



# Operating Systems

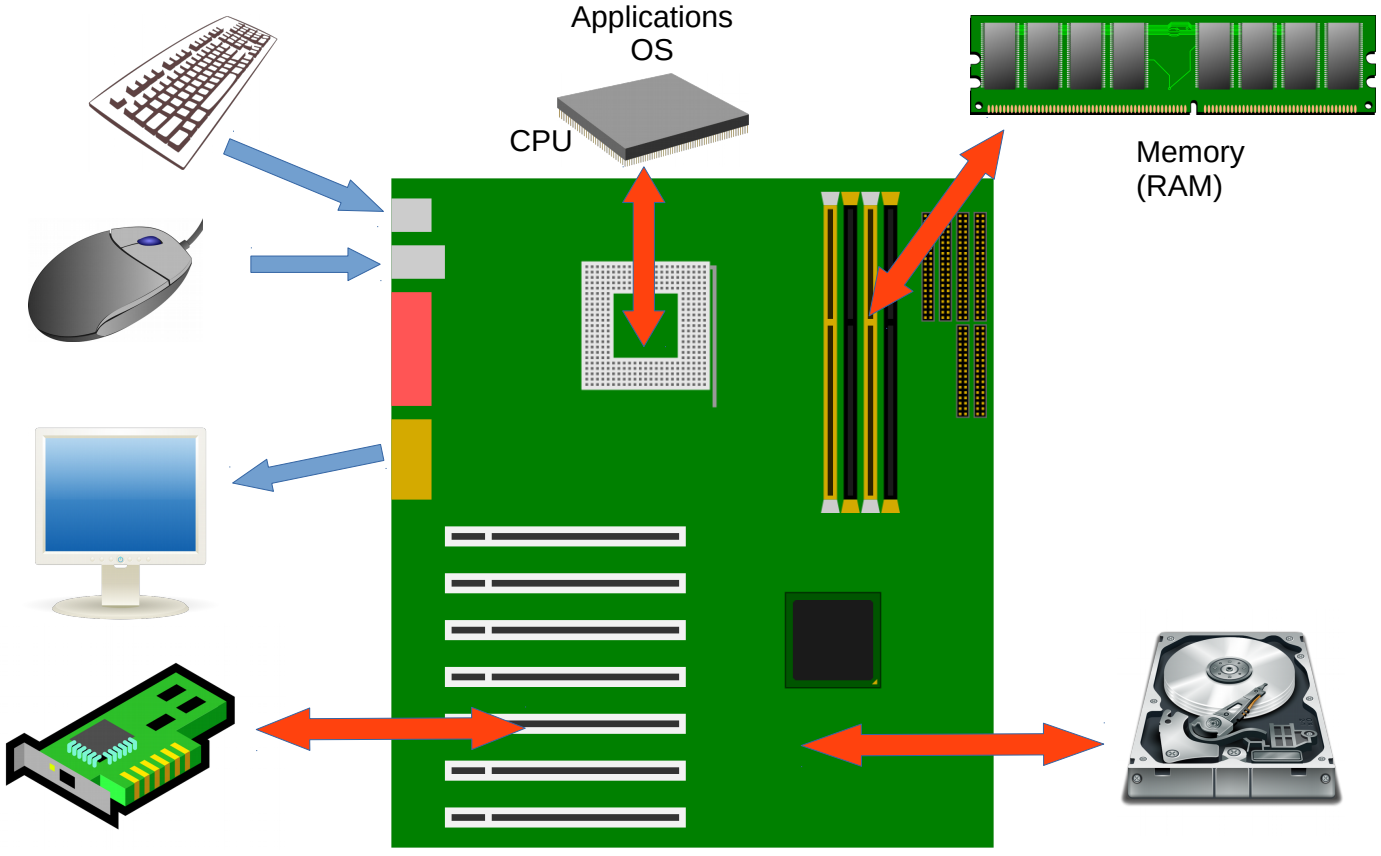
## I. Introduction

Ludovic Apvrille
ludovic.apvrille@telecom-paris.fr
Eurecom, office 470

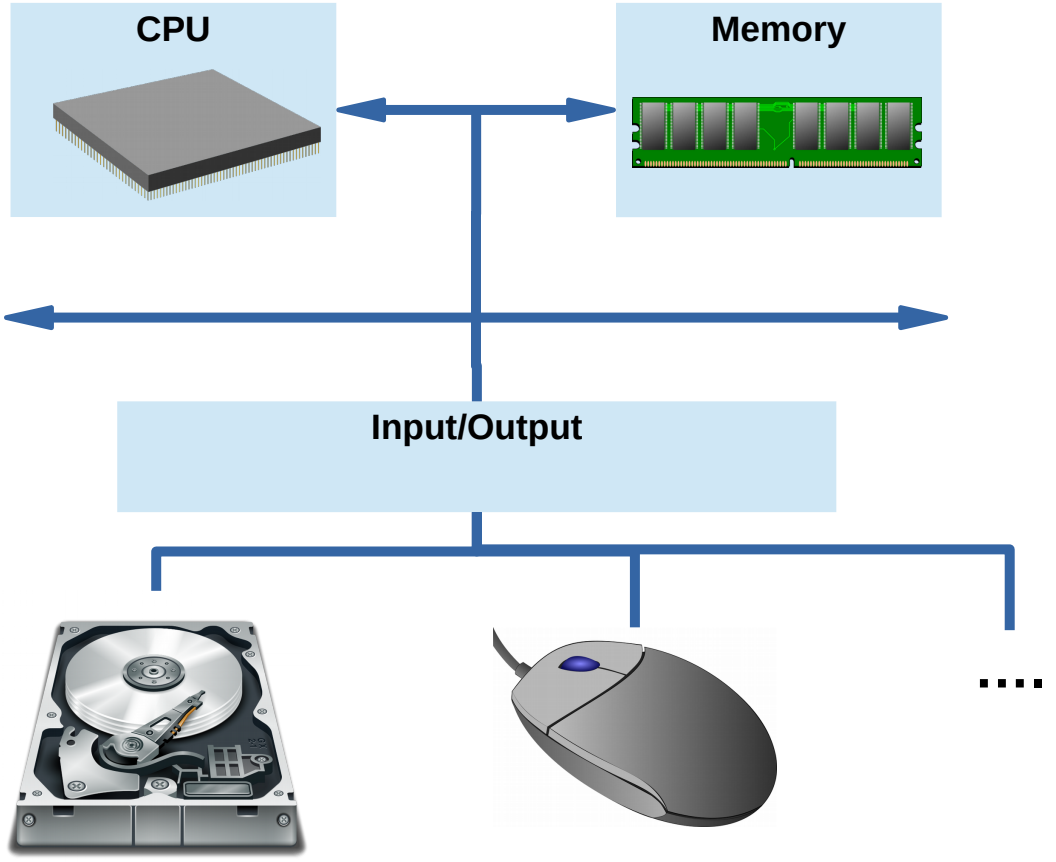
`perso.telecom-paris.fr/apvrille/OS/`

# What is a Computer System?

In other words: what are the main components of a PC?



