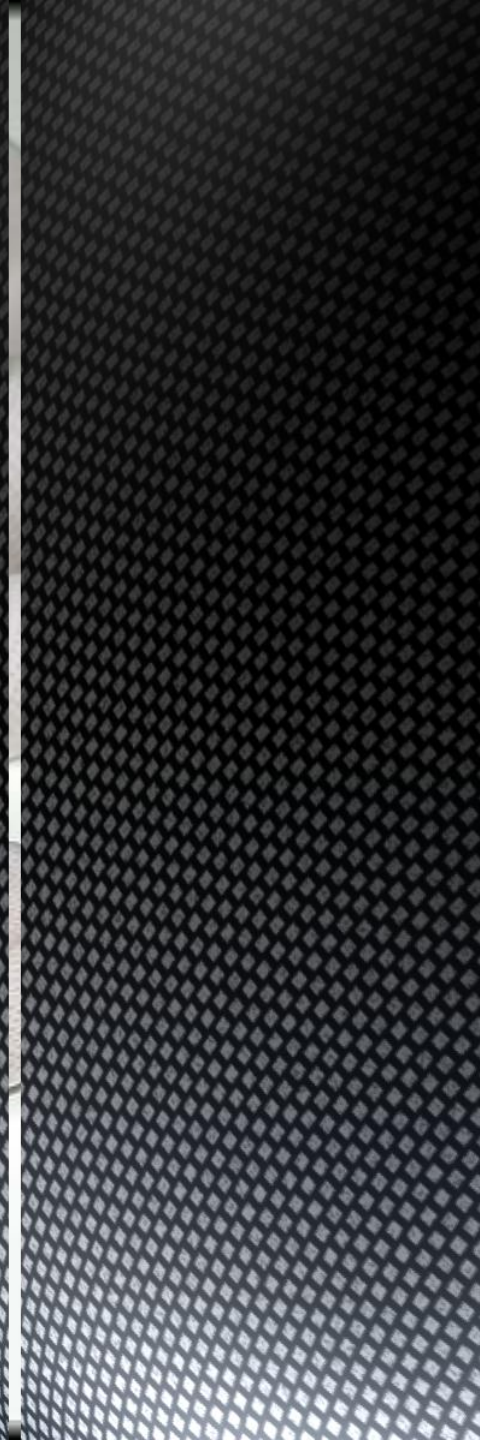


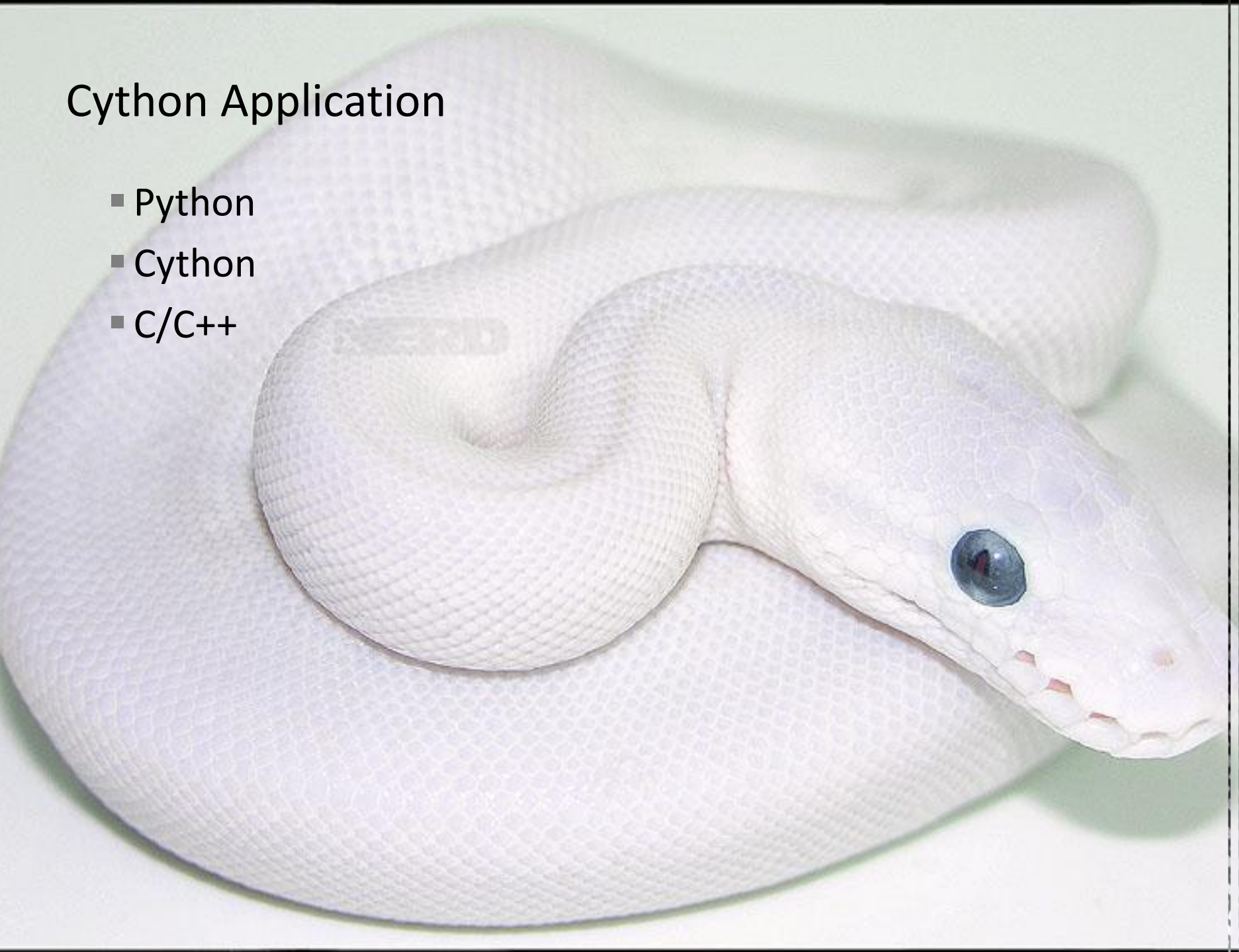
Multi-Level Debugging for Cython

Mark Florisson



Cython Application

- Python
- Cython
- C/C++

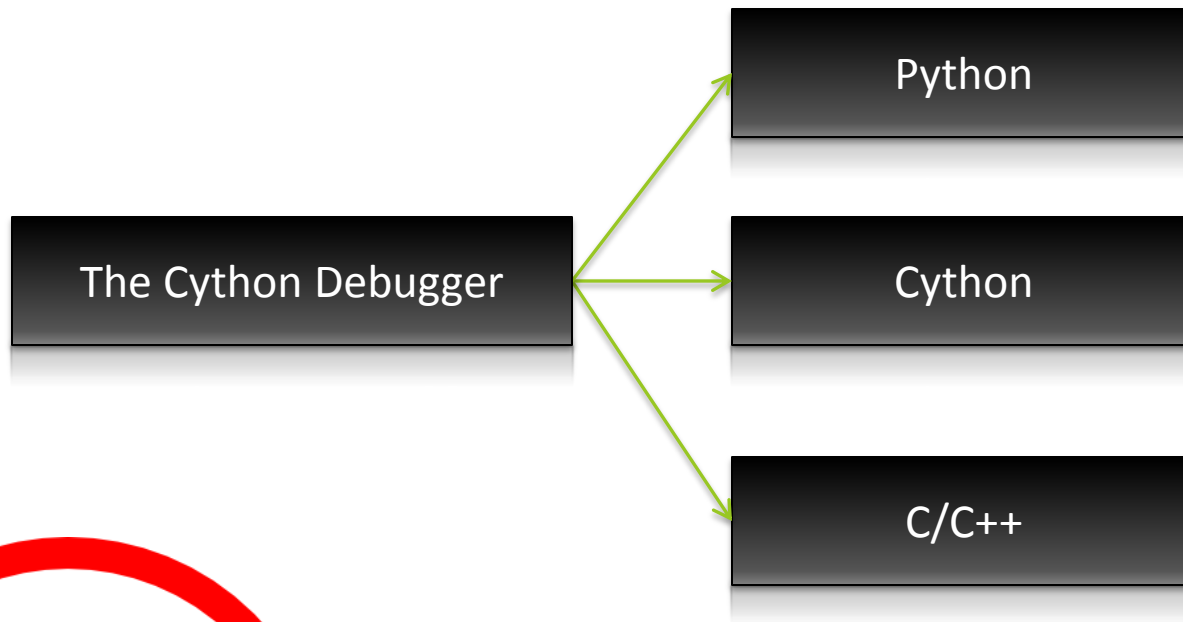


How to debug our application?

Language	Debugger
Python	pdb
C/C++, Fortan, Ada, Obj-C	gdb
Cython	???

- Problems?
 - No Cython Debugger
 - No multi-level debugger...

s/debugger/multi-level debugger/



Show me some code!

```
1 import sys
2 import cython_io
3
4 def hello(fd):
5     cython_io.write(fd, "Hello World!\n")
6
7 if __name__ == '__main__':
8     hello(int(sys.argv[1]))
```


Python: hello.py

```
1 from libc.errno cimport errno as cerrno
2 import os
3
4 cdef extern from "fdwrite.h":
5     size_t fdwrite(int fd, char *buf, size_t bufsize)
6
7
8 def write(fd, string):
9     cdef size_t length = len(string)
10
