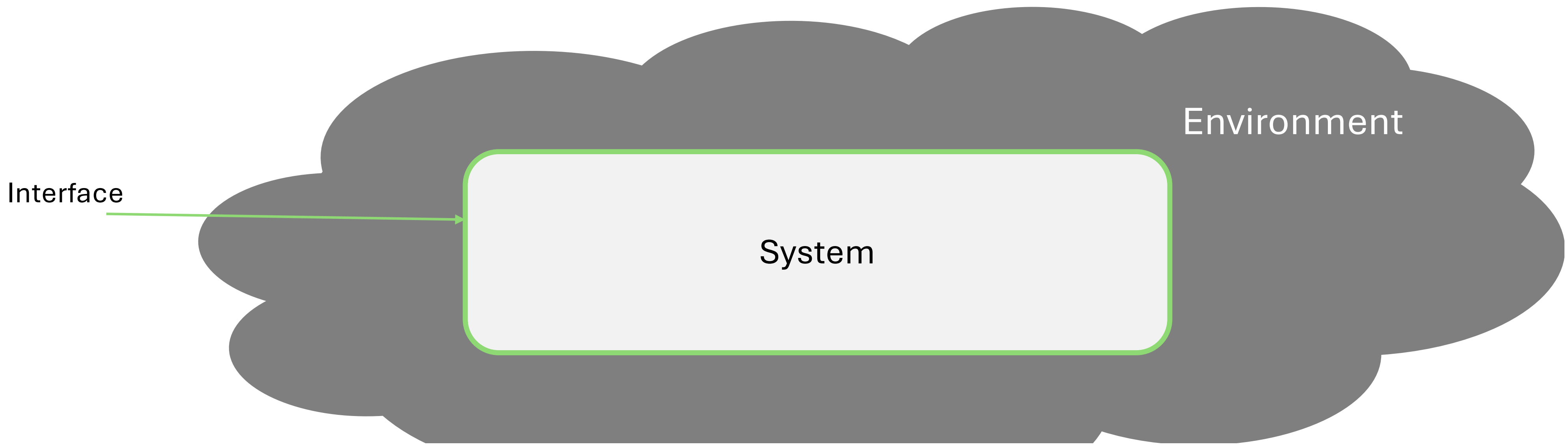
Programming & Systems

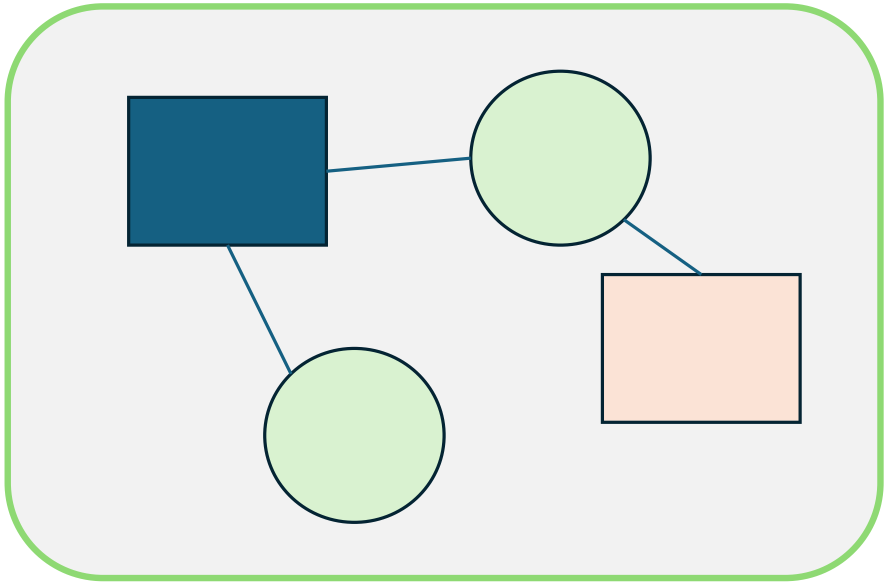
Philippe Bonnet, bonnet@di.ku.dk
HPPS 2025 – 1a

Systems

A system is a set of interconnected components
with a well-defined behavior
at the interface with its environment



Interconnected components



- Modularity
  - Each component is a subsystem
    - We can think about interactions within a module independently of other modules
- Abstraction
  - Exposes external specification
  - Hides complexity of internal implementation
- Layering and Hierarchy
  - Fewer interactions among modules
  - Less propagation of effects

