A Cython Walkthrough

Nick Murdoch

```
Code will be displayed at this size on
white. Slides: tinyurl.com/cywalk
```

Hello

Nick Murdoch
@nickmurdoch

- Python - 12 years
- C - 9 years
- User of Cython - 9 y

So what is Cython?

- A programming language
- Looks like Python
- Smells like Python...

So what is Cython?

tinyurl.com/cywalk

.py / .pyx — Python/Cython source code

↓ cython

.c — C source code

↓ (eg) gcc

.so / .pyd — Importable from CPython

Uses for Cython

1. Faster Python code
2. Call Python from C
3. Call C from Python

Uses for Cython

1. Faster Python code
2. Call Python from C
3. Call C from Python

- Example problem
  - A function to find the maximum product of *n* consecutive digits in a list
  - Eg: digits = [6, 2, 3, 6, 5, 4]; n = 3
  - Result = 6 × 5 × 4 = 120

```python
# bigproduct.py:

def bigproduct(digits, n=13):
    '''Return the biggest product of n
        consecutive digits'''
    best = 0
    for i in range(len(digits) - (n - 1)):



    return best
```

```python
# bigproduct.py:

def bigproduct(digits, n=13):
    '''Return the biggest product of n
        consecutive digits'''
    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            product *= digits[i + j]


    return best
```

```python
# bigproduct.py:

def bigproduct(digits, n=13):
    '''Return the biggest product of n
        consecutive digits'''
    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            product *= digits[i + j]
        if product > best:
            best = product
    return best
```
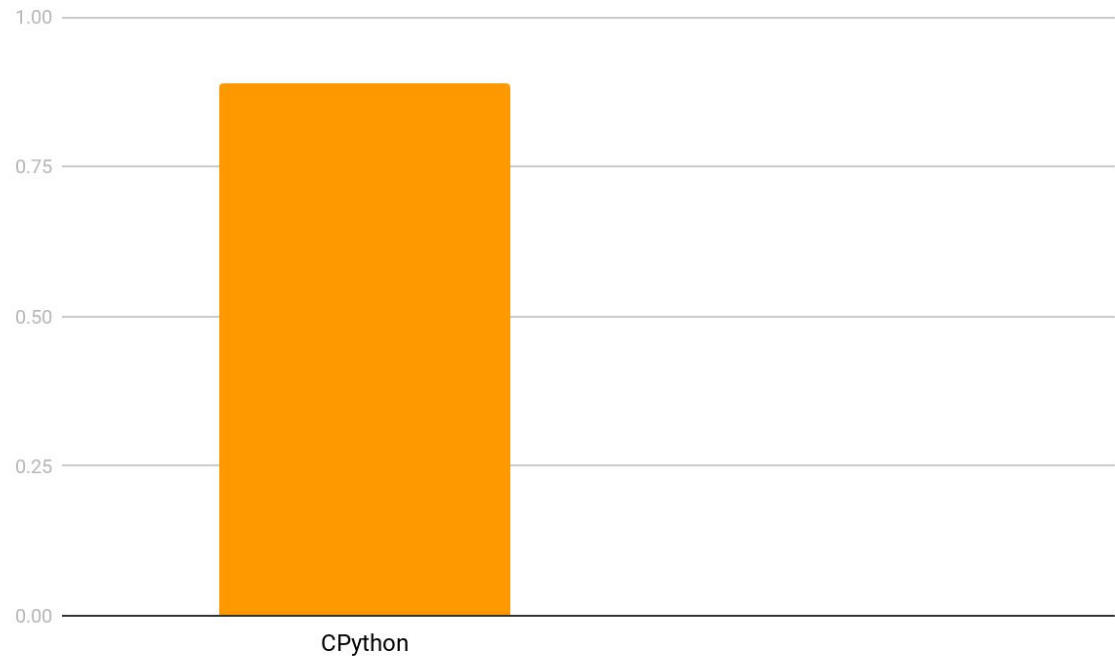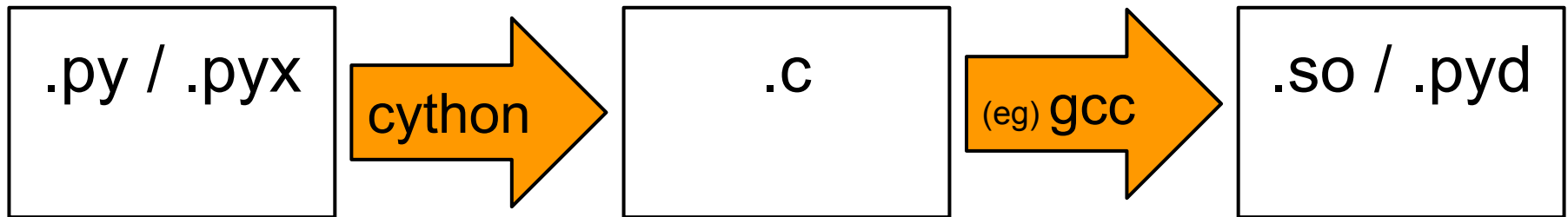
Timing, 1000 runs:

- Pure Python
  - 0.89s

```
.py / .pyx    → cython →    .c    → (eg) gcc →    .so / .pyd
```
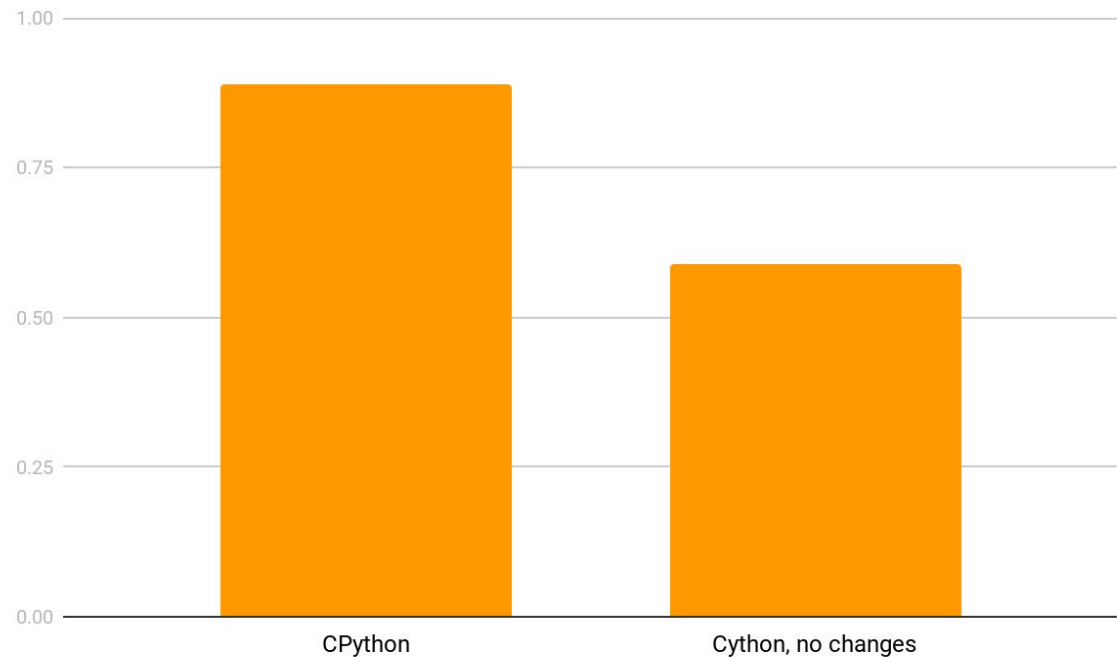
```python
# setup.py:

from distutils.core import setup
    # or `from setuptools import setup`
from Cython.Build import cythonize

setup(
    # …
    ext_modules = cythonize("bigproduct.py")
)
```

Timing, 1000 runs:

- Pure Python:
  - 0.89s
- Cython, no code changes:
  - 0.59s

1. Faster Python code

- What's taking the most time?

- Python!

