The future of Python on the Web

2019.11.29

Michael Droettboom
Staff Data Engineer
Mozilla Corporation

My data journey

UNDER THE ANHEUSER BUSH

MUSIC BY
HARRY VON TILZER.

WORDS BY
ANDREW B STERLING.

HARRY VON TILZER
MUSIC PUBLISHING Co.
37 W 28th ST NEW YORK. CHICAGO. FRISCO. LONDON

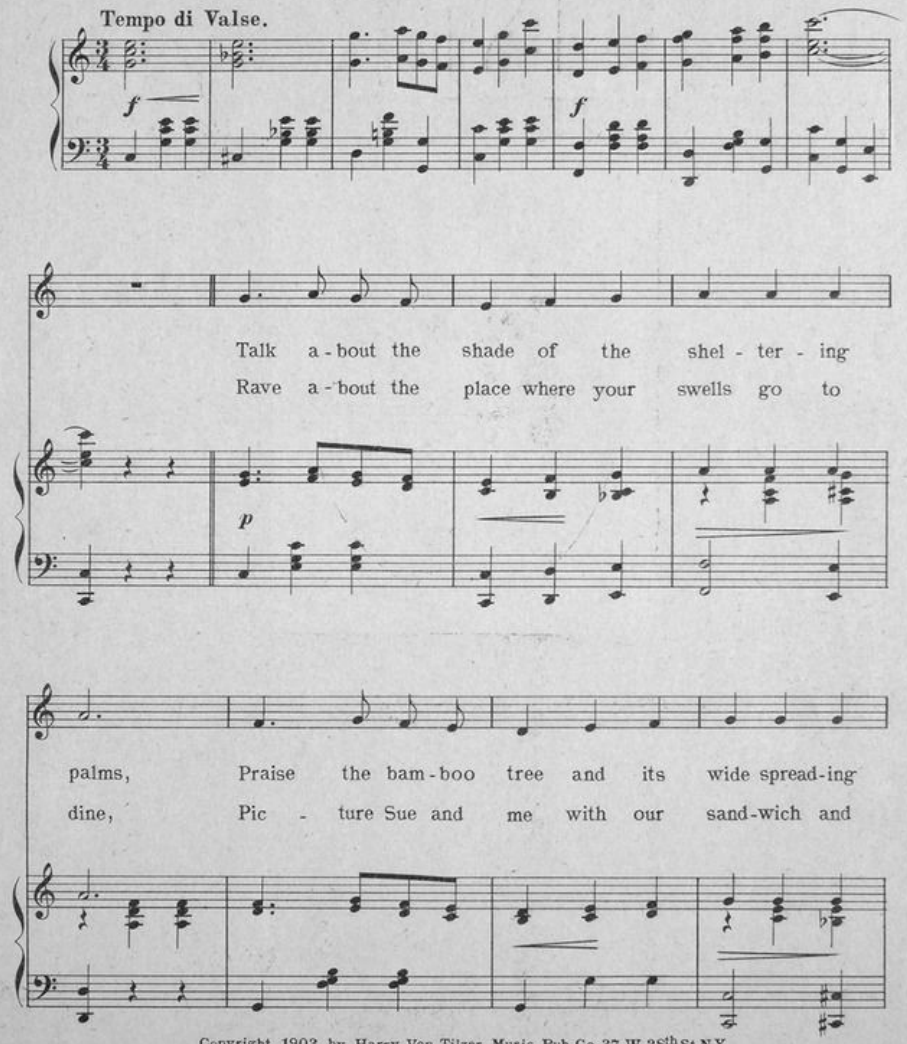
5

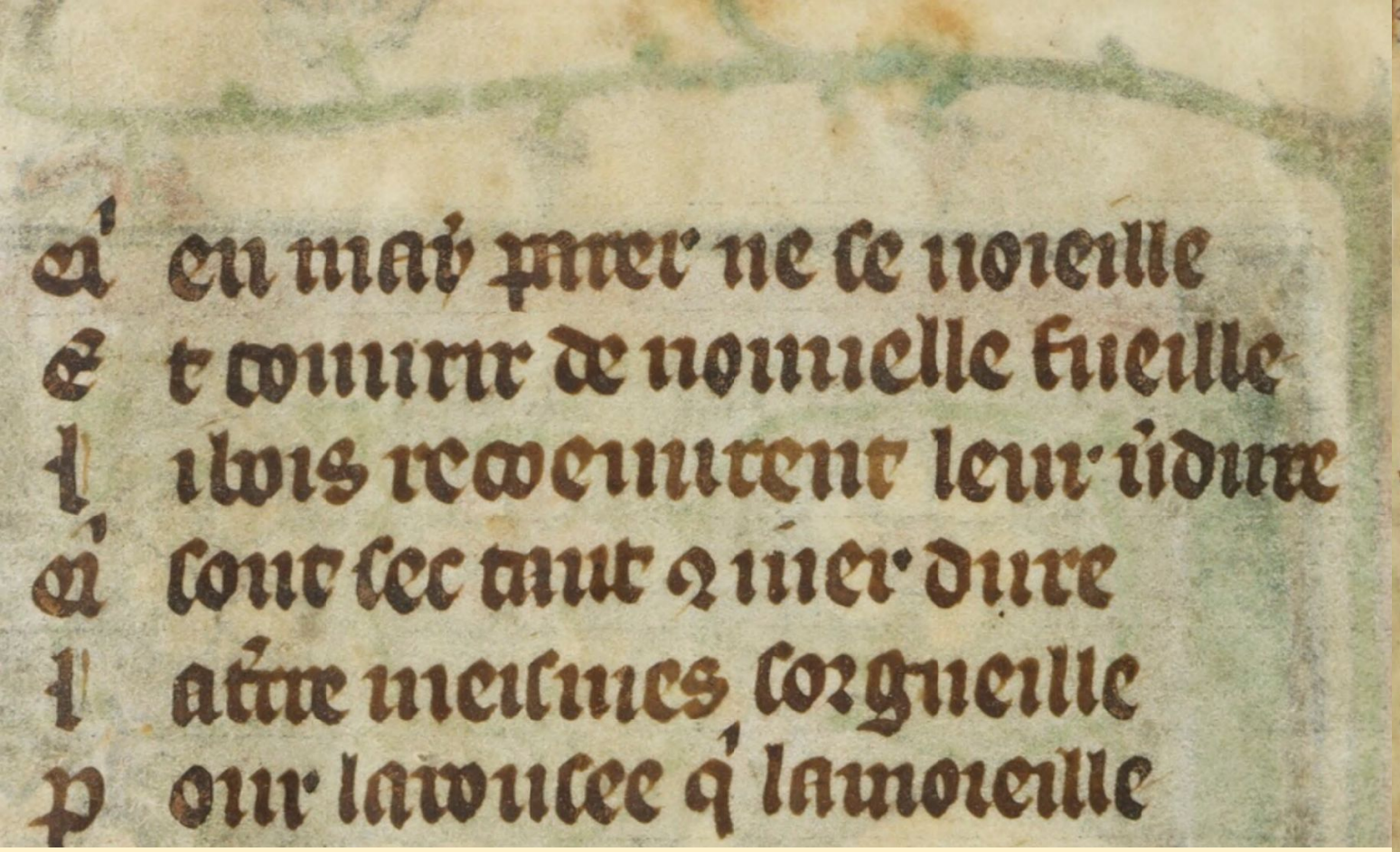
"Under The Anheuser Bush".

