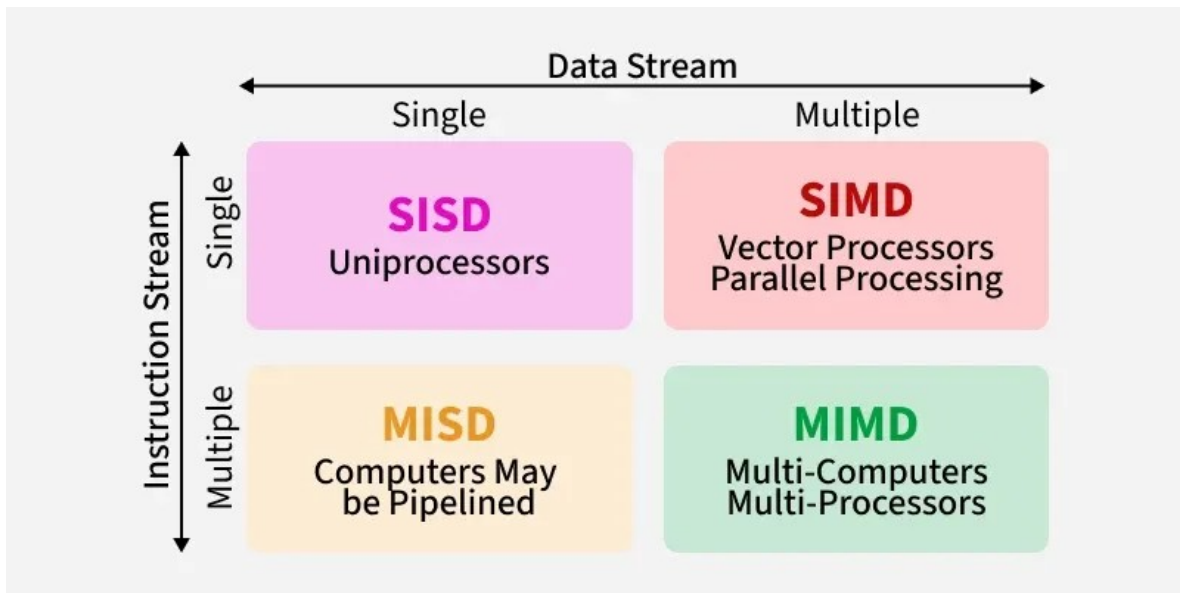


Arquitecturas Modernas de Sistemas

Arquitecturas Paralelas (Taxonomía de Flynn)

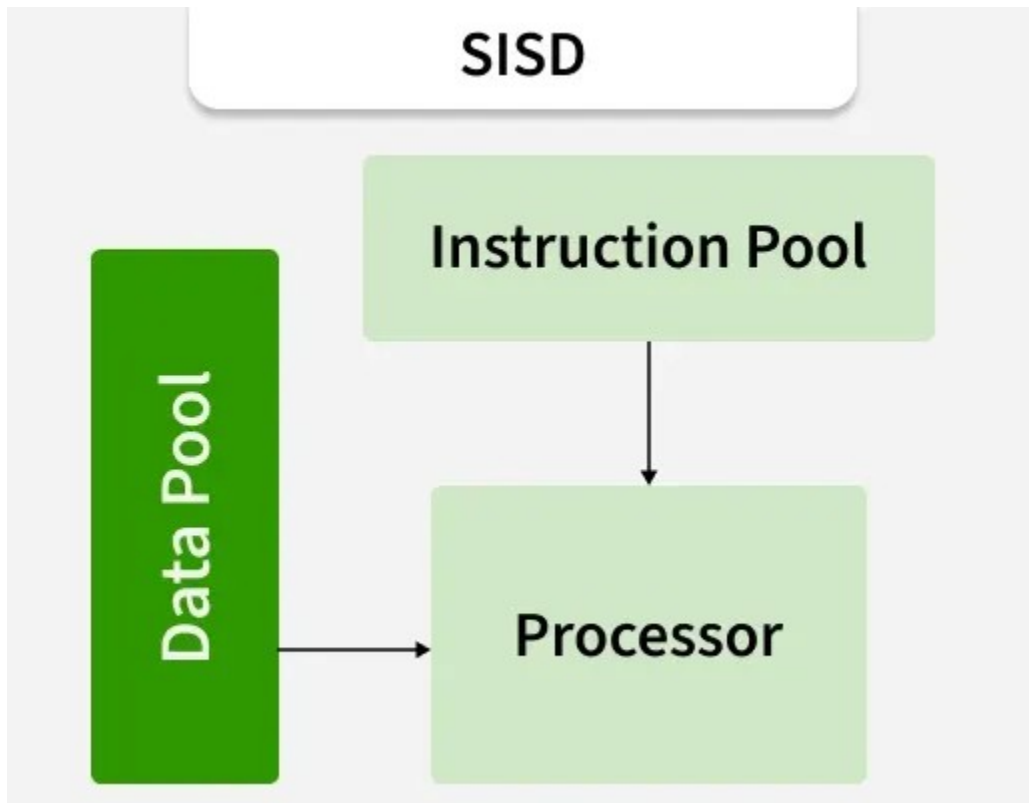


La taxonomía de Flynn es una clasificación fundamental de arquitecturas de computadoras propuesta por Michael Flynn en 1966, *basada en el número de flujos de instrucciones (tareas) y datos concurrentes*. Define cuatro tipos:

- SISD (serie - Single Instruction, Single Data)
- SIMD (paralelo vectorial - Single Instruction, Multiple Data)
- MISD (teórico - Multiple Instruction, Single Data)
- MIMD (multiprocesador paralelo - Multiple Instruction, Multiple Data)

