howest
hogeschool

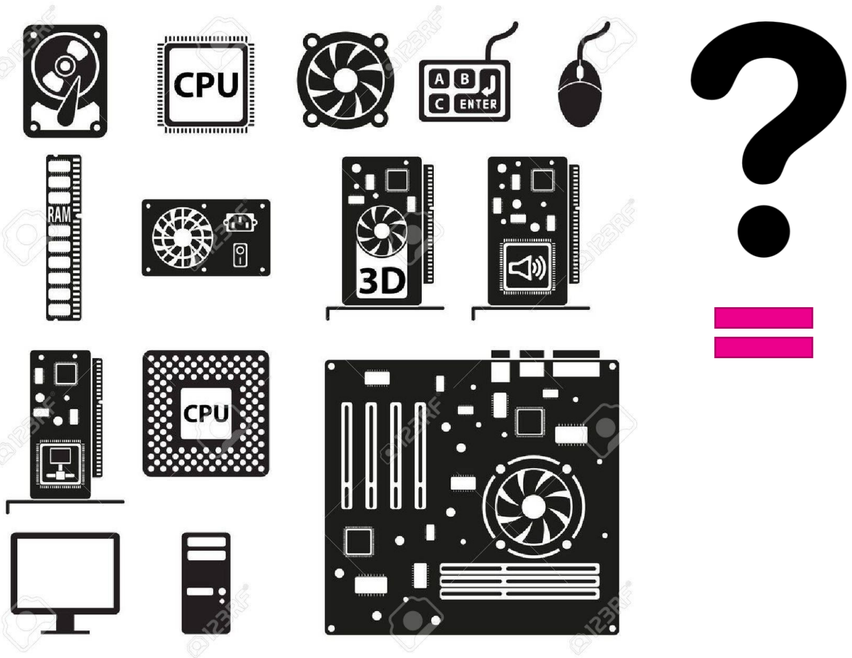
General Concepts: Introduction

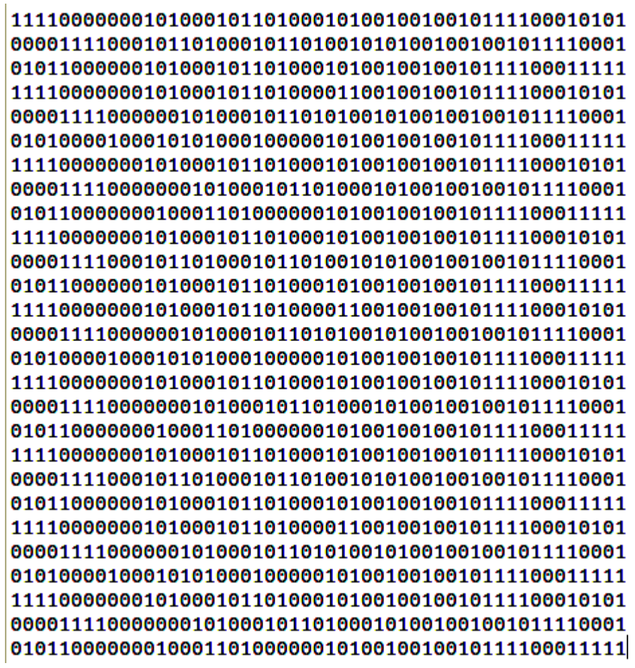
Chris Roets
Guy Van Eeckhout

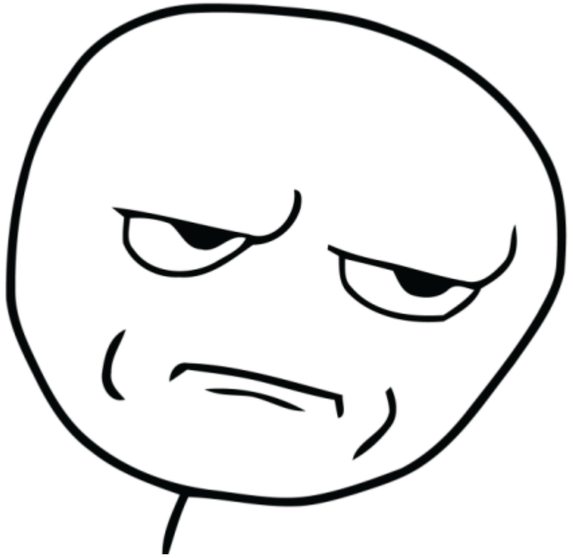
How do computers work?

“Mom, how do computers work?”

“With ones and zeroes, dear”





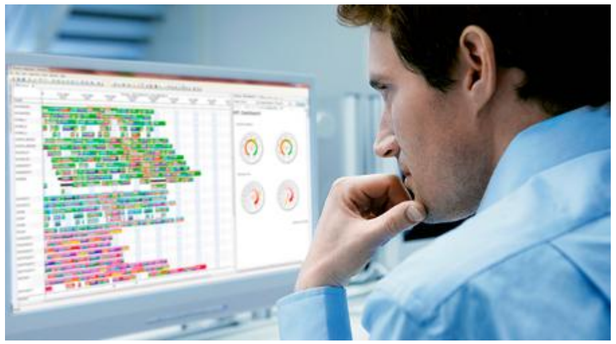


Comparing the computer hardware with a home

How do computers work?

“Dad, how do computers work?”

“Just like me, it accepts the **requests of different children**, and **giving them what they ask**, making his **own decision** of what is most urgent, **decided by mom**. Remember Dad **can handle only one task at once** and **mom comes first**”



Comparing the computer software with a home

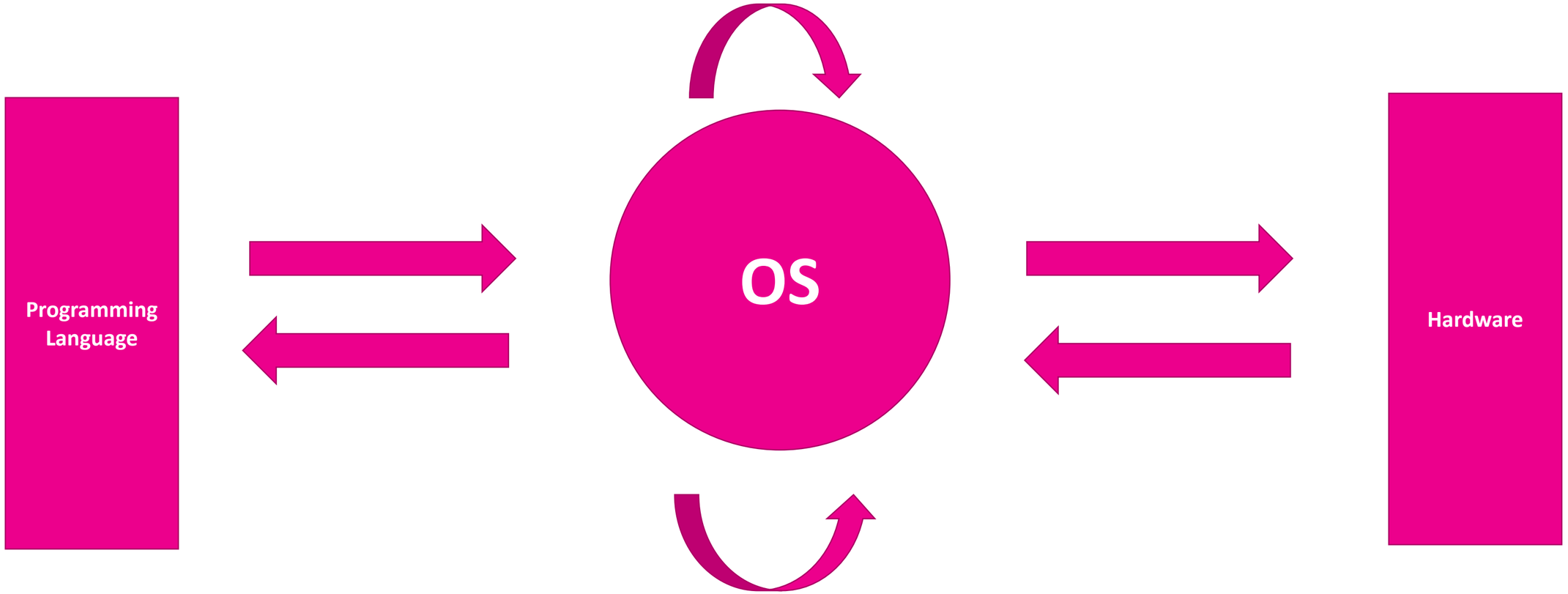
Why learn about operating systems?

** These examples are not necessarily built with these tools! It is just an illustration*

Programming languages

howest hogeschool

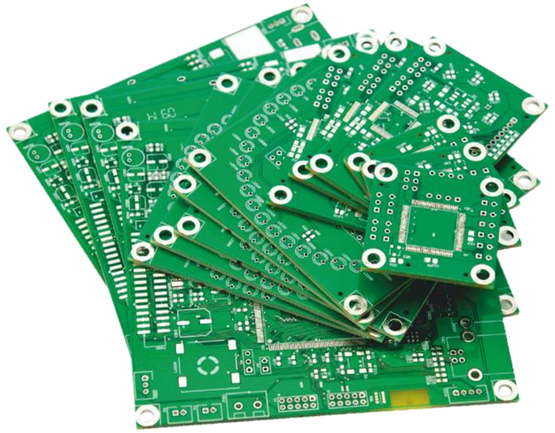
Why learn about operating systems?

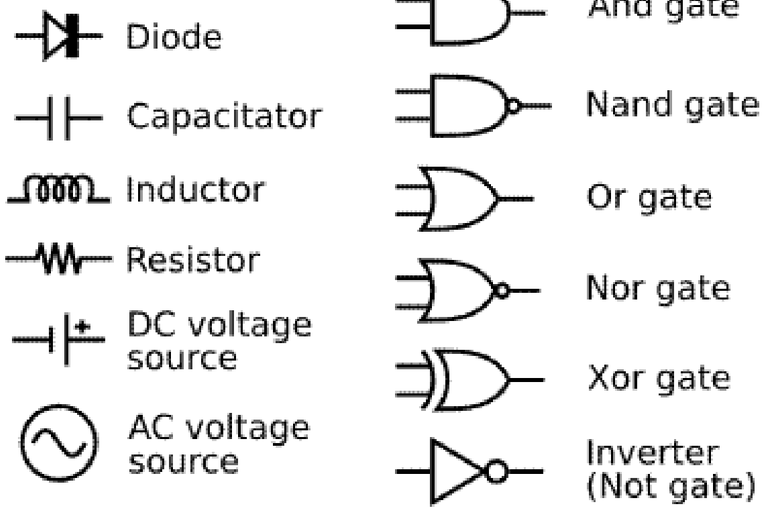


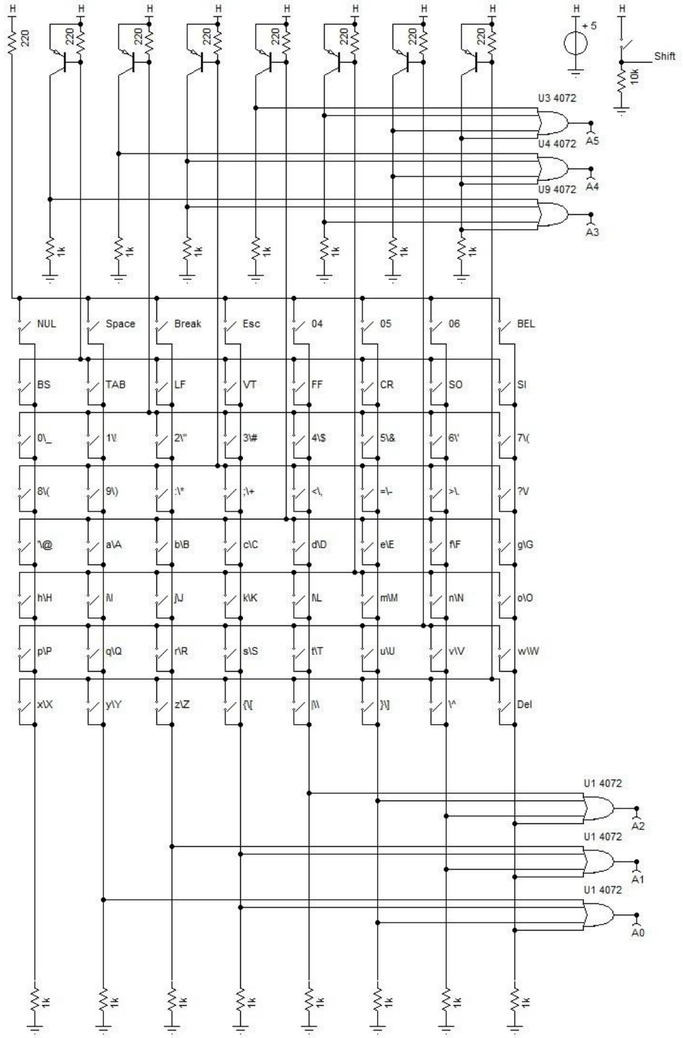
Why learn about operating systems?

Printed Circuit Board

(PCB – hardware)







howest hogeschool

Why learn about operating systems?

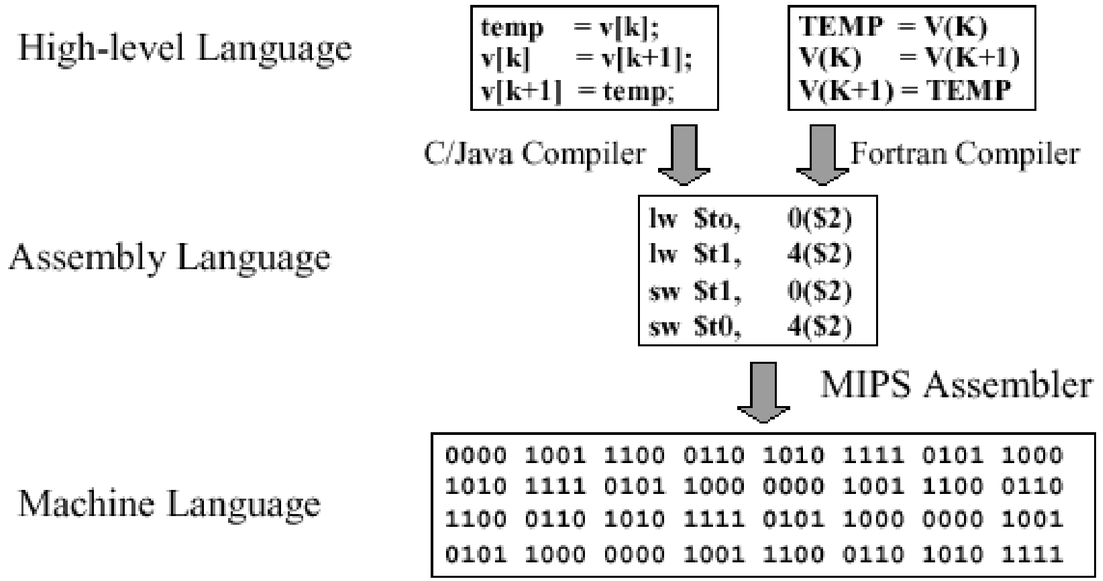
Programming languages

- Java
- Python
- PHP
- C & C++
- …

Differences

- **Object-oriented?**
- **Compiler or Interpreter**
- **High- vs low-level languages**

Assembly & Machine code



Compiler vs. Interpreter

Compiler

- Takes the “entire program” and translates it as a whole into machine code
- Takes more time to analyze the code and compile but the overall execution time afterwards is faster
- Linking (requires more memory)
- Examples: C & C++

Interpreter

- Translates program one statement at a time
- It takes less time to analyze the source code but overall execution time is slower
- No object code is generated, therefore memory efficient
- Examples: Python & Ruby

What about Java?

Why learn about operating systems?

