Make your Python 30x faster!

DANIEL HOGG

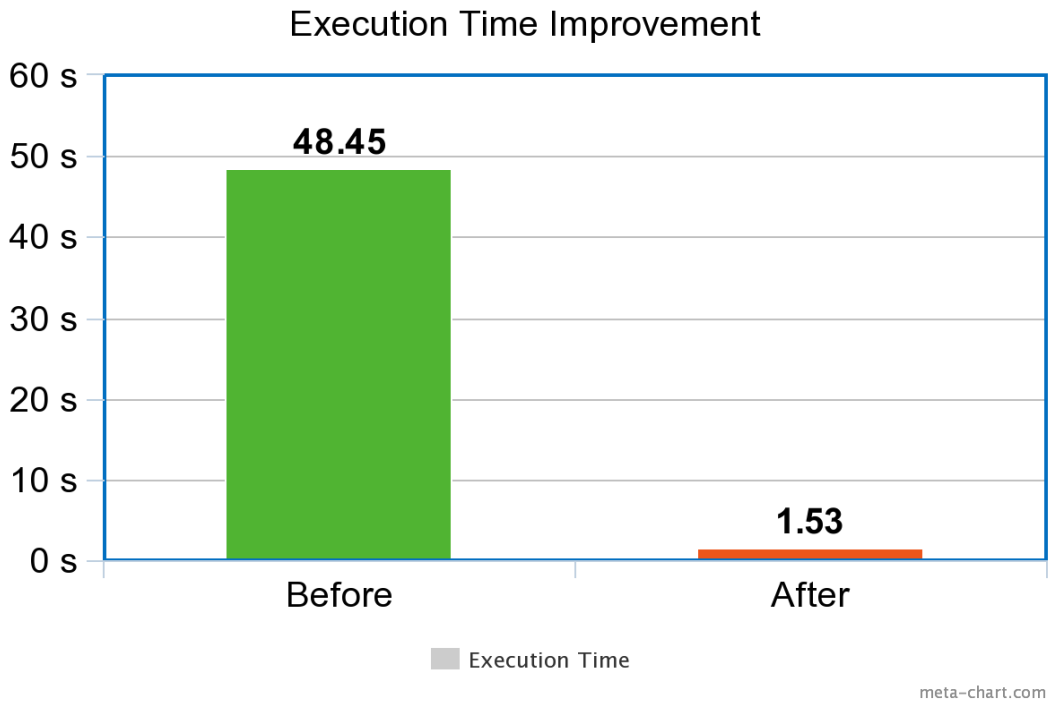
Performance is primary

This is particularly true in atmospheric science, with vast datasets.

1. Computational resource (i.e. server time) is limited
2. Amount of data processed is proportional to execution speed
3. Speed is one of the few downsides of Python

What do we want? Speed.

Performance gain for reading and processing 2.4 GB of gridded atmospheric data.



