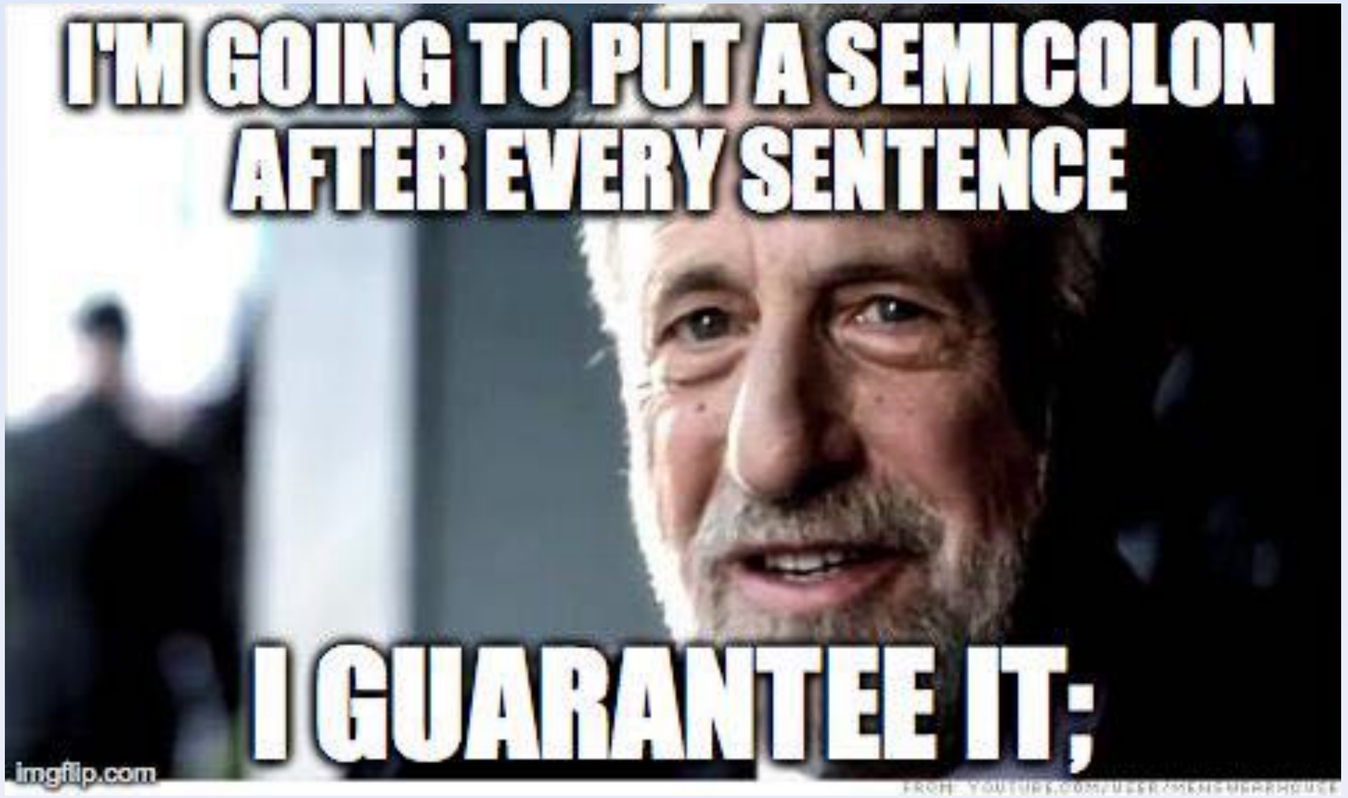
@jbaviat - jb@sqreen.io

Someday... we needed to use V8 from Python.



What we ship:

- is public
- is widely used
- need to be frictionless.


I'M GOING TO PUT A SEMICOLON AFTER EVERY SENTENCE
I GUARANTEE IT;
imgflip.com

Agenda

The need for a binary Python extension

Available choices

Build it (and debug it)

Ship it!

PyMiniRacer: cool JS binding

```
>>> from py_mini_racer import py_mini_racer
>>> ctx = py_mini_racer.MiniRacer()
>>> ctx.eval('1+1')
2
>>> ctx.eval("var x = {company: 'Sqreen'}; x.company")
u'Sqreen'
>>> print ctx.eval(u"'\N{HEAVY BLACK HEART}'")
❤
>>> ctx.eval("var fun = () => ({ foo: 1 });")
>>> ctx.call("fun")
{u'foo': 1}
```

The problem

V8 is C++

How do you run C++ in Python?

