Operating Systems: Introduction

Bachelor's Special Edition

Fabrice BOISSIER<fabrice.boissier@epita.fr>

2021-01-14

References

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

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

- Advanced Programming in the UNIX Environment *[en]*
  UNIX Network Programming, Volume 1: The Sockets Networking API
  UNIX Network Programming, Volume 2: Interprocess Communications
  W. Richard Stevens

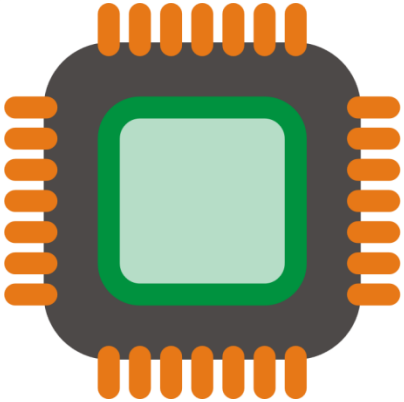
- Operating Systems Concepts *[en]*
  Abraham Silberschatz, Peter Baer Galvin, Greg Gagne

Outline

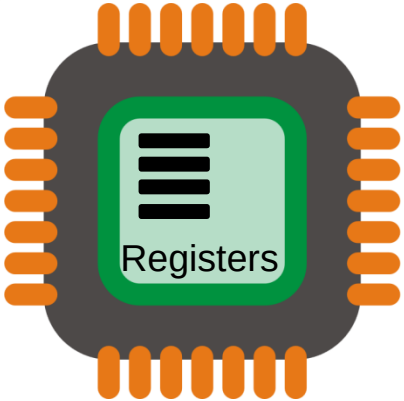
- Basic architecture concepts required
  - Hardware
- What is an OS
  - Why and where do we need OS
  - History of computing
- What is inside an OS
  - Kernel/Services/Applications
  - Main concepts
- UNIX & Linux

Basic architecture concepts required

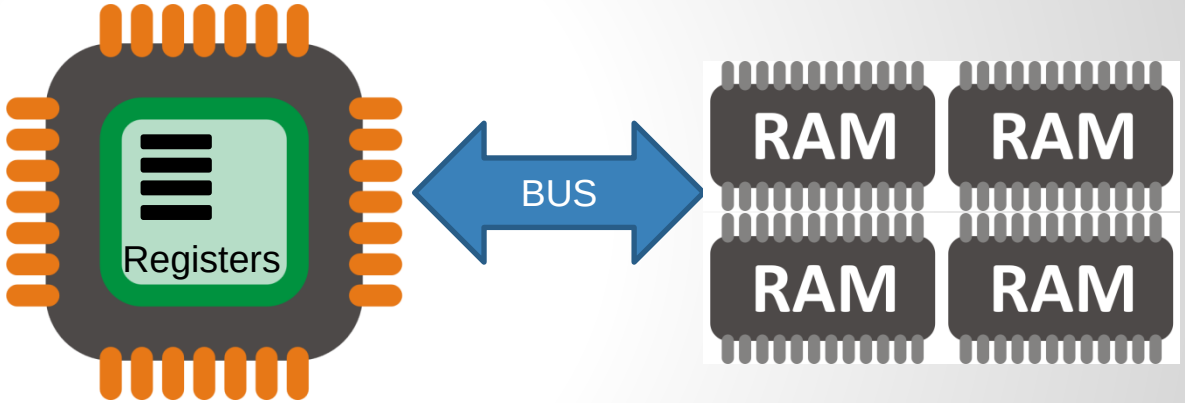
- CPU: central processing unit
  - *Register*: variable of the CPU
- Device: hardware
- Interruption: way to handle an event for the CPU
  - *Hardware interrupt* or *IRQ*: Interruption from hardware
  - *Software interrupt*: Interruption from software



