Virtual Memory (虚拟内存)

Chapter 8

Simple Memory Management

- **Goal:** *To keep many processes in main memory to allow multiprogramming*

- **Limitation:** *require processes to be* ***entirely*** *in main memory before it can execute*
  - *Overlay is a technique to solve this limitation. (Programmers have to do some extra work)*

Characteristics
Paging and Segmentation

- *All memory references within a process are logical addresses that are **dynamically translated** into physical addresses **at run time***
  - *A process may be swapped in and out of main memory such that it occupies different regions at different times during its lifetime*
- *A process may be broken up into pieces that do **NOT** need to located **contiguously** in main memory*
  - *This is permitted by the combination of dynamic run-time address translation and the page/segment table.*

Characteristics
Program Execution

- *Programs often have a lot of code for error handling, which are seldom executed.*
- *The memory allocated for arrays, lists or tables are often much more than real requirements*
- *Some options or features of a program is seldom used.*
- *At each moment, only a small fraction of a program is needed.*

The Breakthrough

- *Thanks to the two characteristics, we come to the* ***breakthrough***
  - *It is* ***not necessary*** *that all of the pages or all of the segments of a process be in main memory during execution*
- *How may this be accomplished?*

Virtual Memory
虚拟存储器

- *Allows the execution of processes that are not completely in main memory*
  - *Programs can be larger than physical memory*
- *Virtual memory abstracts main memory into an extremely large, uniform array of storage*
  - *Frees programmers from the concerns of memory-storage limitations*
- *Allows processes to share files easily and to implement shared memory*
- *Provides an efficient mechanism for process creation*

Execution of a Program (1)

- *Operating system brings into main memory a few pieces of the program*
  - *Resident set - portion of process that is in main memory*
- *An interrupt is generated when an address is needed that is not in main memory*
  - *Operating system places the process in a blocking state and takes control*

Execution of a Program (2)

- *The OS will need to bring into main memory the piece of the process that contains the logical address that caused the access fault*
  - *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 which causes the operating system to place the affected process in the Ready state*

Advantages of Virtual Memory

- *More processes may be maintained in main memory*
  - *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*
- *Less I/O would be needed to load or swap user programs into memory, so user programs would run faster*

Types of Memory

- **Real memory (实存-实存储器)**
  - A process executes only in main memory, thus this memory is referred to as real memory
- **Virtual memory (虚存-虚拟内存)**
  - A programmer or user perceives a potentially much larger memory, which is allocated on disk. It is referred to as virtual memory
  - Allows for effective multiprogramming and relieves the user of tight constraints of main memory

Thrashing
抖动 / 颠簸

- *What is thrashing?*
  - *The processor spends most of its time swapping pieces rather than executing user instructions*
- *Swapping out a piece of a process just before that piece is needed*
  - *The OS will have to get that piece again almost immediately*
  - *Too much of this leads to thrashing.*
- *The OS must be clever in guessing which pieces are least likely to be used in near future.*

Is Virtual Memory Practical? Principle of Locality

- *The **principle of locality** states that program and data references within a process tend to cluster*
  - *Only a few pieces of a process will be needed over a short period of time*
  - *It is possible to make intelligent guesses about which pieces will be needed in the future*
- *The performance of processes in virtual memory environment also confirm the principle of locality.*

Support Needed for Virtual Memory

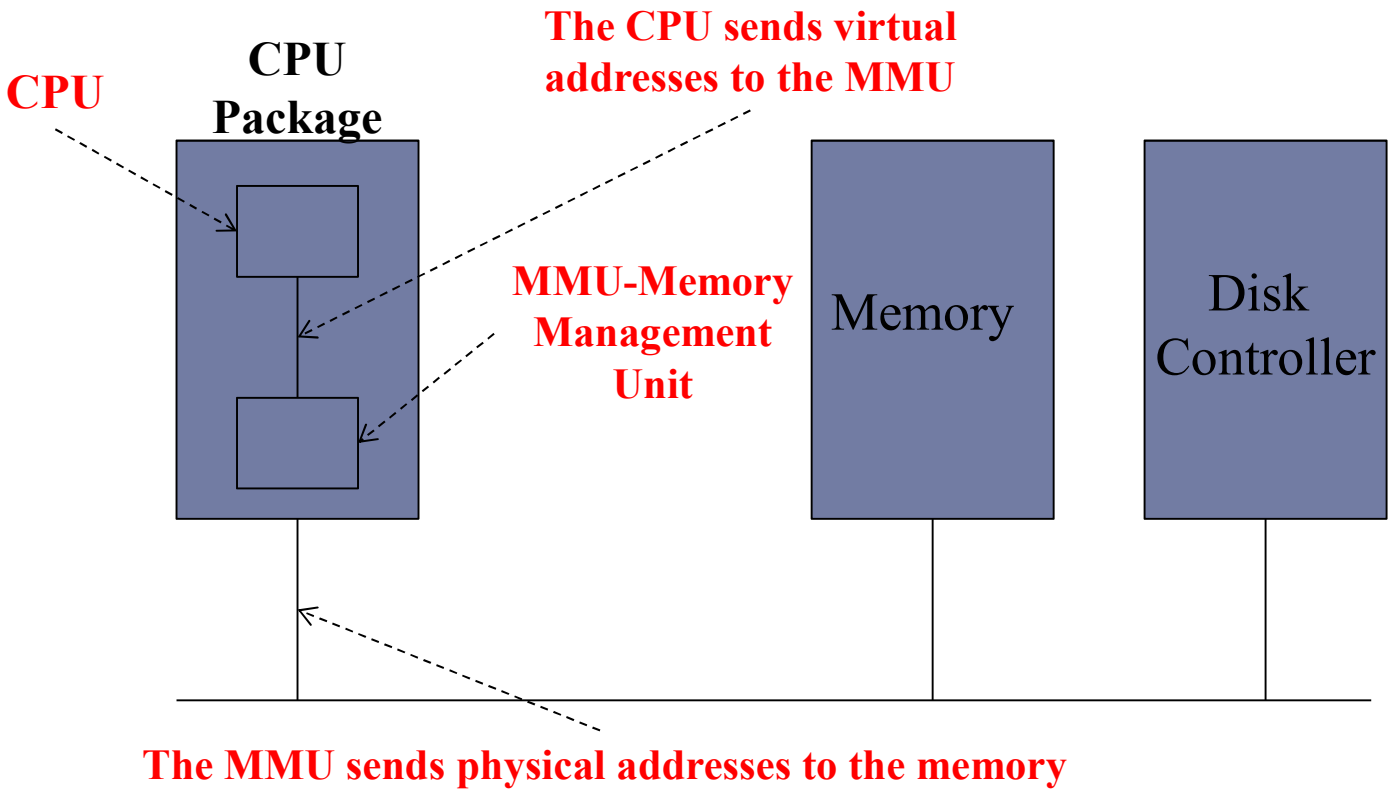
- **Thus, the principle of locality suggests that a virtual memory scheme may work. For it to be practical and effective, two ingredients are needed**
  - First, there must be **hardware support** for the paging and/or segmentation scheme to be employed
  - Second, the **OS must be able to manage** the movement of pages and/or segments between secondary memory and main memory

Contents

- ***Hardware Aspect of Virtual Memory***
  - ***Paging, Segmentation, and combined paging and segmentation***
- ***Issues in Designing Virtual Memory Facility in Operating Systems***

Hardware and Control Structures
硬件与控制结构

Paging


CPU
CPU Package
The CPU sends virtual addresses to the MMU
MMU-Memory Management Unit
Memory
Disk Controller
The MMU sends physical addresses to the memory

Page Table Entry (1)
Basic Information

- **Simple Paging**
  - Each process has its own page table
  - When all of its pages are loaded into main memory, the page table is created and loaded into main memory
  - Each page table entry contains the ***frame number*** *of the corresponding page in main memory*
- **Virtual memory based on paging**
  - *The page table entries become* ***more complex***
  - *The exact layout of entry is highly machine-dependent, but the basic information is roughly the same*

Page Table Entry (2) Control Bits

- *A bit is to indicate whether the corresponding page is present (P) in memory or not.*
  - *Because only some of the pages of a process may be in main memory*
- *Another modified bit is to indicate whether the page has been altered since it was last loaded into main memory*
  - *If no change has been made, the page does not have to be written to the disk when it needs to be replaced*
- *Yet another referenced bit is set whenever a page is referenced, either for reading or writing*
  - *Its value is to help the OS choose a page to evict when a page fault occurs.*

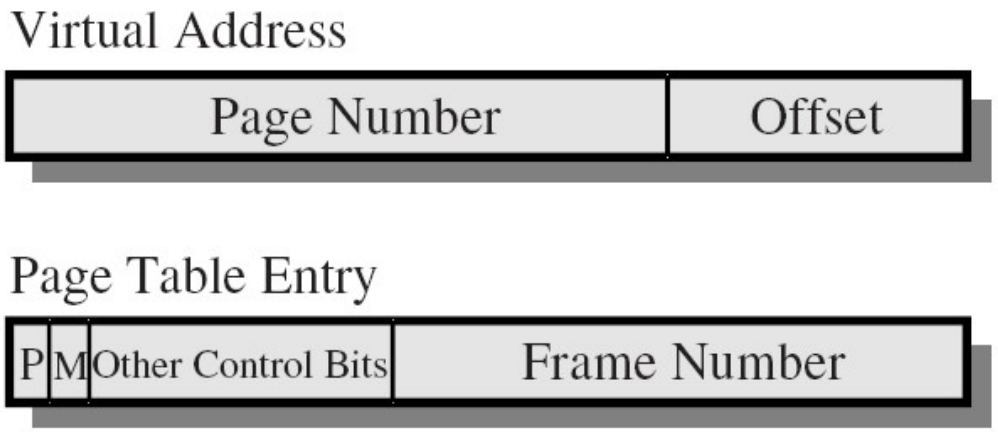
Page Table Entry (3)
Control Bits

- *The protection bits tell what kinds of access are permitted.*
  - *In the simplest form, this field contains 1 bit, with 0 for read/write and 1 for read-only*
  - *A more sophisticated arrangement is having 3 bits, one bit each for reading, writing, and executing the page.*

Page Table Entry (4)
Should PTE Contain Disk Address?

- *The disk address that holds the page when it is not in memory is **NOT** part of the page table*
  - *The page table holds only the information that the hardware needs to translate a virtual address to a physical address*
  - *Information the OS needs to handle page faults is kept in software tables inside the OS. The hardware does not need it.*

Typical Memory Management Formats
Paging Only



A hardware implementation

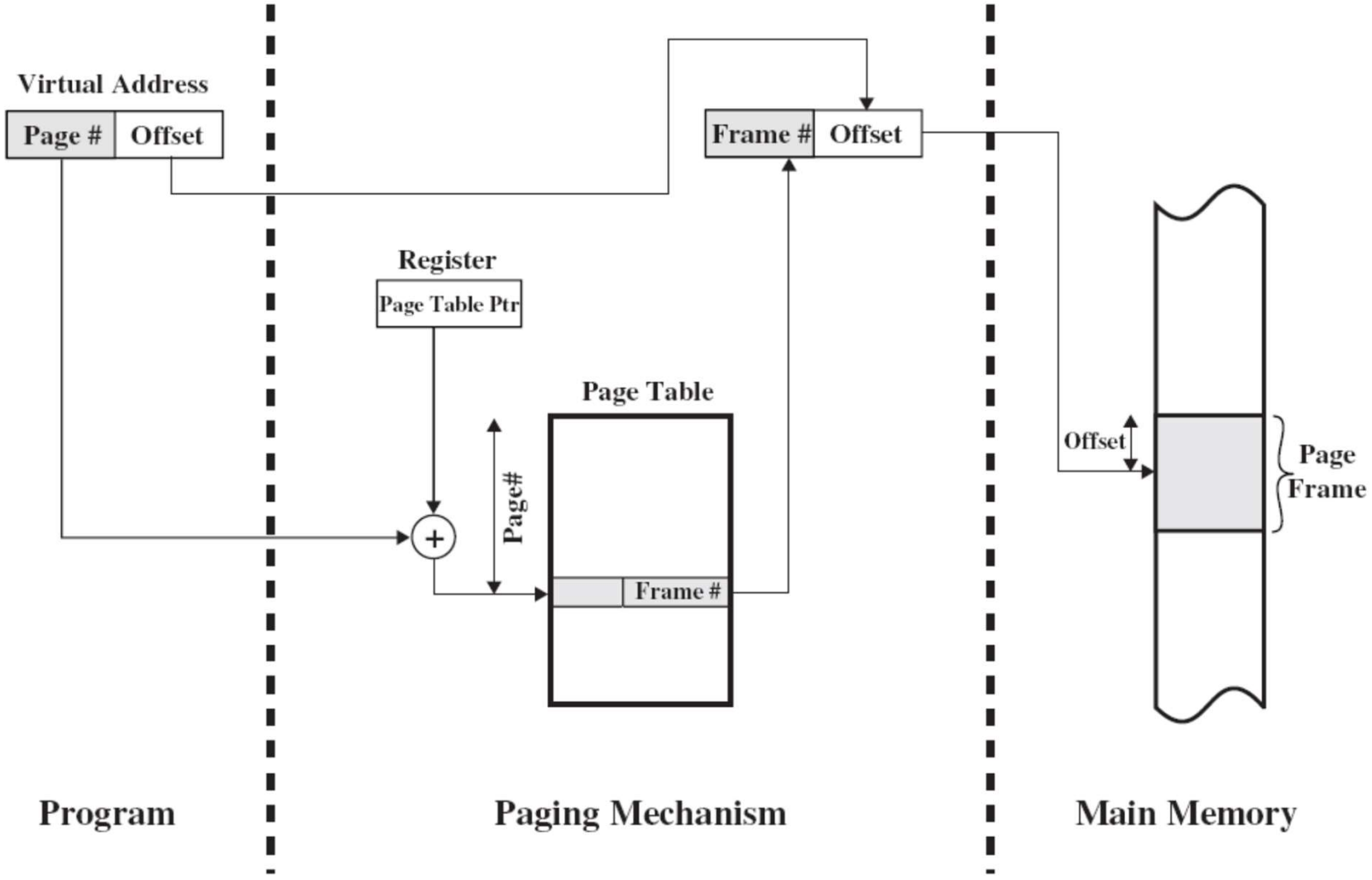


Figure 8.3 Address Translation in a Paging System

Page Table Structure (1)

- *Mathematically speaking, the page table is a function, with the virtual page number as argument and the physical frame number as result*
- *Two major issues must be faced:*
  - *The page table can be extremely large.*
  - *The mapping must be fast*

Page Table Structure (2)
Two major issues

- ***The page table can be extremely large.***
  - *Modern computers use virtual addresses of at least 32 bits. With a 4-KB page size, a 32-bit address space has 1 million pages*
  - *A 64-bit address space has more than you want to contemplate (超过你所预期的)*
- ***The mapping must be fast***
  - *The virtual-to-physical mapping must be done on every memory reference.*
  - *Each instruction requires to make 1, 2, or more page table references, (for a typical instruction has an instruction word, and often a memory operand as well)*

Page Table Structure (3)
Page Table Design and Hardware Solutions

- **Thus, the need for large, fast page mapping is significant.**
  - The simplest design is to have a single page table consisting of an array of fast hardware registers
  - The page table can be entirely in main memory
  - Two-level page tables (Extended to Multilevel page tables)
  - Inverted page table
  - Translation lookaside buffer

Page Table In Registers?

- **When a process is started up, the OS loads the registers with the process' page table, taken from a copy kept in main memory.**
  - **Advantages: During process execution, no more memory references to the page table is required.**
  - **Disadvantages: (1) It is potentially expensive if the page table is large. (2) Loading the full page table at process switch hurts performance**

Page Table Entirely in Main Memory?