- Where are the most calls into Python?

- `$ cython -a bigproduct.py`

  ➜ bigproduct.html

Generated by Cython 0.28.5

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C c
Cython generated for it.

Raw output: bigproduct.c

```
 01:
+02: def bigproduct(digits, n=13):
 03:     '''Return the biggest product of n consecutive digits''
+04:     best = 0
+05:     for i in range(len(digits) - (n - 1)):
+06:         product = 1
+07:         for j in range(n):
+08:             product *= digits[i + j]
+09:         if product > best:
+10:             best = product
+11:     return best
 12:
```

Generated by Cython 0.28.5

<u>Yellow lines</u> hint at Python interaction.
Click on a line that starts with a "+" to see the C c
Cython generated for it.

Raw output: <u>bigproduct.c</u>

```
 01:
+02: def bigproduct(digits, n=13):
 03:     '''Return the biggest product of n consecutive digits'''
+04:     best = 0
+05:     for i in range(len(digits) - (n - 1)):
+06:         product = 1
+07:         for j in range(n):
+08:             product *= digits[i + j]
+09:         if product > best:
+10:             best = product
+11:     return best
 12:
```
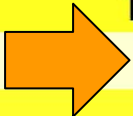
Generated by Cython 0.28.5

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C c
Cython generated for it.

Raw output: bigproduct.c

```
 01:
+02: def bigproduct(digits, n=13):
 03:     '''Return the biggest product of n consecutive digits''
+04:     best = 0
+05:     for i in range(len(digits) - (n - 1)):
+06:         product = 1
    __Pyx_INCREF(__pyx_int_1);
    __Pyx_XDECREF_SET(__pyx_v_product, __pyx_int_1);
+07:         for j in range(n):
+08:             product *= digits[i + j]
+09:         if product > best:
+10:             best = product
+11:     return best
 12:
```

Generated by Cython 0.28.5

<mark>Yellow lines</mark> hint at Python interaction.
Click on a line that starts with a "+" to see the C c
Cython generated for it.

Raw output: bigproduct.c

```
 01:
+02: def bigproduct(digits, n=13):
 03:     '''Return the biggest product of n consecutive digits''
+04:     best = 0
+05:     for i in range(len(digits) - (n - 1)):
+06:         product = 1
    __Pyx_INCREF(__pyx_int_1);
    __Pyx_XDECREF_SET(__pyx_v_product, __pyx_int_1);
+07:         for j in range(n):
+08:             product *= digits[i + j]
+09:         if product > best:
+10:             best = product
+11:     return best
 12:
```
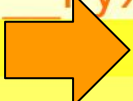
```python
def bigproduct(digits, n=13):
    '''Return the biggest product of n consecutive digits'''
    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
```
```c
__pyx_t_3 = __Pyx_PyObject_CallOneArg(__pyx_builtin_range, __pyx_v_n); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 7, __pyx_L1_error)
__Pyx_GOTREF(__pyx_t_3);
if (likely(PyList_CheckExact(__pyx_t_3)) || PyTuple_CheckExact(__pyx_t_3)) {
  __pyx_t_2 = __pyx_t_3; __Pyx_INCREF(__pyx_t_2); __pyx_t_6 = 0;
  __pyx_t_7 = NULL;
} else {
  __pyx_t_6 = -1; __pyx_t_2 = PyObject_GetIter(__pyx_t_3); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 7, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __pyx_t_7 = Py_TYPE(__pyx_t_2)->tp_iternext; if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 7, __pyx_L1_error)
}
__Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
for (;;) {
  if (likely(!__pyx_t_7)) {
    if (likely(PyList_CheckExact(__pyx_t_2))) {
      if (__pyx_t_6 >= PyList_GET_SIZE(__pyx_t_2)) break;
      #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
      __pyx_t_3 = PyList_GET_ITEM(__pyx_t_2, __pyx_t_6); __Pyx_INCREF(__pyx_t_3); __pyx_t_6++; if (unlikely(0 < 0)) __PYX_ERR(0, 7, __pyx_L1_error)
      #else
      __pyx_t_3 = PySequence_ITEM(__pyx_t_2, __pyx_t_6); __pyx_t_6++; if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 7, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_3);
      #endif
    } else {
      if (__pyx_t_6 >= PyTuple_GET_SIZE(__pyx_t_2)) break;
      #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
      __pyx_t_3 = PyTuple_GET_ITEM(__pyx_t_2, __pyx_t_6); __Pyx_INCREF(__pyx_t_3); __pyx_t_6++; if (unlikely(0 < 0)) __PYX_ERR(0, 7, __pyx_L1_error)
      #else
      __pyx_t_3 = PySequence_ITEM(__pyx_t_2, __pyx_t_6); __pyx_t_6++; if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 7, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_3);
      #endif
    }
  } else {
    __pyx_t_3 = __pyx_t_7(__pyx_t_2);
    if (unlikely(!__pyx_t_3)) {
      PyObject* exc_type = PyErr_Occurred();
      if (exc_type) {
        if (likely(__Pyx_PyErr_GivenExceptionMatches(exc_type, PyExc_StopIteration))) PyErr_Clear();
        else __PYX_ERR(0, 7, __pyx_L1_error)
      }
      break;
    }
    __Pyx_GOTREF(__pyx_t_3);
  }
  __Pyx_XDECREF_SET(__pyx_v_j, __pyx_t_3);
  __pyx_t_3 = 0;
… */
}
__Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
```
```python
            product *= digits[i + j]
        if product > best:
            best = product
    return best
```

1. Faster Python code

... but briefly: `for j in range(n):`

- Pyx_PyObject_CallOneArg(builtin_range, n)
- PyObject_GetIter()
- for (;;) {
  - j = tp_iternext()
  - PyErr_Occurred() (StopIteration? ➔ break)
  - [rest of block] }
- Also some optimisations for lists and tuples

```python
# bigproduct.py:
def bigproduct(digits, n=13):


    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            digit = digits[i + j]
            product *= digit
        if product > best:
            best = product
    return best
```

```python
# bigproduct.pyx:
def bigproduct(digits, n=13):


    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            digit = digits[i + j]
            product *= digit
        if product > best:
            best = product
    return best
```

```python
# bigproduct.pyx:
def bigproduct(digits, unsigned int n=13):



    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            digit = digits[i + j]
            product *= digit
        if product > best:
            best = product
    return best
```

```python
# bigproduct.pyx:
def bigproduct(digits, unsigned int n=13):
    cdef size_t i, j

    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            digit = digits[i + j]
            product *= digit
        if product > best:
            best = product
    return best
```

```
# bigproduct.pyx:
def bigproduct(digits, unsigned int n=13):
    cdef size_t i, j
    cdef unsigned int digit
    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            digit = digits[i + j]
            product *= digit
        if product > best:
            best = product
    return best
```

```
# bigproduct.pyx:
def bigproduct(digits, unsigned int n=13):
    cdef size_t i, j
    cdef unsigned int digit
    cdef unsigned long long product, best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            digit = digits[i + j]
            product *= digit
        if product > best:
            best = product
    return best
```

Generated by Cython 0.28.5

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C co
Cython generated for it.

