A FASTER PYTHON?
YOU HAVE THESE CHOICES

Paul Ross AHL

MAN AHL

https://twitter.com/manahltech

- London based systematic hedge fund since 1987
- $18.8bn Funds Under Management (2017-03-31)
- We are active in 400+ markets in 40+ countries
- We take ~3bn market data points each day
    - https://github.com/manahl/arctic
- 153 people, >20 first languages. And Python!

SECTIONS OF THIS TALK

- Introduction and scope
- A technology taxonomy
- Evaluation criteria

SECTIONS OF THIS TALK

- *Introduction and scope*
- A technology taxonomy
- Evaluation criteria

WHAT THIS IS

- A tour of faster Python alternatives for general purpose computing
- A way of evaluating alternatives
- Reflect on the trade-offs between performance, cost, maintainability etc.

WHAT THIS IS NOT

- Numpy, Pandas
- Concurrency, cacheing etc.
- Definitive recommendations for you
- The only benchmarks here are fake ones

WHY DO WE HAVE TO DO THIS?

- Python is interpreted, every line is eval'd
- Dynamic typing
- Running in a virtual machine
- No JIT (but see PEP-0523)
- Takes little optimisation possibilities compared with compiled languages

THE NEED FOR SPEED?

- Is Python fast enough?
- Where should it go faster?
- Algorithms
- Data structures
- "Premature optimization is the root of all evil (or at least most of it) in programming." - Donald Knuth, *Communications of the ACM*, 1974

SECTIONS OF THIS TALK

- Introduction and scope
- *A technology taxonomy*
- Evaluation criteria

A LOT OF CHOICE…

PARAKEET

Python/C API

Microsoft / Pyjion

boost C++ LIBRARIES

THE PYSTON BLOG

16.16. ctypes

CFFI

SWIG

Cython

shedskin / shedskin

Pythran

pypy

Numba

pybind11

INTEL® DISTRIBUTION FOR PYTHON*

PERFECTION AT LAST

- Can run Python code directly
- No maintenance overhead
- Works with all Python versions, all library code
- Free
- Fully supported
- No bugs
- Perfect debug story
- 100x faster

TECHNOLOGY TAXONOMY

- Little or no code change from Python code
- Some code change
- A different language: C++, Rust etc.

TECHNOLOGY TAXONOMY

- *Little or no code change from Python code*
- Some code change
- A different language: C++, Rust etc.

NO CODE CHANGE - 1x TO 8x

- Python
- Cython (not optimised)
- Pypy
- Shedskin
- Pyston

CYTHON 1.3x

http://cython.org/

```
import math

def std_dev(a):
    mean = sum(a) / len(a)
    sq_diff = [(v - mean)**2 for v in a]
    return math.sqrt(sum(sq_diff) / len(a))
```

CYTHON 1.3x

http://cython.org/

```python
import math

def std_dev(a):
    mean = sum(a) / len(a)
    sq_diff = [(v - mean)**2 for v in a]
    return math.sqrt(sum(sq_diff) / len(a))
```

CYTHON

http://cython.org/

```
/* "cStdDev.pyx":14
 *
 * def pyStdDev(a):
 *     mean = sum(a) / len(a)              # <<<<<<<<<<<<<<
 *     sq_diff = [(v - mean)**2 for v in a]
 *     return math.sqrt(sum(sq_diff) / len(a))
 */
  __pyx_t_1 = PyTuple_New(1); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 14, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __Pyx_INCREF(__pyx_v_a);
  __Pyx_GIVEREF(__pyx_v_a);
  PyTuple_SET_ITEM(__pyx_t_1, 0, __pyx_v_a);
  __pyx_t_2 = __Pyx_PyObject_Call(__pyx_builtin_sum, __pyx_t_1, NULL); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 14, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_t_3 = PyObject_Length(__pyx_v_a); if (unlikely(__pyx_t_3 == -1)) __PYX_ERR(0, 14, __pyx_L1_error)
  __pyx_t_1 = PyInt_FromSsize_t(__pyx_t_3); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 14, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_4 = __Pyx_PyNumber_Divide(__pyx_t_2, __pyx_t_1); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 14, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_4);
  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_v_mean = __pyx_t_4;
  __pyx_t_4 = 0;
```

CYTHON

http://cython.org/

```
/* "cStdDev.pyx":14
 *
 * def pyStdDev(a):
 *     mean = sum(a) / len(a)             # <<<<<<<<<<<<<<
 *     sq_diff = [(v - mean)**2 for v in a]
 *     return math.sqrt(sum(sq_diff) / len(a))
 */
  __pyx_t_1 = PyTuple_New(1); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 14, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __Pyx_INCREF(__pyx_v_a);
  __Pyx_GIVEREF(__pyx_v_a);
  PyTuple_SET_ITEM(__pyx_t_1, 0, __pyx_v_a);
  __pyx_t_2 = __Pyx_PyObject_Call(__pyx_builtin_sum, __pyx_t_1, NULL); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 14, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_t_3 = PyObject_Length(__pyx_v_a); if (unlikely(__pyx_t_3 == -1)) __PYX_ERR(0, 14, __pyx_L1_error)
  __pyx_t_1 = PyInt_FromSsize_t(__pyx_t_3); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 14, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_4 = __Pyx_PyNumber_Divide(__pyx_t_2, __pyx_t_1); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 14, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_4);
  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_v_mean = __pyx_t_4;
  __pyx_t_4 = 0;
```

CYTHON

http://cython.org/

```
/* "cStdDev.pyx":14
 *
 * def pyStdDev(a):
 *     mean = sum(a) / len(a)             # <<<<<<<<<<<<<<
 *     sq_diff = [(v - mean)**2 for v in a]
 *     return math.sqrt(sum(sq_diff) / len(a))
 */
  __pyx_t_1 = PyTuple_New(1); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 14, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __Pyx_INCREF(__pyx_v_a);
  __Pyx_GIVEREF(__pyx_v_a);
  PyTuple_SET_ITEM(__pyx_t_1, 0, __pyx_v_a);
  __pyx_t_2 = __Pyx_PyObject_Call(__pyx_builtin_sum, __pyx_t_1, NULL); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 14, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_t_3 = PyObject_Length(__pyx_v_a); if (unlikely(__pyx_t_3 == -1)) __PYX_ERR(0, 14, __pyx_L1_error)
  __pyx_t_1 = PyInt_FromSsize_t(__pyx_t_3); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 14, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_4 = __Pyx_PyNumber_Divide(__pyx_t_2, __pyx_t_1); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 14, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_4);
  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_v_mean = __pyx_t_4;
  __pyx_t_4 = 0;
```

PYPY 7x

https://pypy.org/

- Just-in-time compiler
- Drop in replacement for Python 2.7, 3.5
- CPython API in beta (but supports CFFI)
- Not completely compatible
- “If you want your code to run faster, you should probably just use PyPy.” - Guido van Rossum

SHEDSKIN

https://github.com/shedskin/shedskin

- Automatic type inferencing to generate C code
- Python 2.4 - 2.6
- Little activity in the past year

PYSTON

https://github.com/dropbox/pyston

- LLVM based JIT compiler
- Backed by Dropbox
- Python 2.7 only
- No Mac OS X
- Project suspended January 2017

TECHNOLOGY TAXONOMY

- Little or no code change from Python code change
- *Some code change*
- A different language: C++, Rust etc.

