References

- Textbook: chapter 8

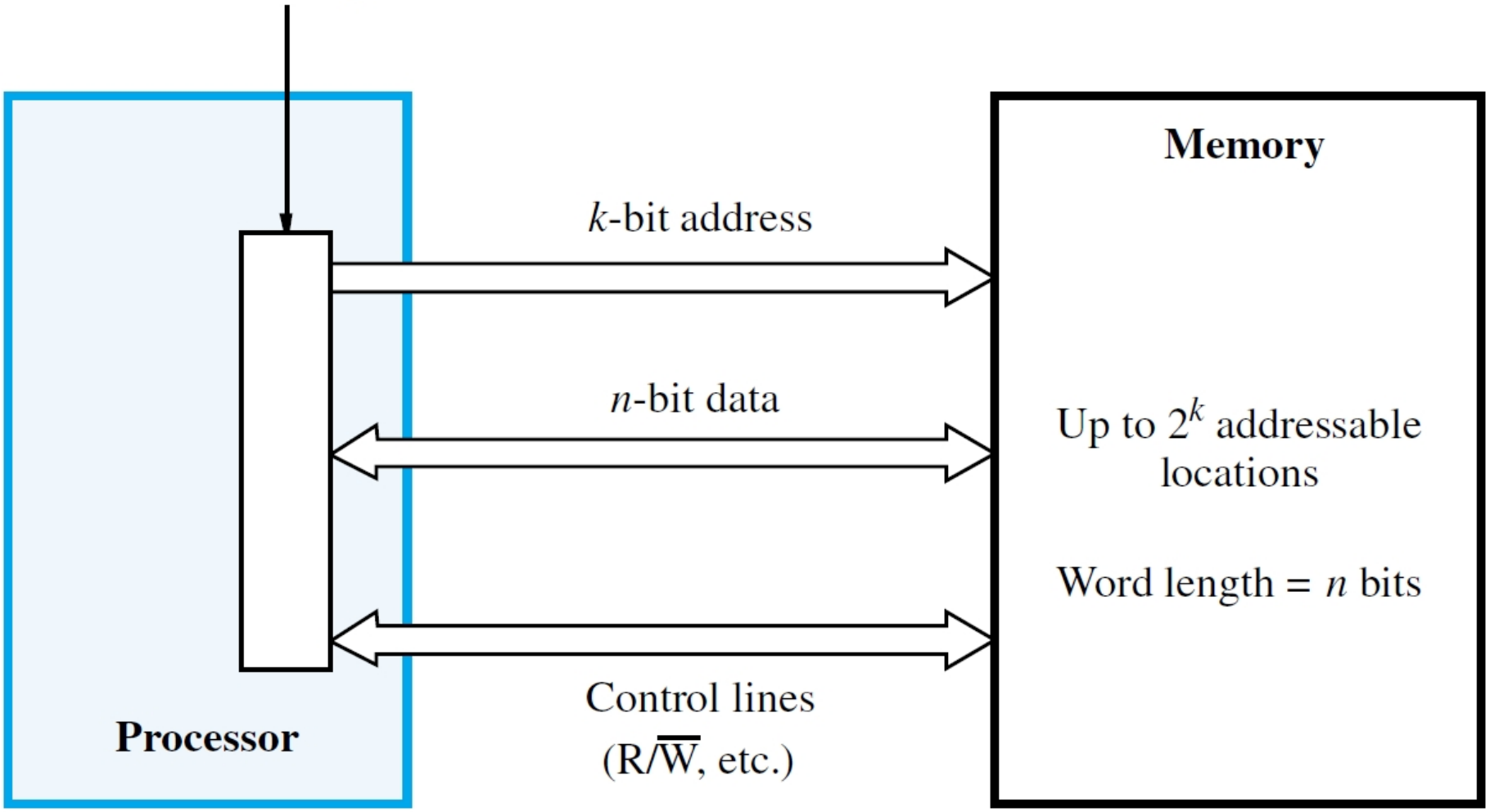
Chapter Outline

- Direct memory access
- Memory hierarchy concepts
- Cache memory and virtual memory

Basic Concepts

- Access provided by process-memory interface
- *Address and data lines*, and also *control lines* for command (Read/Write), timing, data size
- *Memory access time* is time from initiation to completion of a word or byte transfer
- *Memory cycle time* is minimum time delay between initiation of successive transfers
- *Random-access memory (RAM)* means that access time is same, independent of location

