Operating systems

Lecture 7

Michal Vrábel, 27/11/2019

Segmentation

- Division of memory into **segments**
- Segments have varying length
- Logical address – 2 parts
- Similar to dynamic partitioning, but
  - Program may occupy more partitions/segments
  - Segments are not contiguous
- No internal fragmentation, present external fragmentation
- Visible to the programmer - convenience for organizing programs and data
  - Typically, the programmer or compiler will assign programs and data to different segments.
  - The programmer must be aware of the maximum segment size limitation.
- Segment table – starting address, length of a segment
- List of free blocks in memory
- Each process has its own segment table, and when all of its segments are loaded into main memory, the segment table for a process is created and loaded into main memory

Buffer Overflow Attacks

- buffer overflow/overrun:
  - A condition at an interface under which **more input can be placed into a buffer or data-holding area than the capacity allocated, overwriting other information**. Attackers exploit such a condition to crash a system or to insert specially crafted code that allows them to gain control of the system.
- common type of buffer overflow - stack overflow

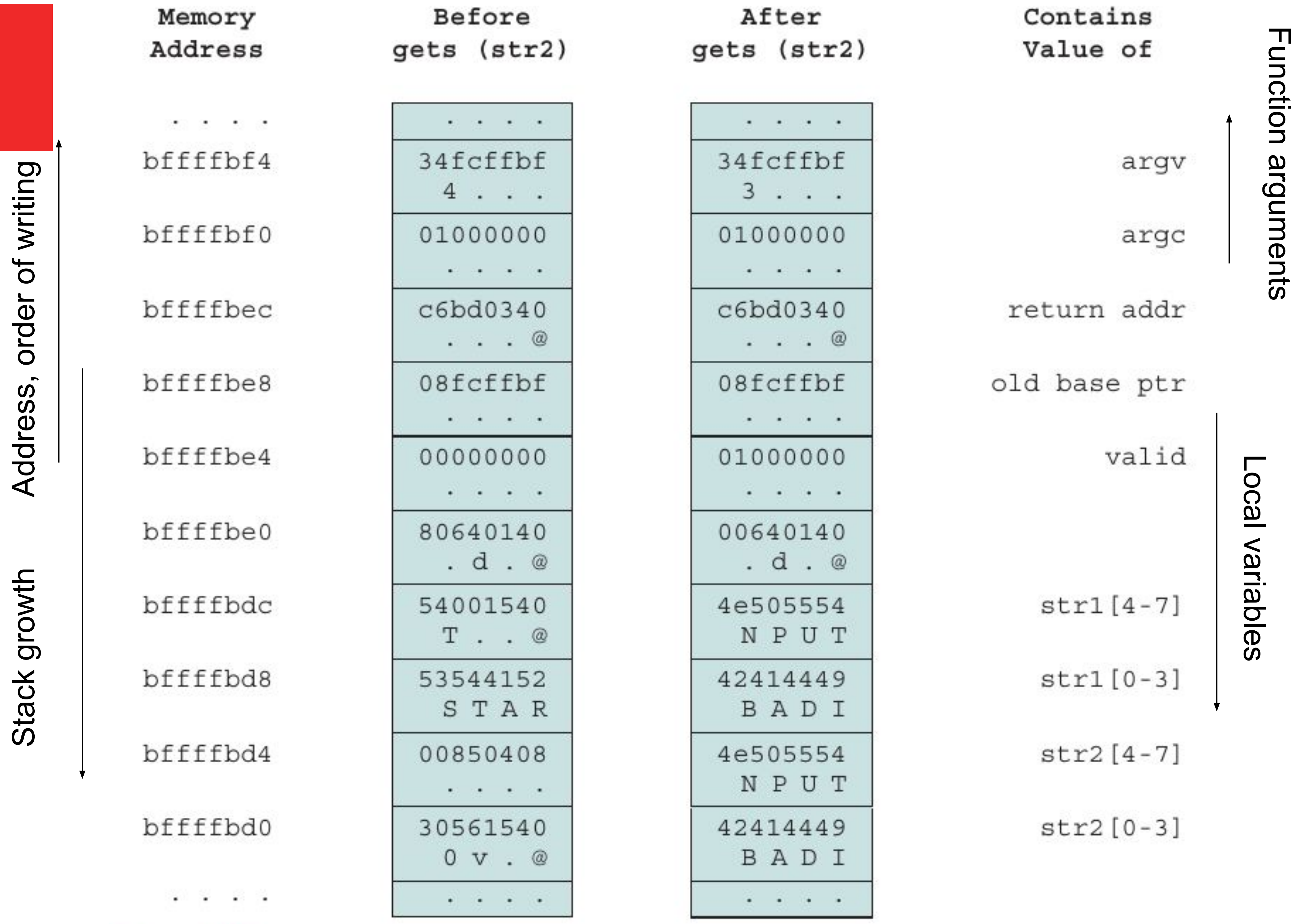
```
int main(int argc, char *argv[]) {
    int valid = FALSE;
    char str1[8];
    char str2[8];

    next_tag(str1);
    gets(str2);
    if (strncmp(str1, str2, 8) == 0)
        valid = TRUE;
    printf("buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);
}
```

(a) Basic buffer overflow C code

```
$ cc -g -o buffer1 buffer1.c
$ ./buffer1
START
buffer1: str1(START), str2(START), valid(1)
$ ./buffer1
EVILINPUTVALUE
buffer1: str1(TVALUE), str2(EVILINPUTVALUE), valid(0)
$ ./buffer1
BADINPUTBADINPUT
buffer1: str1(BADINPUT), str2(BADINPUTBADINPUT), valid(1)
```

(b) Basic buffer overflow example runs

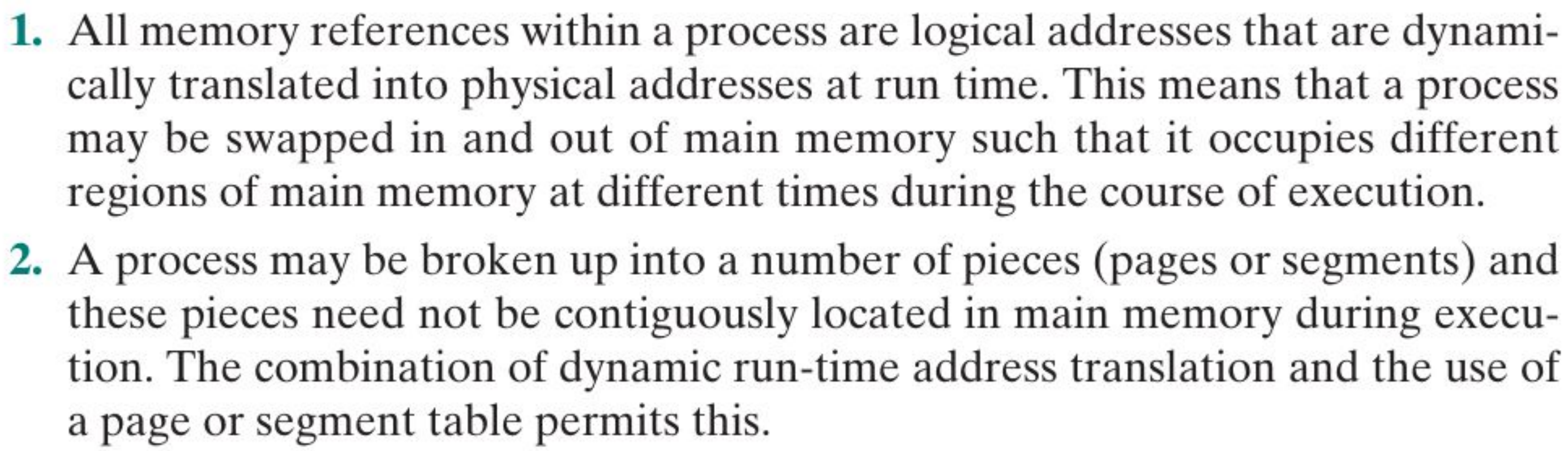
Figure 7.13 Basic Buffer Overflow Example


Address, order of writing
Stack growth
Function arguments
Local variables

Memory Address	Before gets (str2)	After gets (str2)	Contains Value of
. . . .	. . . .	. . . .	
bffffbf4	34fcffbf 4 . . .	34fcffbf 3 . . .	argv
bffffbf0	01000000	01000000	argc
bffffbec	c6bd0340 . . . @	c6bd0340 . . . @	return addr
bffffbe8	08fcffbf	08fcffbf	old base ptr
bffffbe4	00000000	01000000	valid
bffffbe0	80640140 . d . @	00640140 . d . @	
bffffbdc	54001540 T . . @	4e505554 N P U T	str1[4-7]
bffffbd8	53544152 S T A R	42414449 B A D I	str1[0-3]
bffffbd4	00850408	4e505554 N P U T	str2[4-7]
bffffbd0	30561540 0 v . @	42414449 B A D I	str2[0-3]
. . . .	. . . .	. . . .	