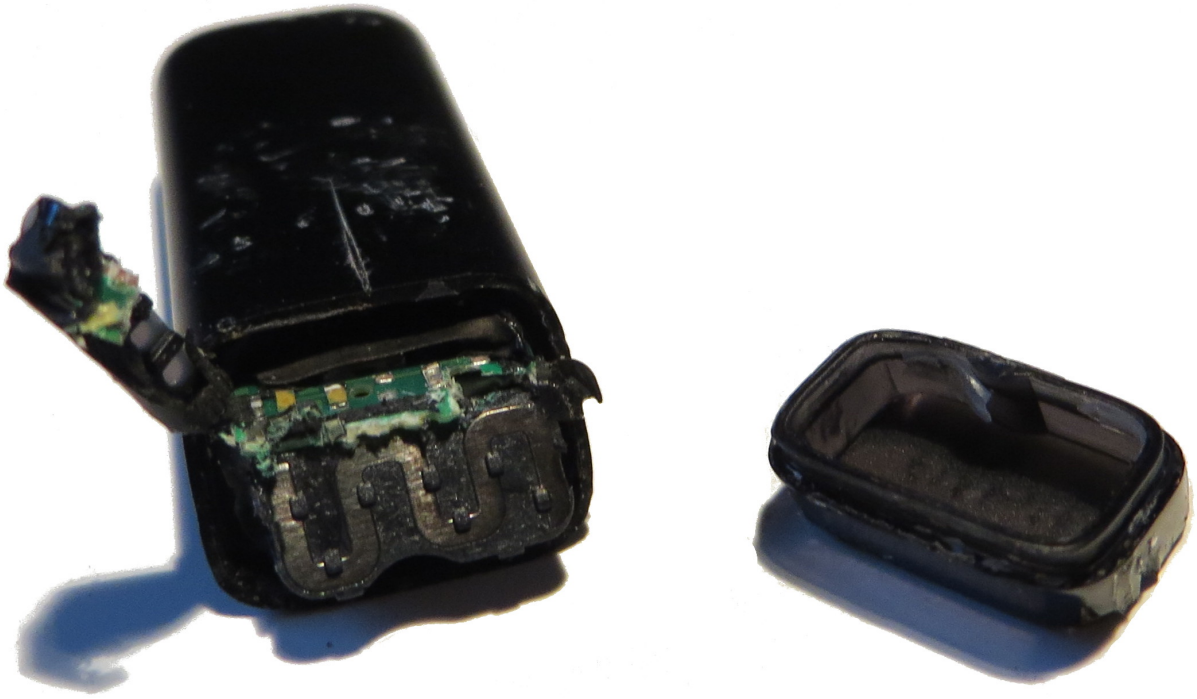
# Computer System: Simplified View



# This is Also a Computer!

**What's inside? Let's see together!**

# Inside a Fitbit

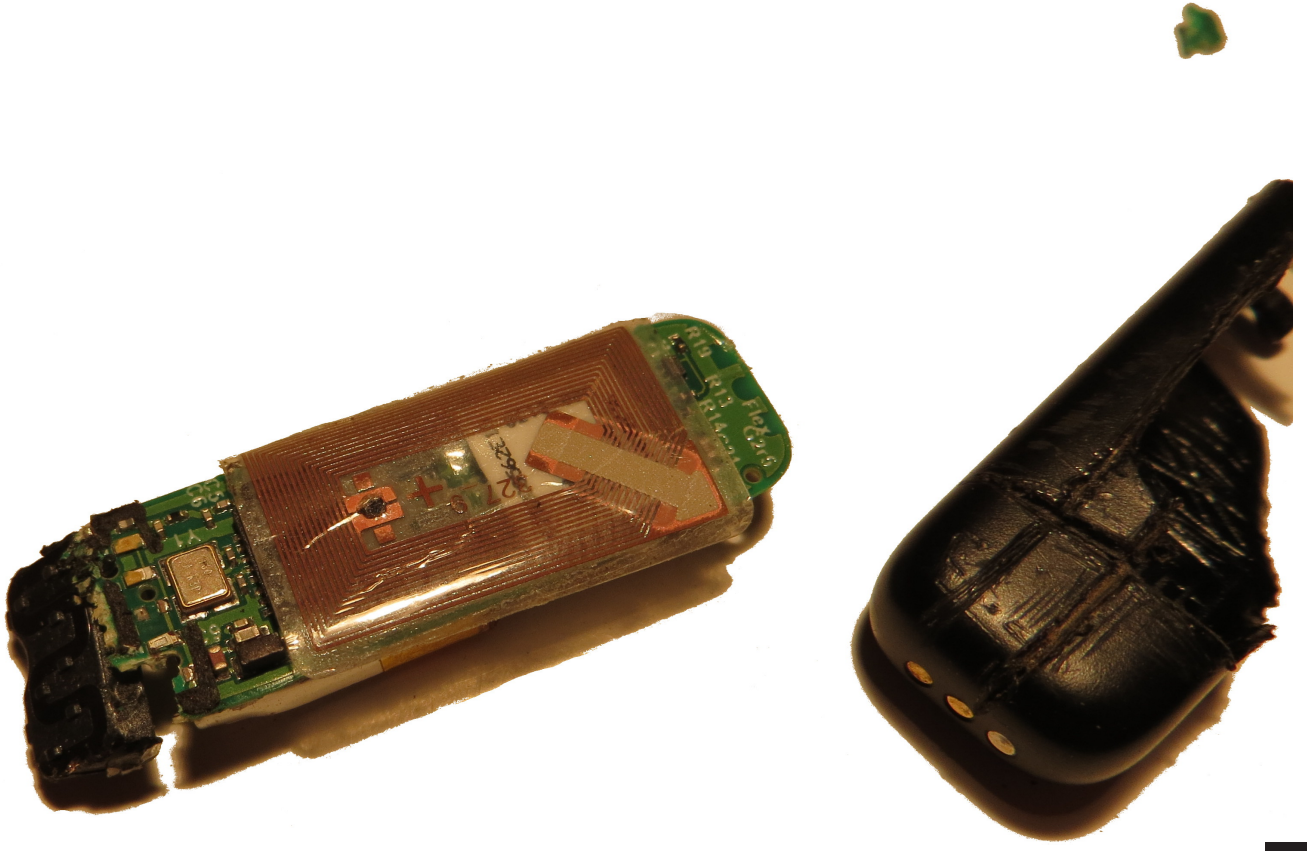


**Don't try this at home!**

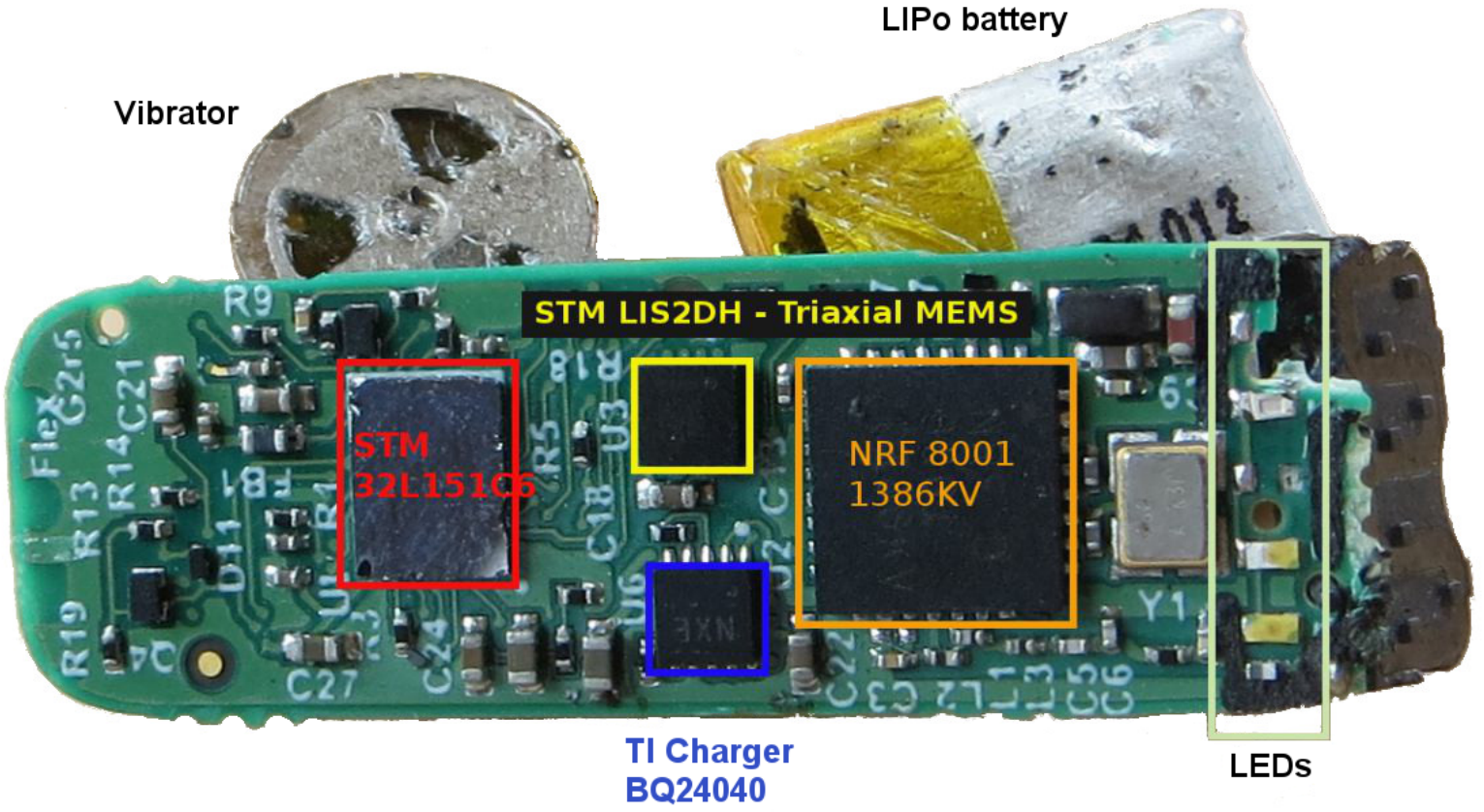
# Inside a Fitbit (Cont.)

