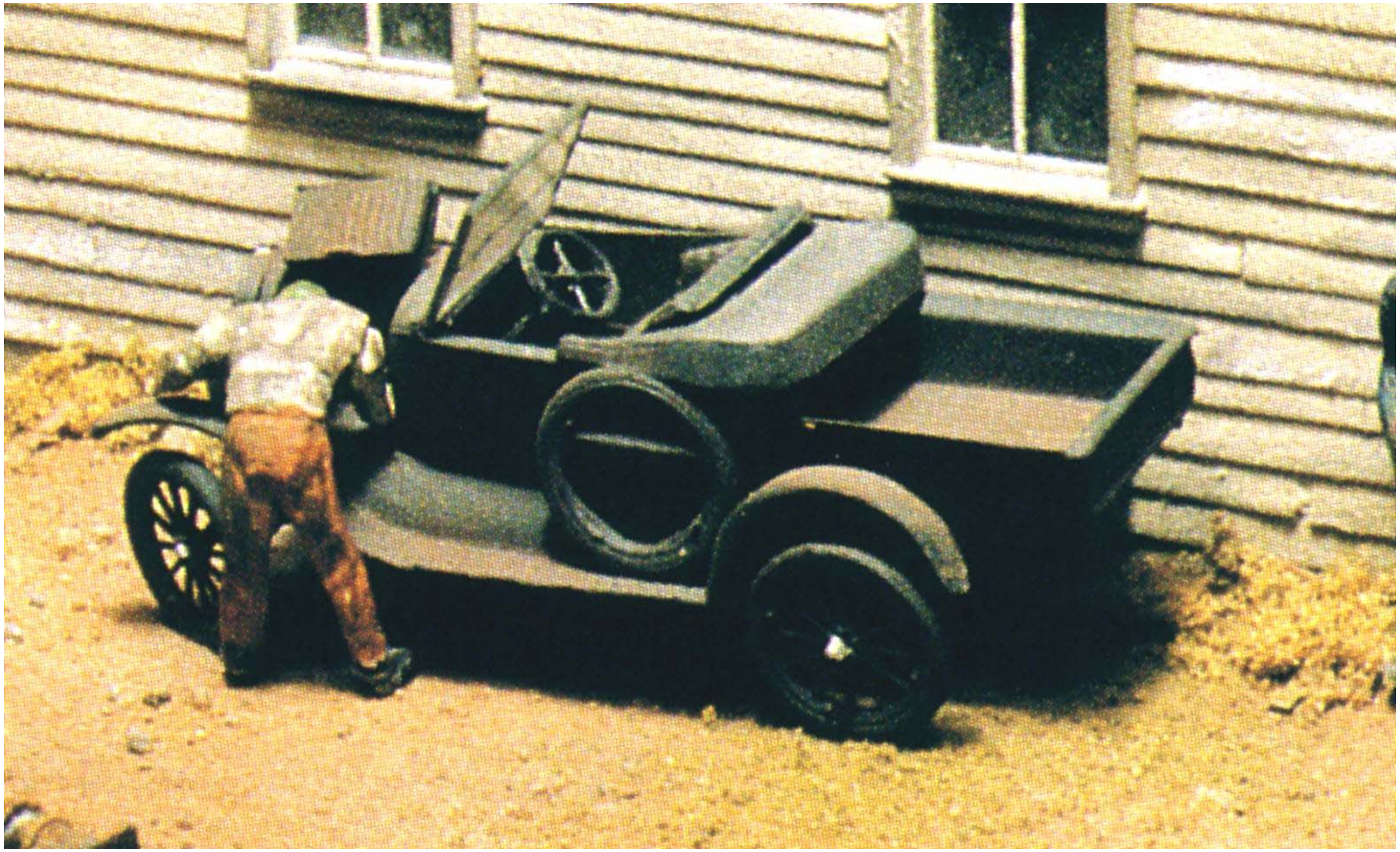
Computer Engineering 1

CT Team: A. Gieriet, J. Gruber, R. Gübeli, M. Meli, M. Rosenthal,
A. Rüst, J. Scheier, M. Thaler

Motivation

- **See what's inside**



Todays Agenda

- What is Computer Engineering?
- Course Content and Organization
- Computer History
- Properties of a Computer System
- von Neumann Architecture
- Hardware Components
  - CPU, Memory, Input/Output, System Bus
- Software Aspects
  - from C to executable
- Interaction of Hardware and Software

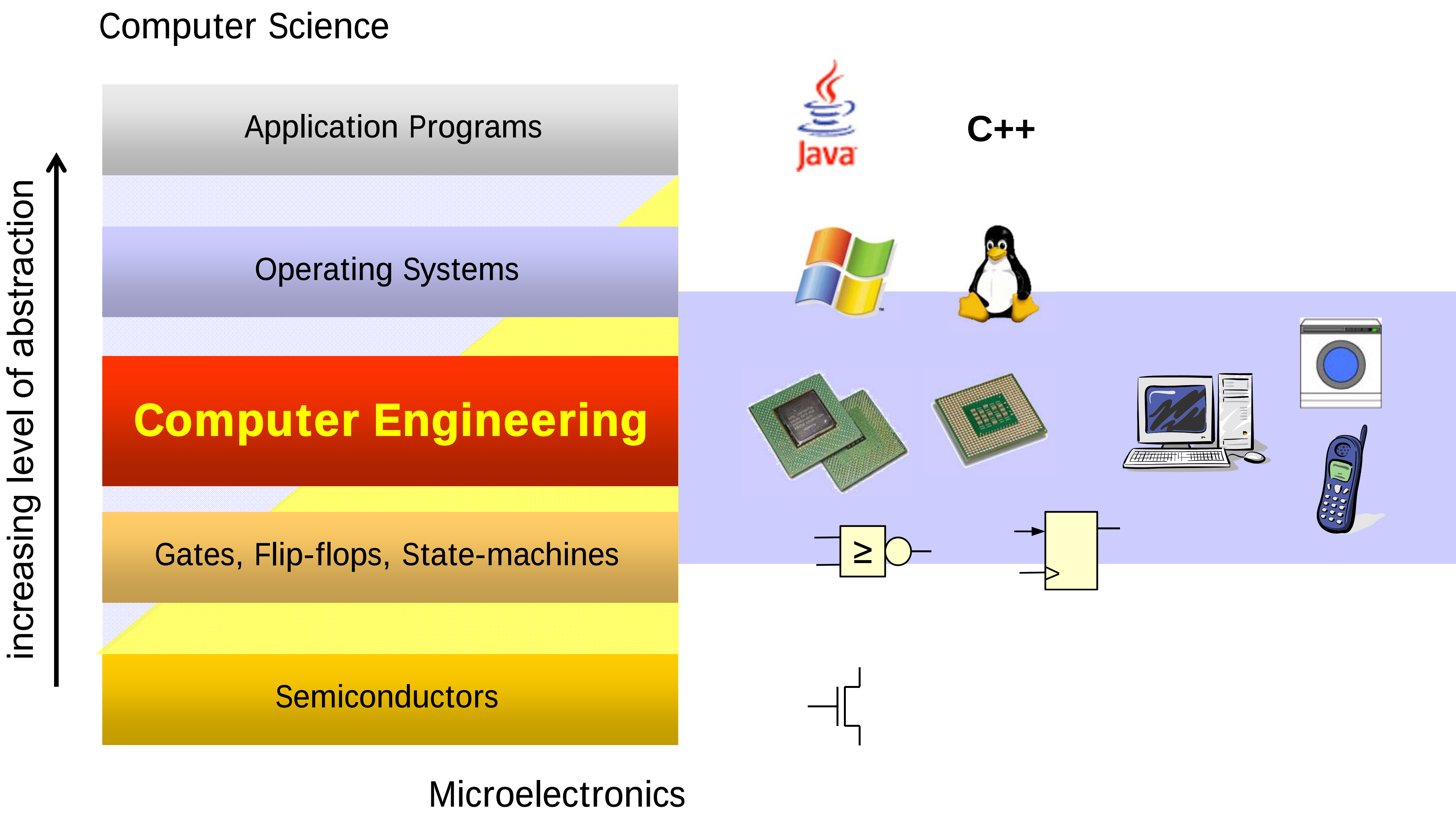
What is Computer Engineering?

- **Computer Engineering** (Technische Informatik)
  - architecture and organization of computer systems
  - combines hardware and software to implement a computer
- Where **Microelectronics** and **Software** meet
  - 70 years of computer hardware
    - 1940s relay / vacuum tubes
    - 1950s transistors
    - 1970s integrated circuits (CMOS[1)])
  - 40 years of software → Computer Science
    - Assembly Language ("Assembler")
    - High Level Language (e.g. C, ...)
    - Object Oriented Programming (C++, Java, ...)
    - Visual Programming (Model Driven Design)

[1)] Complementary metal–oxide–semiconductor

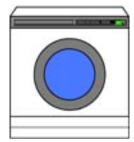
What is Computer Engineering?

