Operating Systems

Chapter 8 Virtual Memory(虚拟内存)

Agenda

- 8.1 Hardware and Control Structure
- 8.2 Operating System Software
- 8.3 Summary

8.1 Hardware and Control Structure

- 8.1.0 Overview
- 8.1.1 Locality and Virtual Memory
- 8.1.2 Paging
- 8.1.3 Segmentation
- 8.1.4 Combined Paging and Segmentation
- 8.1.5 Protection and Sharing

Hardware and Control Structures

- Memory references are dynamically translated into physical addresses at run time(运行时访问地址动态转换)
  - A process may be swapped in and out (换入和换出) of main memory such that it occupies different regions
- A process may be broken up into pieces that do not need to located contiguously in main memory(进程分块)
  - All pieces of a process do not need to be loaded in main memory during execution(部分载入)

Execution of a Program(程序运行)

- Operating system brings into main memory a few pieces of the program
- Resident set(常驻集) - portion of process that is in main memory
- When an address is needed that is not in main memory:
  1. An interrupt is generated, which is known as page fault interrupt(缺页中断)
  2. Operating system places the process in a blocking state

Execution of a Program

3. Piece of process that contains the logical address is brought into main memory
   - Operating system issues a disk I/O Read request
   - Another process is dispatched to run while the disk I/O takes place
   - An interrupt is issued when disk I/O complete
4. operating system place the affected process in the Ready state

Advantages of Breaking up a Process(进程分块的优点)

- More processes may be maintained in main memory(主存中保留多个进程)
  - Only load in some of the pieces of each process
  - With so many processes in main memory, it is very likely a process will be in the Ready state at any particular time
- A process may be larger than all of main memory(支持大于主存的进程)

Types of Memory(存储器类型)

- Real memory(实存)
  - Main memory, the memory provided by computer hardware
- Virtual memory(虚存)
  - The memory to programmer's view
  - Allows for effective multiprogramming
  - Relieves the user of tight constraints of main memory (解除了用户与主存之间的紧密约束)

Thrashing(系统抖动/系统颠簸)

- Thrashing
  - Swapping out a piece of a process just before that piece is needed
  - Too much of this operations may leads the processor spends most of its time swapping pieces rather than executing user instructions

8.1 Hardware and Control Structure

- 8.1.0 Overview
- 8.1.1 Locality局部性 and Virtual Memory
- 8.1.2 Paging
- 8.1.3 Segmentation
- 8.1.4 Combined Paging and Segmentation
- 8.1.5 Protection and Sharing

Principle of Locality(局部性原理)

- Program and data references within a process tend to cluster(程序和数据访问的集簇倾向)
- Only a few pieces of a process will be needed over a short period of time
- Possible to make intelligent guesses about which pieces will be needed in the future
- This suggests that virtual memory may work efficiently

Support Needed for Virtual Memory

- Hardware：
  - support paging and segmentation (硬件要支持分页和分段)
- Software：
  - Operating system must be able to management the movement of pages and/or segments between secondary memory and main memory (操作系统要实现内存管理，使得段/页可以在内存与辅存之间移动)

8.1 Hardware and Control Structure

- 8.1.0 Overview
- 8.1.1 Locality and Virtual Memory
- 8.1.2 Paging
- 8.1.3 Segmentation
- 8.1.4 Combined Paging and Segmentation
- 8.1.5 Protection and Sharing

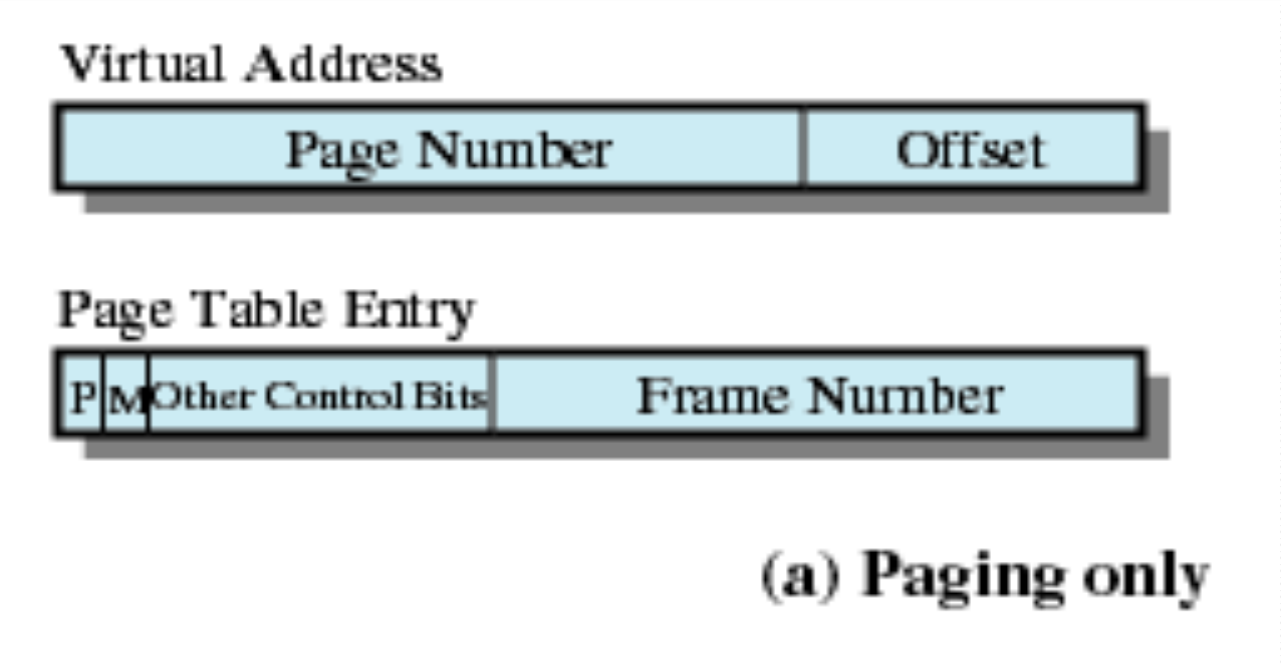
Paging(分页)

- Each process has its own page table(页表)
- Each page table entry(页表项) contains the frame number of the corresponding page in main memory
- Present Bit(存在位):
  - A bit is needed to indicate whether the page is in main memory or not
  - If not in memory, a page fault interrupt will be generated

Paging(分页)

- Modified Bit(修改位):
  - A bit may be needed to indicate whether the page has been modified or not since it was loaded in main memory
  - If no change has been made, the page does not have to be written to the disk when it needs to be swapped out(换出)

Page Table Structure



Address Translation in a Paging System(地址转换系统)

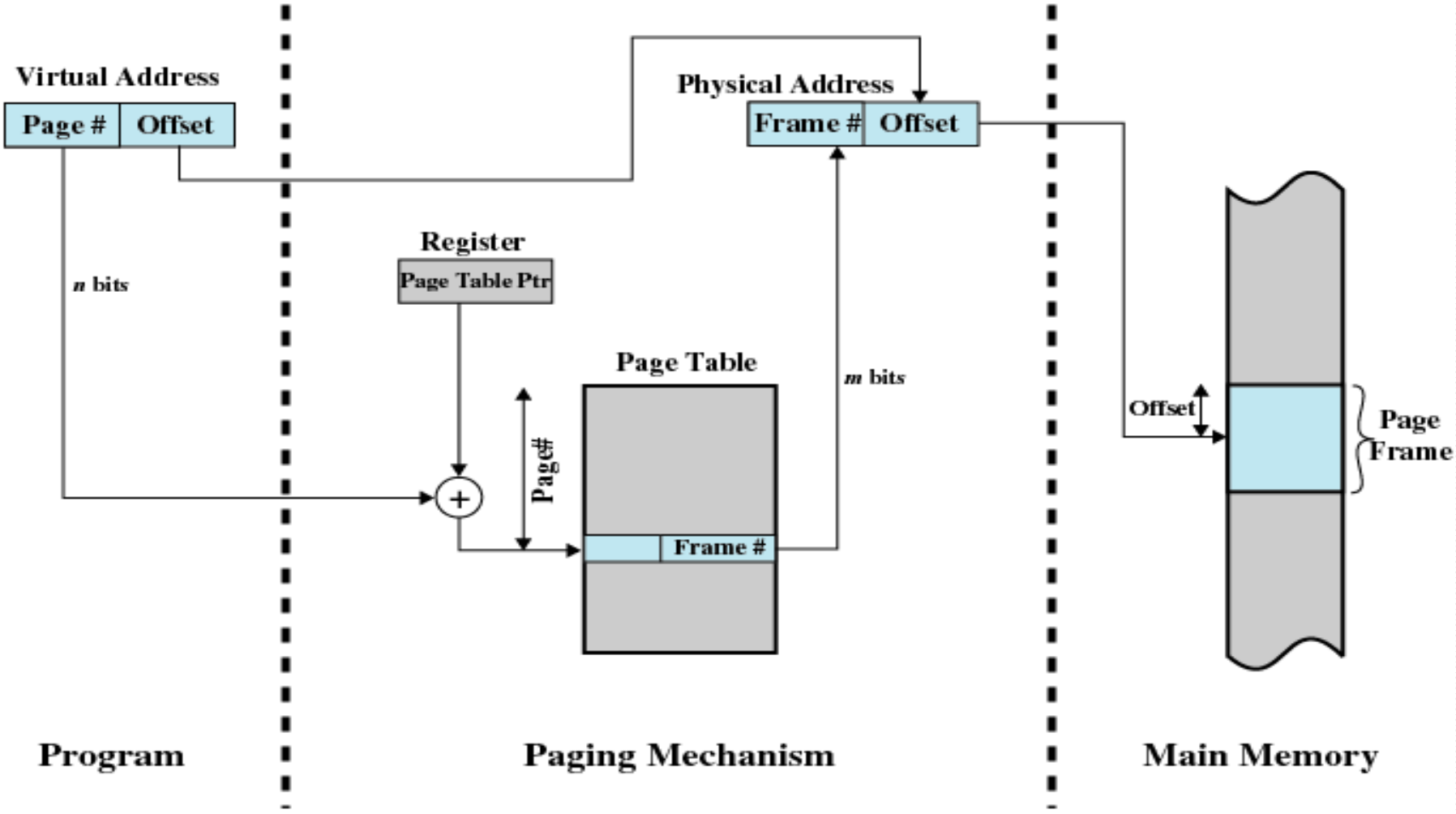


Figure 8.3 Address Translation in a Paging System

Page Tables(页表)

- The entire page table size is proportional to that of the virtual address space, and may take up too much main memory(页表大小与虚拟地址空间成比例增加，可能会很大)
- Page tables are also stored in virtual memory(页表自身在虚存)
- When a process is running, part of its page table is in main memory(部分页表在主存)

Two-Level Scheme for 32-bit Address(32位地址的两级页表方案)