Processor-memory interface



Cache and Virtual Memory

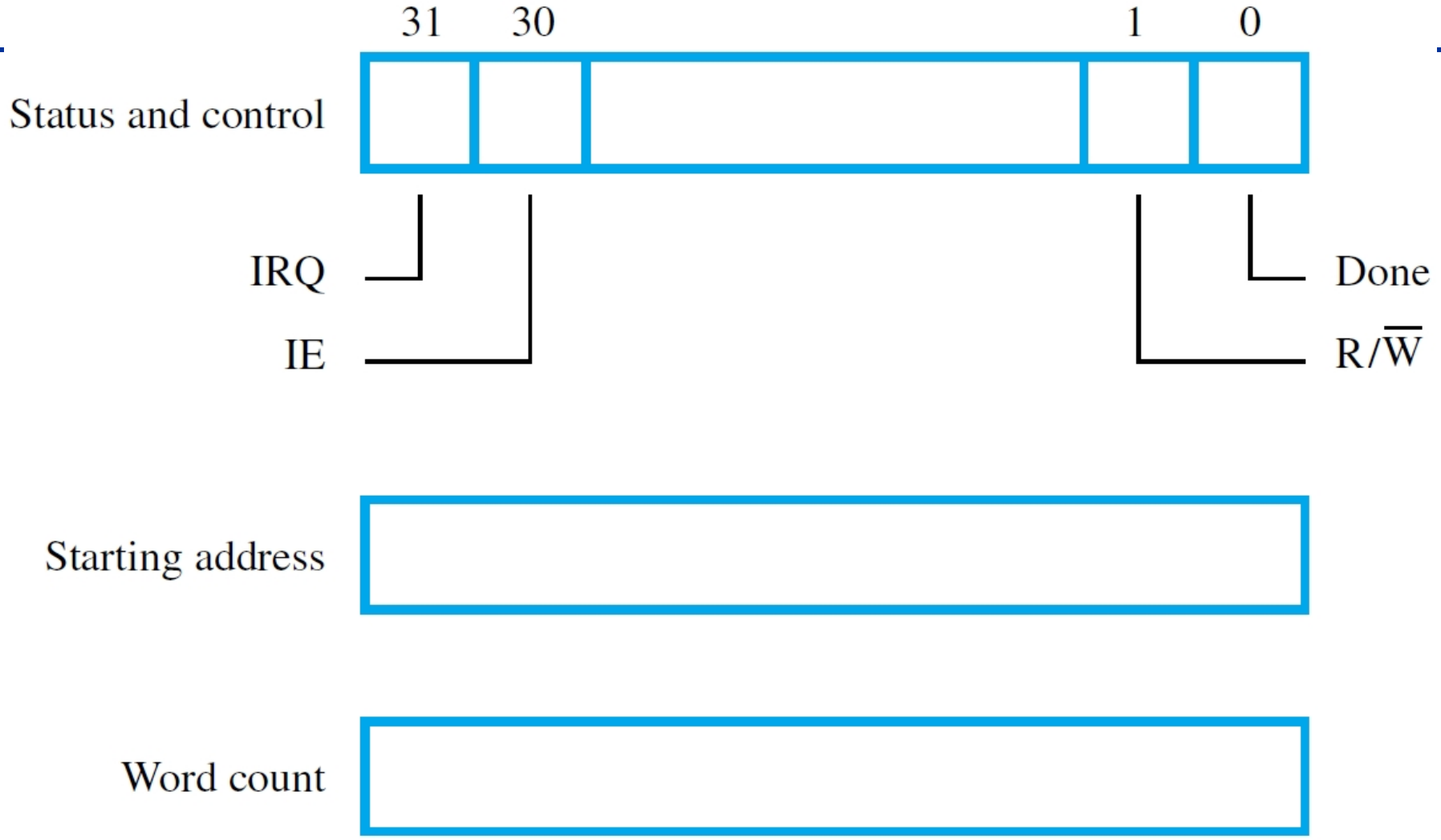
- The main memory is slower than processor
- Cache memory is smaller and faster memory that is used to reduce effective access time
- Holds subset of program instructions and data
- Information for one or more active programs may exceed physical capacity of the memory
- Virtual memory provides larger apparent size by transparently using secondary storage
- Both approaches need efficient *block transfers*