School of Engineering
InES Institute of Embedded Systems



Applications

- **Embedded Systems**
  - often part of a larger system
  - control of devices, facilities, processes
  - wireless sensor networks (WSN)
- **Information Technology**
  - communication networks
  - processing of data
  - multimedia
- **Tools**
  - support of technical and scientific activities
  - simulation and modeling
  - logging and analysis of measurement data



Objectives CT 1

After the course you will be able to

- describe the architecture and the operation of a basic computer system and a processor
- to explain how instructions are executed
- to describe the main architectures and performance features of processors as well as the concept of pipelining
- to comprehend how structures in C are compiled into executable object code and to use this knowledge to eliminate programming errors and to optimize program performance
- to develop, debug and verify basic hardware-oriented programs in C and in assembly language
- to explain the concept of interrupts and exceptions and to implement basic interrupt applications
- to find their way in other microprocessor systems

Course Content CT 1

School of Engineering
InES Institute of Embedded Systems

- **Organization of computer systems**
  - Representation of information
  - Program translation
  - Architecture: CPU, Memory, I/O, Bus
- **CPU: Principle of Operation**
  - Instruction set
  - Program execution
  - Memory map, little endian vs. big endian
- **Data transfer**
  - Addressing modes
  - Integer data types, arrays, pointers
- **Arithmetic and logic operations**
  - Computing with the ALU
  - Integer casting
- **Control flow**
  - Compare and jump instructions
  - Structured programming
- **Machine code**
  - Encoding of instructions and operands
- **Subroutines/functions**
  - Parameter passing
- **Exceptional Control Flow**
  - Hardware interrupts, interrupt service routine, vector table
  - Exceptions (Traps)
- **Computer- and processor architectures**
  - von Neumann vs. Harvard
  - Performance features of processors
  - Pipelining
- **Hardware-oriented programming exercises**
  - Working with cross-compiler, assembler, linker, loader and debugger

Objectives for Today's Lesson

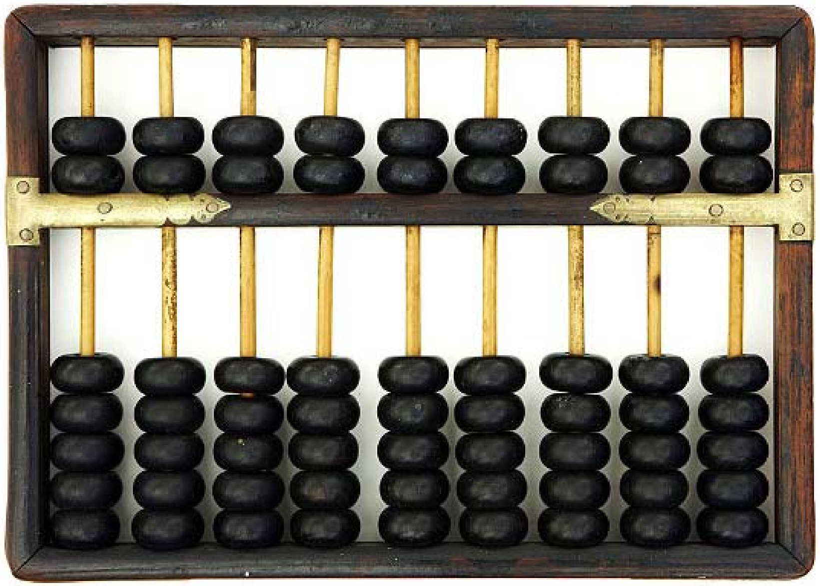
School of Engineering
InES Institute of Embedded Systems

You will be able to

- outline and explain the function of a simple computer system
- name the four main hardware components of a computer system and to describe their functions
- describe different forms of memory and storage
- recall and explain the four translation steps from source code in C to an executable program
- comprehend the use of target and host during development
- explain why knowledge of assembly language is important

Computer History

- **Support for calculations**
  - Babylonian / Chinese — between 1000 und 500 AC: Abacus
  - John Napier — beginning 1600 PC:
    tables for multiplications and logarithms

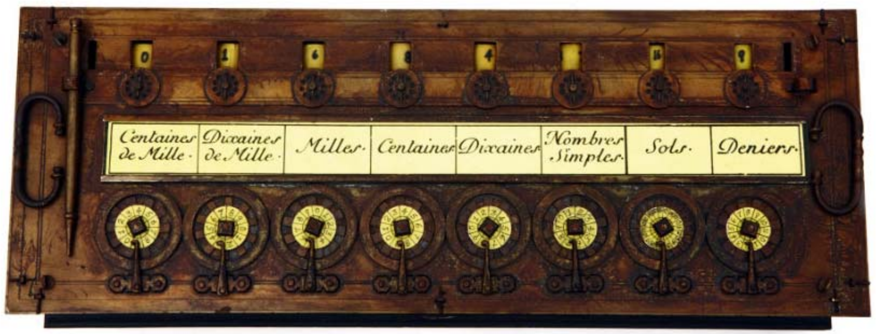


Abacus from *www.computerhistory.org*

Napier's Bones from *www.computerhistory.org*

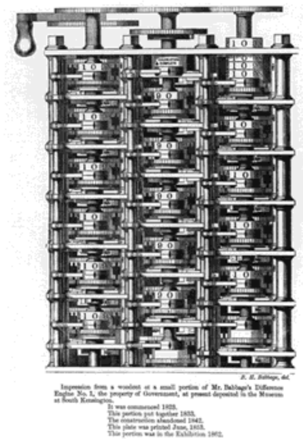
Computer History

- **First mechanical computers: + - (* /)**
  - Leonardo da Vinci (1452 - 1519)
    - around 1500 → rebuilt successfully in 1967
  - Wilhelm Schickard (1592 - 1635)
    - around1625 → no preserved originals, rebuilt
  - **Blaise Pascal (1623 - 1662)**
    - around 1640 → arithmetic machine (Pascaline)
  - Gottfried von Leibnitz (1646 -1716)
    → enhancement of arithmetic machine



replica of a Pascaline from *www.computerhistory.org*

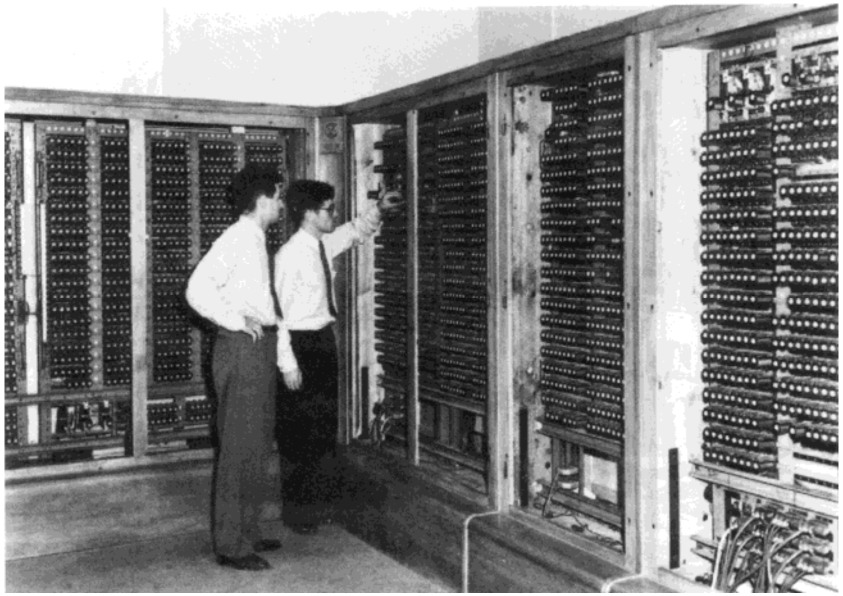
Computer History



- **First mechanical computer in today's sense**
  - Charles Babbage
    - around 1822 "Difference Engine", not completed
    - replaced by "Analytical Machine"
  - Ada Lovelace
    - Mathematician
    - wrote programs for the Analytical Machine
    - Daughter of Lord Byron

