

# Modern C++ Programming

## 1. INTRODUCTION

---

*Federico Busato*

2024-03-29

## Table of Contents

**1 A Little History of C/C++ Programming Language**

**2 Areas of Application and Popularity**

**3 C++ Philosophy**

**4 C++ Weaknesses**

- C++ Alternatives
- Why Switching to a New Language is Hard?

**5 The Course**

*"When recruiting research assistants, I look at grades as the last indicator. I find that **imagination, ambition, initiative, curiosity, drive**, are far better predictors of someone who will do useful work with me. Of course, these characteristics are themselves correlated with high grades, but there is something to be said about a student who decides that a given course is a waste of time and that he works on a side project instead."*

*Breakthroughs don't happen in regular scheduled classes, they happen in side projects. We want people who complete the work they were assigned, but **we also need people who can reflect critically on what is genuinely important"***

*Daniel Lemire, Prof. at the University of Quebec*

### **Academic excellence is not a strong predictor of career excellence**

*“Across industries, research shows that the correlation between grades and job performance is modest in the first year after college and trivial within a handful of years...”*

*“Academic grades rarely assess qualities like creativity, leadership and teamwork skills, or social, emotional and political intelligence. Yes, straight-A students master cramming information and regurgitating it on exams. But career success is rarely about finding the right solution to a problem — it’s more about finding the right problem to solve...”*

*"Getting straight A's requires conformity. **Having an influential career demands originality.***

*This might explain why Steve Jobs finished high school with a 2.65 G.P.A., J.K. Rowling graduated from the University of Exeter with roughly a C average, and the Rev. Dr. Martin Luther King Jr. got only one A in his four years at Morehouse*

*If your goal is to graduate without a blemish on your transcript, you end up taking easier classes and staying within your comfort zone. If you're willing to tolerate the occasional B... **You gain experience coping with failures and setbacks, which builds resilience**"*

*"Straight-A students also miss out socially. More time studying in the library means less time to start lifelong friendships, join new clubs or volunteer...Looking back, I don't wish my grades had been higher. If I could do it over again, I'd study less"*

**Adam Grant, the New York Times**

*"Got a 2.4 GPA my first semester in college. Thought maybe I wasn't cut out for engineering. Today I've landing two spacecraft on Mars, and designing one for the moon.*

*STEM is hard for everyone. Grades ultimately aren't what matters.  
**Curiosity and persistence matter"***

***Ben Cichy**, Chief Software Engineer,  
NASA Mars Science Laboratory*

*“And programming computers was so fascinating. You create your own little universe, and then it does what you tell it to do”*

*Vint Cerf, TCP/IP co-inventor and Turing Award*

*“Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program”*

*Linus Torvalds, principal developer of the Linux kernel*

*“You might not think that programmers are artists, but programming is an extremely creative profession. It's logic-based creativity”*

*John Romero, co-founder of id Software*

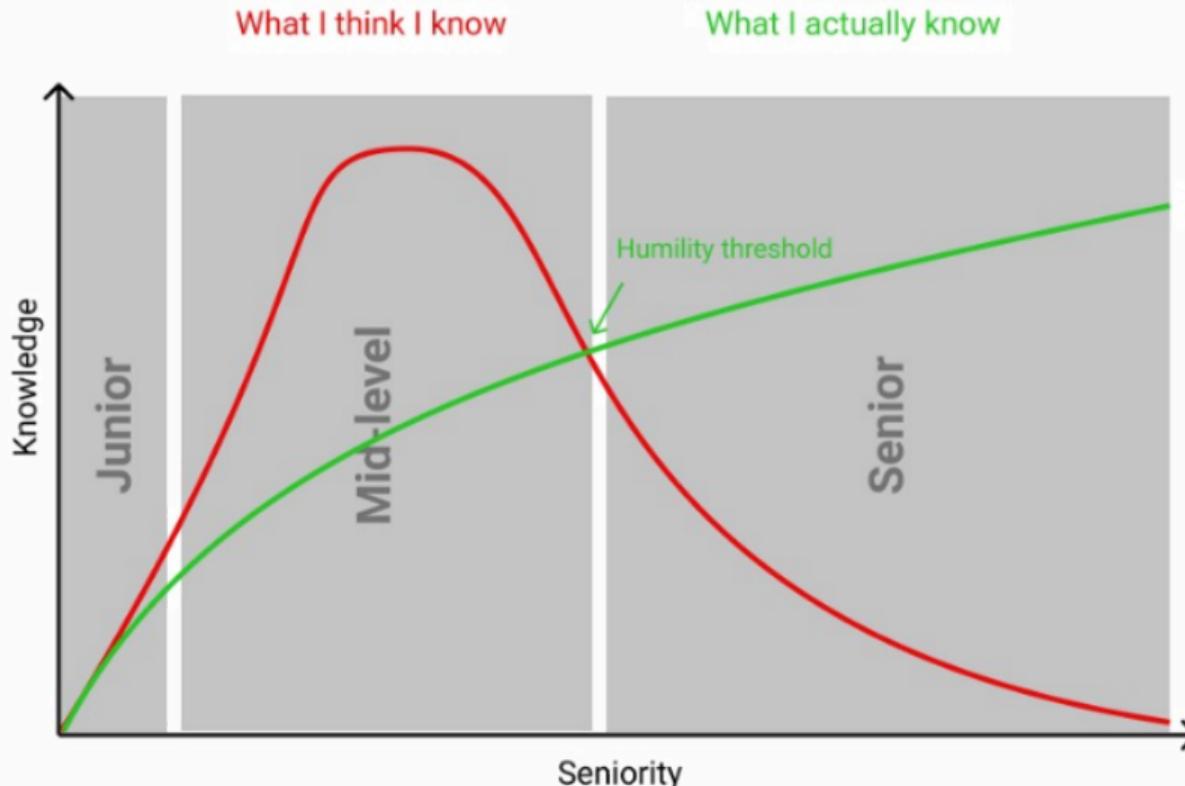
**Creativity** Programming is extremely creative. The ability to perceive the problem in a novel way, provide new and original solutions. Creativity allows recognizing and generating alternatives

**Form of Art** Art is the expression of human creative skills. Every programmer has his own style. Codes and algorithms show elegance and beauty in the same way as painting or music

**Learn** Programming gives the opportunity to learn new things every day, improve own skills and knowledge

**Challenge** Programming is a challenge. A challenge against yourself, the problem, and the environment

# Knowledge-Experience Relation



*“In software development, learning is not a big part of the job.  
**It is the job.**”*

*Woody Zuill*

# A Little History of C/C++ Programming Language

---

# The Assembly Programming Language



A long time ago, in a galaxy far,  
far away....there was **Assembly**

- Extremely simple instructions
- Requires lots of code to do simple tasks
- Can express anything your computer can do
- Hard to read, write
- ...redundant, boring programming, bugs proliferation

```
main:  
.Lfunc_begin0:  
    push rbp  
.Lcfi0:  
.Lcfi1:  
    mov rbp, rsp  
.Lcfi2:  

