Operating Systems

Chapter 8 Virtual Memory (虚拟内存)

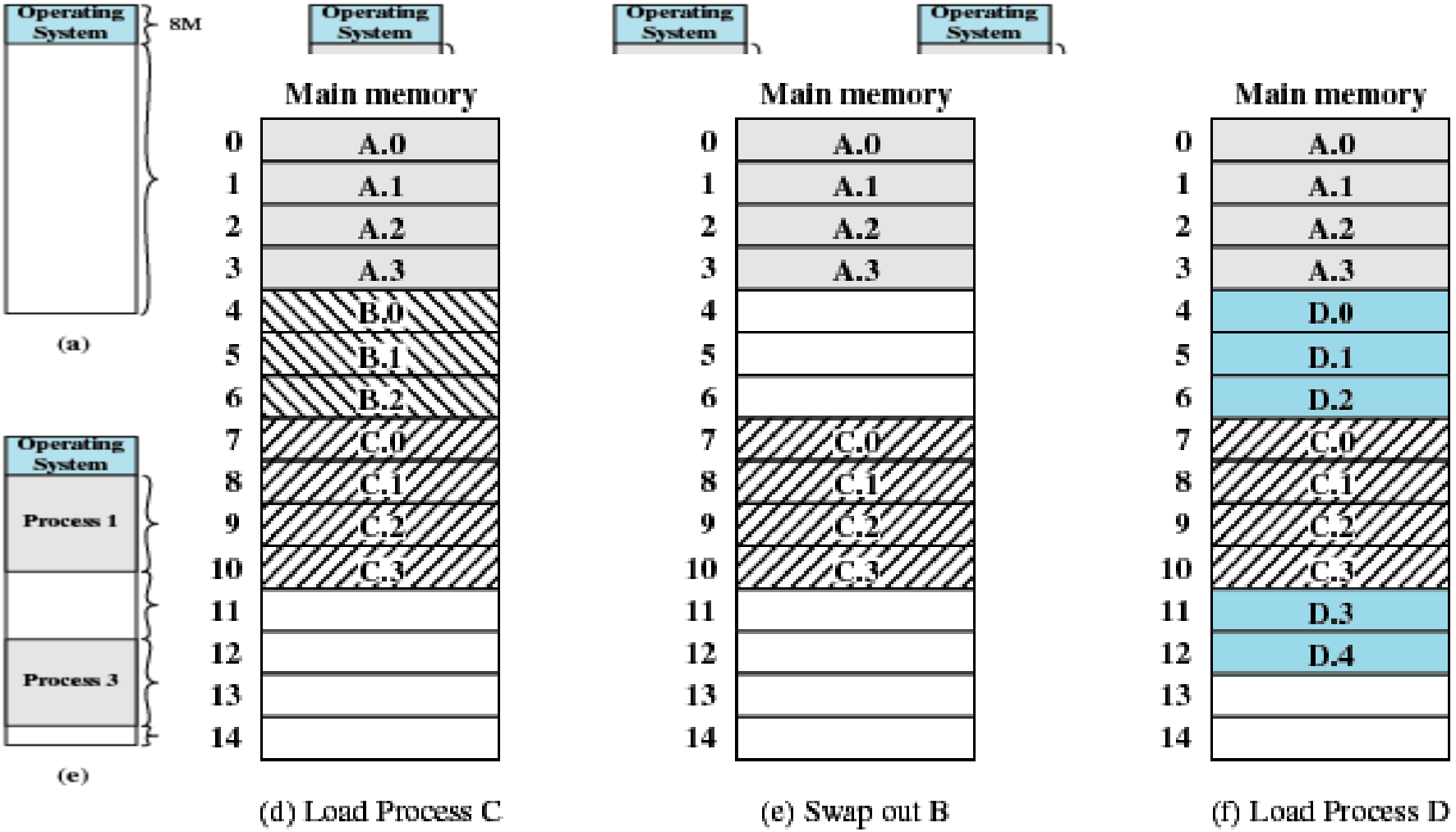
A simple review(1/3)

- In chapter 7:
  - Relocation
    - Memory references are dynamically translated into physical addresses at run time(运行时访问地址动态转换)

– Fixed partition/dynamic partition
– Paging/Segmentation
  - A process may be broken up into pieces that do not need to located contiguously in main memory(进程分块)

– The whole process is resident in memory 进程整体驻留于内存

A simple review (2/3)



Fig

Figure 7.9 Assignment of Process Pages to Free Frames

A simple review (3/3)

- Problem unsolved:
  - Paging: Page table need to be stored somewhere and accessed efficiently
    - TLB, multi-level paging table, Inverted Page Table
  - Limited memory: logic mem <,=,> physical mem
    - Virtual memory, Replacement Policy, Resident Set Management, Clearing Policy

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

8.1.0 Overview(1/6)

- Virtual memory definition: The memory to programmer's view

– 内存 + 部分辅存构成 A storage allocation scheme in which secondary memory can be addressed as though it were part of main memory.
– 逻辑和物理地址自动转换 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.
– 尺寸不受内存限制，受计算机寻址和可用辅存限制 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.

8.1.0 Overview(2/6)

- 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

Table 8.2 Characteristics of Paging and Segmentat

Simple Paging	Virtual Memory Paging
Main memory partitioned into small fixed-size chunks called frames.	
Program broken into pages by the compiler or memory management system.	
Internal fragmentation within frames.	
No external fragmentation.	
Operating system must maintain a page table for each process showing which frame each page occupies.	
Operating system must maintain a free-frame list.	
Processor uses page 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.
	Reading a page into main memory may re-quire writing a page out to disk.

8.1.0 Overview(3/6)

- 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

8.1.0 Overview(4/6)

- 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

8.1.0 Overview(5/6)

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

8.1.0 Overview(6/6)

- Thrashing 系统抖动 / 系统颠簸
  - 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 and Virtual Memory 局部性原理
- 8.1.2 Paging
- 8.1.3 Segmentation
- 8.1.4 Combined Paging and Segmentation
- 8.1.5 Protection and Sharing

8.1.1 Locality and Virtual Memory(1/3)

```
void main()
{
    int arr[50][50];
    for(int i=0;i<50;i++)
        for(int j=0;j<50;j++)
            arr[i][j]=0;
}
```

```
void main()
{
    int arr[50][50];
    for(int j=0;j<50;j++)
        for(int i=0;i<50;i++)
            arr[i][j]=0;
}
```

Locality ： code and data

8.1.1 Locality and Virtual Memory (2/3)

- Locality suggests that virtual memory may work efficiently
  - 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

8.1.1 Locality and Virtual Memory (3/3) ---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 分页
- 8.1.3 Segmentation
- 8.1.4 Combined Paging and Segmentation
- 8.1.5 Protection and Sharing

8.1 Hardware and Control Structure

- 8.1.2 Paging 分页
  - 8.1.2.0 Introduction to Paging
  - 8.1.2.1 Multi-level Paging System 多级页表
  - 8.1.2.2 Inverted Page Table(反向 / 倒排页表)
  - 8.1.2.3 Translation Lookaside Buffer(快表)
  - 8.1.2.4 Page Size

8.1.2.0 Introduction to Paging(1/7)

- 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

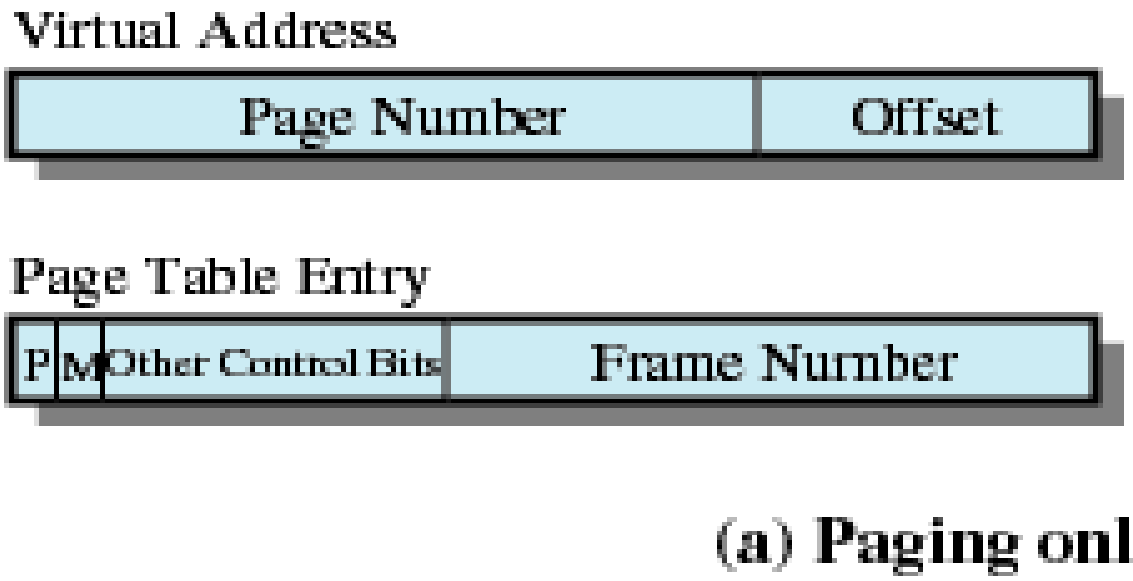
- Present Bit(存在位):
  - A bit is needed to indicate whether the page is in main memory or not
  - If try to access an address that is not in memory, a page fault 缺页 interrupt will be generated

8.1.2.0 Introduction to Paging(2/7)

- 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)

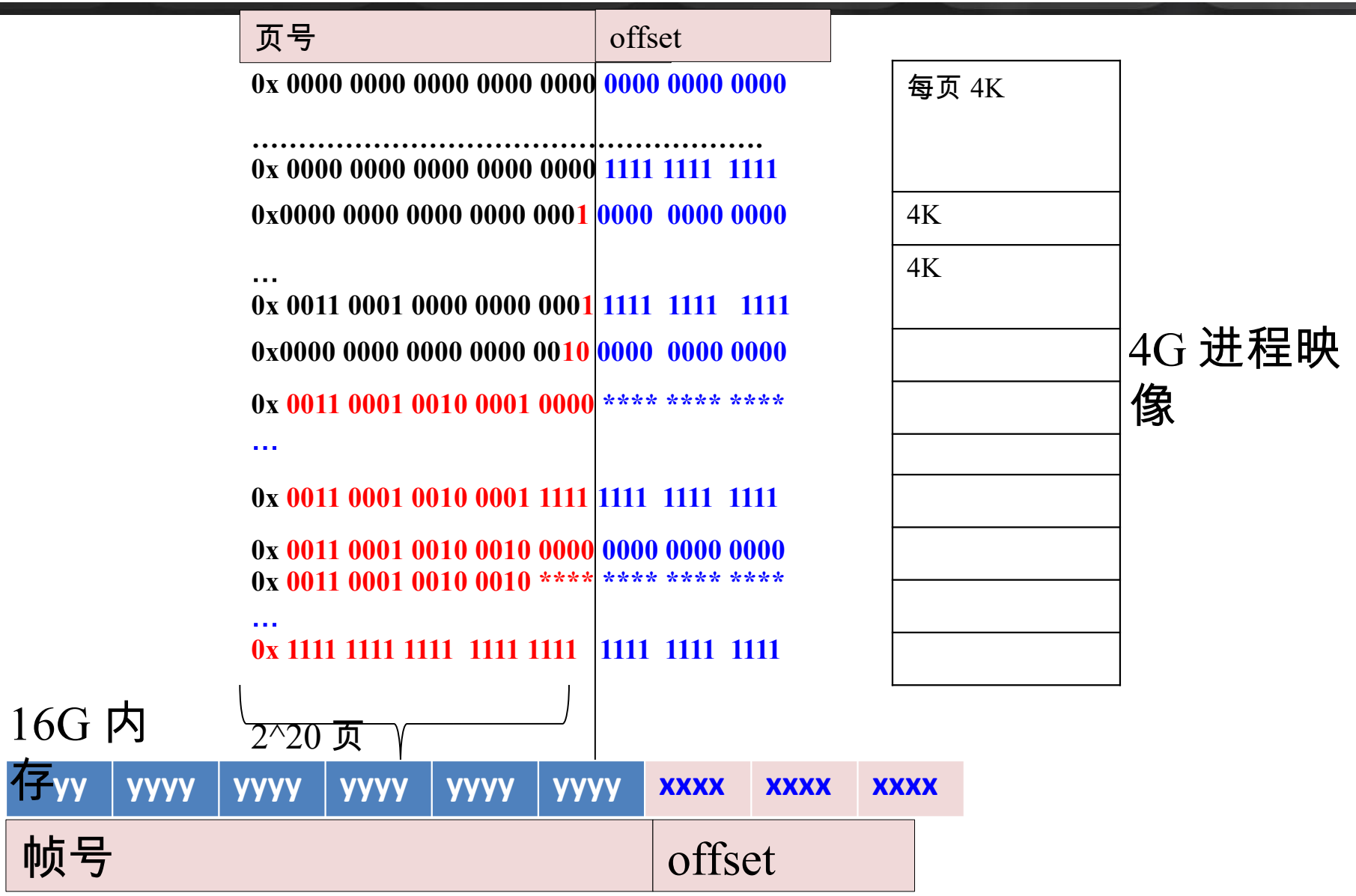
8.1.2.0 Introduction to Paging (3/7)

