Operating Systems: Introduction

Fabrice BOISSIER<fabrice.boissier@epita.fr>

2021-10-12

References

- Linux – Programmation système et réseau *[fr]*
  Joëlle Delacroix

- Advanced Programming in the UNIX Environment *[en]*
  W. Richard Stevens

- Modern Operating Systems *[en/fr]*
  Andrew S. Tanenbaum, Herbert Bos

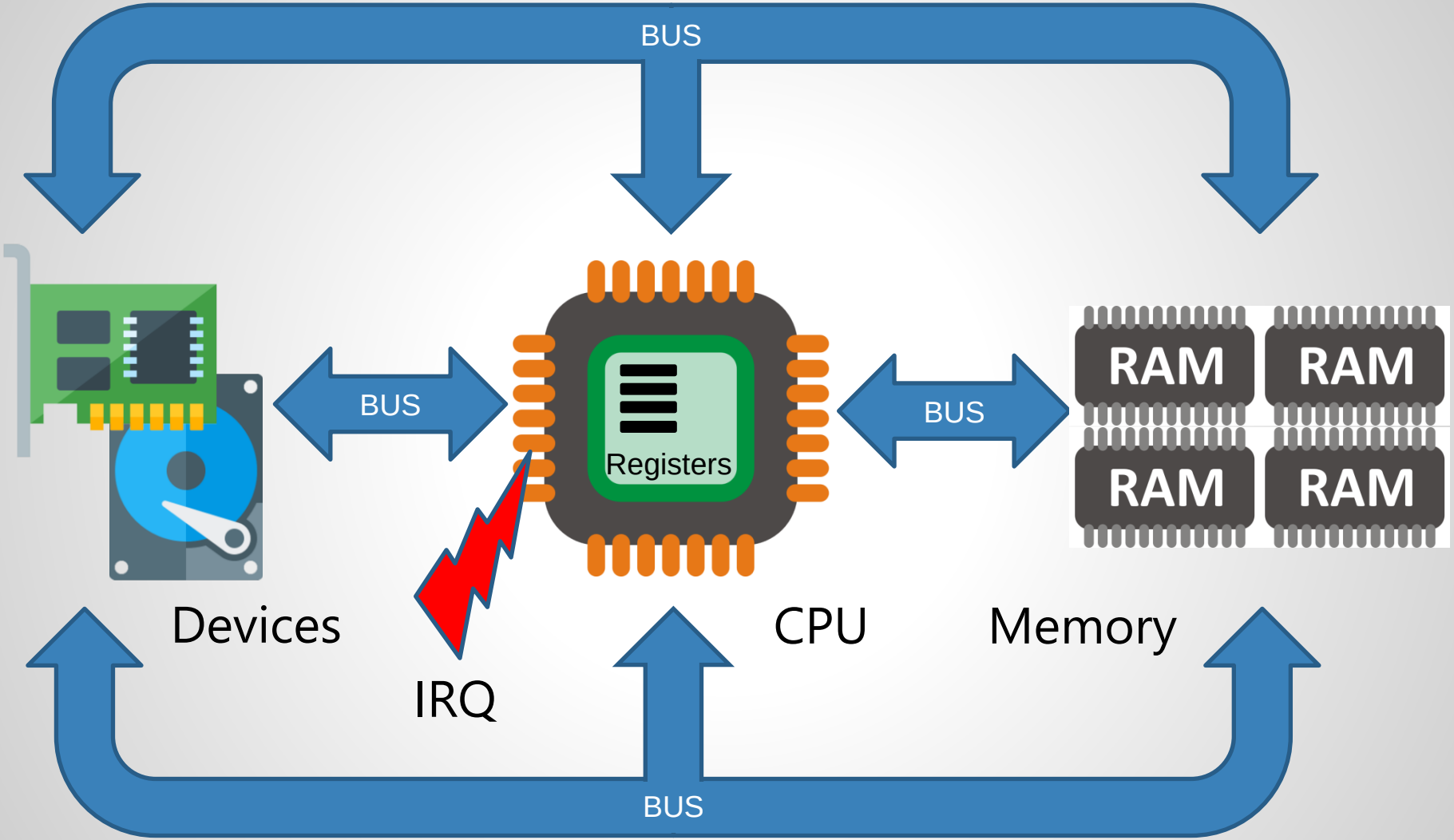
- Operating Systems Concepts *[en]*
  Abraham Silberschatz, Peter Baer Galvin, Greg Gagne

Outline

- Basic architecture concepts required
- What is an OS
  - Why and where do we need OS
- Context of computing
  - History
  - Machine taxonomy
  - OS taxonomy
- What is inside an OS
  - Kernel/Services/Applications
  - Main concepts

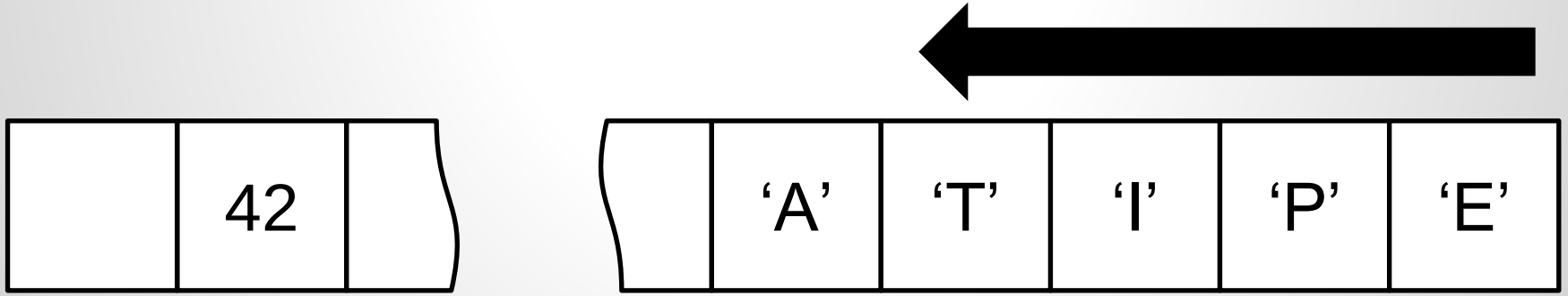
Basic architecture concepts required

- CPU: central processing unit
  - *Register*: variable of the CPU
- Device: hardware
- Interruption: way to handle an event for the CPU
  - *Hardware interrupt* or *IRQ*: Interruption from hardware
  - *Software interrupt*: Interruption from software


BUS
BUS
BUS
Registers
RAM
RAM
RAM
RAM
Devices
IRQ
CPU
Memory
BUS

Basic architecture concepts required

- Memory: huge array of cells containing data
  - Each cell stores a specific size of data
- Stack: specific part of memory that pushes/pulls data
  - Usually at the end of the memory, growing up toward beginning



Basic architecture concepts required

- Protection rings:
  - Model of protection
  - Levels/Layers of autorisations
  - Each level has specific permissions

  - Intel example: 4 levels
    ring 0 = most privileged mode
    …
    ring 3 = least privileged mode (regular mode)

Basic architecture concepts required

- Ring 0 - Supervisor mode - Privileged mode:
  - Context in which the CPU allows more actions
  - *Allows access to restricted memory addresses*
    (you might not want that another program erases your mallocs…)
  - *Allows execution of restricted instructions*
    (you might not want a HALT to happen anytime…)
  - …

What is an Operating System?

