



**UNCUYO**  
UNIVERSIDAD  
NACIONAL DE CUYO



FACULTAD  
DE INGENIERÍA

Licenciatura en Ciencias de la  
Computación

# Arquitecturas Distribuidas

Unidad 1

**Modelos y Arquitecturas Escalables**



**Temas**

→ **Tipos de sistemas paralelos (Clasificación de Flynn)**

Paralelismo a nivel de instrucción

Paralelismo a nivel de hilos (Procesadores multihilo simultáneo)

Paralelismo a nivel de núcleos

Arquitecturas multi núcleo simétricas

Arquitecturas multi núcleo heterogéneas con igual ISA o con ISA diferentes

Sistemas CPU-GPU, CPU-DSP. GPGPU. Introducción a CUDA y OpenCL

Sistemas operativos para multiprocesadores.

Paralelismo a nivel de procesos (Cluster)

Paralelismo a nivel de datos (SIMD)

Sistemas RAID

Performance y escalabilidad

Speedup y eficiencia

Modelos de problemas paralelizables

Paralelismo en el software



## Tipos de sistemas paralelos Clasificación de Flynn

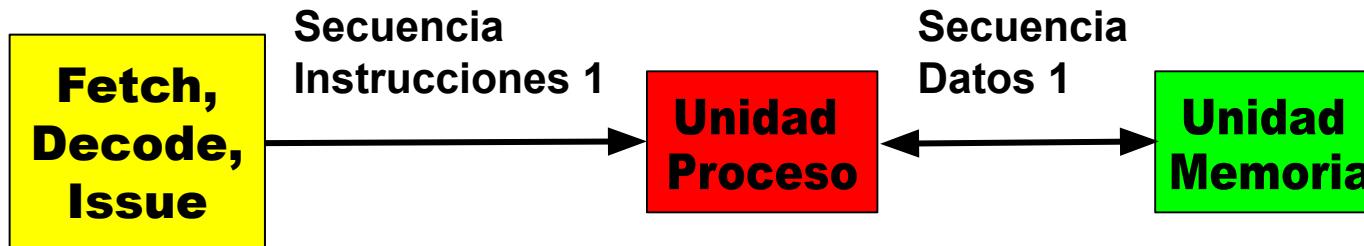
- **SISD:** Single Instruction, Single Data
- **SIMD:** Single Instruction, Multiple Data
- **MISD:** Multiple Instruction, Single Data
- **MIMD:** Multiple Instruction, Multiple Data



# Tipos de sistemas paralelos

## Clasificación de Flynn

- Single Instruction, Single Data (**SISD**): Una única secuencia de instrucciones y una única fuente de datos.
  - Implementación: Una única unidad de ejecución operando sobre datos en una única memoria.
  - Ejemplos: Arquitectura de procesador único (primeras computadoras, sistemas embebidos simples).





## Clasificación de Flynn

- Single Instruction, Multiple Data (SIMD): Una única secuencia de instrucciones y múltiples fuentes de datos.

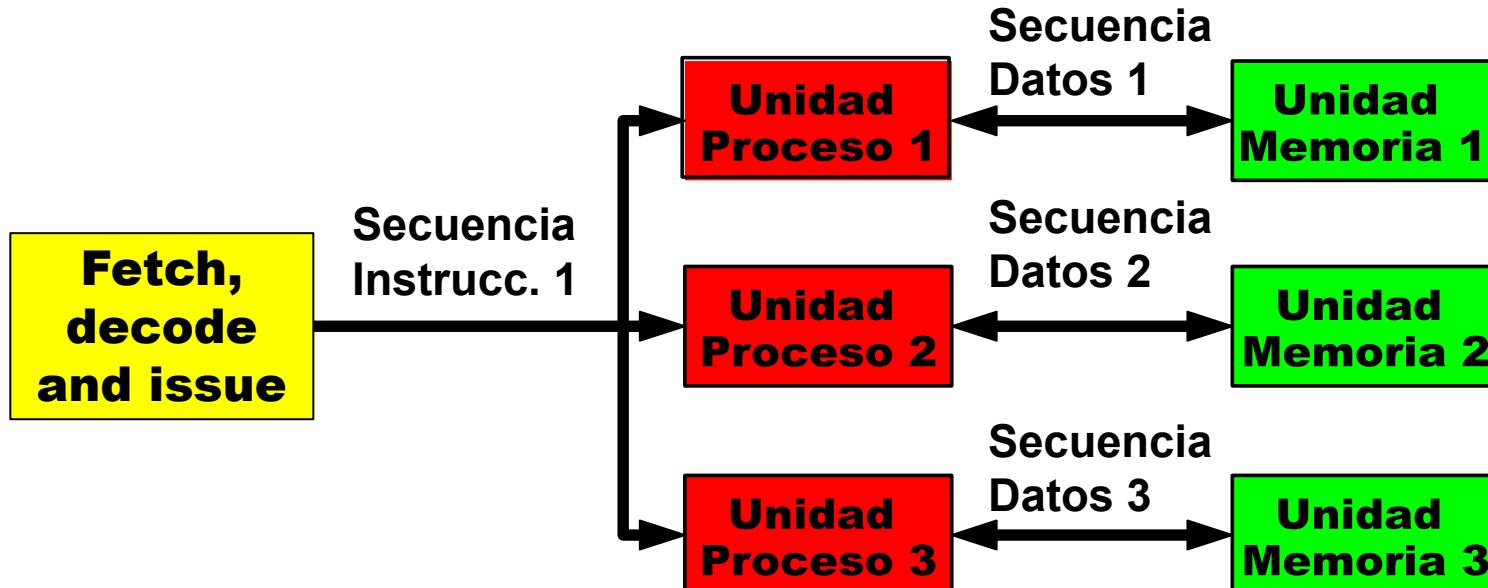
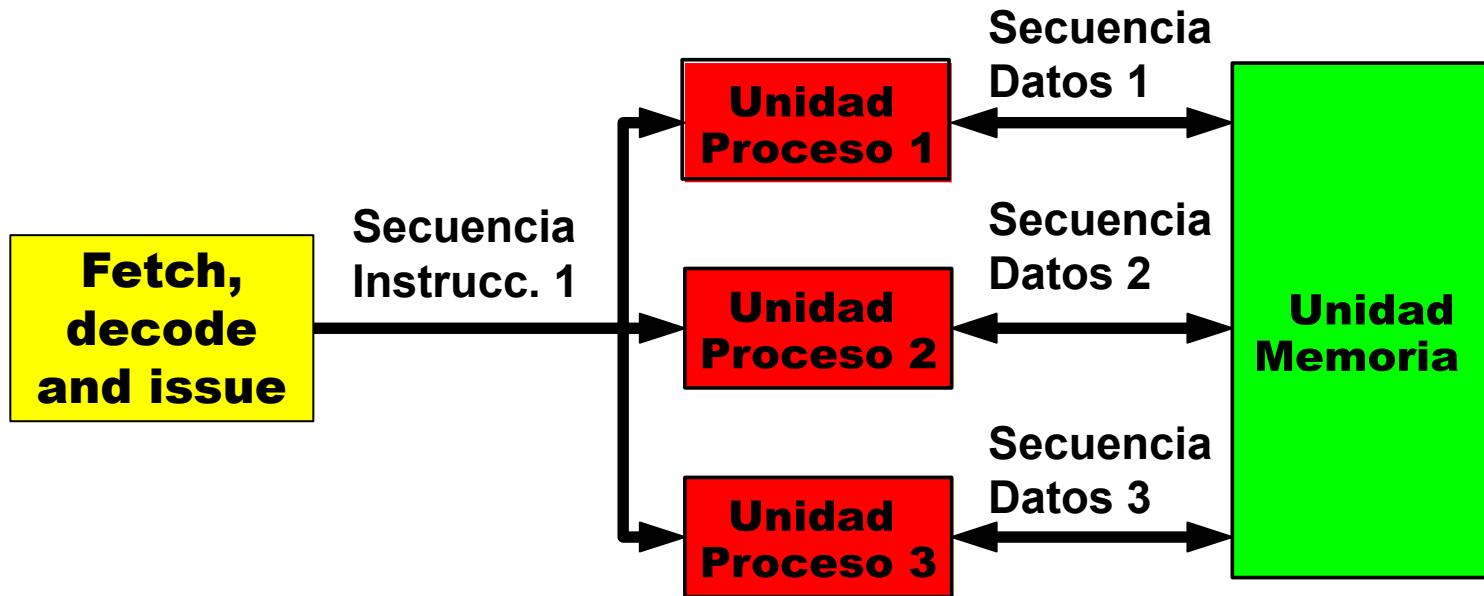


Figura basada en Computer Organization and Architecture 9th Edition William Stallings



## Clasificación de Flynn

- Single Instruction, Multiple Data (SIMD): Una única secuencia de instrucciones y múltiples fuentes de datos.





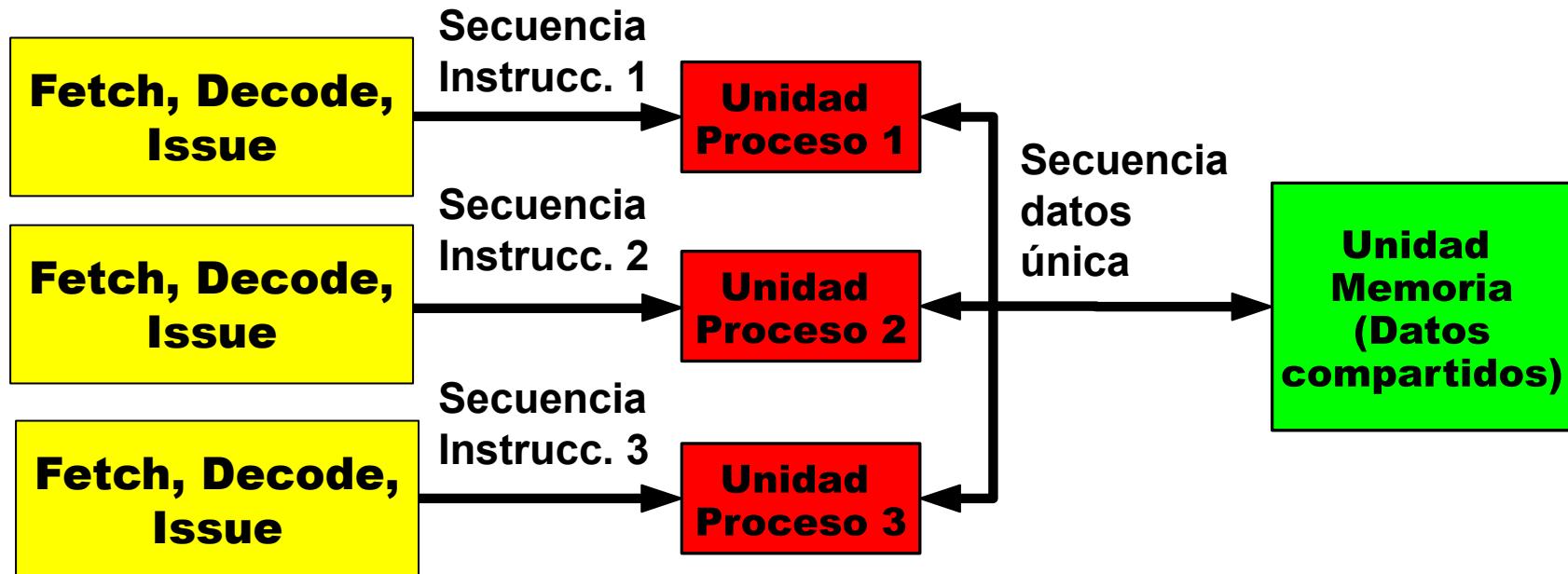
## Clasificación de Flynn

Single Instruction, Multiple Data (SIMD):

- Distintas implementaciones:
  - Una única etapa de búsqueda (fetch), decodificación y envío de instrucciones (issue) enviando esas instrucciones a varias unidades de ejecución (UE). Cada UE ejecuta la instrucción pero sobre distintos datos.
  - Varias ALUs que ejecutan la misma instrucción sobre distintos datos (ALUs superescalar).
  - Pipeline sobre la ALU: Una ALU ejecuta la instrucción sobre diferentes datos “casi” al mismo tiempo (pipeline de datos).
- Ejemplos:
  - Procesadores matriciales o vectoriales.
  - Extensiones MMX o SSE de x86
  - GPUs, GPGPUs

## Clasificación de Flynn

- Multiple instruction, single data (MISD): múltiples secuencias de instrucciones y una única fuente de datos.





## **Clasificación de Flynn**

- Multiple instruction, single data (MISD): múltiples secuencias de instrucciones y una única fuente de datos.
- Implementación: múltiples unidad de ejecución operando sobre datos en una única memoria
- Ejemplos Aplicación:
  - Computación altamente redundante (sistemas que requieren alta tolerancia a fallas)<sup>1</sup>.
  - Búsqueda de patrones<sup>2, 3</sup>.

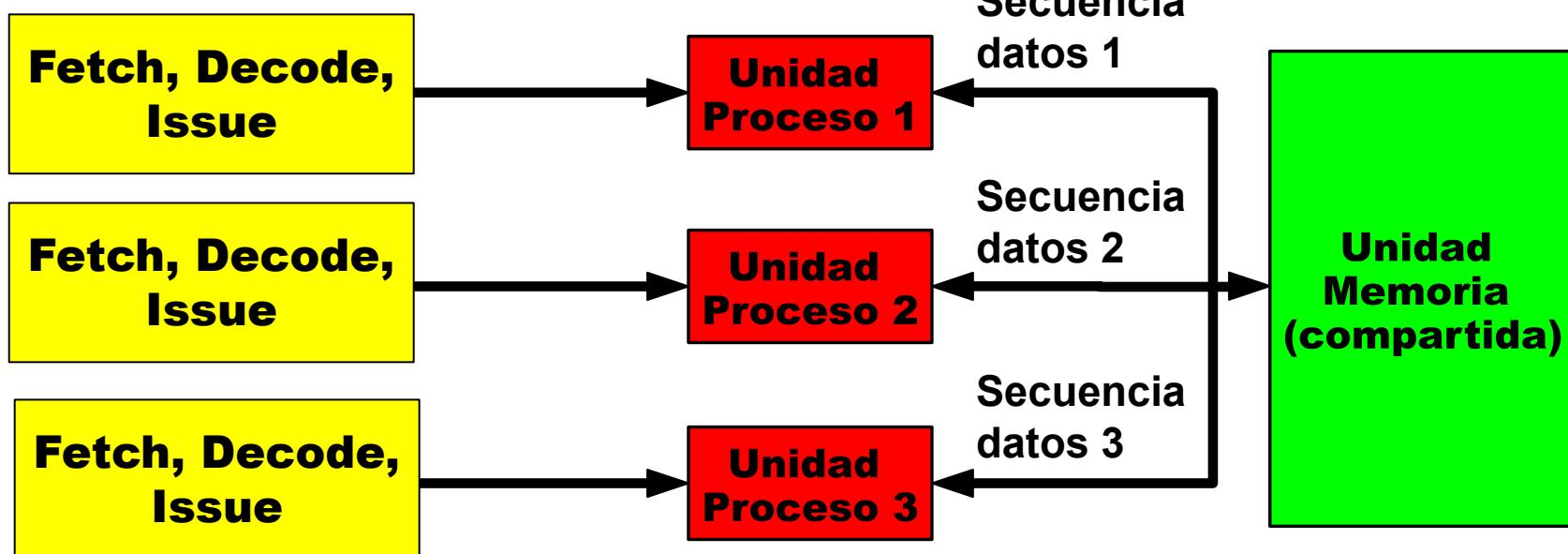
<sup>1</sup> Spector, A.; Gifford, D. (September 1984). "The space shuttle primary computer system". Communications of the ACM. 27 (9): 872–900. doi:10.1145/358234.358246

<sup>2</sup> Arne Halaas, Børge Svingen, Magnar Nedland, Pål Sætrom, Ola Snøve, Jr., and Olaf René Birkeland. 2004. A recursive MISD architecture for pattern matching. IEEE Trans. Very Large Scale Integr. Syst. 12, 7 (July 2004), 727-734.

<sup>3</sup> Algunos autores afirman que éstas arquitecturas no se ajustan a ninguna de los tipos de Flynn

## Clasificación de Flynn

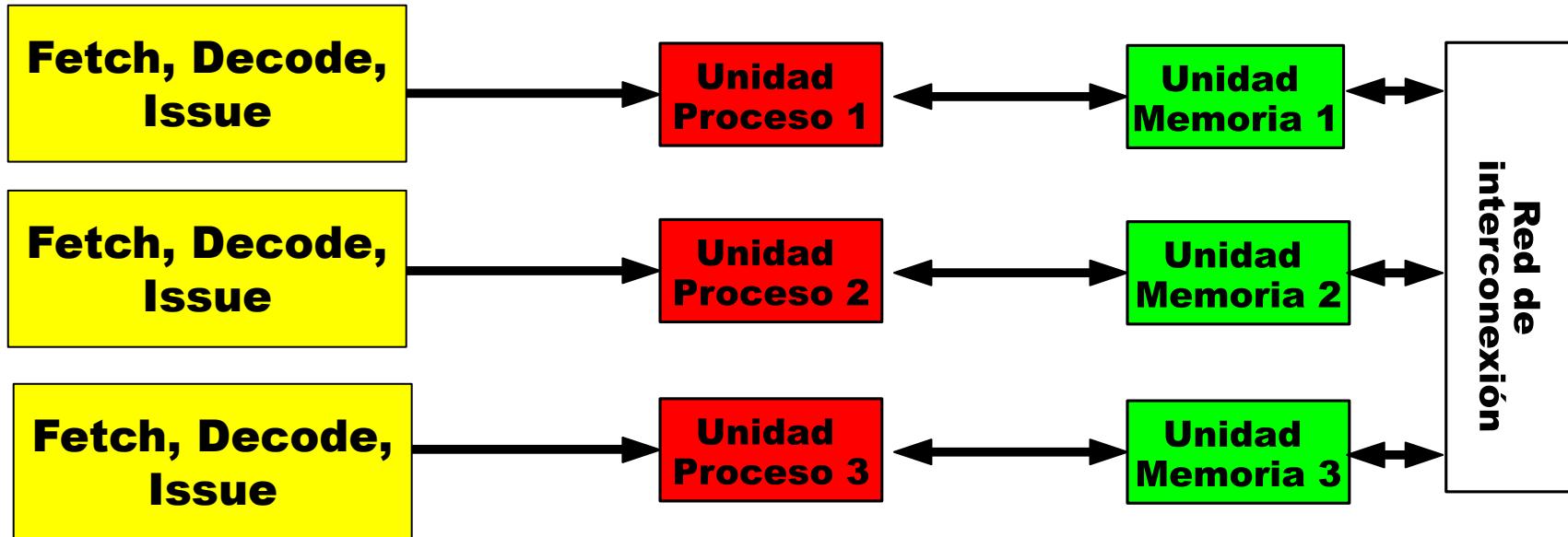
- Multiple instruction, multiple data (MIMD): múltiples secuencias de instrucciones y múltiples fuentes de datos.



Ejemplo: SMP, NUMA

## Clasificación de Flynn

- **Multiple instruction, multiple data (MIMD):**



Ejemplo: Cluster



## Tipos de sistemas paralelos Clasificación de Flynn

- Multiple instruction, multiple data (MIMD):
  - Implementaciones:
    - Memoria compartida: SMP, NUMA.
    - Cada procesador tiene su memoria (Memoria Distribuida): Cluster.

**Nota 1:** Un procesador superescalar **NO** es un ejemplo de MIMD, ya que, si bien puede ejecutar varias instrucciones al mismo tiempo, estas instrucciones forman parte de la misma secuencia de instrucciones. Puede ser SISD o SIMD (SIMD si soporta instrucciones vectoriales como MMX o SSE)

**Nota 2:** En los procesadores multithreading, todos los hilos que se ejecutan al mismo tiempo deben ser parte del mismo proceso, por lo que deben compartir recursos (como memoria caché, espacio de memoria, etc.). Por lo que puede suponerse que los hilos provienen del mismo “programa”. Por lo tanto, estos procesadores son del tipo **SISD o SIMD**.

## Processor organizations

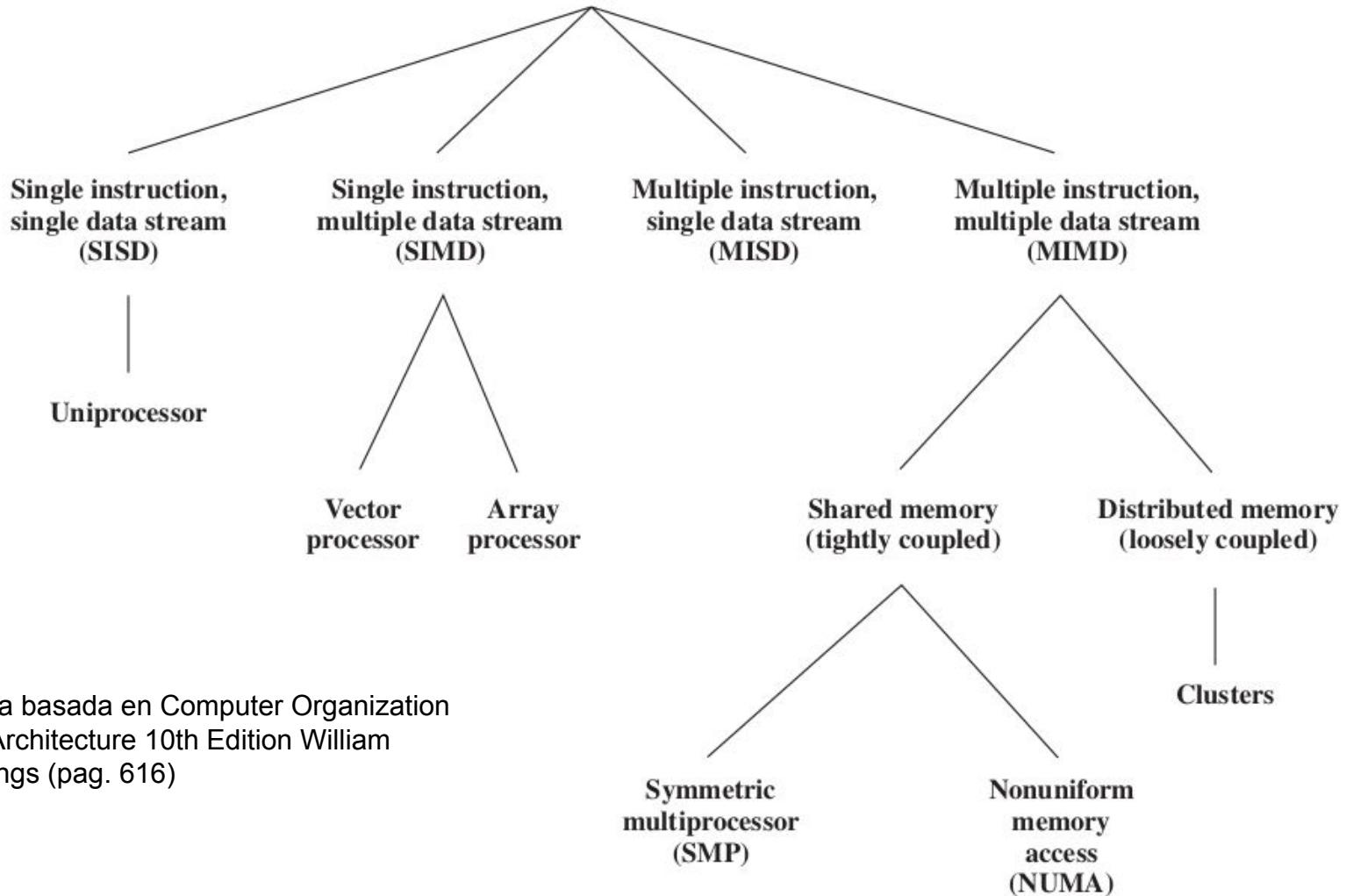


Figura basada en Computer Organization and Architecture 10th Edition William Stallings (pag. 616)



### Temas

Tipos de sistemas paralelos (Clasificación de Flynn)

→ **Paralelismo a nivel de instrucción**

Paralelismo a nivel de hilos (Procesadores multihilo simultáneo)

Paralelismo a nivel de núcleos

Arquitecturas multi núcleo simétricas

Arquitecturas multi núcleo heterogéneas con igual ISA o con ISA diferentes

Sistemas CPU-GPU, CPU-DSP. GPGPU. Introducción a CUDA y OpenCL

Sistemas operativos para multiprocesadores.

Paralelismo a nivel de procesos (Cluster)

Paralelismo a nivel de datos (SIMD)

Sistemas RAID

Performance y escalabilidad

Speedup y eficiencia

Modelos de problemas paralelizables

Paralelismo en el software



## Paralelismo a nivel de instrucción: Pipeline

	$t_1$	$t_2$	$t_3$	$t_4$	$t_5$	$t_6$	$t_7$	$t_8$	$t_9$	$t_{10}$
Instrucción 1	<b>F</b>	<b>D</b>	<b>E1</b>	<b>E2</b>	<b>W</b>					
Instrucción 2		<b>F</b>	<b>D</b>	<b>E1</b>	<b>E2</b>	<b>W</b>				
Instrucción 3			<b>F</b>	<b>D</b>	<b>E1</b>	<b>E2</b>	<b>W</b>			
Instrucción 4				<b>F</b>	<b>D</b>	<b>E1</b>	<b>E2</b>	<b>W</b>		
Instrucción 5					<b>F</b>	<b>D</b>	<b>E1</b>	<b>E2</b>	<b>W</b>	
Instrucción 6						<b>F</b>	<b>D</b>	<b>E1</b>	<b>E2</b>	<b>W</b>

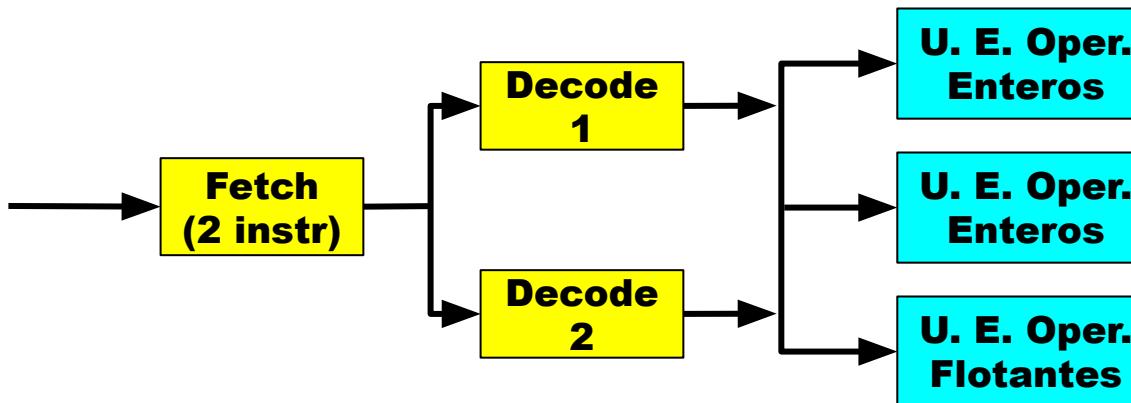


Speedup ideal = k (Para pipeline de k etapas)



## Paralelismo a nivel de instrucción: Superescalar

- Permite leer y ejecutar dos o más instrucciones al mismo tiempo
  - Etapa Fetch lee 2 o más instrucciones por cada acceso a memoria.
  - 2 o más decodificadores.
  - Algunas unidades de ejecución replicadas.

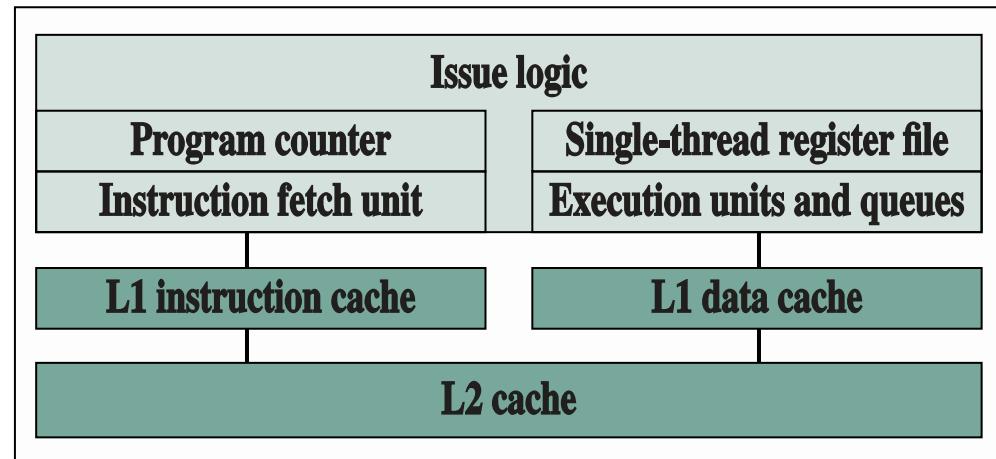


El primer procesador superescalar fue el Intel Pentium (1993), el cual tenía dos unidades de ejecución de enteros separados.



## Paralelismo a nivel de instrucción: Superescalar

<b>F</b>	<b>D</b>	<b>E1</b>	<b>E2</b>	<b>W</b>		
<b>F</b>	<b>D</b>	<b>E1</b>	<b>E2</b>	<b>W</b>		
	<b>F</b>	<b>D</b>	<b>E1</b>	<b>E2</b>	<b>W</b>	
	<b>F</b>	<b>D</b>	<b>E1</b>	<b>E2</b>	<b>W</b>	
		<b>F</b>	<b>D</b>	<b>E1</b>	<b>E2</b>	<b>W</b>
		<b>F</b>	<b>D</b>	<b>E1</b>	<b>E2</b>	<b>W</b>



$$\text{Speedup ideal} = N * k$$

Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 658



## Paralelismo a nivel de instrucción

Problemas: Dependencias de datos y conflictos de recursos. Disminuyen el speedup.

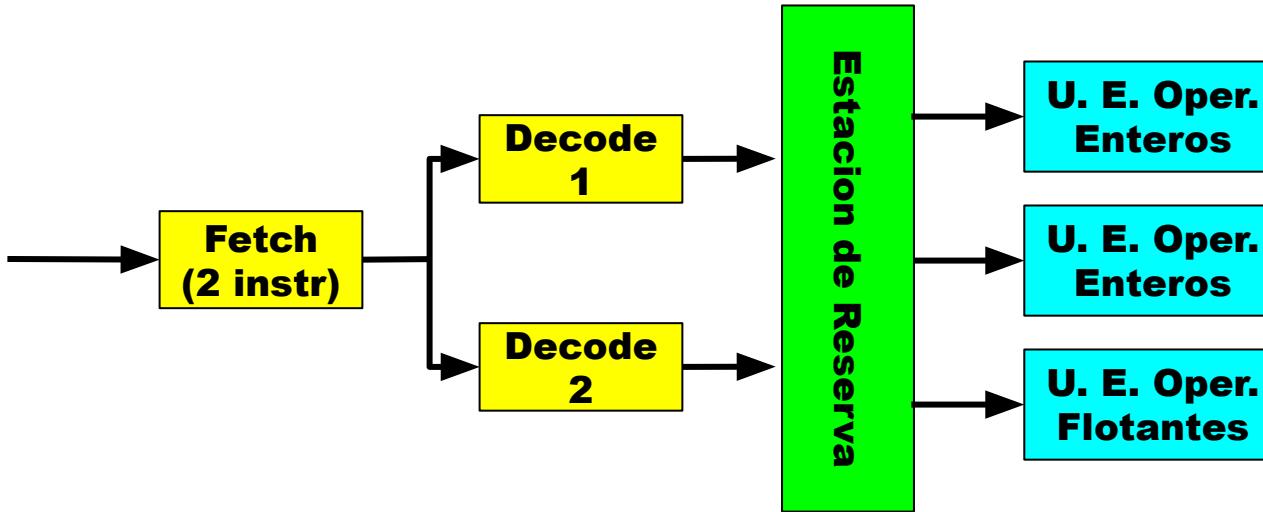
F1	D1	MUL1	MUL2	MUL3	W1								
F2	D2	D2	MUL1	MUL2	MUL3								
	F1	F1	D1	D1	D1	ADD1	ADD2						
	F2	F2	D2	D2	D2	ADD1	ADD2						
			F1	F1	F1	D1	ADD1	ADD2					
			F2	F2	F2	D2	LD1	LD2					

Conflictos de recursos

Dependencia de datos



## Paralelismo a nivel de instrucción: Superescalar fuera de orden

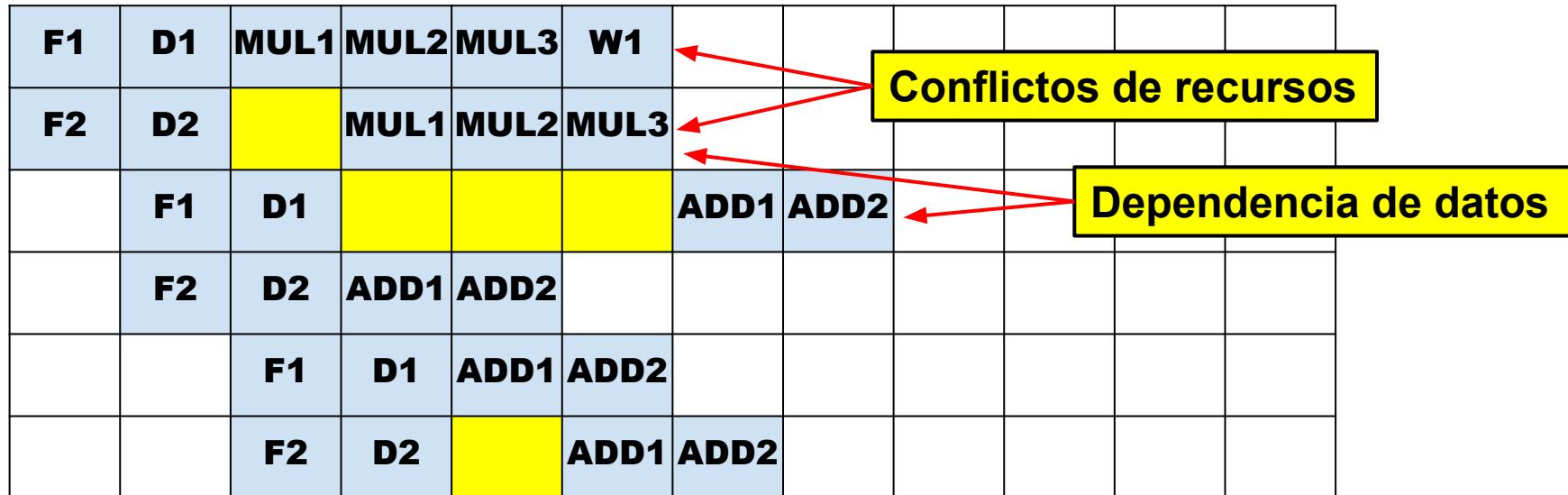


**Speedup ideal = N\*k** (Igual que el superescalar)

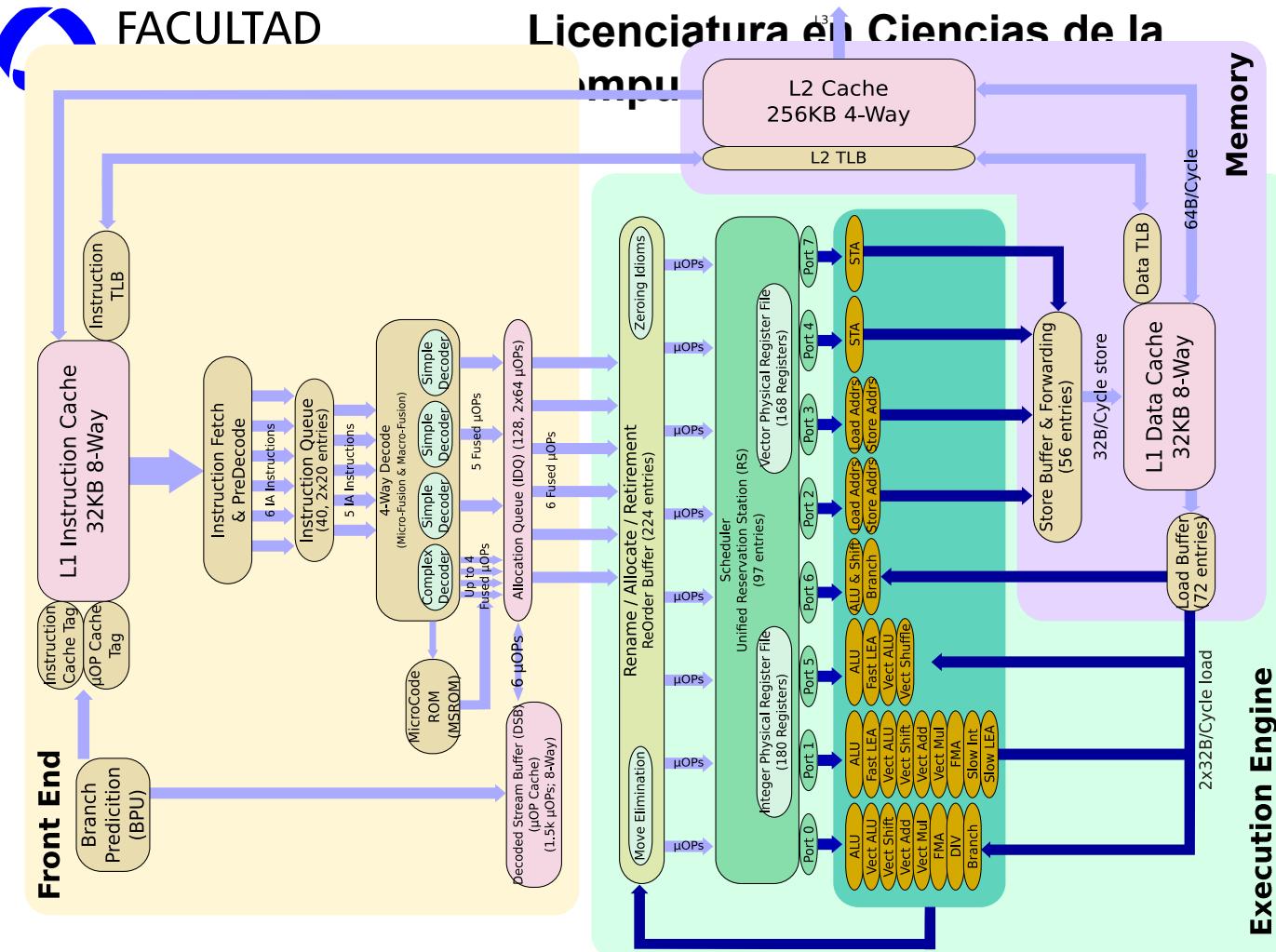
Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 658

# Paralelismo a nivel de instrucción

El procesador fuera de orden acerca el speedup real al ideal para procesadores superescalares.



## Ejemplo 1 de Arquitectura que utiliza ejecución fuera de orden: Intel Kaby Lake

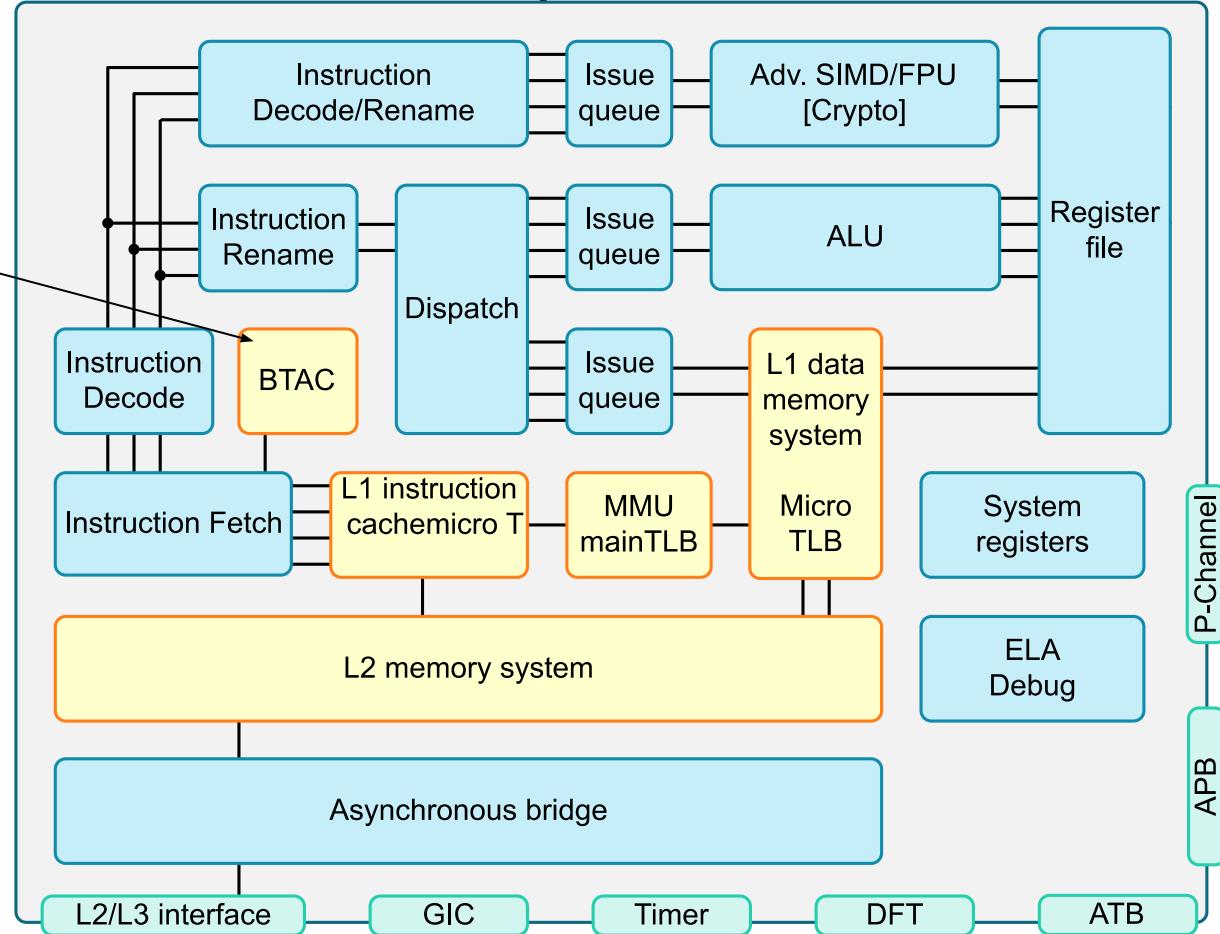




## Ejemplo 2: ARM Cortex A75:

Branch Target  
Address Cache

**BTAC: Branch Target Address Cache**  
**ELA: Embedded Logic Analysis**





## Paralelismo a nivel de instrucciones

Problemas procesador superescalar fuera de orden:

- Aparecen dos tipos de dependencias de datos:
  - Dependencias **WAR** o **antidependencia**
    - Al cambiar el orden de ejecución, un procesador fuera de orden puede generar una dependencia de datos entre instrucciones que no poseían problemas de dependencia.
  - Dependencia **WAW** o **dependencia de salida**.
    - Al escribir datos en un recurso (registro, memoria, I/O), en orden diferente al establecido por el programador, el resultado de la ejecución puede ser diferente al esperado.
- **Las dependencias WAR y WAW se solucionan mediante la técnica de renombrado de registros.**
  - **Requiere hardware adicional que incrementa el consumo de energía, pero mejora el speedup.**



## Paralelismo a nivel de instrucciones

**Problemas** procesador superescalar fuera de orden:

- Amortigua la reducción del speedup por dependencia de datos, pero no lo soluciona por completo. Sigue **siendo un problema**.
- Reducción del speedup debido al tiempo de acceso a memoria.
  - Procesador fuera de orden lo amortigua muy poco.
  - Solución: memoria caché + procesador multi-hilo simultáneo.

F1	D1	ADD1	ADD2						
F2	D2	LD (Acceso a memoria)							
	F1	D1					ADD1	ADD2	
	F2	D2	MUL1	MUL2	MUL3				
		F1	D1	ADD1	ADD2				
		F2	D2			ADD1	ADD2		

Acceso a memoria y  
dependencia de datos.

Dependencia de datos en  
vías diferentes.



### Temas

Tipos de sistemas paralelos (Clasificación de Flynn)

Paralelismo a nivel de instrucción

#### → **Paralelismo a nivel de hilos (Procesadores multihilo simultáneo)**

Paralelismo a nivel de núcleos

- Arquitecturas multi núcleo simétricas

- Arquitecturas multi núcleo heterogéneas con igual ISA o con ISA diferentes

- Sistemas CPU-GPU, CPU-DSP. GPGPU. Introducción a CUDA y OpenCL

- Sistemas operativos para multiprocesadores.

Paralelismo a nivel de procesos (Cluster)

Paralelismo a nivel de datos (SIMD)

Sistemas RAID

Performance y escalabilidad

- Speedup y eficiencia

- Modelos de problemas paralelizables

- Paralelismo en el software



## Multithreading (Multi hilo)

### Ventaja del uso de hilos

- Facilidad para escribir un programa, ya que:
  - El problema puede dividirse en sub-problemas, resolviendo cada sub-problema con diferentes hilos.
  - Pueden usarse llamadas al sistema bloqueantes.
  - Ejemplos de aplicaciones que son más eficientes gracias a los hilos: aplicaciones cliente-servidor, sistemas distribuidos, interfaces de usuario, aplicaciones productor-consumidor, etc.
  - **Permiten crear aplicaciones paralelas con memoria compartida.**

- **Hacer uso más eficiente de arquitecturas que permiten la ejecución de instrucciones en paralelo.**



**Esta es la ventaja que nos interesa en esta materia**



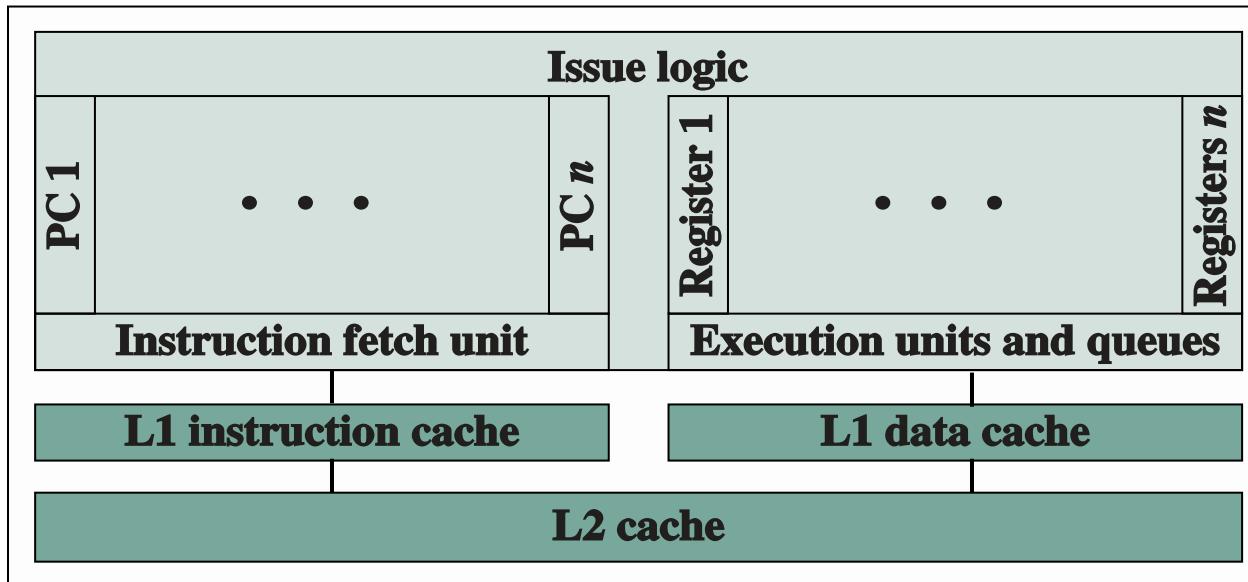
## Multithreading simultaneo sobre procesador de un núcleo

- Procesador superescalar que toma varias instrucciones de **diferentes hilos**, para incrementar la independencia entre instrucciones a ejecutar al mismo tiempo.
- **Incrementa más el speedup**. Mejor performance para tratar con dependencias de datos y accesos a memoria.

