Introduction to computer science:
historical perspective, computer structure, operating systems & languages

GIGA Doctorate School

Christophe Phillips, Ir Ph.D.

Program

▶ Historical perspective

▶ Computer structure

▶ Operating systems

▶ Programming languages

Some wisdom…

"1. Start simple. 2. Get it to work. 3. Then, add complexity."

\- Tom Bredemeier

"One of the best programming skills you can have is knowing when to walk away for awhile."

\- Oscar Godson

Program

▶ Historical perspective

▶ Computer structure

▶ Operating systems

▶ Programming languages

Historical perspective

*Computer science did NOT suddenly appear during World War II out of a genius mind.*

Three parallel streams:

▶ Calculation instruments, from abacus to Pascal's mechanical calculator

▶ Mathematical logic, from al-Khwarizmi (VIIIth century) to Alan Turing (XXth century)

▶ Automats, from antiquity (e.g. Hero of Alexandria's 1st vending machine) to 'Jacquard loom', and great watch & clock makers

Historical perspective

▶ 1837, the "Analytical Engine",
described by **Charles Babbage**
  = 1st mechanical general-purpose computer, including:
  - arithmetic logic unit + integrated memory
  - control flow in the form of conditional branching and loops

▶ 1843, "algorithm" for the Analytical Engine,
by **Ada Lovelace**
  = 1st software (to calculate Bernoulli numbers)
  - set of instructions to solve problems of any complexity
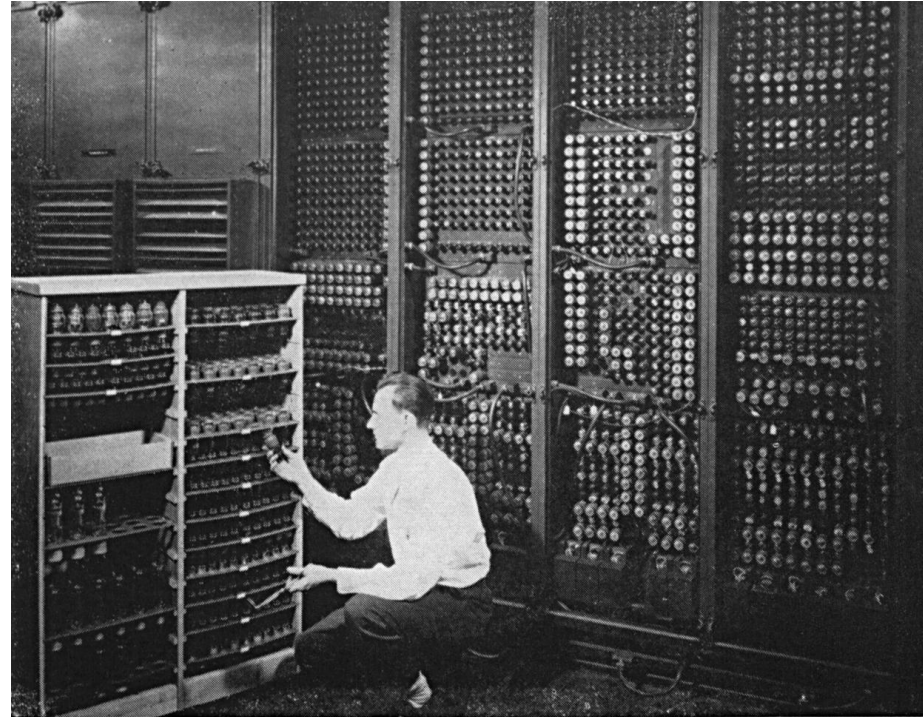 - symbolic representation by numbers of letters, musical notes, etc.
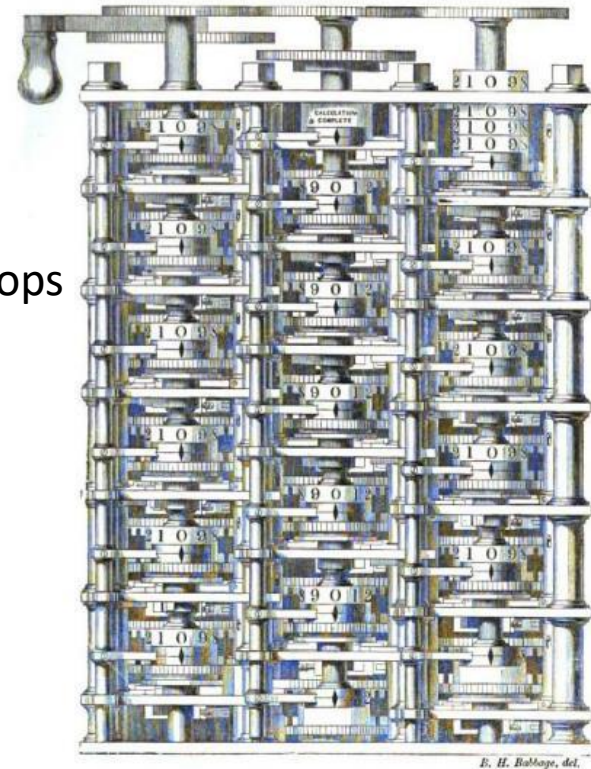
Historical perspective

- 1943-1945, Colossus computer, UK.

