Chapter 4: Memory

Gerardo Bandera Burgueño

Dept. de Arquitectura de Computadores

[Adapted from *Computer Organization and Design, 4th Edition*, Patterson & Hennessy, © 2009, MK]

UNIVERSIDAD DE MÁLAGA

ANDALUCÍA TECH
Campus de Excelencia Internacional
International Campus of Excellence

Index

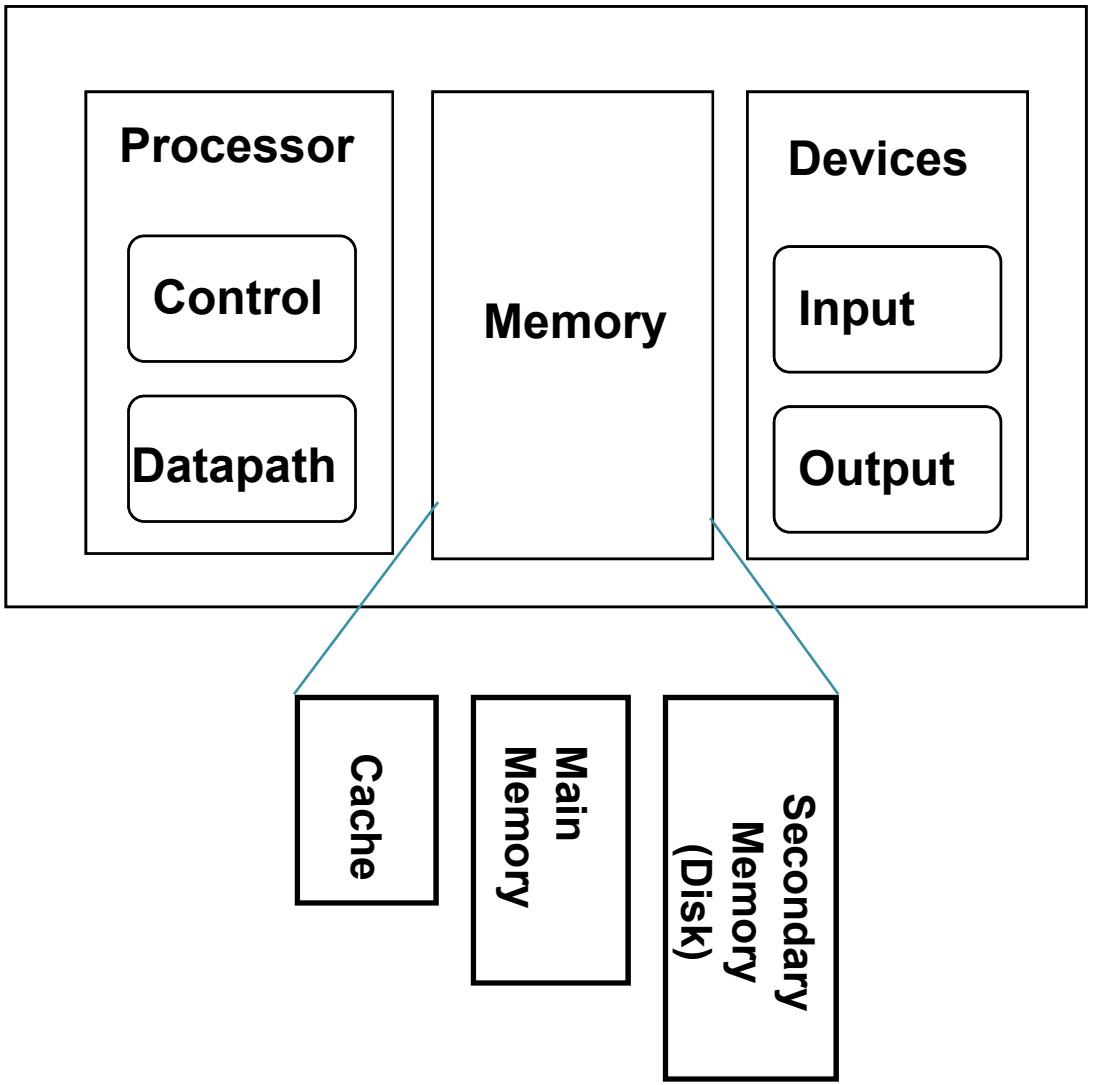
- Introduction
- Cache Memory
- Interleaved Memory
- Virtual Memory

Index

4.1.- Introduction

- Cache Memory
- Interleaved Memory
- Virtual Memory

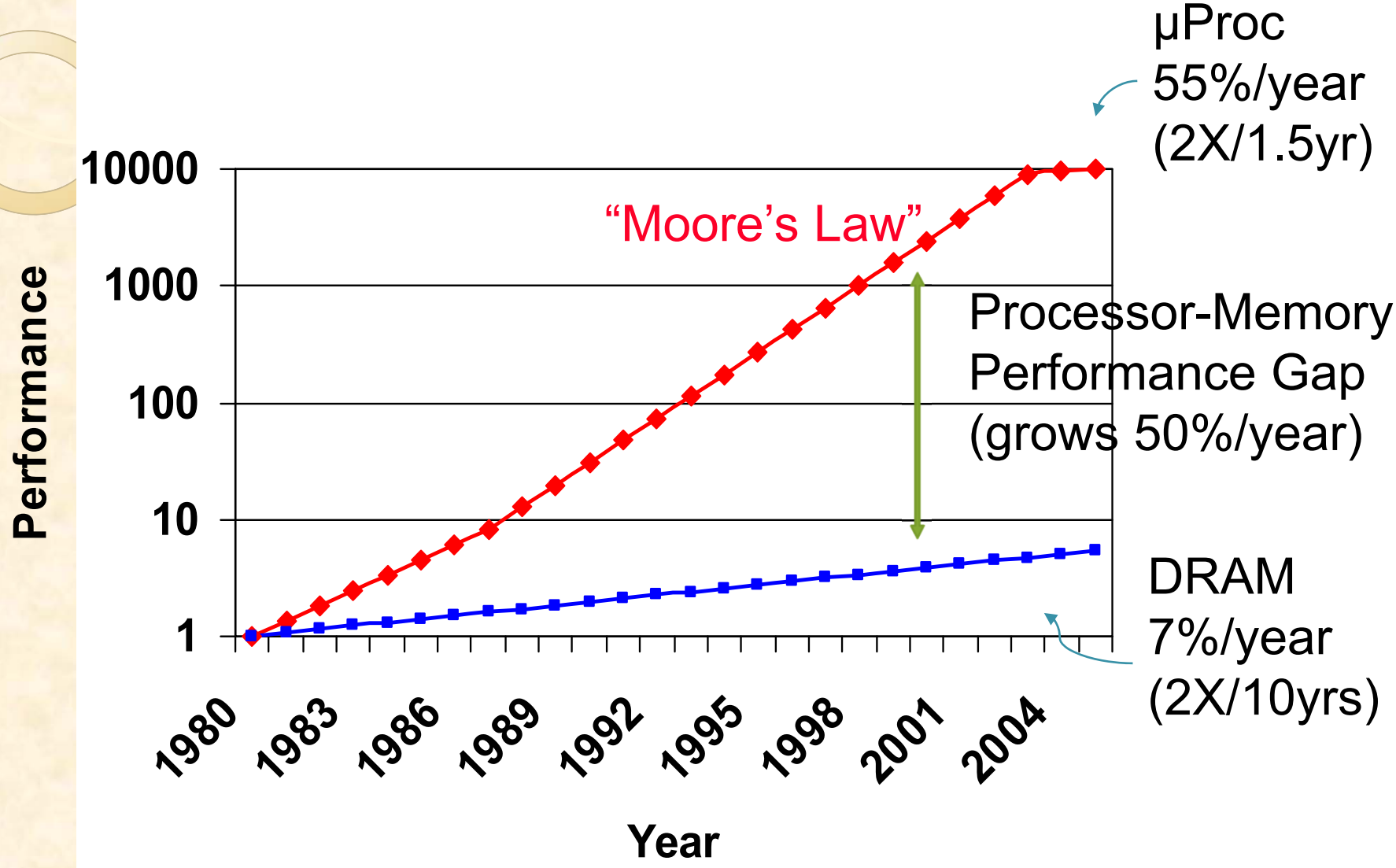
Review: Major Components of a Computer



The Memory Hierarchy Goal

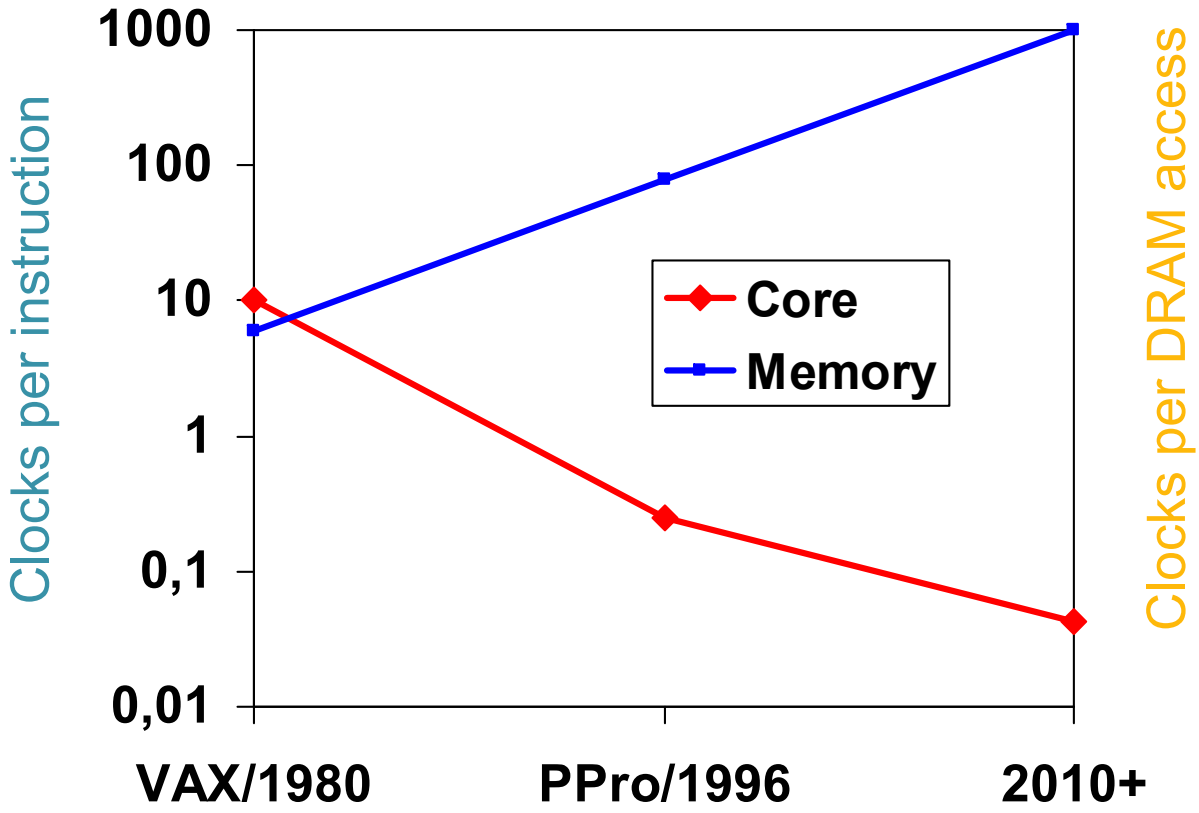
- Fact: Large memories are slow and fast memories are small
- How do we create a memory that gives the illusion of being large, cheap and fast (most of the time)?
  - With hierarchy
  - With parallelism

Processor-Memory Performance Gap



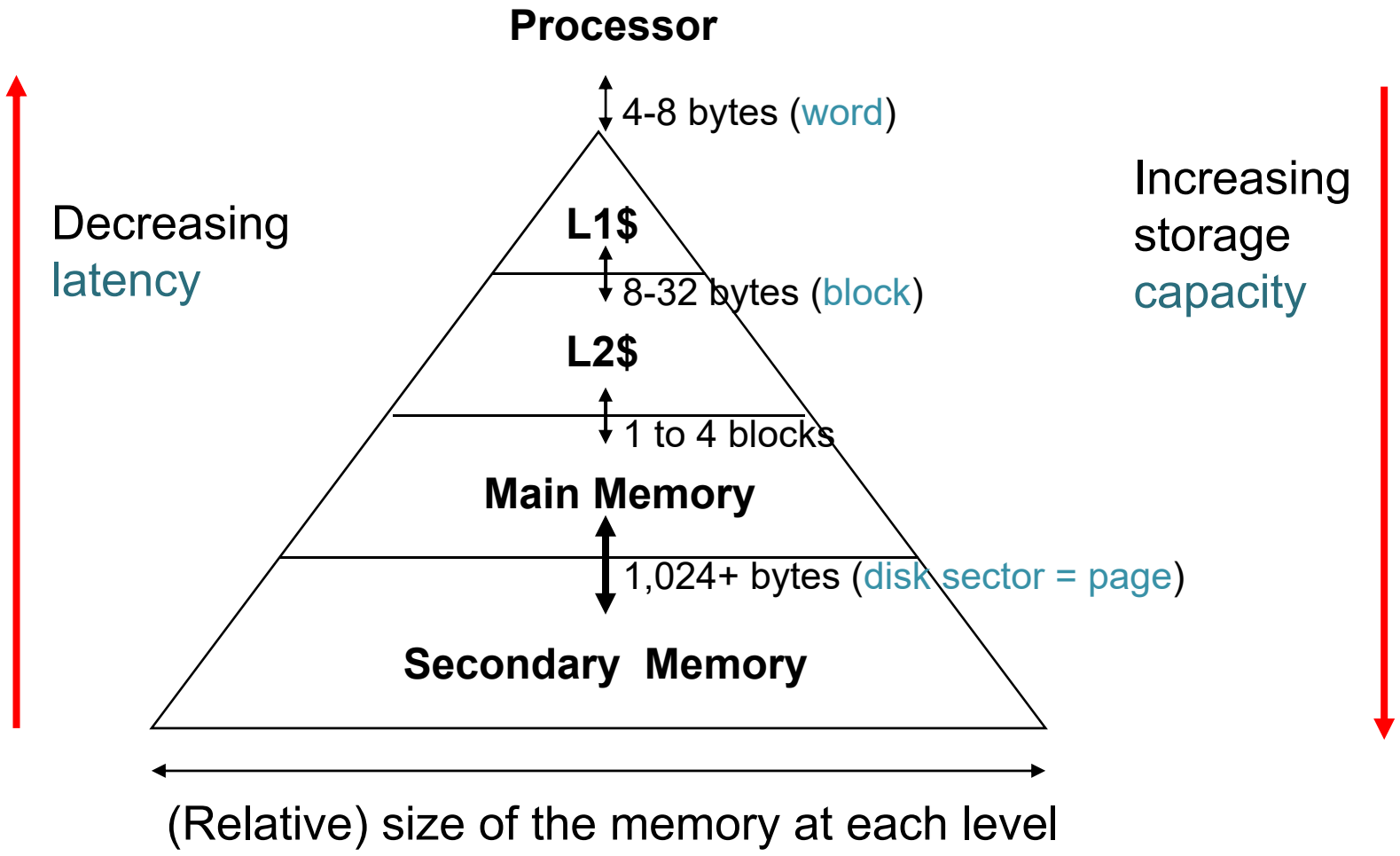
The "Memory Wall"

- Processor vs DRAM speed disparity continues to grow



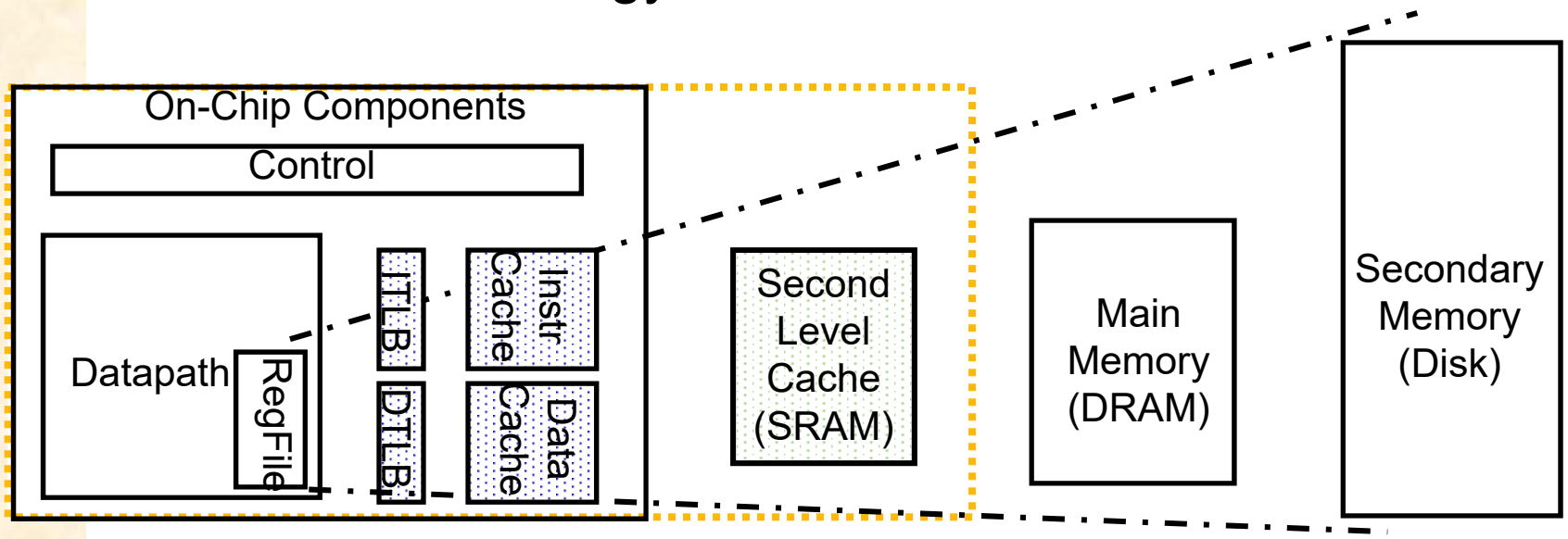
- Good memory hierarchy (cache) design is increasingly important to overall performance

Characteristics of the Memory Hierarchy



A Typical Memory Hierarchy

- Take advantage of the principle of locality to present the user with as much memory as is available in the *cheapest* technology at the speed offered by the *fastest* technology



Speed (cycles):	½'s	1's	10's	100's	10,000's
Size (bytes):	100's	10K's	M's	G's	T's
Cost:	highest				lowest

Memory Hierarchy Technologies

- Caches use *SRAM* for speed and technology compatibility
  - **Fast** (typical access times of 0.5 to 2.5 nsec)
  - Low density (6 transistor cells), higher power, expensive ($2000 to $5000 per GB in 2008)
  - Static: content will last "forever" (as long as power is left on)

- Main memory uses *DRAM* for size (density)
  - Slower (typical access times of 50 to 70 nsec)
  - High density (1 transistor cells), lower power, **cheaper** ($20 to $75 per GB in 2008)
  - Dynamic: needs to be "refreshed" regularly (~ every 8 ms)
    - consumes1% to 2% of the active cycles of the DRAM
  - Addresses divided into 2 halves (row and column)
    - *RAS* or *Row Access Strobe* triggering the row decoder
    - *CAS* or *Column Access Strobe* triggering the column selector

Why Does it Work?

- Temporal Locality (locality in time)
  - If a memory location is referenced then it will tend to be referenced again soon

  ⇒ Keep most recently accessed data items closer to the processor

- Spatial Locality (locality in space)
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

  ⇒ Move blocks consisting of contiguous words closer to the processor

How is the Hierarchy Managed?

- registers ↔ memory
  - by compiler (programmer?)
- cache ↔ main memory
  - by the cache controller hardware
- main memory ↔ disks
  - by the operating system (virtual memory)
  - virtual to physical address mapping assisted by the hardware (TLB)
  - by the programmer (files)

Index

4.1.- Introduction

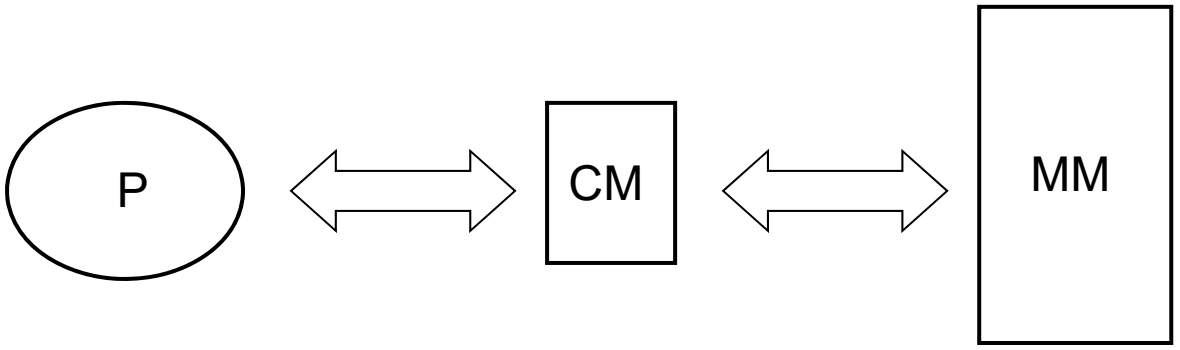
4.2.- Cache Memory

- Interleaved Memory
- Virtual Memory

Cache Basics

- **Definition:**
  - Small and fast memories near the processor.
  - Storing the most frequently used information.

- **Main Goal:**
  - Reduce memory access time.

