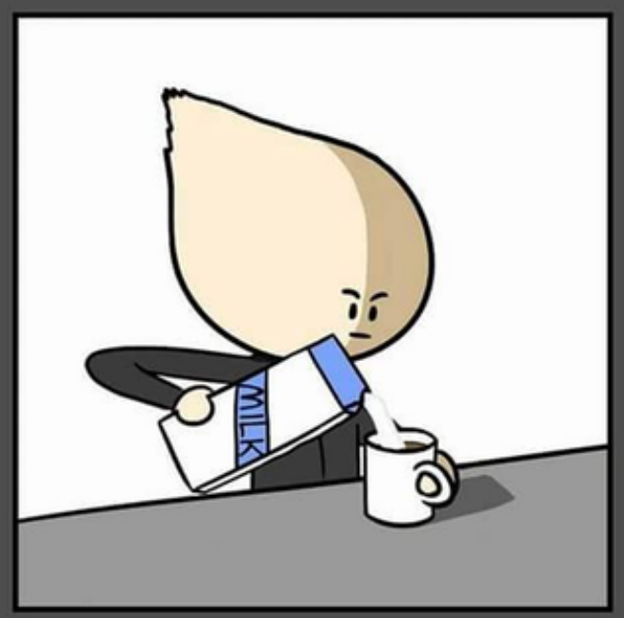
Week 1

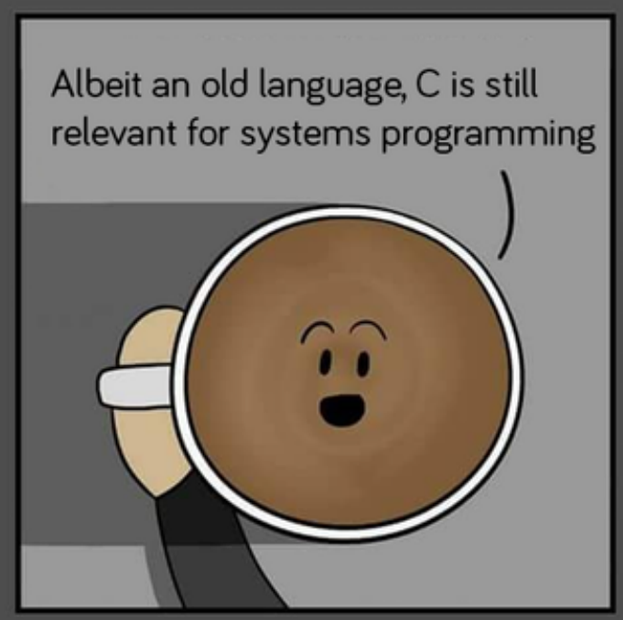
- History lesson
- Basics
- Compiler
- Editor

