

Difference between mistral and a mobile phone

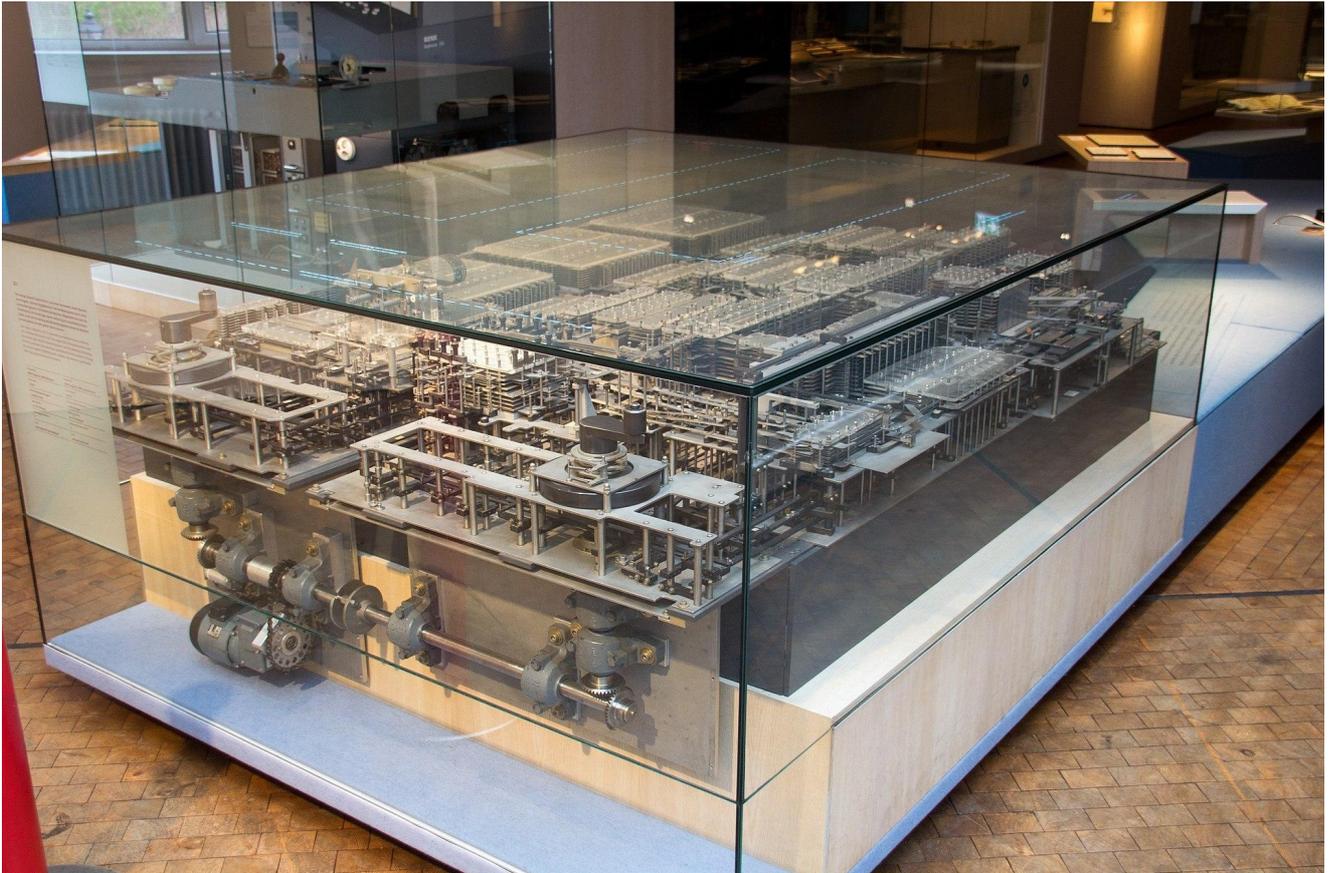
Because this is not the central part of the course and contains no executable code, it will be more a collection of useful information rather than a detailed description of the topic.

Hardware and Software developed side-by-side starting the first half of the 20th century mainly as industrial tools to push forward automation in production and data assesment and analysis but also from the military area.

