Memoria y caché

Esteban Mocskos[1]
basado en trabajo de Diego Garverbetsky[1], Marcelo Risk[1], Alejandro Furfaro[1],
Diego Fernández Slezak[1], Juan Pablo Galeotti[1],
Fernando Schapachnik[1]

[1]Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina

Organización del Computador I,
1er Cuatrimestre de 2018

(2) El problema que nos ocupa hoy

- Tiempos de acceso en ciclos:
  - Registro: 0-1 ciclos.
  - Memoria: 50-200 ciclos.
  - Disco: decenas de millones de ciclos.
- Dicho de otra forma:
  - Registro (suponiendo 3 GHz): aprox. 1/3 de ns.
  - Memoria (tradicional): 20-70 ns.
  - Disco: millones de ns.
- El "sueño del pibe":
  - Discos "hechos de memoria".
  - Memoria "hecha de registros".

(3) Tecnología más de cerca

La memoria es, en realidad:

- Volátil: RAM
  - SRAM ("static"): 2-3 ns.
  - DRAM ("dynamic"): 20-70 ns, pero más densa y requiere menos energía y calor.
- No volátil, ROM: Read Only Memory. ¿Nunca se escribió?
  - En la fábrica, una única vez: ROM.
  - En "casa", una única vez, Programable ROM.
  - Con equipo especial (luz ultravioleta), varias veces, Erasable PROM (90 ns).
  - De manera electrónica, muchas veces, Electrical EPROM (300 ns).
  - Una EEPROM que permite acceso de a bloques (varios bytes), se llama memoria Flash (90 ns).

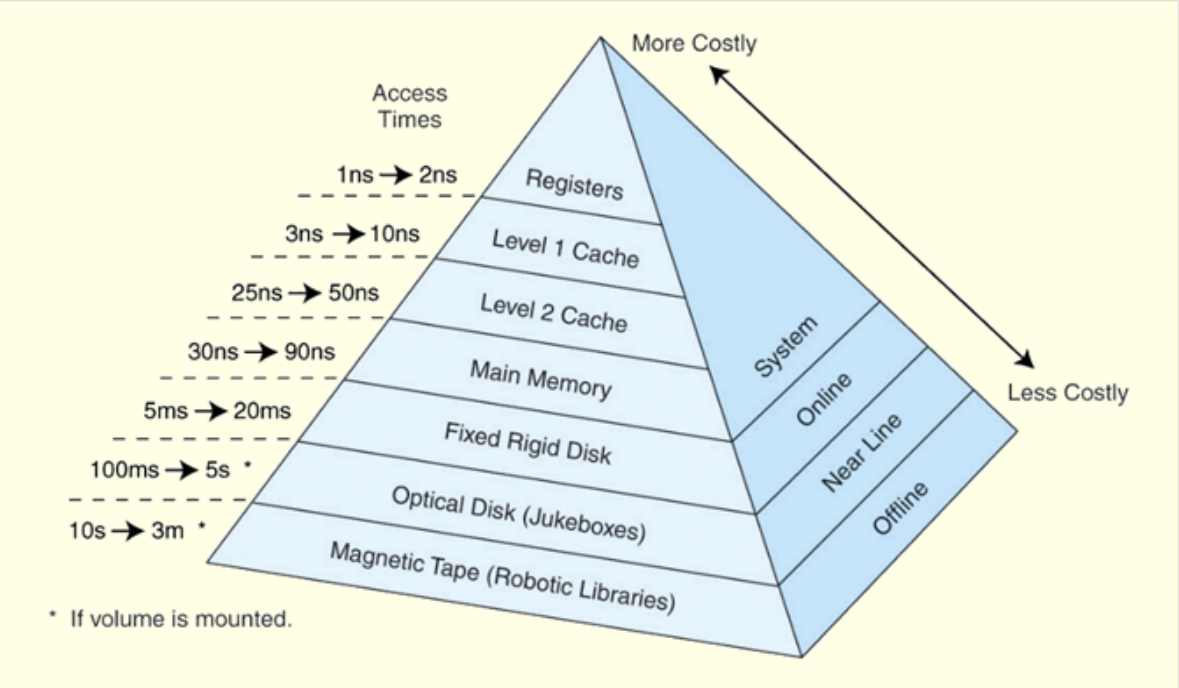
Notar que se cumplió uno de los sueños (excepto por el tamaño, pero en eso estamos).

