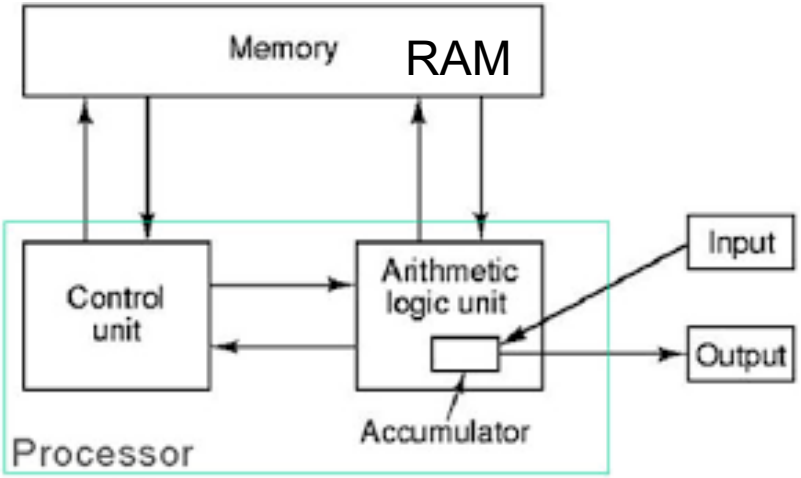
LECTURE 2

- History of computing: von Neumann architecture
- Developments that have led to:
  Fortran, Unix, C, and IBM PC & Apple hardware
- Linux OS
- Languages of HPC (High Performance Computing)
- Physics of integrated circuits and the transformation of technical civilization in the last 50 years

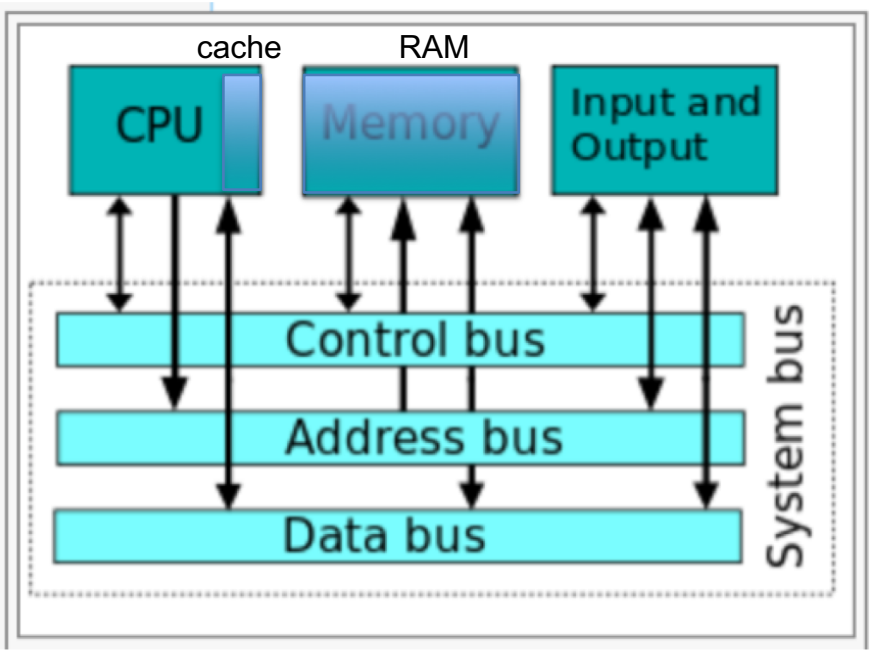
Literature

1. Haigh & Ceruzzi "A new history of modern computing (2021)
2. Joshua Izaac & Jingbo Wang, "Computational Quant. Mechanics" (Undergrad. Lecture Notes in Physics, Springer, 2018) – chapters on Python and Fortran
3. A whole page of literature suggestions for this course
http://planets.utsc.utoronto.ca/~pawel/PHYD57/xx (xx = secret sauce)

Von Neumann computer implements the logic of *Turing machine* – a model, where data and program are stored in the same memory. Von Neumann architecture repeatedly performs fetch-decode-execute-store operations.



