Arquitectura de Computadores

Capítulo 3.
Organización y Estructura de la Memoria: Cachés y Memoria Virtual

Based on the original material of the book:
D.A. Patterson y J.L. Hennessy "Computer Organization and Design: The Hardware/Software Interface" 4th edition.

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Profesores:

G130 y 131: Gustavo Sutter

G136: Francisco J. Gómez Arribas

G135 y 139: Iván González Martínez

Memory Technology

- Static RAM (SRAM)
  - 0.5ns – 2.5ns, $2000 – $5000 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, $20 – $75 per GB
- Magnetic disk
  - 5ms – 20ms, $0.20 – $2 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

Principle of Locality

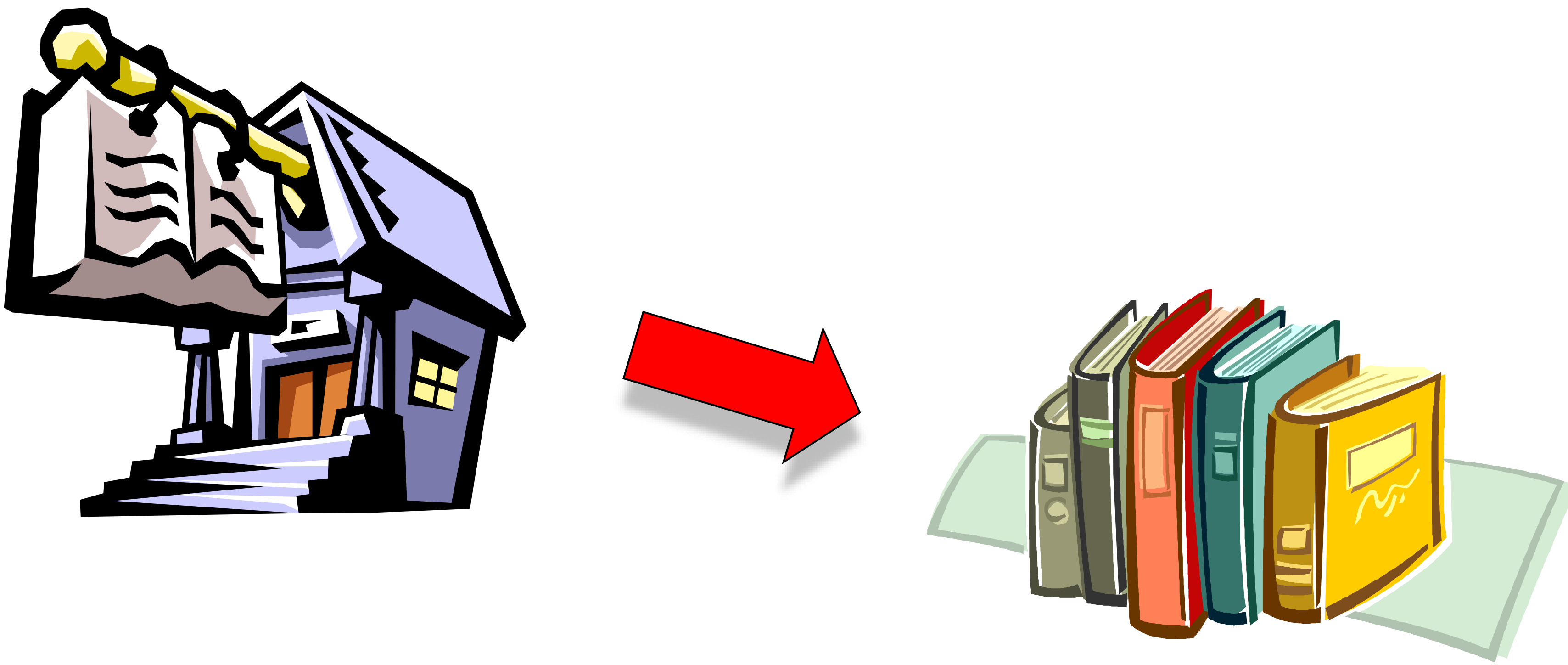
- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - **Cache memory** attached to CPU

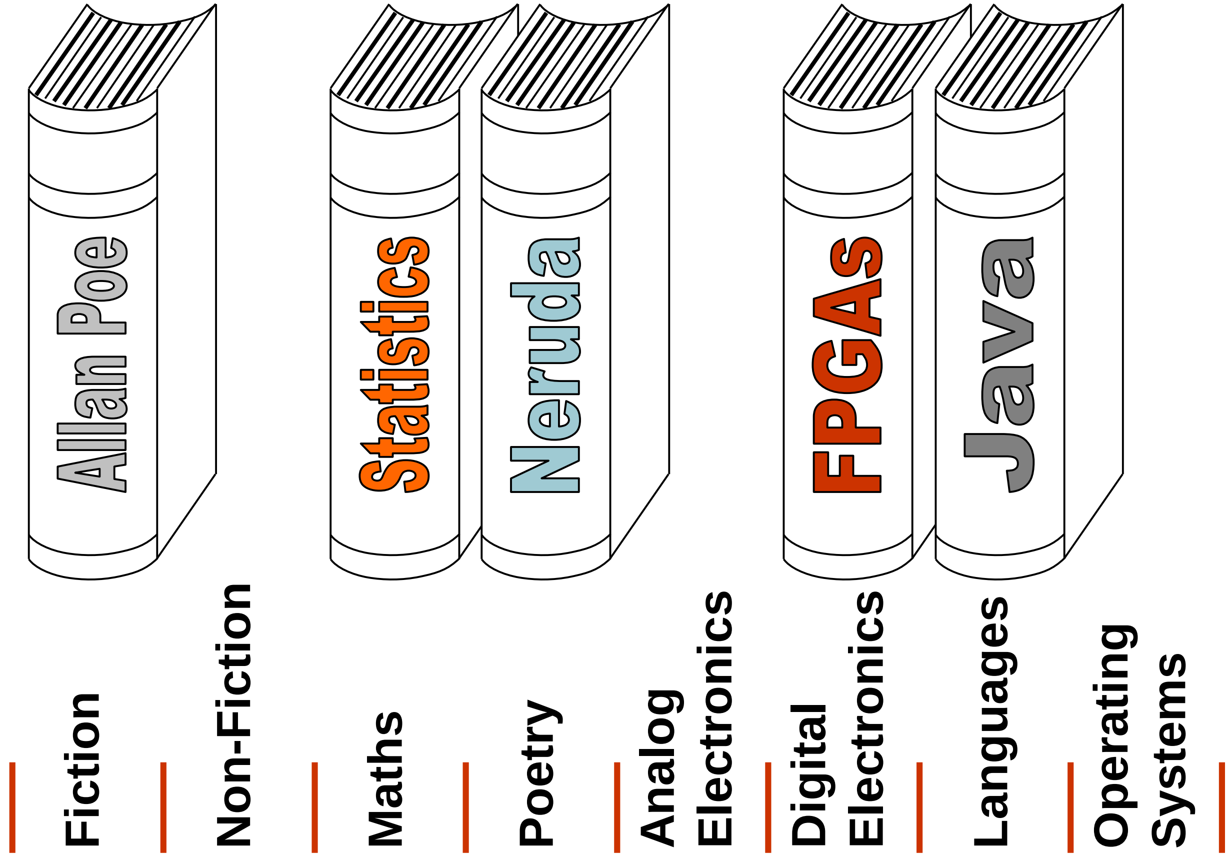
What is a Cache Memory?

- A cache memory is like having a bookshelf in your room instead going to the library



Direct Mapped

- A 8-place direct mapped bookshelf is the one where each place is dedicated to only one theme:

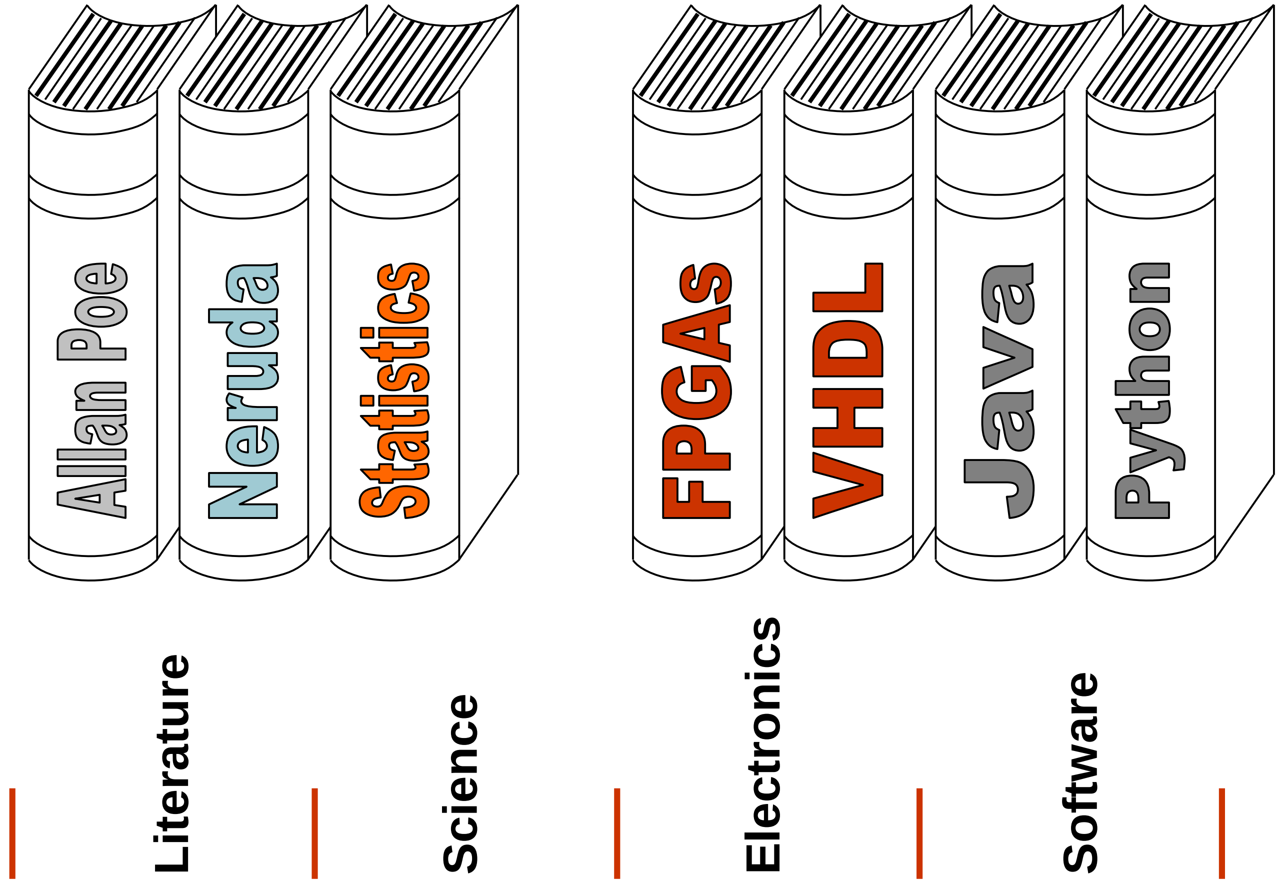


Fully Associative

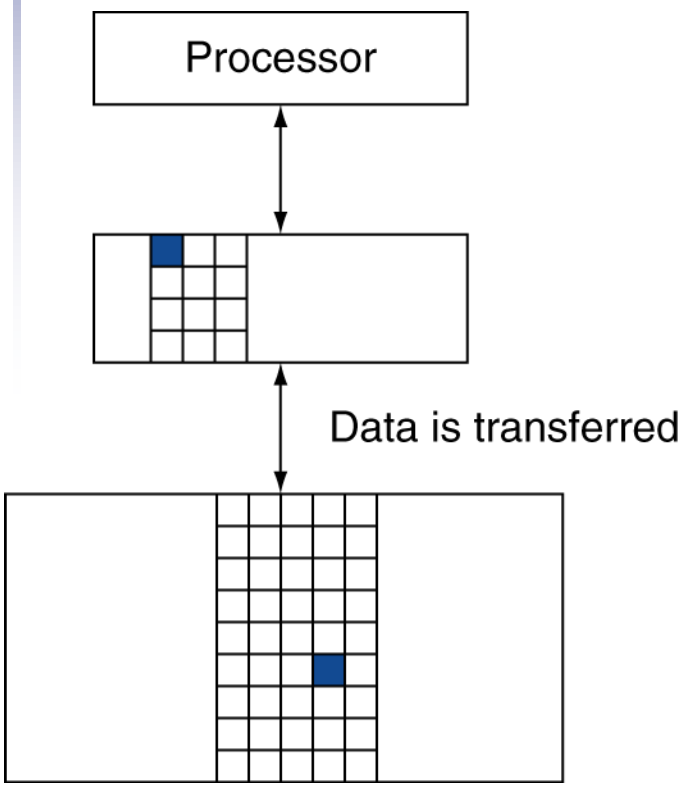
- A 8-place fully associative bookshelf is the one where you can store any book at any place, with a complete freedom:

Set Associative

- A 8-place 2-way set associative bookshelf is the one where each two place are dedicated to one theme:



Memory Hierarchy Levels

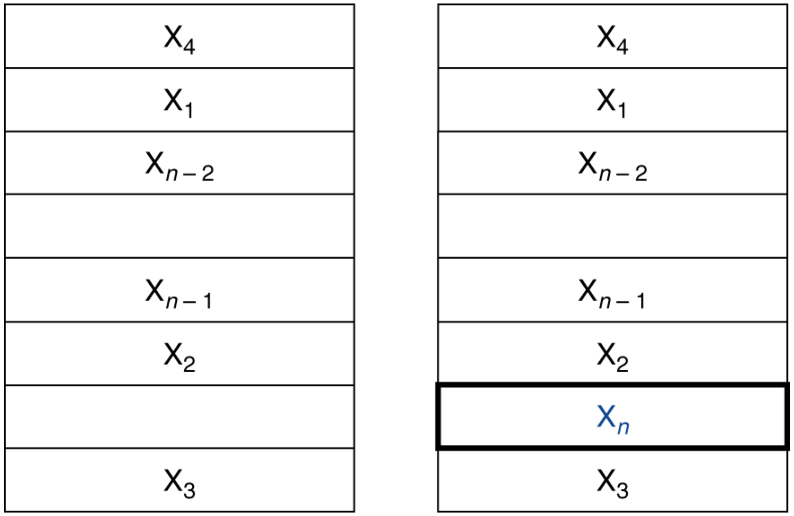


- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level (t_c)
  - Hit: access satisfied by upper level
    - Hit ratio H: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty (t_B)
    - Miss ratio: misses/accesses = 1 – hit ratio = (1-H)
  - Then accessed data supplied from upper level

$$t_{access} = t_c + (1-H)\, t_B$$

Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$



a. Before the reference to X_n b. After the reference to X_n

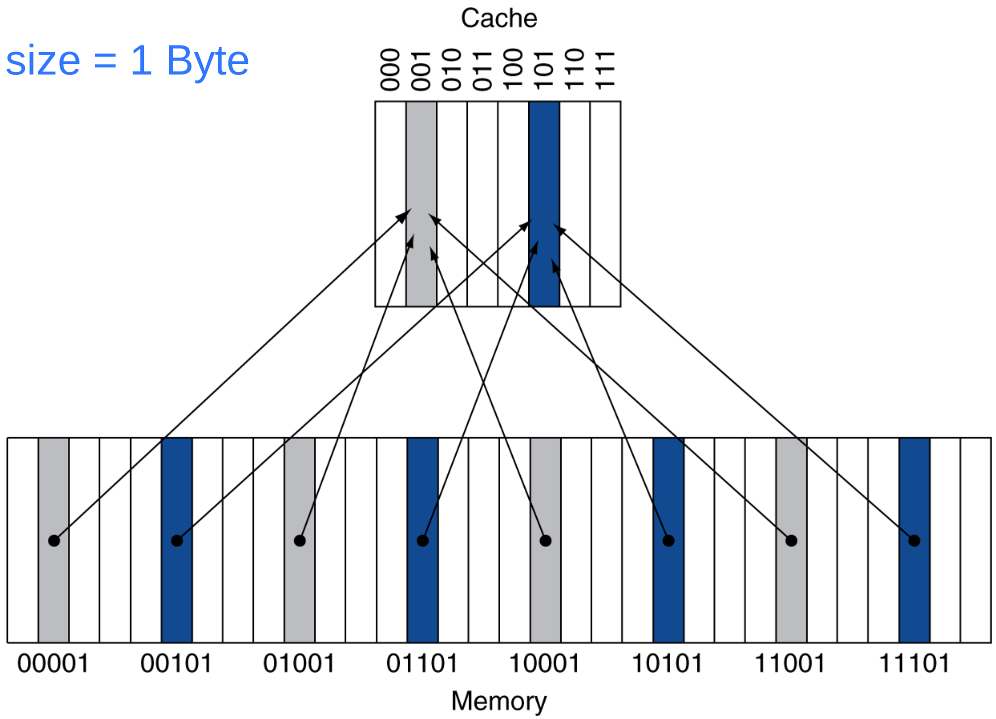
- How do we know if the data is present?
- Where do we look?