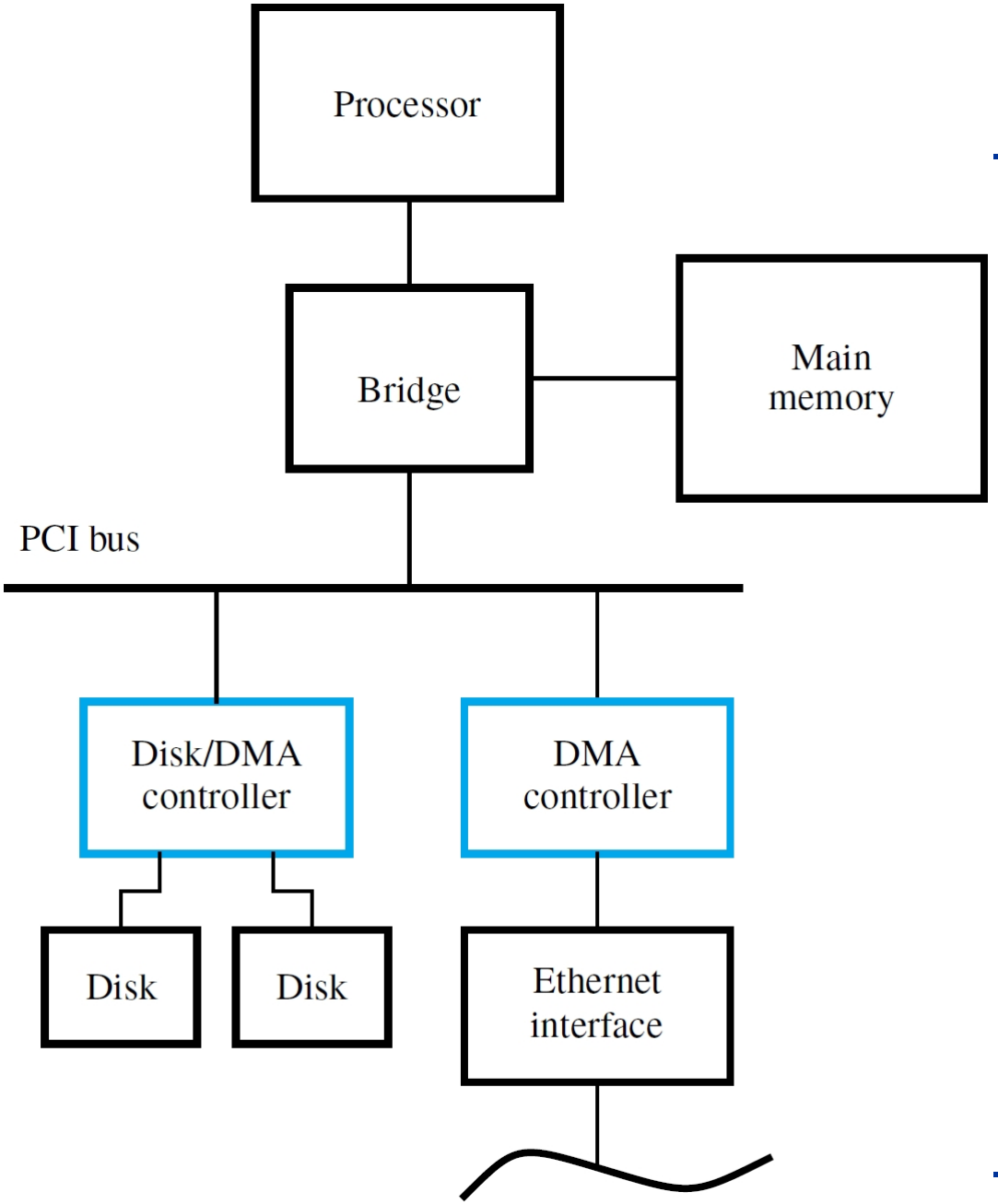
Direct Memory Access

- Program-controlled I/O requires processor to intervene frequently for many data transfers
- Overhead is high because each transfer involves only a single word or a single byte
- Interrupt state-saving and operating system also introduce overheads for small data size
- Alternative: direct memory access (DMA)
- Special unit manages the transfer of larger *blocks* of data between memory & I/O devices

