Clase 03: Cómputo Monoprocesador (continuación)

IIC3533 Computación de Alto Rendimiento

Ignacio Labarca – ijlabarca@uc.cl

Departamento de Ciencia de la Computación
Escuela de Ingeniería
Pontificia Universidad Católica de Chile

Martes 12 de Agosto del 2025

Repaso

Cómputo Monoprocesador

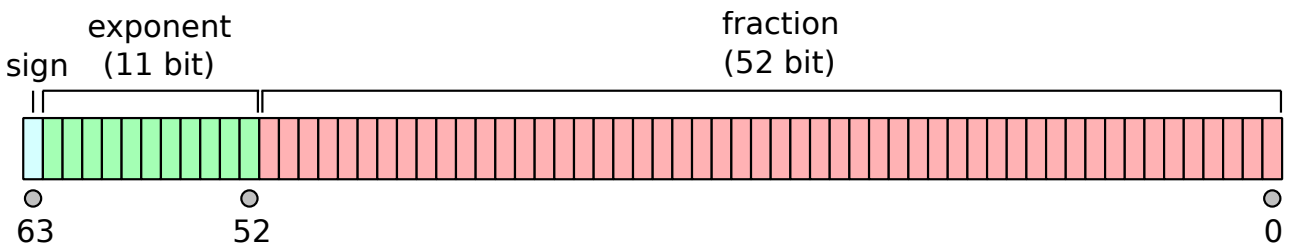
Repaso

Cómputo Monoprocesador

Sistemas de Cómputo Intensivo

Métricas en supercomputadores:

- **FLOP**. Floating Point Operation en precisión doble (8 byte).
- **FLOPS**, o también **FLOP/s**: FLOP por segundo.
- **Byte**. Tamaño de los datos. Precisión doble: 8 byte. Simple: 4 byte.
- **Bit**. Unidad básica. 1 Byte = 8 Bit.



¿Por qué es importante?

Ejecutar código eficientemente depende de

- Algoritmo utilizado.
- Aprovechar la arquitectura.

¿Por qué es importante?

Ejecutar código eficientemente depende de

- Algoritmo utilizado.
- Aprovechar la arquitectura.

Tiempo de ejecución puede variar incluso órdenes de magnitud.

Para algunos problemas, **un único procesador** es suficiente.

