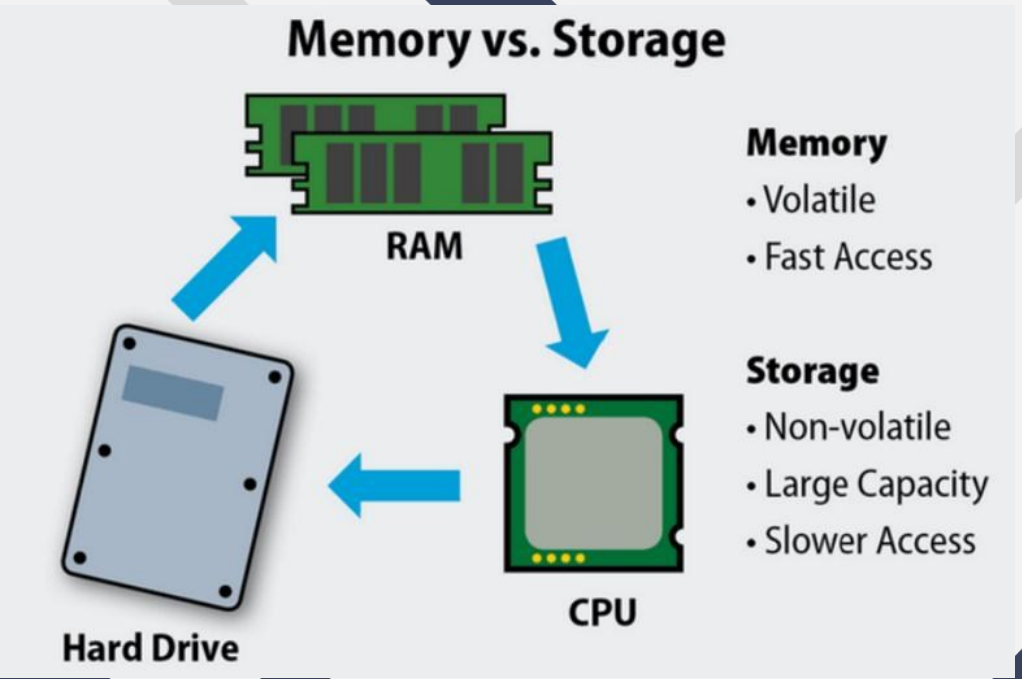


操作系统A

第五章 虚拟存储器

何洪豆 \ 信息科学与工程学院

Email: hhd@ysu.edu.cn

目录

CONTENTS

第五章 虚拟存储器

5.1 虚拟存储器概述

5.1 虚拟存储器概述

第四章所介绍的各种存储器管理方式有一个共同的特点，即它们都要求将一个作业全部装入内存后方能运行。于是，出现了下面这样两种情况：

(1) **有的作业很大**，其所要求的内存空间超过了内存总容量，作业不能全部被装入内存，致使该作业无法运行；

(2) **有大量作业要求运行**，但由于内存容量不足以容纳所有这些作业，只能将少数作业装入内存让它们先运行，而将其它大量的作业留在外存上等待。

5.1.1 常规存储管理方式的特征和局部性原理

1. 常规存储器管理方式的特征

我们把前一章中所介绍的各种存储器管理方式统称为传统存储器管理方式，它们全都具有如下两个共同的特征：

(1) 一次性

(2) 驻留性

5.1.1 常规存储管理方式的特征和局部性原理

2. 局部性原理

程序运行时存在的局部性现象，很早就已被人发现，但直到1968年，P. Denning才真正指出：程序在执行时将呈现出局部性规律，即在一较短的时间内，程序的执行仅局限于某个部分，相应地，它所访问的存储空间也局限于某个区域。

局限性又表现在下述两个方面：

(1) **时间局限性**。
(2) **空间局限性**。

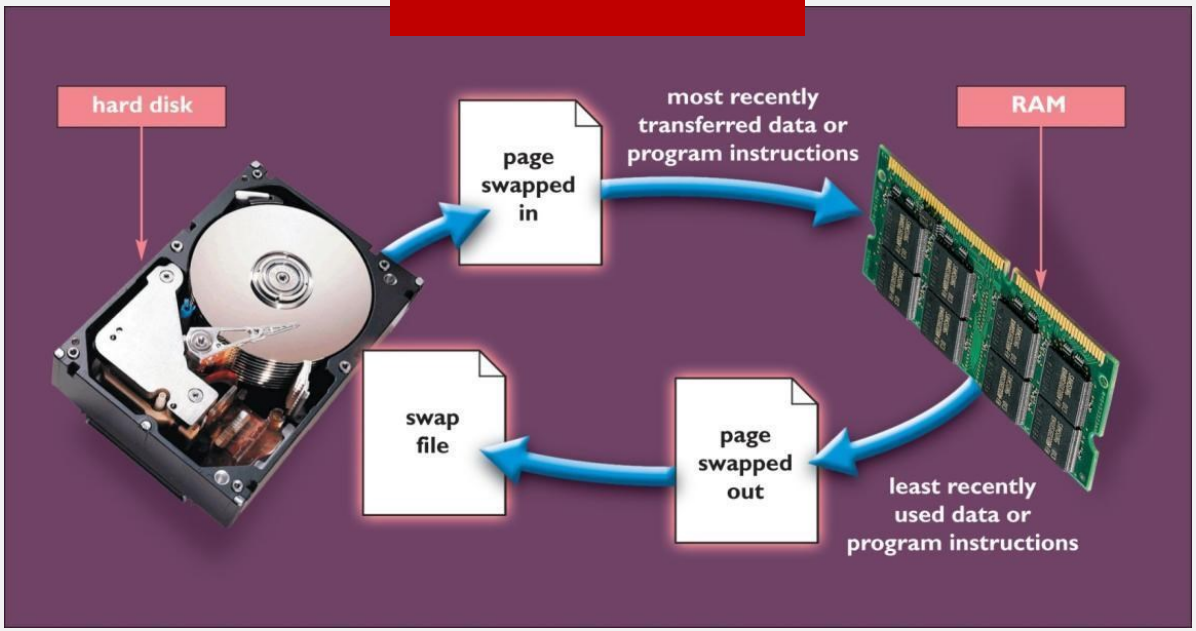
} 局部性原理

5.1.1 常规存储管理方式的特征和局部性原理

3. 虚拟存储器的基本工作情况

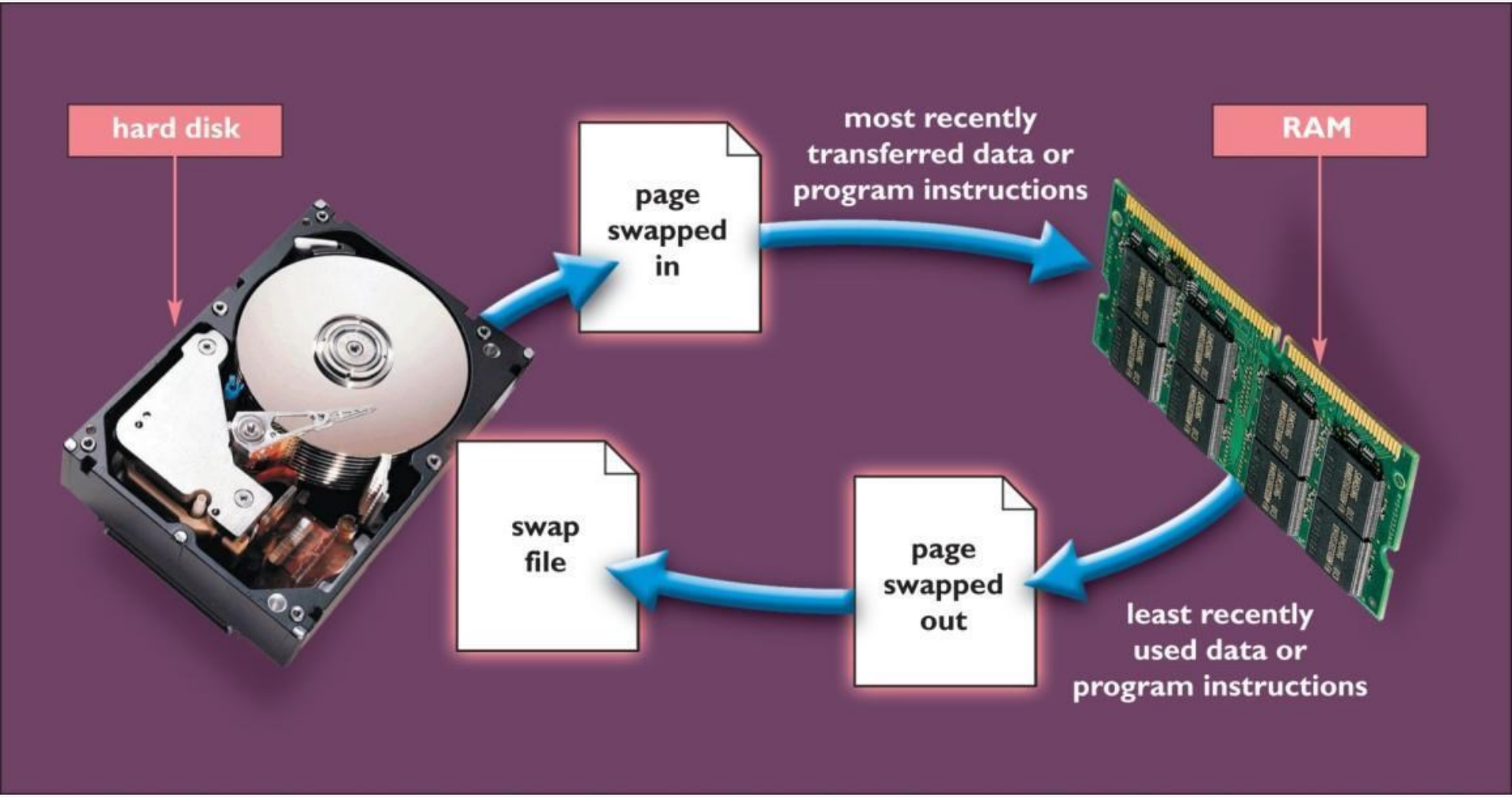
基于局部性原理可知，应用程序在运行之前没有必要将之全部装入内存，而**仅须将那些当前要运行的少数页面或段先装入内存便可运行**，其余部分暂留在盘上。