Page Table Structure



(a) Paging only

8.1.2.0 Introduction to Paging (4/7)



8.1.2.0 Introduction to Paging (5/7)

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

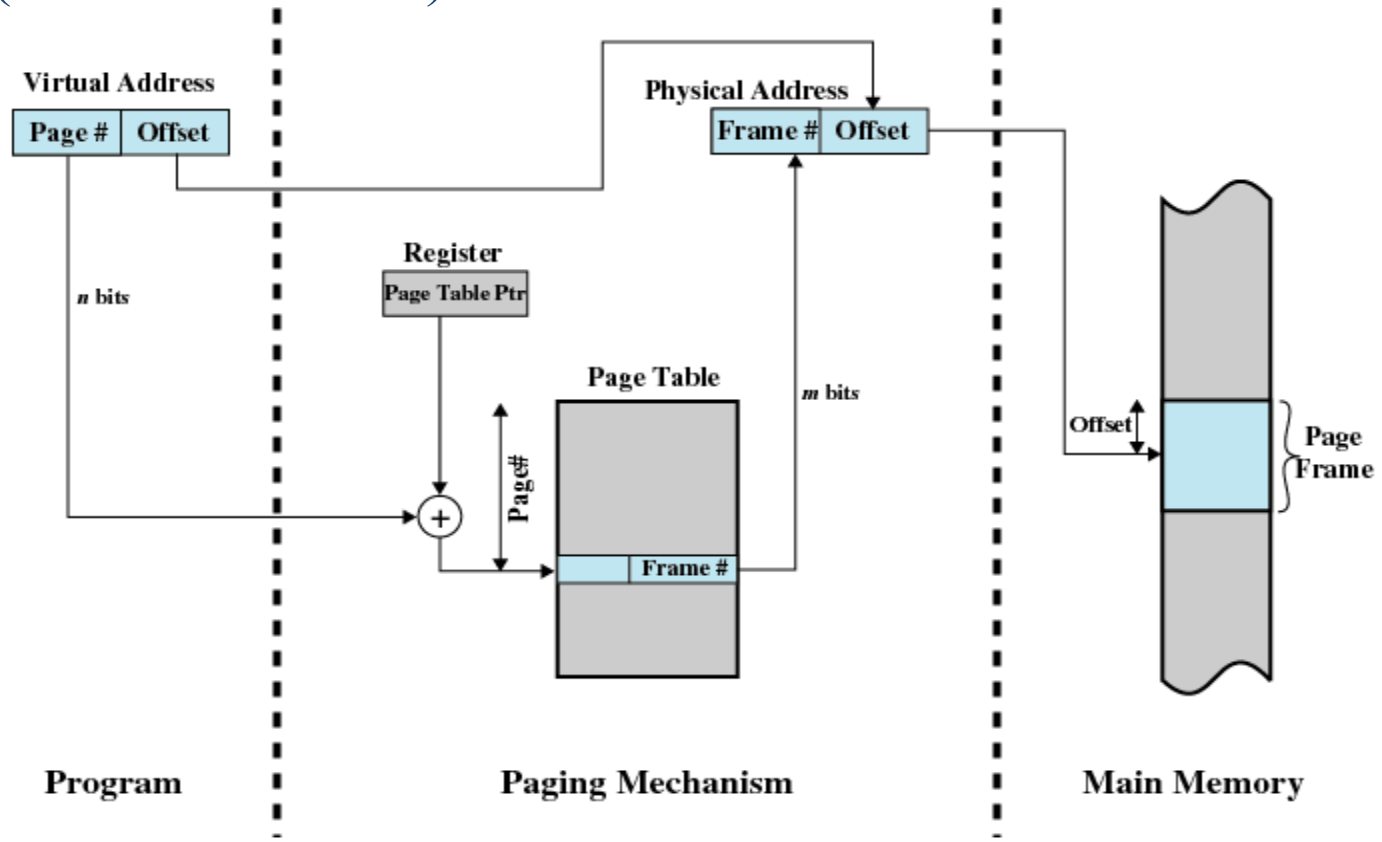


Figure 8.3 Address Translation in a Paging System

8.1.2.0 Introduction to Paging (6/7)

- 考虑：访问效率和存储空间
  - 分页机制中页表的存放问题：内存？
    - 那么一次对内存的访问需要访问两次内存
      - 读内存中的页表项
      - 读物理地址中需要的数据
  - 页表存放需要多大的内存空间？
    - 假设物理内存大小为 1G，共有 32 个进程，每个进程为 4G ，每页大小为 4k ， 每个页表项为 4 字节，那么存储所有页表需要多少空间？
      - 每个进程需要 1M 个页，对应的页表大小为 1M*4B=4MB
      - 32 个进程共需要 128MB

8.1.2.0 Introduction to Paging (7/7)

- 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)

8.1 Hardware and Control Structure

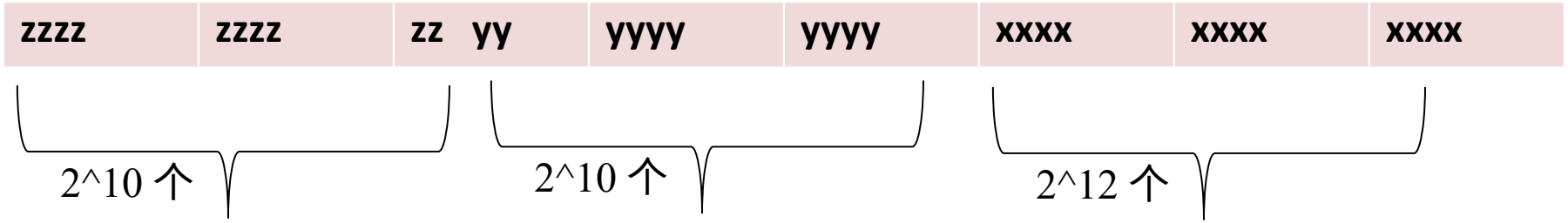
- 8.1.2 Paging 分页
  - 8.1.2.0 Introduction to Paging
  - 8.1.2.1 Multi-level Paging System 多级页表
  - 8.1.2.2 Inverted Page Table(反向 / 倒排页表)
  - 8.1.2.3 Translation Lookaside Buffer(快表)
  - 8.1.2.4 Page Size

8.1.2.1 Multi-level Paging System(1/8)

20203141356　　学号

年级	专业	第几人
2020	314	1356
2019	314	0096

8.1.2.1 Multi-level Paging System(2/8)



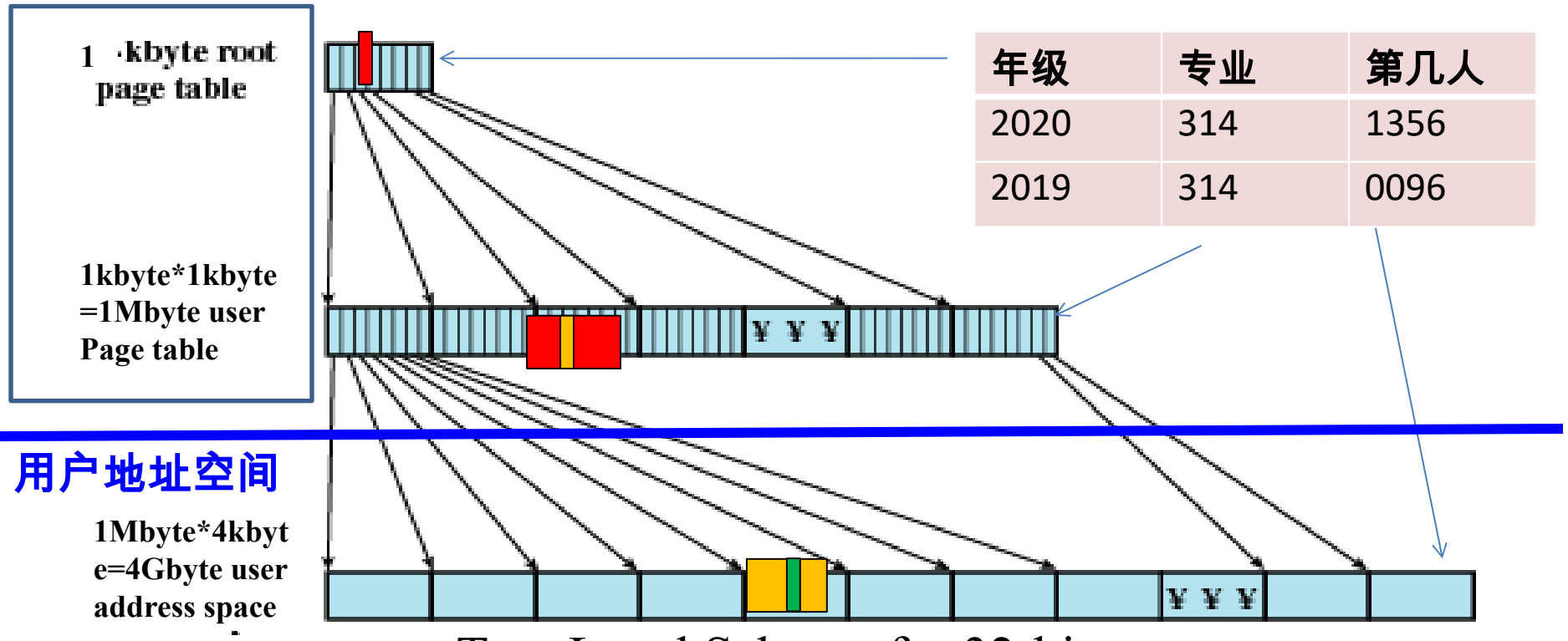
年级	专业	第几人
2020	314	1356
2019	314	0096

学号

20203141356

8.1.2.1 Multi-level Paging System(3/8)

2 级页表



年级	专业	第几人
2020	314	1356
2019	314	0096

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

Figure 8.4 A Two-Level Hierarchical Page Table

8.1.2.1 Multi-level Paging System(4/8)

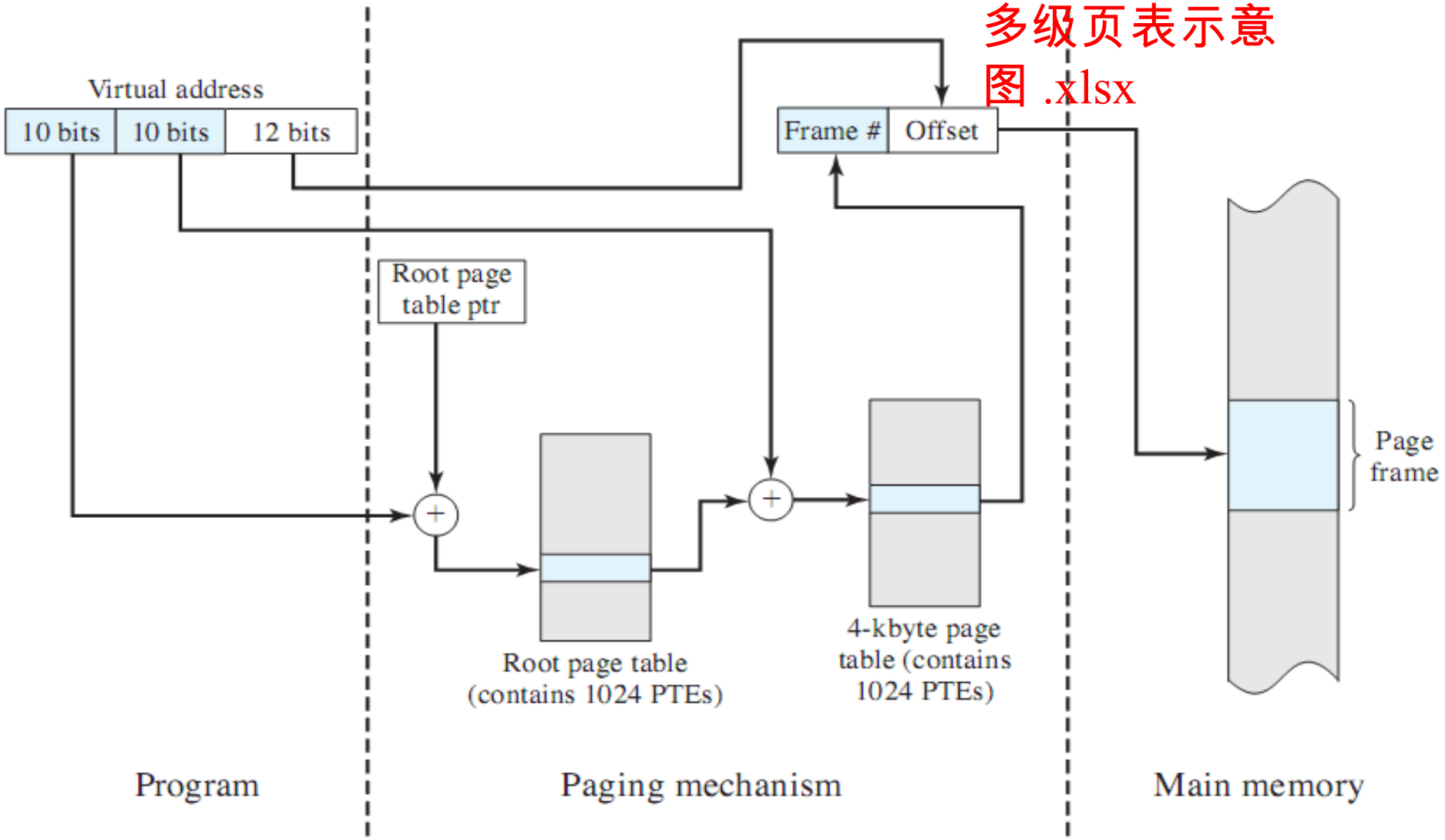


Figure 8.5 **Address Translation in a Two-Level Paging System**

8.1.2.1 Multi-level Paging System(5/8)

- Multi-level paging system
  - 部分解决了存储空间的问题
    - 多级页表为什么省内存？二级页表可以不存在
  - 但访问机制的复杂性导致时间开销进一步增大，需要硬件的支持
  - 考虑：
    - 存储空间过大是因为为每个进程维护一个页表，而且进程所需逻辑空间的大小影响页表的大小
    - **物理内存是一定的**

8.1.2.1 Multi-level Paging System(6/8)

- OpenEuler 下的多级页表
  - 39 位虚地址下，支持 3 级页表和 2 级页表
  - 页表项 8B ，分页粒度 4KB ，每个页框保存 512 项记录： 8B*512 -> 4K

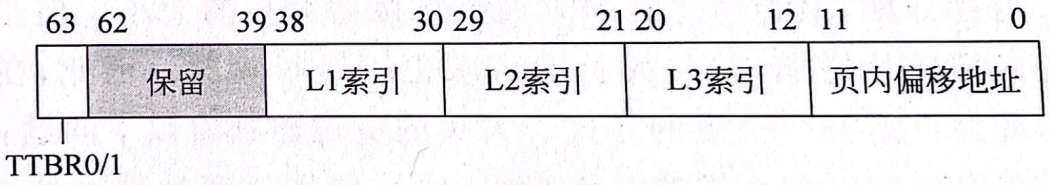


图 5-22　页长度为 4KB，39 位虚拟地址结构

8.1.2.1 Multi-level Paging System(7/8)

- OpenEuler 支持下的标准大页
  - 问题提出：分配 4MB 空间需要 1024 个 4KB 页的页表项，在查询过程中有 1024 次 TLB 不命中和页表查询
  - 解决：大页：二级和一级索引部分用作块描述符：如 L3 为块描述符，则 2^(12+9) 对应页尺寸为 2MB

