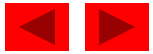
Chapter 10 – Virtual Memory Organization

Outline

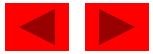
Objectives

- After reading this chapter, you should understand:
  - the concept of virtual memory.
  - paged virtual memory systems.
  - segmented virtual memory systems.
  - combined segmentation/paging virtual memory systems.
  - sharing and protection in virtual memory systems.
  - the hardware that makes virtual memory systems feasible.
  - the IA-32 Intel architecture virtual memory implementation.



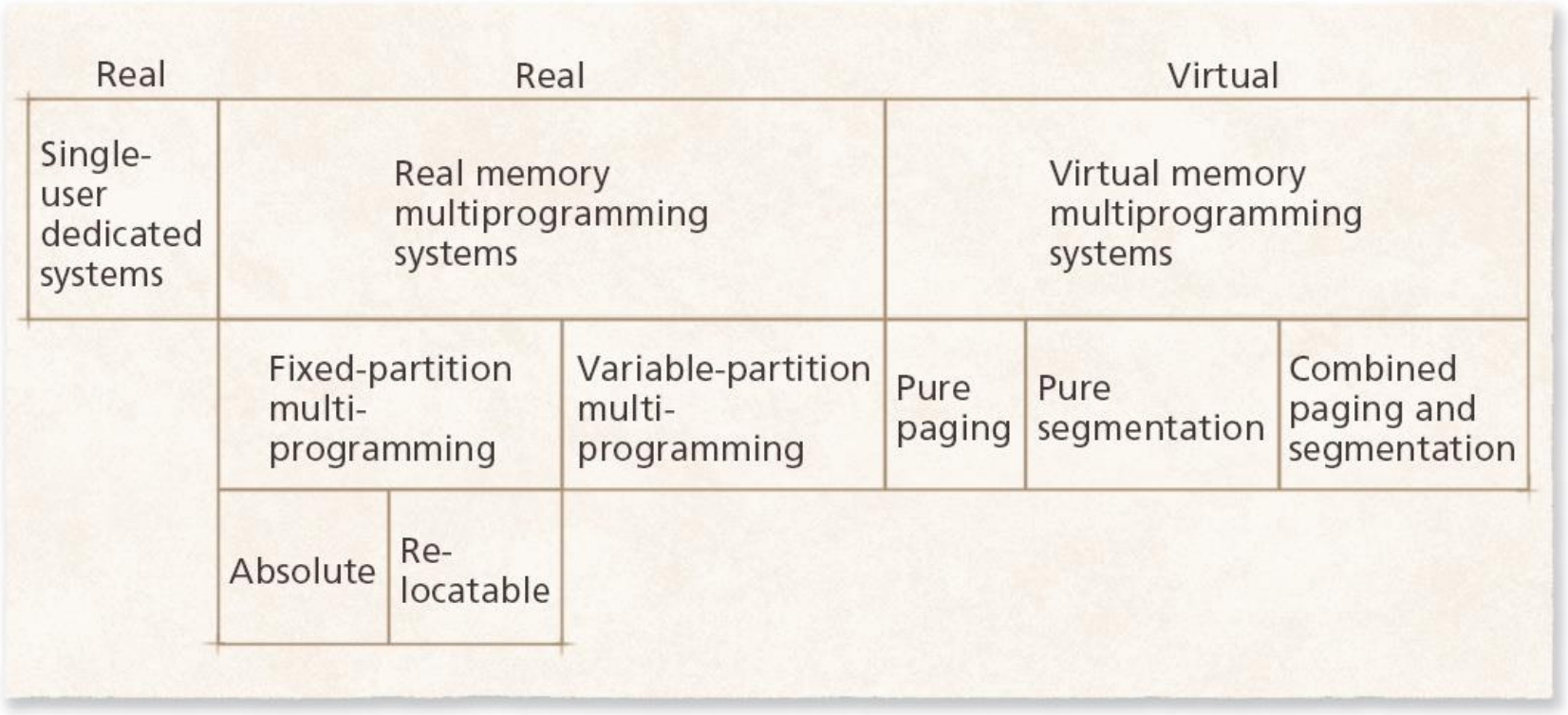
10.1 Introduction

- Virtual memory
  - Solves problem of limited memory space
  - Creates the illusion that more memory exists than is available in system
  - Two types of addresses in virtual memory systems
    - Virtual addresses
      - Referenced by processes
    - Physical addresses
      - Describes locations in main memory
  - Memory management unit (MMU)
    - Translates virtual addresses to physical address



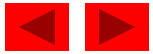
10.1 Introduction

Figure 10.1 Evolution of memory organizations.



10.2 Virtual Memory: Basic Concepts

- Virtual address space, *V*
  - Range of virtual addresses that a process may reference
- Real address space, *R*
  - Range of physical addresses available on a particular computer system
- Dynamic address translation (DAT) mechanism
  - Converts virtual addresses to physical addresses during program execution

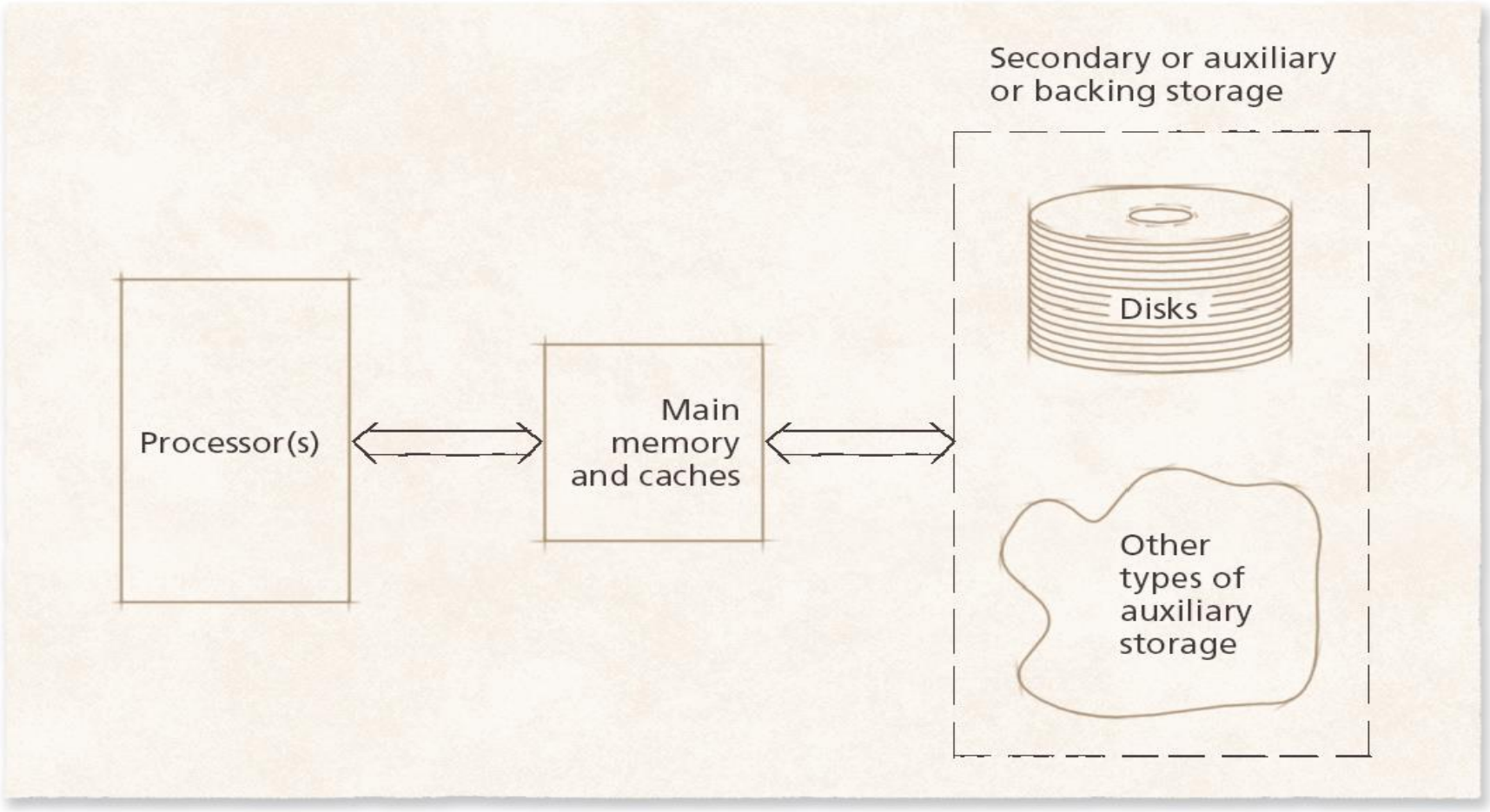


10.2 Virtual Memory: Basic Concepts

- $|V|$ is often much greater than $|R|$
  - OS must store parts of V for each process outside of main memory
  - Two-level storage
    - OS shuttles portions of V between main memory (and caches) and secondary storage

