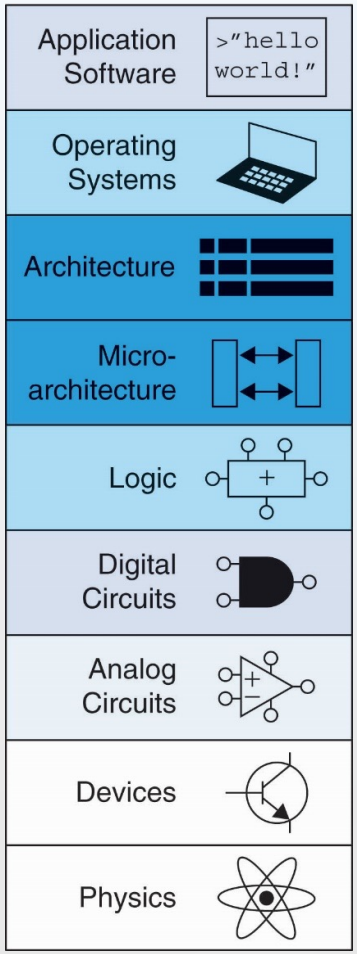
TEMA 2
JERARQUÍA DE MEMORIA

Transparencias basadas en:

- S. Harris and D. Harris, “Digital Design and Computer Architecture: RISC-V Edition”, Morgan Kaufmann

Memory System :: Topics

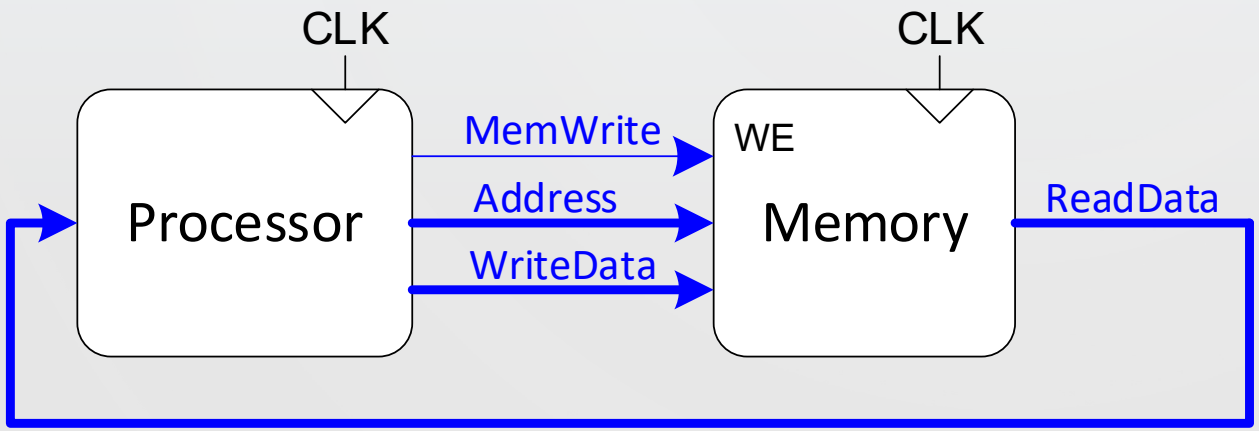
- **Introduction**
- **Memory System Performance Analysis**
- **Caches**
  - **Direct Mapped vs Associative Caches, Spatial Locality, Replacement Policy**
- **Main Memory**
- **Virtual Memory**
  - **Address Translation, Page Table, TLB**



Introduction

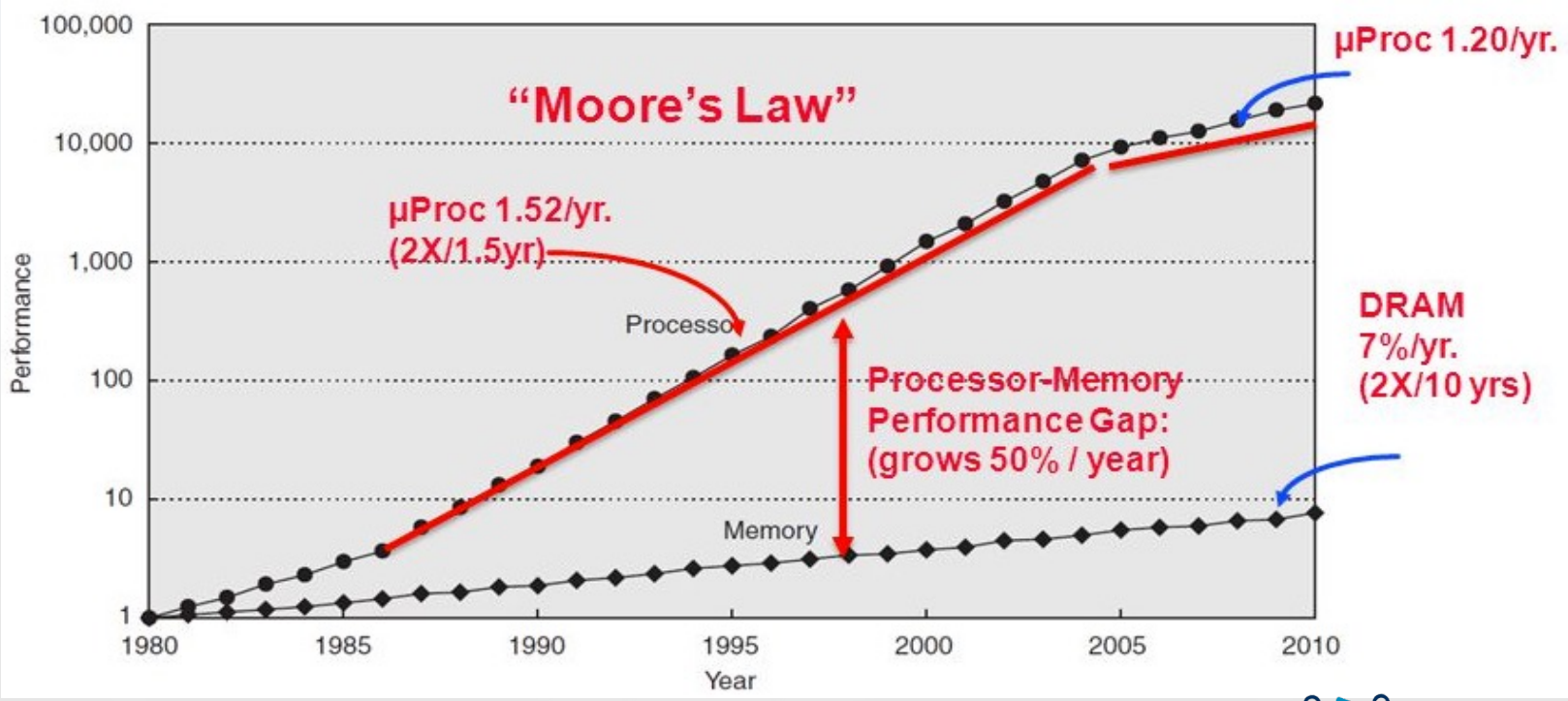
- **Computer performance depends on:**
  - **Processor** performance
  - **Memory system** performance

Processor / Memory Interface:



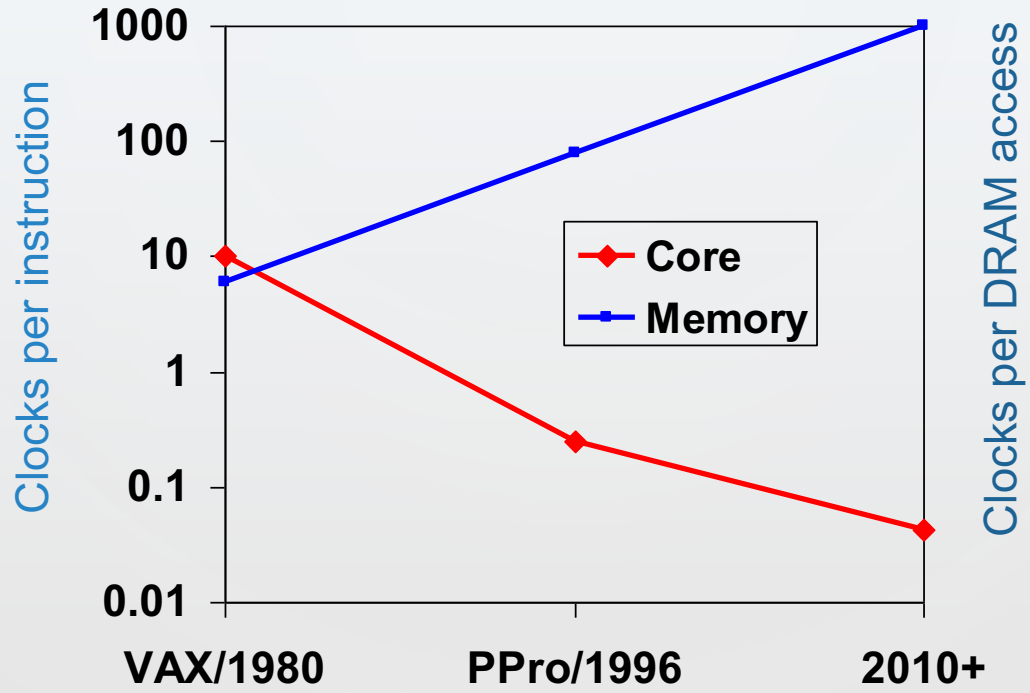
Processor-Memory Gap

- In prior chapters, assumed access memory in 1 clock cycle – but hasn't been true since the 1980's.



THE “MEMORY WALL”

- Processor vs DRAM speed disparity continues to grow



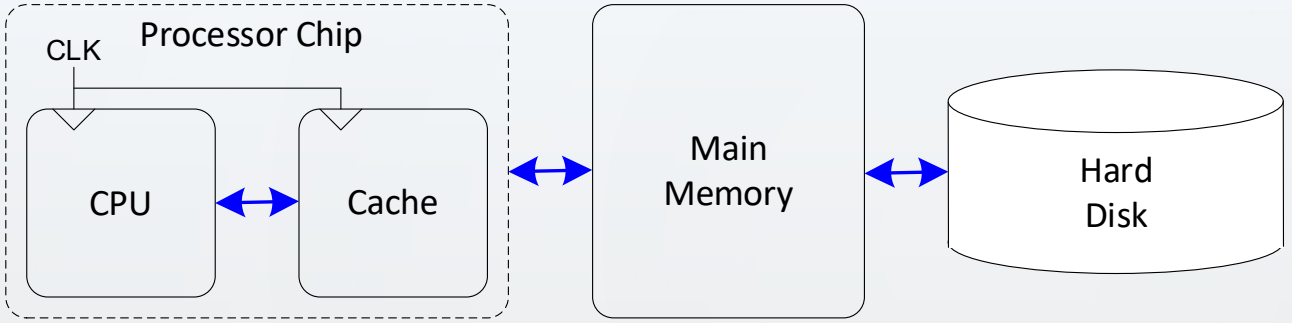
- Good memory hierarchy (cache) design is increasingly important to overall performance

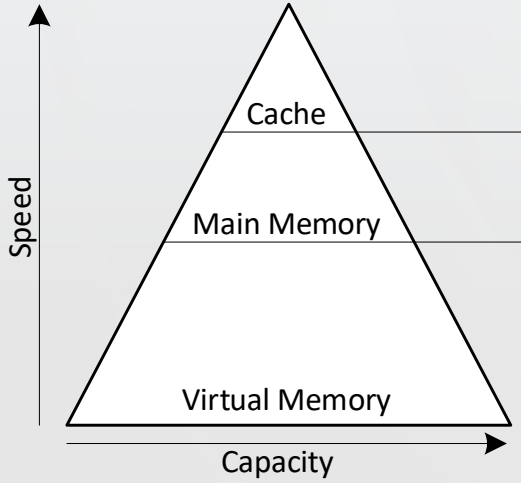
Memory System Challenge

- Make memory system appear as fast as processor
- Use hierarchy of memories
- Ideal memory:
  - **Fast**
  - **Cheap** (inexpensive)
  - **Large** (capacity)

- **But can only choose two!**

Memory Hierarchy





Technology	Price / GB	Access Time (ns)	Bandwidth (GB/s)
SRAM	$100	0.2 - 3	100+
DRAM	$3	10 - 50	30
SSD	$0.10	20,000	0.05 - 3
HDD	$0.03	5,000,000	0.001 - 0.1

MEMORY HIERARCHY TECHNOLOGIES

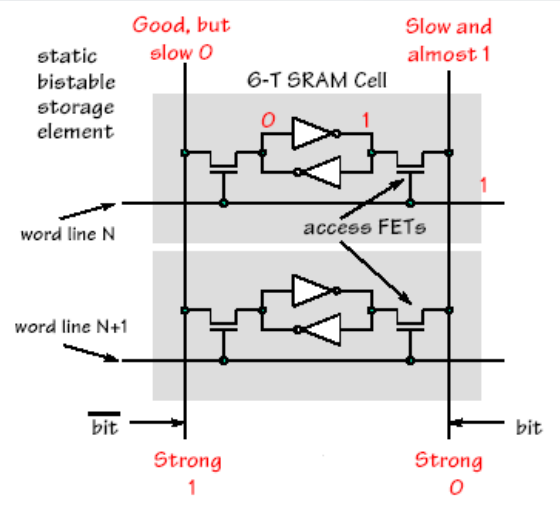
- Caches use *SRAM* for speed and technology compatibility
  - Fast (typical access times of 0.5 to 2.5 nsec)
  - Low density (6 transistor cells), higher power, expensive
  - Static: content will last "forever" (as long as power is left on)

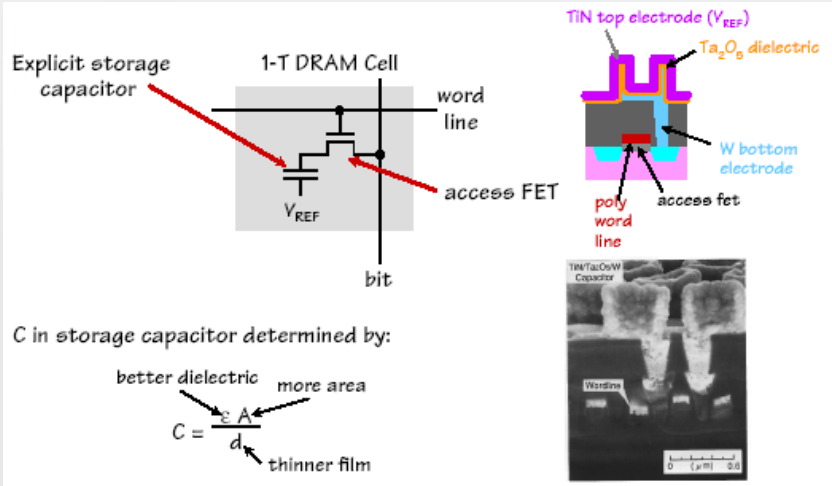
- Main memory uses *DRAM* for size (density)
  - Slower (typical access times of 50 to 70 nsec)
  - High density (1 transistor cells), lower power, cheaper
  - Dynamic: needs to be "refreshed" regularly (~ every 8 ms)
    - consumes1% to 2% of the active cycles of the DRAM

- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of DRAM

MEMORY HIERARCHY TECHNOLOGIES

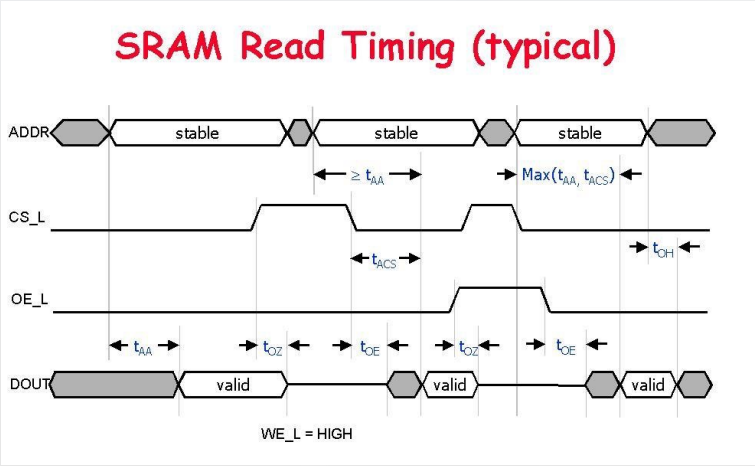
- SRAM cell
  - FF structure
  - Not refreshing
  - 6T per cell (low density, higher power)
  - Expensive (higher cost per bit)
  - **Faster (0.5 to 2.5 nsec)**
- DRAM cell
  - Explicit storage capacitor
  - It needs refreshing (destructive reading)
  - **1T per cell (high density, lower power)**
  - Cheaper (lower cost per bit)
  - Slower (50 to 70 nsec)

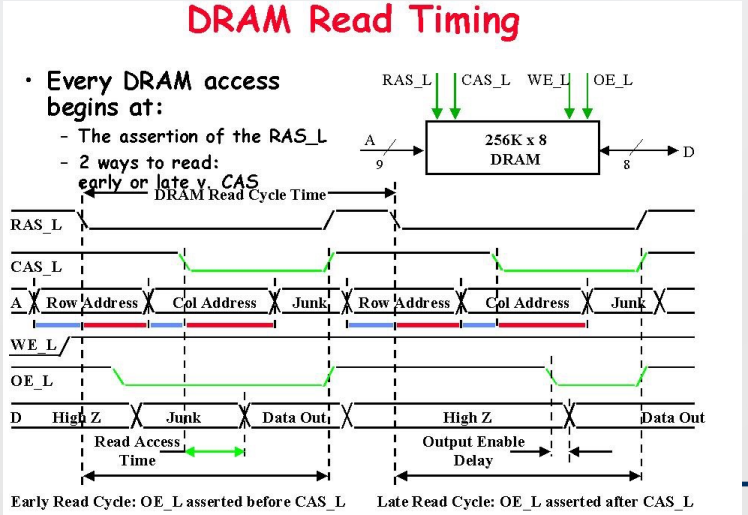




DRAM VS. SRAM

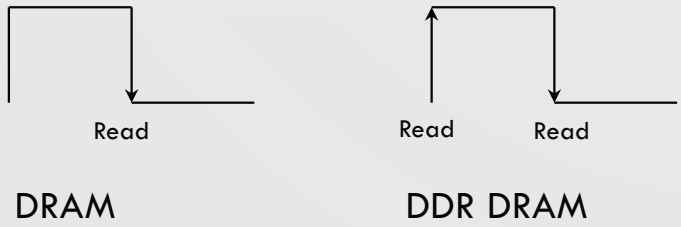
- DRAM read timing slower that SRAM read timing:
  - Dynamic: needs to be "refreshed" regularly (~ every 8 ms)
    - consumes1% to 2% of the active cycles of the DRAM
  - Addresses divided into 2 halves (row and column)
    - *RAS* or *Row Access Strobe* triggering the row decoder
    - *CAS* or *Column Access Strobe* triggering the column selector





ADVANCED DRAM ORGANIZATION

- Bits in a DRAM are organized as a rectangular array
  - 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
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- Double data rate (DDR) DRAM (DDR4)
  - Transfer on rising and falling clock edges



Locality

Exploit locality to make memory accesses fast:

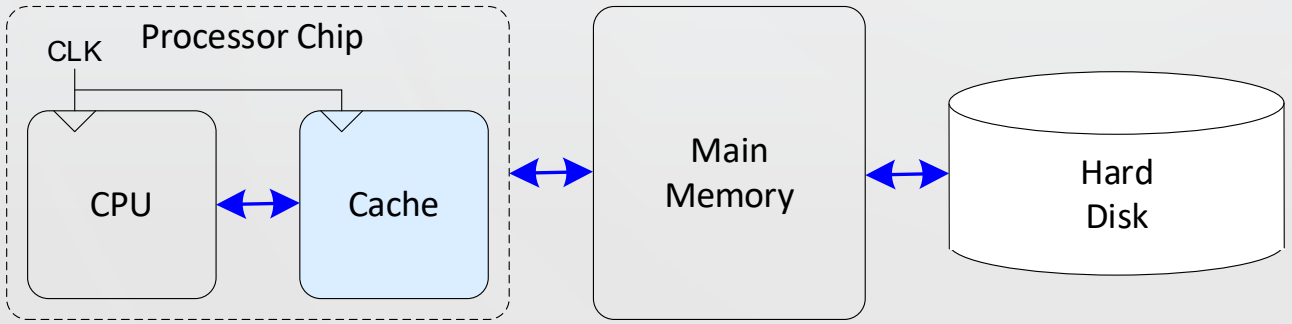
- **Temporal Locality:**
  - Locality in time
  - If data used recently, likely to use it again soon
  - **How to exploit:** keep recently accessed data in higher levels of memory hierarchy
- **Spatial Locality:**
  - Locality in space
  - If data used recently, likely to use nearby data soon
  - **How to exploit:** when access data, bring nearby data into higher levels of memory hierarchy too

Actual processors: 96.8% of memory accesses find the data in the cache

CACHE

Cache

- Highest level in memory hierarchy
- Fast (typically ~ 1 cycle access time)
- Ideally supplies most data to processor
- Usually holds most recently accessed data



Cache Design Questions

- What data is held in the cache?
- How is data found?
- What data is replaced?

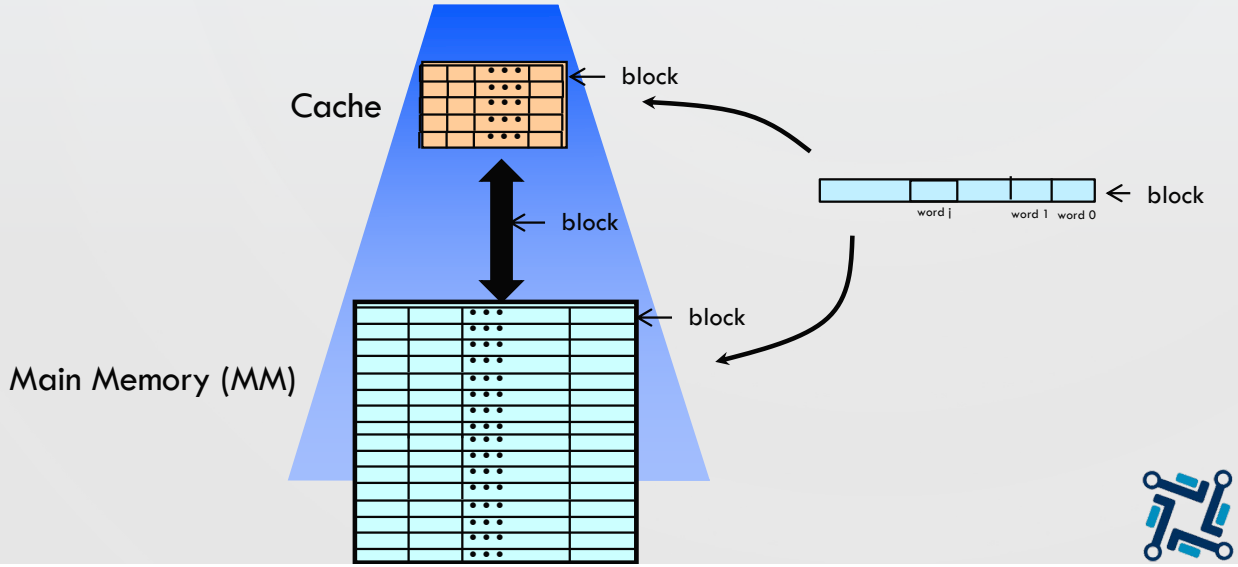
We focus on data loads, but stores follow the same principles.

What data is held in the cache?