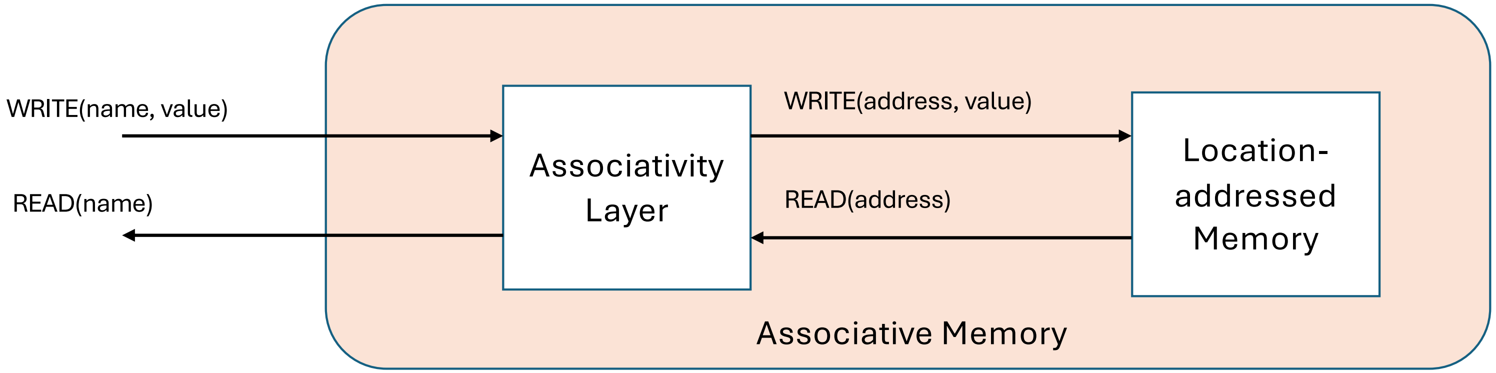
Computer Systems

3 fundamental abstractions for computer systems:

