

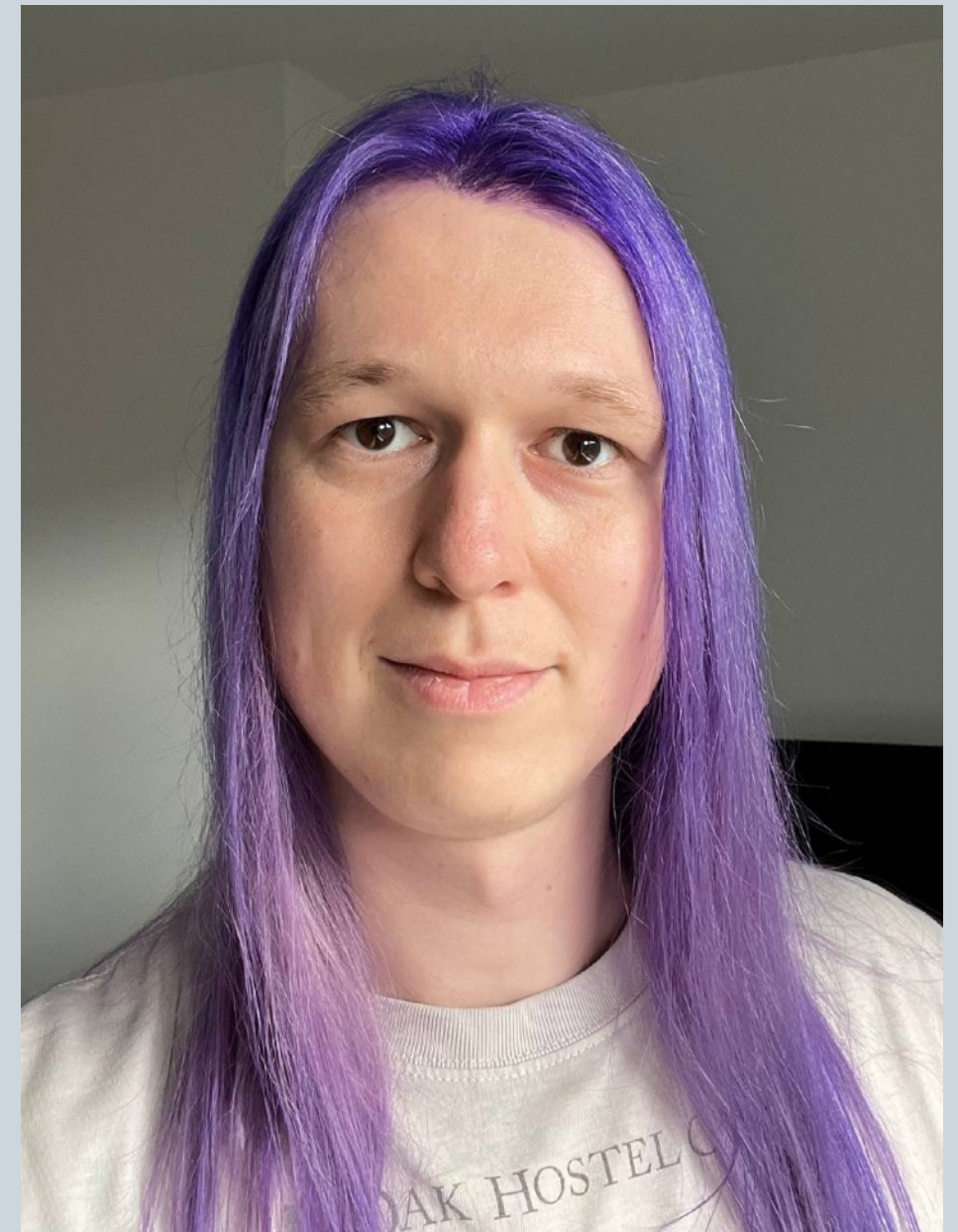
Make Your Python Code Faster

Dr Jost Migenda
they/them

About Me



- ♦ From Berlin & work in London
- ♦ Astrophysics PhD & postdoc ...
 - ♦ Lead maintainer of several Python packages in neutrino physics
- ♦ ... now Senior RSE at KCL
 - ♦ Co-developing a course on Python Profiling & Optimization