**Again: don't try this at home!**

# Inside a Fitbit (Cont.)



# Fitbit: Hardware Components



# What is an Operating System?

## Definition

The most fundamental program of a computer system

## Objectives

- **Make computers convenient to use** i.e. simplify programmers' tasks
  - Abstract hardware concerns
    - e.g., simplify memory allocations
- **Use hardware in an efficient manner**
- **Security**
  - Protect systems from wrong and malicious utilizations

# Layered View of Packaging

GNU/Linux


*File system manager*
*ls*
*firefox*
*Urban terror*
*emacs*
*Kernel sources*
*X Window*
*gcc*
*sh*
*KDE*
*Scheduler*
*libC*

GNU/Linux


- Applications
- Window Manager
- System utilities
- System librairies
- Kernel

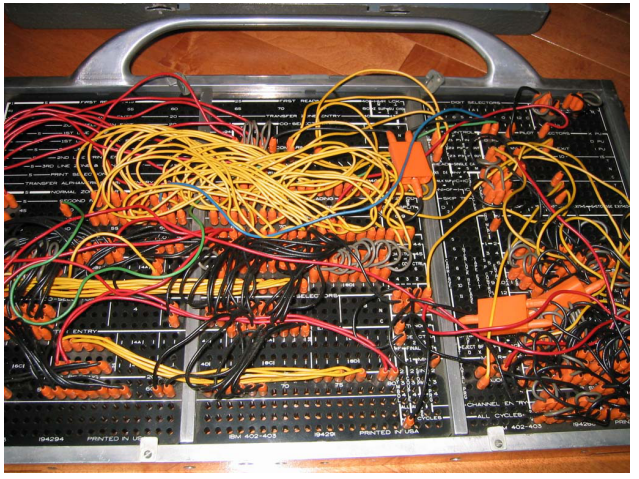
# 1930 → 1965: First Computers

## 1930 → 1955: Vacuum tubes

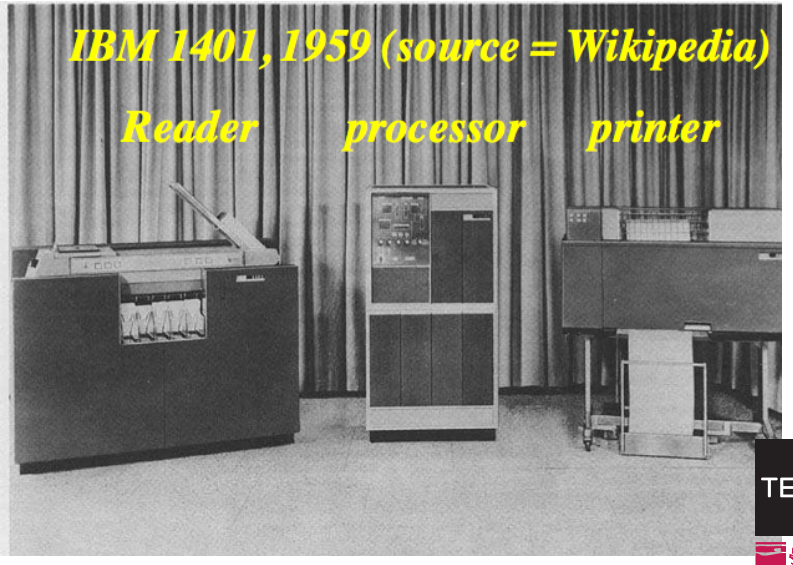
Use of plugboards for programming

## 1955 → 1965: Transistors

- Batch jobs directly written on cards / tapes
  - Assembly language
  - Fortran
- OS
  - FMS - Fortran Monitor System
  - IBSYS - IBM's Operating System



*IBM 402, 1933* (source = Wikipedia)



# 1965 → 1980: Integrated Circuits

- Multiprogramming
  - Can start loading a program while another one is being executed
  - Memory partitioning
- Timesharing
  - CPU partitioning
- Examples
  - IBM OS/360
    - Millions of lines of assembly code
  - MIT CTSS: time sharing system (62) → MULTICS (65) → UNIX
  - Minicomputer: DEC PDP-1 (61)

*Xerox Alto, 1973* *(source = Wikipedia)*

*Apple II, 1977-1988*

*(source = Wikipedia)*

# 1980 → . . . : Large-Scale Integrated Circuits

- Personal Computers, workstations, smartphones
- Graphical user interfaces
- Cloud computing
  - Unawareness of data and jobs execution location
- Real-Time
  - Specialized functionalities for jobs to meet their timing requirements e.g., deadlines
- Examples of OS
  - Solaris, GNU/Linux, macOS, Android, iOS
  - MS-DOS, 3.11, 95, 98, 98SE, ME, NT (NT4, XP, Server 2003, Vista, 7, 8, 10, 11)

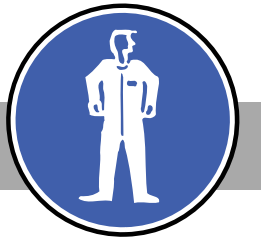
# Main Services

- Program execution
- Resource allocation and release
- I/O operations
- Files handling
- Communication
  - Between programs running on the same computer
  - Between programs running on different computers
- Error detection and handling
  - Hardware failure, illegal memory access, illegal instruction, exception (divide by zero)
- Accounting
- Security (not addressed in this course)

**While ensuring ...**

- Ease of use
- Efficiency
- System protection

# Protection



## Protection of what?

- Hardware
  - Prevent illegal instructions
  - Devices
    - Prevent illegal use of devices
  - Memory
    - Prevent a process from addressing the memory space of other processes and OS
  - CPU
    - Prevent a process from jeopardizing processing resources

## Dual Mode

Protections are based on hardware techniques .... including **Dual Mode**

# Dual Mode of Processors

## User mode

Privileged assembly instructions cannot be executed

- $\rightarrow$ If so, the system "traps"

## Monitor mode

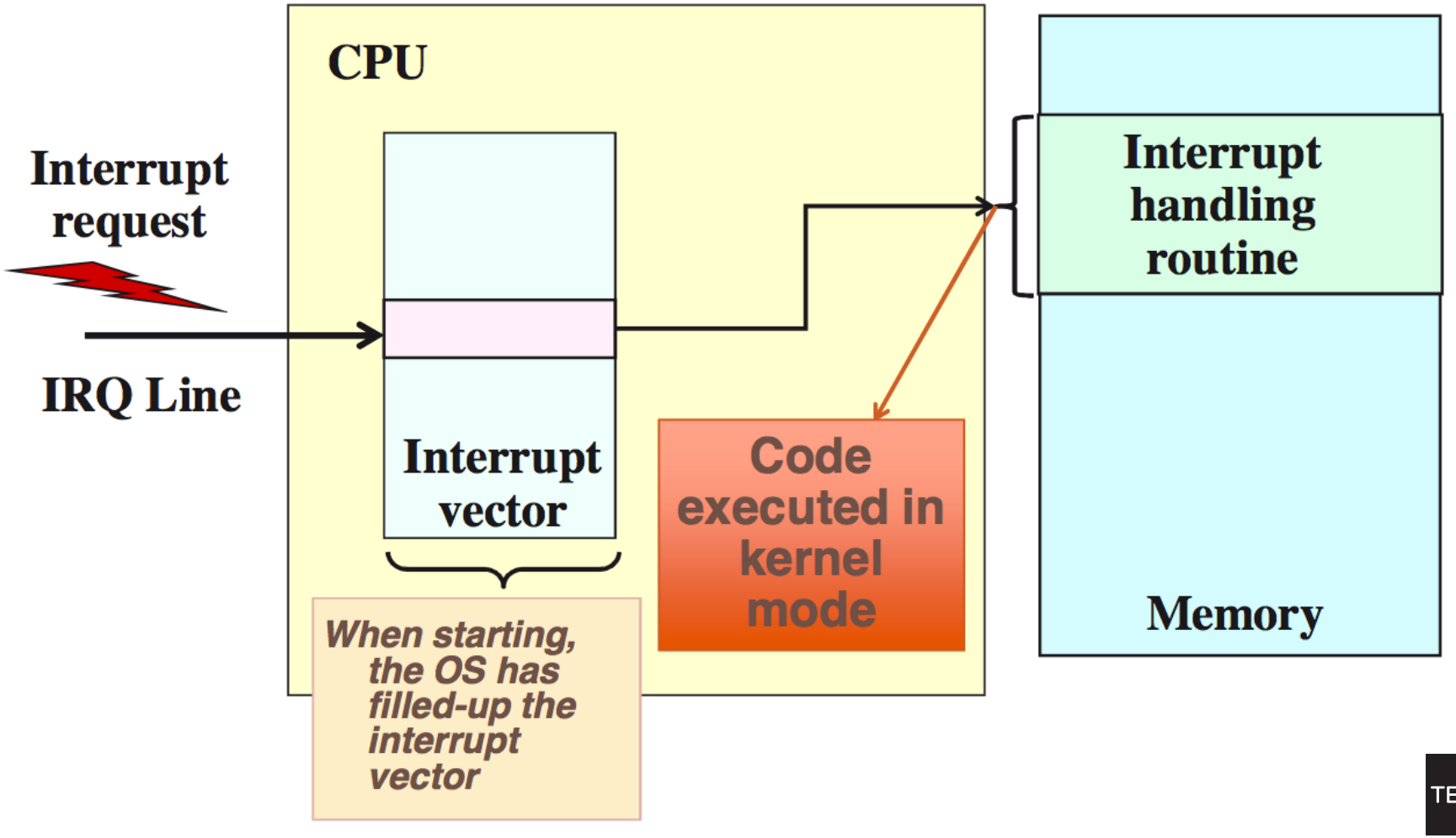
= Supervisor mode, system mode, privileged mode, kernel mode, etc.

- In this mode, privileged assembly instructions can be executed
- **Not** related at all to the *administrator* or *root* of a machine

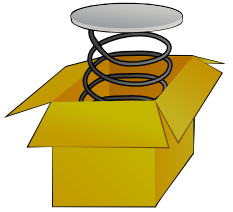
## Mode switching

- Monitor mode $\rightarrow$ user mode: a given assembly instruction
- User mode $\rightarrow$ monitor mode: interrupt (or trap)

# Interrupts


CPU
Interrupt request
IRQ Line
Interrupt vector
When starting, the OS has filled-up the interrupt vector
Code executed in kernel mode
Interrupt handling routine
Memory

# Protection: Use of Dual Mode



1. Hardware starts in monitor mode
2. OS boots in monitor mode
3. OS starts user processes in user mode
   - So, user processes cannot execute privileged instructions
4. When a trap or an interrupt occurs:
   - Hardware switches to monitor mode
   - Routine pointed to by interrupt vector is called
     - Vector was setup by the OS at boot time

So, the Operating System is in monitor mode whenever it gains control i.e., when its code is executed in the CPU

# Hardware Protection

## Goals

Prevent instructions that shall not be executed

- Divide by zero, privileged instruction in user mode, etc.

