PYTHON

Tamás Gál (Tom)

tamas.gal@fau.de

Erlangen Centre for Astroparticle Physics (ECAP)

Live poll during the talk:

http://etc.ch/iyfS

tamasgal.com https://github.com/tamasgal @tamasgal

OVERVIEW

- Python Introduction
- Basic Python Internals
- Libraries and Tools for Scientific Computing

WHO IS THIS CLOWN?

- **Tamás Gál**, born 1985 in Debrecen (Hungary)
- **Astroparticle physicist** at the Erlangen Centre for Astroparticle Physics (**ECAP**) working on the **KM3NeT neutrino detector** experiment and **open science/data**
- **Sysadmin** (DevOps) at **ECAP** (including the ECAP and KM3NeT IT services)
- **Programming** background:
  - Coding enthusiast since ~**1993**
  - First real application written in Amiga Basic (toilet manager, tons of GOTOs ;)
  - Mostly **Python**, **Julia**, JavaScript and C/C++/Obj-C for **work**
  - **Haskell** for **fun**
  - Earlier also Java, Perl, PHP, Delphi, MATLAB, whatsoever…
- Editor: **Vim for ~23 years** and switched to **Emacs** in 2020
- Other: ADV motorbikes, climbing, electronics, modular synths, DIY...
- Find me on: **tamasgal.com** **twitter.com/tamasgal** **github.com/tamasgal**

 Live poll: http://etc.ch/iyfS

PYTHON

PYTHON PROGRAMMING LANGUAGE

- Interpreted high-level general-purpose programming language
- **Object-oriented**, procedural (imperative), functional, structured, reflective
- **Dynamically-typed** and **garbage-collected**
- Designed by **Guide van Rossum**
- "**batteries included**"
- Tries to **avoid premature optimisation**: move time-critical functions to extension modules written in "faster" languages (like C or Fortran)

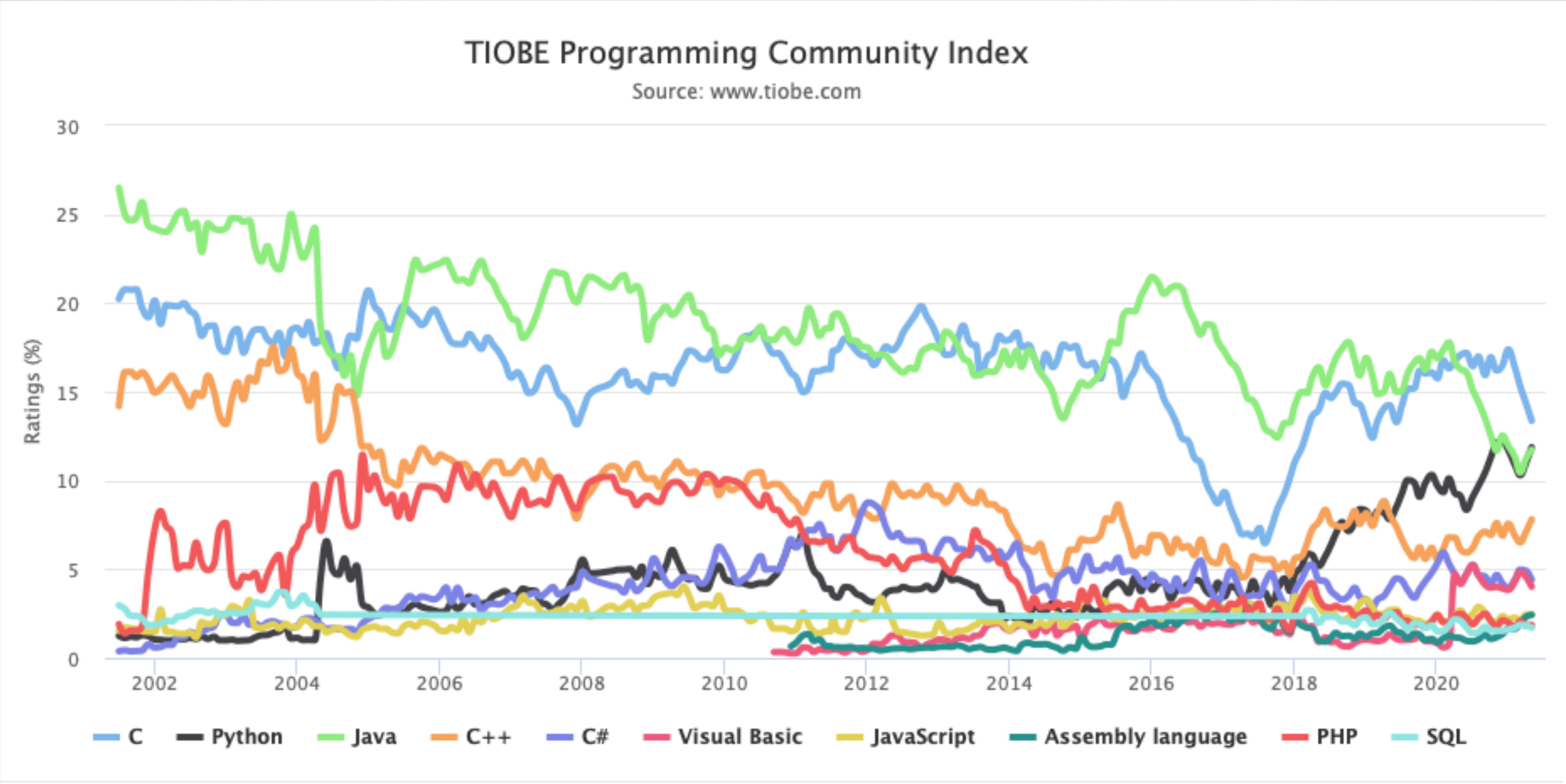
BRIEF HISTORY OF PYTHON

- Rough idea in the **late 1980s**
- Meant to descend the **ABC language** (origin of ideas)
- Python should **fill** the **gap** between **C** and **Shell** scripts
- First line of code in **December 1989** by **Guido van Rossum**
- After one year of development, it was released as "open source"
- Python **2.0** in October **2000**
- Python **3.0** in December **2008**
- July 2018: Guido van Rossum resigned from his **BDFL** (Benevolent Dictator for Life) position
- Python **2.7** End Of Life date: **2020**
- **Current stable release: 3.9.5**

THE ZEN OF PYTHON

```
>>> import this
The Zen of Python, by Tim Peters
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those
```

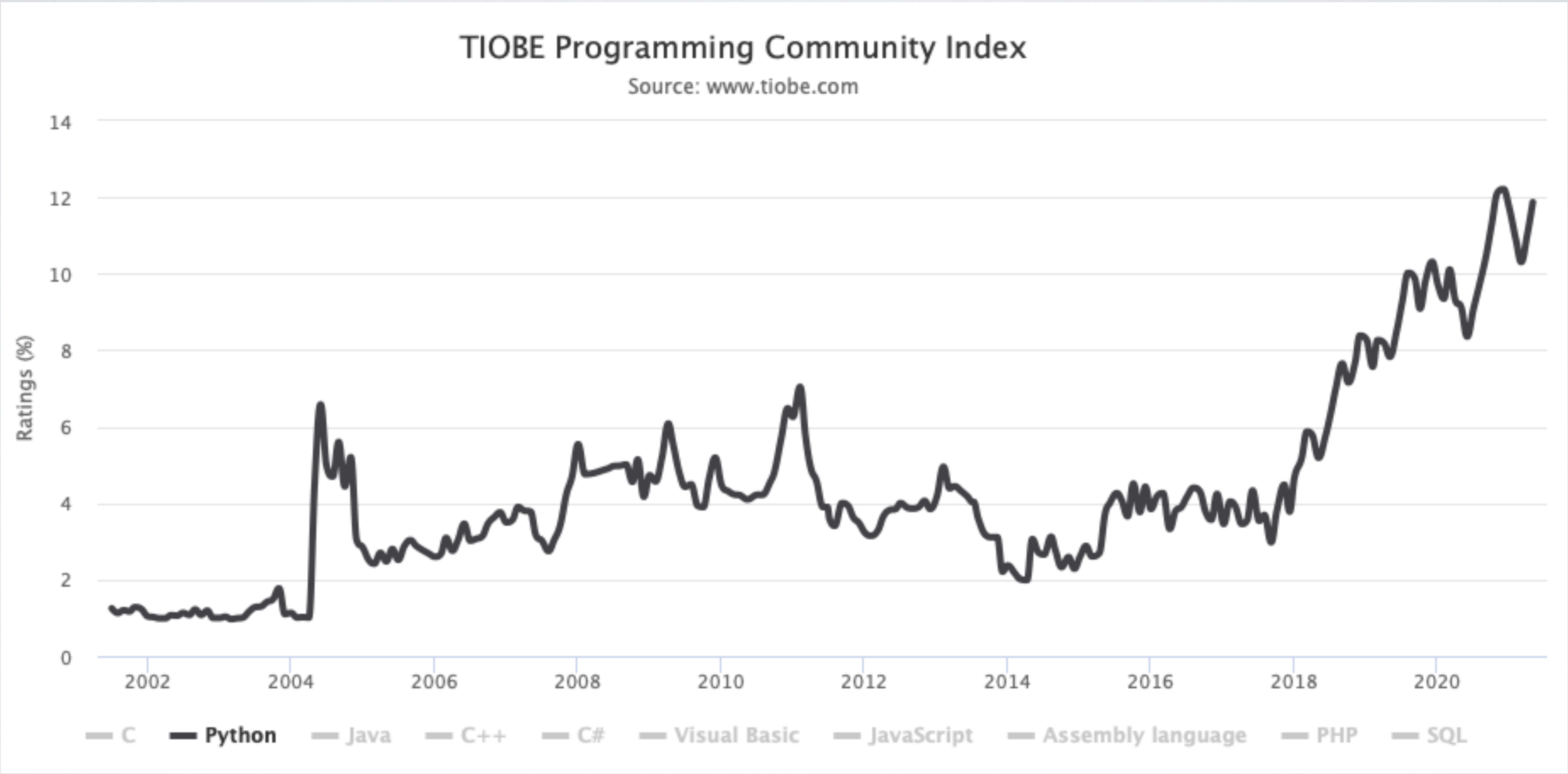
POPULAR LANGUAGES



Python now the second most popular language (according to TIOBE)!
... and has beaten Java and C++ ;)

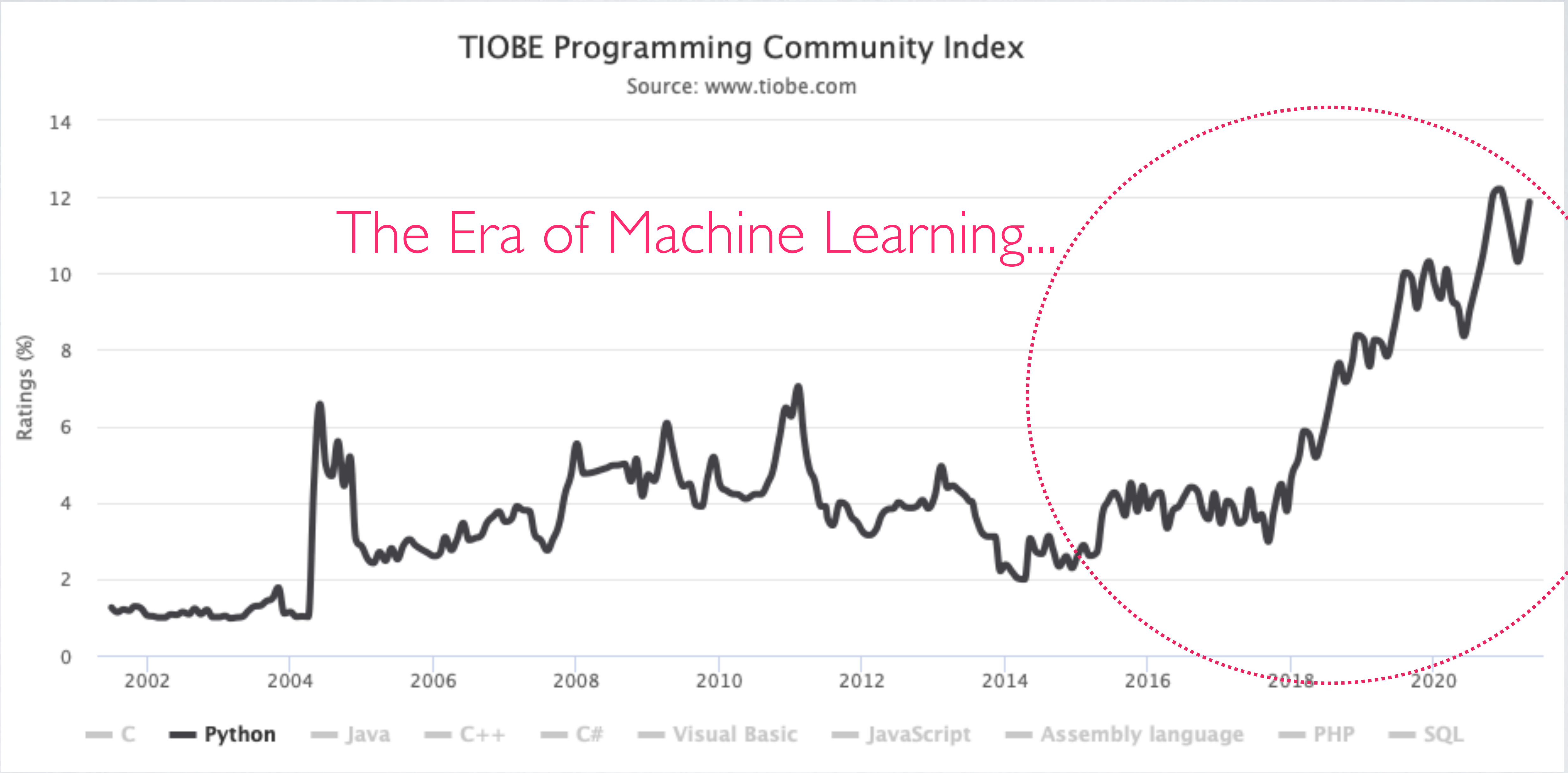
Julia is #30!

PYTHONS POPULARITY



“Programming language of the year” in 2007, 2010, 2018 and 2020.

PYTHONS POPULARITY



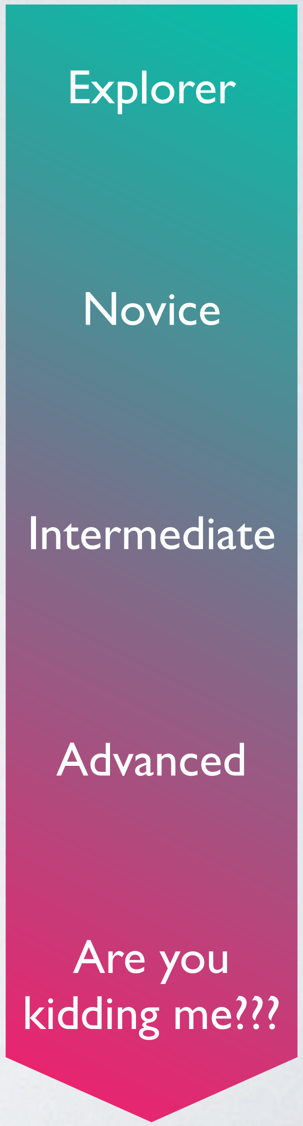
"Programming language of the year" in 2007, 2010, 2018 and 2020.

YOUR JOURNEY THROUGH PYTHON?