CPU= Central processor unit

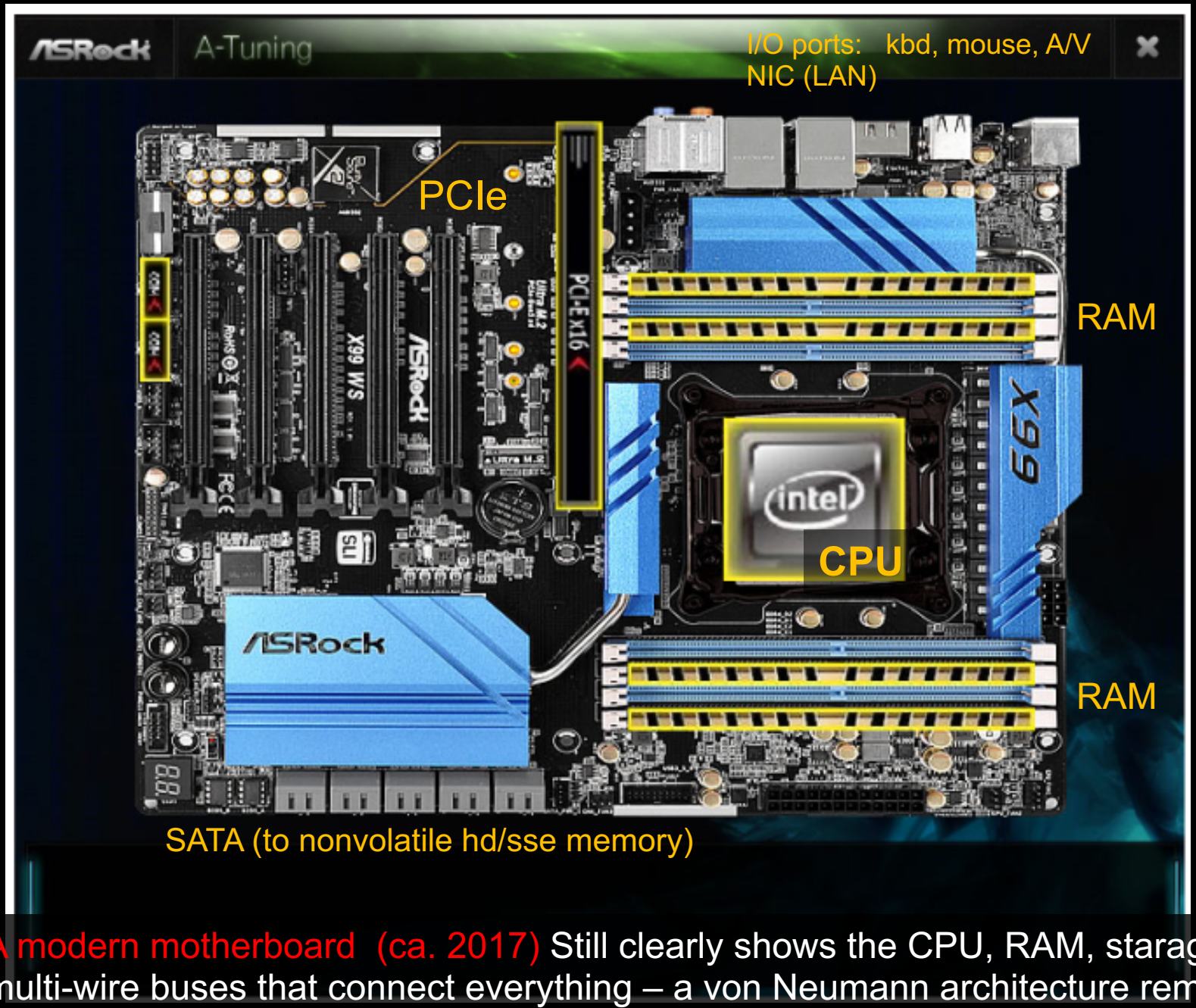


A modern computer architecture:

RAM = Random Access Memory (volatile: transistors need some power to keep states 0 & 1)

Storage = Nonvolatile Memory
(hard disk, solid state memory, replacing earlier: magnetic drum or tape, punched paper)

Several buses are etched on motherboards, fastest ones for CPU-RAM communication. Also important: PCIe = data bus for peripheral devices such as: GPU, MIC, NIC (ethernet), SATA for storage, etc.

