操作系统

第四章 存储器管理

存储管理的功能

- 存储器是系统的重要资源之一。任何程序，数据以及各种控制用的数据结构都必须占用一定的存储空间。
- 存储器由内存（primary storage ）和外存（secondary storage）组成。
- 内存由顺序编址的物理单元组成。
- CPU 要启动输入输出设备后才能使外存与内存交换信息。

第四章 存储器管理

4.1 程序的装入和链接

4.2 连续分配方式

4.3 基本分页存储管理方式

4.4 基本分段存储管理方式

4.5 虚拟存储器的基本概念

4.6 请求分页存储管理方式

4.7 页面置换算法

4.8 请求分段存储管理方式

4.1 程序的装入和链接

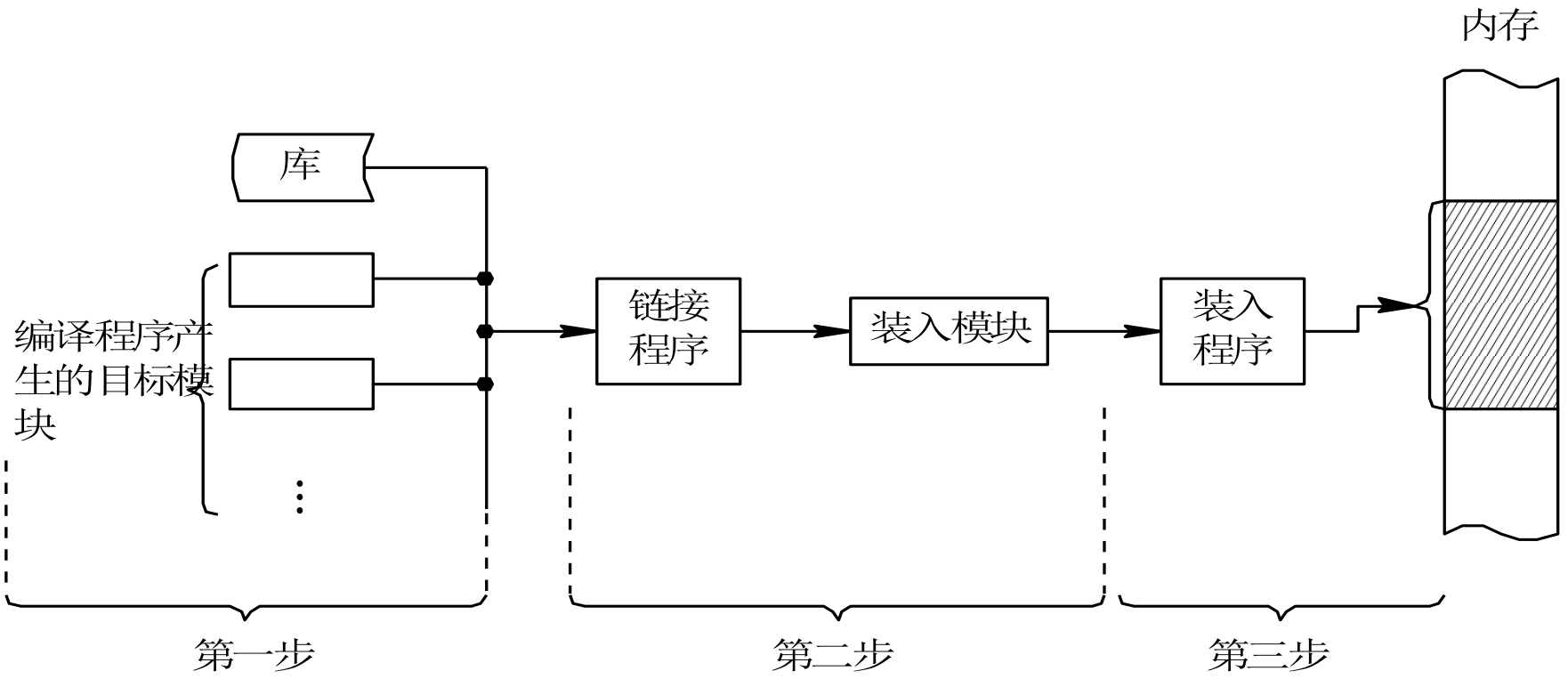
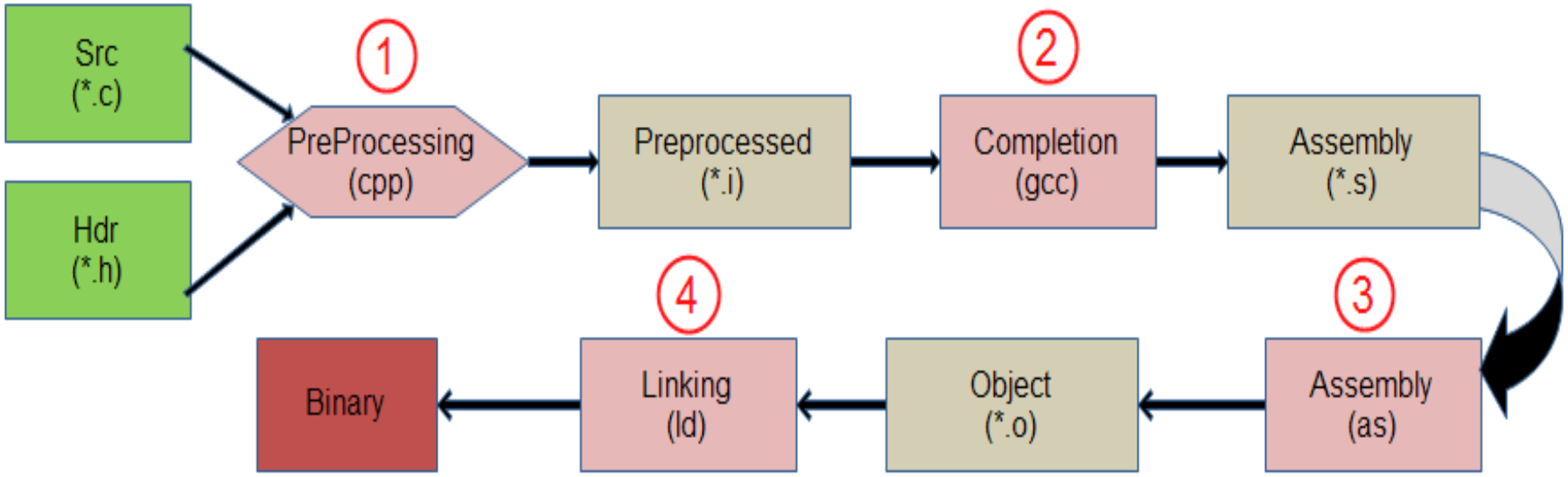
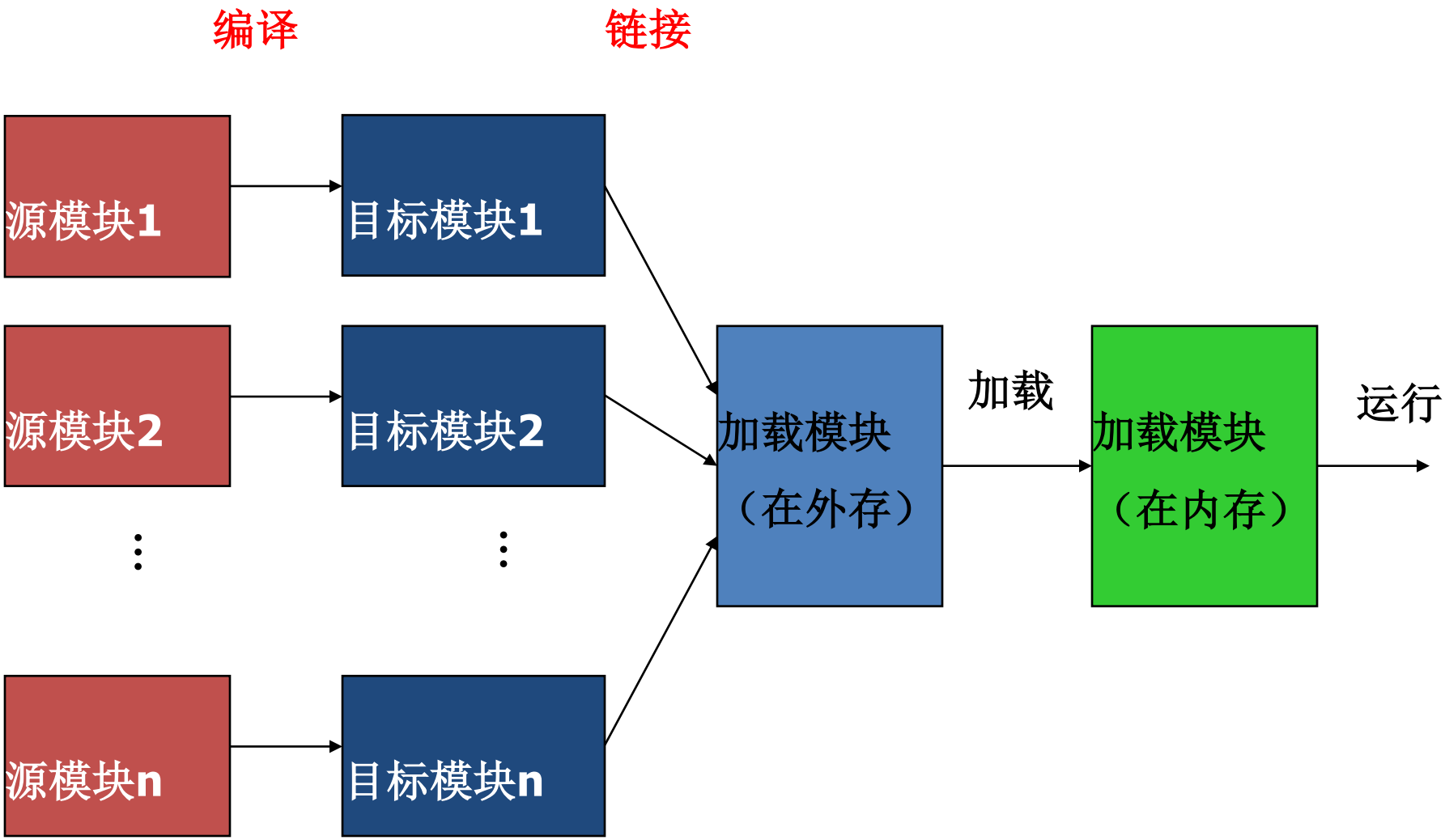


图 4-1 对用户程序的处理步骤

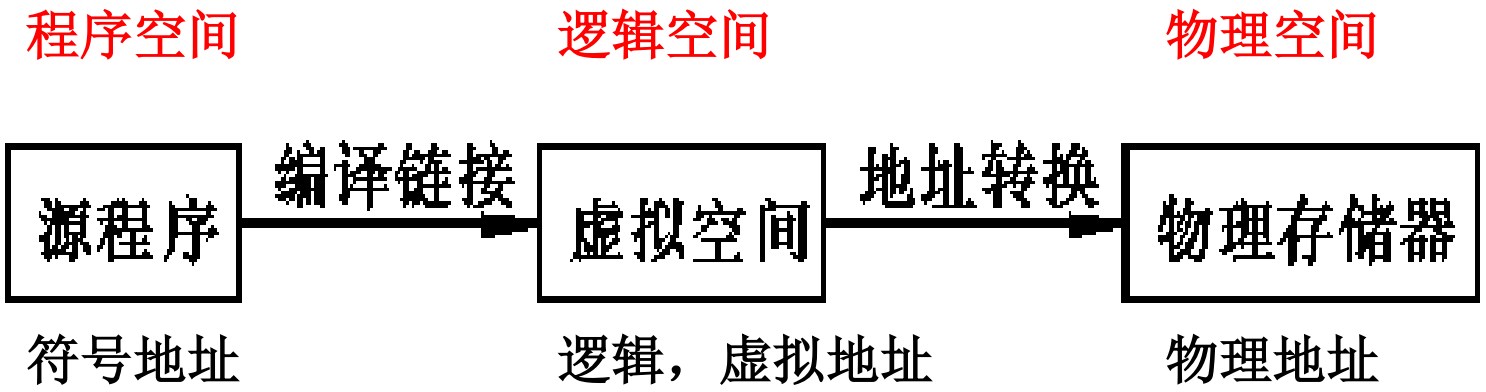
源程序变成可执行程序的过程



程序的加载和链接



地址变换与物理存储器



4.1.1 程序的装入

1.绝对装入方式(Absolute Loading Mode)

程序中所使用的绝对地址，既可在编译或汇编时给出， 也可由程序员直接赋予。

2. 可重定位装入方式(Relocation Loading Mode)

3. 动态运行时装入方式(Denamle Run-time Loading)

在把装入模块装入内存后，并不立即把装入模块中的相对地址转换为绝对地址，而是把这种地址转换推迟到程序真正要执行时才进行。

4.1.2 程序的链接

链接方式：

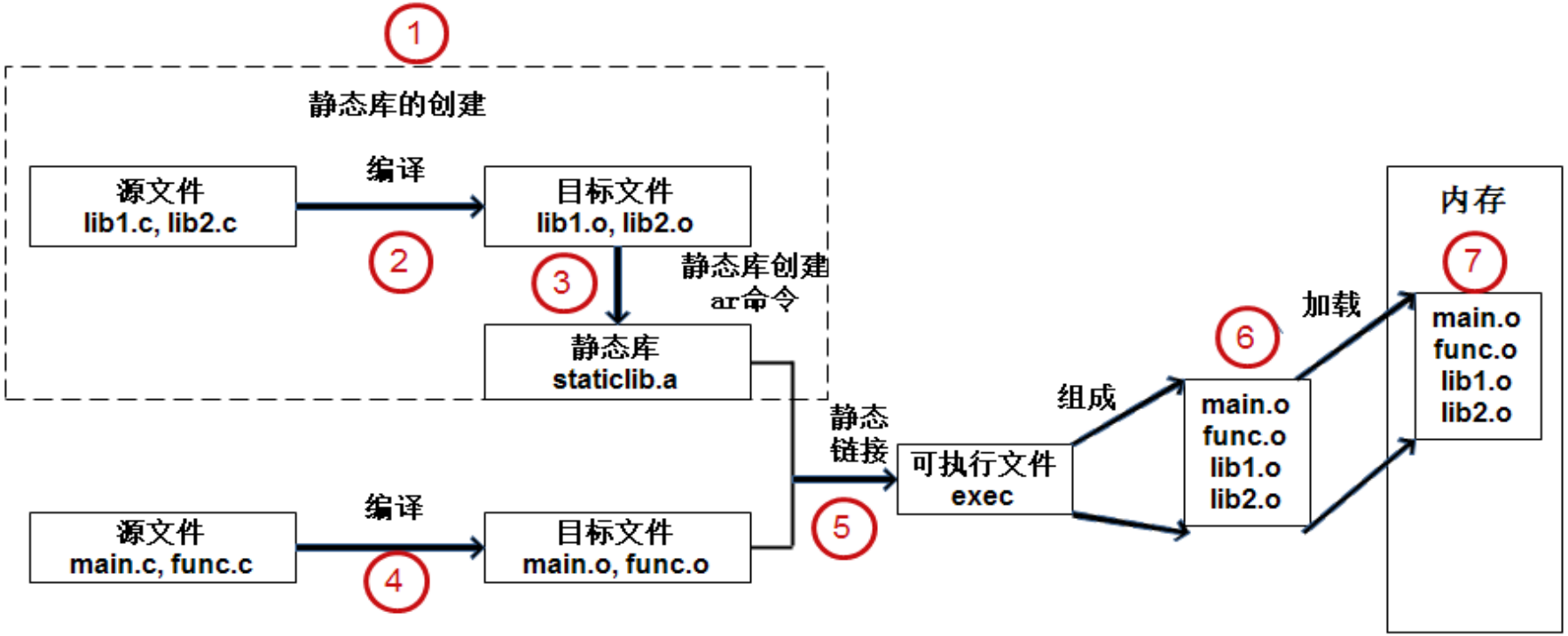
- 静态链接：在程序执行以前由链接程序完成。
- 动态链接：在程序执行过程中根据需要而进行。

每一个进程都有一个虚拟空间（是一维还是多维，由存储管理方式决定）。

每个指令或数据单元都在这个虚拟空间中拥有确定的地址，把这个地址称为虚拟地址（virtual address）。

显然，进程在该空间的地址排列是非连续的，其内存中的物理地址是由虚拟地址经过地址变换机构变换后得到。

静态链接示意图


1
静态库的创建
源文件
lib1.c, lib2.c
编译
2
目标文件
lib1.o, lib2.o
3
静态库创建
ar命令
静态库
staticlib.a
源文件
main.c, func.c
编译
4
目标文件
main.o, func.o
静态
链接
5
可执行文件
exec
组成
6
main.o
func.o
lib1.o
lib2.o
加载
内存
7
main.o
func.o
lib1.o
lib2.o

