第四章　存储器管理

任立勇

2020年4月

目录

4.1 存储系统的层次结构

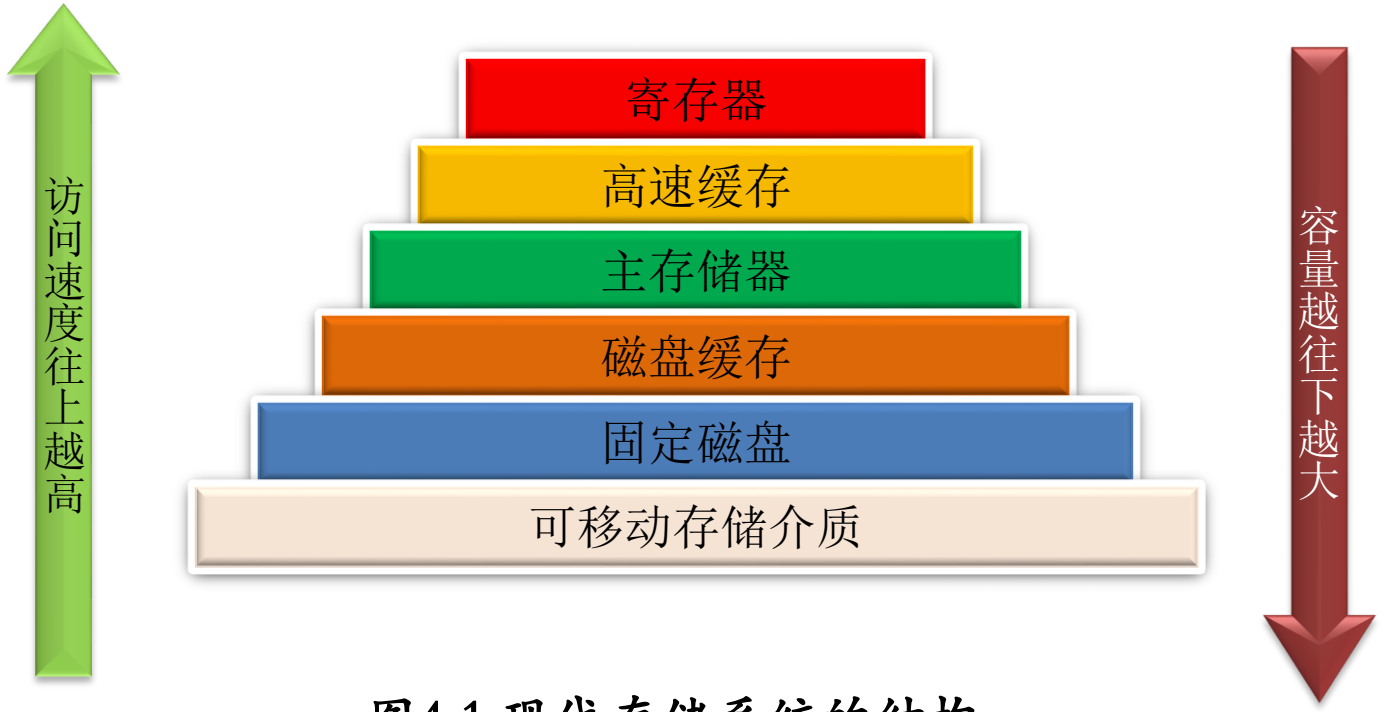


图4-1 现代存储系统的结构

4.1.2 主存储器与寄存器

- 寄存器与CPU有相同的速度，因此能完全与CPU协调工作，但价格却十分昂贵，因此其容量不可能做太大。
- 主存用于保存进程运行时的程序和数据，也称为可执行存储器。处理器从主存中取得指令和数据，分别放入指令寄存器和数据寄存器，或者反之
- 对主存器的访问速度远低于CPU执行指令的的速度，因此引入高速缓存，缓解这一矛盾

4.1.3 高速缓存和磁盘缓存

- 高速缓存介于寄存器和主存器之间的存储器，主要用于备份主存中较常用的指令和数据，以减少对主存器的访问次数。
- 高速缓存的访问比寄存器慢，但其访问速度却远快于主存。容量也介于二者之间。高速缓存的有效性是建立在“程序局部性原理”基础上
- 由于磁盘I/O速度远低于主存，因此设置了磁盘缓存，主要用于暂时存放频繁使用的一部分磁盘数据

4.2 程序的装入和链接

- 如何将一个用户源程序变成一个可在内存中执行的程序，通常要经过3步骤：

1. 编译:由编译程序（Compiler）将用户源代码编译成若个*目标模块*。
2. 链接:由链接程序（Linker）将编译后形成的一组目标模块，以及它们所需要的库函数链接在一起，形成一个完整的*装入模块*。
3. 装入:由装入程序（Loader）将装入模块装入内存。

4.2 程序的装入和链接

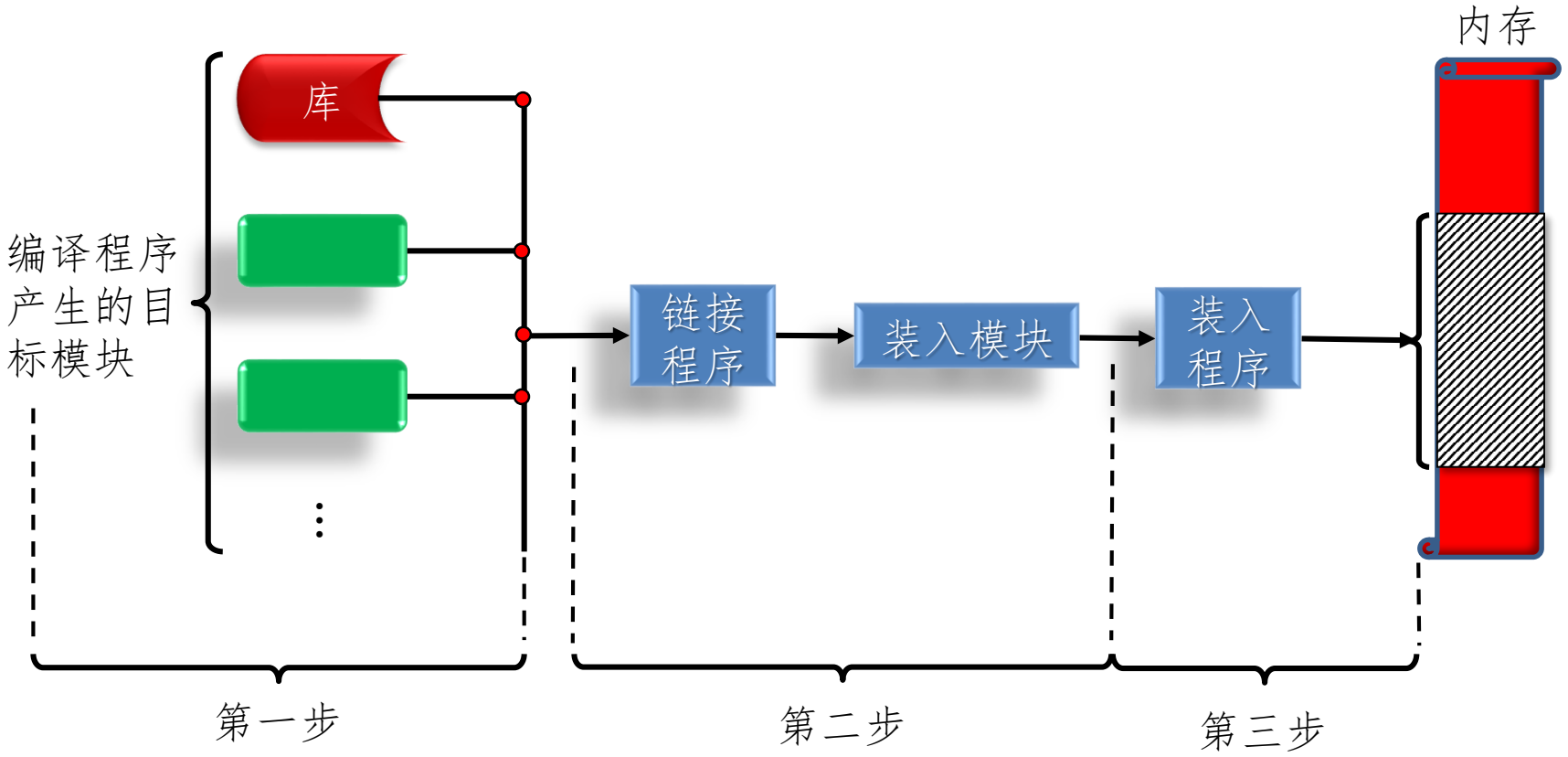


图4-2 对用户程序的处理步骤

4.2.1 程序的装入

(1)．绝对装入方式

如果知道程序将驻留在内存的什么位置，那么，编译程序将产生绝对地址的目标代码。

绝对装入程序按照装入模块中的地址，将程序和数据装入内存。装入模块被装入内存后，由于程序中的*逻辑地址与实际内存地址完全相同，*故不需对程序和数据的地址进行修改。

4.2.1 程序的装入

(2)．可重定位装入方式

- 由装入程序将装入模块装入内存后，装入模块中程序所访问的所有逻辑地址与实际装入内存的物理地址不同 ，必须进行变换。
- *把在装入时对目标程序中指令和数据的变换过程称为重定位。*
- 采用静态重定位方法将程序装入内存，称为可重定位装入方式。
- 因为地址变换是在装入时一次完成的，以后不再改变，故称为静态重定位。

4.2.1 程序的装入

2. 可重定位装入方式(Relocation Loading Mode)

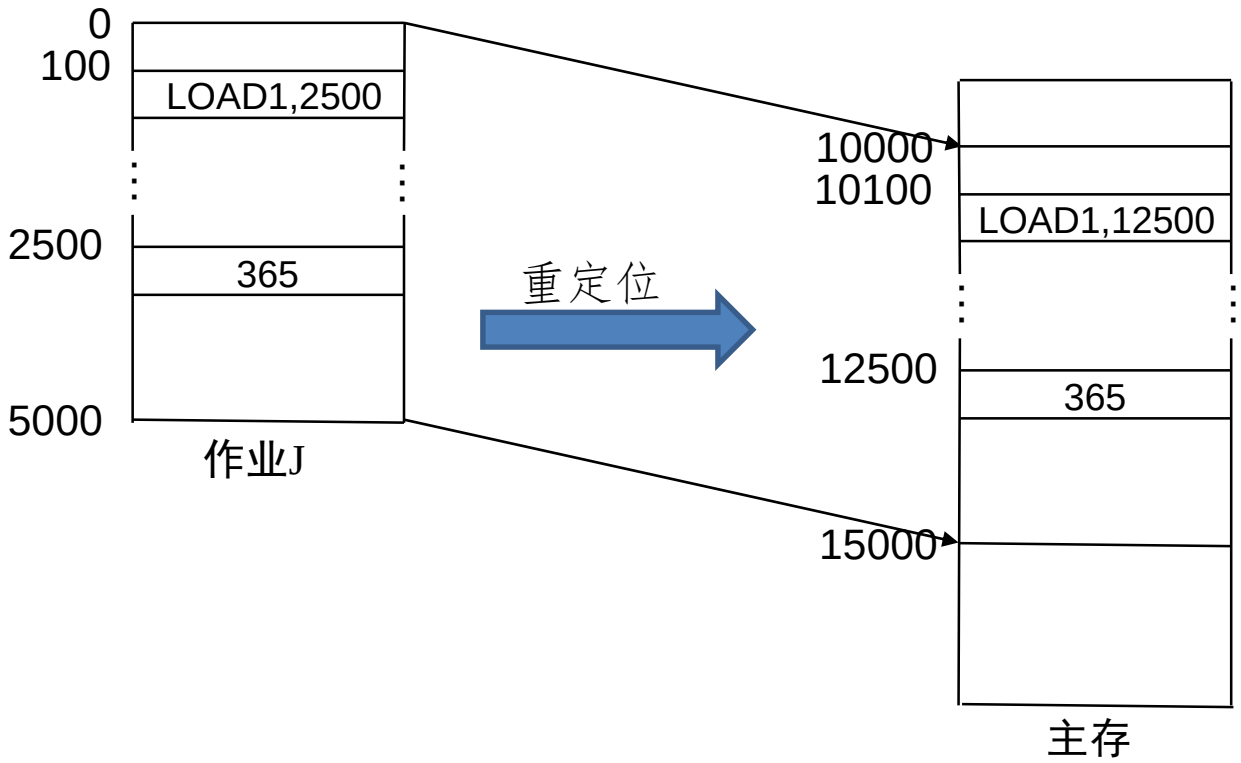


图 4-3 作业装入内存时的情况

4.2.1 程序的装入

(3). 动态运行时装入方式

- 装入程序将目标模块装入内存后，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执行时进行*，在硬件地址变换机构的支持下，随着对每条指令或数据的访问自动进行地址变换，故称为动态重定位。