- ***All the hardware needs is a single register that points to the start of the page table.***
  - *Advantages: This design allows the memory map to be changed at a process switch by reloading one register.*
  - *Disadvantages: Requiring one or more memory references to read page table entries during the execution of each instruction*
- ***Thus, this approach is rarely used in its most pure form, but some variations have much better performance.***

Two-Level Page Tables (1)

- **The entire page table may take up too much main memory**
  - **Because each process can occupy huge amounts of virtual memory**
- **Thus, most virtual memory schemes store page tables in virtual memory rather than in real memory.**
  - **When a process is running, at least a part of its page table must be in main memory**
- **Some processors make use of a two-level scheme to organize large page tables**

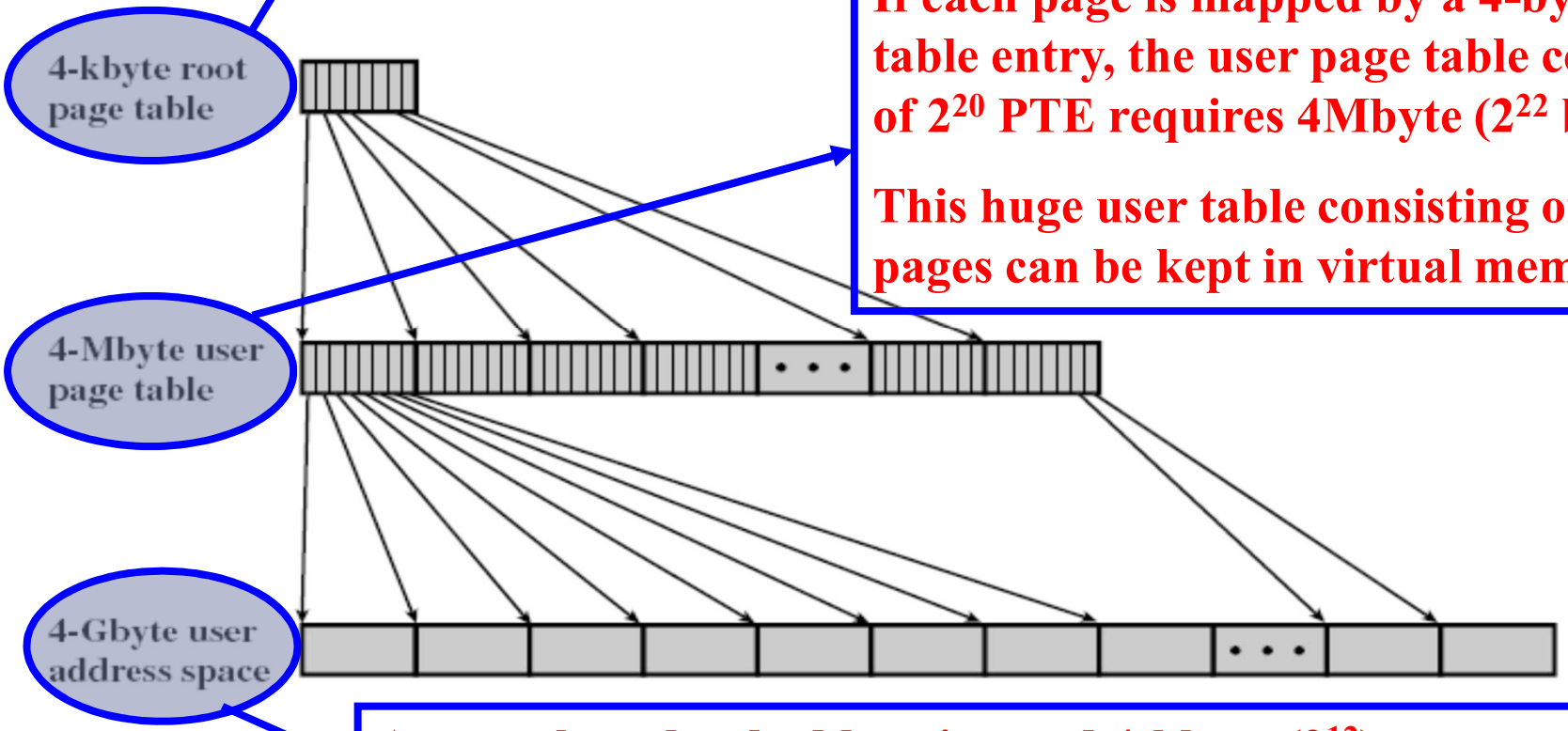
Two-Level Page Table (2)
Two-Level Scheme

- In this scheme, there is a page directory (页目录)
  - Each entry in the page directory points to a page table
- If the length of the page directory is *X*, and the maximum length of a page table is *Y*
  - Then a process can consist of up to *X*×*Y* pages
- Typically, the *maximum length of page table is restricted to be equal to one page*
  - For example: the Pentium processor

The user page table can be mapped by a root page table with 2^{10} PTEs occupying 4 kbyte (2^{12}) of main memory

If each page is mapped by a 4-byte page table entry, the user page table composed of 2^{20} PTE requires 4Mbyte (2^{22} bytes)

This huge user table consisting of 2^{10} pages can be kept in virtual memory



Assume byte-level addressing and 4-kbyte (2^{12}) pages.
The 4-Gbyte virtual address space is composed of 2^{20} pages

Figure 8.4 A Two-Level Hierarchical Page Table [JACO98a]

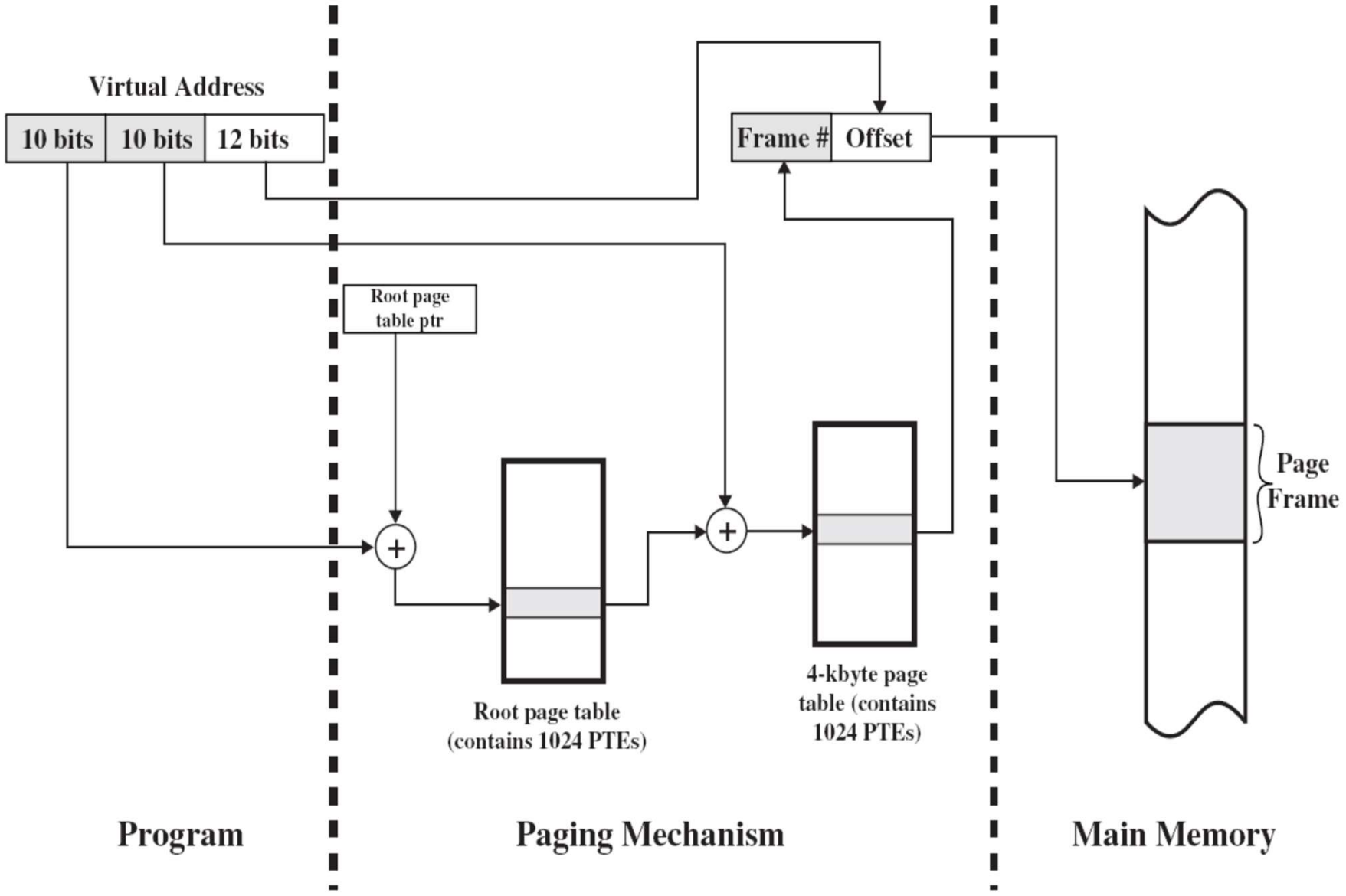


Figure 8.5 Address Translation in a Two-Level Paging System

How about multilevel page table for 64-bit virtual address spaces?

- **Multilevel page tables are generally considered inappropriate for 64-bit computers**
  - If the address space is now 2^{64} bytes, with 4-KB pages, we need six or seven levels of paging, which is a prohibitive number of memory accesses to translate each logical address

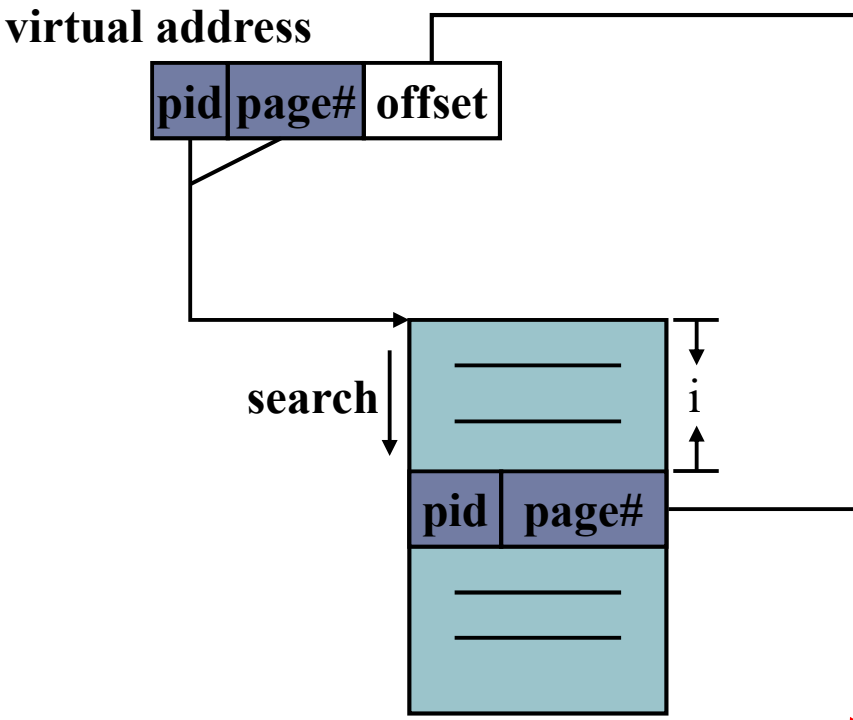
Inverted Page Tables (1)
反向页表

- **If the address space consists of 2^{32} bytes, with 4-KB per page, then over 1 million page table entries are needed**
  - The page table will have to be at least 4 megabytes at a bare minimum.
  - On large systems, this size is probably doable (可行的)
- **However, if the address space consists of 2^{64} bytes, with 4-KB per page, we need a page table with 2^{52} entries.**
  - If each entry is 8 bytes, the table is over 30 million gigabytes
  - Consequently, a different solution is needed for 64-bit paged virtual address spaces. *Inverted page table is one such solution*

Inverted Page Table (2)

- **Inverted page table (反向页表) has been used on the Power PC and on IBM's AS/400**
- **Each entry keeps track of which (process, virtual page) is located in the page frame**
  - ***Only one page table* is in the system**
  - **There is one entry per page frame in real memory, rather than one entry per page of virtual address space.**
  - **Thus, a fixed proportion of real memory is required for the table regardless of the number of processes or virtual pages supported**

Inverted Page Table (3)



When a memory reference occurs, the inverted page table is searched to match <pid, page#>.

If a match is found at entry i, then the physical address <i, offset> is generated

If no match is found, then an illegal address access has been attempted.

This search takes too long, because the whole table might need to be searched for a match

Inverted Page Table (4)

- **Although inverted page table saves vast amount of space, the virtual-to-physical translation becomes much harder**
- **A hash table can be used to accelerate the searching**
  - **It can limit the search to one (or at most a few) page-table entries**
  - **Of course, each access to the hash table adds a memory reference to the procedure, so one address transformation requires at least two real-memory reads: one for the hash table entry and one for the page table.**