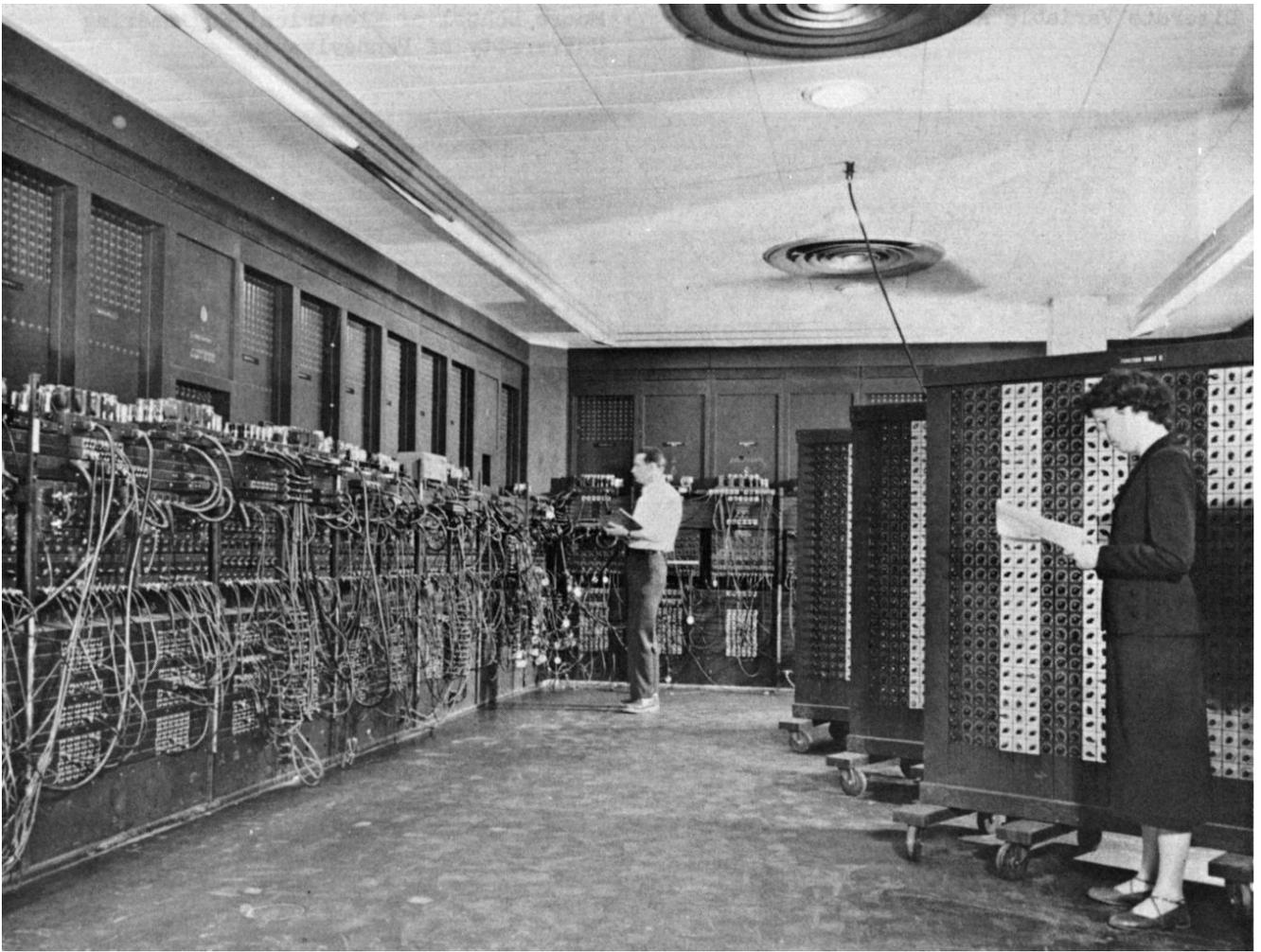
Hardware

Notable mentions:

- Z1: first programmable computer by Konrad Zuse around 1937, purely mechanical, privately financed



- ENIAC: first programmable electronic computer, 1945-1956, 18000 tubes, Army research Lab



Mechanical was not reliable, so the rapid miniaturization for electronic parts pushed the computer hardware from tubes to transistors to [microprocessors beginning in the 1960s](#)

If you are more interested in the history part, check [IBM](#). It shows the close relation between computing, industry, military and society over more than a century.