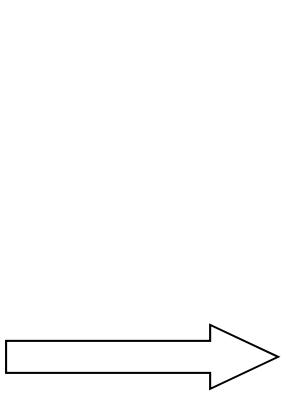


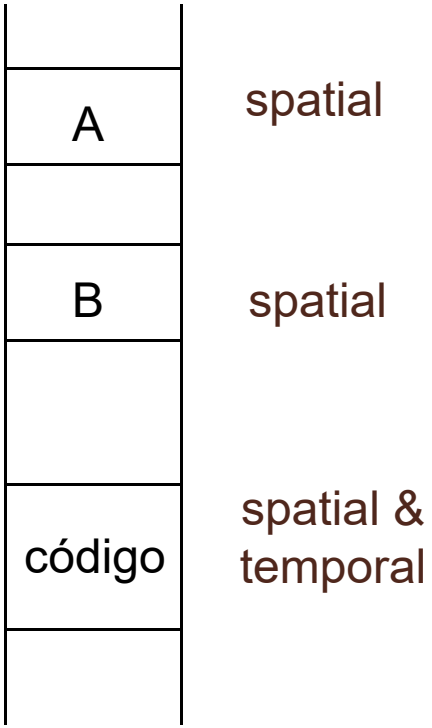
Cache Basics

- **Locality Properties:**
  - Spatial Locality: If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon.
  - Temporal Locality: If a memory location is referenced, then it will tend to be referenced again soon

  - Code example:

```
DO i= 1, 100
   A[i] = 2* B[i]
ENDDO
```

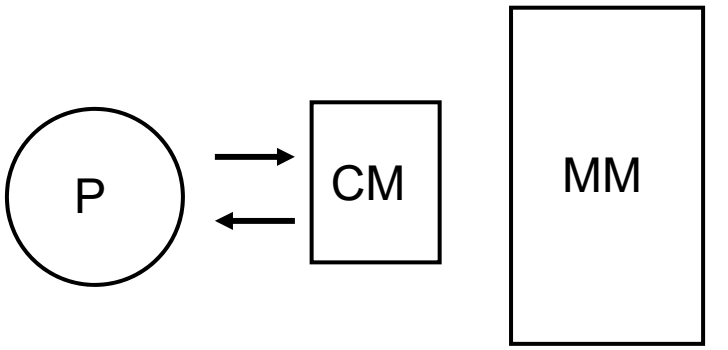




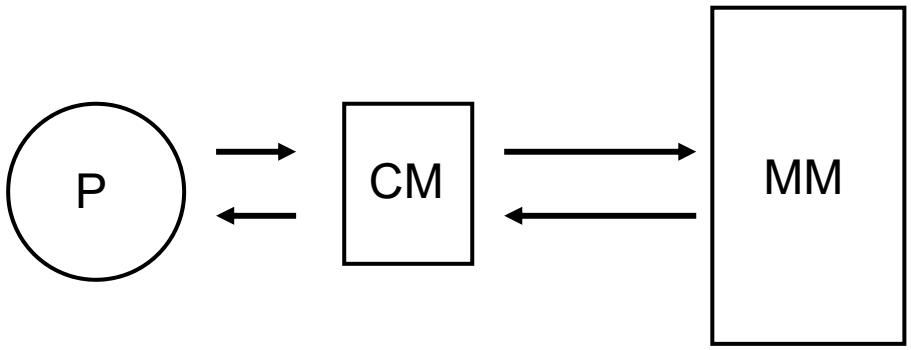
Cache Basics

- **Working:**
  - CM contains duplicated information from MM.
  - On each memory reference → a CM searching is produced:

- HIT: reference is on CM



- MISS: reference is not on CM



Additional goal: reduce the miss rate.

$$\text{Miss rate} = \frac{\#\text{ cache misses}}{\#\text{ memory references}}$$

Cache Terminology

- Block (or line): the minimum unit of information that is present (or not) in a cache
- Hit Rate: the fraction of memory accesses found in a level of the memory hierarchy
- Hit Time: Time to access that level which consists of

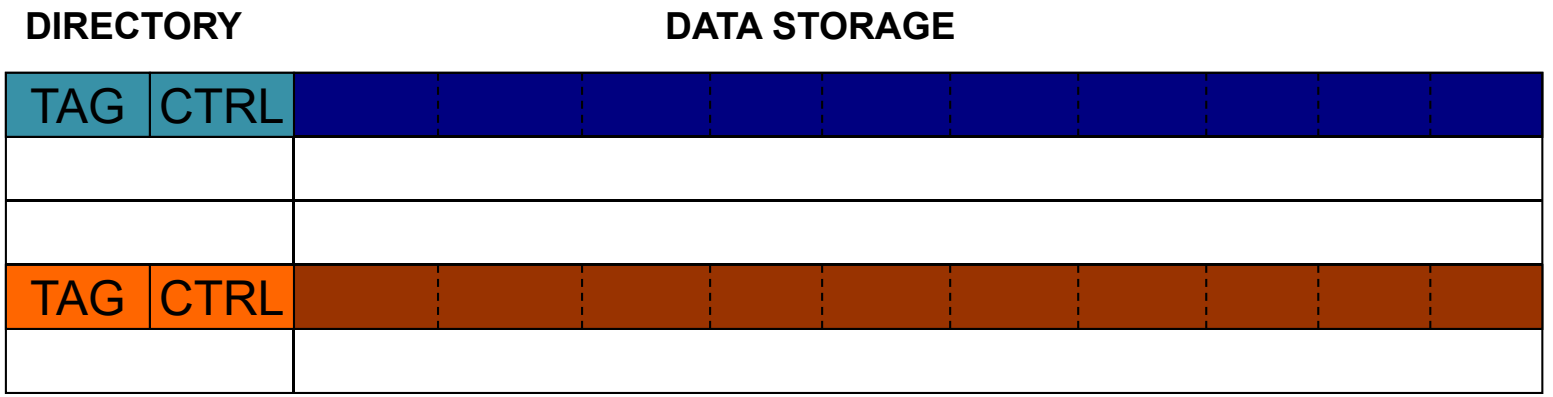
  Time to access the block + Time to determine hit/miss
- Miss Rate: the fraction of memory accesses *not* found in a level of the memory hierarchy $\Rightarrow$ 1 - (Hit Rate)
- Miss Penalty: Time to replace a block in that level with the corresponding block from a lower level which consists of

  Time to access the block in the lower level + Time to transmit that block to the level that experienced the miss + Time to insert the block in that level + Time to pass the block to the requestor

Hit Time << Miss Penalty

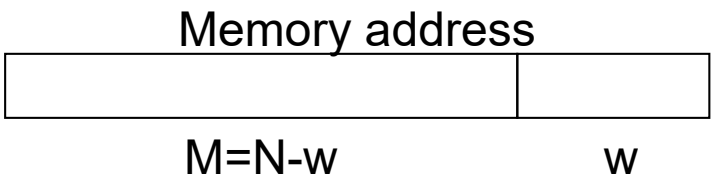
Cache Internals

- **Data Storage:** copy of main memory information.
  - BLOCKS.
- **Cache Directory:** auxiliary information
  - TAGS & Control.



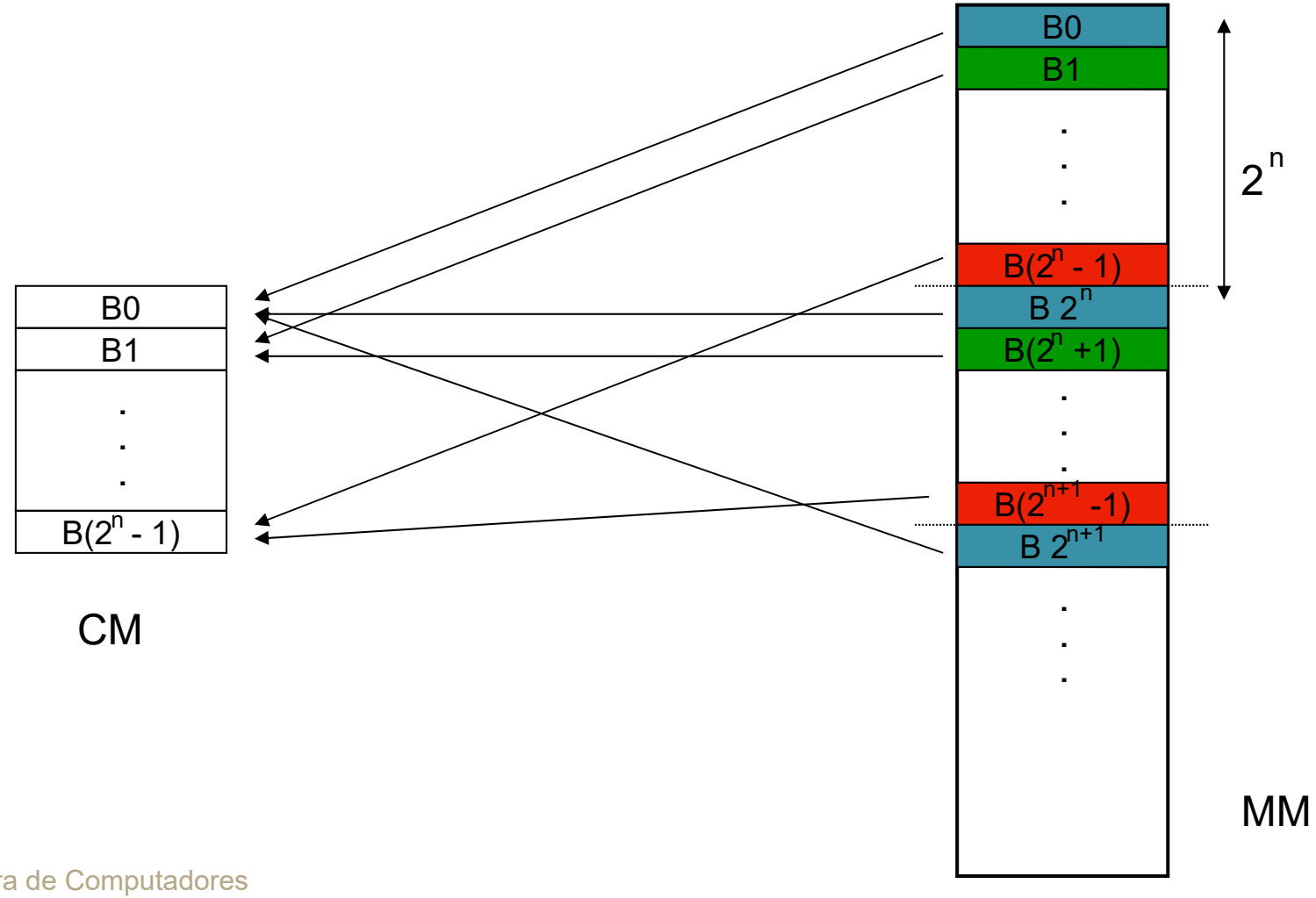
Blocks Placement

- **Where is located a block in CM?**
- **We know that….**
  - # blocks in MM >> size of CM Data Storage.
- **Alternatives:**
  - Direct Mapping
  - Fully Associative
  - Set Associative
- **We will suppose that…**
  - # main memory blocks: 2^N
  - # words per block: 2^w
  - Cache memory capacity (in blocks): 2^n



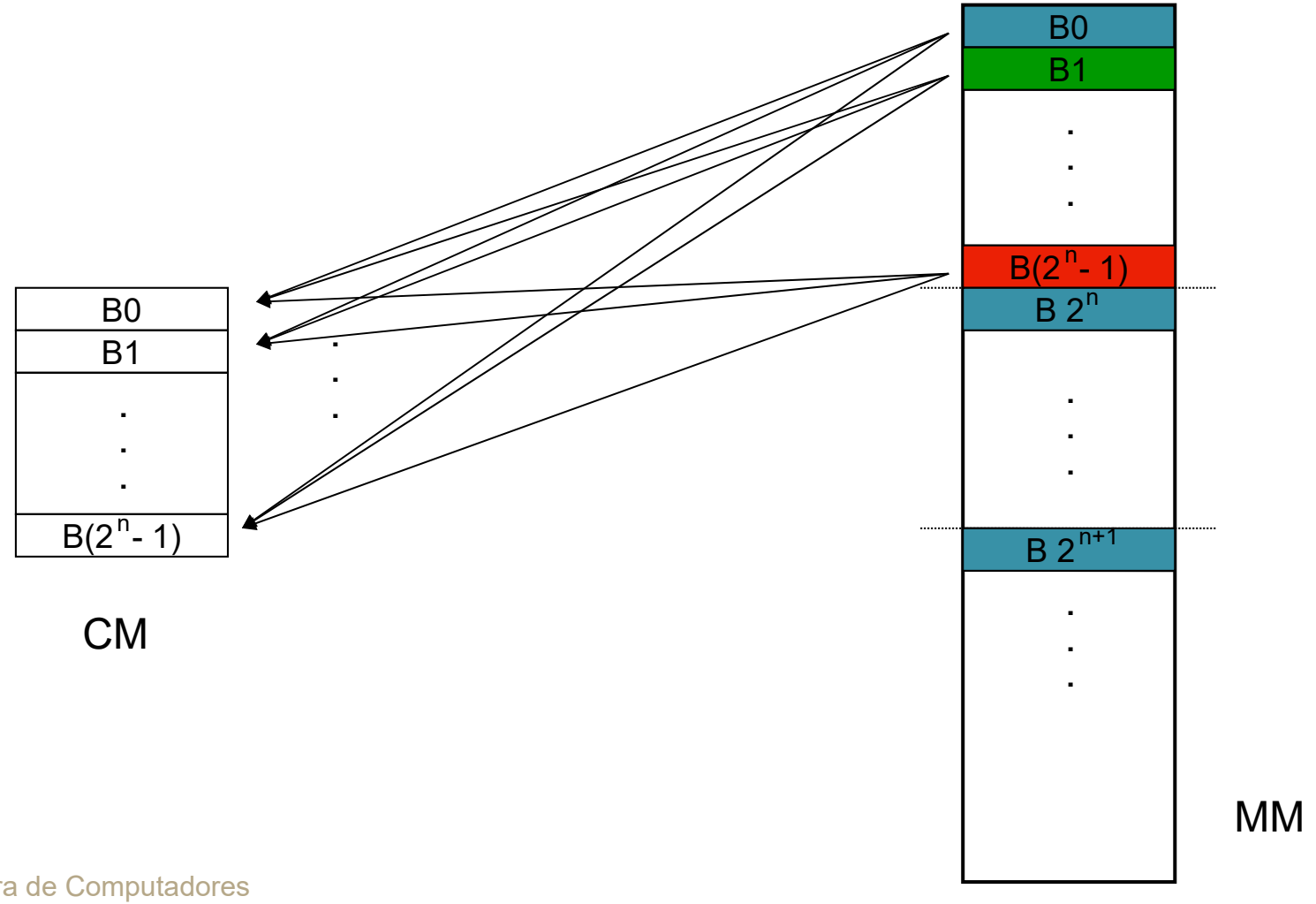
a) Direct-Mapping

- Each memory block will be only on a single CM entry.
- Block-i (MM) → Block-j (CM) (where j = i mod 2^n).



b) Fully-Associative

- Every MM block can be stored on each CM entry.

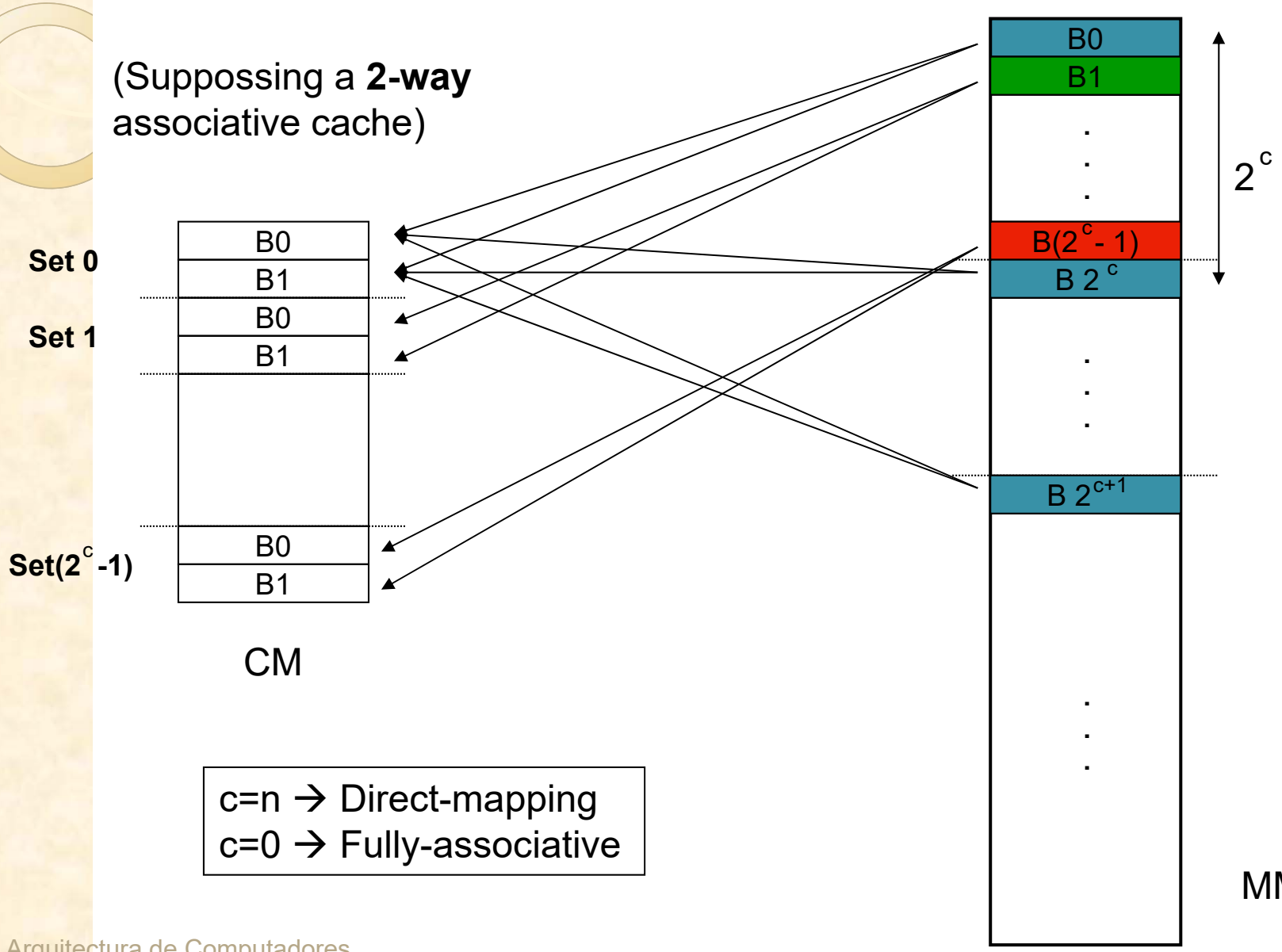


c) Set-Associative

- CM is divided in sets (group of blocks)
- Every set with the same # blocks.
  - 4 blocks per set → 4-way associate cache.
- A given block will be only on a single set….
  - … but could be in every block of this given set (not at the same time).
- As a summary…
  - Direct mapping to the set.
  - Fully-associative mapping inside the set.
  - That is…
    - Block-i (MM) → Set-k (CM) ($k = i \bmod 2^c$).

c) Set-Associative

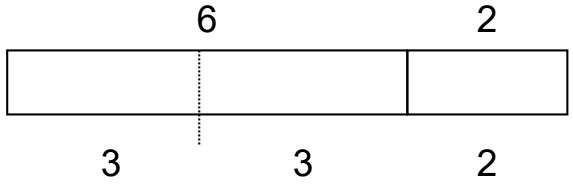
(Suppossing a **2-way** associative cache)



c=n → Direct-mapping
c=0 → Fully-associative

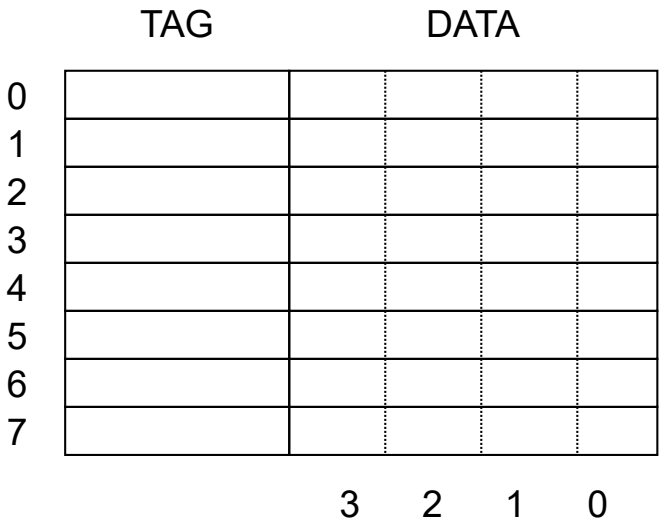
Example: Direct-Mapping

- Supposing we have:
  - 64 blocks in Main Memory → M = 6
  - 4 words per block → w = 2
  - 8 blocks in Cache → n = 3



- Source code:

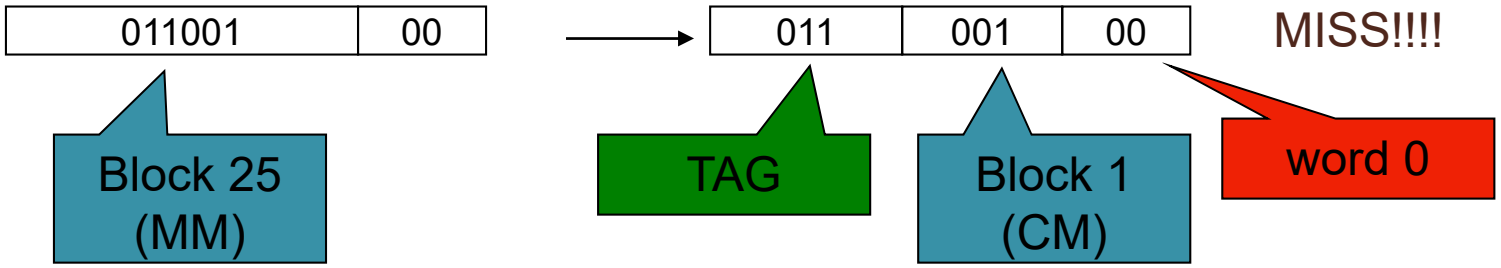
```
LOOP
        100
        17
        79
        50
END LOOP
```

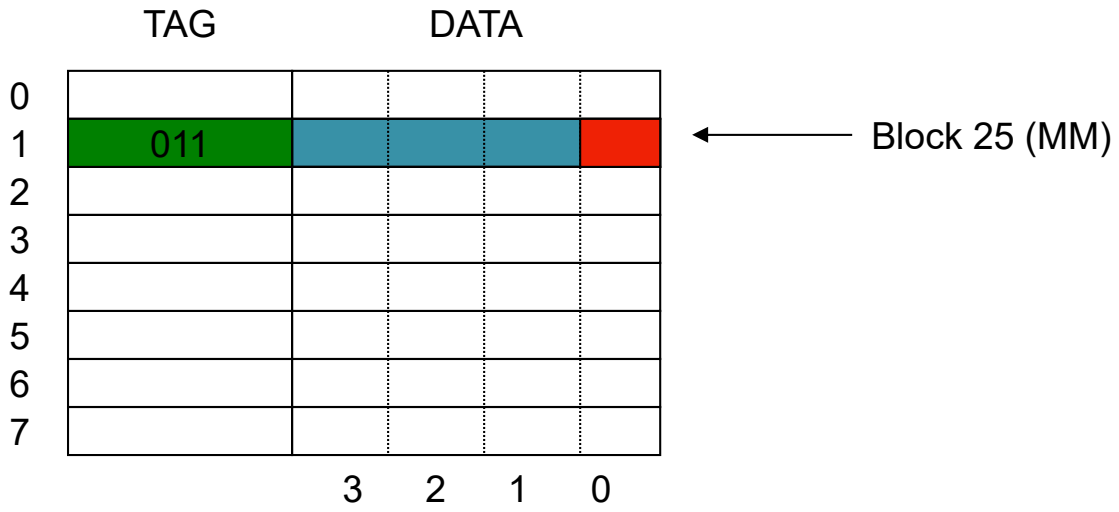


Direct-Mapping

Iteration #1:

- $100_{(10}$:

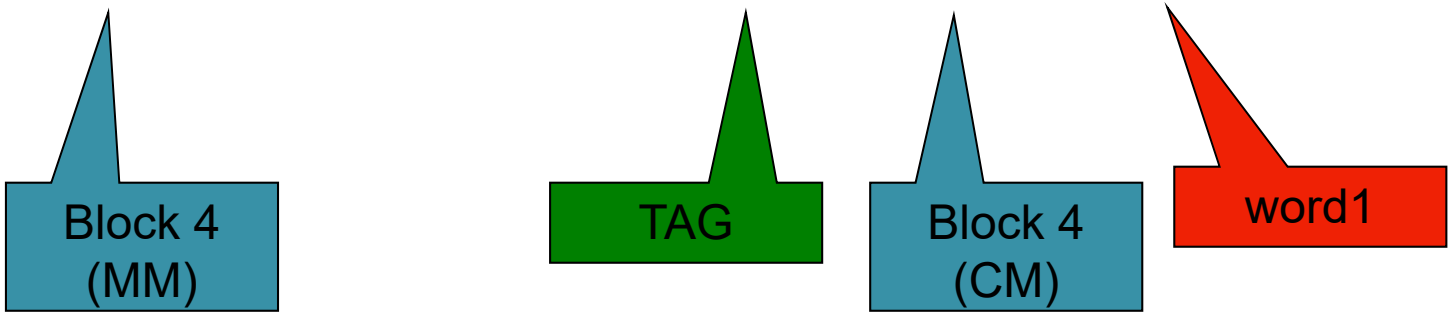


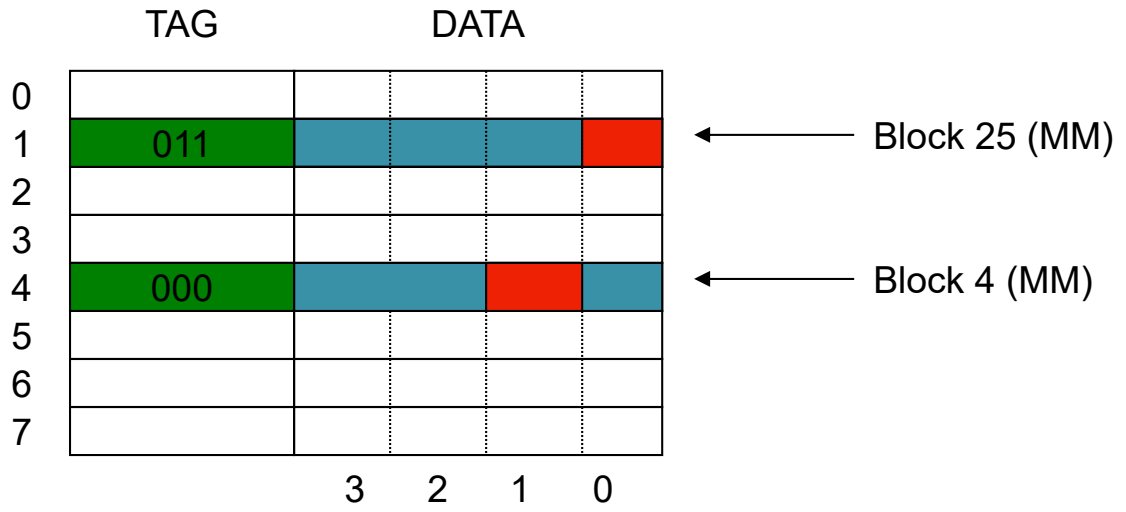


Direct-Mapping

Iteration #1:

- $100_{(10}$: 011001 | 00
- $17_{(10}$: 000100 | 01 → 000 | 100 | 01 MISS!!!!





Direct-Mapping

Iteration #1:

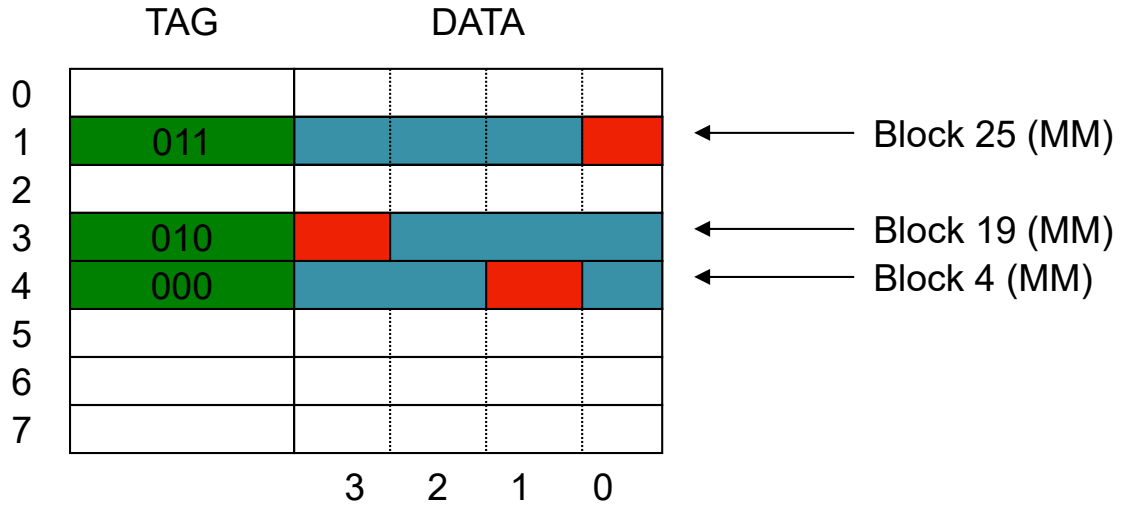
- $100_{(10}$: 011001 | 00
- $17_{(10}$: 000100 | 01
- $79_{(10}$: 010011 | 11 → 010 | 011 | 11 MISS !!!!

Block 19 (MM)

TAG

Block 3 (CM)

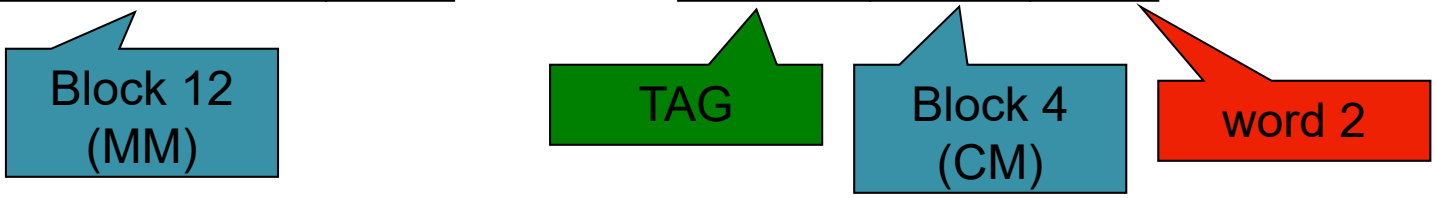
Word 3

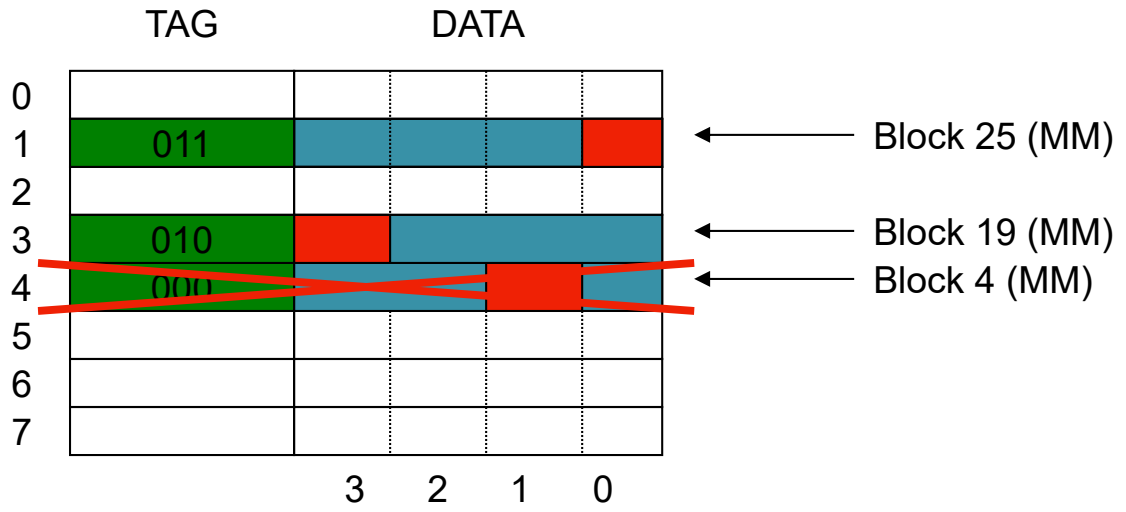


Direct-Mapping

Iteration #1:

- $100_{(10}$: 011001 | 00
- $17_{(10}$: 000100 | 01
- $79_{(10}$: 010011 | 11
- $50_{(10}$: 001100 | 10

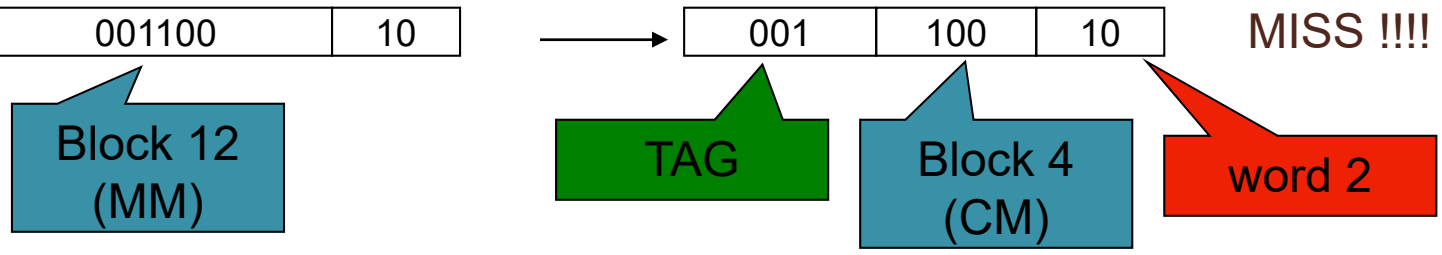


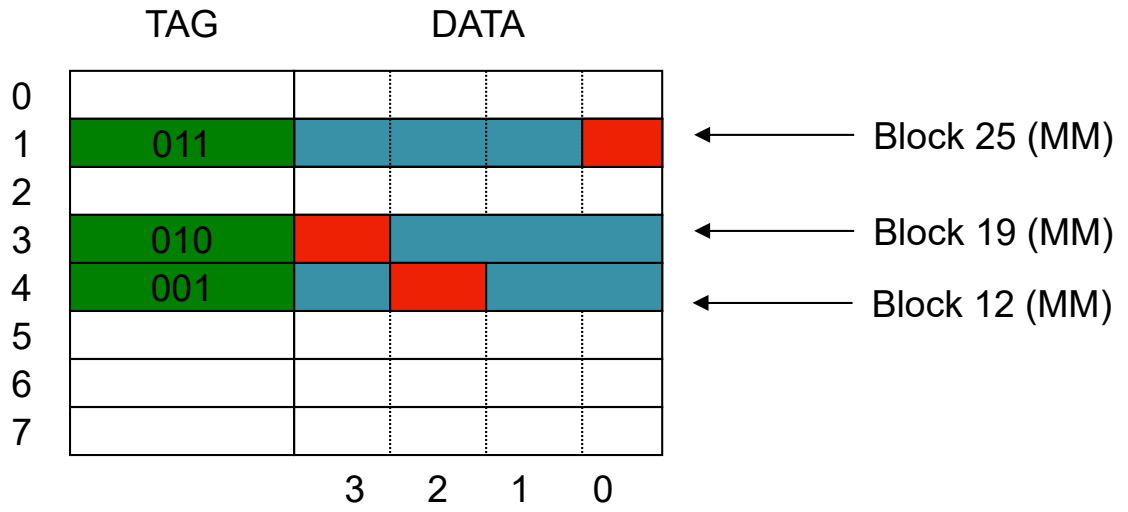


Direct-Mapping

Iteration #1:

- $100_{(10}$: 011001 | 00
- $17_{(10}$: 000100 | 01
- $79_{(10}$: 010011 | 11
- $50_{(10}$: 001100 | 10



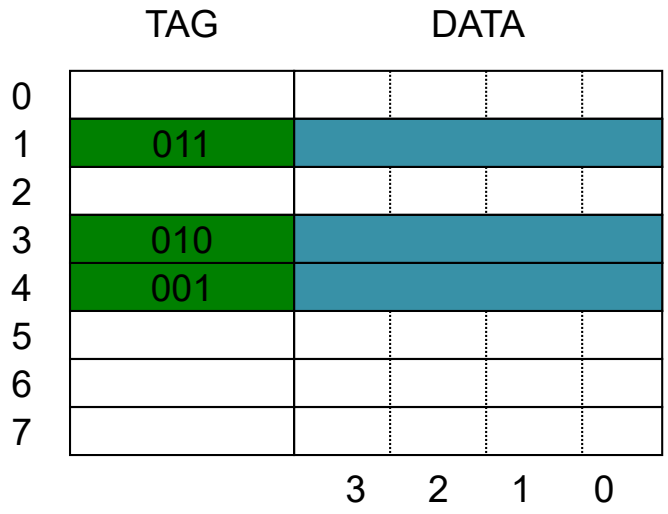


Direct-Mapping

Iteration #2 to …:

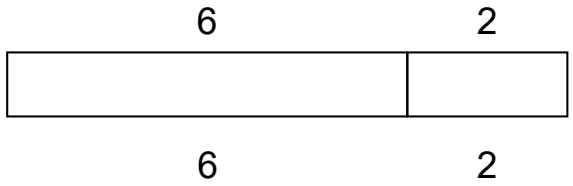
- $100_{(10}$: 011001 | 00 → 011 | 001 | 00 — HIT!!!!
- $17_{(10}$: 000100 | 01 → 000 | 100 | 01 — MISS!!!!
- $79_{(10}$: 010011 | 11 → 010 | 011 | 11 — HIT!!!!
- $50_{(10}$: 001100 | 10 → 001 | 100 | 10 — MISS!!!!

Miss Rate ≅ 0.5



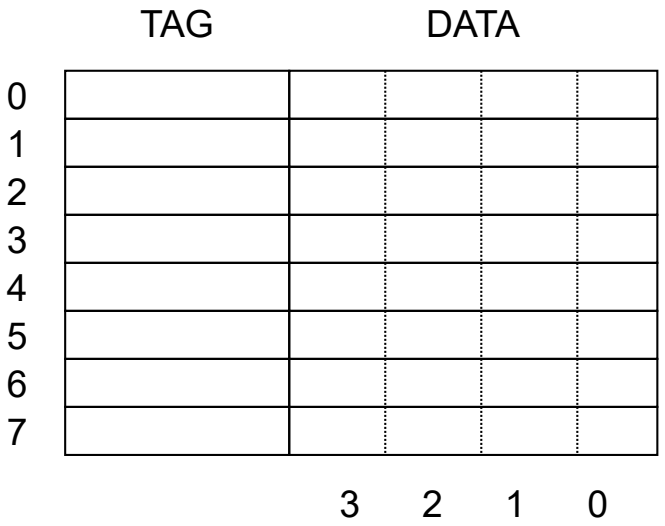
Example: Fully Associative

- Supposing we have:
  - 64 blocks in Main Memory → M = 6
  - 4 words per block → w = 2
  - 8 blocks in Cache → n = 3



- Source code:

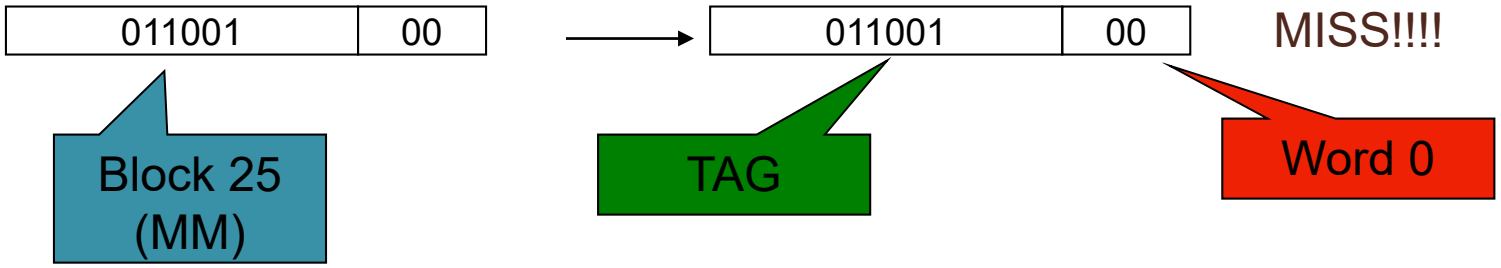
```
LOOP
        100
        17
        79
        50
END LOOP
```

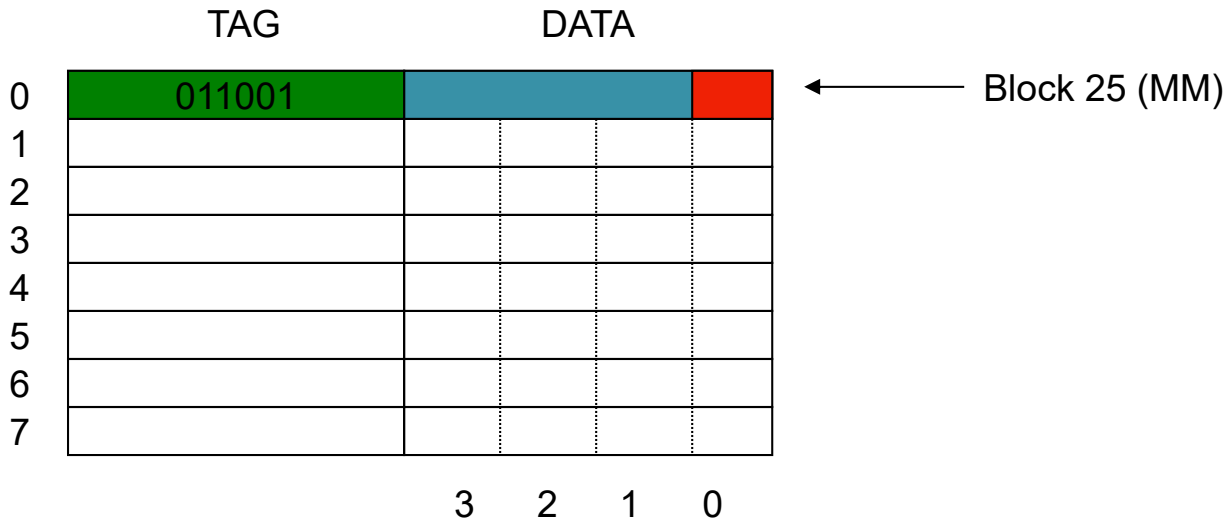


Fully-Associative

Iteration #1:

- $100_{(10}$:





Fully-Associative

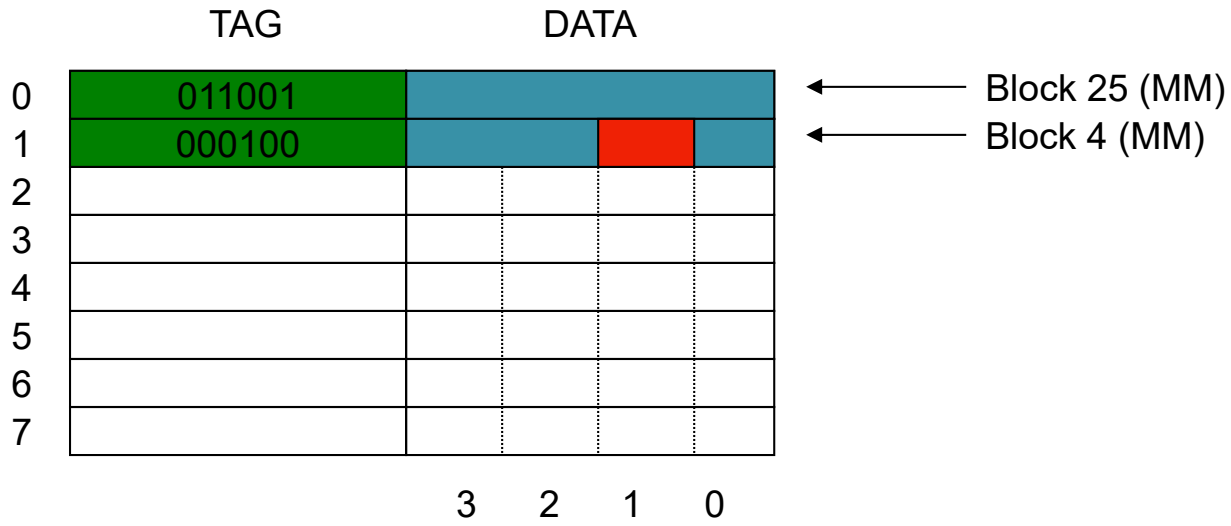
Iteration #1:

- $100_{(10}$: 011001 | 00
- $17_{(10}$: 000100 | 01 → 000100 | 01 MISS!!!!

Block 4 (MM)

TAG

word1



Fully-Associative

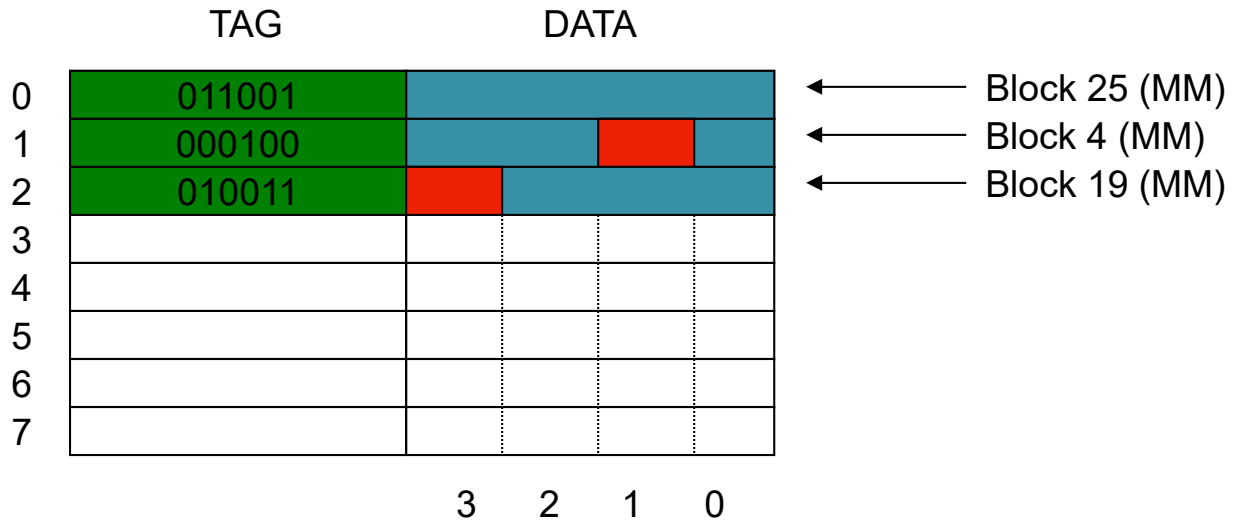
Iteration #1:

- $100_{(10}$: 011001 | 00
- $17_{(10}$: 000100 | 01
- $79_{(10}$: 010011 | 11 → 010011 | 11 MISS !!!!