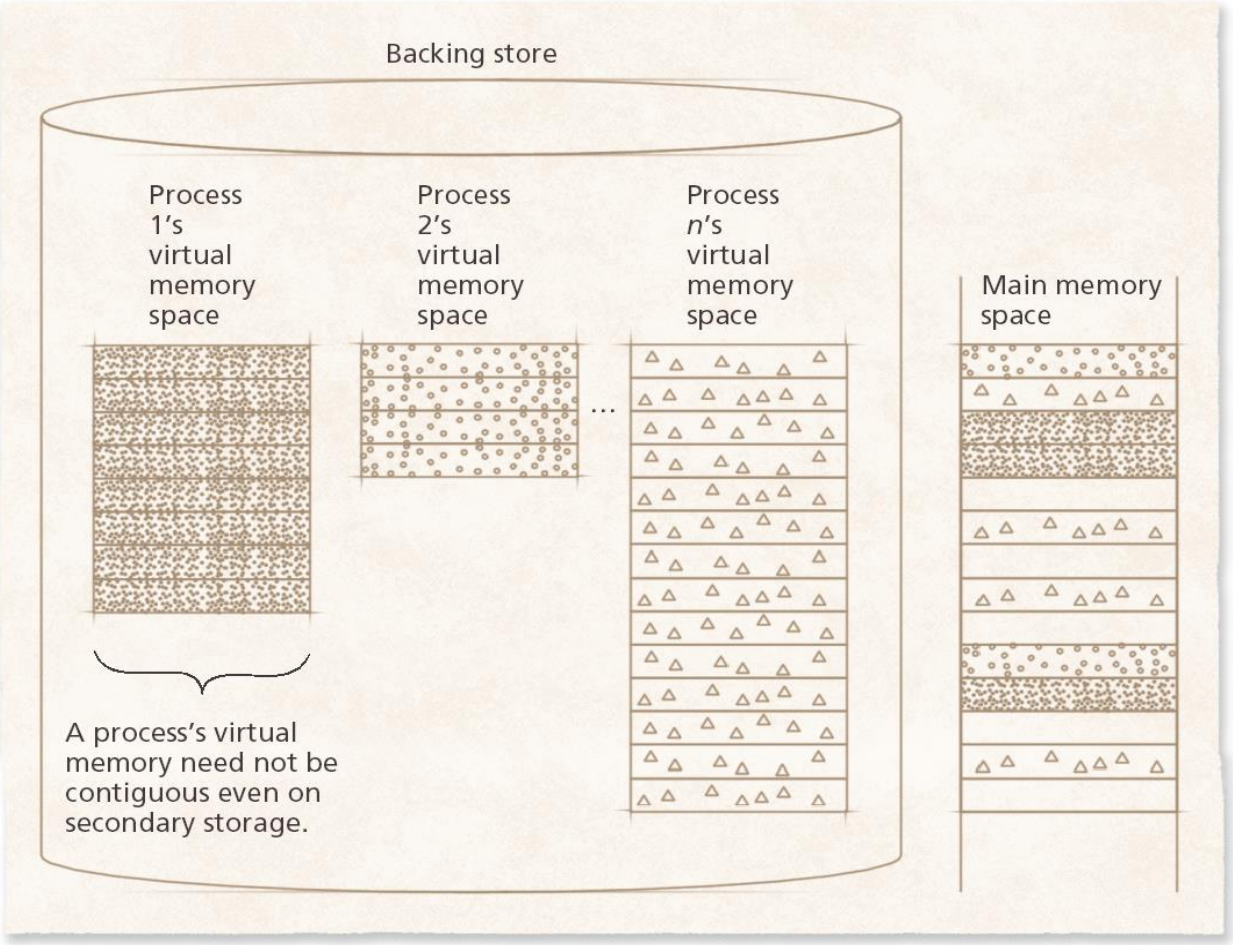
10.2 Virtual Memory: Basic Concepts

Figure 10.2 Two-level storage.



10.2 Virtual Memory: Basic Concepts

Figure 10.3 Pieces of address spaces exist in memory and in virtual storage.

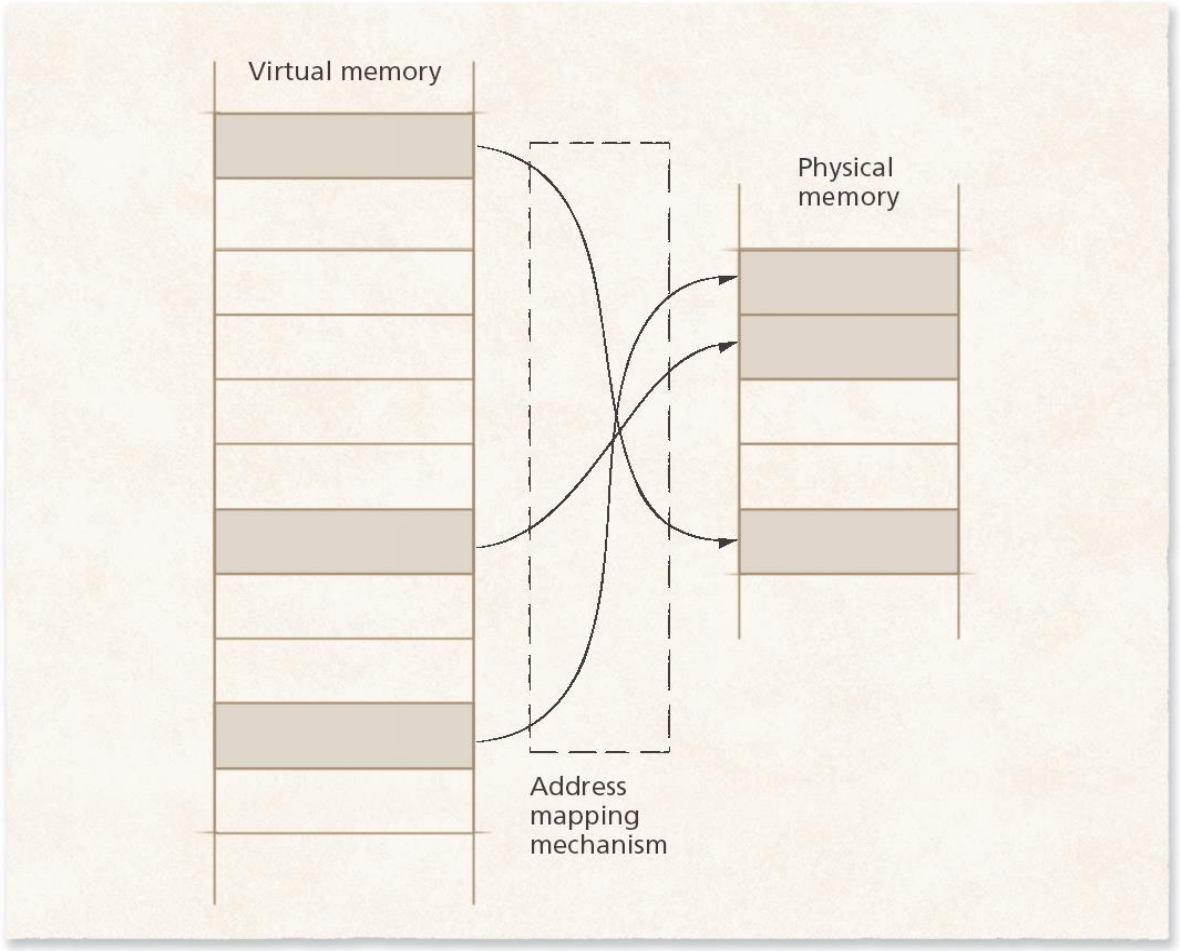


10.3 Block Mapping

- Address translation maps
  - Indicate which regions of a process's virtual address space, *V*, are currently in main memory and where they are located

10.3 Block Mapping

Figure 10.4 Mapping virtual addresses to real addresses.