- Ideally, cache anticipates needed data and puts it in cache
- But impossible to predict future
- Use past to predict future – temporal and spatial locality:
  - **Temporal locality:** copy newly accessed data into cache
  - **Spatial locality:** copy neighboring data into cache too

CACHE – MAIN MEMORY SYSTEM OPERATION

- Both memories a divided in BLOCKS
  - Each block has several words
- When the processor requires one word, it is fetched in the Cache
  - If the word is there (HIT), it is taken and sent to the processor (it takes 1 clock cycle (cc)).
  - If it is not there (MISS), it is searched in the Main Memory (several cc)
- In a miss, the full block of main memory containing the word is copied from the main memory to the cache



THE MEMORY HIERARCHY: TERMINOLOGY

- Block (or line): the minimum unit of information that is present (or not) in a cache
- Hit Rate (P_{hit}): the fraction of memory accesses found in a level of the memory hierarchy
  - $P_{hit} = {Number\ of\ hits}/{Number\ of\ memory\ references}$
  - Hit Time (T_{hit}): Time to access that level which consists of

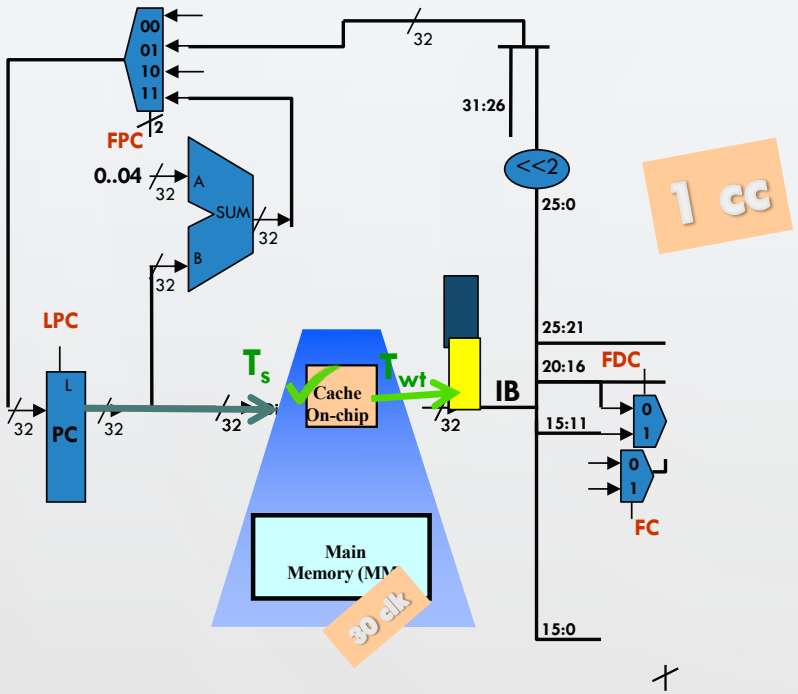
    Time to search the block (hit/miss) (T_s) + Time to transfer the word (T_{wt})

- Miss Rate (P_{miss}): the fraction of memory accesses *not* found in a level of the memory hierarchy $\Rightarrow$ $P_{miss} = 1 - P_{hit}$
  - $P_{miss} = {Number\ of\ misses}/{Number\ of\ memory\ references}$
  - Miss Penalty (TP_{miss}): 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 (T_{acc}) + Time to transmit that block to the level that experienced the miss (T_{blk})

Hit Time << Miss Penalty

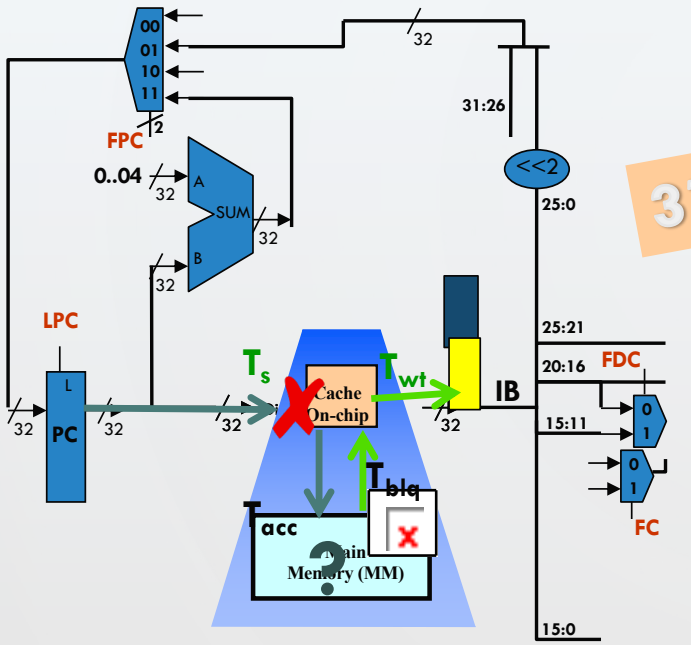
HIT: 1 CC



T_{hit}= Hit time
T_s= Search time
T_{wt}=Word transfer time

$T_{hit} = T_s + T_{wt}$

MISS: 31 CC



31 cc

T_{acc}= MM access time

T_{blq}= Block transfer time

T_{miss}= Miss time

$$T_{miss} = T_s + T_{acc} + T_{blq} + T_{wt}$$

TP_{miss}= Miss penalty (= T_{hit}-T_{miss})

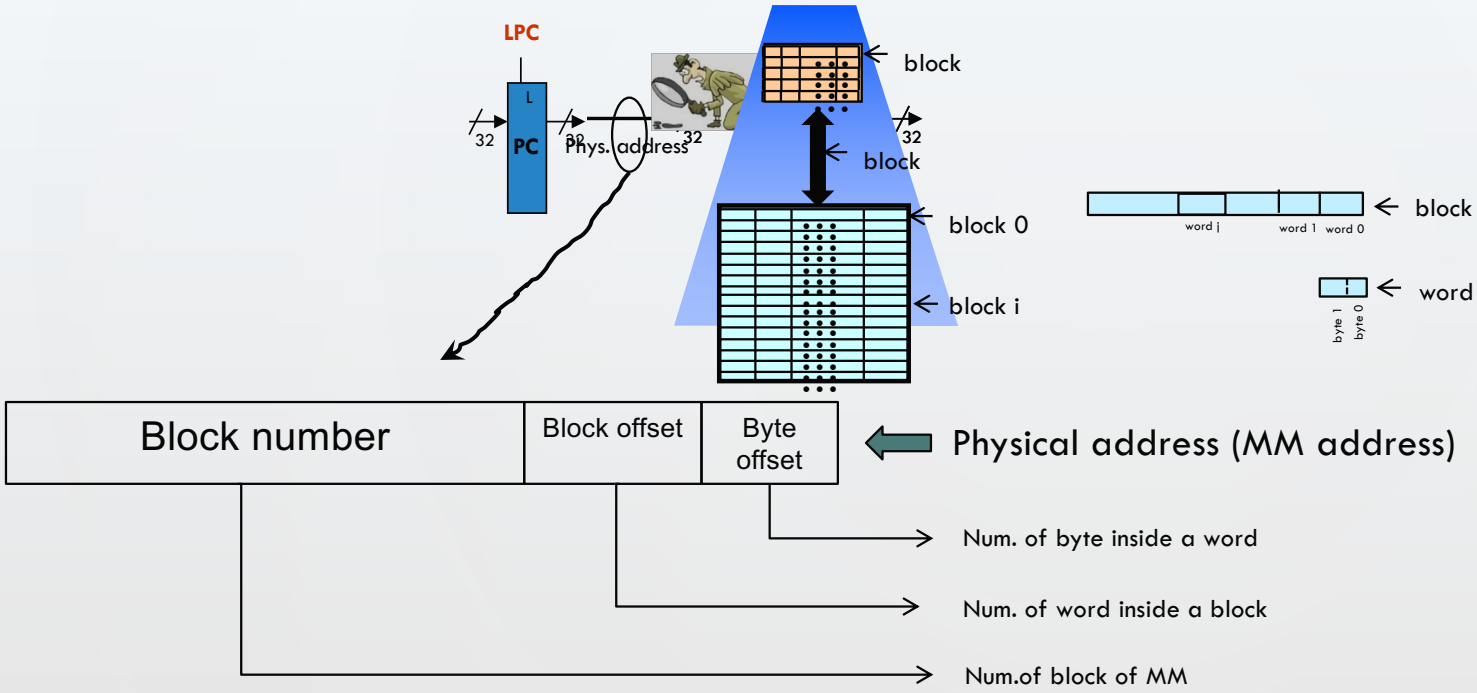
$$TP_{miss} = T_{acc} + T_{blq} = 30 \text{ CC}$$

$$T_{miss} = T_{hit} + TP_{miss}$$

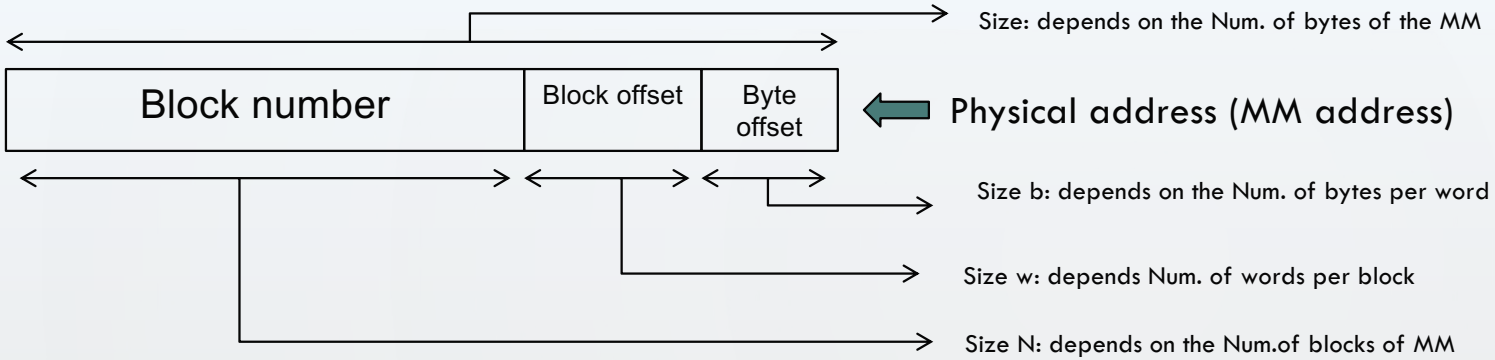
AVERAGE ACCESS TIME

- Hit time is also important for performance
- Average memory access time (AMAT)
  - T_{AMAT} = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I$ache miss rate = 5%
  - T_{AMAT} = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

Address Subdivision

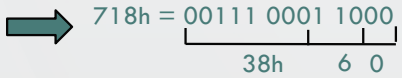


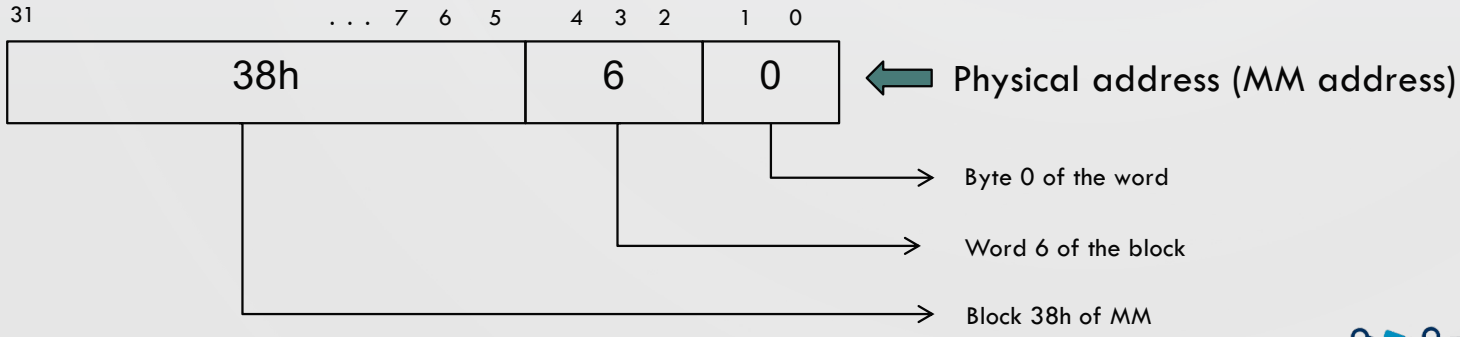
Address Subdivision (Byte- addressable)



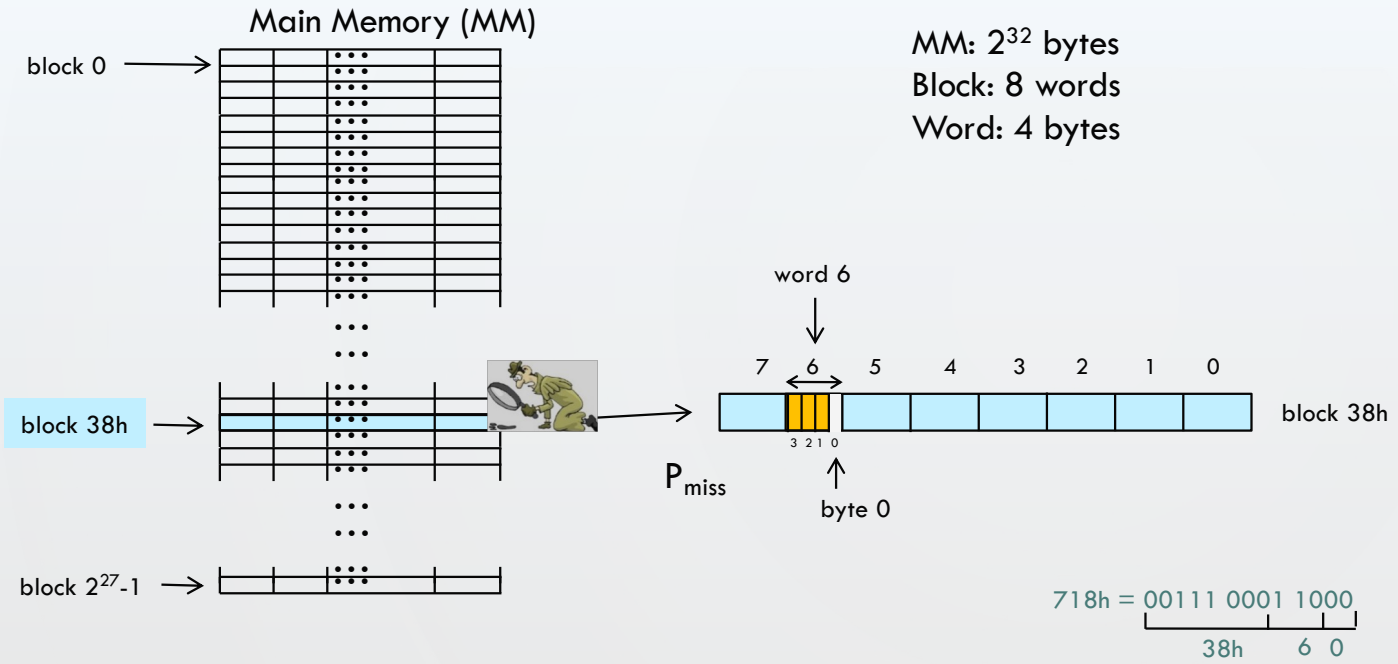
Example byte-addressable: MM of 2^{32} bytes, Words of 4 bytes (2 bits, bit 1,0), blocks of 8 words (3 bits, bits 4,3,2)

Physical address: 718h

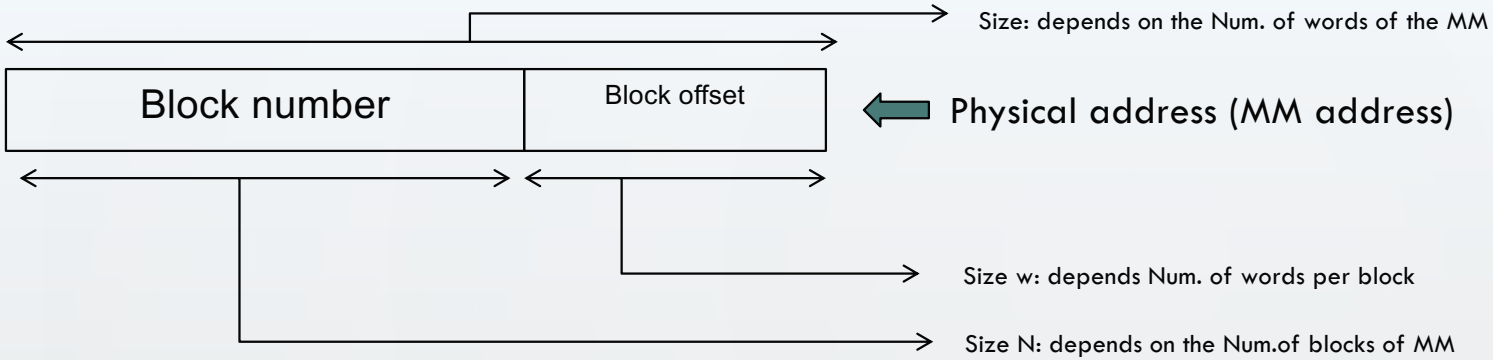




Address Subdivision (Byte- addressable)



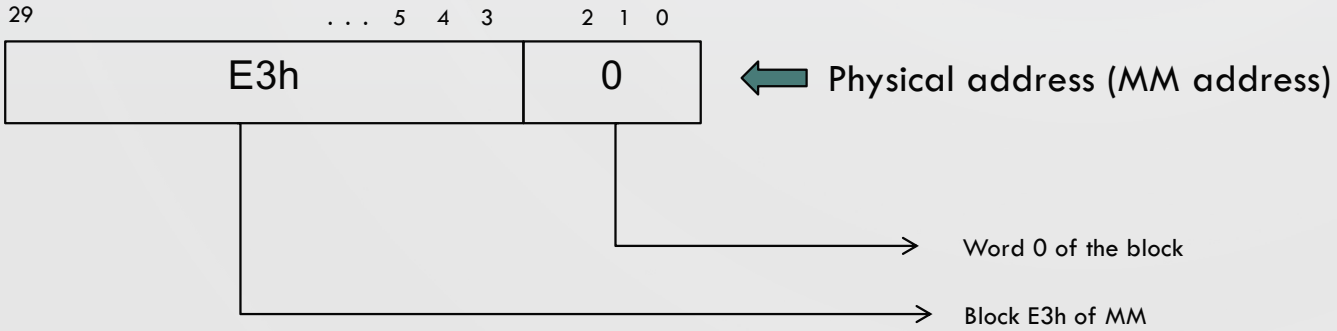
Address Subdivision (Word- addressable)



Example Word addressable: MM of 2^{32} bytes, Words of 4 bytes, blocks of 8 words (3 bits, bits 2,1,0)

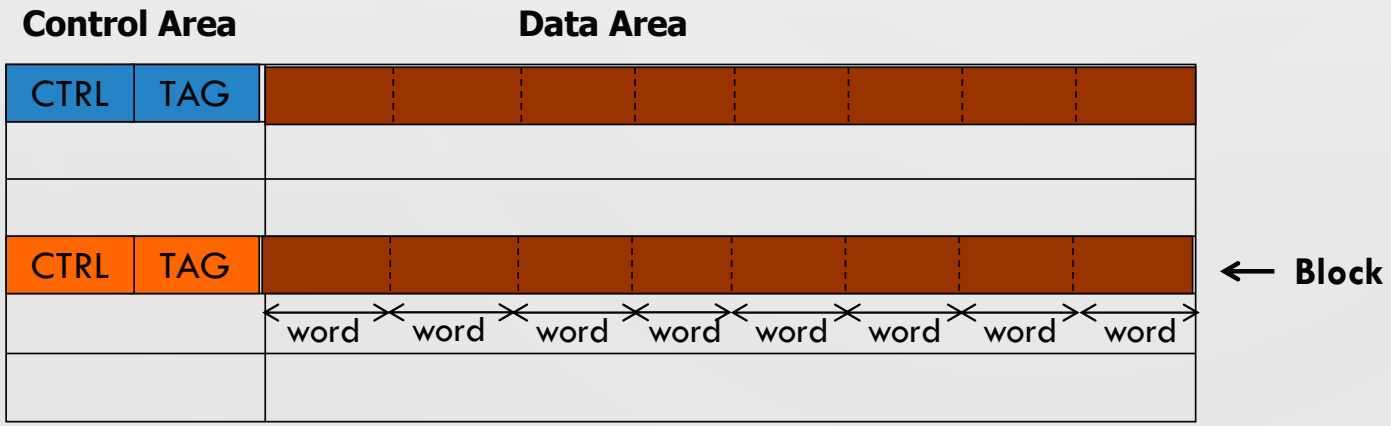
Physical address: 718h ➡ 718h = 00111 0001 1000

E3h 0

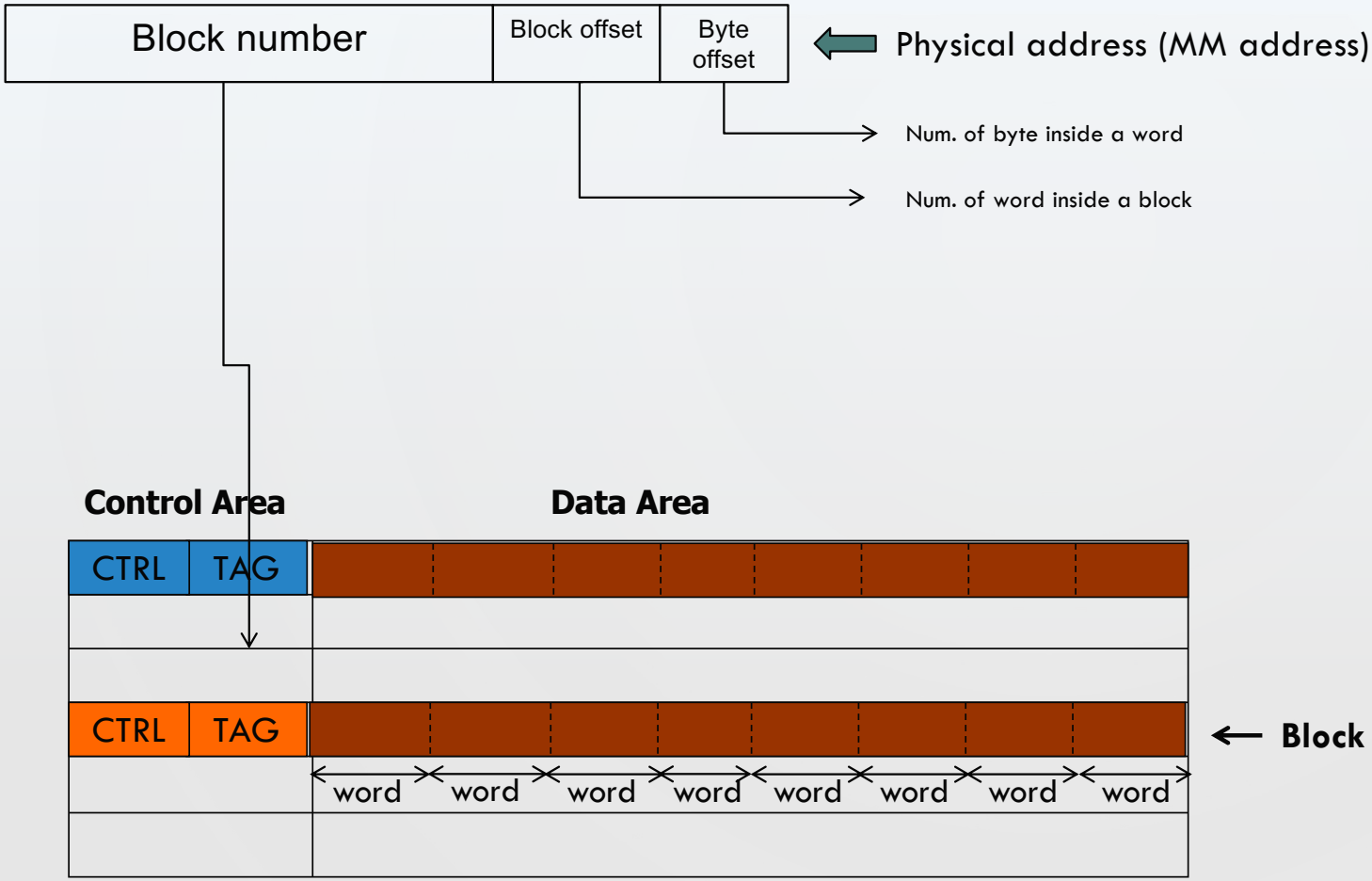


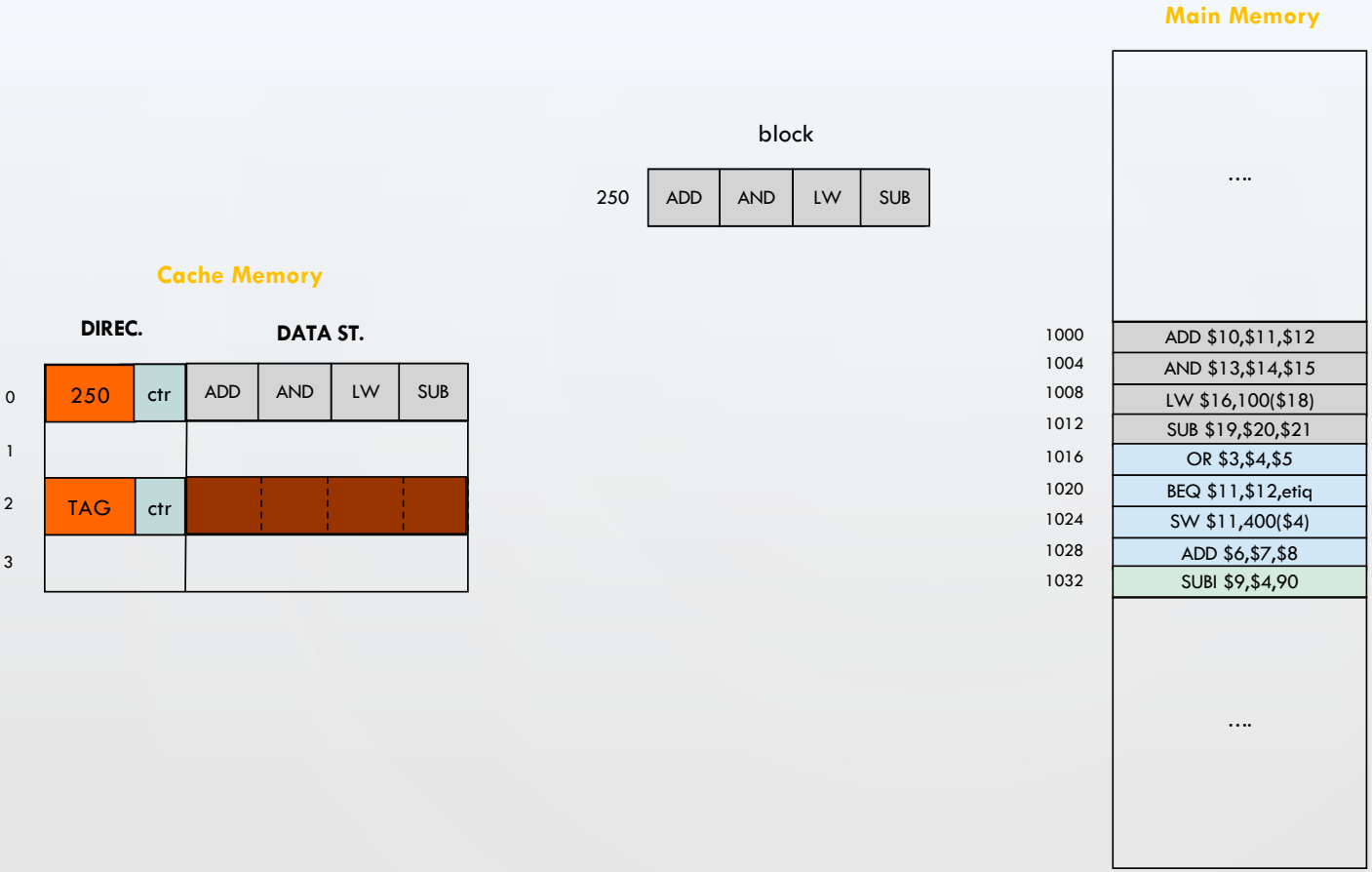
Cache organization

- **Internal cache architecture:**
  - Two parts:
    - **Data area (storage):** Contains copies of some blocks of MM
    - **Control area (directory):** auxiliar information about the current blocks in the cache (i.e., blocks of MM which are currently copied in the CM)
      - Tags (TAG) and control bits (CTRL).