```

In the 1969 **Dennis M. Ritchie** and **Ken Thompson** (AT&T, Bell Labs) worked on developing an operating system for a large computer that could be used by a thousand users. The new operating system was called **UNIX**

The whole system was still written in assembly code. Besides assembler and Fortran, UNIX also had an interpreter for the **programming language B**. A high-level language like B made it possible to write many pages of code task in just a few lines of code. In this way the code could be produced much faster than in assembly

A drawback of the B language was that it did not know data-types (everything was expressed in machine words). Another functionality that the B language did not provide was the use of “structures”. The lack of these things formed the reason for Dennis M. Ritchie to develop the **programming language C**. In 1988 they delivered the final standard definition ANSI C



Dennis M. Ritchie and Ken Thompson

```
#include "stdio.h"

int main() {
    printf("Hello World\n");
}
```

## Areas of Application:

- UNIX operating system
- Computer games
- Due to their power and ease of use, C were used in the programming of the special effects for Star Wars

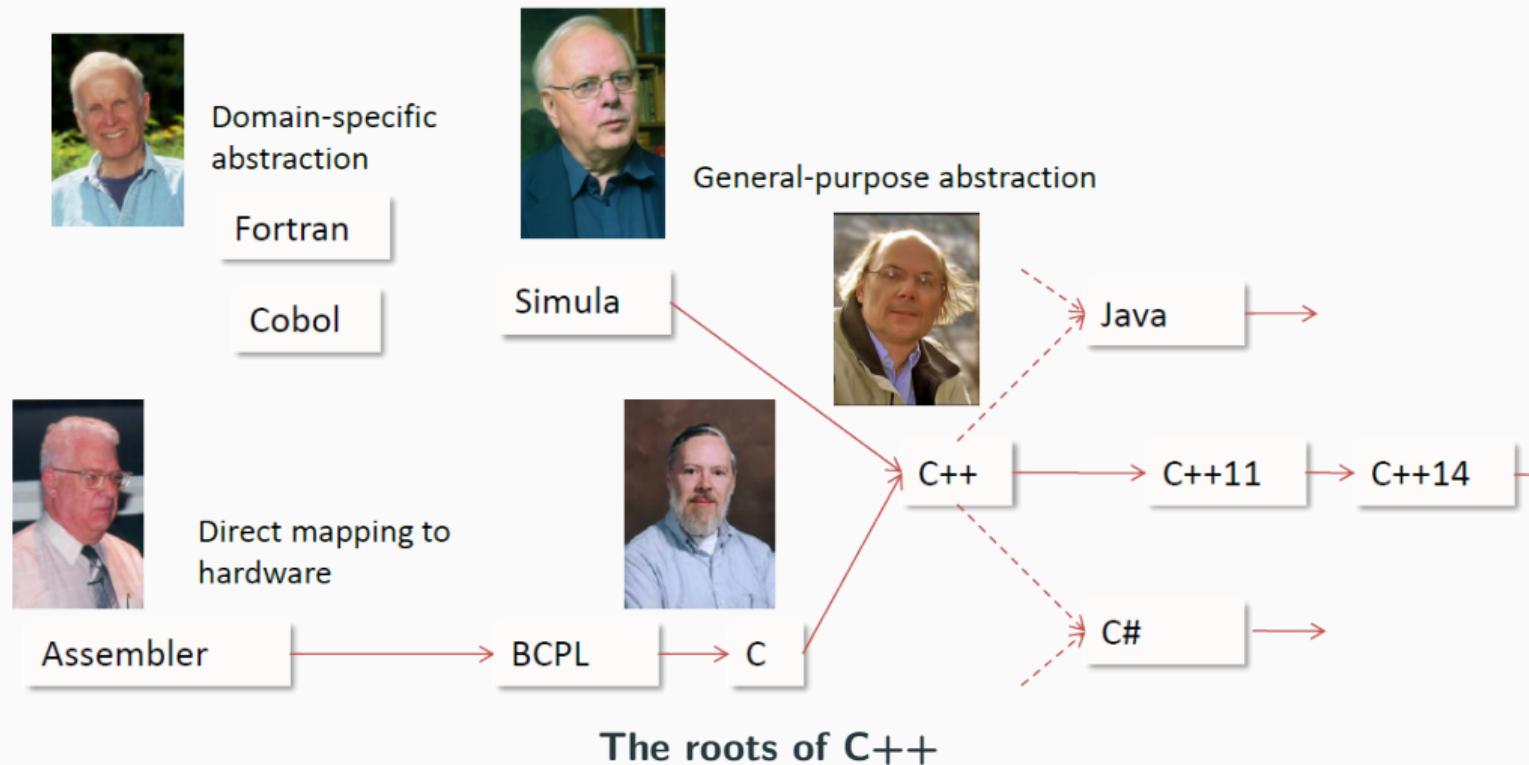


Star Wars - The Empire Strikes Back

The **C++ programming language** (originally named “C with Classes”) was devised by **Bjarne Stroustrup** also an employee from Bell Labs (AT&T). Stroustrup started working on C with Classes in 1979. (The ++ is C language operator)

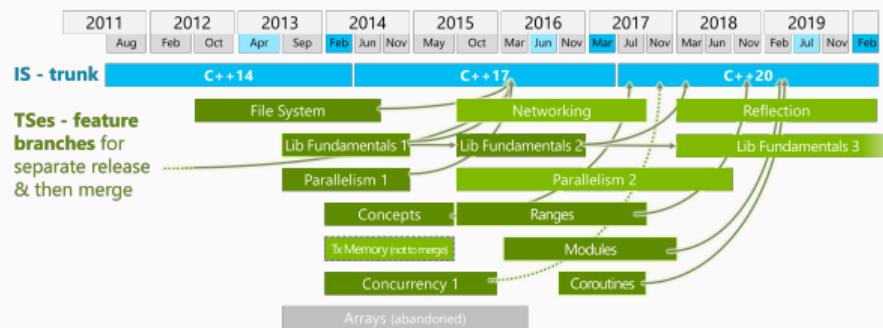
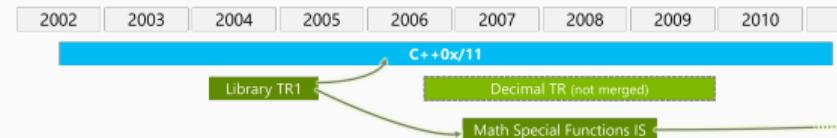
The first commercial release of the C++ language was in October 1985





# A Little History of C++

3/3



## About Evolution

*“If you’re teaching today what you were teaching five years ago, either the field is dead or you are”*