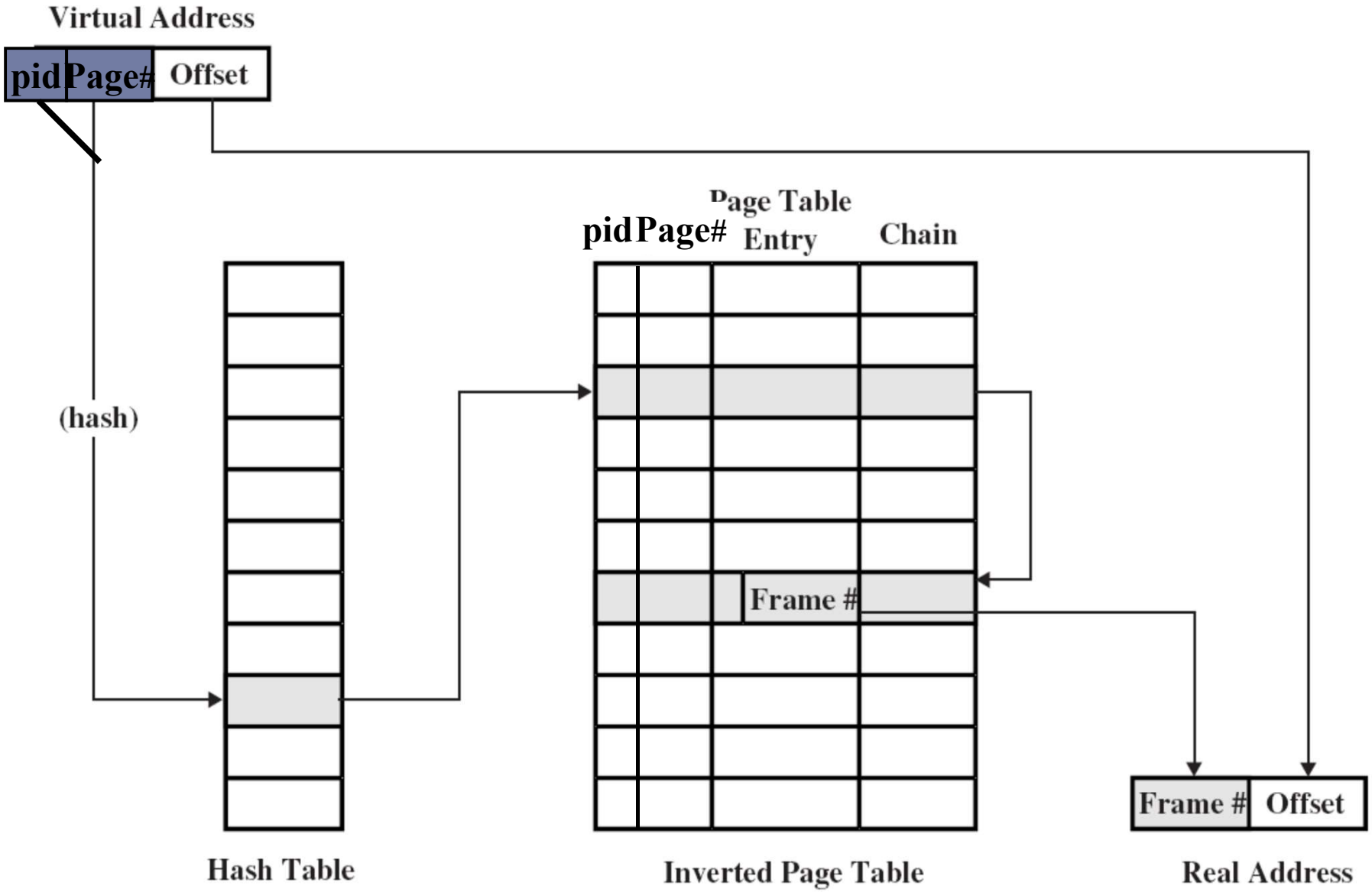
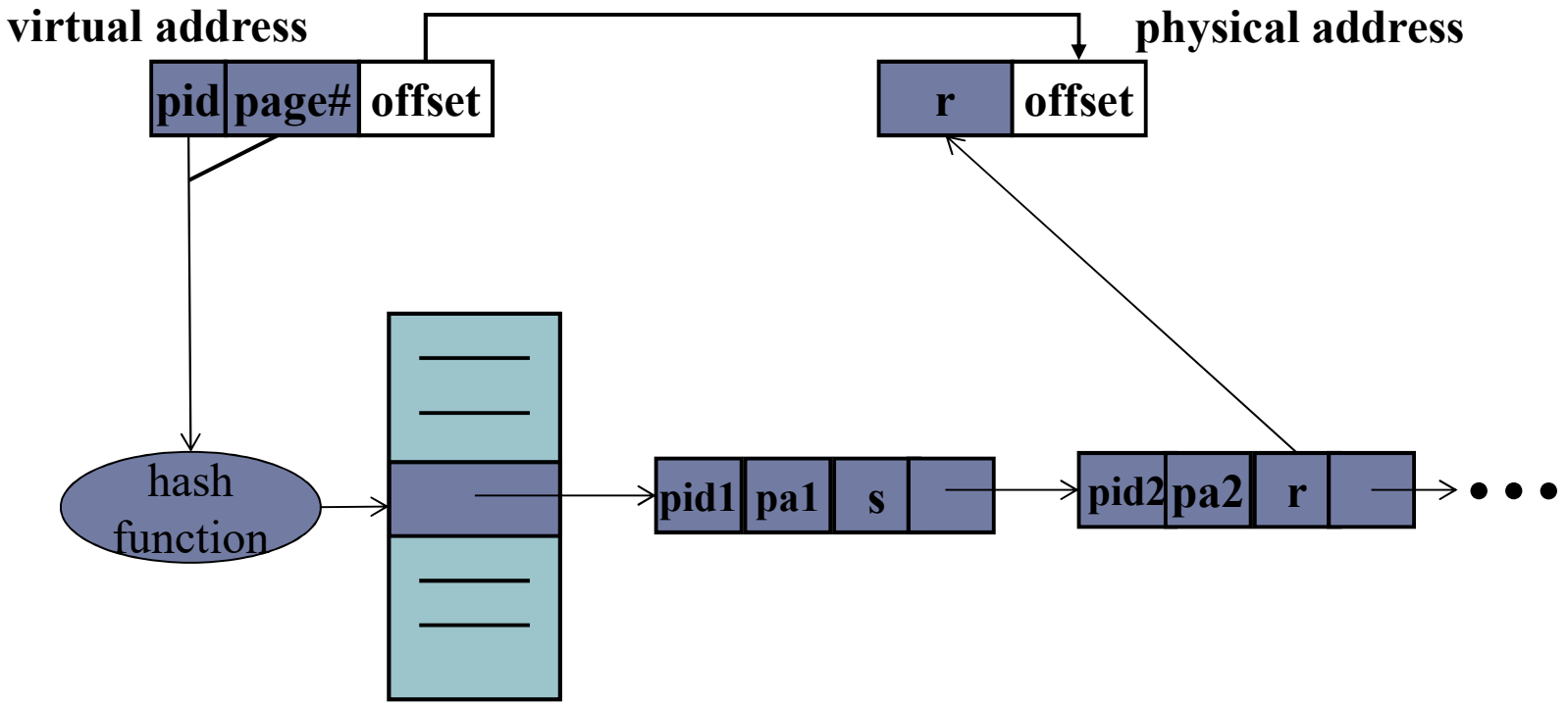


Figure 8.6 Inverted Page Table Structure

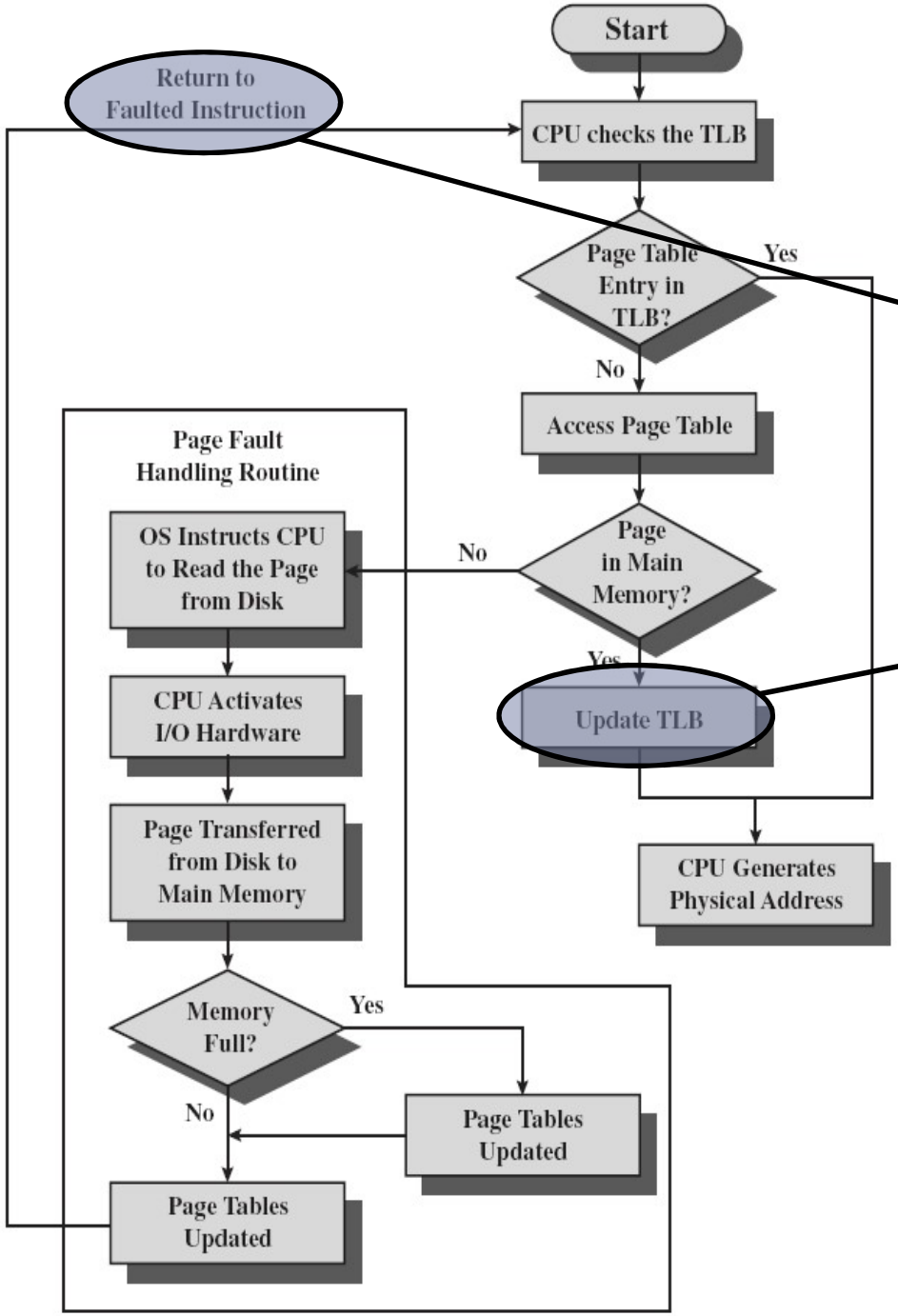

virtual address
physical address
pid
page#
offset
r
offset
hash function
pid1
pa1
s
pid2
pa2
r

Translation Lookaside Buffer (1)
转换旁视缓冲器/快表

- ***Each virtual memory reference can cause two physical memory accesses: one to fetch the page table; the other to fetch the data***
  - *A straightforward virtual memory scheme would double the memory access time*
- ***To overcome this problem a high-speed cache (高速缓存) is set up for page table entries***
  - *called the TLB - Translation Lookaside Buffer*
  - *It maps virtual addresses to physical addresses without going through the page table*

Translation Lookaside Buffer (2)

- *It contains page table entries that have been most recently used*
  - *It functions in the same way as a memory cache*
- *Given a virtual address, the processor hardware will first examine the TLB by comparing the virtual page number with all the entries in TLB simultaneously.*
  - *If the desired page table entry (virtual page number) is present in the TLB (a "TLB hit"), then the frame number is retrieved directly from TLB and the real address is formed without going to the page table*
  - *If not found (TLB miss), then the processor uses the page number to index the process page table*



Re-do the instruction that causes the page fault

It evicts one entry from the TLB and replaces it with the page table entry just looked up

When an entry is purged from the TLB, the modified bit is copied back into the page table entry in memory

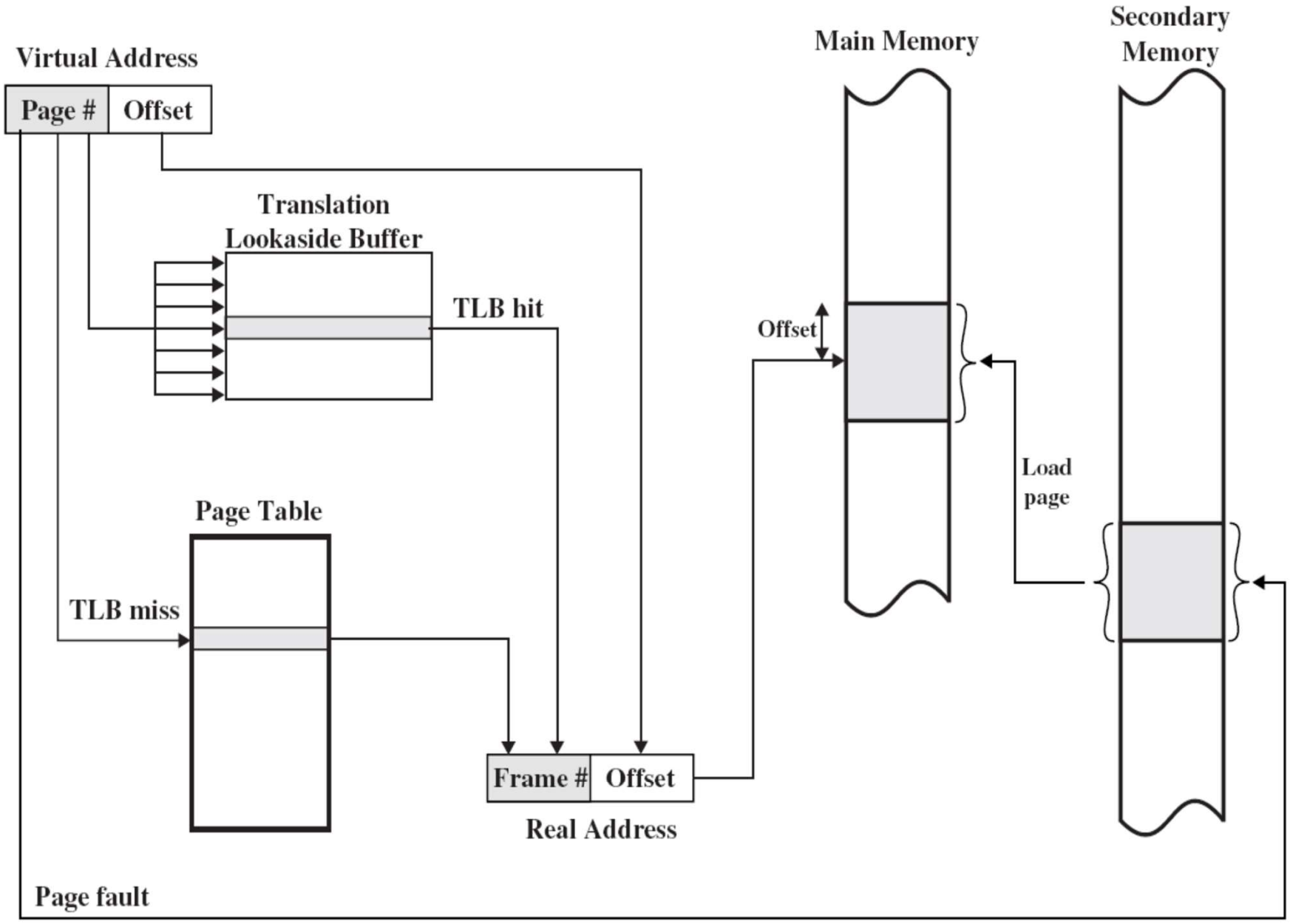


Figure 8.7 Use of a Translation Lookaside Buffer

Translation Lookaside Buffer (3)

- **The processor is equipped with hardware that allows it to interrogate simultaneously a number of TLB entries to determine if there is a match on page number**
  - **This technique is referred to as associative mapping (关联映射)**

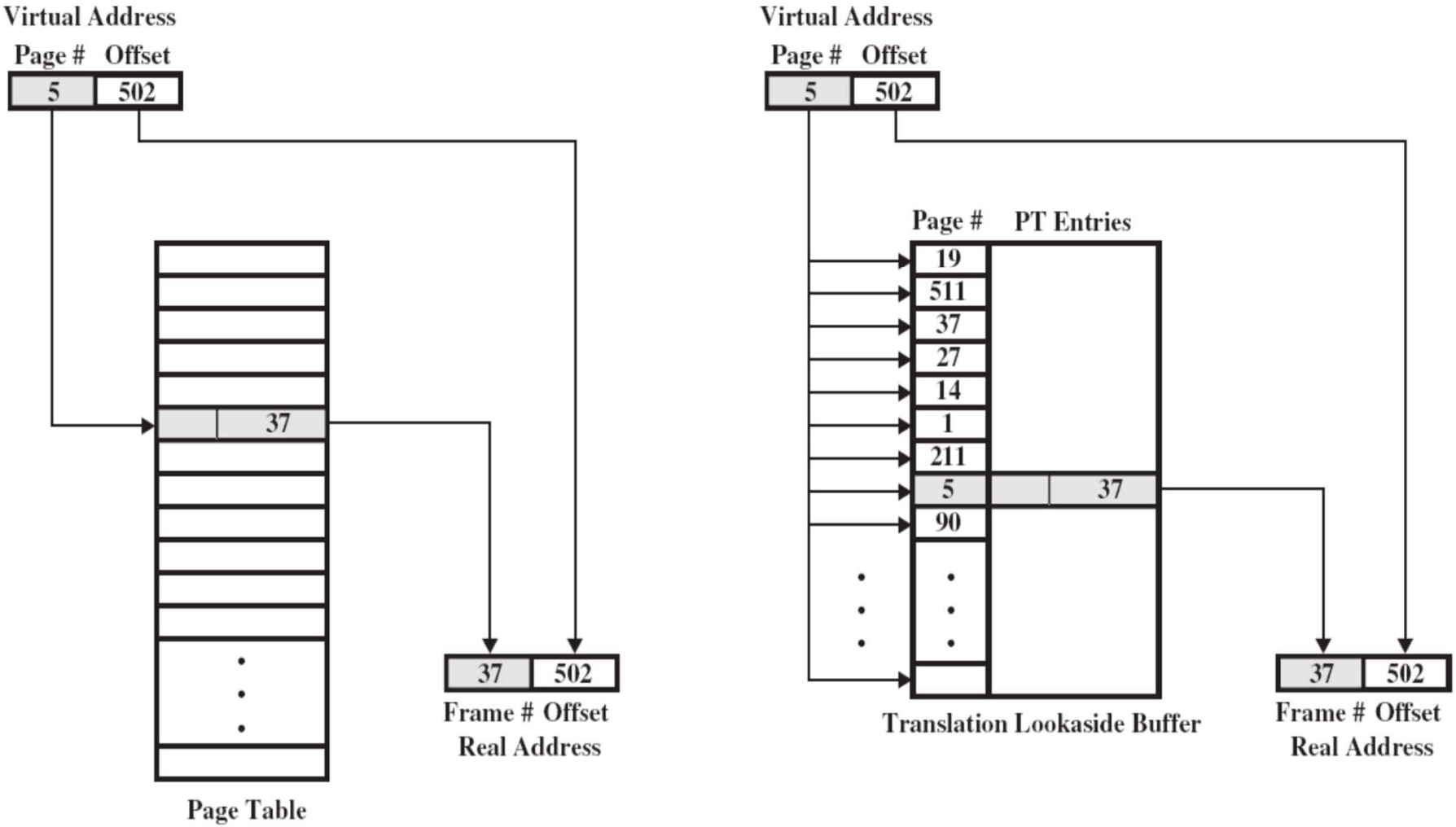


Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

Translation Lookaside Buffer (4)