时空转换思维



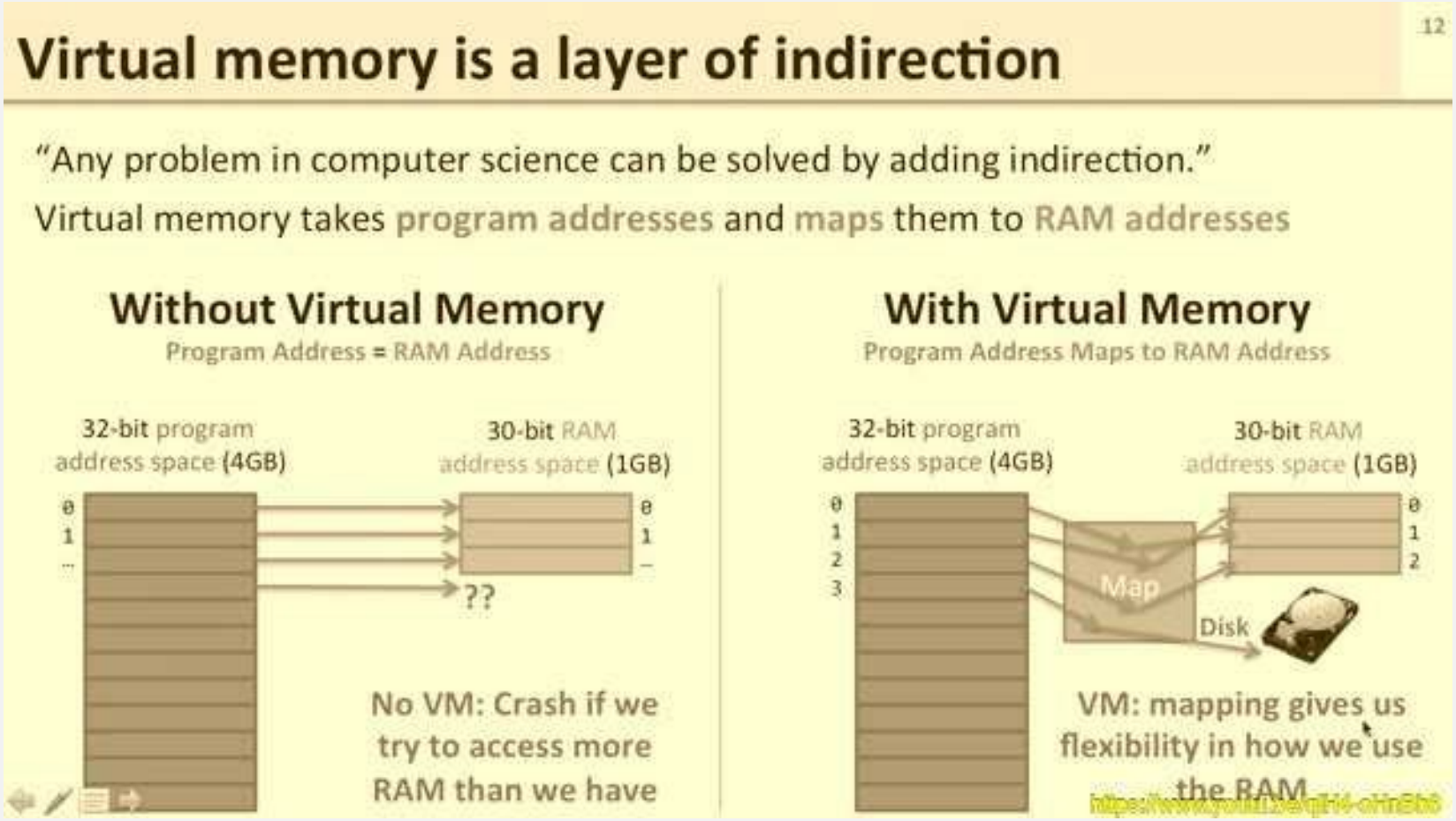
虚拟存储器

- 虚拟存储器通过页面置换来解决程序运行的内存需求问题。



虚拟存储器

- 虚拟存储器是一个间接层。
- “计算机科学中的任何问题都可以通过加入一个间接层来解决。”
- 虚拟存储器获取程序地址空间并将它们映射到RAM空间。



时空转换的创新思维

以时间换空间：以时间为代价换取更大空间

- **虚拟存储器**

  通过多次调入内存实现较大较多程序的运行；

- **手机九宫格键盘**

  以打字时间代价扩大屏幕空间缩小手机空间；

以空间换时间：以空间为代价赢得更多时间

- **缓冲区的引入**

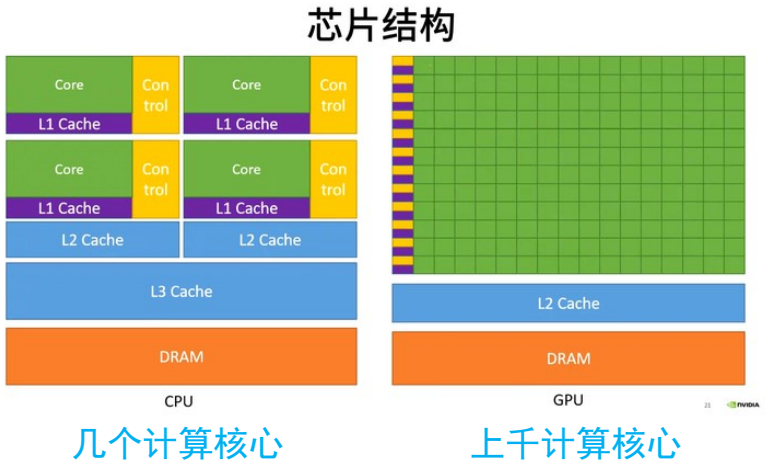
  增加高速缓存减少中断次数以换取运行效率；

- **深度学习GPU运算**

  利用大量简单运算器加快矩阵并行运算速度；

- **农村包围城市战略思想**

  深入广阔农村空间争取抗战胜利的时间。



创新也许并不难

坚持科技是第一生产力、人才是第一资源、创新是第一动力！
—— 二十大报告

- **时空转换思维** —— 数据库索引技术（B+树、红黑树等）

  通过用更多存储空间建立数据索引，加快数据检索的速度；

- **交叉融合思维** —— 人工智能的实际应用

  海康威视：人工智能 + 摄像头，智能报警、人脸识别等；

  智能手表：人工智能 + 手表，心率检测、血氧检测等；

- **简 化 思 维** —— 简化操作模式

  手机从键盘按键模式变成触屏的虚拟键，让手机更加简洁美观；

- **分解扩展思维** —— 分解组成成分 + 扩展解决方法

  比亚迪：汽车分解出动力系统 + 扩展增加电机驱动 → 混动汽车。

开阔创新思维；
加强创新意识；
坚持创新发展。

5.1.2 虚拟存储器的定义和特征

1. 虚拟存储器的定义

当用户看到自己的程序能在系统中正常运行时，他会认为，该系统所具有的内存容量一定比自己的程序大，或者说，用户所感觉到的内存容量会比实际内存容量大得多。但用户所看到的大容量只是一种错觉，是虚的，故人们把这样的存储器称为虚拟存储器。