Cache organization



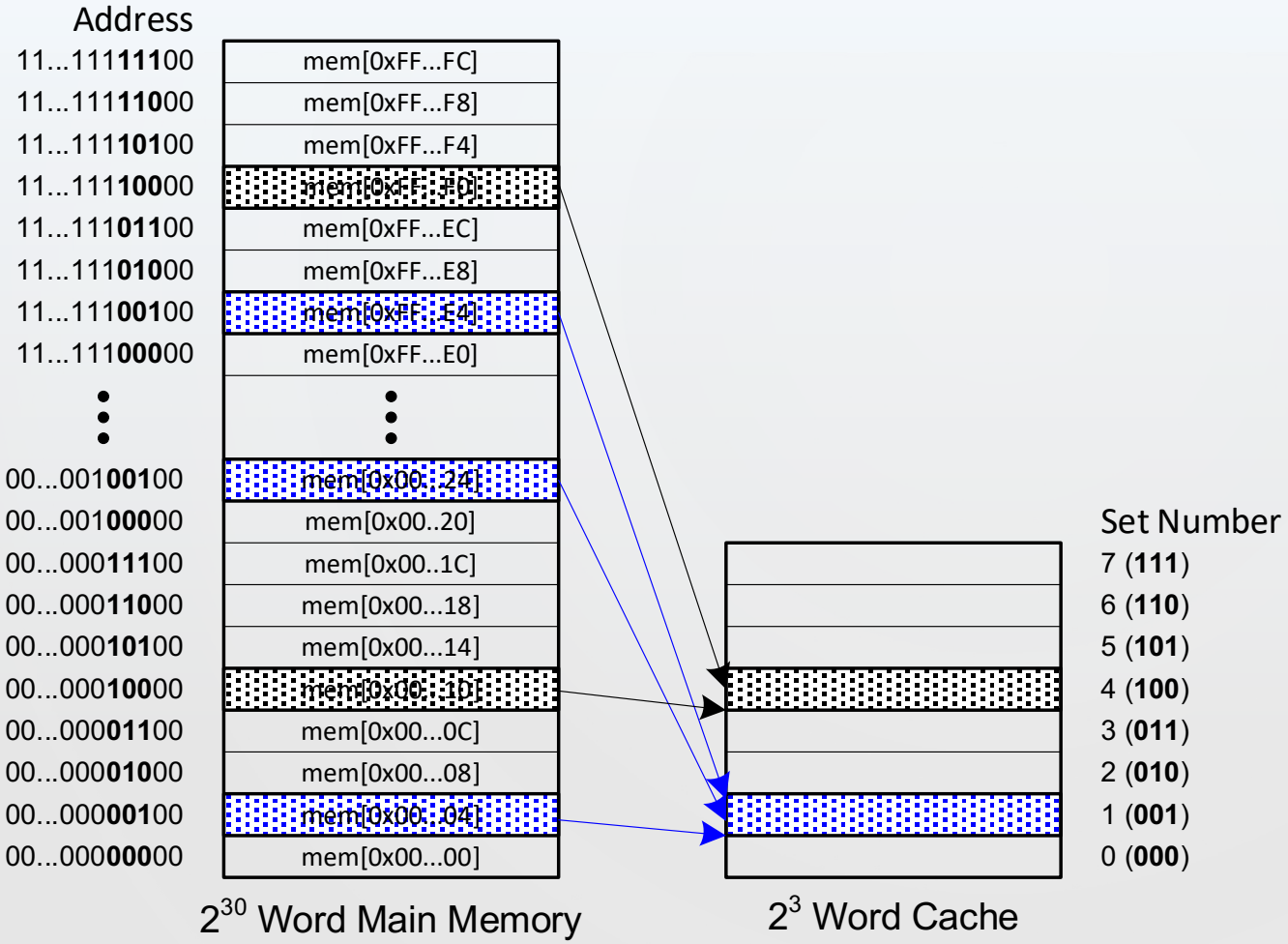

Main Memory
....
1000 ADD $10,$11,$12
1004 AND $13,$14,$15
1008 LW $16,100($18)
1012 SUB $19,$20,$21
1016 OR $3,$4,$5
1020 BEQ $11,$12,etiq
1024 SW $11,400($4)
1028 ADD $6,$7,$8
1032 SUBI $9,$4,90
....
block
250 ADD AND LW SUB
Cache Memory
DIREC.
DATA ST.
0 250 ctr ADD AND LW SUB
1
2 TAG ctr
3

How is data found?

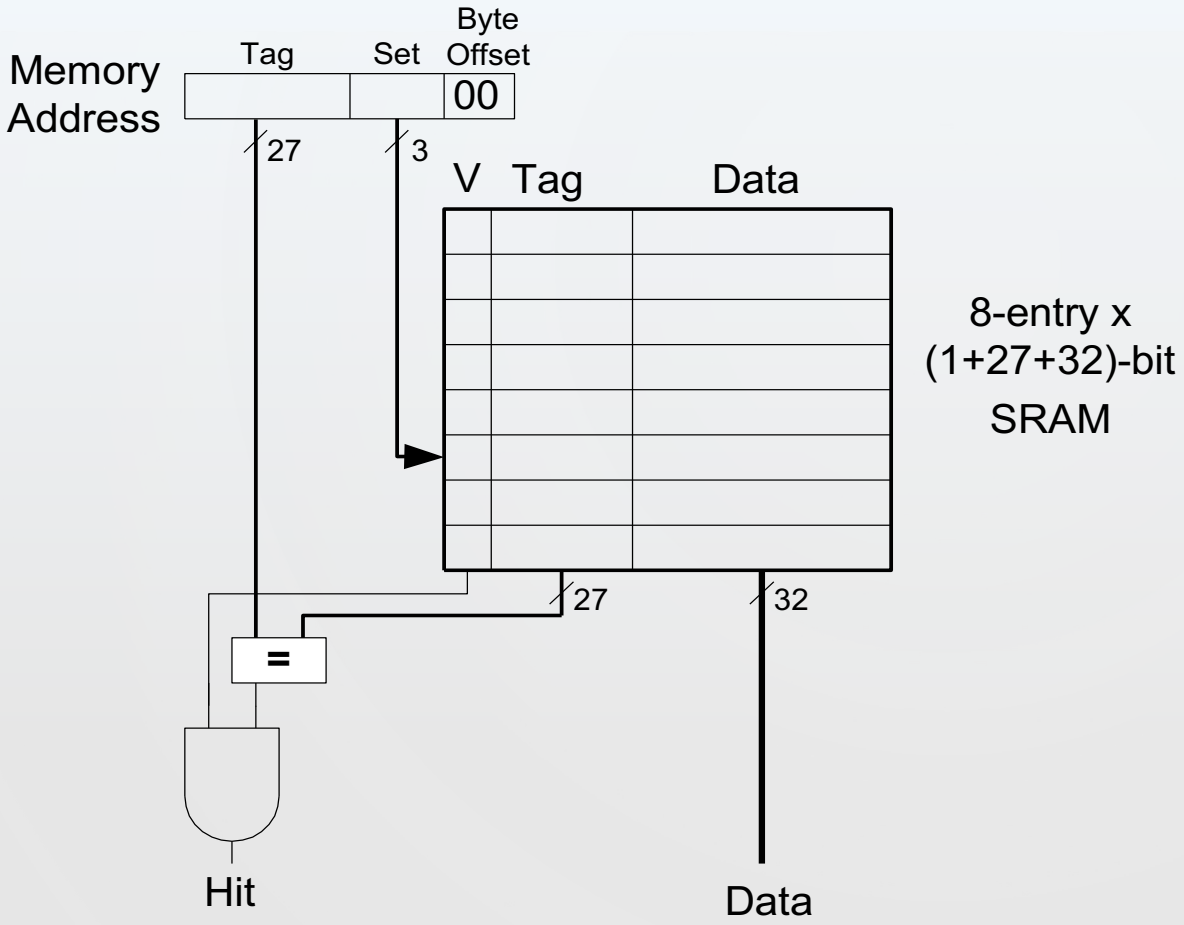
- Cache organized into *S* sets
- Each memory address maps to exactly one set
- Caches categorized by # of blocks in a set:
  - **Direct mapped:** 1 block per set
  - ***N*-way set associative:** *N* blocks per set
  - **Fully associative:** all cache blocks in 1 set
- Examine each organization for a cache with:
  - Capacity (***C*** = 8 words)
  - Block size (***b*** = 1 word)
  - So, number of blocks (***B*** = 8)

Direct Mapped Cache

Posicion Memoria Cache= mod(N, Tamaño MC)

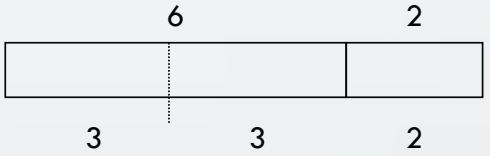


Direct Mapped Cache Hardware



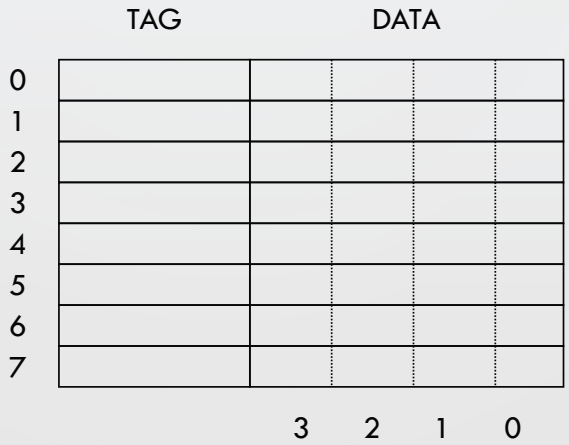
Direct Mapping example

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



- Source code:

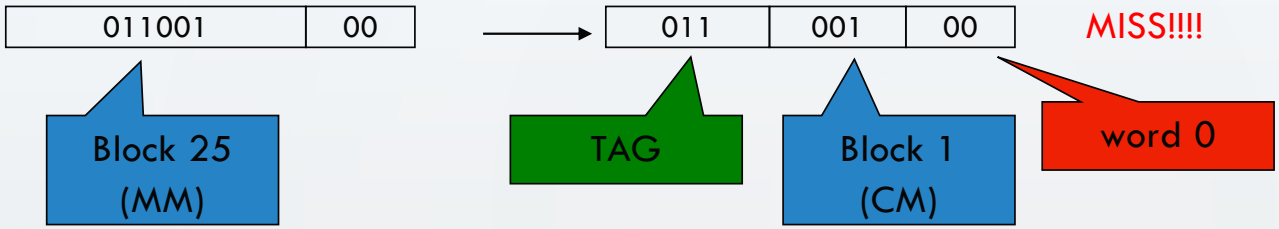
```
LOOP
        100
        17
        79
        50
END LOOP
```

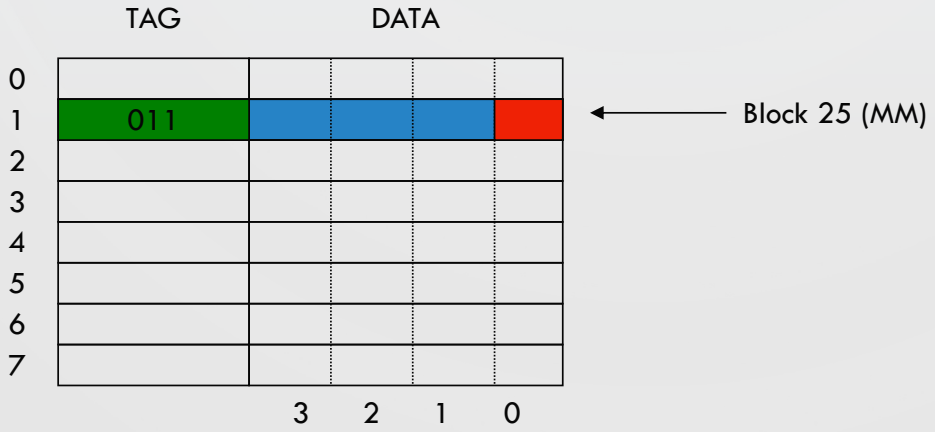


DIRECT-MAPPING

Iteration #1:

- $100_{(10}$:





DIRECT-MAPPING

Iteration #1:

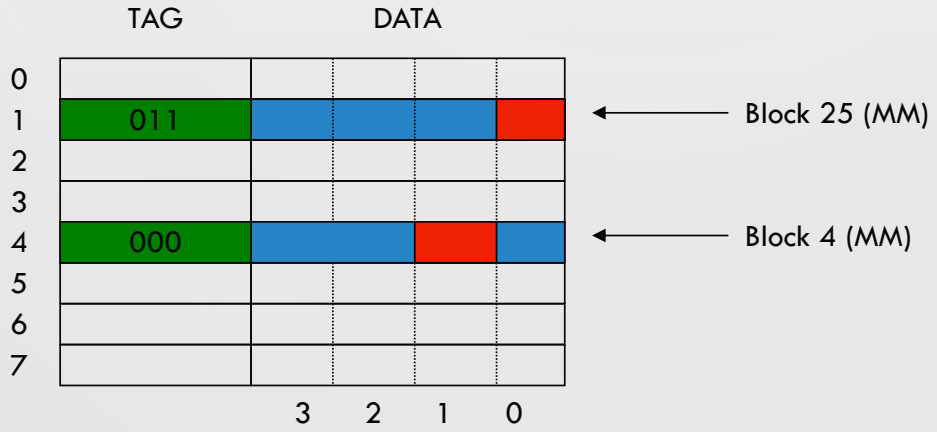
- $100_{(10}$: 011001 | 00
- $17_{(10}$: 000100 | 01 → 000 | 100 | 01 MISS!!!!

Block 4 (MM)

TAG

Block 4 (CM)

word1



DIRECT-MAPPING

Iteration #1:

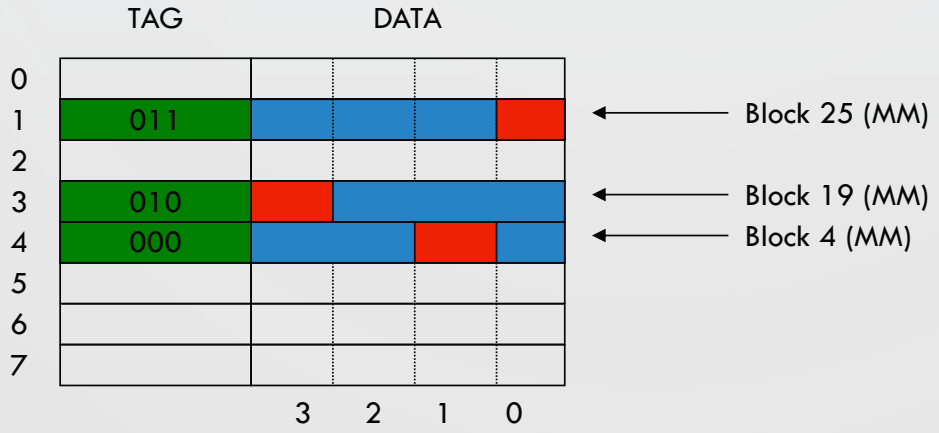
- $100_{(10}$: 011001 | 00
- $17_{(10}$: 000100 | 01
- $79_{(10}$: 010011 | 11 → 010 | 011 | 11 MISS !!!!

Block 19 (MM)

TAG

Block 3 (CM)

Word 3



DIRECT-MAPPING

Iteration #1:

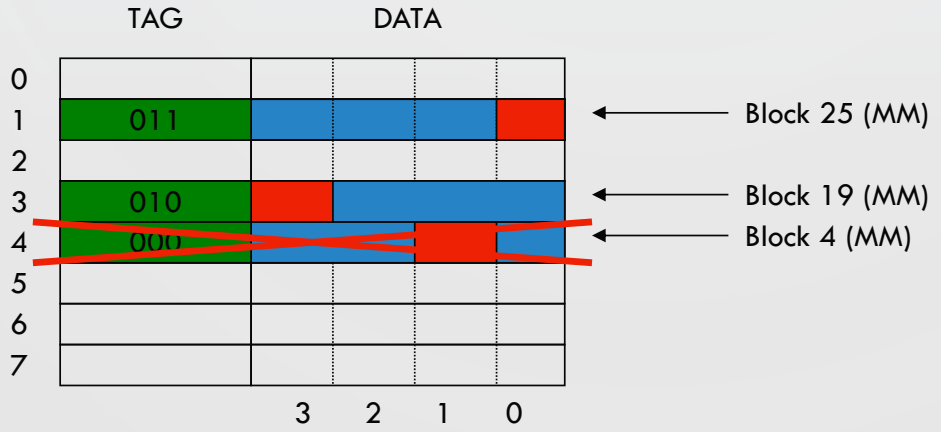
- $100_{(10}$: 011001 | 00
- $17_{(10}$: 000100 | 01
- $79_{(10}$: 010011 | 11
- $50_{(10}$: 001100 | 10 → 001 | 100 | 10 MISS !!!!

Block 12 (MM)

TAG

Block 4 (CM)

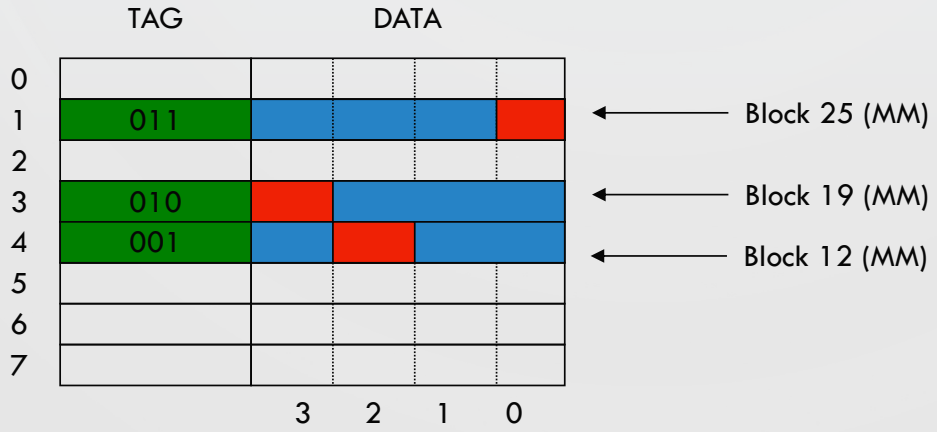
word 2



DIRECT-MAPPING

Iteration #1:

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

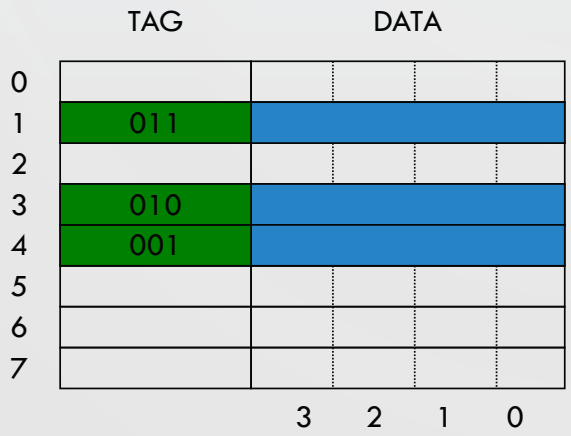


DIRECT-MAPPING

Iteration #2 to …:

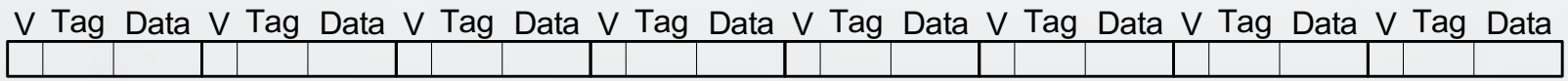
- $100_{(10}$: 011001 | 00 → 011 | 001 | 00 HIT!!!!
- $17_{(10}$: 000100 | 01 → 000 | 100 | 01 MISS!!!!
- $79_{(10}$: 010011 | 11 → 010 | 011 | 11 HIT!!!!
- $50_{(10}$: 001100 | 10 → 001 | 100 | 10 MISS!!!!