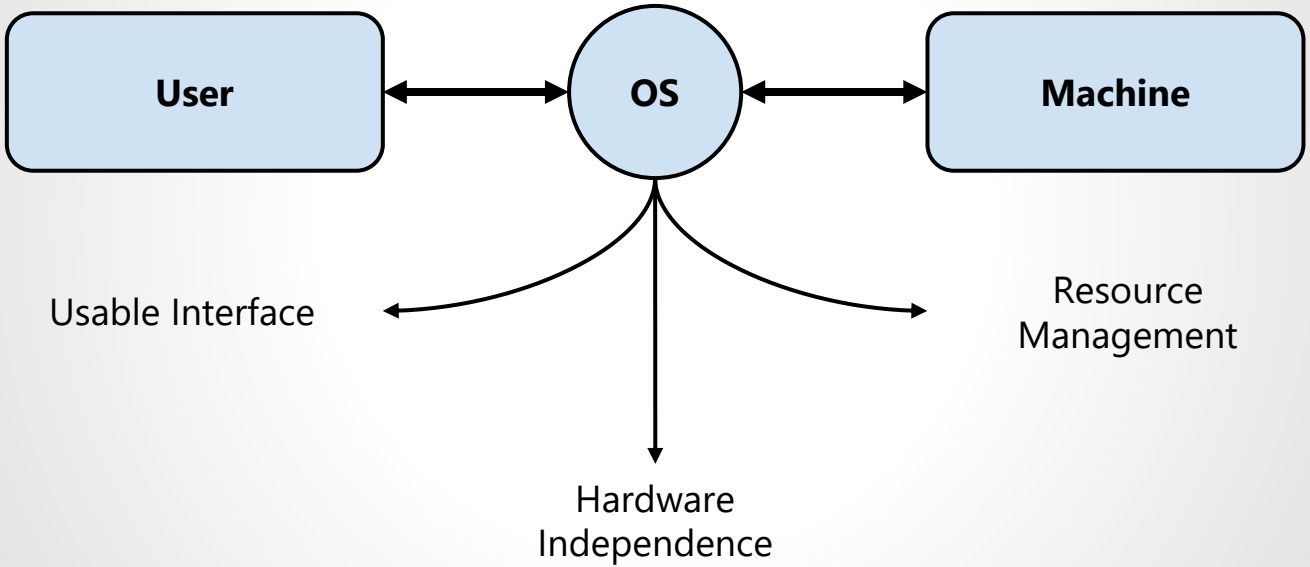
Operating systems perform two essentially unrelated functions: providing application programmers (and application programs, naturally) a clean abstract set of resources instead of the messy hardware ones and managing these hardware resources.

-- Tanenbaum & al.

An operating system is software that manages a computer's hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware.

-- Silberschatz & al.

Operating System: Another definition



Operating System: Another definition

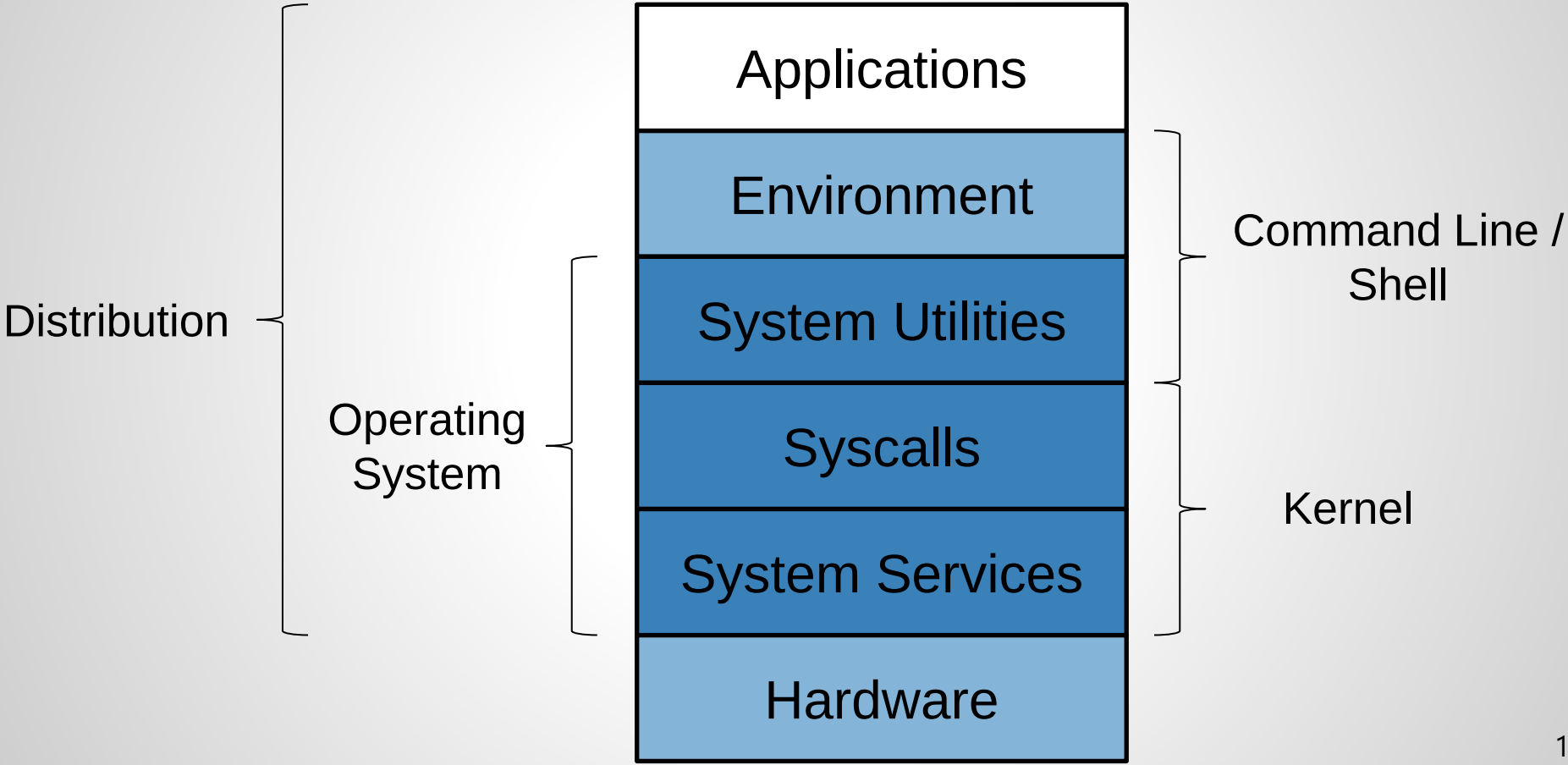
Most used definition:

- Resource allocation
- Resource management

Other:

- Second software layer on bare hardware
  - Firmwares on 1st layer (BIOS, UEFI, SoC, embedded, …)
- Kernel + Utilities above the machine

Operating System: Another definition



Where do we need an Operating System?

- Computer
- Mobile Phone
- VOIP Phone
- Sim cards
- Printer
- Car
- ...