装入时动态链接

装入时动态链接方式有以下优点：

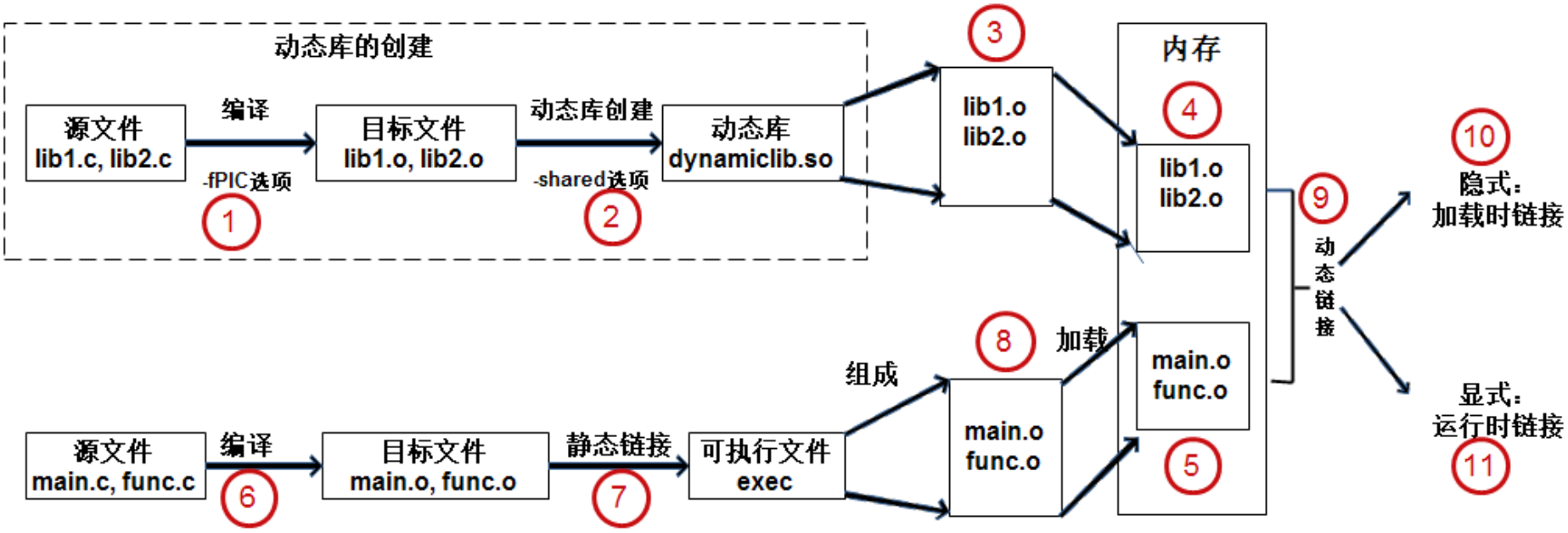
(1) 便于修改和更新。

(2) 便于实现对目标模块的共享。

运行时动态链接

近几年流行起来的运行时动态链接方式，是对上述在装入时链接方式的一种改进。这种链接方式是将对某些模块的链接**推迟到执行时**才执行，亦即，在执行过程中，当发现一个被调用模块尚未装入内存时，立即由OS去找到该模块并将之装入内存， 把它链接到调用者模块上。凡在执行过程中未被用到的目标模块，都不会被调入内存和被链接到装入模块上，这样不仅可加快程序的装入过程，而且可节省大量的内存空间。

动态链接示意图


动态库的创建
源文件
lib1.c, lib2.c
编译
-fPIC选项
1
目标文件
lib1.o, lib2.o
动态库创建
-shared选项
2
动态库
dynamiclib.so
3
lib1.o
lib2.o
内存
4
lib1.o
lib2.o
9
动态链接
10
隐式：
加载时链接
显式：
运行时链接
11
8
加载
组成
main.o
func.o
main.o
func.o
5
源文件
main.c, func.c
编译
6
目标文件
main.o, func.o
静态链接
7
可执行文件
exec

第四章 存储器管理

4.1 程序的装入和链接

4.2 连续分配方式

4.3 基本分页存储管理方式

4.4 基本分段存储管理方式

4.5 虚拟存储器的基本概念

4.6 请求分页存储管理方式

4.7 页面置换算法

4.8 请求分段存储管理方式

4.2 连续分配方式

4.2.1 单一连续分配

这是最简单的一种存储管理方式，但只能用于单用户、单任务的操作系统中。采用这种存储管理方式时，可把内存分为系统区和用户区两部分，系统区仅提供给OS使用，通常是放在内存的低址部分；用户区是指除系统区以外的全部内存空间， 提供给用户使用。