Miss Rate ≅ 0.5



Fully Associative Cache

Posicion Memoria Cache --> libre

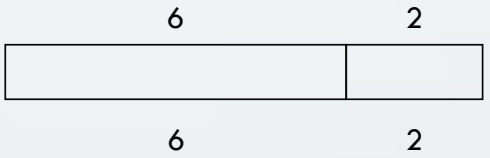


Reduces conflict misses

Expensive to build

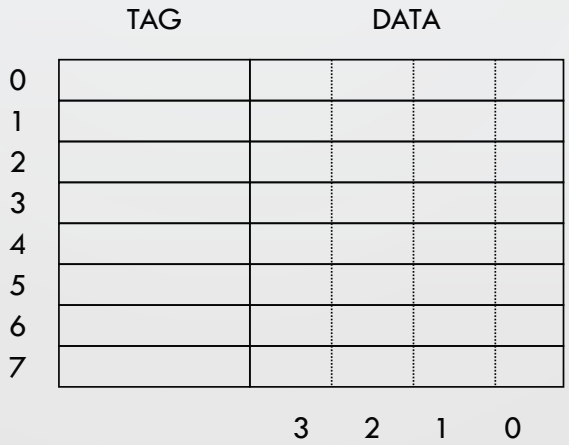
Fully Associative example

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



- Source code:

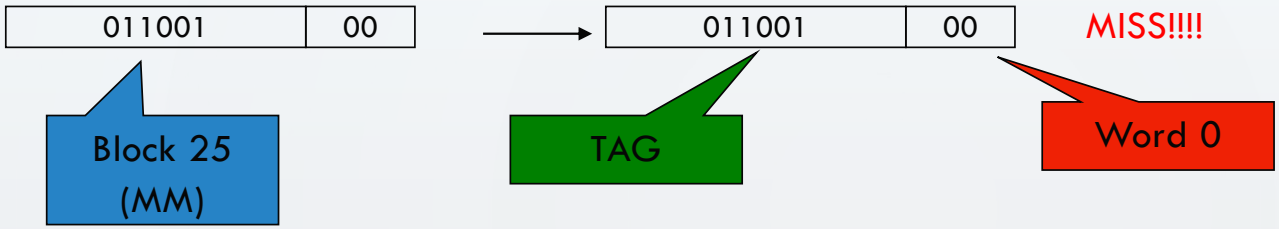
```
LOOP
        100
        17
        79
        50
END LOOP
```

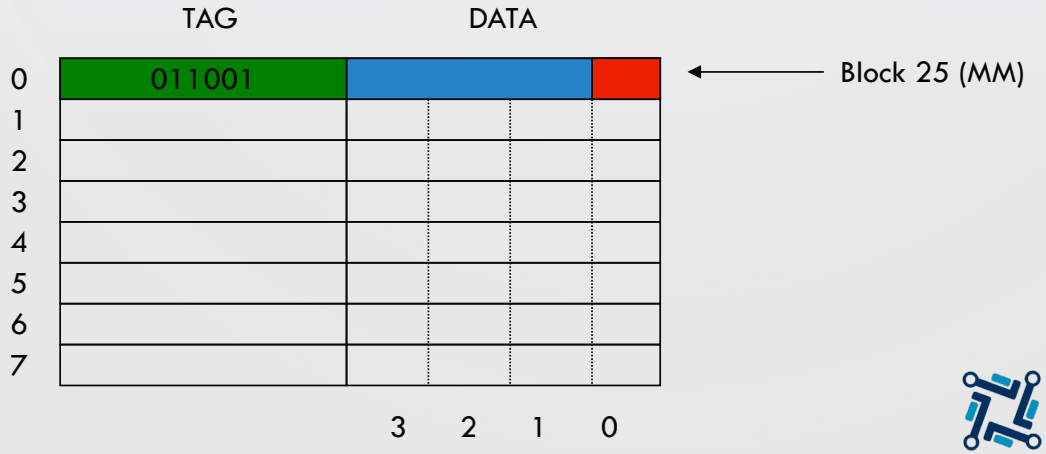


FULLY-ASSOCIATIVE

Iteration #1:

- $100_{(10}$:





FULLY-ASSOCIATIVE

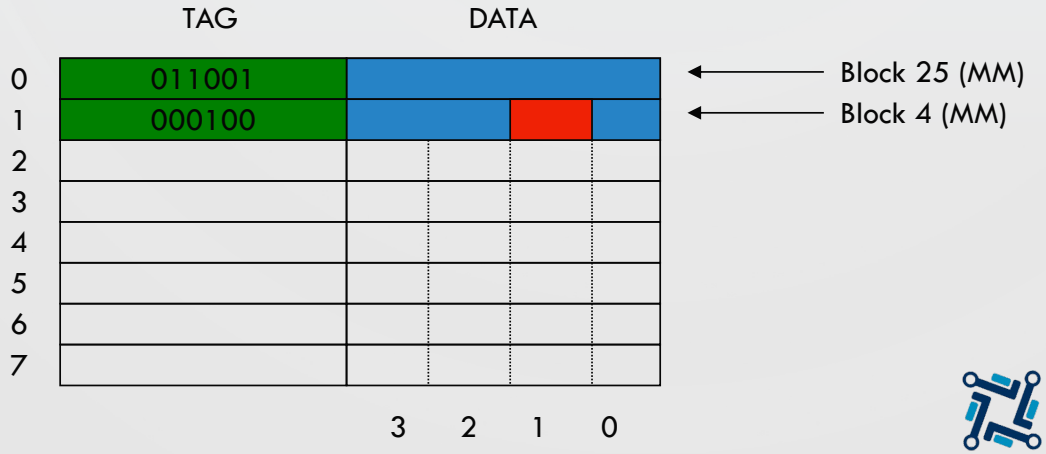
Iteration #1:

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

Block 4 (MM)

TAG

word1



FULLY-ASSOCIATIVE

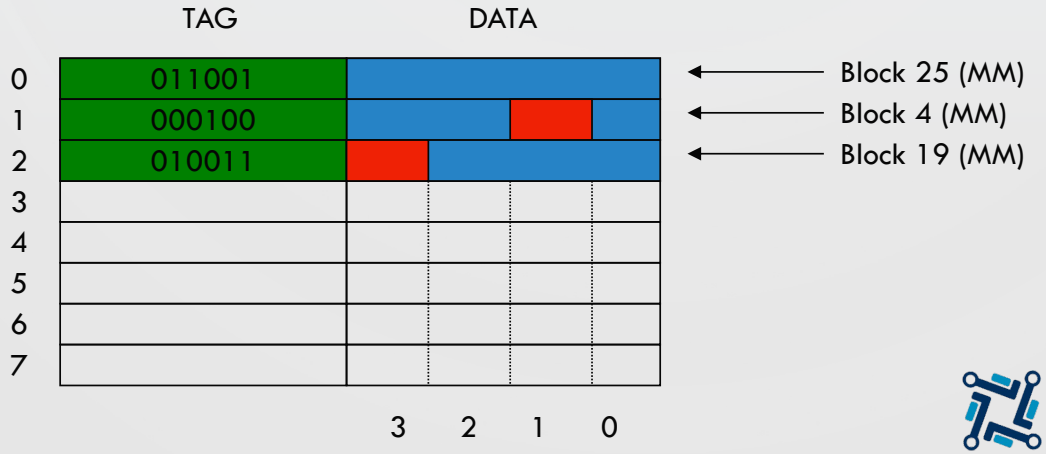
Iteration #1:

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

Block 19 (MM)

TAG

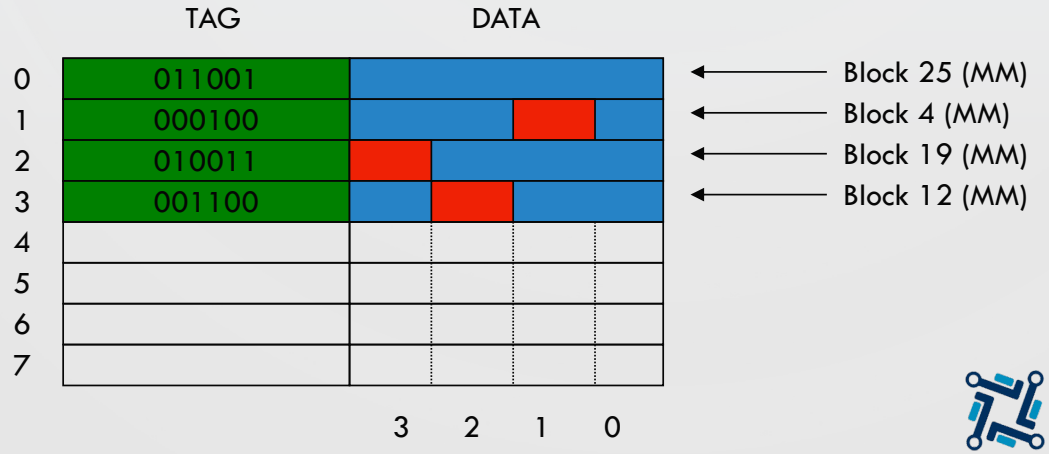
Word 3



FULLY-ASSOCIATIVE

Iteration #1:

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

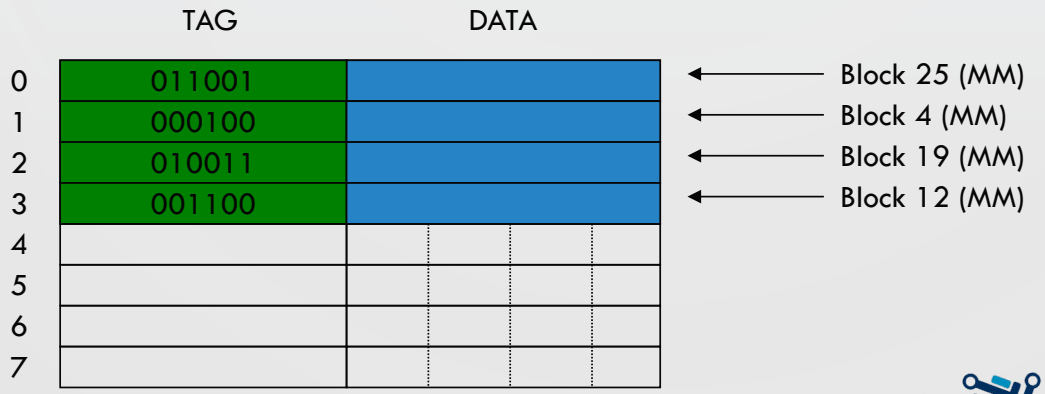


FULLY-ASSOCIATIVE

Iteration #2 to …:

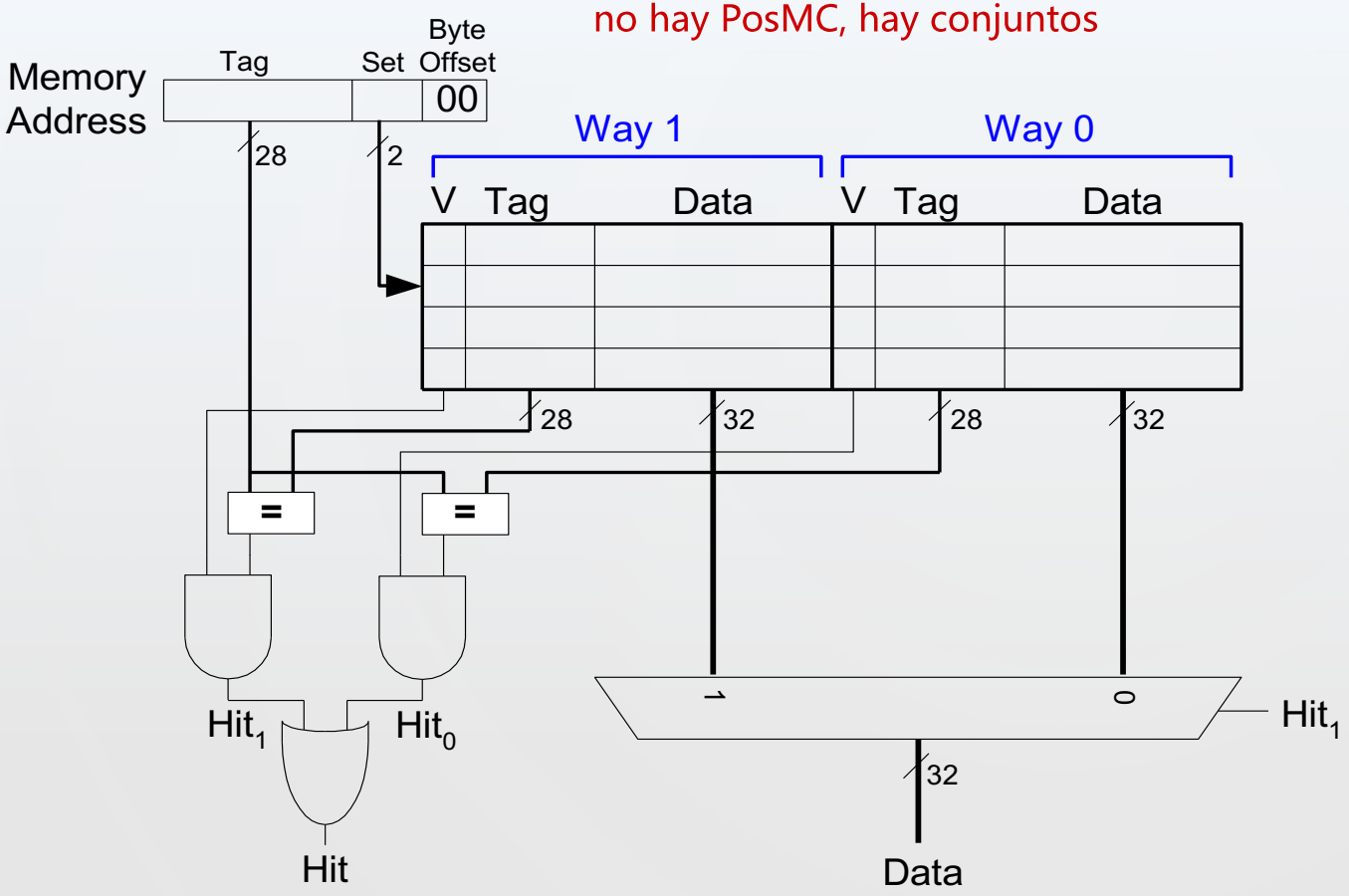
- $100_{(10}$: 011001 | 00 — HIT !!!!
- $17_{(10}$: 000100 | 01 — HIT !!!!
- $79_{(10}$: 010011 | 11 — HIT !!!!
- $50_{(10}$: 001100 | 10 — HIT !!!!

→ Miss Rate ≅ 0



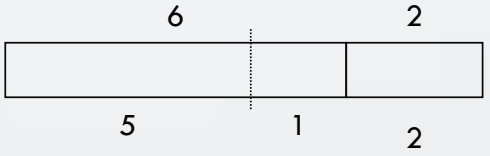
N-Way Set Associative Cache

no hay PosMC, hay conjuntos



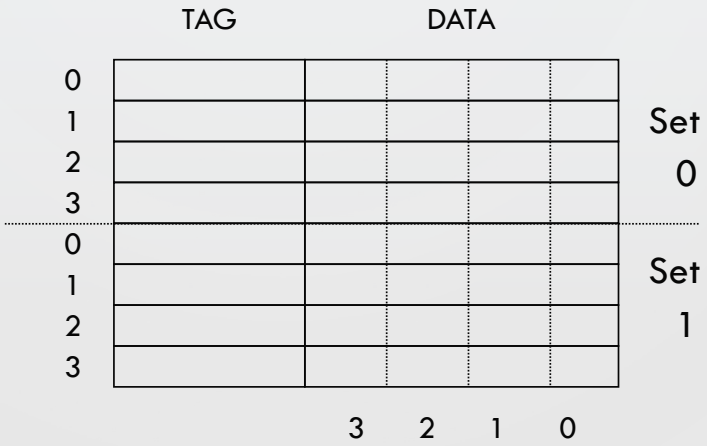
Set Associative example

- 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:

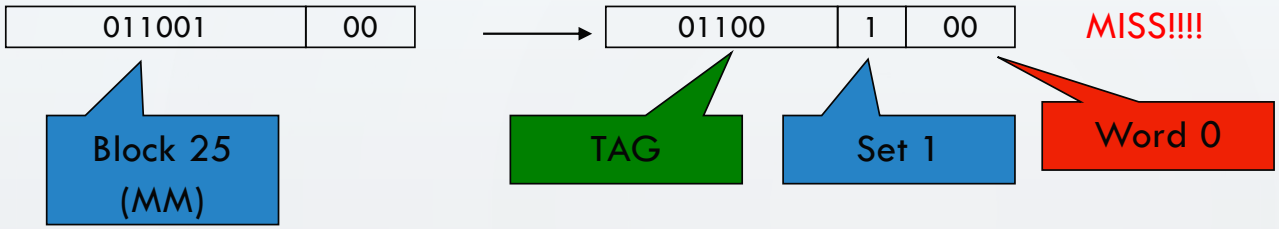
```
LOOP
        100
        17
        79
        50
END LOOP
```

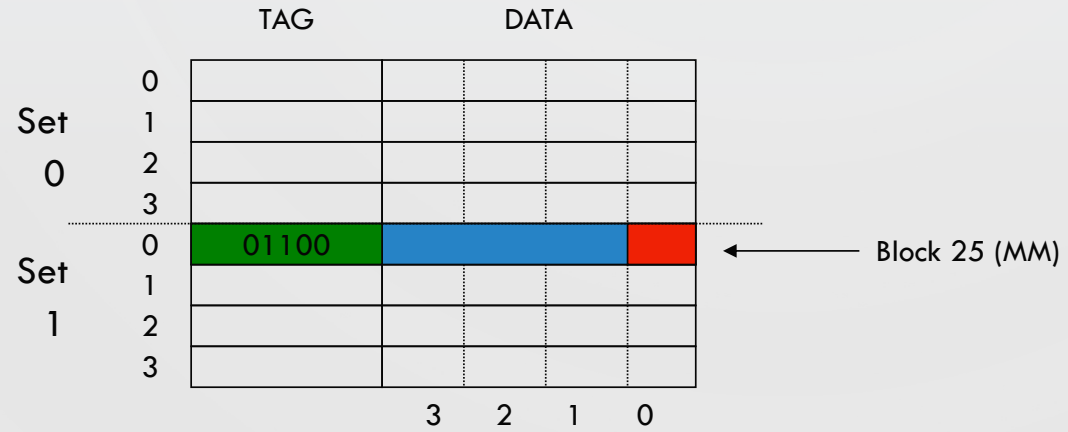


SET-ASSOCIATIVE

Iteration #1:

- $100_{(10}$:

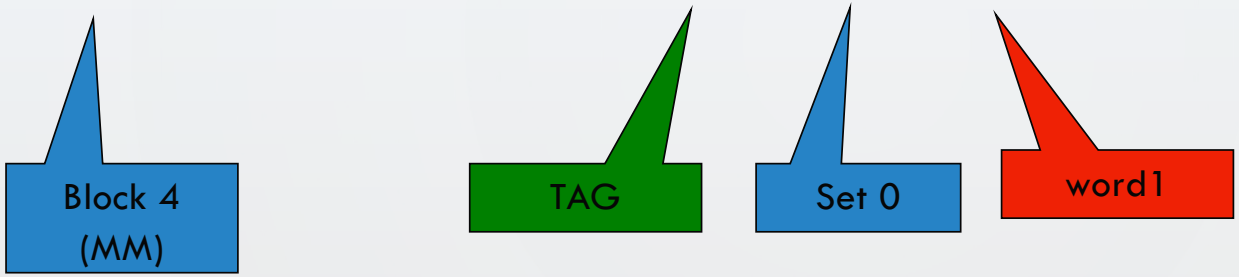


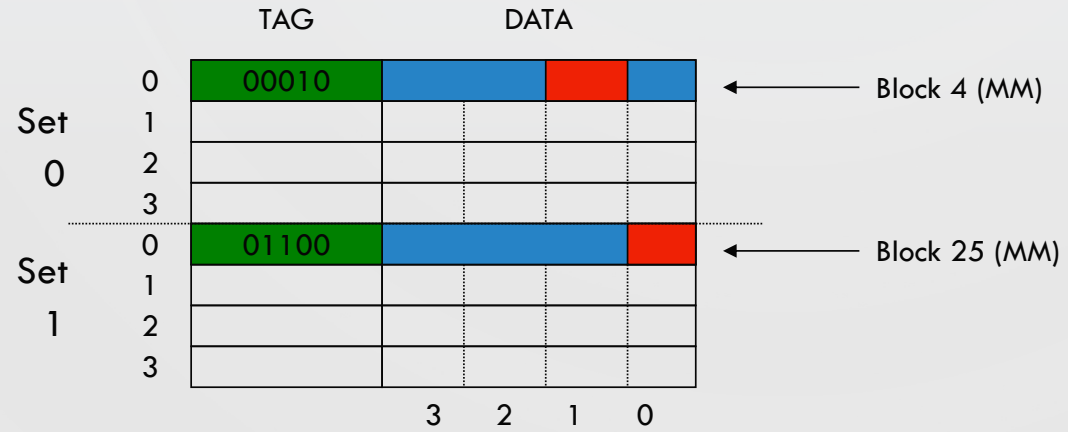


SET-ASSOCIATIVE

Iteration #1:

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





SET-ASSOCIATIVE

Iteration #1:

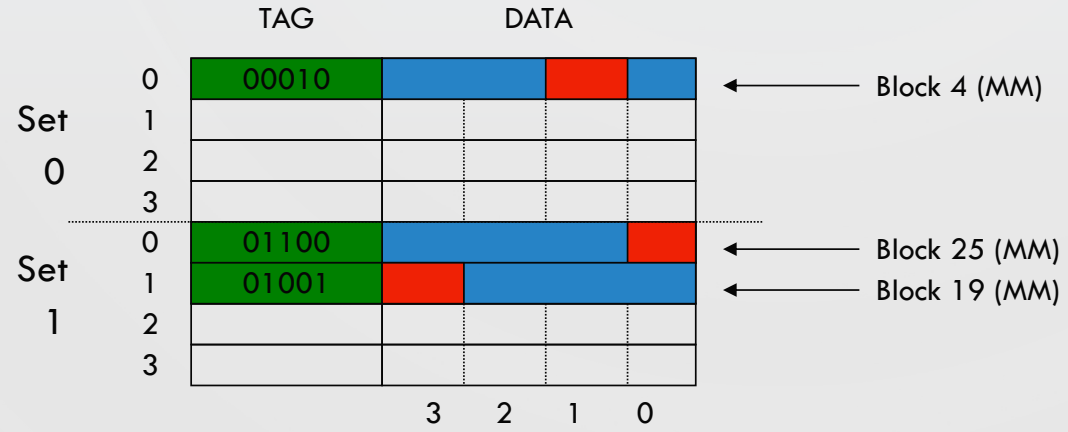
- $100_{(10}$: 011001 | 00
- $17_{(10}$: 000100 | 01
- $79_{(10}$: 010011 | 11 → 01001 | 1 | 11 MISS !!!!