(JUST A VERY ROUGH GUESS, NOT A MEAN GAME)

- Have you ever launched the Python interpreter?
- Wrote for/while-loops or if/else statements?
- …your own functions?
- …classes?
- …list/dict/set comprehensions?
- Do you know what a generator is?
- Have you ever implemented a decorator?
- …a metaclass?
- …a C-extension?
- Do you know and can you explain the output of the following line for Python?

```python
print(5 is 7 - 2, 300 is 302 - 2)
```



ANSWER TO

`print(5 is 7 - 2, 300 is 302 - 2)`

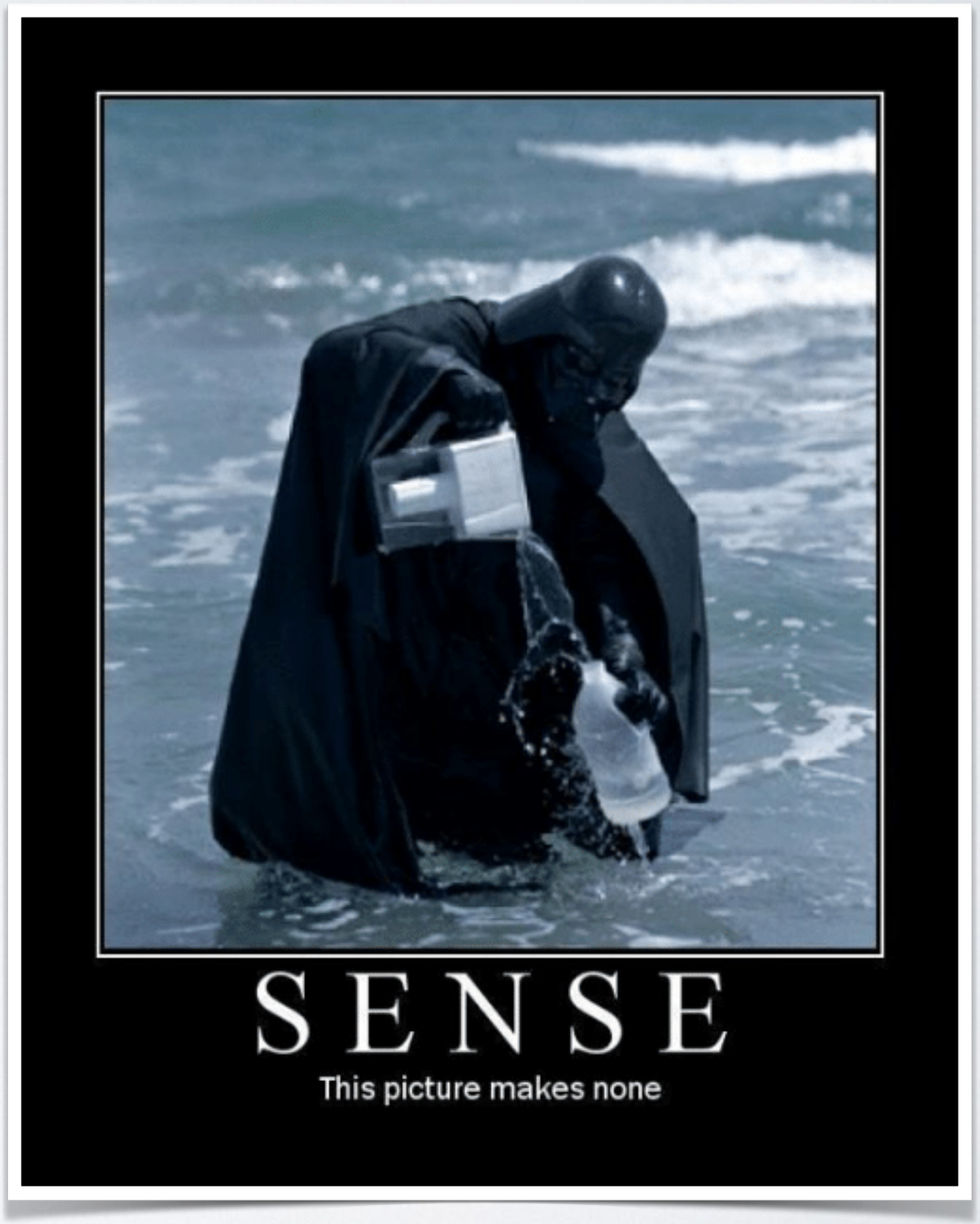
Python 2.7: True, False

Python 3.6: True, False

Python 3.7: True, True

Python 3.8: True, True, and warnings…

Python 3.9: True, True, and warnings…



EXPLANATION OF

`print(5 is 7 - 2, 300 is 302 - 2)`

PyObject* PyLong_FromLong(long v)
Return value: New reference.
Return a new **PyLongObject** object from v, or NULL on failure.

The current implementation keeps an array of integer objects for all integers between -5 and 256, when you create an int in that range you actually just get back a refe should be possible to change of Python in this case is unc

cpython/Include/internal/**pycore_interp.h**

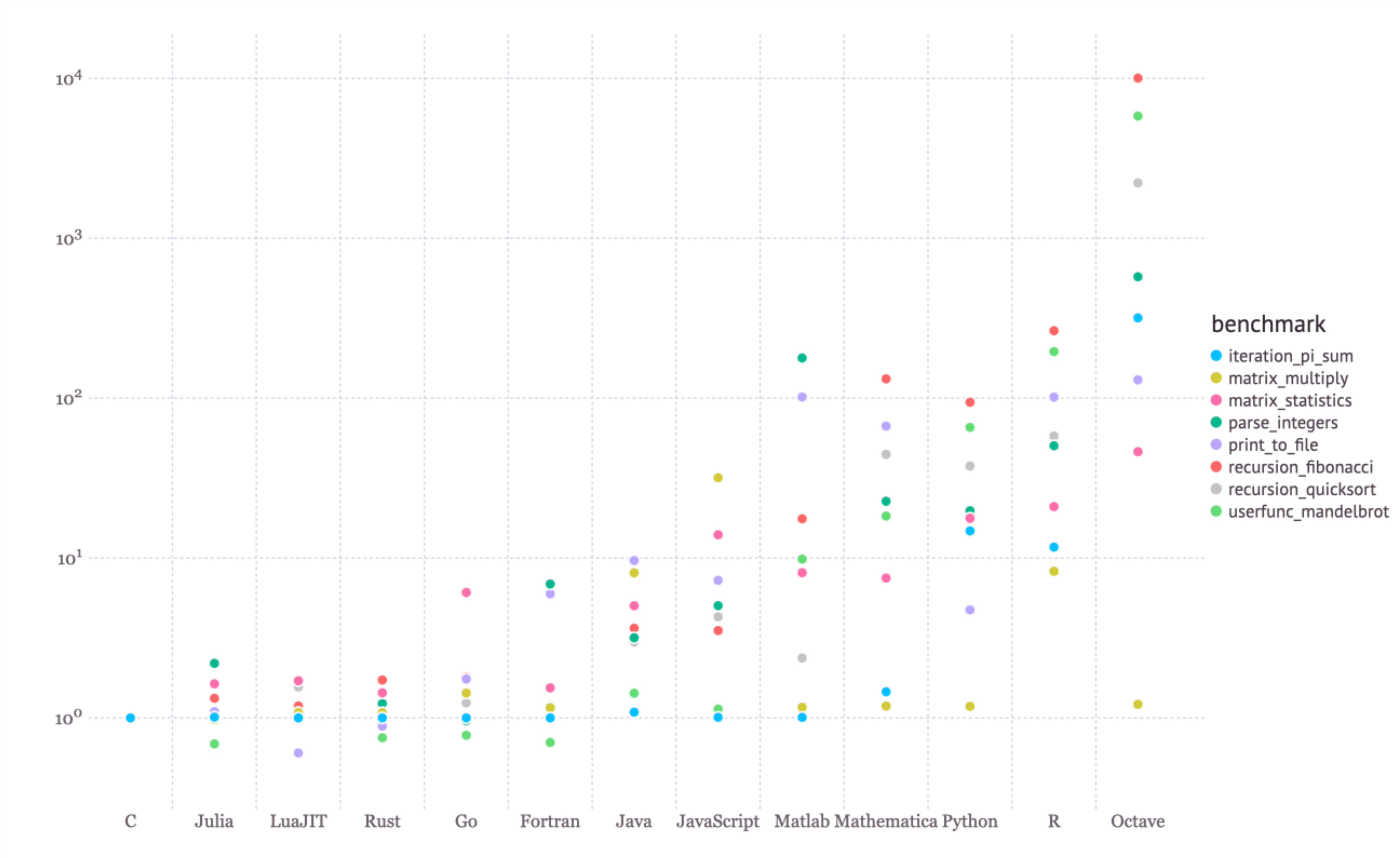
```
/* Small integers are preallocated in this array so that they
   can be shared.
   The integers that are preallocated are those in the range
   -_PY_NSMALLNEGINTS (inclusive) to _PY_NSMALLPOSINTS (not inclusive).
*/
PyLongObject* small_ints[_PY_NSMALLNEGINTS + _PY_NSMALLPOSINTS];
```

In Python 3.7 the constant folding is moved from the peephole optimiser to the new AST optimiser, which effectively avoids the extra allocation. (https://github.com/python/cpython/commit/7ea143ae795a9fd57eaccf490d316bdc13ee9065)

WHY IS PYTHON SO POPULAR (FOR SCIENCE)?

- Ease of use
- Interactive
- Lots of scientific libraries (and machine learning is everywhere)
- Tons of (built-in) useful supplementary functionalities
- General purpose language so that scientists can focus on a single language to rule them all…
- **… can they?**

PERFORMANCE OF LANGUAGES

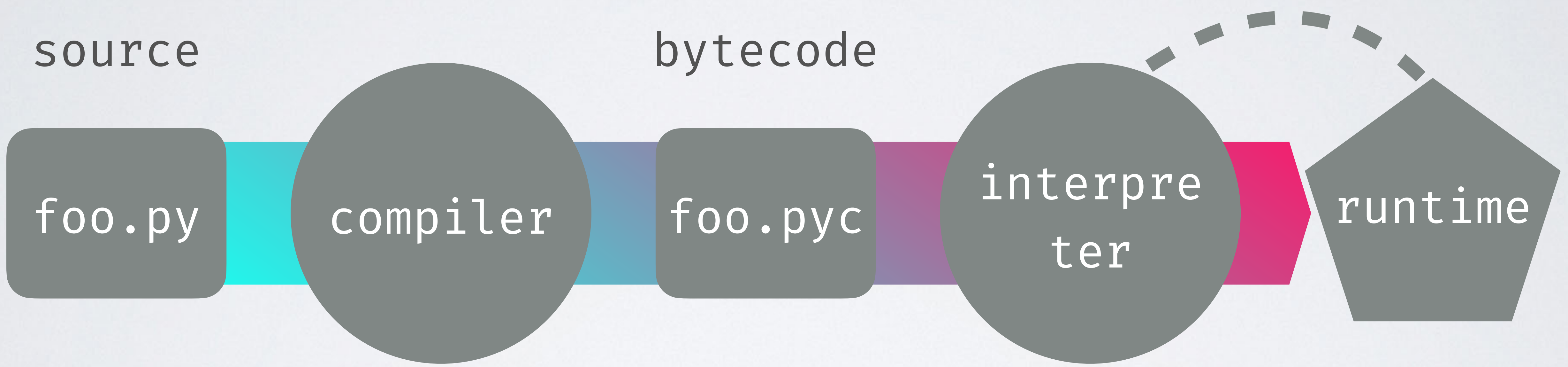


Microbenchmarks from https://julialang.org/benchmarks/

BASIC PYTHON INTERNALS

to understand the performance issues

FROM SOURCE TO RUNTIME



PYTHON BYTECODE

Python 3.6

```
import dis; dis.dis(compile('print(5 is 7 - 2, 300 is 302 - 2)', '', 'single'))
  1           0 LOAD_NAME                0 (print)
              2 LOAD_CONST               0 (5)
              4 LOAD_CONST               6 (5)
              6 COMPARE_OP               8 (is)
              8 LOAD_CONST               3 (300)
             10 LOAD_CONST               7 (300)
             12 COMPARE_OP               8 (is)
             14 CALL_FUNCTION            2
             16 PRINT_EXPR
             18 LOAD_CONST               5 (None)
             20 RETURN_VALUE
```

co_consts[i]

Python 3.7

```
import dis; dis.dis(compile('print(5 is 7 - 2, 300 is 302 - 2)', '', 'single'))
  1           0 LOAD_NAME                0 (print)
              2 LOAD_CONST               0 (5)
              4 LOAD_CONST               0 (5)
              6 COMPARE_OP               8 (is)
              8 LOAD_CONST               1 (300)
             10 LOAD_CONST               1 (300)
             12 COMPARE_OP               8 (is)
             14 CALL_FUNCTION            2
             16 PRINT_EXPR
             18 LOAD_CONST               2 (None)
             20 RETURN_VALUE
```

co_consts[i]

THE TYPE OF A PyObject

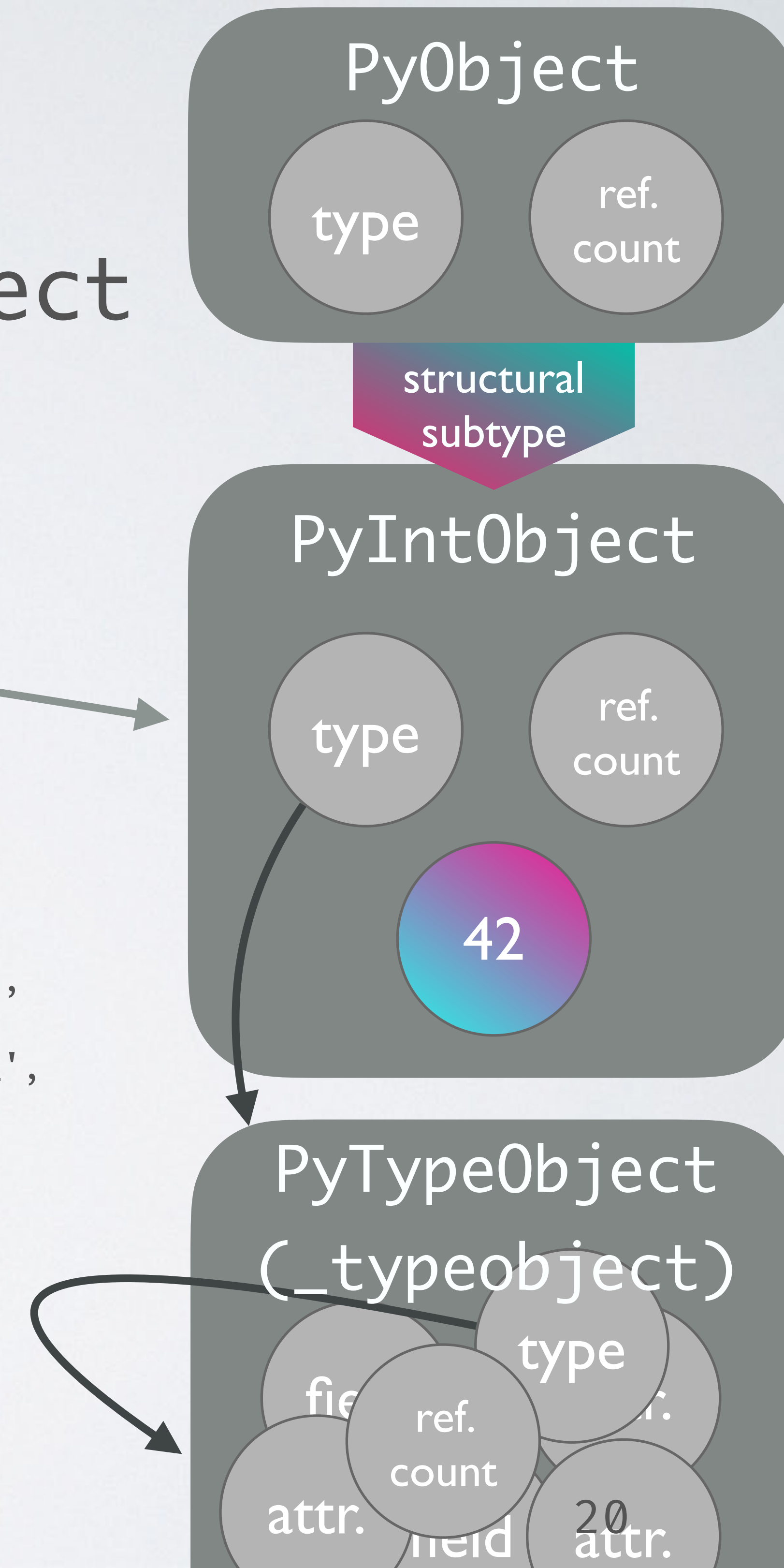
“An object has a ‘type’ that determines what it represents and what kind of data it contains. An object’s type is fixed when it is created. Types themselves are represented as objects. The type itself has a type pointer pointing to the object representing the type ‘type’, which contains a pointer to itself!”