## Mechanisms

- Hardware detects illegal instructions and accordingly generates interrupts
- The control is transferred to the OS
  - Faulty program is aborted
  - Error message (popup window, message in console or terminal)
  - Program's memory may be dumped for debug purpose
    - Under Unix, it is dumped to a file named *core*
- If faulty element = OS: blue screen, kernel panic, ...

# System Calls (a.k.a. Syscalls)

## Definition

Interface between user processes and the Operating System

- Windows: systems calls are included in the Win32/Win64 API
- Solaris

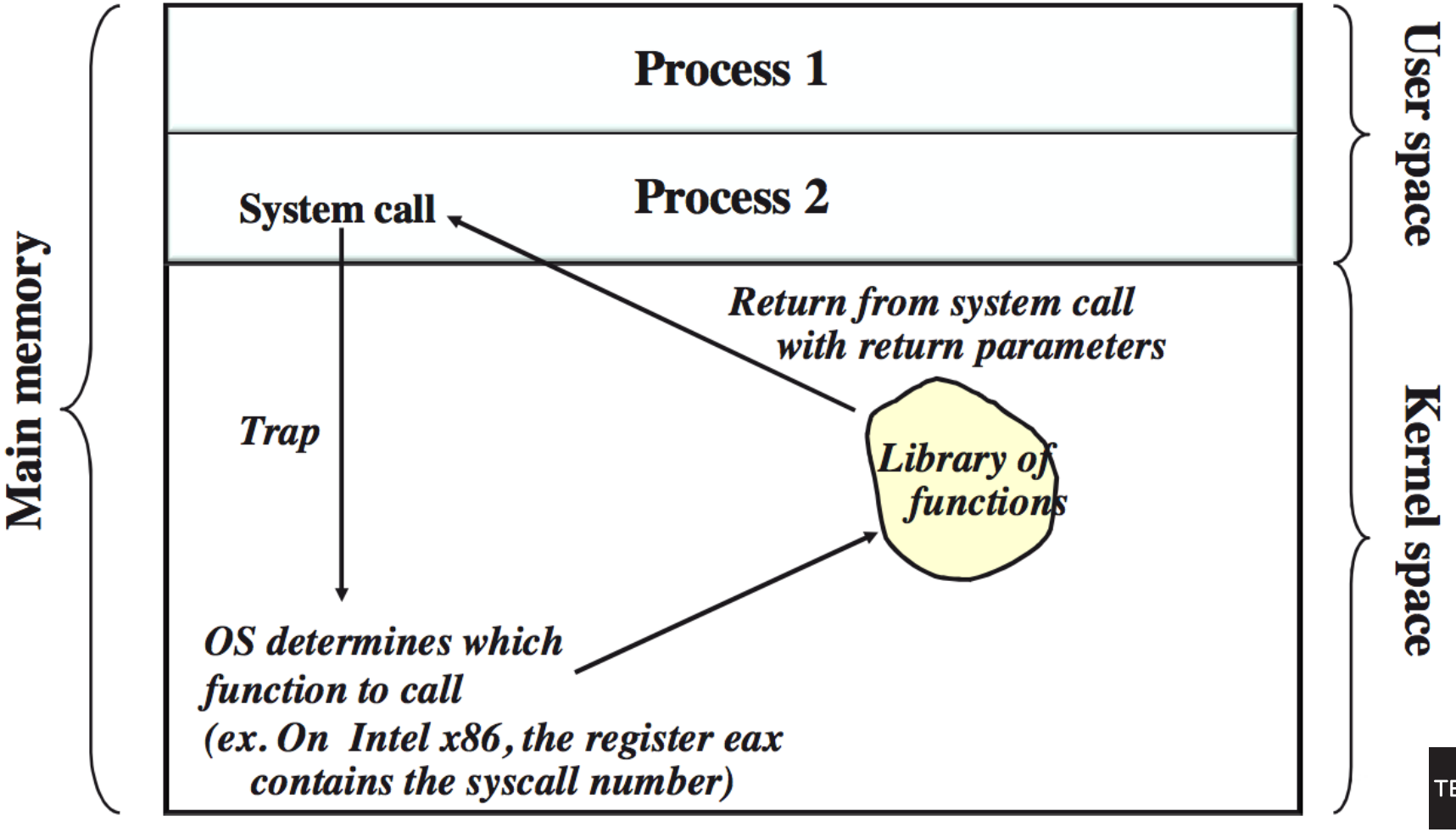
```
$man -s2 read
System Calls
NAME
        read, readv, pread - read from file
SYNOPSIS
#include <unistd.h>
ssize_t read(int fildes, void *buf, size_t nbyte);
...
```

- macOS (Similar result in GNU/Linux)

```
$ man -s2 read
READ(2)  BSD System Calls Manual  READ(2)

NAME
     pread, read, readv -- read input
...
```

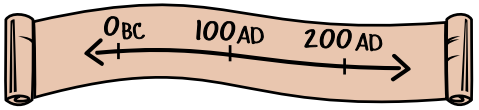
# System Calls: Implementation



# Categories of System Calls

- Process control
  - Create, load, wait event, allocate and free memory, ...
- File manipulation
  - Create, open, close, read, write, attributes management, ...
- Device manipulation
  - Request, read, write, attributes management, ...
- Getting and setting system related information
  - Time management, process management, ...
- Communications
  - Send, receive messages / signals, create communication links, ...

# UNIX: History



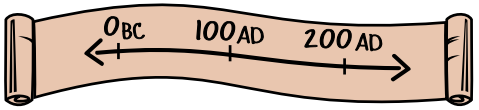

## Idea originated in 1965

- Research lab of AT&T (Bell Labs)
- Idea of Ken Thompson: develop what no computer company was ready to provide i.e. a multi-user and multiprocessing OS
- Multics created in cooperation with MIT and General Electric
- Less complex version of Multics: UNIX, operational at Bell Labs in 1971
  - Fully written in assembly language

## Diffusion in the academic world: 1970 $\rightarrow$ 1972

Code is modified by graduate students to make UNIX more robust

# UNIX: History (Cont.)



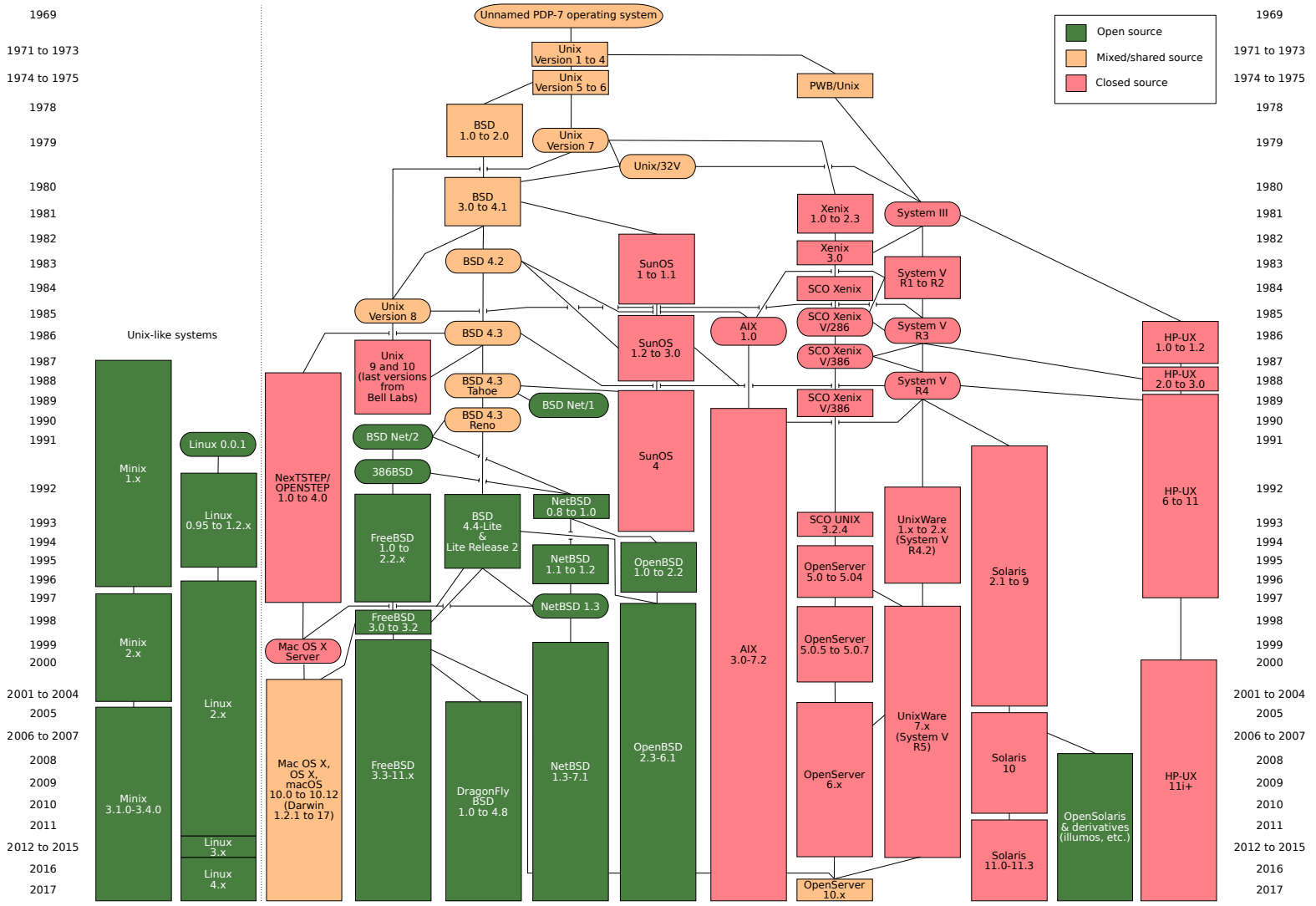

## Improvements: 1973 → 1979

- UNIX rewritten in C
  - C developed by Denn
- BSD Unix (1975): C-shell, virtual memory
- Performance improvements (file systems, etc.)
- Support of more hardware platforms