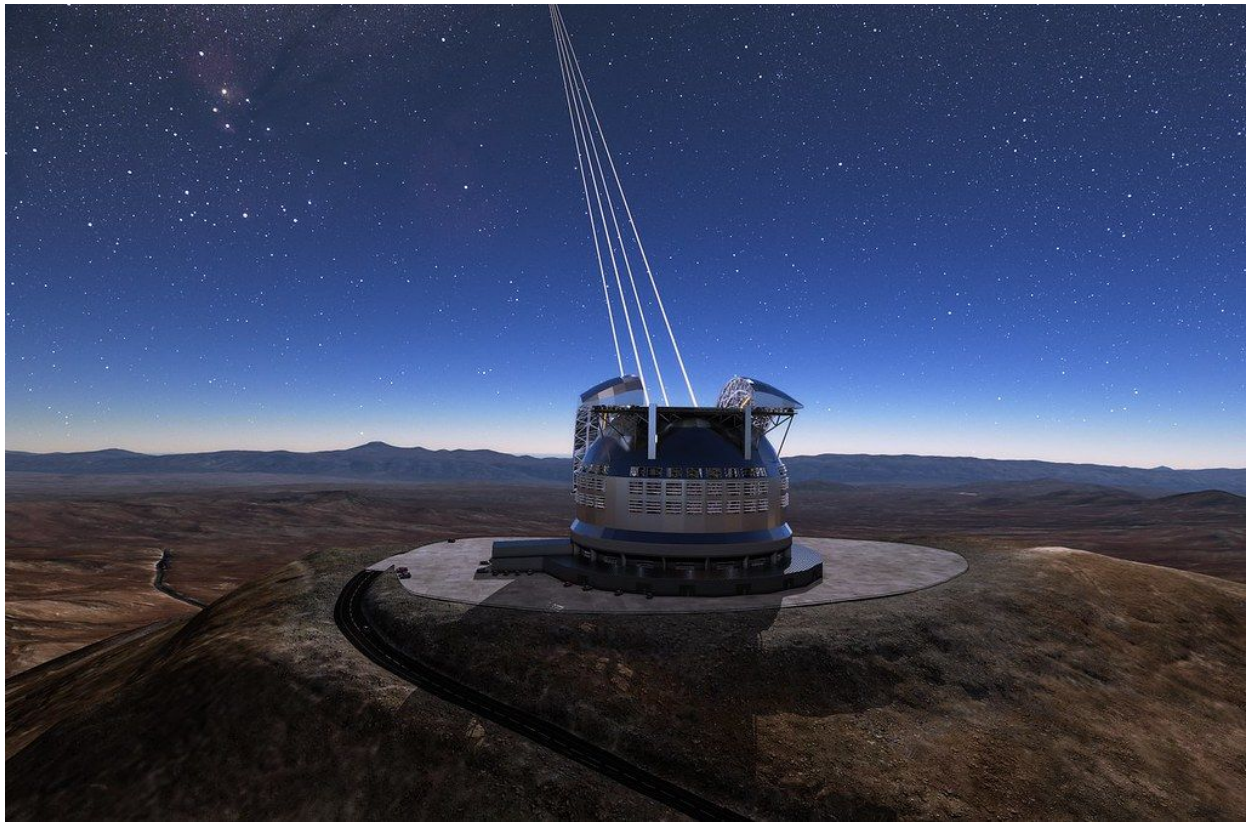
Words by
ANDREW B. STERLING.

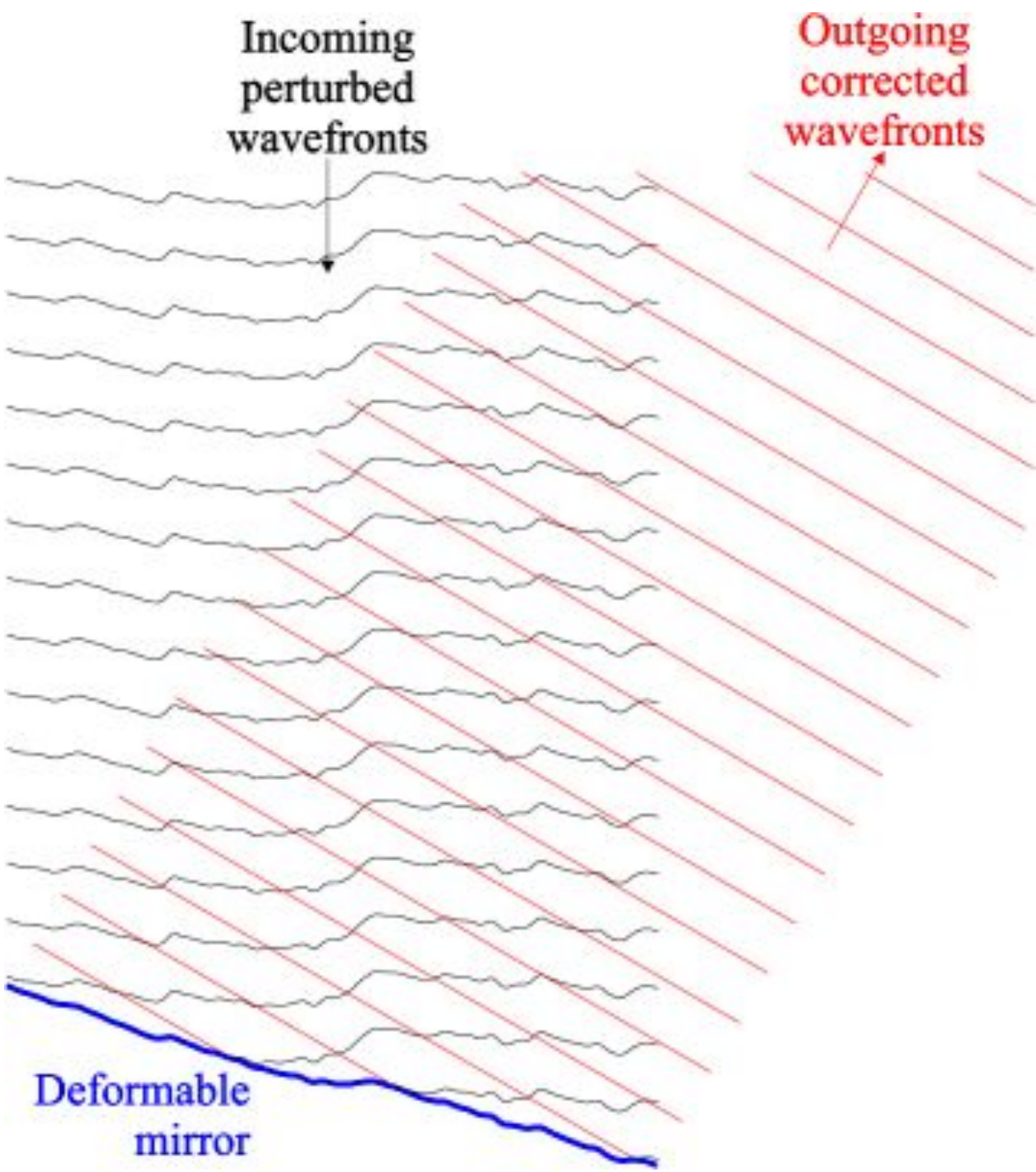
Music by
HARRY VON TILZER.

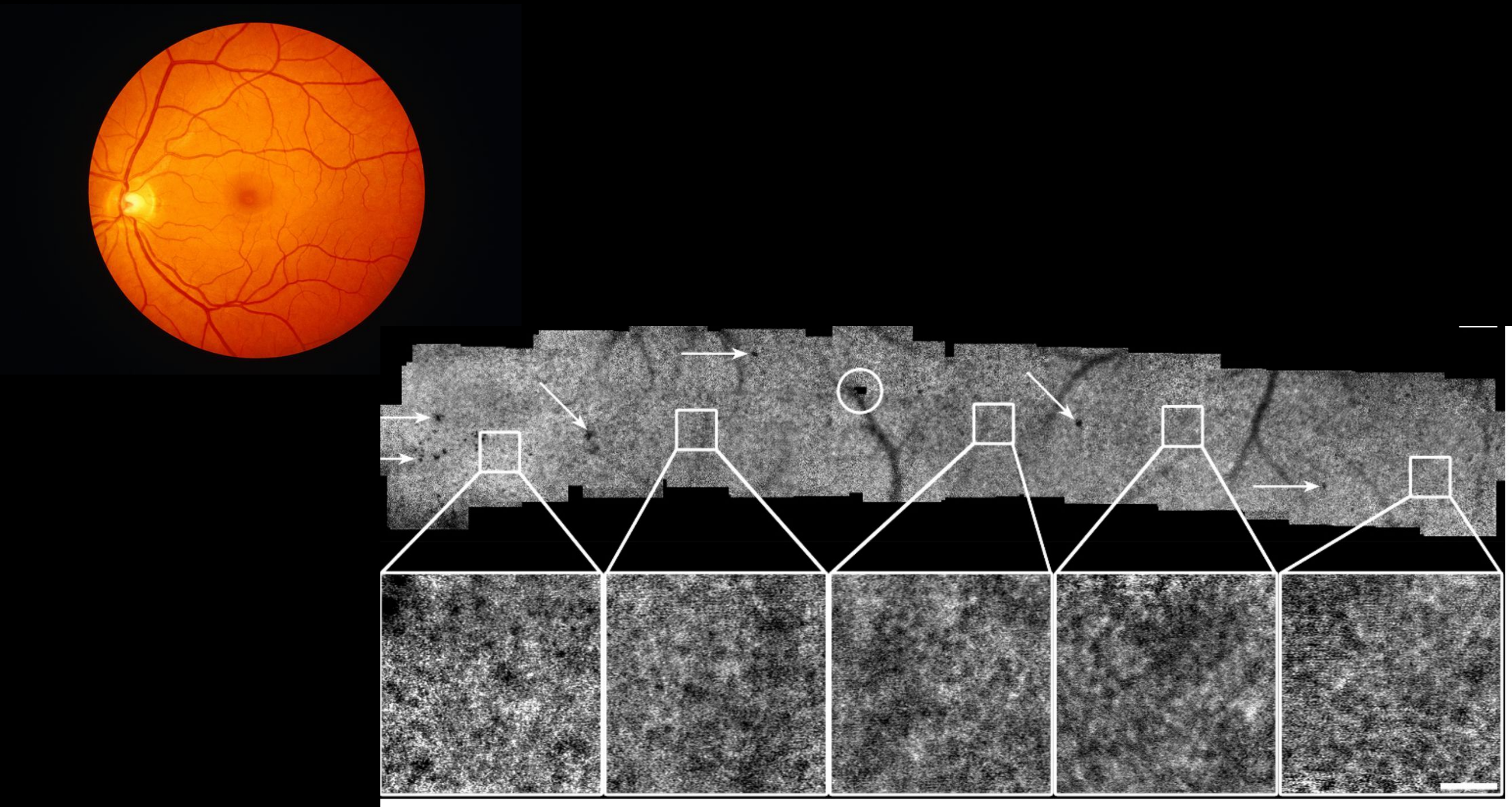
Tempo di Valse.