- ***Some TLBs store address-space identifiers (ASIDs) in each entry of the TLB.***
  - *An ASID uniquely identifies each process and is used to provide address space protection for that process*
  - *An ASID allows the TLB to contain entries for several different processes simultaneously*
- ***If the TLB does not support separate ASIDs, every time a new page table is selected (for instance, each process switch), the TLB must be flushed (or erased)***
  - *To ensure that the next executing process does not use the wrong translation information.*

Translation Lookaside Buffer (5) Process Switch

- On a process switch, some TLB entries can become invalid.
  - The simplest strategy to deal with this is to completely flush the TLB
  - Newer CPUs have more efficient strategies:
    - In the Alpha 21264, each TLB entry is tagged with an "address space number"(ASN), and only TLB entries with an ASN matching the current task are considered valid.
    - In the Intel Pentium Pro, the page global enable (PGE) flag in the register CR4 and the global (G) fla

Page Size (1)

- **Factors to be considered in determining the page size:**
  - Internal fragmentation
  - Page table size
  - Characteristics of secondary-memory devices
  - Locality
  - Number of page faults
  - TLB performance

Page Size
I - Internal fragmentation

- ***Smaller page size, less amount of internal fragmentation***
  - ***With n segments in memory and a page size of p bytes, np/2 bytes will be wasted on internal fragmentation averagely.***

Page Size
II – Page Table Size

- For a given virtual memory space, decreasing the page size increases the number of pages and hence the size of page table

Page Size
III – Characteristics of Secondary Memory

- ***Another factor to consider is that: Secondary memory is designed to efficiently transfer large blocks of data so a large page size is better***
  - ***It might take 64×10msec to load 64 512-byte pages, but only 4×12msec to load 4 8KB pages***

Page Size
IV - Locality

- **With smaller page size, locality can be improved.**
  - A smaller page size allow each page to match program locality more accurately.
  - Total I/O should be reduced
  - Less total allocated memory

Page Size
V – Number of Page Faults

- **How about a system with page size of 1 byte?**
  - **A process of 200KB that used only half of that memory would generate 100K page faults.**
  - **With a page size of 200KB, it would generate only one page faults**
- **To minimize the number of page faults, we need to have a large page size.**

Page Size
VI - TLB performance

- ***Smaller page size, more pages required per process***
  - *More pages per process means larger page tables*
  - *Larger page tables means large portion of page tables in virtual memory →double page faults: first to bring in the needed portion of page table, and second to bring in the process page*
- ***Larger page sizes can improve TLB performance***

Segmentation (1)

- In pure paging, the virtual memory is one-dimensional because the virtual addresses go from 0 to some maximum address, one address after another.
- Segmentation allows the programmer to view memory as multiple address spaces or segments
  - Segments may be of unequal, indeed dynamic, size
- Memory references consist of a (segment number, offset) form of address.

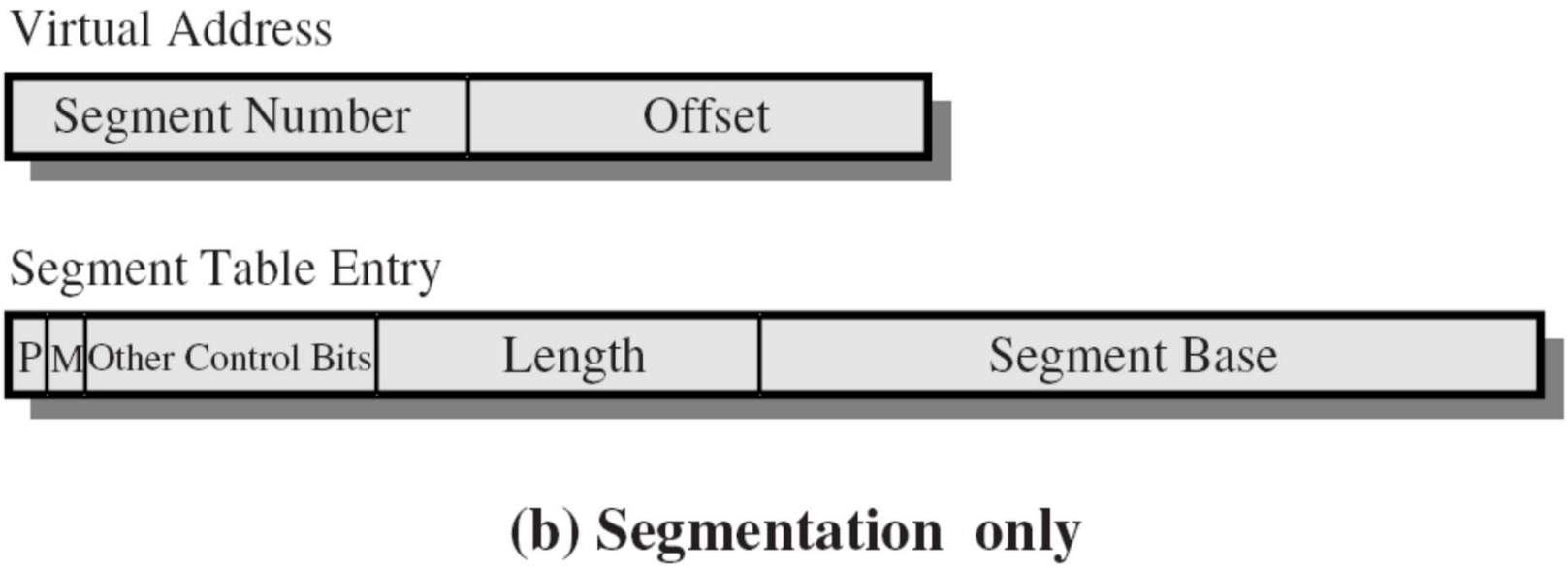
Segmentation (2)
Advantages

- **Simplifies the handling of growing data structures**
  - **If a segment needs to be expanded in main memory, the OS may move it to a larger area if available or swap it out.**
- **Allows programs to be altered and recompiled independently**
- **Lends itself to sharing data among processes**
- **Lends itself to protection**

Segmentation (3)
Segment Tables

- **When considering a virtual memory scheme base on segmentation, it is typical to associate a unique segment table with each process.**
  - Each entry contains the ***starting address*** and the ***length*** of the segment
  - A bit is needed to determine if segment ***is already in*** main memory
  - Another bit is needed to determine if the segment has been ***modified*** since it was loaded in main memory
  - Other control bits for protection and sharing

Segmentation (4)
Segment Table Entries



(b) Segmentation only

The segment table must be in main memory to be accessed

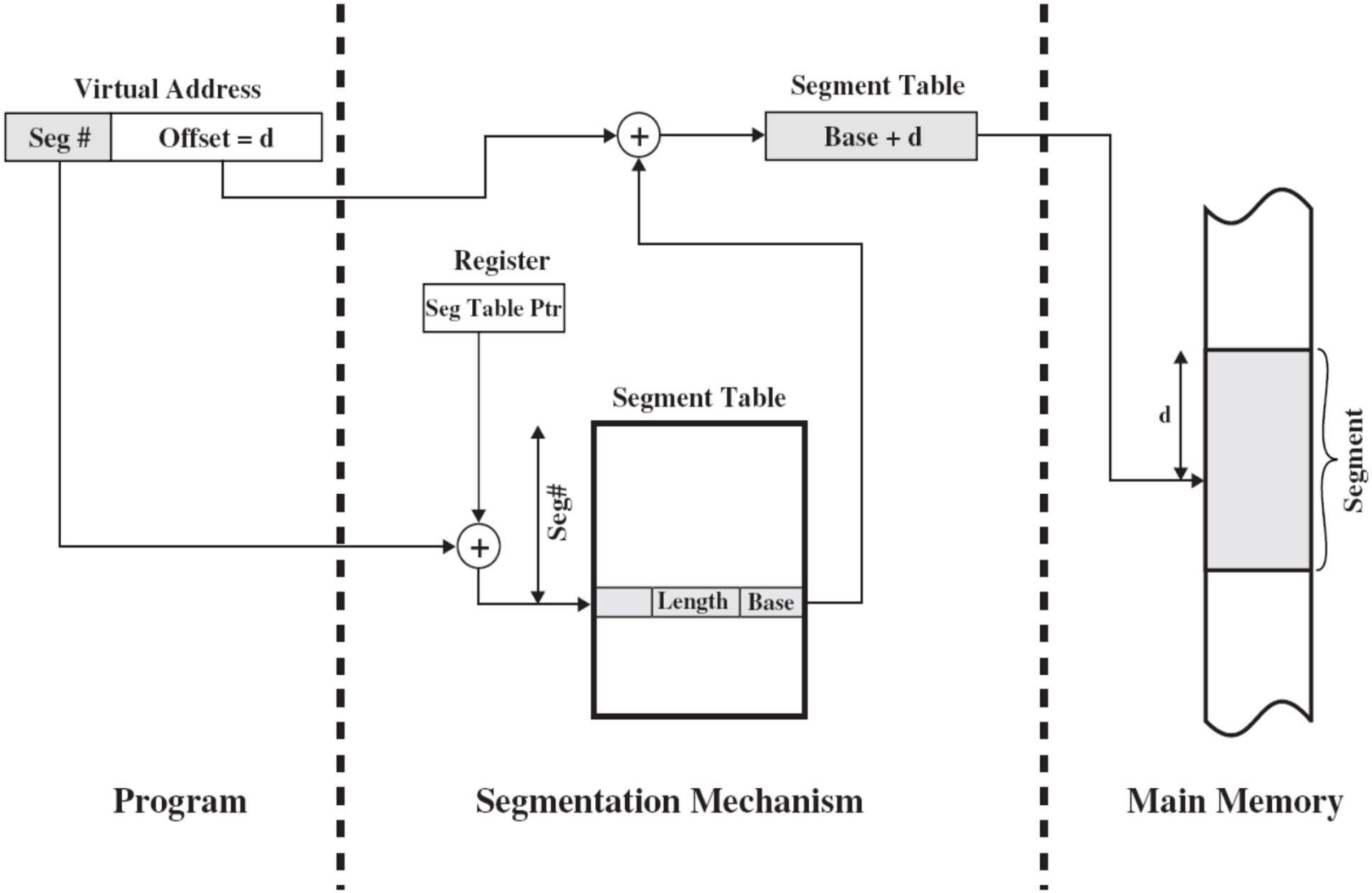


Figure 8.12 Address Translation in a Segmentation System

Paging VS Segmentation

	Paging	Segmentation
Need the programmer be aware that this technique is being used?	NO	YES
How many linear address spaces are there?	1	Many
Can procedures and data be distinguished and separately protected?	NO	YES
Can tables whose size fluctuates be accommodated easily?	NO	YES
Is sharing of procedures between users facilitated?	NO	YES
Why was this technique invented?	To get a large linear address space without buying more physical memory	To break up programs and data into logically independent address spaces and to aid sharing and protection

Combined Paging and Segmentation (1)
Paging and Segmentation

- **Both paging and segmentation have their own strengths**
  - Paging eliminates external fragmentation and thus provides efficient use of main memory
    - **Paging is transparent to the programmer**
  - Segmentation allows for growing data structures, modularity, and support for sharing and protection
    - **Segmentation is visible to the programmer**

Combined Paging and Segmentation (2) How to combine?

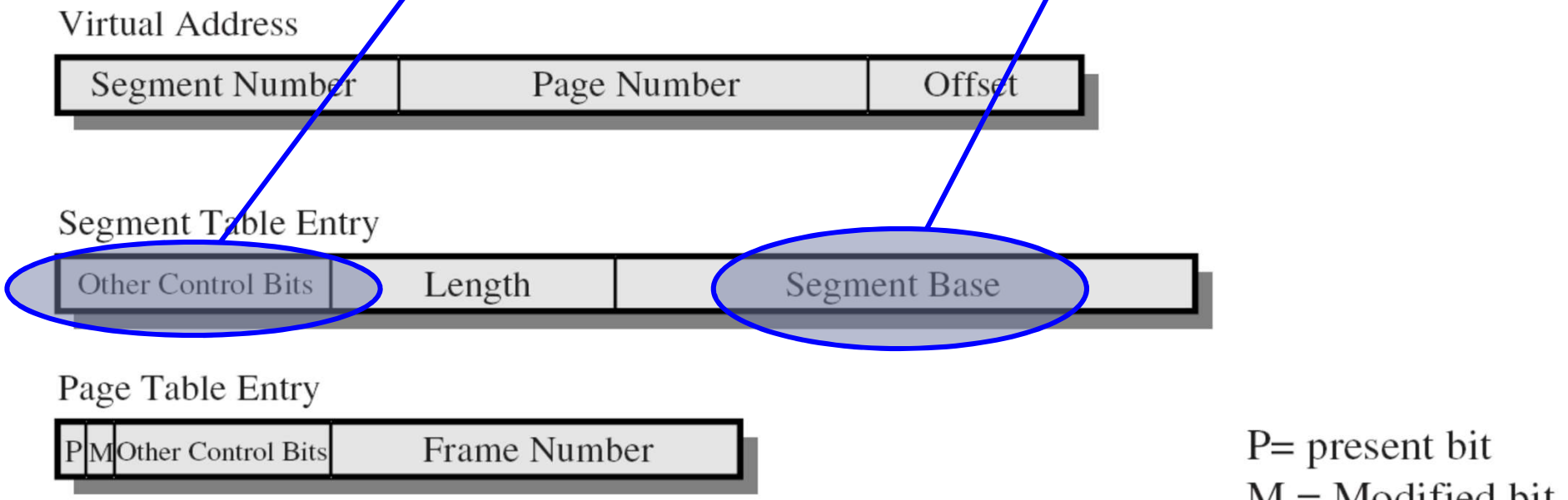
- *A user's address space is broken up into a number of segments, at the discretion of the programmer (由程序员随意决定)*
- *Each segment is, in turn, broken up into a number of fixed-size pages, which are equal in length to a main memory frame*
  - *If a segment has length less than that of a page, it occupies just one page*

Combined Paging and Segmentation (3) Logical Address

- From the programmer's point of view, a logical address still *consists of a segment number and a segment offset.*
- From the system's point of view, the *segment offset is viewed as a page number and a page offset for a page within the specified segment*