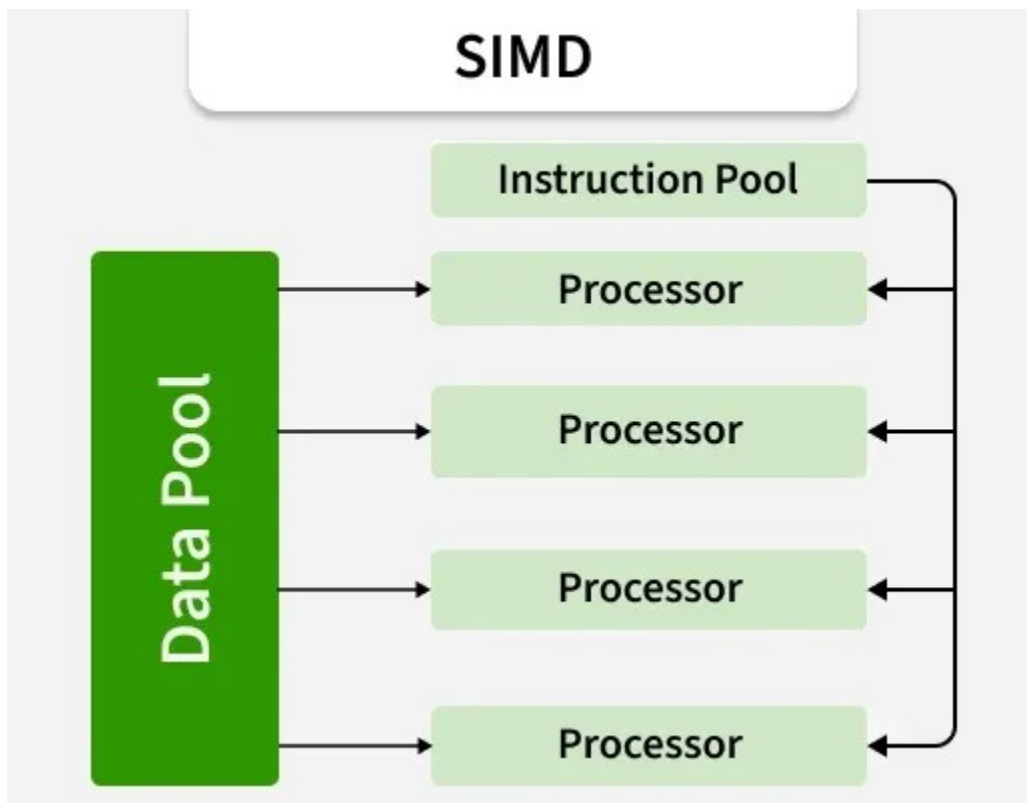
Siendo esencial para entender el diseño de procesadores modernos y el procesamiento paralelo.

SISD (Single Instruction, Single Data)



- Descripción: Un procesador ejecuta una sola instrucción sobre un único conjunto de datos.
- Las instrucciones de máquina se procesan de forma secuencial, y las computadoras de antes de ~2005 adoptan este modelo se conocen popularmente como computadoras secuenciales (SISD (un solo núcleo) - Intel Pentium 4, AMD Athlon) . La mayoría de las computadoras convencionales tienen arquitectura SISD. Todas las instrucciones y los datos que se van a procesar deben almacenarse en la memoria principal.
- Ejemplo: Procesadores Von Neumann tradicionales (un solo core)
- Uso actual: Sistemas embebidos simples, microcontroladores básicos, Computadoras personales antiguas, microcontroladores básicos.

SIMD (Single Instruction, Multiple Data)