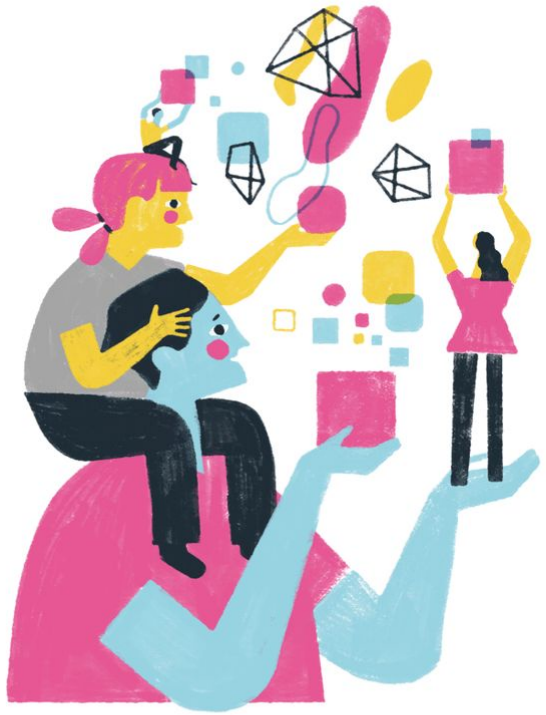

Incoming
perturbed
wavefronts
Outgoing
corrected
wavefronts
Deformable
mirror



Lean Data Practices

Stay Lean

Decide if all your data collection delivers value.



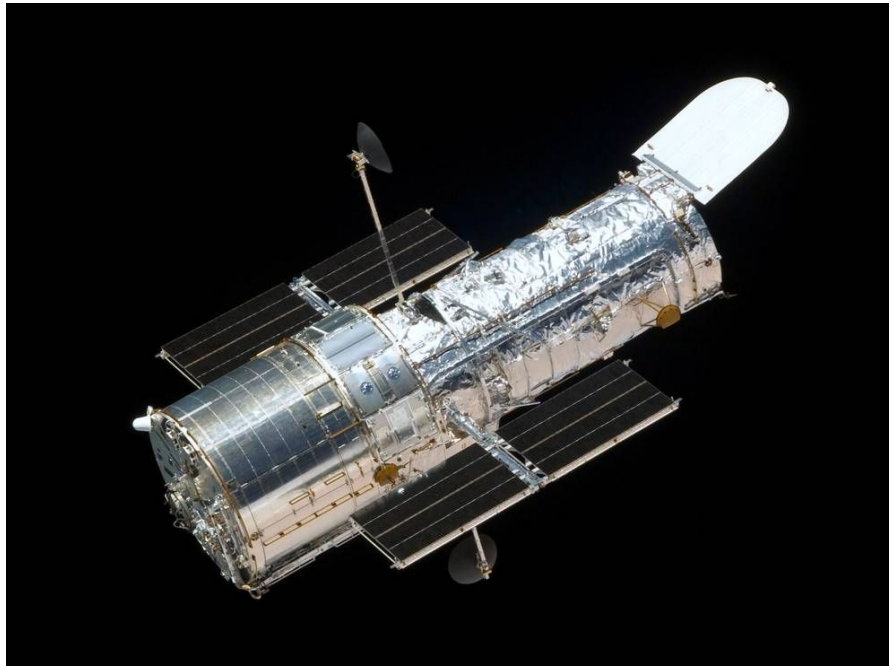
Build Security

Learn how to protect customer data.

Engage Your Users

Keep customers informed and empowered.

https://www.mozilla.org/en-US/about/policy/lean-data/



vs.