The present and modified bits are not needed, because these matters are handled at the page level
Other control bits may be used for purposes of sharing and protection

Here, the segment base refers to a page table



(c) Combined segmentation and paging

A segment table and a number of page tables is associated with each process: One page table per process segment

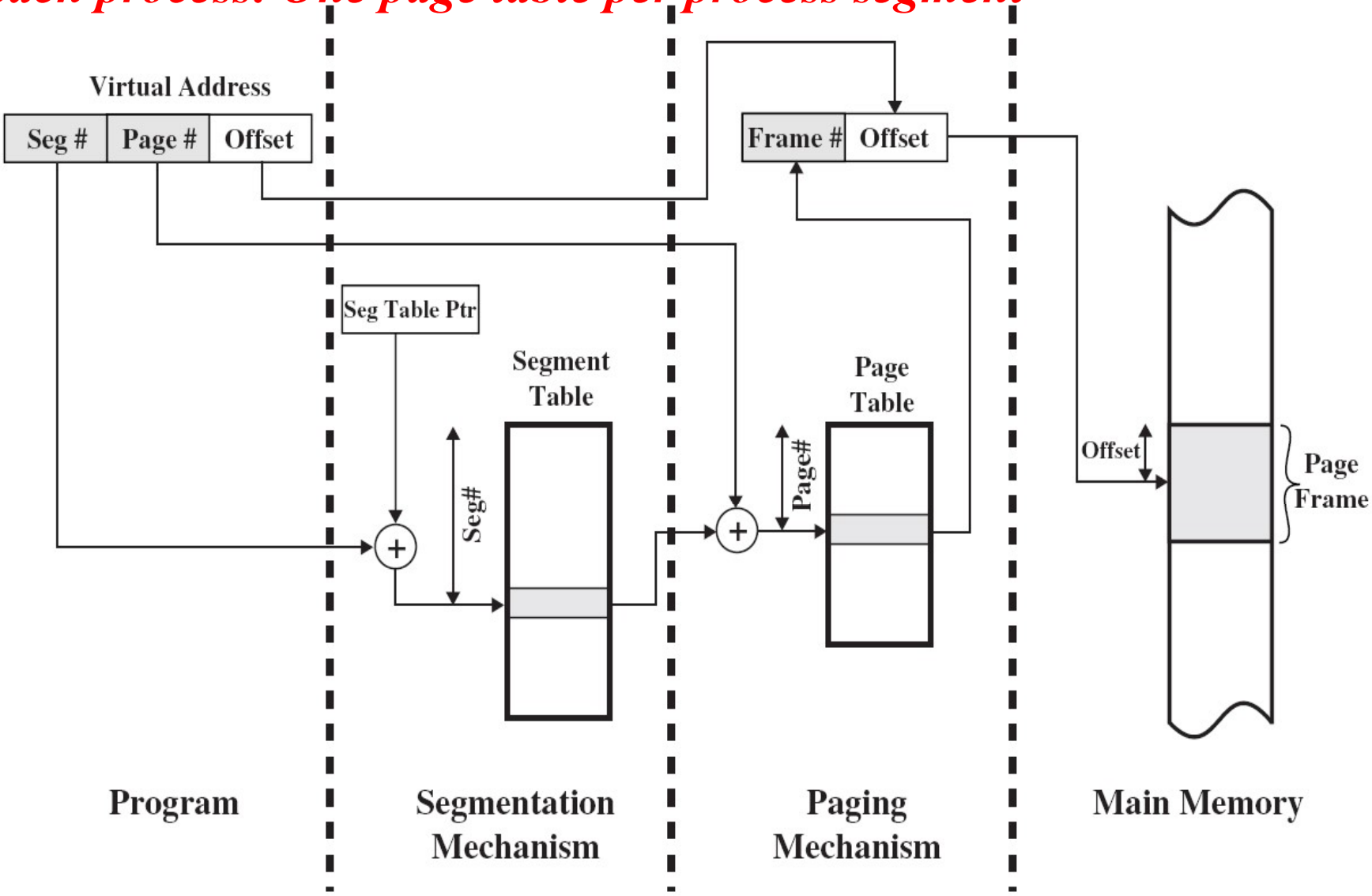
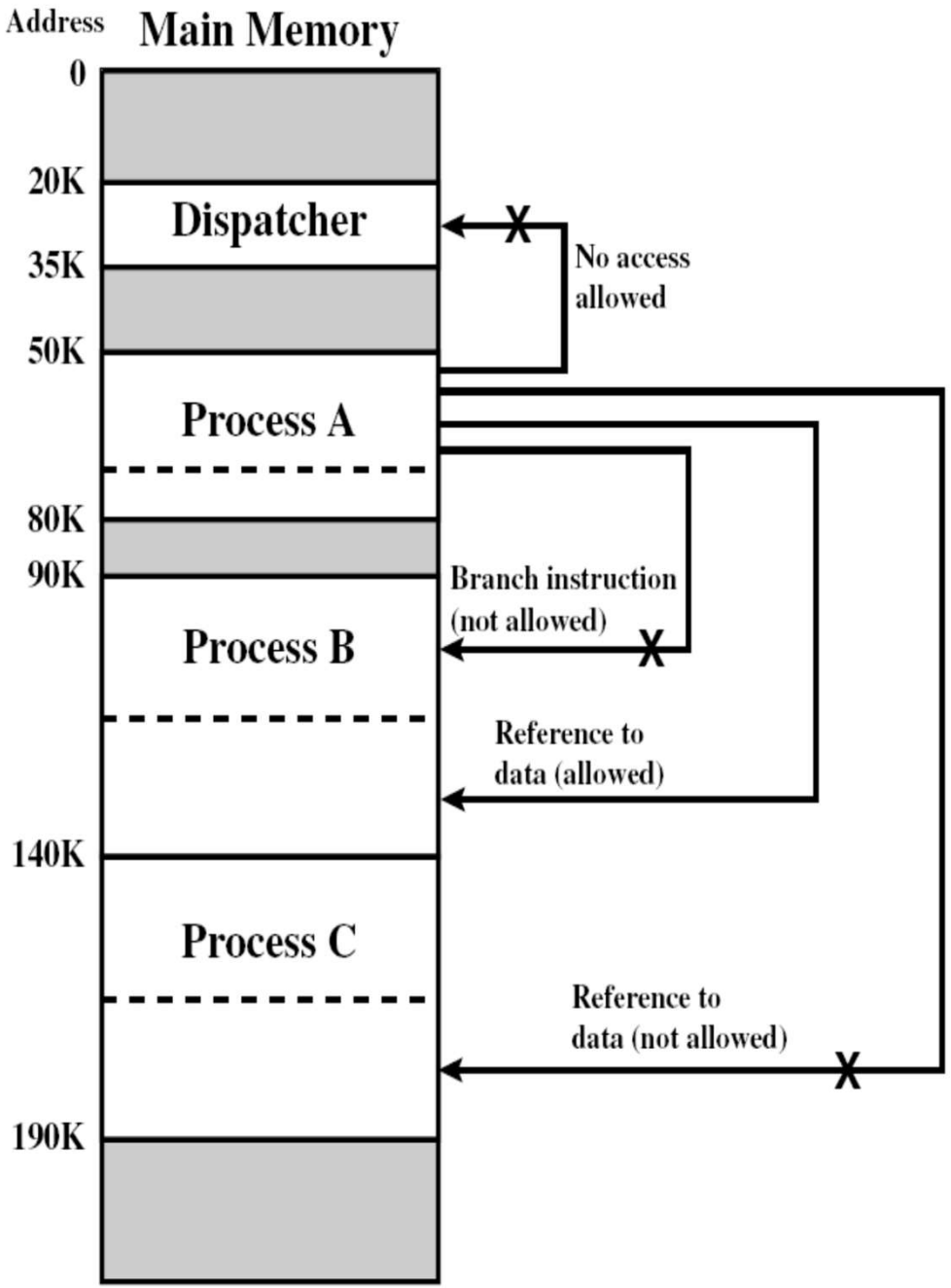


Figure 8.13 Address Translation in a Segmentation/Paging System

Protection and Sharing

- **Segmentation lends itself to (适于) the implementation of protection and sharing policies**
  - Each segment table entry includes a length as well as a base address→protection
  - It is possible for a segment to be referenced in the segment tables of more than one process→sharing
- **The same mechanisms are available in a paging system, but the specification of protection and sharing requirements is more awkward**
  - Because the page structure is invisible to programmer


Address
Main Memory
0
20K
Dispatcher
35K
50K
Process A
80K
90K
Process B
140K
Process C
190K
No access allowed
Branch instruction (not allowed)
Reference to data (allowed)
Reference to data (not allowed)

Operating System Software

Three Fundamental Choices in Designing Memory Management

- ***Whether or not to use virtual memory techniques***
- ***The use of paging or segmentation or both***
- ***The algorithms employed for various aspects of memory management***

The first two choices depends on the hardware platform available. For example, earlier UNIX implementation did not provide virtual memory because the processors on which the system ran did not support paging or segmentation.

Comments about the Fundamental Choices

- ***All important operating systems provide virtual memory*.**
  - With the exception of OSes for some of the older personal computers, such as DOS, and specialized systems
- ***Pure segmentation systems are becoming increasingly rare*.**
  When segmentation is combined with paging, most of the memory management issues confronting the OS designer are in the area of paging
  - Thus, we can concentrate on the issues associated with paging.
- **The choice related to the third fundament choice are the domain of OS software and are subject of this section.**

Operating System Policies for Virtual Memory

- **Key design elements for virtual memory**
  - *Fetch policy* (读取策略): Demand or Prepaging?
  - *Placement policy* (放置策略)
  - *Replacement policy* (替换/置换策略)
  - *Resident set management* (驻留集管理)
  - *Cleaning policy* (清除策略)
  - *Load control* (负载控制): Determining the number of processes that will be resident in main memory
- **Goal: *to minimize the rate at which page faults occur.***

Overhead Caused by Page Faults

- *Deciding which resident page or pages to replace*
- *The I/O of exchanging pages*
- *Process switch: the OS must schedule another process to run during the page I/O, causing a process switch.*

Fetch Policy (1)
读取策略

- Fetch policy determines when a page should be brought into main memory
- Two common alternatives
  - Demand paging (请求式调页): only bring a page into main memory when a reference is made to a location on the page
  - Prepaging (预调页): brings in more pages than needed

Fetch Policy (2)

- ***Prepaging exploits the characteristics of most secondary memory devices*, which have *seek times* (寻道时间) and *rotational latency* (旋转延迟).**
  - **If the pages of a process are stored contiguously in secondary memory, it is *more efficient to bring them in at one time* than bringing one at a time over a period**
  - **This policy is ineffective if most of the extra pages that are brought in are not referenced**

Placement Policy (1)
放置策略

- **The placement policy determines *where* *in real memory a process piece is to reside***
  - **In a *pure segmentation system*, the placement policy is an important design issue**
    - ***Best-fit*, *first-fit*, *next-fit*, and so on.**
  - **For a *system using pure paging or paging combined with segmentation*, placement is usually irrelevant**
    - ***Any page-frame combination has equal efficiency***

Placement Policy (2)
NUMA Multiprocessor

- **Non-Uniform Memory Access (NUMA) Multiprocessor (非一致存储访问多处理器)**
  - *The time for accessing a particular physical location varies with the distance between the processor and the memory module*
- **For NUMA, an automatic placement strategy is desired to *assign pages to the memory module that provides the best performance***

Replacement Policy (1)
替換/置換策略

- **Replacement policy deals with *the selection of a page in memory to be replaced* when a new page must be brought in**
  - **All the frames in main memory are occupied at that time**
- **All replacement policies have the objective:**
  - ***Page removed should be the page least likely to be referenced in the near future***
  - ***Most policies predict the future behavior on the basis of past behavior***

Replacement Policy (2)
Several Interrelated Concepts

- **This topic is difficult to explain because it involves several interrelated 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**

Resident Set Management

Replacement Policy (3)
Frame Locking (帧锁定)

- Some of the frames in main memory may be locked (锁定)
  - If frame is locked, it may not be replaced
- Which ones should be locked?
  - Much of the kernel of the operating system
  - Key control structures
  - I/O buffers
- Locking is achieved by associating a lock bit with each frame

Replacement Policy (4)
Basic Replacement Algorithms

- **Regardless of the resident set management strategy, there are certain basic algorithms for selecting a page to replace**
  - **Optimal policy (最优策略OPT)**
  - **Least recently used (最近最少使用/最久未使用LRU) policy**
  - **First-in-first-out (先进先出FIFO) policy**
  - **Clock policy (时钟策略)**

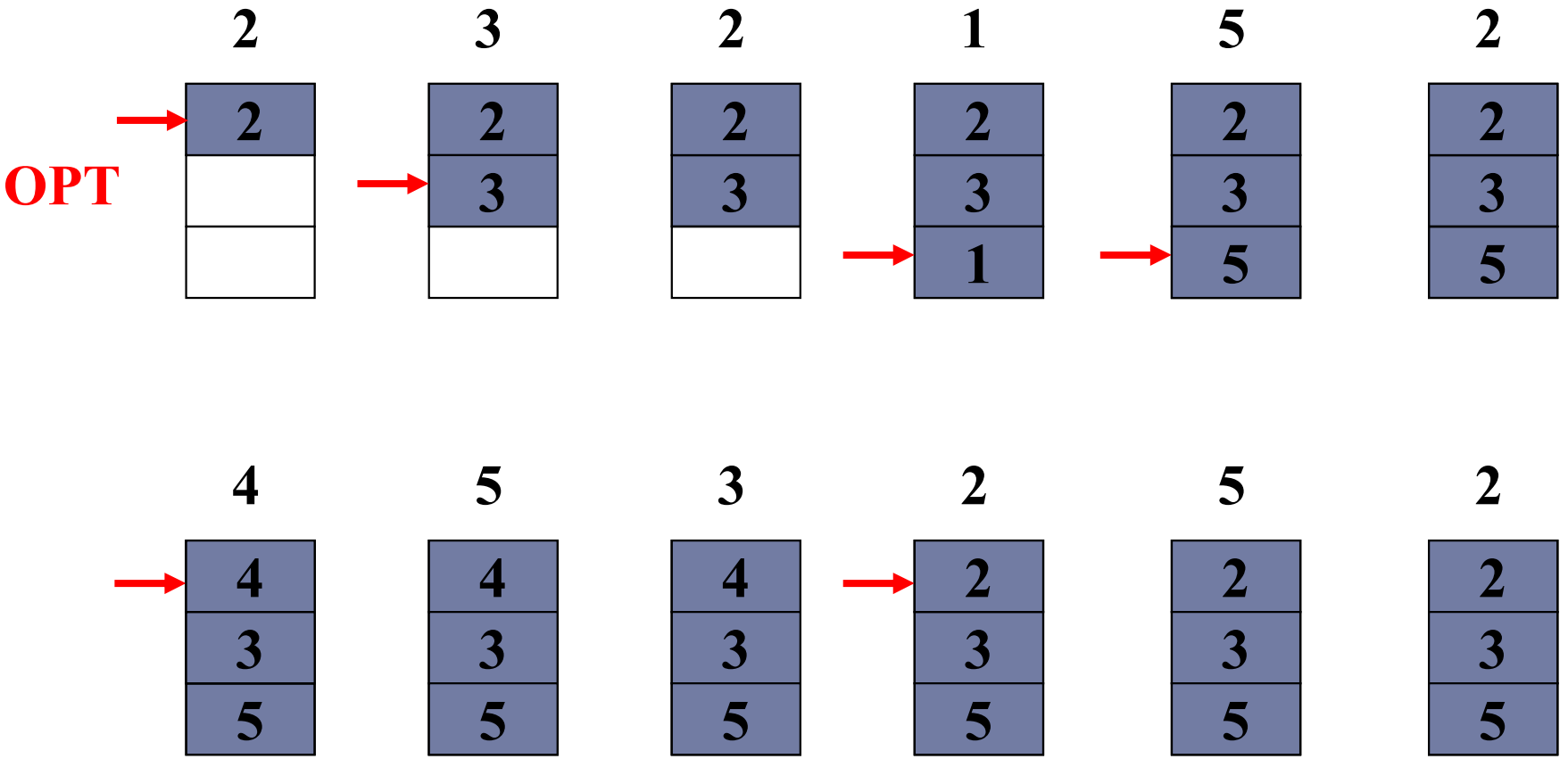
Replacement Policy (5) Algorithm -- Optimal Policy

