UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Overview

COMPUTER ARCHITECTURE AND OPERATING SYSTEMS

2025/26 Spring Term

Jordi Fornés

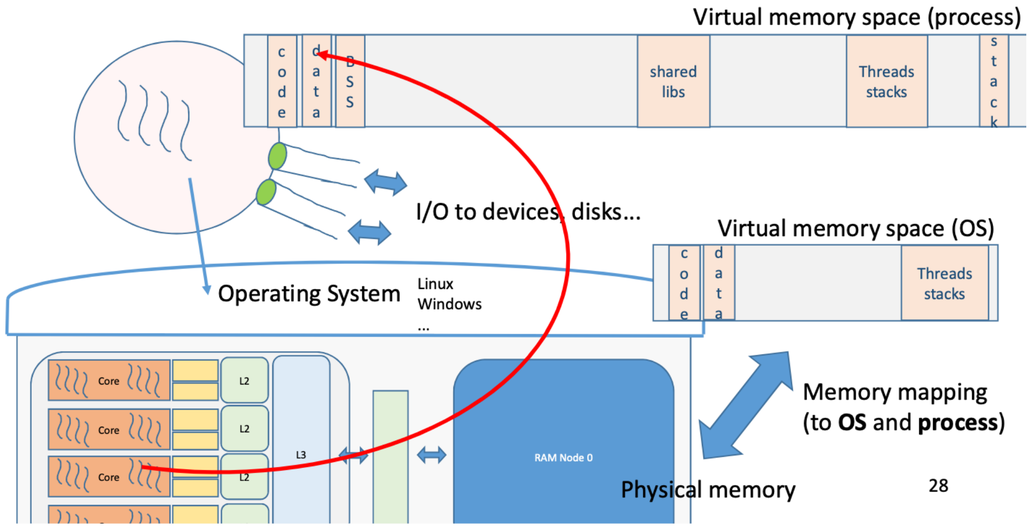
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Instructors

- Jordi Fornés
  - Office hours
    - Monday, 10.00 - 13.00 h. [online]
    - Tuesday, 15.00 - 18.00 h. [online]
  - Room: C6-116, Campus Nord, UPC.
  - E-mail: jfornes@ac.upc.edu

- Course perspective
- Logistics
- Academic integrity
- Overview

Course perspective



- Computer Architecture & Operating Systems
  - Bits, Bytes, Integers and Floats
  - Computer Architecture
  - Operating Systems
    - Process management
    - File systems and Input/Output
    - Memory hierarchy and Virtual memory

Goals

Purpose is to show that by knowing more about the underlying system, one can be more effective as a programmer

- Enable you to
  - Write programs that are more reliable and efficient
  - Incorporate features that require hooks into OS
    - E.g., concurrency, signal handlers

Textbooks

- Randal E. Bryant and David R. O'Hallaron,
  - Computer Systems: A Programmer's Perspective, Third Edition (CS:APP3e), Pearson, 2016
- Abraham Silberschatz, Peter B. Galvin, Greg Gagne,
  - Operating System Concepts, 10th Edition, Wiley, 2019
- David A. Patterson, John L. Hennessy
  - Computer organization and design, Sixth edition, Morgan Kaufmann, 2021

- Course perspective
- Logistics
- Academic integrity
- Overview

Scheduling

	Dilluns	Dimarts	Dimecres	Dijous	Divendres	
9/2/26		Overview			Bits	13/2/26
16/2/26		shell task			Bits	20/2/26
23/2/26		shell task			Bits	27/2/26
2/3/26		data task			CA	6/3/26
9/3/26		data task			CA	13/3/26
16/3/26		data task			Proc	20/3/26
23/3/26		perf task			Proc	27/3/26
30/3/26						3/4/26
6/4/26						10/4/26
13/4/26	midterm1	perf task			Proc	17/4/26
20/4/26		proc task			IO	24/4/26
27/4/26		proc task				1/5/26
4/5/26		proc task			festaFIB	8/5/26
11/5/26		proc task			midterm2	15/5/26
18/5/26		io task			IO	22/5/26
25/5/26		io task			Mem	29/5/26
1/6/26						5/6/26
8/6/26						12/6/26
15/6/26					Final	19/6/26
22/6/26						
29/6/26	Retake					

Midterm	Midterm exam		
Final	Final Exam		
Retake	Retake exam		
	Holiday		
Theory	A5202		
Labs	C6S309		
	midterm1	up to Bits	
	midterm2	up to Proc	
	final	up to Mem	

Theory

- Overview
- Bits
- Computer Architecture
- Process Management
- Input / Output
- Memory Management

Laboratory

- shell scripting
- data puzzles
- performance evaluation
- processing
- input / output

Assessment

- There will be two miderm exams (marks M1, M2), which will last two hours, and a final exam (mark E), which will also last two hours long. All of them administered in person on .
- The exams are open book, closed to the Internet. You are not permitted to use a calculator or any other electronic aid.
- The cross-curricular competence of independent learning is assessed on the basis of the progress reports submitted during the course and accounts for 10% of the final mark. Mark P.
- The final grade (FG) will be calculated as follows:
  - FG = 0.1*P + 0.9*max(E, 0.25*M1 + 0.25*M2 + 0.50*E
- Those who have attended the final exam and have an FG lower than 5 are entitled to a retake exam (reava). This will take place in a laboratory classroom, will last two hours and will consist of theoretical and practical questions, under the same conditions as the rest of the exams. The reava grade will be the final grade of the course, replacing FG.

