南京大学
NANJING UNIVERSITY

计算机与操作系统

第八讲 虚拟存储管理

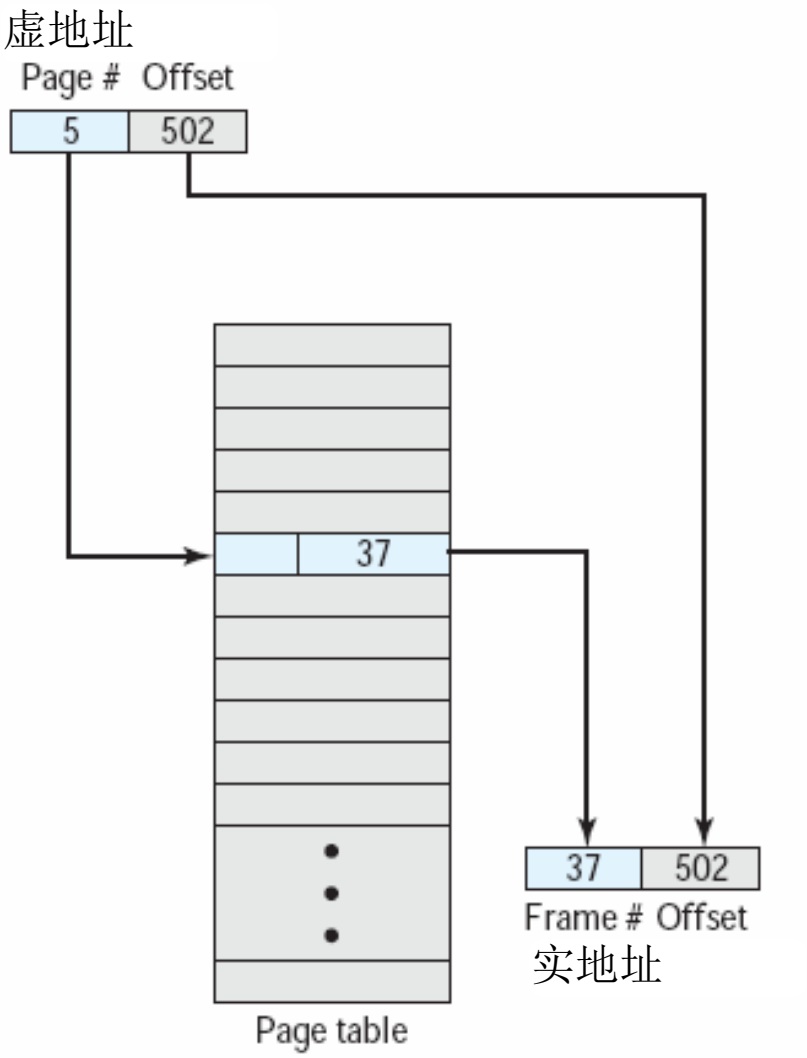
南京大学软件学院

15:59

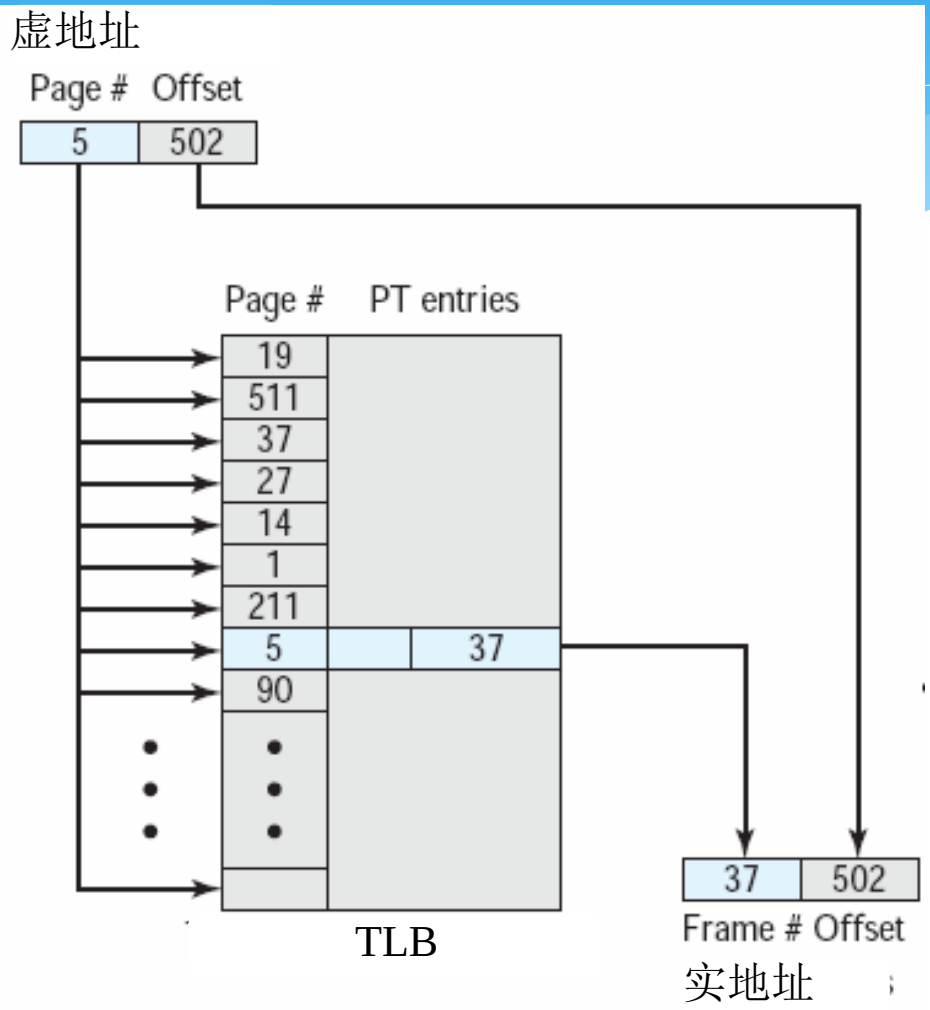
快表-TLB

* 相联存储器
* 快表的格式
* 采用相联存储器后地址转换

页表项的直接映射和关联映射

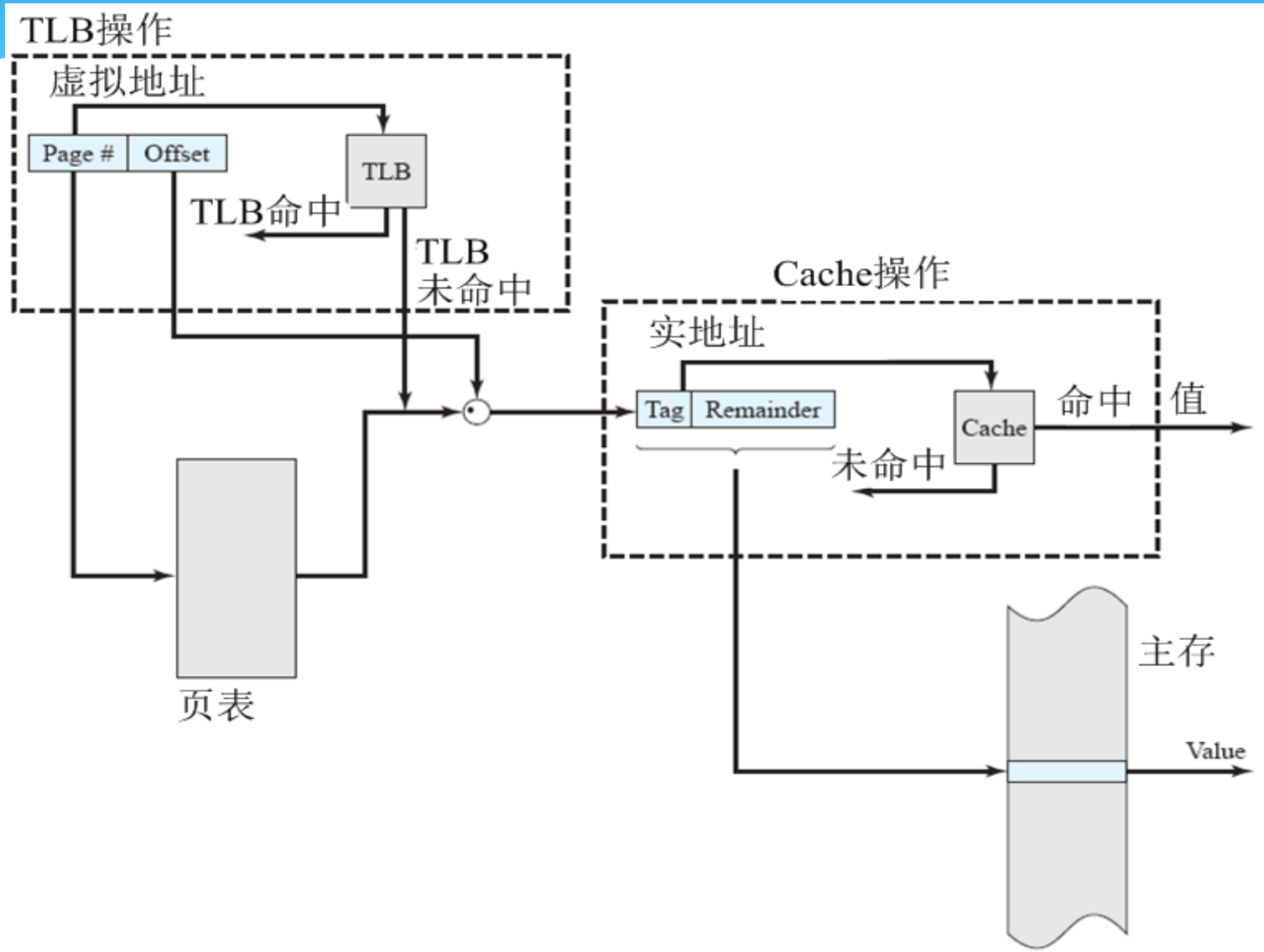


(a) 直接映射

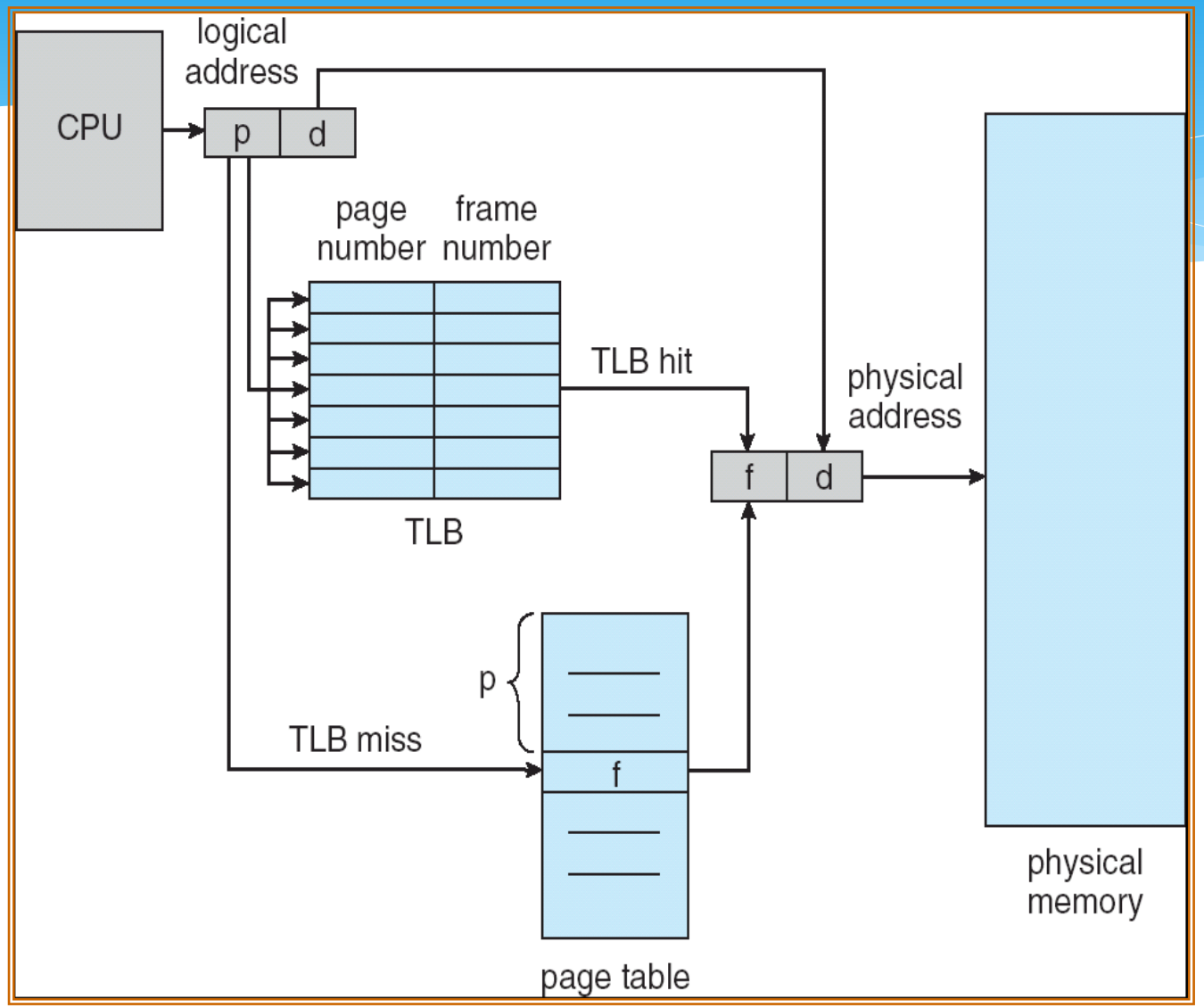


(b) 关联映射

TLB和Cache操作



Paging Hardware With TLB



Effective Access Time

* Associative Lookup = ε time unit
* Assume memory cycle time is f time unit
* Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers
* Hit ratio = α
* **Effective Access Time** (EAT)

$$EAT = (f + \varepsilon)\,\alpha + (2f + \varepsilon)(1 - \alpha)$$
$$= 2f + \varepsilon - \alpha f$$

采用相联存储器的地址转换

假定访问主存时间为100毫微秒，访问相联存储器时间为20毫微秒，相联存储器为32个单元时快表命中率可达90%，按逻辑地址存取的平均时间为：

(100 + 20) × 90% + (100+100+20) × (1-90%) = 130毫微秒

比两次访问主存的时间100毫微秒×2+20 = 200毫微秒下降了三成多。

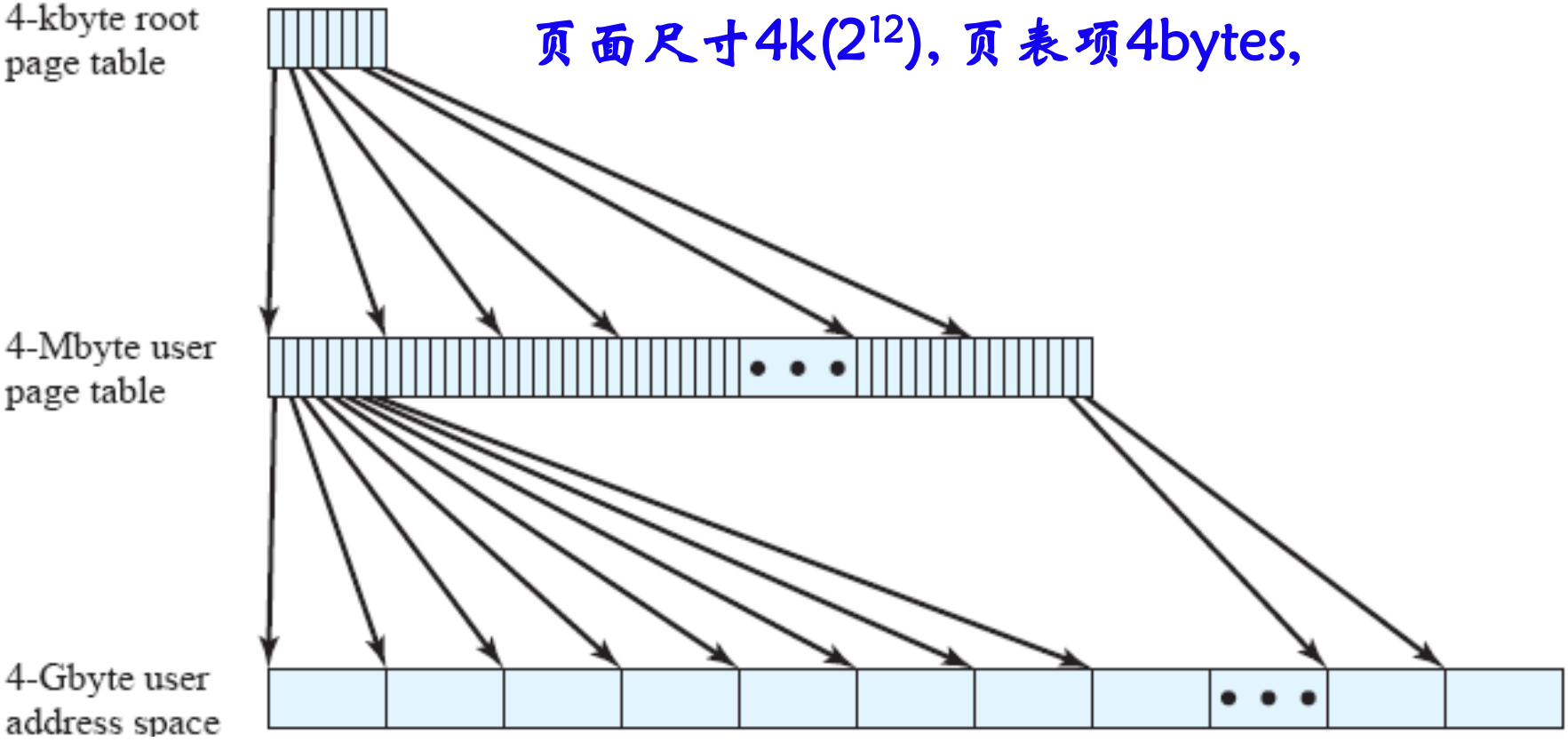
多级页表

* 多级页表的概念
  * 现代计算机普遍支持2^{32}～2^{64}容量的逻辑地址空间，采用分页存储管理时，页表相当大，以Windows为例，其运行的Intel x86平台具有32位地址，规定页面4KB(2^{12})时，那么，4GB(2^{32})的逻辑地址空间由1兆(2^{20})个页组成，若每个页表项占用4个字节，则需要占用4MB(2^{22})连续主存空间存放页表。系统中有许多进程，因此页表存储开销很大。
* 多级页表的具体做法
* 逻辑地址结构
* 逻辑地址到物理地址转换过程