We need some kind of binding between these 2 worlds.

Who needs binaries anyway?

many popular packages:

cryptography
numpy
pymongo
psycopg
simplejson
lxml
sqlalchemy...

People do it! Let’s do it too.

What are our goals?

We want to:

- minimize maintenance
- make setup easy
- make testing easy
- have great performance
- have low memory fingerprint

And (obviously)...

- dev time is a constraint

	built-in	pythonic	Python version independant	open to other languages	high throughput capable
CPython	✓	✓			✓
ctypes	✓		✓	✓	
cffi		✓	✓	✓	✓
Cython		✓			✓
SWIG				✓	✓

ctypes

Built into Python

Binary is Python *independant*:

- can be used on any version
- can be used in other languages!

No tight integration to Python

- *not* high throughput capable
- less Pythonic

Complex syntax (C types wrapped in Python...)

Not for C++

C file

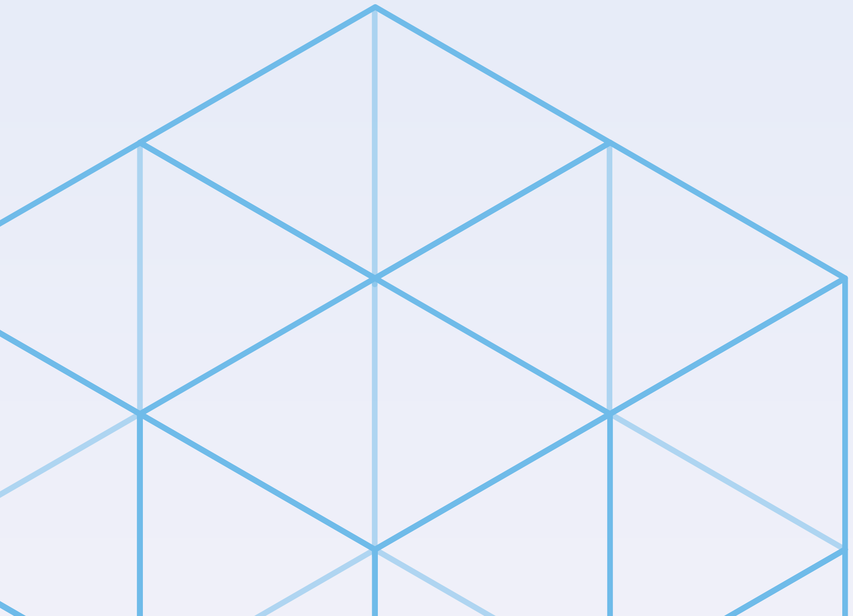
```c
#include <stdio.h>

void hello_world() {
  printf("Hello world!");
}
```

binary object

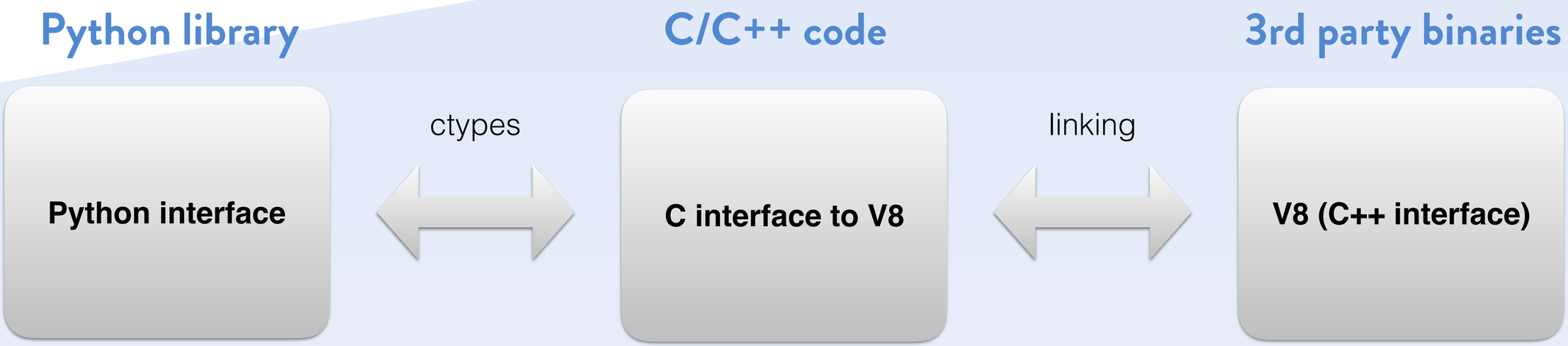
Python interface

```
$ python
>>> path = "./hello.so"
>>> import ctypes
>>> lib = ctypes.cdll.LoadLibrary(path)
>>> lib.hello_world()
Hello world!
```