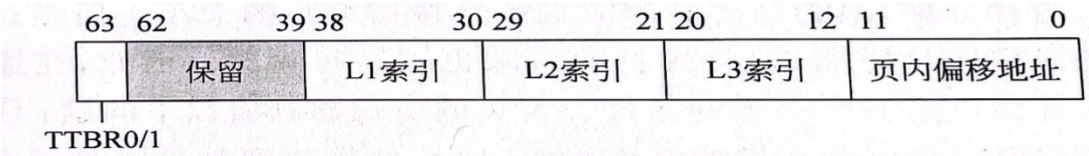


图 5-22　页长度为 4KB，39 位虚拟地址结构

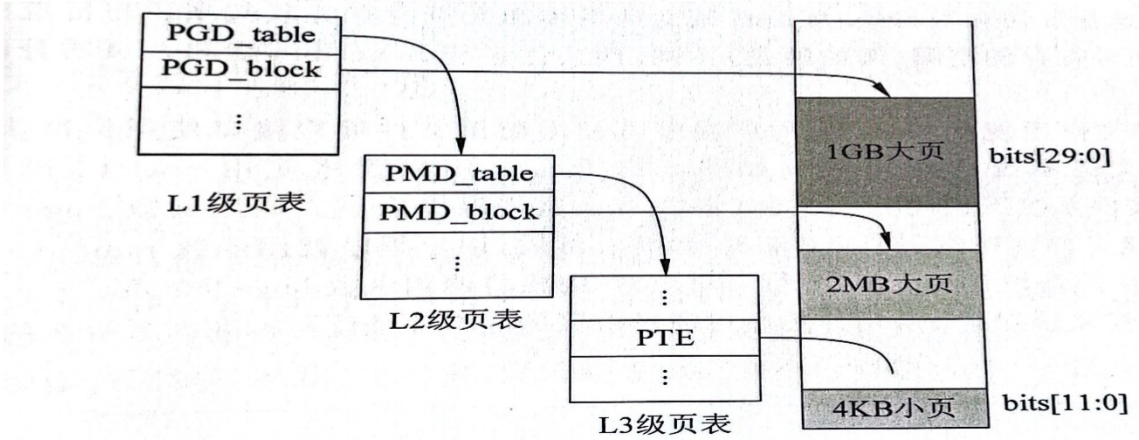


图 5-23　支持大页的块描述符原理

8.1.2.1 Multi-level Paging System(8/8)

- OpenEuler 支持下的标准大页
  - MMU 支持大页机制下的块描述符
  - openEuler 用 hstate 记录大页信息
  - openEuler 采用标准大页池，用户申请空间时，操作系统在大页池寻找空闲大页

8.1 Hardware and Control Structure

- 8.1.2 Paging 分页
  - 8.1.2.0 Introduction to Paging
  - 8.1.2.1 Multi-level Paging System 多级页表
  - 8.1.2.2 Inverted Page Table(反向 / 倒排页表)
  - 8.1.2.3 Translation Lookaside Buffer(快表)
  - 8.1.2.4 Page Size

8.1.2.2 Inverted Page Table(1/8)(反向 / 倒排页表)

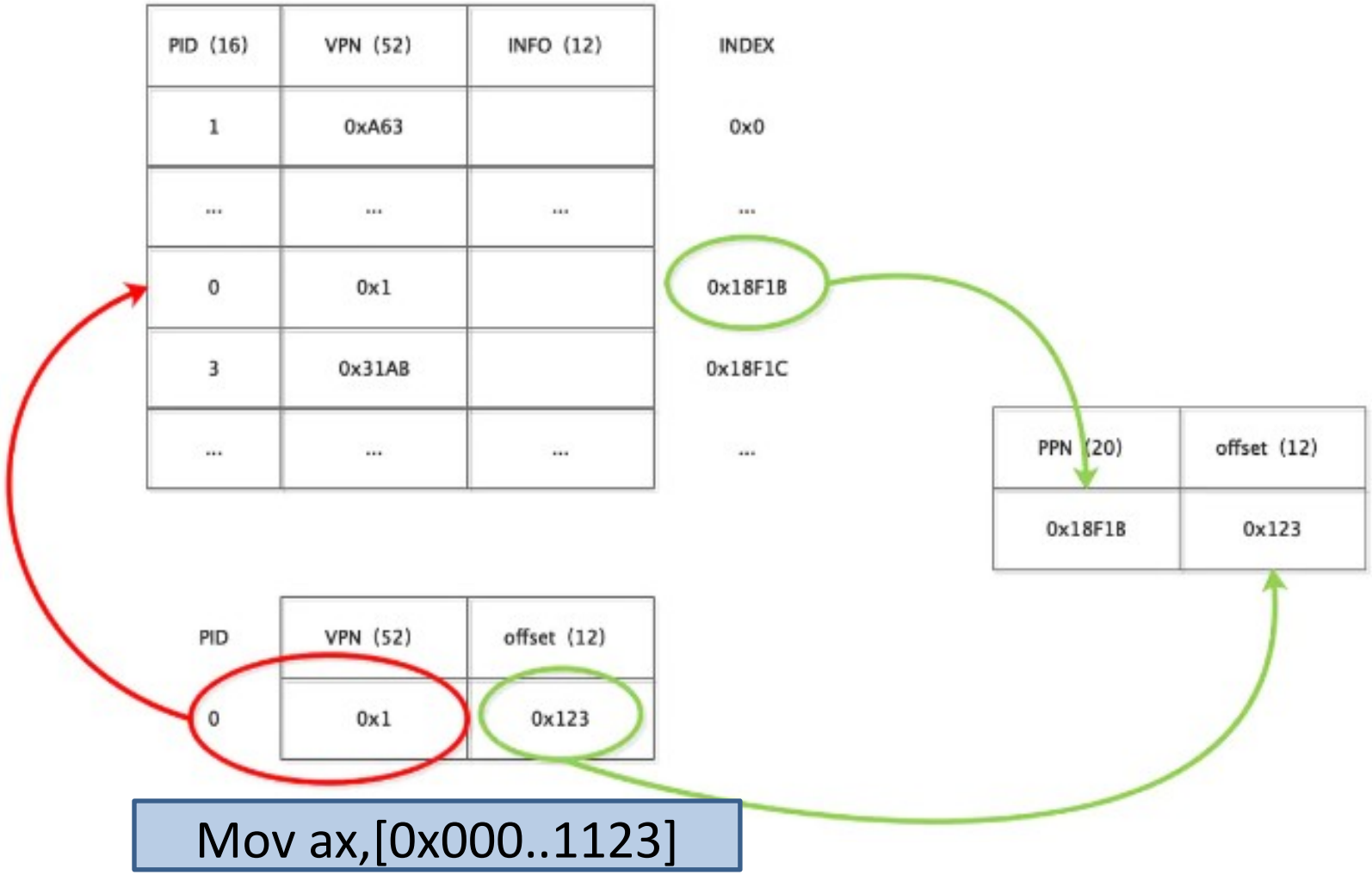
- The page table’s structure is called *inverted* because it indexes page table entries by frame number rather than by virtual page number.

8.1.2.2 Inverted Page Table(2/8)

- **线性反向页表**
  - 线性倒排页表是倒排页表的基本形式，由于它是全局唯一的，并且每个进程有自己独立的地址空间，表项中必须包含进程 ID 。表项由三部分组成，进程 ID （ PID ， 16 位），虚拟页号（ VPN ， 52 位），信息位（ INFO ， 12 位，包含该页的一些保护信息），共 80 位（ 10Bytes ）。物理页框号由表项索引隐含表示。
  - 设虚拟地址空间为 64 位，页面大小为 4KB ，页表项大小为 4Bytes ，物理内存大小为 512MB ，
    - 所有表项占用的内存大小为 2^(64-12) 个 *4B
    - 用反向页表的表项数目为： 2^(29-12) = 128K 个

8.1.2.2 Inverted Page Table(3/8)

- **线性反向页表**



8.1.2.2 Inverted Page Table(4/8)

- **线性反向页表**
  - 为了加快地址转换速度，可以在线性转换表前增加一层散列表。散列表的输入是 PID?? 和 VPN ，输出是倒排页表的索引。利用散列表进行散列时可能发生冲突，可以利用链地址法解决冲突，我们通过在倒排页表项中增加 next 域使其能够构成链表（表头的索引位于散列表中）。
  - Vpn 页号
  - PID 线程号

8.1.2.2 Inverted Page Table(5/8)

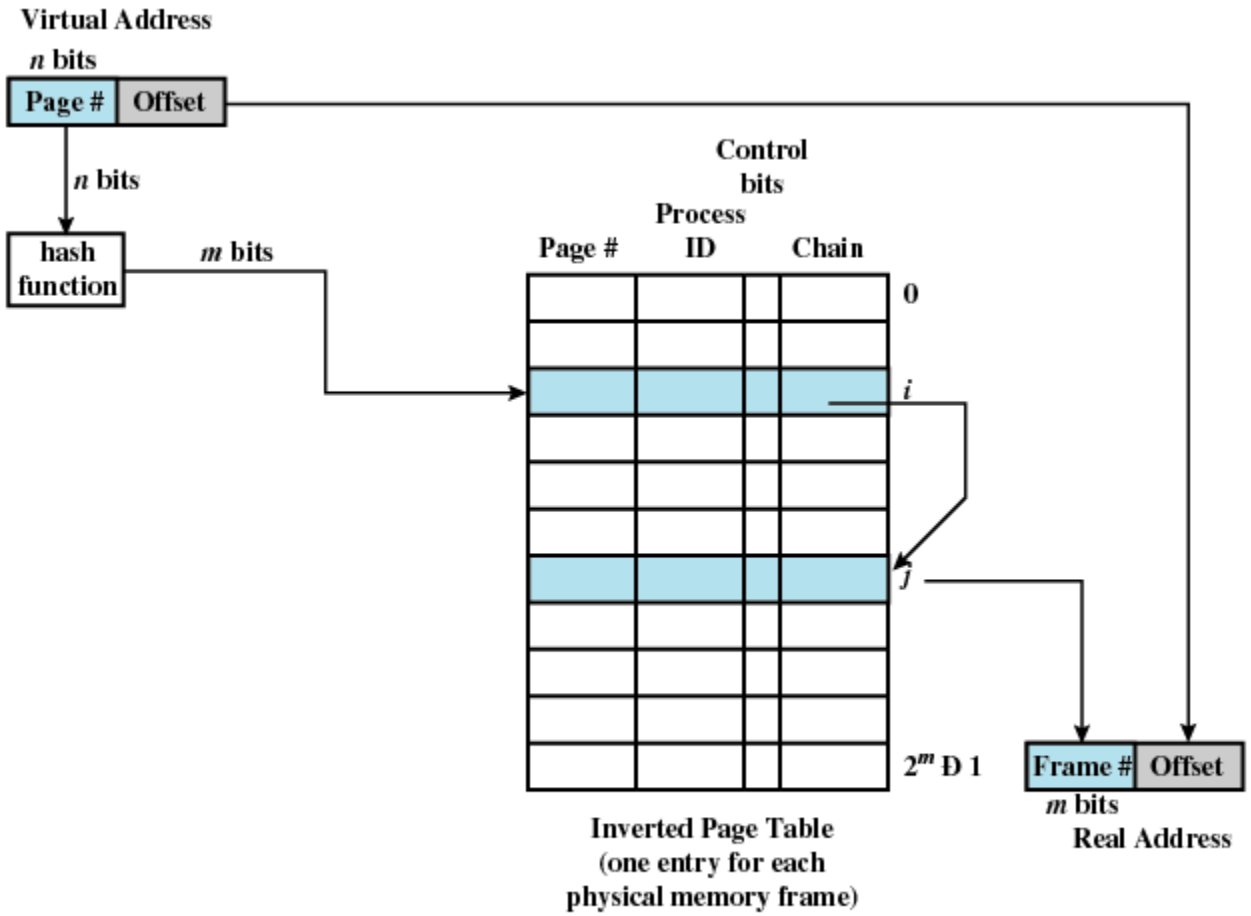


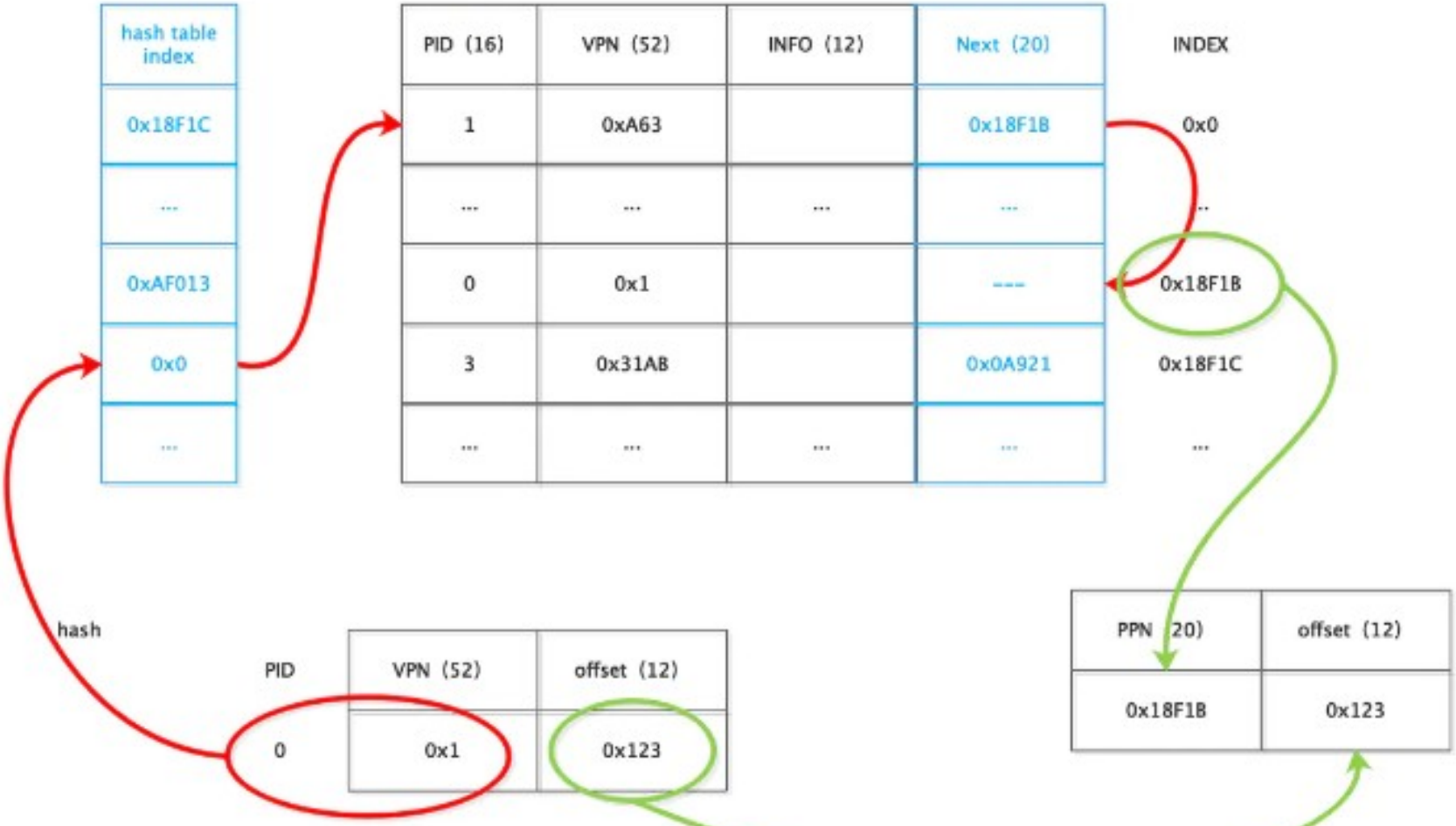
Figure 8.6 Inverted Page Table Structure