Figure 7.14 Basic Buffer Overflow Stack Values

Virtual memory



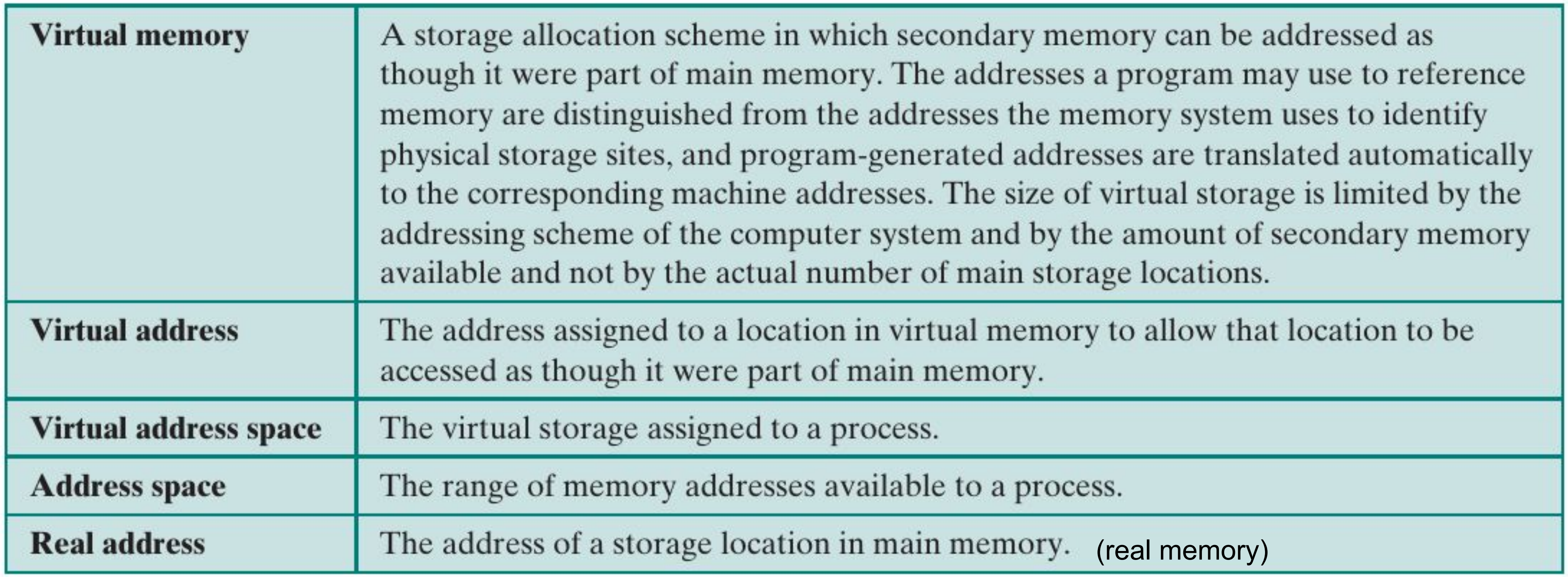
1. All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process may be swapped in and out of main memory such that it occupies different regions of main memory at different times during the course of execution.
2. A process may be broken up into a number of pieces (pages or segments) and these pieces need not be contiguously located in main memory during execution. The combination of dynamic run-time address translation and the use of a page or segment table permits this.

- It is not necessary that all of the pages or all of the segments of a process be in main memory during execution.

Vistual memory terms

- **resident set of the process** - the portion of a process that is actually in main memory at any time

Table 8.1 Virtual Memory Terminology



Virtual memory	A storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of secondary memory available and not by the actual number of main storage locations.
Virtual address	The address assigned to a location in virtual memory to allow that location to be accessed as though it were part of main memory.
Virtual address space	The virtual storage assigned to a process.
Address space	The range of memory addresses available to a process.
Real address	The address of a storage location in main memory. (real memory)

Virtual memory operation

1) If the processor encounters a **logical address that is not in main memory**, it generates an **interrupt indicating a memory access fault**.
2) The OS puts the interrupted process in a **blocking state**.
3) For the execution of this process to proceed later, the OS must **bring into main memory the piece of the process that contains the logical address** that caused the access fault.
4) For this purpose, the OS issues a **disk I/O read request**.
5) After the I/O request has been issued, the OS can dispatch **another process to run** while the disk I/O is performed.
6) Once the desired piece has been brought into main memory, **an I/O interrupt is issued**, giving **control back to the OS**, which places the affected process back into a Ready state.

Virtual memory implications

- More processes may be maintained in main memory.
- A process may be larger than all of main memory

Table 8.2 Characteristics of Paging and Segmentation

Simple Paging	Virtual Memory Paging	Simple Segmentation	Virtual Memory Segmentation
Main memory partitioned into small fixed-size chunks called frames	Main memory partitioned into small fixed-size chunks called frames	Main memory not partitioned	Main memory not partitioned
Program broken into pages by the compiler or memory management system	Program broken into pages by the compiler or memory management system	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)
Internal fragmentation within frames	Internal fragmentation within frames	No internal fragmentation	No internal fragmentation
No external fragmentation	No external fragmentation	External fragmentation	External fragmentation
Operating system must maintain a page table for each process showing which frame each page occupies	Operating system must maintain a page table for each process showing which frame each page occupies	Operating system must maintain a segment table for each process showing the load address and length of each segment	Operating system must maintain a segment table for each process showing the load address and length of each segment
Operating system must maintain a free frame list	Operating system must maintain a free frame list	Operating system must maintain a list of free holes in main memory	Operating system must maintain a list of free holes in main memory
Processor uses page number, offset to calculate absolute address	Processor uses page number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address
All the pages of a process must be in main memory for process to run, unless overlays are used	Not all pages of a process need be in main memory frames for the process to run. Pages may be read in as needed	All the segments of a process must be in main memory for process to run, unless overlays are used	Not all segments of a process need be in main memory for the process to run. Segments may be read in as needed
	Reading a page into main memory may require writing a page out to disk		Reading a segment into main memory may require writing one or more segments out to disk

Locality and Virtual Memory

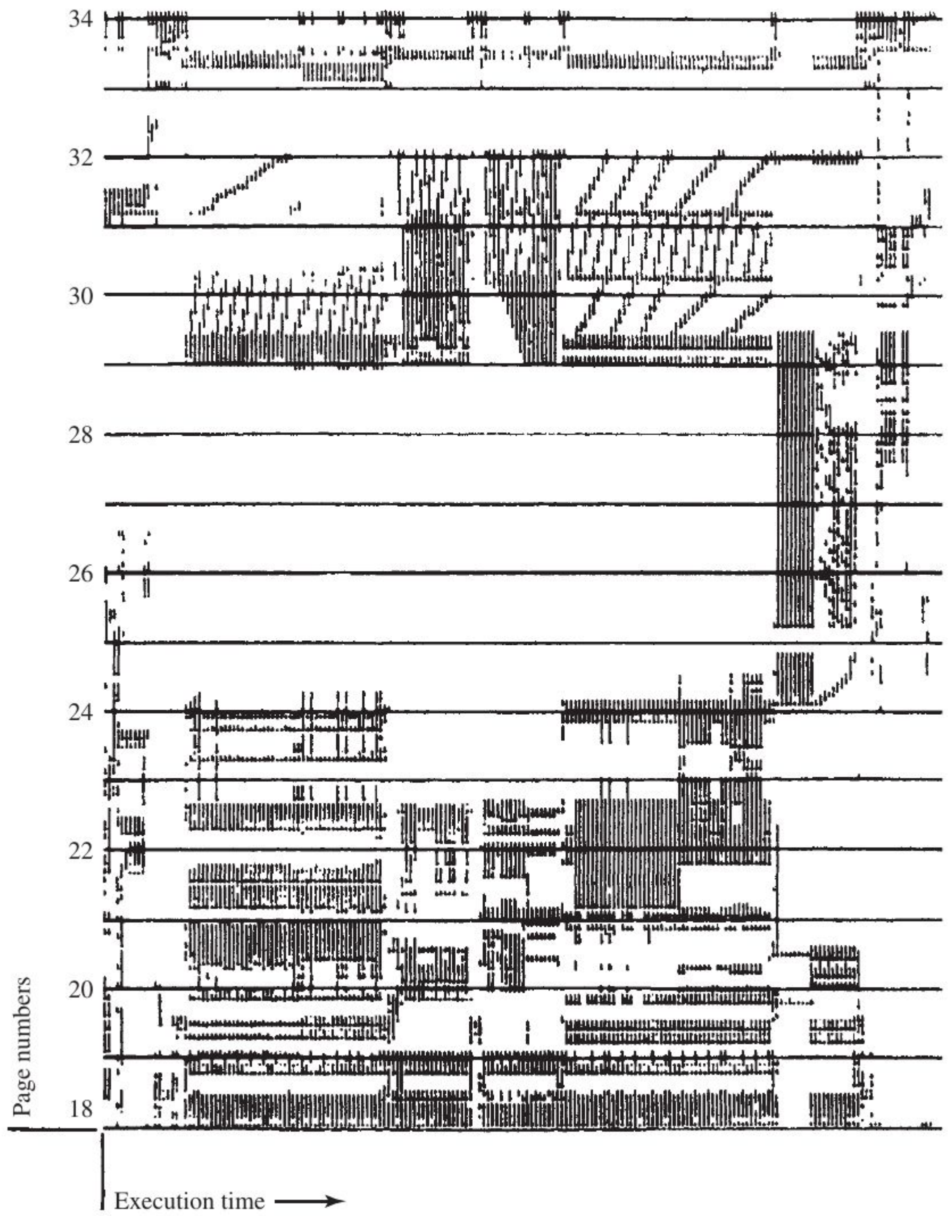


Figure 8.1 **Paging Behavior**

- Need to prevent **thrashing**: The system spends most of its time swapping pieces rather than executing instructions.
- principle of locality

