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

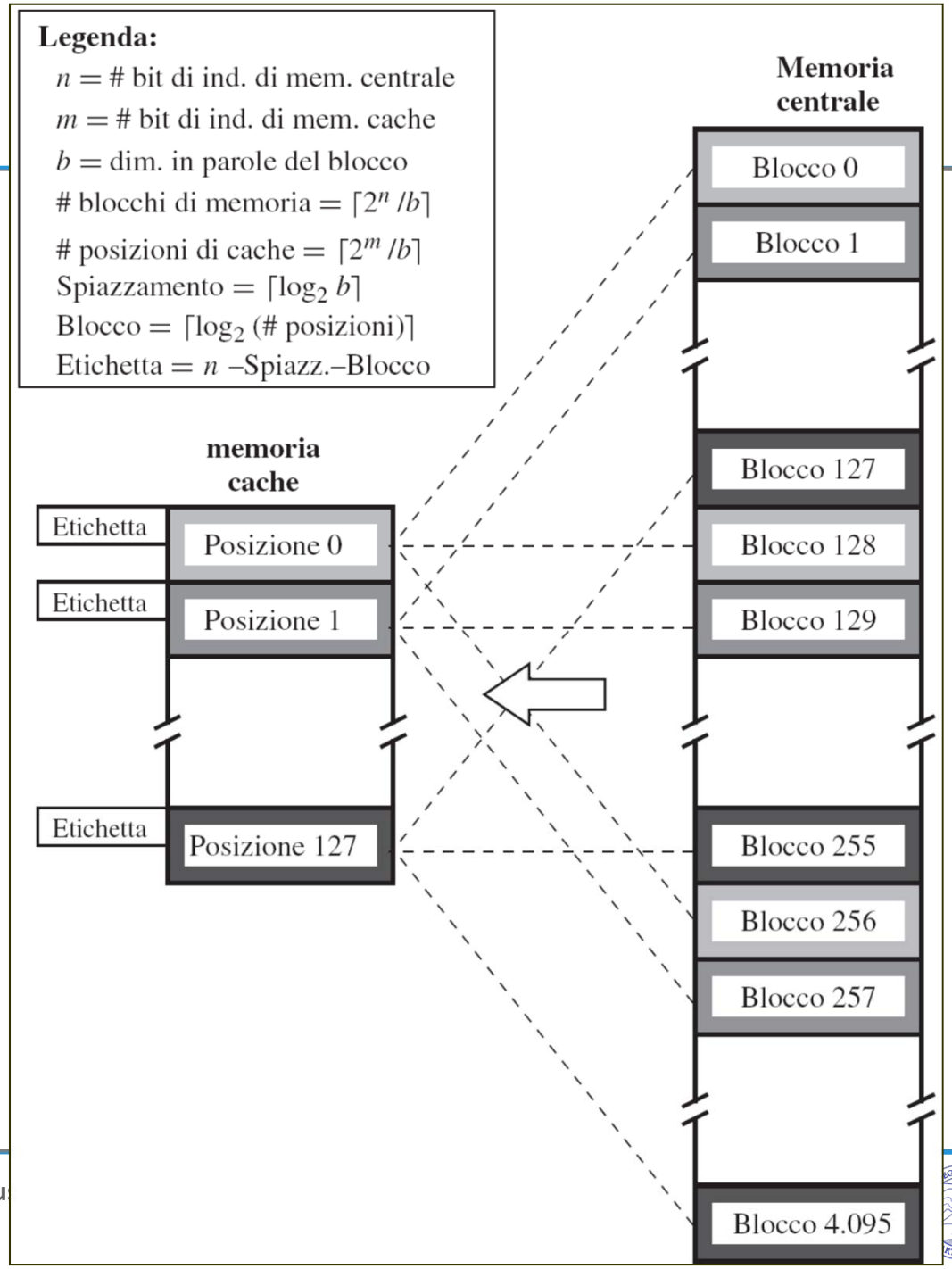
Mapping Functions

- 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

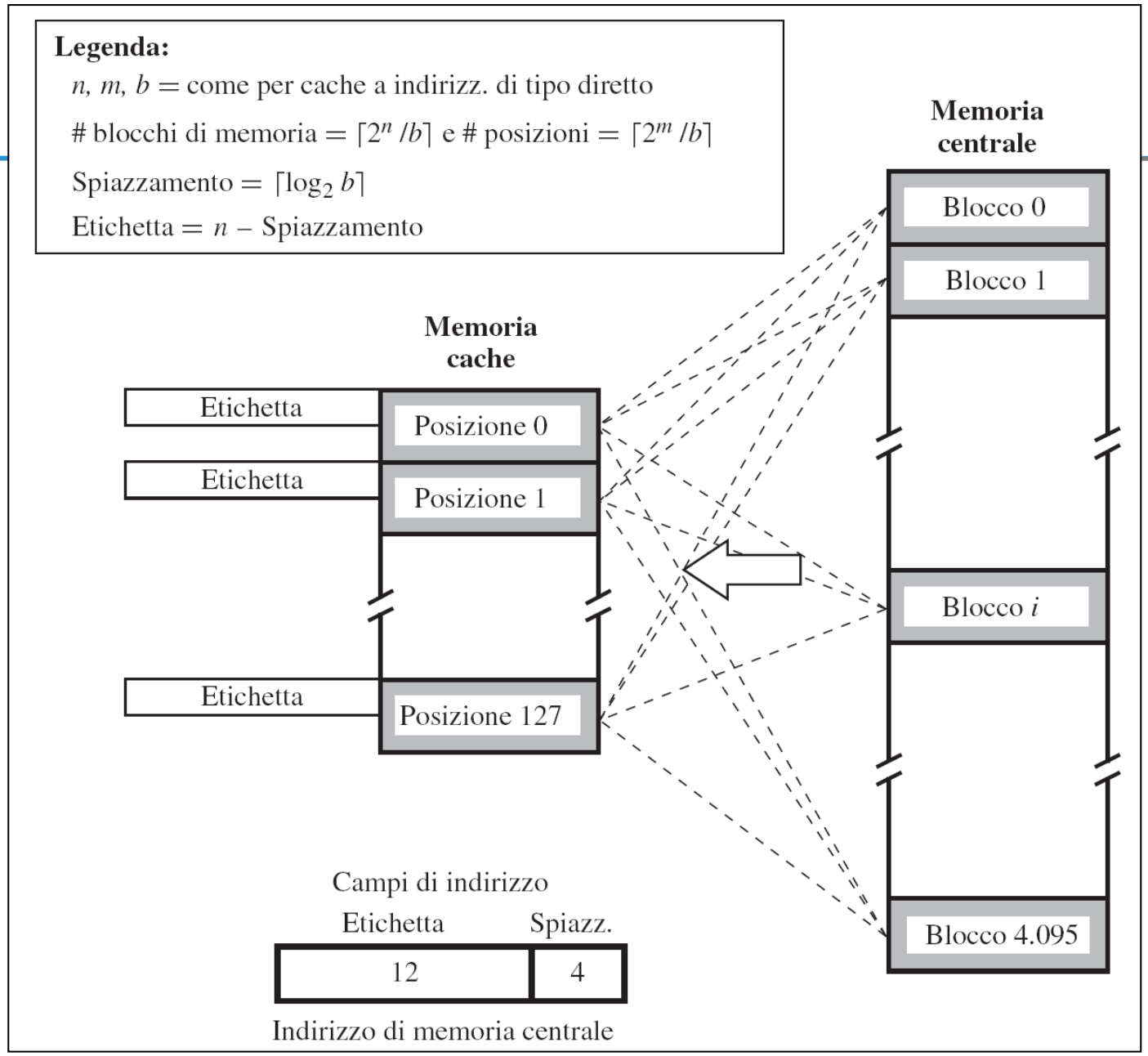
- Simplest approach uses a fixed mapping:
  memory block $j \rightarrow$ cache block (j mod 128)
- Only one unique location for each mem. block
- Two blocks may contend for same location
- New block always overwrites previous block
- Divide address into 3 fields: word, block, tag
- Block field determines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



