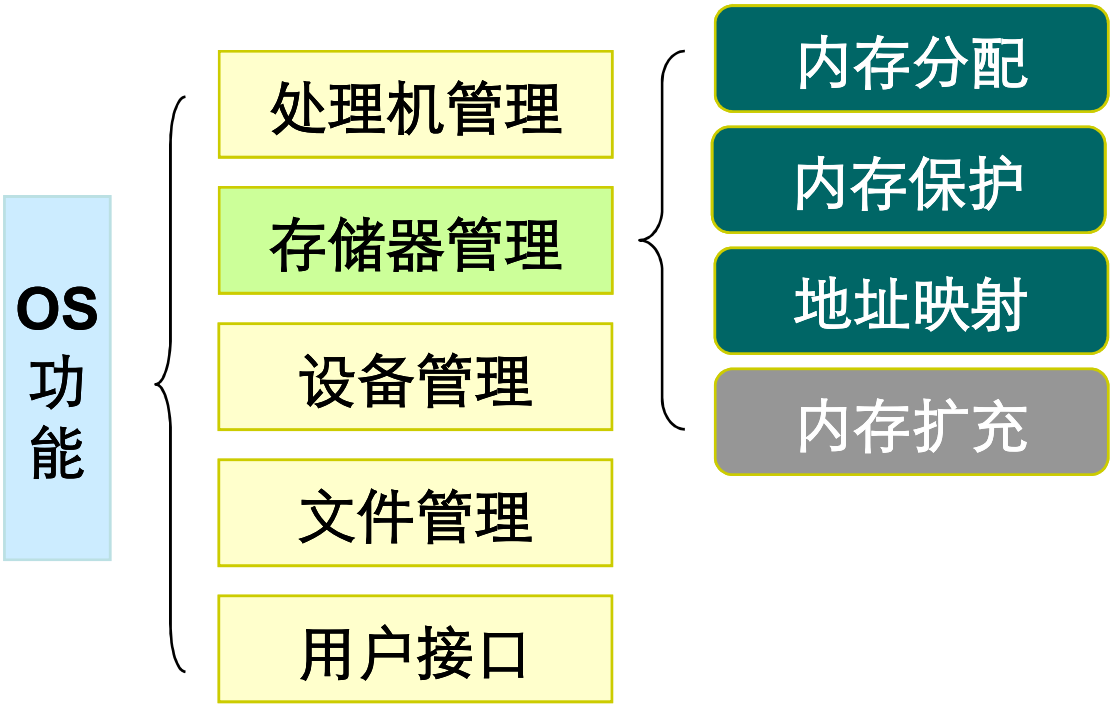
第4章 存储器管理 (5学时)

主讲教师：张春元
联系电话：13876004640
课程邮箱：haidaos@126.com
邮箱密码：zhangchunyuan

本章内容所处位置

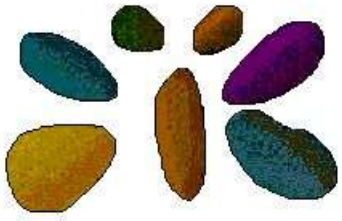


本章主要内容

- 4.1 存储器的层次结构
- 4.2 程序的装入和链接
- 4.3 连续分配存储管理方式
- 4.4 对换
- 4.5 基本分页存储管理方式
- 4.6 基本分段存储管理方式