4.2.1 程序的装入

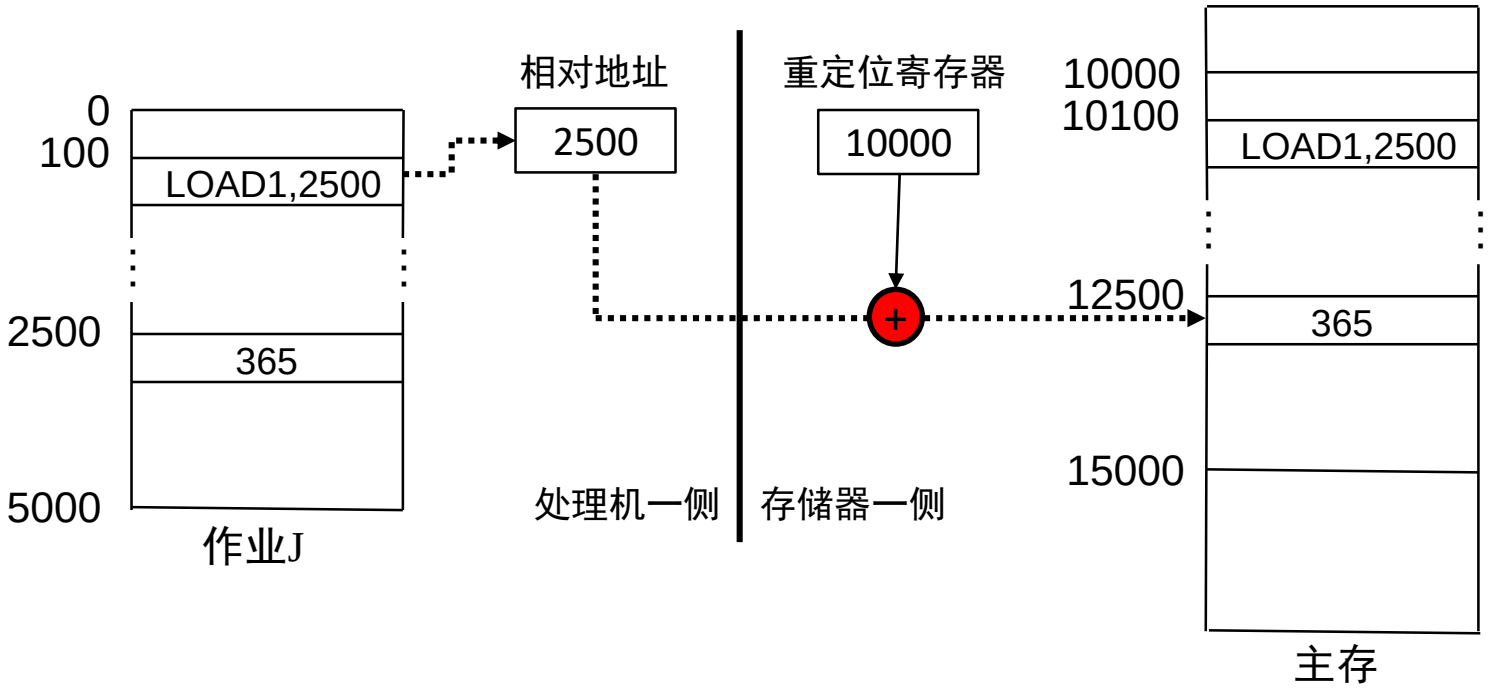


图4-12 动态重定位示意图

•采用动态重定位方法将程序装入内存，称为动态运行时装入方式 。

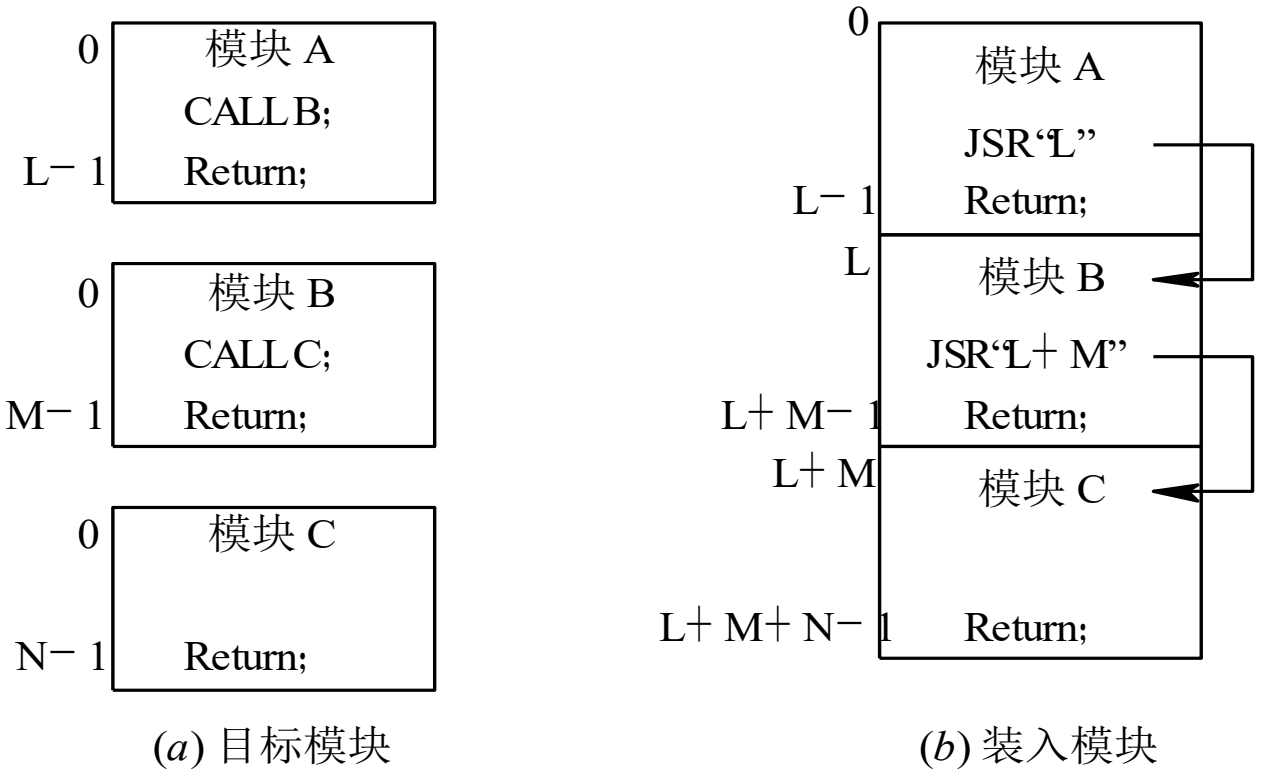
4.2.2 程序的链接

★源程序经过编译后，可得到一组目标模块，再利用链接程序将这组目标模块链接形成装入模块。*根据链接时间的不同*，可把链接分成如下三种：

(1)、静态链接方式。在程序运行之前，先将各目标模块及它们所需的库函数，链接成一个完整的装配模块（又称执行模块），以后不再拆开。我们把这种事先进行链接的方式称为静态链接方式。

程序的链接

1. 静态链接方式(Static Linking)



(*a*) 目标模块　　(*b*) 装入模块

图 4-4 程序链接示意图

程序的链接

①对相对地址进行修改

由编译程序产生的所有目标模块中，使用的都是相对地址，其起始地址都为0，在链接成一个装入模块时修改模块的相对地址。如把原B中的所有相对地址都加上L，把原C中所有相对地址都加上L＋M。

②变换外部引用地址

将每个模块中所用的外部调用符号也都变换为相对地址。例如将call B 变换为JSR “L”

程序的链接

(2)、装入时动态链接

是指将用户源程序编译后所得到的一组目标模块，在装入内存时，采用*边装入边链接*的链接方式。即*分别装入各模块，并且在装入的过程中修改相对地址和外部引用地址。*

程序的链接

- 装入时动态链接方式有以下优点：

①便于修改和更新

若采用动态链接方式，由于各目标模块是分开存放的，所以要修改或更新各目标模块，是件非常容易的事。

②便于实现对目标模块的共享

采用装入时动态链接方式时，OS则很容易将一个目标模块链接到几个应用模块上，实现多个应用程序对该模块的共享。

程序的链接

(3)、运行时动态链接

各模块被独立装入系统，而且也不进行链接，运行时发现引用的地址是相对地址或者外部地址时，才发起链接，寻找正确的引用地址。

优点：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。

该方法是目前最常使用的链接方式。

4.3 连续分配方式

- 连续分配方式，是指为一个用户程序分配一个连续的内存空间。
- 连续分配方式有四种：
  1. 单一连续分配
  2. 固定分区分配
  3. 动态分区分配
  4. 动态重定位分区分配

4.2.1 单一连续分配

- 这是最早、最简单的一种存储分配方式。它规定整个内存的用户区中只驻留一个用户的一个程序，因此该方式只适用于单用户、单任务的操作系统。

4.2.1 单一连续分配

- 为了防止OS的代码和数据被用户进程所破坏，把内存分为系统区和用户区两部分：

1. 系统区：仅提供给0S使用，通常是放在内存的低址部分；
2. 用户区：是指除系统区以外的全部内存空间，提供给唯一的用户使用，存放用户程序和数据。

- 优缺点：简单、内存利用率低。

4.2.2 分区管理-固定分区分配

固定分区分配思想：将内存用户空间划分为若干个固定大小的区域，每个区域称为一个分区（region），在每个分区中只装入一道作业，从而支持多道程序并发设计。

(1). 分区大小

①分区大小相等。当程序太小时，会造成内存空间的浪费 。当程序太大时，一个分区又不足以装入该程序，致使该程序无法运行。

②分区大小不等。可把内存区划成含有多个较小的分区、适量的中等分区及少量的大分区。

4.2.2 分区管理-固定分区分配

(2)．分区分配

1、如何知道哪些分区可以分配？

采用分区描述表记录每个分区的状态信息，如下图所示。

分区编号	大小（KB）	起始地址（KB）	状态
1	30	30	分配
2	40	60	分配
3	30	100	分配
4	40	130	未分配
5	40	170	分配

分区描述表

内存分配情况

3.2.2 分区管理-固定分区分配

- 2、如何分配各分区？

当有作业要装入内存时，内存分配程序检索分区描述表，从中找出尚未使用的最接近大小的分区分配给该作业，然后修改分区的状态；如果找不到合适的分区就拒绝为该作业分配内存。

当程序运行完成时，系统回收内存资源，并修改分区描述表中分区的状态。

4.2.2 分区管理-固定分区分配

- 固定分区式分配 的优缺点：
  - 最早的可运行多道程序的存储管理方式 。
  - 存在“内零头”会造成存储空间的浪费。
- 内零头——在分区内没有利用的部分称为内零头。

