# Operating Systems

## Kernel, System Calls and Processes

Guillaume Salagnac - Lionel Morel

Insa de Lyon – IST OPS

2023–2024

# Some definitions

User = the human in front of the computer

- might be : a “final” user, a developper, depending on context
- interacts directly with the hardware (through screen, keyboard, microphone, etc)

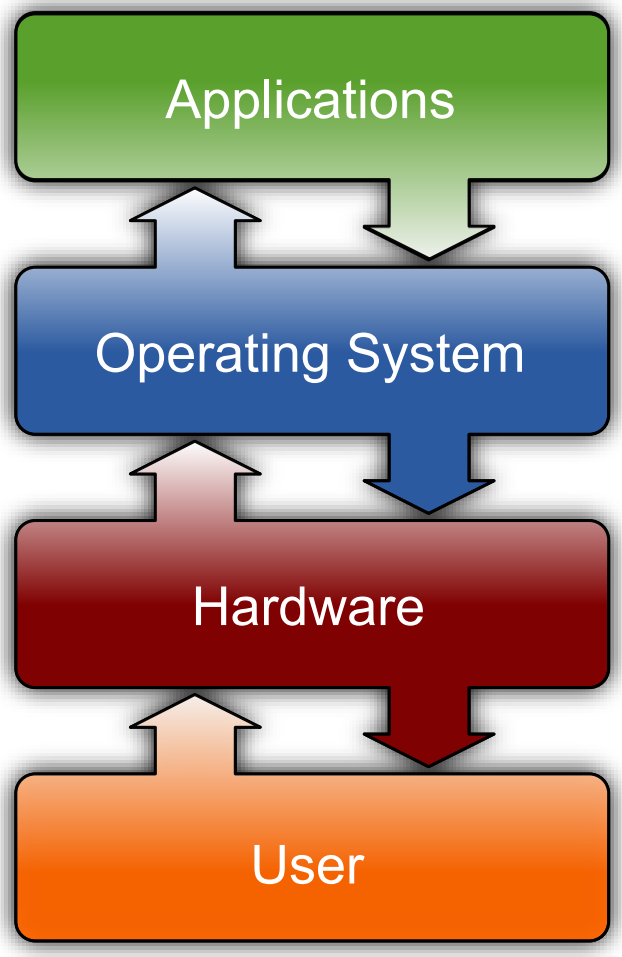
Applications = software with which the final user wants to interact

- messanging app, text processing, music player, web browser, etc.

Hardware = the physical machine

The Operating System is everything else :

- all the infrastructure software : “kernel”, “drivers”, “services”, etc.
- “between the hardware and the applications”



# Role of the OS : two fundamental functions

et largely inter-dependant !