Block 19 (MM)

TAG

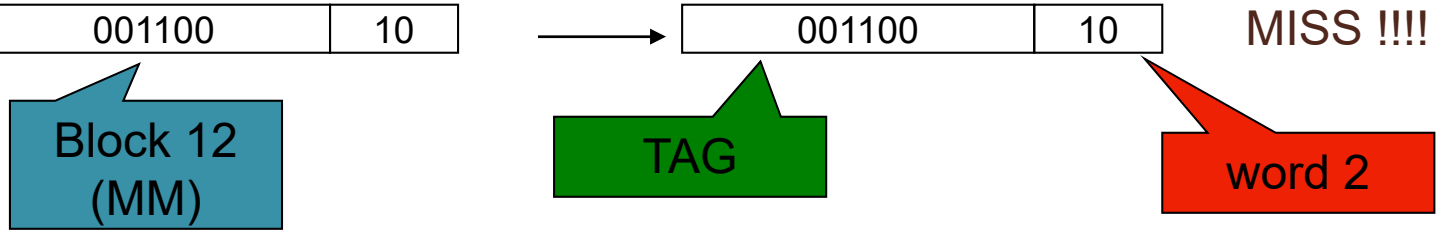
Word 3

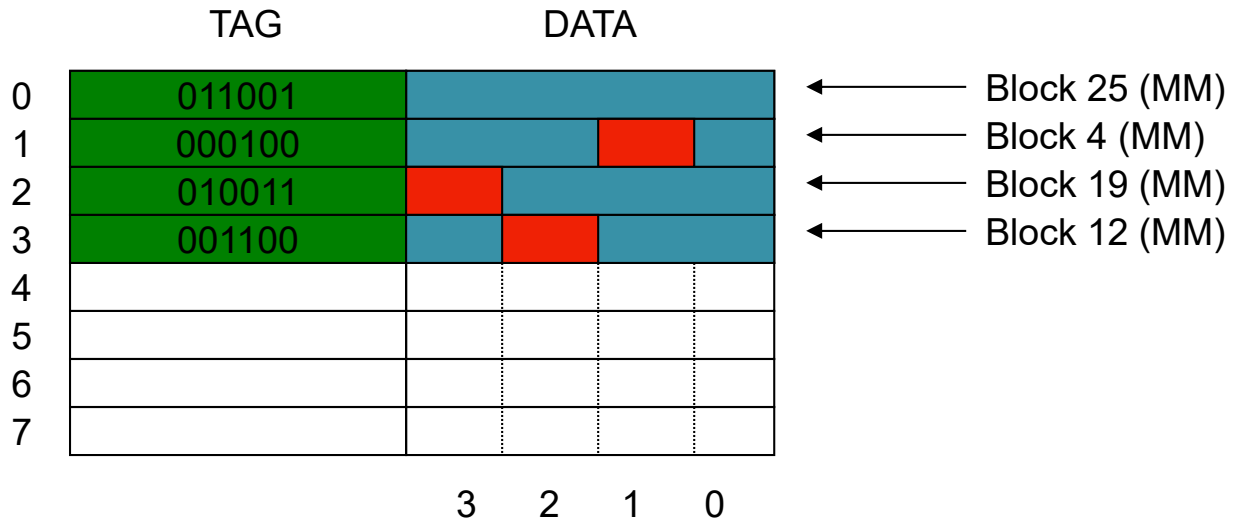


Fully-Associative

Iteration #1:

- $100_{(10}$: 011001 | 00
- $17_{(10}$: 000100 | 01
- $79_{(10}$: 010011 | 11
- $50_{(10}$: 001100 | 10



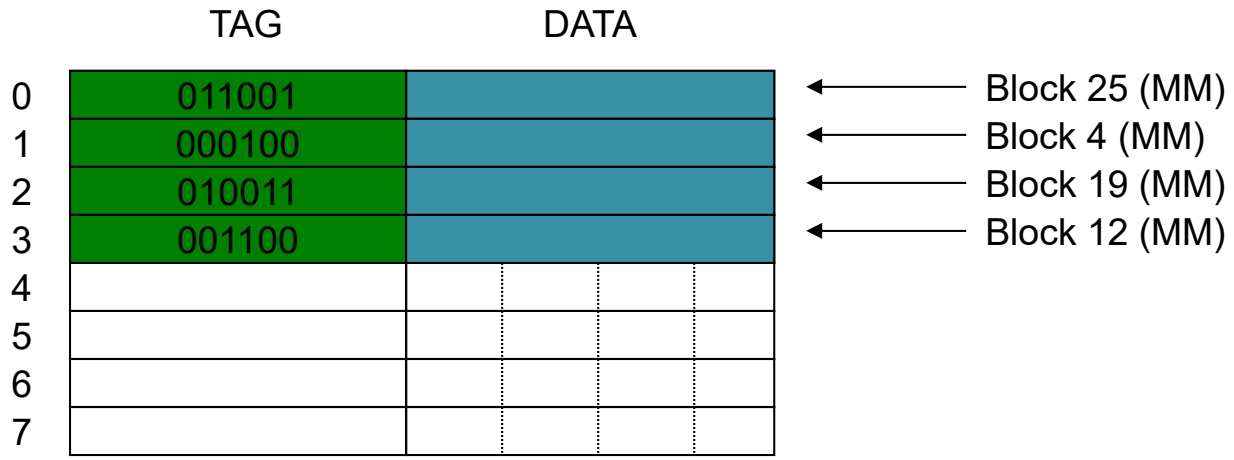


Fully-Associative

Iteration #2 to …:

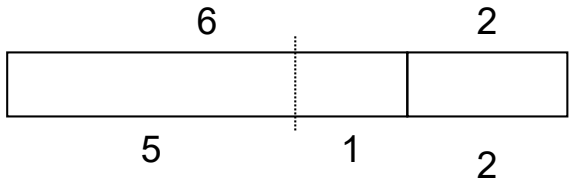
- $100_{(10}$: 011001 | 00 — HIT !!!!
- $17_{(10}$: 000100 | 01 — HIT !!!!
- $79_{(10}$: 010011 | 11 — HIT !!!!
- $50_{(10}$: 001100 | 10 — HIT !!!!

→ Miss Rate ≅ 0



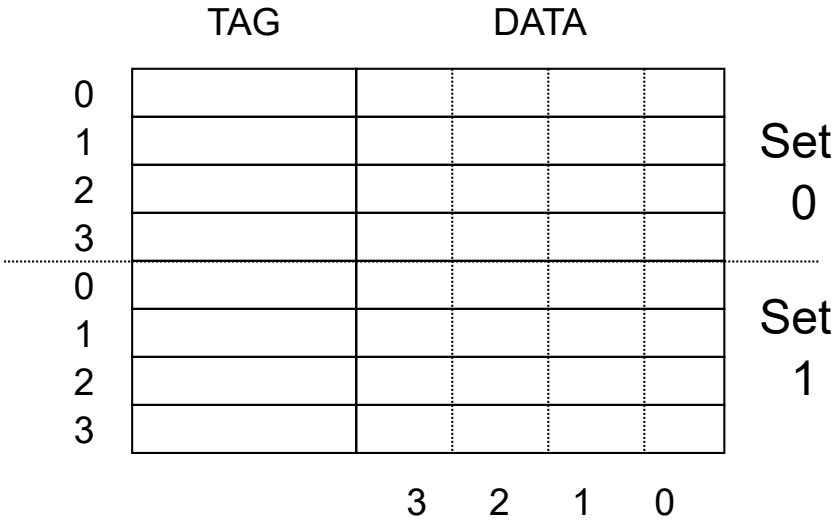
Example: Set Associative

- Supposing we have:
  - 64 blocks in Main Memory → M = 6
  - 4 words per block → w = 2
  - 8 blocks in Cache → n = 3
  - 4-way (2 sets → c = 1)



- Source code:

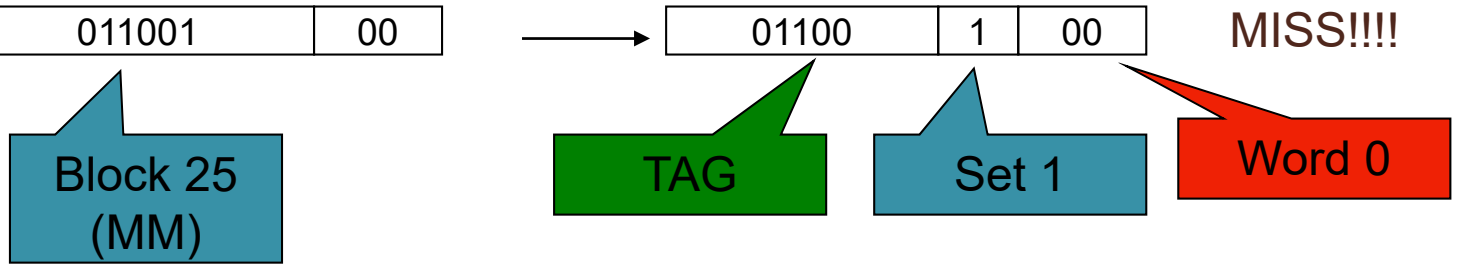
```
LOOP
        100
        17
        79
        50
END LOOP
```

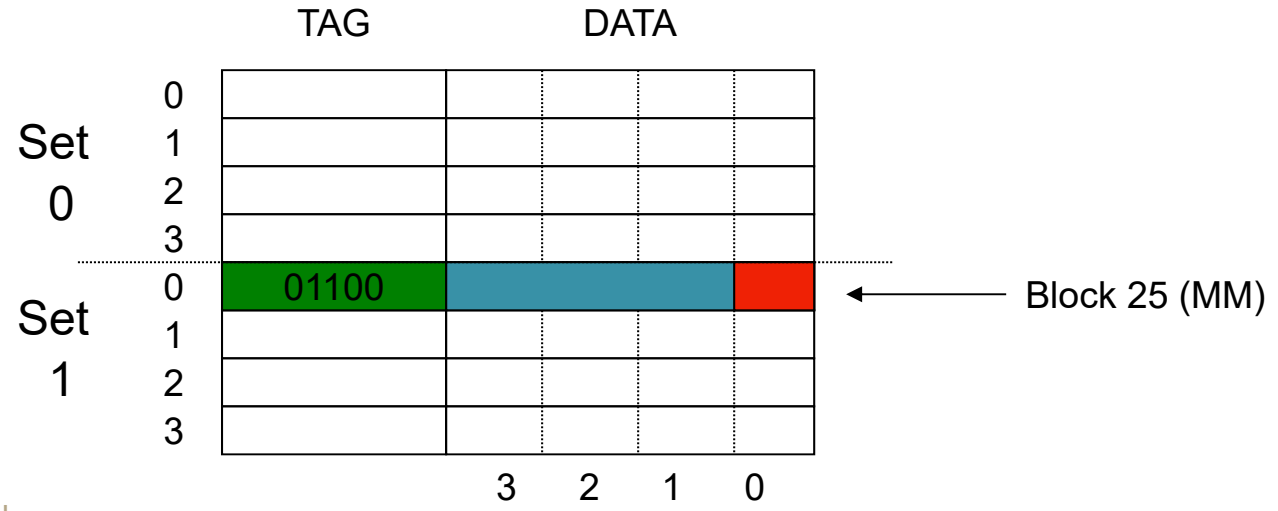


Set-Associative

Iteration #1:

- $100_{(10}$:





Set-Associative

Iteration #1:

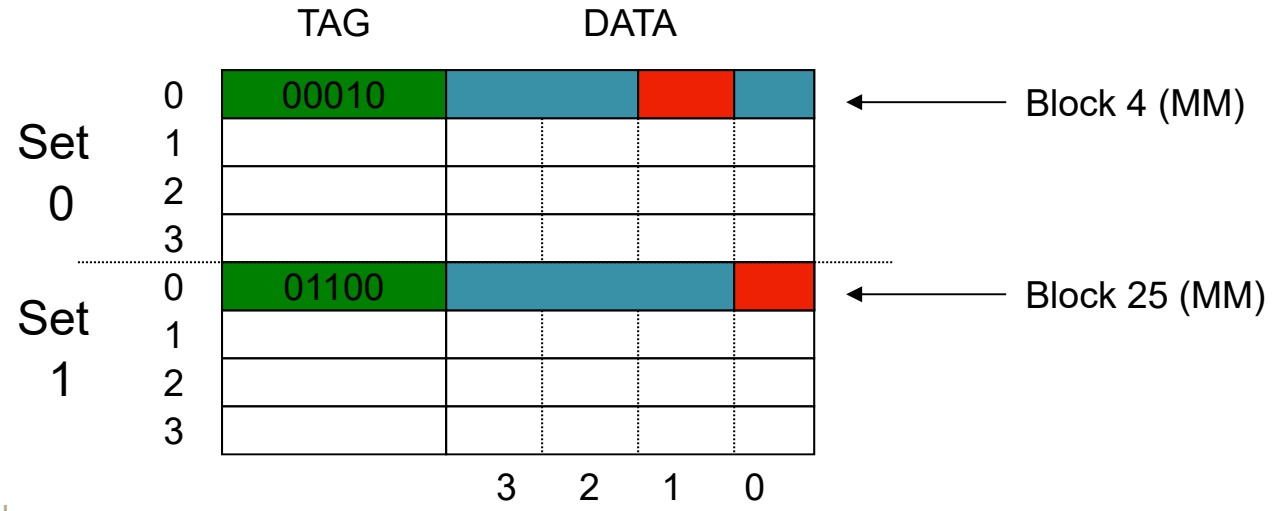
- $100_{(10}$: 011001 | 00
- $17_{(10}$: 000100 | 01 → 00010 | 0 | 01 MISS!!!!

Block 4 (MM)

TAG

Set 0

word1



Set-Associative

Iteration #1:

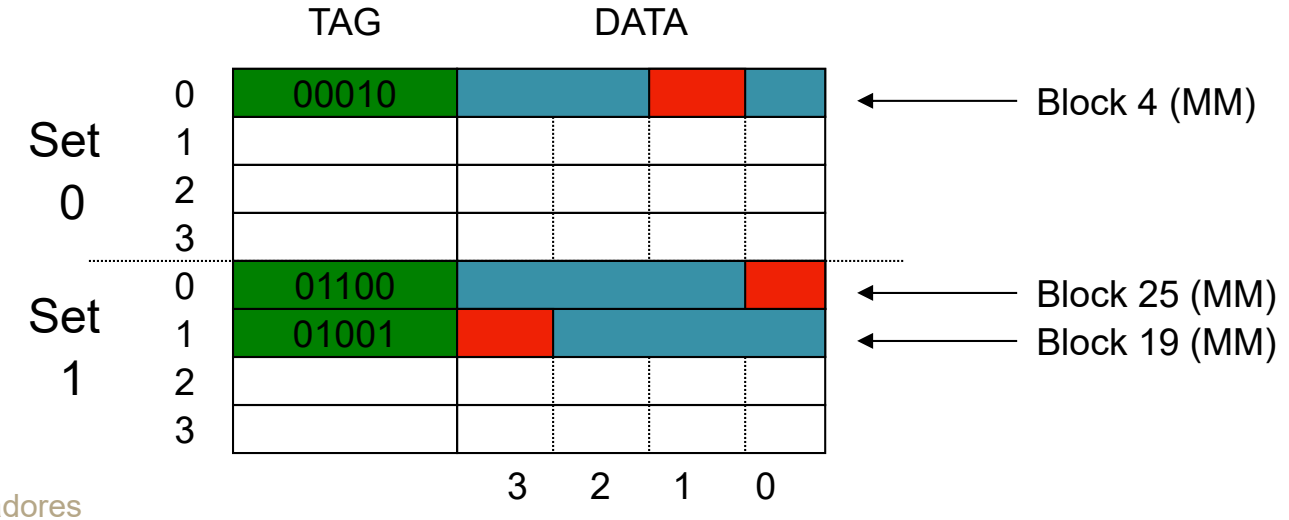
- $100_{(10}$: 011001 | 00
- $17_{(10}$: 000100 | 01
- $79_{(10}$: 010011 | 11 → 01001 | 1 | 11 MISS !!!!

Block 19 (MM)

TAG

Set 1

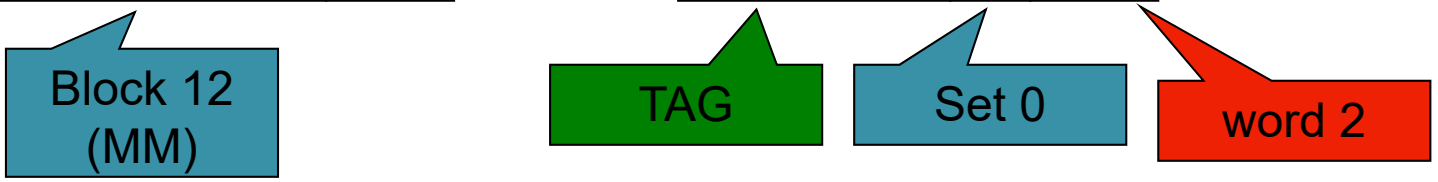
Word 3

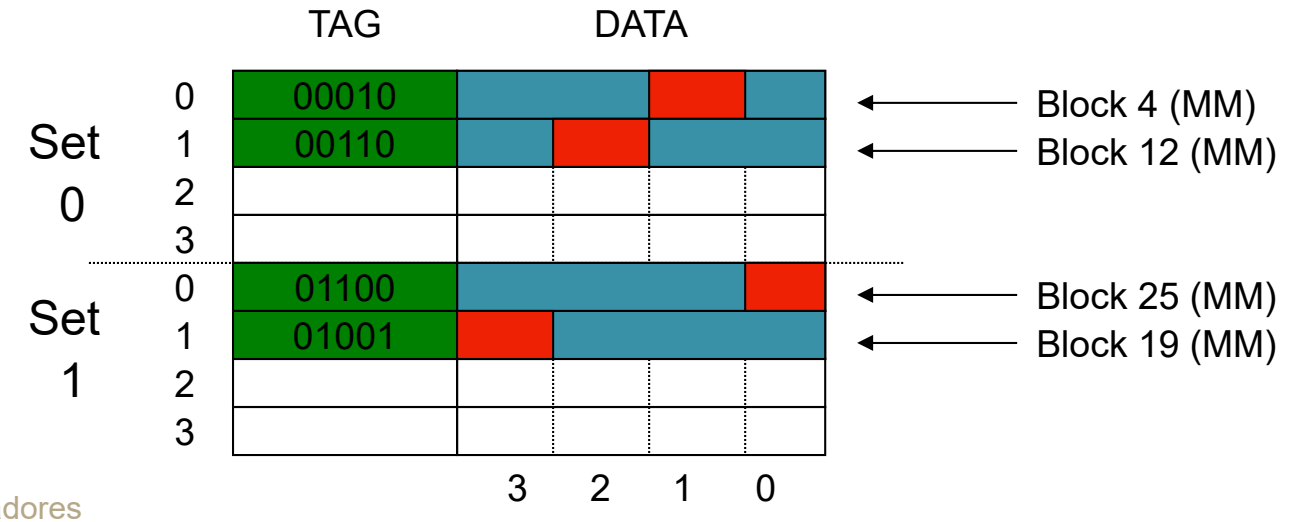


Set-Associative

Iteration #1:

- $100_{(10}$: 011001 | 00
- $17_{(10}$: 000100 | 01
- $79_{(10}$: 010011 | 11
- $50_{(10}$: 001100 | 10



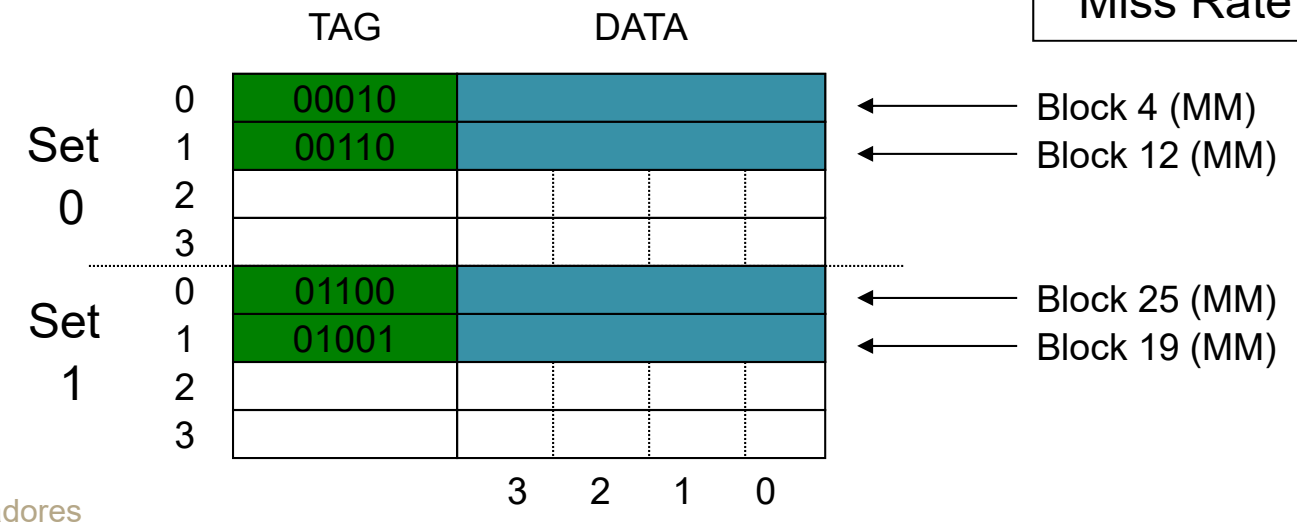


Set-Associative

Iteration #2 to …:

- $100_{(10}$: 011001 | 00 → 01100 | 1 | 00 HIT !!!!
- $17_{(10}$: 000100 | 01 → 00010 | 0 | 01 HIT !!!!
- $79_{(10}$: 010011 | 11 → 01001 | 1 | 11 HIT !!!!
- $50_{(10}$: 001100 | 10 → 00110 | 0 | 10 HIT !!!!

Miss Rate ≅ 0



Cache Replacement

- Direct-Mapping caches do not need a replacement algorigthm.
- Fully-Associative caches:
  - What if the cache is full?
- Set-Associative caches:
  - What if a given set is full?

- Most popular replacement algoritms:
  - FIFO *(First Input, First Output)*: remove oldest entry.
  - LRU *(Least Recently Used)*: remove most unused entry.
  - RANDOM: remove an entry randomly selected.