5.1.2 虚拟存储器的定义和特征

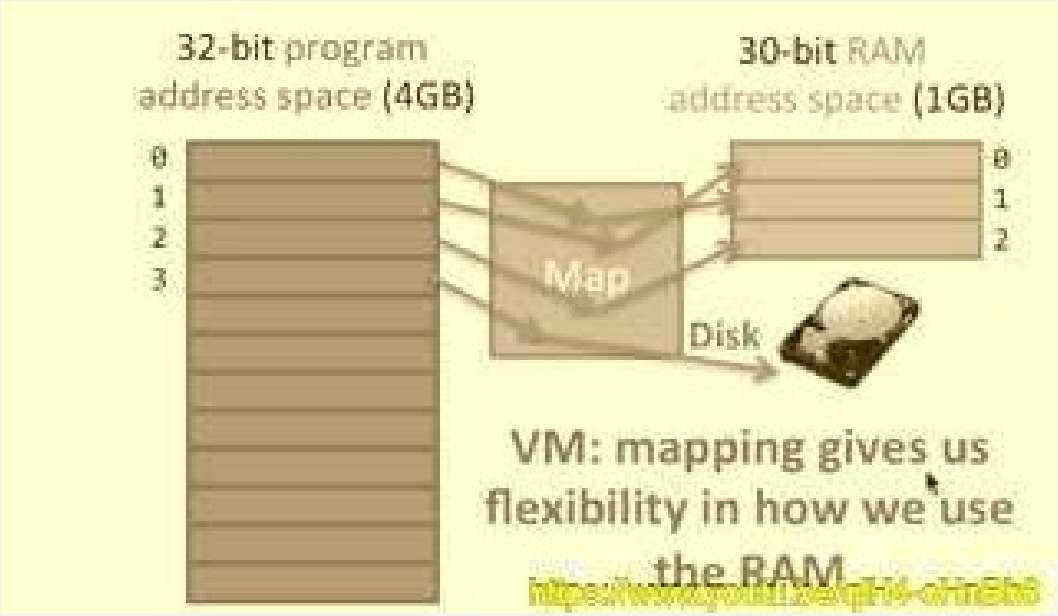
2. 虚拟存储器的特征

与传统的存储器管理方式比较，虚拟存储器具有以下三个重要特征：

(1) 多次性。

(2) 对换性。

(3) 虚拟性。



5.1.3 虚拟存储器的实现方法

1. 分页请求系统

1) 硬件支持

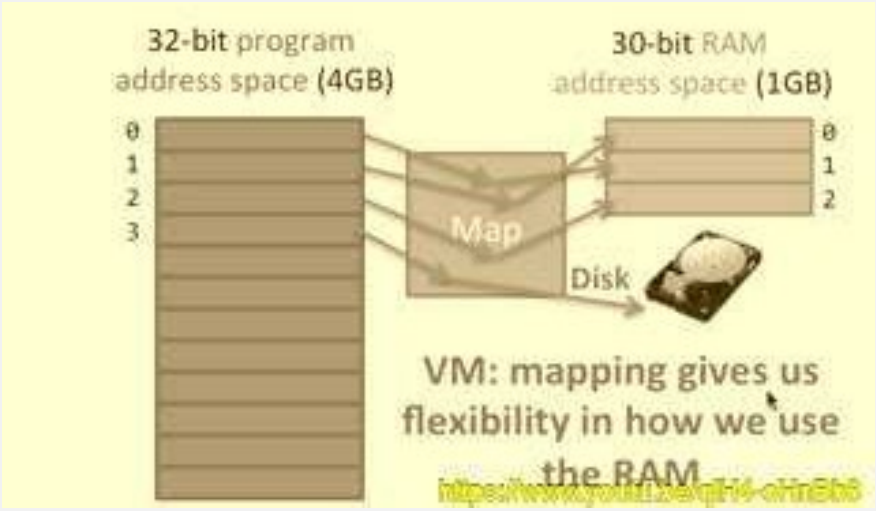
主要的硬件支持有：

(1) 请求分页的页表机制。

(2) 缺页中断机构。

(3) 地址变换机构。

2) 实现请求分页的软件



5.1.3 虚拟存储器的实现方法

2. 请求分段系统

1) 硬件支持

主要的硬件支持有：

(1) 请求分段的段表机制。

(2) 缺段中断机构。

(3) 地址变换机构。

2) 软件支持

第五章 虚拟存储器

5.2 请求分页存储管理方式

5.2.1 请求分页中的硬件支持

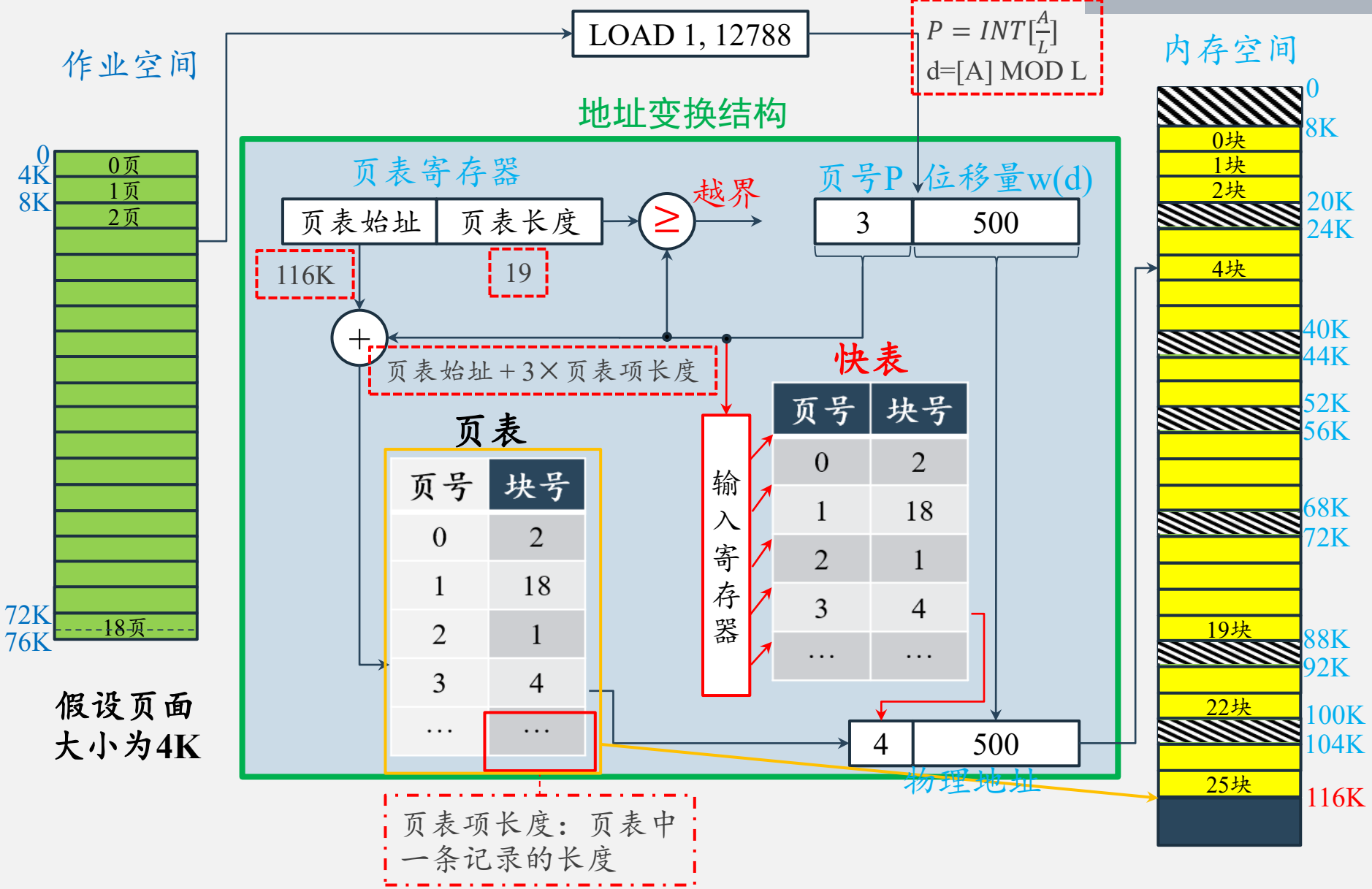
为了实现请求分页，系统必须提供一定的硬件支持。计算机系统除了要求一定容量的内存和外存外，还需要有：

（1）请求页表机制；

（2）缺页中断机构；

（3）地址变换机构。

基本分页存储管理


作业空间
LOAD 1, 12788
$P = INT[\frac{A}{L}]$
d=[A] MOD L
内存空间
地址变换结构
页表寄存器
页表始址
页表长度
116K
19
越界
页号P
位移量w(d)
3
500
页表始址+3×页表项长度
快表
页表
输入寄存器
页号
块号
0
2
1
18
2
1
3
4
4
500
物理地址
0页
1页
2页
18页
0
4K
8K
72K
76K
0块
1块
2块
4块
19块
22块
25块
0
8K
20K
24K
40K
44K
52K
56K
68K
72K
88K
92K
100K
104K
116K
假设页面大小为4K
页表项长度：页表中一条记录的长度

5.2.1 请求分页中的硬件支持

1. 请求页表机制

在请求分页系统中需要的主要数据结构是**请求页表**，其基本作用仍然是将用户地址空间中的逻辑地址映射为内存空间中的物理地址。为了满足页面换进换出的需要，在请求页表中又增加了四个字段。这样，在请求分页系统中的每个页表应含以下诸项：

页号	物理块号	状态位 P	访问字段 A	修改位 M	外存地址

5.2.1 请求分页中的硬件支持

2. 缺页中断机构

(1) 在指令执行期间产生和处理中断信号。

(2) 一条指令在执行期间可能产生多次缺页中断。

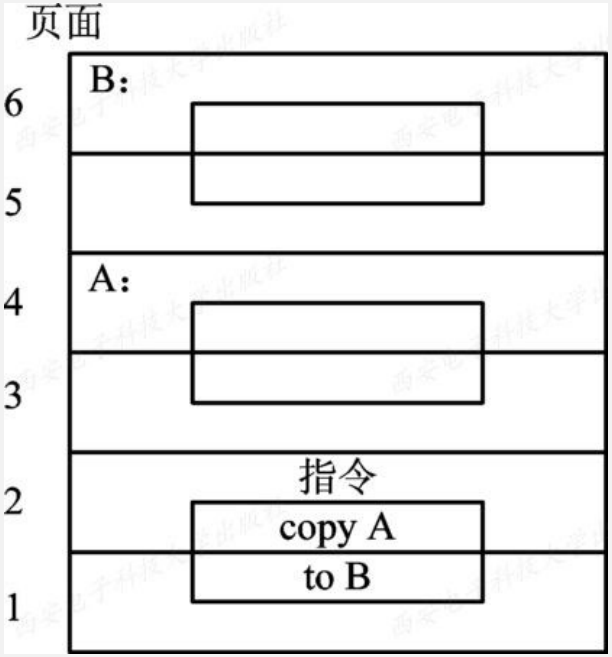


图5-1 涉及6次缺页中断的指令

5.2.1 请求分页中的硬件支持

3. 地址变换机构

请求分页系统中的地址变换机构是在分页系统地址变换机构的基础上，为实现虚拟存储器，再增加了某些功能所形成的，如产生和处理缺页中断，以及从内存中换出一页的功能等等。图5-2示出了请求分页系统中的地址变换过程。

5.2.1 请求分页中的硬件支持

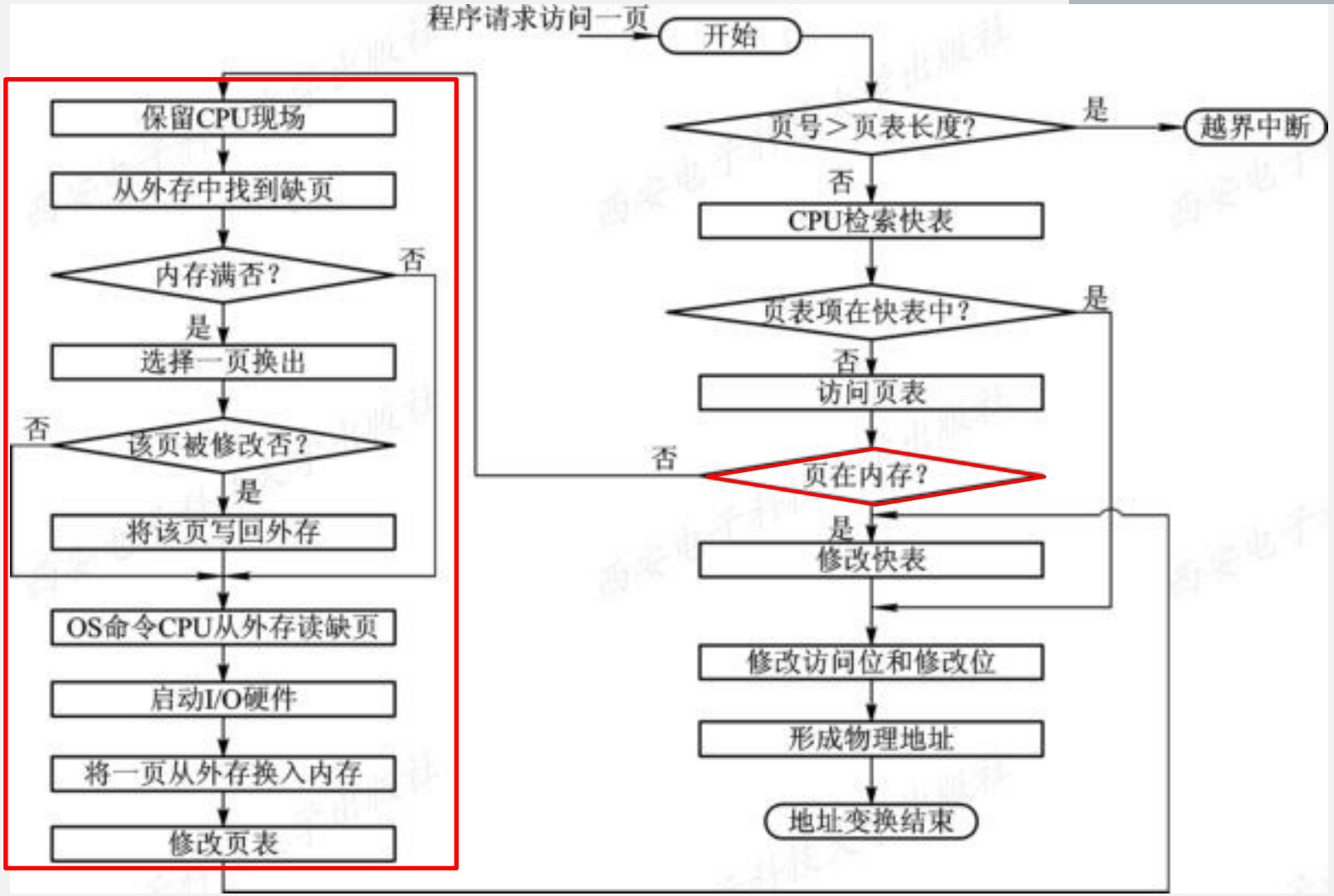


图5-2　请求分页中的地址变换过程

5.2.2 请求分页中的内存分配

1. 最小物理块数的确定

一个显而易见的事实是，随着为每个进程所分配的物理块的减少，将使进程在执行中的缺页率上升，从而会降低进程的执行速度。为使进程能有效地工作，应为它分配一定数目的物理块，但这并不是最小物理块数的概念。

