CS 576 – Systems Security
Introduction to C/C++ Software Security

Georgios (George) Portokalidis

Why Care about C/C++?

- Software in C/C++ is necessary because…
  - ..it is performant
  - …it facilitates communication with or control of the hardware
- A lot other languages still use components in C/C++
- We have inherited a lot of (legacy) software coded in these languages

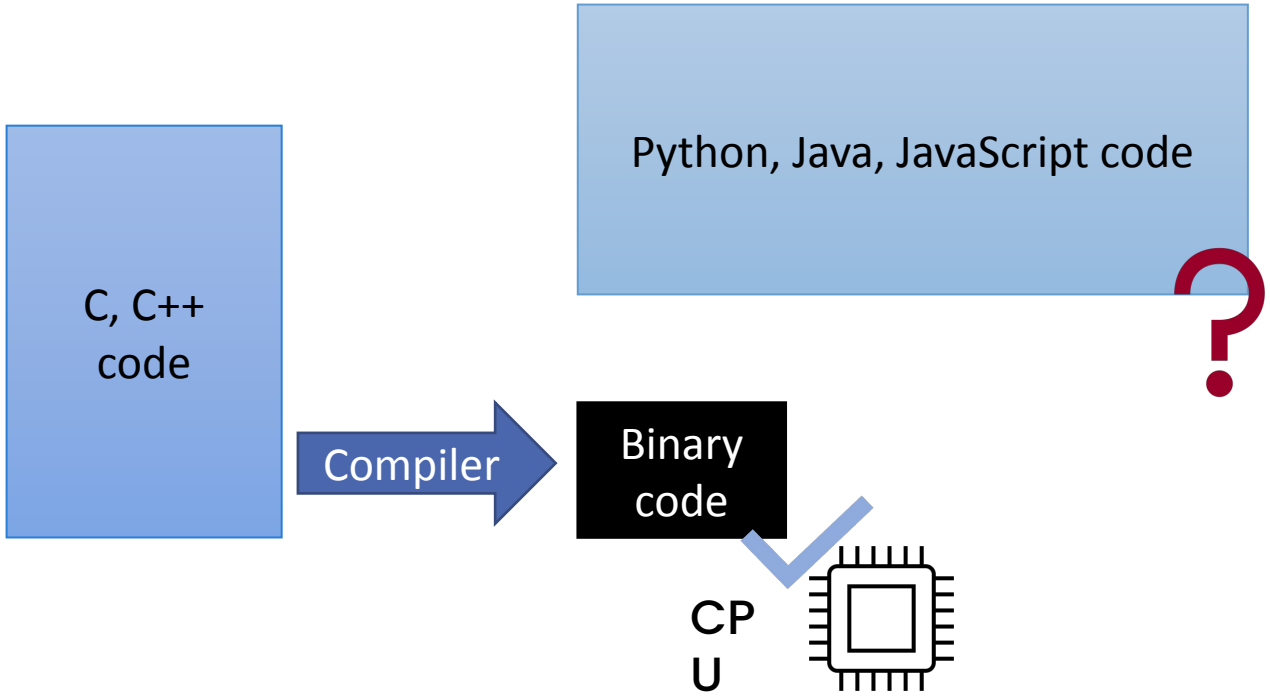
Top Programming Languages 2020

Rank	Language	Type	Score
1	Python		100.0
2	Java		95.3
3	C		94.6
4	C++		87.0
5	JavaScript		79.5
6	R		78.6
7	Arduino		73.2
8	Go		73.1
9	Swift		70.5
10	Matlab		68.4