多级页表的概念

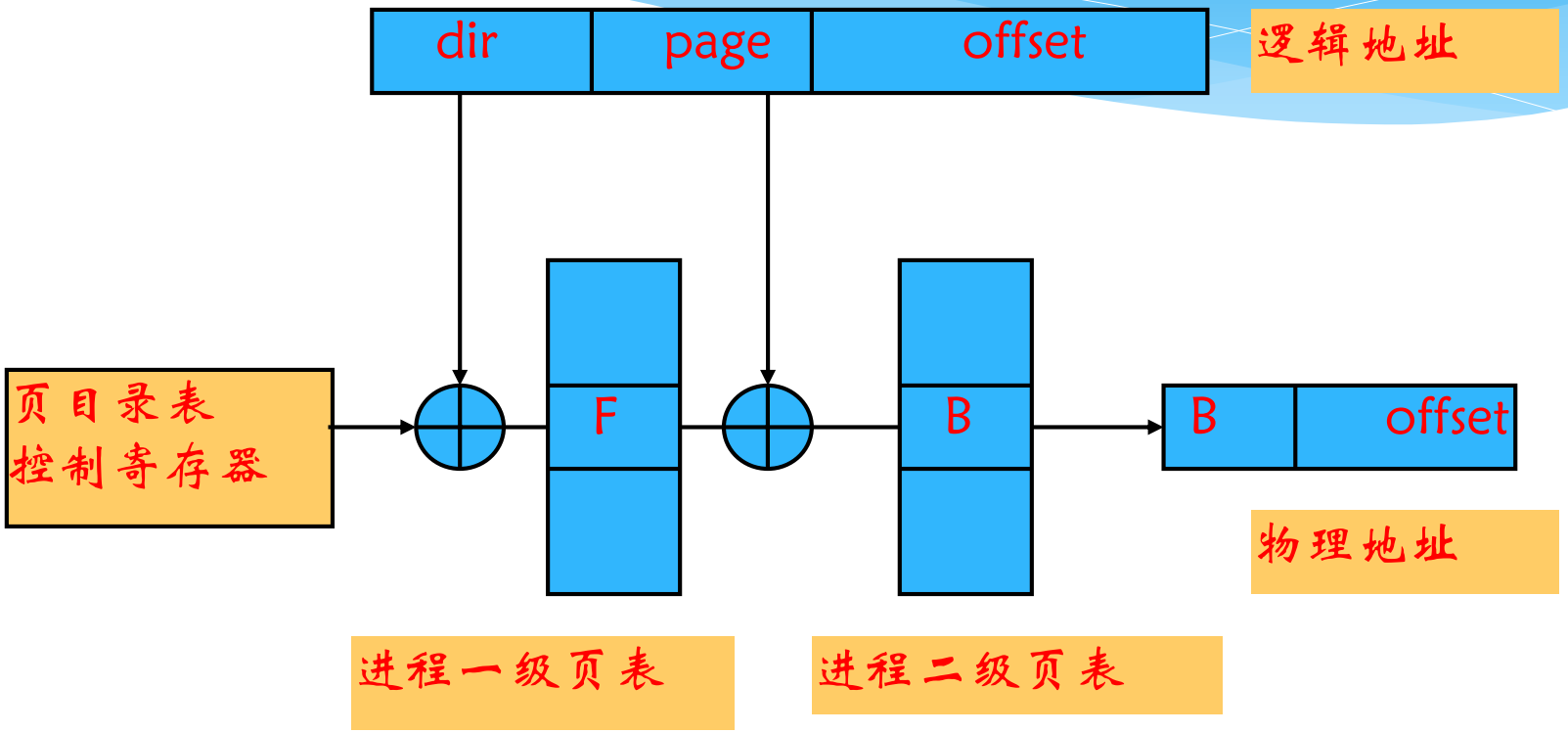
* 系统为每个进程建一张页目录表,它的每个表项对应一个页表页,而页表页的每个表项给出了页面和页框的对应关系,页目录表是一级页表,页表页是二级页表。
* 逻辑地址结构有三部分组成：页目录、页表页和位移

两级页表(32位地址)

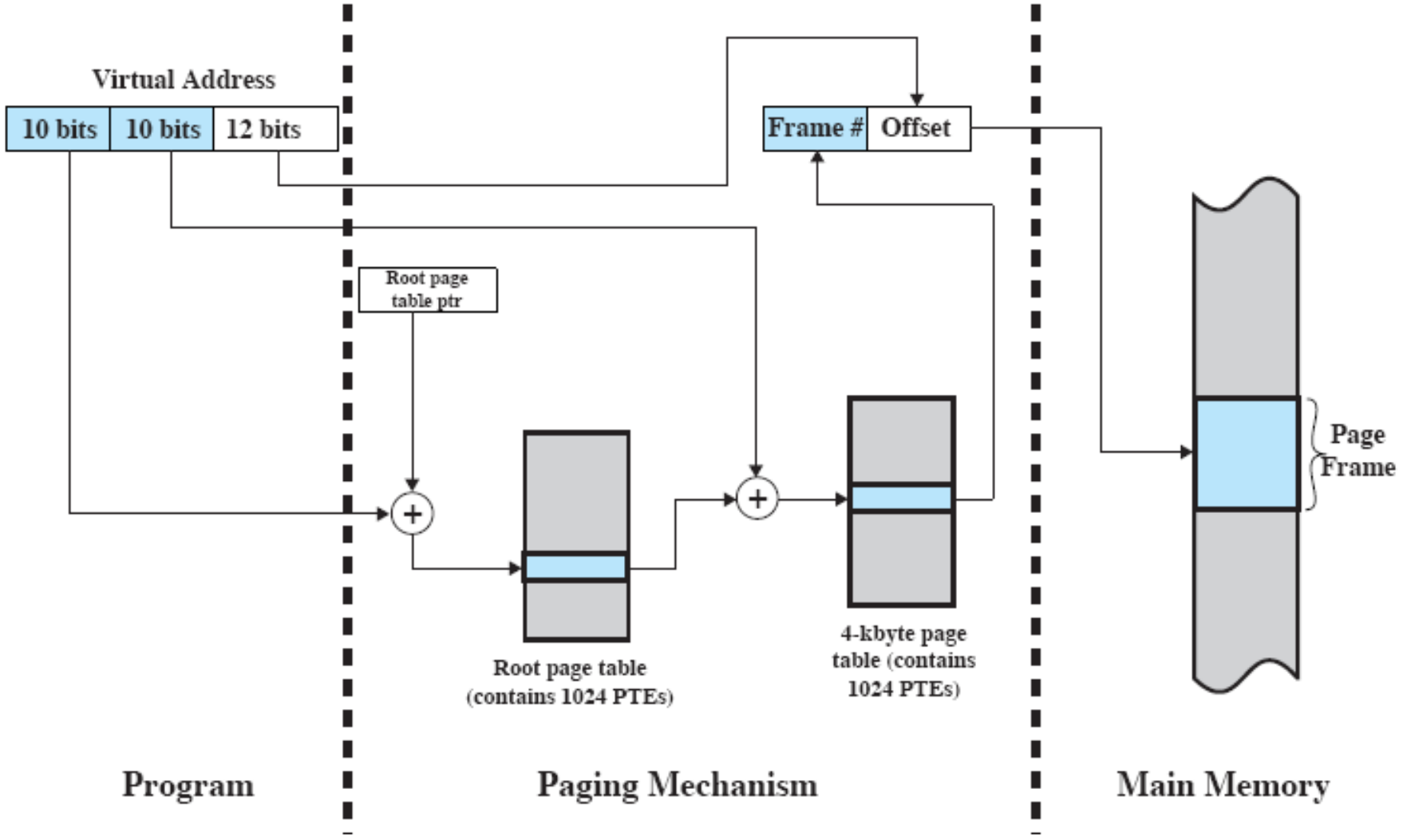


页面尺寸4k(2^{12})，页表项4bytes，

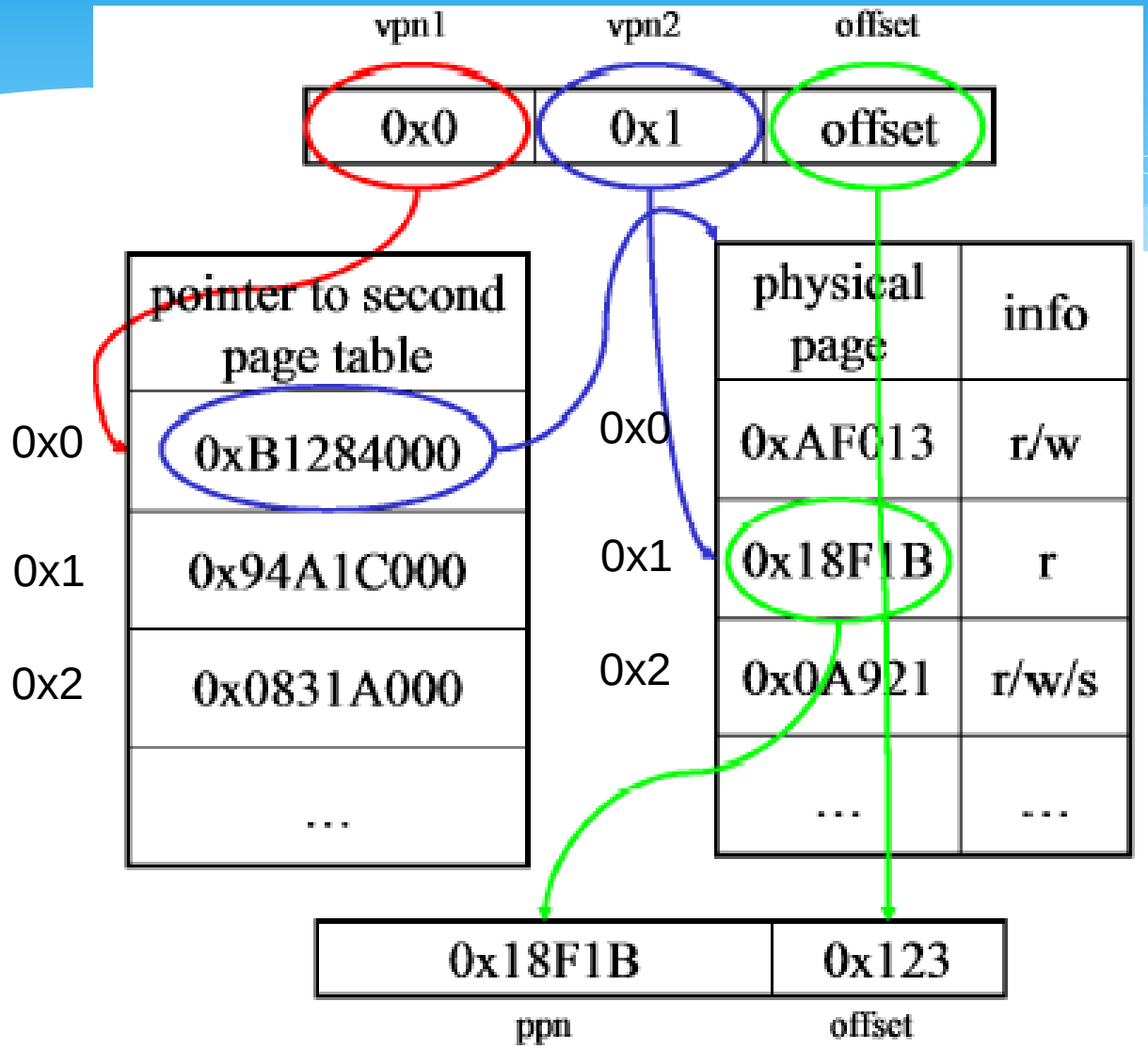
多级页表地址转换过程


dir
page
offset
逻辑地址
页目录表
控制寄存器
F
B
B
offset
物理地址
进程一级页表
进程二级页表

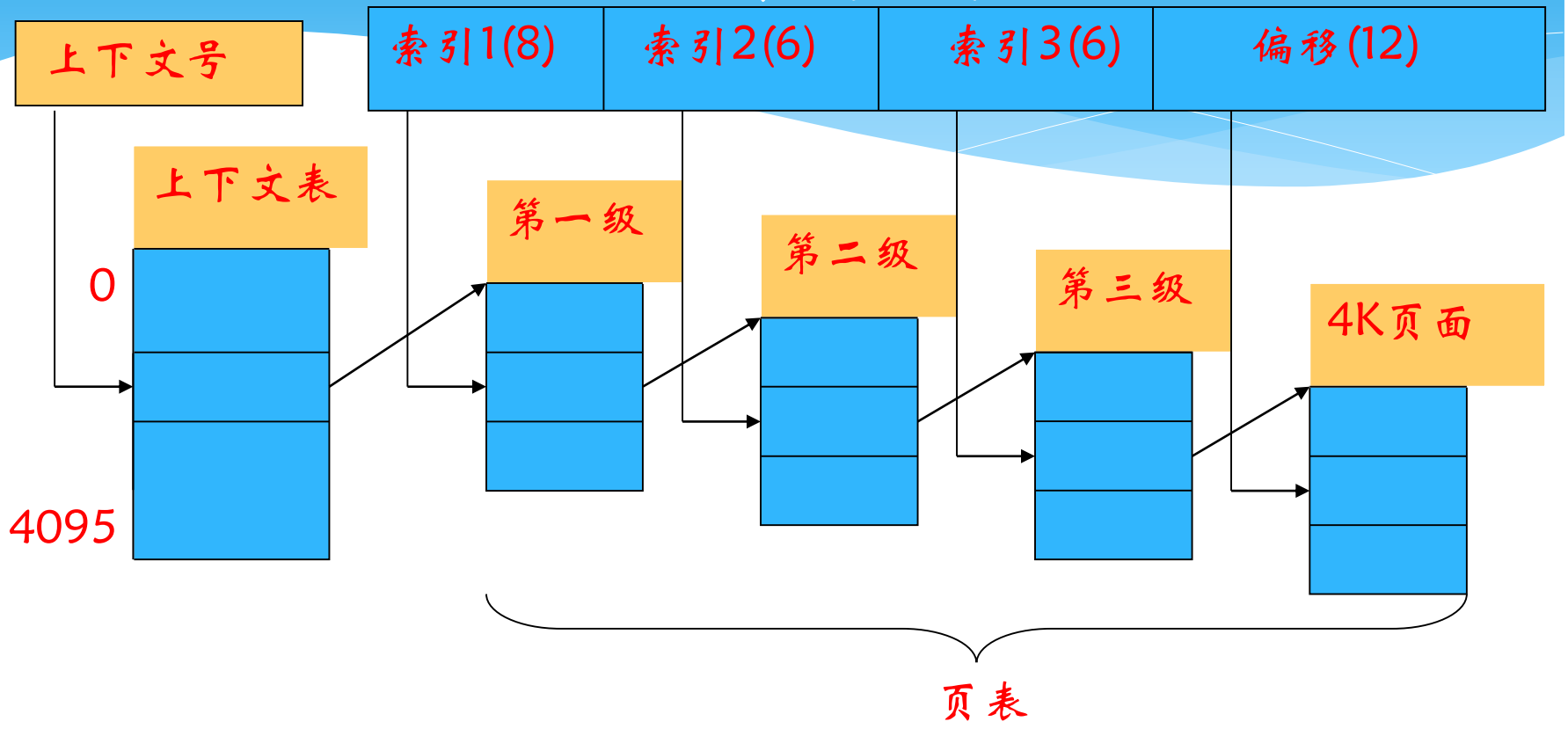
Two-Level Scheme for 32-bit Address



二级页表


vpn1
vpn2
offset
0x0
0x1
offset
pointer to second page table
0x0
0xB1284000
0x1
0x94A1C000
0x2
0x0831A000
...
physical page
info
0x0
0xAF013
r/w
0x1
0x18F1B
r
0x2
0x0A921
r/w/s
...
...
0x18F1B
0x123
ppn
offset

SUN SPARC计算机三级分页结构



问题：增加了寻址时间，在计算机系统中时间与空间总是存在一些矛盾，因此经常会采取折衷的方案，以时间换空间，或者以空间换取时间。

多级页表结构的本质

* 多级不连续导致多级索引。
* 以二级页表为例，用户程序的页面不连续存放，要有页面地址索引，该索引是进程页表；进程页表又是不连续存放的多个页表页，故页表页也要页表页地址索引，该索引就是页目录。
* 页目录项是页表页的索引，而页表页项是进程程序的页面索引。