最小物理块数是指能保证进程正常运行所需要的最小物理块数，当系统为进程分配的物理块数少于此值时，进程将无法运行。

5.2.2 请求分页中的内存分配

2. 内存分配策略

在请求分页系统中，可采取两种内存分配策略，即固定和可变分配策略。在进行置换时，也可采取两种策略，即全局置换和局部置换。于是可组合出以下三种适用的策略。

1) 固定分配局部置换(Fixed Allocation，Local Replacement)

2) 可变分配全局置换(Variable Allocation，Global Replacement)

3) 可变分配局部置换(Variable Allocation，Local Replacement)

5.2.2 请求分页中的内存分配

3. 物理块分配算法

在采用固定分配策略时，如何将系统中可供分配的所有物理块分配给各个进程，可采用下述几种算法：

(1) 平均分配算法，即将系统中所有可供分配的物理块平均分配给各个进程。

(2) 按比例分配算法，即根据进程的大小按比例分配物理块。如果系统中共有n个进程，每个进程的页面数为Si，则系统中各进程页面数的总和为：

$$S = \sum_{i=1}^{n} S_i$$

5.2.2 请求分页中的内存分配

3. 物理块分配算法

又假定系统中可用的物理块总数为m，则每个进程所能分到的物理块数为b_i可由下式计算：

$$b_i = \frac{S_i}{S} \times m$$

这里，b_i应该取整，它必须大于最小物理块数。

5.2.2 请求分页中的内存分配

3. 物理块分配算法

(3) 考虑优先权的分配算法。在实际应用中，为了照顾到重要的、紧迫的作业能尽快地完成，应为它分配较多的内存空间。通常采取的方法是把内存中可供分配的所有物理块分成两部分：一部分按比例地分配给各进程；另一部分则根据各进程的优先权进行分配，为高优先进程适当地增加其相应份额。在有的系统中，如重要的实时控制系统，则可能是完全按优先权为各进程分配其物理块的。

5.2.3 页面调入策略

为使进程能够正常运行，必须事先将要执行的那部分程序和数据所在的页面调入内存。现在的问题是：

(1) 系统应在何时调入所需页面；

(2) 系统应从何处调入这些页面；

(3) 是如何进行调入的。

5.2.3 页面调入策略

1. 何时调入页面

(1) 预调页策略。

如果进程的许多页是存放在外存的一个连续区域中，一次调入若干个相邻的页面会比一次调入一页更高效。因此，考虑采用一种以预测为基础的预调页策略，将那些预计不久后便会访问的页面先调入内存。目前预调页成功率仅约50%。

(2) 请求调页策略。

当进程在运行中需要访问某部分程序和数据时，若发现其所在的页面不在内存，便立即提出请求，由OS将其所需页面调入内存。目前大多采用此策略。

5.2.3 页面调入策略

2. 从何处调入页面

(1) 系统拥有足够的对换区空间，这时可以全部从对换区调入所需页面，以提高调页速度。

(2) 系统缺少足够的对换区空间，这时凡是不会被修改的文件，都直接从文件区调入；而当换出这些页面时，由于它们未被修改，则不必再将它们重写到磁盘(换出)，以后再调入时，仍从文件区直接调入。但对于那些可能被修改的部分，在将它们换出时便须调到对换区，以后需要时再从对换区调入。

(3) UNIX方式。

5.2.3 页面调入策略

3. 页面调入过程

每当程序所要访问的页面未在内存时(存在位为“0”),便向CPU发出一缺页中断，中断处理程序首先保留CPU环境，分析中断原因后转入缺页中断处理程序。

通过查找页表得到该页在外存的物理块后:

(1) 如果此时内存能够容纳新页：启动磁盘I/O，将所缺之页调入内存，然后修改页表和快表;

(2) 如果内存已满：须先按照某种**页面置换算法**，从内存中选出一页准备换出；如果该页未被修改过，可不必将该页写回磁盘；否则，必须将它写回磁盘；然后再把所需的页调入内存，修改页表和快表。

5.2.3 页面调入策略

4. 缺页率

假设一个进程的逻辑空间为n页，系统为其分配的内存物理块数为m(m≤n)。如果在进程的运行过程中，访问页面成功(即所访问页面在内存中)的次数为S，访问页面失败(即所访问页面不在内存中，需要从外存调入)的次数为F，则该进程总的页面访问次数为A＝S＋F，那么该进程在其运行过程中的缺页率即为：

$$f = \frac{F}{A}$$

5.2.3 页面调入策略

4. 缺页率

缺页率受以下几个因素影响：

(1) 页面大小；

(2) 进程所分配的物理块的数目；

(3) 页面置换算法；

(4) 程序固有特性：局部化程度越高，缺页程度越低。

5.2.3 页面调入策略

4. 缺页率

事实上，在缺页中断处理时，当由于空间不足，需要置换部分页面到外存时，选择被置换页面还需要考虑到置换的代价，如页面是否被修改过。没有修改过的页面可以直接放弃，而修改过的页面则必须进行保存，所以处理这两种情况时的时间也是不同的。假设被置换的页面被修改的概率是β，其缺页中断处理时间为t_a，被置换页面没有被修改的缺页中断时间为t_b，那么，缺页中断处理时间的计算公式为

$$t=\beta\times t_a+(1—\beta)\times t_b$$

请求分页存储管理方式——虚拟存储器实现

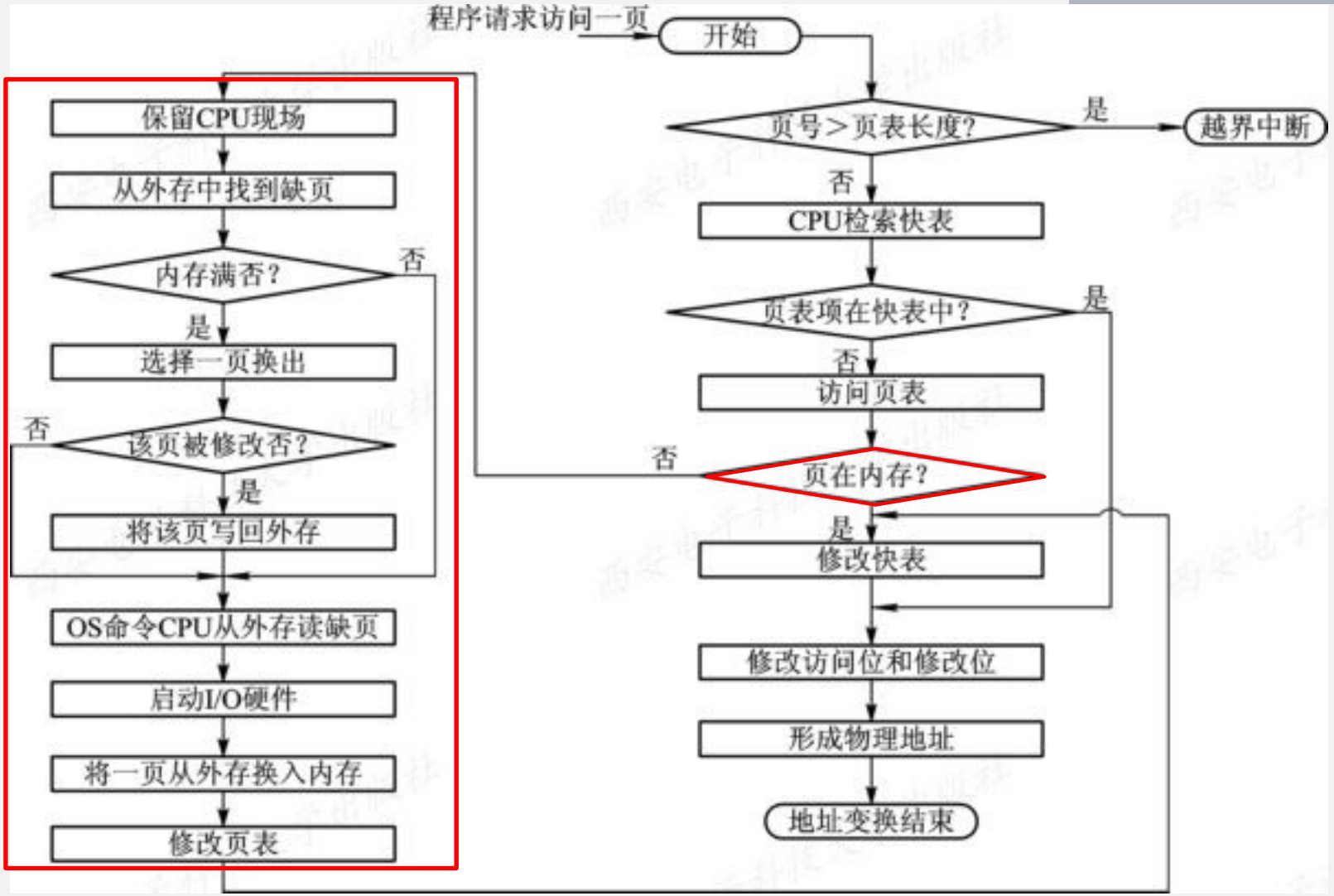


图5-2　请求分页中的地址变换过程

第五章 虚拟存储器

5.3 页面置换算法

5.3 页面置换算法

在进程运行过程中，若其所要访问的页面不在内存，而需把它们调入内存，但内存已无空闲空间时，为了保证该进程能正常运行，系统必须从内存中调出一页程序或数据送到磁盘的对换区中。但应将哪个页面调出，须根据一定的算法来确定。通常，把选择换出页面的算法称为页面置换算法(Page-Replacement Algorithms)。置换算法的好坏将直接影响到系统的性能。