universal

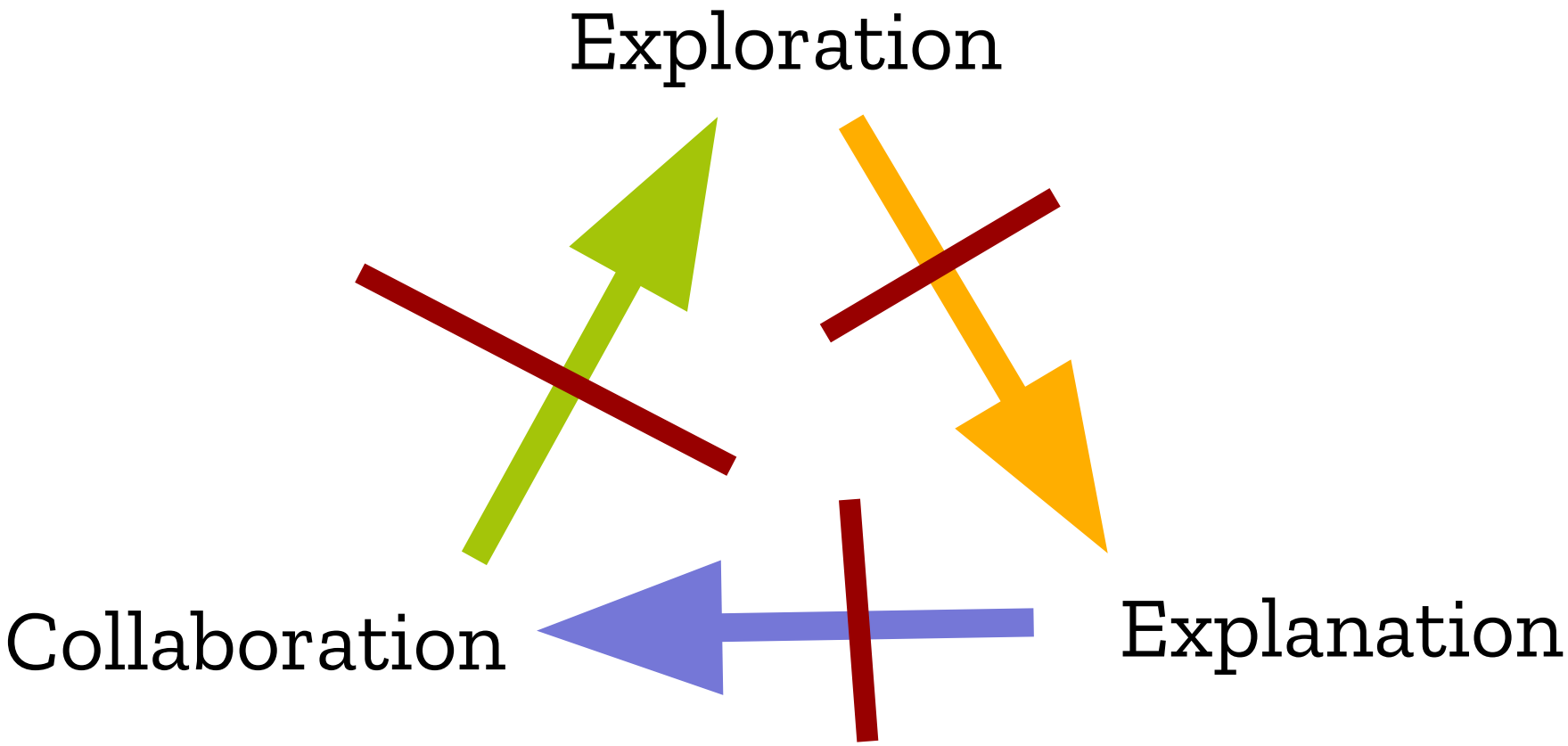
18GB / day

(potentially)
specific

2TB / day

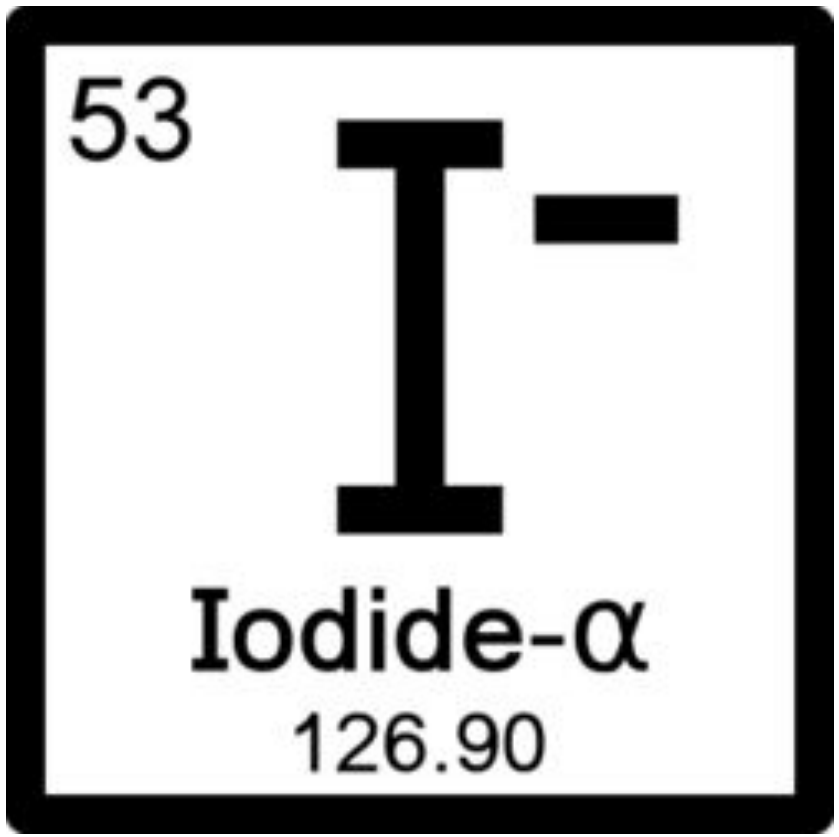
Communicating about Data Science

The lifecycle of data science



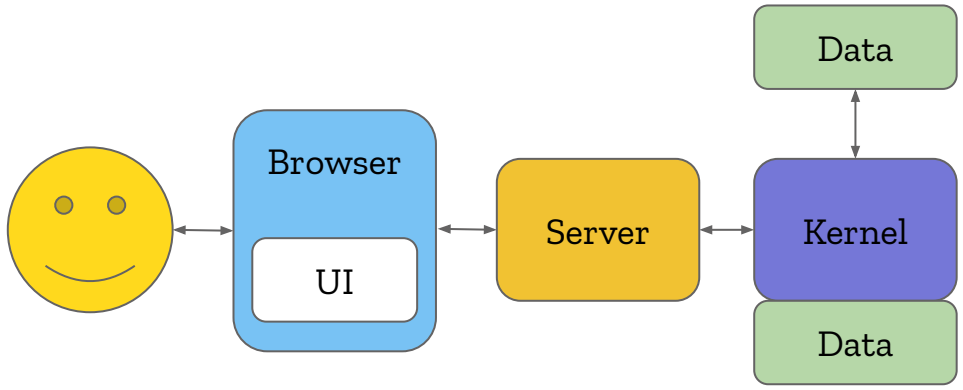
Adam Rule, Aurélien Tabard, James D. Hollan

Exploration and Explanation in Computational Notebooks

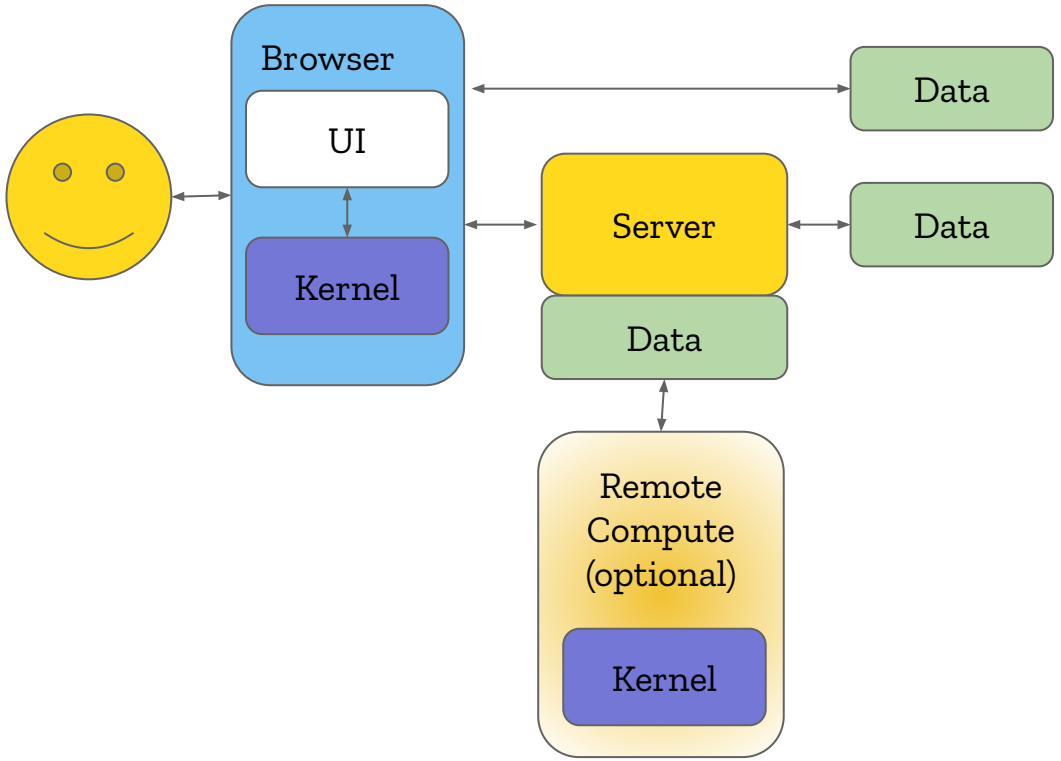

53
I-
Iodide-α
126.90

Architecture

Jupyter-like model



Iodide model



Adapted from: https://jupyter.readthedocs.io/en/latest/architecture/how_jupyter_ipython_work.html#notebooks

Iodide

coolwarm vs. viridis - an investigation

REPORT

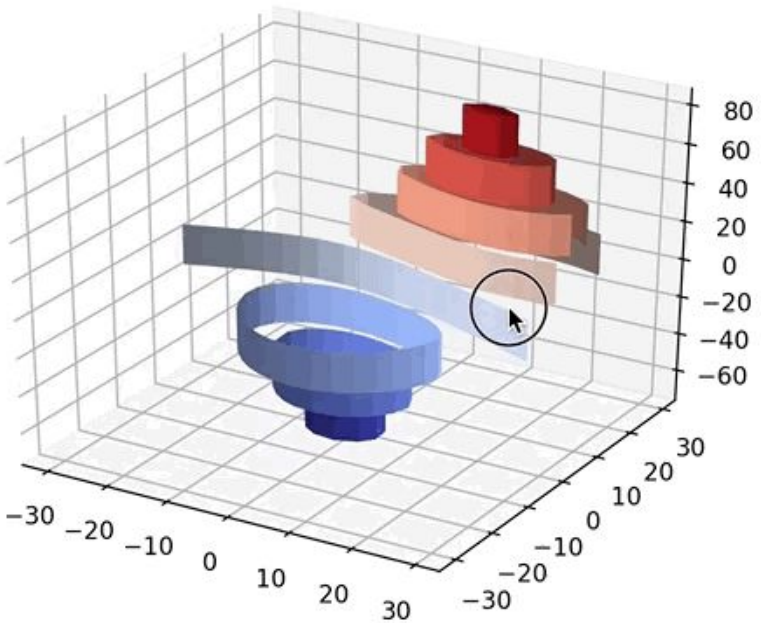
Editor

```
%% md

# `coolwarm` vs `viridis`?

%% py

from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import cm

fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y, Z = axes3d.get_test_data(0.07)
# change the cmap to cm.viridis!
cset = ax.contour(X, Y, Z, extend3d=True, cmap=cm.coolwarm)
ax.clabel(cset, fontsize=9, inline=1)
plt.show()

%% css

h1 {
  text-align:center;
}
```

Report Preview | Console | Workspace

coolwarm vs viridis?

Figure 1



PDF PNG SVG

azimuth=-58 deg, elevation=30 deg

iomd

```
%% md
# This is a markdown header

%% js
el = document.getElementById(“foo”)

%% py
from js import el
el.text = “Hello World!”
```

- Human readable and editable
- Easy for programs to support
- Diffable with standard tools
- See Matlab cell mode, R Markdown, Jupytext (and many others)

Javascript

PROS	CONS
FAST: Some of the best compiler technology of any dynamic language	Legacy “rough edges”
Familiar to many programmers	Not familiar to many data scientists
Large selection of user interface and visualization tools	Lacks a mature data science ecosystem

What if we could bring Python to the browser?