Block 19 (MM)

TAG

Set 1

Word 3



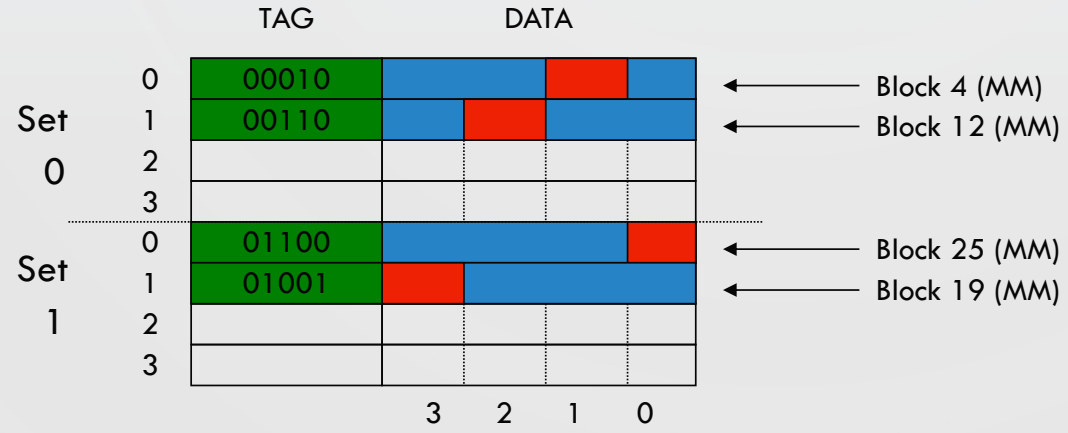
SET-ASSOCIATIVE

Iteration #1:

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

Block 12 (MM)

TAG | Set 0 | word 2

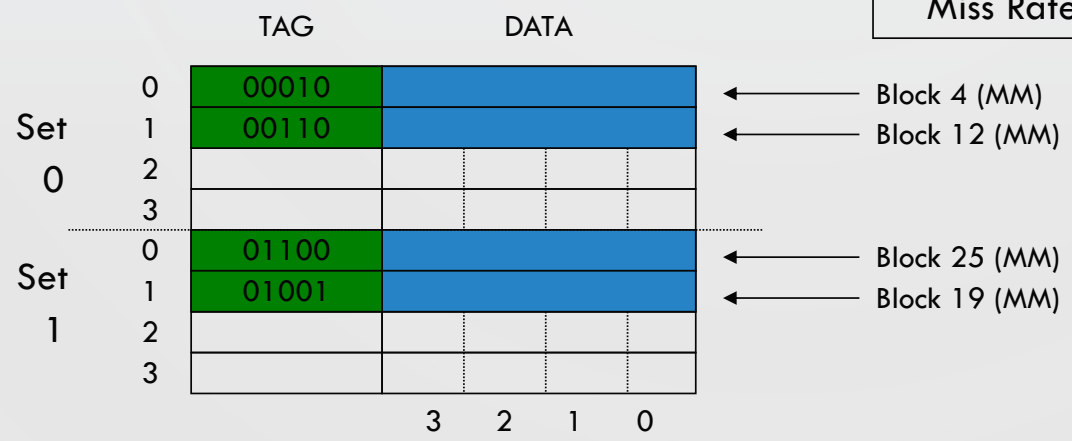


SET-ASSOCIATIVE

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



COSTS OF SET ASSOCIATIVE CACHES

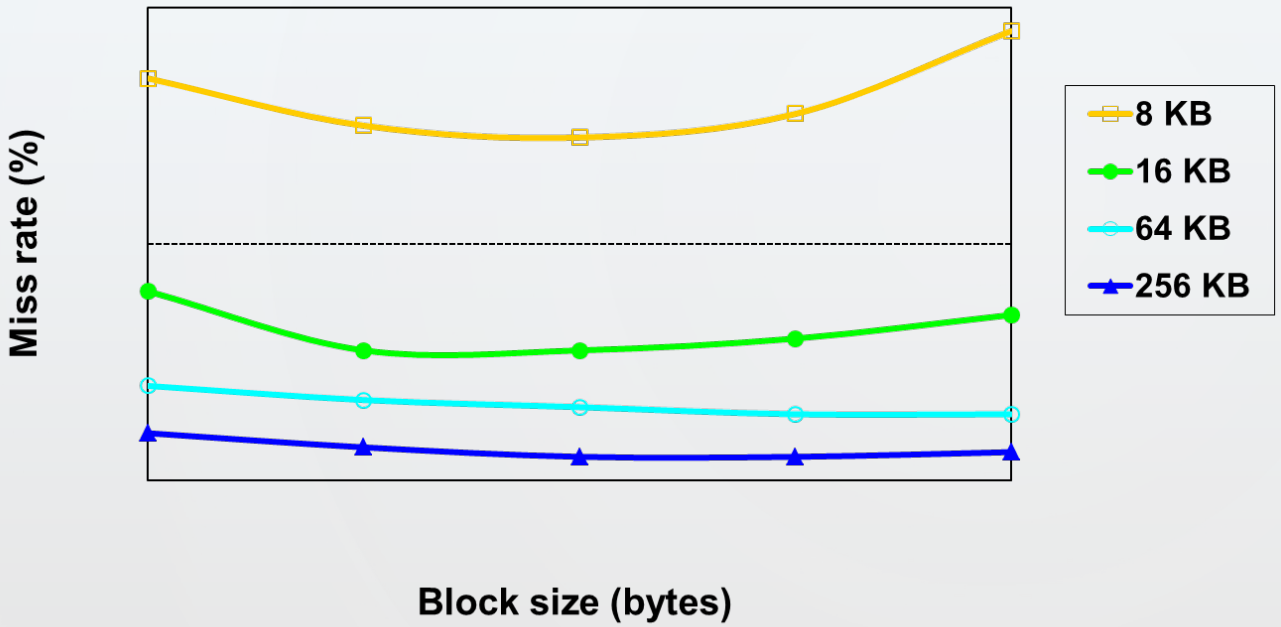
- X-way set associative cache costs
  - X comparators (delay and area)
  - MUX delay (set selection) before data is available
  - Data available after set selection (and Hit/Miss decision). In a direct mapped cache, the cache block is available before the Hit/Miss decision
    - So its not possible to just assume a hit and continue and recover later if it was a miss

Types of Misses

- **Compulsory:** first time data accessed
- **Capacity:** cache too small to hold all data of interest
- **Conflict:** data of interest maps to same location in cache

Miss penalty: time it takes to retrieve a block from lower level of hierarchy

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)

REDUCING CACHE MISS RATES #1

1. Allow more flexible block placement

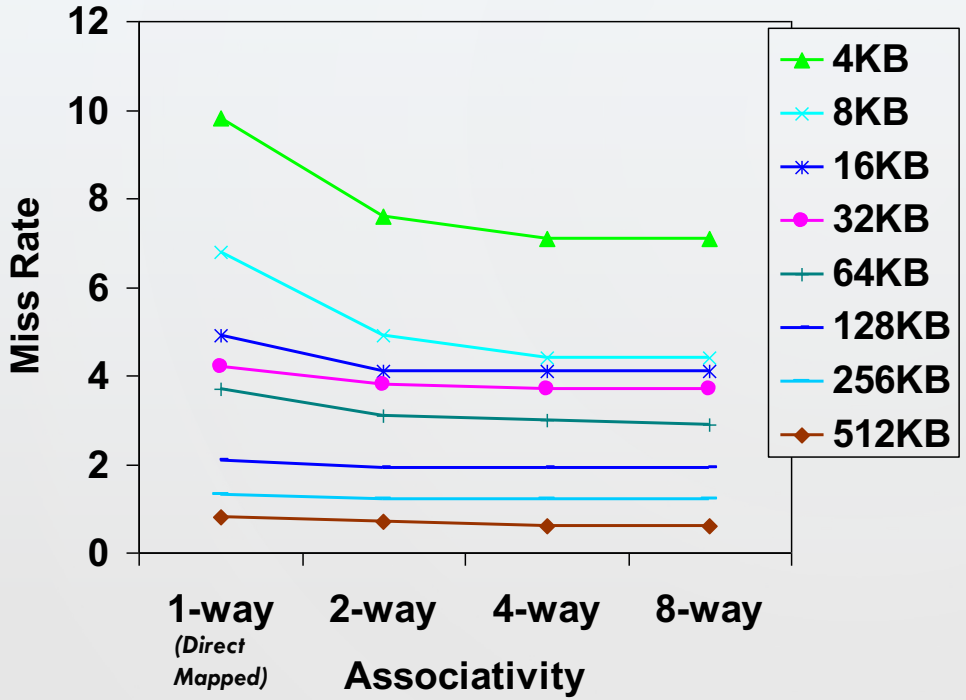
- In a direct mapped cache a memory block maps to exactly one cache block
- At the other extreme, could allow a memory block to be mapped to *any* cache block – fully associative cache

- A compromise is to divide the cache into sets each of which consists of n "ways" (n-way set associative). A memory block maps to a unique set (specified by the index field) and can be placed in any way of that set (so there are n choices)

(block address) modulo (# sets in the cache)

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



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

Cache Organization Recap

- **Capacity:** C
- **Block size:** b
- **Number of blocks in cache:** $B = C/b$
- **Number of blocks in a set:** N
- **Number of sets:** $S = B/N$

Organization	Number of Ways (N)	Number of Sets ($S = B/N$)
Direct Mapped	1	B
N-Way Set Associative	$1 < N < B$	B / N
Fully Associative	B	1

Replacement Policy

- Cache is too small to hold all data of interest at once
- If cache full: program accesses data X and evicts data Y
- ***Capacity miss*** when access Y again
- How to choose Y to minimize chance of needing it again?
  - **Least recently used (LRU) replacement:** the least recently used block in a set evicted

FIFO

Cache

~~1~~ 5
2
3
4

Requested Blocks example: **1, 2, 3, 4, 1, 5,....**

Example #2: **1, 1, 2, 2, 3, 3, 4, 1, 5,....**

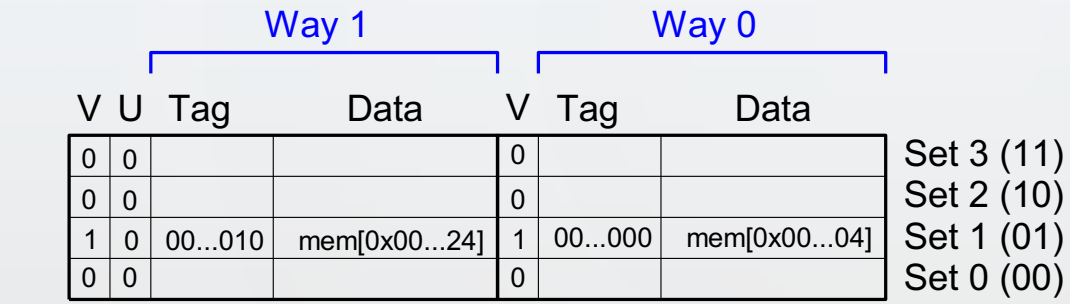
LRU

Cache

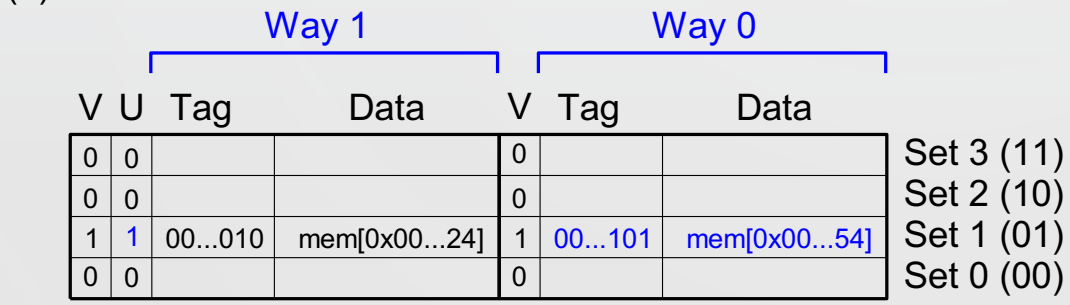
1
~~2~~ 5
3
4

LRU Replacement

```
# RISC-V assembly
lw s1, 0x04(zero)
lw s2, 0x24(zero)
lw s3, 0x54(zero)
```



(a)



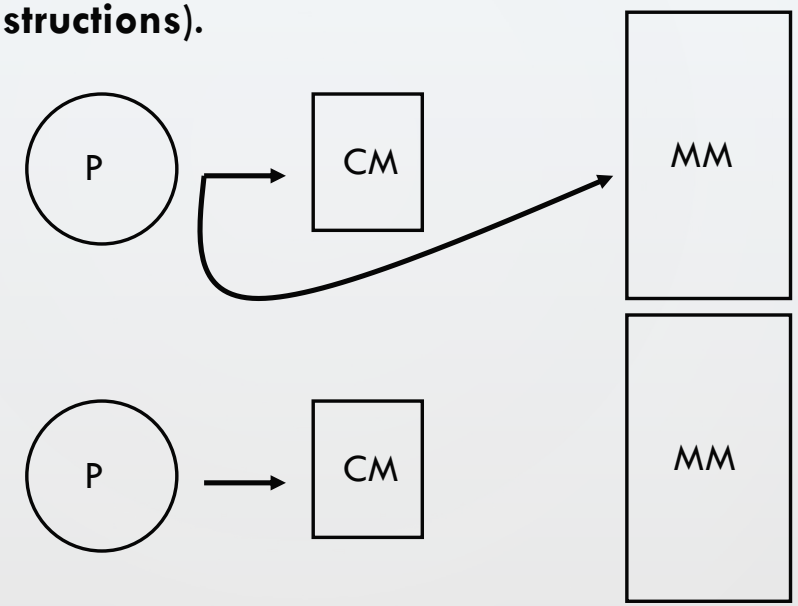
(b)

Write Policy

- Read memory operations are more usual (**Fetch, LWs,…**)…

  … BUT write "also exists" (**SW instructions**).

- Alternatives:
  - **Write-through**
  - **Write-back**



- Additionally considerations….
  - **HITS** or **MISS** in writings:
    - **Write-allocate**
    - **Non-write allocate**

Cache Summary

- **What data is held in the cache?**
  - Recently used data (temporal locality)
  - Nearby data (spatial locality)
- **How is data found?**
  - Set is determined by address of data
  - Word within block also determined by address
  - In associative caches, data could be in one of several ways
- **What data is replaced?**
  - Least-recently used way in the set

MEASURING CACHE PERFORMANCE

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
    - Stalls due to Data and Control Hazards (previous chapter)
  - Memory stall cycles
    - NEW **Cache misses** (I$ + D$)
      - I$ → Instruction Cache
      - D$ → Data Cache

AVERAGE ACCESS TIME

- Hit time is also important for performance
- Average memory access time (AMAT)
  - T_{AMAT} = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, MC miss rate = 5%
  - T_{AMAT} = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

AVERAGE ACCESS TIME

- Hit time is also important for performance
- Average memory access time (AMAT)
  - T_{AMAT} = Hit time + Miss rate × Miss penalty
  - T_{AMAT} = Hit time + Miss rate($I) × Miss penalty($I)+
    + n. acc($D)/NI × Miss rate($D) × Miss penalty($D)
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty ($I & $D)= 80 cycles, I$ miss rate = 2%, $D miss rate=4%, load & store= 36%
  - T_{AMAT} = 1 + 0.02 × 80 + 0.36 x 0.04 x 80 = 3.752 cycles
    - 3.752 ns per instruction

MEASURING CACHE PERFORMANCE

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

$$\text{CPU time} = NI \times CPI \times T_{CC}$$

$$= NI \times \underbrace{(CPI_{base} + \text{Memory-stall cycles}/NI) \times T_{CC}}_{CPI_{effective}}$$

$$\text{Memory-stall cycles} =$$

$$= \text{Num. acc.I\$} \times P_{miss}(I\$) \times TP_{miss}(I\$) + \text{Num. acc. D\$} \times P_{miss}(D\$) \times TP_{miss}(D\$)$$

$$\text{Memory-stall cycles/NI} =$$

$$= 1 \times P_{miss}(I\$) \times TP_{miss}(I\$) + \frac{\text{Num. acc. D\$}}{NI} \times P_{miss}(D\$) \times TP_{miss}(D\$)$$

** CPI_{base} includes the stall due to data & control hazards (perfect cache –no misses)*
** Num. Acc. I$ = NI.*
** Num Acc. D$ = Num. of load and store instructions*

CACHE PERFORMANCE EXAMPLE

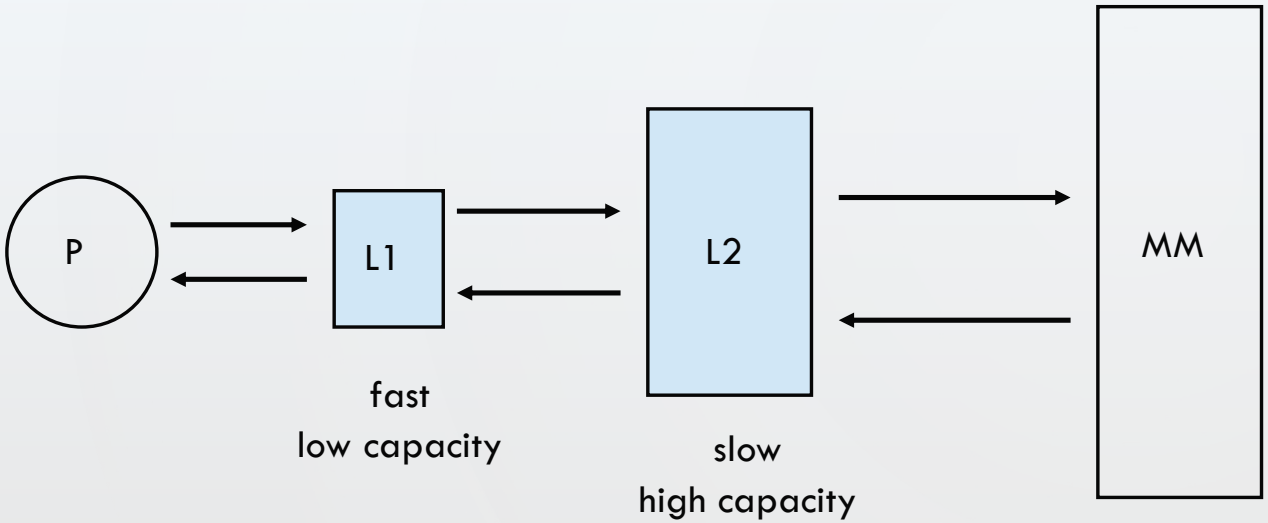
- Given
  - I$ miss rate P_{miss}(I$)= 2%
  - D$ miss rate P_{miss}(D$)= 4%
  - Miss penalty (I$ & D$) TP_{miss}(I$), TP_{miss}(D$) = 80 cycles
  - Base CPI (ideal cache) CPI_{base}= 1.5
  - Load & stores are 36% of instructions *(Num.acc.D$ / num. inst = 0.36)*

- Miss cycles per instruction
  - I$: 0.02 × 80 = 1.6
  - D$: 0.36 × 0.04 × 80 = 1.152
- Effective CPI = 1.5 + 1.6 + 1.152 = 4.252

Multilevel Caches

- Larger caches have lower miss rates, longer access times
- Expand memory hierarchy to multiple levels of caches
- Level 1: small and fast (e.g. 16 KB, 1 cycle)
- Level 2: larger and slower (e.g. 256 KB, 2-6 cycles)
- Most modern PCs have L1, L2, and L3 cache

Multilevel Caches



MULTI-LEVEL CACHE

- With advancing technology have more than enough room on the die for bigger L1 caches *or* for a second level of caches – normally a unified L2 cache (i.e., it holds both instructions and data) and in some cases even a unified L3 cache
- For our example, CPI_{ideal} of 2, 100 cycle miss penalty (to main memory) and a 25 cycle miss penalty (to L2$), 36% load/stores, a 2% (4%) L1 I$ (D$) miss rate, add a 0.5% L2$ miss rate

$$CPI_{effective} = 2 + .02 \times 25 + .005 \times 100 + .36 \times .04 \times 25 + .36 \times .005 \times 100 = 3.54$$

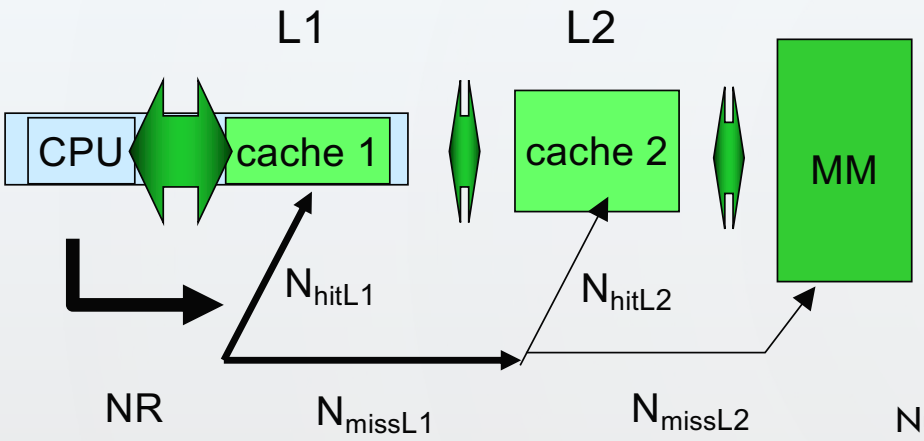
(compared to $CPI_{effective} = 2 + .02 \times 100 + 0.36 \times 0.04 \times 100 = 5.44$ with no UL2$)

MULTILEVEL CACHE DESIGN CONSIDERATIONS

- Design considerations for L1 and L2 caches are very different
  - Primary cache should focus on minimizing hit time in support of a shorter clock cycle
    - Smaller with smaller block sizes
  - Secondary cache(s) should focus on reducing miss rate to reduce the penalty of long main memory access times
    - Larger with larger block sizes
    - Higher levels of associativity
- The miss penalty of the L1 cache is significantly reduced by the presence of an L2 cache – so it can be smaller (i.e., faster) but have a higher miss rate
- For the L2 cache, hit time is less important than miss rate
  - The L2$ hit time determines L1$'s miss penalty
  - L2$ local miss rate >> than the global miss rate

Multi-level cache

✓ Two level cache L_1 y L_2



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

$$NR = N_{hitL1} + N_{missL1} = N_{hitL1} + (N_{hitL2} + N_{missL2})$$

Number of references of L2 is the number of misses of L1

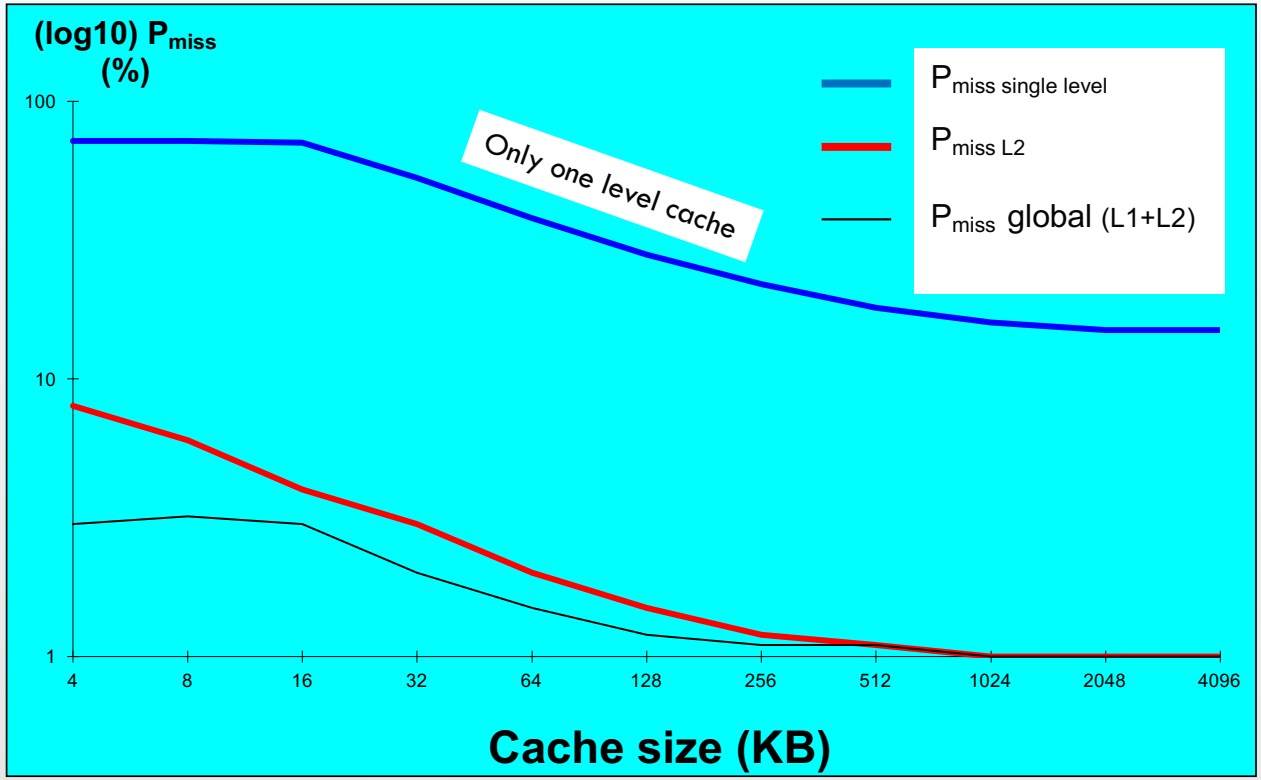
$$P_{missL1} = \frac{N_{missL1}}{NR} \rightarrow \text{Local miss rate L1}$$

$$P_{missL2} = \frac{N_{missL2}}{N_{missL1}} \rightarrow \text{Local miss rate L2}$$

L1. Upper level (closest to CPU)
(For sepatate caches L1I y L1D).