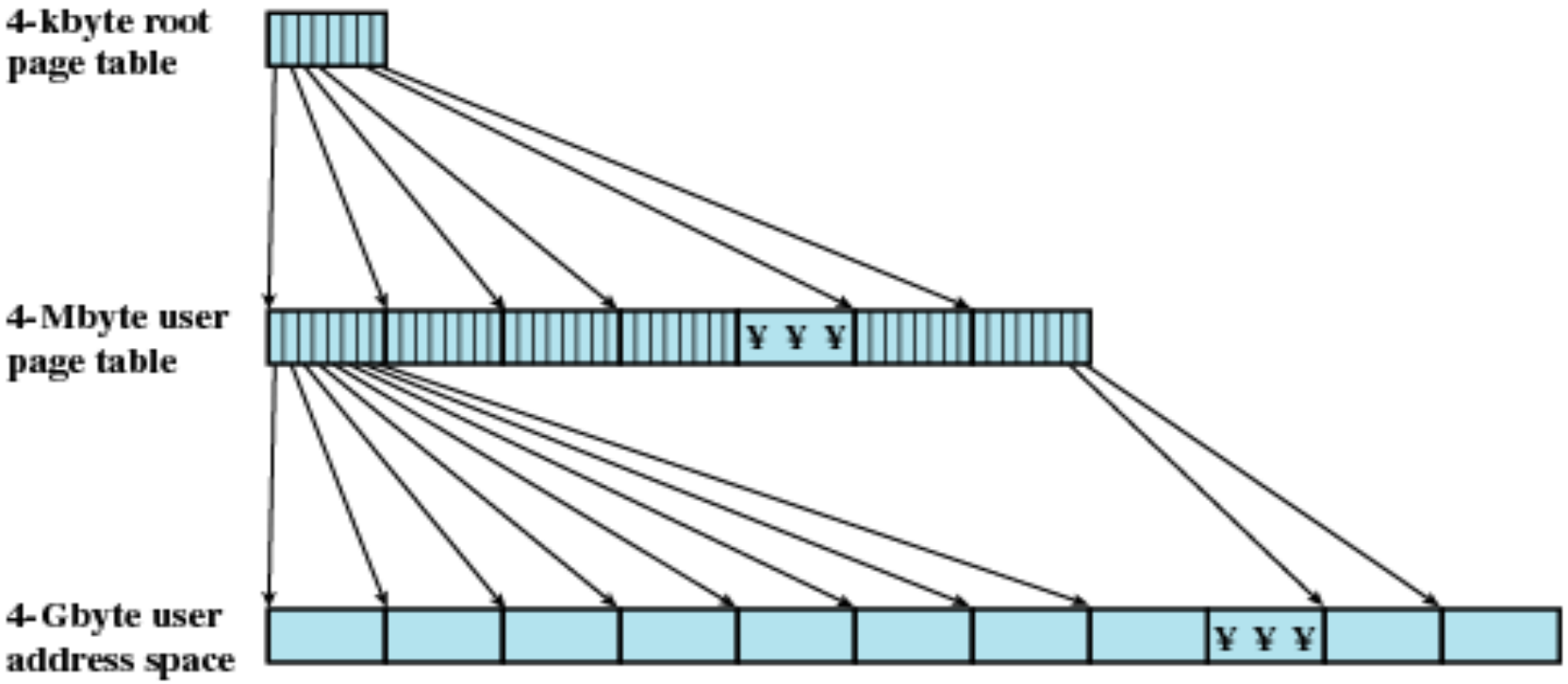


Figure 8.4 A Two-Level Hierarchical Page Table

Two-Level Scheme for 32-bit Address(32位地址的两级页表方案)

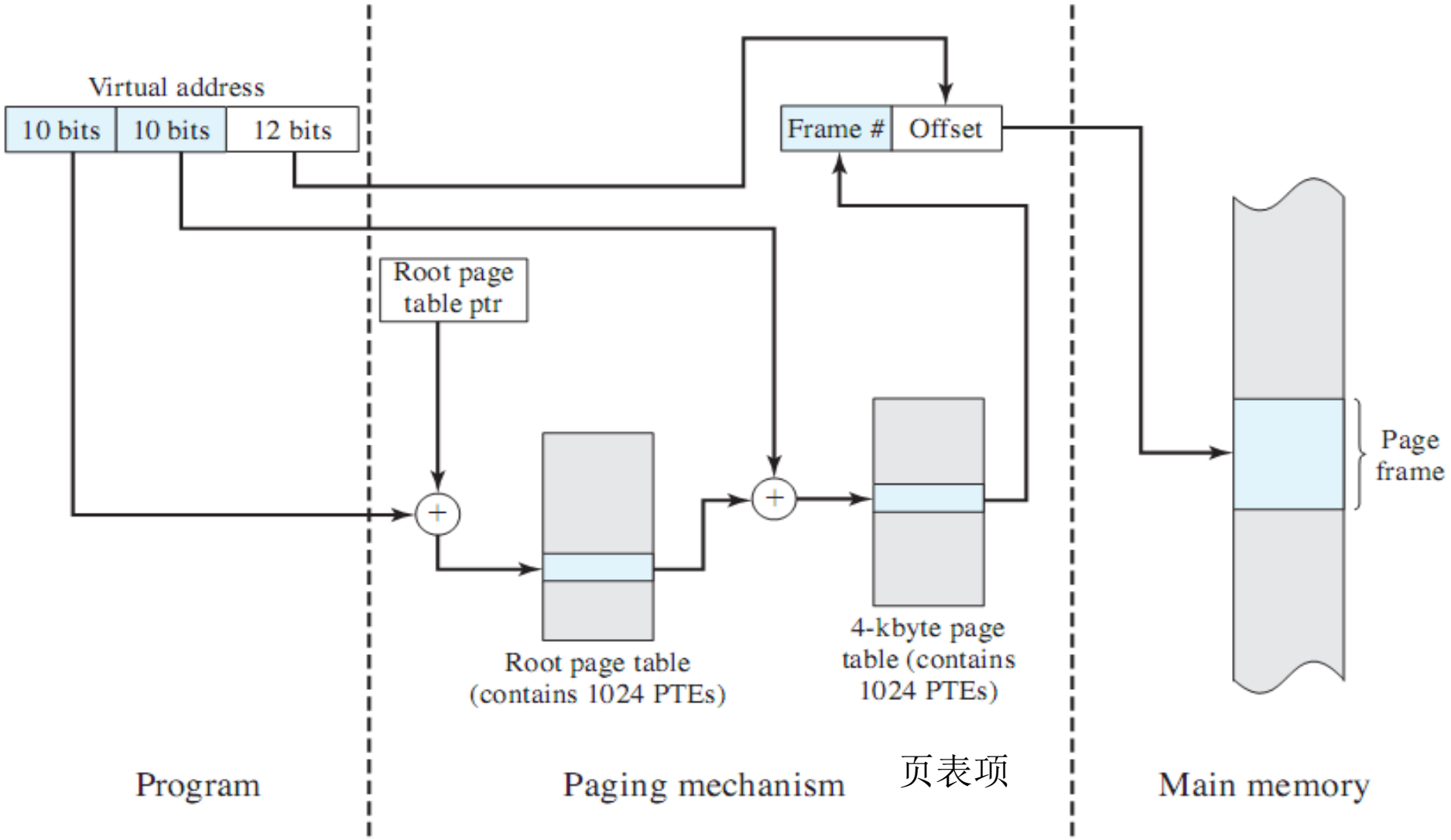


Figure 8.5 Address Translation in a Two-Level Paging System

Inverted Page Table(反向页表)

- Used on PowerPC, UltraSPARC, and IA-64 architecture
- Page number portion of a virtual address is mapped into a hash value
- Hash value points to inverted page table
- Fixed proportion of real memory is required for the tables regardless of the number of processes or virtual pages supported

Inverted Page Table

- Page table entry
  - Page number(页号)
  - Process identifier(进程标志符)
  - Control bits(控制位)
  - Chain pointer(链指针)

Inverted Page Table Structure

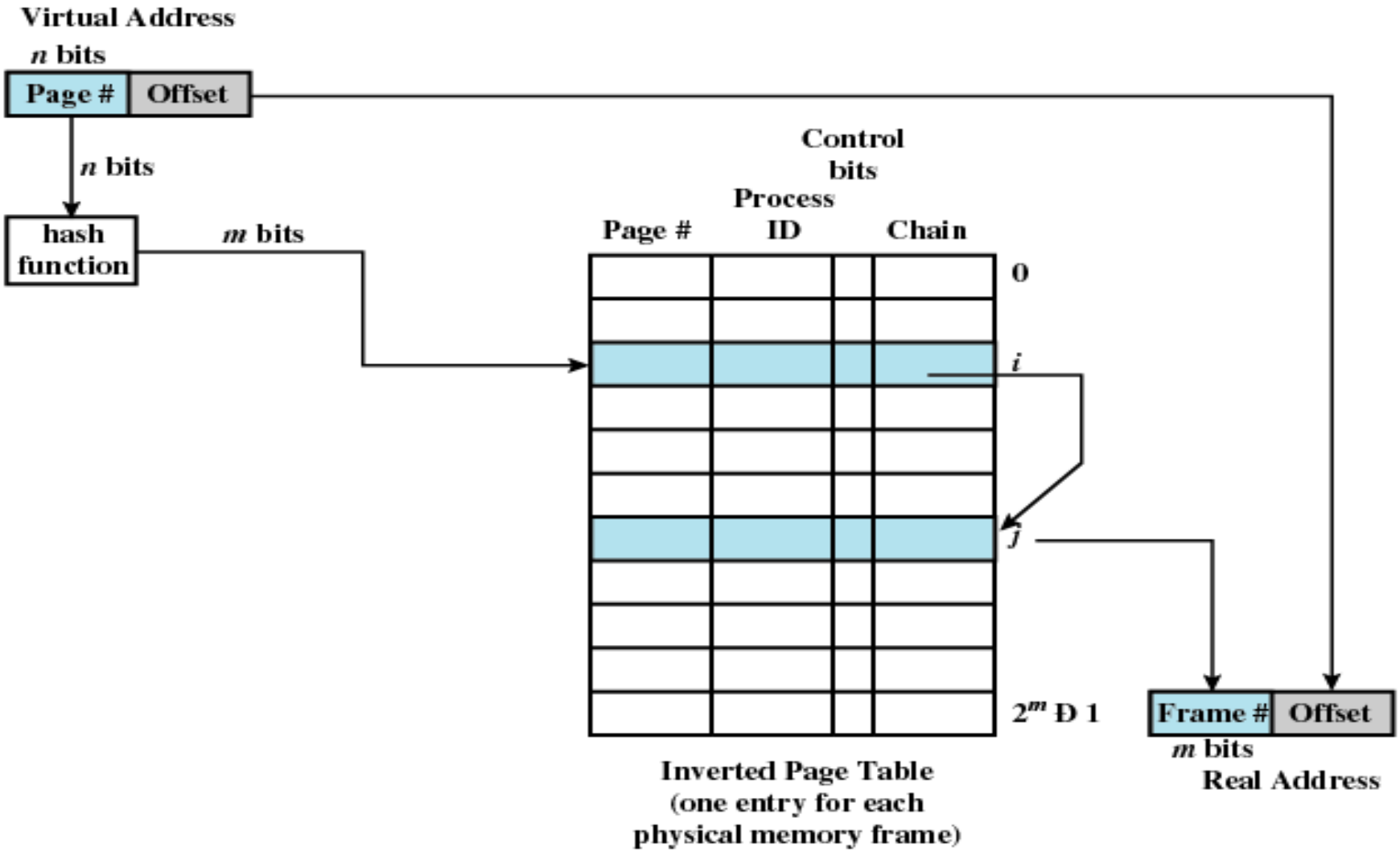


Figure 8.6 Inverted Page Table Structure

Translation Lookaside Buffer(转移后备缓冲器/旁视缓冲器)

