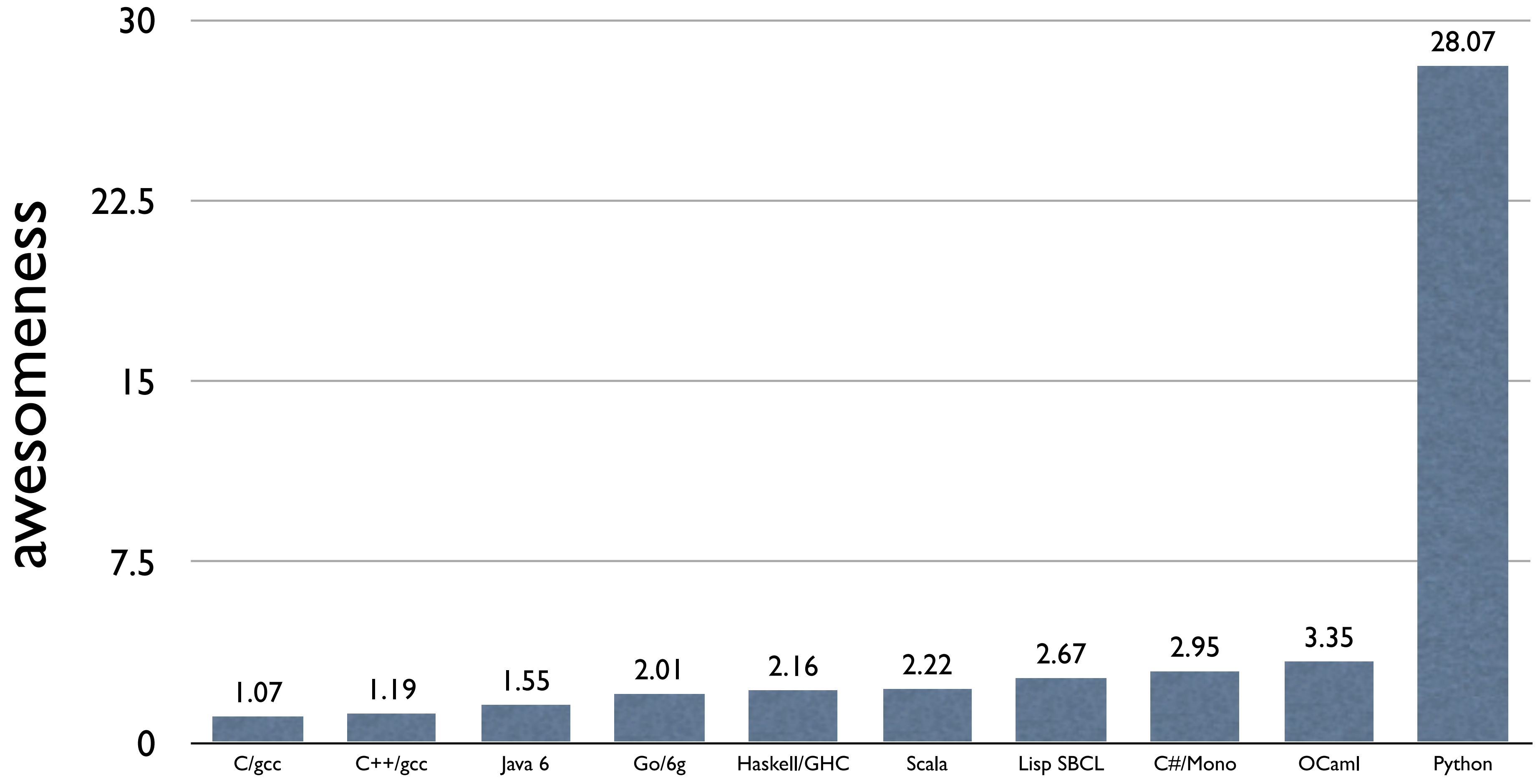