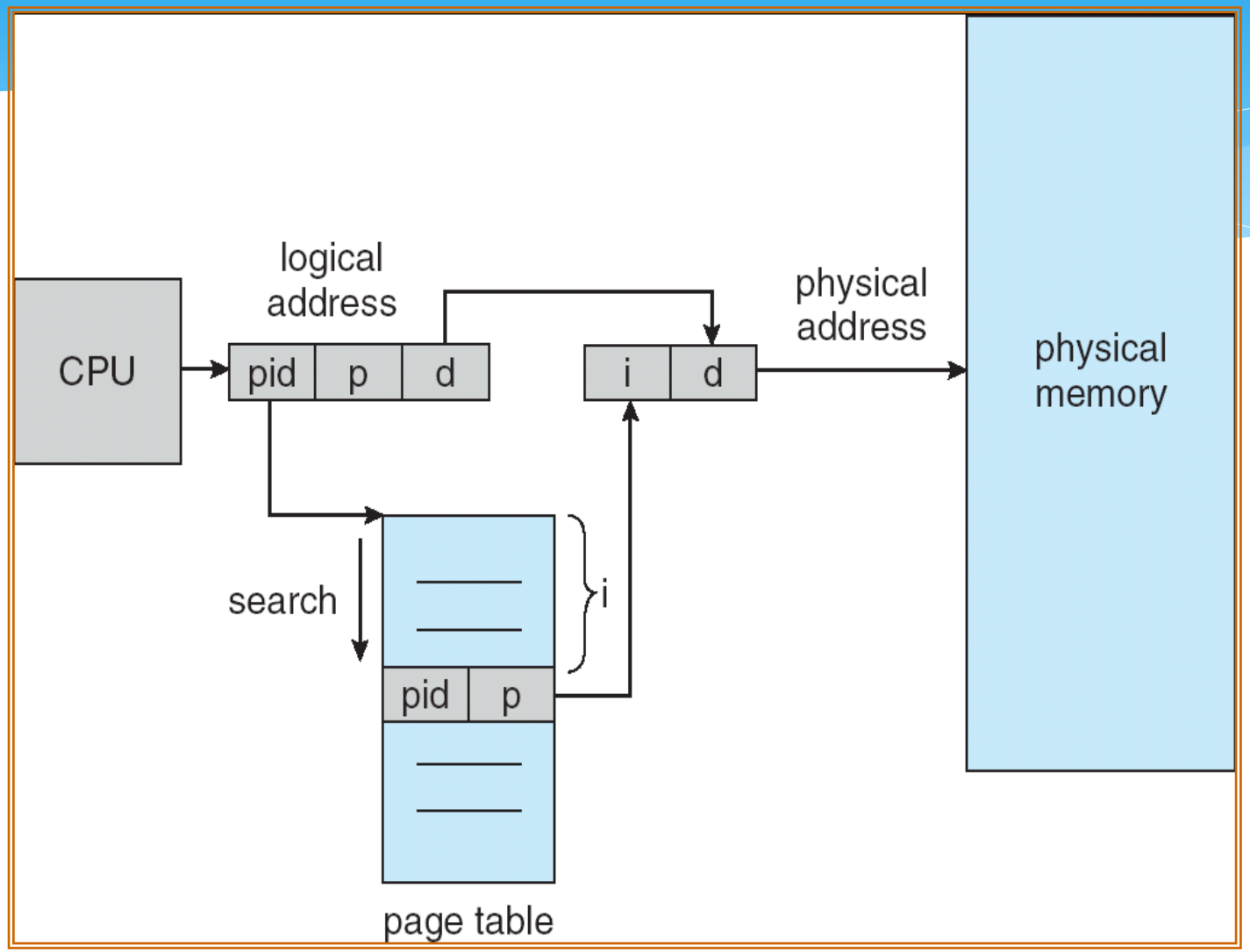
反置页表

* 页表设计的一个重要缺陷是页表的大小与虚拟地址空间的大小成正比
* 在反向页表方法中，虚拟地址的页号部分使用一个简单散列函数映射到哈希表中。哈希表包含一个指向反向表的指针，而反向表中含有页表项。
* 通过这个结构，哈希表和反向表中只有一项对应于一个实存页(面向实存)，而不是虚拟页(面向虚存)。
* 因此，不论由多少进程、支持多少虚拟页，页表都只需要实存中的一个固定部分。
* PowerPC, UltraSPARC, and IA-64

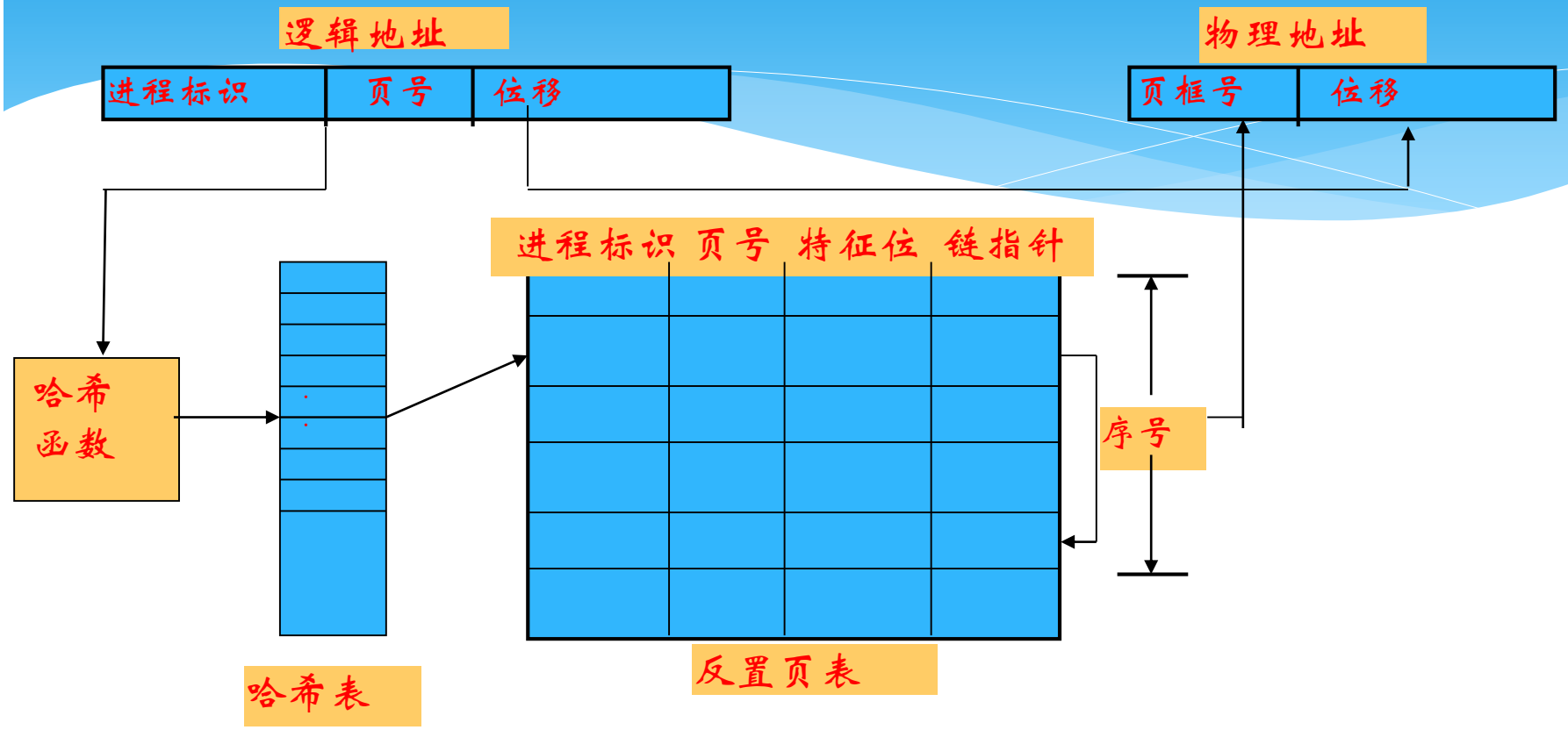
反置页表

* 页号：虚拟地址页号部分。
* 进程标志符：使用该页的进程。页号和进程标志符结合起来标志一个特定的进程的虚拟地址空间的一页。
* 控制位：该域包含一些标记，比如有效、访问和修改，以及保护和锁定的信息。
* 链指针：如果某个项没有链项，则该域为空(允许用一个单独的位来表示)。

反置页表的结构


CPU
logical address
pid
p
d
i
d
physical address
physical memory
search
i
pid
p
page table

反置页表



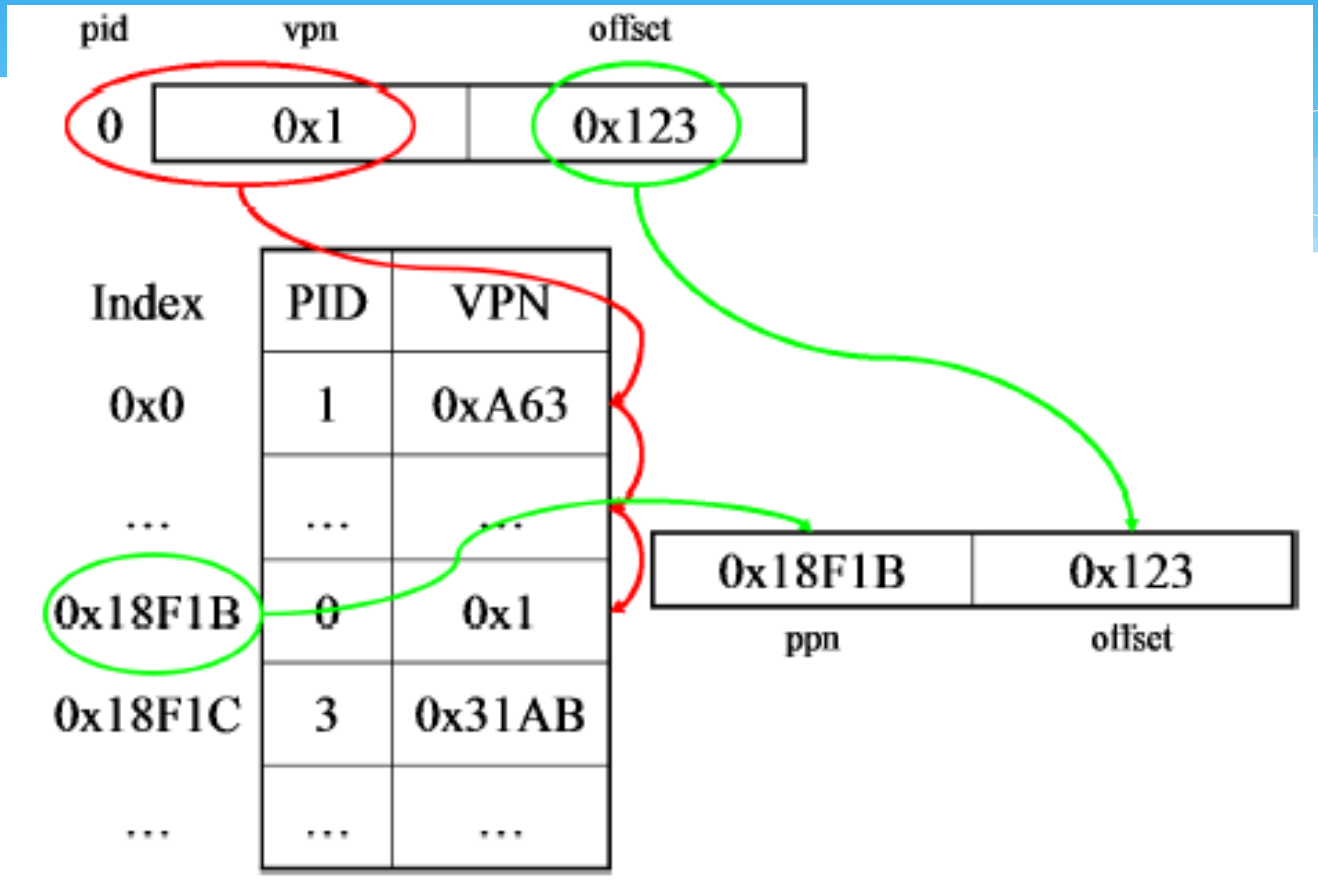
反置页表及其地址转换

反置页表

反置页表地址转换过程如下:

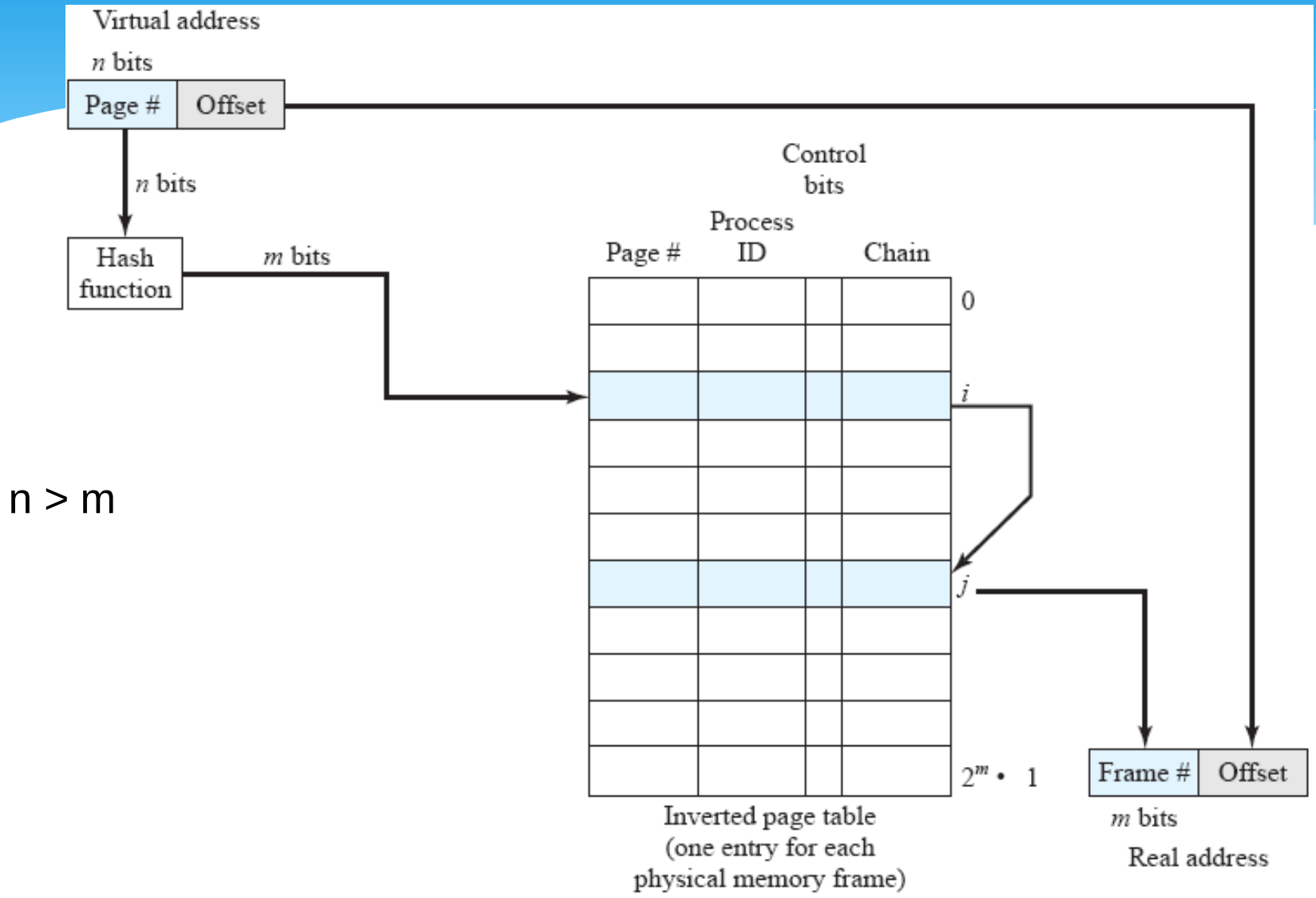
逻辑地址给出进程标识和页号,用它们去比较IPT,若整个反置页表中未能找到匹配的页表项,说明该页不在主存,产生缺页中断,请求操作系统调入;否则，该表项的序号便是页框号,块号加上位移,便形成物理地址。

线性反置页表



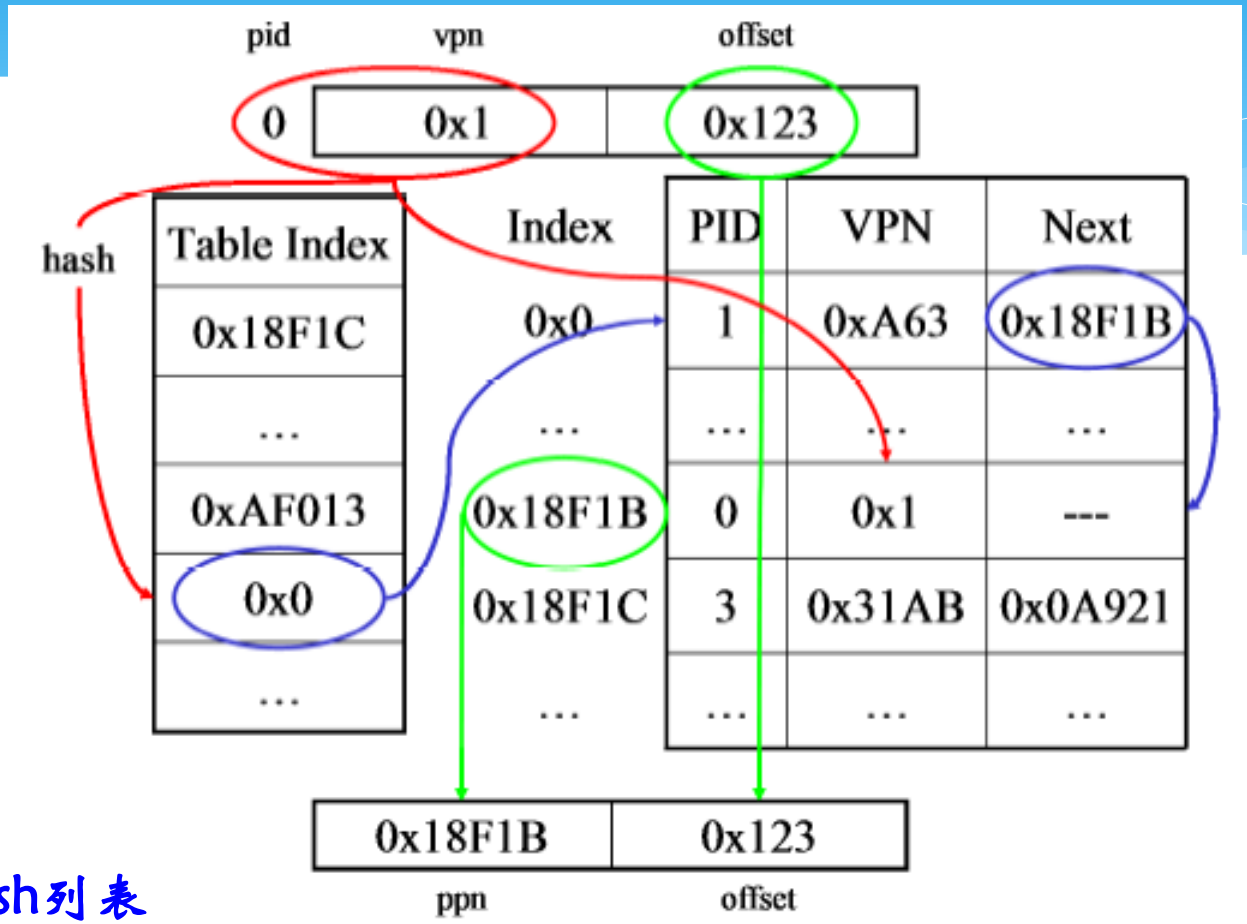
linear inverted page table.

