Corso di Architettura dei Sistemi a Microprocessore

Memory Systems: Cache and Virtual Memory

FITNESS

FAULT AND INTRUSION TOLERANT NETWORKED SYSTEMS

Luigi Coppolino

Contact info

Prof. Luigi Coppolino
luigi.coppolino@uniparthenope.it

Università degli Studi di Napoli "Parthenope"
Dipartimento di Ingegneria

Centro Direzionale di Napoli, Isola C4
V Piano lato SUD - Stanza n. 512

Tel: +39-081-5476702
Fax: +39-081-5476777

References

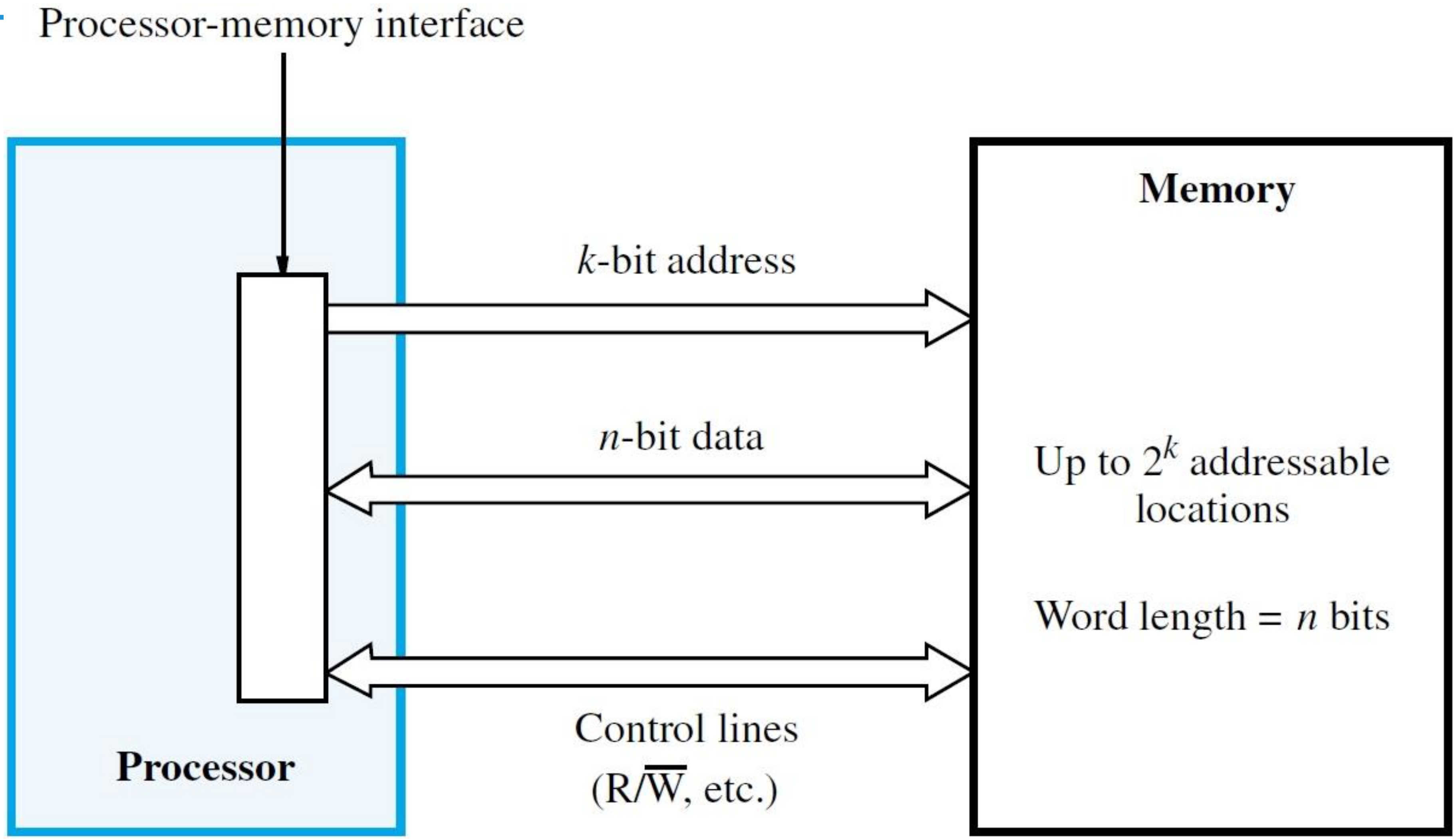
- Textbook: chapter 8

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
Processor
Memory
k-bit address
n-bit data
Control lines
(R/$\overline{\text{W}}$, etc.)
Up to 2^k addressable locations
Word length = n bits