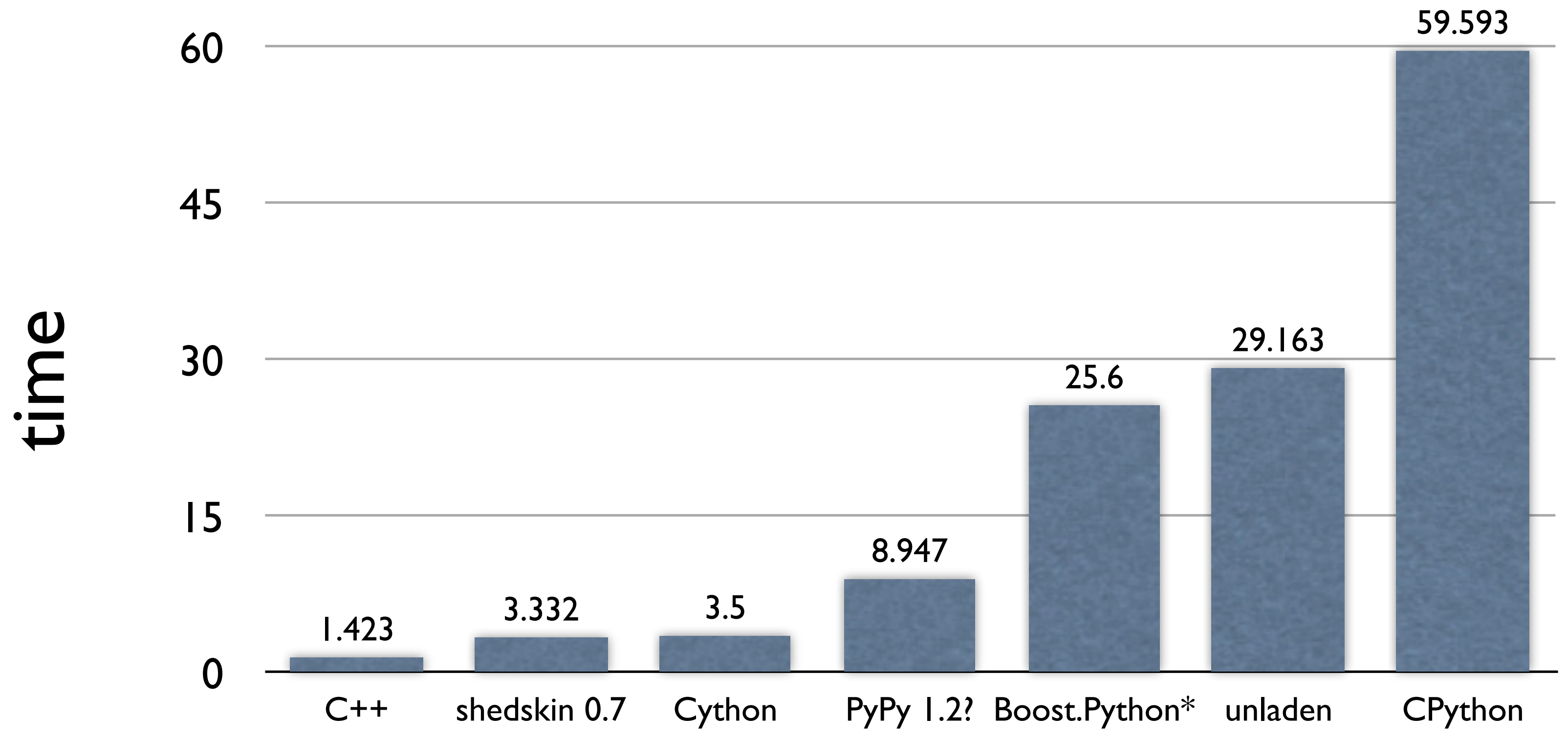
Python is awesome