SOME CODE CHANGE - 10x TO 100x

- Cython - optimised
- Numba (LLVM lite)
- Parakeet
- Pythran

CYTHON - OPTIMISED 62x

http://cython.org/

```python
import math

def std_dev(a):
    mean = sum(a) / len(a)
    sq_diff = [(v - mean)**2 for v in a]
    return math.sqrt(sum(sq_diff) / len(a))
```

```cython
cdef extern from "math.h":
    double sqrt(double m)

from numpy cimport ndarray
cimport numpy as np
cimport cython

@cython.boundscheck(False)
def stdDev_05(ndarray[np.float64_t, ndim=1] a not None):
    cdef Py_ssize_t i
    cdef Py_ssize_t n = a.shape[0]
    cdef double m = 0.0
    for i in range(n):
        m += a[i]
    m /= n
    cdef double v = 0.0
    for i in range(n):
        v += (a[i] - m)**2
    return sqrt(v / n)
```

NUMBA

https://numba.pydata.org/

- Backed by Continuum Analytics
- JIT compiler
- Python 2.7, 3.4+
- numpy 1.7 to 1.11

```
from numba import jit
from numpy import arange

@jit
def sum2d(arr):
    M, N = arr.shape
    result = 0.0
    for i in range(M):
        for j in range(N):
            result += arr[i,j]
    return result

a = arange(9).reshape(3,3)
print(sum2d(a))
```

PARAKEET

https://github.com/cournape/parakeet

- JIT compiler
- Subset of Python 2.7 only
- Supports numpy
- Little activity over last 4 years

```python
from parakeet import jit

@jit
def fast(x, alpha = 0.5, beta = 0.3):
    y = np.empty_like(x)
    for i in xrange(len(x)):
        y[i] = np.tanh(x[i] * alpha + beta)
    return x
```

PYTHRAN

https://pythonhosted.org/pythran/

- Annotate functions
- Use Pythran to generate C++
- Focus on scientific computing
- Supports numpy
- Support for Python 2.7, 3 (beta)

```
def zero(n,m): return [[0]*n for col in range(m)]

#pythran export matrix_multiply(float list list, float list list)
def matrix_multiply(m0, m1):
    new_matrix = zero(len(m0),len(m1[0]))
    for i in range(len(m0)):
        for j in range(len(m1[0])):
            for k in range(len(m1)):
                new_matrix[i][j] += m0[i][k]*m1[k][j]
    return new_matrix

$ pythran mm.py # Generate mm.so
```

TECHNOLOGY TAXONOMY

- Little or no code change from Python code change
- Some code change
- *A different language: C++, Rust etc.*

A DIFFERENT LANGUAGE - 100x

- C/C++ based
  - CPython C Extension
  - ctypes
  - C++
  - CodePy/Boost
  - CFFI
  - SWIG
  - pycxx
  - PyBind11
- Rust, Fortran, Go, Swift

A DIFFERENT LANGUAGE - 100x

- C/C++ based
  - CPython C Extension
  - CFFI
  - PyBind11

A DIFFERENT LANGUAGE - 100x

- C/C++ based
  - *CPython C Extension*
  - CFFI
  - PyBind11

C EXTENSIONS - THE JOY

- It is in C
- Can mix with C++
- You have precise control
- A lot of libraries have efficient C interfaces (looking at you numpy)
- If you write for the standard library you need to be here

C EXTENSIONS - THE AGONY

```
class Noddy:
    def __init__(self, first, last):
        self.first = first
        self.last = last

    def name(self):
        return self.first + " " + self.last
```

