# The C Programming language

*(based on earlier slides by Dag Langmyhr and Michael Welzl)*

- C is the programming language for IN2140
  - This course introduces operating systems and networks.
- Why can't we use Python?
  - Many things that an operating system must do cannot be done in Python at all.
  - Operating systems require attention to detail. Python is excellent for getting results quickly without bothering with details.
- Why can't we use Rust?
  - It is possible to perform critical operations close to the hardware.
  - Learning how to bypass Rust's safety to do it takes a lot of time.

# The C Programming language

- Invented by *Dennis Ritchie*
- for the implementation of the Unix operating system at AT&T's Palo Alto laboratory 1960-75

- Well-known variants:
  - K&R C (Kernighan and Ritchie's C, the original)
  - ANSI C (also called C89)
  - C99
  - C11
  - GNU11 ← the one we use in IN2140

# What is C good for?

- Write large software packages

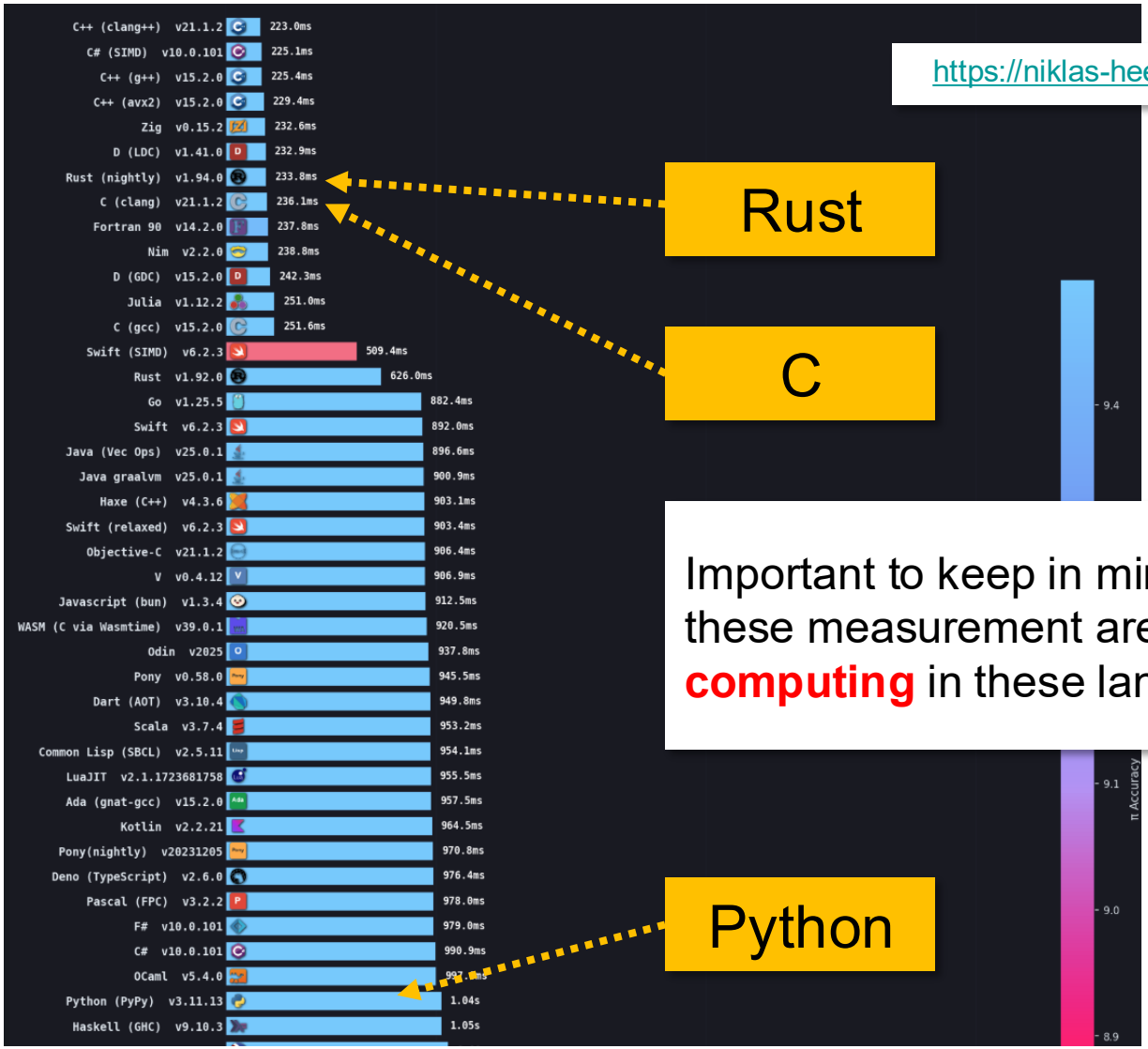
# What is C good for?

