Tema 11:
Caché Avanzado

Arquitectura de Computadoras

Ing. Nicolás Majorel Padilla (npadilla@herrera.unt.edu.ar)

http://microprocesadores.unt.edu.ar/arqcom/

Temas que veremos

- Medición de Performance de Caché.
- Técnicas de reducción de Tasa de Fallos.
- Técnicas de reducción de Penalidad por Fallos.
- Cachés en multiprocesamiento.
- Protocolos de Coherencia de cachés.
- Jerarquías de memoria actuales.

Lectura recomendada

- Computer Organization and Design, RISC-V Edition (2da ed, 2021):
  - Sección 5.4: *Measuring and Improving Cache Performance*
  - Sección 5.10: *Paralellism and Memory Hierarchy: Cache Coherence*
- Para los más curiosos:
  - Sección 5.13: *Real Stuff: The ARM Cortex-A8 and Intel Core i7 Memory Hierarchies*
  - Sección 5.16: *Fallacies and Pitfalls*

Repaso / Estado actual

- Vimos qué es un caché y para qué sirve.
- Cómo se ubica e identifica un bloque, en las distintas organizaciones.
- Parámetros de diseño de una memoria caché:
  - Tamaño total.
  - Tamaño del bloque (Palabras por bloque y Bytes por palabra).
  - Asociatividad.
  - Política de reemplazo.
  - Política de escritura.
- Métricas importantes:
  - Hit time, Miss rate, Miss penalty
- Los parámetros de diseño afectan de distintas maneras a las métricas, generando compromisos.

Impacto de cachés en performance

- Los cachés reemplazan a las memorias ideales de los diseños vistos en temas previos.
- Un acierto en el caché hace que el pipeline trabaje normalmente.
  - **El tiempo de acierto de los cachés es un camino crítico para un pipeline** (puede ser cuello de botella).
    - Aumentar el tiempo de acierto de un caché puede perjudicar a todas las etapas del pipeline.
- Cuando ocurre un fallo de caché, el procesador debe esperar que se resuelva.
  - Modifican el CPI introduciendo nuevas paradas.
  - **CPI = CPI ideal + (AccMem/Inst * Miss rate * Miss Penalty)**

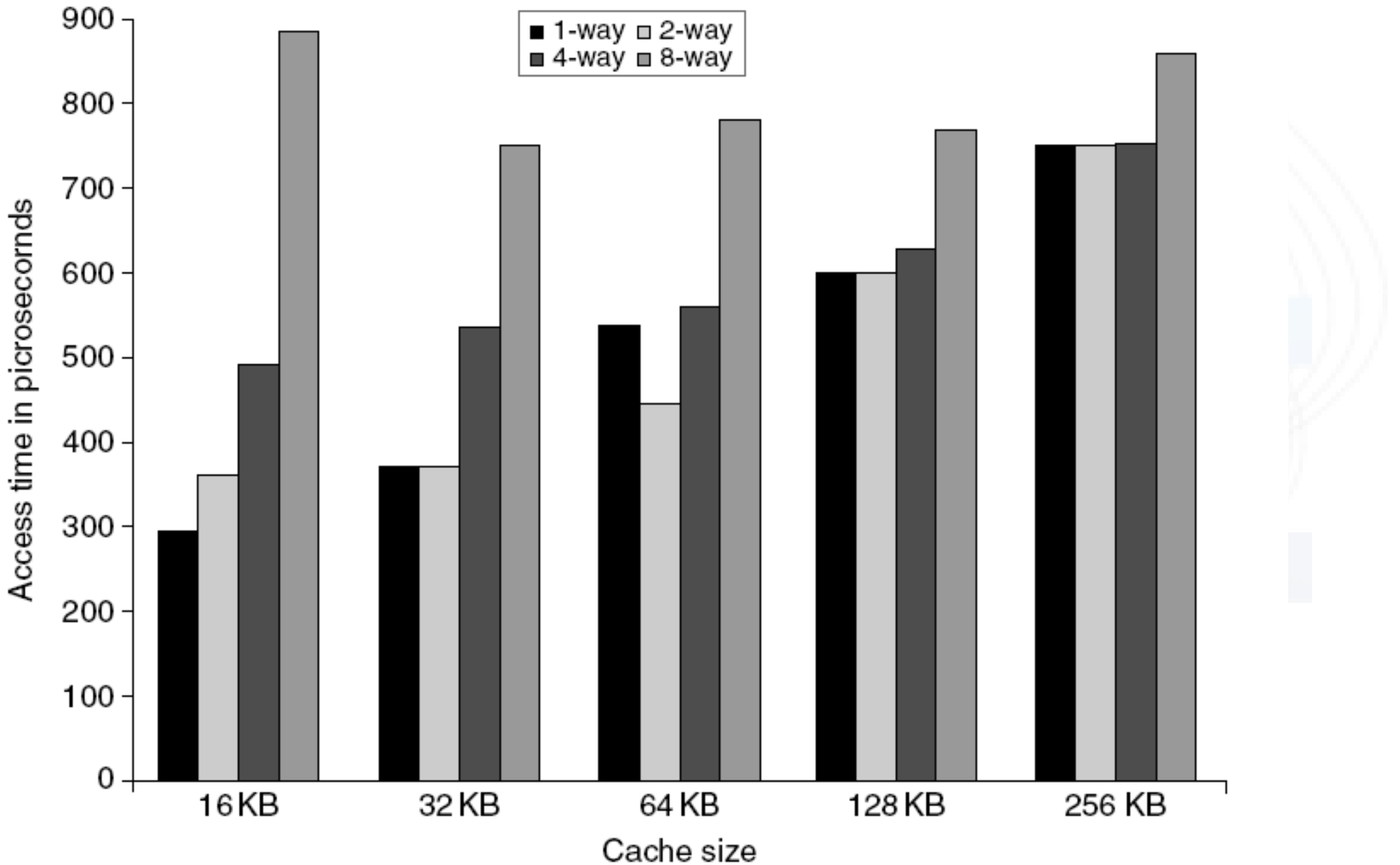
Nueva métrica: AMAT

- Para comparar jerarquías de memoria entre sí suele utilizarse una nueva métrica:
  - Tiempo medio de acceso a memoria (AMAT).
  - **AMAT = Hit Time + Miss rate * Miss Penalty**.
- Sin embargo, hay una métrica que siempre es mejor. *¿Cuál?*
- Para mejorar la performance de una jerarquía de memorias, podemos mejorar cualquiera de sus tres variables principales:
  - Reducir el tiempo de acierto.
  - Reducir la tasa de fallos.
  - Reducir la penalidad por fallos.