C EXTENSIONS - THE AGONY

```c
#include <Python.h>
#include "structmember.h"

typedef struct {
    PyObject_HEAD
    PyObject *first; /* first name */
    PyObject *last;  /* last name */
    int number;
} Noddy;

static void
Noddy_dealloc(Noddy* self)
{
    Py_XDECREF(self->first);
    Py_XDECREF(self->last);
    Py_TYPE(self)->tp_free((PyObject*)self);
}

static PyObject *
Noddy_new(PyTypeObject *type, PyObject *args, PyObject
*kwds)
{
    Noddy *self;

    self = (Noddy *)type->tp_alloc(type, 0);
    if (self != NULL) {
        self->first = PyUnicode_FromString("");
        if (self->first == NULL)
          {
            Py_DECREF(self);
            return NULL;
          }

        self->last = PyUnicode_FromString("");
        if (self->last == NULL)
          {
            Py_DECREF(self);
            return NULL;
          }

        self->number = 0;
    }

    return (PyObject *)self;
}
```

```c
static int
Noddy_init(Noddy *self, PyObject *args, PyObject *kwds)
{
    PyObject *first=NULL, *last=NULL, *tmp;

    static char *kwlist[] = {"first", "last", "number", NULL};

    if (! PyArg_ParseTupleAndKeywords(args, kwds, "|OOi", kwlist,
                                      &first, &last,
                                      &self->number))
        return -1;

    if (first) {
        tmp = self->first;
        Py_INCREF(first);
        self->first = first;
        Py_XDECREF(tmp);
    }

    if (last) {
        tmp = self->last;
        Py_INCREF(last);
        self->last = last;
        Py_XDECREF(tmp);
    }

    return 0;
}


static PyMemberDef Noddy_members[] = {
    {"first", T_OBJECT_EX, offsetof(Noddy, first), 0,
     "first name"},
    {"last", T_OBJECT_EX, offsetof(Noddy, last), 0,
     "last name"},
    {"number", T_INT, offsetof(Noddy, number), 0,
     "noddy number"},
    {NULL}  /* Sentinel */
};

static PyObject *
Noddy_name(Noddy* self)
{
    static PyObject *format = NULL;
    PyObject *args, *result;

    if (format == NULL) {
        format = PyUnicode_FromString("%s %s");
        if (format == NULL)
            return NULL;
    }

    if (self->first == NULL) {
        PyErr_SetString(PyExc_AttributeError, "first");
        return NULL;
    }

    if (self->last == NULL) {
        PyErr_SetString(PyExc_AttributeError, "last");
        return NULL;
    }

    args = Py_BuildValue("OO", self->first, self->last);
    if (args == NULL)
        return NULL;

    result = PyUnicode_Format(format, args);
    Py_DECREF(args);

    return result;
}
```

```c
static PyMethodDef Noddy_methods[] = {
    {"name", (PyCFunction)Noddy_name, METH_NOARGS,
     "Return the name, combining the first and last name"
    },
    {NULL}  /* Sentinel */
};

static PyTypeObject NoddyType = {
    PyVarObject_HEAD_INIT(NULL, 0)
    "noddy.Noddy",             /* tp_name */
    sizeof(Noddy),             /* tp_basicsize */
    0,                         /* tp_itemsize */
    (destructor)Noddy_dealloc, /* tp_dealloc */
    0,                         /* tp_print */
    0,                         /* tp_getattr */
    0,                         /* tp_setattr */
    0,                         /* tp_reserved */
    0,                         /* tp_repr */
    0,                         /* tp_as_number */
    0,                         /* tp_as_sequence */
    0,                         /* tp_as_mapping */
    0,                         /* tp_hash  */
    0,                         /* tp_call */
    0,                         /* tp_str */
    0,                         /* tp_getattro */
    0,                         /* tp_setattro */
    0,                         /* tp_as_buffer */
    Py_TPFLAGS_DEFAULT |
        Py_TPFLAGS_BASETYPE,   /* tp_flags */
    "Noddy objects",           /* tp_doc */
    0,                          /* tp_traverse */
    0,                          /* tp_clear */
    0,                          /* tp_richcompare */
    0,                          /* tp_weaklistoffset */
    0,                          /* tp_iter */
    0,                          /* tp_iternext */
    Noddy_methods,             /* tp_methods */
    Noddy_members,             /* tp_members */
    0,                         /* tp_getset */
    0,                         /* tp_base */
    0,                         /* tp_dict */
    0,                         /* tp_descr_get */
    0,                         /* tp_descr_set */
    0,                         /* tp_dictoffset */
    (initproc)Noddy_init,      /* tp_init */
    0,                         /* tp_alloc */
    Noddy_new,                 /* tp_new */
};
```

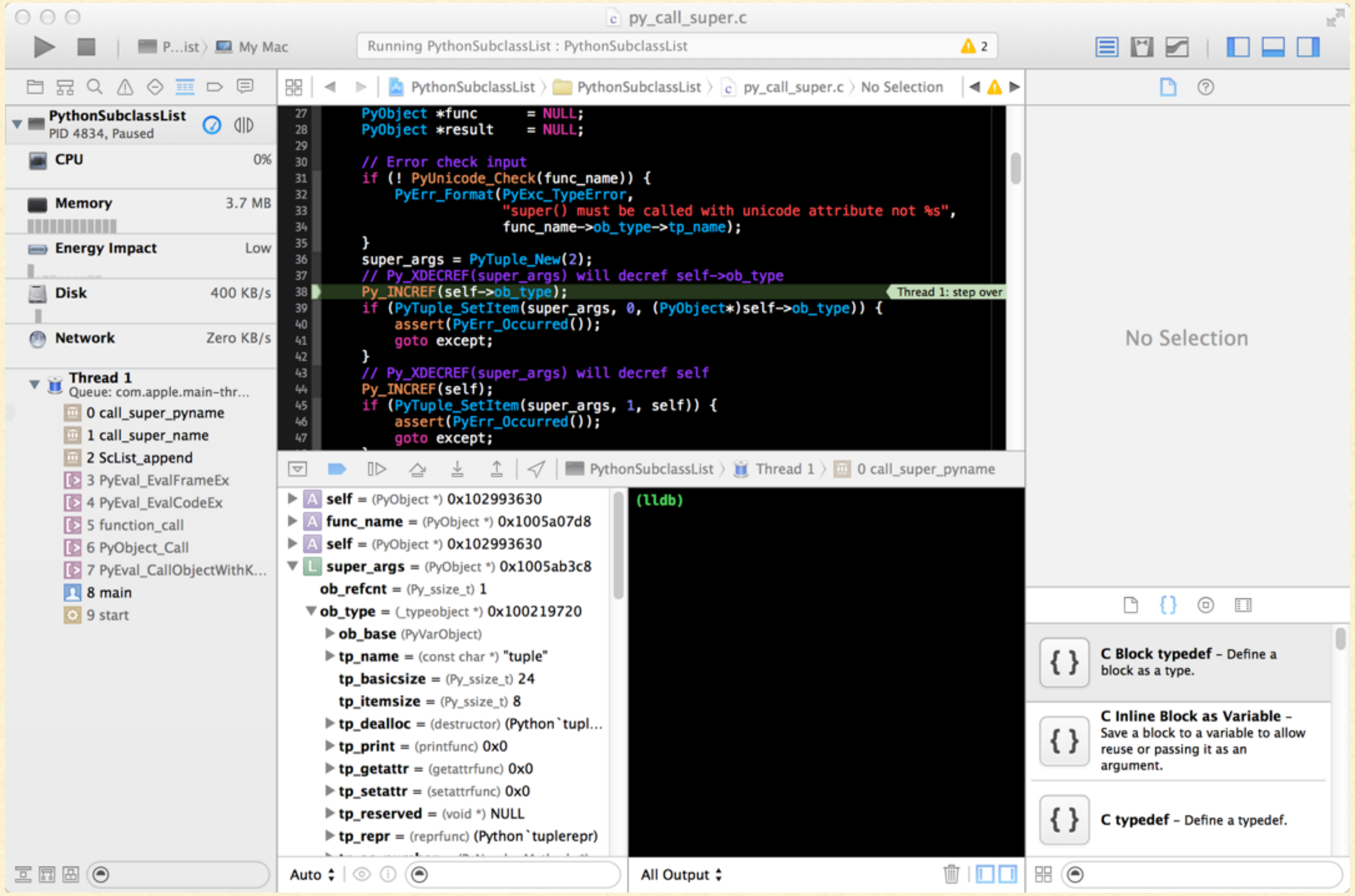
```c
static PyModuleDef noddy2module = {
    PyModuleDef_HEAD_INIT,
    "noddy2",
    "Example module that creates an extension type.",
    -1,
    NULL, NULL, NULL, NULL, NULL
};

PyMODINIT_FUNC
PyInit_noddy2(void)
{
    PyObject* m;

    if (PyType_Ready(&NoddyType) < 0)
        return NULL;

    m = PyModule_Create(&noddy2module);
    if (m == NULL)
        return NULL;

    Py_INCREF(&NoddyType);
    PyModule_AddObject(m, "Noddy", (PyObject *)&NoddyType);
    return m;
}
```