1. Interpreter
2. Memory
3. Communication

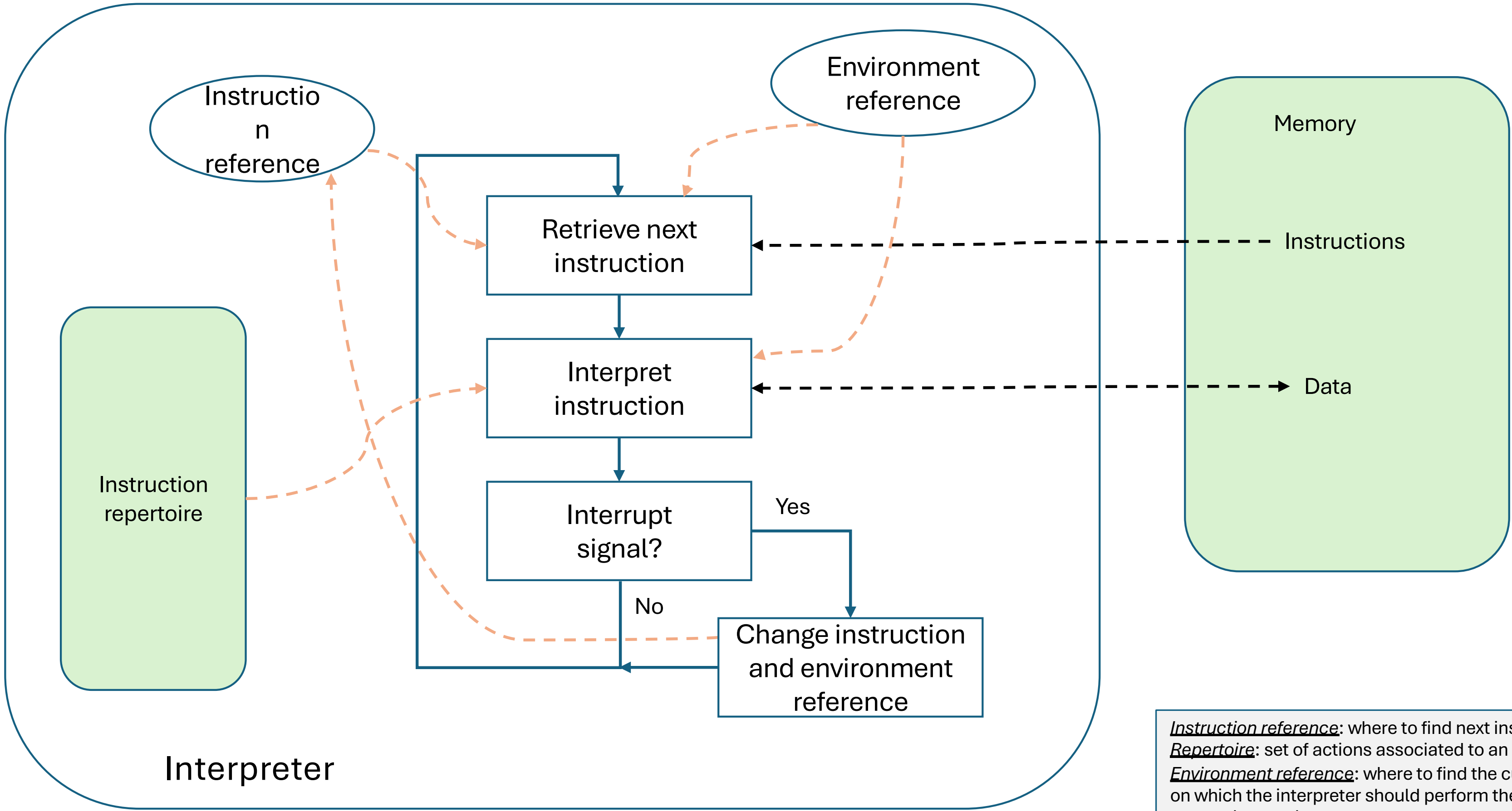
Source: Saltzer and Kaashoek

Memory Abstraction



Source: Saltzer and Kaashoek

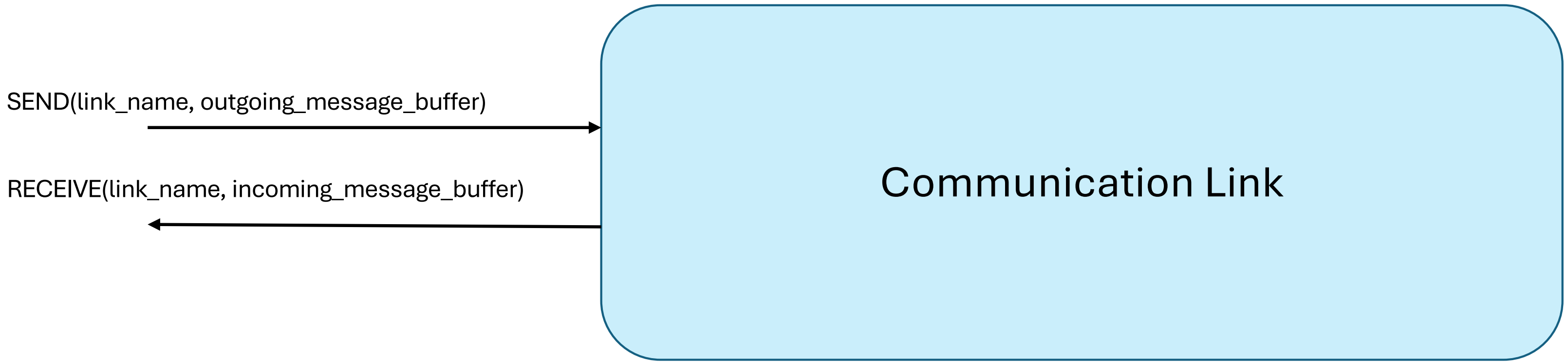
Interpreter Abstraction



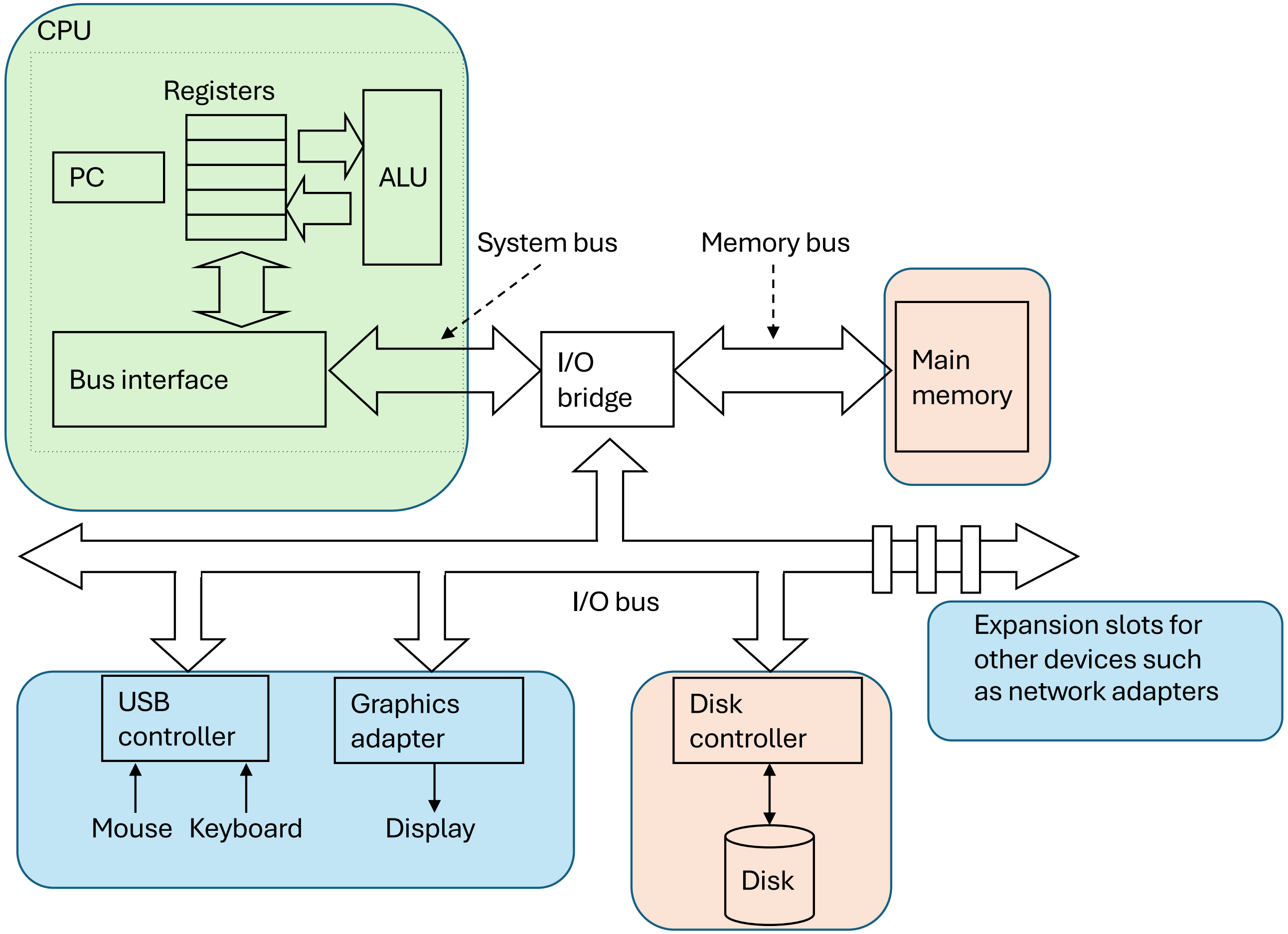
Instruction reference: where to find next instruction
Repertoire: set of actions associated to an instruction
Environment reference: where to find the current state on which the interpreter should perform the actions of the current instruction