(4) Volvamos a la RAM

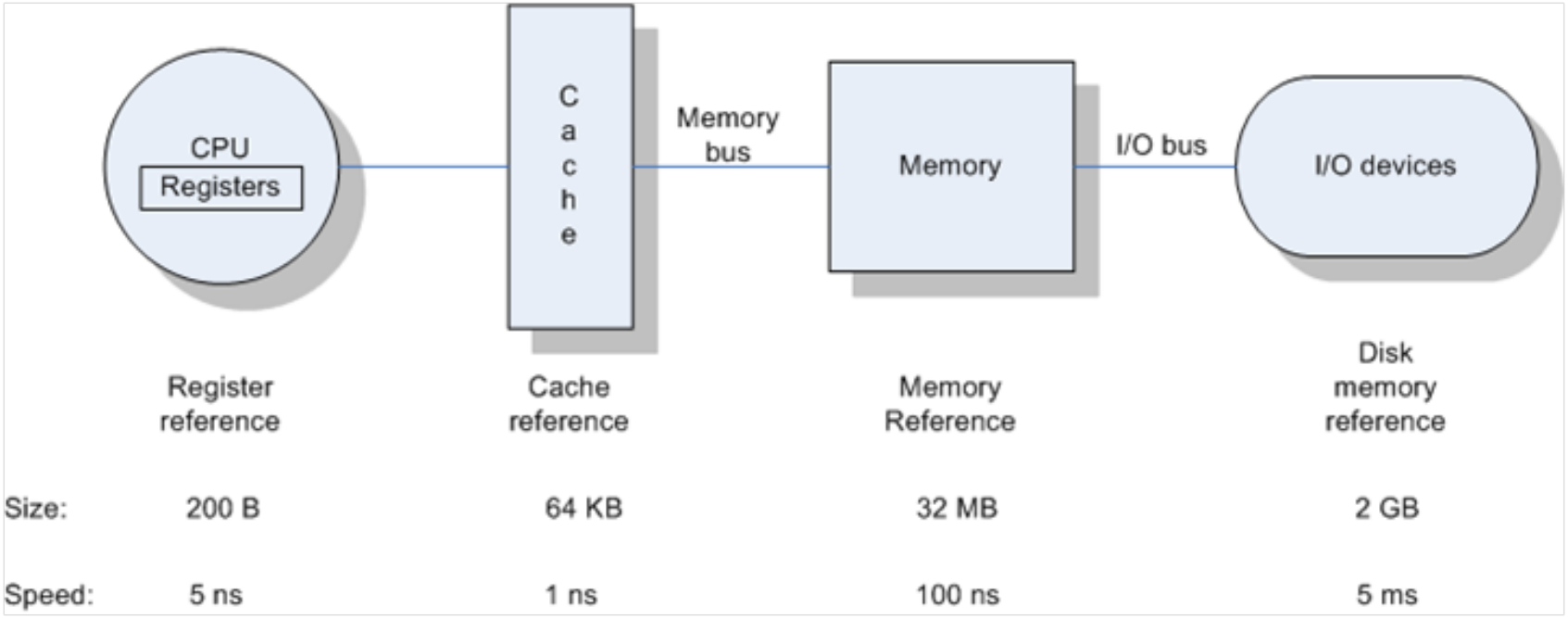
- RAM dinámica (DRAM)
  - Consumo mínimo.
  - Capacidad de almacenamiento comparativamente alta.
  - Costo por bit bajo.
  - Tiempo de acceso alto (lento), debido al circuito de regeneración de carga (20-70 ns).
  - Si construimos el banco de memoria utilizando RAM dinámica, no aprovechamos la velocidad del procesador.
- RAM estática (SRAM)
  - Alto consumo relativo.
  - Capacidad de almacenamiento comparativamente baja.
  - Costo por bit alto.
  - Tiempo de acceso bajo (2-3 ns).
  - Si construimos el banco de memoria utilizando RAM estática, el costo y el consumo de la computadora son altos.

(5) ¿Y si usamos un poco de SRAM?

- ¿Existirá alguna forma de tener un poco de cada una?
- Digamos, dejar los datos de acceso frecuente en SRAM y usar el resto en DRAM...
- A esa memoria SRAM se llama caché.
- Y se configura de esa manera la idea de jerarquía de memoria.



(6) Caché y memoria principal



(7) Memoria caché

- Es una memoria pequeña y rápida ubicada cerca de la CPU, en ella se almacena el código y los datos direccionados frecuentemente.
- Caché: del francés *cacher*, que significa guardar o esconder.
- El tamaño del banco de memoria cache debe ser:
  - Suficientemente grande para que el procesador resuelva la mayor cantidad posible de búsquedas de código y datos en esta memoria asegurando una alta performance
  - Suficientemente pequeña para no afectar el consumo ni el costo del sistema.

(8) ¿Cómo funciona?