- Each virtual memory reference can cause two physical memory accesses
  - One to fetch the page table
  - One to fetch the data
- To overcome this problem a high-speed cache is set up for page table entries
  - Called a Translation Lookaside Buffer (TLB)
  - Contains page table entries that have been most recently used

Use of a Translation Lookaside Buffer

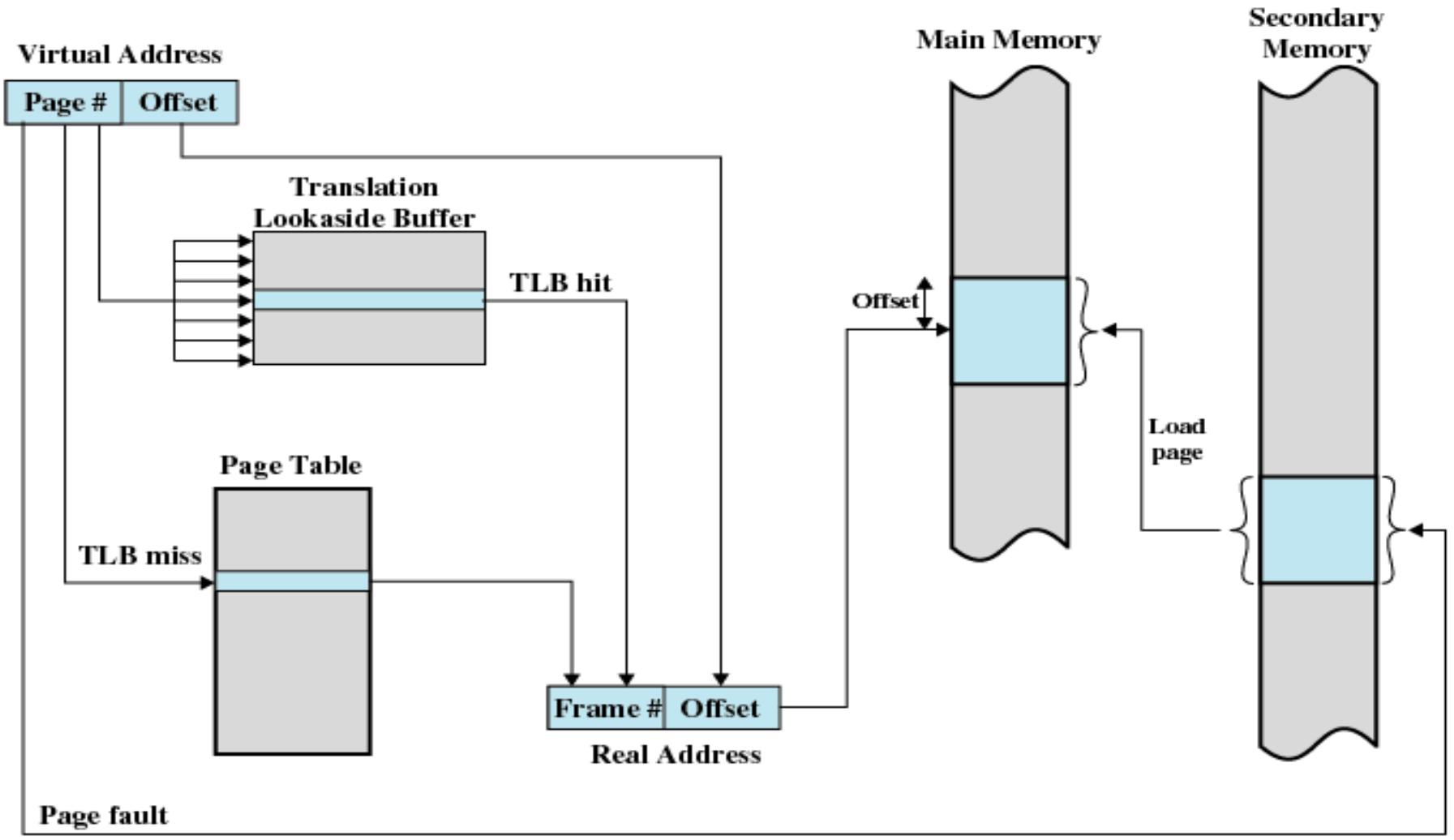


Figure 8.7 Use of a Translation Lookaside Buffer

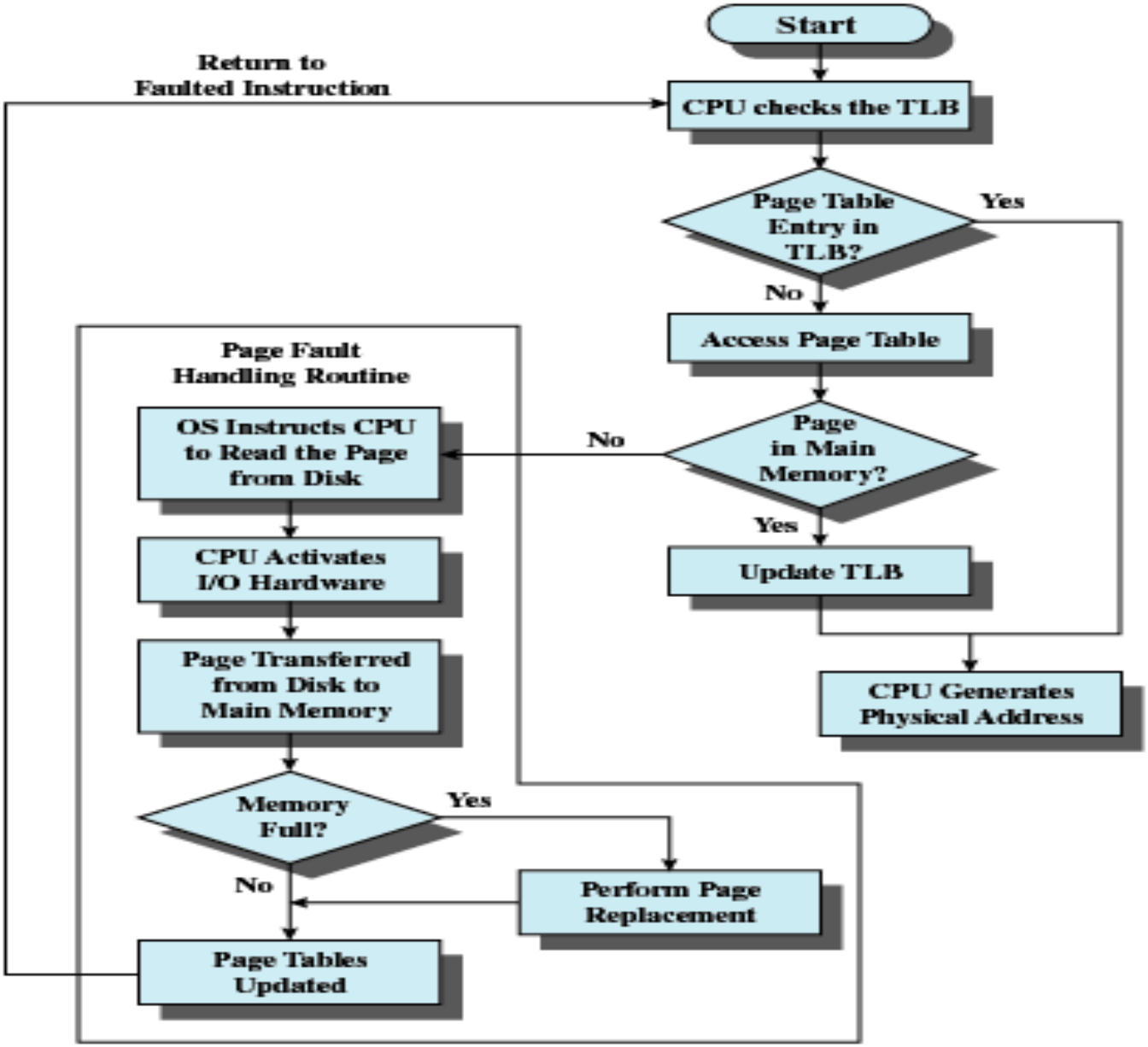


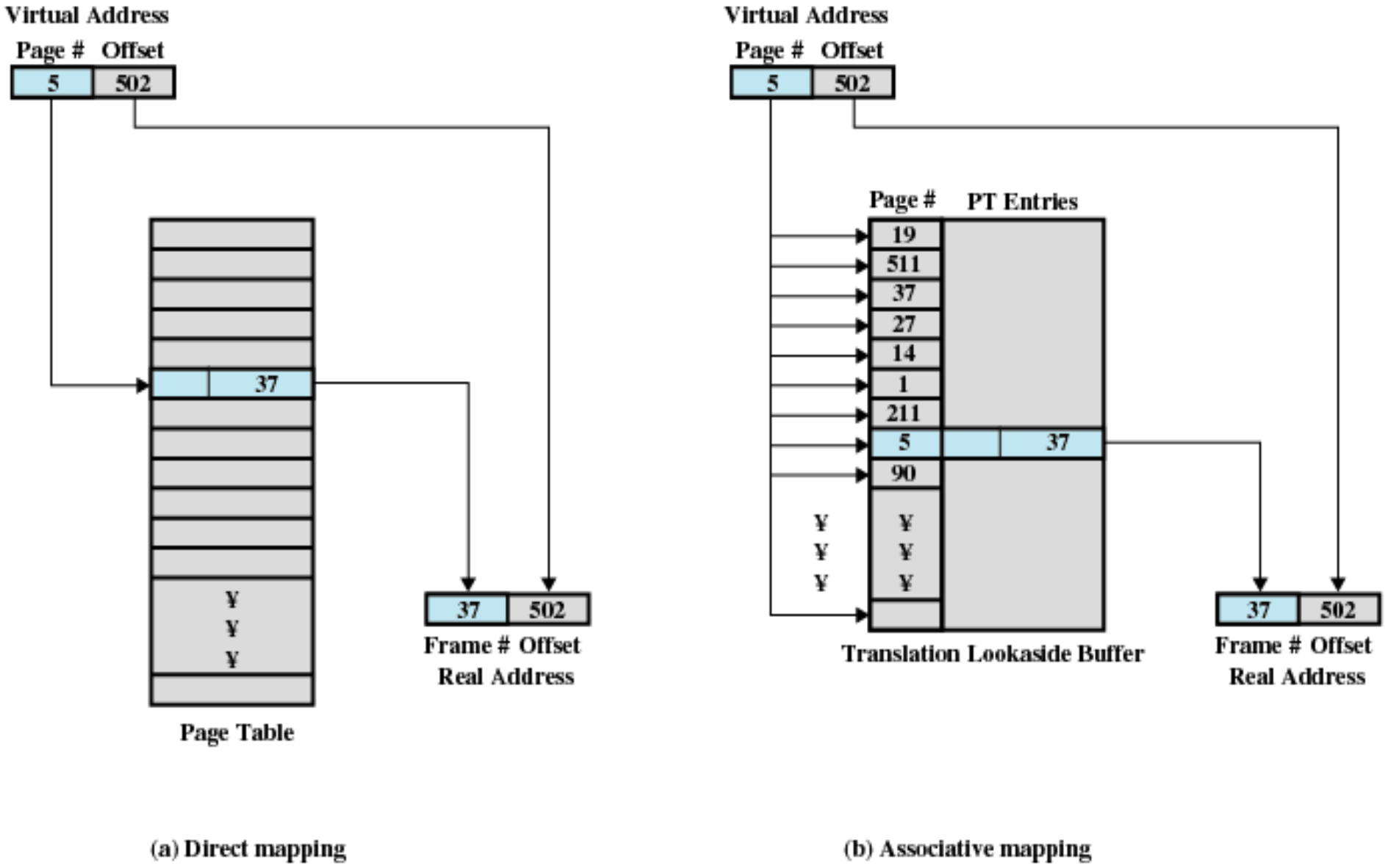
Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