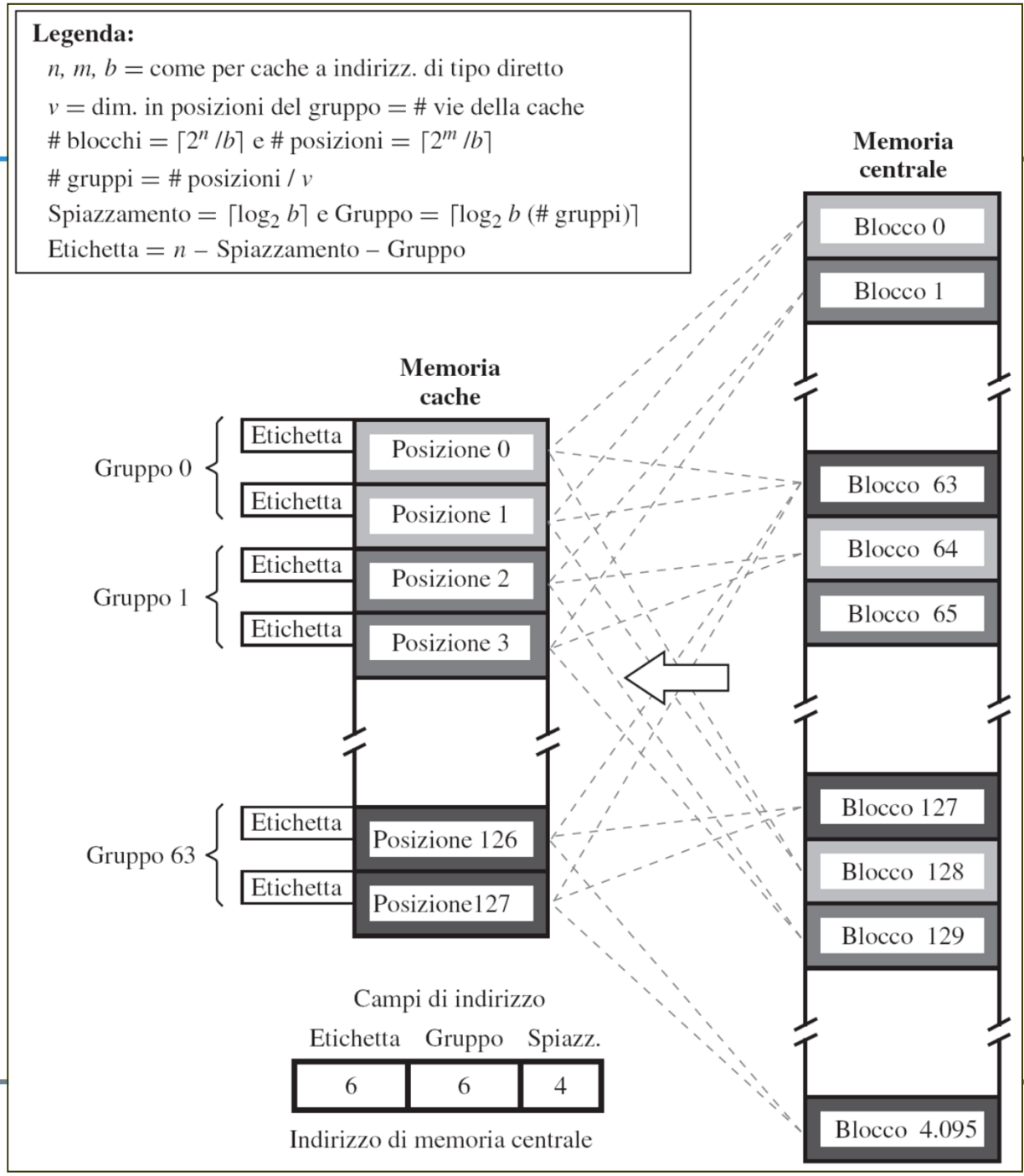
Associative Mapping

- 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

Virtual Memory

- Physical mem. capacity $\leq$ address space size
- A large program or many active programs may not be entirely resident in the main memory
- Use secondary storage (e.g., magnetic disk) to hold portions exceeding memory capacity
- Needed portions are *automatically* loaded into the memory, replacing other portions
- Programmers need not be aware of actions; virtual memory hides capacity limitations