<b>F1</b>	<b>D1</b>	<b>ADD1</b>	<b>ADD2</b>					
<b>F2</b>	<b>D2</b>	<b>LD (Acceso a memoria)</b>						
	<b>F1</b>	<b>D1</b>	<b>MUL1</b>	<b>MUL2</b>	<b>MUL3</b>			
	<b>F2</b>	<b>D2</b>	<b>ADD1</b>	<b>ADD2</b>				
		<b>F1</b>	<b>D1</b>	<b>ADD1</b>	<b>ADD2</b>			
		<b>F1</b>	<b>D1</b>	<b>MUL1</b>	<b>MUL2</b>	<b>MUL3</b>		



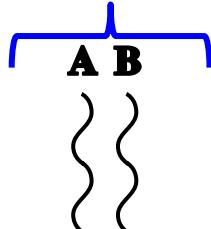
## Multithreading simultaneo sobre procesador de un núcleo





**La microarquitectura Kaby Lake (i3, i5 e i7 año 2017) de Intel responde a este diseño (Hyperthreading)**

**El SO ve 2 núcleos lógicos**



1	A	A	A	B	B	B		
2	A	A	A	A	A	A	B	
3	A	A	A	B	B	B	B	B
4	A	A	A	A	A	A	B	B
5	A	A	B	B	B	B		
6	A	B	B	B	B	B	B	

**Secuencia de entrada a unidades de ejecución**

## Licenciatura en Ciencias de la Computación

**Pipeline**

**Instrucciones hilo A**

**Instrucciones hilo B**

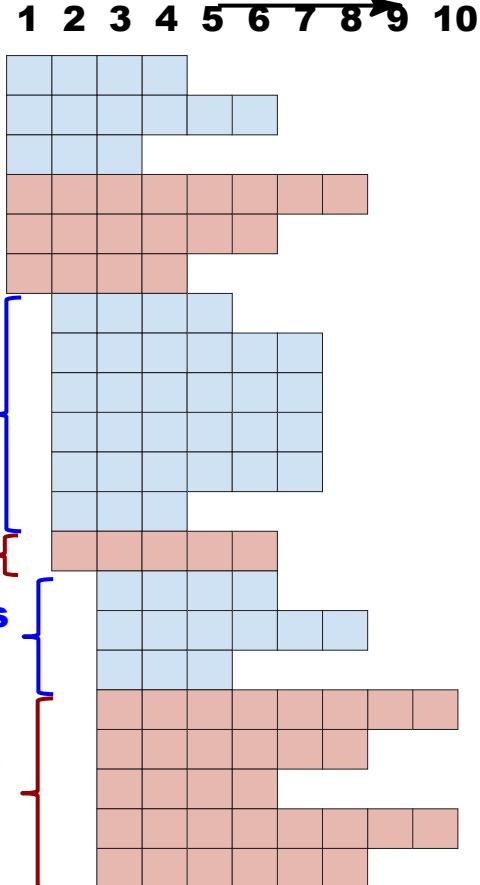
**Instrucciones hilo A**

**Instrucción hilo B**

**Instrucciones hilo A**

**Instrucciones hilo B**

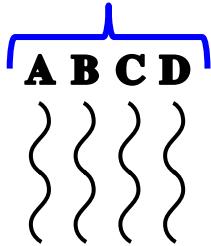
**Ciclos**





**La microarquitectura del ARM Cortex A75 (año 2017) responde a este diseño**

El SO ve 4 núcleos lógicos



Ciclos	1	2	3	4	5	6
1	A A	B B	C	D D		
2	A	B B		D D		
3		B		D		
4	A A		C	D D		
5		B B	C C	D D		
6	A A		C	D		

**Secuencia de entrada a unidades de ejecución**

Instrucciones  
hilo A (núcleo  
1)

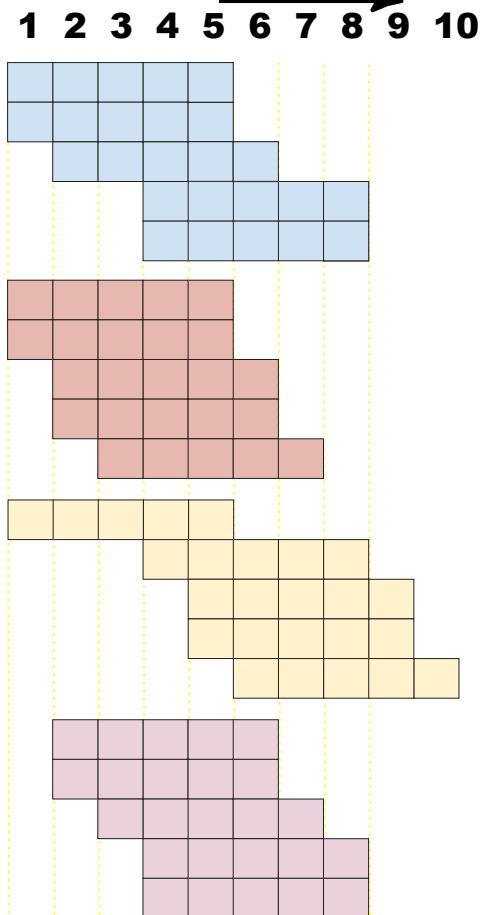
Instrucciones  
hilo B (núcleo  
2)

Instrucciones  
hilo C (núcleo  
3)

Instrucciones  
hilo D (núcleo  
4)

Licenciatura en Ciencias de la  
Computación

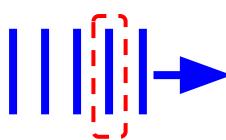
Ciclos



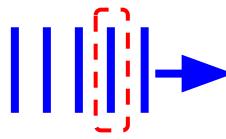


## **Multithreading (Multi hilo)**

Ejemplo de funcionamiento con procesador superescalar de dos vías, con problema de dependencia de datos RAW



F	D	E	E	E	W				
F	D				E	W			
	F	D				E	W		
	F	D				E	W		



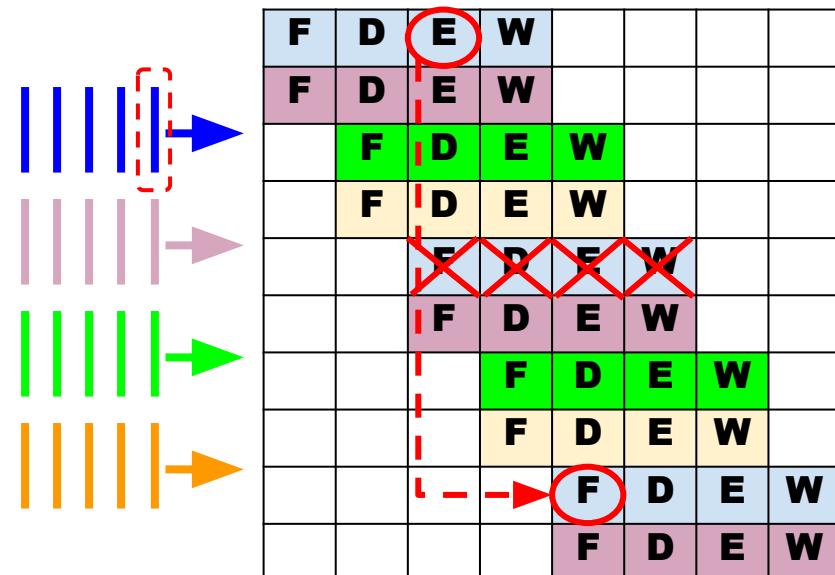
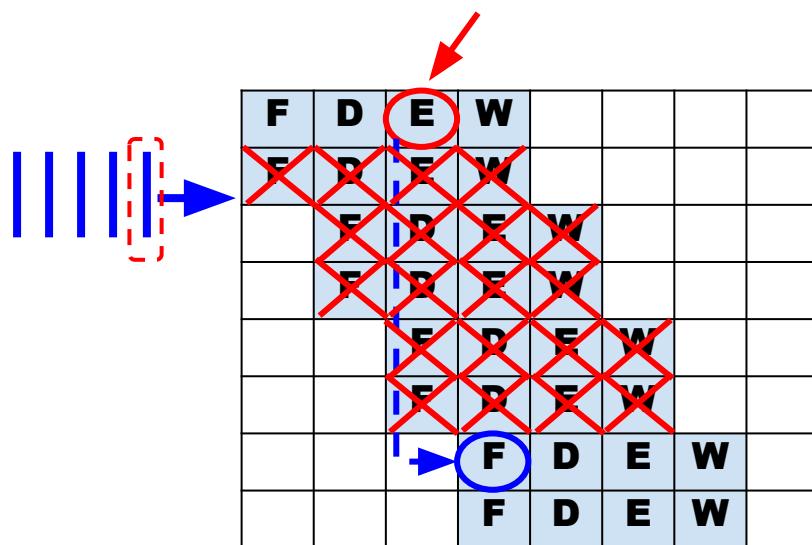
F	D	E	E	E	W				
F	D	E	W						
	F	D	E	W					
	F	D	E	W					

**Si se combina con ejecución fuera de orden, podría no perderse ningún ciclo, ya que en lugar de ejecutar la segunda instrucción del primer hilo (que tiene problemas de dependencia), podría ejecutarse la tercera antes de la segunda.**

# Multithreading (Multi hilo)

Ejemplo de funcionamiento con procesador superescalar de 2 vías, con problema de conflicto estructural (salto condicional)

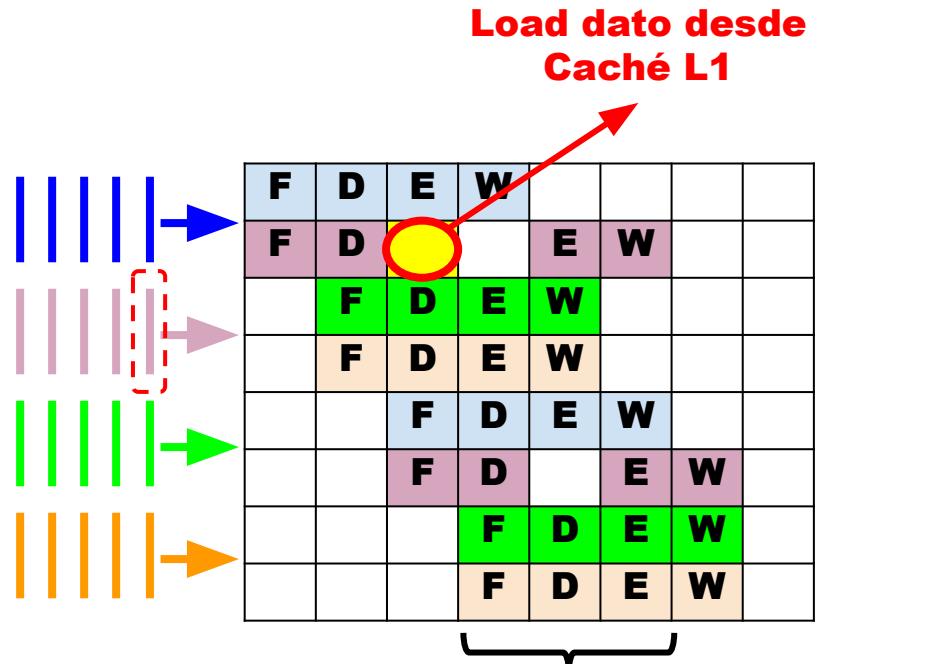
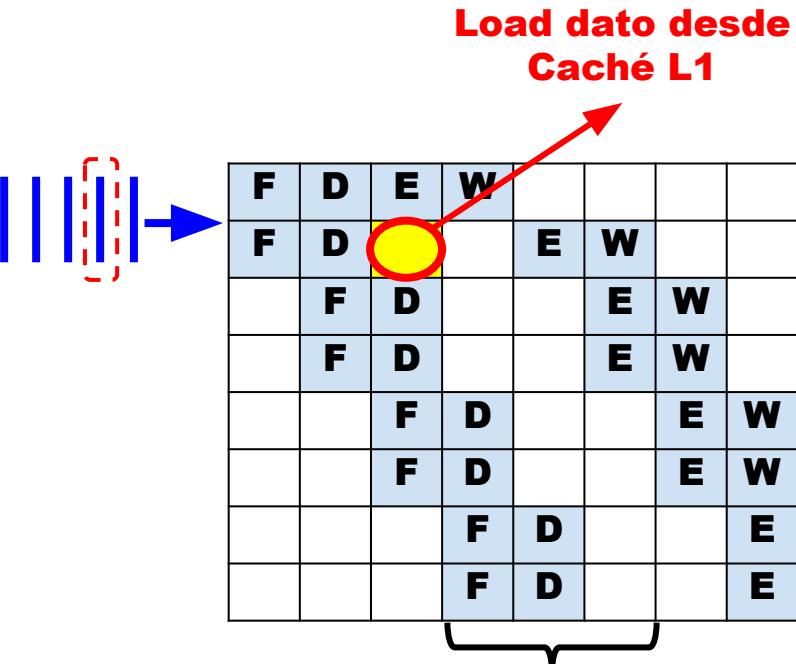
## Instrucción de salto condicional





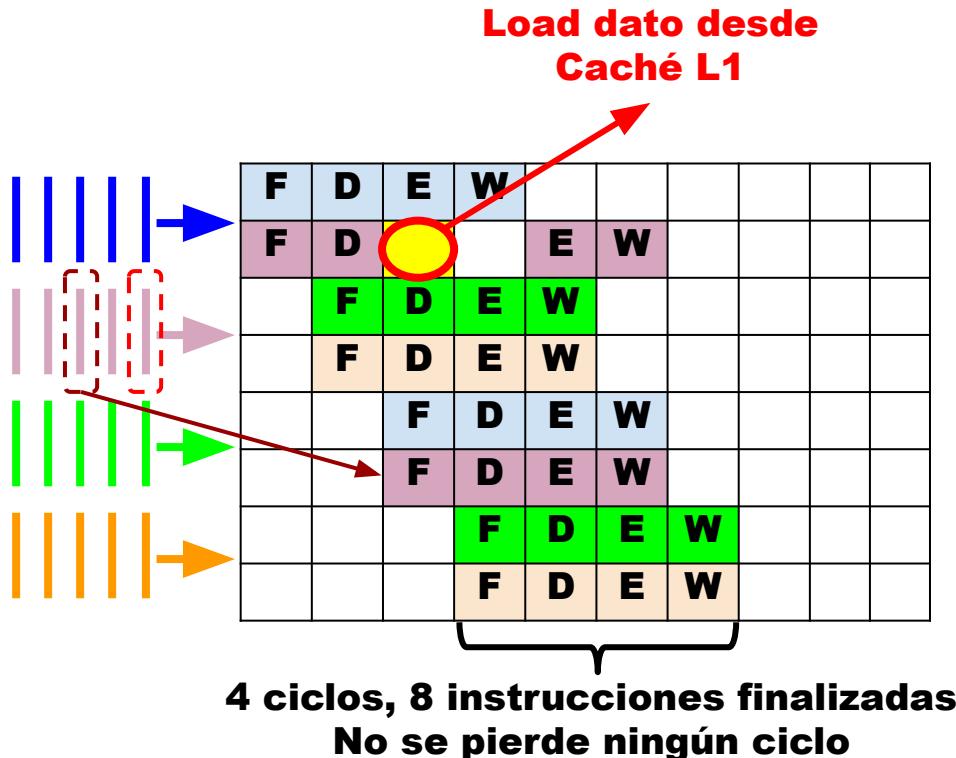
## **Multithreading (Multi hilo)**

Ejemplo de funcionamiento con procesador superescalar de dos vías, acceso a memoria Caché L1 y 4 hilos





Ejemplo de funcionamiento con procesador superescalar de dos vías, acceso a memoria Caché L1, 4 hilos y ejecución fuera de orden.



**Notas:**

- 1) En el ciclo 5 tenemos 3 instrucciones en ejecución. Esto es posible si las instrucciones utilizan unidades de ejecución diferentes.
- 2) En el ciclo 6 hay tres instrucciones escribiendo resultados en memoria, esto es posible si al menos una no escribe en memoria (es decir, uno de los resultados queda en registros).
- 3) En un procesador fuera de orden, si la instrucción 2 del hilo 2 depende de la instrucción 1, el procesador podría ejecutar la instrucción 3 antes de la 2.



## Paralelismo a nivel de hilos: Librería thread de C++

- Permite crear y administrar hilos. Muy similar a threading de Python
  - Es necesario agregar: `#include <thread>`
  - Java: clase **java.lang.Thread**
- Los hilos deben escribirse como funciones.
- Crear hilo:

```
void hilo1(){  
    Declaración de la función  
}
```

} Función que implementa el hilo

- Crear y comenzar ejecución del hilo:

```
std::thread hilo1(function1);
```



## Paralelismo a nivel de hilos: Librería `thread` de C++

- Implementación típica en programas paralelos: lista o arreglo de hilos.
- Esperar a que un hilo finalice para continuar con el programa:  
*mi\_hilo.join()*
  - El programa principal o hilo padre se bloqueará en dicha instrucción hasta que el hilo *mi\_hilo* termine su ejecución.



### **Temas**

Tipos de sistemas paralelos (Clasificación de Flynn)

Paralelismo a nivel de instrucción

Paralelismo a nivel de hilos (Procesadores multihilo simultáneo)



#### **Paralelismo a nivel de núcleos**

Arquitecturas multi núcleo simétricas

Arquitecturas multi núcleo heterogéneas con igual ISA o con ISA diferentes

Sistemas CPU-GPU, CPU-DSP. GPGPU. Introducción a CUDA y OpenCL

Sistemas operativos para multiprocesadores.

Paralelismo a nivel de procesos (Cluster)

Paralelismo a nivel de datos (SIMD)

Sistemas RAID

Performance y escalabilidad

Speedup y eficiencia

Modelos de problemas paralelizables

Paralelismo en el software



## **Problemas del paralelismo a nivel de instrucción o hilos en un solo núcleo**

- Pipeline: complejidad de los circuitos lógicos, interconexión y señales.
- Superescalar: complejidad de los circuitos para manejar dependencia de datos, conflicto de recursos y saltos.
- Multithreading simultáneo sobre un núcleo: El límite en la cantidad de instrucciones que pueden ejecutarse en paralelo sobre un núcleo depende de la cantidad de vías del procesador superescalar.

Incrementar el paralelismo a nivel de instrucciones requiere más transistores.

**Regla de Pollack:** aumentar N veces la complejidad de un núcleo se traduce en un aumento del  $(N)^{1/2}$  en performance.

En términos de cantidad de transistores en el procesador:

- Performance  $\propto$  (área) $^{1/2}$
- Consumo de energía  $\propto$  área
- Duplicar el área incrementa P en un 40%, pero duplica el consumo de energía.



## Arquitecturas multinúcleo

Fundamento: En el caso ideal, si usamos  $N$  núcleos, la performance aumentará  $N$  veces (suponiendo programa con 100% de código paralelizable).

- Dos o más procesadores (llamados núcleos) en un mismo chip **compartiendo la memoria principal**.
- Cada núcleo posee todos los elementos típicos de un procesador (ALU, hardware superescalar y pipeline, registros, sistemas multihilo, etc).
- Pueden (o no) compartir:
  - Diferentes niveles de caché (la L1 usualmente no es compartida).
  - Sectores de memoria principal.
  - Algunos periféricos.
- Pueden (o no) ser fuera de orden o multithreading.



## Arquitecturas multinúcleo

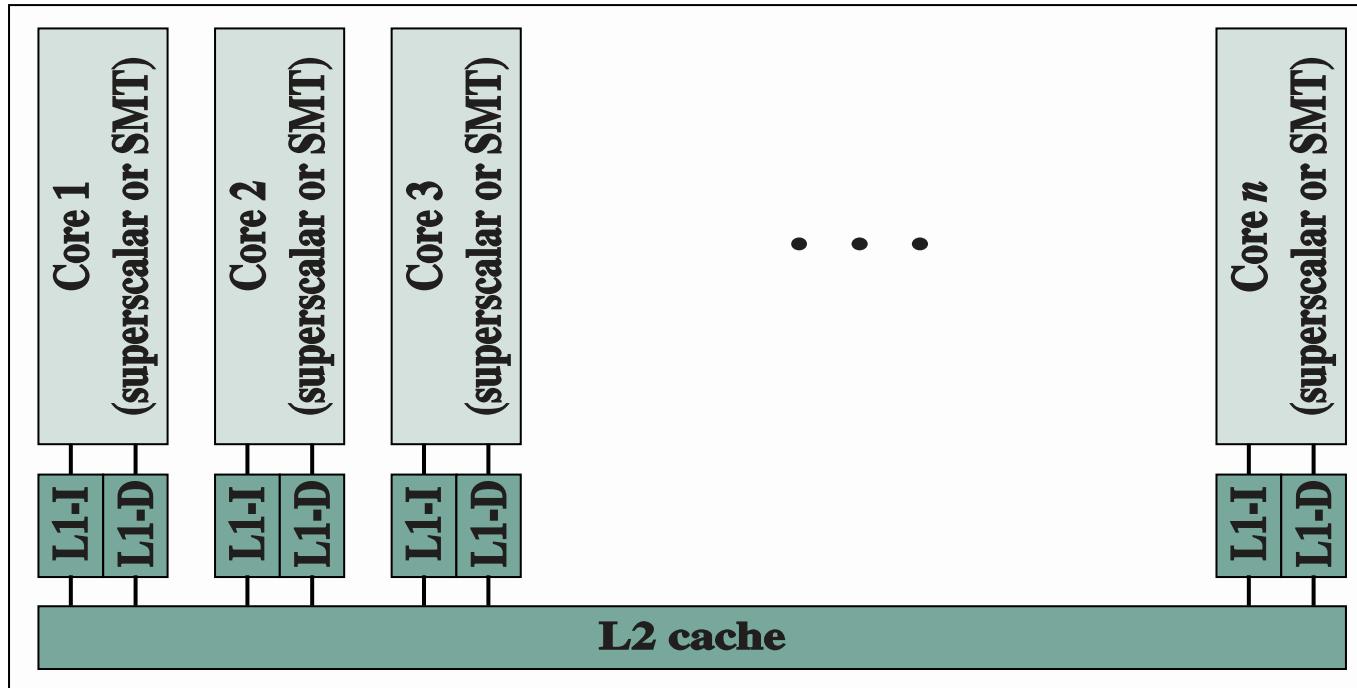


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 658

## Sistemas multicore - Memoria caché

Importancia de la memoria caché

- Si los procesadores comparten el bus, solo uno podrá acceder mientras que los demás esperan.
  - **Mientras más procesadores, cada uno estará mayor cantidad de tiempo esperando poder acceder al bus.**

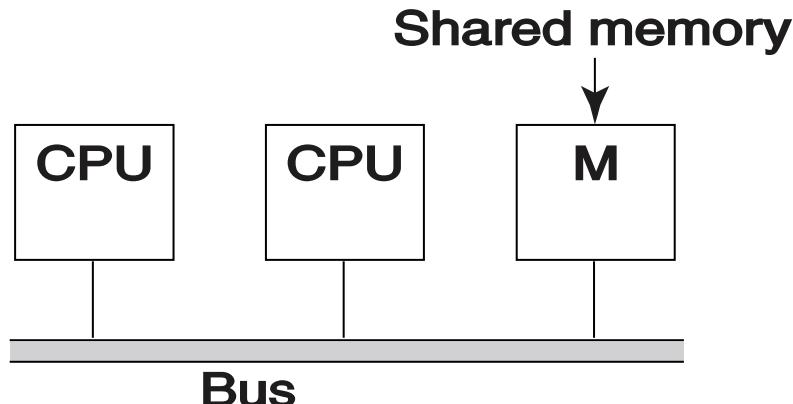


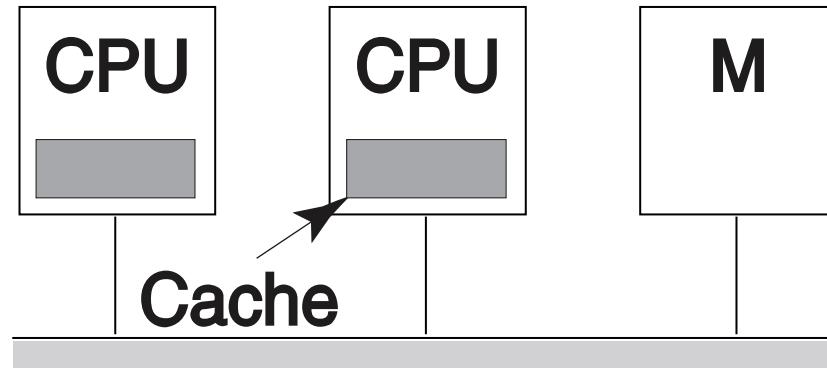
Figura obtenida de: Tanenbaum, Bos, "Modern Operating Systems", 4° edición, página 521.



## Sistemas multicore - Memoria caché

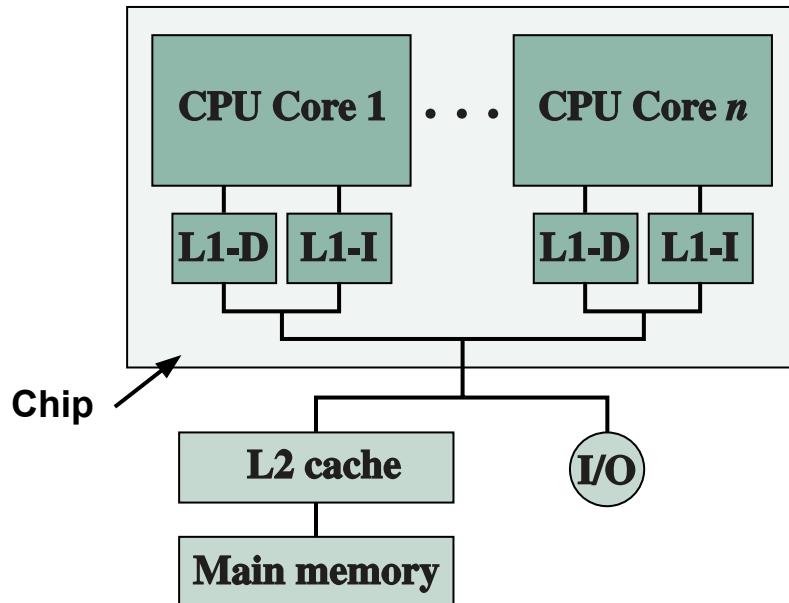
Importancia de la memoria caché

- **Solución: Memoria caché. Varios procesadores pueden trabajar sobre sus cachés al mismo tiempo.**
- Memorias caché grandes disminuyen los accesos a memoria RAM.

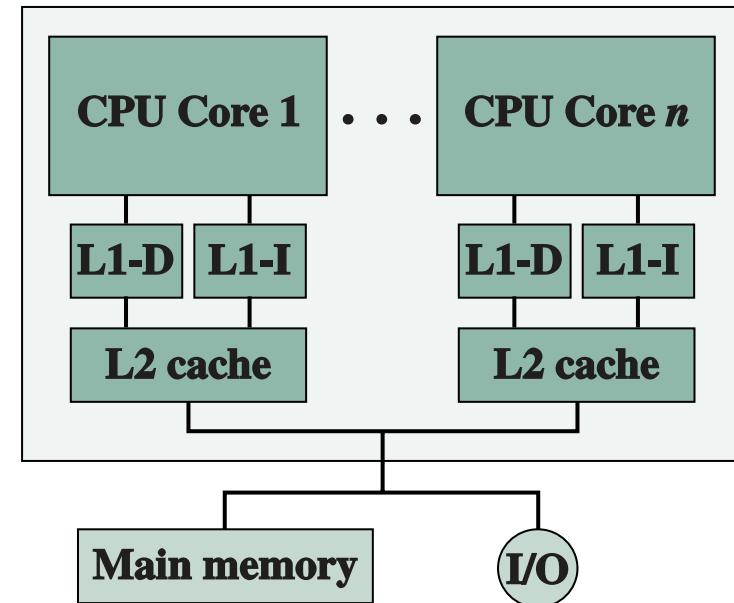




## Sistemas multicore - diferentes arquitecturas de memoria caché



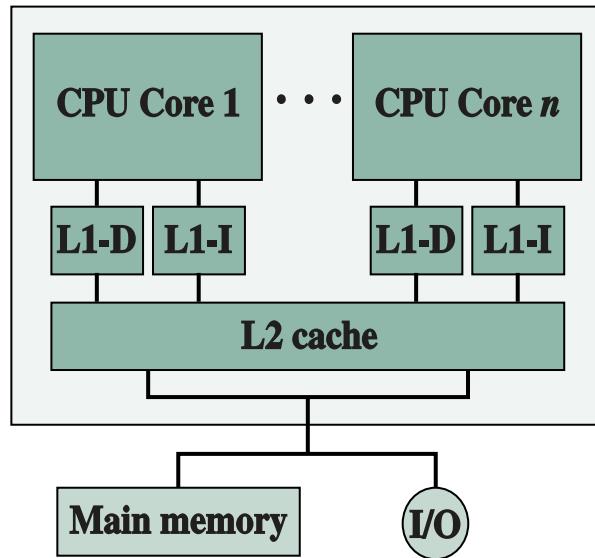
L1 dedicada on chip, L2 compartida  
(primeros procesadores. ARM11)



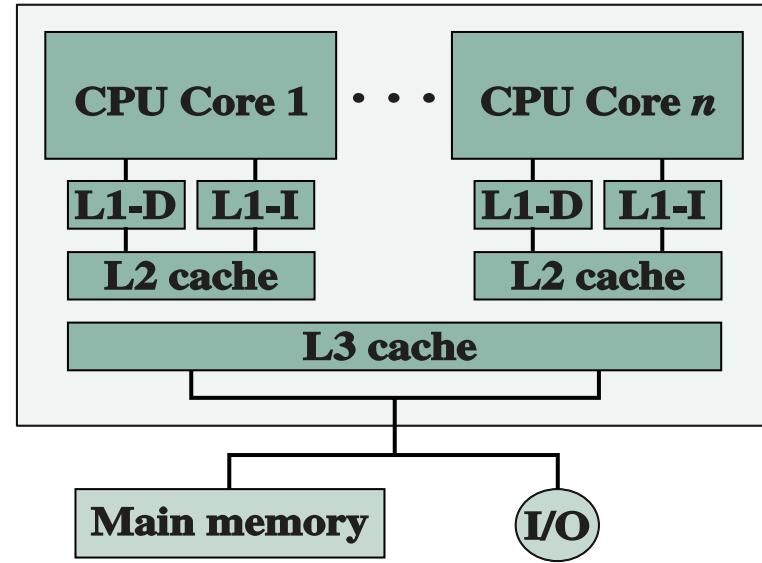
L1 y L2 dedicadas on chip (AMD  
Opteron)



## Sistemas multicore - diferentes arquitecturas de memoria caché



L1 dedicada, L2 compartida on chip  
(Intel Core Duo)



L1 y L2 dedicada, L3 compartida on chip  
(Intel Core i7)



## Sistemas multicore - diferentes arquitecturas de memoria caché

- Necesidad: incremento de la diferencia de velocidad entre el procesador y la memoria principal hace necesario más niveles de caché.

### Beneficios de usar memoria caché compartida dentro del chip.

- Disminuir los accesos a datos que no están en caché (miss). Un procesador trae datos a caché que otro procesador puede usar.
- Comunicación entre núcleos es más rápida. (Sistemas multihilos).
- Confina los problemas de coherencia de memoria caché entre procesadores a cachés dedicadas (los datos en caché compartida no sufren problemas de coherencia entre procesadores). (Sistemas multihilos)



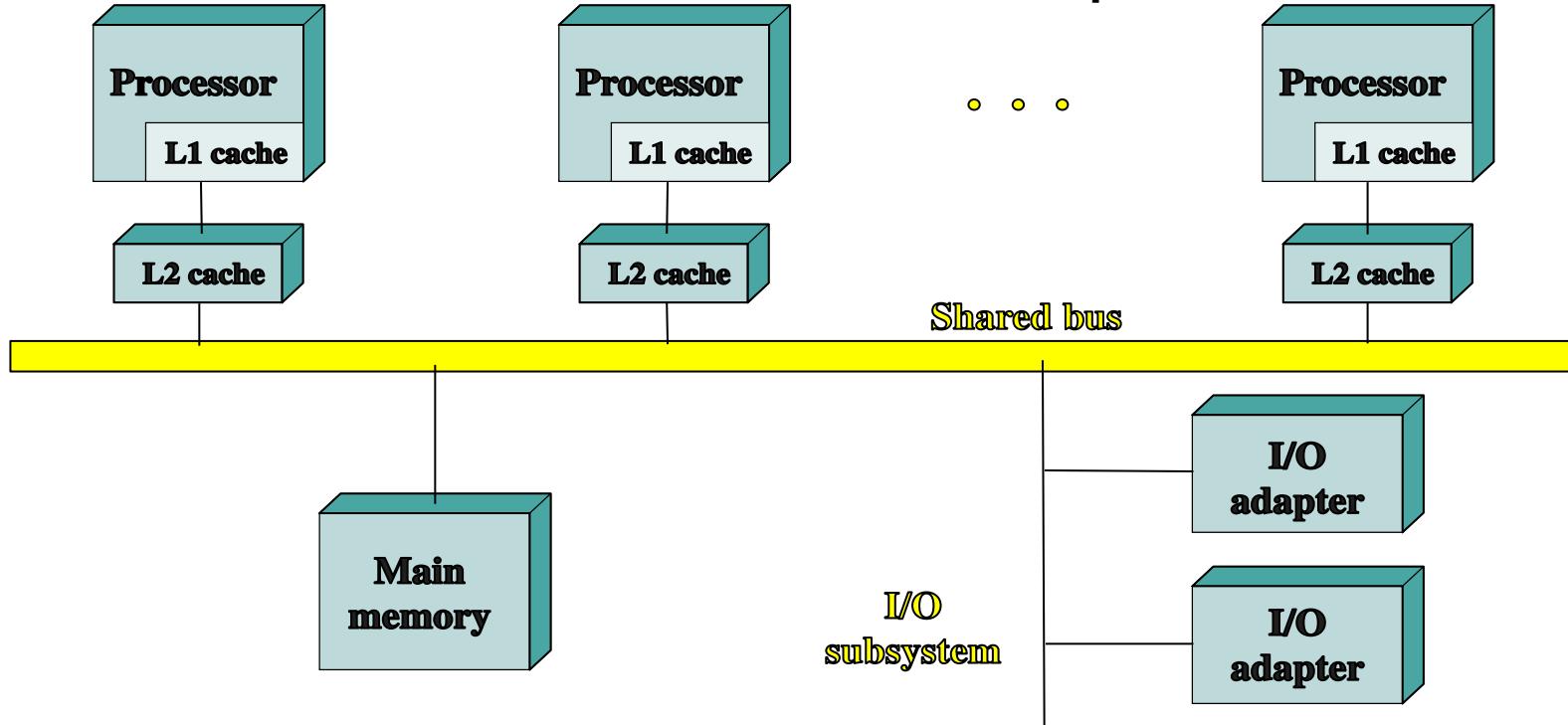
## Sistemas multi-núcleo - Diferentes arquitecturas

- Chip **multi-núcleo simétrico** (visto en Arquitectura de computadoras).
- Chip **multi-núcleo heterogéneo**: Más de un núcleo de diferentes tipos.
  - Multicore heterogéneo con **igual set de instrucciones** (visto en Arquitectura de computadoras).
    - Ejemplo: arquitectura big.Little de ARM.
  - Multicore heterogéneo con **distinto set de instrucciones**.
    - CPU/GPU.
    - CPU/DSP (Digital Signal Processors)



## SMP

- **Los procesadores se conectan a un bus**
- **Los procesadores pueden acceder a bloques separados de memoria (en algunas arquitecturas simultáneamente).**
- **Cada procesador posee su**
  - **ALU**
  - **Registros**
  - **Unidades de ejecución (si es superescalar).**
  - **Uno o más niveles de memoria Caché.**



**SMP - bus compartido en  
tiempo**

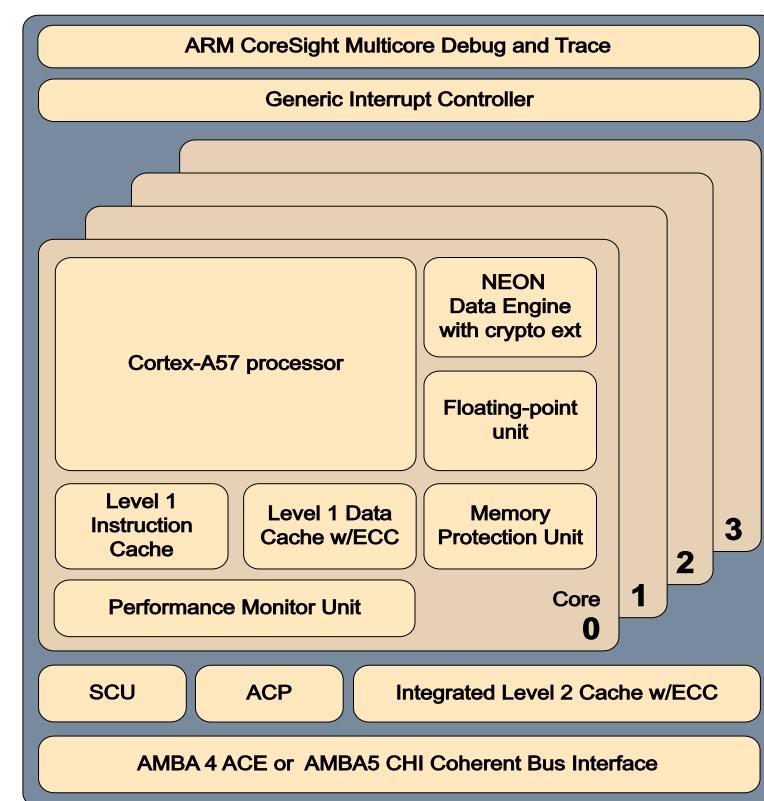
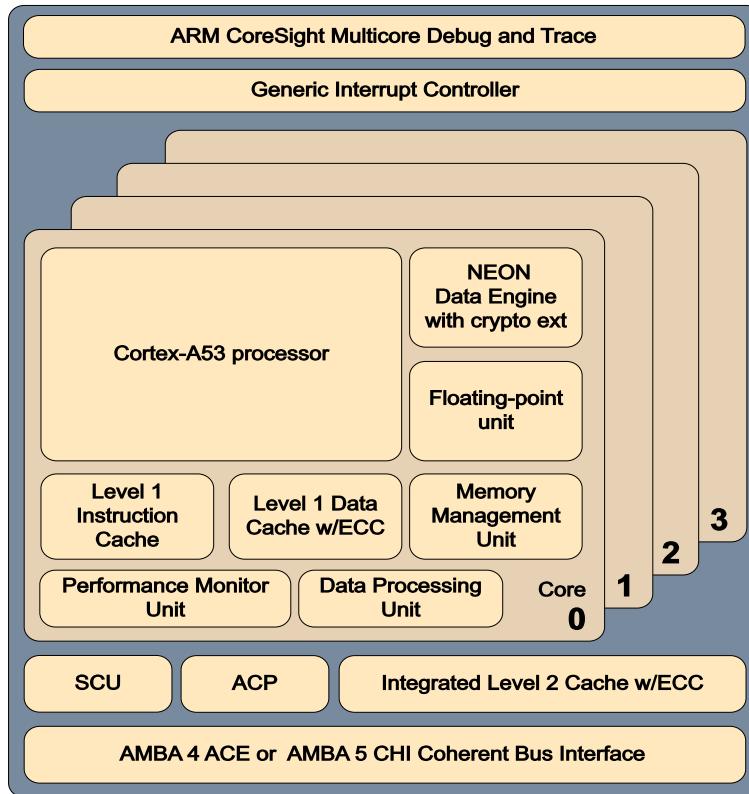


## SMP - bus compartido en tiempo

- **Estructura, buses (control, dirección y datos) e interfaces iguales a un sistema de procesador único.**
- **Desventajas:**
  - **Todas las comunicaciones con la memoria pasan por el bus: el bus es un cuello de botella.**
    - **Se agregan memorias cachés a cada procesador para disminuir los accesos a memoria.**
    - **Problema de coherencia con la memoria caché: Un dato puede estar en varias cachés, si un procesador lo modifica, debe actualizarse en todas las cachés y la memoria principal.**



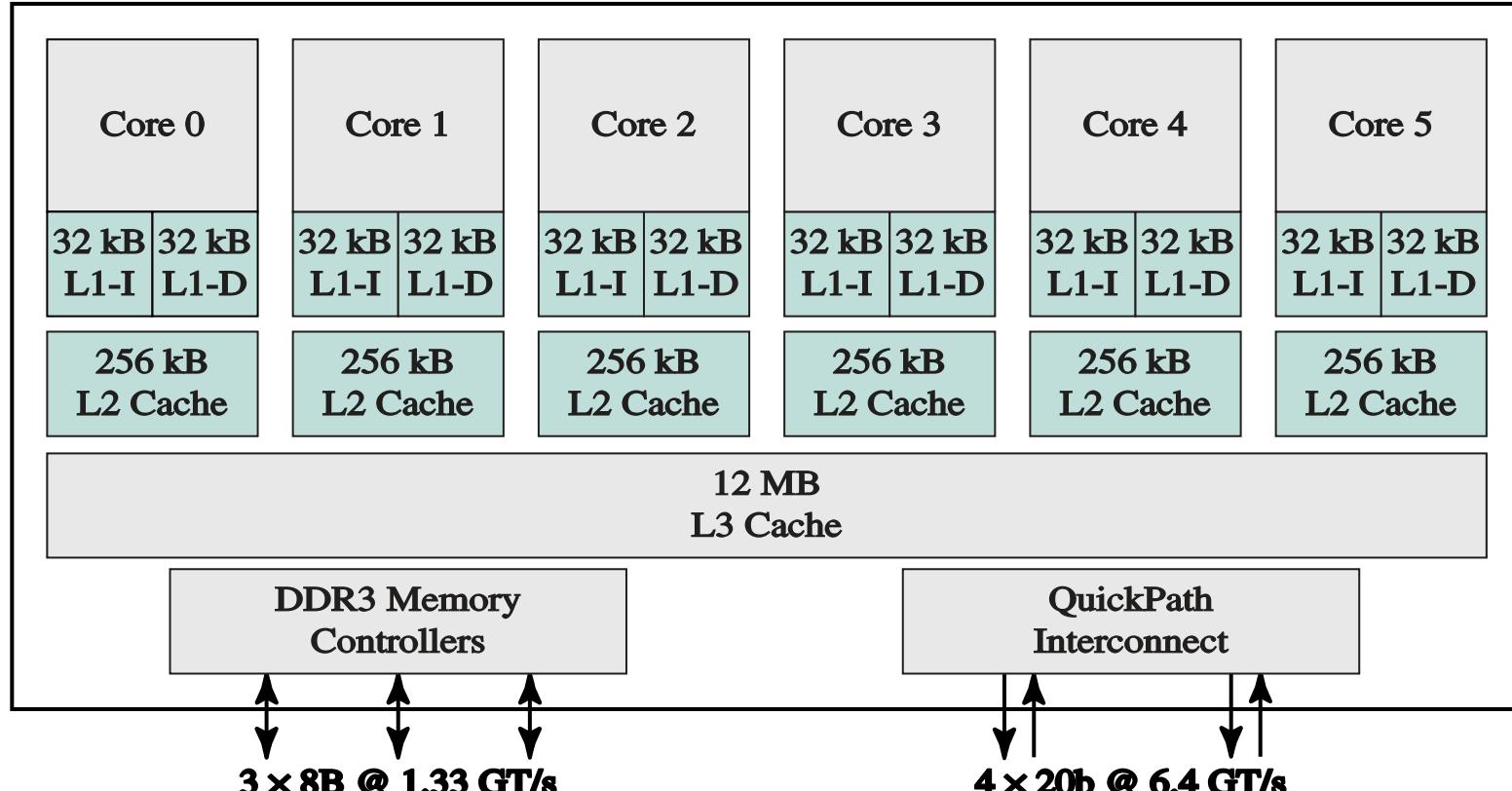
# SMP Ejemplos: ARM núcleos simétricos



Licenciatura en Ciencias de la  
Computación



## **SMP Ejemplos: Intel i7 - 990x**





## Ejemplo: Intel Core i7-990X

- Multi núcleo **simétrico**.
- Etapa de **prefetching**: El procesador examina patrones de acceso a memoria e intenta pre-cargar especulativamente datos en la caché que probablemente sean usados en el futuro.
- Controlador de memoria dentro del propio chip (más velocidad).
- QuickPath Interconnect: bus de alta velocidad de Intel, punto a punto, que permite interconexión con chipsets.
- **SmartCaché**: Nombre dado por Intel al uso compartido de algún nivel de caché (por ejemplo, la L2 en el Intel Core i7-990X).



## Sistemas multicore heterogéneo con set de instrucciones equivalente

- **Núcleos diferentes** con **igual set de instrucciones**.
- Un programa puede ejecutarse en uno u otro núcleo indistintamente (desde el punto de vista lógico).
- Ejemplo: Arquitectura **big.Little de ARM**:
  - Combinan núcleos de alto poder de procesamiento con núcleos de bajo poder de procesamiento pero menor consumo de energía.
  - Teléfonos celulares, notebooks, etc.
  - Driver **cpu\_freq**: Adapta la **frecuencia** y los **núcleos** utilizados al nivel de carga.
    - Muestra periódicamente el nivel de carga.
    - Decide si aumentar la frecuencia o disminuirla, mantener sus valores o migrar entre núcleos.