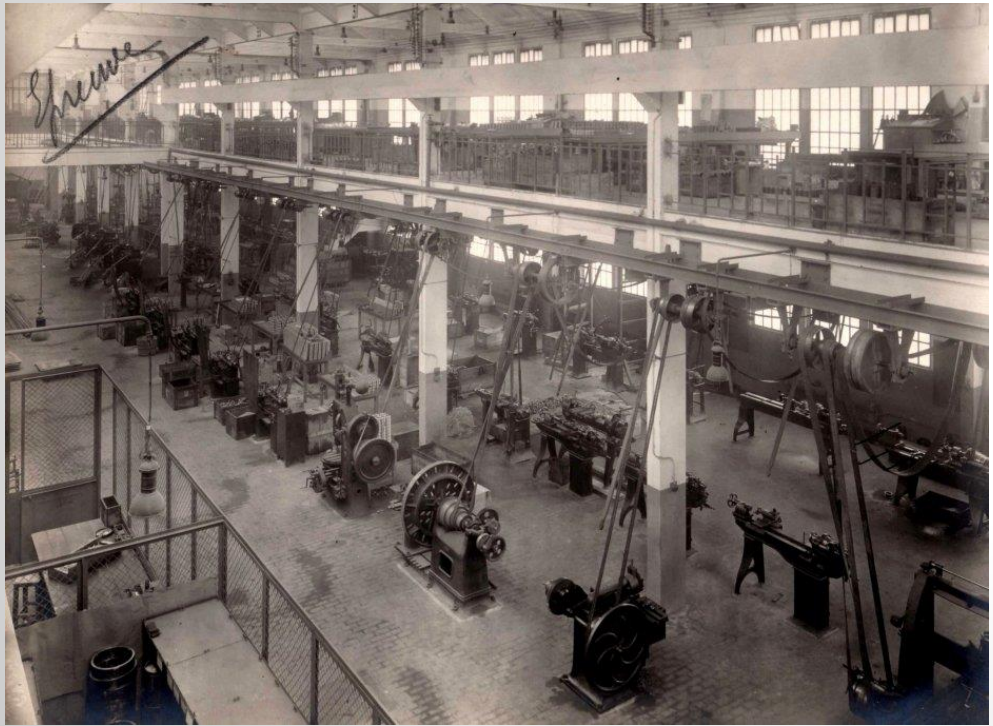
History

- Generations
- Taxonomy of machines and OS

- http://www.computerhistory.org/timeline/computers/

Before/Without computers: Machines

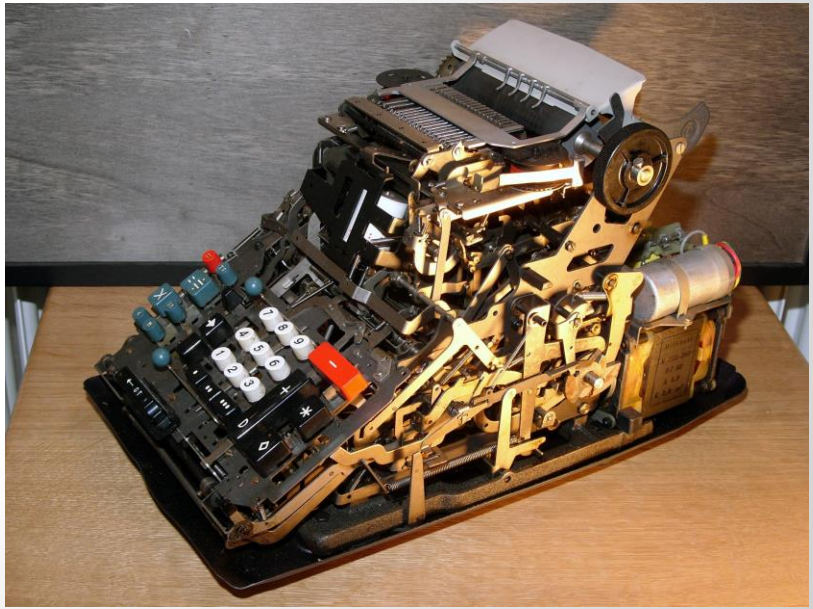
- Manual tasks
- Mechanical (gears, timing belts, …)
- Electromechanical



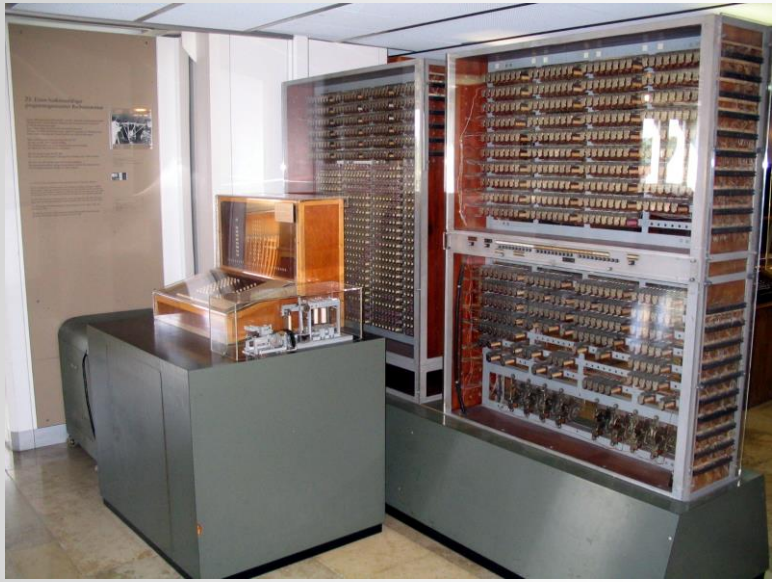
(Watch « Monsieur Patron » clip: *everything* is already here during the 50's-60's, but in mechanical version…)

Don't forget:

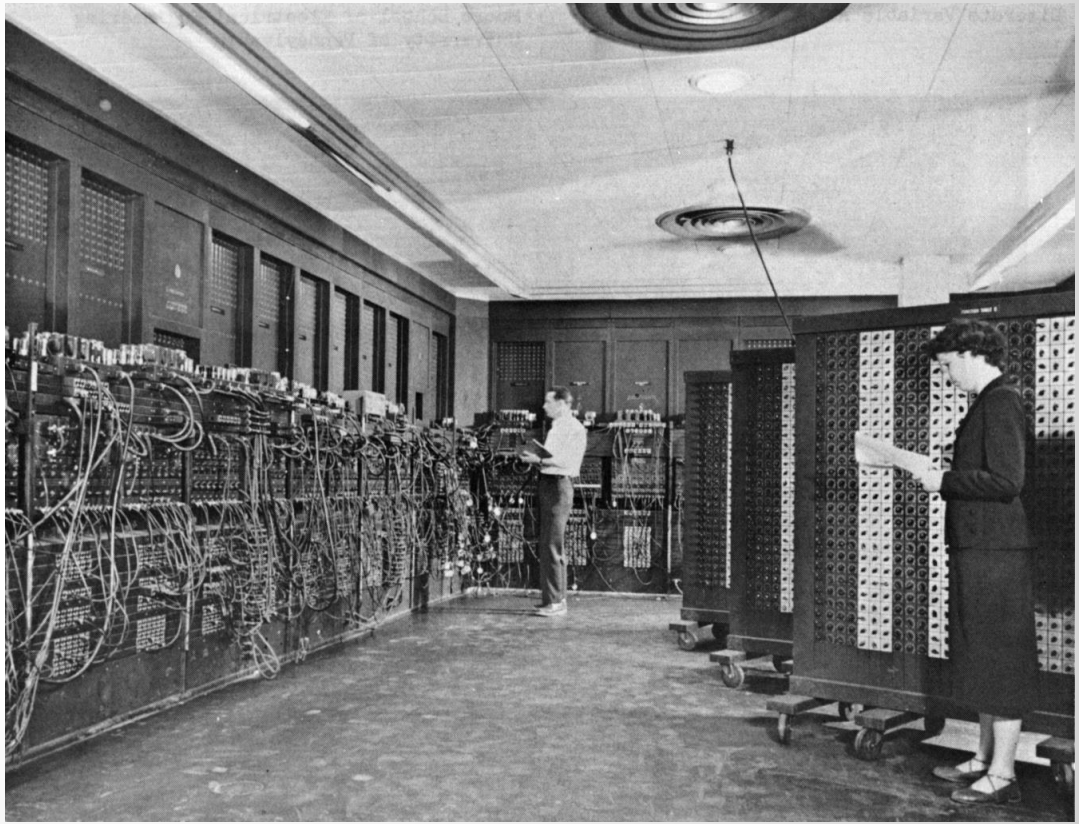
Initially, IT resolved business problems



First Generation (1945-55)