Agenda

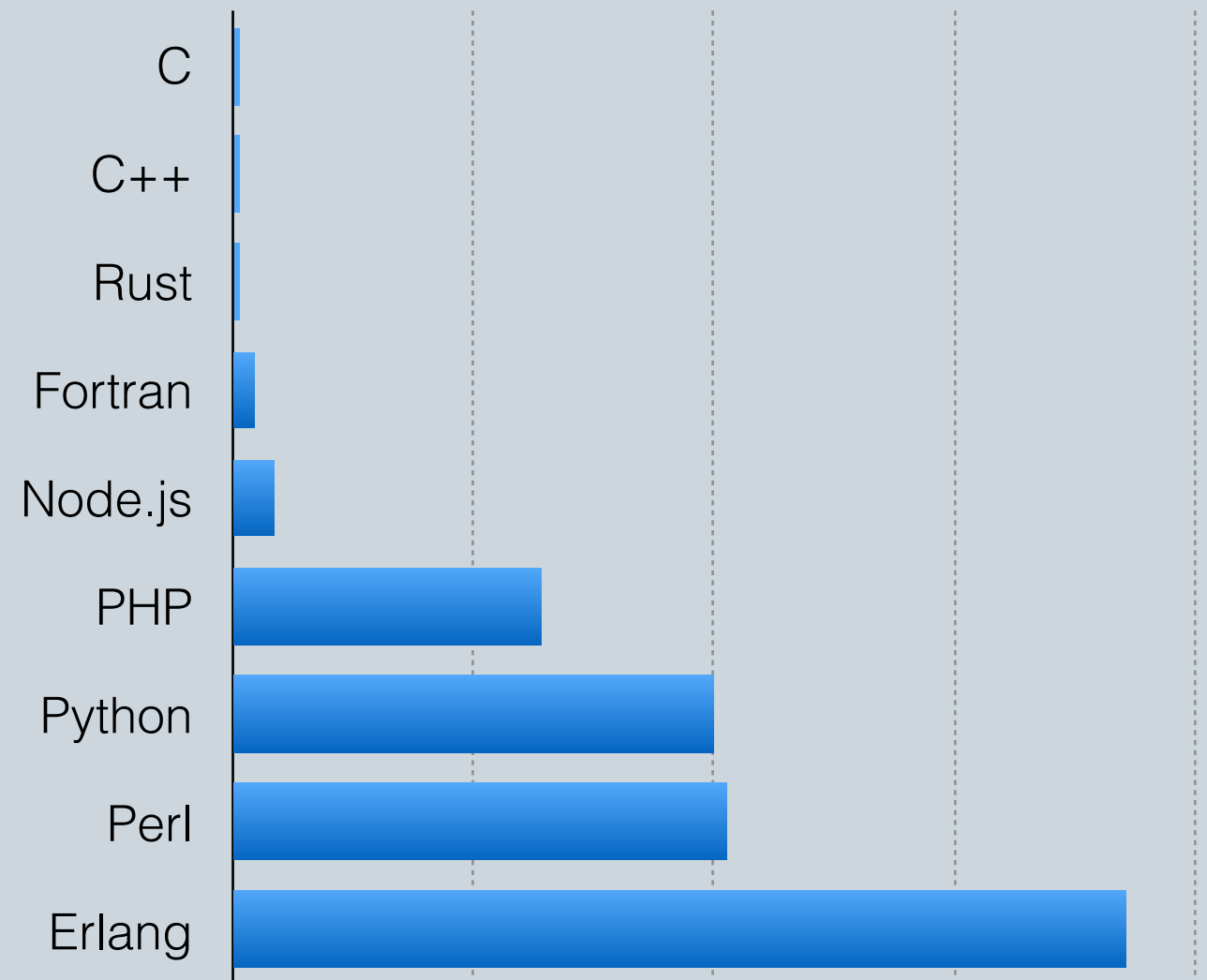
- ♦ Intro: Is Python slow?
- ♦ Profiling: Which parts of my code are slow?
- ♦ Optimization: How do I make them faster?