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice

Location Index (*)=(Block address) modulo (#Blocks in cache)

(*) Block size = 1 Byte



- #Blocks is a power of 2
- Use low-order address bits

Tags and Valid Bits

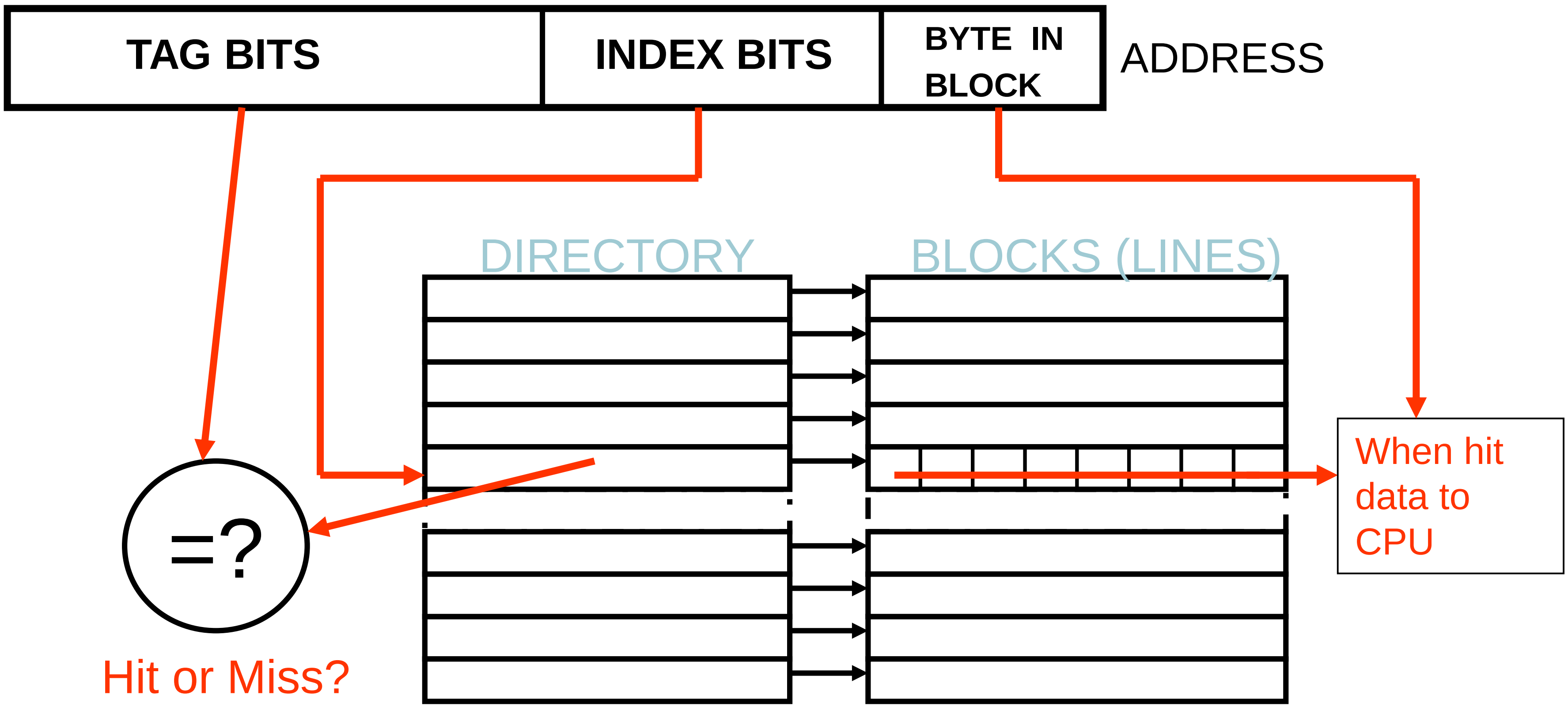
- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Large Block Size

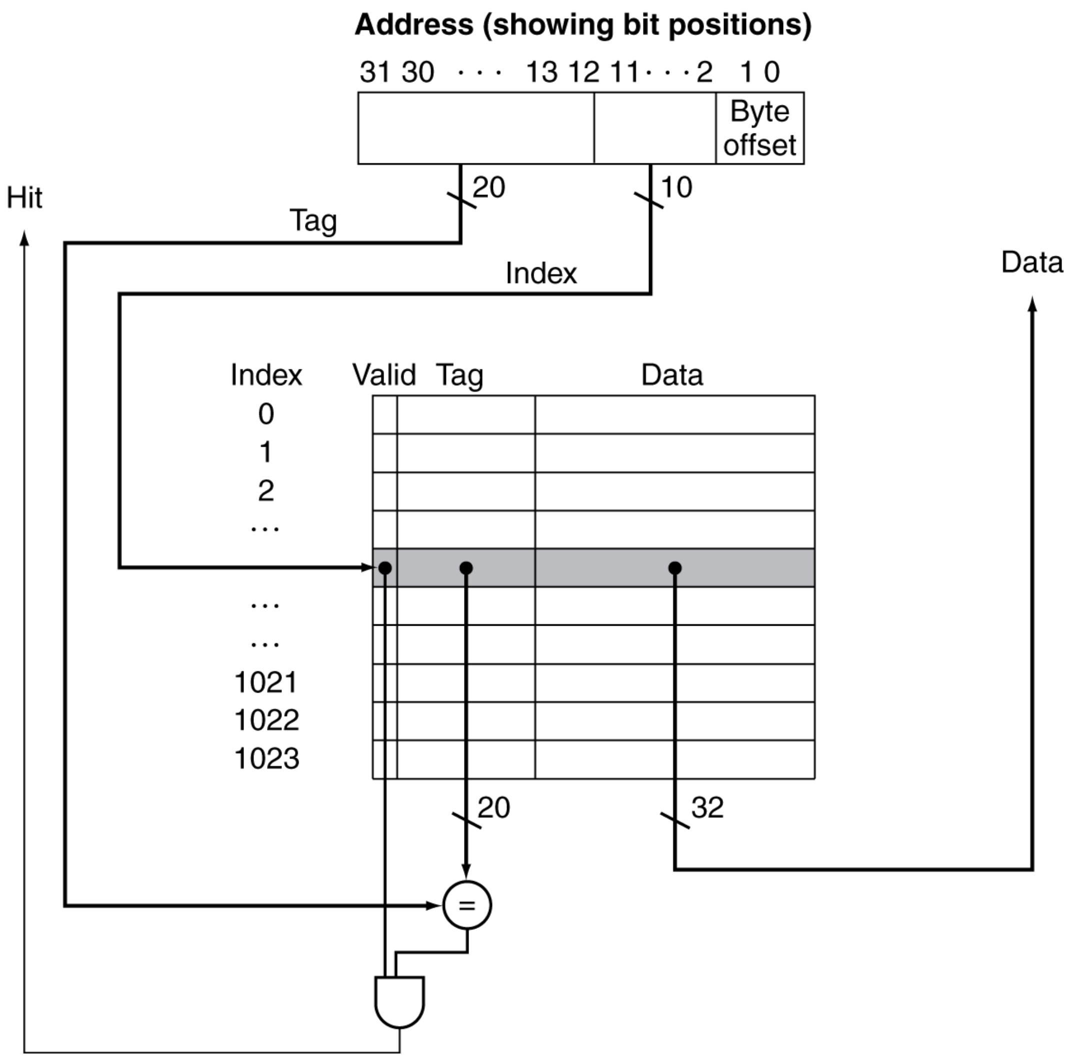
- Direct mapped Cache: 64 blocks, 16 bytes/block
  - To what block location in cache (index) does address 1200 map?
- Block address = $\lfloor 1200/16 \rfloor = 75$
- Block location (index) = 75 modulo 64 = 11

31 … 10	9 … 4	3 … 0
Tag	Index	Offset
22 bits	6 bits	4 bits

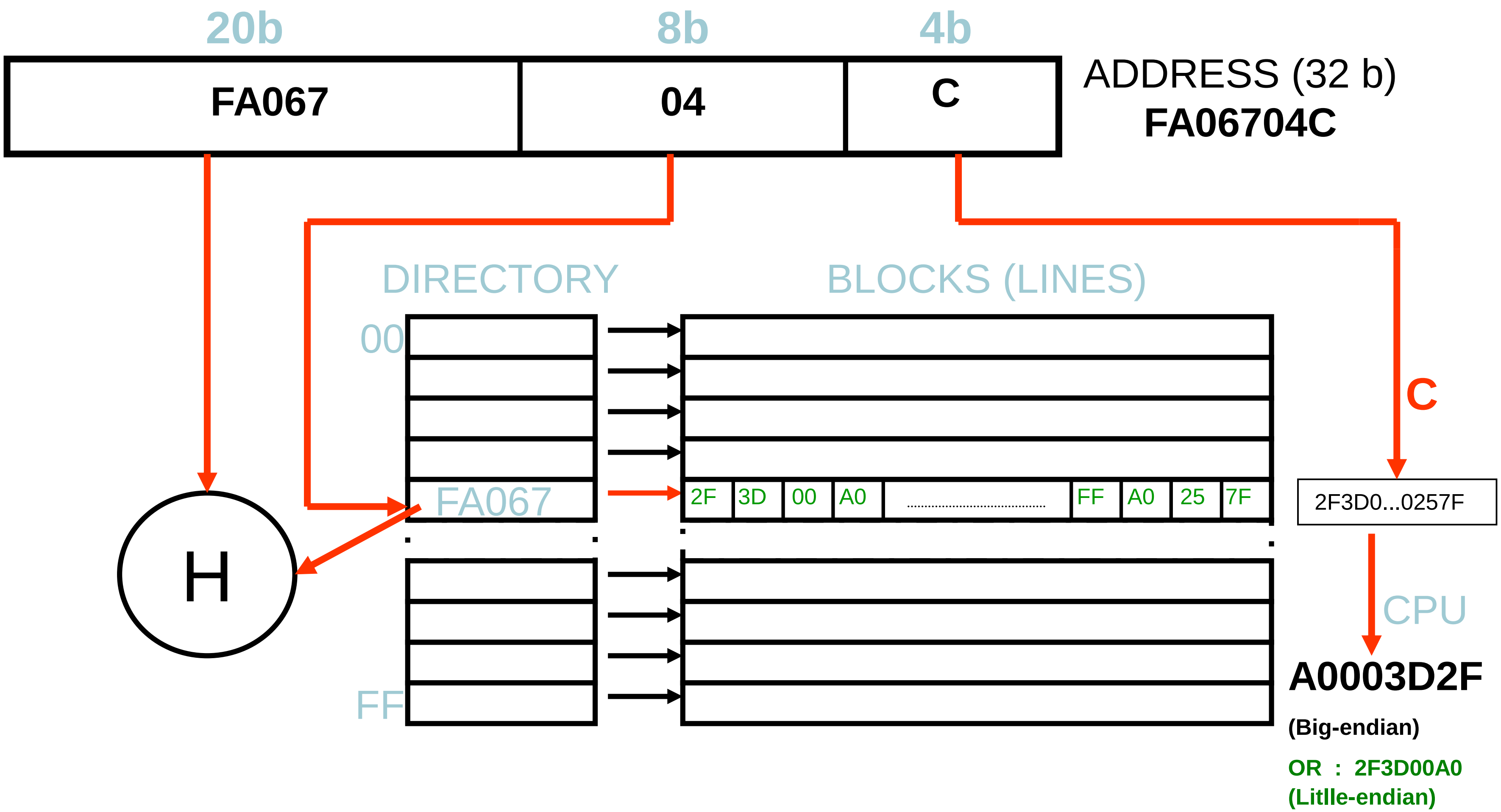
Direct-Mapped Cache



Address Subdivision



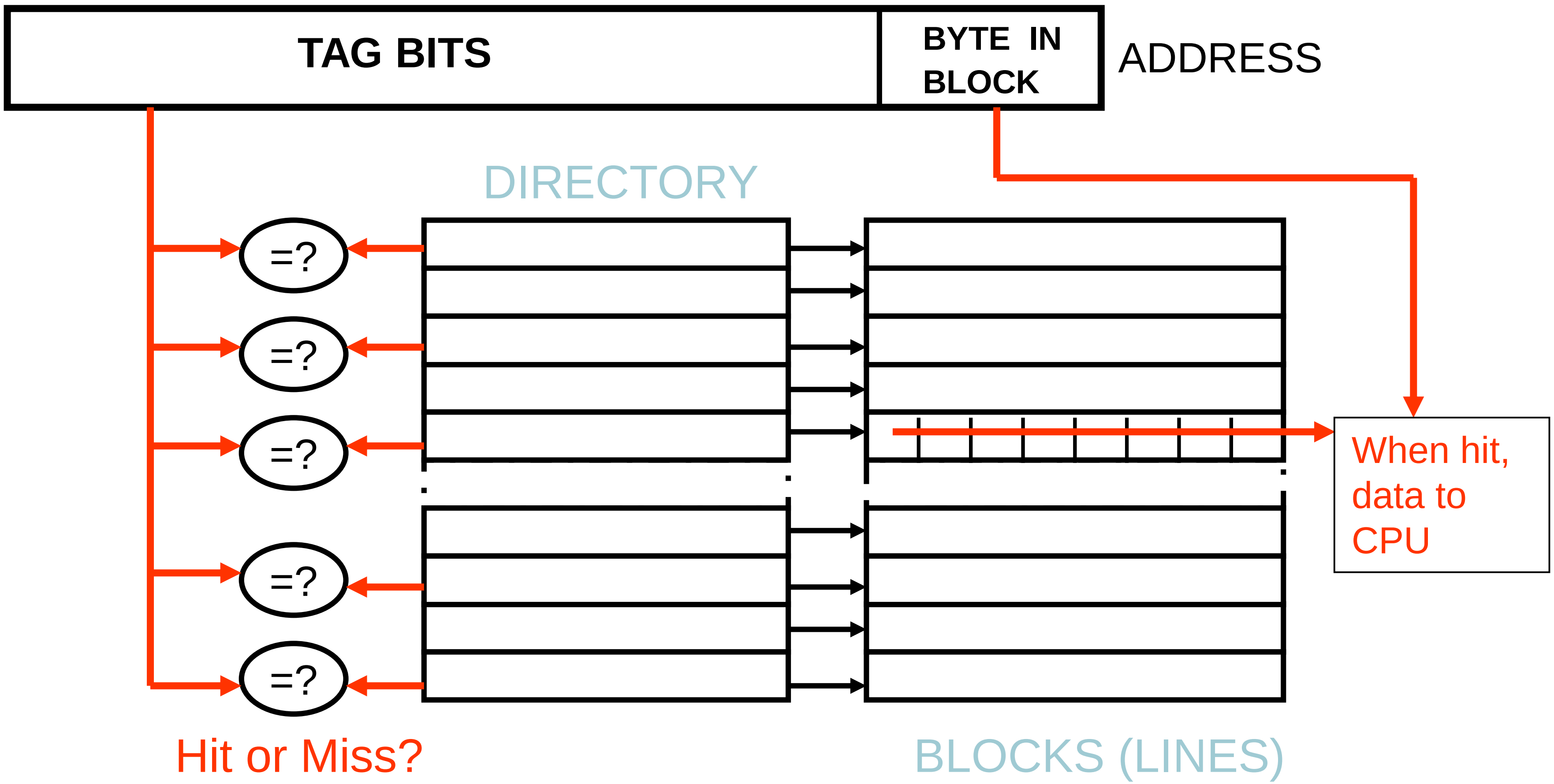
Example:
4KB direct-mapped cache with 16 Bytes/Block