Work Flow of Translation Lookaside Buffer

1. Given a virtual address, processor examines the TLB
2. If page table entry is present (TLB hit), the frame number is retrieved and the real address is formed
3. If page table entry is not found in the TLB (TLB miss), the page number is used to index the process page table

Work Flow of Translation Lookaside Buffer

4. First checks if page is already in main memory
   - If not in main memory a page fault is issued
5. The TLB is updated to include the new page entry



直接映射

关联映射

Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

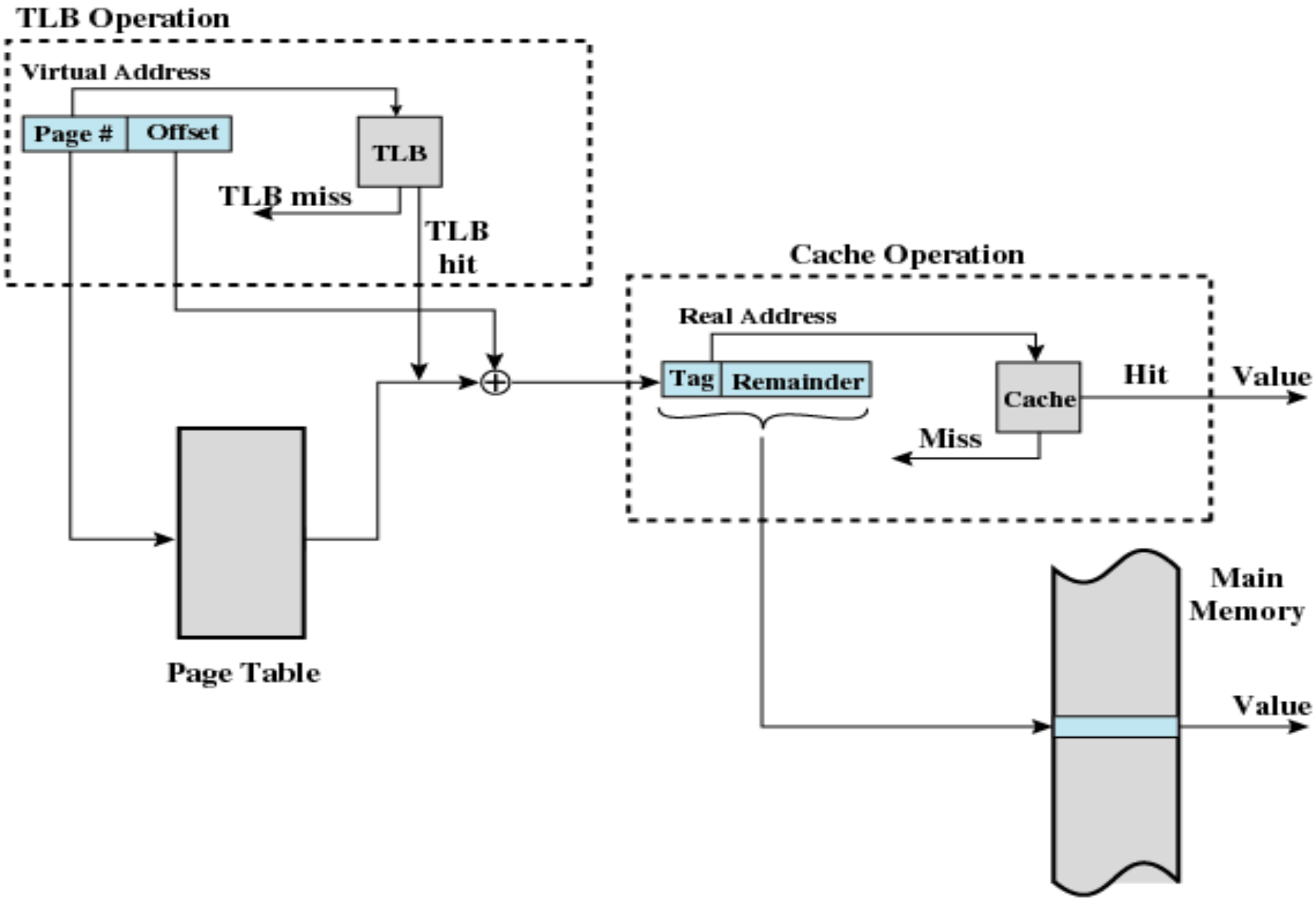


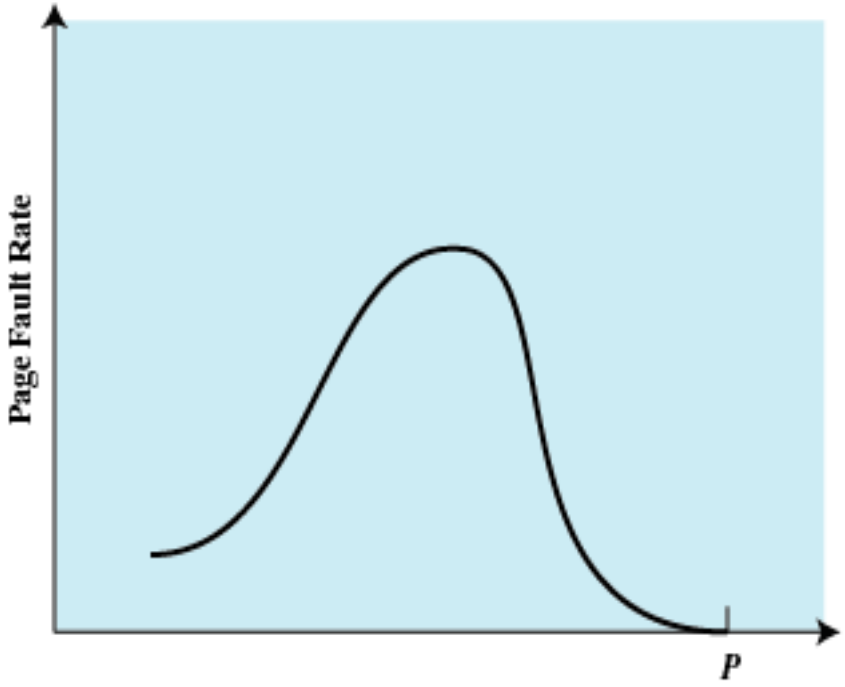
Figure 8.10 Translation Lookaside Buffer and Cache Operation

Page Size(页尺寸/页大小)

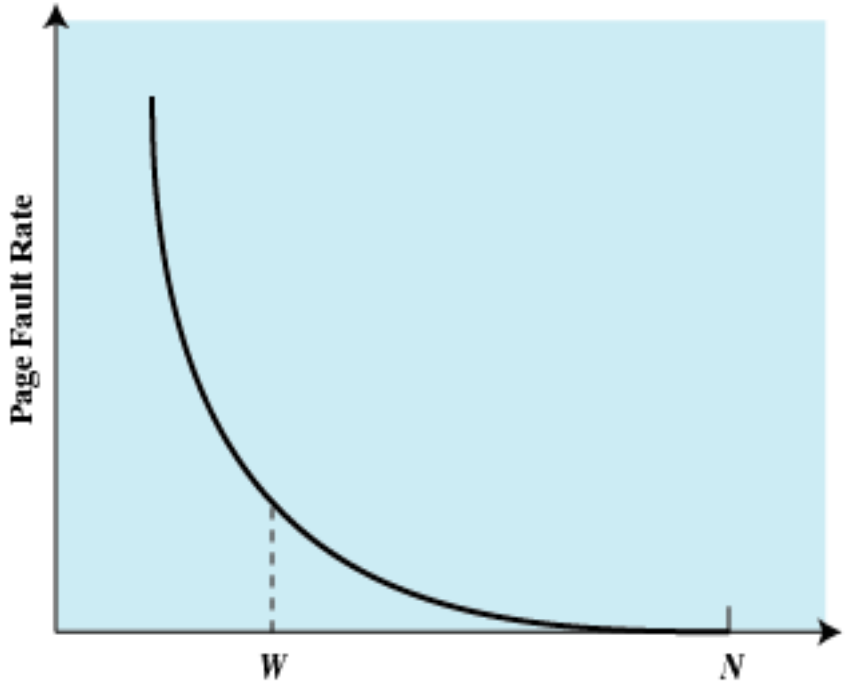
- Smaller page size, less amount of internal fragmentation(页越小，内部碎片越少)
- Smaller page size, more pages required per process，larger page tables(页越小，每个进程需要的页越多，进程的页表就越大)
- Larger page tables means large portion of page tables in virtual memory (页表越大，占有用虚存越多)
- Secondary memory is designed to efficiently transfer large blocks of data so a large page size is better

Page Size

- Small page size, large number of pages will be found in main memory(页越小，进程在内存中的页越多). After a time, the pages in memory will all contain portions of the process near recent references. Page faults rate should be low.
- Increased page size causes pages to contain locations further from any recent reference. The effect of locality is weakened. Page faults rate rises.
- However the page fault rate will begin to fall as the size of page approaches a point.