- 1945-1956, ENIAC (Electronic Numerical Integrator and Computer), USA.

Programs hard coded into the machines with 10000's of switches and plugs!

(…and bugs were real!)

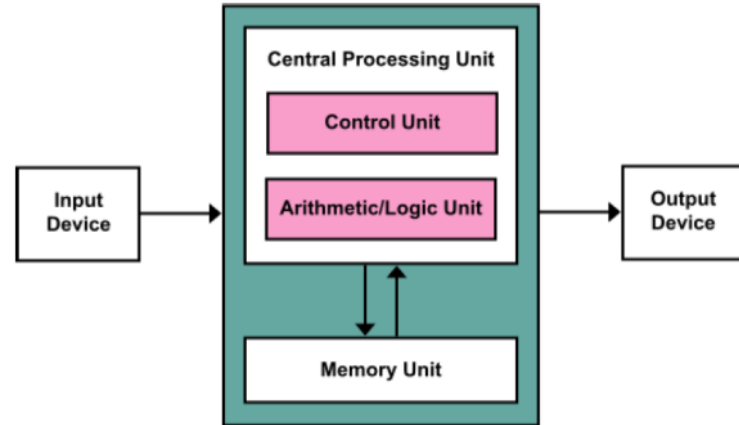Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

Von Neumann architecture

▶ Proposed by Von Neumann in 1945
(unifies Babbage's Analytical Engine
and abstract Turing machine)

▶ Fundamental ideas:

- Bring *input, output* and *program* in "memory unit"

- Operate only on this memory

→ Stored-program computer keeps both <u>program instructions</u>
and <u>data</u> in read-write, "random-access memory" (RAM)!

Hardware innovations

▶ **In the 1950's & 1960's,** tubes are replaced by transistors then integrated circuits.
  → higher density + more reliable
   + less energy consumption (heat!)

▶ **In 1965, Moore's law:**

The number of transistors in an IC doubles every 18 months!

Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.
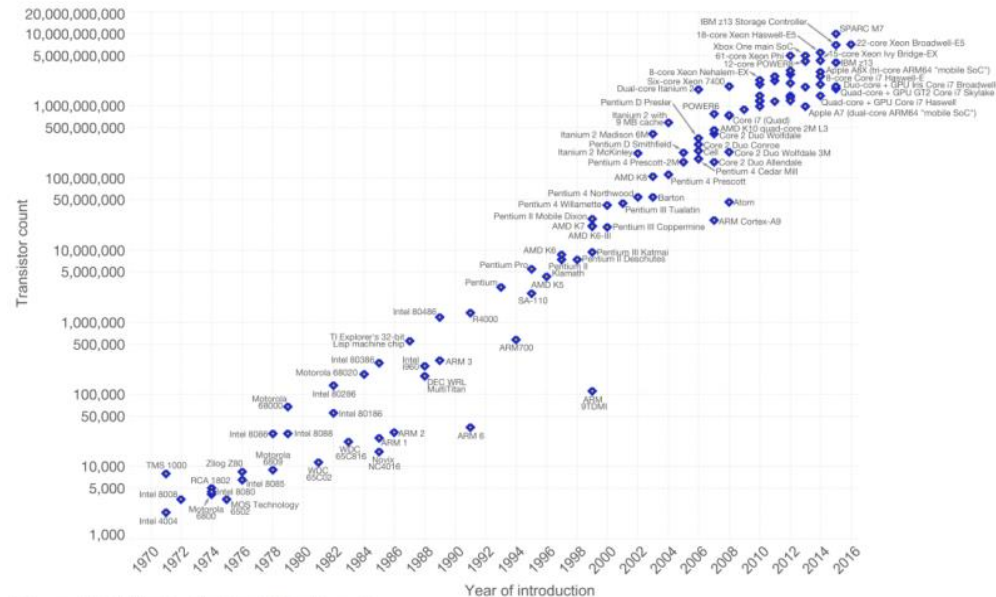
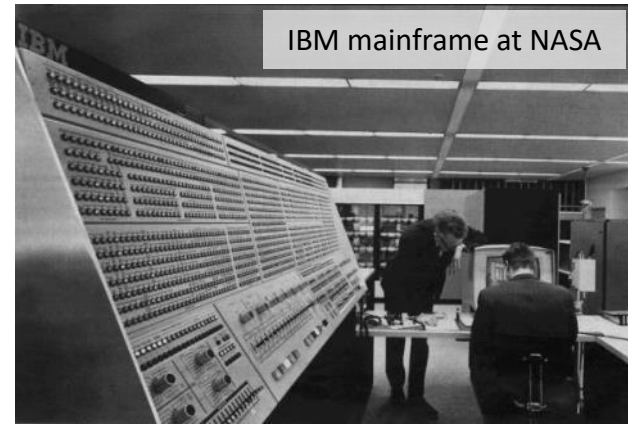# Mainframes vs. mini-computers

1960's & 1970's, development of :