Raw output: bigproduct cythonoverflow.c

Earlier
colouring

```
 01: import cython
 02:
+03: def bigproduct(digits, unsigned int n = 13):
 04:     cdef size_t i, j
 05:     cdef unsigned int digit
+06:     cdef unsigned long long best = 0, product
+07:     for i in range(len(digits) - (n - 1)):
+08:         product = 1
+09:         for j in range(n):
+10:             digit = digits[i + j]
+11:             product *= digit
+12:         if product > best:
+13:             best = product
+14:     return best
```

Timing, 1000 runs:

- Pure Python:
  - 0.89s
- Cython, no code changes:
  - 0.59s
- Cython, fully converted:
  - 0.06s
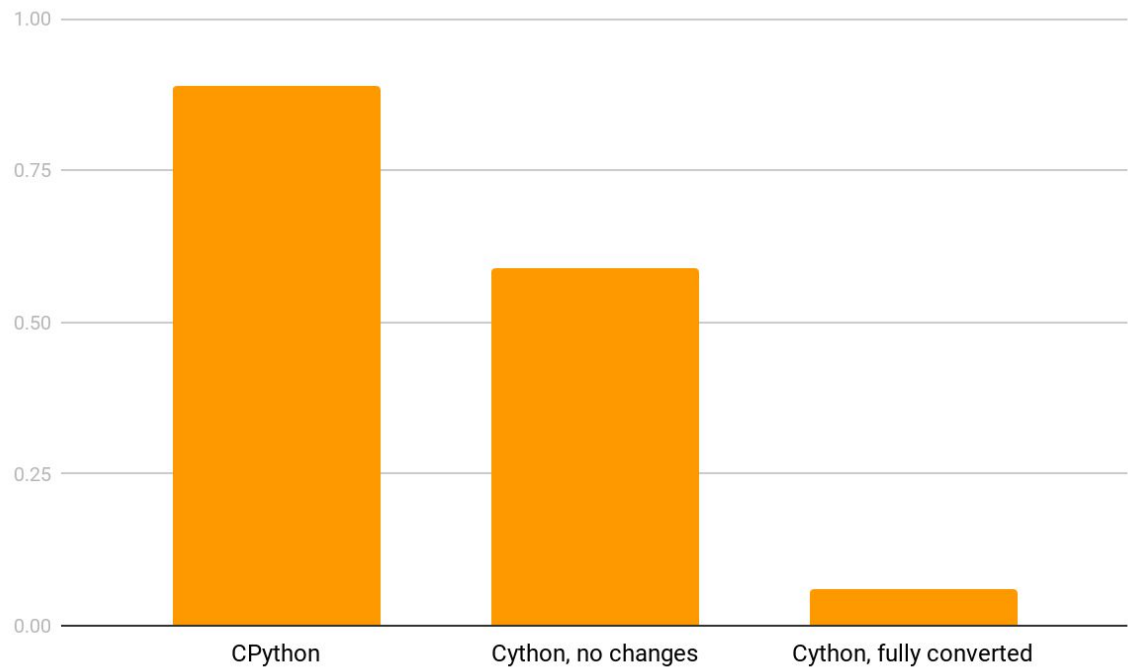
Uses for Cython ➡

1. Faster Python code
2. Call Python from C
3. Call C from Python

But why?

- Script customisable tasks with Python
- C program with Python plugins
- Fast prototyping of features
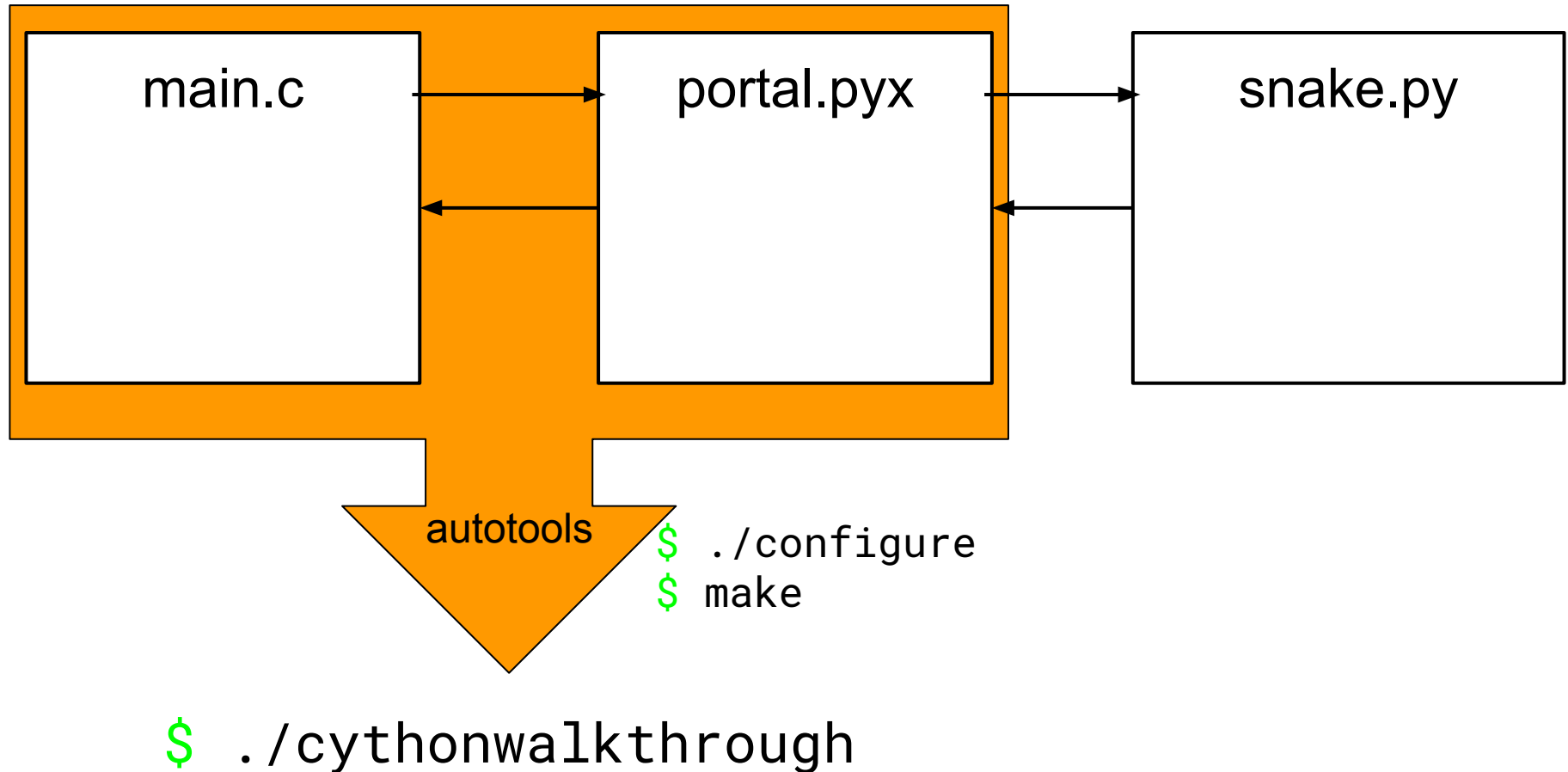
```
main.c  →  portal.pyx  →  snake.py
```

