HYBRID VIGOR

CODING WITH PYTHON & C++

PYTHON

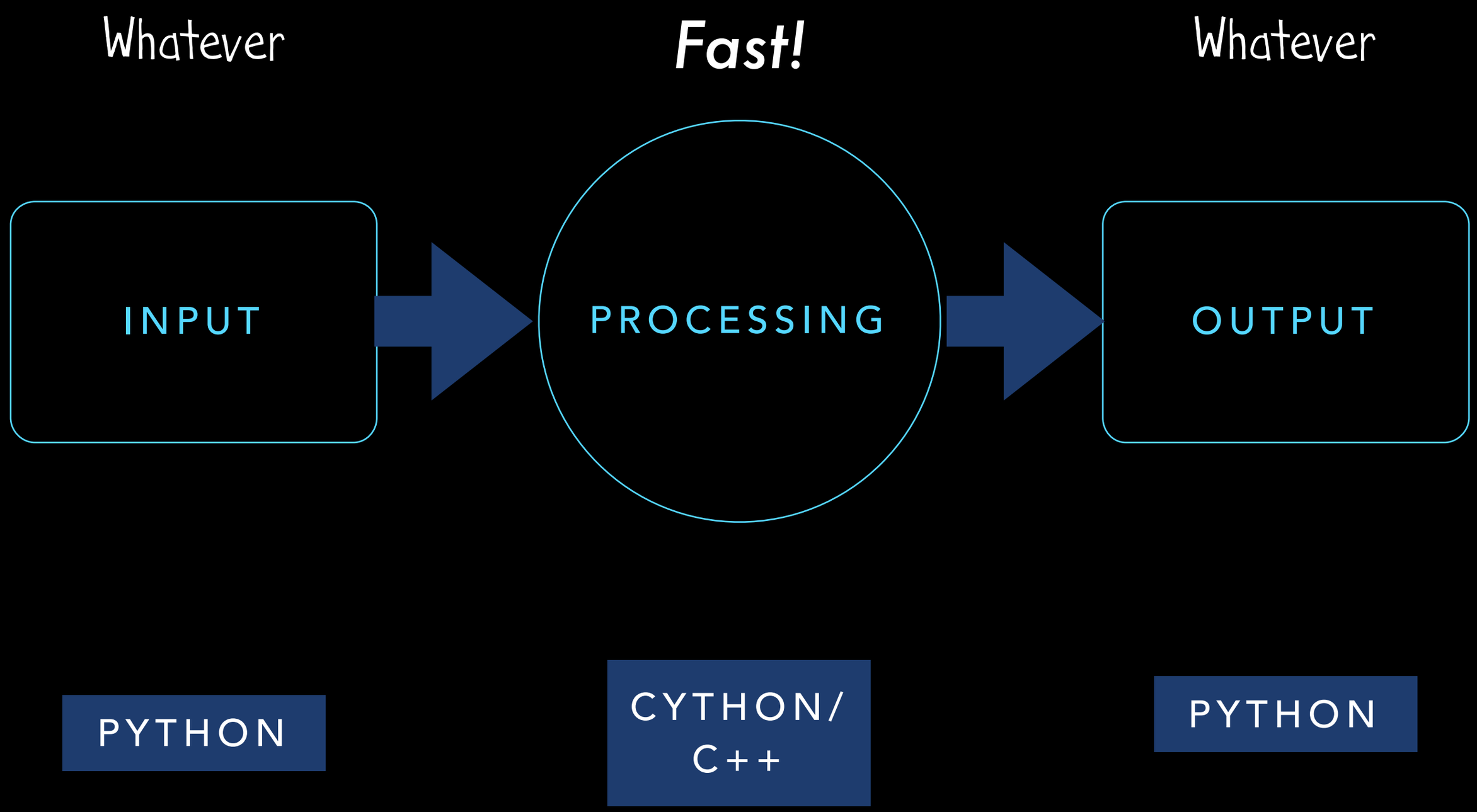
EASY TO LEARN (FORGIVING)
EASY TO WRITE
EASY TO READ
EASY TO STEAL RE-USE CODE
CONNECTS TO EVERYTHING
FUN!