4.2.2 固定分区分配

1. 划分分区的方法

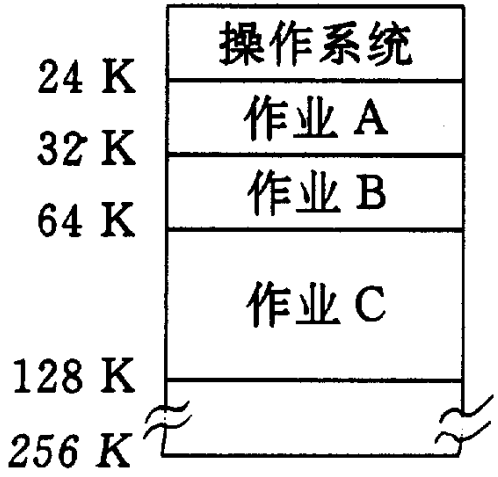
(1) 分区大小相等， 即使所有的内存分区大小相等。

(2) 分区大小不等。

2. 内存分配

分区号	大小(K)	起址(K)	状态
1	12	20	已分配
2	32	32	已分配
3	64	64	已分配
4	128	128	已分配

(a) 分区说明表



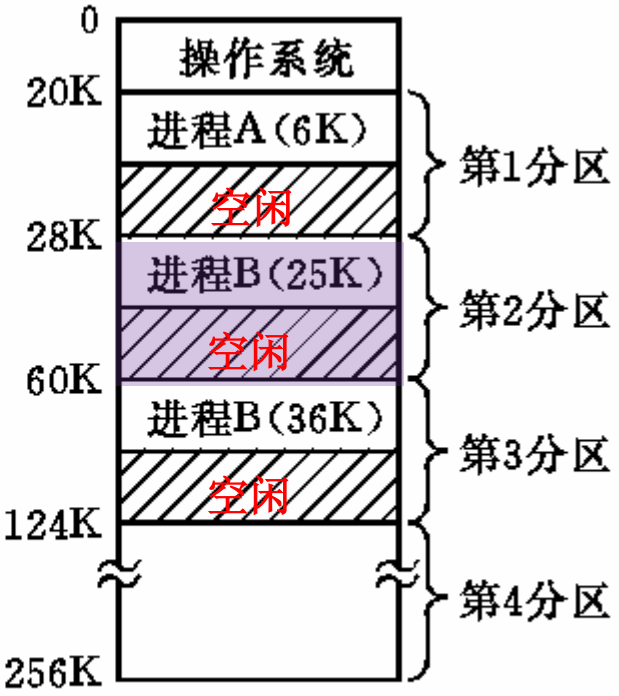
(b) 存储空间分配情况

图 4-4 固定分区使用表

固定分区法

区号	分区长度	起始地址	状态
1	8K	20K	已分配
2	32K	28K	已分配
3	64K	60K	已分配
4	132K	124K	未分配

(a) 分区说明表



(b) 内存状态

注意内部碎片

4.2.3 动态分区分配

1. 分区分配中的数据结构

(1) 空闲分区表。

(2) 空闲分区链。

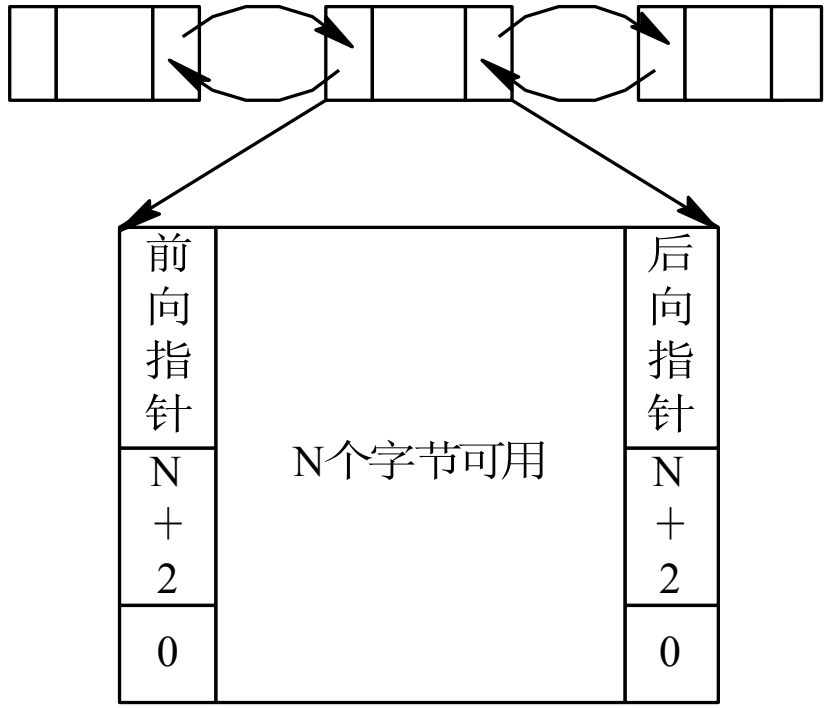


图 4-5 空闲链结构

2. 分区分配算法

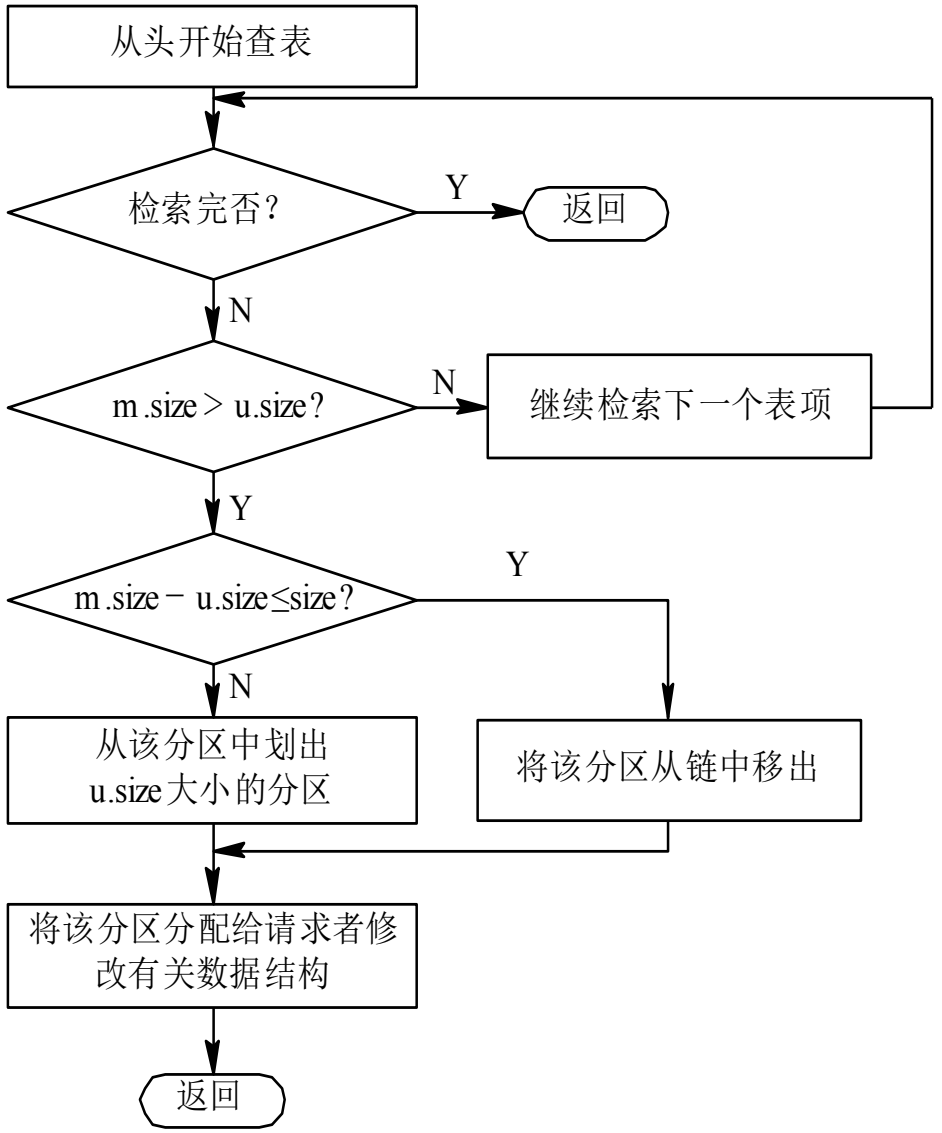
(1) 首次适应算法FF。

(2) 循环首次适应算法，该算法是由首次适应算法演变而成的。

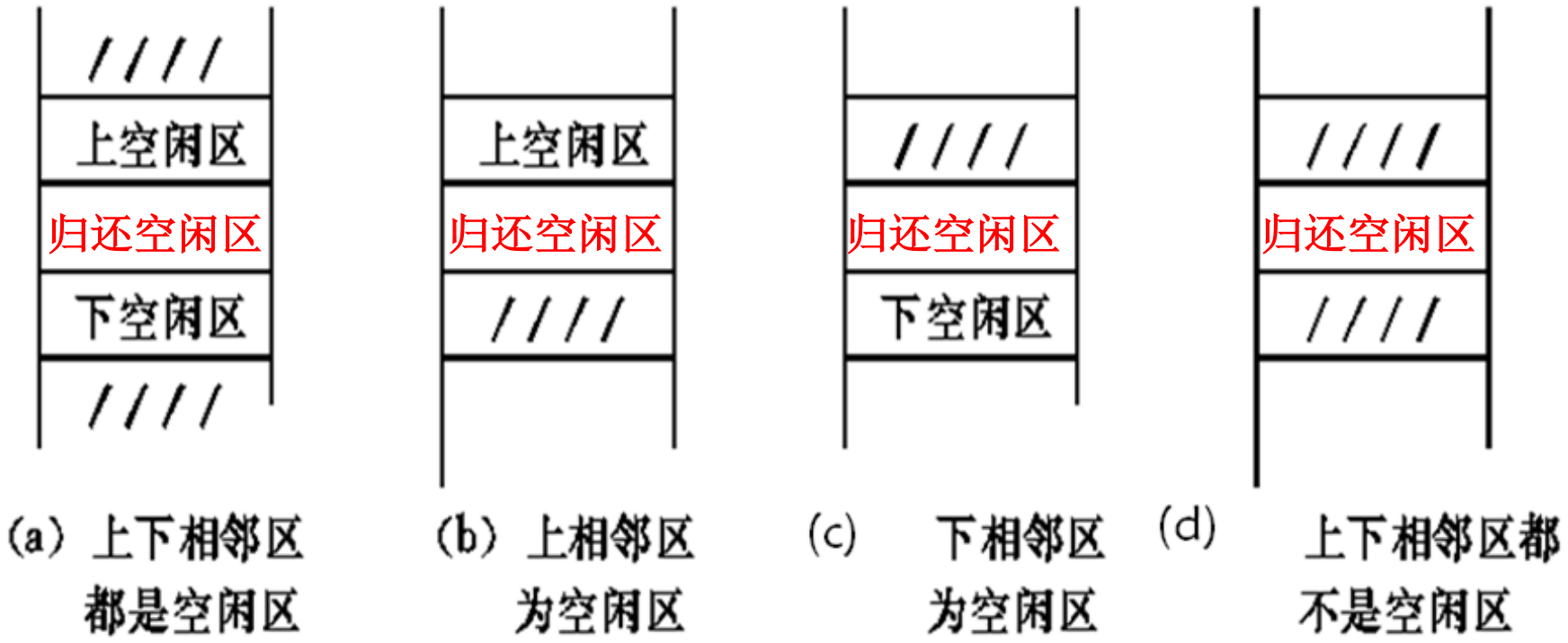
(3) 最佳适应算法。

3. 分区分配操作

1) 分配内存



2) 回收内存



(a) 上下相邻区都是空闲区

(b) 上相邻区为空闲区

(c) 下相邻区为空闲区

(d) 上下相邻区都不是空闲区

大小：3区之和

起始地址：上空闲区

大小：2区之和

起始地址：上空闲区

大小：2区之和

起始地址：空闲区

大小：不变

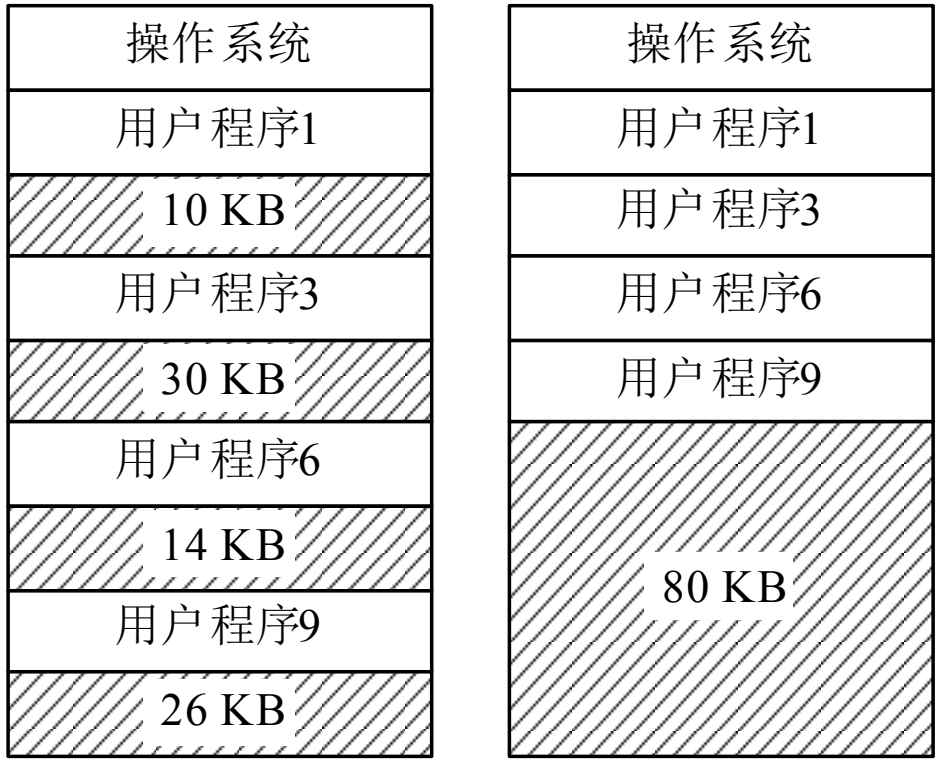
起始地址：空闲区

图 4-7 内存回收时的情况

4.2.4 可重定位分区分配

1. 动态重定位的引入

图 4-8 紧凑的示意图（右）



(*a*) 紧凑前　　(*b*) 紧凑后

2. 动态重定位的实现

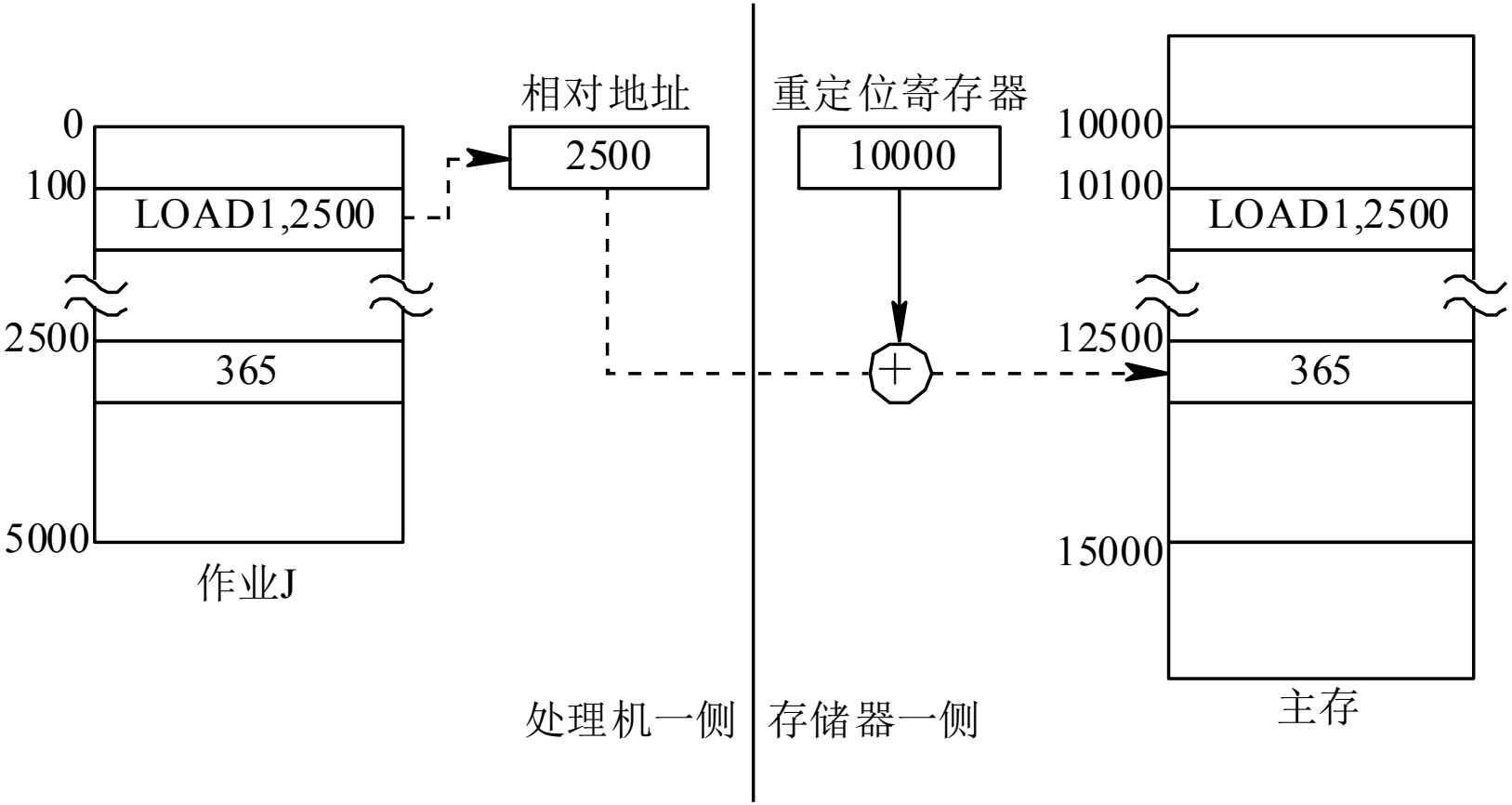


图 4-9 动态重定位示意图

3. 动态重定位分区分配算法

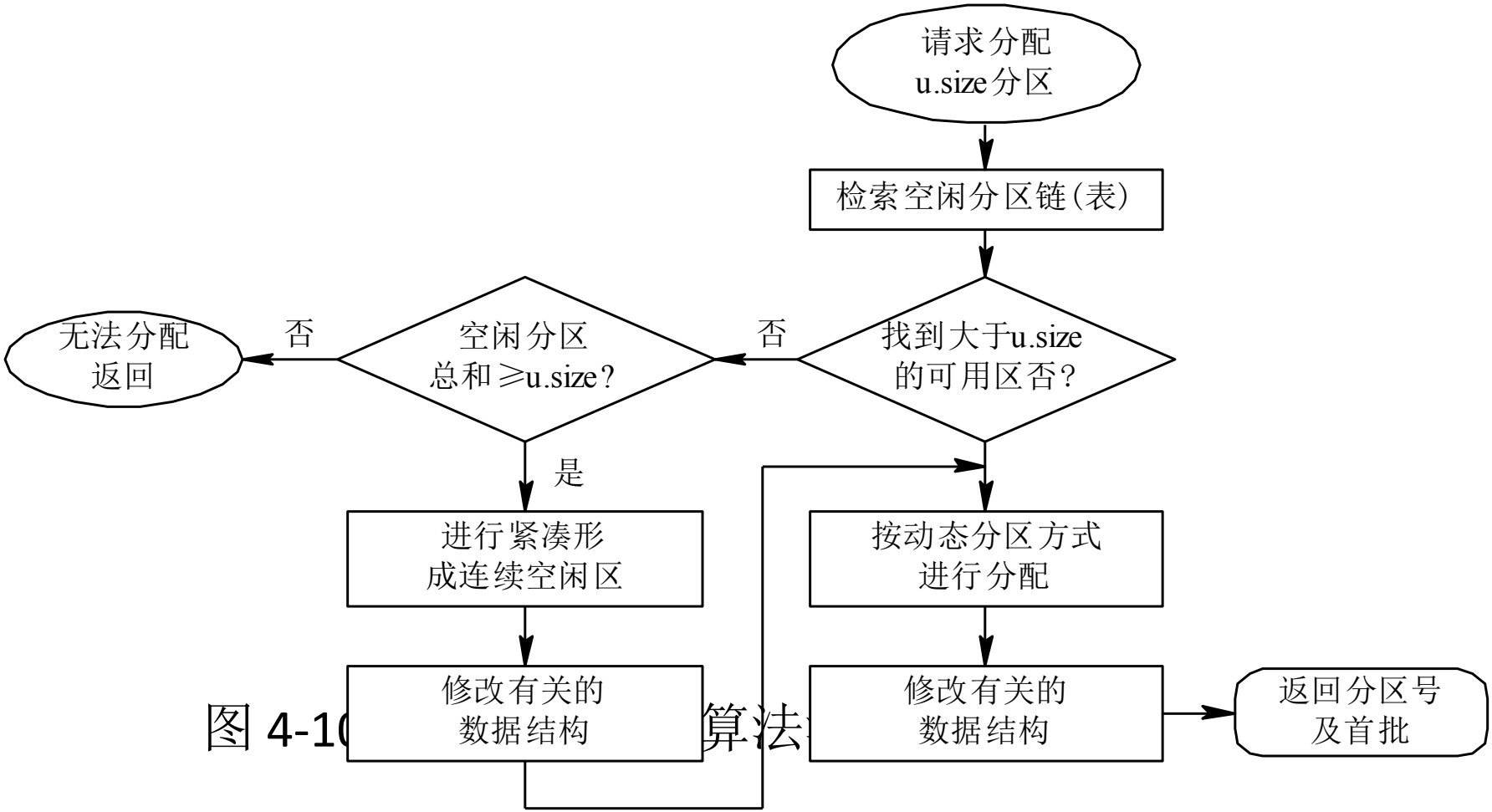


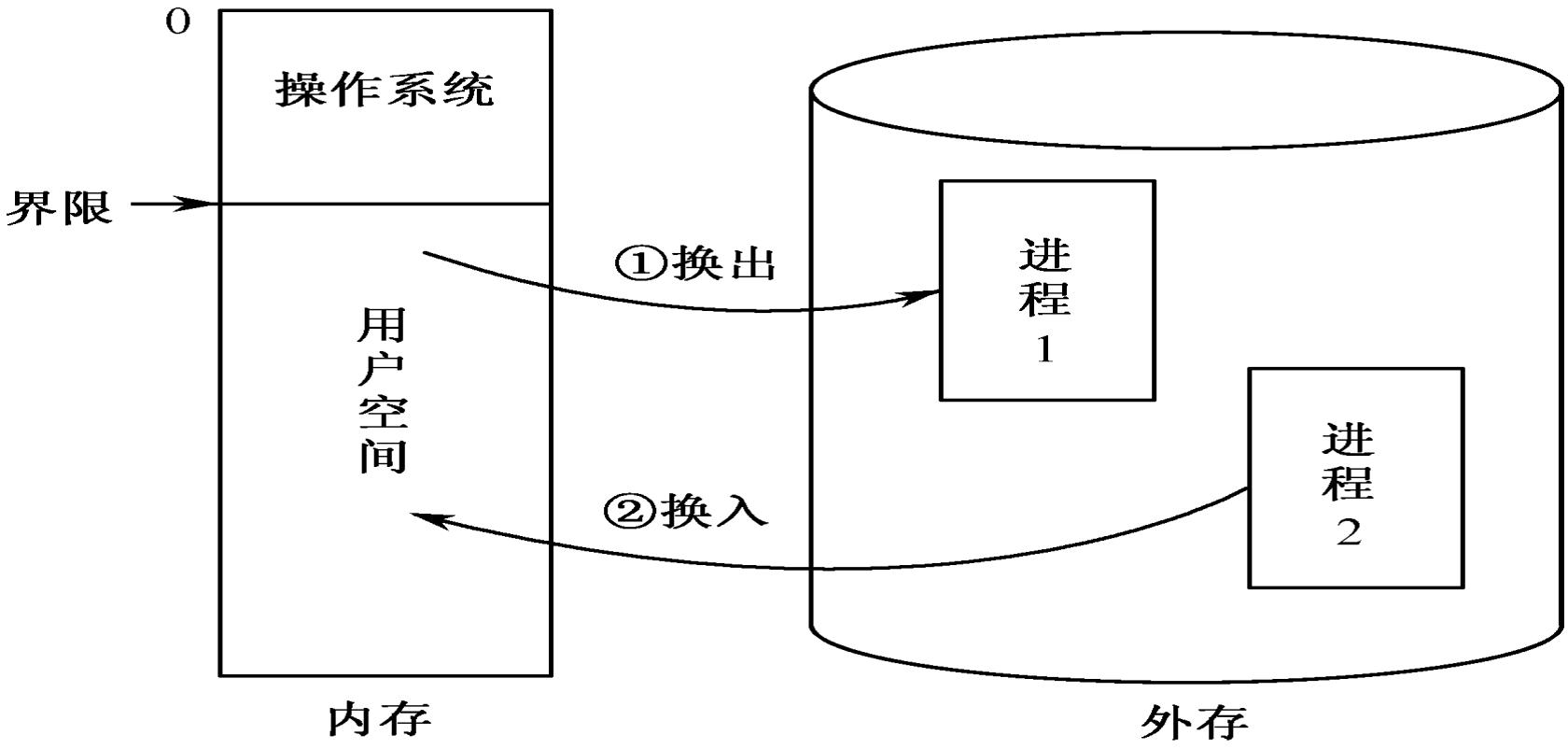
图 4-10 动态重定位分区分配算法

4.2.5 对换(Swapping)

1. 对换的引入

所谓“对换”， 是指把内存中暂时不能运行的进程或者暂时不用的程序和数据，调出到外存上，以便腾出足够的内存空间，再把已具备运行条件的进程或进程所需要的程序和数据，调入内存。对换是提高内存利用率的有效措施。

交换技术示意图



2. 对换空间的管理

目的：

为了能对对换区中的空闲盘块进行管理，在系统中应配置相应的数据结构，以记录外存的使用情况。

形式：

- 内存：在动态分区分配**空闲分区表**或**空闲分区链**。
- 外存：对换区的首址及其大小，即**盘块号**和**盘块数**。

3. 进程的换出与换入

交换进程由换出和换入两个过程组成。

- 换出(swap out)过程：

  把内存中的数据和程序换到外存交换区；

- 换入(swap in)过程：

  把外存交换区中的数据和程序换到内存分区中。

换出和换入过程都要完成与外存设备管理进程通信的任务。

第四章 存储器管理

4.3 基本分页存储管理方式

4.3.1 页面与页表

1. 页面

1) 页面和物理块

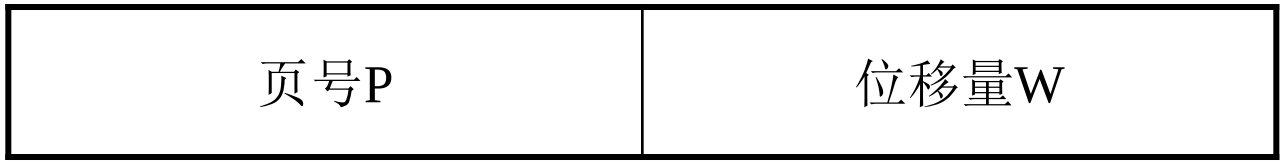
分页存储管理，是将一个进程的逻辑地址空间分成若干个大小相等的片，称为页面或页，并为各页加以编号，从0开始，如第0页、第1页等。相应地，也把内存空间分成与页面相同大小的若干个存储块，称为(物理)块或页框(frame)，也同样为它们加以编号，如0#块、1#块等等。在为进程分配内存时，以块为单位将进程中的若干个页分别装入到多个可以不相邻接的物理块中。由于进程的最后一页经常装不满一块而形成了不可利用的碎片，称之为“页内碎片”。

2) 页面大小

在分页系统中的页面其大小应适中。页面若太小，一方面虽然可使内存碎片减小，从而减少了内存碎片的总空间， 有利于提高内存利用率，但另一方面也会使每个进程占用较多的页面，从而导致进程的页表过长，占用大量内存； 此外，还会降低页面换进换出的效率。然而，如果选择的页面较大，虽然可以减少页表的长度，提高页面换进换出的速度，但却又会使页内碎片增大。因此，页面的大小应选择得适中，且页面大小应是2的幂，通常为512 B~8 KB。