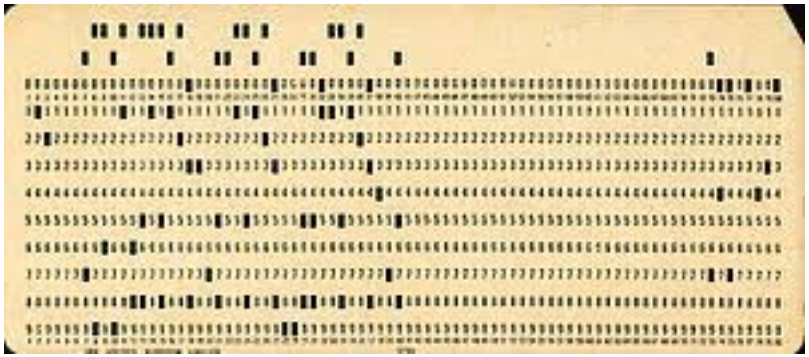


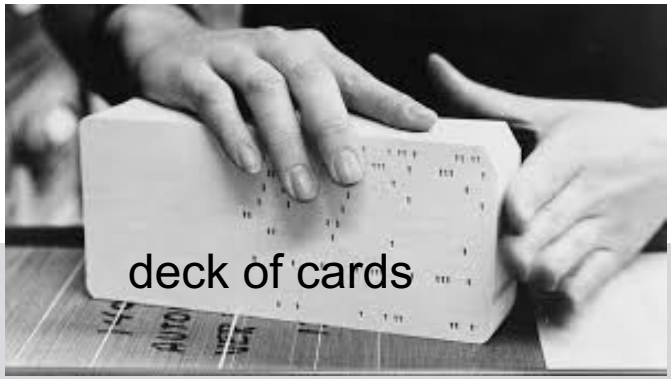
A modern motherboard (ca. 2017) Still clearly shows the CPU, RAM, starage, and multi-wire buses that connect everything – a von Neumann architecture remained essentially unchanged for 70+ years! Is this good? (cf. Backus 1978)

Mid-1960s and 1970s

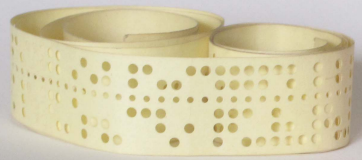
- Data processing needs large permanent storage

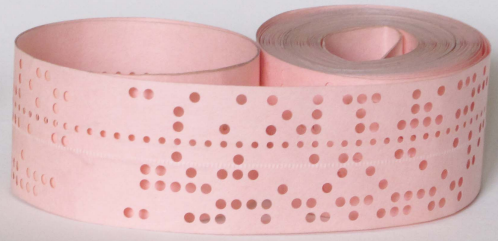
IBM (International Business Machines Co.) grew when it transitioned from decks of punched cards (very popular among businesses)





... and perforated tape storing programs





to magnetic media: tape, drums, disks

Q: How many times more data was stored on a 2013 hard-disk than on 1956 version?

1950s to 1960s

At the end of 1950s and beginning of 1960s many advances gradually appeared:

- Magnetic tape as mass storage of von Neumann architecture:
- Separate operating mem and mass storage, processor, data and program in the same memory
- Magnetic drums playing the role of hard disks, developed later by IBM
- Binary digits have won with decimal digits
- From among many different data formats, 8-bit bytes became de facto standard
- They were handy for character (text) processing, as 1 byte holds 1 ASCII character
- Instead of setting up the computer by hand or from punched paper media, the programming was now done from magnetic media, and the code was no longer a low-level machine assembly code (list of fairly elementary commands understood by the processor (such as, for instance: shift value from register A to B, add register B to register C, store result in register C, write register C to RAM location whose address is stored in register D.)
- instead, *compilers* of *high-level programming languages* appeared