## Commercial versions: 1979 → now

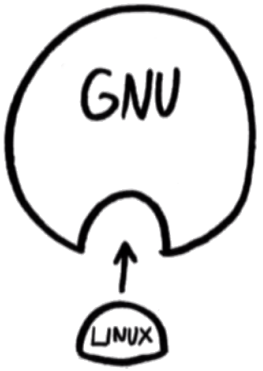
- First commercial UNIX: System III, and then System V
- SunOS/Solaris added networking tools (e.g., NFS)
- Domination of two UNIX: BSD and System V (AT&T)
- Runs on a large majority number of smartphones, tablets, objects, embedded systems

# UNIX: Versions




source = *Wikipedia*

# GNU/Linux (Free Software)





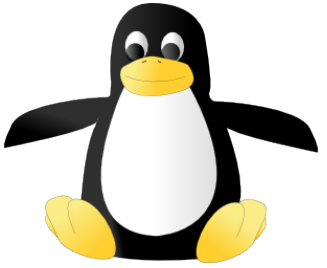
GNU/Linux (a.k.a. Linux) = GNU Operating System + the Linux kernel

## The GNU Operating System

- *GNU's Not Unix!*
- Applications, libraries, and developer tools
- Started in 1984

## The Linux Kernel

- Created in 1991 by Linus Torvalds
- *see next slide*



# First Post by Linus Torvald

comp.os.minix ›

## What would you like to see most in minix?

285 posts by 262 authors G+1

**Linus Benedict Torvalds**

Hello everybody out there using minix -

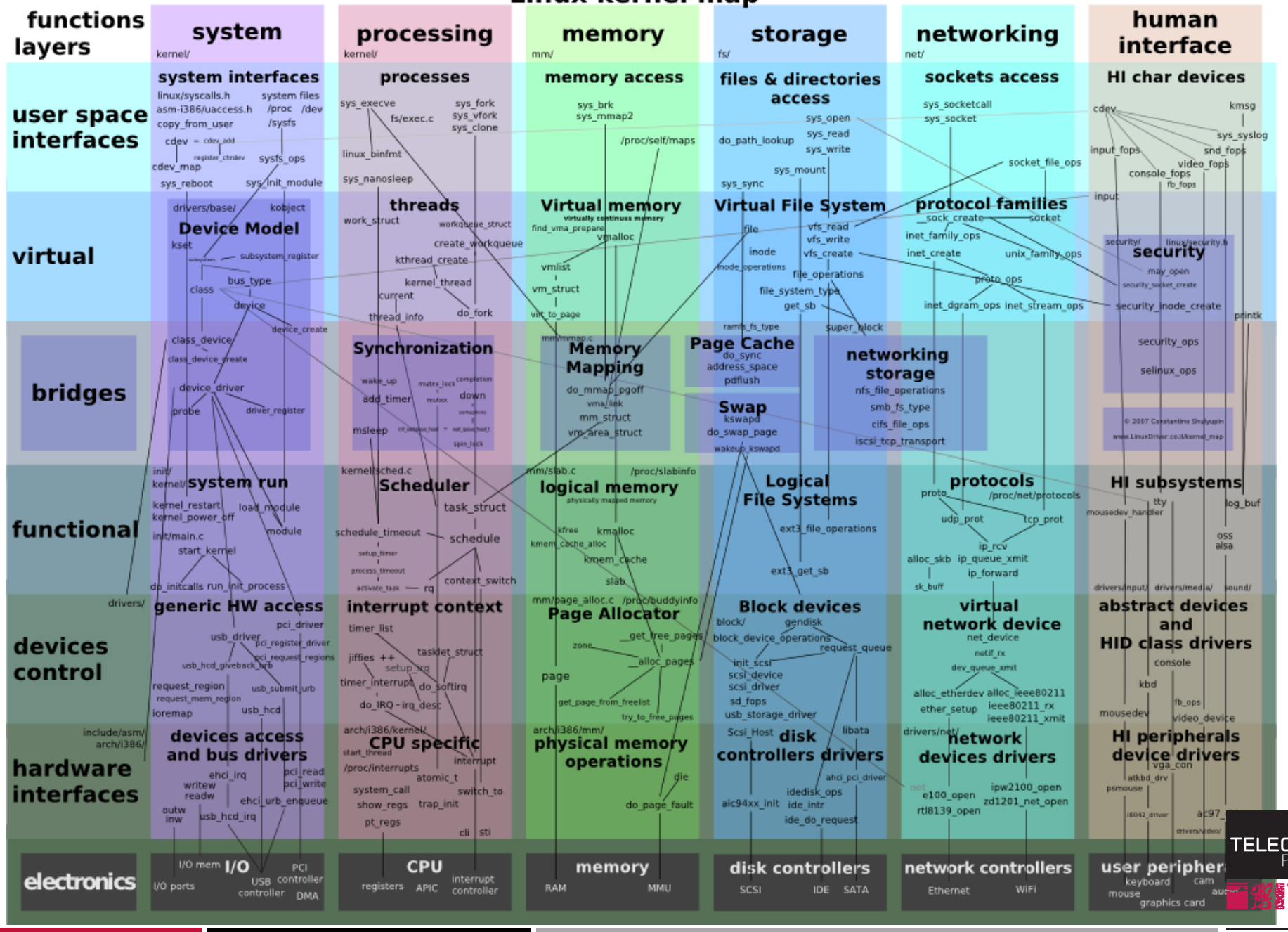
I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

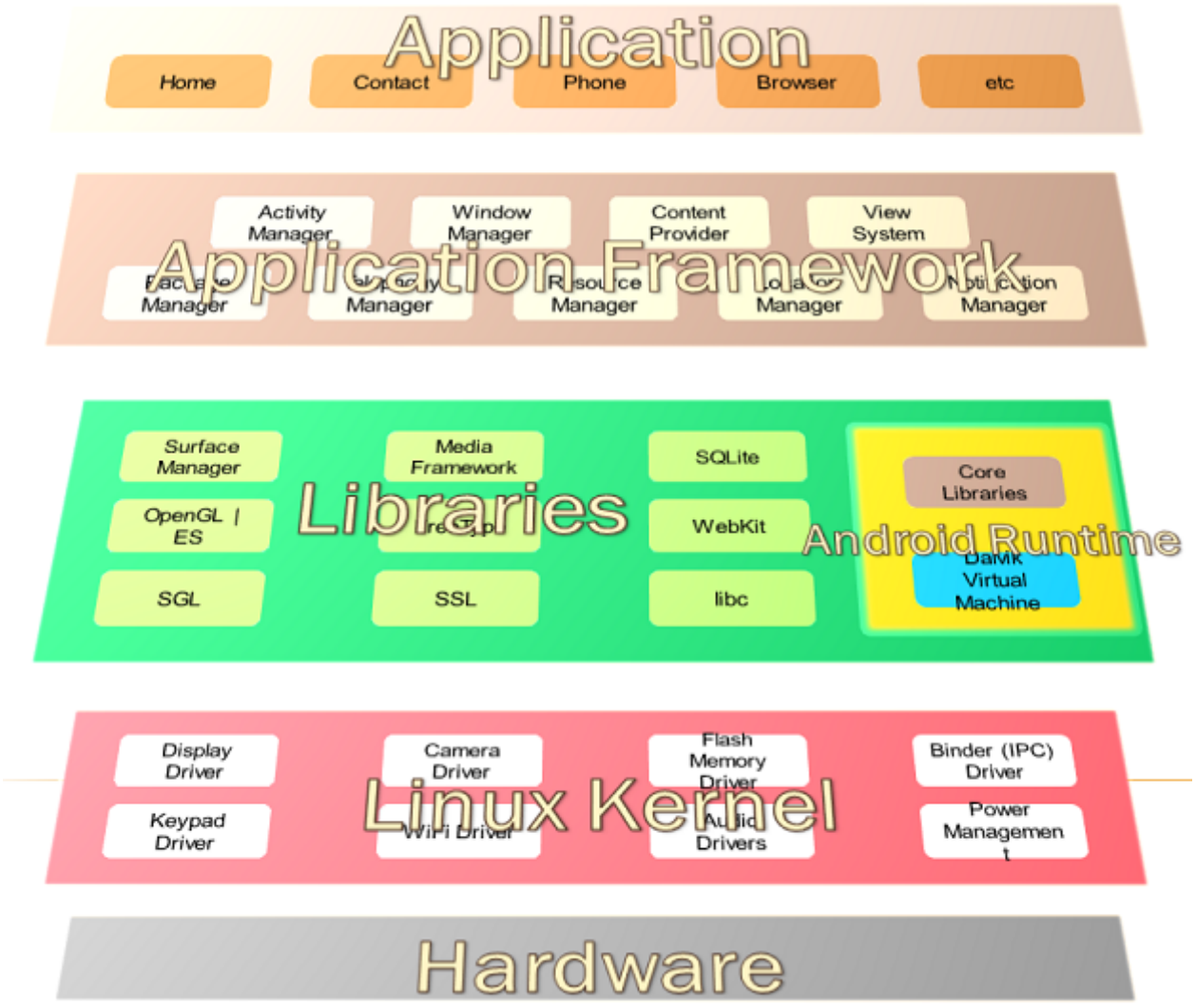
Linus (torv...@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