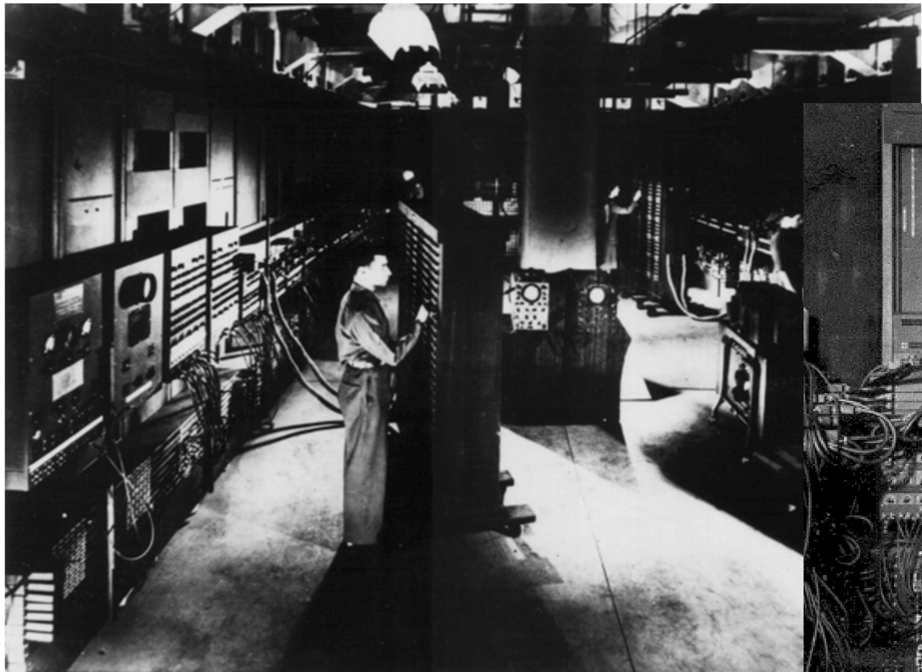
Computer History

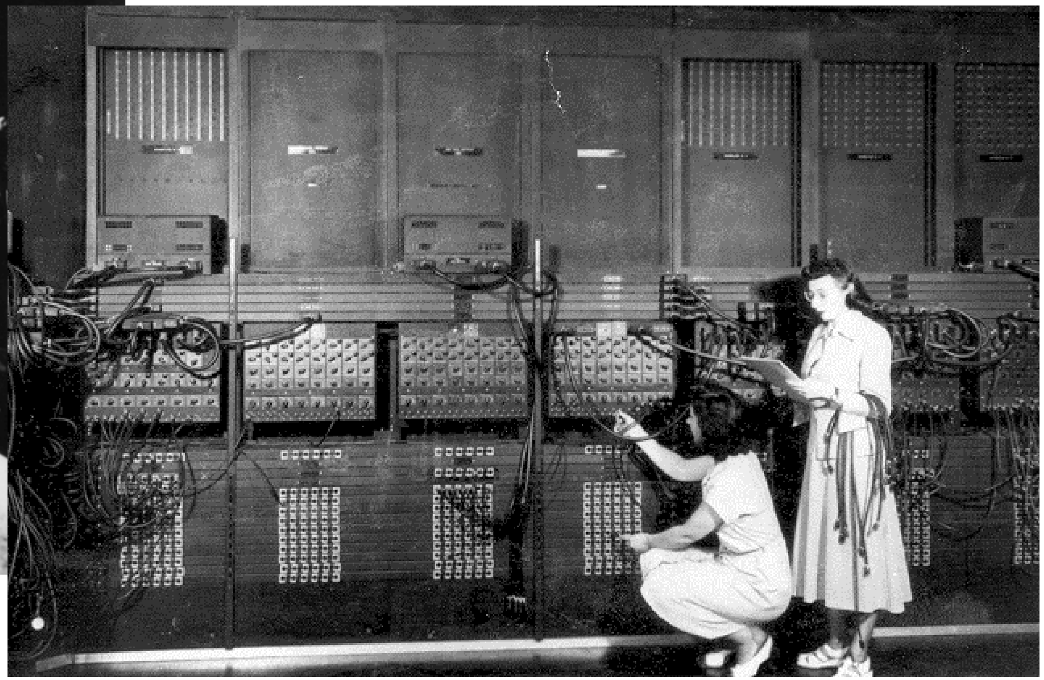
- **First electromechanical computers**
  - Howard H. Aiken
    - Harvard Mark 1, between 1939 and 1944
    - consisting of switches, relays
    - around 750'000 components:
    - 15m x 2.4m x 0.6m, 4500 kg
  - Konrad Zuse, Germany
    - Z3, built in 1941 in Berlin
    - 1944 destroyed by bombing
    - work on Z4 started around 1943
    - used at the ETH from 1950 on



Computer History

- **First electronic "general purpose" computer**
  - J. Presper Eckert and John Mauchly, Univ. of Pensylvenia
    - ENIAC, 1944 → Electronic Numerical Integrator And Calculator
    - around 18'000 tubes, 30 tons, 140 kW, 5'000 additions / s, 1400 m^2

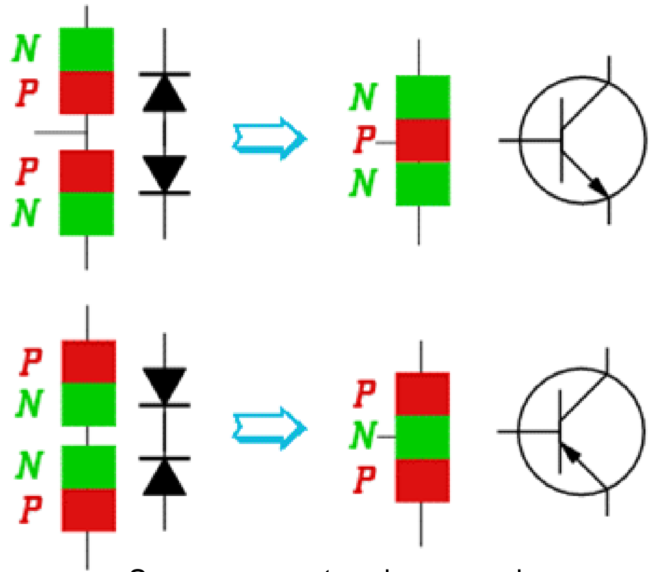




Computer History

■ First transistors

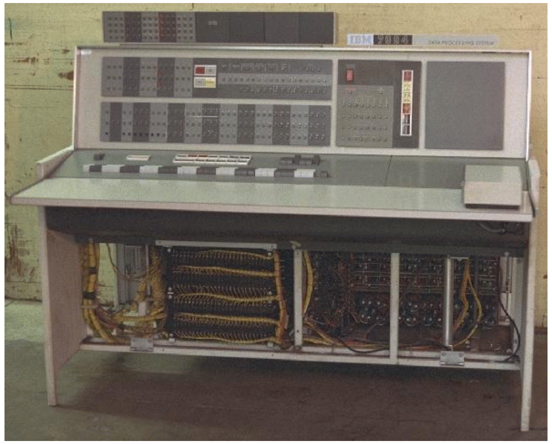
- 1926, patent by Julius Edgar Lilienfeld
- 1947, Germanium transistor
  - W. Shockley, W. Brattain, J. Bardeen
- 1950, Bipolar Transistor
  - William Shockley



Source: www.st-andrews.ac.uk

■ Early transistor-based computers

- around 1957
  - DEC PDP-1
  - IBM 7000
  - NCR & RCA



Computer History

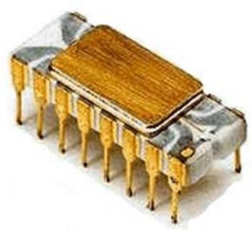
School of Engineering
InES Institute of Embedded Systems

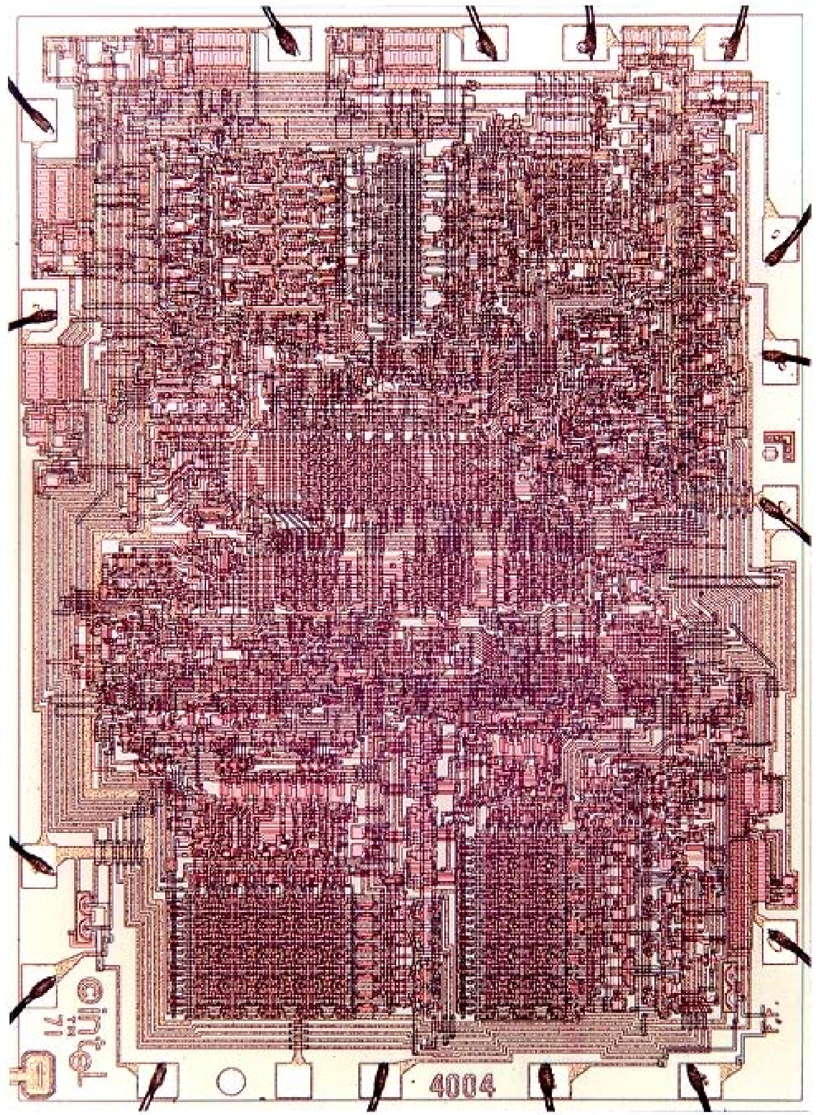
Early integrated circuits (IC)

- 1958, Jack Kilby at Texas Instruments (TI)
  - based on an idea from 1952
  - several components on the same substrate
- 1963, Fairchild, "the 907 device"
  - 2 logic gates
- 1967, Fairchild, "Micromosaic"
  - several 100 transistors
- 1970, Fairchild
  - first 256-bit static RAM

Computer History

- **Early microprocessors**
  - 1971, Intel 4004
    - all CPU components on a single chip
    - 4 Bit, 2300 transistors
    - 12mm^2 (3x4 mm)
  - 1972, Intel 8008
    - 8-bit version of the 4004

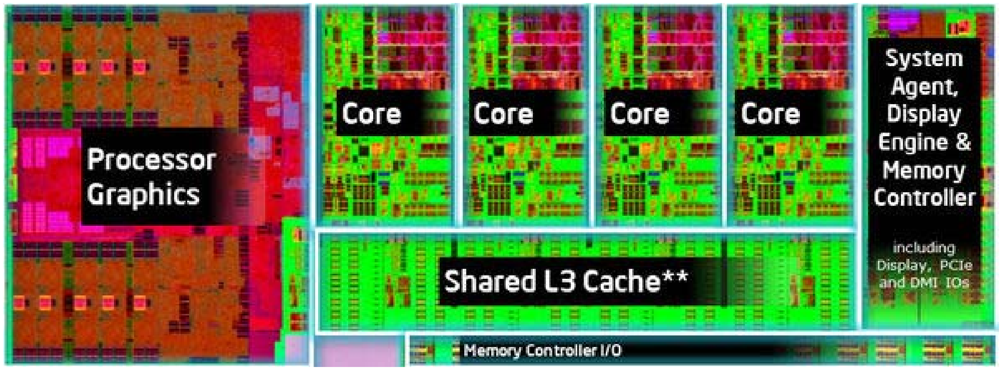




Where are we today?