- 适用于控制多个相同对象的控制系统中

4.2.2 分区管理-动态分区方式

动态分区分配思想：分区数量和大小都不固定，根据进程的实际需要，动态地为之分配内存空间。

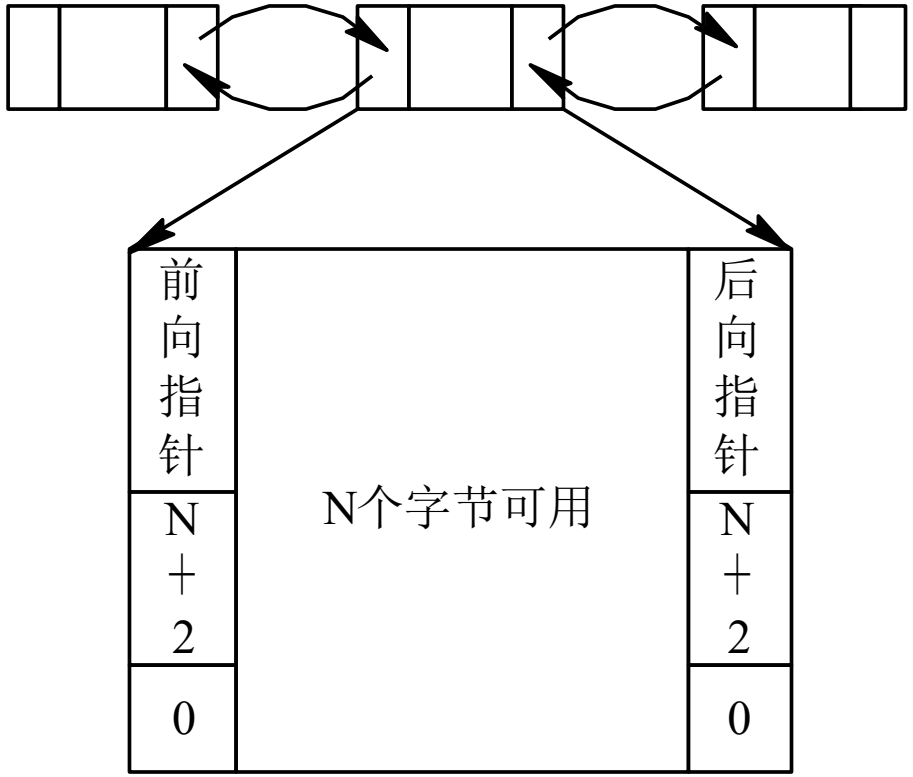
(1)分区分配数据结构

记录系统中各空闲分区的情况，以便分配时能找到可以分配的空间。

①空闲分区表。在系统中设置一张空闲分区表，用于记录每个空闲分区的情况（包含空闲分区号、分区大小、起始地址）。

②空闲分区链表。为了实现对空闲分区的分配和链接，设置前向指针和后向指针，通过前、后向链接指针将所有的空闲分区链接成一个双向链。

空闲分区链表



3. 分区分配操作

1) 分配内存

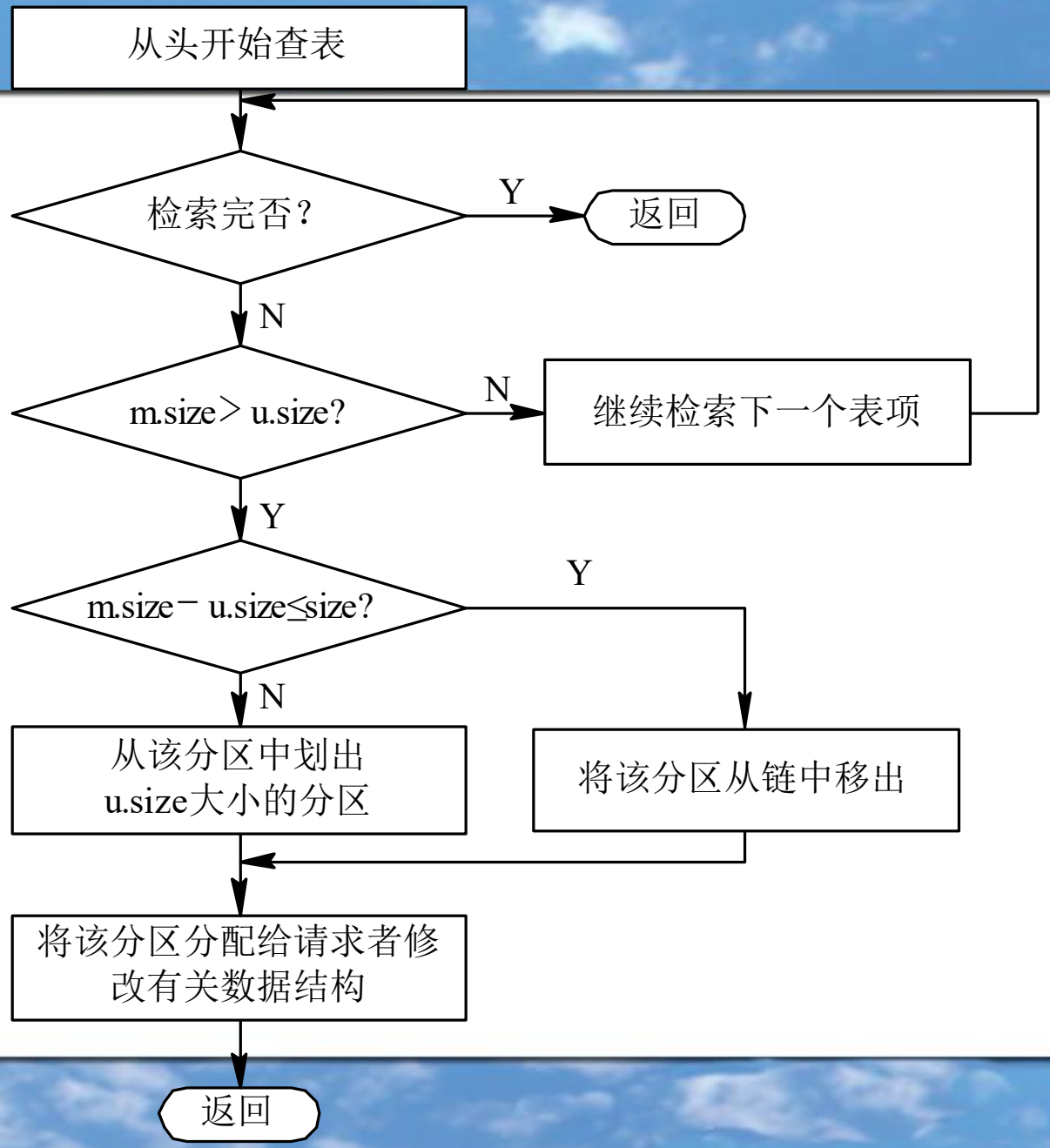
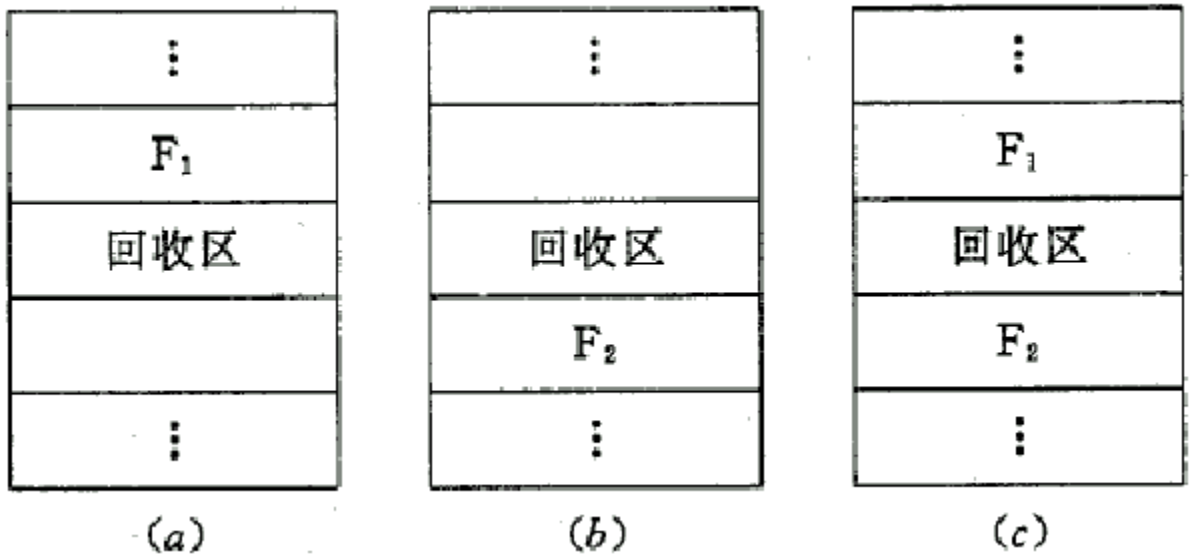


图 4-8 内存分配流程

（3）分区回收

- 当进程运行完毕释放内存时，需合并相邻的空闲分区，形成大的分区，称为合并技术。
- 需要合并的情况有如下图所示的三种，不论哪种情况，只需修改相应的分区信息来完成合并即可。



回收分区前有空闲分区　回收分区后有空闲分区　回收分区前后都有空闲分区

(2) 分区分配算法

①首次匹配(首次适应算法FF)：要求*空闲分区链以地址递增的次序链接*。在分配内存时，从链首开始顺序查找，直至找到一个大小能满足要求的空闲分区为止。

优缺点：

- 利用内存低地址部分空闲分区，为大作业分配大的内存空间创造了条件。
- 低址部分不断被划分，会留下许多难以利用的、很小的空闲分区。
- 增加查找开销

(2) 分区分配算法

②循环匹配(循环首次适应算法)：将所有的空闲分区构成一个循环链表。每次查找时不是从头开始，而是从上次结束位置开始。

优缺点：能使内存中的空闲分区分布得更均匀，从而减少了查找空闲分区时的开销，但这样会缺乏大的空闲分区。

(2) 分区分配算法

③最佳匹配(最佳适应算法):

该算法要求将所有的空闲分区按其容量以从小到大的顺序形成一空闲分区链。从头开始查找，*将表中第一个大于所需求空间大小的空闲区分配给作业*。

优缺点：为大作业分配大的内存空间创造了条件。每次分配后所切割下来的剩余部分总是最小的，这样，在存储器中会留下许多难以利用的小空闲区。

(2) 分区分配算法

④ 最坏适应算法

- 空闲分区按其容量从大到小排成空闲分区表
- 总是挑选一个最大的空闲分区分割给作业使用
- 优点：使剩下的空闲分区不至于太小，产生碎片几率最小，对中、小作业有利，查找效率高
- 缺点：会使存储器中缺乏大的空闲分区

- 以上几种算法都属于顺序搜索法

(2) 分区分配算法

⑤ 快速适应算法（分类搜索法）

- 对于每一类相同容量的空闲分区单独设立一个空闲分区链表
- 设置管理索引表，每个表项对应一种空闲分区类型，并记录该类空闲分区链表表头指针
- 优点：
  - 查找效率高
  - 不会对任何分区产生分割，能保留大的分区，也不会产生内存碎片
- 缺点：
  - 分区归还主存时算法复杂，系统开销较大。
  - 内零头

6 伙伴系统

- 伙伴系统是一种不需要紧凑的动态分区算法。
- 伙伴系统是内存块管理机制，采用二进制数的方式来分配和回收空间。提高回收空间时合并空闲分区的速度，Linux操作系统使用该算法分配和回收页面块。

6 伙伴系统

- 所有分区大小均为2的k次幂（$1 \le k \le m$），通常2^m是整个可分配内存的大小。
- 刚开始时，整个内存区是一个大小为2^m的空闲分区，由于不断划分，将会形成若干个不连续的空闲分区。对于具有相同大小的所有空闲分区，单独设立一个空闲分区双向链表。
- 当进程需要长度为n的存储空间，首先计算一个i值，使$2^{i-1}<n\le 2^i$。然后在大小为2^i空闲分区链表中查找，若找到，则分配，若未找到，则在大小为2^{i+1}的空闲分区链表中查找，若找到，则将该分区分相等的两个分区，一个分配，一个放入2^i的链表中，若未找到，则继续查找2^{i+2}的分区链表，依次类推。

6 伙伴系统

- 与一次分配要进行多次分割一样，一次回收也可能要进行多次合并，如回收大小为2^i的空闲时，若事先已存在2^i的空闲分区，则应将其与伙伴分区合并为大小为2^{i+1}的空闲分区，若事先已存在大小为2^{i+1}的空闲分区，又应与其合并为大小为2^{i+2}的空闲分区，依次类推

Linux内存管理

- Linux采用多种内存分配策略，2.4版采用伙伴系统，原理：
  - 把所有的空闲页面块分为10个块组，每组中块的大小为2的n幂次，如：
    - **第0组块大小 2^0=1页**
    - **第1组块大小 2^1=2页**
    - **第2组块大小 2^2=4页….**
    - **第9组块大小 2^9=512页**
  - 将相同大小的块组织成一个队列。并用位图表示该队列是否有空闲块，如果有则为1；否则为0。
  - 将多个队列组织成一个表，设置位图数组存放每个队列的位图。

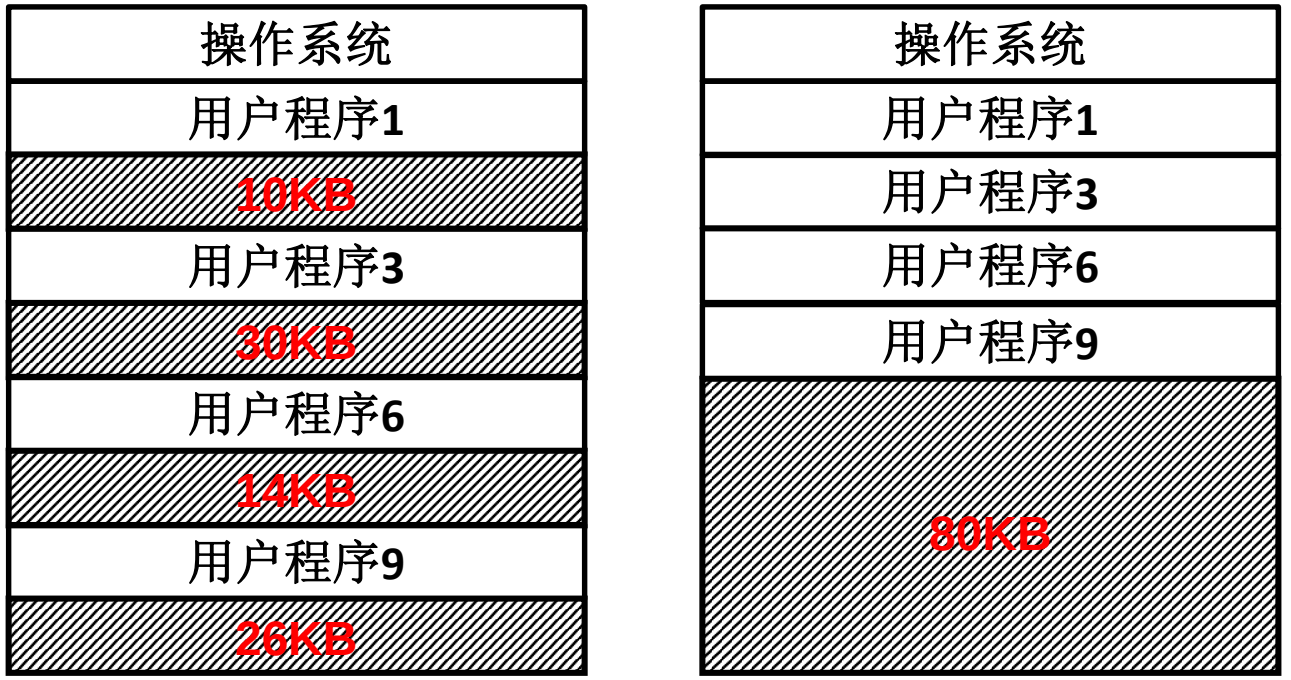
- 优点：分配和回收内存速度快，且不会产生很多小碎片。
- 缺点：内存利用率不高，分配的内存大小为2的幂，，假如只需要65个页面，也需要分配128个页面的块，从而浪费了63个页面，即产生内部碎片。