- Course perspective
- Logistics
- Academic integrity
- Overview

Academic integrity

About cheating

- What is cheating?
  - Sharing code: by copying, retyping, looking at, or supplying a file.
  - Describing: verbal description of code from one person to another.
  - Coaching: helping your friend to write a lab, line by line.
  - Searching the Web for solutions.
  - Copying code from a previous course or online solution.

Academic integrity

About cheating

- What is NOT cheating?
  - Asking the course instructor for help, showing him your code.
  - Explaining how to use systems or tools.
  - Helping others with high-level design issues.
  - Googling a man page.
  - Asking a friend for help with python (but not with your code).
  - Using code examples from book (with attribution).

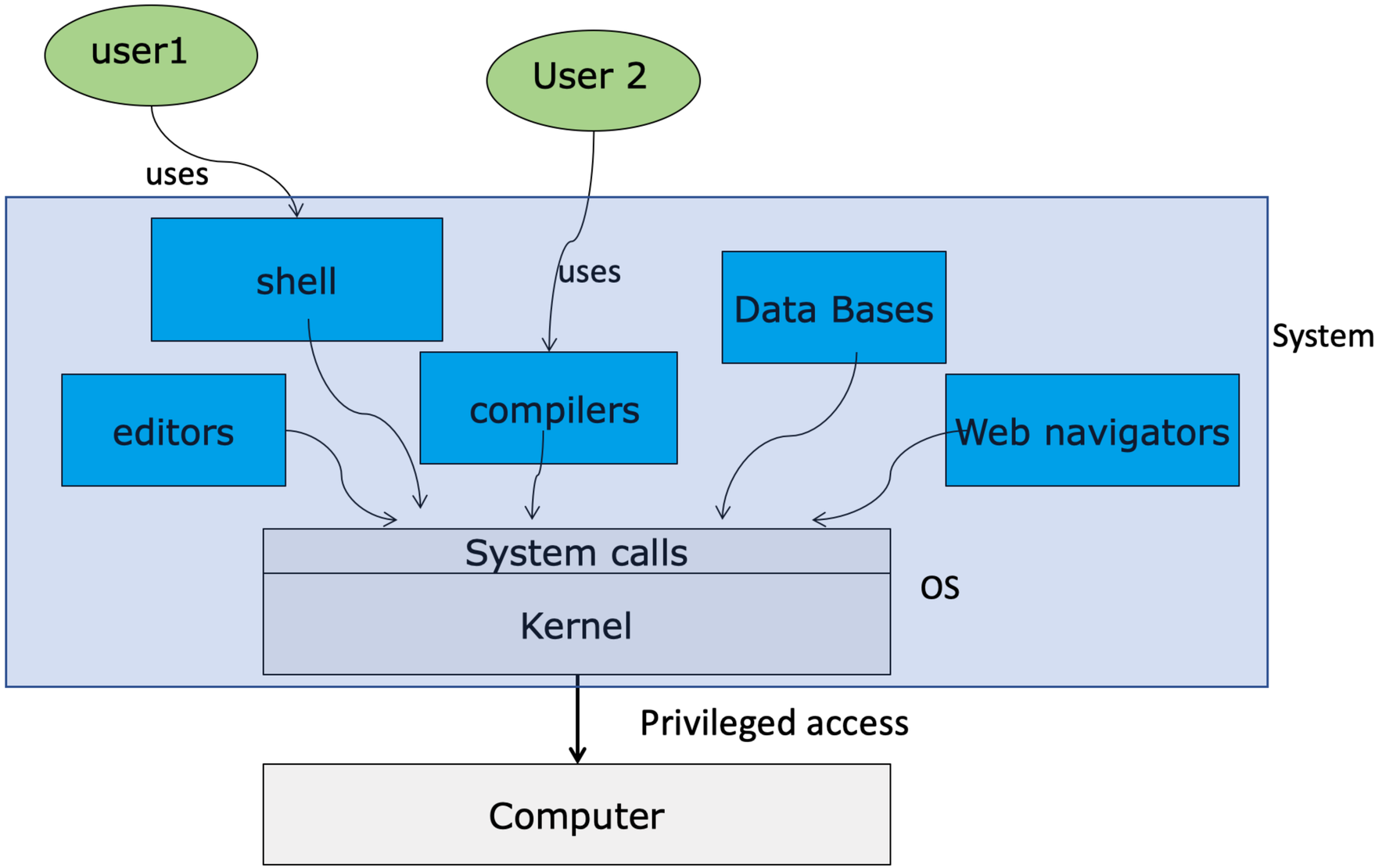
Academic integrity

Cheating: Consequences

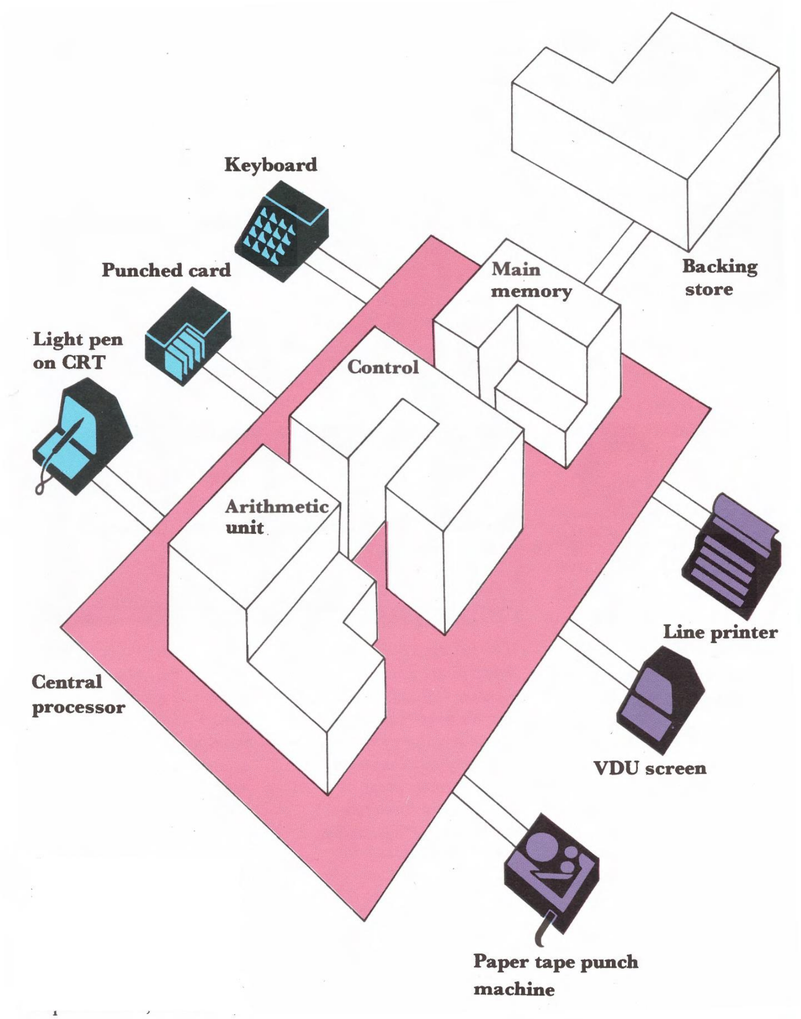
- Best case: -100\% for assignment
- Worst case: Removal from course with failing grade
- Loss of respect by you, the instructors and your colleagues

- Course perspective
- Logistics
- Academic integrity
- Overview

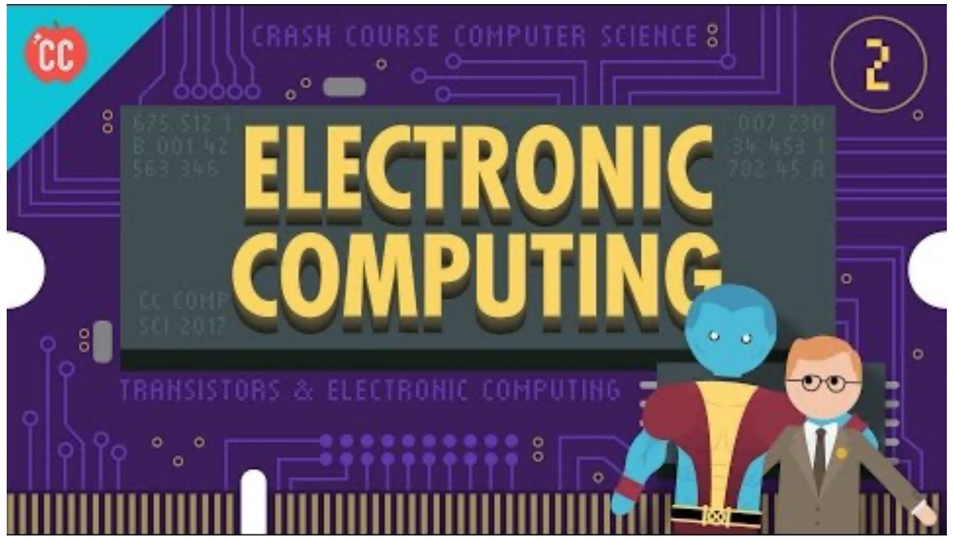
Computer system



Computer Architecture



Source: Patricia Fara,"COMPUTERS", Phelham Books, 1982.