- ~~Write large software packages~~

- Write programs with a **low memory footprint**
- Write programs with **low computation time**

- Accessing and controlling the hardware resources of a computer
- Use the architecture-dependent features of a computer
- Learn to **understand** how **memory** is used, manipulated and managed

# Programs that **run** fast




https://niklas-heer.github.io/speed-comparison/

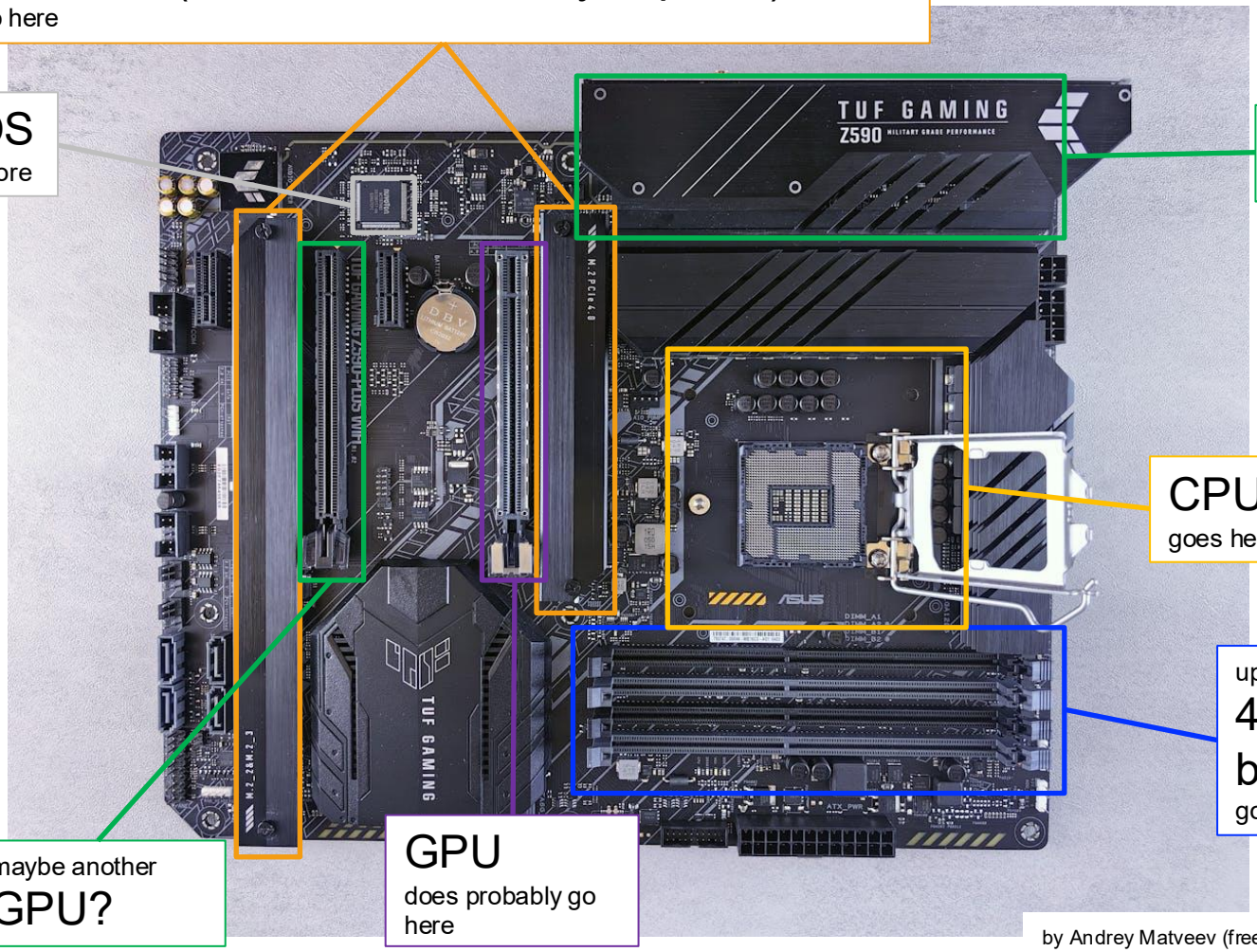
Rust

C

Python

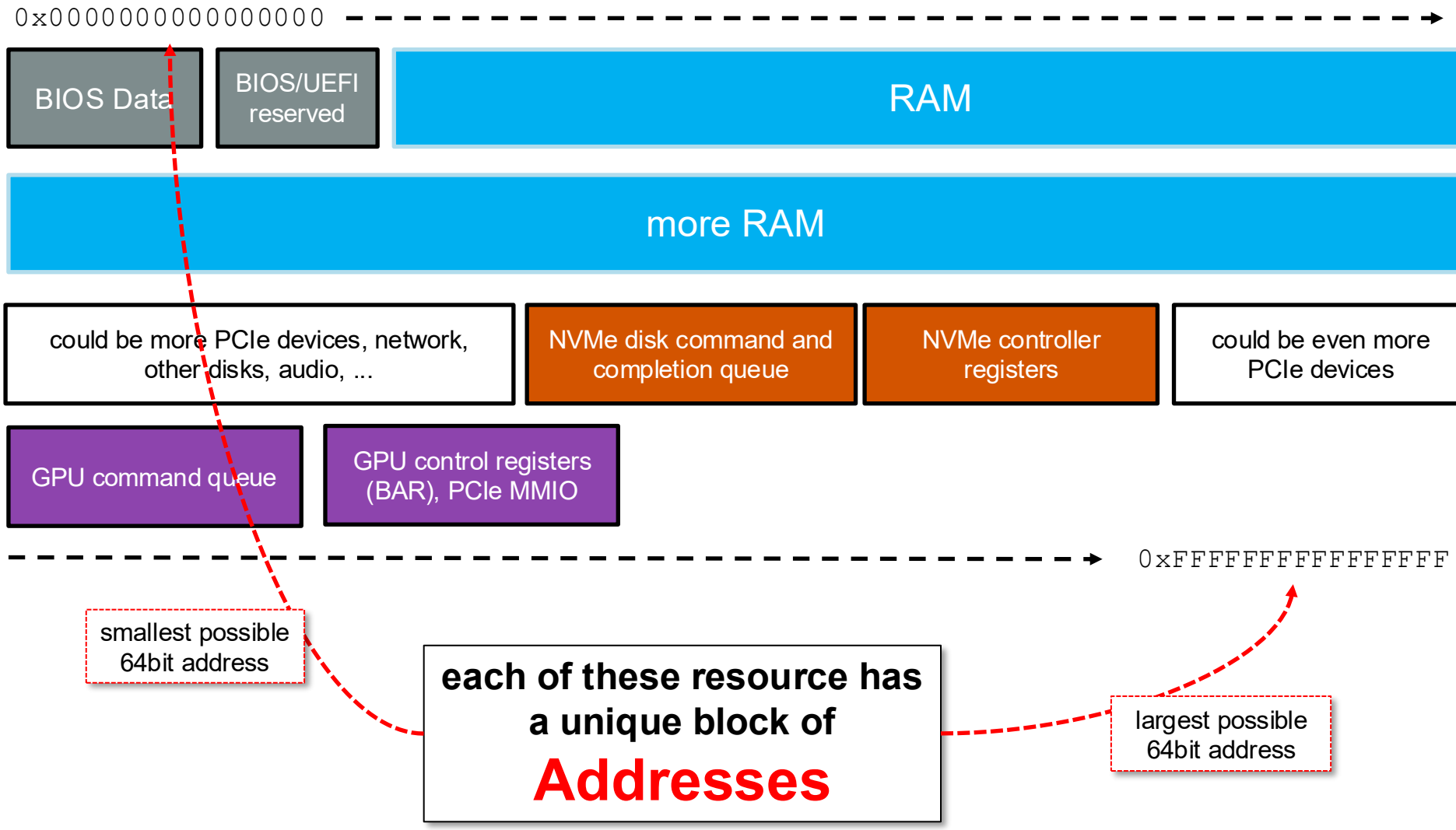
Important to keep in mind:
these measurement are from heavy **computing** in these languages