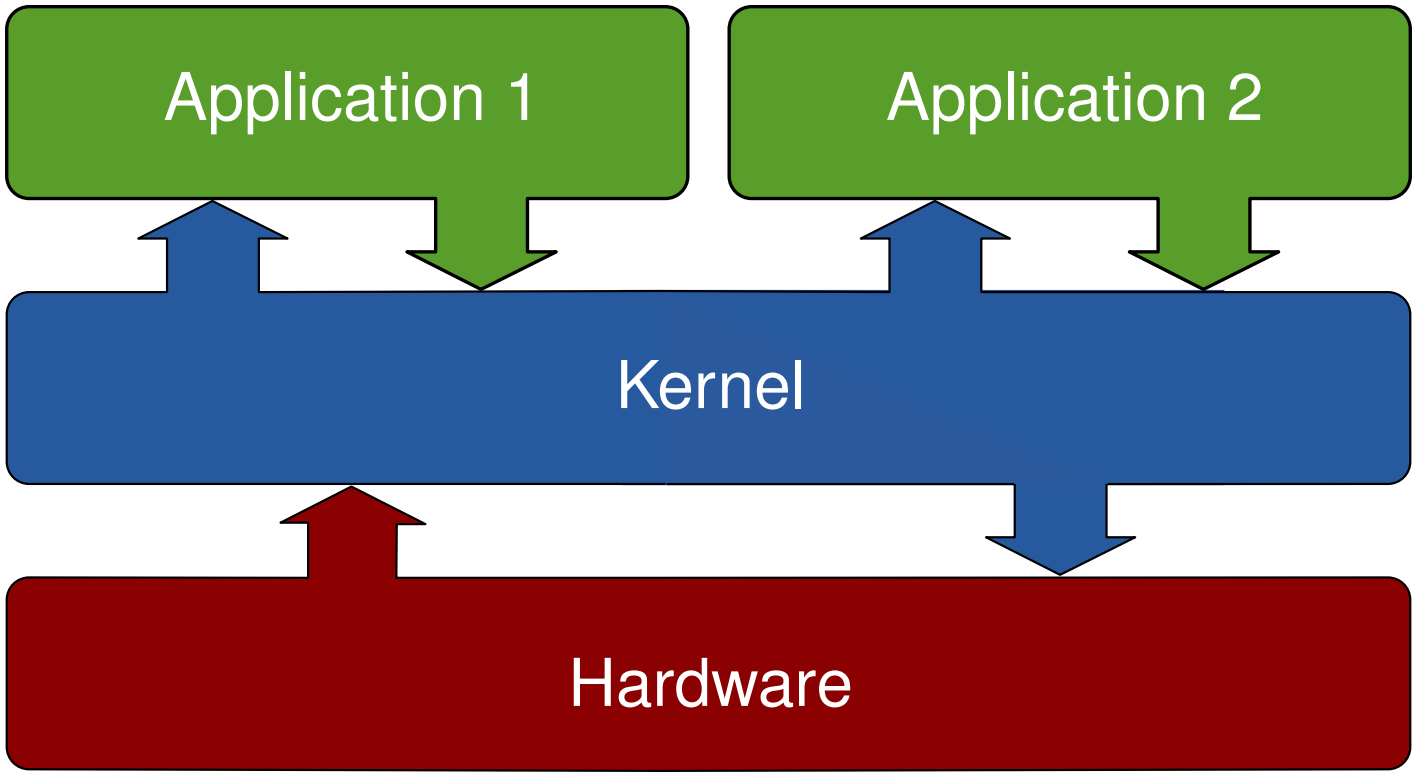
## Virtual Machine

- hides the complexity (of the hardware) under a “nicer” interface
- provides some base services to applications
  - HCI, persistent storage, network access, time management
- allow for the portability of programs
  - make it possible to execute the same program on different hardware

## Resource management

- share resources amongst applications
- exploit available resources
- protect applications and the system itself