Cache Replacement

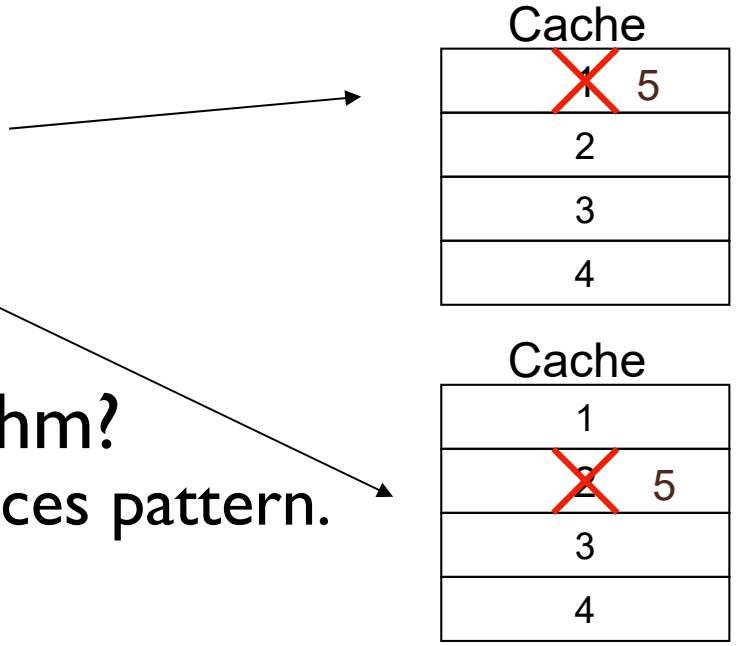
- A simple example….
  - CM with 4 entries, fully-associative.
  - References (block #): 1, 2, 3, 4, 1, 5,....

- FIFO:

  Block removed → 1

- LRU:

  Block removed → 2



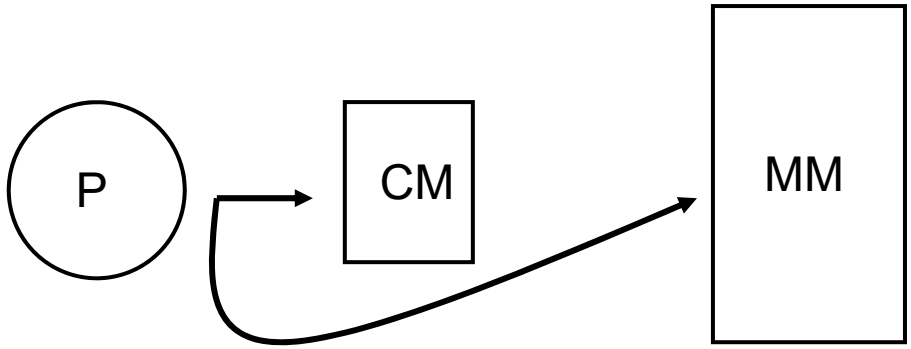
- What is the best algorithm?
  - It depends on the references pattern.
  - LRU is preferred.
  - But it is costly.
  - Alternative → Random.

Write

- Read memory operations are more usual (Fetch, LWs,…)
  - BUT write "also exists" (SW instructions).
- Supposing writings only modify cache memory…
  - When will the Main Memory updated?
- Alternatives:
  - Write-through
  - Write-back
- Additionally considerations….
  - HITS or MISS in writings:
    - Write-allocate
    - Non-write allocate

Write

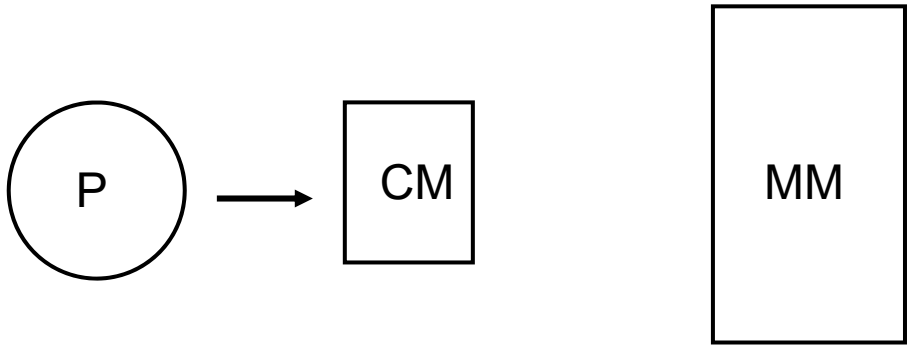
- **Write-through:**



(+) Good consistencia.

(-) CM-MM traffic.

- **Write-back:**

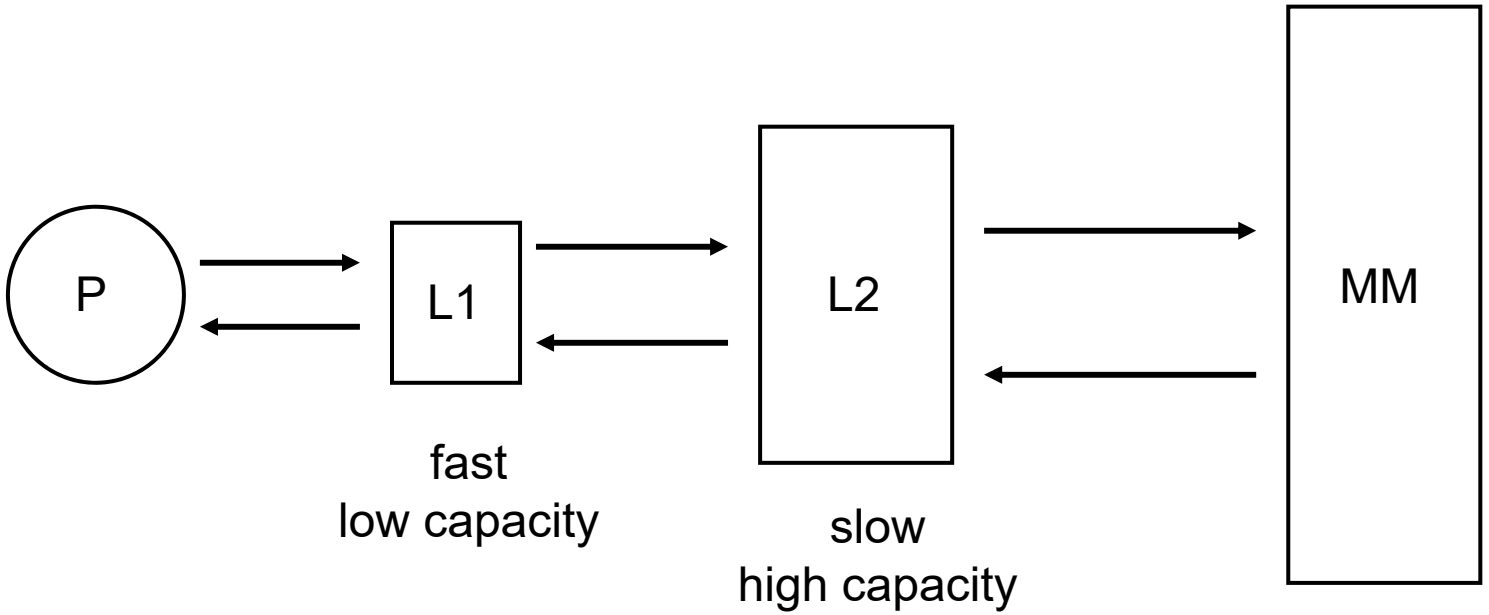


(+) Less CM-MM traffic.

(-) Memory Consistency.

Multi-level cache

- Cache Hierarchy
  - Not only a single cache level….



- Trying to reduce Miss Penalty.

Cache Splitting

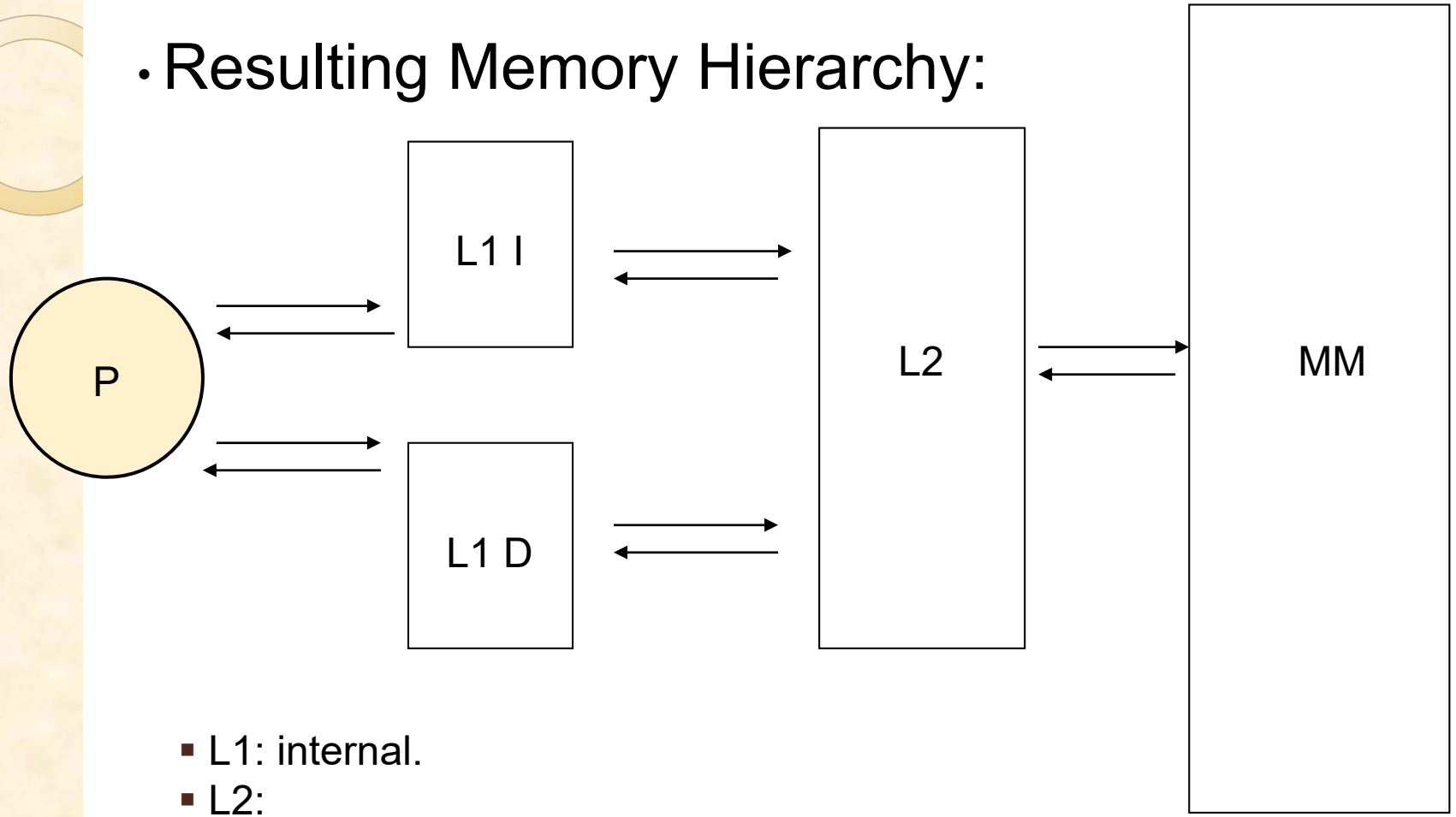
- 2 types of references: Instructions & Data.
  - Cache could be divided in two parts:
    - A less number of collisions.
    - Simultaneous access.
- Some examples….

  In current processors: L1 splitted, L2 unified.

PROCESSOR	L1 (I/D)	L2
Pentium III	16/16	256
Alpha 21364	64/64	1.5MB
PowerPC 7450	32/32	256
UltraSPARC III	32/64	1-8 MB
R10000	32/32	512-16MB
PA 8700	0.75MB/1.5MB	-

Cache Splitting

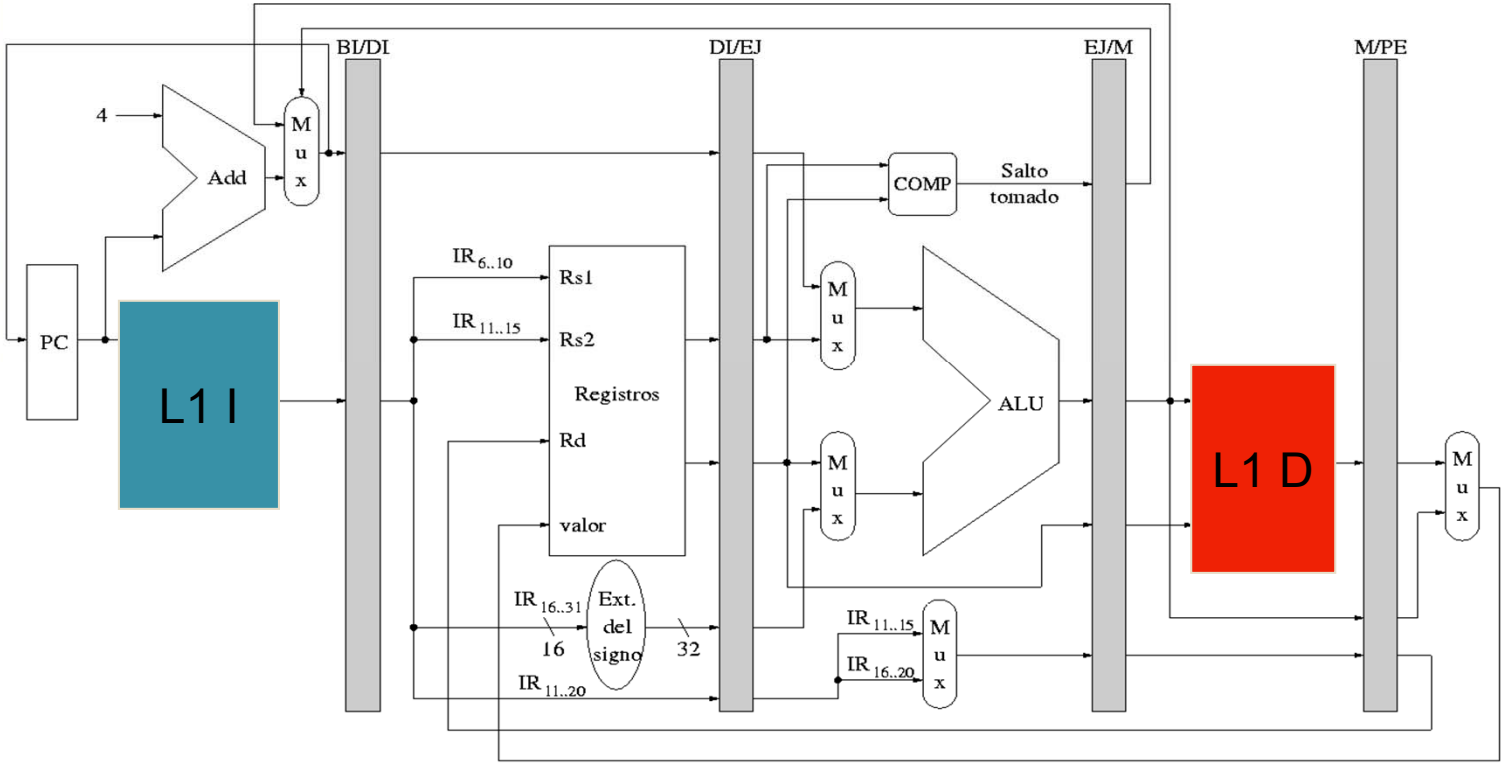
- Resulting Memory Hierarchy:



- L1: internal.
- L2:
  - internal: Pentium III, PowerPC 7450, Alpha 21364
  - external: Alpha 21264, R10000, UltraSparc III

Cache Splitting

- Utility……to be included in a datapath…



Cache Size

- Cost vs. Performance.
- BIG:
  - Más líneas en MC.
  - Más coste.
  - Más ocupación.
  - Más lenta.
  - En general, decrementa el índice de fallos.
- Ejemplos:
  - L1: 16KB - 64KB.
  - L2: 256KB - 2MB.

Block Size

- SMALL:
  - Minimize transference time with MM.
  - Contain less unnecessary information.
  - A smaller CM-MM Bus required.
- BIG:
  - Good for high spatial locality.
  - CM control reduced.

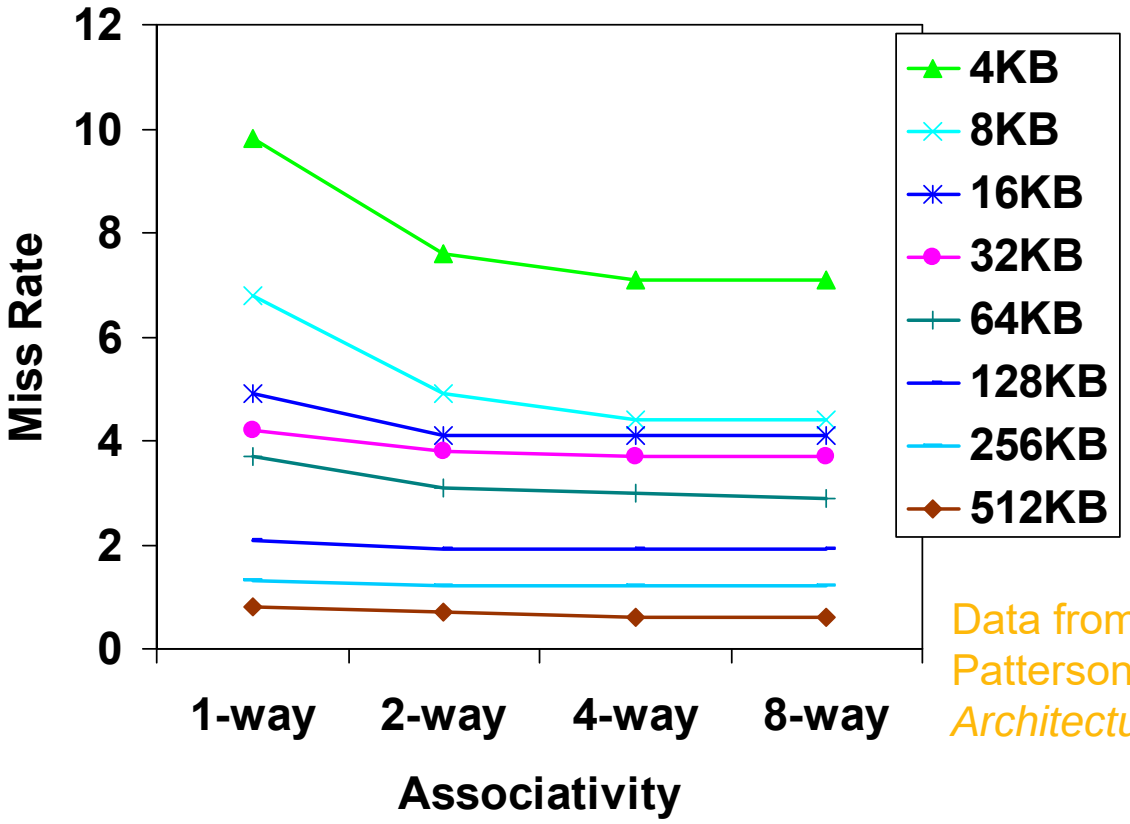
- Typical size: 32 Bytes.

Associativity

- HIGH:
  - Big number of MM blocks combinations in CM at the same time.
  - Costly:
    - Control.
    - Replacement.
- In real processors….
  - Similar performance with Fully-Associative (costly) and 8-way associative (cheaper).
  - Some examples:
    - Direct, 2, 4, 8-ways.
    - 3-ways (L2 in Alpha 21164).

Benefits of Set Associative Caches

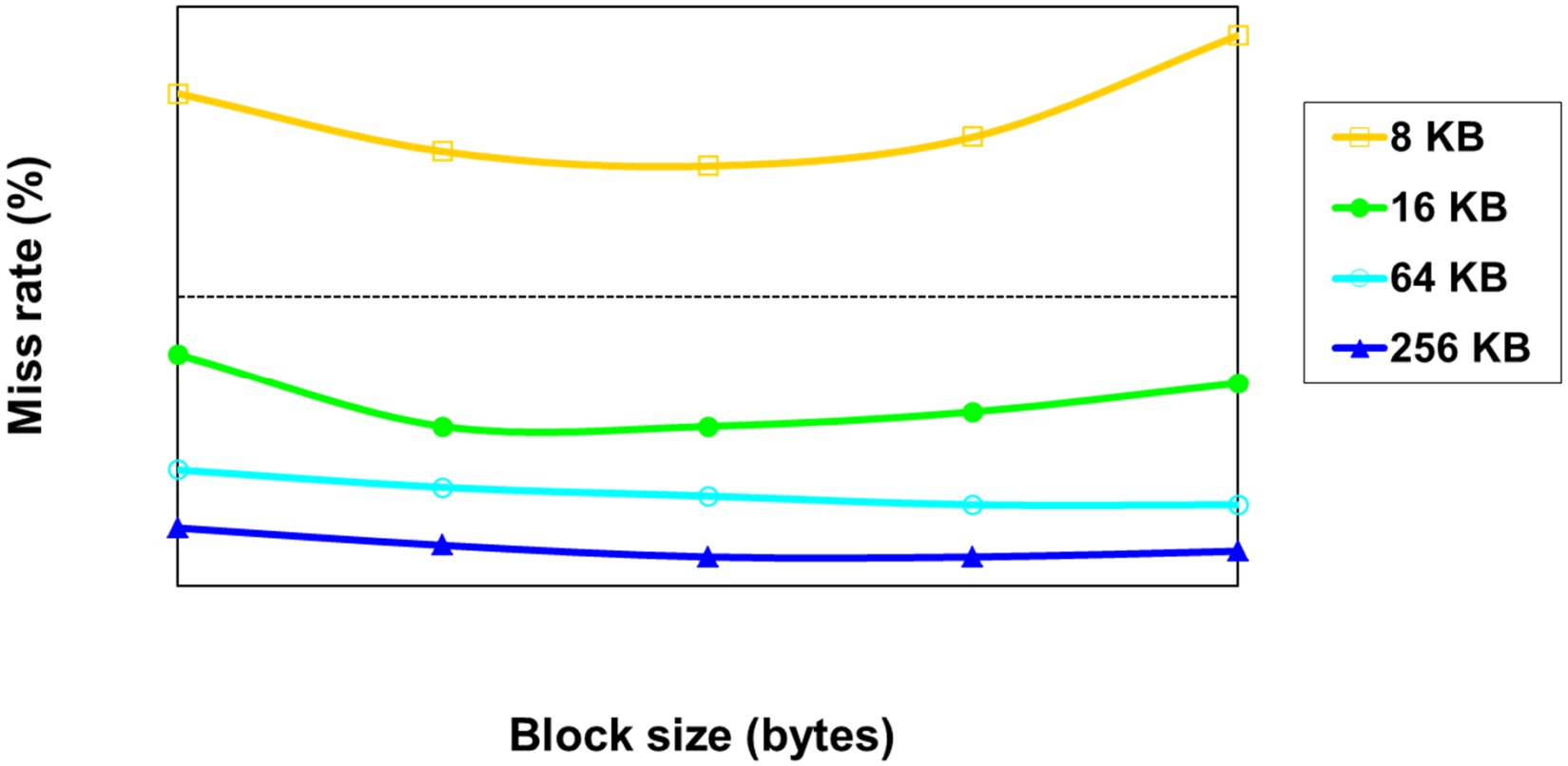
- The choice of direct mapped or set associative depends on the cost of a miss versus the cost of implementation



Data from Hennessy & Patterson, *Computer Architecture*, 2003

- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)

Miss Rate vs Block Size vs Cache Size



- Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing capacity misses)

Cache parameters in real processors….