Cache and Virtual Memory

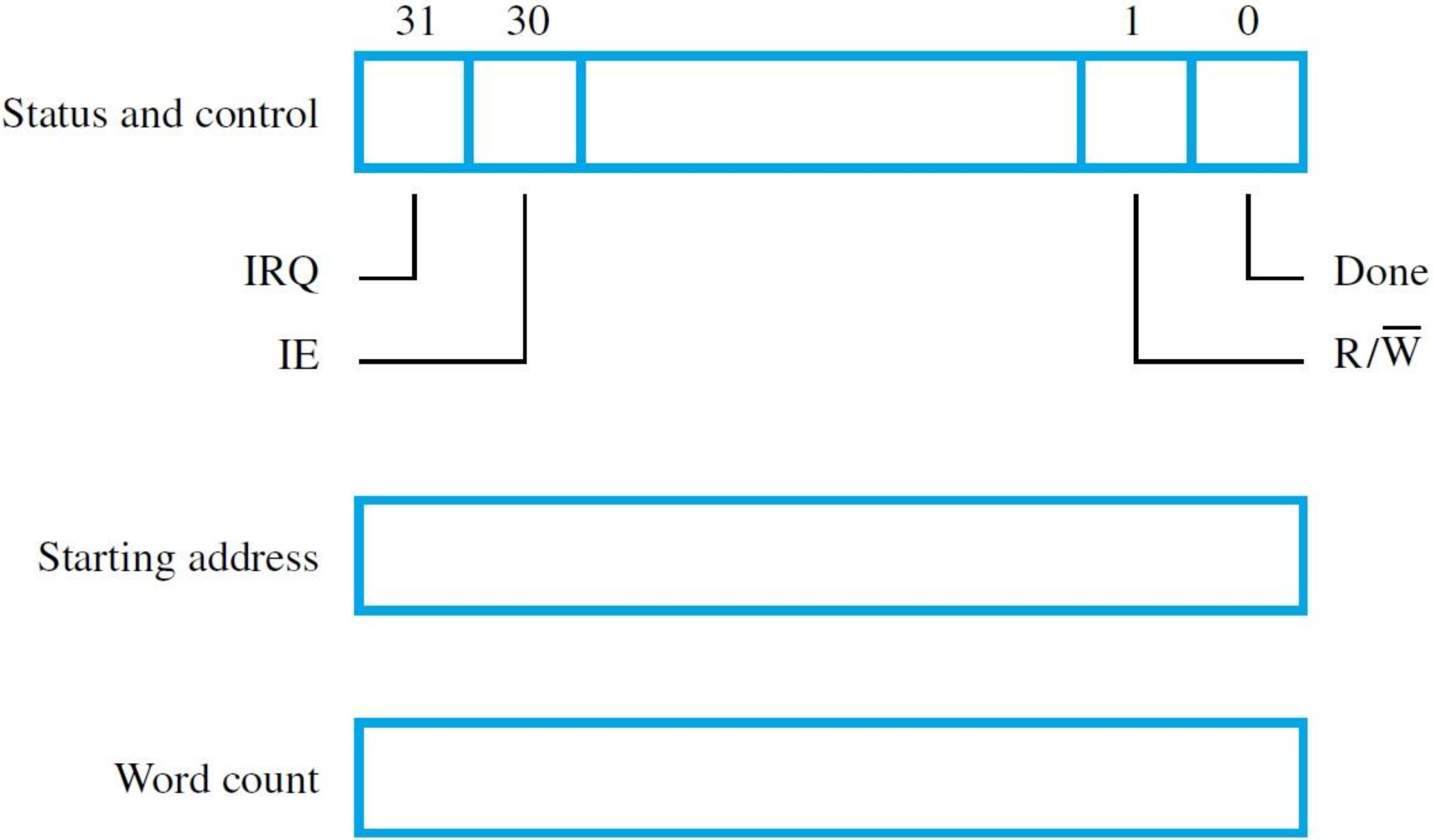
- The main memory is slower than processor
- Cache memory is smaller and faster memory that is used to reduce effective access time
- Holds subset of program instructions and data
- Information for one or more active programs may exceed physical capacity of the memory
- Virtual memory provides larger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