# Linux kernel map

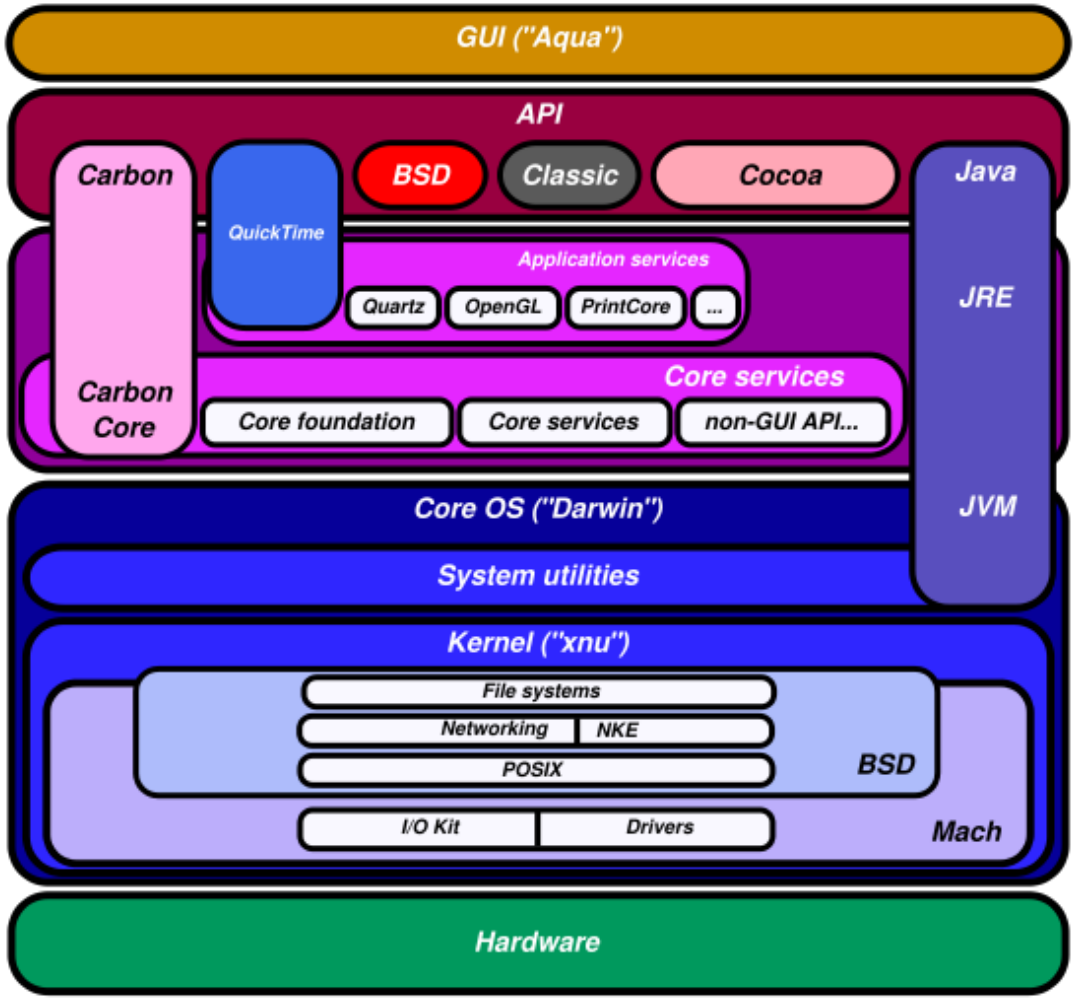


# Android



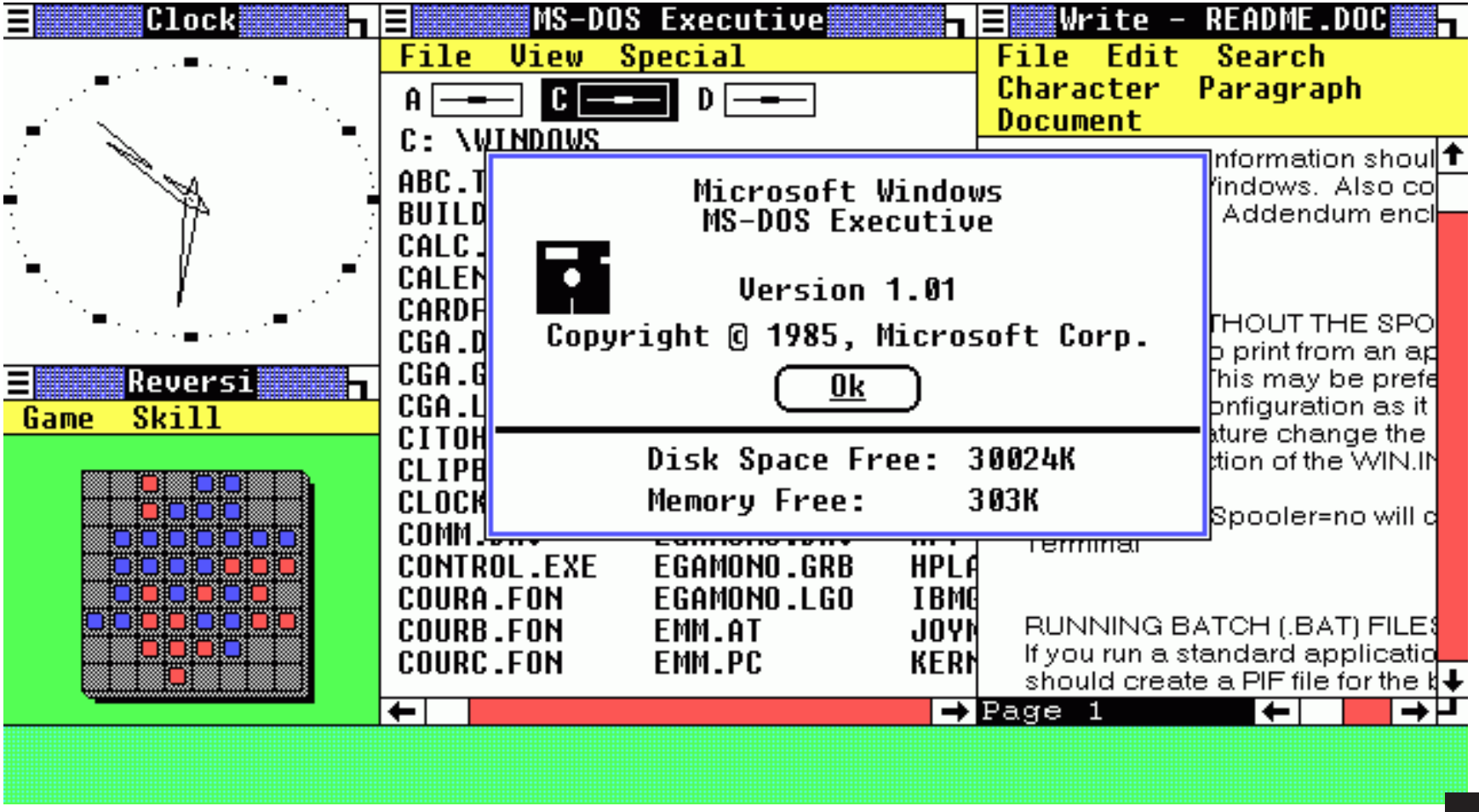

source = *Wikipedia*

# macOS



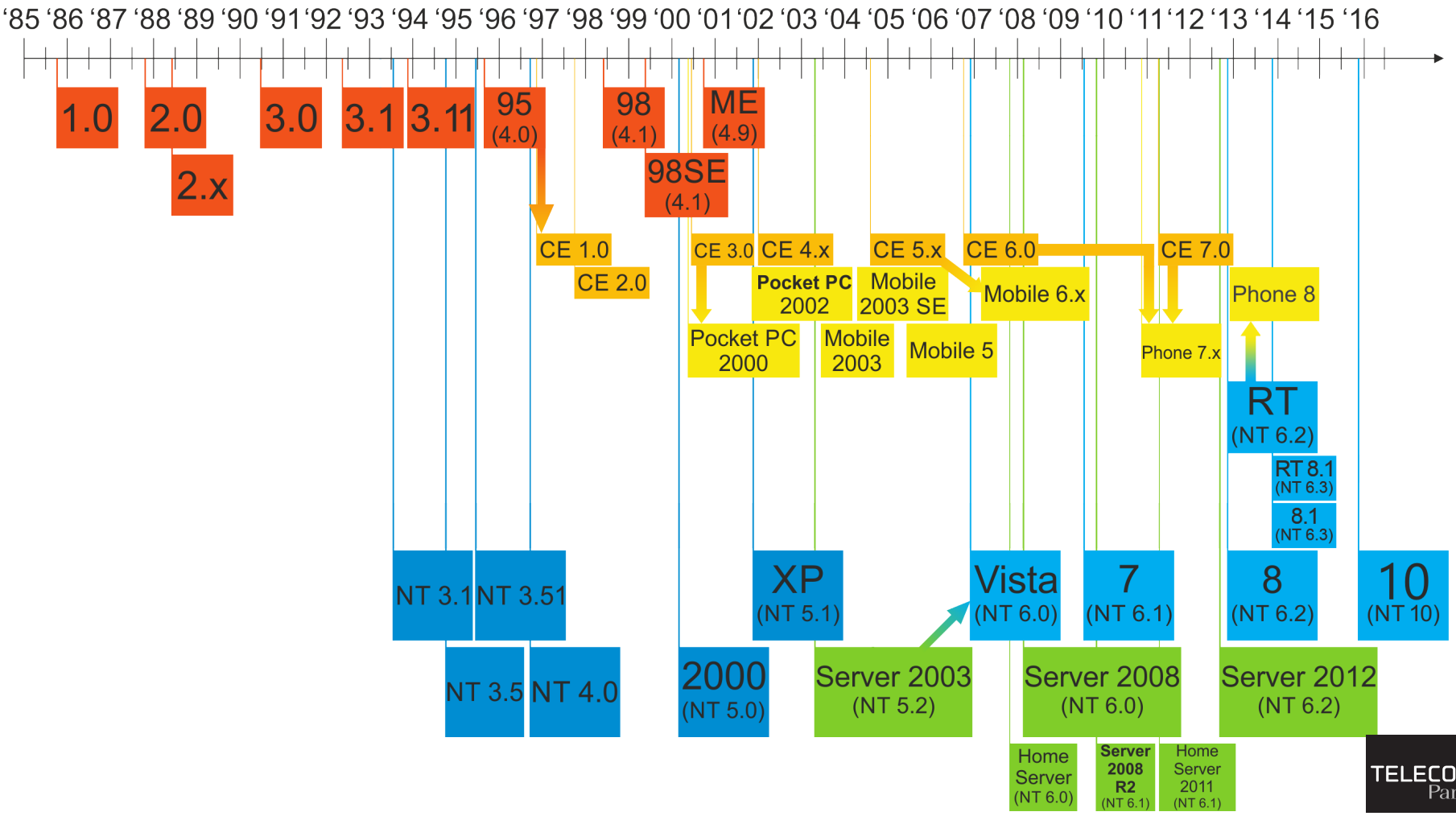

source = *Android sources*

# Windows 1!

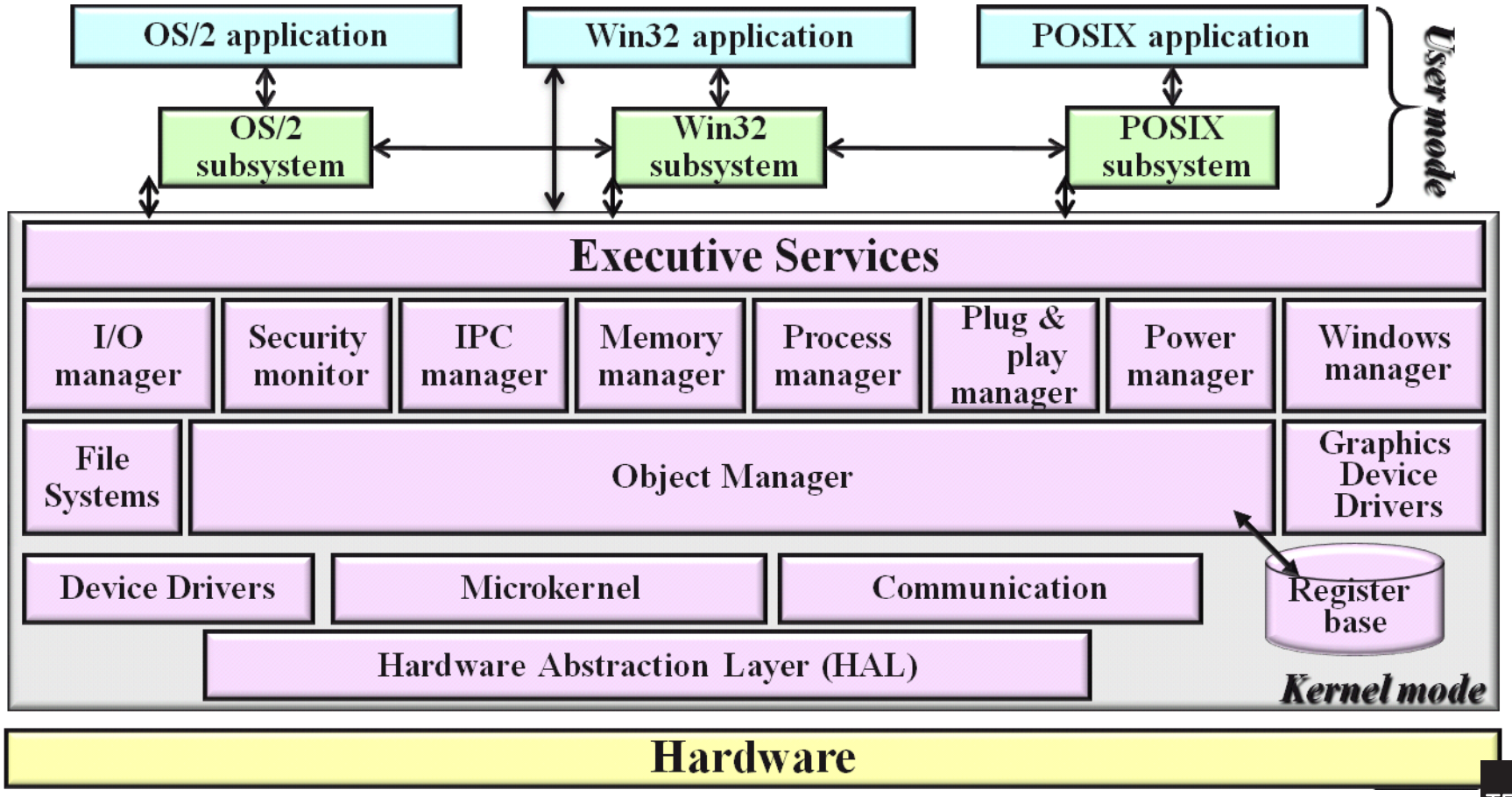



source = *Wikipedia*