Transpiling

Convert Python to Javascript

Python

```python
def fib(n):
    if n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

Javascript

```javascript
export var fib = function(n) {
    if (n == 1) return 0;
    else if (n == 2) return 1;
    else return fib(n - 1) + fib(n - 2)
};
```

transcrypt, pyjs

Transpiling

Convert Python to Javascript

Pros

- Small
- Fast

Cons

- Server-side "ahead of time"
- Subtly different semantics
- Covering all of CPython's functionality is a lot of work
- Keeping up with CPython's progress is a lot of work
- No support for C extensions (Numpy, Scipy, etc.)

Interpreter Porting

Rewrite the Python interpreter and VM in Javascript

C

```c
static int
set_add_entry(
    PySetObject *so,
    PyObject *key,
    Py_hash_t hash
)
{
    while (1) {
        if (entry->hash == hash) {
            PyObject *startkey = entry->key;
            assert(startkey != dummy);
            if (startkey == key)
                goto found_active;
```

Javascript

```javascript
function $add(self, item){
    self.$items.push(item)
    var value = item.valueOf()
    if(typeof value == "number"){
        self.$numbers.push(value)
    }
```

brython, skulpt, batavia

Interpreter Porting

Rewrite Python interpreter and VM in Javascript

Pros

- Can compile and run Python entirely in the browser
- Can embed a transpiler in the browser for a hybrid approach

Cons

- Larger download and slower startup than transpiling
- Subtly different semantics
- Covering all of CPython's functionality is a lot of work
- Keeping up with CPython's progress is a lot of work
- No support for C extensions

WebAssembly

Compile to WebAssembly

Recompile the Python interpreter to WebAssembly

C

```c
static int
set_add_entry(
    PySetObject *so,
    PyObject *key,
    Py_hash_t hash
)
{
    while (1) {
        if (entry->hash == hash) {
            PyObject *startkey = entry->key;
            assert(startkey != dummy);
            if (startkey == key)
                goto found_active;
```

WebAssembly