5.3.1 最佳置换算法和先进先出置换算法

1. 最佳(Optimal)置换算法

最佳置换算法是由Belady于1966年提出的一种理论上的算法。其所选择的被淘汰页面将是以后永不使用的，或许是在最长(未来)时间内不再被访问的页面。采用最佳置换算法通常可保证获得最低的缺页率。但由于人们目前还无法预知，一个进程在内存的若干个页面中，哪一个页面是未来最长时间内不再被访问的，**因而该算法是无法实现的**，但可以利用该算法去评价其它算法。

5.3.1 最佳置换算法和先进先出置换算法

1. 最佳(Optimal)置换算法

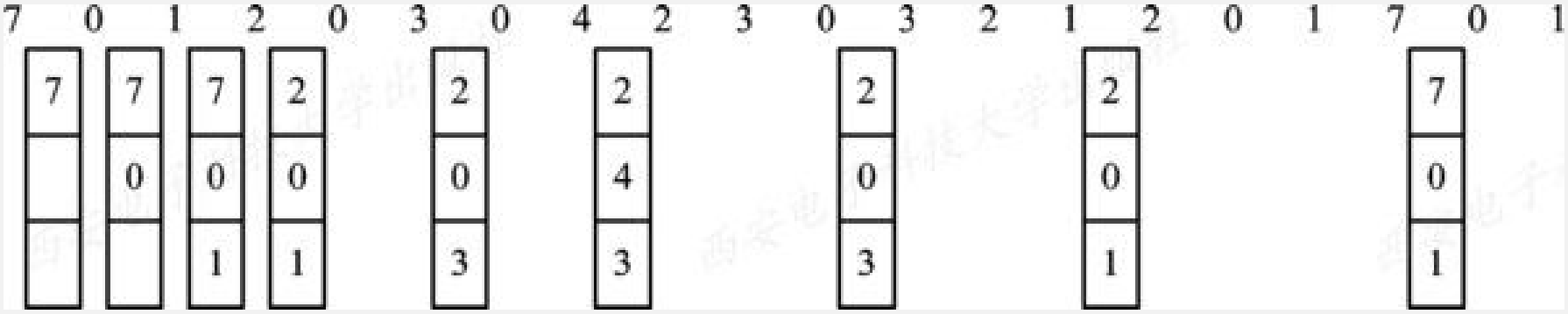


图5-3 利用最佳页面置换算法时的置换图

5.3.1 最佳置换算法和先进先出置换算法

2. 先进先出(FIFO)页面置换算法

FIFO算法是最早出现的置换算法。该算法总是淘汰最先进入内存的页面，即选择在内存中驻留时间最久的页面予以淘汰。该算法实现简单，只需把一个进程已调入内存的页面按先后次序链接成一个队列，并设置一个指针，称为替换指针，使它总是指向最老的页面。但该算法与进程实际运行的规律不相适应，因为在进程中，有些页面经常被访问，比如，含有全局变量、常用函数、例程等的页面，FIFO算法并不能保证这些页面不被淘汰。

5.3.1 最佳置换算法和先进先出置换算法

2. 先进先出(FIFO)页面置换算法

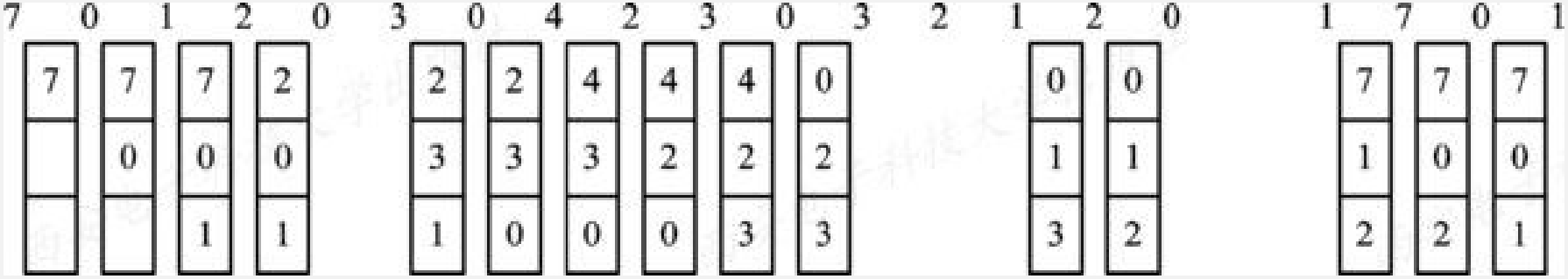


图5-4 利用FIFO置换算法时的置换图

5.3.2 最近最久未使用和最少使用置换算法

1. LRU(Least Recently Used)置换算法的描述

FIFO置换算法的性能之所以较差，是因为它所依据的条件是各个页面调入内存的时间，而页面调入的先后并不能反映页面的使用情况。最近最久未使用(LRU)的页面置换算法是根据页面调入内存后的使用情况做出决策的。

5.3.2 最近最久未使用和最少使用置换算法

1. LRU(Least Recently Used)置换算法的描述

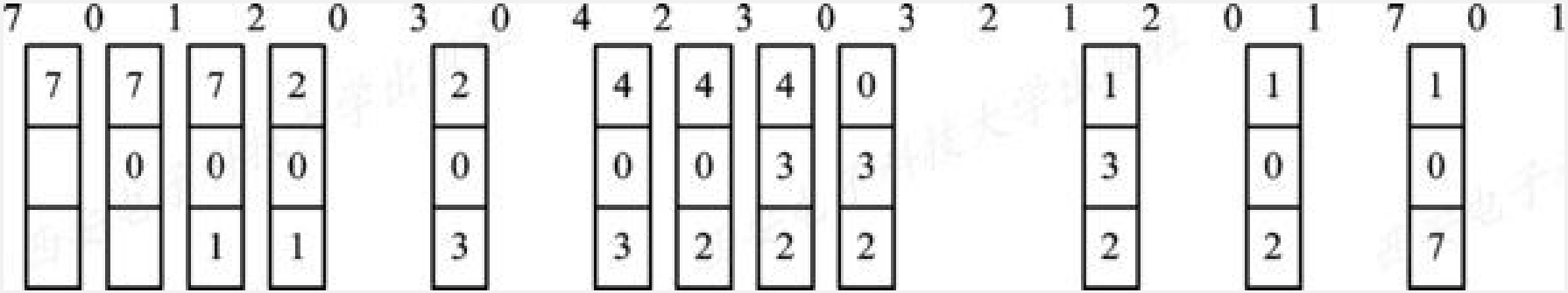


图5-5 LRU页面置换算法

5.3.2 最近最久未使用和最少使用置换算法

2. LRU置换算法的硬件支持

1) 寄存器

为了记录某进程在内存中各页的使用情况，须为每个在内存中的页面配置一个移位寄存器，可表示为

$$R = R_{n-1}R_{n-2}R_{n-3} \cdots R_2R_1R_0$$

当进程访问某物理块时，要将相应寄存器的R_{n-1}位置成1。此时，定时信号将每隔一定时间(例如100 ms)将寄存器右移一位。如果我们把n位寄存器的数看作是一个整数，那么，具有最小数值的寄存器所对应的页面，就是最近最久未使用的页面。

5.3.2 最近最久未使用和最少使用置换算法

2. LRU置换算法的硬件支持

实页 \ R	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	0	1	0	1	0	0	1	0
2	1	0	1	0	1	1	0	0
3	0	0	0	0	0	1	0	0
4	0	1	1	0	1	0	1	1
5	1	1	0	1	0	1	1	0
6	0	0	1	0	1	0	1	1
7	0	0	0	0	0	1	1	1
8	0	1	1	0	1	1	0	1

图5-5 LRU页面置换算法

5.3.2 最近最久未使用和最少使用置换算法

2. LRU置换算法的硬件支持

2) 栈

可利用一个特殊的栈保存当前使用的各个页面的页面号。每当进程访问某页面时，便将该页面的页面号从栈中移出，将它压入栈顶。因此，栈顶始终是最新被访问页面的编号，而栈底则是最近最久未使用页面的页面号。假定现有一进程，它分有五个物理块，所访问的页面的页面号序列为：

4，7，0，7，1，0，1，2，1，2，6

5.3.2 最近最久未使用和最少使用置换算法

2. LRU置换算法的硬件支持

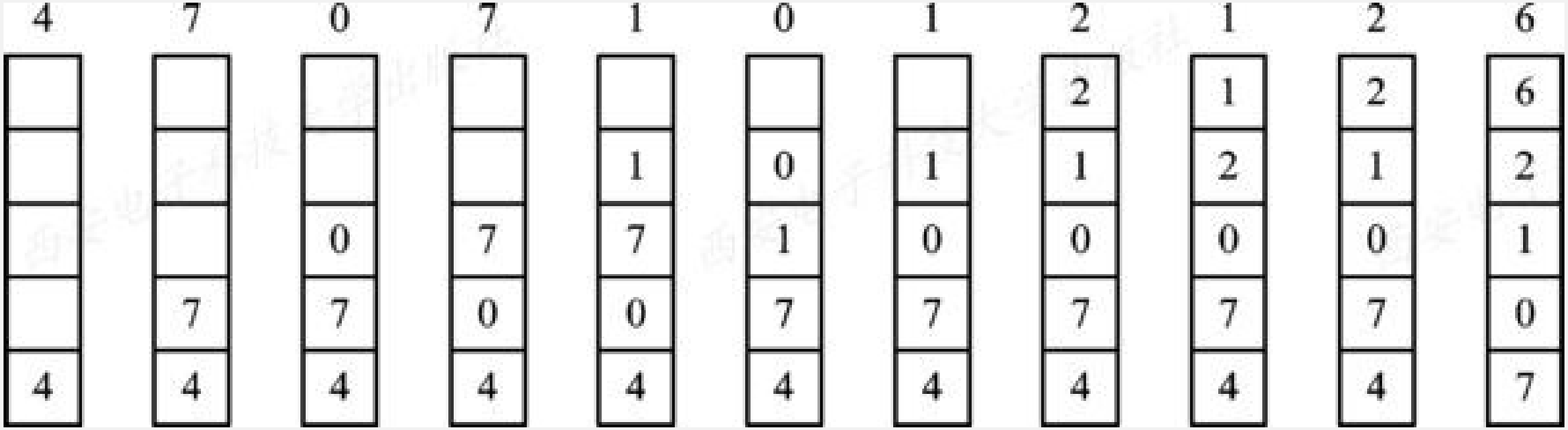


图5-7 用栈保存当前使用页面时栈的变化情况

5.3.2 最近最久未使用和最少使用置换算法

3. 最少使用(Least Frequently Used，LFU)置换算法

在采用LFU算法时，应为在内存中的每个页面设置一个移位寄存器，用来记录该页面被访问的频率。该置换算法选择在最近时期使用最少的页面作为淘汰页。

5.3.3 Clock置换算法

1. 简单的Clock置换算法

虽然LRU算法比较好，但它要求有较多的硬件支持，实现成本较高，实际中大多采用LRU的近似算法，如CLOCK算法。当利用简单Clock算法时，只需为每页设置一位访问位，再将内存中的所有页面都通过链接指针链接成一个循环队列。