CYTHON

C++

MUCH SLOWER TO WRITE
MORE VERBOSE
FUN(?)
FAST ("C" WITH FANCY CLOTHES)

PHYLOGENETICS SOFTWARE



PYTHON EXTENSIONS

helloworld.c

```c
#include <Python.h>

static PyObject* helloworld(PyObject* self)
{
    return Py_BuildValue("s", "Hello, Python extensions!!");
}

static char helloworld_docs[] =
    "helloworld( ): Any message you want to put here!!\n";

static PyMethodDef helloworld_funcs[] = {
    {"helloworld", (PyCFunction)helloworld,
     METH_NOARGS, helloworld_docs},
    {NULL}
};

void inithelloworld(void)
{
    Py_InitModule3("helloworld", helloworld_funcs,
                   "Extension module example!");
}
```

setup.py

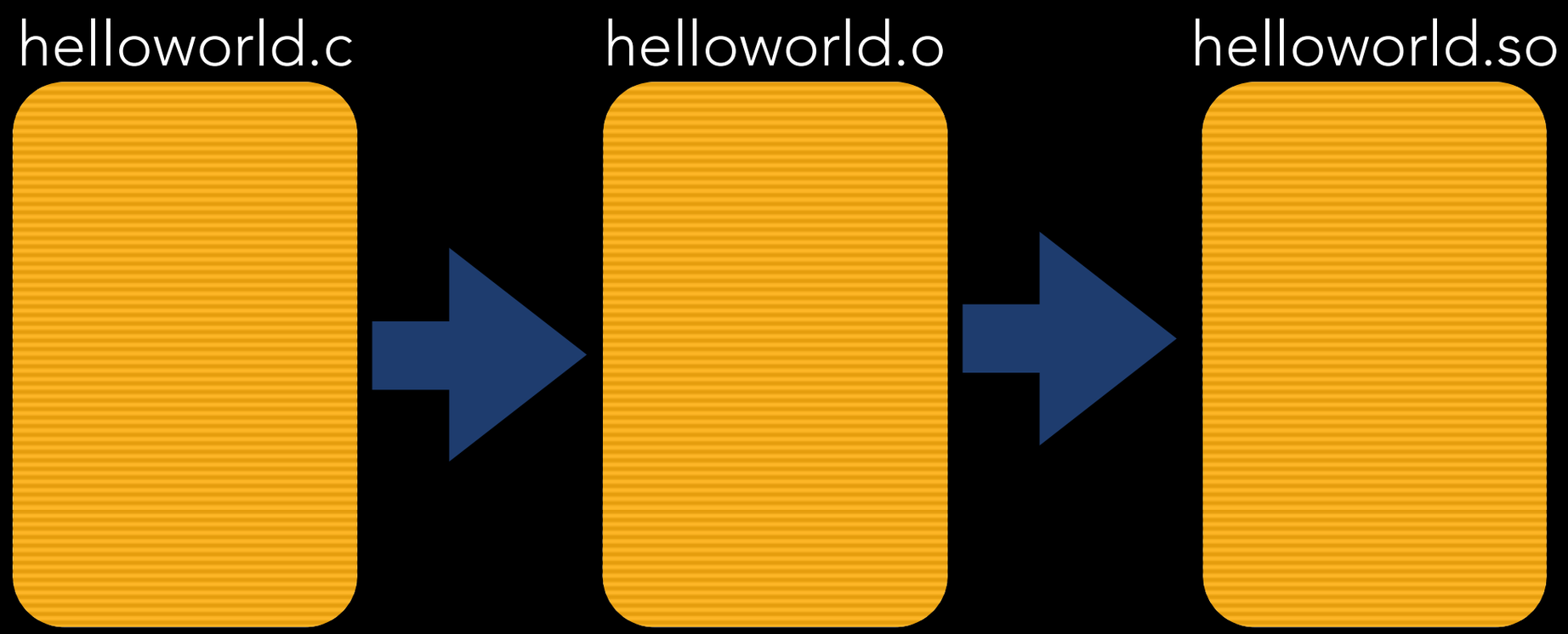
```python
from distutils.core import setup, Extension
setup(name='helloworld', version='1.0',  \
      ext_modules=[Extension('helloworld', ['hello.c'])])
```

run.py

```python
import helloworld

print helloworld.helloworld()
```

http://www.tutorialspoint.com/python/python_further_extensions.htm

```
>python setup.py
```




```
>python run.py
Hello, Python Extensions!!
```