Disassembly | Main.cpp

Address: add(int, int)

Viewing Options

```
int add( int a, int b )
{
push        ebp
mov         ebp,esp
sub         esp,0CCh
push        ebx
push        esi
push        edi
lea         edi,[ebp-0CCh]
mov         ecx,33h
mov         eax,0CCCCCCCCh
rep stos    dword ptr es:[edi]
   int c = a + b;
mov         eax,dword ptr [a]
add         eax,dword ptr [b]
mov         dword ptr [c],eax
   return c;
mov         eax,dword ptr [c]
}
pop         edi
pop         esi
pop         ebx
mov         esp,ebp
pop         ebp
ret
```

Assembly vs. machine code

Machine code bytes	Assembly language statements
	`foo:`
`B8 22 11 00 FF`	`movl $0xFF001122, %eax`
`01 CA`	`addl %ecx, %edx`
`31 F6`	`xorl %esi, %esi`
`53`	`pushl %ebx`
`8B 5C 24 04`	`movl 4(%esp), %ebx`
`8D 34 48`	`leal (%eax,%ecx,2), %esi`
`39 C3`	`cmpl %eax, %ebx`
`72 EB`	`jnae foo`
`C3`	`retl`

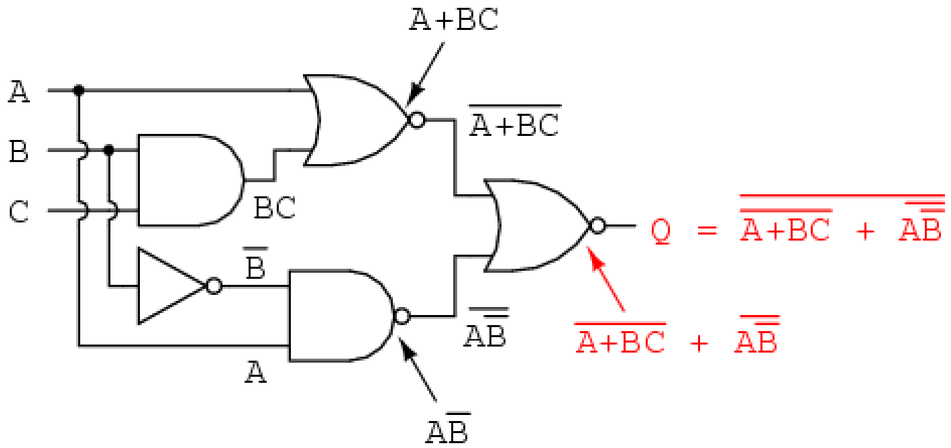
Instruction stream

```
B8 22 11 00 FF 01 CA 31 F6 53 8B 5C 24
04 8D 34 48 39 C3 72 EB C3
```

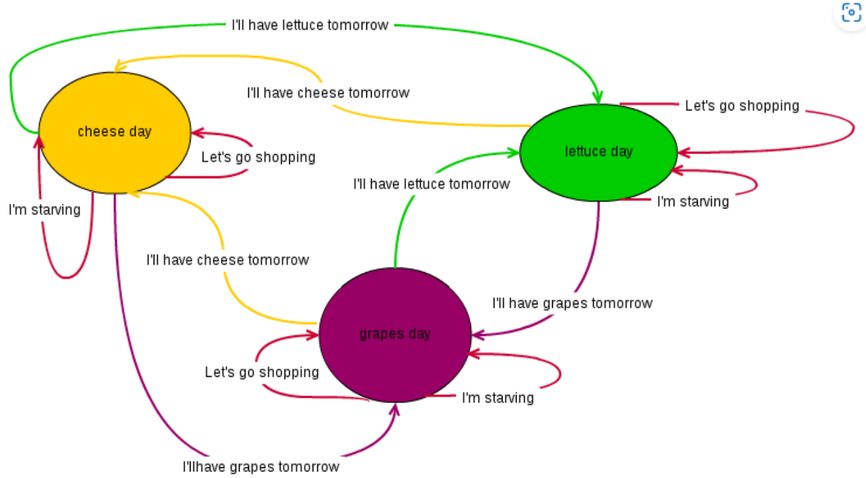
howest hogeschool

Why learn about operating systems?

Remember this?



Consider the following state diagram that describes the life of a strange creature that feeds on cheese, grapes and lettuce but never eats the same kind of food on two consecutive days.



Characteristics:

- Gates (and, or, not)
- Input (1's & 0's)
- Output (1 or 0)

Electronics is **math**!

gate	p1	p2	result
or	0	1	1
and	0	1	0
N	0		1
Nor	0	1	0
Nand	0	1	1
Xor	0	1	1

Why learn about operating systems?

- “So, who is right”

Example: Kingdom Hearts 3 (KH3)

- KH3:
  - Video **game** for **PS4** & **Xbox One**
  - Created by Square Enix with the **Unreal Engine 4**
  - **Unreal Engine 4** is a game engine created by Epic Games in **C++**
    - They made the source code available on GitHub (after signup)
  - **C++** is a programming language and an extension of the **C** language

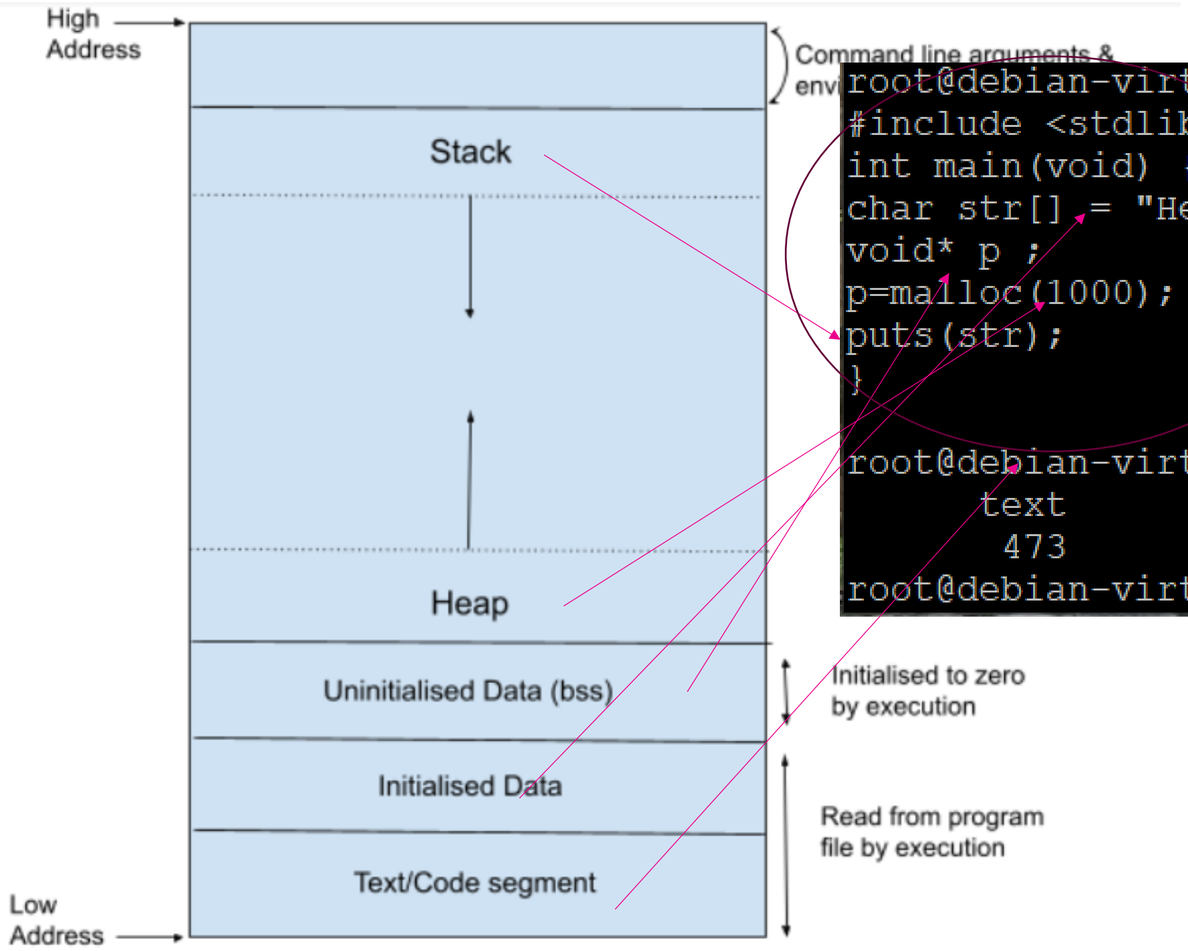
- Try this exercise yourself!

howest
hogeschool

From simple program to containers
A Deep technical dive

Starting from single program
To Operating System Multitasking
Further to Virtual Machines
Up to Containers

A process address space

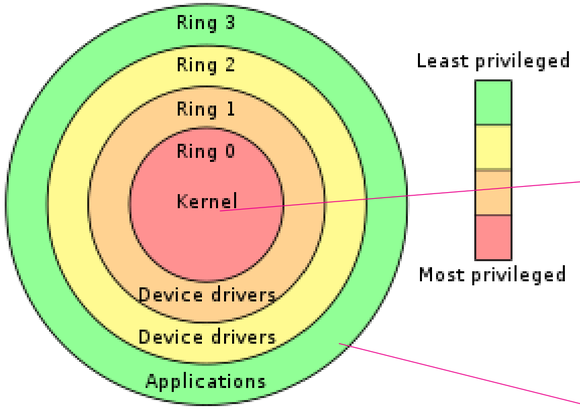


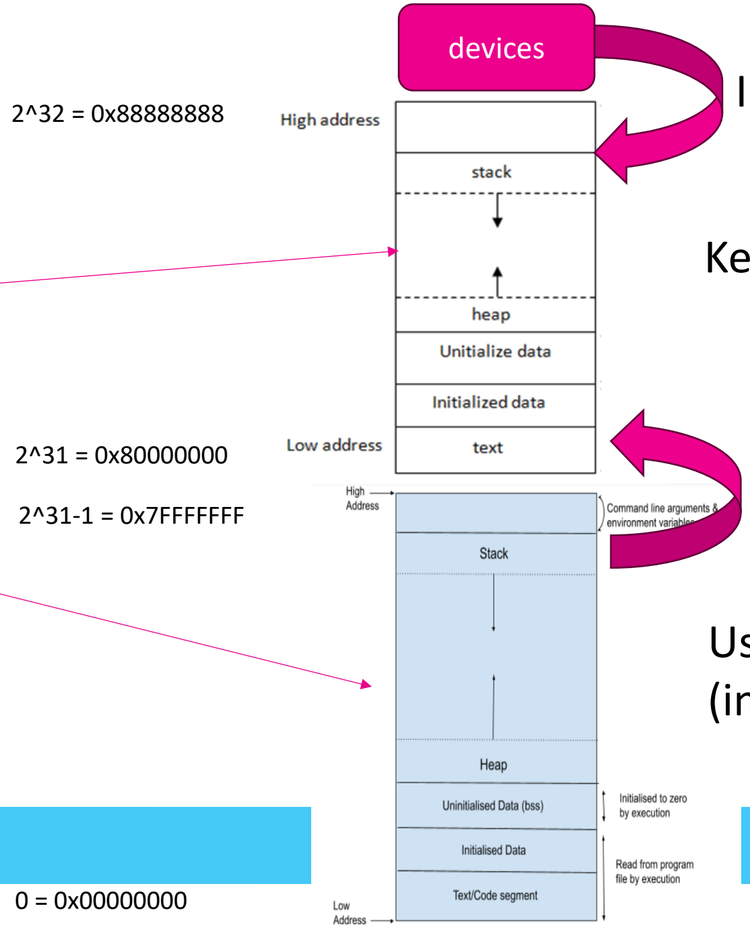

```
root@debian-virtualbox:/home/chris# cat test.c
#include <stdlib.h>
int main(void) {
char str[] = "Hello";
void* p ;
p=malloc(1000);
puts(str);
}

root@debian-virtualbox:/home/chris# size --format=GNU a.out
      text       data        bss      total filename
       473       1669          8       2150 a.out
root@debian-virtualbox:/home/chris#
```

Operating system address space

- [Protection ring - Wikipedia]

Example (details may differ)
- Windows/386 2.0
- Some IOT devices
- Some PLC’s





Interrupts

Kernel virtual address space

System calls

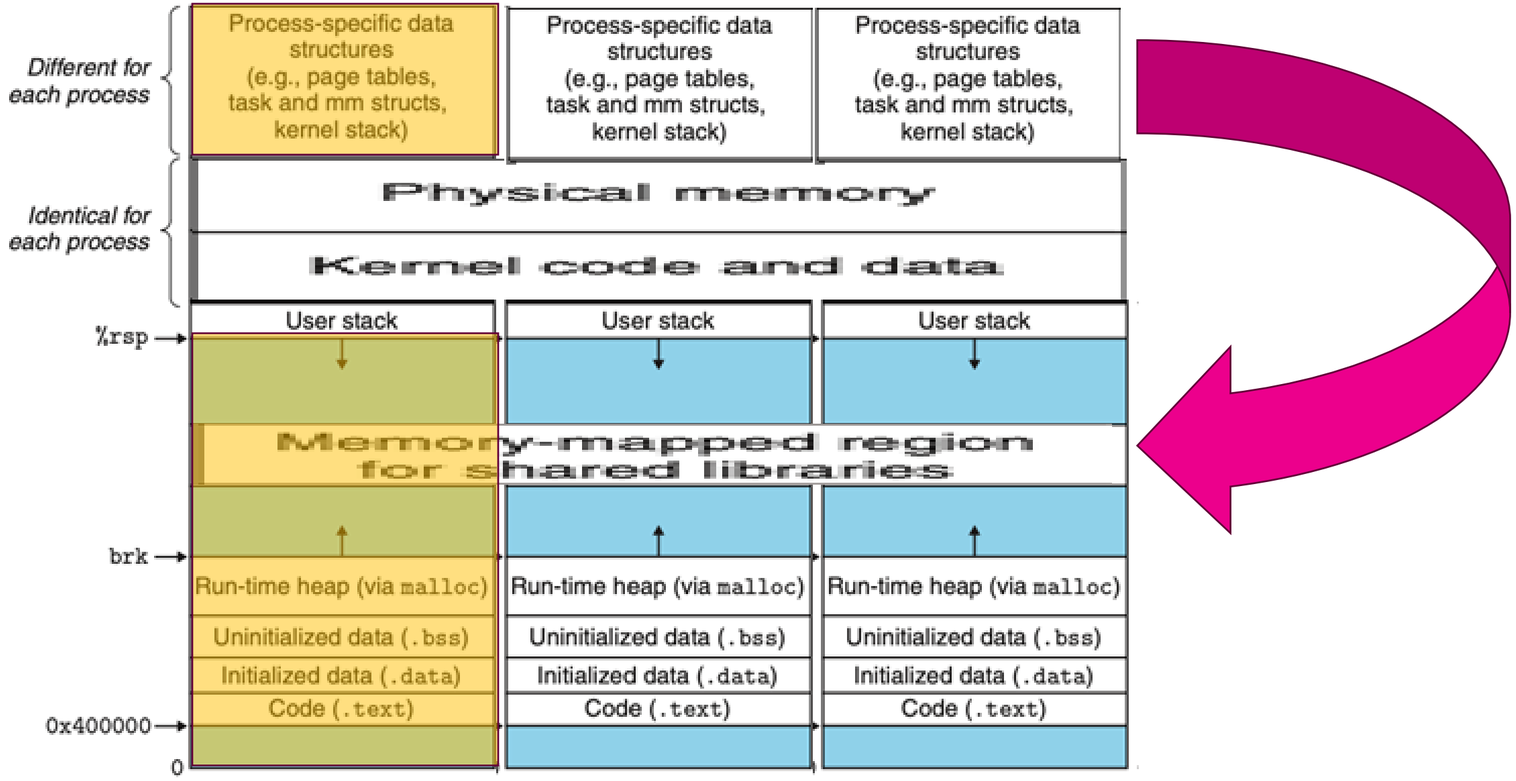
User virtual address space
(initially init/system, PID 1)

Introducing the schedular

Example (details may differ)

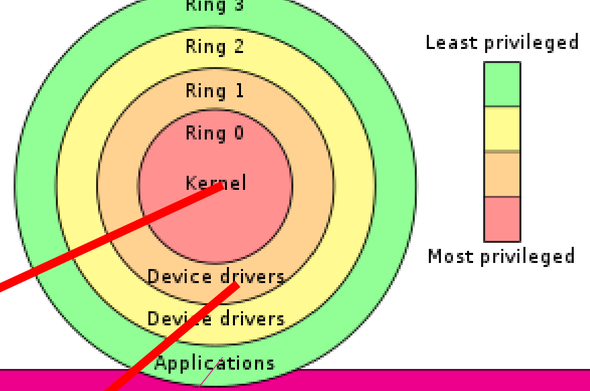
- BSD/SYSV/*ux/Linux
- Windows
- Much more modern OS'S

