Concurrencia y Arquitecturas Multiprocesador

Módulo 4 : Técnicas computacionales avanzadas para modelar fenómenos sociales
Concentración en Economía Aplicada y Ciencia de Datos
ITESM

1 de mayo de 2024

¿Concurrencia?¿Paralelismo?

¿Concurrencia?¿Paralelismo?

Cómputo Concurrente y Distribuido : Las matemáticas de trabajar en equipo

¿Para qué aprovechar las arquitecturas Multicore?

La **Ley de Moore** alcanzó el límite.

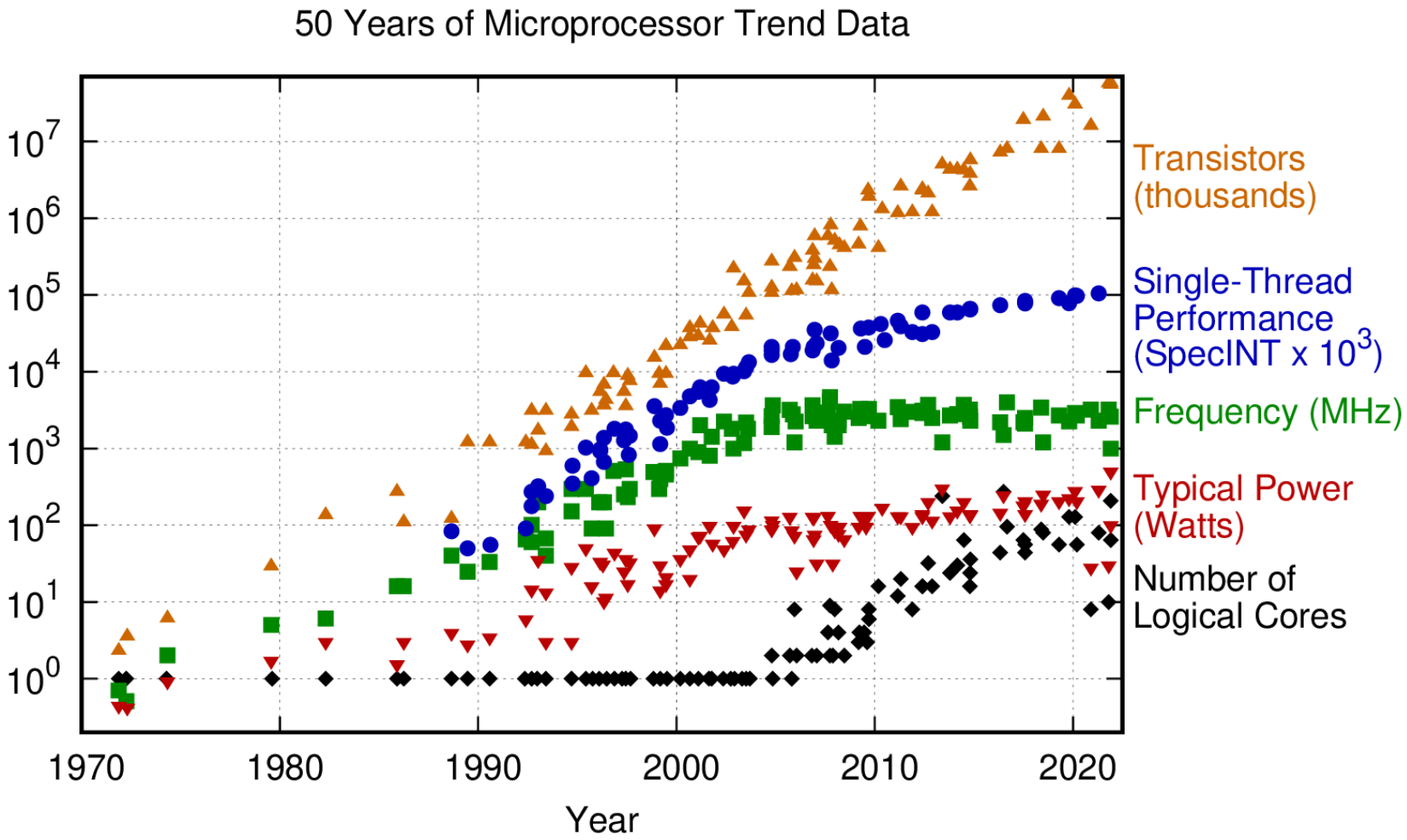


Figura: Single thread performance, CPU clock frequency (MHz), CPU power consumption (watts), and the number of CPU cores from 1970 to 2018. (Horowitz et al. and Rupp, `https://github.com/karlrupp/microprocessor-trend-data`)

Beneficios potenciales

- **Tiempo de ejecución más rápido con más cores.**
- **Trabajar problemas de tamaño grande con más cores.**
- **Eficiencia de energía al hacer más con menos.**

Arquitectura de cómputo

Una arqutectura de cómputo consiste esencialmente de uno o más CPU (procesador) y un bloque de memoria donde se almacenan dátos e instrucciones (código).

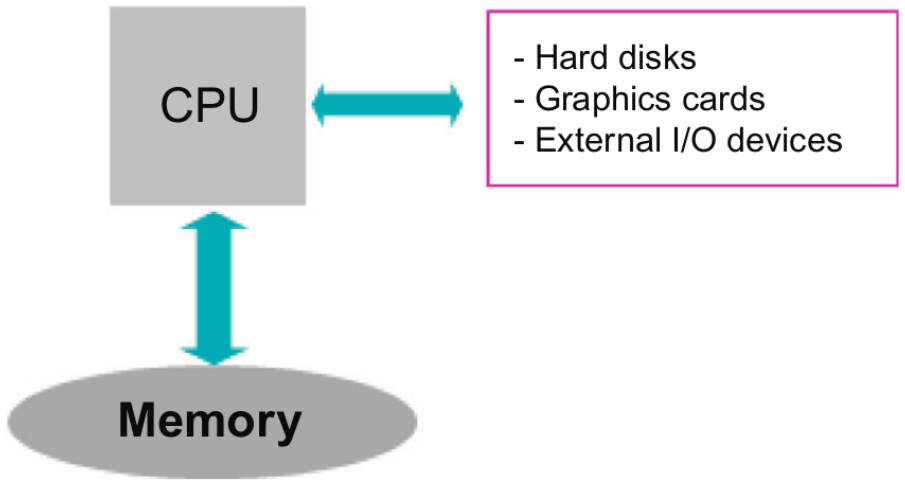


FIGURE 1.1

Schematic architecture of a simple computing platform.

Figura: Arquitectura con **un** CPU. Tomado de Alessandrini (2015)

Arquitectura de cómputo

- El desempeño en el procesamiento de datos depende básicamente del desempeño del CPU, esto es, la velocidad a la que el CPU puede ejecutar las instrucciones básicas.
- Pero también depende criticamente de la tasa a la que los datos pueden ser movidos desde y hacia la memoria : un procesador rápido es inútil si no tiene datos con los que trabajar.

Principio de arquitectónico : los procesadores y la memoria (principal) están muy separados.

Uno de los retos es tratar de reducir el tiempo (latencia) que lleva acceder a la memoria.

Arquitectura de cómputo : latencia y ancho de banda (Matloff, 2015)

- La **latencia** (*latency*) mide la velocidad del canal de comunicación entre los CPUs y la memoria[1]. ¿Cuál es el tiempo de transferencia de un bit desde y hacia la memoria?
- El **ancho de banda** (*bandwidth*) es la cantidad de bits por unidad de tiempo que podemos transferir por el canal de comunicación.