11     if fdwrite(fd, string, length) < length:
12         raise IOError(errno, os.strerror(errno))
```

Cython: cython_io.pyx

Don't forget to link in the C code...

```
1 #include <unistd.h>
2
3 size_t
4 fdwrite(int fd, const char *buf, size_t bufsz)
5 {
6     size_t total = 0;
7     ssize_t written;
8
9     do {
10         written = write(fd, buf, bufsz);
11         if (written == -1)
12             return total;
13
14         total += written;
15         buf += written;
16     } while (written < bufsz);
17
18     return total;
19 }
```



▪ fdwrite.c

Build & Debug

```
1 from distutils.core import setup
2 from Cython.Distutils import build_ext
3 from Cython.Distutils.extension import Extension
4
5 setup(ext_modules=[Extension('cython_io', ['fdwrite.c', 'cython_io.pyx'])],
6       cmdclass={'build_ext': build_ext})
```

Python: setup.py

```
~/helloworld > python setup.py build_ext --inplace --pyrex-gdb
running build_ext
cythoning cython_io.pyx to cython_io.c
building 'cython_io' extension
```

Build

```
~/helloworld > ls cython_debug
cython_debug_info_cython_io  interpreter
~/helloworld > cat cython_debug/interpreter
/usr/bin/python%  
%
```


Running the Debugger

```
~/helloworld ➤ cygdb
GNU gdb (GDB) Fedora (7.2-26.fc14)
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(gdb) cy run hello.py 1
Hello World!
Program exited normally.
(gdb) cy run hello.py 3
Traceback (most recent call last):
  File "hello.py", line 8, in <module>
    hello(int(sys.argv[1]))
  File "hello.py", line 5, in hello
    cython_io.write(fd, "Hello World!\n")
  File "cython_io.pyx", line 13, in cython_io.write (cython_io.c:582)
    raise IOError(errno, os.strerror(errno))
IOError: [Errno 9] Bad file descriptor
Program exited with code 01.
(gdb)
```