autotools

```
$ ./configure
$ make
```

```
$ ./cythonwalkthrough
```

main.c → portal.pyx → snake.py

portal.pyx → main.c

snake.py → portal.pyx

autotools

```
$ ./configure
$ make
```

```
$ ./cythonwalkthrough
```

Freely editable

```
main.c   →   portal.pyx   →   snake.py
         ←                ←   combine()
```
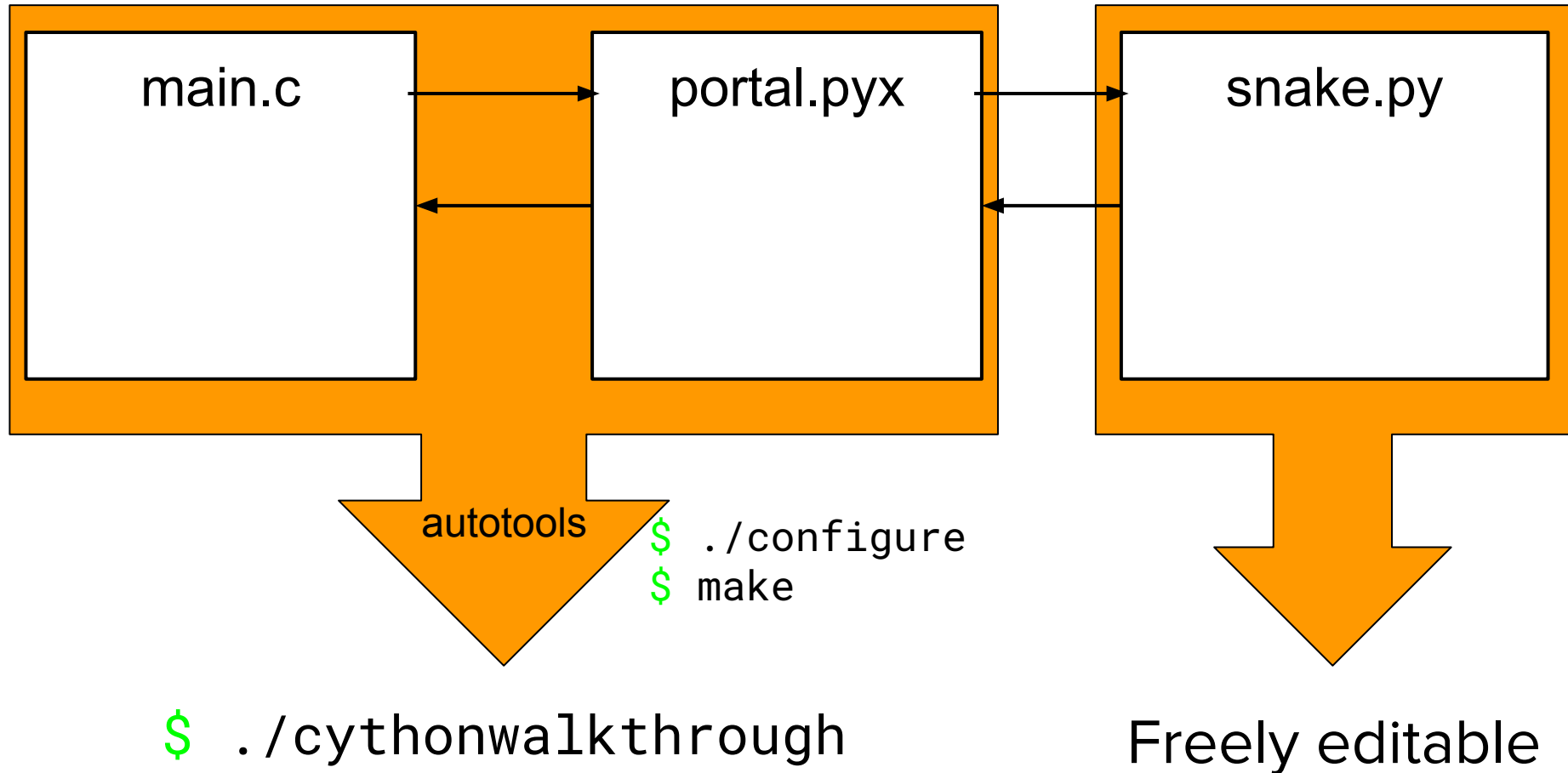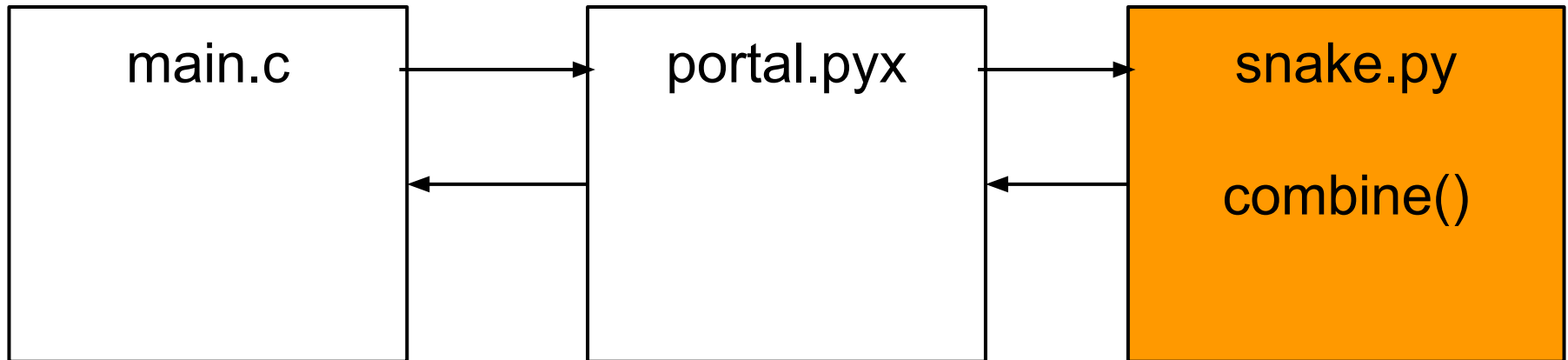
```
# snake.py:

def combine(a, b):
    print('hello from Python')
    return a ** b
```

```
┌─────────────┐      ┌─────────────────┐      ┌─────────────┐
│             │ ───► │   portal.pyx    │ ───► │  snake.py   │
│   main.c    │      │                 │      │             │
│             │ ◄─── │   public int    │ ◄─── │  combine()  │
│             │      │ entrance(int, int)│    │             │
└─────────────┘      └─────────────────┘      └─────────────┘
```
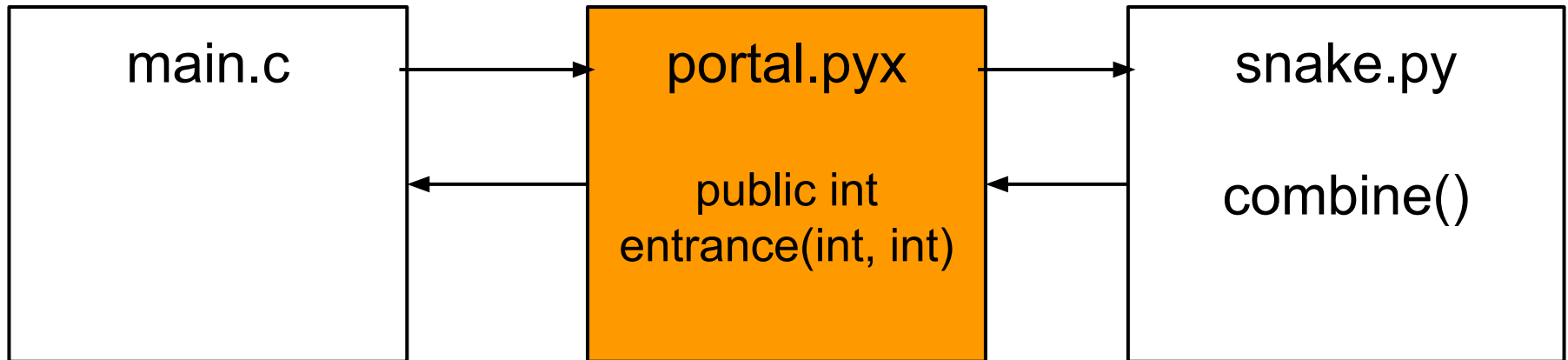
```python
# portal.pyx:

import snake

cdef public int entrance(int a, int b) except? -1:
    print('hello from Cython')
    return snake.combine(a, b)
```

```
┌─────────────┐      ┌──────────────────┐      ┌──────────────┐
│             │ ───> │   portal.pyx     │ ───> │  snake.py    │
│   main.c    │      │                  │      │              │
│             │ <─── │   public int     │ <─── │  combine()   │
│             │      │ entrance(int, int)│     │              │
└─────────────┘      └──────────────────┘      └──────────────┘
```
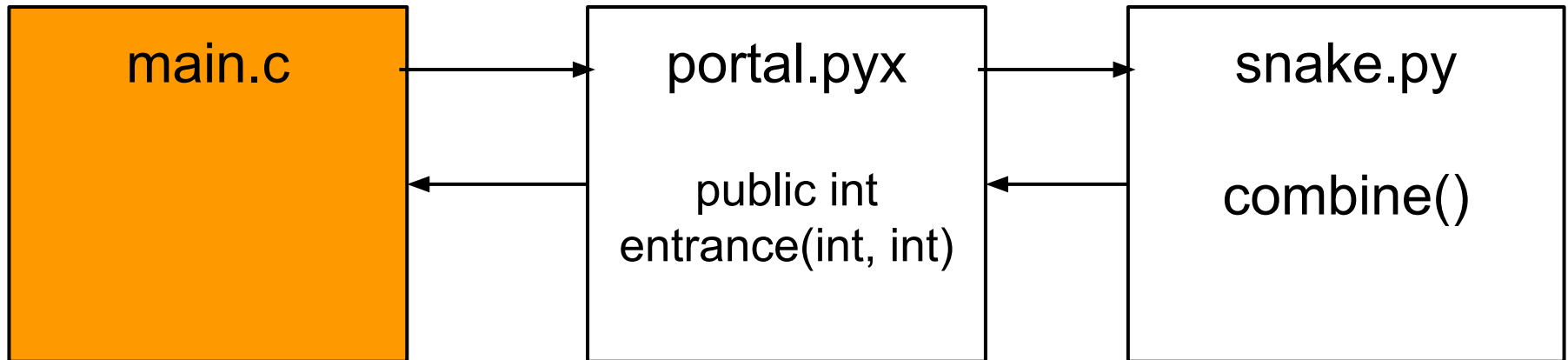
```c
/* main.c: */  /* NB: error handling removed */
#include <Python.h>
#include "portal.h"
int main(int argc, char ** argv) {
    int result;
    printf("hello from C\n");
    PyImport_AppendInittab("portal",