4.3.6 分区管理-紧凑与动态重定位

- 上述的连续分配方式的分配算法都不可避免的产生大量无法利用的碎片，这些无法利用的空闲分区称为“零头”或“碎片”。

- 紧凑技术—将内存中的所有作业进行移动，使它们全都相邻接，这样，可把原来分散的多个小分区合成一个大分区的方法，称为紧凑。

4.3.6 分区管理-紧凑与动态重定位



(a)紧凑前　　(b)紧凑后

紧凑示意图

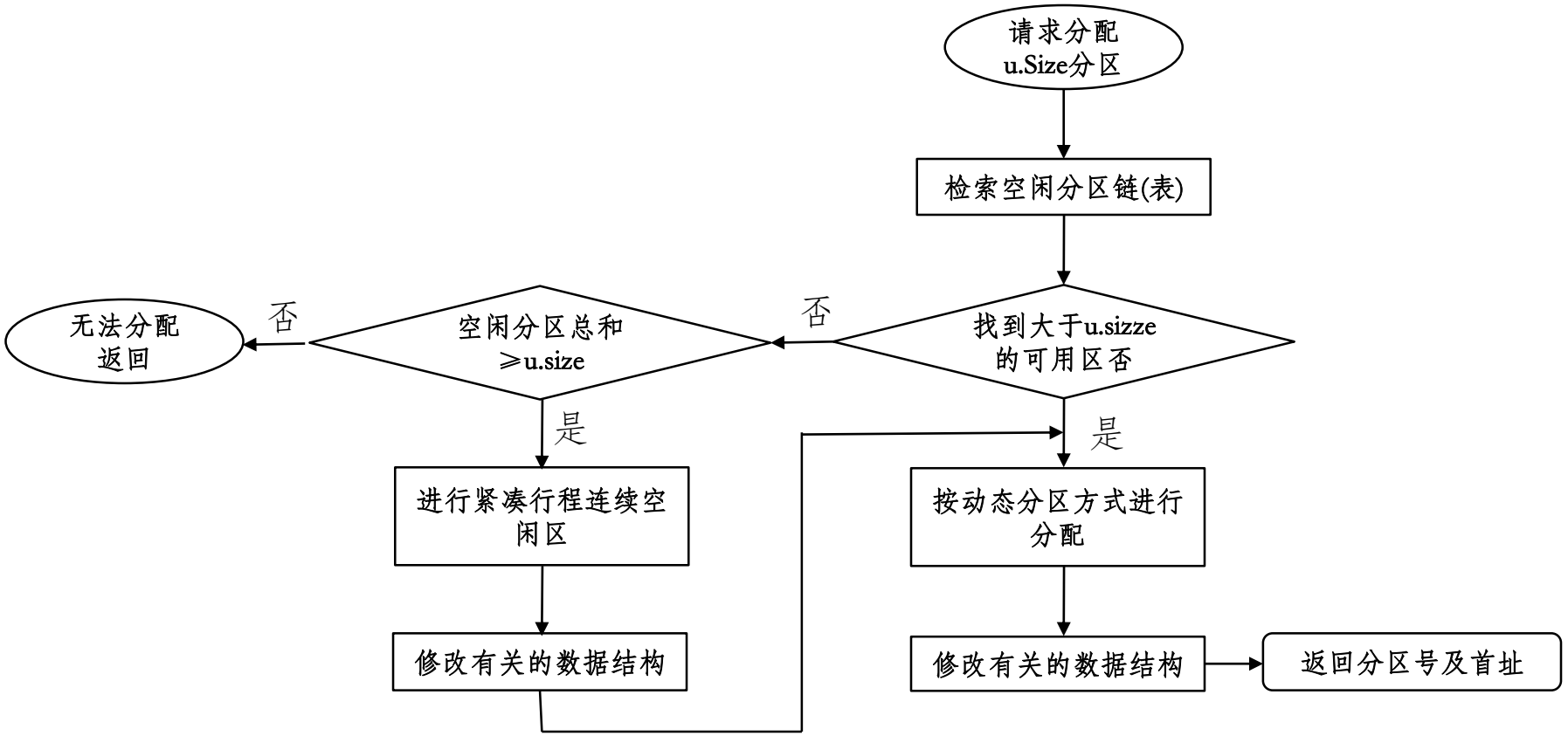
4.3.6 分区管理-紧凑与动态重定位

- 改变程序驻留位置后的地址重定位问题解决方法：

  动态重定位机制需要硬件的支持，增加重定位寄存器（分区首址寄存器），保存分区起始地址，当进行紧凑内存空间时，只需移动分区内容，然后用新的分区起始地址重置分区首址寄存器。

程序中的相对地址= 重定位寄存器+ 相对地址

动态重定位分区分配算法



启动紧凑的时机：1.载入大程序，而无可用空间时，系统自动启动紧凑，2.发现碎片较多，系统比较空闲时。

4.3.7 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。

4.3.7 对换

- 如果对换是以整个进程为单位，便称之为“整体对换”或“进程对换”。
- 如果对换是以“页”或“段”为单位进行的，则称之为“页面对换”或“分段对换”，又称为“部分对换”
- 为了实现进程对换，系统必须能实现三方面的功能：
  - 对换空间的管理
  - 进程的换出
  - 进程的换入

4.3.7 对换

2．对换空间的管理

- 在具有对换功能的OS中，通常把外存分为文件区和对换区。
  - 文件区：为提高存储空间利用率，采用离散分配方式；
  - 对换区：为提高进程换入换出的速度，采用连续分配方式
- 进程在对换区中驻留的时间是短暂的、对换操作较频繁.
- 设置空闲分区表或链表，记录对换区首址（盘块号）、大小（盘块数）
- 分配算法、对换空间的分配与回收与内存动态分区雷同

4.3.7 对换

3．进程的换出与换入

（1）进程的换出

– 系统选择处于阻塞状态且优先级最低的进程作为换出进程，将该进程的程序和数据传送到磁盘的对换区上。

– 然后回收该进程所占用的内存空间，并对该进程的进程控制块做相应的修改。

4.3.7 对换

（2）进程的换入 （一般由调度程序实现）

- 系统定时地查看所有进程的状态，从中找出“就绪”状态但已换出的进程；
- 将其中换出时间最久的进程作为换入进程；
- 有能满足进程需要的内存时可将之换入。

注意：为了避免频繁的换进换出，设置换出的一个时间限制。例如在内存至少驻留2秒钟才能换出。

思考

- 连续分配方式有什么优缺点？
- 有没有更好的分配方式？

4.5 基本分页存储管理方式

- 离散分配方式:为进程分配空间不要求具有连续的空间,可以是多个分离的空间为进程占用。
- 采用离散分配方式的存储管理有：
  - 分页存储管理
  - 段式存储管理
  - 段页式存储管理

4.5.1 页面与页表

1.页面

1）页面和物理块

- 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。
- 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。
- 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”或称为“内零头”。

4.5.1 页面与页表

2）页面大小

- 如果页面太小，虽然可使内存碎片减小，但也会使进程**页表**过长，并减低页面换入换出的效率；
- 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1KB～8KB。

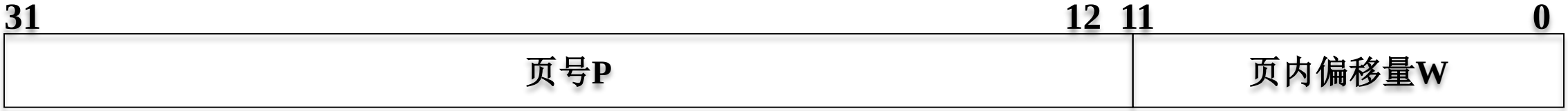
页面与页框大小

- 影响页面与页框大小的主要因素：页内零头、地址转换速度和页面交换效率。
- *较小的页面*有利于减少内零头，从而有利于提高内存的利用率。然而，*较小的页面*也将导致页表过大，从而降低处理机访问页表时的命中率(Hit Rate)。
- 块越大，内/外存之间的数据交换效率越高。因此，对于支持交换技术的系统，*较大的页面*有利于提高页面换进/换出内存的效率。

4.5.1 页面与页表

2、地址结构

- 32位机的分页存储逻辑地址结构：每页大小为4K，地址空间最多允许有1M页



分页管理的逻辑地址表示

- 假设页面大小为L，给定逻辑地址A，则可计算该地址的页号P及业内偏移量W

$$P = INT\left[\frac{A}{L}\right], \qquad W = [A] mod\ L$$

4.5.1 页面与页表

3. 页表

- 在分页系统中，进程可能有多个页，允许将进程的每一页离散地存储在内存的任一物理块中。
- 系统应能保证进程的正确运行，即能在内存中找到每个页面所对应的物理块。
- 为此，系统又为*每个进程建立了一张页面映像（射）表，简称页表*。

4.5.1 页面与页表

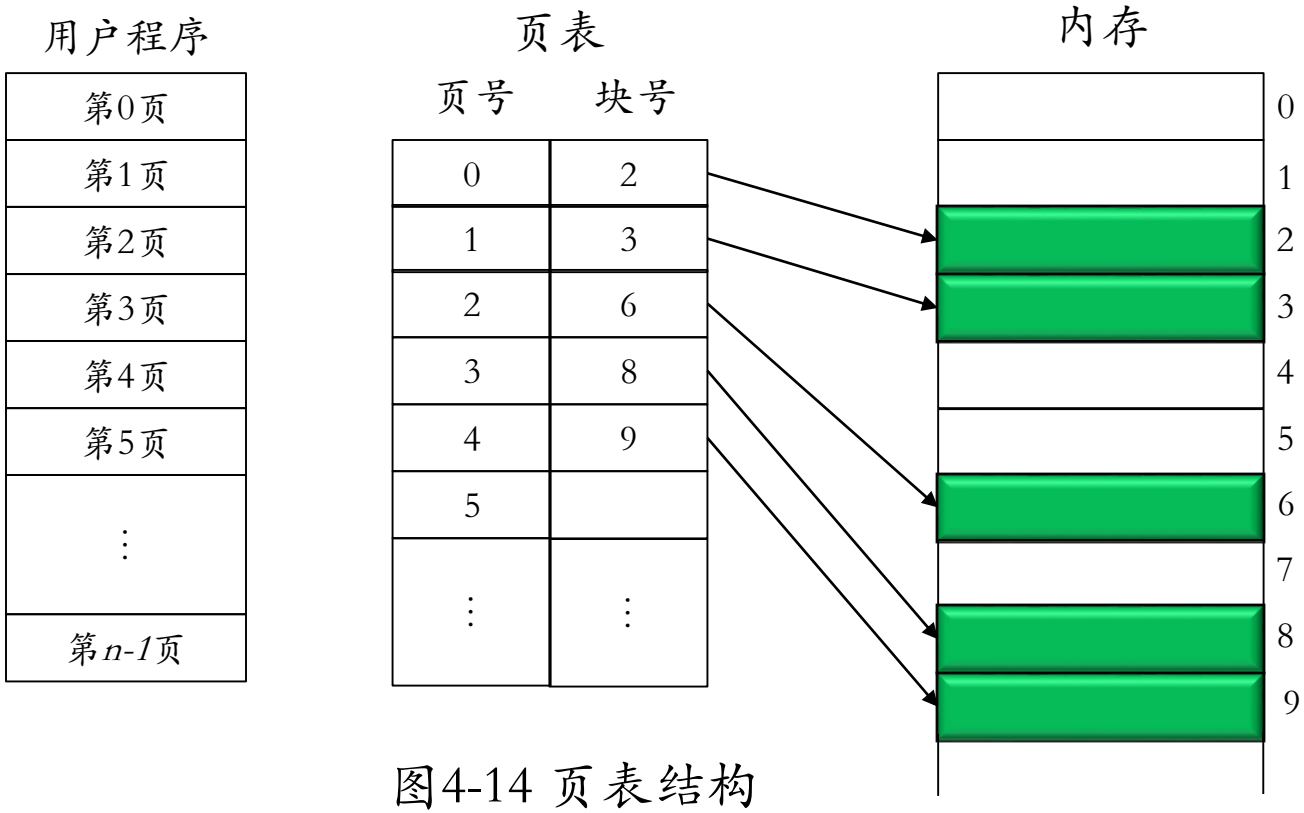
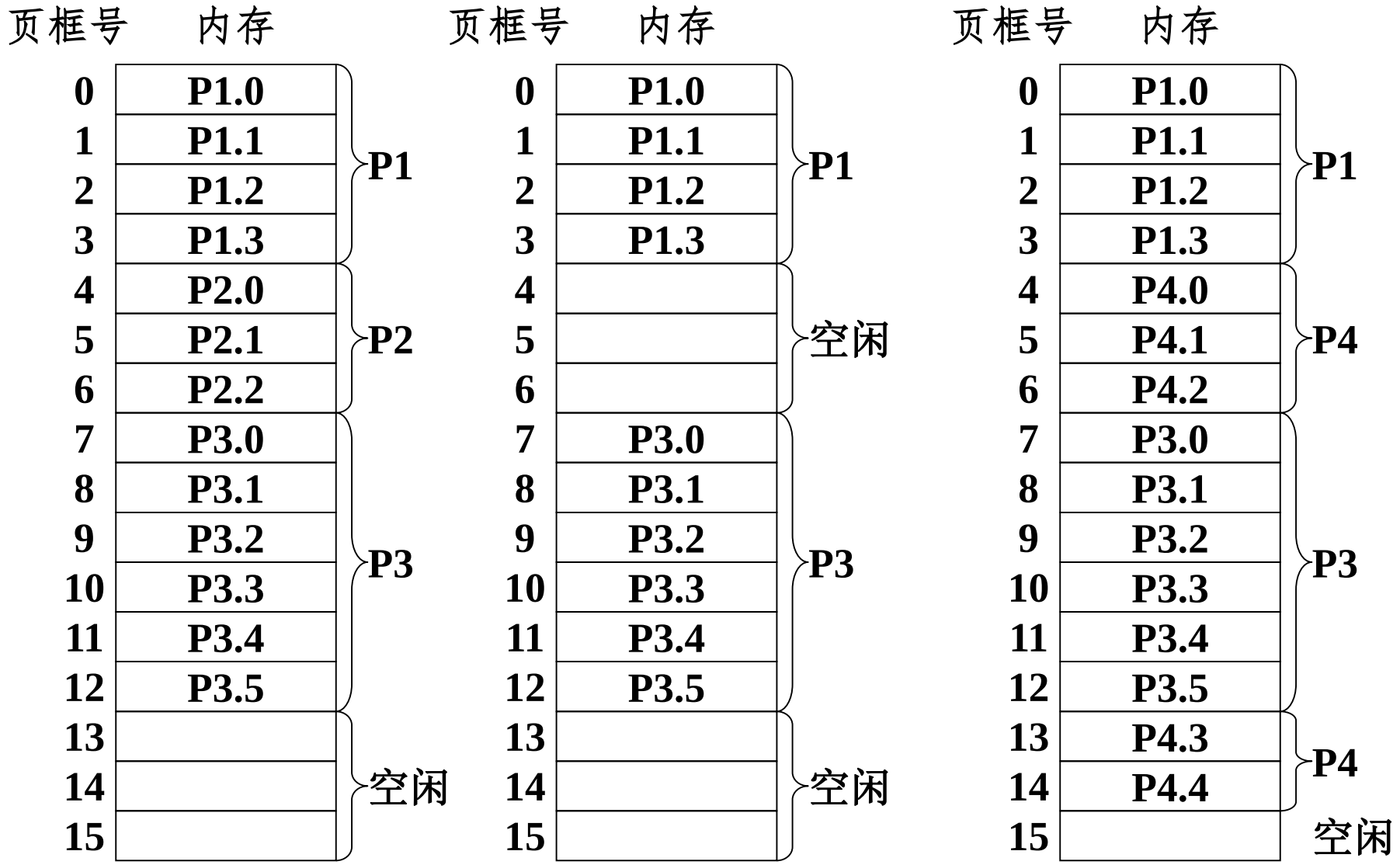