(a) Page Size



(b) Number of Page Frames Allocated

P = size of entire process
W = working set size
N = total number of pages in process

Figure 8.11 Typical Paging Behavior of a Program

Example Page Sizes

Table 8.2 Example Page Sizes

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 kbyes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
PowerPc	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

8.1 Hardware and Control Structure

- 8.1.0 Overview
- 8.1.1 Locality and Virtual Memory
- 8.1.2 Paging
- 8.1.3 Segmentation
- 8.1.4 Combined Paging and Segmentation
- 8.1.5 Protection and Sharing

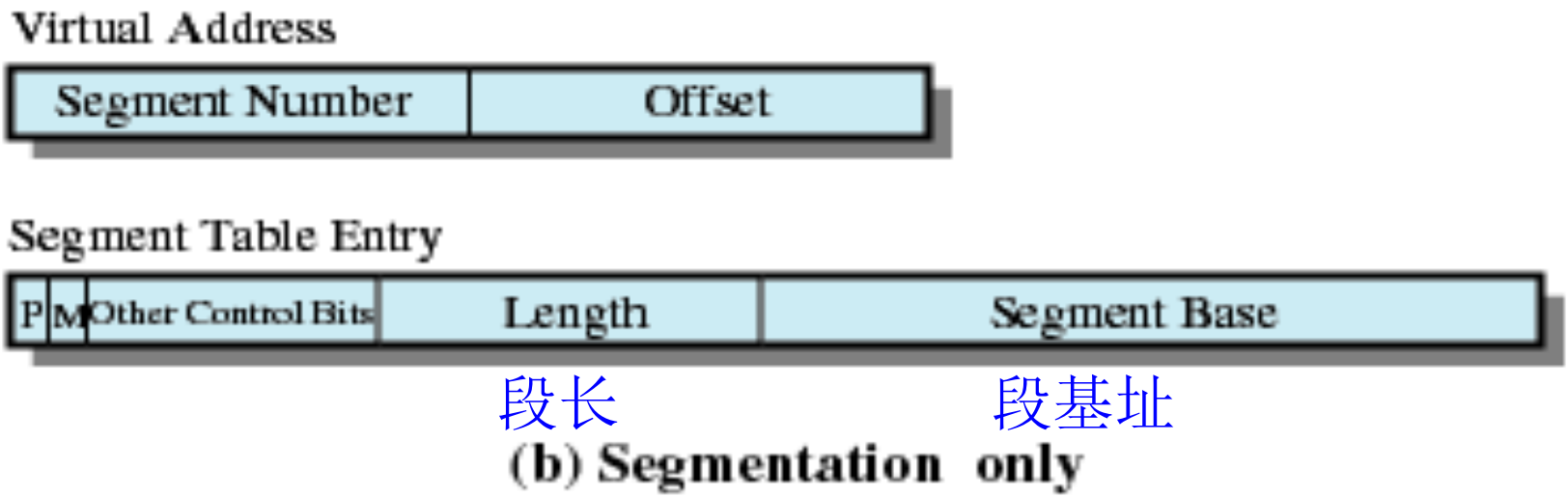
Segmentation(分段)

- Segments may be of unequal, dynamic size (大小不等), advantages:
  - Simplifies handling of growing data structures(简化不断增长的数据结构处理)
  - Allows programs to be altered and recompiled independently (允许程序独立的修改和编译)
  - Lends itself to sharing data among processes (有助于进程间共享)
  - Lends itself to protection (有利于保护)

Segment Tables(段表)

- Each process has its own segment table.
- Each entry of segment table contains the length of the segment(段长) and segment base(段基址)
- A bit is needed to determine if segment is already in main memory(存在位)
- Another bit is needed to determine if the segment has been modified since it was loaded in main memory(修改位)

Segment Table Entries



(b) Segmentation only

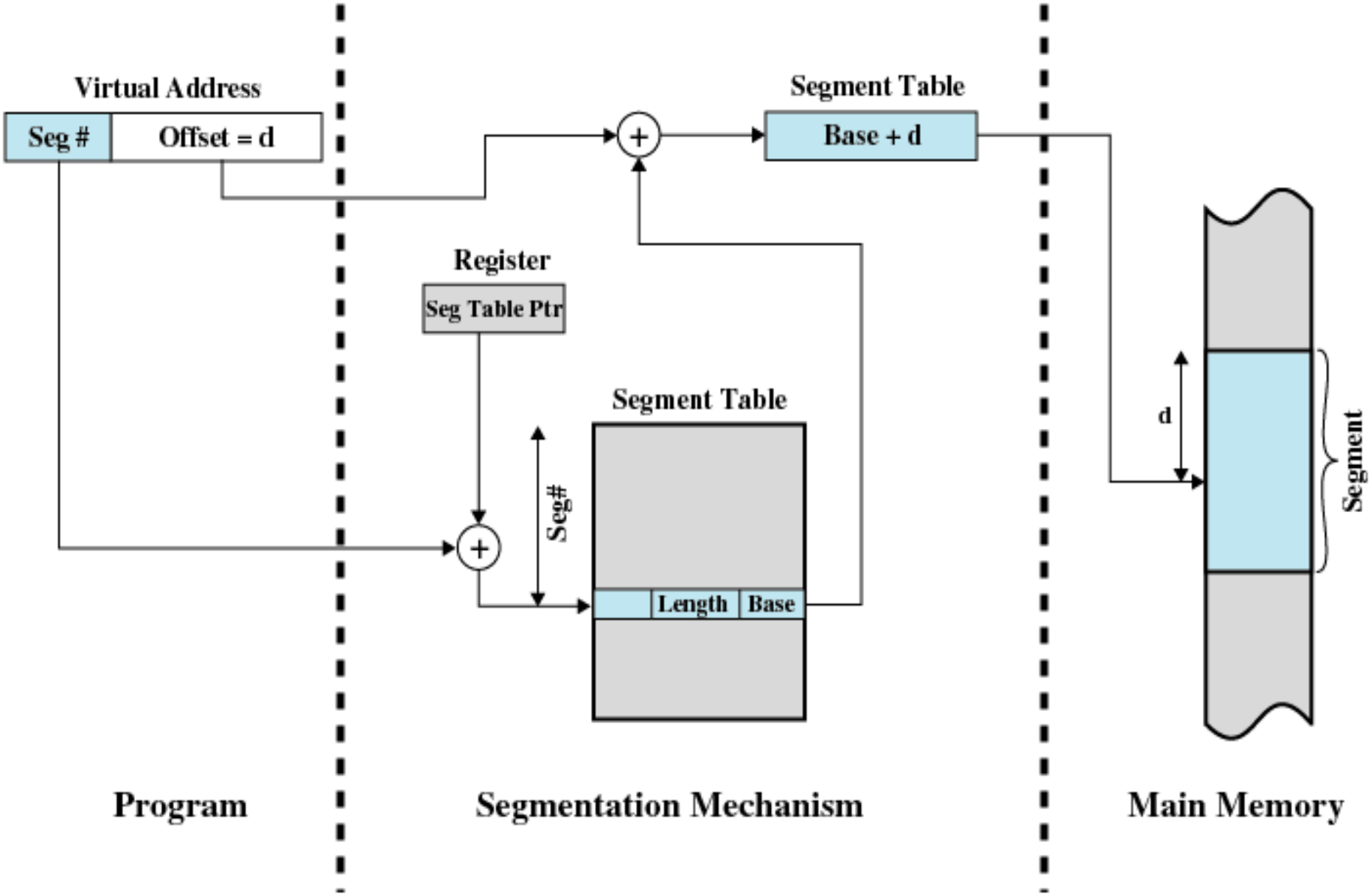


Figure 8.12 Address Translation in a Segmentation System

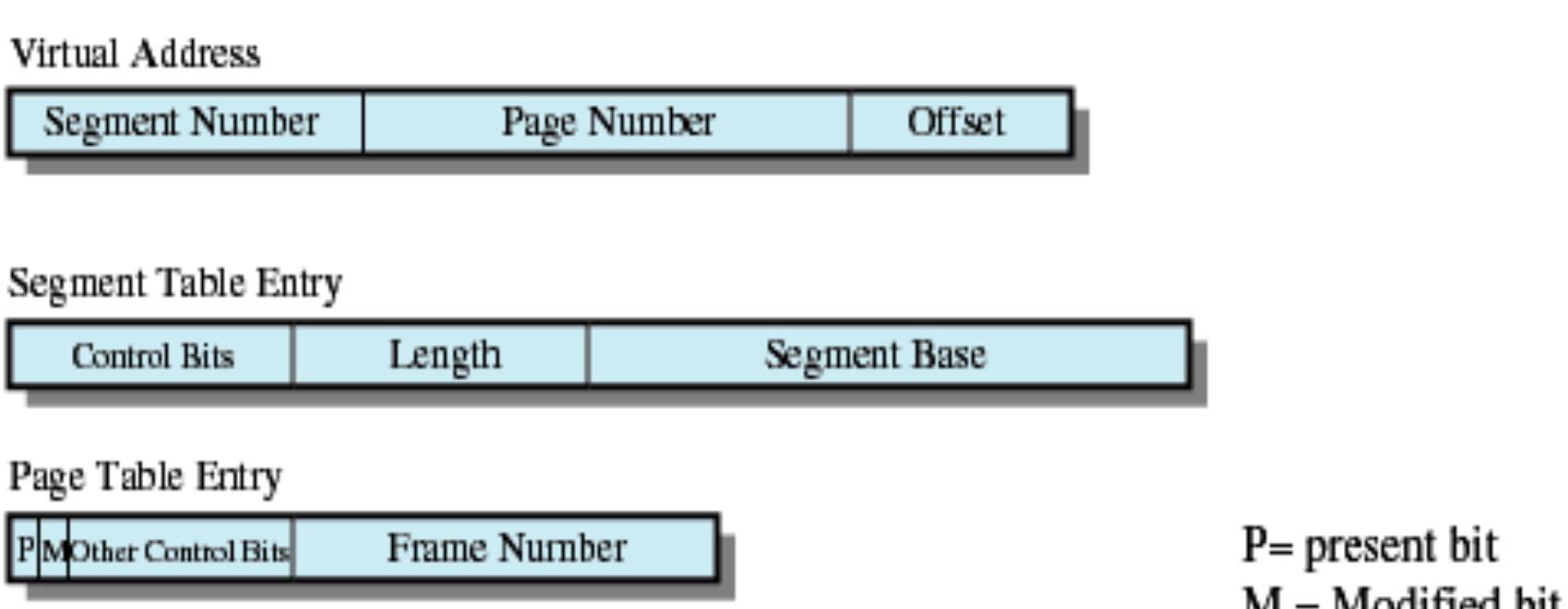
8.1 Hardware and Control Structure

- 8.1.0 Overview
- 8.1.1 Locality and Virtual Memory
- 8.1.2 Paging
- 8.1.3 Segmentation
- 8.1.4 Combined Paging and Segmentation
- 8.1.5 Protection and Sharing

Combined Paging and Segmentation

- Each Process is broken up into a number of segments and each process has a segment table.
- Each segment is, in turn broken up into a number of fixed-size pages, and each segment has a page table
- Segmentation is visible to the programmer
- Paging is transparent to the programmer