                    PyInit_portal);
    Py_Initialize();
```

```
main.c    →    portal.pyx    →    snake.py
          ←                  ←
               public int         combine()
               entrance(int, int)
```
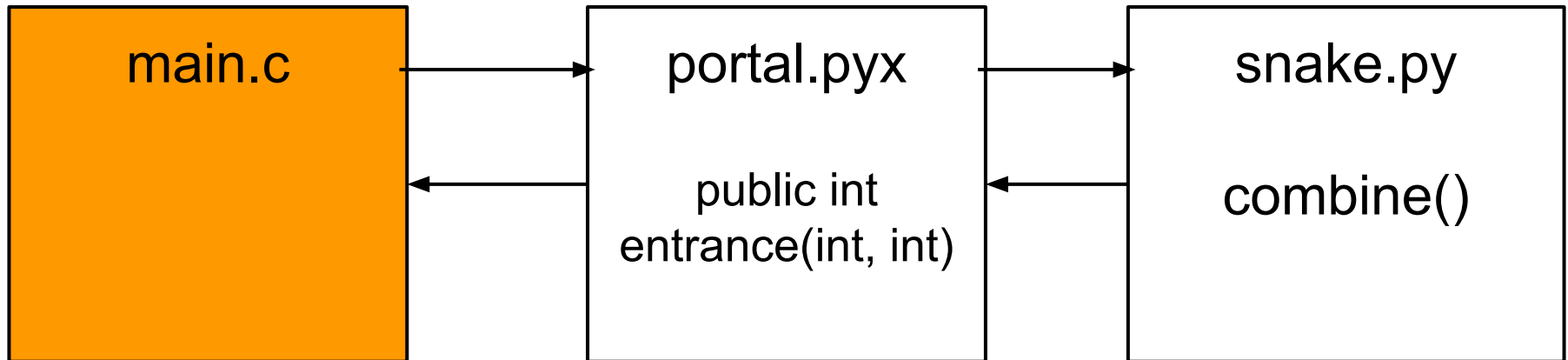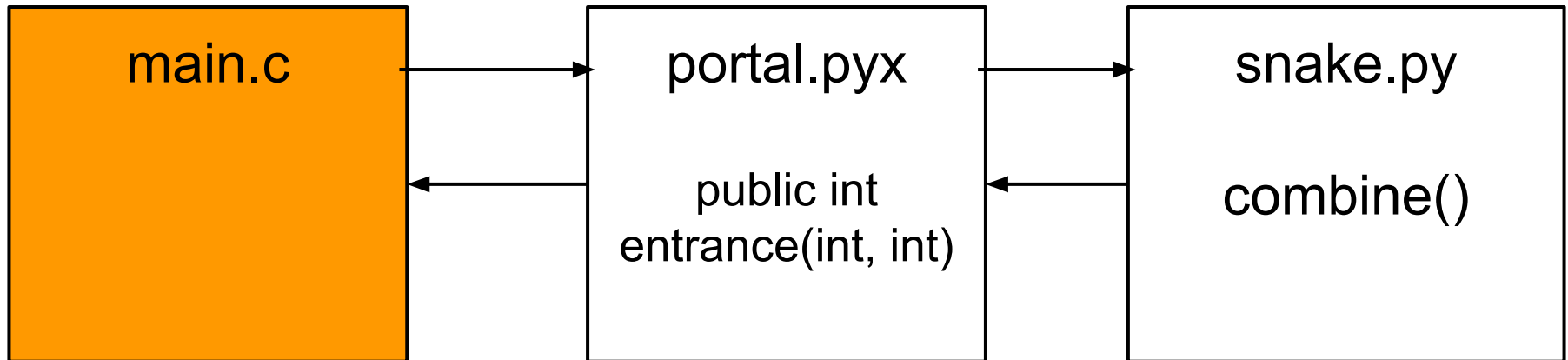
```c
/* main.c: */   /* NB: error handling removed */
#include <Python.h>
#include "portal.h"
int main(int argc, char ** argv) {
    int result;
    printf("hello from C\n");
    PyImport_AppendInittab("portal",
                    PyInit_portal);
```

```
┌─────────────┐      ┌─────────────┐      ┌─────────────┐
│             │ ───▶ │  portal.pyx │ ───▶ │  snake.py   │
│   main.c    │      │             │      │             │
│             │      │  public int │      │  combine()  │
│             │ ◀─── │ entrance(int, int) │ ◀─── │             │
└─────────────┘      └─────────────┘      └─────────────┘
```
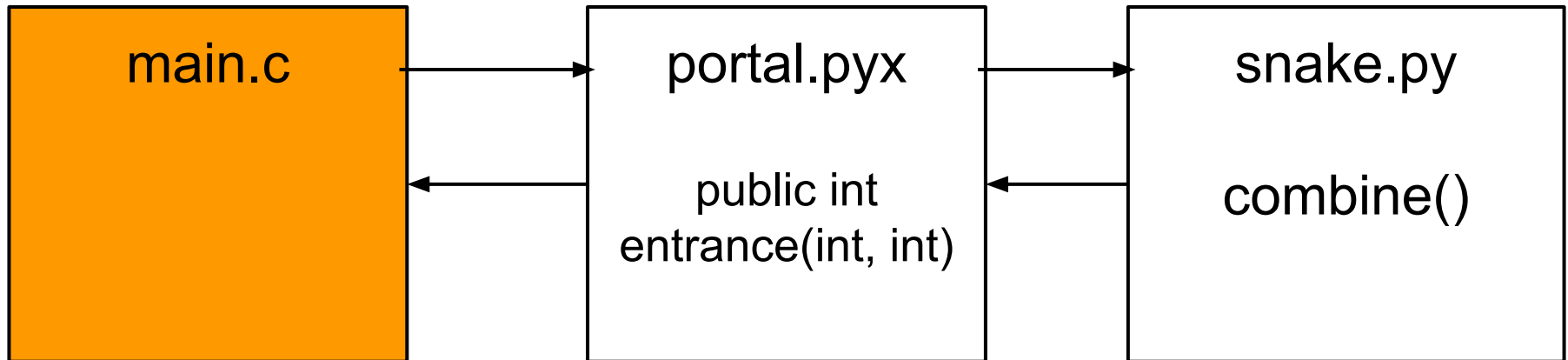
```c
#include <Python.h>
#include "portal.h"
int main(int argc, char ** argv) {
    int result;
    printf("hello from C\n");
    PyImport_AppendInittab("portal",
                    PyInit_portal);
    Py_Initialize();
    PyImport_ImportModule("portal");
```

```
main.c          →          portal.pyx          →          snake.py

                           public int                      combine()
                ←          entrance(int, int)  ←
```

```c
#include "portal.h"
int main(int argc, char ** argv) {
    int result;
    printf("hello from C\n");
    PyImport_AppendInittab("portal",
                           PyInit_portal);
    Py_Initialize();
    PyImport_ImportModule("portal");
    result = entrance(1, 2);   /* defined in
```