Reducción del tiempo de acierto

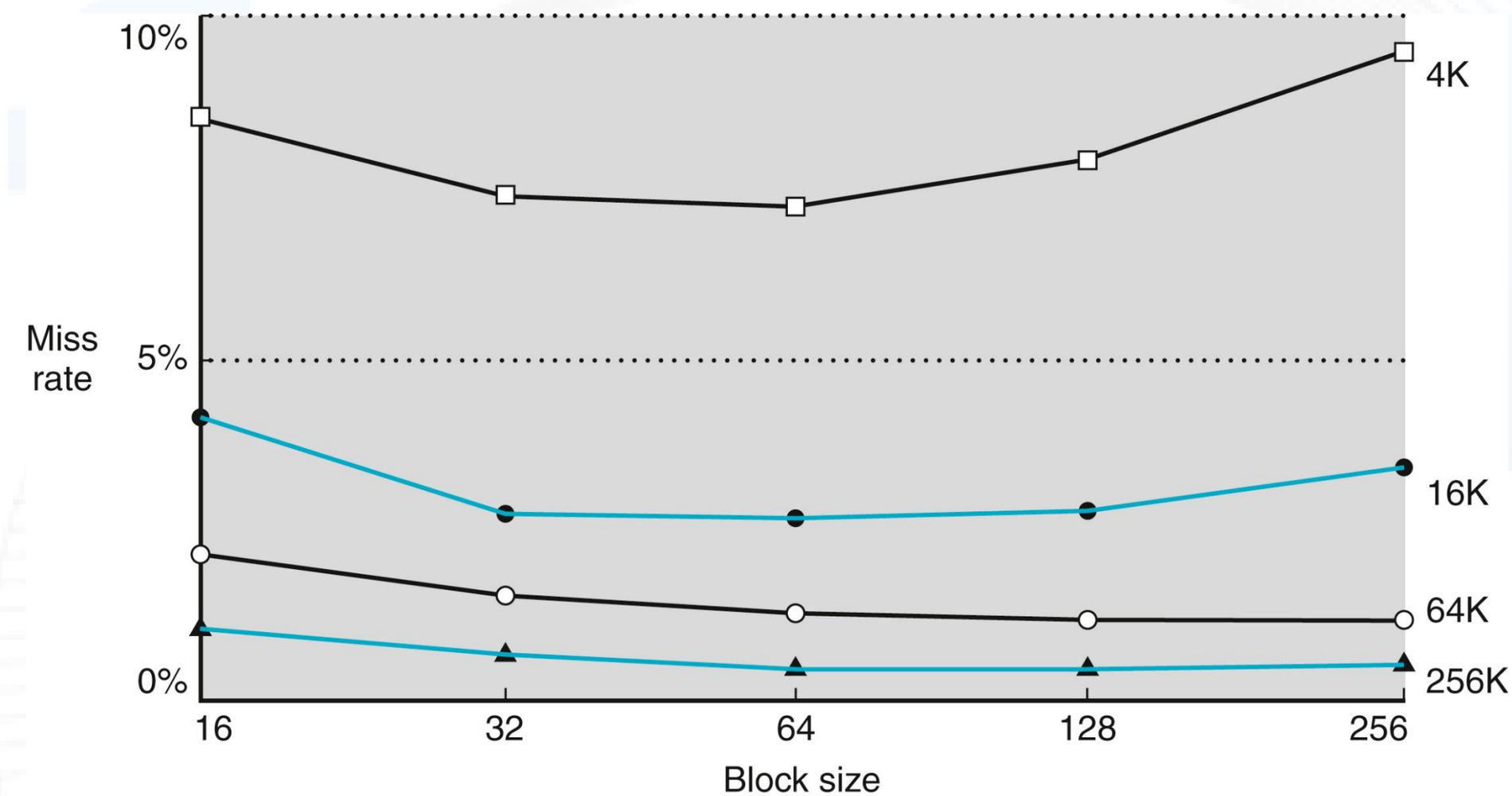
- Reduciendo el tamaño del caché.
  - Principio de Diseño: más pequeño es más rápido.
  - Pero aumenta la tasa de fallos.
- Reduciendo la asociatividad.
  - También aumenta la tasa de fallos.
- Es la más variable más difícil de reducir.
  - Lo que suele hacerse dividir el acceso en varias etapas de menor duración.
    - O sea, aplicar **Pipelining** (Gran Idea en Arquitectura de Computadoras)

Tiempo de acierto vs tamaño y asociatividad



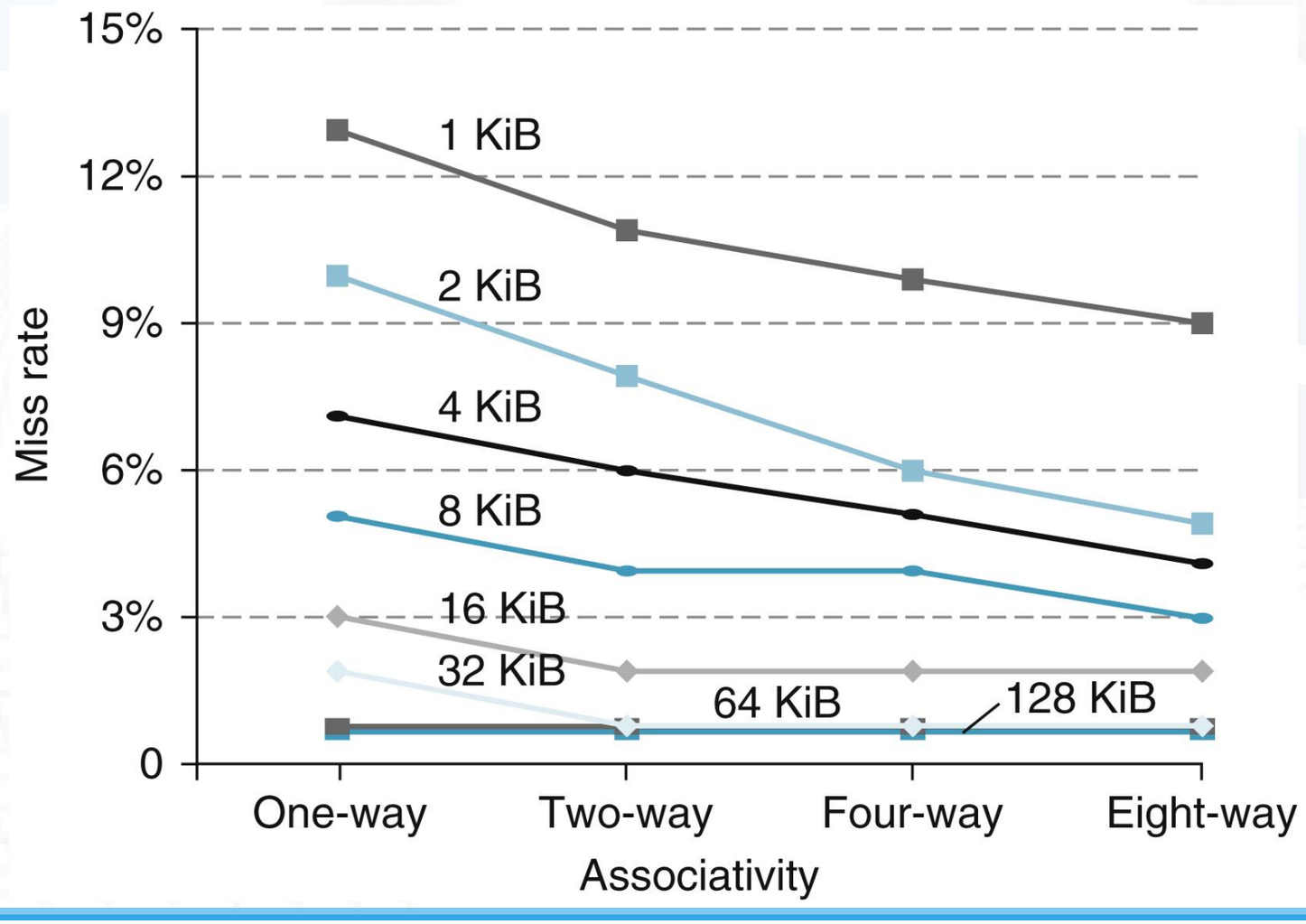
Reducción Tasa de Fallos (1/6)