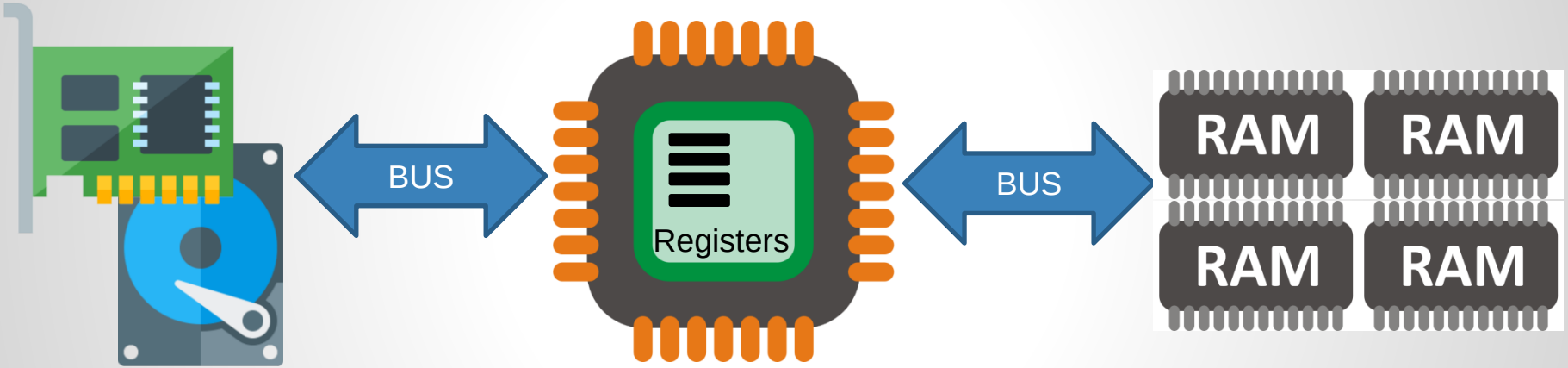
CPU



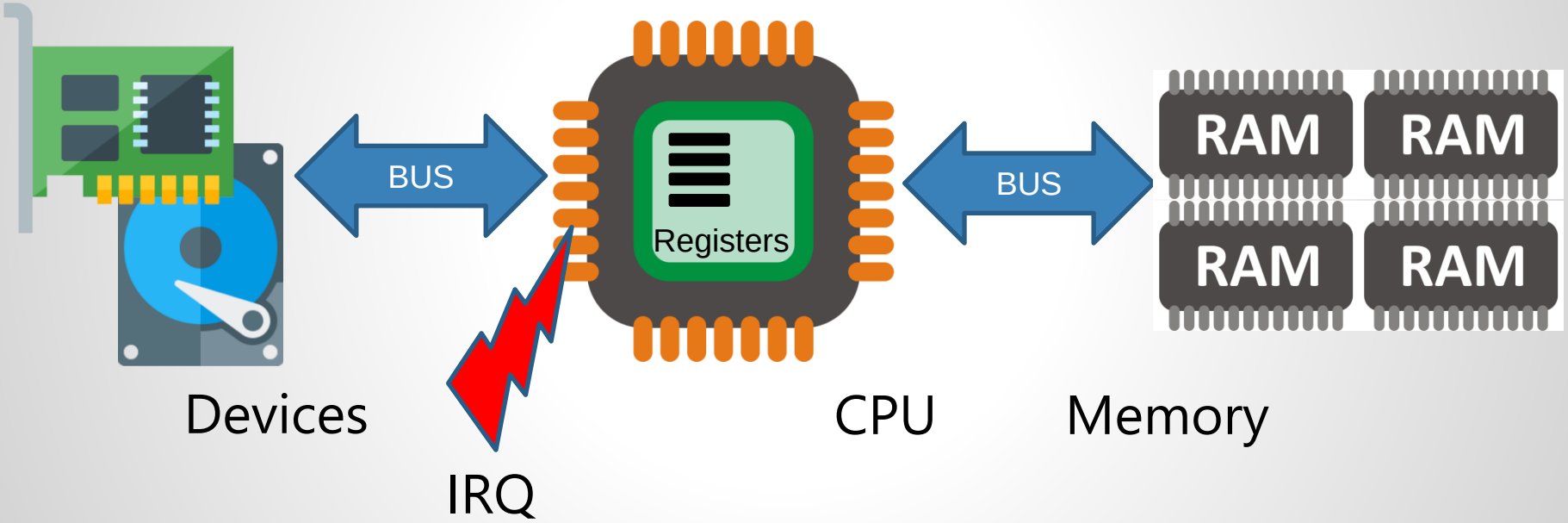
CPU

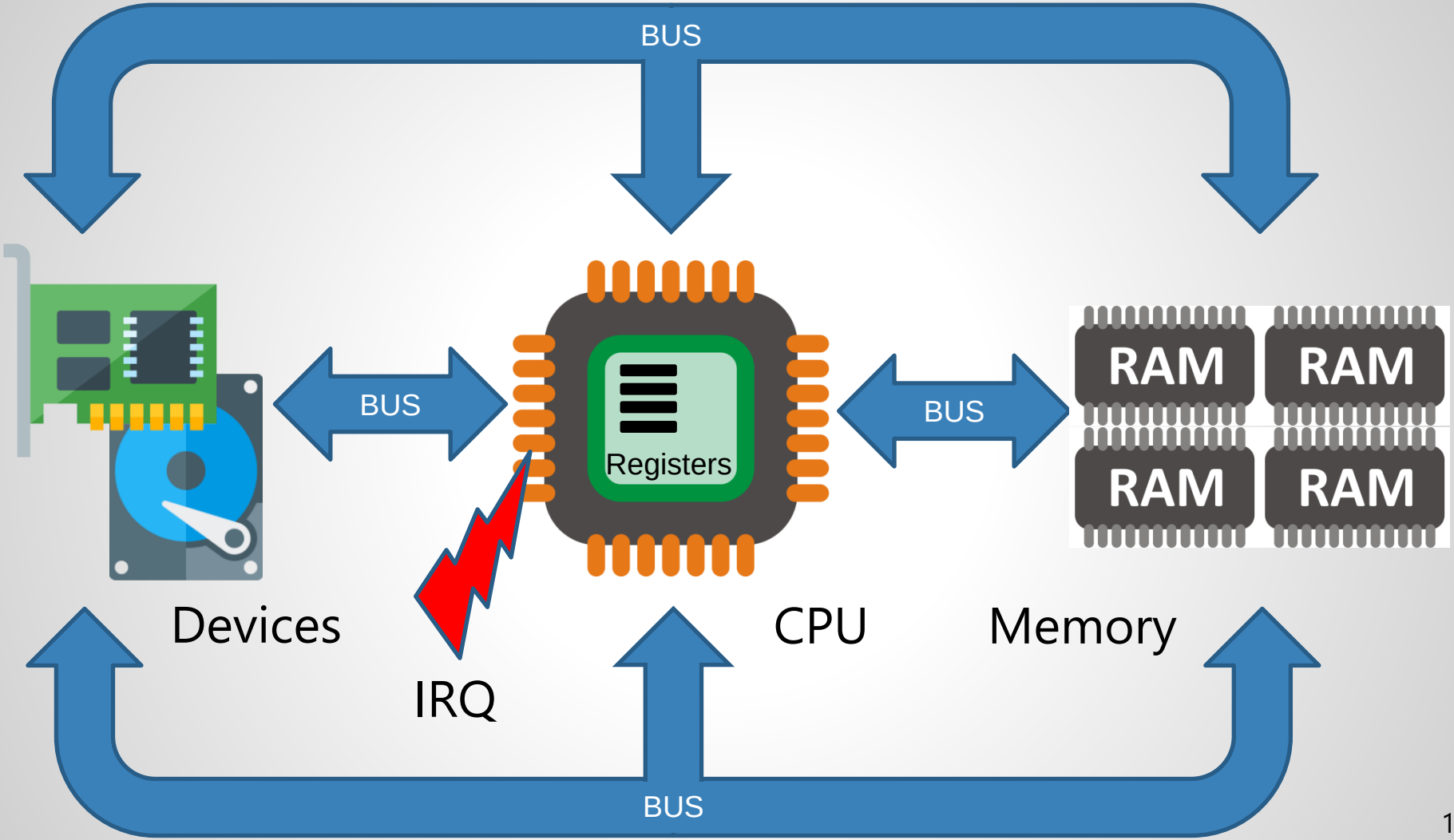


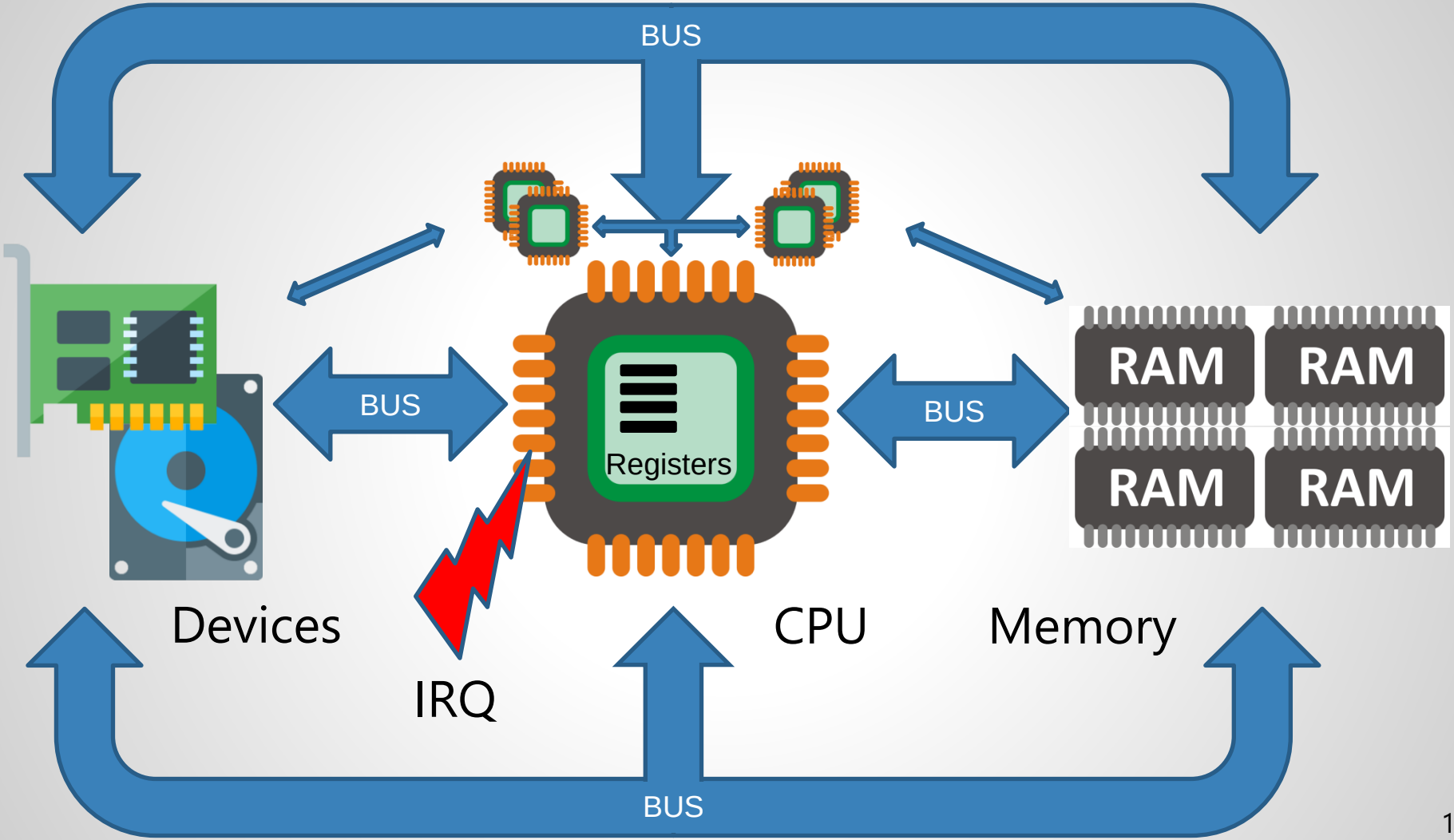
CPU Memory



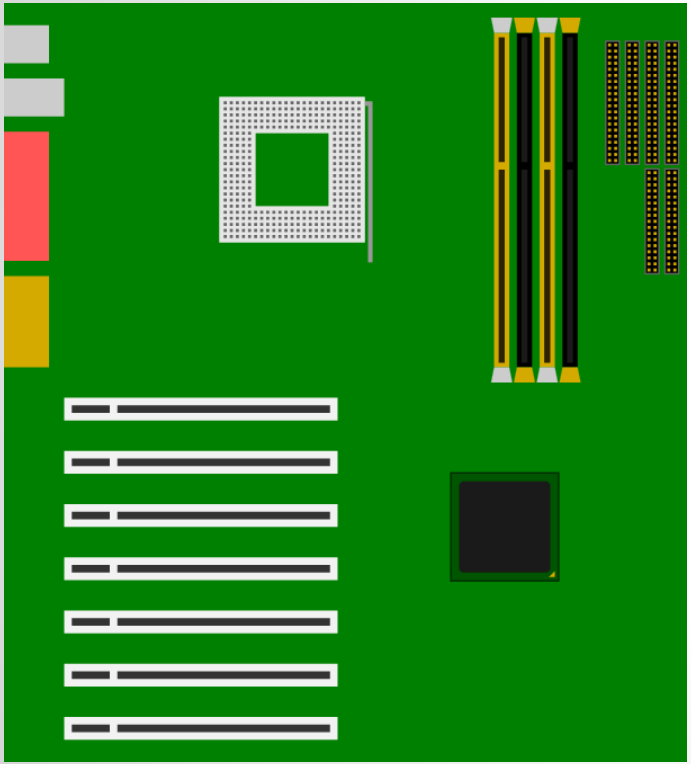
Devices CPU Memory


BUS
Registers
BUS
RAM
RAM
RAM
RAM
Devices
IRQ
CPU
Memory


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


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

Basic architecture concepts required



- The motherboard offers services to the CPU:
  - Interruptions / IRQ
    mouse, keyboard, … send IRQ and data
  - Memory / DMA
    copy this range of memory to that address
  - FSB (Front Side Bus)
    synchronization of all physical components
- …but also limitations:
  - Max number of CPU
  - Max physical memory
  - Everybody is linked to the FSB

Basic architecture concepts required

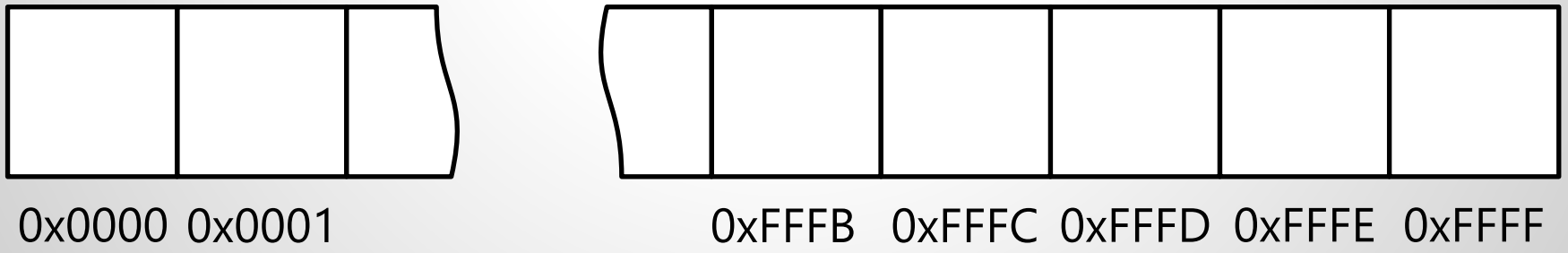
- Memory: huge array of cells containing data

0x0000

0xFFFF

Basic architecture concepts required

- Memory: huge array of cells containing data



Basic architecture concepts required

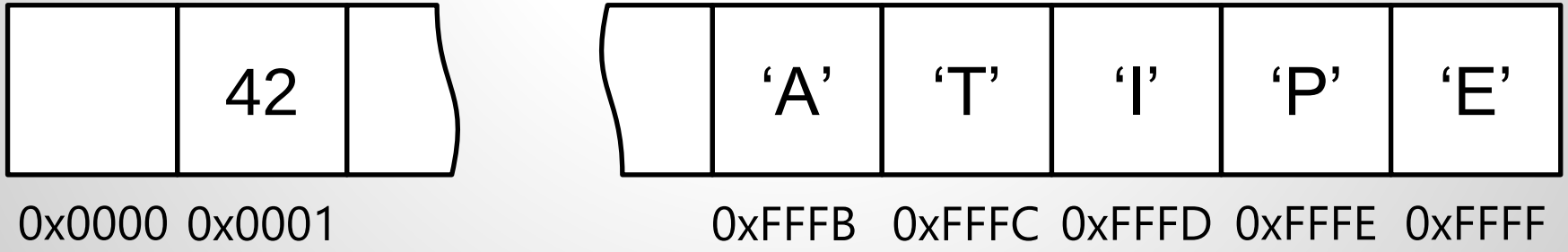
- Memory: huge array of cells containing data

```
int *nb = 0x0001;

(*nb) == 42
```