Based on a course at <https://rse.shef.ac.uk/pando-python/>

Is Python Slow?

- ♦ Yes.
- ♦ ... but that's completely missing the point!

Time to complete a set of example problems (shorter is better)



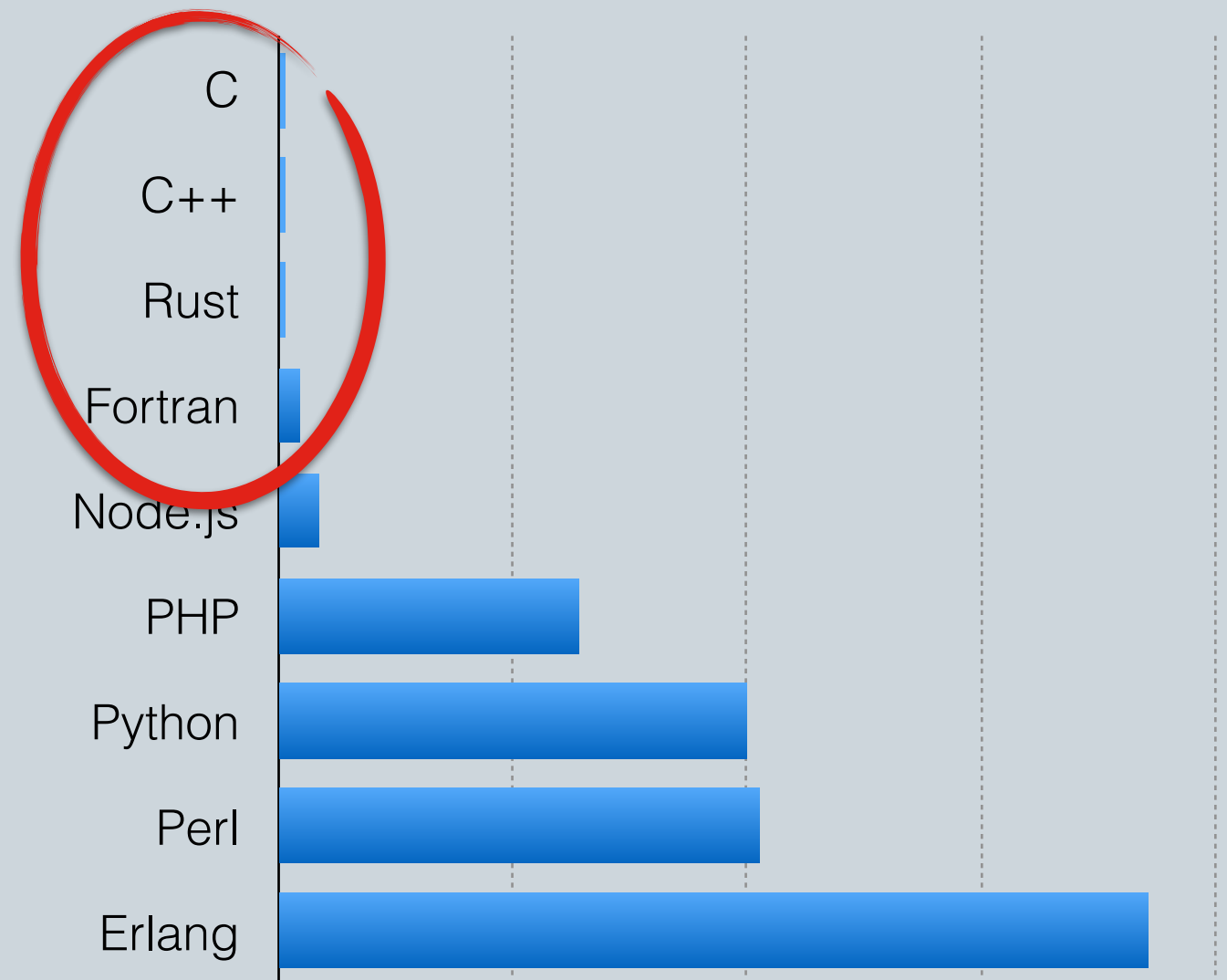
Based on user-submitted implementations.