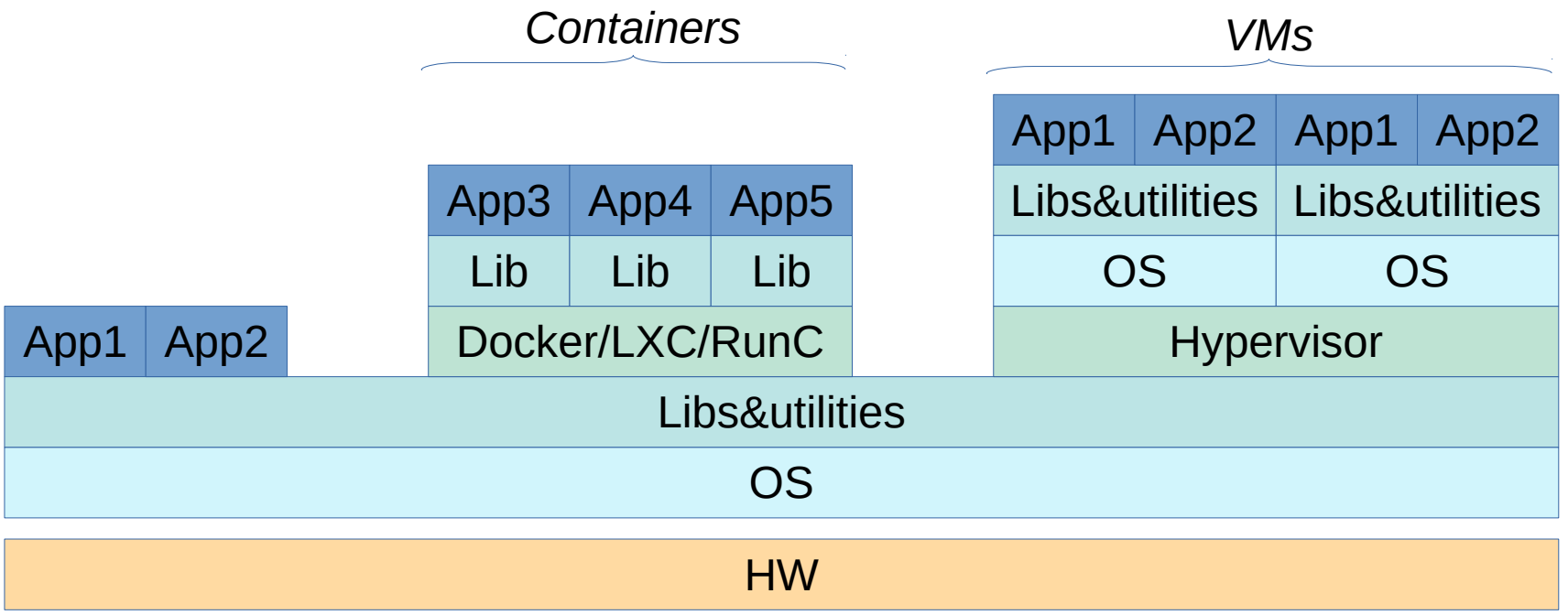
# Windows: Chronology



# Windows NT Architecture (2000/XP/Vista/7/8/10)



# Containers and Virtual Machine Approach



# Containers and Virtual Machines

## VM

Completely isolated guest operating system installation within a normal host operating system (source: *Wikipedia*)

