Introduction to Advanced Python: Why is Python slow?

Kai Striega

November 8, 2023

Who am I?

- Self taught developer
- Work as a software developer at BHP
- Volunteer in the Free and Open Source community

Before I start

Before I start

Here be dragons

This talk introduces advanced topics, quickly. The content is designed to stretch your understanding and to challenge you.

Before I start

Here be dragons

This talk introduces advanced topics, quickly. The content is designed to stretch your understanding and to challenge you.

Questions

I am happy to take questions during the talk, feel free to ask if something doesn't make sense.

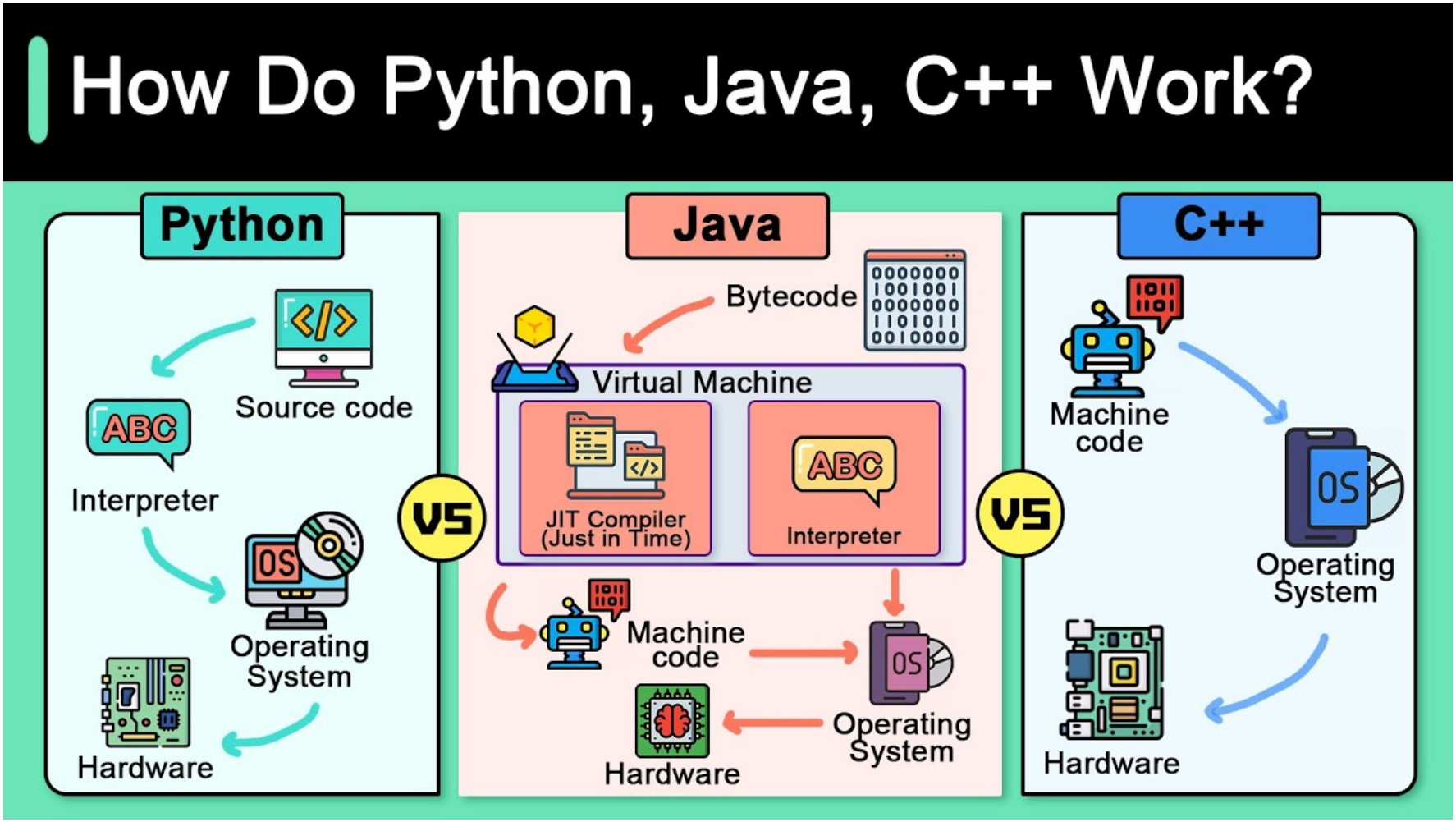
What we're going to do today

- Answer the question: Why is Python slow?
- Compare Python to C
- Discuss when execution time matters

Why C?