Feature List

Type	Command
Execution control	cy run cy cont cy step cy next cy finish
Symbolic debugging	cy break cy locals cy globals cy print cy set
Stack navigation	cy up cy down cy select
Source code listing	cy list
Backtrace formatting	cy bt
Code execution	cy exec

```
(gdb) cy break -p hello
Breakpoint 1 at 0x80d670c: file Python/ceval.c, line 562.
(gdb) cy run hello.py 1
4     def hello(fd):
(gdb) cy step
5         cython_io.write(fd, "Hello World!\n")
(gdb)
8     def write(fd, string):
(gdb)
9         cdef size_t length = len(string)
(gdb)
11        if fdwrite(fd, string, length) < length:
(gdb) cy locals
fd      = 1
length = (size_t) 13
string = 'Hello World!\n'
(gdb) cy print *string
*string = (PyObject) {
    ob_refcnt = 3,
    ob_type = 0x817c8a0
}
(gdb) cy step
6     size_t total = 0;
(gdb) cy list
1     #include <unistd.h>
2
3     size_t
4     fdwrite(int fd, const char *buf, size_t bufsiz)
5     {
> 6         size_t total = 0;
7         ssize_t written;
8
9         do {
10            written = write(fd, buf, bufsiz);
(gdb) □
```

Backtraces

```
(gdb) cy bt
#7  0x00000000080d6700 in <module>() at hello.py:8
      8      hello(int(sys.argv[1]))
#10 0x00000000080d6700 in hello() at hello.py:5
      5      cython_io.write(fd, "Hello World!\n")
#12 0x0000000000112100 in write() at /home/mark/helloworld/cython_io.pyx:11
      11      if fdwrite(fd, string, length) < length:
#13 0x00000000001120a0 in fdwrite() at /home/mark/helloworld/fdwrite.c:6
      6      size_t total = 0;
```

```
(gdb) cy bt -a
#0 0x0000000008057990 in main() at /home/mark/source/code/py/_python-2.6/Modules/python.c:23
      23      return Py_Main(argc, argv);
#1 0x0000000008057bd0 in Py_Main() at /home/mark/source/code/py/_python-2.6/Modules/main.c:577
      577      sts = PyRun_AnyFileExFlags(
#2 0x00000000080fef20 in PyRun_SimpleFileExFlags() at /home/mark/source/code/py/_python-2.6/Python/pythonrun.c:941
      941      v = PyRun_FileExFlags(fp, filename, Py_file_input, d, d,
#3 0x00000000080fea60 in PyRun_FileExFlags() at /home/mark/source/code/py/_python-2.6/Python/pythonrun.c:1337
      1337      ret = run_mod(mod, filename, globals, locals, flags, arena);
#4 0x0000000000000000 in run_mod() at /home/mark/source/code/py/_python-2.6/Python/pythonrun.c:1351
      1351      v = PyEval_EvalCode(co, globals, locals);
#5 0x00000000080dcd10 in PyEval_EvalCode() at /home/mark/source/code/py/_python-2.6/Python/ceval.c:541
      541      return PyEval_EvalCodeEx(co,
#6 0x00000000080dc430 in PyEval_EvalCodeEx() at /home/mark/source/code/py/_python-2.6/Python/ceval.c:3000
      3000      retval = PyEval_EvalFrameEx(f, 0);
#7 0x00000000080d6700 in <module>() at hello.py:8
      8      hello(int(sys.argv[1]))
#8 0x0000000000000000 in call_function() at /home/mark/source/code/py/_python-2.6/Python/ceval.c:3771
      3771      x = fast_function(func, pp_stack, n, na, nk);
#9 0x0000000000000000 in fast_function() at /home/mark/source/code/py/_python-2.6/Python/ceval.c:3836
      3836      retval = PyEval_EvalFrameEx(f, 0);
#10 0x00000000080d6700 in hello() at hello.py:5
      5      cython_io.write(fd, "Hello World!\n")
#11 0x0000000000000000 in call_function() at /home/mark/source/code/py/_python-2.6/Python/ceval.c:3750
      3750      C_TRACE(x, PyCFunction_Call(func, callargs, NULL));
#12 0x0000000000112100 in write() at /home/mark/helloworld/cython_io.pyx:11
      11      if fdwrite(fd, string, length) < length:
#13 0x00000000001120a0 in fdwrite() at /home/mark/helloworld/fdwrite.c:6
      6      size_t total = 0;
```