**Noam Chomsky**



# **Areas of Application and Popularity**

---

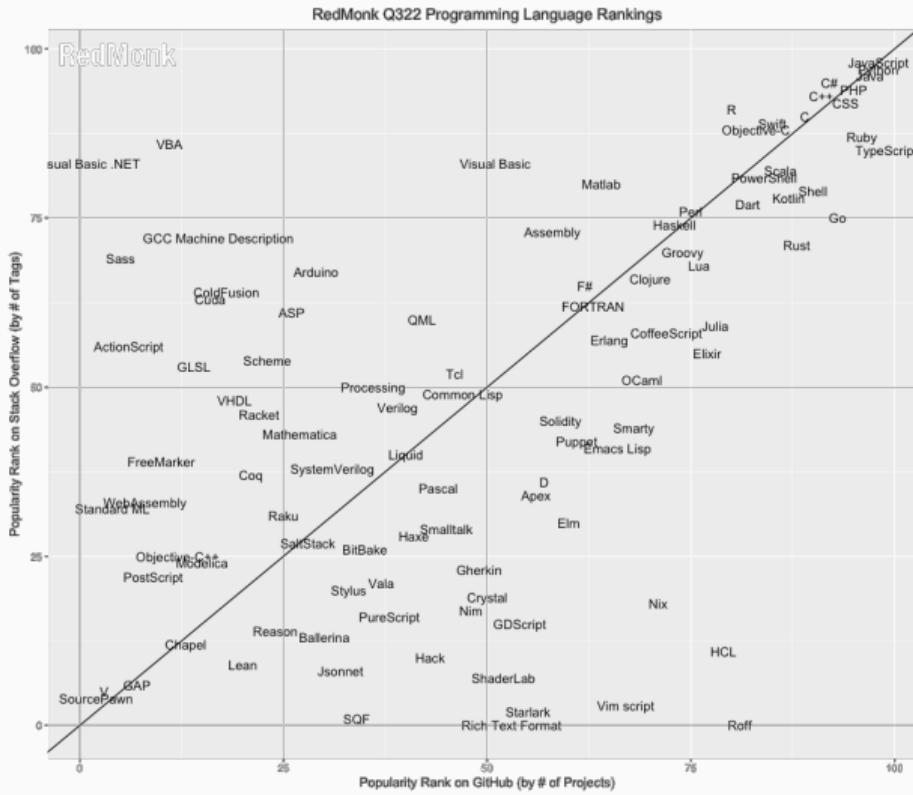
# Most Popular Programming Languages (IEEE Spectrum - 2022)

Rank	Language	Type	Score
1	Python	🌐💻⚙️	100.0
2	Java	🌐📱💻	95.4
3	C	📱💻⚙️	94.7
4	C++	📱💻⚙️	92.4
5	JavaScript	🌐	88.1
6	C#	🌐📱💻⚙️	82.4
7	R	💻	81.7
8	Go	🌐💻	77.7

# Most Popular Programming Languages (TIOBE - December. 2022)

Programming Language	Ratings	Change
 Python	16.66%	+3.76%
 C	<b>16.56%</b>	<b>+4.77%</b>
 C++	11.94%	+4.21%
 Java	11.82%	+1.70%
 C#	4.92%	-1.48%
 Visual Basic	3.94%	-1.46%
 JavaScript	3.19%	+0.90%

# Most Popular Programming Languages (Redmonk - June, 2022)



**There may be more than 200 billion lines  
of C/C++ code globally**

- **Performance is the defining aspect of C++.** No other programming language provides the performance-critical facilities of C++
- **Provide the programmer control over every aspect of performance**
- **Leave no room for a lower level language**

- ***Ubiquity.*** C++ can run from a low-power embedded device to large-scale supercomputers
- ***Multi-Paradigm.*** Allow writing efficient code without losing high-level abstraction
- ***Allow writing low-level code.*** Drivers, kernels, assembly (asm), etc.
- ***Ecosystem.*** Many support tools such as debuggers, memory checkers, coverage, static analysis, profiling, etc.
- ***Maturity.*** C++ has a 40 years history. Many software problems have been already addressed and developing practices have been investigated

- **Operating systems:** Windows, Android, OS X, Linux
- **Compilers:** LLVM, Swift compiler
- **Artificial Intelligence:** TensorFlow, Caffe, Microsoft Cognitive Toolkit
- **Image Editing:** Adobe Premier, Photoshop, Illustrator
- **Web browser:** Firefox, Chrome, etc. + WebAssembly
- **High-Performance Computing:** drug developing and testing, large scale climate models, physic simulations
- **Embedded systems:** IoT, network devices (e.g. GSM), automotive
- Google and Microsoft use C++ for web indexing

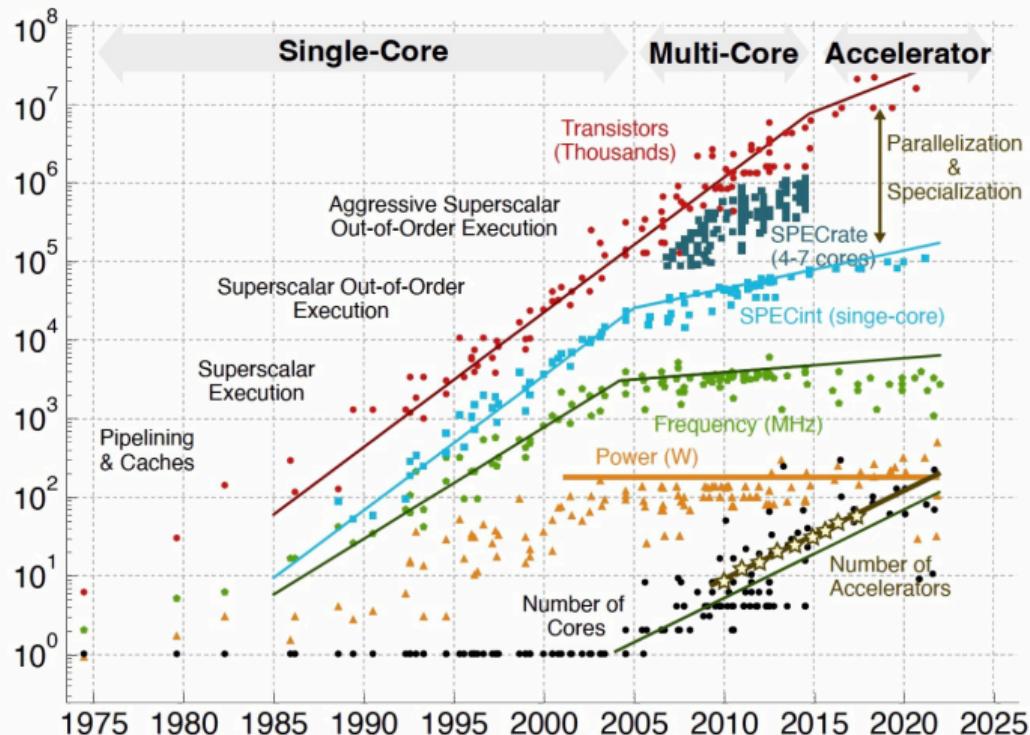
- **Scientific Computing:** CERN/NASA\*, SETI@home, Folding@home
  - **Database:** MySQL, ScyllaDB
  - **Video Games:** Unreal Engine, Unity
  - **Entertainment:** Movie rendering (see Interstellar black hole rendering), virtual reality
  - **Finance:** electronic trading systems (Goldman, JPMorgan, Deutsche Bank)\*\*
- ... and many more**