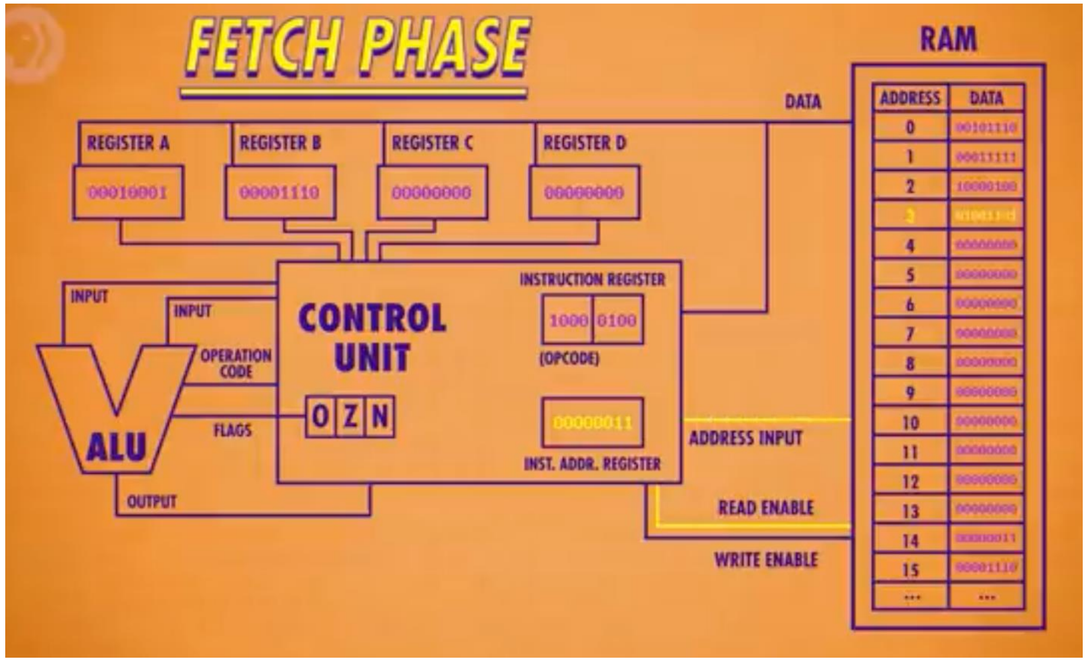
https://youtu.be/LN0ucKNX0hc

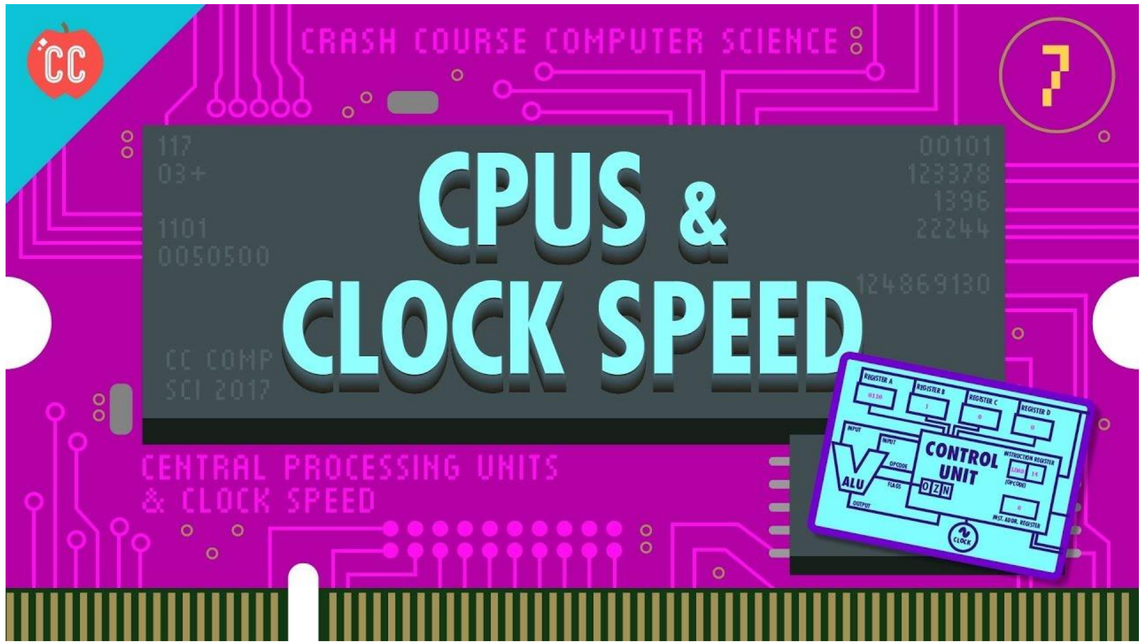
Computer Architecture

- Central Processor Unit
  - Also known as *Processor*
  - It's formed by the Control Unit, an ALU (Arithmetic-Logic Unit) and the Registers (and the clock)
- Memory
  - Random Access Memory or primary storage
  - Where instructions and data are storaged. A sequence of instructions is called a *program*
- Input/Output
  - Devices are the gates to the outside world
- This three components are connected by three busses: control, data and address bus.
  - A bus is a set of wires that transfer bits. Bits are organized in groups of 8, called byte

Computer Architecture

CPU





https://youtu.be/FZGugFqdr60

Computer Architecture

Instruction cycle

- The Control Unit (one per tick) …
  - Uses the Program Counter (PC) to `fetch` an an instruction from primary storage and copy it into the Instruction Register (IR)
  - Reads the pattern of bits in the IR and `decodes` the instruction
  - Based on the decoding, tells the ALU to `execute` the instruction, which means that the ALU manipulates the registers accordingly.

Operating Systems

What's an Operating System?

- The startup program that manages everything.
- The O.S. is a software that manages hardware resources.
- It acts like intermediary between applications and hardware.
- Provides a runtime environment between convenient and efficient to run programs.
- Kenel internals. It defines data structures to manage HW and algorithms to decide how to use it.
- Kernel API.: It offers a set of functions to ask for system services.