PCB : Process control block

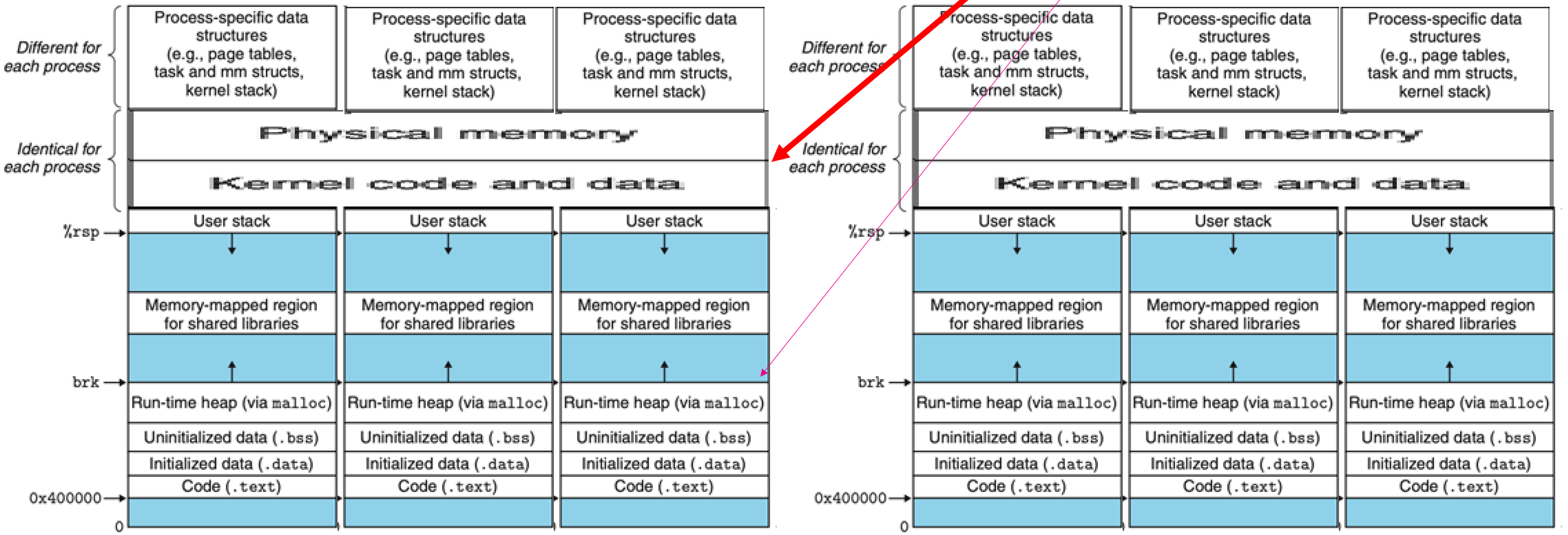


howest hogeschool

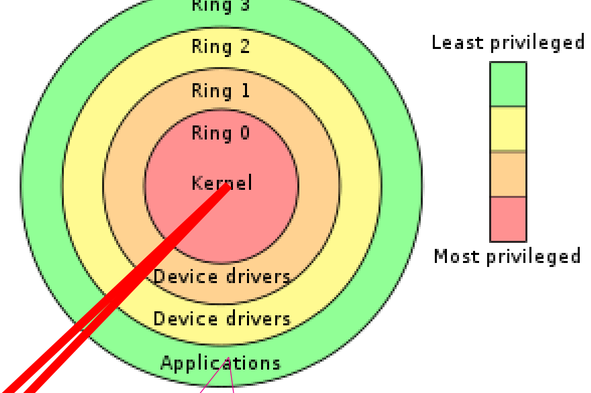
Full Virtualization (CPU emu)

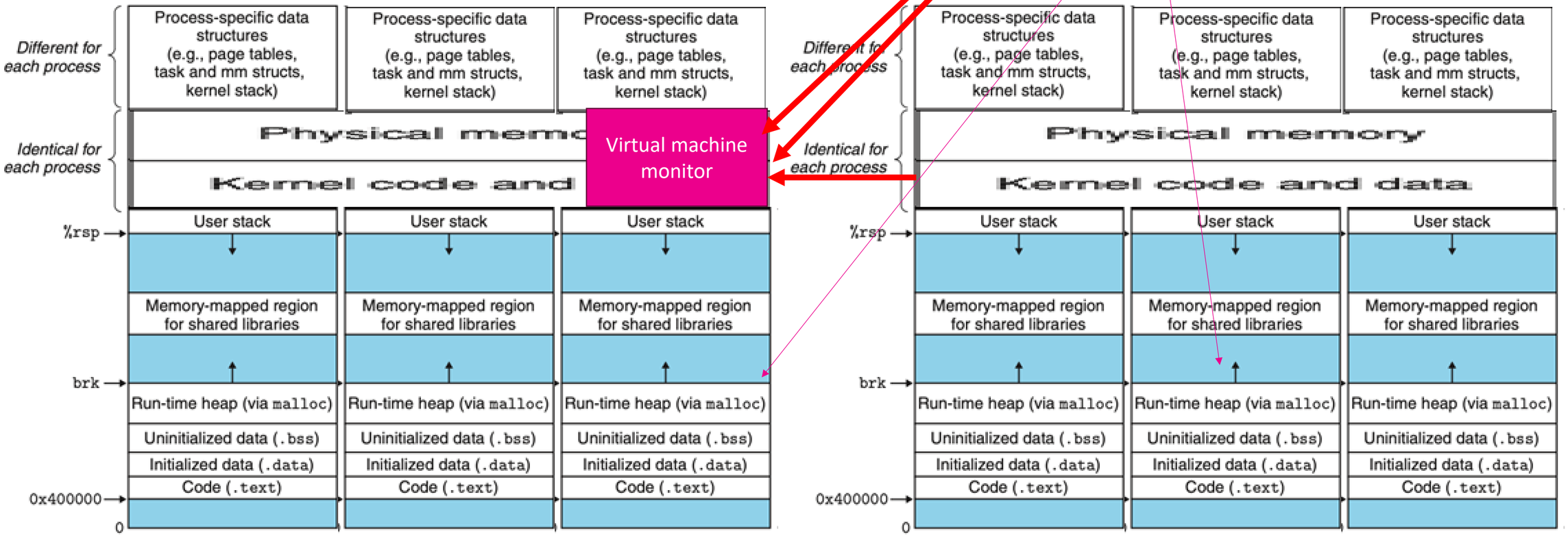


Hypervisor : critical code only, things that could influence another vm (eg : i/o)



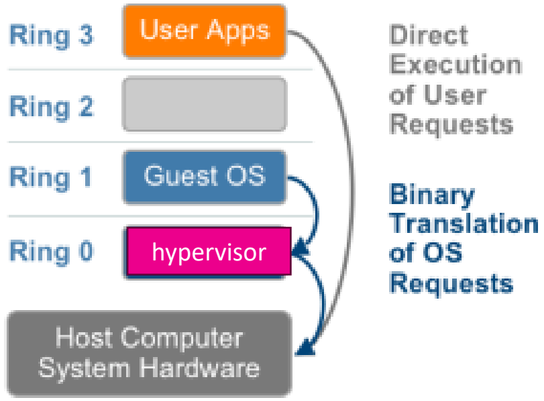
Paravirtualization (OS assisted)

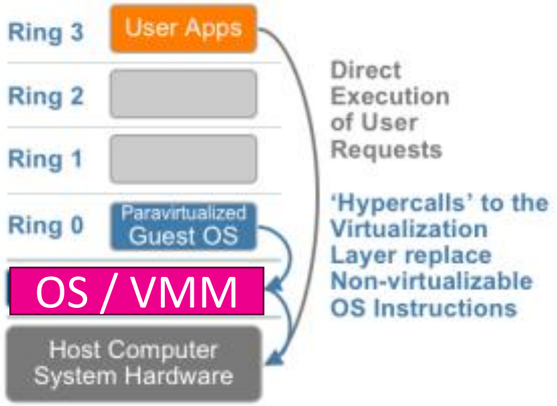




Comparing

Hypercalls are similar to kernel system calls. They allow the guest OS to communicate with the hypervisor.





Full Virtualisation (CPU emu)

- **Vmware ESX**
- **Hyper-V**
- **KVM**

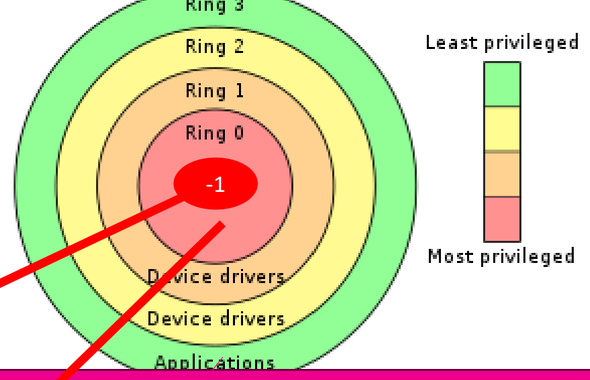
Paravirtualization (OS assisted)

- **VMWare player**
- **Oracle Virtual Box**
- **Microsoft virtual server (discontinued)**

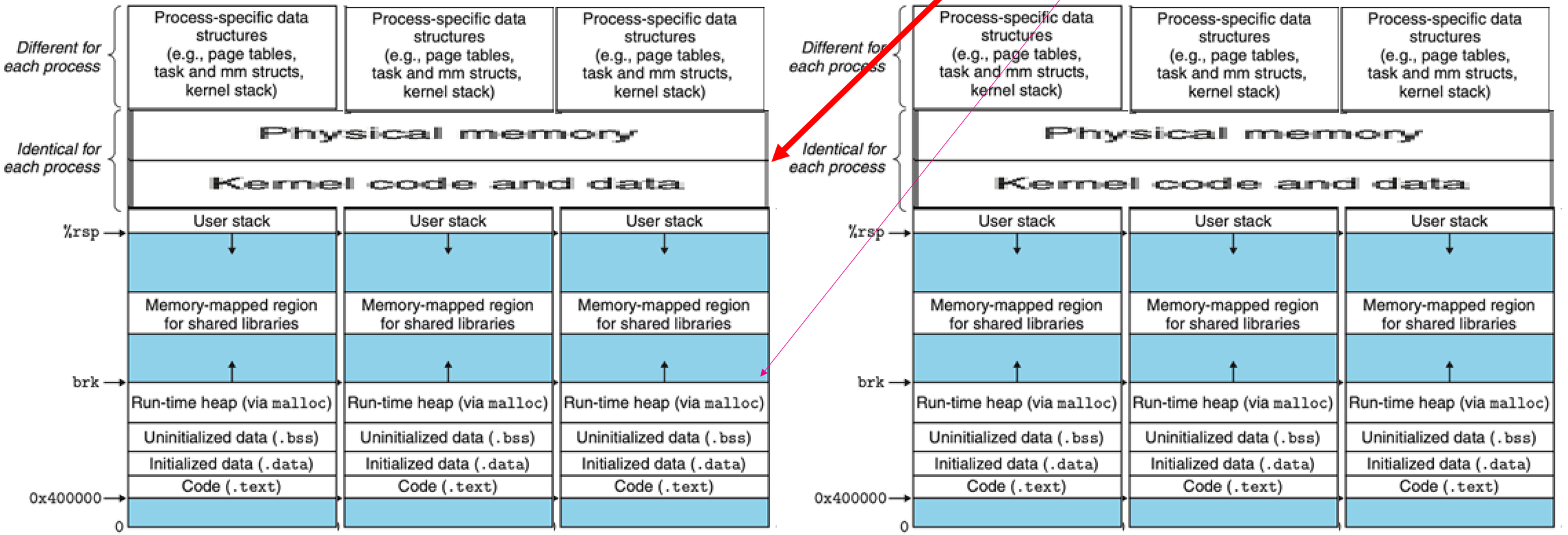
Difference between Full Virtualization and Paravirtualization -

howest hogeschool

Hardware Virtualisation



Hypervisor

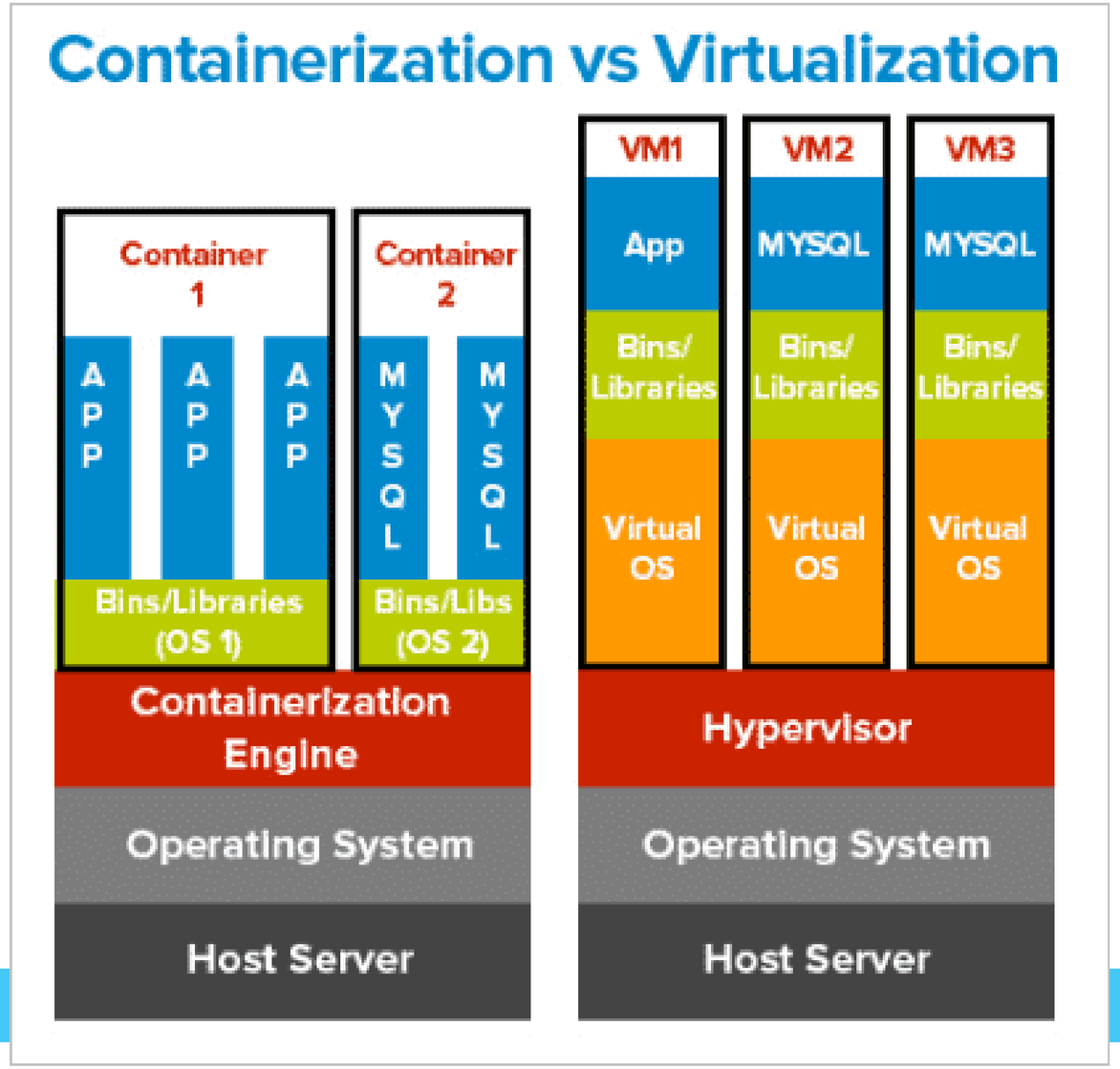


Containers what problems to resolve

- In traditional virtualization each app runs on a separate OS
  - OS VM, filesystems, shared libraries are duplicated
- An app depends on other apps of libraries (versions), not necessary the OS
- The need to quickly deploy as “microservices”

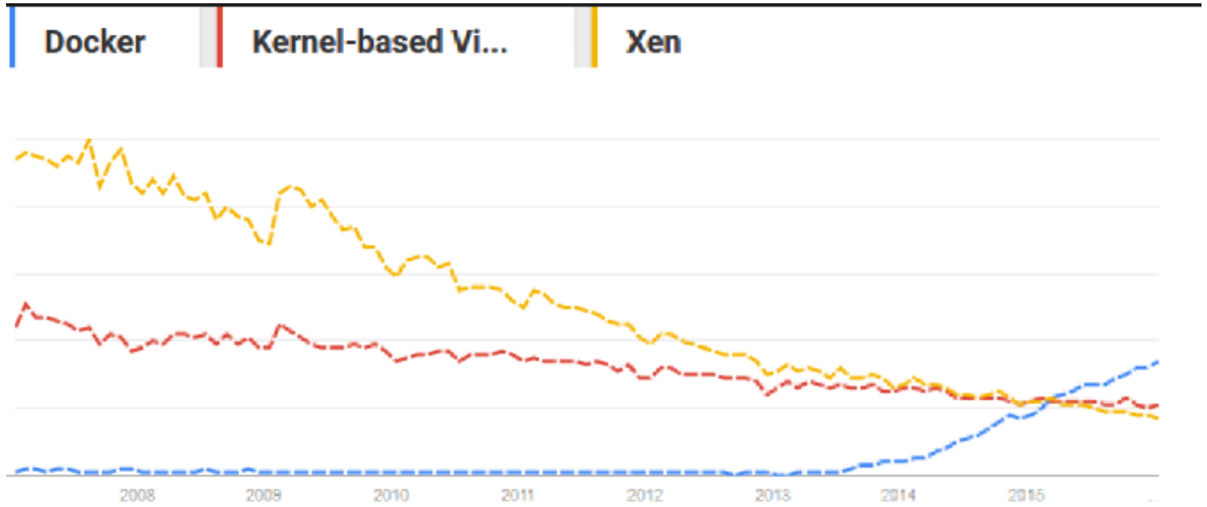
How do containers solve this problem?
Put simply, a container consists of an entire runtime environment: an application, plus all its dependencies, libraries and other binaries, and configuration files needed to run it, bundled into one package. By containerizing the application platform and its dependencies, differences in OS distributions and underlying infrastructure are abstracted away.