[1] En plataformas de memoria compartida. En un cluster, la latencia mide la velocidad de comunicación en la red de nodos de cómputo

Arquitectura de cómputo : latencia y ancho de banda (Matloff, 2015)

- La noción de latencia describe el tiempo que le toma a un carro ir de un extremo a otro de puente[2].
- El ancho de banda sería el número de carros saliendo de un extremo por unidad de tiempo.

[2] Asume que todos los carros van a la misma velocidad.

Arquitectura de cómputo : latencia y ancho de banda (Matloff, 2015)

- Podemos reducir la latencia al incrementar el límite de velocidad permitido en el puente.
- Podemos incrementar el ancho de banda an agregar más carriles.
- El tiempo en segundos de enviar un mensage de n-bytes, con una latencia de l segundos y un ancho de banda de b $\frac{text bytes}{\text{segundos}}$ es

$$l + \frac{n}{b}$$

Se asume que no hay otros mensajes contendiendo por el canal de comunicación.

Arquitecturas Multiprocesador de Memoria Compartida

En la arquitectura llamada *symmetric multiprocessor* una interconexión de red conecta varios procesadores a un bloque de memoria compartida **compartido**.

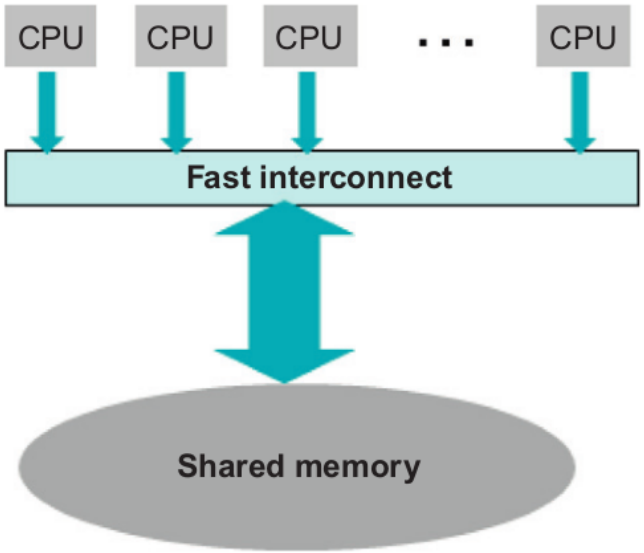


FIGURE 1.2

Symmetric multiprocessor shared memory platform.

Figura: Tomado de Alessandrini (2015)

Arquitecturas Multiprocesador de Memoria Compartida

- Un *multiprocesador* consiste de multiples procesadores, cada uno de estos ejecutando un programa secuencial.
- *Symmetric* significa que todos los CPU tienen un estatus equivalente respecto a los accesos de memoria.
- La memoria es compartida, todos los CPU pueden acceder a todo el espacio de direcciones de memoria.

Arquitecturas Multiprocesador de Memoria Compartida

- Un *multiprocesador* consiste de multiples procesadores, cada uno de estos ejecutando un programa secuencial.
- *Symmetric* significa que todos los CPU tienen un estatus equivalente respecto a los accesos de memoria.
- La memoria es compartida, todos los CPU pueden acceder a todo el espacio de direcciones de memoria.

Problema : Coherencia en los accesos a memoria compartida evitan que las plataformas SMP escalen a grandes cantidades de CPU.

Arquitecturas Multiprocesador de Memoria Compartida

- Un *multiprocesador* consiste de multiples procesadores, cada uno de estos ejecutando un programa secuencial.
- *Symmetric* significa que todos los CPU tienen un estatus equivalente respecto a los accesos de memoria.
- La memoria es compartida, todos los CPU pueden acceder a todo el espacio de direcciones de memoria.

Problema : Coherencia en los accesos a memoria compartida evitan que las plataformas SMP escalen a grandes cantidades de CPU.

- Los sistemas de memoria compartida actuales no exceden los 60 CPU.

Arquitecturas Multiprocesador de Memoria Compartida: Beneficios

- Pueden realizar eficiente **multitasking**[3] asignando diferentes aplicaciones (**procesos**, **threads**) en diferentes CPU.
- Multitasking es la capacidad del Sistema Operativo de correr varias aplicaciones al mismo tiempo.
- En un entorno multitasking , la totalidad de los recursos de la computadora (memoria, archivos, tiempo de CPU) se asignan a diferentes aplicaciones y se administran de tal manera que cada una de ellas utiliza los recursos de acuerdo con políticas de prioridad particulares.
- Por otro lado, una aplicación **multithread** puede beneficiarse de la disponibilidad de una cantidad razonable de CPU para mejorar el rendimiento del proceso.

[3] *Task* es una unidad fundamental de computación secuencial.

Preemptive[4] multitasking y cooperative multitasking (Fowler, 2022)

Preemptive multitasking

- El sistema operativo decide cómo cambiar entre qué aplicación se está ejecutando actualmente a través de un proceso llamado división de tiempo (*time slicing*).
- El evento en el que el sistema operativo deja a la aplicación que está atendiendo para ejecutar otra aplicación se le conoce como **cambio de contexto**.
- Para realizar un cambio de contexto, el sistema operativo debe guardar el estado del CPU y el puntero de instrucción (*instruction pointer*) para la aplicación que se está ejecutando actualmente, averiguar a qué aplicación cambiar y recargar el estado de la CPU para la aplicación a la que se está cambiando (Williams, 2019).

[4] Se puede traducir como *Cambio de contexto*

Preemptive multitasking y cooperative multitasking (Fowler, 2022)

Cooperative multitasking

- El sistema operativo decide **NO** cómo cambiar entre qué aplicación: la programadora explícitamente define los puntos en la aplicación donde se cede la prioridad de ejecución a otras aplicaciones.
- La aplicación *coopera* cediendo la prioridad de ejecución a otras aplicaciones.

Procesos, threads, multithreading y multiprocessing (Fowler, 2022)

Procesos

- Un proceso es la ejecución de una aplicación que tiene un espacio de memoria reservado al que otras aplicaciones no pueden acceder.
- Se pueden ejecutar múltiples procesos en una sola máquina.
- Si estamos en una máquina que tiene un CPU con varios núcleos, podemos ejecutar varios procesos **simultáneamente**.
- Si contamos con un CPU con un solo núcleo, aún podemos tener varias aplicaciones ejecutándose **en el mismo intervalo de tiempo**, a través de la división del tiempo (*time slicing*).

Procesos, threads, multithreading y multiprocessing (Fowler, 2022)

Secuencial

- El término secuencial puede ser usado de dos maneras. La primera significa que cierto conjunto de tareas deben ejecutarse en un orden estricto. La segunda se refiere a una limitación que impone el sistema en el orden de ejecución de las tareas.

Concurrencia

- Decimos que dos tareas están pasando concurrentemente si están pasando **en el mismo intervalo de tiempo**.

Paralelismo

- Decimos que dos tareas están corriendo de forma paralela si no solo dichas tareas están corriendo concurrentemente, sino que también se están pasando de forma **simultánea**.

