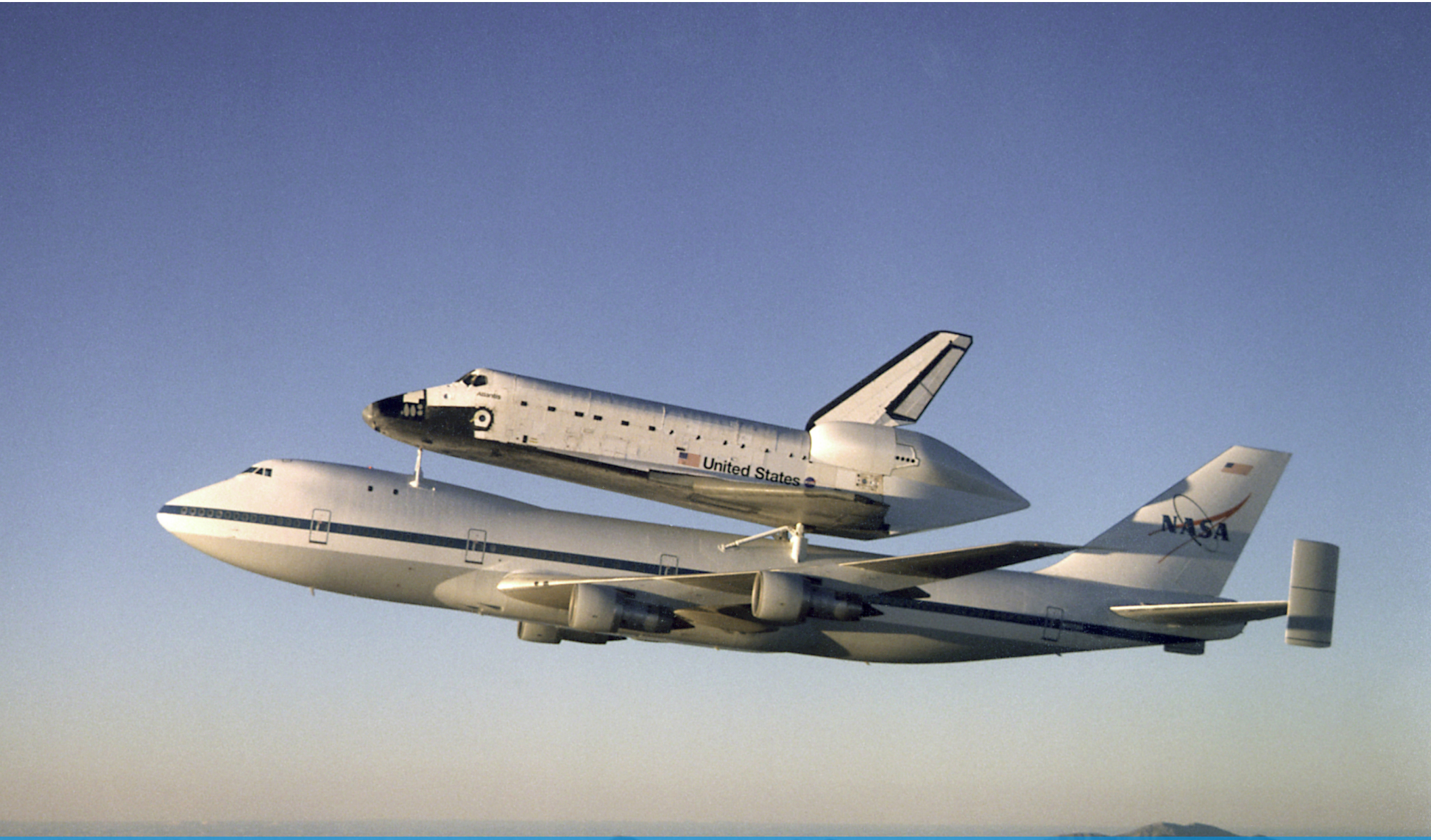


Foreign Function Interfaces for Web Developers

Tristan Fisher - August 2015

Slides and code: github.com/tristanfisher/ffi4wd

a.k.a. Language interoperability for coders with impatient users.

About me

/tristanfisher

/users/559633

tristanfisher.com

* Been using Python for roughly 3 years
* Employed by SinglePlatform (one of this year's Gold Sponsors)

This talk is focused on using programming language interoperability to make web applications faster.

(Generalizations made; examples not exhaustive)

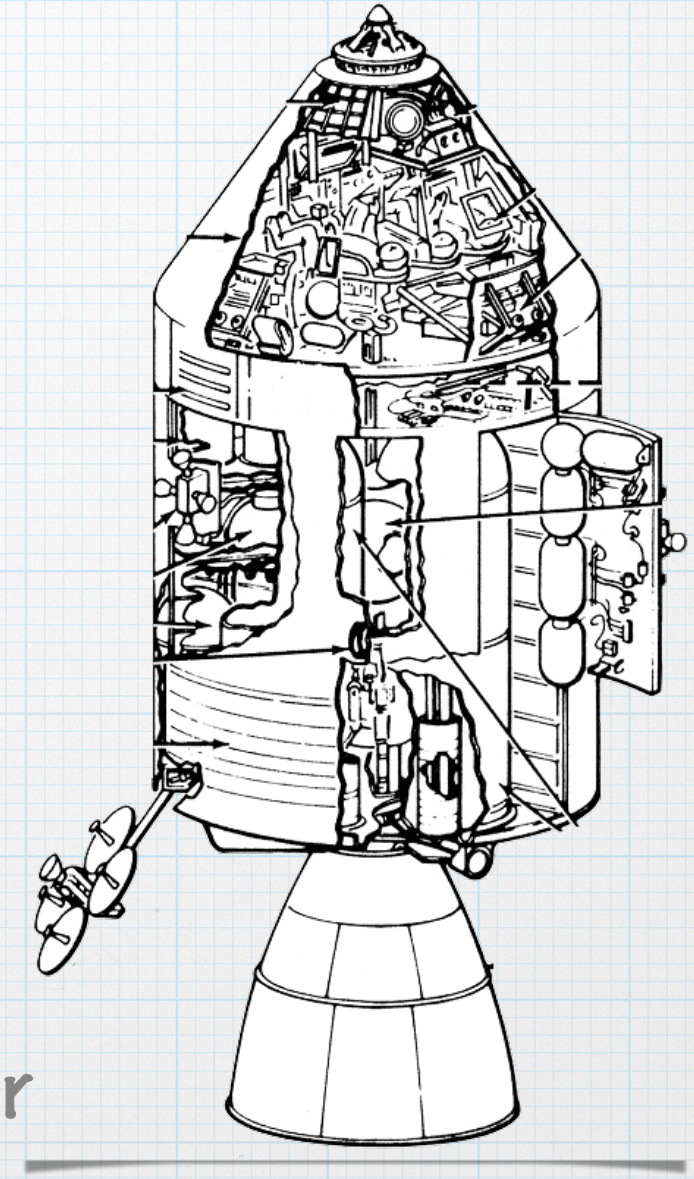
Target audience

* General understanding of Python
* Some Web Development experience
* Linux or systems experience beneficial

Effort was made to be approachable for people newer to these topics.

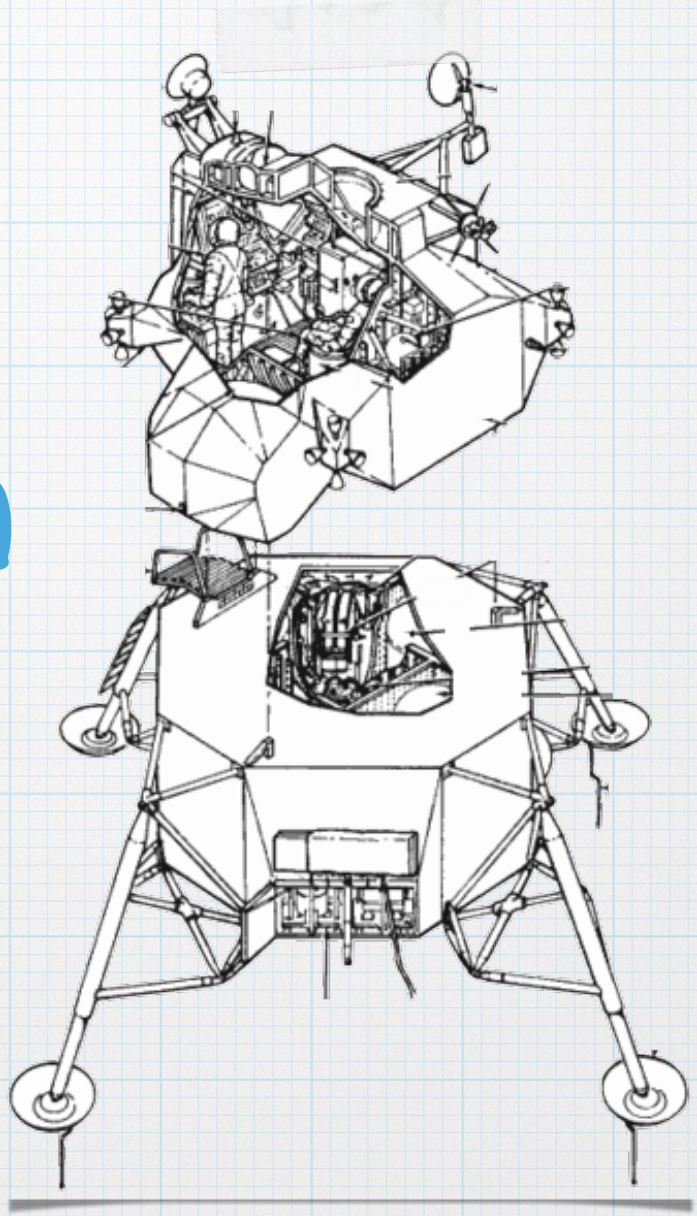
Talk objectives

* Remind you of the real-world constraints your code faces
* Show how abstractions affect us
* Introduce ways of making your Python application code run faster



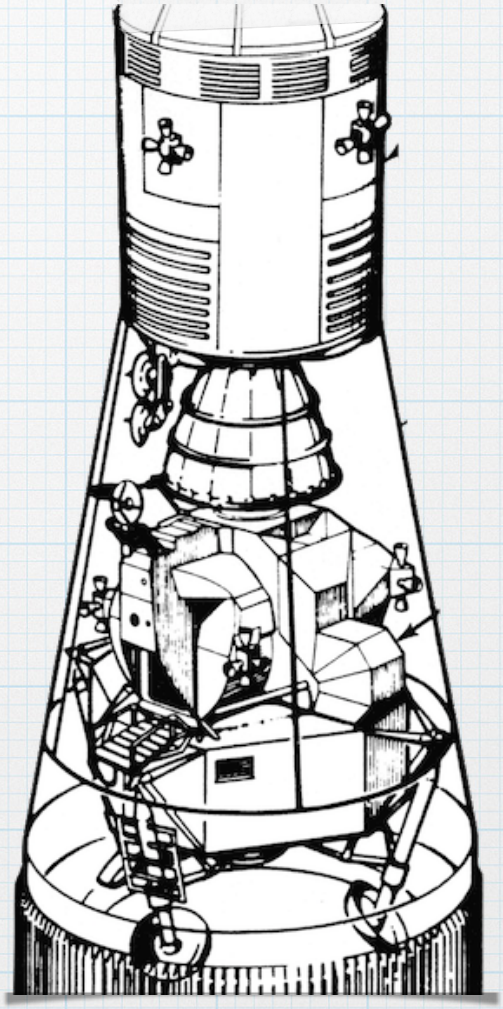
Plan of approach

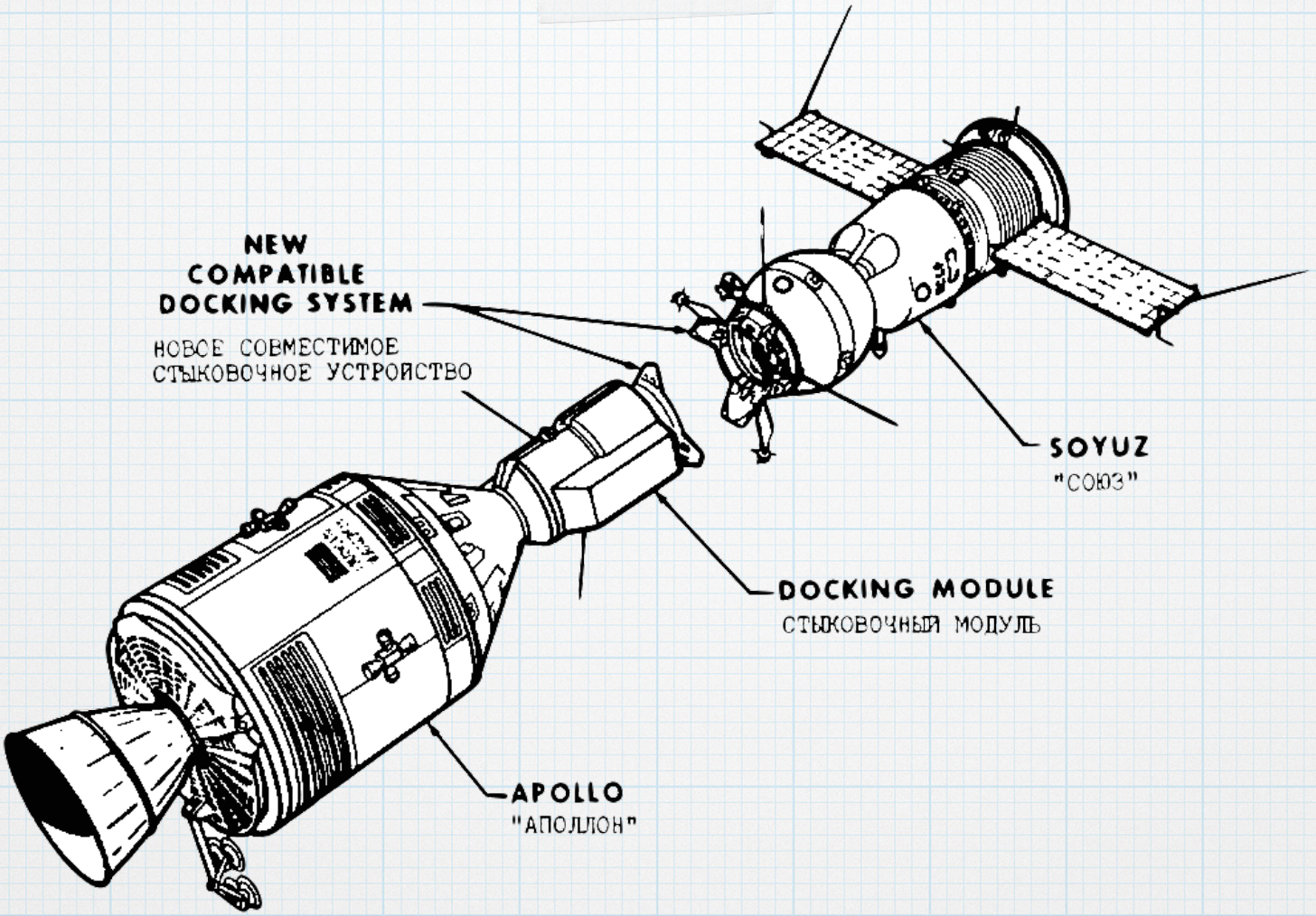
* Shared-vocabulary and background
* Overview of Python code at runtime
* Introduce a real-world example
* Talk about Python and Foreign Function Interfaces (FFI)
* Topic review and close with some notes