- Aumentar el tamaño del bloque, para aprovechar localidad espacial.
  - Pero disminuye la localidad temporal. Compromiso.
  - También aumenta la penalidad por fallo.



Reducción Tasa de Fallos (2/6)

- Aumentar la Asociatividad, para disminuir los fallos por conflicto.
  - Pero aumenta el tiempo de acierto.
  - Aumenta el costo.



Reducción Tasa de Fallos (3/6)

- Aumentar el tamaño del caché, para disminuir los fallos por capacidad.
  - Aumenta el tiempo de acierto.
  - Más costoso y consume más energía.

- Como los fallos por Capacidad son los más comunes (según el modelo de las Tres C), esta es la técnica más usada, a pesar de sus desventajas.
  - Una variante del Principio de Diseño: **Hacer más rápido el caso más común**.

Reducción Tasa de Fallos (4/6)

- Prebúsqueda por Sw (***Software Prefetching***).
  - El ISA provee instrucciones especiales para buscar datos en memoria.
    - x86 posee cinco instrucciones PREFETCH diferentes.
    - Actúan como “tips” de localidad.
  - El compilador (o el programador) se encarga de analizar la traza de memoria e insertar estas instrucciones.
    - En tiempo de compilación, no de ejecución.
  - Aprovechar y cargar datos cuando no se esté accediendo a memoria.
- Estas instrucciones también consumen tiempo.
  - Es un compromiso evaluar si valen la pena o no.
  - Tener también en cuenta la ejecución especulativa.

Reducción Tasa de Fallos (5/6)

- Prebúsqueda por Hw (***Hardware Prefetching***).
  - Similar a la anterior, pero en tiempo de ejecución.
- En caso de fallo, buscar dos bloques (o más) en vez de uno.
  - Haciendo uso de **Paralelismo** (Gran Idea en Arquitectura de Computadoras)
  - El o los bloques extras se guardan en un buffer de “pendientes”.
- Al ocurrir un nuevo fallo, revisar también este buffer en simultáneo con el caché.
  - Si el dato buscado se encuentra en un bloque que está en el buffer, se lo carga en el caché desde allí, sin pagar penalidad.

Reducción Tasa de Fallos (5/6)

- Por otra parte, también se va analizando la secuencia de accesos a memoria y se intenta predecir el siguiente.
  - Haciendo **Predicción** (Gran Idea en Arquitectura de Computadoras)
  - Si la memoria principal está desocupada, lo van trayendo por las dudas.
  - Idea bastante similar a la predicción de saltos.
- En ambos casos, se asume que existe el suficiente ancho de banda de memoria principal para no generar paradas adicionales.
- Todos los procesadores modernos actuales tienen múltiples *prefetchers*.
  - Totalmente transparentes para los programas de usuario.

Reducción Tasa de Fallos (6/6)

- **Optimizaciones del compilador.**
- Se puede reordenar instrucciones o datos para mejorar localidad espacial.
- Unificar vectores (*merging arrays*), para mejorar localidad espacial.
- Intercambiar el orden de lazos anidados (*loop interchange*).
  - Para aprovechar el orden en que los datos están almacenados en memoria.
  - Por ejemplo, al recorrer matrices.
- Fusionar lazos que iteran sobre las mismas variables, pero con operaciones independientes (*loop fusion*).
- Dividir datos grandes en bloques, para aprovechar localidad espacial (*blocking*).
  - Por ejemplo, para multiplicar matrices o al trabajar con imágenes.

Reducción de Penalidad por fallos (1/5)

- Mejorar la tecnología de memoria principal.
  - DDR2, DDR3, DDR4, DDR5.
    - Cada vez trabajando a mayor frecuencia, aumentando el **ancho de banda**.

Tipo	Año	Frec bus	BW pico
DDR	2000	100 - 200 MHz	3,2 GB/s
DDR2	2003	400 MHz	6,4 GB/s
DDR3	2007	400 - 1067 MHz	12,8 GB/s
DDR4	2014	800 - 1600 MHz	25,6 GB/s
DDR5	2020	2400 - 3200 MHz	51,2 GB/s