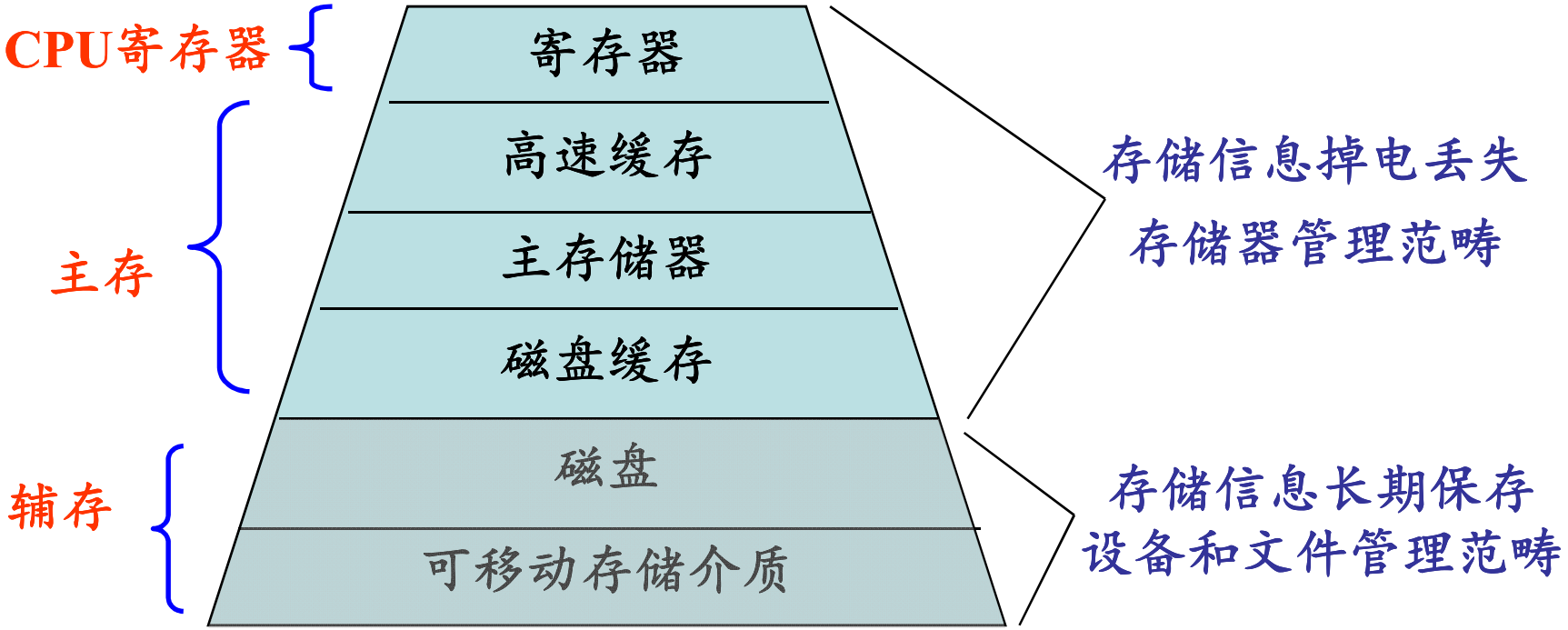
4.1 存储器的层次结构

- 4.1.1 多级存储器结构
- 4.1.2 主存储器和寄存器
- 4.1.3 高速缓存和磁盘缓存



4.1.1 多级存储器结构

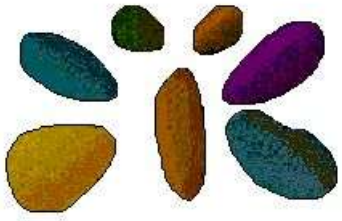
❖ 1、计算机系统存储层次结构



寄存器与主存储器也称可执行存储器，其与辅存的访问机制不一样，后者需要通过I/O设备才能访问。

4.1 存储器的层次结构

- 4.1.1 多级存储器结构
- 4.1.2 主存储器和寄存器
- 4.1.3 高速缓存和磁盘缓存

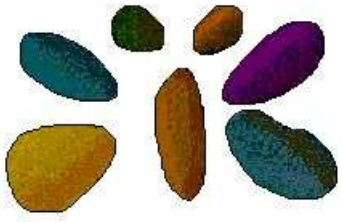


4.1.2 主存储器和寄存器

- 1、主存储器（简称内存、主存）
  - 主要作用：用于保存进程运行时的程序和数据，因而也称可执行存储器。
  - 工作机制：CPU中的控制器从主存中提取数据和程序指令，将它们装入寄存器；或者CPU将寄存器中的数据存入到主存中。
- 2、寄存器
  - 主要作用：用于加速主存的访问速度，与CPU具有相匹配的数据存取速度。
  - 工作机制：属CPU的重要组成部分，与CPU中的运算器和控制器等一起协调工作。

4.1 存储器的层次结构

- 4.1.1 多级存储器结构
- 4.1.2 主存储器和寄存器
- 4.1.3 高速缓存和磁盘缓存



4.1.3 高速缓存和磁盘缓存

❖ 1、高速缓存

* 主要作用：加速CPU对主存的访问、提高程序执行速度。
* 工作机制：根据程序执行的局部性原理，将主存中一些经常访问的信息置于高速缓存中，以减少CPU对主存的访问。