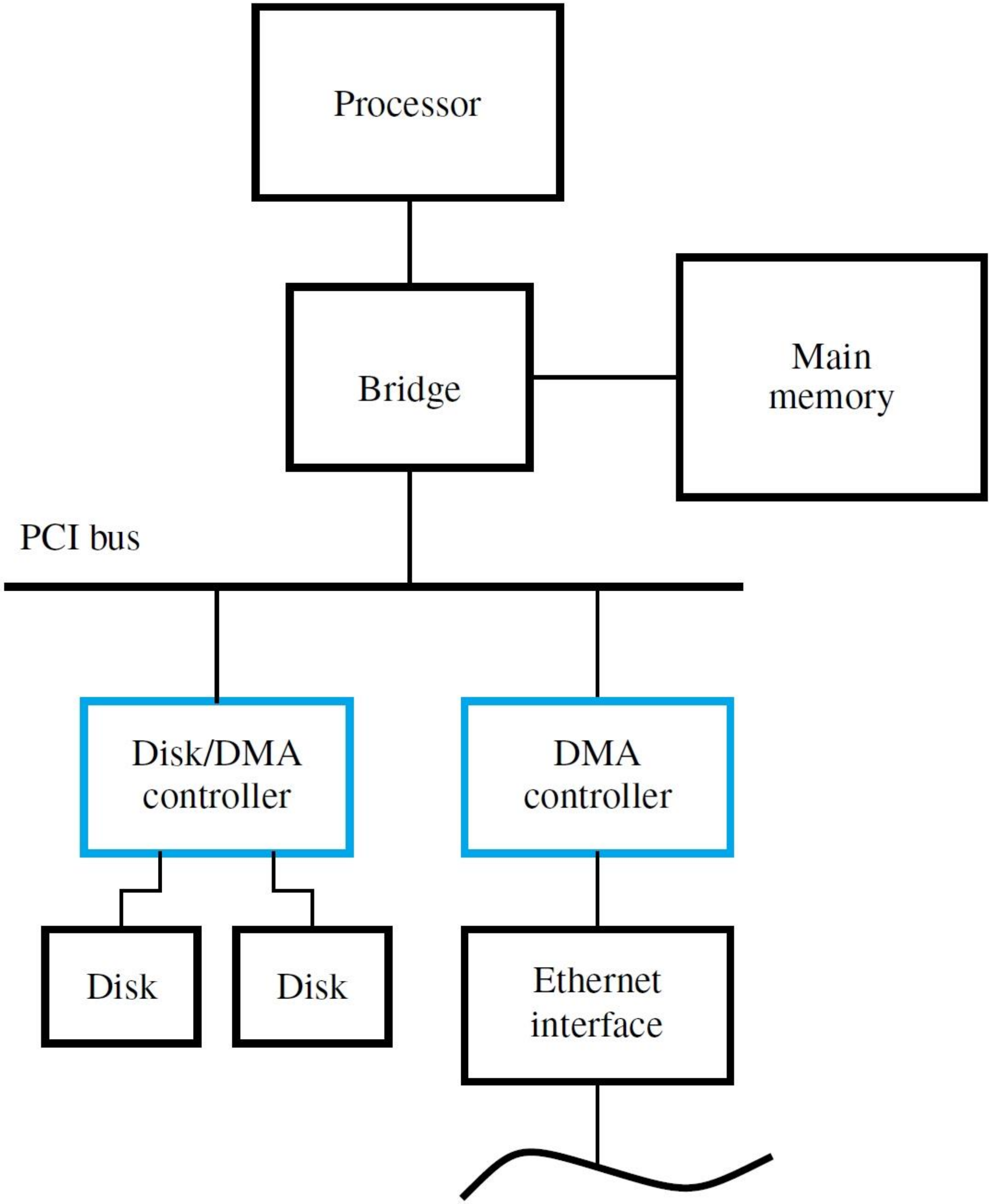
Direct Memory Access

- 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

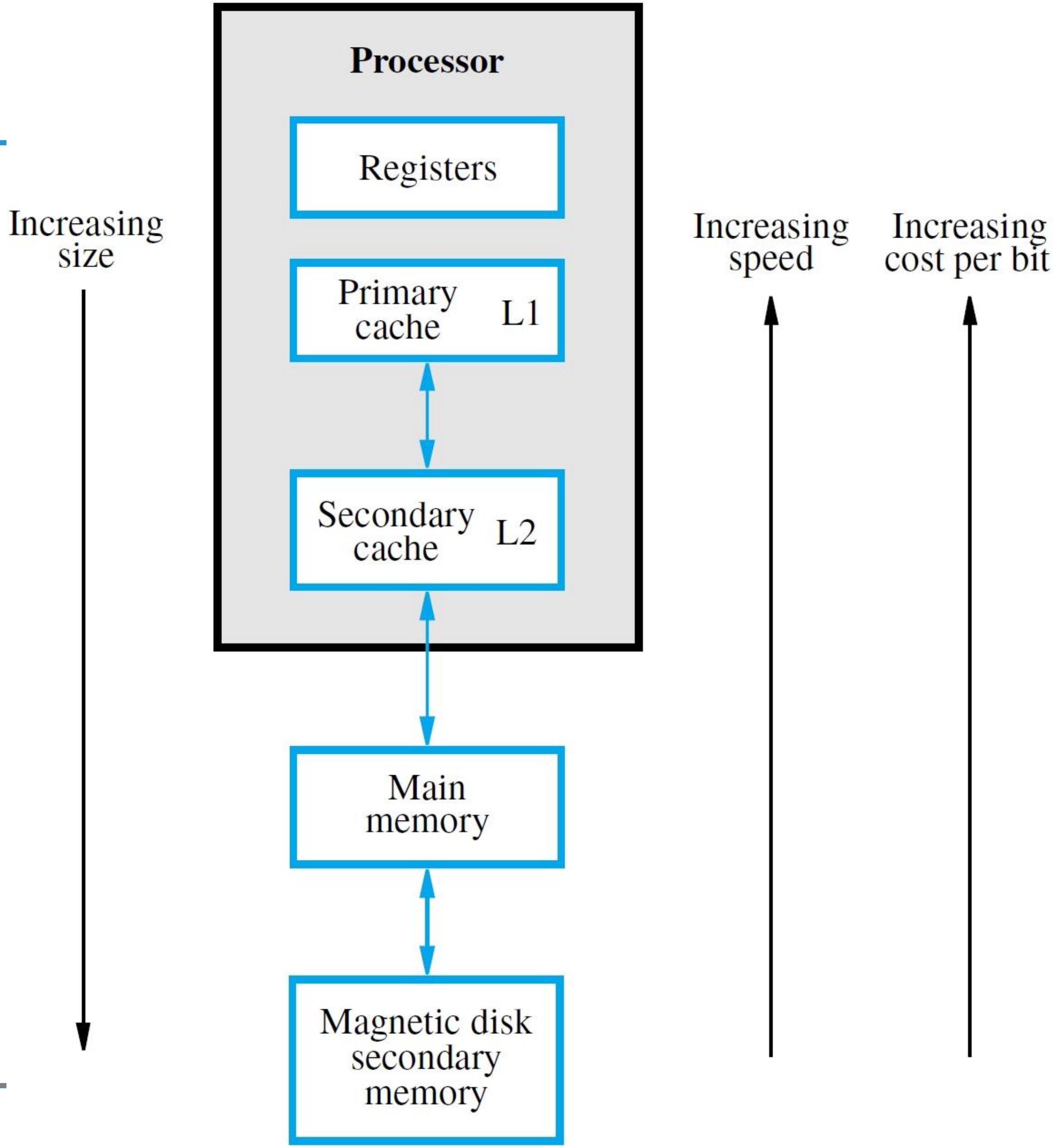
- *DMA controller* is shared, or in each I/O device
- Performs individual memory accesses that would have been done by the processor
- Keeps track of progress with address counter
- Processor initiates DMA controller activity after writing information to special registers (starting address, count, Read/Write, 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