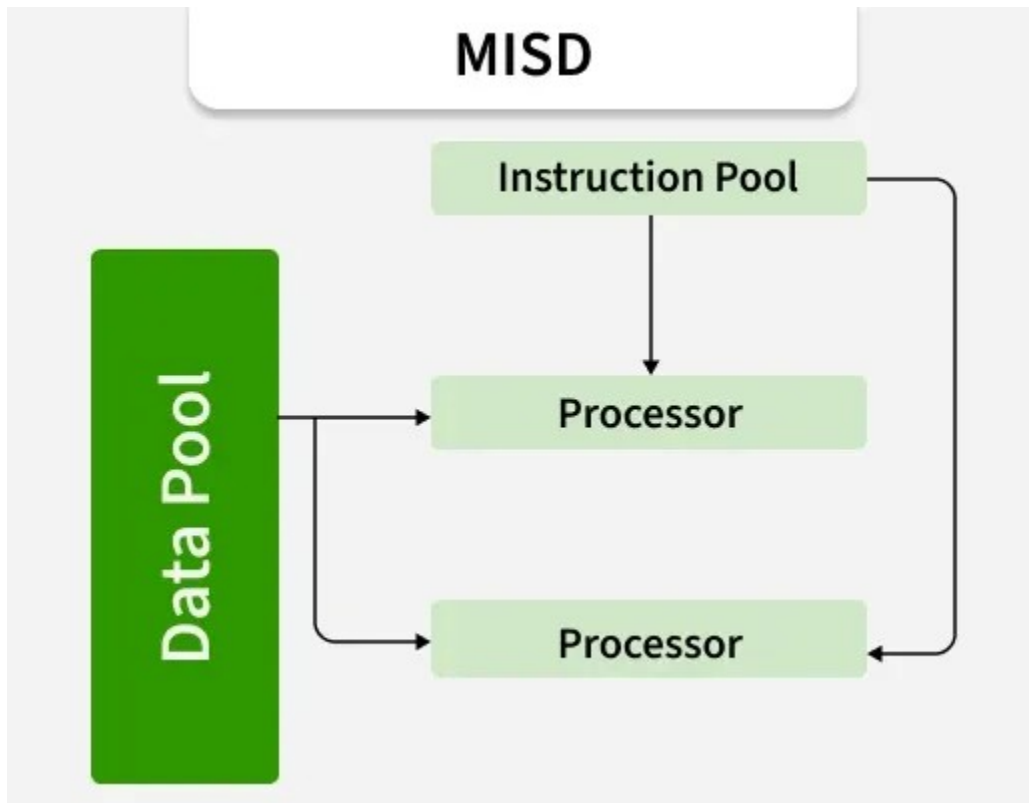


- Descripción: Una sola instrucción se ejecuta simultáneamente sobre múltiples datos
- Es una máquina multiprocesador capaz de ejecutar la misma instrucción en todas las CPU, pero operando con diferentes flujos de datos. Las máquinas basadas en un modelo SIMD son muy adecuadas para la computación científica, ya que implican numerosas operaciones con vectores y matrices.
- Para que la información pueda transmitirse a todos los elementos de procesamiento (PE), los elementos de datos organizados de los vectores pueden dividirse en múltiples conjuntos (N conjuntos para N sistemas PE (elementos de procesamiento)), y cada elemento de procesamiento (PE) puede procesar un conjunto de datos. El sistema SIMD más representativo son los procesadores gráficos.

Característica GPU	¿Por qué es SIMD?
Miles de "cores" simples	Todos ejecutan la misma instrucción al mismo tiempo
Pixels/shaders (sombreado) en paralelo	Cada core trabaja con datos diferentes (diferentes píxeles)
Misma operación para todos	Ej: "aplicar brillo" a miles de píxeles simultáneamente

- Características:
 - Ideal para procesamiento de vectores, matrices, gráficos, video
 - Extensiones en procesadores modernos: SSE, AVX (x86); NEON, SVE (ARM)
 - Más eficiente energéticamente que MIMD para tareas homogéneas
- Aplicaciones:
 - Procesamiento de imágenes y video (filtros, efectos)
 - Inteligencia Artificial y Machine Learning (operaciones matriciales)
 - Simulaciones científicas
 - Criptografía

MISD (Multiple Instruction, Single Data)

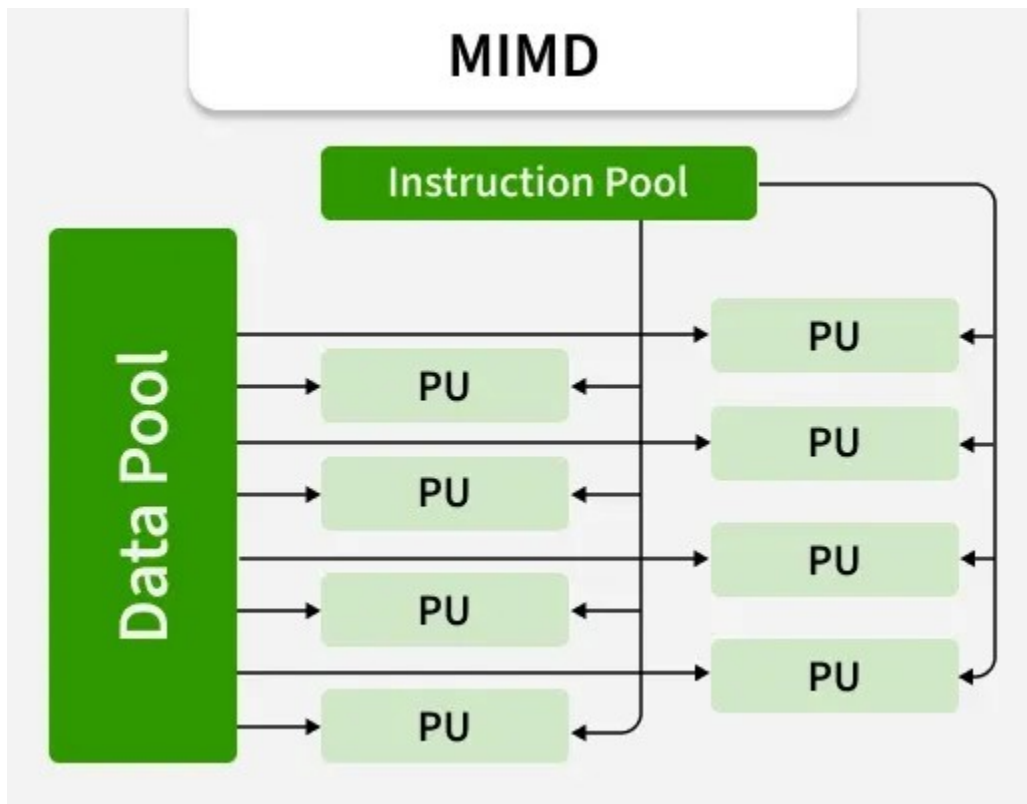


- Descripción: Múltiples flujos de instrucciones operan sobre un único dato
- Es una máquina multiprocesador capaz de ejecutar diferentes instrucciones en diferentes procesadores, pero todos ellos operando sobre el mismo conjunto de datos.
- Ejemplo $Z = \sin(x) + \cos(x) + \tan(x)$. El sistema realiza diferentes operaciones sobre el mismo conjunto de datos. Las máquinas construidas con el modelo MISD no son útiles en la mayoría de las aplicaciones; se han construido algunas, pero ninguna está disponible comercialmente.
- Raro en la práctica, pero se encuentra en:
 - Sistemas de tolerancia a fallos (redundancia)
 - Procesamiento de señales con múltiples filtros sobre la misma señal
 - Ejemplos teóricos: Computadoras tolerantes a fallos (mismos datos, diferente algoritmo para verificar).