COFFEE, YOU'RE MY ONLY FRIEND

The C programming language is archaic and useless in 2017

HMM, TOO DARK


MILK


Albeit an old language, C is still relevant for systems programming

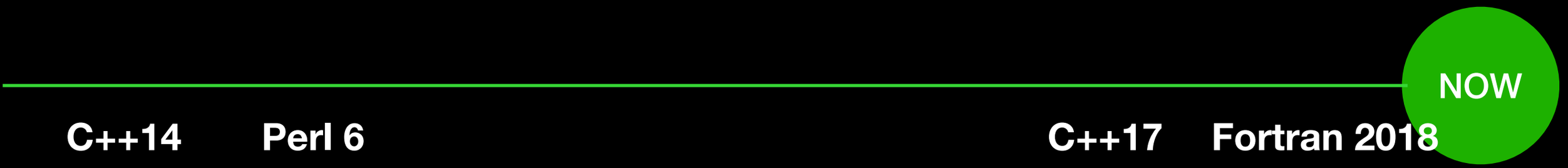
PERFECT

RaphComic

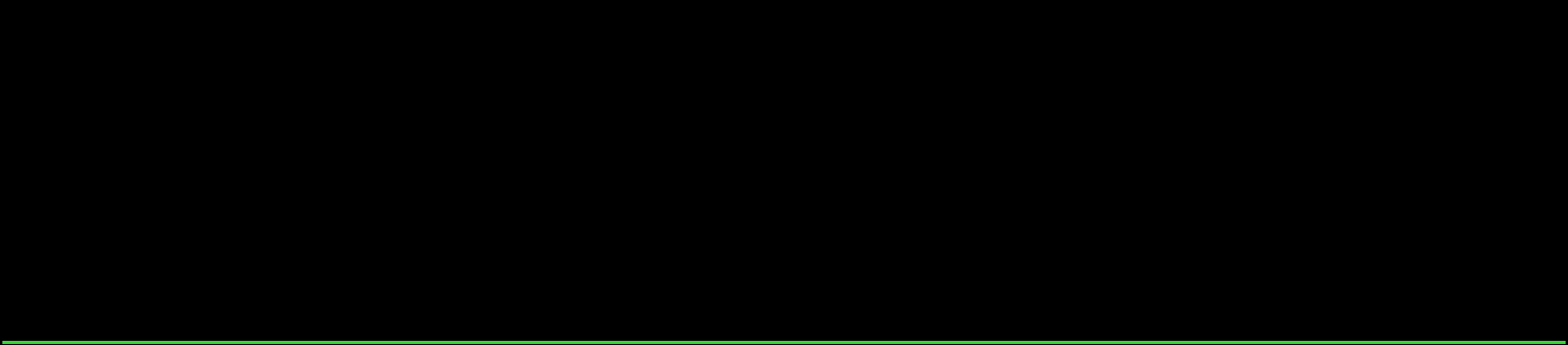
Over the years many programming languages were created

- Many vanished after some years of popularity
- Some remain in very obscure places (e.g. FORTRAN)
- Languages either evolve or go extinct
- What we use today could be dead in some years
- It has happened before

2014 - 2018



2010 - 2014



2000 - 2010

C# D Scratch Scala C++03 Groovy F# Closure Nim Go

1990 - 2000

J Haskell Python Brainfuck Lua Java PHP Ruby Javascript

1980 - 1990

C
with classes

Objective C

C++

Bash

1970 - 1980

C Prolog

Matlab

1960 - 1970

1950 - 1960

Fortran

COBOL

C

- High Level Assembler
- Almost nothing is made automatically
- Much behavior is undefined / compiler dependent behavior
- Errors are subtle and there are no exceptions. Unless your operating system kills the process, it continues running

Why bother?

- Fast
- Really fast
- As fast as you can possibly get without using assembler
- If nothing is done automatically, you define everything that happens -> low memory overhead
- For some use cases only option, e.g. Arduino

What will you learn?

- How to use C for practical problems
- How to debug archaic languages
- Writing fast and minimal code
- Good understanding of stack/heap/memory

Why do I use my free time to teach this, if there is a course that gives actual credits?

- Teach you how to do C in practice, asides lecture
- Show you how to be productive with C
- Explain how to find and fix bugs in C, which is ridiculously difficult
- Clean coding and TDD in practice

What can you do afterwards?

- Program on embedded devices like a pro
- Learn C++
- Tell all your friends that C is the best language ever

Getting started

- Download Visual Studio Code
- Only for Windows users: Download and install a C compiler
- Download Zeal/Dash and the C docset to look up the documentation

Hello World!

- Save file as test.c
- gcc test.c
- ./a.out

```c
#include <stdio.h>

int main(void){
  printf(„Hello World!“);
  return 0;
}
```

Hello World!

- Save file as test.c
- gcc test.c
- ./a.out

```c
#include <stdio.h>

int main(void){
   printf(„Hello World!“);
   printf(„Hello World!“);
   return 0;
}
```

- Output looks weird
- Hint: \n indicates a new line

Printf

- Printf is often used for formatted output
- E.g. we want to print the content of an int

- printf(„%i“, someInt);
  printf(„%c“, someChar);
  printf(„%s“, someString);
  printf(„%i + %i“, int1, int2);

Compiler dependent behavior

- Save file as test.c
- gcc test.c
- ./a.out
- What is your result?

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

int main(void){
   int* x;
   x = malloc(sizeof(int));
   printf("%i\n",*x );
   return 0;
}
```

Compiler dependent?

> **C11, § 7.22.3.4 The `malloc` function**
>
> The `malloc` function allocates space for an object whose size is specified by size and whose **value is indeterminate**.

- C is a specification agreed upon by a committee
- They decide for example what should happen if you malloc
- Compiler manufacturers still need to declare some behavior
- Therefore different compiler can yield different results