- **Optimal policy selects for replacement that page for which *the time to the next reference is the longest***
  - **It is *the best algorithm*, because it results in the fewest number of page faults**
  - **Problem: It is *impossible to implement*, because it needs have perfect knowledge of future events**

This example assumes a fixed frame allocation (fixed resident set size) for this process of three frames

The page reference stream is: **2 3 2 1 5 2 4 5 3 2 5 2**

(页访问流)

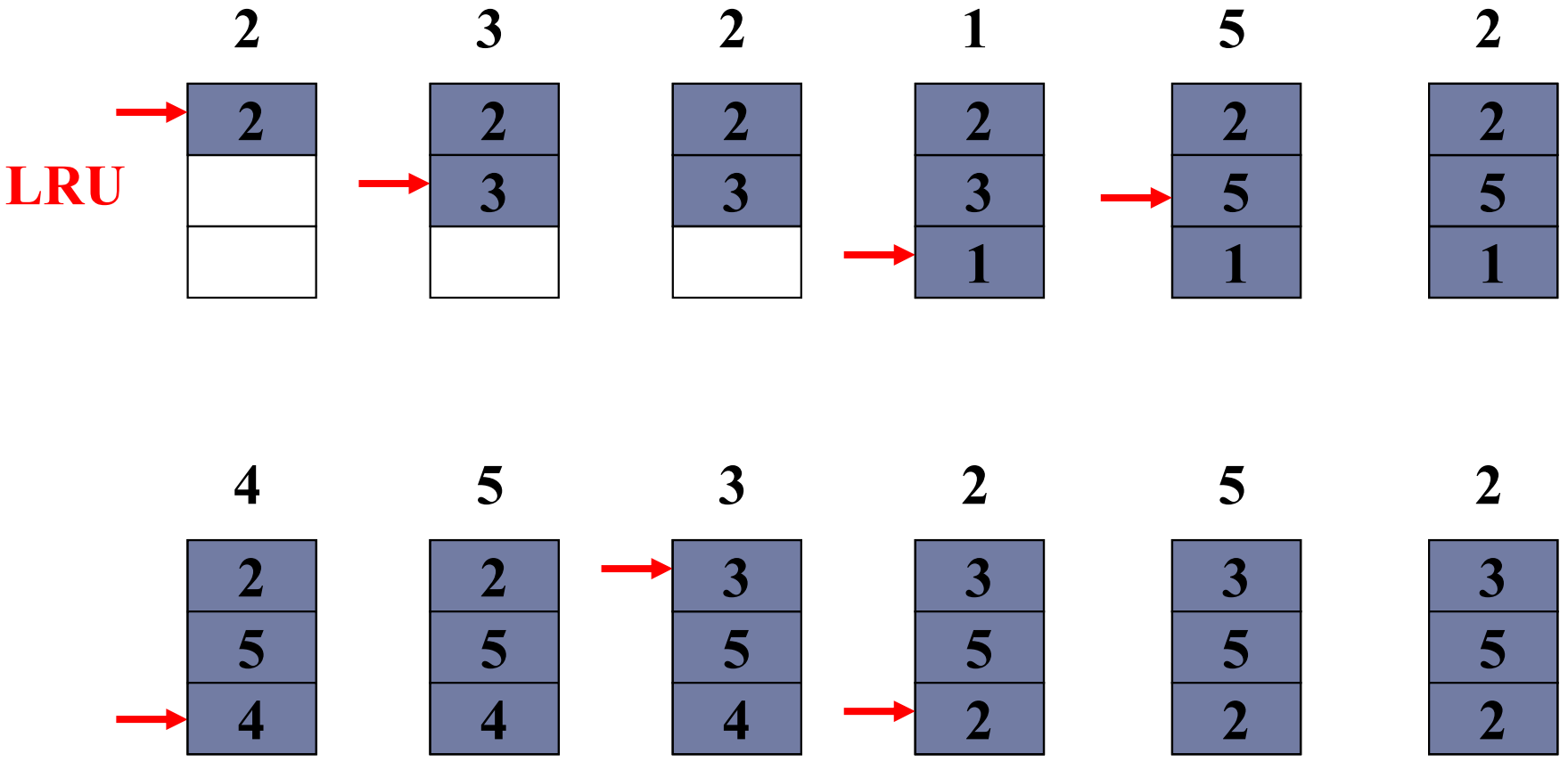


Replacement Policy (6) Algorithm -- Least Recently Use

- **Least recently used (LRU) policy replaces the page that *has not been referenced for the longest time***
  - ***By the principle of locality, this should be the page least likely to be referenced in the near future***
- **Problem: Difficulty in implementation.**
  - **Each page could be tagged with *the time of last reference*. This would require a great deal of overhead.**

This example assumes a fixed frame allocation (fixed resident set size) for this process of three frames

The page address stream is: **2 3 2 1 5 2 4 5 3 2 5 2**



Least Recently Use Implementation by Linked List

- ***Linked List: to keep a linked list of page numbers.***
  - ***When a page is referenced, it is removed from the linked list and put on the head of the linked list.***
  - ***In this way, the head of the linked list is always the most recently used page, while the tail is the LRU page***

Least Recently Use Implementation by Counters

- ***Counters: each page-table entry is associated with a time-of-use field, and a logical clock or counter is added to the CPU***
  - *The clock is incremented for every memory reference*
  - *When a page if referenced, the contents of the clock is copied to the time-of-use filed in the PTE for that page*

Least Recently Use Implementation by Matrix

- ***Matrix: for a machine with n page frames, the LRU hardware maintain a matrix of n×n bits (initially all 0)***
  - ***When page frame k is referenced, the hardware sets all the bits of row k to 1, then sets all the bits of column k to 0.***
  - ***At any instance, the row whose binary value is lowest is the least recently used.***

	0	1	2	3
0	0	1	1	1
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

	0	1	2	3
0	0	0	1	1
1	1	0	1	1
2	0	0	0	0
3	0	0	0	0

	0	1	2	3
0	0	0	0	1
1	1	0	0	1
2	1	1	0	1
3	0	0	0	0

	0	1	2	3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	1
3	1	1	1	0

	0	1	2	3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	1
3	1	1	0	0

	0	1	2	3
0	0	0	0	0
1	1	0	1	1
2	1	0	0	1
3	1	0	0	0

	0	1	2	3
0	0	1	1	1
1	0	0	1	1
2	0	0	0	1
3	0	0	0	0

	0	1	2	3
0	0	1	1	0
1	0	0	1	0
2	0	0	0	0
3	1	1	1	0

	0	1	2	3
0	0	1	0	0
1	0	0	0	0
2	1	1	0	1
3	1	1	0	0

	0	1	2	3
0	0	1	0	0
1	0	0	0	0
2	1	1	0	0
3	1	1	1	0

LRU using a matrix when pages are referenced in the order 0, 1, 2, 3, 2, 1, 0, 3, 2, 3.

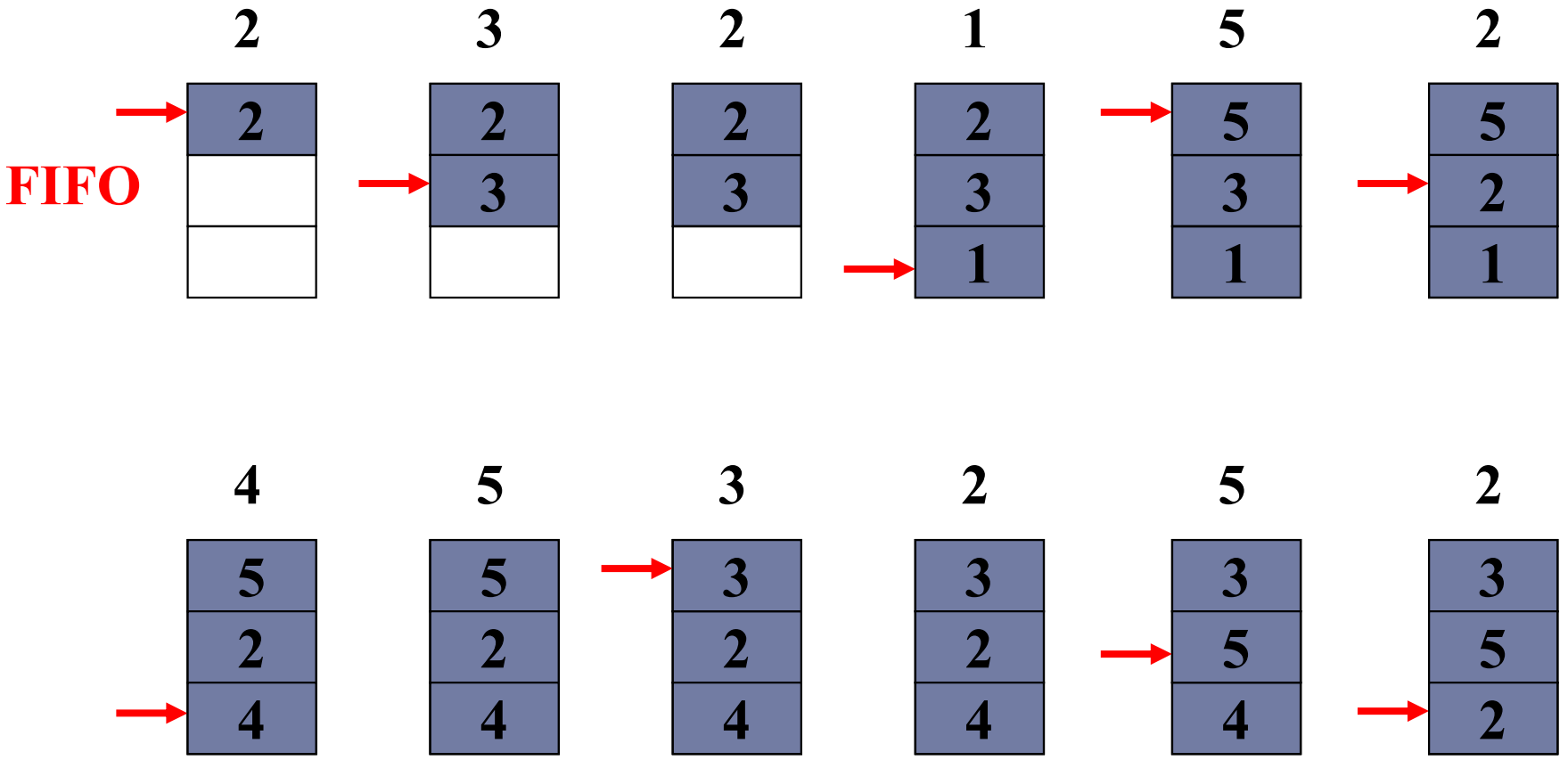
Replacement Policy (8) Algorithm -- First-In-First-Out

- **First-in, first-out (FIFO) policy treats page frames allocated to a process as a circular buffer, pages are removed in round-robin style**
  - *Page that has been in memory the longest is replaced*
  - It is the simplest replacement policy to implement
    - A pointer that circles through the page frames of the process
  - Problem:These pages may be needed again very soon
    - There will often be regions of program or data that are heavily used throughout the life of a program.Those pages will be repeatedly paged in and out by the FIFO algorithm

FIFO is best suited for programs that reference segments in sequence

This example assumes a fixed frame allocation (fixed resident set size) for this process of three frames

The page address stream is: **2 3 2 1 5 2 4 5 3 2 5 2**



Belady's Anomaly (1)
What is it?

- ***What is Belady's anomaly (Belady 异态)?***
  - ***The paging algorithm has worse performance when the amount of primary memory allocated to the process is increased.***
- ***Which class of replacement algorithms is susceptible to (易受…影响) Belady's anomaly?***

Belady's Anomaly (2)
An Example

- Consider the page reference stream:
  0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4
  as it is processed by FIFO with three frames allocated
- How about four frames allocated?
- Please calculate the numbers of page faults in these two situations!

Belady's Anomaly (3)
Stack Algorithms

- ***What are stack algorithms?***
  - *The set of replacing algorithms in which the set of pages loaded with an allocation of m frames is always a subset of the set of pages that has a frame allocation of m+1*
  - *This property is called the inclusion property.*
- ***Algorithms that satisfy the inclusion property are not subject to Belady's anomaly.***

Replacement Policy
The Use Bit

- *A number of other algorithms are tried to approximate the performance of LRU while imposing little overhead*
- *Most such algorithms make use of an additional bit (called a use bit) associated with each frame*
  - *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*

Additional-Reference-Bits Algorithm

- *Keep an 8-bit byte for each page in a table in memory*
- *At regular intervals, a timer interrupt transfers control to the OS*
- *OS shifts the reference bit for each page into the high-order bit of its 8-bit byte, shifting the other bits right by 1 bit and discarding the low-order bit.*

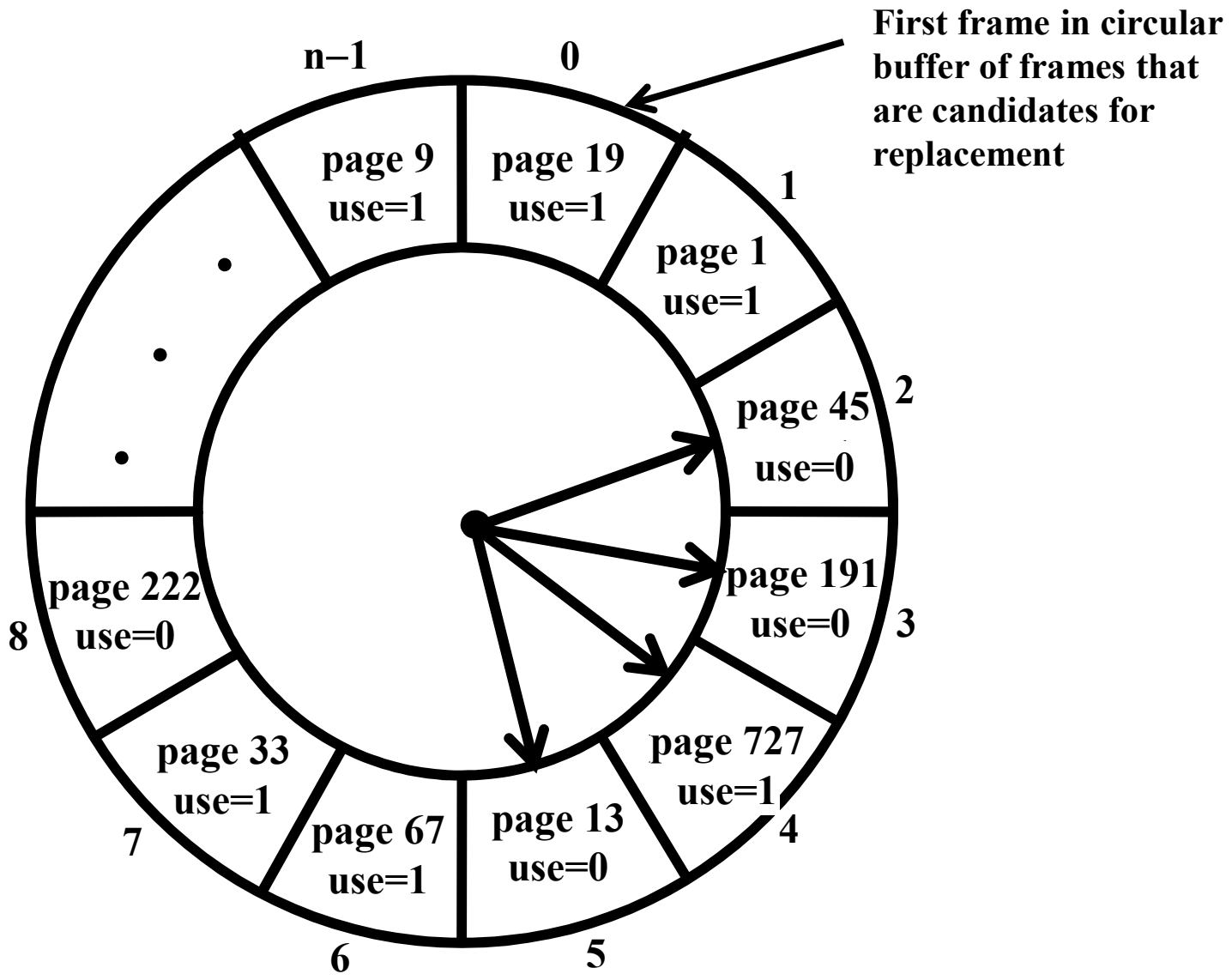
The page with the lowest number is the LRU page