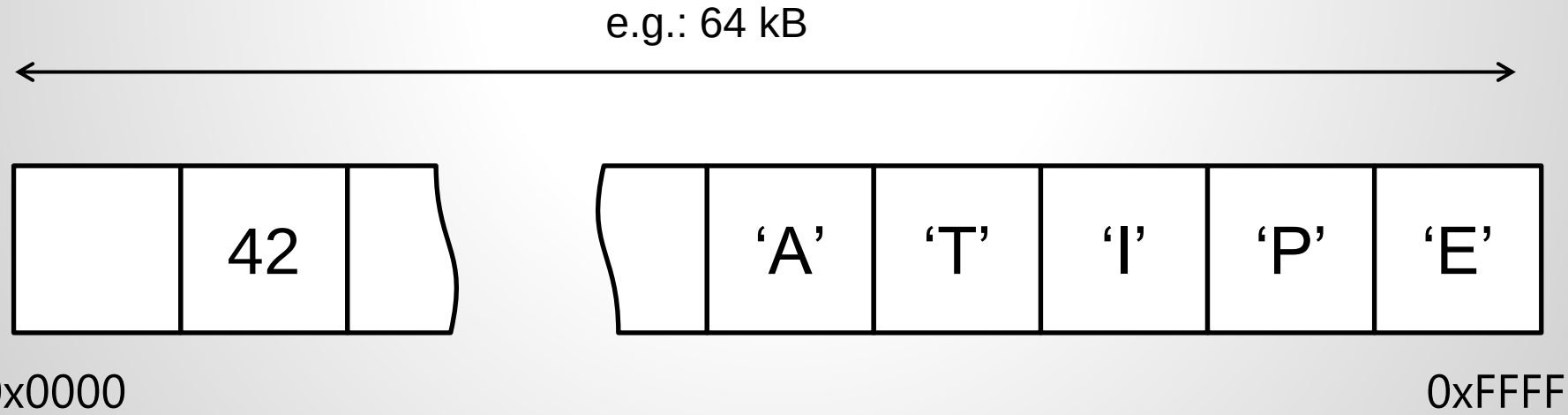
```
char *str = 0xFFFB;

str[0] = 'A';    // 0xFFFB
str[1] = 'T';    // 0xFFFC
str[2] = 'I';     // 0xFFFD
…
```



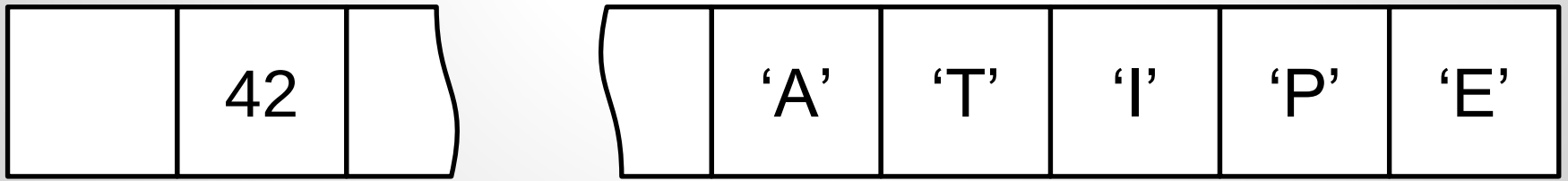
Basic architecture concepts required

- Memory: huge array of cells containing data



Basic architecture concepts required

- Memory: huge array of cells containing data
  - Each cell stores a specific size of data

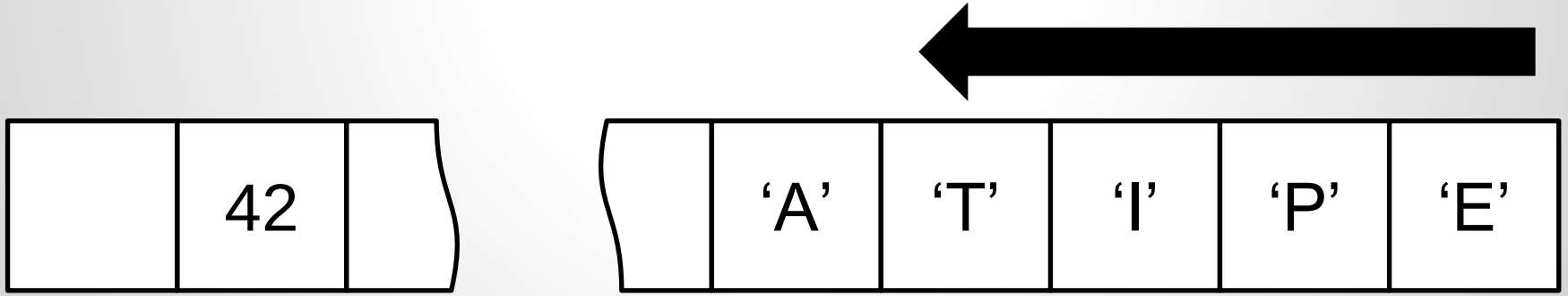


0x0000

0xFFFF

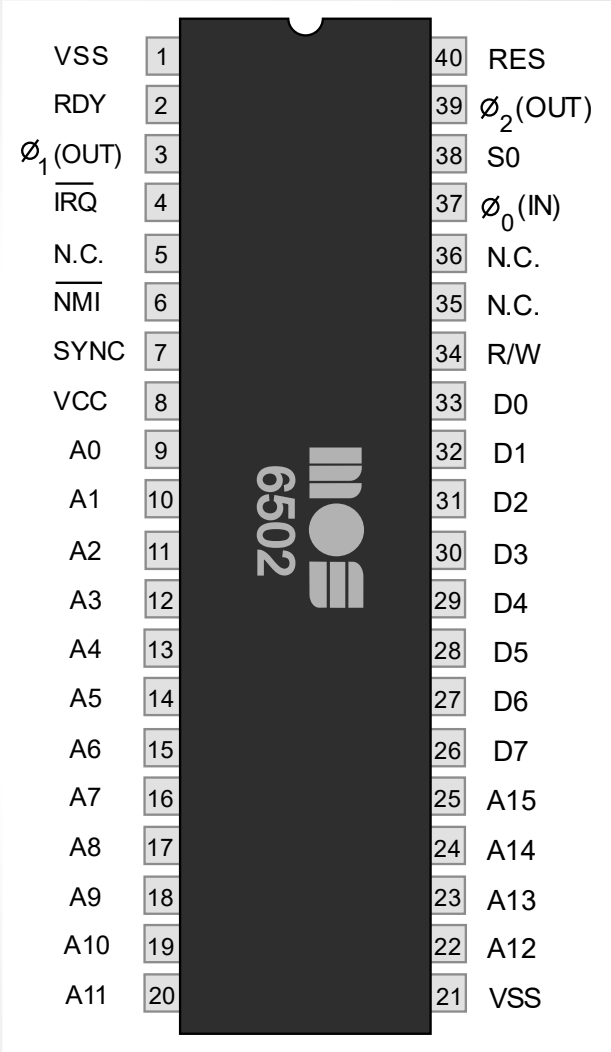
Basic architecture concepts required

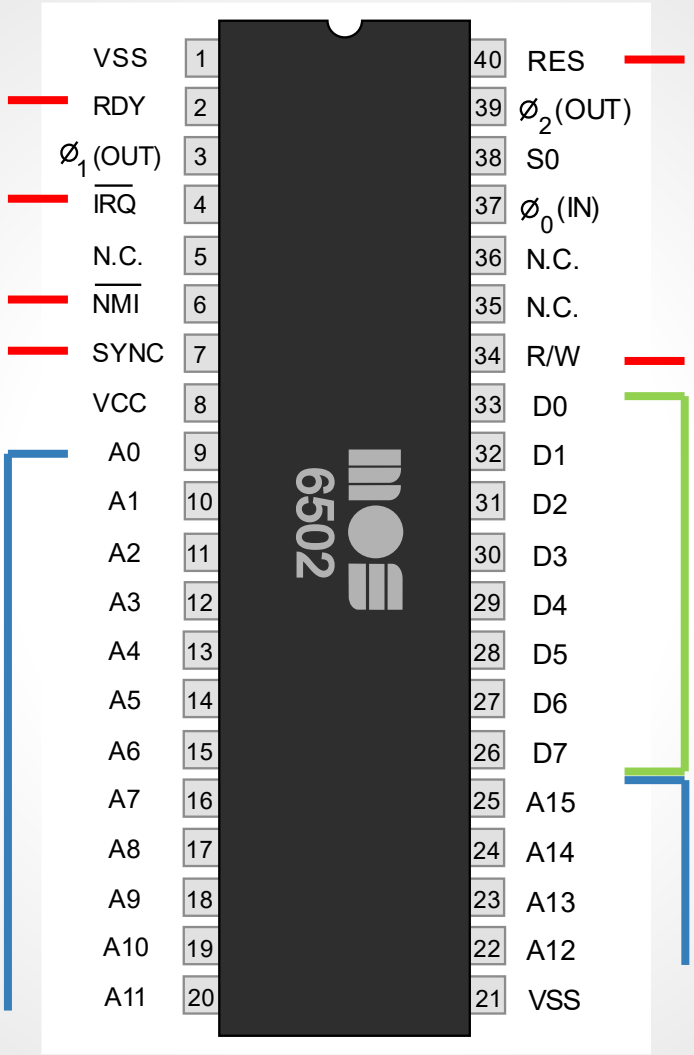
- Memory: huge array of cells containing data
  - Each cell stores a specific size of data
- Stack: specific part of memory that pushes/pulls data
  - Usually at the end of the memory, growing up toward beginning



CPUs have 3 main buses

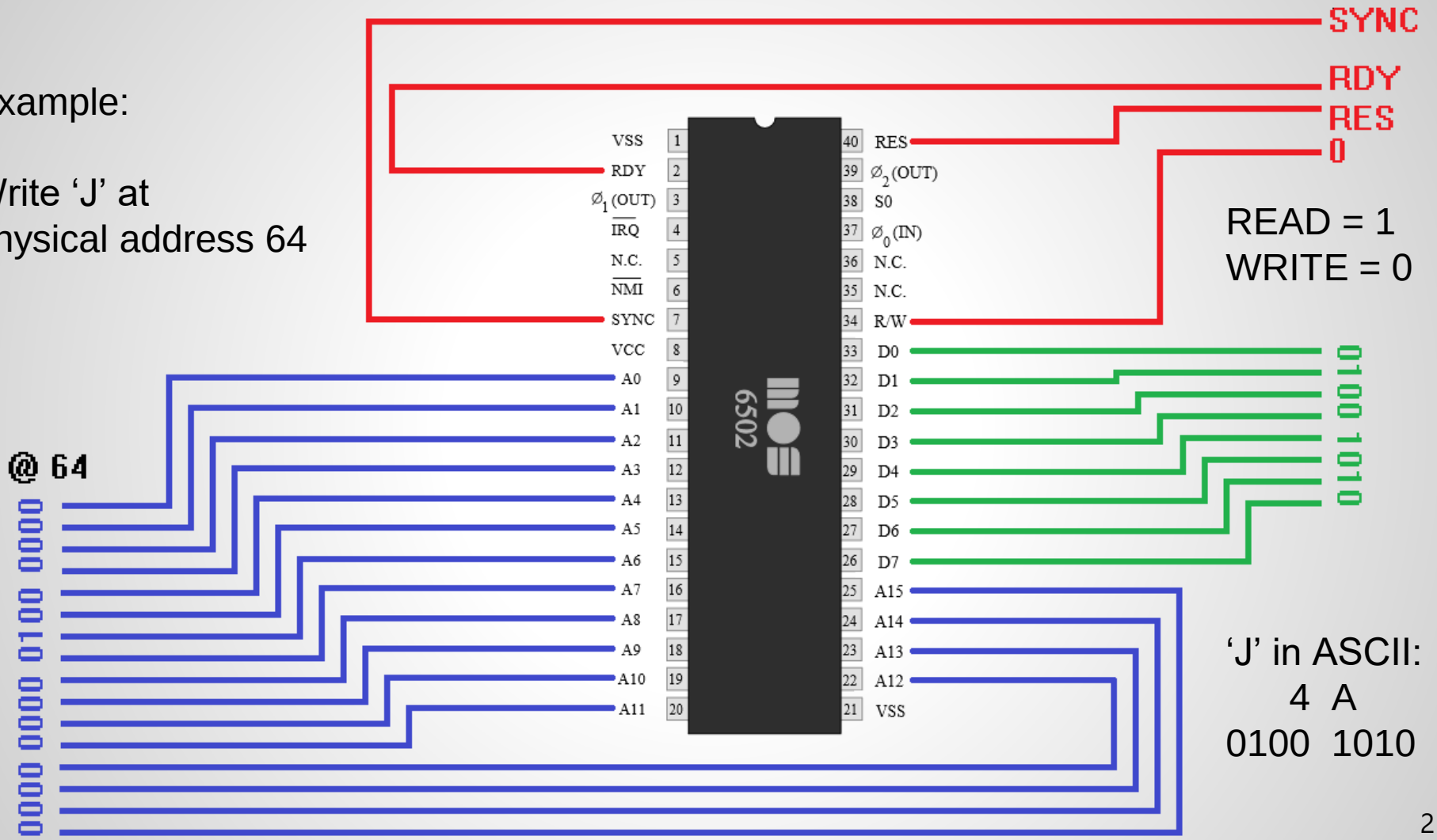
- Data Bus
  - For transfering data
- Address Bus
  - For selecting address
- Control Bus
  - For communicating with devices