图4-14 页表结构

注：页表项中还应增加一些字段，例如增加控制字段用以保护页块中的内容

例如

- 假设内存能提供16个空闲页框，进程P1被分割成4个页面，装入内存中的0号至3号页框。进程P2被分割成3个页面，装入4号至6号页框。进程P3被装入7号至12号页框。
- 此时，进程P4请求分配5个页框大小的存储空间，但内存只有3个空闲页框。于是，暂时将不运行的P2交换出内存。
- 然后，再将P4装入4、5、6、13、14号页框。



进程装入到离散的页框中

各进程的页表结构及其内容

页号	页框号
0	0
1	1
2	2
3	3

进程P1页表

页号	页框号
0	-
1	-
2	-

进程P2页表

页号	页框号
0	7
1	8
2	9
3	10
4	11
5	12

进程P3页表

页号	页框号
0	4
1	5
2	6
3	13
4	14

进程P4页表

4.5.2 地址变换

1、基本的地址变换机构

- 硬件机制，实现逻辑地址到物理地址的转换。在系统中只设置一个页表寄存器PTR (Page-Table Register)，在其中存放当前运行的进程的页表在内存中的起始地址，和此进程的页表长度。
- 当进程真正投入运行时，从进程控制块PCB中读出其页表起始地址，并用它设置页表寄存器，以后地址转换时直接从PTR中获得页表的起始地址。

4.5.2 地址变换

- 分页系统中的地址变换过程如下：

（1） 根据逻辑地址,计算出页号和页内偏移量；

（2）从PTR中得到页表首址，然后检索页表，查找指定页面对应的页框号；

（3）用页框号乘以页面大小获得其对应页框的起始地址，并将其送入物理地址的高端。

（4）将页内偏移量送入物理地址低端，形成完整的物理地址。

4.5.2 地址变换

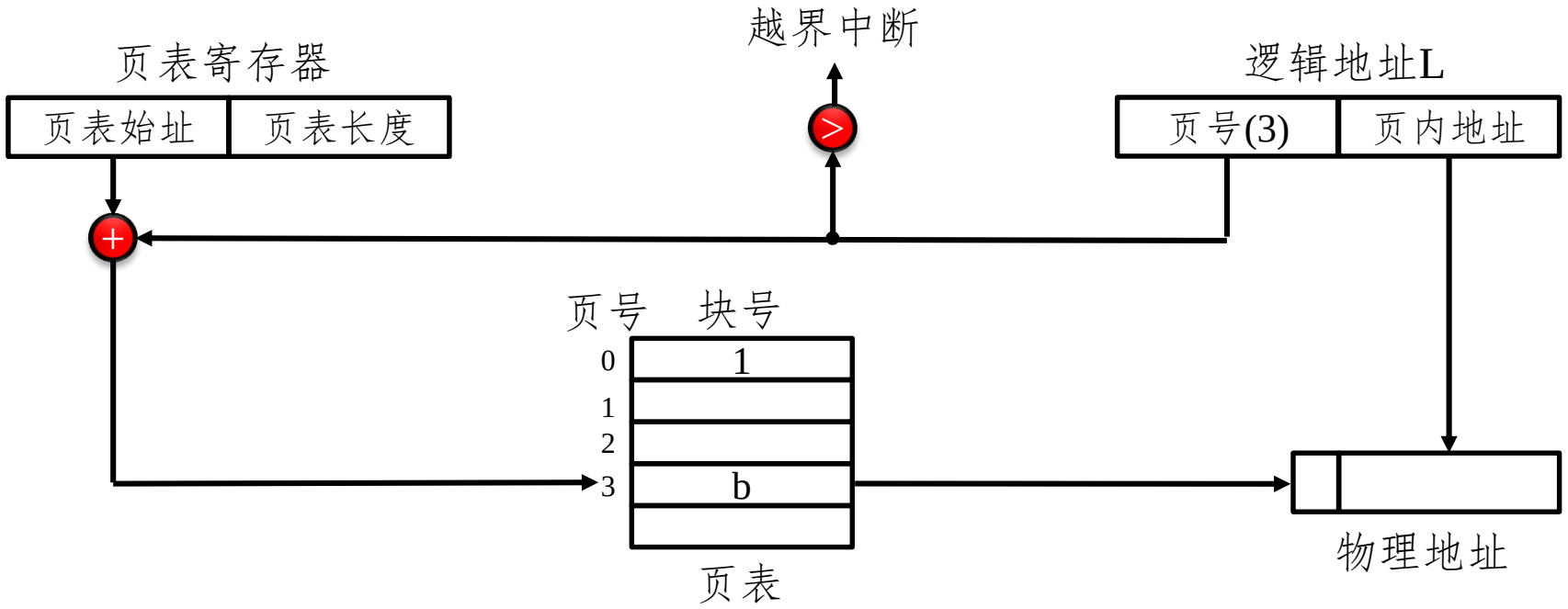


图4-15 分页系统的地址变换机构

4.5.2 地址变换

2、快表

- 分页系统：处理机每次存取指令或数据至少需要访问两次物理内存
  — 第一次访问页表，第二次存取指令或数据，（**得不偿失**）

- 为了提高地址变换速度，为进程页表设置一个专用的高速缓冲寄存器，称为快表、TLB(Translation Lookaside Buffer)，或联想存储器（Associative Memory）

快表的工作原理

- 快表的工作原理类似于系统中的数据高速缓存(cache)，其中*专门保存当前进程最近访问过的一组页表项*。
- 进程最近访问过的页面在不久的将来还可能被访问。(局部性原理）
- 由于成本的原因，快表一般不可能太大，通常仅能存放16-512个表项（命中率可达到90%）。

分页系统地址转换

- 通过根据逻辑地址中的页号，查找快表中是否存在对应的页表项。
- 若快表中存在该表项，称为命中（hit），取出其中的页框号，加上页内偏移量，计算出物理地址。
- 若快表中不存在该页表项，称为命中失败，则再查找页表，找到逻辑地址中指定页号对应的页框号。同时，更新快表，将该表项插入快表中。并计算物理地址。

具有快表的地址变换机构

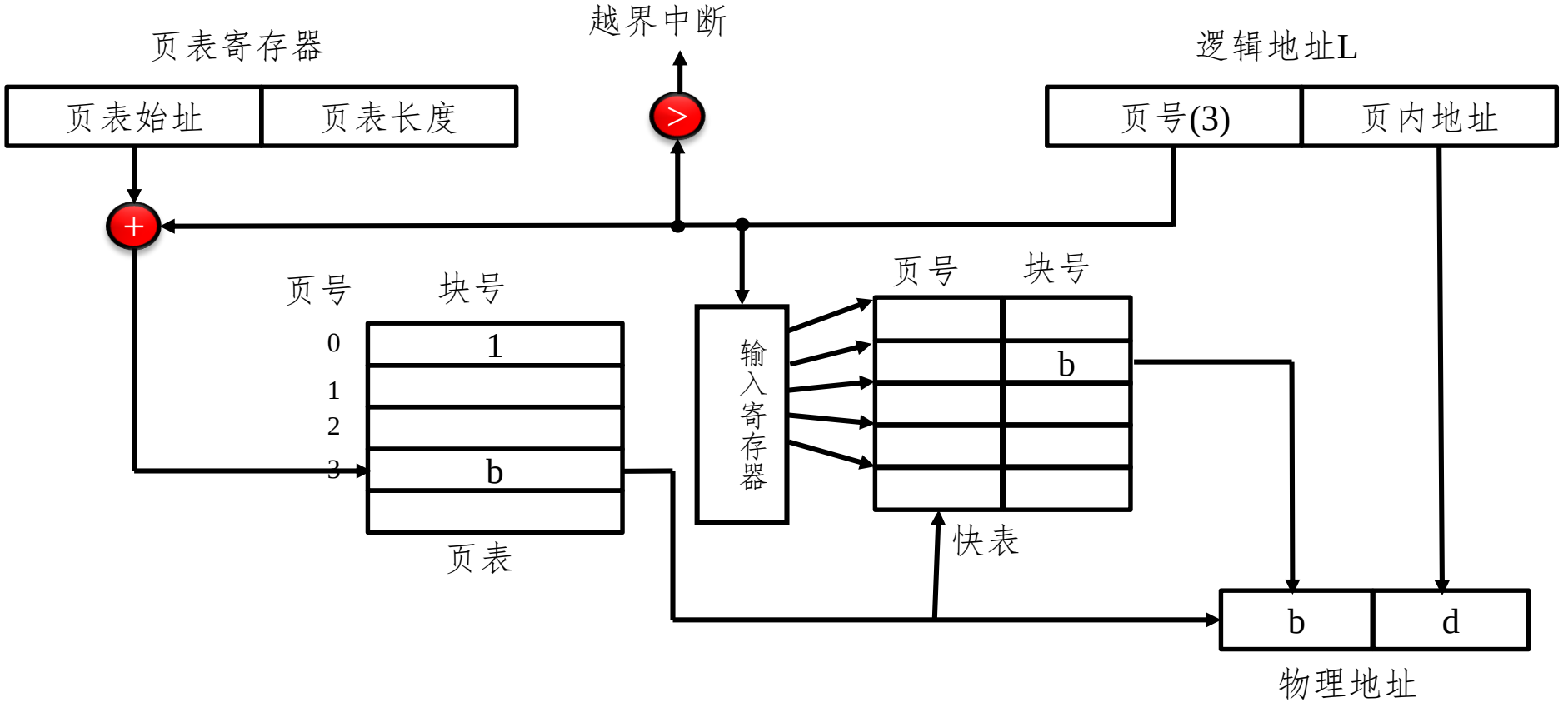
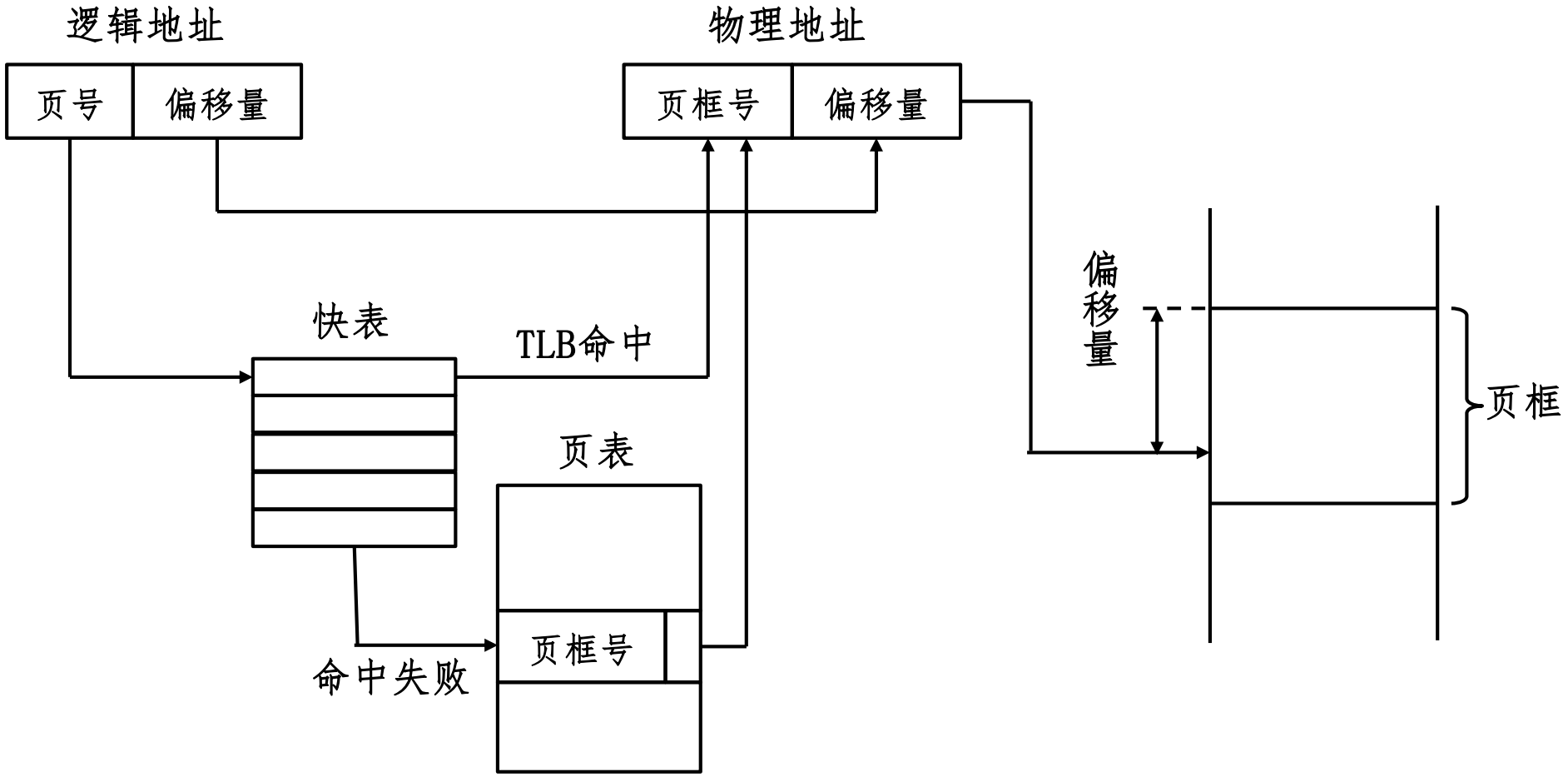