反置页表


Virtual address
n bits
Page #
Offset
n bits
Hash function
m bits
Control bits
Process ID
Page #
Chain
0
i
j
2^m • 1
Inverted page table (one entry for each physical memory frame)
Frame #
Offset
m bits
Real address

n > m

哈希线性反置页表



使用Hash列表

页表的结构称为“反向”是因为它使用帧号而不是虚拟页号来索引页表项

第八讲 虚拟存储管理

* 8.1 虚拟存储概念
* 8.2 虚拟分页技术
* 8.3 虚拟分段技术
* 8.4 虚拟分页的操作系统软件

8.1 虚拟存储概念

本主题教学目标

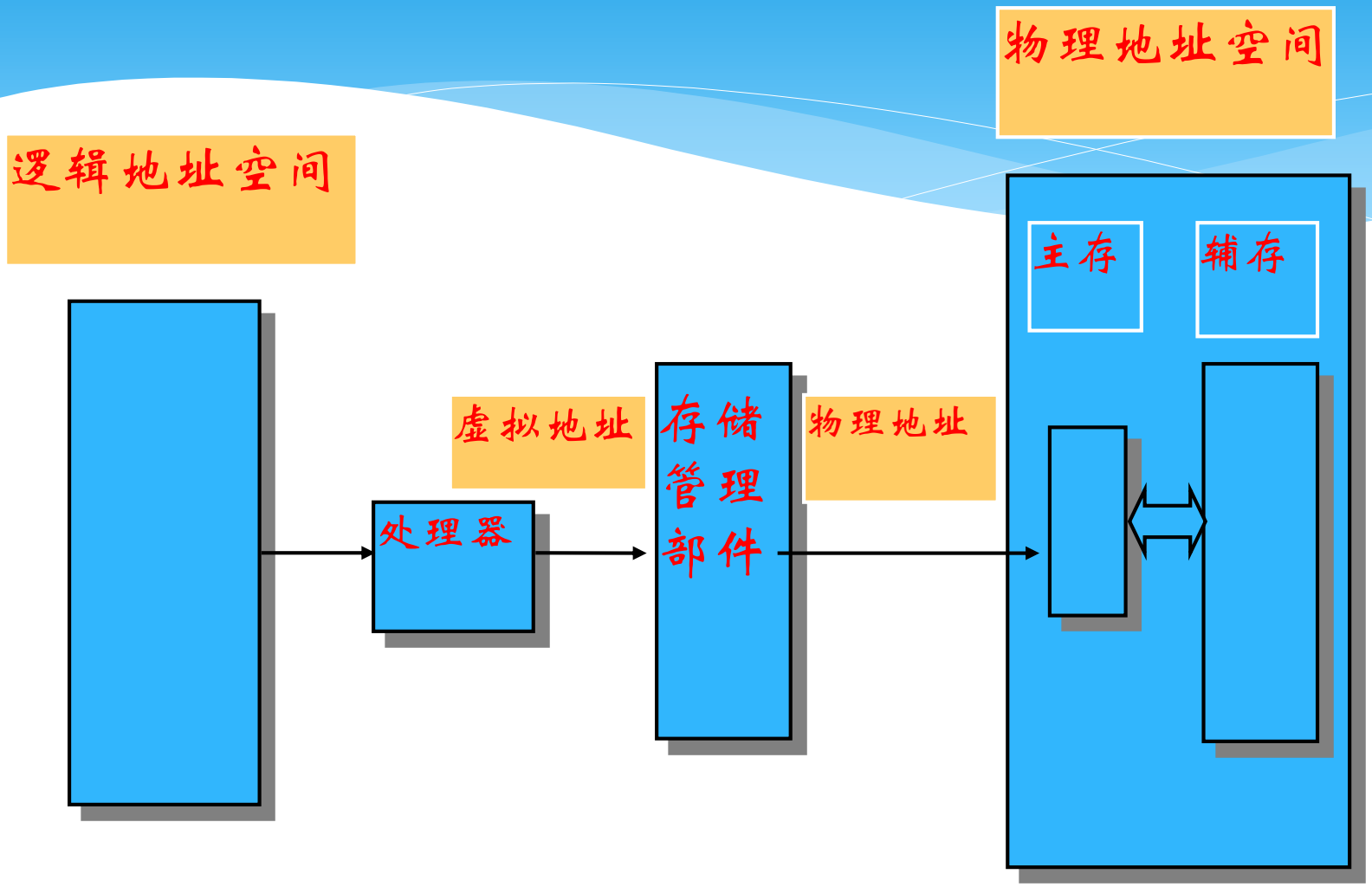
1. 了解虚拟存储管理的基本特点
2. 掌握页式虚拟存储管理
3. 掌握段式虚拟存储管理
4. 掌握页面调度策略与算法
5. 了解Pentium的地址转换机构

Virtual Memory

* Introduction to VM
* Paging
* Segmentation
* OS Software for Paging
* Pentium Example

8.1 虚拟存储概念

8.1 虚拟存储概念



虚拟存储的意义

* 虚拟存储器可定义如下：在具有层次结构存储器的计算机系统中，采用自动实现部分装入和部分替换功能，能从逻辑上为用户提供一个比物理主存容量大得多的，可寻址的一种“主存储器”
* 实际上虚拟存储器对用户隐蔽可用物理存储器的容量和操作细节，它的容量与物理主存大小无关，而受限于计算机的地址结构及可用的磁盘容量，如Intel x86的地址线是32位，则程序可寻址范围是4GB，Windows便为应用进程提供一个4GB的逻辑主存

区别：虚存管理与对换技术

* 虚存管理与对换技术虽说都是在主存和磁盘之间交换信息，但却有很大区别，
  * 对换以进程为单位，当它所需主存大于当前系统拥有量时，无法被对换进主存工作
  * 而虚存管理以页或段为单位，即使进程所需主存大于当前系统拥有的主存总量，仍然能正常运行，因为，系统可将其他进程的一部分页或段换出到磁盘上

硬件和控制结构

* 进程中的所有存储器访问都是逻辑地址，这些逻辑地址在运行时动态地被转换成物理地址
  * 这意味着一个进程可以被换入或换出主存，使得进程在执行过程中的不同时刻，占据主存中的不同区域
* 一个进程可以划分成许多块(页或段)，在执行过程中，这些块不需要连续地位于主存中
  * 动态运行时地址转换和页表或段表机制使这一点成为可能

程序的执行

* 当新进程放入内存时，操作系统仅读取包含程序开始处的一个或者几个块，进程执行中的任何时候都在主存中的部分被定义成进程的常驻集(Resident set)
* 当进程执行时，只要所有的存储器访问都是要访问常驻集中的单元，执行就可以顺利进行
* 如果处理器需要访问一个不在主存中的逻辑地址，则会产生一个中断，说明产生了内存访问故障
  * 操作系统把被中断的进程置于阻塞状态
  * 操作系统产生一个磁盘I/O读请求
  * 产生I/O请求后，在执行磁盘I/O期间，操作系统可以调度另一个进程运行
  * 一旦需要的块被取进主存，则产生一个I/O中断，操作系统把由于缺少块而被阻塞的进程置回就绪态

部分装入的优点

* 在主存中保留多个进程
  * 由于对任何特定的进程都仅仅装入它的某些块，因此就有足够的空间来放置更多的进程，因此，在任何时刻这些进程中可能至少有一个处于就绪态，于是处理器得到了更有效的利用
* 进程可以比主存的全部空间大
  * 如果没有这种方案，程序员就必须清楚地知道有多少主存空间可用，如果编写的程序太大，程序员就必须把程序分成块，按照某种覆盖策略分别加载
  * 通过基于分页或分段的虚拟内存，这项工作可以由操作系统和硬件完成，操作系统在需要时，自动把进程块装入主存

存储的类型

* 实存储器
  * 主存
* 虚拟存储器
  * 磁盘上的存储
  * 虚存允许更有效的多道程序设计，并解除了用户与主存之间没有表的紧密约束

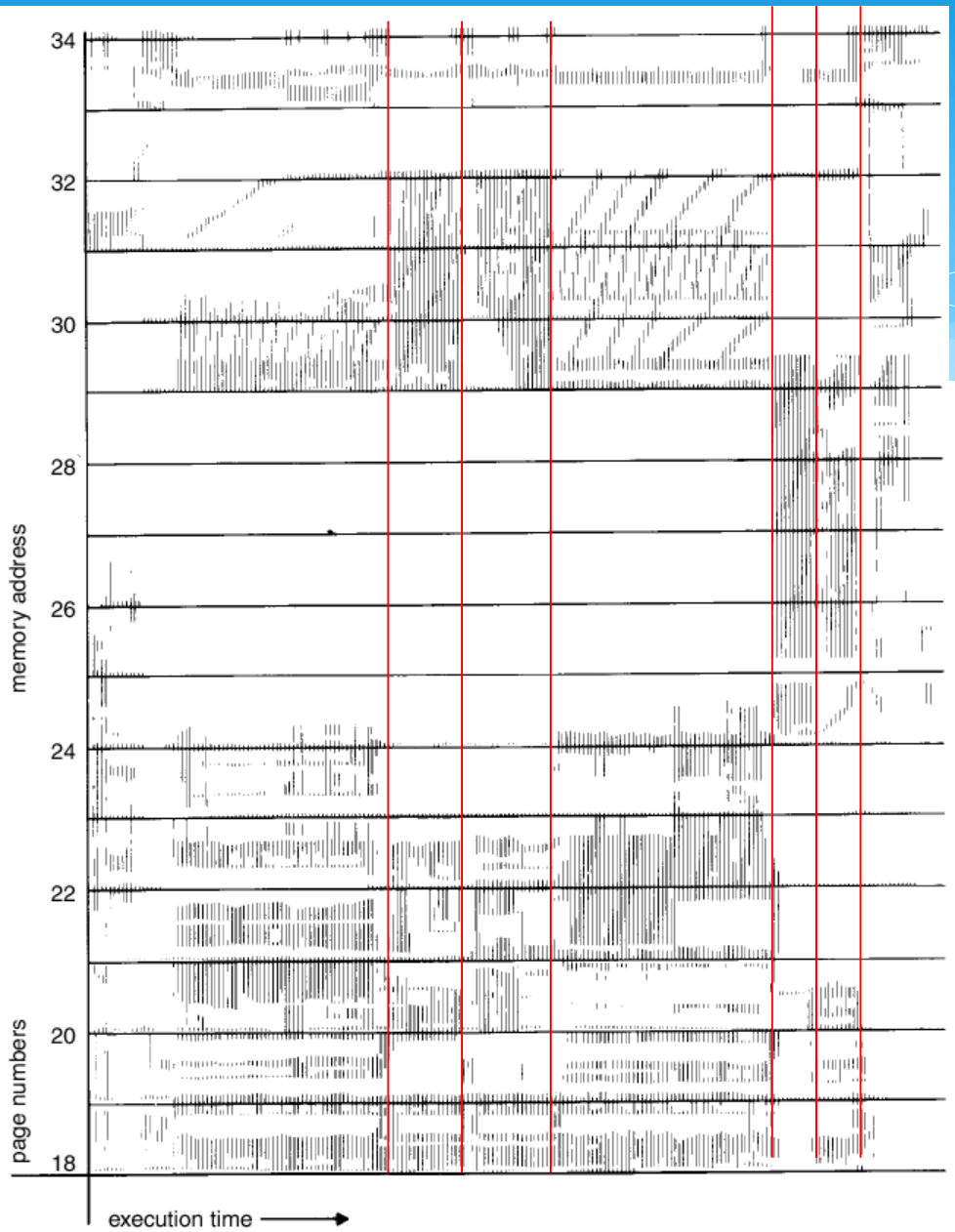
抖动

* 如果一块正好将要被用到之前扔出，操作系统有不得不很快把它取回来，太多的这类操作会导致一种称为系统抖动的情况
* 在处理缺页中断期间，处理器的大部分时间都用于交换块，而不是用户进程的执行指令

程序局部性原理(1)

* 指程序在执行过程中的一个较短时间内，所执行的**指令地址或操作数地址分别局限于一定的存储区域中**。又可细分时间局部性和空间局部性
* 早在1968年P. Denning研究程序执行时的局部性原理，对此进行研究的还有Knuth(分析一组学生的Fortran程序)、Tanenbaum (分析操作系统的过程)、Huck(分析通用科学计算程序)，发现程序和数据的访问都有聚集成群的倾向
* 某存储单元被使用，其相邻存储单元很快也被使用(称空间局部性spatial locality)，
* 或者最近访问过的程序代码和数据，很快又被访问(称时间局部性temporal locality)

分页下的运行情况


34
32
30
28
26
24
22
20
18
memory address
page numbers
execution time

程序局部性原理(2)

* (1) 程序中只有少量分支和过程调用，存在很多顺序执行的指令
* (2) 程序含有若干循环结构，由少量代码组成，而被多次执行
* (3) 过程调用的深度限制在小范围内，因而，指令引用通常被局限在少量过程中
* (4) 涉及数组、记录之类的数据结构，对它们的连续引用是对位置相邻的数据项的操作
* (5) 程序中有些部分彼此互斥，不是每次运行时都用到

* 经验与分析表明，程序具有局部性，进程执行时没有必要把全部信息调入主存，只需装入一部分的假设是合理的，部分装入的情况下，只要调度得当，不仅可正确运行，而且能在主存中放置更多进程，充分利用处理器和存储空间。

虚拟内存的技术需要

* 必须有对所采用的分页或分段方案的硬件支持
* 操作系统必须有管理页或者段在主存和辅助存储器之间移动的软件。

8.2 虚拟分页技术

Paging

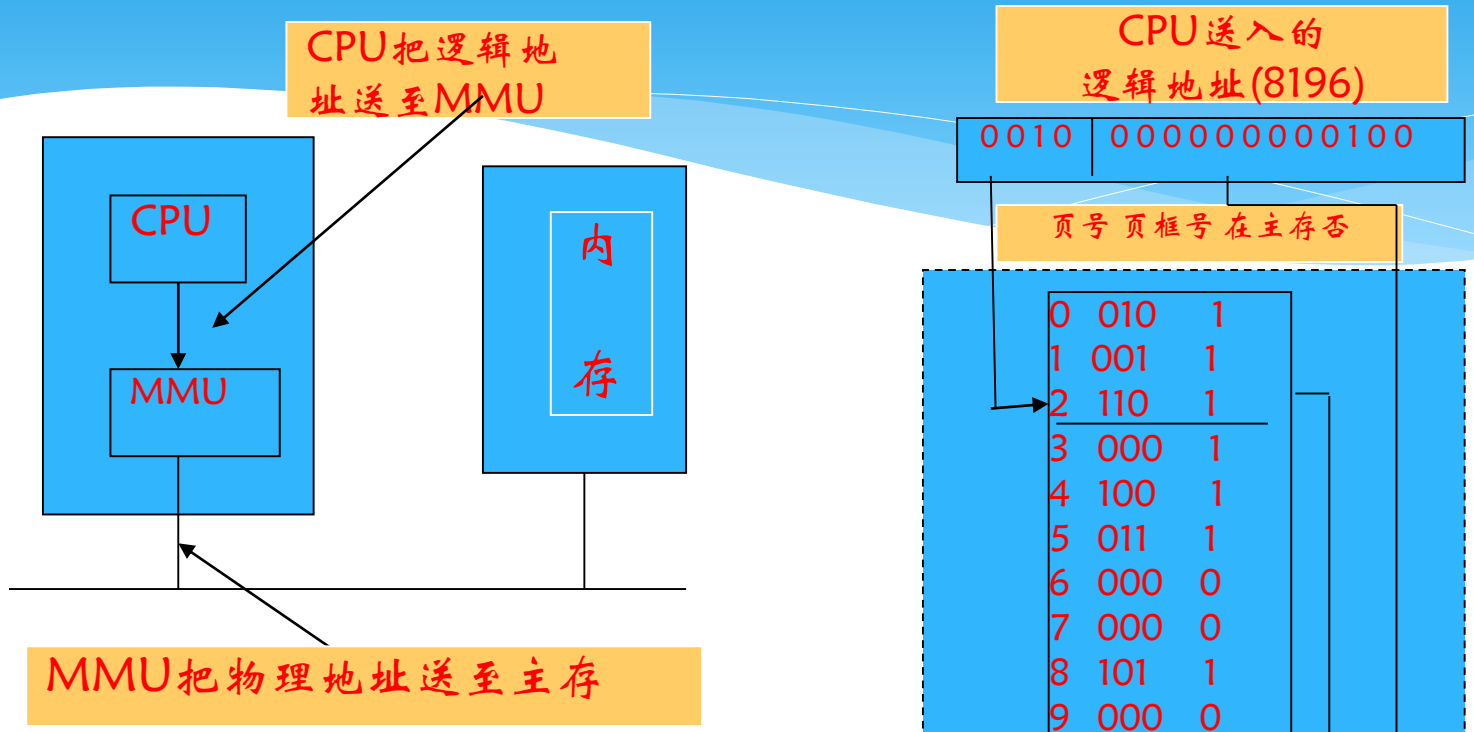
* Each process has its own page table
* Each page table entry contains the frame number of the corresponding page in main memory
* A bit is needed to indicate whether the page is in main memory or not
* Another modify bit is needed to indicate if the page has been altered since it was last loaded into main memory
* If no change has been made, the page does not have to be written to the disk when it needs to be swapped out