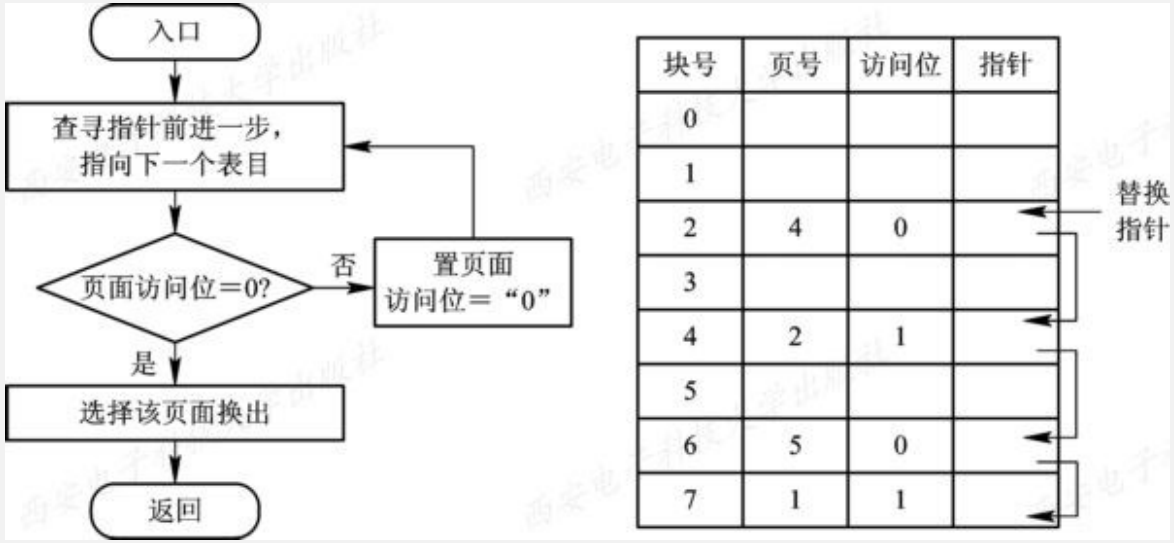


图5-8 简单Clock置换算法的流程和示例

5.3.3 Clock置换算法

2. 改进型Clock置换算法

在将一个页面换出时，如果该页已被修改过，便须将该页重新写回到磁盘上；但如果该页未被修改过，则不必将它拷回磁盘。换而言之，对于修改过的页面，在换出时所付出的开销比未修改过的页面大，或者说，置换代价大。在改进型Clock算法中，除须考虑页面的使用情况外，还须再增加一个因素——置换代价。

5.3.4 页面缓冲算法(Page Buffering Algorithm，PBA)

1. 影响页面换进换出效率的若干因素

(1) 页面置换算法。

(2) 写回磁盘的频率。

(3) 读入内存的频率。

5.3.4 页面缓冲算法(Page Buffering Algorithm，PBA)

2. 页面缓冲算法PBA

在系统中，内存分配策略采用可变分配和局部置换方式，系统为每个进程分配一定数目的的物理块，系统自己保留一部分空闲物理块。

1) 空闲页面链表

该链表是一个空闲的物理块表，是系统掌握的空闲物理块，用于分配给频繁发生缺页的进程，以降低该进程的缺页率。当进程需要读入一个页面时，可将链表中的第一个物理块给它使用。当有一个未被修改的页要换出内存时，实际上不将它换出外存，而是把它们所在的物理块挂载空闲链表的表尾，它是有数据的，下次再使用时可直接取下，不用读外存。

5.3.4 页面缓冲算法(Page Buffering Algorithm，PBA)

2. 页面缓冲算法PBA

2) 修改页面链表

由已修改的页面所形成的链表，目的是为了减少已修改页面换出的次数。当进程需要将一个已修改的页面换出时，系统并不立即把它换出到外存上，而是将它所在的物理块挂在修改页面的链表的末尾。等下次使用时，若该页面依然存在，则可直接取下使用。这样做的目的是：降低将已修改页面写回磁盘的频率，降低将磁盘内容读入内存的频率。

5.3.4 页面缓冲算法(Page Buffering Algorithm，PBA)

2. 页面缓冲算法PBA

PBA算法的主要特点是：

① 显著地降低了页面换进、换出的频率，使磁盘I/O的操作次数大为减少，因而减少了页面换进、换出的开销；

② 正是由于换入换出的开销大幅度减小，才能使其采用一种较简单的置换策略，如先进先出(FIFO)算法，它不需要特殊硬件的支持，实现起来非常简单。

1) 空闲页面链表

2) 修改页面链表

5.3.5 访问内存的有效时间

与基本分页存储管理方式不同，在请求分页管理方式中，内存有效访问时间不仅要考虑访问页表和访问实际物理地址数据的时间，还必须要考虑到缺页中断的处理时间。

(P181页)

(1) 被访问页在内存中，且其对应的页表项在快表中。

$$EAT = \lambda + t$$

(2) 被访问页在内存中，且其对应的页表项不在快表中。

$$EAT = \lambda + t + \lambda + t = 2(\lambda + t)$$

λ—访问(读取或更新)快表，t—访问(读取或更新)页表

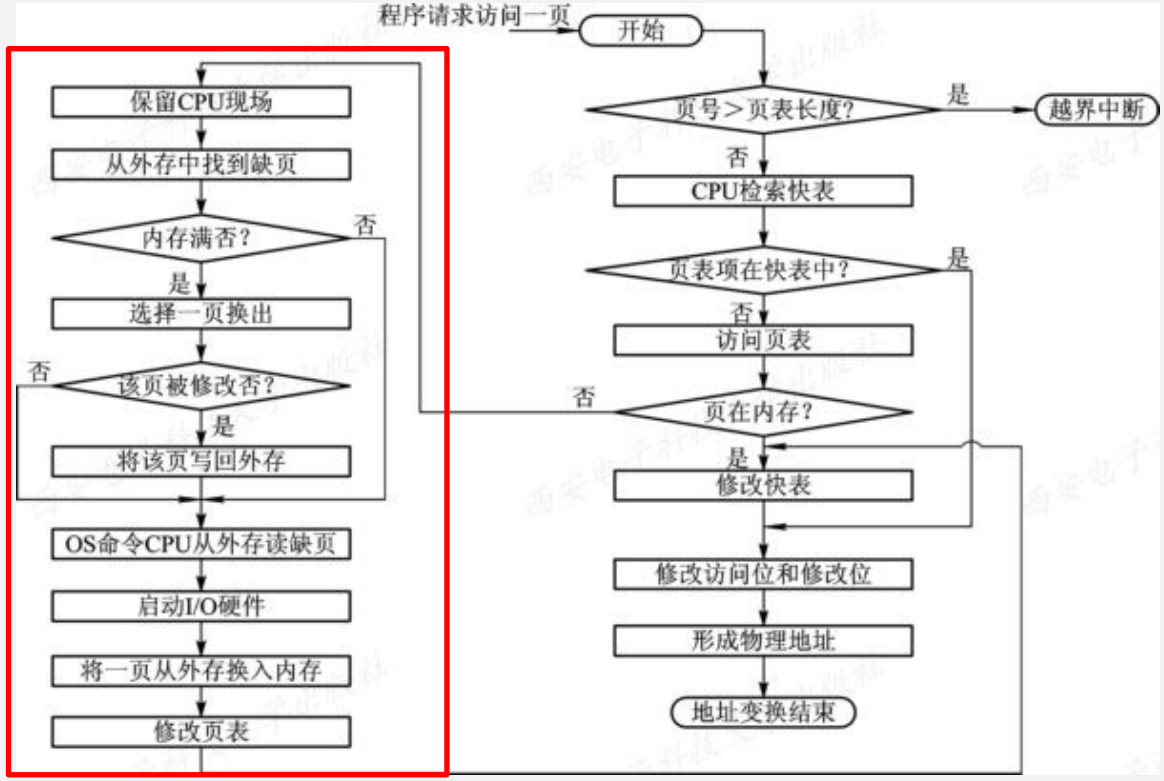
5.3.5 访问内存的有效时间

(3) 被访问页不在内存中。

$$EAT = \lambda + t + \varepsilon + \lambda + t = \varepsilon + 2(\lambda + t)$$

λ—访问(读取或更新)快表，t—访问(读取或更新)页表， ε—缺页中断处理

缺页中断处理



第五章 虚拟存储器

5.4 “抖动”与工作集

5.4 “抖动”与工作集

由于请求分页式虚拟存储器系统的性能优越，在正常运行情况下，它能有效地减少内存碎片，提高处理机的利用率和吞吐量，故是目前最常用的一种系统。但如果在系统中运行的进程太多，进程在运行中会频繁地发生缺页情况，这又会对系统的性能产生很大的影响，故还须对请求分页系统的性能做简单的分析。

5.4.1 多道程序度与“抖动”

1. 多道程序度与处理机的利用率

由于虚拟存储器系统能从逻辑上扩大内存，这时，只需装入一个进程的部分程序和数据便可开始运行，故人们希望在系统中能运行更多的进程，即增加多道程序度，以提高处理机的利用率。