▶ **mainframes**

  - Large centralised infrastructure
    → high performance

  - Passive terminals
    → submit 'batches'

▶ **mini-computers**

  - All-in-one machine
    → direct interaction

  - Small and cheap (actually still pretty big and expensive…)

IBM mainframe at NASA

Micro-computer

In 1971, first Intel microprocessor

→ All main elements of a computer in 1 integrated circuit & no wiring, except on 'motherboard'

→ the **micro-computer**

In 1975, Altair 8800 (Bill Gates & Paul Allen) & Apple 1 (Steve Jobs and Steve Wozniak)

Personal Computer (PC)

▶ 1977, TRS-80

▶ 1979, Apple 2 with 1st spreadsheet software

▶ 1982, Commodore 64 → gaming

▶ 1982, IBM-PC with

- Intel "x86" architecture (still used now)

- MS-DOS from Microsoft

Software industry

▶ Software, originally part of the computer and thus "free"

▶ Increasing distinction between "hardware" and "software"

▶ Since the 70's a 80's, more standardized hardware

→ standardized and specific software:

› operating system: Unix, MS-Dos (later on Windows), Macintosh System 1 (later on Mac OS), Linux,…

› applications: spreadsheet, text editing, games, image & audio processing, high-level programming,…

Algorithm

Definition:

an algorithm is an unambiguous specification of how to solve a class of problems.

An algorithm
- ▶ expressed within a finite amount of space and time
- ▶ in a well-defined formal language for calculating a function.
- ▶ starting from an initial state and initial "input",
- ▶ the instructions describe a computation that, when executed, proceeds through a finite number of well-defined successive states,
- ▶ eventually producing "output" and terminating at a final ending state.

Turing machine

The (abstract) Turing machine models a <u>machine</u> with

▶ tape = infinite series of cells with 1's or 0's

▶ control unit = finite set of elementary instructions

▶ Input/output = to read, write or move the tape

Example: *"in state 42, if the symbol seen is 0, write a 1; if the symbol seen is 1, change into state 17; in state 17, if the symbol seen is 0, write a 1 and change to state 6; etc."*

Given any computer algorithm, a Turing machine capable of simulating that algorithm's logic can be constructed.

References
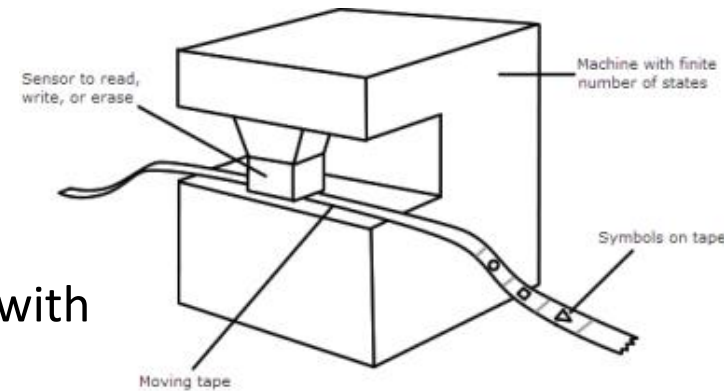
- https://en.wikipedia.org/wiki/Abacus
- https://en.wikipedia.org/wiki/Mechanical_calculator
- https://en.wikipedia.org/wiki/Muhammad_ibn_Musa_al-Khwarizmi
- https://en.wikipedia.org/wiki/Alan_Turing
- https://en.wikipedia.org/wiki/Jacquard_loom
- https://en.wikipedia.org/wiki/Hero_of_Alexandria
- https://en.wikipedia.org/wiki/Charles_Babbage
- https://en.wikipedia.org/wiki/Analytical_Engine
- https://en.wikipedia.org/wiki/Mechanical_computer
- https://en.wikipedia.org/wiki/Ada_Lovelace
- https://en.wikipedia.org/wiki/Bernoulli_number
- https://en.wikipedia.org/wiki/Colossus_computer
- https://en.wikipedia.org/wiki/Algorithm
- https://en.wikipedia.org/wiki/Turing_machine
- https://en.wikipedia.org/wiki/Von_Neumann_architecture
- https://en.wikipedia.org/wiki/Moore%27s_law
- https://en.wikipedia.org/wiki/Computer