# Locating a few hardware pieces

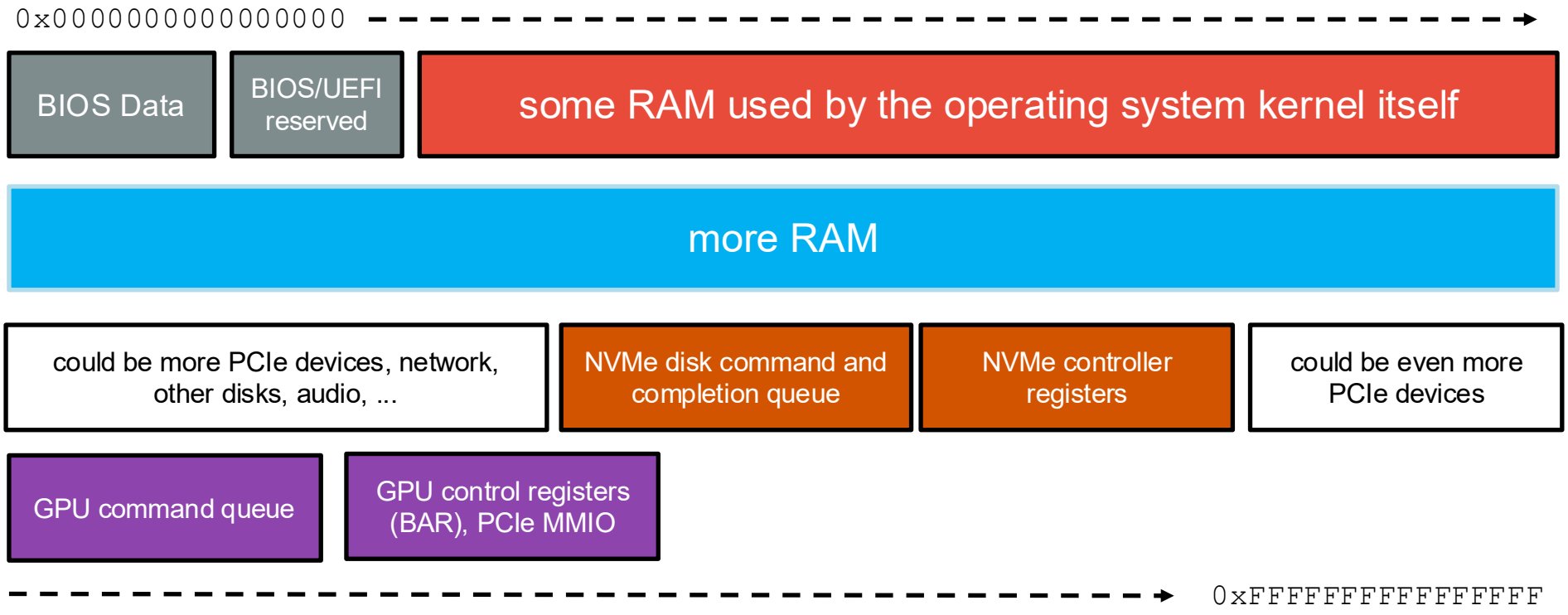



by Andrey Matveev (free to use)