CYTHON EXTENSIONS

hello.pyx

```
def hello():
    return "Hello from Cython"

~
~
~
~
~
~
```

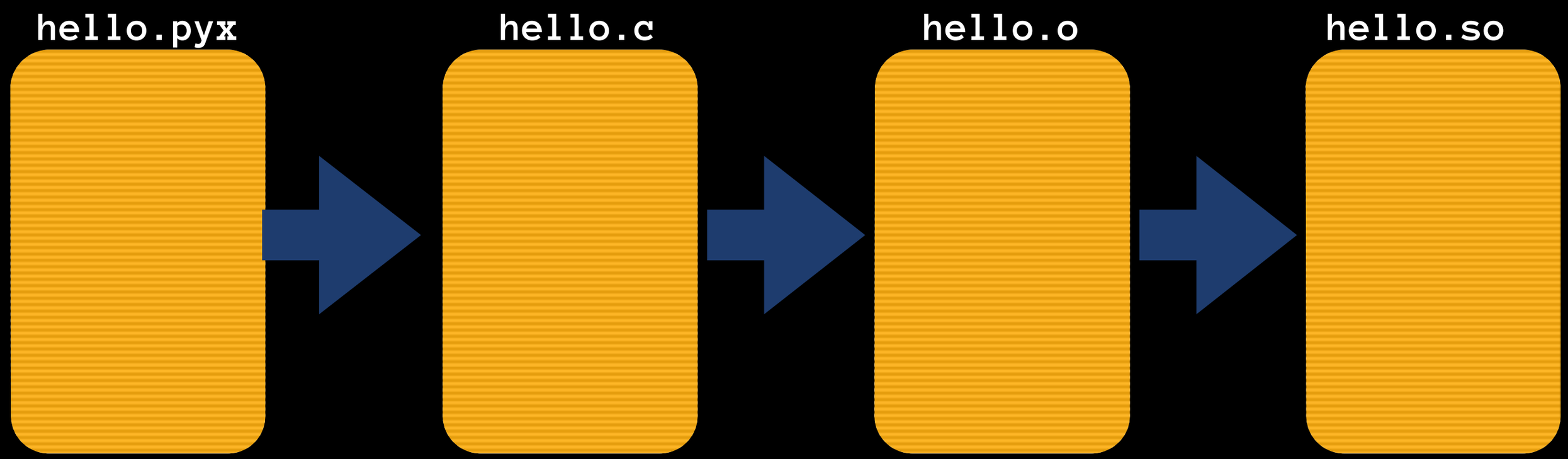
setup.py

```
from distutils.core import setup
from Cython.Build import cythonize
import sys

# HACK
sys.argv = ['setup.py', 'build_ext', '--inplace']

setup(
    name = "hello app",
    ext_modules = cythonize('hello.pyx'),  # accepts a glob pattern
)
~
~
~
~
```

run.py

```
import hello

print hello.hello()
~
~
```

```
>python setup.py
```




```
>python run.py
Hello from Cython
```

```c
/* Generated by Cython 0.21.1 */

#define PY_SSIZE_T_CLEAN
#ifndef CYTHON_USE_PYLONG_INTERNALS
#ifdef PYLONG_BITS_IN_DIGIT
#define CYTHON_USE_PYLONG_INTERNALS 0
#else
#include "pyconfig.h"
#ifdef PYLONG_BITS_IN_DIGIT
#define CYTHON_USE_PYLONG_INTERNALS 1
#else
#define CYTHON_USE_PYLONG_INTERNALS 0
#endif
#endif
#endif
#include "Python.h"
#ifndef Py_PYTHON_H
    #error Python headers needed to compile C extensions, please install developm
#elif PY_VERSION_HEX < 0x02060000 || (0x03000000 <= PY_VERSION_HEX && PY_VERSION_
    #error Cython requires Python 2.6+ or Python 3.2+.
#else
#define CYTHON_ABI "0_21_1"
#include <stddef.h>
#ifndef offsetof
#define offsetof(type, member) ( (size_t) & ((type*)0) -> member )
#endif
#if !defined(WIN32) && !defined(MS_WINDOWS)
  #ifndef __stdcall
    #define __stdcall
  #endif
  #ifndef __cdecl
    #define __cdecl
  #endif
  #ifndef __fastcall
    #define __fastcall
  #endif
```

+0 ~0 -0 master 01-hello-world/hello.c 1% 26: 1

Over 1000 lines of C!