Procesos, threads, multithreading y multiprocessing

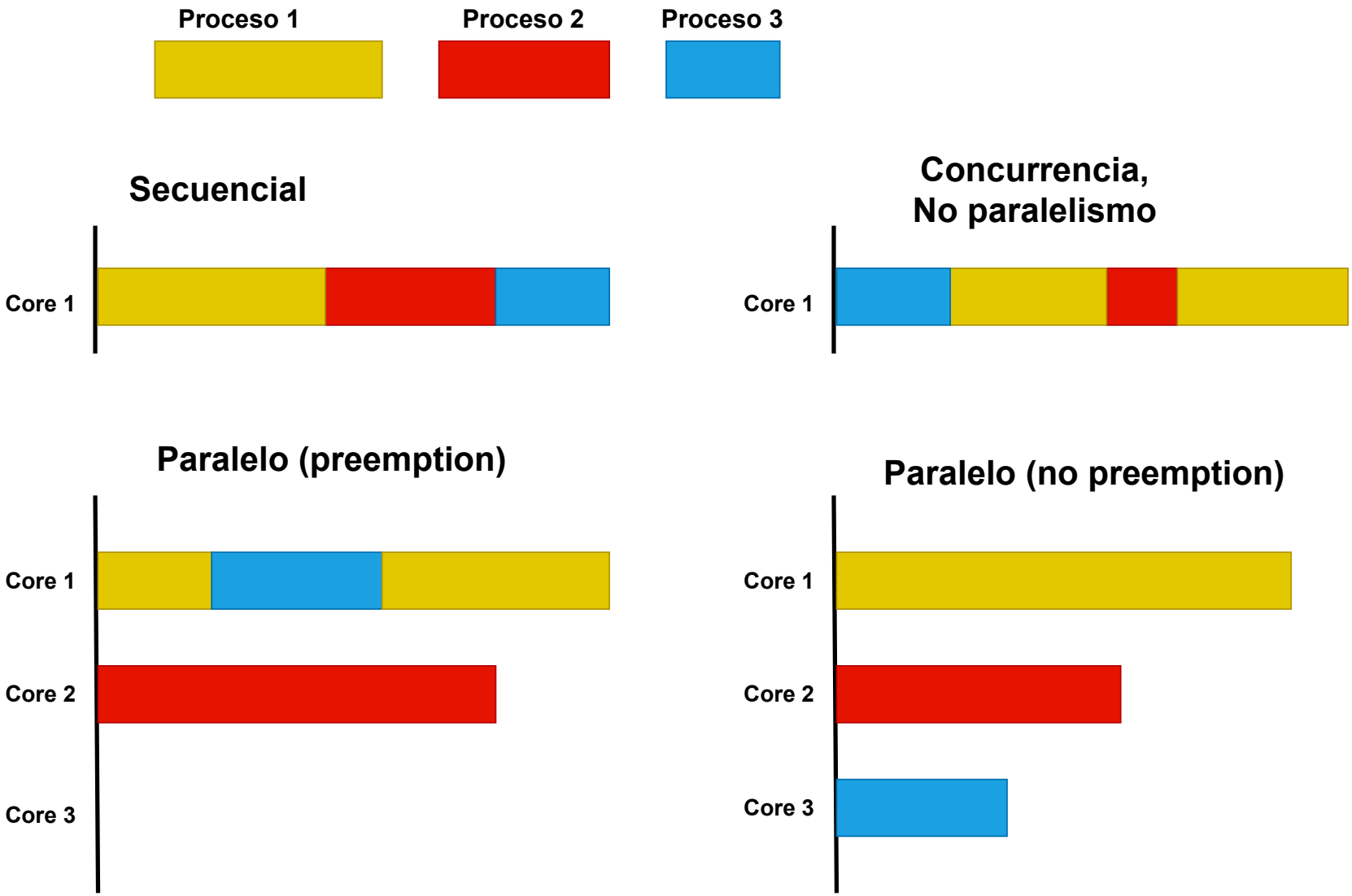


Figura: Adaptado de Rodrigues Antao (2023)

Threads (hilos) (Fowler, 2022)

- Un Thread (hilo) se le denomina un *proceso ligero* que ejecuta un programa secuencial.
- Son la construcción más pequeña que puede administrar un sistema operativo. Es la unidad básica de despacho.
- Son más simples que los procesos.
- No tienen memoria propia como la tiene un proceso: comparten la memoria del proceso que los creó.
- Los threads están asociados con el proceso que los creó.
- Un proceso siempre estará asociado a un thread, usualmente conocido como **main thread**.

Threads (hilos) (Fowler, 2022)

- Como los procesos, los threads pueden ser asignados a distintos cores del CPU, y el sistema operativo puede también hacer cambio de contexto via time slicing.
- El cambio de contexto entre threads es más eficiente que el cambio de contexto entre procesos dado que los recursos globales son compartidos por todos los threads en la aplicación, de manera que no necesitan ser guardados cuando se produce el cambio entre threads.

Threads (hilos) (Fowler, 2022)

Un proceso puede crear otros threads, los cuales se conocen como **worker threads**.
Cuando nuestra aplicación tiene más de un thread, se dice que es *multithreading*.

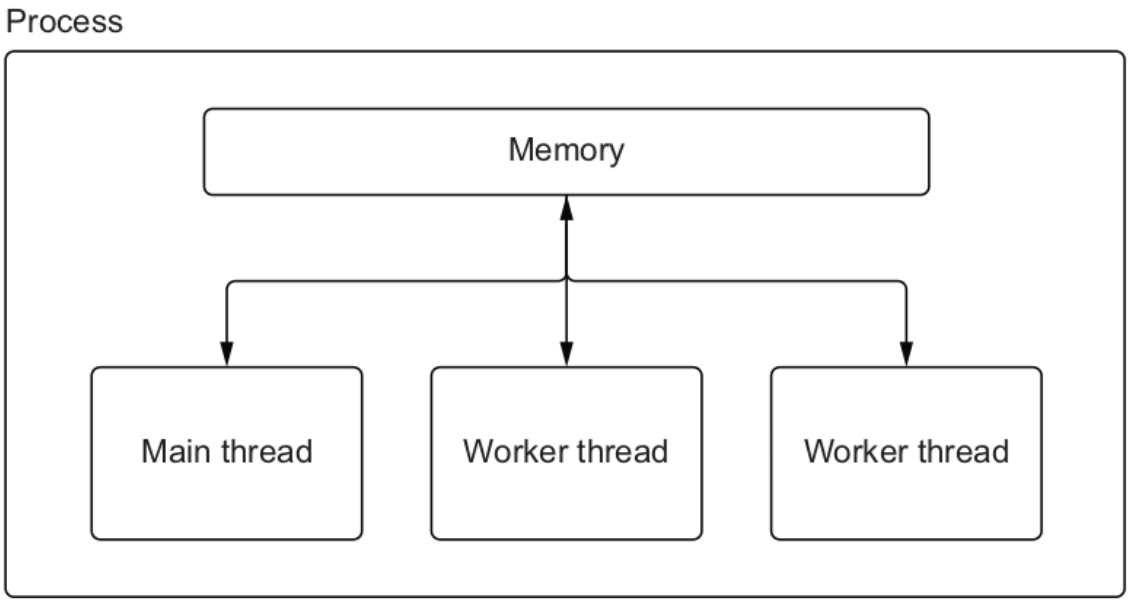


Figure 1.4 A multithreaded program with two worker threads and one main thread, each sharing the process's memory

Figura: Tomado de Fowler (2022)

¿¸Por qué usar Concurrencia? (Williams, 2019)

- Las aplicaciones multithreaded con una forma común de explotar la concurrencia en lenguajes de programación.
- Hay principalmente dos razones para usar concurrencia en una aplicación:
  - Separación de responsabilidades (*separation of concerns*).
  - Desempeño (*performance*). Hay dos formas de usar la concurrencia para mejorar el desempeño:
    - ★ *Task parallelism*: Divide una tarea en partes y corre cada parte en paralelo. **Dificultad** : dependencia entre las partes.
    - ★ *Data parallelism*: Cada thread realiza la misma operación en diferentes partes de los datos. **Dificultad** : hay datos que son **compartidos**, se necesitan mecanismos de **sincronización** (coordinación) entre los threads.

Leyes fundamentales del cómputo paralelo: Ley de Amdahl

El *SpeedUp* es métrica para evaluar el beneficio potencial de un programa paralelo al medir el tiempo que un solo procesador tarda en realiza una tarea(de tamaño fijo) contra en tiempo que toma completarse la misma tarea con N procesadores paralelos.