C EXTENSIONS - THE AGONY

- It is in C
- Reference counts and memory allocation
- It is specialised and expensive to write
- Testing is problematic
- Debugging: GDB is fine, IDEs are a little tricky to set up

DEBUGGING C EXTENSIONS IN XCODE

http://pythonextensionpatterns.readthedocs.io/en/latest/debugging/debug_in_ide.html


py_call_super.c
P...ist › My Mac
Running PythonSubclassList : PythonSubclassList
2
PythonSubclassList › PythonSubclassList › py_call_super.c › No Selection
PythonSubclassList
PID 4834, Paused
CPU 0%
Memory 3.7 MB
Energy Impact Low
Disk 400 KB/s
Network Zero KB/s
Thread 1
Queue: com.apple.main-thr...
0 call_super_pyname
1 call_super_name
2 ScList_append
3 PyEval_EvalFrameEx
4 PyEval_EvalCodeEx
5 function_call
6 PyObject_Call
7 PyEval_CallObjectWithK...
8 main
9 start
Thread 1: step over
PythonSubclassList › Thread 1 › 0 call_super_pyname
Auto
All Output
No Selection
C Block typedef - Define a block as a type.
C Inline Block as Variable - Save a block to a variable to allow reuse or passing it as an argument.
C typedef - Define a typedef.

```c
    PyObject *func       = NULL;
    PyObject *result     = NULL;

    // Error check input
    if (! PyUnicode_Check(func_name)) {
        PyErr_Format(PyExc_TypeError,
                     "super() must be called with unicode attribute not %s",
                     func_name->ob_type->tp_name);
    }
    super_args = PyTuple_New(2);
    // Py_XDECREF(super_args) will decref self->ob_type
    Py_INCREF(self->ob_type);
    if (PyTuple_SetItem(super_args, 0, (PyObject*)self->ob_type)) {
        assert(PyErr_Occurred());
        goto except;
    }
    // Py_XDECREF(super_args) will decref self
    Py_INCREF(self);
    if (PyTuple_SetItem(super_args, 1, self)) {
        assert(PyErr_Occurred());
        goto except;
```

```
self = (PyObject *) 0x102993630
func_name = (PyObject *) 0x1005a07d8
self = (PyObject *) 0x102993630
super_args = (PyObject *) 0x1005ab3c8
  ob_refcnt = (Py_ssize_t) 1
  ob_type = (_typeobject *) 0x100219720
    ob_base (PyVarObject)
    tp_name = (const char *) "tuple"
    tp_basicsize = (Py_ssize_t) 24
    tp_itemsize = (Py_ssize_t) 8
    tp_dealloc = (destructor) (Python`tupl...
    tp_print = (printfunc) 0x0
    tp_getattr = (getattrfunc) 0x0
    tp_setattr = (setattrfunc) 0x0
    tp_reserved = (void *) NULL
    tp_repr = (reprfunc) (Python`tuplerepr)
```

```
(lldb)
```

A DIFFERENT LANGUAGE - 100x

- C/C++ based
  - CPython C Extension
  - *CFFI*
  - PyBind11

CFFI

https://bitbucket.org/cffi/cffi/src

https://cffi.readthedocs.io/en/latest/

- Allows you to directly call C code from within Python
- Can also be hooked up to C++ code
- Abstracts away much of the build and interface code

C EXTENSIONS - THE AGONY

```python
class Noddy:
    def __init__(self, first, last):
        self.first = first
        self.last = last

    def name(self):
        return self.first + " " + self.last
```