Build it



Overview



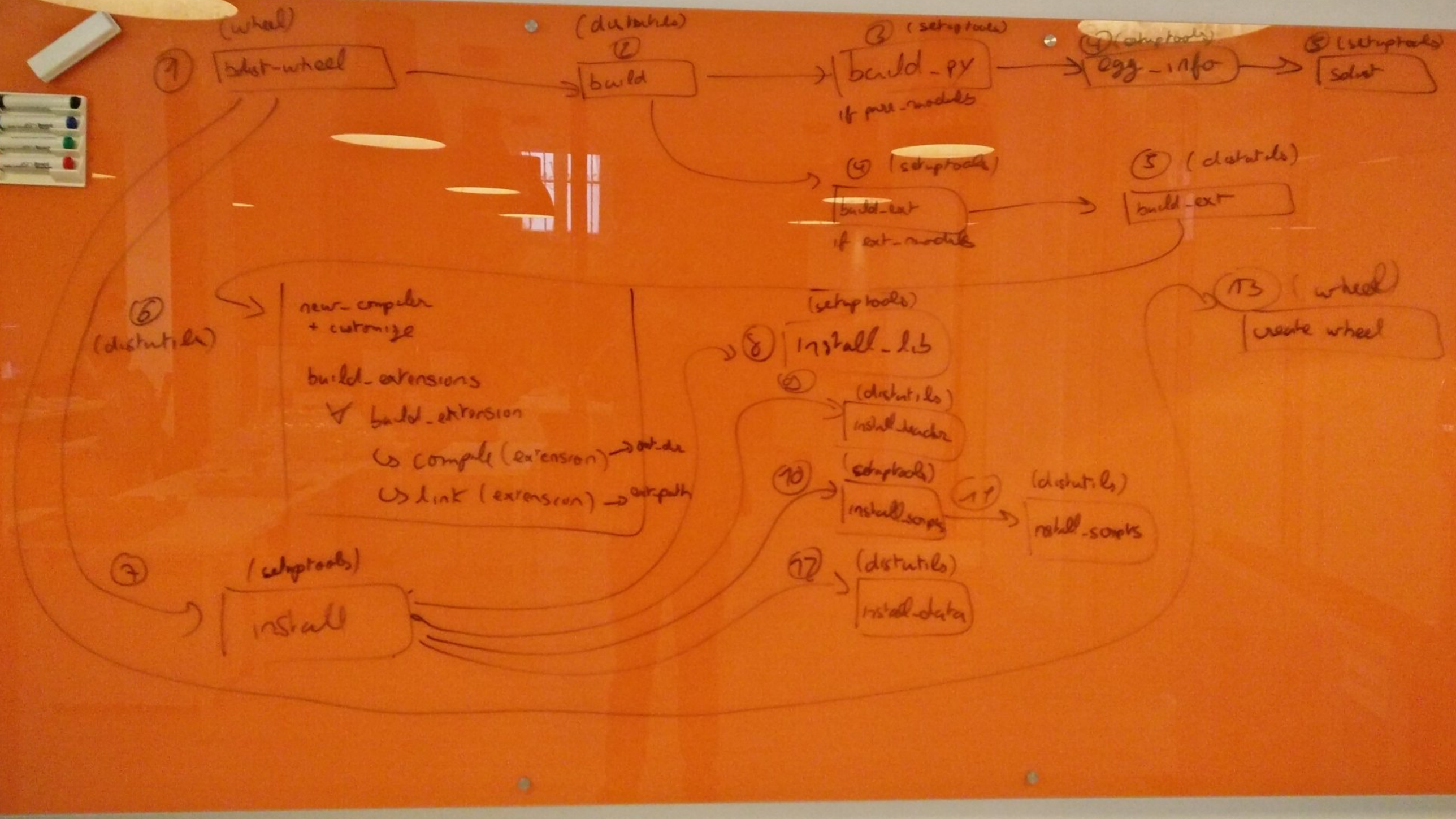

```
import ctypes

class PyMiniRacer(object):
…
```

```
#include <v8.h>

int miniracer_init();
…
```

```
V8 library (libv8.a)
V8 headers (v8.h)
```