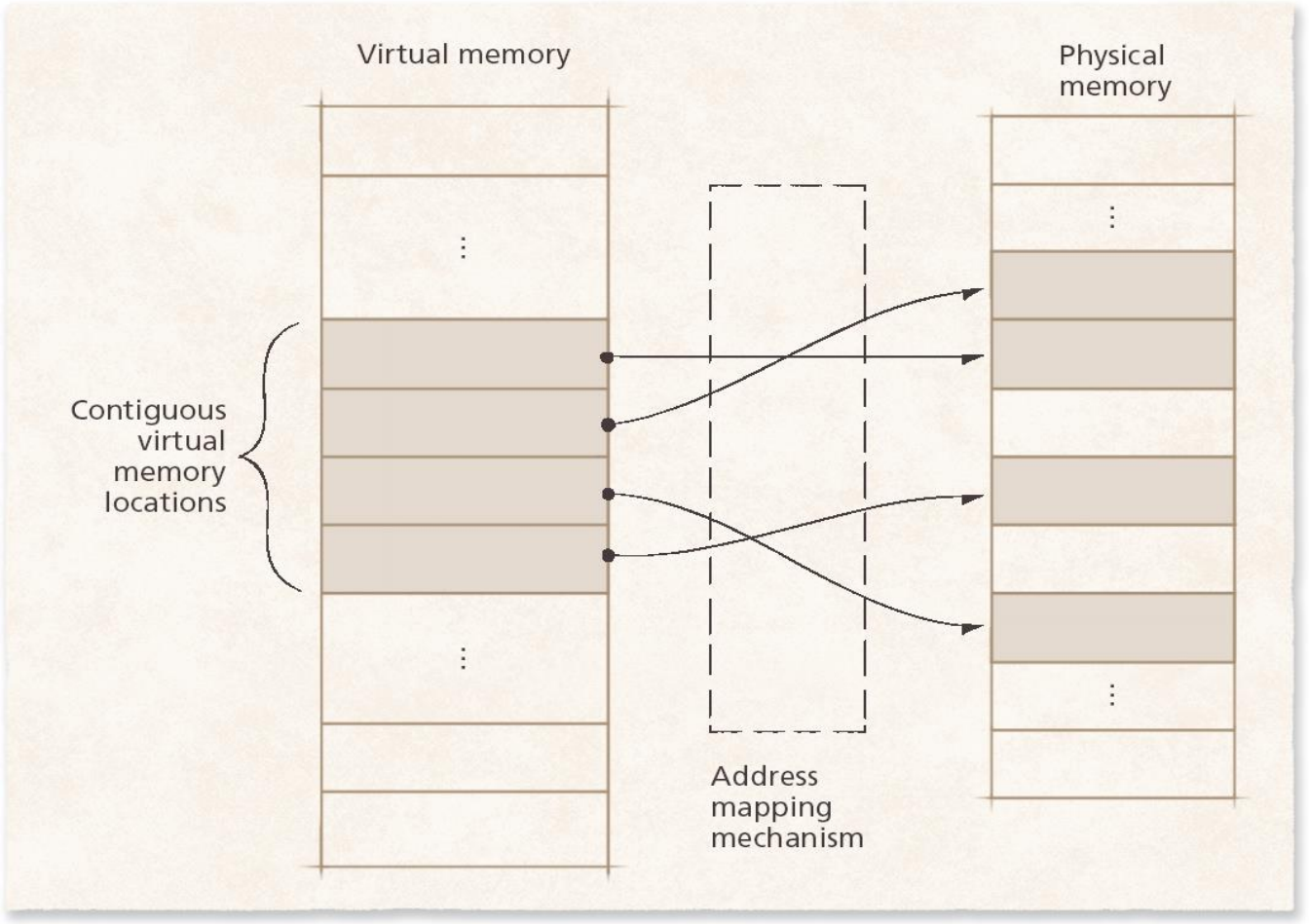


10.3 Block Mapping

- Artificial contiguity
  - Contiguous virtual addresses may not correspond to contiguous real memory addresses

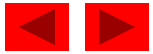
10.3 Block Mapping

Figure 10.5 Artificial contiguity.



10.3 Block Mapping

- Pages
  - Blocks are fixed size
  - Technique is called paging
- Segments
  - Blocks maybe of different size
  - Technique is called segmentation
- Block mapping
  - System represents addresses as ordered pairs

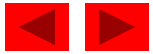


10.3 Block Mapping

Figure 10.6 Virtual address format in a block mapping system.

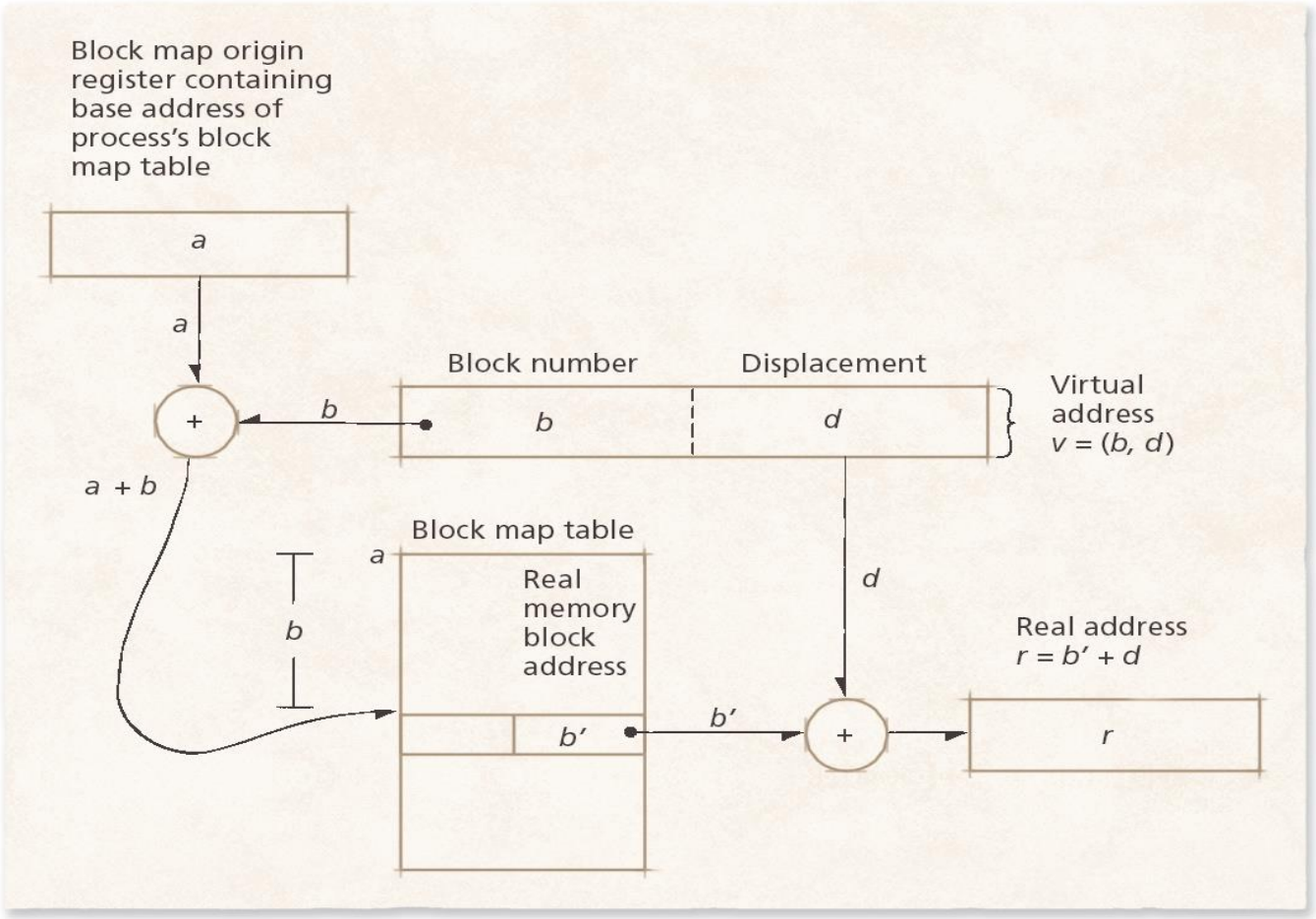
10.3 Block Mapping

- Given a virtual address $v = (b, d)$
  - Block map origin register stored in a
  - Block number, b, is added to a to locate the appropriate entry in the block map table
  - Block map table entry yields the address, b', of the start of block b in main memory
  - Displacement d is added to b' to form the real address, r



10.3 Block Mapping

Figure 10.7 Virtual address translation with block mapping.



10.4 Paging

- Paging uses fixed-size block mapping
  - Virtual address in paging system is an ordered pair $v = (p, d)$
    - p is the number of the page in virtual memory on which the referenced item resides
    - d is the displacement from the start of page p at which the referenced item is located

10.4 Paging

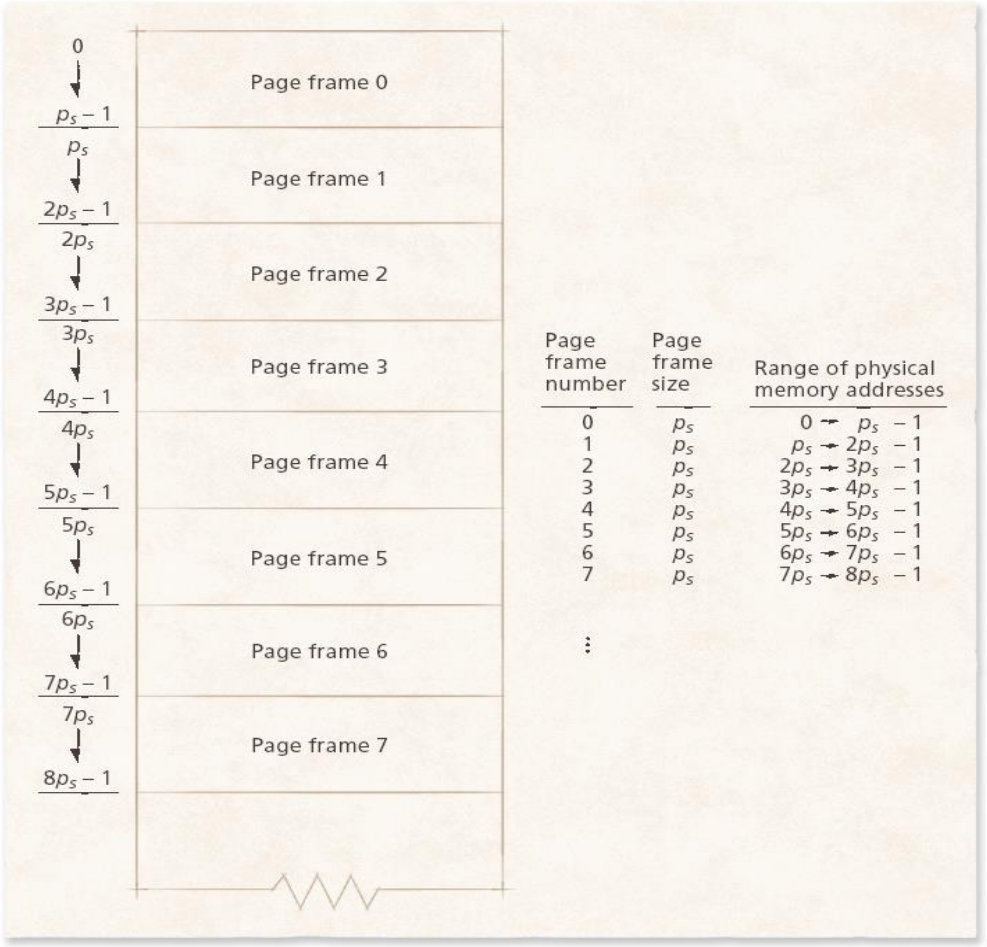
Figure 10.8 Virtual address format in a pure paging system.