C EXTENSIONS - THE AGONY

```c
#include <Python.h>
#include "structmember.h"

typedef struct {
    PyObject_HEAD
    PyObject *first; /* first name */
    PyObject *last;  /* last name */
    int number;
} Noddy;

static void
Noddy_dealloc(Noddy* self)
{
    Py_XDECREF(self->first);
    Py_XDECREF(self->last);
    Py_TYPE(self)->tp_free((PyObject*)self);
}

static PyObject *
Noddy_new(PyTypeObject *type, PyObject *args, PyObject
*kwds)
{
    Noddy *self;

    self = (Noddy *)type->tp_alloc(type, 0);
    if (self != NULL) {
        self->first = PyUnicode_FromString("");
        if (self->first == NULL)
          {
            Py_DECREF(self);
            return NULL;
          }

        self->last = PyUnicode_FromString("");
        if (self->last == NULL)
          {
            Py_DECREF(self);
            return NULL;
          }

        self->number = 0;
    }

    return (PyObject *)self;
}
```

```c
static int
Noddy_init(Noddy *self, PyObject *args, PyObject *kwds)
{
    PyObject *first=NULL, *last=NULL, *tmp;

    static char *kwlist[] = {"first", "last", "number", NULL};

    if (! PyArg_ParseTupleAndKeywords(args, kwds, "|OOi", kwlist,
                                      &first, &last,
                                      &self->number))
        return -1;

    if (first) {
        tmp = self->first;
        Py_INCREF(first);
        self->first = first;
        Py_XDECREF(tmp);
    }

    if (last) {
        tmp = self->last;
        Py_INCREF(last);
        self->last = last;
        Py_XDECREF(tmp);
    }

    return 0;
}


static PyMemberDef Noddy_members[] = {
    {"first", T_OBJECT_EX, offsetof(Noddy, first), 0,
     "first name"},
    {"last", T_OBJECT_EX, offsetof(Noddy, last), 0,
     "last name"},
    {"number", T_INT, offsetof(Noddy, number), 0,
     "noddy number"},
    {NULL}  /* Sentinel */
};

static PyObject *
Noddy_name(Noddy* self)
{
    static PyObject *format = NULL;
    PyObject *args, *result;

    if (format == NULL) {
        format = PyUnicode_FromString("%s %s");
        if (format == NULL)
            return NULL;
    }

    if (self->first == NULL) {
        PyErr_SetString(PyExc_AttributeError, "first");
        return NULL;
    }

    if (self->last == NULL) {
        PyErr_SetString(PyExc_AttributeError, "last");
        return NULL;
    }

    args = Py_BuildValue("OO", self->first, self->last);
    if (args == NULL)
        return NULL;

    result = PyUnicode_Format(format, args);
    Py_DECREF(args);

    return result;
}
```

```c
static PyMethodDef Noddy_methods[] = {
    {"name", (PyCFunction)Noddy_name, METH_NOARGS,
     "Return the name, combining the first and last name"
    },
    {NULL}  /* Sentinel */
};

static PyTypeObject NoddyType = {
    PyVarObject_HEAD_INIT(NULL, 0)
    "noddy.Noddy",             /* tp_name */
    sizeof(Noddy),             /* tp_basicsize */
    0,                         /* tp_itemsize */
    (destructor)Noddy_dealloc, /* tp_dealloc */
    0,                         /* tp_print */
    0,                         /* tp_getattr */
    0,                         /* tp_setattr */
    0,                         /* tp_reserved */
    0,                         /* tp_repr */
    0,                         /* tp_as_number */
    0,                         /* tp_as_sequence */
    0,                         /* tp_as_mapping */
    0,                         /* tp_hash  */
    0,                         /* tp_call */
    0,                         /* tp_str */
    0,                         /* tp_getattro */
    0,                         /* tp_setattro */
    0,                         /* tp_as_buffer */
    Py_TPFLAGS_DEFAULT |
        Py_TPFLAGS_BASETYPE,   /* tp_flags */
    "Noddy objects",           /* tp_doc */
    0,                          /* tp_traverse */
    0,                          /* tp_clear */
    0,                          /* tp_richcompare */
    0,                          /* tp_weaklistoffset */
    0,                          /* tp_iter */
    0,                          /* tp_iternext */
    Noddy_methods,             /* tp_methods */
    Noddy_members,             /* tp_members */
    0,                         /* tp_getset */
    0,                         /* tp_base */
    0,                         /* tp_dict */
    0,                         /* tp_descr_get */
    0,                         /* tp_descr_set */
    0,                         /* tp_dictoffset */
    (initproc)Noddy_init,      /* tp_init */
    0,                         /* tp_alloc */
    Noddy_new,                 /* tp_new */
};
```

```c
static PyModuleDef noddy2module = {
    PyModuleDef_HEAD_INIT,
    "noddy2",
    "Example module that creates an extension type.",
    -1,
    NULL, NULL, NULL, NULL, NULL
};

PyMODINIT_FUNC
PyInit_noddy2(void)
{
    PyObject* m;

    if (PyType_Ready(&NoddyType) < 0)
        return NULL;

    m = PyModule_Create(&noddy2module);
    if (m == NULL)
        return NULL;

    Py_INCREF(&NoddyType);
    PyModule_AddObject(m, "Noddy", (PyObject *)&NoddyType);
    return m;
}
```

CFFI

https://bitbucket.org/cffi/cffi/src

https://cffi.readthedocs.io/en/latest/

```
from cffi import FFI

ffi = FFI()
ffi.cdef("""
    typedef struct {
        char first[128];
        char last[128];
    } Noddy;
""")

noddy = ffi.new("Noddy*")
noddy.first = b"Paul"
noddy.last = b"Ross"
ffi.string(noddy.first) + b' ' + ffi.string(noddy.last)
# b'Paul Ross'
```

A DIFFERENT LANGUAGE - 100x

- C/C++ based
  - CPython C Extension
  - CFFI
  - *PyBind11*

PYBIND11

https://github.com/pybind/pybind11

- Header only C++ library
- Makes it easy to write C extensions
- Similar in concept to Boost.Python
- C++11

C EXTENSIONS - THE AGONY

```python
class Noddy:
    def __init__(self, first, last):
        self.first = first
        self.last = last

    def name(self):
        return self.first + " " + self.last
```