Operating Systems

Steps

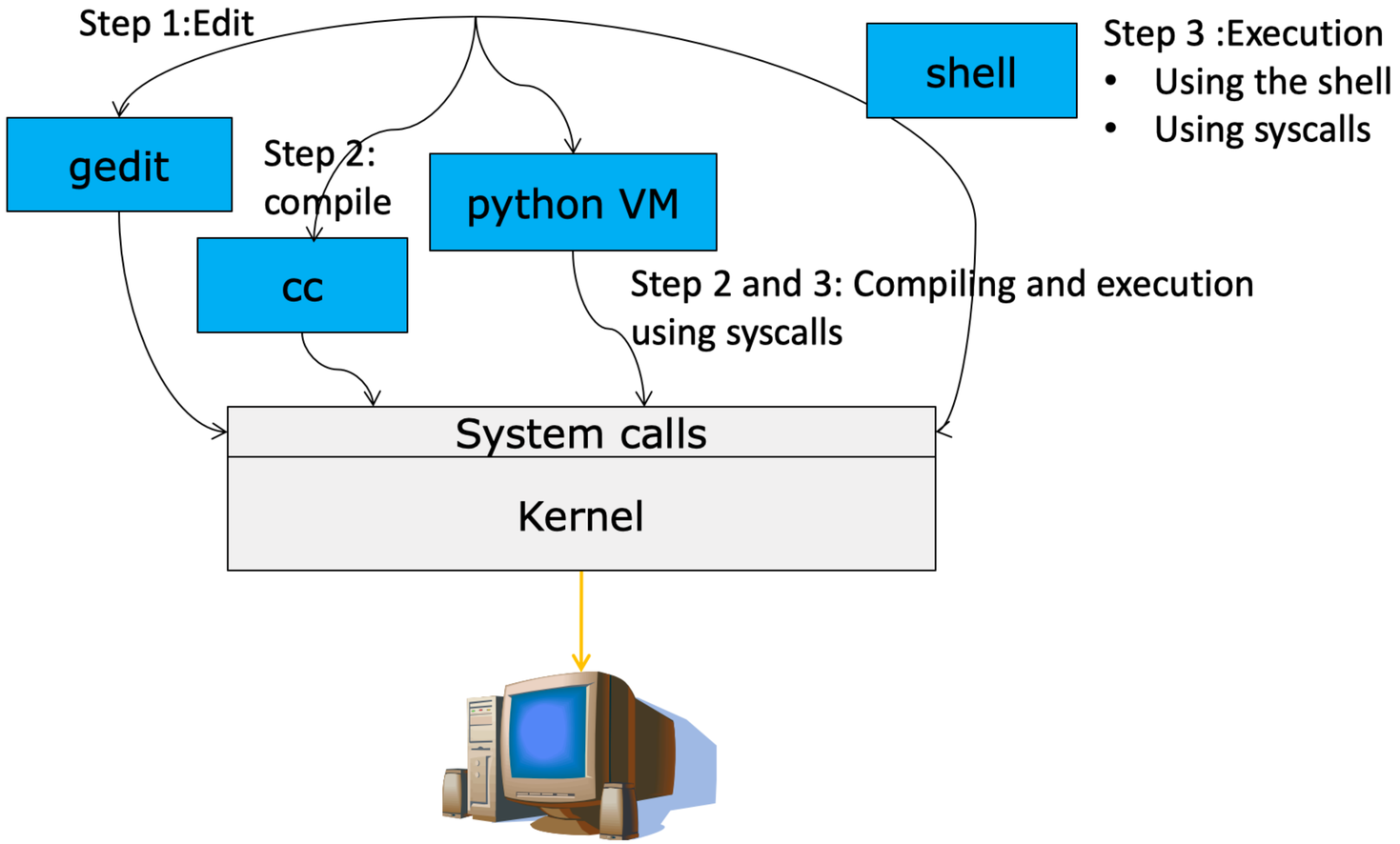
- Boot
  - Executed when the system is switched on, the kernel code is loaded in memory.
  - Interrupts and basic HW configurations are initialized.
  - It starts the system access mechanism: login, shell, etc.
- Usage
  - Develop new applications
  - Execute applications
- Shutdown
  - Executed when system is switched off.
  - Saves persistent information, stop devices, etc.

Operating Systems

What to expect from the OS?

- It offers a *user friendly* environment
  - It abstracts the user from the different kind of “systems”
- It offers a *safe/robust* execution environment
  - Safe from the point of view of accessing HW correctly and from the point of view of user’s interaction
- It offers an *efficient* execution management
  - Fine grain knowledge of HW
  - Many users/programs sharing resources provides a better resource utilization