10.4 Paging

- Page frame
  - Fixed-size block of main memory
  - Begins at a main memory address that is an integral multiple of fixed page size (p_s)

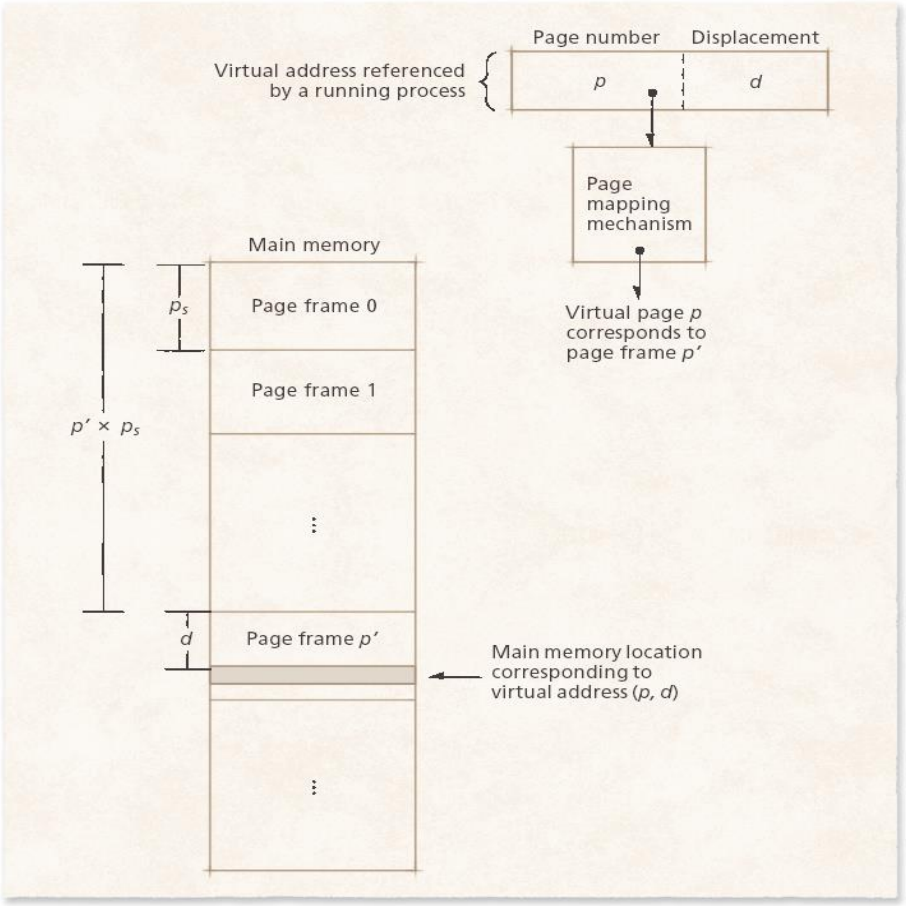
10.4 Paging

Figure 10.9 Main memory divided into page frames.



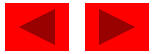
10.4 Paging

Figure 10.10 Correspondence between virtual memory addresses and physical memory addresses in a pure paging system.



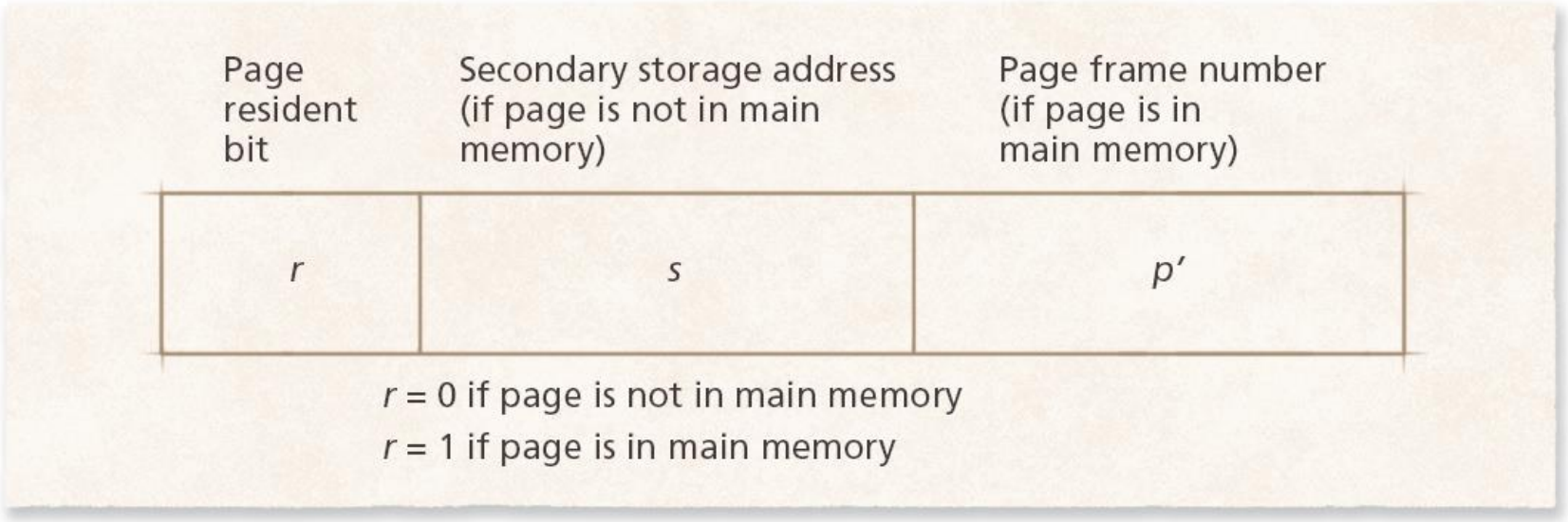
10.4 Paging

- Page table entry (PTE)
  - Indicates that virtual page *p* corresponds to page frame *p´*
  - Contains a resident bit to indicate if page is in memory
    - If so, PTE stores the page's frame number
    - Otherwise, PTE stores the location of the page on secondary storage



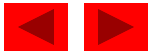
10.4 Paging

Figure 10.11 Page table entry.