C EXTENSIONS - THE AGONY

```c
#include <Python.h>
#include "structmember.h"

typedef struct {
    PyObject_HEAD
    PyObject *first; /* first name */
    PyObject *last;  /* last name */
    int number;
} Noddy;

static void
Noddy_dealloc(Noddy* self)
{
    Py_XDECREF(self->first);
    Py_XDECREF(self->last);
    Py_TYPE(self)->tp_free((PyObject*)self);
}

static PyObject *
Noddy_new(PyTypeObject *type, PyObject *args, PyObject
*kwds)
{
    Noddy *self;

    self = (Noddy *)type->tp_alloc(type, 0);
    if (self != NULL) {
        self->first = PyUnicode_FromString("");
        if (self->first == NULL)
          {
            Py_DECREF(self);
            return NULL;
          }

        self->last = PyUnicode_FromString("");
        if (self->last == NULL)
          {
            Py_DECREF(self);
            return NULL;
          }

        self->number = 0;
    }

    return (PyObject *)self;
}
```

```c
static int
Noddy_init(Noddy *self, PyObject *args, PyObject *kwds)
{
    PyObject *first=NULL, *last=NULL, *tmp;

    static char *kwlist[] = {"first", "last", "number", NULL};

    if (! PyArg_ParseTupleAndKeywords(args, kwds, "|OOi", kwlist,
                                      &first, &last,
                                      &self->number))
        return -1;

    if (first) {
        tmp = self->first;
        Py_INCREF(first);
        self->first = first;
        Py_XDECREF(tmp);
    }

    if (last) {
        tmp = self->last;
        Py_INCREF(last);
        self->last = last;
        Py_XDECREF(tmp);
    }

    return 0;
}


static PyMemberDef Noddy_members[] = {
    {"first", T_OBJECT_EX, offsetof(Noddy, first), 0,
     "first name"},
    {"last", T_OBJECT_EX, offsetof(Noddy, last), 0,
     "last name"},
    {"number", T_INT, offsetof(Noddy, number), 0,
     "noddy number"},
    {NULL}  /* Sentinel */
};

static PyObject *
Noddy_name(Noddy* self)
{
    static PyObject *format = NULL;
    PyObject *args, *result;

    if (format == NULL) {
        format = PyUnicode_FromString("%s %s");
        if (format == NULL)
            return NULL;
    }

    if (self->first == NULL) {
        PyErr_SetString(PyExc_AttributeError, "first");
        return NULL;
    }

    if (self->last == NULL) {
        PyErr_SetString(PyExc_AttributeError, "last");
        return NULL;
    }

    args = Py_BuildValue("OO", self->first, self->last);
    if (args == NULL)
        return NULL;

    result = PyUnicode_Format(format, args);
    Py_DECREF(args);

    return result;
}
```

```c
static PyMethodDef Noddy_methods[] = {
    {"name", (PyCFunction)Noddy_name, METH_NOARGS,
     "Return the name, combining the first and last name"
    },
    {NULL}  /* Sentinel */
};

static PyTypeObject NoddyType = {
    PyVarObject_HEAD_INIT(NULL, 0)
    "noddy.Noddy",             /* tp_name */
    sizeof(Noddy),             /* tp_basicsize */
    0,                         /* tp_itemsize */
    (destructor)Noddy_dealloc, /* tp_dealloc */
    0,                         /* tp_print */
    0,                         /* tp_getattr */
    0,                         /* tp_setattr */
    0,                         /* tp_reserved */
    0,                         /* tp_repr */
    0,                         /* tp_as_number */
    0,                         /* tp_as_sequence */
    0,                         /* tp_as_mapping */
    0,                         /* tp_hash  */
    0,                         /* tp_call */
    0,                         /* tp_str */
    0,                         /* tp_getattro */
    0,                         /* tp_setattro */
    0,                         /* tp_as_buffer */
    Py_TPFLAGS_DEFAULT |
        Py_TPFLAGS_BASETYPE,   /* tp_flags */
    "Noddy objects",           /* tp_doc */
    0,                          /* tp_traverse */
    0,                          /* tp_clear */
    0,                          /* tp_richcompare */
    0,                          /* tp_weaklistoffset */
    0,                          /* tp_iter */
    0,                          /* tp_iternext */
    Noddy_methods,             /* tp_methods */
    Noddy_members,             /* tp_members */
    0,                         /* tp_getset */
    0,                         /* tp_base */
    0,                         /* tp_dict */
    0,                         /* tp_descr_get */
    0,                         /* tp_descr_set */
    0,                         /* tp_dictoffset */
    (initproc)Noddy_init,      /* tp_init */
    0,                         /* tp_alloc */
    Noddy_new,                 /* tp_new */
};
```

```c
static PyModuleDef noddy2module = {
    PyModuleDef_HEAD_INIT,
    "noddy2",
    "Example module that creates an extension type.",
    -1,
    NULL, NULL, NULL, NULL, NULL
};

PyMODINIT_FUNC
PyInit_noddy2(void)
{
    PyObject* m;

    if (PyType_Ready(&NoddyType) < 0)
        return NULL;

    m = PyModule_Create(&noddy2module);
    if (m == NULL)
        return NULL;

    Py_INCREF(&NoddyType);
    PyModule_AddObject(m, "Noddy", (PyObject *)&NoddyType);
    return m;
}
```

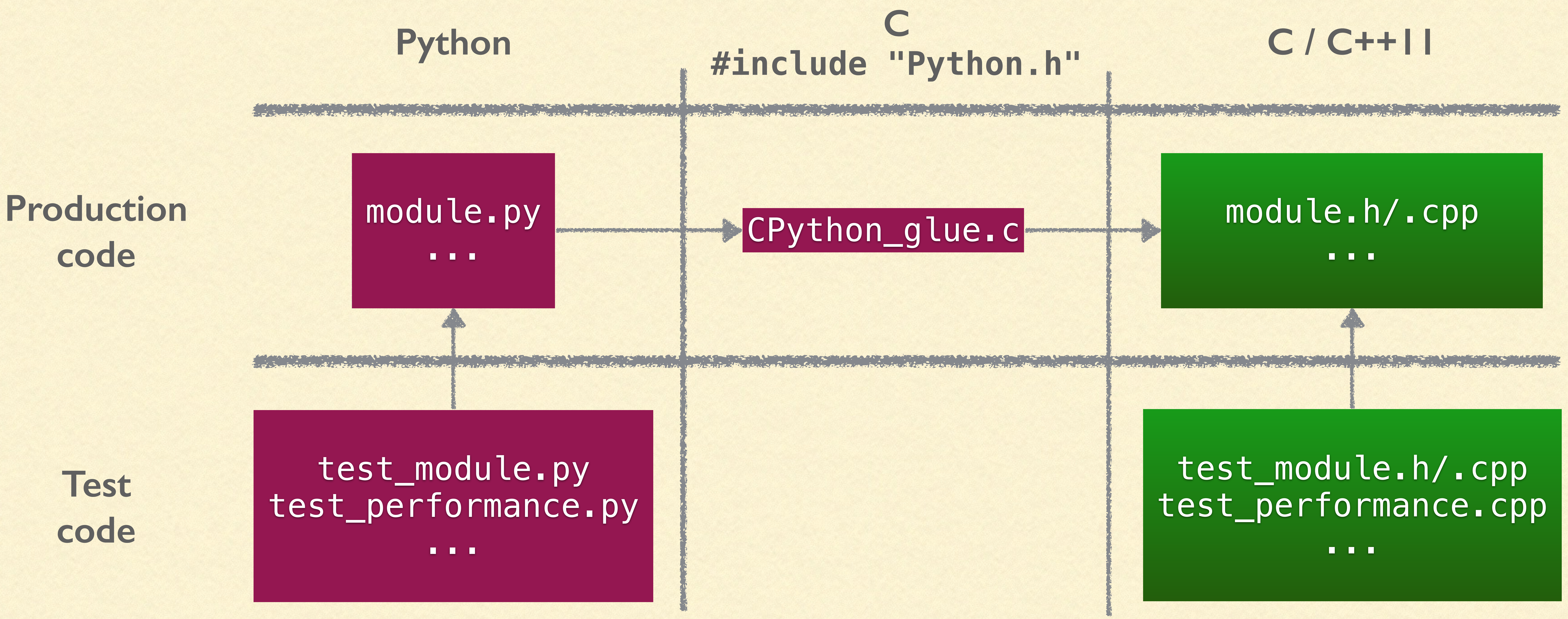
PYBIND11

```
struct Noddy {
    Noddy(const std::string &first, const std::string &last) : first(first), last(last) { }
    std::string name() { return first + " " + last; }

    std::string first;
    std::string last
};

#include <pybind11/pybind11.h>

namespace py = pybind11;

PYBIND11_MODULE(noddy, m) {
    py::class_<Noddy>(m, "Noddy")
        .def(py::init<const std::string &, const std::string &>())
        .def("name", &Noddy::name);
}
```

A DIFFERENT LANGUAGE - 100x

- C/C++ based
  - CPython C Extension
  - CFFI
  - PyBind11

CODE INTERFACES



TECHNOLOGY TAXONOMY

- Little or no code change from Python code change
- Some code change
- A different language: C++, Rust etc.