Repaso

Cómputo Monoprocesador

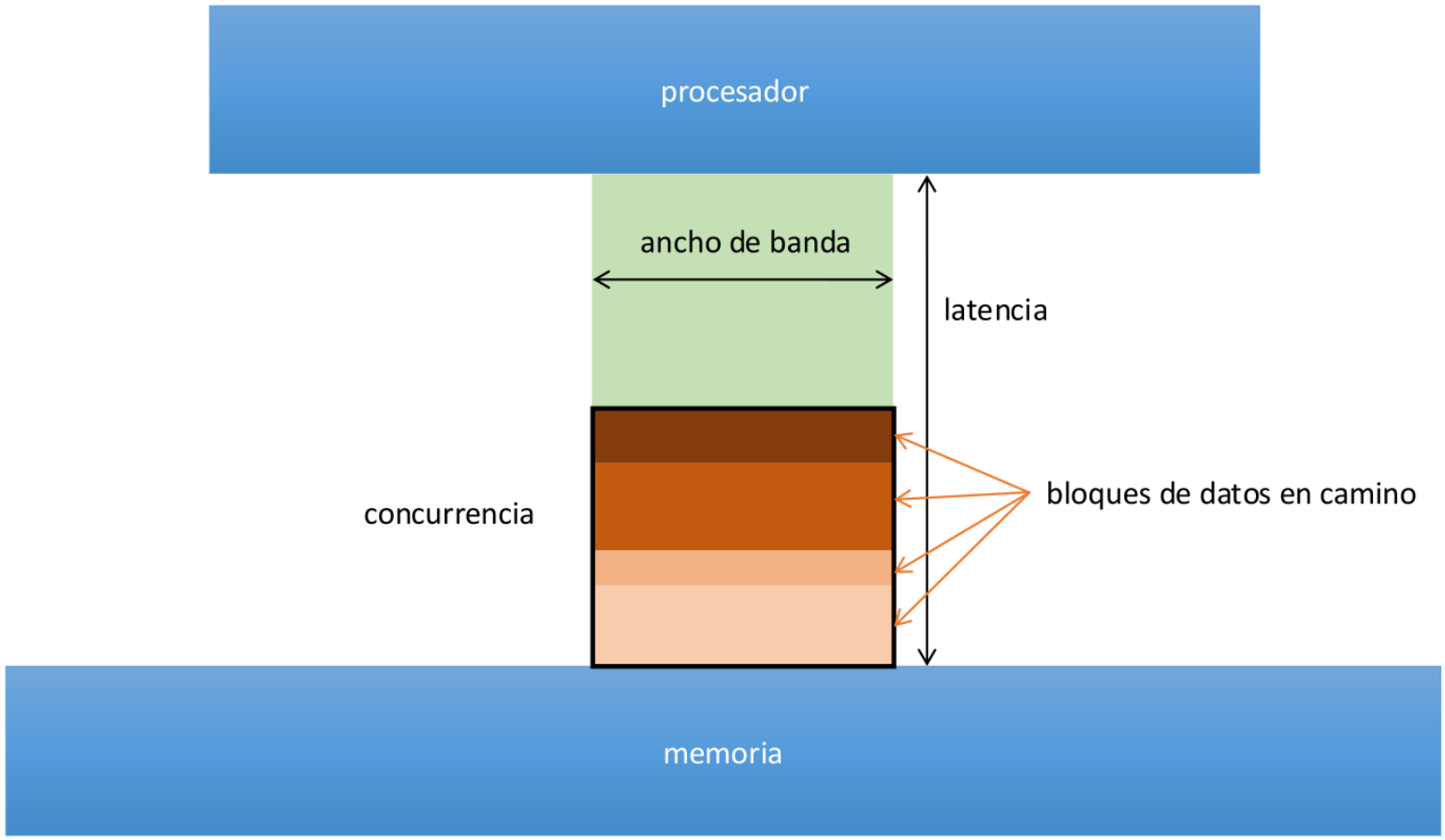
Arquitectura Von Neumann

Ciclo:

- **Fetch**. Traer.
- **Execute**. Ejecutar.
- **Store**. Guardar.

En la práctica, el ciclo es más complicado.

Flujo de Datos



Arquitectura Von Neumann

Ciclo:

- **Instruction Fetch**.
  Cargar instrucción en el procesador de acuerdo a *Program Counter*.
- **Instruction Decode**.
  Determinar qué operación es y qué operandos necesita.
- **Memory fetch**.
  Traer operandos desde memoria a registros (si es necesario).
- **Execute**.
  Ejecuta la operación, leyendo y escribiendo registros.
- **Write-back**.
  Guarda el resultado en memoria (si es necesario).

Arquitectura Von Neumann

El funcionamiento básico es

$$\text{Fetch} \longrightarrow \text{Decode} \longrightarrow \text{Memory fetch} \longrightarrow \text{Execute} \longrightarrow \text{Write-back}$$

pero los computadores modernos introducen varios elementos:

ILP: Instruction Level Parallelism

- **Localidad**. Luego de leer una dirección, la siguiente es más rápida.
- **Caché**. Datos quedan en jerarquías intermedias de memoria.
- **Pipelining**. Varias instrucciones se procesan al mismo tiempo en distintas etapas.

En general se ejecuta una instrucción por ciclo.

Arquitectura Von Neumann

La velocidad del procesador parece ser la métrica principal, pero

- Algoritmos *cpu-bound*: limitados por velocidad del procesamiento.
- Algoritmos *memory-bound*: limitados por velocidad de acceso a memoria.