— object.h

DATA IN PYTHON

- Every piece of data is a PyObject

```
>>> dir(42)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__',
'__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getnewargs__', '__gt__', '__hash__', '__index__', '__init__',
'__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__',
'__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__',
'__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__',
'__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__',
'__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__',
'__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__',
'__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'bit_length',
'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real',
'to_bytes']
```



YOUR BEST FRIEND AND WORST ENEMY:

GIL - Global Interpreter Lock

- The **GIL** prevents parallel execution of (Python) bytecode
- Even though Python has **real threads**, they never execute code at the same time
- **Context switching** between threads creates overhead (the user cannot control thread-priority)
- Threads perform pretty bad on CPU bound tasks
- They do a great job speeding up I/O heavy tasks

THREADS AND CPU BOUND TASKS

single thread:

```
N = 100000000
```

```
def count(n):
    while n != 0: n -=1
```

```
%time count(N)
```

```
CPU times: user 5.59 s, sys: 32.5 ms, total: 5.62 s
Wall time: 7.71 s
```

two threads:

```
from threading import Thread
```

```
def count_threaded(n):
    t1 = Thread(target=count, args=(N/2,))
    t2 = Thread(target=count, args=(N/2,))
    t1.start()
    t2.start()
    t1.join()
    t2.join()
```

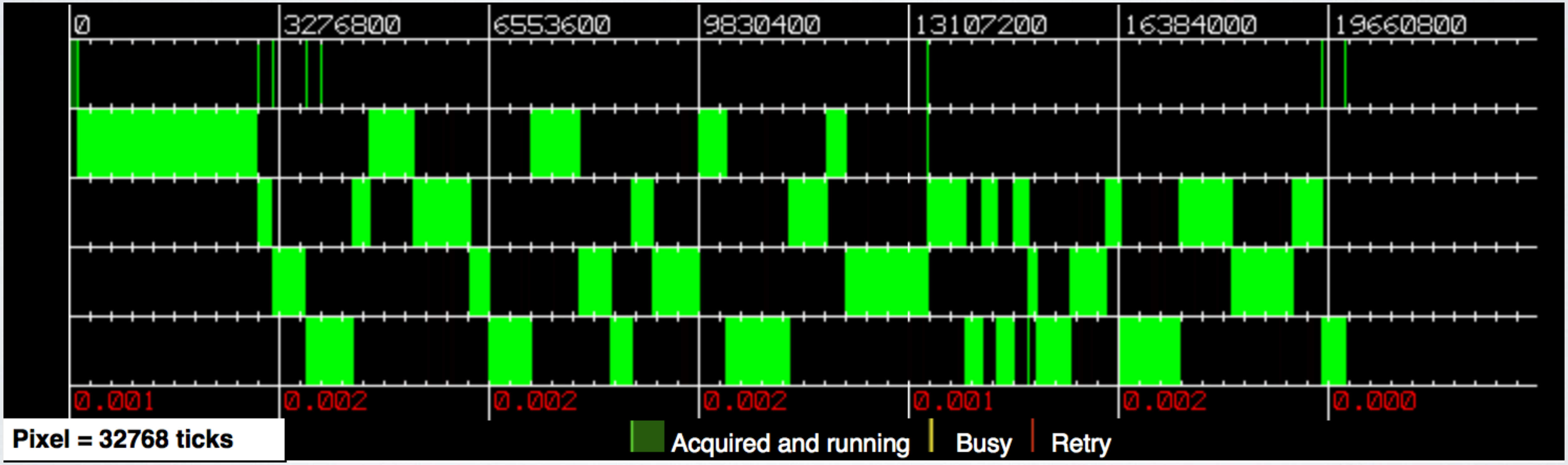
```
%time count_threaded(N)
```

```
CPU times: user 7.18 s, sys: 31 ms, total: 7.21 s
Wall time: 9.01 s
```

This is probably not really what you expected...

THREADS FIGHTING FOR THE GIL

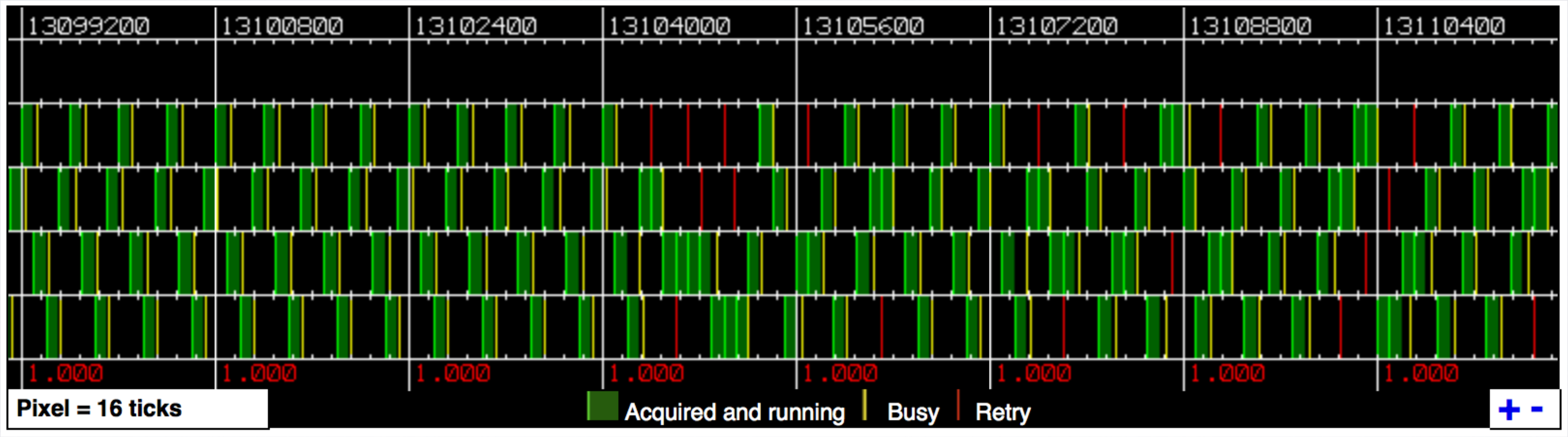
OS X: 4 threads on 1 CPU (Python 2.6)



By David M Beazley: http://dabeaz.com/GIL/gilvis

THREADS FIGHTING FOR THE GIL

OS X: 4 threads on 4 CPUs (Python 2.6)



By David M Beazley: http://dabeaz.com/GIL/gilvis

OK, huge overhead for every single object, no real parallel execution of code…

How should Python ever compete with all those super fast C/Fortran libraries?

C-extensions and interfacing C/C++/Fortran!

Those can release the GIL and do the heavy stuff in the background.

A DUMB SPEED COMPARISON

CALCULATING THE MEAN OF 1000000 NUMBERS

pure Python:

```python
def mean(numbers):
    return sum(numbers)/len(numbers)
```

```python
numbers = list(range(1000000))
%timeit mean(numbers)
```

```
8.59 ms ± 234 µs per loop
```

Numba (~8x faster):

```python
@nb.jit
def numba_mean(numbers):
    s = 0
    N = len(numbers)
    for i in range(N):
        s += numbers[i]
    return s/N
```

```python
numbers = np.random.random(1000000)
%timeit numba_mean(numbers)
```

```
1.1 ms ± 6.64 µs per loop
```

NumPy (~13x faster):

```python
numbers = np.random.random(1000000)
%timeit np.mean(numbers)
```

```
638 µs ± 38.3 µs per loop
```

Julia (~16x faster):

```julia
numbers = rand(1000000)
@benchmark mean(numbers)
```

```
BenchmarkTools.Trial:
  memory estimate:  16 bytes
  allocs estimate:  1
  --------------
  minimum time:     464.824 µs (0.00% GC)
  median time:      524.386 µs (0.00% GC)
  mean time:        544.573 µs (0.00% GC)
  maximum time:     2.095 ms (0.00% GC)
  --------------
  samples:          8603
  evals/sample:     1
```

CRAZY LLVM COMPILER OPTIMISATIONS

SUMMING UP NUMBERS FROM 0 TO N=100,000,000

pure Python:

```
def simple_sum(N):
    s = 0
    for i in range(N):
        s += i
    return s
```

```
%time simple_sum(N)
```

```
CPU times: user 7.13 s, sys: 103 ms, total: 7.23 s
Wall time: 7.43 s

4999999950000000
```

NumPy (~80x faster):

```
np_numbers = np.array(range(N))

%time np.sum(np_numbers)
```

```
CPU times: user 84 ms, sys: 2.65 ms, total: 86.6 ms
Wall time: 91.1 ms

4999999950000000
```

Numba (~300000x faster):

```
@nb.jit
def simple_sum(N):
    s = 0
    for i in range(N):
        s += i
    return s
```

```
%time numba_sum(N)
```

```
CPU times: user 11 µs, sys: 3 µs, total: 14 µs
Wall time: 21.9 µs

4999999950000000
```

Julia (~7000000x faster):

```
function simple_sum(N)
    s = 0
    for i ∈ 1:N
        s += i
    end
    return s
end
```

```
simple_sum (generic fu
```

```
@time simple_sum(N)
```

```
0.000002 seconds (5

4999999950000000
```

```
        pushq   %rbp
        movq    %rsp, %rbp
        xorl    %eax, %eax
Source line: 3
        testq   %rdi, %rdi
        jle     L32
        leaq    -1(%rdi), %rax
        leaq    -2(%rdi), %rcx
        mulq    %rcx
        shldq   $63, %rax, %rdx
        leaq    -1(%rdx,%rdi,2), %rax
Source line: 6
L32:
        popq    %rbp
        retq
        nopw    %cs:(%rax,%rax)
```

 Live poll: http://etc.ch/iyfS

PYTHON LIBRARIES

for scientific computing

NUMFOCUS

OPEN CODE = BETTER SCIENCE

pandas

NumPy

Matplotlib

SymPy

IPython

Jupyter

AstroPy

SciPy

Not part of NumFocus but covered in this talk:

Numba

Numexpr

AwkwardArray

NUMFOCUS
OPEN CODE = BETTER SCIENCE

SCIPY

Scientific Computing Tools for Python

THE SCIPY STACK

- Core packages
  - SciPy Library: numerical algorithms, signal processing, optimisation, statistics etc.
  - NumPy
  - Matplotlib: 2D/3D plotting library
  - pandas: high performance, easy to use data structures
  - SymPy: symbolic mathematics and computer algebra
  - IPython: a rich interactive interface to process data and test ideas
  - Jupyter: notebooks to document and code at the same time
  - nose: testing framework for Python code
- Other packages:
  - Chaco, Mayavi, Cython, Scikits (scikit-learn, scikit-image), h5py, PyTables and much more

https://www.scipy.org

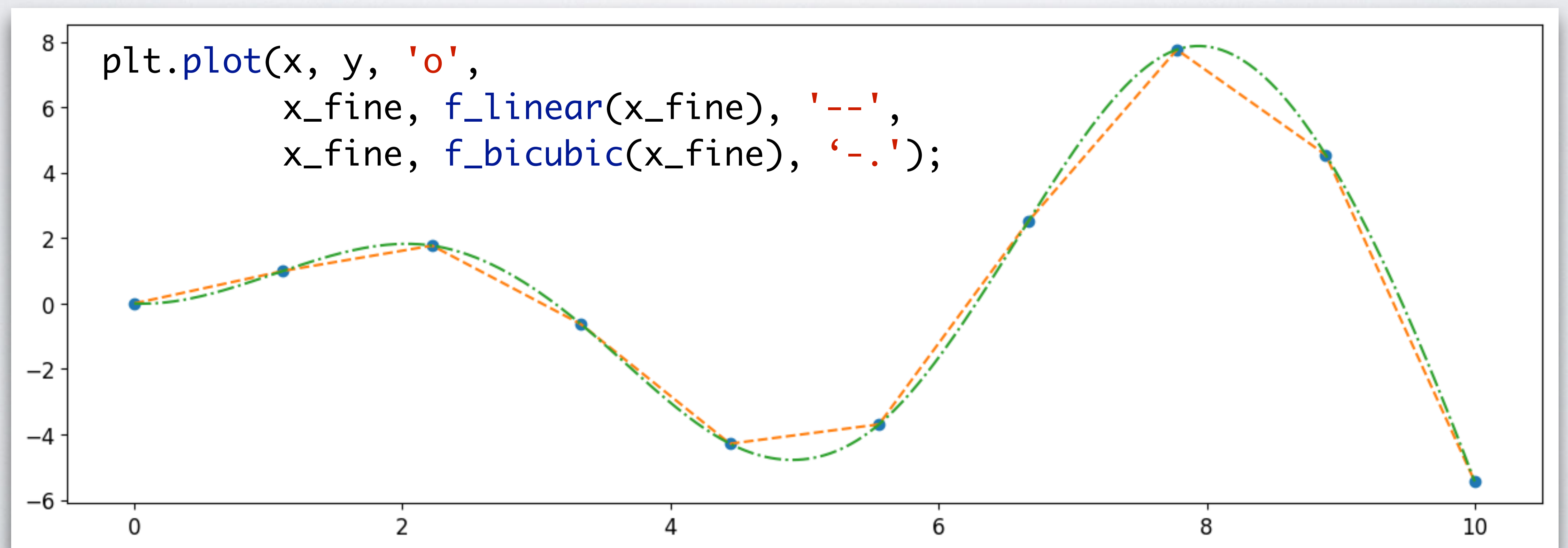
SCIPY CORE LIBRARY

- Clustering package (scipy.cluster)
- Constants (scipy.constants)
- Discrete Fourier transforms (scipy.fftpack)
- Integration and ODEs (scipy.integrate)
- Interpolation (scipy.interpolate)
- Input and output (scipy.io)
- Linear algebra (scipy.linalg)
- Miscellaneous routines (scipy.misc)
- Multi-dimensional image processing (scipy.ndimage)
- Orthogonal distance regression (scipy.odr)
- Optimization and root finding (scipy.optimize)
- Signal processing (scipy.signal)
- Sparse matrices (scipy.sparse)
- Sparse linear algebra (scipy.sparse.linalg)
- Compressed Sparse Graph Routines (scipy.sparse.csgraph)
- Spatial algorithms and data structures (scipy.spatial)
- Special functions (scipy.special)
- Statistical functions (scipy.stats)
- Statistical functions for masked arrays (scipy.stats.mstats)

SCIPY INTERPOLATE

```python
from scipy import interpolate

x = np.linspace(0, 10, 10)
y = np.sin(x)

x_fine = np.linspace(0, 10, 500)

f_linear = interpolate.interp1d(x, y, kind='linear')
f_bicubic = interpolate.interp1d(x, y, kind='cubic')
```

```python
plt.plot(x, y, 'o',
         x_fine, f_linear(x_fine), '--',
         x_fine, f_bicubic(x_fine), ‘-.');
```



NUMFOCUS
OPEN CODE = BETTER SCIENCE

NUMPY

Numerical Python

NUMPY

NumPy is the fundamental package for scientific computing with Python.

