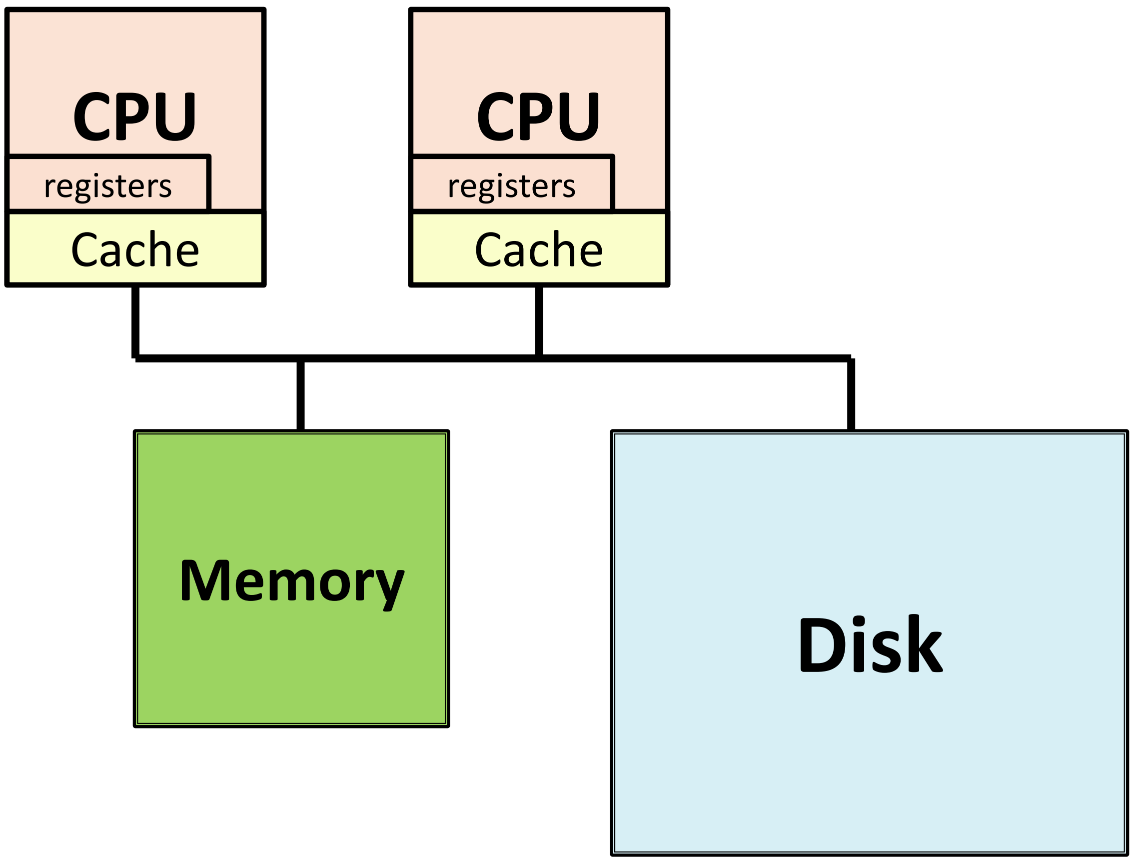
Operating Systems

1. Introduction

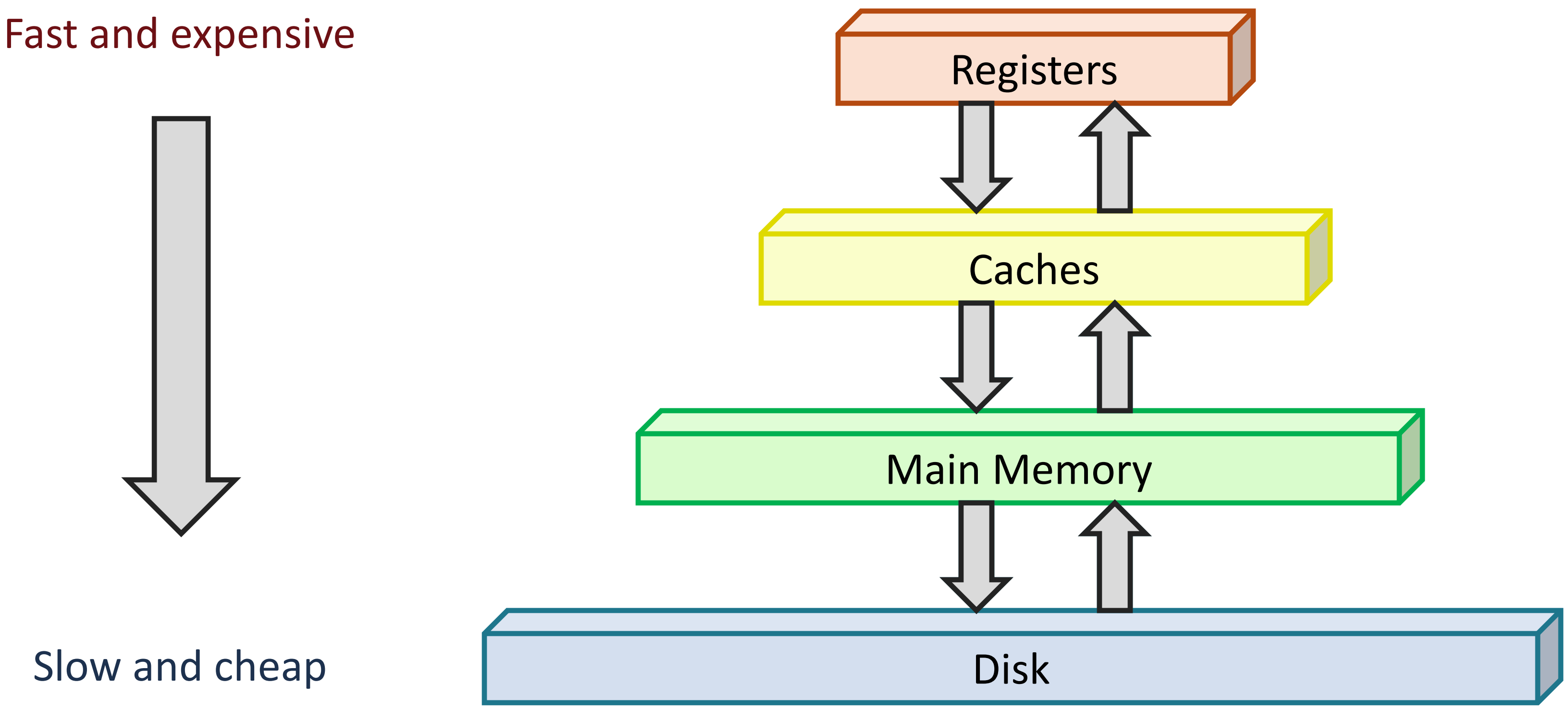
Pablo Prieto Torralbo

DEPARTMENT OF COMPUTER ENGINEERING AND ELECTRONICS

Remember...



Remember...



Fast and expensive
Slow and cheap
Registers
Caches
Main Memory
Disk

Remember…

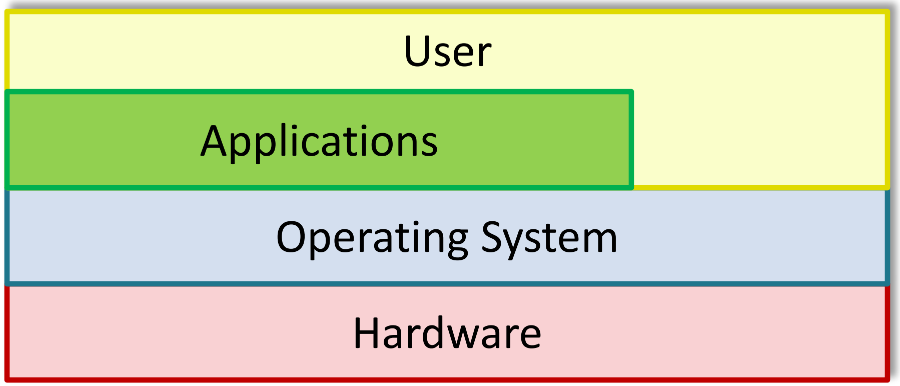
Level	1	2	3	4
Name	Register	Cache	Main Memory	Disk Storage
Size	<1KB	From 32KB to > 16MB	Tens of GB	Hundreds of GB to TB
Technology	Custom memory multiple ports (CMOS)	On-chip CMOS SRAM	CMOS DRAM	Electrical, Magnetic or Optical disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5000000
Managed by	Compiler	Hardware	Operating System	Operating System