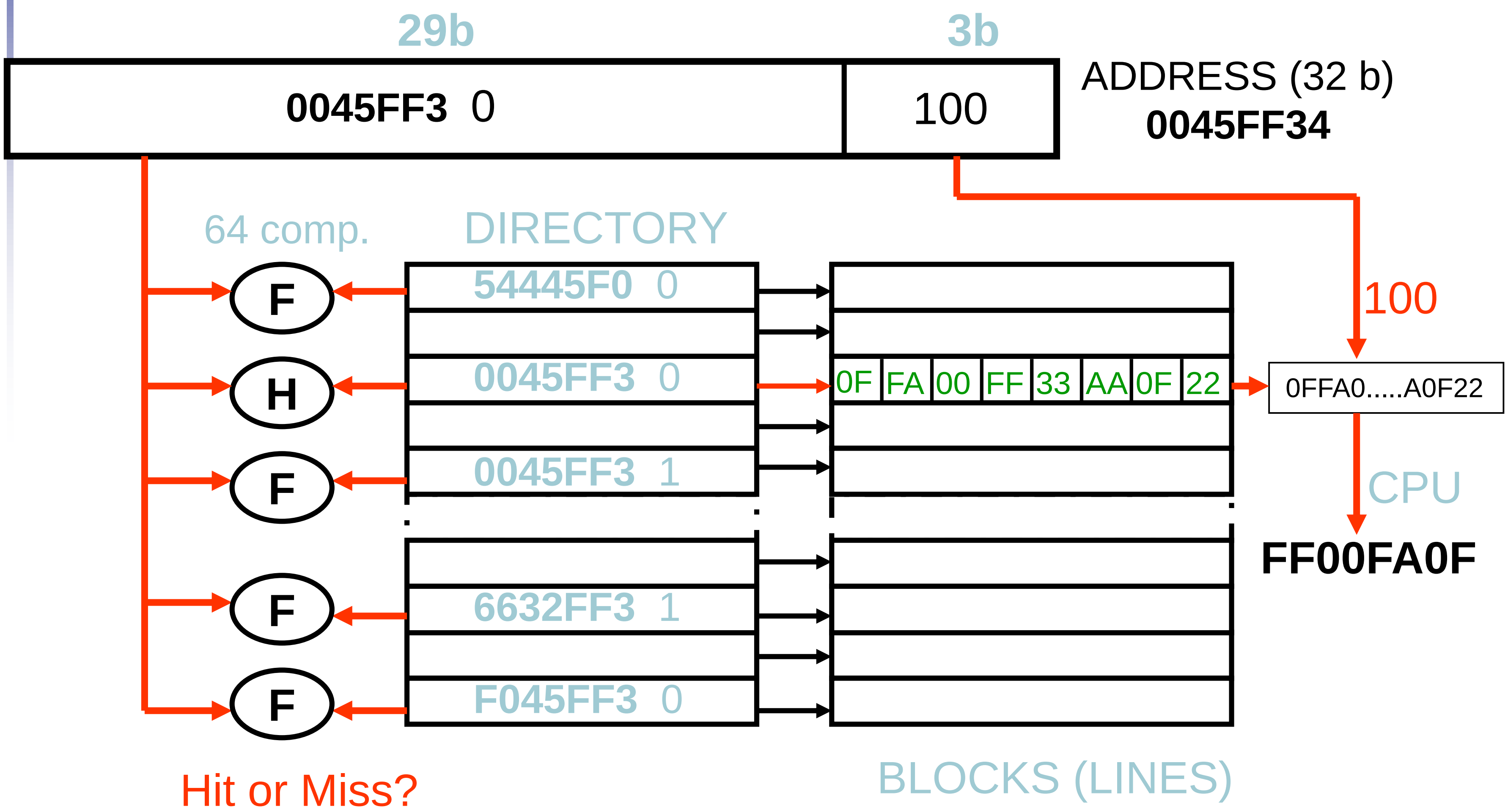
Associative Caches

- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block address) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

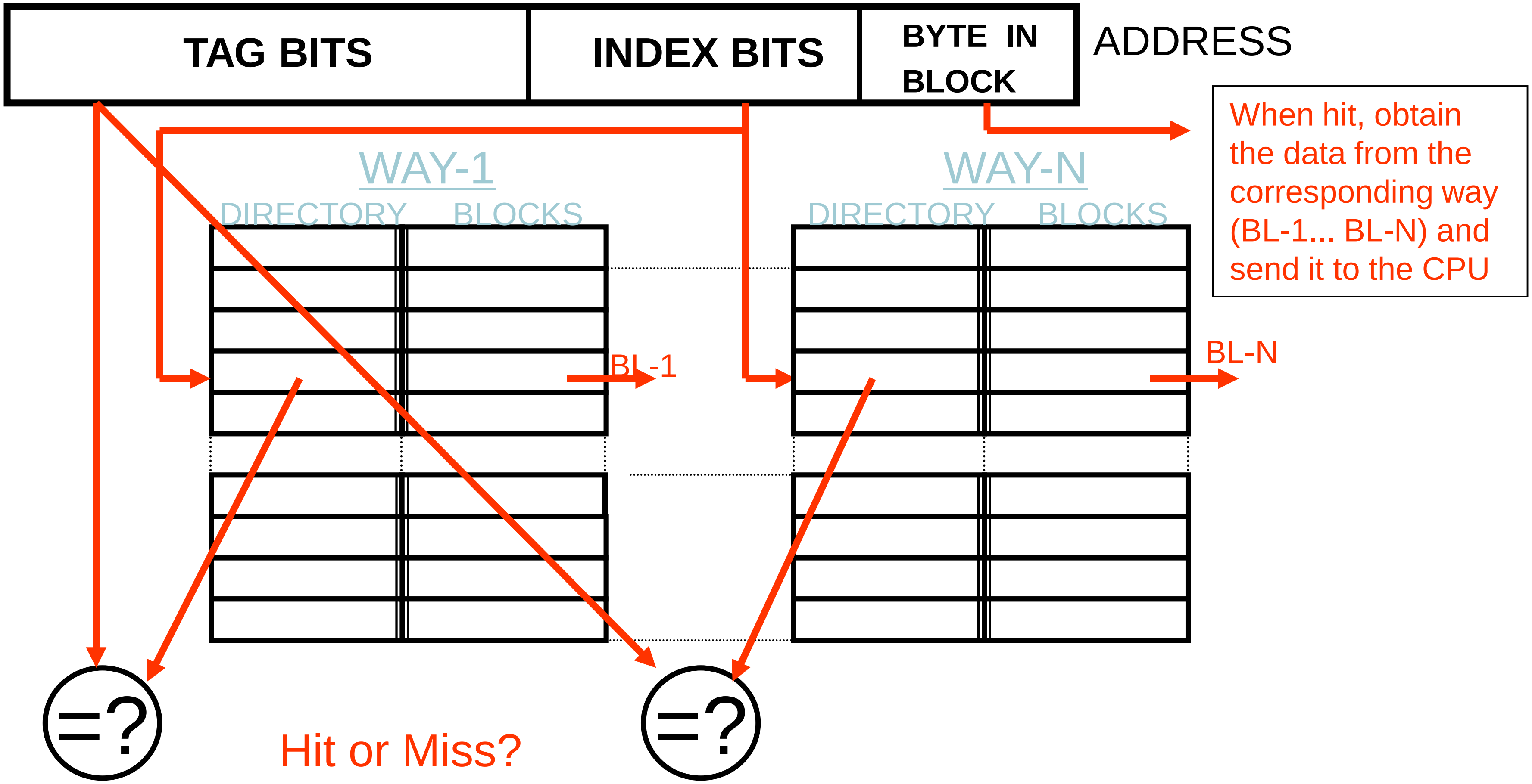
Fully-Asociative Cache



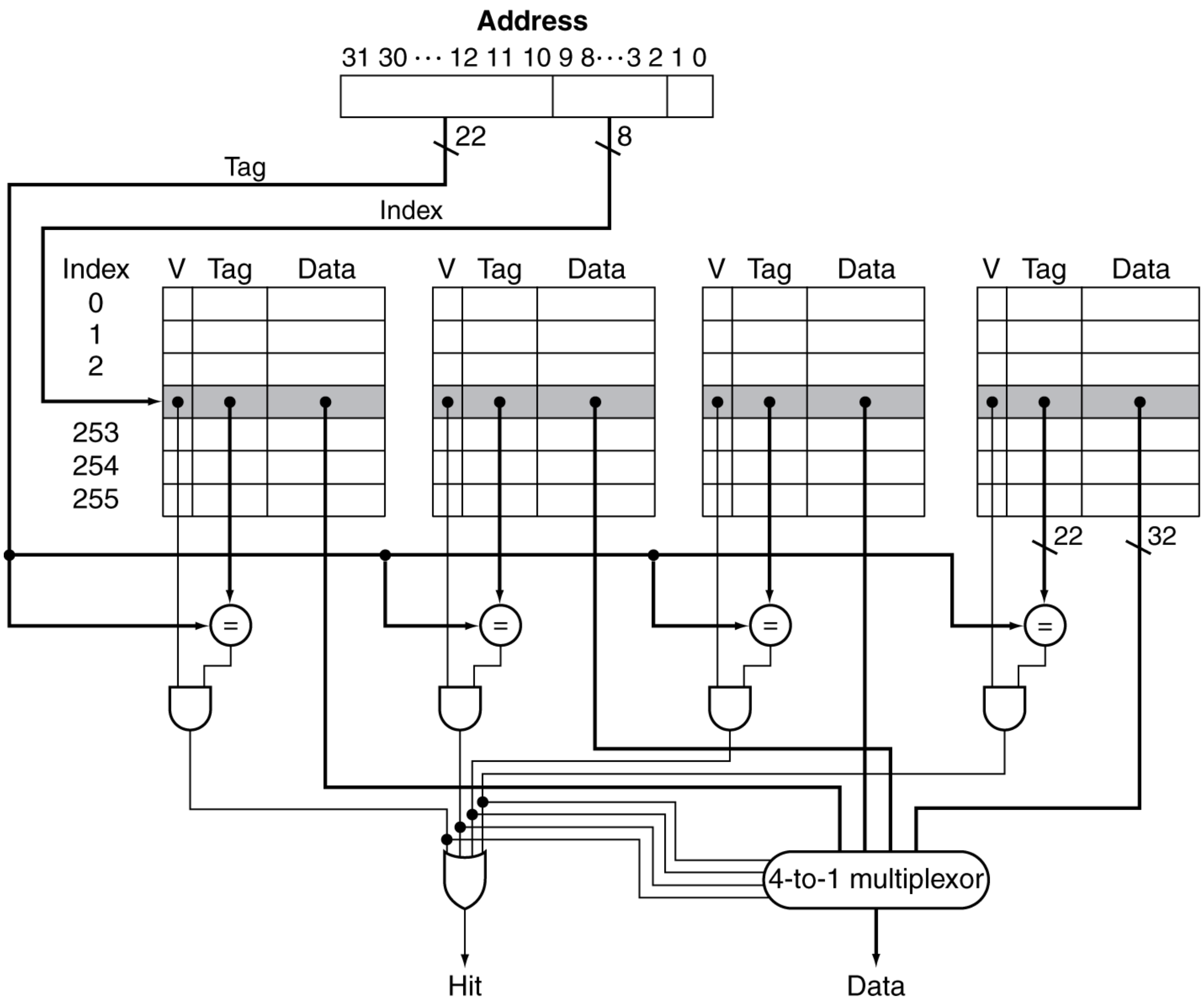
Example:
512 bytes fully-associative cache with 8 bytes/block