Second Chance Algorithm

- **Second change algorithm is a simple modification to FIFO**
  - **It inspect the use bit of the oldest page**
    - **If it is 0, the page is both old and unused, so it is replaced immediately**
    - **If the use bit is 1, the bit is cleared, the page is put onto the end of the list of pages, and its load time is updated as though it had just arrived in memory.**
- **It can avoids the problem of throwing out a heavily used page**

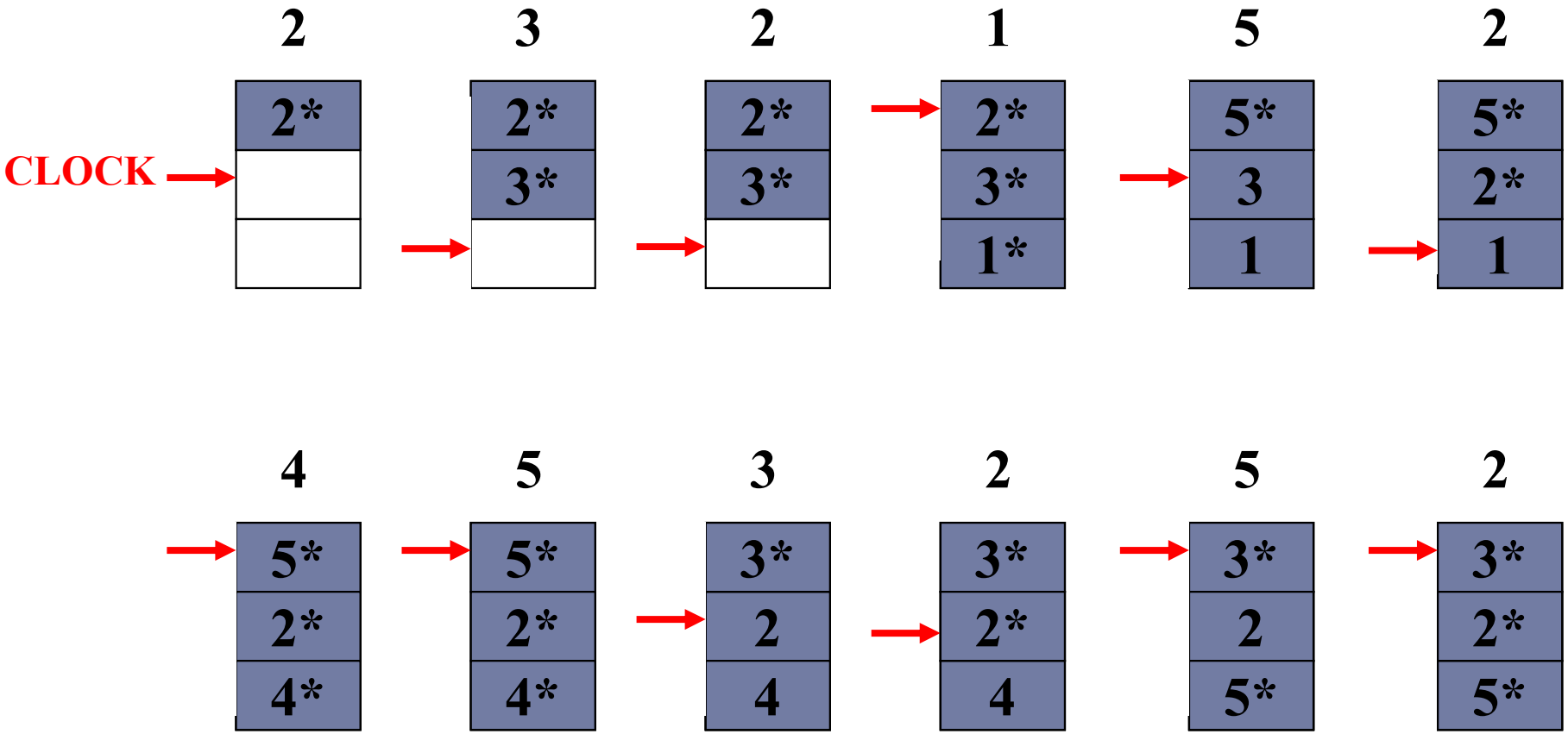
Replacement Policy (10) Algorithm -- Clock Policy II

- **Clock algorithm is one way to implement the second-chance policy.**
  - It keeps all the candidate frames on a **circular buffer** (in the form of a clock), associated with a pointer.
  - When it is time to replace a page, the OS scan the buffer to find a frame with a use bit set to zero.
    - **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


First frame in circular buffer of frames that are candidates for replacement
n−1
0
1
2
3
4
5
6
7
8
page 9
use=1
page 19
use=1
page 1
use=1
page 45
use=0
page 191
use=0
page 727
use=1
page 13
use=0
page 67
use=1
page 33
use=1
page 222
use=0

This example assumes a fixed frame allocation (fixed resident set size) for this process of three frames

The page address stream is: **2 3 2 1 5 2 4 5 3 2 5 2**



Replacement Policy (11) Algorithm -- Clock Policy III

- **The clock algorithm is similar to FIFO**
  - **Except that any frame with a use bit of 1 is passed over by the clock algorithm.**
  - **It avoids the problem of throwing out a heavily used page**

An Enhanced Clock Policy (1)

- The clock algorithm can be enhanced by increasing the number of bits that it employs

An Enhanced Clock Policy (2)
Categories of Frames

- Each frame falls into one of four categories
  - Not accessed recently, not modified (u=0, m=0)
  - Accessed recently, not modified (u=1, m=0)
  - Not accessed recently, modified (u=0, m=1)
  - Accessed recently, modified (u=1, m=1)

Why use this adverb?

? What does it mean?

An Enhanced Clock Policy (3)
The Algorithm

- **Step 1: Scan the frame buffer from the current position. The first frame encountered with (u=0, m=0) is selected for replacement**
  - **During this scan, make no change to the use bit.**
- **Step 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 passed**
- **Step 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**

An Enhanced Clock Policy (4) Advantage

- **The advantage of this algorithm over the simple clock algorithm is that:**
  - **The pages that are unchanged are given preference for replacement**
    - **Because the modified page must be written out before being replaced.**

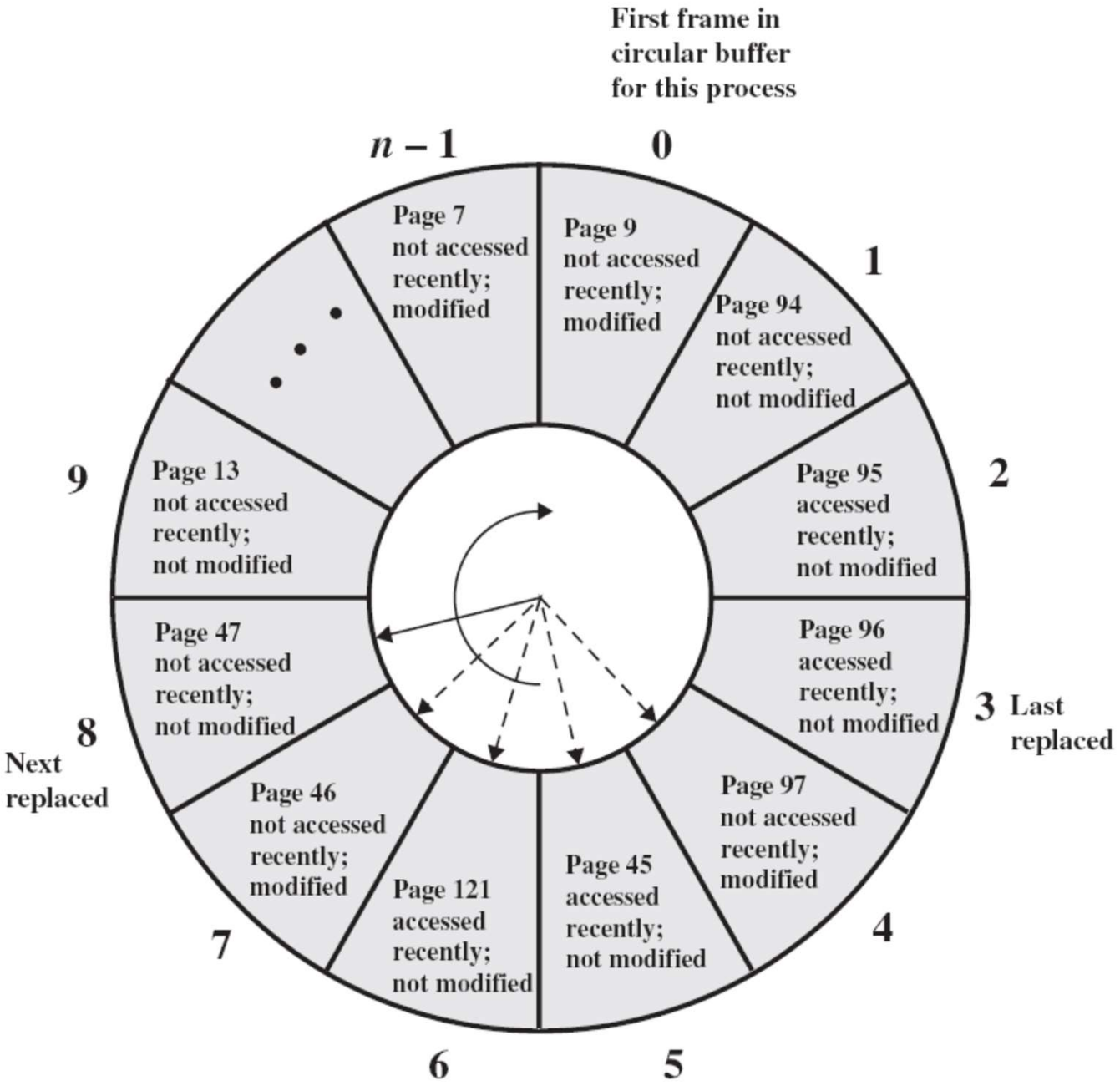


Figure 8.18 The Clock Page-Replacement Algorithm [GOLD89]

Counting-Based Policies

- *LFU – Least Frequently Used*
  - *The page with the smallest count be replaced.*
  - *A large reference count indicates an actively-used page*
  - *However, some pages were heavily used in history and will not be need in future any more.*
  - *Remedy: shift the counts right by 1 bit at regular intervals, decaying average usage count*
- *MFU – Most Frequently Used*
  - *Page with the smallest count was probably just brought in and has yet to be used*

Which One is the Best?

- It is easy to find examples of programs for which any given policy is better than the others
  - In [Denning, P. 1980] IEEE Transactions on Software Engineering

Page Buffering (1)
A Simple Approach

- ***The cost of replacing a page that has been modified is greater than for one that has not.***
  - *How to deal with it? – Page buffering*
- ***System keeps a pool of free frames. When a page fault occurs:***
  - *A victim page is chosen as before. However, the desired page is read into a free frame from the pool before the victim is written out.*
- ***This procedure allows the process to restart as soon as possible, without waiting for the victim page to be written out.***
  - *When the victim is later written out, its frame is added to the free-frame pool*

Page Buffering (2)
The VAX VMS Approach - I

- *The page replacement algorithm is simple FIFO*
  - *A replaced page is not lost but rather is assigned to one of two lists:*
    - *The free page list if the page has not been modified,*
    - *Or the modified page list if the page has been modified*
  - *NOTE: the page is not physically removed!!*
    - *Instead, only the entry in the page table for this page is removed.*

Page Buffering (3)
The VAX VMS Approach - II

- **What happens when a page is to be read in?**
  - **The free page list is a list of page frames available for reading in pages.**
  - **VMS tries to keep some small number of frames free at all times.**
  - **When a page is to be read in, the page frame at the head of the list is used, destroying the page that was there.**

Page Buffering (4)
The VAX VMS Approach - III

- **What happens when a page is to be replaced?**
  - When an unmodified page is to be replaced, it remains in memory and its page frame is added to the tail of the free page list
  - When a modified page is to be written out and replaced, its page frame is added to the tail of the modified page list

Page Buffering (5)
The VAX VMS Approach - IV Advantages

- **The page to be replaced remains in memory**
  - If the process references that page, it is returned to the resident set of that process at little cost.
- **Modified pages are written out in clusters rather than one at a time.**
  - This reduces the amount of disk access time
- **Modified pages can be written out when the paging device is idle.**

Resident Set Management (1)
驻留集管理

- ***The operating system must decide how many pages of a particular process to bring in.***
- ***Several factors in mind***
  - *The smaller the amount of memory allocated to a process, the more processes that can reside in main memory at any one time*
  - *If a relatively small number of pages of a process are in main memory, the rate of page faults will be rather high*
  - *Beyond a certain size, additional allocation of main memory to a particular process will have no noticeable effect on the page fault rate*

Resident Set Management (2)
Resident Set Size

- **Fixed-allocation (固定分配策略)**
  - gives a process a fixed number of page frames within which to execute
  - when a page fault occurs, one of the pages of that process must be replaced
- **Variable-allocation (可变分配策略)**
  - number of pages allocated to a process varies over the lifetime of the process

Resident Set Management (3) Replacement Scope (替换范围)

- A local replacement scope (局部替换范围)chooses only among the resident pages of the process
  - A fixed resident set implies a local replacement policy
- A global replacement scope (全局替换范围) considers all unlocked pages in main memory as candidates for replacement
  - A variable-allocation policy can clearly employ a global replacement policy
- Variable allocation and local replacement is also a valid combination

Resident Set Management (4)
Fixed Allocation, Local Scope

- **A process is running in main memory with a fixed number of frames**
  - When a page fault occurs, a resident page of this process is selected to be replaced
- **It is necessary to decide ahead of time the amount of allocation to give to a process (based on the application type and the requested amount)**
  - If allocation is too small →a high page fault
  - If allocation is unnecessarily large→too few processes in main memory→considerable processor idle or considerable time spent in swapping

Resident Set Management (5)
Variable Allocation, Global Scope

- It is perhaps the easiest to implement
  - It is 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, chooses a page currently in main memory to replace
    - The replaced page can belong to any of the resident processes
    - The process that suffers the reduction in resident set size may not be optimum

Resident Set Management (6)
Variable Allocation, Local Scope I

- 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
- From time to time, reevaluate allocation, and increase or decrease it to improve overall performance

Resident Set Management (7)
Variable Allocation, Local Scope II

- **How to make the decision to increase or decrease a resident set size?**

Working Set Strategy (1)