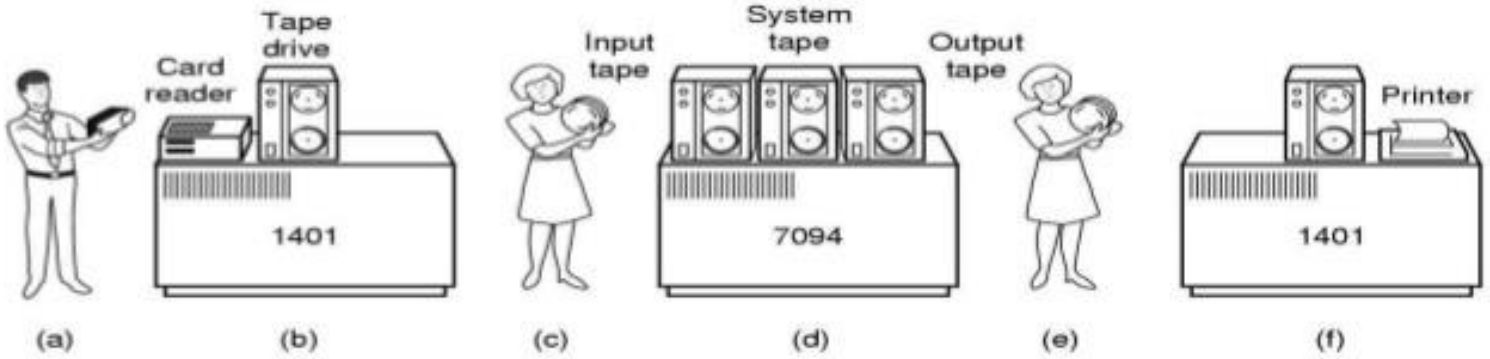


You « code » your program directly by linking/unlinking boxes each of which executes a single function (full of electric relays + ferrite toroid memory)



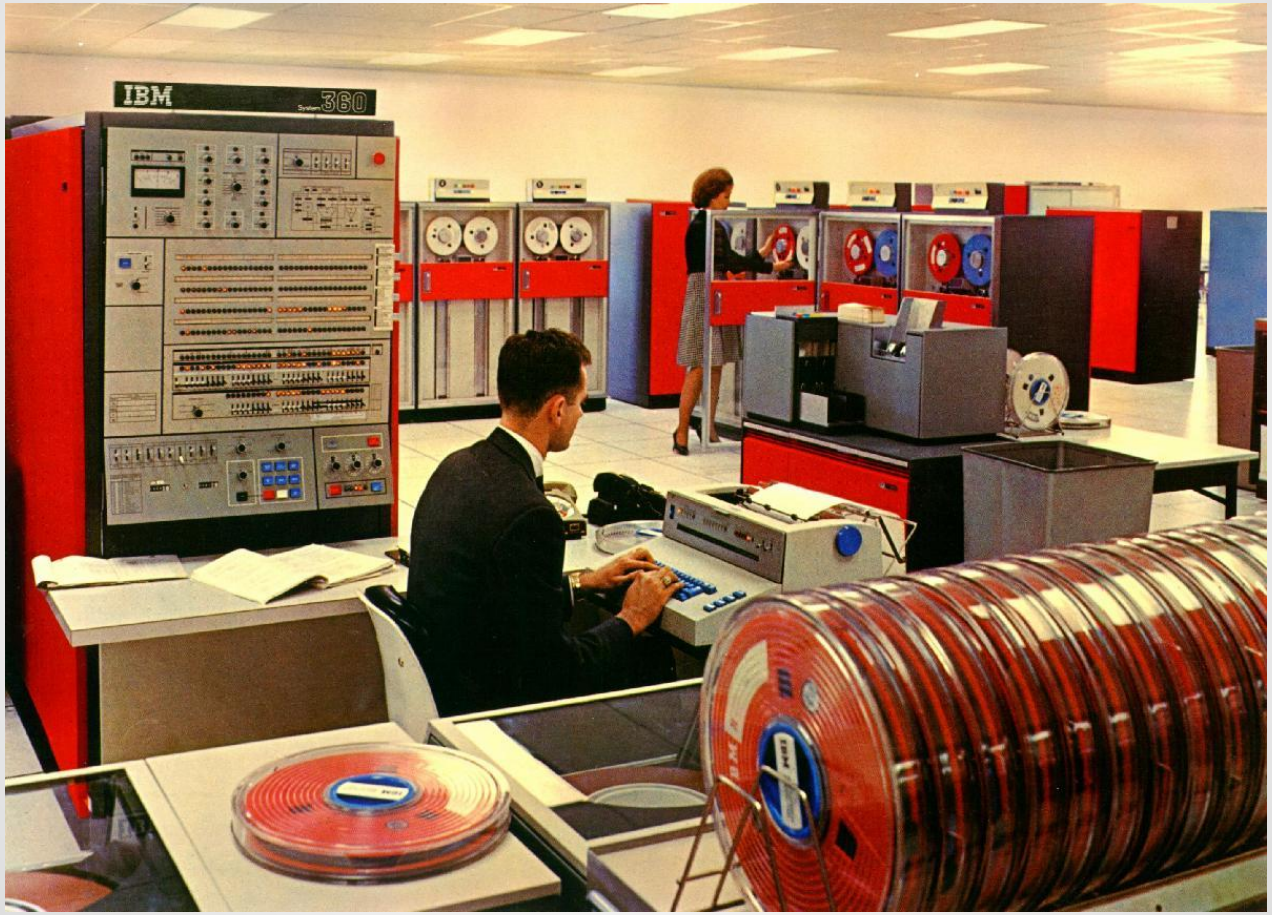
Second Generation (1955-65)

Evolution of Operating Systems (1)



Early batch system
- bring cards to 1401
- read cards to tape
- put tape on 7094 which does computing
- put tape on 1401 which prints output

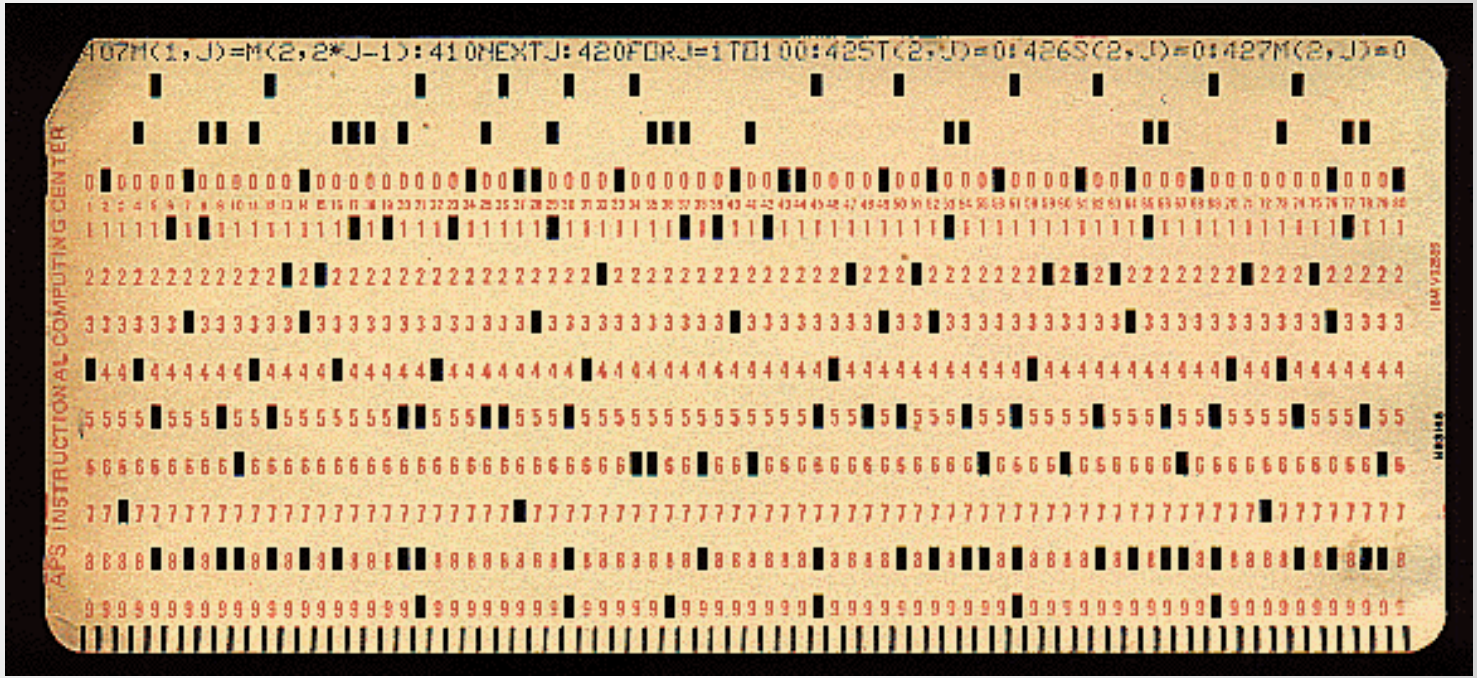
Second Generation (1955-65)