This talk and its source code is online

Grab the FFI playground after the talk:
github.com/tristanfisher/ffi4wd





Establishing a shared vocabulary and background

Talk environment

* CPython[*]: The Python language implemented in the C programming language. We'll use version 3.4.3.
* Flask: A web application framework
* Virtual Machine (VM): A software emulation of the critical details of the underlying computer hardware

[*] Most popular, default implementation (also the reference specification)

In this talk, I may use "Python" meaning "CPython version 3.4.3."

Expressing effort

* Work: Effort (use of energy) that returns a result
* Efficiency: The ratio of results from work in relation to effort expended (low to high)
* Latency: The time between when a request for work is made and when result is perceived

Expressing effort

* Deadline: Expectation of work finished by a given time
* Consequence: Result of a deadline being met or missed

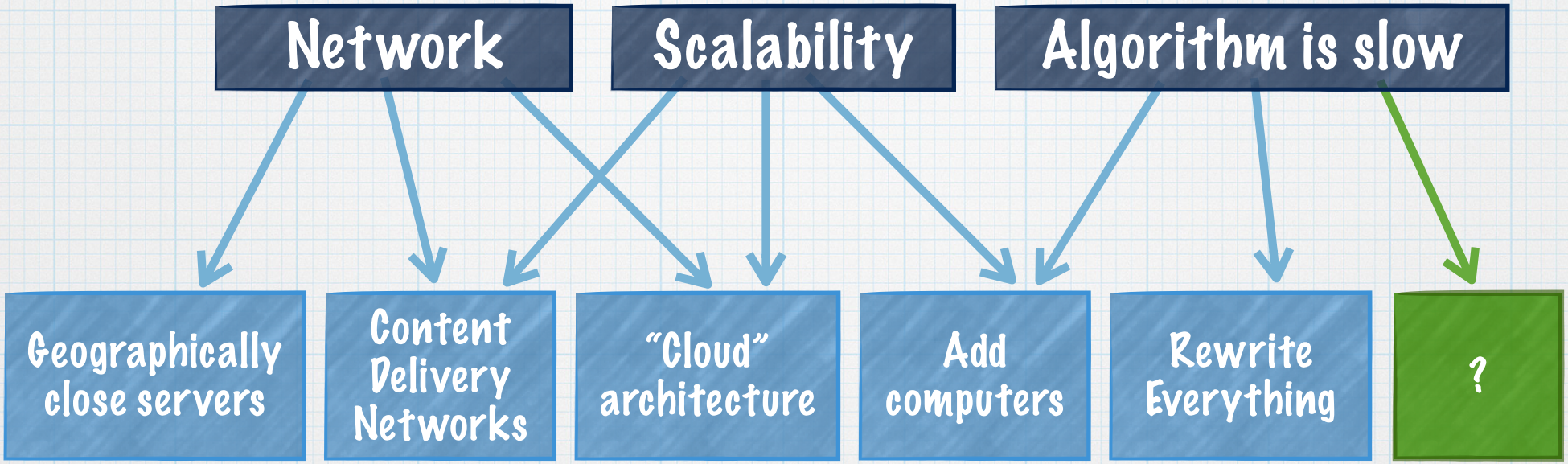
Web Application users start losing context after 1 second of latency and give up after 10 seconds[11].

User request deadlines

Deadline Seconds	Deadline Type	Consequence If Met	Consequence If Failed
0 <= 1	Soft	User happy	User loses focus
> 1 <= 10	Hard	User placated	User is gone. Work done is wasted.

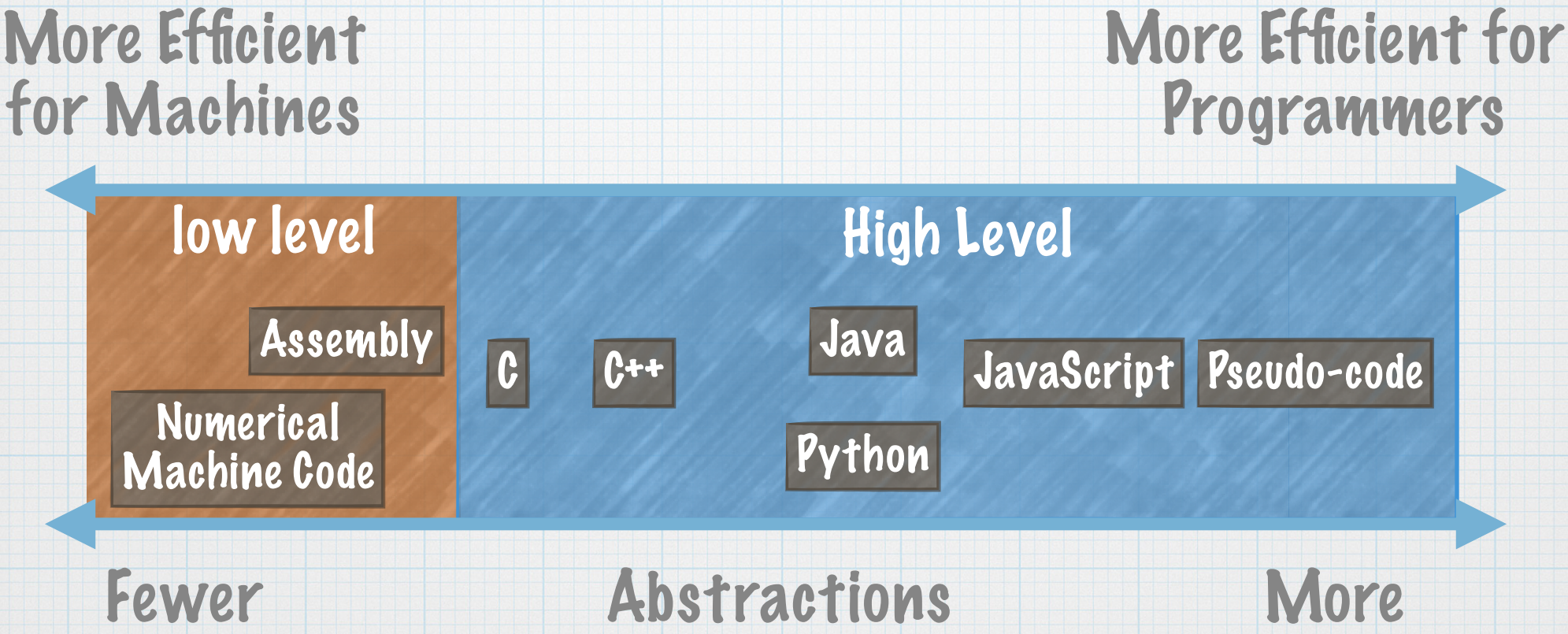
Your application not allowed to take more than 10 seconds for 'real-time' actions.

Overcoming bottlenecks we can control



Maybe we can make the same algorithm more efficient for the computer?

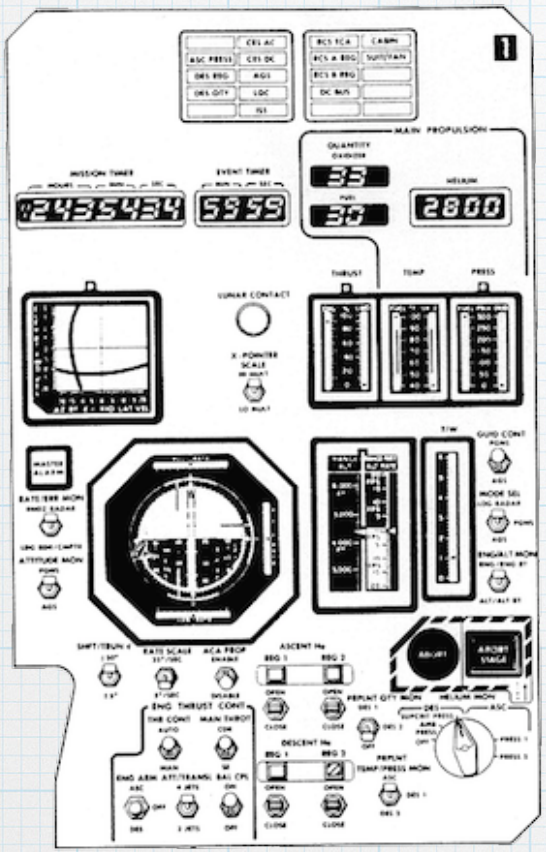
Language "levels"

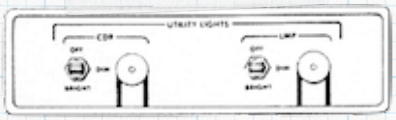


Efficiency: "The ratio of results from work in relation to effort expended (low to high)"

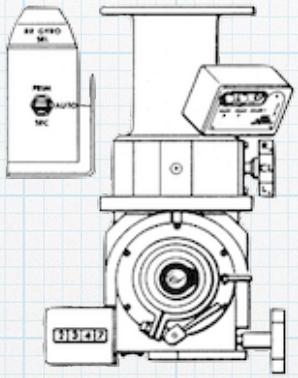
Python, C, C++ are all high-level languages

Computers require code in lower-level languages to operate

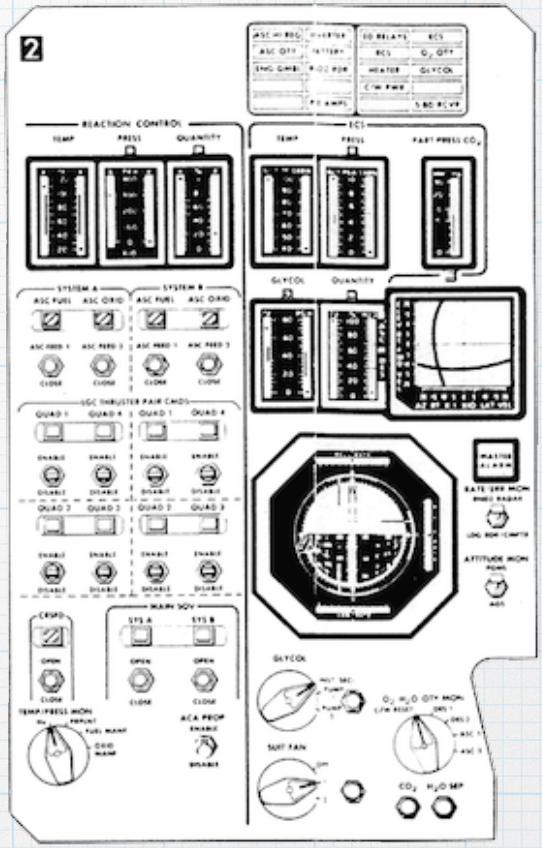




UTILITY LIGHT SWITCH ASSEMBLY



ALIGNMENT OPTICAL TELESCOPE (AOT)



Overview of High Level Languages at runtime

Python compiles its code to run on a "Virtual Machine" (VM)

Our Python code

```
>>> hello = """
... print("Hello, world")
... """
```

Compiler[*]

```
>>> code_object = compile(hello, "<string>", "eval")
```

CPython bytecode

```
>>> code_object.co_code
b 'e\x00\x00d\x00\x00\x83\x01\x00S'
```