A benchmark

http://geetduggal.wordpress.com/2010/11/25/speed-up-your-python-unladen-vs-shedskin-vs-pypy-vs-c/



*totally made up (based on http://chrischou.files.wordpress.com/2010/02/cython-vs-boost-python1.png)

Cython 101

- Cython translates "Python" into C++ calling the C Python API

fib2.pyx

```
def fib(int n):
    l = []
    cdef int a=0, b=1
    for i in range(n):
        l.append(b)
        a, b = b, a+b
    return l
```

fib.py

```
def fib(n):
    l = []
    a, b = 0, 1
    for i in range(n):
        l.append(b)
        a, b = b, a+b
    return l
```

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

setup(cmdclass = {'build_ext': build_ext},
      ext_modules = [Extension("fib2", ["fib2.pyx"], language="c")]
      )
```

Cython Compiling

- fib2.pyx ==cython==> fib2.c ==gcc==> fib2.o ==gcc==> fib2.so
- name it fib2.so instead of fib.so to avoid conflict with fib.py
- --install-lib . to copy .so files to current dir

```
[<lhuang@Mac OS X:~/install/svector/tutorial>] python setup.py install --install-lib .

running install
running build
running build_ext
running install_lib
running install
running build
running build_ext
cythoning fib_cy.pyx to fib_cy.cpp
building 'fib_cy' extension
gcc-4.2 -fno-strict-aliasing -fno-common -dynamic -arch i386 -arch x86_64 -g -O2 -
DNDEBUG -g -O3 -I/Library/Frameworks/Python.framework/Versions/2.7/include/python2.7 -c
fib_cy.cpp -o build/temp.macosx-10.6-intel-2.7/fib_cy.o
c++ -arch i386 -arch x86_64 -isysroot / -g -bundle -undefined dynamic_lookup -arch i386
-arch x86_64 -isysroot / -g build/temp.macosx-10.6-intel-2.7/fib_cy.o -o build/
lib.macosx-10.6-intel-2.7/fib_cy.so
running install_lib
copying build/lib.macosx-10.6-intel-2.7/fib_cy.so -> .
```

Compiled Cython Program (.c)

```
/* Generated by Cython 0.14 on Wed Jun  1 15:38:37 2011 */

#include "Python.h"

...

  /* "/Users/lhuang/install/svector/tutorial/fib_cy.pyx":2
 * def fib(int n):
 *     l = []              # <<<<<<<<<<<<<<
 *     cdef int a=0, b=1
 *     for i in range(n):
 */
  __pyx_t_1 = PyList_New(0); if (unlikely(!__pyx_t_1)) {__pyx_filename = __pyx_f[0];
__pyx_lineno = 2; __pyx_clineno = __LINE__; goto __pyx_L1_error;}
  __Pyx_GOTREF(((PyObject *)__pyx_t_1));
  __Pyx_DECREF(((PyObject *)__pyx_v_l));
  __pyx_v_l = __pyx_t_1;
  __pyx_t_1 = 0;

  /* "/Users/lhuang/install/svector/tutorial/fib_cy.pyx":3
 * def fib(int n):
 *     l = []
 *     cdef int a=0, b=1             # <<<<<<<<<<<<<<
 *     for i in xrange(n):
 *         l.append(b)
 */
  __pyx_v_a = 0;
  __pyx_v_b = 1;
```

Compiled Cython Program (.c)

```
/* "/Users/lhuang/install/svector/tutorial/fib_cy.pyx":4
 *     l = []
 *     cdef int a=0, b=1
 *     for i in range(n):             # <<<<<<<<<<<<<<
 *         l.append(b)
 *         a, b = b, a+b
 */
  __pyx_t_2 = __pyx_v_n;
  for (__pyx_t_3 = 0; __pyx_t_3 < __pyx_t_2; __pyx_t_3+=1) {
    __pyx_v_i = __pyx_t_3;

/* "/Users/lhuang/install/svector/tutorial/fib_cy.pyx":5
 *     cdef int a=0, b=1
 *     for i in xrange(n):
 *         l.append(b)             # <<<<<<<<<<<<<<
 *         a, b = b, a+b
 *     return l
 */
    if (unlikely(__pyx_v_l == Py_None)) {
      PyErr_SetString(PyExc_AttributeError, "'NoneType' object has no attribute
'append'"); {__pyx_filename = __pyx_f[0]; __pyx_lineno = 5; __pyx_clineno = __LINE__;
goto __pyx_L1_error;}     }
    __pyx_t_1 = PyInt_FromLong(__pyx_v_b); if (unlikely(!__pyx_t_1)) {__pyx_filename =
__pyx_f[0]; __pyx_lineno = 5; __pyx_clineno = __LINE__; goto __pyx_L1_error;}
    __Pyx_GOTREF(__pyx_t_1);
    __pyx_t_4 = PyList_Append(__pyx_v_l, __pyx_t_1); if (unlikely(__pyx_t_4 == -1))
{__pyx_filename = __pyx_f[0]; __pyx_lineno = 5; __pyx_clineno = __LINE__; goto
__pyx_L1_error;}
    __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
```

*very clever: for loop detected! but should always use **xrange** in your .pyx or .py!*

Compiled Cython Program (.c)

```
/* "/Users/lhuang/install/svector/tutorial/fib_cy.pyx":6
 *     for i in xrange(n):
 *         l.append(b)
 *         a, b = b, a+b             # <<<<<<<<<<<<<<
 *     return l
 */
    __pyx_t_4 = __pyx_v_b;
    __pyx_t_5 = (__pyx_v_a + __pyx_v_b);
    __pyx_v_a = __pyx_t_4;
    __pyx_v_b = __pyx_t_5;
  }