---

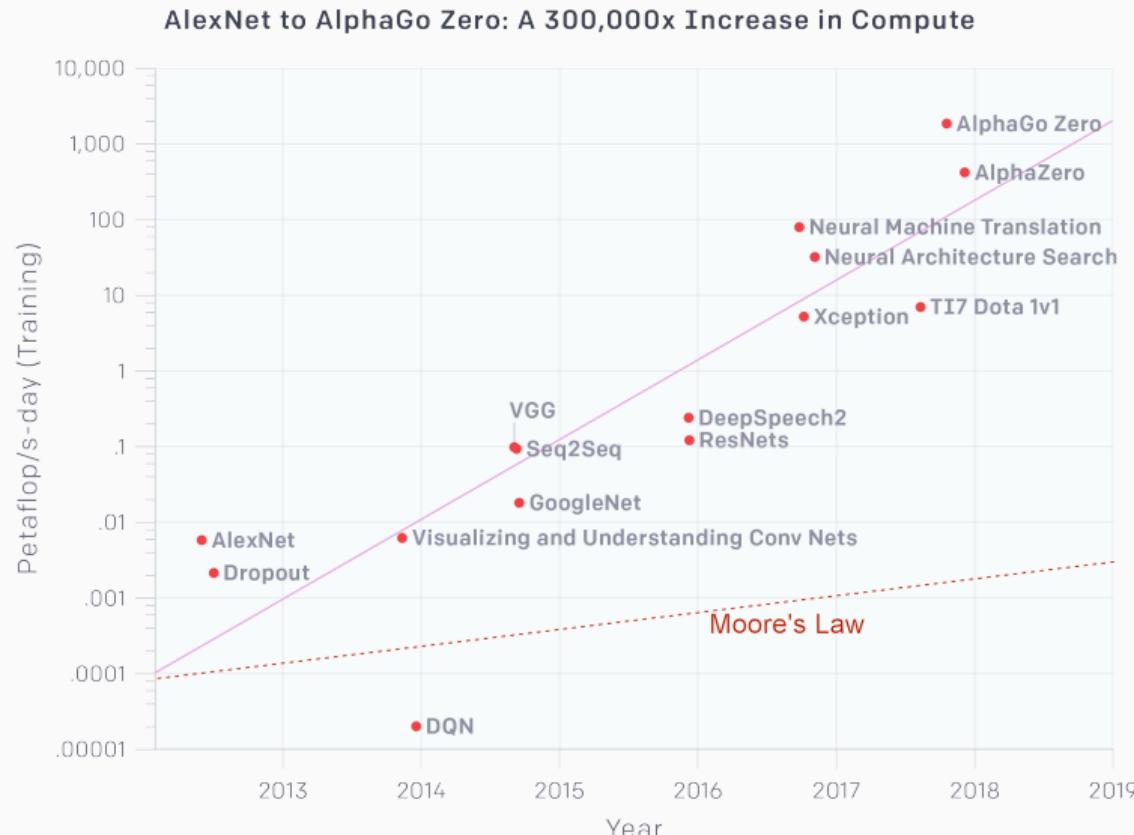
\* The flight code of the NASA Mars drone for the **Perseverance** Mission, as well as the **Webb telescope** software, are mostly written in C++ [github.com/nasa/fprime](https://github.com/nasa/fprime), James Webb Space Telescope's Full Deployment

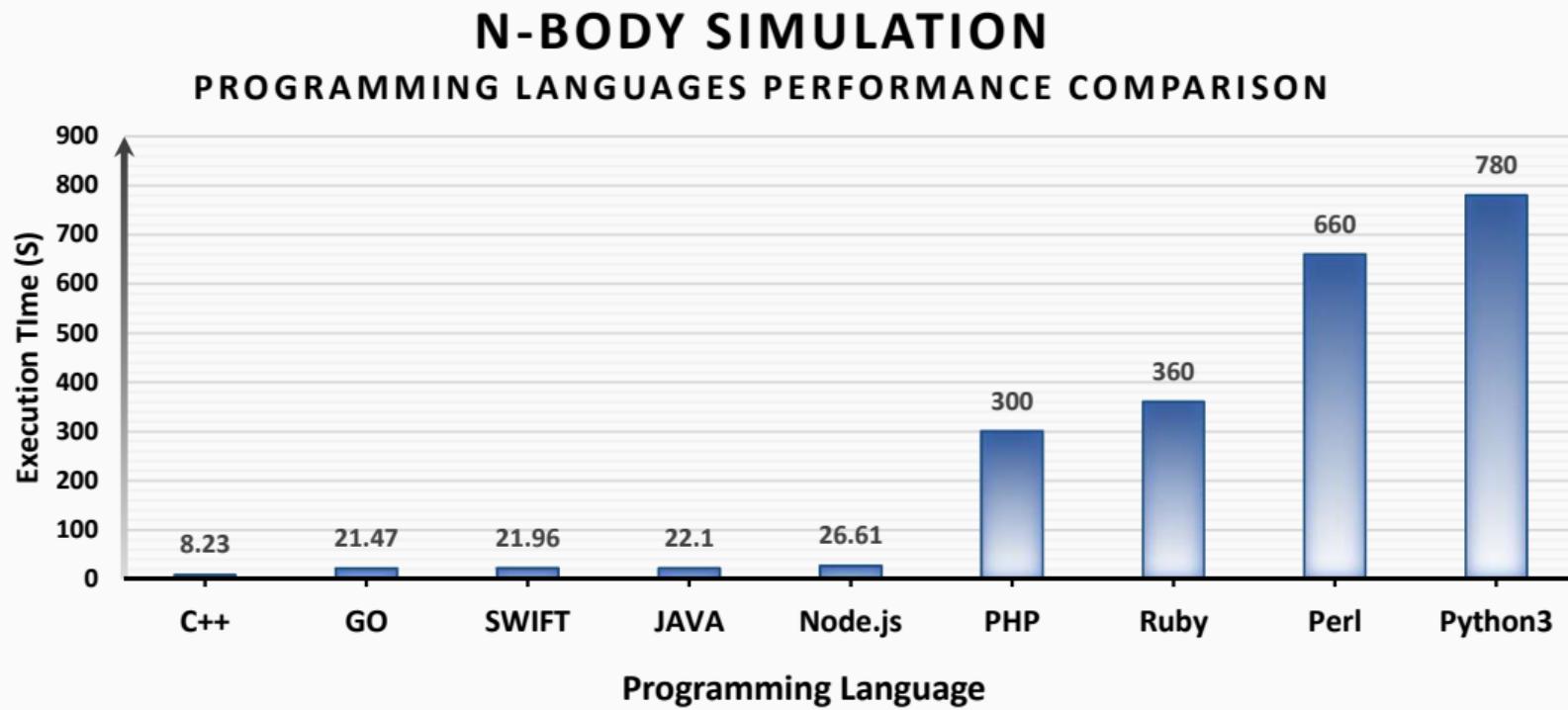
# Why C++ is so Important?