CYTHON SYNTAX

It's like python, with extra bits

sum.pyx

```
def sum_1():
    total = 0
    for i in range(1000):
        total += i
    return total

def sum_2():
    cdef int total = 0
    for i in range(1000):
        total += i
    return total

def sum_3():
    cdef int total = 0
    cdef int i
    for i in range(1000):
        total += i
    return total

~
~
~
~
~
~
```

```
$ python run.py
sum_1() x10000 0.34024 sec
sum_2() x10000 0.46766 sec
sum_3() x10000 0.00051 sec
```

```
$ cython -a sum.pyx
$ open sum.html
```

Raw output: sum.c

```
+01: def sum_1():
+02:     total = 0
+03:     for i in range(10):
+04:         total += i
+05:     return total
 06:
+07: def sum_2():
+08:     cdef int total = 0
+09:     for i in range(10):
+10:         total += i
+11:     return total
 12:
+13: def sum_3():
+14:     cdef int total = 0
 15:     cdef int i
+16:     for i in range(10):
+17:         total += i
+18:     return total
 19:
```

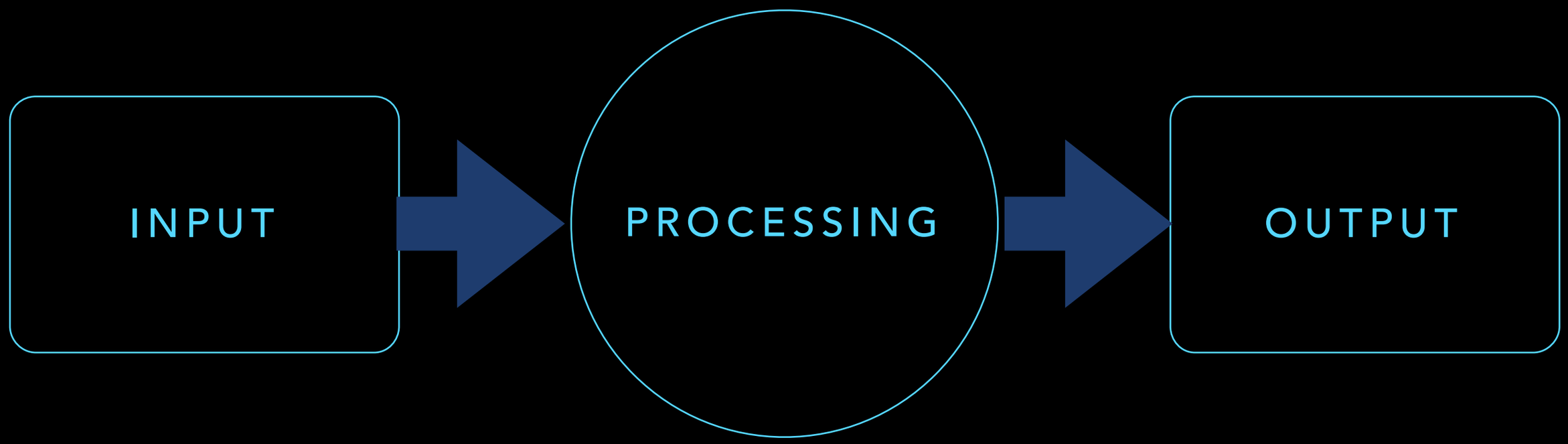
Raw output: sum.c

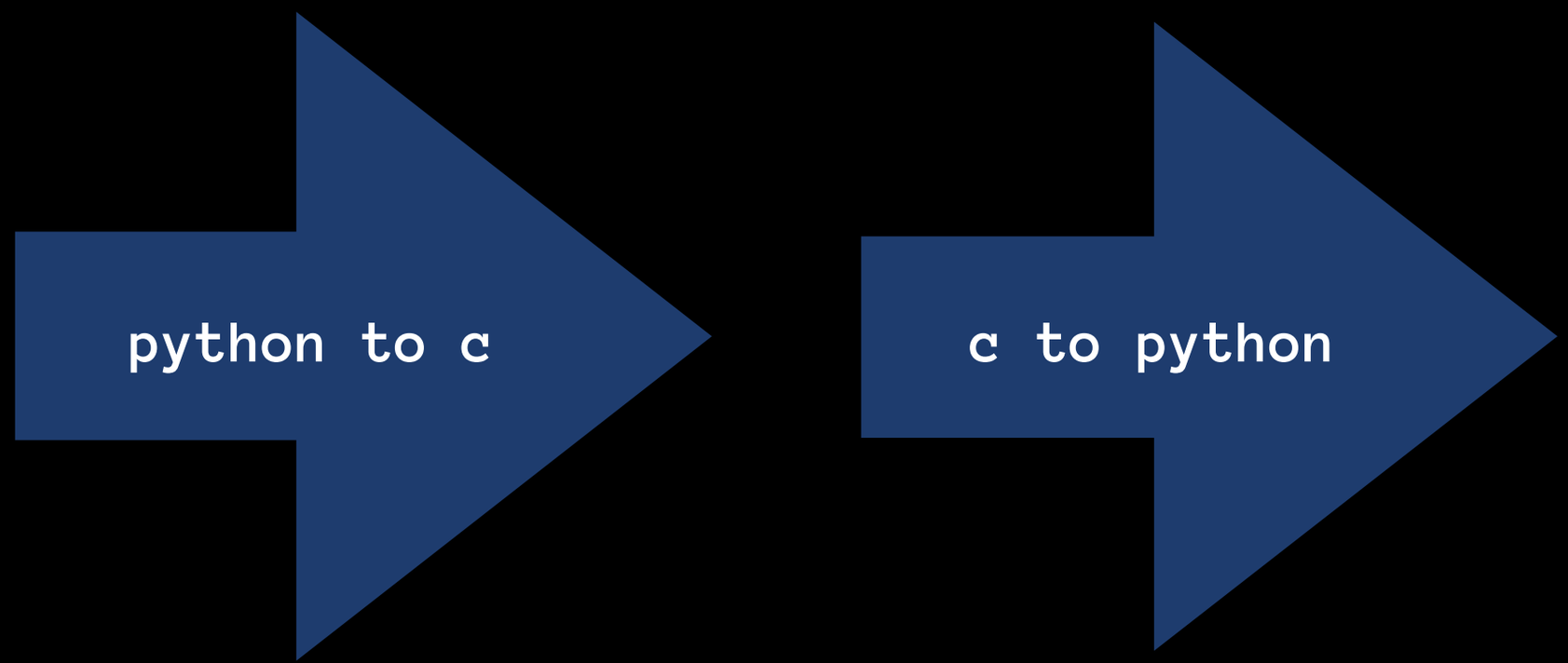
```
+01: def sum_1():
+02:     total = 0
+03:     for i in range(10):
+04:         total += i
+05:     return total
 06:
+07: def sum_2():
+08:     cdef int total = 0
+09:     for i in range(10):
+10:         total += i