Program

▶ Historical perspective

▶ **Computer structure**
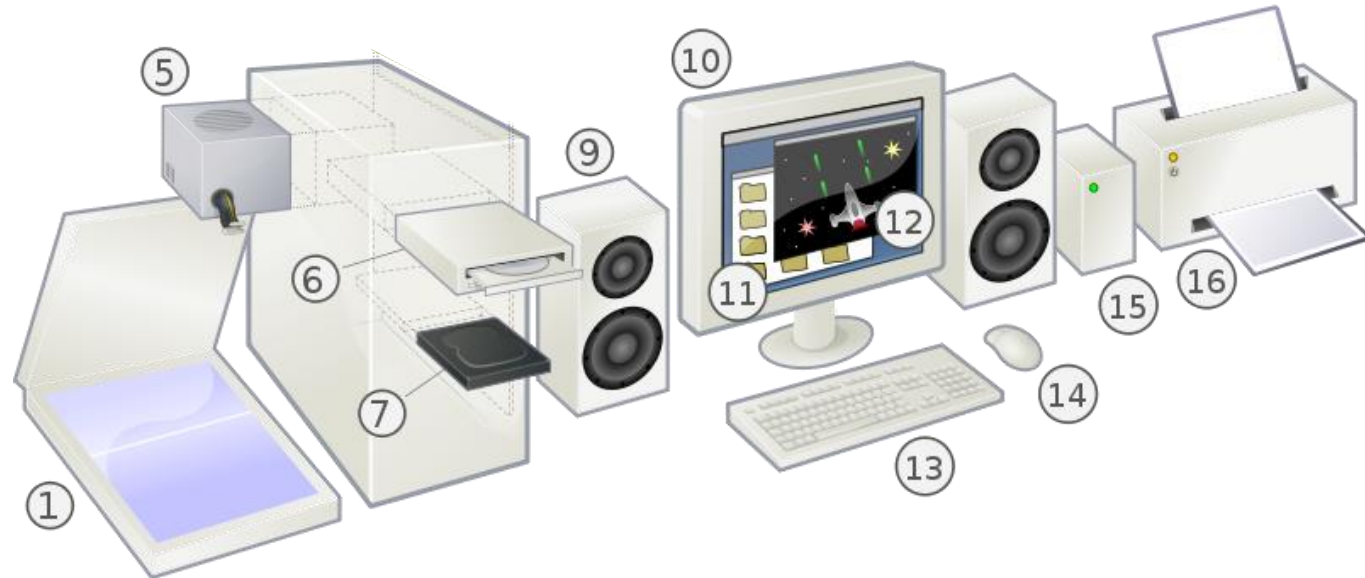
▶ Operating systems

▶ Languages
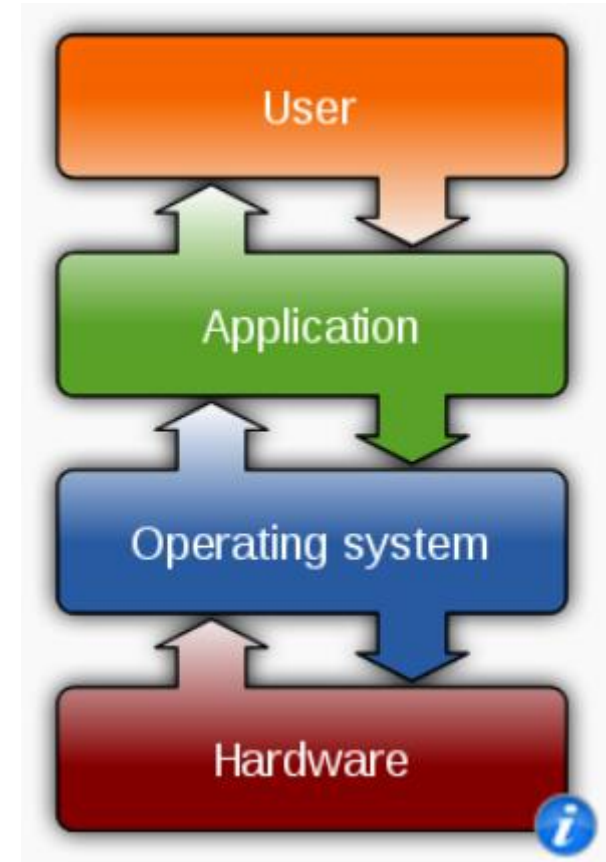
Outside the "motherboard"

Peripherals:

screen, keyboard, mouse, speaker, printer, scanner, power supply,... plugged to the "motherboard"

The "motherboard"

- CPU = microprocessor
  - executes the instructions.
  - includes very fast local memory locale, aka "cache"
- RAM = central memory
  - quickly read/write instructions and data
  - lost when power is off
- GPU (*Graphics Processing Unit*)
  - like CPU but parallelized infrastructure
  - generates and stores image frames
- Northbridge = connecting fast components
- Southbridge = handles slower input/output
  - hard-drives (internal/external)
  - USB peripherals (keyboard, mouse, USB stick,…)
  - network connexions, incl. Wi-Fi & cable
- Inter-connexion through "data bus"
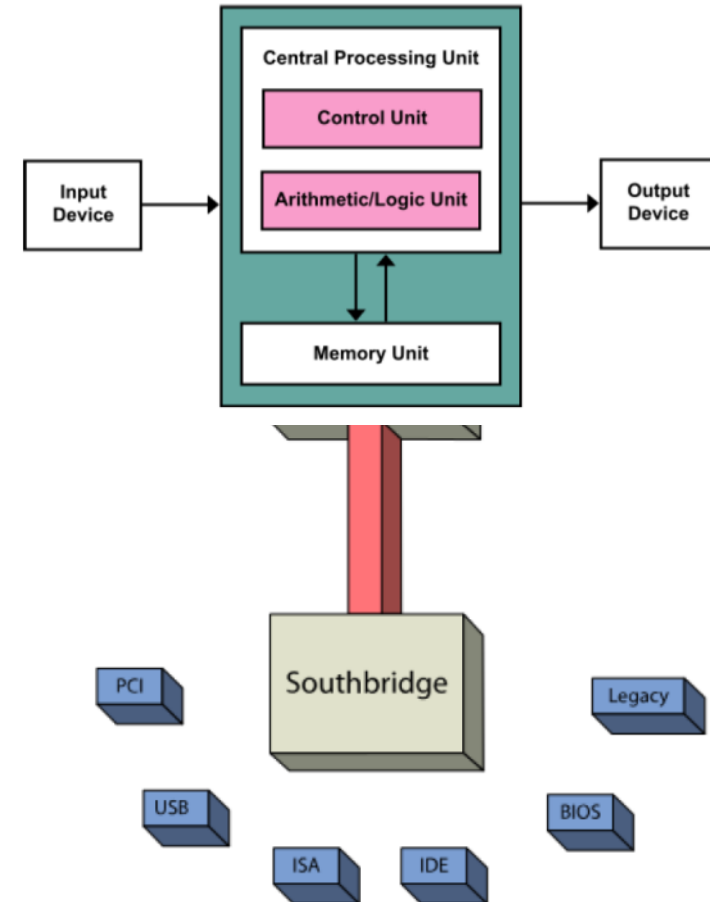
Different types of memory