# The OS and its kernel




**Definition : the Kernel**

The kernel is that par of the Operating System that is not an application

# The OS is a “nice” interface to HW for applications



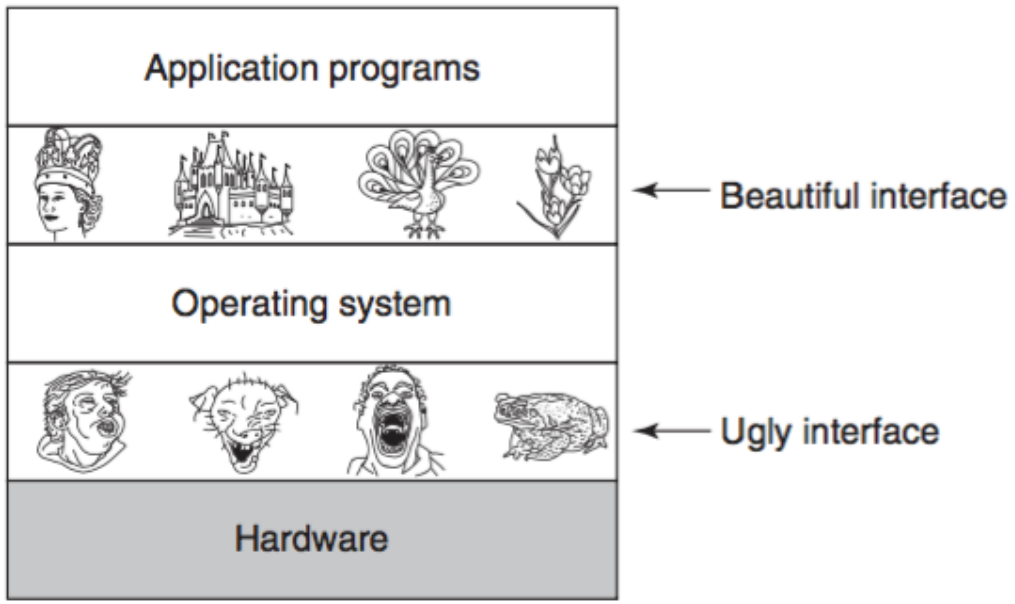
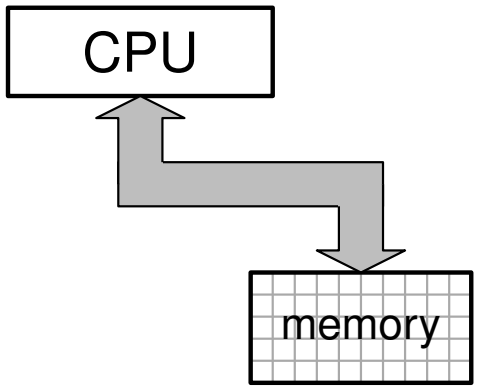



**Figure 1-2.** Operating systems turn ugly hardware into beautiful abstractions.

source : Tanenbaum. *Modern Operating Systems (4th ed, 2014)*. page 5

# Applications use the CPU in “restricted mode”




**Reminder : le cycle de Von Neumann**

while True do :

Fetch an instruction from “memory”

decode its bits : what operation, what operands, etc.