Second Generation (1955-65)

Second Generation (1955-65)

- Punch card contains the « code », and eventually the « data » (and mainly the launcher/script)
- (E)BCDIC: character encoding



Second Generation (1955-65)

- Machines: Mainframes
  - Each « huge » machine has a precise role:
    Calculate, read punch card, print results, store on magnetic tape

- IBM example:

Machine: System 360 (later: S/370, S/380, S/390, zArch)

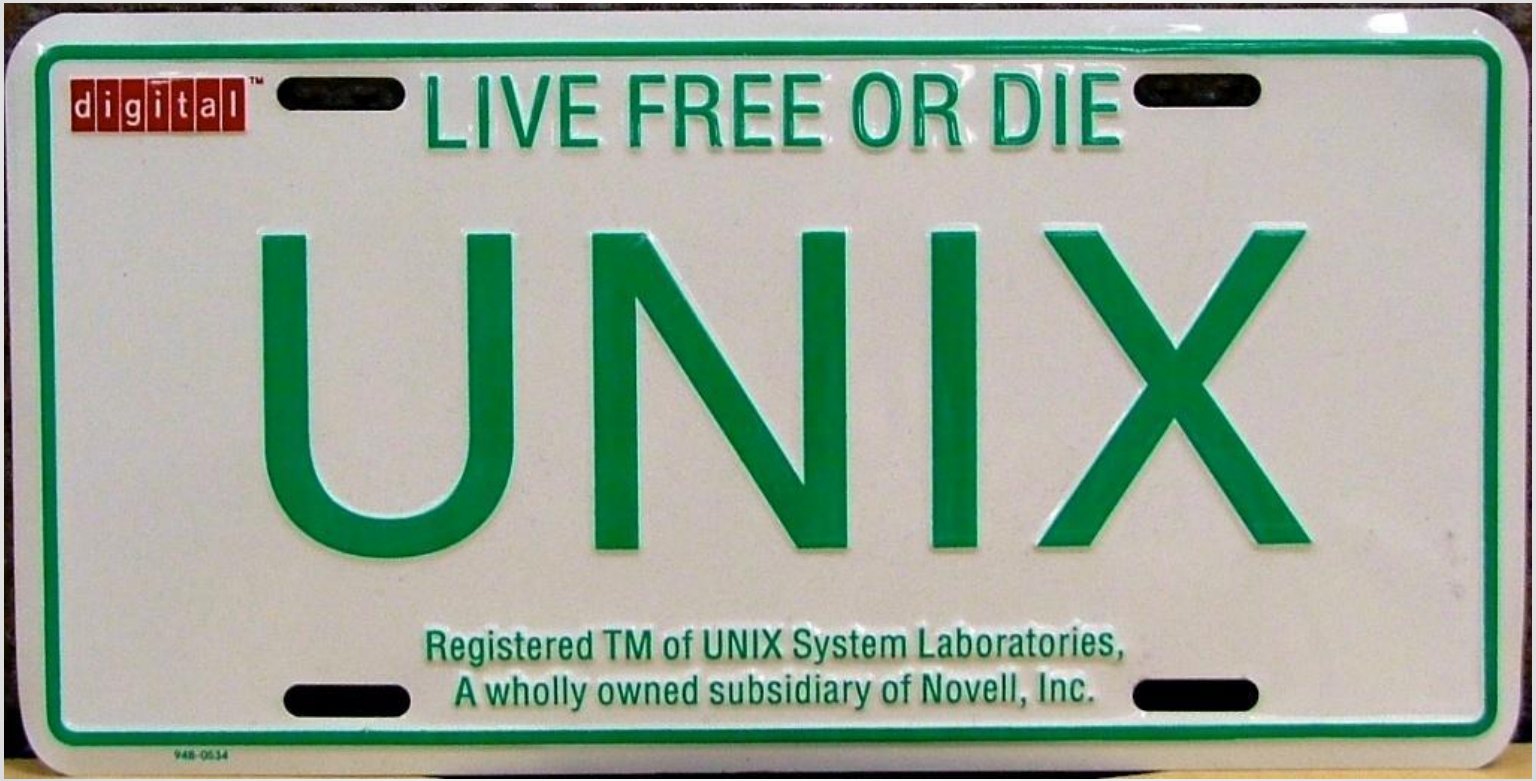
OS: OS/360 (later: OS/370, MVS, z/OS)

Programs compiled for OS/360 are still compatible nowadays

- Other: Burroughs (B5000), General Electrics (GE-635), …

Third Generation (1965-1980)

Third Generation (1965-1980)



Third Generation (1965-1980)

- Machines: Minis *(future « servers »)*
  - Smaller furnitures than mainframes
  - Affordable for middle and somewhat small companies
  - Beginning of the « invasion » of computers in the companies

- Examples:
  DEC (VAX & PDP), CII (Mitra 15), Télémécanique (SOLAR 16), HP (HP-3000), …

- OS: a lot… VMS, and obviously UNIX

Fourth Generation (1980 - …)

Fourth Generation (1980 - …)

- Machines: Personal Computer
  - Very small machines affordable by consumers
  - Very simple at the beginning
  - Everybody (rich enough) can own a computer

- Examples:
- zx80, VIC-20, Atari ST, …
- …and obviously the IBM 5150 / IBM PC

- Tons of OS… + some UNIX, CP/M, Macintosh, MS-DOS

Fourth or fifth generation?...

OS taxonomy?

It is difficult to split an operating system into a single category. Historically, each specific task has a specific operating system tailored for it. Now the barriers between the different workloads are blurred.

Each company tried its own concept. If a company succeeded, everybody copied it.

History: OS for a specific workload

- Mainframe
- Mini/Server
- Multiprocessor
- Personal Computer
- Embedded (and Mobile)
- Real-Time (Soft or Hard)
- Smart Card

History: Why that much differences?

- Computing power was limited
- For each workload a different way to handles tasks
  - Single-Task, Single-User
  - Multi-Task, Single-User
  - Multi-Task, Multi-User
- More computing power, more complexity forces OS designers and vendor to simplify and reuse more components

History: What is the actual status

- General purpose Operating Systems:
  - Desktop (MS Windows, macOS, Linux, Chrome OS)
  - Server (Linux, MS Windows Server, few UNIX)
  - Mainframe (z/OS, zLinux, GCOS)
  - Multiprocessor (most of the general cpus we find are now multi-core at least)
  - Embedded (Linux or specific OS on multiple kind of devices, cars, cameras, printers… Android or iOS for mobile and smartphones)

- Real Time Operating Systems
  - Hard Real-Time
  - Soft Real-Time

Desktop

- Application driven
- Multimedia
  - Screen
  - Graphic acceleration
  - Input devices (keyboard, mouses, …)
  - Multiple kind of devices hooked on it
- *Interaction with the user*
- Windows, macOS, Linux *(tons of distributions…)*

Server

Server Operating Systems

- UNIX-like & Linux
  - Certification for being « UNIX compliant »
  - Available on nearly « every » CPU architecture possible
  - Linux is not a UNIX… it is a project recode from scratch

- Windows Server
  - Very user friendly thanks to a graphical user interface (GUI)
  - Has a full console version since 2010~2012

- IBM i *(formerly known as OS/400)*
  - Runs on « IBM i » machines (previously known as AS/400)
  - Absolute abstraction with the hardware

Mainframe

Mainframe Operating System

- z/OS (IBM), zLinux (IBM), z/VM (IBM), GCOS (Bull/Atos)