Combined Segmentation and Paging



(c) Combined segmentation and paging

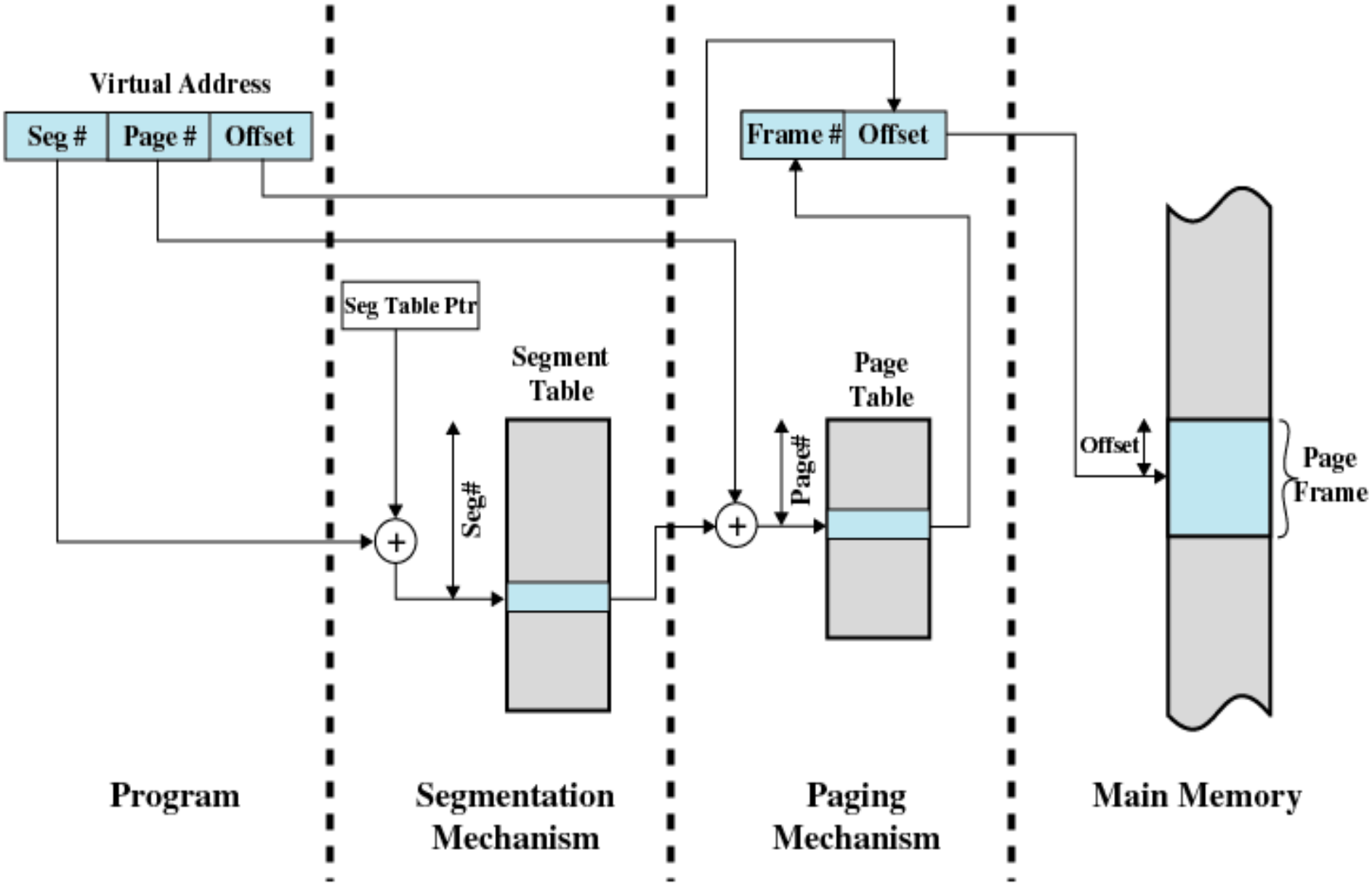


Figure 8.13 Address Translation in a Segmentation/Paging System

8.1 Hardware and Control Structure

- 8.1.0 Overview
- 8.1.1 Locality and Virtual Memory
- 8.1.2 Paging
- 8.1.3 Segmentation
- 8.1.4 Combined Paging and Segmentation
- 8.1.5 Protection and Sharing

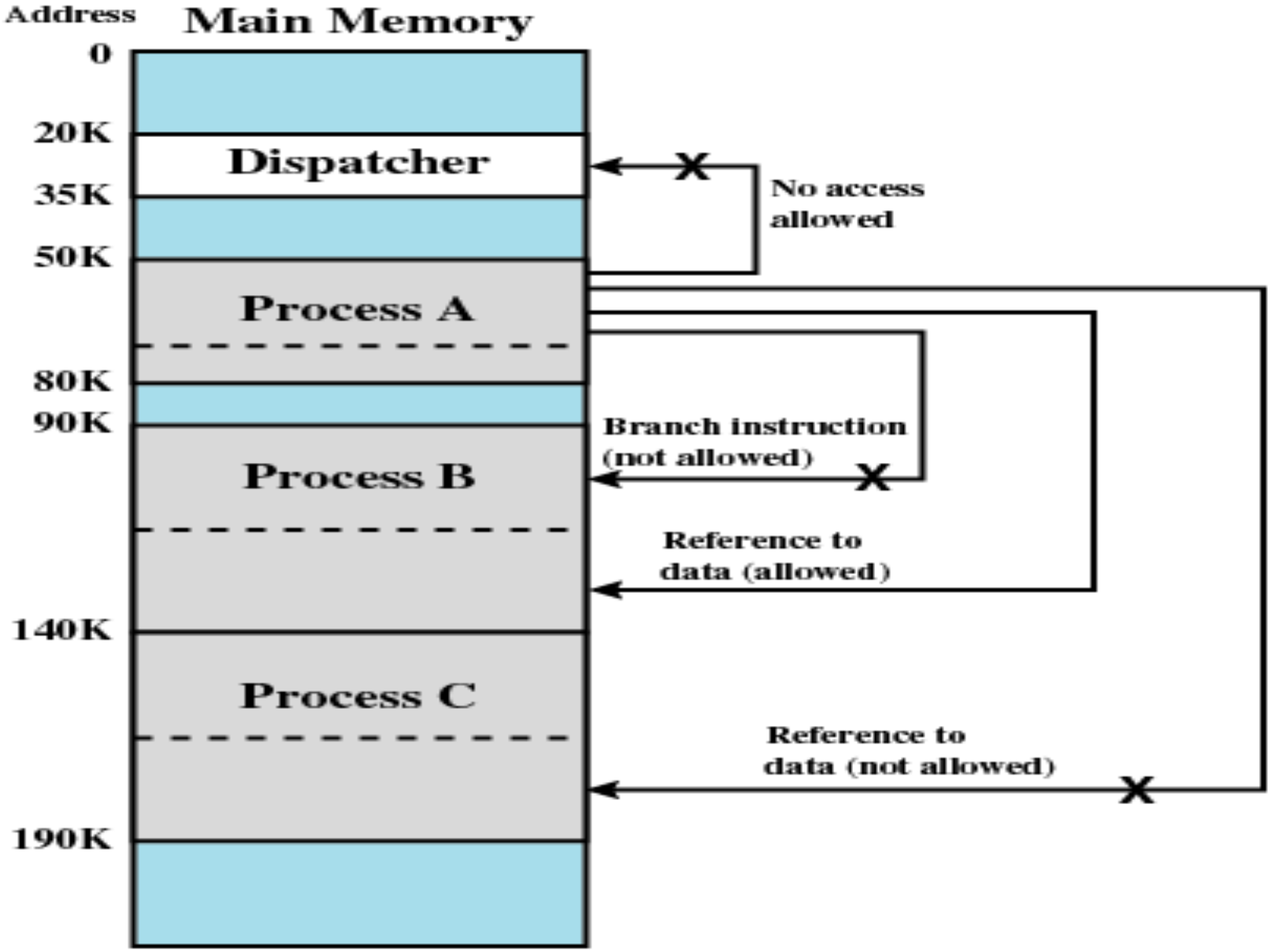


Figure 8.14 Protection Relationships Between Segments

Agenda

- 8.1 Hardware and Control Structure
- 8.2 Operating System Software
- 8.3 Summary

8.2 Operating System Software

- 8.2.1 Fetch Policy
- 8.2.2 Placement Policy
- 8.2.3 Replacement Policy
- 8.2.4 Resident Set Management
- 8.2.5 Clearing Policy
- 8.2.6 Load Control

Fetch Policy(读取策略)

- Fetch Policy: Determines when a page should be brought into memory
  - ***Demand paging***(请求分页式) only brings pages into main memory when a reference is made to a location on the page
    - Many page faults when process first started
  - ***Prepaging***(预约分页式) brings in more pages than needed
    - More efficient to bring in pages that reside contiguously on the disk

8.2 Operating System Software

- 8.2.1 Fetch Policy
- 8.2.2 Placement Policy
- 8.2.3 Replacement Policy
- 8.2.4 Resident Set Management
- 8.2.5 Clearing Policy
- 8.2.6 Load Control

Placement Policy(放置策略)

- Placement Policy: Determines where in real memory a process piece is to reside
  - Important in a segmentation system and nonuniform memory access(NUMA, 非一致存储器访问) system (放置策略对分段系统和NUMA系统影响大)
  - Paging or combined paging with segmentation hardware performs address translation and placement is usually irrelevant(放置策略对分页和段页式系统影响不大)

8.2 Operating System Software

- 8.2.1 Fetch Policy
- 8.2.2 Placement Policy
- 8.2.3 Replacement Policy
- 8.2.4 Resident Set Management
- 8.2.5 Clearing Policy
- 8.2.6 Load Control

Replacement Policy(置换策略)

- Placement Policy: Which page is replaced?
  - Page removed should be the page least likely to be referenced in the near future（最近访问可能性最小的页）
  - Most policies predict the future behavior on the basis of past behavior（基于过去的行为预测未来的行为）

Frame Locking(帧锁定)

- Frame Locking
  - If frame is locked, it may not be replaced, some examples:
    - Kernel of the OS
    - Important Control structures of OS
    - I/O buffers
  - Associate a lock bit with each frame

Basic Replacement Algorithms

1. Optimal policy(OPT, 最佳)
   - Selects for replacement that page for which the time to the next reference is the longest(下次访问距当前时间最长的页)
   - Impossible to have perfect knowledge of future events