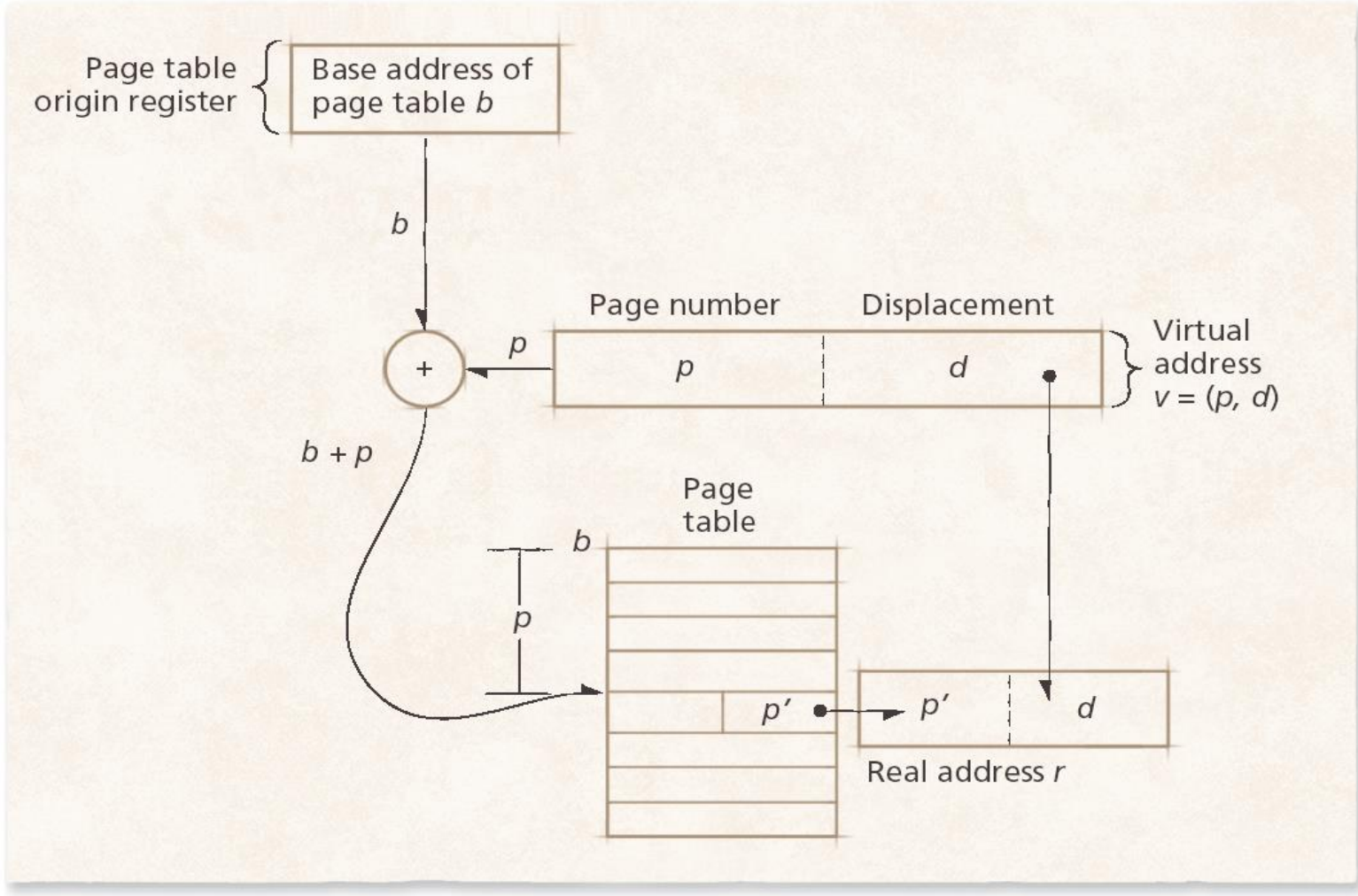
10.4.1 Paging Address Translation by Direct Mapping

- Direct mapping
  - Dynamic address translation under paging is similar to block address translation
  - Process references virtual address $v = (p, d)$
    - DAT adds the process's page table base address, b, to referenced page number, p
    - $b + p$ forms the main memory address of the PTE for page p
    - System concatenates p' with displacement, d, to form real address, r



10.4.1 Paging Address Translation by Direct Mapping

Figure 10.12 Paging address translation by direct mapping.

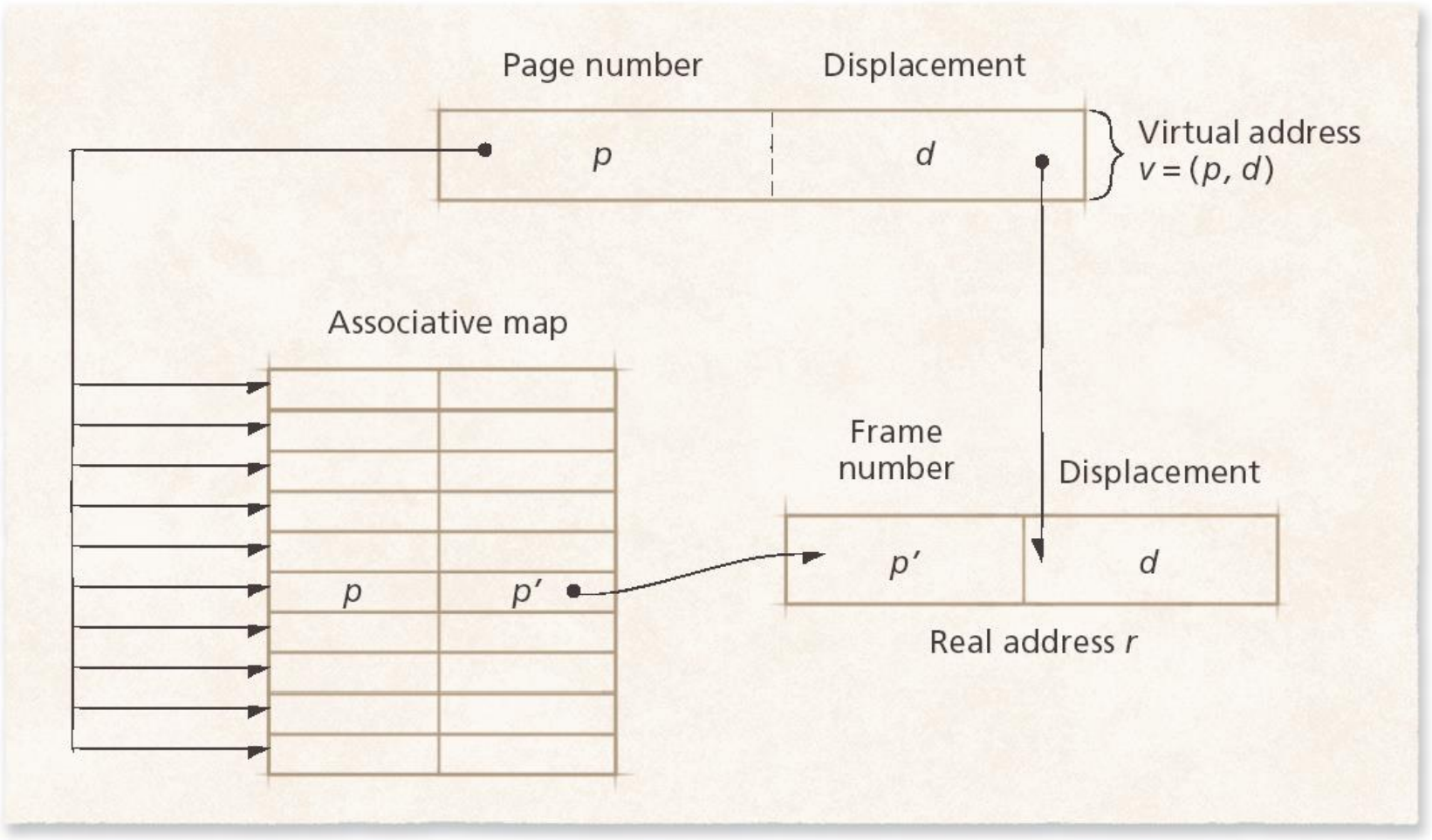


10.4.2 Paging Address Translation by Associative Mapping

- Maintaining entire page table in cache memory is often not viable
  - Due to cost of high-speed, location-addressed cache memory and relatively large size of programs
- Increase performance of dynamic address translation
  - Place entire page table into content-addressed associative memory
  - Every entry in associative memory is searched simultaneously
  - Content-addressed cache memory is also prohibitively expensive

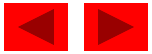
10.4.2 Paging Address Translation by Associative Mapping

Figure 10.13 Paging address translation with pure associative mapping.



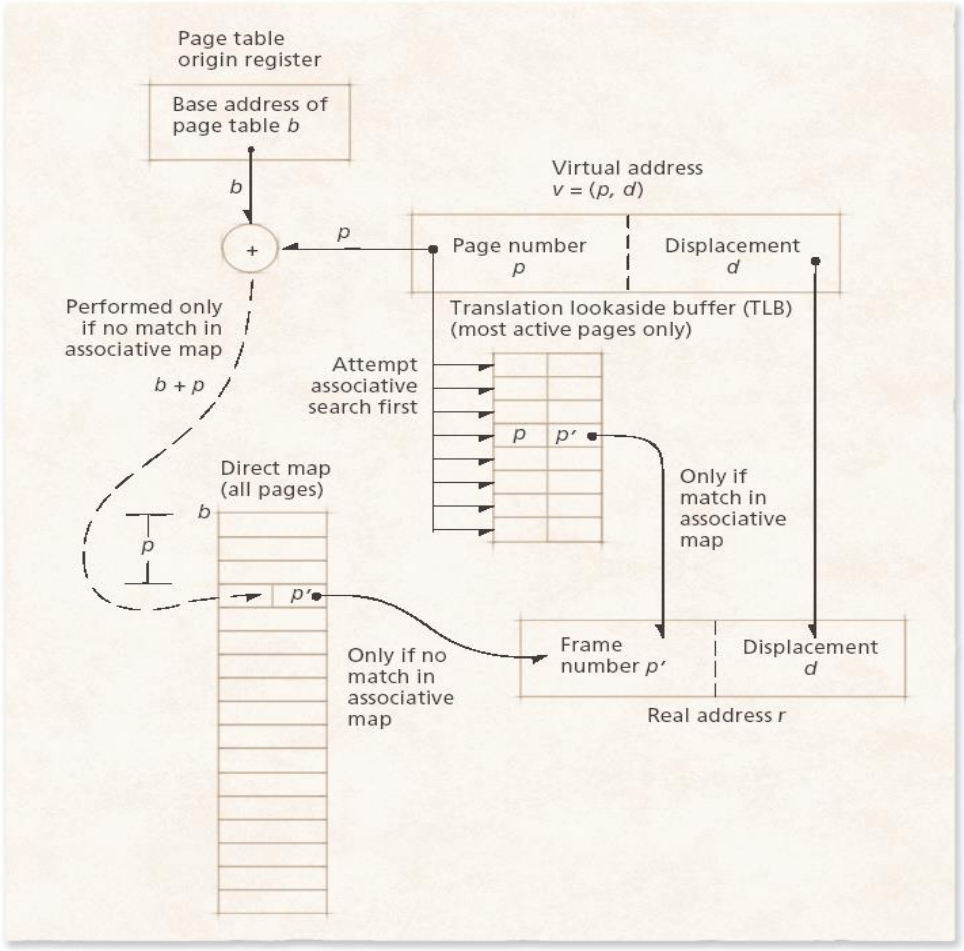
10.4.3 Paging Address Translation with Direct/Associative Mapping