main.c	portal.pyx	snake.py
	public int entrance(int, int)	combine()

```
int main(int argc, char ** argv) {
    int result;
    printf("hello from C\n");
    PyImport_AppendInittab("portal",
                        PyInit_portal);
    Py_Initialize();
    PyImport_ImportModule("portal");
    result = entrance(1, 2);  /* portal.h */
    printf("Result is %d\n", result);
```
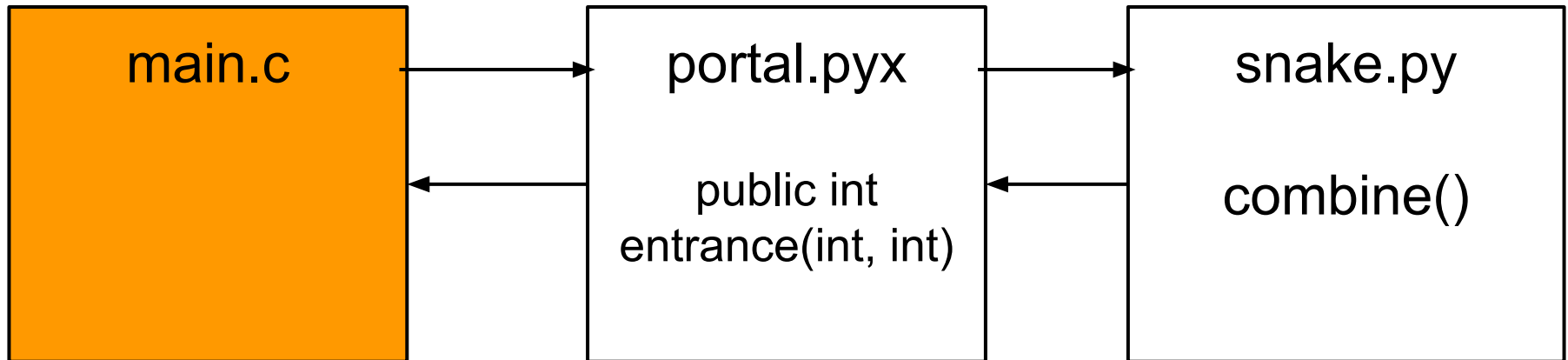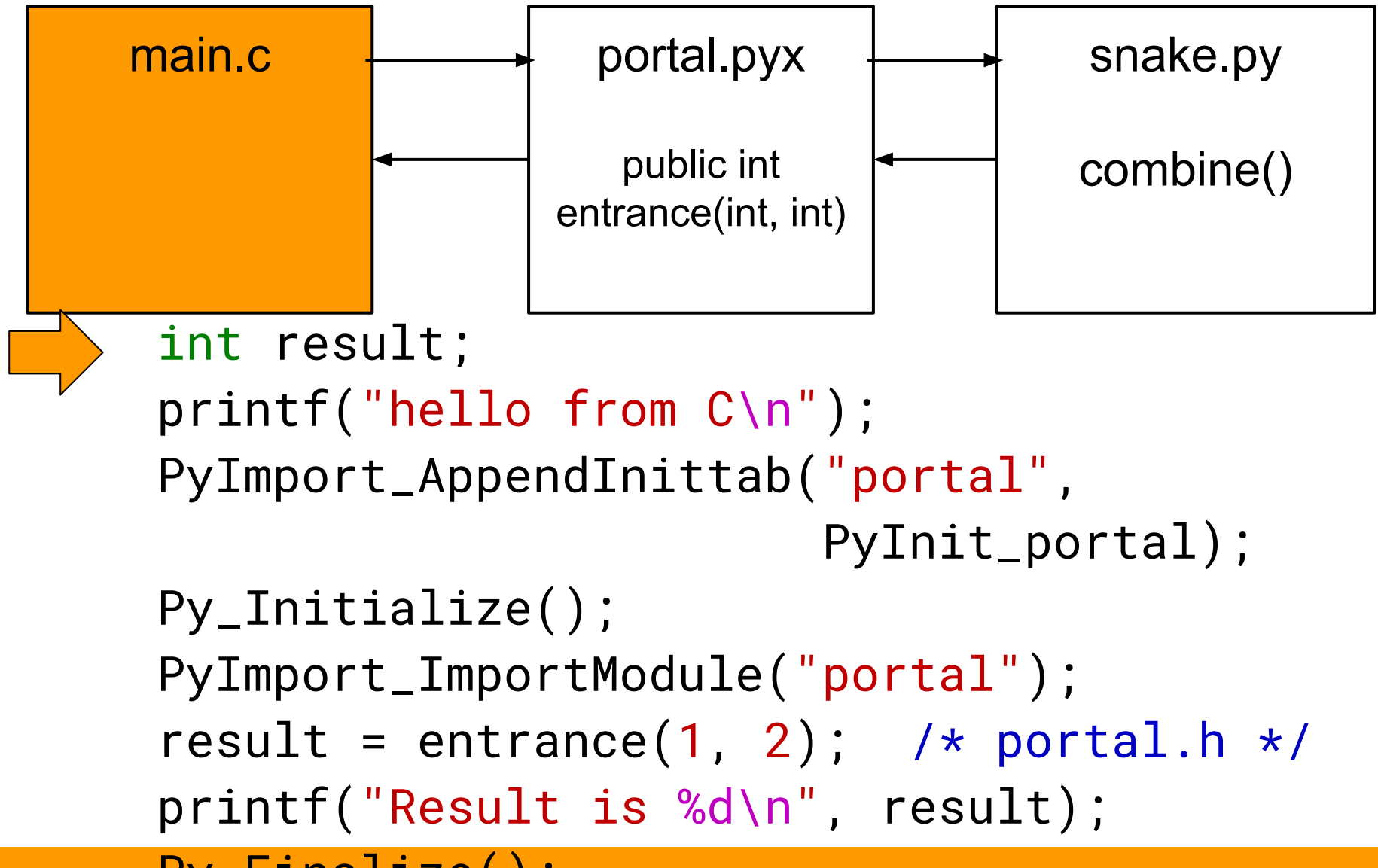
main.c	portal.pyx public int entrance(int, int)	snake.py combine()

```
int result;
printf("hello from C\n");
PyImport_AppendInittab("portal",
                    PyInit_portal);
Py_Initialize();
PyImport_ImportModule("portal");
result = entrance(1, 2);  /* portal.h */
printf("Result is %d\n", result);
Py_Finalize();
```

```
main.c          portal.pyx          snake.py

                public int          combine()
                entrance(int, int)
```

```c
printf("hello from C\n");
PyImport_AppendInittab("portal",
                    PyInit_portal);
Py_Initialize();
PyImport_ImportModule("portal");
result = entrance(1, 2);   /* portal.h */
printf("Result is %d\n", result);
Py_Finalize();
}
```
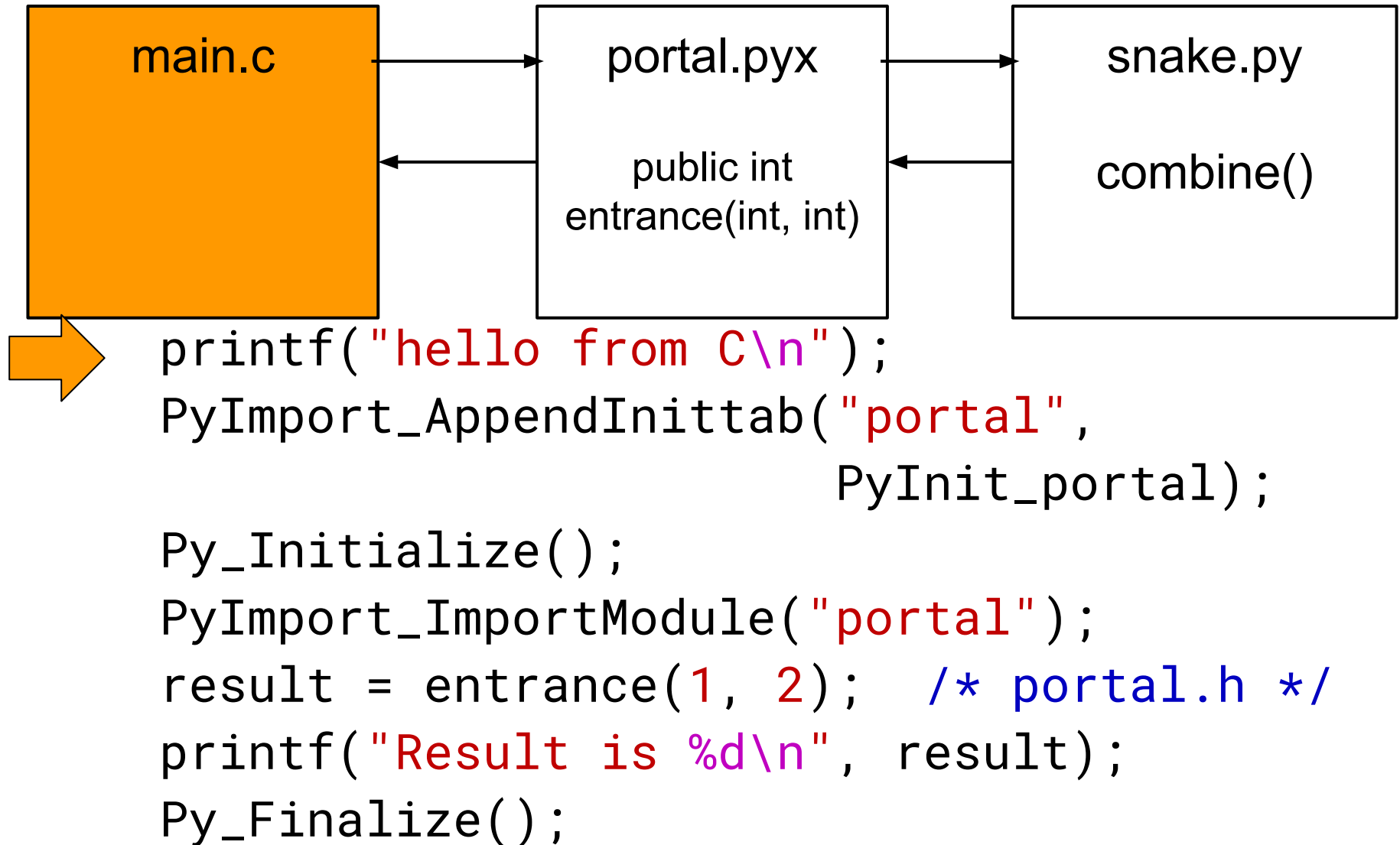
```
+-----------+        +-------------------+        +-----------+
|           |   -->  |    portal.pyx     |   -->  | snake.py  |
|  main.c   |        |                   |        |           |
|           |   <--  |    public int     |   <--  | combine() |
|           |        | entrance(int, int)|        |           |
+-----------+        +-------------------+        +-----------+
```

```
PyImport_AppendInittab("portal",
                          PyInit_portal);
Py_Initialize();
PyImport_ImportModule("portal");
result = entrance(1, 2);   /* portal.h */
printf("Result is %d\n", result);
Py_Finalize();
}
```
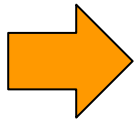
```
┌──────────────┐     ┌──────────────────┐     ┌──────────────────┐