School of Engineering
InES Institute of Embedded Systems

- **Intel Core i7 quad core**
  - Name Haswell
  - ~1600 Mio. transistors
  - ~177 mm^2
  - 3.5 GHz
  - 22nm gate length

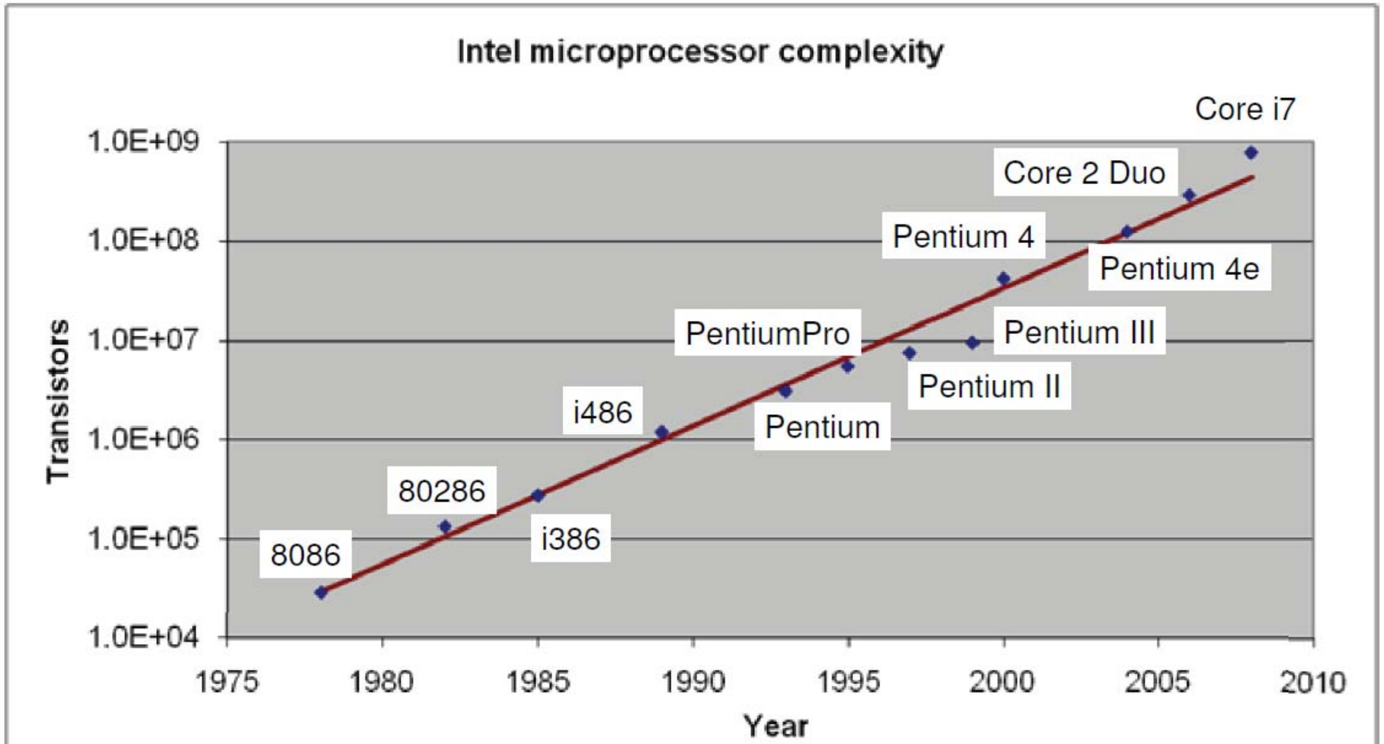


- **Area**
  - A_{i7} = ~15 · A_{4004}

- **Transistors**
  - T_{i7} = ~ 695'000 · T_{4004}

Moore's Law

- **Gordon Moore, Intel**
  - 1965: "The number of transistors per IC doubles every year"
  - somewhat slower since 1965 → doubles every 18 months
  - i.e. exponential growth

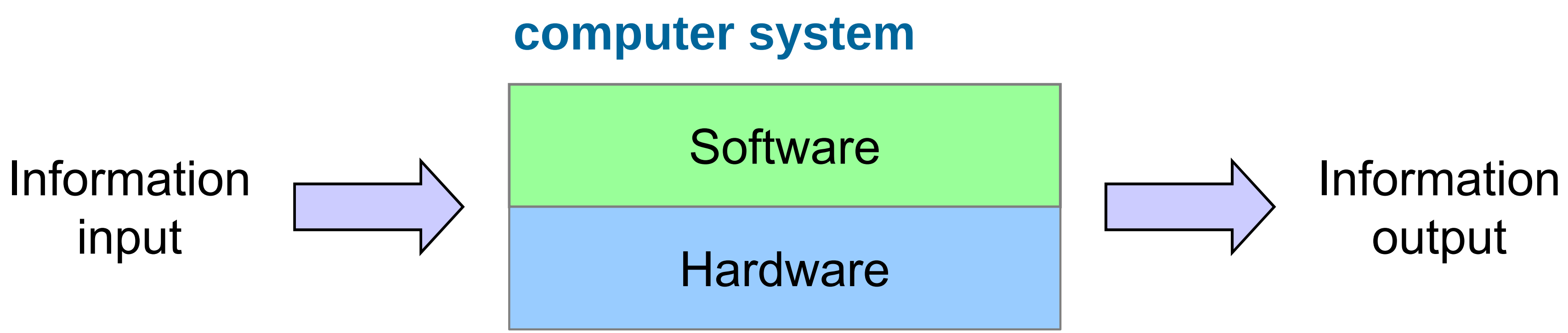


Computer Systems: R. Bryant, D. O'Hallaron

$1.0E+06 = 10^6$

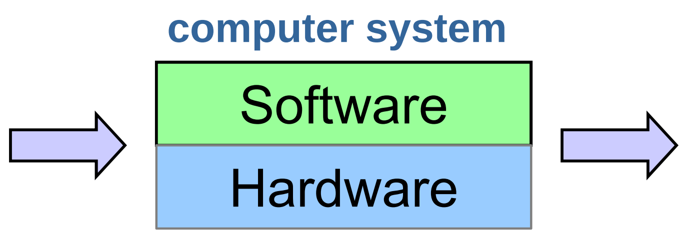
Properties of a Computer System

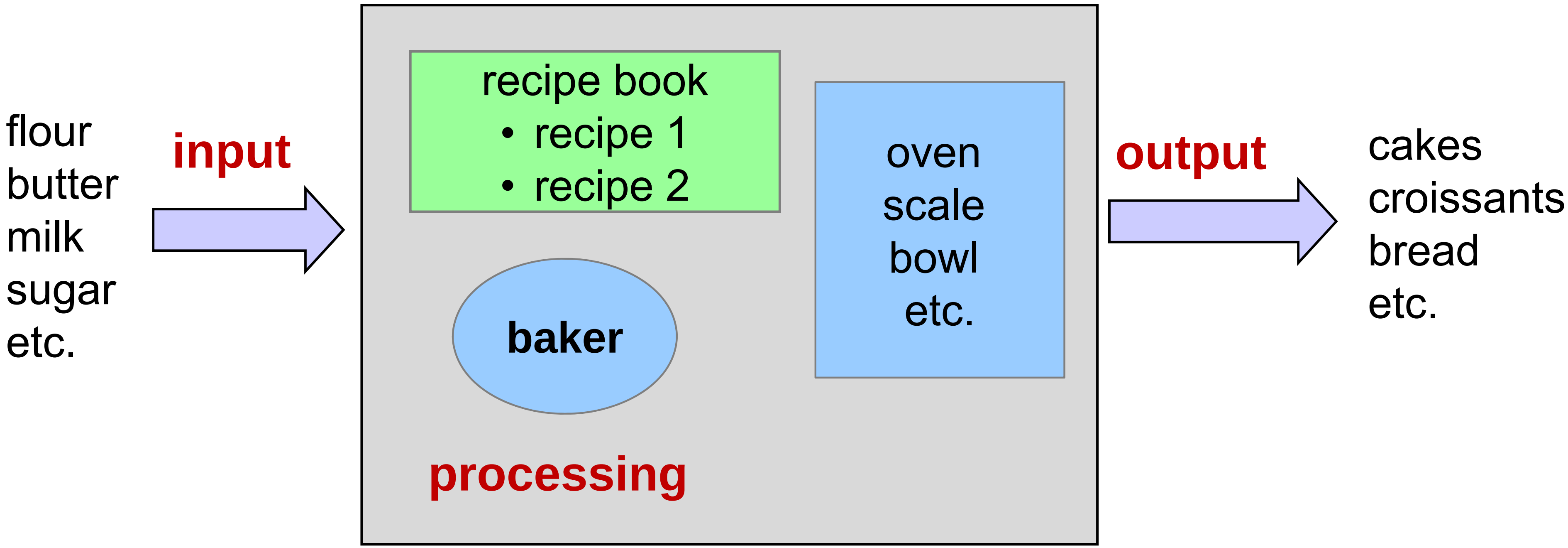
- **A computer system is a device that**
  - processes input
  - takes decisions based on the outcome
  - and outputs the processed information
- **Hardware and software work together → application**
  - often a common hardware is used for many different applications
  - application is defined by the software
    - e.g. controls for washing machines, vending machines, ….