Python Virtual Machine

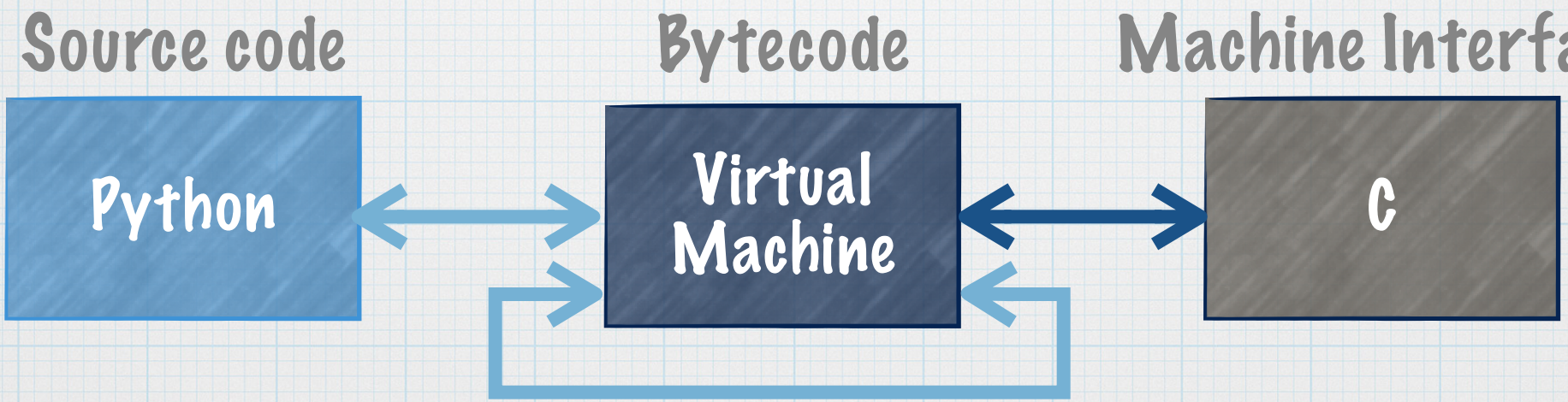
[*] Bytecode compilation implementation here: https://hg.python.org/cpython/file/3.4/Python/ceval.c

Python bytecode

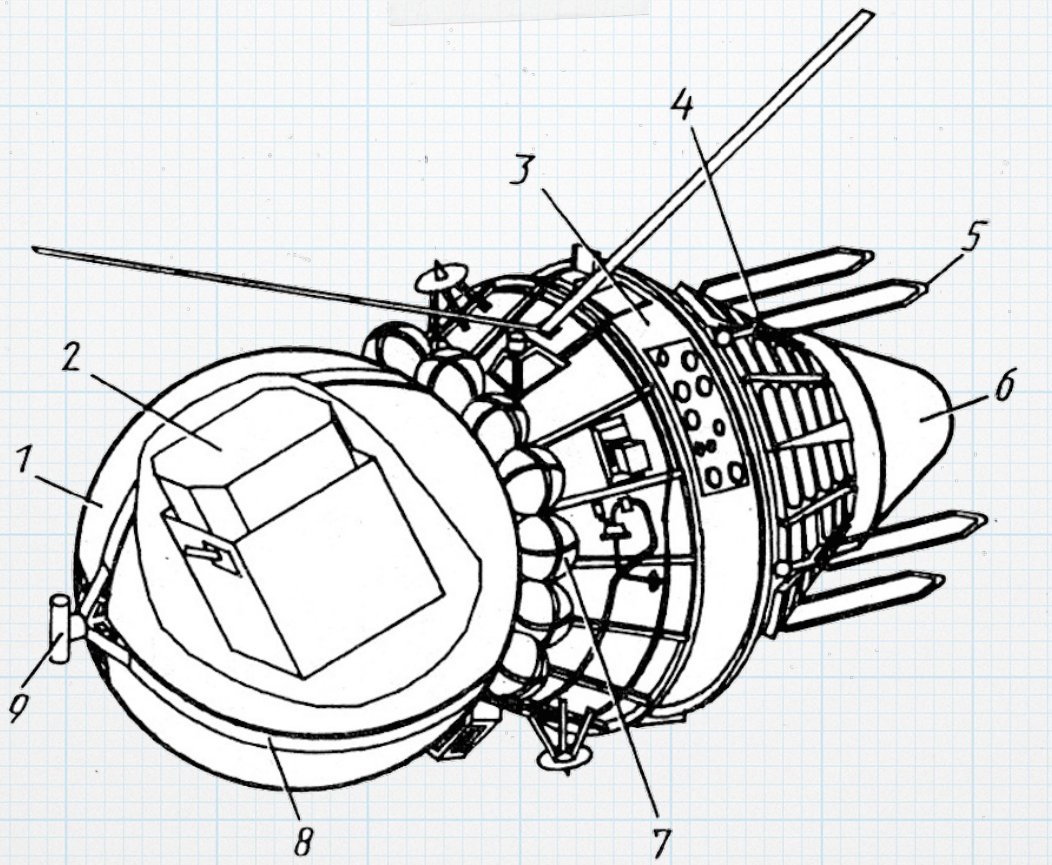
* Python code is converted to bytecode when the interpreter reads source files (e.g. `python app.py`)
* Conversion to bytecode makes portability between platforms easier — software emulation handles hardware interaction

Python bytecode

* Python always compiles to bytecode.
* Makes a "Virtual Machine" abstraction more efficient
* Where Global Interpreter Lock and threading happens



Important details of Bytecode/C interaction here: https://hg.python.org/cpython/file/34/Python/compile.c#l104



Solving an example problem

Efficiently visit these cities and return home

new york, ny, usa

portland, me, usa

seattle, wa, usa

san fransciso, ca, usa

denver, co, usa

austin, tx, usa

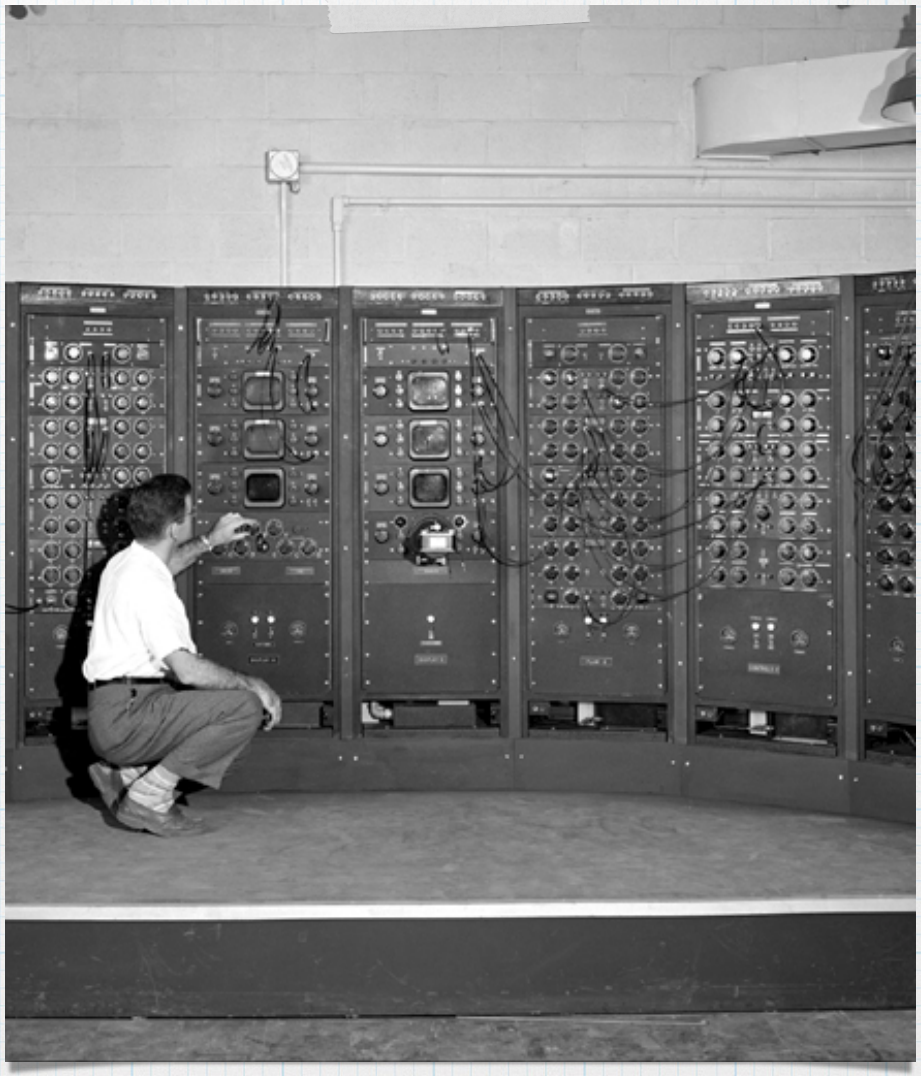
montreal, qc, ca

vancouver, bc, ca

anchorage, ak, usa

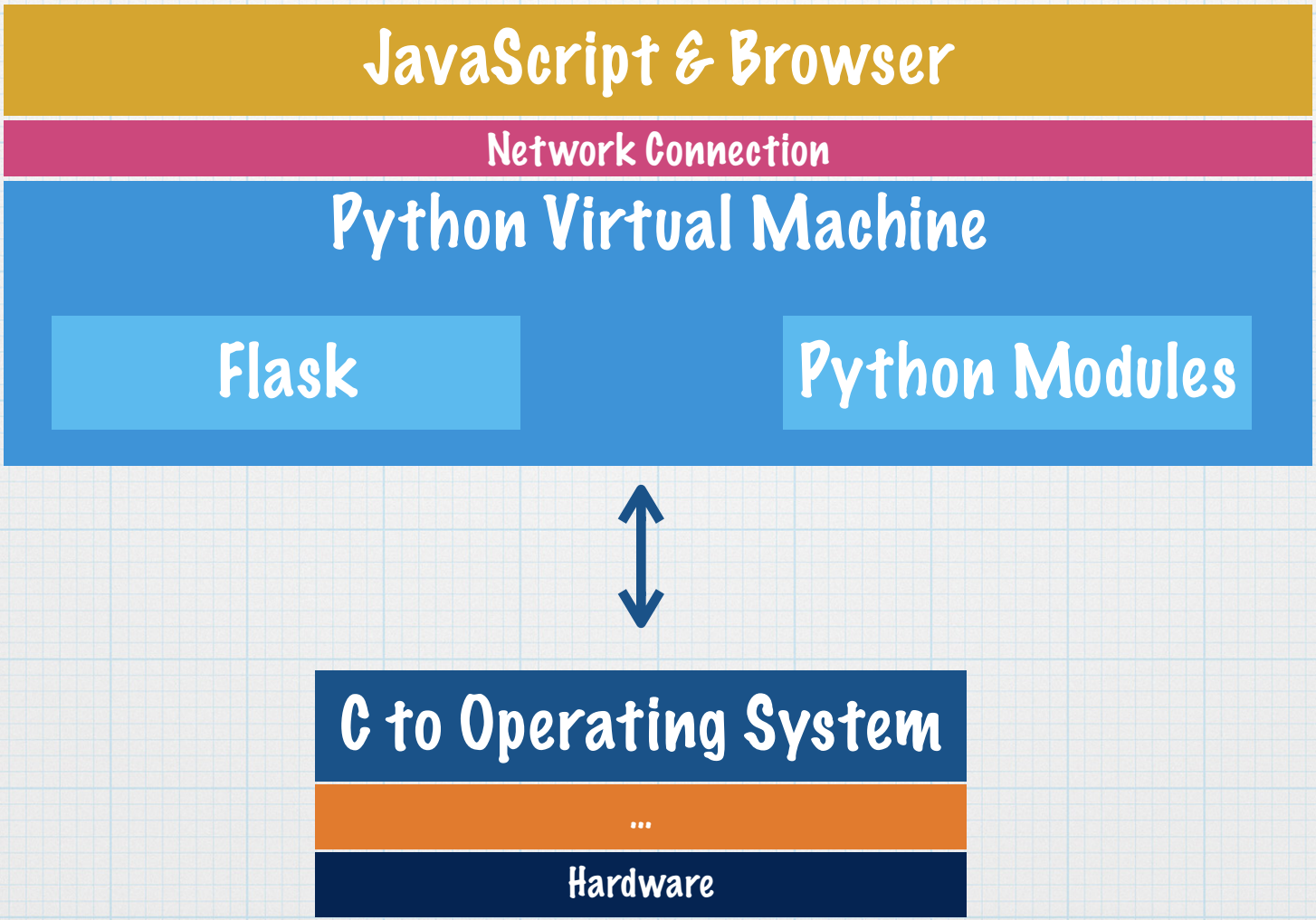
toronto, on, ca

berlin, germany



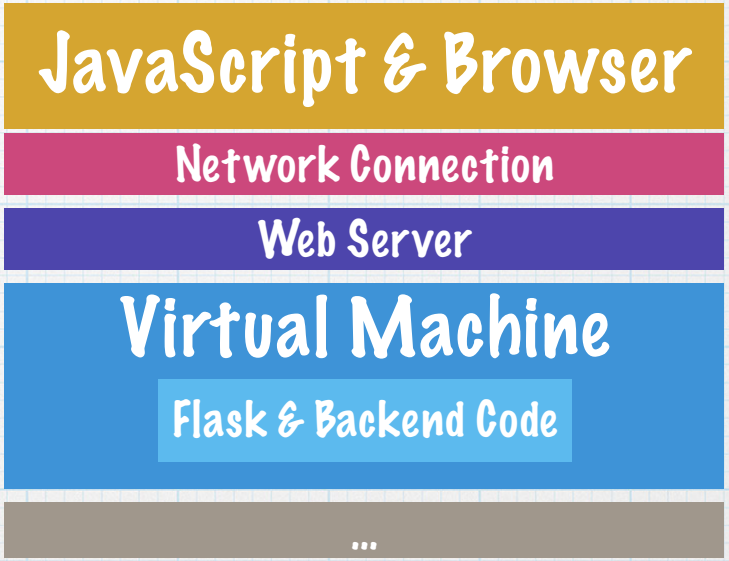
Let's make it a web service.