2. 地址结构

分页地址中的地址结构如下：



对某特定机器，其地址结构是一定的。若给定一个逻辑地址空间中的地址为A，页面的大小为L，则页号P和页内地址d可按下式求得：

$$P = INT\left[\frac{A}{L}\right]$$

$$d = [A]MODL$$

3. 页表

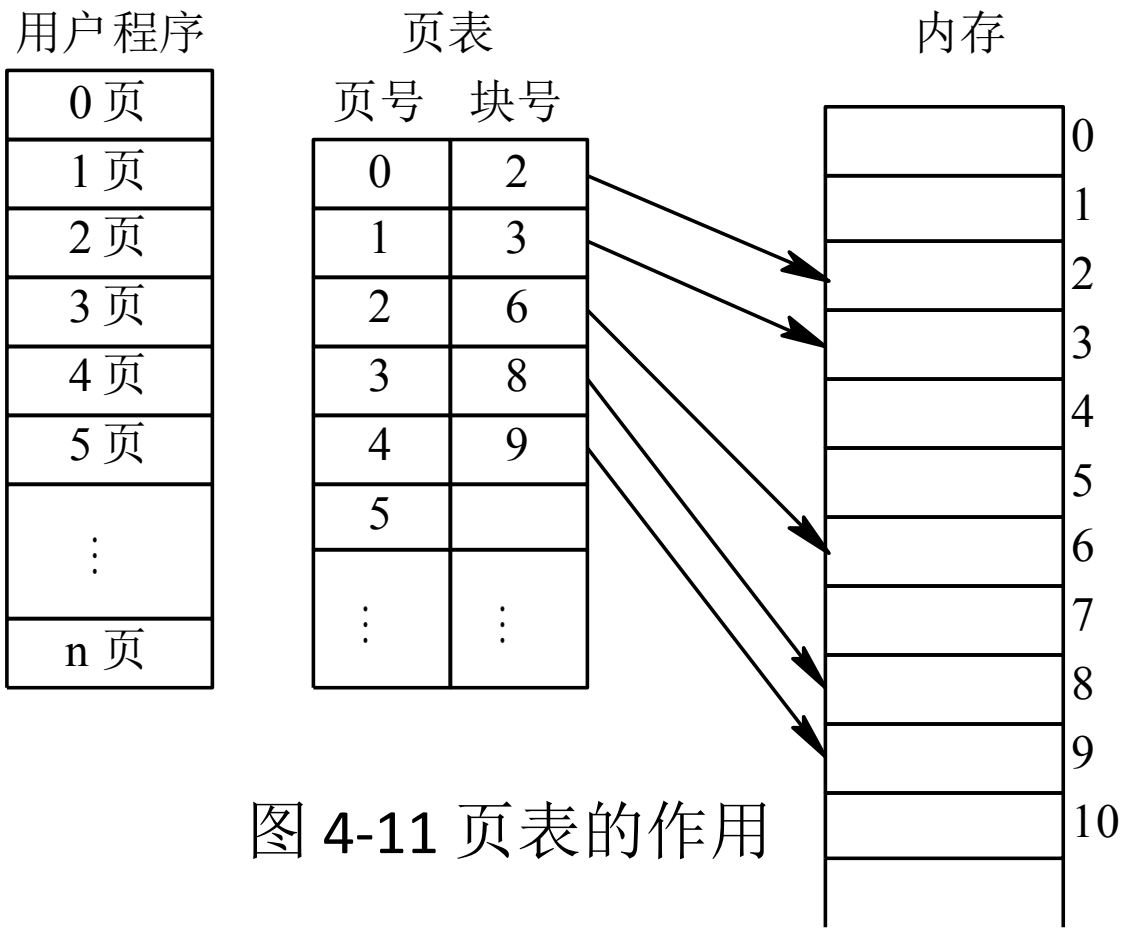


图 4-11 页表的作用

4.3.2 地址变换机构

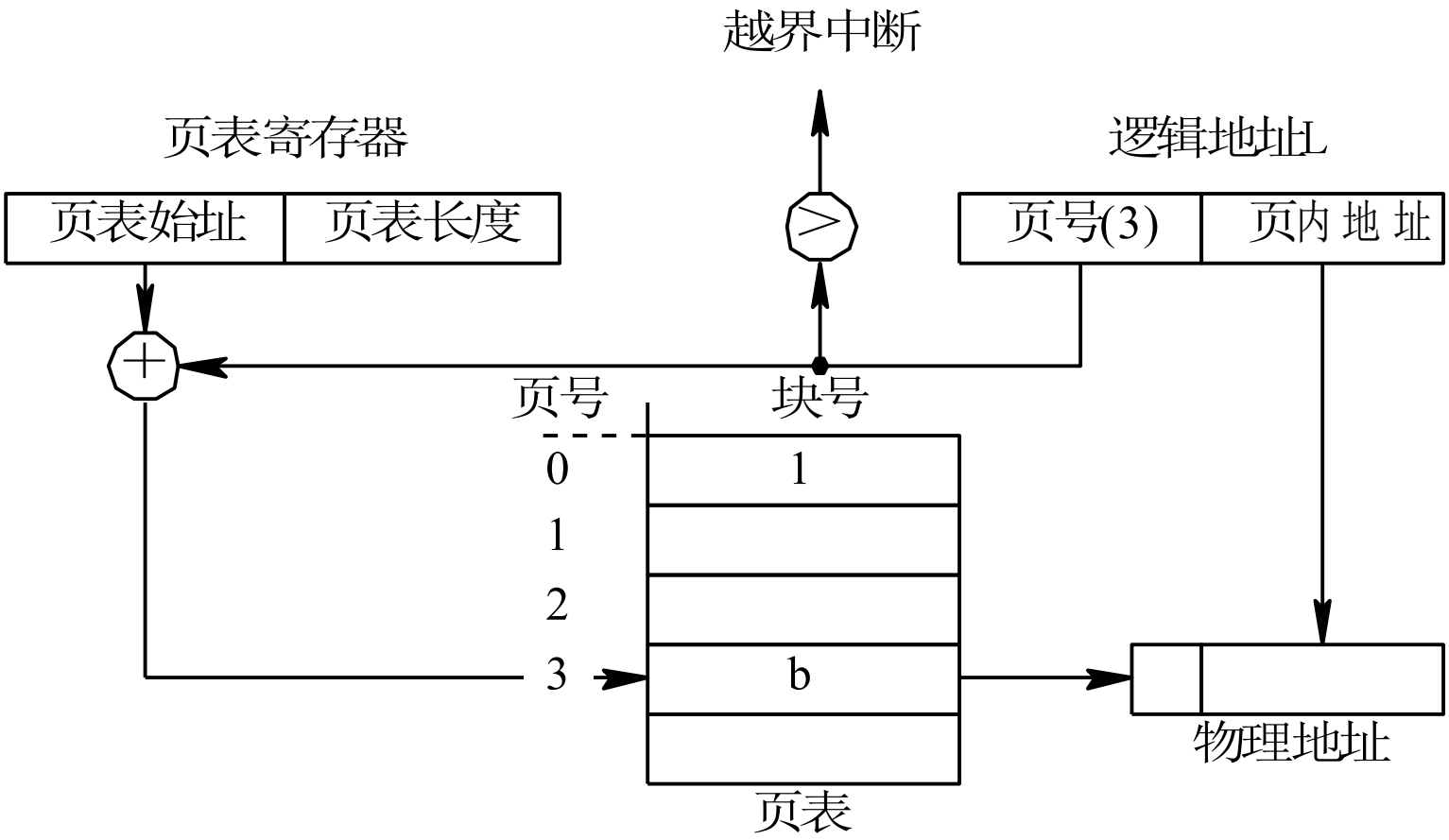


图 4-12 分页系统的地址变换机构

例：若在一分页存储管理系统中， 某作业的页表如下图所示。已知页面大小为1024字节。试将逻辑地址1011，2148，3000，4000，5012转化为相应的物理地址。

页号	页面号(块号)
0	2
1	3
2	1
3	6

步骤：

1.将逻辑地址转换为页号和页内偏移地址；

2. 根据页表找到块号，然后根据公式，求得物理地址。

解:设页号为P,页内偏移为W,逻辑地址为A,页面大小为L,则:

P = int(A / L) 取整（逻辑地址/页面大小）=页号

W =A mod L 取余（逻辑地址/页面大小）=页内偏移地址

页号	页面号(块号)
0	2
1	3
2	1
3	6

对于逻辑地址1011

P=int(1011/l024)=0

W=1011 mod 1024=1011

查页表第0页在第2块,物理地址为2*1024+1011=3059。

对于逻辑地址2148

P=int(2l48/1024)=2

w=2148 mod 1024 =100

查页表第2页在第l块,物理地址为1*1024+100=1124。

页号	页面号(块号)
0	2
1	3
2	1
3	6

对于逻辑地址3000

P=int(3000/1024)=2

w=3000 mod 1024=952

查页表第2页在第l块,物理地址为1*1024+952= 1976。

对于逻辑地址4000

P=int(4000/1024)=3

w=4000 mod 1024=928

查页表第3页在第6块,物理地址为6*1024+928=7072。

对于逻辑地址5012

P=int(5012/A024)=4

w=5012 mod 1024=916

因页号超过页表长度,该逻辑地址非法，实现了保护.

例：在一分页存储管理系统中，逻辑地址长度为16位，页面大小为4096(2^{12})字节，可见页号只有有4位。现有逻辑地址为2F6A，且第0、l、2页依次存放在物理块5、10、11中，问相应的物理地址为多少?

解：由题可知，页式系统的逻辑地址结构

逻辑地址2F6A的二进制表示如下：

P(页号)　　W(页内地址)

0010111101101010

由此可知逻辑地址2F6A的页号为2，对应第11号物理块，用十六进制表示块号为B，所以物理地址为BF6A。

2. 具有快表的地址变换机构

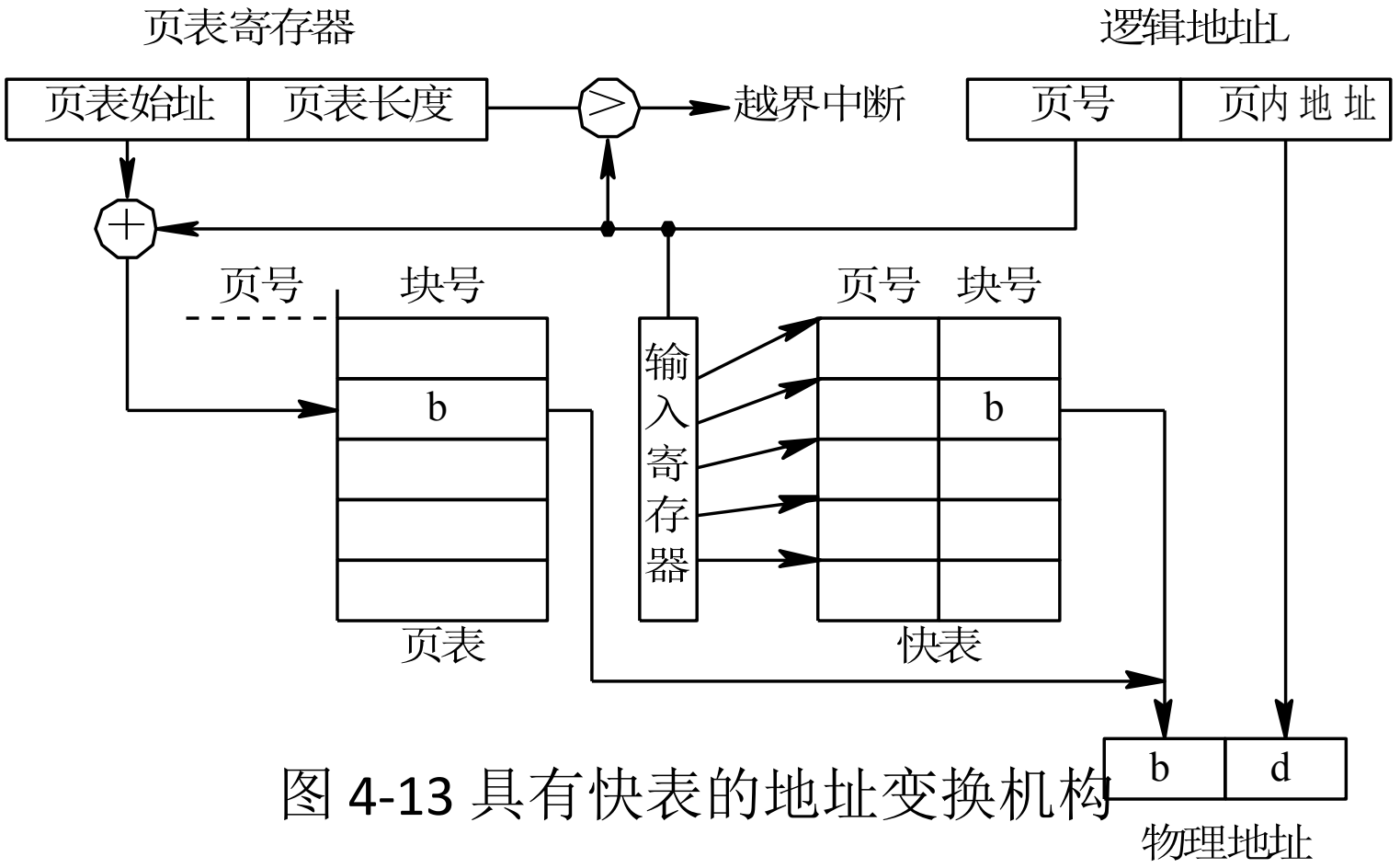


图 4-13 具有快表的地址变换机构

地址变换过程全部由硬件地址变换机构自动完成。

由于页表是驻留在内存的某个固定区域中。取一个数据或指令至少要访问内存两次以上。

第一次访问页表以确定所取数据或指令的物理地址。

第二次是根据地址取数据或指令。使得执行指令的速度慢了一倍。

提高查找速度的办法：

1.为了避免多次进入内存可把页表放在寄存器中，但寄存器价格较贵。

2.在地址变换机构中加入一个高速联想存储器，构成一张快表（Cache）。

在快表中，存入那些当前执行进程的的页表，每次查找都先从快表查，这样可以提高查找速度。

例题：

假定访问主存时间为100毫微秒，访问联想存储器时间为20毫微秒，联想存储器为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：

$$（100＋20）\times 90\%＋(100+100+20)\times(1-90\%)＝130$$

快表　　内存　　毫微秒

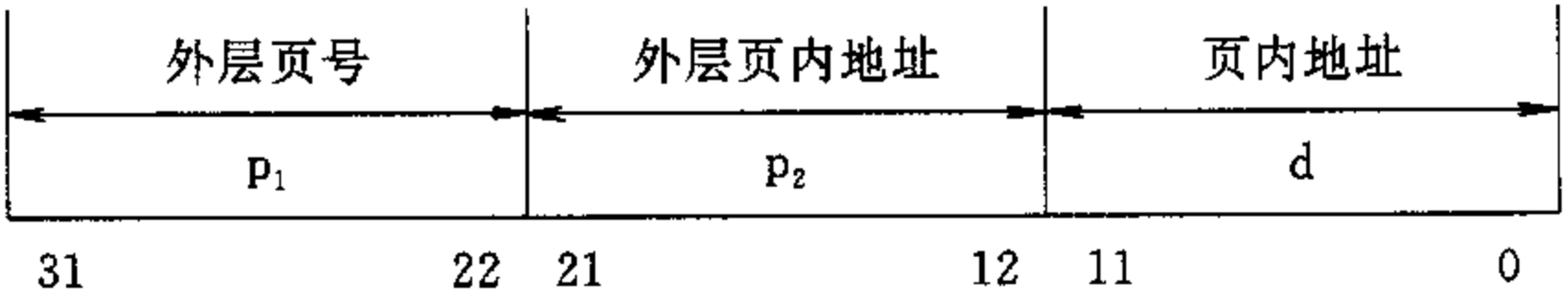
二次访问主存的时间：100毫微秒×2＝200毫微秒。

4.3.3 两级和多级页表

现代的大多数计算机系统，都支持非常大的逻辑地址空间(2^{32}~2^{64})。在这样的环境下，页表就变得非常大，要占用相当大的内存空间。例如，对于一个具有32位逻辑地址空间的分页系统，规定页面大小为4 KB即2^{12} B，则在每个进程页表中的页表项可达1兆个之多。又因为每个页表项占用一个字节， 故每个进程仅仅其页表就要占用4 KB的内存空间，而且还要求是连续的。 可以采用这样两个方法来解决这一问题：① 采用离散分配方式来解决难以找到一块连续的大内存空间的问题：② 只将当前需要的部分页表项调入内存， 其余的页表项仍驻留在磁盘上，需要时再调入。

1. 两级页表(Two-Level Page Table)