▶ RAM,
- fast but not persistent $\rightarrow$ used by microprocessor (data & operations)
- limited to a few Gb

▶ Hard drive
- slower but persistent $\rightarrow$ used to store data, code, OS
- up to several Tb

▶ Cache
- inside the CPU $\rightarrow$ super fast but built in

▶ ROM/BIOS (Basic Input Output System)
- boot firmware and power management firmware

Faster & more power

Get some

▶ multi-core processor
▶ higher clock speed
▶ larger RAM
▶ faster data transfer

Caveats

▶ need specific software/compiler to parallelize operations
▶ depend on nature of data and processing pipeline
▶ depend on mass-storage solution (access & r/w time!)
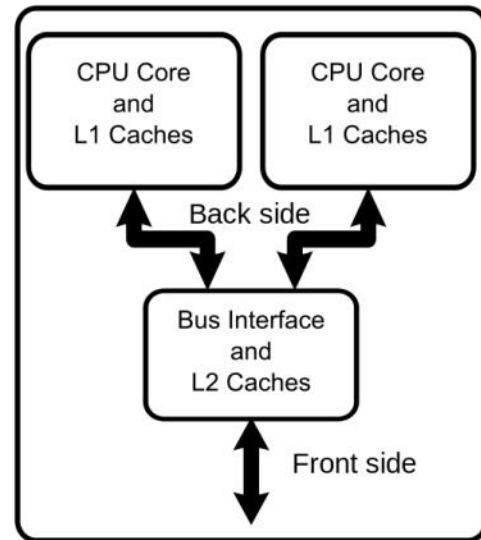
Faster & more power, cluster

High performance computing (HPC) with a cluster
- set of connected computers with multi-core processor, split into "nodes".
- parallelization of the processing, at low or high level
- shared resources

Caveats
- shared resources
- specific code & no GUI
- data & results transfer and management

Consortium des Équipements de Calcul Intensif

CECI, available & accessible at ULiege:

| Cluster	Host	CPU type	CPU count*	RAM/node	Network	Filesystem**	Accelerator	Max time	Preferred jobs***
NIC5	ULiège	Rome 2.9 GHz	4672 (73 x 64)	256 GB..1 TB	HDR Ib	BeeGFS 520 TB	None	2 days	▦ MPI
Hercules2	UNamur	Naples 2 GHz SandyBridge 2.20 GHz	1024 (30 x 32 + 2 x 64) 512 (32 x 16)	64 GB..2 TB	10 GbE	NFS 20 TB	None	15 days	⋮ serial / ≡ SMP
Dragon2	UMons	SkyLake 2.60 GHz	592 (17 x 32 + 2 x 24)	192..384 GB	10 GbE	RAID0 3.3 TB	4x Volta V100	21 days	⋮ serial / ≡ SMP
Lemaitre3	UCL	SkyLake 2.3 GHz Haswell 2.6 GHz	1872 (78 x 24) 112 (4 x 28)	95 GB 64 GB	Omnipath	BeeGFS 440 TB	None	2 days 6 hours	▦ MPI
Dragon1	UMons	SandyBridge 2.60 GHz	416 (26 x 16) 32 (2x16)	128 GB	GbE	RAID0 1.1 TB	4x Tesla C2075, 4x Tesla Kepler K20m	41 days	⋮ serial / ≡ SMP

http://www.ceci-hpc.be/

References

- https://en.wikipedia.org/wiki/Computer_architecture
- https://en.wikipedia.org/wiki/Computer_hardware
- https://en.wikipedia.org/wiki/Graphics_processing_unit
- https://en.wikipedia.org/wiki/Central_processing_unit
- https://en.wikipedia.org/wiki/Bus_(computing)
- https://en.wikipedia.org/wiki/BIOS
- https://en.wikipedia.org/wiki/Multi-core_processor
- https://en.wikipedia.org/wiki/Computer_cluster
- http://www.ceci-hpc.be/
- "Eléments d'informatique médicale", RADI2008-1, Sébastien Jodogne