- Compromise between cost and performance
  - Most PTEs are stored in direct-mapped tables in main memory
  - Most-recently-used PTEs are stored in high-speed set-associative cache memory called a Translation Lookaside Buffer (TLB)
    - If PTE is not found in TLB, the DAT mechanism searches the table in main memory
  - Can yield high performance with relatively small TLB due to locality
    - A page referenced by a process recently is likely to be referenced again soon



10.4.3 Paging Address Translation with Direct/Associative Mapping

Figure 10.14 Paging address translation with combined associative/direct mapping.

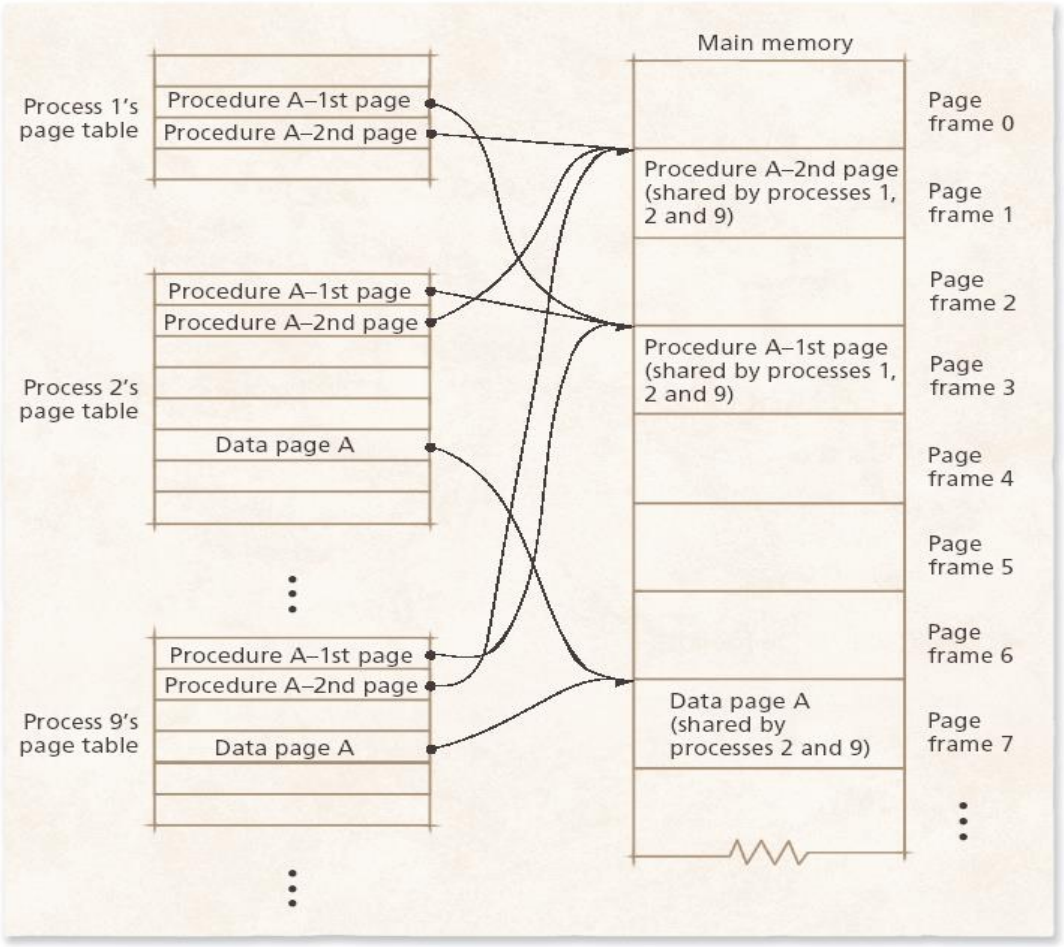


10.4.6 Sharing in a Paging System

- Sharing in multiprogramming systems
  - Reduces memory consumed by programs that use common data and/or instructions
  - Requires system identify each page as sharable or nonsharable

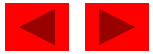
10.4.6 Sharing in a Paging System

Figure 10.18 Sharing in a pure paging system.



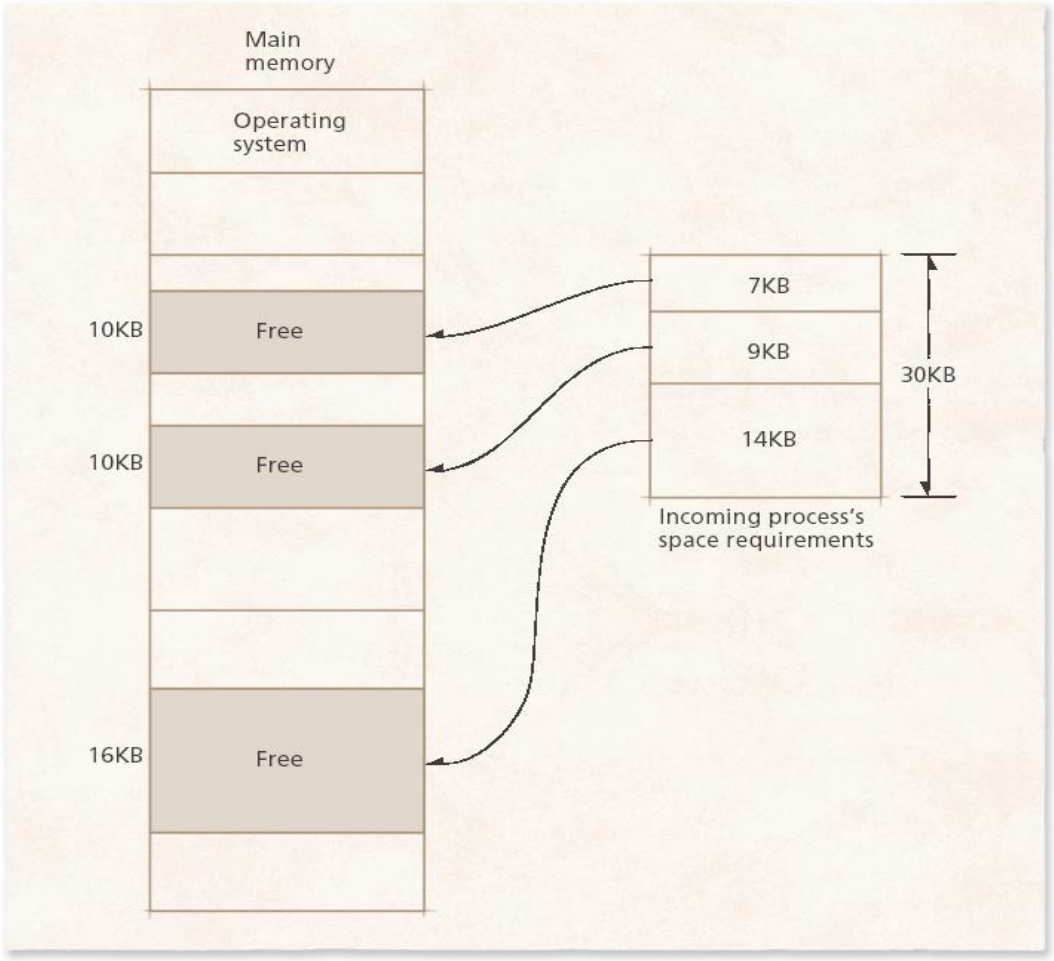
10.5 Segmentation

- Segment
  - Block of program's data and/or instructions
  - Contains meaningful portion of the program (e.g. procedure, array, stack)
  - Consists of contiguous locations
  - Segments need not be the same size nor must they be adjacent to one another in main memory
- A process may execute while its current instructions and referenced data are in segments in main memory



10.5 Segmentation

Figure 10.19 Noncontiguous memory allocation in a real memory segmentation system.



10.5 Segmentation

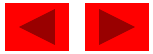
- Process references a virtual memory address $v = (s, d)$
  - s is the segment number in virtual memory
  - d is the displacement within segment s at which the referenced item is located

10.5 Segmentation

Figure 10.20 Virtual address format in a pure segmentation system.