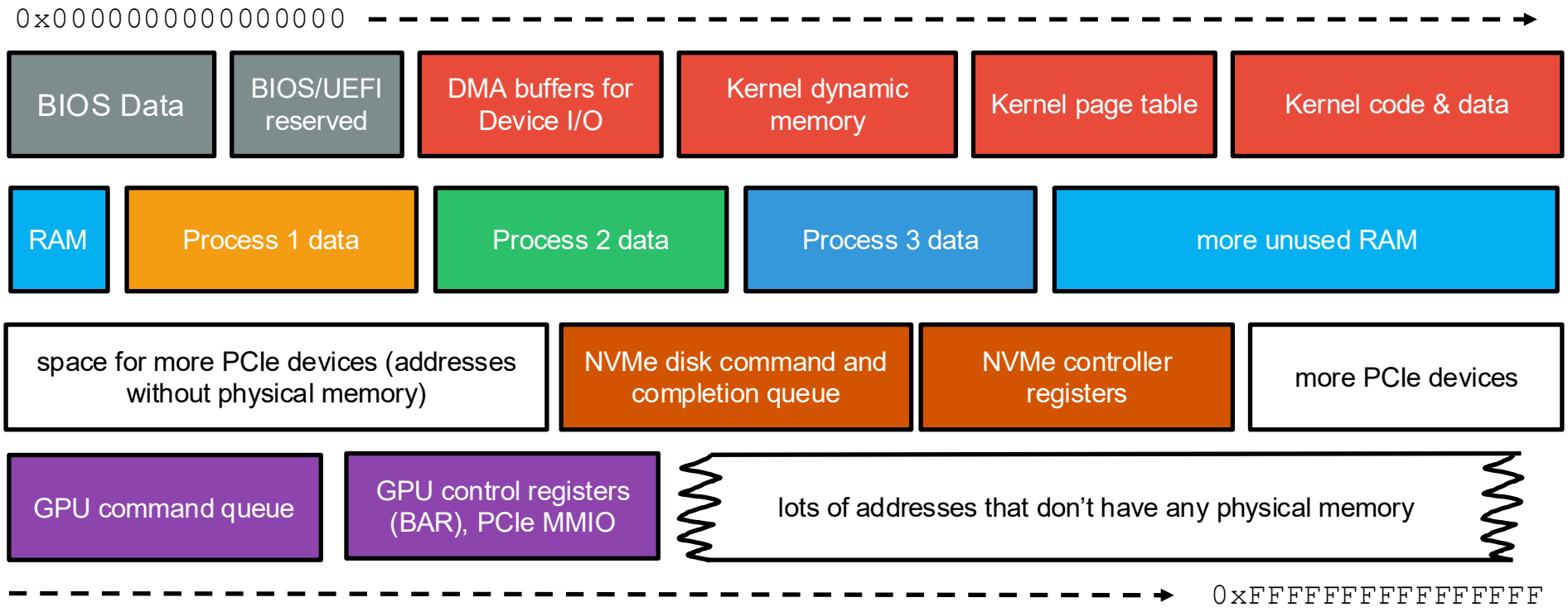
# The CPU must control all of this



# The CPU must control all of this

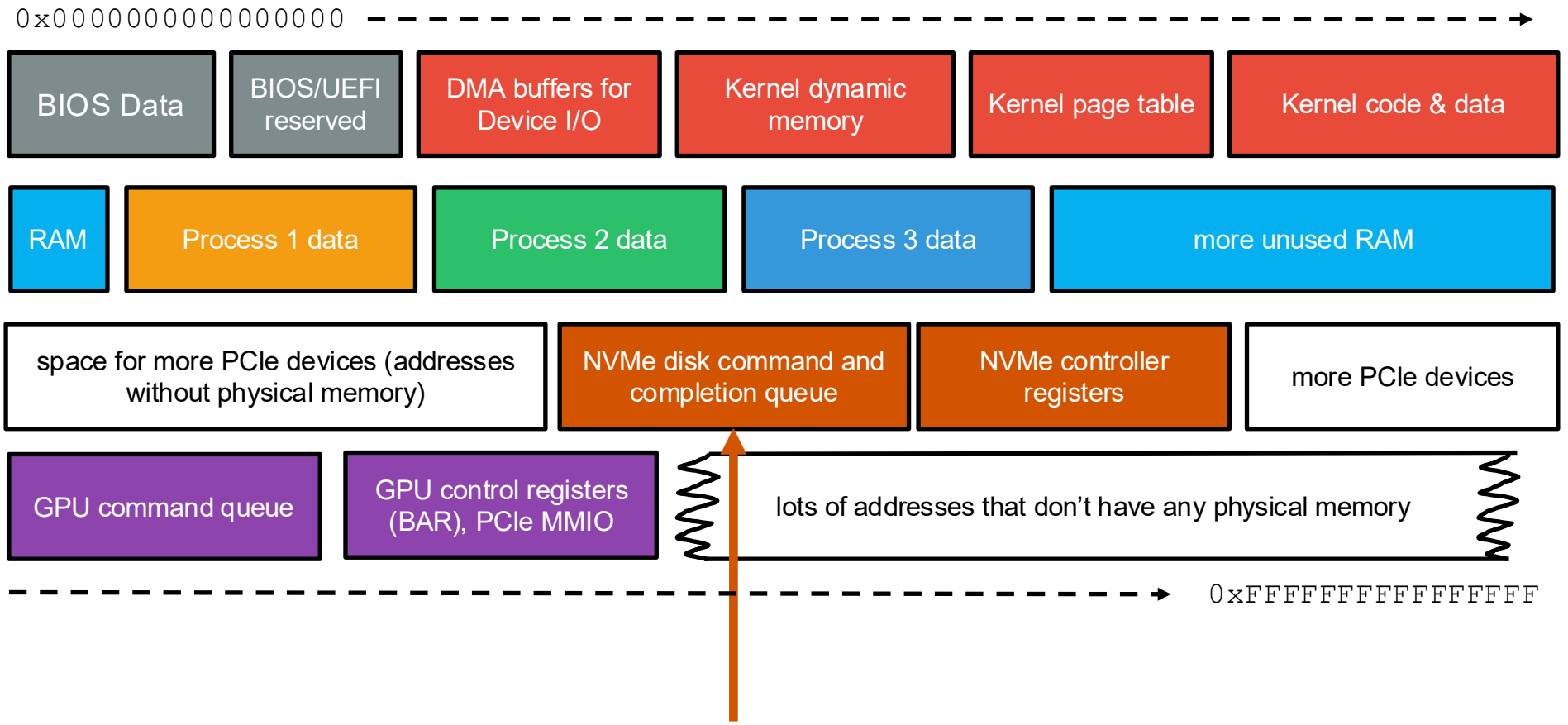


# The real power of C: memory manipulation



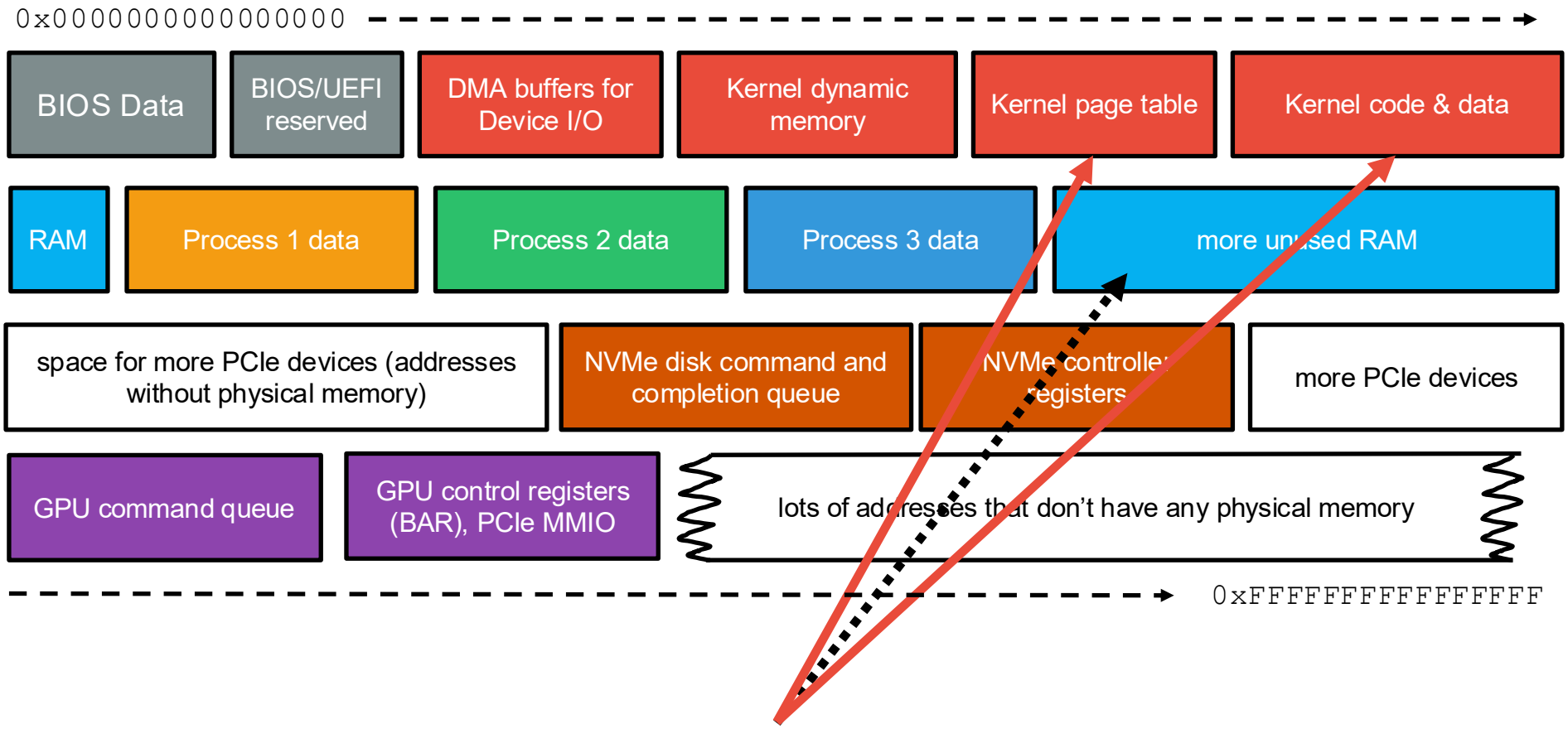

Operating systems
must be capable of
making changes everywhere

# The real power of C: memory manipulation



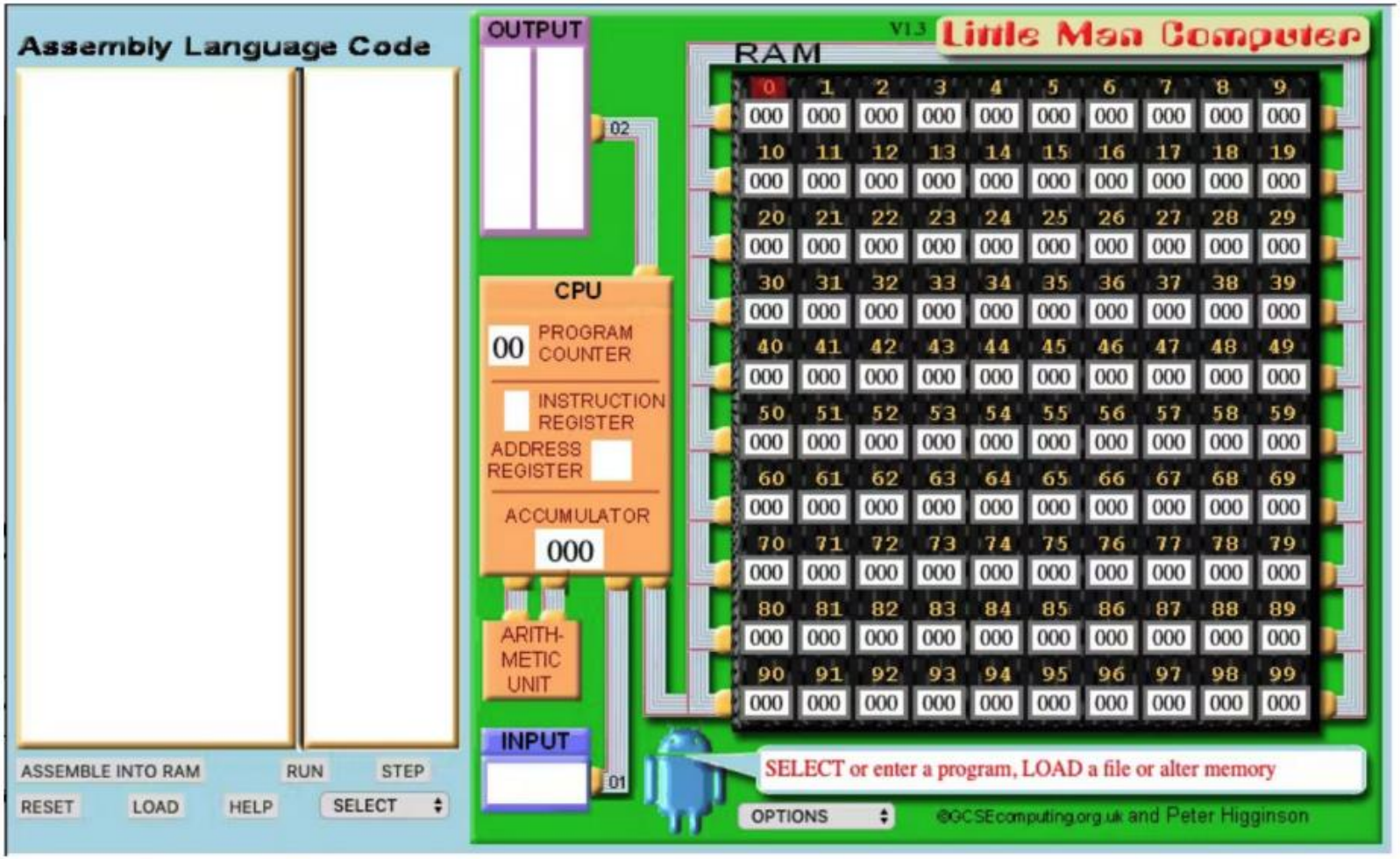

send commands to a disk, listen to responses

# The real power of C: memory manipulation




start a new process
(think of a process as an instance of a program)

