



Multithreading vs Multiprocessing vs Async

Fisa
(Juan Pedro Fisanotti / @fisadev)

Cuál uso? Cuándo?
Por qué?

Concurrencia y Paralelismo

Concurrencia

“Hacer muchas cosas juntas”

Paralelismo

“Hacer muchas cosas al mismo tiempo”

Concurrencia

“Fisa programa y mira twitter”

“Sofi programa y Fisa mira twitter”

Paralelismo

“Sofi programa y Fisa mira twitter”

Paralelismo: una forma de lograr
Concurrencia

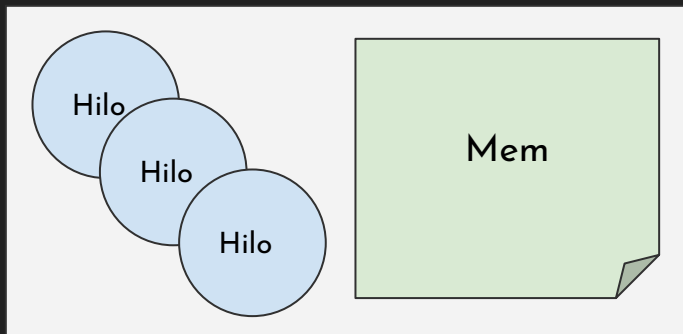
Pero **no** la única

Hablemos de computadoras

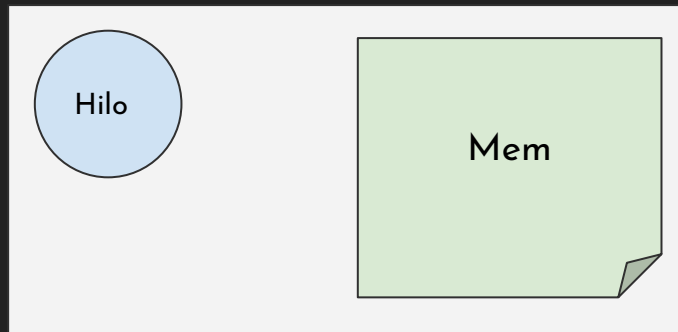
(con simplificaciones)



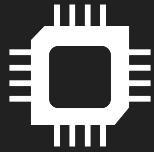
Proceso 1



Proceso 2



1 core




1 proceso/hilo corriendo en
cada instante

Concurrencia (“un poquito
cada uno”) pero **no**

Paralelismo

Paralelismo implica N cores



Context switches

Cambiar de hilo/proceso tiene su costo de CPU

Todo un mundo de nuevos problemas y soluciones (ej: race conditions, locks, etc)



Esperas de I/O

Esperas de operaciones de
disco, red, etc

Cuánto? depende del programa

Si un hilo/proceso está
esperando, tiene sentido “darle”
tiempo de cpu?

Es más complicado compartir datos en
multiprocessing

Programas sin esperas de I/O

Paralelismo ayuda a reducir tiempos
Concurrencia sin Paralelismo **no** los reduce!

1 core? Nada nos va a ayudar :(



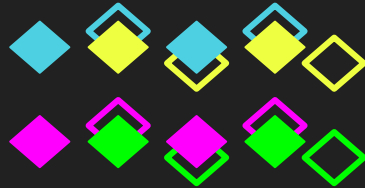
Programas con esperas de I/O

Concurrencia sin Paralelismo **igual ayuda** a acelerar



Lo más óptimo es combinar ambas cosas!

Paralelismo con varios cores, concurrencia en cada core



Hablemos de Python





Multi threading

Simple! Módulo **threading** permite lanzar hilos y controlarlos.

Paralelismo?

No en CPython!* (el oficial)

Sí en otros pythons (N cores)

(*) por la famosa GIL



Multi threading

Los hilos comparten memoria :)

Ej: creo una lista en un hilo y
agrego items desde otro, sin
hacer nada raro.



Multi threading

En casos de muchas tareas (hilos) o esperas de I/O muy largas, el context switch hace que no sea eficiente.

[ejemplo de código]



Multi processing

Simple! El módulo **multiprocessing** nos permite lanzar procesos y controlarlos.

Paralelismo? Si hay N cores sí!



Multi processing

Cada uno con su memoria

Mucha memoria repetida
(python, libs, etc)

Si quiero compartir datos?
Comunicación entre procesos,
no tan trivial, pero Python
ayuda



Multi processing

En casos de muchas tareas (procesos) o esperas de I/O muy largas, el context switch hace que no sea eficiente.

[ejemplo de código]

Y Async???

Mejor explicado acá:

bit.ly/fisa_async

:)



Async

1 proceso, 1 hilo, N corrutinas

Corrutinas: funciones que se pueden pausar para dejar que otras corran

Un loop va haciendo correr un poco cada una, respetando las pausas



Async

1 proceso → Memoria
compartida, como en
multithreading



Async

1 hilo → Solo Concurrencia,
no Paralelismo

Pero **nosotros** decimos cuándo
se hace el context switch y
hasta cuándo

Ej: “pausá esta corrutina hasta
que termine de escribirse tal
dato al disco”



Async

Más eficiente, sobre todo en mucha carga de tareas o I/O

Más seguro/controlable/
reproducible

Un poco más complicado
A la vez, más natural a la hora de
controlar context switches

No todas las libs lo soportan

[ejemplo de código]

Multithreading
vs
Multiprocessing
vs
Async

Programa sin esperas de I/O?

Multiprocessing y N cores

Ninguna otra cosa va a ayudar



Programa con esperas de I/O?

Depende

Si la **performance** es muy importante:

Nada le gana a Async+Multiprocessing, pagando el
costo de la complejidad



Si el **control** de los context switches o
reproducibilidad es importante:

Async suele hacerlo más natural, una vez que lo
aprendiste a usar



Si no, Multithreading si hay mucha/compleja **data compartida**, o Multiprocessing en caso contrario