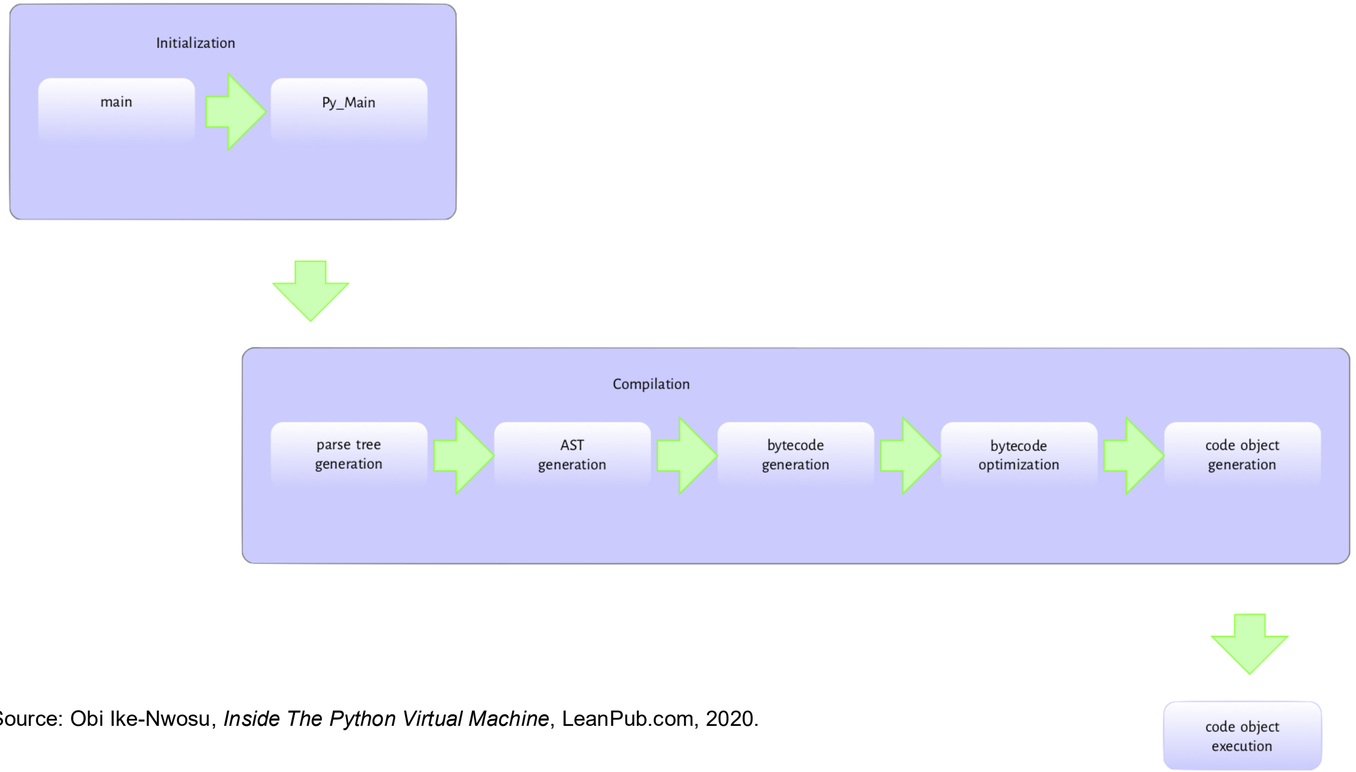
Application development environment



Flow during the execution of CPython source code

- A method of executing a Python script is to pass it as an argument to the Python interpreter as such

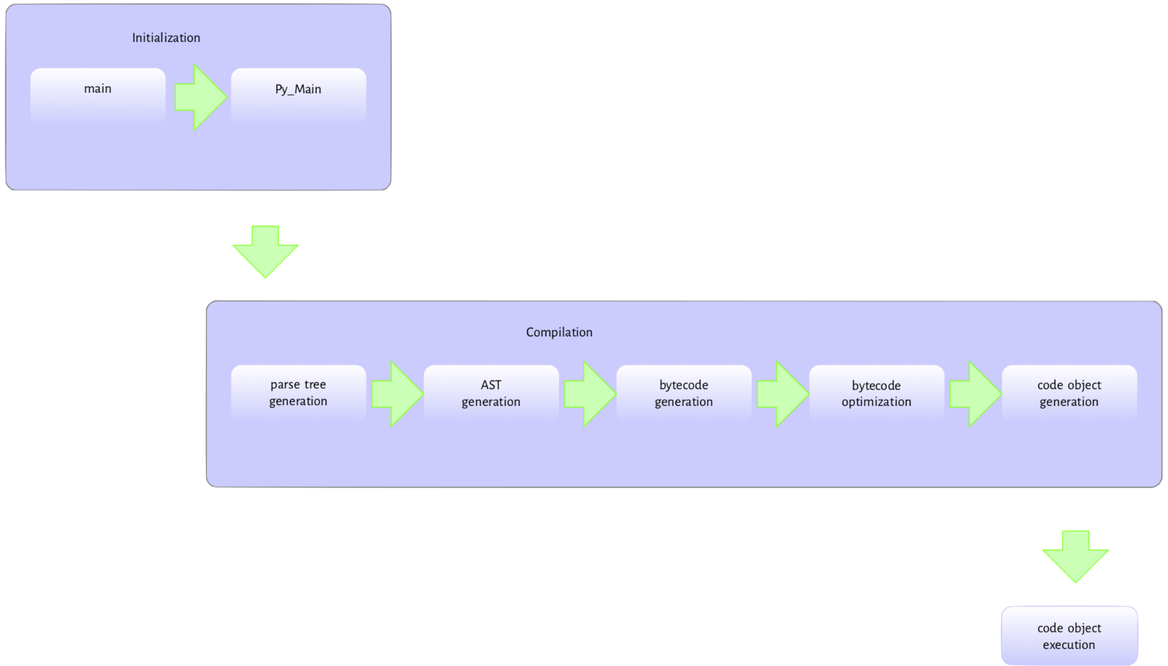
```
$ python hellow_world.py
```



Source: Obi Ike-Nwosu, *Inside The Python Virtual Machine*, LeanPub.com, 2020.