execute the operation and store its result

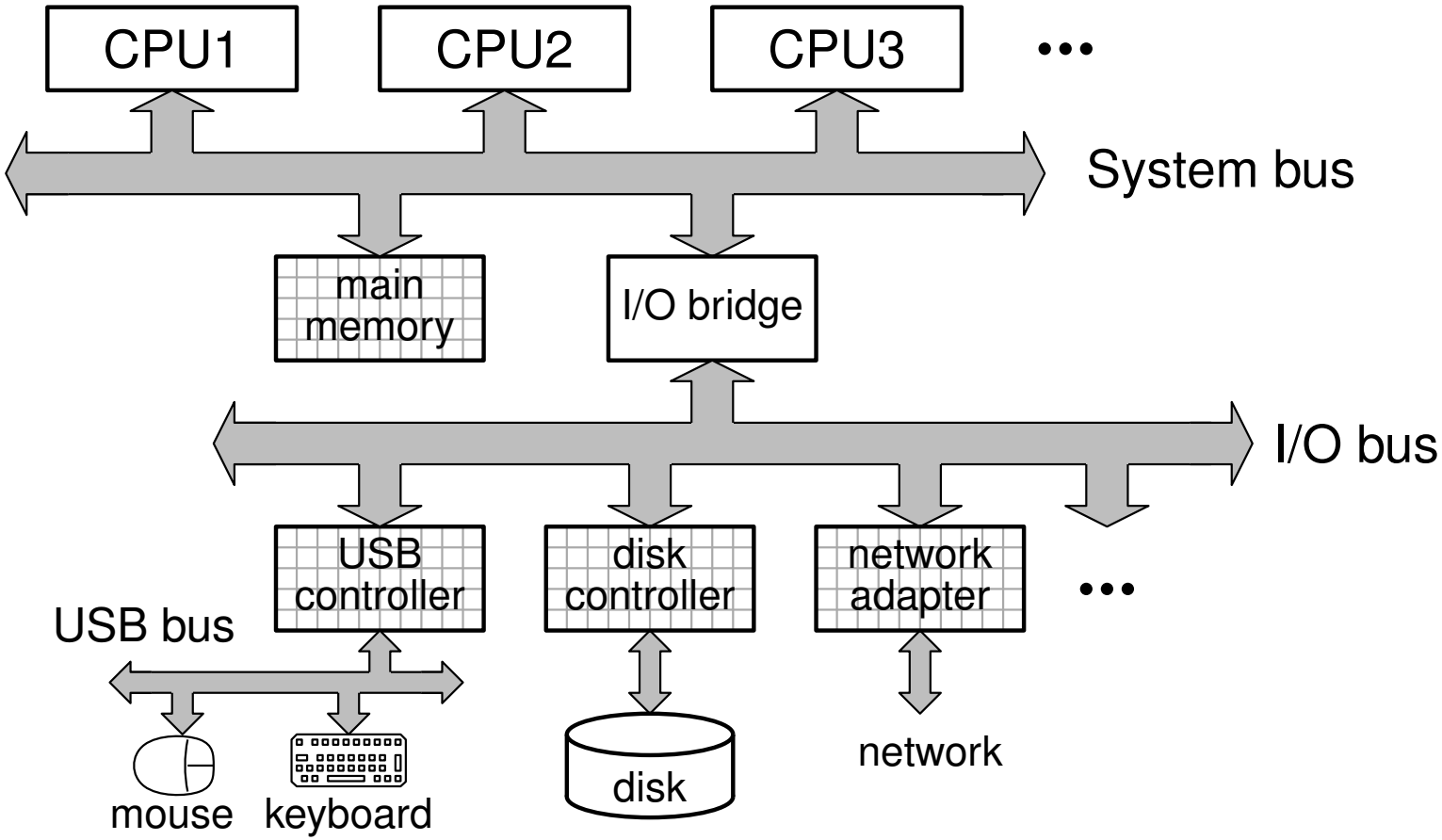
repeat

**Definition : restricted mode = slave mode = ring 3 = user mode**

- the application has a partial view of the machine : 1 CPU + 1 mémoire
- some instructions and some addresses are forbidden
- useful to execute application code without fearing to break anything

- available instructions : ALU operations, memory reads and writes, jumps

# Kernel = CPU use in “supervisor mode”




**Definition : supervisor mode = ring 0 = kernel mode = privileged mode**

- direct access to hardware : needed for executing kernel functions and drivers as well
- SW$\mapsto$HW = Memory-mapped I/O  HW$\mapsto$SW = Interruptions

- NB : this is the default mode when the machine is booted

# Changing the execution mode : traps

Problem : how can an application invoke a function of the kernel ?

BAD solution : allow application to jump to functions that are inside the kernel.

- jump destination can be chosen arbitrarily by application ▶ security breaches
- At some point, we need to move from restricted to supervisor mode ▶ when ? how ?

Solution : provide a dedicate CPU instruction

- examples : `TRAP` (68k), `INT` (x86), `SWI` (ARM), `SYSCALL` (x64)
- software interrupt = trap = exception
- how it works :
  - save the CPU context (register content)
  - switch to supervisor mode
  - jump to an address in the kernel code. The address is pre-determined, and well know (solves the security breach problem)

# System call : principle

**system call = syscall**

Function located in the kernel, invoked by a user process through a trap

Application-side :

- the call is invoked with a `TRAP` instruction
- independant from the programming language used
- generally encapsulated inside library functions (eg : libc)