```
(func (;1839;) (type 4) (param i32 i32 i32) (result
i32)
   (local i32 i32 i32 i32 i32 i32 i32 i32 i32 i32)
   ...
   if  ;; label = @1
     block  ;; label = @2
       block  ;; label = @3
         block  ;; label = @4
           loop  ;; label = @5
             block  ;; label = @6
               block (result i32)  ;; label = @7
                 block  ;; label = @8
```

PyPy.js, cpython-wasm, Pyodide

Compile to WebAssembly

Recompile the Python interpreter to WebAssembly

Pros

- It's the same as upstream CPython
- Everything that can work does work
- Supports C extensions (Numpy, Scipy etc.)
- Performance on par with native code

Cons

- Very large download sizes
- High memory usage

Tradeoffs

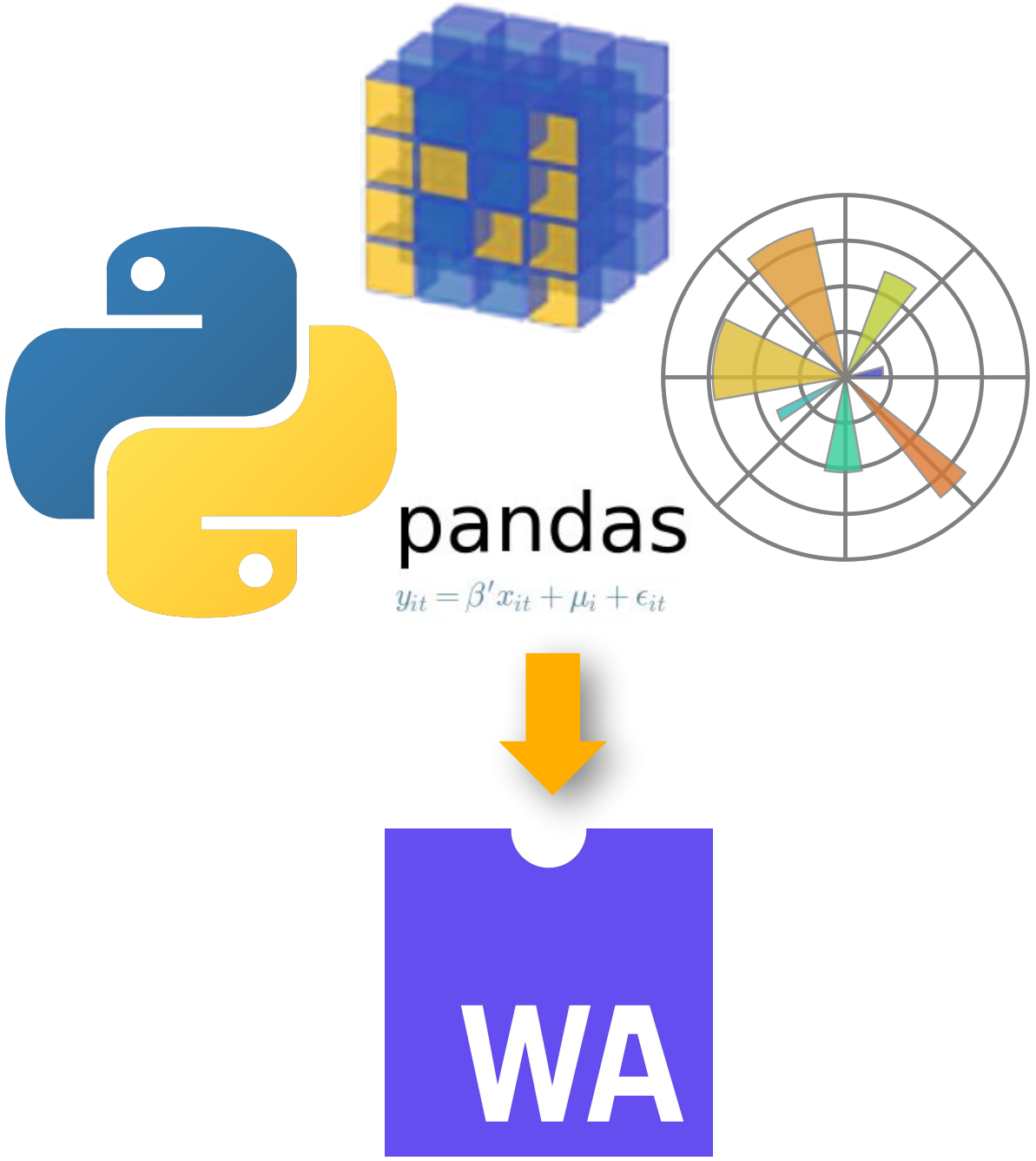
	Transpiling	Porting	Recompiling interpreter
Download size	Small	Medium	Large
Memory usage	Small	Medium	Large
Similarity to upstream	Low	Medium	High
Easily track upstream changes			✔️
Supports C extensions			✔️

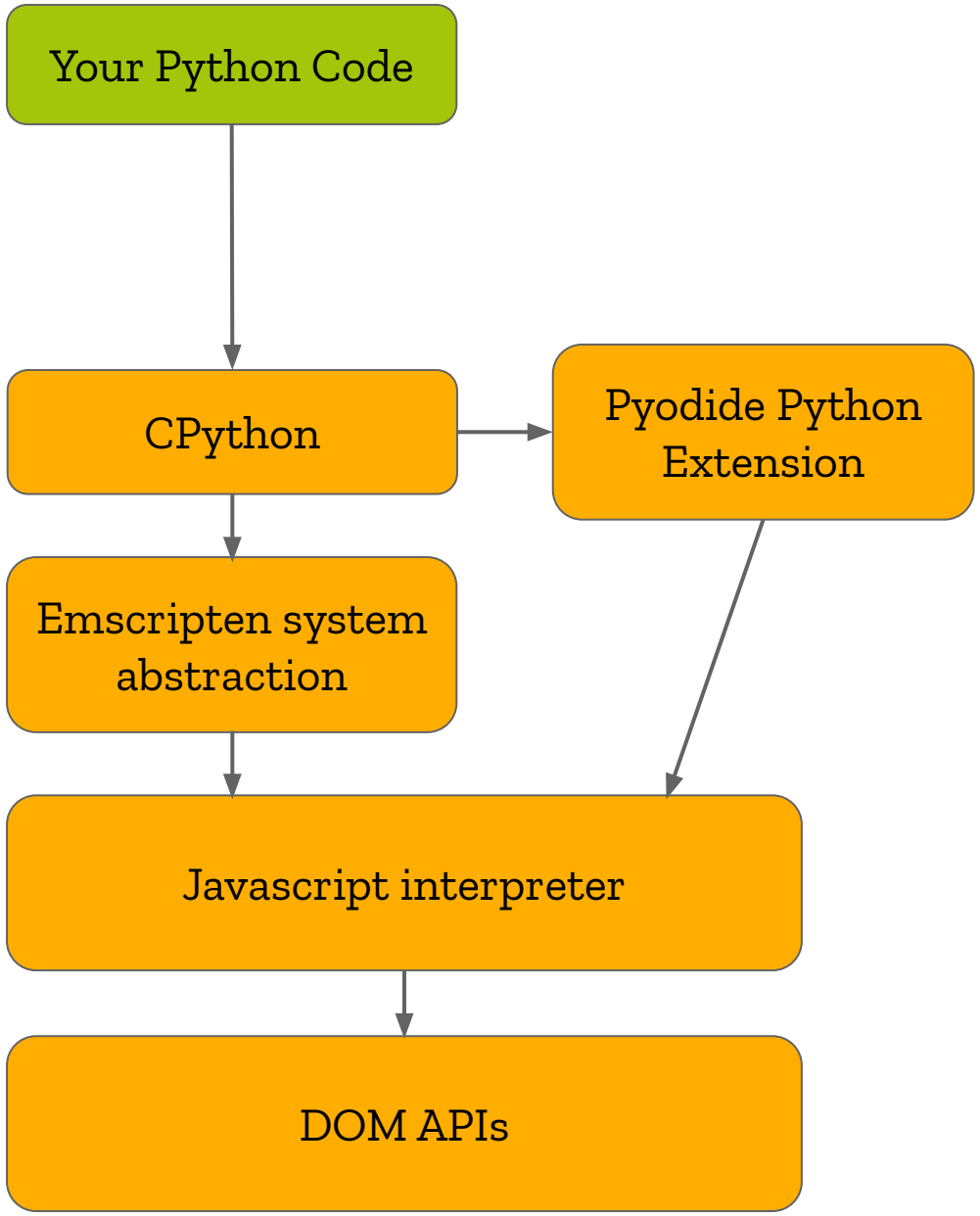
Pyodide

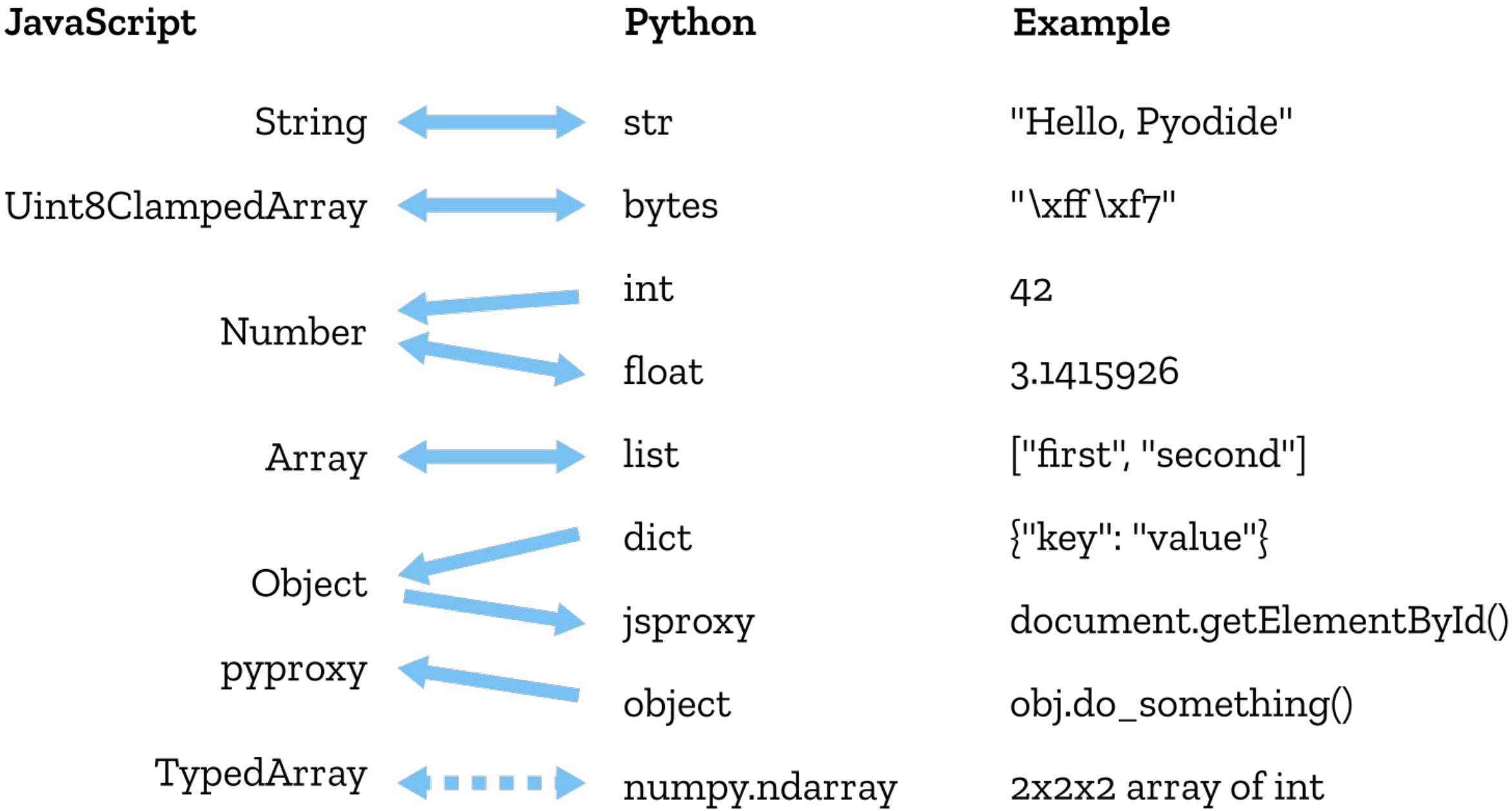