Basic parts of a computer system

Almost all computers follow the so-called [von Neumann architecture](#) from 1945:

- Input and Output
- Central processing for arithmetics/logics and memory controlling
- Memory

What a computer can do is basically

- write to memory
- read from memory
- compute with the memory

Different chipsets offer different operations with single instructions collected under the term [Assembler or Assembly language](#). Hardware architecture you might know:

- X86 32/64bit: Xbox, Playstation 4/5, Laptops, Desktops, Mistral
- Arm: smartfon, tablet, recent HPC-systems like Fugaku
- MIPS: 64bit, routers
- RISC: old SGI HPC systems
- PowerPC: based on RISC, IBM PC and HPC systems, Apple hardware before Intel-era, Playstation 3

Memory in action

input program

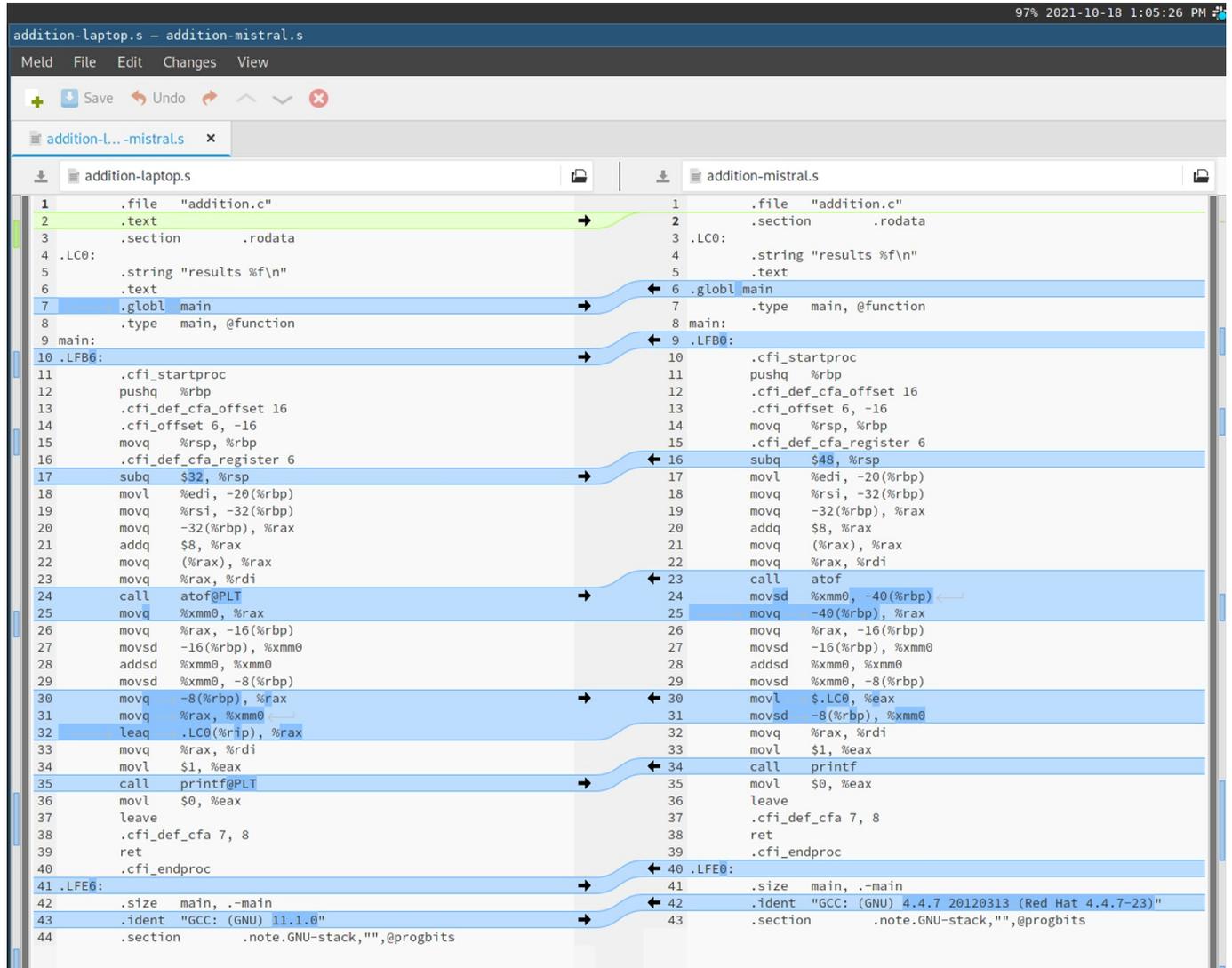
```

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

int main(int argc, char **argv) {
    double x = atof(argv[1]);
    double result = x+x;
    printf("results %f\n", result);
    return 0;
}

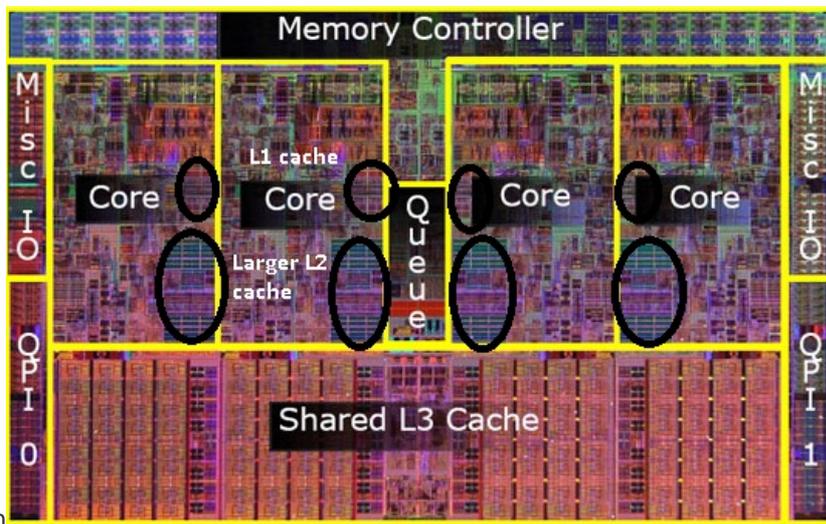
```