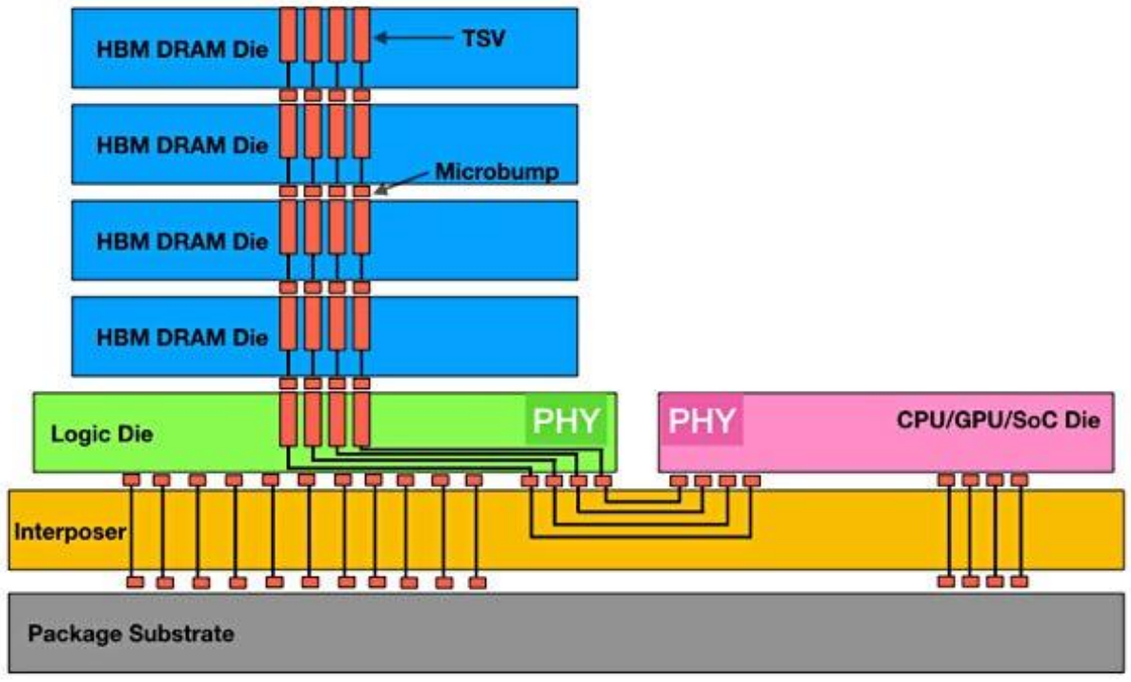
  - Se estima que el ancho de banda de memoria principal aumenta aproximadamente 23% /año.

Reducción de Penalidad por fallos (1/5)

- Actualmente, un acceso a memoria DDR4 demora aprox. 80 ns.
  - También se estima que la **latencia** de memoria principal **aumenta** un 4% /año.
    - Por problemas que veremos pronto.
- Esto es una mejora e investigación permanente.
  - Cada vez más aplicaciones son limitadas por el ancho de banda de memoria.
    - Y esto ocurre porque el tamaño de los cachés de los microprocesadores es fijo.

Memorias HBM

- Para contrarrestar las limitaciones de ancho de banda de memoria, los procesadores para muy alta performance incorporan memorias HBM (*High Bandwidth Memory*) en reemplazo de DDR.
- DRAM integrada al microprocesador directamente en el mismo *package*.
  - Como los *chiplets* que vimos en el Tema 02, pero apiladas verticalmente.
  - Permite un bus de transferencia más ancho (típicamente de 1024 bits **por *stack***).



- Ejemplos: NVIDIA Tesla P100 (HBM2, 2016), NVIDIA Hopper GH100 (HBM3, 2022) y todos los procesadores para IA.

Memorias HBM

- 18x más ancho de banda que las DDR4.
- Además reducen aprox. 70% el consumo de energía con respecto a las DDR4.
  - Por estar integradas en el mismo *package*.
- Tienen menos capacidad que las DDR, no son fácilmente escalables y son muy costosas.
  - Nuevamente, por estar integradas en el mismo *package*.
- Actualmente (2026) se está planteando que las memorias HBM están por enfrentar una *Memory Capacity Wall* y un *Memory Bandwith Wall*.
  - Por problemas de costos la primera, y de necesidades la segunda.
  - La solución a estos problemas por ahora viene dada por CXL (que veremos en el Tema 13).

Reducción de Penalidad por fallos (2/5)

- No esperar que se cargue el bloque completo.
  - ***Early restart***: se van cargando las palabras del bloque, y cuando llega la palabra solicitada se transfiere el control al procesador y se continúa la carga en paralelo.
  - ***Critical word first***: se trae de memoria la palabra solicitada primero, y se devuelve el control al procesador, continuando con la carga del resto del bloque en paralelo.
- Útiles con bloques grandes.
  - *Early restart* muy utilizado en cachés de instrucciones, ya que aprovecha mejor la secuencialidad.
- No siempre generan mejoras.
  - Si se solicita una palabra distinta del mismo bloque que se está cargando en paralelo…

Reducción de Penalidad por fallos (3/5)

- **Priorizar lecturas sobre escrituras.**
- En un caché Write-Back ocurre un fallo y hay que reemplazar un bloque modificado.
  - Lo normal sería escribir el bloque modificado en memoria, y luego traer el nuevo bloque solicitado.
  - Es mejor usar un **buffer de escritura**, copiar el bloque modificado allí y traer el nuevo bloque solicitado al caché.
    - El procesador toma el control apenas recibe el nuevo bloque solicitado.
    - El buffer de escritura escribe el bloque modificado en memoria cuando pueda.
- En un caché Write-Through puede ocurrir algo similar.
  - Riesgo RAW con alguna escritura aún en el buffer.
  - Revisar el buffer de escritura para ver si no contiene algún dato a leer.

Reducción de Penalidad por fallos (4/5)

- **Múltiples niveles de caché.**
  - Colocar entre el caché (ahora denominado primer nivel, o L1) y la memoria principal, un nuevo nivel de caché (ahora denominado segundo nivel, o L2).
  - L2 es más grande y más lento que L1, pero más rápido que la memoria.
  - L2 atiende los fallos en L1.
  - La penalidad por fallos en L1 es ahora el AMAT de L2.
- Del mismo modo, se puede colocar un caché L3 entre L2 y Memoria.
  - Genéricamente, penalidad por fallos (Ln) = AMAT (Ln+1)

Consideraciones sobre Múltiples niveles

- En el último nivel de caché (**LLC**, *Last Level of Cache*) se prioriza tener una muy baja tasa de fallos.
  - Para evitar acceder a Memoria Principal.
  - Y por el mismo motivo, suele ser de tipo *Write-Back*.
- En el primer nivel de caché, L1, se prioriza tener un tiempo de acierto mínimo.
  - Porque influye directamente en la duración de las etapas del pipeline.
  - Por el mismo motivo, suele haber cachés L1 separados para datos e instrucciones.

Reducción de Penalidad por fallos (5/5)

- Hasta ahora, todos los cachés que vimos eran bloqueantes.
  - Implica que el procesador se detenía hasta resolver el fallo.
- **Cachés no bloqueantes.**
  - Al tener un fallo, el caché continúa sirviendo accesos de otras instrucciones. **Acierto bajo fallo**.
  - El fallo se sigue procesando en paralelo.
  - La penalidad es sólo el tiempo que efectivamente está parado esperando datos.
  - Requiere que el caché tenga al menos dos puertos de acceso.
  - Usado en cachés de datos de procesadores con ejecución fuera de orden.
- Agregando más puertos de acceso al caché, podemos solapar múltiples fallos en simultáneo, y reducir aún más la penalidad efectiva.
  - Acierto bajo múltiples fallos. O fallo bajo fallo.
  - Incrementa significativamente la complejidad del caché.
  - Permite ocultar notoriamente la alta latencia de la memoria principal.