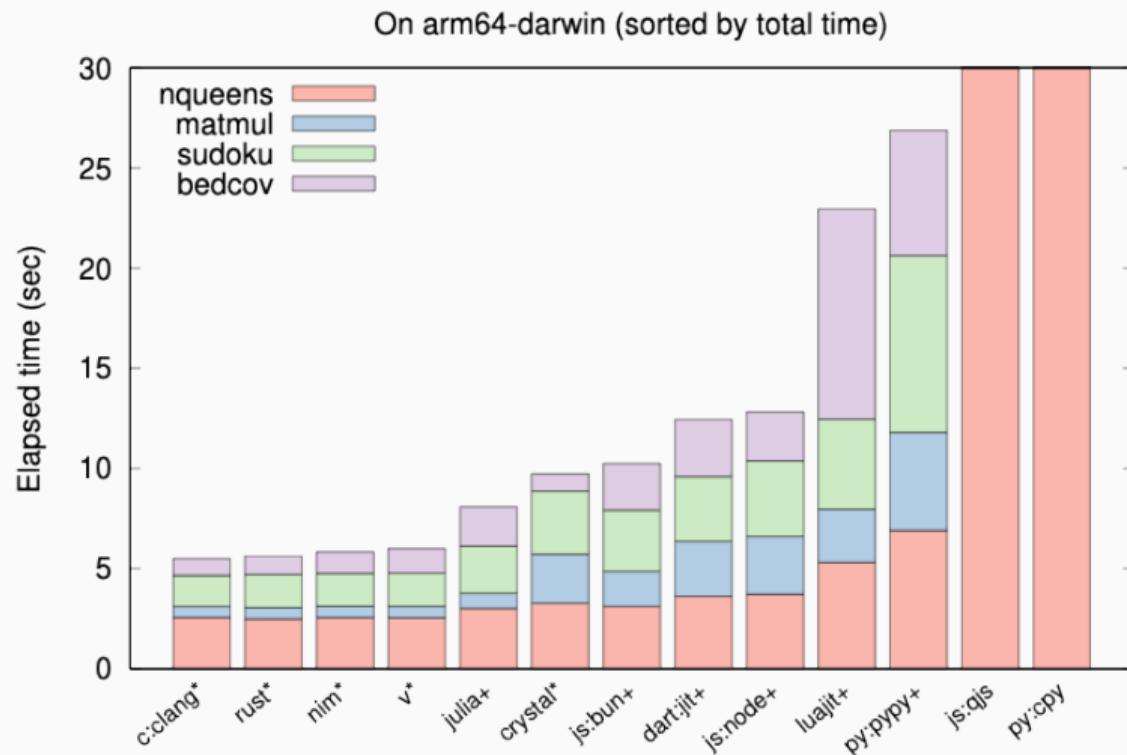
## The End of Historical Performance Scaling