Compile it with `gcc -S addition.c` to assembly on two different systems (mistral + laptop) with GCC compiler leads to two slightly different files. See `addition-laptop.s` and `addition-mistral.s` in the repository.



Memory Hierarchy

type	access	size	cost
registers	5ns	1e2	part of CPU
caches (SRAM)	10ns	1e6	100.00 \$/MB
main memory (DRAM)	100ns	1e9	1.00 \$/MB
hard disk	5000ns	1e11	.05 \$/MB



Real Size: 1cmx1cm

Software

Together with programmable hardware the languages evolved. Hence Konrad Zuse was again the first: [Plankalkül](#) non-published book from 1945 because of WWII. Another old standing member in the family of programming languages is FORTRAN from 1950s ([Good overview of languages](#))

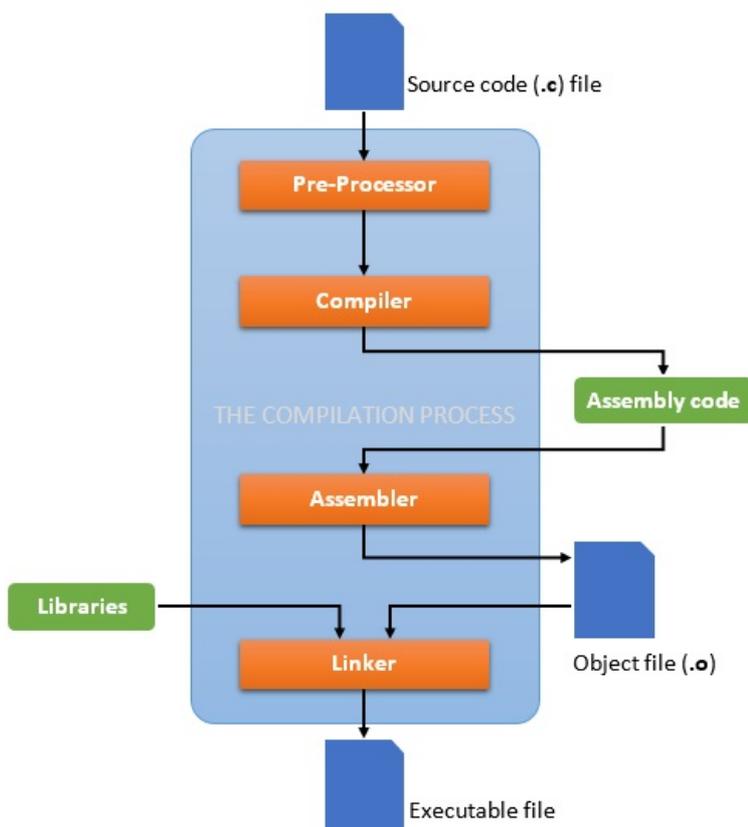
Basic goal:

- expression of mathematical formula (computation!)
- expression of algorithms -> solving problems!