Pipelining

Pipelining

Tenemos un *pipeline* de 5 etapas, para 4 instrucciones.

Se realiza según los ciclos de reloj del procesador.

Instr.	Pipeline stage							
1	IF	ID	MEM	EX	WB			
2		IF	ID	MEM	EX	WB		
3			IF	ID	MEM	EX	WB	
4				IF	ID	MEM	EX	WB
Cycle:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

¿Cuánto gano con un *pipelining*?

FPU Tradicional: Obtener n resultados toma $T(n) = n \times \ell \times \tau$

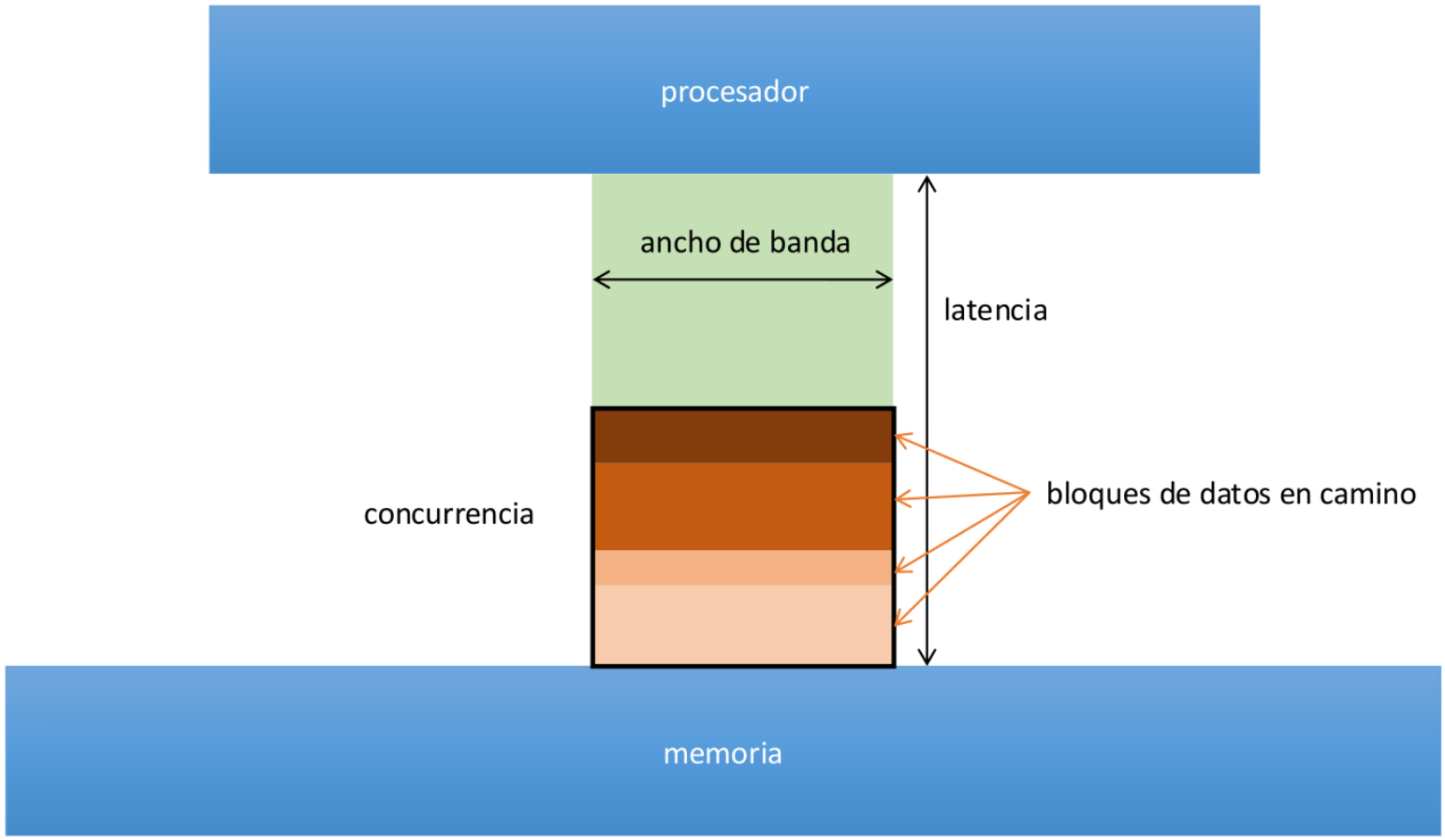
- ℓ : número de etapas
- τ : *clock cycle time*.
- Tasa de resultados es $n/T(n)$, $r_{\text{serial}} = \frac{n}{T(n)} = \frac{1}{\ell \times \tau}$.