        __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_total); if (unlikely(!__pyx_t_1)) {__pyx_filena
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_t_5 = PyNumber_InPlaceAdd(__pyx_t_1, __pyx_v_i); if (unlikely(!__pyx_t_5)) {__pyx_
        __Pyx_GOTREF(__pyx_t_5);
        __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
        __pyx_t_6 = __Pyx_PyInt_As_int(__pyx_t_5); if (unlikely((__pyx_t_6 == (int)-1) && PyErr_
        __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
        __pyx_v_total = __pyx_t_6;
+11:     return total
 12:
+13: def sum_3():
+14:     cdef int total = 0
 15:     cdef int i
+16:     for i in range(10):
+17:         total += i
+18:     return total
 19:
```

Raw output: sum.c

```
+01: def sum_1():
+02:     total = 0
+03:     for i in range(10):
+04:         total += i
+05:     return total
 06:
+07: def sum_2():
+08:     cdef int total = 0
+09:     for i in range(10):
+10:         total += i
+11:     return total
 12:
+13: def sum_3():
+14:     cdef int total = 0
 15:     cdef int i
+16:     for i in range(10):
    for (__pyx_t_1 = 0; __pyx_t_1 < 10; __pyx_t_1+=1) {
      __pyx_v_i = __pyx_t_1;
+17:         total += i
      __pyx_v_total = (__pyx_v_total + __pyx_v_i);
    }
+18:     return total
 19:
```