8.1.2.2 Inverted Page Table(6/8)

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

  - Only one table for all processes

8.1.2.2 Inverted Page Table(7/8)



Mov ax,[0x000..1123]

8.1.2.2 Inverted Page Table(8/8)

- 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

8.1 Hardware and Control Structure

- 8.1.2 Paging 分页
  - 8.1.2.0 Introduction to Paging
  - 8.1.2.1 Multi-level Paging System 多级页表
  - 8.1.2.2 Inverted Page Table(反向 / 倒排页表)
  - 8.1.2.3 Translation Lookaside Buffer(快表)
  - 8.1.2.4 Page Size

8.1.2.3 Translation Lookaside Buffer(1/7)(快表)

- 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 借鉴高速缓存设计
  - Called a Translation Lookaside Buffer (TLB)
  - Contains page table entries that have been most recently used

8.1.2.3 Translation Lookaside Buffer(2/7)

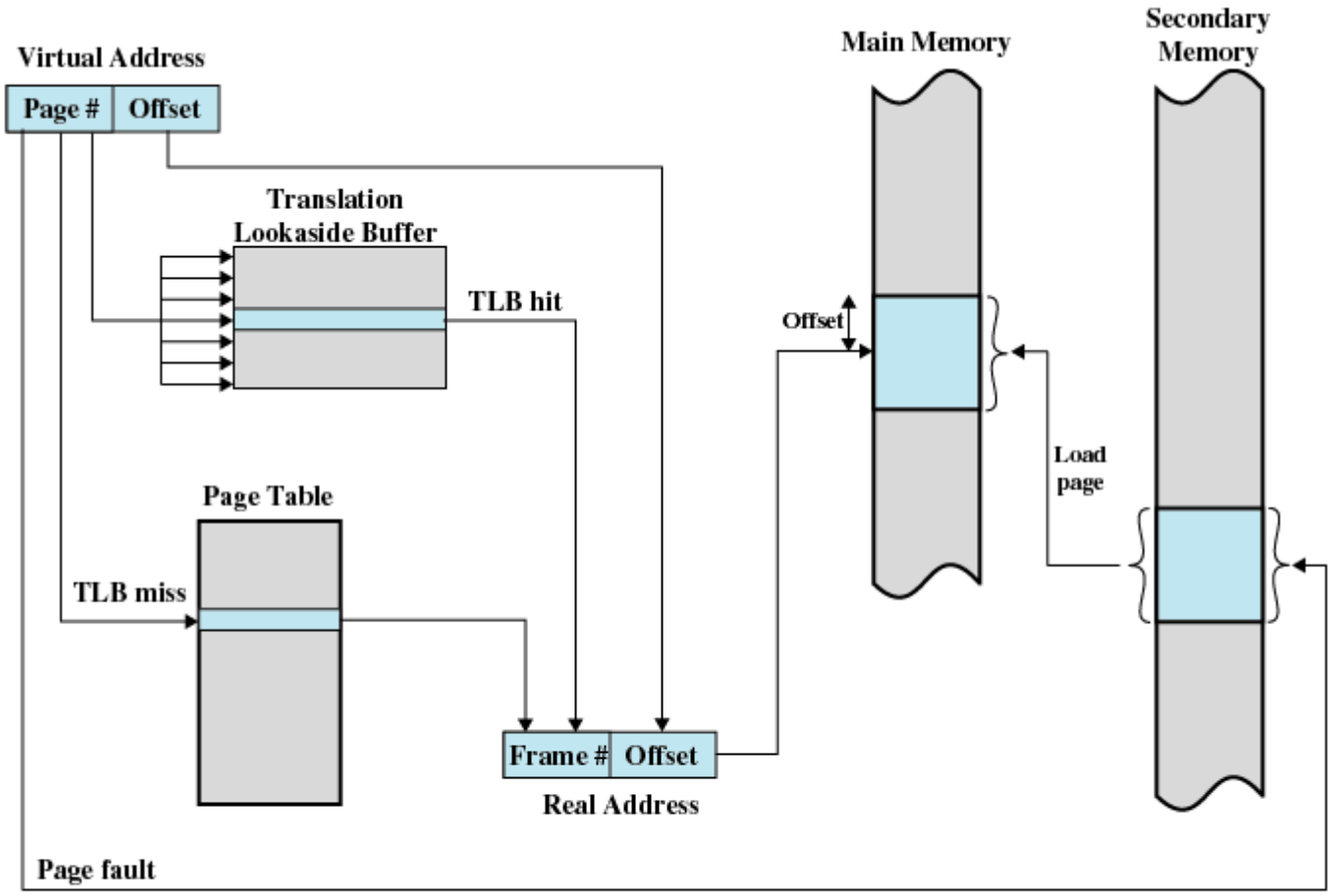


Figure 8.7 Use of a Translation Lookaside Buffer

8.1.2.3 Translation Lookaside Buffer(3/7)

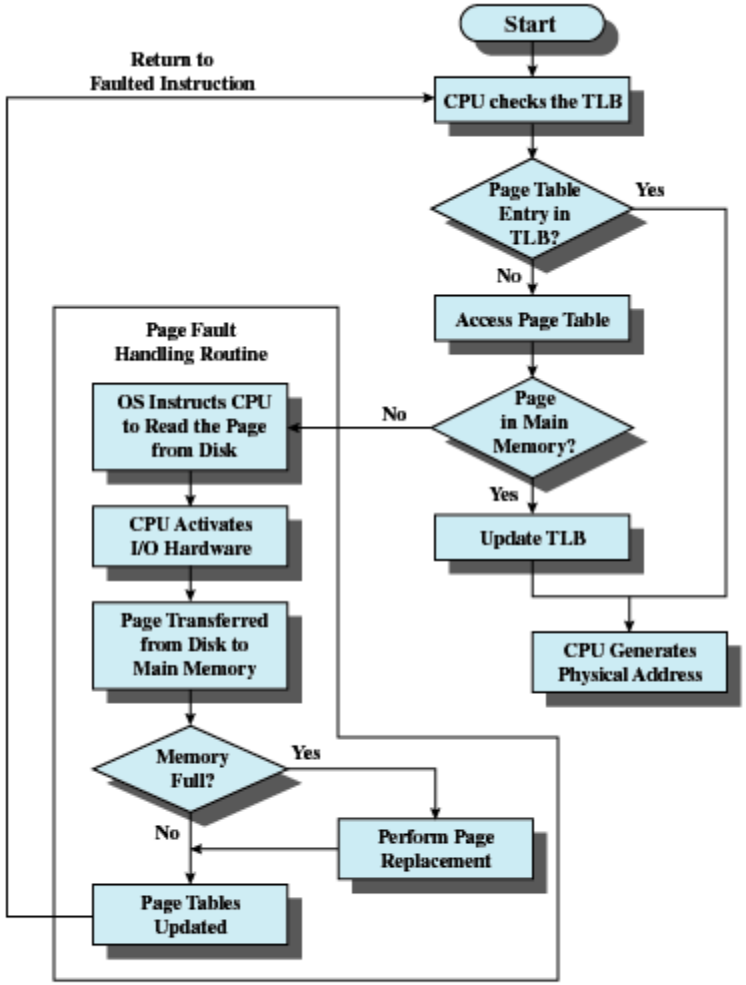


Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

8.1.2.3 Translation Lookaside Buffer(4/7)

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

8.1.2.3 Translation Lookaside Buffer(5/7)

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

8.1.2.3 Translation Lookaside Buffer(6/7)

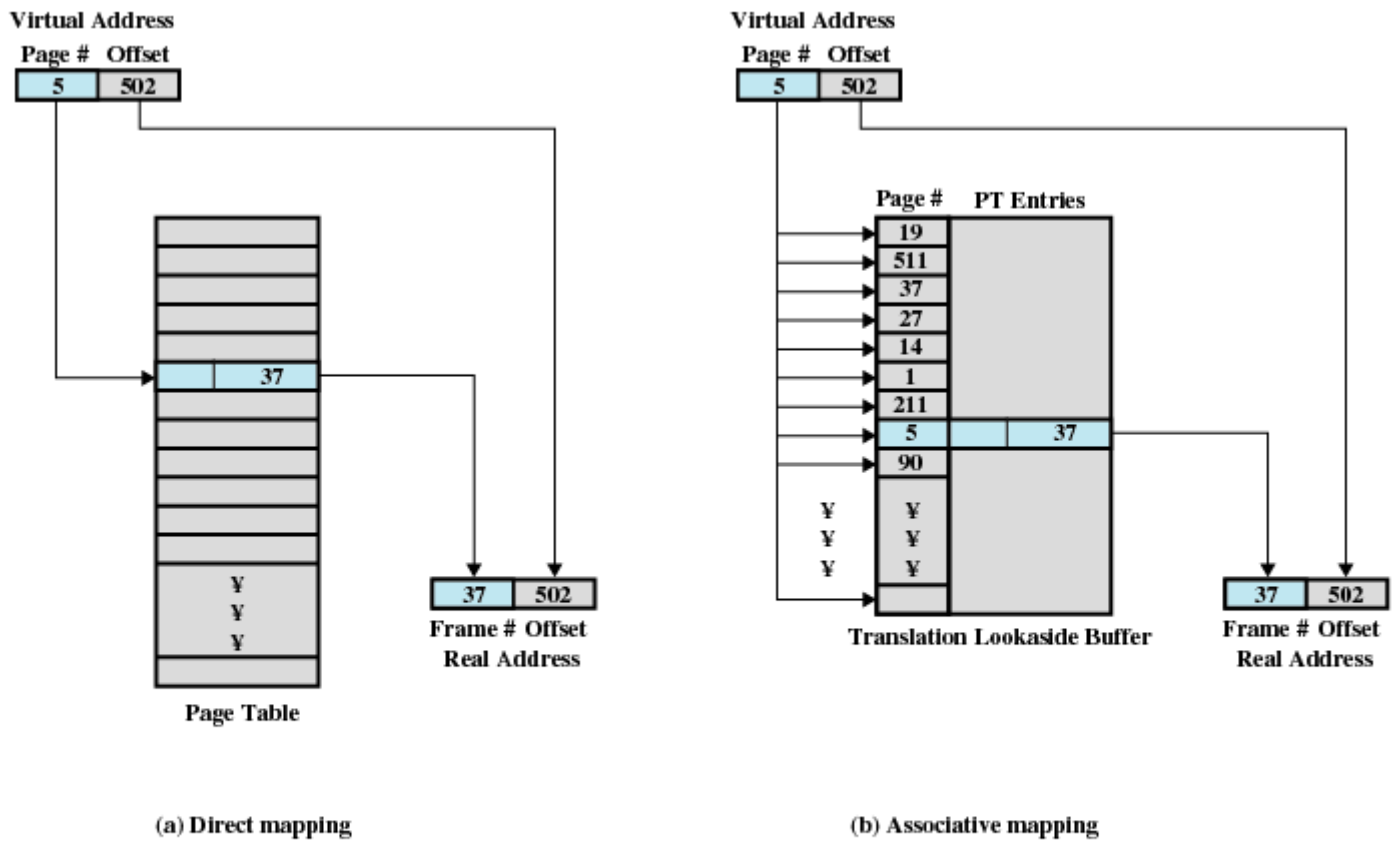


Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

直接映射

关联映射

8.1.2.3 Translation Lookaside Buffer(7/7)

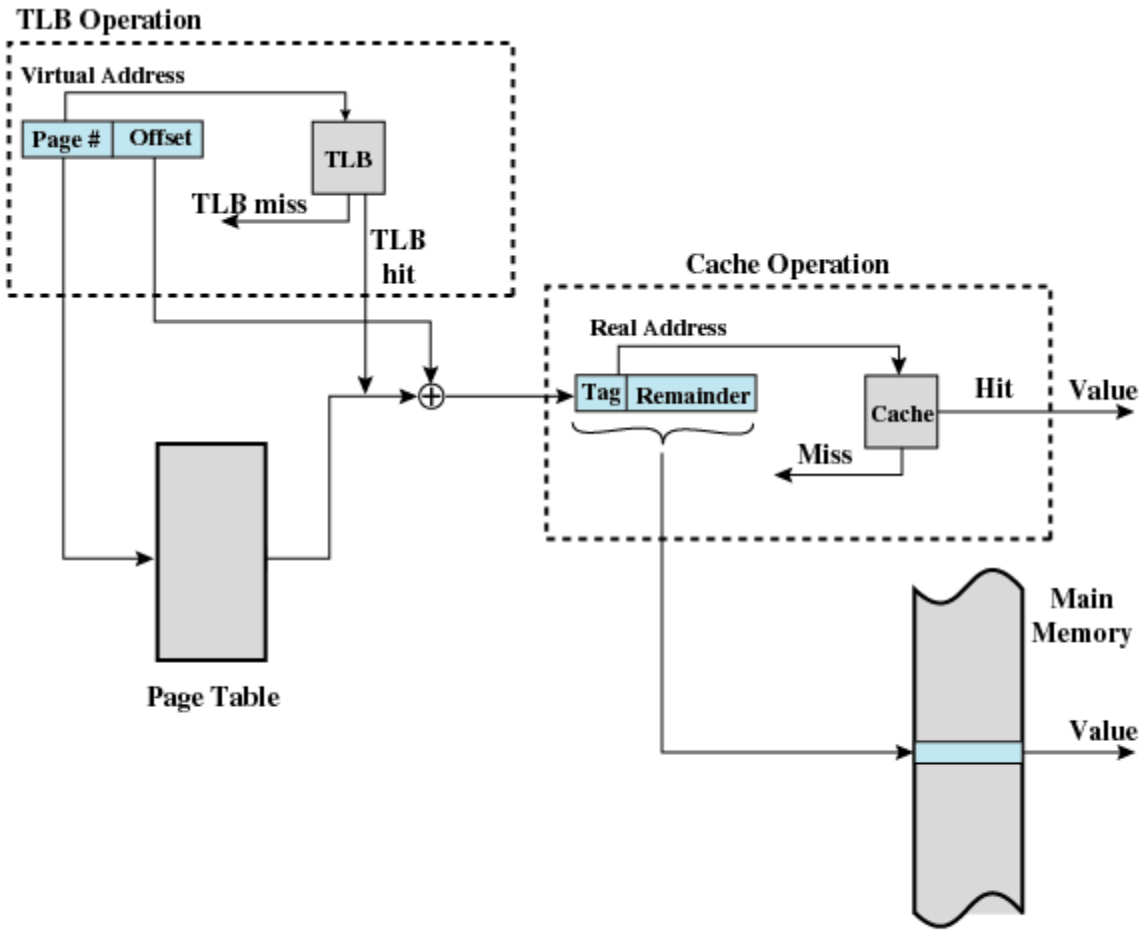


Figure 8.10 Translation Lookaside Buffer and Cache Operation

8.1.2.4 Page Size(1/4)(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(DMA 传递大块效率高)

8.1.2.4 Page Size(2/4)

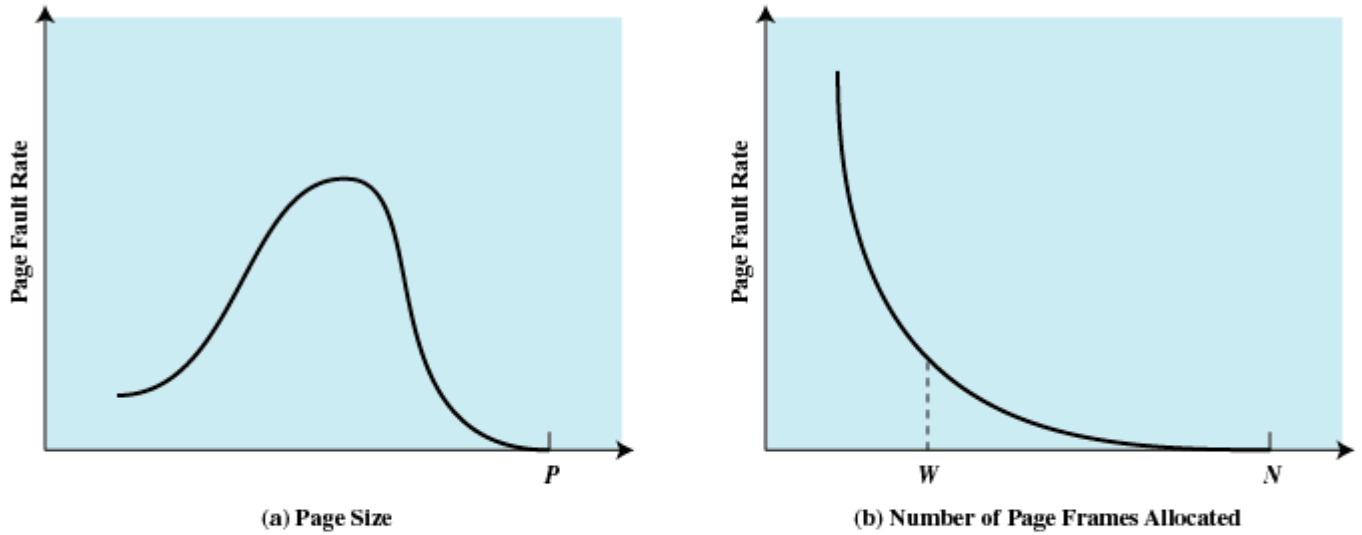
- Small page size, large number of pages will be found in main memory(页越小，进程在内存中的页越多)