Code Execution & Data Modification

```
(gdb) cy break write
Function "__pyx_pf_9cython_io_write" not defined.
Breakpoint 1 (__pyx_pf_9cython_io_write) pending.
(gdb) cy run hello.py 3
8     def write(fd, string):
(gdb) cy next
9         cdef size_t length = len(string)
(gdb) cy locals
fd      = 3
string = 'Hello World!\n'
(gdb) cy set fd = PyInt_FromLong(1)
(gdb) cy finish
Hello World!
5         cython_io.write(fd, "Hello World!\n")
(gdb) cy exec fd, cython_io
(3, <module 'cython_io' from '/home/mark/helloworld/cython_io.so'>)
(gdb) cy exec print spam
Traceback (most recent call last):
  File "<string>", line 1, in <module>
NameError: name 'spam' is not defined
(gdb) cy cont
Program exited normally.
(gdb) □
```


Exceptions & Control Flow

```
(gdb) cy break -p hello
Breakpoint 1 at 0x80d670c: file Python/ceval.c, line 562.
(gdb) cy run hello.py 3
4     def hello(fd):
(gdb) cy next
5         cython_io.write(fd, "Hello World!\n")
(gdb)
An exception was raised: exceptions.IOError(9, 'Bad file descriptor')
8         hello(int(sys.argv[1]))
(gdb) cy bt
#7  0x000000000080d6700 in <module>() at hello.py:8
      8         hello(int(sys.argv[1]))
(gdb) call PyErr_Clear()
(gdb) return $cy_eval("None")
(gdb) cy cont
Program exited normally.
(gdb) □
```

Watchpoints

```
(gdb) cy next
2      mylist = []
(gdb)
3      for x in 1, 2, 3:
(gdb) print $cy_cname("mylist")
$1 = "__pyx_v_mylist"
(gdb) ptype PyListObject
type = struct {
    Py_ssize_t ob_refcnt;
    struct _typeobject *ob_type;
    Py_ssize_t ob_size;
    PyObject **ob_item;
    Py_ssize_t allocated;
}
(gdb) watch ((PyListObject *) __pyx_v_mylist)->ob_size
Hardware watchpoint 2: ((PyListObject *) __pyx_v_mylist)->ob_size
(gdb) cy cont
Old value = 0
New value = 1
71      self->allocated = new_allocated;
(gdb) cy up
#19 0x0000000000308bb0 in func() at /home/mark/helloworld/watch.pyx:4
      4      mylist.append(x)
(gdb) □
```

GNU Debugger



GDB Python API

GDB(Python)

```
(gdb) python import sys; print sys.version
2.7 (r27:82500, Sep 16 2010, 18:03:06)
[GCC 4.5.1 20100907 (Red Hat 4.5.1-3)]
```

GDB(Python(GDB))

```
(gdb) python gdb.execute("cy print *string")
*string = (PyObject) {
  ob_refcnt = 3,
  ob_type = 0x817c8a0
}
```

Inspect the C stack

```
(gdb) python print gdb.selected_frame().find_sal()
symbol_and_line for cython_io.c, line 519
```