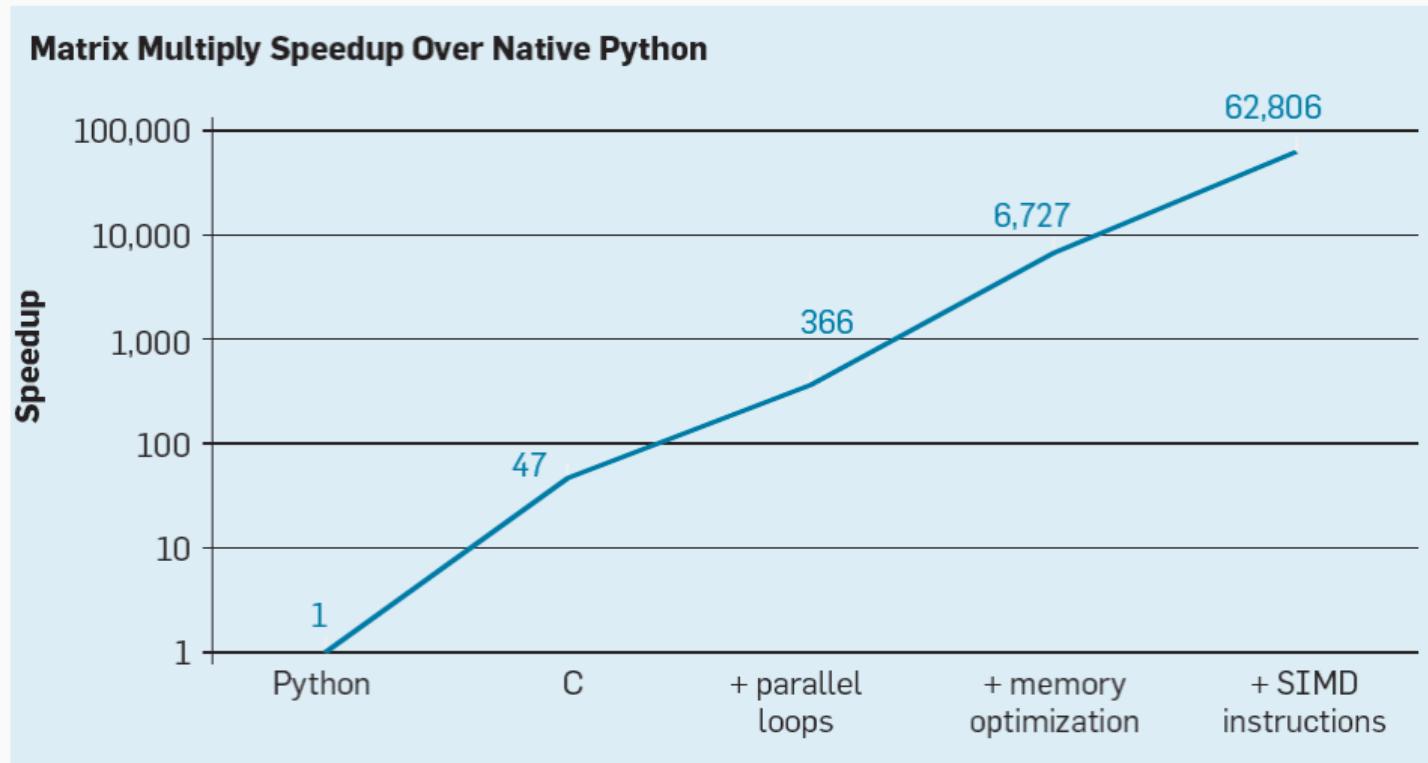


# An Important Example... (AI Evolution)





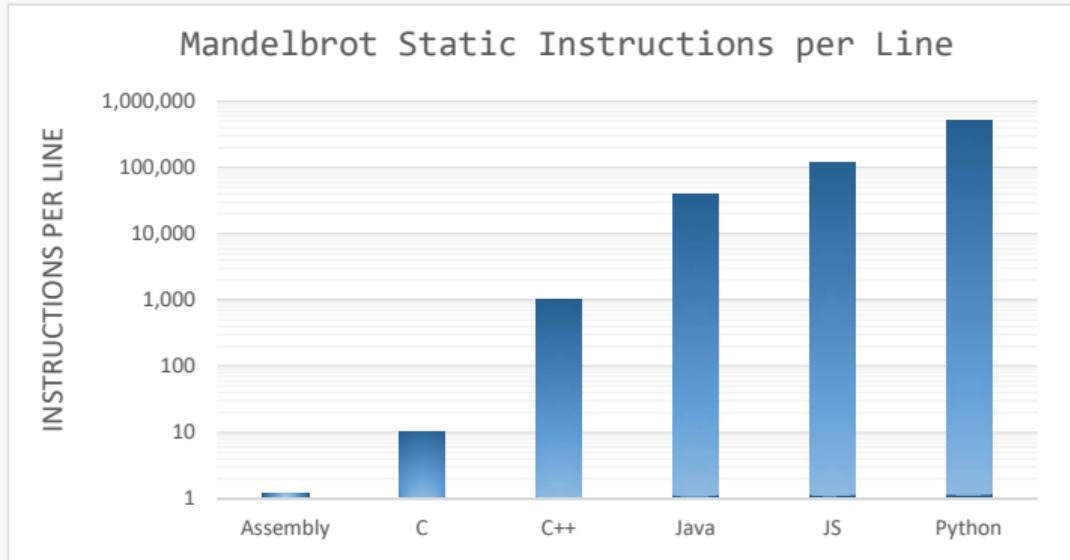
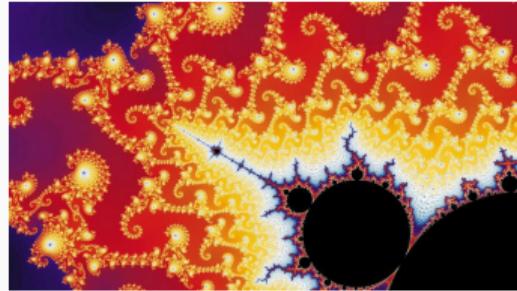




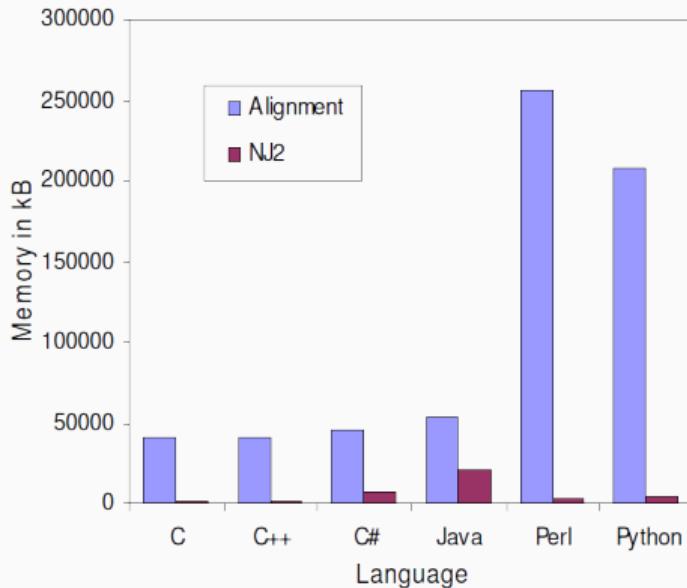
## Hello World

Language	Execution Time
C (on my machine)	0.7 ms
C	2 ms
Go	4 ms
Crystal	8 ms
Shell	10 ms
Python	78 ms
Node	110 ms
Ruby	150 ms
jRuby	1.4 s

# Performance/Expressiveness Trade-off



# Memory Usage



Memory usage comparison of the  
Neighbor-Joining and global alignment programs

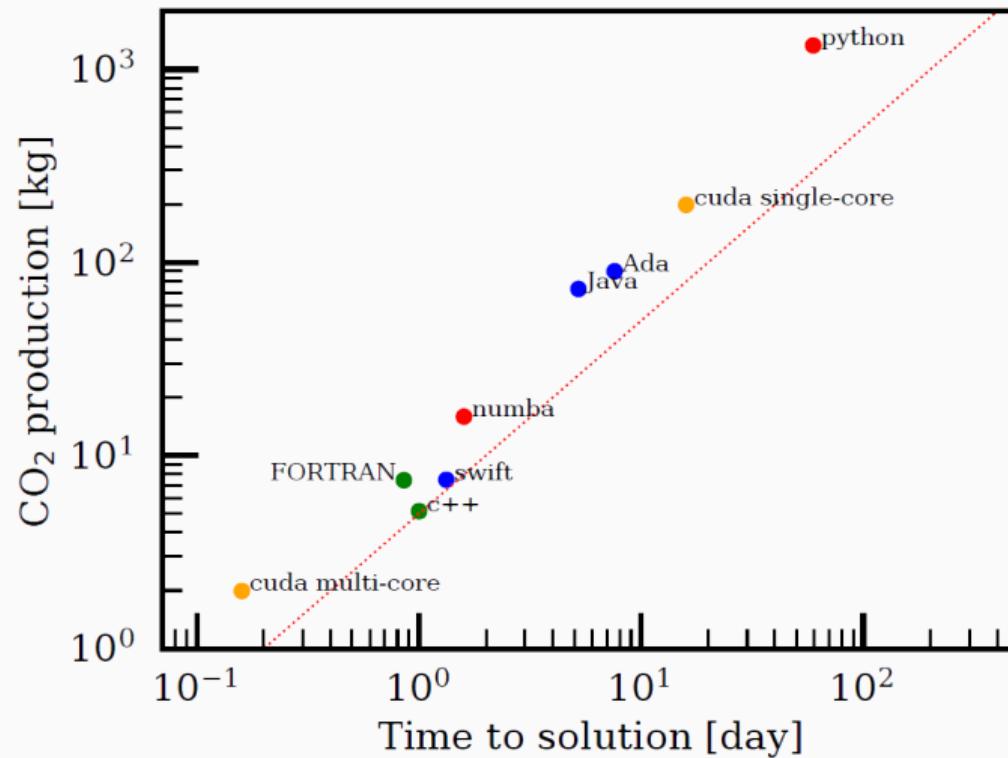
---

A comparison of common programming languages used in bioinformatics (BMC Informatic)

# Energy Efficiency

	Energy		Time
(c) C	1.00	(c) C	1.00
(c) Rust	1.03	(c) Rust	1.04
(c) C++	1.34	(c) C++	1.56
(c) Ada	1.70	(c) Ada	1.85
(v) Java	1.98	(v) Java	1.89
(c) Pascal	2.14	(c) Chapel	2.14
(c) Chapel	2.18	(c) Go	2.83
(v) Lisp	2.27	(c) Pascal	3.02
(c) Ocaml	2.40	(c) Ocaml	3.09
(c) Fortran	2.52	(v) C#	3.14
(c) Swift	2.79	(v) Lisp	3.40
(c) Haskell	3.10	(c) Haskell	3.55
(v) C#	3.14	(c) Swift	4.20
(i) Hack	24.02	(i) PHP	27.64
(i) PHP	29.30	(v) Erlang	36.71
(v) Erlang	42.23	(i) Jruby	43.44
(i) Lua	45.98	(i) TypeScript	46.20
(i) Jruby	46.54	(i) Ruby	59.34
(i) Ruby	69.91	(i) Perl	65.79
(i) Python	75.88	(i) Python	71.90
(i) Perl	79.58	(i) Lua	82.91

# CO<sub>2</sub> Production



# C++ Philosophy

---

*Do not sacrifice **performance** except as a last resort*

## **Zero Overhead Principle** (zero-cost abstraction)

*"it basically says if you have an abstraction it should not cost anything compared to write the equivalent code at lower level"*