Properties of a Computer System

- **Analogy → bakery**
  - baker ↔ processor
  - recipe book ↔ software
  - tools ↔ HW-resources

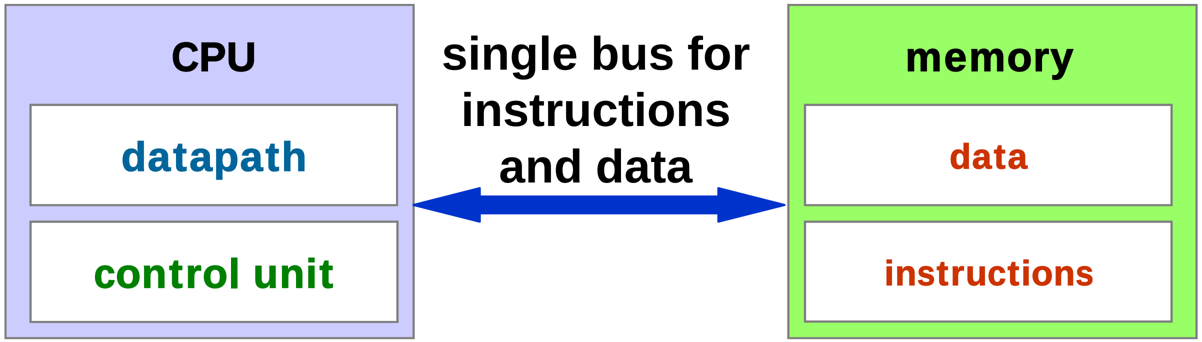




von Neumann Architecture

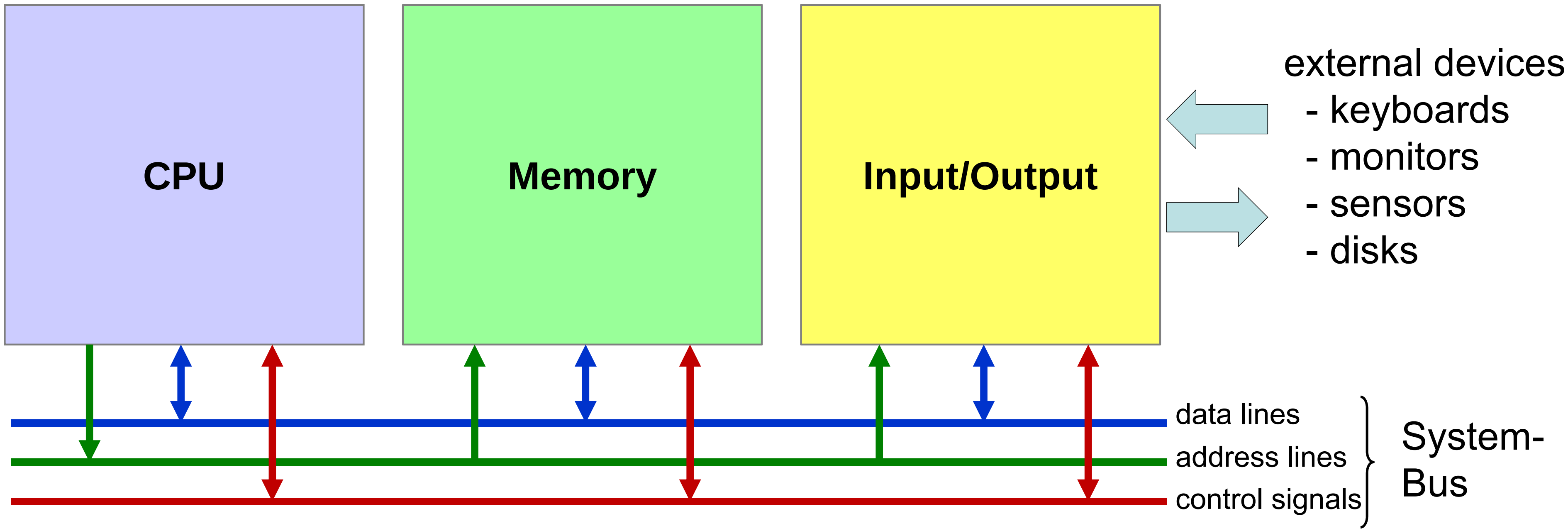
- **Many of today's computers are based on ideas of John von Neumann in the year 1945**
- **Properties**
  - **instructions** and **data** are stored in the same memory
  - **datapath** executes arithmetic and logic operations and holds intermediate results
  - **control unit** reads and interprets instructions and controls their execution

Von Neumann in the 1940s
en.wikipedia.org

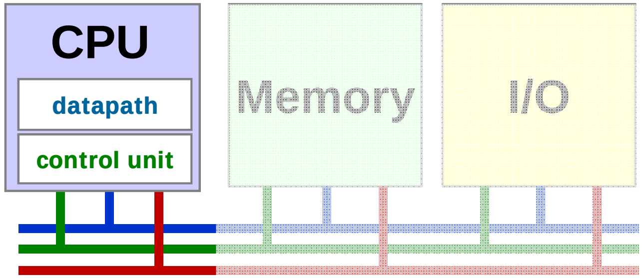


Hardware Components

- **CPU** — **Central Processing Unit or processor**
- **Memory** — **stores instructions and data**
- **Input / Output** — **interface to external devices**
- **System-Bus** — **electrical connection of blocks**



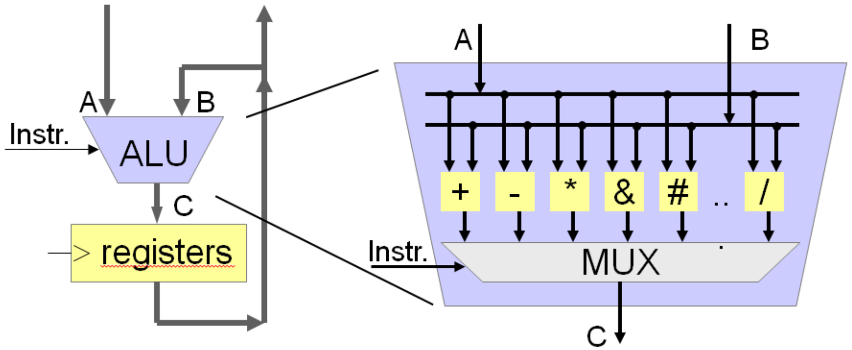
HW Components: CPU



School of Engineering
InES Institute of Embedded Systems

Datapath

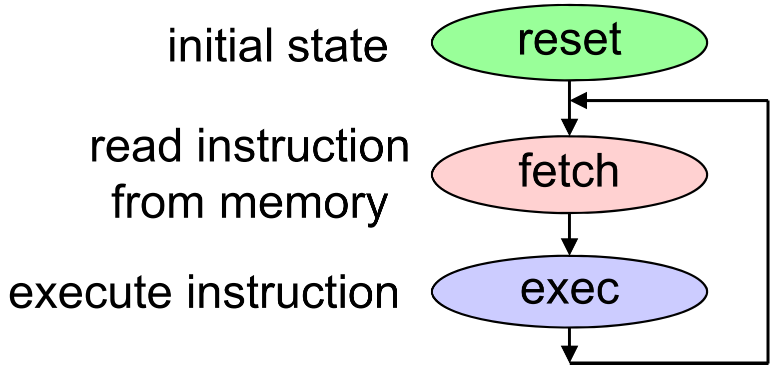
- ALU: Arithmetic and Logic Unit
  - performs arithmetic/logic operations



- registers
  - fast but limited storage inside CPU
  - hold intermediate results
- 4 / 8 / 16 / 32 / 64 bits wide