Our Python web stack



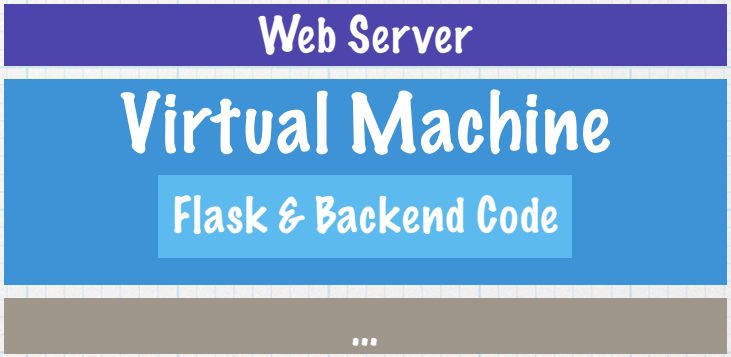
Connection overview

* User browser talks to the web server
* Web Server talks to Flask
* Flask calls backend code to return to the web server
* Web server sends content to user browser
* User browser decodes and renders content, runs JavaScript code
* This feedback loop repeats

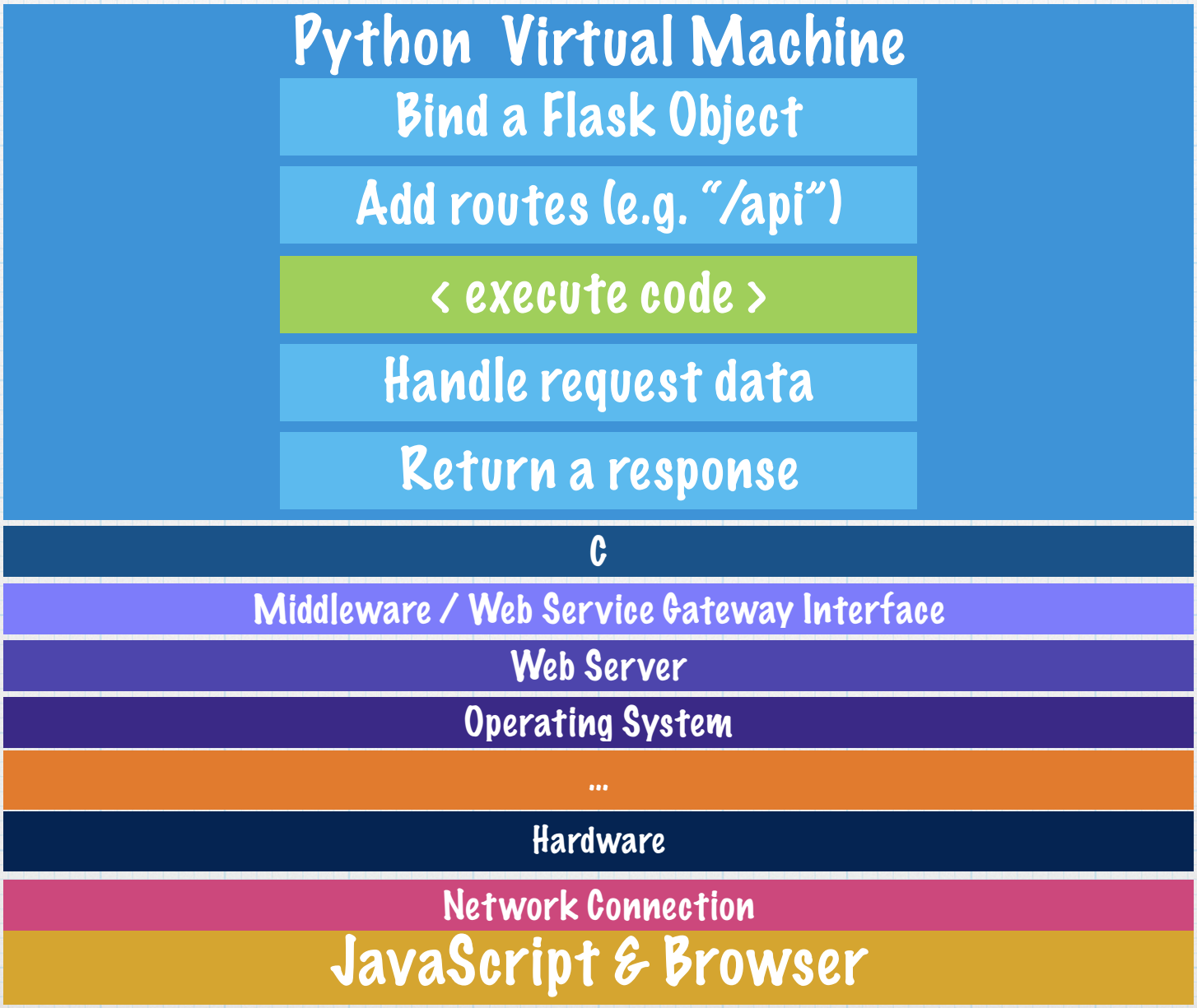


Flask and backend code

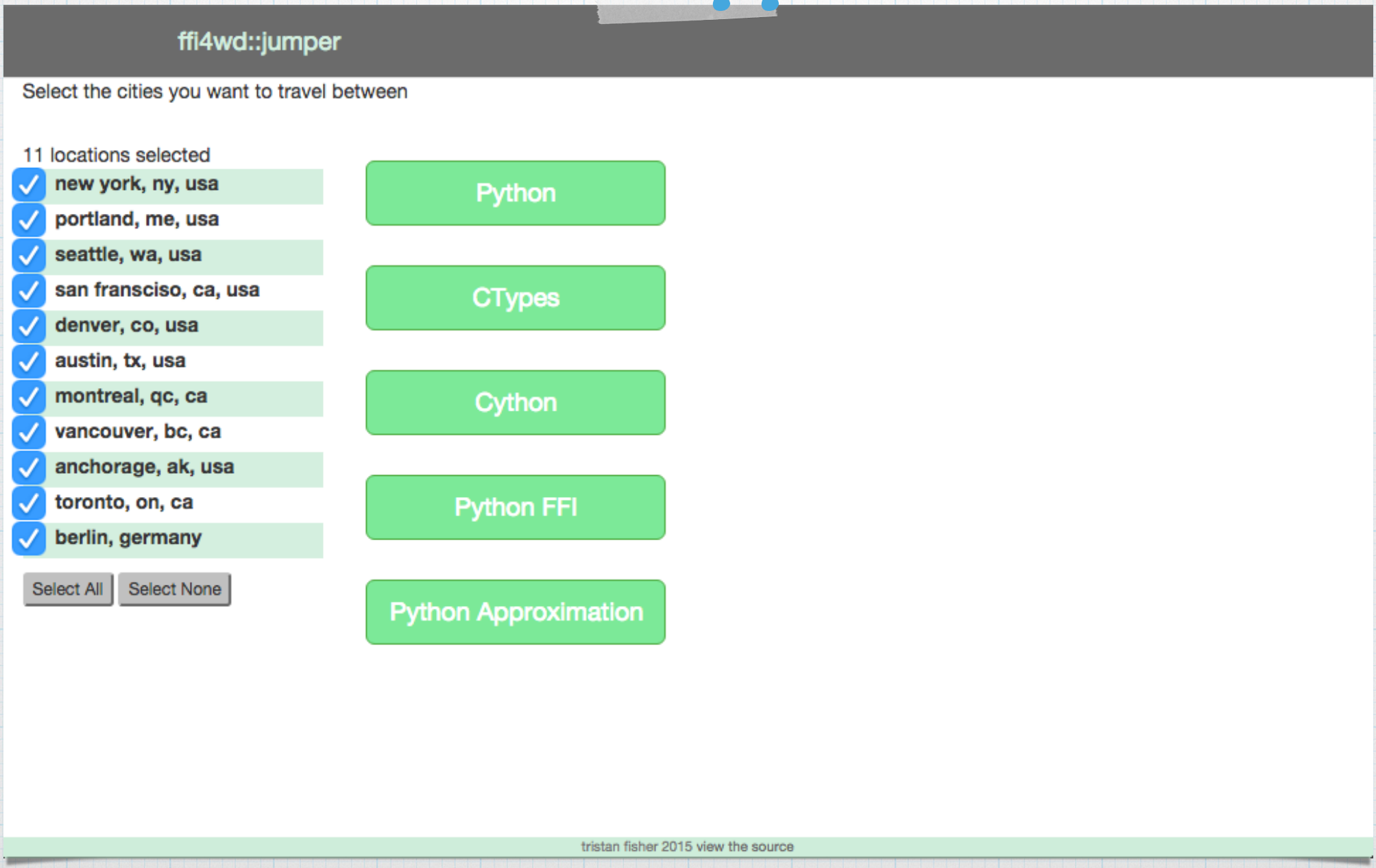
* Flask gets/sends data from its interpreter/service gateway
* Flask talks to the Python VM
* VM calls code that may not be aware of computer networks
* VM calls on compiled code, abstractions continue to diminish



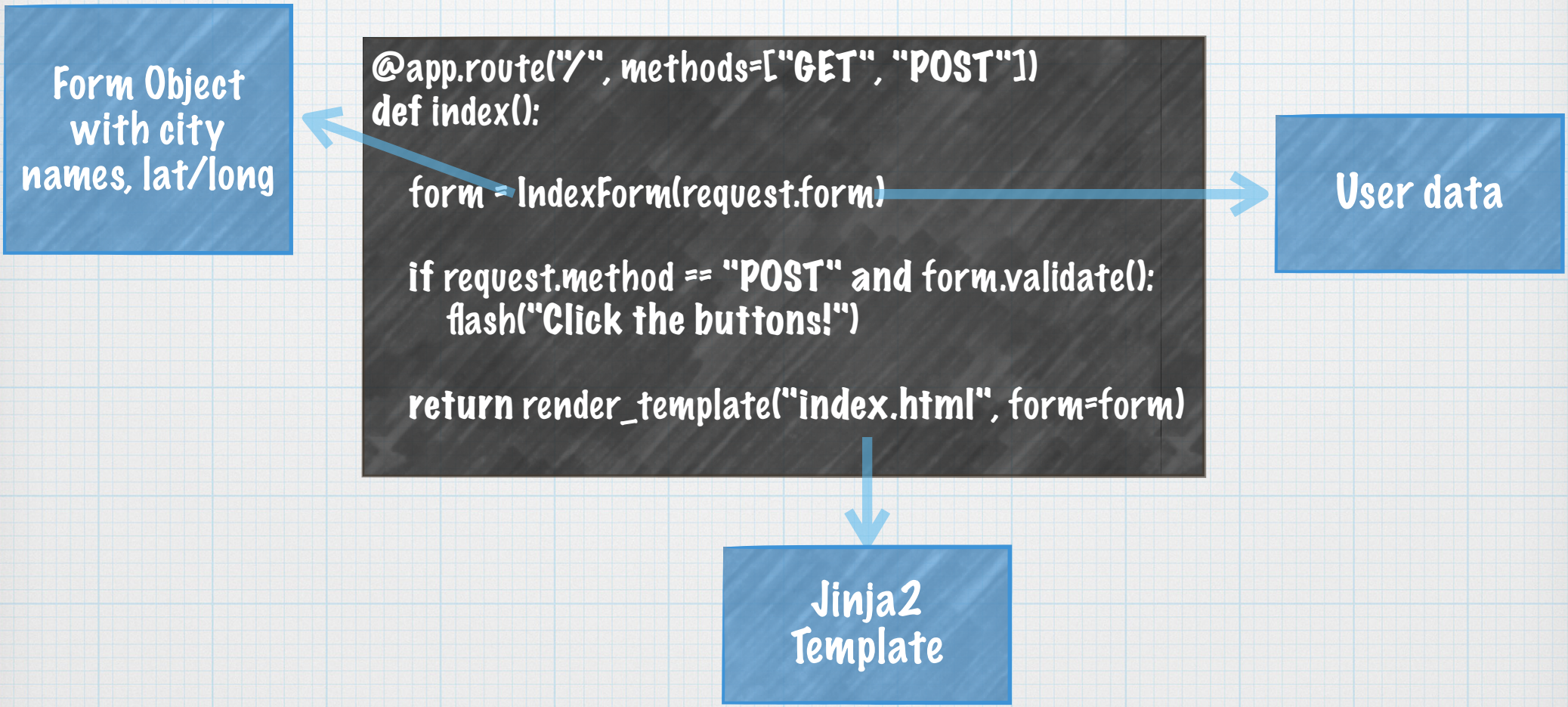
Quick Flask overview



Our Web Application

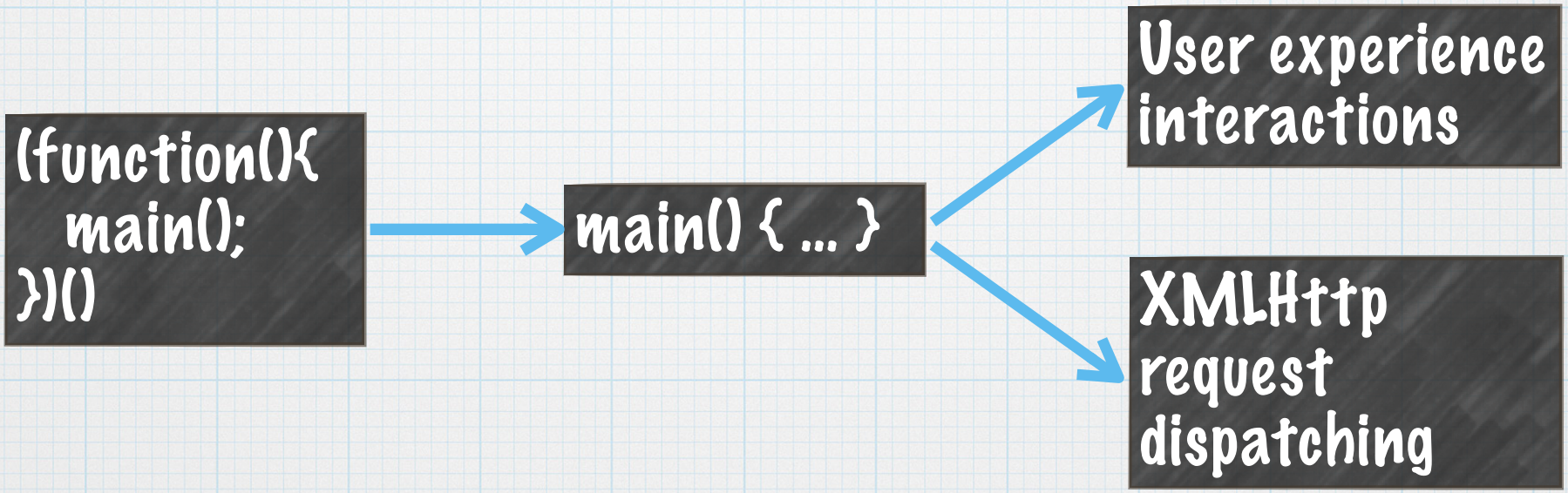


Flask serves the content



https://github.com/tristanfisher/ffi4wd/blob/master/jumper/views.py

JavaScript is loaded by the browser and executes



https://github.com/tristanfisher/ffi4wd/blob/master/jumper/static/js/main.js

XMLHTTPRequest to RESTful endpoint

```
function calculate_trip(backend_name, checkbox_list, backend_list, output){

    dispatch_endpoint_array = {
        "backend_python": "/api/python",
        "backend_ctypes": "/api/ctypes",
       [...]
   }
...
var city_arrays = get_city_selections(checkbox_list);
...
var request = $.ajax({
    method: "POST",
    url: endpoint,
    dataType: 'json',
    data: { "json": JSON.stringify(city_arrays) },
    success: function(data, textStatus, jqXHR){[...]
})
...
```