逻辑地址结构可描述如下：



两级页表结构

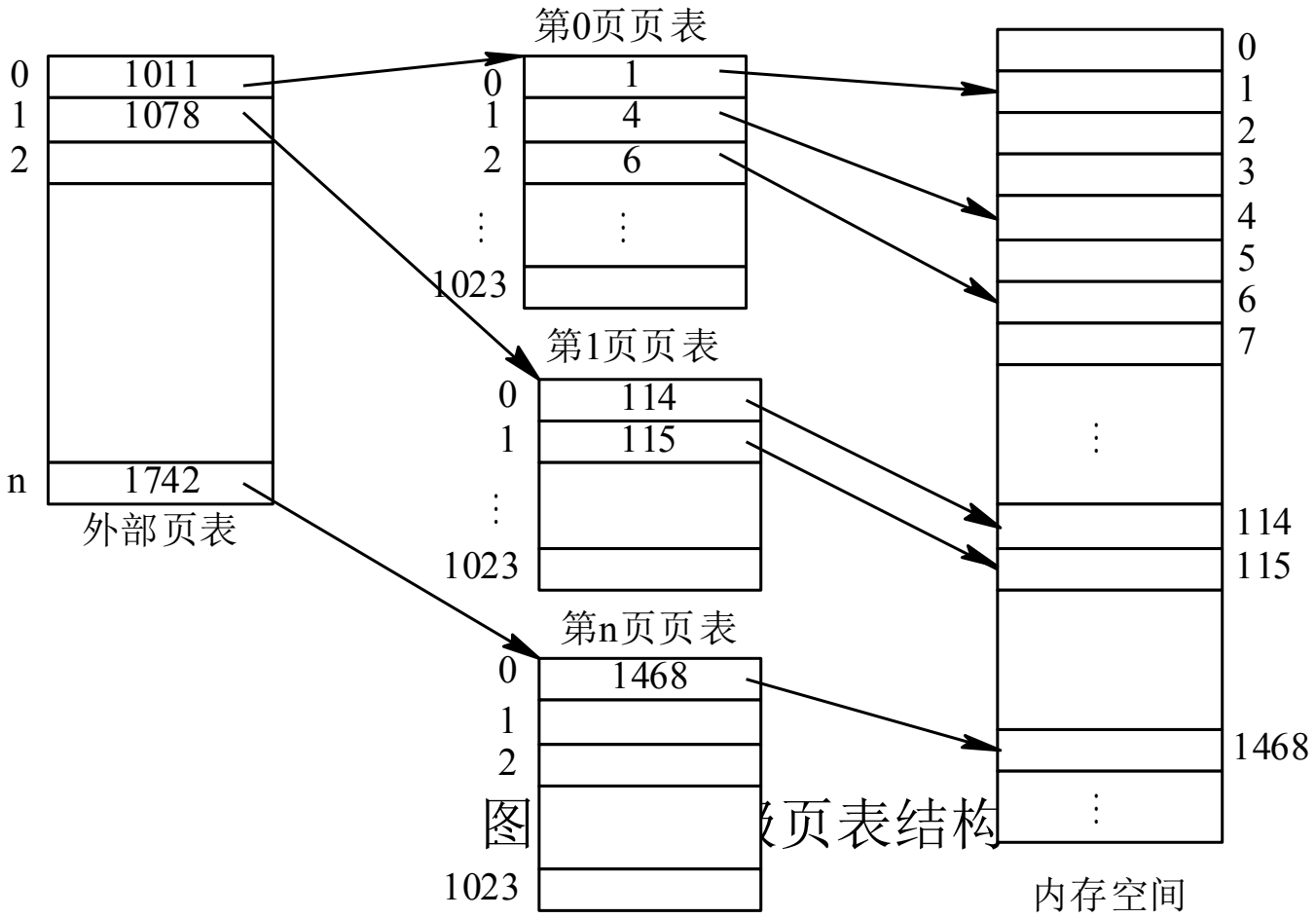


图 级页表结构

具有两级页表的地址变换机构

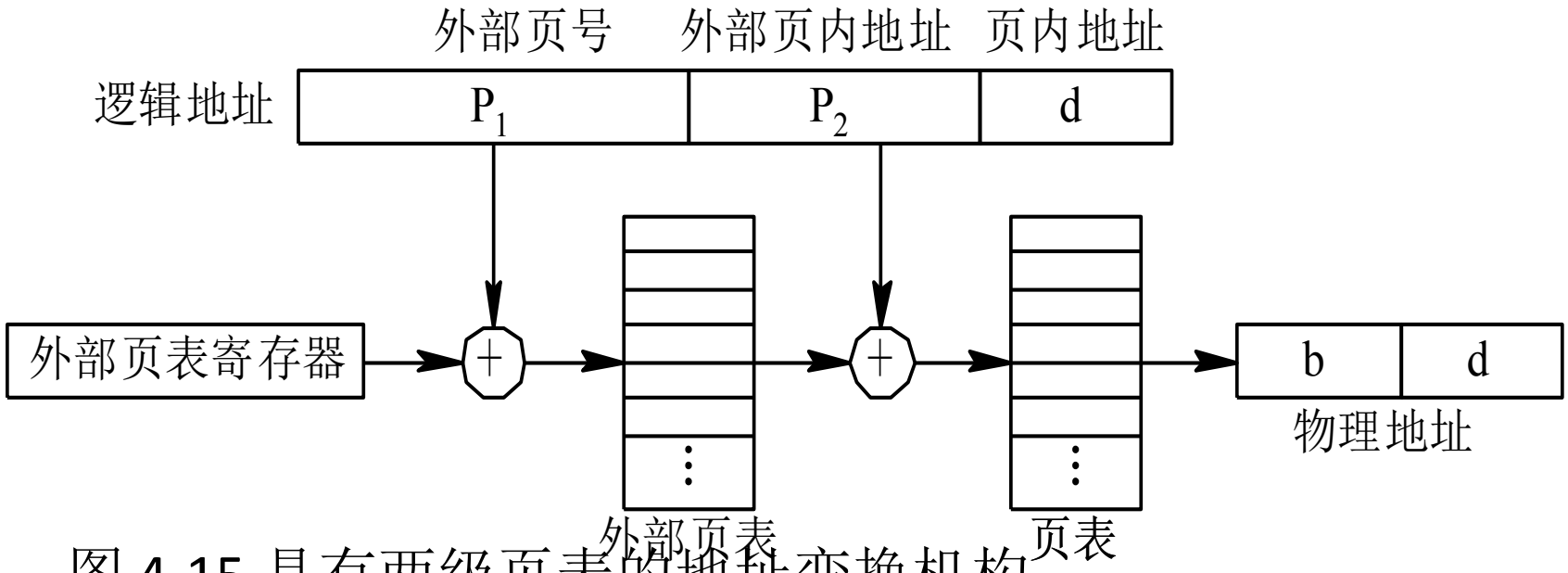


图 4-15 具有两级页表的地址变换机构

2. 多级页表

- 对于32位的机器，采用两级页表结构是合适的；但对于64位的机器，如果页面大小仍采用4 KB即2^{12} B，那么还剩下52位， 假定仍按物理块的大小(2^{12}位)来划分页表，则将余下的42位用于外层页号。此时在外层页表中可能有4096 G个页表项， 要占用16384 GB的连续内存空间。 必须采用多级页表，将外层页表再进行分页，也是将各分页离散地装入到不相邻接的物理块中，再利用第2级的外层页表来映射它们之间的关系。
- 对于64位的计算机，如果要求它能支持2^{64}(=1844744 TB)规模的物理存储空间，则即使是采用三级页表结构也是难以办到的；而在当前的实际应用中也无此必要。

第四章 存储器管理

4.4 基本分段存储管理方式

4.4.1 分段存储管理方式的引入

引入分段存储管理方式， 主要是为了满足用户和程序员的下述一系列需要：囍

1) 方便编程囍

2) 信息共享

3) 信息保护

4) 动态增长囍

5) 动态链接

4.4.2 分段系统的基本原理

1. 分段

分段地址中的地址具有如下结构：

段号	段内地址

2. 段表

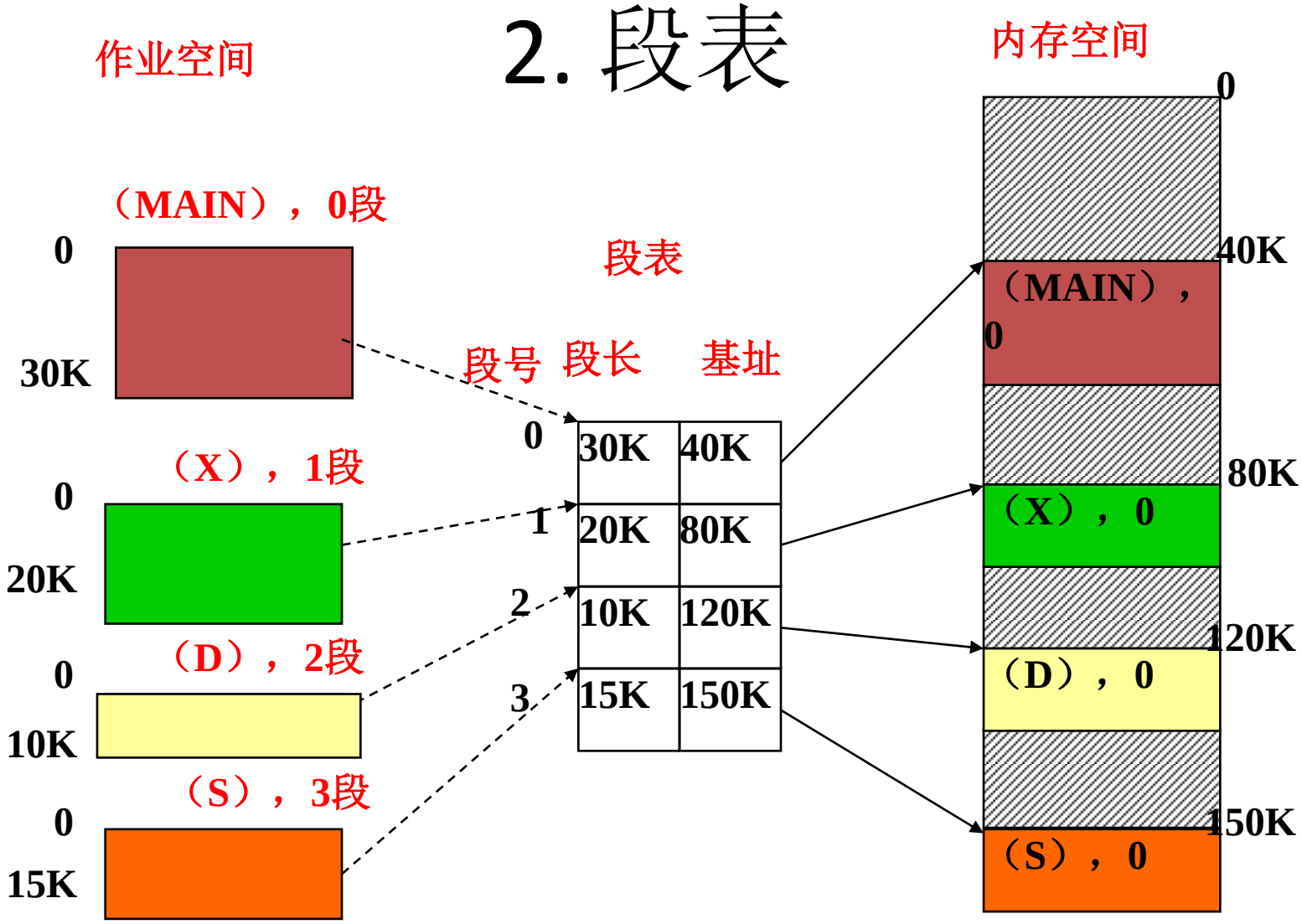


图 4-16 利用段表实现地址映射

3. 地址变换机构

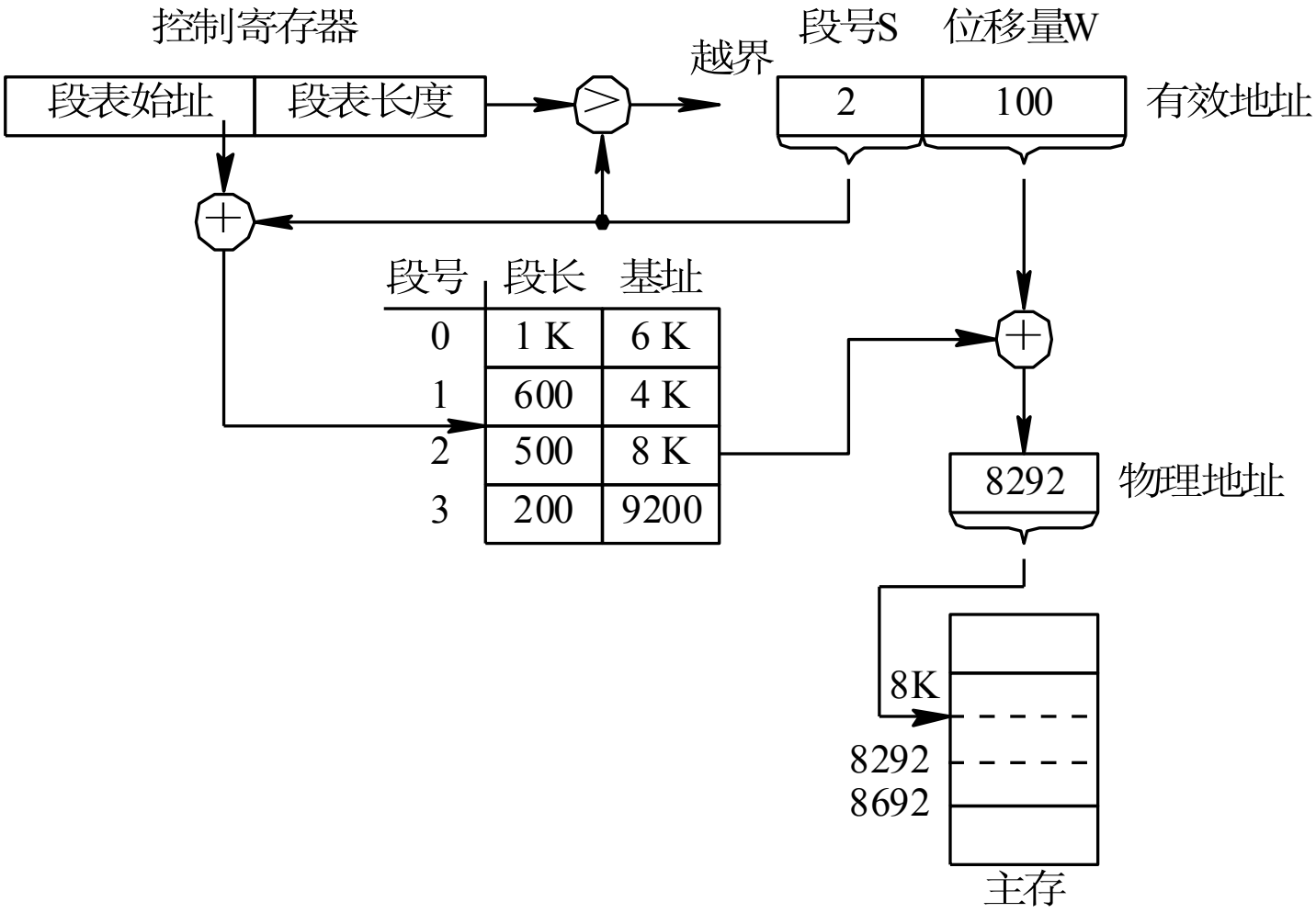


图 4-17 分段系统的地址变换过程

4. 分页和分段的主要区别

(1) 页是信息的物理单位，分页是为实现离散分配方式，以消减内存的外零头， 提高内存的利用率。段则是信息的逻辑单位，它含有一组其意义相对完整的信息。 分段的目的是为了能更好地满足用户的需要。

(2) 页的大小固定且由系统决定，由系统把逻辑地址划分为页号和页内地址两部分，是由机器硬件实现的；而段的长度却不固定， 决定于用户所编写的程序，通常由编译程序在对源程序进行编译时划分。

(3) 分页的作业地址空间是一维的，即单一的线性地址空间，程序员只需利用一个记忆符，即可表示一个地址； 而分段的作业地址空间则是二维的，程序员在标识一个地址时，既需给出段名， 又需给出段内地址。