SECTIONS OF THIS TALK

- Introduction and scope
- A technology taxonomy
- *Evaluation criteria*

A LOT OF CHOICE…

PARAKEET

Python/C API

Microsoft / Pyjion

boost C++ LIBRARIES

THE PYSTON BLOG

16.16. ctypes

CFFI

SWIG

Cython

shedskin / shedskin

Pythran

pypy

Numba

pybind11

INTEL® DISTRIBUTION FOR PYTHON*

EVALUATION CRITERIA

- Who you are
- Technical criteria
- Non-technical criteria

EVALUATION CRITERIA

- *Who you are*
- Technical criteria
- Non-technical criteria

WHO YOU ARE

- You are probably not Google/Facebook/MS etc.
- What constraints your culture imposes on you
- What skills you have, or can acquire (or lose)

EVALUATION CRITERIA

- Who you are
- *Technical criteria*
- Non-technical criteria

TECHNICAL CRITERIA

- Dependencies
- Supported Python versions
- Core, standard library and 3rd party library support
- Benchmarks

OBSTACLES TO BENCHMARKING

- Measurement errors
  - Measuring the wrong thing
- Bad statistics
- Cognitive biases
  - Confirmation bias
  - Fixation error

BENCHMARK PITFALLS

RUN	C	D
1	5	18
2	8	8
3	13	8
4	9	8
5	11	8
6	14	8
7	10	8
8	4	8
Mean	**9.3**	**9.3**
Std.Dev.	**3.5**	**3.5**

COMBINING BENCHMARK RESULTS

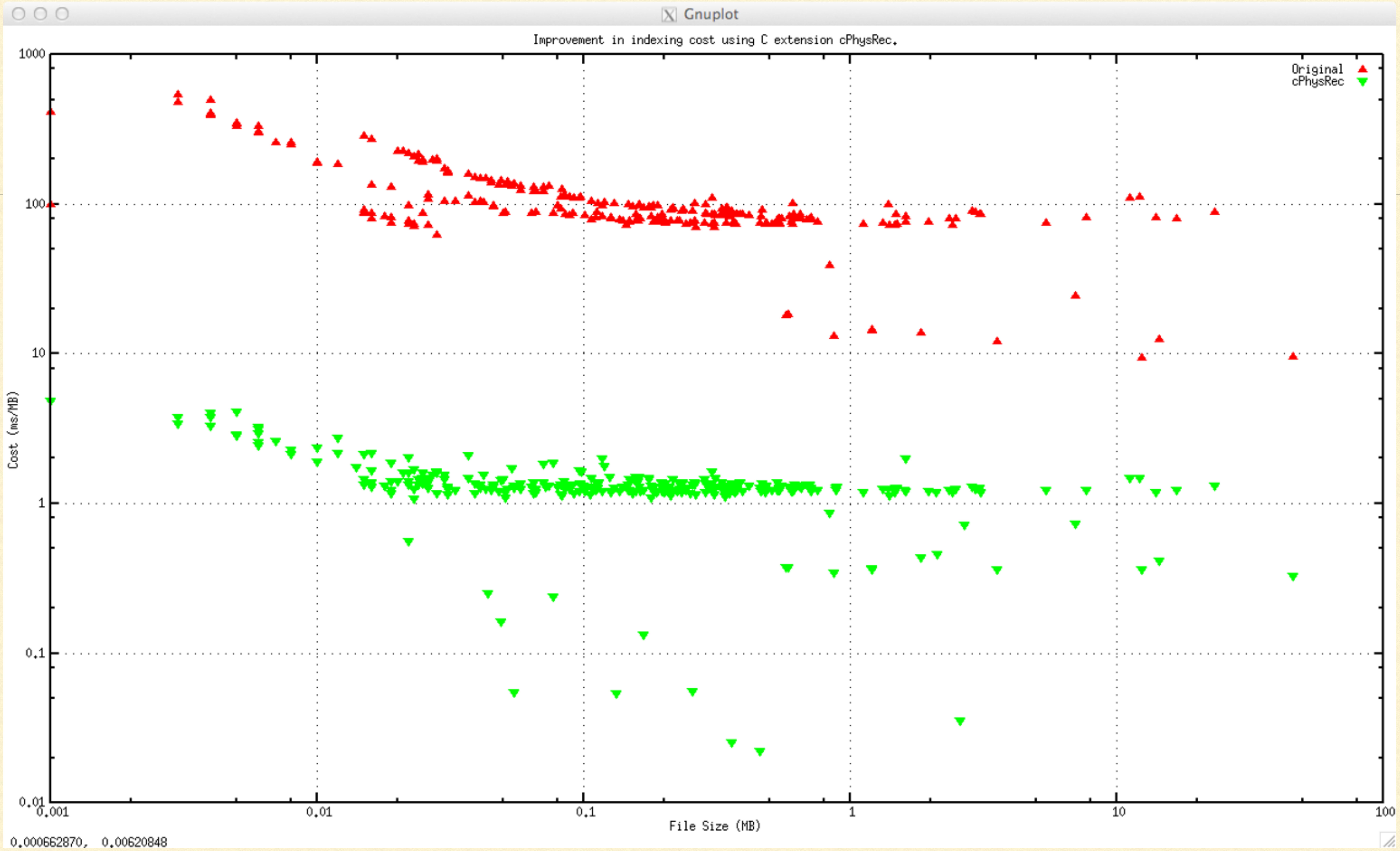
	G	H
Test 1	226	263
Test 2	18	9
Test 3	8	4
Test 4	16	8
Test 5	12	6
Test 6	10	5
Test 7	6	3
Test 8	4	2
Mean	38	38

COMBINING BENCHMARK RESULTS

	G	H	H/G
Test 1	226	263	1.2
Test 2	18	9	0.5
Test 3	8	4	0.5
Test 4	16	8	0.5
Test 5	12	6	0.5
Test 6	10	5	0.5
Test 7	6	3	0.5
Test 8	4	2	0.5
Mean	38	38	