Flow during the execution of CPython source code

- The Python executable is a C program like any other C program such as the Linux kernel or a simple hello world program in C
- The executable's entry point is the `main` method in the `Programs/python.c`



Source: Obi Ike-Nwosu, *Inside The Python Virtual Machine*, LeanPub.com, 2020.

Tracing system calls `hello_world.c`

Terminal — ssh -X nuca — 161×68

```
jfornes@nuca:~/TIACOS% strace ./hello_world
execve("./hello_world", ["./hello_world"], [/* 43 vars */]) = 0
brk(NULL)                               = 0x173a000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f077ead0000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=310140, ...}) = 0
mmap(NULL, 310140, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f077ea84000
close(3)                                = 0
open("/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360\10\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2076824, ...}) = 0
mmap(NULL, 3967456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f077e4e3000
mprotect(0x7f077e6a3000, 2093056, PROT_NONE) = 0
mmap(0x7f077e8a2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bf000) = 0x7f077e8a2000
mmap(0x7f077e8a8000, 14816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f077e8a8000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f077ea83000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f077ea82000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f077ea81000
arch_prctl(ARCH_SET_FS, 0x7f077ea82700) = 0
mprotect(0x7f077e8a2000, 16384, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ)     = 0
mprotect(0x7f077ead1000, 4096, PROT_READ) = 0
munmap(0x7f077ea84000, 310140)          = 0
write(1, "Hello world\n", 12Hello world
)           = 12
exit_group(12)                          = ?
+++ exited with 12 +++
```

Tracing system calls `hello_world.py`

Head

```
jfornes@nuca:~/TIACOS% strace python3 hello_world.py
execve("/usr/bin/python3", ["python3", "hello_world.py"], [/* 43 vars */]) = 0
brk(NULL)                               = 0x16bd000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1595b95000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=310140, ...}) = 0
mmap(NULL, 310140, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1595b49000
close(3)                                = 0
open("/usr/lib64/libpython3.7m.so.1.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240{\6\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=3335456, ...}) = 0
mmap(NULL, 5565264, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1595422000
mprotect(0x7f15956e2000, 2093056, PROT_NONE) = 0
mmap(0x7f15958e1000, 454656, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2bf000) = 0x7f15958e1000
mmap(0x7f1595950000, 133968, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f1595950000
close(3)                                = 0
open("/lib64/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0``\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=135496, ...}) = 0
mmap(NULL, 2212904, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1595205000
mprotect(0x7f159521d000, 2093056, PROT_NONE) = 0
mmap(0x7f159541c000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17000) = 0x7f159541c000
mmap(0x7f159541e000, 13352, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f159541e000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1595b48000
open("/lib64/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\r\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=18808, ...}) = 0
mmap(NULL, 2109680, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1595001000
mprotect(0x7f1595004000, 2093056, PROT_NONE) = 0
mmap(0x7f1595203000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f1595203000
close(3)                                = 0
open("/lib64/libutil.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20\16\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=14080, ...}) = 0
mmap(NULL, 2105616, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1594dfe000
mprotect(0x7f1594e00000, 2093056, PROT_NONE) = 0
mmap(0x7f1594fff000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7f1594fff000
close(3)                                = 0
open("/lib64/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260U\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1147632, ...}) = 0
mmap(NULL, 3178760, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1594af5000
mprotect(0x7f1594bfd000, 2093056, PROT_NONE) = 0
mmap(0x7f1594dfc000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x107000) = 0x7f1594dfc000
close(3)                                = 0
open("/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
```

Tracing system calls `hello_world.py`

Tail

```
512 open("hello_world.py", O_RDONLY)        = 3
513 fcntl(3, F_GETFD)                       = 0
514 fcntl(3, F_SETFD, FD_CLOEXEC)           = 0
515 fstat(3, {st_mode=S_IFREG|0644, st_size=25, ...}) = 0
516 ioctl(3, TCGETS, 0x7ffd00ff1cd0)       = -1 ENOTTY (Inappropriate ioctl for device)
517 lseek(3, 0, SEEK_CUR)                   = 0
518 fstat(3, {st_mode=S_IFREG|0644, st_size=25, ...}) = 0
519 read(3, "print ('Hello world\\n')\n\n", 4096) = 25
520 lseek(3, 0, SEEK_SET)                   = 0
521 read(3, "print ('Hello world\\n')\n\n", 4096) = 25
522 read(3, "", 4096)                       = 0
523 close(3)                               = 0
524 write(1, "Hello world\n", 12Hello world
525 )           = 12
526 write(1, "\n", 1
527 )                       = 1
528 rt_sigaction(SIGINT, {SIG_DFL, [], SA_RESTORER, 0x7f15952163b0}, {0x7f1595622190, [], SA_RESTORER, 0x7f15952163b0}, 8) = 0
529 brk(0x177f000)                          = 0x177f000
530 sigaltstack(NULL, {ss_sp=0x1703180, ss_flags=0, ss_size=8192}) = 0
531 sigaltstack({ss_sp=NULL, ss_flags=SS_DISABLE, ss_size=0}, NULL) = 0
532 brk(0x176e000)                          = 0x176e000
533 exit_group(0)                           = ?
534 +++ exited with 0 +++
535
```