Pipelined FPU: $T(n) = (s + \ell + n - 1) \times \tau$.

- s : costo de inicialización (*setup*).
- La primera operación recorre todo el *pipeline*.
- Después, las siguientes terminan una a la vez.
- Tasa de resultados:

$$\frac{n}{T(n)} = \frac{n}{(s + \ell + n - 1)\tau} \longrightarrow \frac{1}{\tau}$$

Flujo de Datos



Movimiento de datos

Acceso a memoria es la principal limitante.

- **Jerarquías de memoria**.
- **Latencia**: tiempo desde la solicitud del dato, hasta que el dato llega.
  - Transferencia entre memorias. Medida en ciclos o en *ns*.
  - *Memory stall*: procesador está bloqueado esperando datos.
- **Ancho de banda**
  - Tasa de llegada de datos.
  - Posterior a periodo de latencia inicial.
  - Medido en (kilo, mega, giga) Byte/s o en (kilo, mega, giga) Byte/ciclo.
  - Incluye: velocidad del canal (*bus speed*), ancho del bus, canales por *socket*, cantidad de *sockets*.
- **Tiempo de transferencia**: $T(n) = \alpha + \beta n$
  - α: latencia, β: inverso del ancho de banda (tiempo por byte).

Caché

- Registros se acceden mediante instrucciones de *assembler* .
  - Los registros son la unidad básica de memoria en el procesador.
- Movimientos caché-memoria se hacen por hardware .
  - No podemos intervenir en cómo se transmite a través del *software*.
- Menor margen para intervenir en el uso de caché vs uso de registros.