DMA Controller

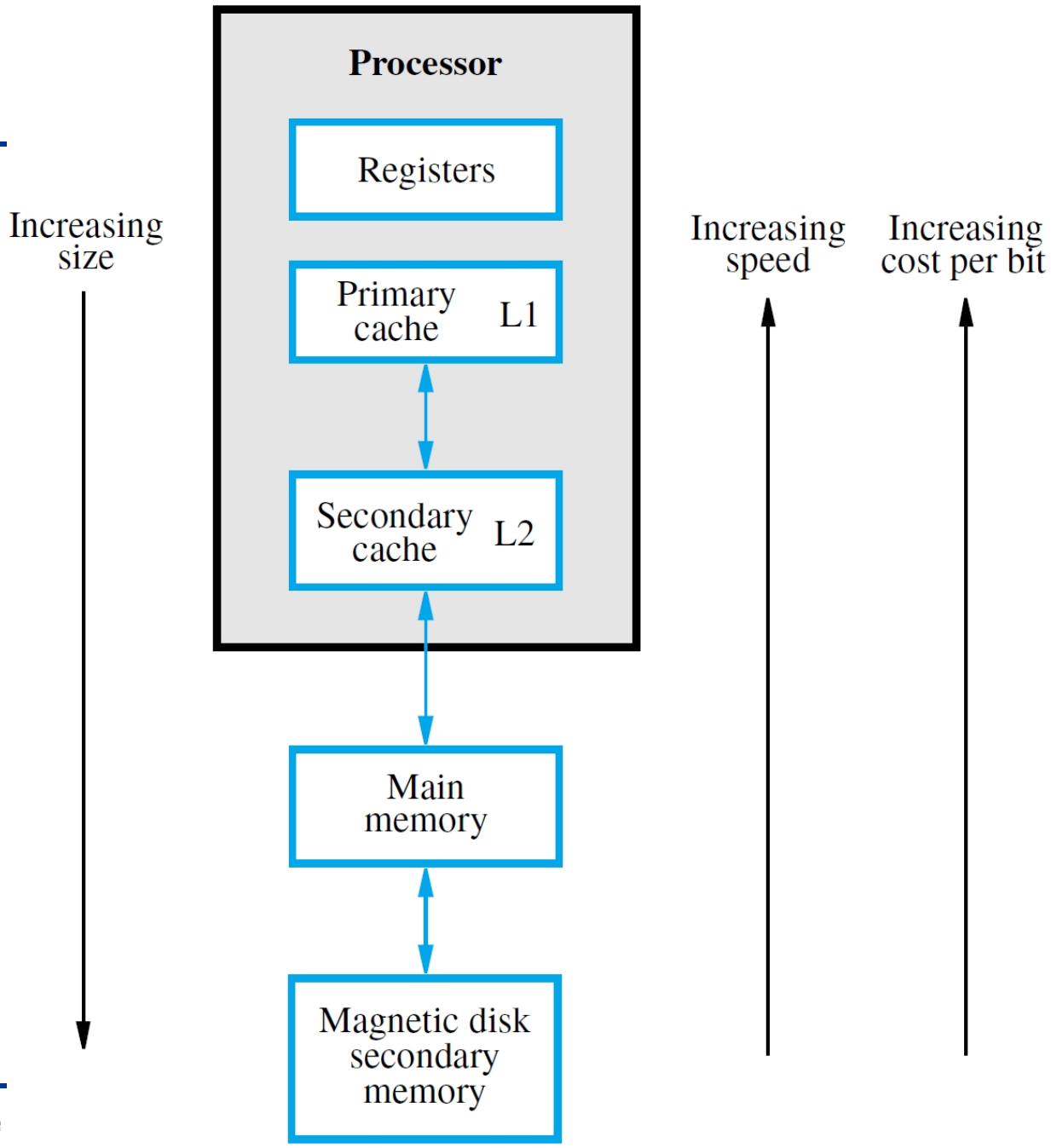
- *DMA controller* is shared, or in each I/O device
- Performs individual memory accesses that would have been done by the processor
- Keeps track of progress with address counter
- Processor initiates DMA controller activity after writing information to special registers (starting address, count, Read/Write, etc.)
- Processor interrupt used to signal completion
- DMA controller examples: disk and Ethernet


31
30
1
0
Status and control
IRQ
IE
Done
R/$\overline{W}$
Starting address
Word count


Processor
Bridge
Main memory
PCI bus
Disk/DMA controller
DMA controller
Disk
Disk
Ethernet interface

Memory Hierarchy

- Ideal memory is fast, large, and inexpensive
- Not feasible, so use memory hierarchy instead
- Exploits program behavior to make it *appear* as though memory is fast and large
- Recognizes speed/capacity/cost features of different memory technologies
- Fast static memories are closest to processor
- Slower dynamic memories for more capacity
- Slowest disk memory for even more capacity