- IBM machine: z15
- z/OS *(formerly known as MVS)*
  - Look'n feel of the OS/360, but with latest upgrades (POSIX compatible)
- zLinux
  - Linux working on a giant machine

- Atos/Bull machine : BullSequana M Series
- GCOS 7 & 8

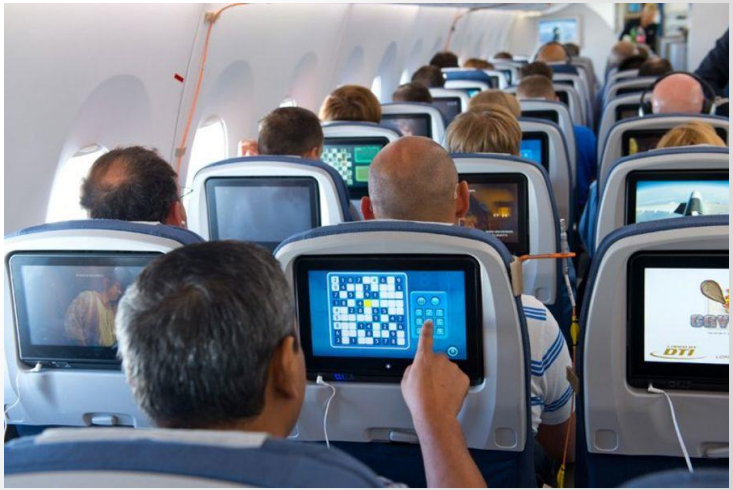
Specific case: Supercomputers

Specific case: Supercomputers / HPC

- Not really « one » computer like a mainframe
  - Well, back in time, it was « one » computer (Cray, …)
- Combination of a lot of small servers (« nodes »)
  - Each will make calculus
- Dedicated to calculate
  - Code a program, launch it, wait a long time for the result…
  - …like « calculators » and mainframes in 1st and 2nd generations
- Currently runs specific OS
  - Dedicated Linux for HPC (RHCS) and/or softwares (slurm)

Specific case: Embedded Systems / Mobile & IoT

- (Very) Small factors (IoT, Raspberry pi, …)
- Mobile (Tablets, GPS, …) and smartphones
  - Android
  - iOS
  - Windows: Windows IoT, Windows Embedded family (WinCE, …)



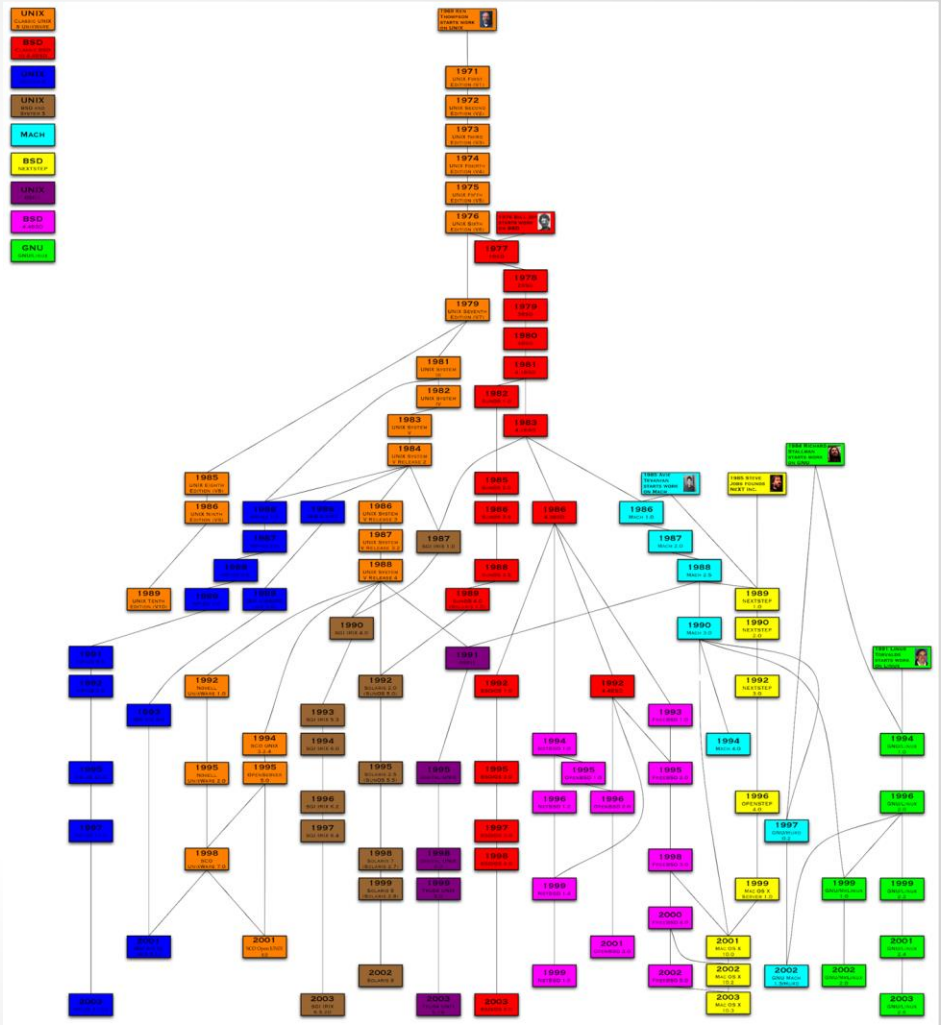
Specific case: Embedded Systems / Real-Time

Specific case: Embedded Systems / Real-Time

- Computer system directly linked to mechanical system
  - *Usually environment with constraints, and/or human life involved*
- Dedicated processor: DSP (Digital Signal Processor)
  - DSP controls signal (tension, current, …) instead of only bits/bytes
- Specific OS for industry and real-time
  - Must comply with very strict certification (formal verification, …)
  - Eventually, Linux in real time

UNIX

- [Core of this course]
- Big manufacturers wrote their own UNIX for their own CPU architecture
- …compatibility?
  - Rather…



Unixes Jungle

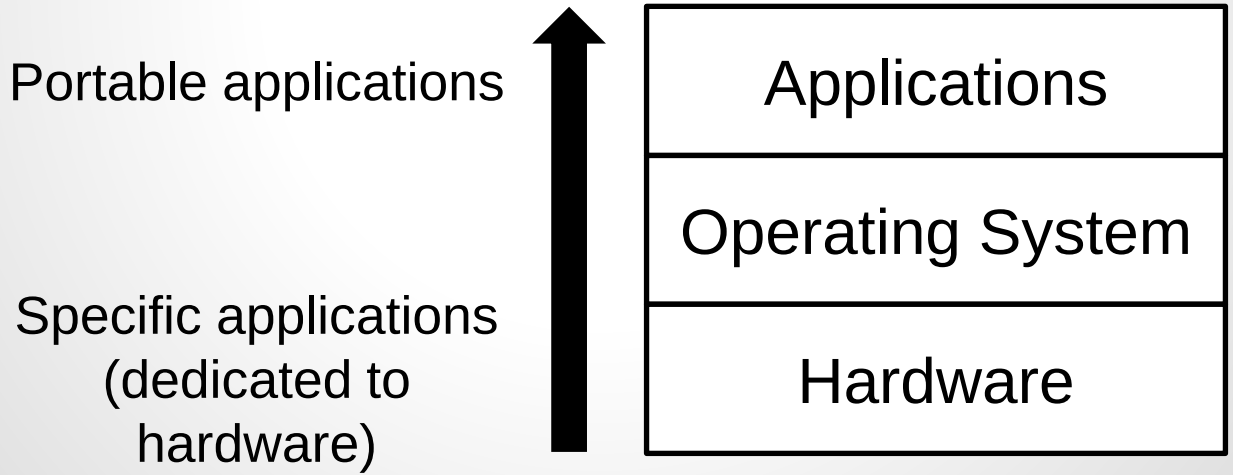
- 1969 – 1979 : Unix 1-6
  - *“SVR4” stands for “System V Release 4”*
- 1978 : BSD
- 1992 : FreeBSD, 4.4 BSD, NetBSD
- 1994 : OpenBSD
- 1987 : NextStep
- 2001 : Mac OS X
- 1987 : Minix
- 1991 : Linux

UNIX / POSIX

- POSIX (Portable Operating System Interface)
  - UNIX was the perfect candidate
- Family of standards for maintining compatibility between OS
  - Functionalities, Functions and Syscalls, Commands, Libraries, …
  - Not every UNIX & Linux is completely POSIX
- Non-UNIX OS can be POSIX compliant (or at least partially)
  - Windows (WSL, POSIX subsystem, …)
  - z/OS (USS/UNIX System Services)

Standardization & Portability

- POSIX helps to maintain a standard layer and common mechanisms
  - Programs are « easier » to port on various OS
  - *(well « easier » may not be so easy…)*



What is inside an OS?