Siendo P la fracción paralela de la tarea, S la fracción serial ($P + S = 1$), el *SpeedUp* es:

$$SpeedUp(N) = \frac{1}{S + \frac{P}{N}} \tag{1}$$

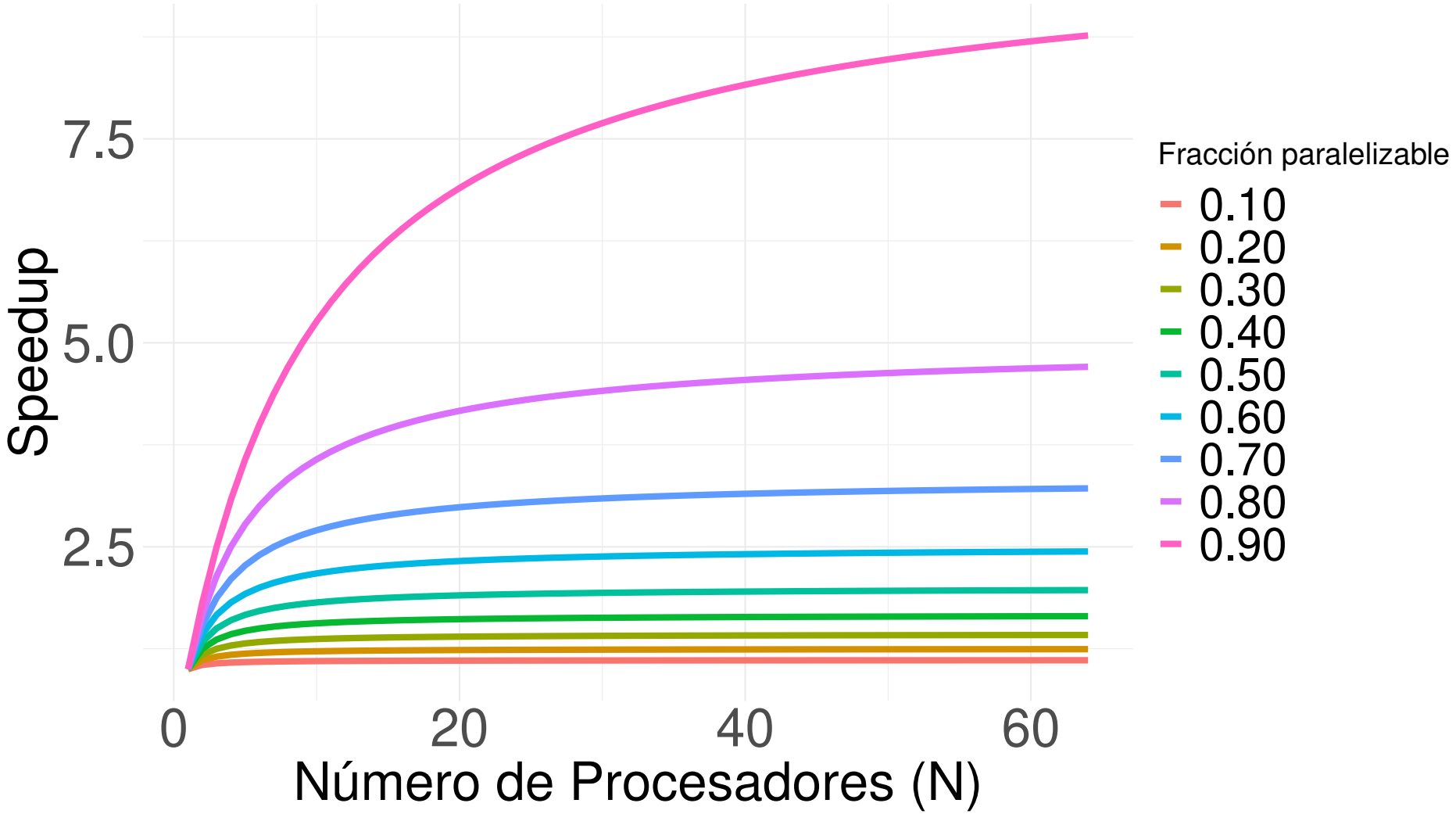
Leyes fundamentales del cómputo paralelo: Ley de Amdahl

En una situación ideal, para un algoritmo completamente paralelizable[5], $P = 1$, de manera que

$$SpeedUp(N) = N \tag{2}$$

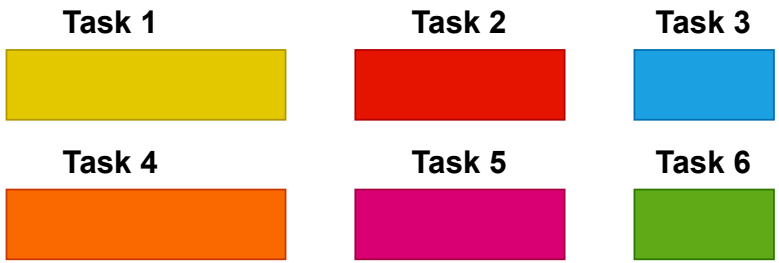
[5] Y sin considerar el tiempo de comunicación entre los procesadores y la memoria

Leyes fundamentales del cómputo paralelo: Ley de Amdahl



¿Todo está perdido?

¿Todo está perdido?



$$\frac{1}{S + \frac{P}{N}}$$

¡¡ CONCURRENCIA !!

Multiprocesamiento (Multiprocessing)

- Multithreading no es la única forma de explotar la concurrencia: podemos también crear **múltiples procesos** que trabajen de forma concurrente.
- Se le conoce como **Multiprocesamiento**.
- En este enfoque, una proceso **padre** (*parent*) crea uno o más procesos **hijos** (*child*) que se encarga de administrar.
- El proceso padre puede distribuir trabajo a los procesos hijo.

Multiprocesamiento (Multiprocessing)

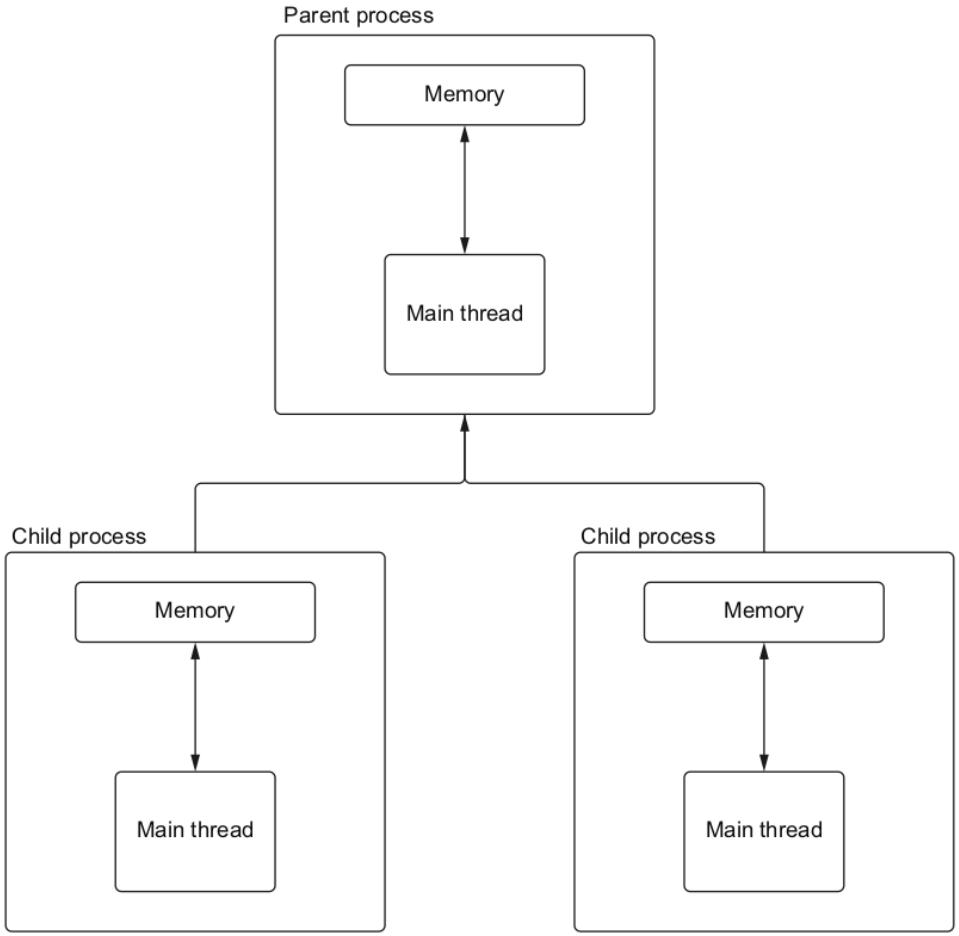


Figure 1.5 An application utilizing multiprocessing with one parent process and two child processes