<https://benchmarksgame-team.pages.debian.net/benchmarksgame/>

Is Python Slow?

- ♦ Yes.
- ♦ ... but that's completely missing the point!
- ♦ It's often “fast enough”
- ♦ It's much faster to write
- ♦ ... and it can use faster languages under the hood

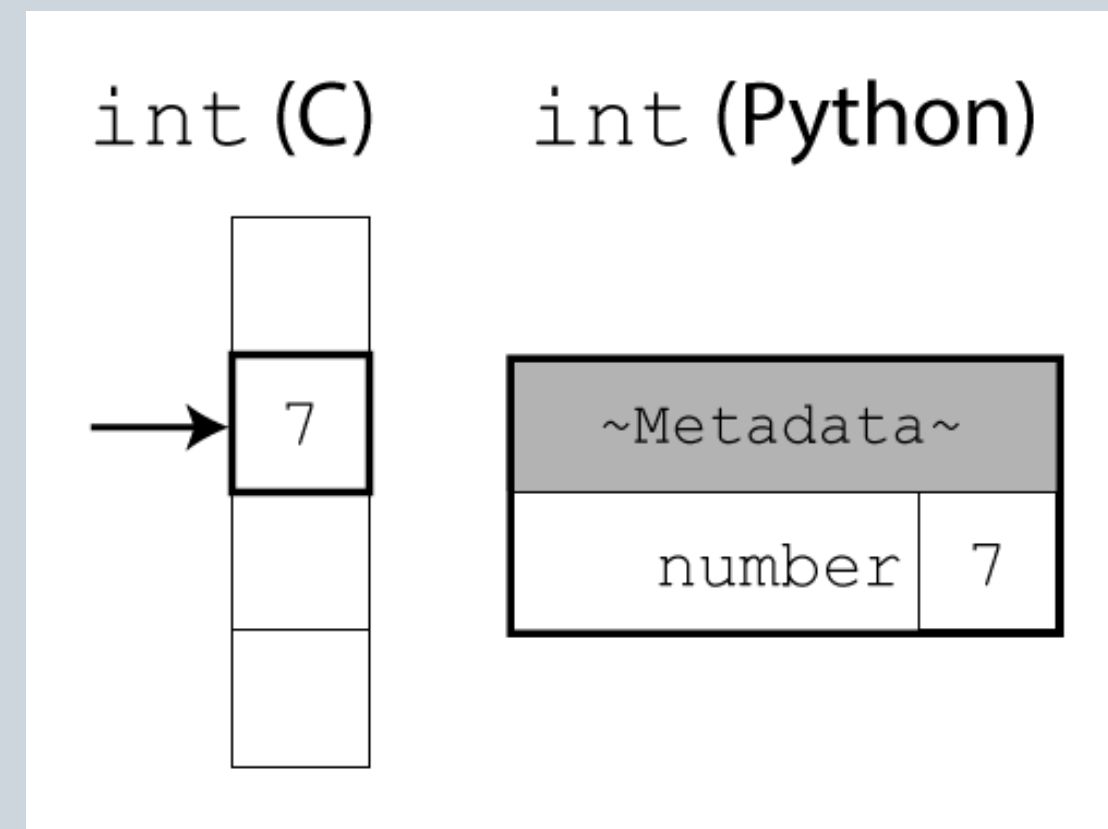
Time to complete a set of example problems (shorter is better)



Based on user-submitted implementations.
<https://benchmarksgame-team.pages.debian.net/benchmarksgame/>

Why Is Python Slow?