分页式虚拟存储系统

分页式虚拟存储系统的硬件支撑(1)

主存管理单元MMU完成逻辑地址到物理地址的转换功能，它接受逻辑地址作为输入，物理地址作为输出，直接送到总线上，对主存单元进行寻址。

分页式虚拟存储系统的硬件支撑(2)



MMU的位置、功能和16个4KB页面情况下MMU的内部操作

MMU送出的物理地址

MMU主要功能

(1)管理硬件页表基址寄存器

(2)分解逻辑地址

(3)管理快表TLB

(4)访问页表

(5)发出缺页中断或越界中断，并将控制权交给内核存储管理处理

(6)设置和检查页表中各个特征位

缺页中断处理的过程

步1 阻塞请求缺页的进程

步2 根据页号查外页表，找到该页存放的磁盘物理地址

步3 查看主存是否有空闲页框，如有则找出一个，修改主存管理表和相应页表项内容，转步6

步4 如主存中无空闲页框，按替换算法选择淘汰页面，检查它曾被写过或修改过吗？若未则转步6；若是则转步5

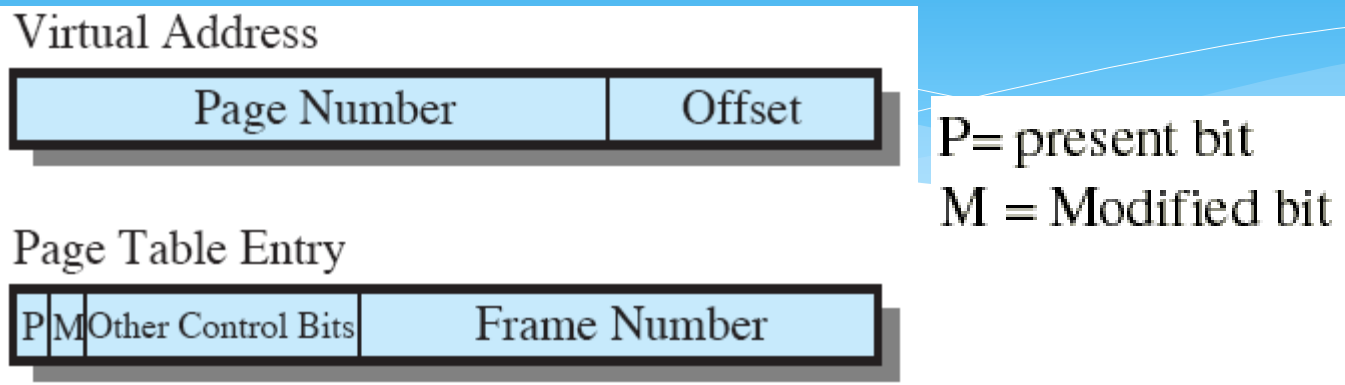
步5 该淘汰页面被写过或修改过，则把它的内容写回磁盘原先位置

步6 进行调页，把页面装入主存所分配的页框中，同时修改进程页表项

步7 返回进程断点，重新启动被中断的指令

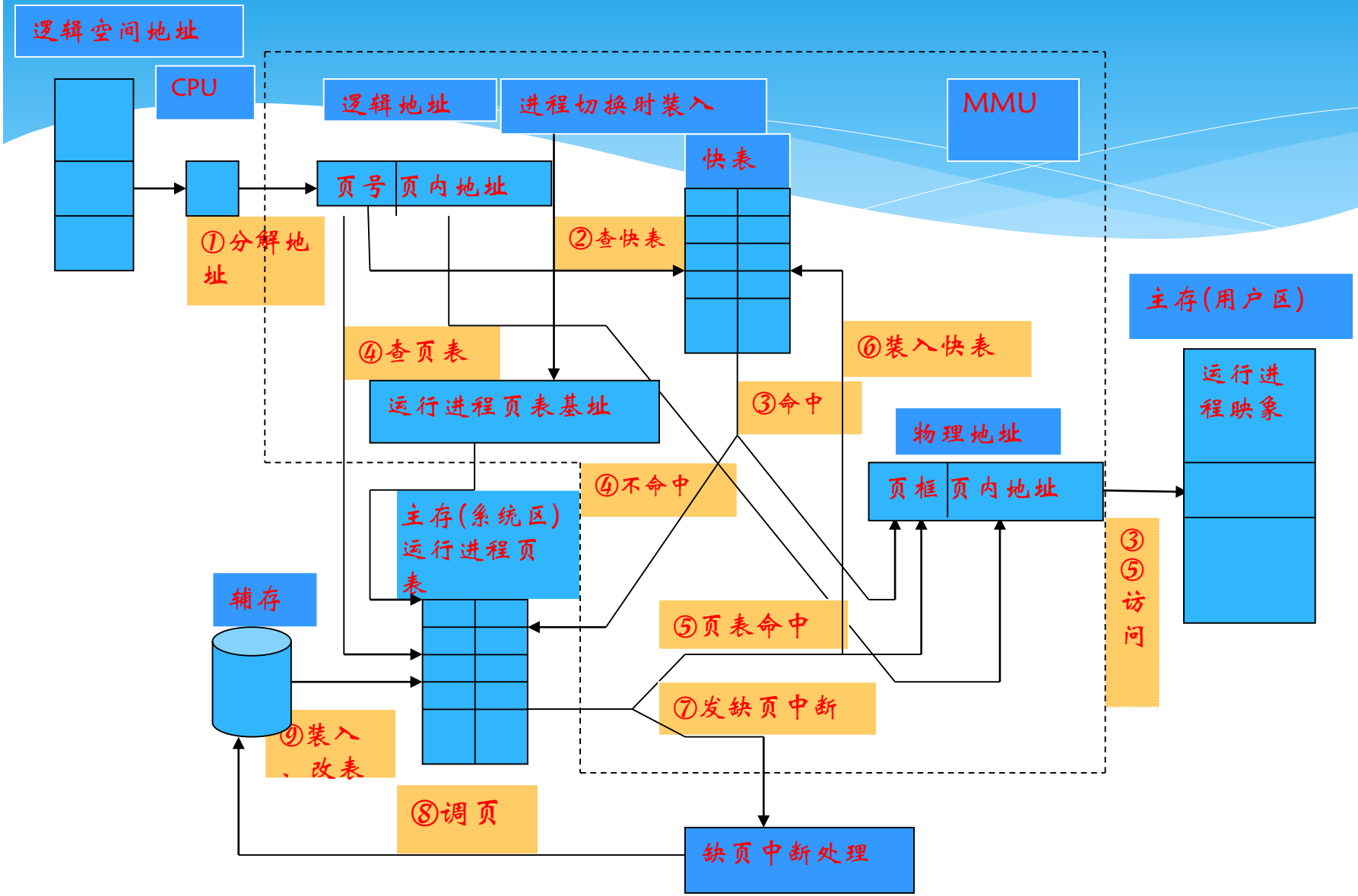
Paging

- Typically, each process has its own page table

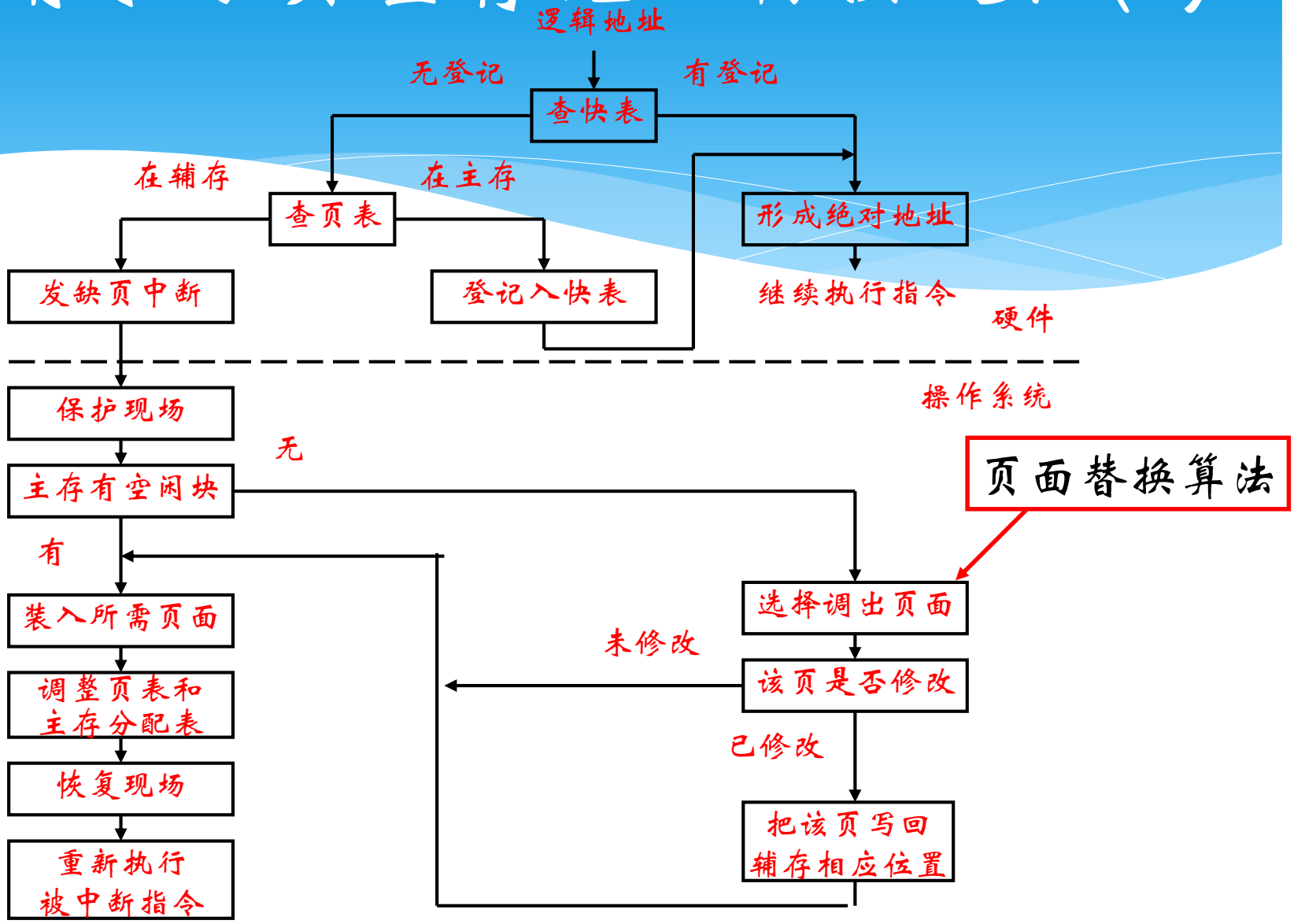


* Each page table entry contains a present bit to indicate whether the page is in main memory or not.
  * If it is in main memory, the entry contains the frame number of the corresponding page in main memory
  * If it is not in main memory, the entry may contain the address of that page on disk or the page number may be used to index another table (often in the PCB) to obtain the address of that page on disk

请求分页虚存地址转换过程(1)

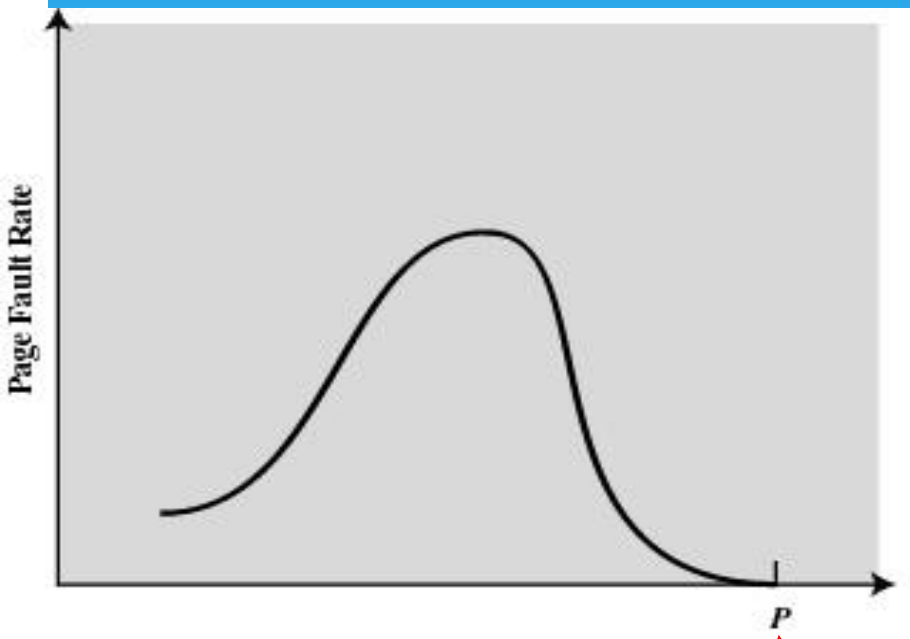


请求分页虚存地址转换过程(2)

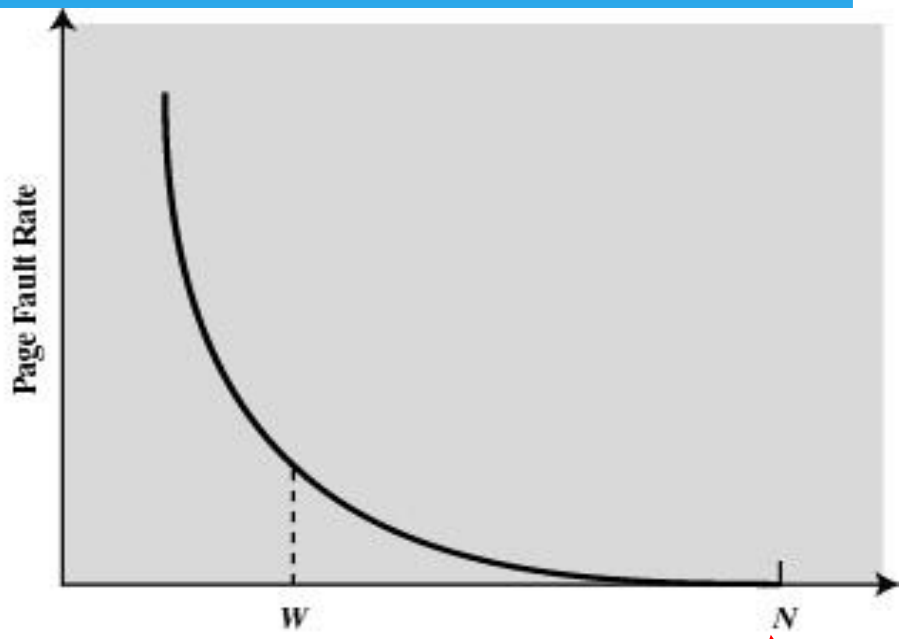


Page Size

* Smaller page size, less amount of internal fragmentation
* Smaller page size, more pages required per process
* More pages per process means larger page tables, and larger page tables means large portion of page tables in virtual memory
* Secondary memory is designed to efficiently transfer large blocks of data so a large page size is better



(a) Page Size



(b) Number of Page Frames Allocated

退化为固定分区

极限为页面全装载

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

Page Size

* Multiple page sizes provide the flexibility needed to effectively use a TLB
* Most operating system support only one page size

Combined Paging and Segmentation

* Paging is transparent to the programmer
* Paging eliminates external fragmentation
* Segmentation is visible to the programmer
* Segmentation allows for growing data structures, modularity, and support for sharing and protection
* Each segment is broken into fixed-size pages

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

8.3 虚拟分段技术

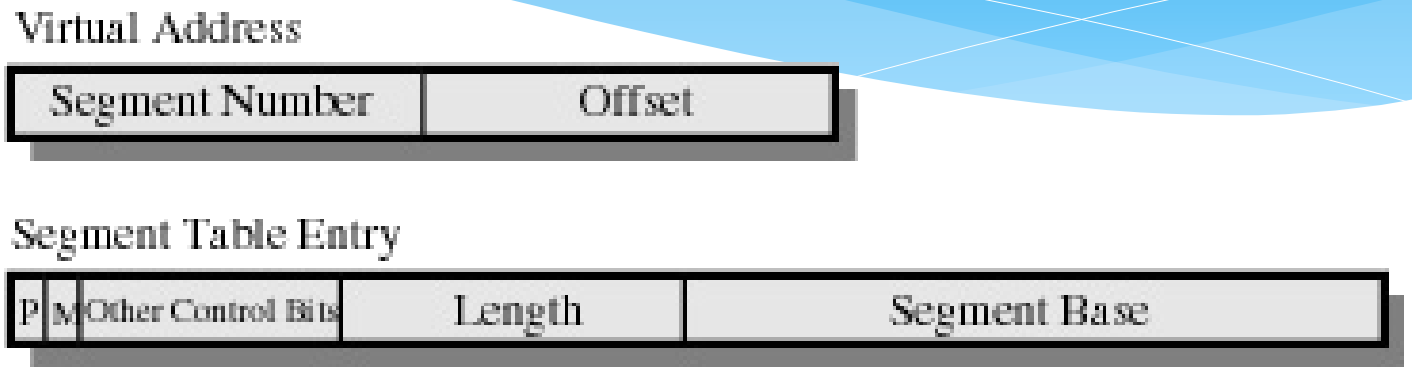
Segmentation

* May be unequal, dynamic size
* Simplifies handling of growing data structures
* Allows programs to be altered and recompiled independently
* Lends itself to sharing data among processes
* Lends itself to protection