Figura: Tomado de Fowler (2022)

Comunicación en concurrencia multiprocesamiento y multithreading

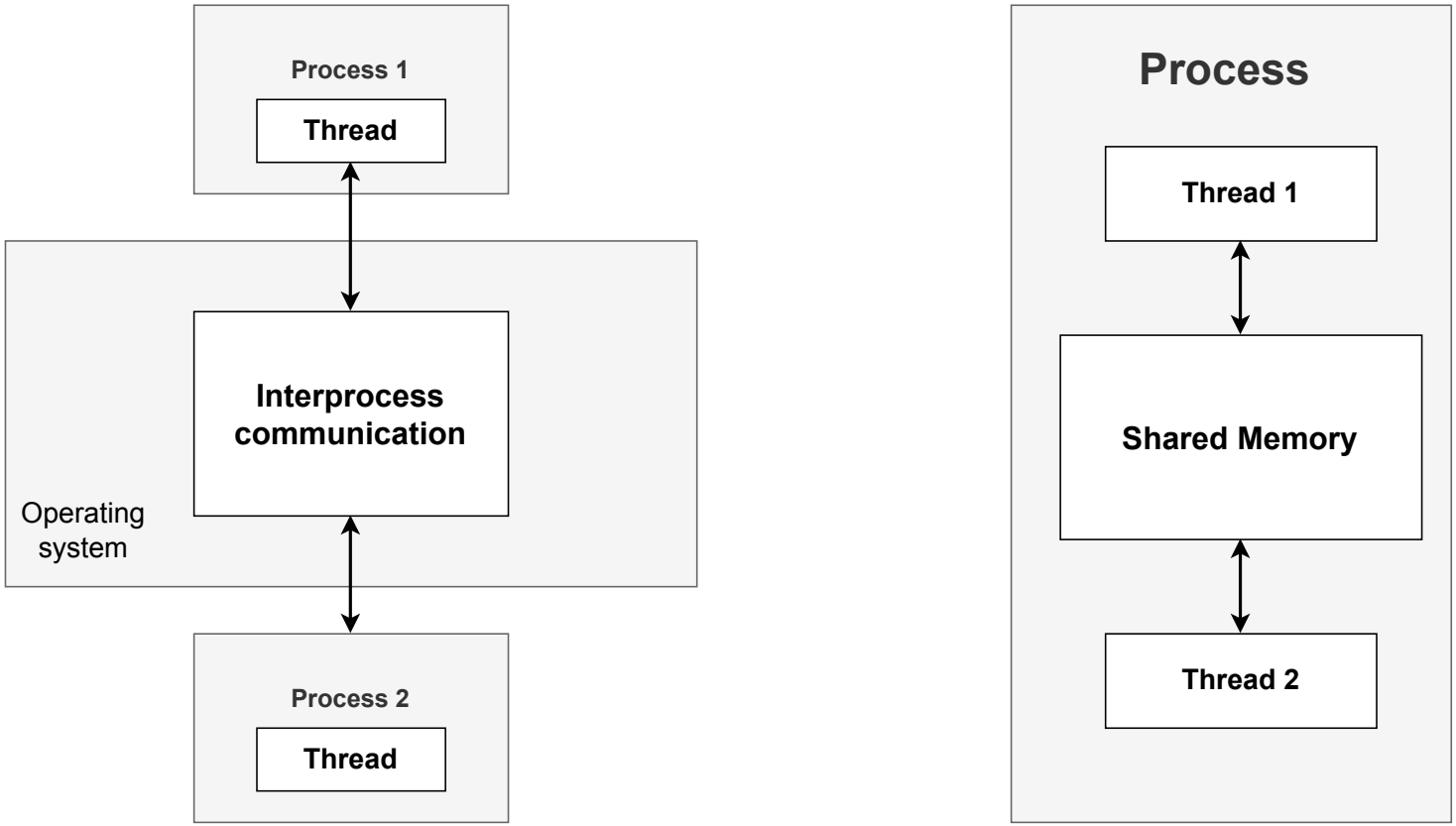


Figura: Adaptado de Fowler (2022)

Global Interpreter Lock

- El **Global Interpreter Lock** es uno de los temas más controversiales en la comunidad Python.
- El GIL previene que un proceso de Python ejecute más de una instrucción bytecode en cualquier momento.
- Esto significa que aunque tengamos una máquina multicore que ejecuta múltiples threads, un proceso de Python puede tener ejecutando sólo **un thread a la vez**.
- ¿Por qué existe el GIL? Por la forma como administra la memoria **CPython**[6]
- En CPython, la memoria es administrada principalmente por un proceso conocido como *reference counting*, el cual realiza un seguimiento de la cantidad de referencias que existen a un objeto en particular (diccionario, lista, conjunto, etc).

[6] CPython se puede entender como un *intérprete* y *compilador*, dado que compila el código de Python en un código de bytes antes de interpretarlo

¡Adiós GIL!

Yann LeCun
@ylecun

No more GIL!
Python code can now breathe multiple gulps of fresh air simultaneously.

> **Soumith Chintala** @soumithchintala · 29 jul.
> No More GIL!
> the Python team has officially accepted the proposal.
>
> Congrats @colesbury on his multi-year brilliant effort to remove the GIL, and a heartfelt thanks to the Python Steering Council and Core team for a thoughtful plan to make this a reality.
>
> discuss.python.org/t/a-steering-c...

12:52 a. m. · 30 jul. 2023 · **262,2 mil** Reproducciones

140 Retweets **17** Citas **1.375** Me gusta **92** Elementos guardados

Yann LeCun
@ylecun

An explanation of why eliminating the Global Interpreter Lock in Python enables more efficient multithreading.
The GIL elimination was made possible by a multiyear heroic effort led by Sam Gross at Meta.

Traducir Tweet

> **Dmytro Dzhulgakov** @dzhulgakov · 30 jul.
> GIL in Python will be no more. Huge win for AI ecosystem. Congrats to @colesbury - it took 4+ years of amazing engineering and advocacy.
>
> Many parts of @PyTorch could become simpler: DataLoader, Multi-gpu support (DDP), Python deployment (torch::deploy), ...
>
> Here's why. twitter.com/soumithchintal...

2:03 p. m. · 30 jul. 2023 · **199,2 mil** Reproducciones

114 Retweets **7** Citas **968** Me gusta **152** Elementos guardados

Figura: `https://twitter.com/ylecun/status/1685543873664229376?s=20`
`https://twitter.com/ylecun/status/1685742892071923712?s=20`

Global Interpreter Lock

- Buena noticia: multiprocesamiento (*i.e.*, multiples procesos corriendo de forma simultánea) **NO** es afectado por el GIL.
- **GIL sólo afecta a threads**.
- Podemos desarrollar soluciones paralelas en Python usando multiprocesamiento.