API Endpoints

```
blueprint_api = Blueprint("blueprint_python", __name__, url_prefix="/api")

@blueprint_api.route("/python_approximation", methods=["POST"])
...
@blueprint_api.route("/python", methods=["POST"])
...
@blueprint_api.route("/cython", methods=["POST"])
...
@blueprint_api.route("/ctypes", methods=["POST"])
...
@blueprint_api.route("/ffi", methods=["POST"])
...
```

https://github.com/tristanfisher/ffi4wd/blob/master/jumper/blueprints/api.py

Call Python backend, reply to request

```
from .backends.backend_dispatcher import backend_python
...
@blueprint_api.route("/python", methods=["POST"])
def api_python():

    user_data = req_to_tuple(request.form)
    start = time()
    result = backend_python(user_data)
    end = time()
    time_elapsed_seconds = format(end - start, '.6f')

    return jsonify(
        backend='python',
        result=result[0],
        backend_runtime=result[1],
        total_runtime=time_elapsed_seconds
    )
```

The backend doesn't know about the internet

```
def backend_python(data, approach='dfs', debug=False):
    graph_vertices_and_edges = generate_weights(data)
    start_time = time()
    if approach=='dfs':
        iteration_limit = factorial(len(graph_vertices_and_edges))
        result = calculate_dfs(graph_vertices_and_edges,
                               'new york, ny, usa',
                               iteration_limit,
                               debug)
    else:
        iteration_limit = calculate_number_edges(len(graph_vertices_and_edges))
        result = calculate_shortest_neighbors_trip(graph_vertices_and_edges, 'new york, ny, usa',
iteration_limit, debug)
    end_time = time()
    time_elapsed_seconds = format(end_time - start_time, '.6f')
    return result, time_elapsed_seconds
```

Pure Python solution

ffi4wd::jumper

Select the cities you want to travel between

11 locations selected

- [x] new york, ny, usa
- [x] portland, me, usa
- [x] seattle, wa, usa
- [x] san fransciso, ca, usa
- [x] denver, co, usa
- [x] austin, tx, usa
- [x] montreal, qc, ca
- [x] vancouver, bc, ca
- [x] anchorage, ak, usa
- [x] toronto, on, ca
- [x] berlin, germany

Select All | Select None

Python

C types

Cython

Python FFI

> 10 seconds

Run time: 24.090447s

Distance: 22999.26931015208km

1 : new york, ny, usa
2 : portland, me, usa
3 : montreal, qc, ca
4 : berlin, germany
5 : anchorage, ak, usa
6 : vancouver, bc, ca
7 : seattle, wa, usa
8 : san fransciso, ca, usa
9 : denver, co, usa
10 : austin, tx, usa
11 : toronto, on, ca
12 : new york, ny, usa

Python is efficient for the programmer...

	9 cities	10 cities	11 Cities
User Perceived Latency (seconds)	~0.19-0.23	~1.9-2.05	~20.5-24
Hard/Soft Deadline	Pass/Pass	Pass/Fail	Fail/Fail
Consequence	Happy user	User loses focus	Waste of work

(Run on 1.7Ghz Intel i7 - a ridiculously over-powered CPU for a "web server")

Confirm a bottleneck exists

Check the system your code is running on first.

```
$ ps -eo %cpu,%mem,command | grep "[pP]ython"
100.0  0.3 /usr/.../3.4/.../Python ./manage.py runserver
```

100% of a CPU core for the process, no multi-processing in use; bottleneck exists.

Find the bottleneck

```
def test_backend_python():
    #test_11_cities is test data
    cmd = "backend_python(test_11_cities)"
    cProfile.run(cmd)
```

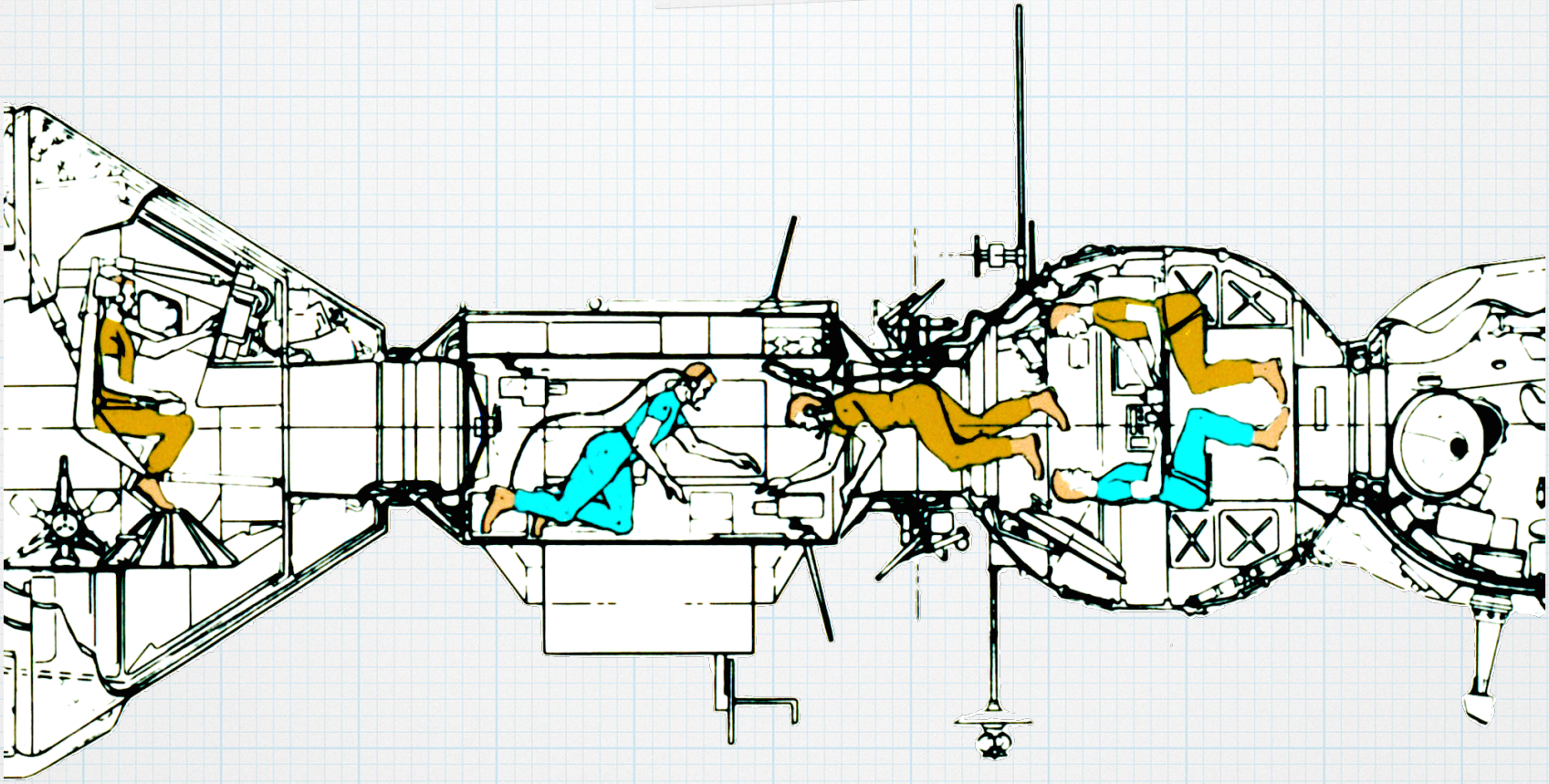
```
         19728203 function calls in 32.306 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000   32.305   32.305 <string>:1(<module>)
      110    0.000    0.000    0.000    0.000 backend_python.py:11(haversine)
        1    0.000    0.000   32.305   32.305 backend_python.py:249(backend_python)
        1    0.000    0.000    0.001    0.001 backend_python.py:42(generate_weights)
        1   27.663   27.663   32.305   32.305 backend_python.py:73(calculate_dfs)
      110    0.000    0.000    0.000    0.000 {built-in method asin}
      220    0.000    0.000    0.000    0.000 {built-in method cos}
        1    0.000    0.000   32.306   32.306 {built-in method exec}
        1    0.000    0.000    0.000    0.000 {built-in method factorial}
        1    0.000    0.000    0.000    0.000 {built-in method format}
 19728203    1.638    0.000    1.638    0.000 {built-in method len}
      440    0.000    0.000    0.000    0.000 {built-in method radians}
      220    0.000    0.000    0.000    0.000 {built-in method sin}
      110    0.000    0.000    0.000    0.000 {built-in method sqrt}
        2    0.000    0.000    0.000    0.000 {built-in method time}
 13493010    1.866    0.000    1.866    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
  9864101    1.137    0.000    1.137    0.000 {method 'pop' of 'list' objects}
```

More consequences of the bottleneck

* When the thread is running, no other work can happen (due to G.I.L.)
* This means no other work can happen in this process. Requests will stack up.

I don't want to redesign the algorithm.

I want it to be more efficient to meet the deadline.