## Sistemas multicore heterogéneo set de instrucciones equivalente

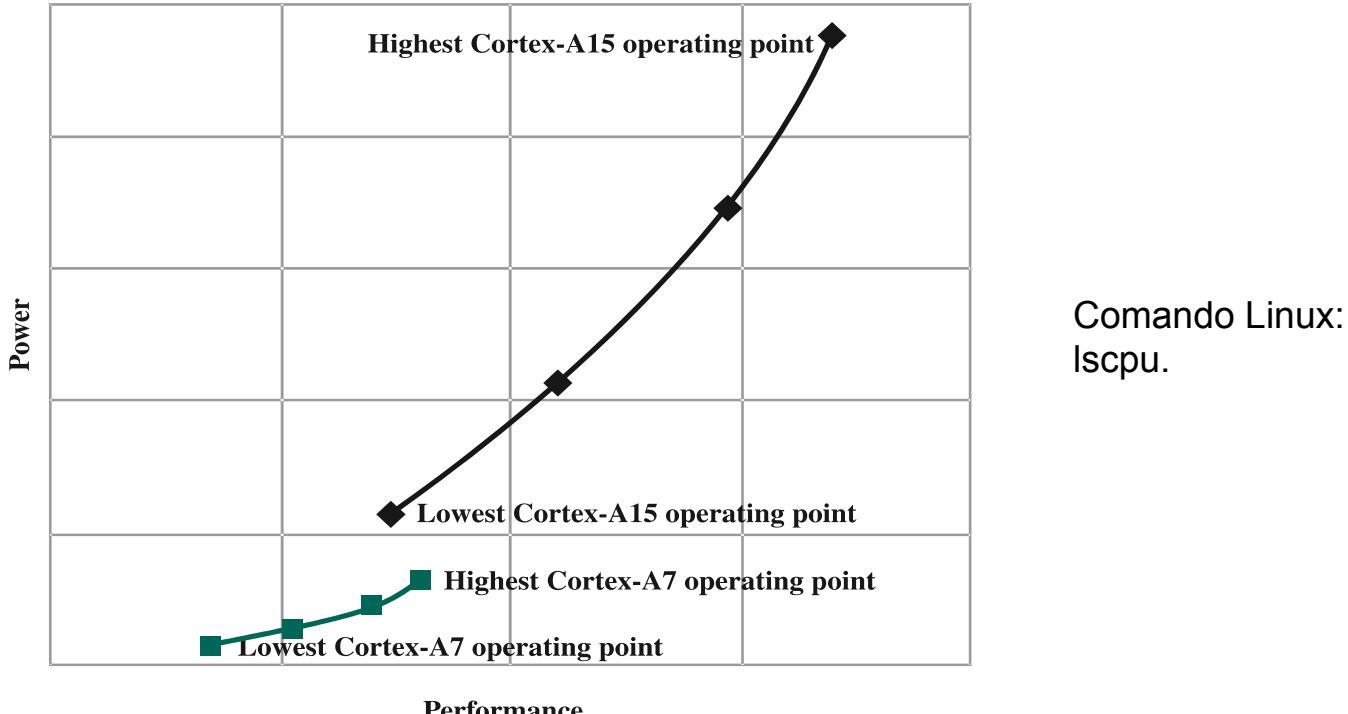
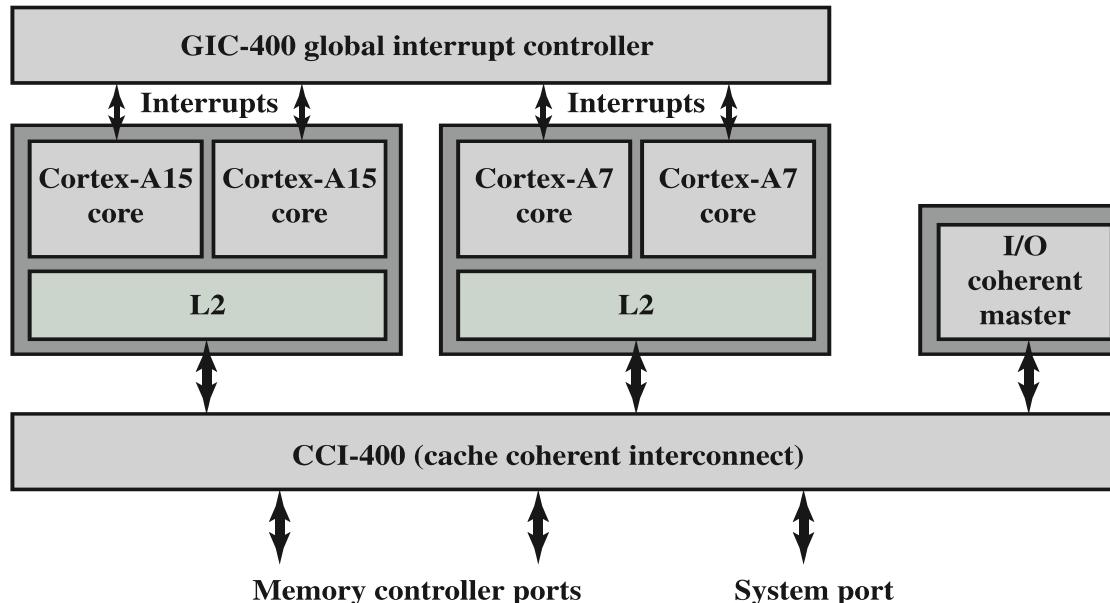


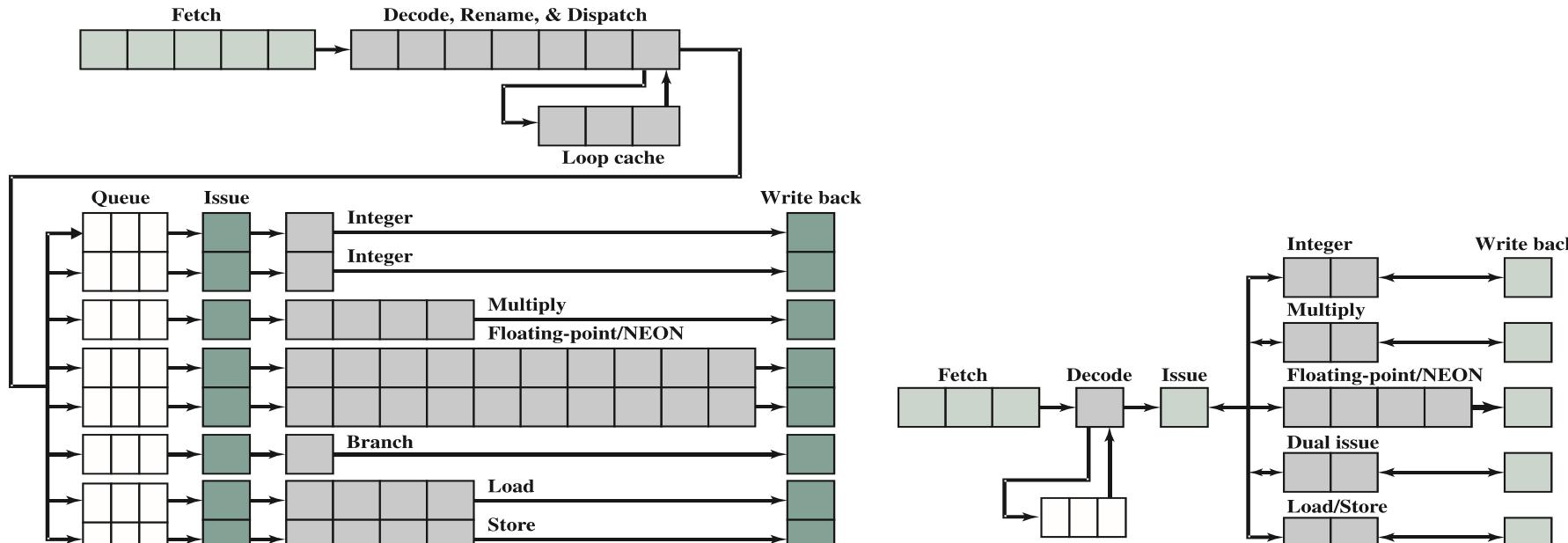
Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 673

## Sistemas multicore heterogéneo set de instrucciones equivalente Arquitectura big.Little de ARM



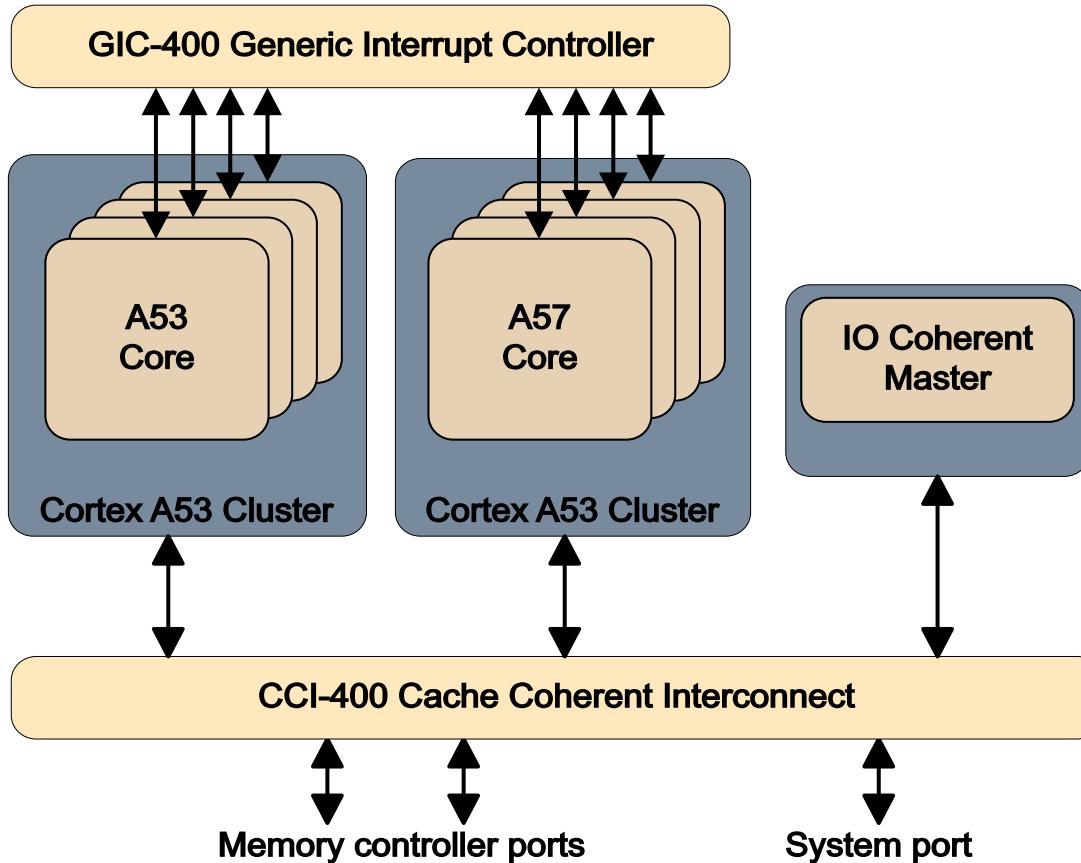
- El Cortex-A15 consume el triple de energía que el Cortex-A7
- El Cortex-A15 es el doble de potente que el Cortex-A7

## Sistemas multicore heterogéneo set de instrucciones equivalente



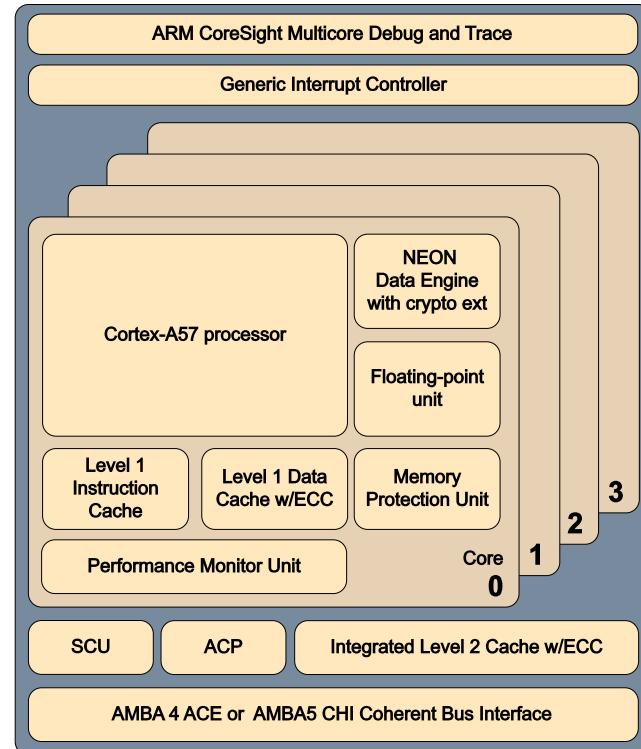
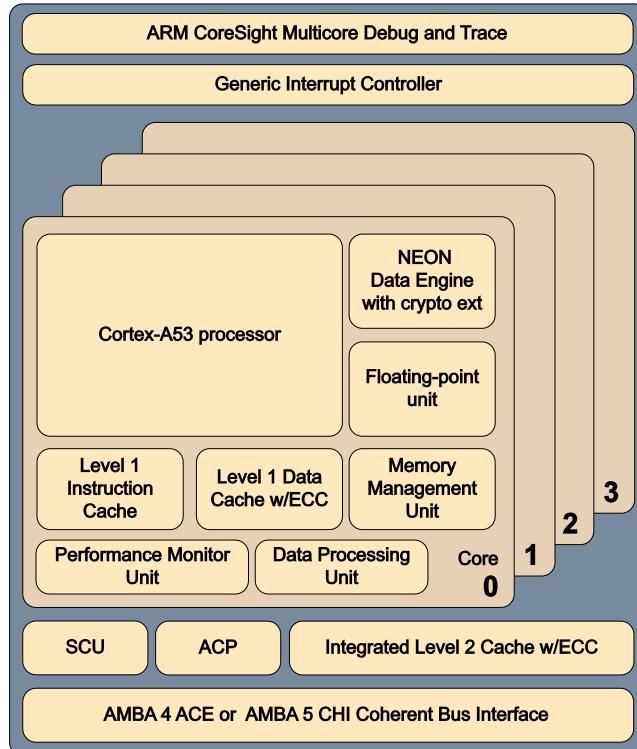
Pipeline ARM Cortex A-15 (superescalar fuera de orden de 3 vías)

Pipeline ARM Cortex A-7 (superescalar en orden 2 vías)





## Procesadores ARM Cortex A-53 y Cortex A-57 (Snapdragon 810)



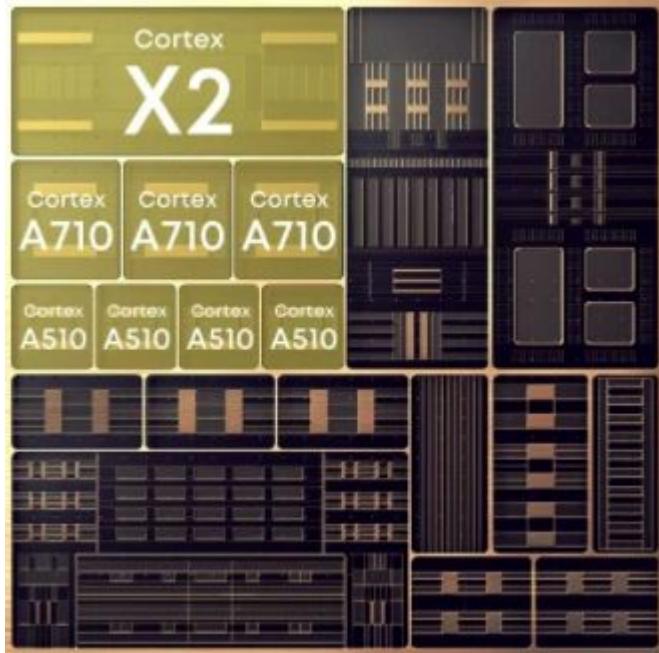


## Procesadores ARM Cortex A-53 y Cortex A-57

- SCU (Snoop Control Unit): Unidad de coherencia memoria caché. Protocolos MESI o MOESI.
- AMBA (Advanced Microcontroller Bus Architecture):
- Cortex A-53
  - Pipeline de 8 etapas, superescalar de 2 vías, en orden.
  - De uno a 4 núcleos por chip.
- Cortex A-57:
  - Pipeline de 15 o más etapas, superescalar de 3 vías, fuera de orden.
  - De uno a 4 núcleos por chip.
- El Cortex A-53 puede operar en conjunto con el Cortex A-57 mediante big.Little arquitectura.



## Qualcomm Snapdragon 8 Gen 1



Cortex X2: 3 GHz, L2 1MB

Cortex A710: 2.5 GHz.

Cortex A710: 1.8 GHz

Caché L3 compartida de 6 MB.



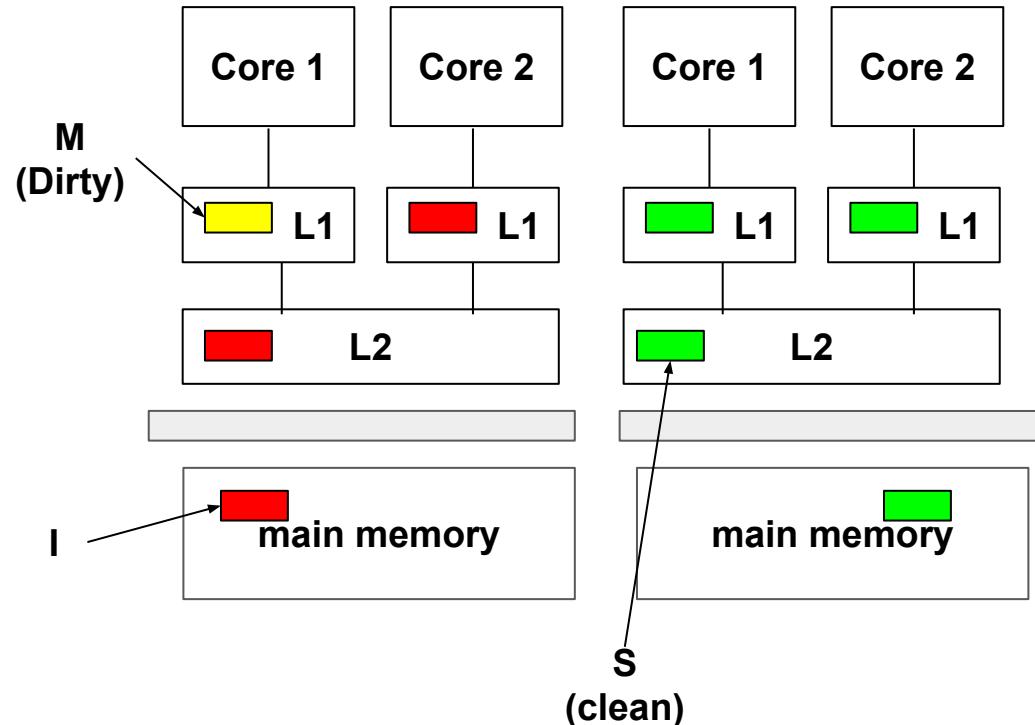
## Arquitectura big.Little de ARM. Selección de los núcleos a emplear

- Migración.
  - Dos núcleos, uno de alto y otro de bajo poder de procesamiento funcionan en par. El SO ve al par como un solo núcleo.
  - Solo uno de los núcleos de un par puede funcionar en un momento dado.
  - El driver `cpu_freq` maneja la frecuencia y tensión de cada núcleo, como también el pasar de uno a otro de un par.
- Multiproceso.
  - Los núcleos trabajan independientemente.
  - El SO ve todos los núcleos.
  - Un scheduler (global task scheduling) asigna **hilos** a los núcleos.
  - Cada tarea puede ejecutarse en el núcleo acorde.
  - Los núcleos que no están en uso pueden apagarse.



## Protocolos de Coherencia de Caché

- Un dato puede estar en una o varias cachés y en memoria principal.
- **Dirty**: El dato ha sido modificado en una caché y no ha sido actualizado a memoria principal.
- **Clean**: El dato está actualizado en todas las cachés y la memoria principal.





## Protocolos de Coherencia de Caché

- Soluciones por **Software**:
  - El compilador busca situaciones de riesgo y marca datos como inseguros.
  - No puede prever todas las situaciones.
  - No tiene una efectividad del 100%.
  - **No requiere hardware ni consumo de energía extra.**
- Soluciones por **Hardware**:
  - Usualmente las soluciones por hardware son las conocidas como Protocolos de Coherencia de Caché.
  - Transparente al programador y al compilador.
  - **Detecta todas las situaciones de falta de coherencia**. Las señales generadas cuando se detecta un problema de coherencia **consumen energía**.



## Protocolos de Coherencia de Caché

- Soluciones por hardware:
  - Protocolos **Snoopy**:
    - Actúa de forma **distribuida** (todos los controladores de caché contribuyen a resolver el problema).
    - El controlador de caché que detecta un cambio en su caché genera señales por broadcast. Todos los controladores deben escuchar esas señales todo el tiempo.
    - Dos tipos:
      - **Write-invalidate**: cuando una caché es escrita, todas las otras copias se invalidan.
        - Es el más utilizado.
        - Cada línea de caché requiere bits extras para ser marcado en diferentes estados.
      - **Write-update**: cuando una caché es escrita, actualiza todas las otras copias.



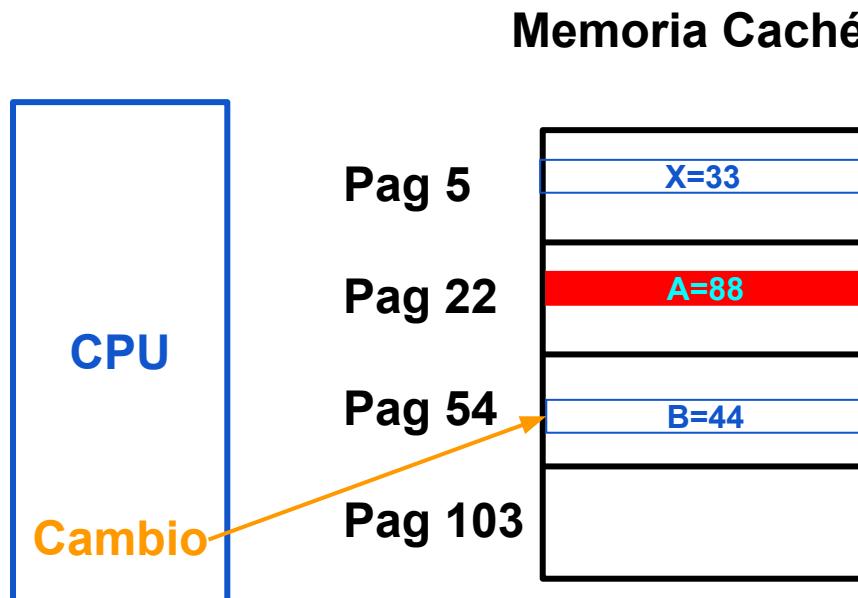
## Protocolos de Coherencia de Caché

- Soluciones por hardware:
  - Protocolos **directorio**:
    - Utiliza:
      - Un controlador centralizado (parte del controlador de memoria).
      - Directorio: contiene información sobre el contenido de todas las cachés y su estado.
    - Antes de escribir una línea, cada controlador local de caché debe solicitar acceso al controlador centralizado.
    - El controlador centralizado genera señales de invalidación y/o actualización para las otras cachés.
    - El controlador centralizado es un **cuello de botella**.
    - Utilizado en sistemas de gran tamaño con redes de interconexión complejas.

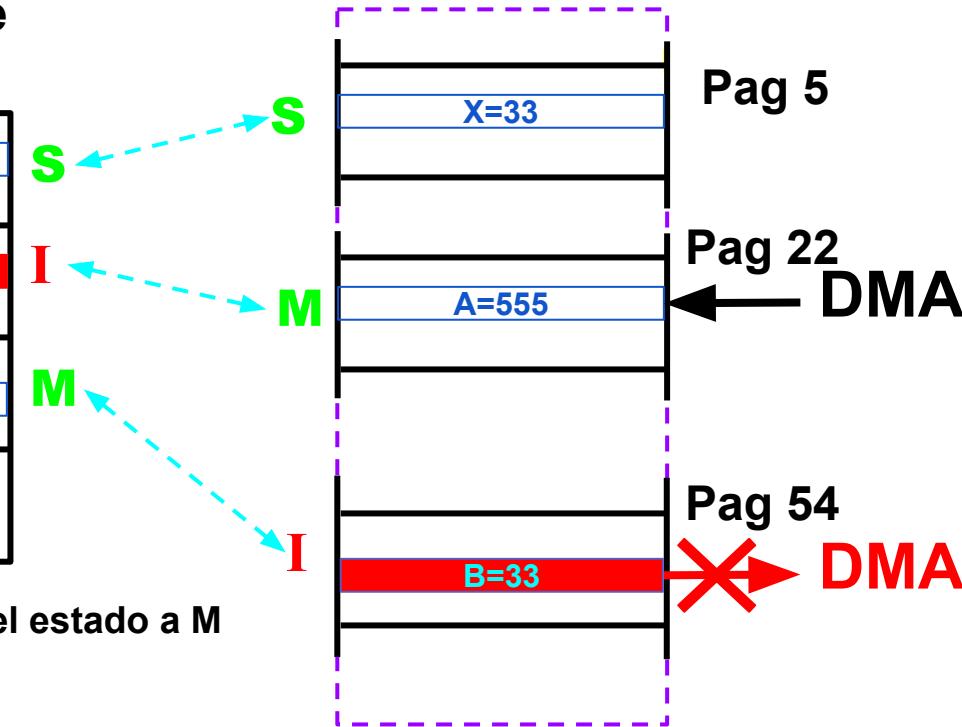


# Algoritmos de coherencia de memoria caché

- Protocolo MSI (Modified, Shared, Invalid).

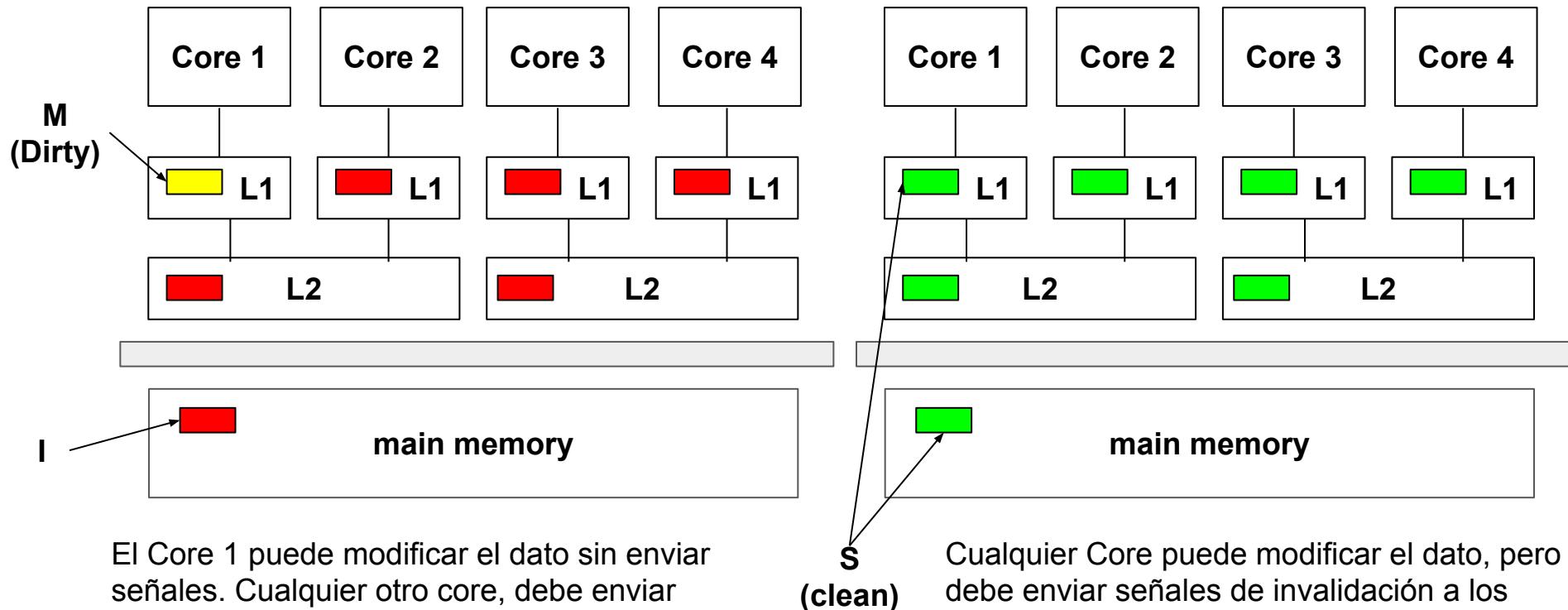


Memoria Principal



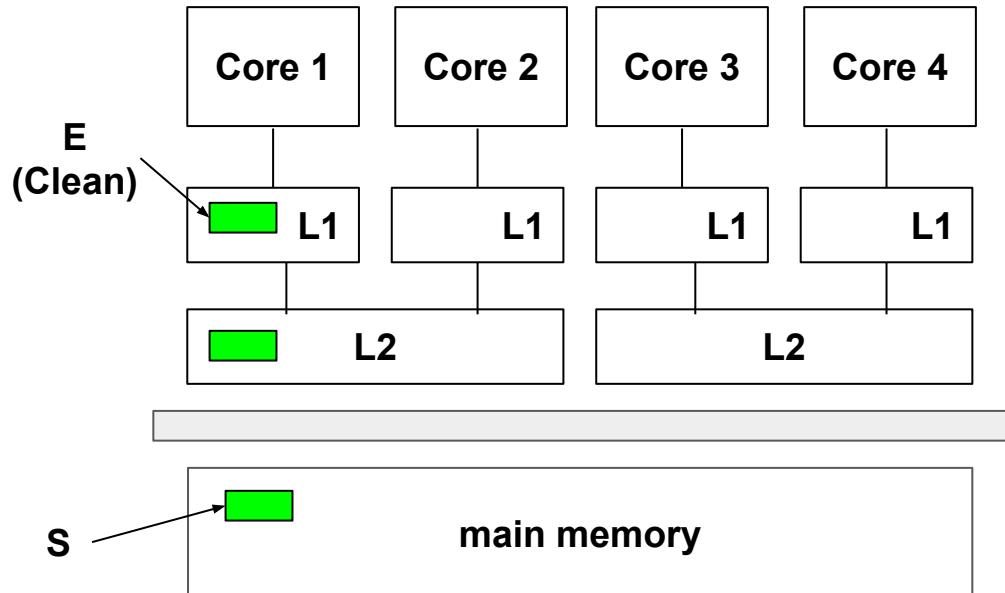
S: Se puede leer y escribir. Una escritura cambia el estado a M  
M: Se puede leer y escribir  
I: No se puede leer ni escribir

## Sistemas multicore - Coherencia memoria caché: Protocolo MESI





## Sistemas multicore - Coherencia memoria caché: Protocolo MESI



El Core 1 puede modificar el dato sin enviar señales, pero debe cambiar el estado a M.

Si el dato está solo en la caché del core 1:

- MSI: Estará en estado S. Cuando el Core 1 lo modifique, enviará señales de invalidación a la memoria principal y también enviará **innesariamente** señales de invalidación a los demás núcleos.
- MESI: Estará en estado E. Cuando el Core 1 lo modifique, enviará señales de invalidación solo a la memoria principal.

## Sistemas multicore - Coherencia memoria caché

- Protocolo MESI (Modified, Exclusive, Shared, Invalid), apto para sistemas multicore homogéneos.

	Modified	Exclusive	Shared	Invalid
Clean/Dirty	<b>Dirty</b>	<b>Clean</b>	<b>Clean</b>	-
La copia en memoria es:	desactualizada	actualizada	actualizada	-
Una escritura genera señales en el bus?	NO	No	Genera señales de invalidación	Pide actualización
Hay copias en otras cachés?	No	No	Quizás	Quizás
Puede reenviar?	Si	Si	Si	No

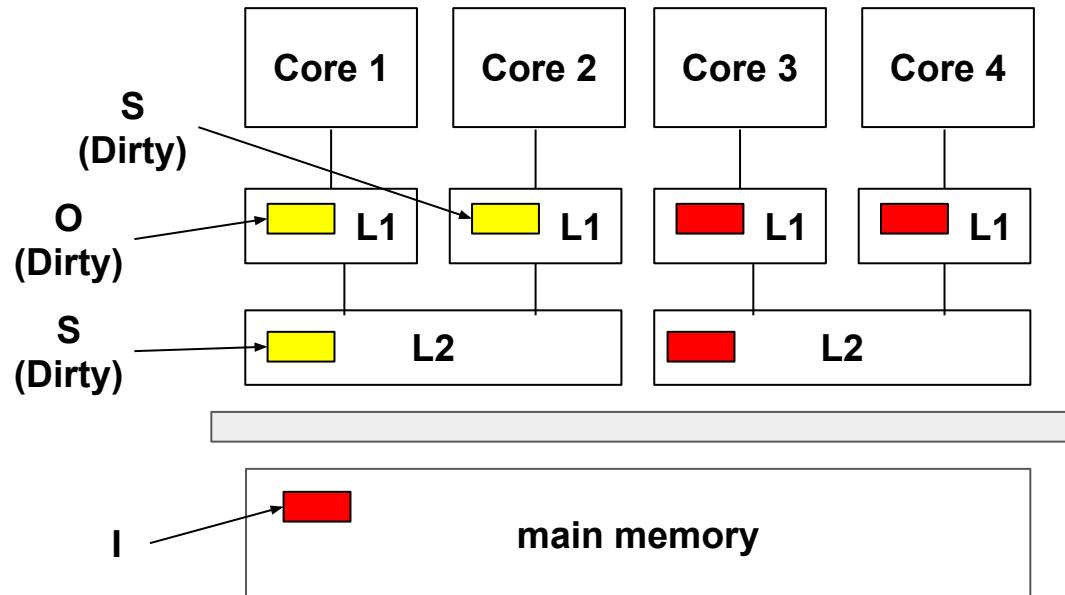


## Sistemas multicore - Coherencia memoria caché

- En arquitecturas heterogéneas, los procesadores se agrupan **conjuntos**, y cada conjunto puede tener caché compartida, pero no compartida con otros grupos.
  - Arquitectura big.Little, los procesadores más potentes pueden tener su L2 compartida, y los menos potentes otra L2 compartida.
  - CPU/GPU, los CPUs pueden tener una L3 compartida, y los GPU otra L3 compartida.
- Protocolo **MOESI**, se agrega otro estado: **Owned**
  - Un dato está en varias cachés y no ha sido actualizado a la memoria principal (Dirty).
  - Solo una de esas cachés tiene permiso de actualizar el dato a la memoria principal ante una señal miss (ese procesador está en estado Owned, los demás en shared).



## Sistemas multicore - Coherencia memoria caché: Protocolo MESI



Si el núcleo 3 o 4 envía una petición de actualización:

- MESI: Los núcleos 1 y 2 responderán.
- MOESI: Solo el núcleo 1 responderá.



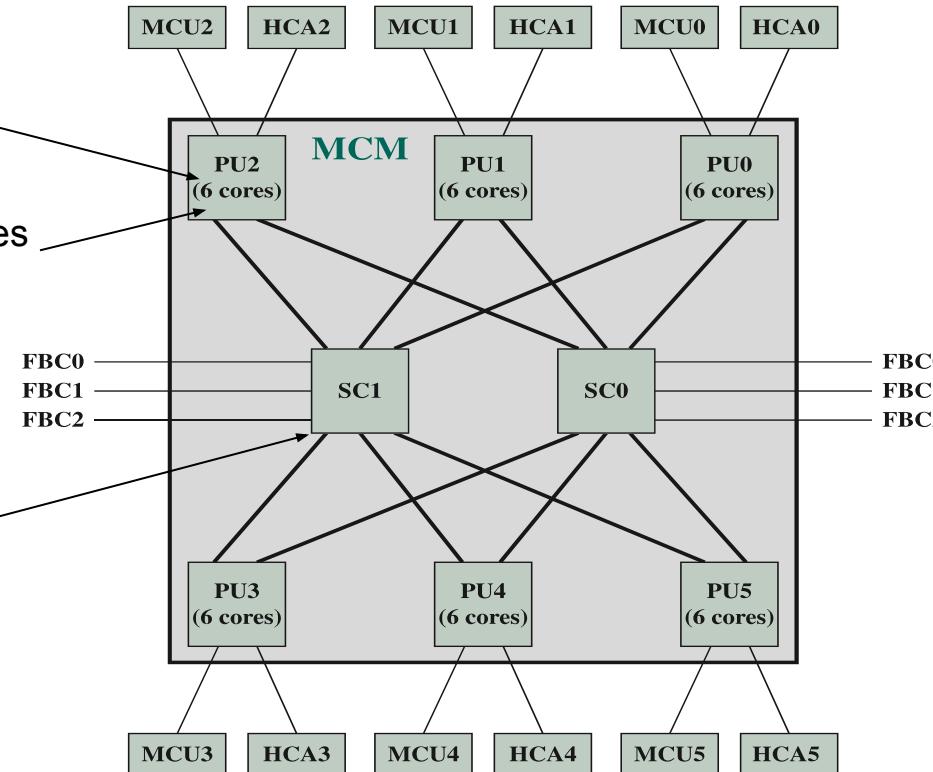
## Sistemas multicore heterogéneos - Coherencia memoria caché

- Protocolo MOESI.

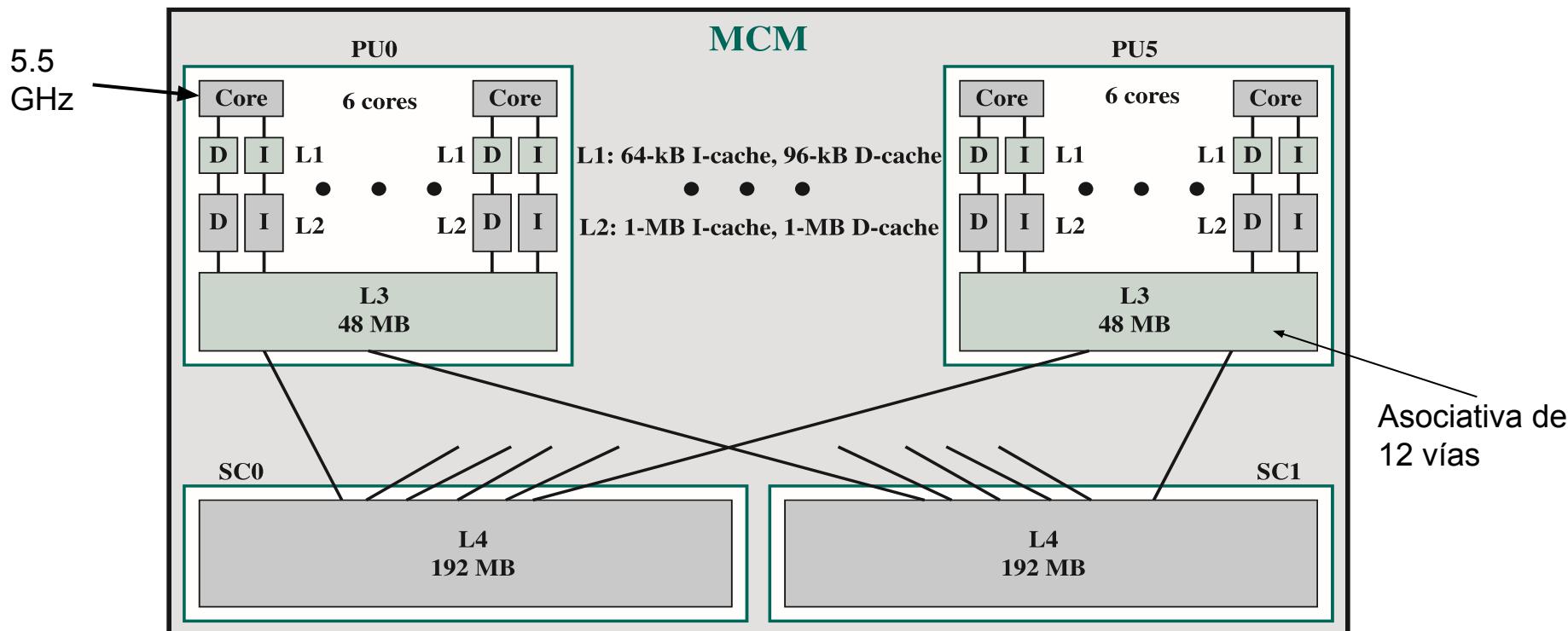
	Modified	Owned	Exclusive	Shared	Invalid
Clean/Dirty	Dirty	Dirty <sup>1</sup>	Clean	Clean/Dirty <sup>1</sup>	-
Unico?	Si	Si	Si	No	-
Puede escribir sin cambiar de estado?	Si	Si	Si	No	-
Puede reenviar?	Si	Si	Si	No	-

<sup>1</sup>Puede haber varias memorias cachés compartiendo un dato no actualizado en memoria principal. Solo una estará en estado Owned, las demás están en estado shared.

## Ejemplo: mainframe IBM zEnterprise EC12



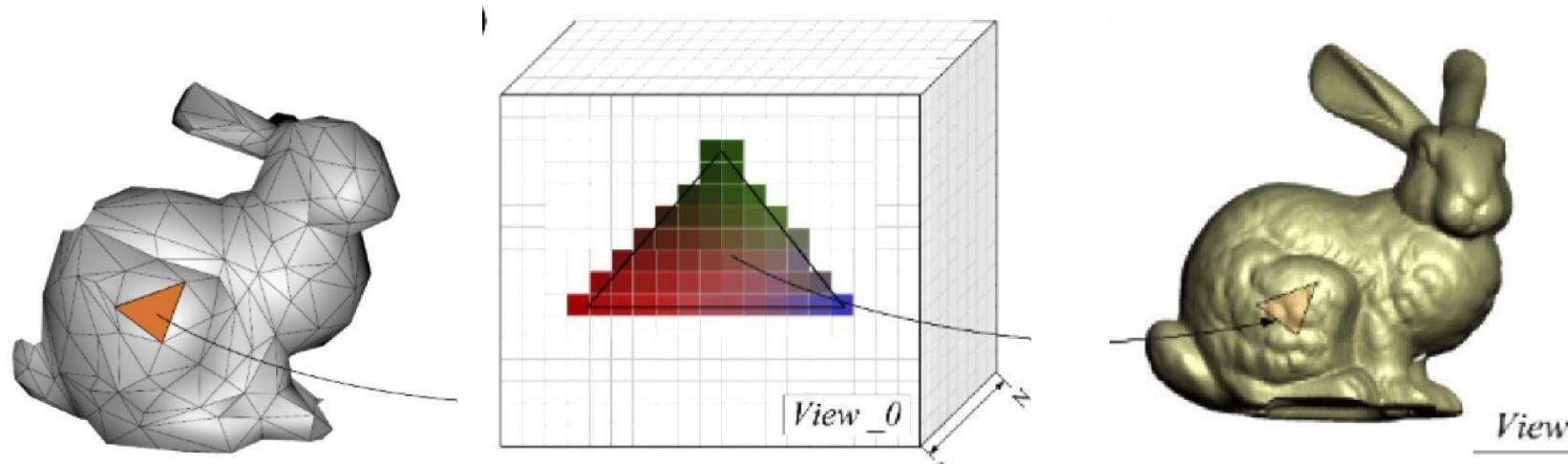
## Ejemplo: mainframe IBM zEnterprise EC12





## Multicore heterogéneo con diferentes set de instrucciones - GPU y GPGPU

- GPU: Unidad de procesamiento gráfico.
  - Gran paralelismo a nivel de hilos (operaciones simples en paralelo).
  - Inicialmente pensadas para computación gráfica.
    - Renderización: Generar imagen a partir de datos.





- GPU: Actualmente poseen miles de núcleos
  - Cada uno con pocas unidades de ejecución específicas (operaciones enteras y/o con punto flotante) y gran cantidad de registros.
- GPGPU: (General Purpose GPU) GPU para procesamiento general.
  - Se aprovecha la elevada capacidad de procesamiento paralelo de las GPU.
  - Implementaciones:
    - GPU como co-procesadores.
    - Máquinas con GPU como procesador central + un CPU para tareas auxiliares (clúster).



## Multicore heterogéneo - CPU/GPU

- Comparación GPU - CPU (AMD A10 5800K)

	CPU	GPU
<b>Clock frequency (GHz)</b>	3.8	0.8
<b>Cores</b>	4	384
<b>FLOPS/core</b>	8	2
<b>GFLOPS</b>	121.6	614.4

Tabla obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 669

- NVIDIA Tesla V100

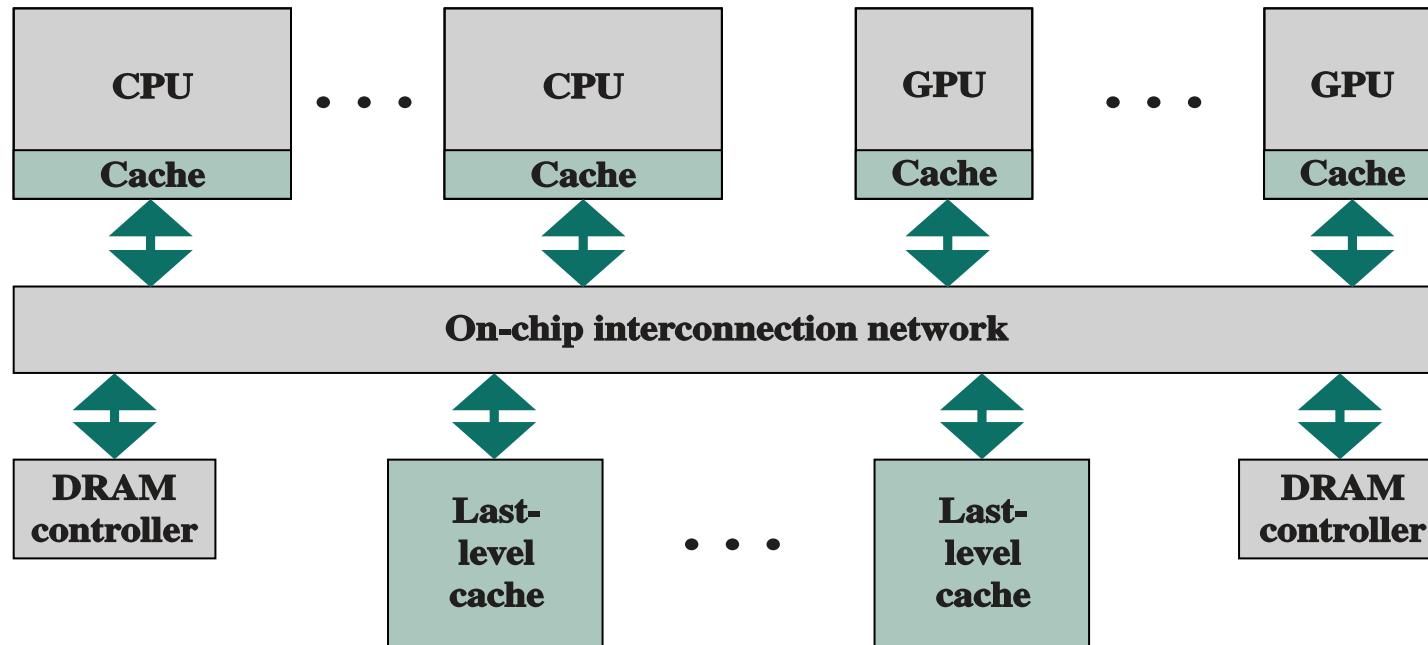
	CPU	GPU
Núcleos	24	2688
Clock (GHz)	2.7	1.38
GFlops	1037	7419



	<b>GeForce RTX 3090 Ti</b>	<b>GeForce RTX 3090</b>	<b>GeForce RTX 3080 Ti</b>	<b>GeForce RTX 3080</b>	<b>GeForce RTX 3070 Ti</b>	<b>GeForce RTX 3070</b>
Núcleos CUDA de NVIDIA	10752	10496	10240	8960 / 8704	6144	5888
Frecuencia Modificada (GHz)	1.86	1.70	1.67	1.71	1.77	1.73
Tamaño de la Memoria	24 GB	24 GB	12 GB	12 GB / 10 GB	8 GB	8 GB
Tipo de Memoria	GDDR6X	GDDR6X	GDDR6X	GDDR6X	GDDR6X	GDDR6



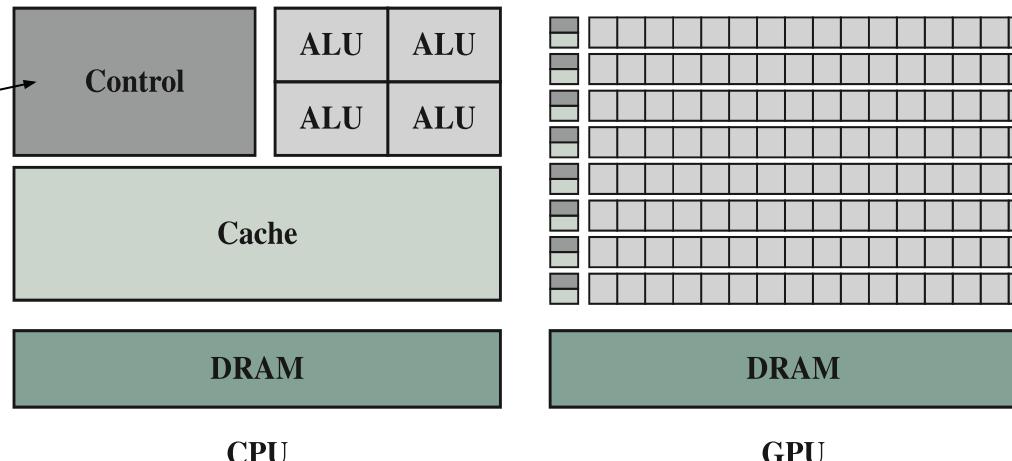
## Multicore - CPU/GPU



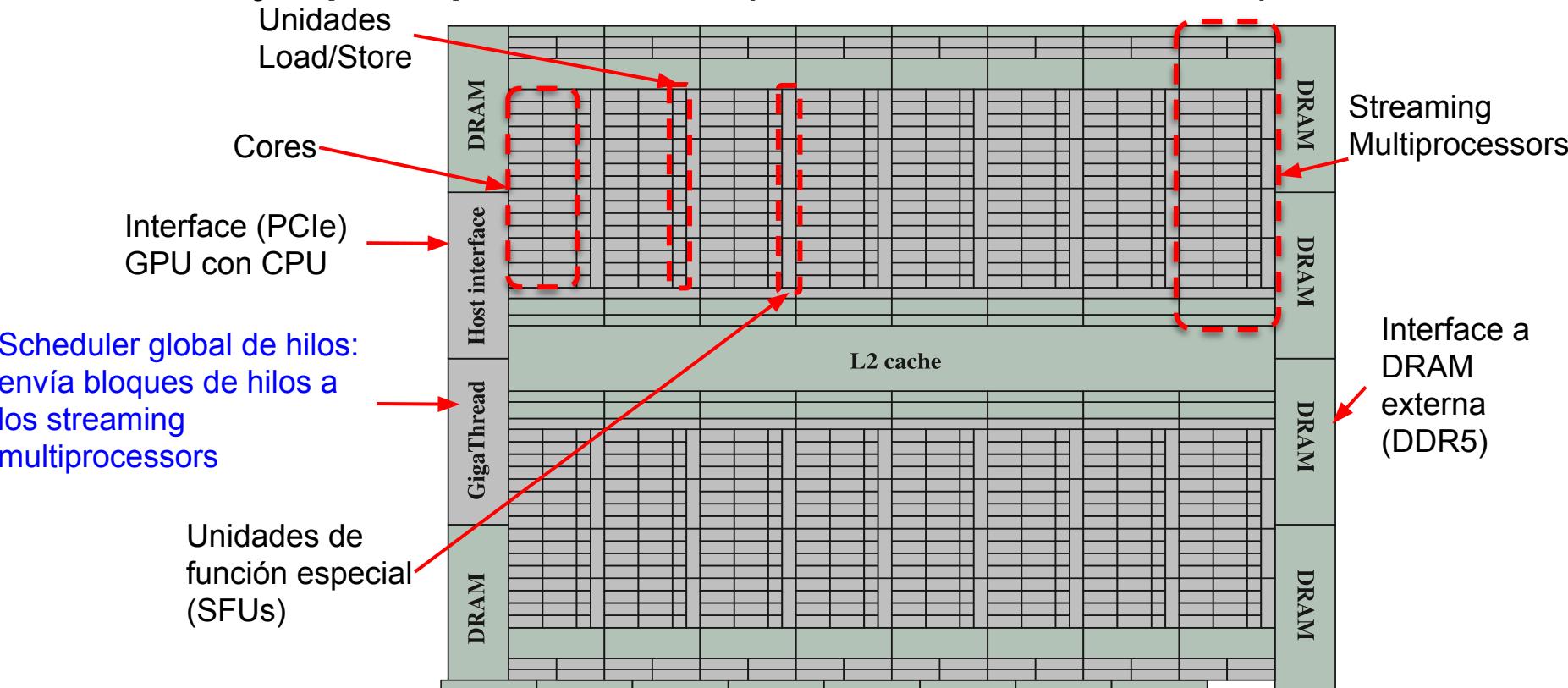
## CPU vs GPU

- CPU: Mucho más caché y lógica de control.
- GPU: Mucho más área dedicada a unidades aritméticas

Lógica de control:  
fuera de orden,  
predicción de  
saltos,  
dependencia de  
datos, etc.