4.4.3 信息共享

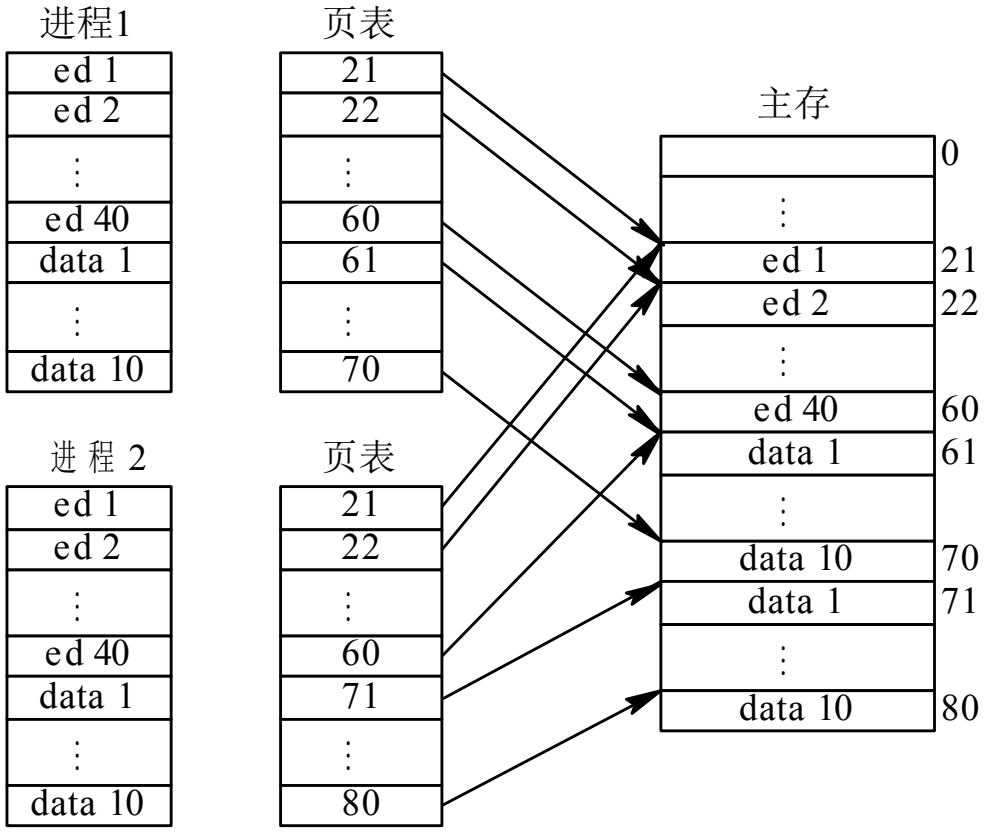


图 4-18 分页系统中共享editor的示意图

4.4.4 段页式存储管理方式

1. 基本原理

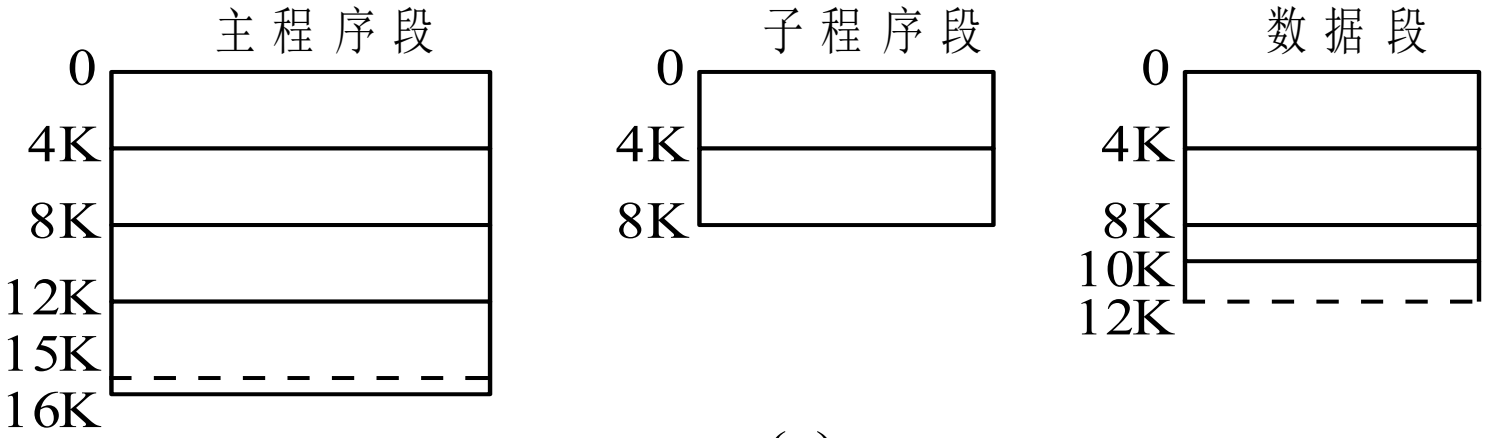


图 4-19 作业地址空间和地址结构

利用段表和页表实现地址映射

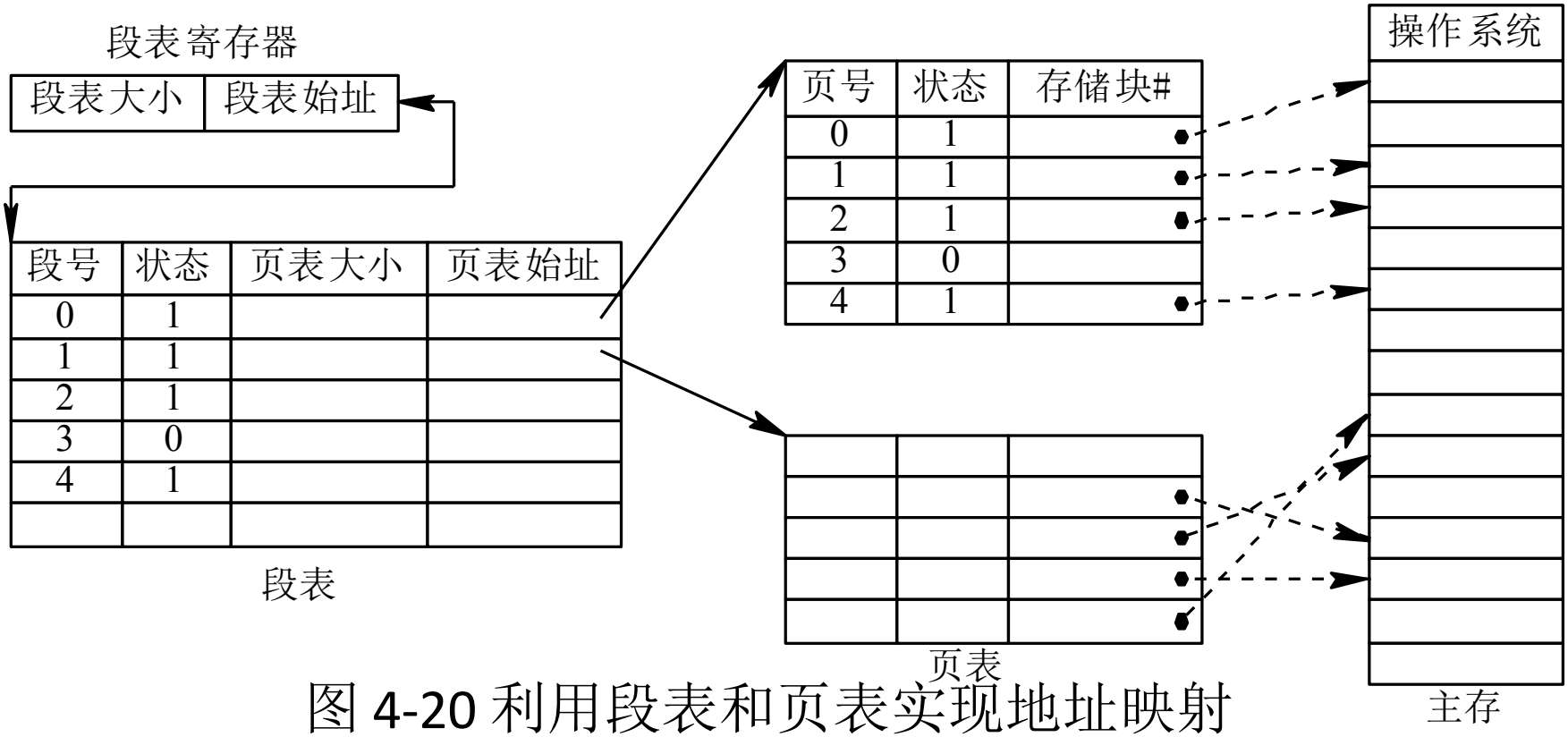


图 4-20 利用段表和页表实现地址映射

2. 地址变换过程

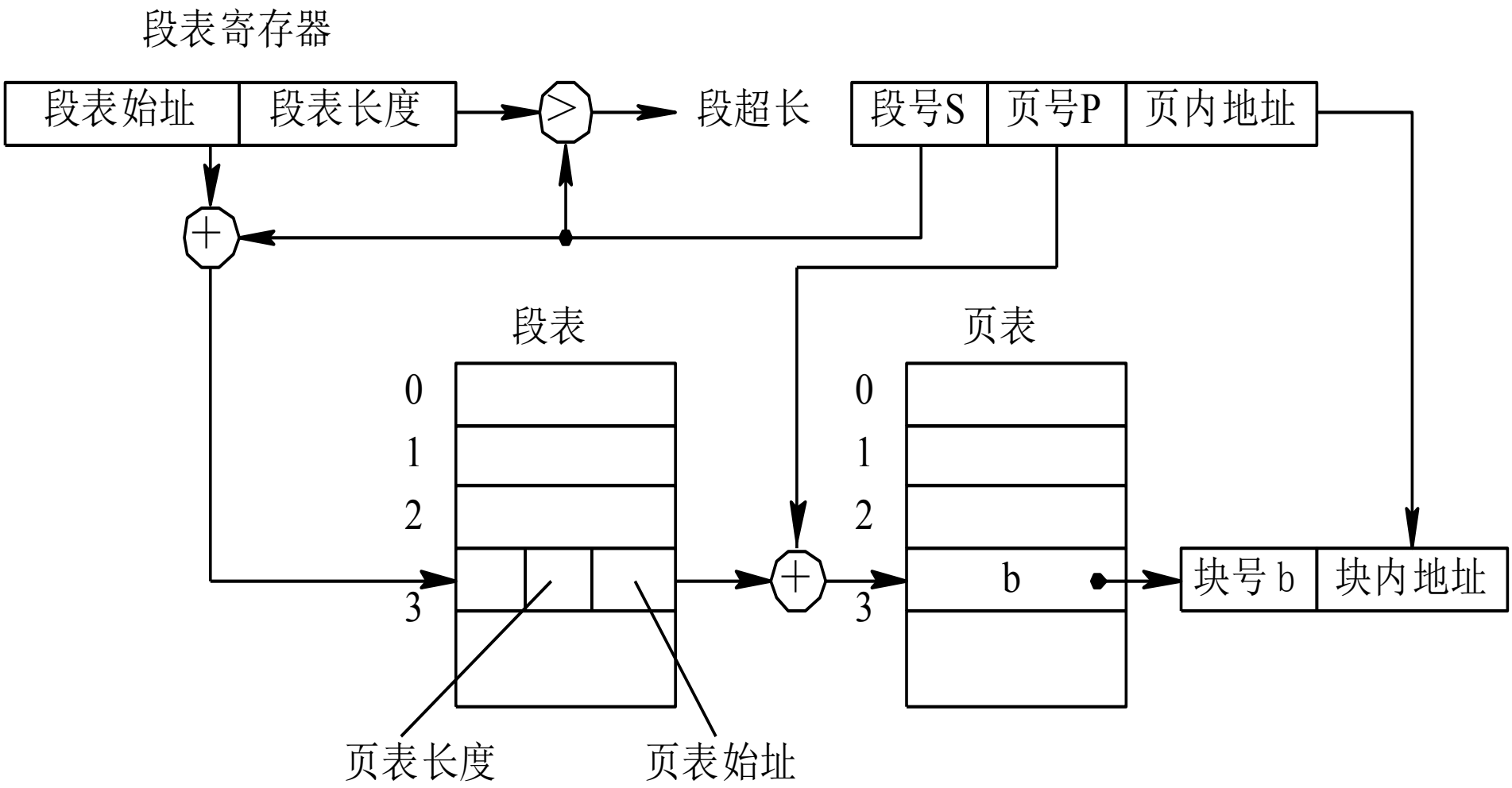


图 4-21 段页式系统中的地址变换机构

动态地址变换过程

段表和页表应该放在内存中一块固定的区域。

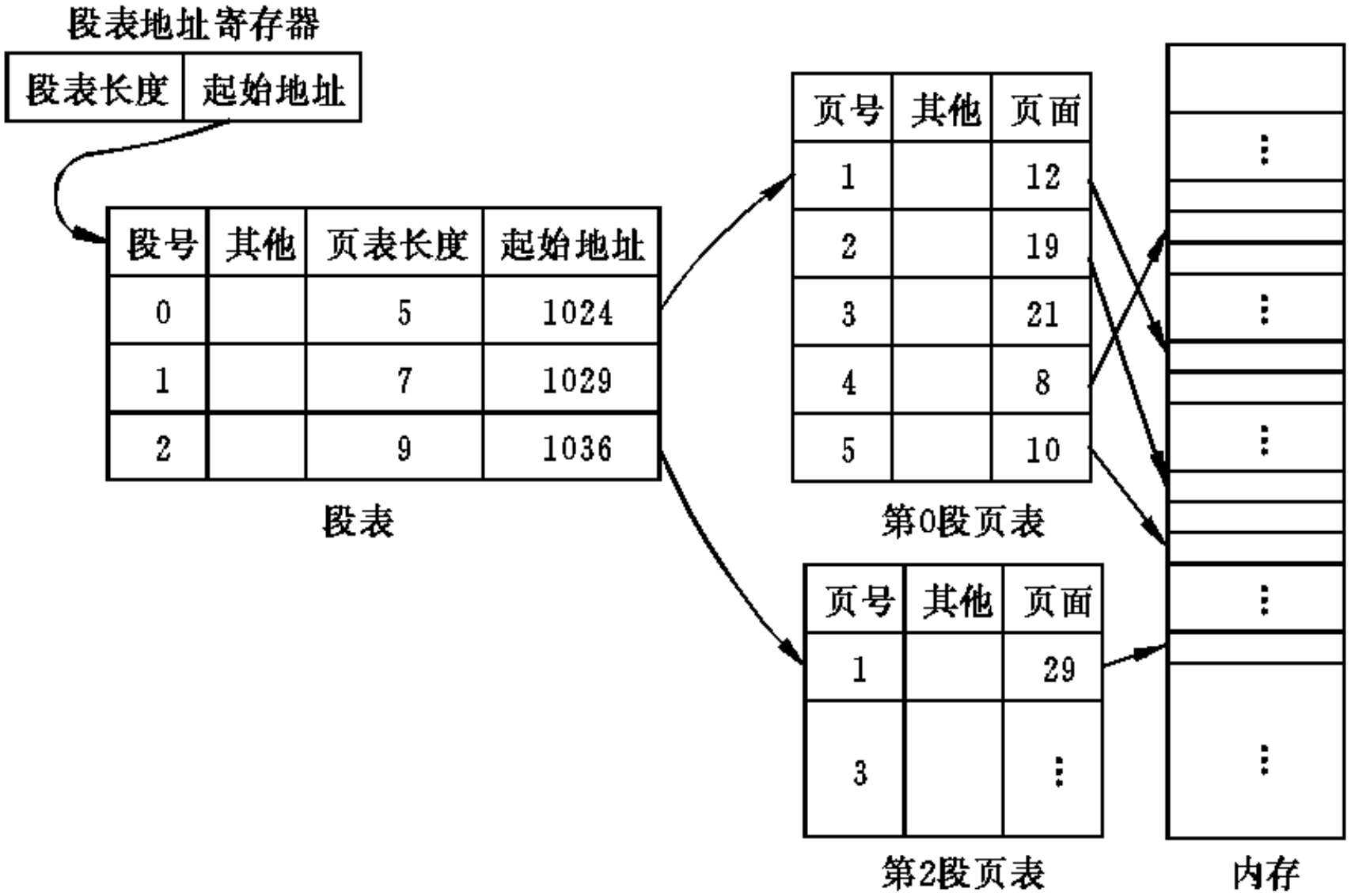
在段页式管理系统中，要对内存中指令或数据进行一次存取的话，至少需要访问三次以上的内存。

- 第一次是由段表地址寄存器得到段表始址去访问段表，由此取出对应段的页表地址。
- 第二次则是访问页表得到所要访问单元的物理地址。
- 第三次才能访问真正需要访问的物理单元。

显然，这将使CPU的执行指令速度大大降低。

为了提高地址转换速度，也可设置快速联想寄存器（快表）。

段页式管理中段表、页表与内存的关系



第四章 存储器管理

4.5 虚拟存储器的基本概念

4.5.1 虚拟存储器的引入

1. 常规存储器管理方式的特征囍

(1)一次性。

(2) 驻留性。

2. 局部性原理

早在1968年， Denning.P就曾指出：

(1) 程序执行时， 除了少部分的转移和过程调用指令外，在大多数情况下仍是顺序执行的。

(2) 过程调用将会使程序的执行轨迹由一部分区域转至另一部分区域， 但经研究看出，过程调用的深度在大多数情况下都不超过5。

(3) 程序中存在许多循环结构， 这些虽然只由少数指令构成， 但是它们将多次执行。囂

(4) 程序中还包括许多对数据结构的处理， 如对数组进行操作， 它们往往都局限于很小的范围内。

2. 局部性原理

- 局限性又表现在下述两个方面：
- (1) 时间局限性。如果程序中的某条指令一旦执行， 则不久以后该指令可能再次执行；如果某数据被访问过， 则不久以后该数据可能再次被访问。产生时间局限性的典型原因，是由于在程序中存在着大量的循环操作。
- (2) 空间局限性。一旦程序访问了某个存储单元，在不久之后，其附近的存储单元也将被访问，即程序在一段时间内所访问的地址，可能集中在一定的范围之内，其典型情况便是程序的顺序执行。