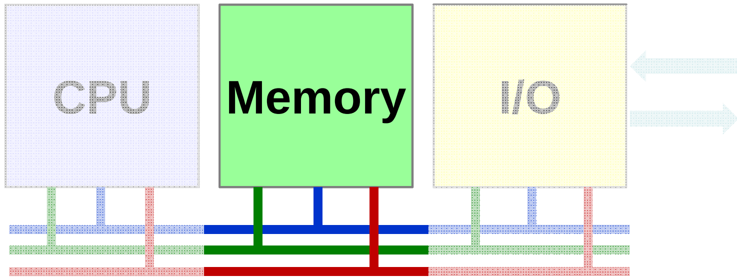
Control Unit

- Finite State Machine (FSM)
  - reads and executes instructions

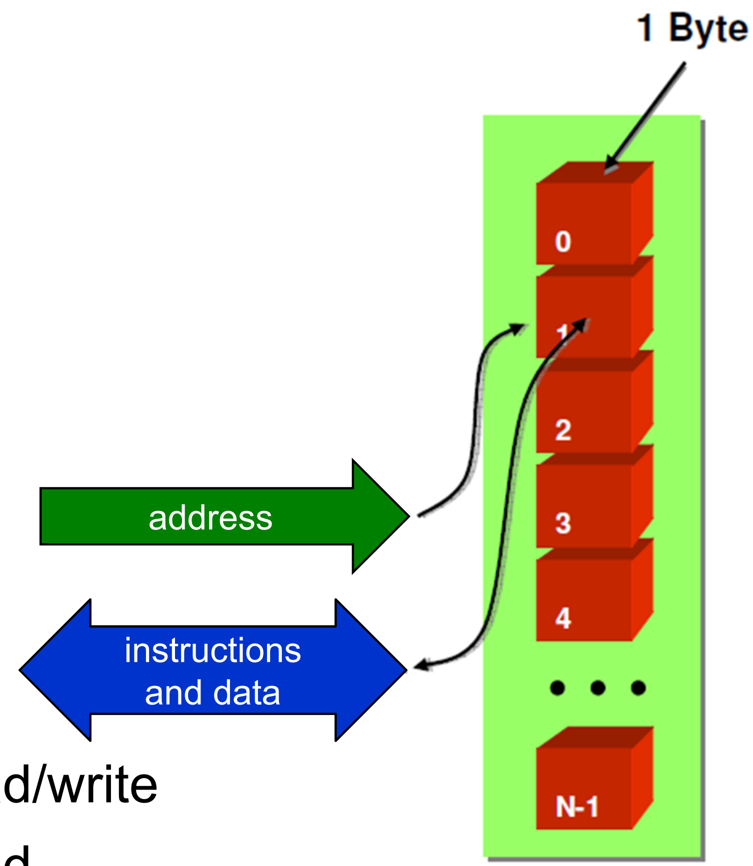


- types of operations
  - data transfer: registers ←→ memory
  - arithmetic and logic operations
  - jumps

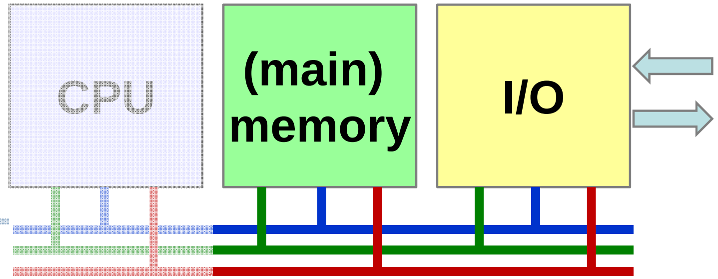
HW Components



- **Memory**
  - a set of storage cells
    - 8 bit → 1 byte
  - smallest addressable unit
    - one byte
    - one address per byte
  - 2^N addresses
    - from 0 to 2^N-1
    - can be read and sometimes written
    - RAM Random Access Memory read/write
    - ROM Read Only Memory read

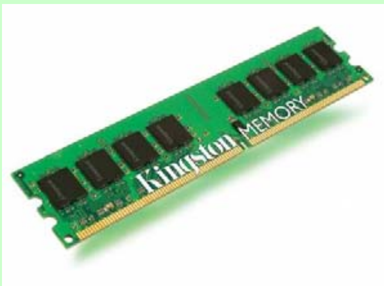


HW Components



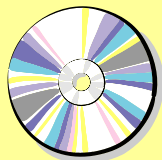
Main memory - Arbeitsspeicher

- central memory
- connected through System-Bus
- access to individual bytes
- volatile (flüchtig)
  - SRAM – Static RAM
  - DRAM – Dynamic RAM
- non-volatile (nicht-flüchtig)
  - ROM factory programmed
  - flash in system programmable



Secondary storage

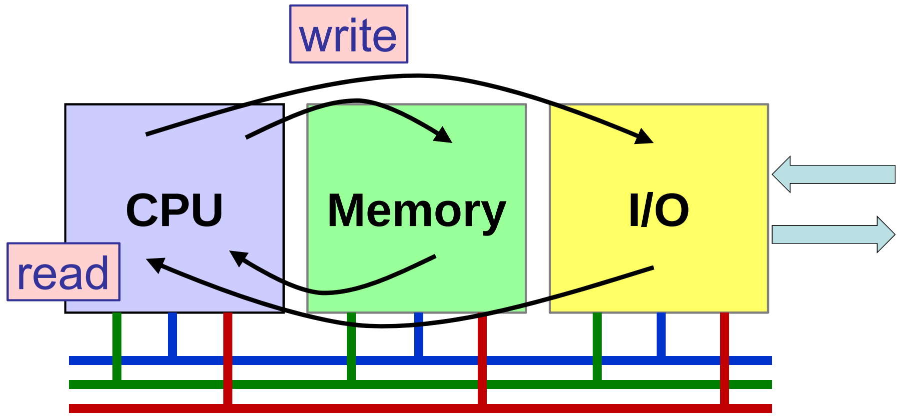
- long term or peripheral storage
- connected through I/O-Ports
- access to blocks of data
- non-volatile
- slower but lower cost
  - magnetic hard disk, tape, floppy
  - semiconductor solid state disk
  - optical CD, DVD
  - mechanical punched tape/card



HW Components

School of Engineering
InES Institute of Embedded Systems

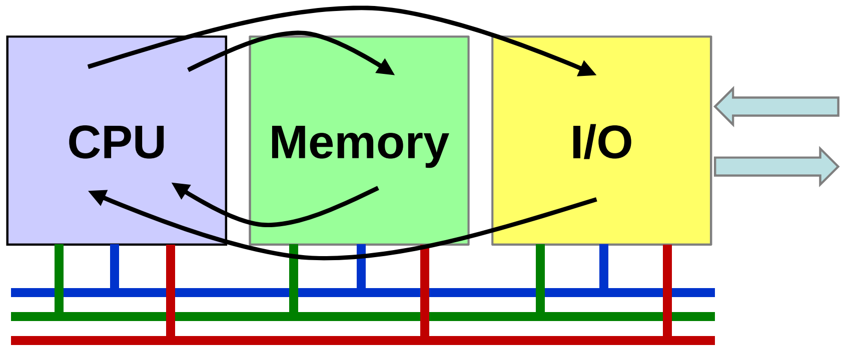
- **System-Bus**
  - CPU writes or reads data from/to memory or I/O



  - **address lines**
    - CPU drives the desired address onto the address lines
      - ► to which address does the CPU write?
      - ► from which address does the CPU read?
      - ► analogy → address on an envelope of a letter
    - number of addresses = 2^n → n = number of address lines
      - ► n = 16 → 2^{16} = 65'536 addresses → 64 KBytes
      - ► n = 20 → 2^{20} = 1'048'576 addresses → 1 MBytes

HW Components

- **… System-Bus**

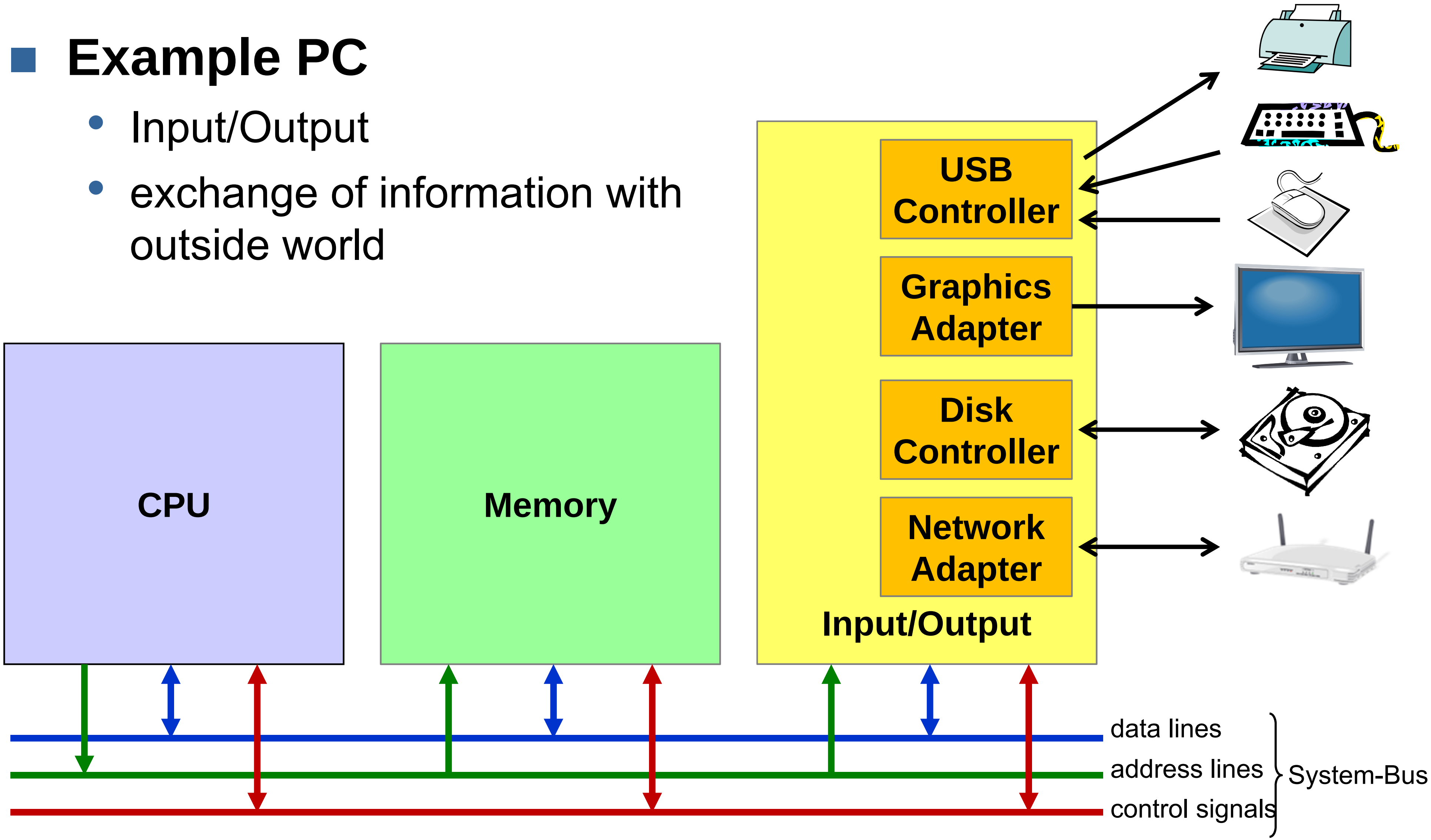


  - **control signals**
    - CPU tells whether the access is read or write
    - CPU tells when address and data lines are valid → bus timing
  - **data lines**
    - transfer of data
      - ▶ analogy the letter that's inside the envelope
      - ▶ write CPU provides data → memory receives data
      - ▶ read CPU receives data ← memory provides data
    - 4/8/16/32/64 data lines