## Ejemplo arquitectura GPU (NVIDIA Fermi architecture)





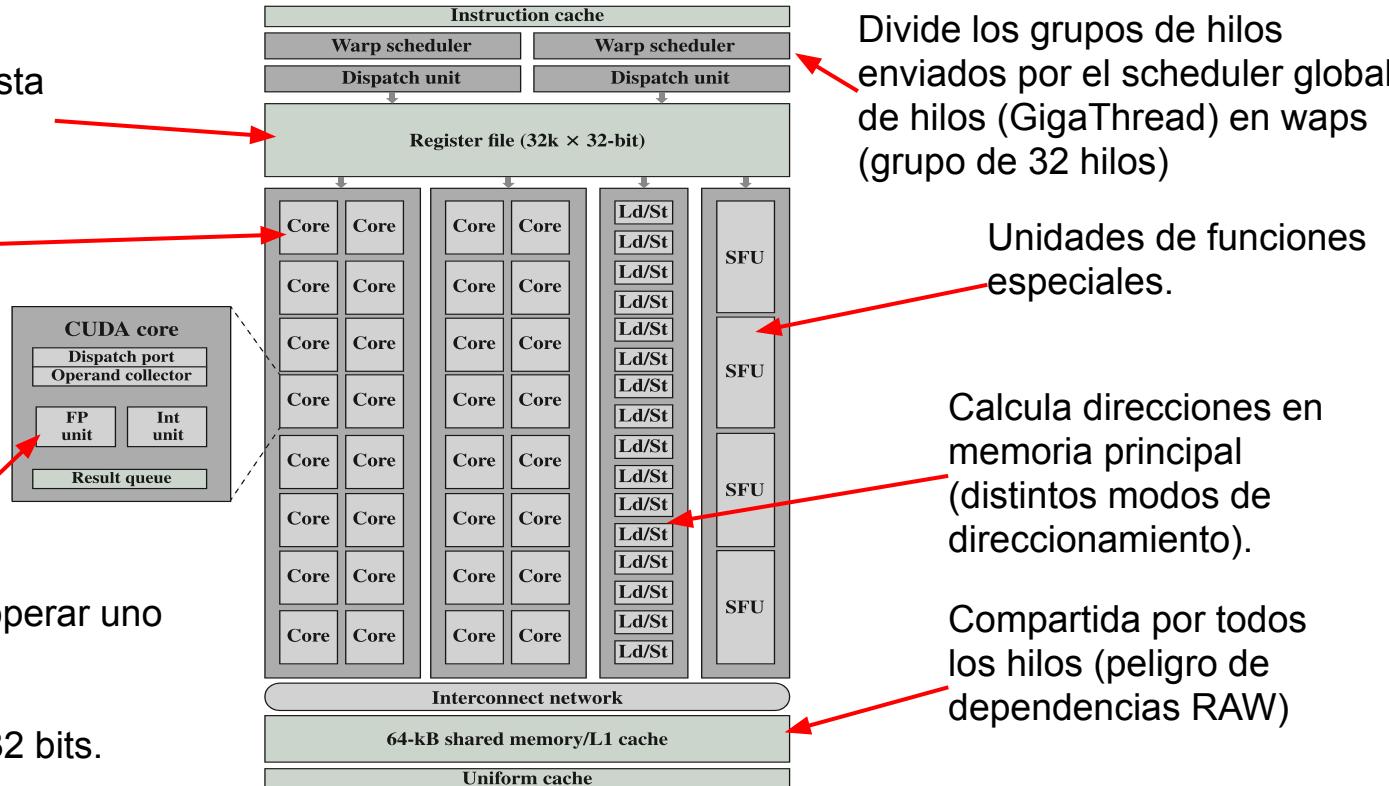
## Ejemplo arquitectura GPU (NVIDIA Fermi architecture)

- 512 núcleos.
  - 32 núcleos (streaming processors) por streaming Multiprocessors, 16 multiprocesadores de flujos.
- Los hilos se dividen en:
  - Bloques: El GigaThread (scheduler global de hilos) asigna los bloques de hilos a los streaming processors.
  - Wraps: bloque de 32 hilos que comienzan en la misma dirección y poseen thread IDs consecutivos.
    - Los “dual warp scheduler” dividen los bloques de hilos se dividen en wraps.
    - Cada hilo posee su contador de programa y registros asignados.

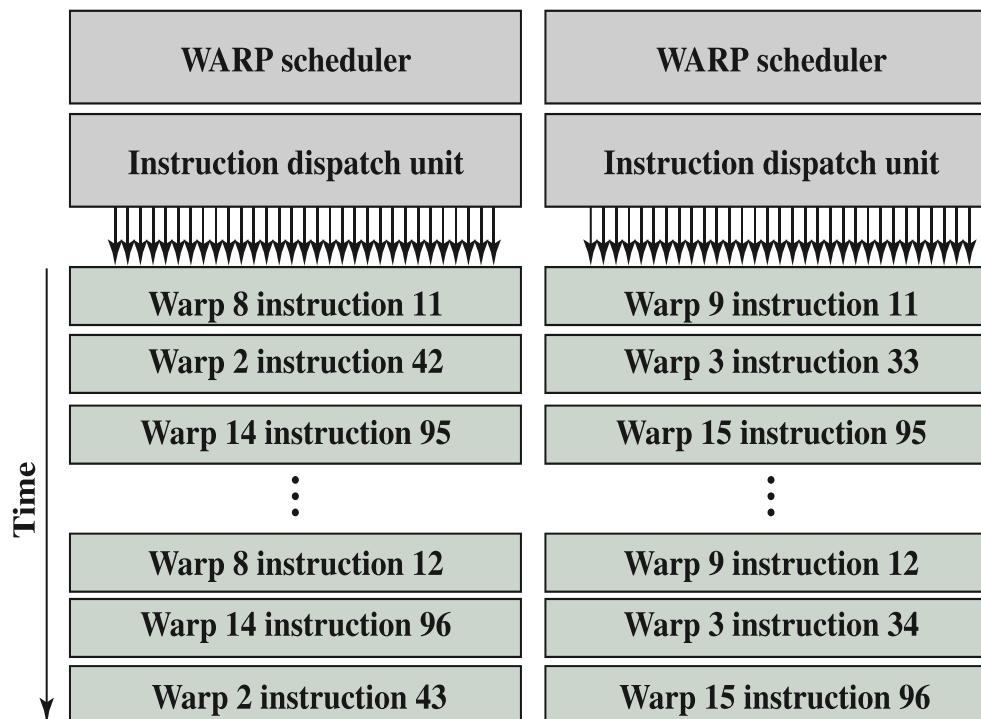
## GPUs arquitectura: Multiprocesador de flujos (NVIDIA Fermi architecture)

A cada hilo se le asignan hasta 64 registros dedicados (no compartidos)

Cada core ejecuta un hilo



## GPUs arquitectura: scheduler y dispatch (NVIDIA Fermi architecture)



Divide los grupos de hilos enviados por el scheduler global de hilos (GigaThread) en waps.

Cada warp scheduler asigna la mitad de un warp (16 hilos) a un conjunto de núcleos, unidades de load/store o unidades de funciones especiales.



## GPUs arquitectura: scheduler y dispatch (NVIDIA Fermi architecture)

Factores que afectan la performance:

- Conflicto de recursos (especialmente operaciones trascendentales).
- Saltos.
- Accesos a memoria (atenuados).

Optimización:

- Los bloques de hilos deben ser mayores a la cantidad de núcleos en un streaming processor, y ser múltiplos de 32.



## GPUs arquitectura: scheduler y dispatch (NVIDIA Fermi architecture)

Unidades de procesamiento:

- Cores: Pueden realizar una operación con enteros (32 o 64 bits) o con operandos en punto flotante de simple precisión (32 bits) en un clock del reloj.
- Unidades de funciones especiales: realizan operaciones trascendentales (seno, coseno, raíz cuadrada, etc.) en un ciclo de reloj.
- Load store unit: acceden a memoria (utilizan los métodos de direccionamiento conocidos).



## Multicore - CPU/GPU

- Intercambio de datos entre la CPU y la GPU
  - Memorias dedicadas para la transferencia entre los CPUs y las GPUs:
    - El hilo o proceso (CPU) copia los datos en la memoria del GPU.
    - La GPU procesa los datos.
    - El hilo o proceso recupera los datos desde la memoria del GPU.
    - **Ventaja: No hay problemas de coherencia.**
    - **Desventaja: Pérdida de tiempo.**
  - Memorias compartidas:
    - CPUs y GPUs ven todo el espacio de memoria.
    - **Ventaja: Mayor velocidad (no es necesario que los datos pasen por la memoria intermedia)**
    - **Desventaja: Problemas de coherencia de la caché.**



## GPUs arquitectura: (NVIDIA Fermi) organización de memoria.

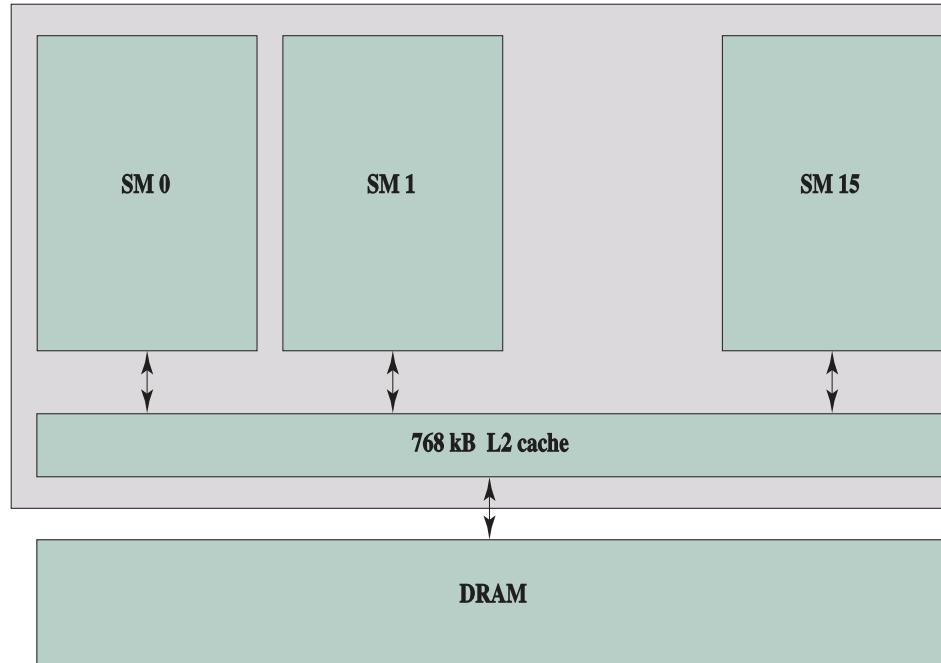
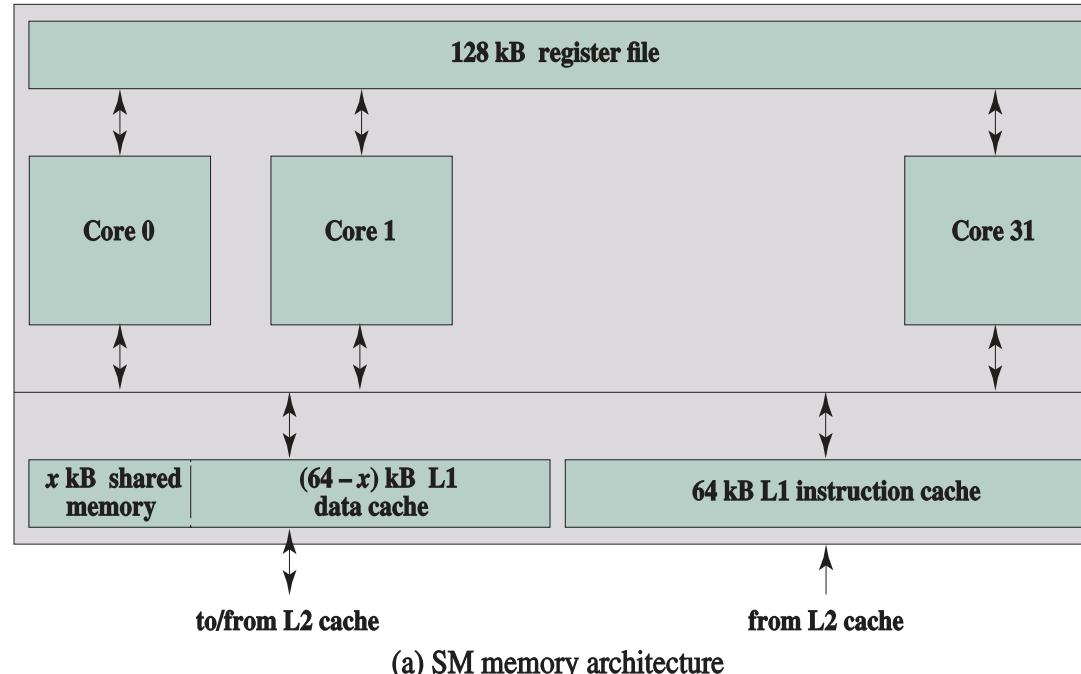


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 699



## GPUs arquitectura: (NVIDIA Fermi) organización de memoria.

### Memoria de un multiprocesador de flujos



- A cada hilo se le asigna un conjunto de registros.
- Shared memory: Memoria compartida por los hilos. Facilita la comunicación entre hilos.



## Uso de memoria en CUDA.

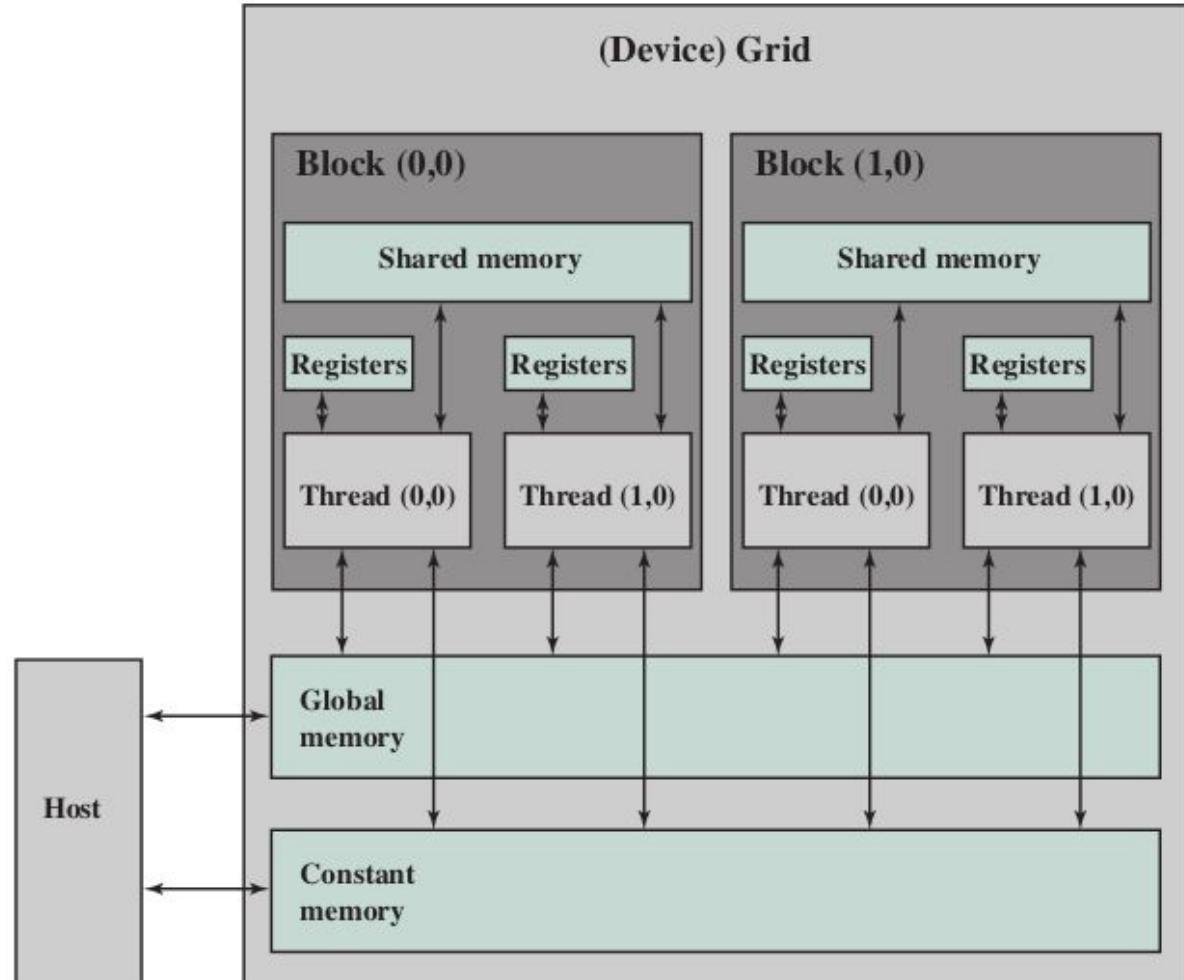
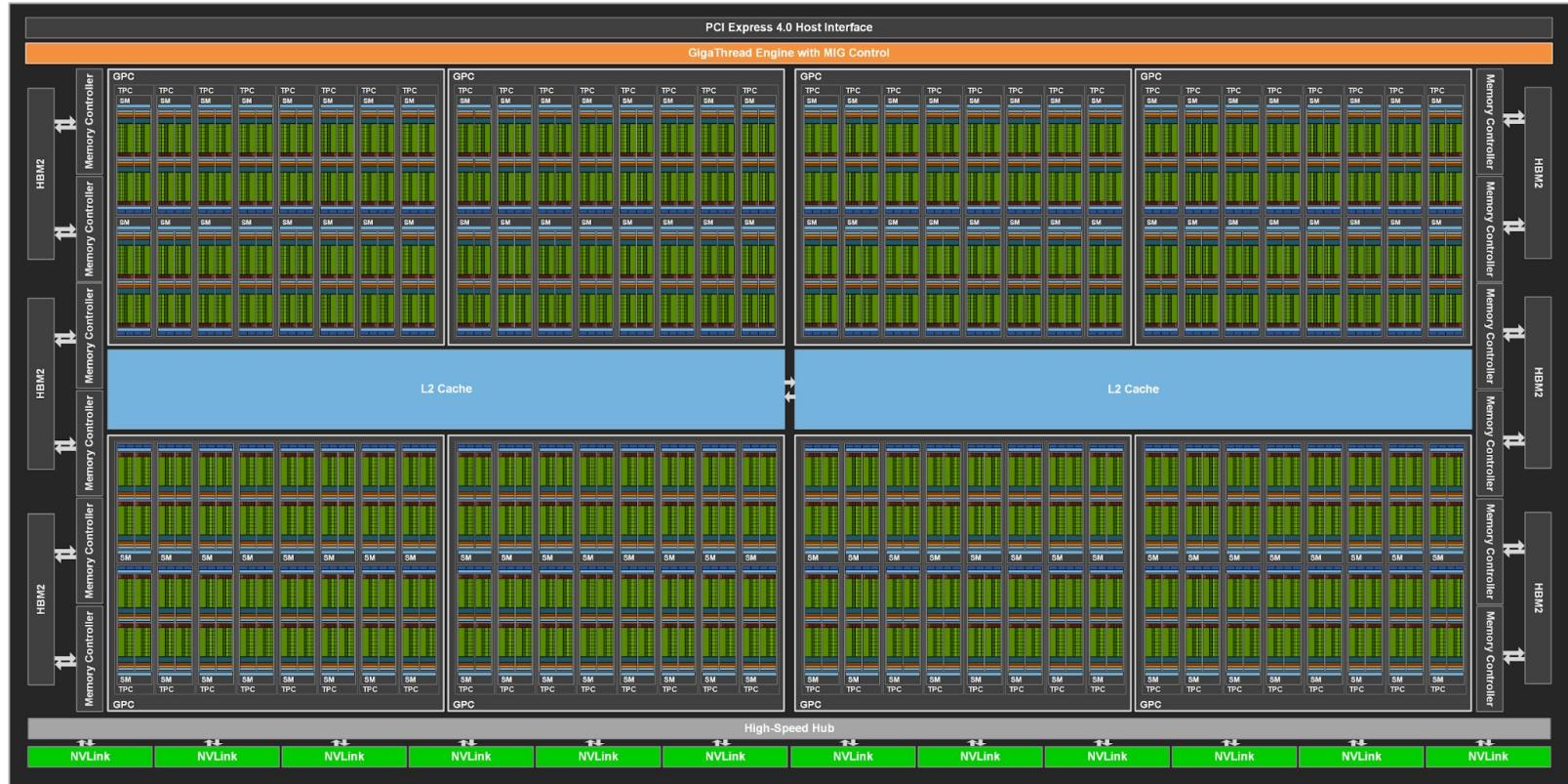


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 700.



## Ejemplo: Arquitectura Nvidia Ampere





## Ejemplo: Arquitectura Nvidia Ampere



Parte superior de  
un SM

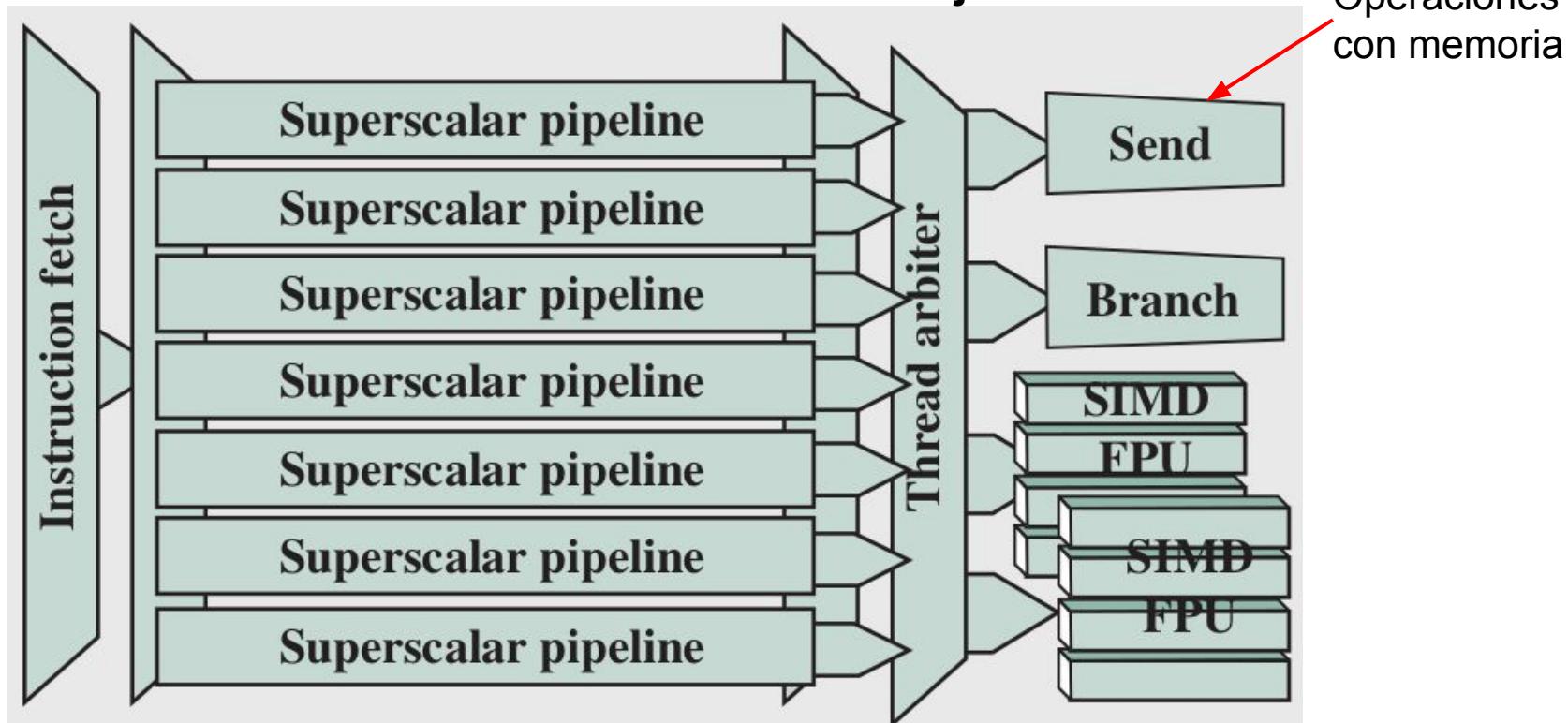


## Ejemplo: Arquitectura Nvidia Ampere

- Lanzado al mercado en 2020.
- Aplicación:
  - Cloud computing, Análisis de datos, computación científica, gaming, servicios 5G, renderizado.
- Tensor cores: operaciones con matrices (gran speedup para multiplicaciones de matrices).
- Ejemplo de implementación: NVIDIA A100 (7 nm).
  - 108 SMs
  - 6912 FP32 cores (64 por SM)
  - 3456 FP64 cores (32 por núcleo)
  - 6912 INT32 cores (64 por SM)
  - 19.5 TFLOP (operaciones de 32 bits)
  - 400 W (TDP)



## GPU Intel GEN 8: Unidad de ejecución



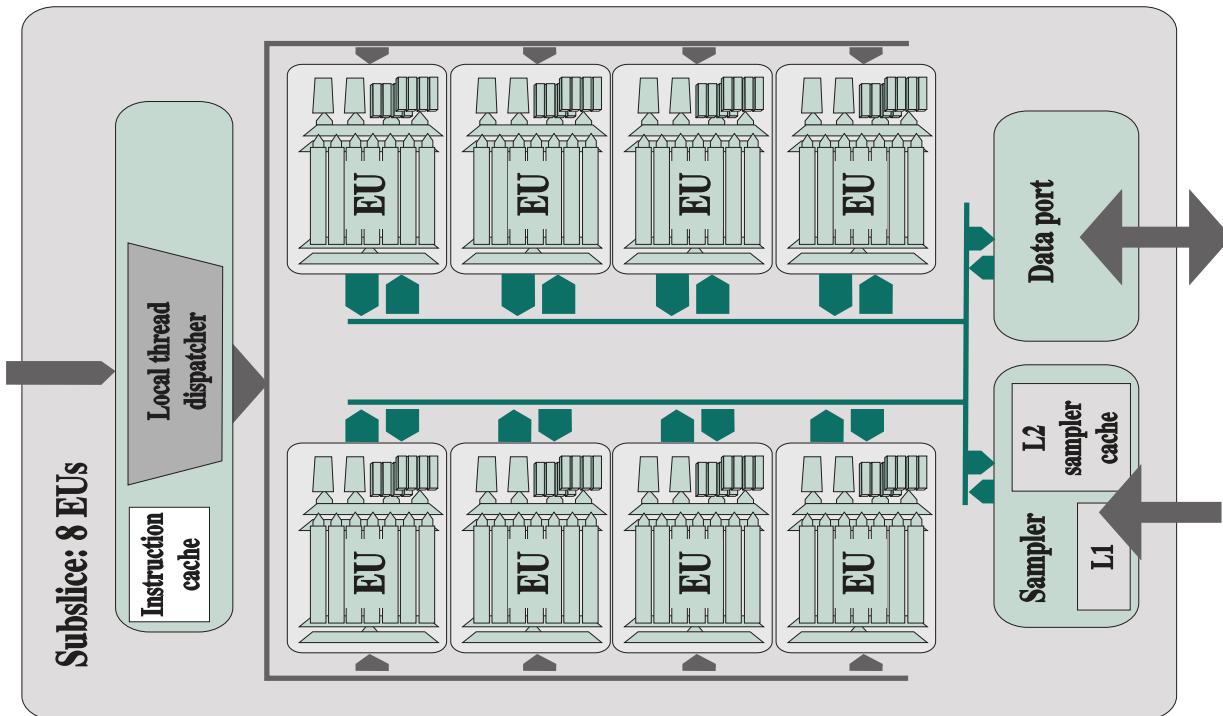


## GPU Intel GEN 8: Unidad de ejecución

- Multithreading simultáneo de 7 hilos.
- 128 registros por hilo de 32 bytes.
  - Cada registro almacena 8 datos de 32 bits.
  - Pueden agruparse para almacenar datos de mayor longitud.
- SIMD FPU (floating-point units): operaciones SIMD con flotantes o enteros.
  - Cada EU Puede ejecutar operaciones SIMD sobre 16 datos.
- 4 operaciones al mismo tiempo (2 SIMD FPU, 1 Send y 1 Branch).
- Procesadores Intel i3, i5, i7, core M, Atom (móviles), Xeon (servidores), etc.

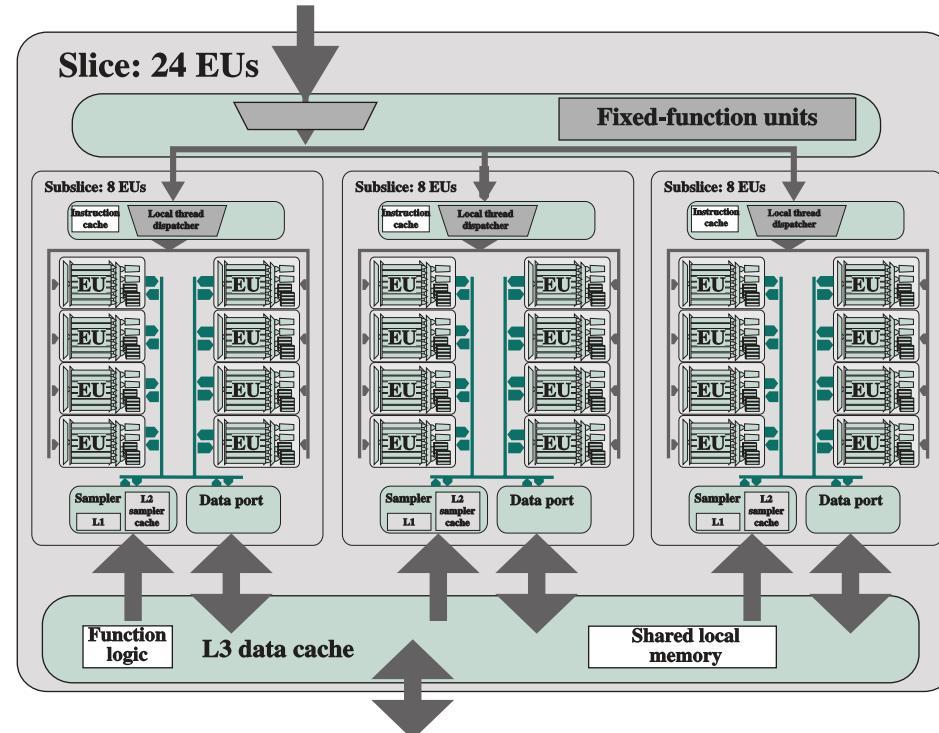
## GPU Intel GEN 8: Subslice

- 8 unidades de ejecución por subslice (56 hilos).
- Sampler: operaciones sobre imágenes (formatos, etc.)

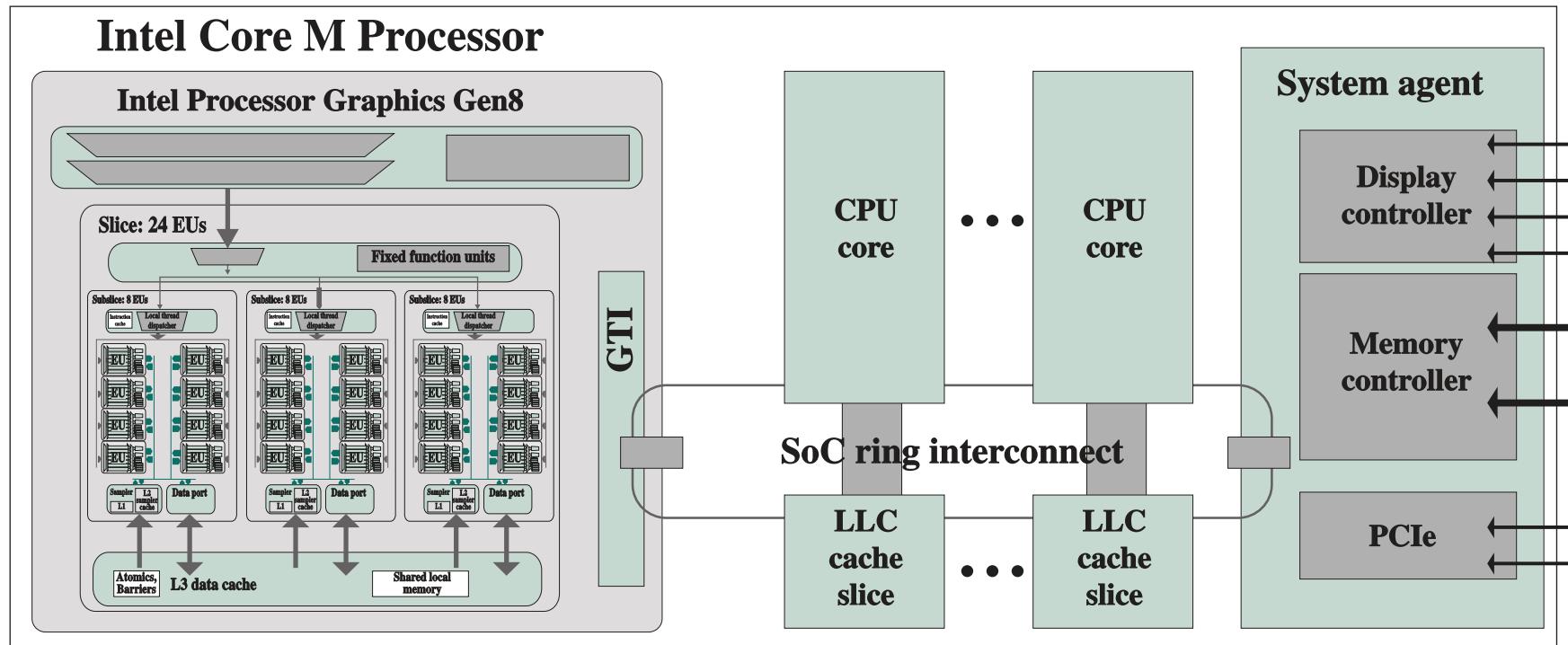


## GPU Intel GEN 8: Slice

- 3 subslices por slice.
- L3 dividida en bancos para permitir varios accesos simultáneos (1 por banco).
- Un GPU GEN 8 puede incluir 1 o varios slices.
- Shared local memory: memoria compartida. Permite a las UE compartir variables temporales.



## GPU Intel GEN 8: Ejemplo Procesador Intel Core M



GPU Intel Gen 8.  
GPU del procesador  
Intel Core i7-11370  
(2021)

## Processor Graphics

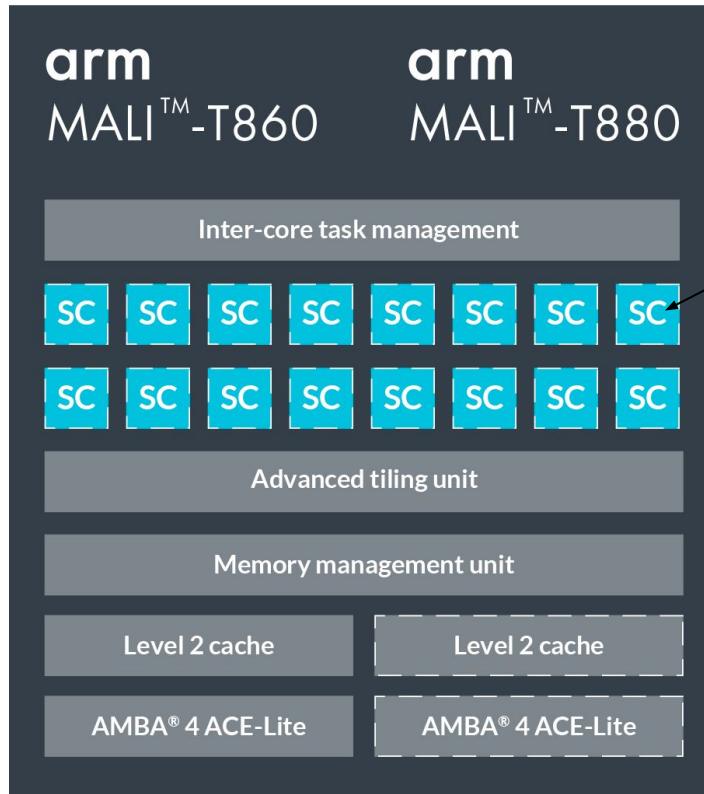
Processor Graphics ‡	Intel® Iris® Xe Graphics
Graphics Max Dynamic Frequency	1.35 GHz
Graphics Output	eDP 1.4b, MIPI-DSI 2.0, DP 1.4, HDMI 2.0b
Execution Units	96 ← 672 hilos
Max Resolution (HDMI)‡	4096x2304@60Hz
Max Resolution (DP)‡	7680x4320@60Hz
Max Resolution (eDP - Integrated Flat Panel)‡	4096x2304@60Hz
DirectX® Support	12.1
OpenGL® Support	4.6
OpenCL® Support	3.0

Información obtenida de:

<https://ark.intel.com/content/www/us/en/ark/products/196655/intel-core-i711370h-processor-12m-cache-up-to-4-80-ghz-with-ipu.html>

## GPU ARM Mali

- Soporte para OpenCL.
- 256 hilos.
- 16 registros de 128 bits.



16 Núcleos (20 GFLOP aproximadamente)

Ver algunos ejemplos en:  
<https://armkeil.blob.core.windows.net/developer/Files/pdf/product-brief/arm-gpu-processor-comparison-table.pdf>



## Procesador Qualcomm Snapdragon 845

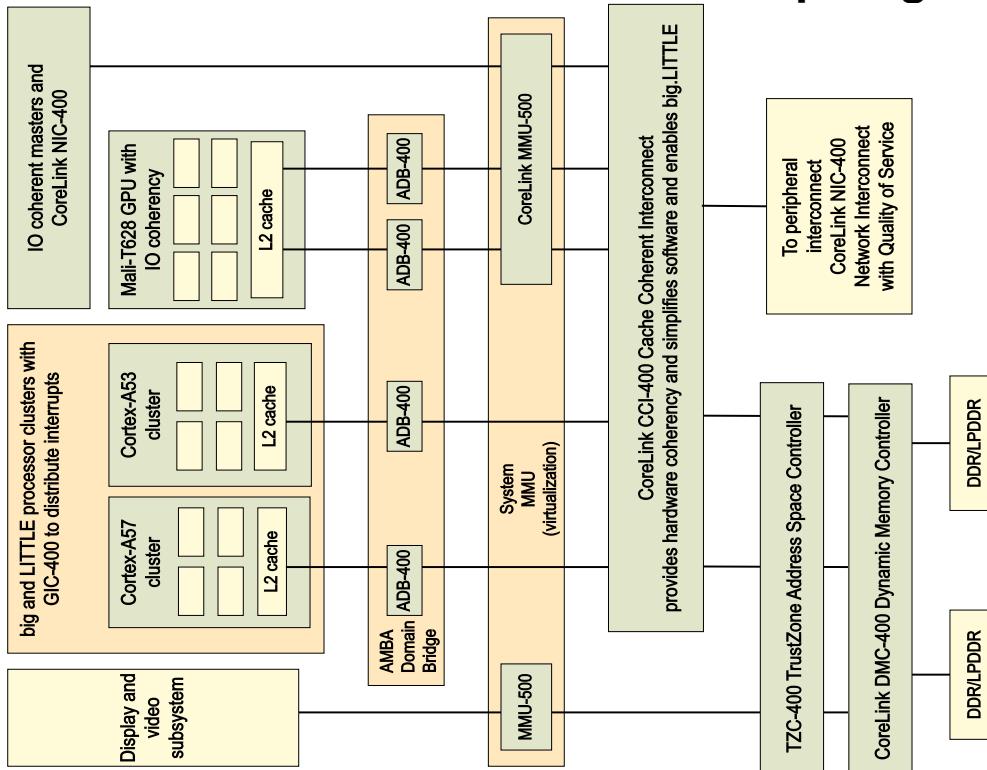


Figura obtenida de ARM Cortex-A Series Programmer's Guide for ARMv8-A Version: 1.0, página 14-19



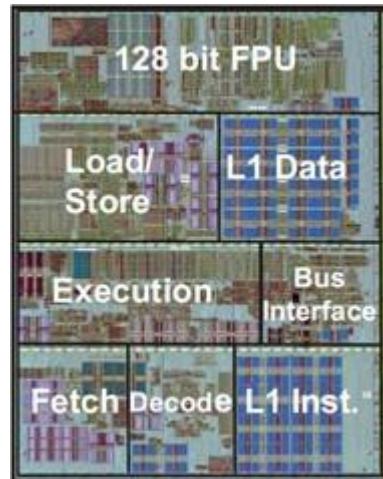
## Ejemplo de GPU AMD: AMD Radeon™ RX 6950 XT

- 5120 núcleos agrupados en 80 unidades de cómputo.
  - AMD organiza los núcleos de sus GPUs en unidades de cómputo de 64 núcleos cada uno.
- + 320 Unidades de textura. 1.89 a 2.31 GHz.
- 47 TFlops.
- Caché: 128 MB
- Memoria: 16 GB.

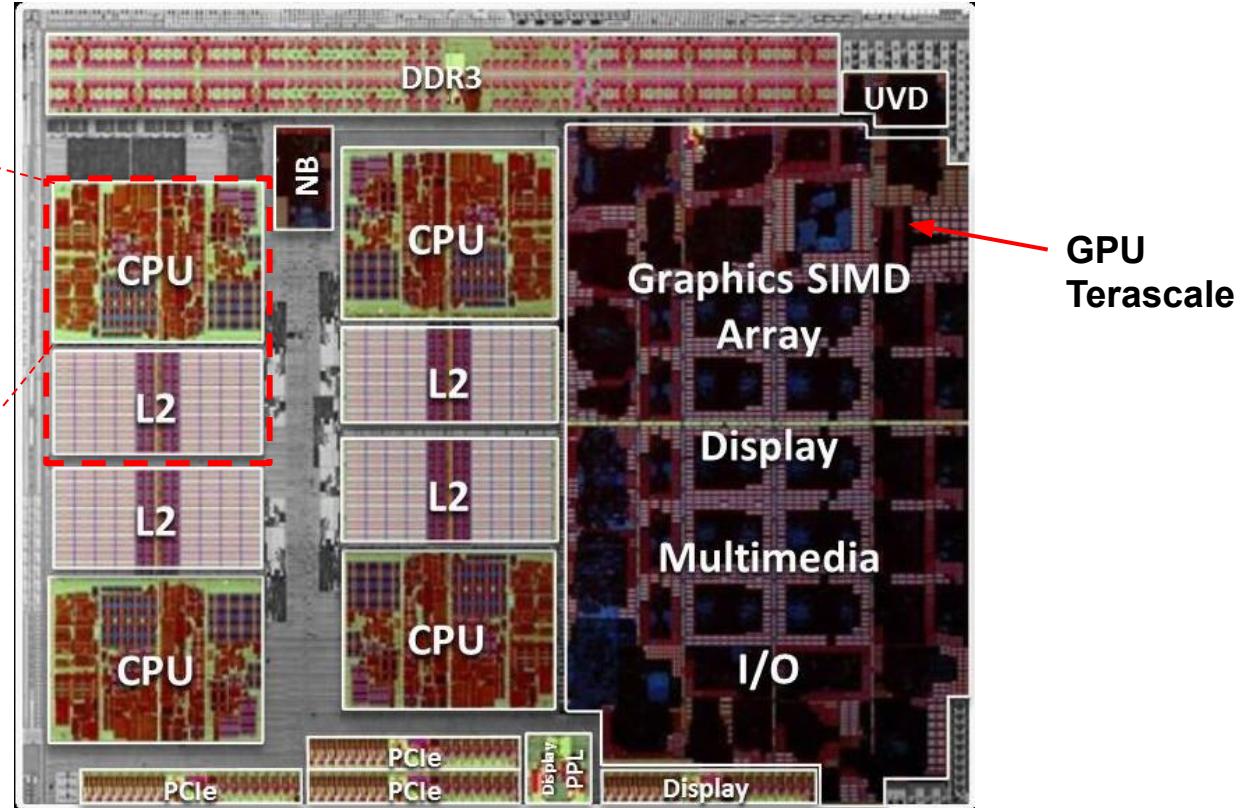
<https://www.amd.com/es/products/graphics/amd-radeon-rx-6950-xt>



## Ejemplo de procesador AMD: Llano APU



CPU microarquitectura k10  
(multi-hilo, superescalar  
fuera de orden)



GPU  
Terascale



## Herramientas de desarrollo

- CUDA: creado por Nvidia (2007).
  - Funciona desde la serie de GPUs Nvidia G8x (2006) en adelante (productos compatibles: <https://developer.nvidia.com/cuda-gpus>).
- OpenCL (Open Computing Language): Lenguaje libre. Pensado para ser utilizado con todas las GPU (AMD, Intel, Nvidia) y también CPU.
  - Creado por Apple. Mantenido por Grupo Khronos.
- Otros ejemplos:

<https://armkeil.blob.core.windows.net/developer/Files/pdf/product-brief/arm-gpu-processor-comparison-table.pdf>

## Conceptos básicos de CUDA

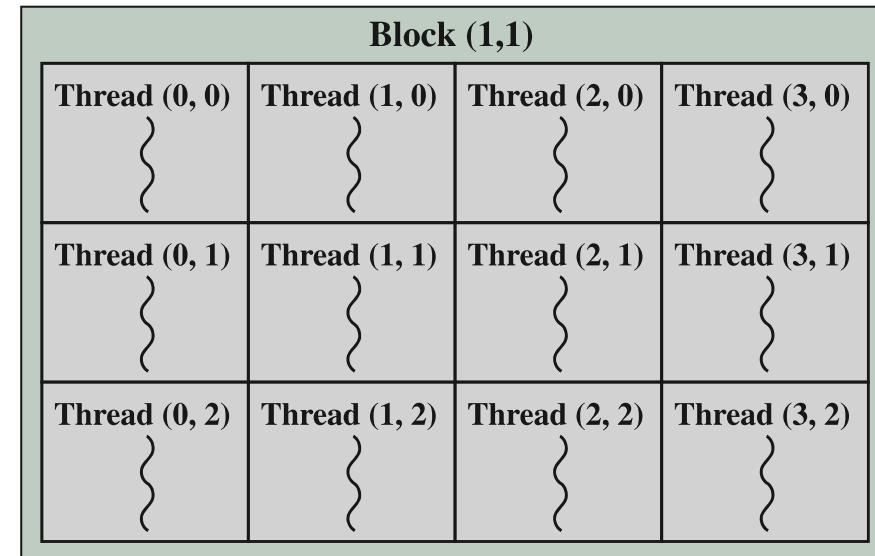
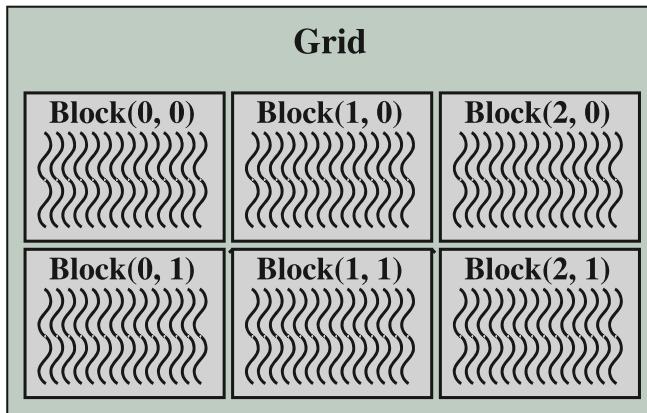
- CUDA (Compute Unified Device Architecture): Modelo de programación creado por NVidia,
  - Lenguaje de programación basado en C++/C.
    - Wrappers para otros lenguajes como Java, Python, etc.
  - Compilador.
  - Herramientas de desarrollo.



## Conceptos básicos de CUDA

- Un programa en CUDA tiene tres partes:
  - Código que corre en el CPU (llamado host).
  - Código que corre en el GPU (llamado device), llamado kernel.
  - Código encargado de la transferencia de datos entre CPU y GPU.
- Kernel: Código de los hilos a paralelizar, agrupados en bloques (blocks).
  - Cada bloque se asigna a un multiprocesador (no se divide).
  - El número total de bloques es llamado grid.
- El programador debe decidir:
  - El número total de hilos que tendrá un kernel.
  - Cómo se agruparán los hilos en bloques.
    - Cada multiprocesador puede aceptar un número máximo de hilos.
    - El número total de bloques debe ser igual o mayor al número total de multiprocesadores (para evitar desaprovechar multiprocesadores).

## Conceptos básicos de CUDA





## Ejemplo de programa en CUDA

```
#include <stdio.h>
/*Librerías necesarias*/
__global__ void sumar_vectores(int *V1, int* V2 , int* V3)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    V3[i]=V1[i]+V2[i]
}
__global__ void sumar_matrices(int *M1, int* M2 , int* M3)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    int j = blockIdx.y*blockDim.y + threadIdx.y;
    int k = blockIdx.z*blockDim.z + threadIdx.z;
    M3[i][j][k]=M1[i][j][k] + M2[i][j][k];
}
```

Global indica que la siguiente función se correrá en paralelo

threadIdx.x indica el número de hilo. Tiene 3 dimensiones.

blockIdx.x indica el número de bloque (3 dimensiones)  
blockDim.x indica el tamaño del bloque



## Ejemplo de programa en CUDA

```
int main(void)
{
/*inicialización matrices*/

cudaMalloc(.....);
cudaMemcpy(.....);           Asignar memoria.  
                             Copiar datos a la GPU.

sumar_vectores<<<Número de bloques, hilos por   Ejecución de los hilos
bloque>>>(V1, V2 , V3);

cudaMemcpy(.....);           Copiar datos a la CPU.
cudaFree(d_count);
```



## Multicore - CPU/DSP

- DSP (Digital Signal Processors)
  - Procesadores que permiten procesamiento rápido de señales digitalizadas inicialmente analógicas (voz, video, etc.).
    - Desplazamiento y suma, multiplicación y suma, transformada rápida de Fourier, comparación, multiplicación de matrices, etc.
    - Unidad de performance: GMACS (Giga Multiply-Accumulate operations per second)
- CPU/DSP: teléfonos celulares, placas de sonido, cámaras digitales, modems, etc.

## Multicore - CPU/DPS (Texas Instruments 66AK2H12)

Multicore Shared  
Memory Controller

Caché L1 y L2  
dedicadas (cada núcleo  
trabaja sobre bloques  
de datos  
independientes).