The scientific Python stack, compiled to WebAssembly

Pyodide

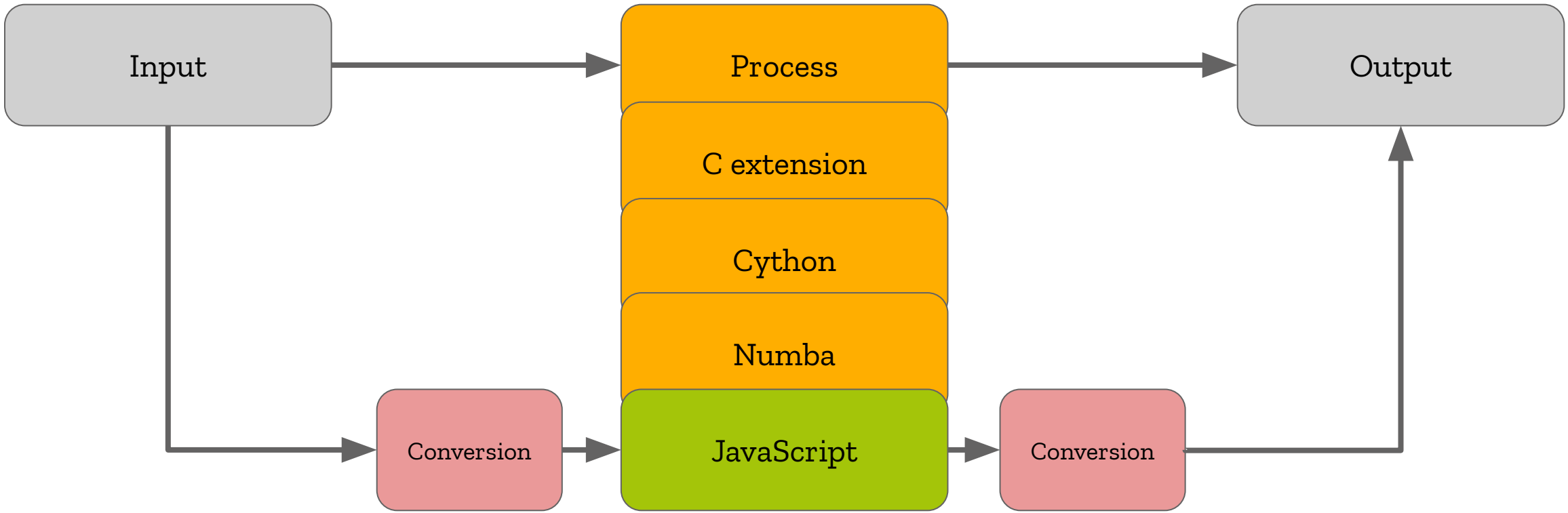
- Upstream CPython
- numpy, pandas, matplotlib, scipy
- "pip install" pure Python wheels



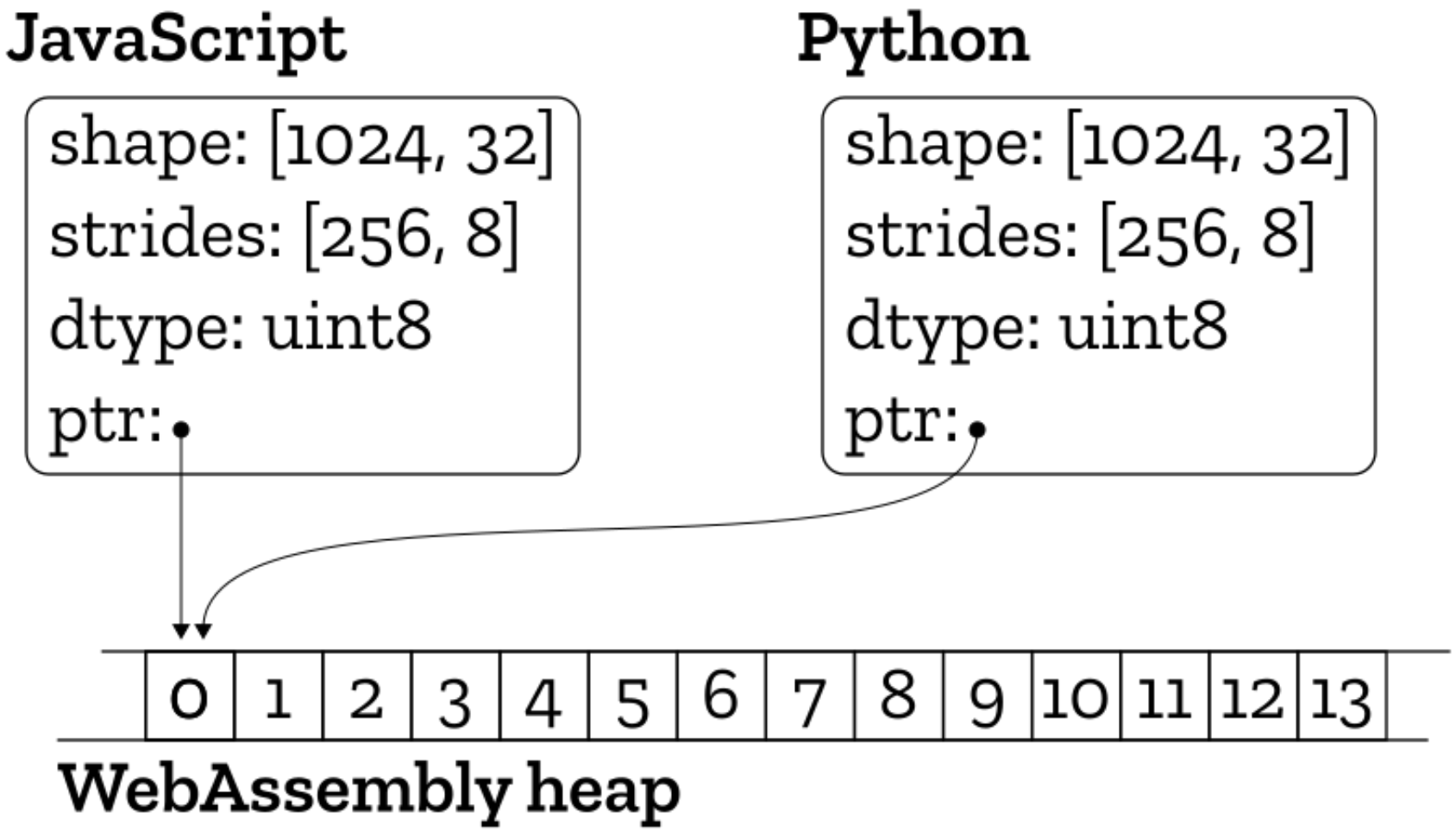

Your Python Code
CPython
Pyodide Python Extension
Emscripten system abstraction
Javascript interpreter
DOM APIs


JavaScript
Python
Example
String
str
"Hello, Pyodide"
Uint8ClampedArray
bytes
"\xff\xf7"
Number
int
42
float
3.1415926
Array
list
["first", "second"]
Object
dict
{"key": "value"}
jsproxy
document.getElementById()
pyproxy
object
obj.do_something()
TypedArray
numpy.ndarray
2x2x2 array of int

Accelerating Python



Sharing arrays with zero copying



Future?

APACHE ARROW

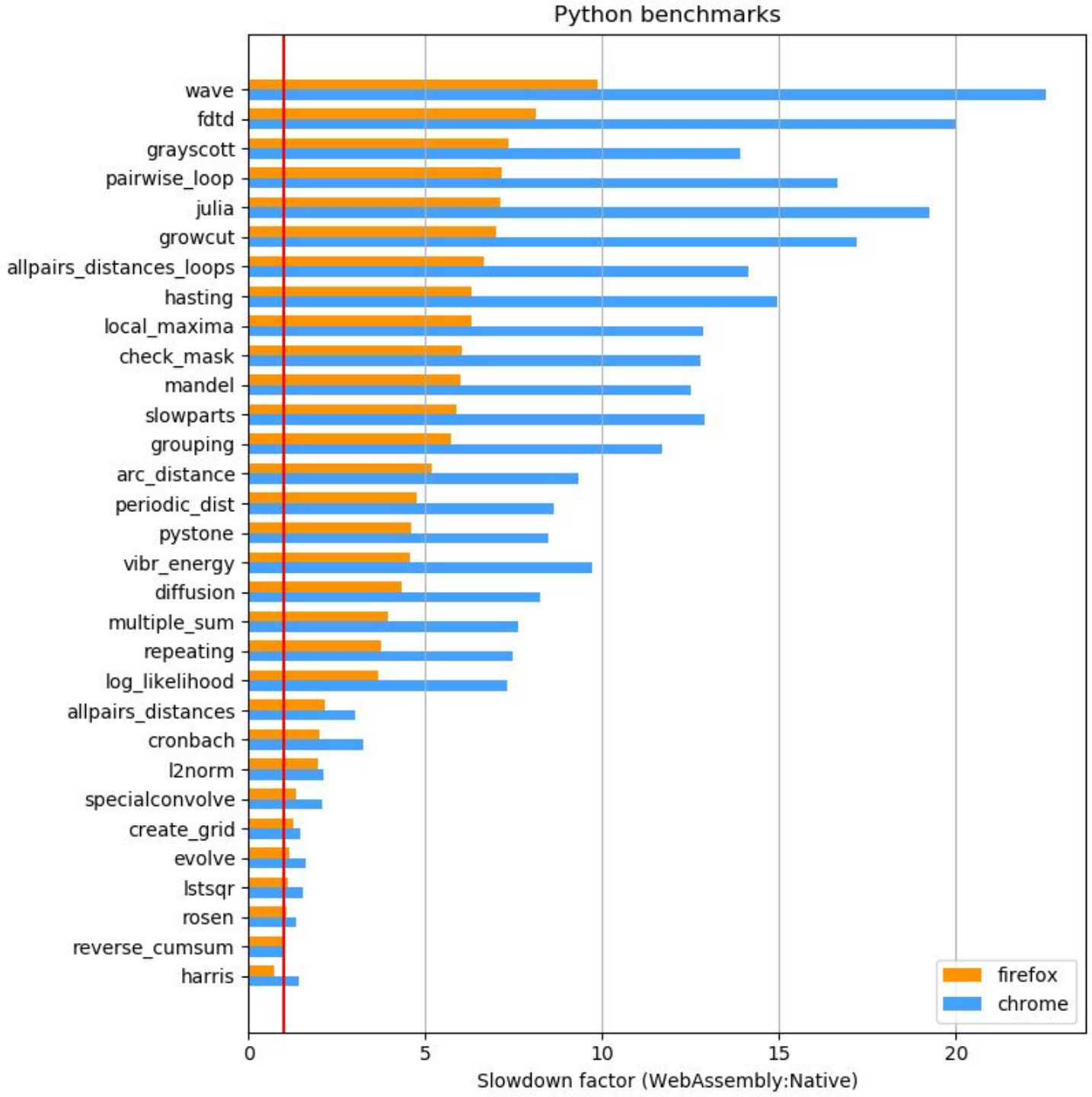
XND

The Web API

- DOM
- Graphics: Canvas, WebGL
- Audio: WebAudio, WebRTC
- Video: HTMLMediaElement
- Device: Notifications, WebBluetooth
- Storage: Client-side storage

Pyodide Demo

Performance



https://github.com/serge-sans-paille/numpy-benchmarks

Ways to get more performance

- Cython
- Numba
- PyPy
- Apache Arrow