  - As time goes on during execution, the pages in memory will all contain portions of the process near recent references. Page faults low.

- Increased page size causes pages to contain locations further from any recent reference. Page faults rise.

- However the page fault rate will begin to fall as the size of page approaches a point.

- Huge Page 大页模式 ppt page 32

8.1.2.4 Page Size(3/4)



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

8.1.2.4 Page Size(4/4)

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 (分段)
- 8.1.4 Combined Paging and Segmentation
- 8.1.5 Protection and Sharing

8.1.3 Segmentation(1/4)

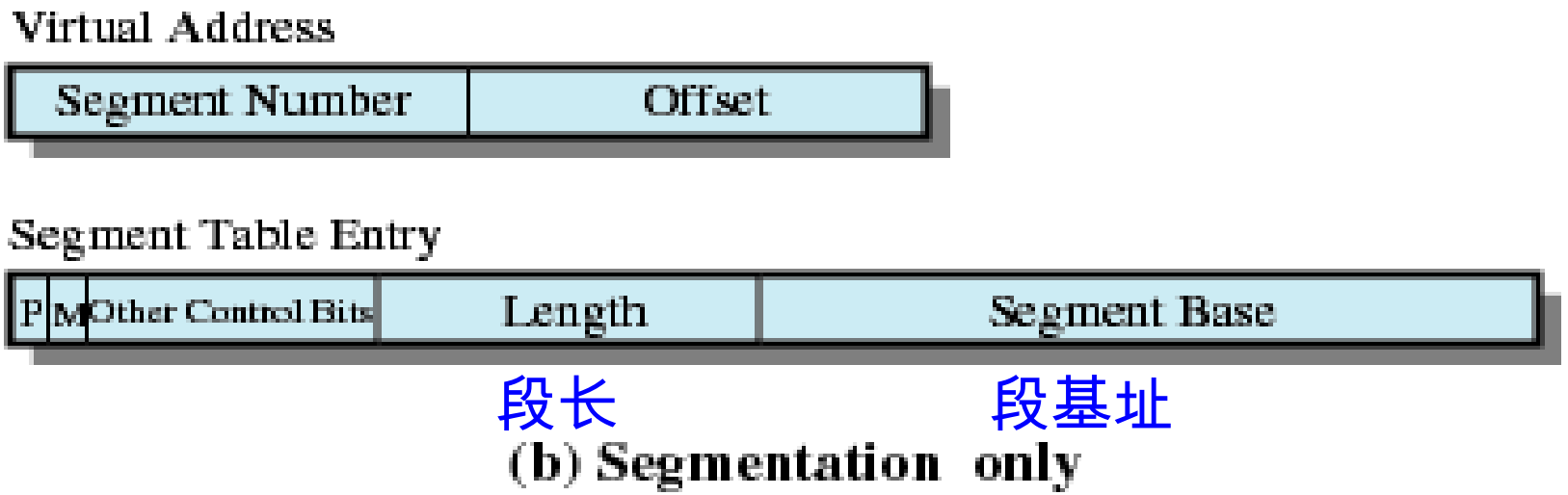
- May be unequal, dynamic size (大小不等), advantages:
  - Simplifies handling of growing data structures(简化不断增长的数据结构的处理)
  - Allows programs to be altered and recompiled independently (允许程序独立的修改和编译)
  - Lends itself to sharing data among processes (有助于进程间共享)
  - Lends itself to protection (有利于保护)
- Disadvantage?
  - 书上没有讨论从硬盘到内存加载时的单位问题

8.1.3 Segmentation(2/4)

- Segment Tables(段表)
  - Corresponding segment in main memory(对应一个内存段)
  - Each entry 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)

8.1.3 Segmentation(3/4)

Segment Table Entries



8.1.3 Segmentation(4/4)

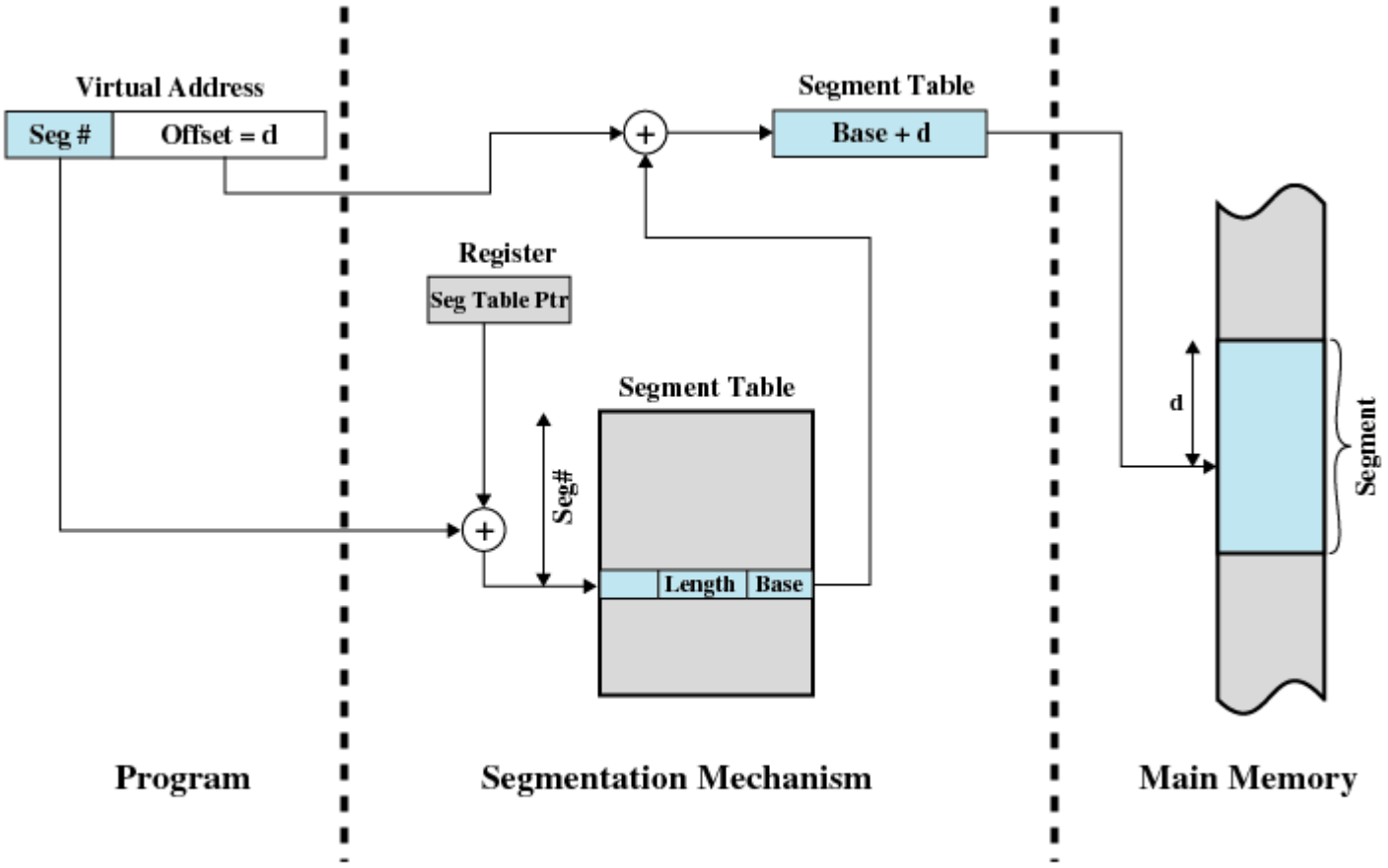


Figure 8.12 Address Translation in a Segmentation System

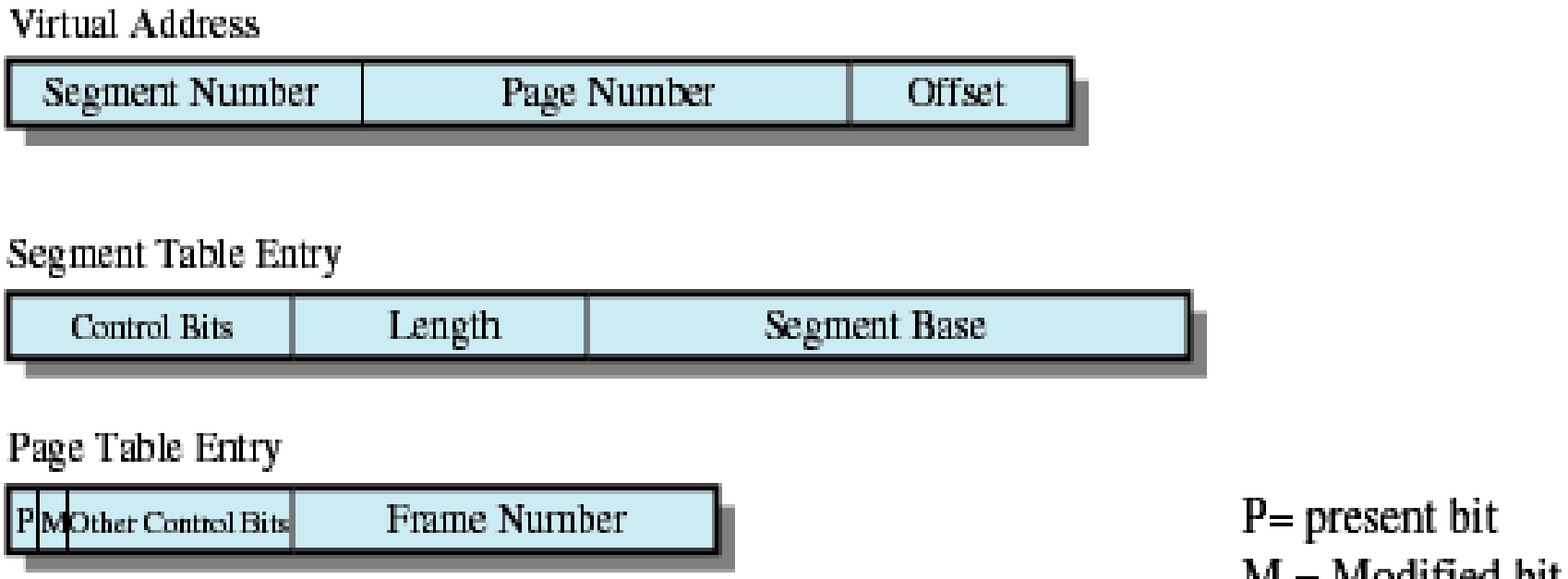
8.1 Hardware and Control Structure

- 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

8.1.4 Combined Paging and Segmentation(1/3)

- Paging is transparent to the programmer
- Segmentation is visible to the programmer
- Each segment is broken into fixed-size pages
- Each Process has a segment table and several segments
- Each segment has a page table