│              │ ──► │   portal.pyx     │ ──► │   snake.py       │
│   main.c     │     │                  │     │                  │
│              │     │   public int     │     │   combine()      │
│              │ ◄── │ entrance(int, int)│ ◄── │                  │
└──────────────┘     └──────────────────┘     └──────────────────┘
```
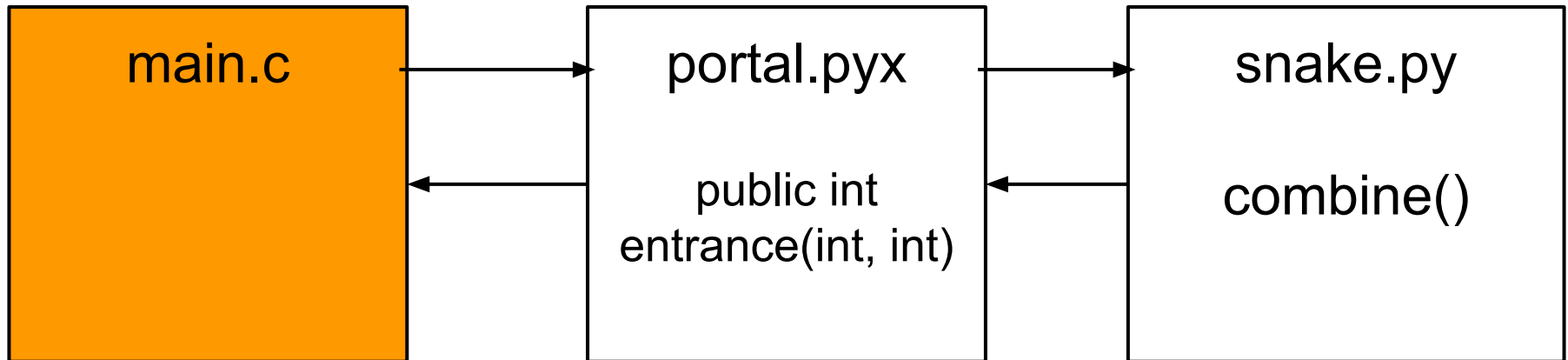
```c
PyImport_AppendInittab("portal",
                       PyInit_portal);
Py_Initialize();
PyImport_ImportModule("portal");
result = entrance(1, 2);   /* portal.h */
printf("Result is %d\n", result);
Py_Finalize();
}
```
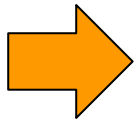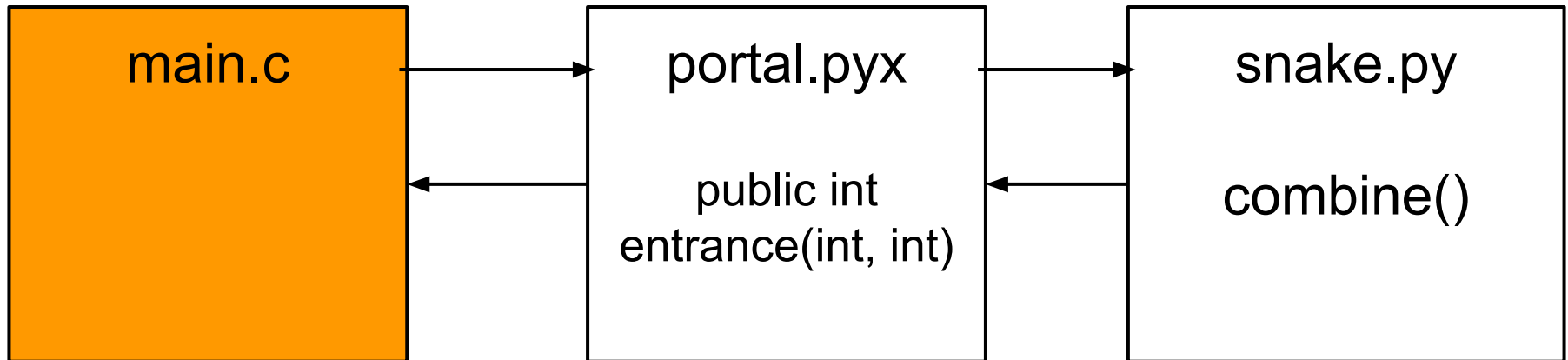
```
┌─────────────┐      ┌─────────────────┐      ┌─────────────┐
│             │ ───► │   portal.pyx    │ ───► │  snake.py   │
│   main.c    │      │                 │      │             │
│             │ ◄─── │   public int    │ ◄─── │  combine()  │
│             │      │ entrance(int, int)│    │             │
└─────────────┘      └─────────────────┘      └─────────────┘
```

```
PyImport_AppendInittab("portal",
                       PyInit_portal);
Py_Initialize();
PyImport_ImportModule("portal");
result = entrance(1, 2);  /* portal.h */
printf("Result is %d\n", result);
Py_Finalize();
}
```

```
┌─────────────┐        ┌───────────────┐       ┌──────────────┐
│             │───────▶│   portal.pyx  │──────▶│   snake.py   │
│   main.c    │        │               │       │              │
│             │◀───────│   public int  │◀──────│   combine()  │
│             │        │entrance(int, int)│    │              │
└─────────────┘        └───────────────┘       └──────────────┘
```

```
PyImport_AppendInittab("portal",
                       PyInit_portal);
Py_Initialize();
PyImport_ImportModule("portal");
result = entrance(1, 2);  /* portal.h */
printf("Result is %d\n", result);
Py_Finalize();
}
```