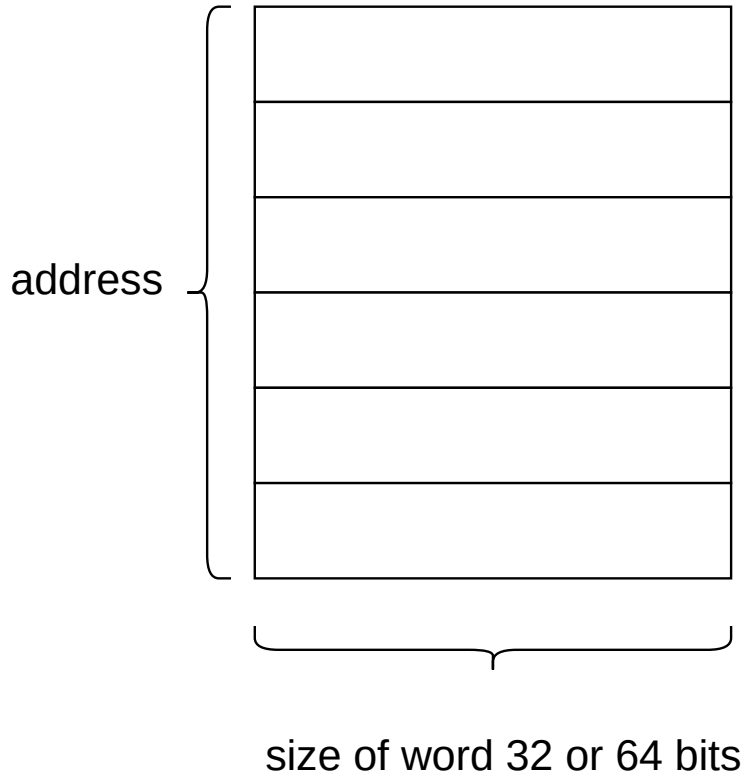
Global Interpreter Lock

- En teoría, el GIL afecta el desempeño de nuestros programas.
- En la práctica, rara vez es la fuente de problemas que no se pueden superar.
- Por ejemplo, el GIL es liberado cuando ocurren operaciones IO[7] dado que no estamos interacturando directamente con objetos de Python.
- Esto nos permite utilizar threads para hacer operaciones concurrentes de IO.

[7] Por ejemplo, hacer solicitudes a un servidor

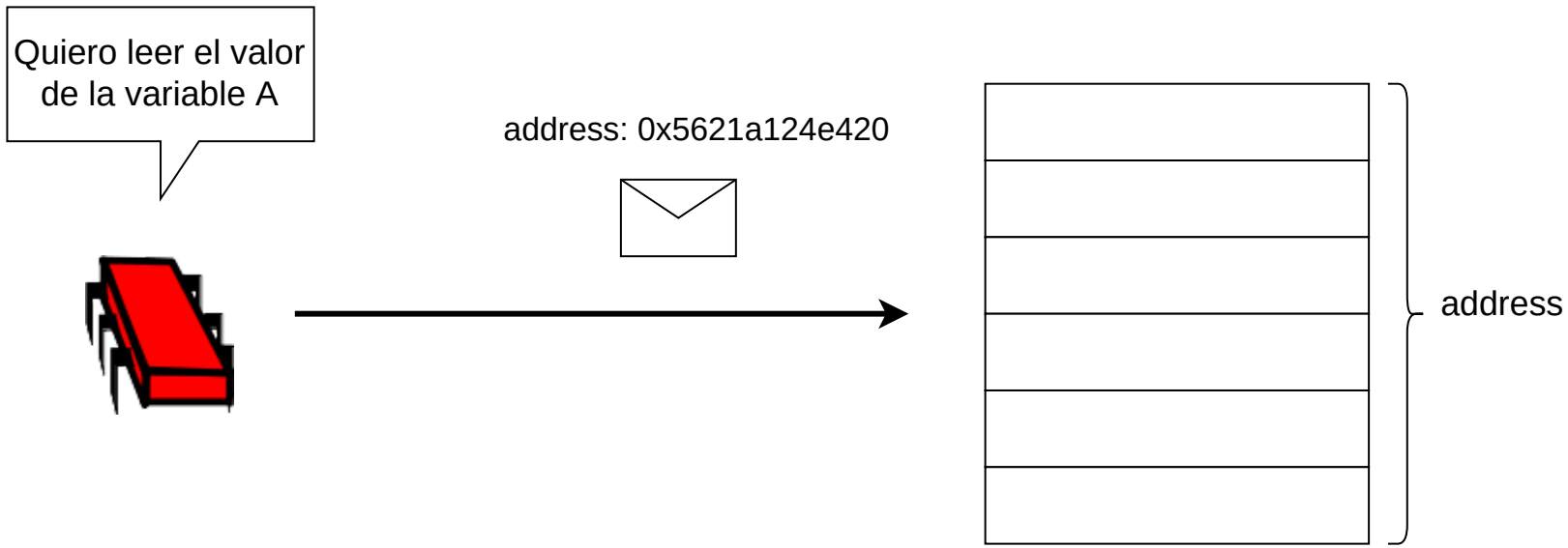
Memoria y cachés

- Los procesadores comparten una **memoria principal**, la cual es un arreglo de **palabras** (*words*), indizadas por una **dirección** (*address*).



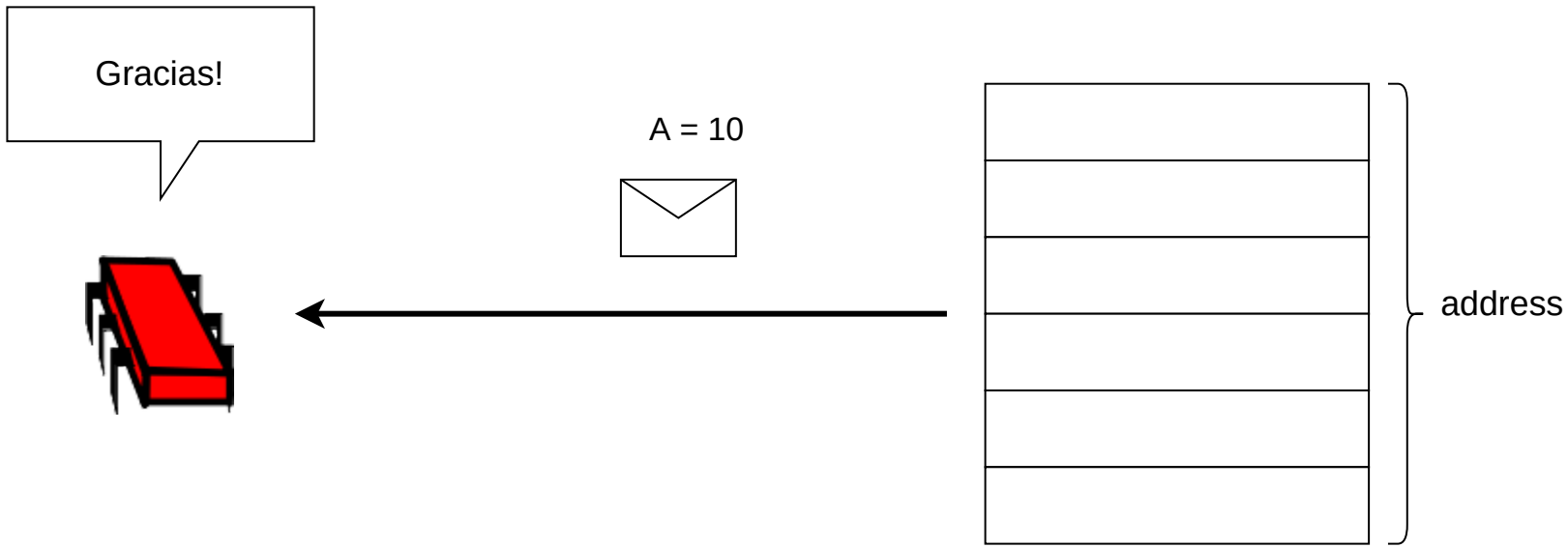
Memoria y cachés

Un procesador lee un valor de la memoria enviando un mensaje que contiene la dirección deseada a la memoria.