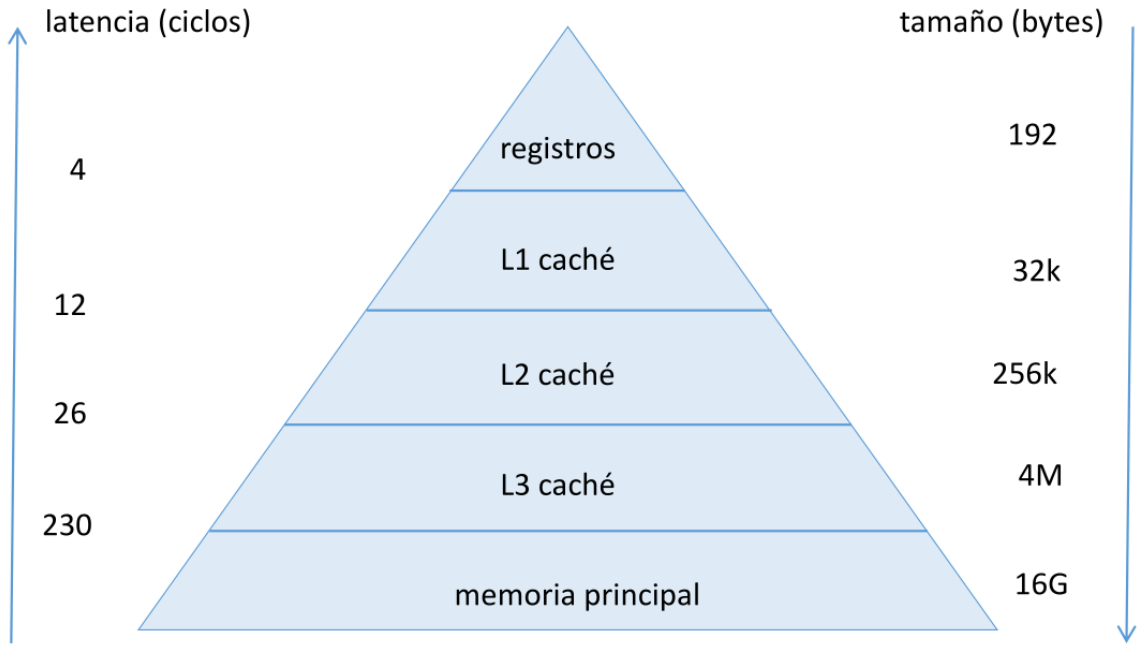
Caché

- El caché es el punto intermedio entre registros y memoria principal.
- Existen varios **niveles** de memoria caché.
- Se caracterizan por baja latencia y alto ancho de banda.
- Los datos se mantienen en la memoria caché de manera temporal.

Memoria Principal $\longrightarrow$ Memoria caché $\longrightarrow$ Registros

Caché

- L1 tiene **32 bytes de ancho de banda**, 2 operandos para 2 operaciones, 4 operaciones por ciclo.
- Existe L1d para datos y L1i para instrucciones.
- L1 y L2 suelen estar en el chip del procesador.



Caché

La ventaja de tener memoria caché surge con reutilizar datos.

- Datos que ya están en memoria caché se acceden más rápido.
  - **Localidad temporal**. Útil para datos que se volverán a usar proximamente.
- Memoria caché es menor, los datos no se pueden quedar para siempre.
  - Se debe dar lugar a nuevos datos, por lo que es necesario sobreescribir.
- Datos se copian en cada nivel en su viaje a los registros.
  - Cada nivel funciona como un *checkpoint* para guardar los datos.

Caché: hit / miss

- Diremos *cache hit* cuando solicitemos datos que ya se encuentran en caché.

  Ejemplo: Solicitamos el elemento *x[0]* y con eso la línea de caché trae elementos contiguos de memoria.

  Posteriormente, solicitamos *x[1]*, que ya se encuentra en L1.

- Diremos *cache miss* cuando no podemos encontrar el dato en caché.

  Ejemplo: Solicitamos el elemento *x[0]* y con eso la línea de caché trae elementos contiguos de memoria.

  Posteriormente, solicitamos *x[100]*, que **no** se encuentra en caché.

Caché miss

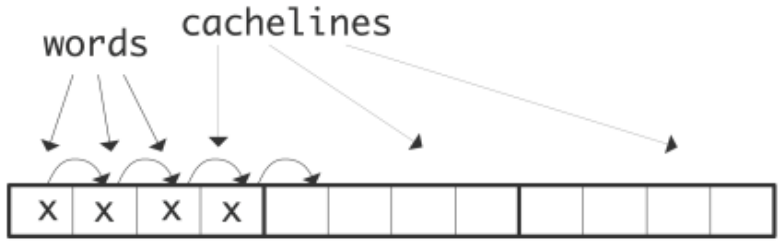
Existen distintos casos:

- **Compulsory cache miss**.
  Dato faltante .
- **Capacity cache miss**.
  Caché está lleno y hay que sobreescribir.
- **Conflict miss**.
  Ambos datos son necesarios y se quieren almacenar en el mismo lugar .
- En multicore: **invalidation miss**.
  El valor ha sido actualizado por otro núcleo .