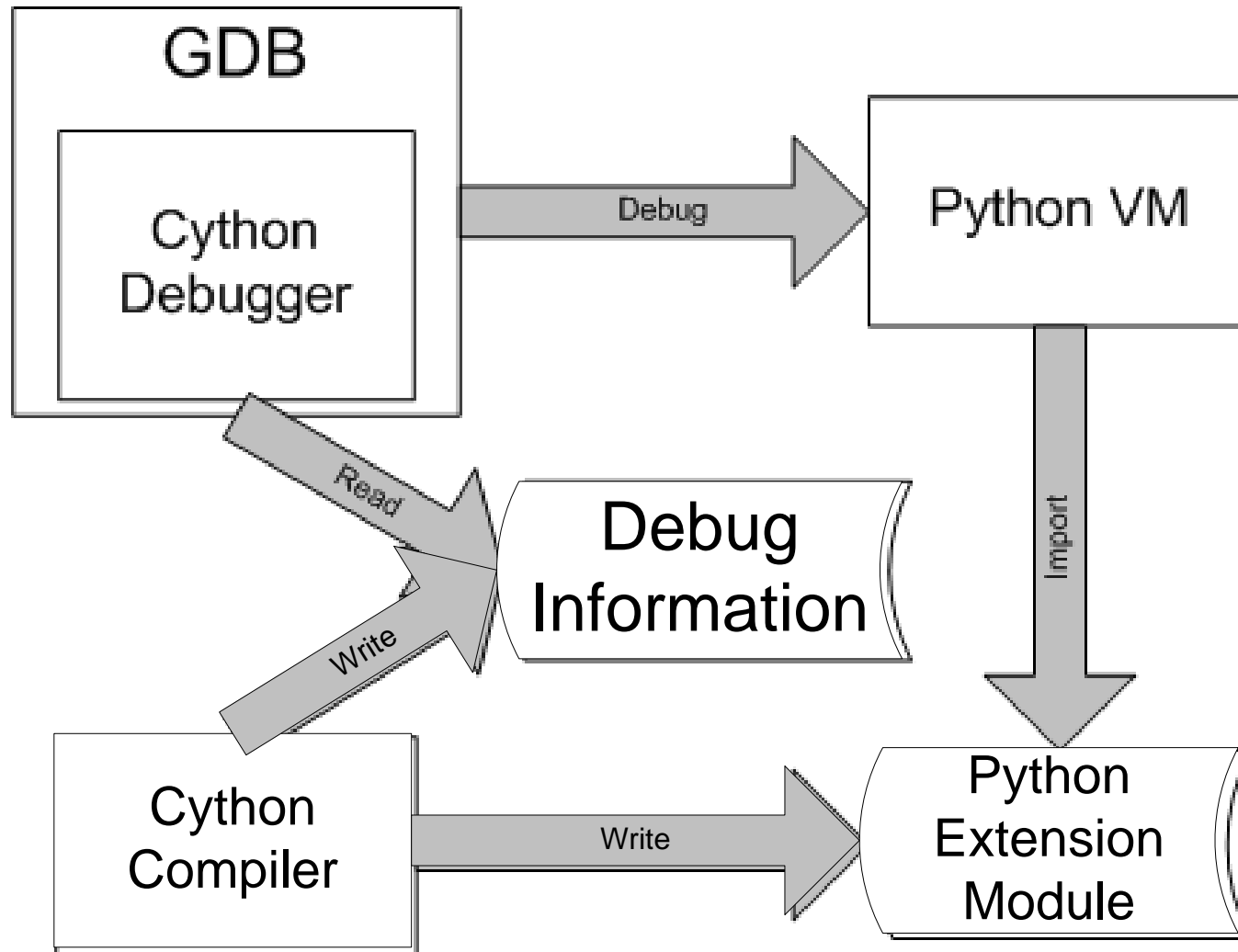
Inspect Values and Types

```
(gdb) python
>frame = gdb.selected_frame()
>print frame.read_var("__pyx_v_string")["ob_refcnt"]
>print frame.read_var("__pyx_v_string").type
>end
3
PyObject *
```