8.1.4 Combined Paging and Segmentation(2/3)



(c) Combined segmentation and paging

8.1.4 Combined Paging and Segmentation(3/3)

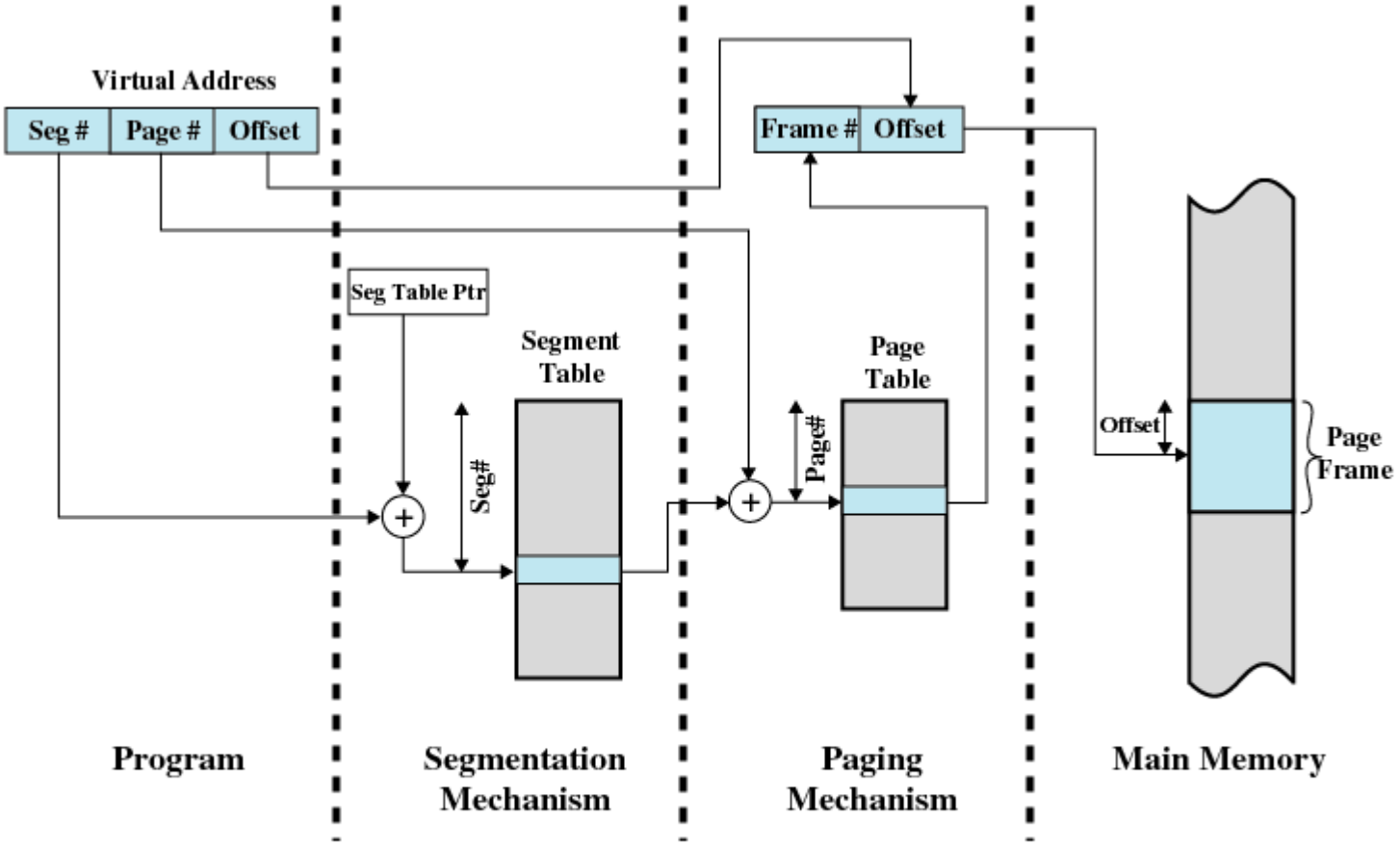


Figure 8.13 Address Translation in a Segmentation/Paging System

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

8.1.5 Protection and Sharing(1/2)

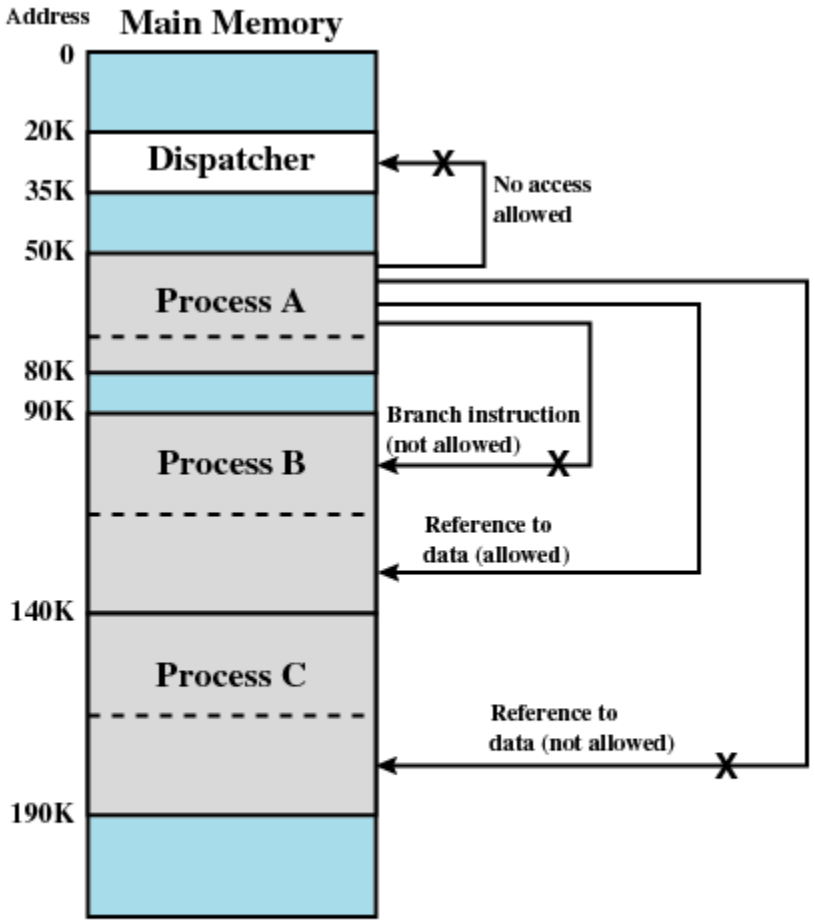


Figure 8.14 Protection Relationships Between Segments

8.1.5 Protection and Sharing (2/2)

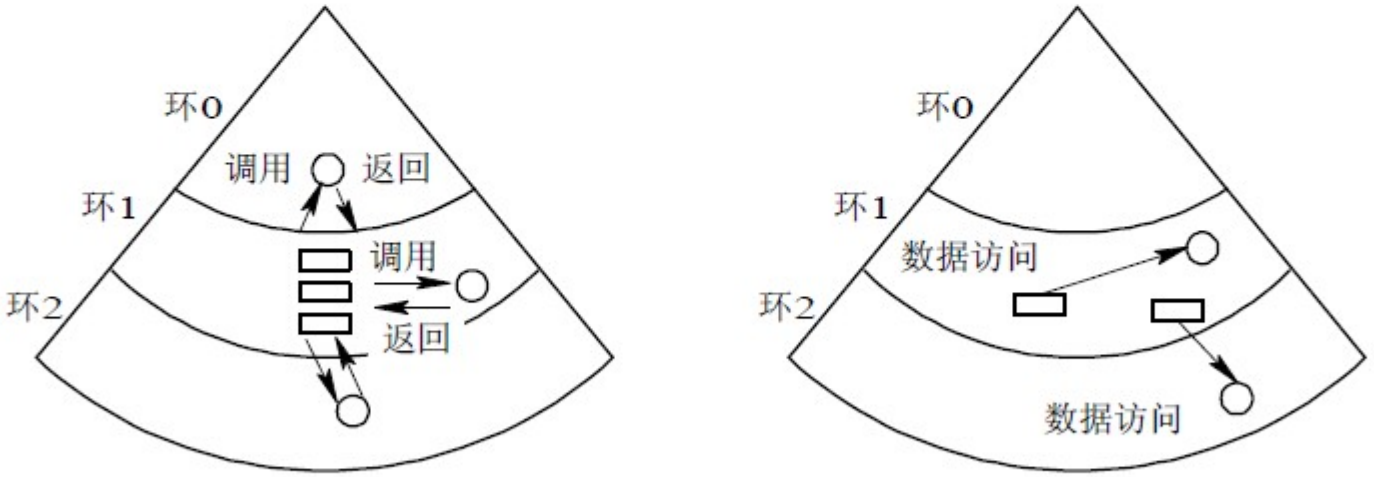
3) 环保护机构

这是一种功能较完善的保护机制。在该机制中规定：低编号的环具有高优先权。OS 核心处于 0 环内；某些重要的实用程序和操作系统服务占居中间环；而一般的应用程序则被安排在外环上。在环系统中，程序的访问和调用应遵循以下规则：

(1) 一个程序可以访问驻留在相同环或较低特权环中的数据。

(2) 一个程序可以调用驻留在相同环或较高特权环中的服务。

图 4-35 示出了在环保护机构中的调用程序和数据访问的关系。



(a) 程序间的控制传输　　(b) 数据访问

图 4-35　环保护机构

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 ： 何时放
- 8.2.2 Placement Policy ：放物理内存哪 (NUMA)
- **8.2.3 Replacement Policy ：满了怎么替换**
- **8.2.4 Resident Set Management ：推广至全局性**
- 8.2.5 Clearing Policy
- 8.2.6 Load Control

8.2.1 Fetch Policy(1/1)(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

8.2.2 Placement Policy(1/1)(放置策略)

- Placement Policy: Determines where in real memory a process piece is to reside
  - Important in a nonuniform memory access(NUMA, 非一致存储器访问) system (放置策略对 NUMA 处理器影响大)
  - Not that important in segmentation system
    - Can use first fit, best fit, etc.
  - 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
  - 针对单个进程：重点讨论各种置换方法
- 8.2.4 Resident Set Management
  - 针对所有进程：全局置换
    - 如何评估系统性能和各个进程分配的页帧数？
- 8.2.5 Clearing Policy
- 8.2.6 Load Control

8.2.3 Replacement Policy(1/16)(置换策略)

- Basic Rules of Placement Policy
  - Frame Locking 帧锁定
    - 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

8.2.3 Replacement Policy(2/16)

- Basic Rules of Placement Policy
  - Easy to implement and Efficient 简单高效
  - 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）

8.2.3 Replacement Policy(3/16)

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
   - 主要用于对比评估其它算法
     - 先跑一次得出数据，然后再运行第二次，以此数据和其它算法对比

8.2.3 Replacement Policy(4/16)

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.
     - 为什么这样说？请分组讨论从如何实现 LRU 的角度说明

8.2.3 Replacement Policy(5/16)

- openEuler
  - 采用 LRU 策略
  - 5 个链表记录不同的页类型
    - 非活跃匿名页 LRU 链表
    - 活跃匿名页 LRU 链表
    - 非活跃文件页 LRU 链表
    - 活跃文件页 LRU 链表
    - 不可回收 LRU 链表
  - OS 从非活跃链表尾部开始回收记录

8.2.3 Replacement Policy(6/16)

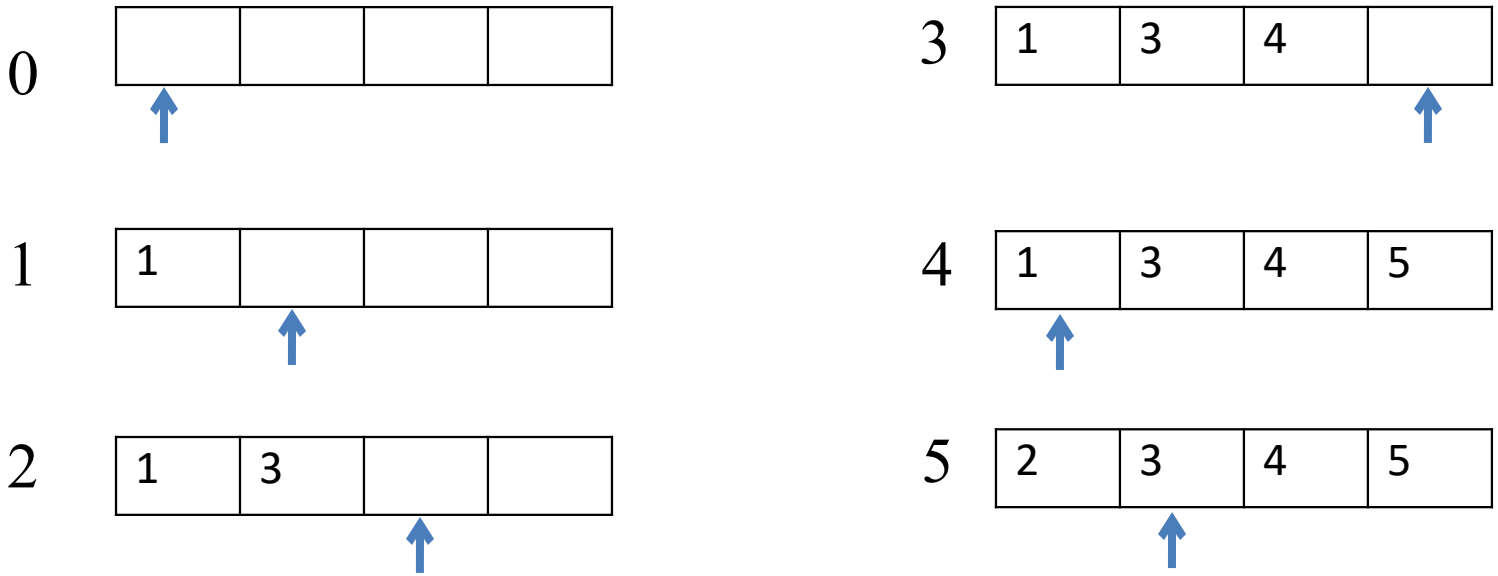
3. First-in, first-out (FIFO，先进先出)
   - Treats page frames allocated to a process as a circular buffer(把分配给进程的页帧看做是一个循环缓冲区)
   - Pages are removed in round-robin(循环) style
   - Simplest replacement policy to implement
   - Page that has been in memory the longest is replaced
   - These pages may be needed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