- gives us a powerful N-dimensional array object: *ndarray*
- broadcasting functions
- tools for integrating C/C++ and Fortran
- linear algebra, Fourier transform and random number capabilities
- most of the scientific libraries build upon NumPy

NUMPY: ndarray

```
a = np.arange(6)
a
```

```
array([0, 1, 2, 3, 4, 5])
```

ndim: 1
shape: (6,)

Contiguous array in memory with a fixed type,
no pointer madness!

C/Fortran compatible memory layout,
so they can be passed to those
without any further efforts.

NUMPY: ARRAY OPERATIONS AND ufuncs

```
a * 23
```

```
array([  0,  23,  46,  69,  92, 115])
```

```
a**a
```

```
array([   1,    1,    4,   27,  256, 3125])
```

easy and intuitive element-wise operations

a ufunc, which can operate both on scalars and arrays (element-wise)

```
np.exp(a)
```

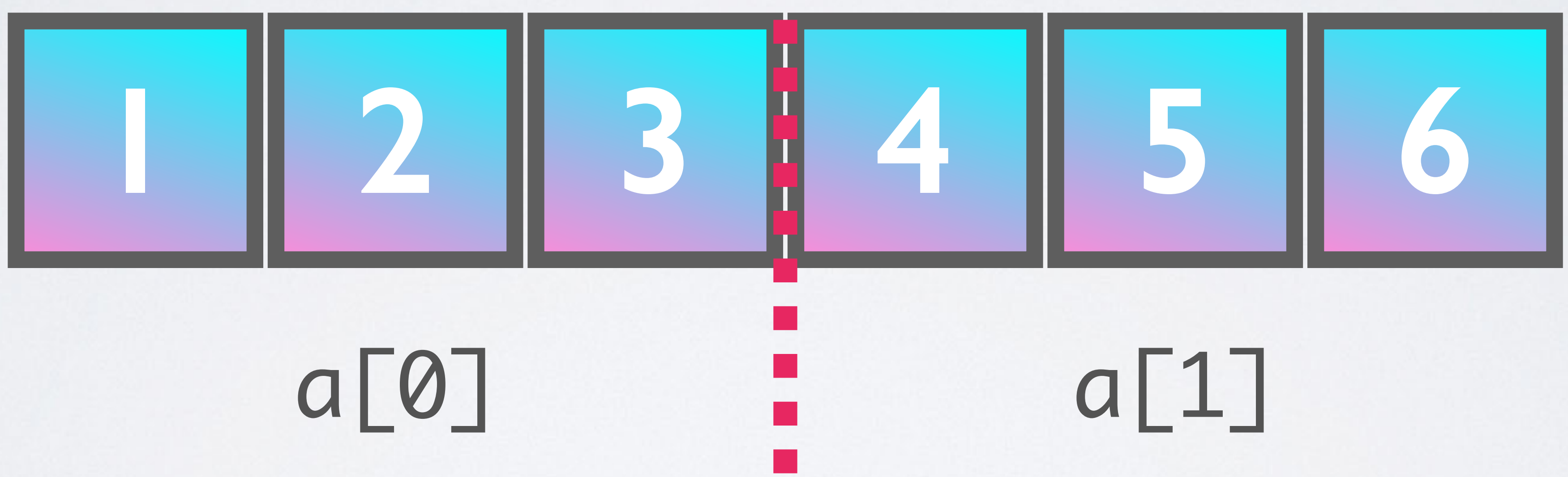
```
array([   1.        ,    2.71828183,    7.3890561 ,   20.08553692,
         54.59815003,  148.4131591 ])
```

RESHAPING ARRAYS

```
a = np.arange(6)
a
```

```
array([0, 1, 2, 3, 4, 5])
```

ndim: 1
shape: (6,)




```
a.reshape(2, 3)
```

```
array([[0, 1, 2],
       [3, 4, 5]])
```

No rearrangement of the elements in memory but setting the iterator limits internally!

RESHAPING ARRAYS IS CHEAP

```
a = np.arange(10000000)
```

```
%timeit b = a.reshape(100, 5000, 20)
563 ns ± 8.18 ns per loop (mean ± std.
```

Don't worry, we will discover NumPy in the hands-on workshop!

NUMFOCUS

OPEN CODE = BETTER SCIENCE

matplotlib

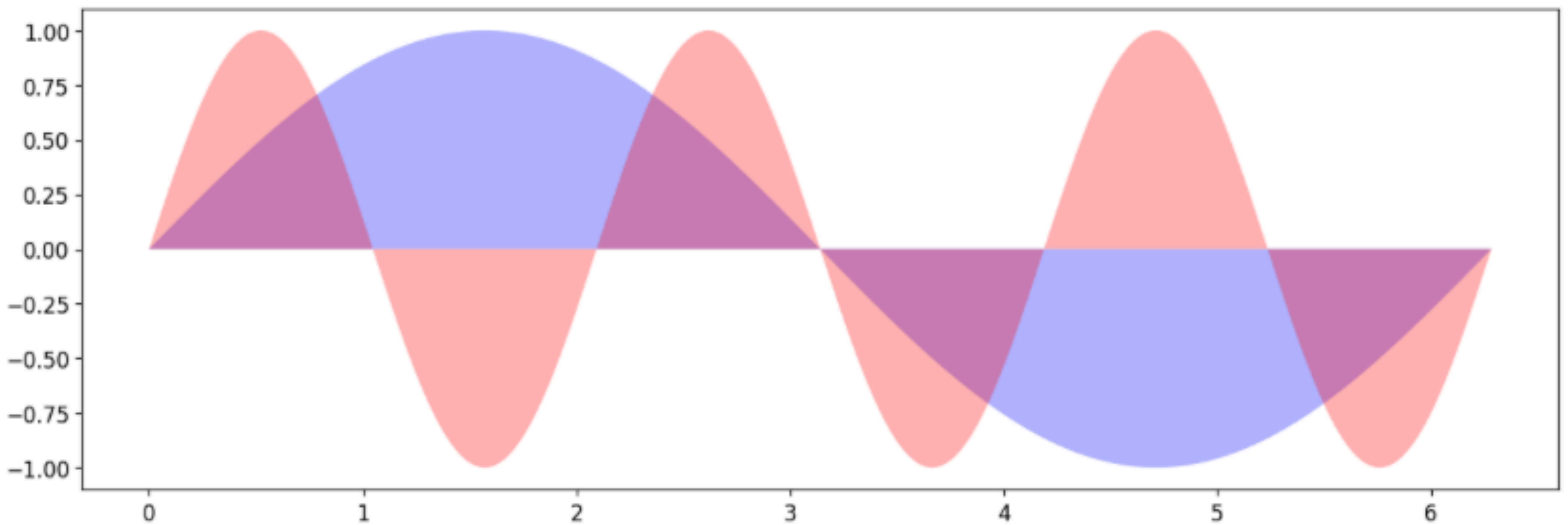
MATPLOTLIB

A Python plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments.

- Integrates well with IPython and Jupyter
- Plots, histograms, power spectra, bar charts, error chars, scatterplots, etc. with an easy to use API
- Full control of line styles, font properties, axes properties etc.
- The easiest way to get started is browsing its wonderful gallery full of thumbnails and copy&paste examples: http://matplotlib.org/gallery.html

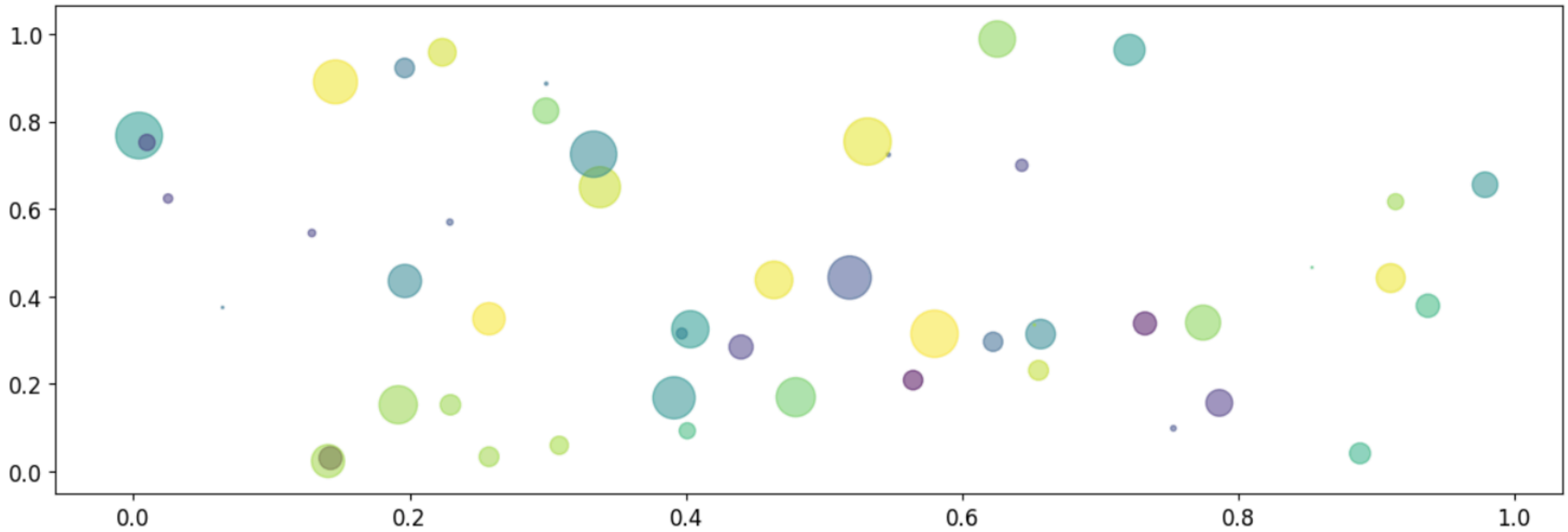
MATPLOTLIB EXAMPLE

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi, 500)
y1 = np.sin(x)
y2 = np.sin(3 * x)

fig, ax = plt.subplots()
ax.fill(x, y1, 'b', x, y2, 'r', alpha=0.3)
plt.show()
```



MATPLOTLIB EXAMPLE

```python
import numpy as np
import matplotlib.pyplot as plt

N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = np.pi * (15 * np.random.rand(N))**2

plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```



NUMFOCUS

OPEN CODE = BETTER SCIENCE

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

PANDAS

A Python Data Analysis Library inspired by data frames in R:

- gives us a powerful data structure: DataFrame
- database-like handling of data
- integrates well with NumPy
- wraps the Matplotlib API (which can also cause troubles ;)
- has a huge number of I/O related functions to parse data: CSV, HDF5, SQL, Feather, JSON, HTML, Excel, and more…

THE DataFrame

A table-like structure, where you can access elements by row and column.

```
hits = pd.read_hdf("event_file.h5", "events/23")
hits.head(3)
```

	channel_id	dom_id	event_id	id	pmt_id	time	tot	triggered
0	25	808430036	0	0	0	30652287	21	0
1	18	808430036	0	0	0	30656200	16	0
2	15	808430449	0	0	0	30648451	26	0

THE DataFrame

Lots of functions to allow filtering, manipulating and aggregating the data to fit your needs.

```
active_doms = hits.pivot_table(index='event_id',
                               values='dom_id',
                               aggfunc=lambda x: set(x))
```

Don't worry, we will discover Pandas in the hands-on workshop!

NUMFOCUS

OPEN CODE = BETTER SCIENCE

PyTables

PYTABLES

HIERARCHICAL DATASETS IN PYTHON

- An HDF5 library for Python
- Database-like approach to data storage
- Features like indexing and fast “in-kernel” queries
- Custom system to represent data types
- Used in Pandas as HDF5 I/O backend

PYTABLES

HIERARCHICAL DATASETS IN PYTHON

```python
import numpy as np
import tables as tb
```

```python
data = np.array([(1,2), (3,4)], dtype=[('a', int), ('b', float)])
```

```python
data
```

```
array([(1, 2.), (3, 4.)], dtype=[('a', '<i8'), ('b', '<f8')])
```

```python
with tb.File('foo.h5', 'w') as tbfile:
    tab = tbfile.create_table('/', 'data', data.dtype)
    tab.append(data)
```

created with PyTables

opened in Julia

```julia
using HDF5
```

```julia
data = h5read("foo.h5", "/data")
```

```
2-element Array{HDF5.HDF5Compound{2},1}:
 HDF5.HDF5Compound{2}((1, 2.0), ("a", "b"), (Int64, Float64))
 HDF5.HDF5Compound{2}((3, 4.0), ("a", "b"), (Int64, Float64))
```

H5PY

H5PY

- An alternative HDF5 wrapper
- Feels more pythonic than PyTables
- Maps the HDF5 feature set to NumPy as closely as possible
- Lightweight and highly performant

H5PY

HIERARCHICAL DATASETS IN PYTHON

```python
import h5py as h5
import numpy as np
```

```python
data = np.array([(1,2), (3,4)], dtype=[('a', int), ('b', float)])
```

```python
with h5.File('bar.h5', 'w') as h5file:
    h5file.create_dataset('data', data=data)
```

created with h5py

opened in Julia

```julia
using HDF5
```

```julia
data = h5read("bar.h5", "/data")
```

```
2-element Array{HDF5.HDF5Compound{2},1}:
 HDF5.HDF5Compound{2}((1, 2.0), ("a", "b"), (Int64, Float64))
 HDF5.HDF5Compound{2}((3, 4.0), ("a", "b"), (Int64, Float64))
```

sponsored by

CONTINUUM® ANALYTICS

NUMBA

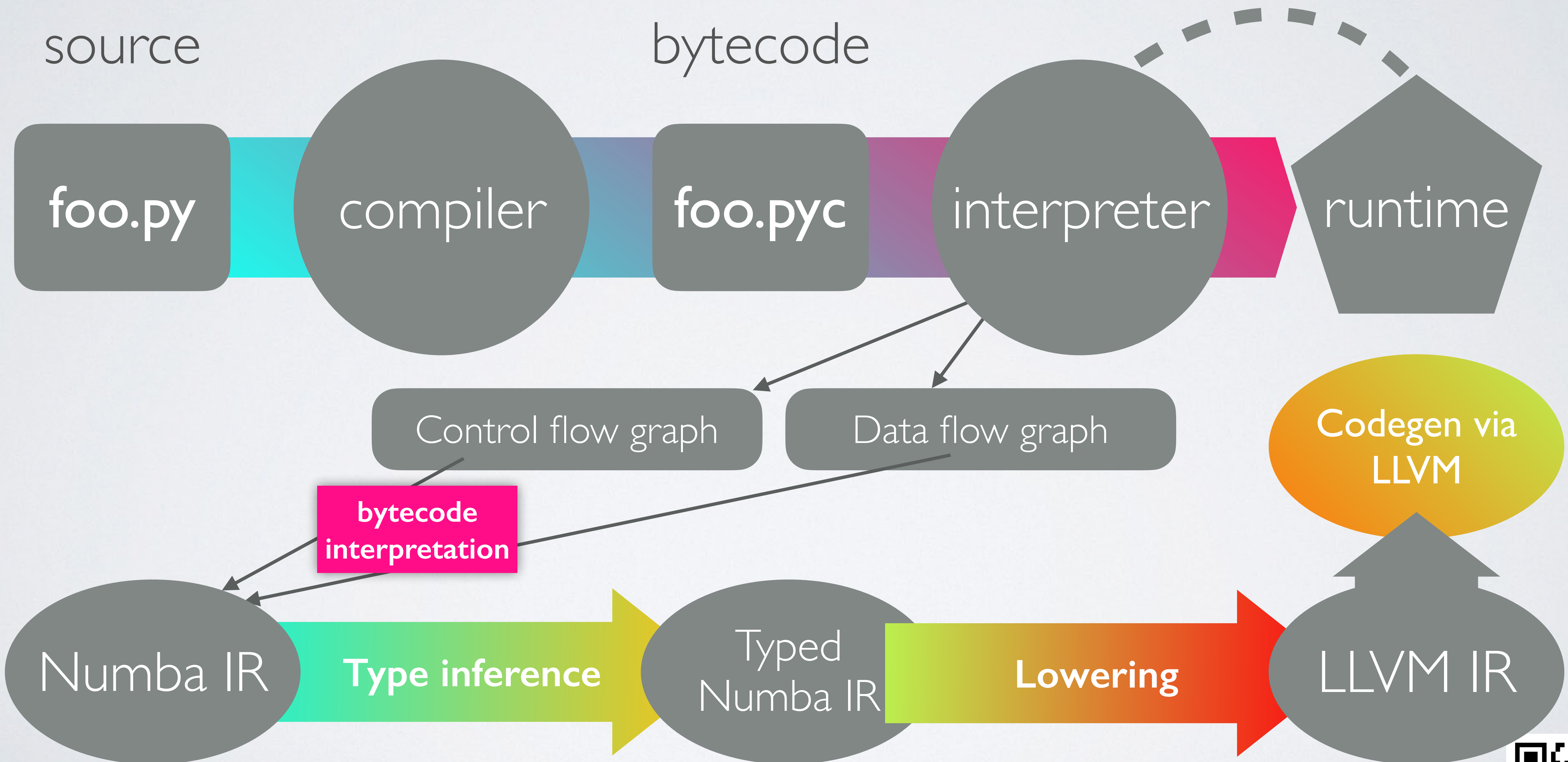
JIT (LLVM) compiler for Python

NUMBA

Numba is a **compiler** for Python array and numerical functions that gives you the power to speed up code written directly in Python.