Segment Tables

* corresponding segment in main memory
* Each entry contains 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

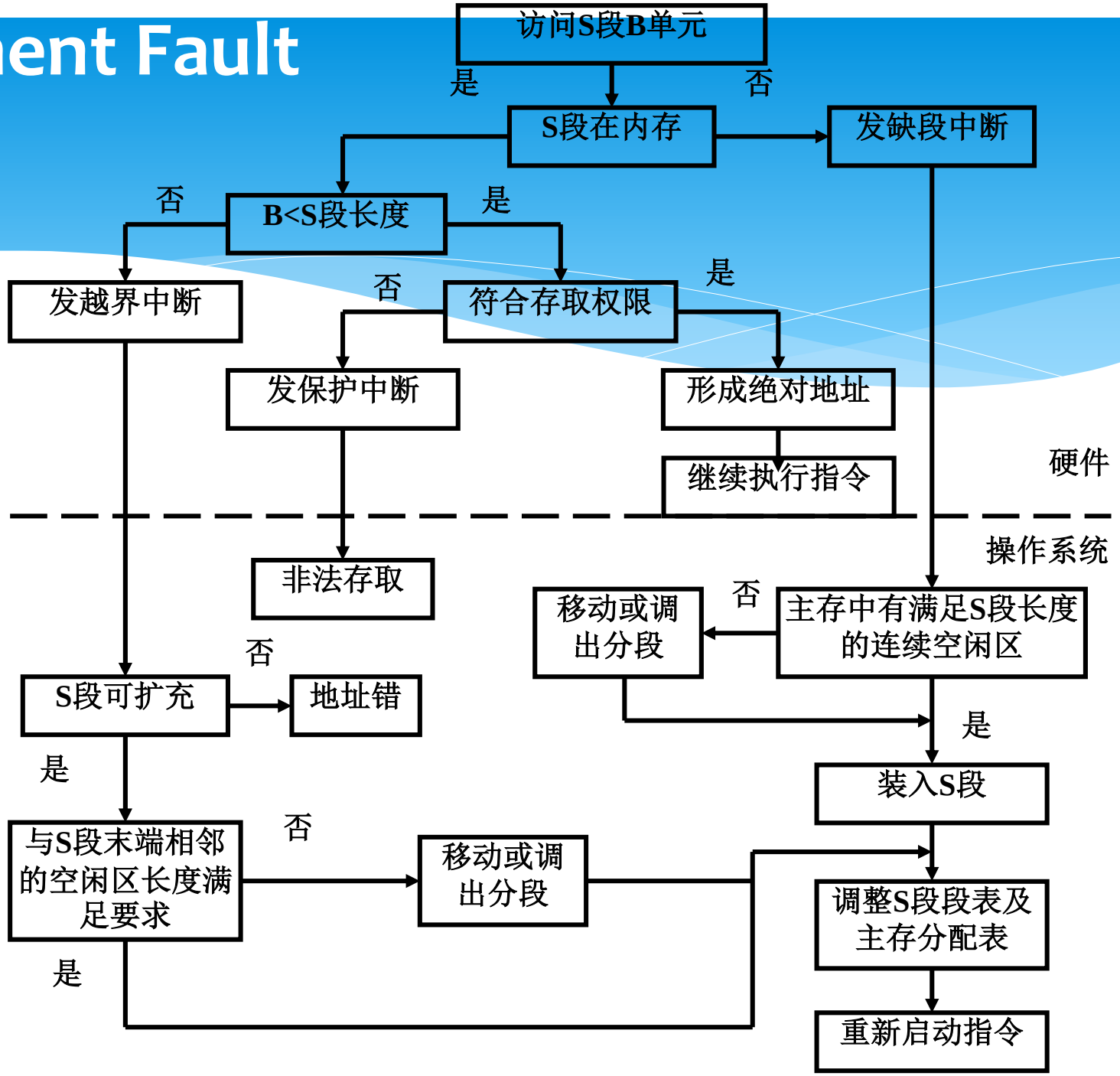
Segment Table Entries



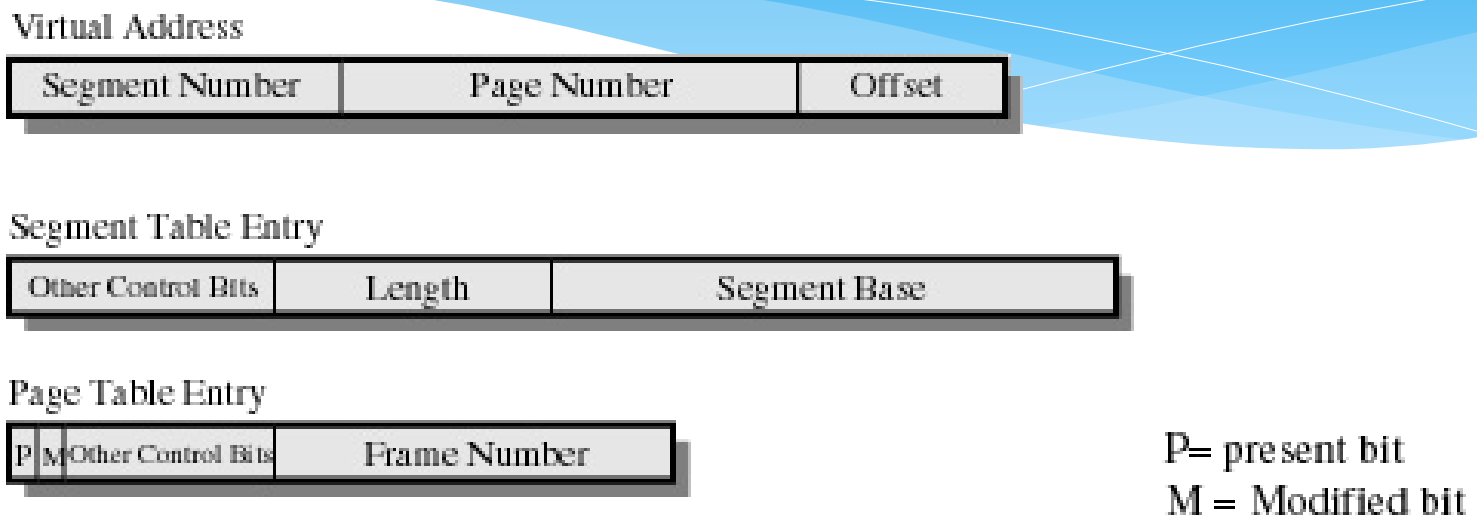
(b) Segmentation only

Figure 8.2 Typical Memory Management Formats

Segment Fault



Combined Segmentation and Paging



(c) Combined segmentation and paging

Figure 8.2 Typical Memory Management Formats

8.4 虚拟分页的操作系统软件

Fetch Policy

* Fetch Policy
  * Determines when a page should be brought into memory
  * Demand paging: 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: brings in more pages than needed
    * More efficient to bring in pages that reside contiguously on the disk

Placement Policy

* **Determines where in real memory a process piece is to reside**

Replacement Policy

* **Replacement Policy**
  * **Which page is replaced?**
  * **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

* Frame Locking: If frame is locked, it may not be replaced
  * Kernel of the operating system
  * Control structures
  * I/O buffers
* Associate a lock bit with each frame

影响缺页中断率的因素(1)

* 假定作业p共计n页，系统分配给它的主存块只有m块(1≤m≤n)。如果作业p在运行中成功的访问次数为s，不成功的访问次数为F，则总的访问次数A为：

$$A = S + F$$

又定义：

$$f = F / A$$

影响缺页中断率的因素(2)

称f为缺页中断率。影响缺页中断率f的因素有：

(1)主存页框数。

(2)页面大小。

(3)页面替换算法。

(4)程序特性。

程序局部性例子

* 程序将数组置为“0”，假定仅分得一个主存块，页面尺寸为128个字，数组元素按行存放，开始时第一页在主存。
* int A[128][128];　　int A[128][128];
* for(int j=0;j<128;j++)　　for(int i=0;i<128;i++)
  for(int i=0;i<128;i++)　　for(int j=0;j<128;j++)
* A[i][j]=0;　　A[i][j]=0;
* 128×128-1　　128-1

一种极端情况，教材pp.264

Basic Replacement Algorithms

* **Optimal policy**
  * **Selects for replacement that page for which the time to the next reference is the longest**
  * **Impossible to have perfect knowledge of future events**

Basic Replacement Algorithms

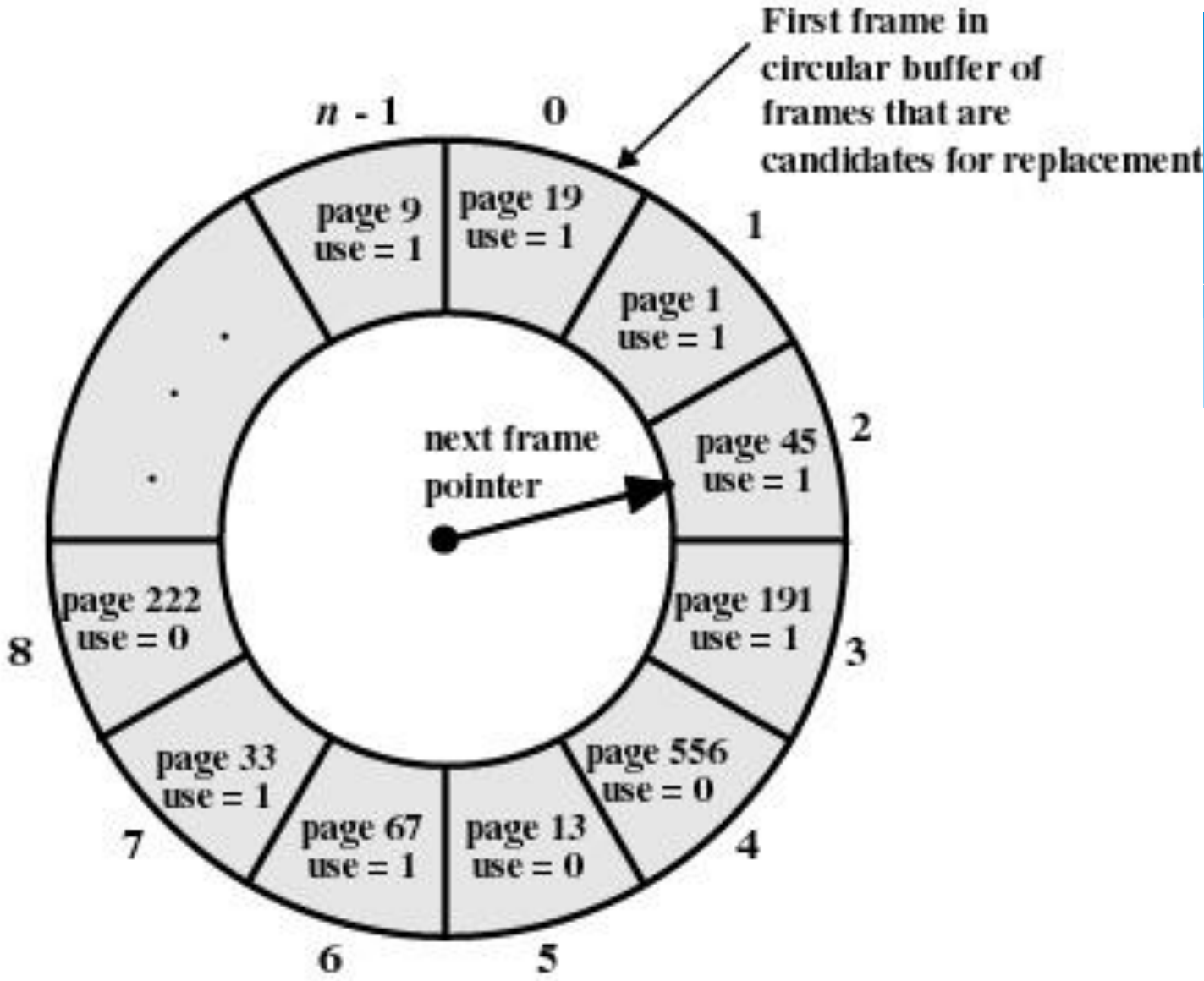
* First-in, first-out (FIFO)
  * Treats page frames allocated to a process as a circular buffer
  * Pages are removed in round-robin style
  * Simplest replacement policy to implement
  * Page that has been in memory the longest is replaced
  * These pages may be needed again very soon

Basic Replacement Algorithms

* Least Recently Used (LRU)
  * 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
  * Each page could be tagged with the time of last reference. This would require a great deal of overhead.

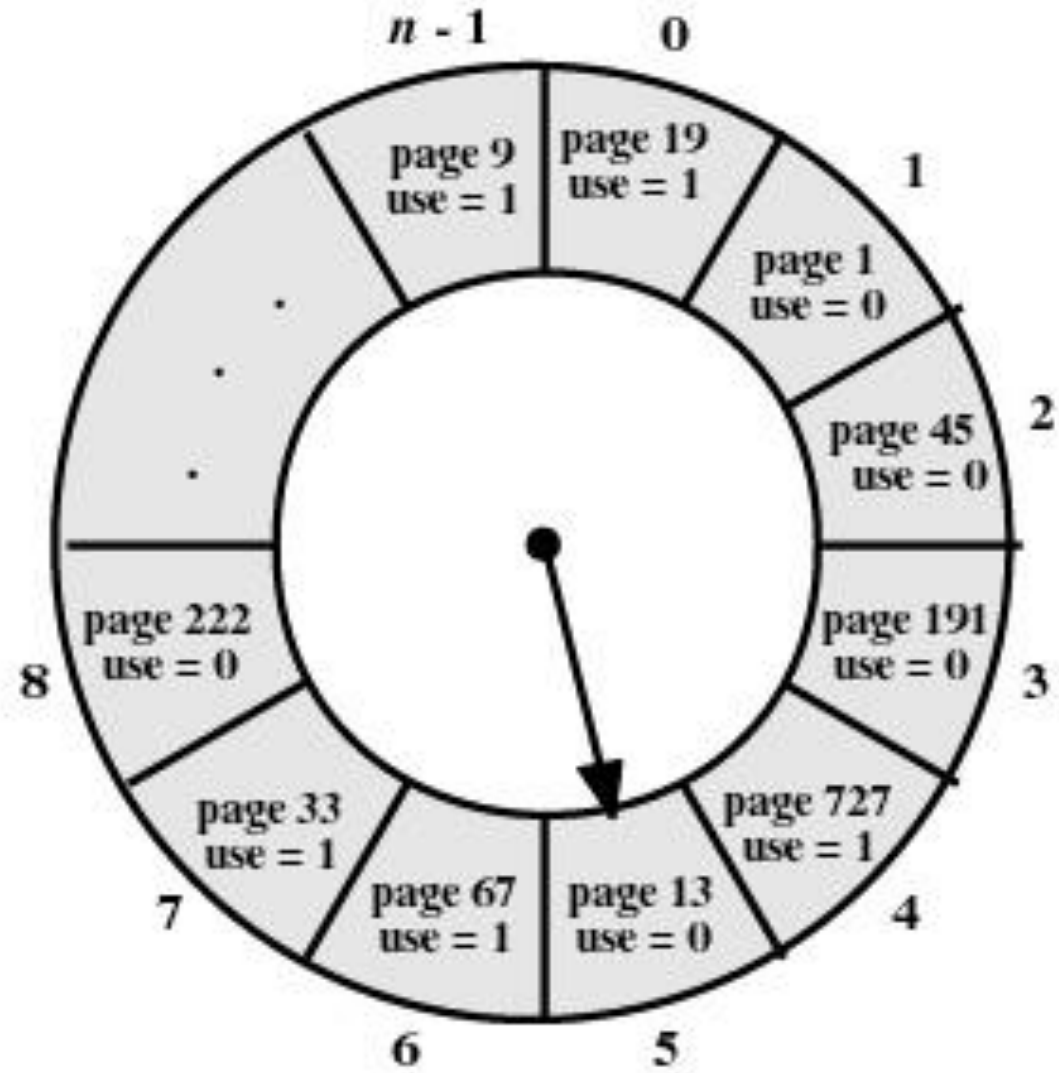
Basic Replacement Algorithms

* Clock Policy
  * Additional bit called a use bit
  * When a page is first loaded in memory, the use bit is set to 1
  * When the page is referenced, the use bit is set to 1
  * When it is time to replace a page, the first frame encountered with the use bit set to 0 is replaced.
  * During the search for replacement, each use bit set to 1 is changed to 0