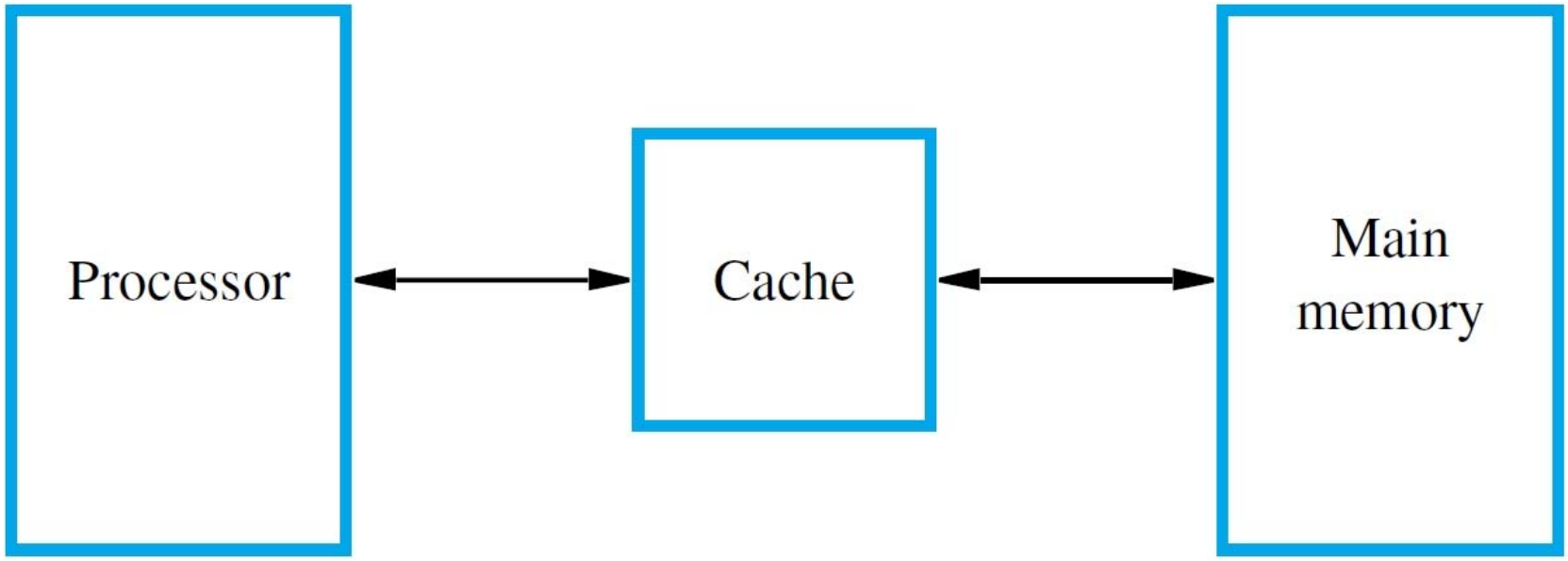

Processor
Registers
Primary cache
L1
Secondary cache
L2
Main memory
Magnetic disk secondary memory
Increasing size
Increasing speed
Increasing cost per bit

Memory Hierarchy

- Processor registers are fastest, but do not use the same address space as the memory
- Cache memory often consists of 2 (or 3) levels, and technology enables on-chip integration
- Holds copies of program instructions and data stored in the large external main memory
- For very large programs, or multiple programs active at the same time, need more storage
- Use disks to hold what exceeds main memory

Caches and Locality of Reference

- The cache is between processor and memory
- Makes large, slow main memory *appear* fast
- Effectiveness is based on locality of reference
- Typical program behavior involves executing instructions in loops and accessing array data
- Temporal locality: instructions/data that have been recently accessed are likely to be *again*
- Spatial locality: *nearby* instructions or data are likely to be accessed after current access


Processor
Cache
Main memory

More Cache Concepts

- To exploit spatial locality, transfer *cache block* with multiple adjacent words from memory
- Later accesses to nearby words are fast, provided that cache still contains the block
- *Mapping function* determines where a block from memory is to be located in the cache
- When cache is full, *replacement algorithm* determines which block to remove for space

Cache Operation

- Processor issues Read and Write requests as if it were accessing main memory directly
- But control circuitry first checks the cache
- If desired information is present in the cache, a *read* or *write hit* occurs
- For a read hit, main memory is not involved; the cache provides the desired information
- For a write hit, there are two approaches

Handling Cache Writes

- *Write-through protocol*: update cache & mem.
- *Write-back protocol*: only updates the cache; memory updated later when block is replaced
- Write-back scheme needs *modified* or *dirty bit* to mark blocks that are updated in the cache
- If same location is written repeatedly, then write-back is much better than write-through
- Single memory update is often more efficient, even if writing back unchanged words

Handling Cache Misses

- If desired information is not present in cache, a *read* or *write miss* occurs
- For a read miss, the block with desired word is transferred from main memory to the cache
- For a write miss under write-through protocol, information is written to the main memory
- Under write-back protocol, first transfer block containing the addressed word into the cache
- Then overwrite location in cached block