- MISD es principalmente un concepto teórico. El único ejemplo "real" aceptado son los sistemas de tolerancia a fallos (como el Space Shuttle) donde múltiples computadoras ejecutan diferentes algoritmos sobre los mismos datos para detectar errores por consenso.
- Para efectos prácticos, en computación moderna no vas a encontrar un procesador MISD comercial. Es más un caso de estudio académico que una arquitectura que se use a diario.

Ejemplo	¿Por qué es MISD?	Contexto
Computadoras de vuelo del Space Shuttle	Múltiples computadoras ejecutaban diferentes algoritmos sobre los mismos datos de sensores y votaban para detectar errores	Tolerancia a fallos crítica
Sistemas de redundancia en aviónica	Varias unidades de procesamiento aplican distintas instrucciones al mismo dato para verificar resultados	Seguridad aeroespacial
Systolic arrays	Red de nodos donde datos fluyen y cada nodo aplica una operación diferente en pipeline. (En una arquitectura MISD tipo pipeline, un único flujo de datos de entrada va pasando de un procesador a otro. Cada procesador ejecuta una instrucción o etapa diferente, lo que significa que el pipeline global se reparte a lo largo de la red de nodos.)	Procesamiento de señales, IA (TPUs)
Múltiples filtros sobre una misma señal	Diferentes algoritmos de filtrado (paso alto, paso bajo, etc.) procesan la misma señal de audio/radar en paralelo	Procesamiento de señales

MIMD (Multiple Instruction, Multiple Data)



- Descripción: Múltiples procesadores ejecutan diferentes instrucciones sobre diferentes datos simultáneamente
- Un sistema MIMD es una máquina multiprocesador capaz de ejecutar múltiples instrucciones sobre múltiples conjuntos de datos. Cada procesador (PE) en el modelo MIMD tiene flujos de instrucciones y datos independientes; por lo tanto, las máquinas construidas con este modelo pueden ejecutar cualquier tipo de aplicación. A diferencia de las máquinas SIMD y MISD, los procesadores en las máquinas MIMD funcionan de forma asíncrona.
- Las máquinas MIMD se clasifican en modelos de memoria compartida y memoria distribuida, según cómo se conectan los procesadores a la memoria. En un sistema MIMD de memoria compartida (acoplamiento estrecho), todos los procesadores utilizan la misma memoria global y la comunicación se realiza a través de ella. Cualquier cambio realizado por un procesador es visible para todos los demás. Ejemplos de ello son los sistemas SMP de Silicon Graphics y Sun/IBM.

- En un sistema MIMD de memoria distribuida (acoplamiento flexible), cada procesador tiene su propia memoria local y se comunican a través de una red de interconexión (como un árbol o una malla).
- Los sistemas de memoria compartida son más fáciles de programar, pero más difíciles de escalar y más vulnerables a fallos, ya que un fallo puede afectar a todo el sistema. Por el contrario, los sistemas de memoria distribuida son más escalables y tolerantes a fallos, puesto que cada procesador es independiente. En la práctica, los sistemas MIMD de memoria distribuida se consideran generalmente superiores para la computación de alto rendimiento y a gran escala, debido a su escalabilidad, fiabilidad y capacidad para gestionar tareas complejas de forma eficiente. En los los sistemas MIMD de memoria distribuida, si el Procesador A quiere un dato del Procesador B, no puede acceder a su memoria directamente; tiene que mandarle un mensaje a través de una red de comunicación.
- Tipos:
 - Multiprocesadores (memoria compartida): Todos los cores acceden a una memoria común
 - Multicomputadores (memoria distribuida): Cada procesador tiene su propia memoria, se comunican por mensajes
- Ejemplos modernos:
 - CPUs multicore actuales (Intel Core, AMD Ryzen, Apple Silicon)
 - Supercomputadoras
 - Clusters de servidores

	CPU (Intel/AMD)	GPU (NVIDIA/AMD)
Arquitectura Flynn	MIMD (cores independientes) + SIMD (AVX)	SIMD masivo
Número de cores	8-64 cores complejos	10,000+ cores simples
Tipo de tareas	Todo tipo (general)	Tareas homogéneas y repetitivas
Ejemplo	Navegar, compilar, bases de datos	Renderizado 3D, IA, minería

GPU = SIMD puro y extremo

CPU moderna = MIMD (multicore) con unidades SIMD dentro de cada core

Ambas usan SIMD, pero las GPUs lo llevan al máximo porque están diseñadas específicamente para tareas donde la misma operación se repite millones de veces sobre datos diferentes.

Analogía

- **SISD** = Un cocinero, una receta, un plato (Von Neumann clásico)
- **SIMD** = 8 cocineros, **misma receta**, 8 platos diferentes (GPU procesando píxeles)
- **MISD** = 2 cocineros, **recetas diferentes**, mismo plato (tolerancia a fallos)
- **MIMD** = 8 cocineros, cada uno con su receta y sus ingredientes (CPU multicore moderna)

Arquitectura	¿Existe masivamente?	Ejemplos cotidianos
SISD	Obsoleto	PCs de los 90s
SIMD	Sí	GPUs, AVX/NEON en CPUs
MISD	Solo en nichos críticos	Space, aviónica militar
MIMD	Sí	Tu laptop, tu celular, servidores

<https://www.geeksforgeeks.org/computer-organization-architecture/computer-architecture-flynns-taxonomy/>