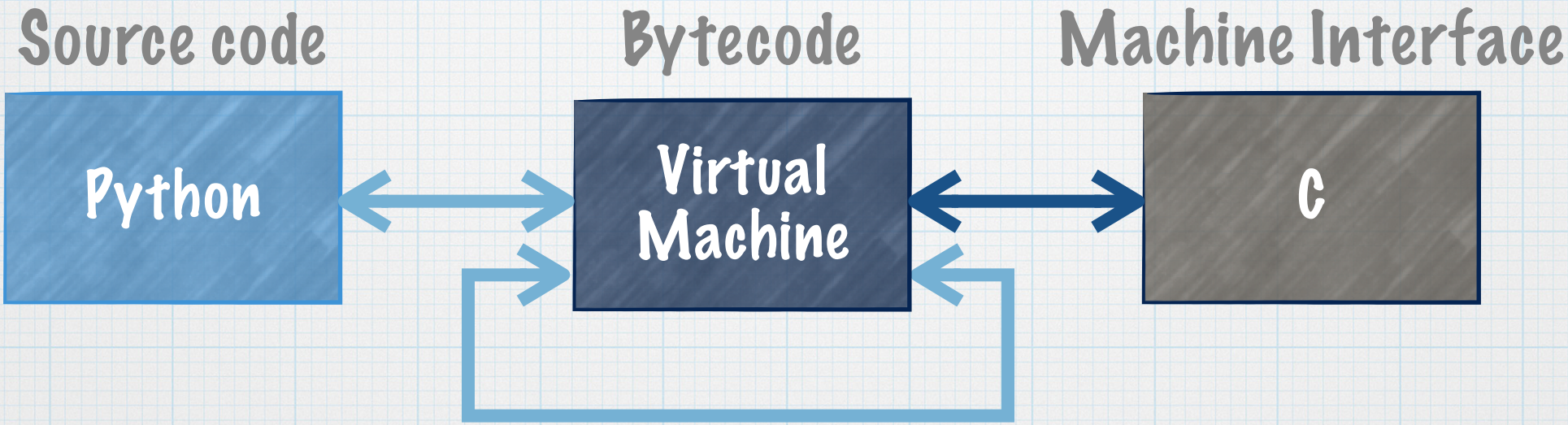


Python and Foreign Function Interfaces

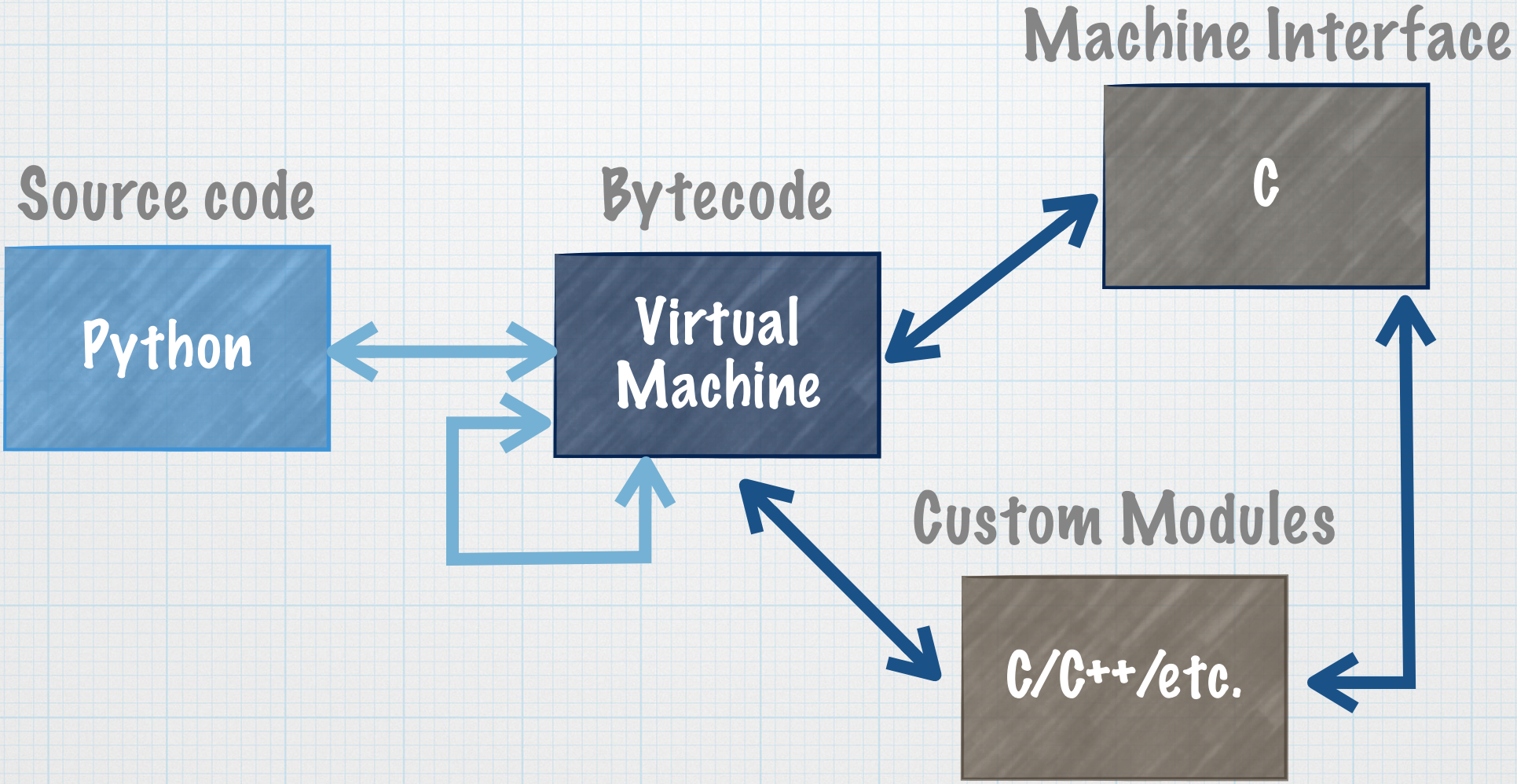
Foreign Function Interfaces allow code written in one programming language to call another within the same process.

e.g. Python and C++ in the same process.

Python already indirectly calls code in lower level languages



We can add custom lower-level extensions



CPython Foreign Function Interfaces Overview

* Write required functions for Python<->C compatibility
* Write your application/logic code
* Pass all interactions with Python through your compatibility functions
* Compile your code to a shared object
* Call your code from Python

CPython Extension Requirements

1) `#include <Python.h>` for your code to call CPython functions

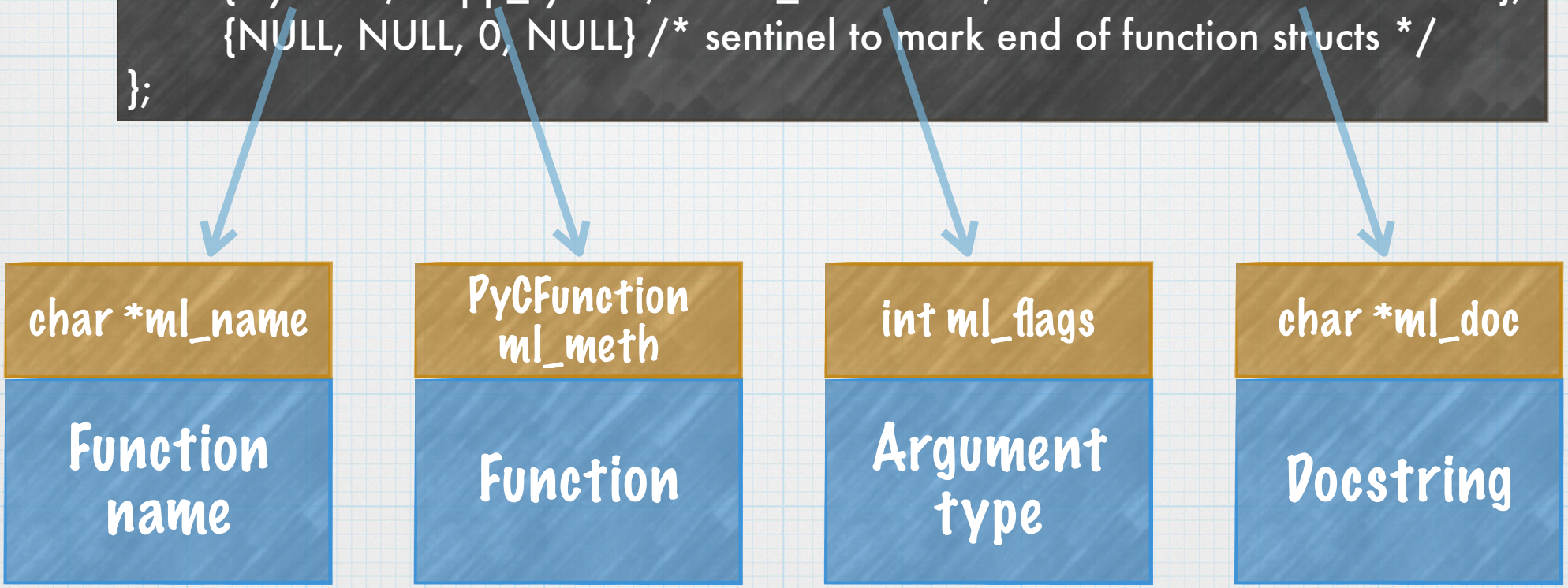
2) A C function to interface with Python

```
static PyObject *
FunctionName( PyObject *self, PyObject *args ){
–(or)–
FunctionName(PyObject *self, PyObject *args, PyObject *kwargs){
–(or)–
FunctionName( PyObject *self ){
  ...
  return PyLong_FromLong(return_code);
}
```

CPython Extension Requirements

3) A mapping table of C to Python-callable names

```
static PyMethodDef FFICppMethods[] = {
        {"system", fficpp_system, METH_VARARGS, "Execute a command shell."},
        {NULL, NULL, 0, NULL} /* sentinel to mark end of function structs */
};
```



CPython Extension Requirements

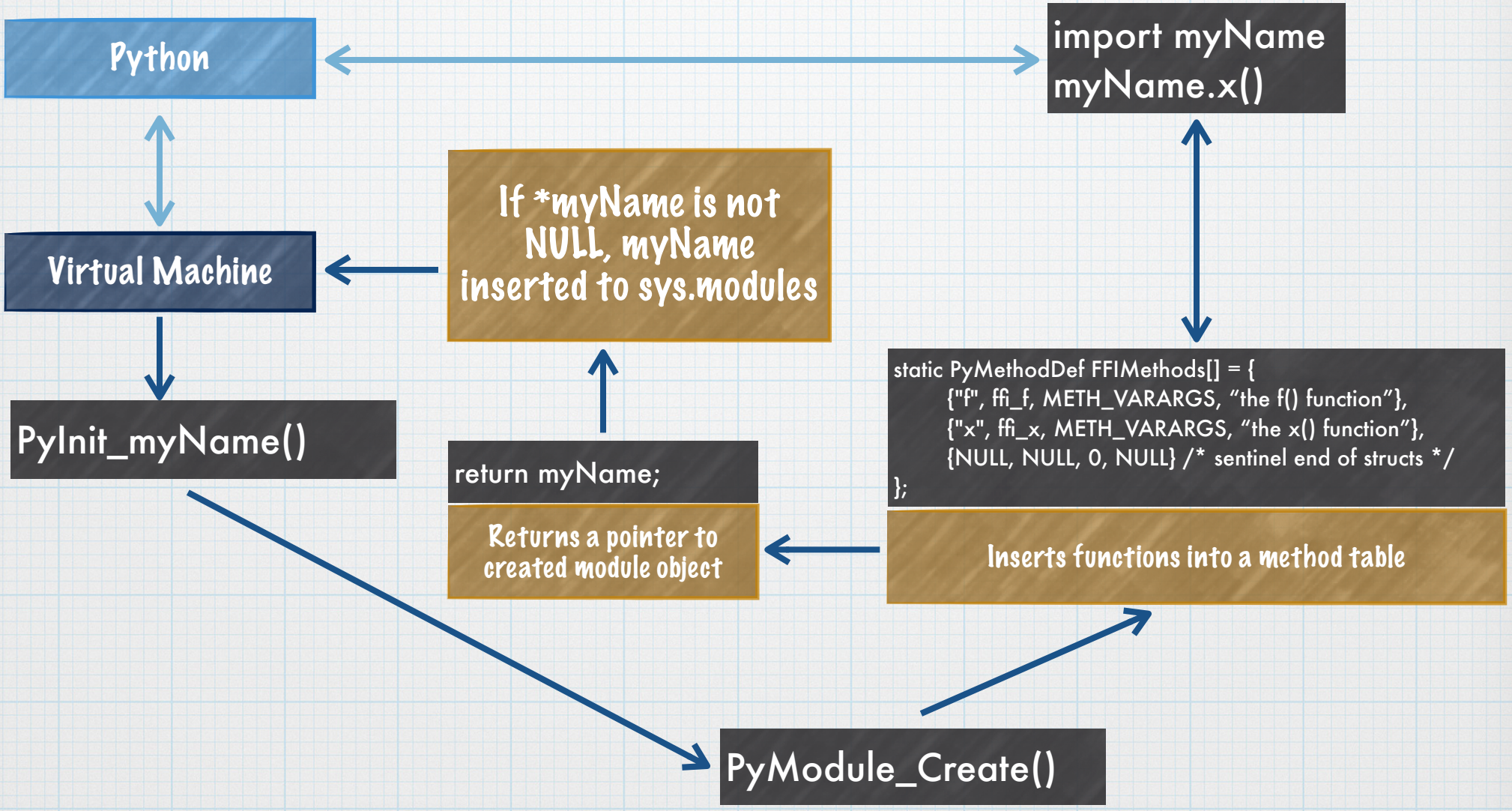
4) A module definition

```
static struct PyModuleDef fficppmodule = {
        PyModuleDef_HEAD_INIT,
        "fficpp",     /* module name */
        NULL,         /* module documentation (e.g. static char docs[] = "blah")*/
        -1,           /* size of per-interpreter module state. -1 for state in global vars */
        FFICppMethods   /* module mapping table */
};
```

5) An initialization function

```
PyMODINIT_FUNC
PyInit_fficpp(void)
{
    return PyModule_Create(&fficppmodule);
}
```

Extension import process



We must compile our lower-level extensions

* Compiling decreases abstractions and improves performance at the cost of portability
* Code must be compiled for each platform (`git clone app_repo` & `./source.cc` won't work)

Makefiles 101

Makefiles are great for unambiguous build scripts

```
#example makefile section:
…
cython:
    @echo "Making Cython required files..."
    cd $(SRC_CYTHON); $(PYTHON) setup.py build_ext –inplace;
    mv $(SRC_CYTHON)*.so $(DST_CYTHON).
…
```

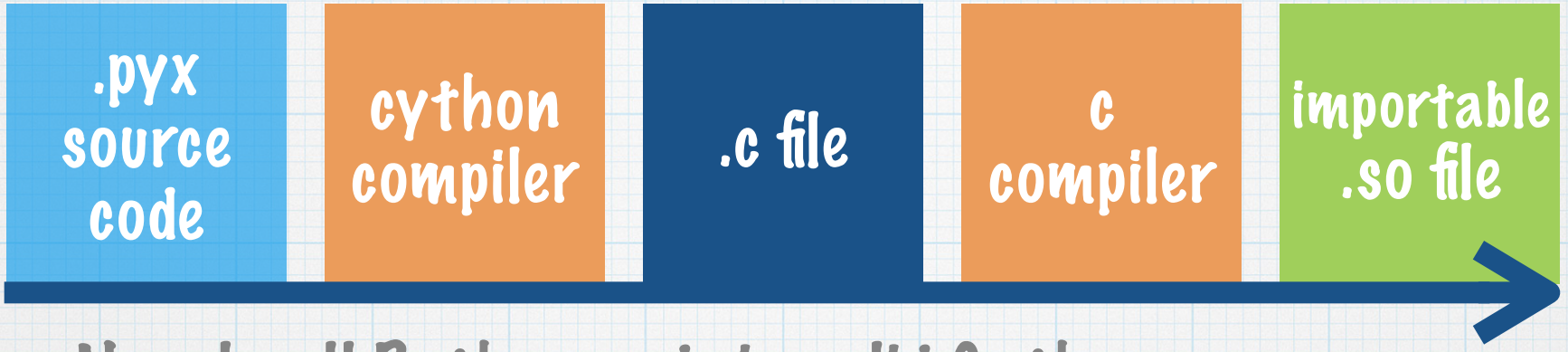
```
$ make cython
```

"Change into the source code directory and use Python's great distutils module"