How to put this together?

```
$ cat setup.py
from distutils.core import setup, Extension

extension = Extension('hello', ['hello.c'])
setup(name='hello',
      version='1.0',
      ext_modules=[extension])
$ python setup.py build
running build
running build_ext
building 'hello' extension
clang […] -c hello.c -o hello.o
creating build/lib.macosx-10.6-intel-2.7
clang -bundle […] hello.o -o hello.so
```


(wheel)
1 bdist_wheel
(distutils)
2 build
3 (setuptools) build_py
if pure_modules
4 (setuptools) egg_info
5 (setuptools) sdist
6 (setuptools) build_ext
if ext_modules
5 (distutils) build_ext
(distutils)
new_compiler
+ customize
build_extensions
∀ build_extension
compile (extension) → ext_dir
link (extension) → ext_path
(setuptools)
8 install_lib
9 (distutils) install_headers
10 (setuptools) install_scripts
11 (distutils) install_scripts
12 (distutils) install_data
13 (wheel) create wheel
7 (setuptools) install

Crashes?

Python stack trace

```
$ python run_me.py
  File "client.py", line 1227, in lpush
    return self.execute_command('LPUSH', name, *values)
  File "client.py", line 578, in execute_command
    connection.send_command(*args)
  File "connection.py", line 563, in send_command
    self.send_packed_command(self.pack_command(*args))
  File "connection.py", line 538, in send_packed_command
    self.connect()
  File "connection.py", line 442, in connect
    raise ConnectionError(self._error_message(e))
ConnectionError: Error 61 connecting to localhost:6379.
Connection refused.
```

C stack trace

```
$ python run_me.py
Program terminated with signal
SIGSTOP, Aborted.
```


WE NEED TO GO
DEEPER
memegenerator.net

Debugging binaries

Generate *core files* in this way:

```
$ ulimit -c unlimited
$ python run_me.py
[1]    28653 abort (core dumped)
$ ls /cores/
-r--------  1 jb  admin   711M 4 april 01:48 core.12922
```

And just read it

```
$ lldb -c core.28653 (or gdb -c core.28653)
(lldb) bt
* thread #1, stop reason = signal SIGSTOP
  * frame #0: 0x0000106da8b0d mini_racer_extension.bundle`PyMiniRacer_eval_context(ContextInfo*, char*) + 125
    frame #1: 0x0000106da94ed mini_racer_extension.bundle`eval_context + 29
    frame #2: 0x07fff9673ff14 libffi.dylib`ffi_call_unix64 + 76
    frame #3: 0x07fff9674079b libffi.dylib`ffi_call + 923
    frame #4: 0x0000106d48723 _ctypes.so`_ctypes_callproc + 591
    frame #5: 0x0000106d42d44 _ctypes.so`PyCData_set + 2354
    frame #6: 0x000010688e202 Python`PyObject_Call + 99
```

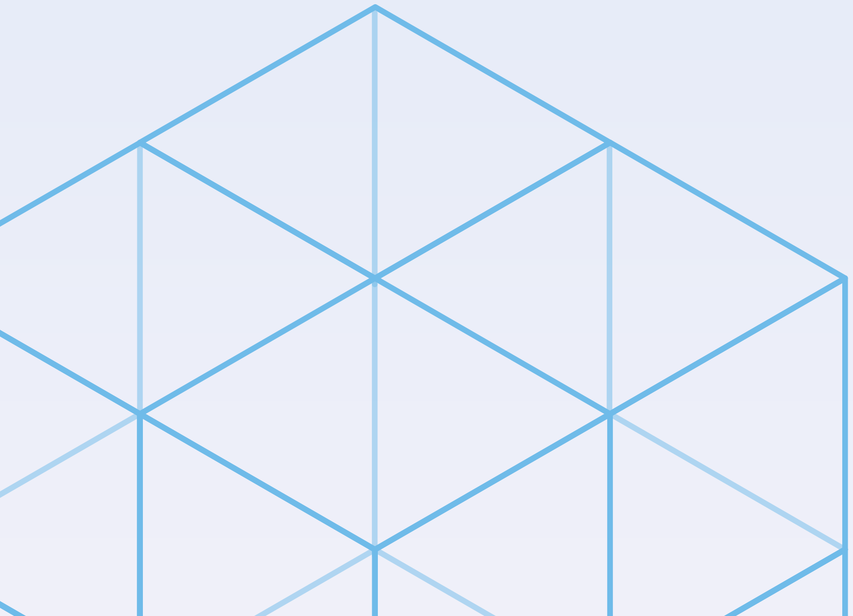
Your C code

Python

On OSX, you cal also check the crash reports here:

```
$ ls /Library/Application\ Support/CrashReporter/
```

What C/C++ compiler are you using?