3. 虚拟存储器定义

所谓虚拟存储器，是指具有请求调入功能和置换功能，能从逻辑上对内存容量加以扩充的一种存储器系统。其逻辑容量由内存容量和外存容量之和所决定，其运行速度接近于内存速度，而每位的成本却又接近于外存。可见，虚拟存储技术是一种性能非常优越的存储器管理技术，故被广泛地应用于大、中、小型机器和微型机中。

4.5.2 虚拟存储器的实现方法

1. 分页请求系统

(1) 硬件支持。

① 请求分页的页表机制，它是在纯分页的页表机制上增加若干项而形成的，作为请求分页的数据结构；② 缺页中断机构，即每当用户程序要访问的页面尚未调入内存时 便产生一缺页中断，以请求OS将所缺的页调入内存；③ 地址变换机构， 它同样是在纯分页地址变换机构的基础上发展形成的。

(2) 实现请求分页的软件。

4.5.3 虚拟存储器的特征

1.多次性

2.对换性

3. 虚拟性

第四章 存储器管理

4.6 请求分页存储管理方式

4.6.1 请求分页中的硬件支持

1. 页表机制

页号	物理块号	状态位P	访问字段A	修改位M	外存地址

2. 缺页中断机构

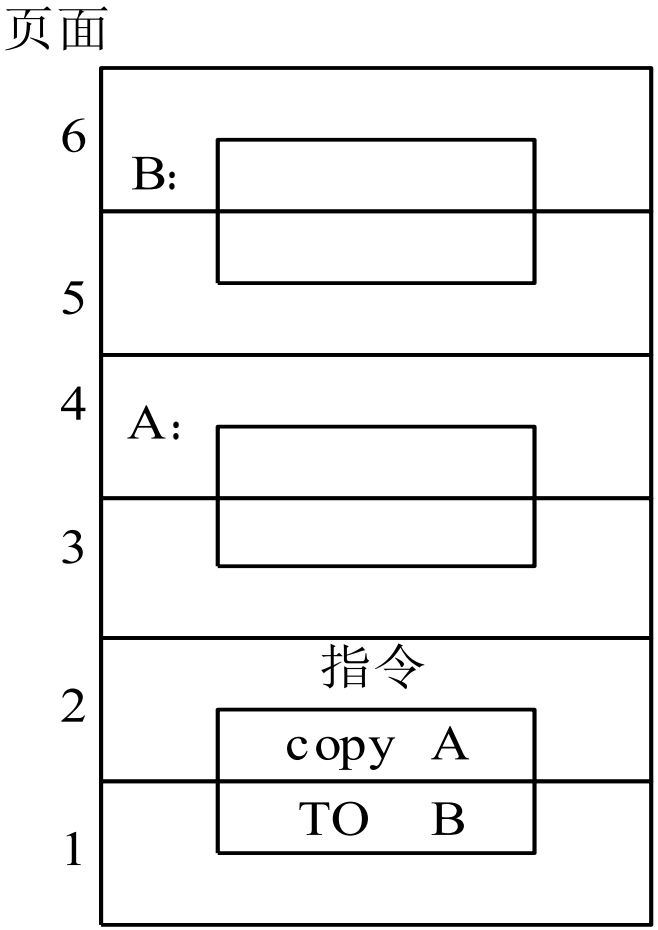


图 4-22 涉及6次缺页中断的指令

3. 地址变换机构

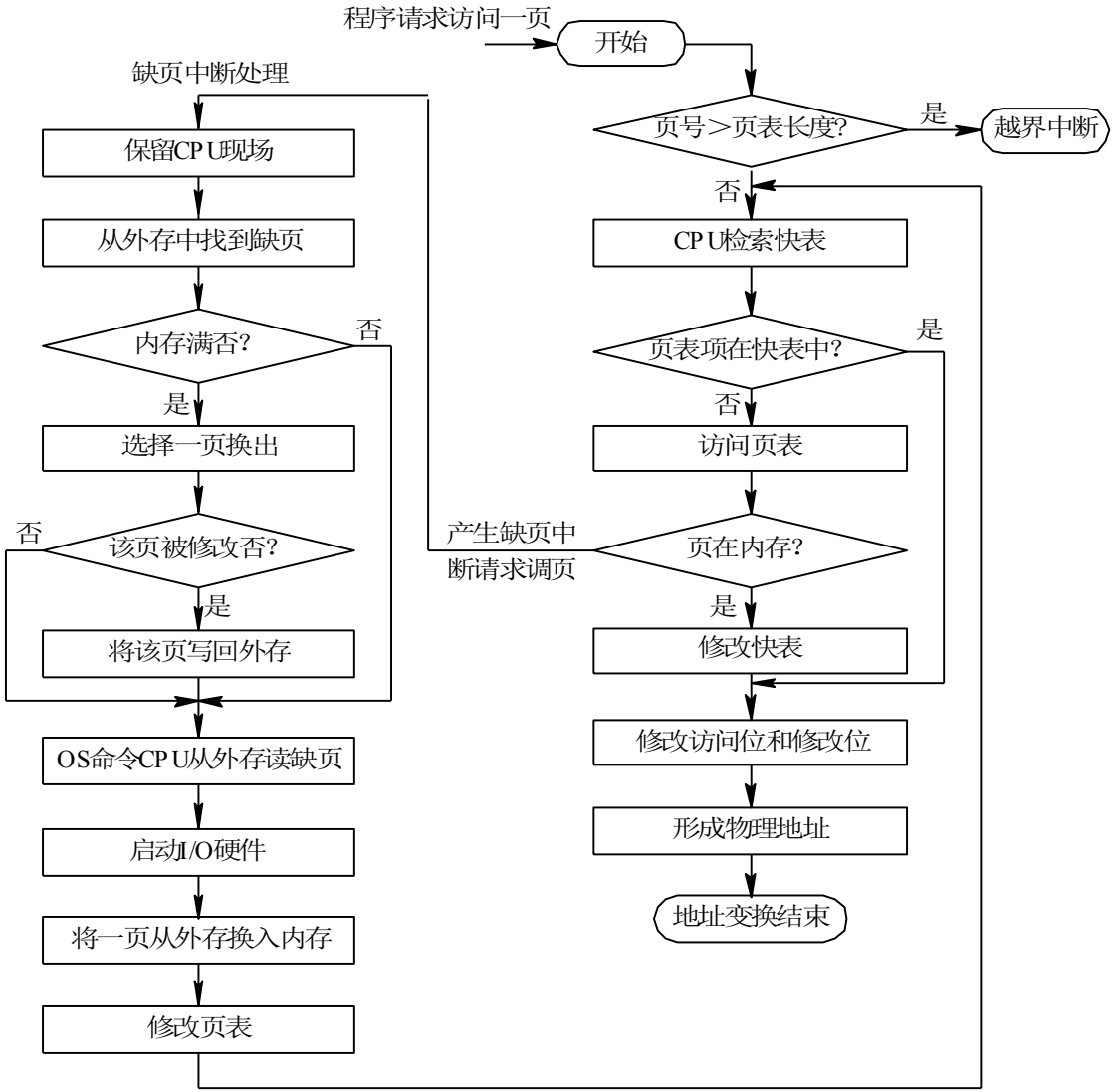


图 4-23 请求分页中的地址变换过程

4.6.2 内存分配策略和分配算法

1. 最小物理块数的确定

是指能保证进程正常运行所需的最小物理块数。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。进程应获得的最少物理块数与计算机的硬件结构有关，取决于指令的格式、功能和寻址方式。对于某些简单的机器，若是单地址指令且采用直接寻址方式，则所需的最少物理块数为2。其中，一块是用于存放指令的页面，另一块则是用于存放数据的页面。如果该机器允许间接寻址时，则至少要求有三个物理块。对于某些功能较强的机器，其指令长度可能是两个或多于两个字节，因而其指令本身有可能跨两个页面，且源地址和目标地址所涉及的区域也都可能跨两个页面。

2. 物理块的分配策略

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

2) 可变分配全局置换

3) 可变分配局部置换

3. 物理块分配算法

1) 平均分配算法

这是将系统中所有可供分配的物理块，平均分配给各个进程。 例如，当系统中有100个物理块，有5个进程在运行时，每个进程可分得20个物理块。这种方式貌似公平，但实际上是不公平的，因为它未考虑到各进程本身的大小。如有一个进程其大小为200页，只分配给它20个块，这样，它必然会有很高的缺页率；而另一个进程只有10页，却有10个物理块闲置未用。

3. 物理块分配算法

2) 按比例分配算法囍

这是根据进程的大小按比例分配物理块的算法。如果系统中共有n个进程，每个进程的页面数为S_i，则系统中各进程页面数的总和为：

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

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，将有：$b_i = \frac{S_i}{S} \times m$

b应该取整，它必须大于最小物理块数。

3. 物理块分配算法

3) 考虑优先权的分配算法囍

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，适当地增加其相应份额后，分配给各进程。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来为各进程分配其物理块的。

4.6.3 调页策略

1. 何时调入页面

1) 预调页策略囍

2) 请求调页策略

2. 从何处调入页面

- 在请求分页系统中的外存分为两部分：用于存放文件的文件区和用于存放对换页面的对换区。通常，由于对换区是采用连续分配方式，而事件是采用离散分配方式，故对换区的磁盘I/O速度比文件区的高。这样，每当发生缺页请求时，系统应从何处将缺页调入内存，可分成如下三种情况：
- (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。为此，在进程运行前，便须将与该进程有关的文件，从文件区拷贝到对换区

2. 从何处调入页面

(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，由于它们未被修改而不必再将它们换出，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。由于与进程有关的文件都放在文件区，故凡是未运行过的页面，都应从文件区调入。而对于曾经运行过但又被换出的页面，由于是被放在对换区，因此在下次调入时，应从对换区调入。由于UNIX系统允许页面共享，因此， 某进程所请求的页面有可能已被其它进程调入内存，此时也就无须再从对换区调入。

3. 页面调入过程

每当程序所要访问的页面未在内存时，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。该程序通过查找页表，得到该页在外存的物理块后， 如果此时内存能容纳新页，则启动磁盘I/O将所缺之页调入内存，然后修改页表。如果内存已满，则须先按照某种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；但如果此页已被修改， 则必须将它写回磁盘，然后再把所缺的页调入内存， 并修改页表中的相应表项，置其存在位为“1”，并将此页表项写入快表中。在缺页调入内存后，利用修改后的页表， 去形成所要访问数据的物理地址，再去访问内存数据。