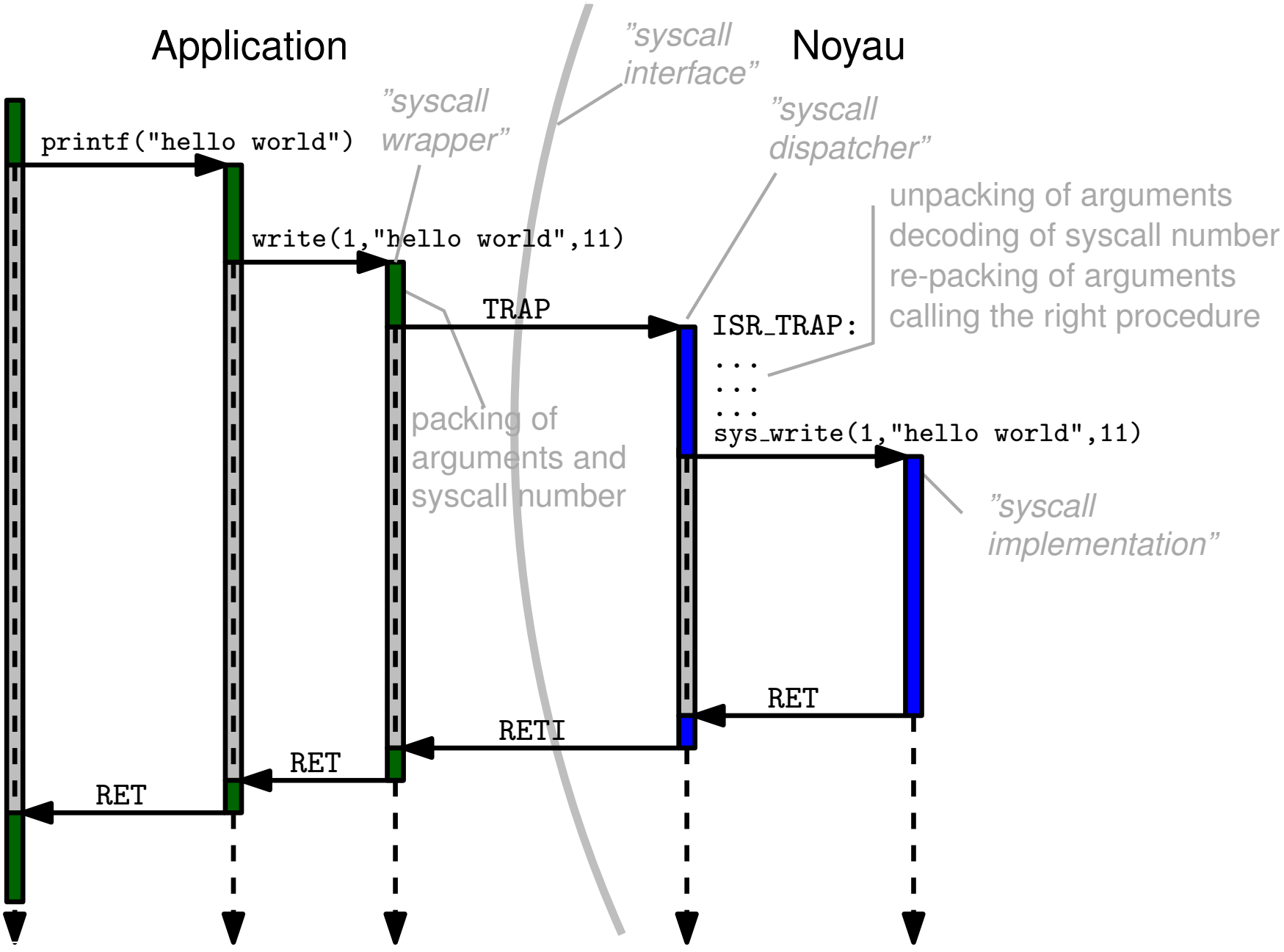
Kernel-side :

- The `TRAP` instruction makes the CPU jump into a dedidated routine (an ISR)
- that itself calls the “right” function, corresponding to the desired system call
- finally (when the system call is finished) hands the CPU back to the application, through a `RETI` (return from interrupt) instruction.