Remember…

- **A running program does one very simple thing:
  it executes instructions (many million every second):**
  - The processor fetches an instruction from memory.
  - Decodes the instruction (figures out which instruction it is).
  - Executes it (does what it is supposed to do).
- **After an instruction is done, the processor moves on to the next one, and so on, and so on until the program completes.**
- **These are the basics of the Von Neumann model:**
  - Modern processors are quite a lot more complicated:
    - Multiple instructions at a time.
    - Out of Order execution…
    - … Not the topic of this course.

What is an Operating System?

- A program that acts as an intermediary between a user of a computer, the software and the computer hardware.



- A software layer to abstract away and manage details of hardware resources.
- A piece of software that makes the system easy to use.

Operating System Goals

- Execute user programs and make solving user problems easier.
- Make the computer system convenient to use.
- Use the computer hardware in an efficient manner.
- Control access to shared resources.

How does the OS provide this?

- **Resource Management:**
  - Share resources correctly.
- **An OS mediates programs’ access to hardware resources:**
  - Computation (CPU).
  - Volatile storage (memory) and persistent storage (disk, etc.).
  - Network communications (TCP/IP stacks, Ethernet cards, etc.).
  - Input/output devices (keyboard, display, sound card, etc.).
- **Advantages of OS providing resource management?:**
  - Protect applications from one another.
  - Provide efficient use/access to resources:
    - Cost, time, energy…
  - Provide fair access to resources.
- **Challenges:**
  - What are the correct mechanisms?
  - What are the correct policies?

How does the OS provide this?

- **Abstraction:**
  - The OS takes hardware resources and transforms them into logical versions and provides well-defined interfaces for those resources (**system calls**).
- **What is a resource?:**
  - Anything valuable (e.g. CPU, memory, disk…).
- **What abstraction does modern OS typically provide for each resource?:**
  - **CPU:** process and/or threads.
  - **Memory:** address space.
  - **Disk:** files.
  - **Network:** sockets.
- **Advantages of OS providing abstraction?:**
  - Allow applications to reuse common facilities.
  - Make different devices look the same.
  - Provide higher-level or more useful functionality.

Why bother with an OS?

- **Application benefits:**
  - Programming simplicity:
    - See high-level abstractions (files) instead of low-level hardware details (device registers).
    - Abstractions are reusable across many programs.
  - Portability (across machine configurations or architectures):
    - Device independence (3Com card or Intel card).
- **User benefits:**
  - Safety:
    - Program “sees” its own machine (virtualization).
    - OS protects programs from each other.
    - OS fairly distributes resources across programs.
  - Efficiency (cost and speed):
    - Share one computer across many users.
    - Concurrent execution of multiple programs.
- **Behavior of the OS impacts the entire machine.**

Three Pieces

- **How to cover all the topics relevant to operating systems? - Three pieces:**
  - **Virtualization:**
    - Make each application believe it has computer resources to itself.
    - Take physical hardware and make a software version that is sharable and easier to use.
    - Example:
      - **CPU**: multiple programs can run “at the same time”.
      - **Memory**: programs see a linear range of addresses.
  - **Concurrency:**
    - Events are occurring simultaneously and may interact with one another.
    - OS must be able to handle concurrent events, maintaining correctness.
    - Easy case: Concurrency from **independent** processes.
    - Tricky case: Concurrency from **interacting** processes.
  - **Persistence:**
    - Keep data safe from crashes/reboots.

We will follow:

Remzi H. Arpaci-Dusseau & Andrea C. Arpaci-Dusseau: *«Operating Systems: Three Easy Pieces»*. http://www.ostep.org

Virtualizing the CPU

- **Let’s run some programs:**
  - cpu.c.