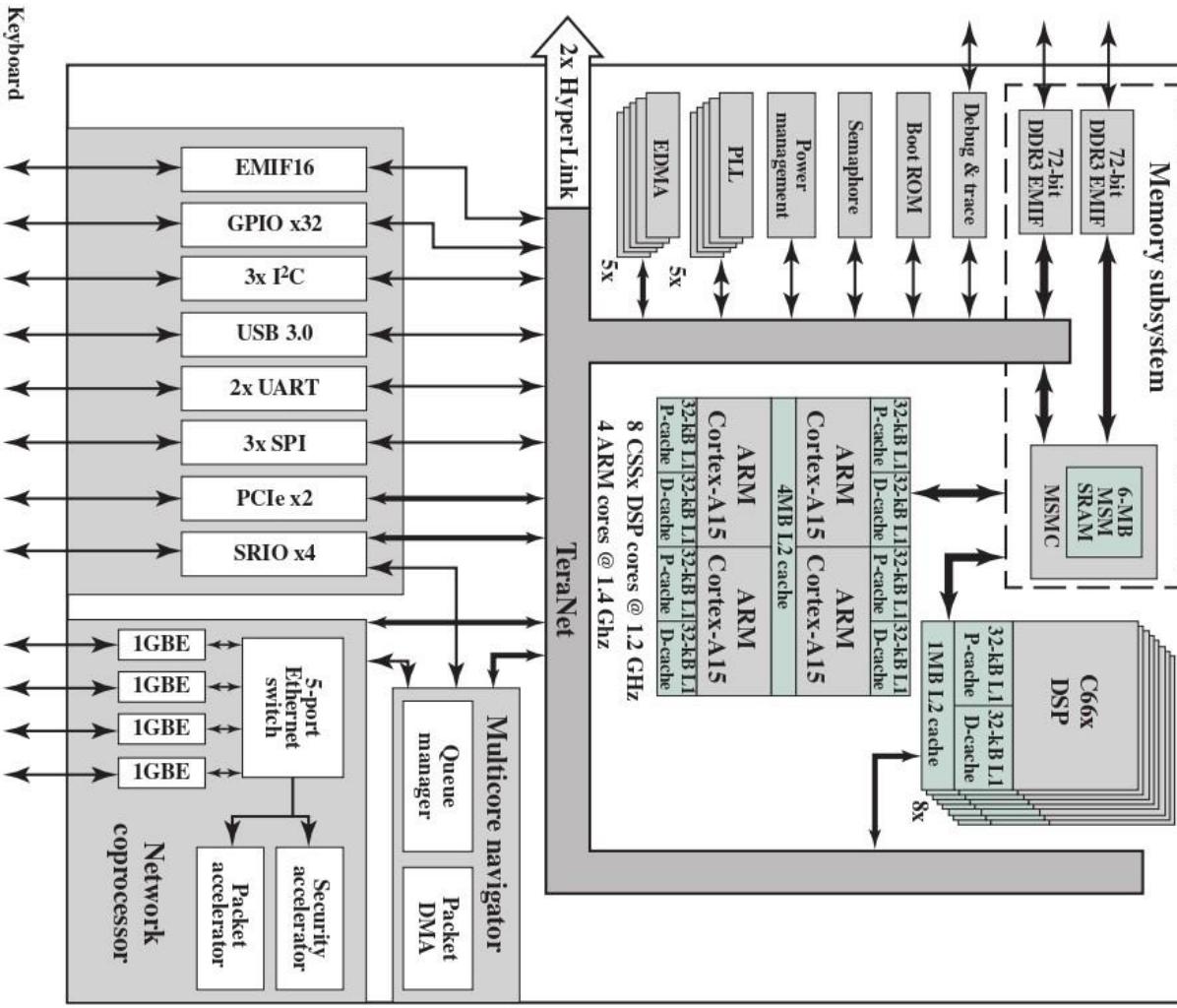


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10<sup>º</sup> edición, página 670



## Núcleo DSP C66x

- 352 GMACS, 198 GFLOPS, 19,600 MIPS.
- 128 bits.
  - SIMD (2 datos de 64 bits, 4 de 32 bits).
  - L1P 32 KB y L1D 32 KB
  - L2 8 MB.

Local power/sleep controller

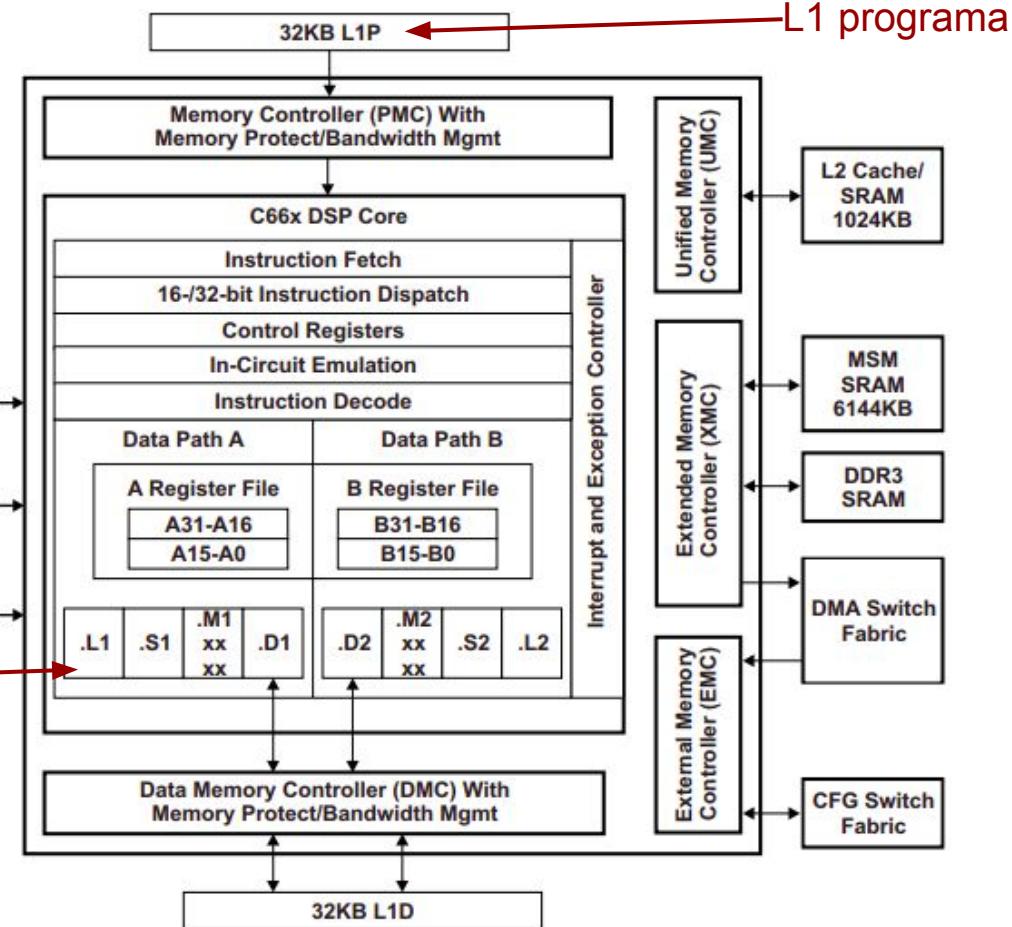
Boot Controller

PLLC

GPSC

Unidades de ejecución

Figura obtenida de “TMS320C66x DSP CPU and Instruction Set”, pag 1-5,  
<https://www.ti.com/lit/ug/sprugh7/sprugh7.pdf?ts=1692211789687>





## Ejemplo: Snapdragon780G

Lanzamiento:  
765: 3/2020  
768: 12/2020  
780: 3/2021

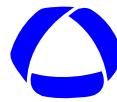
Qualcomm Snapdragon Premium SoCs			
SoC	Snapdragon 765 Snapdragon 765G	Snapdragon 768G	Snapdragon 780G
CPU	1x Cortex-A76 @ 2.3GHz (non-G) @ 2.4GHz (765G)  1x Cortex-A76 @ 2.2GHz  6x Cortex-A55 @ 1.8GHz	1x Cortex-A76 @ 2.8GHz  1x Cortex-A76 @ 2.4GHz  6x Cortex-A55 @ 1.8GHz	<b>1x Cortex-A78</b> @ 2.4GHz  <b>3x Cortex-A78</b> @ 2.2GHz  4x Cortex-A55 @ 1.9GHz
GPU	Adreno 620	Adreno 620  +15% perf over 765G	Adreno 642  +50% perf over 768G
DSP / NPU	Hexagon 696 HVX + Tensor  5.4TOPS AI (Total CPU+GPU+HVX+Tensor)		<b>Hexagon 770</b> Scalar+Tensor+Vector  12TOPS AI (Total CPU+GPU+DSP)
Memory Controller	2x 16-bit CH  @ 2133MHz LPDDR4X / 17.0GB/s		

**DynamIQ**

Neural  
Processin  
g Unit



**TOPS: Tera  
Operations  
Per Second**



<b>Encode/ Decode</b>	2160p30, 1080p120 H.264 & H.265  10-bit HDR pipelines	
<b>Integrated Modem</b>	Snapdragon X52 Integrated  (LTE Category 24/22) DL = 1200 Mbps 4x20MHz CA, 256-QAM UL = 210 Mbps 2x20MHz CA, 256-QAM  (5G NR Sub-6 4x4 100MHz + mmWave 2x2 400MHz) DL = 3700 Mbps UL = 1600 Mbps	<b>Snapdragon X53 Integrated</b>  (LTE Category 24/22) DL = 1200 Mbps 4x20MHz CA, 256-QAM UL = 210 Mbps 2x20MHz CA, 256-QAM  (5G NR Sub-6 4x4 100MHz) DL = 3300 Mbps UL = ? Mbps
<b>Mfc. Process</b>	Samsung 7nm (7LPP)	Samsung 5nm (5LPE)

Cortex A-55: Superescalar de dos vías en orden (sucesor del Cortex A-53)

Cortex A-76: Superescalar de cuatro vías fuera de orden

Cortex A-78: Superescalar de cuatro vías fuera de orden

Fuente: Twitter de Snapdragon



## Sistemas Operativos sobre Procesadores multinúcleo con memoria compartida

Memoria dividida, múltiples estructuras de datos del SO, código del SO compartido

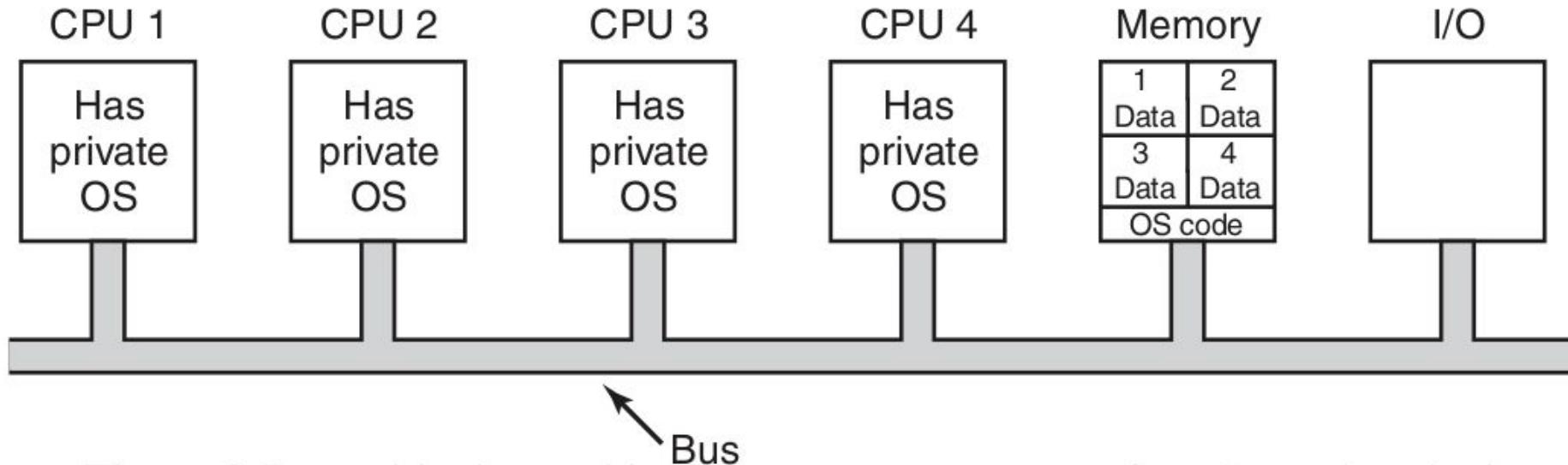


Figura obtenida de: Tanenbaum, Bos, "Modern Operating Systems", 4° edición, página 531.



## **Linux CPU Scheduler para procesadores multinúcleo Múltiples estructuras de datos del SO, código del SO compartido**

- Cada procesador maneja su scheduler.
- Desventajas:
  - Balance de carga ineficiente (un núcleo puede tener exceso de carga y otro estar ocioso).
  - Procesos paralelizables (multihilos) no aprovechan la presencia de varios núcleos.
  - No hay procesos compartidos entre procesadores.
  - Distribución estática e ineficiente de la memoria.
  - No puede haber caché de disco (un bloque podría estar “sucio” en una o varias cachés).
- Dispositivos de I/O compartidos.
- Comunicación a través de memoria compartida.
- Llamadas al sistema capturadas y manejadas por el propio procesador



## Sistemas Operativos sobre Procesadores multinúcleo Sistemas operativos Master-Slave

- El SO, estructura de tablas y scheduler es manejada por un solo procesador: el master.

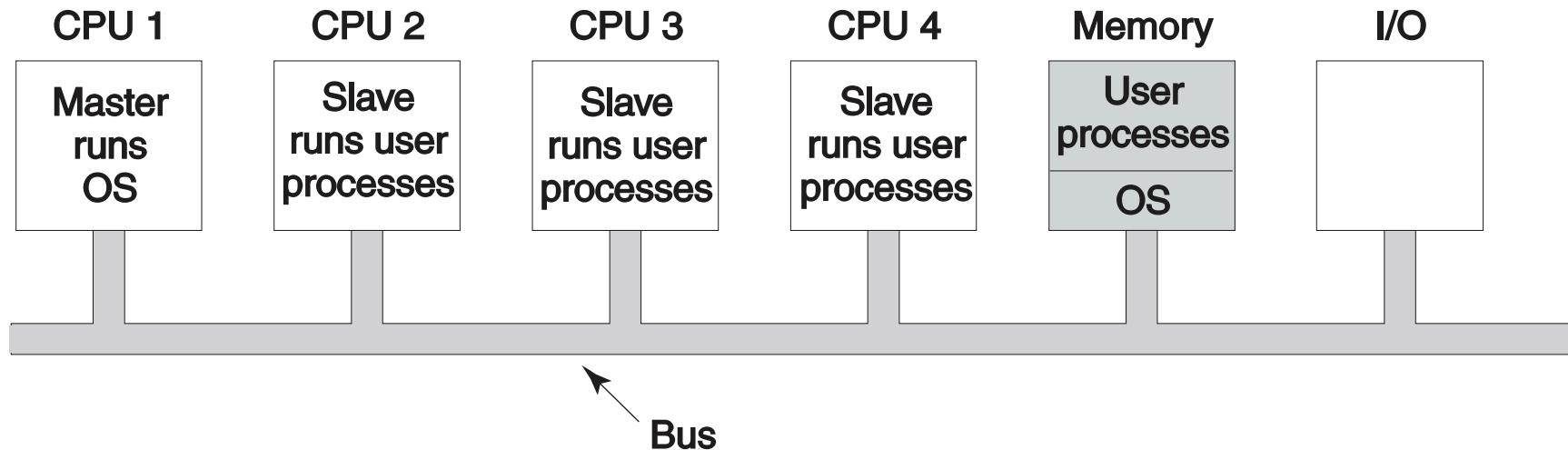


Figura obtenida de: Tanenbaum, Bos, "Modern Operating Systems", 4° edición, página 532.



## Sistemas Operativos sobre Procesadores multinúcleo Multiprocesadores Master-Slave

- Los procesos de usuario pueden correr en cualquier núcleo (incluso el master).
- Los núcleos slaves petitionan al núcleo master procesos para ejecutar a medida que se desocupan.
- Todas las llamadas al sistema son atendidas por el procesador 1.
- Las páginas sobre las que trabaja cada proceso pueden alojarse en cualquier parte de la memoria física.
- **Un solo buffer de caché de disco. No hay problemas de coherencia de caché.**
- Problema: el núcleo master es un cuello de botella.
  - Mientras más núcleos, más llamadas al sistema que son manejadas por el master.
- Actualmente usado por SO que corren en procesadores multi núcleos con pocos núcleos.



## Sistemas Operativos sobre Procesadores multinúcleo SO multiprocesadores simétricos

- Una copia del SO y estructura de datos, cualquier núcleo puede correr el SO.

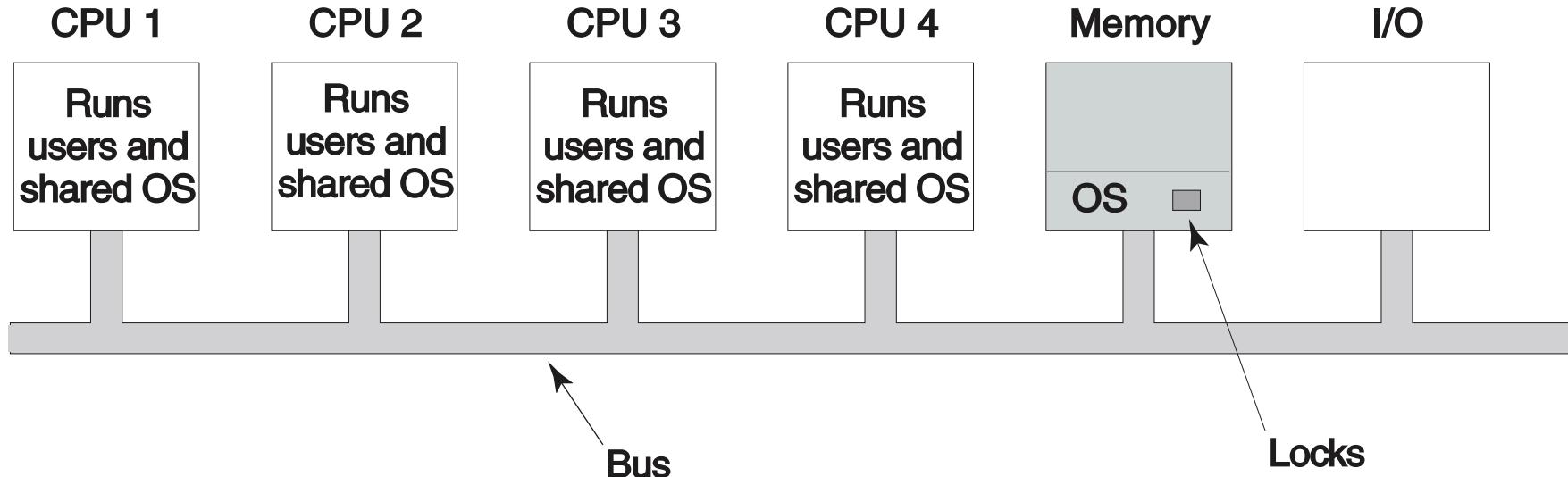


Figura obtenida de: Tanenbaum, Bos, "Modern Operating Systems", 4° edición, página 533.



## Sistemas Operativos sobre Procesadores multinúcleo

- **No hay un núcleo master. No hay cuello de botella por núcleo master.**
- Llamadas al sistema: Son atendidas por el procesador que corre el proceso que produce la llamada.
- Dos modelos:
  - Big kernel lock: asocia un **Mutex** a todo el código del SO (El SO puede ser accedido por **un núcleo a la vez**).
    - Adecuado para sistemas con pocos núcleos (muchos núcleos originarían colas de espera grandes).
  - SO dividido en partes (bloques) independientes. Cada bloque es ejecutado en un núcleo diferente (por ejemplo, 1- scheduler, 2 - gestión del sistema de archivos, 3 - manejo de páginas, pila TCP/IP, etc.).
    - Cada parte del SO posee su mutex (solo un núcleo puede correr el bloque).
    - Puede haber recursos utilizados por varios bloques. Se requiere un mutex para evitar acceso simultáneo.
    - Difícil de diseñar (deadlock).



top - 17:57:44 up 6:30, 1 user, load average: 0,54, 0,83, 0,84  
Tareas: 253 total, 1 ejecutar, 252 hibernar, 0 detener, 0 zombie  
%Cpu(s): 1,8 usuario, 0,7 sist, 0,0 adecuado, 96,7 inact, 0,8 en espera, 0,0 hardw int,  
Mib Mem : 7838,3 total, 1757,2 libre, 2753,3 usado, 3327,8 búfer/caché  
Mib Intercambio: 19073,0 total, 19071,2 libre, 1,8 usado. 4357,9 dispon Mem

PID	USUARIO	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	HORA+	ORDEN	P
285	root	19	-1	298496	211316	209532	S	0,7	2,6	2:15.65	systemd-journal	2
187	root	0	-20	0	0	0	I	0,3	0,0	0:01.10	kworker/2:1H-kblo+	2
14239	root	20	0	0	0	0	I	0,3	0,0	0:01.17	kworker/1:1-events	1
1	root	20	0	169872	12936	8312	S	0,0	0,2	0:02.58	systemd	1
2	root	20	0	0	0	0	S	0,0	0,0	0:00.02	kthreadd	1
3	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_gp	0
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_par_gp	0
5	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	netns	0
9	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	mm_percpu_wq	0
10	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_tasks_rude_	0
11	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_tasks_trace	0
12	root	20	0	0	0	0	S	0,0	0,0	0:00.89	ksoftirqd/0	0
13	root	20	0	0	0	0	I	0,0	0,0	0:18.56	rcu_sched	1
14	root	rt	0	0	0	0	S	0,0	0,0	0:00.16	migration/0	0
15	root	-51	0	0	0	0	S	0,0	0,0	0:00.00	idle_inject/0	0
17	root	20	0	0	0	0	S	0,0	0,0	0:00.00	cpuhp/0	0
18	root	20	0	0	0	0	S	0,0	0,0	0:00.00	cpuhp/1	1
19	root	-51	0	0	0	0	S	0,0	0,0	0:00.00	idle_inject/1	1
20	root	rt	0	0	0	0	S	0,0	0,0	0:00.35	migration/1	1
21	root	20	0	0	0	0	S	0,0	0,0	0:00.75	ksoftirqd/1	1
23	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker/1:0H-even+	1
24	root	20	0	0	0	0	S	0,0	0,0	0:00.00	cpuhp/2	2

Comando top Linux. f: agregar columna. u: elegir usuario.

Ultimo núcleo utilizado

a

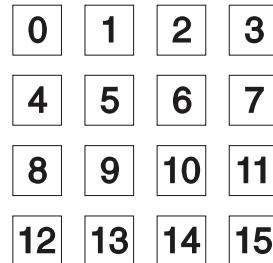


## Multiprocesadores - Cuestiones relacionadas con los SO Schedulers

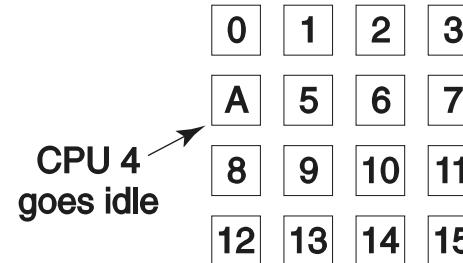
- Los schedulers sobre sistemas multinúcleo son bidimensionales, se debe decidir:
  - Qué tarea (procesos o hilos) ejecutar a continuación.
  - En qué núcleo ejecutar la tarea.
- Los schedulers buscan:
  - Afinidad de tarea con núcleos.
  - Ejecutar grupos de hilos relacionados en grupos de núcleos.
  - Gang Scheduling.

## Multiprocesadores - Cuestiones relacionadas con los SO

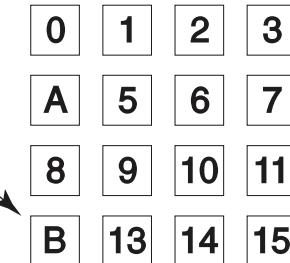
Schedulers compartidos - hilos independientes - una sola tabla de prioridades.



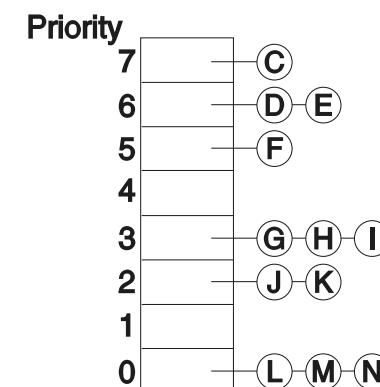
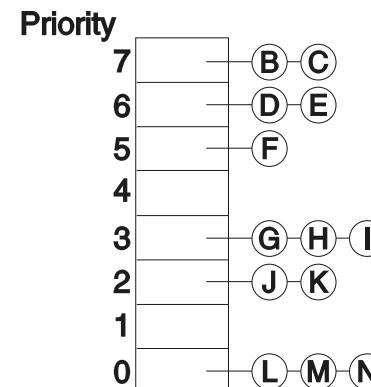
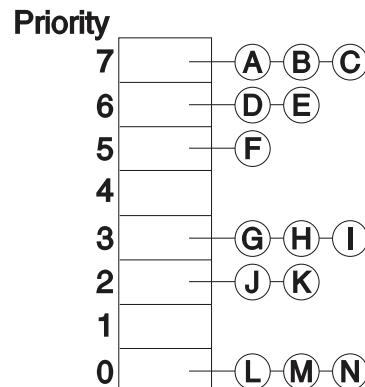
CPU



CPU 4  
goes idle



CPU 12  
goes idle





## Multiprocesadores - Cuestiones relacionadas con los SO Schedulers

- Schedulers de dos niveles (afinidad de hilos con los mismos núcleos).
  - Cuando se crea una tarea, se agenda en un scheduler global.
  - Cuando una tarea pasa a ejecución, se agenda en un scheduler local propio del núcleo en el cual se ejecuta.
  - Los núcleos toman hilos primero de los schedulers locales.
  - Si los núcleos no poseen hilos que ejecutar, toman hilos del scheduler global.
  - La decisión de cuándo tomar una tarea de un scheduler local o de un scheduler global da lugar a distintos algoritmos.
  - Ventaja
    - Se minimiza la competencia entre núcleos para acceder a un scheduler global.
    - Aprovechar la afinidad de caché.
    - Distribución de carga entre núcleos.

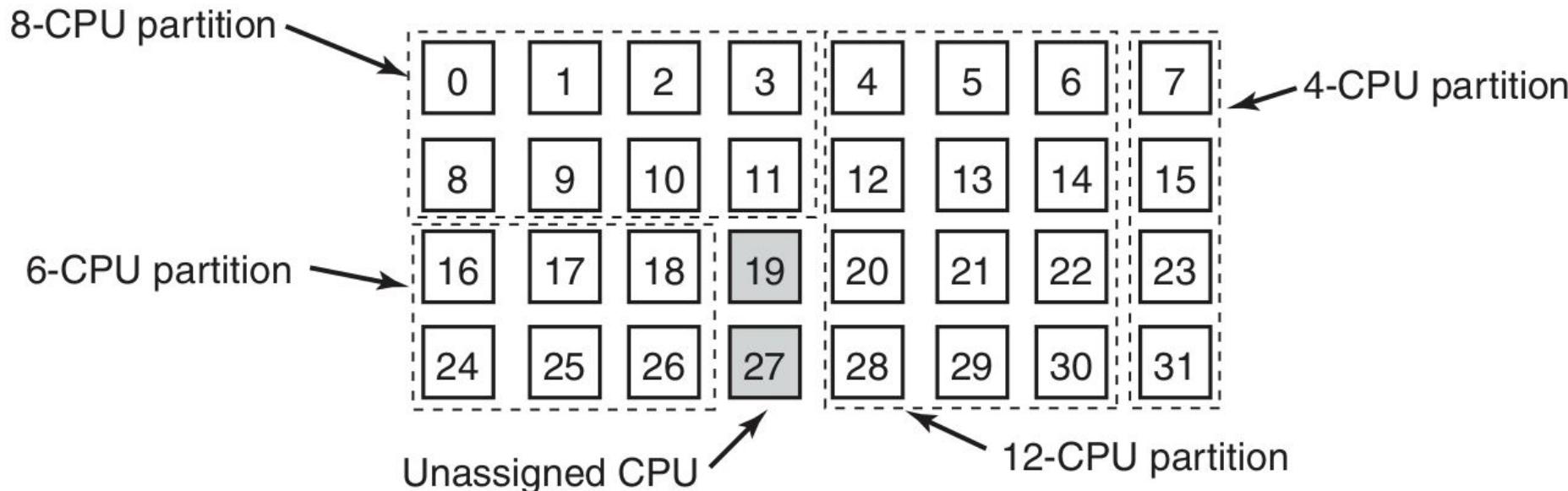


## Multiprocesadores - Cuestiones relacionadas con los SO Schedulers

- La asignación es en base a bloques de hilos. Si los hilos son creados al mismo tiempo (generalmente por un mismo proceso) se asignan a bloques de núcleos.
- Al momento de crear los hilos, el algoritmo busca que haya varios núcleos libres. Cuando están disponibles varios núcleos, los los hilos se asignan al grupo.
- Los núcleos son divididos en grupos, para asignar los grupos de núcleos a hilos de un mismo proceso.
- Las particiones de núcleos pueden cambiar a medida que los procesos entran y salen de ejecución y que la carga total de todos los núcleos trabajo varía.

## Multiprocesadores - Cuestiones relacionadas con los SO Schedulers

Space sharing: agendar varios hilos al mismo tiempo sobre varios CPUs.



Ejemplo de partición de 32 núcleos. La partición es temporal y puede cambiar cuando un grupo de hilos termina su trabajo.

Figura obtenida de: Tanenbaum, Bos, "Modern Operating Systems", 4° edición, página 542



## Multiprocesadores - Cuestiones relacionadas con los SO Schedulers

### Gang Scheduling

- Grupos de hilos relacionados son agendados como una unidad.
- Todos los hilos de la unidad se ejecutan al mismo tiempo en diferentes núcleos.
- Todos los hilos de la unidad comienzan y terminan sus ranuras de tiempo juntas

	núcleos					
	0	1	2	3	4	5
Time slot	A <sub>0</sub>	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	A <sub>4</sub>	A <sub>5</sub>
0	B <sub>0</sub>	B <sub>1</sub>	B <sub>2</sub>	C <sub>0</sub>	C <sub>1</sub>	C <sub>2</sub>
1	D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	E <sub>0</sub>
2	E <sub>1</sub>	E <sub>2</sub>	E <sub>3</sub>	E <sub>4</sub>	E <sub>5</sub>	E <sub>6</sub>
3	A <sub>0</sub>	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	A <sub>4</sub>	A <sub>5</sub>
4	B <sub>0</sub>	B <sub>1</sub>	B <sub>2</sub>	C <sub>0</sub>	C <sub>1</sub>	C <sub>2</sub>
5	D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	E <sub>0</sub>
6	E <sub>1</sub>	E <sub>2</sub>	E <sub>3</sub>	E <sub>4</sub>	E <sub>5</sub>	E <sub>6</sub>
7						

Figura obtenida de: Tanenbaum, Bos, "Modern Operating Systems", 4° edición, página 545

## Interrupciones en procesadores multinúcleos: ARM Generic interrupt controller

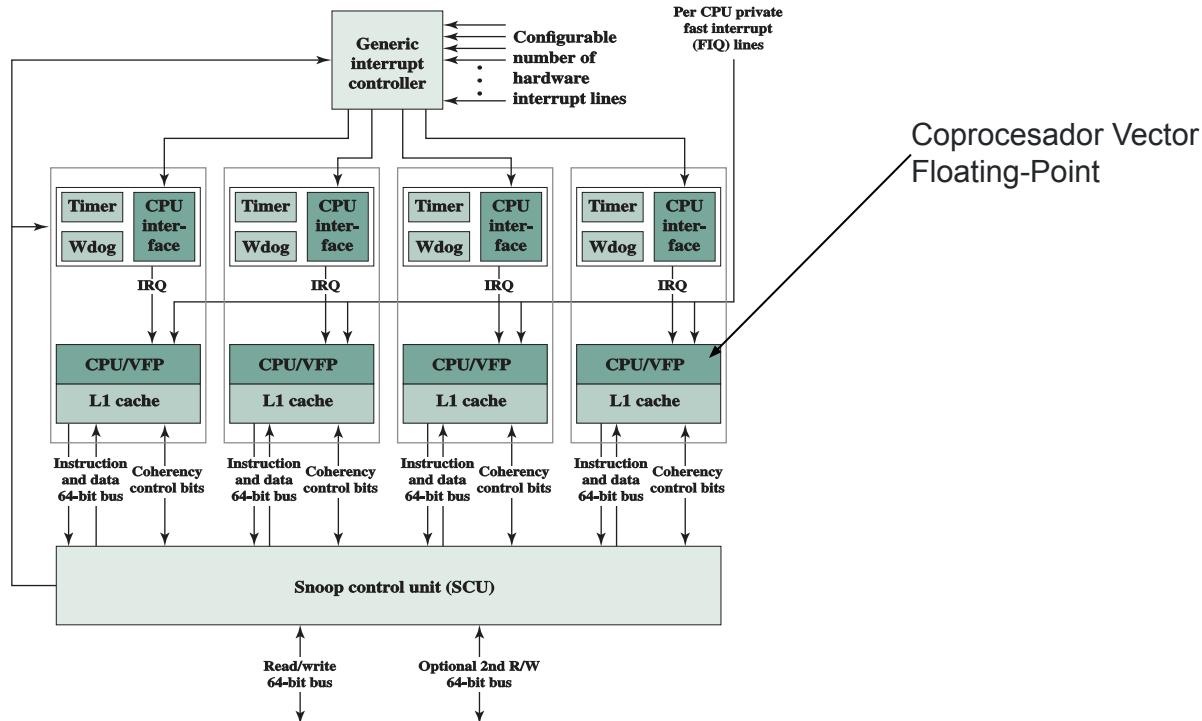


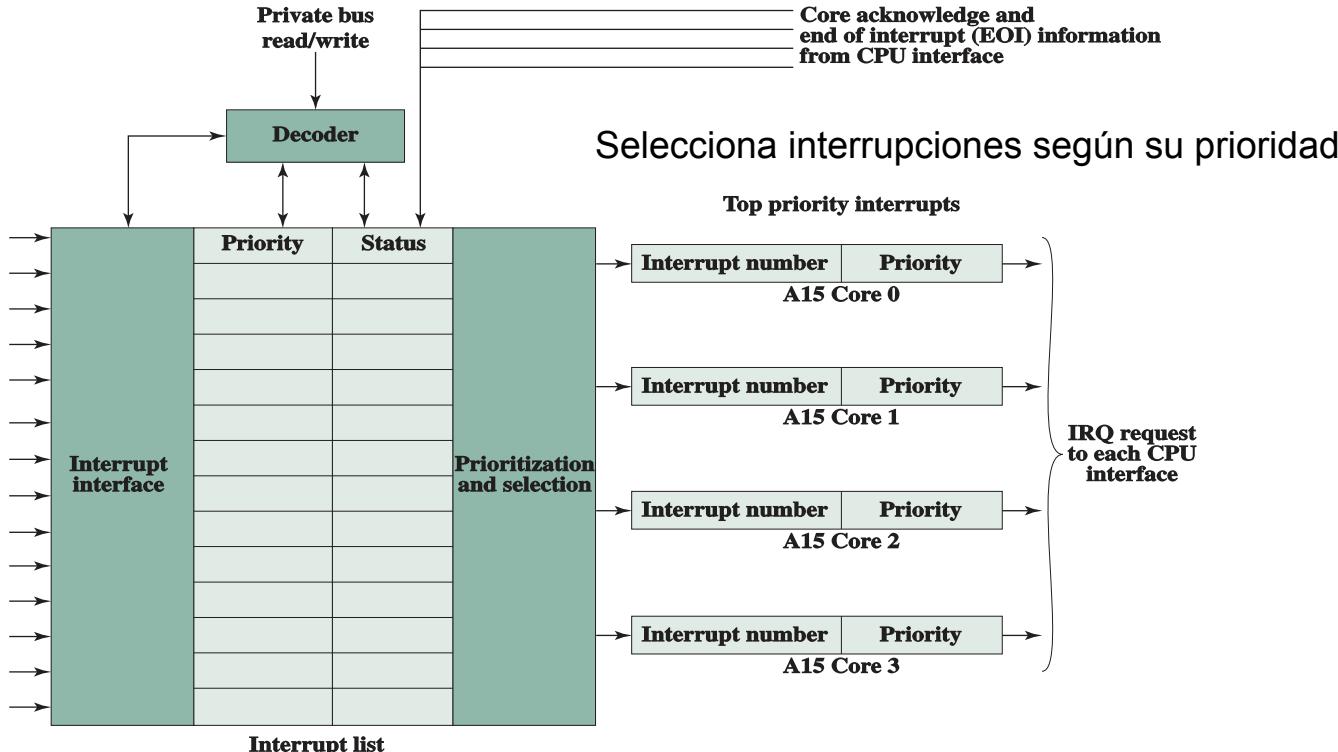
Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 677

## Interrupciones en procesadores multinúcleos: ARM Generic interrupt controller

- CPU Interface: interfaz que permite **debug** (permite detener el procesador, ir paso por paso, analizando el estado de los registros en cada paso).
- Timer: Temporizador, puede generar interrupciones.
- Snoop control unit: Encargado de controlar la coherencia entre cachés.
- Generic interrupt controller: Maneja las interrupciones.
  - Maneja las prioridades y enmascaramiento de interrupciones.
  - Distribuye las interrupciones externas a un procesador, a un grupo o a todos. Espera a que uno la acepte.
    - Una interrupción puede ser dirigida a un núcleo específico o a cualquier núcleo.
  - Contiene información del estado de las interrupciones.
  - Permite a los núcleos generar interrupciones (por software). Utilizado para la comunicación entre hilos de un mismo proceso corriendo en diferentes núcleos (interrupción + mensaje en memoria compartida).

## Interrupciones en procesadores multinúcleos: ARM Generic interrupt controller

Status:  
Inactiva  
Pendiente  
Activa





### **Temas**

Tipos de sistemas paralelos (Clasificación de Flynn)

Paralelismo a nivel de instrucción

Paralelismo a nivel de hilos (Procesadores multihilo simultáneo)

Paralelismo a nivel de núcleos

Arquitecturas multi núcleo simétricas

Arquitecturas multi núcleo heterogéneas con igual ISA o con ISA diferentes

Sistemas CPU-GPU, CPU-DSP. GPGPU. Introducción a CUDA y OpenCL

Sistemas operativos para multiprocesadores.

→ **Paralelismo a nivel de procesos (Cluster)**

Paralelismo a nivel de datos (SIMD)

Sistemas RAID

Performance y escalabilidad

Speedup y eficiencia

Modelos de problemas paralelizables

Paralelismo en el software



## Multicomputadoras

- Problemas de los sistemas multiprocesador:
  - Problemas de **coherencia memoria caché difícil de resolver** (debido a que hay una memoria principal, RAM, compartida).
  - **Costo elevado** (muchos núcleos, memorias cachés, hardware para lidiar con problemas de coherencia, etc).
  - Sistemas operativos complejos (control de bloqueos, schedulers).
  - **Difícil de escalar**, escalabilidad limitada (no se puede agregar núcleos a un chip ya construido).
- **Multi-computadora o Cluster**: Sistema formado por varias **computadoras completas interconectadas** (poseen memoria principal dedicada o memoria distribuida) + **middleware** que permite que resuelvan en conjunto un problema.



## Sistemas multi-computadora

- Parallelismo a nivel de computadoras.
  - Sistemas multicomputadoras **fuertemente acoplados** (cluster)
    - Objetivo: speedup
    - Mucha comunicación entre los procesos.
    - Constructivamente:
      - Las computadoras están **geográficamente cerca** (en una misma sala).
      - **Redes de interconexión específicas** de alta velocidad.
      - **Adaptaciones** en las arquitecturas de la computadoras.
  - Sistemas multicomputadoras **débilmente acoplados** (sistemas distribuidos, cloud computing, grid computing).
    - Objetivo: compartir recursos.
    - Poca comunicación entre los procesos.
    - Constructivamente: distribuidas geográficamente (diferentes partes del planeta).



## Clusters

- Comunicación mediante paso de mensajes (**MPI** por message passing interface).
- Tecnología utilizada para construir supercomputadoras (<https://www.top500.org/>).
- Programación paralela en la cual se ejecutan muchas instancias de un mismo programa en diferentes computadoras (paralelo).
- Dos grandes tipos de clusters según como se construyen:
  - **Beowulf clusters**: computadoras, SO y hardware de red no diseñados para un clúster de alta velocidad.
    - Nodo master (asigna trabajos a los demás nodos y es la interfaz con el usuario) + nodos trabajadores. Todos los nodos corren un middleware.
  - Cluster de hardware especializado o **clusters COW** (cluster of workstation).

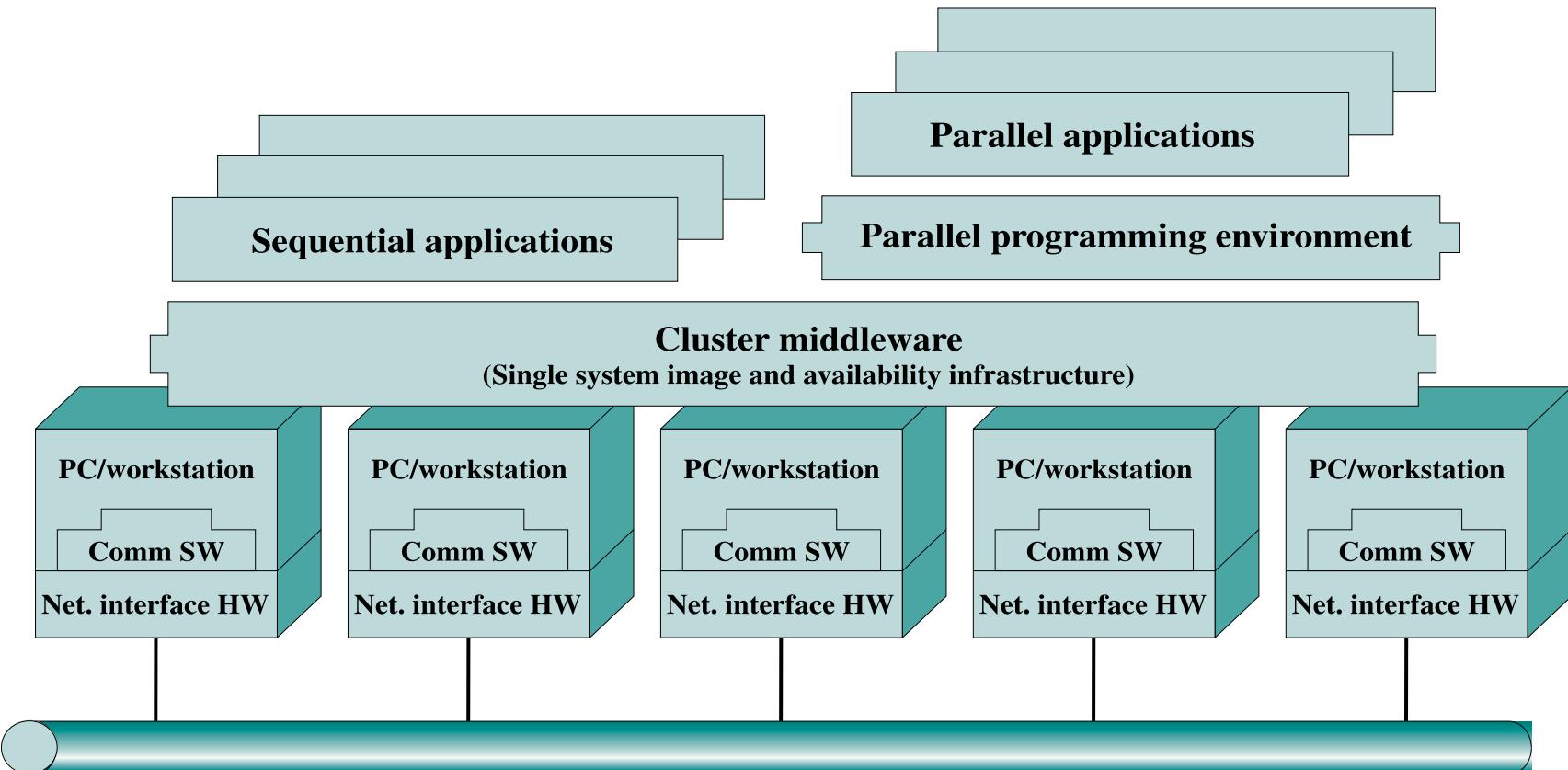
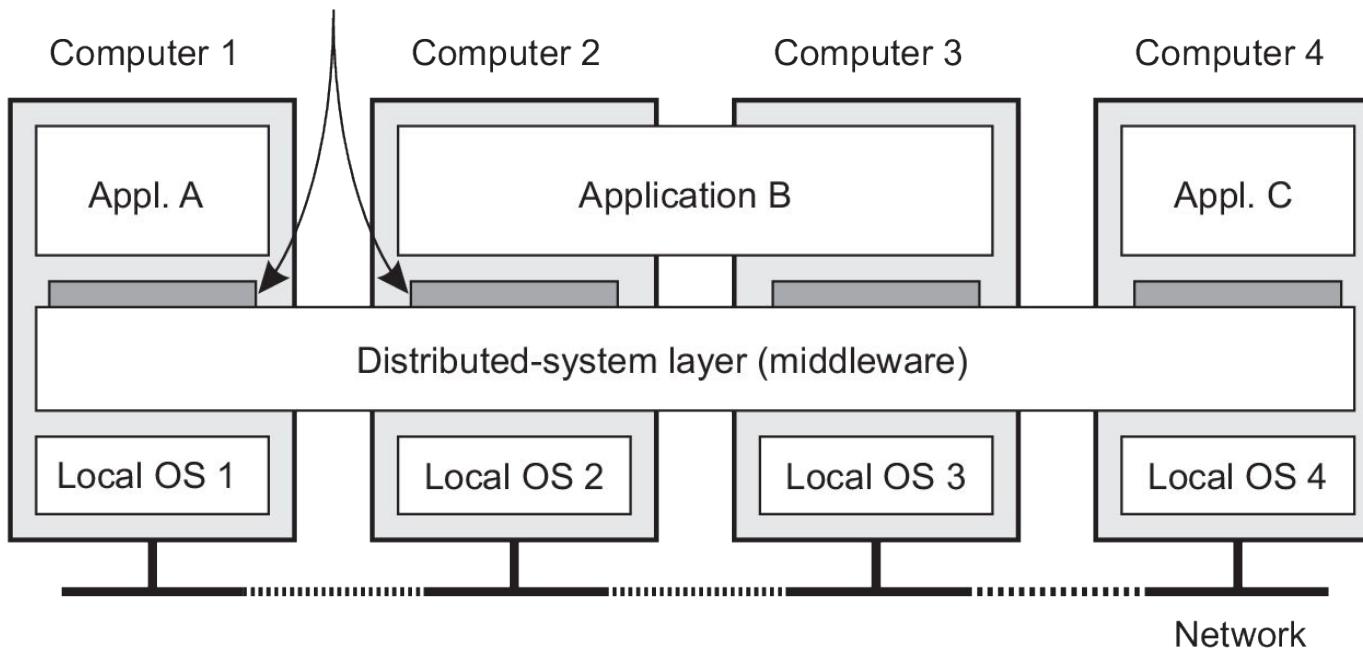


Figura basada en Computer Organization and Architecture 9th Edition William Stallings



## Sistemas distribuidos

Same interface everywhere





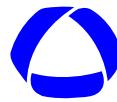
## Cluster Beowulf

- Objetivo: Crear cluster con **computadoras, SO y hardware de red no diseñados para un clúster de alta velocidad.**
  - Computadoras PC o notebook comunes.
  - Comunicación sobre TCP/IP sobre Ethernet o WiFi.
- Basados en PVM y MPI.
  - PVM (Parallel Virtual Machine): herramienta de software diseñada para permitir que computadoras en una red compartan sus recursos (procesador y memoria) y se muestren como un único sistema multiprocesador.
  - MPI (Message Passing Interface): Estándar de bibliotecas para emplearse en programas que se ejecuten sobre varios procesadores (C, C++, Fortran) intercambiando mensajes.



## Cluster Beowulf

- Primer cluster Beowulf (1996): creado en CESDIS (Center of Excellence in Space Data and Information Sciences, financiado por la NASA) con el objetivo de procesar grandes bases de datos.:
  - 16 computadoras con procesador Intel DX4 (familia 80486) de 100 MHz y L1 de 16 KB (no tenía L2).
  - Red Ethernet de 10 Mbps.
  - 1.25 GFLOP.
- Cluster Beowulf NASA 1997:
  - Fusión de dos clusters Beowulf.
  - 199 computadoras procesador P6
  - 10.1 GFLOP.



## Cluster Beowulf

- Existe gran cantidad de herramientas de software para la implementación de cluster Beowulf (sobre todo sobre Linux):
  - Drivers para placas de red.
  - Middleware para Linux.
    - Implementan MPI (OpenMPI, MPICH, Spectrum MPI).
    - Herramientas de gestión.
  - Herramientas para utilizar GPU.