- 实现：链表
- Belady 现象：物理页帧越多，缺页可能增加

8.2.3 Replacement Policy(7/16)

3. 实现：链表 / 队列 / 堆栈？

访问序列： 1 3 4 5 2 4 ？

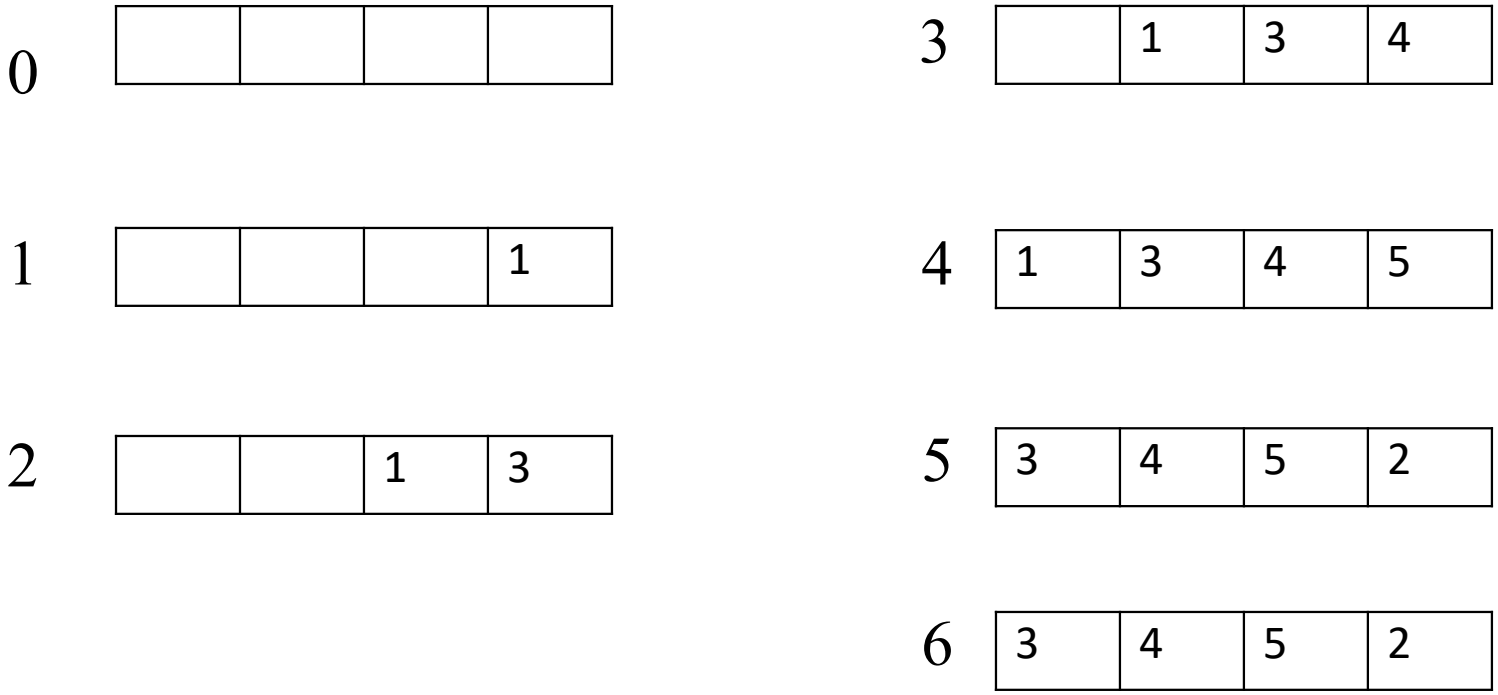


8.2.3 Replacement Policy(8/16)

3. 实现：链表 / 堆栈

队列

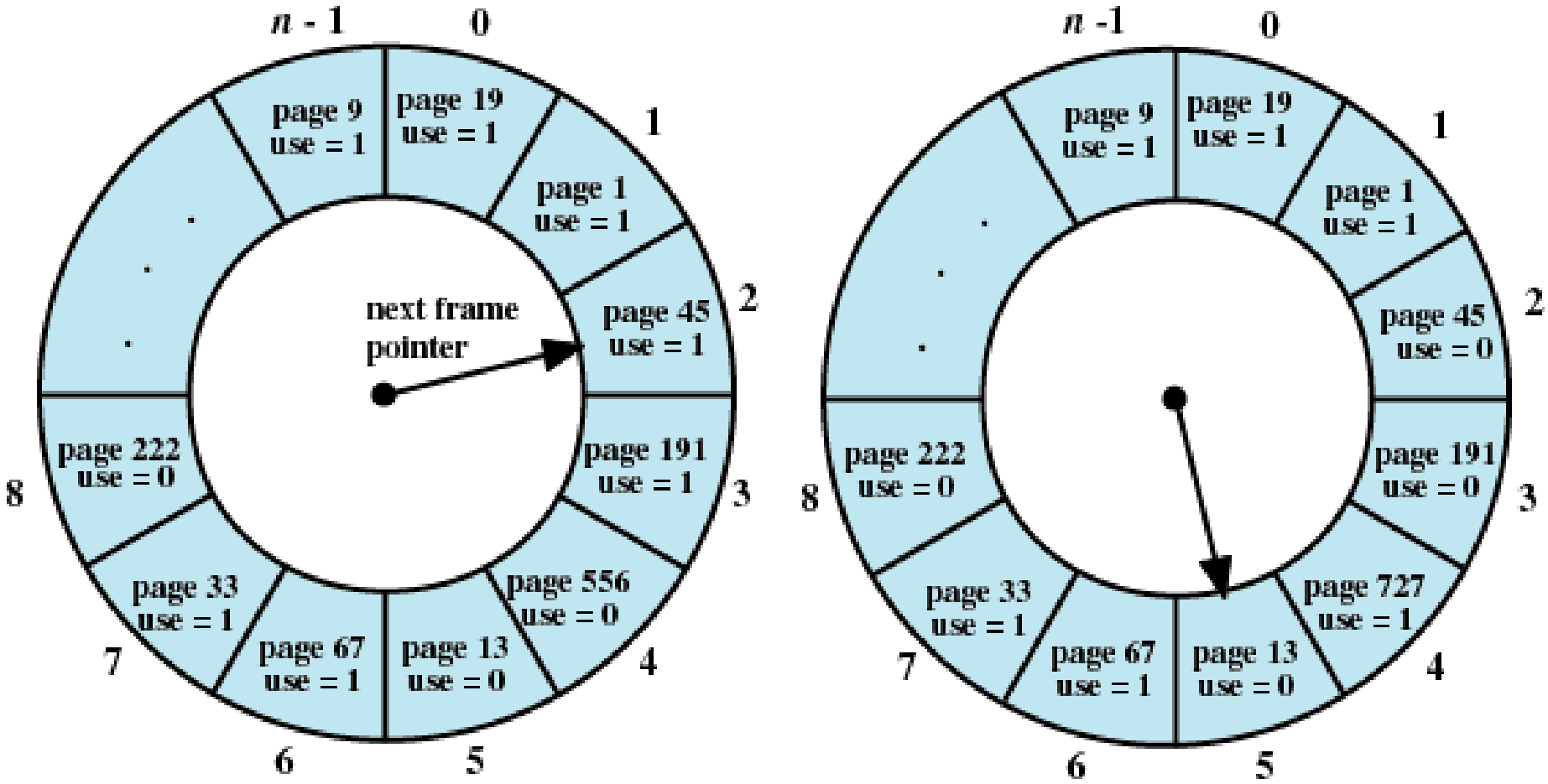
1 3 4 5 2 4



8.2.3 Replacement Policy(9/16)

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
   - Once set use bit to 1 or 0, the pointer move to next position

8.2.3 Replacement Policy(10/16)



(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

8.2.3 Replacement Policy(11/16)

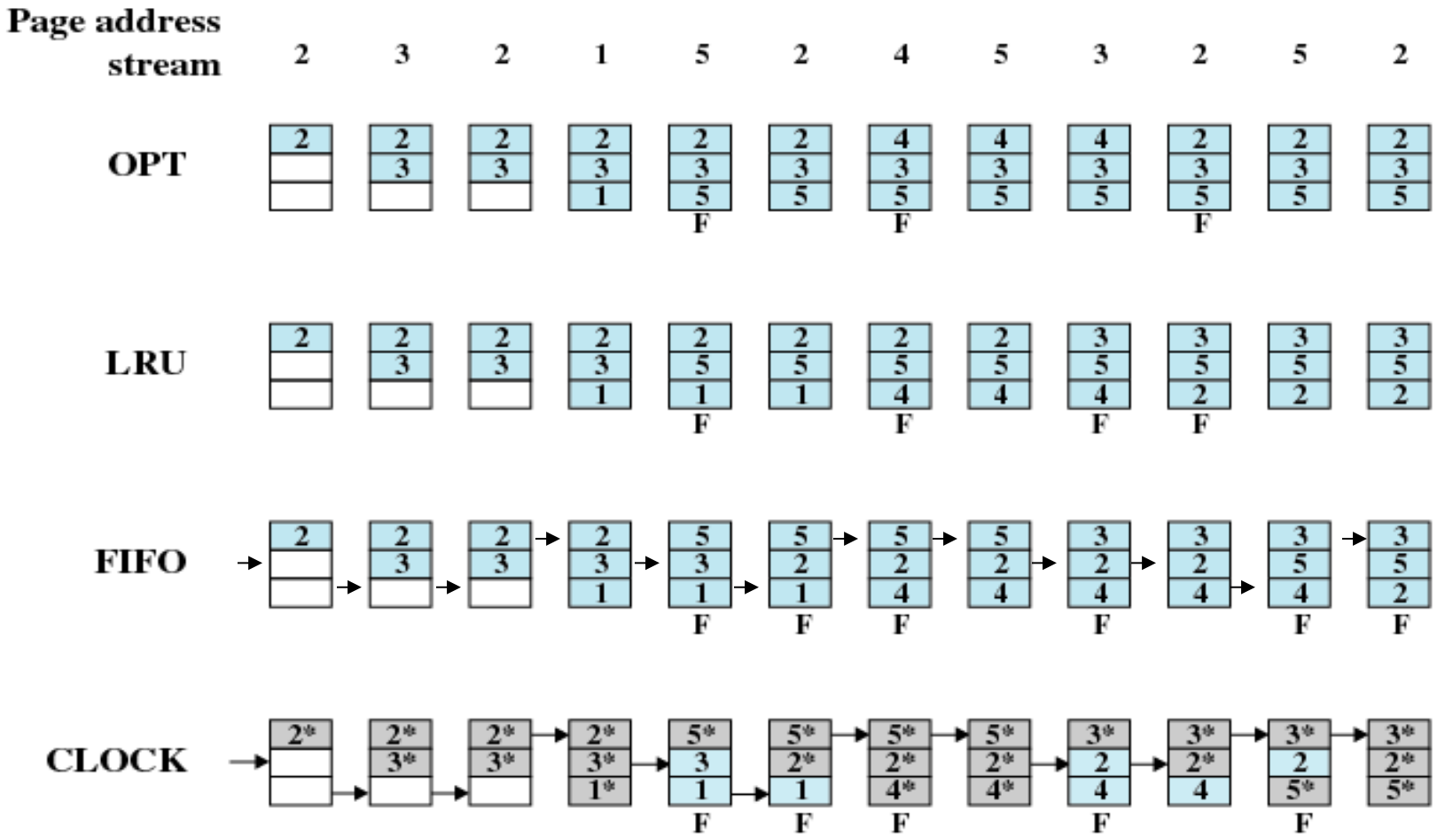


Figure 8.15 Behavior of Four Page-Replacement Algorithms

8.2.3 Replacement Policy(12/16)

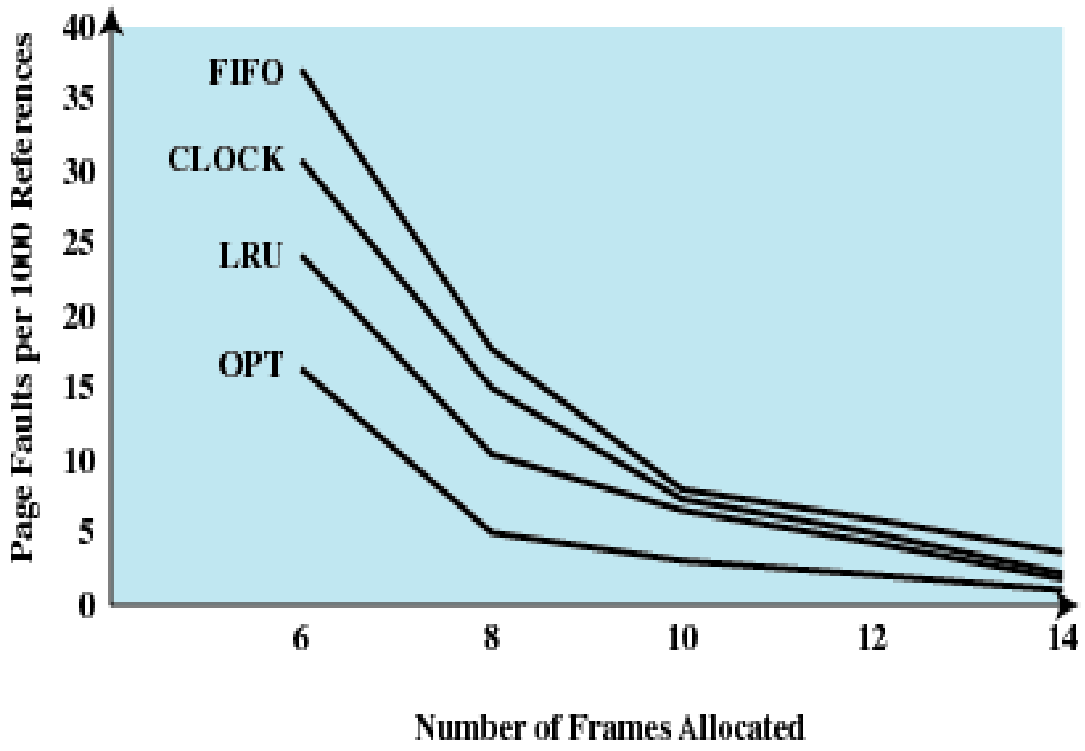


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

8.2.3 Replacement Policy(13/16)

Clock Policy Using Use Bit and Modified Bit 增强二次机会法

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

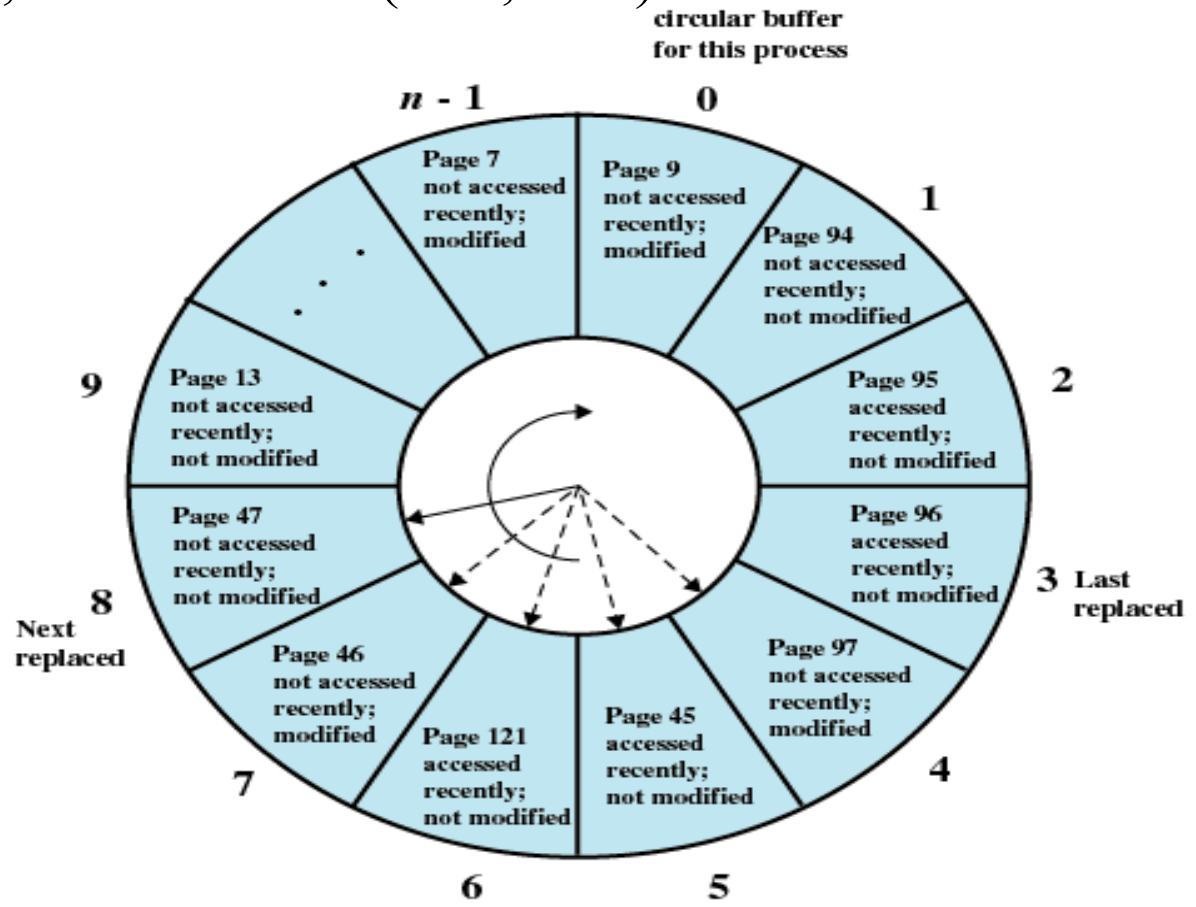


Figure 8.18 The Clock Page-Replacement Algorithm [GOLD89]