Basic Replacement Algorithms

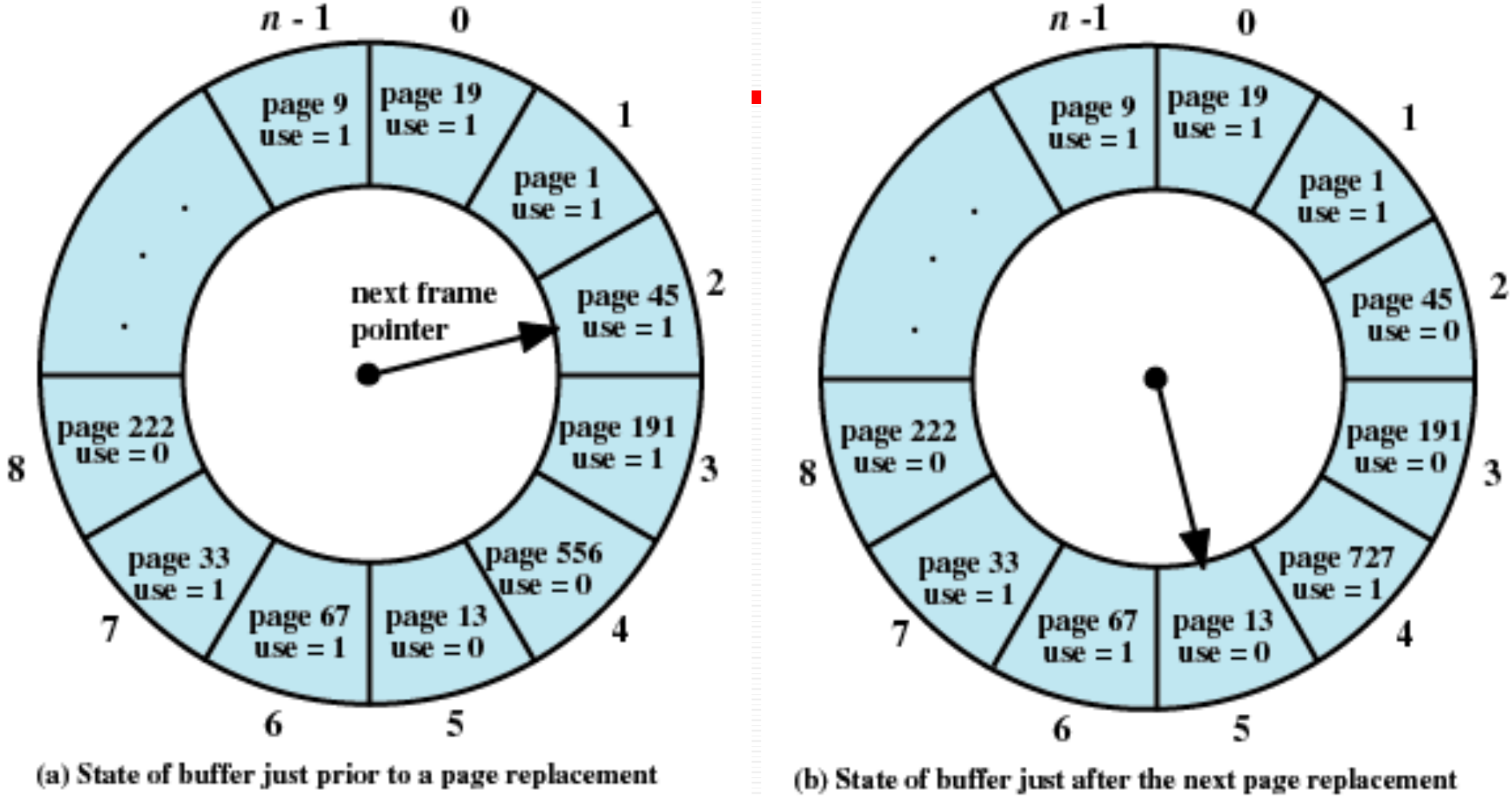
2. Least Recently Used (LRU, 最近最少使用)
   - Replaces the page that has not been referenced for the longest time(上次访问距当前时间最长的页)
   - By the principle of locality, this should be the page least likely to be referenced in the near future
   - Each page could be tagged with the time of last reference. This would require a great deal of overhead.

Basic Replacement Algorithms

3. First-in, first-out (FIFO, 先进先出)
   - Treats page frames allocated to a process as a circular buffer(把分配给进程的页帧看做是一个循环缓冲区)
   - Pages are removed in round-robin(循环) style
   - Simplest replacement policy to implement
   - Page that has been in memory the longest is replaced
   - These pages may be needed again very soon

Basic Replacement Algorithms

4. Clock Policy(时钟策略)
   - Additional bit called a use bit(使用位)is association with every page
   - When a page is first loaded in memory, the use bit is set to 1
   - When the page is referenced, the use bit is set to 1
   - When it is time to replace a page, the first frame encountered with the use bit set to 0 is replaced
   - During the search for replacement, each use bit set to 1 is changed to 0



(a) State of buffer just prior to a page replacement

(b) State of buffer just after the next page replacement

Figure 8.16 Example of Clock Policy Operation

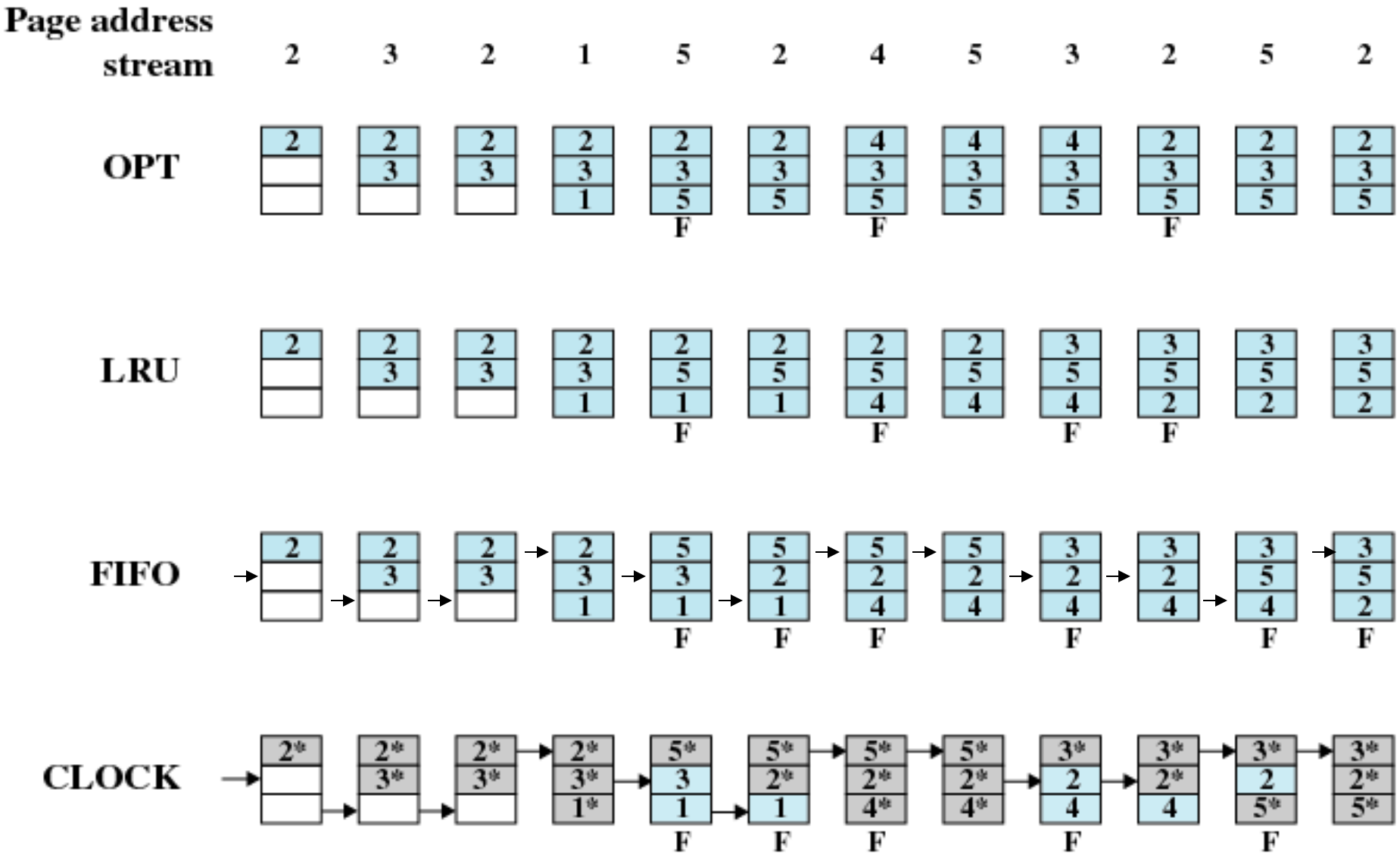


Figure 8.15 Behavior of Four Page-Replacement Algorithms

Comparison of Placement Algorithms

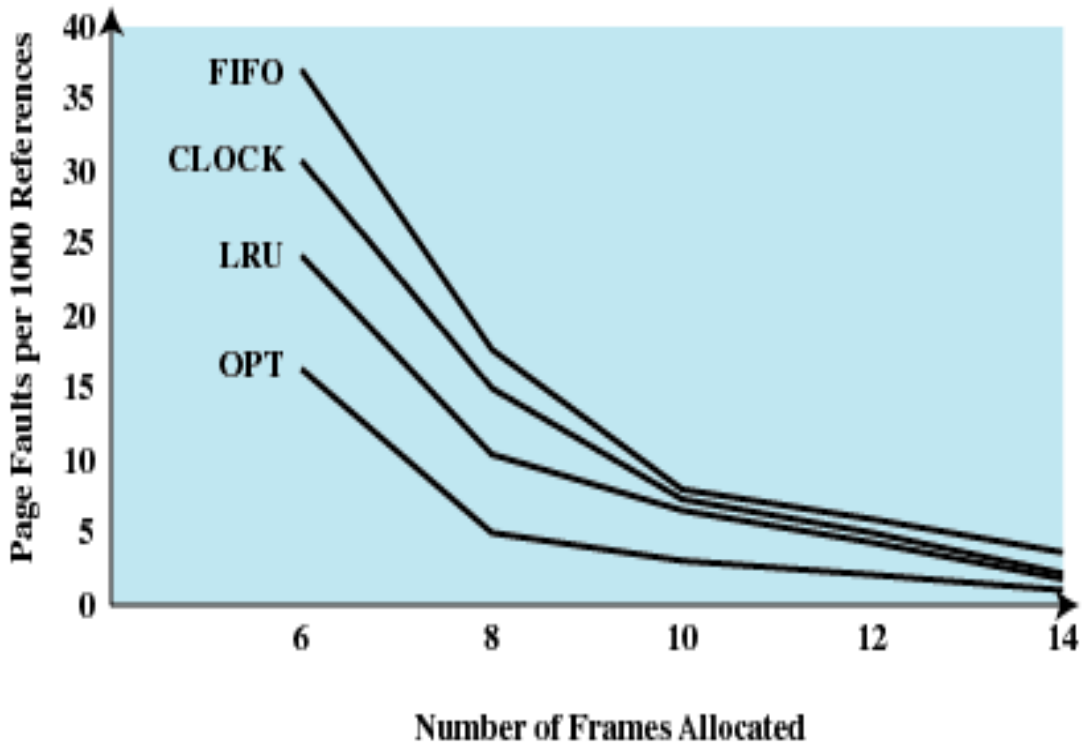


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

Clock Policy Using Use Bit and Modified Bit

- Not accessed recently, not modifie (u=0;m=0)
- Not accessed recently, modified (u=0;m=1)
- Accessed recently, not modified (u=1;m=0)
- Accessed recently, modified (u=1;m=1)