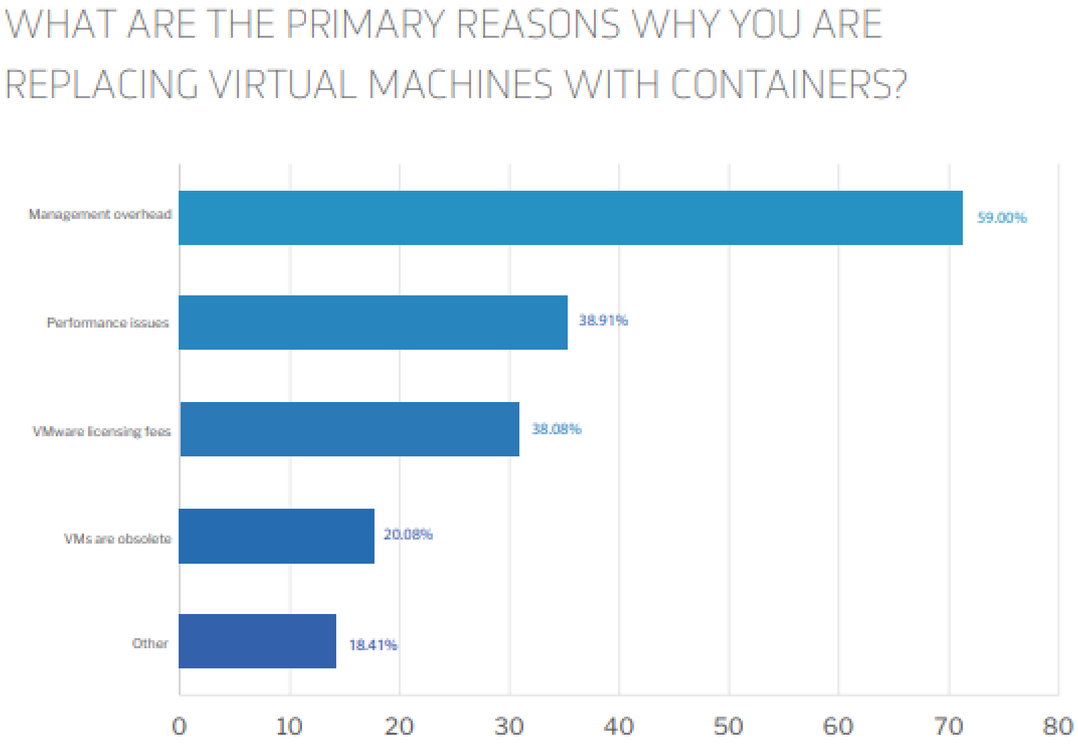
Containers architecture



howest hogeschool

Virtual machines against Containers



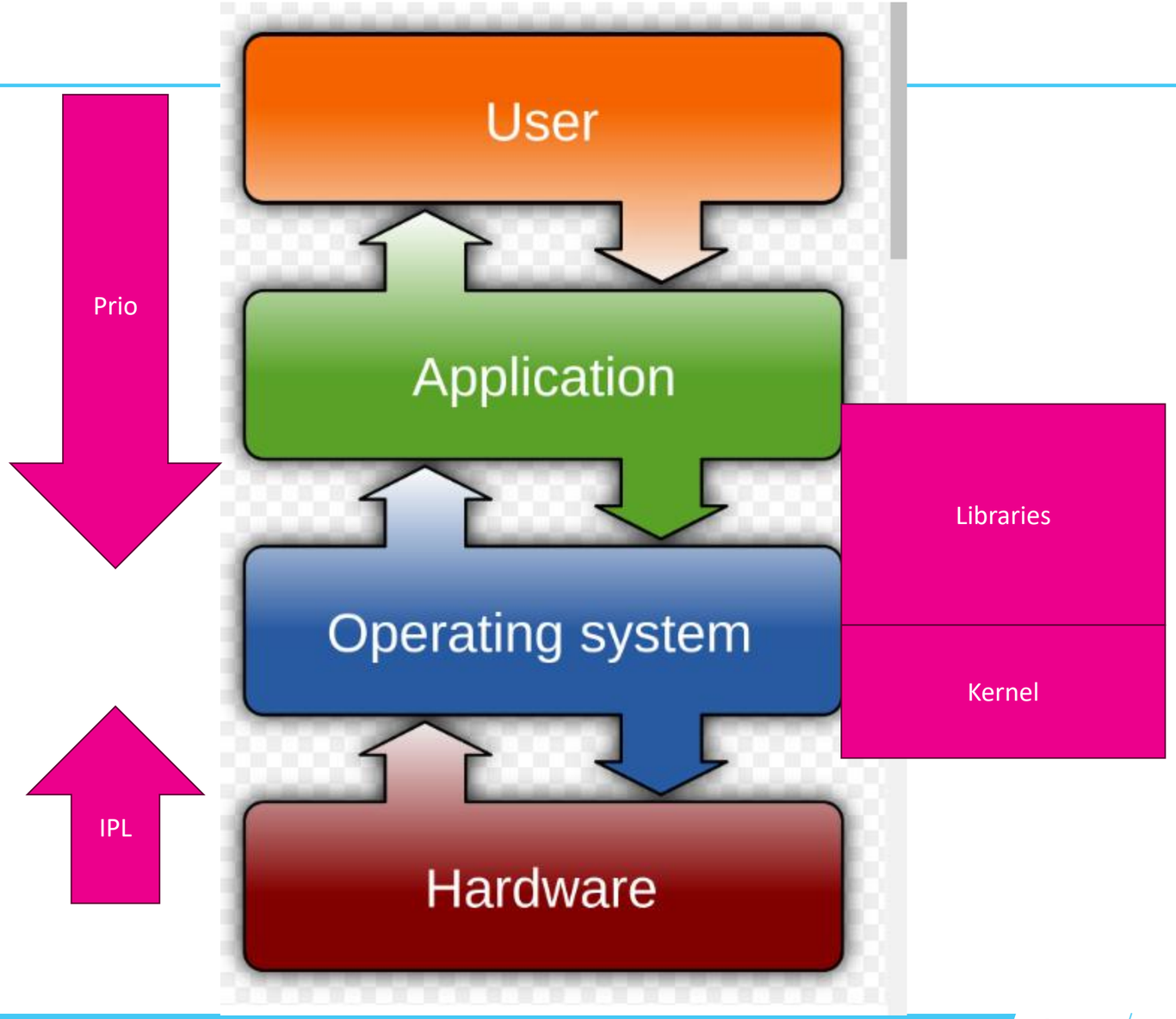


Management overhead (59 percent), performance (39 percent), VMware licensing fees (38 percent) are playing a significant role in driving organizations to move away from virtual machines in favor of containers. Remarkably, a total of 20 percent said they already believe virtual machines are obsolete.

howest
hogeschool

Back one level higher

Operating Systems



The Kernel

Run-Time Organization

- Top Half
  - per-process stack
  - library of shared code
  - maintains process structure (always resident)
  - maintains user structure (can be swapped out)
  - division between process/user dependent on memory
  - never preempted for another process (but can yield the processor)
  - can block interrupts by setting processor priority level (see discussion on bottom half)

Soft clock (milliseconds)

CPU Frequency (nanoseconds)

Hard clock (microseconds)

Run-Time Organization II

- Bottom Half
  - handles hardware interrupts
  - asynchronous activities (wrt Top Half)
  - special kernel stack (might not be a process running)
  - Top and bottom half coordinate around work queues using mutexes
  - top half starts I/O requests, waits for bottom half to finish

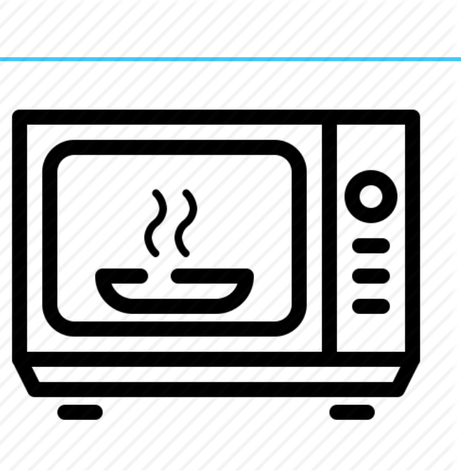
howest hogeschool

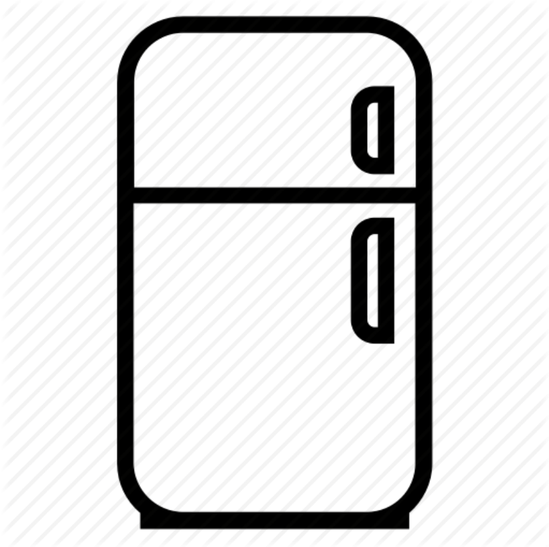
What about?

Game consoles?

Microwaves – oven?

IoT devices / wearables?





howest hogeschool

What about emulators & virtualization & simulation?

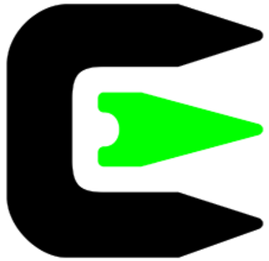
JVM

(Java Virtual Machine)

WSL

vs

Cygwin

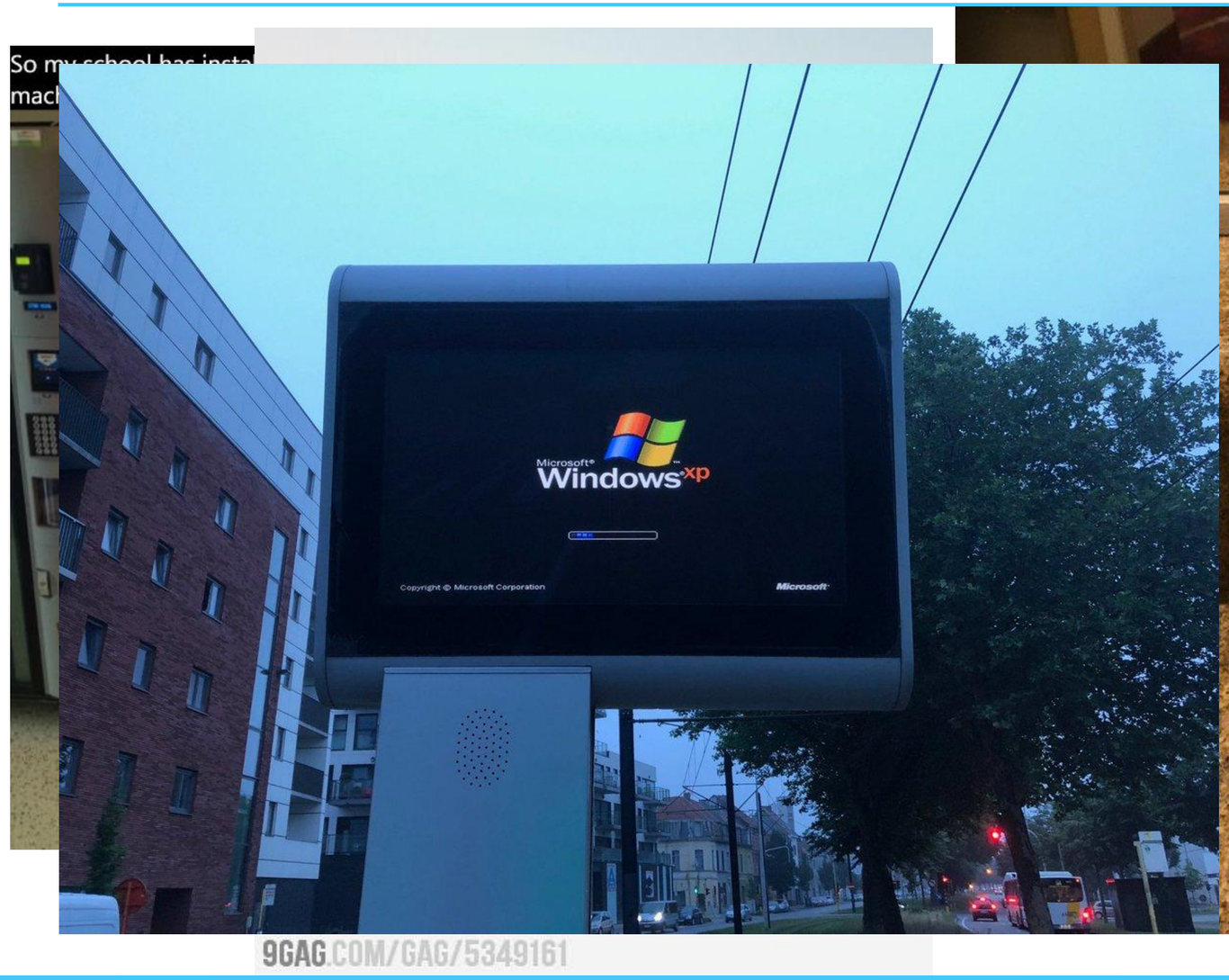


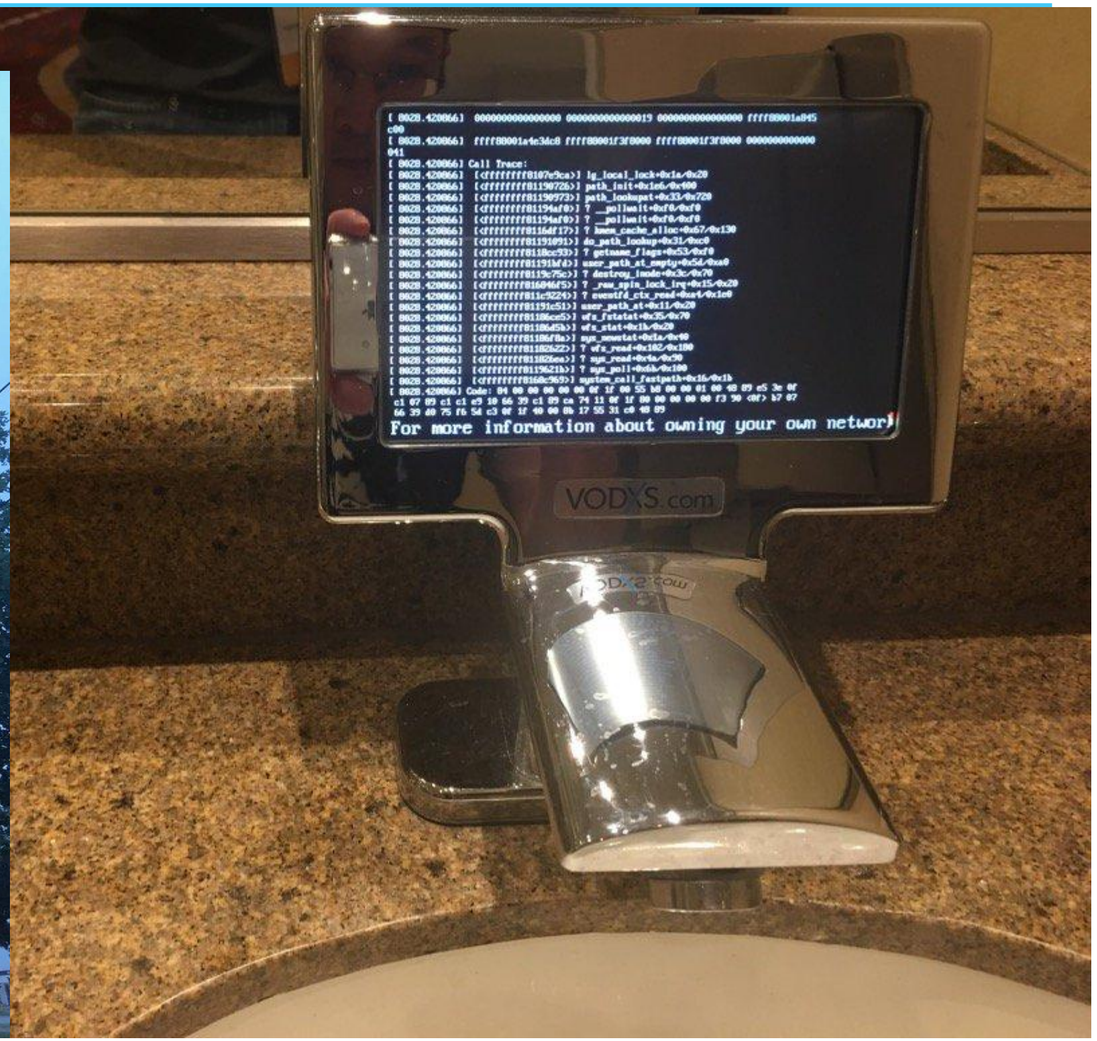
vs

MinGW

howest hogeschool

Operating Systems are everywhere

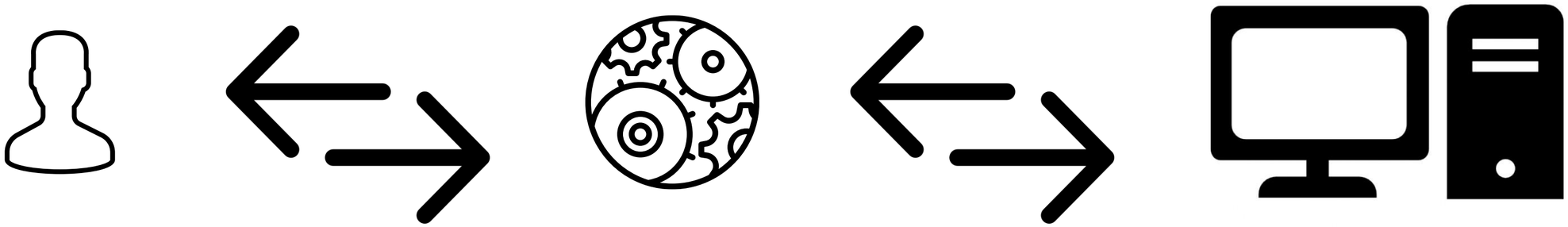




Definition

Informal definition

A device that only translates a basic input in one action has/is no OS. If there is some sort of managing by itself or it acts as something between the user and the device (hardware) we can speak about an operating system.