# The real power of C: memory manipulation



# C – pros and cons

- Advantages
  - Compiles to really fast code
  - Well-standardized

- Disadvantages
  - No built-in error handling
  - Code is not inherently portable

- Both
  - Freedom ... do what you want but don’t expect help

# Hello World

All teaching books start with an example like this:

```c
#include <stdio.h>

int main(void)
{
    printf("Hello 2140!\n");
    return 0;
}
```

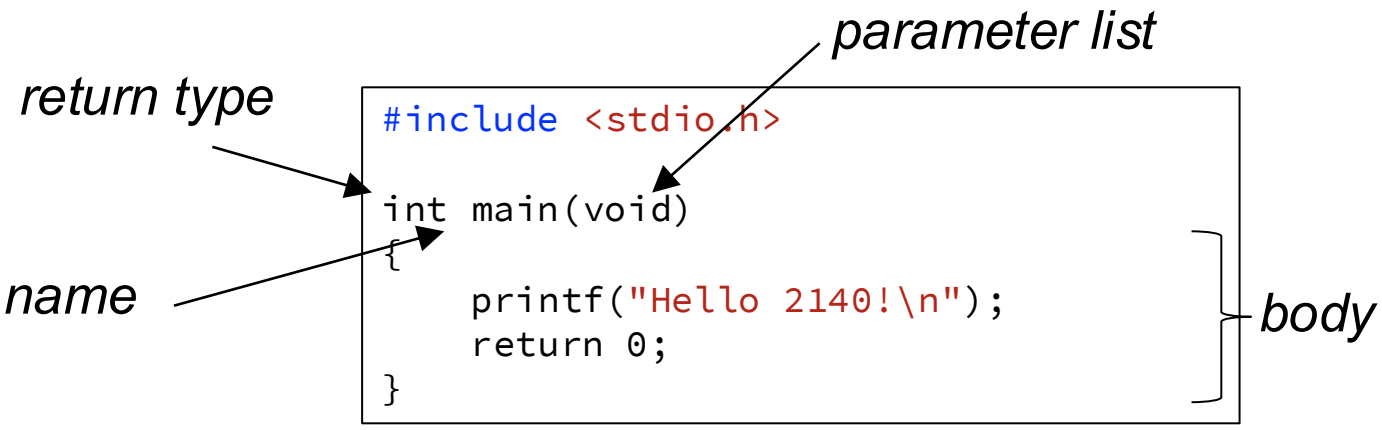
(Actually, Kernighan & Ritchie's first book started this fashion)

Getting to the IFI Linux machine login.ifi.uio.no:

```
ssh -Y -J griff@login.uio.no griff@login.ifi.uio.no
```

# What does it all mean?

C functions are similar to Python functions, but their specifications include more details.



# What does it all mean?

`#include` – same job as `import` but **much more primitive**
It does *literally* mean copy the content of `stdio.h` into this file before translating it to machine language.

```c
#include <stdio.h>

int main(void)
{
    printf("Hello 2140!\n");
    return 0;
}
```

Files are included because they can contain, among other things, the declarations of functions the libraries provide to us.

Like here: `printf` – one of the standard output functions.

# What does it all mean?

`printf` - a function that prints text into your terminal.

It can do a lot more than just printing a string. But first things first.

```c
#include <stdio.h>

int main(void)
{
    printf("Hello 2140!\n");
    return 0;
}
```

`"Hello 2140!\n"`

This is a **C-string**. Much more primitive than a Python string.
How many bytes of space does it take up?