- *compiler* = program that checks and translated your program into the set of low-level instructions for processor (still readable by humans, but program in such assembly language was ~20 times longer than the high-level code in language similar to English+math symbols). Finally the so-called linker (part of compiler in modern times) was translating the assembly language to binary executable containing 00011100110101001000101010 11...

mark_description "Intel(R) Fortran Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 15.0.3.187 Build 2”; .file "fortran+om-laplace-sp.f90”

(Dis)assembly code – CPU’s can only understand such language

(...) Much too low-level; we will not code in assembler!

Instructions to shift bytes (data, addresses) into and out of registers, do conditional jumps, write to RAM, do logical operations such as XOR, and more. Many lines of assembly program are needed to do each instruction in hi-level language.

```
 (...)
movsd     %xmm1, data_mp_t1_(%rip)                        #113.5
        movl      1950048+data_mp_grid_(%rip), %r12d      #114.2
        jne       ..B1.18       # Prob 50%                #115.10
                                # LOE rbx r14 r15 r12d r13d xmm1
..B1.13:                        # Preds ..B1.12
        movl      $-1, %esi                               #115.14
        lea       32(%rsp), %rdi                          #115.14
        movq      $0x1208384ff00, %rdx                    #115.14
        movl      $__STRLITPACK_9.0.3, %ecx               #115.14
        xorl      %eax, %eax                              #115.14
        lea       232(%rsp), %r8                          #115.14
        movq      $15, 232(%rsp)                          #115.14
        movq      $__STRLITPACK_8, 240(%rsp)              #115.14
        movsd     %xmm1, 288(%rsp)                        #115.14
call      for_write_seq_lis                               #115.14
                                # LOE rbx r14 r15 r12d r13d
        movsd     288(%rsp), %xmm1
```

1950s to 1960s

- Hardware and software, previously inextricably bound, now became independent:

- The first widely use computer language which was hardware platform-independent was **FORTRAN** (1957) = Formula Translator, created by John Werner Backus, who worked at IBM on the IBM 704.
- Read: J. Backus "*History of Fortran I, II, and III" (*1978) on our /xx page.

- This language was loved by academia and the first computer science departments (CS = computer science, just trying to emerge and define itself in opposition to EE). Fortran resembled ***meta-code*** written in plain English language.

- It produced executable programs running as fast as a hand-coded assembly code. Nobody believed at first that automatic compilation project will do that! Many businesses were ask about the required features, but the only company that took it seriously was United Aircraft Corp., which delegated an engineer to work full-time with the Fortran team at IBM (cf. Backus 1978).

- Then **COBOL** was created by the U.S. government committee & mandated on all machines it was purchasing. (COBOL = Common Business Oriented Language.) This forced a compatibility/uniformity across different machines. It was good for databases, payrolls, and some airline reservation systems

COBOL was also supposed to encourage long variable names for better clarity of programs (to supplement/replace comments?) but it failed at that, which was realized when the software engineers looked at some ancient codes, fearing the so-called Year 2000 Problem (Y2K). Very few COBOL programs are in use. COBOL was not a general programming language, and was never popular in academia.
However, if you know it, today the US government and some banks will pay you top salary figures for maintaining some ancient legacy codes that they have trouble modifying.

While COBOL is now as good as dead, Fortran (Fortran 90, 95, 2003, 2015 versions) thrives in academia, though is now almost unknown in businesses and in Computer Science departments, which have switched to teaching programming in **C, C++, Java** and now mostly in **Python** (to which we return below).

In 70s and 80s, CS gurus worked hard at giving Fortran a bad name (based on supposed harmfulness of 'goto' instruction, cf. Dijkstra 1968), while promoting new languages, trying to make programming easier, more reliable, and easier to prove correct. CS was quickly creating them and then dropping. Even today, scores of programming languages are created every year. They number in thousands, some useful for certain purposes and some just amusing.
The programming paradigms such as *object-oriented* and more recent *functional programming* do not aim at producing fast code, as a priority. Therefore, most such languages are not good at number crunching (C++ can do it with some effort despite, not thanks to object-oriented slant. The functional programming promotes the exclusive use of immutable objects, a no-no in efficient numerical science.)