Program

▶ Historical perspective

▶ Computer structure

▶ **Operating systems**

▶ Programming languages
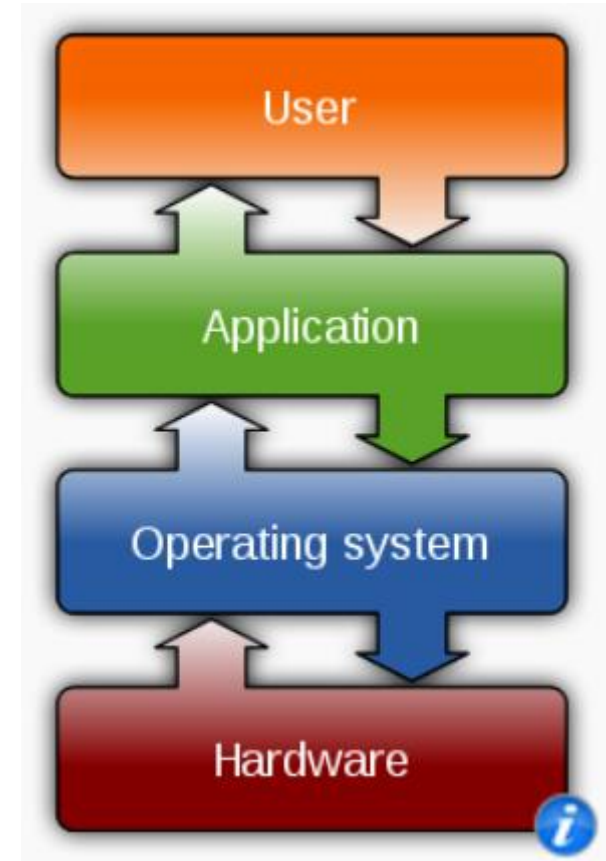
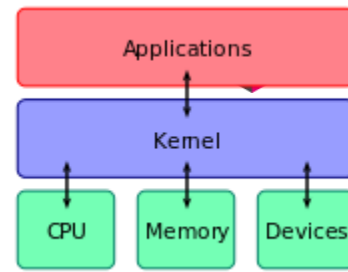# OS definition

An "**operating system**" (OS)
= first program launched at start up!
= system fundamental software to

▶ manage computer hardware and software resources,

▶ protects system's integrity against incorrect apps, and

▶ provide common services for computer programs:

OS tasks

▶ **Process management**.
allocate resources to processes, enable processes to share and exchange information, protect the resources of each process from other processes and enable synchronization among processes

▶ **Memory management**.
management of computer memory resource

▶ **File system**.
controls how data is stored and retrieved

▶ **Device drivers.**
operates or controls a particular type of device attached to a computer

▶ **Networking.**
allows "nodes" to share resources

▶ **Interrupts**.
signal to the processor emitted by hardware or software indicating an event that needs immediate attention

▶ **Security.**
protection of computer systems from theft or damage to their hardware, software or electronic data, as well as from disruption or misdirection of the services they provide

▶ **I/O.**
communication between an information processing system, such as a computer, and the outside world, possibly a human or another information processing system

Current main players

▶ Windows

▶ Mac OS

▶ Linux

▶ (Unix)

▶ (Android & iOS on smartphones/tablets)
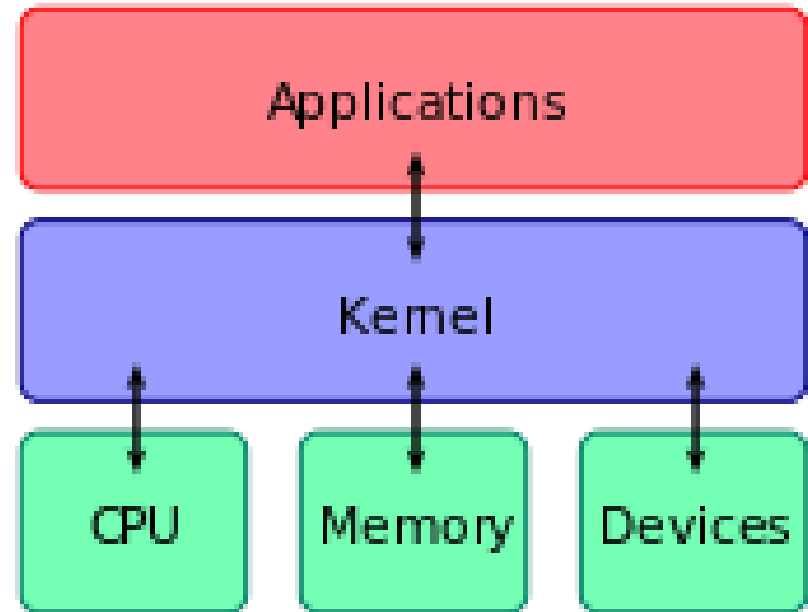
Linux

▶ **Open-source OS, since 1991**
- → free to use, copy, modify but not to sell
- → multiple distributions and flavours
- → supported by a large community

▶ **On PC:**
- now (almost) as easy to use as a Win/Mac with simple GUI
- typically runs open-source software: Open Office, Gimp,…
- usually more secured than Windows/Mac

▶ **On servers & clusters**
- Standard OS → need to know command line

Mac OS   vs   Windows

- 1st version in 1984

- Since 2001, based on a Unix kernel

- Proprietary to Apple, i.e. closed & €€€

- With GUI *and* command line

- 2nd most common OS on PC's.