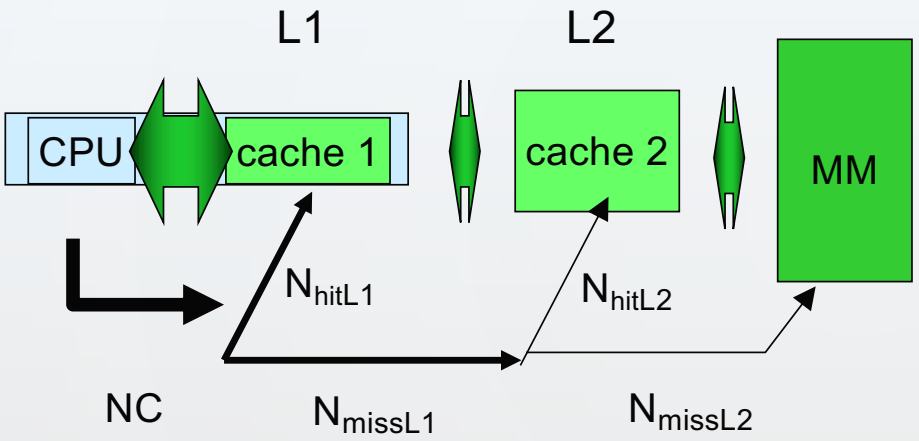
L2. Lower level (Secondary cache).
(Normally it is unified)

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


(log10) P miss
(%)
100
10
1
Only one level cache
P miss single level
P miss L2
P miss global (L1+L2)
4
8
16
32
64
128
256
512
1024
2048
4096
Cache size (KB)

Multi-level cache

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



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

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

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

EXAMPLE

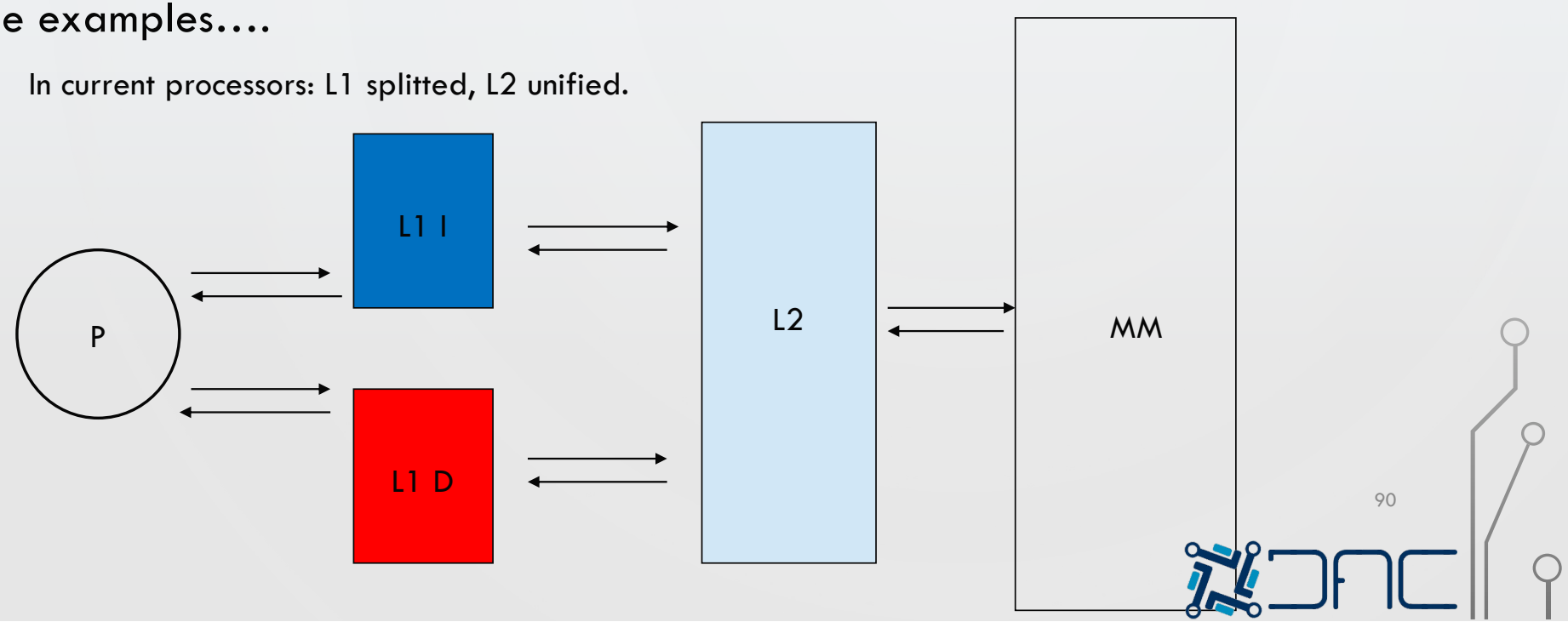
- Given
  - 2500 references NR= 2500 N_{missL1}=50
  - 50 misses at L1 → N_{missL2}=5
  - 5 misses at L2 →
  - Miss penalty for L2 : 100 cc → TP_{missL2}=100
  - Access time for L2: 12 cc → T_{hitL2} = 12
  - Hit time at L1: 1 cc
- Global miss rate?
  - P_{missL1} = 50/2500 = 0,02 P_{missL2} = 5/50 = 0.10
  - P_{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

  Withouh L2: AMAT=1 + 0,02 x 100= 3

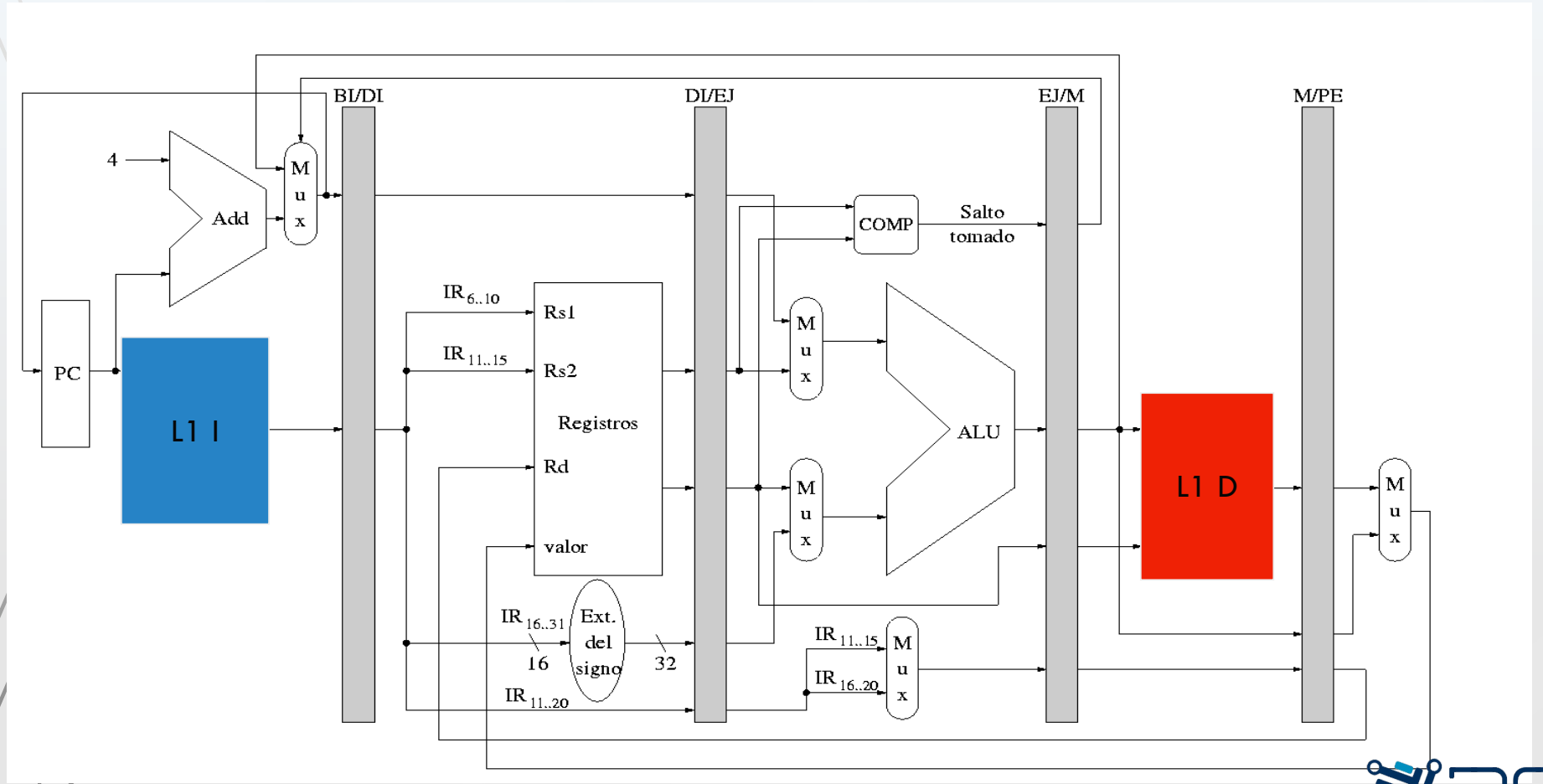
Cache Splitting

- 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.



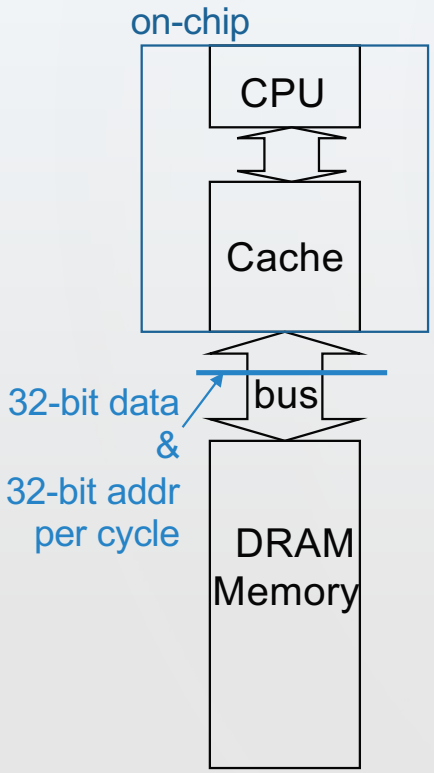
Cache Splitting



Characteristic	ARM Cortex-A53	Intel Core i7
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	Configurable 16 to 64 KiB each for instructions/data	32 KiB each for instructions/data per core
L1 cache associativity	Two-way (I), four-way (D) set associative	Four-way (I), eight-way (D) set associative
L1 replacement	Random	Approximated LRU
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, variable allocation policies (default is Write-allocate)	Write-back, No-write-allocate
L1 hit time (load-use)	Two clock cycles	Four clock cycles, pipelined
L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
L2 cache size	128 KiB to 2 MiB	256 KiB (0.25 MiB)
L2 cache associativity	16-way set associative	8-way set associative
L2 replacement	Approximated LRU	Approximated LRU
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L2 hit time	12 clock cycles	10 clock cycles
L3 cache organization	–	Unified (instruction and data)
L3 cache size	–	8 MiB, shared
L3 cache associativity	–	16-way set associative
L3 replacement	–	Approximated LRU
L3 block size	–	64 bytes
L3 write policy	–	Write-back, Write-allocate
L3 hit time	–	35 clock cycles

MEMORY SYSTEMS THAT SUPPORT CACHES

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

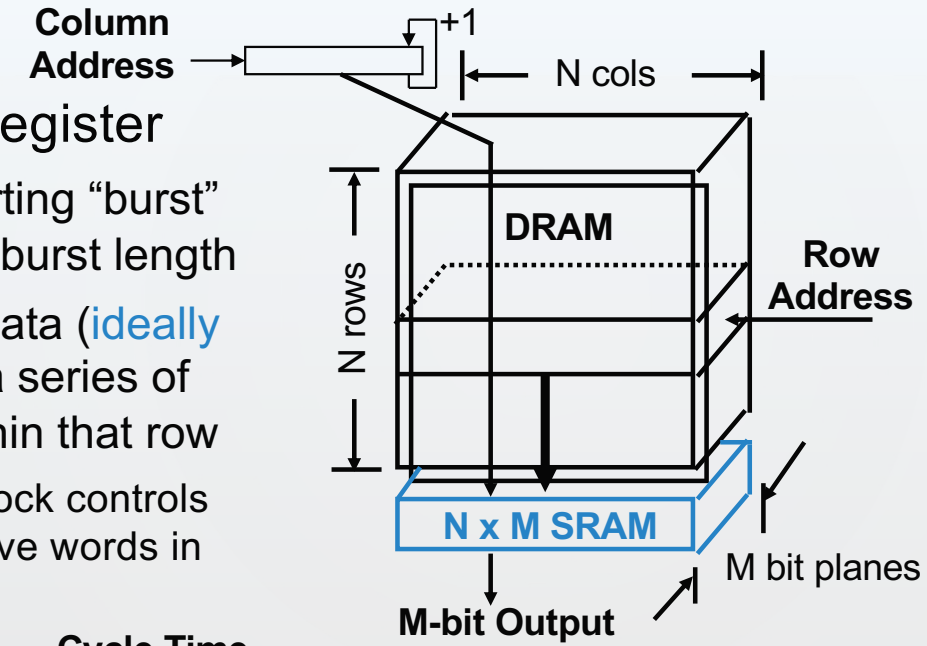


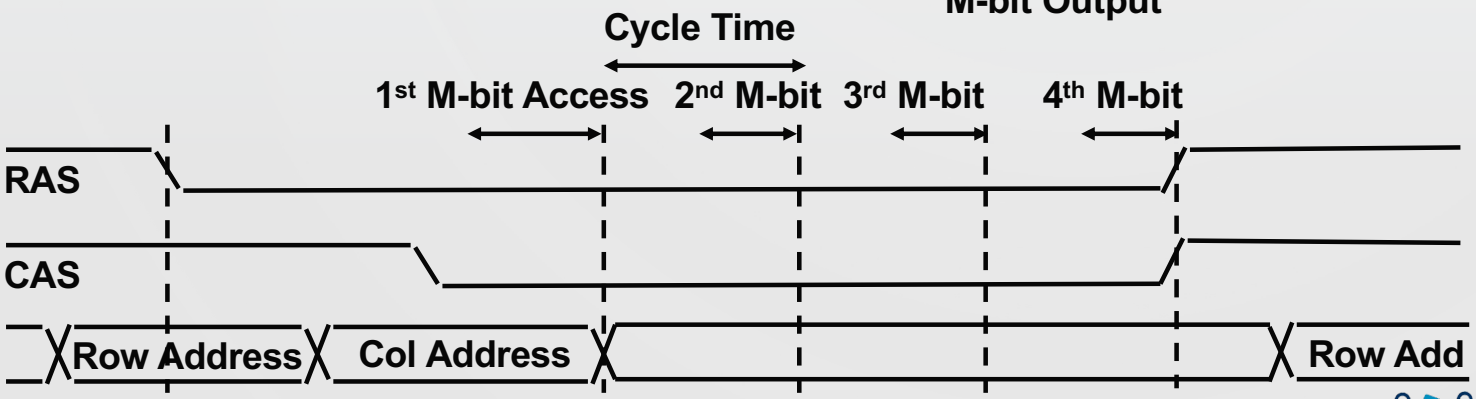
One word wide organization (one word wide bus and one word wide memory)

- 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

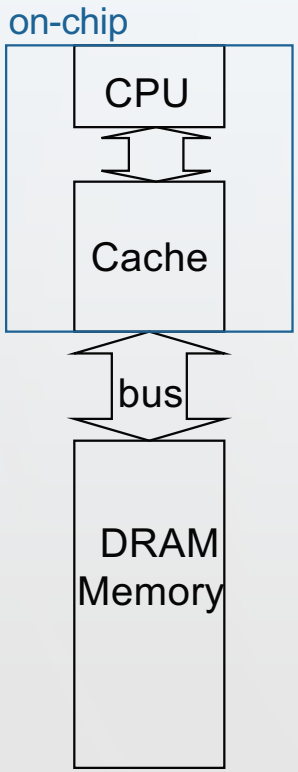
REVIEW: (DDR) SDRAM OPERATION (BURST MODE)

- After a row is read into the SRAM register
  - Input CAS as the starting "burst" address along with a burst length
  - Transfers a burst of data (ideally a cache block) from a series of sequential addr's within that row
    - The memory bus clock controls transfer of successive words in the burst



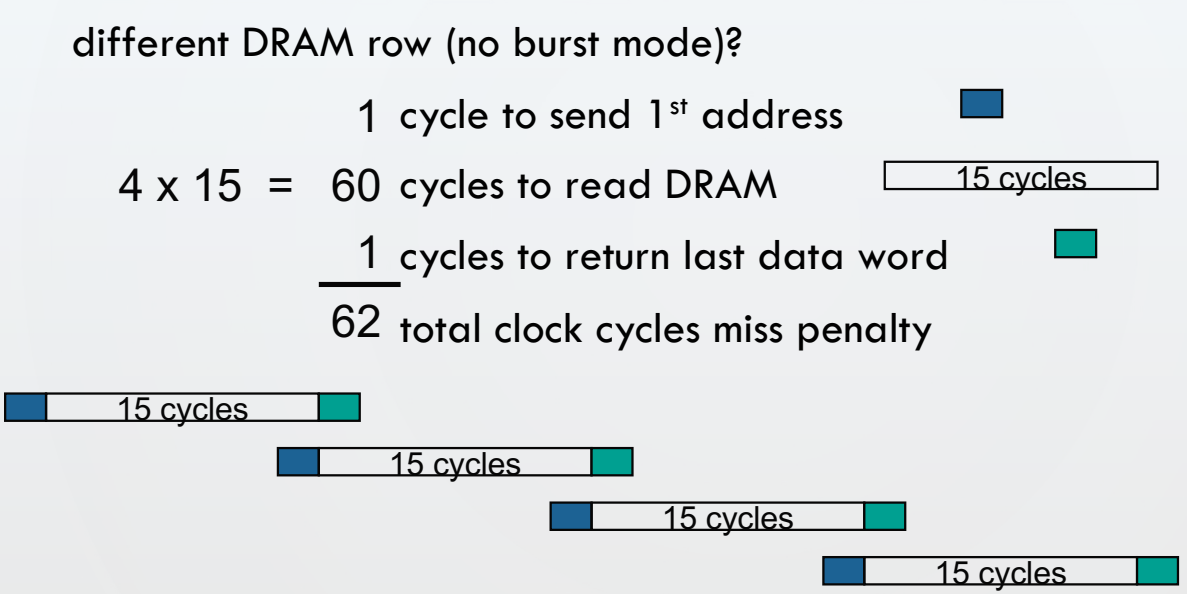


ONE WORD WIDE BUS, FOUR WORD BLOCKS



- What if the block size is four words and each word is in a different DRAM row (no burst mode)?

	1	cycle to send 1st address
4 x 15 =	60	cycles to read DRAM
	1	cycles to return last data word
	62	total clock cycles miss penalty

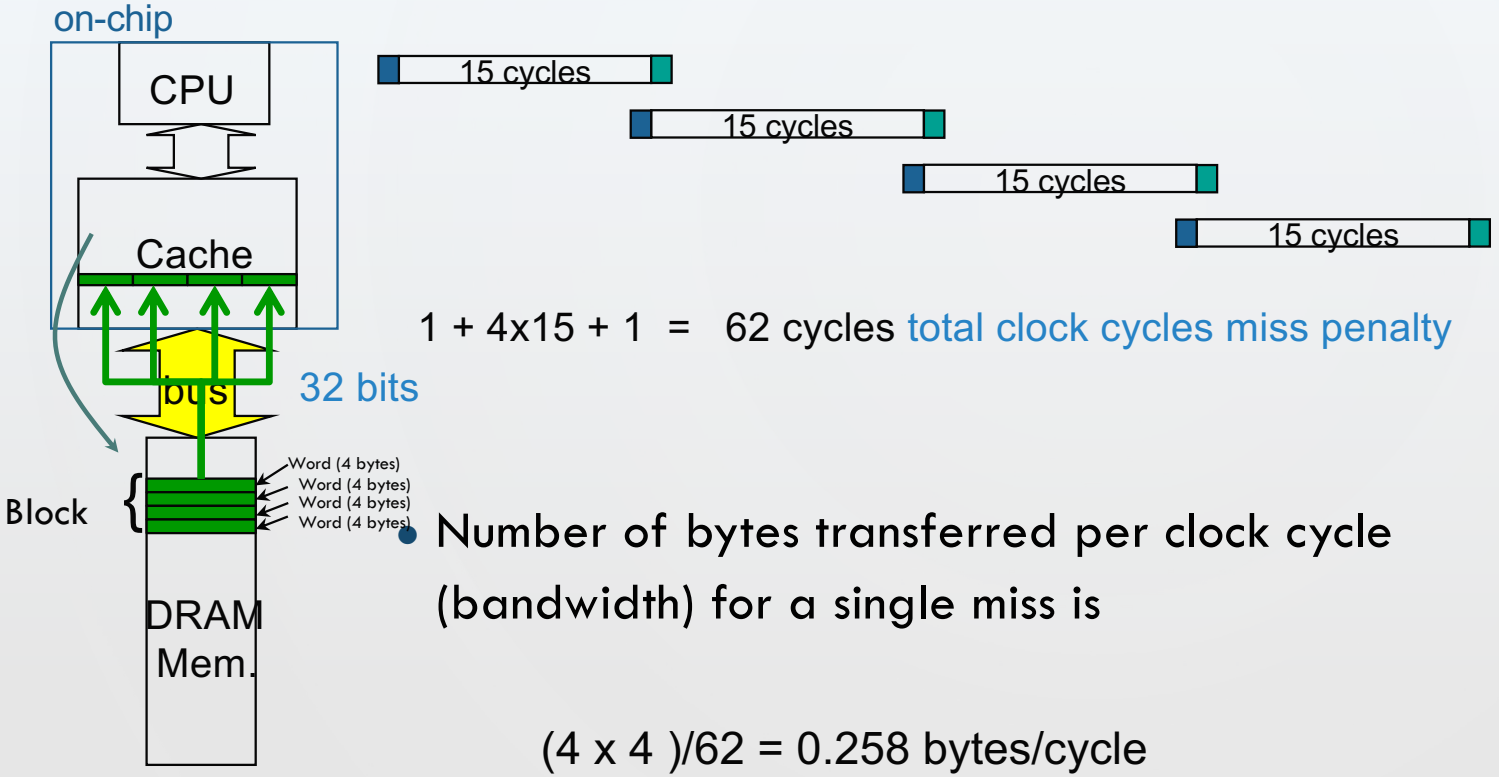


- Number of bytes transferred per clock cycle (bandwidth) for a single miss is

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

ONE WORD WIDE BUS, FOUR WORD BLOCKS

- To copy a block (4 words/block):

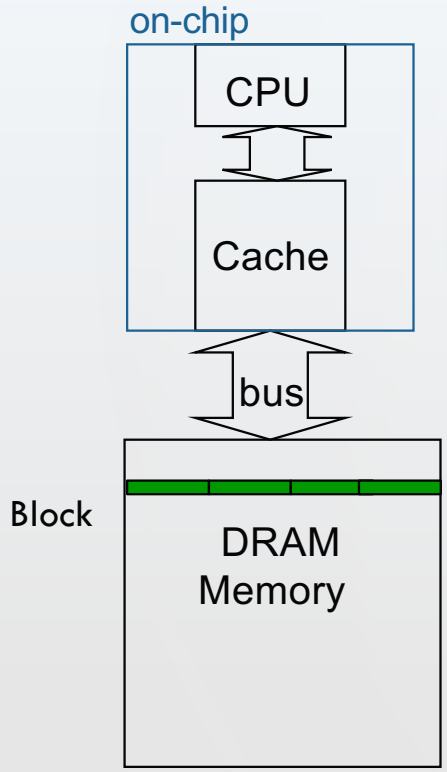


1 + 4x15 + 1 = 62 cycles total clock cycles miss penalty

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

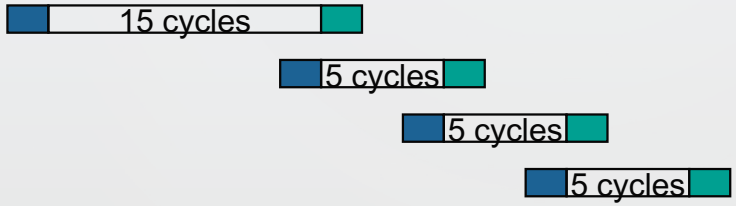
(4 x 4)/62 = 0.258 bytes/cycle

ONE WORD WIDE BUS, FOUR WORD BLOCKS



- What if the block size is four words and all words are in the same DRAM row (burst mode)?