Modelos de memoria MIMD

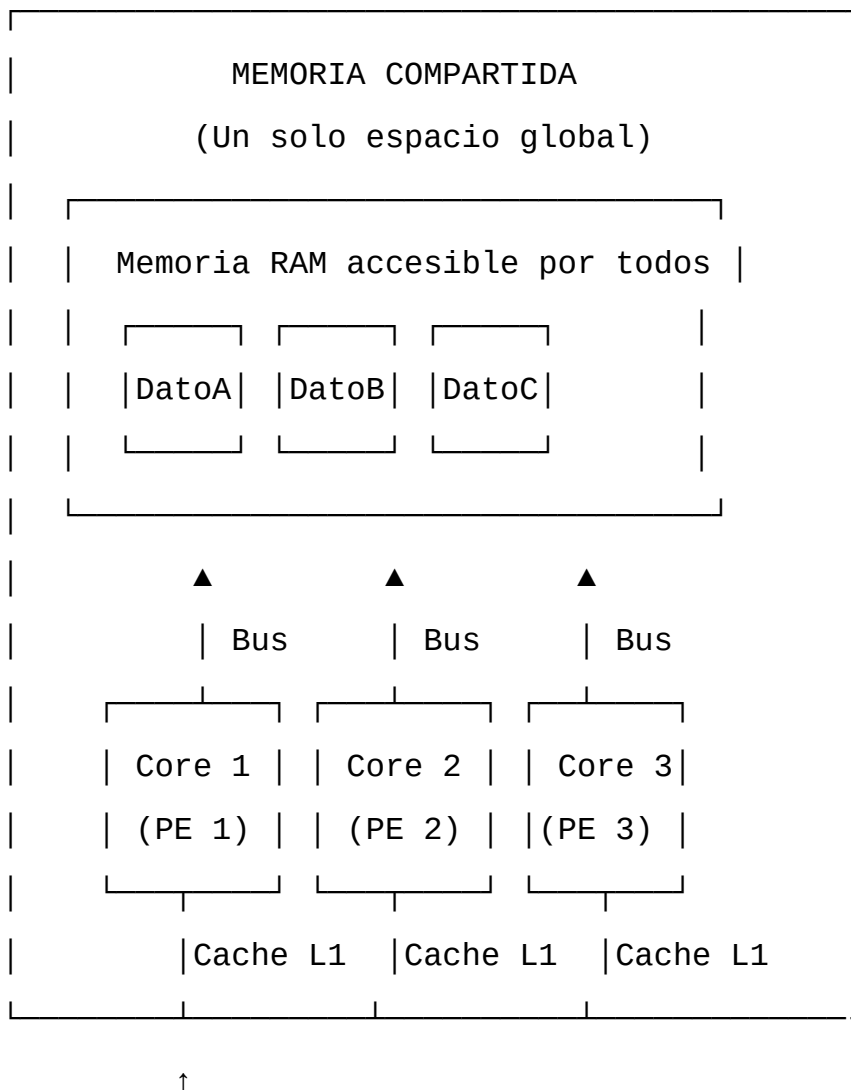
MIMD = Múltiples procesadores independientes

En MIMD, cada procesador tiene su **propio flujo de instrucciones** y su **propio flujo de datos**. Es como tener varias computadoras trabajando en paralelo, pero conectadas entre sí.

La pregunta clave es: **¿Cómo comparten información esos procesadores?**

Ahí entran los dos modelos de memoria:

Modelo 1: Memoria Compartida (Shared Memory)



Todos ven los mismos datos

Si Core 1 modifica DatoA, Core 2 lo ve

Características

Aspecto	Detalle
Cómo se comunican	Leyendo/escribiendo en la misma memoria
Programación	Más fácil (parece un solo programa con hilos)
Ejemplo físico	Tu laptop multicore (Intel Core, AMD Ryzen, Apple Silicon)
Arquitectura	SMP (Symmetric Multi-Processing - Multiprocesamiento Simétrico) - Arquitectura de hardware en la que dos o más procesadores (o núcleos) idénticos comparten una única memoria RAM central y un solo sistema operativo. La palabra "Simétrico" es la clave: significa que ningún procesador es el "jefe" ni tiene privilegios sobre los demás. Cualquier procesador puede ejecutar cualquier tarea del sistema operativo o de tus aplicaciones.

Ventajas

- **Fácil de programar:** Usas pthread, OpenMP o simplemente hilos de tu lenguaje favorito
- **Compartir datos es rápido:** No hay que "enviar" nada, solo leer de memoria

Desventajas

- **Problema de coherencia de caché:** Si Core 1 y Core 2 tienen copias del mismo dato en caché, ¿cuál es el valor real?
- **No escala bien:** A más procesadores, el bus de memoria se satura ("cuello de botella")
- **Vulnerable a fallos:** Si falla la memoria compartida, todo el sistema cae

Ejemplo de código (conceptual)

c

```
// Ambos cores acceden a la misma variable global  
  
int contador = 0; // En memoria compartida
```

// Core 1 ejecuta:

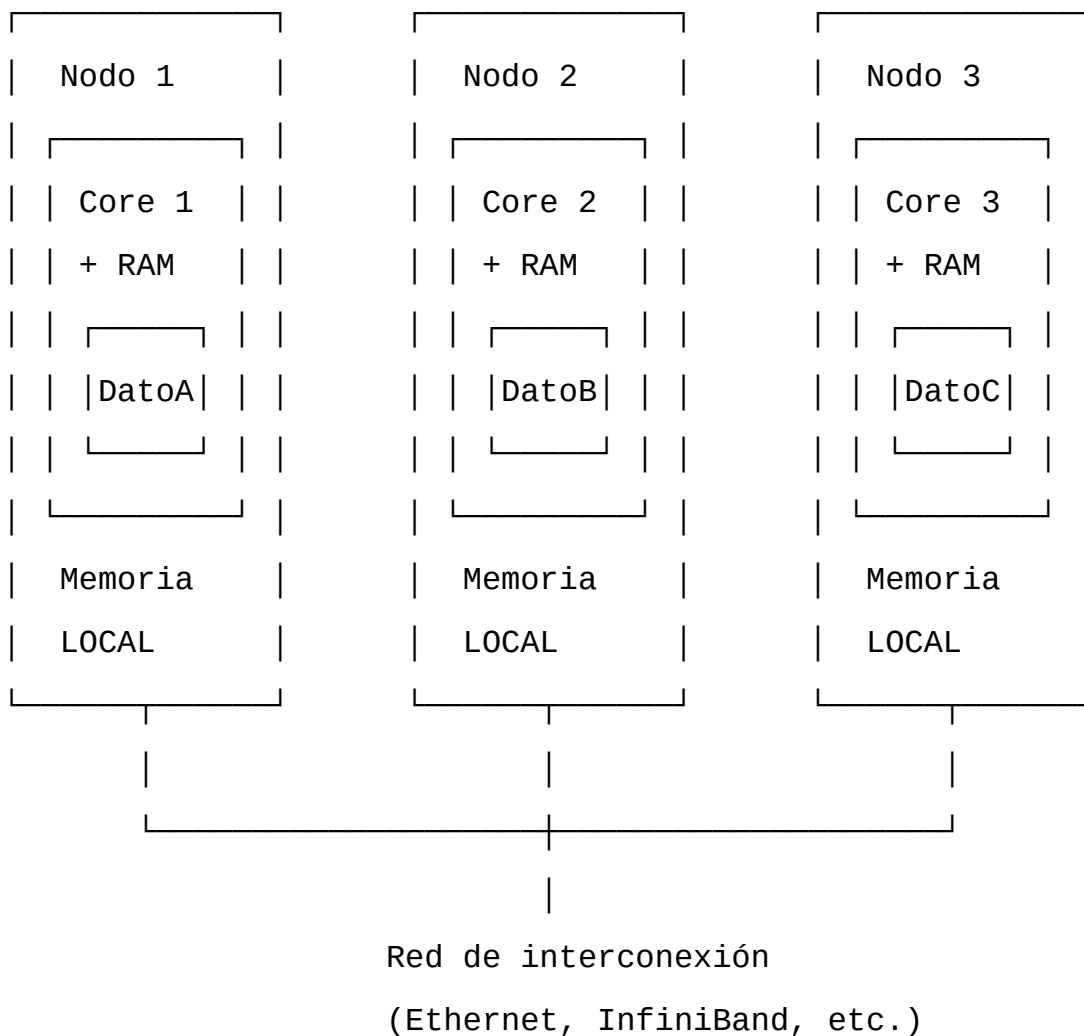
contador = contador + 1;

// Core 2 ejecuta simultáneamente:

contador = contador + 1;

// ⚠ PROBLEMA: ¿El resultado es 2? ¡No siempre! (race condition)

Modelo 2: Memoria Distribuida (Distributed Memory)



Características

Aspecto	Detalle
Cómo se comunican	Enviando mensajes a través de la red
Programación	Más compleja (necesitas "enviar" y "recibir" datos explícitamente)
Ejemplo físico	Clusters de servidores, supercomputadoras
Arquitectura	MPP (Massively Parallel Processing - Procesamiento Masivamente Paralelo). + Arquitectura "Shared-Nothing" (Nada compartido): Cada nodo es una computadora completa en sí misma. No comparten CPU, ni memoria RAM, ni almacenamiento de disco. + Escalabilidad masiva: Es increíblemente fácil de expandir. Si necesitas más potencia, simplemente añades más nodos a la red. + Puede escalar a petabytes de datos sin los cuellos de botella de la memoria compartida. Paso de mensajes: Como el Procesador A no puede ver la memoria del Procesador B, se comunican enviándose "mensajes" con datos a través de la red interna.

Ventajas

- **Escalabilidad masiva:** Puedes tener miles de nodos
- **Tolerancia a fallos:** Si un nodo falla, los demás siguen funcionando
- **Cada nodo es independiente:** No hay contención por el bus de memoria

Desventajas

- **Programación más difícil:** Debes gestionar explícitamente qué datos enviar a quién.
 - El Procesador 1 tiene su propia memoria RAM y el Procesador 2 la suya. El Procesador 2 no tiene forma física de mirar dentro de la memoria del Procesador 1.
- **Comunicación lenta:** Enviar datos por red es mucho más lento que acceder a RAM local.
- **Duplicación de datos:** Si varios nodos necesitan el mismo dato, debe existir en varias memorias.

- Al no existir una memoria RAM central donde todos puedan mirar el mismo archivo, el sistema se ve obligado a hacer "fotocopias" idénticas del mismo dato y guardarlas en la memoria privada de cada nodo que lo necesite.
- Dos grandes consecuencias de la duplicación: Desperdicio de memoria RAM e El problema de la actualización (Inconsistencia).