...
```

correctly handles simultaneous assignment

If I wrote it myself in CPython API

```
static PyObject *__pyx_pf_6fib_cy_0fib(PyObject *__pyx_self, PyObject *__pyx_arg_n) {

  __pyx_v_n = __Pyx_PyInt_AsInt(__pyx_arg_n);
  __pyx_t_1 = PyList_New(0);
  __pyx_v_a = 0;
  __pyx_v_b = 1;

  for (__pyx_t_3 = 0; __pyx_t_3 < __pyx_t_n; __pyx_t_3+=1) {

    __pyx_t_1 = PyInt_FromLong(__pyx_v_b);
    __pyx_t_4 = PyList_Append(__pyx_v_l, __pyx_t_1);
    __Pyx_DECREF(__pyx_t_1);

    __pyx_t_4 = __pyx_v_b;
    __pyx_t_5 = (__pyx_v_a + __pyx_v_b);
    __pyx_v_a = __pyx_t_4;
    __pyx_v_b = __pyx_t_5;
  }

  ...
}
```

(almost) identical

```
static PyObject *
fib(PyObject *self, PyIntObject *arg)
{
  int a = 0, b = 1, c, n, i;

  n = PyInt_AsLong(arg); // checks error

  PyObject *p;
  PyObject *list = PyList_New(0);
  for (i=0; i < n; i++) {
    p = PyInt_FromLong(b); // returns ne
    PyList_Append(list, p);
    Py_DECREF(p);  // important; otherwi
    c = a+b;
    a = b;
    b = c;
  }
  return list;
}
```

Variant: Cython with Python Types

- cdef to define C (instead of Python) types

fib2.pyx

```
def fib(int n):
    l = []
    cdef int a=0, b=1
    for i in range(n):
        l.append(b)
        a, b = b, a+b
    return l
```

fib3.pyx

```
def fib(n):
    l = []
    a, b = 0, 1
    for i in range(n):
        l.append(b)
        a, b = b, a+b
    return l
```

identical to fib.py!

Benchmark

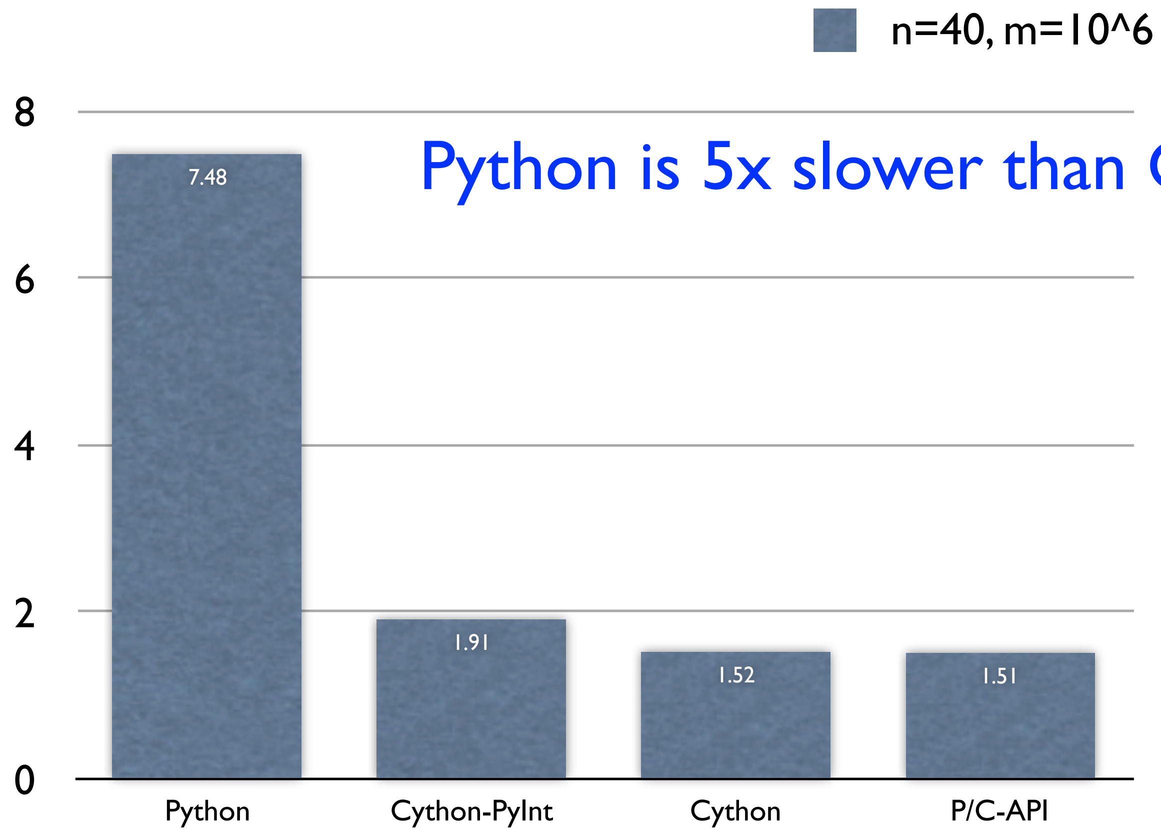
- python vs cython-cint vs cython-pyint vs cpython API