How to put software into action

As mentioned above software exists in different abstraction levels but today mostly in the form a text file. In the early days [punch cards](#) were used instead because there were easier to read by machines ([Fortran example](#)). But let's stick to text files:

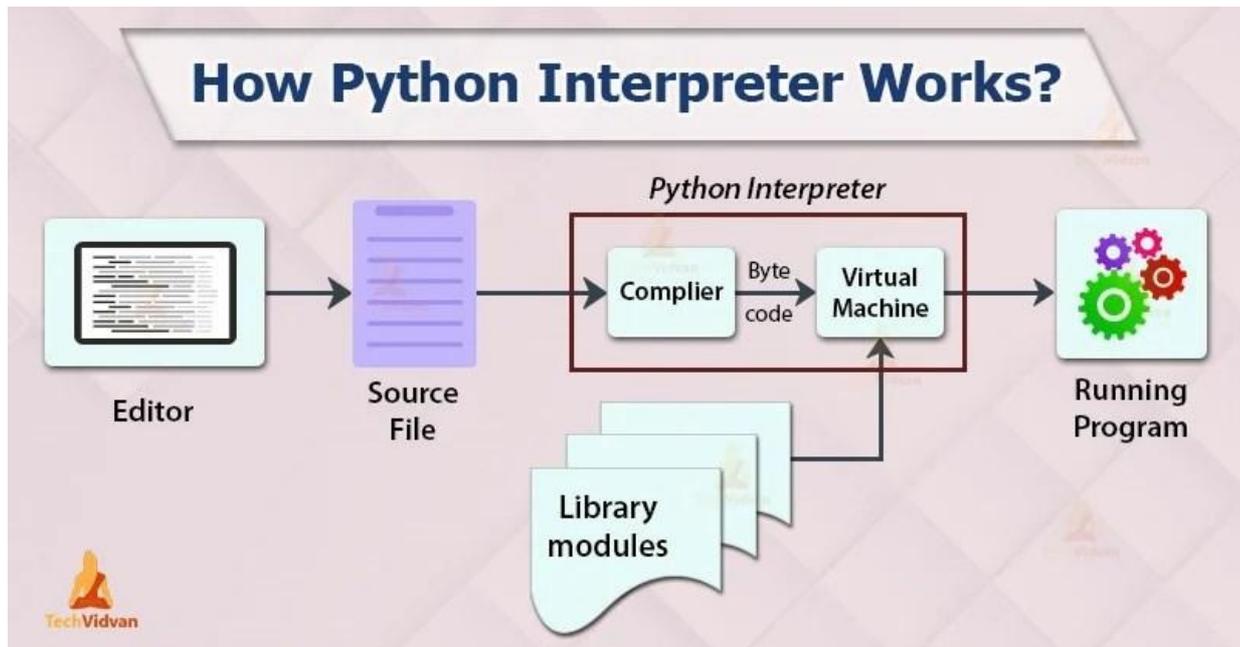
Basic compilation process



The compiler translates the source through several steps into a machine language and outputs an executable

program, which can then later be used as often as needed.

But Python is *NOT* a compiled language! True, the difference is not so huge. here is how it works with python:



In interpreter incorporates all necessary steps of the compiler and linker *AND* executes the program. Both methods, Compilation and Interpretation have pros and cons:

- compiled programmes need to be re-compiled every time the source code is changed
- compiled programmes contain machine code, only. This gives a big advantage in runtime performance
- interpreter don't bother the programmer with complex compilation processes and linking problems
- programs for an interpreted language are *source code*: very easy to debug and change
- the development cycle with an interpreter language is faster, because there is no extra compilation step. Of course there are techniques to combine both types of languages ;-)

Example from above in python (invented by [Guido von Rossum](#))

```
import sys
x = float(sys.argv[1])
result = x*2
print("result = %s",result)
```

Running (Compilation + Execution) with `python addition.py 3`

Further readings

[This collection of free online books](#) is a good start

Mistral configuration

<https://www.dkrz.de/up/systems/mistral/configuration>