第四章 存储器管理

4.7 页面置换算法

4.7.1 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。

1. 最佳(Optimal)置换算法

假定系统为某进程分配了三个物理块， 并考虑有以下的页面号引用串：

7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1

进程运行时， 先将7，0，1三个页面装入内存。 以后， 当进程要访问页面2时， 将会产生缺页中断。此时OS根据最佳置换算法， 将选择页面7予以淘汰。

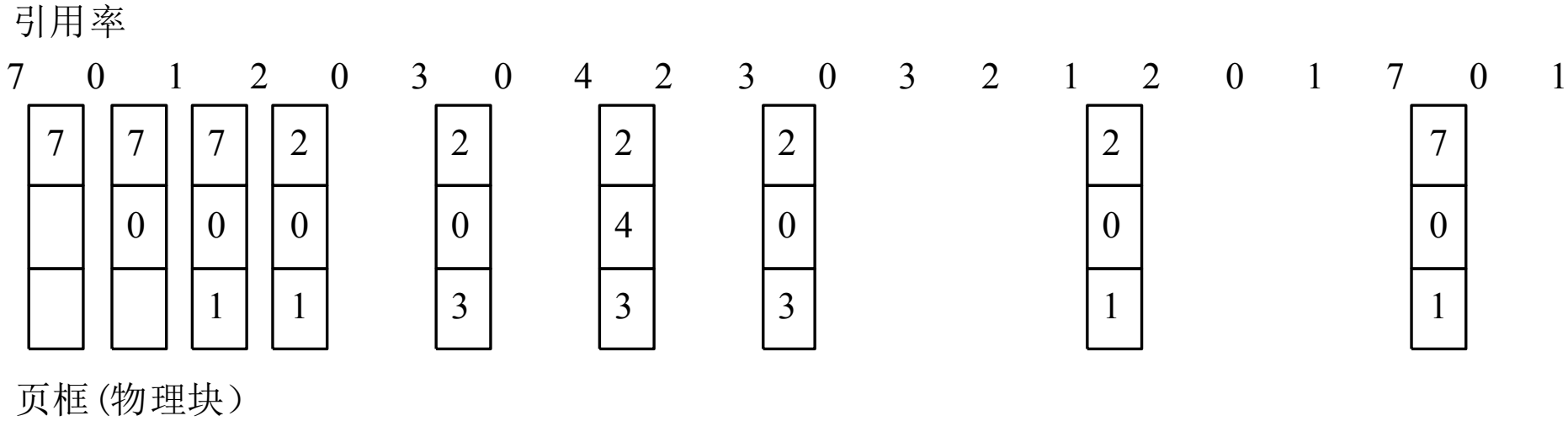


图 4-24 利用最佳页面置换算法时的置换图

2. 先进先出(FIFO)页面置换算法

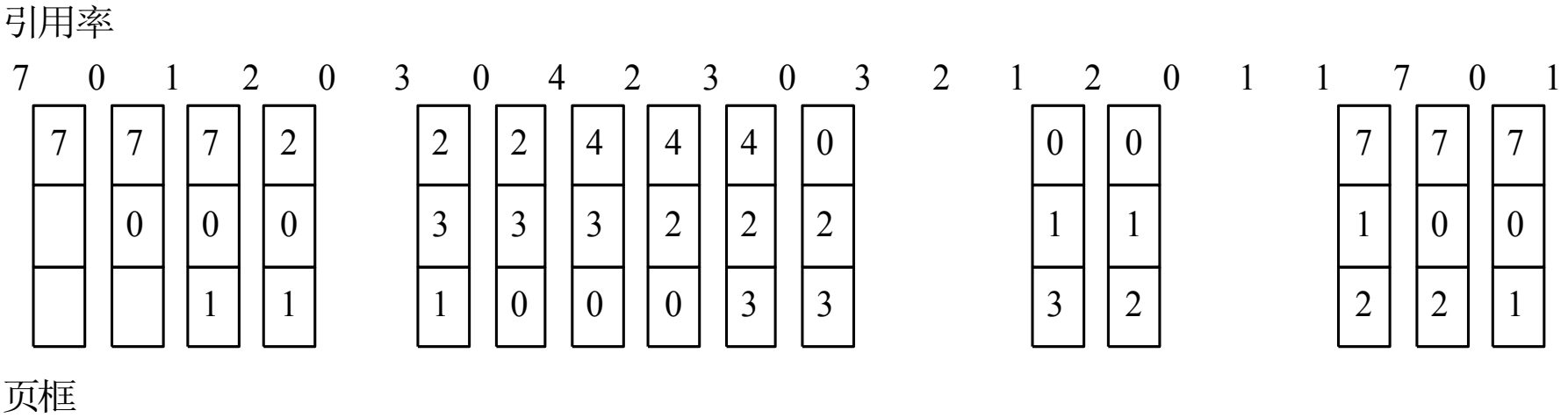


图 4-25 利用FIFO置换算法时的置换图

4.7.2 最近最久未使用(LRU)置换算法

1. LRU(Least Recently Used)置换算法的描述

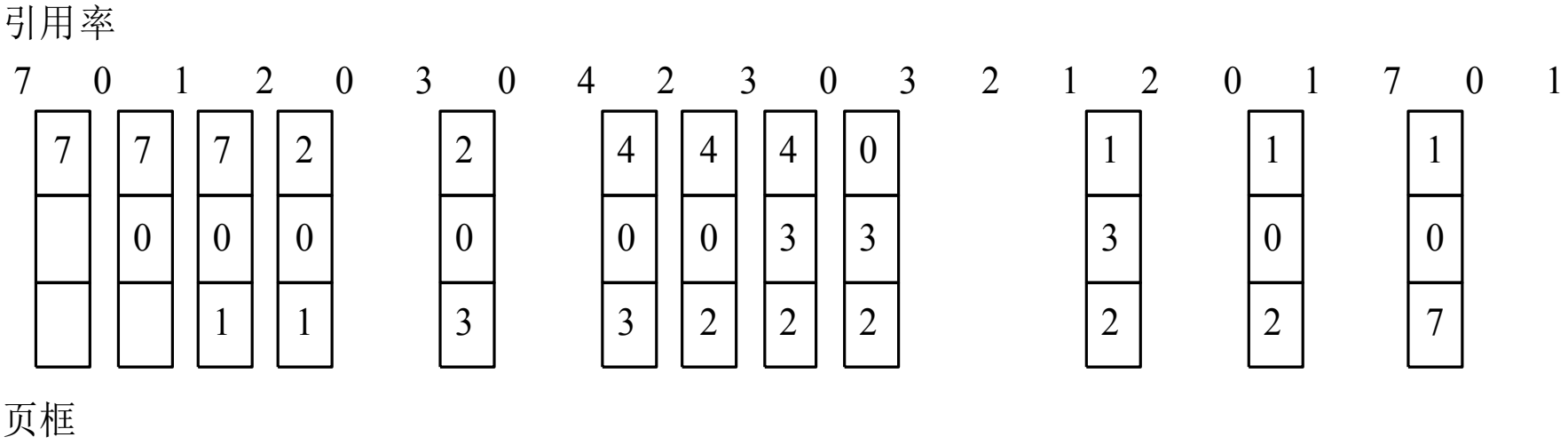


图 4-26 LRU页面置换算法

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

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

LRU置换算法示例

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

2) 栈

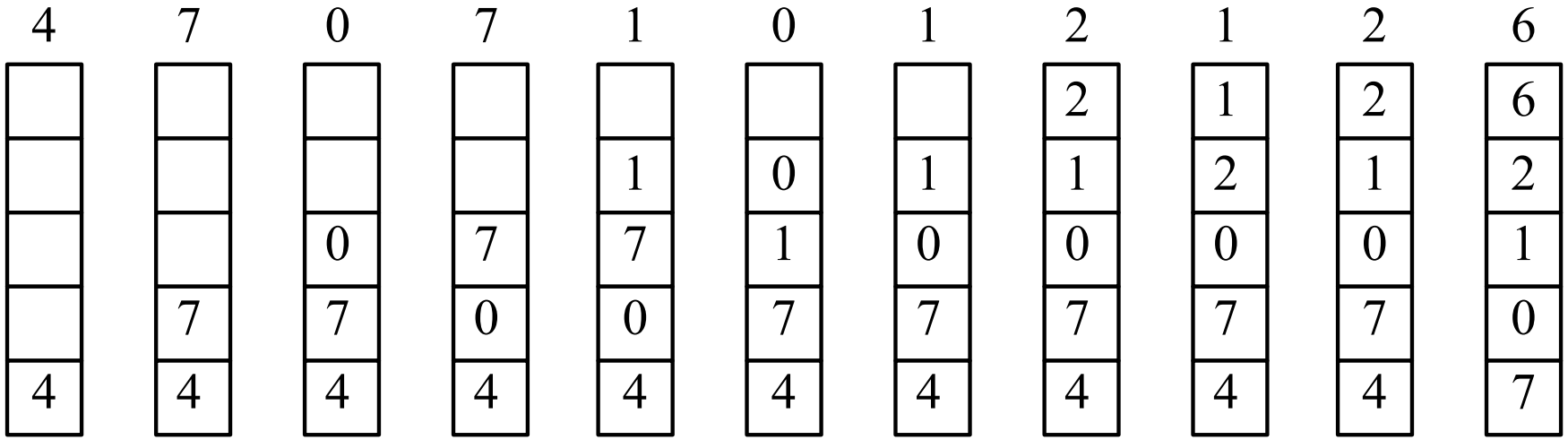


图 4-28 用栈保存当前使用页面时栈的变化情况

4.7.3 Clock置换算法

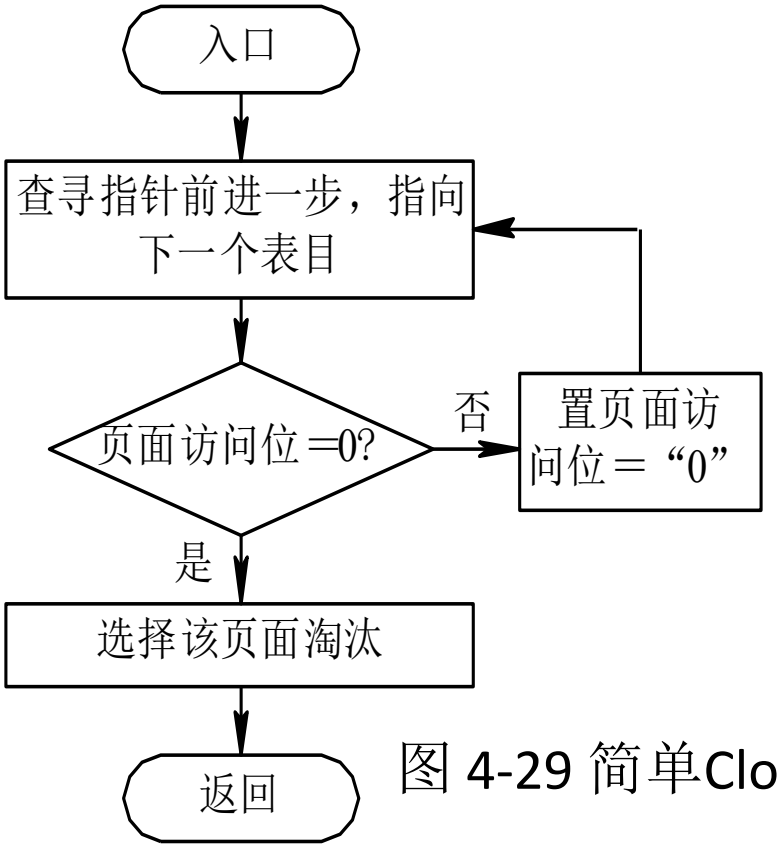


图 4-29 简单Clock置

块号	页号	访问位	指针
0			
1			
2	4	0	
3			
4	2	1	
5			
6	5	0	
7	1	1	

替换指针

4.7.4 其它置换算法

1. 最少使用置换算法

2. 页面缓冲算法

第四章 存储器管理

4.8 请求分段存储管理方式

4.8.1 请求分段中的硬件支持

1. 段表机制

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

1. 段表机制

在段表项中， 除了段名(号)、 段长、 段在内存中的起始地址外， 还增加了以下诸项：囂

(1) 存取方式。

(2) 访问字段A。

(3) 修改位M。

(4) 存在位P。

(5) 增补位。

(6) 外存始址。

2. 缺段中断机构

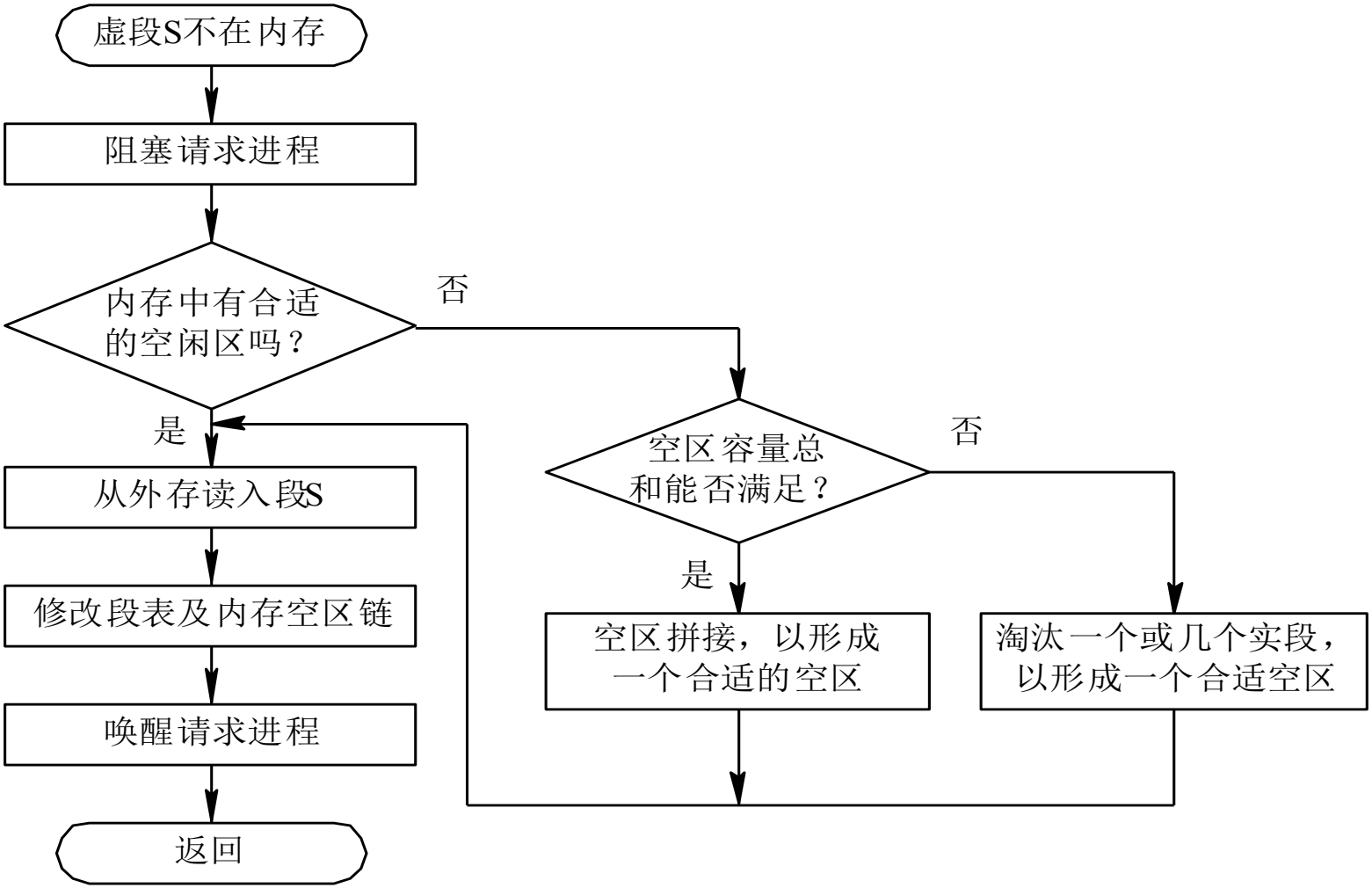


图 4-30 请求分段系统中的中断处理过程

3. 地址变换机构

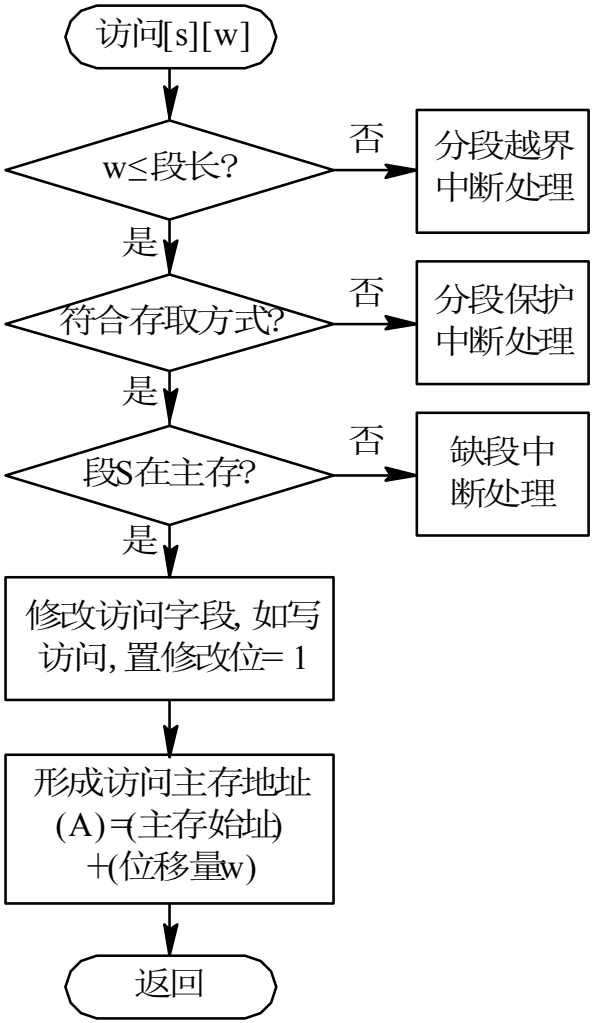


图 4-31 请求分段系统的地址变换过程

4.8.2 分段的共享与保护

1. 共享段表

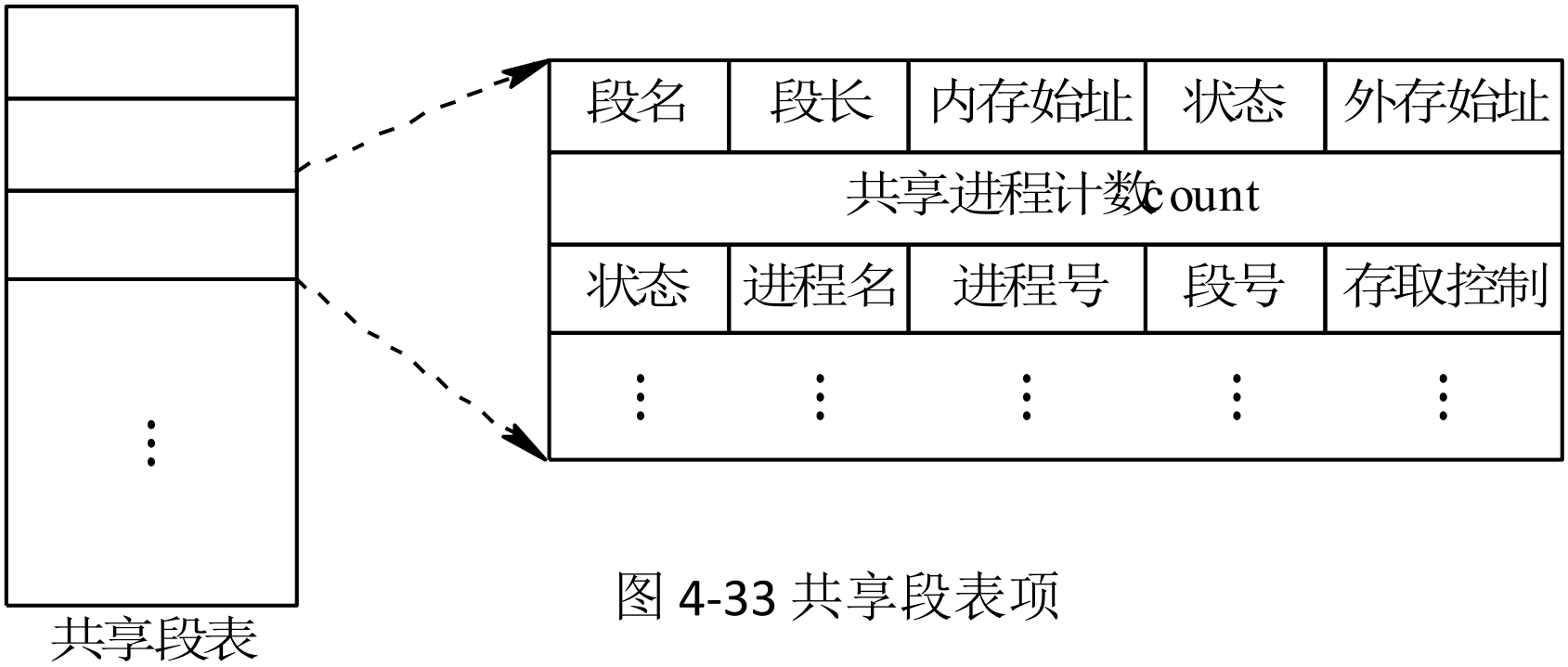


图 4-33 共享段表项

2. 共享段的分配与回收

1) 共享段的分配

在为共享段分配内存时，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表项，填写有关数据，把count置为1；之后，当又有其它进程需要调用该共享段时，由于该共享段已被调入内存，故此时无须再为该段分配内存，而只需在调用进程的段表中，增加一表项，填写该共享段的物理地址；在共享段的段表中，填上调用进程的进程名、存取控制等，再执行count：=count+1操作，以表明有两个进程共享该段。

2) 共享段的回收

- 当共享此段的某进程不再需要该段时，应将该段释放， 包括撤在该进程段表中共享段所对应的表项，以及执行count：=琎count-1操作。若结果为0，则须由系统回收该共享段的物理内存，以及取消在共享段表中该段所对应的表项， 表明此时已没有进程使用该段；否则(减1结果不为0)， 则只是取消调用者进程在共享段表中的有关记录。

3. 分段保护

1) 越界检查

2) 存取控制检查

(1)只读

(2) 只执行

(3) 读/写

3) 环保护机构

(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。