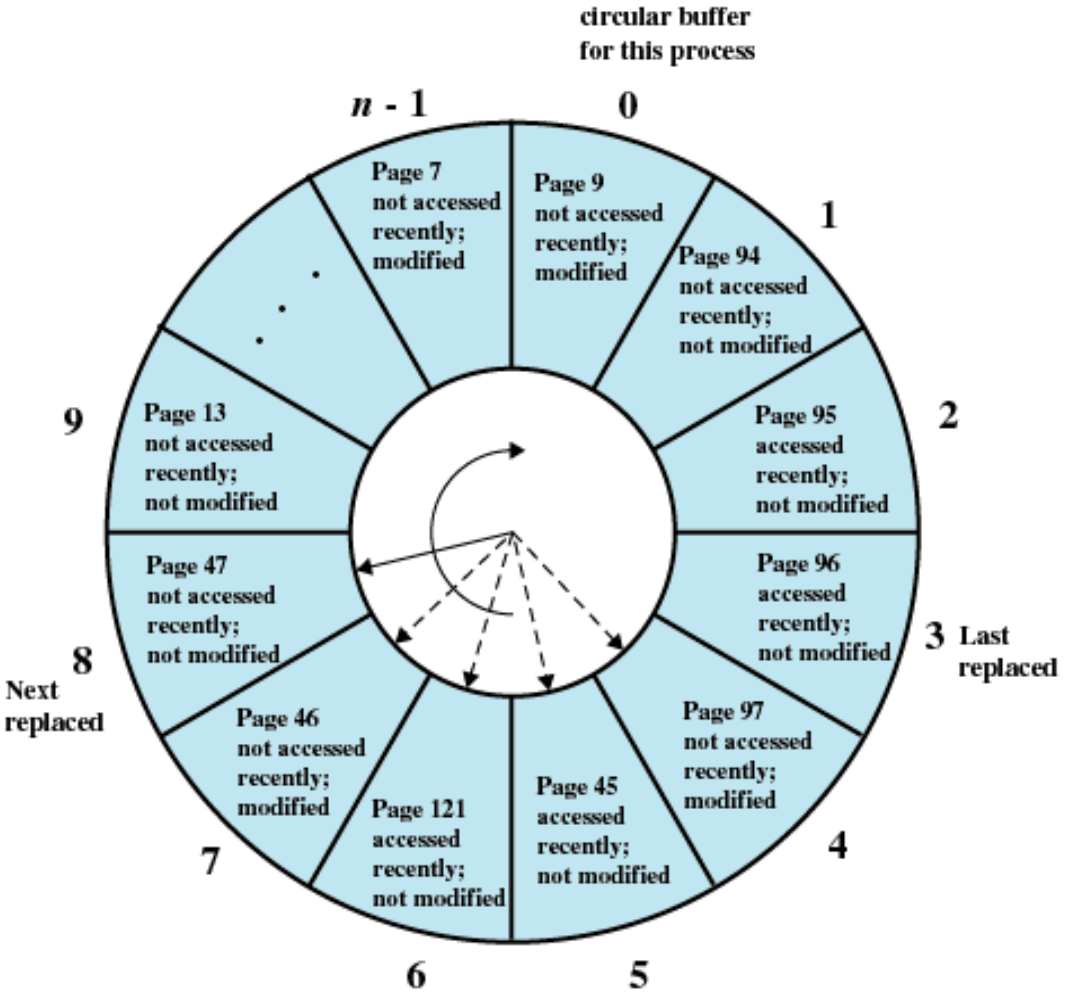


Figure 8.18 The Clock Page-Replacement Algorithm [GOLD89]

Clock Policy Using Use Bit and Modify Bit

1. Beginning at the current position of the pointer, scan the frame buffer. During this scan, make no changes to the use bit. The first frame encountered with (u = 0; m = 0) is selected for replacement.
2. If step 1 fails, scan again, looking for the frame with (u = 0; m = 1).The first such frame encountered is selected for replacement. During this scan, set the use bit to 0 on each frame that is bypassed.
3. If step 2 fails, the pointer should have returned to its original position and all of the frames in the set will have a use bit of 0. Repeat step 1 and, if necessary, step 2. This time, a frame will be found for the replacement.

Page Buffering(页缓冲)

- Replaced page is added to one of two lists
  - Free page list if page has not been modified (空闲页表)
  - Modified page list (修改页表)
  - Modified pages are written out in clusters rather than one at a time

8.2 Operating System Software

- 8.2.1 Fetch Policy
- 8.2.2 Placement Policy
- 8.2.3 Replacement Policy
- 8.2.4 Resident Set Management
- 8.2.5 Clearing Policy
- 8.2.6 Load Control

Resident Set Size(驻留集大小)

- Fixed-allocation(固定分配)
  - Gives a process a fixed number of pages within which to execute
  - When a page fault occurs, one of the pages of that process must be replaced
- Variable-allocation(可变分配)
  - Number of pages allocated to a process varies over the lifetime of the process

Replace Scope(替换范围)

- Local Replace Policy(局部置换)
  - Chooses only among the resident pages of the process that generated the page fault in selecting a page to replace
- Global Replace Policy(全局置换)
  - Considers all unlocked pages in main memory as candidates for replacement, regardless of which process owns a particular page

Fixed Allocation, Local Scope(固定分配，局部置换)

- Decide ahead of time the amount of allocation to give a process (事先确定给进程分配多少页)
- If allocation is too small, there will be a high page fault rate (如果太少，导致高缺页率)
- If allocation is too large there will be too few programs in main memory (如果分配太多，内存中驻留的程序会过少)

Variable Allocation, Global Scope(可变分配，全局置换)

- Easiest to implement
- Adopted by many operating systems
- Operating system keeps list of free frames
- Free frame is added to resident set of process when a page fault occurs
- If no free frame, replaces one from any of the resident processes

Variable Allocation, Local Scope(可变分配，局部置换)

- When new process added, allocate number of page frames based on application type, program request, or other criteria
- When page fault occurs, select page from among the resident set of the process that suffers the fault
- Reevaluate(重新评估) allocation of processes from time to time and increase or decrease allocation to improve overall performance

8.2 Operating System Software

- 8.2.1 Fetch Policy
- 8.2.2 Placement Policy
- 8.2.3 Replacement Policy
- 8.2.4 Resident Set Management
- 8.2.5 Clearing Policy
- 8.2.6 Load Control

Cleaning Policy(清除策略)

- Demand cleaning(请求式清除)
  - A page is written out only when it has been selected for replacement
- Precleaning(预约式清除)
  - Pages are written out in batches before they are needed

Cleaning Policy

- Best approach uses page buffering
  - Replaced pages are placed in two lists
    - Modified and unmodified
  - Pages in the modified list are periodically written out in batches
  - Pages in the unmodified list are either
    - reclaimed if referenced again
    - or lost when its frame is assigned to another page

8.2 Operating System Software

- 8.2.1 Fetch Policy
- 8.2.2 Placement Policy
- 8.2.3 Replacement Policy
- 8.2.4 Resident Set Management
- 8.2.5 Clearing Policy
- 8.2.6 Load Control

Load Control(加载控制)

- Determines the number of processes that will be resident in main memory
- Too few processes, many occasions when all processes will be blocked and much time will be spent in idle or swapping
- Too many processes will lead to thrashing

Multiprogramming Level(多道程序设计级)

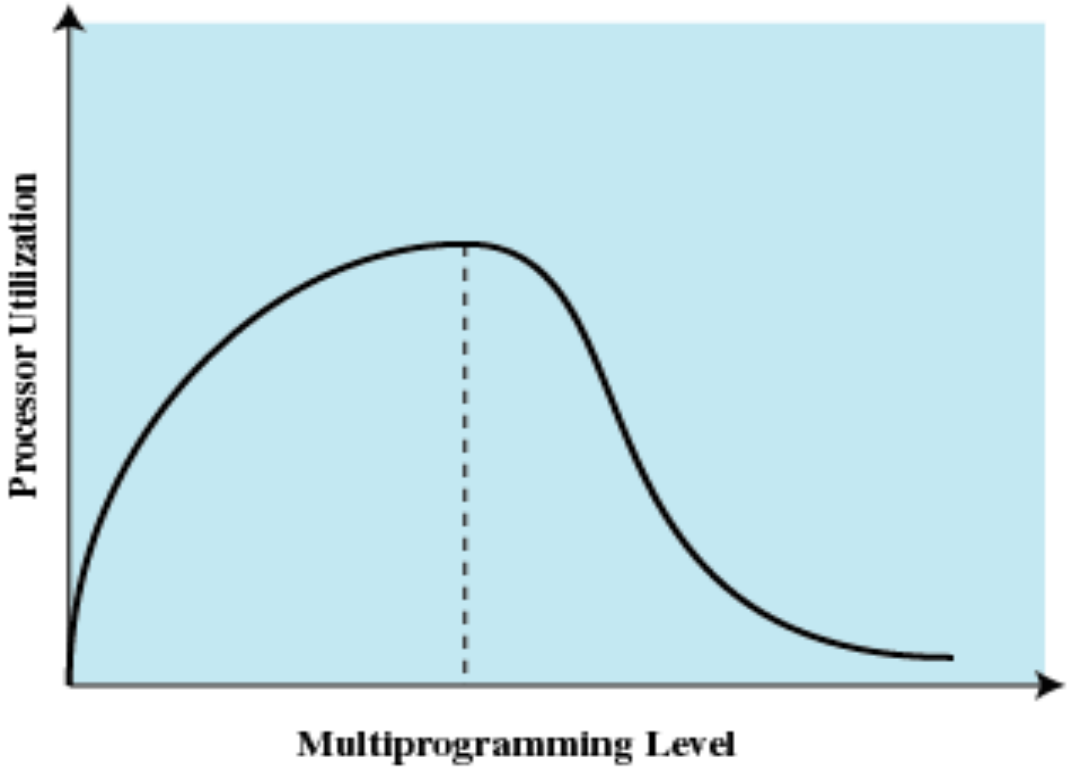


Figure 8.21 Multiprogramming Effects

Process Suspension(进程挂起)

1. Lowest priority process(最低优先级进程)
2. Faulting process(页错误进程)
   - This process does not have its working set in main memory so it will be blocked anyway
3. Last process activated(最后被激活的进程)
   - This process is least likely to have its working set resident

Process Suspension

4. Process with smallest resident set(驻留集最小的进程)
   - This process requires the least future effort to reload
5. Largest process(占用空间最大的进程)
   - Obtains the most free frames
6. Process with the largest remaining execution window(具有最大剩余执行窗口的进程)

Agenda

- 8.1 Hardware and Control Structure
- 8.2 Operating System Software
- 8.3 Summary