Memory leaks

Python: 🤦‍♀️

C: 😿

Calling a leaking C function from Python...

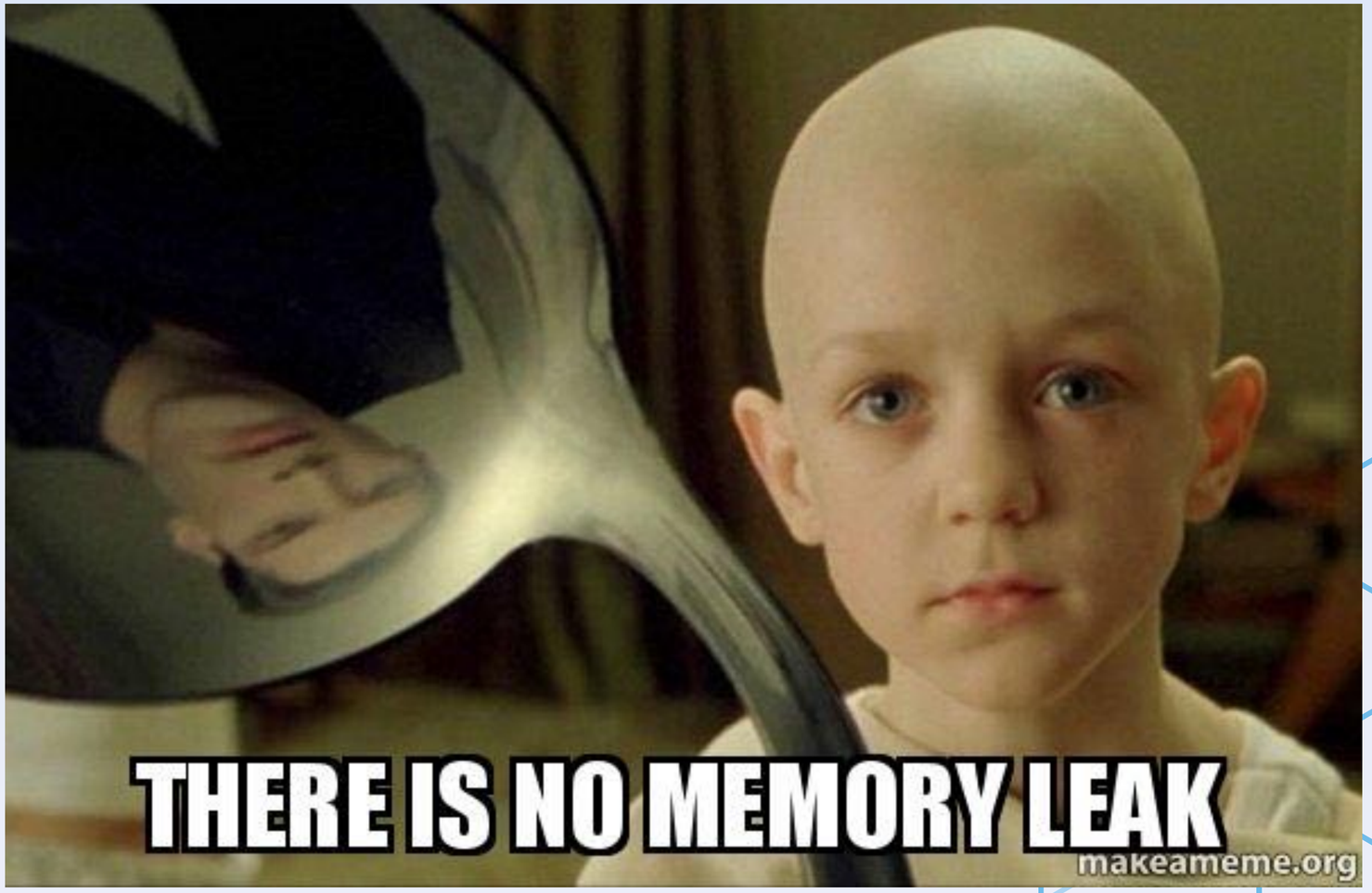
—> you'll never get this memory back.