Pretty Printers

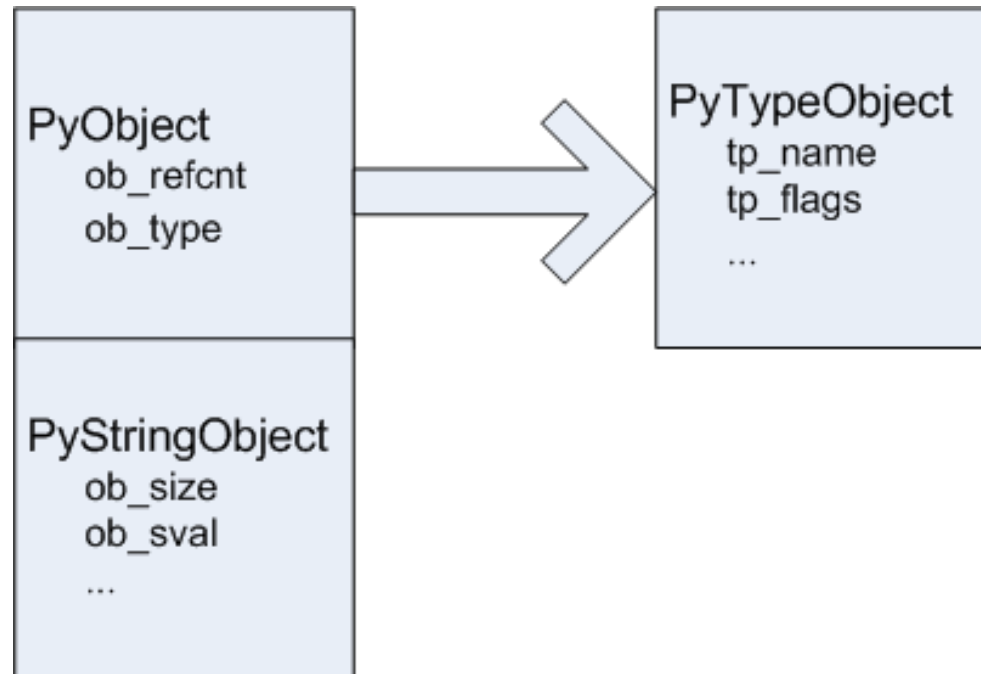
```
(gdb) print /r $cy_cvalue("string")
$2 = (PyObject *) 0x8205548
```


Implementation



The Python Debugger

- Extension of Tools/gdb/libpython.py
 - Written by David Malcolm
 - Additions: breakpoints, execution control, code execution
- **Debug information is dynamic**



```
(gdb) set $s = (PyStringObject *) $cy_cvalue("string")
(gdb) print $s->ob_sval@$s->ob_size
$1 = {"H", "e", "l", "l", "o", " ", "W", "o", "r", "l", "d", "!", "\n"}
```

Cool stuff

- GDB goodies

- Conditional breakpoints ('cond')
- Execute anything on breakpoint hit ('commands')
- 'display' a variable on every stop

- Attach to a running process

```
cygdb -- python `pidof python`
```

- Closure support

- Entirely safe!

- Compatible with Python 2 and 3

- Pretty prints most builtin types

- Add your own!

```
(gdb) cy list
1      def outer(a):
2          def inner(b):
> 3              print a, b
4          return inner
5      outer(1)(2)
(gdb) cy locals
a = 1
b = 2
```

```
(gdb) cy locals
mydict = {'spam': 'ham'}
mylist = ['eggs', 'bacon']
myset = {'python', 'cython'}
mytuple = (0, 1, None)
```

Final Notes

- Access macros: -g3
- Reference counting:
 - Py_IncRef/Py_DecRef
- “unwind-on-signal”

```
(gdb) set unwindonsignal on
(gdb) start
Temporary breakpoint 2 at 0x80483bd: file test.c, line 4.

Temporary breakpoint 2, main () at test.c:4
4      puts("execute this!");
(gdb) call abort()

Program received signal SIGABRT, Aborted.
0x00110416 in __kernel_vsyscall ()
The program being debugged was signaled while in a function called from GDB.
GDB has restored the context to what it was before the call.
To change this behavior use "set unwindonsignal off".
Evaluation of the expression containing the function
(abort) will be abandoned.
(gdb) cont
execute this!

Program exited normally.
(gdb) □
```


Conclusion

- Coherent multi-language debugging
- Powerful debugger
- Easy & convenient

More...

- Part of Cython 0.14
- Patch to Python 3.3
- More details on docs.cython.org
- GDB API: <http://sourceware.org/gdb/onlinedocs/gdb/Python-API.html>
- PyCon 2011: talks by David Malcolm:
 - Dude, Where's My RAM? A deep dive into how Python uses memory (<http://blip.tv/file/4878749>)
 - Using Python to debug C and C++ code (<http://blip.tv/file/4877544?filename=Pycon-PyCon2011UsingPythonToDebugCAndCCodeUsingGdb402.m4v>)

Future work...

- Bringing debugger functionality to frontends
 - (Eclipse CDT, Emacs, Insight, DDD, etc)