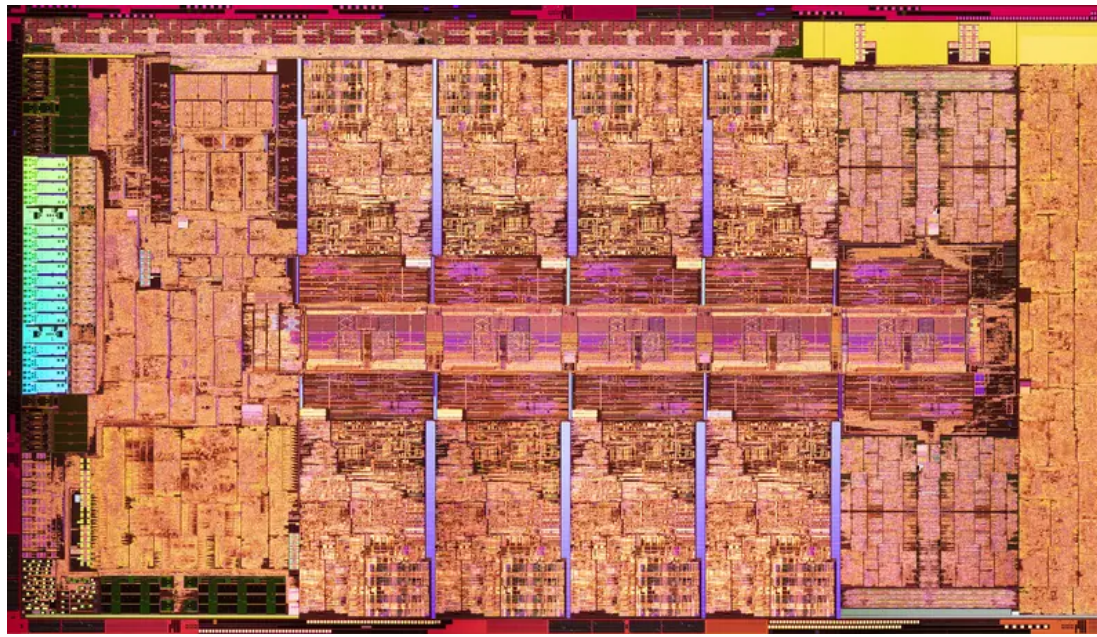
- C is *very* fast
- C is compiled
- CPython is written in C
- Many Python extensions are written in C

What is a *compiled* langauge



What the Central Processing Unit (CPU) does

- The CPU is responsible for executing instructions provided by software programs
- It performs calculations, manages data movement, and coordinates the operations of various hardware components
- It can be thought of as the "brain" of the computer



What we are measuring

- What does it mean for a programming language to be fast?

  Execution Speed is the amount of time taken for the program to complete a particular task
- Many other measures of performance exists, execution speed is the simplest to work with
- We are measuring how quickly the CPU can execute the instructions generated by that language

What limits the performance of the CPU?

Memory Bound How quickly we can get data to the CPU

CPU Bound How quickly the CPU can execute instructions

Today's Example

```
1 if __name__ == "__main__":
2     total = 0
3     limit = 6_074_000_999
4     for number in range(limit):
5         total += number
6     print(f"{total=}")
```

Is Python slow?

Is Python slow?

```
time python sum_of_first_n_numbers.py
```

Is Python slow?

```
time python sum_of_first_n_numbers.py

total=18446744064889498501
real 5m56.581s
user 5m56.518s
sys 0m0.004s
```

Today's Example, but in C

```c
#include <stdio.h>
#include <inttypes.h>

int main() {
    const uint64_t limit = 6074000999;
    uint64_t total = 0;
    for (uint64_t i = 0; i < limit; ++i) {
        total += i;
    }
    printf("total=%" PRIu64 "\n", total);
}
```

What about C?

What about C?

```
time bin/sum_of_first_n_numbers_O3
```

What about C?

```
time bin/sum_of_first_n_numbers_O3

total=18446744064889498501
real 0m0.000s
user 0m0.000s
sys 0m0.000s
```

Why?