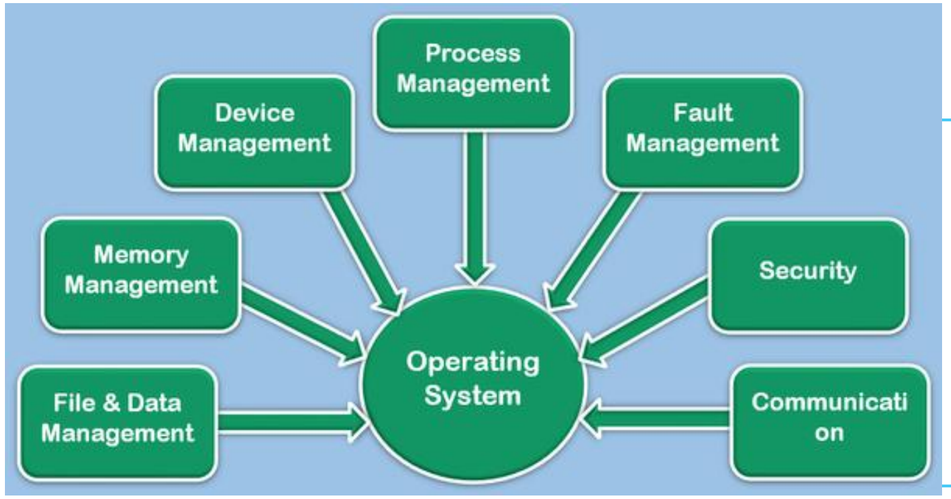
OS

howest hogeschool

Definition

Formal definition

“There is a body of **software**, in fact, that is responsible for making it easy to run programs (even allowing you to seemingly run many at the same time), **allowing** programs to share memory, enabling programs to interact with devices, and other fun **stuff** like that. That body of software is called the **operating system (OS)**, as it is **in charge** of making sure the system operates correctly and efficiently in an easy-to-use manner.”



Source quote:

http://pages.cs.wisc.edu/~remzi/OSTEP/intro.pdf

Source image:

https://www.studentsclub.xyz/2019/02/What-is-Operating-System.html

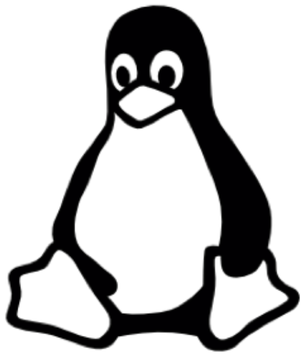
Operating System characteristics

Summarized:

- What is an operating system?
  - It’s software
  - A set of programs
- What is the **function** of an operating system?
  - Management of system resources (hardware), to improve operations of the system
  - Communicates with the user to hide the complexity of the system

Different types of Operating Systems

- There are different type of OS's. It all comes back to:
  - The **point of view of the user**
    - User friendliness
    - Not (too) technical
    - Customization
  - The **point of view of the system**
    - Efficiency
    - Management of data and/or hardware
  - The **focus** has shifted!
    - In former times the focus was on system aspects and/or data processing
    - Modern OS's focus on the user



Types of OS's

Some **examples** of **classifications** of OS's:

- Single user OS
- Multi-tasking
- Batch Processing
- Multi-programming
- Multi-processing
- Real Time Operating System
- Distributed Data processing
- …

No need to study this list by heart

Questions:

- What is a mainframe?
- OS?
- Still used?

howest hogeschool

Goal(s) of the OS

To be the manager!

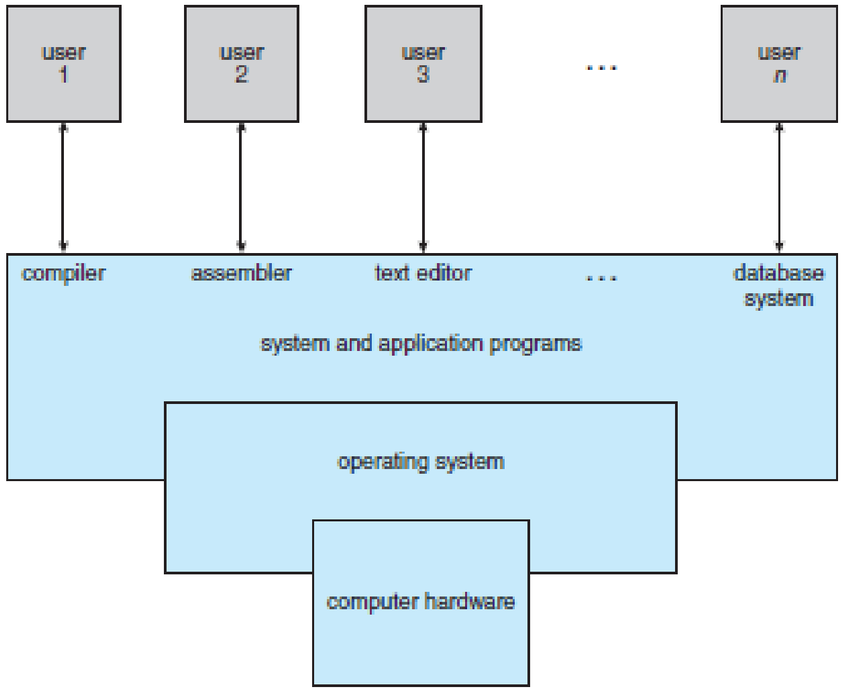


Figure 1.1 Abstract view of the components of a computer system.

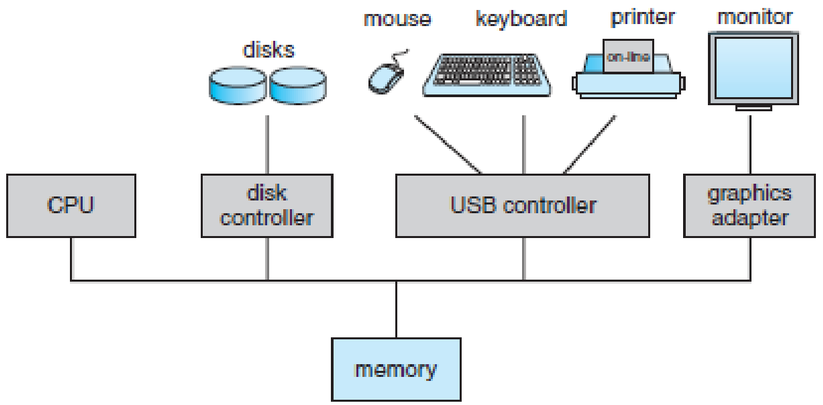


Figure 1.2 A modern computer system.

Goal(s) of the OS

Some functions of the OS (this list is not complete):

- Booting
- **Memory management**
- Loading & Execution
- Data security
- Disk/device management (in general – hardware management)
- **Process management**
- Avoid same requests at the same time (locking)

The kernel – Overview

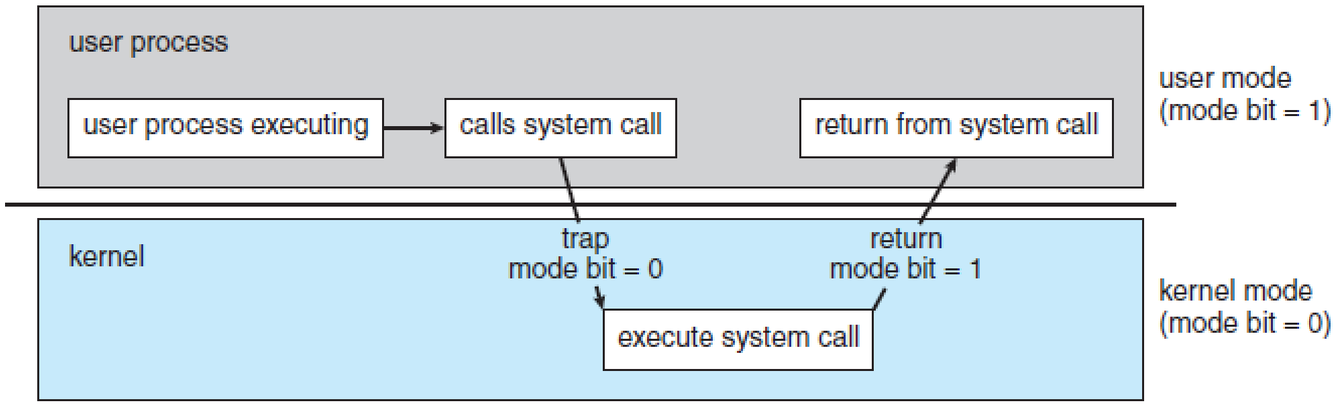


Figure 1.10 Transition from user to kernel mode.

Kernel space vs **User space!**

- Implemented with a **bit**
- **Kernel** has complete control and handles (almost) everything

Kernelspace = system mode – priviliged mode – supervisor mode – secure mode – unrestricted mode

Userspace = ordinary mode – user mode – restricted mode

The kernel – System calls

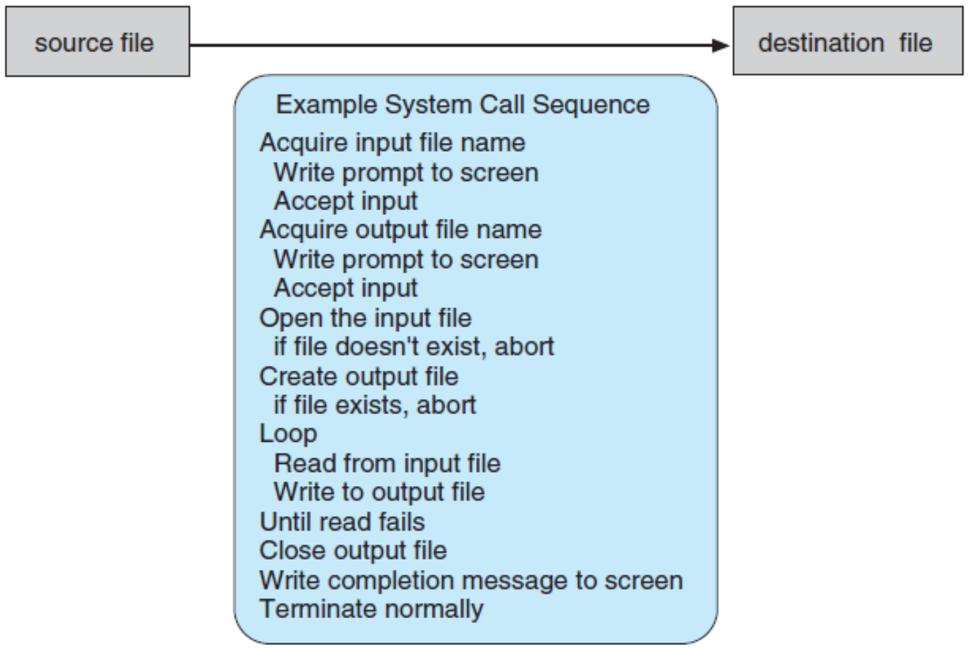


Figure 2.5 Example of how system calls are used.

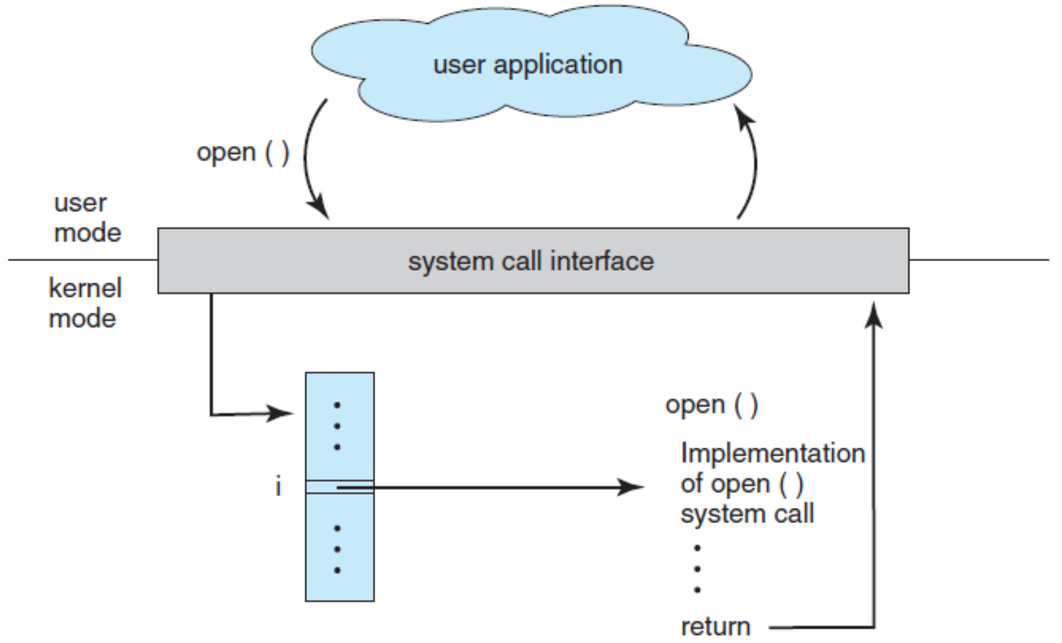


Figure 2.6 The handling of a user application invoking the `open()` system call.

Fallout: Reading Kernel Writes From User Space

Source: https://mdsattacks.com/files/fallout.pdf

“ In particular, the Meltdown attack leaks information from the operating system kernel to user space, completely eroding the security of the system.”