Comparativa directa

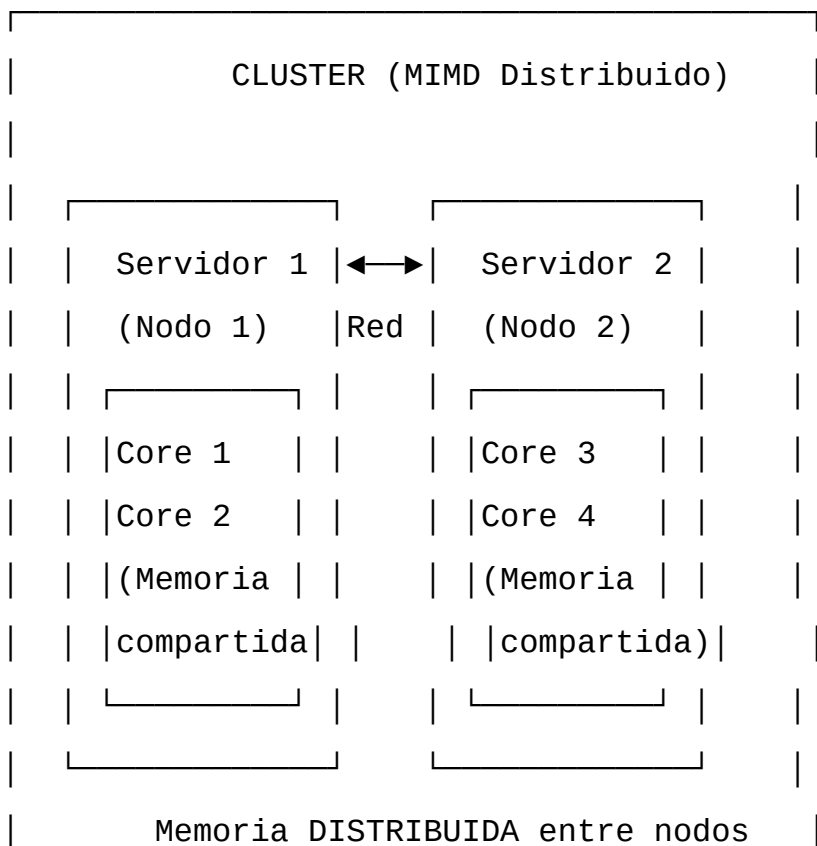
Característica	Memoria Compartida	Memoria Distribuida
Espacio de memoria	Uno solo, global	Cada nodo tiene el suyo
Comunicación	Implícita (leer/escribir)	Explícita (mensajes)
Velocidad de acceso	Rápida (nanosegundos)	Lenta (microsegundos a milisegundos)
Escalabilidad	Limitada (docenas de cores)	Masiva (miles de nodos)
Tolerancia a fallos	Baja (un punto de fallo)	Alta (nodos independientes)
Ejemplos	Tu PC, laptop, servidor SMP	Google Cloud, AWS clusters, supercomputadoras

Analogía

	Memoria Compartida	Memoria Distribuida
Analogía	Una cocina con varios cocineros compartiendo la misma mesa de trabajo	Varias cocinas en diferentes casas, comunicándose por teléfono
Compartir receta	Dejas la receta en la mesa, todos la leen	Llamas por teléfono y le dictas la receta al otro cocinero
Problema	Si la mesa es pequeña, se amontonan	La llamada telefónica es lenta y puedes tener información desactualizada

Híbridos: Lo mejor de ambos mundos

Muchos sistemas modernos combinan ambos:



Memoria COMPARTIDA dentro de cada
nodo (multicore)

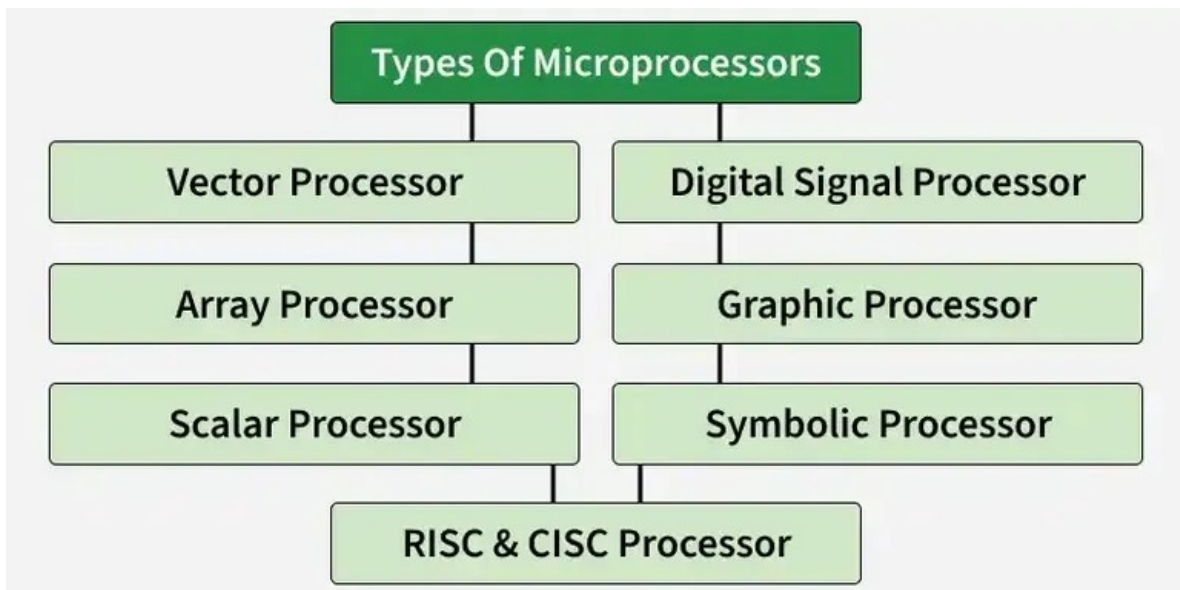
Ejemplo real: Una supercomputadora como Summit (IBM/Oak Ridge):

- Cada nodo tiene múltiples GPUs y CPUs que comparten memoria
- Pero hay miles de nodos conectados por red, cada uno con su propia memoria

Conclusión

Memoria Compartida	Para paralelismo "ligero" en una sola máquina (tu laptop, tu servidor)
Memoria Distribuida	Para paralelismo "masivo" entre muchas máquinas (la nube, supercomputadoras)

TIPOS DE PROCESAMIENTO: ESCALAR VS VECTORIAL



Procesamiento Escalar (Ejemplo SISD)

Es un procesador que ejecuta datos escalares. El procesador escalar más simple procesa únicamente instrucciones enteras utilizando operandos de punto fijo

A diferencia del punto flotante, donde la "coma" se mueve para manejar números muy grandes o muy pequeños, en el punto fijo la posición del punto decimal no cambia durante los cálculos).