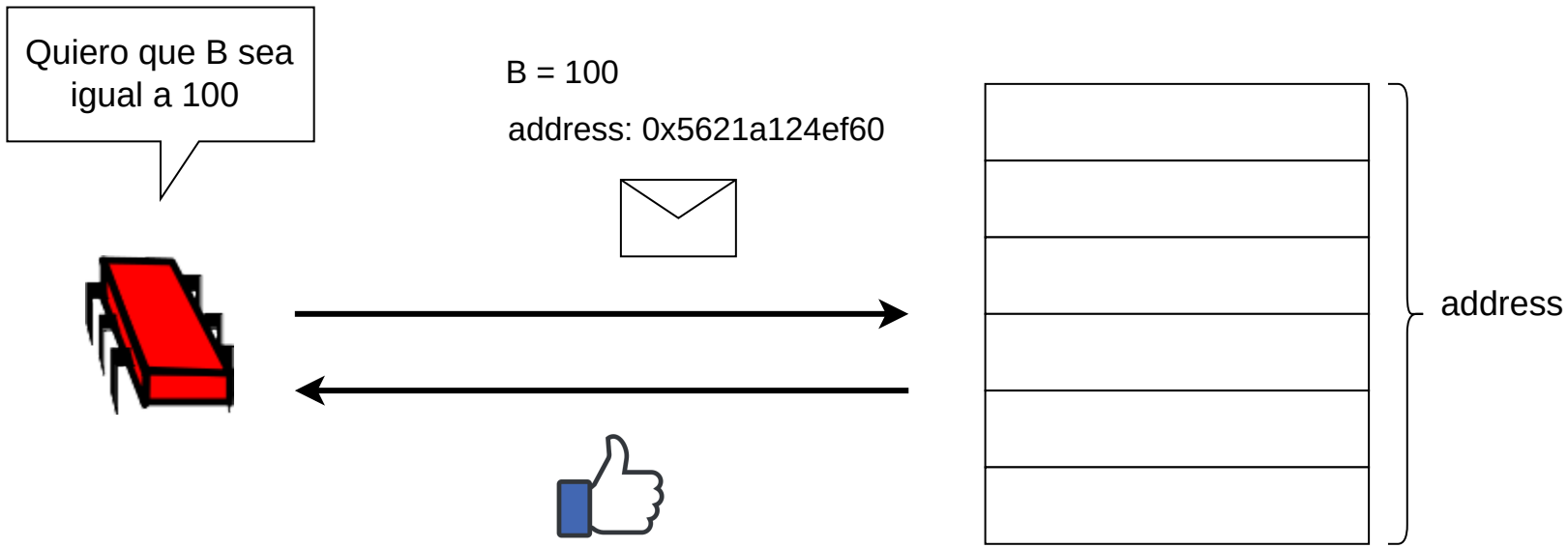
Memoria y cachés

El mensaje de respuesta contiene los datos asociados, es decir, el contenido de la memoria en esa dirección.



Memoria y cachés

Un procesador escribe un valor enviando la dirección y los nuevos datos a la memoria, y la memoria envía una confirmación cuando se han instalado los nuevos datos.



Memoria y cachés

RECORDEMOS

Principio de arquitectónico : los procesadores y la memoria principal están muy separados.

Un acceso a la memoria principal puede tomar cientos de ciclos[8], por lo que existe un peligro real de que un procesador pueda pasar gran parte de su tiempo simplemente esperando el memoria para responder a las solicitudes.

[8] **Ciclo**: el tiempo que tarda un procesador en buscar y ejecutar una sola instrucción.

Memoria y cachés

- Los sistemas modernos alivian este problema introduciendo uno o más **cachés**: pequeñas memorias que se sitúan más cerca de los procesadores y por lo tanto son más rápidos que la memoria principal.
- En términos lógicos, los cachés están situados entre el procesador y la memoria.
- La **memoria** es en realidad una **jerarquía de componentes** que almacenan datos, que van desde uno o más niveles de cachés pequeños y rápidos hasta una memoria principal grande y relativamente lenta:

Memoria y cachés

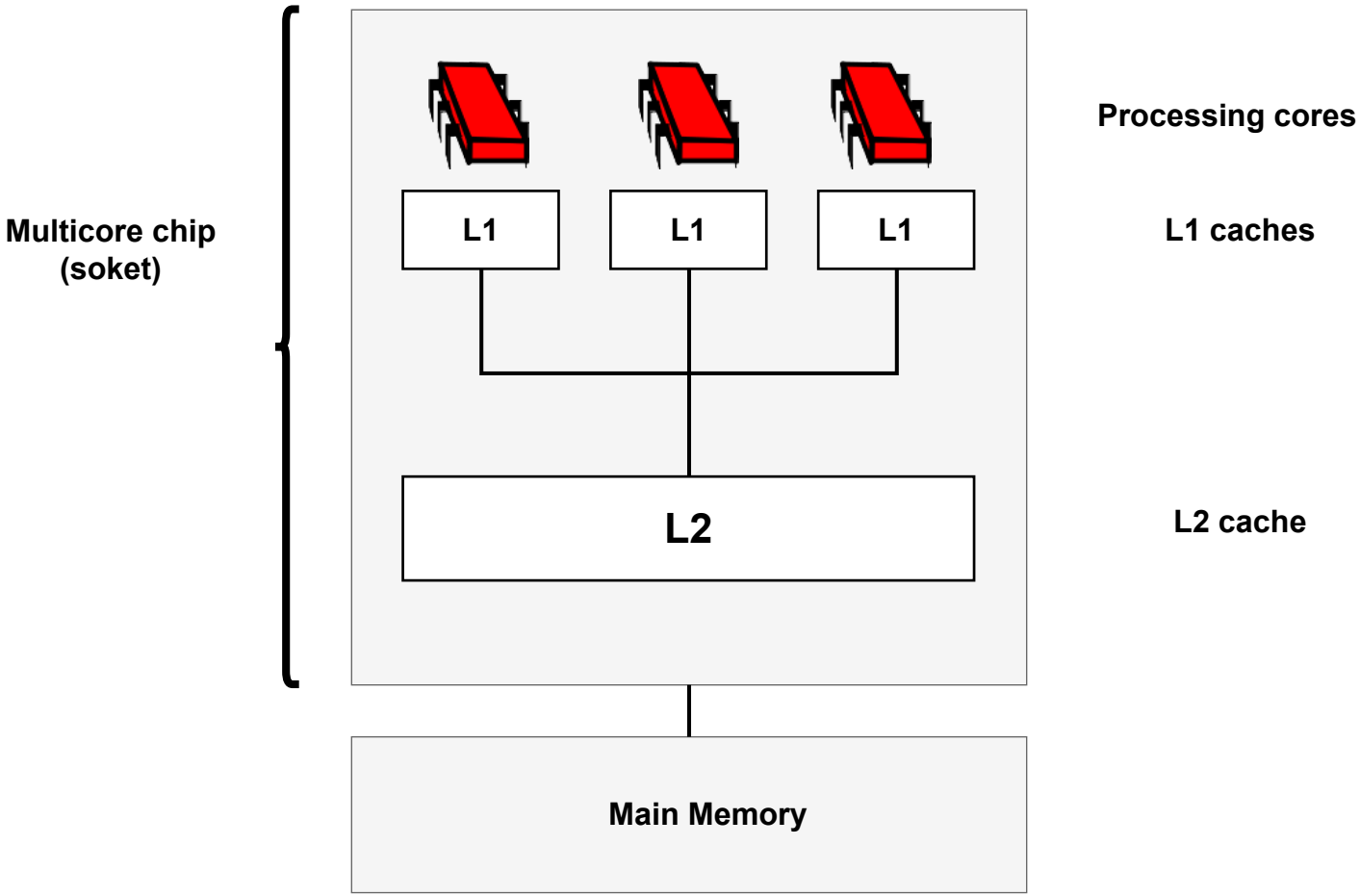


Figura: Arquitectura multicore SMP. El caché L2 está en el chip y es compartido por todos los cores. La memoria principal está fuera del chip. Adaptado de Herlihy et al. (2021)