Trying to make a long story [of languages] short

- The was ALGOL, an extra clean definition of a language, but mainly for writing metacode in papers.
- There was Pascal, and for a while Turbo-Pascal had popularity and performance
-
- **Python** is a scripting language, *an interpreter*, and on its own will always produce a relatively slow-running program (slow runtime execution).

- Writing and debugging a program in Python is often simpler than in the other, more powerful, languages. One reason is the huge user base, allowing inexperienced programmers to understand & fix errors by browsing for answers on internet.
- So many snippets of code exist on the web that LLMs (Large Language Models like ChatGPT) are getting proficient in cobbling together Python scripts for often appearing tasks.
- Pre-packaged modules help Python tap into the strength of more powerful languages. Guess in what language the crucial modules *Matplotlib* and *Numpy* of now-dominant Python were originally written? Fortran, the language explicitly created to perform numerical calculations.

- The only language other than Fortran, starting from the same goal of top-speed, large-scale scientific computation (including parallelism), is **Julia** created at MIT. However it is not widespread yet, or demonstrably superior to Fortran or C(++).

Mid-1960s to 1970s hardware

IBM captured 70% of global computer sales, mostly with very expensive ***mainframes*** (or supercomputers, bought by big firms, government centers and the universities). The rest of the market was served by Univac, Computer Data Corp., Sci. Data Systems, and Cray, which tried to emulate IBMs scale, salesforce and success.

DEC = Digital Equipment Corporation appeared as a small company in 1960s, located in old Abbotts Mills near Boston, benefitting from MIT connections. DEC, later called **Digital**, started a quiet revolution in the market by offering:

- much smaller ***minicomputers***, costing much much less than mainframes. For instance, model PDP-8 costed $18k instead of IBM/360 for $2 million. It was interactive.
- similar speed on some tasks, while much less data storage capability [All circuits in 1960s were non-integrated but based on fast-switching transistors.]
- innovation in comp. architecture: *the interrupts* in CPU operation to serve input/output requests from the peripheral devices (printers, storage, keyboards, monitors, NICs)
- different & much less pretentious culture in workplace & sales

OEMs (Orig. Equip. Manufacturers) appeared when Digital opened up their architecture, encouraging modifications & add-ons

1970s and later...

- PDP-7 on which Unix was born, see below, 4k 18-bit op. memory
- **PDP-8** (left picture) still little RAM but fast and cheap

- **PDP-11:** 128k RAM
- 2 MB removable hard disk
- > 30 kFLOPS arithmetic speed
- interactive consoles, time-sharing operating system, as opposed to batch input on mainframes

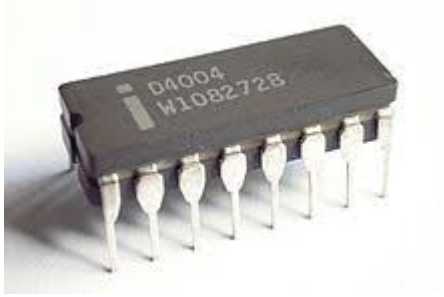
- Languages such as ALGOL, Pascal, and other became popular but later faded away. IBM's PL/I (Programming Lang. I) was a commercial failure.
- **DEC** offered the first **UNIX operating systems** on PDP-line computers
- **C** appeared in 1973 and later **C++**, high-level languages that unlike numerical computation-oriented Fortran for "number crunching", were not meant for scientific simulations. They were great for writing operating systems (fully C-based Unix in 1973, later **Linux** in 1990s, which is a modified Unix), and for interfacing at low level with hardware (low-level here means using elementary, simple instructions).

THE C PROGRAMMING LANGUAGE

Interactive use of Digital's PDP-11 minicomputers (here shown in year 1977) was a harbinger of and a model for personal computers invented right then and getting huge popular in 1980s.