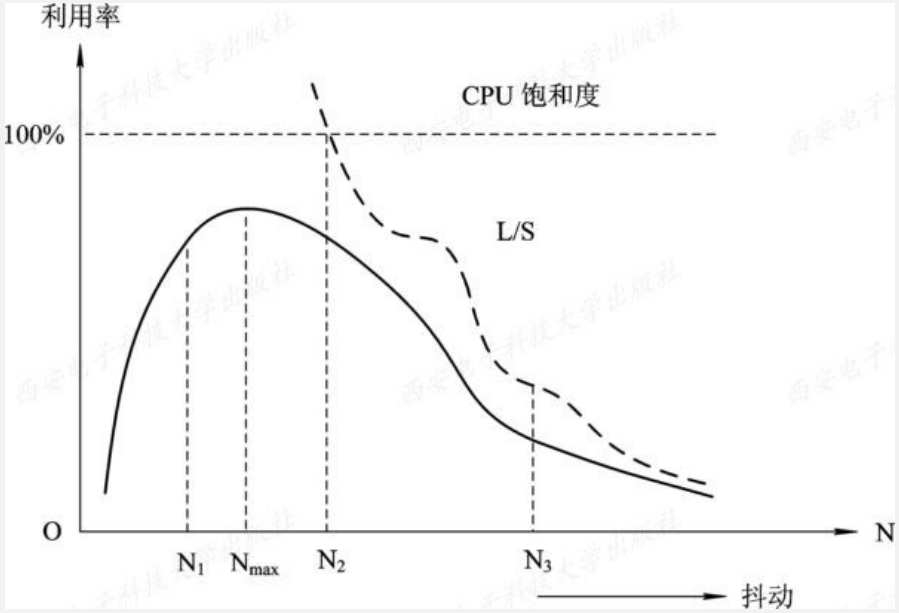


图5-9 处理机的利用率

5.4.1 多道程序度与“抖动”

2. 产生“抖动”的原因

发生“抖动”的根本原因是，同时在系统中运行的进程太多，由此分配给每一个进程的物理块太少，不能满足进程正常运行的基本要求，致使每个进程在运行时，频繁地出现缺页，必须请求系统将所缺之页调入内存。这会使得在系统中排队等待页面调进/调出的进程数目增加。显然，对磁盘的有效访问时间也随之急剧增加，造成每个进程的大部分时间都用于页面的换进/换出，而几乎不能再去做任何有效的工作，从而导致发生处理机的利用率急剧下降并趋于0的情况。我们称此时的进程是处于“抖动”状态。

第五章 虚拟存储器

5.5 请求分段存储管理方式

5.5.1 请求分段中的硬件支持

为了实现请求分段式存储管理，应在系统中配置多种硬件机构，以支持快速地完成请求分段功能。与请求分页系统相似，在请求分段系统中所需的硬件支持有段表机制、缺段中断机构，以及地址变换机构。

5.5.1 请求分段中的硬件支持

1. 请求段表机制

在请求分段式管理中所需的主要数据结构是请求段表。在该表中除了具有请求分页机制中有的访问字段A、修改位M、存在位P和外存始址四个字段外，还增加了存取方式字段和增补位。这些字段供程序在调进、调出时参考。下面给出请求分段的段表项。

段名	段长	段基址	存取方式	访问字段 A	修改位 M	存在位 P	增补位	外存始址

存取方式：如果字段为两位，表示只执行、只读和允许读写；

增补位：本段在运行过程中是否做过动态增长。

5.5.1 请求分段中的硬件支持

2. 缺段中断机构

在请求分段系统中采用的是请求调段策略。每当发现运行进程所要访问的段尚未调入内存时，便由缺段中断机构产生一缺段中断信号，进入OS后，由缺段中断处理程序将所需的段调入内存。与缺页中断机构类似，缺段中断机构同样需要在一条指令的执行期间产生和处理中断，以及在一条指令执行期间，可能产生多次缺段中断。但由于分段是信息的逻辑单位，因而不可能出现一条指令被分割在两个分段中，和一组信息被分割在两个分段中的情况。缺段中断的处理过程如图5-12所示。

5.5.1 请求分段中的硬件支持

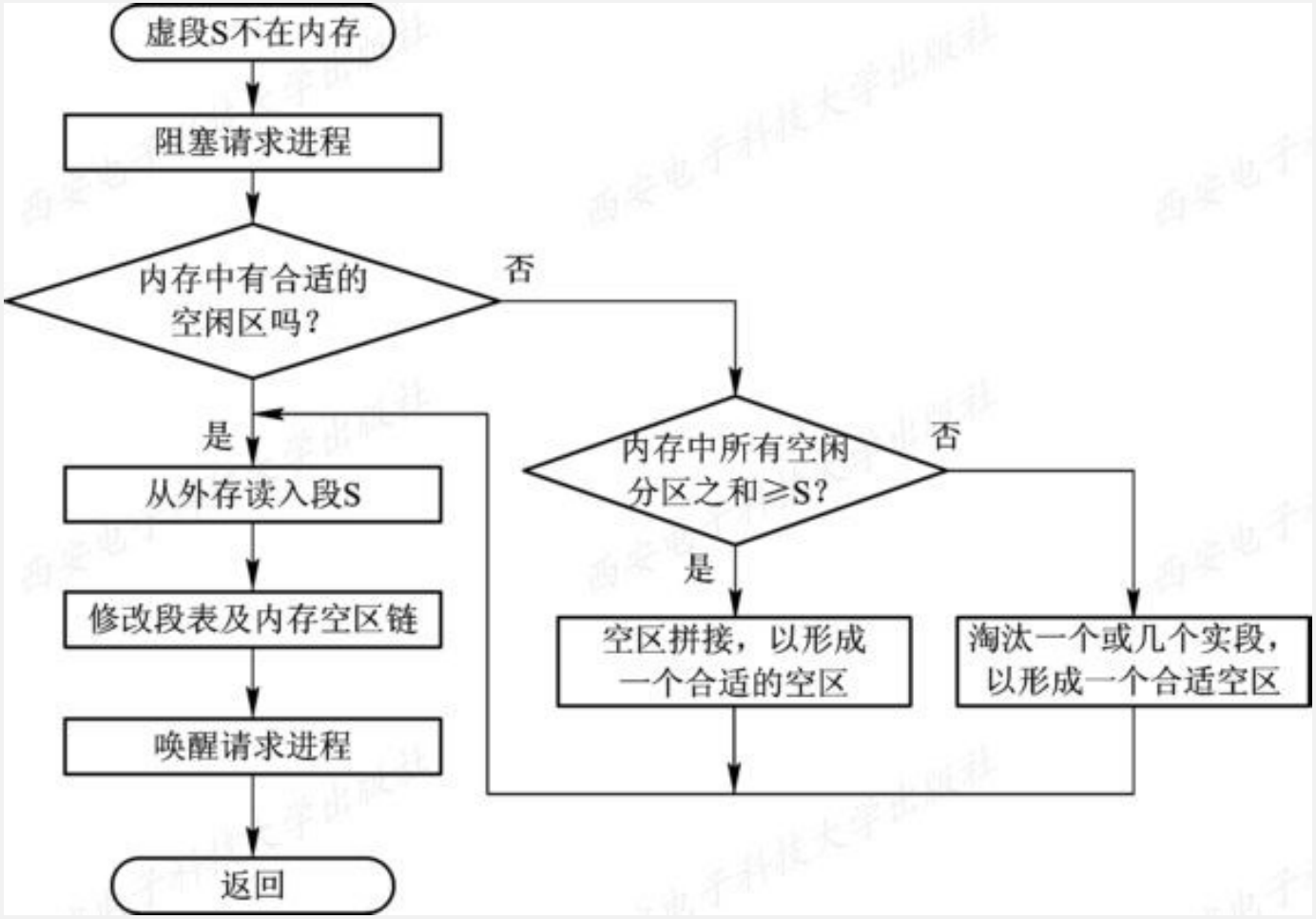


图5-12 请求分段系统中的中断处理过程

5.5.1 请求分段中的硬件支持

3. 地址变换机构

请求分段系统中的地址变换机构是在分段系统地址变换机构的基础上形成的。因为被访问的段并非全在内存，所以在地址变换时，若发现所要访问的段不在内存，必须先将所缺的段调入内存，并修改段表，然后才能再利用段表进行地址变换。为此，在地址变换机构中又增加了某些功能，如缺段中断的请求及处理等。图5-13示出了请求分段系统的地址变换过程。