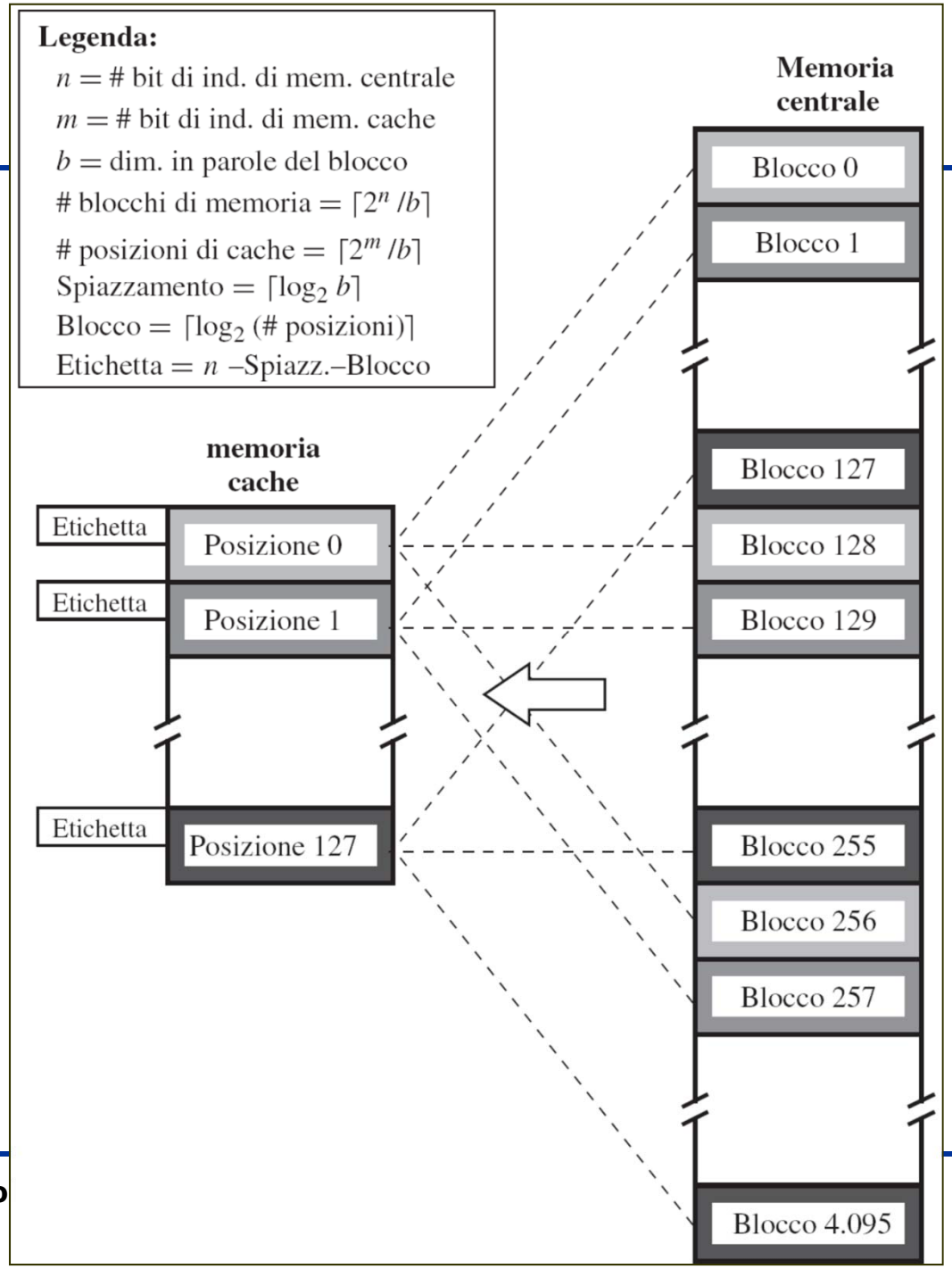
Mapping Functions

- Block of consecutive words in main memory must be transferred to the cache after a miss
- The *mapping function* determines the location
- Study three different mapping functions
- Use small cache with 128 blocks of 16 words
- Use main memory with 64K words (4K blocks)
- *Word*-addressable memory, so 16-bit address