Source: Saltzer and Kaashoek

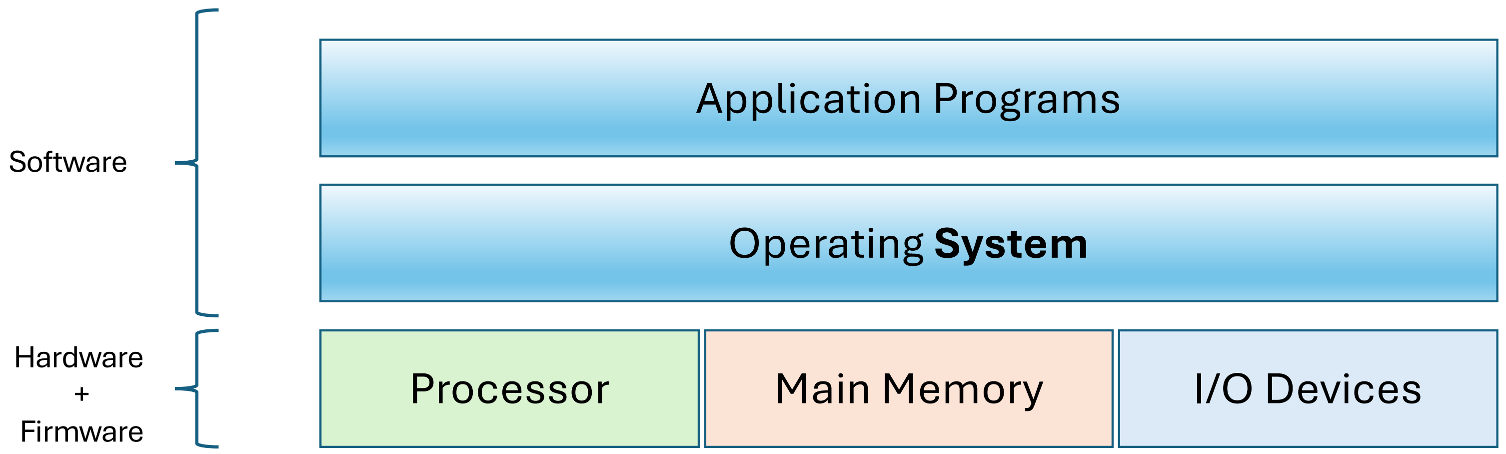
Communication Abstraction



Computer Hardware



Layered view of a Computer System



Programming

Functions

A function maps inputs into outputs.

A function computes a value and stops.

Key properties:
- termination
- correctness (given a specification)

Sequences of states

Programs that run forever (e.g., operating system, http server)

A program execution is represented by a a sequence of states (i.e., assignment of values to variables).

Key properties:
-**safety**: nothing wrong will happen
-**liveness**: something good will happen

Programming

Functions

A function maps inputs into outputs.

A function computes a value and stops.

Key properties:
- Termination (= safety)
- correctness (= liveness)

Sequences of states

Programs that run forever (e.g., operating system)

A program execution is represented by a a sequence of states (i.e., assignment of values to variables).

Key properties:
-**safety**: nothing wrong will happen
-**liveness**: something good will happen

[Lamport talk on "Thinking above the Code"]

System programming

Check out **Tiger Style**
https://tigerstyle.dev

- Direct hardware control
  - Processor, memory, I/O devices
- Focus on performance and resource utilization
  - Chasing inefficiencies
  - Explicit resource allocation and management (e.g., memory allocation)
  - Leveraging hardware characteristics
  - Abstraction vs. performance trade-off
- Focus on safety
  - Simple and explicit control flow
  - Explicit upper bounds on queues, loops to control worst cases

https://github.com/tigerbeetle/tigerbeetle/blob/ac75926f8868093b342ce2c64eac1e3001cf2301/docs/TIGER_STYLE.md

C

"C is quirky, flawed, and an enormous success. While accidents of history surely helped, it evidently satisfied a need for a **system implementation language** efficient enough to displace assembly language, yet sufficiently abstract and fluent to describe algorithms and interactions in a wide variety of environments. "

http://csapp.cs.cmu.edu/3e/docs/chistory.html

The evolution of C

Genealogy

Algol 60 (1960) -> BCPL (1967) -> B (1970) -> C (1972)

Unix operating system & ecosystem
(kernel, editors, compilers, build tools)

C standards

K&R (1978) –> ANSI C (1990) -> c99 (1999) -> c11 (2011) -> c2x/c23 (2024)

Spirit of C

(a) *Trust the programmer.*

(b) *Don't prevent the programmer from doing what needs to be done.*

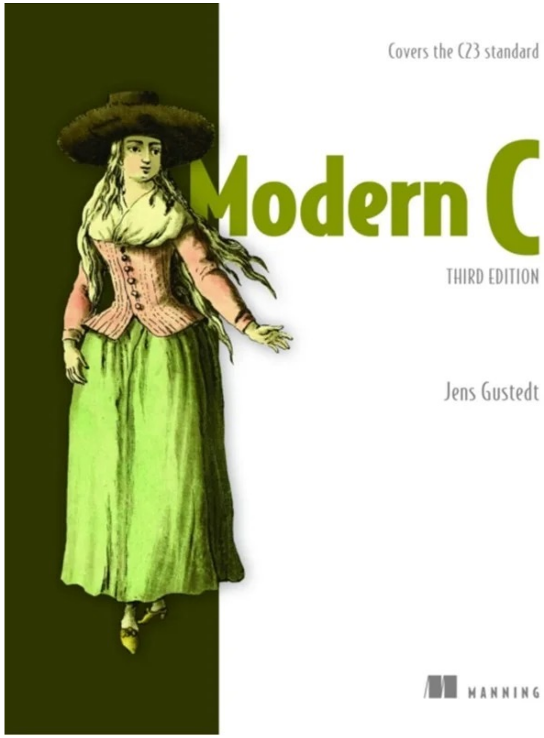
(c) *Keep the language small and simple.*