1. La CPU solicita una posición de memoria a la memoria caché.
2. El controlador de caché verifica si se encuentra en la memoria caché.
3. Si lo está (hit), provee el dato al CPU sin acceder a la memoria principal.
4. Si no está (miss), el controlador de caché la busca en la memoria principal, y la almacena en la caché para futuras lecturas.

- En la caché cada bloque se guarda como: tag + data + valid (1 bit).

(9) Hit/Miss

- Se dice que se logra un *hit* cuando se accede a un ítem (ya sea dato o código) y éste se encuentra en la memoria caché.
- En caso contrario, se dice que el resultado del acceso es un *miss*.
- Se busca un *hit rate* lo más alto posible:

$$\text{hit rate} = \frac{\#\ \text{hits}}{\#\ \text{accesos a memoria}}$$

¿Funciona?

- Surgen varias preguntas:
  - ¿Qué hacer cuando se llena? Más adelante.
  - ¿Se volverá a pedir lo que se puso en el caché?
- Los accesos a memoria suelen tener las propiedades de *localidad temporal* y *localidad espacial*.
  - **Localidad temporal**: volverá a ser referenciado pronto (pensar en ciclos y variables).
  - **Localidad espacial**: datos cercanos al actual serán inmediatamente referenciados (pensar en ejecución secuencial).

(11) Variables de diseño de memoria caché

- ¿Cuántos niveles y de qué tamaño cada uno?
- ¿Qué hacer cuando se llena? $\rightarrow$ política de sustitución.
- ¿Dónde se almacenan los elementos? $\rightarrow$ función de correspondencia.
- ¿Cada cuánto se escriben en memoria los cambios? $\rightarrow$ política de escritura.
- Cuando se lee, ¿Cuántos bytes se traen? $\rightarrow$ tamaño de línea.

(12) Caché de mapeo directo

- Tanto la memoria como la caché se dividen en bloques de tamaño B, llamados *líneas*.
- El tamaño de la caché da la cantidad de líneas, $N = \text{tamaño}/B$.
- El bloque de mem X va al bloque de caché $X \pmod N$.
- Se usa el campo *tag* para saber de qué bloque se trata.
- Dada una dirección, sus $\log_2 N$ bits más significativos determinan el tag.
- **Ventaja**: Es fácil (barato de implementar).
- La localidad espacial está dada por el bloque.
- El reemplazo podría ser ineficiente si dos bloques de memoria muy usados compiten por el mismo bloque de caché.

(13) Caché (completamente) asociativa

- Permite que cada bloque de memoria principal pueda cargarse en cualquier línea de la caché.
- La etiqueta (tag) identifica unívocamente un bloque de la memoria principal.
- Para determinar si un bloque está en la caché se debe examinar todas los tags almacenados en la caché.
- PERO, esta búsqueda se hace en paralelo por hardware.
- Es decir, es más flexible pero más cara.
- Cuando se llena, ¿Qué líneas desalojamos?

(14) Políticas de reemplazo de líneas de caché

- FIFO: First In, First Out.
- Random
- LRU: Least Recently Used
  - Desalojamos lo que tiene más tiempo sin usarse.
  - Se almacena timestamp del instante en el que se agregó a la caché. Se desaloja la línea con `timestamp` menor.
- LFU: Least Frequently Used
  - Desalojamos lo que ha sido menos referenciado.
  - Un contador se incrementa por cada lectura. Se desaloja la línea con menor contador5.

(15) Caché asociativa por conjuntos

- Trabajo como en mapeo directo, pero el tag no me lleva a un único bloque sino a M líneas distintas.
- M es 2, 3, un número pequeño.
- Trata de combinar la simplicidad del mapeo directo con un poco más de flexibilidad.
- Lean los detalles del libro.