	Intel Nehalem	AMD Barcelona
L1 cache organization & size	Split I$ and D$; 32KB for each per core; 64B blocks	Split I$ and D$; 64KB for each per core; 64B blocks
L1 associativity	4-way (I), 8-way (D) set assoc.; ~LRU replacement	2-way set assoc.; LRU replacement
L1 write policy	write-back, write-allocate	write-back, write-allocate
L2 cache organization & size	Unified; 256MB (0.25MB) per core; 64B blocks	Unified; 512KB (0.5MB) per core; 64B blocks
L2 associativity	8-way set assoc.; ~LRU	16-way set assoc.; ~LRU
L2 write policy	write-back	write-back
L2 write policy	write-back, write-allocate	write-back, write-allocate
L3 cache organization & size	Unified; 8192KB (8MB) shared by cores; 64B blocks	Unified; 2048KB (2MB) shared by cores; 64B blocks
L3 associativity	16-way set assoc.	32-way set assoc.; evict block shared by fewest cores
L3 write policy	write-back, write-allocate	write-back; write-allocate

Cache parameters in real processors….

	Intel P4	AMD Opteron
L1 organization	Split I$ and D$	Split I$ and D$
L1 cache size	8KB for D$, 96KB for trace cache (~I$)	64KB for each of I$ and D$
L1 block size	64 bytes	64 bytes
L1 associativity	4-way set assoc.	2-way set assoc.
L1 replacement	~ LRU	LRU
L1 write policy	write-through	write-back
L2 organization	Unified	Unified
L2 cache size	512KB	1024KB (1MB)
L2 block size	128 bytes	64 bytes
L2 associativity	8-way set assoc.	16-way set assoc.
L2 replacement	~LRU	~LRU
L2 write policy	write-back	write-back

Problema

Sea un código en ensamblador del MIPS como éste:

```
     #0    lw $1, %42
L2:  #1    sw $1, %50
     #2    j L1
               ...
L1:  #35   sw $0, %42
     #36   sw $2, %66
     #37   lw $3, %82
     #38   lw $1, %90
     #39   bne $1, $0, L2
```

donde se supone que la instrucción "bne" salta la primera vez y no lo hace la segunda. Las instrucciones se han enumerado empezando por "#" según su orden de ubicación en memoria empezando por la posición 0 (#35 significa instrucción número 35). Los operandos precedidos por "%" simbolizan una petición de datos, y el número que ahí aparece significa la línea de MP donde se encuentran. Todos los números ahí mostrados están en decimal.

Supongamos que tenemos L1 I (directa) y L1 D (asociativa de 4 cjtos y LRU). Ambas son de 128B y con un tamaño de línea de 8B.

Sea pide:

a) Indicar la secuencia de líneas solicitadas por el procesador.

b) Describir la evolución de los directorios de las MCs, calculando el índice de fallos.

a) Indicar la secuencia de líneas solicitadas por el procesador.

```
     #0    lw $1, %42
L2:  #1    sw $1, %50
     #2    j L1
               ...
L1:  #35   sw $0, %42
     #36   sw $2, %66
     #37   lw $3, %82
     #38   lw $1, %90
     #39   bne $1, $0, L2
```

- Líneas de D´s:

42, 50, 42, 66, 82, 90

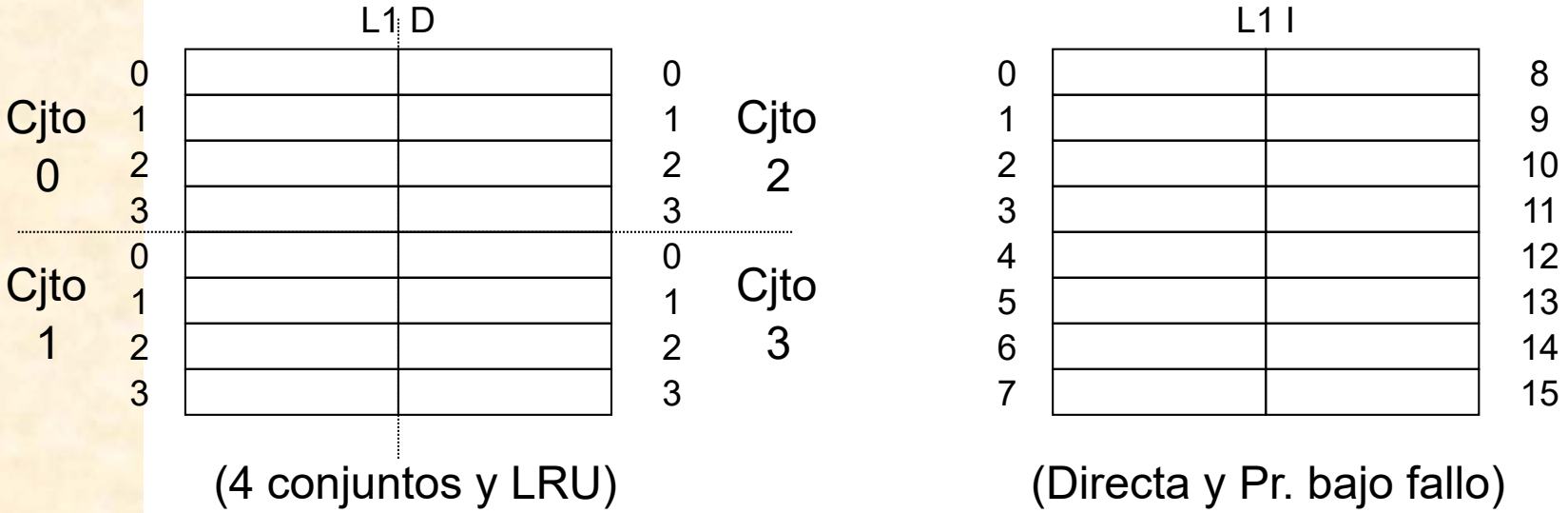
- Líneas de I´s:

(tamaño de I del MIPS = 4B → 2 I's por línea)

0, 0, 1, 17, 18, 18, 19, 19

b) Describir la evolución de los directorios de las MCs, calculando el índice de fallos y el número de precargas.

Número de líneas en MC: 128B/8B = **16 líneas**



(4 conjuntos y LRU)

(Directa y Pr. bajo fallo)

- líneas de D´s: **42, 50, 42, 66, 82, 90**
- líneas de I´s: **0, 0, 1, 17, 18, 18, 19, 19**

Instrucciones (OJO!!!!, comprob ahora SIN precarga !!!!!

Block #	COMMENTS
0	Busco en línea-0 (0 mod 16 = 0) de MC (FALLO, vacía); Demando línea-0 (en línea-0 de MC) y precargo línea-1 (en línea-1 de MC)
0	ACIERTO
1	ACIERTO, la he precargado
17	Busco en línea-1 de MC (FALLO, ocupada por otra); Demando línea-17(línea-1) y pr. la 18(línea-2)
18	ACIERTO, la he precargado
18	ACIERTO
19	Busco en línea-3 de MC (FALLO, vacía); Demando línea-19 (línea-3) y pr. Línea-20 (línea-4)
19	ACIERTO
0	ACIERTO
1	FALLO, ocupada por otra; Demando línea-1 (línea-1) y pr. línea-2 (línea-2)
17	FALLO, la acabo de reemplazar; Demando línea-17(línea-1) y pr. la 18(línea-2)
18	ACIERTO, la acabo de precargar
18	ACIERTO
19	ACIERTO, la tenía de la anterior iteración
19	ACIERTO

Instrucciones

- Directorio resultante:

L1 I

0	0000 (0)		8
1	0001 (17)		9
2	0001 (18)		10
3	0001 (19)		11
4	0001 (20)		12
5			13
6			14
7			15

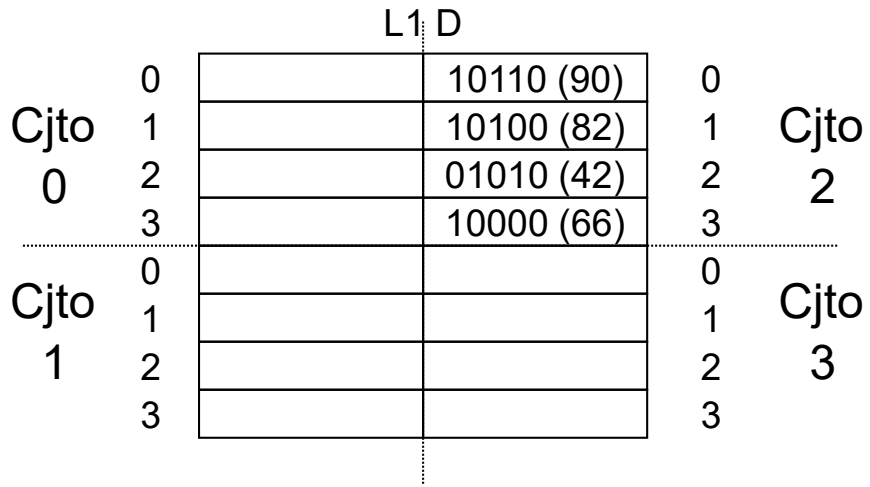
- Miss Rate = 5 / 15

Datos

Block #	COMMENTS
42	Busco en el cjto 2 (42 mod 4 = 2) de MC (FALLO, vacía); Demando línea 42 (cjto 2, línea 0(vacía))
50	Busco en el cjto 2 de MC (FALLO); Demando línea 50 (cjto 2, línea 1(vacía))
42	ACIERTO
66	Busco en el cjto 2 de MC (FALLO); Demando línea 66 (cjto 2, línea 2(vacía))
82	Busco en el cjto 2 de MC (FALLO); Demando línea 82 (cjto 2, línea 3(vacía))
90	Busco en el cjto 2 de MC (FALLO); Demando línea 90 (cjto 2, línea 1(LRU))
50	Busco en el cjto 2 de MC (FALLO); Demando línea 50 (cjto 2, línea 0(LRU))
42	Busco en el cjto 2 de MC (FALLO); Demando línea 42 (cjto 2, línea 2(LRU))
66	Busco en el cjto 2 de MC (FALLO); Demando línea 66 (cjto 2, línea 3(LRU))
82	Busco en el cjto 2 de MC (FALLO); Demando línea 82 (cjto 2, línea 1(LRU))
90	Busco en el cjto 2 de MC (FALLO); Demando línea 90 (cjto 2, línea 0(LRU))

Datos

- Directorio resultante:



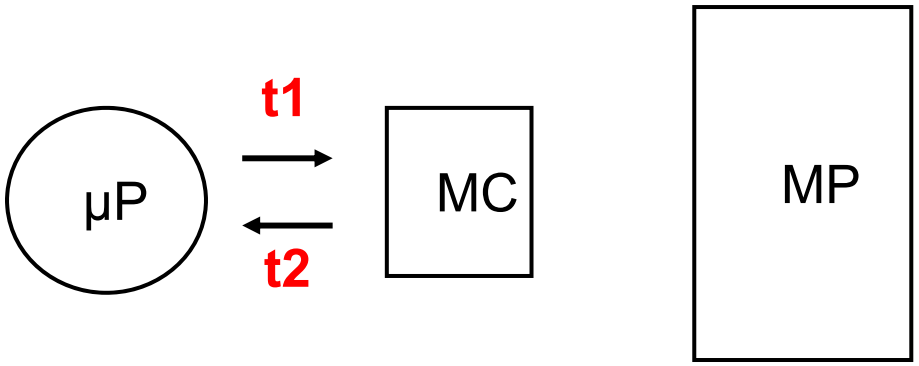
- Miss Rate = 10 / 11

Cache Performance

- HIT time (TA):

TA = tiempo de acierto = t1+t2

Tiempo de la operación de memoria cuando hay un acierto



t1: tiempo de búsqueda

Tiempo de búsqueda empleado en determinar si la palabra está o no en caché

t2: tiempo transferencia de palabra

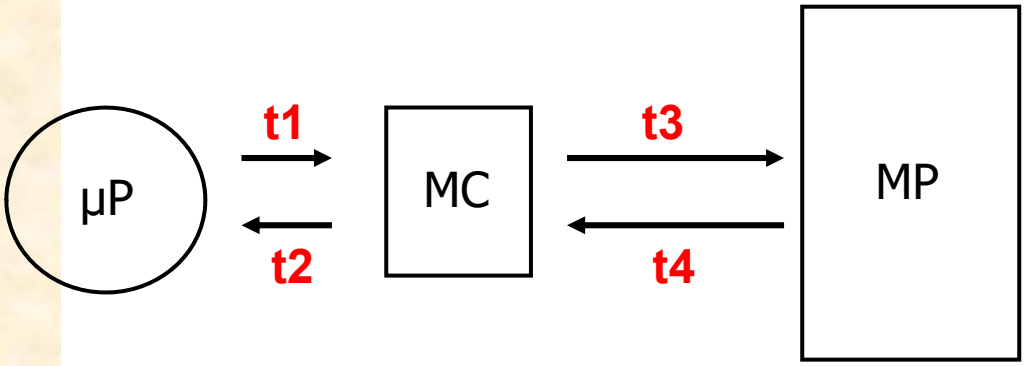
Tiempo en proporcionar la palabra al dispositivo a la CPU

Cache Performance

- MISS time (TF):

TF = tiempo de fallo = t1+ t3 + t4 +t2 = TA + PF

Tiempo de la operación de memoria cuando hay un fallo



PF = Penalización por fallo = t3 + t4 (Miss Penalty)

t3: tiempo de acceso a MP

Tiempo para acceder al bloque en MP (a su 1ª palabra). Está relacionado con la latencia de la memoria principal

t4: tiempo de transferencia de bloque

Tiempo para transferir el bloque de MP a caché. Está relacionado con el ancho de banda entre la caché y MP

Cache Performance

TMA = Tiempo medio de acceso a memoria (AMAT) =
tiempo medio requerido para satisfacer una referencia de memoria

$$\frac{\text{tiempos de aciertos} + \text{tiempos de fallos}}{\text{número de accesos a memoria}} = \frac{TA \times NA + TF \times NF}{NR} =$$

NA = NR - NF

$$= \frac{TA \times NR - TA \times NF + TF \times NF}{NR} = TA - TA \times FF + TF \times FF = TA + FF\,(TF - TA) =$$

FF = NF / NR (Miss Rate)

TF = TA + PF

$$= \boxed{TA + FF \times PF}$$

Cache Performance

- Tiempo de Ciclo (**TC**): Tiempo que determina la frecuencia máxima a la que puede trabajar un procesador.

- En el diseño de los procesadores, se procura que el tiempo de acierto sea un ciclo de reloj:

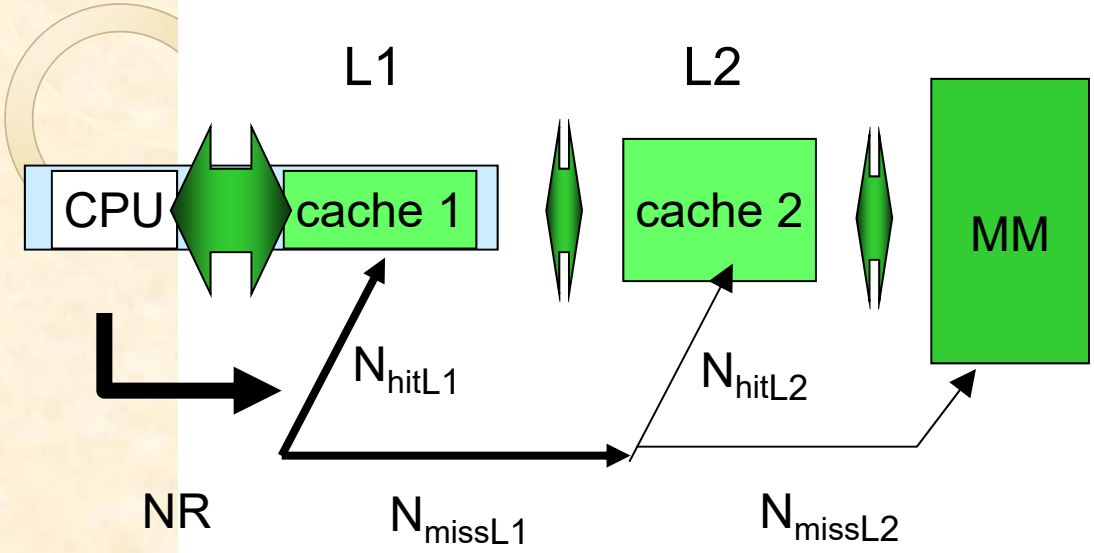
  → **TA = TC**

- Por tanto, para satisfacer este requisito de diseño, TC está condicionado por muchos factores: tamaño, tamaño de bloque, organización, asociatividad, etc…

Sources of Cache Misses

- **Compulsory** (cold start or process migration, first reference):
  - First access to a block, "cold" fact of life, not a whole lot you can do about it.
  - If you are going to run "millions" of instruction, compulsory misses are insignificant
  - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- **Capacity:**
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size (may increase access time)
- **Conflict** (collisions):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size.
  - Solution 2: increase associativity (may increase access time)

Multi-level cache: Global vs Local Miss Rate



NR = Number of References
N_{hit} = Number of Hits
N_{miss} = Number of Misses

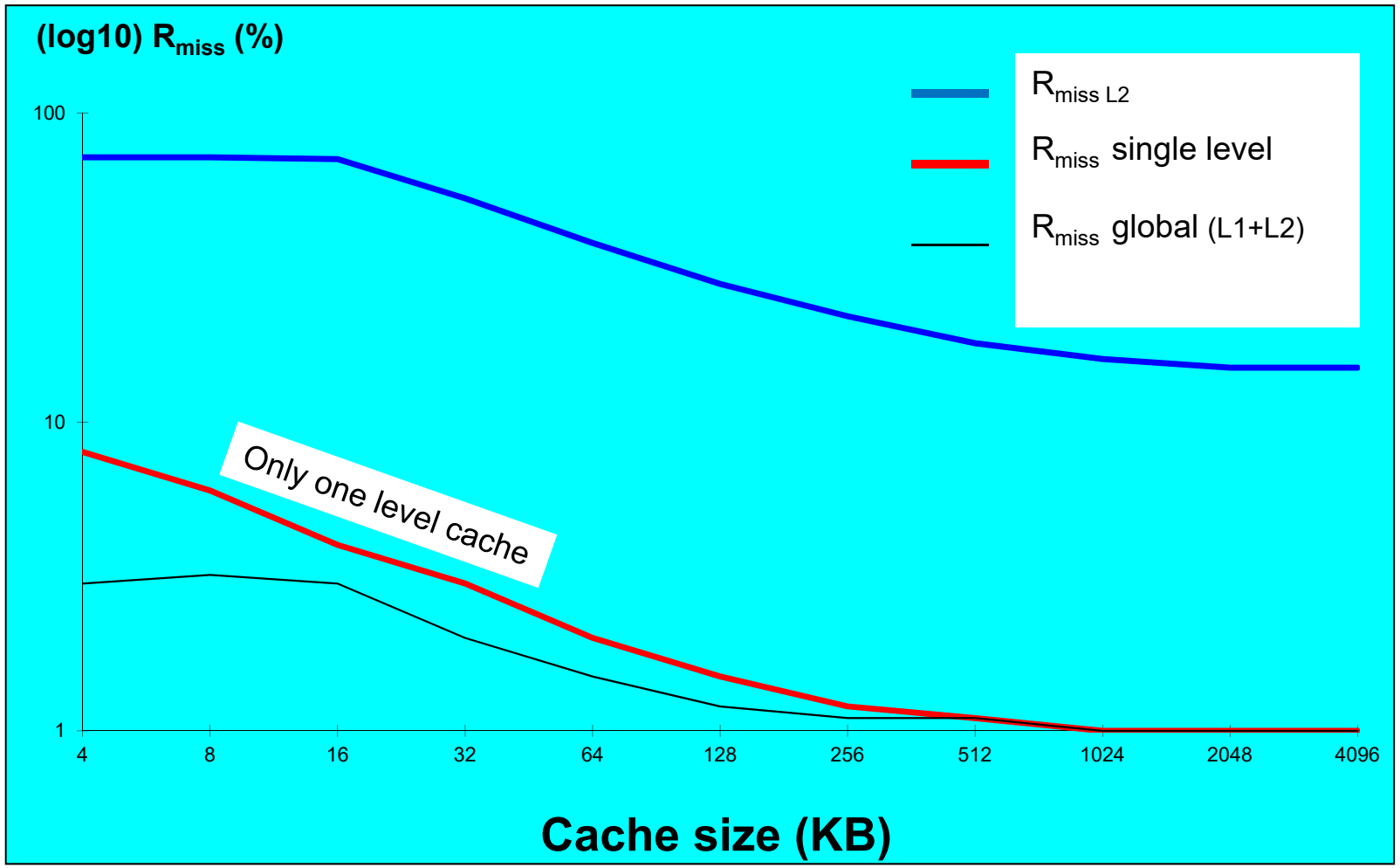
$$NR = N_{hitL1} + N_{missL1} = N_{hitL1} + (N_{hitL2} + N_{hitL2})$$

$$R_{missL1} = \frac{N_{missL1}}{NR} \rightarrow \textbf{Local miss rate L1}$$

$$R_{missL2} = \frac{N_{missL2}}{N_{missL1}} \rightarrow \textbf{Local miss rate L2}$$

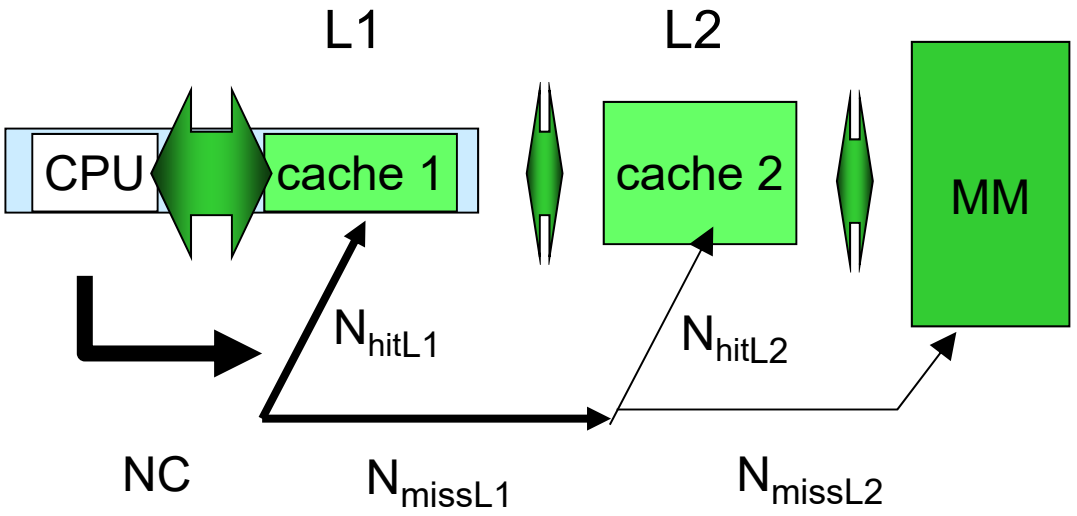
$$R_{miss} = \frac{N_{missL2}}{NR} = \frac{N_{missL1}}{NR} \times \frac{N_{missL2}}{N_{missL1}} = \mathbf{R_{missL1} \times R_{missL2}} = \textbf{global miss rate}$$

Multi-level cache vs. Single-level cache (Miss rate)



Multi-level cache: AMAT

$$AMAT = T_{hit} + R_{miss} \times P_{miss}$$



$$AMAT = AMAT_{L1} = T_{hit\ L1} + R_{miss\ L1} \times P_{missL1}$$

$$= AMAT_{L2} = T_{hit\ L2} + R_{miss\ L2} \times P_{missL2}$$

$$\rightarrow \quad AMAT = T_{hit\ L1} + R_{miss\ L1} \times (T_{hitL2} + R_{miss\ L2} \times P_{missL2})$$

Multi-level cache: AMAT example

- Given:
  - 2500 references (NR= 2500)
  - 50 misses at L1 (N_{missL1}=50)
  - 5 misses at L2 (N_{missL2}=5)
  - Miss penalty for L2= 100 cc (P_{missL2}=100)
  - Access time for L2= 12 cc (T_{hitL2} = 12)
  - Hit time at L1= 1 cc
- **Global miss rate??**
  - R_{missL1} = 50/2500 = 0,02 R_{missL2} = 5/50 = 0.10
  - R_{miss} = 0,02 x 0.10 = 0,002
- **Average Memory Access Time??**
  - $AMAT_{L2}$= 12 + 0.10x100 = 22
  - AMAT= 1 + 0,02 x 22 = 1.44

Measuring Cache Performance

- Assuming cache hit costs are included as part of the normal CPU execution cycle, then:

$$\text{CPU time} =$$
$$= IC \times CPI_{effective} \times CC =$$
$$= IC \times (CPI_{base} + CPI_{mem}) \times CC$$

- *CPI_{base}: includes stalls due to data & control hazards (i.e. CPI of previous chapter)*
- *CPI_{mem}: includes stalls due to cache misses:*

Memory-stall cycles/num. inst.