图4-16 具有快表的地址变换机构

具有快表的分页系统地址变换过程



具有快表的分页系统地址变换过程

4.5.4 两级和多级页表

- 大逻辑地址空间，页表非常大，需要占用相当大的内存空间。
- 比如，32位逻辑地址空间，假设页面大小为4KB（2^{12}），则4GB（2^{32}）的逻辑地址空间将被划分成2^{20}个页面。
- 若采用一级页表，则其内将包含1兆（2^{20}）个页表项。若按字节寻址，一个页表项占4B，则一级页表需要占用4MB（2^{22}）内存空间。不可能将4MB的页表保存在一个连续区中。
- *那么，如何处理大页表的存储与检索呢？*

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，（即引入两级页表）；

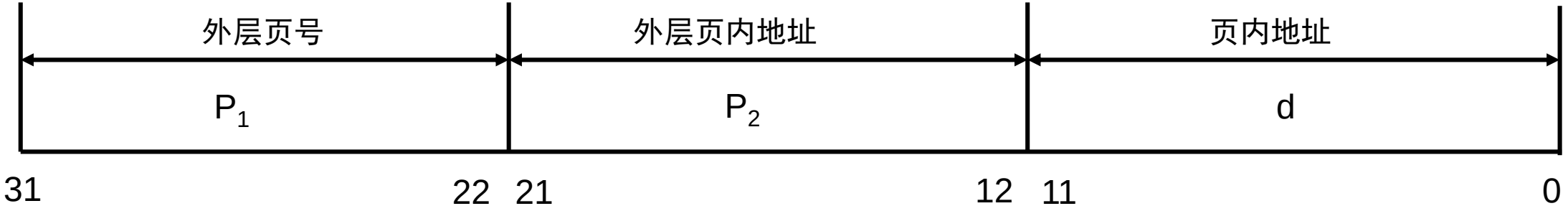
② 只将当前需要的部分页表项调入内存， 其余的页表项仍驻留在磁盘上，需要时再调入。

二级页表

- 将一个大页表全部保存在内存中。
- 首先，将其分割，并离散地存储在内存的多个页框中。
- 为之建立二级页表，记录被分割的各个页表页面存储在哪些页框中，也称为外层页表（Outer Page Table）。
- 对于4GB的进程，若采用二级页表，则对应的二级页表结构如图：

1. 两级页表(Two-Level Page Table)

逻辑地址结构可描述如下：



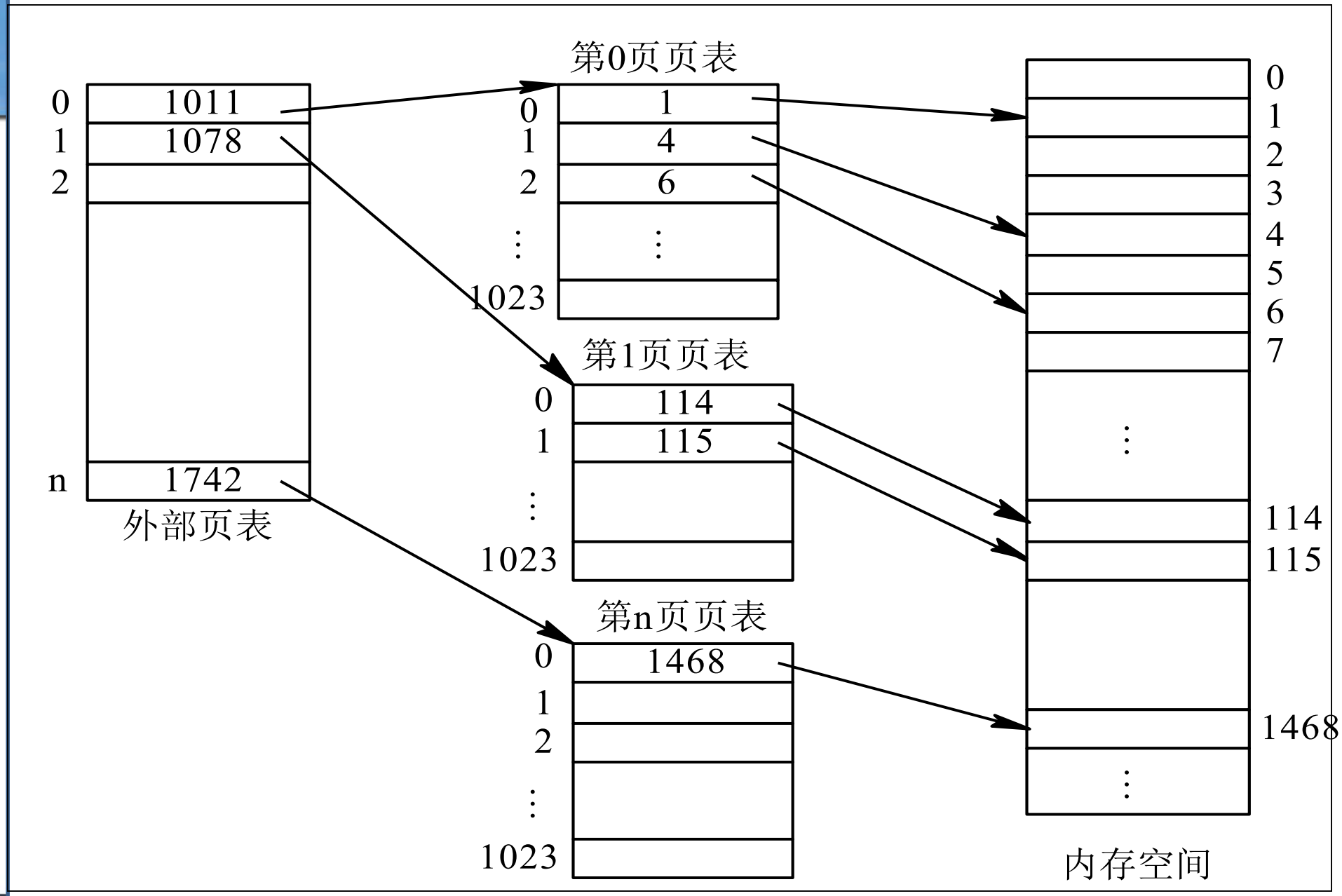
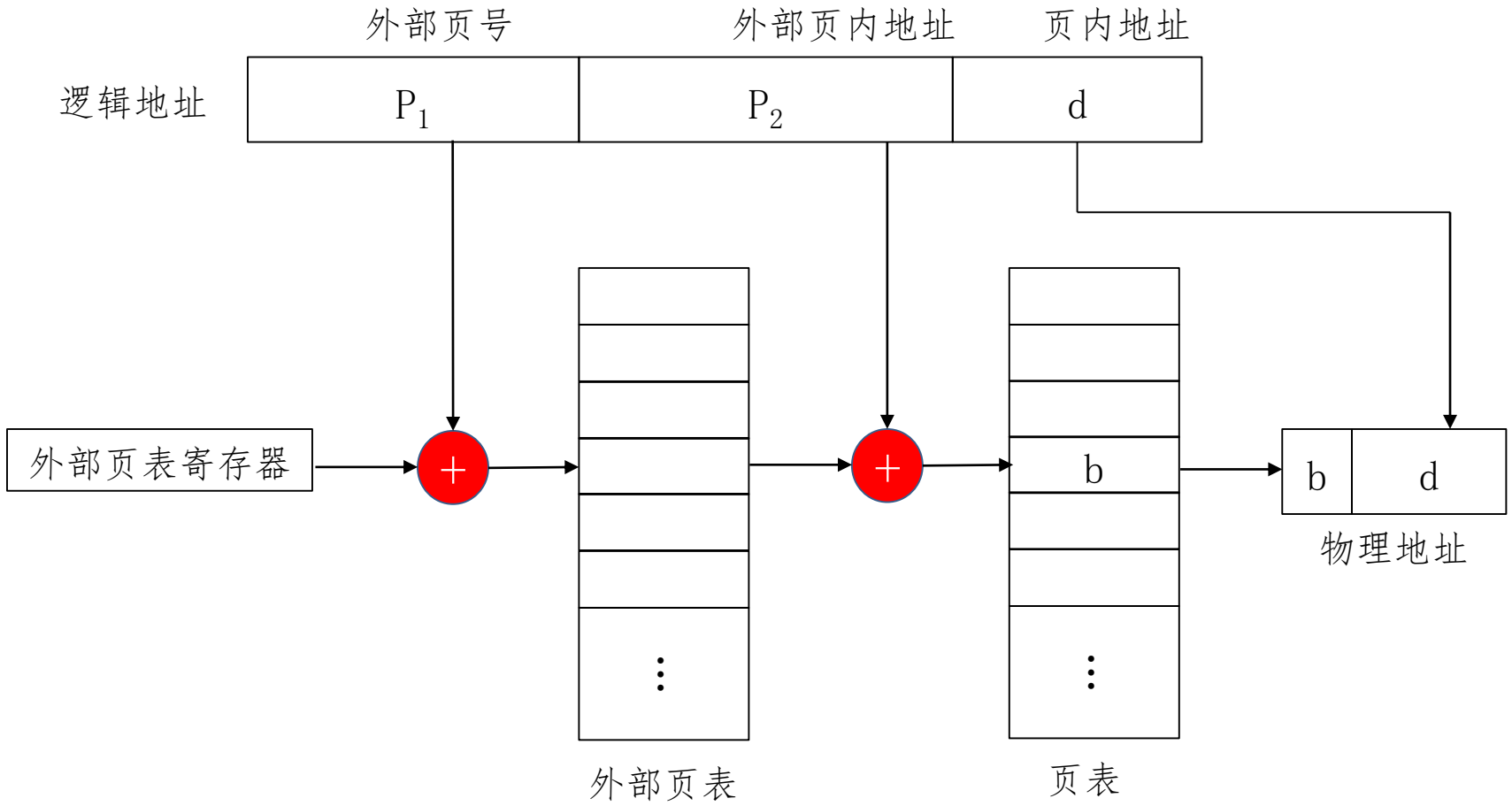

第0页页表
第1页页表
第n页页表
外部页表
内存空间
0
1011
1078
n
1742
0
1
1023
1
4
6
1023
0
114
1
115
1023
0
1468
1023
0
1
2
3
4
5
6
7
114
115
1468

图4-18 两级页表结构

1. 两级页表(Two-Level Page Table)

- 地址变换机构中*增设外层页表寄存器*，用于存放外层页表的始址。
- 利用逻辑地址中的外层页号，作为外层页表的索引，从中找到指定页表分页的始址，再利用指定页表分页的索引，找到指定的页表项，即该页在内存的物理块号。
- 用该块号和页内地址即可构成访问的内存物理地址。

具有两级页表的地址变换机构



（2）多级页表

- 对于64位的机器，采用两级页表仍然有困难，必须采用多级页表。
- 将外层页表再进行分页，也就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，来实现分页存储管理。

4.5.5 反置页表

- 反置页表的引入
- 反置页表中的地址转换

- 这一部分请自学

对分页存储管理的评价

- 彻底消除了外零头,仅存在很少的内零头, 提高了内存利用率。
- 分页操作由系统自动进行,一个页面不能实现某种逻辑功能。用户看到的逻辑地址是一维的，无法调试执行其中的某个子程序或子函数。
- 采用分页技术不易于实现存储共享，也不便于程序的动态链接。

4.5 基本分段存储管理方式

- 1.引入分段的原因

（1）方便编程

根据编程人员的需要将程序分成代码段、数据段等独立信息段。

（2）信息共享

根据编程人员的需要对这些段进行更有效的信息共享和保护。

（3）信息保护

分段管理方式能更有效和方便地实现信息保护功能。

（4）动态增长

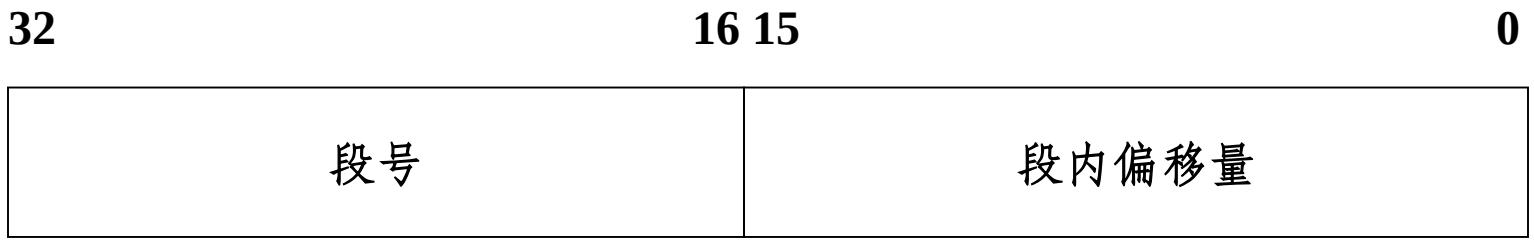
分段存储管理方式能较好地解决数据段增长 。

（5）动态链接

程序运行时，先将主程序所对应的目标程序装入内存并启动运行，当运行过程中又需要调用某段时，才将该段调入内存并进行链接。

4.5.2 分段存储基本原理

- 程序由若干逻辑段组成，每个段有自己的名字和长度。程序的逻辑地址是由段名（段号）和段内偏移量决定。每个段的逻辑地址从0开始编址.