Accessing the kernel code

Execution modes: privileged/not privileged

- The CPU must be able to tell the difference between instructions from normal user code and instructions from the kernel code. This is how it can guarantee hardware security (from non-expert or malicious users) and user resources from other users.
- This support must be provided by the HW, otherwise security can not be guarantee
- We need, at least, two levels of privileges
- We will refer to them as:
  - User vs. kernel modes
  - User vs. system modes
  - Privileged vs. not-privileged modes

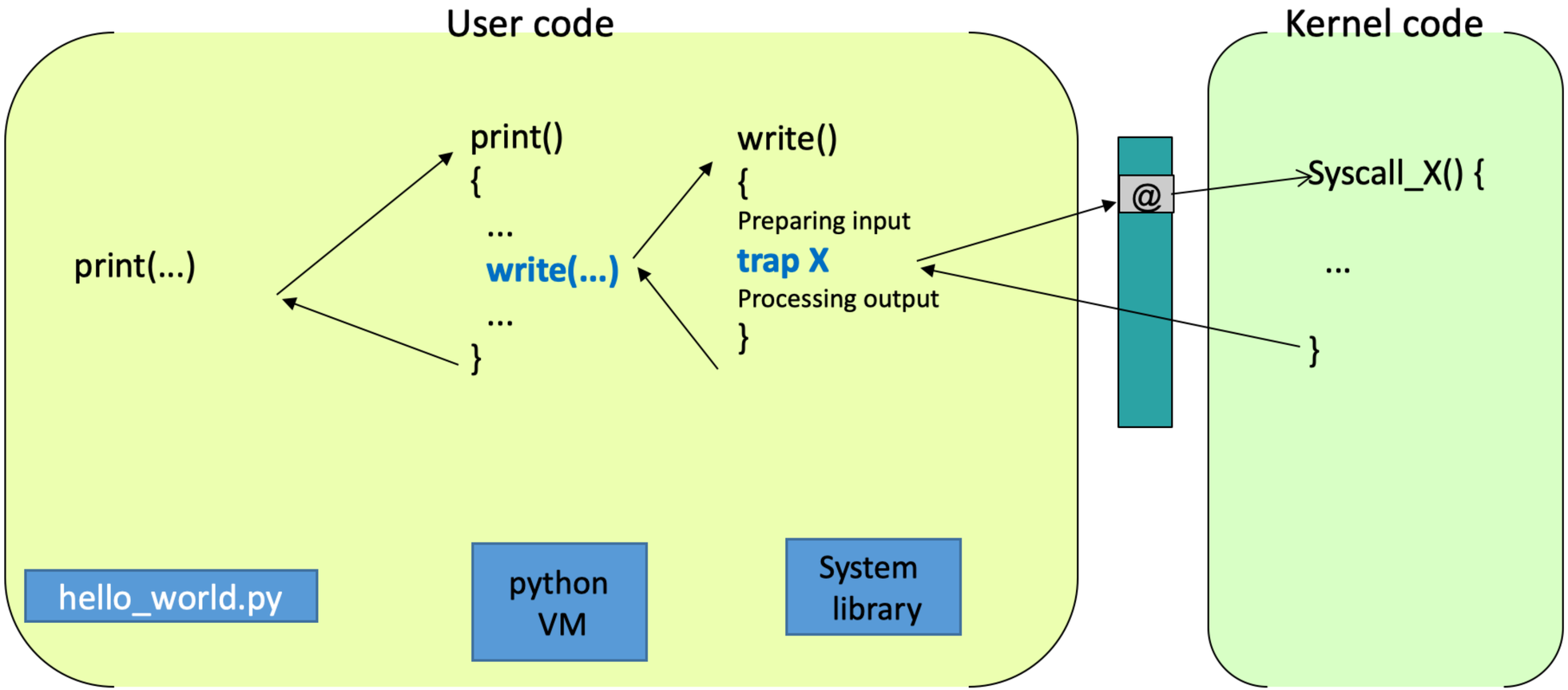
When the kernel code is executed?

Just 3 cases

- When an exception occurs: exceptions are generated by the CPU when some problem occurs during the execution of one instruction
- When the user code executes a system call request (executing a special instruction)
- When an interrupt occurs: interrupts are generated by hardware devices
  - These events haven’t a fixed frequency, and it could (potentially) pass a lot of time between them. To avoid this situation, the kernel configure the CPU clock to generate an interrupt periodically (every 10ms for instance).
    - With this extra event, the kernel can check the system status every 10ms.

The whole picture

From user code to kernel code



https://manpages.ubuntu.com/manpages/hirsute/en/man2/syscalls.2.html}