Measuring Cache Performance

CPI_{mem} = Memory-stall cycles/num. inst.

Memory-stall cycles = Inst.-stall cycles + Data-stall cycles

- Inst.-stall cycles = IC × Miss Rate (Instr.) × Miss Penalty (Instr.)
- Data-stall cycles = #Data accesses × Miss Rate (D) × Miss Penalty (D)

AND/OR

Memory-stall cycles = Read-stall cycles + Write-stall cycles

- Read-stall cycles = #Reads × Miss Rate (R) × Miss Penalty (R)
- Write-stall cycles = #Writes × Miss Rate (W) × Miss Penalty (W)

but for Write-back caches….

- Write-stall cycles = (#Writes × Miss Rate (W) × Miss Penalty (W)) + #write buffer stalls

Cache Performance example: I$ and D$

- Given:
  - I$ miss rate R_{miss}(I$)= 2%
  - D$ miss rate R_{miss}(D$)= 4%
  - Miss penalty (I$ & D$) P_{miss}(I$), P_{miss}(D$) = 80 cycles
  - Base CPI (ideal cache) CPI_{base}= 1.5
  - Load & stores are 25% of instructions *(Num.acc.D$ / num. inst = 0.25)*

- Miss cycles per instruction:
  - I$: 0.02 × 80 = 1.6
  - D$: 0.25 × 0.04 × 80 = 0.8

- $CPI_{effective}$ = 1.5 + 1.6 + 0.8 = 3.9

Cache Performance example: Multi-level cache

- Given:
  - $CPI_{base} = 2$
  - 100 cycle miss penalty (to main memory)
  - 25 cycle miss penalty (to L2)
  - 36% load/stores
  - 2% L1I miss rate
  - 4% L1D miss rate
  - 0.5% L2 miss rate (I & D)

- $CPI_{effective} = 2 + 0.02 \times 25 + 0.36 \times 0.04 \times 25 + 0.005 \times 100 + 0.36 \times 0.005 \times 100 = 3.54$

- $CPI_{effective} = 5.44$ (with no L2)

Index

4.1.- Introduction

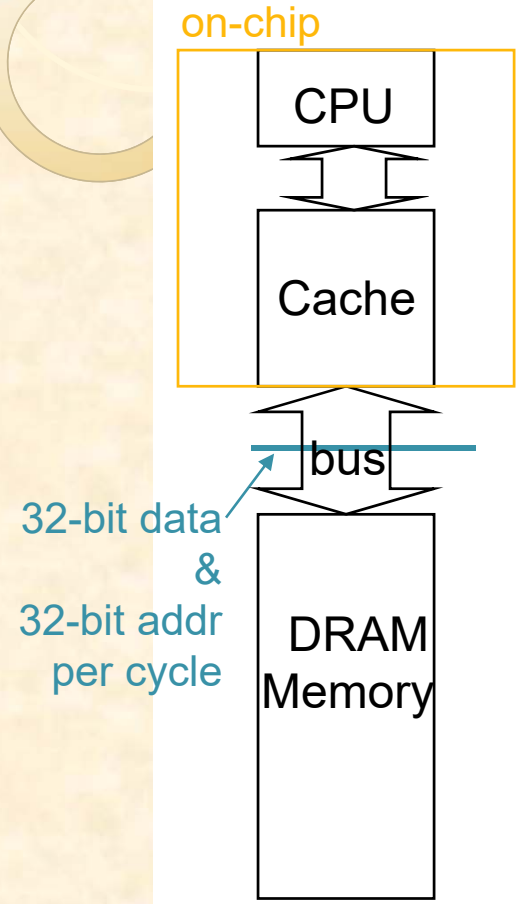
4.2.- Cache Memory

4.3.- Interleaved Memory

- Virtual Memory

Memory Systems that Support Caches

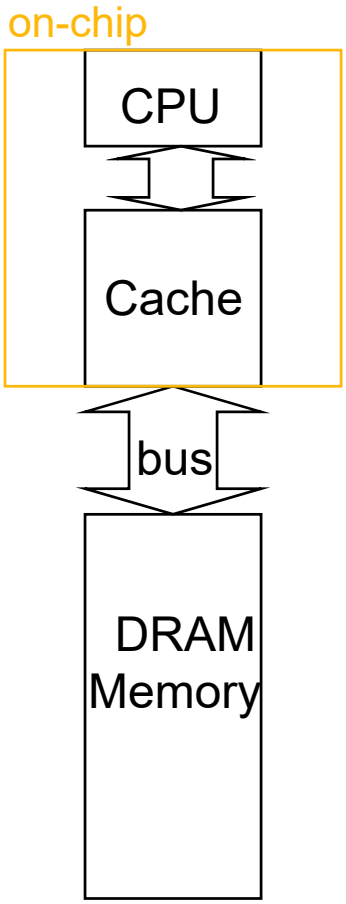
- The off-chip interconnect and memory architecture can affect overall system performance in dramatic ways



One word wide organization (one word wide bus and one word wide memory)

- Assume
  1. 1 memory bus clock cycle to send the addr
  2. 15 memory bus clock cycles to get the 1st word in the block from DRAM (row cycle time), 5 memory bus clock cycles for 2nd, 3rd, 4th words (column access time)
  3. 1 memory bus clock cycle to return a word of data
- Memory-Bus to Cache bandwidth
  - number of bytes accessed from memory and transferred to cache/CPU per memory bus clock cycle

One Word Wide Bus, One Word Blocks



- If the block size is one word, then for a memory access due to a cache miss, the processor will have to stall for the number of cycles required to return one data word from memory

  cycle to send address……

  cycles to read DRAM……

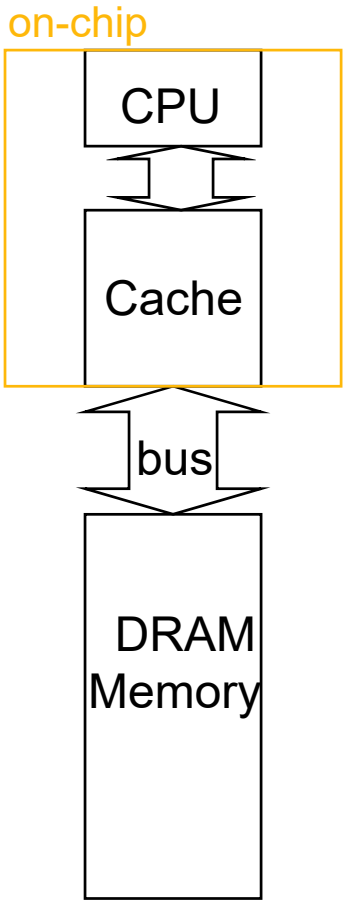
  cycle to return data……..... ____

  TOTAL clock cycles miss penalty...

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is:

  bytes per memory bus clock cycle….

One Word Wide Bus, One Word Blocks



- If the block size is one word, then for a memory access due to a cache miss, the processor will have to stall for the number of cycles required to return one data word from memory

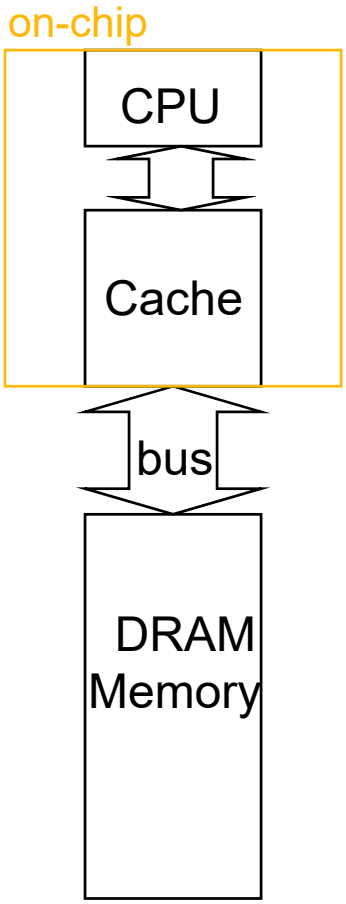
  cycle to send address…… 1
  cycles to read DRAM…… 15
  cycle to return data…….... 1

  TOTAL clock cycles miss penalty... 17

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is:

  bytes per memory bus clock cycle….
  4/17 = 0.235

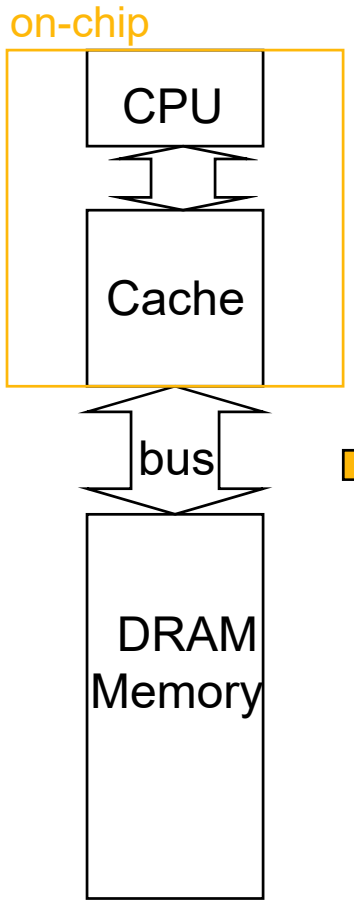
One Word Wide Bus, Four Word Blocks



- What if the block size is four words and each word is in a different DRAM row?
  - cycle to send 1st address……
  - cycles to read DRAM……..
  - cycles to return last data word…...
  - TOTAL clock cycles miss penalty

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is:
  - bytes per clock…..

One Word Wide Bus, Four Word Blocks



- What if the block size is four words and each word is in a different DRAM row?

  cycle to send 1st address…… 1

  cycles to read DRAM…….. 4 *15 =60

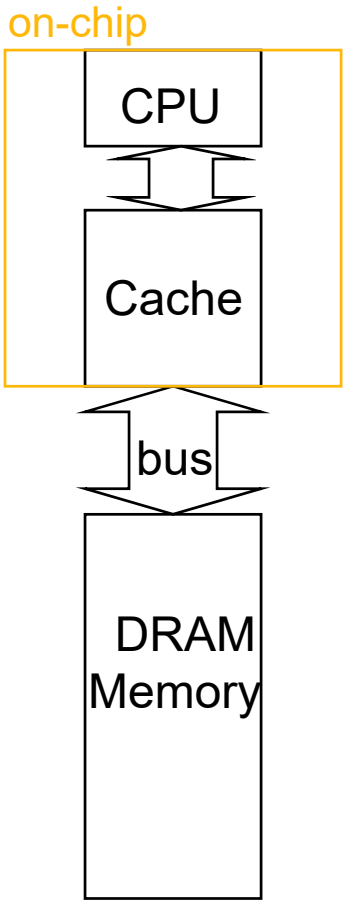
  cycles to return last data word…... 1

  TOTAL clock cycles miss penalty 62

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is:

  bytes per clock….. (4 x 4)/62 = 0.258

One Word Wide Bus, Four Word Blocks



- What if the block size is four words and each word is in the same DRAM row?

  cycle to send 1st address……

  cycles to read DRAM……..

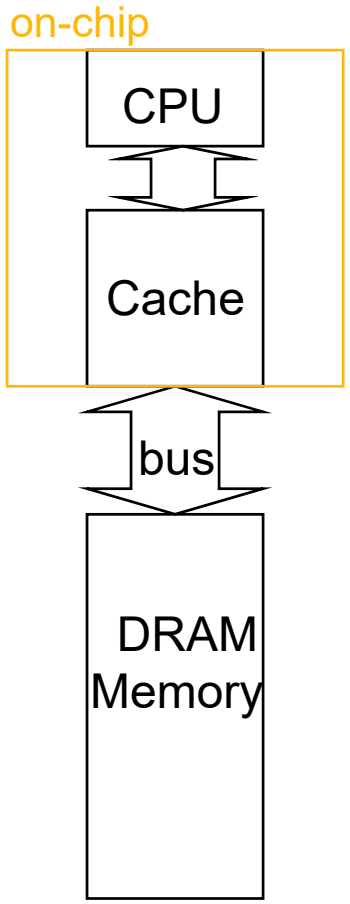
  cycles to return last data word…...

  TOTAL clock cycles miss penalty

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is:

  bytes per clock…..

One Word Wide Bus, Four Word Blocks



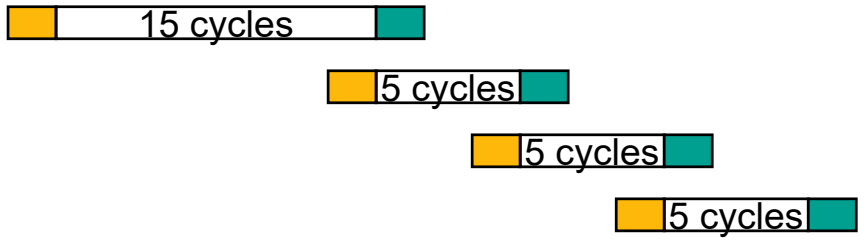
- What if the block size is four words and each word is in the same DRAM row?

  cycle to send 1st address…… 1

  cycles to read DRAM……. 15+ 3*5 =30

  cycles to return last data word…... 1

  TOTAL clock cycles miss penalty 32



- Number of bytes transferred per clock cycle (bandwidth) for a single miss is:

  bytes per clock….. (4 x 4)/32 = 0.5

Interleaved Memory, One Word Wide Bus

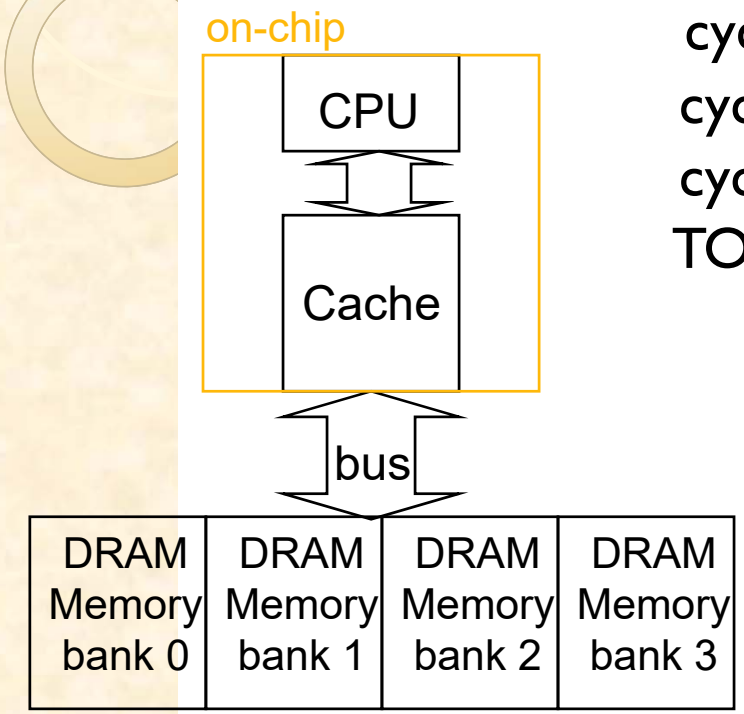
- For a block size of four words

  cycle to send 1st address……

  cycles to read DRAM banks…….

  cycles to return last data word…

  TOTAL clock cycles miss penalty



- Number of bytes transferred per clock cycle (bandwidth) for a single miss is:

  bytes per clock…..

Interleaved Memory, One Word Wide Bus

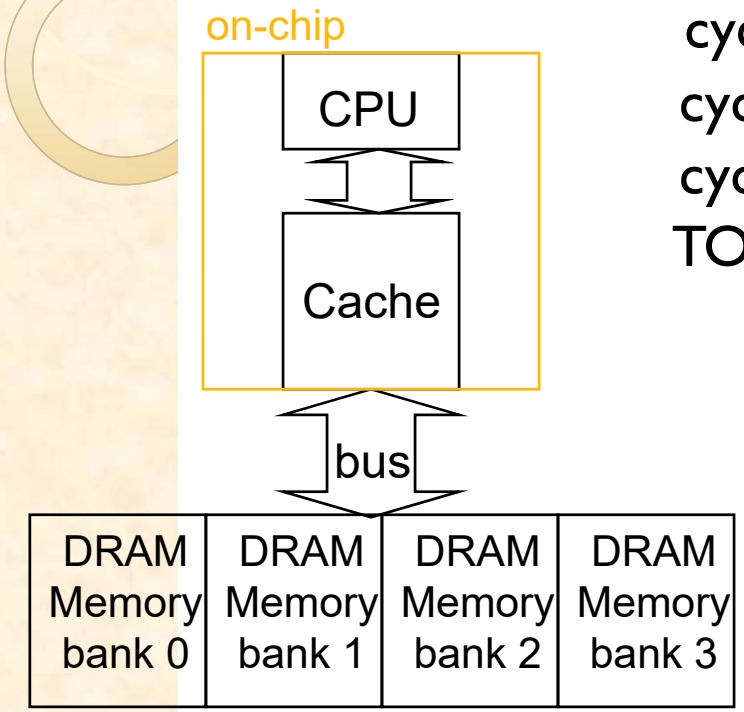
- For a block size of four words

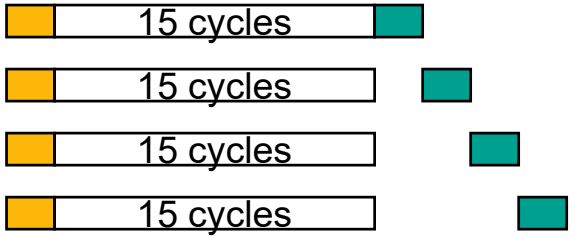
  cycle to send 1st address…… 1

  cycles to read DRAM banks……. 15

  cycles to return last data word… 4 * 1 = 4

  TOTAL clock cycles miss penalty 20





- Number of bytes transferred per clock cycle (bandwidth) for a single miss is:

  bytes per clock….. (4 x 4)/20 = 0.8

Index

4.1.- Introduction

4.2.- Cache Memory

4.3.- Interleaved Memory

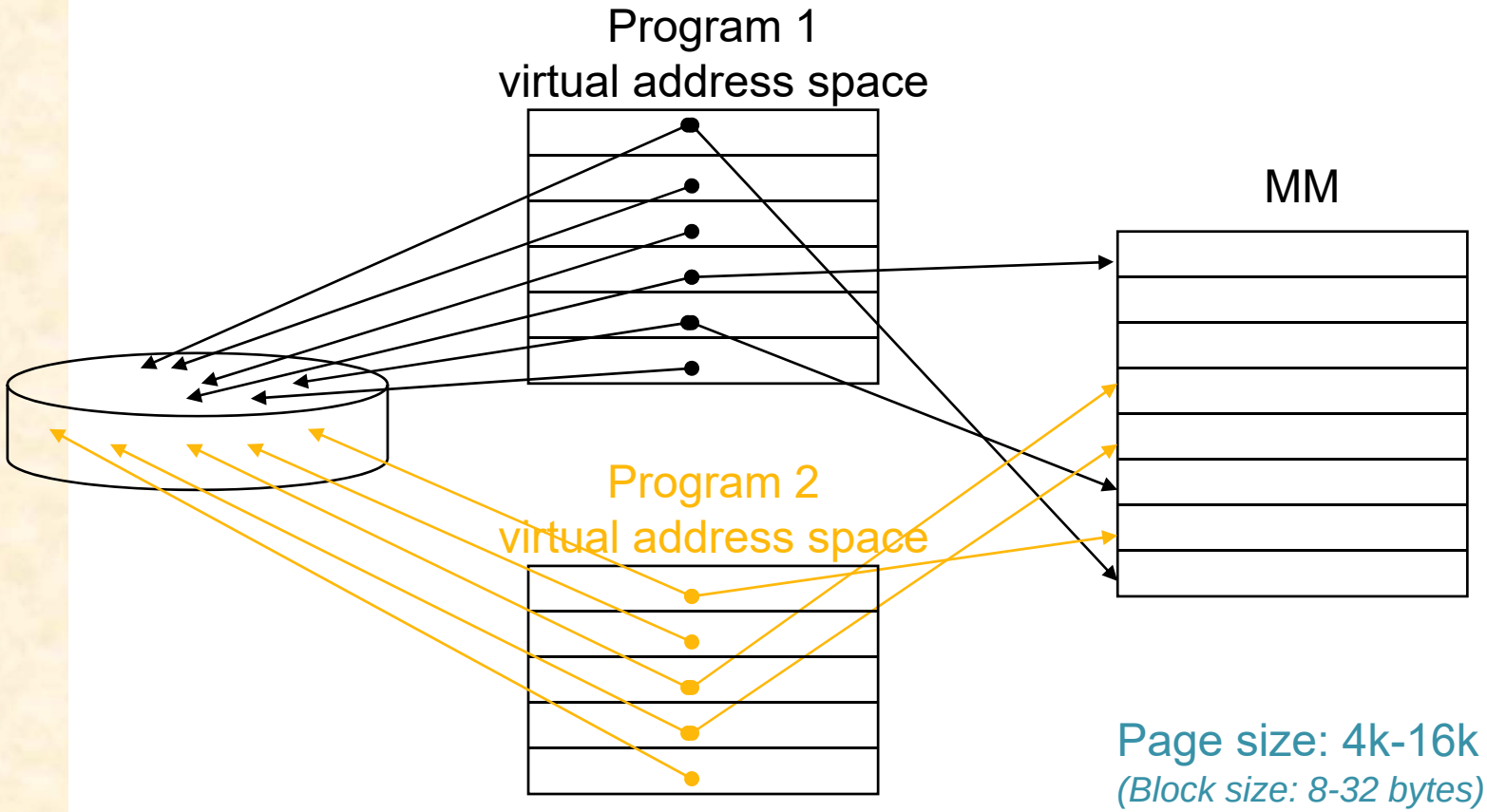
4.4.- Virtual Memory

Virtual Memory

- Use main memory as a "cache" for secondary memory
  - Allows efficient and safe sharing of memory among multiple programs
  - Provides the ability to easily run programs larger than the size of physical memory
  - Simplifies loading a program for execution by providing for code relocation (i.e., the code can be loaded anywhere in main memory)
- What makes it work? – again the Principle of Locality
  - A program is likely to access a relatively small portion of its address space during any period of time
- Each program is compiled into its own address space – a "virtual" address space
  - During run-time each virtual address must be translated to a physical address (an address in main memory)