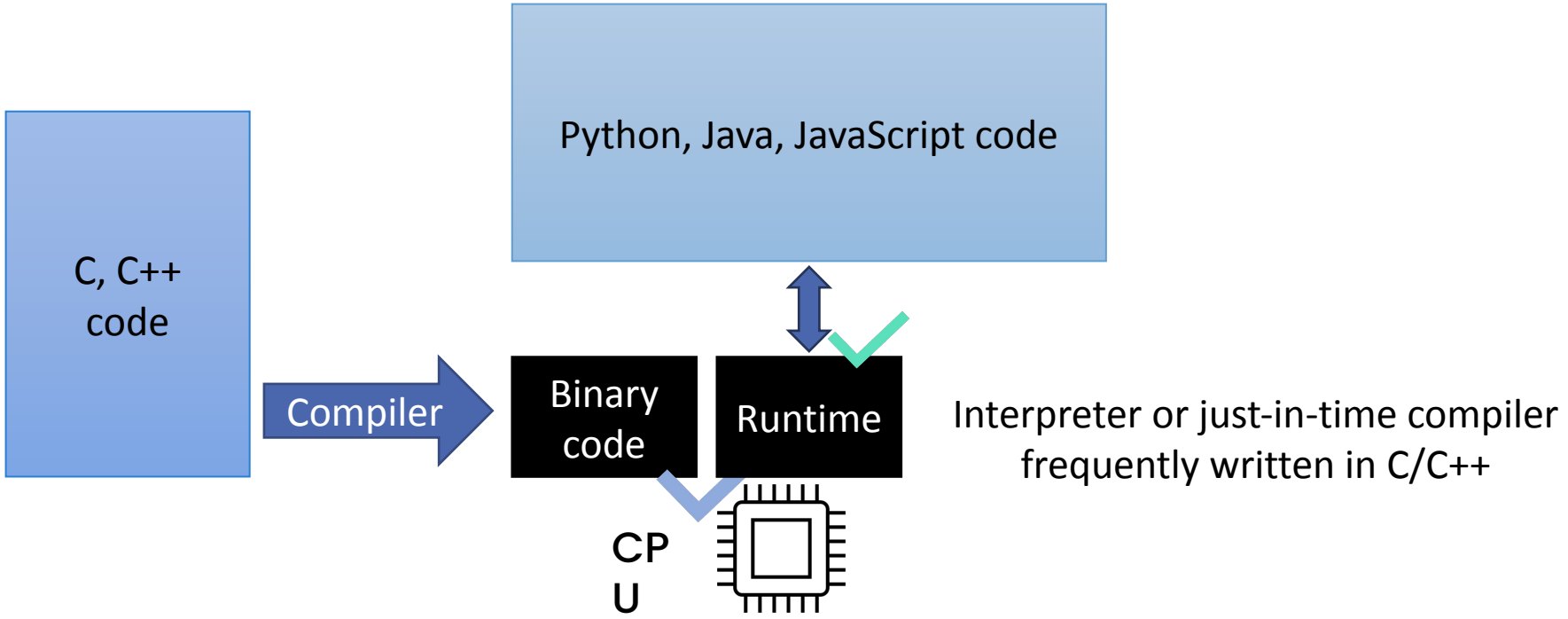
https://spectrum.ieee.org/at-work/tech-careers/top-programming-language-2020