- *0m0.000s* seems like a bug. . . it's not!
- C is a **compiled** langauge
- The compiler can perform optimizations on the code

Optimising Compilers

- Our problem is always going to have the same answer
- The compiler is able to notice this
- And calculates the result ahead of time
- So how do we get around this?

What if we don't use optimizations?

What if we don't use optimizations?

```
time bin/sum_of_first_n_numbers_arg_O0
```

What if we don't use optimizations?

```
time bin/sum_of_first_n_numbers_arg_O0

total=18446744064889498501
real 0m2.020s
user 0m2.010s
sys 0m0.010s
```

What we've done so far

- We've turned a 6-minute problem into a 2-second problem
- By changing the language
- This speedup exists even without optimizations

Why?

- Interpreted
- Dynamically Typed
- Global Interpreter Lock
- Language Design
- Flexibility
- Memory Organisation

Memory Latencies

	Latency
L1 Cache	$<$1ns
L2 Cache	4ns
Main Memory	100ns
SSD	16 μs
Round Trip (Datacenter)	0.5ms
Round Trip (US to Europe)	150 ms

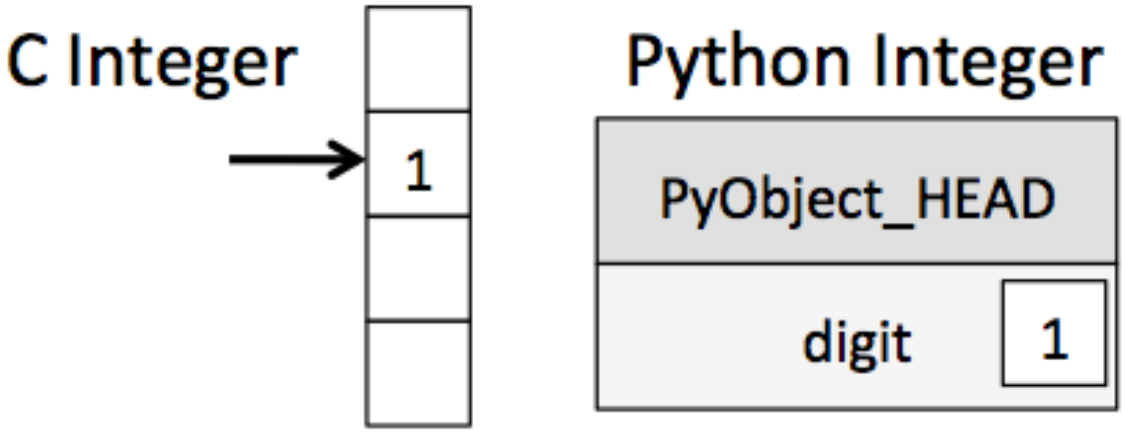
Memory Latencies

	Latency
L1 Cache	<1 second
L2 Cache	4 seconds
Main Memory	100 seconds
SSD	4.5 hours
Round Trip (Datacenter)	5.8 days
Round Trip (US to Europe)	5 years