- uses **LLVM** to boil down pure Python code to **JIT optimised machine code**
- only **accelerates** selected **functions decorated** by yourself
- **native code** generation for **CPU** (default) and **GPU**
- **integration** with the **Python scientific software stack** (thanks to **NumPy**)
- runs side by side with regular Python code or third-party C extensions and libraries
- great **CUDA** support
- **N-core** scalability by releasing the GIL (beware: no protection from race conditions!)
- create **NumPy** ufuncs with the @[gu]vectorize decorator(s)

FROM SOURCE TO RUNTIME



NUMBA JIT-EXAMPLE

```
numbers = np.arange(1000000).reshape(2500, 400)
```

```
def sum2d(arr):
    M, N = arr.shape
    result = 0.0
    for i in range(M):
        for j in range(N):
            result += arr[i,j]
    return result
```

```
@nb.jit
def sum2d_jit(arr):
    M, N = arr.shape
    result = 0.0
    for i in range(M):
        for j in range(N):
            result += arr[i,j]
    return result
```

289 ms ± 3.02 ms per loop

2.13 ms ± 42.6 µs per loop

~135x faster, with a single line of code

NUMBA VECTORIZE-EXAMPLE

```
a = np.arange(1000000, dtype='f8')
b = np.arange(1000000, dtype='f8') + 23
```

NumPy:

```
np.abs(a - b) / (np.abs(a) + np.abs(b))
```

23 ms ± 845 µs per loop

Numba @vectorize:

```
@nb.vectorize
def nb_rel_diff(a, b):
    return abs(a - b) / (abs(a) + abs(b))

rel_diff(a, b)
```

3.56 ms ± 43.2 µs per loop

~6x faster

NUMEXPR

initially written by David Cooke

Routines for the fast evaluation of array expressions element-wise by using a vector-based virtual machine.

NUMEXPR USAGE EXAMPLE

```
import numpy as np
import numexpr as ne

a = np.arange(5)
b = np.linspace(0, 2, 5)

ne.evaluate("a**2 + 3*b”)


array([  0. ,   2.5,   7. ,  13.5,  22. ])
```

NUMEXPR SPEED-UP

```
a = np.random.random(1000000)
```

NumPy:

```
2 * a**3 - 4 * a**5 + 6 * np.log(a)
```

82.4 ms ± 1.88 ms per loop

Numexpr with 4 threads:

```
ne.set_num_threads(4)

ne.evaluate("2 * a**3 - 4 * a**5 + 6 * log(a)")
```

7.85 ms ± 103 µs per loop

~10x faster

NUMEXPR – SUPPORTED OPERATORS

- Logical operators: &, |, ~
- Comparison operators:
  <, <=, ==, !=, >=, >
- Unary arithmetic operators:
  -
- Binary arithmetic operators:
  +, -, *, /, **, %, <<, >>

NUMEXPR – SUPPORTED FUNCTIONS

- where(bool, number1, number2): number -- number1 if the bool condition is true, number2 otherwise.
- {sin,cos,tan}(float|complex): float|complex -- trigonometric sine, cosine or tangent.
- {arcsin,arccos,arctan}(float|complex): float|complex -- trigonometric inverse sine, cosine or tangent.
- arctan2(float1, float2): float -- trigonometric inverse tangent of float1/float2.
- {sinh,cosh,tanh}(float|complex): float|complex -- hyperbolic sine, cosine or tangent.
- {arcsinh,arccosh,arctanh}(float|complex): float|complex -- hyperbolic inverse sine, cosine or tangent.
- {log,log10,log1p}(float|complex): float|complex -- natural, base-10 and log(1+x) logarithms.
- {exp,expm1}(float|complex): float|complex -- exponential and exponential minus one.
- sqrt(float|complex): float|complex -- square root.
- abs(float|complex): float|complex -- absolute value.
- conj(complex): complex -- conjugate value.
- {real,imag}(complex): float -- real or imaginary part of complex.
- complex(float, float): complex -- complex from real and imaginary parts.
- contains(str, str): bool -- returns True for every string in `op1` that contains `op2`.
- sum(number, axis=None): Sum of array elements over a given axis. Negative axis are not supported.
- prod(number, axis=None): Product of array elements over a given axis. Negative axis are not supported.

NUMFOCUS

OPEN CODE = BETTER SCIENCE

astropy

A Community Python Library for Astronomy

THE HISTORY OF ASTROPY

(standard situation back in 2011)

- Example Problem: `convert from EQ J2000 RA/Dec to Galactic coordinates`
- Solution in Python
    - pyast
    - Astrolib
    - Astrophysics
    - PyEphem
    - PyAstro
    - Kapteyn
    - ???

huge discussion
started in June 2011
series of votes

astropy

First public version (v0.2) presented and described in the following paper:
http://adsabs.harvard.edu/abs/2013A%26A...558A..33A

ASTROPY CORE PACKAGE

A community-driven package intended to contain much of the core functionality and some common tools needed for performing astronomy and astrophysics with Python.

- **Data structures and transformations**
  - constants, units and quantities, N-dimensional datasets, data tables, times and dates, astronomical coordinate system, models and fitting, analytic functions
- **Files and I/O**
  - unified read/write interface
  - FITS, ASCII tables, VOTable (XML), Virtual Observatory access, HDF5, YAML, …
- **Astronomy computations and utilities**
  - cosmological calculations, convolution and filtering, data visualisations, astrostatistics tools

ASTROPY AFFILIATED PACKAGES

- Tons of astronomy related packages
- which are not part of the core package,
- but has requested to be included as part of the Astropy project's community

ASTROPY EXAMPLE

```
from astropy.utils.data import download_file
from astropy.io import fits

image_file = download_file('http://data.astropy.org/tutorials/FITS-images/HorseHead.fits')
```

```
Downloading http://data.astropy.org/tutorials/FITS-images/HorseHead.fits [Done]
```

← downloading via HTTP

```
fits.info(image_file)
```

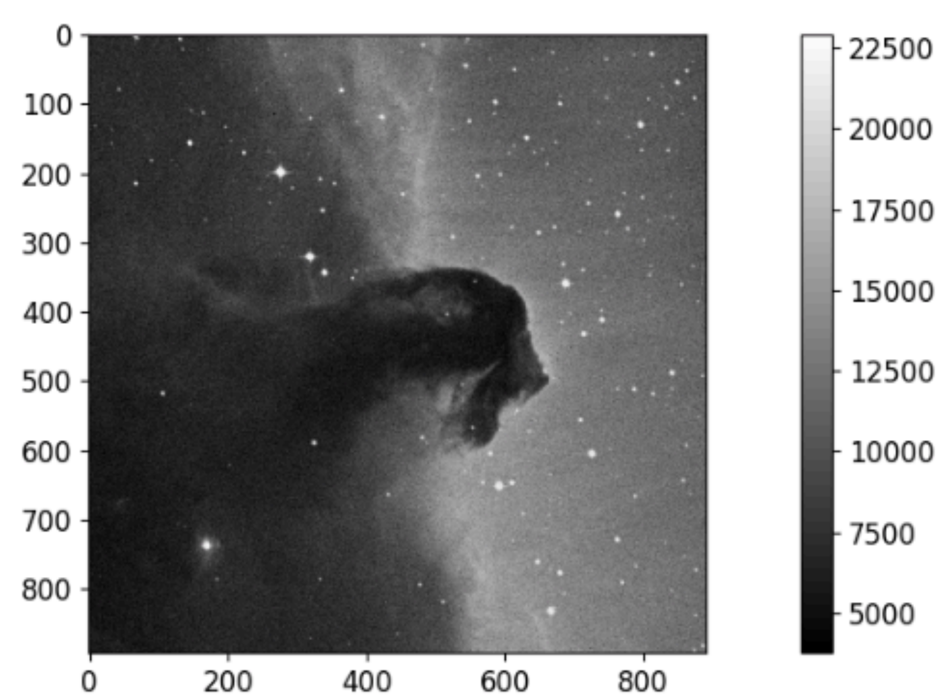
```
Filename: /Users/tamasgal/.astropy/cache/download/py3/2c9202ae878ecfcb60878ceb63837f5f
No.    Name         Type      Cards   Dimensions   Format
  0  PRIMARY     PrimaryHDU     161   (891, 893)   int16
  1  er.mask     TableHDU        25   1600R x 4C   [F6.2, F6.2, F6.2, F6.2]
```

← checking some FITS meta

```
image_data = fits.getdata(image_file, ext=0)
```

← extracting image data

```
plt.figure()
plt.imshow(image_data, cmap='gray');
plt.colorbar();
```



← plotting via Matplotlib

ASTROPY EXAMPLE

```python
from astropy.coordinates import SkyCoord
import astropy.units as u

m13 = SkyCoord.from_name('m13')
m13
```

```
<SkyCoord (ICRS): (ra, dec) in deg
    ( 250.4234583,  36.4613056)>
```

```python
m13.ra, m13.ra.to(u.hourangle)
```

```
(<Longitude 250.4234583 deg>, <Longitude 16.69489722 hourangle>)
```

Don’t worry, we will discover AstroPy in the hands-on workshop!

NUMFOCUS

OPEN CODE = BETTER SCIENCE

SymPy

A Python library for symbolic mathematics.

SYMPY

- It aims to become a full-featured computer algebra system (CAS)
- while keeping the code as simple as possible
- in order to be comprehensible and easily extensible.
- SymPy is written entirely in Python.
- It only depends on mpmath, a pure Python library for arbitrary floating point arithmetic

SIMPY

- solving equations
- solving differential equations
- simplifications: trigonometry, polynomials
- substitutions
- factorisation, partial fraction decomposition
- limits, differentiation, integration, Taylor series
- combinatorics, statistics, …
- much much more

SIMPY EXAMPLE

Base Python

```
In [1]: import math

In [2]: math.sqrt(8)
Out[2]: 2.8284271247461903

In [3]: math.sqrt(8)**2
Out[3]: 8.000000000000002
```

SymPy

```
In [4]: import sympy

In [5]: sympy.sqrt(8)
Out[5]: 2*sqrt(2)

In [6]: sympy.sqrt(8)**2
Out[6]: 8
```

SIMPY EXAMPLE

```
In [15]: x, y = sympy.symbols('x y')

In [16]: expr = x + 2*y

In [17]: expr
Out[17]: x + 2*y

In [18]: expr + 1
Out[18]: x + 2*y + 1

In [19]: expr * x
Out[19]: x*(x + 2*y)

In [20]: sympy.expand(expr * x)
Out[20]: x**2 + 2*x*y
```

SIMPY EXAMPLE

```
In [1]: import sympy

In [2]: from sympy import init_printing, integrate, diff, exp, cos, sin, oo

In [3]: init_printing(use_unicode=True)

In [4]: x = sympy.symbols('x')

In [5]: diff(sin(x)*exp(x), x)
Out[5]:
 x           x
e ⋅sin(x) + e ⋅cos(x)

In [6]: integrate(exp(x)*sin(x) + exp(x)*cos(x), x)
Out[6]:
 x
e ⋅sin(x)

In [7]: integrate(sin(x**2), (x, -oo, oo))
Out[7]:
√2⋅√π
─────
  2
```

Awkward Array

by Jim Pivarski (SciKit-HEP)

https://github.com/scikit-hep/awkward-1.0

https://awkward-array.org

AWKARD ARRAY

MOTIVATION

- **NumPy arrays** are **rectangular** tables or tensors: **cannot express variable-length** structures
- **Tree-like data** (very common in HEP) is difficult to express with NumPy arrays -- in an efficient way
- **Speed** and **performance** are crucial
- Easy to use and interactive interfaces for commonly used operations like cuts and aggregations

AWKARD ARRAY

- Written in **C++** and designed to work with **Python**
- Has **Numba** support to take it on the next level!
- Offers lots of functions to work with **ragged/jagged** data

AWKARD ARRAY

- Written in **C++** and designed to work with **Python**
- Has **Numba** support to take it on the next level!
- Supports arbitrary **tree representations** with as many jagged/ragged structures as you need
- Offers lots of functions to work with **ragged/jagged** data

```
In [1]: import awkward as ak

In [2]: arr = ak.Array([[1,2,3], [4,5], [6,7,8,9]])

In [3]: arr
Out[3]: <Array [[1, 2, 3], [4, 5], [6, 7, 8, 9]] type='3 * var * int64'>

In [4]: arr[:,0]
Out[4]: <Array [1, 4, 6] type='3 * int64'>

In [5]: ak.mean(arr, axis=1)
Out[5]: <Array [2, 4.5, 7.5] type='3 * ?float64'>
```

AWKARD ARRAY

- All kinds of structures are understood and "type stable"