GETTING DATA IN & OUT


INPUT
PROCESSING
OUTPUT



- Basic types (int/float/string) are automatic
- Complex data?
- 2 ways: Numpy and C++ containers

numpy_test.pyx

```
# distutils: include_dirs = NUMPY_PATH
# cython: wraparound=False
# cython: cdivision=True
# cython: boundscheck=False

cimport numpy as np
import numpy

def sum_array(np.npy_double[:, :] arr):
    cdef:
        size_t i, j, maxi, maxj
        double total

    maxi = arr.shape[0]
    maxj = arr.shape[1]

    output = numpy.zeros(maxi)
    cdef:
        np.npy_double[:] numpy_output = output

    for i in range(maxi):
        total = 0.0
        for j in range(maxj):
            total += arr[i, j]
        numpy_output[i] = total

    return output

~
```

setup.py

```python
from distutils.core import setup
from Cython.Build import cythonize
import sys
import numpy

# HACK
sys.argv = ['setup.py', 'build_ext', '--inplace']

setup(
    name = "tutorial app",
    ext_modules = cythonize(
        '*.pyx',
        aliases = {'NUMPY_PATH': numpy.get_include()},
    ),
)
~
~
~
~
~
~
```

run.py

```python
import numpy
import numpy_test

arr = numpy.random.uniform(0.0, 1.0, (10, 10))
print arr
print arr.sum(axis=1)
print numpy_test.sum_array(arr)
```

```
$ python run.py
[[  4.54105797e-01   7.65364017e-01   6.12619333e-01   8.77424111e-01
    6.32056419e-01   7.75373849e-01   2.46884938e-01   4.47860114e-01
    1.62014999e-01   8.98034441e-01]
 [  8.53395405e-03   1.68066584e-01   1.80012758e-01   1.07537136e-01
    3.32722587e-01   3.83744452e-01   9.06816280e-02   1.82682212e-02
    7.81411272e-01   4.73416355e-01]
 [  4.15811863e-01   3.82673855e-01   1.71541640e-01   4.79506188e-01
    8.79467165e-01   3.44003649e-01   8.77906821e-02   8.23502967e-01
    6.93018703e-01   7.86383446e-01]
 [  6.88509461e-01   1.40541462e-01   2.80048255e-01   6.95036216e-01
    4.81759835e-04   7.21842144e-01   1.79343638e-01   3.76973780e-01
    3.49202148e-01   5.36043505e-01]
 [  9.58163507e-02   4.39099243e-01   8.77928321e-01   4.60450858e-01
    3.58451963e-01   7.69028381e-01   9.14248212e-01   8.87815443e-01
    1.50878021e-01   7.43187013e-01]
 [  8.14148990e-01   3.22414773e-01   2.10553860e-02   7.92996405e-02
    1.60045328e-01   6.03488104e-02   8.55185068e-01   8.99609698e-01
    9.00184374e-01   5.80212321e-01]
 [  3.52722981e-01   5.19253576e-01   4.97861488e-01   3.59872860e-01
    2.19956428e-01   3.92001532e-01   6.67249417e-02   3.70162464e-01
    2.49242875e-01   3.54085032e-01]
 [  2.86633186e-01   6.50578598e-01   9.92535319e-01   9.10387074e-01
    9.15315268e-01   7.59902285e-01   3.64823042e-01   4.62678340e-01
    6.05862378e-01   8.64974512e-01]
 [  9.76060586e-01   7.14995067e-01   2.39875802e-01   1.24098307e-01
    7.00780033e-01   9.63891885e-01   1.52248155e-01   7.54379113e-01
    5.88919399e-01   1.51361233e-01]
 [  3.14883767e-01   2.39727313e-01   6.75604464e-01   8.53599657e-01
    4.83992712e-01   4.40402028e-01   2.05120182e-01   3.81040745e-01
    6.38881174e-01   1.10188688e-01]]
[ 5.87173802  2.54439495  5.06370016  3.96802237  5.69690381  4.69250439
  3.38188418  6.81369     5.36660958  4.34344073]
[ 5.87173802  2.54439495  5.06370016  3.96802237  5.69690381  4.69250439
  3.38188418  6.81369     5.36660958  4.34344073]
```