Parallel processing: Out of scope

	Python	**C**
Single-threaded	1x	50x
Multithreading	12x	600x

Holy Wars: Programming Edition

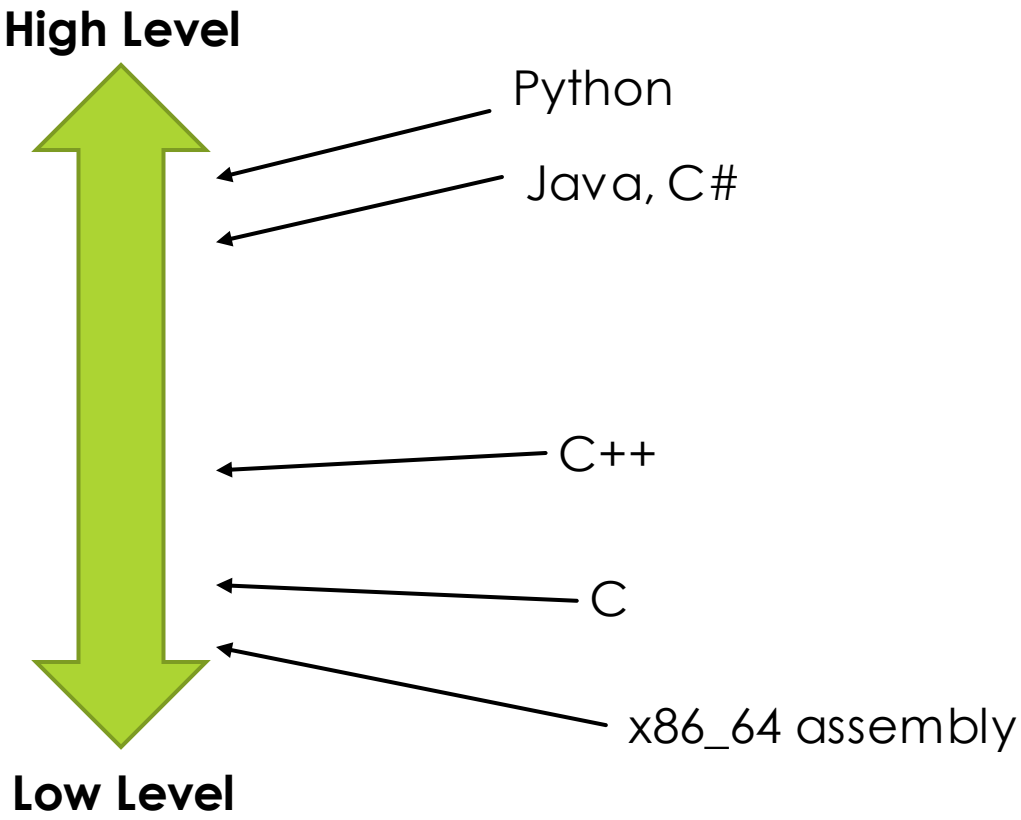
- A lot of the internet is polluted with people having pointless opinionated arguments about the merits of various programming languages.
- This principle is most clear when seen in the context of discussions of programming languages and frameworks.
- You could probably fill an entire library with the number of internet arguments people have had on topics such as "Java vs. C++" or "Python sucks, use Ruby on Rails", ad nauseum.

High & Low Level Languages

The higher 'level' a language is, the more you are removed from machine-level instructions.

Conversely, lower 'level' you are, the more the details of your machine's architecture start to become relevant.

'Higher' and 'lower' should not be seen as 'better' or 'worse', simply different programming contexts.

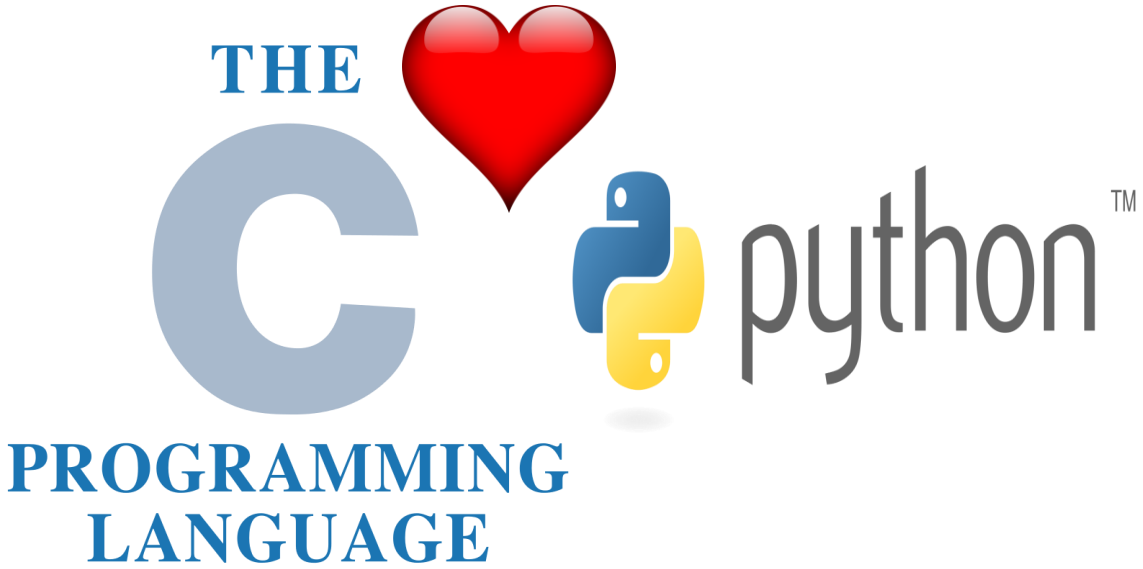


C vs Python?