Memoria y Multiprocesamiento

- En el Tema 09 vimos que los *Multicore* actuales implementan multiprocesamiento con acceso compartido a memoria.
- Vimos los aspectos relativos a tener múltiples procesadores accediendo simultáneamente a memoria.
  - División de datos, sincronización, etc.
- Ahora, cambiamos el foco y veremos cómo afecta a la memoria ser accedida por múltiples procesadores.
- Dos aspectos:
  - Cómo se conectan los múltiples procesadores a memoria.
  - Cómo se mantiene la coherencia de los datos.

Acceso compartido a memoria

- Hay dos posibles formas de conexión:
  - **Acceso Uniforme a Memoria (UMA):** misma latencia para todos.
    - El LLC (actualmente, L3) es un bloque único, compartido entre todos los procesadores.
    - Mantienen privados el resto de los cachés (L1 y L2).
    - Como el LLC es compartido, se utiliza para intercambiar datos sin tener que acceder a la memoria.
  - **Acceso No Uniforme a Memoria (NUMA)**.
    - Los procesadores poseen porciones locales del LLC.
    - Se mantiene que el LLC es compartido.
    - Pero en este caso, la latencia de acceso no es siempre la misma.
      - Dependerá si el dato buscado está en un pedazo de LLC que está “cerca” o “lejos” del procesador que lo pide.

UMA vs. NUMA

- Inicialmente, los *multicore* tipo UMA eran los más populares.
  - Porque la memoria compartida era única, y el acceso uniforme para todos.
  - Inicialmente la memoria principal (DRAM) y luego el LLC (L3).
- En una conexión NUMA, la memoria “remota” normalmente tiene una latencia mayor que la memoria “local”.
- Sin embargo, un sistema NUMA puede proveer latencias menores.
  - Más específicamente, menores en algunos casos y peores en otros.
- Si se logra hacer que la mayoría de los accesos a memoria se correspondan con la memoria de menor latencia…
  - …¡entonces el promedio es menor!
  - Pero sin un buen manejo de NUMA podría ocurrir lo contrario.

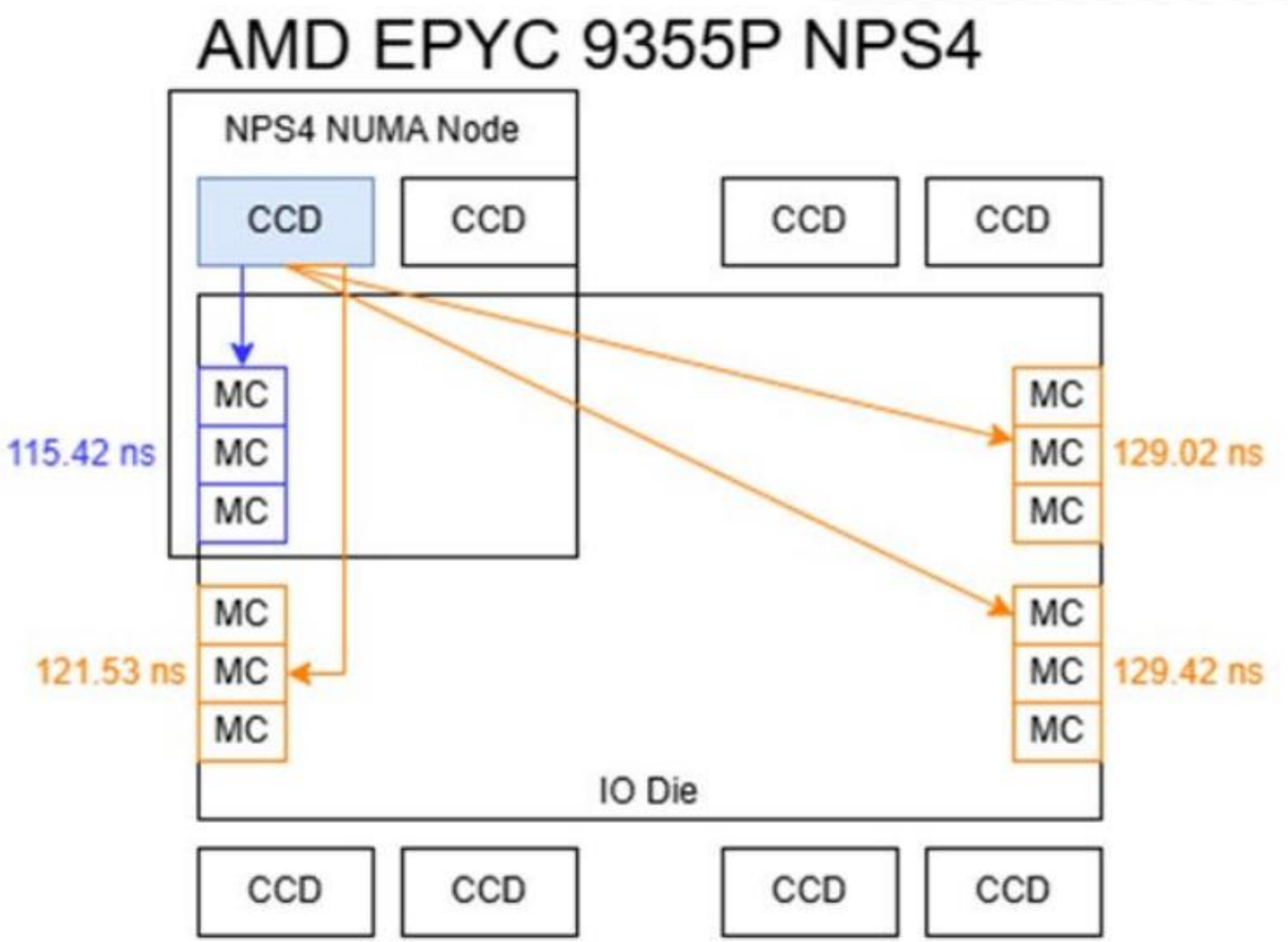
UMA vs. NUMA

- El primero en intentarlo en un *multicore* fue AMD, en 2019, con su arquitectura Zen 2.
  - Impulsado por su tecnología de *chiplets*, vistas en el Tema 02.
- A partir de entonces, son cada vez más populares las conexiones tipo NUMA en los sistemas *multicore*.
- Es crítico que los sistemas NUMA tengan soporte del SO.
  - Para definir afinidades de procesador o de memoria: forzar a un programa a ejecutarse en un determinado procesador.
  - Para hacer un balance de carga automático: mover un programa a donde están sus datos.