- ♦ C deals with raw data in memory
- ♦ In Python, everything is a (fairly complex) object



- ♦ Handle e.g. data types & memory management
→ gain programmer performance by sacrificing code performance
- ♦ Metadata in header requires more memory (3.5× for an integer) & extra work to “unwrap”



Why Is Python Slow?

- Simple code example:

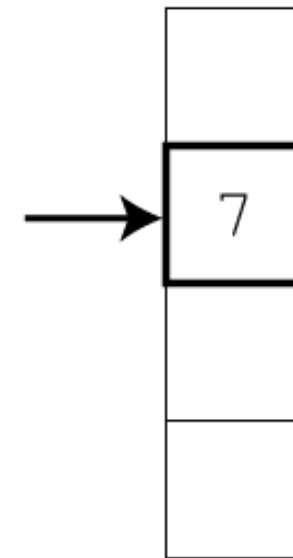
`a = 1`

`b = 2`

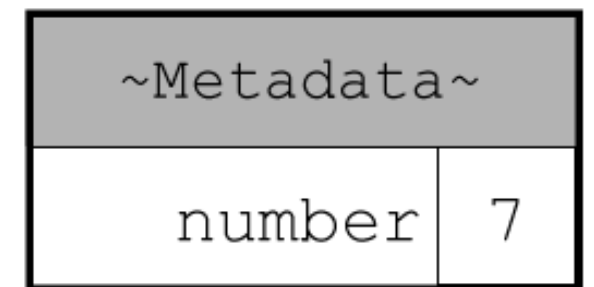
`c = a + b`

- “Unwrap” a
- Get value of a
- “Unwrap” b
- Get value of b
- Add values of a and b to get value of c
- “Re-wrap” value of c

`int (C)`



`int (Python)`



Agenda

- ♦ Intro: Is Python slow?
- ♦ Profiling: Which parts of my code are slow?
- ♦ Optimization: How do I make them faster?

Based on a course at <https://rse.shef.ac.uk/pando-python/>

Profiling

- ♦ **Performance profiling:**

The process of **measuring the performance** of a program or script, to **understand where time is being spent** during execution.

- ♦ Can also do memory profiling, GPU profiling, ...
(Beyond the scope of today's talk!)

- ♦ Any code that will run for more than a few minutes over its lifetime can benefit from profiling.

- ♦ Probably not worth it for quick one-off scripts 🙄



: The `timeit` Module

- ♦ Simple but effective
- ♦ Becomes impractical for profiling more complicated programs

```
1  from timeit import default_timer
2
3
4  t_a = default_timer()
5
6  # Load data from file here ...
7
8  t_b = default_timer()
9
10 # Analyse data here ...
11
12 t_c = default_timer()
13
14 # Format results and write them to file ...
15
16 t_d = default_timer()
17
18 print(f>Loading data took {t_b - t_a:.2f} seconds")
19 print(f>Analysing data took {t_c - t_b:.2f} seconds")
20 print(f>Writing results took {t_d - t_c:.2f} seconds")
```

Can also use `timeit.timeit(...)` for averaging runtime over many runs.