(d) *Provide only one way to do an operation.*

(e) *Make it fast, even if it is not guaranteed to be portable.*

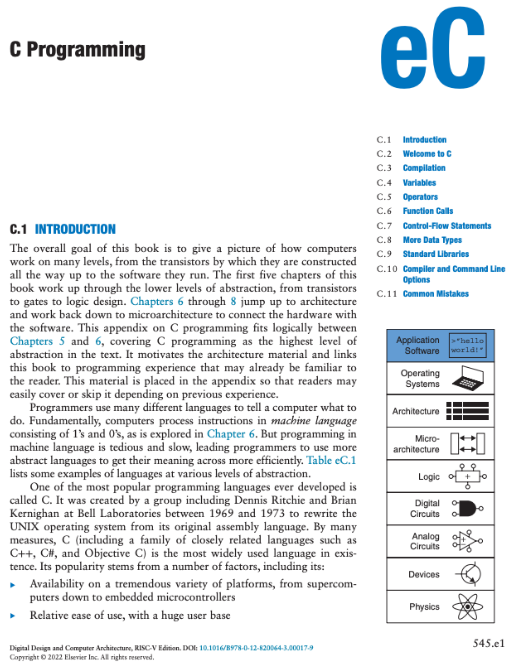
(f) *Make support for safety and security demonstrable*

C books



[ModernC @ INRIA](#)

[ModernC @ Manning (download source code)](#)



C Programming

eC

C.1 INTRODUCTION

The overall goal of this book is to give a picture of how computers work on many levels, from the transistors by which they are constructed all the way up to the software they run. The first five chapters of this book work up through the lower levels of abstraction, from transistors to gates to logic design. Chapters 6 through 8 jump up to architecture and work back down to microarchitecture to connect the hardware with the software. This appendix on C programming fits logically between Chapters 5 and 6, covering C programming as the highest level of abstraction in the text. It motivates the architecture material and links this book to programming experience that may already be familiar to the reader. This material is placed in the appendix so that readers may easily cover or skip it depending on previous experience.

Programmers use many different languages to tell a computer what to do. Fundamentally, computers process instructions in *machine language* consisting of 1's and 0's, as is explored in Chapter 6. But programming in machine language is tedious and slow, leading programmers to use more abstract languages to get their meaning across more efficiently. Table eC.1 lists some examples of languages at various levels of abstraction.

One of the most popular programming languages ever developed is called C. It was created by a group including Dennis Ritchie and Brian Kernighan at Bell Laboratories between 1969 and 1973 to rewrite the UNIX operating system from its original assembly language. By many measures, C (including a family of closely related languages such as C++, C#, and Objective C) is the most widely used language in existence. Its popularity stems from a number of factors, including its:

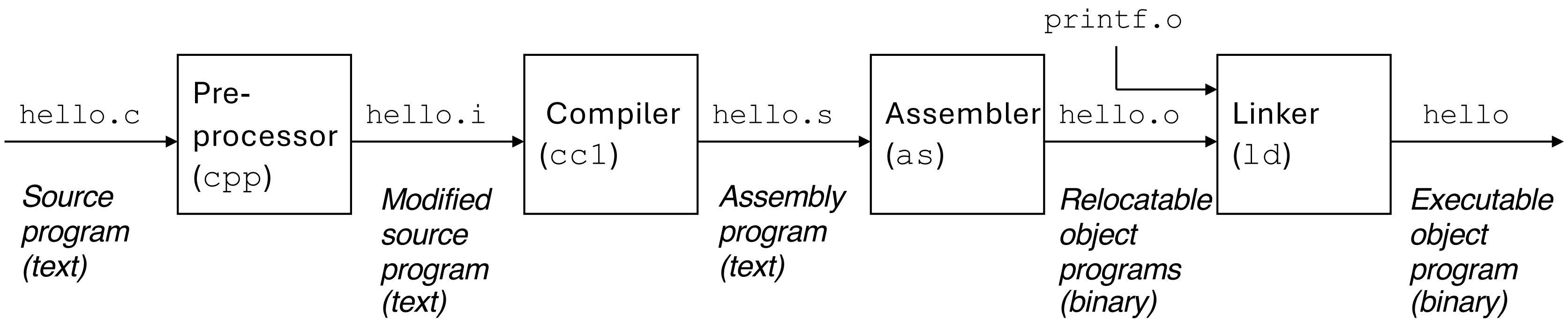
- Availability on a tremendous variety of platforms, from supercomputers down to embedded microcontrollers
- Relative ease of use, with a huge user base