分段管理的逻辑地址表示

- 系统采用动态分区技术，将物理内存动态地划分成许多尺寸不相等的分区。

- 当一个进程被装入物理内存时，*系统将为该进程的每一段独立地分配一个分区。*同一进程的多个段不必存放在连续的多个分区中。

分段系统的基本数据结构

- 段表 ： 每个进程建立一个段表，用于描述进程的分段情况，记载进程的各个段到物理内存中分区的映射情况。其中包含段号、段基址、段长以及对本段的存取控制权限等信息。

- 空闲分区表 ： 用于记载物理内存中的空闲分区情况。

段表

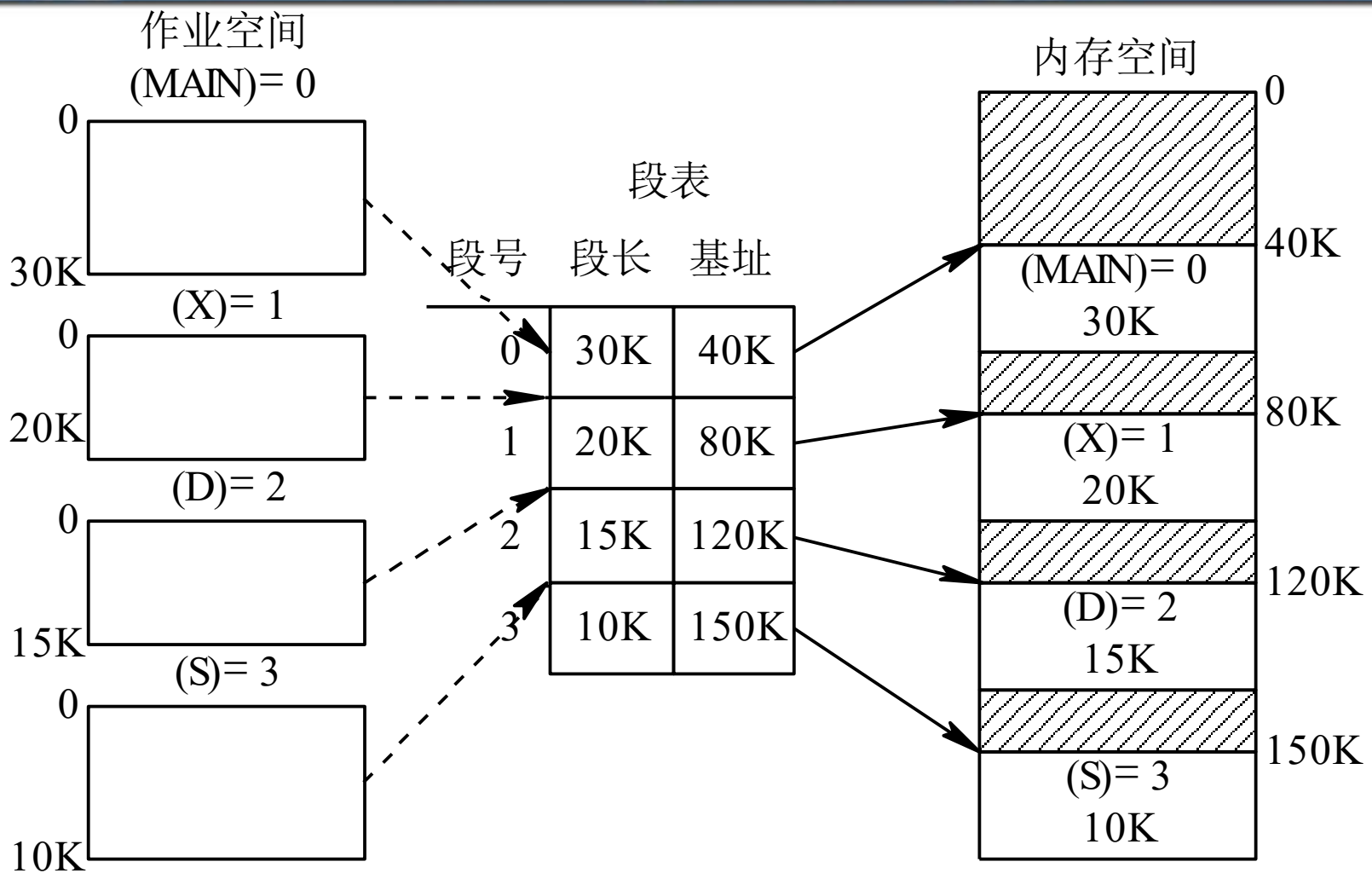


图 4-19 利用段表实现地址映射

段表寄存器

- 段表同样被保存在物理内存中。
- *段表寄存器：*实现快速地址变换，用来存放当前执行进程的段表在物理内存中的起始地址（即基址）。
- 当创建进程，将进程的程序和数据装入内存时，系统为之建立段表，并将段表的起始地址填入进程的PCB中。
- 当进程被调度执行时，取出其PCB中的段表首址，填入段表寄存器中。

地址变换和存储保护

（1）根据逻辑地址中的段号检索进程段表，获得指定段对应的段表项；

（2）判断是否地址越界。比较逻辑地址中的段内偏移量与段表项中的段长，若超过段的长度，则产生存储保护中断（该中断将由操作系统进行处理）；否则，转（3）；

（3）把逻辑地址中的段内偏移量与段表表项中的段基址相加，从而得到物理地址。

地址变换机构

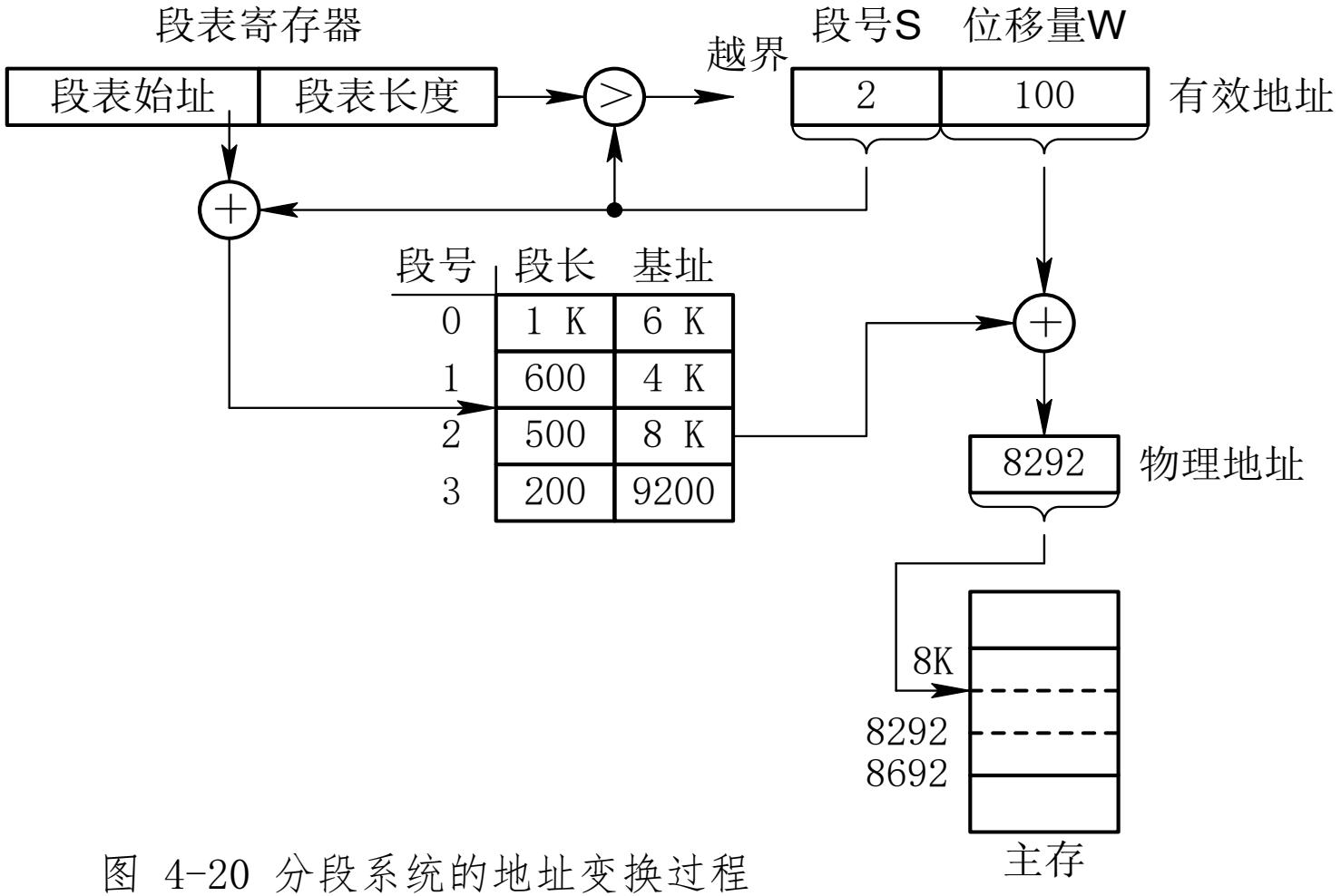


图 4-20 分段系统的地址变换过程

对分段系统的评价

- 有效消除了内零头，提高了存储利用率。
- 允许子程序独立编译和修改，而不需要重新编译或链接其它相关子程序。
- 容易实现存储共享。
- 具有较高的安全保障。
- 很容易满足程序段的动态增长需要 。
- 通常需要编译器的支持。

4. 分页和分段的比较（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。

4. 分页和分段的比较（主要区别）

（2）页的大小固定且由系统决定，因而在系统中只能有一种大小的页面，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，根据信息的性质来划分。

（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。

4.6.3 信息共享

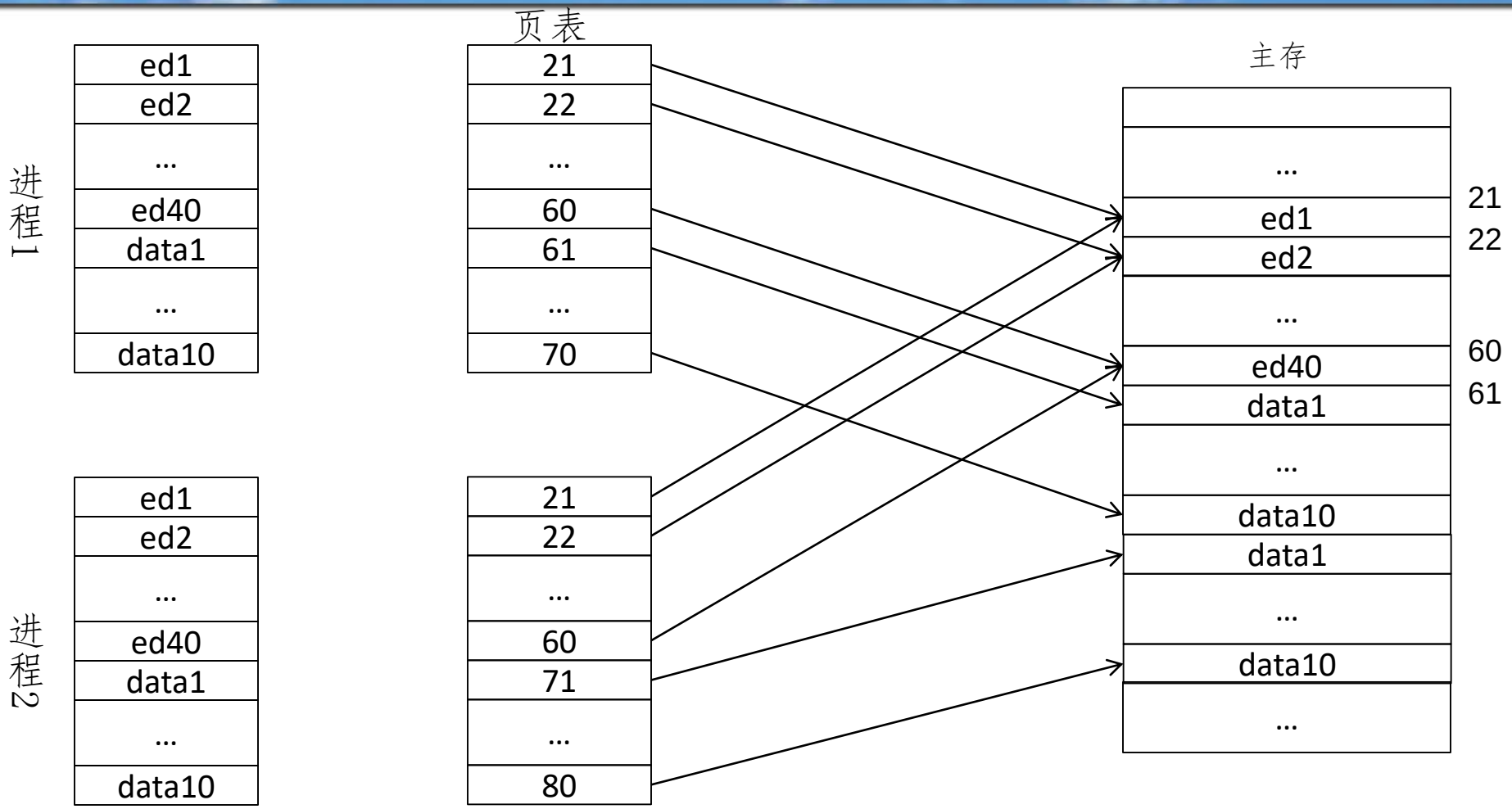
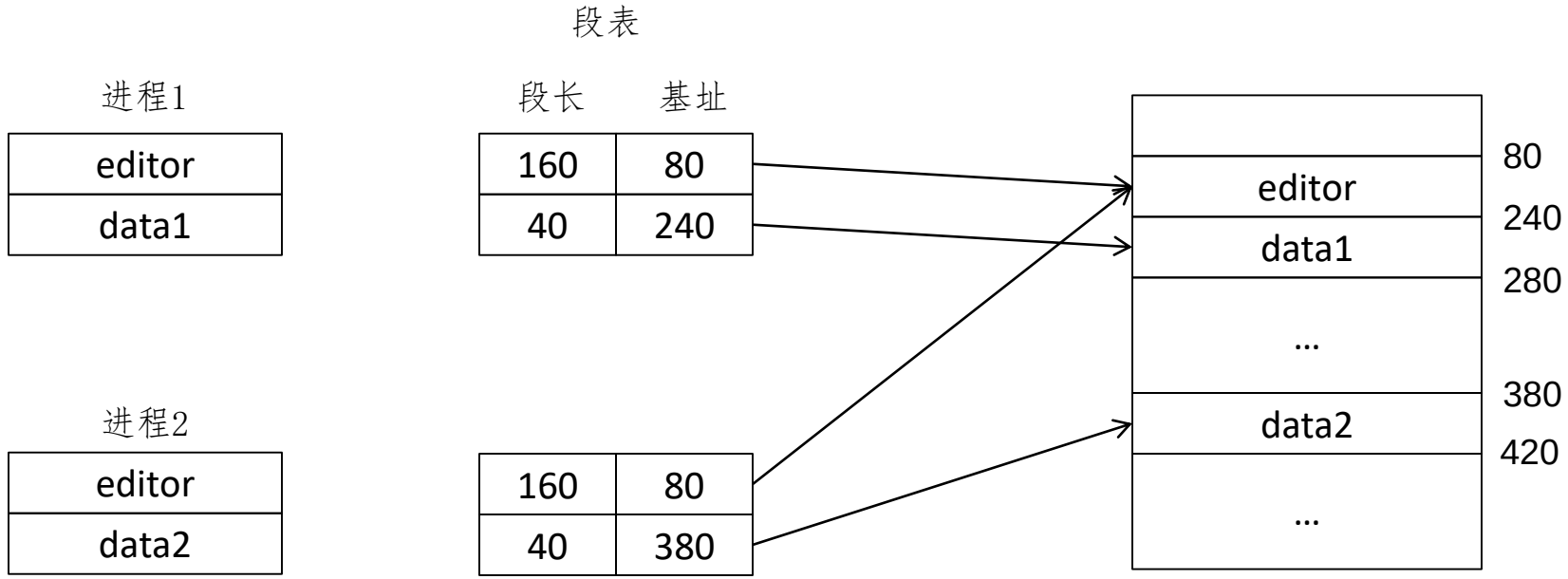