Source: IEEE Spectrum

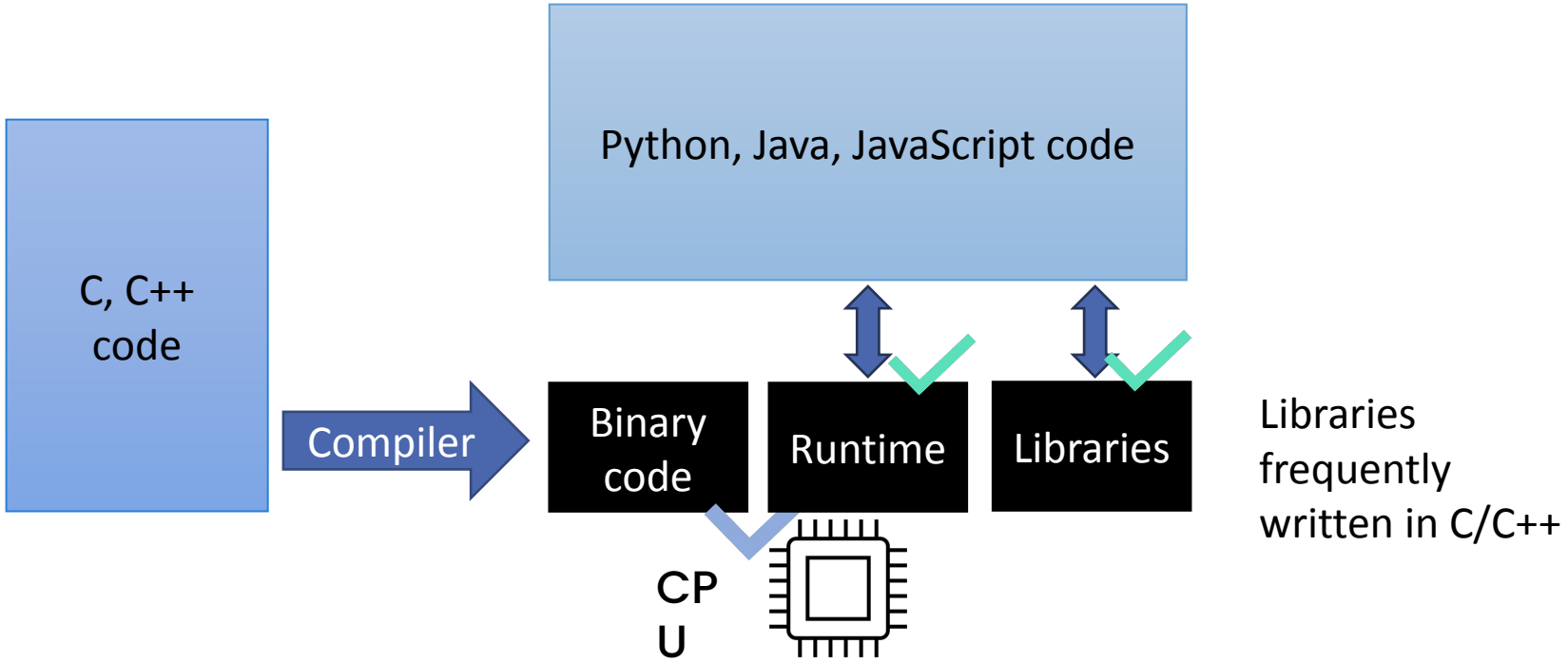
CPUs Only Understand Binary



CPUs Only Understand Binary

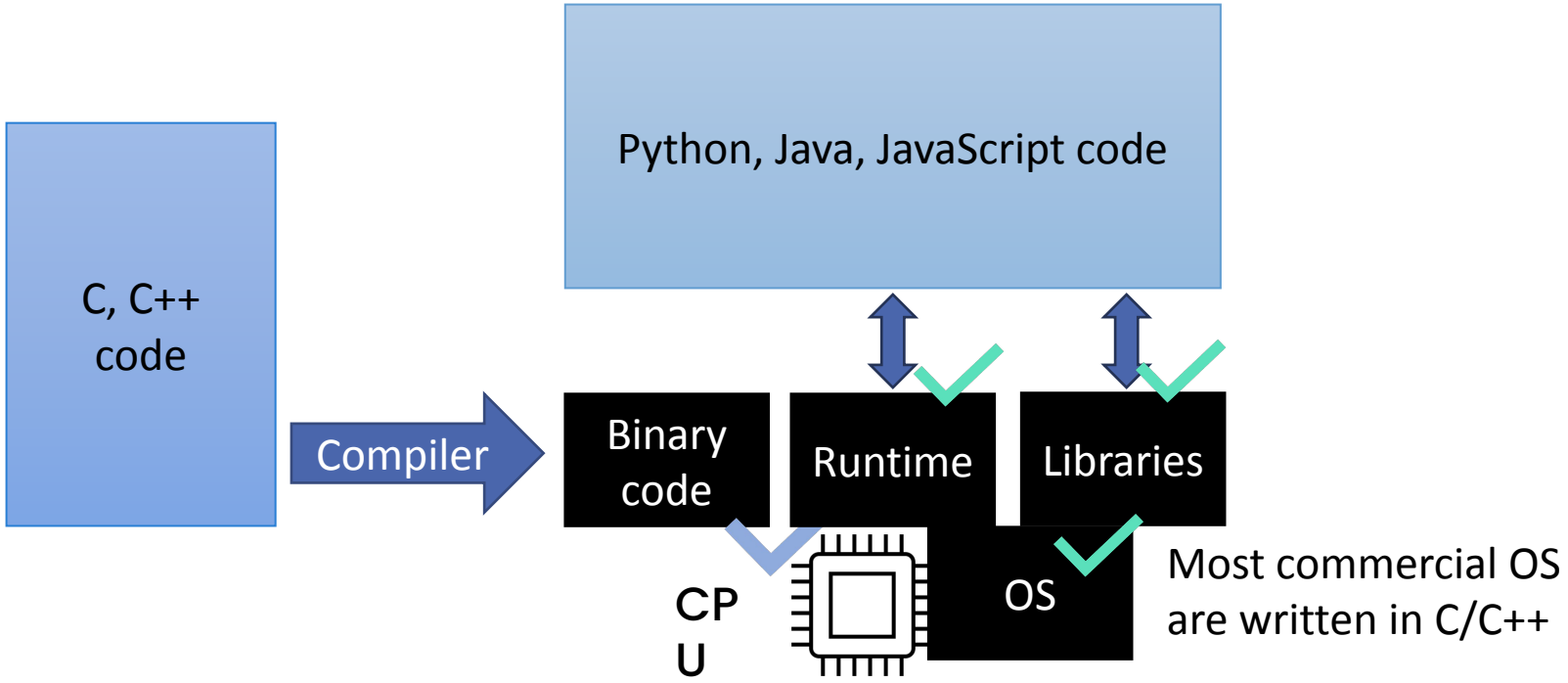


CPUs Only Understand Binary

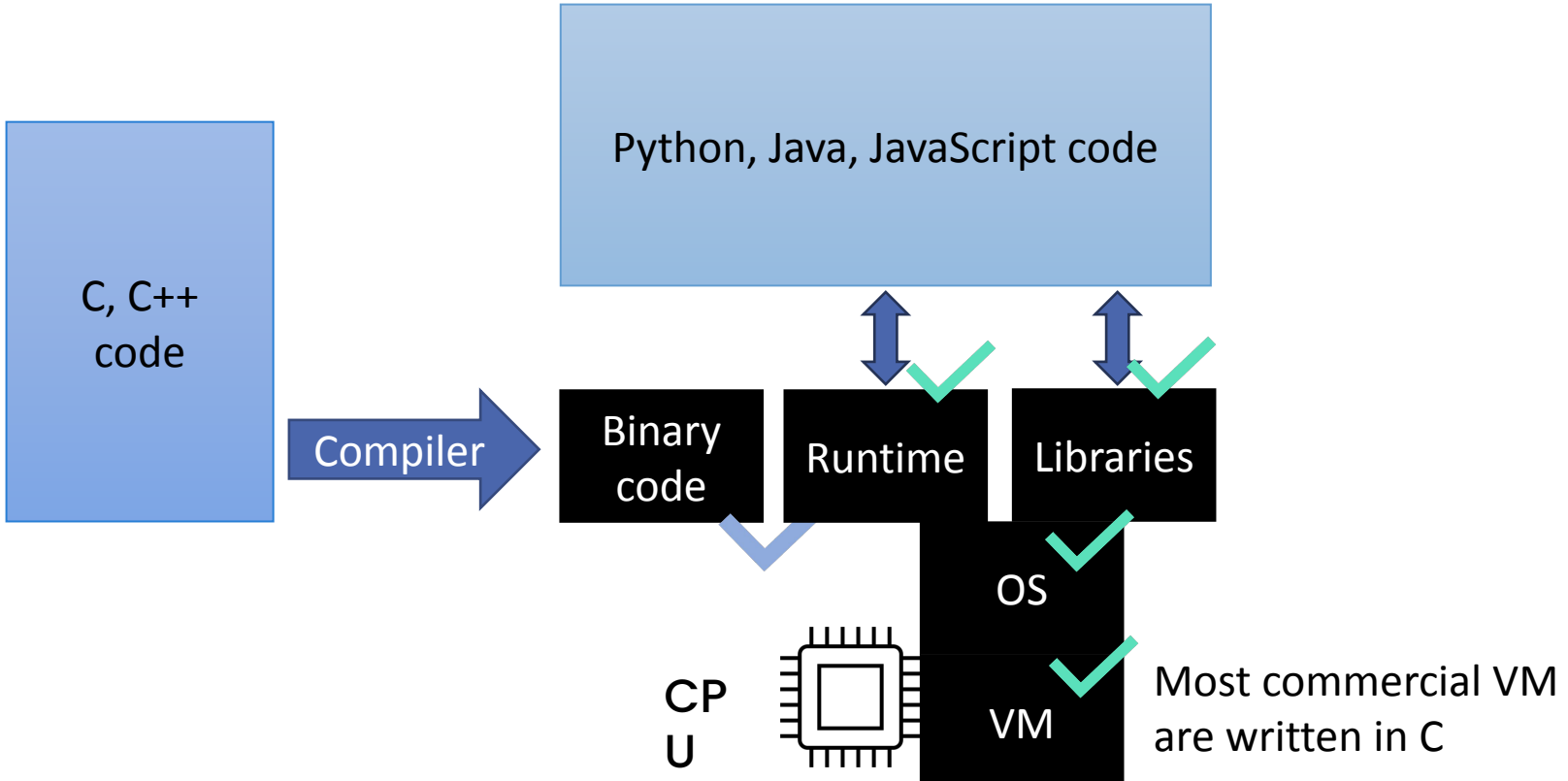


Libraries frequently written in C/C++

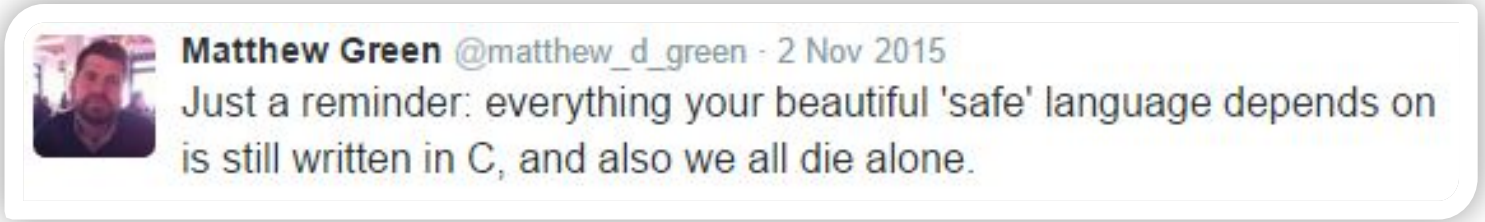
CPUs Only Understand Binary

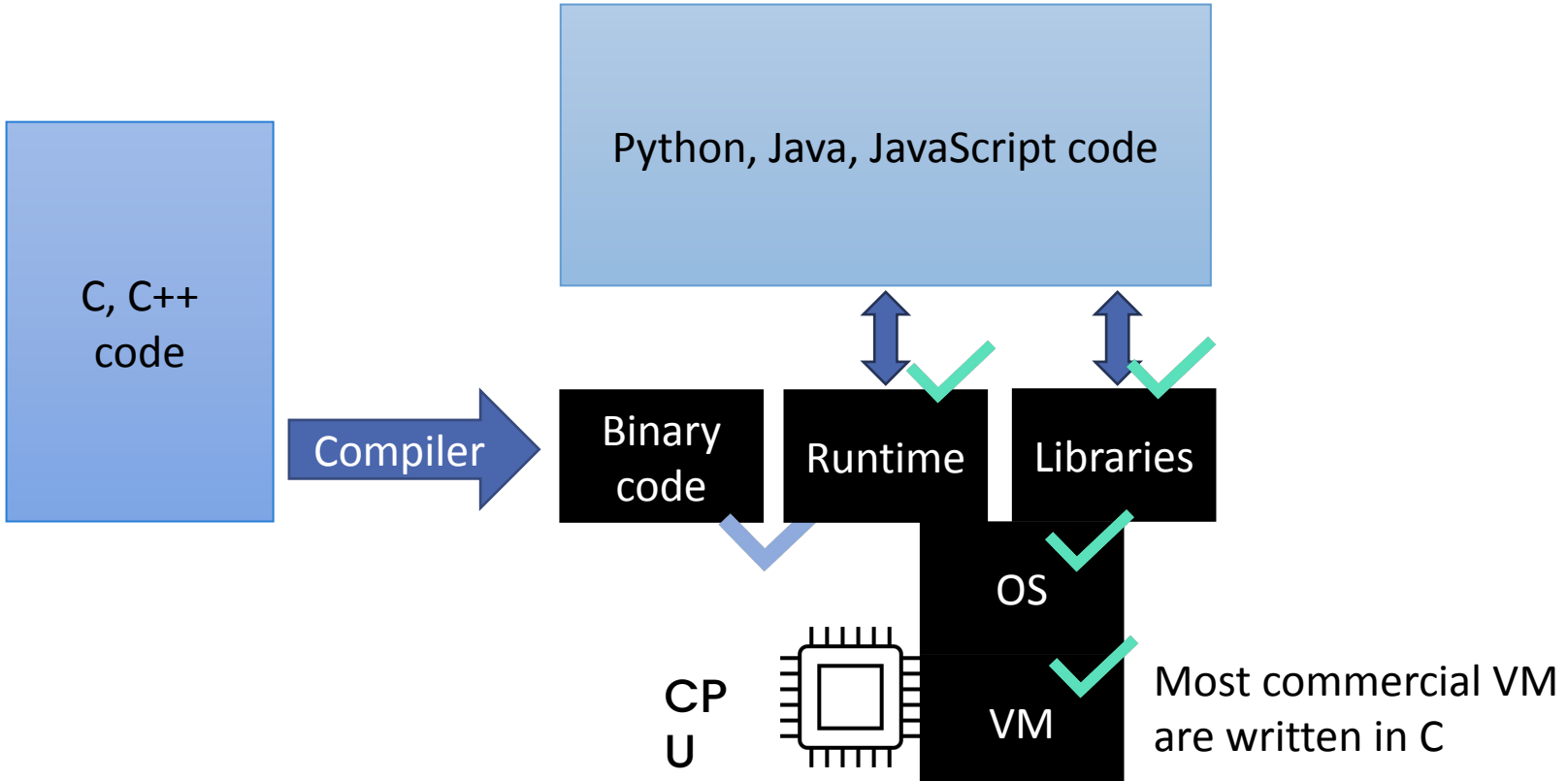


CPUs Only Understand Binary



CPUs Only Understand Binary





Comparing Languages

C/C++

- Compiles to machine code
- Typed but weakly enforced
- Low-level memory access
- User manages memory

Python, Perl, PHP

- Dynamically typed (duck type)
  - Types are checked for suitability at run time