Digital Design and Computer Architecture, RISC-V Edition. DOI: 10.1016/B978-0-12-820064-3.00017-9

[Harris & Harris, Appendix C](#)

https://github.com/gcc-mirror/gcc

C compilation phases



$ gcc -save-temps hello.c –o hello

$ gcc –std=c11 – pedantic –Wall –Wextra –Werror hello.c –o hello

C characteristics

- C is an imperative programming language.
- C is a permissive statically typed language.
- Standard library contains essential functions
    - Print to console
    - Input and output
    - Memory allocation

Zen of zig

Zig (ziglang.org):
No hidden control flow.
No hidden memory allocations.
No preprocessor, no macros.
Compile-time code execution and lazy evaluation.

```
$ zig version
0.15.2
```

- Communicate intent precisely.
- Edge cases matter.
- Favor reading code over writing code.
- Only one obvious way to do things.
- Runtime crashes are better than bugs.
- Compile errors are better than runtime crashes.
- Incremental improvements.
- Avoid local maximums.
- Reduce the amount one must remember.
- Focus on code rather than style.
- Resource allocation may fail; resource deallocation must succeed.
- Memory is a resource.

Next weeks

Focus on the core of everything in systems programming:

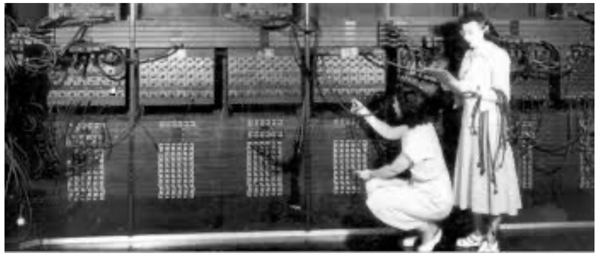
Digital representation

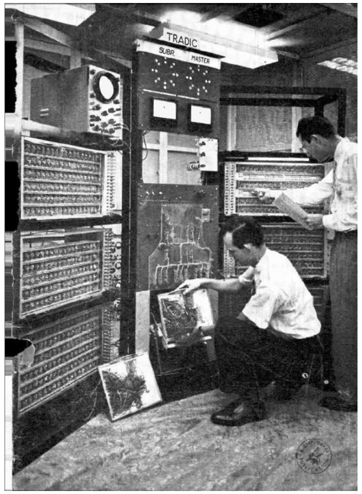
- Digital representation in memory …
  - Representation of data: integer, float, arrays
  - Representation of programs
- .. and on disk
  - Text and binary files

Computers are digital

Eniac – 1st digital computer (1945)

0 and 1 encoded through vacuum tubes





Tradic (1954)

0 and 1 encoded through transistors

- A bit (b) is 0 or 1
- **A byte (B) is 8 bits**
- A kilobyte (KB) is 10^3 B
- A megabyte (MB) is 10^6 B
- A gigabyte (GB) is 10^9 B
- A terrabyte (TB) is 10^{12} B
- A petabyte (PB) is 10^{15} B
- KB, MB, … is different than KiB, MiB, …
  - KiB = 2^{10} bytes; MiB = 2^{20} bytes, …

Binary and hexadecimal

0000	1000	0	8	0	8
0001	1001	1	9	1	9
0010	1010	2	10	2	A
0011	1011	3	11	3	B
0100	1100	4	12	4	C
0101	1101	5	13	5	D
0110	1110	6	14	6	E
0111	1111	7	15	7	F
Binary		Decimal		Hexadecimal	

Binary and hexadecimal

0b00000000	0x00
0b00000001	0x01
…	..
0b00011100	0x1C
…	..
ob11111111	0xFF

2 Hexadecimal numbers => 8 bits

Further Reading