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

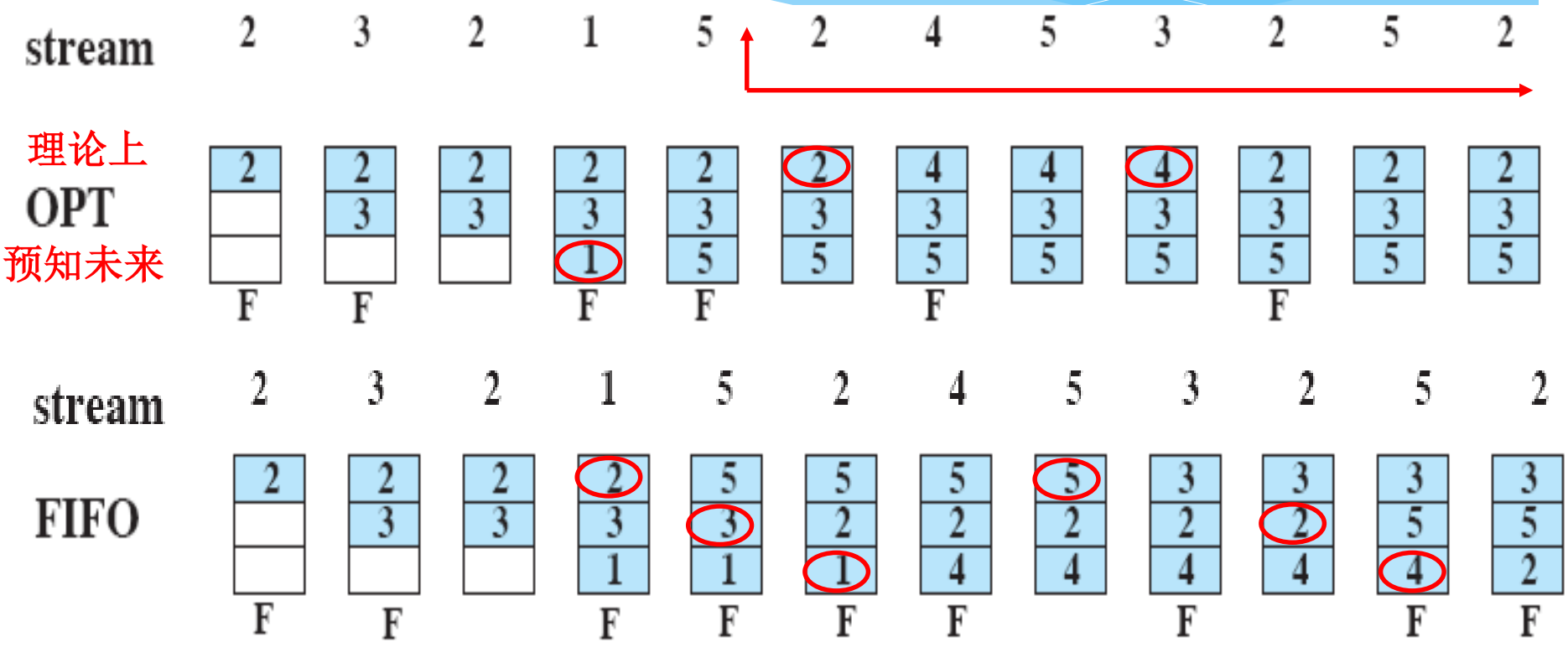
Figure 8.16 Example of Clock Policy Operation

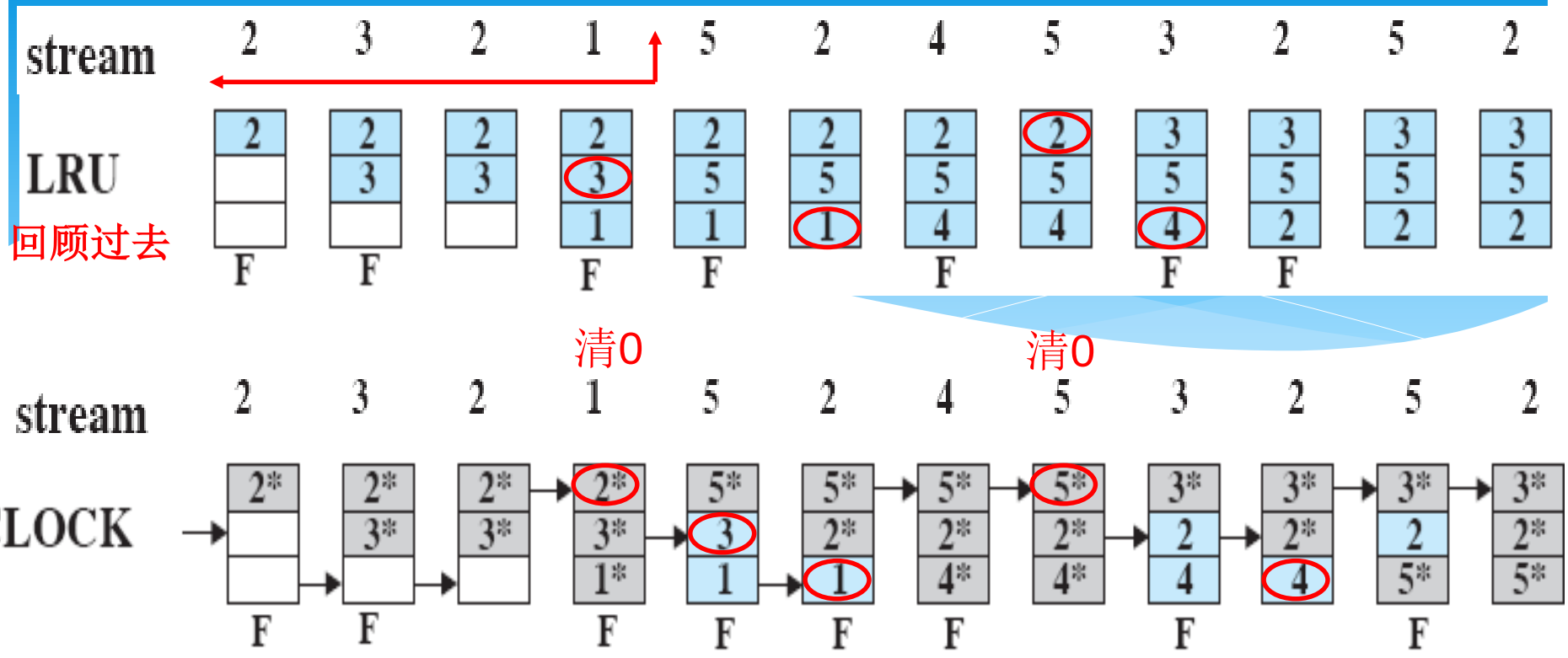


(b) State of buffer just after the next page replacement

Figure 8.16 Example of Clock Policy Operation

示例:页面替换算法


stream
2 3 2 1 5 2 4 5 3 2 5 2
理论上
OPT
预知未来
F F F F F F
stream
2 3 2 1 5 2 4 5 3 2 5 2
FIFO
F F F F F F F F F



星号表示相应的使用位等于1，箭头表示指针的当前位置。

当一页被替换时，指向下一帧。 虽然早就进来，但是最近使用过，所以不急着替换

当需要替换一页时，扫描缓冲区，查找使用位被置为0的一帧。

每当遇到一个使用位为1的帧时，就将该位重新置为0；

如果在这个过程开始时，所有帧的使用位均为0，选择遇到的第一个帧替换；

如果所有帧的使用位为1，则指针在缓冲区中完整地循环一周，把所有使用位都置为0，并且停留在最初的位置上，替换该帧中的页。

Belady's Anomaly (Belady异常)

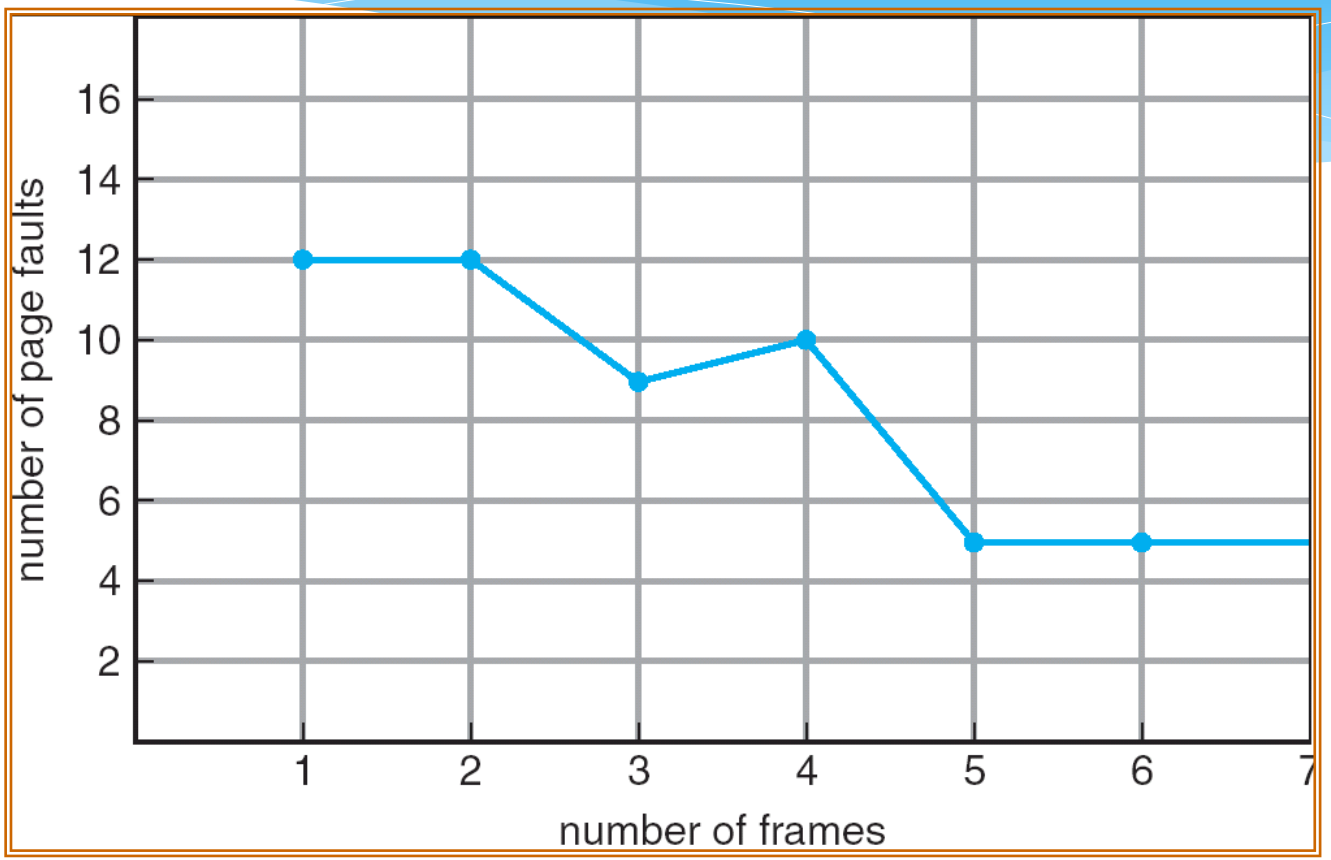
Belady 异常

FIFO	1	2	3	4	1	2	5	1	2	3	4	5

more frames ⇒ more page faults

对应《操作系统教程(第4版)》pp.265 **"Belady异常"**

FIFO Illustrating Belady's Anomaly (FIFO算法的Belady异常)



对应《操作系统教程(第4版)》pp.265 "Belady异常"

Comparison of Placement Algorithms

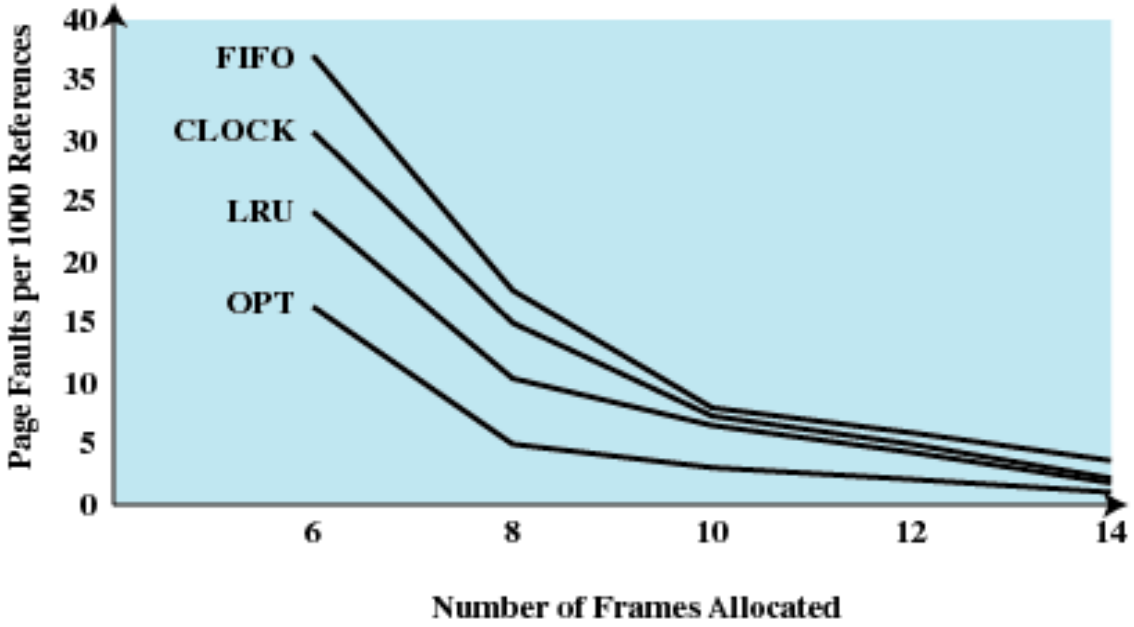


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

这是一个整体的性能比较，对于个案，有可能Clock优于LRU

Basic Replacement Algorithms

* Page Buffering
  * Replaced page is added to one of two lists
    * free page list if page has not been modified
    * modified page list
  * Replaced page remains in memory
    * If referenced again, it is returned at little cost
    * Modified pages are written out in cluster

Resident Set Size

* **Fixed-allocation**
  * **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**
  * **number of pages allocated to a process varies over the lifetime of the process**

Fixed Allocation, Local Scope

* Number of frames allocated to process is fixed
* Page to be replaced is chosen from among the frames allocated to the process

Variable Allocation, Global Scope

* Number of frames allocated to process is variable
* Page to be replaced is chosen from all frames
* Easiest to implement
* Adopted by many operating systems
* Operating system keeps list of free frames
* Free frame is added to resident set of process when a page fault occurs

Variable Allocation, Local Scope

* Number of frames allocated to process is variable
* Page to be replaced is chosen from among the frames allocated to the process
* 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 allocation from time to time

局部页面替换算法

1) 局部最佳页面替换算法
2) 工作集模型和工作集置换算法
3) 模拟工作集替换算法
4) 缺页频率替换算法

1) 局部最佳页面替换算法(1)

* 实现思想：进程在时刻t访问某页面，如果该页面不在主存中，导致一次缺页，把该页面装入一个空闲页框
* 不论发生缺页与否，算法在每一步要考虑引用串，**如果该页面在时间间隔(t, t+τ)内未被再次引用，那么就移出；否则，该页被保留在进程驻留集中**
* τ为一个系统常量，间隔(t, t+τ)称作滑动窗口。例子中τ=3

局部最佳页面替换算法(2)

时刻 t	0	1	2	3	4	5	6	7	8	9	10
引用串	P_4	P_3	P_3	P_4	P_2	P_3	P_5	P_3	P_5	P_1	P_4
P_1	—	—	—	—	—	—	—	—	—	√	—
P_2	—	—	—	—	√	—	—	—	—	—	—
P_3	—	√	√	√	√	√	√	√	—	—	—
P_4	√	√	√	√	—	—	—	—	—	—	√
P_5	—	—	—	—	—	—	√	√	√	—	—
In_t		P_3			P_2		P_5			P_1	P_4
Out_t					P_4	P_2			P_3	P_5	P_1

局部最佳页面替换算法(3)

时刻	引用串	驻留集		Out_t	滑动窗口
		已驻留	In_t		
T0	P4	P4			(0,0+3)看到p4
T1	P3	P4	P3		(1,1+3)看到p3, p4
T2	P3	P3,p4			(2,2+3)看到p3, p4
T3	P4	P3,p4			(3,3+3)看到p3, p4
T4	P2	P3	P2	p4	(4,4+3)中看不到p4
T5	P3	P3		P2	(5,5+3)中看不到p2
T6	P5	P3	P5		(6,6+3)看到p3, p5
T7	P3	P3, P5			(7,7+3)看到p3, p5
T8	P5	P5		P3	(8,8+3)中看不到p3
T9	P1		P1	P5	(9,9+3)中看不到p5
T10	P4		P4	P1	(10,10+3)中看不到p1

缺页总数为5次，驻留集大小在1-2之间变化，任何时刻至多两个页框被占用，通过增加τ值，缺页数目可减少，但代价是花费更多页框。

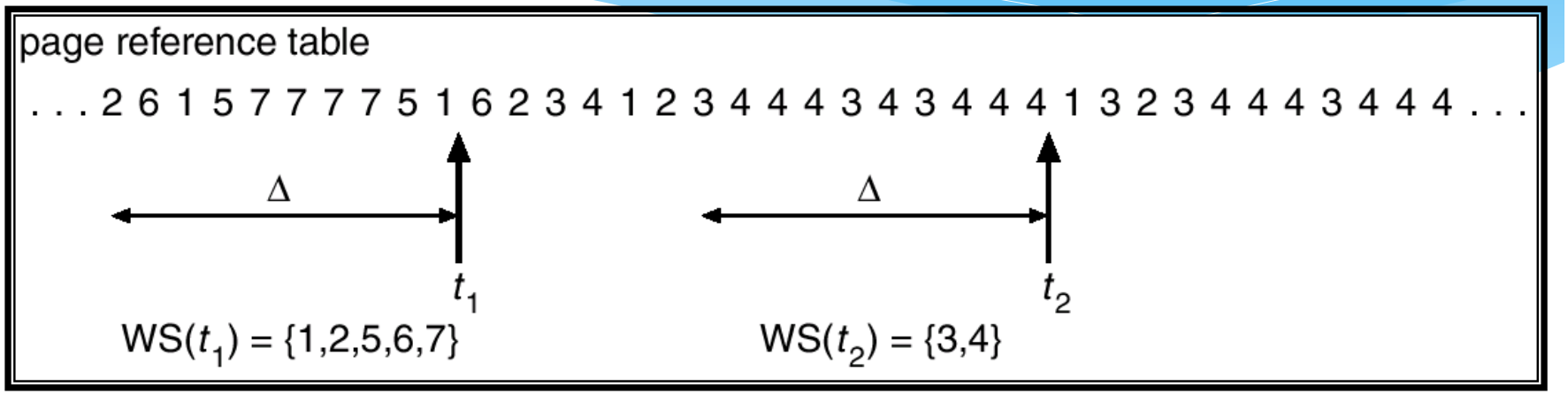
2) 工作集模型和工作集置换算法

* 进程工作集指“在某一段时间间隔内进程运行所需访问的页面集合”
* 实现思想：工作集模型用来对局部最佳页面替换算法进行模拟实现，不向前查看页面引用串，而是基于程序局部性原理向后看
* 任何给定时刻，进程不久的将来所需主存页框数，可通过考查其过去最近的时间内的主存需求做出估计