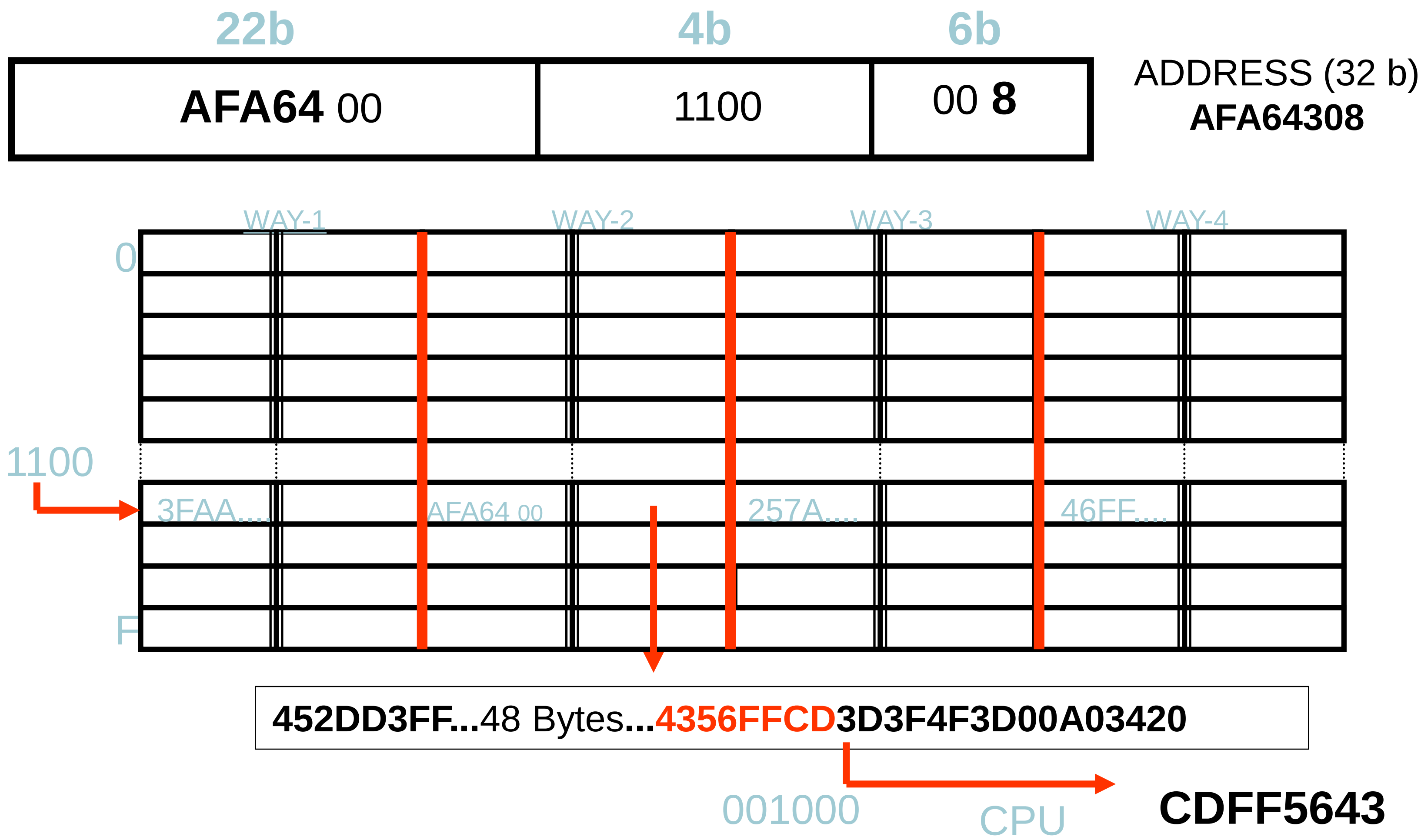
Set (N-Way) Associative Cache



Set Associative Cache Organization



Example:
4KB, 4-Way associative cache with 64 Bytes/Block



How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Sources of Misses

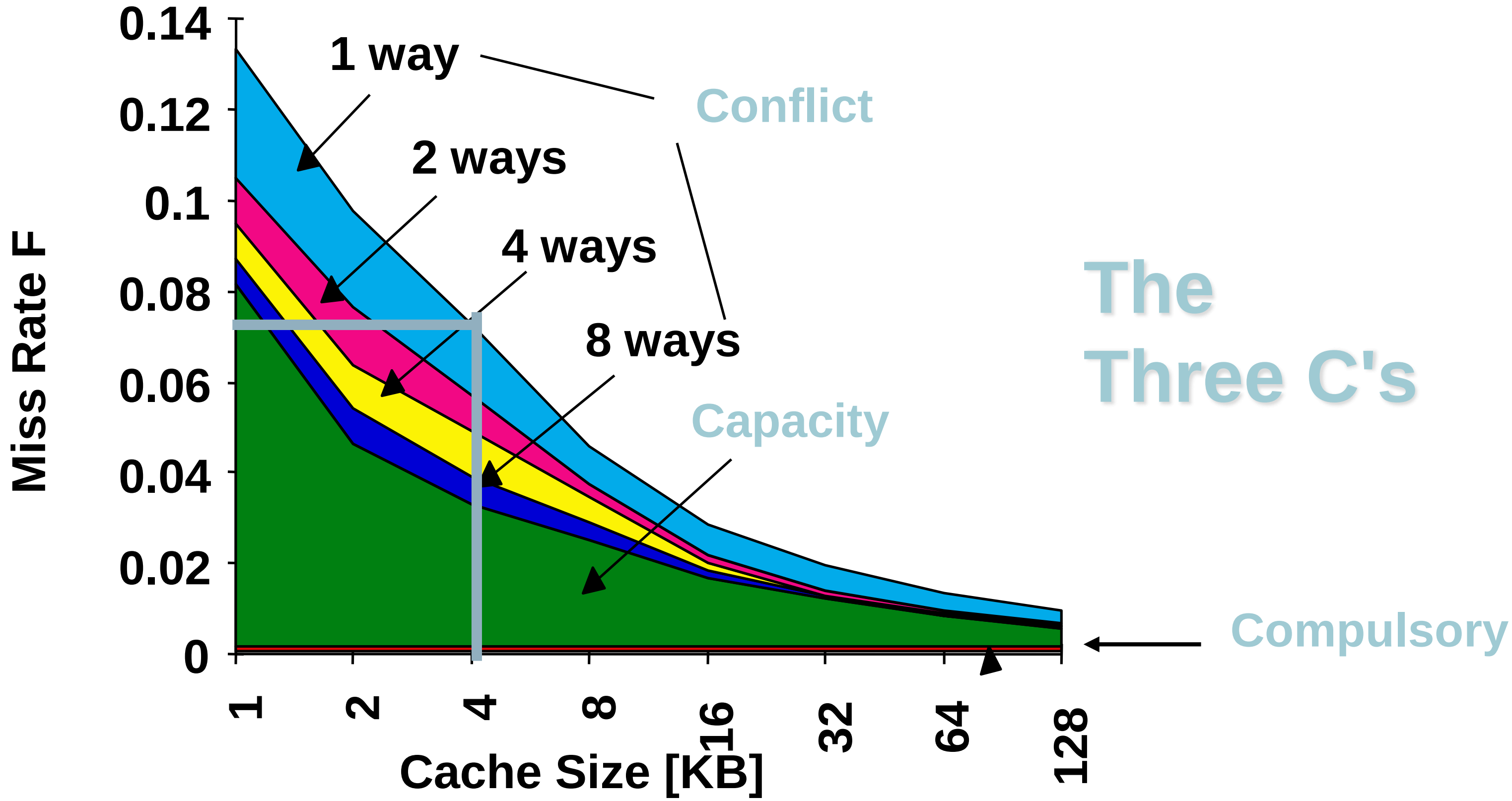
- Compulsory misses (aka cold start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully associative cache of the same total size

Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks $\Rightarrow$ fewer of them
    - More competition $\Rightarrow$ increased miss rate
  - Larger blocks $\Rightarrow$ pollution
- Larger miss penalty
  - Can override benefit of reduced miss rate

Reducing Miss Rates

- 2:1 Cache Rule: “The miss rate of a size N direct-mapped cache ≈ the one of a 2-way and size N/2 set-associative cache”



Cache Design Trade-offs

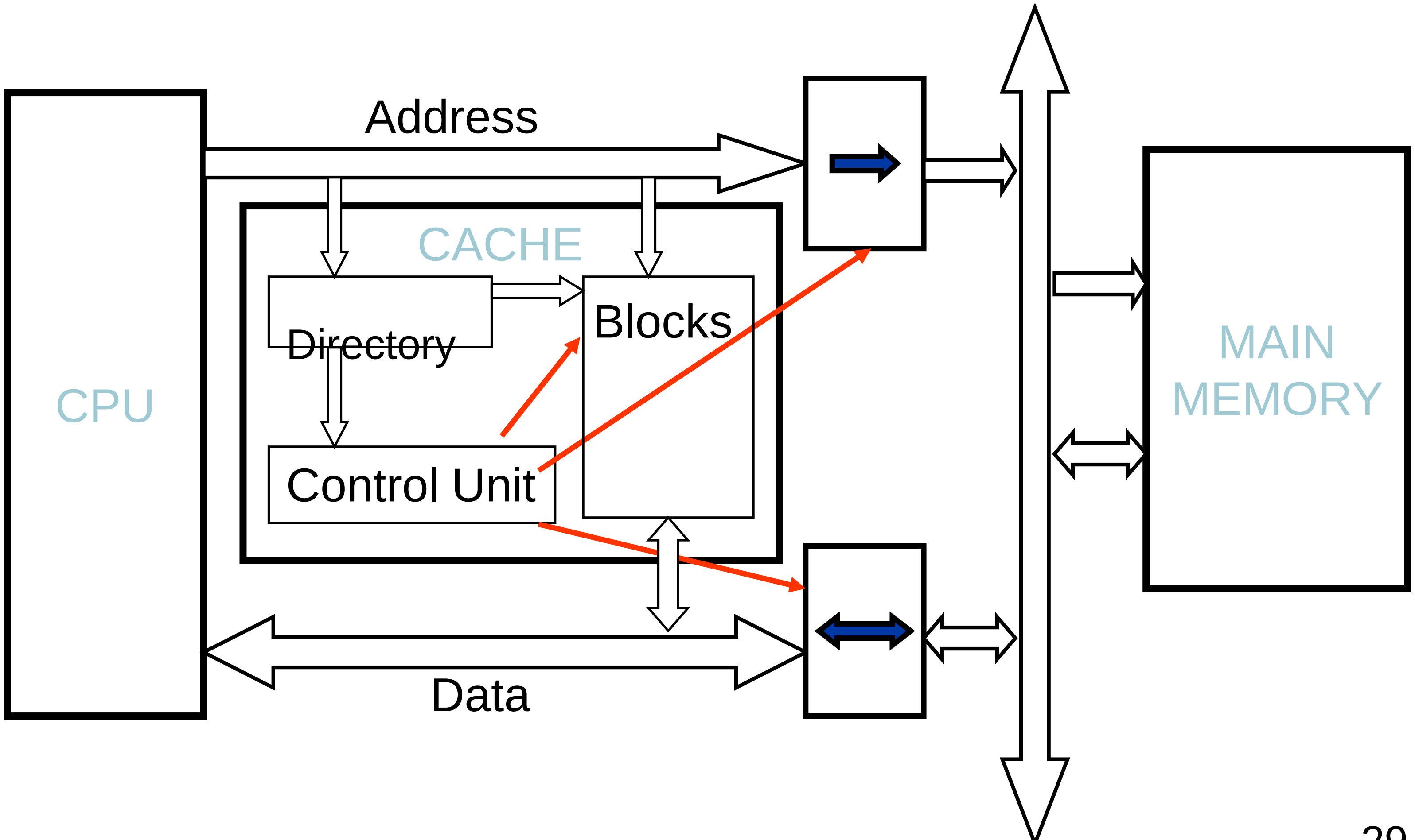
Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

The Memory Hierarchy

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy

Cache memories

Principles of operation



Cache memories

Main Blocks

- Cache Directory
  - Depending on the cache organization, it can be implemented with SRAM or CAM (*Content Addressable Memory).*
  - Used to know if the address asked by the processor is located in the cache. If there is a *hit,* it returns the position where the data is stored in the cache
- Cache blocks
  - Implemented with SRAM
  - Each line (also called block) contains a certain number of bytes, which are accessed each time a cache hit occurs
- Control unit
  - Generates the control signals for the cache (state machine).

Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Finding a Block

Associativity	Location method	Tag comparisons
Direct mapped	Index	1
n-way set associative	Set index, then search entries within the set	n
Fully associative	Search all entries	#entries
	Full lookup table	0

- Hardware caches
  - Reduce comparisons to reduce cost

Block Placement

- Determined by associativity
  - Direct mapped (1-way associative)
    - One choice for placement
  - n-way set associative
    - n choices within a set
  - Fully associative
    - Any location
- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

Replacement Policy

- Choice of entry to replace on a miss
  - Direct mapped: no choice
  - Set associative
    - Prefer non-valid entry, if there is one
    - Otherwise, choose among entries in the set
  - Least-recently used (LRU)
    - Choose the one unused for the longest time
      - Simple for 2-way, manageable for 4-way, too hard beyond that
  - Random
    - Gives approximately the same performance as LRU for high associativity

Replacement Policy

Example: Cache N-A4W: counter 2bits (LRU)

Which is the LRU block?

BLOCK REFERENCED	C_{B0}	C_{B1}	C_{B2}	C_{B3}	STATE	LRU
Initial state	0	0	0	0	Empty blocks	B0,B1,B2,B3
Error cache access	0	1	1	1	B0 full	B1,B2, B3
Error cache access	1	0	2	2	B0,B1 full	B2,B3
Hit in B0	0	1	2	2	B0,B1 full	B2,B3
Error cache access	1	2	0	3	B0,B1,B2 full	B3
Error cache access	2	3	1	0	All blocks full	B1
Hit in B1	3	0	2	1	All blocks full	B0
Error cache access	0	1	3	2	All blocks full	B2
Error cache access	1	2	0	3	All blocks full	B3

Write Policy

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state

Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = $1 + 0.1 \times 100 = 11$
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Write-Back

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first

Write Allocation

- What should happen on a write miss?
- Alternatives for write-through
  - Allocate on miss: fetch the block
  - Write around: don’t fetch the block
    - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
  - Usually fetch the block

Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses
- With simplifying assumptions:

$$\text{Memory stall cycles}$$

$$= \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: $0.02 \times 100 = 2$
  - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 =2.72 times faster

Average Access Time

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

Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache

Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 400 cycles
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Performance ratio = 9/3.4 = 2.6

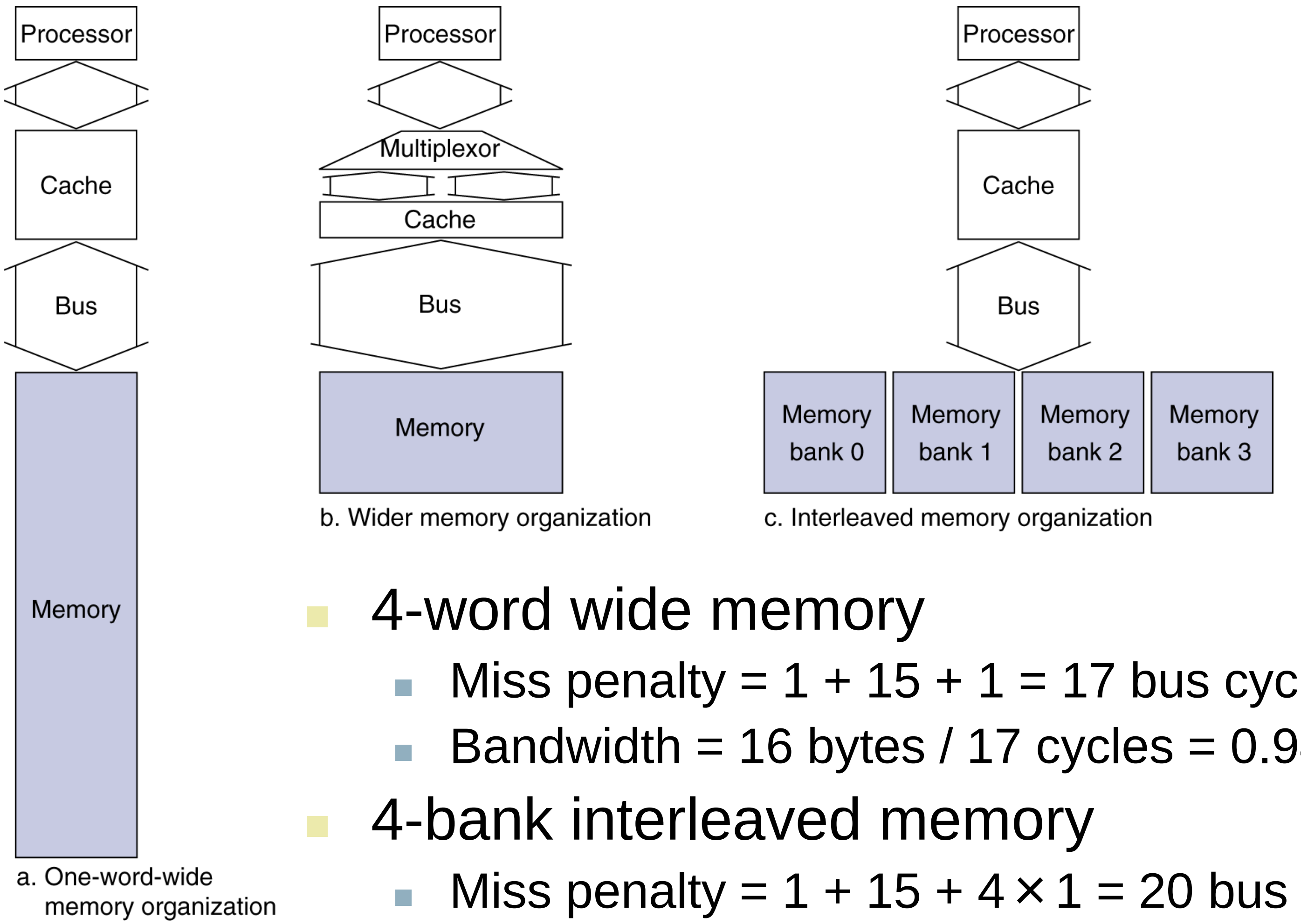
Multilevel Cache Considerations

- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

Main Memory Supporting Caches

- Use DRAMs for main memory
  - Fixed width (e.g., 1 word)
  - Connected by fixed-width clocked bus
    - Bus clock is typically slower than CPU clock
- Example cache block read
  - 1 bus cycle for address transfer
  - 15 bus cycles per DRAM access
  - 1 bus cycle per data transfer
- For 4-word block, 1-word-wide DRAM
  - Miss penalty = $1 + 4 \times 15 + 4 \times 1$ = 65 bus cycles
  - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle

Increasing Memory Bandwidth



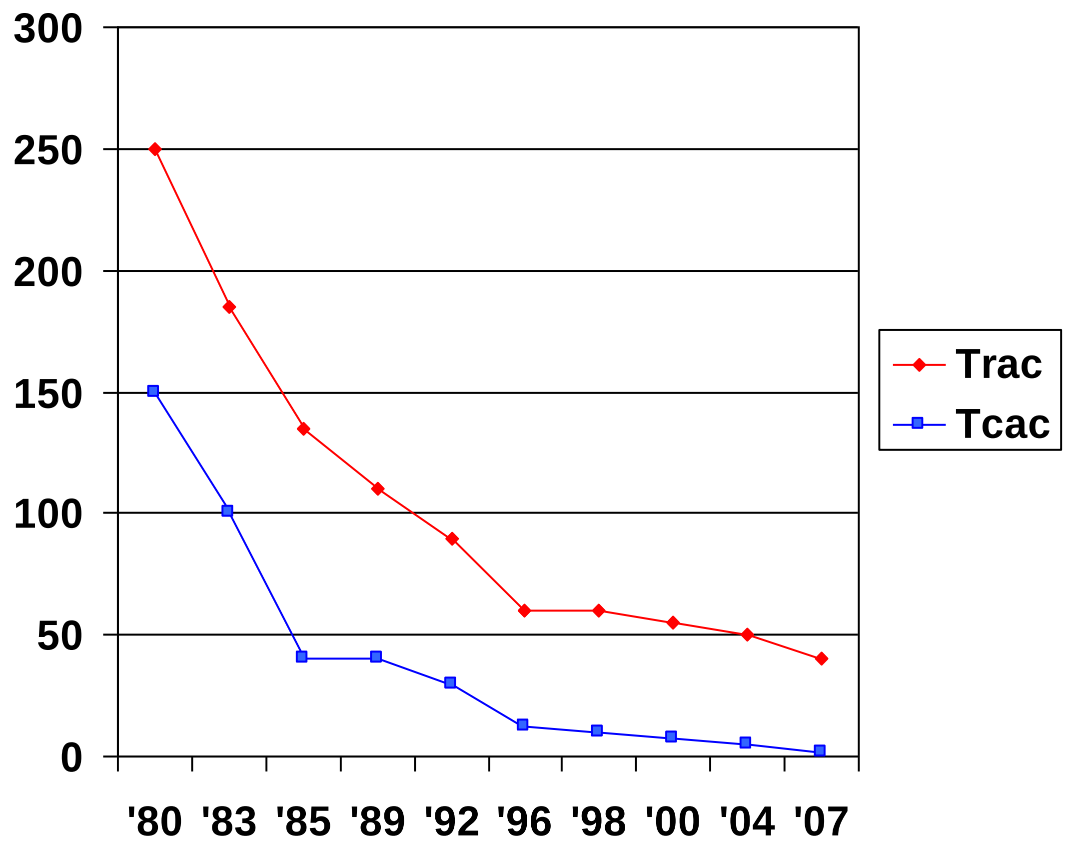
- 4-word wide memory
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle
- 4-bank interleaved memory
  - Miss penalty = 1 + 15 + 4 × 1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