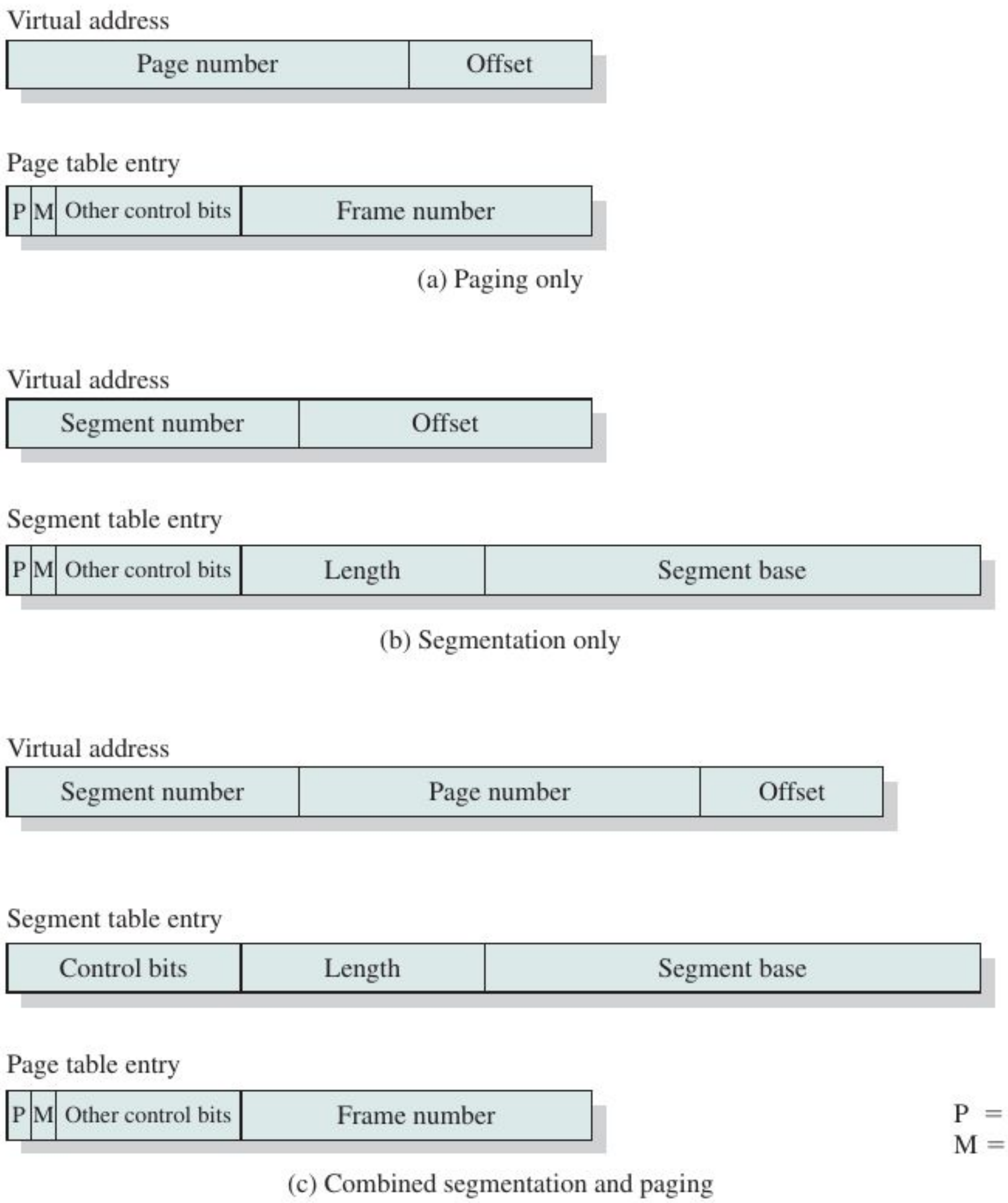


Figure 8.2 **Typical Memory Management Formats**

P - only some of the pages of a process may be in main memory, a bit is needed in each page table entry to **indicate whether the corresponding page is present**

M - modify bit, indicating whether the **contents of the corresponding page have been altered since the page was last loaded** into main memory.
If there has been no change, then it is not necessary to write the page out when it comes time to replace the page in the frame that it currently occupies.

Other control bits may also be present. For example, if protection or sharing is managed at the page level, then bits for that purpose will be required.

Typically, the page number field is longer than the frame number field

P = present bit
M = modified bit

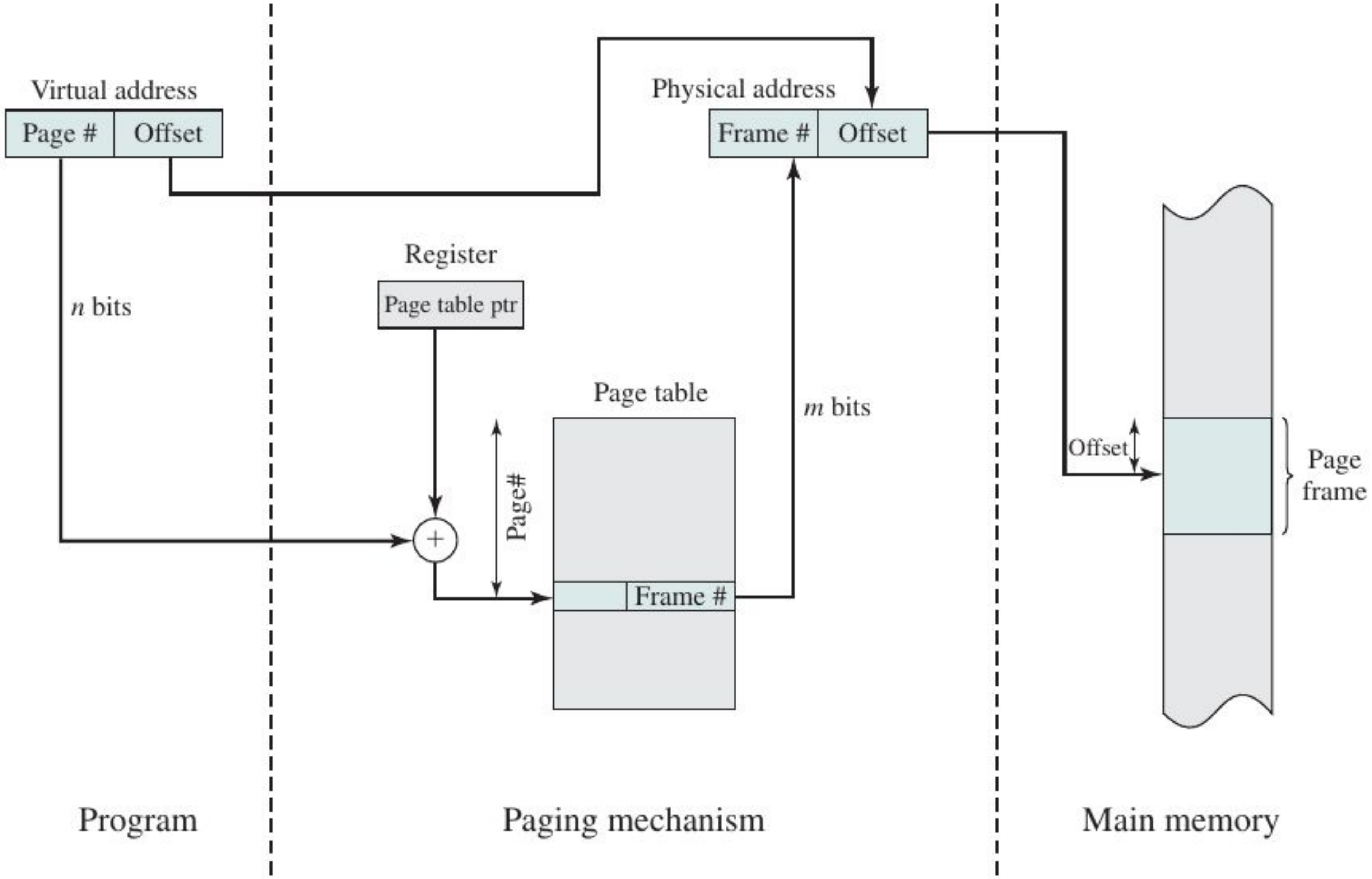


Figure 8.3 **Address Translation in a Paging System**

!!!!

Page table structure

- the page table is of variable length, depending on the size of the process, we cannot expect to hold it in registers.
- Instead, **the page table must be in main memory** to be accessed.
- the amount of memory devoted to page tables alone could be unacceptably high
  - most virtual memory schemes **store page tables in virtual memory** rather than real memory
  - 2^{31}= 2 Gbytes of virtual memory.
    Using 2^9 = 512-byte pages means that as many as 2^{22} page table entries are required per process.
- **page tables are subject to paging** just as other pages
- When a process is running, at least a part of its page table must be in main memory, including the page table entry of the currently executing page
- Some processors make use of a **two-level scheme to organize large page tables**

Two-level page organization scheme

- **page directory**, in which each entry points to a page table.
- process can consist of up to X × Y pages.
- Typically, the maximum length of a page table is restricted to be equal to one page (Pentium)

4-Kbyte (2^{12}) pages
4-Gbyte (2^{32}) virtual address space is composed of 2^{20} pages.

If each of these pages is mapped by a 4-byte page table entry => PTEs requiring 4 Mbytes (2^{22}) – kept in virtual memory (occupying 2^{10} pages)

User page table is mapped by a **root page table** with 2^{10} PTEs occupying 4 Kbytes (2^{12}) of main memory, always remains in main memory