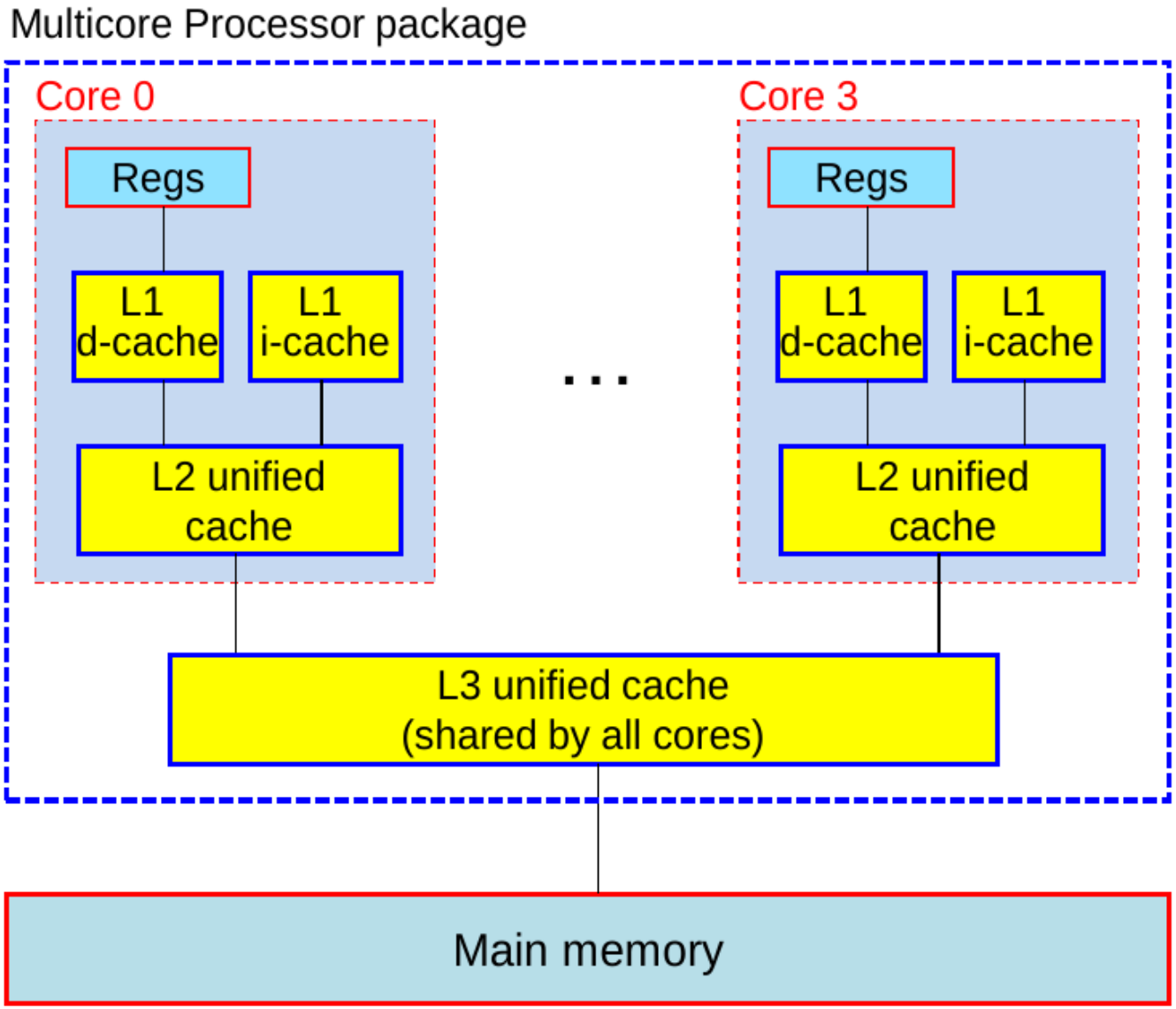
(16) Políticas de escritura

- Si la línea afectada NO se encuentra cargada en caché
  - Write-no-allocate: no se carga la línea afectada.
  - Write-allocate: se carga a caché la línea afectada.
- Si la línea afectada se encuentra cargada en caché
  - Write-through: en ese momento se escribe tanto la caché como la memoria.
  - Write-back: sólo se actualiza la caché; se escribe a memoria cuando se desaloja la línea.

(17) Caché en x86

- 80386: sin caché en el chip.
- 80486: 8KB usando líneas de 16 bytes organizados como un caché asociativo por conjuntos de 4 vías.
- Pentium (todas las versiones): dos cachés L1 en el chip (datos e instrucciones).
- Pentium III: caché L3 fuera del chip.
- Pentium 4
  - L1 caches: 8KB, líneas de 64 bytes, asociativo de 4 vías.
  - L2 cache: 256KB, líneas de 128 bytes, asociativo de 8 vías, alimenta a los cachés L1.
  - L3 cache on chip.

(18) Caché en multicores



(19) Coherencia de caché

- Cada núcleo tiene su propia caché privada.
- Tranquilamente, dos núcleos podrían tener copia de los mismos elementos.
- ¿Qué pasa si uno de los núcleos lo modifica?
- Coherencia de caché: el sistema asegura que todos los núcleos perciben que tienen la misma vista de la memoria.
- Mantener la coherencia de caché es caro y no escala.
- Una de las opciones es hacer *snooping*: cada controlador de caché verifica si otro núcleo escribe un elemento, si se modifica, se marca la línea como no válida.

(20) Bibliografía

- Null, capítulo 6.