Nodos NUMA

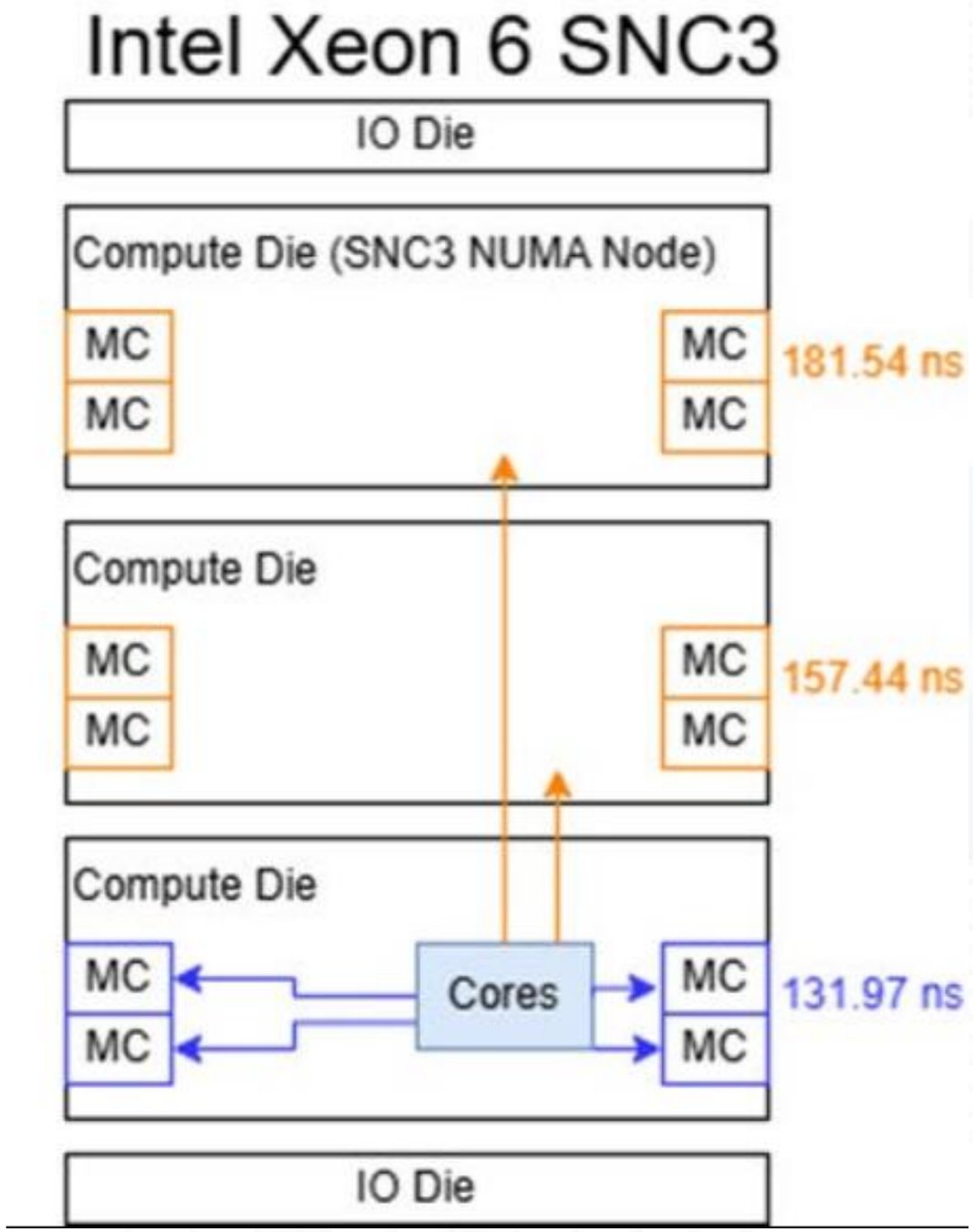
- Se define como Nodo NUMA al conjunto de núcleos con su correspondiente memoria local.
- Los procesadores modernos permiten configurar cuántos nodos NUMA poseen.
  - Esta configuración se realiza desde la BIOS.
  - Por ejemplo, AMD define modos NPS 1/2/4 (*Nodes Per Socket*).
    - NPS1 implica UMA: todos los núcleos en el mismo nodo.
    - NPS2 divide la cantidad de núcleos en dos mitades, y NPS4 en cuatro cuadrantes.
  - Intel tiene un esquema similar, pero con diferente nombre (SNC, *Sub-NUMA Clustering*).
  - Dentro de RISC-V, Ventana utiliza una configuración bastante similar a los NPS de AMD.

Nodos NUMA – Ejemplo AMD



- Cada CCD representa 4 núcleos en SMP.
- Cada MC representa un controlador de memoria.
- Se observa la división en cuadrantes.
- Se observa la diferencia en las latencias.

Nodos NUMA – Ejemplo Intel



- El esquema es diferente, por la organización de los *chiplets*.
- Se observa la diferencia en las latencias.

Nodos NUMA – Mapas de calor

- Para definir las latencias entre los distintos nodos NUMA, se realizan pruebas de estrés que determinan mapas de calor como el siguiente:

	Mem Node			
CPU Node	0	1	2	3
0	115.488	122.624	129.312	129.024
1	122.784	116.704	129.92	128.192
2	137.6	137.504	121.696	128.384
3	130.592	133.088	127.584	116.64

- La idea principal es que los programas se ejecuten en las zonas verdes, que suelen ser diagonal, y evitar las zonas rojas.
- En caso de un esquema UMA, el mapa es todo del mismo color.

Problema de Coherencia de Cachés

- Como la memoria es compartida entre los múltiples núcleos, es necesario que se provea un mecanismo para mantener la **coherencia de datos**.
- Coherencia es mantener la ilusión de que los datos son almacenados en una única gran memoria.
- Problema: dos (o más) procesadores diferentes podrían acceder a la misma dirección de memoria compartida y obtener valores diferentes.
  - *¿Cómo es posible?*

Problema de Coherencia de Cachés

- Un ejemplo con dos procesadores (núcleos) con cachés *write-through* (*write-back* presentan un comportamiento similar):

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A stores 1 into X	1	0	1

- ***¿Qué sucede si luego CPU B quiere leer X?***
- ***Y si un CPU C quiere luego leer X, ¿qué valor debería leer?***

- En caso que el CPU B desee leer el valor de X, debe proveerse un mecanismo para que lea el valor correcto, desde memoria o desde donde sea.