Two Programs Sharing Physical Memory

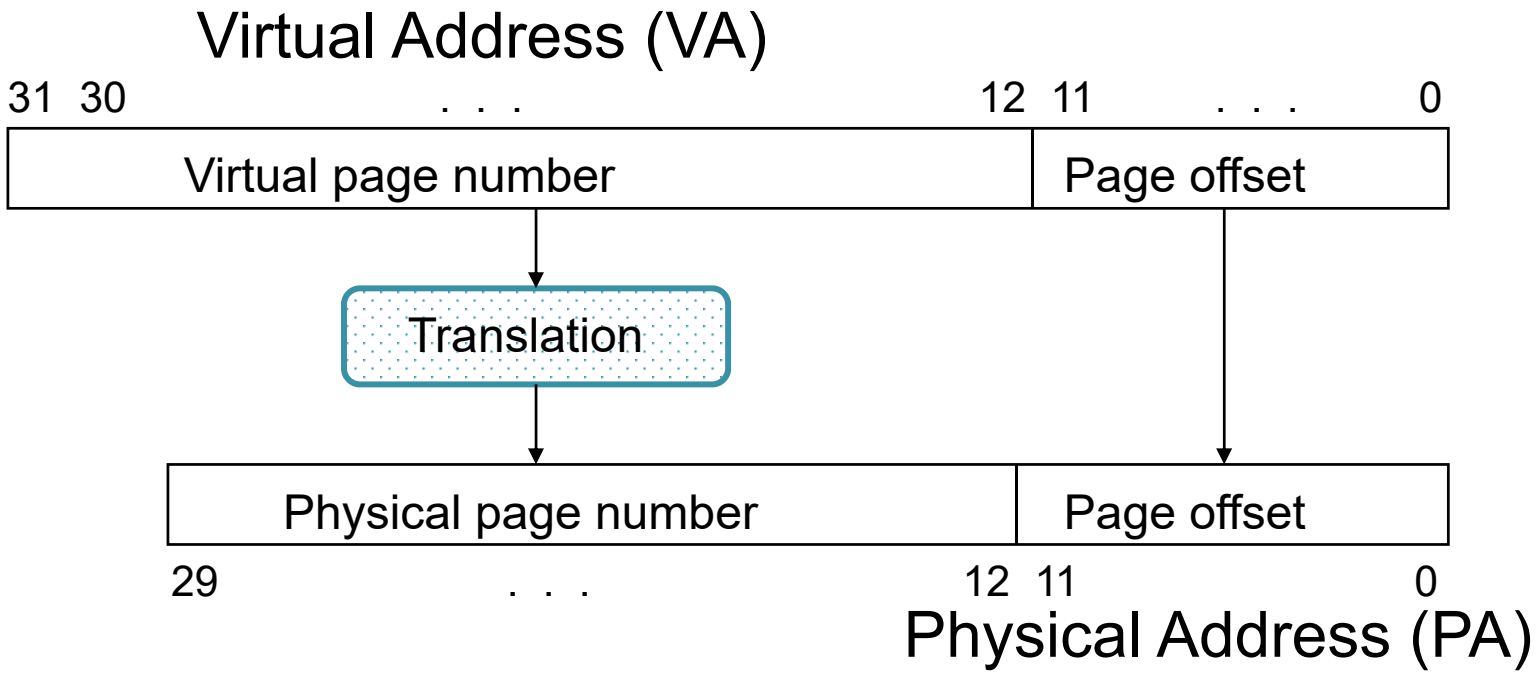
- A program's address space is divided into pages (all one fixed size) or segments (variable sizes).
  - The starting location of each page (either in MM or in secondary memory) is contained in the program's page table



Page size: 4k-16k
(Block size: 8-32 bytes)

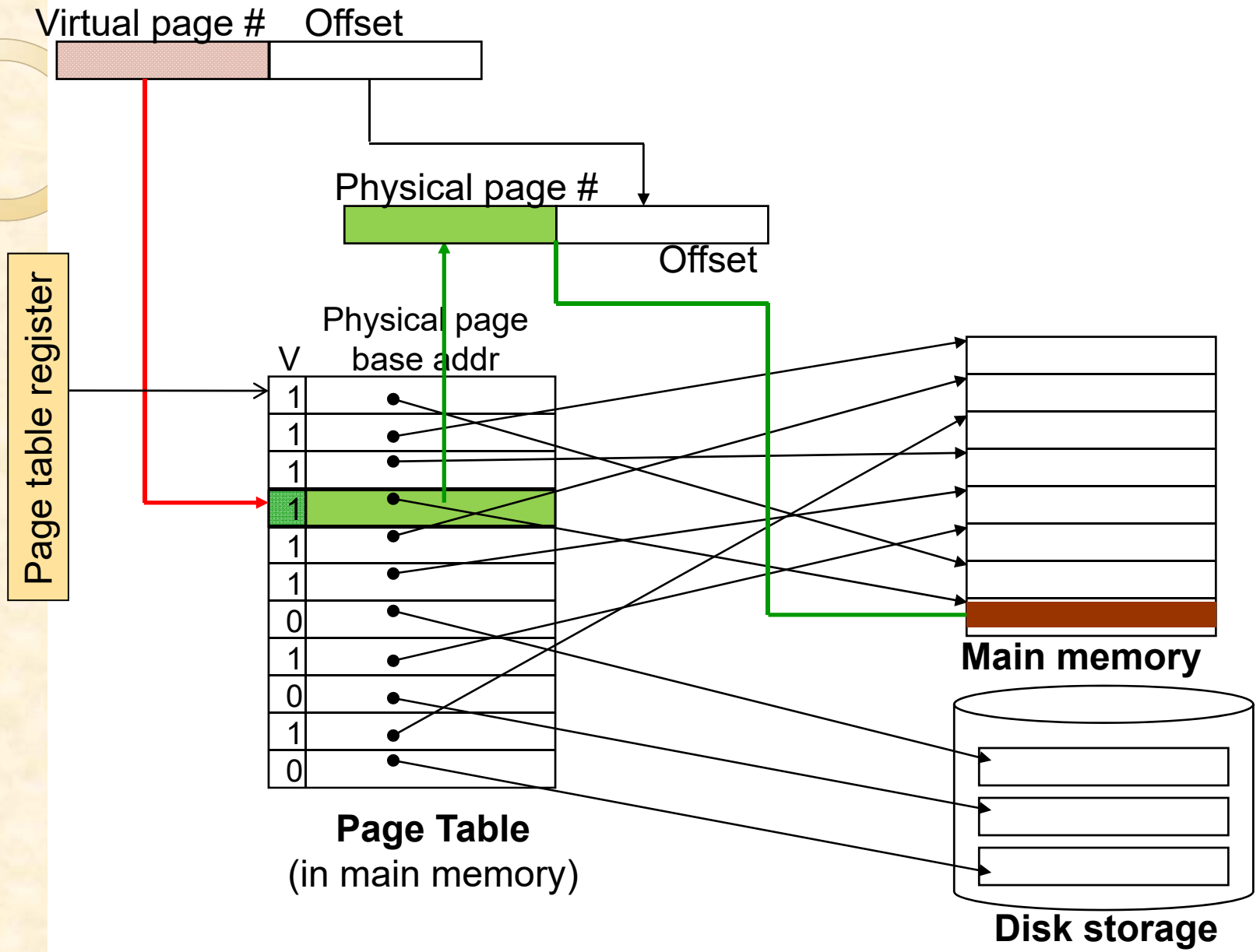
Address Translation

- A virtual address is translated to a physical address by a combination of hardware and software



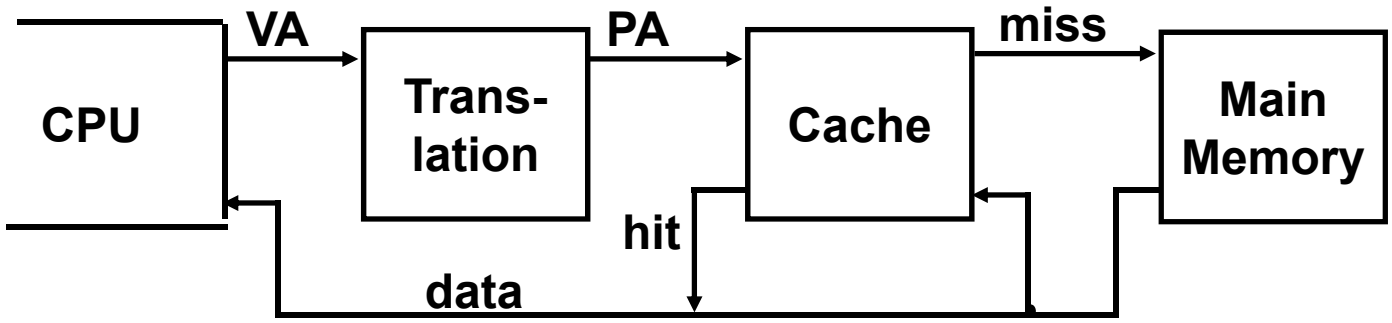
- So each memory request *first* requires an address translation from the virtual space to the physical space
  - A virtual memory miss (i.e., when the page is not in physical memory) is called a page fault → penalty: 1.000.000 cycles

Address Translation Mechanism



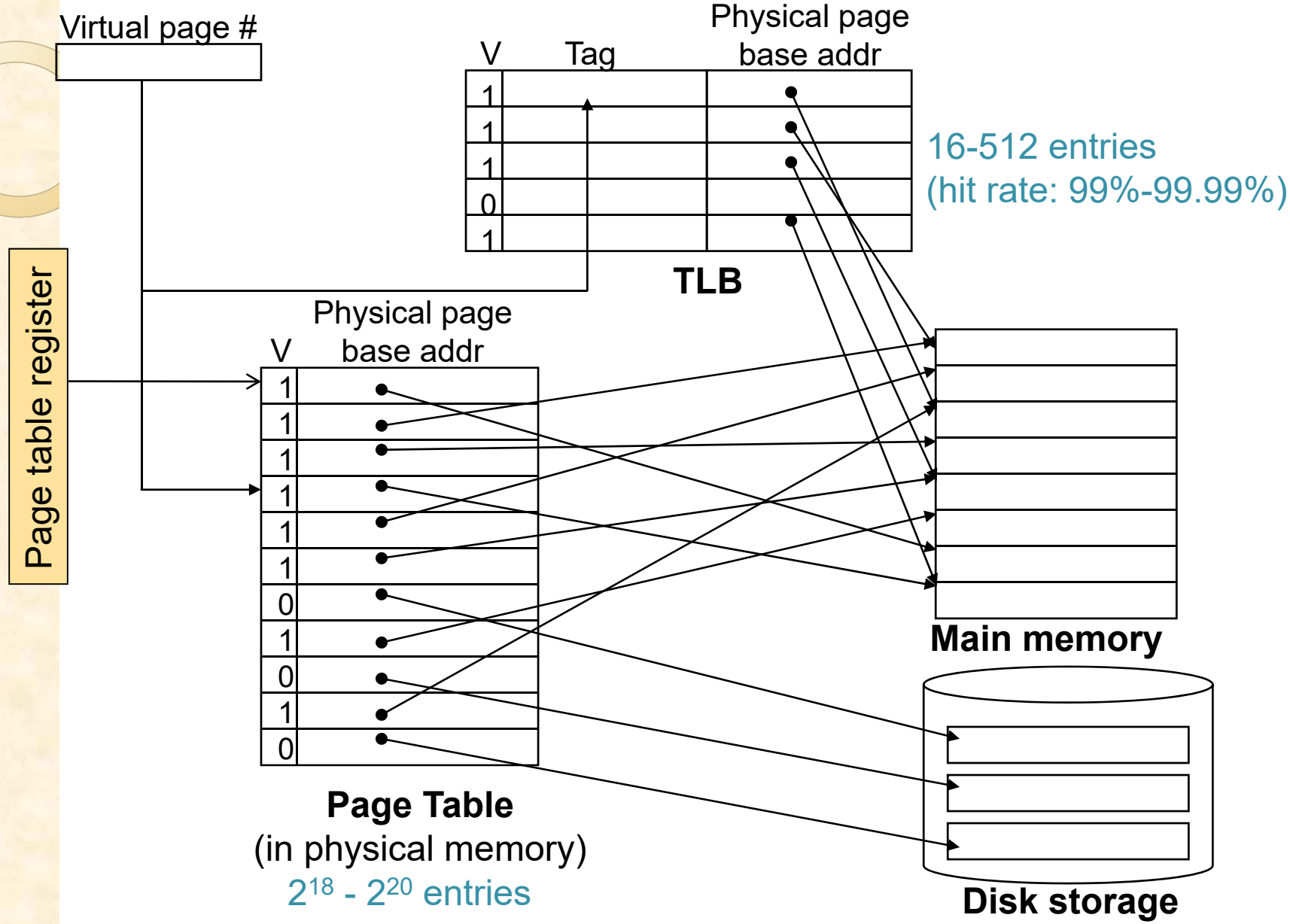
Virtual Addressing with a Cache

- Thus it takes an *extra* memory access to translate a VA to a PA



- This makes memory (cache) accesses very expensive (if every access was really *two* accesses)
- The hardware fix is to use a Translation Lookaside Buffer (TLB) – a small cache that keeps track of recently used address mappings to avoid having to do a page table lookup

Making Address Translation Fast



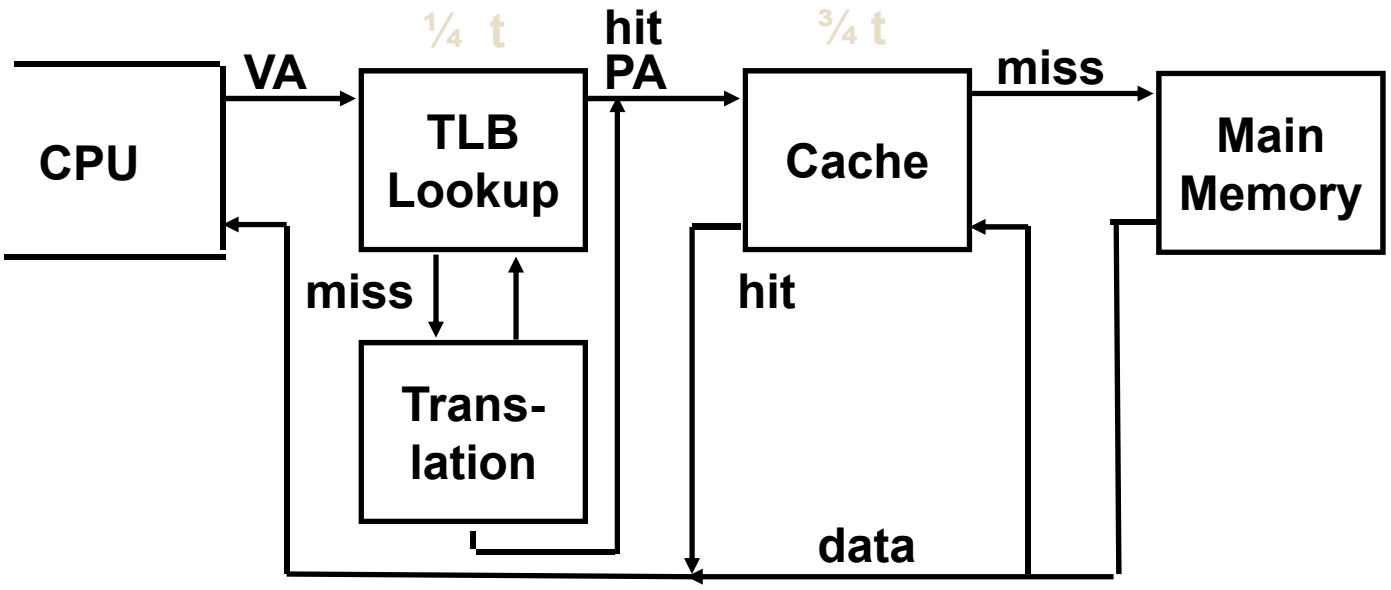
Translation Lookaside Buffers (TLBs)

- Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped

V	Virtual Page #	Physical Page #	Dirty	Ref	Access

- TLB access time is typically smaller than cache access time (because TLBs are much smaller than caches)
  - TLBs are typically not more than 512 entries even on high end machines

A TLB in the Memory Hierarchy



- A TLB miss – is it a page fault or merely a TLB miss?
  - If the page is loaded into main memory, then the TLB miss can be handled (in hardware or software) by loading the translation information from the page table into the TLB
    - Takes 10's of cycles to find and load the translation info into the TLB
  - If the page is not in main memory, then it's a true page fault
    - Takes 1,000,000's of cycles to service a page fault
- TLB misses are much more frequent than true page faults

TLB Event Combinations

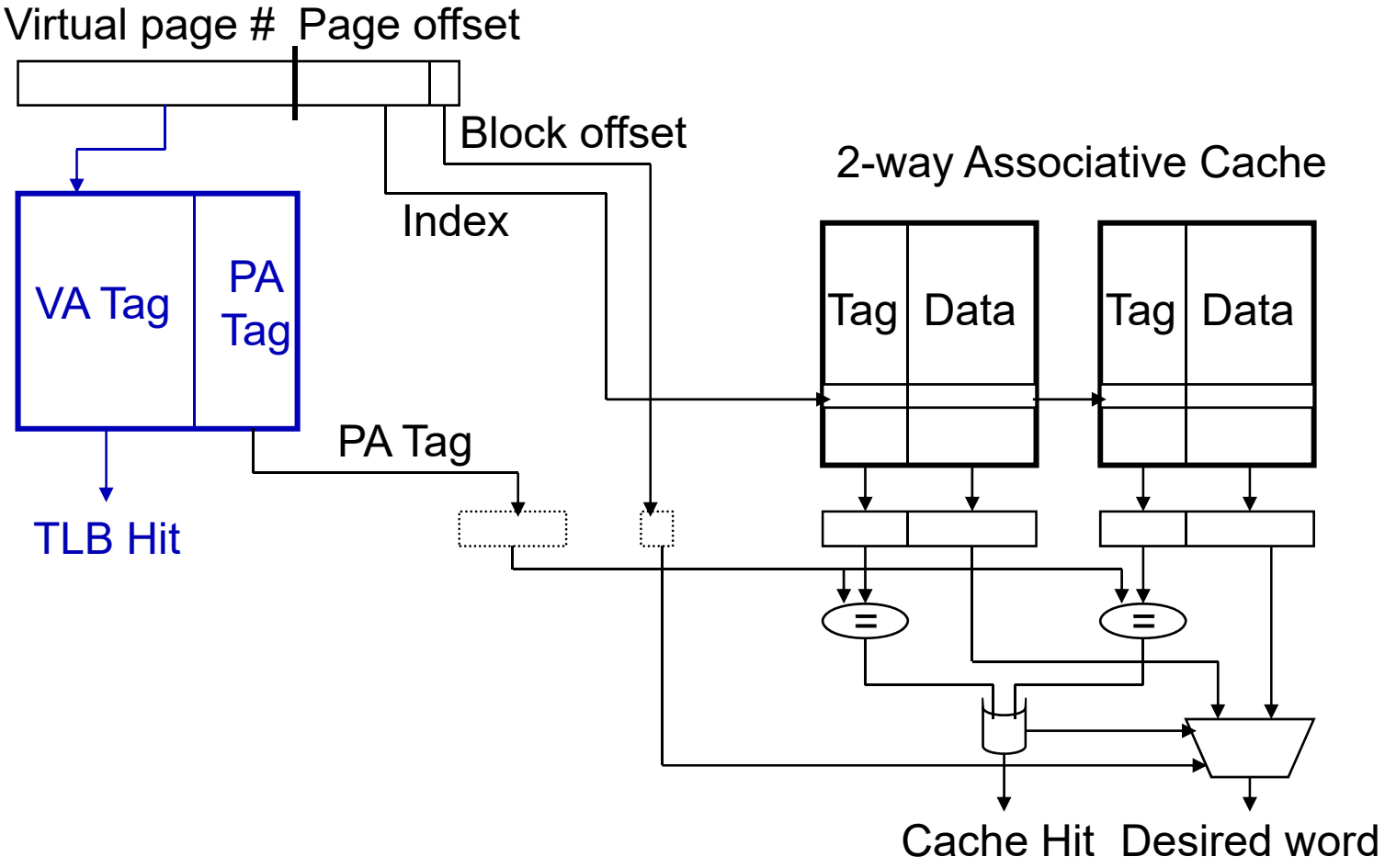
TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	
Hit	Hit	Miss	
Miss	Hit	Hit	
Miss	Hit	Miss	
Miss	Miss	Miss	
Hit	Miss	Miss/ Hit	
Miss	Miss	Hit	

TLB Event Combinations

TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	Yes – what we want!
Hit	Hit	Miss	Yes – although the page table is not checked if the TLB hits
Miss	Hit	Hit	Yes – TLB miss, PA in page table
Miss	Hit	Miss	Yes – TLB miss, PA in page table, but data not in cache
Miss	Miss	Miss	Yes – page fault
Hit	Miss	Miss/ Hit	Impossible – TLB translation not possible if page is not present in memory
Miss	Miss	Hit	Impossible – data not allowed in cache if page is not in memory

Reducing Translation Time

- Can overlap the cache access with the TLB access
  - Works when the high order bits of the VA are used to access the TLB while the low order bits are used as index into cache



The HW/SW Boundary

- What parts of the virtual to physical address translation is done by or assisted by the hardware?
  - Translation Lookaside Buffer (TLB) that caches the recent translations
    - TLB access time is part of the cache hit time
    - May allot an extra stage in the pipeline for TLB access
  - Page table storage, fault detection and updating
    - Page faults result in interrupts (precise) that are then handled by the OS
    - Hardware must support (i.e., update appropriately) Dirty and Reference bits (e.g., ~LRU) in the Page Tables
  - Disk placement
    - Bootstrap (e.g., out of disk sector 0) so the system can service a limited number of page faults before the OS is even loaded