Control Bus
Address Bus
Data Bus
VSS 1
RDY 2
Ø1 (OUT) 3
IRQ 4
N.C. 5
NMI 6
SYNC 7
VCC 8
A0 9
A1 10
A2 11
A3 12
A4 13
A5 14
A6 15
A7 16
A8 17
A9 18
A10 19
A11 20
40 RES
39 Ø2 (OUT)
38 S0
37 Ø0 (IN)
36 N.C.
35 N.C.
34 R/W
33 D0
32 D1
31 D2
30 D3
29 D4
28 D5
27 D6
26 D7
25 A15
24 A14
23 A13
22 A12
21 VSS
6502

Example:

Write ‘J’ at
physical address 64



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 3 - User mode:
  - Context in which the CPU allows the minimum actions
  - *Regular memory addresses can be accessed*
    (areas not reserved for the system or specific management)
  - *Regular instructions can be executed*
    (move, arithmetic, logic, branch, …)

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?

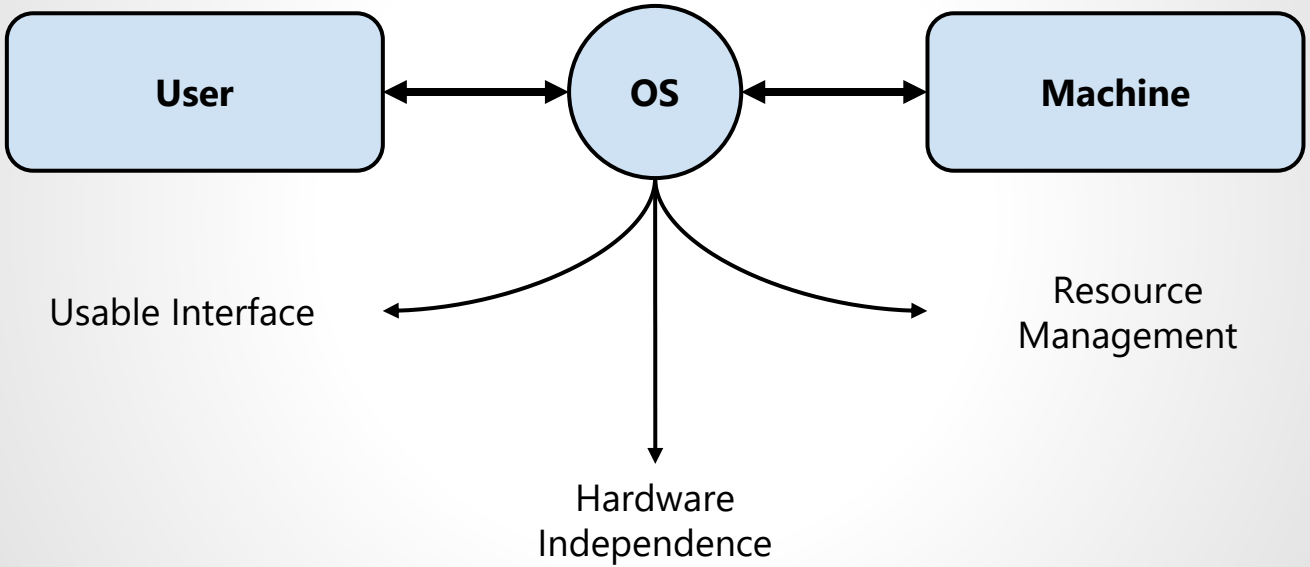
Operating systems perform two essentially unrelated functions: providing application programmers (and application programs, naturally) a clean abstract set of resources instead of the messy hardware ones and managing these hardware resources.

-- 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

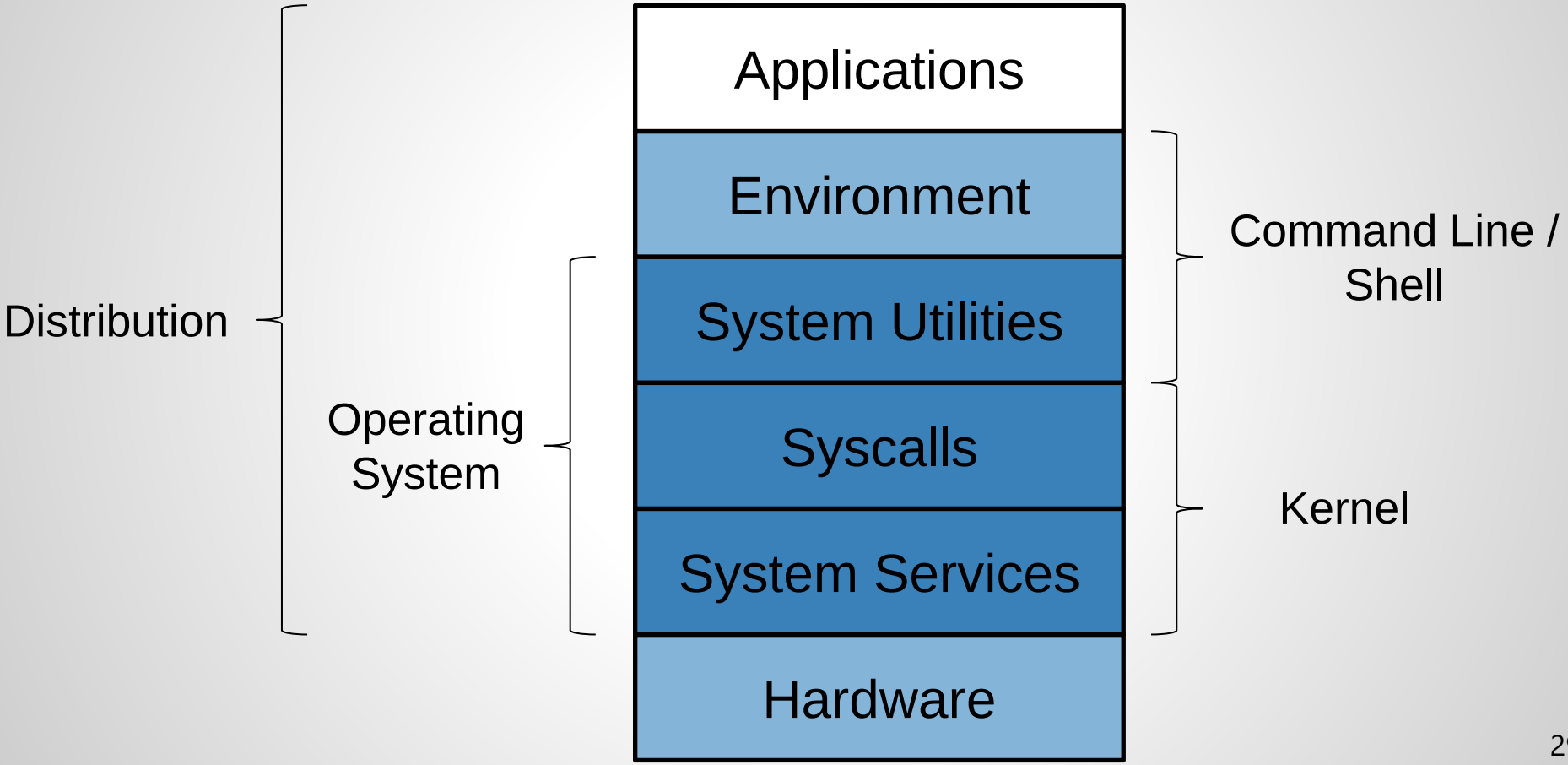
Most used definition:

- 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
- ...