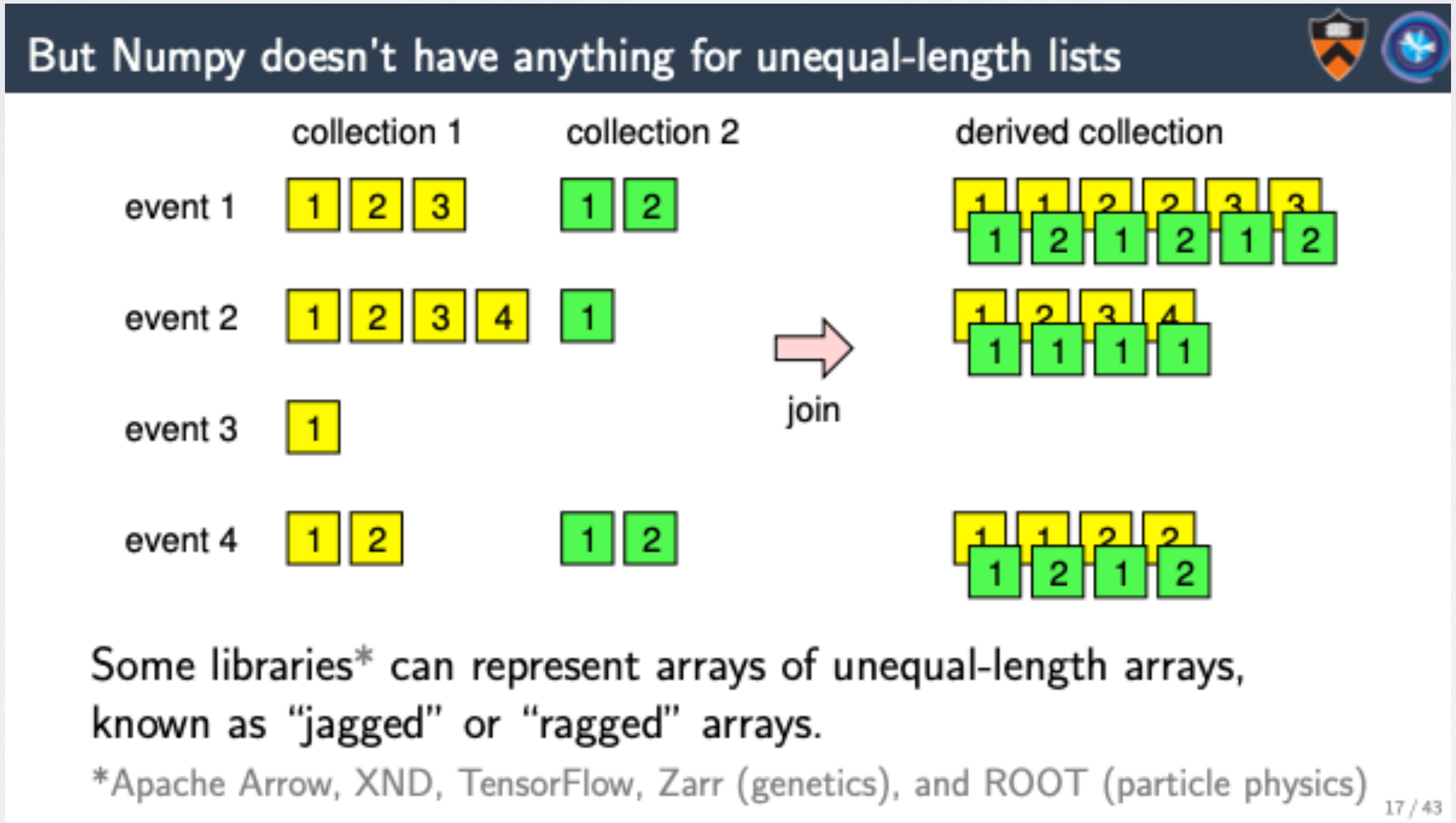
```
In [1]: import awkward as ak

In [2]: arr = ak.Array([(True, 1), (False, 3), (False, 9)])

In [3]: arr
Out[3]: <Array [(True, 1), (False, 3), (False, 9)] type='3 * (bool, int64)'>
```

```
In [1]: import awkward as ak

In [2]: arr = ak.Array([{"pos_x": [3, 45, 65], "pos_y": [5, 4, 6]}, {"pos_x": [1, 3], "pos_y": [5, 6]}])

In [3]: arr
Out[3]: <Array [{pos_x: [3, 45, 65, ... pos_y: [5, 6]}] type='2 * {"pos_x": var * int64,...'>

In [4]: arr.pos_x
Out[4]: <Array [[3, 45, 65], [1, 3]] type='2 * var * int64'>
```

AWKWARD ARRAY

- A very nice **introduction** by **Jim** himself (just search for "**awkward array**" on YouTube): https://www.youtube.com/watch?v=2NxWpU7NArk

- The slide is taken from the presentation above:



NUMFOCUS

OPEN CODE = BETTER SCIENCE

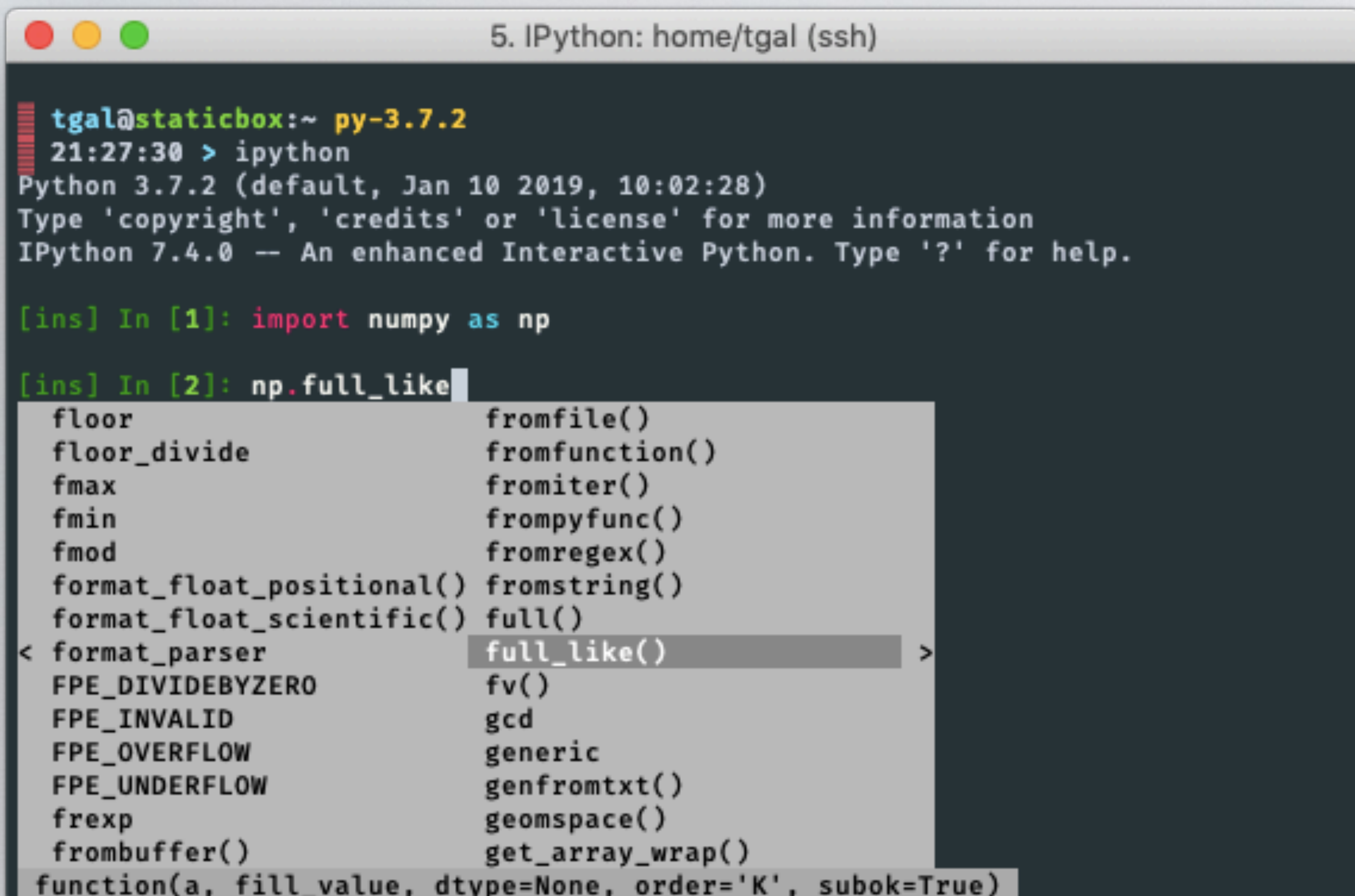
IP[y]:

IPython

IPYTHON

- The interactive Python shell!
- Object introspection
- Input history, persistent across sessions
- Extensible tab completion
- “Magic” commands (basically macros)
- Easily embeddable in other Python programs and GUIs
- Integrated access to the pdb debugger and the Python profiler
- Syntax highlighting
- real multi-line editing
- Provides a kernel for Jupyter
- …and such more!

IPYTHON



- Synax highlighting
- TAB completion
- Function signatures
- etc…

NUMFOCUS
OPEN CODE = BETTER SCIENCE

jupyter

Project Jupyter is an open source project that offers a set of tools for interactive and exploratory computing.

JUPYTER

- Born out of the IPython project in 2014
- Jupyter provides a console and a notebook server for all kinds of languages
  (the name Jupyter comes from **Ju**lia, **Pyt**hon and **R**)
- An easy way to explore and prototype
- Notebooks support Markdown and LaTeX-like input and rendering
  - Allows sharing code and analysis results
  - Extensible (slideshow plugins, JupyterLab, VIM binding, …)

JUPYTER CONSOLE

A terminal frontend for kernels which use the Jupyter protocol.

1. tamasgal@greybox: ~ (zsh)

```
tamasgal@greybox:~  km3net
08:30:16 > jupyter kernelspec list
Available kernels:
  haskell      /Users/tamasgal/Library/Jupyter/kernels/haskell
  julia-0.5    /Users/tamasgal/Library/Jupyter/kernels/julia-0.5
  julia-0.6    /Users/tamasgal/Library/Jupyter/kernels/julia-0.6
  km3net       /Users/tamasgal/Library/Jupyter/kernels/km3net
  python3      /Users/tamasgal/.pyenv/versions/3.6.0/envs/km3net/share/jupyter/kernels/python3
tamasgal@greybox:~  km3net
08:30:26 >
```

1. jupyter console (python3.6)

```
tamasgal@greybox:~  aois
08:27:48 > jupyter console
Jupyter console 5.1.0

Python 3.6.1 (default, May 23 2017, 21:09:20)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.0.0 -- An enhanced Interactive Python. Type '?' for help.



In [1]:
```

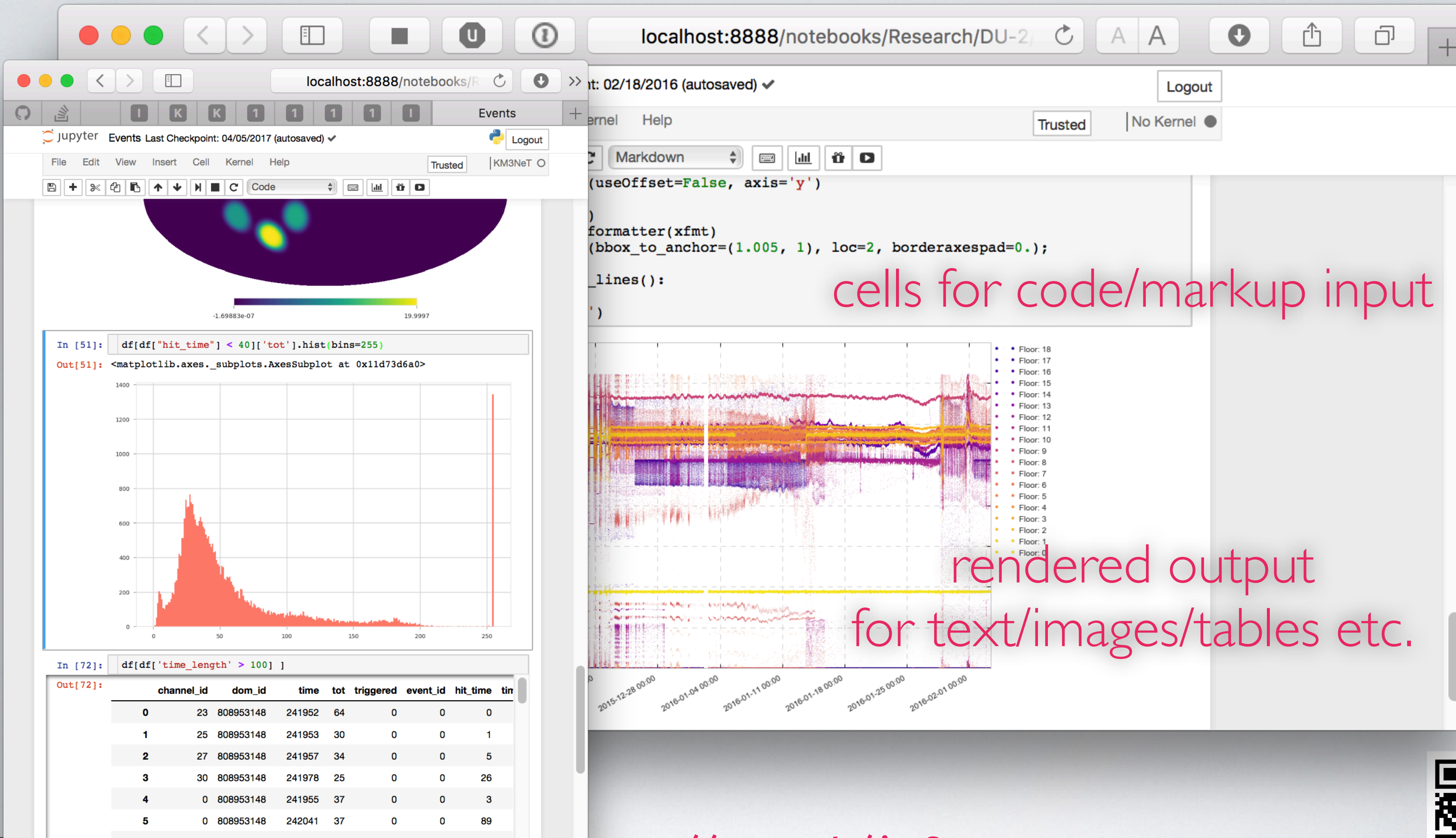
1. jupyter console --kernel=julia-0.5 (python3.6)

```
tamasgal@greybox:~  km3net
08:32:05 > jupyter console --kernel=julia-0.5
Starting kernel event loops.
Jupyter console 5.1.0

Julia: A fresh approach to technical computing.


In [1]: f(α) = cos(2α) * √2
Out[1]: f (generic function with 1 method)

In [2]:
```

JUPYTER NOTEBOOK

- A Web-based application suitable for capturing the whole computation process:
  - developing
  - documenting
  - and executing code
  - as well as communicating the results.
- Two main components:
  - a web application: a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output.
  - notebook documents: a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects.

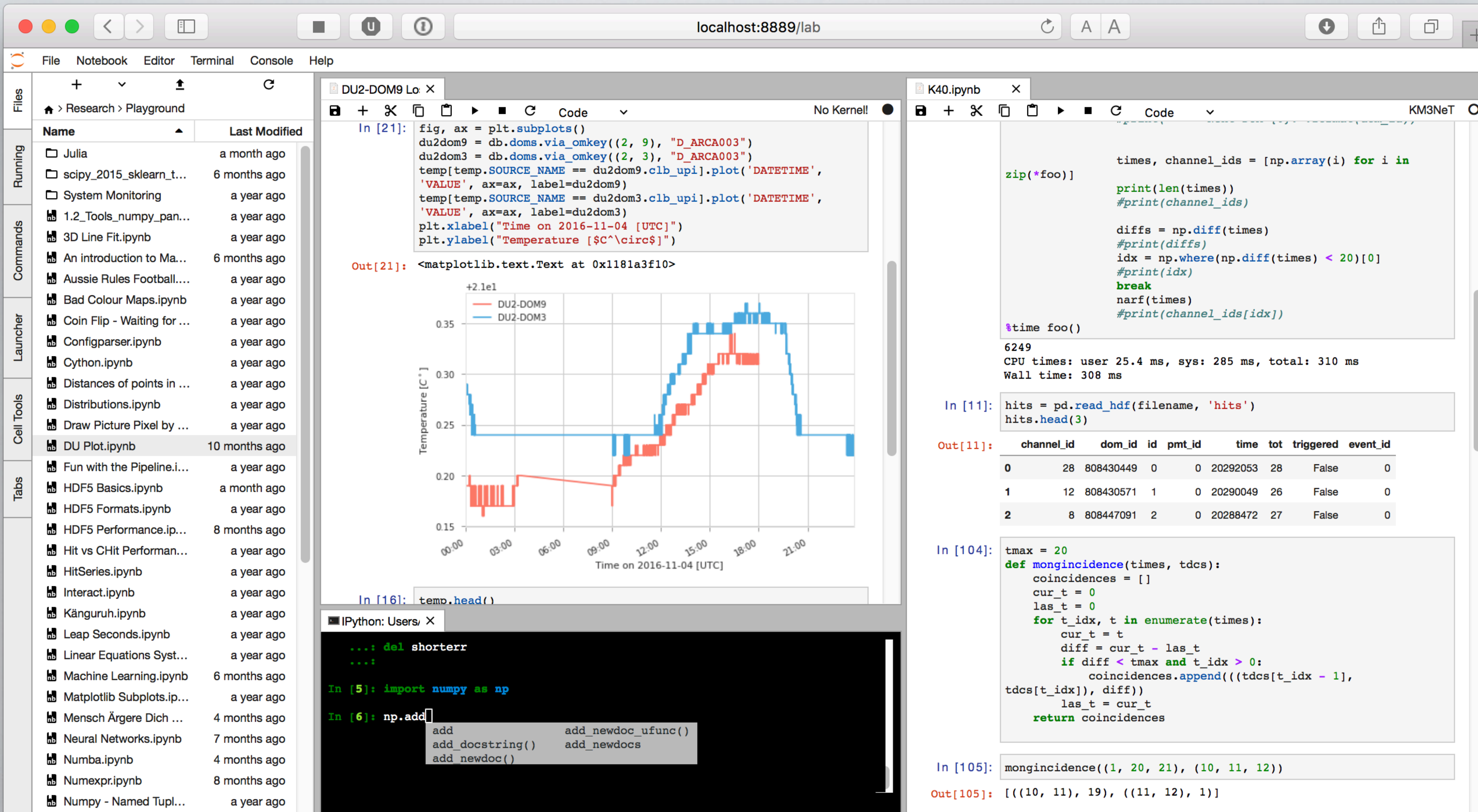
JUPYTER NOTEBOOK



JUPYTERLAB

- The next level of interacting with notebooks
- Extensible: terminal, text editor, image viewer, etc.
- Supports editing multiple notebooks at once
- Drag and drop support to arrange panes