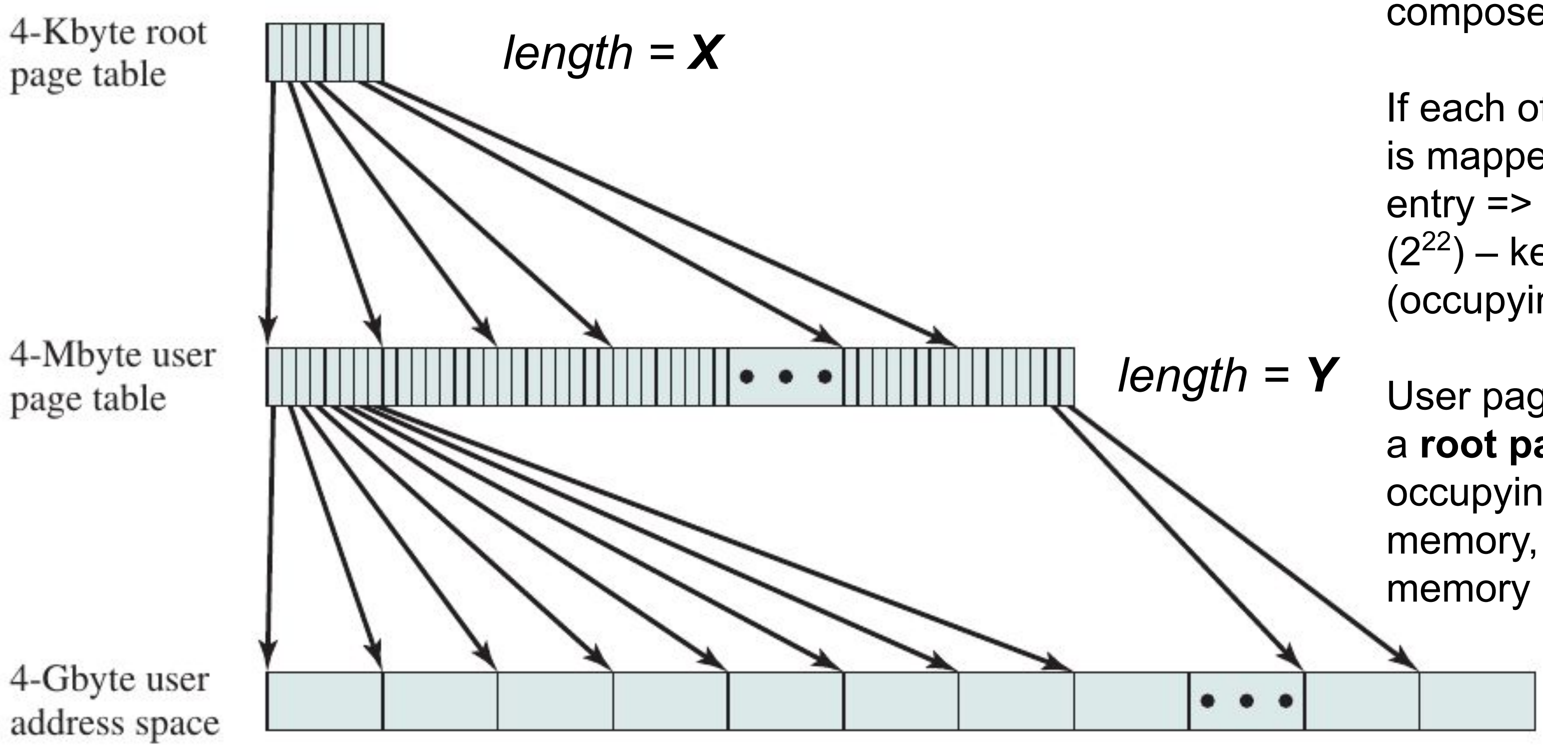


Figure 8.4 A Two-Level Hierarchical Page Table

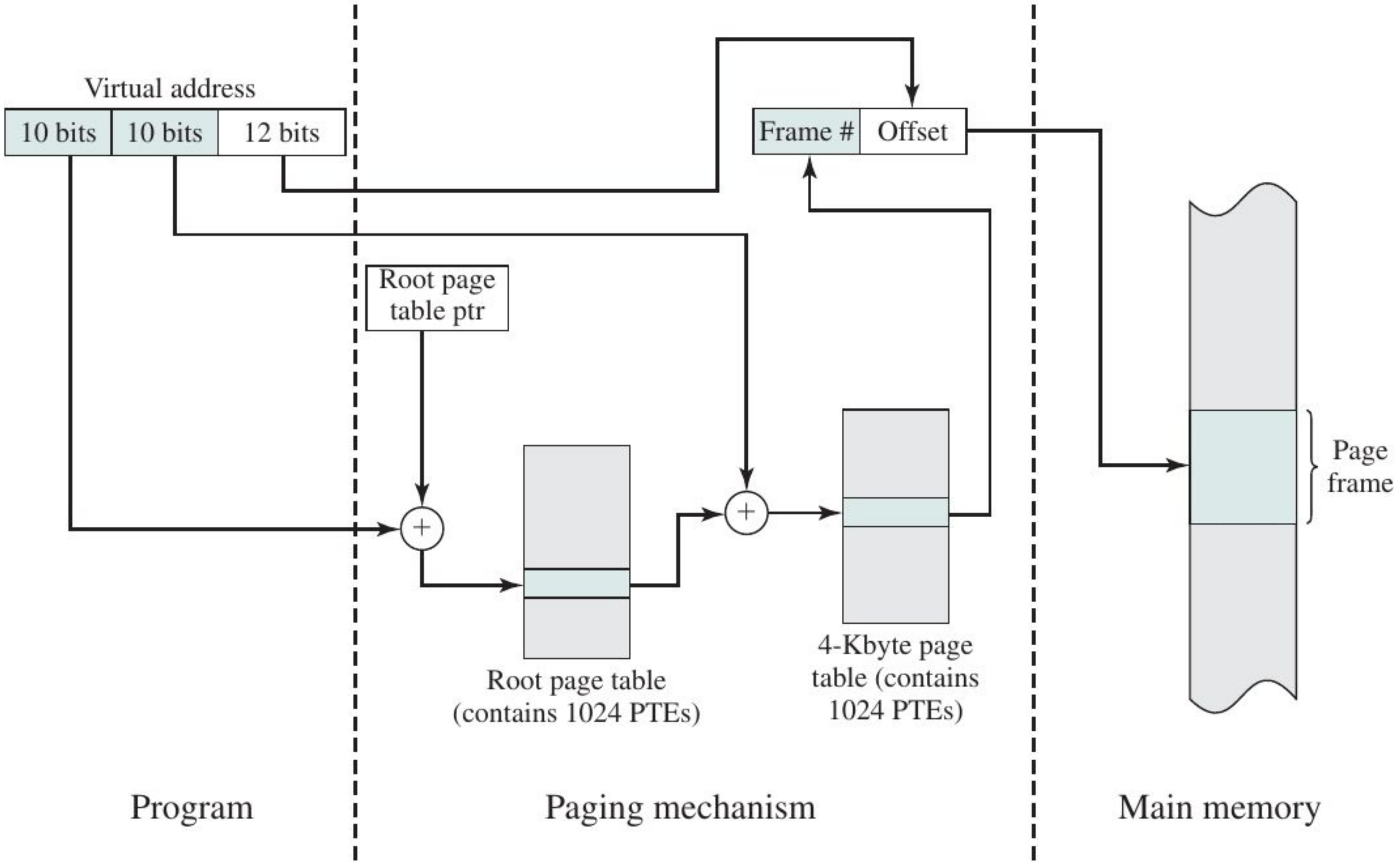


Figure 8.5 **Address Translation in a Two-Level Paging System**

Inverted page table

- A drawback of the type of page tables that we have been discussing is that their size is proportional to that of the virtual address space.
- PowerPC, UltraSPARC, and the IA-64 architecture
- the page number portion of a virtual address is mapped into a hash value using a simple hashing function
- hash value is a pointer to the inverted page table, which contains the page table entries
- fixed proportion of real memory is required for the tables regardless of the number of processes or virtual pages supported

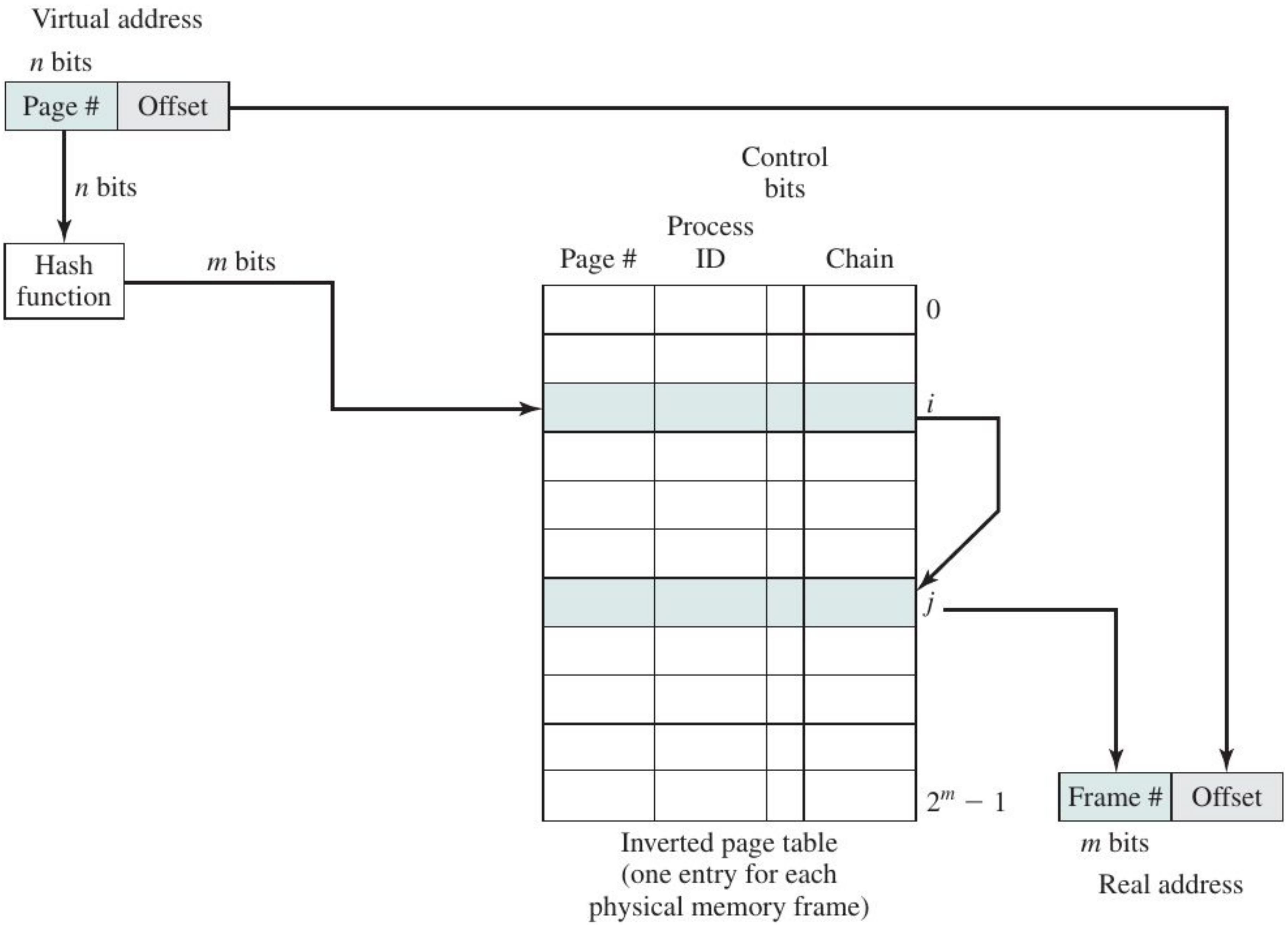


Figure 8.6 **Inverted Page Table Structure**

Translation lookaside buffer

- In principle, every virtual memory reference can cause two physical memory accesses:
  - one to fetch the appropriate page table entry and
  - one to fetch the desired data.
- TLB - special high-speed cache for page table entries
- functions in the same way as a memory cache
- Cannot simply index into the TLB based on page number. Instead, each entry in the TLB must include the page number as well as the complete page table entry.
  - **associative mapping**

- **If the desired page table entry is present** (**TLB hit**), then the frame number is retrieved and the real address is formed.
- **If the desired page table entry is not found** (**TLB miss**), then the processor uses the page number to index the process page table and examine the corresponding page table entry.
  - **If the "present bit" is set**, then the page is in main memory, and the processor can retrieve the frame number from the page table entry to form the real address. The processor also updates the TLB to include this new page table entry.
  - **if the present bit is not se**t, then the desired page is not in main memory and a memory access fault, called a page fault, is issued.
    - At this point, we leave the realm of hardware and invoke the OS, which loads the needed page and updates the page table.