We'll cover FFI via tools

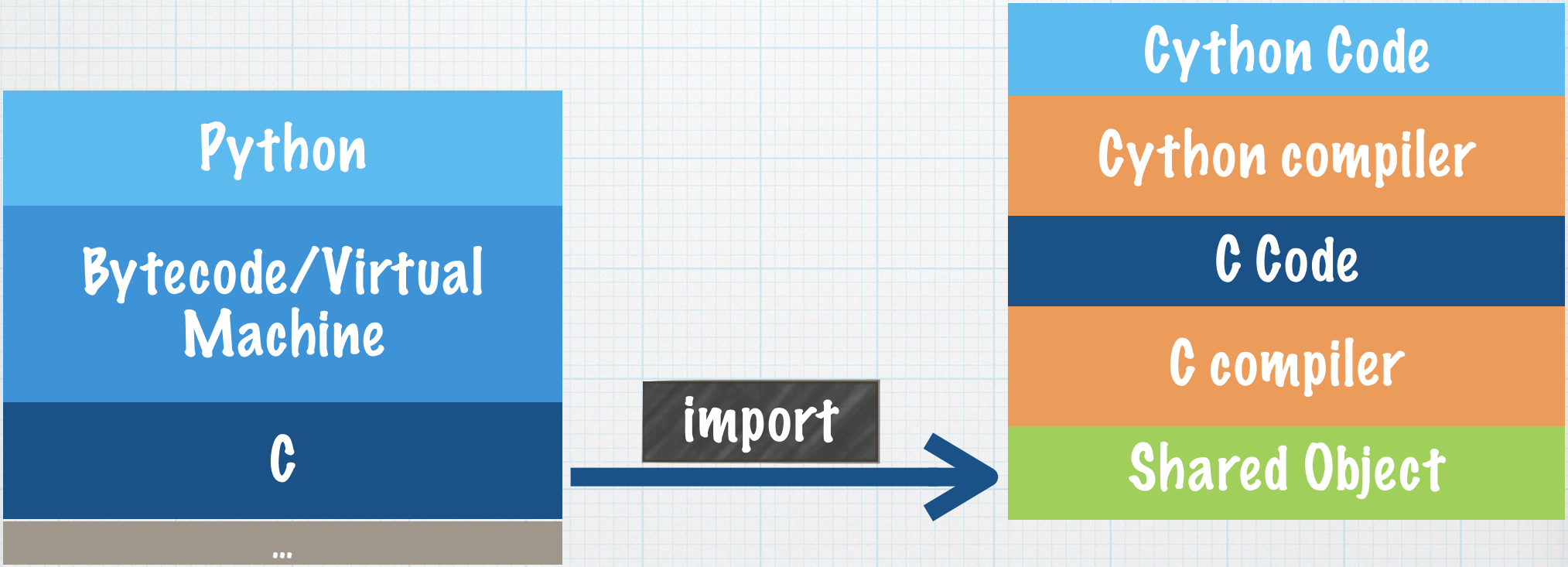
* Cython - Superset of Python with C data types and code compiled to C instead of bytecode.
* CTypes (stdlib) - Library that gives access to C data types, libc functions, and calling shared objects

Cython



* Nearly all Python code is valid Cython
* You can write Cython-specific code (not compatible with Python) to get further performance increases
* Commonly distinguished with a .pyx extension
* Cython code must be compiled

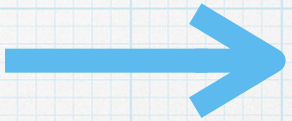
Cython overview



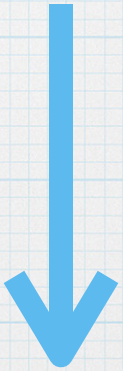
Cython (.pyx) source code is compiled to a shared object (.so), which is imported in Python as a native module.

Cython can be normal Python that you compile

```
def haversine(lat1, long1, lat2, long2, planet_radius=EARTH_RADIUS_KM):
    phi_Lat = radians(lat2 - lat1)
    phi_Long = radians(long2 - long1)
    lat1 = radians(lat1)
    lat2 = radians(lat2)

    a = sin(phi_Lat/2)**2 + \
        cos(lat1) * cos(lat2) * \
        sin(phi_Long/2)**2

    c = 2 * asin(sqrt(a))
    return planet_radius * c
```



compile to C



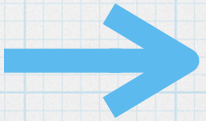
```
...
    static PyObject *__pyx_pw_20ffi_cython_py_compat_1haversine(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
static char __pyx_doc_20ffi_cython_py_compat_haversine[] = "";
static PyMethodDef __pyx_mdef_20ffi_cython_py_compat_1haversine = {"haversine", (PyCFunction)__pyx_pw_20ffi_cython_py_compat_1haversine,
METH_VARARGS|METH_KEYWORDS, __pyx_doc_20ffi_cython_py_compat_haversine};
static PyObject *__pyx_pw_20ffi_cython_py_compat_1haversine(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
  PyObject *__pyx_v_lat1 = 0;
  PyObject *__pyx_v_long1 = 0;
  PyObject *__pyx_v_lat2 = 0;
  ...
```

Or Python-incompatible Cython

```
cdef float EARTH_RADIUS_KM = 6372.8
cdef extern from "math.h":
    float sinf(float x)
    float cosf(float x)
...

cdef float haversine(float lat1, float long1, float lat2,
                     float long2, float planet_radius=EARTH_RADIUS_KM):
    phi_Lat = radians(lat2 - lat1)
    phi_Long = radians(long2 - long1)
    lat1 = radians(lat1)
    lat2 = radians(lat2)

    a = sinf(phi_Lat/2)**2 + cosf(lat1) * cosf(lat2) * sinf(phi_Long/2) ** 2
...
```

Interfacing directly with C

Declaring return and argument types



```
static float __pyx_v_10ffi_cython_EARTH_RADIUS_KM;
static float __pyx_f_10ffi_cython_haversine(float, float, float, float, struct __pyx_opt_args_10ffi_cython_haversine *__pyx_optional_args); /*proto*/
...
#define __Pyx_MODULE_NAME "ffi_cython"
```

$ make cython

After running our makefile, we have shared-object binaries:

```
jumper/blueprints/backends/modules/cython
├── ffi_cython.so
└── ffi_cython_py_compat.so
```

```
$ file jumper/blueprints/backends/modules/cython/*.so

ffi_cython.so:            Mach-O 64-bit bundle x86_64
ffi_cython_py_compat.so: Mach-O 64-bit bundle x86_64
```

Calling our Cython module

```
from .backends.modules.cython.ffi_cython_py_compat \
    import backend_cython as backend_cypy
...
@blueprint_api.route("/cython", methods=["POST"])
def api_cython():

    user_data = req_to_tuple(request.form)
    start = time()
    result = backend_cypy(user_data)
    end = time()
    time_elapsed_seconds = format(end - start, '.6f')

    return jsonify(
        backend='cython',
        result=result[0],
        backend_runtime=result[1],
        total_runtime=time_elapsed_seconds
    )
```

Only the bottleneck-code is being replaced!

Cython (Python compatible)

ffi4wd::jumper

Select the cities you want to travel between

11 locations selected

- [x] new york, ny, usa
- [x] portland, me, usa
- [x] seattle, wa, usa
- [x] san fransciso, ca, usa
- [x] denver, co, usa
- [x] austin, tx, usa
- [x] montreal, qc, ca
- [x] vancouver, bc, ca
- [x] anchorage, ak, usa
- [x] toronto, on, ca
- [x] berlin, germany

Select All | Select None

Python

CTypes

Cython

Python FFI

Closer, but still > 10 seconds

Run time: 11.948986s

Distance: 22999.26931015208km

1 : new york, ny, usa

2 : portland, me, usa

3 : montreal, qc, ca

4 : berlin, germany

5 : anchorage, ak, usa

6 : vancouver, bc, ca

7 : seattle, wa, usa

8 : san fransciso, ca, usa

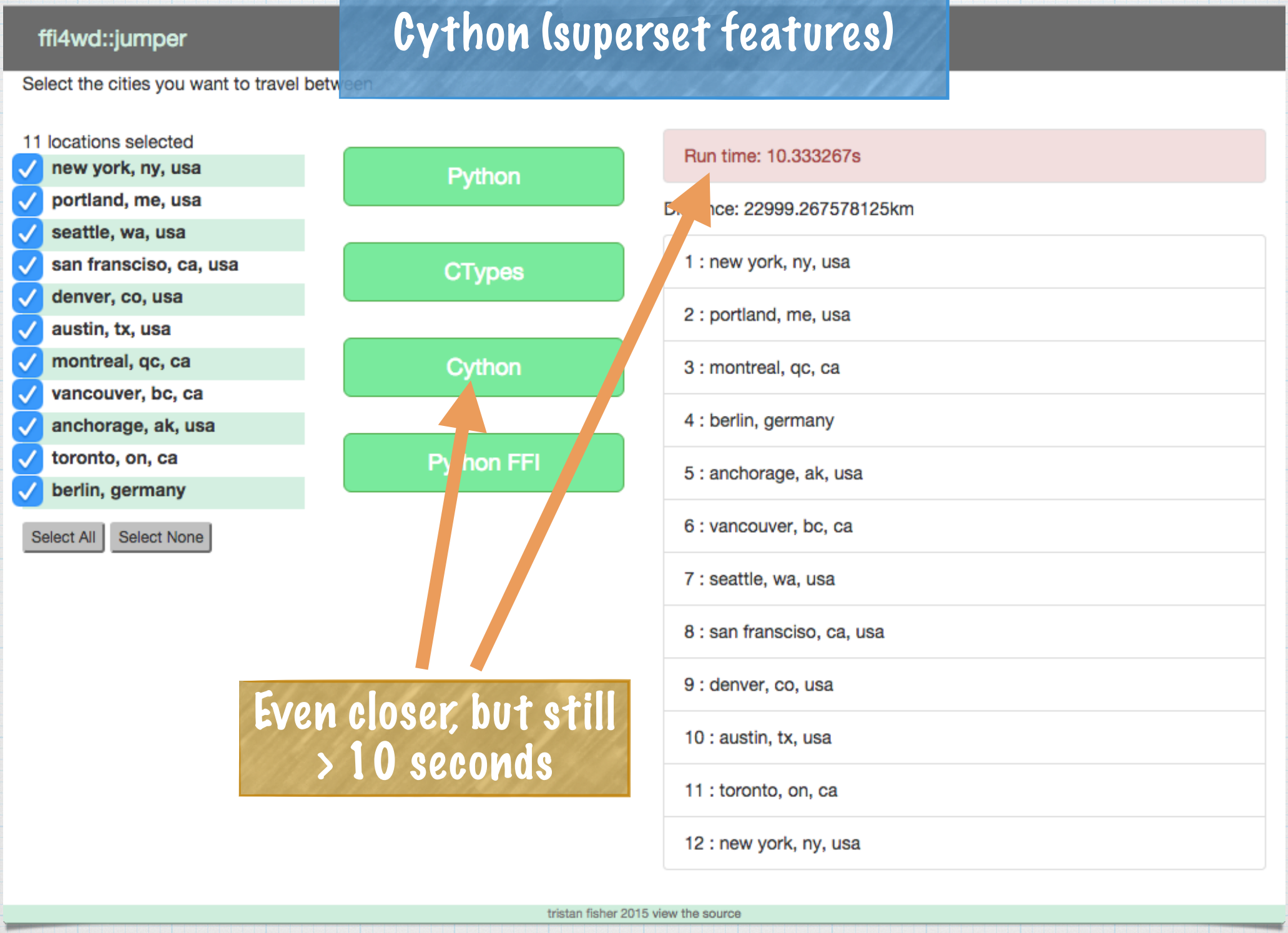
9 : denver, co, usa

10 : austin, tx, usa

11 : toronto, on, ca

12 : new york, ny, usa

Cython (superset features)



Cython has varying efficiency

Python-compatible Cython

	9 cities	10 cities	11 Cities
User Perceived Latency (seconds)	~0.10-0.13	~1.0-1.3	~11.9-13.8
Hard/Soft Deadline	Pass/Pass	Pass/Pass-Fail	Fail/Fail
Consequence	Happy user	User may lose focus	Waste of work

Cython using superset features

	9 cities	10 cities	11 Cities
User Perceived Latency (seconds)	~0.09-0.12	~0.98-1.1	10.3-11.5
Hard/Soft Deadline	Pass/Pass	Pass/Pass-Fail	Fail/Fail
Consequence	Happy user	User may lose focus	Waste of work

(still 100% CPU utilization)

Has the bottleneck changed?

```
      4 function calls in 12.786 seconds

Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.000    0.000   12.786   12.786 <string>:1(<module>)
     1    0.000    0.000   12.786   12.786 {built-in method exec}
     1   12.786   12.786   12.786   12.786 {ffi_cython_py_compat.backend_cython}
     1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

Python compatible

```
      4 function calls in 10.731 seconds

Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.000    0.000   10.730   10.730 <string>:1(<module>)
     1   10.730   10.730   10.730   10.730 {built-in method backend_cython}
     1    0.000    0.000   10.731   10.731 {built-in method exec}
     1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

Cython specific

```
$ ps -eo %cpu,%mem,command | grep "[pP]ython"
100.1  0.1 /usr/../3.4/Python ./manage.py runserver
```

More info how to debug: http://docs.cython.org/src/userguide/debugging.html

'built-in' means 'not bytecode'

Python compatible

```
      4 function calls in 12.786 seconds

Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.000    0.000   12.786   12.786 <string>:1(<module>)
     1    0.000    0.000   12.786   12.786 {built-in method exec}
     1   12.786   12.786   12.786   12.786 {ffi_cython_py_compat.backend_cython}
     1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

Cython specific

```
      4 function calls in 10.731 seconds

 Ordered by: standard name

 ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      1    0.000    0.000   10.730   10.730 <string>:1(<module>)
      1   10.730   10.730   10.730   10.730 {built-in method backend_cython}
      1    0.000    0.000   10.731   10.731 {built-in method exec}
      1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

Cython tips

Instead of:

```
from x.y.z import my_data
def generate_weights(my_data):
    ...
```

Use closures and argument passing:

```
def generate_weights(data):
    ...
```

Use distutils to save time generating compile flags.