C and Python: **Best Friends**

Two primary reasons:

1. **C** is the *implementation language* of **Python**
2. Both deal with very different problem spaces



How to improve performance?

Three approaches, from simplest to more advanced:

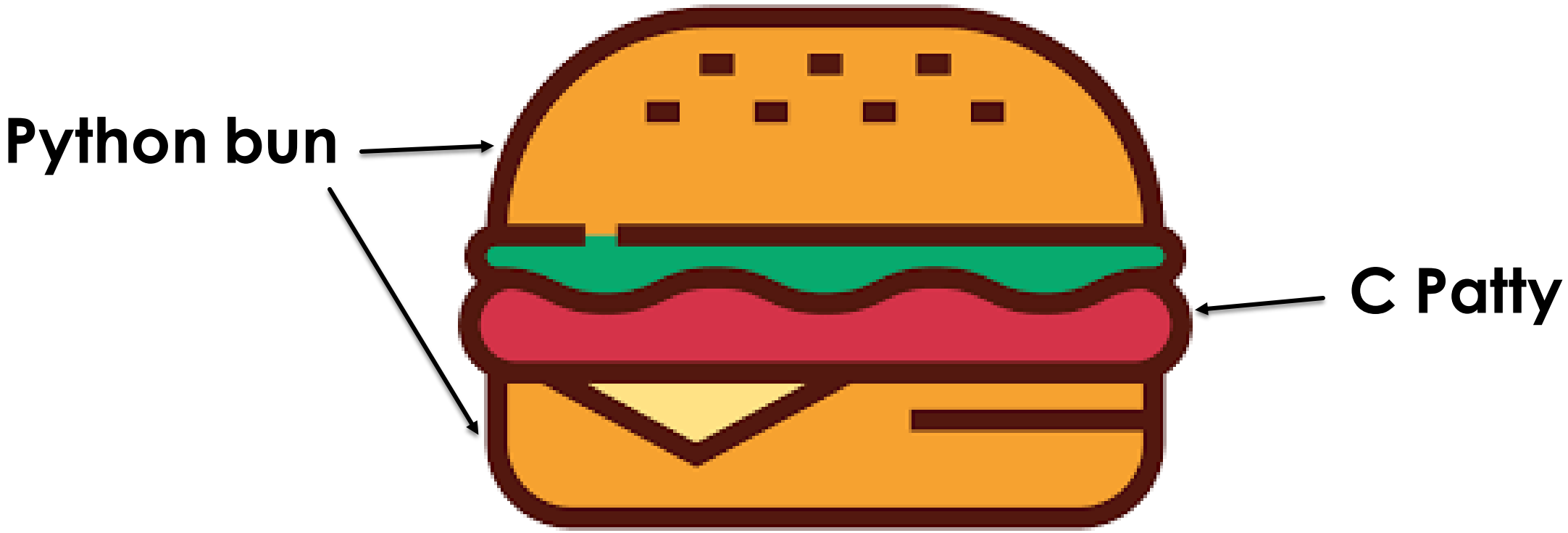
1. Use packages that have efficient C bindings
2. Find pre-written C code, compile it and call the binaries from Python
3. Write your own C code, make direct function calls to C from your Python code

Approach #1: Use efficient packages

Some important packages that use C:

- **numpy**
- **scipy**
- **NetCDF4**
- **grib2**
- **Proj4**

Approach #2: Call a C binary



5 minute break

Appendix: Cython/CPython confusion

CPython	• **Default implementation of Python** • **No performance gain**
Cython	• Compiled superset of Python • Performance gain • Think about the burger

Exercise: Let's call a C binary from Python!

1. https://www.github.com/danielhogg/cbin/releases

2. To begin, don't worry about compiling. Just find your OS (Windows/MacOS/Linux) and download the correct binary.

3. The download will also include an 18MB sample dataset -> 'sample_data.sbf'

Exercise: Hints

The **generate_sample** binary expects one (and only one) command line argument, which is the path of the sample_data.sbf file. For example on Windows it might be "C:\\Users\\YourName\\Downloads\\sample_data.sbf"

To call the binary from python, use either one of the following commands:

- os.system()
- subprocess.call()

- You should not have to write more than 10-15 lines of python to make this work!