During the course, all programming is in **C**.

Virtualizing the CPU

- **Even though we have only one processor, somehow all four of these programs seems to be running at the same time.**
- **The Operating System (with some help from hardware) provides the illusion that the system has a large number of virtual CPUs (at least one for each process).**
- **This is what we call virtualizing the CPU.**
- **If two or more programs want to run at a particular time, which one should run?:**
  - **Policy** (what/when will be done).
  - **Mechanisms** (how to do it).

Virtualizing Memory

- **Physical memory is very simple:**
  - Memory is just an array of bytes.
  - One must specify an address to be able to access the data stored there.
  - To write (update) memory, one must specify also the data to be written.
- **Memory is accessed all the time while a program runs:**
  - A program keeps all of its data structures and instructions in memory.
- **Let's run some programs:**
  - mem.c.

Virtualizing Memory

- **Running multiple instances (disabling Address Space Randomization):**
  - Seems each running program has allocated memory at the same address.
  - Each seems to be updating the value independently.
  - Seems each program has its own private memory, instead of sharing the physical memory.

- **OS is virtualizing memory:**
  - Each process accesses its own **private virtual address space**, which the OS maps onto physical memory.
  - Each running program is mapped separately.
  - One running program does not affect the address space of other running programs (or the OS itself).
  - This is what we call **Virtual Memory**.

Concurrency

- **Processes are devoted their own resources:**
  - Program Counter.
  - Processor Registers.
  - Address Space :
    - Code.
    - Heap (data).
    - Stack (Stack Pointer).
  - User ID and state flags.
  - OS Resources (files, network connections…).

Concurrency

- **What if a program (e.g.: web server) wants to use multiple processors (e.g.: handle multiple requests concurrently).**

- **There is a "light" kind of process: Thread:**
  - Threads share:
    - Address Space:
      - Code.
      - Heap (data).
    - Privileges.
    - OS resources.
  - Each thread has its own:
    - Program Counter.
    - Processor Registers.
    - Stack Pointer.

Concurrency

- **Sharing resources opens new problems.**

- **Let's run some programs:**
  - threads.v0.c.

Concurrency

- **With higher values for loops, final value differs from the expected one:**
  - Also different values on different runs.

- **Instructions are executed one at a time:**
  - The key part of the program (counter increment) takes three instructions:
    - Load from memory to register.
    - Increment register value.
    - Store back into memory.
  - What if the other thread executes in between?

Persistence

- **Information lifetime is longer than lifetime of one process.**
- **Machine may be rebooted, lose power or crash unexpectedly.**
- **We need hardware and software to store data persistently:**
  - Hardware: I/O device (hard drive, solid-state drives).
  - Software (OS): File system.
- **OS does not create a private-virtualized disk for each application. Rather, information is usually shared between processes/users in files.**
- **Example:**
  - You use an editor to write a C program → Then you compile the source code into an executable → Then you run the executable (maybe another user does one of these steps).

Design Goals

- **Provide Abstractions to make the system easy to use:**
  - OS as a standard library. System calls.
- **High Performance:**
  - Minimize the overhead of the OS (extra time, extra space…).
  - Trade-off: Virtualization/easy to use vs. Performance. Perfection is not always attainable.
- **Ensure some Fairness:**
  - Avoid starvation. New trade-off.
- **Provide Protection between applications as well as the OS:**
  - Many programs running at the same time, but not harming others.
  - Isolation is the key.
- **Reliability:**
  - If the OS fails, all applications running on the system fail as well.
- **Other more specific goals:**
  - Energy efficiency (green world).
  - Security (interconnected world).
  - Mobility (small devices...).
- **Each system considers some goals more important than others.**

History

- **Operating Systems were born and evolve out of need.**
- **In the 1950s there are scarcely tens of computers in the world:**
  - “I think there is a world market for maybe five computers”:
    - Even though it is probably a fallacy attributed to T.J Watson (IBM 1943), it reflects the computer situation.
- **Computers were quite expensive, and machine time was more valuable than person time...:**
  - IBM 7090: $2.9 million.
  - Resource Management and efficiency were important.