C++ CONTAINERS

test_cpp.pyx

```
# distutils: language = c++

from libcpp.map cimport map as cpp_map
from libcpp.vector cimport vector
from libcpp.string cimport string

def make_map():
    cdef cpp_map[string,int] mymap

    mymap['bob'] = 10
    mymap['sue'] = 100

    return mymap

def make_vector():
    cdef vector[int] myvec
    cdef int i

    for i in range(20):
        myvec.push_back(i)

    return myvec

ctypedef cpp_map[int, string] lookup_t

def get_map(lookup_t lookup):

    lookup[10] = 'ten'
    lookup[20] = 'twenty'

    return lookup
```

run.py

```
from test_cpp import *

print make_map()
print make_vector()

d = {
    1 : "bla",
    2 : "boodle",
}
print get_map(d)

d['error'] = 99
print get_map(d)
~
```

```
$ python run.py
{'bob': 10, 'sue': 100}
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19]
{1: 'bla', 2: 'boodle', 20: 'twenty', 10: 'ten'}
Traceback (most recent call last):
  File "run.py", line 13, in <module>
    print get_map(d)
  File "test_cpp.pyx", line 26, in test_cpp.get_map
(test_cpp.cpp:875)
    def get_map(lookup_t lookup):
  File "stringsource", line 215, in
map.from_py.__pyx_convert_map_from_py_int____std_3a_
_3a_string (test_cpp.cpp:1084)
TypeError: an integer is required
```

WRAPPING C++ CLASSES

wrapped.hpp

```cpp
#pragma once

#include <random>
#include <string>

namespace mystuff {

typedef std::mt19937 random_engine_t;

struct MyClass
{
    MyClass(std::string name, size_t seed);
    double uniform();

    std::string name;
    random_engine_t engine;
};

}
```

wrapped.cpp

```cpp
#include "wrapped.hpp"

using namespace mystuff;

MyClass::MyClass(std::string name_, size_t seed_)
    : name(name_)
    , engine(seed_)
{
}

double MyClass::uniform()
{
    std::uniform_real_distribution<> uni;
    return uni(engine);
}
```

wrapped.pxd

```
from libcpp.string cimport string

cdef extern from "wrapped.hpp" namespace "mystuff":
    cdef cppclass MyClass:
        MyClass(string name, size_t seed)
        double uniform()
        string name
~
~
```

_wrap.pyx

```
# distutils: language = c++

from wrapped cimport *
from libcpp cimport string

cdef class PyClass:
    cdef:
        MyClass *_this

    def __cinit__(self, string name, size_t seed):
        self._this = new MyClass(name, seed)

    def __dealloc__(self):
        if self._this != NULL:
            del self._this

    def uniform(self):
        return self._this.uniform()

    def named_pairs(self, size=5):
        return [(self.name, self._this.uniform()) for _ in range(size)]

    property name:
        def __get__(self):
            return self._this.name
        def __set__(self, string name):
            self._this.name = name
```

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Build import cythonize
import sys

# HACK
sys.argv = ['setup.py', 'build_ext', '--inplace']

extensions = [
    Extension(
        "_wrap",
        ["_wrap.pyx", "wrapped.cpp"],
        extra_compile_args = [
            '-Wno-unused-function',
            '-stdlib=libc++',
            '-std=c++11',
            '-mmacosx-version-min=10.8',
        ],
    )
]
setup(
    name='wrapping-example',
    ext_modules=cythonize(
        extensions,
    )
)

~
```

```python
from _wrap import PyClass

monkey = PyClass('monkey', 10)
for i in range(5):
    print monkey.uniform(),
print

horsey = PyClass('horsey', 10)
print horsey.named_pairs(5)

horsey.name = 'doggy'
print horsey.named_pairs(5)
```

```
$ python run.py
0.298761158663 0.49458992831 0.443014946215 0.831911361951
0.583321742131
[('horsey', 0.29876115866269), ('horsey', 0.49458992830960824),
('horsey', 0.4430149462152384 6), ('horsey', 0.831911361950636 9),
('horsey', 0.583321742131027 2)]
[('doggy', 0.025171733746490593), ('doggy', 0.7092080102763285),
('doggy', 0.2655661277846737 6), ('doggy', 0.2636028477384501),
('doggy', 0.1503778672982798 4)]
```

WHY BOTHER?

- Tiger (Tree Independent Generation of Evolutionary Rates)
- FAST_tiger (~10,000 lines of code)
- Tigger (~500 lines of code)
- Tigger is faster.

- Start with python
- Make the algorithm work, clearly
- Write a cython version
- Use the python version in the test suite