JUPYTERLAB



localhost:8889/lab
File Notebook Editor Terminal Console Help
Files
Running
Commands
Launcher
Cell Tools
Tabs
Research › Playground
Name
Last Modified
Julia a month ago
scipy_2015_sklearn_t... 6 months ago
System Monitoring a year ago
1.2_Tools_numpy_pan... a year ago
3D Line Fit.ipynb a year ago
An introduction to Ma... 6 months ago
Aussie Rules Football.... a year ago
Bad Colour Maps.ipynb a year ago
Coin Flip - Waiting for ... a year ago
Configparser.ipynb a year ago
Cython.ipynb a year ago
Distances of points in ... a year ago
Distributions.ipynb a year ago
Draw Picture Pixel by ... a year ago
DU Plot.ipynb 10 months ago
Fun with the Pipeline.i... a year ago
HDF5 Basics.ipynb a month ago
HDF5 Formats.ipynb a year ago
HDF5 Performance.ip... 8 months ago
Hit vs CHit Performan... a year ago
HitSeries.ipynb a year ago
Interact.ipynb a year ago
Känguruh.ipynb a year ago
Leap Seconds.ipynb a year ago
Linear Equations Syst... a year ago
Machine Learning.ipynb 6 months ago
Matplotlib Subplots.ip... a year ago
Mensch Ärgere Dich ... 4 months ago
Neural Networks.ipynb 7 months ago
Numba.ipynb 4 months ago
Numexpr.ipynb 8 months ago
Numpy - Named Tupl... a year ago
DU2-DOM9 Lo
Code
No Kernel!
In [21]:
fig, ax = plt.subplots()
du2dom9 = db.doms.via_omkey((2, 9), "D_ARCA003")
du2dom3 = db.doms.via_omkey((2, 3), "D_ARCA003")
temp[temp.SOURCE_NAME == du2dom9.clb_upi].plot('DATETIME', 'VALUE', ax=ax, label=du2dom9)
temp[temp.SOURCE_NAME == du2dom3.clb_upi].plot('DATETIME', 'VALUE', ax=ax, label=du2dom3)
plt.xlabel("Time on 2016-11-04 [UTC]")
plt.ylabel("Temperature [C°]")
Out[21]: <matplotlib.text.Text at 0x1181a3f10>
+2.1e1
DU2-DOM9
DU2-DOM3
Temperature [C°]
Time on 2016-11-04 [UTC]
In [16]: temp.head()
IPython: Users
...: del shorterr
In [5]: import numpy as np
In [6]: np.add
add
add_docstring()
add_newdoc()
add_newdoc_ufunc()
add_newdocs
K40.ipynb
KM3NeT
times, channel_ids = [np.array(i) for i in zip(*foo)]
print(len(times))
#print(channel_ids)
diffs = np.diff(times)
#print(diffs)
idx = np.where(np.diff(times) < 20)[0]
#print(idx)
break
narf(times)
#print(channel_ids[idx])
%time foo()
6249
CPU times: user 25.4 ms, sys: 285 ms, total: 310 ms
Wall time: 308 ms
In [11]: hits = pd.read_hdf(filename, 'hits')
hits.head(3)
Out[11]:
channel_id dom_id id pmt_id time tot triggered event_id
0 28 808430449 0 0 20292053 28 False 0
1 12 808430571 1 0 20290049 26 False 0
2 8 808447091 2 0 20288472 27 False 0
In [104]:
tmax = 20
def mongincidence(times, tdcs):
coincidences = []
cur_t = 0
las_t = 0
for t_idx, t in enumerate(times):
cur_t = t
diff = cur_t - las_t
if diff < tmax and t_idx > 0:
coincidences.append(((tdcs[t_idx - 1], tdcs[t_idx]), diff))
las_t = cur_t
return coincidences
In [105]: mongincidence((1, 20, 21), (10, 11, 12))
Out[105]: [((10, 11), 19), ((11, 12), 1)]

JUPYTERHUB

- JupyterHub creates a multi-user Hub which spawns, manages, and proxies multiple instances of the single-user Jupyter notebook server

- A nice environment for teaching

- Great tool for collaborations (ask your IT admin ;)

NUMFOCUS

OPEN CODE = BETTER SCIENCE

nteract

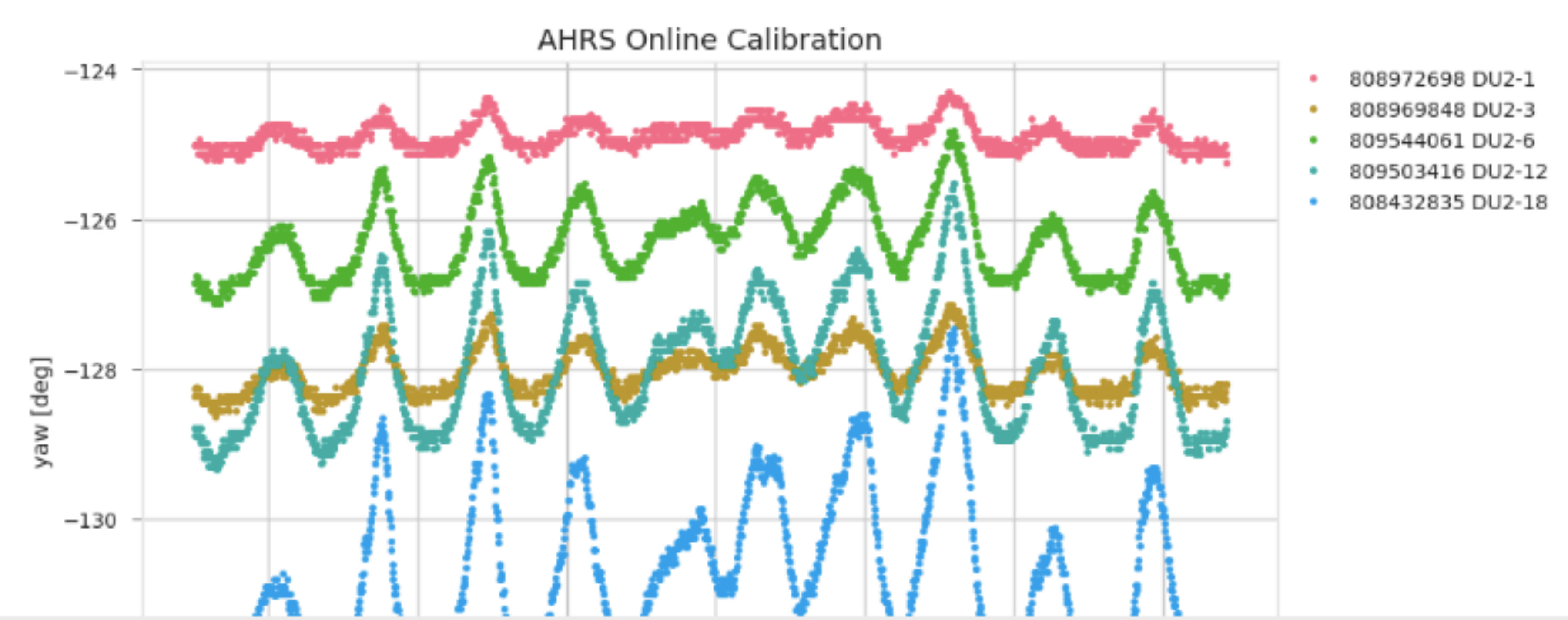
NTERACT

- stand-alone desktop application for developing computational notebooks
- integrates into your system and file browser
- convenient tool to quick-look notebooks, without the need to launch a Jupyter server or a browser
- easy setup: discovers all available kernels (most of the time ;)

NTERACT

~/Documents/Research/AHRS/AHRS ORCA.ipynb - not connected

```
    ax.xaxis.set_major_formatter(xfmt)
    ax.plot(data.datetime, getattr(data, 'yaw'),
            linestyle='none', marker='.',
            label="{} DU{}-{}".format(dom_id, du, floor))
plt.title("AHRS Online Calibration")
plt.xlabel("time [UTC]")
plt.ylabel("{} [deg]".format('yaw'))
fig.autofmt_xdate()
plt.legend(bbox_to_anchor=(1.005, 1), loc=2, borderaxespad=0.)
plt.savefig("{}.pdf".format('yaw'))
plt.show()
```



python3 | launching

Not saved yet

SOME OTHER USEFUL LIBRARIES

SEABORN

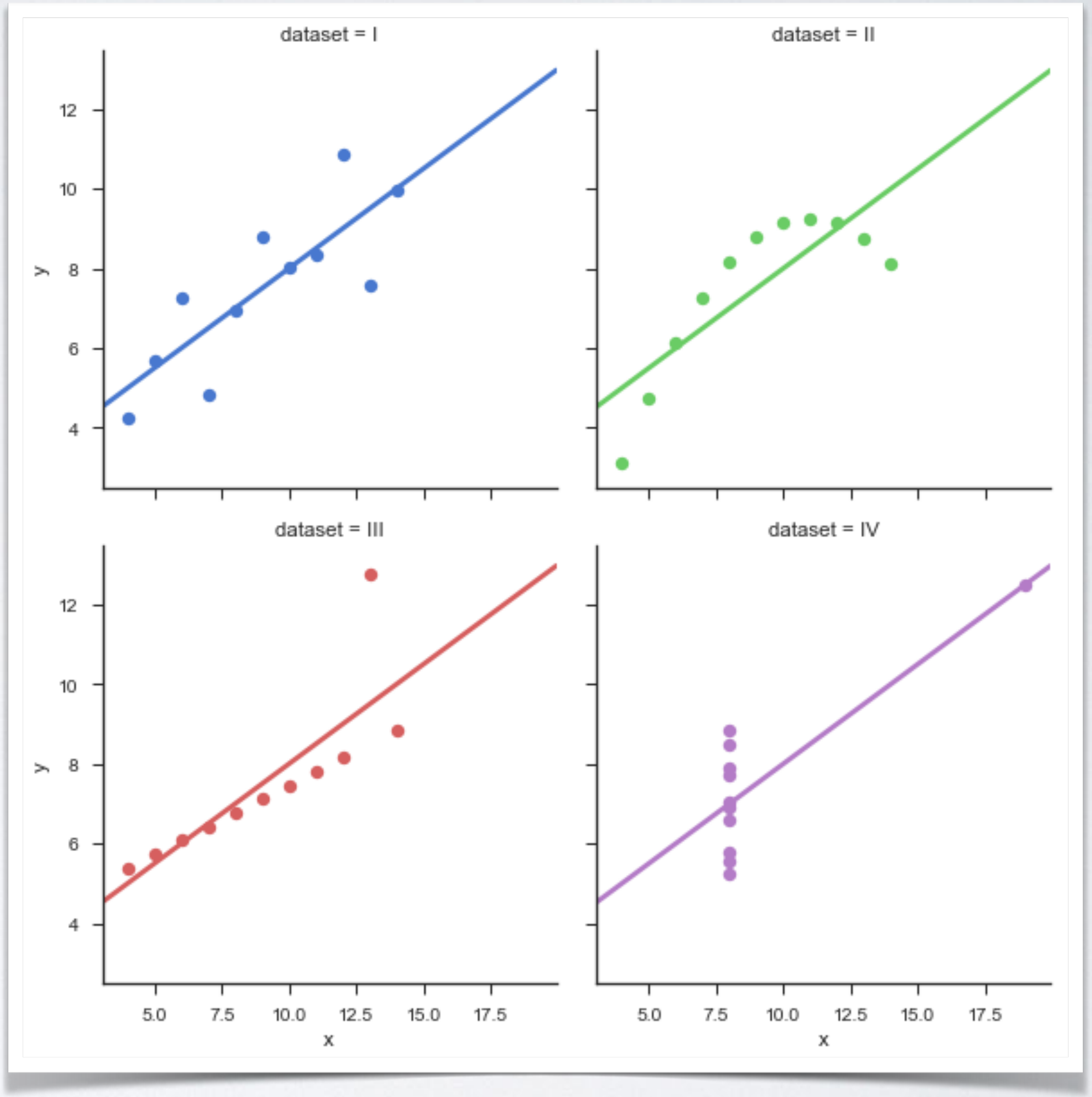
statistical data visualisation

uses matplotlib as backend

https://seaborn.pydata.org

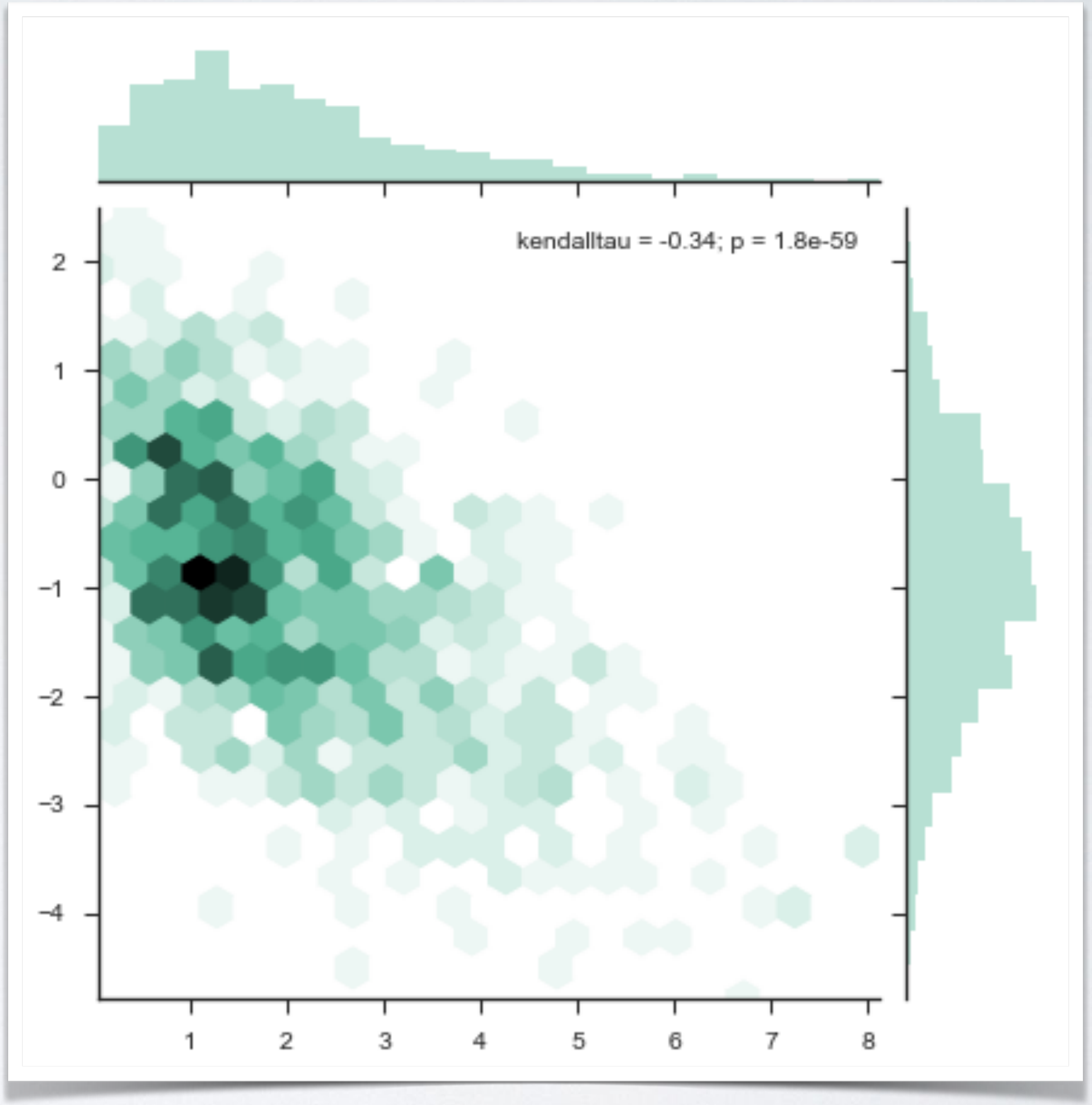
CONVENIENT WRAPPER FUNCTIONS FOR MATPLOTLIB

```
import seaborn as sns
sns.set(style="ticks")

df = sns.load_dataset("anscombe")

# Show the results of a linear regression
# within each dataset
sns.lmplot(x="x", y="y", col="dataset",
           hue="dataset", data=df,
           col_wrap=2, ci=None,
           palette="muted", size=4,
           scatter_kws={"s": 50, "alpha": 1})
```