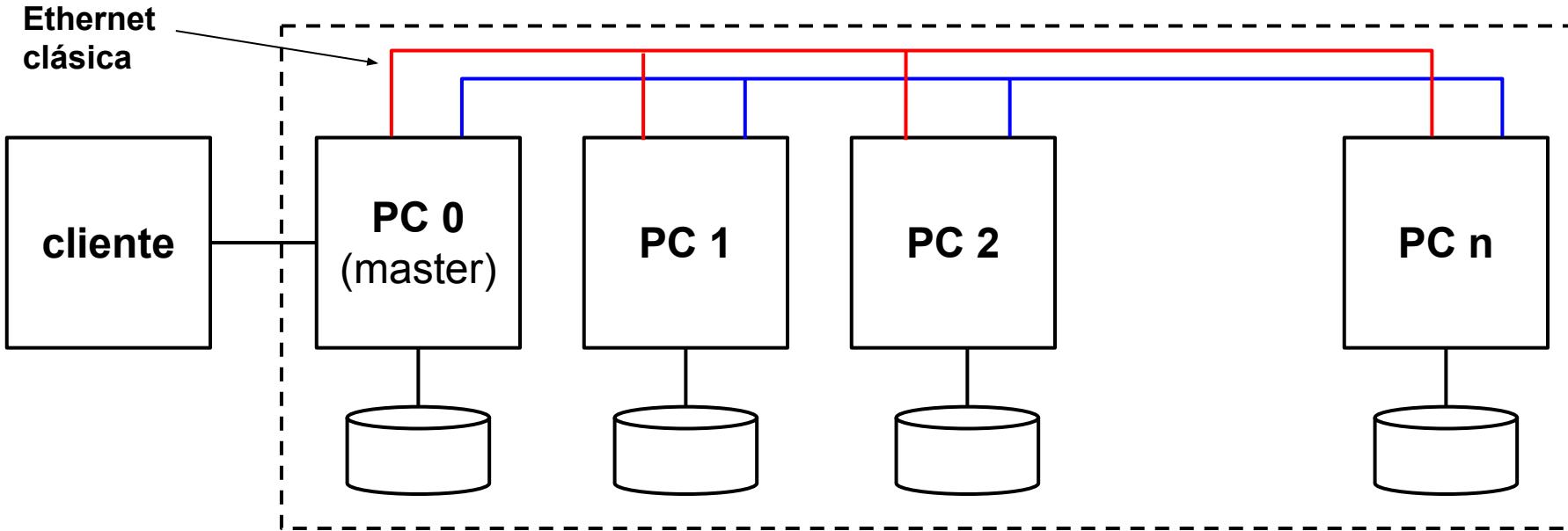


## Cluster Beowulf

- Ventajas:
  - Los componentes son de bajo costo.
  - Es fácilmente replicable.
  - Fácilmente escalable.
    - Es fácil agregar un nuevo nodo.
    - Los nodos pueden ser diferentes.
  - Ningún vendedor posee derechos sobre los productos.
- Desventajas:
  - Menor performance que un clúster construido con hardware específico.
    - Los nodos poseen menor velocidad.
    - Las redes poseen mayor latencia y menor ancho de banda.
  - El tamaño del SO es algo mayor debido al agregado de librerías.



## Cluster Beowulf: Primeras arquitecturas



Puede haber uno o más master, dependiendo el número total de nodos

Figura basada en: Ridge, Becker, Merkey, "Beowulf Harnessing the Power of Parallelism in a Pile-of- PCs"



## Cluster Beowulf: Primeras arquitecturas

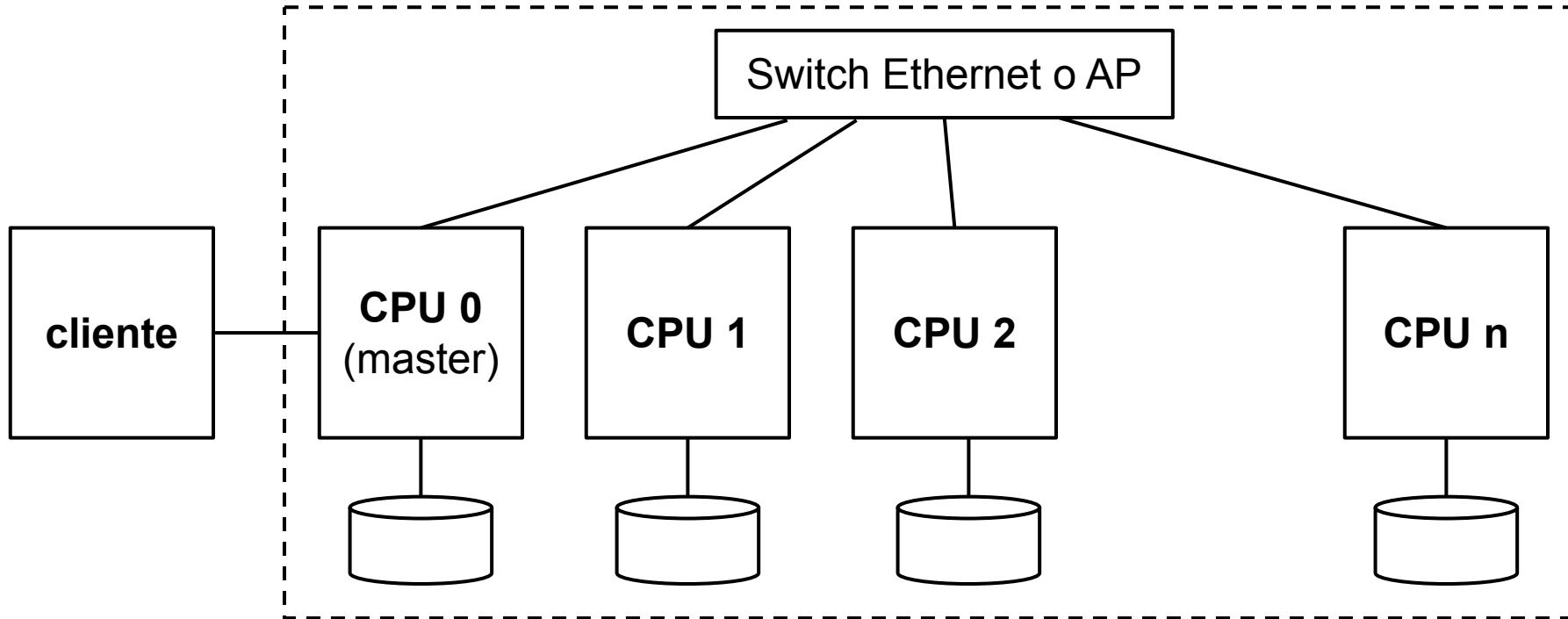


Figura basada en: Ridge, Becker, Merkey, "Beowulf Harnessing the Power of Parallelism in a Pile-of- PCs"



## Cluster Beowulf: Scyld ClusterWare

- Suite (paquete de programas) para la implementación y monitoreo de Clústers.
- Implementa librerías de MPI y herramientas de gestión.
- Implementación basado en web.
- Corre sobre la mayoría de las distribuciones de Linux.
- Permite ver estadísticas de funcionamiento:
  - % de uso de los CPUs
  - Uso de los discos.
  - Porcentaje de uso de la memoria RAM.
- Para configurar un Clúster Beowulf Scyld, debe configurarse una serie de archivos en el nodo master:
  - */etc/Beowulf/config*: Main configuration file
  - */etc/Beowulf/fdisk*: Partición de discos a cargar en los nodos.



## Clúster construido con hardware específico

- Red de interconexión de **alta velocidad**.
- Usualmente las computadoras poseen solo un procesador (multinúcleo), memoria, placas de red y discos duros.
- Topologías de red que garanticen baja latencia.
- RAM de la interfaz de red mapeada en el espacio de memoria del usuario para permitir a los procesos de usuario copiar datos directamente a la RAM de la interfaz.
  - Problema: Distintos procesos que quieren enviar y recibir datos (usual con procesos de usuario y de sistema) accediendo al mismo tiempo a la memoria de la interfaz.
    - Dos interfaces, una para procesos de usuario y otra para el SO.
    - Interfaces de red con distintas colas de datos (para distintos procesos).

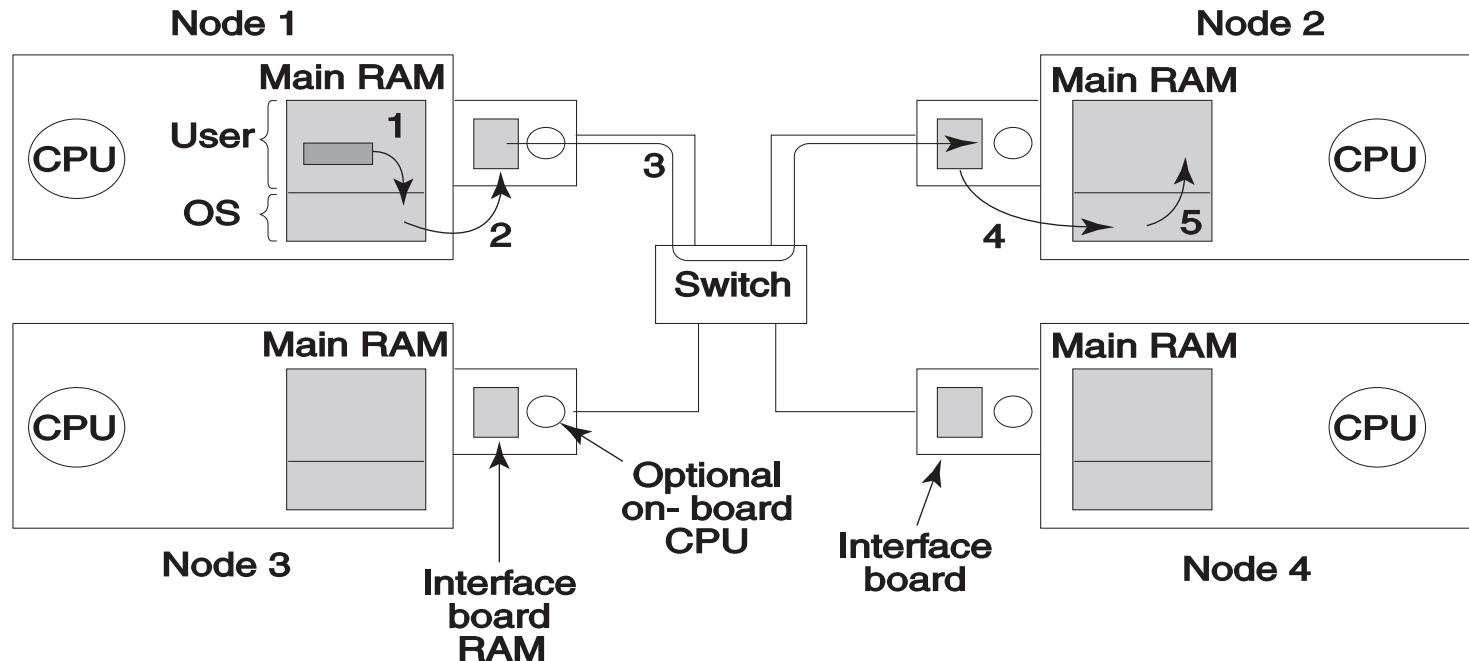


## **Sistemas multi-computadora fuertemente acoplados (Clusters)**

Características de la **interfaz de red** y **SO** de un nodo en un cluster

- Memoria RAM de las interfaces de red de tamaño importante.
  - Si se usara la RAM principal como buffer, se competiría con otros periféricos que acceden al bus, (usualmente PCIe), y no se podría garantizar velocidad cte.
  - La RAM de la tarjeta de red debe recibir el mensaje completo.

## Necesidad de memoria RAM en las interfaces de red.





- 1° cluster más potente del mundo.
- 1.1 HexaFLOPS.
- 21 Mw
- 8.7 millones de núcleos AMD 3rd Generation EPYC 2GHz
- Estados Unidos

- Red de interconexión: HPE Slingshot-11

<sup>1</sup> Datos que varían frecuentemente  
Fuente: <https://www.top500.org/>



## Ejemplo: Fugaku

Licenciatura en Ciencias de la  
Computación



- 2º cluster más potente del mundo.
- 442 PetaFLOPS.
- 29 Mw
- 7.6 millones de núcleos A64FX 48C 2.2GHz.
- ARMv8.2-A (primer ARM de 64 bits)
- Japón

- 5.09 PB de memoria.
- Red de interconexión: Infiniband

<sup>1</sup> Datos que varían frecuentemente  
Fuente: <https://www.top500.org/>



## Ejemplo: Serafin

- Cluster más potente de argentina
- 156 Teraflops
- Centro de Cómputo de Alto Desempeño (CCAD), Córdoba
- 60 nodos.
- 3840 núcleos (120 procesadores AMD EPYC 7532).
- <https://ccad.unc.edu.ar/equipamiento/cluster-serafin/>





## Ejemplo: TOKO

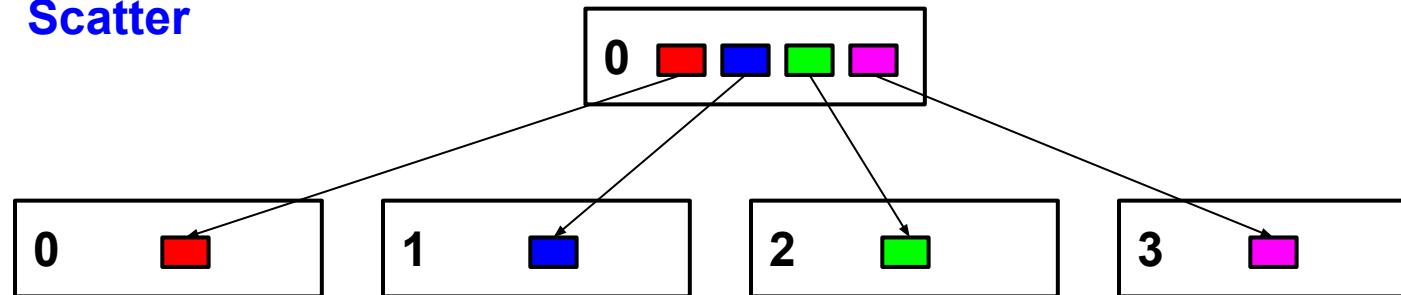
- Cluster más potente del oeste argentino
- ITIC y FCEN, UNCuyo
- 390 núcleos (procesadores AMD de diferentes modelos y GPGPUs NVidia)
- 743 GB de Memoria.
- <http://toko.uncu.edu.ar/>



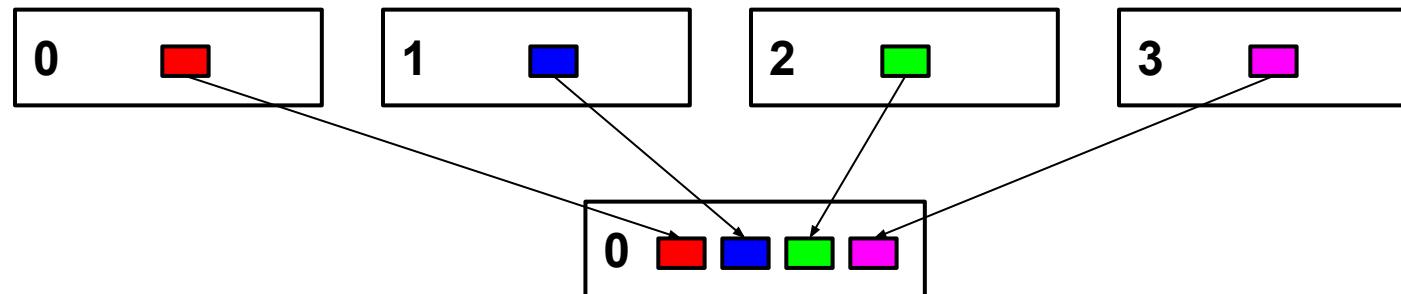


## MPI: Primitivas scatter y gather (esparcir y reunir)

- **Scatter**



- **Gather**





### Temas

Tipos de sistemas paralelos (Clasificación de Flynn)

Paralelismo a nivel de instrucción

Paralelismo a nivel de hilos (Procesadores multihilo simultáneo)

Paralelismo a nivel de núcleos

Arquitecturas multi núcleo simétricas

Arquitecturas multi núcleo heterogéneas con igual ISA o con ISA diferentes

Sistemas CPU-GPU, CPU-DSP. GPGPU. Introducción a CUDA y OpenCL

Sistemas operativos para multiprocesadores.

Paralelismo a nivel de procesos (Cluster)

→ **Paralelismo a nivel de datos (SIMD)**

Sistemas RAID

Performance y escalabilidad

Speedup y eficiencia

Modelos de problemas paralelizables

Paralelismo en el software



## Procesadores Vectoriales y SIMD

Primeros  
procesadores  
vectoriales

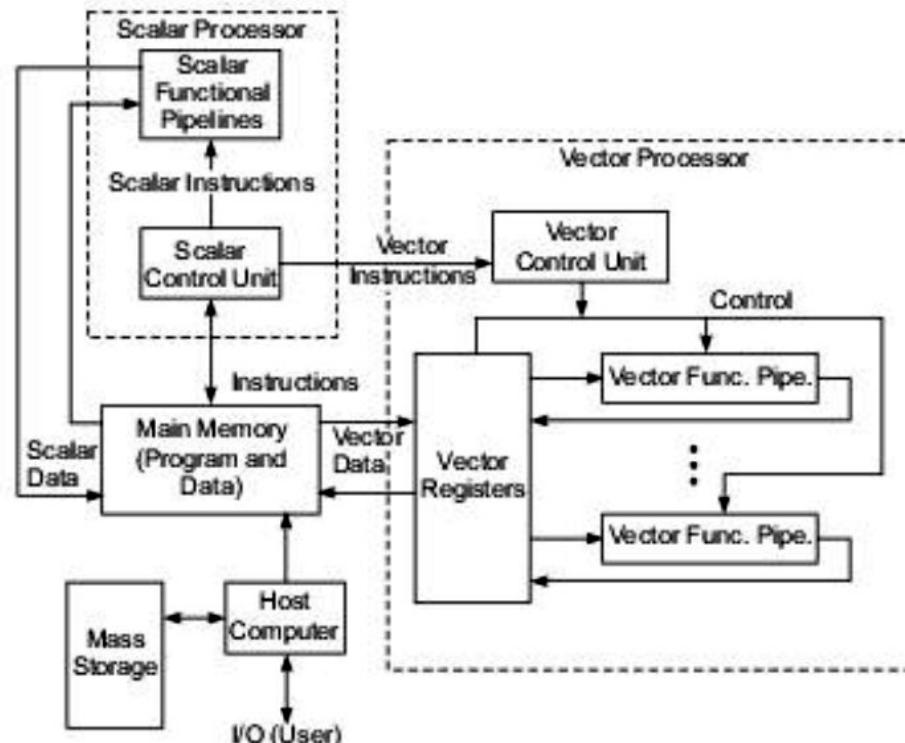


Figura obtenida de Kai Hwang, "Advanced Computer Architecture Parallelism Scalability Programmability", edición 2001, página 25

## Procesadores Vectoriales y SIMD

Procesadores  
vectoriales  
actuales (SIMD)

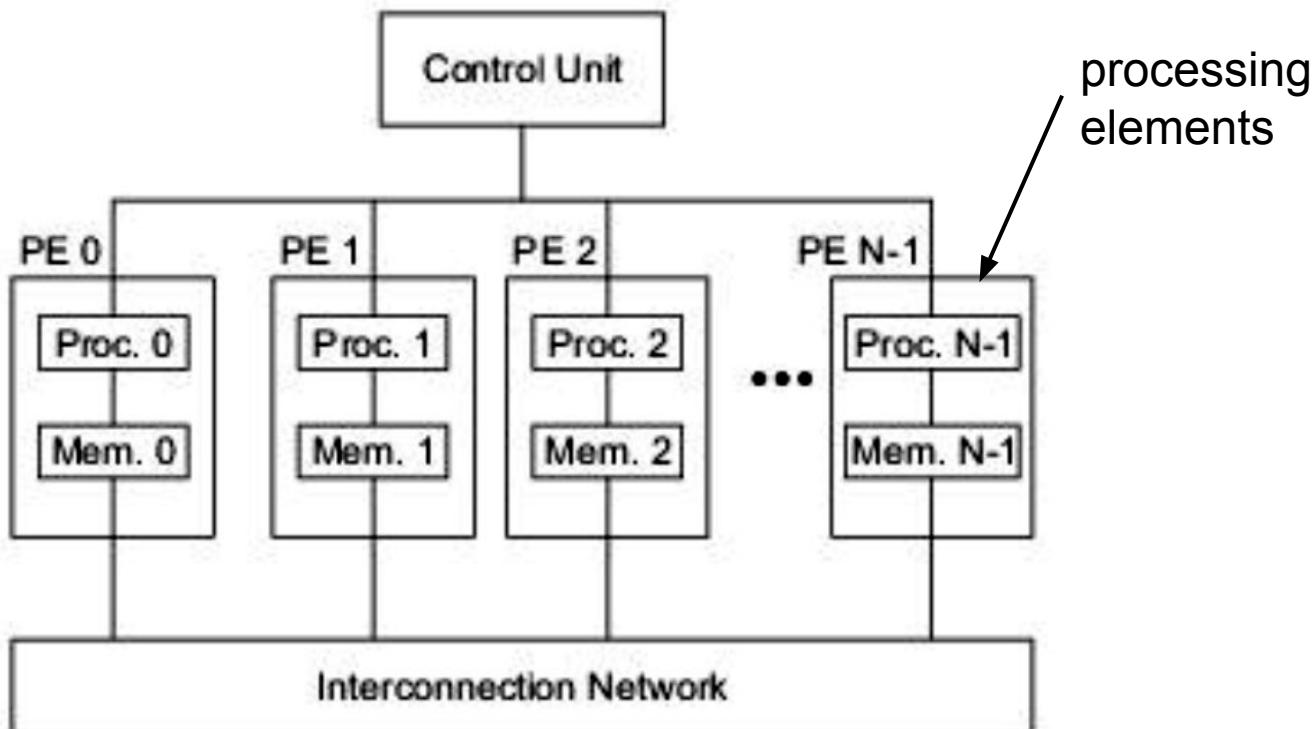
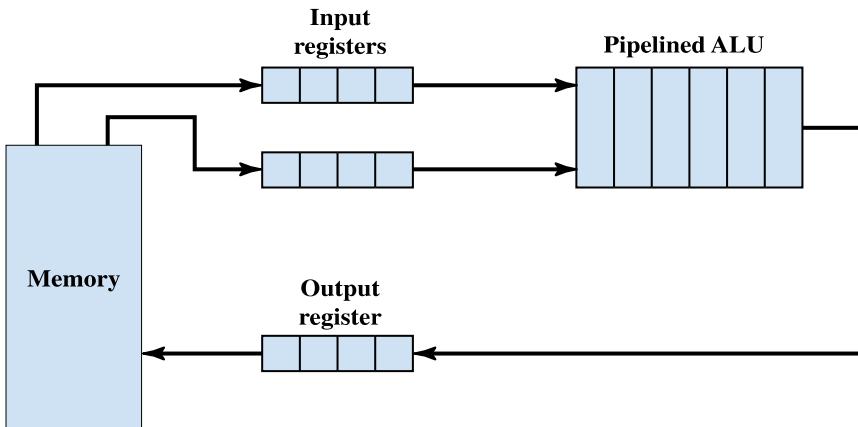


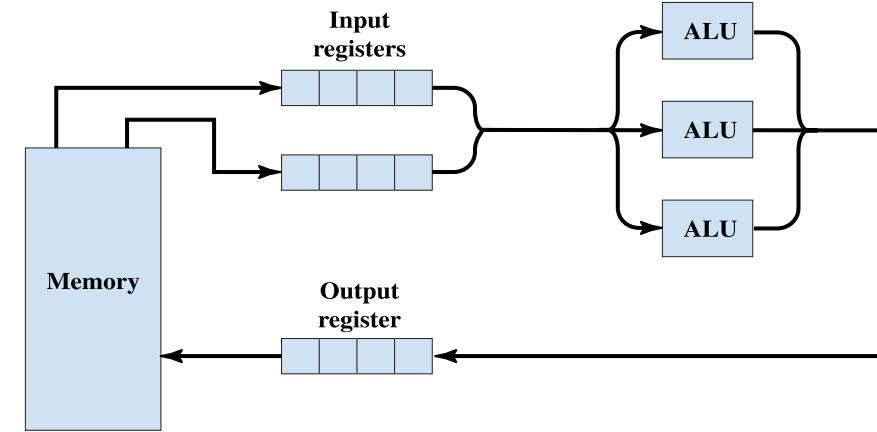
Figura obtenida de Kai Hwang, "Advanced Computer Architecture Parallelism Scalability Programmability", edición 2001, página 27



## Procesadores Vectoriales y SIMD: Implementaciones



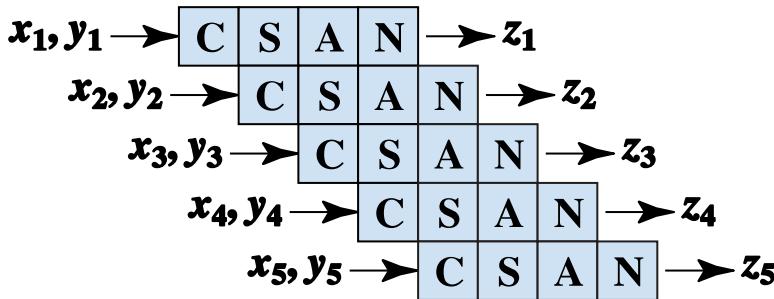
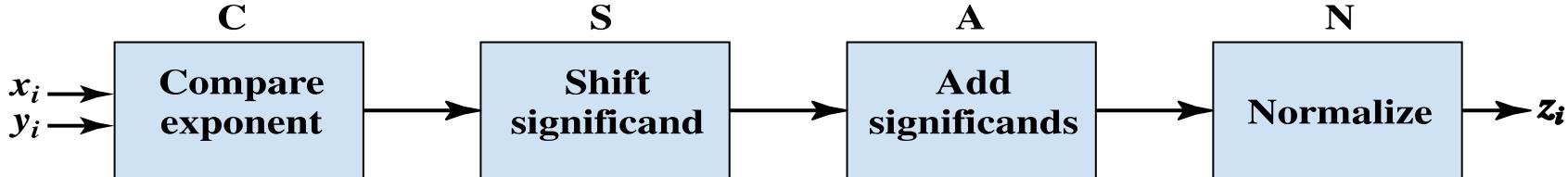
**Pipelined ALU**



**Parallel ALUs**



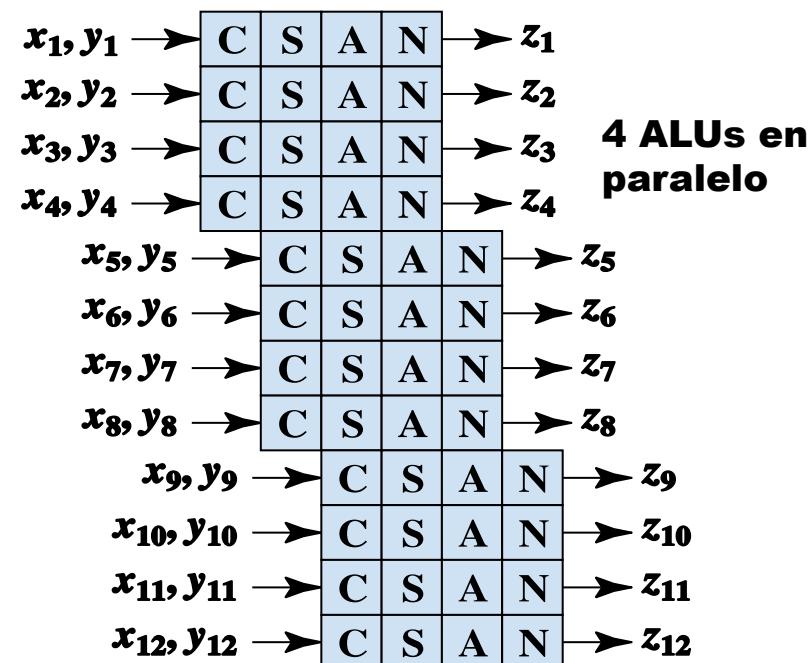
## Ejemplo implementación suma punto flotante



**Pipelined ALU**

**Nota: los vectores pueden tener cientos de componentes. Se operan en paralelo grupos de componentes según la ALU disponible.**

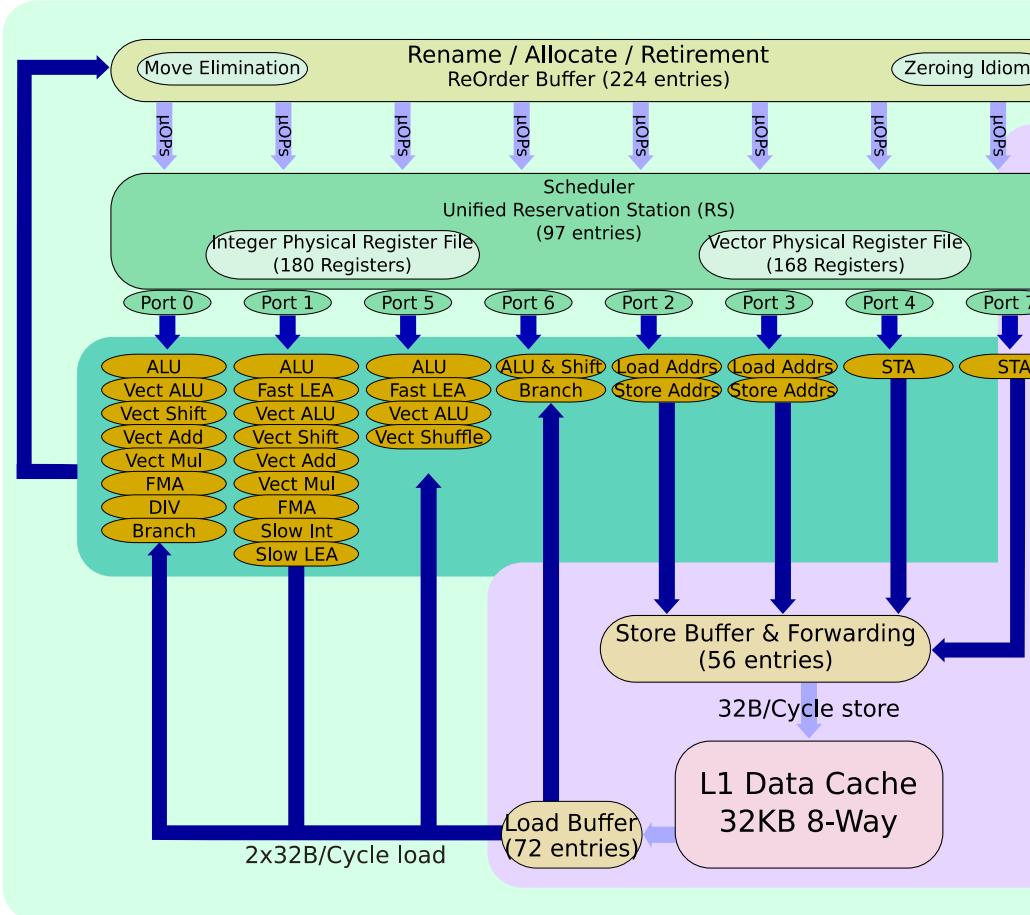
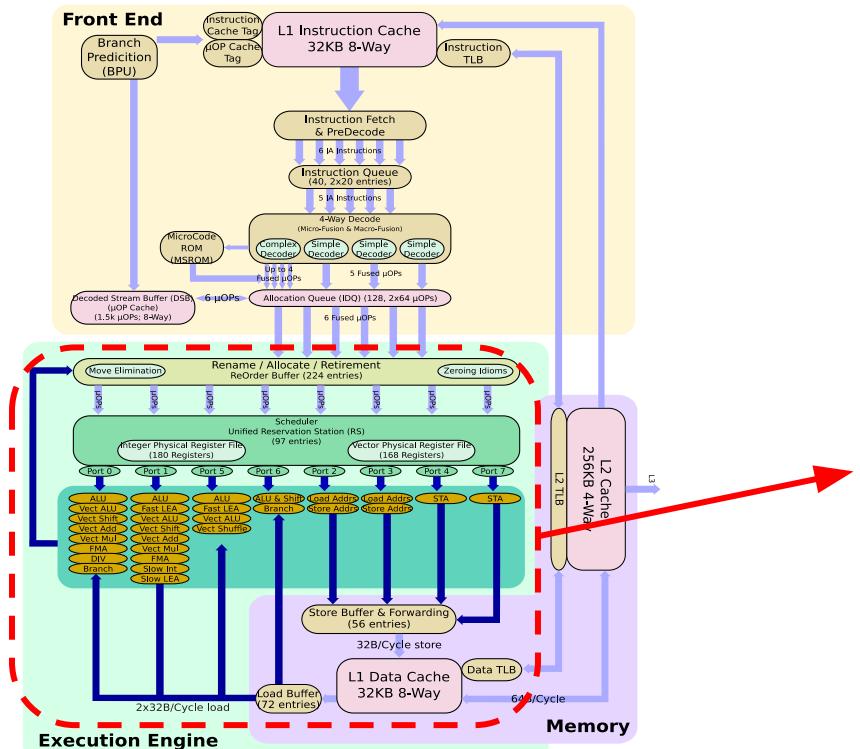
Figura basada en Computer Organization and Architecture 9th Edition William Stallings





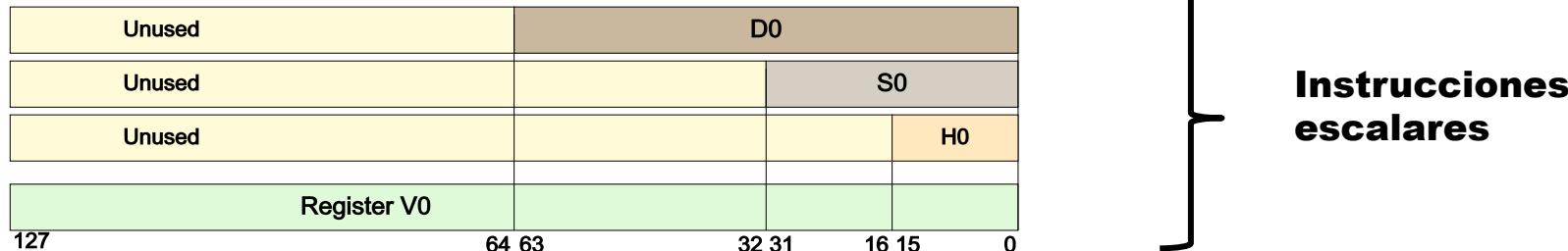
## Computación vectorial: Ejemplo: extensión SSE de x86

- SSE (Streaming SIMD Extensions). 1999, mejora a la extensión MMX.
- Ultima versión: SSE 4 (2006)
- Agrega 70 instrucciones (la mayoría de punto flotante) y nuevos registros (conjunto de registros XMM)
- Pensado para procesamiento digital de imágenes y procesamiento gráfico.
- Ejemplo de operaciones:
  - ADDPS (Add Packed Single-Precision Floating-Point Values) (argumentos:  $\text{DEST}[31:0] \leftarrow \text{SRC1}[31:0] + \text{SRC2}[31:0]$ )



## Ejemplo procesadores SIMD: Extensión NEON de ARM

- Extensión NEON en ARM-64 bits: 32 registros de 128 bits (V0 a V31) (utilizados también por instrucciones escalares con números de punto flotante)
- Extensión NEON en ARM-32 bits: Se combinan registros de 64 bits para formar registros de 128 bits que se comportan como la extensión NEON para ARM-64 bits.





## Ejemplo procesadores SIMD: Extensión NEON de ARM

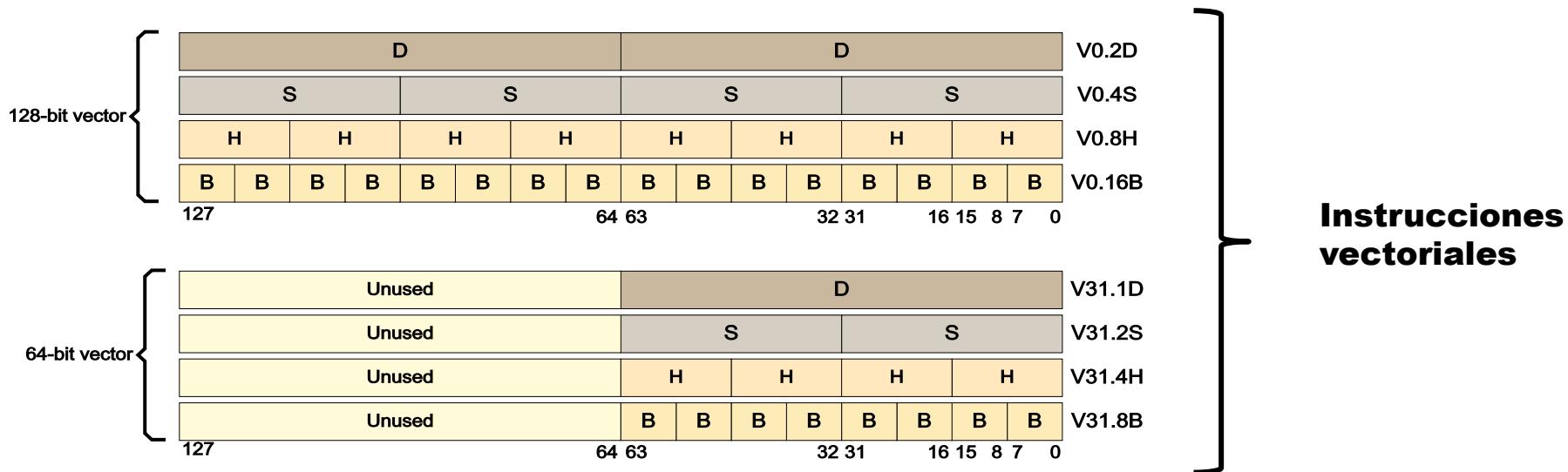
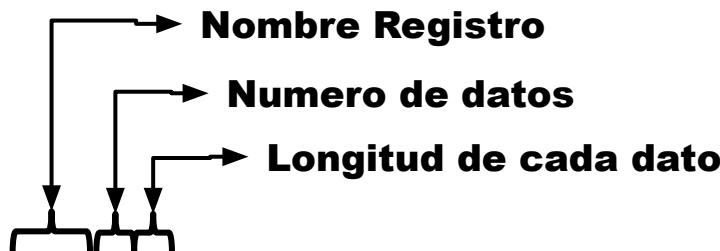


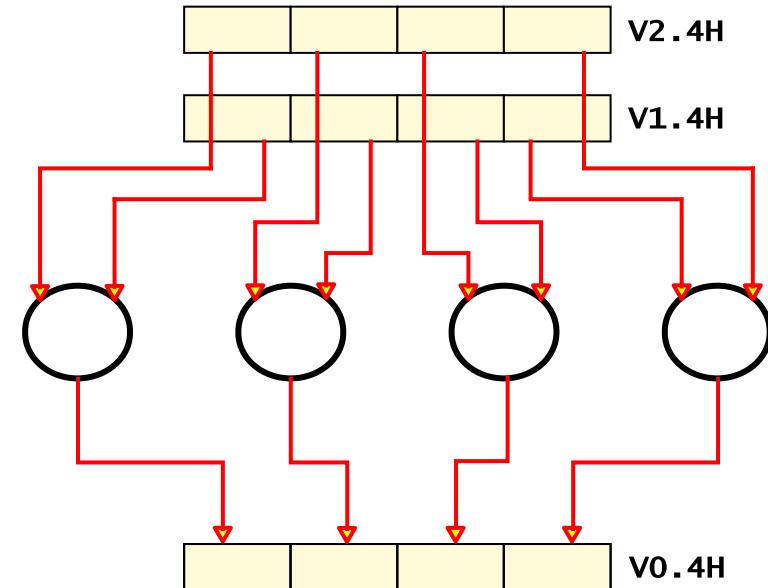
Figura obtenida de ARM Cortex -A Series, Programmer's Guide for ARMv8-A, Version: 1.0, pag 4-17 a 4-20



## Ejemplo procesadores SIMD: Extensión NEON de ARM

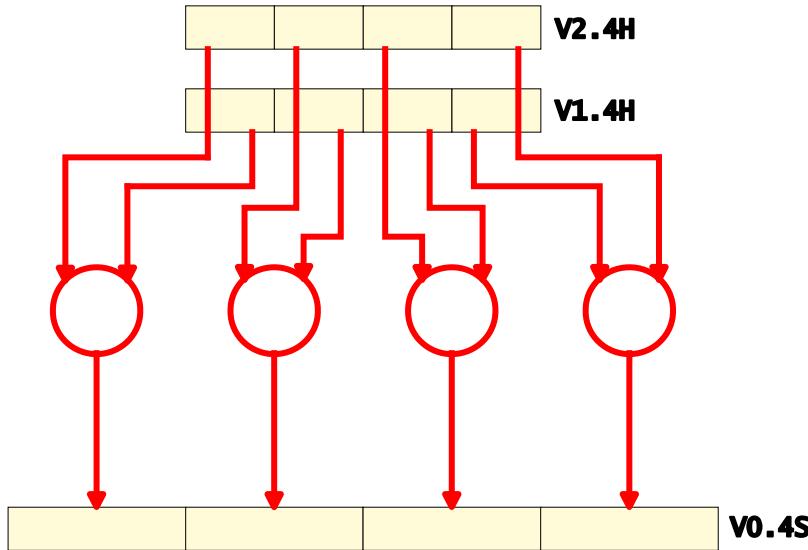


**ADD V0.4H, V1.4H, V2.4H**  
**(V0=V1+V2)**

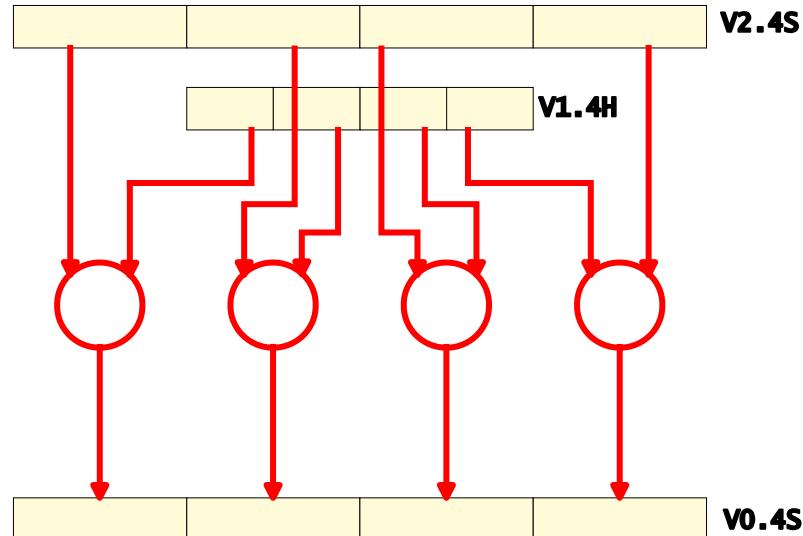




## Ejemplo procesadores SIMD: Extensión NEON de ARM



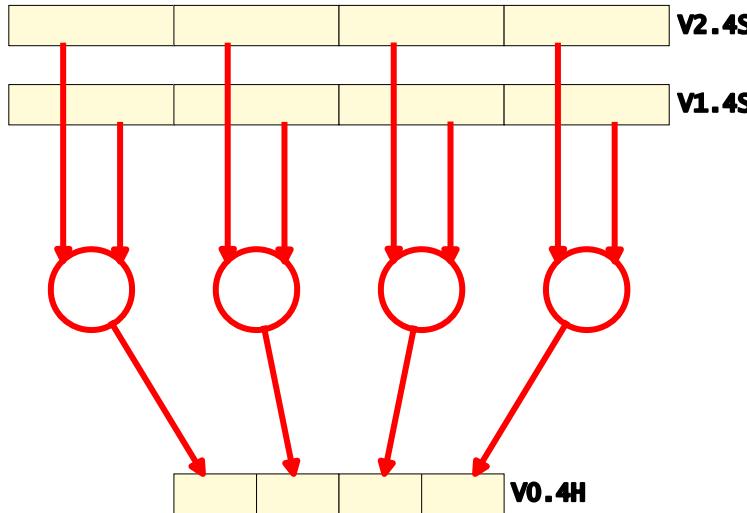
**SADDL V0.4S, V1.4H, V2.4H**



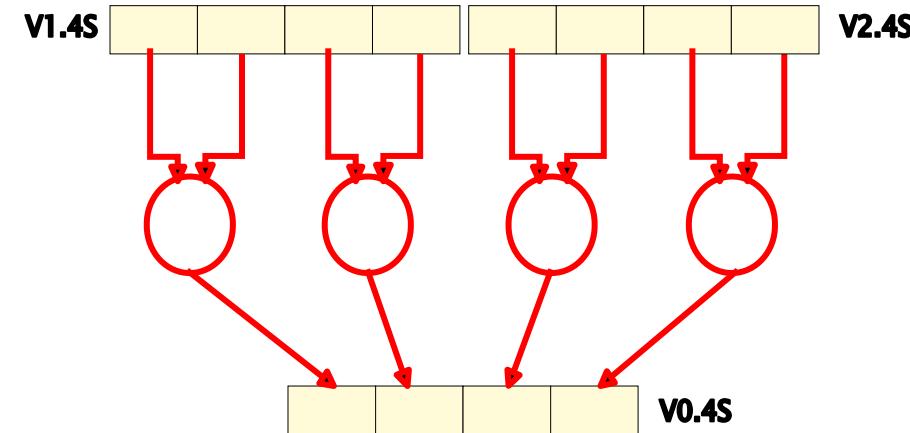
**SADDW V0.4S, V1.4H, V2.4S**



## Ejemplo procesadores SIMD: Extensión NEON de ARM



**SUBHN V0.4H, V1.4S, V2.4S**



**ADDP V0.4S, V1.4S, V2.4S  
(pairwise operations)**



### **Temas**

Tipos de sistemas paralelos (Clasificación de Flynn)

Paralelismo a nivel de instrucción

Paralelismo a nivel de hilos (Procesadores multihilo simultáneo)

Paralelismo a nivel de núcleos

Arquitecturas multi núcleo simétricas

Arquitecturas multi núcleo heterogéneas con igual ISA o con ISA diferentes

Sistemas CPU-GPU, CPU-DSP. GPGPU. Introducción a CUDA y OpenCL

Sistemas operativos para multiprocesadores.

Paralelismo a nivel de procesos (Cluster)

Paralelismo a nivel de datos (SIMD)

→ **Sistemas RAID**

Performance y escalabilidad

Speedup y eficiencia

Modelos de problemas paralelizables

Paralelismo en el software



## Sistemas multi-computadora - Sistemas RAIDs

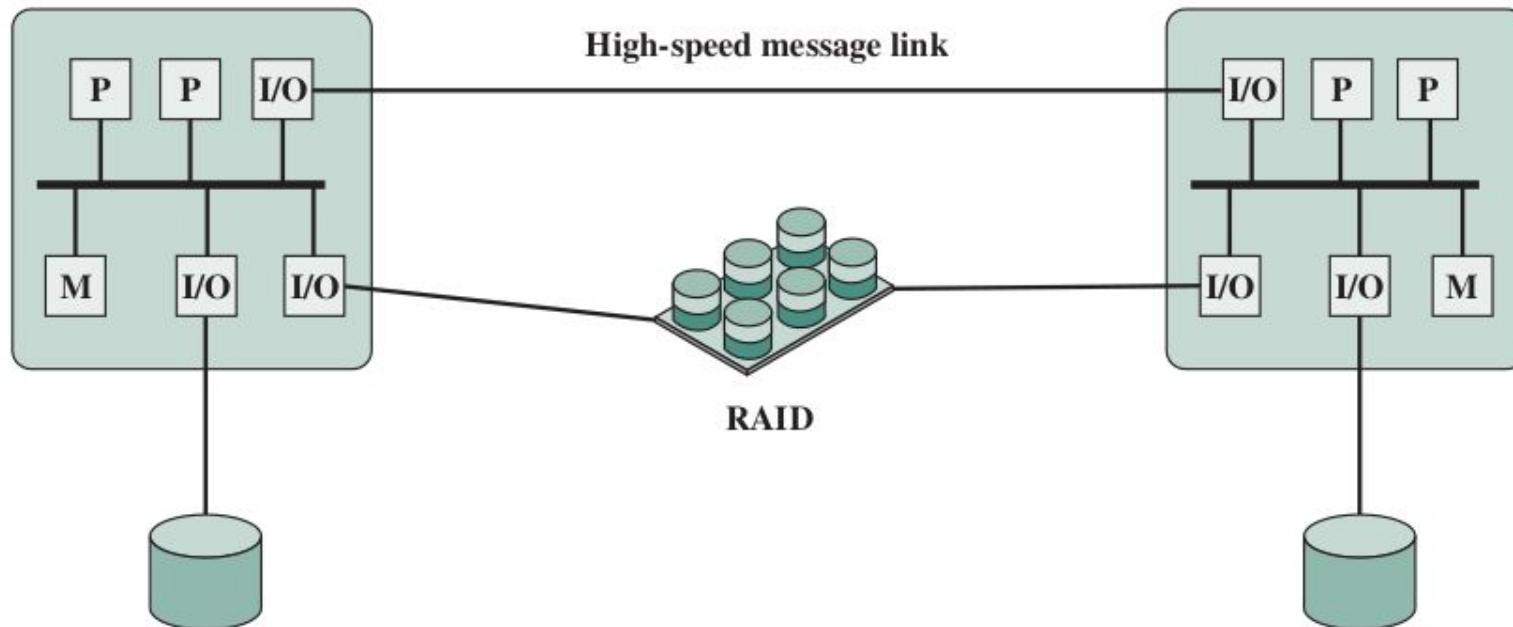


Figura basada en Computer Organization and Architecture 10th Edition William Stallings, pag. 634

Primera publicación: David A. Patterson, Garth Gibson, and Randy H. Katz. 1988. A case for redundant arrays of inexpensive disks (RAID). SIGMOD Rec. 17, 3 (June 1988), 109–116.