```
#!/usr/bin/env python

'''compares python, cython, and cpython api '''

from time import time
import sys

def timeit(func):
    t = time()
    for _ in range(m):
        x = func(n)[-1]
    print x
    return time() - t

n, m = 40, 1000000
print "n=%d m=%d" % (n, m)

module = __import__(sys.argv[1])
print "%10f\t%s" % (timeit(module.fib), module)
```

```
$ ./test_fibs.py fib
n=40 m=1000000
102334155
  7.595778      <fib.pyc>

$ ./test_fibs.py fib2  # cython-cint
n=40 m=1000000
102334155
  1.537607      <fib2.so>

$ ./test_fibs.py fib3  # cython-pyint
n=40 m=1000000
102334155
  1.994384      <fib3.so>

$ ./test_fibs.py fibc  # API
n=40 m=1000000
102334155
  1.516284      <fibc.so>
```

Benchmark

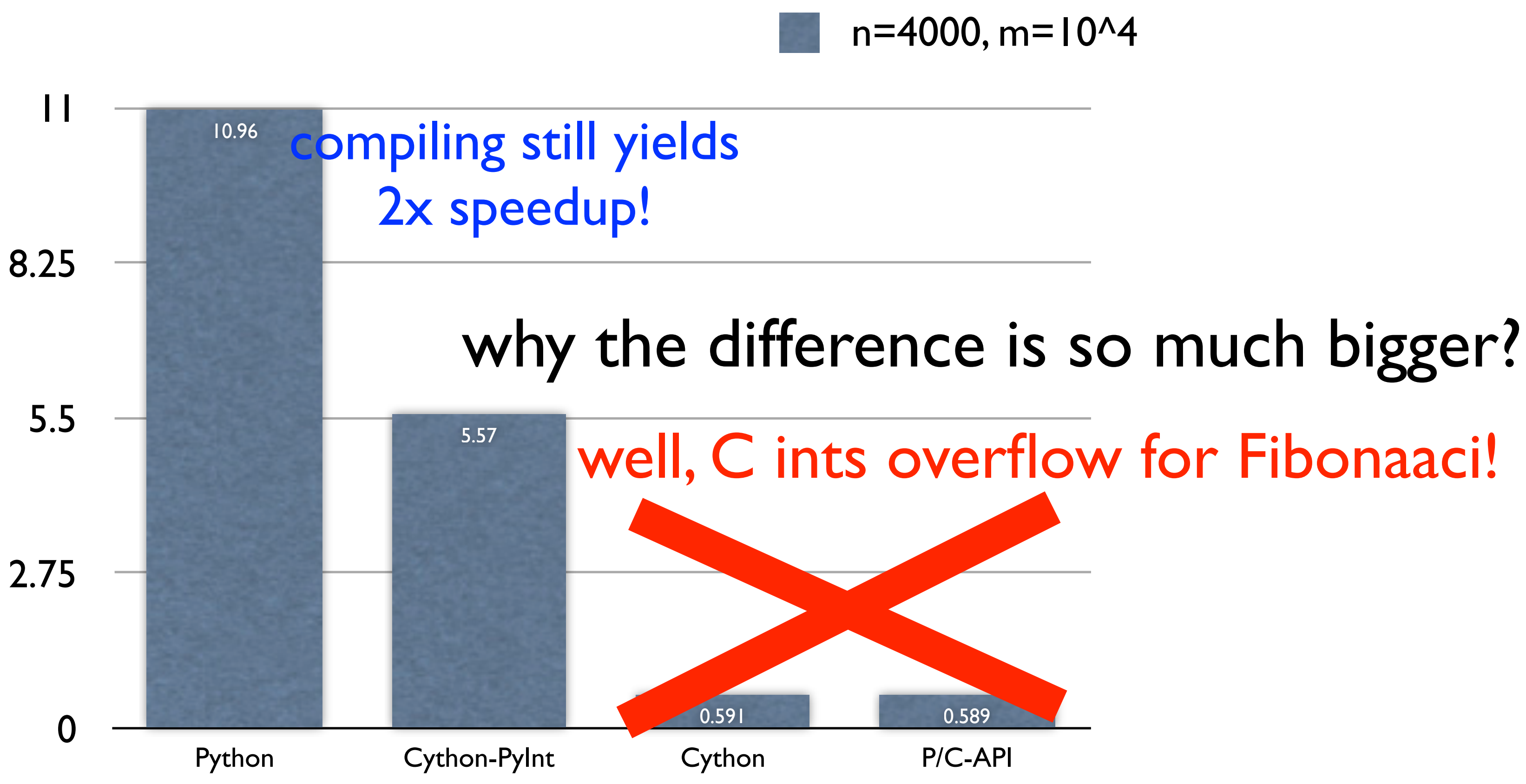
- python vs cython vs python/c-api
  - cython variation: with “cdef int” or python int
  - python/c-api variation: Py_DECREF() or not



Python is 5x slower than Cython/C-API

Benchmark

- python vs cython vs python/c-api
  - cython variation: with “cdef int” or python int
  - python/c-api variation: Py_DECREF() or not



compiling still yields 2x speedup!

why the difference is so much bigger?

well, C ints overflow for Fibonaaci!

Arbitrary Precision Arithmetic

- Python has transparent arbitrary precision arithmetic!
  - Python int range depends on 32-bit or 64-bit architecture
  - when exceeds the range, int becomes long automatically
  - Python long has arbitrary precision (and will stay long)