*"so I have say a matrix multiply it should be written in a such a way that you could not drop to the C level of abstraction and use arrays and pointers and such and run faster"*

**Bjarne Stroustrup**

36/56

*Enforce **safety at compile time** whenever possible*

## **Statically Typed Language**

*"The C++ compiler provides type safety and catches many bugs at compile time instead of run time (a critical consideration for many commercial applications.)"*

[www.python.org/doc/FAQ.html](http://www.python.org/doc/FAQ.html)

- The *type annotation* makes the code more readable
- Promote compiler optimizations and runtime efficiency
- Allow users to define their own type system

- **Programming model:** *compartmentalization*, only add features if they solve an actual problem, and allow *full control*
- **Predictable runtime** (under constraints): no garbage collector, no dynamic type system → *real-time systems*
- **Low resources:** low memory and energy consumption → *restricted hardware platforms*
- **Well suited for static analysis** → *safety critical software*
- **Portability** → Modern C++ standards are highly portable

## Who is C++ for?

*“C++ is for people who want to use hardware very well and manage the complexity of doing that through abstraction”*

**Bjarne Stroustrup**

*“a language like C++ is not for everybody. It is generated via sharp and effective tool for professional basically and definitely for people who aim at some kind of precision”*

**Bjarne Stroustrup**

## Suggested Introduction Video



# C++ Weaknesses

---

... and why teaching C++ as first programming language is a bad idea?

C++ is the hardest language from students to master

- *More languages in one*
  - Standard C/C++ programming
  - Preprocessor
  - Object-Oriented features
  - Templates and Meta-Programming
- *Huge set of features*
- *Worry about memory management*
- *Low-level implementation details*: pointer arithmetic, structure, padding, undefined behavior, etc.
- *Frustrating*: compiler/runtime errors (e.g. seg. fault)

*"C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows your whole leg off"*

**Bjarne Stroustrup**, Creator of the C++ language

*"The problem with using C++...is that there's already a strong tendency in the language to require you to know everything before you can do anything"*

**Larry Wall**, Creator of the Perl language

*"Despite having 20 years of experience with C++, when I compile a non-trivial chunk of code for the first time without any error or warning, I am suspicious. It is not, usually, a good sign"*

**Daniel Lemire**, Prof. at the University of Quebec

## Backward-compatibility

“**Dangerous defaults and constructs**, often originating from C, cannot be removed or altered”

“Despite the hard work of the committee, **newer features sometimes have flaws that only became obvious after extensive user experience**, which cannot then be fixed”

“C++ practice has put an **ever-increasing cognitive burden** on the developer for what I feel has been very little gain in productivity or expressiveness and at a huge cost to code clarity”

C++ critics and replacements:

- Epochs: a backward-compatible language evolution mechanism
- Goals and priorities for C++
- Carbon Language
- Circle C++ Compiler
- Cppfront: Can C++ be 10x simpler & safer ... ?

## C++ Alternatives: Rust

Rust (1.0, 2015) has been Stack Overflow's most loved language for eight years in a row. Rust focuses on performance and zero-abstraction overhead as C++. It is designed to prevent many vulnerabilities that affect C++, especially memory bugs, enforcing constraints at compile type. In addition, it promotes cross-platform compatibility

*“First-time contributors to Rust projects are about 70 times less likely to introduce vulnerabilities than first-time contributors to C++ projects”*

*Tracey et al.*<sup>1</sup>

---

<sup>1</sup> Grading on a Curve: How Rust can Facilitate New Contributors while Decreasing Vulnerabilities

- CISA, NSA: The Case for Memory Safe Roadmap
- Octoverse: The Fastest Growing Languages
- Secure by Design: Google’s Perspective on Memory Safety

## C++ Alternatives: Zig

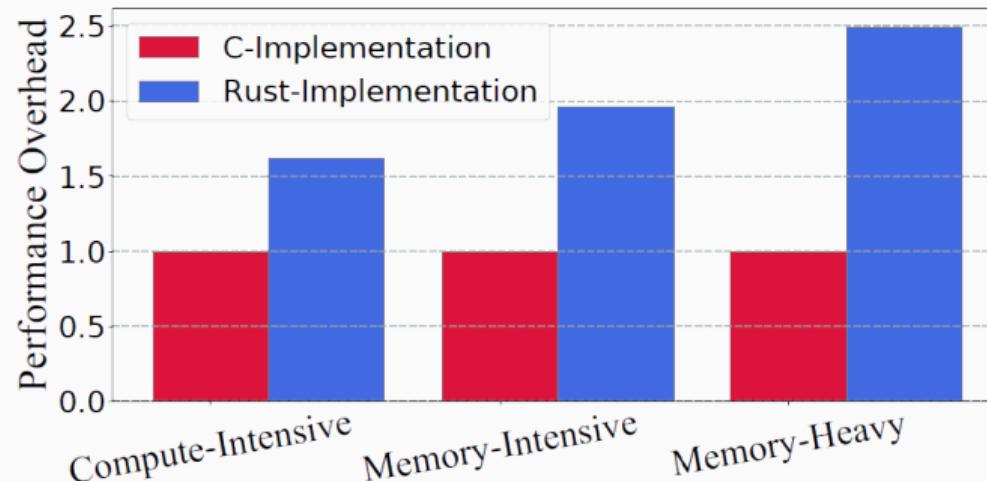
**Zig** (2016) is a minimal open-source programming language that can be intended as replacement of C. Zig supports compile time generics, reflection and evaluation, cross-compiling, and manual memory management. It is made to be fully interoperable with C and also includes a C/C++ compiler.

- **No perfect language.** There are always newer '*shining*' languages
- **Alignment.** Force all developers to switch to the new language
- **Interoperability.** Hundreds of billion lines of existing code. Must interoperate with C and C++ code imposing serious design constraints
- **Ecosystem.** Lack of tools and libraries developed in the last four decades
- **Time and Cost.** Converting a codebase of 10 million lines: 500 developers, 5 years, \$1,400,000,000<sup>1</sup>

---

<sup>1</sup> Bjarne Stroustrup: Delivering Safe C++

- Performance overhead of safe programming languages



- 
- Towards Understanding the Runtime Performance of Rust
  - How much does Rust's bounds checking actually cost?
  - How to avoid bounds checks in Rust (without unsafe!)
  - Is coding in Rust as bad as in C++?



Lukasz Olejnik, Ph.D, LL.M  
@lukOlejnik

...

There are 220bn lines of COBOL code in use today (1.5bn new lines/year). COBOL is the foundation of 43% of all banking systems. Such systems handle \$3 trillion of daily commerce. COBOL handles 95% of all ATM card-swipes, 80% of all in-person credit card transactions.

*Every second spent trying to understand the language is one not spent understanding the problem*

# The Course

---

# The Course

Days 1 - 10

Teach yourself variables, constants, arrays, strings, expressions, statements, functions,....



Days 11 - 21

Teach yourself program flow, pointers, references, classes, objects, inheritance, polymorphism, ....



Days 22 - 697

Do a lot of recreational programming. Have fun hacking but remember to learn from your mistakes.