# Example system calls

- `read()`, `write()`, `fork()`, `gettimeofday()`...
- several hundreds of syscalls in linux

# System call : progress



# Processes

Applications are executed on the “userland virtual machine” :

- restricted instruction set (CPU in user mode)
  - no access to low-level HW mechanisms (interrupts, MMIO)
- memory read/write forbidden to some addresses
  - eg : code and data of the kernel, peripherals

Protection by «sandboxing» : a new instance of virtual machine is created for each application that starts execution

**process**

“A program during its execution”

The operating system is both an illusionist (VM) and a sub-contractor (HW)

# Processes : remark

Intuitions :

- a process = a program + its execution state
- execution state = values contained in registers + memory

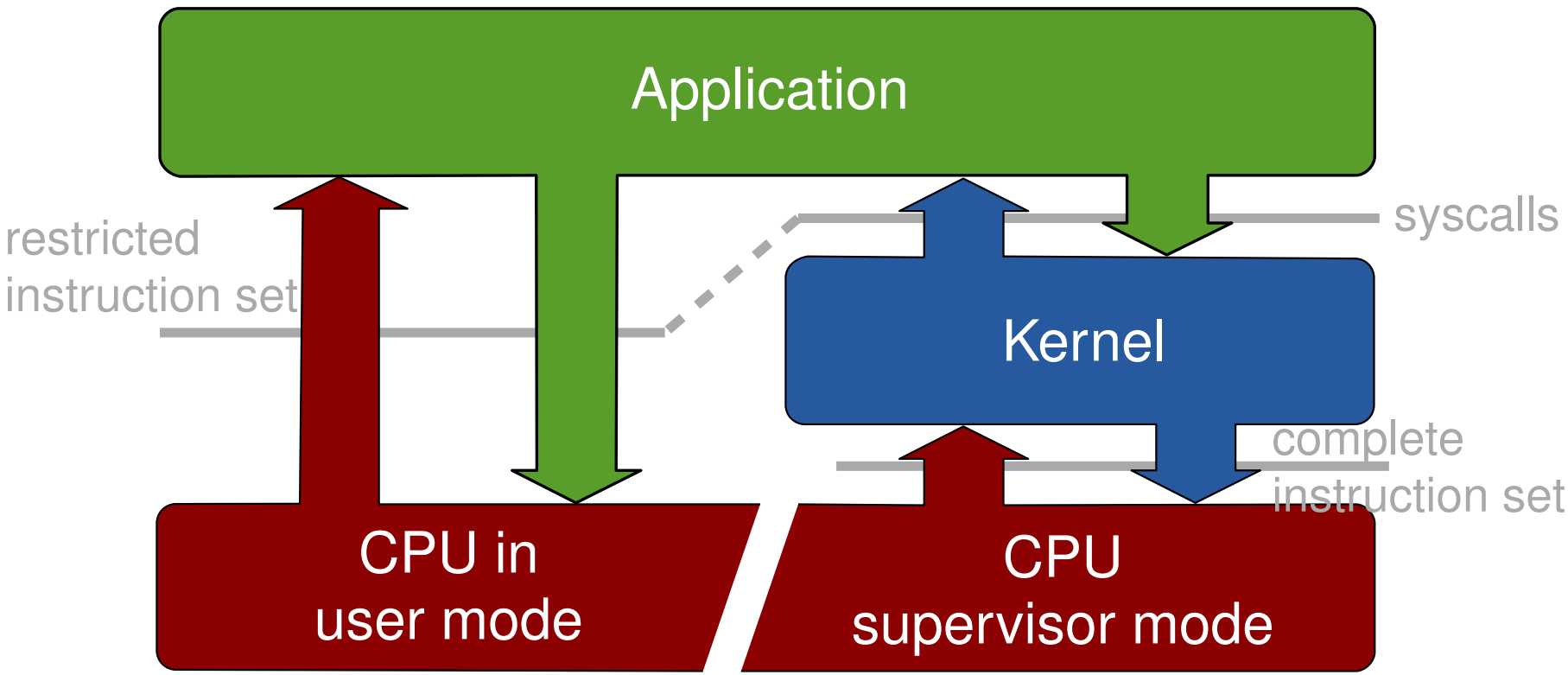
The kernel :

- shares hardware resources amongst processes
- creates/recycles processes when needed
  - in the kernel : a Process Control Block for each living process
  - PCB = id card of the process
  - contents (amongst other things) : (PID) number, list of open files...