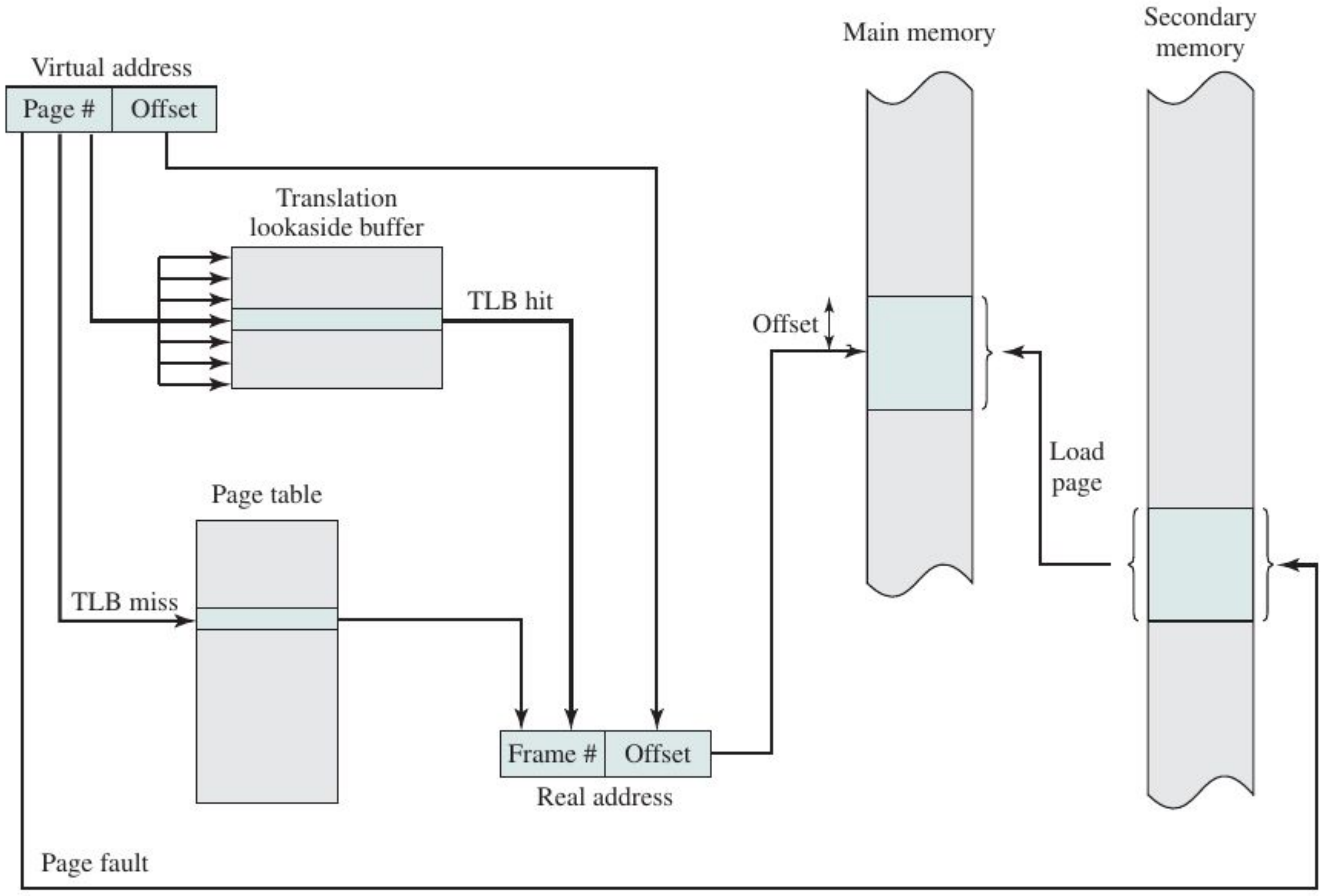


Figure 8.7 Use of a Translation Lookaside Buffer

Translation lookaside buffer - operation

- **If the desired page table entry is present** (**TLB hit**), then the frame number is retrieved and the real address is formed.
- **If the desired page table entry is not found** (**TLB miss**), then the processor uses the page number to index the process page table and examine the corresponding page table entry.
  - **If the “present bit” is set**, then the page is in main memory, and the processor can retrieve the frame number from the page table entry to form the real address. The processor also updates the TLB to include this new page table entry.
  - **if the present bit is not se**t, then the desired page is not in main memory and a memory access fault, called a page fault, is issued.
    - At this point, we leave the realm of hardware and invoke the OS, which loads the needed page and updates the page table.

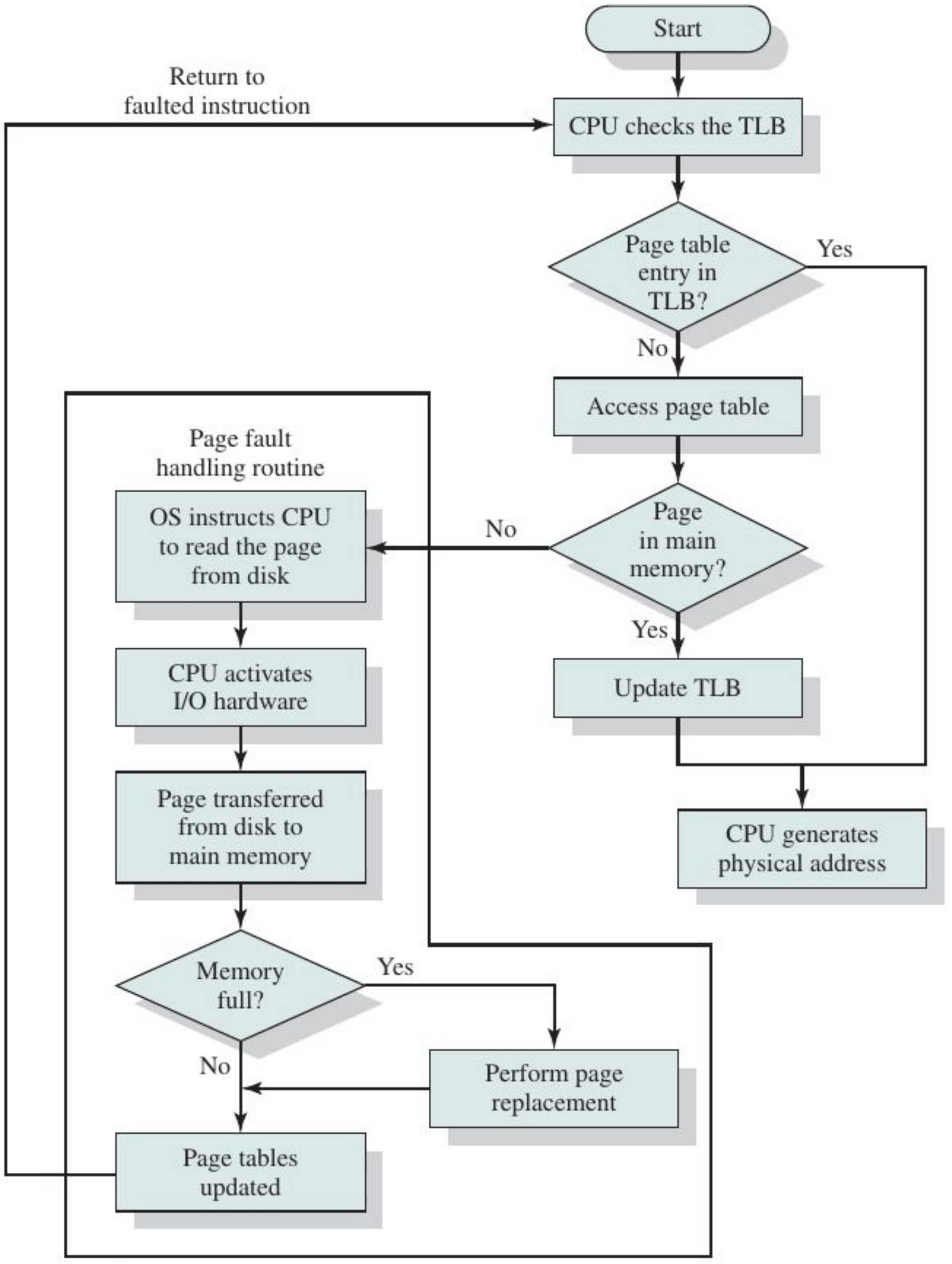

Start
CPU checks the TLB
Return to faulted instruction
Page table entry in TLB?
Yes
No
Access page table
Page in main memory?
No
Yes
Update TLB
CPU generates physical address
Page fault handling routine
OS instructs CPU to read the page from disk
CPU activates I/O hardware
Page transferred from disk to main memory
Memory full?
Yes
No
Perform page replacement
Page tables updated

Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB)

associative mapping **-** Cannot simply index into the TLB based on page number. Instead, each entry in the TLB must include the page number as well as the complete page table entry.