图4-21 分页系统中共享editor的示意图

4.6.3 信息共享


段表
进程1
段长
基址
editor
data1
160
80
40
240
进程2
editor
data2
160
80
40
380
editor
data1
...
data2
...
80
240
280
380
420

4.6.4 段页式存储管理方式

- 分页和分段存储管理方式都各有其优缺点。分页系统能有效地提高内存利用率，而分段系统则能很好地满足用户需要，将两者结合成一种新的存储管理方式系统，称为“段页式系统”。

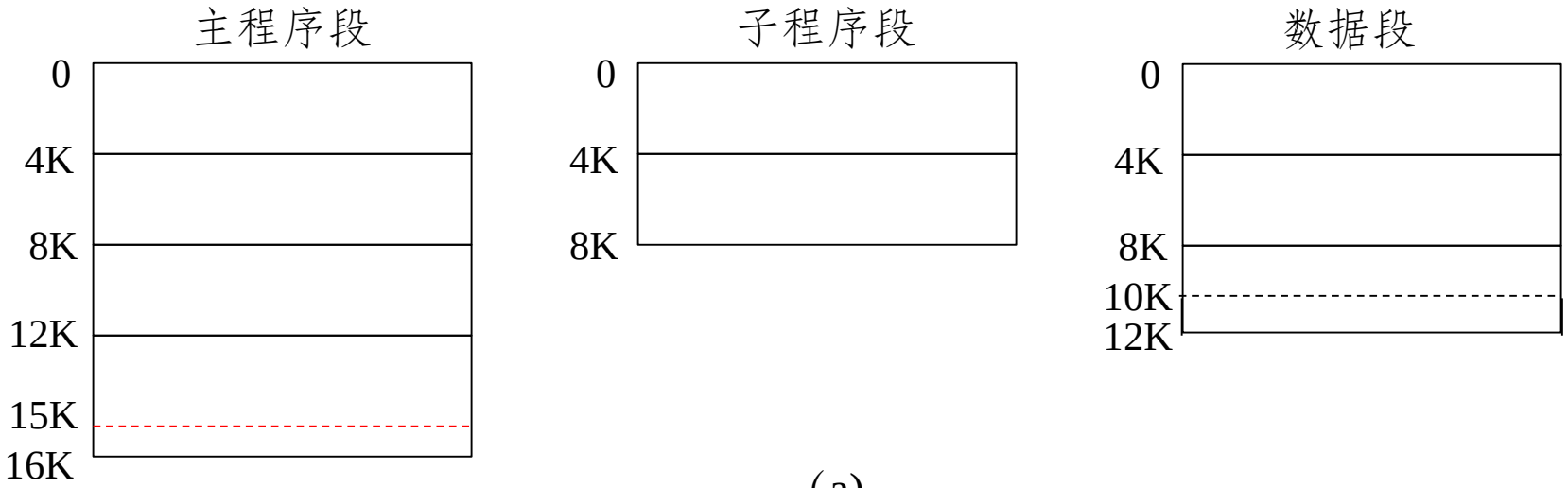
1．基本原理

段页式系统的基本原理：采用分段方法组织用户程序，采用分页方法分配和管理内存。

先将用户程序分成若干个段，再把每个段分成若干个页，并为每一个段赋予一个段名。

4.6.4 段页式存储管理方式

1. 基本原理



(a)

段号(S)	段内页号(P)	页内地址(W)

(b)

图4-23 作业地址空间和地址结构

4.6.4 段页式存储管理方式

- 用于管理的数据结构：段表、页表，如图

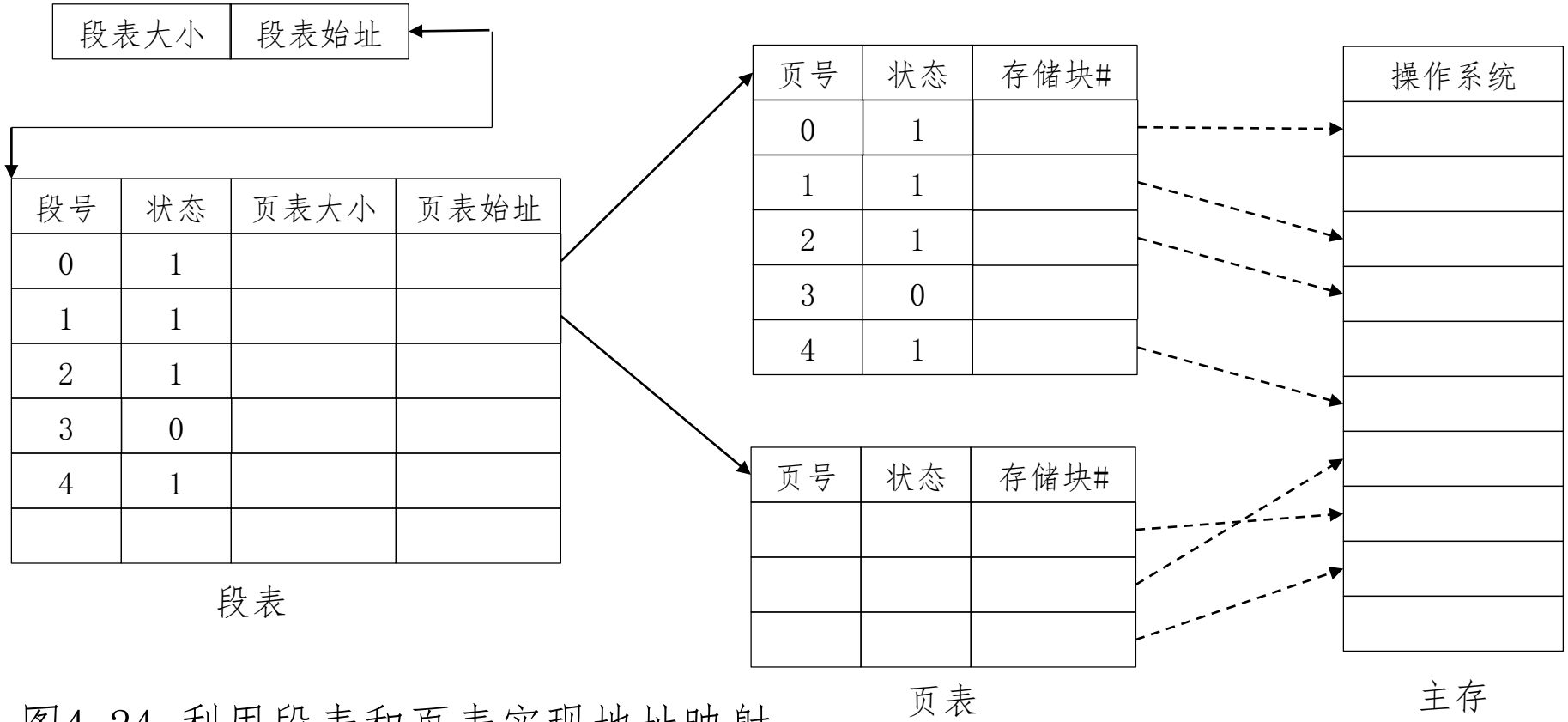


图4-24 利用段表和页表实现地址映射

4.6.4 段页式存储管理方式

2. 地址变换过程

① 首先，从段表寄存器获得进程*段表的起始地址*，根据该地址，查找进程的段表。

② 然后，根据逻辑地址指定的段号检索段表，找到对应段的*页表起始地址*。

③ 再根据逻辑地址中指定的页号检索该页表，找到对应*页所在的页框号*。

④ 最后，用页框号加上逻辑地址中指定的页内偏移量，形成物理地址。

4.6.4 段页式存储管理方式

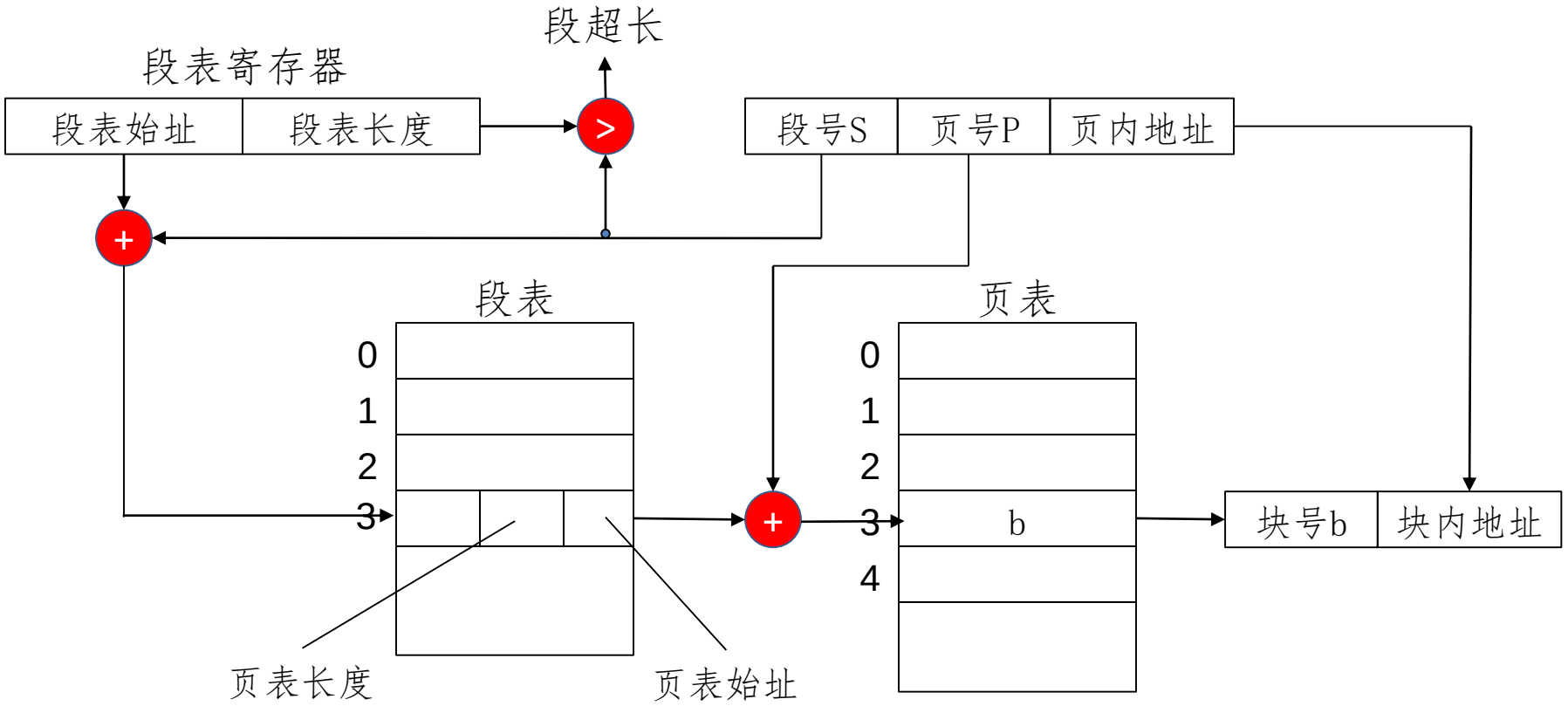


图4-25　段页式系统中的地址变换机构

第4章 作业

1. 在动态分区分配方式中，应如何将各空闲分区链接成空闲分区链
2. 分区存储管理中常用哪些分配策略？比较它们的优缺点
3. 基于离散分配时所用到的基本单位不同，可将离散分配分为哪几种
4. 什么是页表，页表的作用是什么
5. 在分页系统中如何实现地址变换的
6. 在具有快表的段页式存储管理方式中，如何实现地址变换
7. 试全面比较连续分配和离散分配方式