Direct Mapping

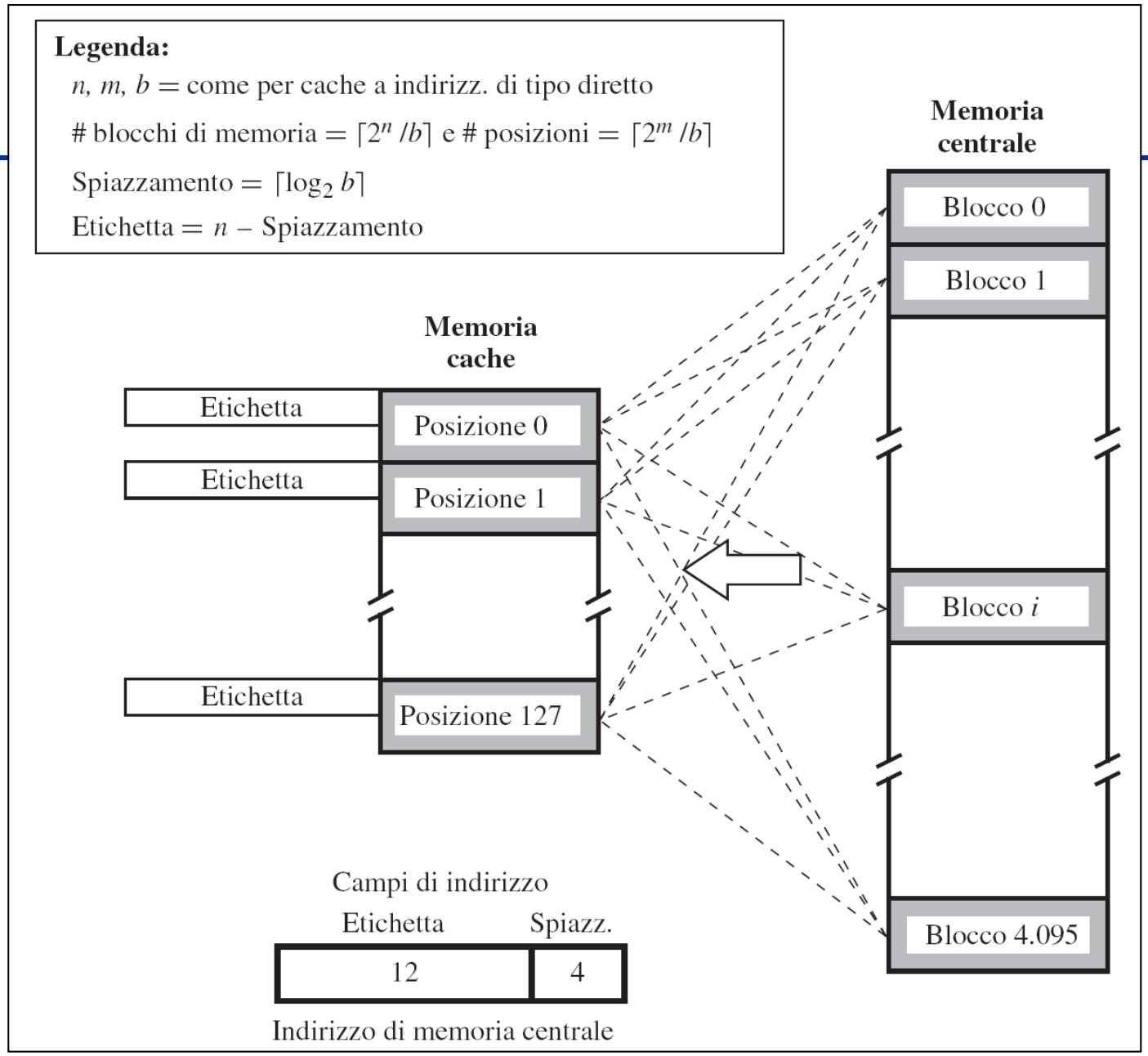
- Simplest approach uses a fixed mapping:
  memory block $j \rightarrow$ cache block (j mod 128)
- Only one unique location for each mem. block
- Two blocks may contend for same location
- New block always overwrites previous block
- Divide address into 3 fields: word, block, tag
- Block field determines location in cache
- Tag field from original address stored in cache
- Compared with later address for hit or miss