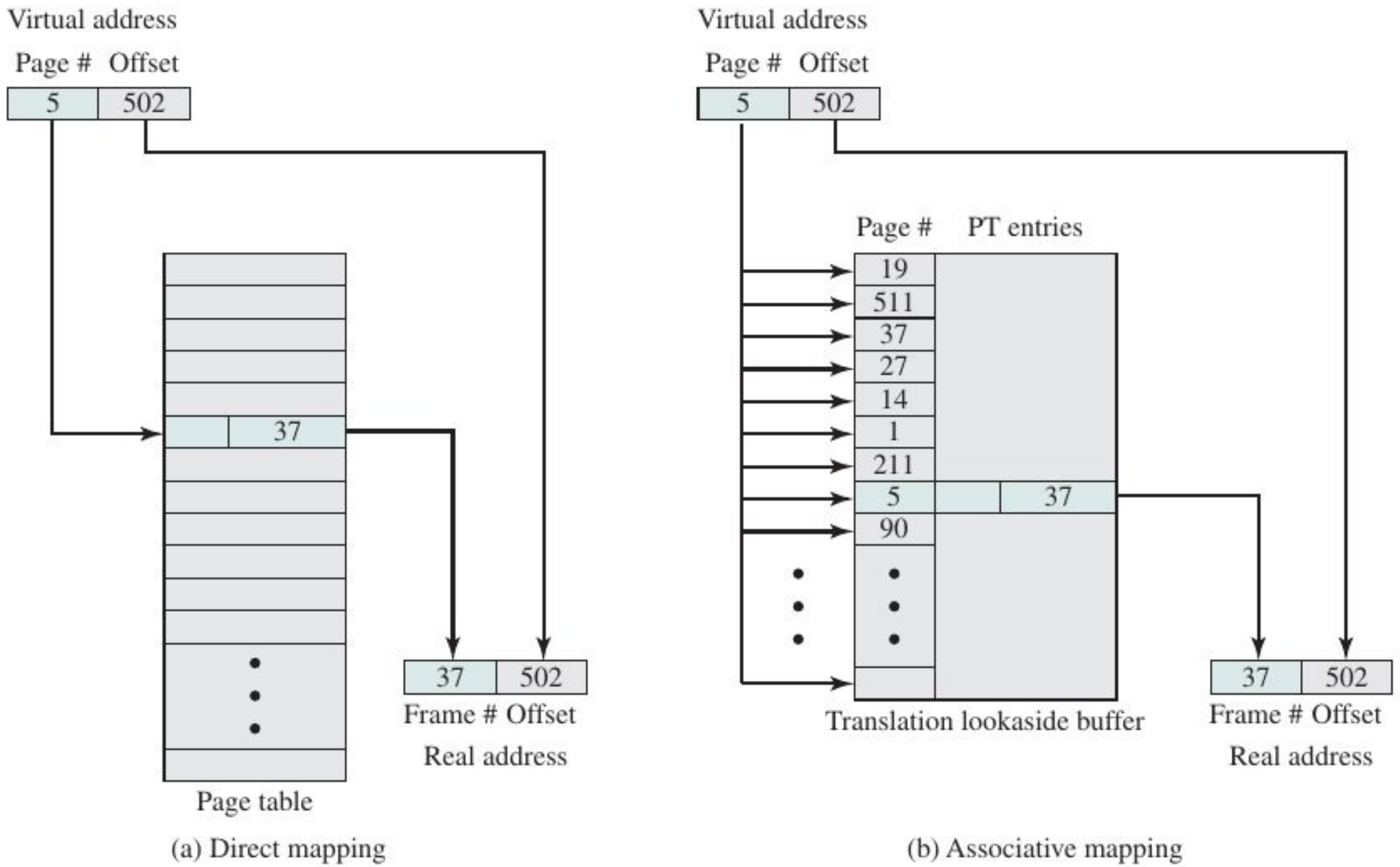


Figure 8.9 **Direct versus Associative Lookup for Page Table Entries**

the virtual memory mechanism must interact with the cache system (not the TLB cache, but the main memory cache).

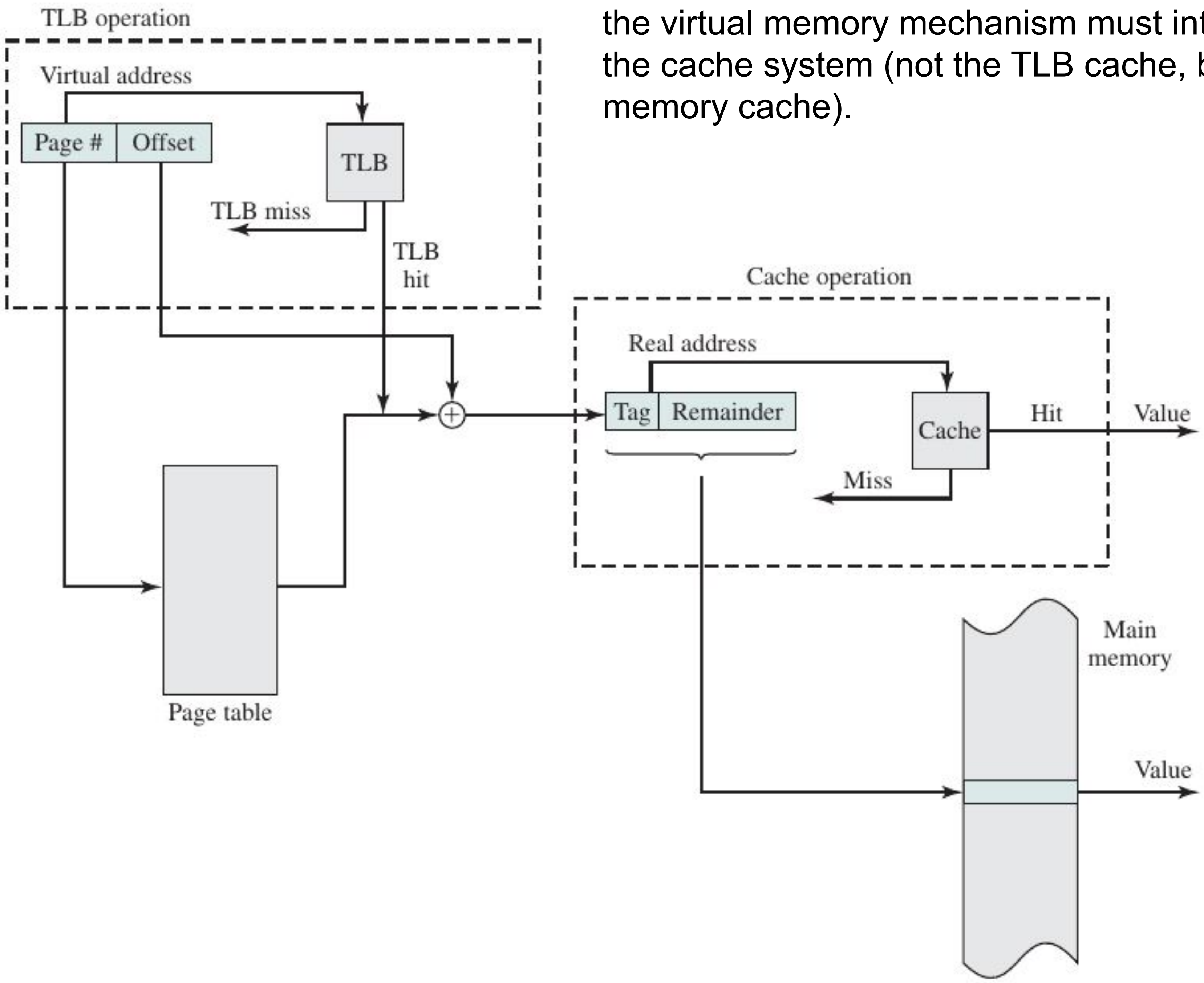
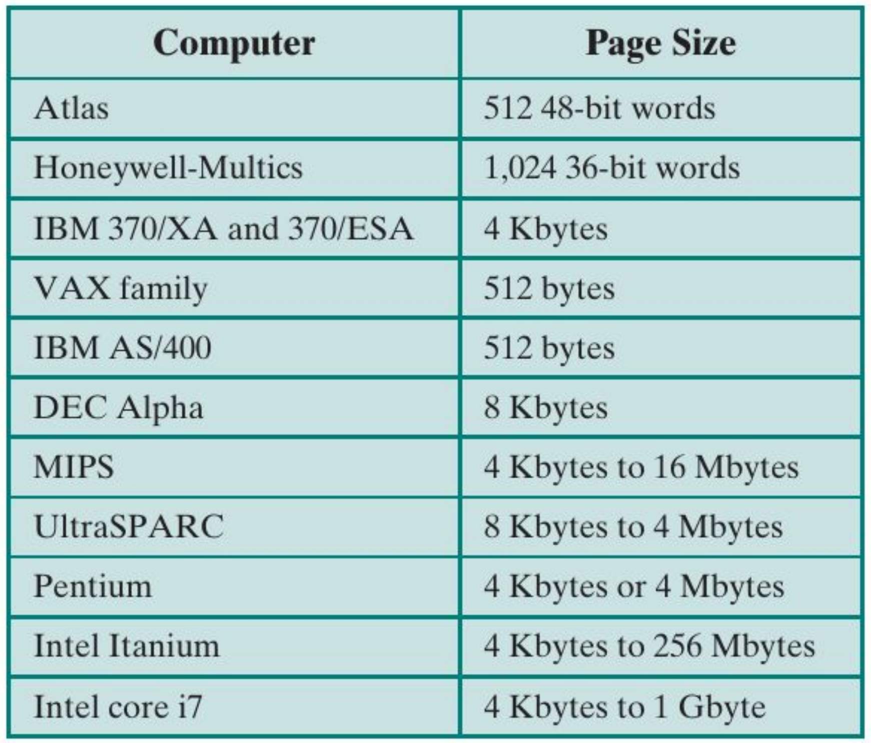


Figure 8.10 Translation Lookaside Buffer and Cache Operation

Page size

- Considering: Internal fragmentation, number of pages,
- effect of size on page faults – impact on locality of reference