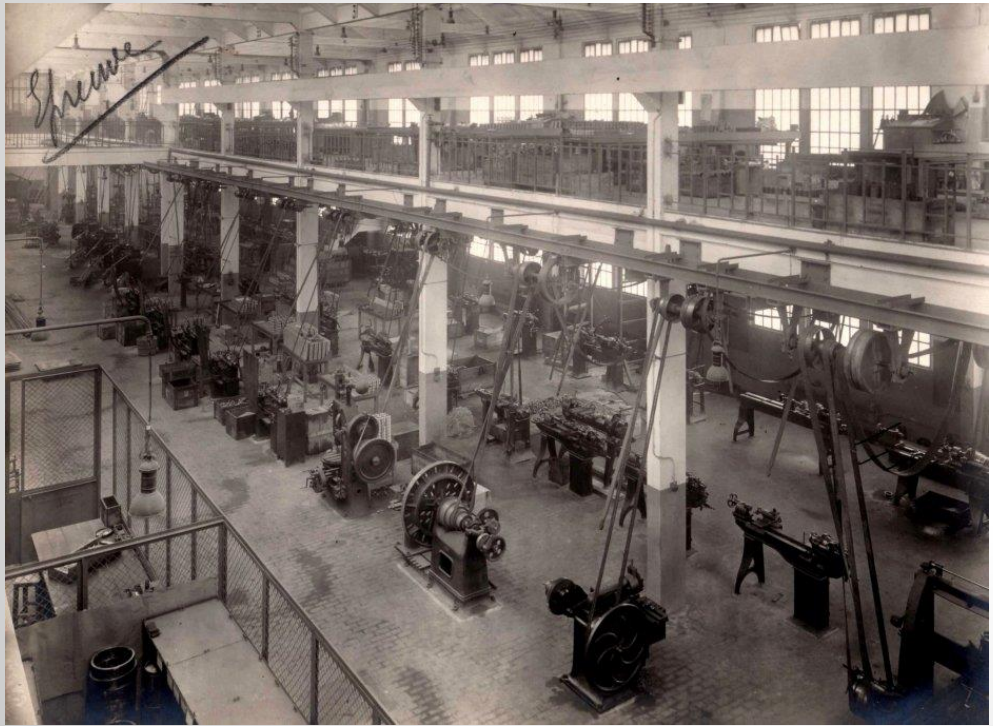
History

- Generations
- Taxonomy of machines and OS

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

Before/Without computers: Machines

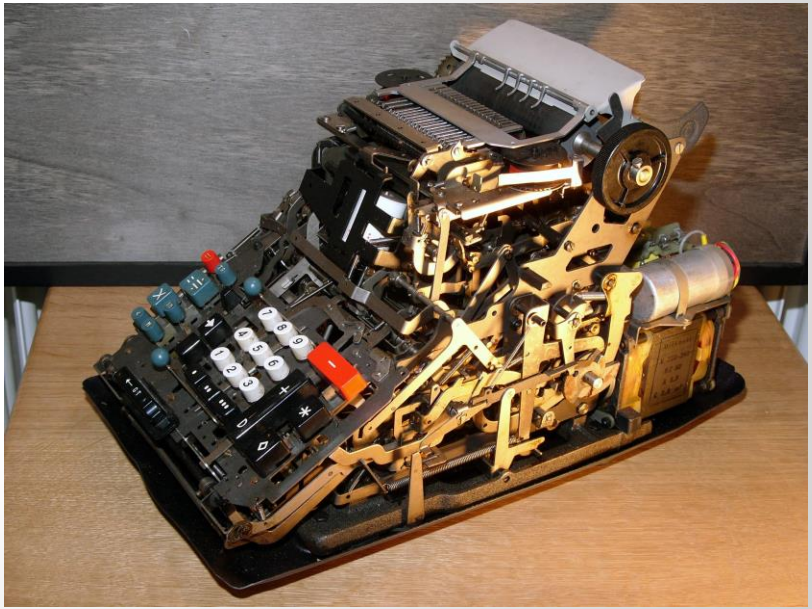
- Manual tasks
- Mechanical (gears, timing belts, …)
- Electromechanical



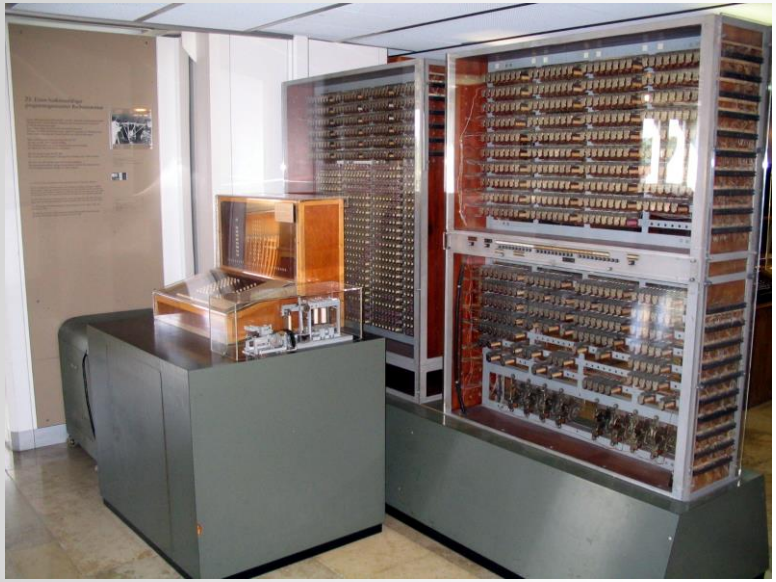
(Watch « Monsieur Patron » clip: *everything* is already here during the 50's-60's, but in mechanical version…)

Don't forget:

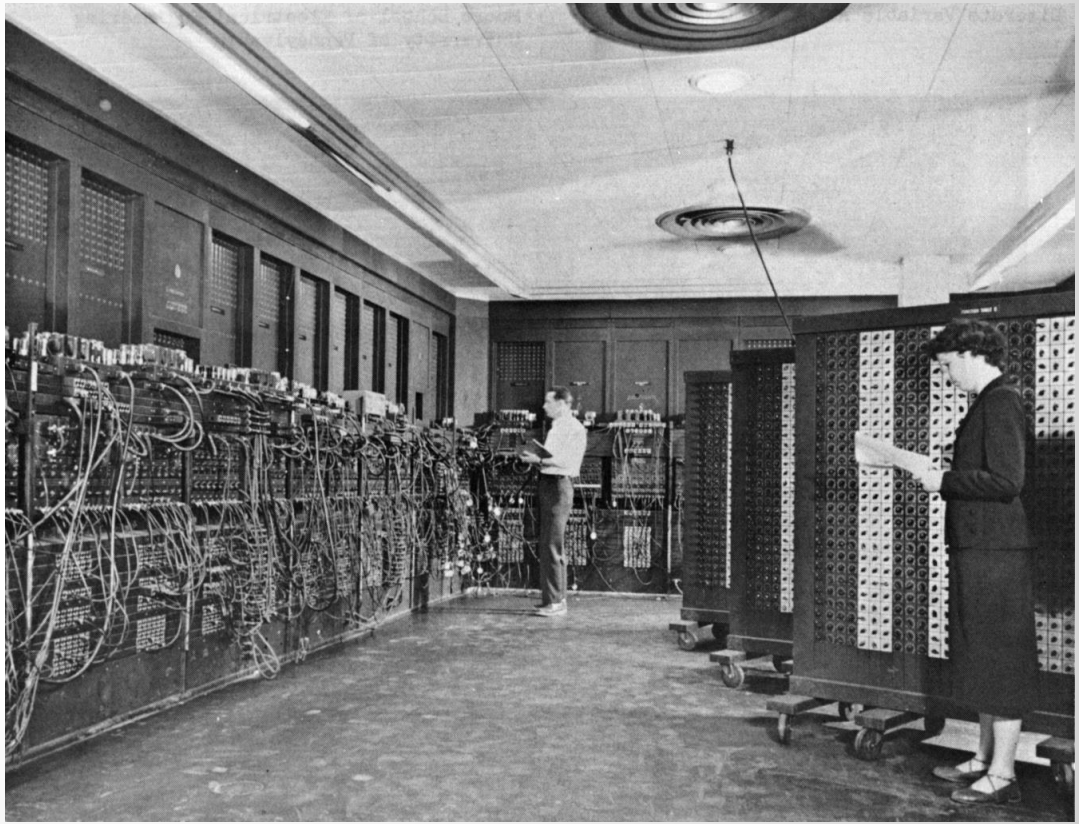
Initially, IT resolved business problems



First Generation (1945-55)

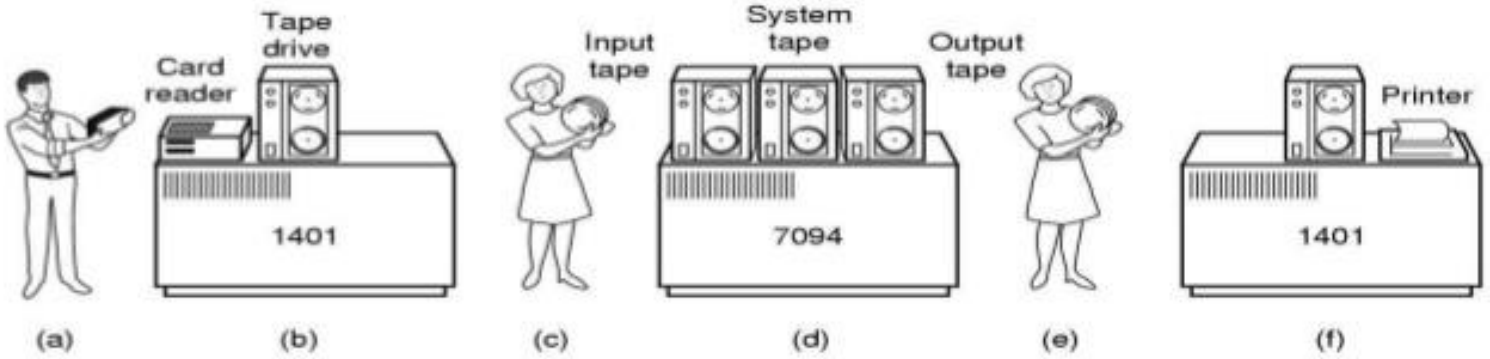


You « code » your program directly by linking/unlinking boxes each of which executes a single function (full of electric relays + ferrite toroid memory)



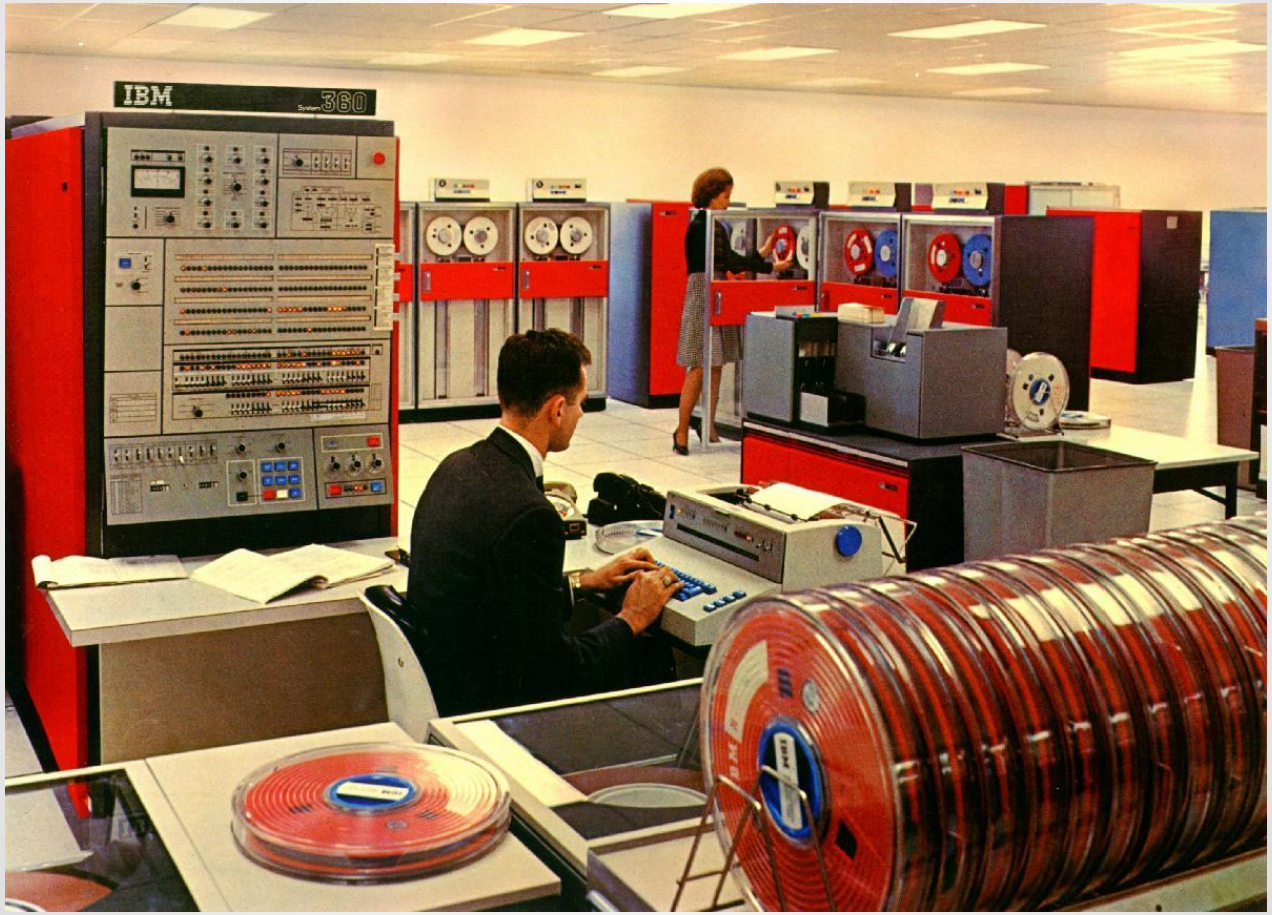
Second Generation (1955-65)