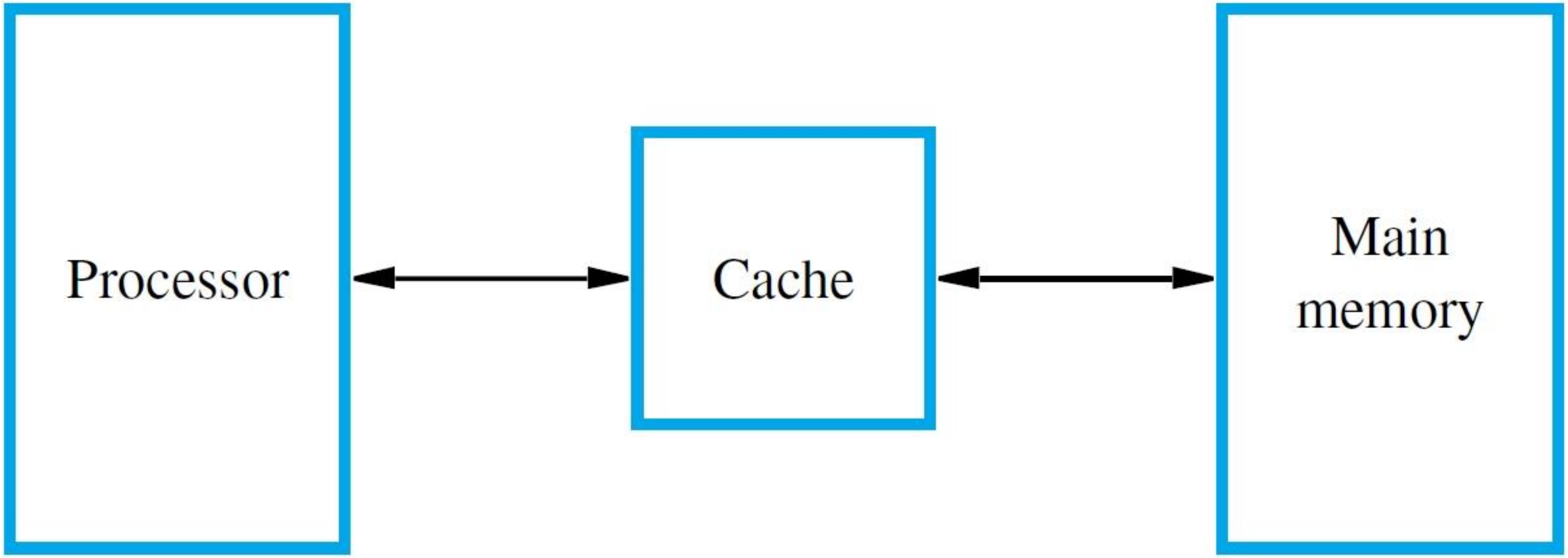
Increasing cost per bit

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

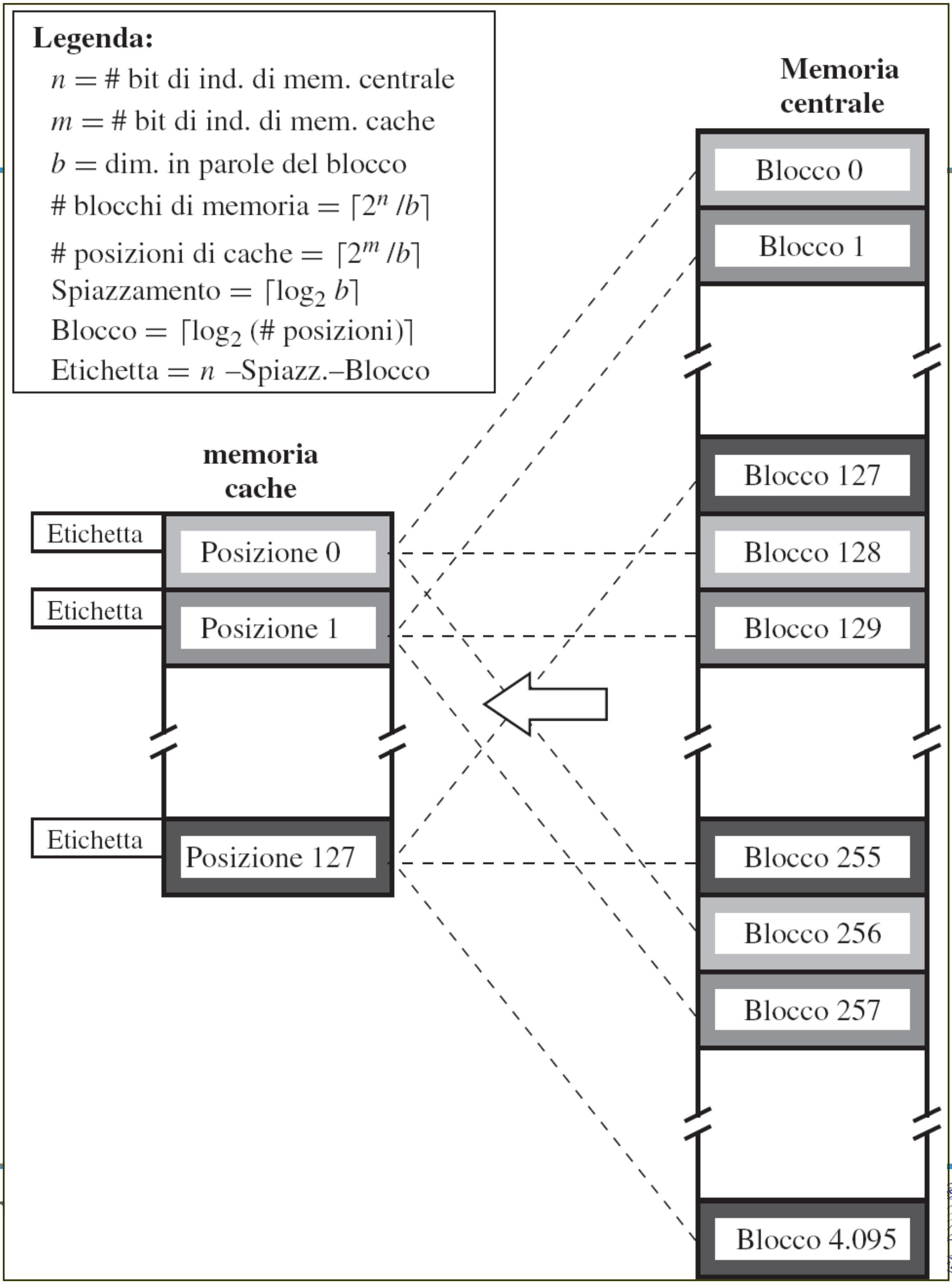
Mapping Functions

- 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* → 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