## **Sistemas multi-computadora - Sistemas RAIDs**

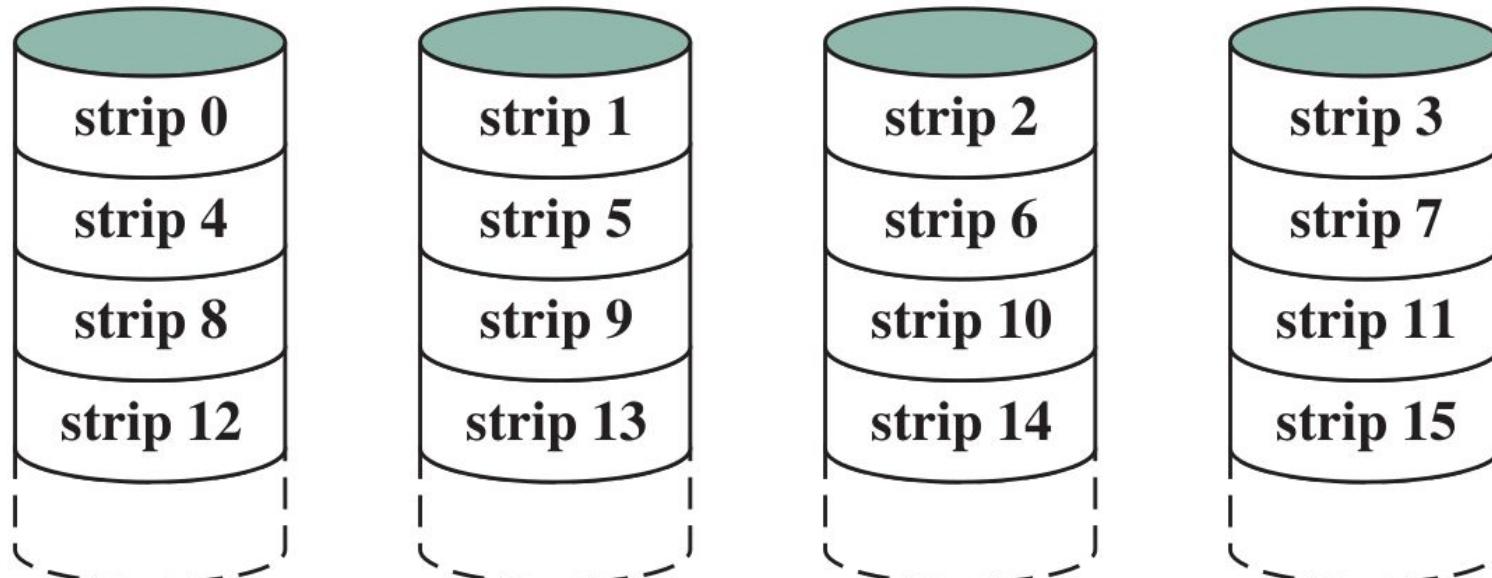
- Fundamento: la velocidad de los discos de almacenamiento masivo ha evolucionado más lentamente que los procesadores.
- Los sistemas RAID proveen:
  - Velocidad.
  - Redundancia.
  - Mayor almacenamiento.
- Formados por arrays de discos duros que pueden ser accedidos simultáneamente.
- Estandarizados por SNIA (Storage Networking Industry Association). (ARM, AMD, Intel, Sisco, Dell, etc).
  - Define 7 arquitecturas, llamadas niveles, numeradas del 0 al 6.
- El sistema operativo ve un solo disco lógico.



## Sistemas multi-computadora - Sistemas RAIDs

### RAID 0

- Los discos son divididos en strips de igual tamaño. Los datos son divididos en strips y distribuidos en los discos siguiendo una secuencia round-robin.



Logical Disk

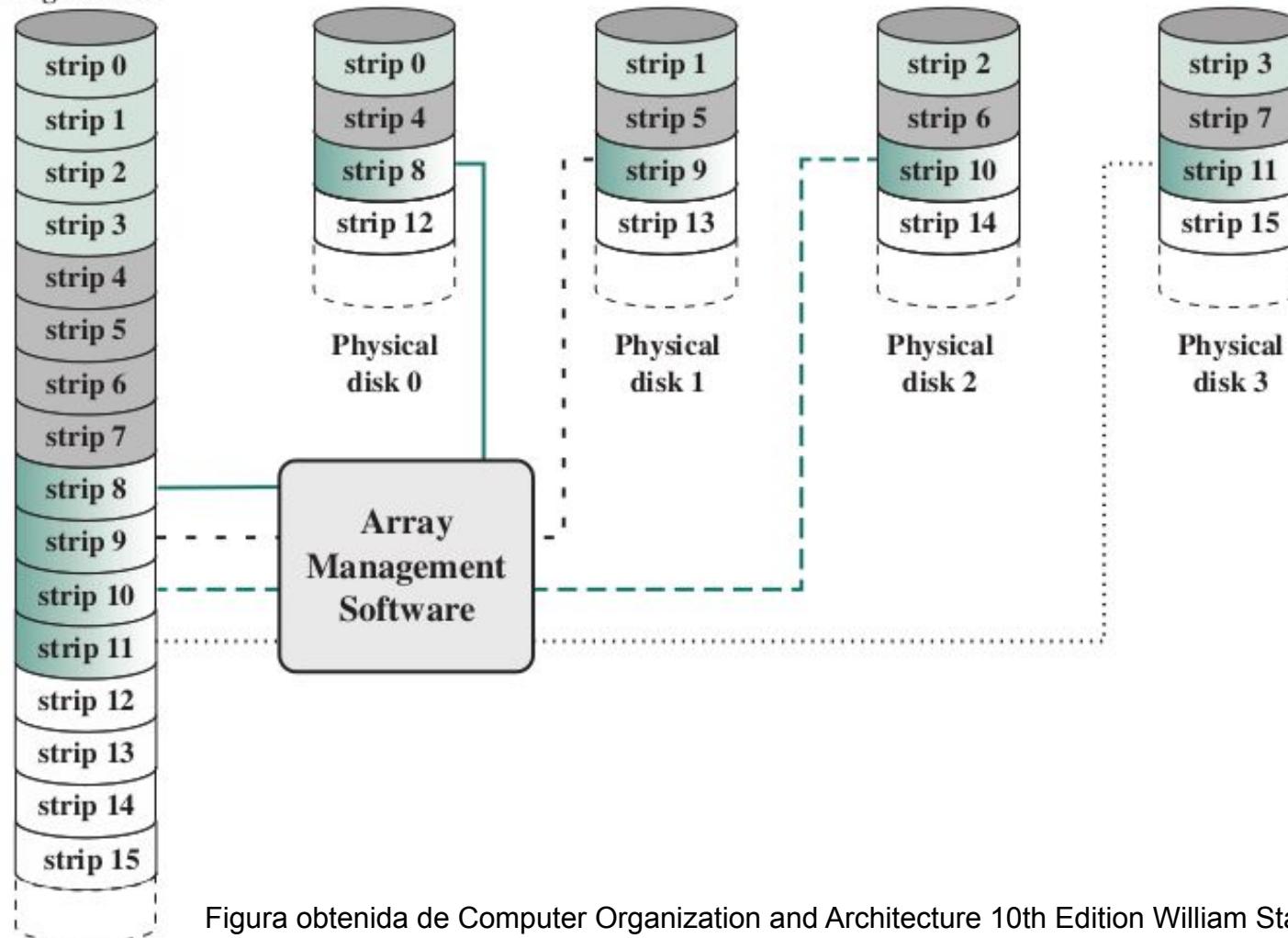


Figura obtenida de Computer Organization and Architecture 10th Edition William Stallings, pag. 209



## **RAID 0**

- Objetivo: Alta velocidad de lectura y escritura.
  - N veces más rápido que cada disco individual (N: el número de discos).
- Tamaño típico de los strips: 64KB-128KB.
- Redundancia: Ninguna.
  - Si un disco falla, se pierden todos los datos.
- Disponibilidad<sup>1</sup>: menor que un disco simple (si hay N discos, hay mayor probabilidad que uno falle).
- Tamaño del disco lógico: N\*(tamaño del menor disco).
- Número mínimo de discos: 2.
- Lecturas simultáneas<sup>2</sup>: No (Los datos siempre se dividen en muchos strips)
- Aplicaciones: Producción y edición de video o cualquier aplicación que requiere gran ancho de banda (bits por segundo).

<sup>1</sup>Disponibilidad: Capacidad de estar disponible a los usuarios o clientes todo el tiempo. Relacionado con la redundancia y robustez (si una falla hace que deje de estar operativo, pierde disponibilidad).

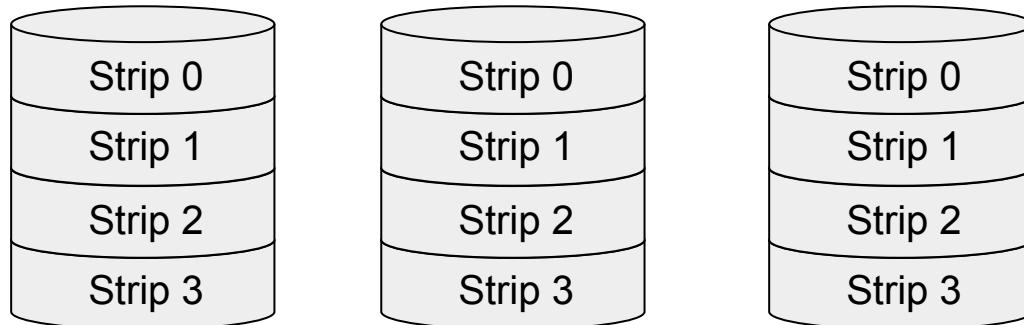
<sup>2</sup>Possibilidad de leer dos o más datos independientes al mismo tiempo.



## Sistemas multi-computadora - Sistemas RAIDs

### RAID 1

- Todo el contenido se duplica (o multiplica) en discos espejo.
- Objetivo: Alta redundancia y disponibilidad.
  - Discos que pueden fallar: todos menos 1.





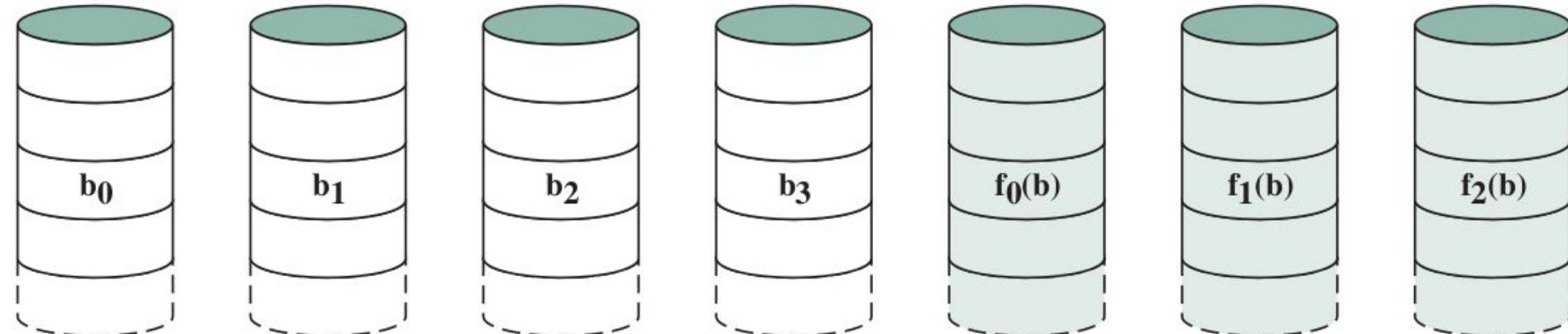
## Sistemas multi-computadora - Sistemas RAIDs - Tipos RAID 1

- Tamaño típico de los strips: 64KB-128KB (igual que en RAID 0).
  - No es obligatorio usar strips, pero es lo habitual.
- Velocidad de lectura: Alta.
  - Pueden leerse en paralelo diferentes porciones de un strip de cada disco.
- Velocidad de escritura: velocidad del disco más lento
  - Cada dato debe escribirse todos los discos.
- Disponibilidad: muy alta.
- Tamaño del disco lógico: Tamaño del menor disco.
- Número mínimo de discos: 2.
- Lecturas simultáneas: No
- Aplicaciones: Datos de contabilidad o información financiera de empresas, datos críticos, aplicaciones que requieren el mayor grado de disponibilidad.



## Sistemas multi-computadora - Sistemas RAIDs - Tipos RAID 2

- Los datos se dividen en strips muy pequeños, de un byte o palabra.
- Se aplica un código de detección y corrección de errores bit a bit.
  - Usualmente código Hamming (poder de detección: 2 bits, poder de corrección de 1 bits) .
  - Puede fallar un disco sin pérdida de disponibilidad.



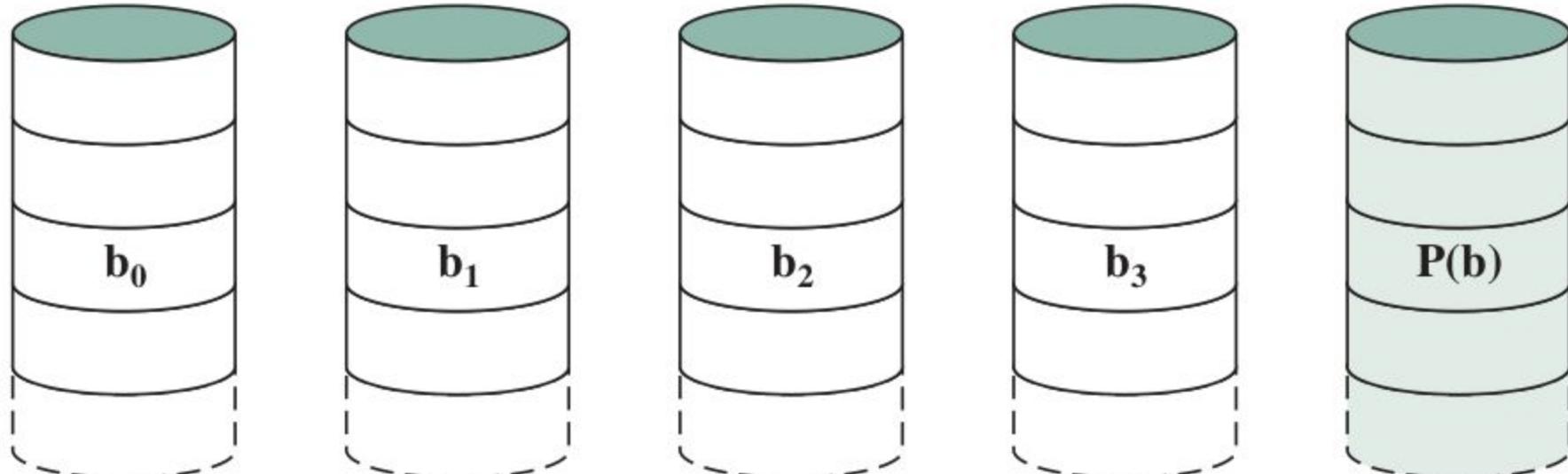


## **Sistemas multi-computadora - Sistemas RAIDs - Tipos RAID 2**

- Objetivo: Alta redundancia y disponibilidad.
- Alta velocidad de lectura y escritura (los sistemas RAID 2 y 3 son los más rápidos para lectura y escritura de datos grandes).
- Disponibilidad: muy alta.
- Tamaño del disco lógico:  $N^*$ (disco más pequeño). Siendo N el número de discos de datos (sin contar los discos de redundancia).
- Número mínimo de discos: 5.
- Lecturas simultáneas: NO
- Aplicaciones: No posee aplicaciones actualmente.
  - Útiles en ambientes donde se producen errores de lectura frecuentes, lo cual no es usual en los discos duros actuales (son muy confiables).
  - Dificultad: los cabezales de los discos deben estar alineados.
  - Uso ineficiente de discos (ver RAID 3).

## Sistemas multi-computadora - Sistemas RAIDs - Tipos **RAID 3**

- Los datos se dividen en strips muy pequeños, de un byte o palabra (Como en RAID 2).
- Se utiliza código de paridad aplicado a nivel de bits.





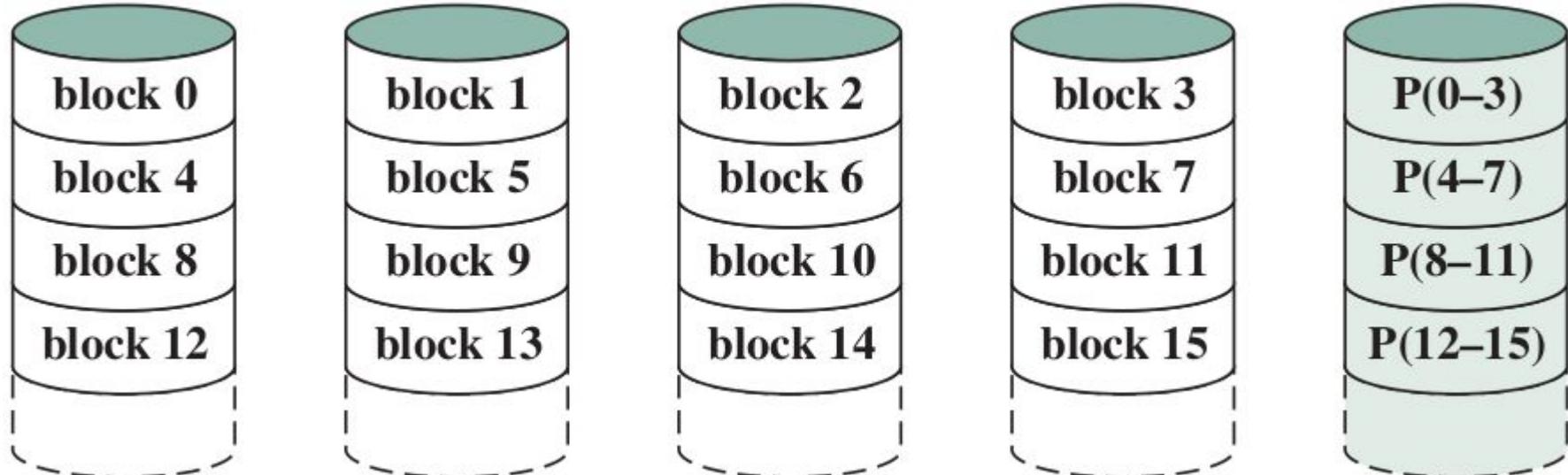
## RAID 3

- Objetivo: Alta redundancia y disponibilidad (mediante código de paridad).
- Alta velocidad de lectura y escritura (menor que en RAID 0 por el tiempo necesario para ejecutar el algoritmo de detección y corrección de errores).
  - Los discos deben estar sincronizados (misma posición del cabezal) para lograr altas velocidades.
- Redundancia: un disco con los bits de paridad.
  - Puede fallar un disco. Los datos pueden recuperarse de los demás (Un código de paridad permite corregir un error en un bit si se conoce el bit que falla. No pueden corregirse errores aleatorios).
- Disponibilidad: muy alta.
- Tamaño del disco lógico:  $N^*$ (tamaño del disco más pequeño).
- Número mínimo de discos: 3.
- Lecturas simultáneas: No.
- Aplicaciones: streaming, edición de video e imágenes, aplicaciones que requieren alto throughput y disponibilidad.



## Sistemas multi-computadora - Sistemas RAIDs - Tipos RAID 4

- Similar a RAID 3 (código de paridad bit a bit y disco de bits de paridad), pero los datos se dividen en strips de tamaño mayor que los RAID 0, 1, 2 o 3 (128KB o 256 KB).





## **Sistemas multi-computadora - Sistemas RAIDs - Tipos RAID 4**

- Objetivo: Disponibilidad, alta velocidad de acceso, permitir acceso a diferentes datos en paralelo.
- Redundancia: código de paridad aplicada a nivel de bits.
  - Puede fallar un disco. Los datos pueden recuperarse de los demás.
- Alta disponibilidad.
- Tamaño del disco lógico:  $N^*$ (disco más pequeño), N=número de discos de datos.
- Número mínimo de discos: 3
- Velocidad de lectura
  - Alta para datos grandes (aprovecha el paralelismo si el dato está distribuido en muchos discos).
  - Media o baja para datos pequeños (no puede aprovechar el paralelismo si el dato está distribuido en pocos discos, o en solo 1).
  - Pueden realizarse varias lecturas simultáneas de datos independientes.



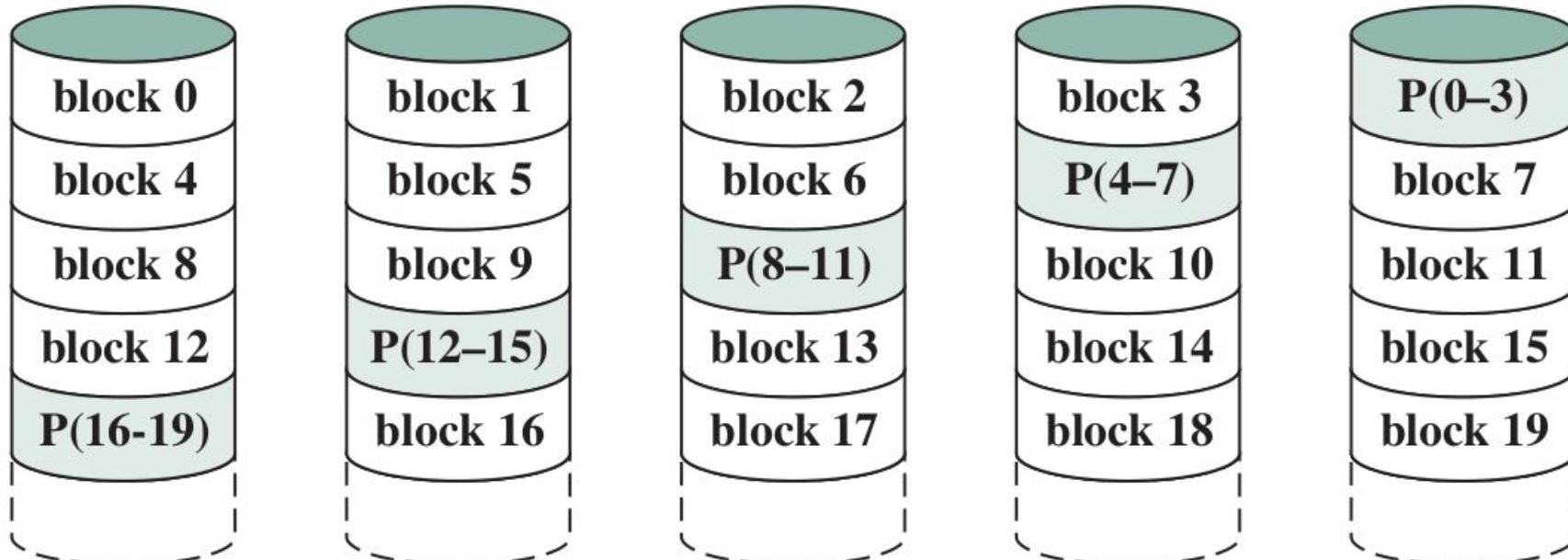
## **Sistemas multi-computadora - Sistemas RAIDs - Tipos RAID 4**

- Velocidad de escritura:
  - Baja para datos pequeños (menor a la velocidad de un disco si el dato ocupa un solo bloque).
    - El nuevo bit de paridad se calcula en función de los datos anteriores, los datos nuevos y el anterior bit de paridad. (deben leerse los discos involucrados, calcular paridad y luego escribir los discos).
      - $b_{\text{PARIDAD(NUEVO)}} = b_{\text{PARIDAD(ANTERIOR)}} \oplus \text{bit}_{\text{QUE CAMBIA(DESPUES)}} \oplus \text{bit}_{\text{QUE CAMBIA(ANTES)}}$
  - Alta velocidad datos grandes (Si el dato ocupa toda una fila, la nueva paridad puede leerse de los datos nuevos, antes de escribirlos al disco).
- Lecturas simultáneas: Si (mayor ventaja de RAID 4).
- Aplicaciones: Actualmente no se utiliza, se utiliza RAID 5.
  - Problema de RAID 4: Para escritura de varios datos en paralelo que ocupen varias filas, el disco de paridad es un cuello de botella.



## Sistemas multi-computadora - Sistemas RAIDs - Tipos RAID 5

- Muy similar a RAID 4, pero la información de paridad se distribuye entre todos los discos.





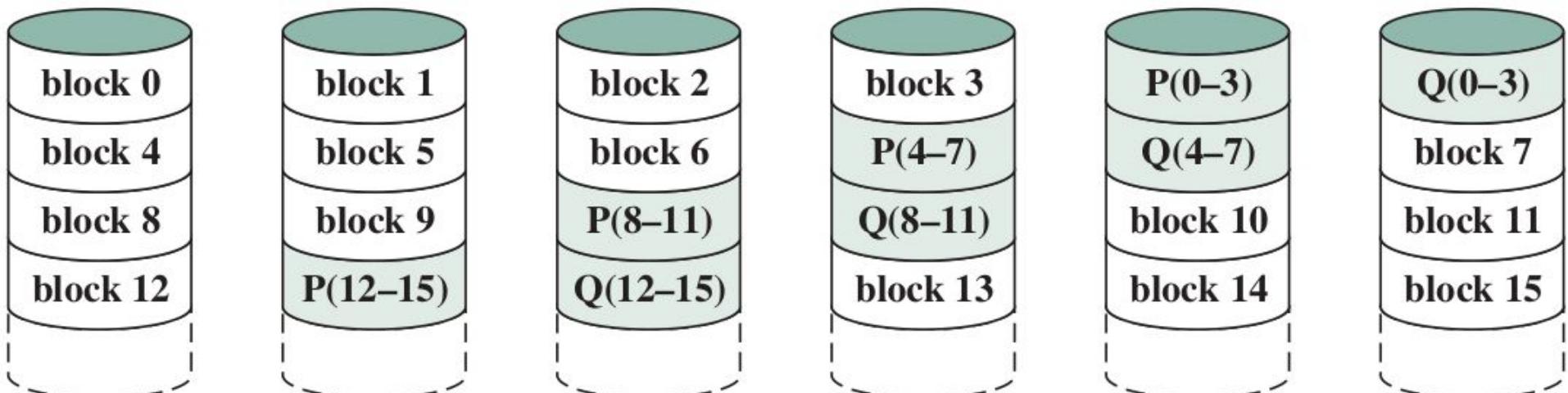
## Sistemas multi-computadora - Sistemas RAIDs - Tipos RAID 5

- Posee todas las características de RAID 4, pero mayor velocidad para la escritura de muchos datos en paralelo.
  - Escrituras a varias filas de bloques simultáneas no verán al disco de paridad como un cuello de botella.
- Aplicaciones: servidores de archivos, base de datos, servidores web. Actualmente es el sistema RAID más utilizado.



## Sistemas multi-computadora - Sistemas RAIDs - Tipos RAID 6

- Igual a RAID 5, pero agrega más redundancia.
  - Redundancia mediante dos algoritmos de detección de errores. Algoritmo de paridad (disco P, igual a RAID 5) y otro algoritmo diferente (Q).





## Sistemas multi-computadora - Sistemas RAIDs - Tipos **RAID 6**

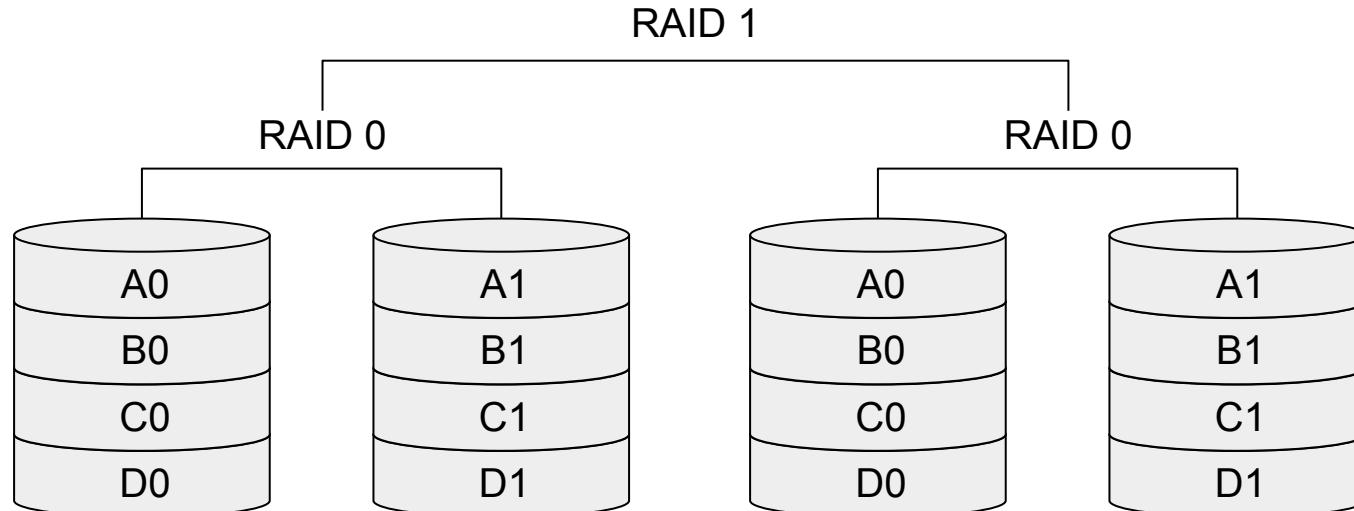
- Objetivo: Alta redundancia y disponibilidad, con alta velocidad de acceso.
- Velocidades de lectura y escritura similares a RAID 5 (un poco menor por tener que calcular un segundo algoritmo).
- Redundancia: Se utilizan dos algoritmos que generan un bit de redundancia. Un disco para cada algoritmo.
  - Pueden fallar dos discos.
- Tamaño del disco lógico: Igual a RAID 5.
- Número mínimo de discos: 4 (2 más 2 de detección de errores).
- Pueden realizarse varias lecturas simultáneas.
- Aplicaciones actuales: Datos críticos (más económico que RAID 1 cuando los discos son 5 o más).



## Sistemas multi-computadora - Sistemas RAIDs

### RAID 0+1

- Los discos se dividen en grupos. Cada grupo constituye un RAID 0. Los grupos forman un RAID 1.
- Al menos 4 discos.
- Se deben añadir discos en grupos (2 para el ejemplo de la figura).

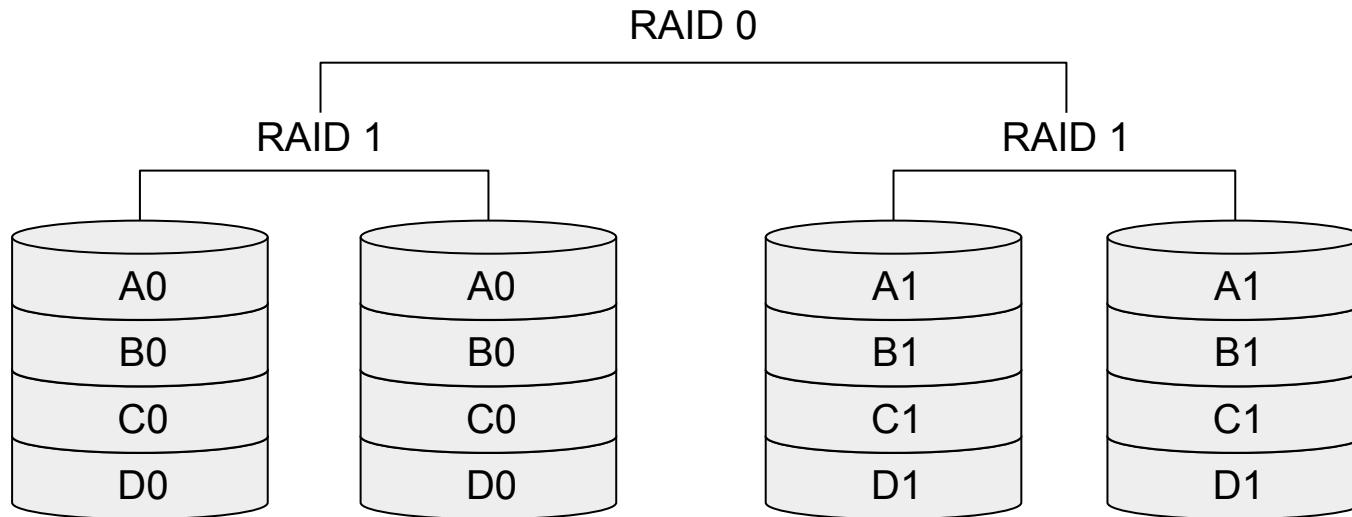




## Sistemas multi-computadora - Sistemas RAIDs - Tipos

### RAID 1+0

- Los discos se dividen en grupos. Cada grupo constituye un RAID 1. Los grupos forman un RAID 0.





### **Temas**

Tipos de sistemas paralelos (Clasificación de Flynn)

Paralelismo a nivel de instrucción

Paralelismo a nivel de hilos (Procesadores multihilo simultáneo)

Paralelismo a nivel de núcleos

- Arquitecturas multi núcleo simétricas

- Arquitecturas multi núcleo heterogéneas con igual ISA o con ISA diferentes

- Sistemas CPU-GPU, CPU-DSP. GPGPU. Introducción a CUDA y OpenCL

- Sistemas operativos para multiprocesadores.

Paralelismo a nivel de procesos (Cluster)

Paralelismo a nivel de datos (SIMD)

Sistemas RAID



### **Performance y escalabilidad**

- Speedup y eficiencia

- Modelos de problemas paralelizables

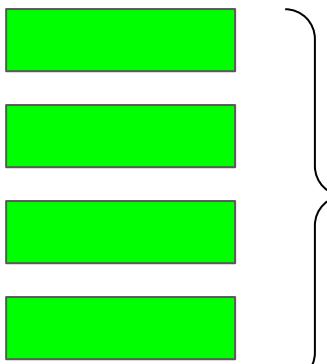
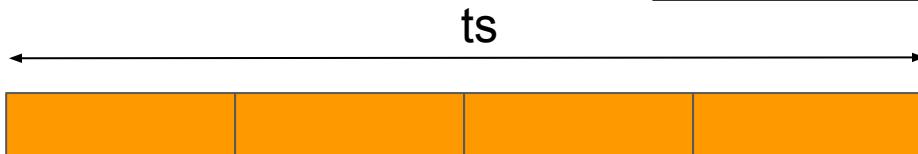
- Paralelismo en el software



## Medidas de performance: Speedup

**Speedup ideal:**  $S(n) = \frac{ts}{tm}$

ts=Tiempo total n tareas ejecución secuencial.  
tm=Tiempo de cada tarea ejecutándose en paralelo.



Speedup ideal, sin overhead de comunicación  
y con código 100% paralelizable

$$\text{Speedup: } S(n) = \frac{ts}{tm} = \frac{ts}{\frac{ts}{n}} = n$$

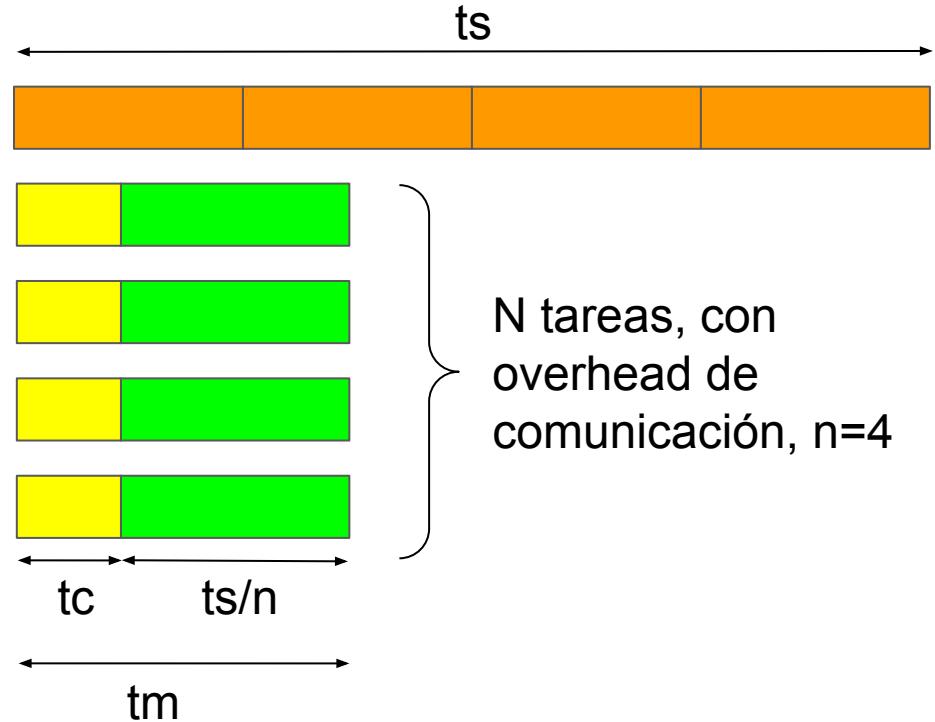
**Speedup Ideal**



## Speedup considerando overhead de comunicación

$$tm = \frac{ts}{n} + tc$$

Tiempo usando por cada  
procesador para tareas de  
comunicación





## Speedup considerando overhead de comunicación

$$tm = \frac{ts}{n} + tc$$

Speedup:  $S(n) = \frac{\frac{ts}{n}}{\frac{ts}{n} + tc} = \frac{n}{1 + n \frac{tc}{ts}}$

$tc \ll ts \rightarrow S = n$

$n \rightarrow \infty$

Caso ideal ( $tc=0$ ):  $S \rightarrow \infty$

Caso real ( $tc > 0$ ):  $S = ts/tc$

**Por esto es importante que el tiempo de  
comunicación sea lo más pequeño posible.**



## **Eficiencia o escalabilidad**

$$\xi = \frac{S}{n} \quad \text{Speedup por procesador}$$

Ideal:  $S = n \longrightarrow \xi = 1$  ← Significa que si se aumenta k veces el número de procesadores, aumenta k veces el speedup

Con  $tc > 0$ :

$$S = \frac{n}{1 + n \frac{tc}{ts}}$$

$$\xi = \frac{1}{1 + n \frac{tc}{ts}}$$

Si  $tc/ts = 0$ ,  $\xi = 1$   
Si  $tc/ts > 0$ ,  $\xi = 0$



## Escalabilidad de hardware

- La **escalabilidad** (o grado de escalabilidad) indica el grado de acercamiento de un sistema real a un sistema linealmente escalable.
  - Indica la habilidad de utilizar eficientemente recursos añadidos.
- Depende fuertemente de la topología<sup>1</sup> y la velocidad de la red de interconexión (overhead de comunicación).
- Una arquitectura paralela es **linealmente escalable** si:
  - Si se puede expandir a un sistema de mayor tamaño (ej: número de procesadores o computadoras) con un aumento lineal de performance.
  - La **escalabilidad lineal es la situación ideal**, imposible de lograr en la práctica.

<sup>1</sup>Se verá en la unidad 2 de la materia

## Escalabilidad de hardware

- Escalabilidad:
  - Speedup ideal = n (Speedup de arquitectura linealmente escalable).

$$\text{Escalabilidad} = \frac{\text{Speedup real}}{\text{Speedup ideal}} = \frac{\text{Speedup real}}{n}$$

- Coincide con la definición de eficiencia:

$$\xi = \frac{\text{Speedup real}}{n}$$

- Definición alternativa de sistema linealmente escalable: mantiene fija la eficiencia a medida que n aumenta.



## Ejecución paralela con código no parallelizable (serial)

```
For  $i \leftarrow 1, n$ 
     $c(i) \leftarrow a(i) + b(i);$ 
```

*done in parallel, each processor does one addition*

$\text{Sum} \leftarrow 0;$

```
For  $j \leftarrow 1, n$ 
     $\text{sum} \leftarrow \text{sum} + c(j);$ 
```

*only one processor can do this (serial section)*

$\text{Average} \leftarrow \text{sum}/n;$

```
For  $k \leftarrow 1, n$ 
     $a(k) \leftarrow a(k) - \text{average};$ 
     $b(k) \leftarrow b(k) - \text{average}$ 
```

*done in parallel, each processor updates its value*



## Ejecución paralela con código no parallelizable

- Para analizar el efecto de la fracción de código no parallelizable, debemos considerar el tipo de problema a paralelizar.
  - Modelo de Amdahl o problema de tamaño fijo.
    - Problemas donde el tiempo para resolver el problema es el parámetro crítico.
  - Modelo de Gustafson-Barsis o problema de tamaño variable proporcional a la cantidad de tareas que pueden ejecutarse en paralelo.
    - Problemas donde se adapta la precisión del resultado o tamaño de los modelos a los recursos disponibles.
  - Modelo de Sun y Ni o de memoria total limitada.
    - Problemas con grandes cantidades de datos.

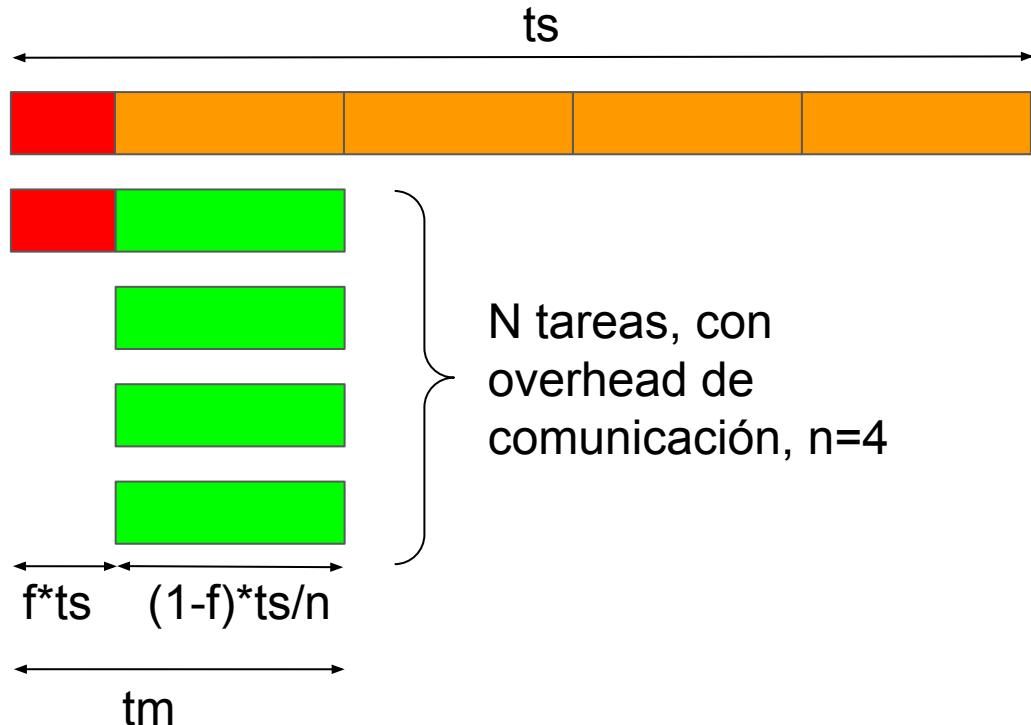
No confundir modelo de Amdahl con Ley de Amdahl. La Ley de Amdahl (“La mejora global que se consigue al mejorar un componente de un sistema de cómputo depende de la fracción de tiempo que se use dicho componente”) está planteada para problemas que siguen el modelo de Amdahl.