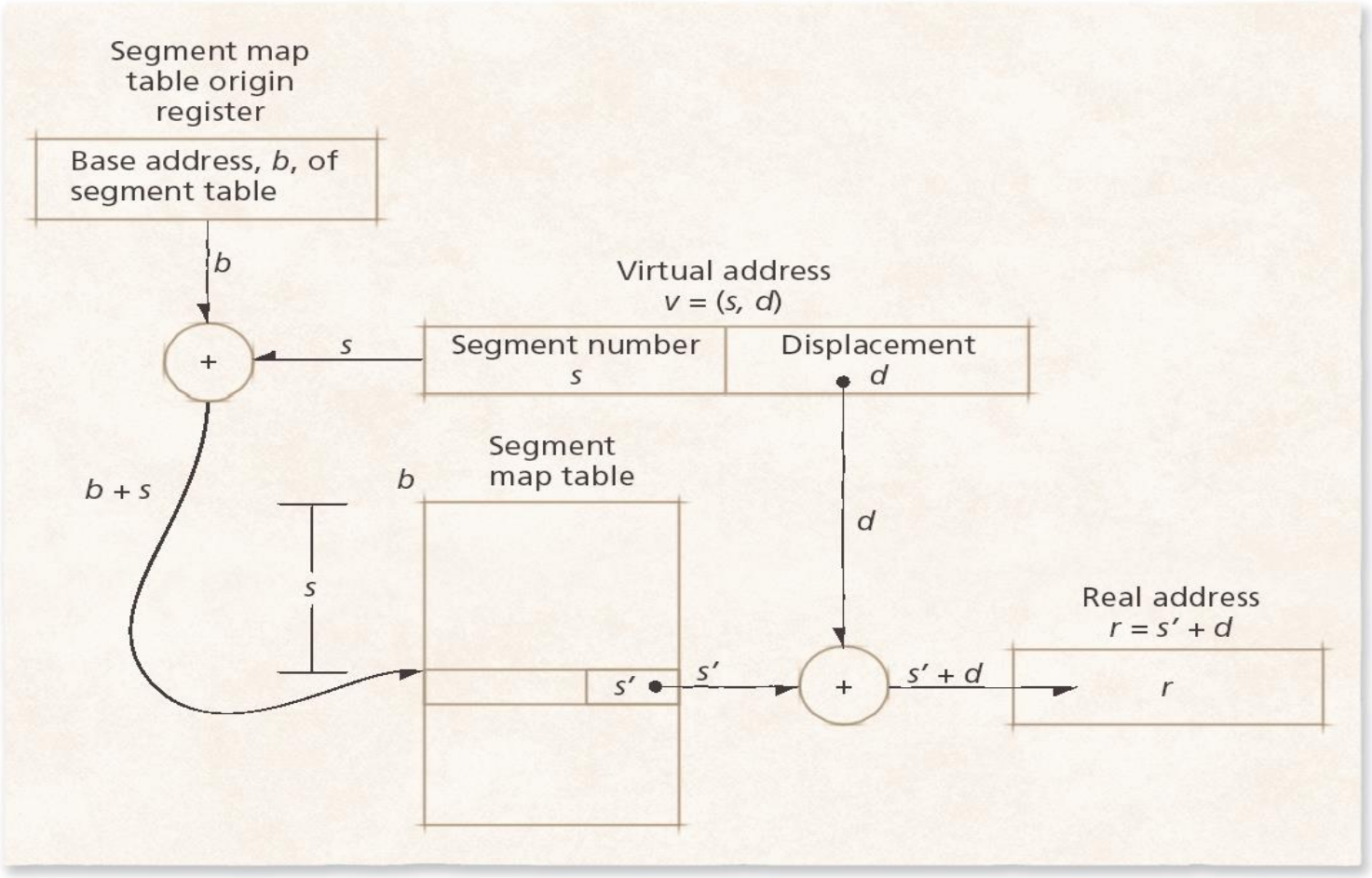
10.5.1 Segmentation Address Translation by Direct Mapping

- Process references a virtual memory address $v = (s, d)$
  - DAT adds the process's segment map table base address, b, to referenced segment number, s
  - $b + s$ forms the main memory address of the segment map table entry for segment s
  - System adds s' to the displacement, d, to form real address, r



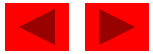
10.5.1 Segmentation Address Translation by Direct Mapping

Figure 10.21 Virtual address translation in a pure segmentation system.



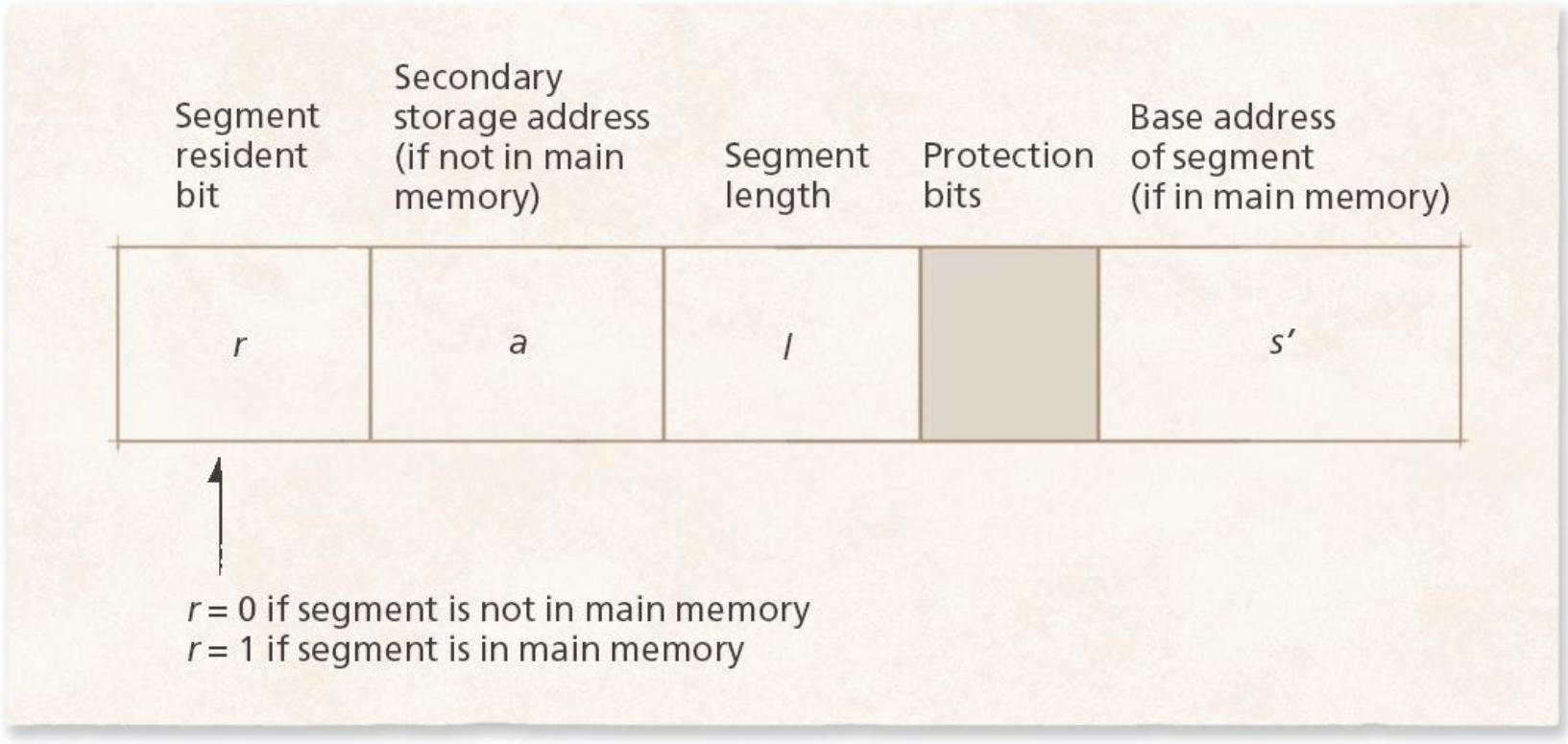
10.5.1 Segmentation Address Translation by Direct Mapping

- Segment map table entry
  - Indicates that segment *s* starts at real memory address *s´*
  - Contains a resident bit to indicate if segment is in memory
    - If so, it stores the segment base address
    - Otherwise, it stores the location of the segment on secondary storage
  - Also contains a length field that indicates the size of the segment
    - Can be used to prevent a process from referencing addresses outside the segment



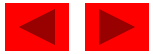
10.5.1 Segmentation Address Translation by Direct Mapping

Figure 10.22 Segment map table entry.