进程工作集

* 指“在某一段时间间隔内进程运行所需访问的页面集合”，W(t，Δ)表示在时刻t-Δ到时刻t之间((t-Δ，t))所访问的页面集合，进程在时刻t的工作集
* Δ是系统定义的一个常量。变量Δ称为“工作集窗口尺寸”，可通过窗口来观察进程行为，还把工作集中所包含的页面数目称为“工作集尺寸”
* Δ=3

Working-set model



Sequence of Page References	Window Size, •			
	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

命中

工作集替换示例

时刻 t	0	1	2	3	4	5	6	7	8	9	10
引用串	P_1	P_3	P_3	P_4	P_2	P_3	P_5	P_3	P_5	P_1	P_4
P_1	√	√	√	√	—	—	—	—	—	√	√
P_2	—	—	—	—	√	√	√	√	—	—	—
P_3	—	√	√	√	√	√	√	√	√	√	√
P_4	√	√	√	√	√	√	√	—	—	—	√
P_5	√	√	—	—	—	—	√	√	√	√	√
In_t		P_3			P_2		P_5			P_1	P_4
Out_t			P_5		P_1			P_4	P_5		

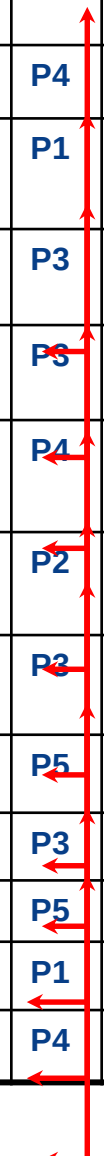
其中，p1在时刻t=0被引用，p4在时刻t=-1被引用，而p5在时刻t=-2被引用，Δ=3

工作集替换示例进一步说明

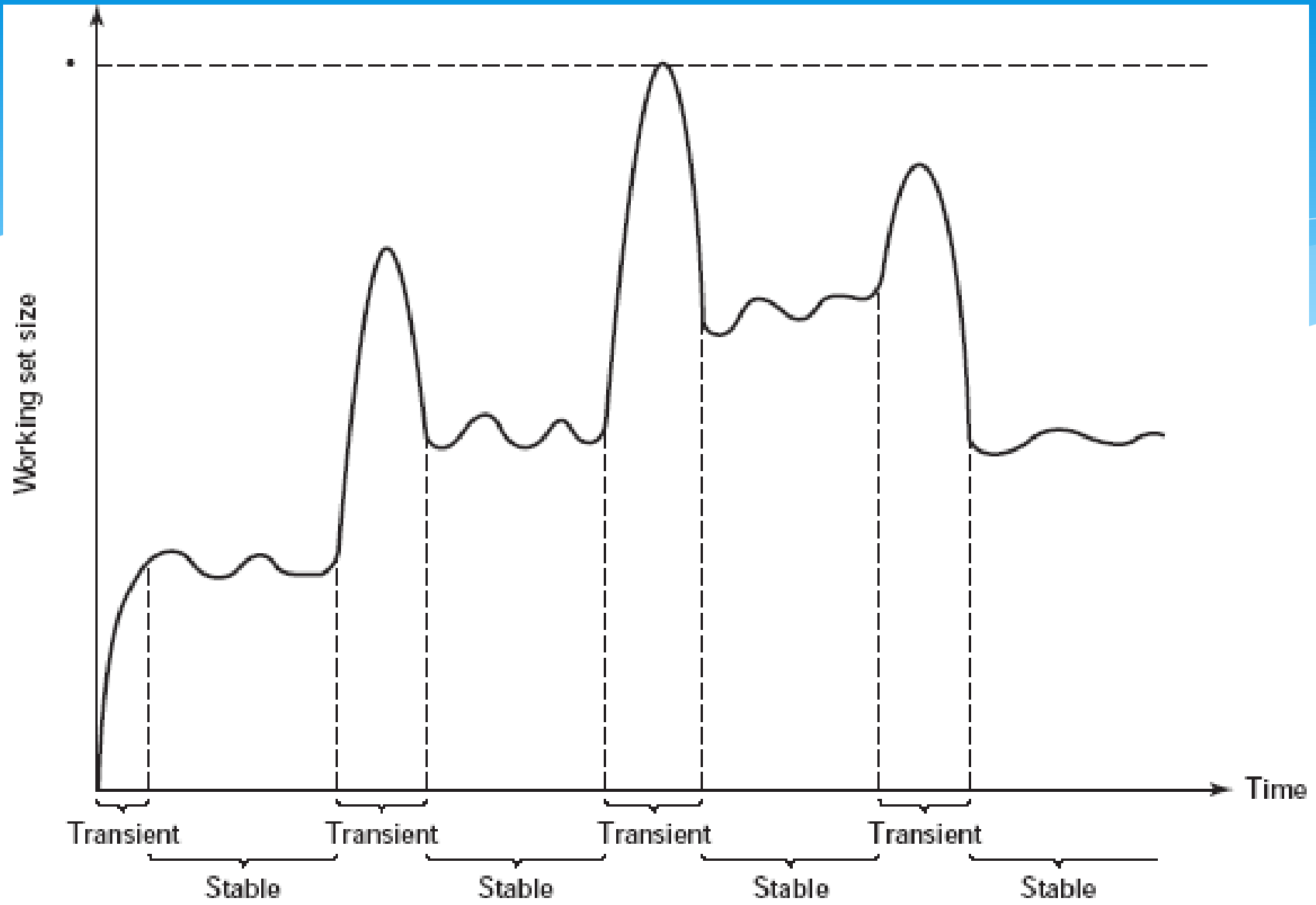
时刻t	-2	-1	0	1	2	3	4	5	6	7	8	9	10
引用串	p5	p4	p1	p3	p3	p4	p2	p3	p5	p3	p5	p1	p4
p1			√	√	√	√	—	—	—	—	—	√	√
p2			—	—	—	—	√	√	√	√	—	—	—
p3			—	√	√	√	√	√	√	√	√	√	√
p4		√	√	√	√	√	√	√	√	—	—	—	√
p5	√	√	√	√	—	—	—	—	√	√	√	√	√
In_t				p3			p2		p5			p1	p4
OUT_t					p5		p1			p4	p2		

工作集页面替换算法

时刻	引用串	工作集		Outi	(t-Δ，t)= (t-3，t)
		已驻留	Ini		
T-2	P5		P5		
T-1	P4	P5	P4		
T0	P1	P4,p5	P1		
T1	P3	P1,P4,p5	P3		(1-3,1)看到p1, p3 p4, p5
T2	P3	P1,P3,p4		p5	(2-3,2)看到p1, p3, p4。 P5出。
T3	P4	P1,P3,p4			(3-3,3)看到p1, p3, p4
T4	P2	P3,p4	P2	P1	(4-3,4)看到p2, p3, p4。 P1出。
T5	P3	P2,P3, P4			(5-3,5)看到p2, p3, p4
T6	P5	P2,P3, P4	P5		(6-3,6)看到p2, p3, p4, p5
T7	P3	P2,P3, P5		P4	(7-3,7)看到p2, p3, p5。 P4出。
T8	P5	P3, P5		P2	(8-3,8)看到p3, p5。 P2出。
T9	P1	P3, P5	P1		(9-3,9)看到p1, p3, P5
T10	P4	P1,P3, P5	P4		(10-3,10)看到p1, p3, p4,p5



工作集的大小会随着命中率而调整



Typical Graph of Working Set Size [MAEK87]

Cleaning Policy

* Demand cleaning
  * a page is written out only when it has been selected for replacement
* Precleaning
  * pages are written out in batches

Cleaning Policy

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

Load Control

* Determines the number of processes that will be resident in main memory
* Too few processes, sometimes all processes will be blocked
* Too many processes will lead to thrashing

Process Suspension

* Lowest priority process
* Faulting process
  * this process does not have its working set in main memory so it will be blocked anyway
* Last process activated
  * this process is least likely to have its working set resident

Process Suspension

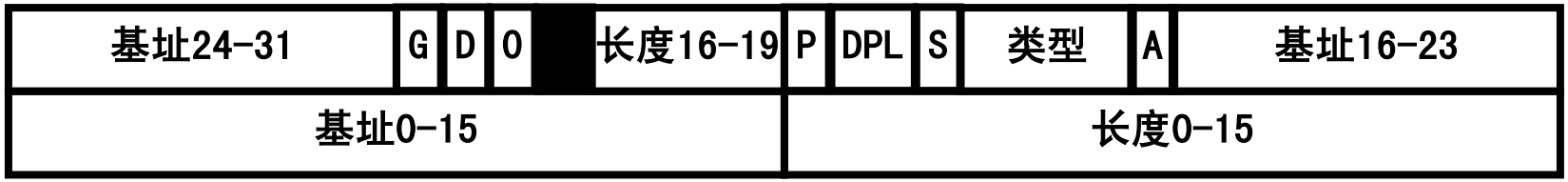
* Process with smallest resident set
  * this process requires the least future effort to reload
* Largest process
  * obtains the most free frames
* Process with the largest remaining execution window

实例研究：Intel的Pentium

* Pentium的分段机制
  * 可以包括16K个独立的段
  * 每个段最多可以容纳10亿个32位字
  * 局部描述符表LDT：每个程序都有，描述局部于每个程序的段，包括代码、数据、堆栈等
  * 全局描述符表GDT：所有程序共享，描述系统段，包括操作系统自己

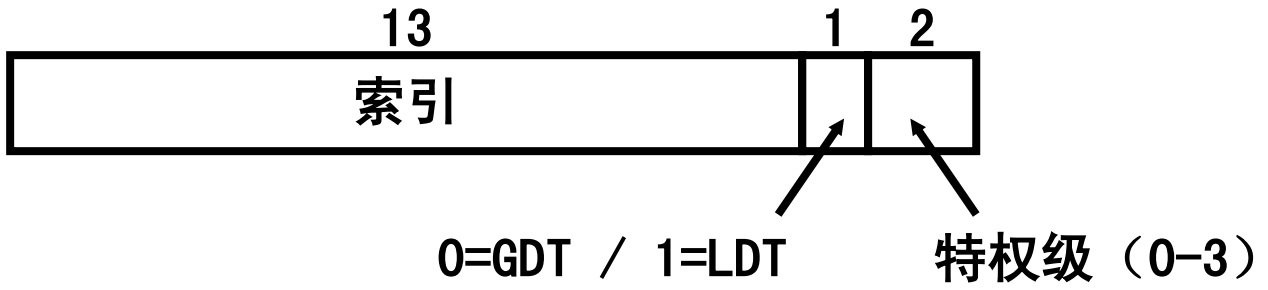
实例研究：Intel的Pentium

* Pentium的分段机制——描述符结构
  * 基地址共32位（分三处合并）
  * 长度位共20位，长度单位是字节或页(4KB)
  * G位用于描述颗粒大小，即段长度的计量单位
  * D位：D=1，为32位段；D=0，为16位段
  * P位表示内存段是否在物理主存中
  * Dpl位（2位）表示特权级（0—3）
  * S位为段位，S=1，应用程序段；S=0，系统段
  * 类型字段（3位）表示内存段类型，如可执行代码段、只读数据段、调用门等等
  * A位为访问位，表示是否访问过内存段



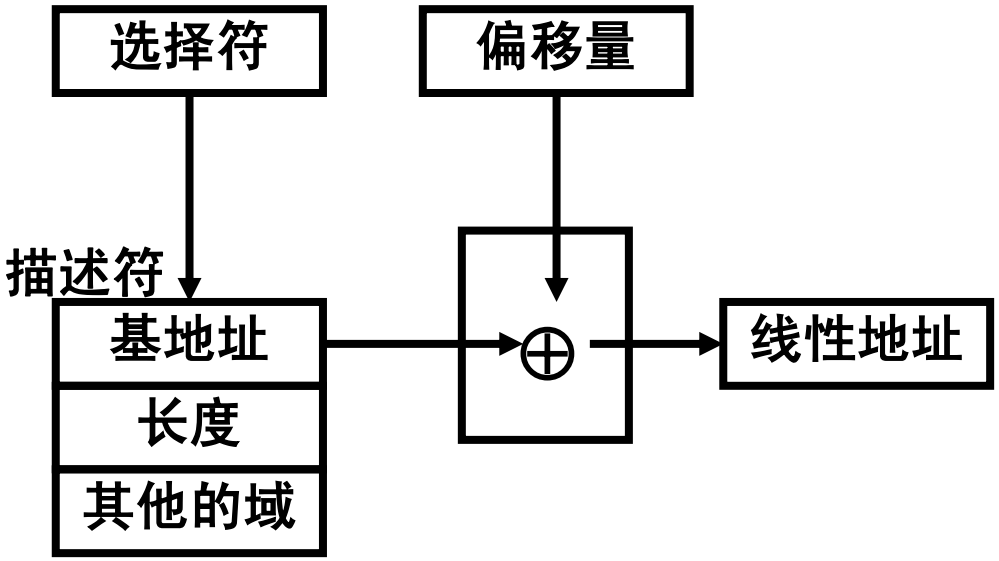
实例研究：Intel的Pentium

* Pentium的分段机制
  * 段选择符selector
  * 6个段寄存器：存放选择符
  * 微程序寄存器：存放描述符



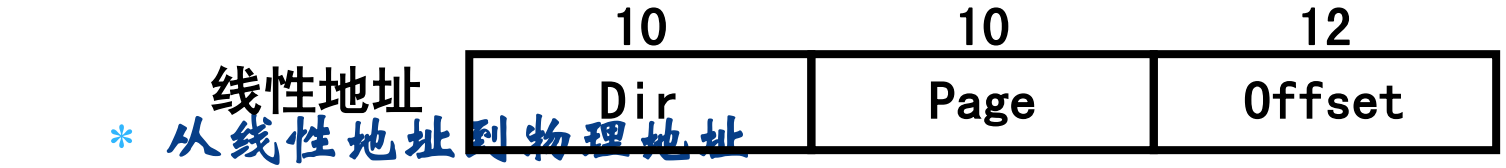
实例研究：Intel的Pentium

* Pentium的分段机制
  * 地址转换和存储保护

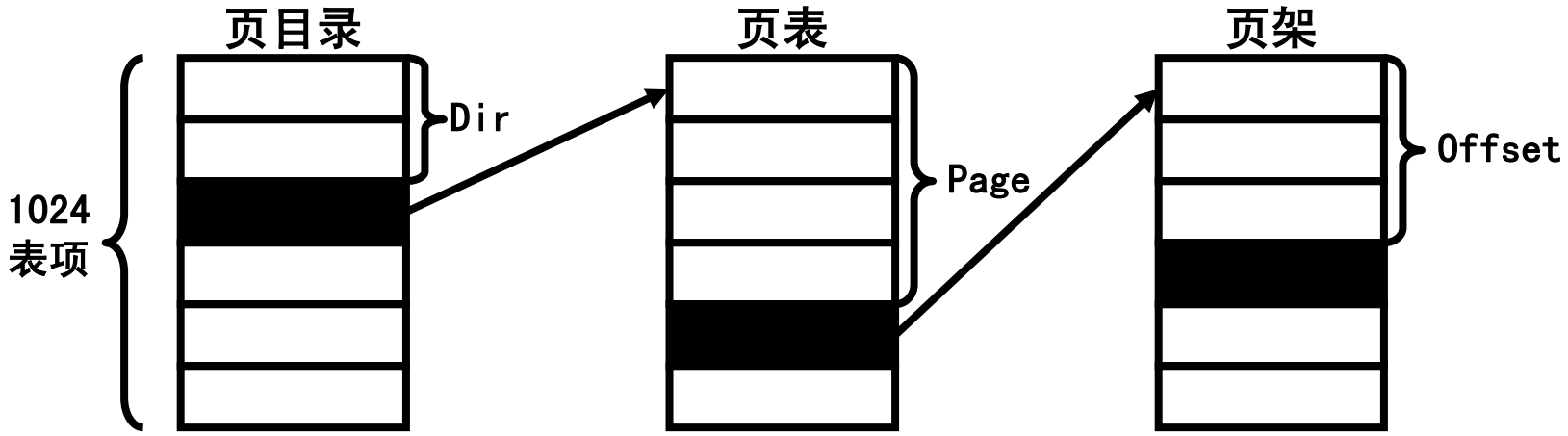


实例研究：Intel的Pentium

* Pentium的分页机制
  * 线性地址的组成



  * 从线性地址到物理地址



实例研究：Intel的Pentium

* Pentium的分页机制——页目录项和页表项的结构
  * 均为32位
  * 其中20位是页表位置/页架号
  * D位：修改位，为淘汰做准备
  * A位：读/写引用位，LRU法使用
  * P位：存在位，页面在内存时，P=1
  * U/S位：用户/管理员位，若U/S=0，该页为一个管理员页面（OS页面），用户不能访问
  * R/W位：读/写位，R/W=1时允许页修改；R/W=0时，不允许页修改；通常程序区的R/W=0

本主题教学目标

1. 了解虚拟存储管理的基本特点
2. 掌握页式虚拟存储管理
3. 掌握段式虚拟存储管理
4. 掌握页面调度策略与算法
5. 了解Pentium的地址转换机构