## Ejecución paralela con código no parallelizable según modelo de Amdahl

- Sin importar el overhead

$$tm = f * ts + (1-f) * \frac{ts}{n}$$

f: Fracción de código no  
parallelizable (serial)

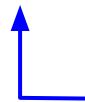




## Ejecución paralela con código no parallelizable según modelo de Amdahl

- Sin importar el overhead

$$tm = f * ts + (1-f) * \frac{ts}{n}$$



Fracción de código no paralelizable (serial)

$$S = \frac{\frac{ts}{n}}{f * ts + \frac{(1-f) * ts}{n}} = \frac{n}{1 + (n-1) * f}$$

$f = 1$  (ninguna parte del código es paralelizable)  $\Rightarrow S = 1$

$n \rightarrow \infty \Rightarrow S = 1/f$

Sin importar la arquitectura ni  $n$ , el  $S$  máximo está limitado por la porción de código no paralelizable



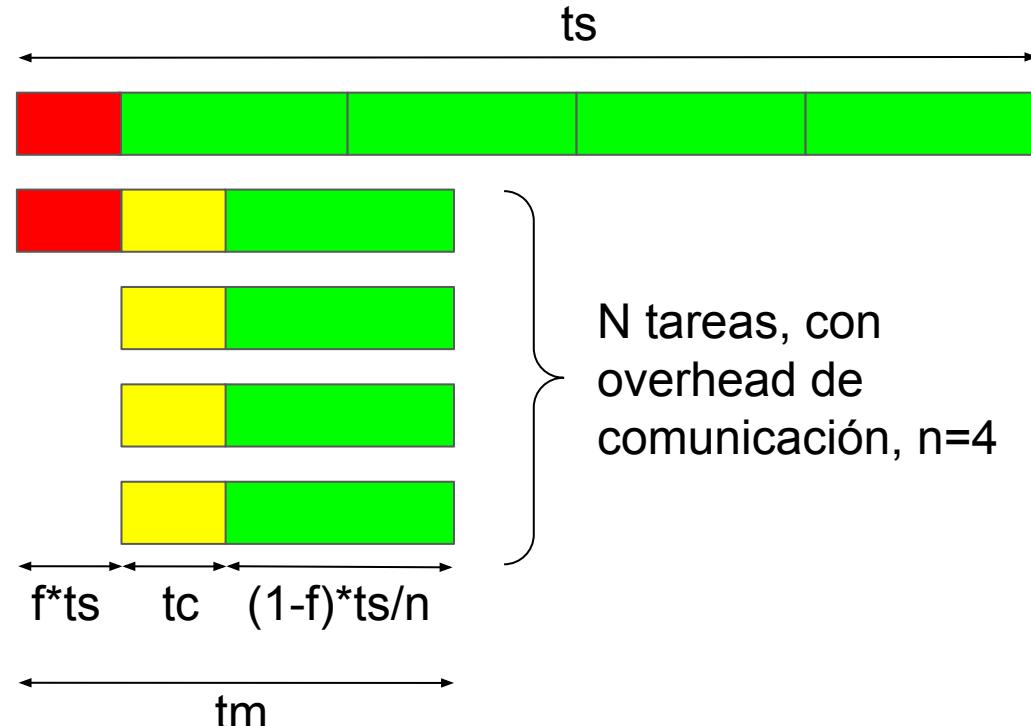
## Ejecución paralela con código no parallelizable según modelo de Amdahl

- Considerando overhead y código no parallelizable

$$tm = f * ts + \frac{(1-f) * ts}{n} + tc$$

tc=Tiempo de comunicación  
(overhead)

f=Fracción de código no  
parallelizable





## Ejecución paralela con código no parallelizable según modelo de Amdahl

- Considerando overhead y código no paralelizable

$$tm = f * ts + \frac{(1-f) * ts}{n} + tc$$

$$S = \frac{\frac{ts}{n}}{f * ts + \frac{(1-f) * ts}{n} + tc} = \frac{n}{1 + (n-1) * f + n * \frac{tc}{ts}}$$

$$\text{Si } n \rightarrow \infty \Rightarrow S = \frac{1}{f + tc/ts}$$



## Ejecución paralela con código no parallelizable según modelo de Amdahl

- Sin Considerar overhead y considerando código paralelizable

$$\xi = \frac{S}{n} = \frac{1}{1 + (n-1) * f}$$

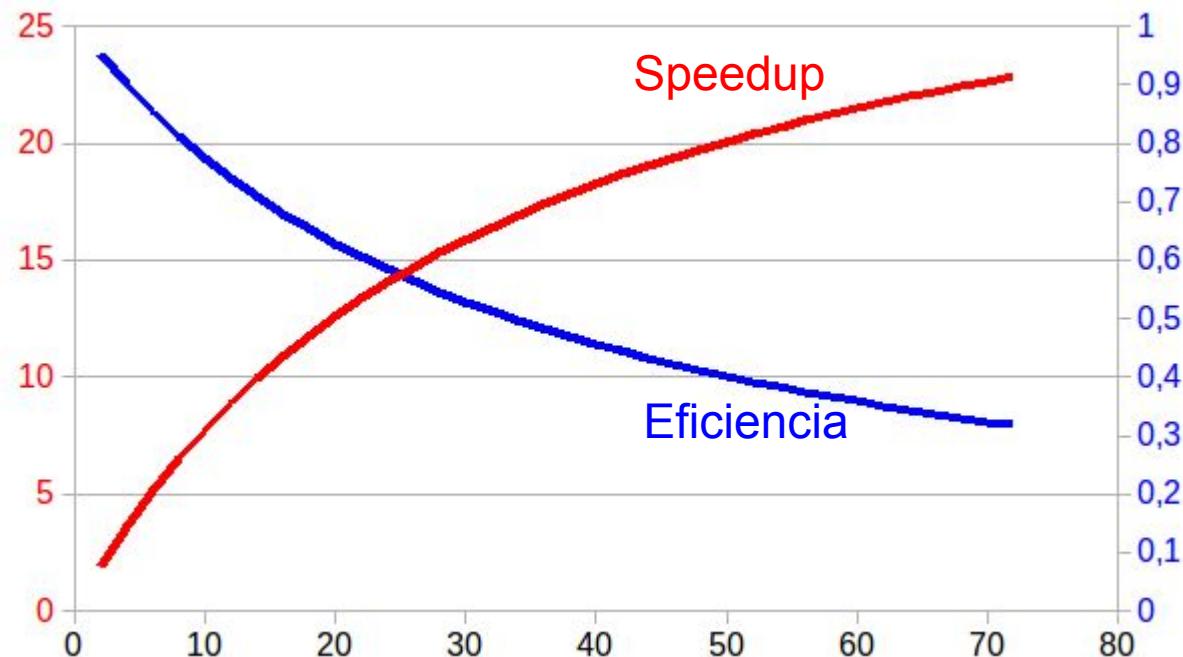
Ejemplo: f=0.2  
n=4;  $\xi=0.625$   
n=100;  $\xi=0.0481$

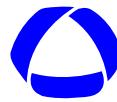
- Con overhead y código paralelizable

$$\xi = \frac{S}{n} = \frac{1}{1 + (n-1) * f + n * \frac{tc}{ts}}$$

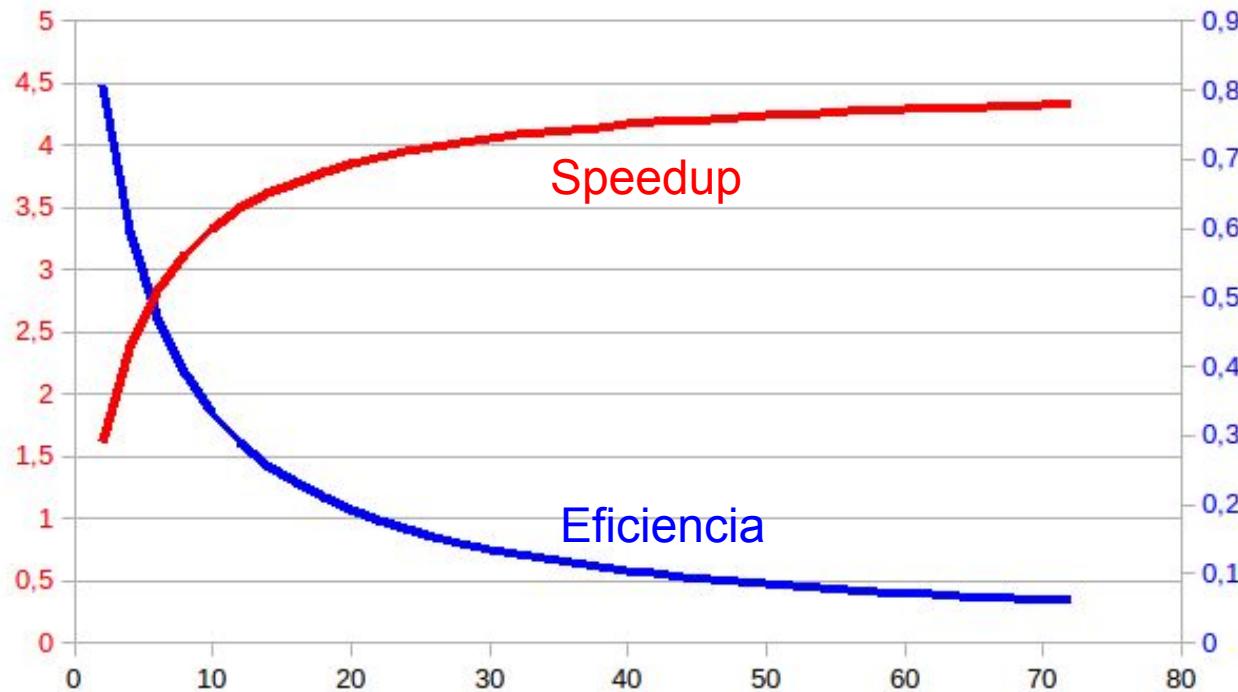


Ejemplo con  $(tc/ts) = 0,02$  y  $f = 0,04$  ( $S_{max} = 25$ )



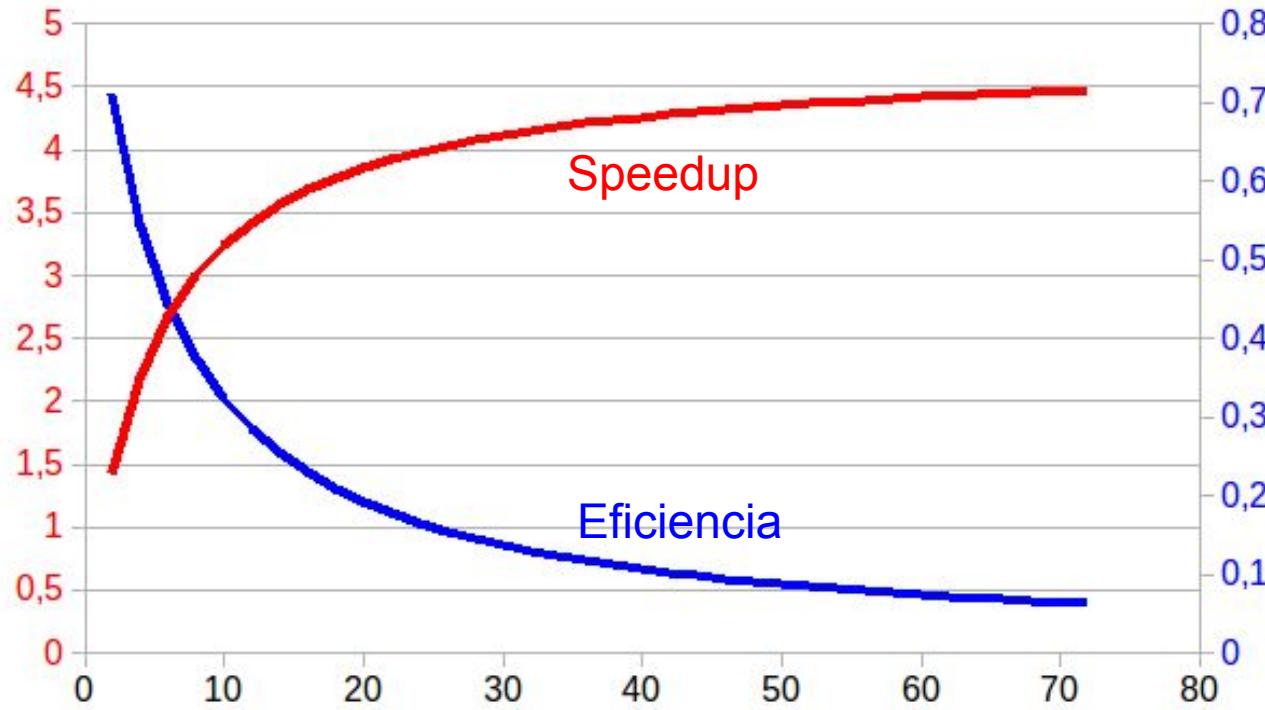


Ejemplo con  $(tc/ts) = 0,02$  y  $f = 0,2$  ( $S_{max} = 4.54$ )





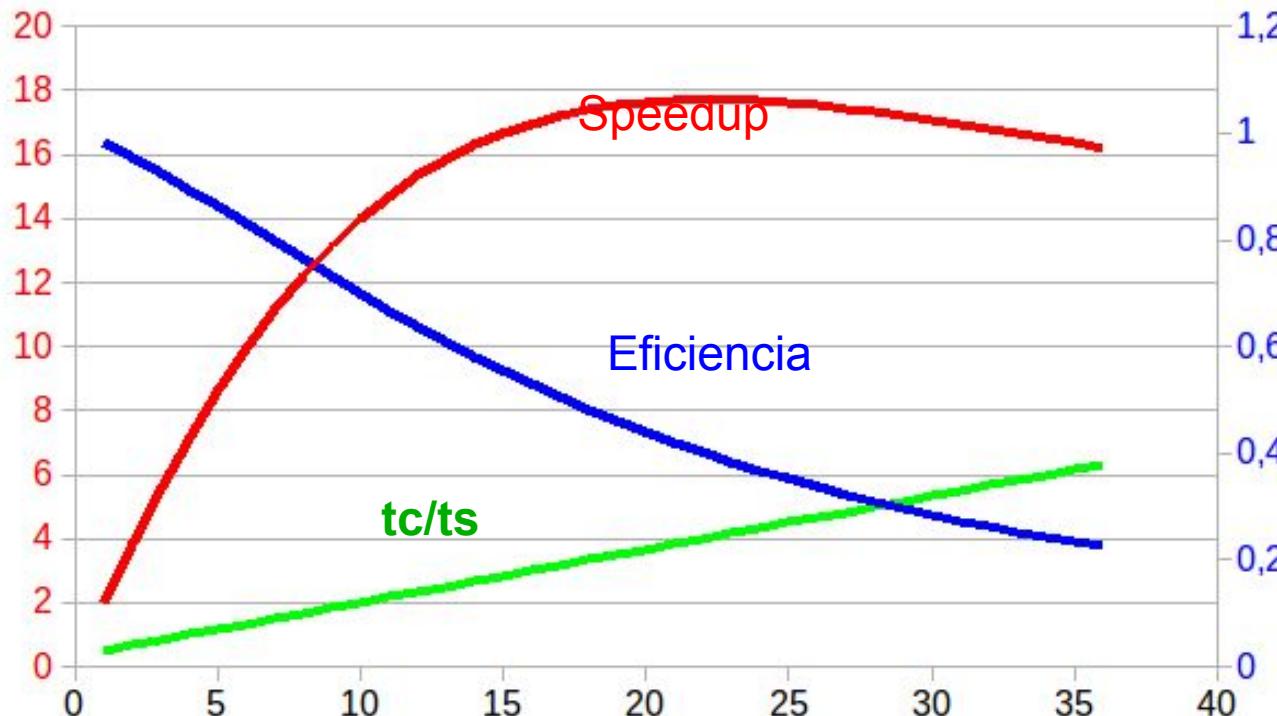
Ejemplo con  $(tc/ts) = 0,2$  y  $f = 0,01$





Ejemplo con  $(tc/ts)$  variable en función de  $n$  y  $f = 0,01$

Nota:  $tc$  depende de la cantidad de tareas a ejecutar en paralelo, del hardware de comunicaciones y del software.





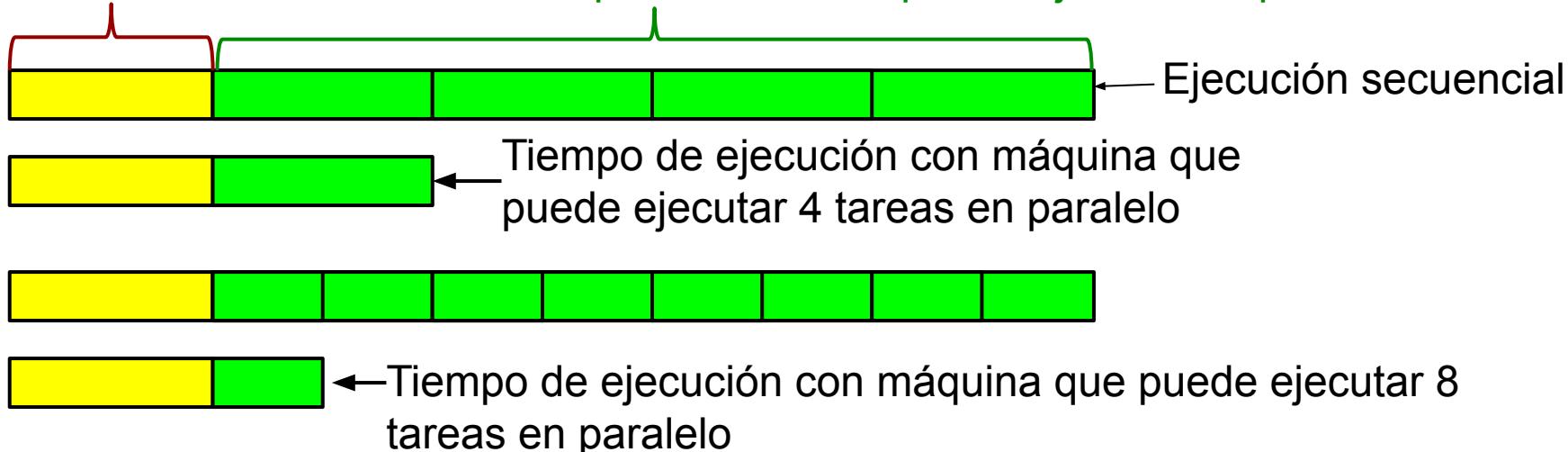
## Ejecución paralela con código no parallelizable según modelo de Amdahl

- A medida que agregamos más procesadores (o computadoras) el speedup **no aumenta linealmente**.
- Mientras mayor la porción de código no paralelizable, menor la ganancia de speedup al agregar más procesadores (disminuye la eficiencia).
- El **overhead de comunicación** (tiempo de comunicación entre procesadores) tiene un impacto importante sobre el speedup.
  - Si el tiempo de comunicación entre procesadores depende del número de procesadores  $n$  (lo cual ocurre en la realidad) puede imponer un máximo a la curva speedup en función de  $n$ .

## Speedup según Ley de Amdahl

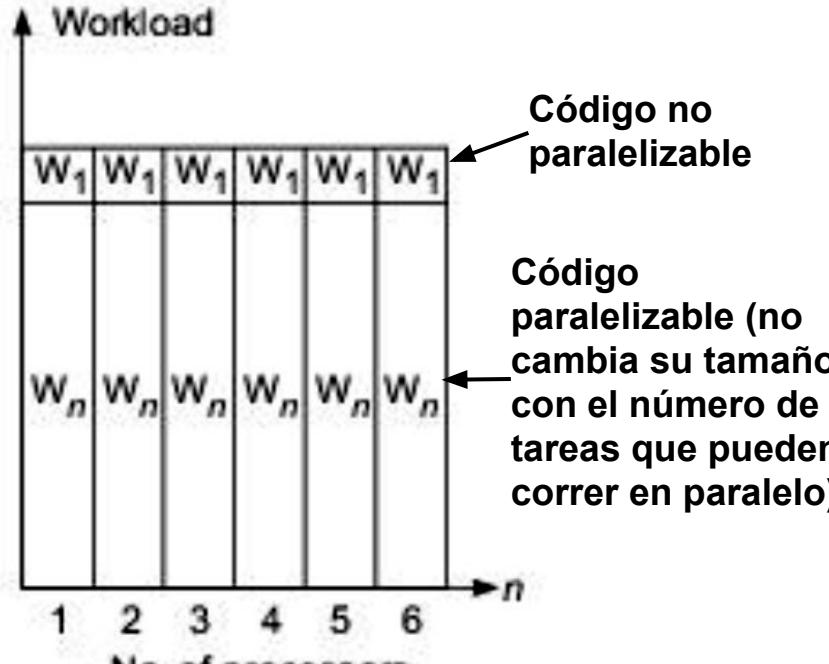
- Modelo de **Amdahl** supone que el tiempo total o **tamaño total del problema es fijo**.
  - Si aumenta n, disminuye el tiempo de ejecución en cada procesador.
  - Ejemplo: Problemas donde se busca el menor tiempo de ejecución.

Código no paralelizable de tamaño fijo dividido de acuerdo al número de tareas que el hardware puede ejecutar en paralelo

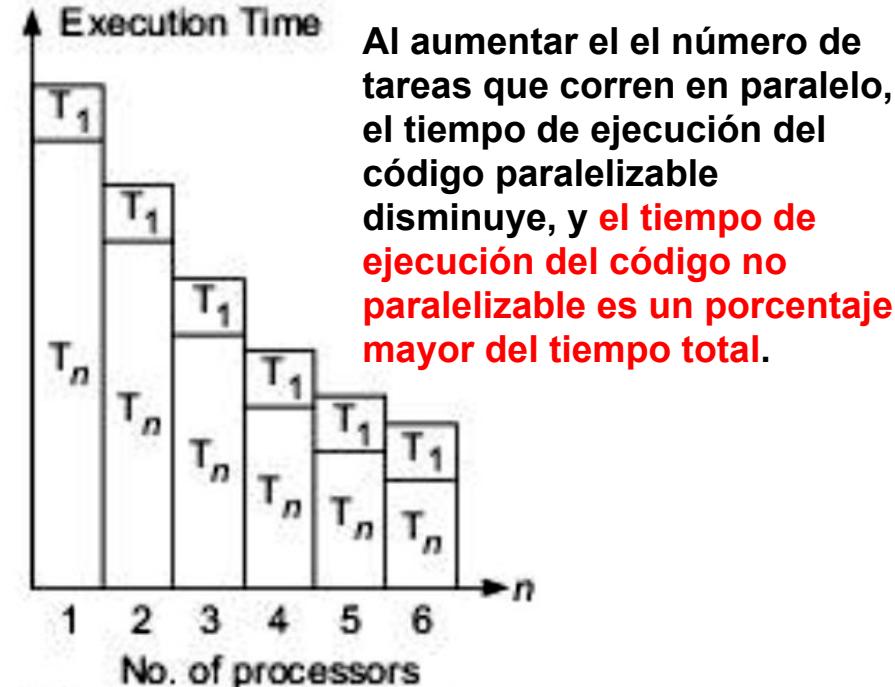




## Speedup según Ley de Amdahl



(a) Fixed workload



(b) Decreasing execution time

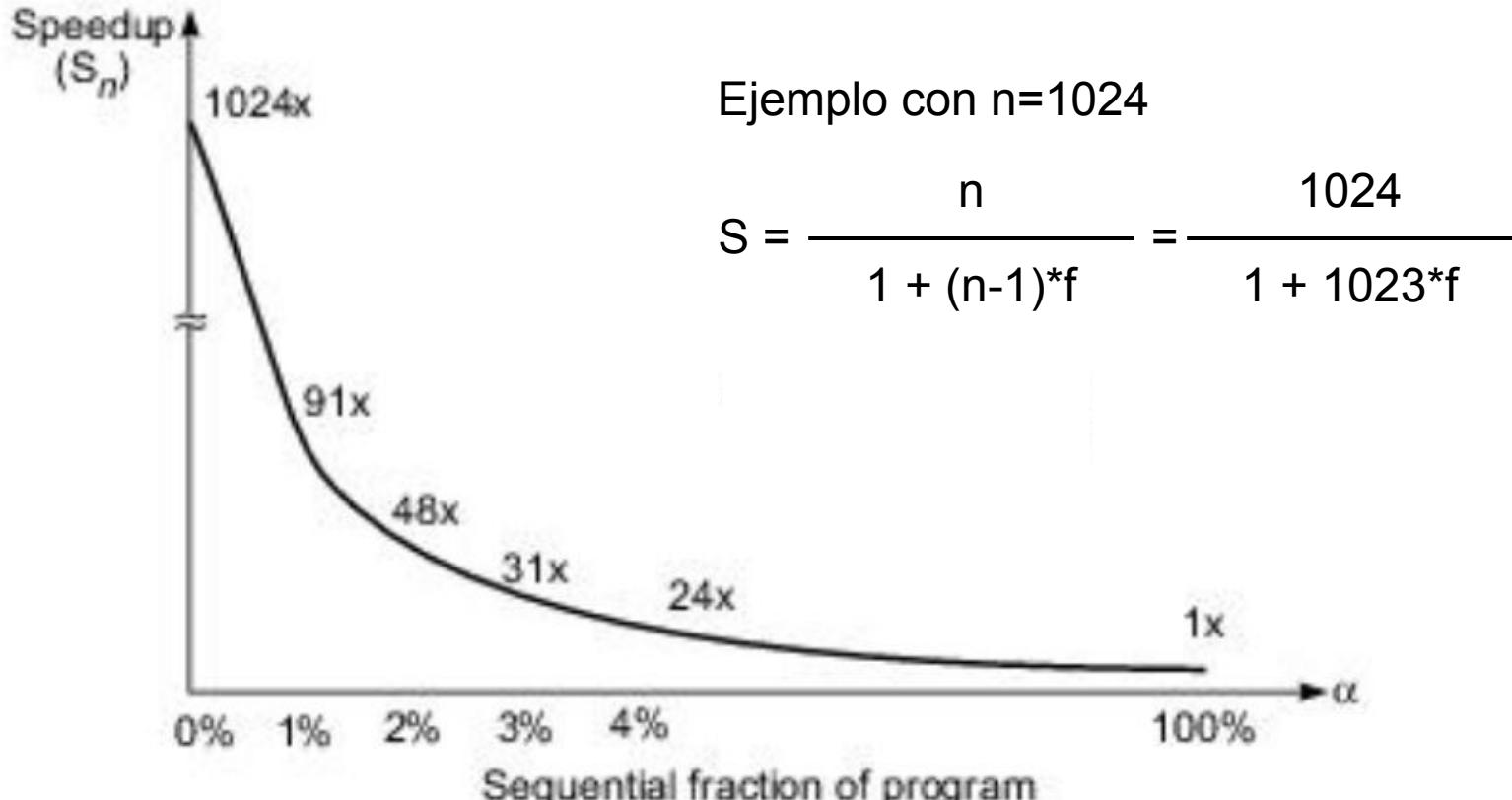


Figura obtenida de Kai Hwang, "Advanced Computer Architecture Parallelism Scalability Programmability", 2º edición, página 110

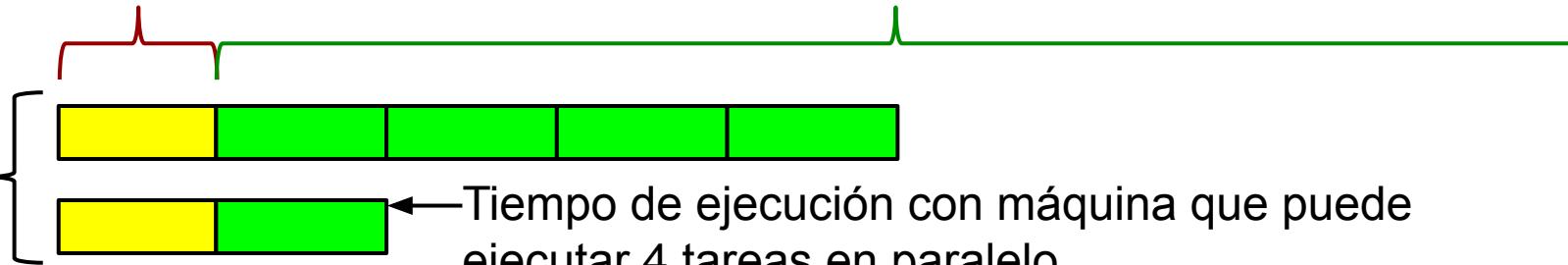
## Speedup según Gustafson-Barsis

- Supone que los tiempos  $t_{SEC}$  y  $t_{PAR}$  son constantes, y el tiempo total o **tamaño del problema aumenta con el valor de n**.

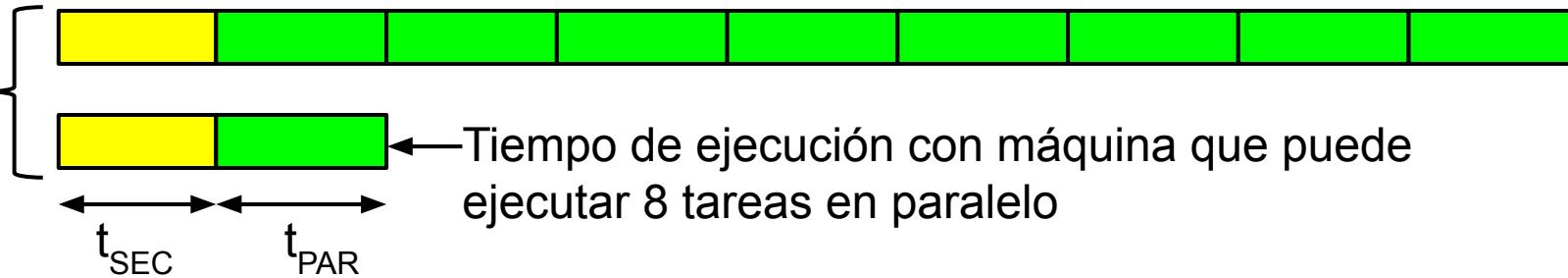
Código no  
paralelizable

Código paralelizable cuyo tamaño aumenta de acuerdo al  
número de tareas que el hardware puede ejecutar en paralelo

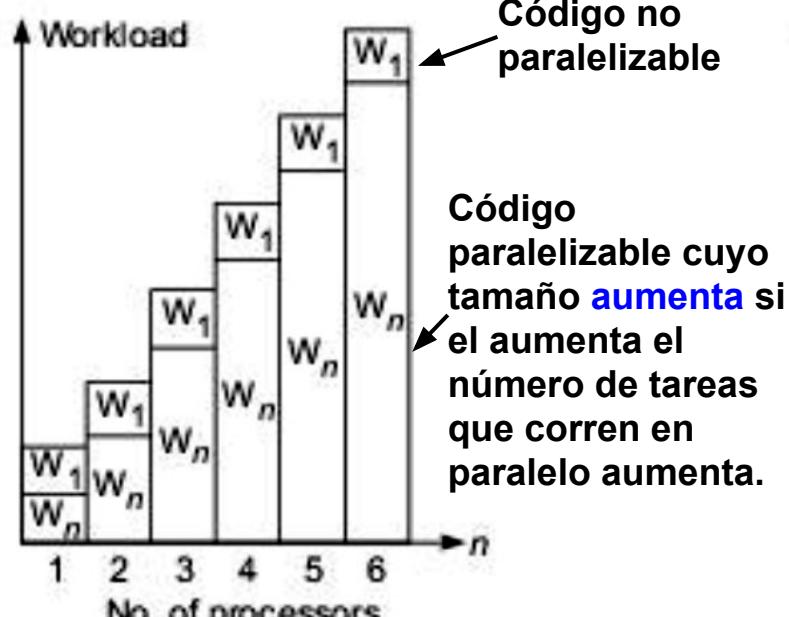
4 núcleos



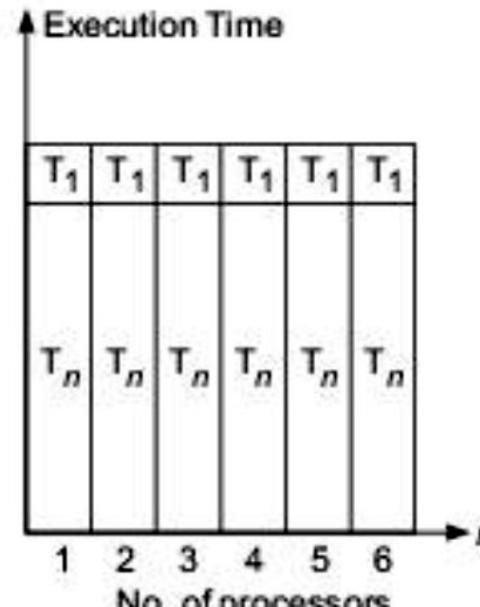
8 núcleos



## Speedup según Ley de Gustafson-Barsis



(a) Scaled workload



(b) Fixed execution time

Al aumentar el el número de tareas que corren en paralelo, el tiempo de ejecución del código paralelizable no cambia, y el tiempo de ejecución del código no paralelizable representa siempre el mismo porcentaje del tiempo total.



## Speedup: Ley de Gustafson-Barsis

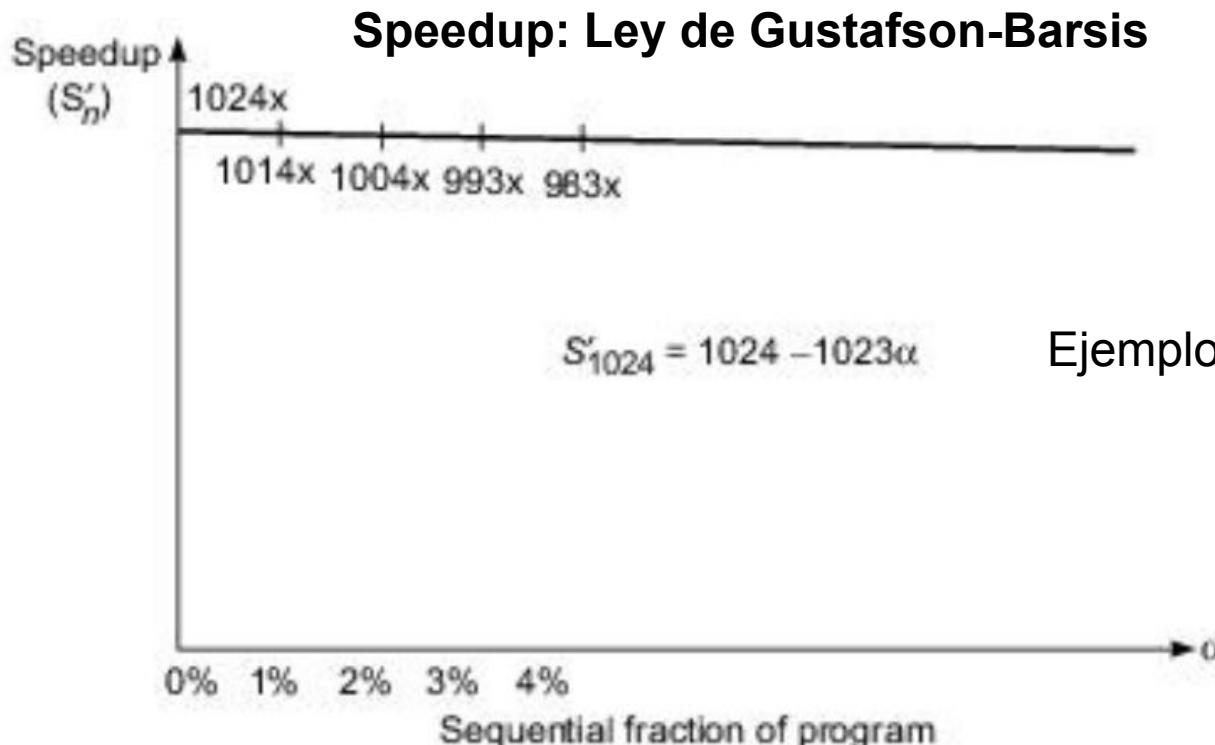
$$S = \frac{t_{SEC} + t_{PAR} * n}{t_{SEC} + t_{PAR}} \quad (\text{Ver figura filmina anterior})$$

$$\begin{aligned} a &= \frac{t_{SEC}}{t_{SEC} + t_{PAR}} & (1 - a) &= 1 - \frac{t_{SEC}}{t_{SEC} + t_{PAR}} = \frac{t_{SEC} + t_{PAR} - t_{SEC}}{t_{SEC} + t_{PAR}} = \frac{t_{PAR}}{t_{SEC} + t_{PAR}} \\ \Rightarrow & \end{aligned}$$

$$S = \frac{t_{SEC} + t_{PAR} * n}{t_{SEC} + t_{PAR}} = \frac{t_{SEC}}{t_{SEC} + \frac{t_{PAR} * n}{t_{PAR}}} + \frac{t_{PAR} * n}{t_{SEC} + \frac{t_{PAR} * n}{t_{PAR}}} = a + (1 - a)n$$

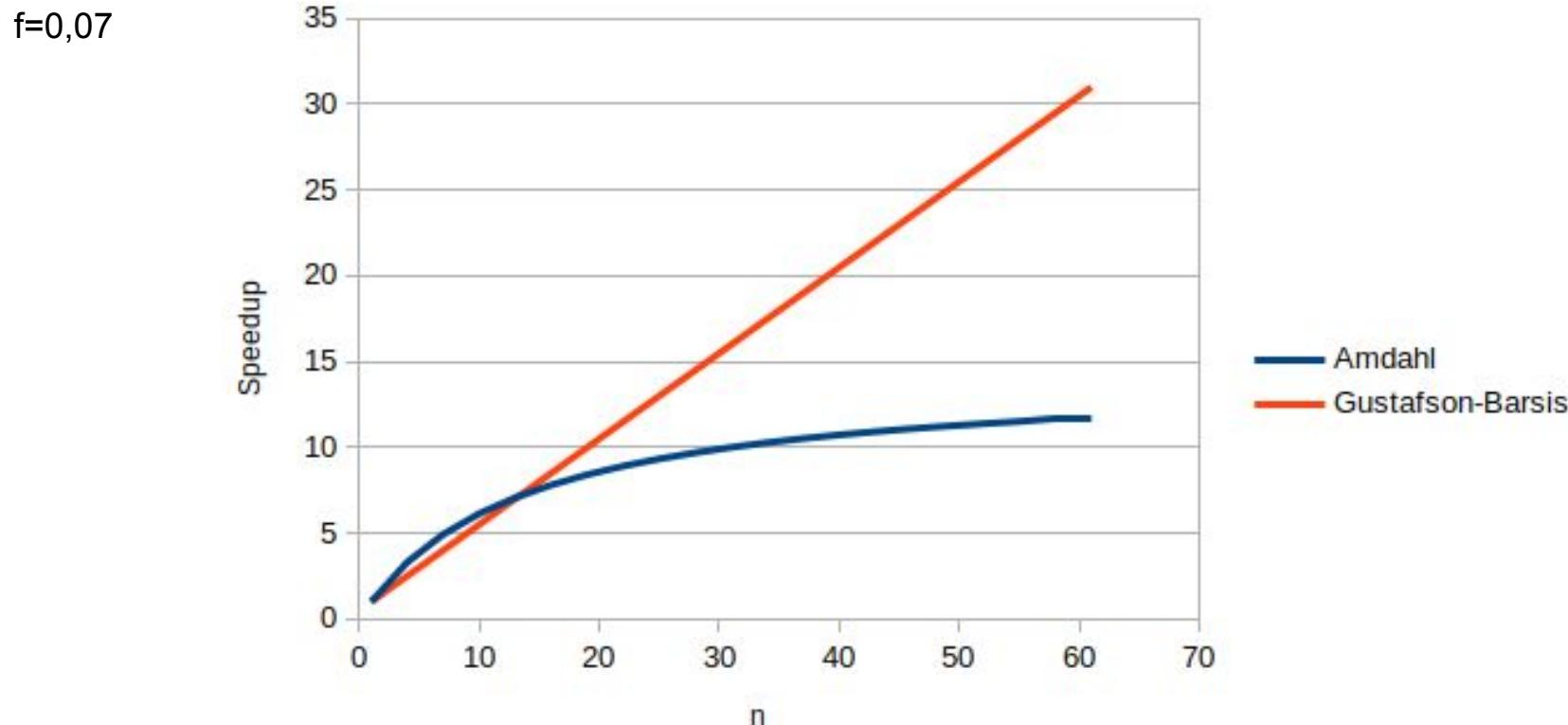
$a < 1$

Si  $n \rightarrow \infty \Rightarrow S \rightarrow \infty$





## Speedup





## Speedup según Gustafson-Barsis

- La **Ley de Amdahl** indica que **el speedup tiene un límite superior para problemas de tamaño fijo.**
  - Conclusión muy negativa, ya que indica que “no vale la pena” incrementar la cantidad de procesadores dado un valor de  $f$  para conseguir mayores speedup.
- La **Ley de Gustafson-Barsis** indica que **el speedup no posee límite** superior si el **tamaño del problema crece linealmente** con el número de procesadores.
  - **Conclusión muy positiva, ya que indica que si vale la pena incrementar el número de procesadores para conseguir mayores speedup.**
- Ambos suponen modelos ideales, pero los problemas reales se acercan más al modelo de la **Ley de Gustafson** ya que:
  - Los problemas requieren la mayor precisión posible (mayor número de decimales, menor tamaño de grilla), lo cual se logra incrementando  $n$ .
  - El tamaño de los problemas se adapta al  $n$  disponible, y no al revés.



## Modelo de Sun y Ni o de speedup limitado por memorias

- Modelo propuesto por Xian-He Sun y Lionel Li.
- Supone que el tamaño total de memoria es limitado, y es el factor que limita el speedup.
  - En sistemas multicomputadoras (memoria distribuida), la memoria es la suma de las memorias de todas las computadoras.
    - Al sumar computadoras, se agrega memoria.
- El tamaño total del problema es variable.
- El tiempo de trabajo de cada procesador es variable.
- Ejemplos: Aplicaciones con grandes cantidades de datos, donde la memoria disponible es la limitación, no el poder de procesamiento de los CPUs.



## Modelo de Sun y Ni o de speedup limitado por memorias

- W: carga de trabajo; M: requerimiento de memoria.
- $W=g(M)$ ;  $M=g^{-1}(W)$
- $G(n)$  indica el incremento en W si la memoria aumenta n veces.
  - $G(n)=1$ , problema de tamaño fijo. Equivalente al modelo de Amdahl.
  - $G(n)=n$ , problema equivalente al modelo de Gustafson.
  - $G(n)>n$ , la carga de trabajo que el sistema puede realizar se incrementa a mayor velocidad que el incremento en memoria.

$$S = \frac{T_s + T_p * G(n)}{T_s + t_p * G(n) / n}$$

## Modelo de Sun y Ni o de speedup limitado por memorias

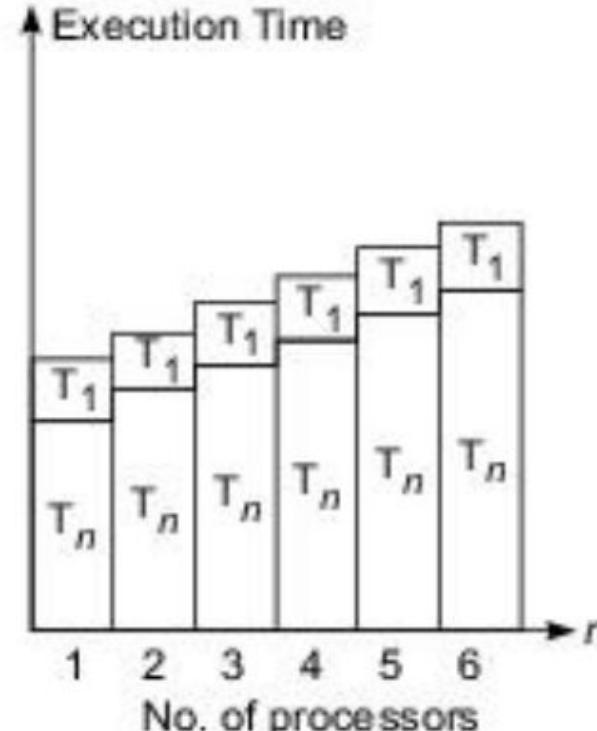
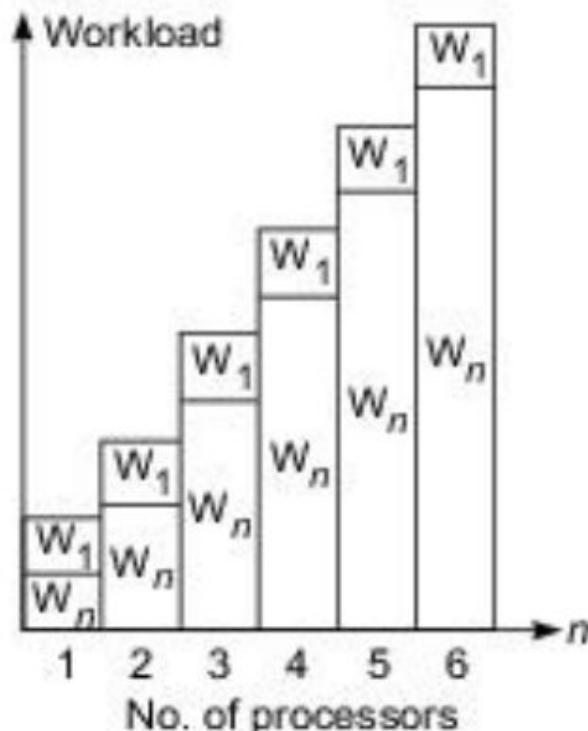
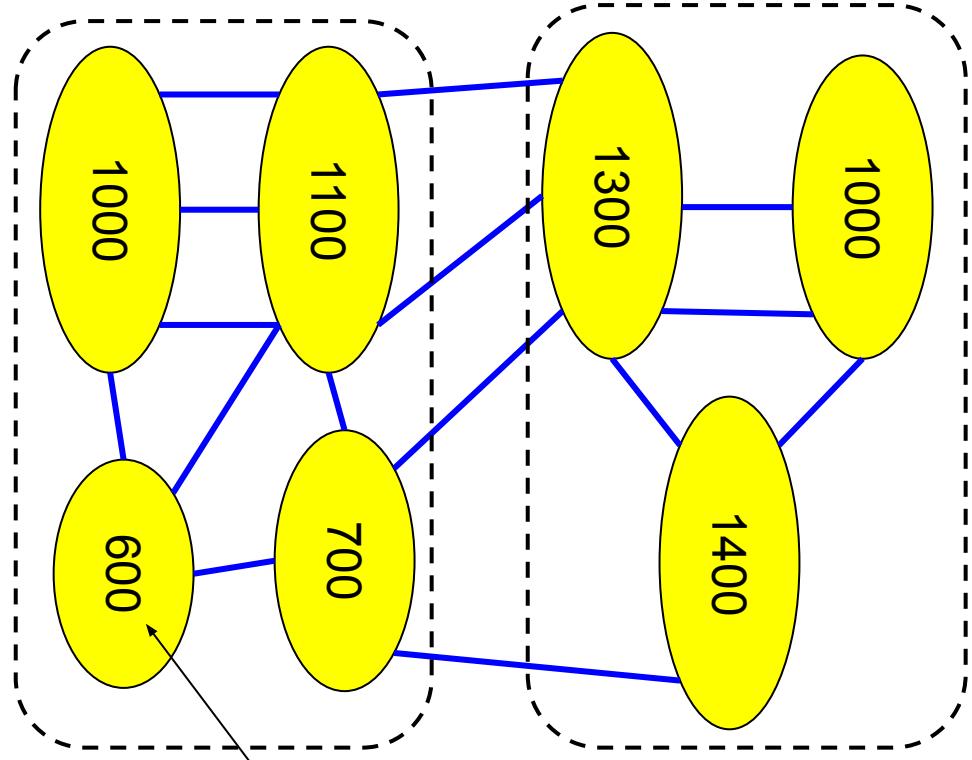


Figura obtenida de Kai Hwang, "Advanced Computer Architecture Parallelism Scalability Programmability", 2º edición, página 115

## Paralelismo en el software: Granularidad

- **Tamaño de grano:** Tamaño de los segmentos o porciones de código (número de instrucciones) que se ejecutan en paralelo para resolver una tarea.
- Varios niveles. Cada nivel de granularidad posee diferentes características (latencias, mecanismos de comunicación, herramientas de desarrollo, plataformas de hardware adecuadas, etc.).
- Mientras mayor el tamaño de los segmentos de código o granos (mayor número de instrucciones), mayor cantidad de problemas puede resolver un solo segmento de código o grano, y menor la cantidad de comunicaciones.
- Fuertemente relacionado con la latencia en la comunicación.
- La decisión del tamaño de grano depende del problema a resolver y de la o las computadoras disponibles (su hardware, sistema operativo, etc.).
- Las aplicaciones usualmente combinan varios niveles de granularidad.

## Paralelismo en el software: Granularidad y cantidad de comunicaciones



Número de instrucciones



## Granularidad

- Mientras menor la granularidad, mayor el overhead en la comunicación:
  - Motivos:
    - Es más difícil separar las tareas en segmentos de código pequeños independiente.
    - Mayor cantidad de conmutaciones entre segmentos de código.
    - Mayor necesidad de sincronizar tareas.
  - Consecuencia: **Menor la escalabilidad del sistema.**



## Paralelismo en el software: Granularidad

Miles de instrucciones

Programas multiproceso.

Miles de instrucciones

Procesos que resuelven un  
mismo problema.

Cientos o pocos miles  
de instrucciones

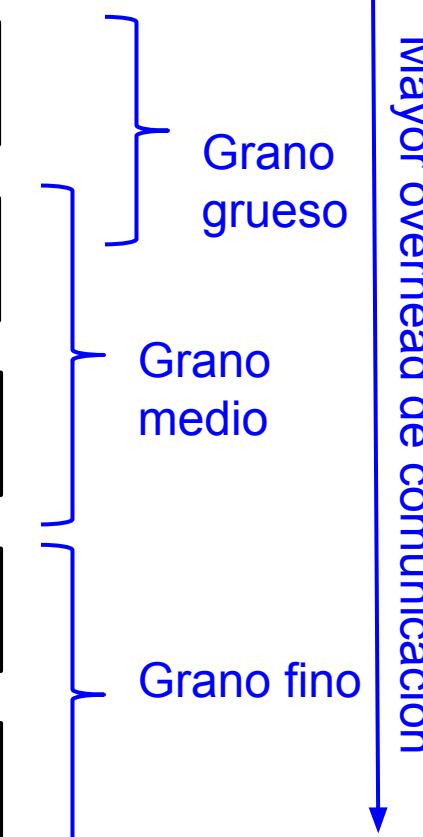
Subprocesos, hilos.

500 instrucciones

Lazos no recursivos, SIMD

2-20 instrucciones

Instrucción



Major scheduling overhead

## Paralelismo en el software: Granularidad

### Plataforma típica

Quién decide  
que paralelizar

Programas multiproceso.

Multicomputadora débilmente  
acoplada, computadora multitarea.

SO, usuario.

Procesos de una misma  
tarea.

Multinúcleo, multicomputadoras  
fuertemente acopladas (clústers).

Usuario (procesos).

Subprocesos, hilos.

Multinúcleo, procesadores con  
multihilo simultáneo. GPU

Usuario (hilos),  
compilador, SO.

Lazos no recursivos, SIMD

Procesadores SIMD, vectoriales,  
GPU.

Compilador (lazos)  
usuario (SIMD GPU).

Instrucción

Pipeline, superescalar, fuera de  
orden.

Procesador,  
Compilador.



## Paralelismo en el software: Granularidad

### Comunicación

### Herramientas de desarrollo

Programas multiproceso.

Mensajes (poca comunicación).

MPI, protocolos capa aplicación TCP/IP, sockets.

Procesos de una misma tarea.

Mensajes.

MPI

Subprocesos, hilos.

Memoria compartida.

Librerías progr multi-hilo, OpenCL, CUDA, OpenMP.

Lazos no recursivos, SIMD

Registros, memoria caché, memoria compartida.

CUDA, OpenCL, instrucciones SIMD.

Instrucción

Registros, memoria caché.

Lo realiza el procesador.



## Paralelismo en el software: Grado y perfil de paralelismo

- Grado de paralelismo del software: **Cantidad de sub-tareas** (hilos, procesos, etc.) **ejecutadas en paralelo en determinado momento, suponiendo que el hardware permite la ejecución de infinitas sub-tareas** (procesador que puede ejecutar infinitos hilos, infinitos núcleos o cluster con infinitas computadoras).
  - Es una función discreta del tiempo.
  - Es una característica del software.
    - Existen herramientas de software que permiten obtener el perfil.
  - Si excede el número de procesadores, algunas tareas paralelizables deben ejecutarse secuencialmente.
- Perfil de paralelismo: Gráfico del grado de paralelismo en función del tiempo.
- Paralelismo promedio: Grado de paralelismo promedio.



## Paralelismo en el software: Grado y perfil de paralelismo

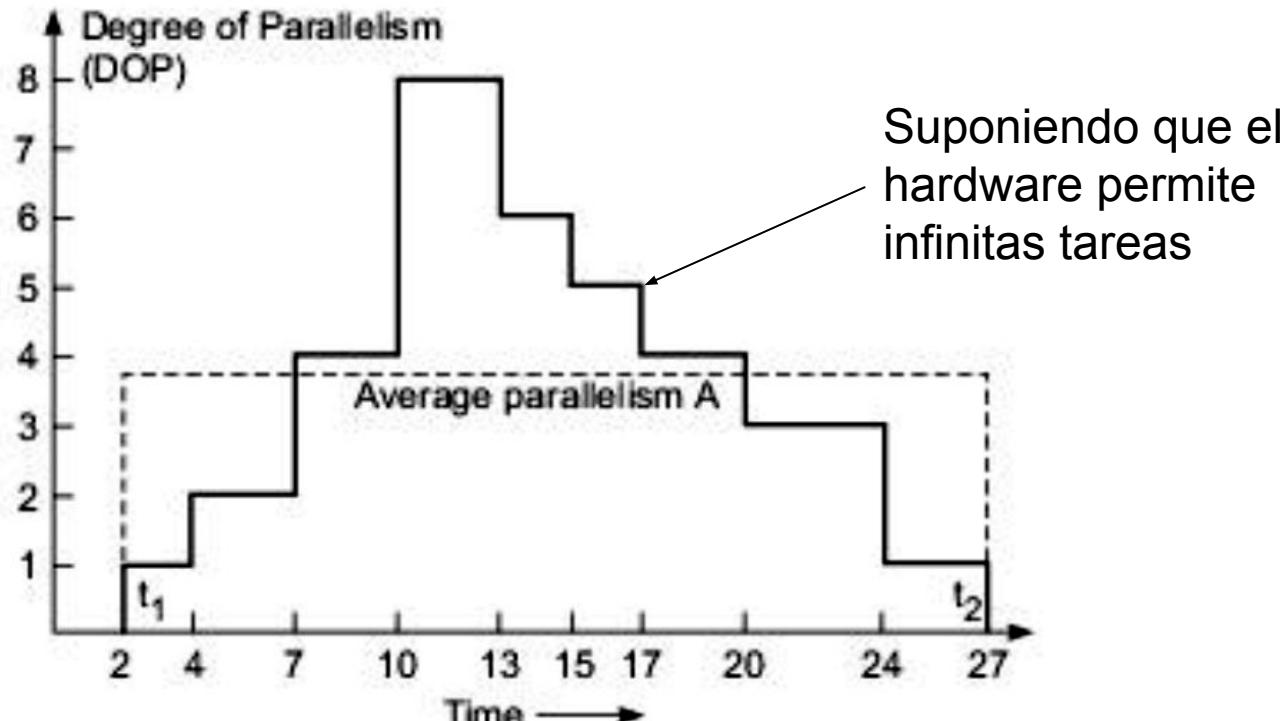


Figura obtenida de Kai Hwang, "Advanced Computer Architecture Parallelism Scalability Programmability", edición 2001, página 91



## Posibles medidas para incrementar el paralelismo en el software

- Scheduling eficiente: Que las tareas se realicen en el menor tiempo posible.
  - Problema NP-hard.
- Aumento en la granularidad: granularidad de grado fino requiere mayores retardos (comunicación, cambio de tarea, etc).
  - Empaquetar tareas de tamaño pequeño (grano pequeño) con alto nivel de comunicación en tareas de mayor tamaño (grano medio o grande).
- Duplicar tareas para ejecutar en diferentes procesadores para eliminar tiempo de comunicación.



## **Bibliografía:**

- William Stallings, "Computer Organization and Architecture", 10º edición, editorial Pearson, año 2016.
- Tanenbaum and Bos, "Modern Operating Systems", 4º edición, editorial Pearson, año 2015.
- Kai Hwang, "Advanced Computer Architecture, Parallelism, Scalability, Programmability", 2º edición, editorial Mc Graw-Hill, año 2011.
- Hesham and Mostafa, "Advanced Computer Architecture and Parallel Processing", 1º edición, editorial Wiley, año 2005.
- ARM, "ARM Cortex-A Series Programmer's Guide for ARMv8-A", Version 1.0, año 2015