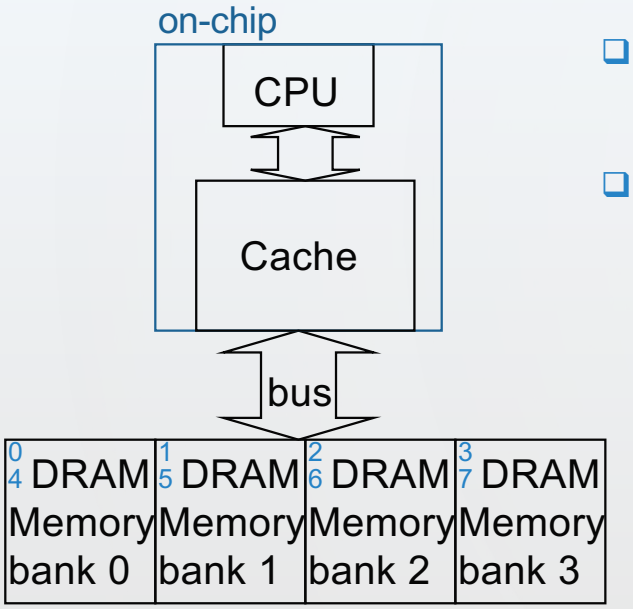
1	cycle to send 1st address
15 + 3*5 = 30	cycles to read DRAM
1	cycles to return last data word
32	total clock cycles miss penalty



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

$$(4 \times 4)/32 = 0.5 \text{ bytes per clock}$$

INTERLEAVED MEMORY, ONE WORD WIDE BUS

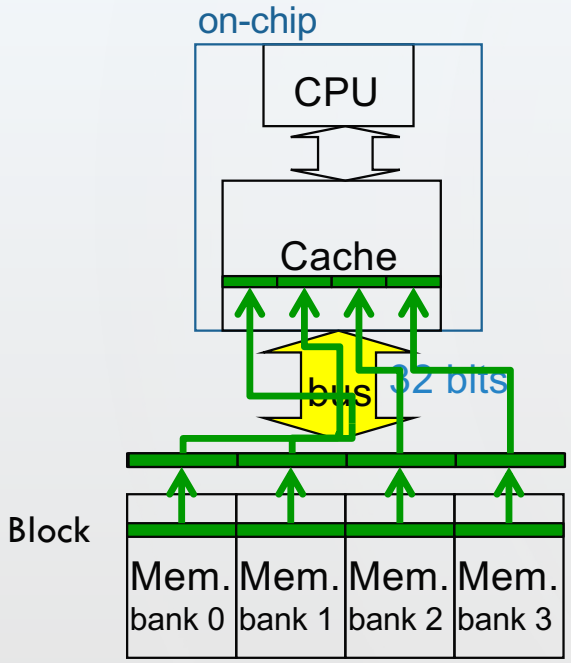


- Consecutive addresses goes to consecutive memory modules
- *Address* ***Module***

0	0
1	1
2	2
3	3
4	0
…	

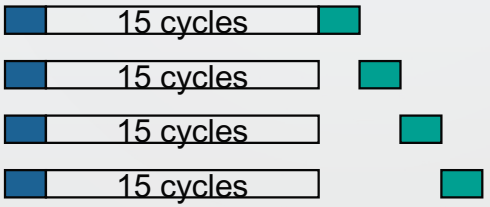
One access → 4 words in parallel

INTERLEAVED MEMORY, ONE WORD WIDE BUS



- For a block size of four words

	1	cycle to send 1st address
	15	cycles to read DRAM banks
4*1 =	4	cycles to return last data word
	20	total clock cycles miss penalty

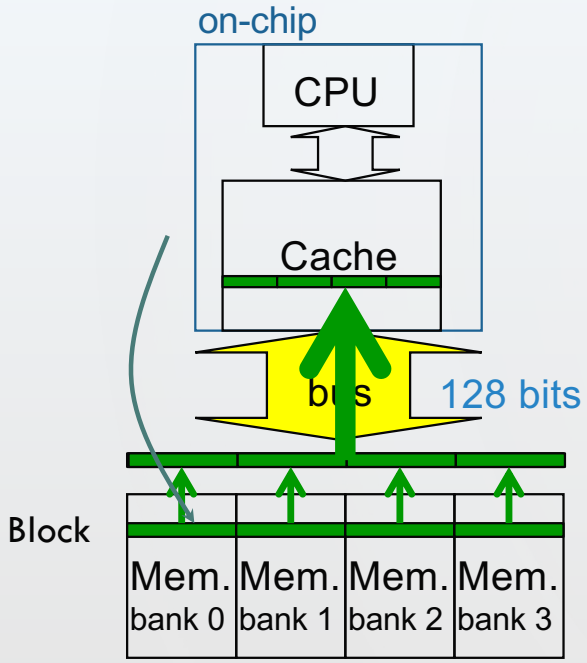


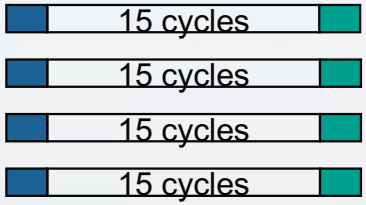
- Number of bytes transferred per clock cycle (bandwidth) for a single miss is

(4 x 4)/20 = 0.8 bytes per cycle

INTERLEAVED MEMORY, ONE WORD WIDE BUS

- An improvement: wider bus (128 bits)





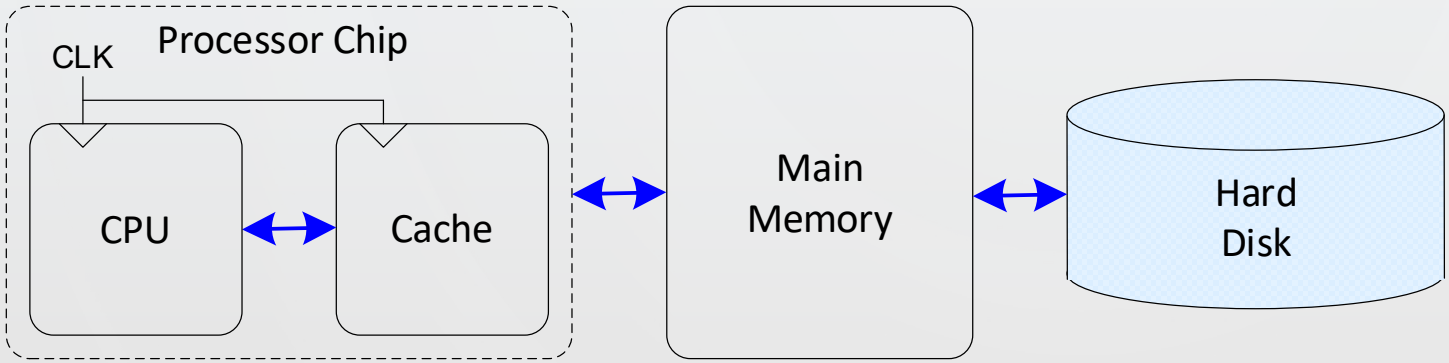
1+ 15 + 1 = 17 cycles total clock cycles miss penalty

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

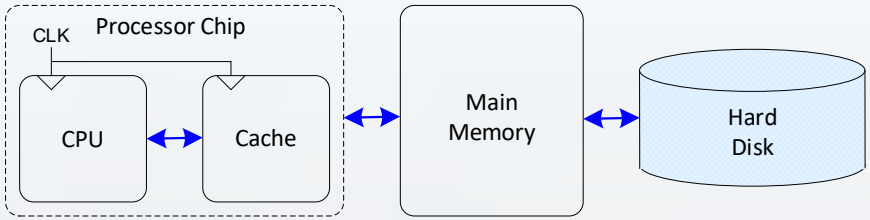
(4 x 4)/17 = 0.94 bytes per cycle

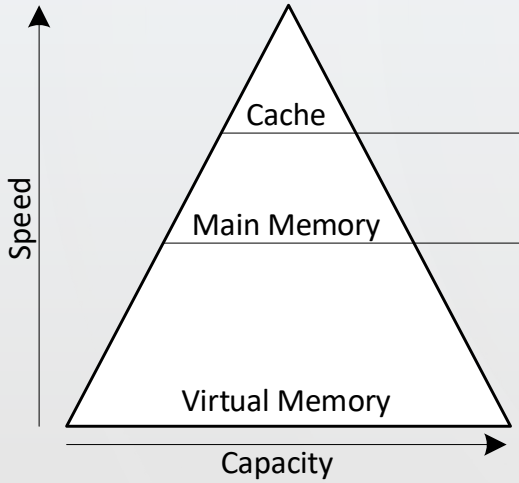
Virtual Memory

- Gives the illusion of bigger memory
- Main memory (DRAM) acts as cache for hard disk



Memory Hierarchy



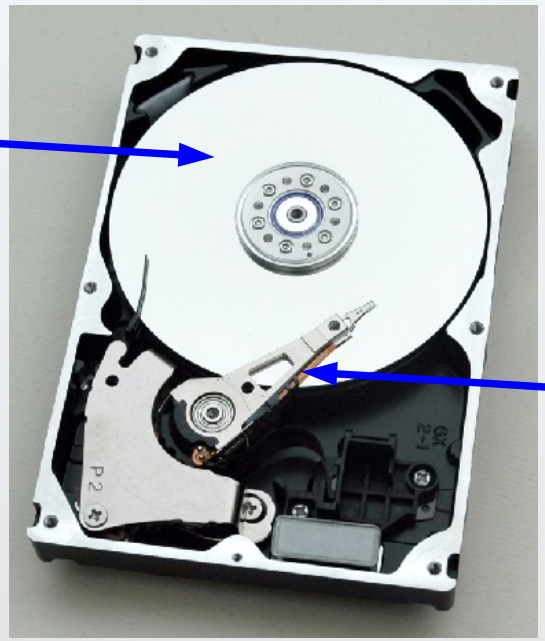


Technology	Price / GB	Access Time (ns)	Bandwidth (GB/s)
SRAM	$100	0.2 - 3	100+
DRAM	$3	10 - 50	30
SSD	$0.10	20,000	0.05 - 3
HDD	$0.03	5,000,000	0.001 - 0.1

- **Physical Memory:** DRAM (Main Memory)
- **Virtual Memory:** Hard drive
  - Slow, Large, Cheap

Memory Hierarchy

Hard Disk Drive



Takes milliseconds to *seek* correct location on disk

Solid State Drive

Arshane88 / CC BY-SA 4.0 / Wikimedia Commons

Virtual Memory

- **Virtual addresses**
  - Programs use virtual addresses
  - Entire virtual address space stored on a hard drive
  - Subset of virtual address data in DRAM
  - CPU translates virtual addresses into ***physical addresses*** (DRAM addresses)
  - Data not in DRAM fetched from hard drive
- **Memory Protection**
  - Each program has own virtual to physical mapping
  - Two programs can use same virtual address for different data
  - Programs don't need to be aware others are running
  - One program (or virus) can't corrupt memory used by another

Virtual Memory

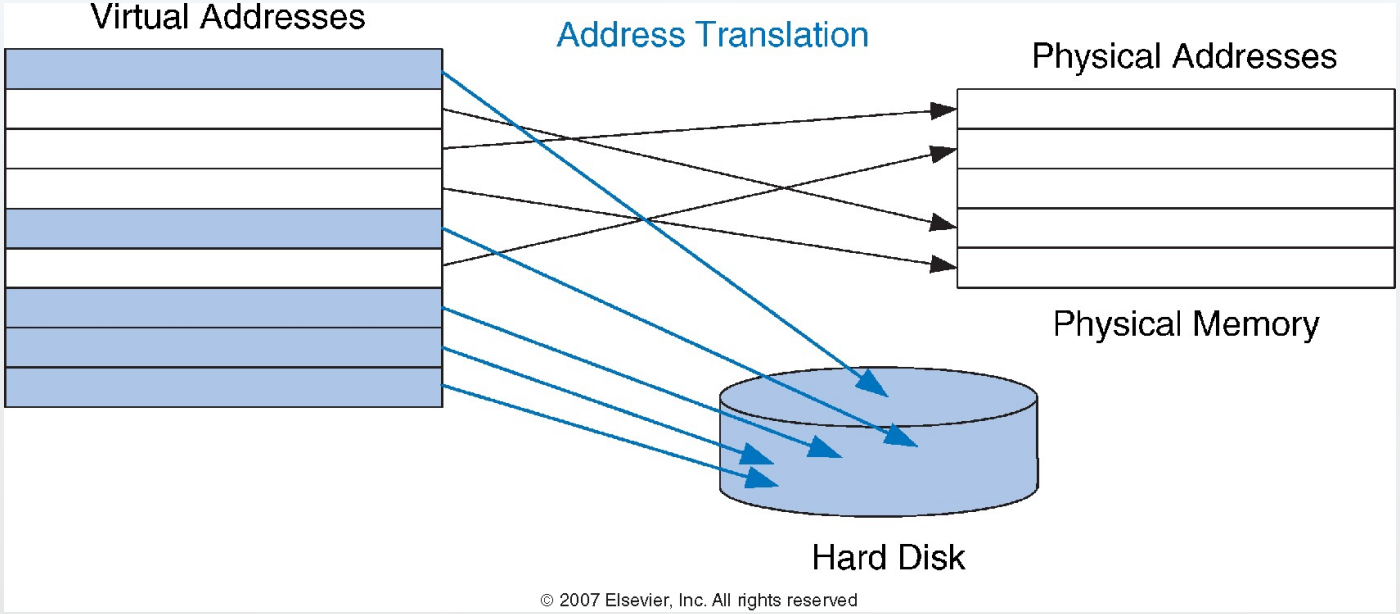
Cache	Virtual Memory
Block	Page
Block Size	Page Size
Block Offset	Page Offset
Miss	Page Fault
Tag	Virtual Page Number

Physical memory acts as cache for virtual memory

Virtual Memory Definitions

- **Page size:** amount of memory transferred from hard disk to DRAM at once
- **Address translation:** determining physical address from virtual address
- **Page table:** lookup table used to translate virtual addresses to physical addresses

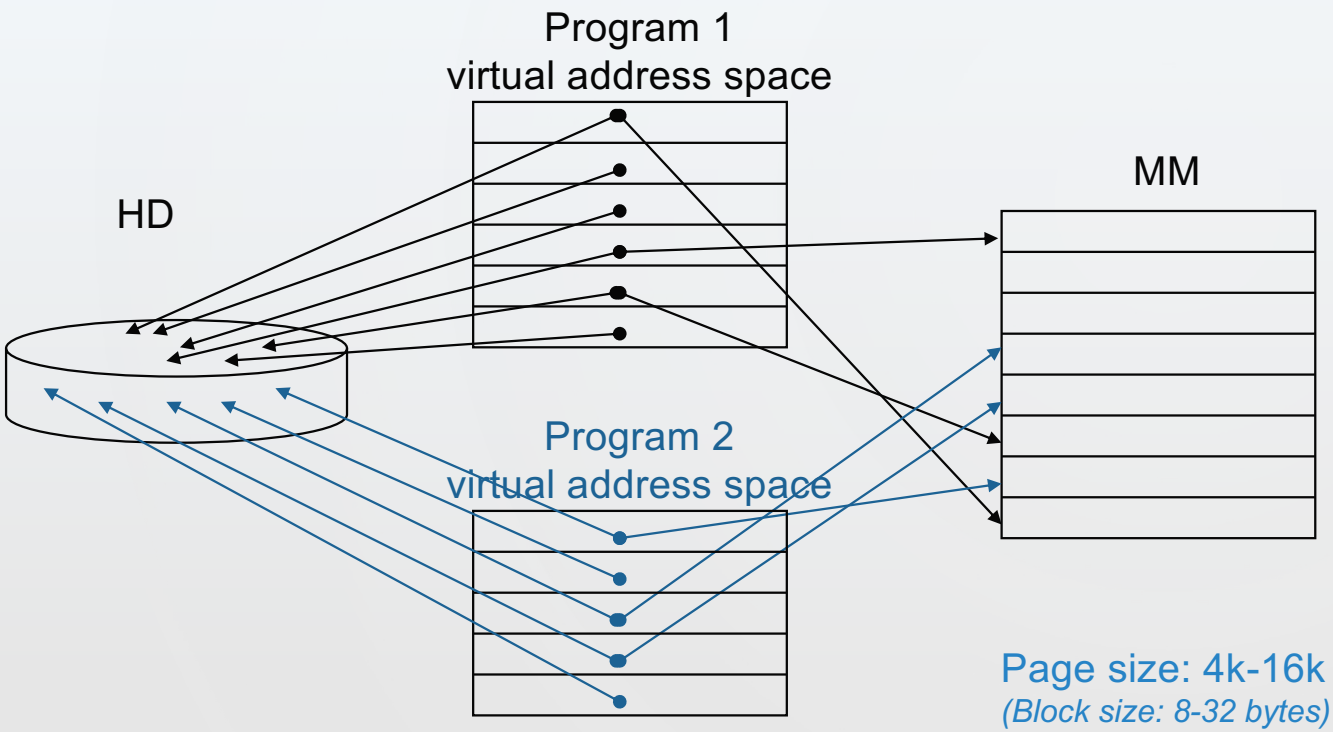
Virtual Memory Definitions



Most accesses hit in physical memory

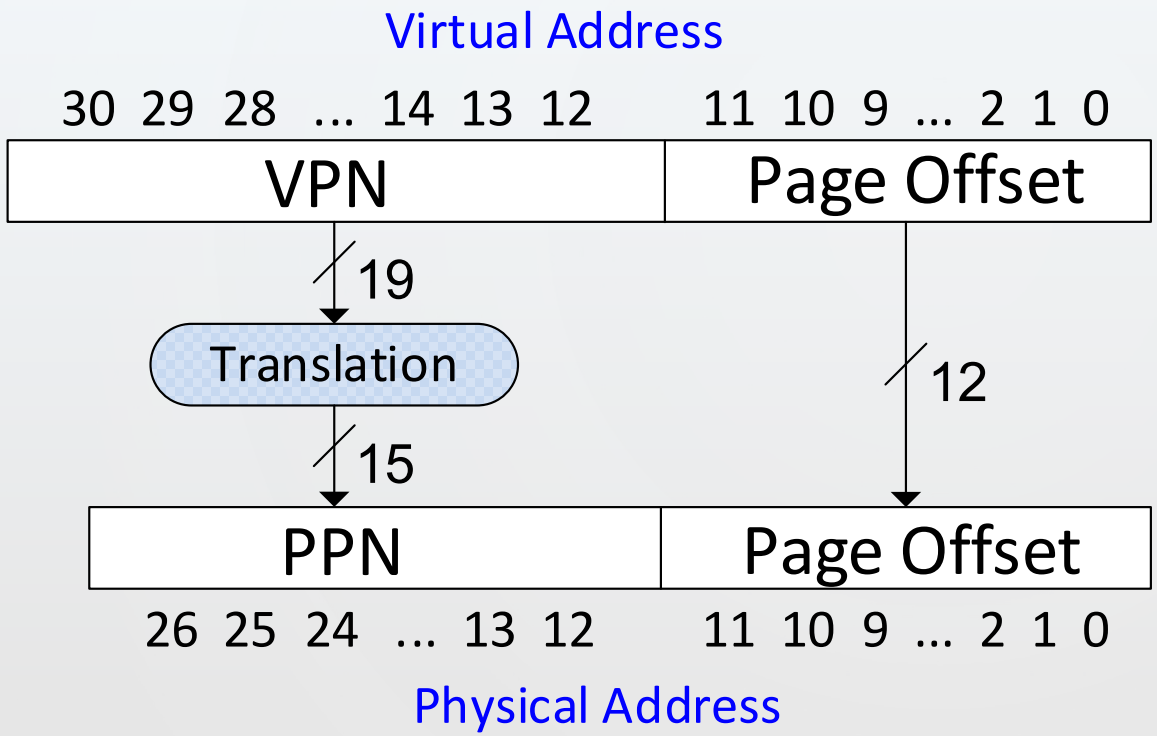
But programs have the **large capacity** of virtual memory

Virtual Memory example



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

Address Translation

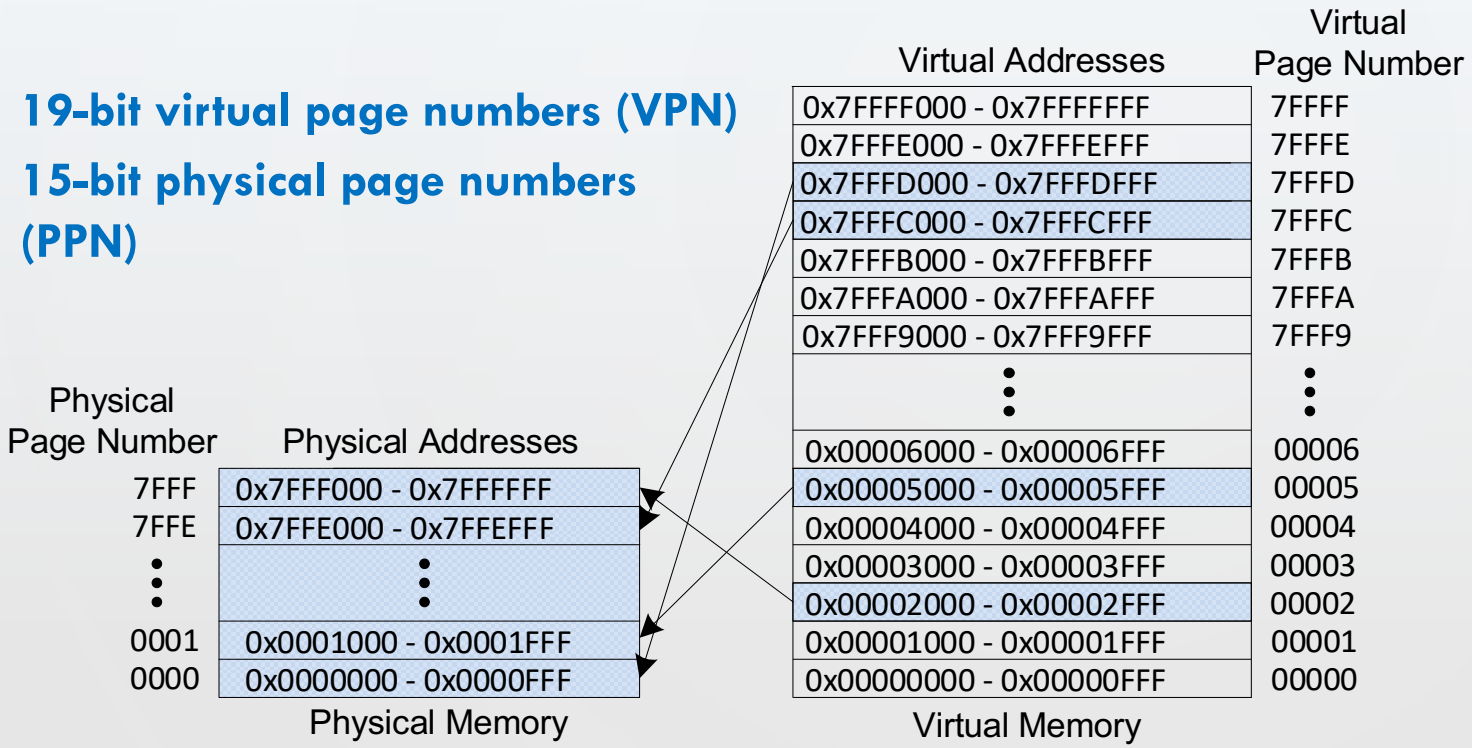


Virtual Memory Example

- **System:**
  - Virtual memory size: 2 GB = 2^{31} bytes
  - Physical memory size: 128 MB = 2^{27} bytes
  - Page size: 4 KB = 2^{12} bytes
- **Organization:**
  - Virtual address: **31** bits
  - Physical address: **27** bits
  - Page offset: **12** bits
  - # Virtual pages = $2^{31}/2^{12}$ = **2^{19}** (VPN = **19** bits)
  - # Physical pages = $2^{27}/2^{12}$ = **2^{15}** (PPN = **15** bits)

Virtual Memory Example

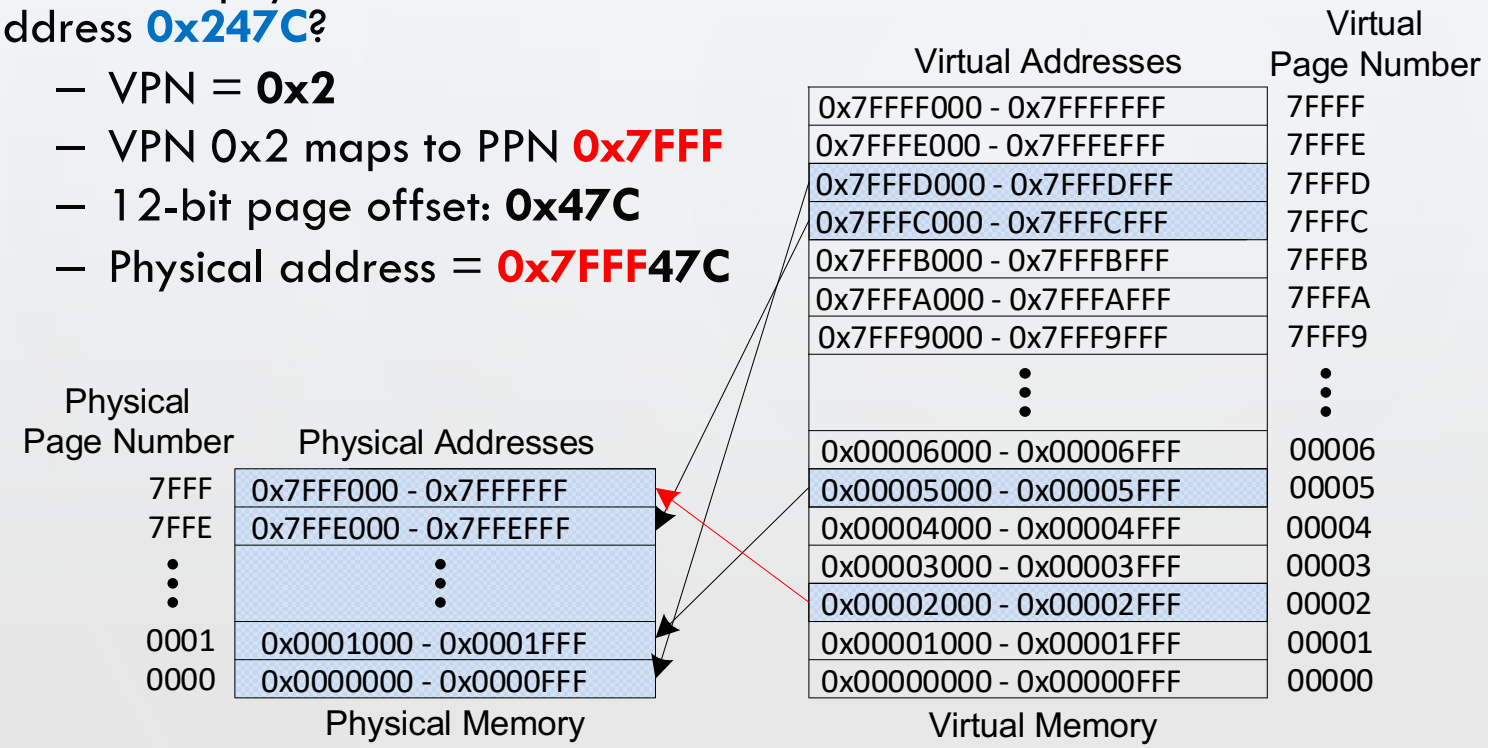
- **19-bit virtual page numbers (VPN)**
- **15-bit physical page numbers (PPN)**



Virtual Memory Example

What is the physical address of virtual address **0x247C**?