5.5.1 请求分段中的硬件支持

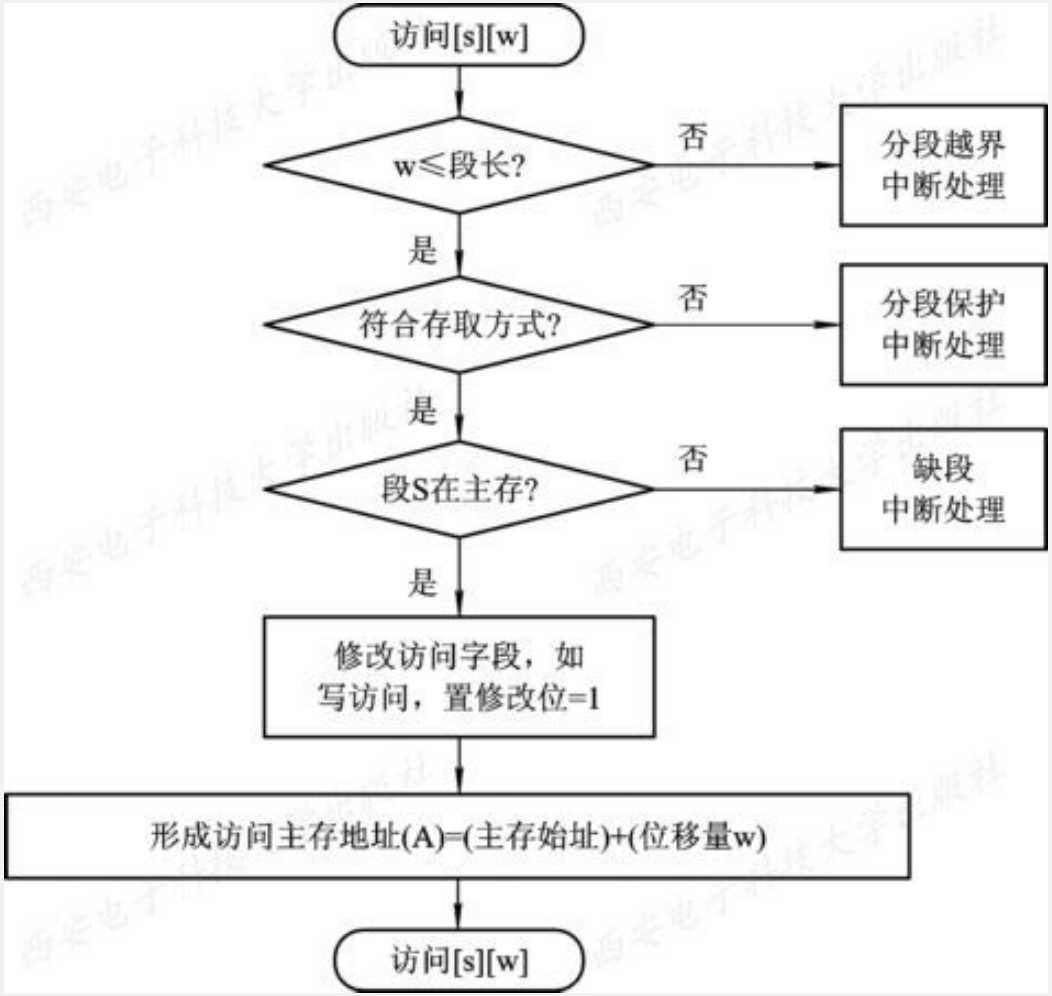


图5-13 请求分段系统的地址变换过程

5.5.2 分段的共享与保护

1. 共享段表

为实现分段共享，在系统中配置一张共享段表，如下。

(1) 共享进程计数count。

(2) 存取控制字段。

(3) 段号。

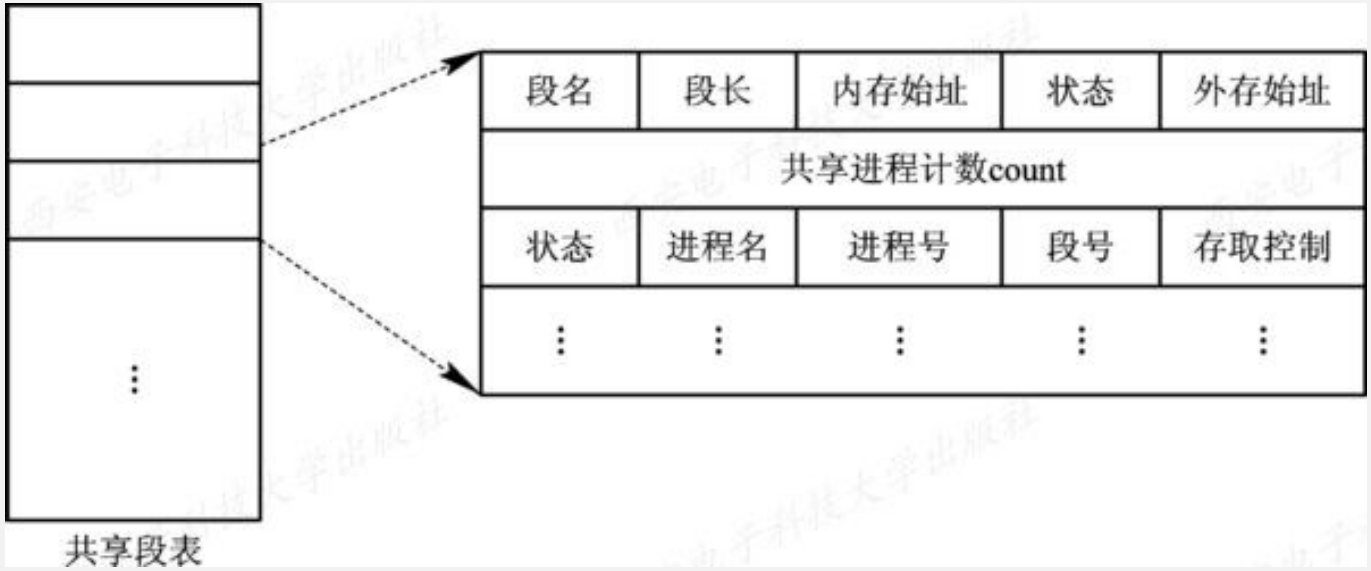


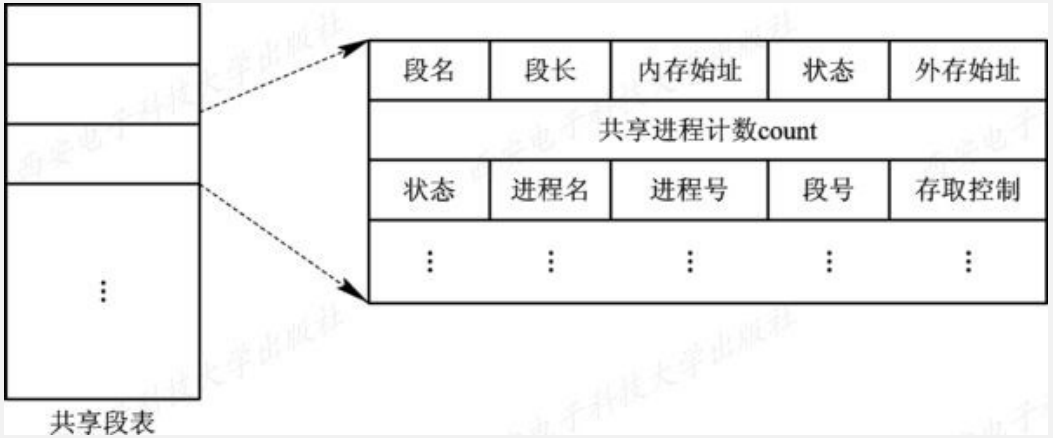
图5-14 共享段表项

5.5.2 分段的共享与保护

2. 共享段的分配与回收

1) 共享段的分配

对第一个请求使用该共享段的进程，由系统为该共享段分配一物理区，再把共享段调入该区，同时将该区的始址填入请求进程的段表的相应项中，还须在共享段表中增加一项，填写请求使用该共享段的进程名、段号、和存取控制等有关数据，把count置为1。

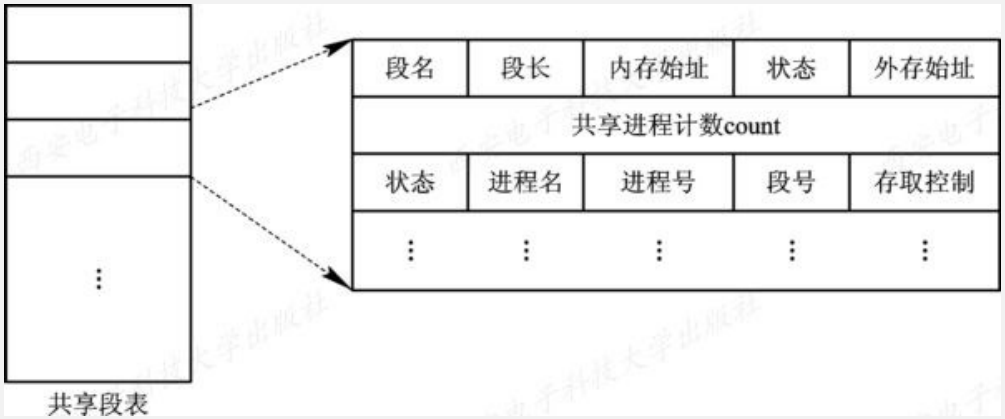


5.5.2 分段的共享与保护

2. 共享段的分配与回收

1) 共享段的分配

当有其他进程需要调用该共享段时，只需在调用进程的段表中增加一表项，填写共享段的物理地址。在共享段的段表中增加一个表项，填上调用进行的信息，执行count=count+1操作。



5.5.2 分段的共享与保护

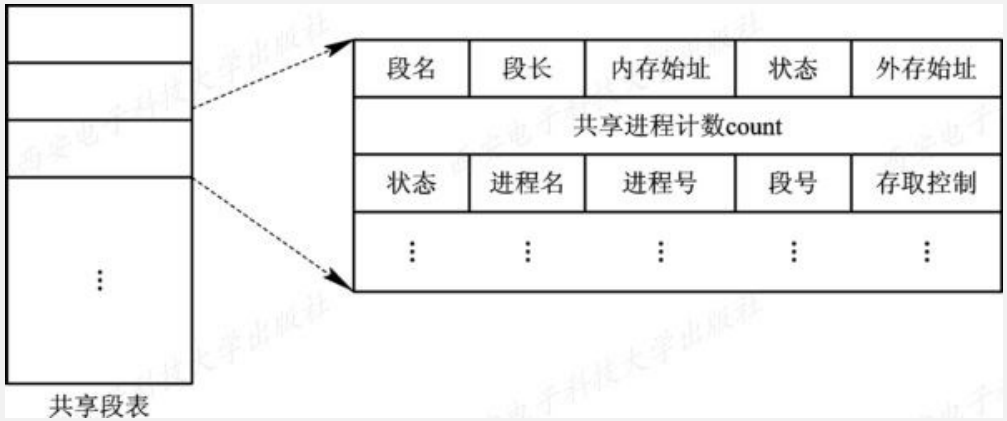
2. 共享段的分配与回收

2) 共享段的回收

撤消该进程段表中共享段所对应的表项，执行count = count – 1。

若结果为0，须由系统回收该共享段的物理内存，取消在共享段表中该段所对应的表项。

否则，只是取消调用者进程在共享段表中的有关记录。



5.5.2 分段的共享与保护

3. 分段保护

在分段系统中，由于每个分段在逻辑上是相对独立的，因而比较容易实现信息保护。目前，常采用以下几种措施来确保信息的安全。

1) 越界检查

2) 存取控制检查

3) 环保护机构

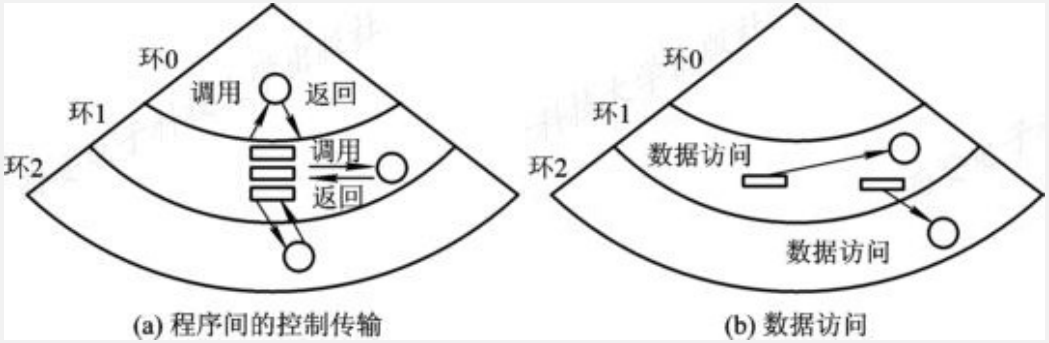


图5-15 环保护机构

谢谢大家！ Q & A