COMBINING BENCHMARK RESULTS

	G	H	H/G
Test 1	226	263	1.2
Test 2	18	9	0.5
Test 3	8	4	0.5
Test 4	16	8	0.5
Test 5	12	6	0.5
Test 6	10	5	0.5
Test 7	6	3	0.5
Test 8	4	2	0.5
Geo	19	11	


Gnuplot
Improvement in indexing cost using C extension cPhysRec.
Original ▲
cPhysRec ▼
Cost (ms/MB)
File Size (MB)
1000
100
10
1
0.1
0.01
0.001
0.01
0.1
1
10
100
0.000662870, 0.00620848

RANGE OF BENCHMARKS

- Speed
- Memory
- I/O
- Load testing
- Trends
- Combinations
- Production monitoring

EVALUATION CRITERIA

- Who you are
- Technical criteria
- *Non-technical criteria*

NON-TECHNICAL CRITERIA

- Ease of installation and deployment
- Dependencies
- Ease of writing
- Ease of maintenance
- Debugging and tools story
- Future proof?

FUTURE PROOF?

The past is no guide to the future (but it is the best we have)

- Python versions
- Development status
  - Age?
  - Maintained?
  - GitHub stars?
  - Has backers?
  - Fixes are quick?
  - Accepts PRs?

FUTURE PROOF?

The past is no guide to the future (but it is the best we have)

- Who is using it?
- Consultancy?

SECTIONS OF THIS TALK

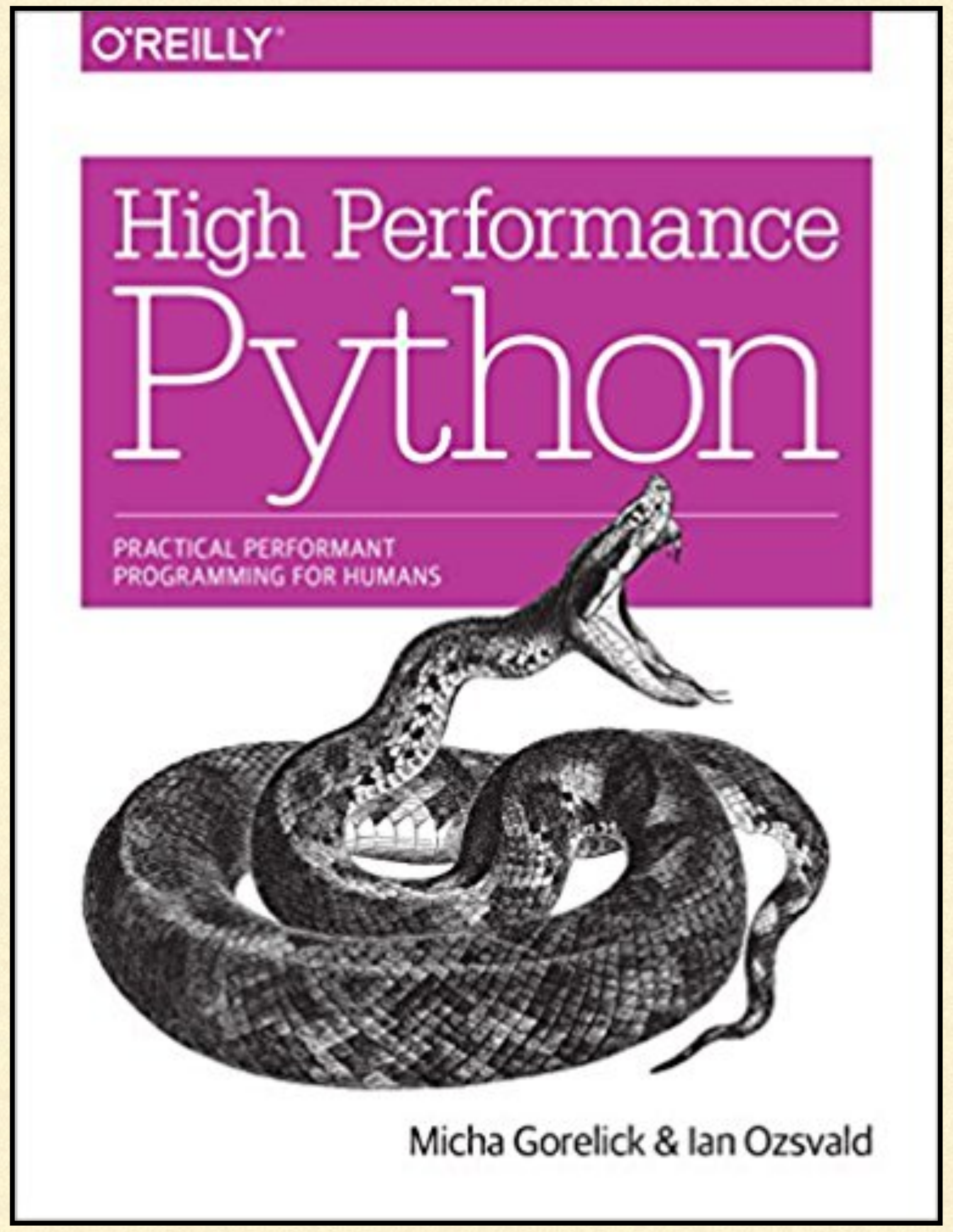
- Introduction and scope
- A technology taxonomy
- Evaluation criteria

SUMMARY

- Choose what is appropriate for *your* organisation and *your* product
- Recognise the trade-offs implicit in that choice
- Benchmark if you must
- Non-technical criteria are as important as technical ones

SUMMARY

- Choose what is appropriate for *your* organisation and *your* product
- Recognise the trade-offs implicit in that choice
- Benchmark if you must
- Non-technical criteria are as important as technical ones



OTHER OPINIONS

- Monday and Wednesday
  - M. MÜLLER *Faster Python Programs - Measure, don't Guess*
  - J. BEVILACQUA *Call a C API from Python becomes more enjoyable with CFFI*
  - A. SVETLOV *Optimizing Python code with Cython*
  - A. CUNI *The joy of PyPy JIT: abstractions for free*
- Friday
  - I. SMIRNOV *pybind11 - seamless operability between C++11 and Python*
  - A. RIGO *PyPy meets Python 3 and Numpy*

QUESTIONS?

https://github.com/paulross

https://github.com/manahl

https://twitter.com/manahltech