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

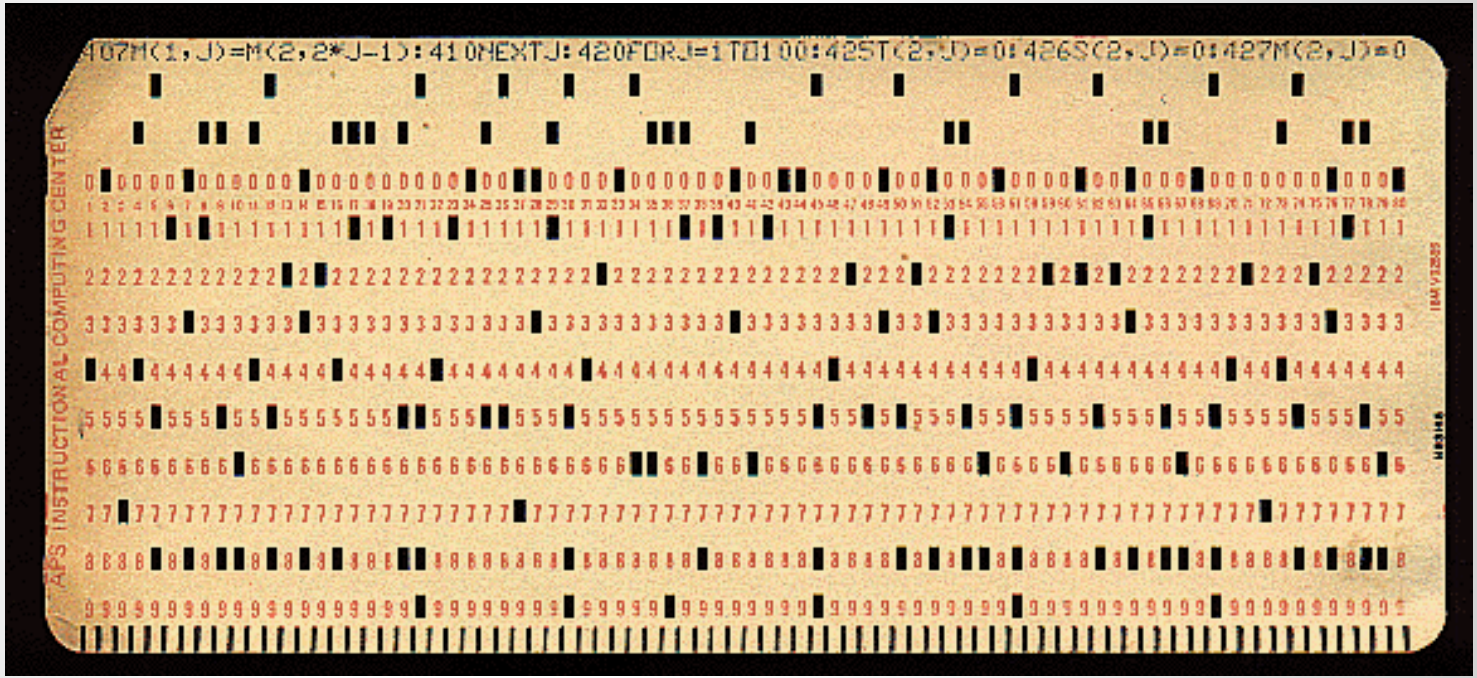
Second Generation (1955-65)



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)

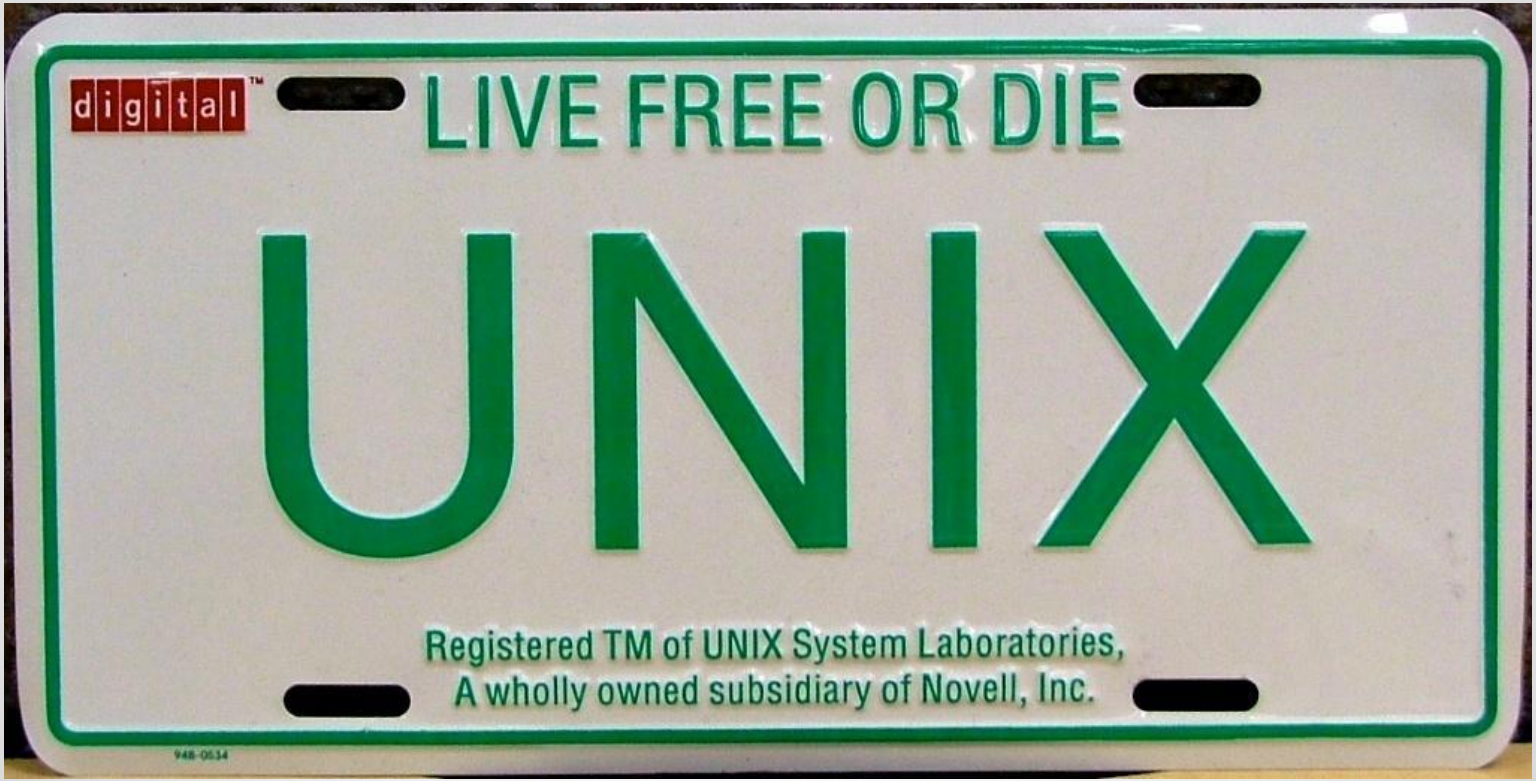
OS: OS/360 (later: OS/370, MVS, z/OS)

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 processors 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

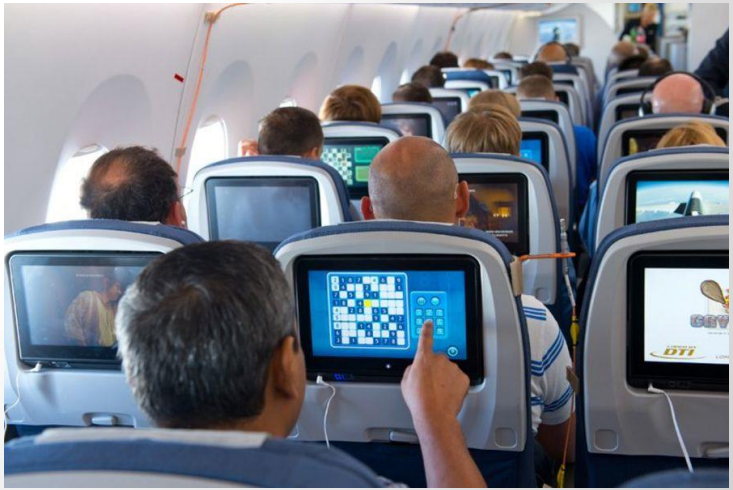
Specific case: Supercomputers

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, …)