Days 698 - 3648

Interact with other programmers. Work on programming projects together. Learn from them.



Days 3649 - 7781

Teach yourself advanced theoretical physics and formulate a consistent theory of quantum gravity.



Days 7782 - 14611

Teach yourself biochemistry, molecular biology, genetics,...



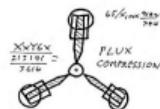
Day 14611

Use knowledge of biology to make an age-reversing potion.



Day 14611

Use knowledge of physics to build flux capacitor and go back in time to day 21.



Day 21

Replace younger self.



As far as I know, this is the easiest way to

"Teach Yourself C++ in 21 Days".

## The Course

Don't forget: The right name of the course should be  
*“Introduction to Modern C++ Programming”*

For many topics in the course, there are more than one book devoted to present the concepts in detail

# The Course

**The primary goal of the course is to drive who has previous experience with C/C++ and object-oriented programming to a proficiency level of (C++) programming**

- *Proficiency*: know what you are doing and the related implications
- Understand what problems/issues address a given language feature
- Learn engineering practices (e.g. code conventions, tools) and hardware/software techniques (e.g. semantic, optimizations) that are not strictly related to C++

What the course **is not**:

- A theoretical course on programming
- A high-level concept description

What the course **is**:

- A practical course, prefer examples to long descriptions
- A “quite” advanced C++ programming language course

# The Course

## Organization:

- 22 lectures
- ~1,500 slides
- C++03 / C++11 / C++14 / C++17 / C++20 / (C++23) / (C++26)

## Roadmap:

- Review C concepts in C++ (built-in types, memory management, preprocessing, etc.)
- Introduce object-oriented and template concepts
- Present how to organize the code and the main conventions
- C++ tool goals and usage (debugger, static analysis, etc.)

# Who I Am

**Federico Busato, Ph.D.**

[federico-busato.github.io](https://federico-busato.github.io)

- Senior Software Engineer at Nvidia,  
CUDA Mathematical Libraries
- Lead engineer of the Sparse Linear Algebra group
- **Research/Work interests:**
  - Linear Algebra
  - Graph Algorithms
  - Parallel/High-Performance Computing
  - Code Optimization



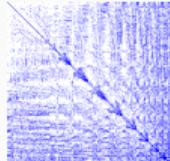
NOT a C++ expert/“guru”, still learning



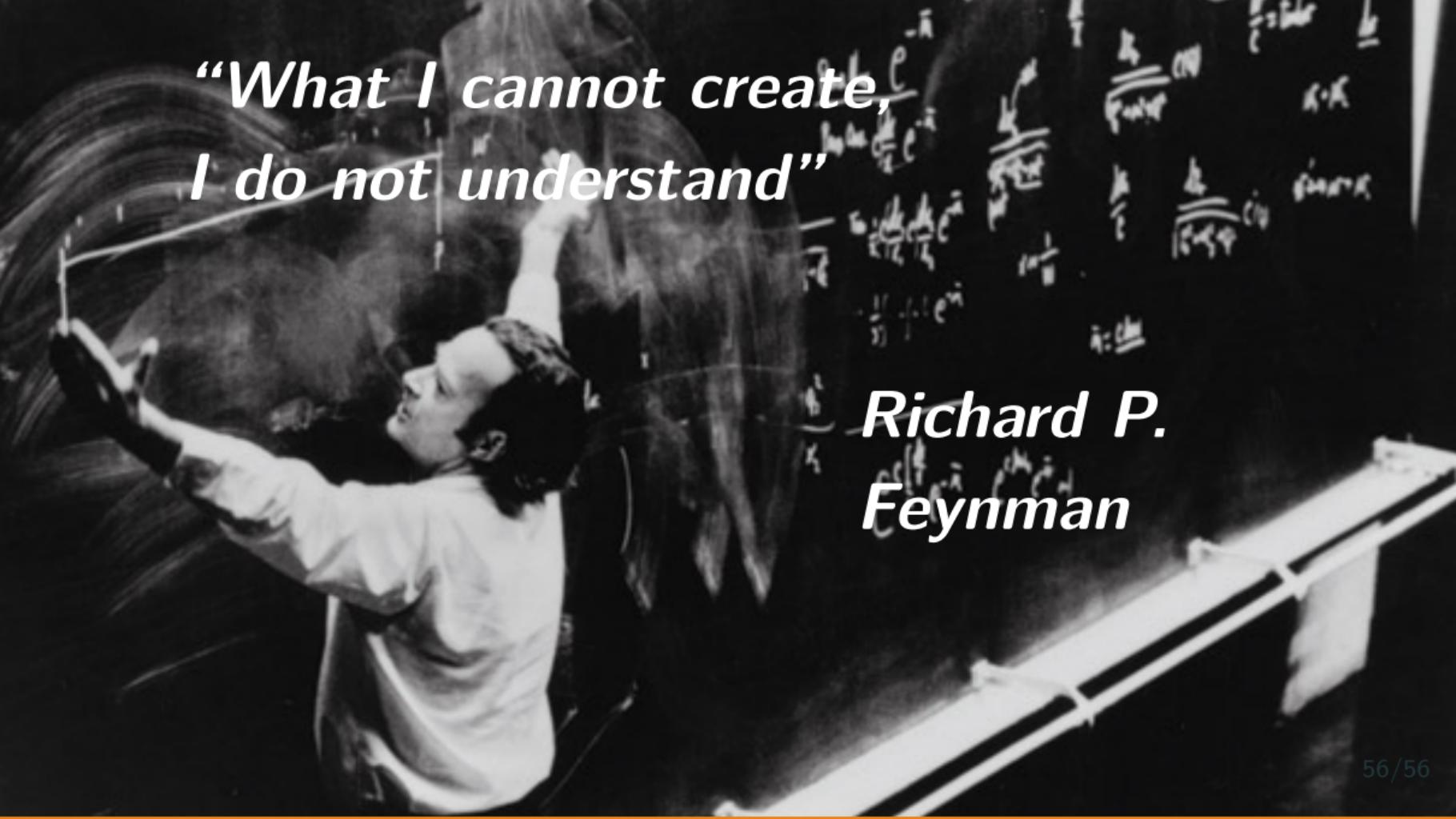
Follow @fedebusato

# A Little Bit about My Work

Our projects:



- cuSPARSE** GPU-accelerated sparse linear algebra library (matrix-matrix multiplication, triangular solver, etc.), part of the **CUDA Toolkit** (8M downloads every year)
- cuSPARSELt** Specialized library for sparse matrix-matrix multiplication that exploits the most advanced GPU features such as Sparse Tensor Cores
- NVPL Sparse** CPU-accelerated (ARM) sparse linear algebra library
- Top500 HPCG** NVIDIA Supercomputing benchmark that performs a fixed number of multigrid preconditioned (using a symmetric Gauss-Seidel smoother) conjugate gradient (PCG) iterations



*“What I cannot create,  
I do not understand”*

*Richard P.  
Feynman*

*“The only way to learn a new programming language is by writing programs in it”*

*Dennis Ritchie*

*Creator of the C programming language*