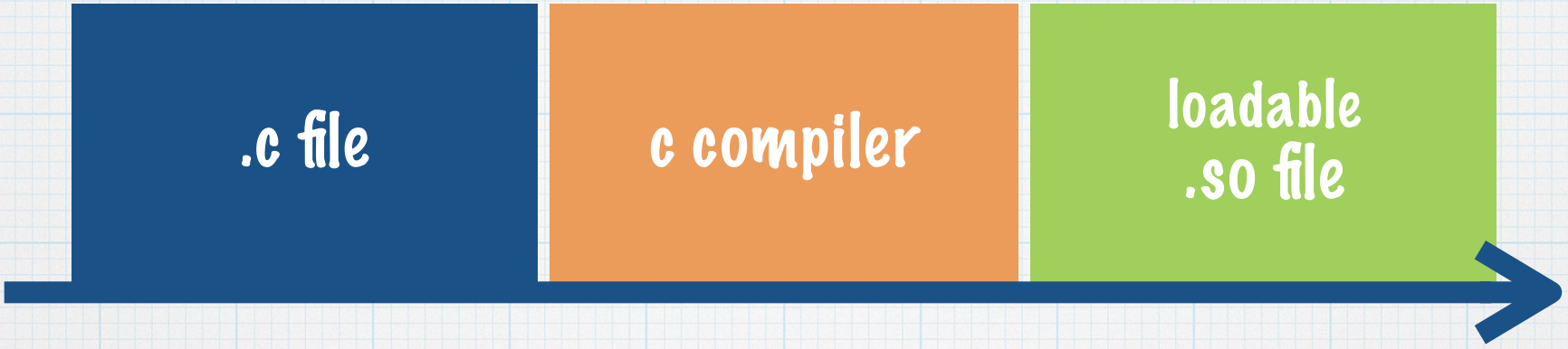
```
python setup.py build_ext –inplace

clang … -fno-commo -dynamic …ffi_cython.o
clang -bundle -undefined dynamic_lookup -L … ffi_cython.so
```

There's treasure in /usr/include/

```
cdef extern from "(alternative function to call).h"
```

Run "cython -a" on your .pyx files. It does something amazing.

CTypes



* Great making C data types available and for calling shared objects
* Handles most CPython FFI-interface setup and initialization for you
* Including shared-objects is a function call, not just an import

CTypes: write, compile, load in Python

C++

```
#include <iostream>
class HelloWorld{
    public:
        void example_world(){
            std::cout << "hello world" << std::endl;
        }
};

// name mangling, make sure we can call this from Python
extern "C" {
    HelloWorld* HelloWorld_new(){
        return new HelloWorld();
    }

    void HelloWorld_example_world(HelloWorld* world){
        world->example_world();
    }
}
```

Python

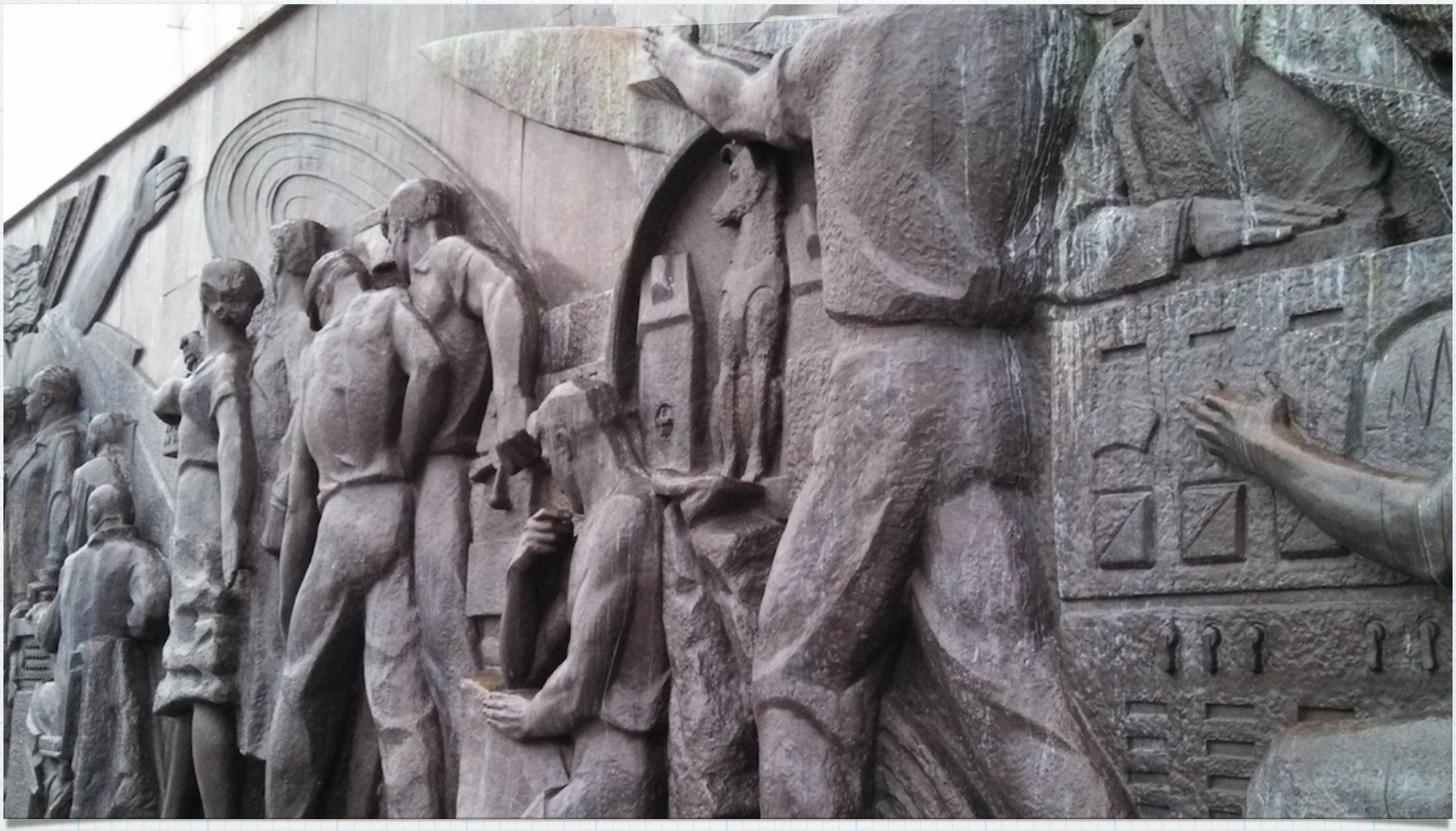
```
from ctypes import cdll

path = "libffi_ctypes.so"
lib = cdll.LoadLibrary(path)

class FFICTypes:
    def __init__(self):
        self.obj = lib.HelloWorld_new()

    def example_world(self):
        lib.HelloWorld_example_world(self.obj)

t = FFICTypes()
t.example_world()
```

Note that we didn't need to manually fulfill any of the CPython extension requirements.

"Raw" C++ FFI development

* You only need to 'wrap' the functions called by Python
* C++ does "name-mangling"; the C++ to C FFI has to be used for Python-callable functions:

```
extern "C" {
PyMODINIT_FUNC
PyInit_modname(){
  return PyModule_Create(&fficppmodule);
}
```



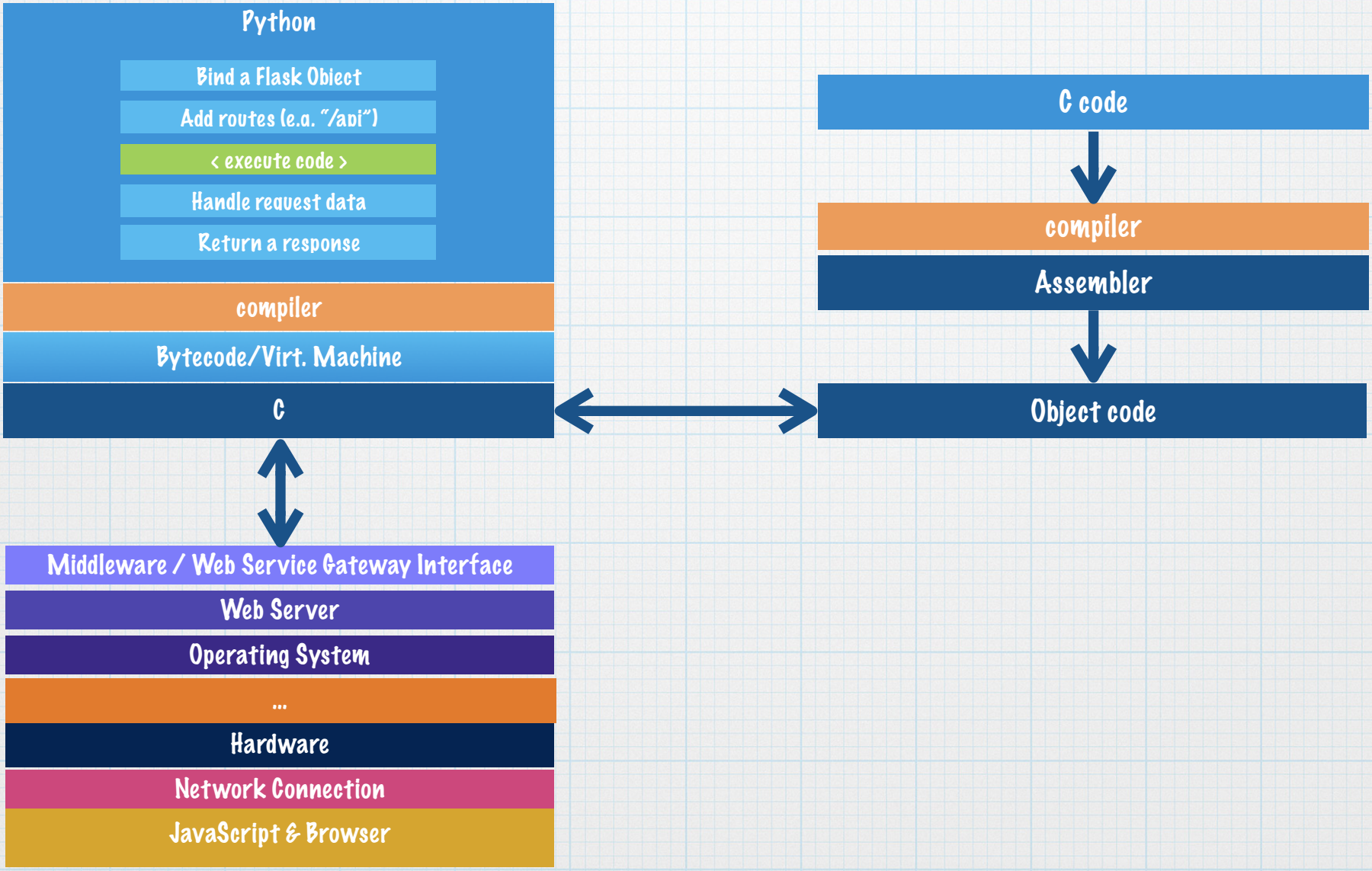
Topic review and closing notes

Stay in Python when you can, which is almost always for Web Development.

Python is efficient for programmers, meets most deadlines

* Application plumbing fast enough to not be worth mentioning in this talk
* Web framework not compiled, only machine-inefficient parts
* Python amazing for logic validation — write, refactor, improve

The Foreign Function Interface



Use highly machine-efficient approximations if possible

ffi4wd::jumper

Select the cities you want to travel between

11 locations selected

- [x] new york, ny, usa
- [x] portland, me, usa
- [x] seattle, wa, usa
- [x] san fransciso, ca, usa
- [x] denver, co, usa
- [x] austin, tx, usa
- [x] montreal, qc, ca
- [x] vancouver, bc, ca
- [x] anchorage, ak, usa
- [x] toronto, on, ca
- [x] berlin, germany

Select All | Select None

Python

CTypes

Cython

Python FFI

Python Approximation

Run time: 0.000241s

Distance: 24189.975385858408km

1 : new york, ny, usa
2 : portland, me, usa
3 : montreal, qc, ca
4 : toronto, on, ca
5 : denver, co, usa
6 : austin, tx, usa
7 : san fransciso, ca, usa
8 : seattle, wa, usa
9 : vancouver, bc, ca
10 : anchorage, ak, usa
11 : berlin, germany
12 : new york, ny, usa

Very fast, but not exact

Foreign Function Interfaces are useful for more than performance

* Creativity
* Pragmatic education - prototype in Python and use the "guest language" for interesting components
* Intellectually satisfying

Closing notes

* Profile your code before trying to optimizations (is `print()` expensive?)
* Only optimize and use lower level languages for the slow parts
* Offloading easy to write, often-called functions isn't a bad idea
* When lost, source dive: https://hg.python.org/cpython/file/3.4/Modules/

Q & A

Image credits and references

Images from NASA	Other Images	References
"Shuttle Atlantis returning to Kennedy Space Center"	"Voskhod-3KD" D.I. Kozlov et al., Конструирование Автоматических Космических Алларатов, Mashinstroenie, 1996	[1] https://www.igvita.com/slides/2013/breaking-1s-mobile-barrier.pdf
Apollo Service Module		[2] derived from http://norvig.com/21-days.html#answers
Apollo Lunar Lander		
Apollo Launch Configuration		
"Apollo-Soyuz Rendezvous and Docking Test Project"		
Lunar Module Controls and Displays		
Analog Computing Machine in Fuel Systems Building		
"IBM Electronic Data Processing Machine"		
S74-05269 (Internal Arrangement of Docked Configuration)		
40 x 80 Foot Data Acquisition System		

Images not otherwise attributed either public domain or created by me.

Thank you!

Code and presentation: github.com/tristanfisher/ffi4wd

/tristanfisher tristanfisher.com