word	block	tag
5	7	4



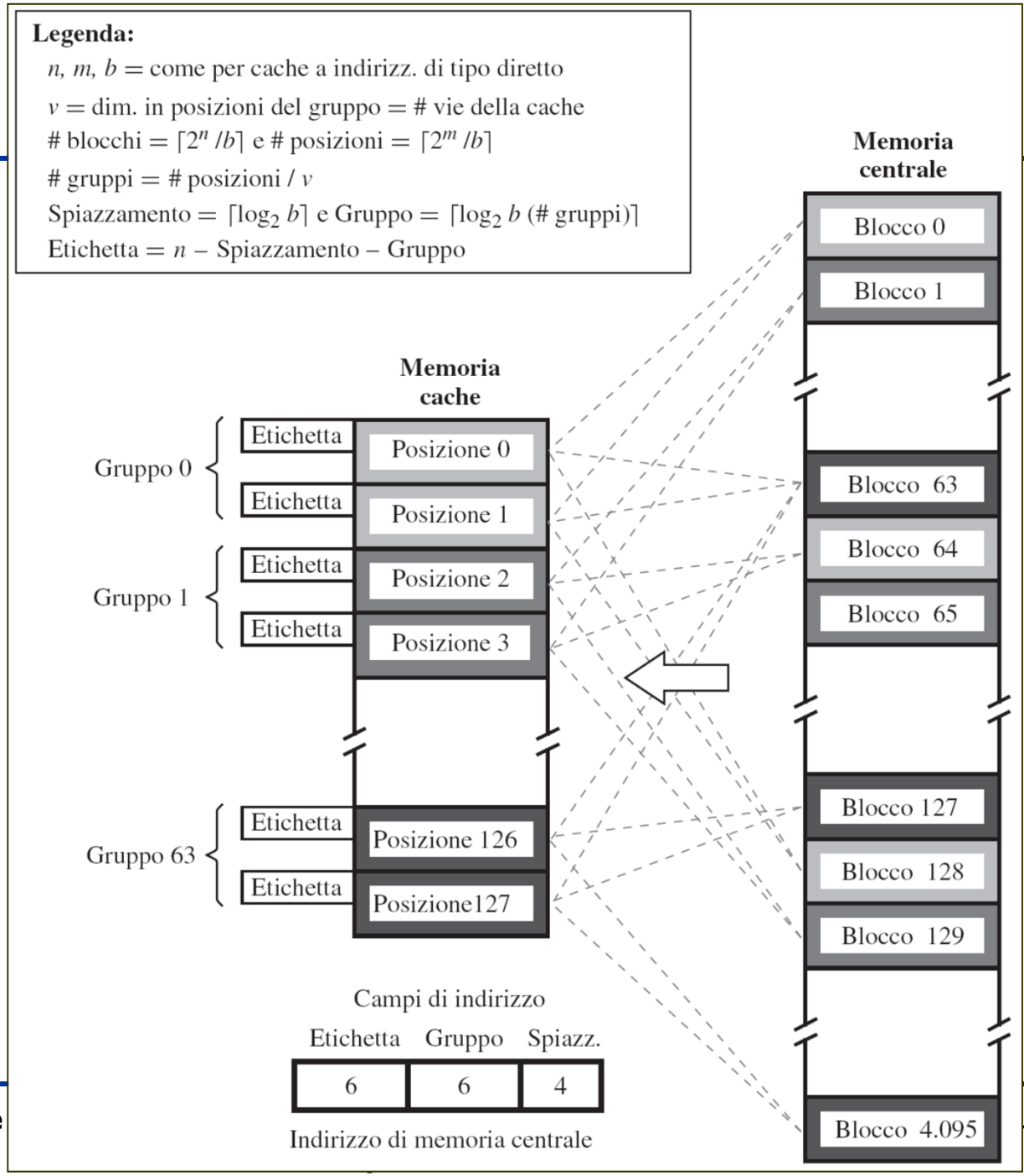
Associative Mapping

- Full flexibility: locate block anywhere in cache
- Block field of address no longer needs any bits
- Tag field is enlarged to encompass those bits
- Larger tag stored in cache with each block
- For hit/miss, compare all tags simultaneously in parallel against tag field of given address
- This *associative search* increases complexity
- Flexible mapping also requires appropriate replacement algorithm when cache is full


Legenda:
n, m, b = come per cache a indirizz. di tipo diretto
# blocchi di memoria = ⌈2^n /b⌉ e # posizioni = ⌈2^m /b⌉
Spiazzamento = ⌈log2 b⌉
Etichetta = n – Spiazzamento
Memoria centrale
Blocco 0
Blocco 1
Blocco i
Blocco 4.095
Memoria cache
Etichetta
Posizione 0
Etichetta
Posizione 1
Etichetta
Posizione 127
Campi di indirizzo
Etichetta
Spiazz.
12
4
Indirizzo di memoria centrale

Set-Associative Mapping

- Combination of direct & associative mapping
- Group blocks of cache into *sets*
- Block field bits map a block to a unique set
- But any block within a set may be used
- Associative search involves only tags in a set
- Replacement algorithm is only for blocks in set
- Reducing flexibility also reduces complexity
- *k* blocks/set $\rightarrow$ *k*-way set-associative cache
- Direct-mapped = 1-way; associative = all-way


Legenda:
n, m, b = come per cache a indirizz. di tipo diretto
v = dim. in posizioni del gruppo = # vie della cache
# blocchi = ⌈2^n /b⌉ e # posizioni = ⌈2^m /b⌉
# gruppi = # posizioni / v
Spiazzamento = ⌈log2 b⌉ e Gruppo = ⌈log2 b (# gruppi)⌉
Etichetta = n – Spiazzamento – Gruppo
Memoria centrale
Blocco 0
Blocco 1
Blocco 63
Blocco 64
Blocco 65
Blocco 127
Blocco 128
Blocco 129
Blocco 4.095
Memoria cache
Gruppo 0
Gruppo 1
Gruppo 63
Etichetta
Posizione 0
Posizione 1
Posizione 2
Posizione 3
Posizione 126
Posizione127
Campi di indirizzo
Etichetta Gruppo Spiazz.
6 6 4
Indirizzo di memoria centrale

Stale Data

- Each block has a valid bit, initialized to 0
- No hit if valid bit is 0, even if tag match occurs
- Valid bit set to 1 when a block placed in cache
- Consider direct memory access, where data is transferred from disk to the memory
- Cache may contain *stale* data from memory, so valid bits are cleared to 0 for those blocks
- Memory→disk transfers: avoid stale data by *flushing* modified blocks from cache to mem.

LRU Replacement Algorithm

- Replacement is trivial for direct mapping, but need a method for associative mapping
- Consider temporal locality of reference and use a *least-recently used (LRU)* algorithm
- For *k*-way set associativity, each block in a set has a counter ranging from from 0 to *k*–1
- Hitting on a block clears its counter value to 0; others originally lower in set are incremented
- If set is full, replace the block with counter=3