# What does it all mean?

`printf` - a function that prints text into your terminal.

It can do a lot more than just printing a string. But first things first.

```c
#include <stdio.h>

int main(void)
{
    printf("Hello 2140!\n");
    return 0;
}
```

**Important!**
**We will get back to this**

`"Hello 2140!\n"`

This is a **C-string**. Much more primitive than a Python string.

How many bytes of space does it take up? → who guessed **13**?

One for every character in `Hello 2140!` -> 11

One for the newline character `\n` -> 1

One for a hidden **0**-byte as end-of-string marker -> 1

# What does it all mean?

`main`
a standalone C program starts with the function that is named `main`

```c
#include <stdio.h>

int main(void)
{
    printf("Hello 2140!\n");
    return 0;
}
```

`int` ◂---------- `return 0;`
It says that the program returns an integer (`int`) value when it ends. Terminals expect 0 for no error and negative values for errors. Positive values are sometimes computation results.

# Compilation

To translate the program into runnable machine instructions, use:

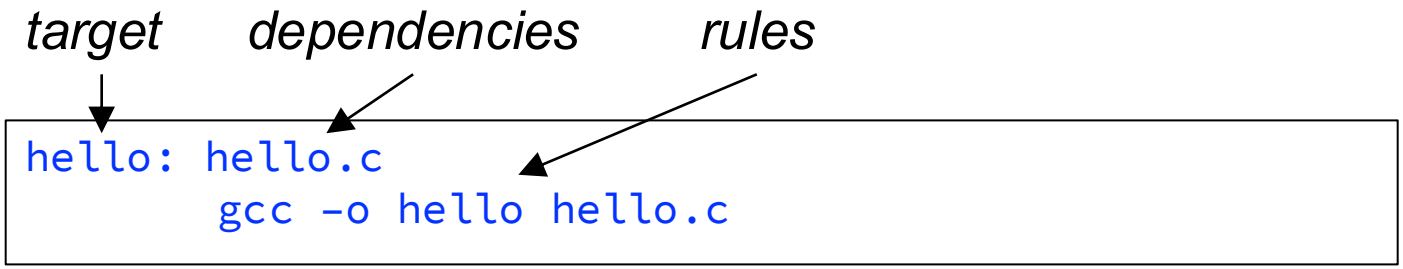
```
$ gcc -o hello hello.c
```

This can now be executed without an interpreter (without the huge hidden Python machinery)

```
$ ./hello
```

# Makefile

It can be tedious to type this every time you made a change. `make` helps. Write a file called **Makefile**.

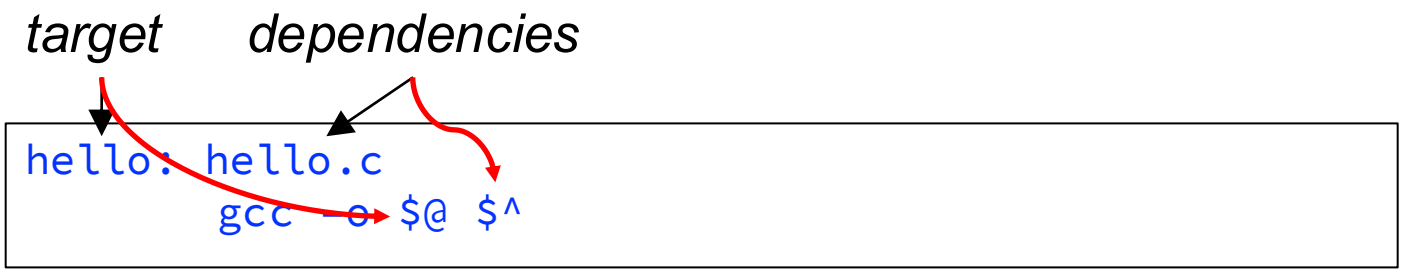



```
$ make
```

If any dependency has changed since the creation of target, *make* a new target using the rules.

# Makefile

If you want to type less (but remember more):




```
$ make
```

# Functions

## Functions must be declared before use:

```c
#include <stdio.h>

void printme()
{
    printf("Hello 2140!\n");
}

int main(void)
{
    printme();
    return 0;
}
```

# Functions

## Not possible:

```
#include <stdio.h>

int main(void)
{
    printme();
    return 0;
}

void printme()
{
    printf("Hello 2140!\n");
}
```

## Possible:

```
#include <stdio.h>

void printme();

int main(void)
{
    printme();
    return 0;
}

void printme()
{
    printf("Hello 2140!\n");
}
```

# Functions

## Modular programming is very important

hello.c

```
#include "out.h"

int main(void)
{
    printme();
    return 0;
}
```

out.c

```
#include <stdio.h>

void printme()
{
    printf("Hello 2140!\n");
}
```

Makefile

```
hello: hello.c out.c
	gcc -o $@ $^

hello: out.h
```

out.h

```
#ifndef OUT_H
#define OUT_H

void printme();

#endif
```