~ May 2019

howest hogeschool

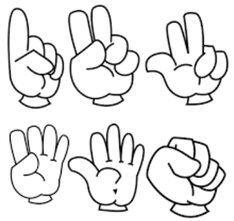
howest
hogeschool

Time for some Computer Science History

History of Computers

Computer → need to automate math (Add)

- First “the hand & fingers”
- Chinese abacus (= *telraam* in Dutch)
- Pascal’s Calculator
  - (1642)



howest hogeschool

History of Computers

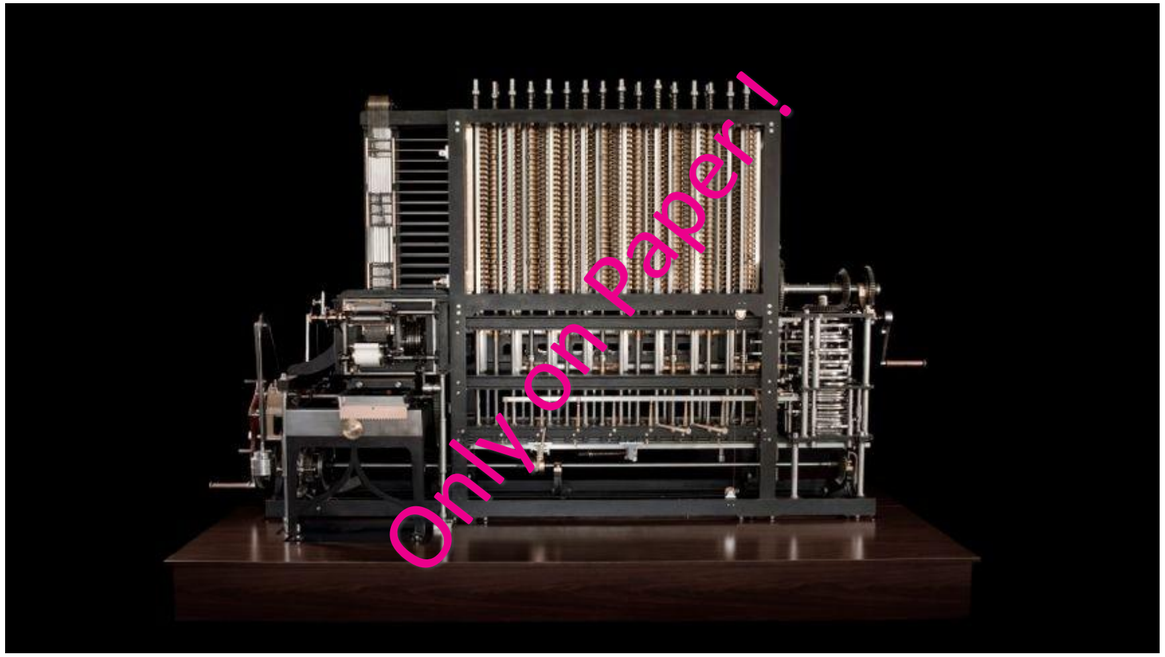
Computer → need to automate math (add/subtract/multiply/devide)

- Leibniz Calculating Machine
  - Start in 1671
  - Finished in 1694

howest hogeschool

History of Computers

- First “computer”?
  - 1822 – Charles Babbage (steam-powered)



- Ada Lovelace (Byron)
  - Credited as worlds first computer programmer
  - Helped Babbage with the **Analytical Engine/Machine**
  - “Ada” programming language

"Hello, world!" in Ada [edit]

A common example of a language's syntax is the Hello world program: (hello.adb)

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Hello is
begin
   Put_Line ("Hello, world!");
end Hello;
```

This program can be compiled by using the freely available open source compiler GNAT, by executing

```
gnatmake hello.adb
```

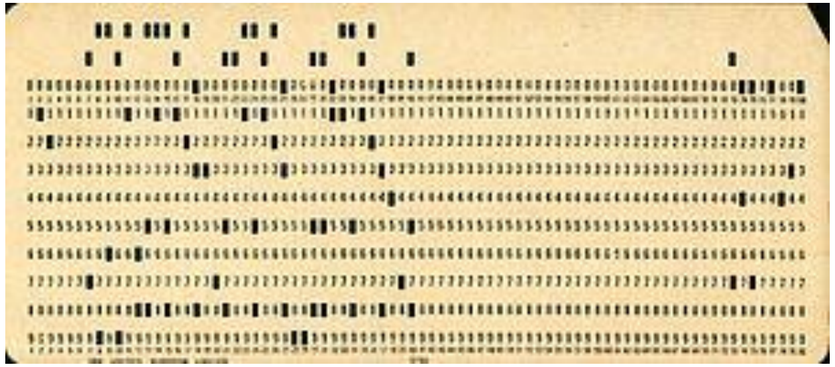
Src: https://en.wikipedia.org/wiki/Ada_(programming_language)

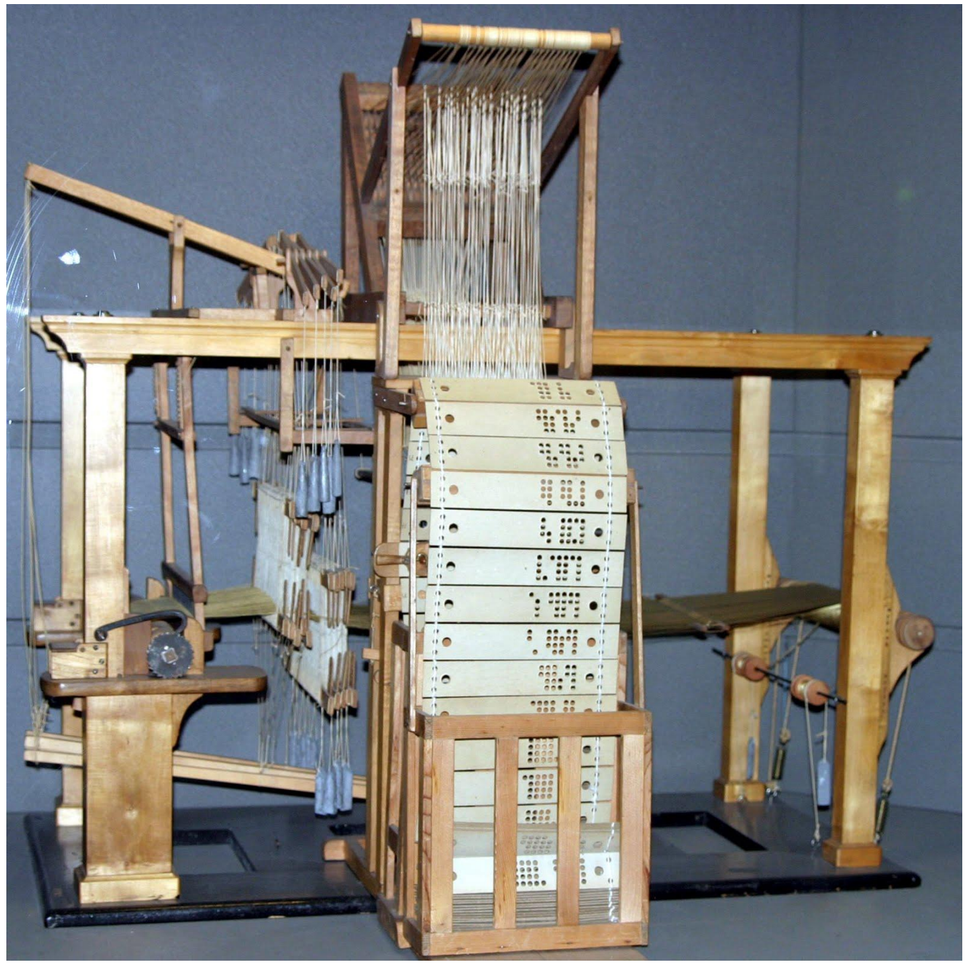
howest hogeschool

History of Computers

Computer → need to automate math (Counting)

- Jacquard's Loom (*Weefgetouw* in Dutch)
  - 1804
  - Punch Card (*Ponskaart* in Dutch)
    - = "Program" = "set of instructions"
- Tabulating Machine (H. Hollerith)
  - 1890
  - ~ IBM

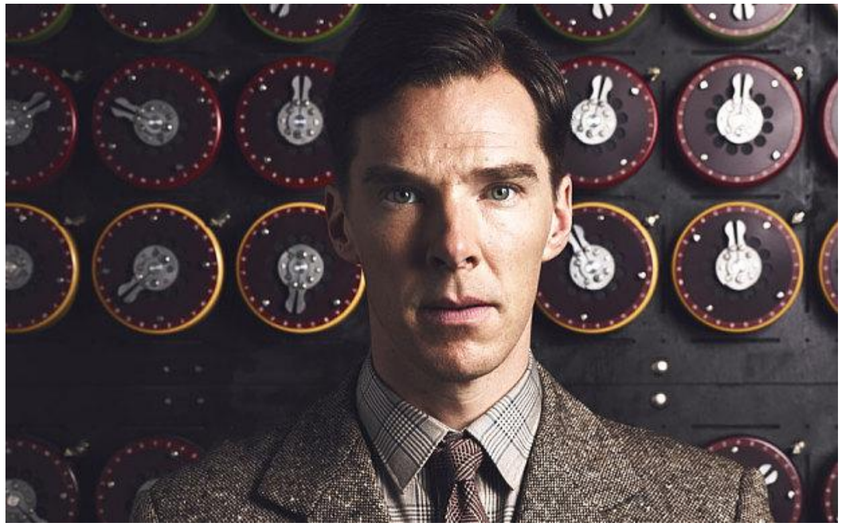


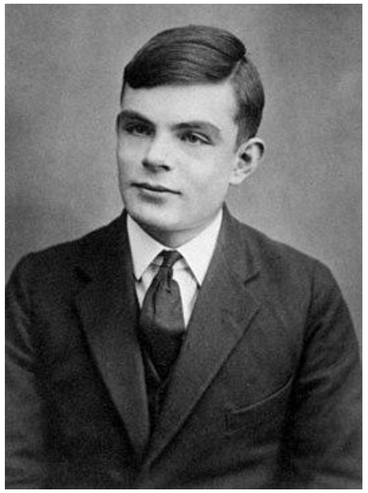


howest hogeschool

History of Computers

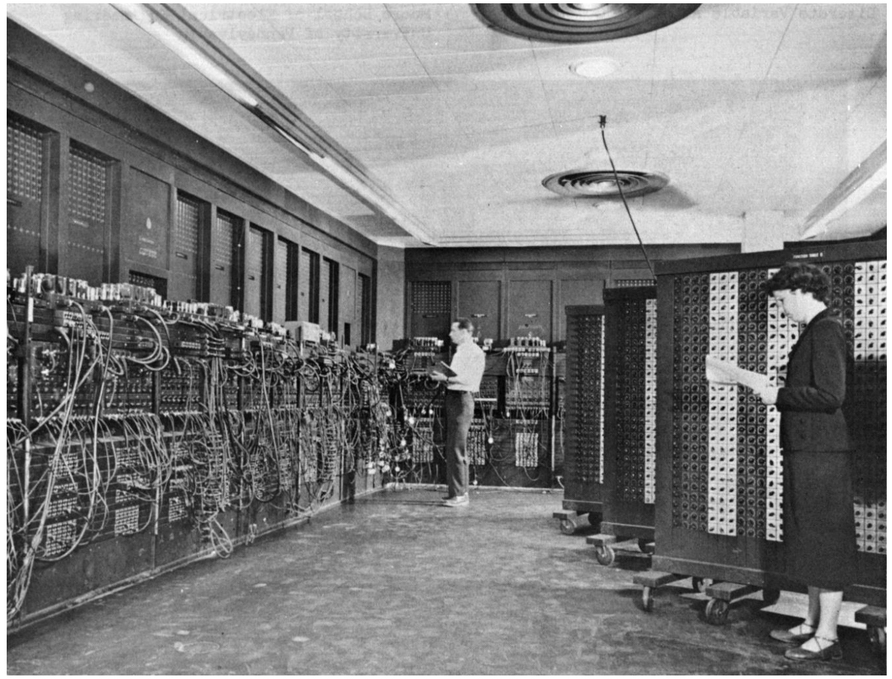
- First programmable computer?
  - 1936-1938 – Konrad Zuse
  - It is considered to be the first electromechanical binary programmable computer, and the first really functional modern computer.
- 1936 – **Alan Turing**
  - **Enigma!**





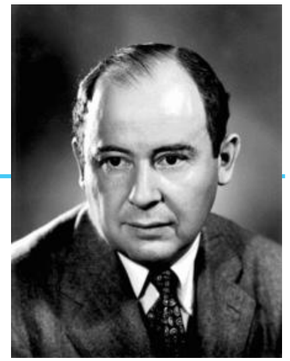
History of Computers

- *(…) – this enumeration does not list everything!*
  - 1937 – 1944 **MARK-I**
  - 1943 – 1946 construction of the **ENIAC** (US)
  - 1949 – the **EDSAC** (England)
    - First one to run graphical game = OXO (tic-tac-toe)
  - 1949 – **UNIVAC** mainframes
- First commercial computer?
  - Remember Konrad Zuse?
    - 1942 started working on Z4, sold in 1950
  - 1953 – **IBM** the 701
  - 1955 – first pc with RAM

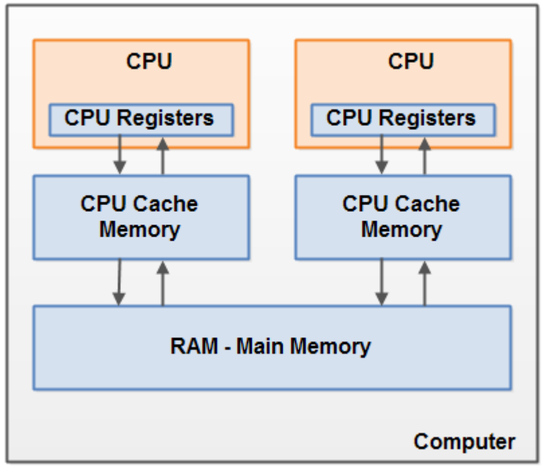


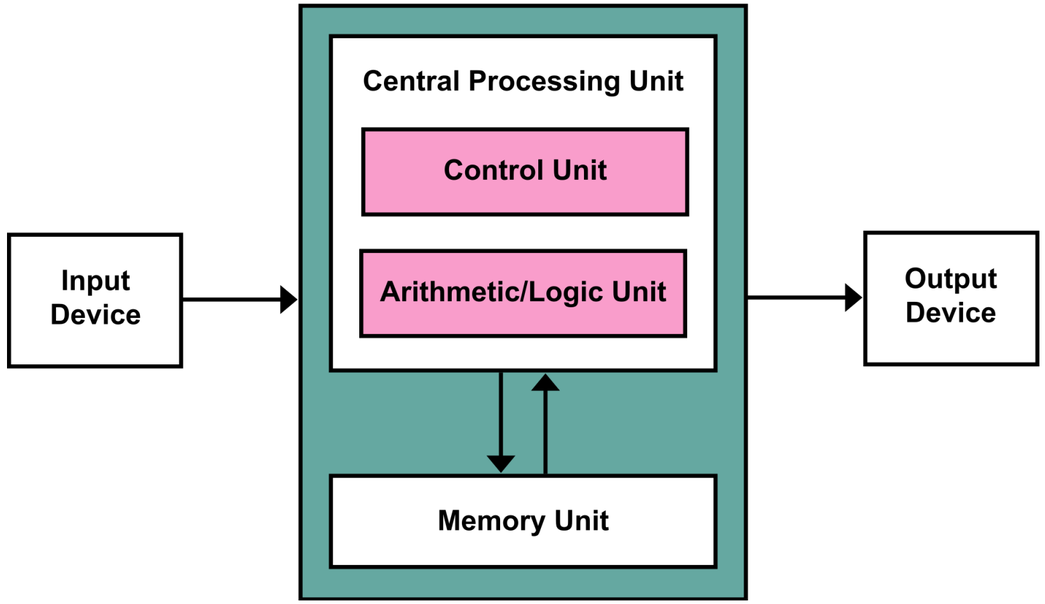
howest hogeschool

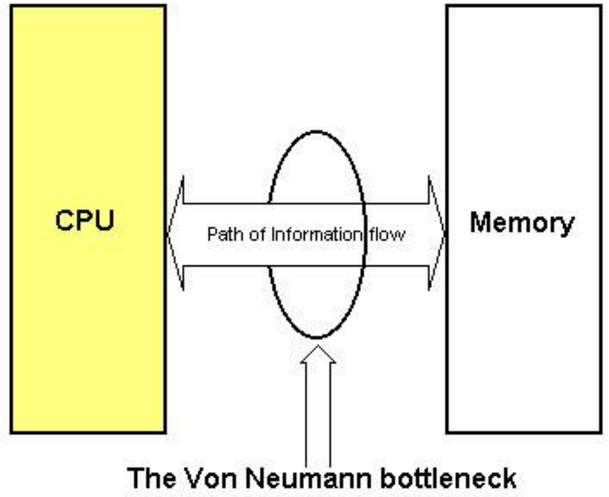
Von Neumann-architecture



RAM – model







Vs. Harvard Architecture: https://www.youtube.com/watch?v=4nY7mNHLrLk

howest hogeschool


SYSTEM 2404

Timing issues

- Clock cycle : CPU frequency defines the number of gate changes per second
  - Eg : 3 Mherz : 3 billion instruction (cycles)per second
- Hardclock ticks : Each interrupt is referred to as a *tick*.
  - On modern computers, the clock ticks 1 million times per second. At each tick, the system updates the current time of day as well as user-process and system timers. Note that the time can wrap around. On a 32-bit system where CLOCKS_PER_SEC equals 1.000.000 this function will return the same value approximately every 72 minutes.
  - On POSIX-conformant systems, time_t is a signed integer type and its values represent the number of seconds elapsed since the epoch, which is 00:00:00 on January 1, 1970, Coordinated Universal Time.