Protocolos de Coherencia de Cachés

- Estos mecanismos se denominan **protocolos de coherencia de caché**.
- Se basan en llevar un registro de cómo y cuándo es compartido cada bloque de caché.
  - Cada bloque mantiene información sobre su compartición junto con la de su estado.
- Los protocolos de coherencia más populares son de tipo ***snooping***.
  - Todos los cachés son accesibles desde un **único bus compartido**.
  - Las transacciones en ese bus se realizan de a una por vez.
  - Todos los cachés monitorean permanentemente ese bus compartido para saber si tienen una copia del bloque que se está solicitando.
    - Si detectan que en el bus hay una transacción sobre un bloque del cual tienen una copia, entonces realizan alguna acción.

Protocolo simple de Coherencia

- Cuando un procesador A escribe en una dirección de memoria X, se marcan como inválidas todas las copias de X que puedan existir en los cachés de los demás procesadores (***write invalidate***).
- Esto implica que una escritura hace que un dato sea exclusivo de un procesador.
- Genera nuevos fallos, porque puede que el procesador B haya leído antes el dato X, y al intentar leerlo nuevamente no lo encuentra.
  - Estos son los fallos por Coherencia (la cuarta ‘C’ que habíamos mencionado en el Tema 10).

Protocolo simple de Coherencia

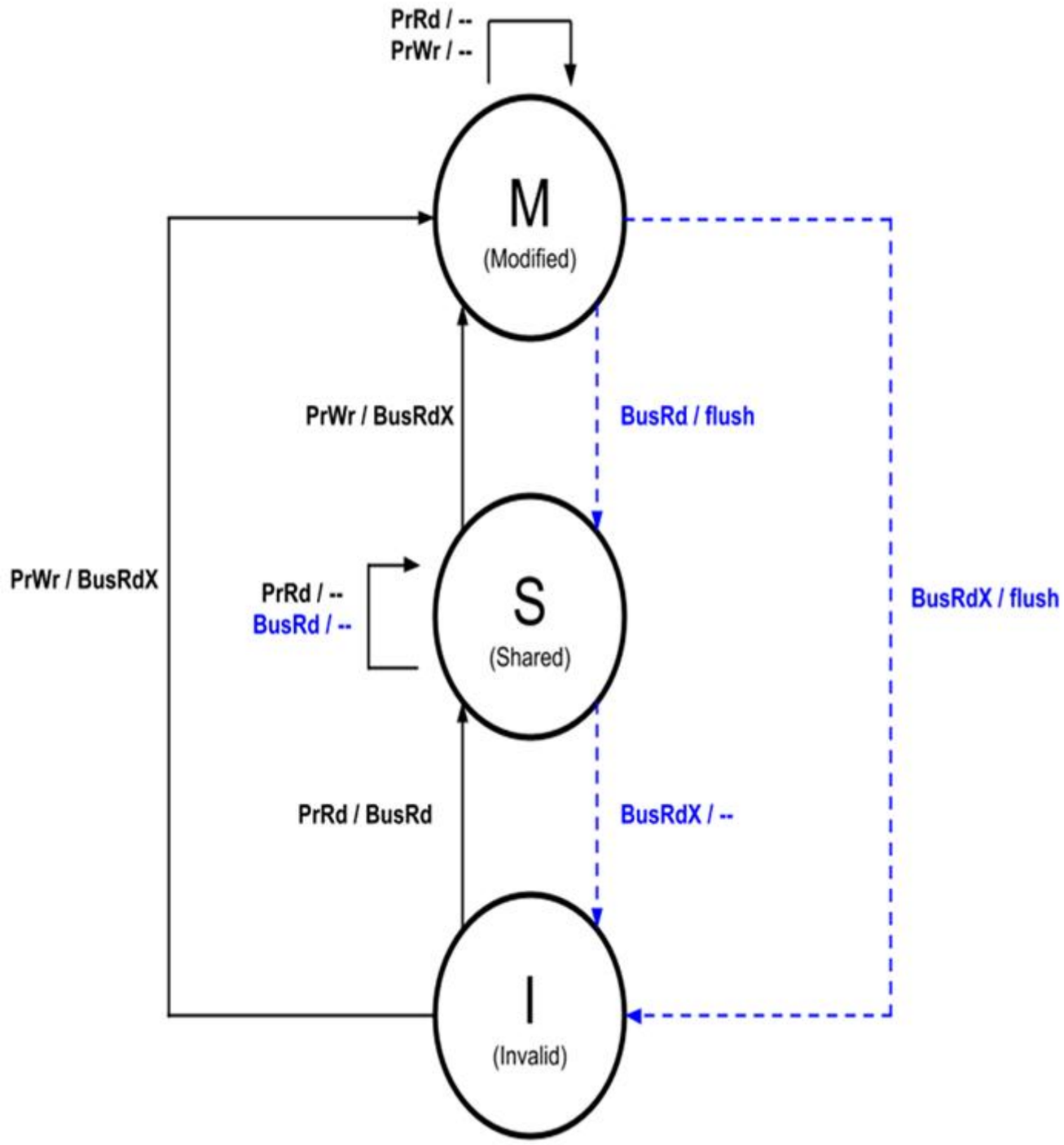
- Si un procesador B quiere leer una dirección de memoria X que es exclusiva de otro procesador A, se fuerza a que A haga un *write-back* a memoria de X y luego se produce la lectura.
  - En realidad, cuando A hace el *write-back*, al mismo tiempo le pasa el valor de X a B, porque el bus es compartido.
- Repasando el ejemplo visto anteriormente (ahora con cachés *write-back*):

Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

Protocolo simple de Coherencia

- Estos protocolos son usualmente implementados como una MEF en el LLC de cada procesador.
  - Estrictamente, en el límite entre memoria privada y memoria compartida.
  - Últimamente, **son implementados por las mismas interconexiones on-chip**.
- En la versión más simple, un bloque puede estar en uno de tres estados:
  - Inválido (I): el caché no posee este bloque.
  - Modificado (M): el bloque es exclusivo, sólo este caché tiene la versión válida.
  - Compartido (S): igual que en memoria, puede estar en otros procesadores.
- En una versión mejorada, se agrega un cuarto estado para evitar algunas transacciones por el bus: Exclusivo, pero no modificado.
  - El protocolo pasa a denominarse MESI, y es bastante popular.
- Hay más versiones, que agregan más estados.
  - P.ej: los procesadores Ryzen de AMD implementan un protocolo de 7 estados, MDOESFI.
- Además, hay otros protocolos que no son *write-invalidate*, sino *write-update*.

MEF para Protocolo de Coherencia MSI



- Transiciones en negro son generadas por el procesador (Pr).
- Transiciones azules son generadas por el Bus (otros procesadores).
- Cada transición muestra el evento que la genera / su resultado.
- Las transiciones al estado M son por escrituras (acierto o fallo).
  - Invalidan el bloque en los demás cachés.
- Cuando un caché tiene un bloque en estado M que es solicitado en el bus, hace un write-back (*flush*).
  - Y pasa el bloque al estado S.
- BusRd vs BusRdX: el primero es por fallos de lectura, el segundo por fallos de escritura (invalida el bloque en los demás caches).