History - Mainframes

NSA: IBM 7950 HARVEST

NASA: IBM 7090

History

- **Early days: OS just as a Library:**
  - Instead of having each programmer write low-level I/O handling code, the OS provides such APIs (Application Program Interface).
  - Usually, on a mainframe, only one program at a time controlled by a human operator:
    - Human OS (e.g. scheduler) → Be nice with the operator.
  - Computer cannot be interactive:
    - Too expensive having a user sit in front of a computer, most of the time idle.

History

- **Automatic mechanisms need Protection:**
  - OS code is special → control of common devices:
    - File system → privacy.
  - System call (Atlas computing systems) instead of libraries:
    - Privilege instructions executed by the OS (kernel).
    - Requires hardware privilege levels: kernel mode as opposed to user mode.
    - E.g.: a program wants to initiate an I/O request to the disk:
      - 1. The program issues a **system call**.
      - 2. Hardware special instruction (**trap**) transfers the control to a pre-specified **trap handler** and raises privilege level to **kernel mode**.
      - 3. OS now has full access to hardware and does the required service.
      - 4. OS passes control back to the user via special **return-from-trap** instruction while lowering the privilege level to **user mode**.

History – mini computers

IBM System34

DEC PDP-8

Nokkia Mikko 3

History

- **Multiprogramming Era:**
  - Computers more affordable (minicomputer):
    - → More people working on computer systems.
    - → New ideas (multiprogramming).
  - Make better use of machine resources:
    - Instead of one job at a time, the OS loads a number of jobs into memory and switches between them.
    - Particularly important due to I/O being slow.
  - Memory protection became important.
  - Concurrency issues.

- **UNIX operating system (Ken Thompson and Dennis Ritchie) at Bell Labs:**
  - Take ideas from Multics (mainly), TENEX and Berkeley Time-Sharing System but made them simpler.