```
struct timespec {
time_t   tv_sec;      /* seconds */
long     tv_nsec;     /* nanoseconds */
};
```

- Softclock timer interrupt
  - Each 200 ms the kernel interrupts for time sharing
  - Highest prio processes get rescheduled, sharing 200ms for the same prio processes
- Local time conversion changes the OS time value depending on the world

Year 2038 problem - Wikipedia

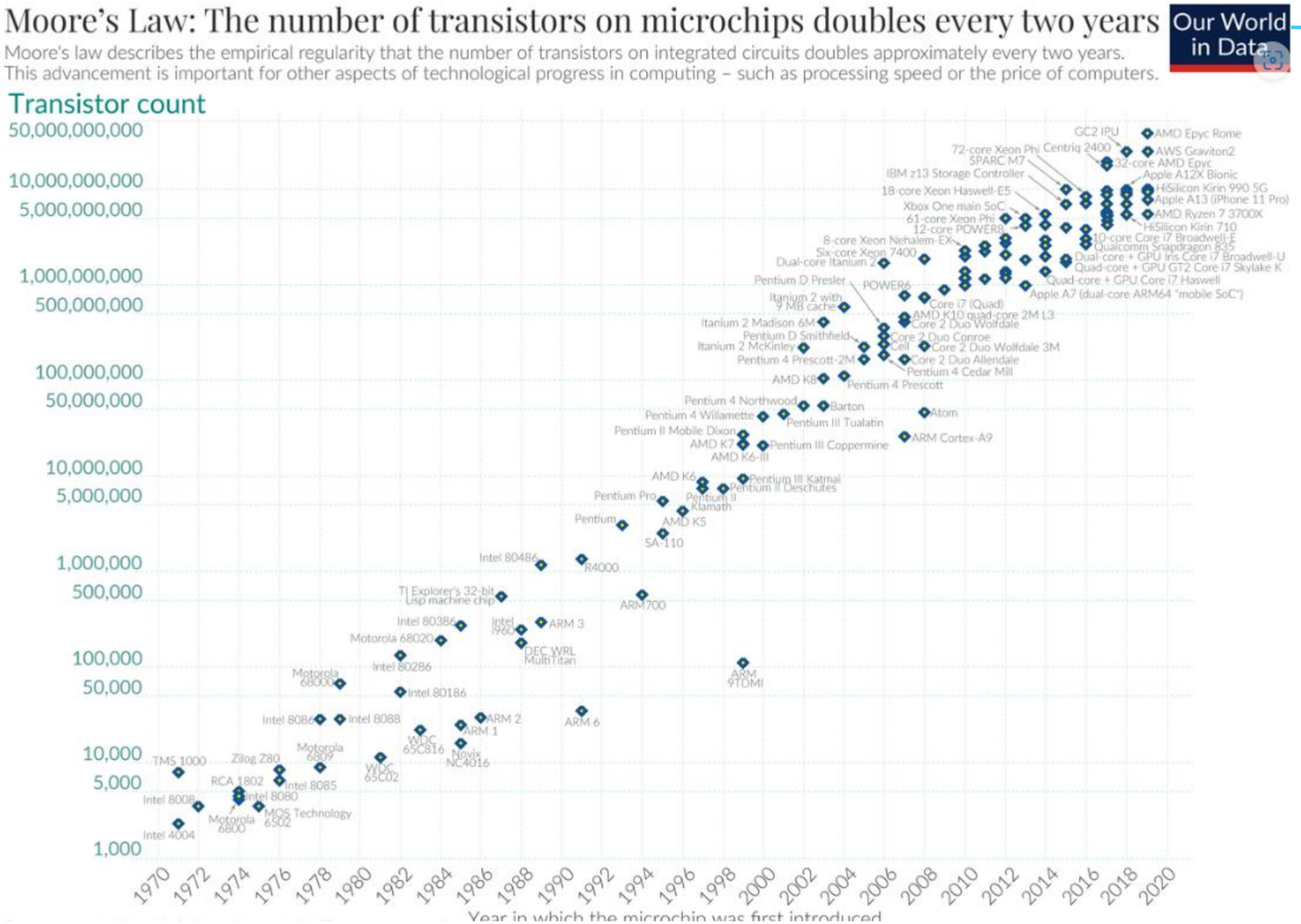
howest hogeschool

Recap: Moore's Law

The number of transistors on a microchip doubles every two years

(though the cost of computers is halved)

Size: µm → nm



[Transistor count - Wikipedia]

howest hogeschool

History of Computers

- *Academic world and business collaborate*
  - *Create new computers*
  - Create OPERATING SYSTEMS for HARDWARE
  - Vice Versa
- Bell Labs,MIT, General Electric
  - GE-645
  - Multi-user !
  - Multics

Beards

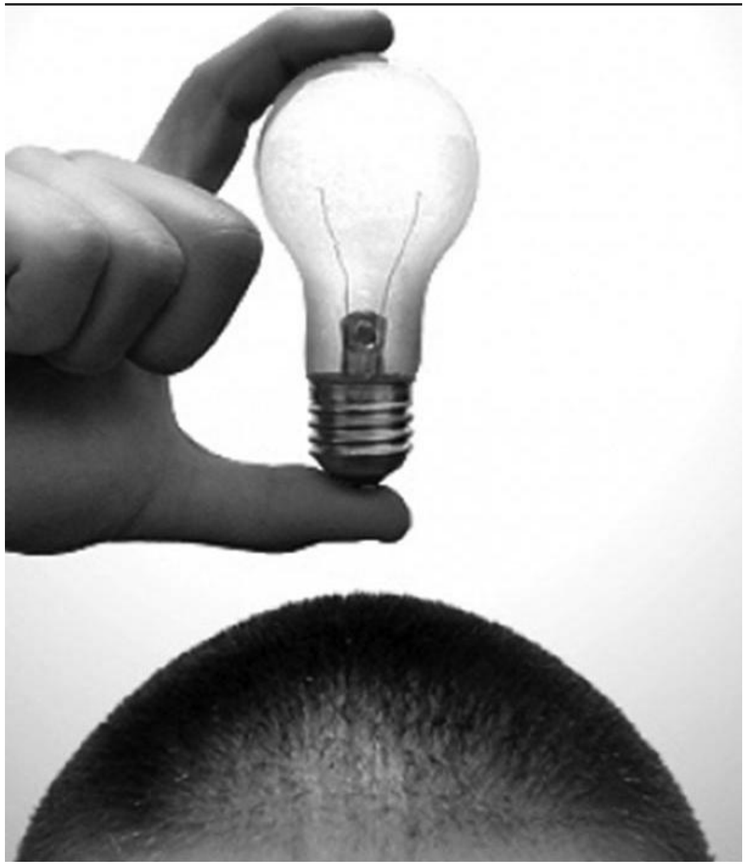
Ken Thompson

- UNIX
  - For PDP-7
- B Programming Language
- Google
  - GO Language
- UTF-8

Dennis Ritchie

- C !
- “SysV & BSD are the spiritual successors of Linux”

The Unix Philosophy



- 4 core principles
  - → Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features".
  - → Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.
  - → Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them
  - → Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them

The Unix Philosophy

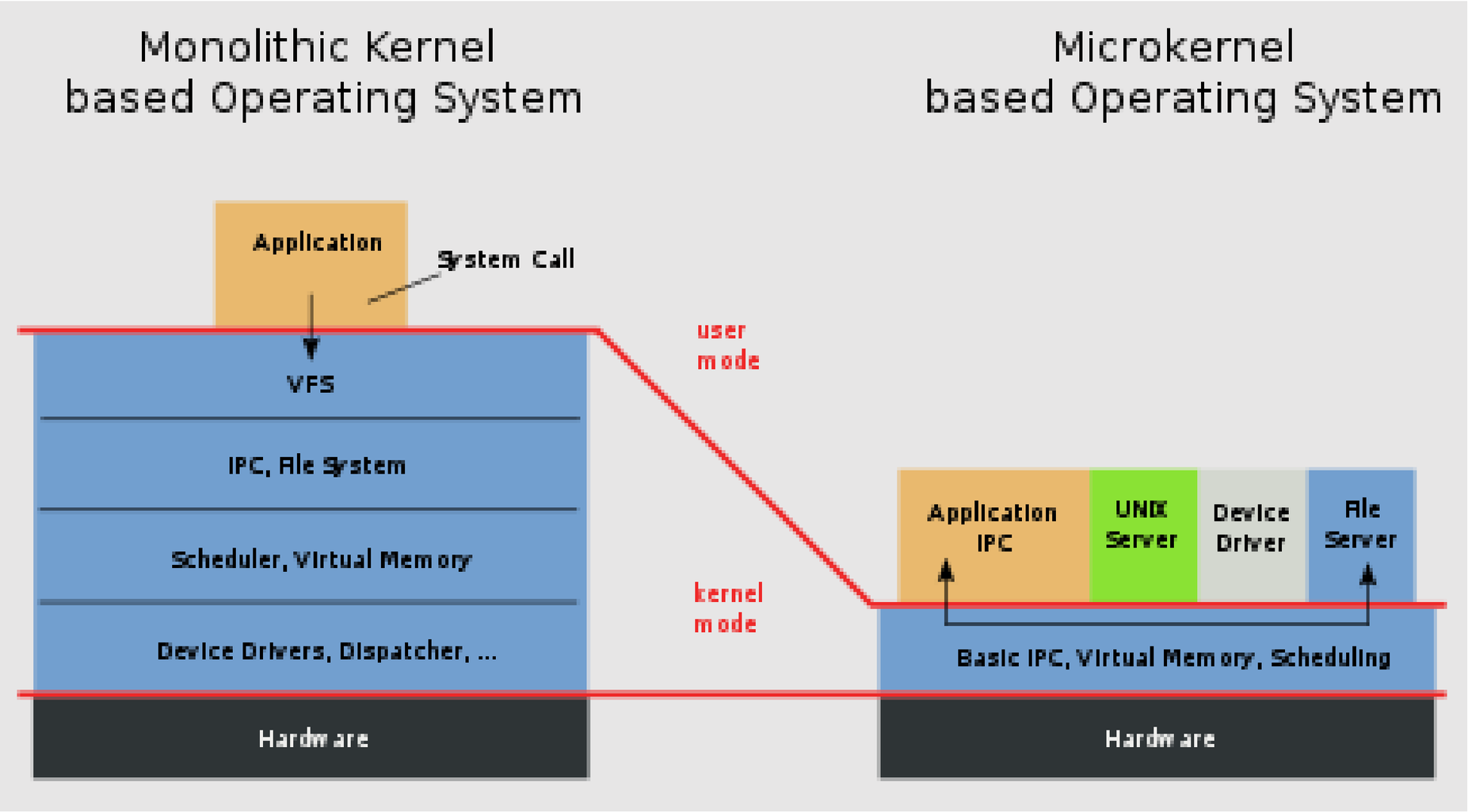
Based on the first UNIX developers

- Cultural & ethical approaches
- Minimalist and modular software development

Build simple, short, clear, modular, and extensible code that can be easily maintained and repurposed by developers other than its creators

Composability as opposed to monolithic design

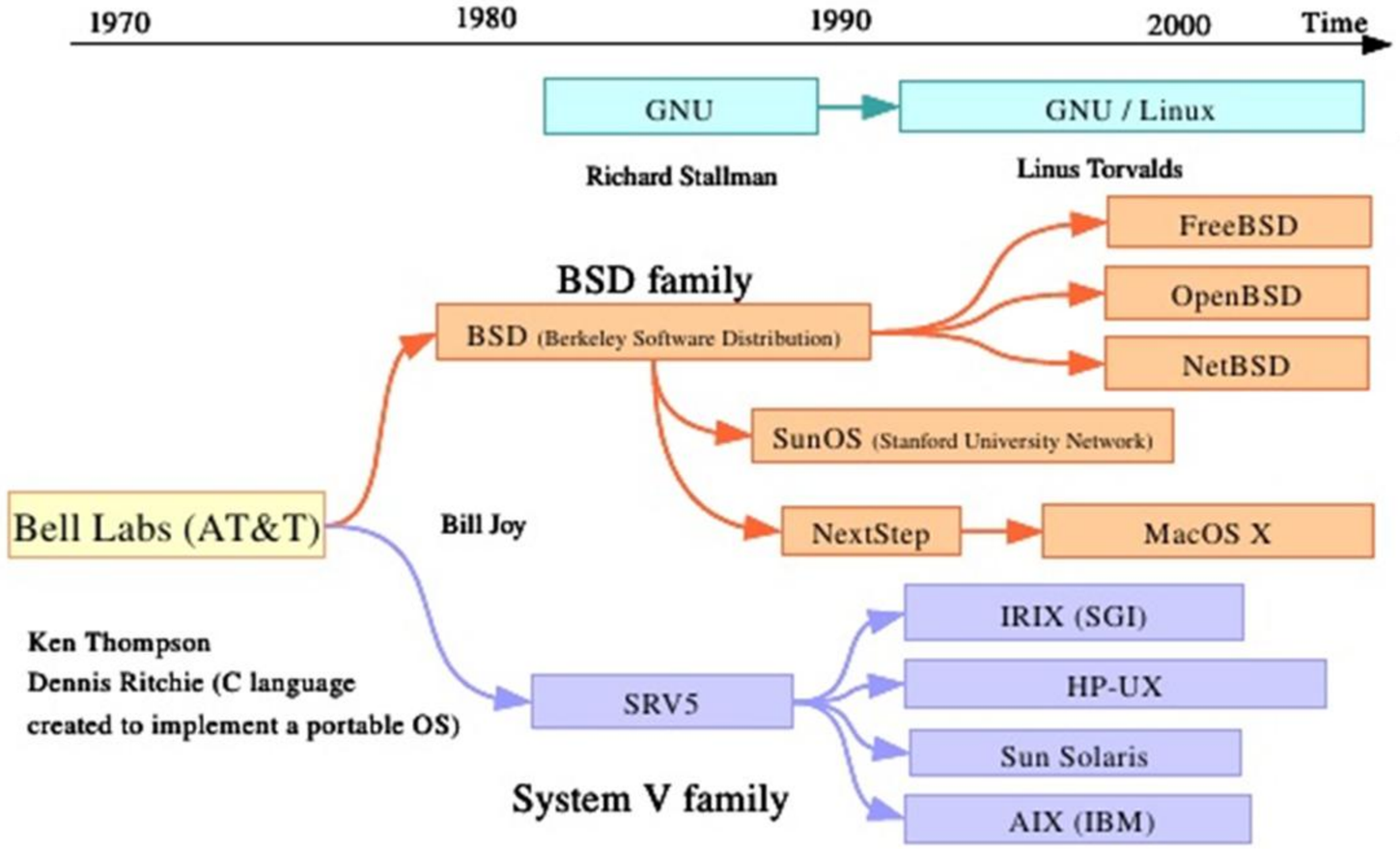
Worse is better



The impact of Unix

- Interactivity
- Nominal fee
- Targeted for educational and lower spec hardware
- Fun to hack !
- Assembler => C (or another language)
- Hierarchical file system with arbitrarily nested subdirectories
- command interpreter is “just another program”
- syntax for regular expressions

The offspring of Unix

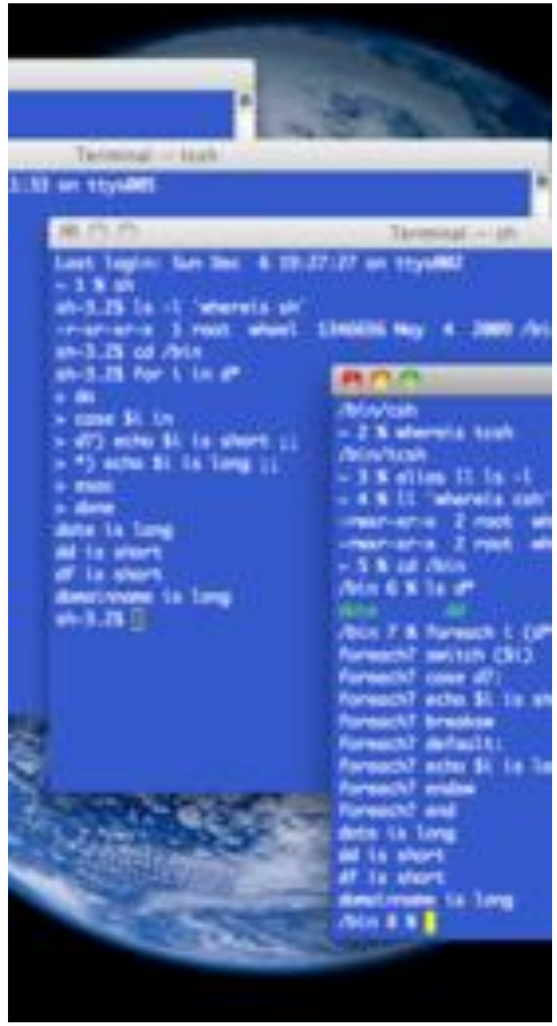


Then what happened ?

- Licence madness !
- Commercial Initiatives
  - →BSD
    - →MacOS
  - →SystemV
    - →Solaris, AIX, HP-UX
- Open Software Foundation
  - →True64 Unix
- Linux / OpenBSD
- Lots of good insights
  - →STANDARDIZATION
    - →POSIX
    - →UTF-8

Standardization : Posix

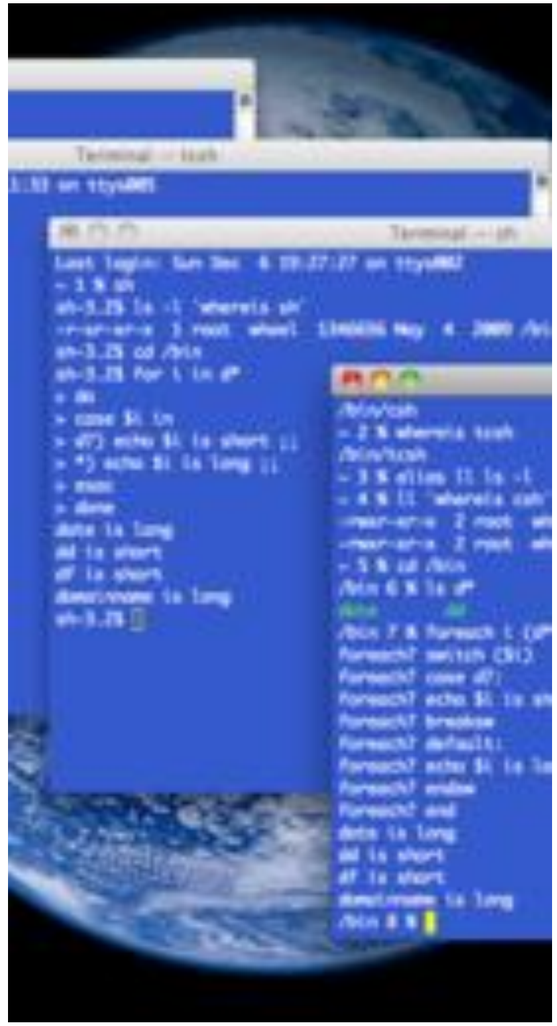
- Portable Operating System Interface
  - → IEEE
  - → https://www.computer.org/
  - → Compatibility between operating systems
  - → API
  - → Utility Interfaces
  - → Unix was the standard !
  - → “Manufacturer neutral”



howest hogeschool

Posix

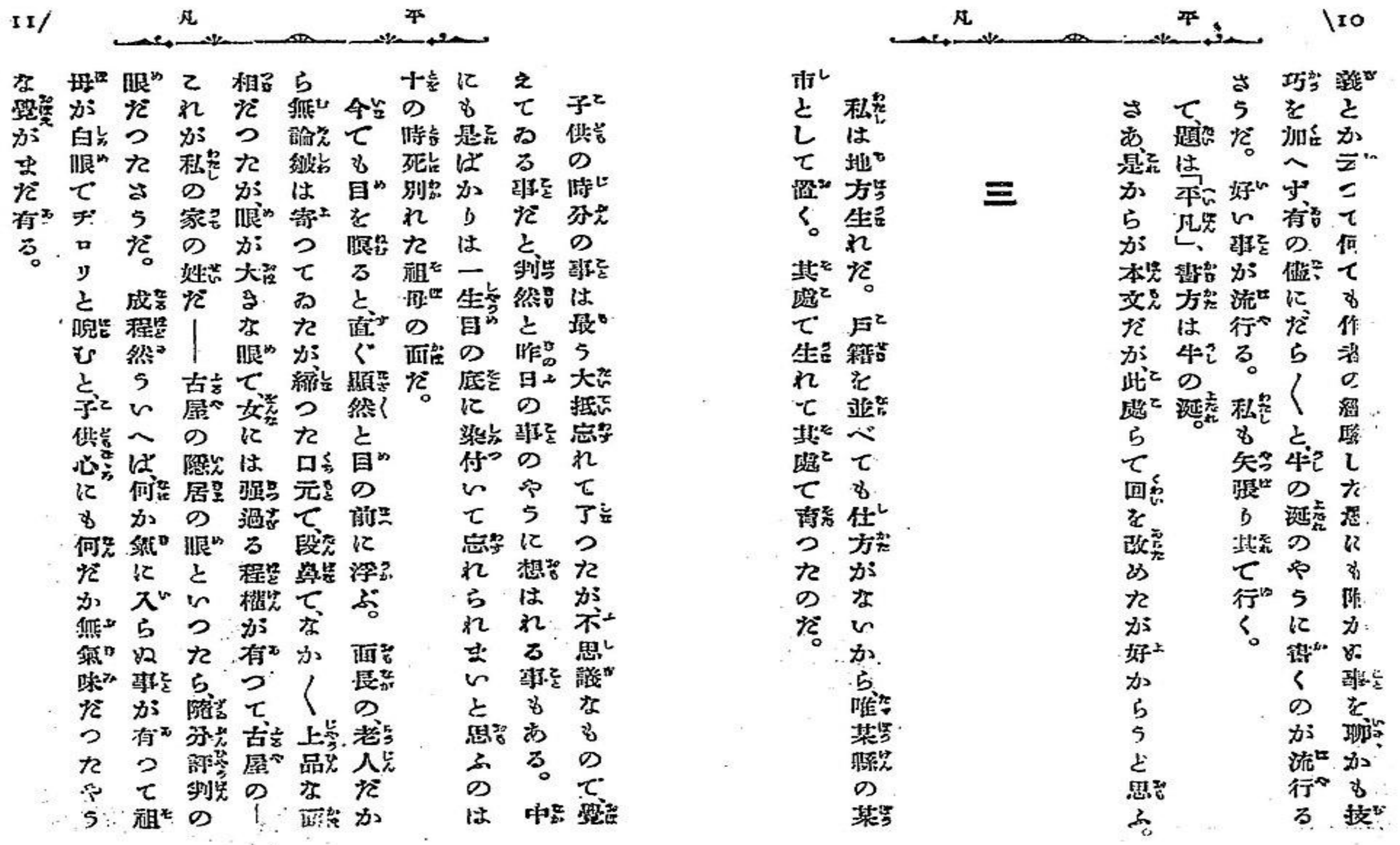
- What did it standardize ?
  - →CLI
  - →Scripting Interface
  - →Also many user level programs
    - →awk, echo, ed, …
  - →Required program-level services
    - →basic I/O: file, terminal, and network
  - →standard threading library API
    - →Pthreads
  - →Sockets
  - →Regular Expressions.
  - →Defined Portable Character Set (PCS = https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap06.html#tagtcjh_3)



howest hogeschool

Characters & Character encoding

- ASCII CODE (covers PCS) [ASCII table characters and symbols (ascii-code.com)]
  - →1963 (IEEE)
  - →0-127 values
  - →Encodes 128 characters/numbers/…
- Should be more than enough ?
- “Higher” symbols are listed in another Encoding Set (e.g. SJIS, EUC, …)
- ASCII is a subset of that Encoding Set
- UNICODE has become the standard
- UTF-32, UTF-16, UTF-8



平凡 10

議とか云つて何でも作者の經驗した愚にも附かない事を、聊かも技巧を加へず有の儘に、だら／＼と、牛の涎のやうに書くのが流行る。さうだ。好い事が流行る。私も矢張り其で行く。

で、題は「平凡」、書方は牛の涎。

さあ、是からが本文だが、此處らで回を改めたが好からうと思ふ。

三

私は地方生れだ。戸籍を並べても仕方がないから、唯某縣の某市として置く。其處で生れて其處で育つたのだ。

11 平凡

子供の時分の事は最う大抵忘れて了つたが、不思議なもので、覺えてゐる事だと、判然と昨日の事のやうに想はれる事もある。中にも是ばかりは一生目の底に染付いて忘れられまいと思ふのは十の時死別れた祖母の面だ。

今でも目を瞑ると、直ぐ顯然と目の前に浮ぶ。面長の、老人だから無論皺は寄つてゐたが、締つた口元で、段鼻で、なか／＼上品な面相だつたが、眼が大きな眼で、女には強過る程權が有つて、古屋の──これが私の家の姓だ──古屋の隱居の眼といつたら、隨分評判の眼だつたさうだ。成程然ういへば、何か氣に入らぬ事が有つて祖母が白眼でヂロリと睨むと、子供心にも何だか無氣味だつたやうな覺がまだ有る。

- ASCII : original English only text representation using 7 bits
  - Values between 0 to 127, 0x0 to 0x80, 0 to 1111111
- How about none-English
  - Values above will not fit into one byte
  - Solution
    - Fill in the first byte starting one 1’s ending with 0 to explain the number of bytes
    - Additional bytes will always start with 10

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0xxxxxxx			