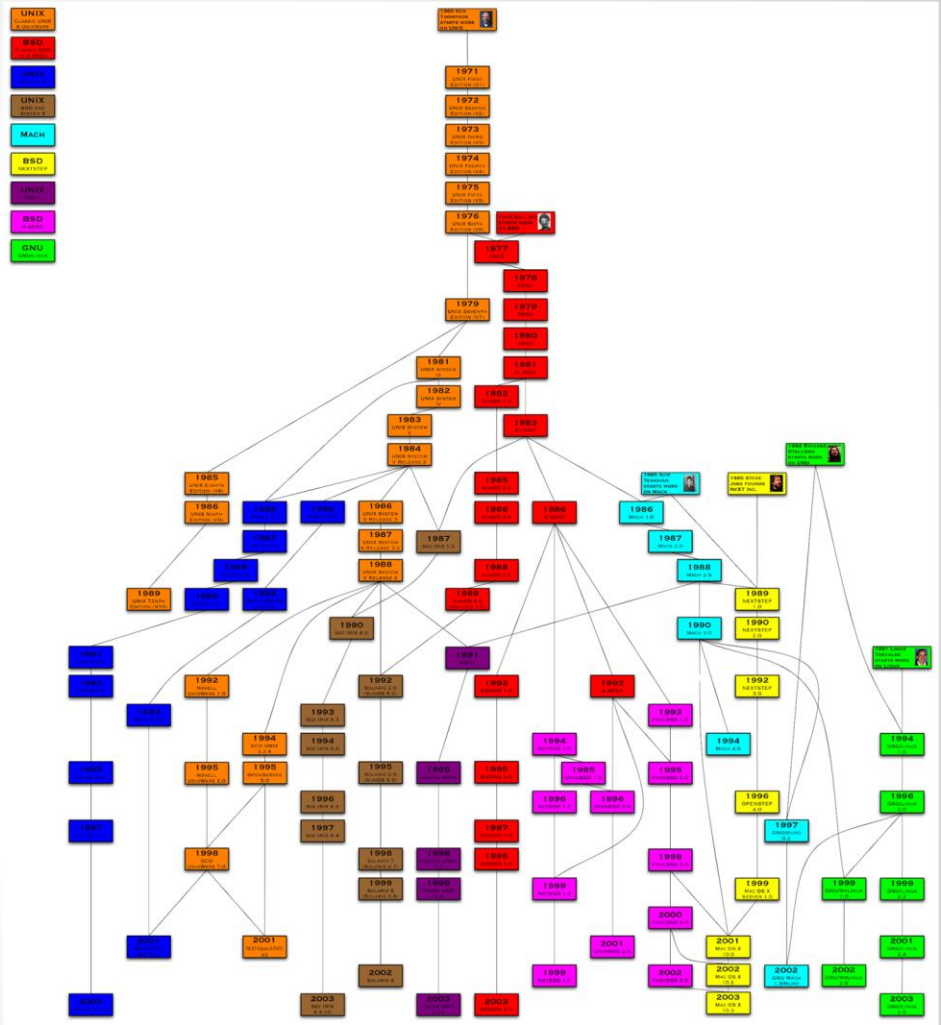
Specific case: Real-Time

Specific case: 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

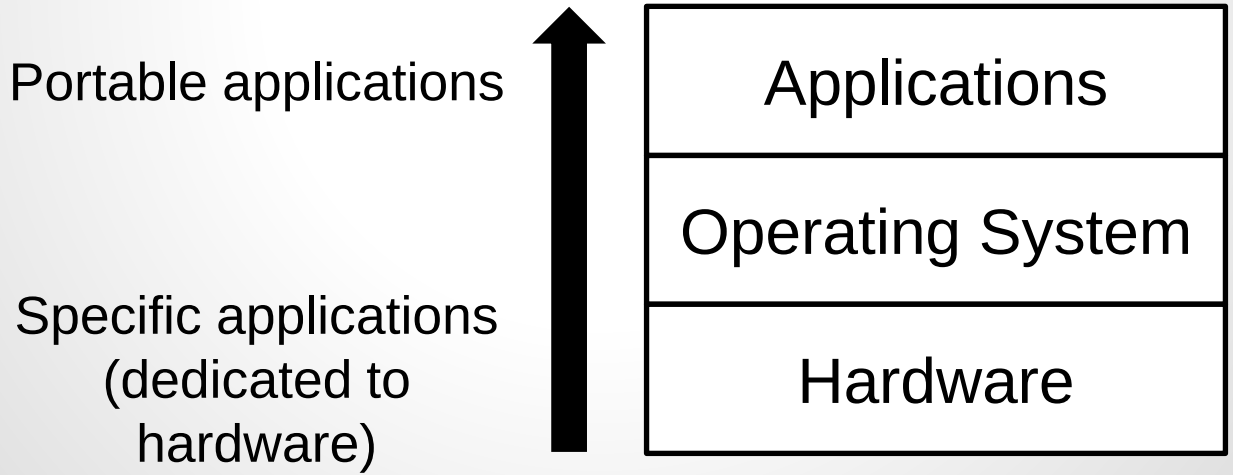
- 1969 – 1979 : Unix 1-6
  - *“SVR4” stands for “System V Release 4”*
- 1978 : BSD
- 1992 : FreeBSD, 4.4 BSD, NetBSD
- 1994 : OpenBSD
- 1987 : NextStep
- 2001 : Mac OS X
- 1987 : Minix
- 1991 : Linux

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

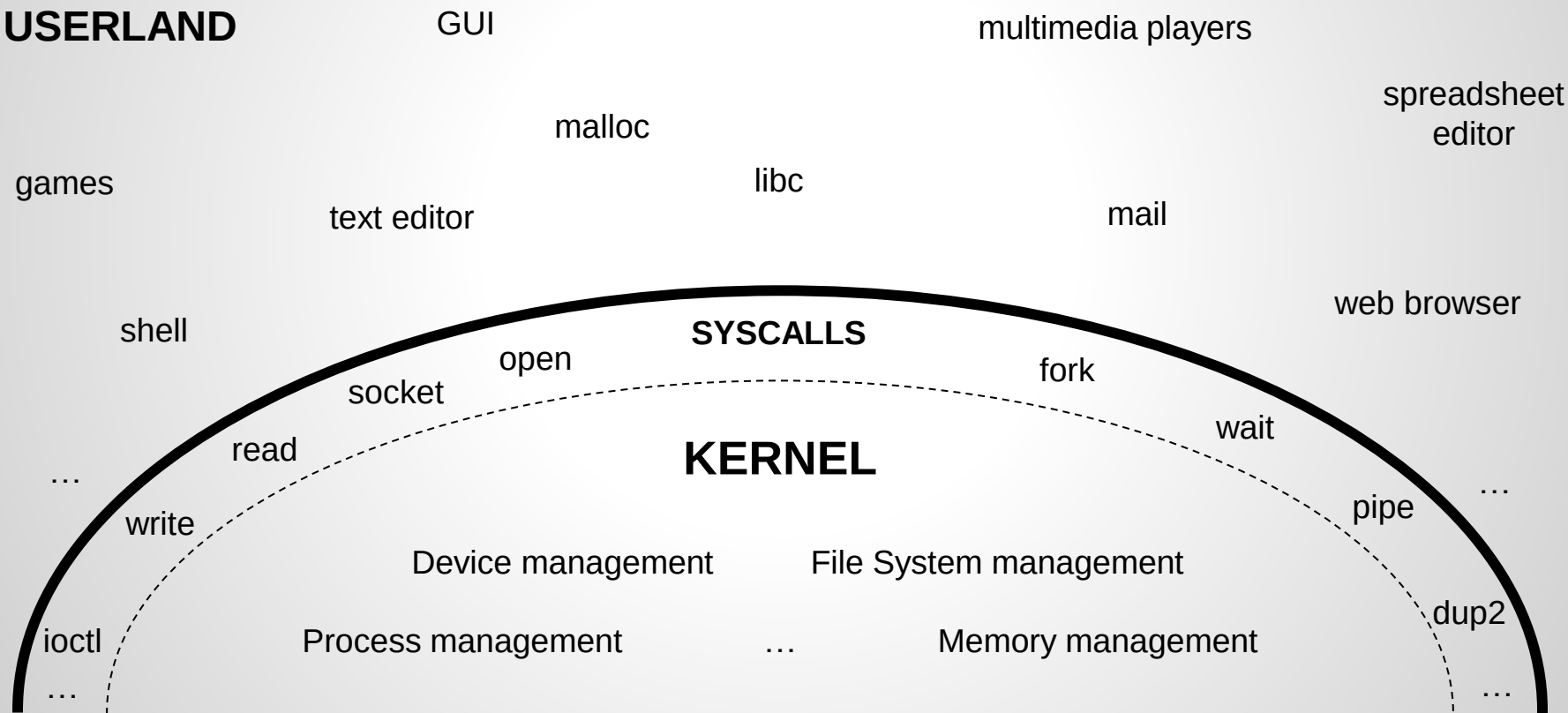
Shell: a command interpreter (that launches programs)

System call: a function provided by the kernel

Some Glossary

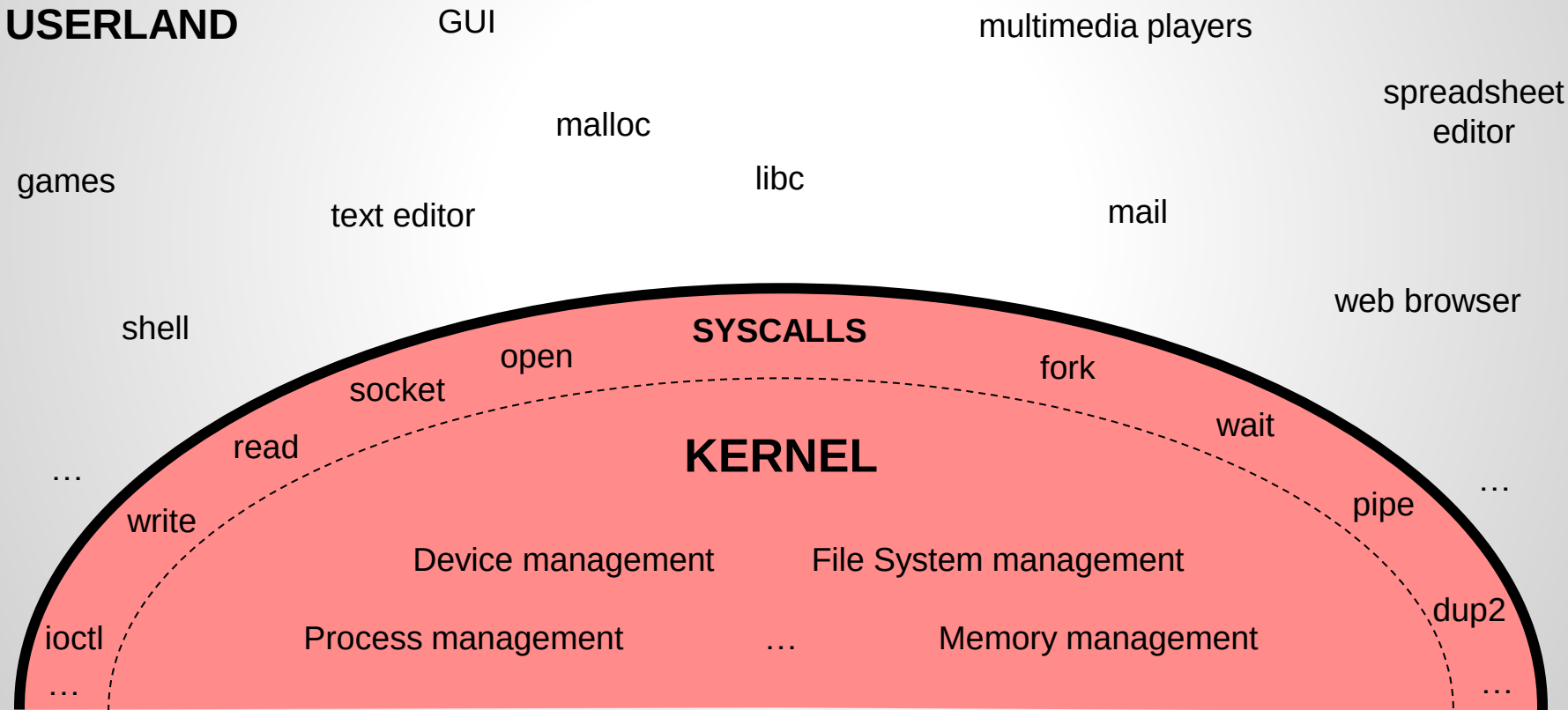
- Kernel mode/Kernel land: privileged mode
- User mode/User land: unprivileged mode
- System Call/Syscall:
  - Functionally: Allows a program to ask a service to the kernel
  - Technically: Mechanism for switching from user land to kernel land
  - Might be a software interruption…
  - …or a specific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