- Usually more secure than Windows OS but limited to proprietary (and €€€) hardware…

- 1st version in 1985

- Originally graphical operating system shell for MS-DOS. Now most common OS on PC's

- Proprietary to Microsoft, i.e. closed & €€€

- Other MS software, like Office, are €€€

- More exposed to security issues but runs one all sorts of hardware built

All OS's in one computer

Use "virtual machines" (VM) to execute

▶ an entire OS, with applications, on a virtual hardware ("**system VM**")

▶ a program in a platform-independent environment ("**process VM**")

on the same physical machine, i.e. original hardware and OS.

→ easy to create, copy, kill, relaunch, distribute,…

For example:

▶ System VM with 'VirtualBox'

▶ Process VM with 'Docker' or 'Singularity', aka "container"

References

- https://en.wikipedia.org/wiki/Operating_system
- https://en.wikipedia.org/wiki/Linux
- https://en.wikipedia.org/wiki/MacOS
- https://en.wikipedia.org/wiki/Microsoft_Windows
- https://en.wikipedia.org/wiki/Virtual_machine
- https://www.virtualbox.org/
- https://www.docker.com/
- https://sylabs.io/docs/

Program

▶ Historical perspective

▶ Computer structure

▶ Operating systems

▶ **Programming languages**
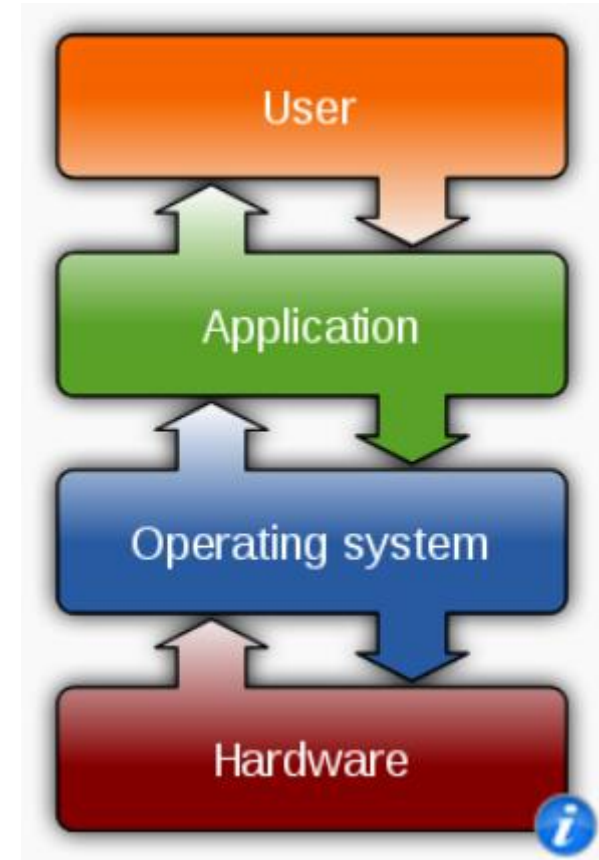
Some wisdom...

"Programming: when the ideas turn into the real things."

\- Maciej Kaczmarek

"The most important single aspect of software development is to be clear about what you are trying to build."

\- Bjarne Stroustrup

Classic ones for scientific computing

▶ ...

▶ Fortran

▶ C/C++

▶ Java

▶ Python

▶ R

▶ Perl

▶ Matlab/Octave

▶ Julia

▶ ...

NOTE: There exist 1000' of them!!!

Low vs. high level of programing

▶ closer to hardware vs. relying on intermediate software layer

▶ more complicated vs. easier code writing

▶ tedious vs. more abstract

▶ faster and efficient vs. usually a bit less so

A "compiler" or "interpreter" translates code to 'machine code' to create an executable program or execute it.

High level program example

The perfect Vanilla Cake:

1. *Preheat oven to 350º F.*
2. *Prepare three 8-inch cake pans by spraying with baking spray or buttering and lightly flouring.*
3. *Combine flour, baking powder, baking soda, and salt in a large bowl. Whisk through to combine. Set aside.*
4. *Cream butter until fluffy and then add sugar. Cream together for about 8 more minutes.*
5. *Add eggs, one at a time, and mix just until combined.*
6. *Add flour mixture and buttermilk, alternately, beginning and ending with flour.*
7. *Add vanilla and mix until thoroughly combined.*
8. *Divide among pans and bake for 25-30 minutes until edges turn loose from pan and toothpick inserted into middle of cake comes out clean.*
9. *Remove from the oven and allow to cool for about 10 minutes.*
10. *Turn out onto wire cooling racks and allow to cool completely.*