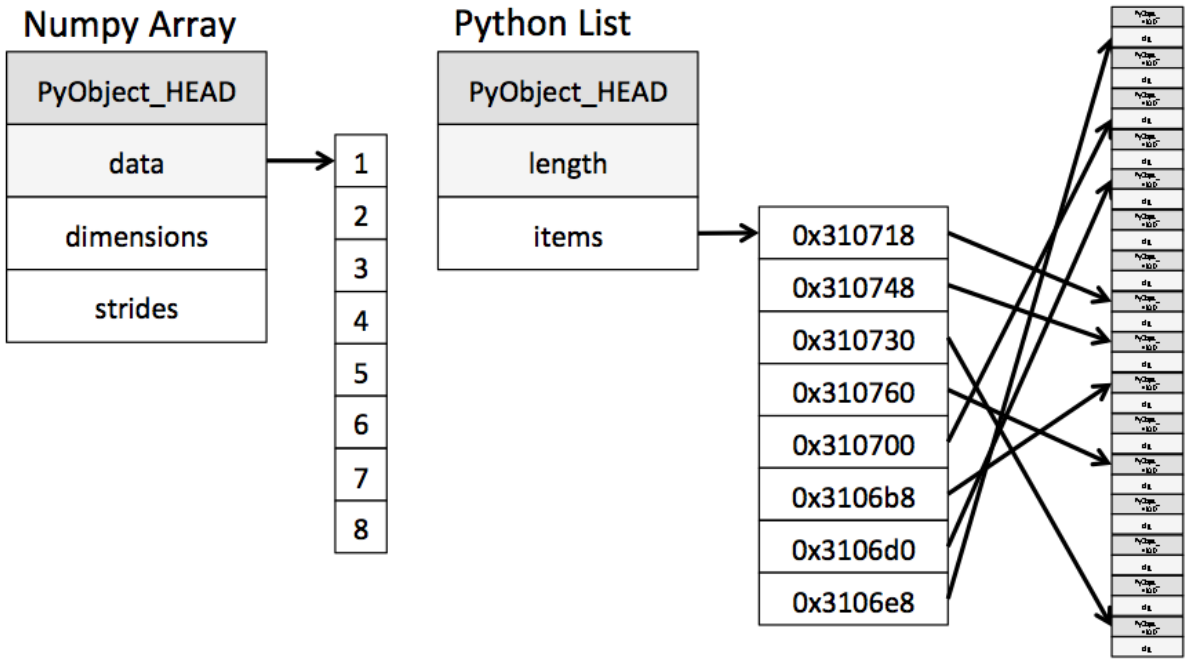
PyObjects and memory access

- In Python *everything* is an object.
- In the source code, this is implemented as a *PyObject*
- A *PyObject* can be thought of as a wrapper around some data and associated metadata

A Python int is more than a C int



This extends to more complex data structures



Performance Considerations

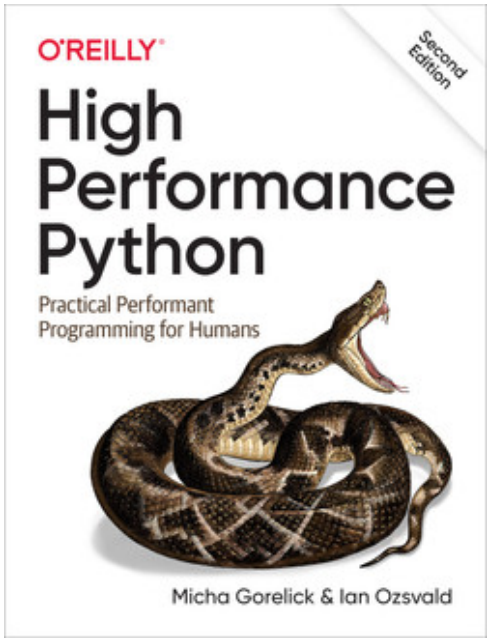
- Want to get data from the *fastest* possible source
- Usually that will be a *cache* or *memory* (RAM)
- Want to avoid network access as much as possible

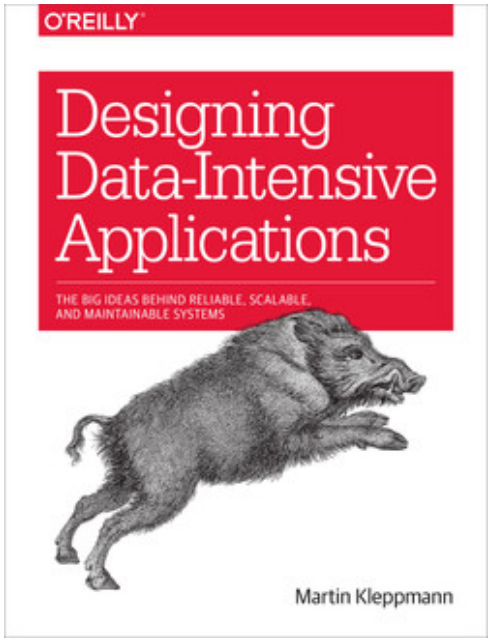
Python's slowness doesn't matter

- The number of problems where Python is *truly* the bottleneck is very small
- Developers are expensive, Python is quick to develop with
- Many problems can be solved by throwing more hardware at them (horizontally scalable)
- Many problems are bound by network time, not CPU time

Further Recommendations

- https://www.oreilly.com/library/view/high-performance-python/9781492055013/
- https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/
- https://bytebytego.com/





ByteByteGo