Inside the OS

- Kernel: basic functionalities, syscalls
- Services: Accounts (login, nsswitch), Display Service
- Libraries: support functions, APIs (libc, libm, OpenGL)
- Applications: Web browser, Text Editor, Shell
- Support Applications: Terminal Emulator

What is a kernel?

- Software that runs in privileged mode
- “Heart” of the system
- Critical part (no error allowed inside)
- Delivers the first abstractions
  - Hardware independence
  - Basic resources management
- This is a jungle too

Some Glossary

User: an account

Root: the (most) privileged account

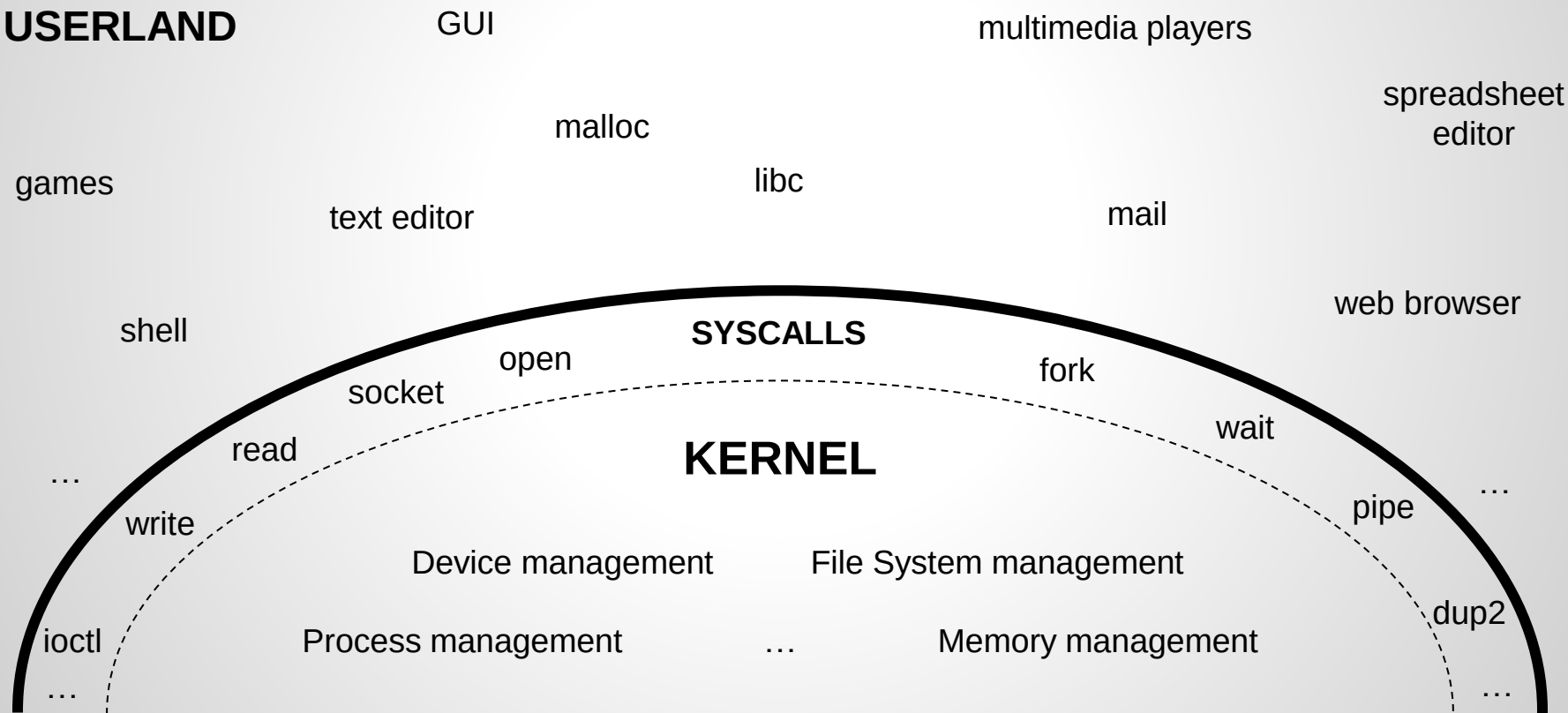
Shell: a command interpreter (that launches programs)

System call: a function provided by the kernel

Some Glossary

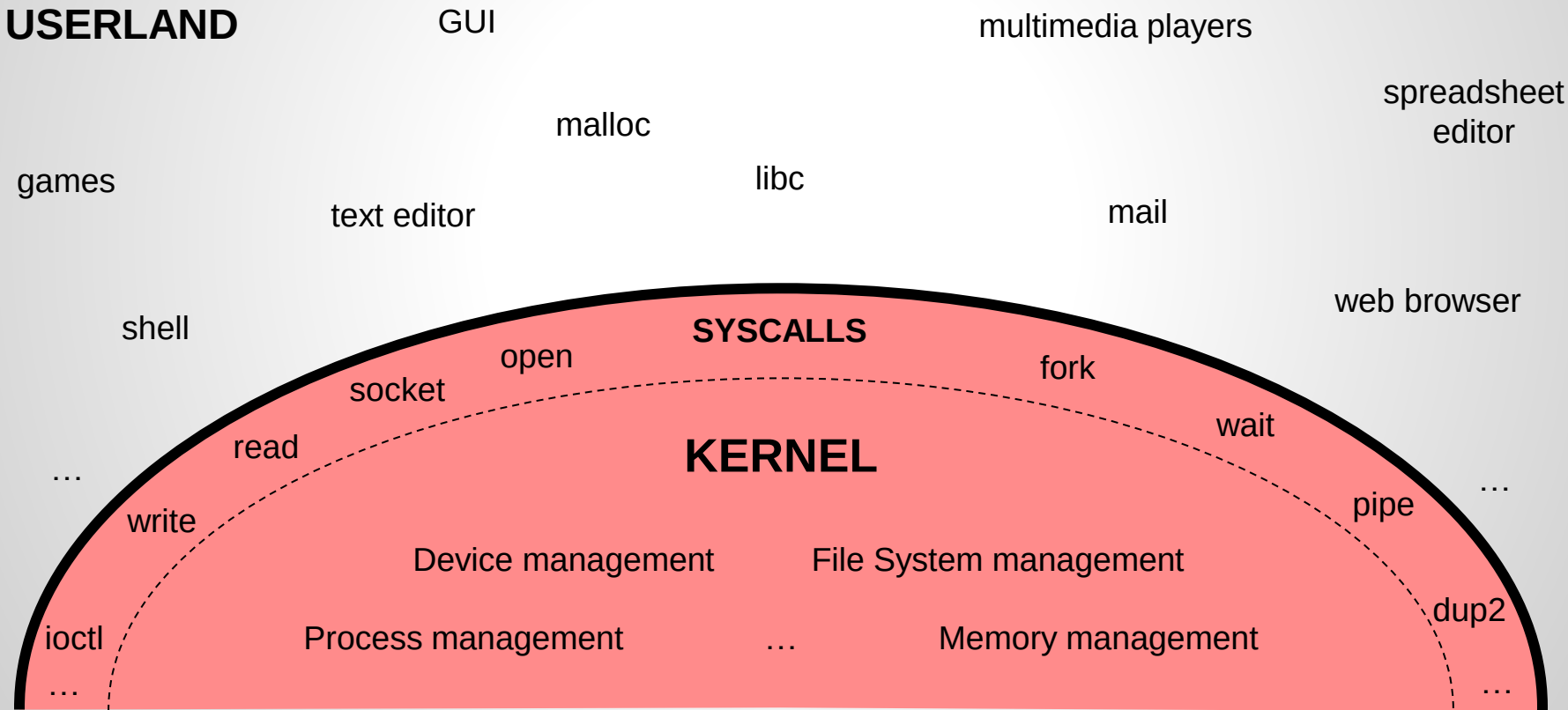
- Kernel mode/Kernel land: privileged mode
- User mode/User land: unprivileged mode
- System Call/Syscall:
  - Functionally: Allows a program to ask a service to the kernel
  - Technically: Mechanism for switching from user land to kernel land
  - Might be a software interruption…
  - …or a specific instruction (SYSENTER/SYSEXIT, SYSCALL/SYSRET, …)