HW Components

Example PC

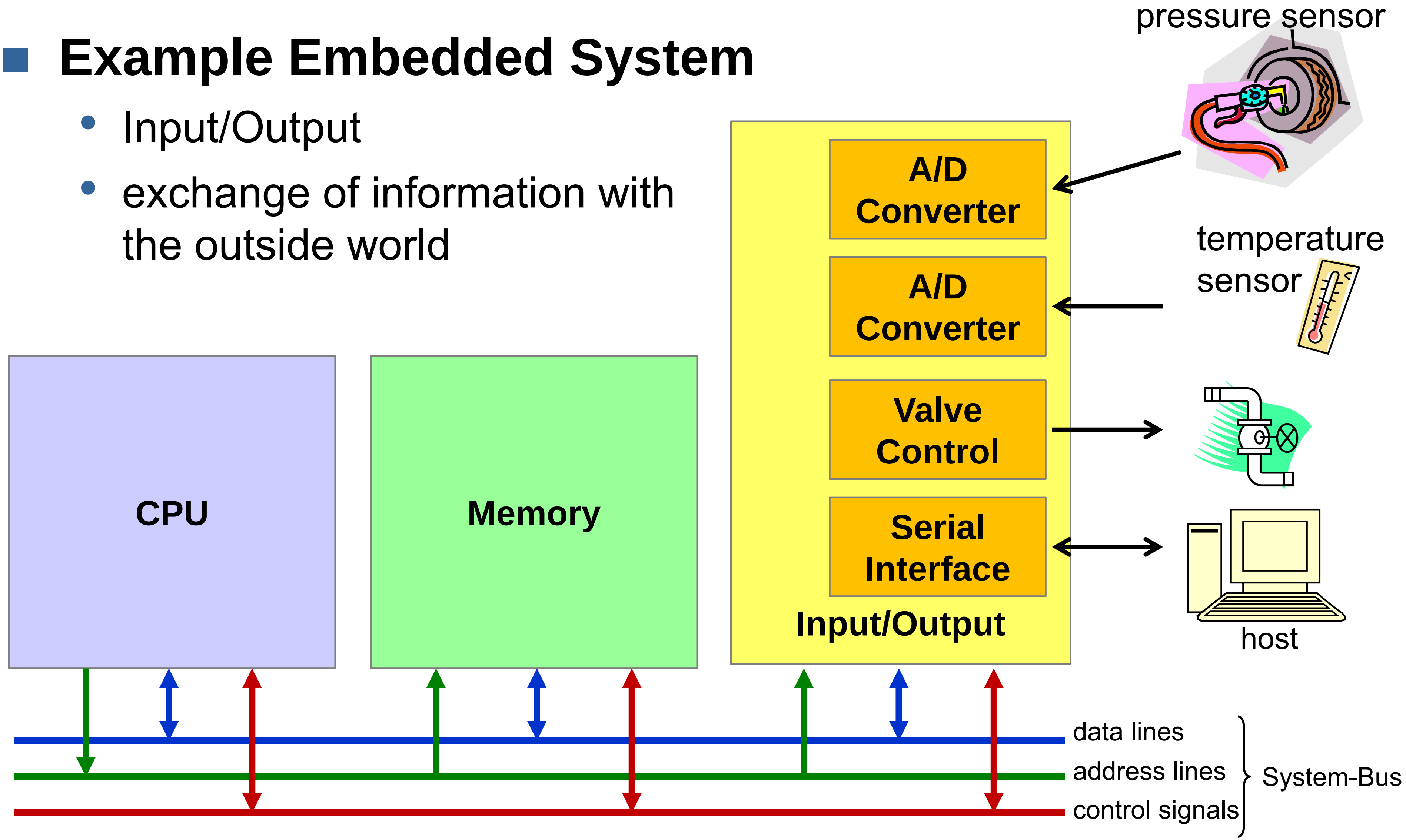
- Input/Output
- exchange of information with outside world



HW Components

Example Embedded System

- Input/Output
- exchange of information with the outside world



Software Aspects

So far

- CPU reads instructions from memory and executes them

But

- How to process a program in a high level language like C so that a CPU can interpret the instructions?
- What is needed for a program in C to allow execution on a CPU?
- What does the path from the C source code to the executable object file look like?

Software Aspects

- **Programmer writes `hello.c` in a text editor**

```
#include <stdio.h>

int main(void) {
    printf("hello world\n");
}
```

- **`hello.c` is stored in ASCII format on disk**

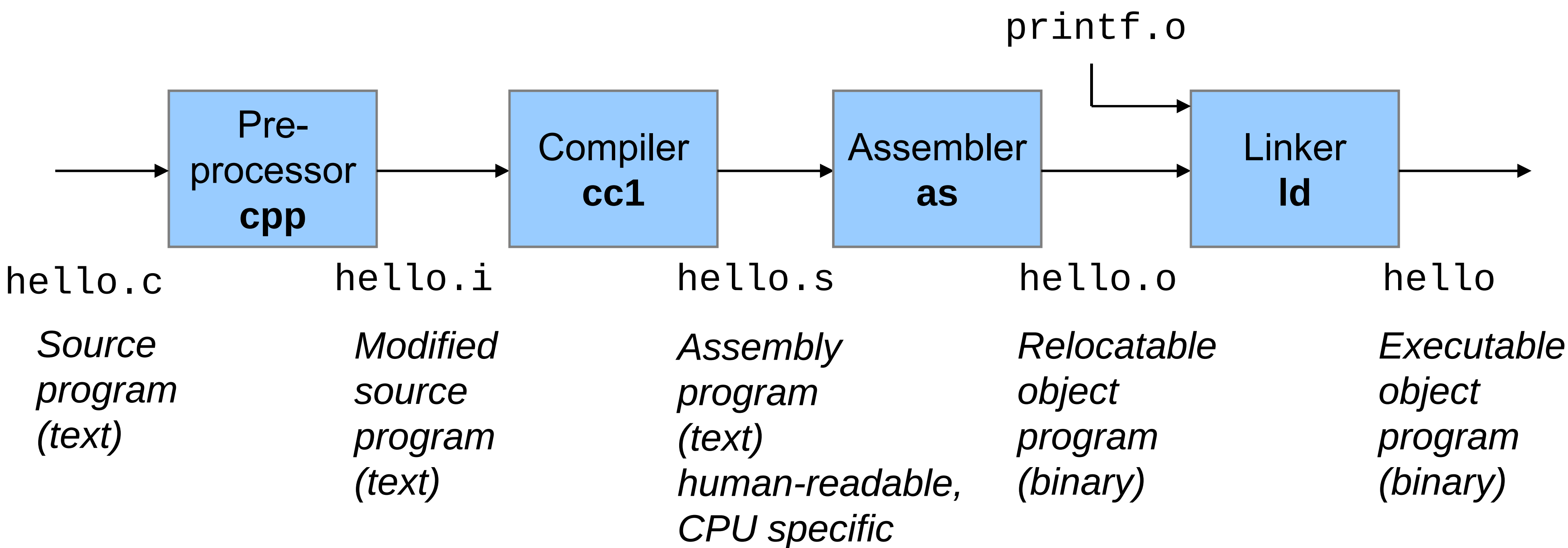
#	i	n	c	l	u	d	e	<sp>	<	s	t	d	i	o	.
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111	46
h	>	\n	\n	i	n	t	<sp>	m	a	i	n	(	)	\n	{
104	62	10	10	105	110	116	32	109	97	105	110	40	41	10	123
\n	<sp>	<sp>	<sp>	<sp>	p	r	i	n	t	f	(	"	h	e	l
10	32	32	32	32	112	114	105	110	116	102	40	34	104	101	108
l	o	,	<sp>	w	o	r	l	d	\	n	"	)	;	\n	}
108	111	44	32	119	111	114	108	100	92	110	34	41	59	10	125

Software Aspects

School of Engineering
InES Institute of Embedded Systems

From C to executable

- Each C-Statement in `hello.c` has to be translated into a sequence of instructions in machine language
- Example **gcc** (The GNU Compiler Collection)
- `gcc hello.c` calls 4 different programs