Valgrind is your friend

```
$ valgrind ./myExtension
```

Rely on clang analyser

```
$ clang --analyze file.c
Warning: memory is never freed
Warning: condition is never true
[…]
```


THERE IS NO MEMORY LEAK
makeameme.org

Other memory issues

Valgrind is (again) your friend

- use after free
- non aligned accesses
- uninitialized accesses

Use clang address sanitiser

```
$ clang --asan file.c
Warning: use after free
[…]
```

Taking checks to the next level

Rely on clang analyser

```
$ clang --analyze file.c
Warning: memory is never freed
Warning: condition is never true
[…]
```

Worth having it in your build system!

Fuzz it!

American Fuzzy Lop: best fuzzer

http://lcamtuf.coredump.cx/afl/

That's **awesome**… but do everything else first.

Abuse the Python unit tests

Unit test in C is *painful* but cool in Python

Do rely on Python's unit test capabilities:

- Test multithreading capabilities
- Test for memory leaks
- Test for performance & performance regressions

Ship it

Ever had trouble installing packages?

Failed to install Python Cryptography package with PIP and setup.py

165

When I try to install the Cryptography package for Python through either `pip install cryptography` or by downloading the package from their site and running `python setup.py`, I get the following error:

```
:26: warning: unknown conversion type character 'z' in format [-
:26: warning: too many arguments for format [-Wformat-extra-args
:30: warning: unknown conversion type character 'z' in format [-
:30: warning
```

pyca / cryptography

Watch

Code | Issues 60 | Pull requests 17 | Projects 0 | Pulse | Graphs

Having trouble with pip install cryptography #2872

Closed **mrbm** opened this issue on 14 Apr 2016 · 11 comments

mrbm commented on 14 Apr 2016

Having some strange errors that are blocking me from installing cryptography

Python Cryptography Error Message

Server

luketheterrible

Hey Everybody!

but when I did, nothing was updated or installed.

created	last reply	1	1.4k	2	4
Jan '16	Feb '16	reply	views	users	links


NO WAIT, I'M
NOT READY...
imgflip.com

This packages rely on C/C++ code.

They need to build this code.

This is done during pip install.

Python packaging history

Python 2.4	2004	**sdist (source distribution)**
		eggs
Python 3.3	2012	**wheels** → **(built distribution)**
Python 3.6	2016	**—> manylinux wheels** ❤️

manylinux wheels

Python standard: PEP503

Compatible on most (real world) Linux

Only in pip >= 8.1

Need to build on many platforms

Binaries need to be built on

🚨🚨CentOS 5🚨🚨

Wheels or compiler?

Wheels

- iso builds (crash can be reproduced)
- you need to maintain *many* packages

Compiler

- one build per user
- only one package
- but harder to install…

Many packages... How many?

2.x
3.5
3.6
3.7

→

- wide Unicode
- regular Unicode

Linux 32/64 (ARM?)
macOS 32/64
maybe Windows 32/64 (ARM?)

→ 3+1+1

→ 2+2

} 20 wheels to publish


"Help me!" Exclamation mark.

Wheels or compiler?

Preferred way:

- pubish the wheels
- also publish the non compiled version

An you can do it lean...

Why CentOS 5?

A compiled program relies on 3rd party libraries:

- libc
- libstdc++
- ...

a program compiled with libc 2.1 won't run with libc 2.20

Yes: something built on Ubuntu 16 may not run on Ubuntu 14

Why CentOS 5 (again)?

One of the oldest libc that can be found

It is said mandatory by PEP503

- there is no *need to comply*
- but your wheels won't be as compatible as possible

PEP503 provides CentOS 5 Dockerfile with Python versions

https://github.com/pypa/manylinux#docker-images

Testing binaries

The wheel was built on old Linux

Now let's *test it* on other distributions.

Docker is will help:

```
$ for tag in 12.04 14.04 16.04; do
    docker run --rm ubuntu:${tag} bash -c "pip install mypkg; mypkg-tests »
    if [ $? -ne 0 ]; then echo "Failure on ubuntu:${tag}"; fi
done;
```

Profit!

Questions?