Table 8.3 Example Page Sizes



Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1,024 36-bit words
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 Kbytes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
Intel Itanium	4 Kbytes to 256 Mbytes
Intel core i7	4 Kbytes to 1 Gbyte

Segmentation

- allows the programmer to view memory as consisting of multiple address spaces or segments

1) It simplifies the handling of growing data structures. Changing segment size based on size of a data structure.
2) It allows programs to be altered and recompiled independently, without requiring the entire set of programs to be relinked and reloaded.
3) It lends itself to sharing among processes. A programmer can place a utility program or a useful table of data in a segment that can be referenced by other processes.
4) It lends itself to protection - privileges

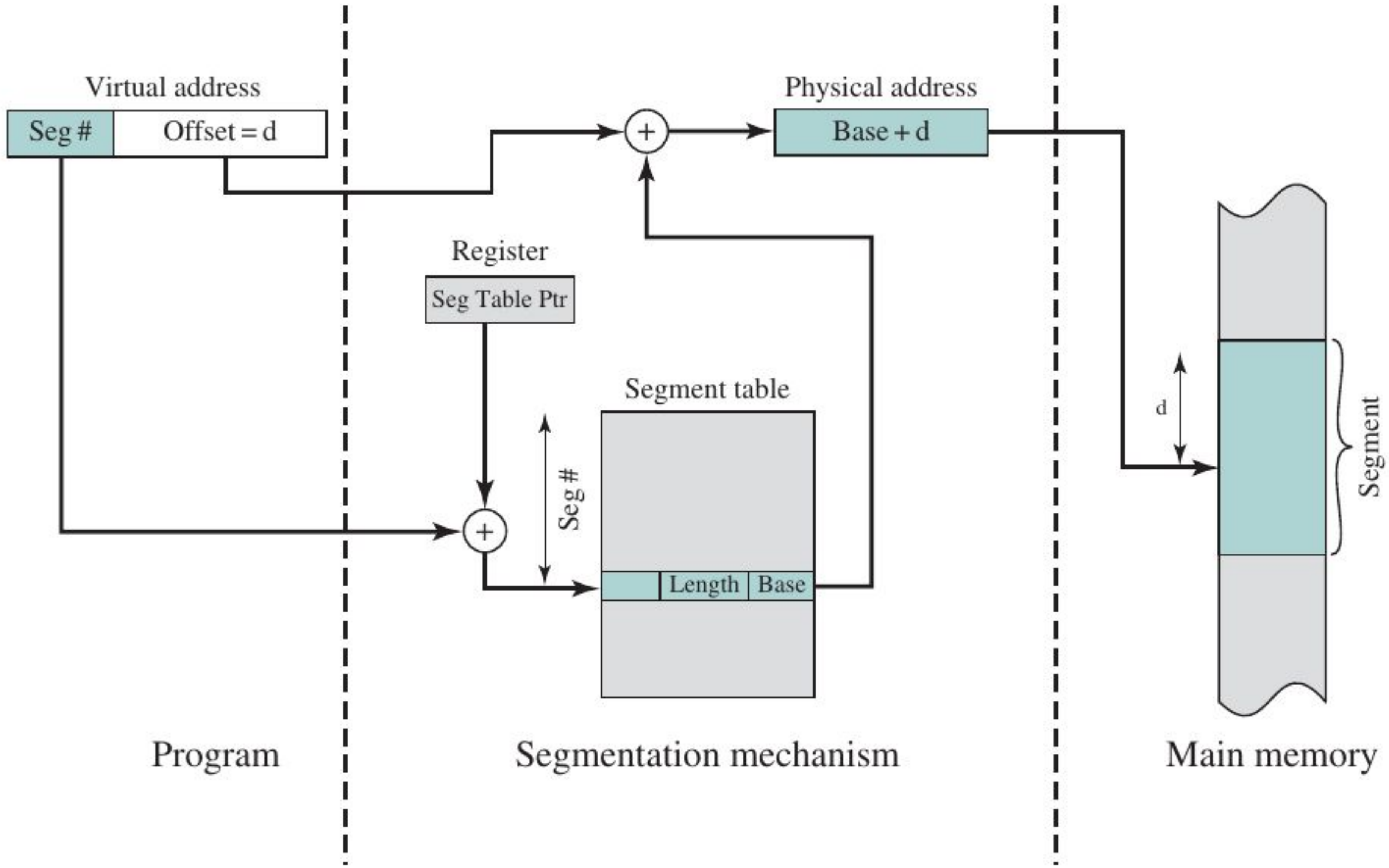


Figure 8.12 **Address Translation in a Segmentation System**

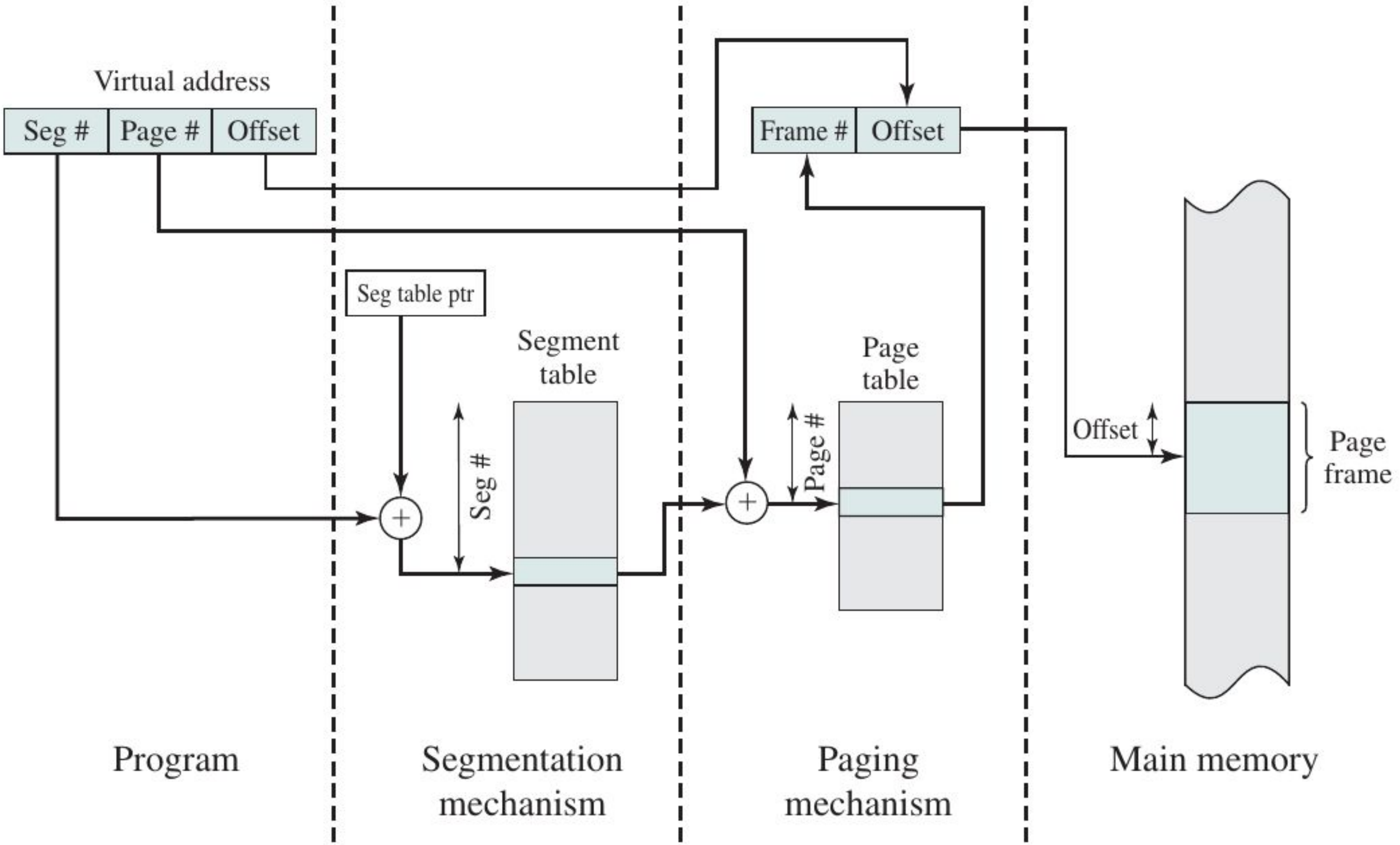


Figure 8.13 **Address Translation in a Segmentation/Paging System**

Segmentation lends itself to the implementation of protection and sharing policies. Because each segment table entry includes a length as well as a base address, a program cannot inadvertently access a main memory location beyond the limits of a segment. To achieve sharing, it is possible for a segment to be referenced in the segment tables of more than one process. The same mechanisms are, of course, available in a paging system. However, in this case the page structure of programs and data is not visible to the programmer, making the specification of protection and sharing requirements more awkward.