Main concepts



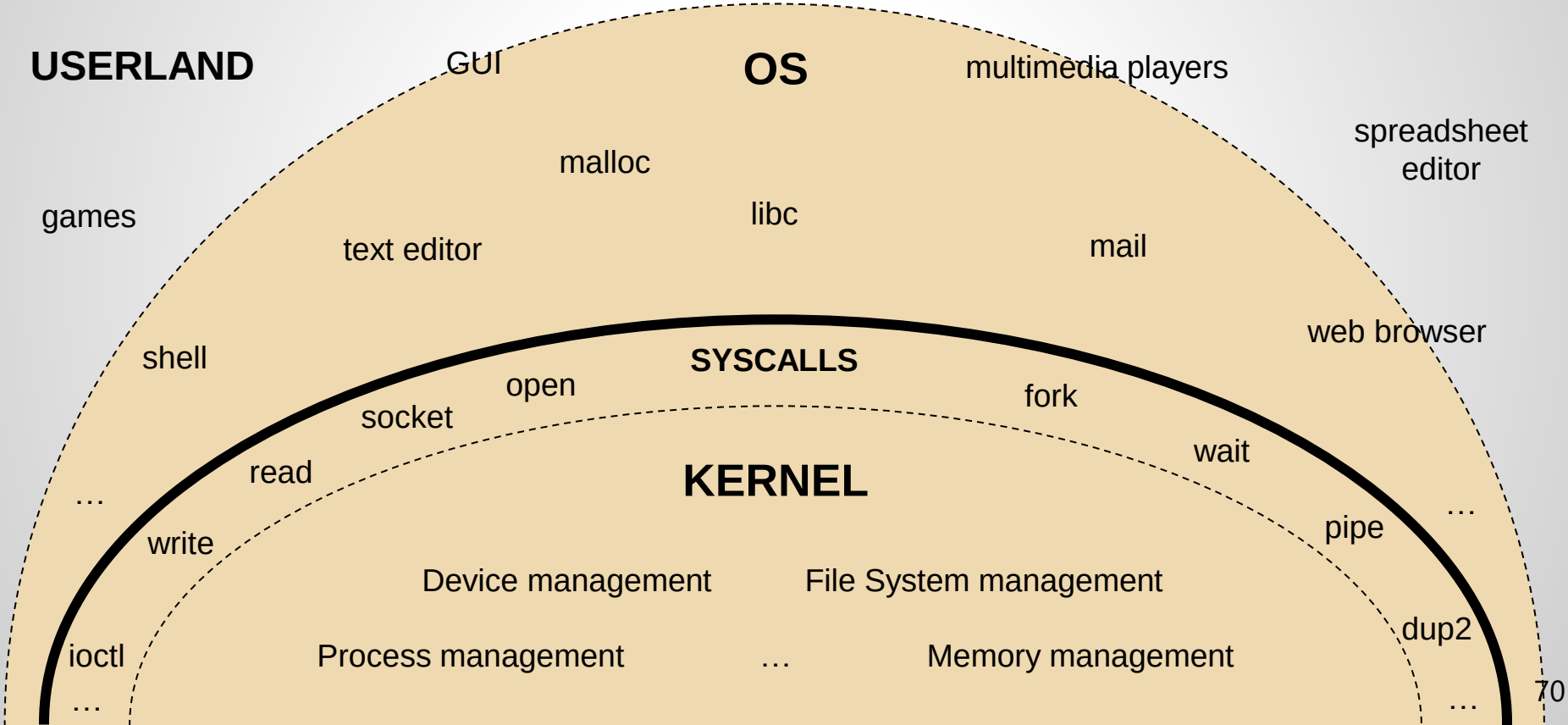
Main concepts

Everything not in kernel, is in userland



Main concepts

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

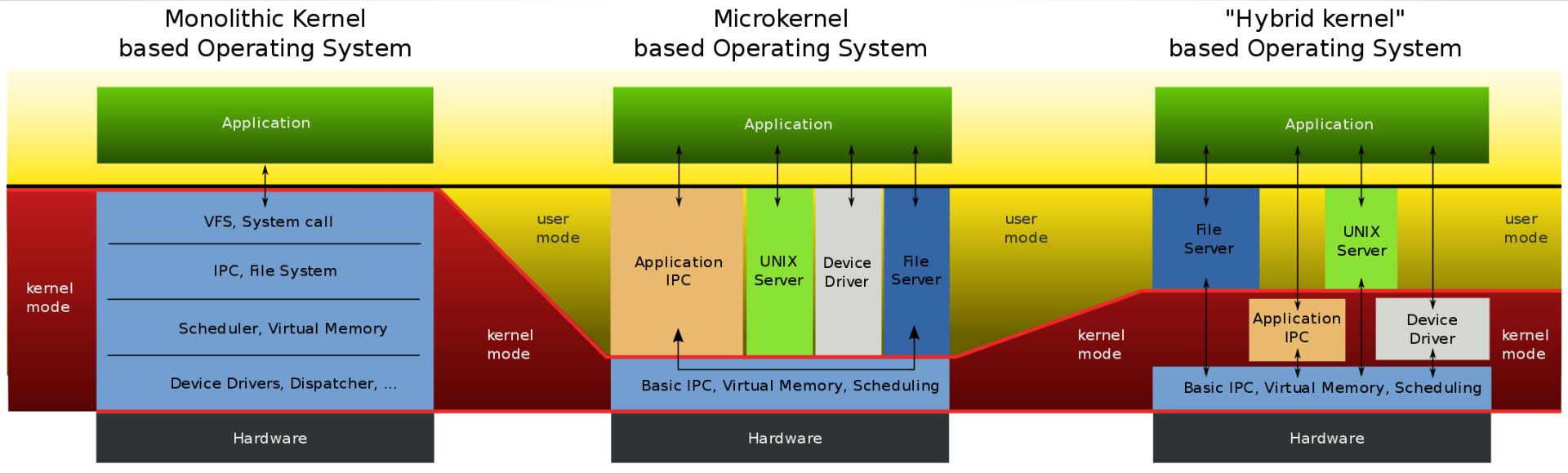


Kernel design: multiple models

- 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)

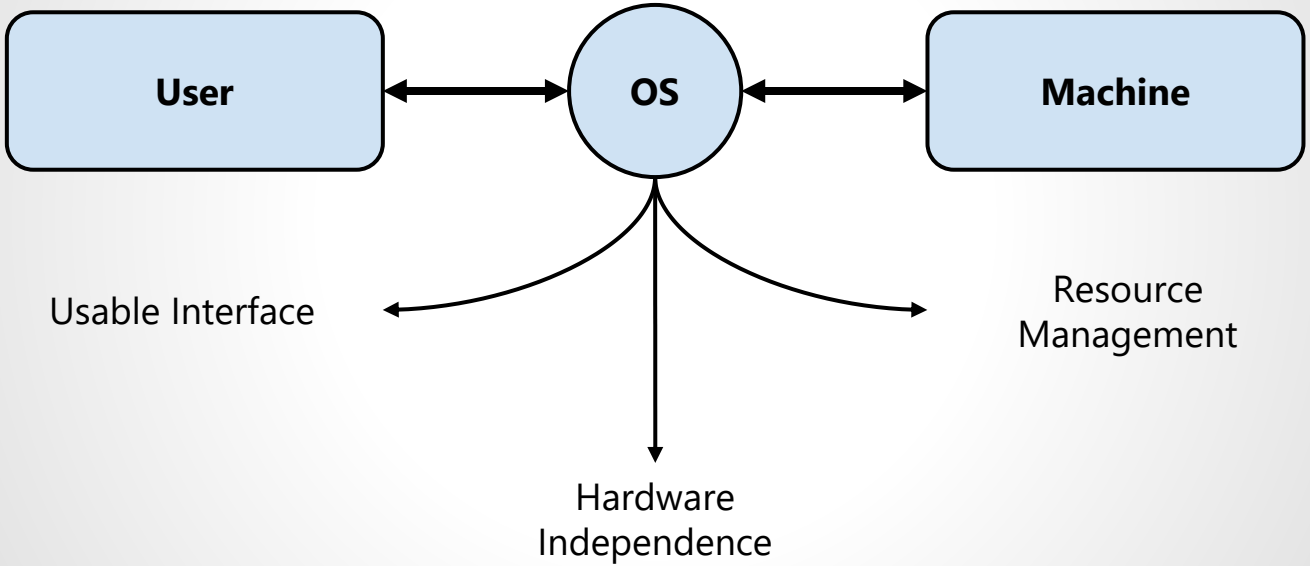
- Extreme cases: Nanokernel, Exokernel, Unikernel
  - Fewest code as possible, kernel as a library with each program, …

Kernel design: multiple models



Main concepts

Reminder:
The OS exposes an abstraction of the machine



UNIX & Linux

UNIX

- Family of operating systems
  - 2 branches: System V (SVR), BSD
  - SUS (Single UNIX Specification), POSIX
- Delivers the first abstractions
  - Hardware independence
  - Basic resources management
- “Everything is a file”

Linux

- UNIX re-written from scratch
  - One kernel (sometimes modified): www.kernel.org
  - GNU re-written softwares (glibc, gls, gawk, …)

- Distributions
  - Group of preinstalled softwares for dedicated usages
    Kali Linux for pentest, Debian for servers, Ubuntu for client, …
  - Some might be sold with technical support and patches
    Red Hat, SUSE, …

- Mainly POSIX-compliant
  - Don’t worry: the common UNIX things are all here…

UNIX/Linux: do not fear terminals

- CLI: Command Line Interface
  - Type everything in a terminal (directly sends characters)
  - Call programs by their names, and add parameters
  - Perfect for precise configuration and automation

- GUI: Graphical User Interface *(since 80s~90s)*
  - Graphical environment with buttons
  - Call programs with buttons and clicks
  - Simplified for business users, but heavier to run and manage
    However: precise configuration might be available in a file

UNIX/Linux: do not fear terminals

- CLI: Command Line Interface
  - Windows: MS-DOS *(BATCH language)*, PowerShell, WSL *(Windows Subsystem for Linux)*
  - UNIX/Linux: shell (sh, csh & tcsh, bash, ksh/Korn-shell, zsh, …)

- GUI: Graphical User Interface
  - Windows: explorer.exe
  - UNIX/Linux: X server (X11, Xorg, …) + window manager (dwm, xfce, Gnome, KDE, …)