Problema de Falsa coherencia

- Adicionalmente, puede haber otro problema en bloques de múltiples palabras.
- Los movimientos en caché siempre son a nivel de bloques.
- Puede ocurrir que dos variables compartidas no relacionadas X e Y estén ubicadas en el mismo bloque.
  - El procesador A escribe siempre X y el procesador B escribe siempre Y.
  - Ambos están permanentemente invalidando todo el bloque.
  - Generan fallos innecesarios y tráfico por el bus compartido.
- Se puede evitar teniendo mucho cuidado al ubicar las variables en memoria.

Ejemplos de jerarquías actuales (1/4)

Caché	Caract.	ARM Cortex X2	Samsung M4	Intel Core Ultra 9 285K	AMD Ryzen 9 HX 370	Apple M1 Firestorm	IBM Telum II
	Año	2023	2019	2024	2024	2020	2024
L1 Inst (por núcleo)	Tamaño	64 KiB	64 KiB	64 KiB	32 KiB	**192 KiB**	128 KiB
	Asociat.	4-way	4-way	16-way	8-way	6-way	8-way
	Bloque	64 Bytes	128 Bytes	64 Bytes	64 Bytes	64 Bytes	**256 Bytes**
	Escritura	Write-back		Write-back	Write-back		
	Reemplazo	LRU					
	Fallos simult.	20	12		**124**		
	Hit Time	4T	4 T	4 T	4 T	**3 T**	
L1 Datos (por núcleo)	Tamaño	64 KiB	64 KiB	48 KiB	48 KiB	**128 KiB**	128 KiB
	Asociat.	4-way	8-way	12-way	12-way	8-way	8-way
	Bloque	64 Bytes	64 Bytes	64 Bytes	64 Bytes	64 Bytes	256 Bytes
	Escritura	Write-back		Write-back	Write-back		
	Reemplazo	LRU					
	Fallos simult.	20	12		**124**		
	Hit Time	4T	4 T	4 T	4 T	3 T	

Ejemplos de jerarquías actuales (2/4)

Caché	Caract.	ARM Cortex X2	Samsung M4	Intel Core Ultra 9 285K	AMD Ryzen 9 HX 370	Apple M1 Firestorm	IBM Telum II
	Año	2023	2019	2024	2024	2020	2024
L2 (por núcleo)	Tamaño	1 MiB	512 KiB	3 MiB	1 MiB	12 MiB	**36 MiB**
	Asociat.	8-way	8-way	12-way	**16-way**	12-way	8-way
	Bloque	64 Bytes	64 Bytes	64 Bytes	64 Bytes	128 Bytes	**256 Bytes**
	Escritura	Write-back		Write-back	Write-back		
	Reemplazo	LRU					
	Fallos simult	46				150	
	Hit Time	11 T	12 T	17 T	14 T	16 T	20 T
L3 (compartido)	Tamaño	6 MiB	4 MiB	36 MiB	24 MiB	**NO TIENE**	**360 MiB**
	Asociat.	16-way	16-way	12-way	16-way		**32-way**
	Bloque	64 Bytes	64 Bytes	64 Bytes	64 Bytes		256 Bytes
	Escritura	Write-back		Write-back	Write-back		
	Reemplazo	LRU					
	Fallos simult	94		128	45		
	Hit Time	51 T	37 T	84 T	50 T		63 T

Ejemplos de jerarquías actuales (3/4)

- En general, todos estos microprocesadores tienen un L1 con un tiempo de acierto de 4 ciclos.
  - Disimulan la latencia con múltiples etapas (***superpipelining***) y siendo no bloqueantes, pudiendo iniciar nuevos pedidos en cada ciclo.
- Se nota una tendencia de aumento en los tiempos de acierto de los L3, medido en ciclos. ***¿Por qué?***
- El L3 del procesador Telum II de IBM **es virtual, no físico**.
  - Compuesto por 8 L2 de 36 MB de cada núcleo, más 2 extras.
  - Al sacar un bloque de un L2, va a parar a otro L2 que se use menos.
  - El Telum II también posee un caché L4 virtual, de 2.8 GiB, compuesto por 8 L3 virtuales, con un acceso de 48.5 ns (267 T) .

Ejemplos de jerarquías actuales (4/4)

- Es destacable que el Apple Firestorm no posee caché L3.
  - Lo compensa con los L1 y L2 más grandes.
- Intel agrega en algunos de sus procesadores un **caché L1.5**:
  - Antes del L2, de 192 KiB, 12 vías y 9 T de tiempo de acceso.
- Por otra parte, se especula que Intel lanzará procesadores en 2026 con un bLLC (*big* LLC) de 144 MiB, en su nueva arquitectura Nova Lake.
  - Por encima del L3, o sea que podría funcionar como un L4

Otro enfoque: *Waferscale*

- *¿Recuerdan el chip de Cerebras que vimos en el Tema 02?*
  - Posee 40 GiB de memoria SRAM integrada.
  - Tiempo de acierto 1 T. Frecuencia de clock = 850 MHz.
- No posee jerarquía de memoria. Es un modelo plano.
  - Evita retardos de la jerarquía.
    - Es como si no tuviese fallos.
- Provee un enorme ancho de banda a memoria.
  - Siempre y cuando todo el *working set* entre en la memoria.
- Es otra manera de solucionar la *Memory Wall*, directamente llevando la memoria dentro del procesador.
  - Pero recordemos que no es una solución genérica, sino que tiene una única función, resuelve un solo tipo de problemas.

Resumen final

- La jerarquía de memoria tiene un gran impacto en la performance
  - Hit Time (de L1) influye directamente en la duración del ciclo T.
  - Miss Rate y Miss Penalty influyen directamente en el CPI.
- AMAT como métrica para comparar jerarquías de memoria.
  - El AMAT del nivel n+1 es la penalidad por fallo del nivel n.
- 6 técnicas para reducir la tasa de fallos.
  - Hacer los cachés lo más grandes que se pueda, lo más asociativos que se pueda y usar *prefetching*.
- 5 técnicas para reducir la penalidad por fallos.
  - Cachés multinivel, cachés no bloqueantes, y mejoras continuas en la tecnología de memoria principal.

Resumen final

- Cachés con múltiples procesadores, dos problemas nuevos: cómo conectarlos y mantener la coherencia.
- NUMA es más complejo que UMA, pero más utilizado en la actualidad, por brindar mejor performance.
- La coherencia se mantiene mediante protocolos, implementados como MEF en las interconexiones entre memoria privada y compartida.
- El diseño de la jerarquía de memorias es un factor crítico de diferenciación entre las arquitecturas modernas.