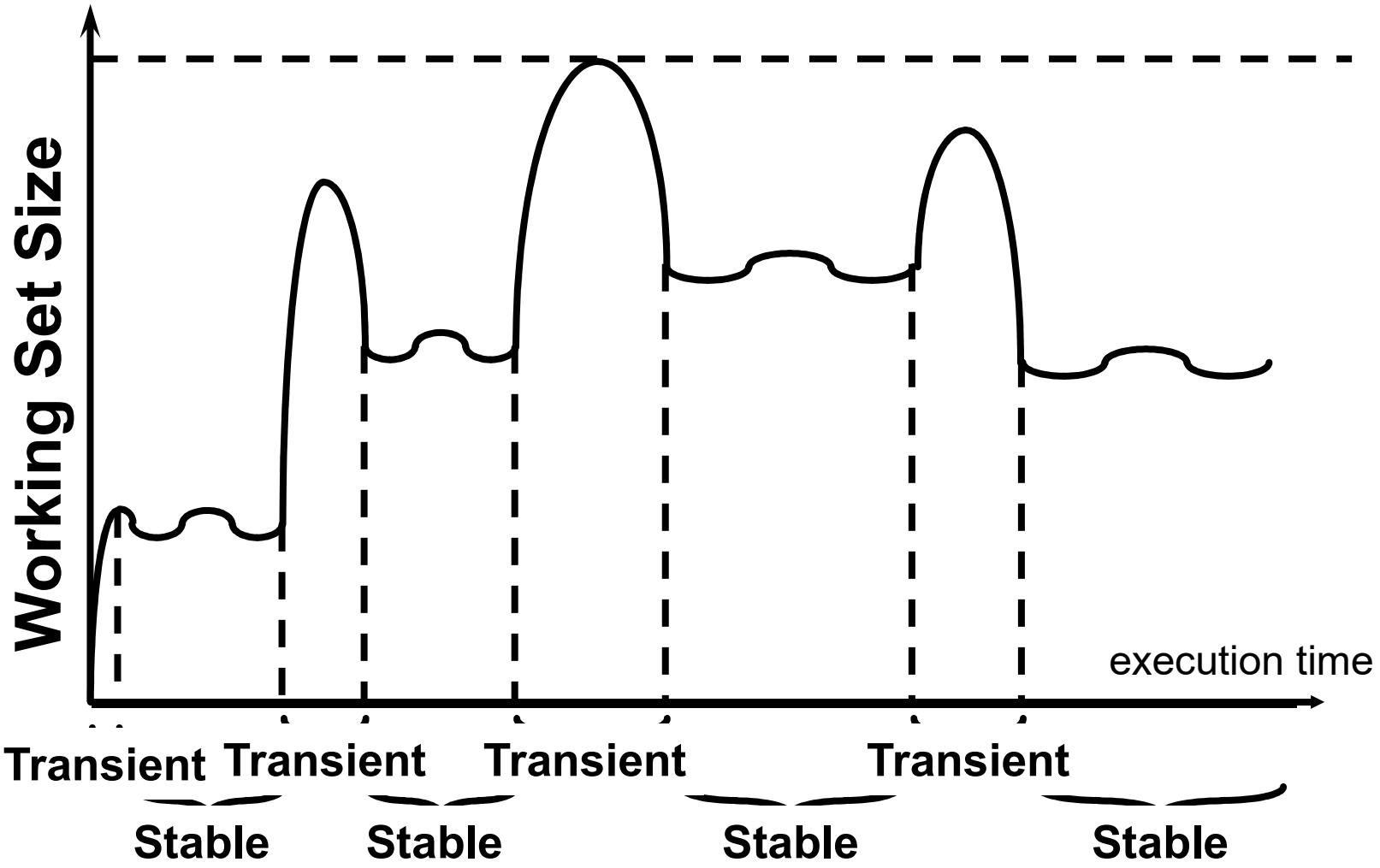
- **Working Set is a concept introduced and popularized by Denning, P.,**
  - **It has a profound impact on virtual memory design**
- **What is the working set?**
  - **At any instant of time *t*, the set of all the pages used by the Δ most recent memory references is call the working set $w(\Delta, t)$.**

It is hard to implement:
(1)the system has to maintain a shift register of length Δ;
(2)at each memory reference, the register is shifted left one position and the most recently referenced page number is inserted on the right;
(3)At a page fault, read out the contents of the register, and remove the duplicate pages.

Working Set Strategy (2)
Working Set Window

- **The variable Δ defines the working set window, over which the process is observed**
  - **The working set size will be a nondecreasing function of the window size**
  - **The larger the window size, the larger the working set**

Sequence of Page References	Window Size, Δ			
	2	3	4	5
24	24	24	24	24
15	24 15	24 15	24 15	24 15
18	15 18	24 15 18	24 15 18	24 15 18
23	18 23	15 18 23	24 15 18 23	24 15 18 23
24	23 24	18 23 24	•	•
17	24 17	23 24 17	18 23 24 17	15 18 23 24 17
18	17 18	24 17 18	•	18 23 24 17
24	18 24	•	24 17 18	•
18	•	18 24	•	24 17 18
17	18 17	24 18 17	•	•
17	17	18 17	•	•
15	17 15	17 15	18 17 15	24 18 17 15
24	15 24	17 15 24	17 15 24	•
17	24 17	•	•	17 15 24
24	•	24 17	•	•
18	24 18	17 24 18	17 24 18	15 17 24 18



The working set size can vary over time for a fixed value of Δ

The transient periods reflect a shift of the process to a new locality. During the transition phase, some of the pages from the old locality remain within the window.

Working Set Strategy (3)
Working Set Strategy for Resident Set Size

- **The concept of working set can be used to guide a strategy for resident set size:**
  - Monitor the working set of each process
  - Periodically remove from the resident set of a process those pages that are not in its working set
  - ***A process may execute only if its working set is in main memory*** **(i.e. if its resident set includes its working set)**

Working Set Strategy (4)
Unfortunate Problems

- **The past does not always predict the future.**
  - Both the size and the membership of the working set will change over time
- **A true measurement of working set for each process is impractical**
  - It would be necessary to time-stamp every page reference for every process using the virtual time of the process and then maintain a time-ordered queue of pages
- **The optimal value of Δ is unknown and in any case would vary.**

Working Set Strategy (5)
An Approximation

- **One approximation is to use execution time instead of counting back Δ memory references**
  - **The working set with parameter Δ for a process at virtual time *t* is the set of pages of that process that have been referenced in the last Δ virtual time units**
  - **Virtual time means the time that elapses while the process is actually in execution**
    - **Virtual time may be measured in instruction cycles, with each executed instruction equaling one time unit**

Page Fault Frequency Algorithm (1)
缺页频率PFF – Basic Idea

- **Page fault frequency is an important parameter for the memory allocation decision process**
  - **A higher page fault frequency indicates that a process is running inefficiently because it is short of page frames**
    - **The resident set for the process should be increased**
  - **A low page fault frequency indicates that increasing the number of allocated frames will not considerably increase efficiency**
    - **It might result in waste of main memory**
    - **One or more frames could be freed to improve system performance**

Page Fault Frequency Algorithm (2) Details

- **The PFF algorithm requires a use bit to be associated with each page in memory**
  - This bit is set to 1 when the page is accessed
- **A threshold *F* is defined**
- **When a page fault occurs, the OS notes the virtual time since the last page fault for that process**
  - This could be done by maintaining a counter of page references
  - If the virtual time is less than *F*, then a page is added to the resident set of the process.
  - Otherwise, discard all pages with a use bit of zero, and shrink the resident set accordingly. At the same time, reset the use bit on the remaining pages of the process to 0.

Page Fault Frequency Algorithm (3) Time and Frequency

- **The time between page faults is the reciprocal of the page fault rate/frequency**
- **The use of simple time measurement is a reasonable compromise that allows decisions about resident set size to be based on the page fault rate.**
  - **Although it seems to be better to maintain a running average of the page fault rate**

Page Fault Frequency Algorithm (4) A Major Flaw

- **PFF does not perform well during the transient periods when there is a shift to a new locality.**
  - During interlocality transitions (当进程在局部性之间进行过渡时), the rapid succession of page faults causes the resident set of a process to swell before the pages of the old locality are expelled
  - With PFF, no page ever drops out of the resident set before *F* virtual time units (F is the threshold)
  - the sudden peaks of memory demand may produce unnecessary process deactivations and reactivations, with the corresponding undesirable switching and swapping overheads

Variable-Interval Sampled Working Set (1)
采样间隔可变的工作集(VSWS)

- **VSWS attempts to deal with the phenomenon of interlocality transition (局部性过渡)**
- **The VSWS evaluates the working set of a process at sampling instances based on elapsed virtual time**

Variable-Interval Sampled Working Set (2)
采样间隔可变的工作集(VSWS)

- At the beginning of a sampling interval, the use bits of all the resident pages for the process are reset
- At the end, only the pages that have been referenced during the interval will have their use bit set
  - These pages are retained in the resident set of the process throughout the next interval, while the others are discarded
- Thus, the resident set size can only decrease at the end of an interval.
- The resident set size remains fixed or grows during an interval, because any faulted pages are added to the resident set during each interval.

Variable-Interval Sampled Working Set (3)
采样间隔可变的工作集

- **The VSWS policy is driven by three parameters:**
  - **M:** ***the minimum duration of the sampling interval***
  - **L:** ***the maximum duration of the sampling interval***
  - **Q:** ***the number of page faults that are allowed to occur between sampling instances***

Variable-Interval Sampled Working Set (4)
采样间隔可变的工作集

- **The VSWS policy works as follows:**
  - **If the virtual time since the last sampling instance reaches *L*, then suspend the process and scan the use bits**
  - **If, prior to an elapsed virtual time of *L*, *Q* page faults occur**
    - **If the virtual time since the last sampling instance is less than *M*, then wait until the elapsed virtual time reaches *M* to suspend the process and scan the use bits**
    - **If the virtual time since the last sampling instance is greater than or equal to *M*, suspend the process and scan the use bits**

Variable-Interval Sampled Working Set (5)
采样间隔可变的工作集

- **How to select the parameter values?**
  - The parameter values should make the sampling normally be triggered by the occurrence of the Qth page fault after last scan
  - The other two parameters (M and L) provide boundary protection for exceptional conditions.
- **How does the VSWS policy reduce the peak memory demands caused by abrupt interlocality transitions?**
  - VSWS increases the sampling frequency, and hence the rate at which unused pages drop out of the resident set, when the paging rate increases

Cleaning Policy (1)
清除策略

- **A cleaning policy is concerned with determining *when a modified page should be written out to secondary memory***
  - **It is the opposite of a fetch policy**
- **Two common alternatives**
  - **Demand cleaning** (请求清除): **A page is written out only when it has been selected for replacement**
    - **Page writes are minimized**
  - **Precleaning** (预清除): **Modified pages are written out before their page frames are needed, so that pages can be written out in batches** (成批地写出)

Cleaning Policy (2) Drawbacks

- **With precleaning**
  - It makes little sense to write out hundreds or thousands of pages, if most of them will be modified again before they are replaced
- **With demand cleaning**
  - A page fault may have to wait for two page transfers before it can be unblocked

Cleaning Policy (3)
Approach with Page Buffering

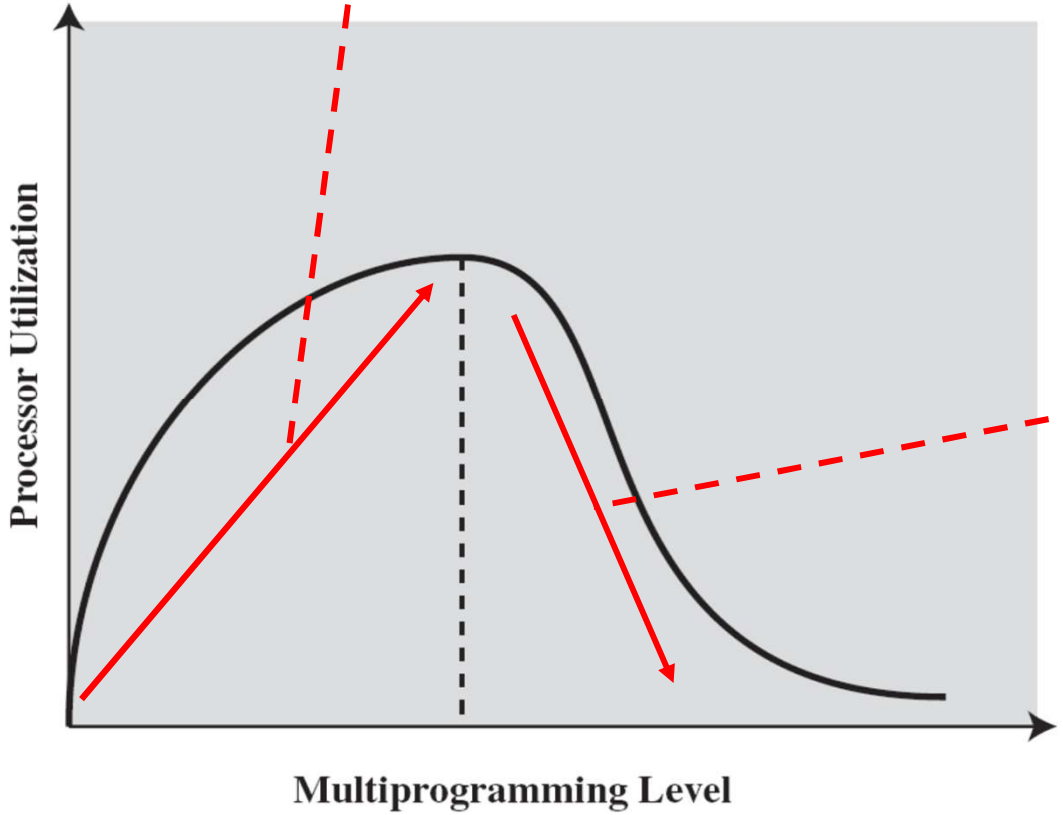
- **A better approach uses page buffering**
  - Clean only pages that are replaceable, but decouple the cleaning and replacement operations
  - Replaced pages are placed in two lists: Modified and unmodified
    - Pages in the modified list are periodically written out in batches and moved to the unmodified list
    - Pages in the unmodified list are either reclaimed if referenced again or lost when its frame is assigned to another page

Load Control (1)
负载控制

- **Load control is concerned with determining the number of processes that will be resident in main memory**
  - The number of processes that are resident in main memory is referred to as multiprogramming level (多道程序度)
- **If *too few processes*, there will be many occasions when all processes will be blocked and much time will be spent in swapping**
- **If *too many processes* are resident, the average resident set size will be inadequate, which will lead to thrashing (抖动/颠簸)—frequent page faults**

Load Control (2)
Multiprogramming Level

At the beginning, the higher the multiprogramming level is, the less the chance that all resident processes are blocked



After a certain point where the average resident set is inadequate, the number of page faults rises dramatically, and the processor utilization collapses

Load Control (3)
Thrashing (抖动/颠簸)

- **A process is thrashing if it is spending more time paging than executing**
- **Swapping out a piece of a process *just before* that piece is needed**
  - **The OS will have to get that piece again almost immediately**
  - ***Too much* of this leads to thrashing.**

Load Control (4)

- There are a number of ways to approach this problem.
- A working set or page fault frequency algorithm implicitly incorporates load control
  - *Only those processes whose resident set is sufficiently large are allowed to execute*

Load Control (5)
Process Suspension

- If the degree of multiprogramming is to be reduced, one or more of the currently resident processes must be suspended (swapped out)

Load Control (6)
Which process to suspend?

- **Lowest priority process: This implements a scheduling policy.**
- **Faulting process: The faulting process does not have its working set resident.**
- **Last process activated: This is least likely to have its working set resident**
- **Process with smallest resident set: This will require the least future effort to reload.**
- **Largest process: This obtains the most free frames, making additional deactivations unlikely happen**
- **Process with the largest remaining execution window: this approximates a shortest-processing-time-first scheduling discipline.**