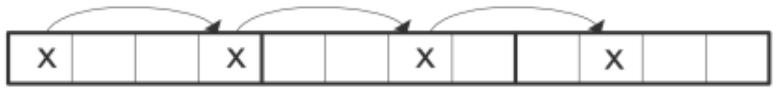
Patrones de acceso

Datos de caché no se leen individualmente, sino por bloques.

- Los datos se mueven en una **línea de caché** (*cacheline*).



- La línea de caché suele ser de 64 o 128 bytes (8 ó 16 números de punto flotante).
- Código suele usar localidad espacial.
- Precaución con los saltos en un for-loop:



Cache mapping

Si un dato se guarda en el caché, ¿dónde se ubica?

Dirección memoria principal $\longrightarrow$ Dirección memoria caché

- ¿Cómo se determina en qué lugar de la memoria caché almacenaremos un determinado dato proveniente de la memoria RAM?
- No es posible asignar a cada dato un lugar distinto, porque RAM es mayor que Caché.
- Necesitamos un **mapeo** que, según de dónde viene el dato, asigne un lugar en la memoria caché.
- Este mapeo forma parte del **hardware**. ¿Qué puede significar esto?

Cache mapping

Si un dato se guarda en el caché, ¿dónde se ubica?

Dirección memoria principal $\longrightarrow$ Dirección memoria caché

- Mapeo directo
- Dirección en memoria principal: 32 bits
  (para distinguir entre GB de RAM).

  Es el número **identificador** de una dirección en la memoria.
- Dirección en caché: 16 bits
  (para distinguir entre KiB de memoria caché).
- Mapeamos los últimos 16 bits: rápido y sencillo,
  pero ¿cuál es el problema?

Mapeo directo

Supongamos un caché L1 de 8KiB, con direcciones de 16 bits.

```
1       double A[3][8192];
2       for (i=0; i<512; i++)
3       a[2][i] = ( a[0][i]+a[1][i] )/2.;
```

- Cada fila coincidirá en los primeros 16 bits de su dirección de memoria.
- El mapeo directo las llevará a la misma dirección en caché.
- ¿Qué ocurre en el ciclo?

Cache mapping

Si un dato se guarda en el caché, ¿dónde se ubica?

Dirección memoria principal $\longrightarrow$ Dirección memoria caché

- Caché asociativo: los datos pueden ir donde sea (en la memoria caché)
- A esto se le llama **completamente asociativo**.
- Lamentablemente, es costoso de construir.