: Function Profiling

- ♦ How many times is the function called?
- ♦ How long does it take? **In total? Per call?**
- ♦ Use the cProfile module in the Python standard library

- ♦ Go from:

```
python my_script.py input.csv
```

to:

```
python -m cProfile my_script.py input.csv
```

28 function calls in 4.751 seconds

Ordered by: cumulative time

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.000	0.000	4.751	4.751	{built-in method builtins.exec}
1	0.000	0.000	4.751	4.751	profiling.py:1(<module>)
1	0.000	0.000	4.751	4.751	profiling.py:3(a_1)
11	4.751	0.432	4.751	0.432	{built-in method time.sleep}
3	0.000	0.000	2.743	0.914	profiling.py:9(b_1)
3	0.000	0.000	1.514	0.505	profiling.py:16(c_1)
3	0.000	0.000	1.229	0.410	profiling.py:19(c_2)
1	0.000	0.000	1.003	1.003	profiling.py:13(b_2)
3	0.000	0.000	0.314	0.105	profiling.py:23(d_1)
1	0.000	0.000	0.000	0.000	{method 'disable' of '_lsprof.Profiler' objects}

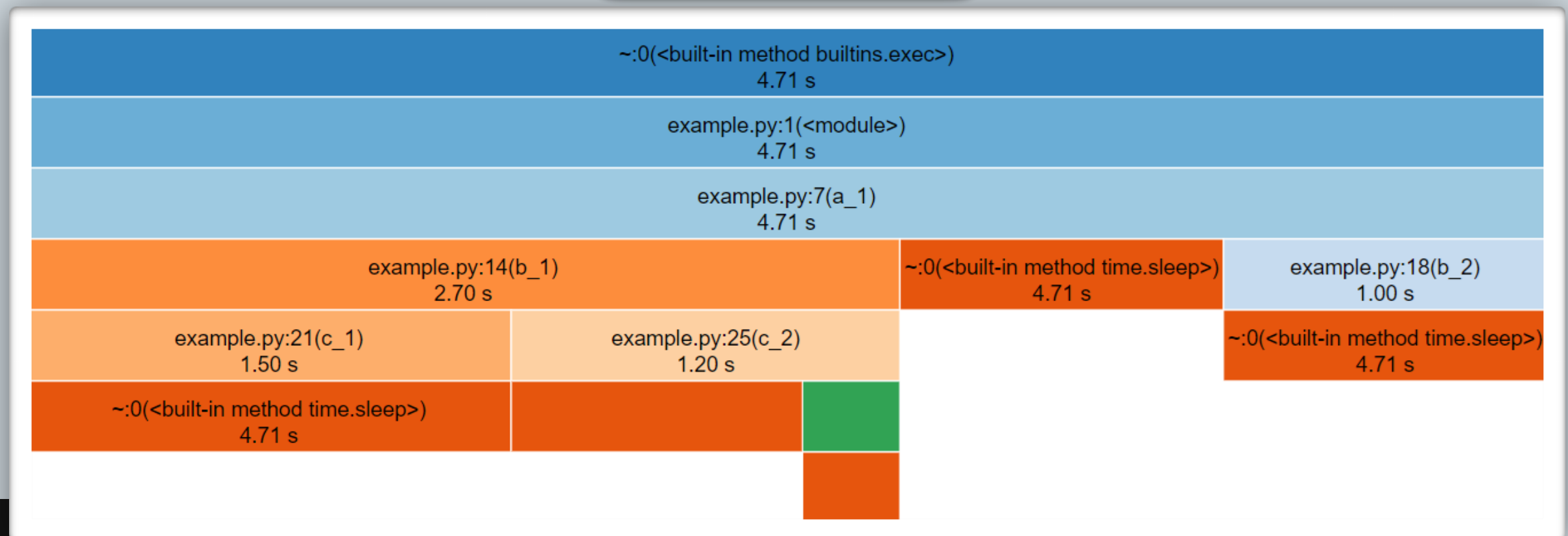
Cumulative time:

includes running time of all child functions called by this function



: Function Profiling

- ♦ How many times is the function called?
- ♦ How long does it take? **In total? Per call?**