Main concepts



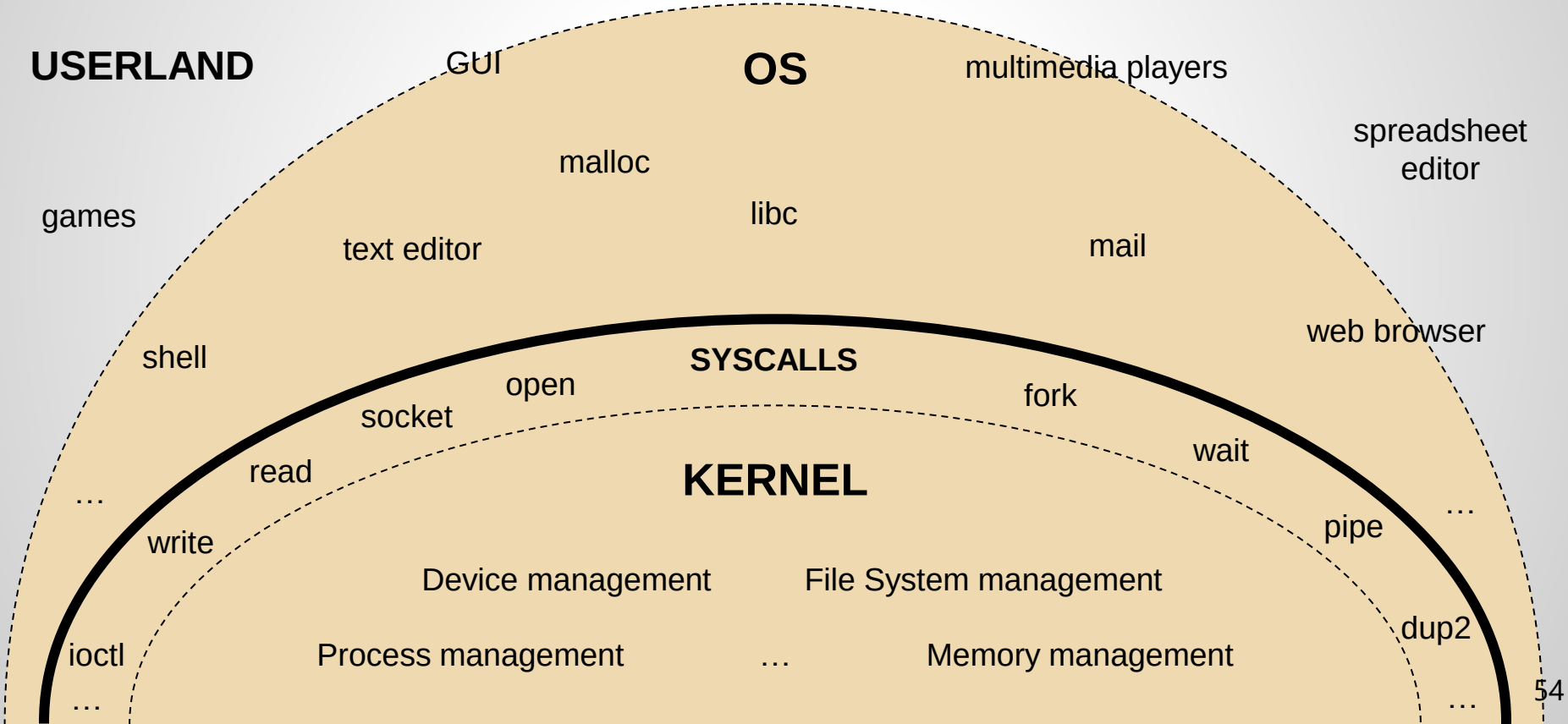
Main concepts

Everything not in kernel, is in userland



Main concepts

An OS includes a kernel, libraries, and utilities (some graphics, some CLI only)

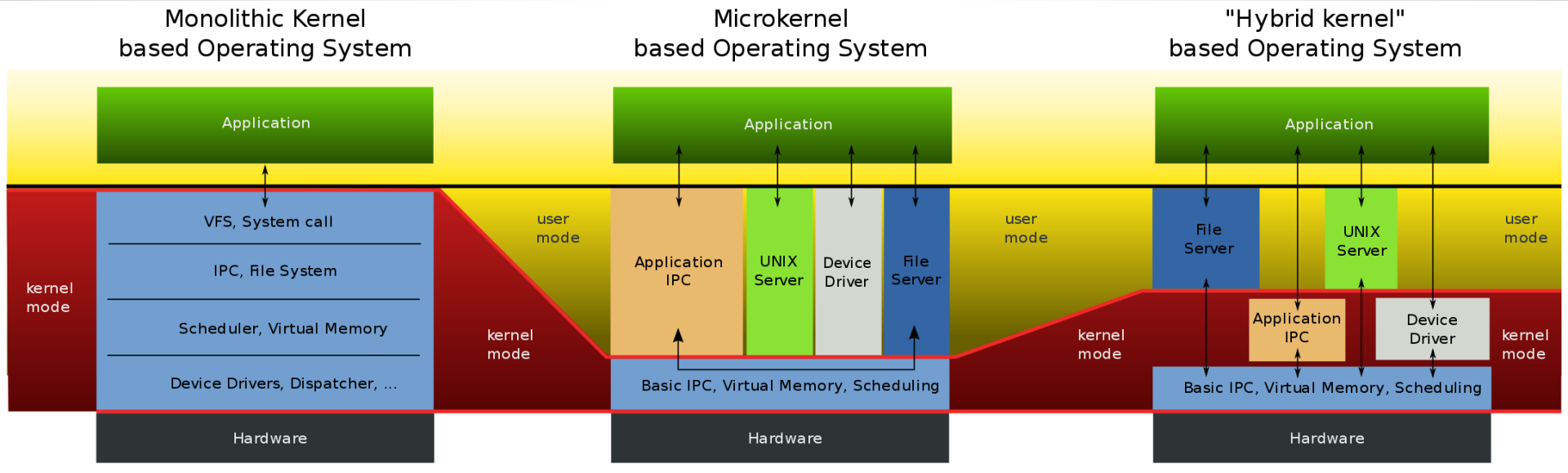


Kernel design: multiple models

- Monolithic kernel
  - Every kernel service is running in kernel mode (drivers, FS, …)
  - One giant abstraction exposed to applications
- Microkernel
  - Minimum kernel services run in kernel mode (scheduling, memory, IPC)
  - Kernel works like a broker:
    applications and services send/receive messages
- Hybrid/Modular kernel
  - Some services are in userland, some are in kerneland
  - Fewer message passing (less kernel/user switching that take time)

- Extreme cases: Nanokernel, Exokernel, Unikernel
  - Fewest code as possible, kernel as a library with each program, …

Kernel design: multiple models



Main concepts

Reminder:
The OS exposes an abstraction of the machine