- Strong typed
  - Operations are checked for safety
- Interpreted
  - PHP now also uses JIT
- Automatic memory management

Java, C#

- Java
  - Compiles to bytecode, run by the Java virtual machine
    - Initially interpreted, quickly just-in-time translated
- C#
  - Mix of compile and JIT
- Type safe and strongly typed
- Automatic memory management
- Implicit memory access

Comparing Languages

C/C++

- Compiles to machine code
- **Typed but weakly enforced**
- **Low-level memory access**
- **User manages memory**

Java, C#

- Java

he Java virtual

time translated

- Type safe and strongly typed
- Automatic memory management
- Implicit memory access

- C/C++ are not memory safe languages
- Most of the responsibility for creating correct and secure code falls to the developer

Python, Perl, PHP

- Dynamically typed (duck type)
  - Types are checked for suitability at run time
- Strong typed
  - Operations are checked for safety
- Interpreted
  - PHP now also uses JIT
- Automatic memory management

Developer Error Example

- What happens when line 6 executes?

```
1.  void foo()
2.  {
3.      int a;
4.          char buffer[4];
5.          …
6.          buffer[4] = ‘A’;
7.          …
8.  }
```

Developer Error Example

- What happens when line 6 executes?

```
1.  void foo()
2.  {
3.      int a;
4.          char buffer[4];
5.          …
6.          buffer[4] = ‘A’;
7.          …
8.  }
```

This is classified as ”undefined behavior”

Whatever you guessed may be correct

We Are Going to Learn About

- Prevalent defects in C/C++ programs: overflows, format strings, temporal bugs, and other memory errors
- Exploitation techniques: code injection, code-reuse, data-only attacks
  - We learn offense to better understand
    - how effective defenses are
    - to be able to design better defenses
    - the actual risk facing a software system
- Defenses that harden (mitigate exploitation or eliminate bug class) in C/C++ programs
  - Existing: ASLR, canaries, DEP, etc.
  - Recent and upcoming: CFI, CET, etc.
  - Not just abstractions, but also **mechanisms**