Advanced DRAM Organization

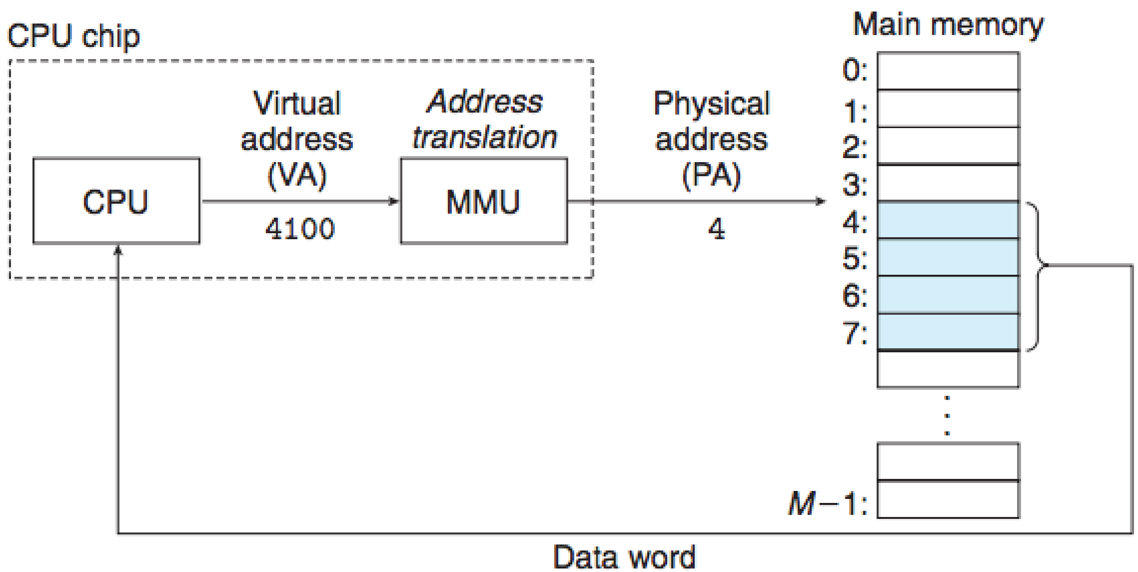
- Bits in a DRAM are organized as a rectangular array
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges
- Quad data rate (QDR) DRAM
  - Separate DDR inputs and outputs

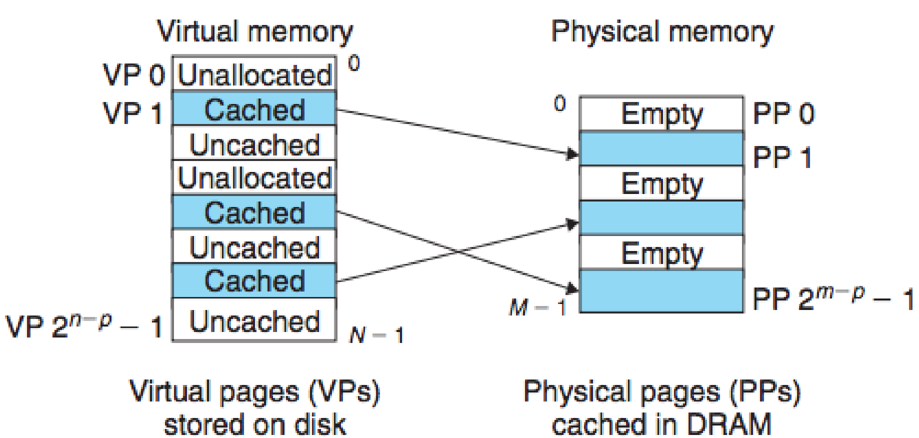
DRAM Generations

Year	Capacity	$/GB
1980	64Kbit	$1500000
1983	256Kbit	$500000
1985	1Mbit	$200000
1989	4Mbit	$50000
1992	16Mbit	$15000
1996	64Mbit	$10000
1998	128Mbit	$4000
2000	256Mbit	$1000
2004	512Mbit	$250
2007	1Gbit	$50



Virtual Memory



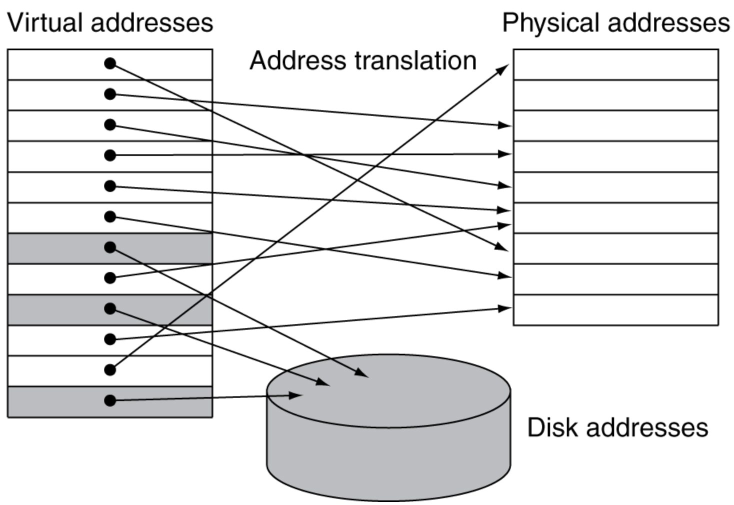


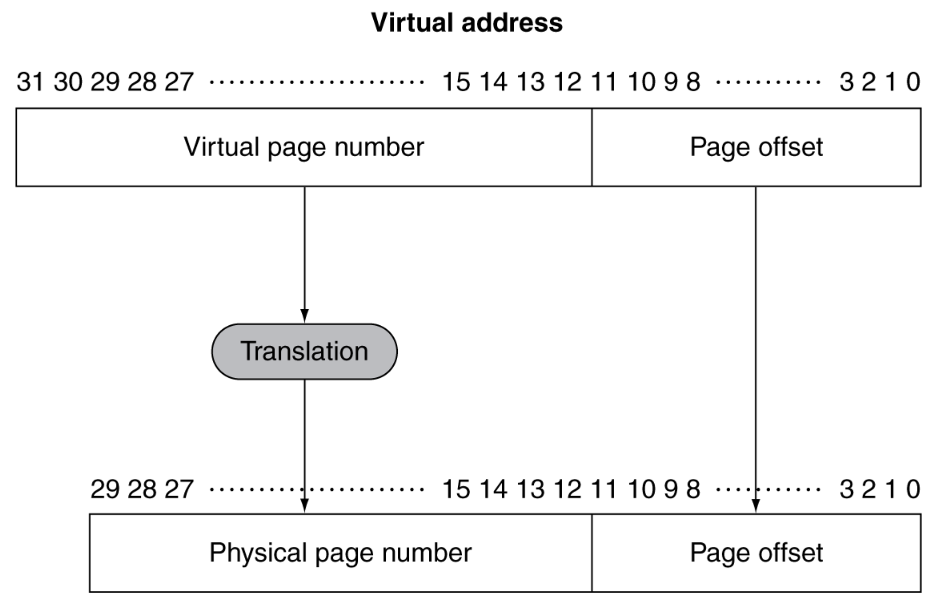
Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware (MMU) and the operating system (OS)
- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
  - VM “block” is called a page
  - VM translation “miss” is called a page fault

Address Translation

- Fixed-size pages (e.g., 4K)





- Variable-size segments
- Segments with fixed-size pages

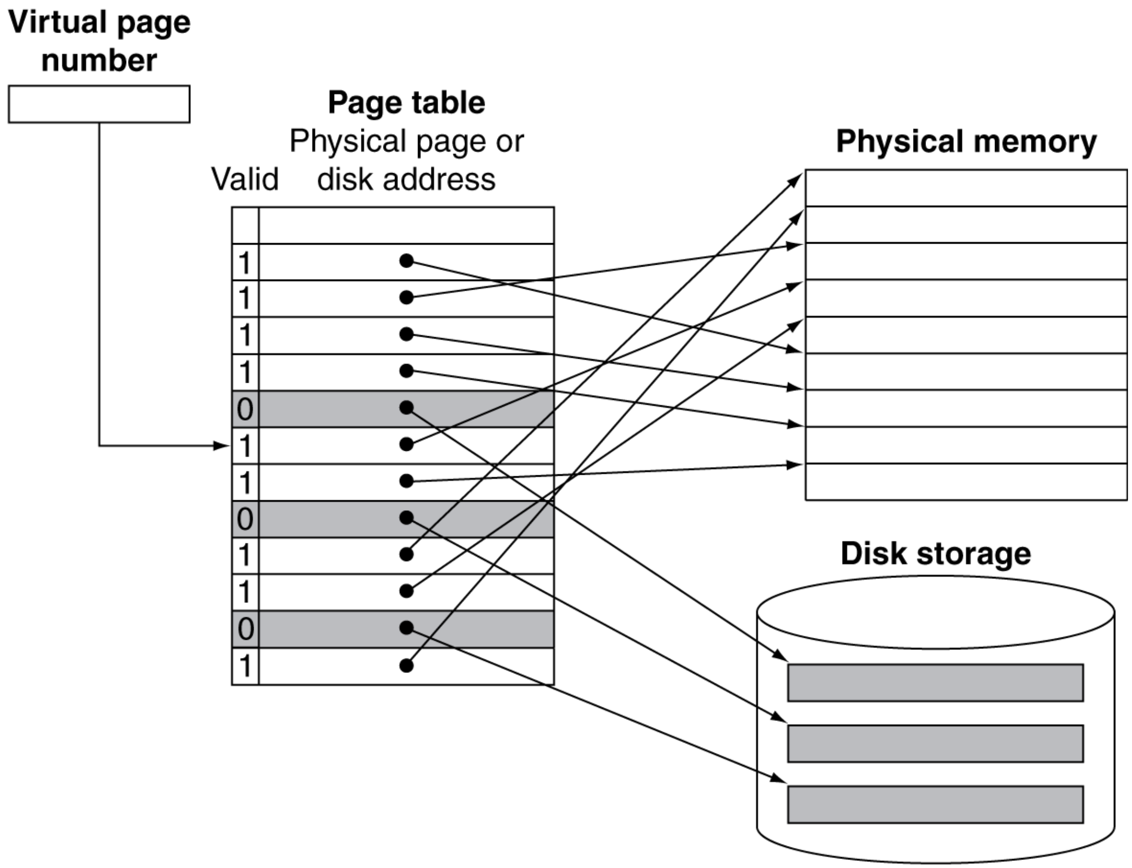
Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code
- Try to minimize page fault rate
  - Fully associative placement
  - Smart replacement algorithms

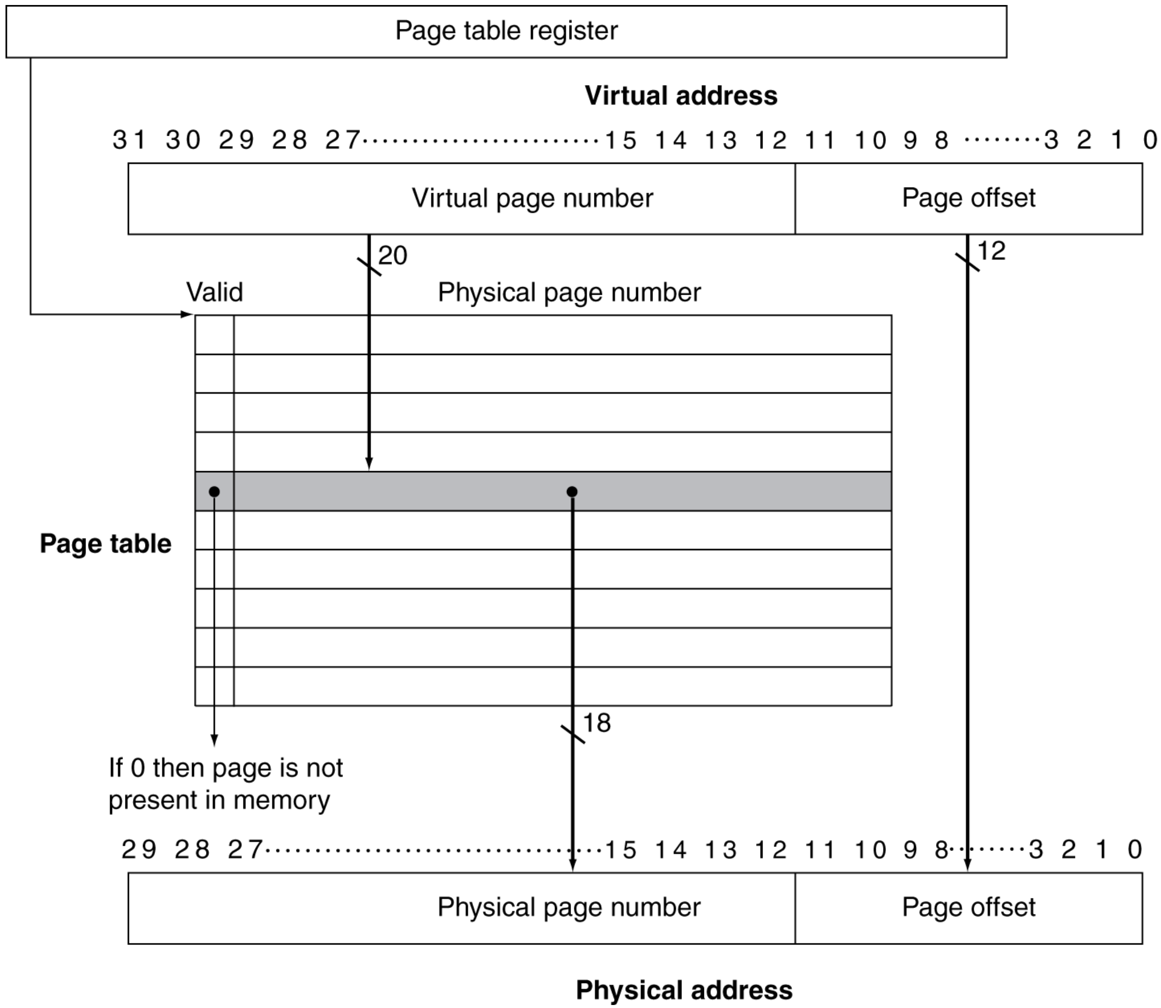
Page Tables

- Stores placement information
  - Array of page table entries (PTE), indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If page is not present
  - PTE (Page Translation Entry) can refer to location in swap space on disk