Apple Computer Inc. was founded in 1976 by Steve Jobs and Steve Wozniak, with a vision to improve on that achievement & go much further toward

- miniaturization allowed by small ***microprocessors***, and
- personalization: personal computing, personal publishing and all other things personal & social on Apple, Mac, ibook, imac, macbook, itunes, iphone, ipad, ...



- **Programmable calculator and PC revolution** based on *semiconductor technology in microchips* or *microprocessors.*
- In 1971-1974, **Intel** (4004; 8008, 8-bit) and **Motorola** (8-bit 6800)
- Intel 8086 was a 16-bit CPU (central processing unit) from 1978,
  → "x86_16" (today: x86_64, i.e. 64-bit processors).
- Motorola 68000 was a 16/32-bit CPU from 1979
- In 1980s and 1990s computers spread outside academia, big business and military. Microcomputers or desktop Personal Computers (PCs) moved into every house and school, into the backpacks as laptops, and finally into our pockets as smartphones, not for scientific computation but rather for a simpler(?) task of communication (networking). This interesting history is outside the scope of our course, even though for your computing you will likely choose a laptop.

- Simple languages like BASIC, and in late 1970s/early 1980s the scripting languages MATLAB, and IDL have appeared. One of their descendent is now hugely popular:
- **Python** *interpreter* was created in 1989 and became popular in 1990s and 2000s.
- Creator, Guido van Rossum, was named the Benevolent Dictator for Life. Now the project is guided by Steering Council. Current version: Python3, appeared in 2008.

- In the 2004 a new and still not widely known but powerful and elegant language Julia was created by professors and students at MIT. Produces multithreaded code often executing almost as fast as C/Fortran. One day it may replace Python in science applications, though not necessarily in other worlds, such as business. Computer Sci. & Computational Sci. have *different* goals & favor different languages.

1980s and 1990s From Unix to Operating Systems created as community projects

AT&T Bell Labs in 1969-1971: Thomson and Ritchie use PDP-7 to create Unix OS.

In 1973 rewritten in new language C, and unveiled.

At UC Berkeley in late 70s a strain of Unix called BSD appears (Berkeley Software Distribution)

Commercial Unix:
Sun Micro:Solaris
Hewlett-P:HP-UX
SCO: Xenix
AT&T: System V
SGI: Irix

GNU Project starts in 1984, with a goal of a free version of Unix. Created many programs but failed to produce an OS ***kernel***

In 1991 in Finland, Linus Torvalds begins creation out of GNU, BSD and X-windows software of a Unix-like OS ***kernel.*** Calls it ***Linux***

1994-2007 **Linux** OS & flavors/distributions = packaged OS filled with slightly different components appear:
Red Hat, Fedora, FreeBSD, Caldera, Debian, SuSE, Mint, Gentoo, Ubuntu, CentOS, etc.
Also: MacOS, and Linux-kernel OS's with quite different GUI = graphical user interface: Android (2007), Chrome (2009)

We will return to the further historical development of hardware a bit later, after we explore the operating system types that conquered the world (not only of HPC, but also of web servers). Incidentally, learn some Internet history here.

Sobell – A practical guide to Linux Commands – 2018
Gedris – Intro to Linux Command Shell for Beginners – 2003

Good web resources:
www.linfo.org - on various topics about Linux, see all the links on that page, among others
www.linfo.org/command_index.html - list commands with further explanations on the use
www.linfo.org/how-to_index.html - how-to guides
Also, see the Links section on our course web page.