Ordered by: cumulative time

ncalls	totttime	percall	cumtime	percall	filename:lineno(function)
1	0.000	0.000	4.751	4.751	{built-in method builtins.exec}
1	0.000	0.000	4.751	4.751	profiling.py:1(<module>)
1	0.000	0.000	4.751	4.751	profiling.py:3(a_1)
11	4.751	0.432	4.751	0.432	{built-in method time.sleep}
3	0.000	0.000	2.743	0.914	profiling.py:9(b_1)
3	0.000	0.000	1.514	0.505	profiling.py:16(c_1)
3	0.000	0.000	1.229	0.410	profiling.py:19(c_2)
1	0.000	0.000	1.003	1.003	profiling.py:13(b_2)
3	0.000	0.000	0.314	0.105	profiling.py:23(d_1)
1	0.000	0.000	0.000	0.000	{method 'disable' of '_lsprof.Profiler' objects}

Cumulative time:
includes running time of all child functions called by this function



: Line Profiling

- Records time to execute individual lines of code
- Gives very detailed information, but can slow down the code significantly!
 - The `line_profiler` module requires marking which function(s) to profile

```
1 from line_profiler import profile
2
3 @profile
4 def is_prime(number):
5     if number < 2:
6         return False
7     for i in range(2, int(number**0.5) + 1):
8         if number % i == 0:
9             return False
10    return True
11
12 print(is_prime(1087))
```



: Line Profiling

- Records time to execute individual lines of code
- Gives very detailed information, but can slow down the code significantly!
 - The `line_profiler` module requires marking which function(s) to profile

```
Wrote profile results to profiling.py.lprof
Timer unit: 1e-06 s
```

```
Total time: 3e-05 s
```

```
File: profiling.py
```

```
Function: is_prime at line 3
```

Line #	Hits	Time	Per Hit	% Time	Line Contents
3					@profile
4					def is_prime(number):
5	1	1.0	1.0	3.3	if number < 2:
6					return False
7	32	4.0	0.1	13.3	for i in range(2, int(number**0.5) + 1):
8	31	24.0	0.8	80.0	if number % i == 0:
9					return False
10	1	1.0	1.0	3.3	return True

Agenda

- ♦ Intro: Is Python slow?
- ♦ Profiling: Which parts of my code are slow?
- ♦ Optimization: How do I make them faster?

Based on a course at <https://rse.shef.ac.uk/pando-python/>

How Do I Make My Code Faster?

1. STOP!

2. Ask yourself: Is it worth the trade-off?

- ♦ How much time should I spend optimizing code that's already “fast enough”?
- ♦ Does the optimization make the code harder to understand?
- ♦ How large is the risk of introducing bugs if I rewrite this code? (Do you have tests you can run?)

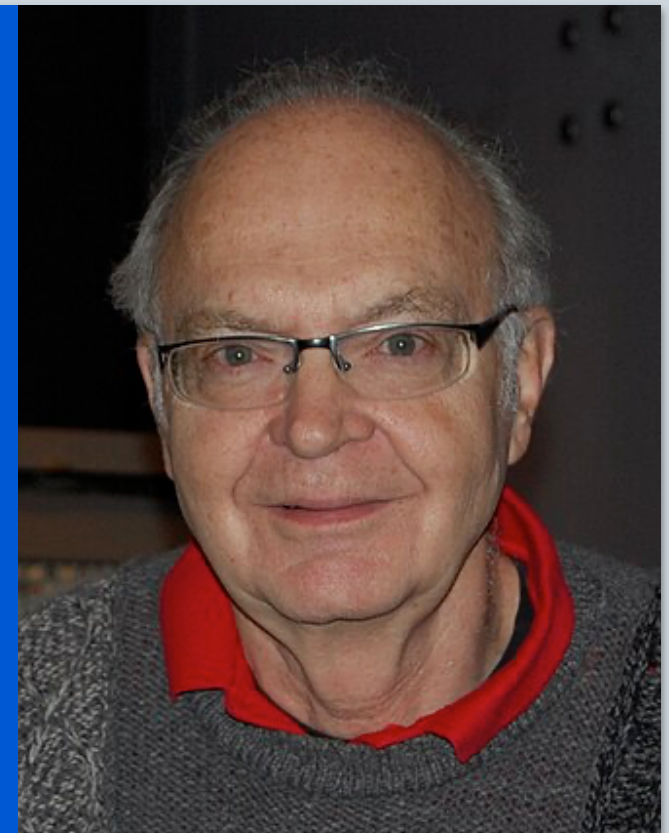
How Do I Make My Code Faster?