```
>>> sys.maxint
9223372036854775807
>>> math.log(sys.maxint, 2)
63.0
>>> sys.maxint + 1
9223372036854775808L
>>> -sys.maxint
-9223372036854775807
>>> -sys.maxint - 1
-9223372036854775808
>>> -sys.maxint - 2
-9223372036854775809L
>>> sys.maxint + 1 - 2
9223372036854775806L
>>> int(0L), int(9223372036854775808L)
0, 9223372036854775808L
```

C/C++	int	long	long long
32-bit	32		64
64-bit	32	64	

```
n=100 m=10000
-980107325
  0.029162      <module 'fib_c'>

n=100 m=10000
354224848179261915075
  0.172386      <module 'fib_py'>
```

Conclusions So Far

- Python/C API is much faster than Python (5x speedup)
  - but tricky and involved: reference counts
  - using API improves our understanding of Python (under the hood)
- Cython translates subset of Python to C++ calling C API
  - implicit use of API; almost as fast as API in most tasks
  - reading compiled .cpp <==improves==> understanding of API
- BTW Python has arbitrary precision arithmetic unlike C/C++
- but where are the real advantages of API over Cython??
  - if you want to modify existing Python API, e.g., defaultdict
  - if you want more flexible OOP (defining new extension types)
  - if you want better portability of your module (w/o Cython)

Cython for OOP

- motivations: Python int is both high-precision and immutable
- immutable int/float makes dictionary update very slow!
  - immutable x+=y is interpreted as x=x+y; many hashes!
- what if we just want fixed-precision and mutable int/double?

```
class A(object):
    def __hash__(self):
        print "hash"
        return 1

>>> a = A()
>>> d = {}
>>> d[a] = 1
hash
>>> d[a] += 1
hash
hash
```

```
>>> from collections import \
... defaultdict
>>> d = defaultdict(int)
>>> d[a]
hash
hash
0
>>> b = A()
>>> d[b] += 1
hash
hash
hash
```

Cython for OOP

- here public is important, otherwise v is not accessible
- calling __iadd__ (but not +=) avoids double hash problem

```
cdef class Double:
    cdef public double v

    def __init__(self, v=0):
        self.v = v

    def __iadd__(self, other):
        self.v += float(other)
        return self

    def __float__(self):
        return self.v

    def __repr__(self):
        return str(self.v)
```

```
>>> d = defaultdict(Double)
>>> d[a]
hash
hash
0.0
>>> b = A()
>>> d[b] += 1
hash
hash
hash
>>> d[b] += 1
hash
hash
>>> d[b].__iadd__(1)
hash
2.0
>>> c = A()
>>> d[c].__iadd__(1)
hash
hash
1.0
```