Compiled vs. Interpreted language

▶ Compiler → generate machine code from source code

- typically lower-level language

- no cross-platform support for exec code (i.e. need to recompile on specific OS)

▶ Interpreter → step-by-step executors of source code

- typically higher-level language

- easy (at least should be easier) cross-platform

▶ Both available for most *high-level* language

▶ Can be a mix of both

Wrapping vs. number crunching

▶ Wrapping → relies on other bits of code

- typically higher-level language

- glues and pipelines different operations

▶ Number crunching → does the job on some data

- typically lower-level language

- takes in data and parameters to calculate an output

▶ Both available for most *high-level* language

▶ Can be a mix of both

FORTRAN

▶ appeared in 1957, developed by IBM

▶ latest release "Fortran 2018" in November 2018

▶ designed for computationally intensive scientific and engineering applications

▶ low level language (but dynamic memory allocation)
$\rightarrow$ very efficient when compiled

▶ portable on any hardware and OS

▶ popular language for "high-performance computing" and is used for programs that benchmark and rank the world's fastest supercomputers

C and C++

C

▶ started in 1973, ISO standardized in 1989, latest release in June 2018

▶ low level language (e.g. memory management)
$\rightarrow$ very efficient when compiled

▶ portable on any hardware and OS

C++

▶ based on C with added object-oriented & other programing features

▶ started in 1979, ISO standardized since 1998, latest release in December 2020

Java

▶ appeared in May 1995, latest release in September 2021

▶ developed by Oracle Corp

▶ high-level general-purpose language (class-based and object-oriented)

▶ ideally "write once, run anywhere" (WORA)

▶ applications are typically compiled to bytecode that can run on any *"Java Virtual Machine"* (JVM) regardless of computer architecture

▶ open-source compiler (GNU GPL)

▶ fairly stable

Python

▶ started in 1991, latest version (3.10.2) out in January 2022.

▶ interpreted high-level programming language for general-purpose programming

▶ dynamic type system, object-oriented and automatic memory management.

▶ relies on large and comprehensive standard library.

▶ interpreters available for many OS's.

▶ open source with community-based development model

▶ still evolving: main version
(https://fr.wikipedia.org/wiki/Python_(langage)#Historique_des_versions) and libraries

R

▶ started in 1992, stable since 2000, latest release in November 2021

▶ open source with community-based development model

▶ high-level interpreted language, free software environment (GNU GPL)

▶ mostly used among statisticians and data miners for developing statistical software and data analysis.

▶ pre-compiled binary versions available for most OS's.

▶ command line interface, plus graphical front-ends and IDE's ("Integrated Development Environments).

Perl

▶ started in 1987, v5.34 released in May 2021

▶ originally general-purpose Unix scripting language to make report processing easier

▶ Built in C, free software environment (GNU GPL)

▶ very good at text processing without the arbitrary data-length limits

▶ high-level, general-purpose, interpreted, dynamic programming languages

▶ nicknamed the "duct tape": glue language and perceived inelegance

Matlab/Octave

▶ started in 1984 by MathWorks, based on C and Lapack libraries (in Fortran...)

▶ multi-paradigm numerical computing environment, good at matrix manipulations, implementation of algorithms

▶ can interface with programs written in C, C++, Java, and Python.

▶ large number of users-contributed (open source) packages

▶ but proprietary programming language → €€€ license

▶ yearly releases with new features but fairly stable (back compatibility!)

▶ Octave = free alternative to Matlab but not 100% compatible or as efficient

Julia

▶ started in 2012, v1.7.2 released in February 2022

▶ free open-source language, runs on most OS's

▶ high-level general-purpose dynamic programming language

▶ originally designed for high-performance numerical analysis and computational science

▶ allows concurrent, parallel and distributed computing, and direct calling of C and Fortran libraries

▶ includes efficient libraries for floating-point calculations, linear algebra, random number generation, and regular expression matching.

▶ other libraries are available from the community

Some wisdom…

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

- Dennis Ritchie

Still

▶ some algorithm/coding principles remain the same across languages

▶ pick the language of your community/appropriate for your data

▶ do not reinvent the wheel

Some further wisdom...

*"The good news about computers is that
they do what you tell them to do.*

*The bad news is that
they do what you tell them to do."*

\- Ted Nelson

References

- https://addapinch.com/best-vanilla-cake-recipe/
- https://en.wikipedia.org/wiki/Software
- https://en.wikipedia.org/wiki/C_(programming_language)
- https://en.wikipedia.org/wiki/C%2B%2B
- https://en.wikipedia.org/wiki/Java_(programming_language)
- https://en.wikipedia.org/wiki/Python_(programming_language)
- https://en.wikipedia.org/wiki/R_(programming_language)
- https://en.wikipedia.org/wiki/Perl
- https://en.wikipedia.org/wiki/MATLAB
- https://en.wikipedia.org/wiki/GNU_Octave
- https://en.wikipedia.org/wiki/Julia_(programming_language)
- https://twitter.com/CodeWisdom

Thank you for your attention!