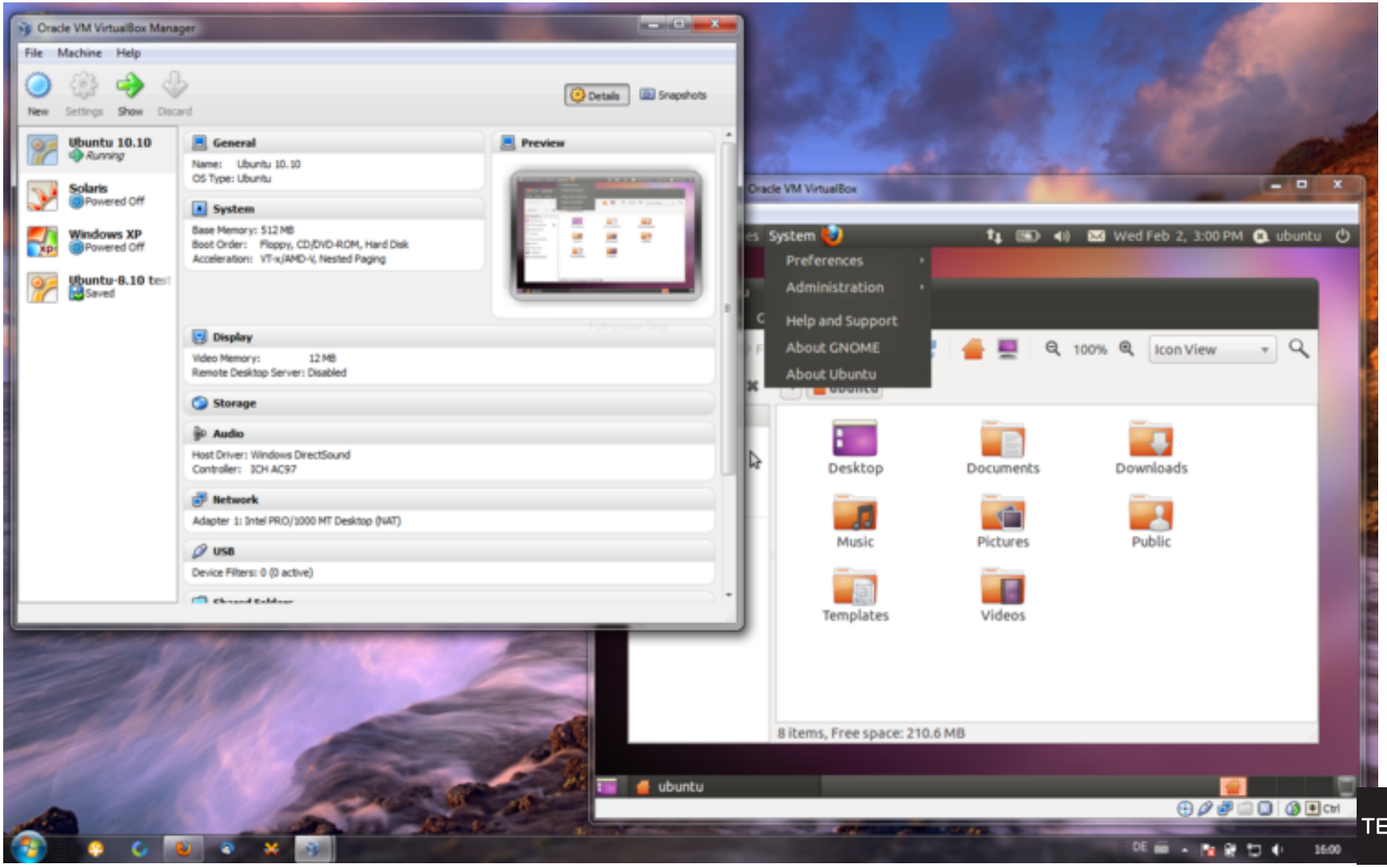
- The underlying layers of a virtual machine are considered as bare hardware
  - i.e., a guest OS thinks it is running alone on the machine
- The interface offered by a virtual machine is identical, whatever the underlying layers

## Container

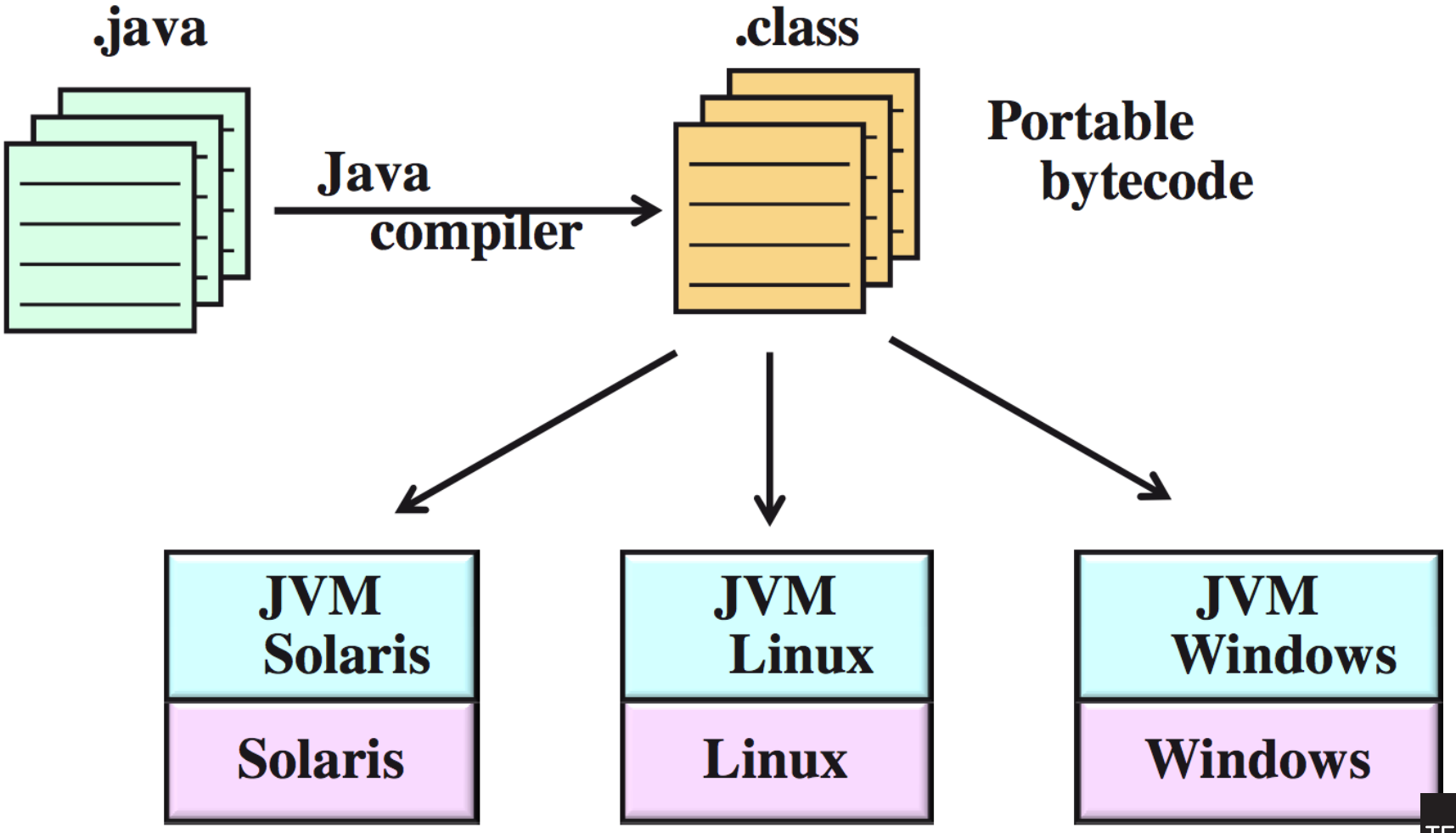
Package of an application and all its dependencies so as to seamlessly exexute this application in any (Linux) environment and isolate this application from others

- Reuses as much as possible resources of the OS

# Example #1: *VirtualBox*



# Example #2: *Java Virtual Machine*



# Inside the Java Virtual Machine