Legenda:

n = # bit di ind. di mem. centrale

m = # bit di ind. di mem. cache

b = dim. in parole del blocco

blocchi di memoria = $\lceil 2^n / b \rceil$

posizioni di cache = $\lceil 2^m / b \rceil$

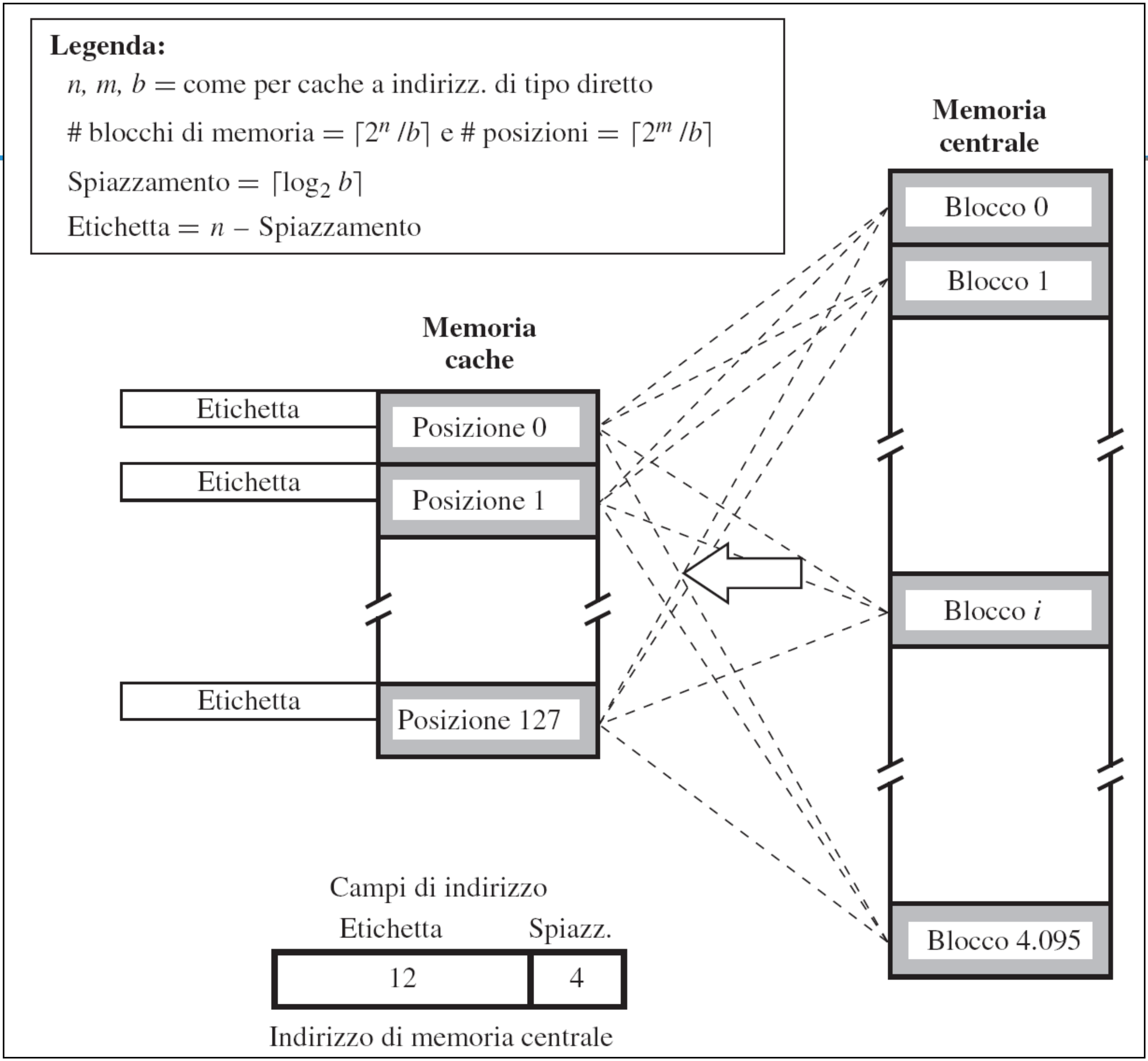
Spiazzamento = $\lceil \log_2 b \rceil$

Blocco = $\lceil \log_2$ (# posizioni)$\rceil$

Etichetta = n –Spiazz.–Blocco

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 = $\lceil 2^n / b \rceil$ e # posizioni = $\lceil 2^m / b \rceil$

Spiazzamento = $\lceil \log_2 b \rceil$

Etichetta = n – Spiazzamento

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 → *k*-way set-associative cache
- Direct-mapped = 1-way; associative = all-way

Legenda:

n, m, b = come per cache a indirizz. di tipo diretto

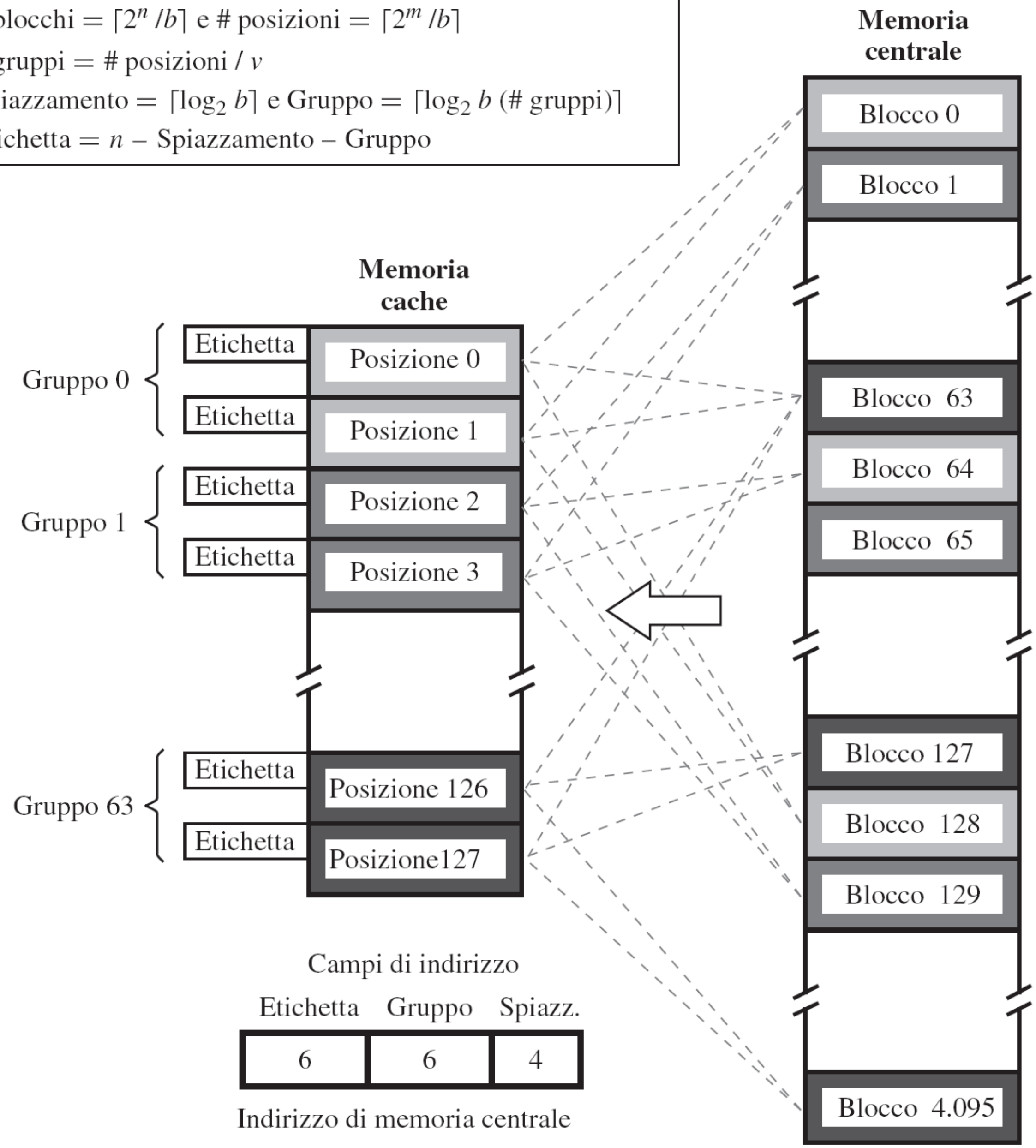
v = dim. in posizioni del gruppo = # vie della cache

blocchi = $\lceil 2^n / b \rceil$ e # posizioni = $\lceil 2^m / b \rceil$

gruppi = # posizioni / v

Spiazzamento = $\lceil \log_2 b \rceil$ e Gruppo = $\lceil \log_2 b$ (# gruppi)$\rceil$

Etichetta = n – Spiazzamento – Gruppo



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

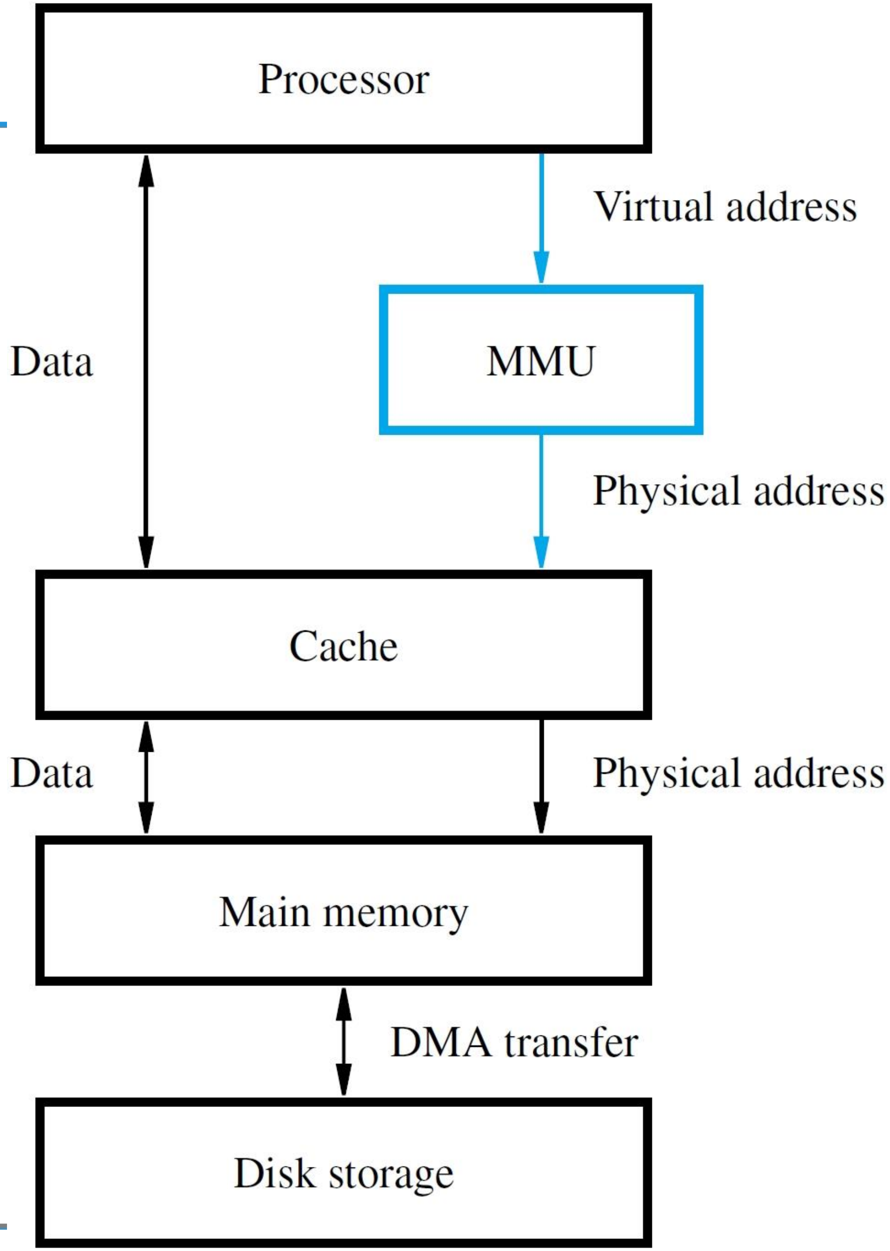
- Physical mem. capacity $\leq$ address space size
- A large program or many active programs may not be entirely resident in the main memory
- Use secondary storage (e.g., magnetic disk) to hold portions exceeding memory capacity
- Needed portions are *automatically* loaded into the memory, replacing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