Un procesador escalar potente procesa tanto enteros como números de punto flotante. Contiene una ALU (Unidad Aritmético-Lógica) de enteros y una unidad de punto flotante (FPU) en el mismo chip de CPU. Un procesador escalar puede ser CISC o RISC. Un procesador superescalar contiene múltiples pipelines y ejecuta más de una instrucción por ciclo de reloj.

Una sola instrucción opera sobre un único dato a la vez. Es el modelo tradicional de computación secuencial.

Características:

- Ejecución paso a paso, una operación por ciclo
- Instrucción simple: Operación $\rightarrow$ Dato único $\rightarrow$ Resultado único
- Base de la arquitectura Von Neumann clásica

Ejemplo visual — Sumar 8 veces $a_n + b_n$:

Sin vectorial (escalar):

Ciclo 1: ADD $a_0, b_0 \rightarrow r_0$

Ciclo 2: ADD $a_1, b_1 \rightarrow r_1$

Ciclo 3: ADD $a_2, b_2 \rightarrow r_2$

...

Ciclo 8: ADD $a_7, b_7 \rightarrow r_7$

Total: 8 ciclos de reloj

Analogía: Como un cocinero preparando un plato a la vez. Eficiente en trabajo detallado, pero lento para tareas repetitivas masivas.

Procesamiento Vectorial (Ejemplo SIMD)

Un tipo de CPU diseñada para realizar operaciones matemáticas con múltiples elementos de datos (vectores) simultáneamente, a diferencia de un procesador escalar que procesa un elemento a la vez. Esto permite una ejecución más rápida y eficiente de cálculos matemáticos y científicos. Se utilizan comúnmente en supercomputadoras, servidores y algunas GPU especializadas.

Una sola instrucción opera simultáneamente sobre múltiples datos agrupados en un "vector" (array de datos).

Características:

- Una instrucción → múltiples datos → múltiples resultados
- Los datos deben ser del mismo tipo y operación homogénea
- Requiere registros vectoriales anchos (128, 256, 512 bits)

Ejemplo visual — Sumar 8 números:

Con vectorial (SIMD):

Ciclo 1: ADD $[a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7], [b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7]$

→ $[r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7]$

Total: 1 ciclo de reloj (8 operaciones en paralelo)

Analogía: Como una cadena de ensamblaje donde 8 cocineros preparan 8 platos idénticos simultáneamente.

Comparativa Escalar vs Vectorial

Característica	Procesamiento Escalar	Procesamiento Vectorial
Instrucciones	Una por dato	Una para múltiples datos
Paralelismo	Ninguno (secuencial)	Paralelo de datos
Ciclos necesarios	N (para N datos)	1 (para N datos, hasta el ancho del vector)
Aplicaciones ideales	Lógica de control, decisiones condicionales, código secuencial	Procesamiento de imágenes, video, audio, IA, matrices, gráficos
Limitación	No aprovecha paralelismo inherente en datos	Requiere datos homogéneos y operaciones idénticas
Ejemplo arquitectura	CPU tradicional Von Neumann	Extensiones SIMD: SSE, AVX (x86); NEON, SVE (ARM)

Aplicaciones Prácticas del Procesamiento Vectorial

Aplicación	Ejemplo concreto	Beneficio del vectorial
Procesamiento de imágenes	Aplicar filtro de brillo a 4K foto	Procesar 8-16 píxeles simultáneamente
Video	Codificación H.264/H.265	Compresión en tiempo real
Audio	Aplicar ecualizador a pista musical	Procesar múltiples muestras en paralelo
Criptografía	Cifrado AES, hashing SHA	Cifrar múltiples bloques simultáneamente
Simulaciones científicas	Dinámica de fluidos, predicción climática	Cálculos masivos en supercomputadoras

<https://www.geeksforgeeks.org/computer-organization-architecture/types-of-microprocessors/>

Componentes Básicos de una Computadora

- Unidad Central de Procesamiento (CPU)
 - Función: Ejecuta instrucciones y procesa datos.
 - Subcomponentes:
 - ALU (Unidad Aritmético-Lógica): Realiza operaciones matemáticas.
 - Unidad de Control: Coordina las operaciones.
 - Registros: Almacenamiento temporal de datos e instrucciones.
- Memoria
 - Tipos:
 - RAM: Volátil, almacena datos e instrucciones en ejecución.
 - ROM: No volátil, guarda firmware (ej: BIOS).
 - Caché: Memoria rápida para acceso frecuente.
- Dispositivos de Entrada/Salida (E/S)
 - Ejemplos: Teclado, monitor, discos duros, tarjetas de red.
- Buses
 - Función: Canales de comunicación entre componentes.
 - Tipos:
 - Bus de Datos: Transfiere datos.
 - Bus de Direcciones: Indica ubicaciones en memoria.
 - Bus de Control: Gestiona operaciones.