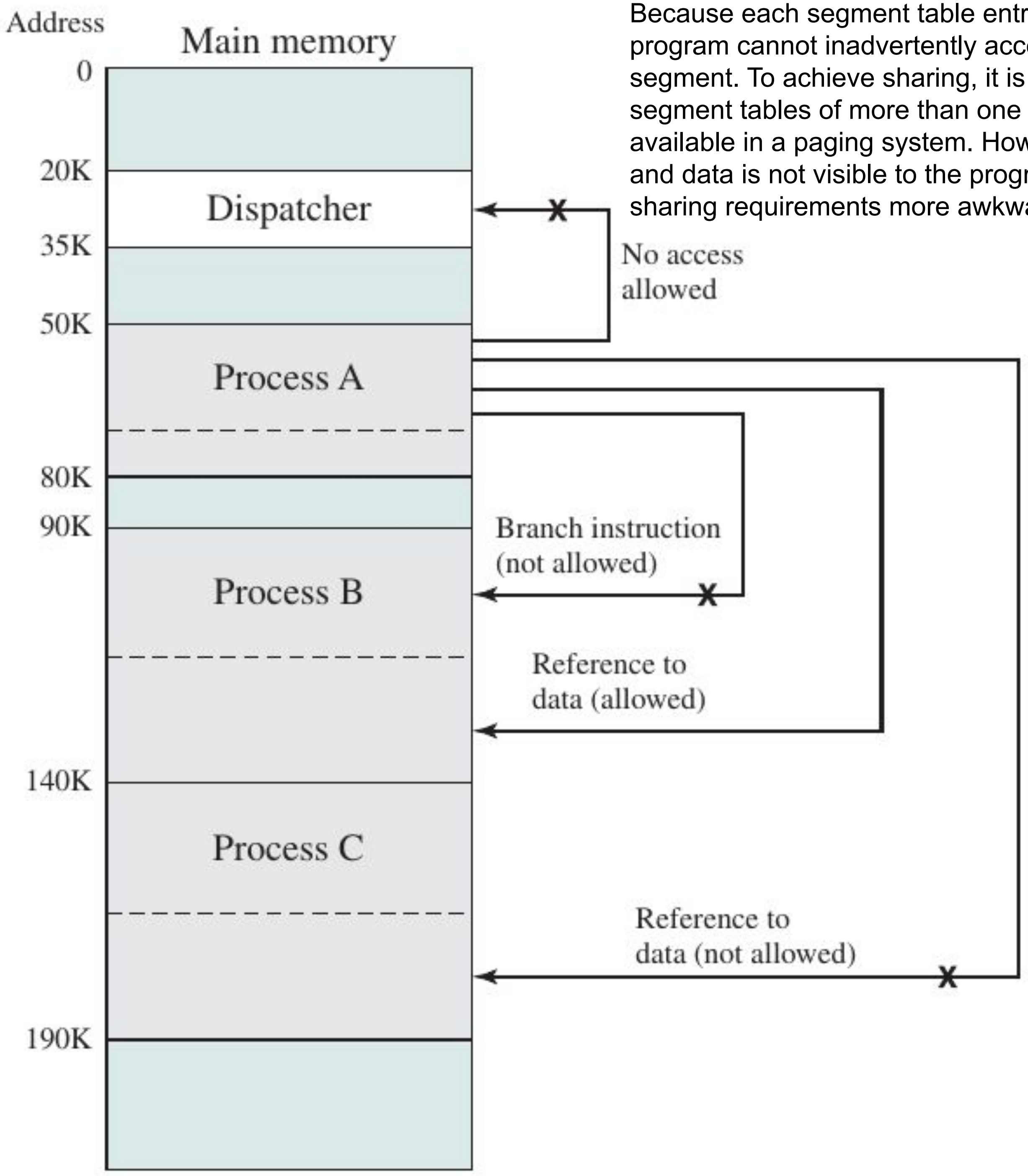


Figure 8.14 **Protection Relationships between Segments**

Table 8.4 Operating System Policies for Virtual Memory

Fetch Policy
- Demand paging
- Prepaging

Placement Policy

Replacement Policy
- Basic Algorithms
  - Optimal
  - Least recently used (LRU)
  - First-in-first-out (FIFO)
  - Clock
- Page Buffering

Resident Set Management
- Resident set size
  - Fixed
  - Variable
- Replacement Scope
  - Global
  - Local

Cleaning Policy
- Demand
- Precleaning

Load Control
- Degree of multiprogramming

Fetch policy and Cleaning policy

- **Fetch policy**
  - **determines when a page should be brought into main memory**
  - Alternatives
    - **Demand paging** - only when a reference is made to a location on that page
    - **Prepaging** - pages other than the one demanded by a page fault are brought in.
- **Cleaning policy**
  - Determining when a modified page should be written out to secondary memory
  - Alternatives
    - **demand cleaning** - only when a page has been selected for replacement
    - **Precleaning** - page is written out but remains in main memory until the page replacement, pages can be written out in batches

Placement policy

- **determines where in real memory a process piece is to reside**
- pure segmentation system - best-fit, first-fit, and so on
- pure paging or paging combined with segmentation - irrelevant because the address translation hardware and the main memory access hardware can perform their functions for any page-frame combination with equal efficiency
- Concern on nonuniform memory access (NUMA) multiprocessor

Replacement policy

- deals with the selection of a page in main memory to be replaced when a new page must be brought in
- Concepts:
  - How many page frames are to be allocated to each active process
  - Whether the set of pages to be considered for replacement should be limited to those of the process that caused the page fault or encompass all the page frames in main memory
  - Among the set of pages considered, which particular page should be selected for replacement

Replacement policy

- Frame locking
  - When a frame is locked, the page currently stored in that frame may not be replaced.
  - Kernel, key control structures
  - Sticky bit?
- Basic algorithms
  - Optimal (impossible),
  - Least recently used (LRU),
  - First-in-first-out (FIFO)
  - Clock $\longrightarrow$

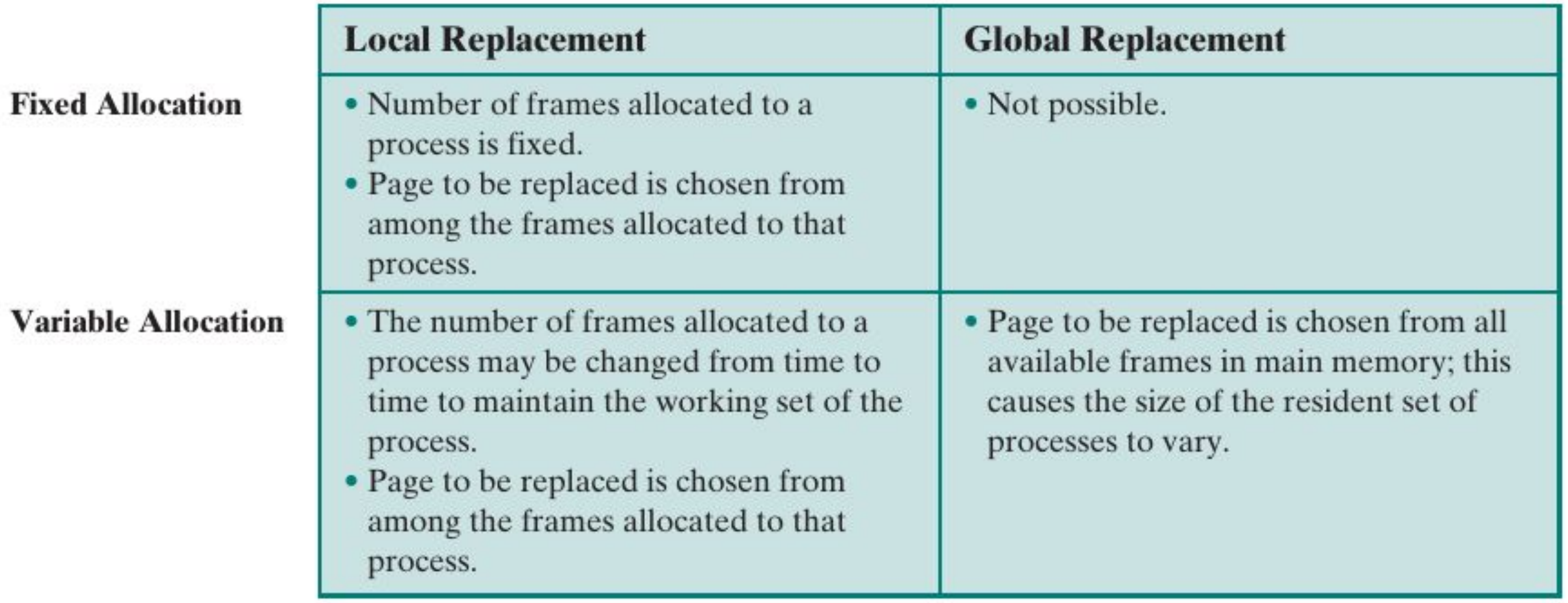
8.2 / OPERATING SYSTEM SOFTWARE 365

- Page buffering – replaced pages remain in memory “longer”, less I/O

Resident Set Management

- Resident set size - not necessary and indeed may not be possible to bring all of the pages of a process into main memory to prepare it for execution
  - Fixed-allocation, variable-allocation
- Replacement scope

Table 8.5 Resident Set Management



	Local Replacement	Global Replacement
Fixed Allocation	• Number of frames allocated to a process is fixed. • Page to be replaced is chosen from among the frames allocated to that process.	• Not possible.
Variable Allocation	• The number of frames allocated to a process may be changed from time to time to maintain the working set of the process. • Page to be replaced is chosen from among the frames allocated to that process.	• Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.

Table 8.6 UNIX SVR4 Memory Management Parameters

Page Table Entry

Page frame number
Refers to frame in real memory.

Age
Indicates how long the page has been in memory without being referenced. The length and contents of this field are processor dependent.

Copy on write
Set when more than one process shares a page. If one of the processes writes into the page, a separate copy of the page must first be made for all other processes that share the page. This feature allows the copy operation to be deferred until necessary and avoided in cases where it turns out not to be necessary.

Modify
Indicates page has been modified.

Reference
Indicates page has been referenced. This bit is set to 0 when the page is first loaded and may be periodically reset by the page replacement algorithm.

Valid
Indicates page is in main memory.

Protect
Indicates whether write operation is allowed.

Disk Block Descriptor

Swap device number
Logical device number of the secondary device that holds the corresponding page. This allows more than one device to be used for swapping.

Device block number
Block location of page on swap device.

Type of storage
Storage may be swap unit or executable file. In the latter case, there is an indication as to whether the virtual memory to be allocated should be cleared first.

Page Frame Data Table Entry

Page state
Indicates whether this frame is available or has an associated page. In the latter case, the status of the page is specified: on swap device, in executable file, or DMA in progress.

Reference count
Number of processes that reference the page.

Logical device
Logical device that contains a copy of the page.

Block number
Block location of the page copy on the logical device.

Pfdata pointer
Pointer to other pfdata table entries on a list of free pages and on a hash queue of pages.

Swap-Use Table Entry

Reference count
Number of page table entries that point to a page on the swap device.

Page/storage unit number
Page identifier on storage unit.