Virtual Memory

- Programs written assuming full address space
- Processor issues *virtual* or *logical address*
- Must be translated into *physical address*
- Proceed with normal memory operation
  when addressed contents are in the memory
- When no current physical address exists, perform actions
  to place contents in memory
- System may select any physical address;
  no unique assignment for a virtual address

Memory Management Unit

- Implementation of virtual memory relies on a *memory management unit (MMU)*
- Maintains virtual→physical address mapping to perform the necessary translation
- When no current physical address exists, MMU invokes operating system services
- Causes transfer of desired contents from disk to the main memory using DMA scheme
- MMU mapping information is also updated

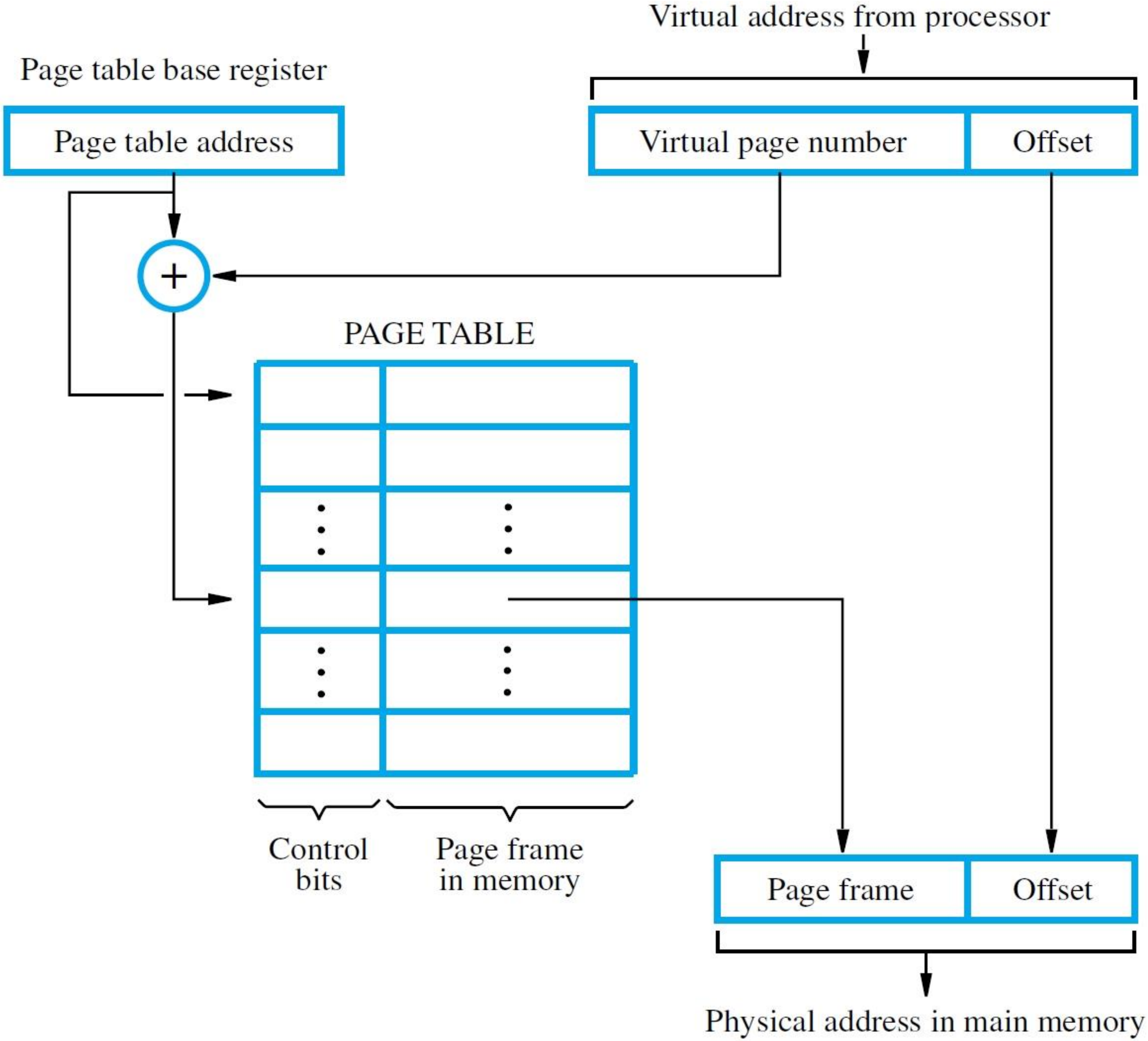

Processor
Virtual address
Data
MMU
Physical address
Cache
Data
Physical address
Main memory
DMA transfer
Disk storage