U+0080	U+07FF	110xxxxx	10xxxxxx		
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
U+10000	[nb 2]U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

UTF-8 ?

ASCII ? 127 values so how many bytes ?
What about non-Ascii symbols ?

- Look up the code point of the symbol (= value to be stored binary)
- Find the number of bytes needed (see table below)
- Fill the bytes as specified
  - Byte 1 : show how many bytes will be needed in most significant bits
  - Byte 2 – x : start with ’10’ in most significant 2 bits
  - Byte 1 – x : fill with binary value

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0xxxxxxx			
U+0080	U+07FF	110xxxxx	10xxxxxx		
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
U+10000	[nb 2]U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

So Unicode is a representation of 4 character that represents the hexadecimal value

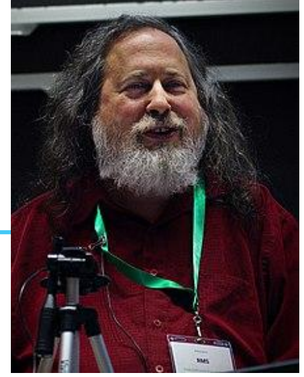
- Euro symbol has
  - Unicode 20AC
  - decimal 8364
  - Binary : 00100000.10101100, so you have 14 bits
  - How does 14 bits fits in x + 6 bits (7 or 5+6 or 4+2*6or 3+3*6)
  - Split up into 6 bit per byte starting at the end : 10.000010.101100
  - To fit this into utf-8 11100010.10000010 10101100
  - 11100010=E2 , 10000010=82 , 10101100=AC
  - encoded in utf8 it translates to following bytes e2 82 ac

UTF-8 Example : the € (Currency symbols - EU Vocabularies - Publications Office of the EU (europa.eu))

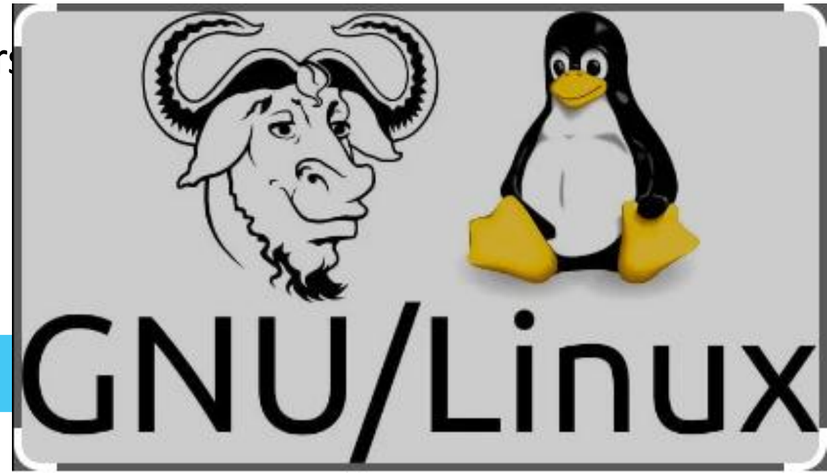
1. The Unicode code point for € is U+20AC (8364).
2. This code point lies between U+0800 and U+FFFF, so this will take three bytes to encode.
3. `20AC` becomes in binary `0010 0000 1010 1100`. The two leading zeros are added because a three-byte encoding needs exactly sixteen bits from the code point.
4. Because the encoding will be three bytes long, its leading byte starts with three 1s, then a 0 (`1110...`)
5. The four most significant bits of the code point are stored in the remaining low order four bits of this byte (`11100010`), leaving 12 bits of the code point yet to be encoded (`...0000 1010 1100`).
6. All continuation bytes contain exactly six bits from the code point. So the next six bits of the code point are stored in the low order six bits of the next byte, and `10` is stored in the high order two bits to mark it as a continuation byte (so `10000010`).
7. Finally the last six bits of the code point are stored in the low order six bits of the final byte, and again `10` is stored in the high order two bits (`10101100`).
8. All together, the three bytes look like this: `11100010 10000010 10101100` or E282AC in hex form.

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0xxxxxxx			
U+0080	U+07FF	110xxxxx	10xxxxxx		
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
U+10000	[nb 2]U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

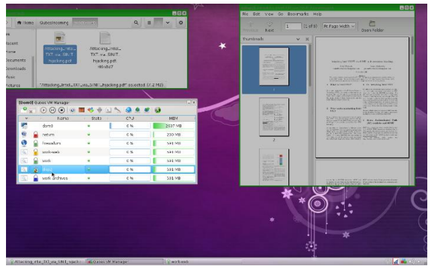
What’s GNU ?

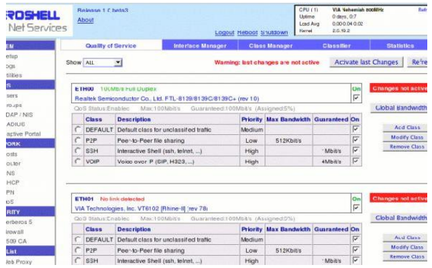


- While Posix is a portable interface, GNU ceates a free software to implement this interface
- Free to run a program for any purpose
- Freedom to study the mechanics of the program and modify it
- Freedom to redistribute copies
- freedom to improve and change modified versions for public use
- Rewrite “unix” to the highest standardization
- Linus Torvalds made the firs
  - →(GNU)LINUX !!
  - →1991
- OPEN SOURCE OS



A few Linux Distros you may (not) have come across (yet)

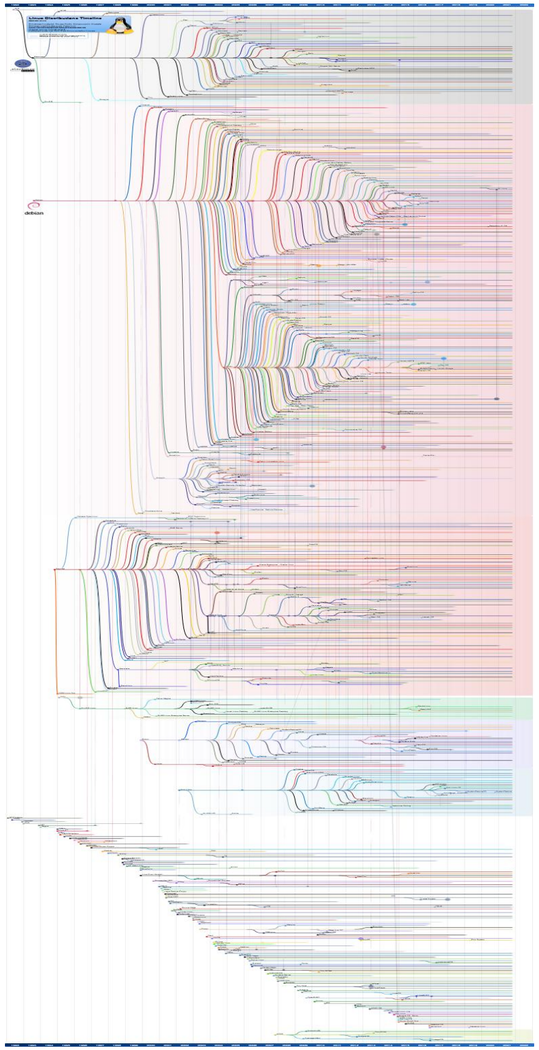




But there ARE others…

howest hogeschool

The Linux Family Tree (source : distrowatch)



Full resolution :
https://leho-howest.instructure.com/files/2537618/download?download_frd=1

History of Other OS's

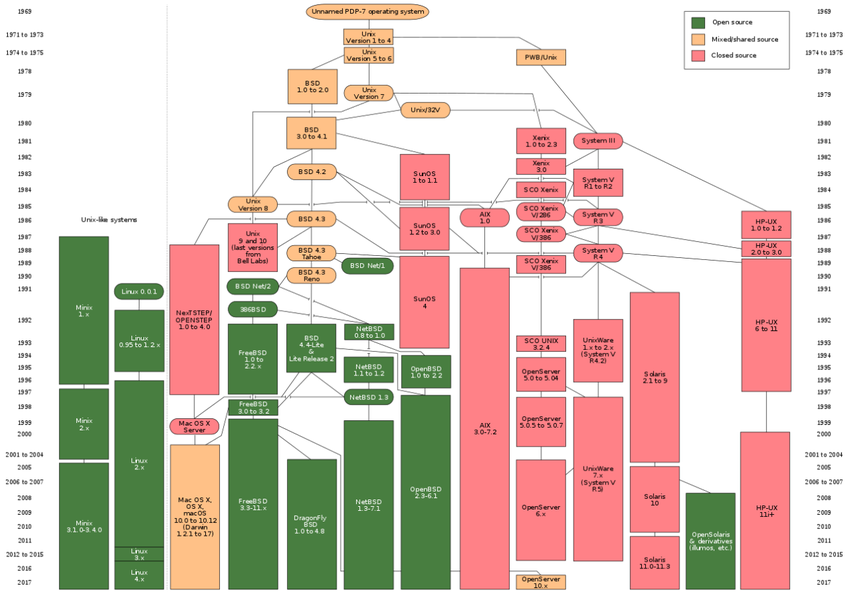
- **IBM 5100** is the first portable computer in **1975**
  - Now it starts going fast…
- **Apple 1 – 1976**
- **IBM PC – 1981** and used **MS-DOS**
- **Atari TOS (1985)**
- **Amiga OS (1985)**
- **MS-DOS** vs **Windows NT**
  - https://en.wikipedia.org/wiki/Windows_NT

History – Some extra's

Crash Course: https://www.youtube.com/watch?v=26QPDBe-NB8

History of Microsoft: https://www.youtube.com/watch?v=JmtPWvT1vp8

https://www.theguardian.com/technology/2014/oct/02/from-windows-1-to-windows-10-29-years-of-windows-evolution



https://en.wikipedia.org/wiki/Unix#/media/File:Unix_history-simple.svg

https://upload.wikimedia.org/wikipedia/commons/1/1b/Linux_Distribution_Timeline.svg

The entire course in 13 minutes aka. topics we will discuss

https://www.youtube.com/watch?v=-uleG_Vecis

howest hogeschool