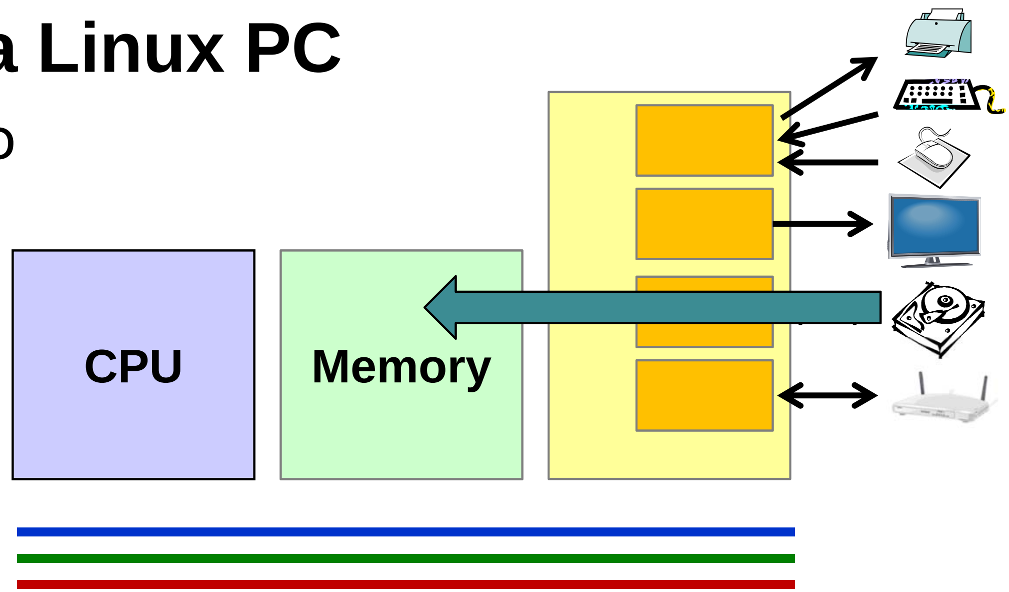


R. Bryant/D. O'Hallaron

Software Aspects

Program execution on a Linux PC

- load executable **`hello`** into memory and execute it

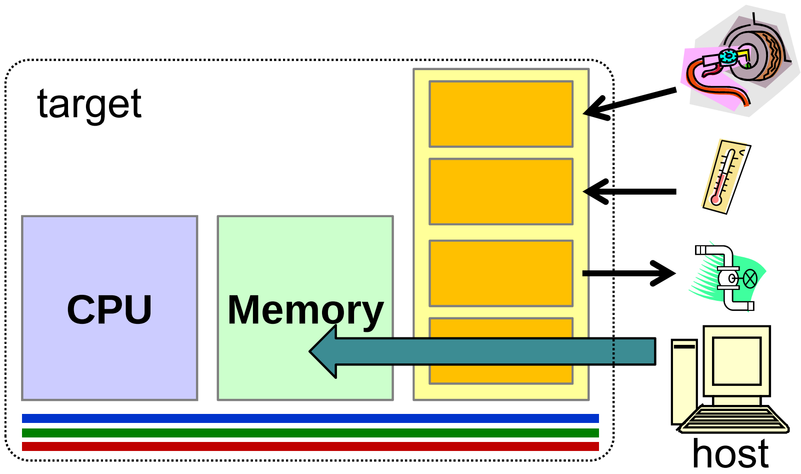


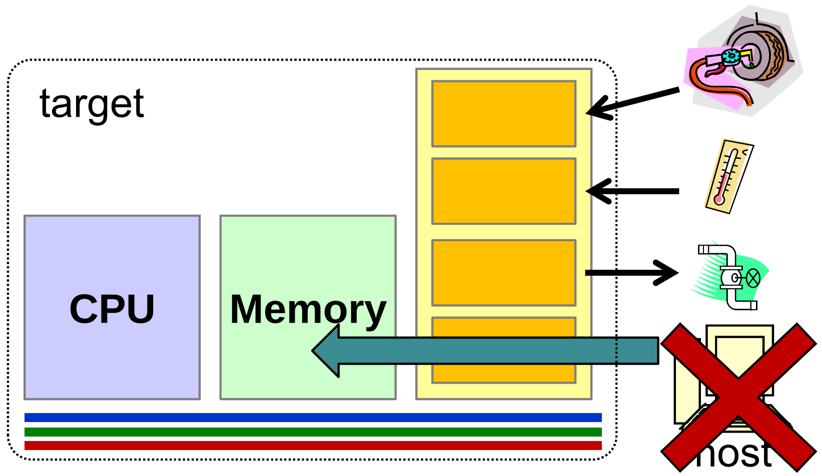
- typing **`./hello`** in a shell
  - transfers the executable from disk to memory (RAM)
- operating system
  - creates a new process
  - jumps to start of **`main()`** function and begins execution

Software Aspects

Program execution on Small Scale Embedded System

- Host vs. Target





Software development on host

- Compiler/Assembler/Linker on host
- Loader on target loads executable from host to RAM
- Loader copies executable from RAM into non-volatile memory (FLASH)
  → Firmware Update

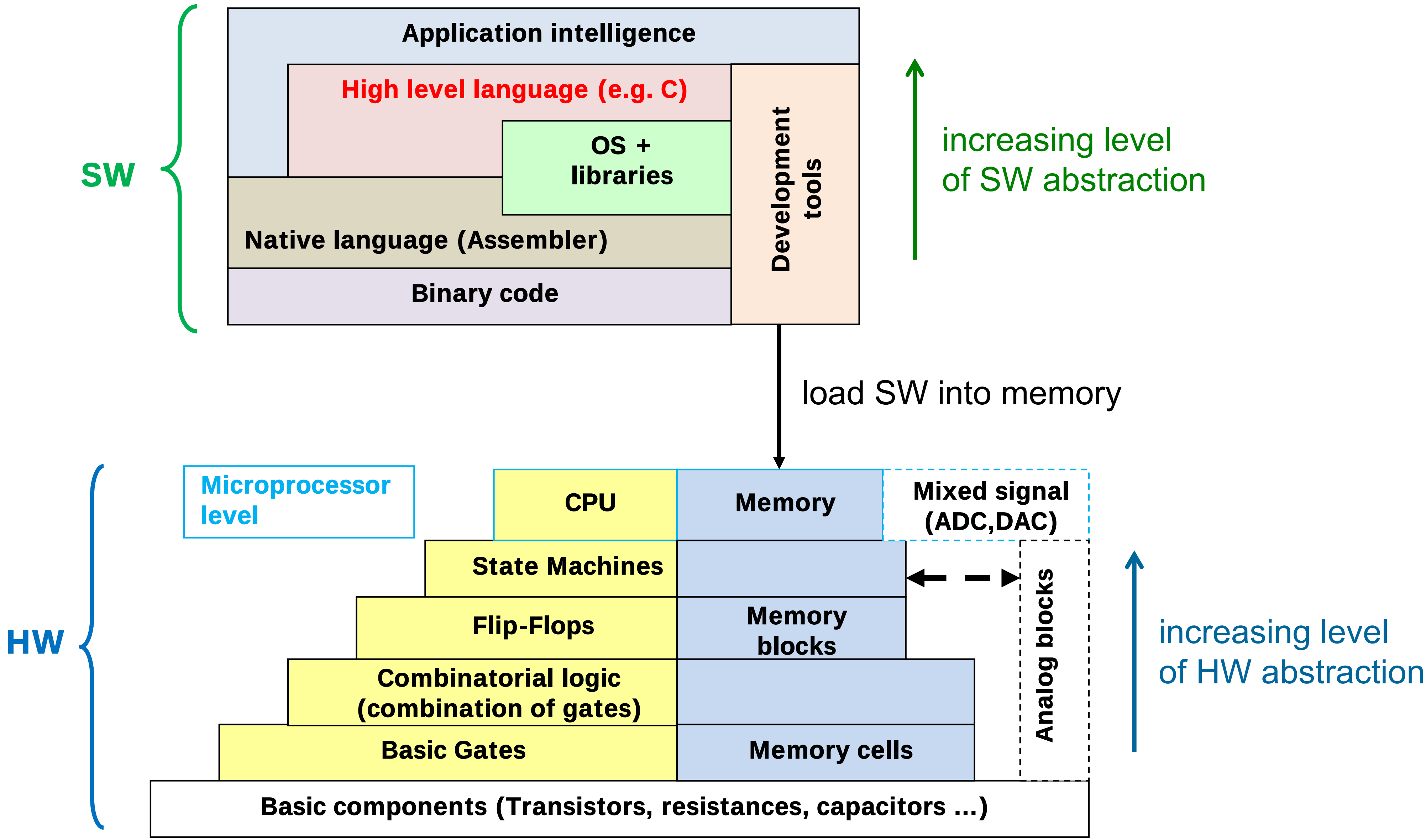
System operation <u>without</u> host

- Loader jumps to `main()` and starts execution
- Instruction fetch often takes place directly from FLASH

Software Aspects

- **Why learn assembly language?**
  - few engineers write assembly code
    - use of High Level Languages (HLL) and compilers more efficient
- **But**
  - assembly language yields understanding on machine level
    - understanding helps to avoid programming errors in HLL
  - increase performance
    - understand compiler optimizations
    - find causes for inefficient code
  - implement system software
    - boot Loader, operating systems, interrupt service routines
  - localize and avoid security flaws
    - e.g. buffer overflow

Interaction of HW and SW



Conclusion

- **Computer system → hardware and software**
- **Hardware**
  - CPU — Central Processing Unit or microprocessor (µP)
  - memory — stores instructions and data
  - I/O — input and output devices
  - system bus — electrical connection of blocks
- **Software**
  - source code in high level language (C)
  - assembly code → machine-oriented, human readable
  - object code → machine instructions in binary without libraries
  - executable → executable object file including libraries
- **Target vs. Host**