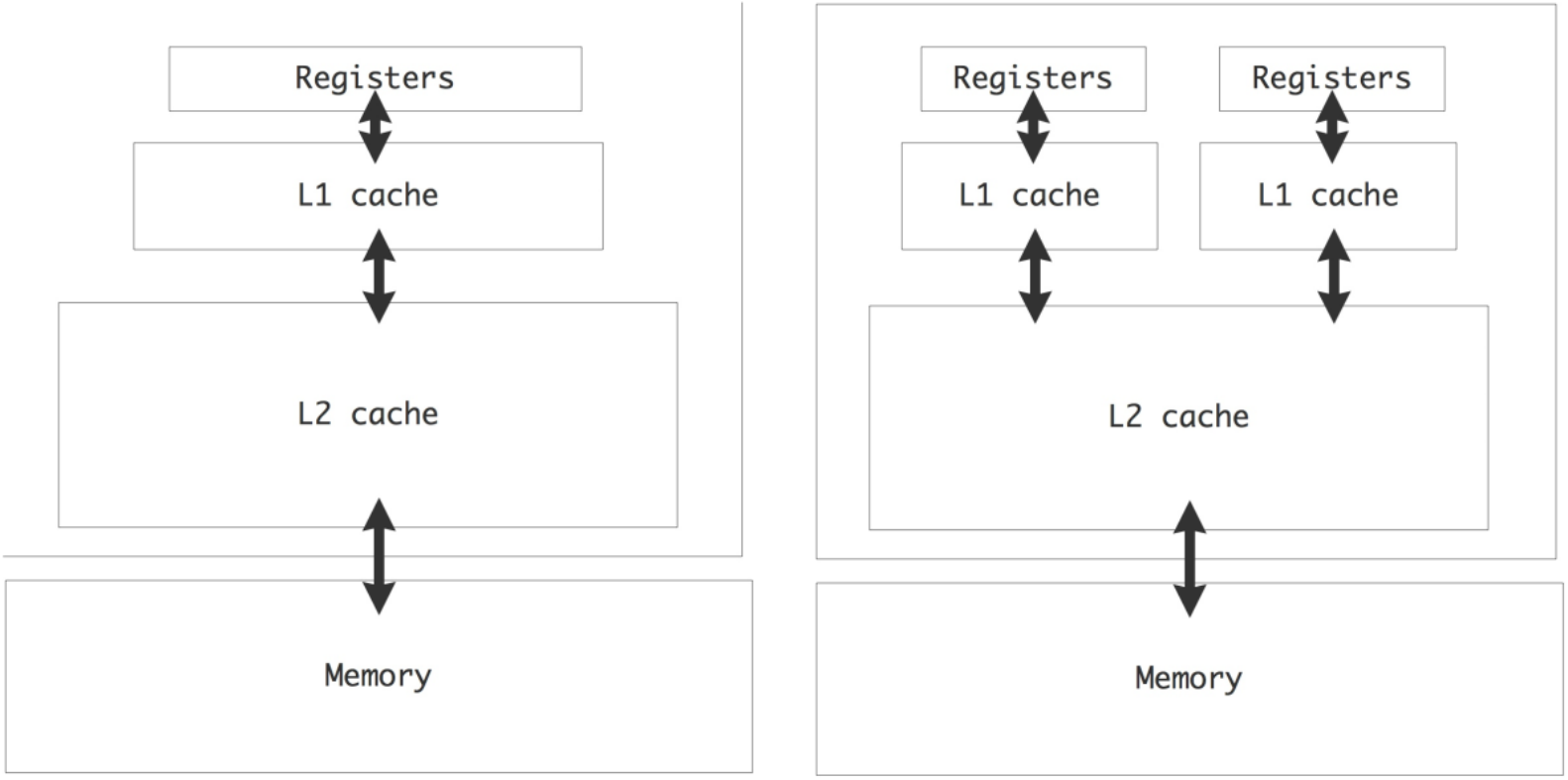
Cache mapping

Si un dato se guarda en el caché, ¿dónde se ubica?

Dirección memoria principal $\longrightarrow$ Dirección memoria caché

- Caché asociativo: los datos pueden ir a más de un lugar (en la memoria caché)
- A esto se le llama **K-asociativo**.
- $K \geq 2$.
- En Linux: `getconf -a | grep CACHE` y `lscpu | grep cache`

Caché y multicore



Coherencia

- Existe el potencial de que ocurra un conflicto si más de un procesador tiene una copia de un mismo dato.
- **Coherencia caché**.
  Asegurarse que todos los datos en caché sean copias precisas de la memoria principal.
- Si un procesador modifica, el otro procesador debe actualizar su copia.
- Pueden existir niveles de caché **privados** para un procesador.

Estados respecto a un dato

Protocolo de coherencia

- **Scratch**. La línea de caché no contiene copias del dato.
- **Valid**. La línea de caché es una copia correcta de memoria principal.
- **Reserved**. La línea de caché es la única copia del dato.
- **Dirty**. La línea de caché ha sido modificada, pero aún no re-escrita en la memoria.
- **Invalid**. El dato está en otros procesadores, y se ha realizado una modificación en alguno.

Estados respecto a un dato

Modelo MSI

Un modelo más simple es MSI:

- **Modified**. Ha sido modificado y necesita ser escrito.

  Shared. Presente en al menos un caché, no modificado.

- **Invalid**. No presente, o bien ha sido modificado en otro caché.