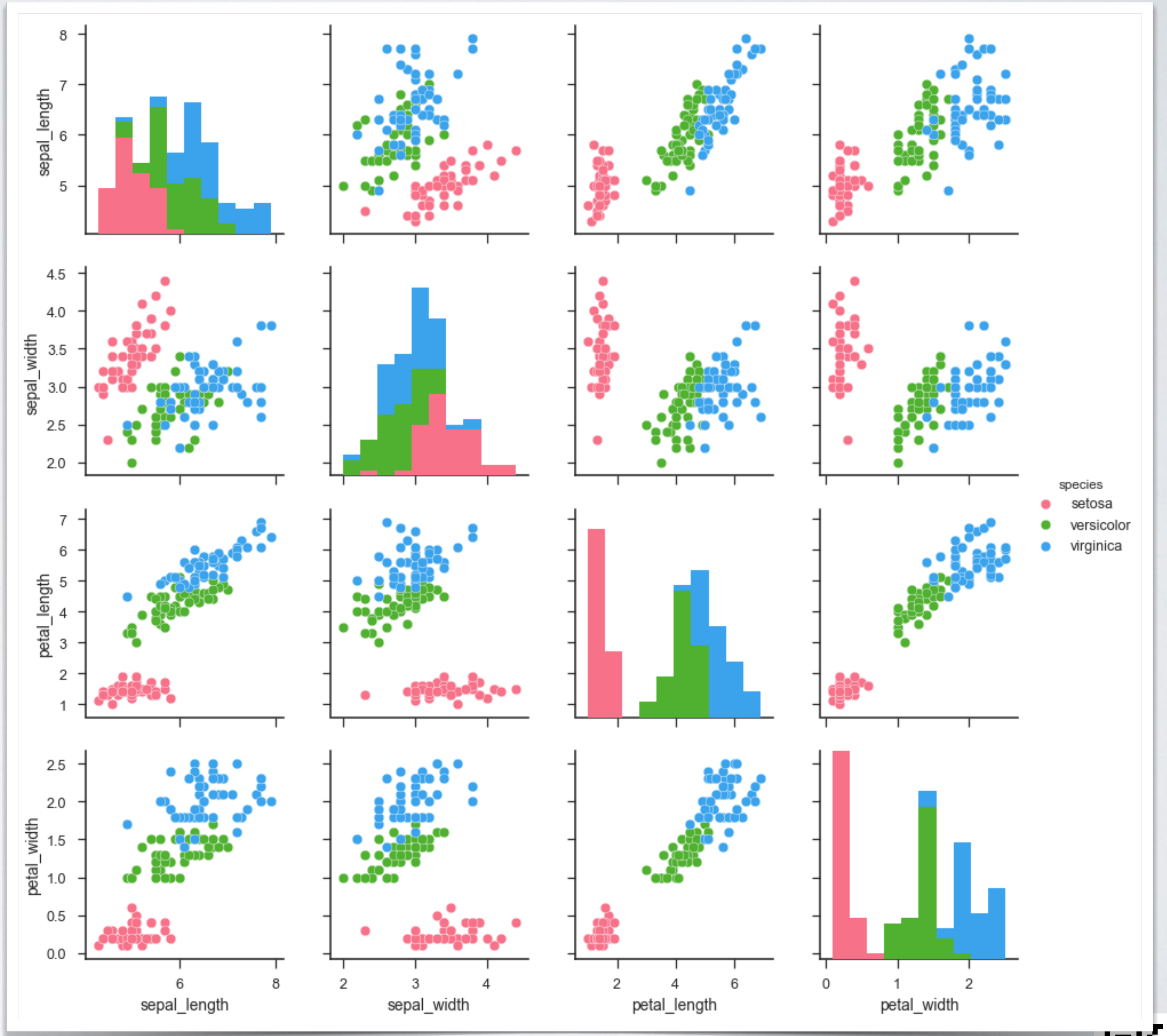
CONVENIENT WRAPPER FUNCTIONS FOR MATPLOTLIB

```
import numpy as np
from scipy.stats import kendalltau
import seaborn as sns
sns.set(style="ticks")

rs = np.random.RandomState(11)
x = rs.gamma(2, size=1000)
y = -.5 * x + rs.normal(size=1000)

sns.jointplot(x, y, kind="hex",
              stat_func=kendalltau,
              color="#4CB391")
```



CONVENIENT WRAPPER FUNCTIONS FOR MATPLOTLIB

```
import seaborn as sns
sns.set(style="ticks",
        color_codes=True)

iris = sns.load_dataset("iris")
sns.pairplot(iris,
             hue="species",
             palette="husl")
```



DOCOPT

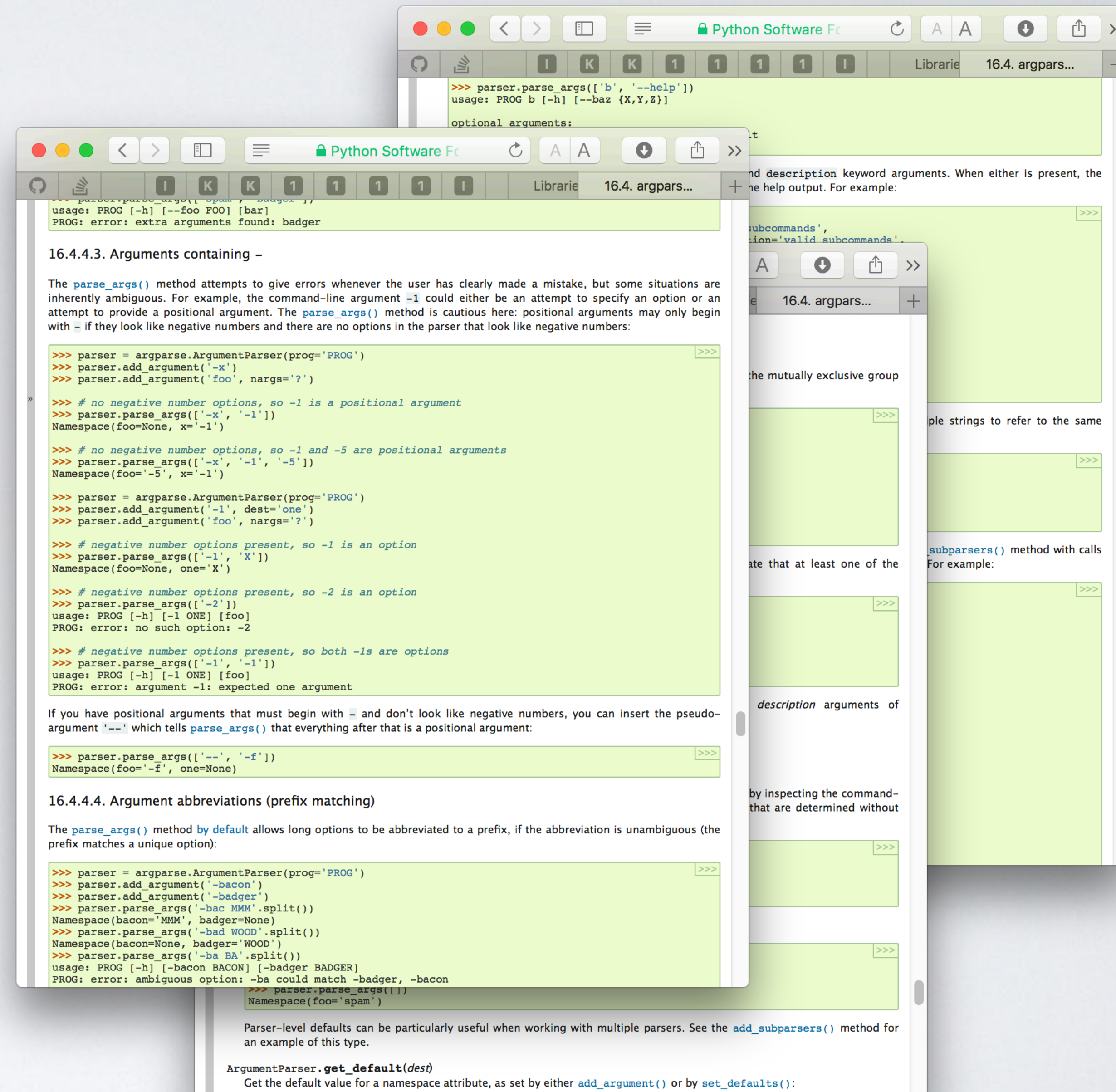
creates beautiful command-line interfaces

by Vladimir Keleshev

https://github.com/docopt/docopt

ARGPARSE/OPTPARSE

Many classes and functions,
default values,
extensive documentation,
very hard to memorise
a basic setup.



DOCOPT

```
#!/usr/bin/env python
"""
Naval Fate.

Usage:
  naval_fate ship new <name>...
  naval_fate ship <name> move <x> <y> [--speed=<kn>]
  naval_fate ship shoot <x> <y>
  naval_fate mine (set|remove) <x> <y> [--moored|--drifting]
  naval_fate -h | --help
  naval_fate --version

Options:
  -h --help     Show this screen.
  --version     Show version.
  --speed=<kn>  Speed in knots [default: 10].
  --moored      Moored (anchored) mine.
  --drifting    Drifting mine.
"""
from docopt import docopt
arguments = docopt(__doc__, version='Naval Fate 2.0')
```

DOCOPT

```
naval_fate ship Guardian move 10 50 --speed=20
```

```
arguments =
{
  "--drifting": false,
  "--help": false,
  "--moored": false,
  "--speed": "20",
  "--version": false,
  "<name>": [
    "Guardian"
  ],
  "<x>": "10",
  "<y>": "50",
  "mine": false,
  "move": true,
  "new": false,
  "remove": false,
  "set": false,
  "ship": true,
  "shoot": false
}
```

CLICK

a mature command line utility interface package

http://click.pocoo.org

CLICK

- Much more advanced compared to docopt
- The no.1 choice if you want to go crazy with command line utilities

```python
import click

@click.command()
@click.option('--count', default=1, help='Number of greetings.')
@click.option('--name', prompt='Your name',
              help='The person to greet.')
def hello(count, name):
    """Simple program that greets NAME for a total of COUNT times."""
    for x in range(count):
        click.echo('Hello %s!' % name)

if __name__ == '__main__':
    hello()
```

SO, WHAT NOW?

FINAL PERSONAL THOUGHTS

I spent a lot of time optimising Python code in the past years, here is a short summary of my personal experience.

- There were **several attempts to make Python itself faster** w.r.t. low level programming, **none of them are satisfying** (PyPy, Pythran etc.), **many of them were abandoned**
- **Think twice** (or more) **before you bake Cython** or any other static compilation **into your project**. The two language problem is real and it's hard to get it right. The performance gain is often disillusioning compared to the work, workarounds and "mess" one needs to deal with later on.
- Me and my lovely dev-team made the **best experiences with numba**
    - no clutter or double bookkeeping, no (static) compilation
    - **minimal dependencies** (basically only LLVMlite)
    - often orders of magnitudes faster than comparable low level algorithms utilising custom Cython class instances or ctypes
    - dict support, finally! (v0.43+)
    - **downside: the code is super slow without numba** …
- When it comes to high performance code using Python, you have to **think in numpy** arrays and cannot easily model your own datatypes like e.g. in C or C++ (structs, classes …)
- A very nice alternative for awkwardly structured data: Awkward Arrays!

MY RECEIPT FOR PERFORMANT PYTHON CODE

- **Avoid massive amounts of Python class instances**
  (e.g. don't create a class for a Point and then a list of 10 million points!)
- **Use numpy arrays for large homogenous data, awkward for "ragged data"**
  (w.r.t. the "points" example above, create a 3xN numpy recarray instead, so you can access points.x, points.y and point.z. Subclass the array if you need some special functionality)
- **Vectorisation is a good idea (most of the time).**
  For basic operations, you most likely find a dedicated function in numpy or scipy.
- **Try to reuse already allocated memory** (allocations are expensive!)
- **Always profile first, before you do heavy optimisations!**
  "[…] premature optimization is the root of all evil." –D. Knuth

  Keep in mind, this doesn't mean that you sit down and hack together code, whatever works, this is not what Donald meant! Take care of the basic principles of performant code from the very beginning, otherwise you will have a hard time to refactor.
- **Do not reinvent the wheel.**
  You mostly find a lib which does what you need, better, faster and for no cost.

Ohne more thing ...

AN EXAMPLE WHY IT'S SO HARD TO MAKE PYTHON FAST?

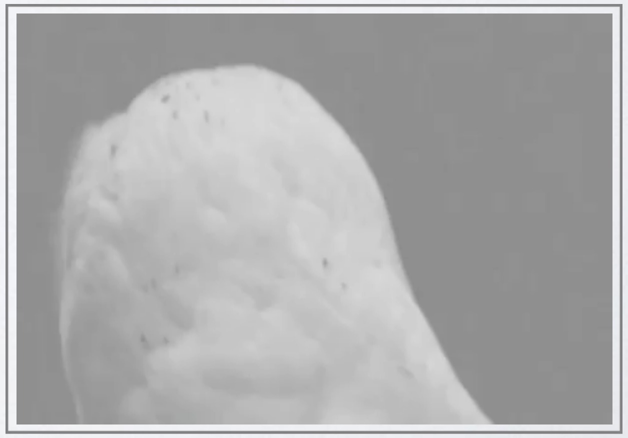
JUST A SIMPLE, BUT CRUCIAL ASPECT…

- Python lets you do anything.
- Here is a "pure" function, written in Python:

```
def square(x):
    return float(x)**2
```

- Every decent compiler should now be able to optimise code using this function (repeated calls, tail recursion elimination, inlining, thread safety guarantees, etc.)

```
import builtins
builtins.float = int
```



THANK YOU!

...also many thanks to the whole organising committee!

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824064 (ESCAPE, the European Science Cluster of Astronomy & Particle Physics ESFRI Research Infrastructures).