- General purpose GPU
- Distributed computing

What doesn't work

Probably never

- Raw network sockets
- Subprocesses
- Access to the host filesystem

Someday

- threads
- async
- SIMD
- General Purpose GPU computing

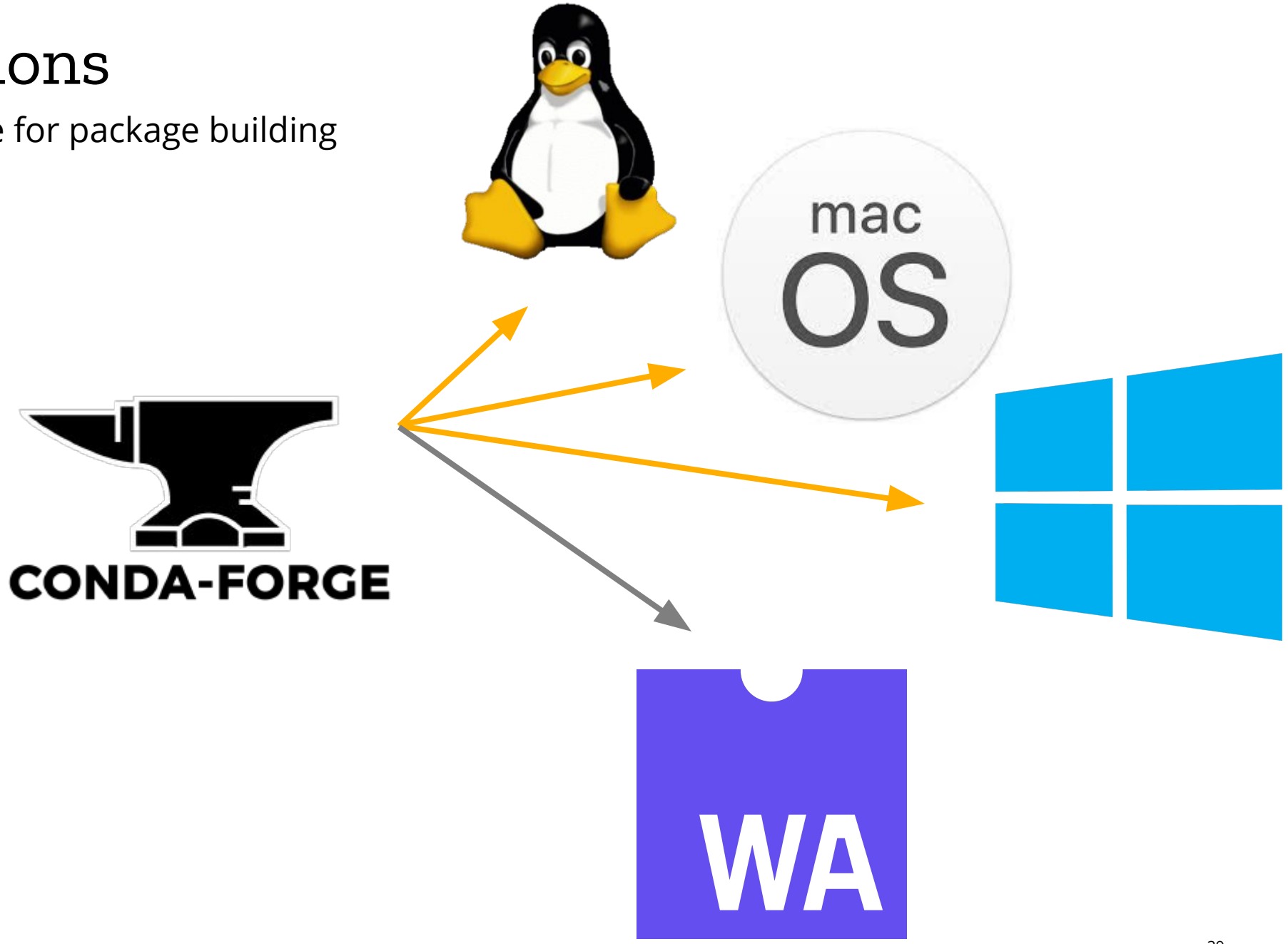
Monolithic Libraries

package	Total size	Loaded at import
Scipy	65MB	11MB
Pandas	50MB	43MB
Matplotlib	20MB	13MB
Numpy	20MB	11MB

* values are for native x86_64 Python

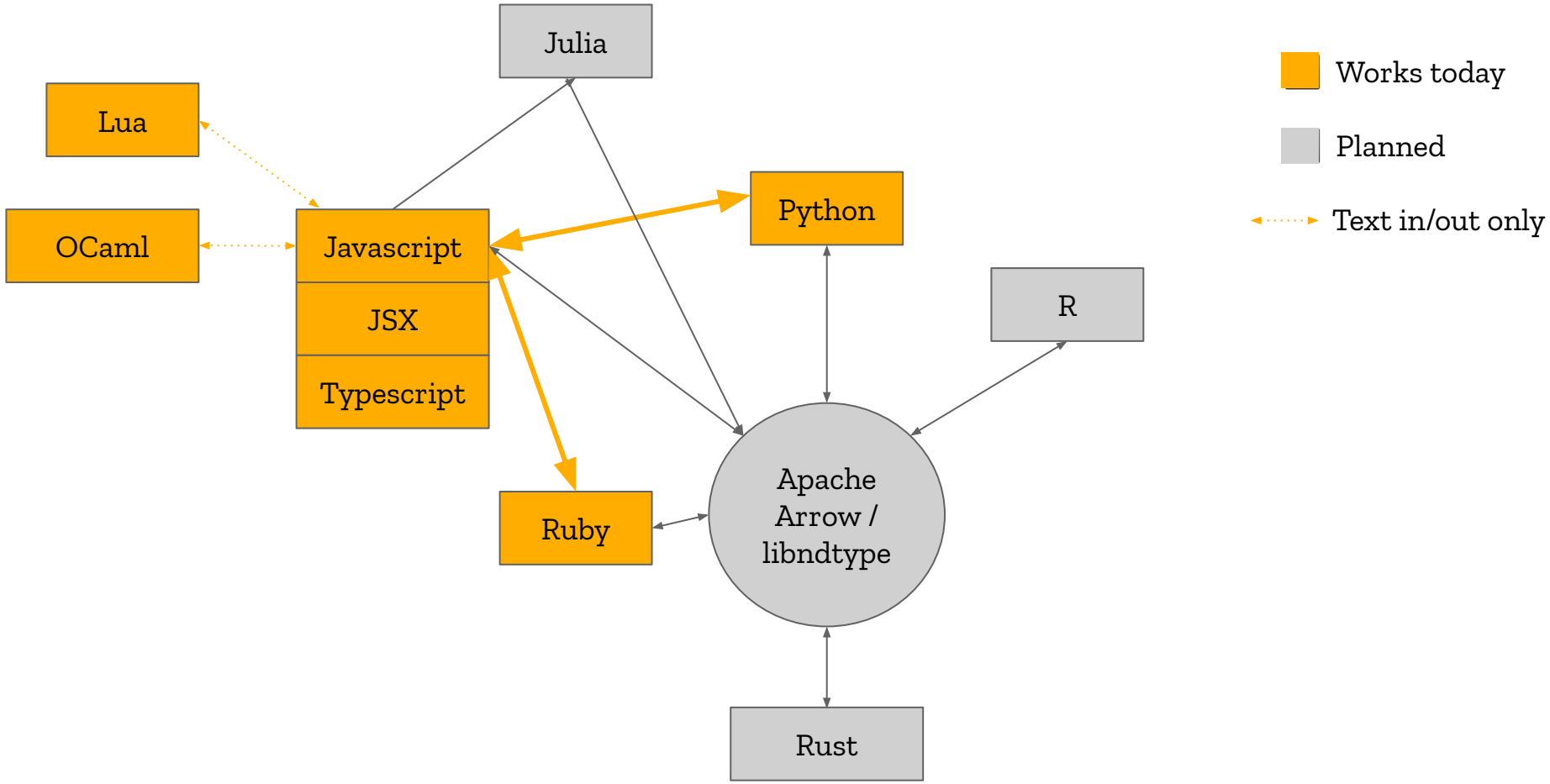
Future Directions

conda forge infrastructure for package building



Future directions

Language interoperability



Come build with us!

We're open source on github

http://github.com/iodide-project/

We need:

- Experimenters
- Designers
- Programmers
- Writers
- Bug hunters

Our team

Brendan Colloran
Hamilton Ulmer
William Lachance
Michael Droettboom
Teon Brooks
John Karahalis
Rob Miller
Jannis Leidel
...

Devin Bayly

Dhiraj Barnwal

Roman Yurchak
Kirill Smelkov

Madhur Tandon

...and many other community contributors

Check us out at:

iodide.io

github.com/iodide-project

mdroettboom@mozilla.com