2. Call Python from C
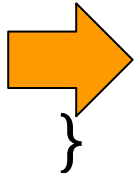
```
$ ./configure

$ make

$ ./cythonwalkthrough

hello from C
hello from Cython
hello from Python
Result is 81
```

Uses for Cython
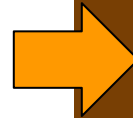
1. Faster Python code
2. Call Python from C
3. Call C from Python

But why?

- C files included in your Python project
- External libraries
- Anything that provides C/C++ header files

Example:

- Let's draw a picture with libgd ("graphics draw")
- https://libgd.github.io/

```python
# setup.py:

from setuptools import setup, Extension
from Cython.Build import cythonize

setup(
    # …
    ext_modules=cythonize(
        Extension('*', ['pygd/*.pyx'],
                  libraries=['gd'])),
)
```

```
# pygd/gd.pyx:

cdef extern from "gd.h":
```

- Definitions so Cython can generate correct C code
- Should match (reasonably) with actual C headers

```
# pygd/gd.pyx:

cdef extern from "gd.h":
    ctypedef struct gdImage
    ctypedef struct gdPoint:
        int x
        int y
```

```
# pygd/gd.pyx:

cdef extern from "gd.h":
    ctypedef struct gdImage
    ctypedef struct gdPoint:
        int x
        int y
```
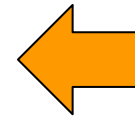
Not interested
in contents

```
# pygd/gd.pyx:

cdef extern from "gd.h":
    ctypedef struct gdImage
    ctypedef struct gdPoint:
        int x
        int y
```
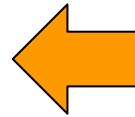
⬅ Only need to include members we use

```
# pygd/gd.pyx:

cdef extern from "gd.h":
    ctypedef struct gdImage
    ctypedef struct gdPoint:
        int x
        int y
    cdef gdImage * gdImageCreate(
        int sx, int sy)
    cdef void gdImageDestroy(
        gdImage * im)
    cdef void gdImageFilledPolygon(
        gdImage * im, gdPoint *p, int n, int c)
    # …
```

```
# pygd/gd.pyx:

cdef extern from "gd.h":
    ctypedef struct gdImage
    ctypedef struct gdPoint:
        int x
        int y
    cdef gdImage * gdImageCreate(
        int sx, int sy)
    cdef void gdImageDestroy(
        gdImage * im)
    cdef void gdImageFilledPolygon(
        gdImage * im, gdPoint *p, int n, int c)
    # …
```

```
# pygd/gd.pyx:

cdef extern from "gd.h":
    ctypedef struct gdImage
    ctypedef struct gdPoint:
        int x
        int y
    cdef gdImage * gdImageCreate(
        int sx, int sy)
    cdef void gdImageDestroy(
        gdImage * self?)
    cdef void gdImageFilledPolygon(
        gdImage * self?, gdPoint *p, int n, int
    # …
```

```
# pygd/gd.pyx:

cdef class Image:
    cdef gdImage * cobj
```

```python
# pygd/gd.pyx:

cdef class Image:
    cdef gdImage * cobj

    def __cinit__(self,
                  int size_x, int size_y,
                  bgcolor=(255, 255, 255)):
        self.cobj = gdImageCreate(size_x,
                                  size_y)
        if self.cobj is NULL:
            raise MemoryError(
                'Unable to create gdImage')
```

```python
# pygd/gd.pyx:

cdef class Image:
    cdef gdImage * cobj

    def __cinit__(self, int size_x, …):
        # …

    def __dealloc__(self):
        if self.cobj:
            gdImageDestroy(self.cobj)
```

```
# pygd/gd.pyx:

cdef class Image:
    cdef gdImage * cobj

    def __cinit__(self, int size_x, …):
        # …

    def __dealloc__(self):
        # …

    def filled_polygon(self,
                       *points,
                       color=(0, 0, 0)):
        # points: eg [(1,2), (4,6), (9,3)]
```

```
def filled_polygon(self, *points, color=…):
    cdef gdPoint * cpoints = <gdPoint *>calloc(
        len(points), sizeof(gdPoint))
```

```python
def filled_polygon(self, *points, color=…):
    cdef gdPoint * cpoints = <gdPoint *>calloc(
        len(points), sizeof(gdPoint))
    if cpoints is NULL:
        raise MemoryError()
```

```python
def filled_polygon(self, *points, color=…):
    cdef gdPoint * cpoints = <gdPoint *>calloc(
        len(points), sizeof(gdPoint))
    if cpoints is NULL:
        raise MemoryError()
    try:



    finally:
        free(cpoints)
```

```python
def filled_polygon(self, *points, color=…):
    cdef gdPoint * cpoints = <gdPoint *>calloc(
        len(points), sizeof(gdPoint))
    if cpoints is NULL:
        raise MemoryError()
    try:
        for i, point in enumerate(points):
            cpoints[i].x = point[0]
            cpoints[i].y = point[1]


    finally:
        free(cpoints)
```

```python
def filled_polygon(self, *points, color=…):
    cdef gdPoint * cpoints = <gdPoint *>calloc(
        len(points), sizeof(gdPoint))
    if cpoints is NULL:
        raise MemoryError()
    try:
        for i, point in enumerate(points):
            cpoints[i].x = point[0]
            cpoints[i].y = point[1]
        gdImageFilledPolygon(
            self.cobj, cpoints, len(points), …)
    finally:
        free(cpoints)
```

```python
# example.py:

import pygd


img = pygd.Image(150, 150, bgcolor=(40, 65, 90))
img.filled_polygon(*s_blue, color=(70, 135, 185))
img.filled_polygon(*s_yellow, color=(255, 215, 65))
img.filled_polygon(*eye_1, color=(255, 255, 255))
img.filled_polygon(*eye_2, color=(255, 255, 255))

with open('example.png', 'wb') as f:
    img.dump(f)
```

```python
# example.py:

import pygd


img = pygd.Image(150, 150, bgcol
img.filled_polygon(*s_blue, col
img.filled_polygon(*s_yellow, c                    )
img.filled_polygon(*eye_1, colo
img.filled_polygon(*eye_2, colo


with open('example.png', 'wb') a
    img.dump(f)
```

Final thoughts

- Mix Python and C
- Alternatives
  - PyPy
  - cffi
  - ctypes

Code and slides:

- github.com/flexo/cythonwalkthrough

Questions:

- Ask me!
- Here until Tuesday

Thank you

Nick Murdoch
@nickmurdoch