1. STOP!

2. Ask yourself: Is it worth the trade-off?

“We should forget about small efficiencies, say, about 97% of the time: premature optimization is the root of all evil.”

— Donald Knuth



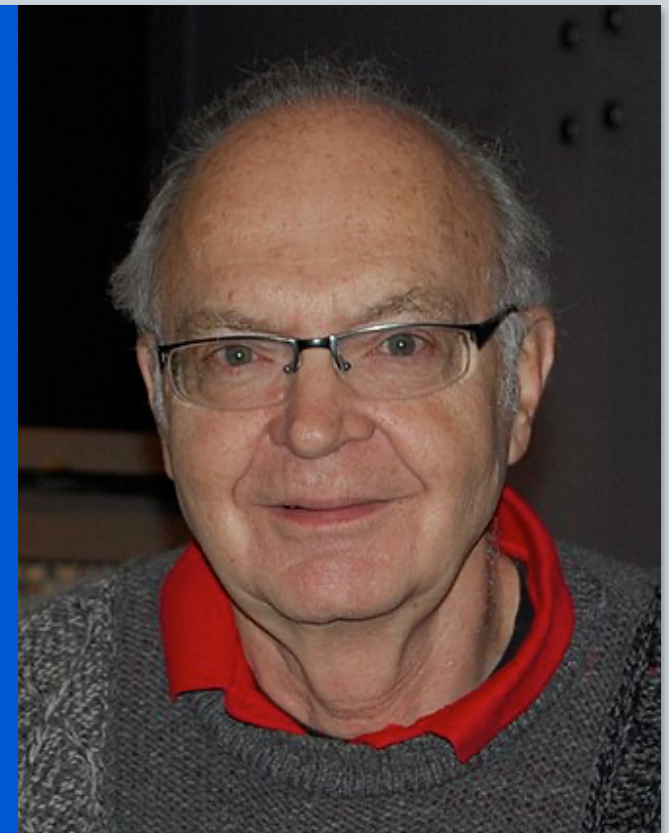
How Do I Make My Code Faster?

1. STOP!

2. Ask yourself: Is it worth the trade-off?

“We should forget about small efficiencies, say, about 97% of the time: premature optimization is the root of all evil.”

— Donald Knuth



3. If yes ... here are some tips and tricks

Optimization

- ♦ Using language features & the standard library effectively
- ♦ Basics of data structures & algorithms
- ♦ Scientific Python: NumPy, Pandas *et al.*
- ♦ Updating Python, disk/network latency & more
- ♦ <https://rse.shef.ac.uk/pando-python/optimisation-using-python.html>

- ♦ The remainder of this talk went through the course materials and discussed some highlights, instead of using slides.
- ♦ Please see <https://rse.shef.ac.uk/pando-python/optimisation-using-python.html>

Summary

- ♦ “Premature optimisation is the root of all evil”
— *Donald Knuth*
- ♦ Profile your code to find the slow parts
 - ♦ Manually (timeit), per function (cProfile) or per line (line_profiler)
- ♦ Speed up those slow parts
 - ♦ Use Python language features, functions & data structures effectively
 - ♦ Parallelise calculations with NumPy *et al.*
 - ♦ ...
- ♦ Full 1-day course: <https://rse.shef.ac.uk/pando-python/>