Homework :

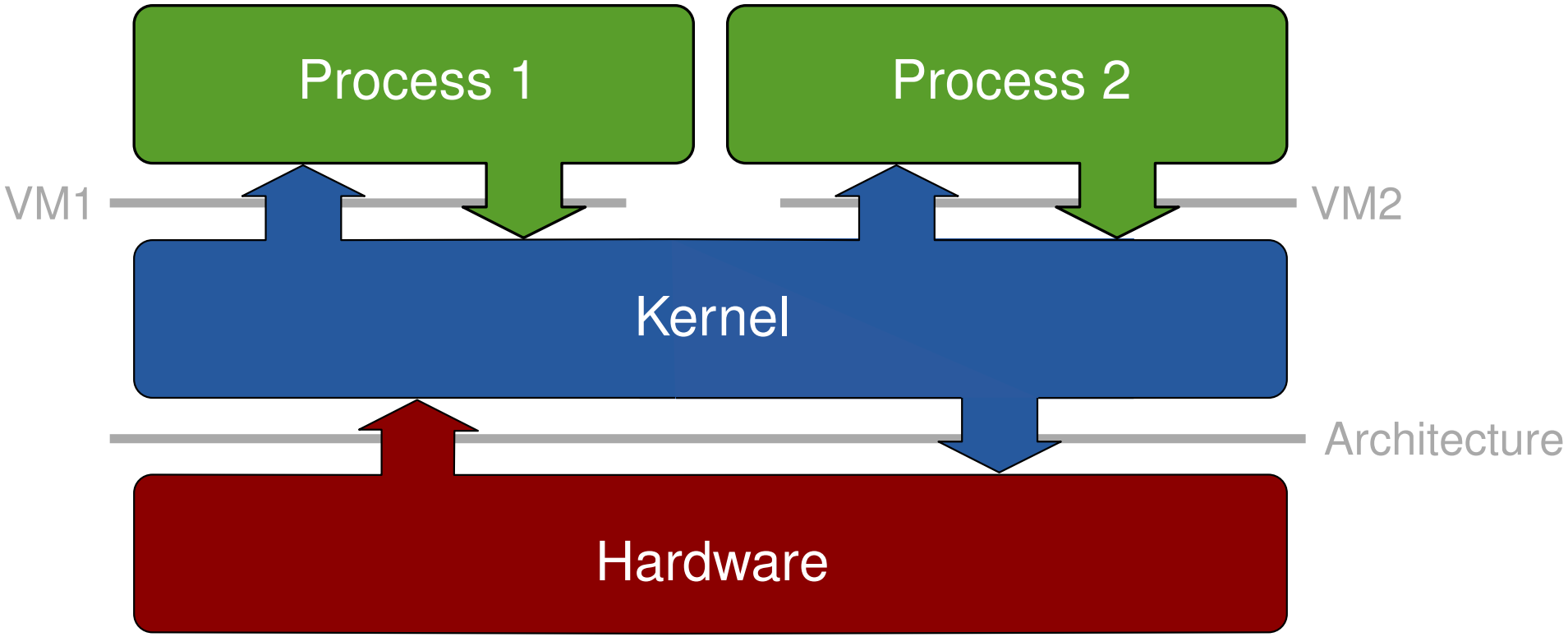
- try the commands `ps aux` and `top`
  - then `man ps` and `man top`
- Finally try : `strace ./monprogramme`

# The userland VM




- application code executed by CPU in user mode
- to make a call to the kernel : user the system call interface

# Where does the OS stand....




- each application that executes is in a userland process
- The kernel virtualises and manages accesses to the HW