Mapping Pages to Storage

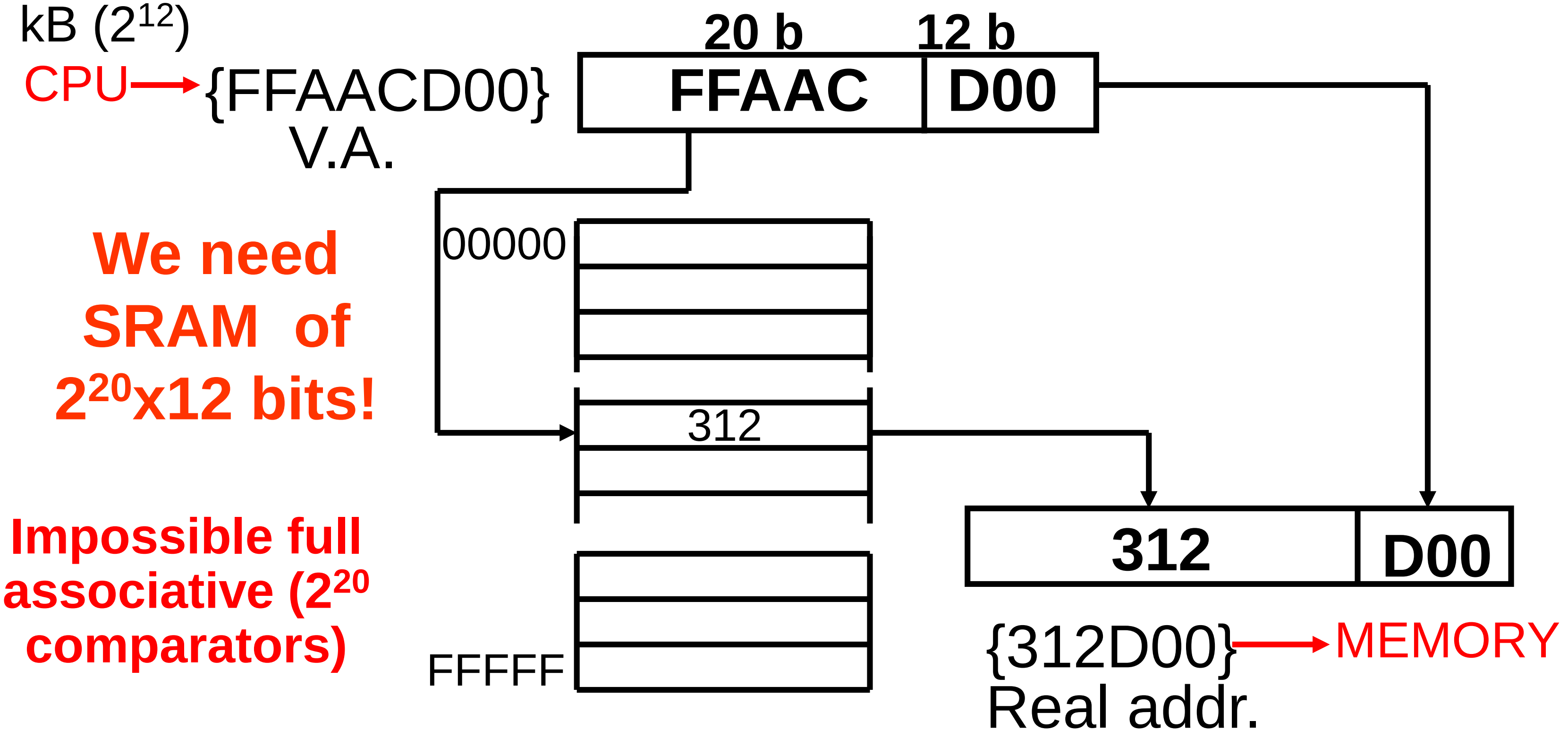


Translation Using a Page Table



Translation Using a Page Table

Example: Virtual memory: 4 GB (2^{32}), real: 16 MB (2^{24}). Page size: 4 kB (2^{12})



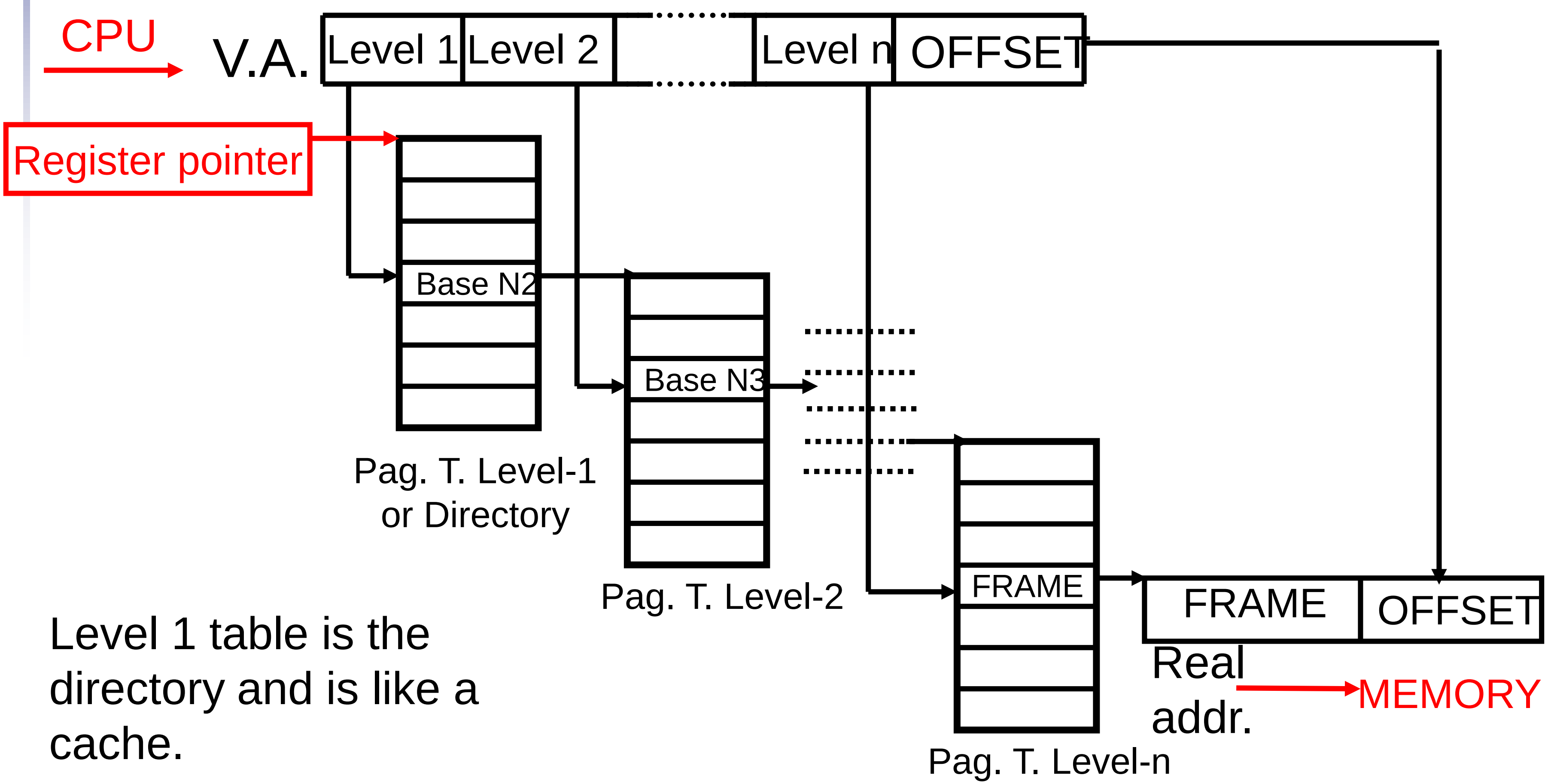
We need SRAM of 2^{20}x12 bits!

Impossible full associative (2^{20} comparators)

To reduce the page table size, it is built according to the process requirements.

Translation Using a Page Table

- Multi-level Page Table.
  - **To reduce the size of the page table (not all the sub-tables reside in memory)**

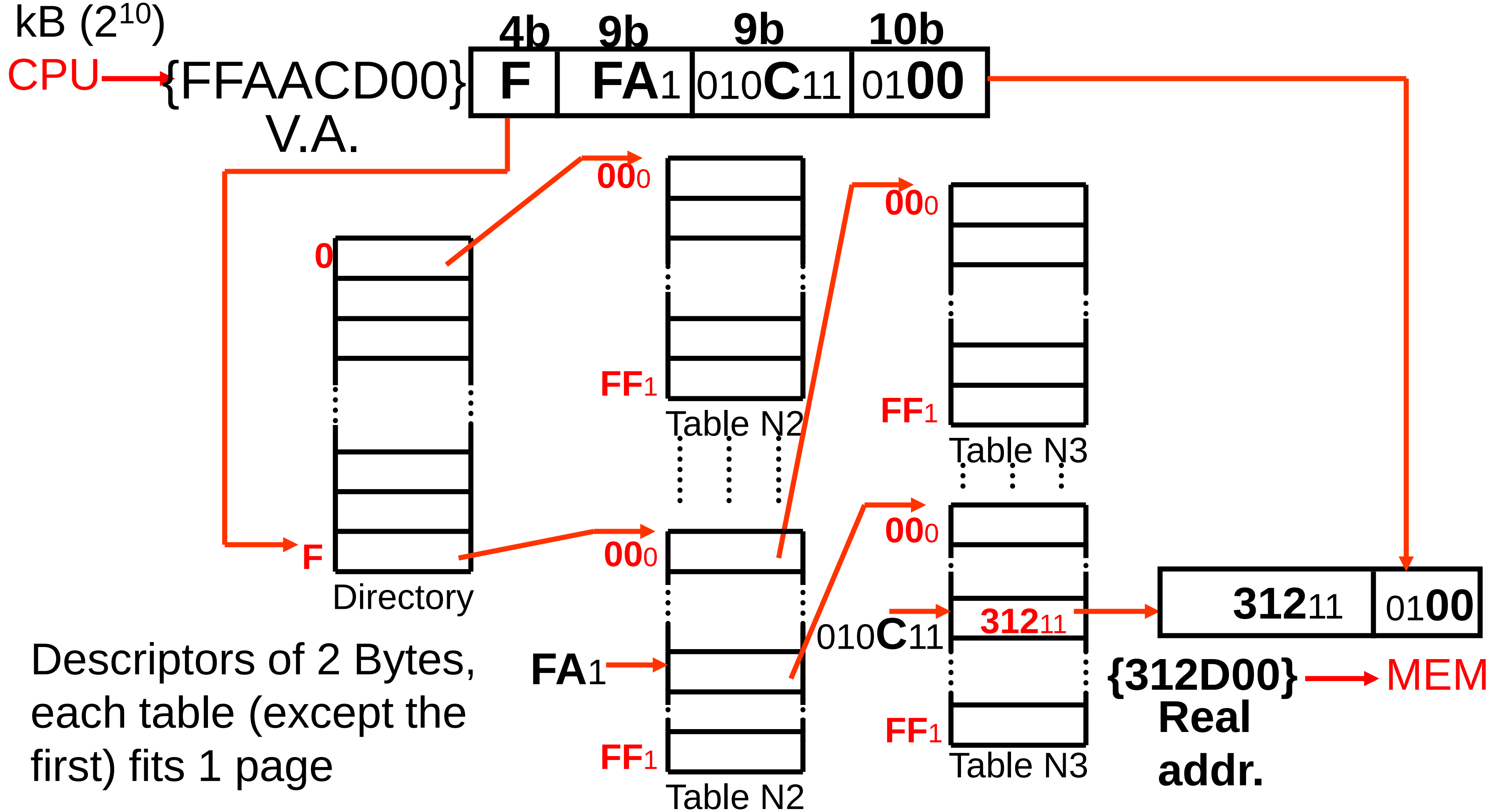


Level 1 table is the directory and is like a cache.

Translation Using a Page Table

Page Table with 3 levels

Example: Virtual Memory: 4 GB (2^{32}), Real: 16 MB (2^{24}). Page size: 1 kB (2^{10})

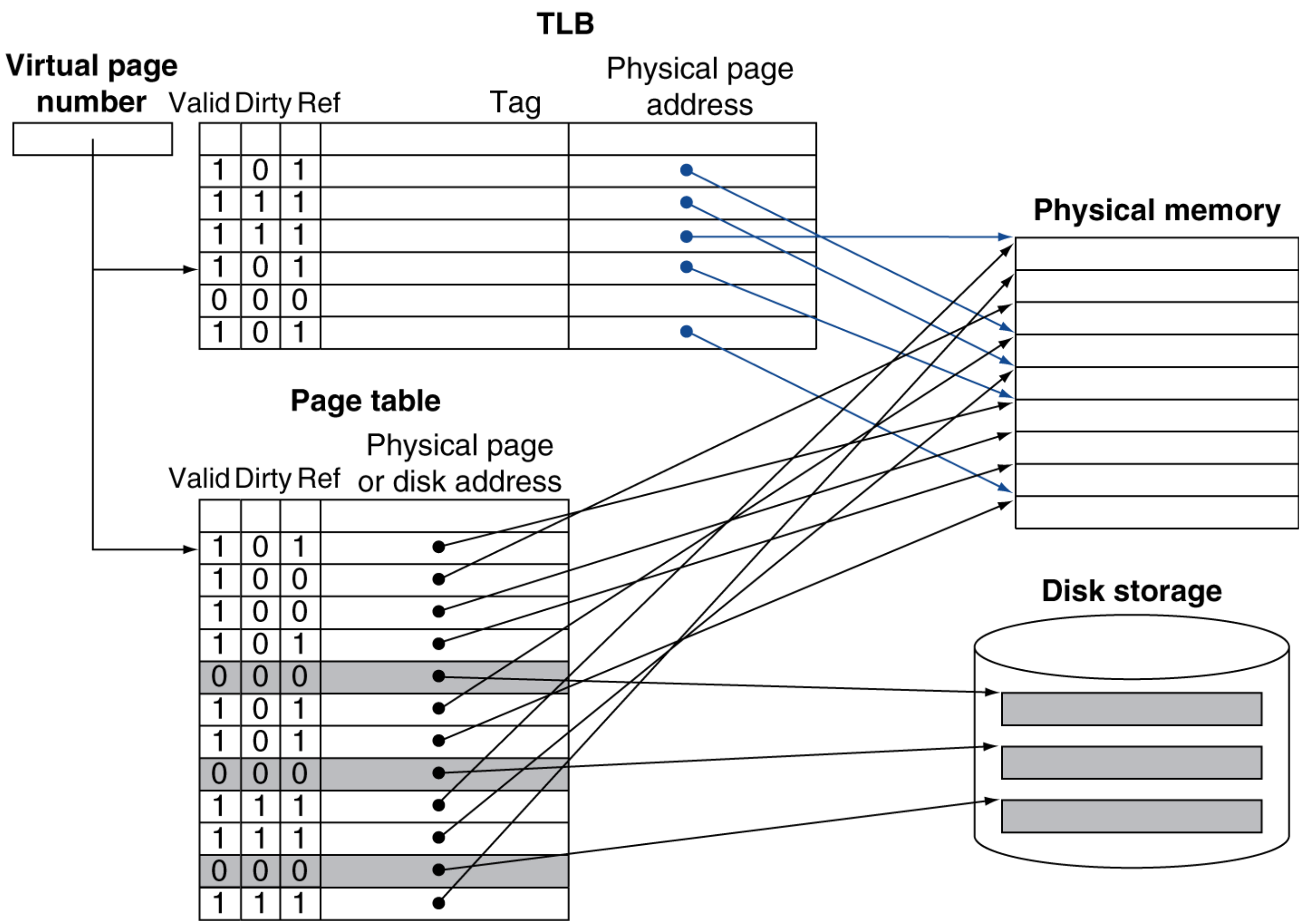


Descriptors of 2 Bytes, each table (except the first) fits 1 page

Fast Translation Using a TLB

- Address translation would appear to require extra memory references
  - One to access the PTE (Page Translation Entry)
  - Then the actual memory access
- But access to page tables has good locality
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - Misses could be handled by hardware or software