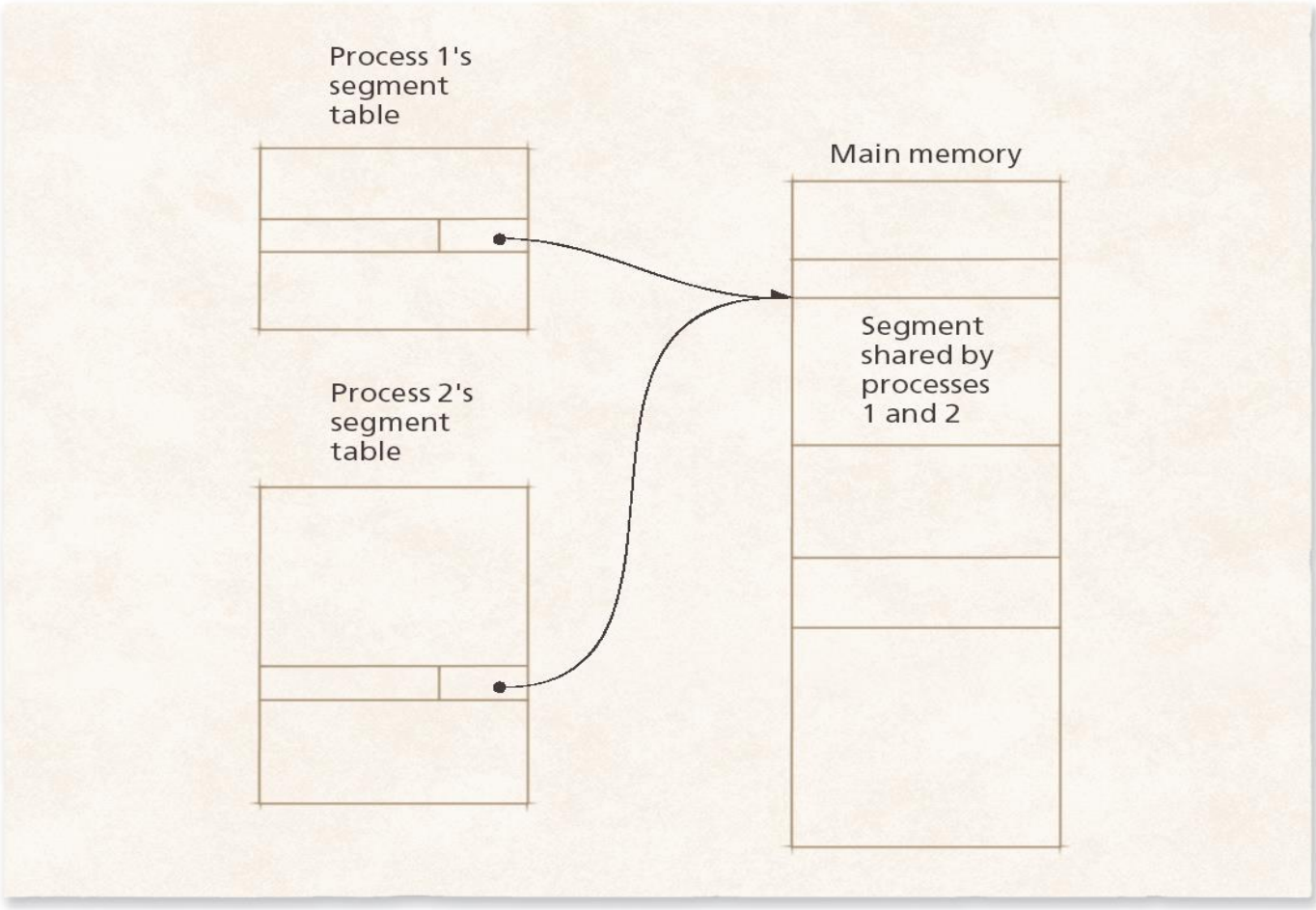
10.5.2 Sharing in a Segmentation System

- Sharing segments can incur less overhead than sharing in direct-mapped pure paging system
  - Potentially fewer map table entries need to be shared



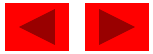
10.5.2 Sharing in a Segmentation System

Figure 10.23 Sharing in a pure segmentation system.



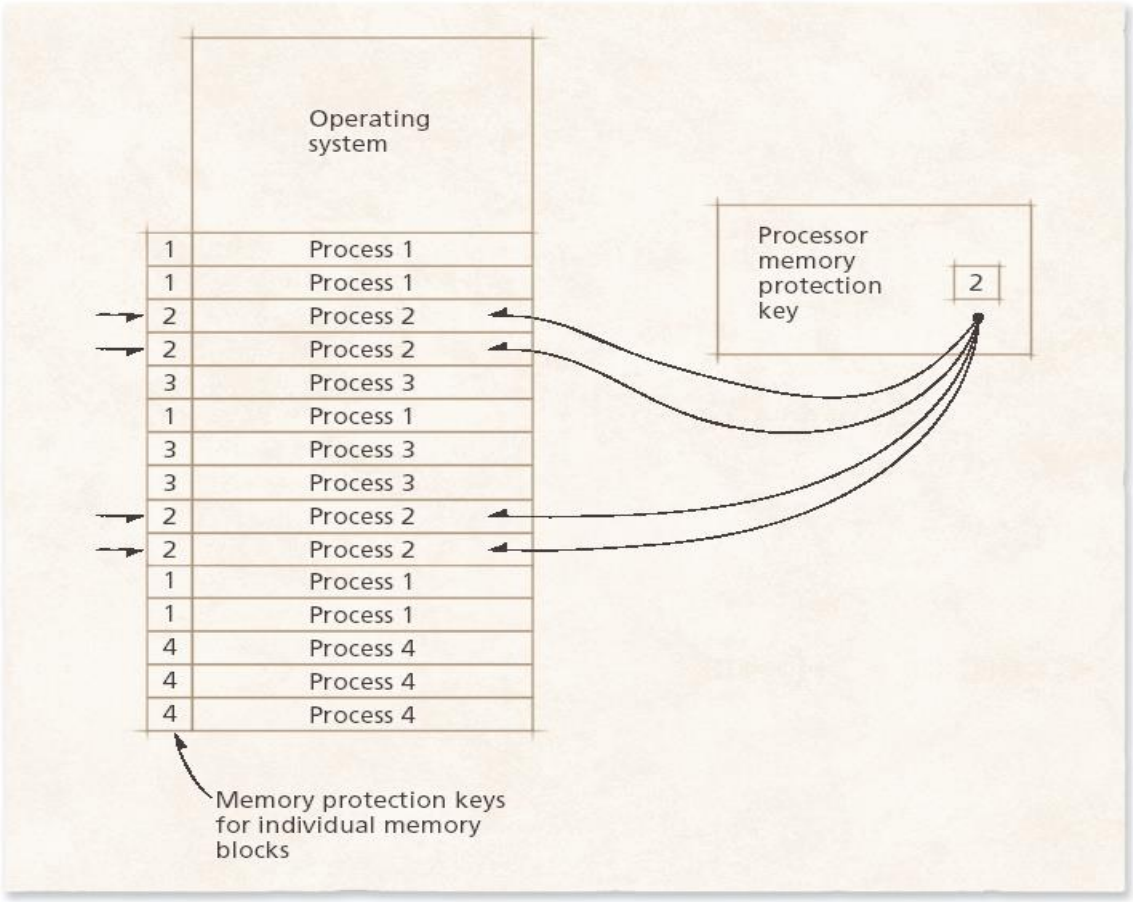
10.5.3 Protection and Access Control in Segmentation Systems

- One scheme for implementing memory protection in segmentation systems is memory protection keys
- Protection key
  - Associated with process
- If protection key for the processor and the requested block are the same, the process can access the segment



10.5.3 Protection and Access Control in Segmentation Systems

Figure 10.24 Memory protection with keys in noncontiguous memory allocation multiprogramming systems.



10.5.3 Protection and Access Control in Segmentation Systems

- A more common scheme is to use protection bits that specify whether a process can read, write, execute code or append to a segment

10.5.3 Protection and Access Control in Segmentation Systems

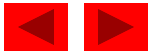
Figure 10.25 Access control types.

Type of access	Abbreviation	Description
Read	R	This segment may be read.
Write	W	This segment may be modified.
Execute	E	This segment may be executed.
Append	A	This segment may have information added to its end.

10.5.3 Protection and Access Control in Segmentation Systems

Figure 10.26 Combining read, write and execute access to yield useful access control modes.

Mode	Read	Write	Execute	Description	Application
Mode 0	No	No	No	No access permitted	Security.
Mode 1	No	No	Yes	Execute only	A segment made available to processes that cannot modify it or copy it, but that can run it.
Mode 2	No	Yes	No	Write only	These possibilities are not useful, because granting write access without read access is impractical.
Mode 3	No	Yes	Yes	Write/execute but cannot be read	
Mode 4	Yes	No	No	Read only	Information retrieval.
Mode 5	Yes	No	Yes	Read/execute	A program can be copied or executed but cannot be modified.
Mode 6	Yes	Yes	No	Read/write but no execution	Protects data from an erroneous attempt to execute it.
Mode 7	Yes	Yes	Yes	Unrestricted access	This access is granted to trusted users.



10.5.3 Protection and Access Control in Segmentation Systems

- Protection bits are added to the segment map table entry and checked when a process references an address
  - If the segment is not in memory, a segment-missing fault is generated
  - If $d > l$, a segment-overflow exception is generated
  - If the operation (e.g., read, write, execute, append) is not allowed, a segment-protection exception is generated

10.5.3 Protection and Access Control in Segmentation Systems

Figure 10.27 Segment map table entry with protection bits.