APPENDICES

A funny and informative history of a myriad of programming languages

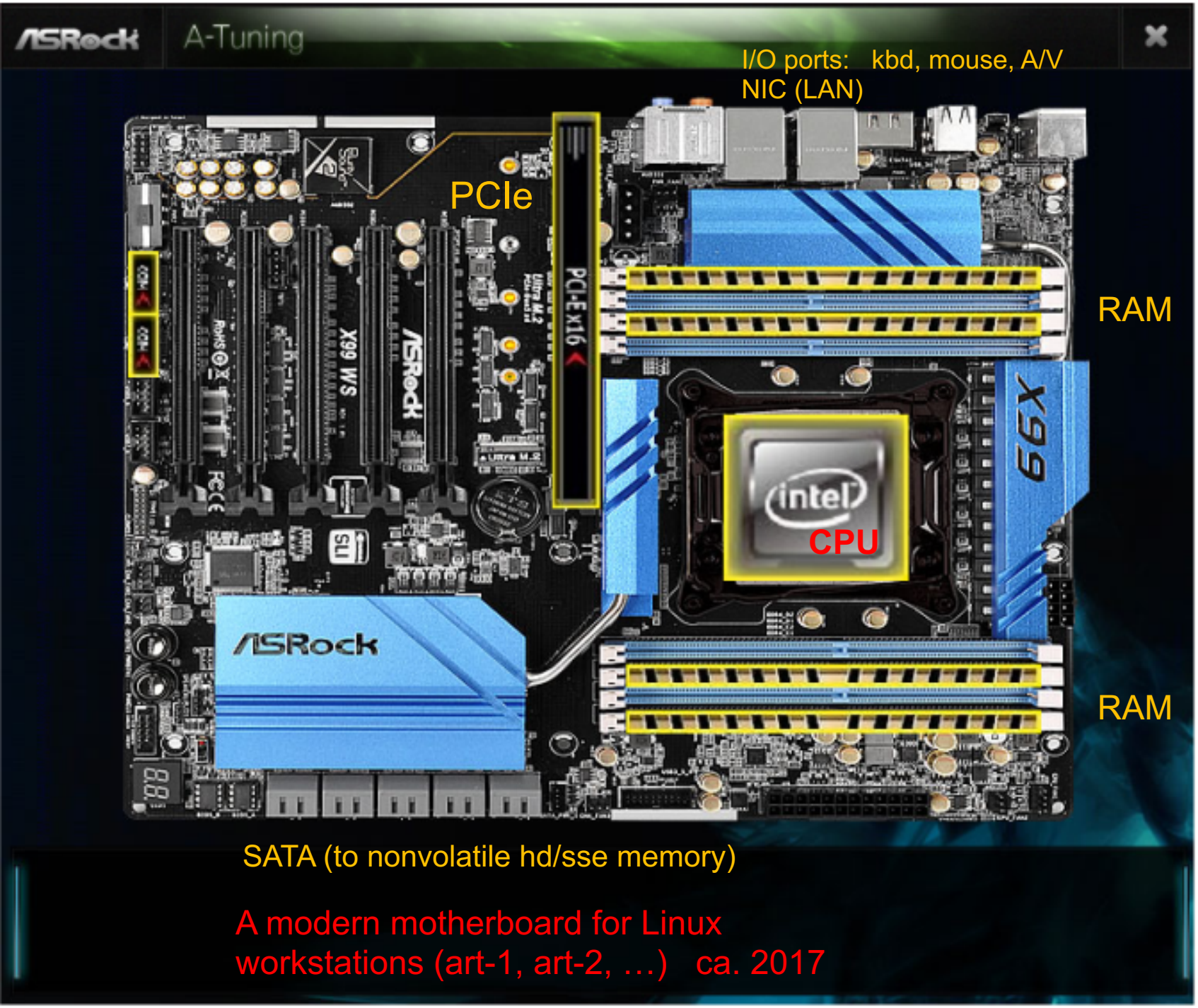
is on Youtube under the title

“**The Worst Programming Language Ever - Mark Rendle - NDC Oslo 2021**”

URL (Universal Resource Locator)

https://www.youtube.com/watch?v=vcFBwt1nu2U

It tries to gather all the WORST features of the discussed languages to create the title language. (No criticism of Fortran? I guess it does not have to many bad features.)


ASRock
A-Tuning
I/O ports: kbd, mouse, A/V
NIC (LAN)
PCIe
PCI-E x16
RAM
X99
intel
CPU
RAM
SATA (to nonvolatile hd/sse memory)
A modern motherboard for Linux
workstations (art-1, art-2, …) ca. 2017