Memoria y cachés

- Cuando un procesador intenta leer un valor de una dirección de memoria determinada, primero busca si el valor ya está en el caché y, de ser así, no necesita para realizar el acceso (más lento) a la memoria.
- Si el valor deseado se encuentra en el caché, se dice que el procesador logra un **hit** de caché y, de lo contrario, logra un **miss** de caché.
- Esta jerarquía de memoria mejora el rendimiento de la memoria en dos maneras:
  - Reduce la latencia de la memoria para los datos utilizados recientemente.
  - Reduce el número de accesos a la memoria principal, limitando así la uso de la interconexión de red y la demanda de ancho de banda.

Tiempo de acceso de la Jerarquía de Memoria

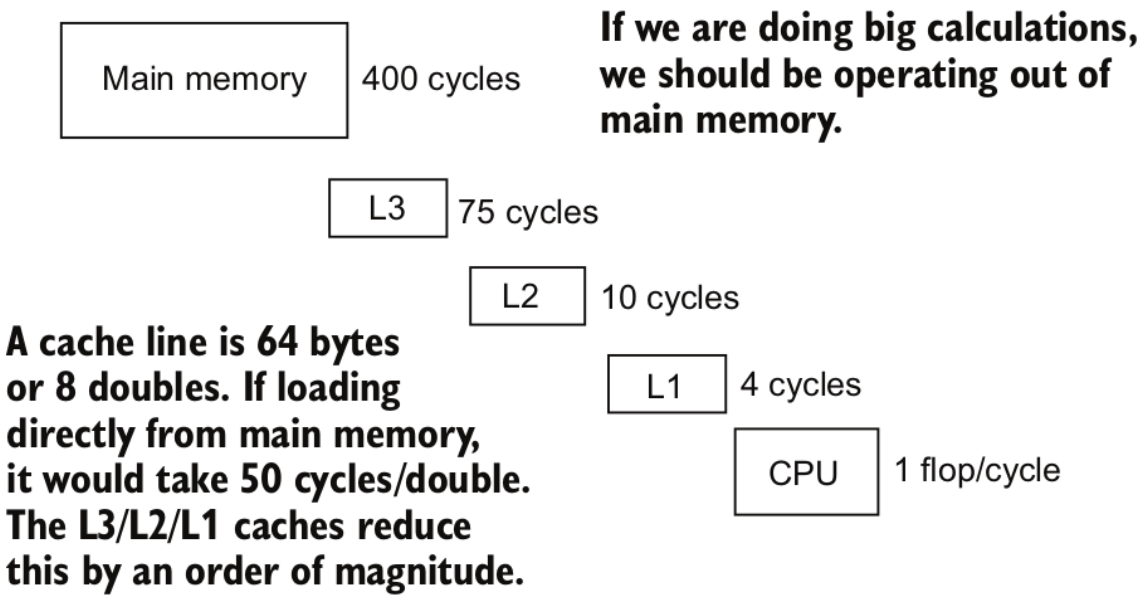


Figure 3.5 Memory hierarchy and access times. Memory is loaded into cache lines and stored at each level of the cache system for reuse.

Figura: Tomado de Robey and Zamora (2021)

Tiempo de acceso de la Jerarquía de Memoria

Table 6.1 Memory hierarchy with sizes and access times for a hypothetical, but realistic modern desktop

Type	Size	Access time
CPU		
L1 cache	256 KB	2 ns
L2 cache	1 MB	5 ns
L3 cache	6 MB	30 ns
RAM		
DIMM	8 GB	100 ns

Figura: Tomado de Rodrigues Antao (2023)

Granularidad

- Si un procesador lee o escribe una ubicación de memoria, también es probable que lea o escriba ubicaciones *cercanas*.
- Cuando se accede a una ubicación de memoria para una lectura, se copia en memoria caché un bloque completo de memoria
- Los bloques de memoria copiados de esta forma se denominan **líneas de caché**, un grupo de palabras vecinas contiguas.
- Los cachés suelen operar con una **granularidad** mayor que una sola palabra: un caché contiene un grupo de palabras vecinas, **líneas de caché**.

Operaciones vectorizadas

- Los procesadores tienen unidades vectoriales especiales que pueden cargar y operar en más de un elemento de datos a la vez.
- **Vectorización** es el proceso de agrupar operaciones para que se pueda realizar más de una a la vez.

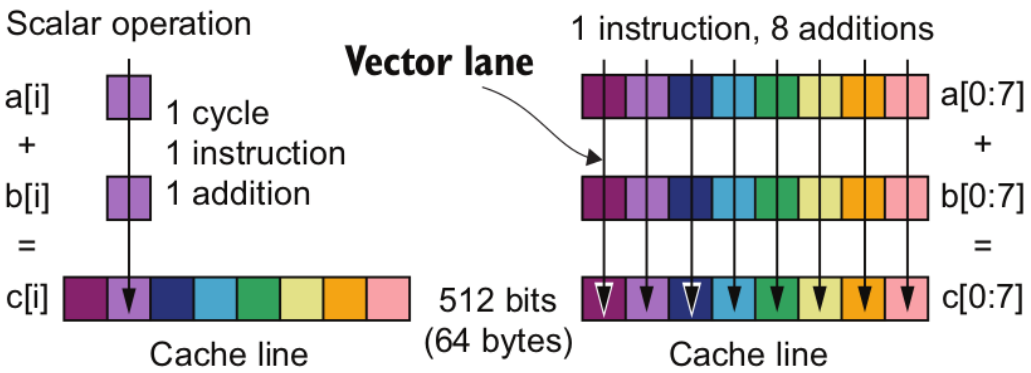


Figure 6.1 A scalar operation does a single double-precision addition in one cycle. It takes eight cycles to process a 64-byte cache line. In comparison, a vector operation on a 512-bit vector unit can process all eight double-precision values in one cycle.

Figura: Tomado de Rodrigues Antao (2023)

References I

Alessandrini, V. (2015). *Shared memory application programming: Concepts and strategies in Multicore application programming*. Morgan Kaufmann.

Fowler, M. (2022). *Python Concurrency with asyncio*. Manning Publications.

Herlihy, M., Shavit, N., Luchangco, V., and Spear, M. (2021). *The Art of Multiprocessor Programming*. Morgan Kaufmann, Boston, second edition edition.

Matloff, N. (2015). *Parallel computing for data science: with examples in R, C++ and CUDA*, volume 28. CRC Press.

Robey, R. and Zamora, Y. (2021). *Parallel and High Performance Computing*. Manning Publications, first edition edition.

Rodrigues Antao, T. (2023). *Fast Python: High performance techniques for large datasets*. Manning Publications.

Williams, A. (2019). *C++ concurrency in action*. Manning Publications.