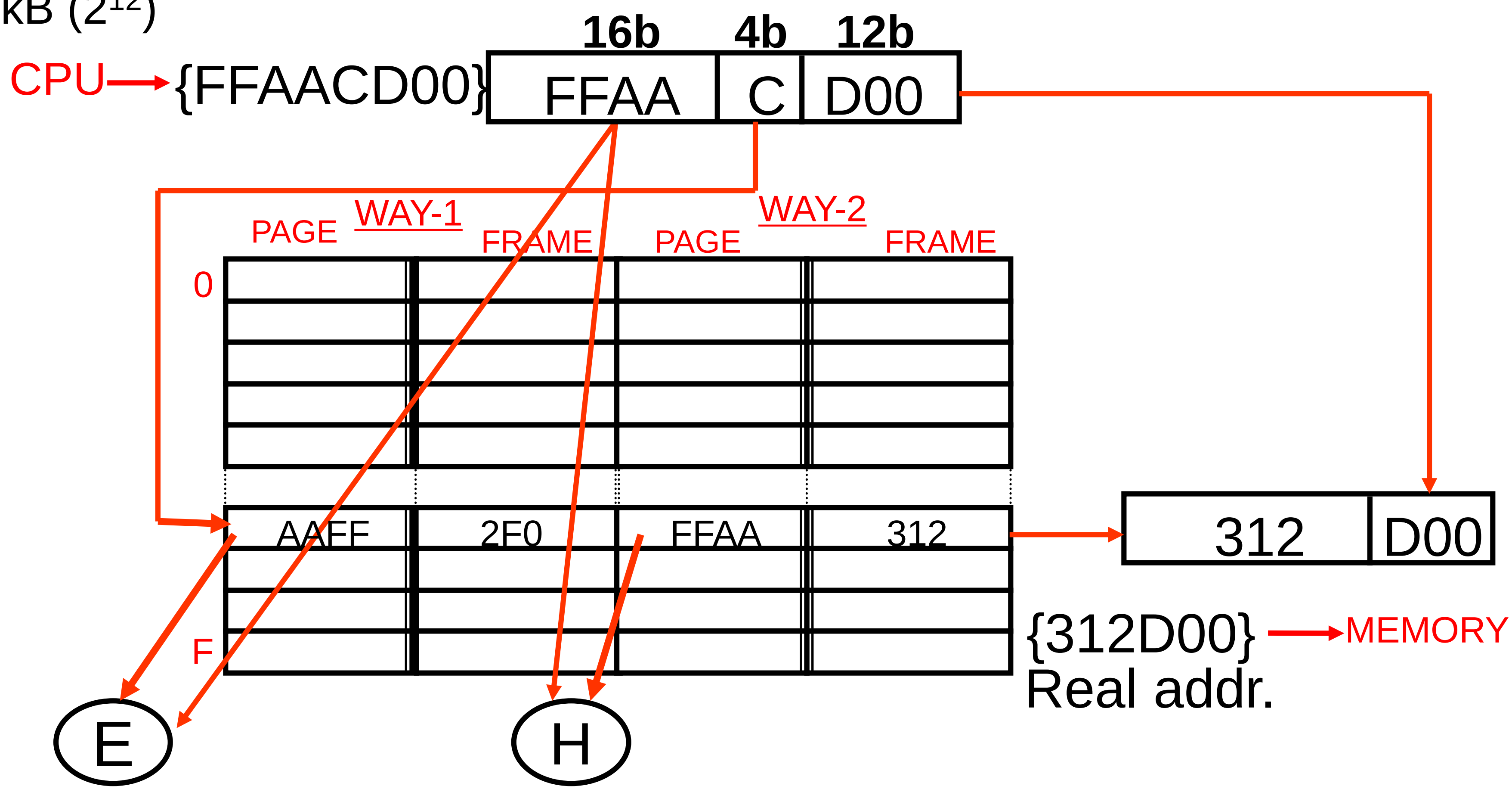
Fast Translation Using a TLB



Fast Translation Using a TLB

TLB N-way Associative, 2 ways, 16 entries/way

Ejemplo: Virtual Memory: 4 GB (2^{32}), Real: 16 MB (2^{24}). Page size: 4 kB (2^{12})



TLB Misses

- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction

TLB Miss Handler

- TLB miss indicates
  - Page present, but PTE not in TLB
  - Page not present
- Must recognize TLB miss before destination register overwritten
  - Raise exception
- Handler copies PTE from memory to TLB
  - Then restarts instruction
  - If page not present, page fault will occur

Page Fault Handler

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
  - Restart from faulting instruction

Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written

Page table and TLB information

The information stored in a TLB or page table entry is called descriptor and contains:

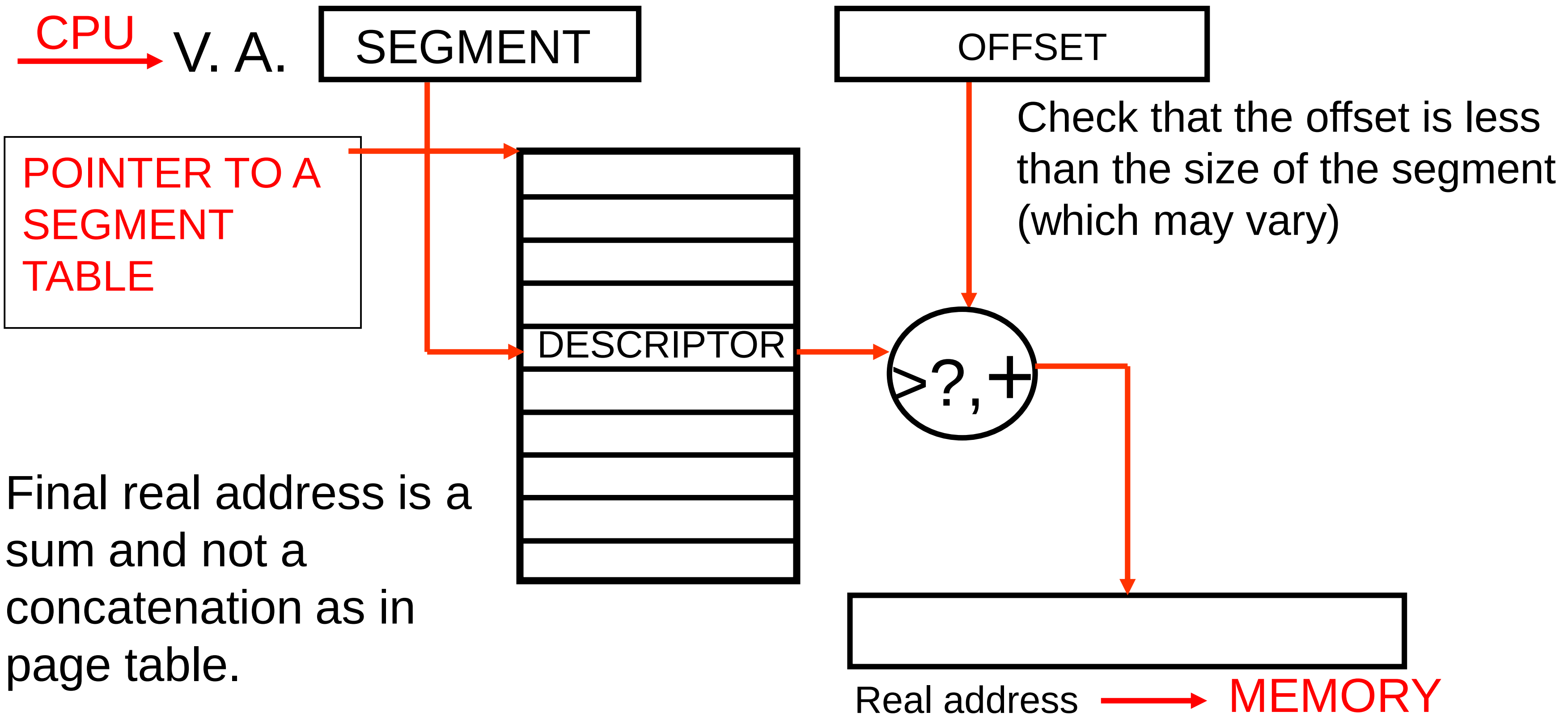
- Page frame: It gives the real address (Real address = FRAME & OFFSET)
- Bits to control:
  - Present bit: '1' indicates the page referenced resides in main memory
  - Use bit: '1' to indicate that some element of the page has been referenced. It is used to decide which page is replaced.
  - Dirty bit:'1' to indicate that some data in the page has been modified (written).
  - Protection bits: supervisor, only-readable, non-cacheable, used by the OS.
  - Replacement bits: to apply the replacement algorithms (LRU, etc).

Memory Protection

- Different tasks can share parts of their virtual address spaces
  - But need to protect against errant access
  - Requires OS assistance
- Hardware support for OS protection
  - Privileged supervisor mode (aka kernel mode)
  - Privileged instructions
  - Page tables and other state information only accessible in supervisor mode
  - System call exception (e.g., syscall in MIPS)

Translation using a Segment Table

The address is divided into segment and offset



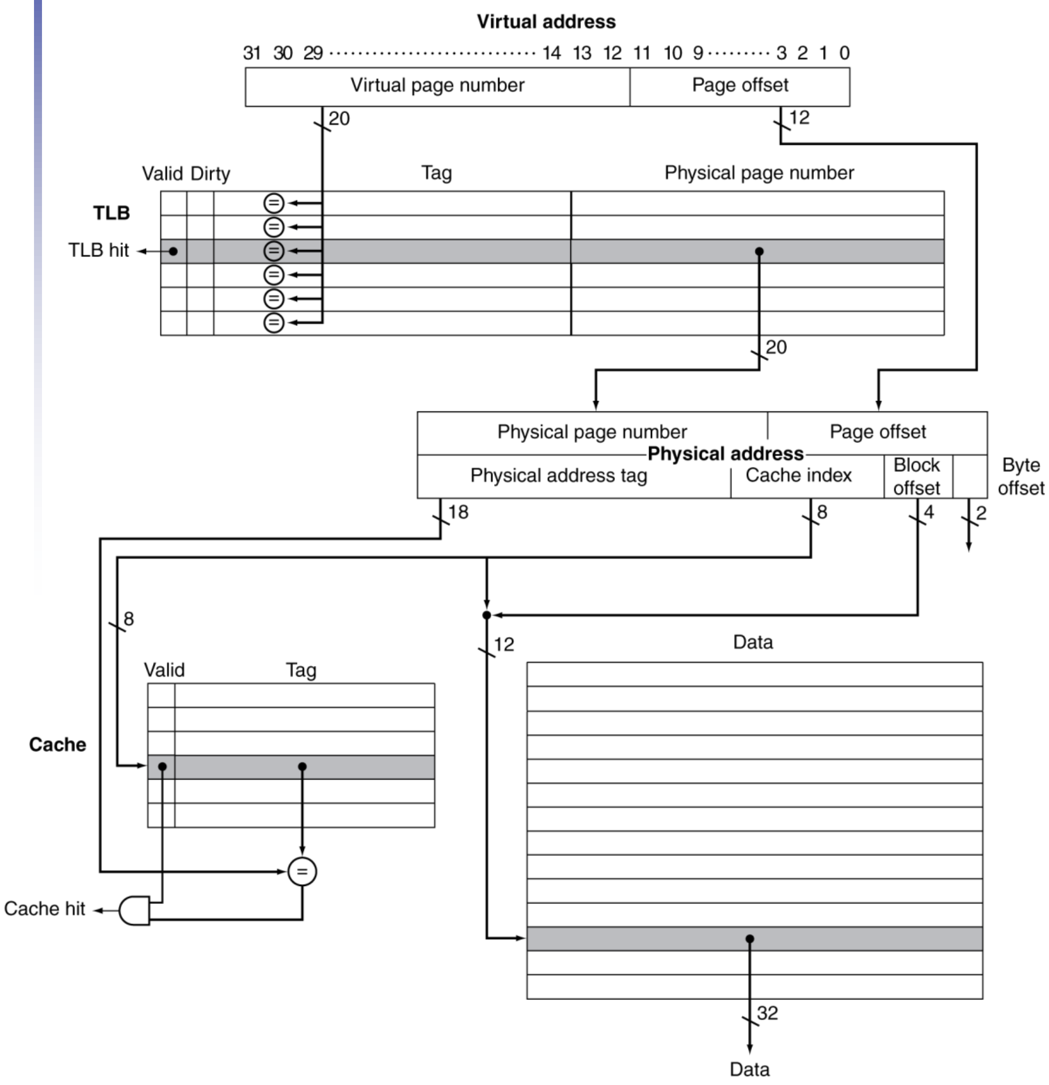
Final real address is a sum and not a concatenation as in page table.

Translation using a Segment Table

What does the segment descriptor contain?

- **Segment start address:** It is added to the offset to compute the real address
- **Segment size:** It must be greater than the offset
- **Bits to control**
  - Present bit in main memory
  - Protection bit: against write operations (code segment)
  - Exclusion bit: to restringe the access (system security)
- **Bits for replacement algorithms: LRU**

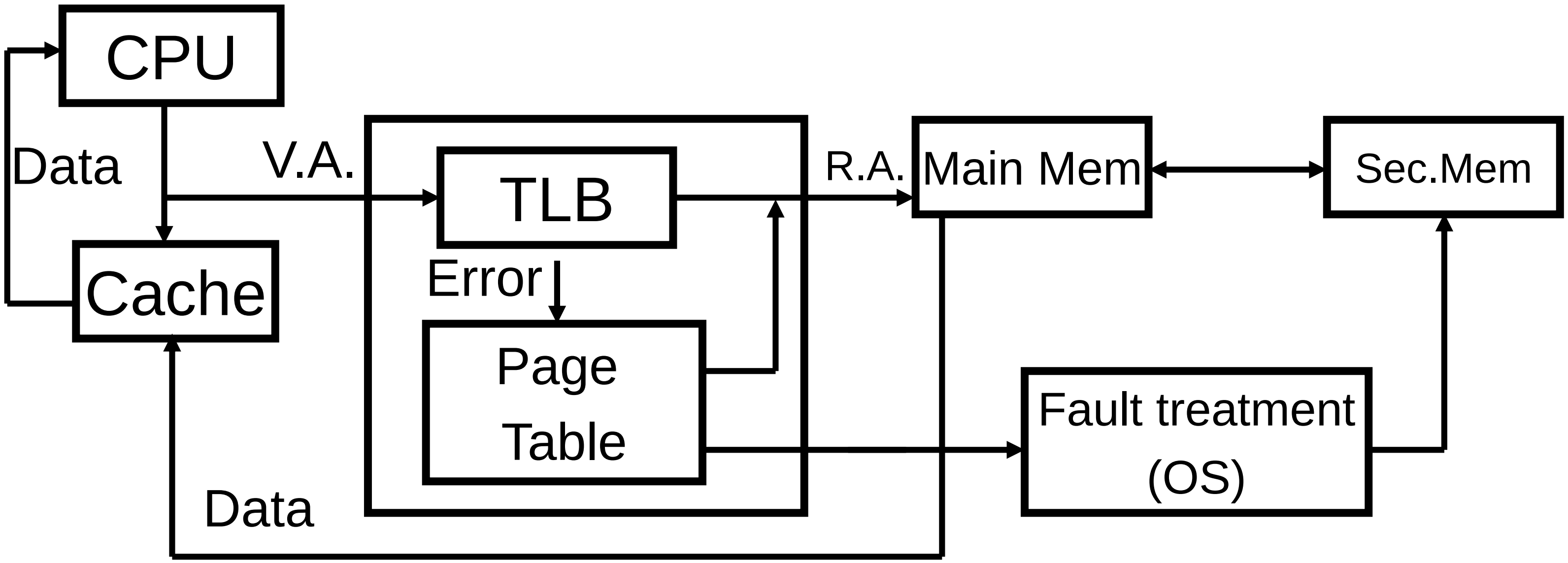
TLB and Cache Interaction



- If cache tag uses physical address
  - Need to translate before cache lookup
- Alternative: use virtual address tag
  - Complications due to aliasing
    - Different virtual addresses for shared physical address

TLB and Cache Interaction

- Virtual Cache (from virtual address)



❑Same time access to cache and TLB

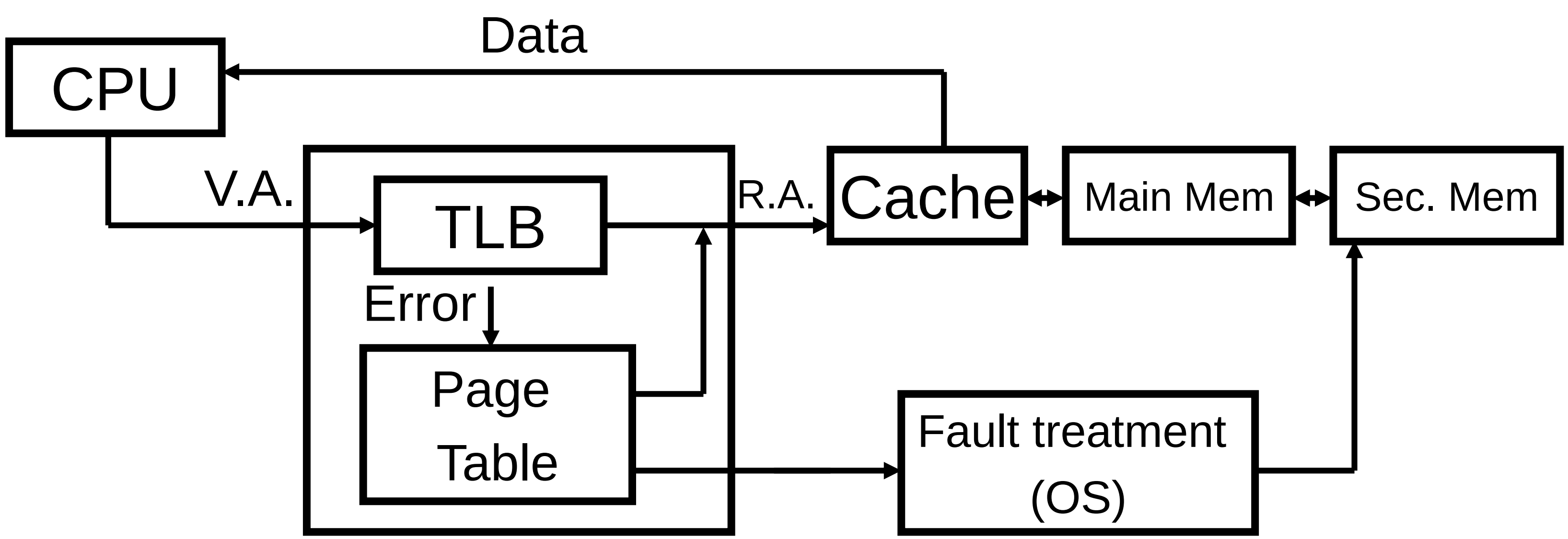
❑ Memory access time: hit cache, t_c, error cache, $t_{TLB}+t_B+t_c$