Address Translation

- Use fixed-length unit of *pages* (2K-16K bytes)
- Larger size than cache blocks due to slow disks
- For translation, divide address bits into 2 fields
- Lower bits give *offset* of word within page
- Upper bits give *virtual page number (VPN)*
- Translation preserves offset bits, but causes VPN bits to be replaced with *page frame* bits
- *Page table* (stored in the main memory) provides information to perform translation

Page Table

- MMU must know location of page table
- *Page table base register* has starting address
- Adding VPN to base register contents gives location of corresponding entry about page
- If page is in memory, table gives frame bits
- Otherwise, table may indicate disk location
- Control bits for each entry include a valid bit and modified bit indicating needed copy-back
- Also have bits for page read/write permissions



Virtual address from processor

Page table base register

Page table address

Virtual page number

Offset

+

PAGE TABLE

Control bits

Page frame in memory

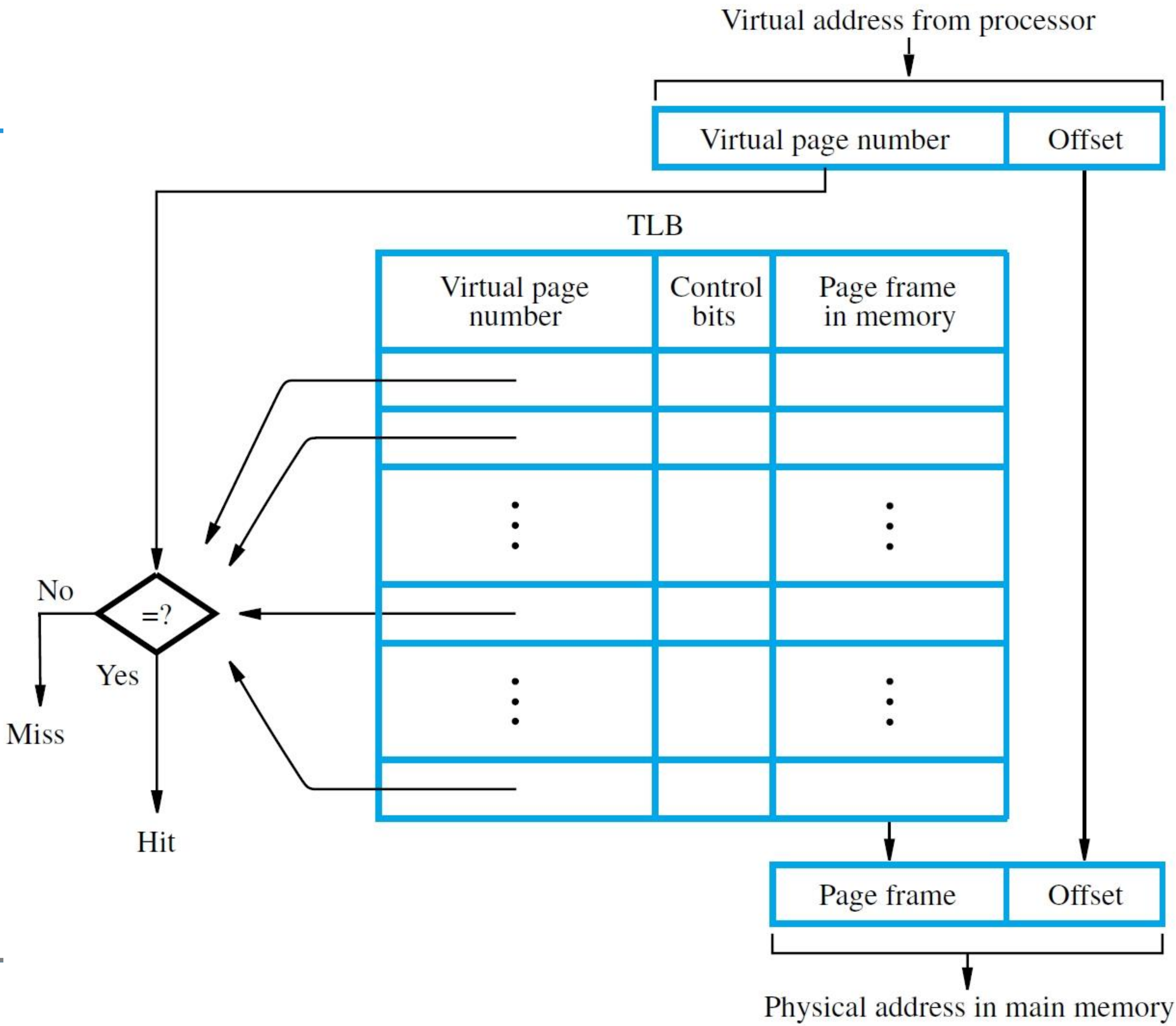
Page frame

Offset

Physical address in main memory

Translation Lookaside Buffer

- MMU must perform lookup in page table for translation of every virtual address
- For large physical memory, MMU cannot hold entire page table with all of its information
- *Translation lookaside buffer (TLB)* in the MMU holds recently-accessed entries of page table
- Associative searches are performed on the TLB with virtual addresses to find matching entries
- If miss in TLB, access full table and update TLB


Virtual address from processor
Virtual page number
Offset
TLB
Virtual page number
Control bits
Page frame in memory
No
=?
Yes
Miss
Hit
Page frame
Offset
Physical address in main memory

Page Faults

- A *page fault* occurs when a virtual address has no corresponding physical address
- MMU raises an interrupt for operating system to place the containing page in the memory
- Operating system selects location using LRU, performing copy-back if needed for old page
- Delay may be long, involving disk accesses, hence another program is selected to execute
- Suspended program restarts later when ready

Concluding Remarks

- Memory hierarchy is an important concept
- Considers speed/capacity/cost issues and also reflects awareness of locality of reference
- Leads to caches and virtual memory
- Importance of block transfers for efficiency