8.2.3 Replacement Policy(14/16)

Clock Policy Using Use Bit and Modified Bit 增强二次机会法

1. Scan, if (0,0), replace; if not found jump to 2
2. Rescan, meet first (0,1), replace; during scan, set all used bit to 0; if not found, jump to 3
3. Rescan, set all used bit to 0 and repeat 1 and 2

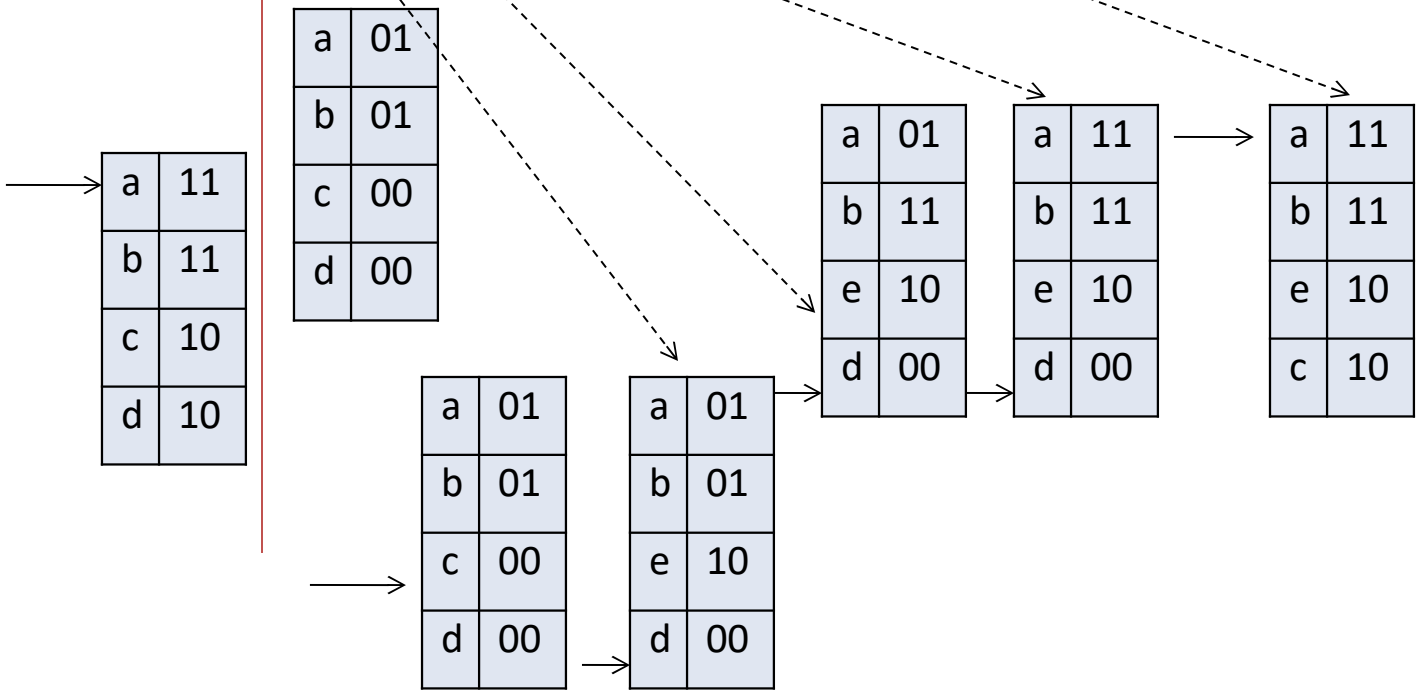
used	modified
0	0
0	1
1	0
1	1

used	modified
replaced	replaced
0	1
1	0
1	1

8.2.3 Replacement Policy(15/16)

prev	c	aw	d	bw	e	b	aw	b	c
a	a	a	a	a	a	a	a	a	a
b	b	b	b	b	b	b	b	b	b
c	c	c	c	c	e	e	e	e	e
d	d	d	d	d	d	d	d	d	c
					F				F

Clock Policy Using Use Bit and Modified Bit
增强二次机会法



8.2.3 Replacement Policy(16/16)

- Page Buffering(页缓冲)
- 1. 维护有一个空闲帧缓冲池：先回空闲帧缓冲，再选中牺牲者，减少时间开销
- 2.Replaced page is added to one of two lists
  - Free page list if page has not been modified (空闲页表) and 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 驻留集管理
- 8.2.5 Clearing Policy
- 8.2.6 Load Control

8.2.4 Resident Set Management(1/10)

两者有关系吗？

- Working Set 工作集
  - 进程 P 正在访问的页面集合称为它的工作集 w(k,t)：在任意时刻 t ，前 k 次访问内存的页面的集合。例如 w(5,7) 即页面 1,2,4,5
- Resident Set 驻留集
  - 目前进程 P 驻留在 OS 分配的 N 个页帧物理内存中的页，如下时刻 7 ，驻留集 2,4,5 。

time	0	1	2	3	4	5	6	7	8	9	10	11
Page tracks	2	3	2	1	5	2	4	5	3	2	5	2
Stack LRU	2	3	2	1	5	2	4	5	3	2	5	2
		2	3	2	1	5	2	4	5	3	2	5
				3	2	1	5	2	4	5	3	3
PF					F		F		F	F		

8.2.4 Resident Set Management(2/10)

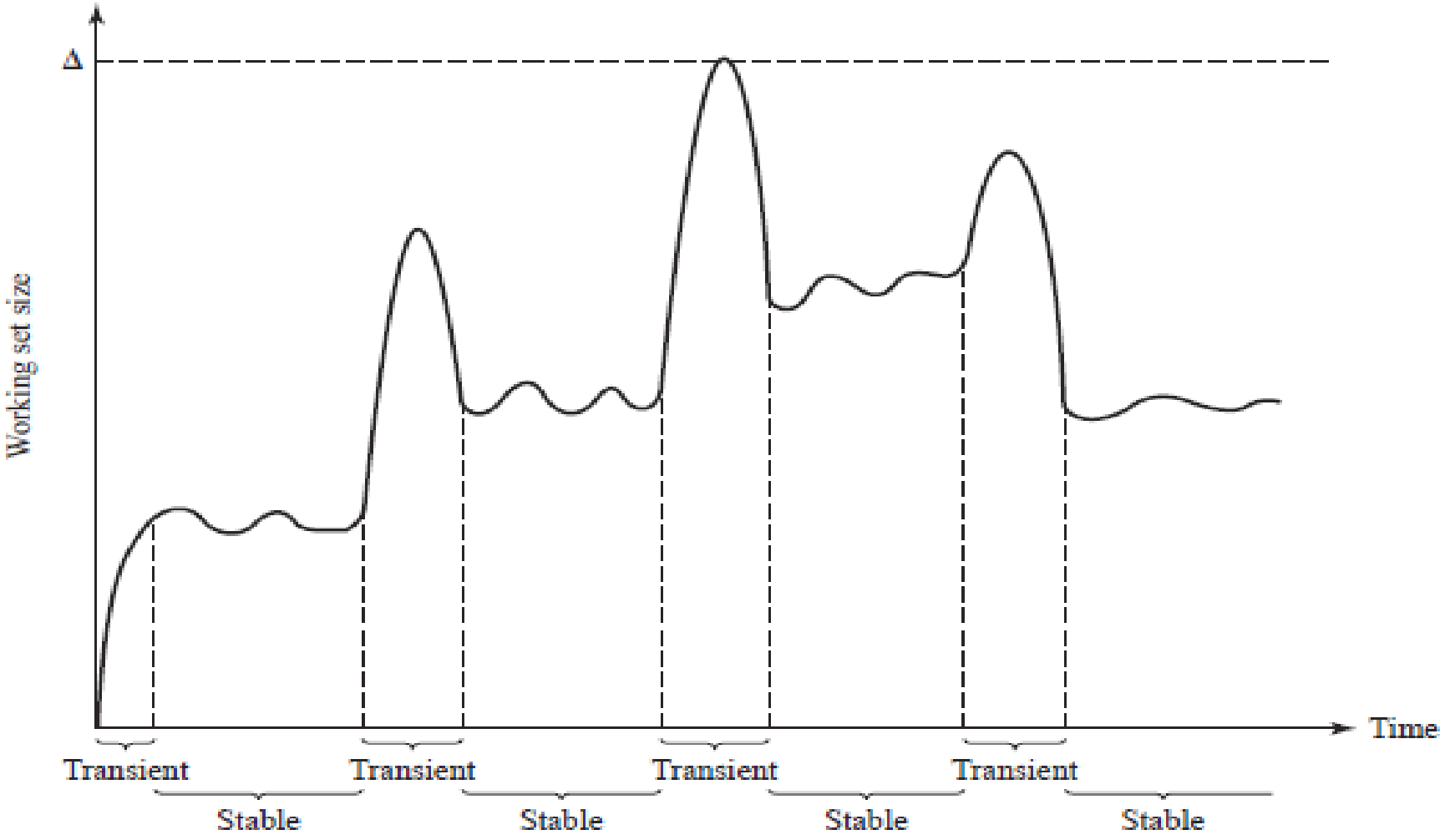


Figure 8.18 Typical Graph of Working Set Size [MAEK87]

8.2.4 Resident Set Management(3/10)

- Resident Set Size(驻留集大小)
  - 这对单个进程意味着什么？对系统意味着什么？
  - 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

8.2.4 Resident Set Management(4/10)

- 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
    - One process gets one more page indicates the other may lose one page if no more spare pages in system

8.2.4 Resident Set Management(5/10)

- 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)
    - too small, high page fault rate (如果太少，导致高缺页率)
    - too large ， too few programs in main memory (如果分配太多，内存中驻留的程序会过少)

8.2.4 Resident Set Management(6/10)

- Variable Allocation, Global Scope(可变分配，全局置换)
  - Easiest to implement/Adopted by many operating systems
  - OS 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
    - No better policy for selecting a victim process, so page buffering is prefered

8.2.4 Resident Set Management(7/10)

- 基于工作集的全局置换算法　k=4(前 k 次)
  - 只要前 K 次不访问就换出，在 K 次内访问过的页驻留

time	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11
Page tracks	5	1	3	2	3	2	1	5	2	4	5	3	2	3	2
page1		☺	☺	☺	☺	x	☺	☺	☺	☺	×				
page2				☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺
page3			☺	☺	☺	☺	☺	☺	×			☺	☺	☺	☺
page4										☺	☺	☺	☺	×	
page5	☺	☺	☺	☺	×			☺	☺	☺	☺	☺	☺	☺	×
PF	F	F	F	F			F	F		F		F			

- k 变化？动态驻留集

8.2.4 Resident Set Management(8/10)

- 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

8.2.4 Resident Set Management(9/10)

- Reevaluate: How?
  - Clue ： Page Fault is very low/high that indicates ?
  - Page fault frequency PFF 缺页频率
  - PFF 受以下因素影响：
    - 置换算法
    - 程序局部性
    - Page size
    - 分配给进程的 Frame 数量
- 缺页率的高低如何定义？
  - 缺页发生的间隔等于一个缺页中断服务的时间时，系统响应效率较好，否则会明显感觉程序运行缓慢
  - 避免频繁变化 (程序有过渡期和稳定期图 8.18)

8.2.4 Resident Set Management(10/10)

- 1. window size = 2
  - 1. 发生 PF 则添加该页到工作集
  - 2. 评估：如果当前 PF 距上次 PF 的两次 PF 间隔 > 时间窗口 2, 则在第二次 PF 发生时，将两次 PF 间隔内没有访问的页移出工作集；如果间隔 <=2 ，仅增加当前缺失页到工作集，不移除

time	0	1	2	3	4	5	6	7	8	9		10	11
Page tracks	2	4	2	1	5	1	2	3	5	3	2	3	2
page1				☺	☺	☺	☺	☺	☺	☺	☺	☺	☺
page2	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺
page3								☺	☺	☺	☺	☺	☺
page4		☺	☺	☺	☺	☺	☺	×					
page5	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺
PF	F	F		F				F					

- 2. VSWS on textbook

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

8.2.5 Cleaning Policy(1/2)(清除策略)

- Write back the modified pages
- 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

8.2.5 Cleaning Policy(2/2)

- Combined two policy with page buffering
- Best approach uses page buffering 页缓冲
  - 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

8.2.6 Load Control(1/4)(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 抖动

- 有什么指标可以用于观察并发度是否过高或过低

8.2.6 Load Control(2/4)

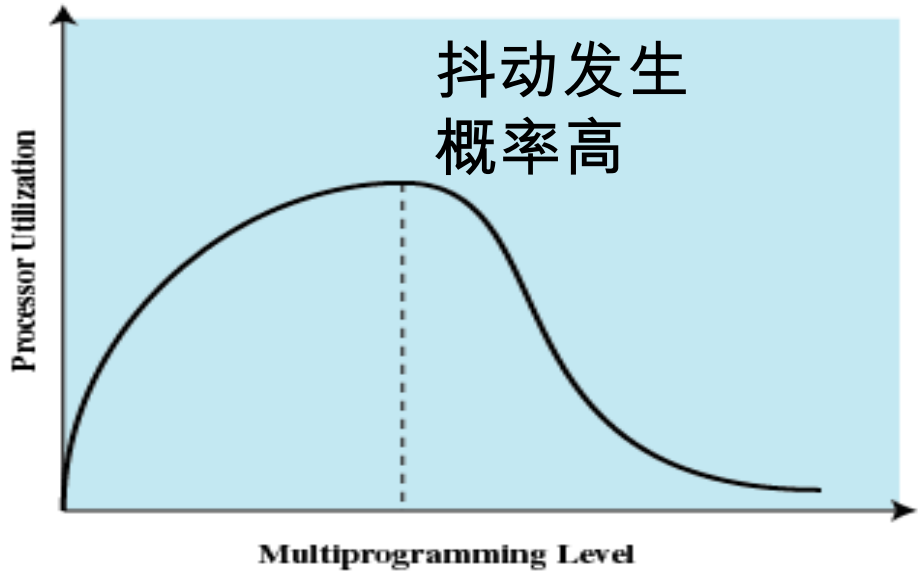


Figure 8.21 Multiprogramming Effects

8.2.6 Load Control(3/4)

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 工作集没有全部驻留

8.2.6 Load Control(4/4)

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，进程被放入就绪队列尾前运行时间很短)

Agenda

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