History – Microcomputers

Amstrad PC 1512

Asus Zenfone 6

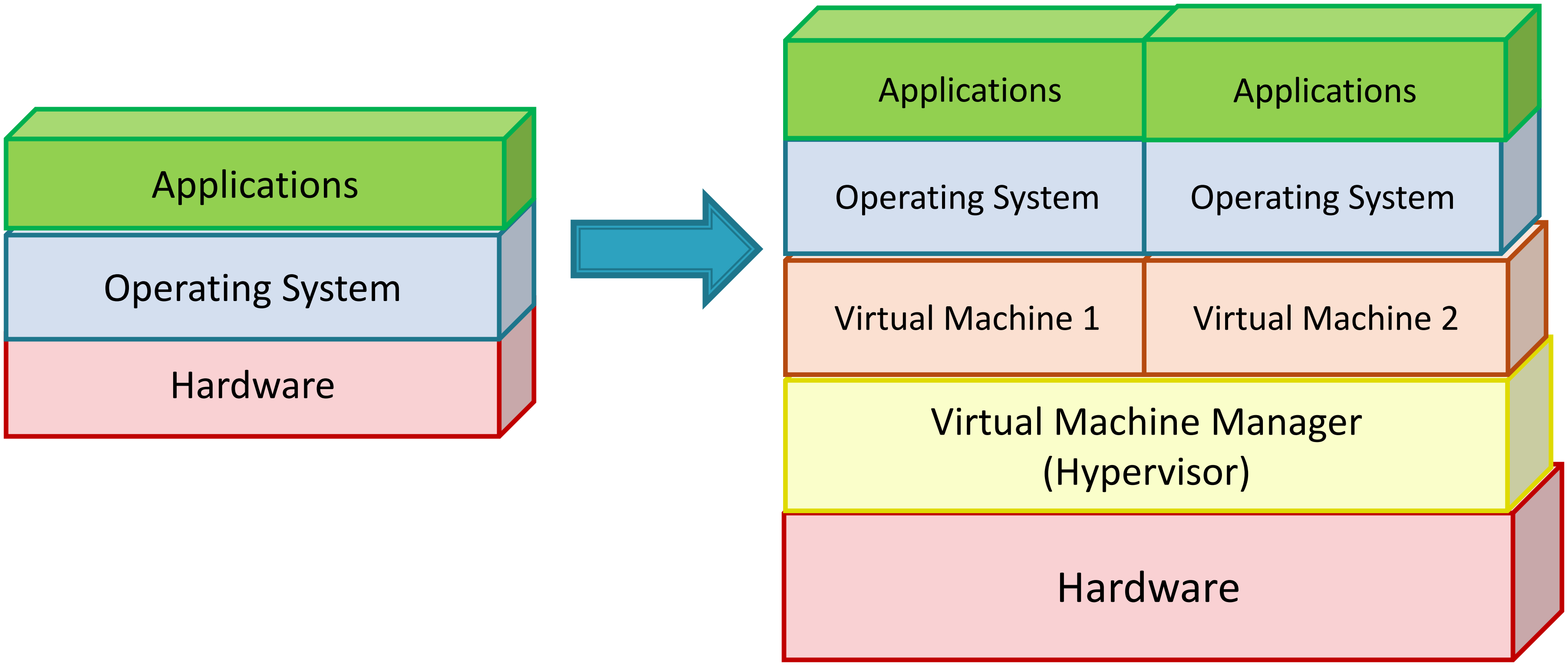
Macbook Air

Google glass

History

- **Modern Era:**
  - Cheaper and faster machine: Personal Computer (PC):
    - Led by Apple early machines and the IBM PC.
    - One machine per desktop.
  - OS at first leap backwards:
    - DOS (Disk Operative System from MS) didn't support memory protection.
    - First versions of Mac OS could make processes get stuck in an infinite loop.
  - Nowadays OS include features expected in a mature system:
    - Mac OS X (with UNIX at its core).
    - Windows, starting in particular with windows NT.
  - Even cell phones run operating systems (such as Linux).
  - Nowadays Operating Systems continue to develop, providing more features and making modern systems better and easier for users and applications.

History



- **Cloud Computing:**
  - Amazon EC2.
  - Microsoft Azure.
  - Google Compute Engine.

History - UNIX

- **Originally Unics (Uniplexed Information and Computing System) influenced by the mainframe OS Multics from MIT.**
- **Developed at Bell Labs by Ken Thompson (and Dennis Ritchie) on a PDP-7 computer.**
- **Main ideas:**
  - Multiple jobs at a time (concept of process).
  - Multiple users. The shell (command-line interpreter).
  - File system. Used as inter-process communication.
  - Small powerful programs that could be connected to form larger workflows:
    - The shell includes primitives such as pipes to enable meta-level programming.

History - UNIX

- **The UNIX environment was friendly to programmers and developers:**
  - Provides a compiler for the new C programming language.
  - Also provides a text editor.
- **Originally written in Assembler, kernel was re-written in C in 1972.**
- **The authors gave out copies of the OS to anyone who asked (including Universities, Companies and Government):**
  - An early form of open-source software.
- **The accessibility and readability of the code leads others to play with the kernel and add new features:**
  - Berkeley System Distribution (BSD) by a group led by Bill Joy (who later founded Sun Microsystems):
    - Advanced virtual memory, file system and network subsystem.

History - UNIX

- **Many companies have their own variants:**
  - SunOS from Sun Microsystems.
  - AIX from IBM.
  - HPUX from HP.
  - IRIX from SGI.
  - OS X from Apple…

- **UNIX almost disappears:**
  - AT&T/Bell expensive license.
  - Companies try to make profit from it.
  - Windows was introduced.

History - Linux

- **A young Finnish student named Linus Torvalds writes his own version of UNIX:**
  - Improvement of Minix (Tanenbaum academic OS).
  - Borrows heavily the ideas and principles of UNIX.
  - But not the code, completely rewritten.
- **The code became public and many others around the world helped → GNU/Linux was born:**
  - As well as the modern open-source software movement.
- **In the internet era, most companies (Google, Amazon, Facebook…) chose Linux as it is free and easily modified to suit their needs.**
- **Linux also gets into smart phones via Android.**