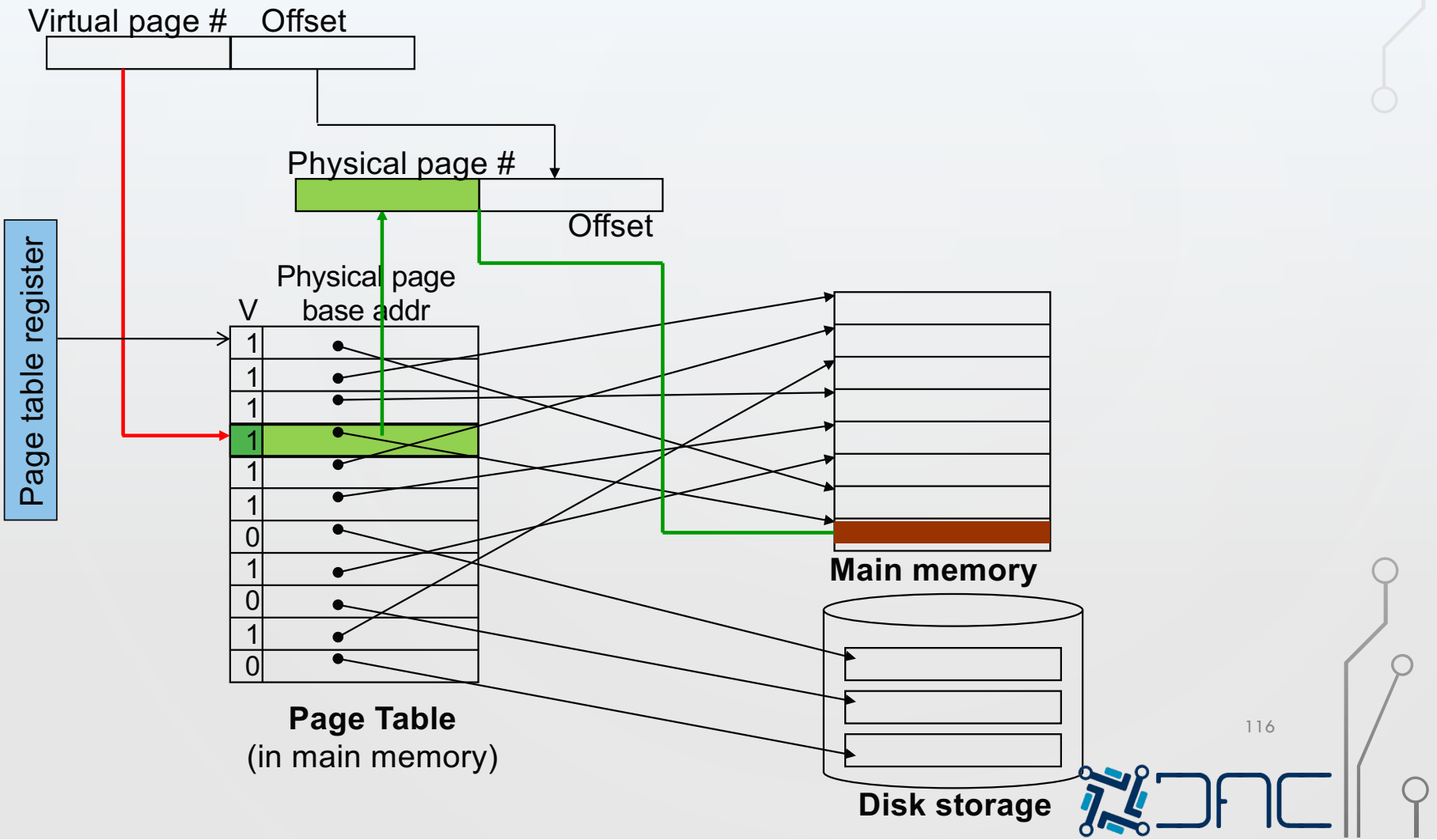
- VPN = **0x2**
- VPN 0x2 maps to PPN **0x7FFF**
- 12-bit page offset: **0x47C**
- Physical address = **0x7FFF47C**



Page Table

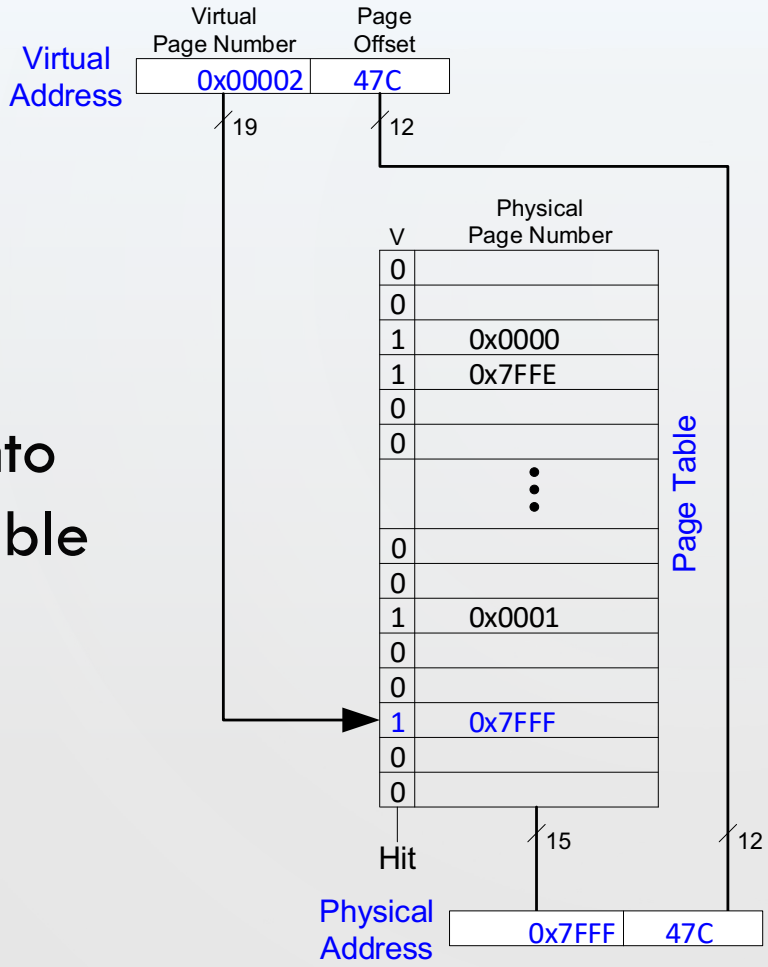
- How to Perform Translation
- **Page table**
  - Entry for each virtual page
  - Entry fields:
    - **Valid bit:** 1 if page in physical memory
    - **Physical page number:** where the page is located

Page Table Mechanism



Page Table Example

VPN is **index** into page table



Page Table Example 1

What is the physical address of virtual address **0x5F20**?

- VPN = **5**
- Entry 5 in page table VPN 5 => physical page **1**
- Physical address: **0x1F20**

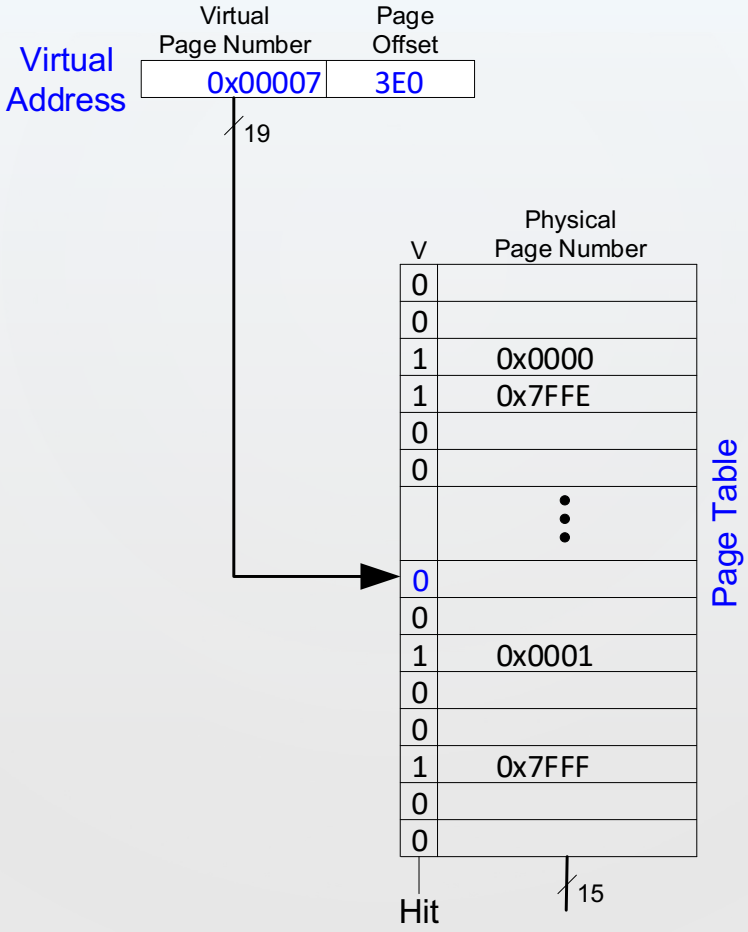
V	Physical Page Number	Virtual Page Number
0		7FFFF
0		7FFFE
1	0x0000	7FFFD
1	0x7FFE	7FFFC
0		7FFFB
0		7FFFA
	⋮	⋮
0		00007
0		00006
1	0x0001	00005
0		00004
0		00003
1	0x7FFF	00002
0		00001
0		00000

Page Table

Page Table Example 2

What is the physical address of virtual address **0x73E4**?

- VPN = **7**
- Entry 7 is invalid
- Virtual page must be ***paged*** into physical memory from disk



Page Table Challenges

- Page table is **large**
  - usually located in physical memory
- Load/store requires **2 main memory accesses:**
  - one for translation (page table read)
  - one to access data (after translation)
- Cuts memory performance in half
  - ***Unless we get clever…***

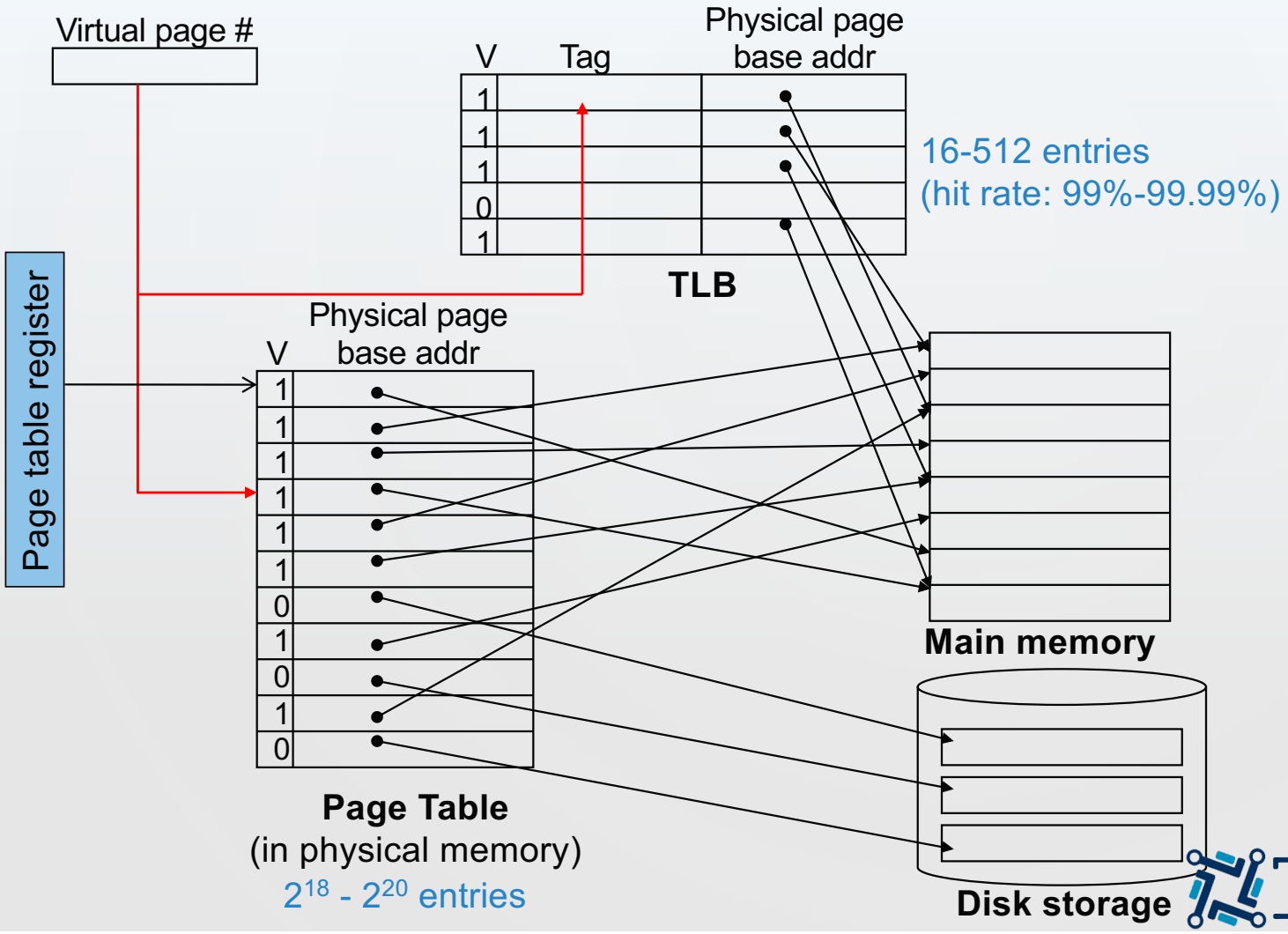
Translation Lookaside Buffer (TLB)

- Small cache of most recent translations
- Reduces number of memory accesses for *most* loads/stores from 2 to 1

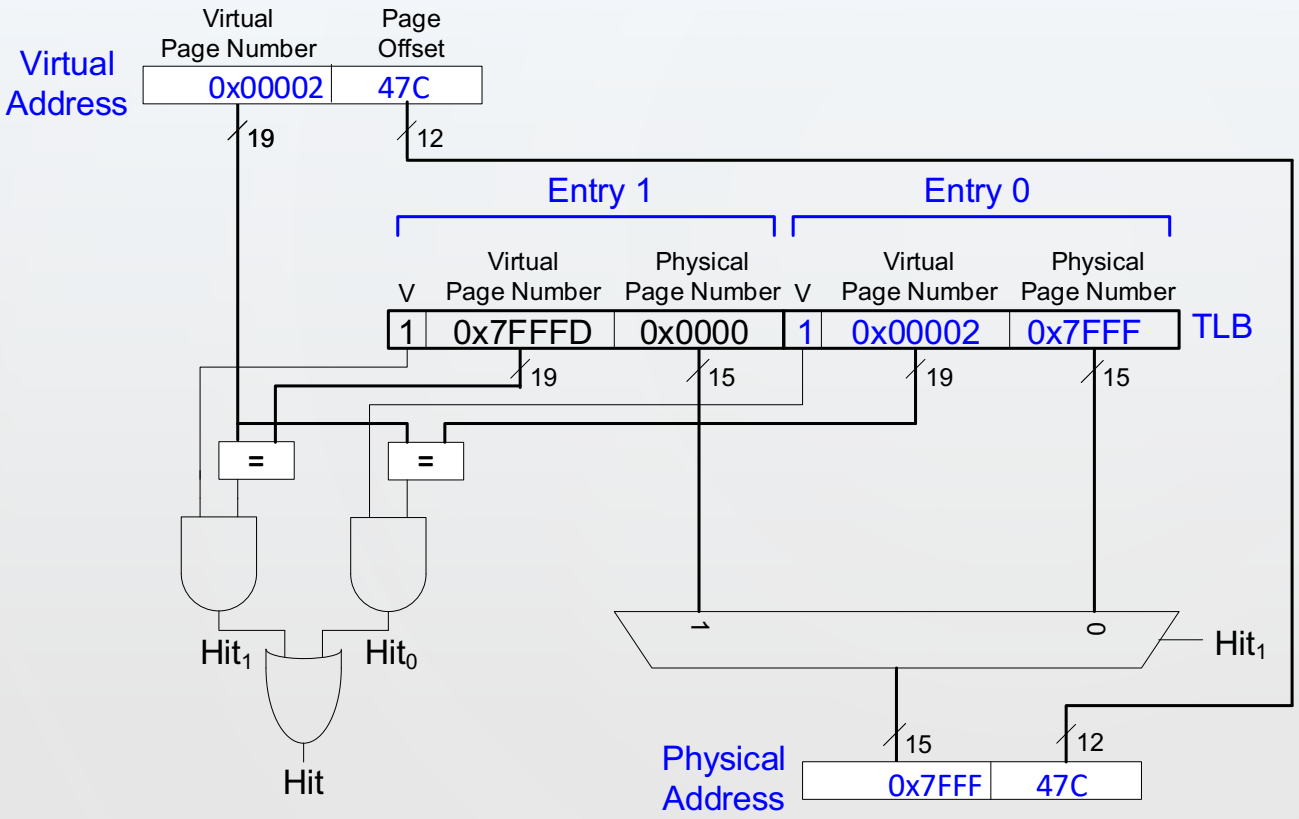
TLB

- **Page table accesses:** high temporal locality
  - Large page size, so consecutive loads/stores likely to access same page
- **TLB**
  - **Small:** accessed in < 1 cycle
  - Typically **16 - 512 entries**
  - **Fully associative**
  - **> 99%** hit rates typical
  - **Reduces number of memory accesses** for most loads/stores from 2 to 1

TLB mechanism



Example: 2-entry TLB



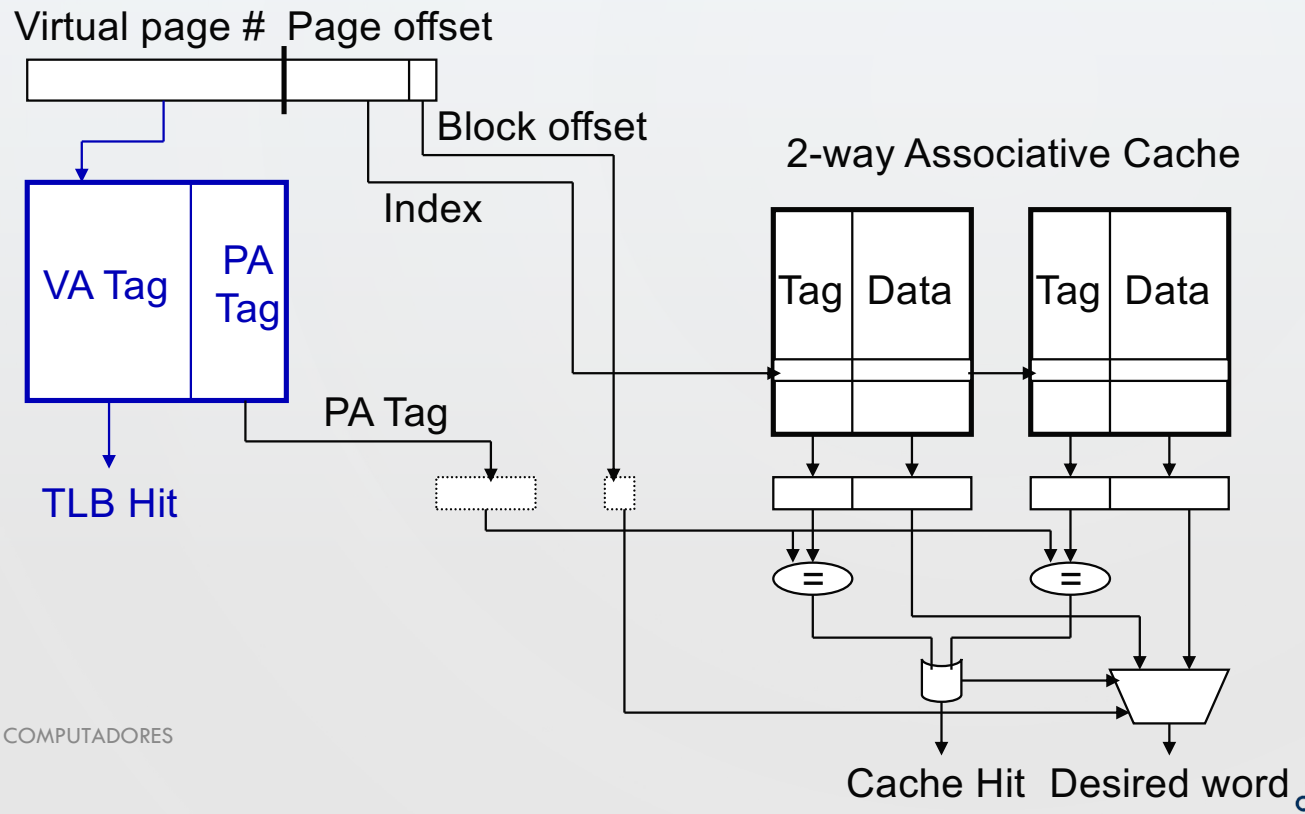
TLB – Page Table – Cache

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	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



Virtual Memory Summary

- Virtual memory increases **capacity**
- A subset of virtual pages in physical memory
- **Page table** maps virtual pages to physical pages – address translation
- A **TLB** speeds up address translation
- Different page tables for different programs provides **memory protection**

Memory Protection

- **Multiple processes** (programs) run at once
- Each process has its **own page table**
- Each process can use **entire virtual address space**
- A process can only access a **subset of physical pages**: those mapped in its own page table