❑*aliasing*: two virtual addresses to the same real address -> 2 entries in virtual cache for the same data

❑ Cache problem with different processes: there can be virtual addresses duplicated. To avoid this, a process identifier is added to the virtual address

TLB and Cache Interaction

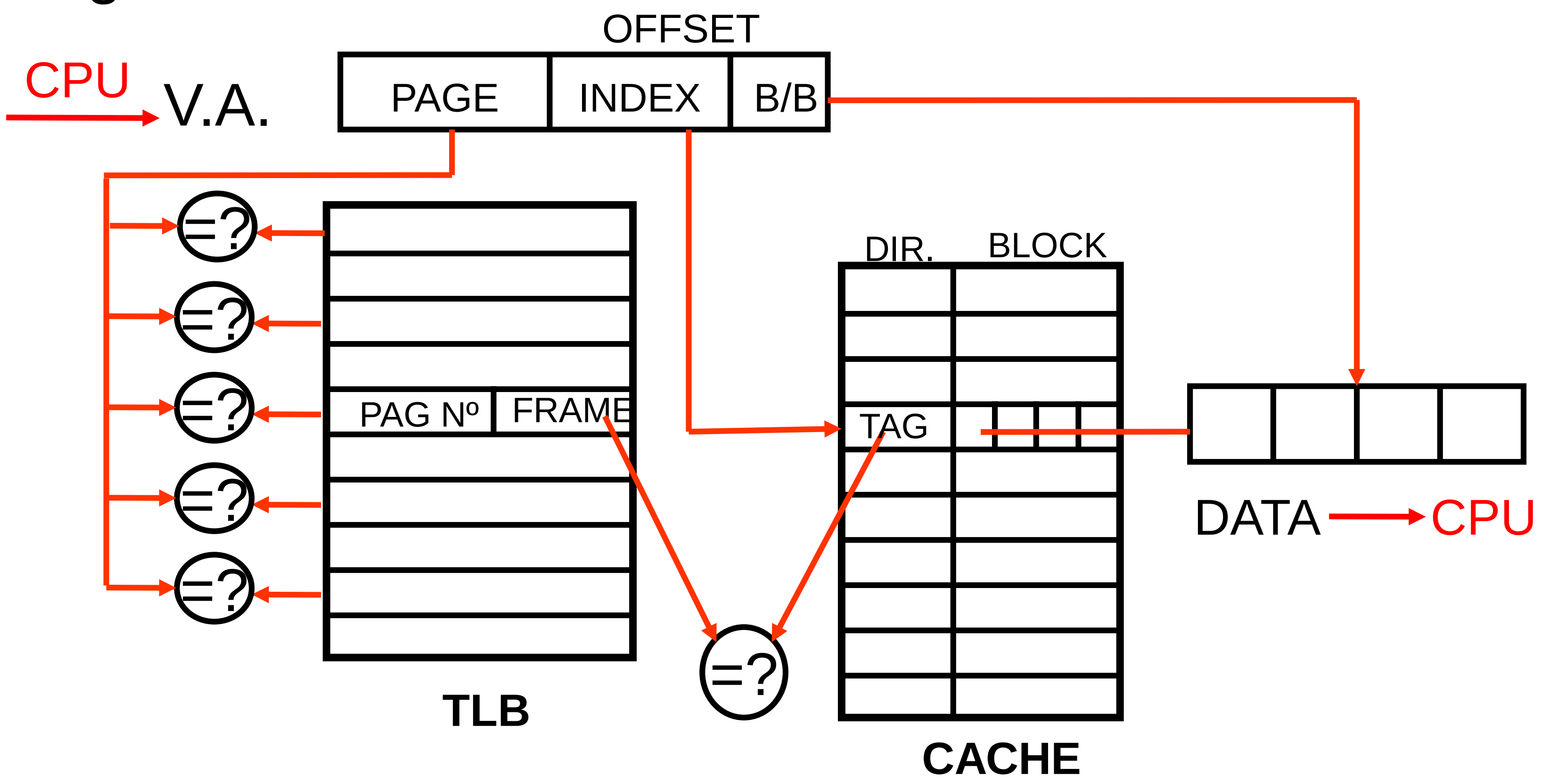
- Real Cache (from real address)



- Minimum memory access time: TLB time + cache time
- Solved having several address spaces
- To speed-up, page offset can contain the index and the byte in block of the cache

TLB and Cache Interaction

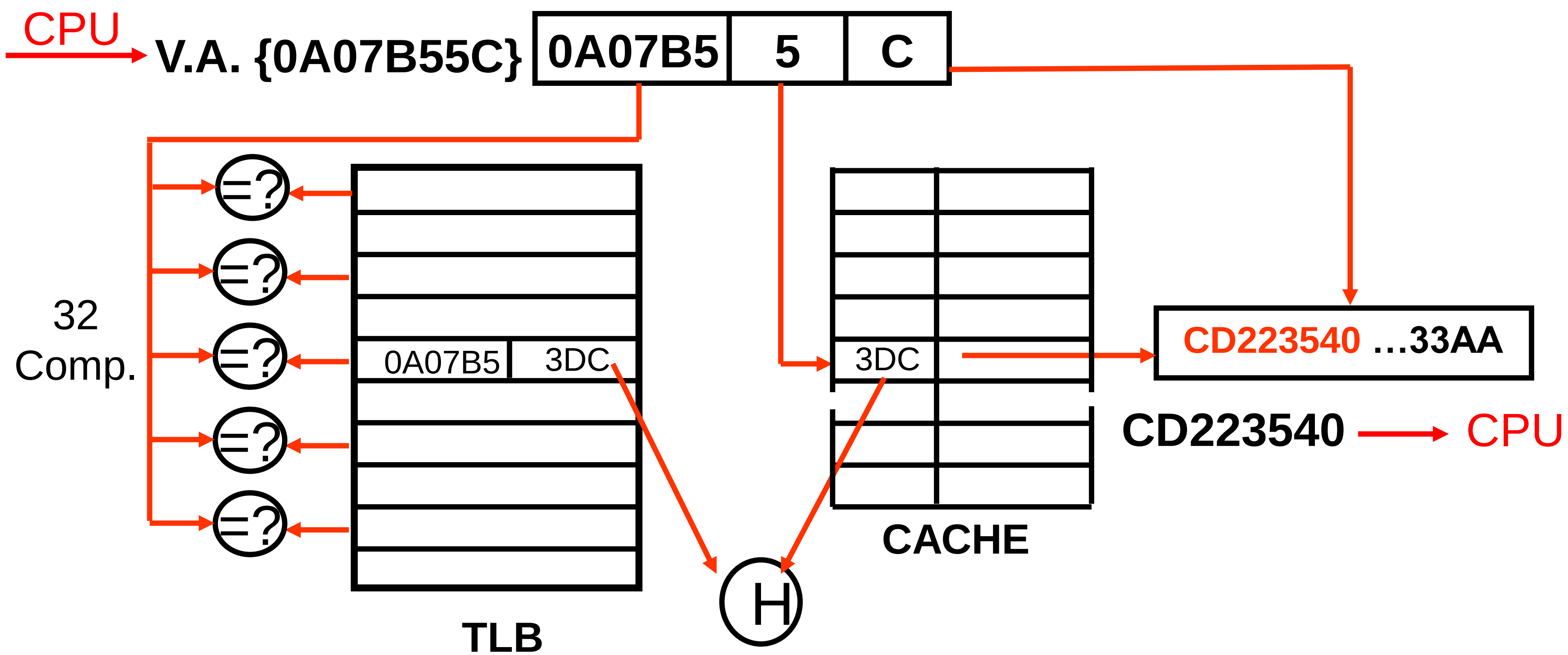
- Real cache with parallel access to the TLB frame and cache tag. Next, compare between frame and tag.



TLB and Cache Interaction

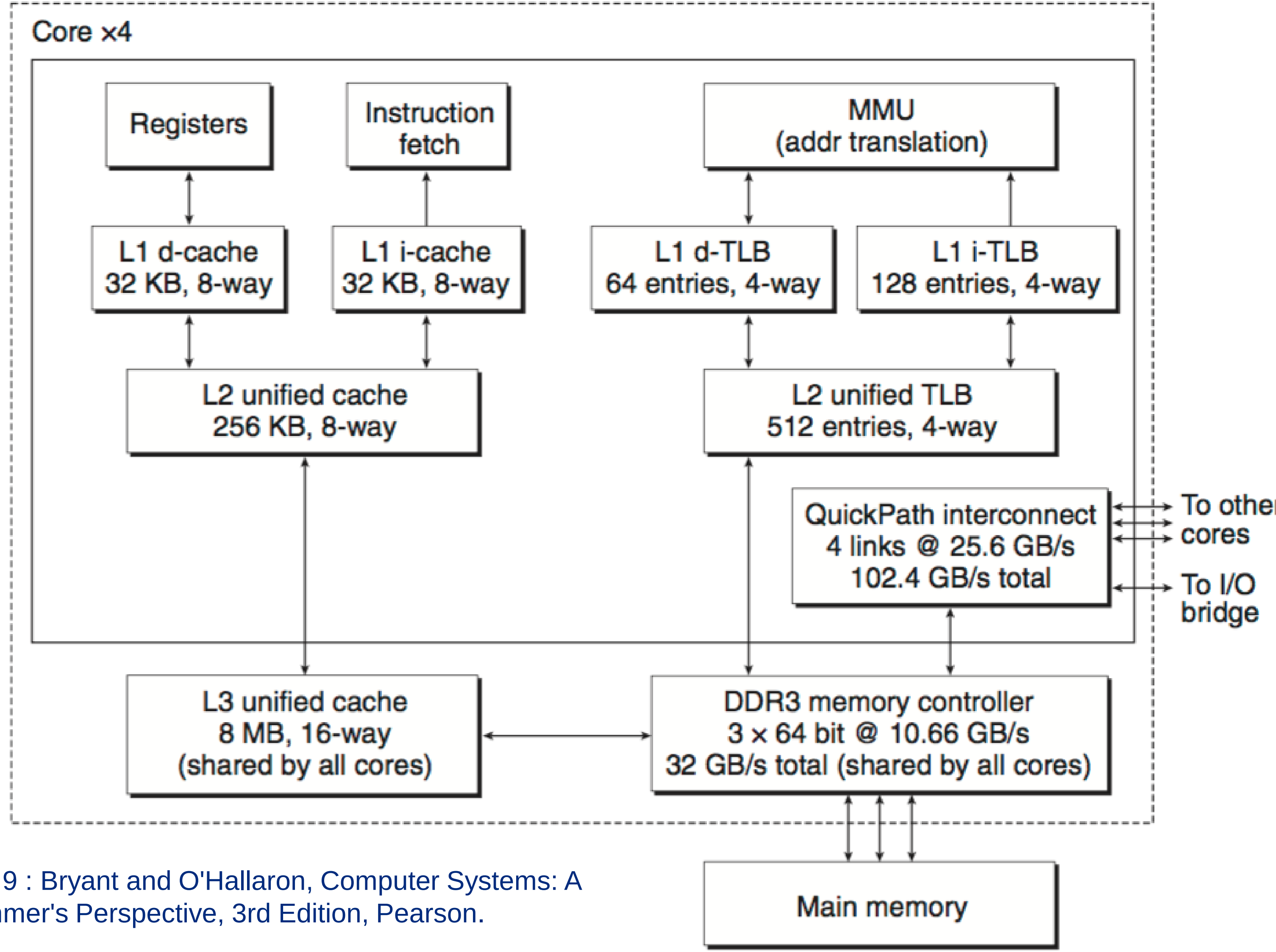
Example.- V.A.: 32b; R.A.: 20b; Pag. size: 256 Bytes

TLB FA 32 entries; Cache DM 256 Bytes, 16 B/B.



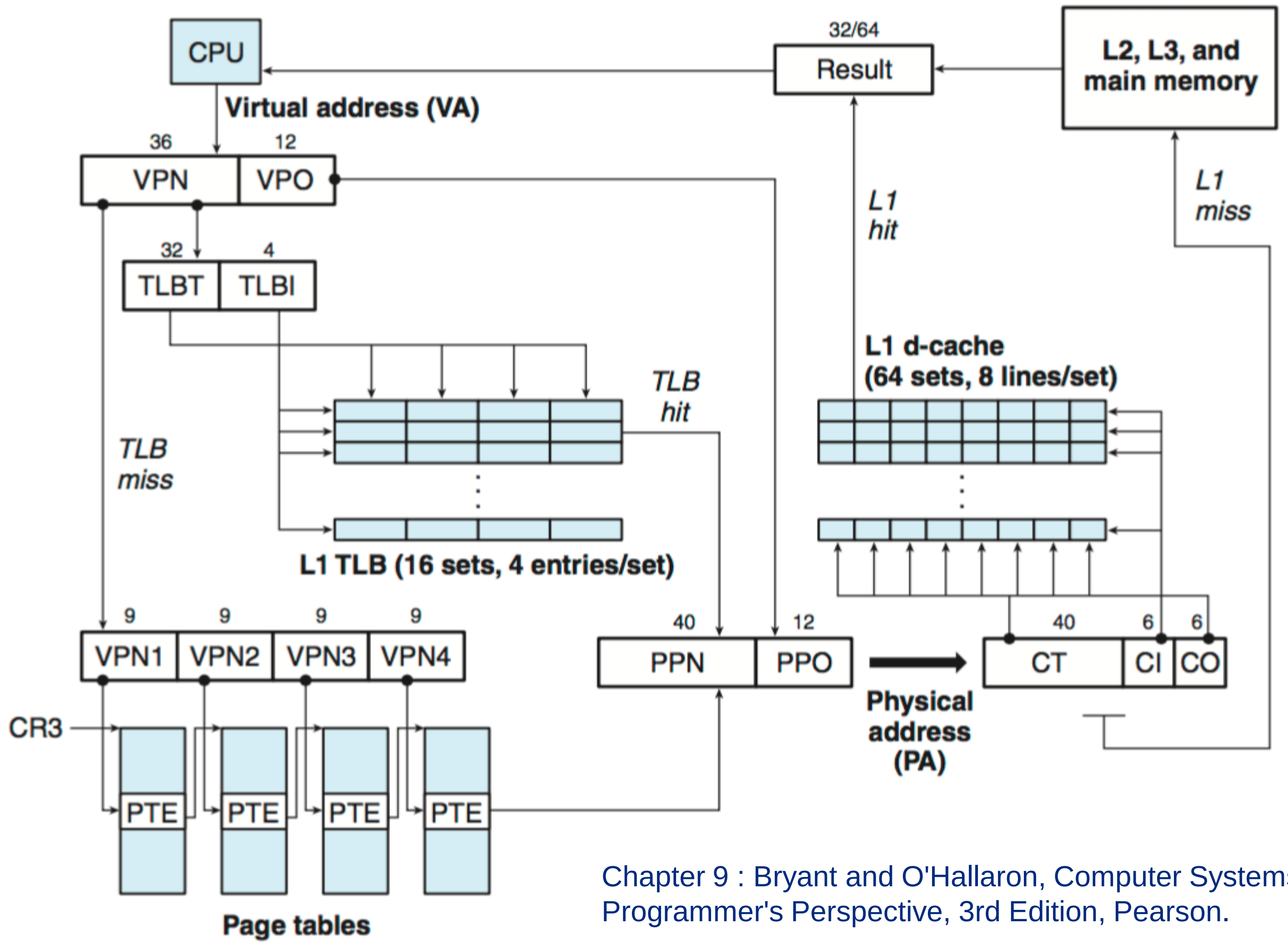
"Offset must include the real cache index so that we can access TLB and cache in parallel"

The Core i7 Memory System



Chapter 9 : Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, 3rd Edition, Pearson.

The Core i7 Memory System



Chapter 9 : Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, 3rd Edition, Pearson.

Figure 9.22 **Summary of Core i7 address translation.** For simplicity, the i-caches, i-TLB, and L2 unified TLB are not shown.

Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories ☹
  - Caching gives this illusion ☺
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk
- Memory system design is critical for multiprocessors