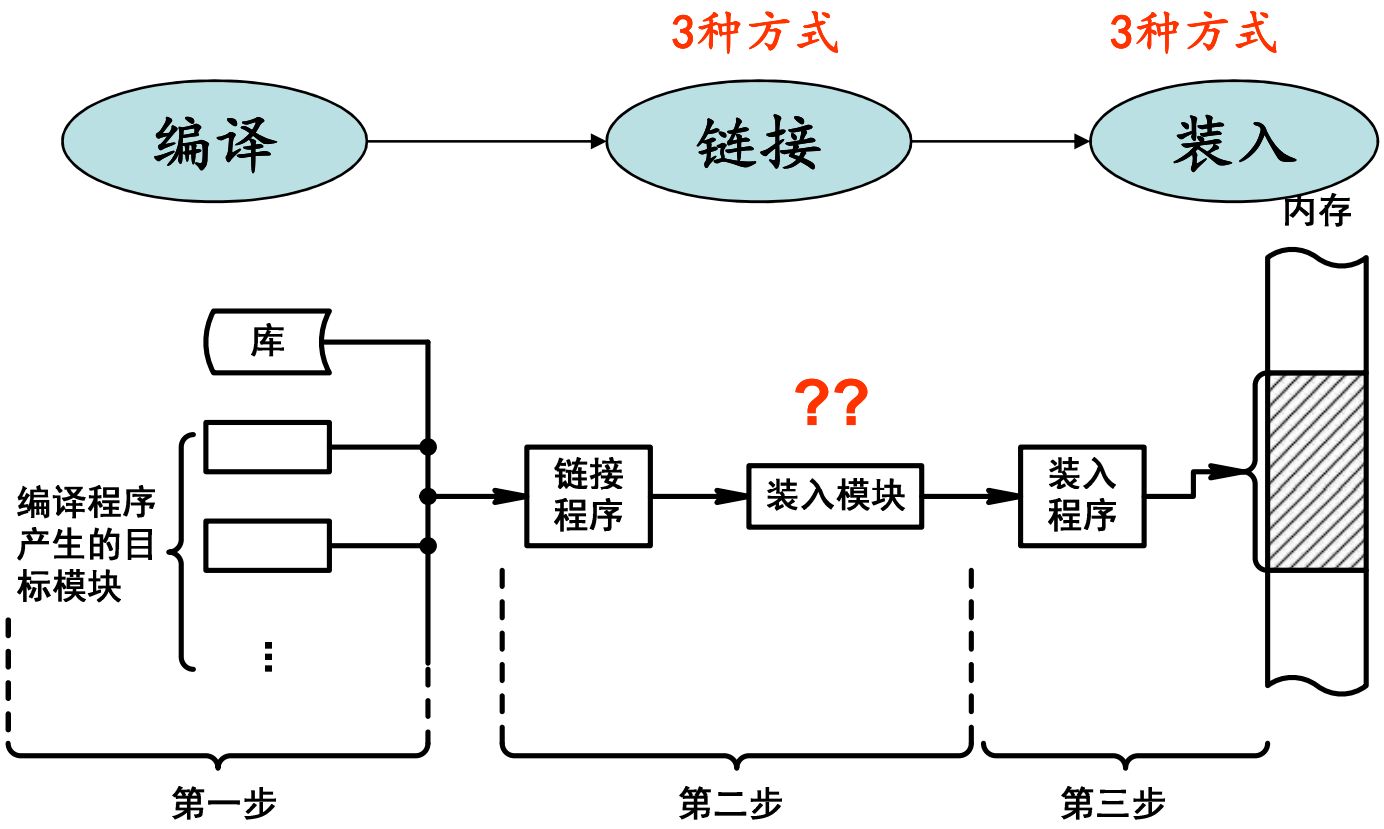
❖ 2、磁盘缓存

* 主要作用：加速CPU对磁盘的访问速度。
* 工作机制：磁盘缓存本身并不是一种实际存在的存储介质，其利用主存中的存储空间，来暂存需频繁从磁盘中读出（或写入）的信息，以减少对磁盘的访问次数。

本章主要内容

- 4.1 存储器的层次结构
- 4.2 程序的装入和链接
- 4.3 连续分配存储管理方式
- 4.4 对换
- 4.5 基本分页存储管理方式
- 4.6 基本分段存储管理方式

4.2 程序的装入和链接



- 4.2.1 程序的装入
- 4.2.2 程序的链接

4.2.1 程序的装入

❖ 1、绝对装入方式（Absolute Loading Mode）

* 基本原理

- 是指编译器在编译时就已知道所编译的目标代码将置于内存什么位置，从而直接将源程序编译成使用绝对地址（内存地址、物理地址）的目标代码，装入时便可按照装入模块中的地址直接将程序和数据装入内存。
- 绝对装入方式只适用于单道系统。

* 绝对地址产生的两种方式

- （1）由编译器完成
  - 编程时采用符号地址，经编译再转换成绝对地址
- （2）由程序员直接赋予

4.2.1 程序的装入

❖ 2、可重定位装入方式与静态重定位

* 重定位

- 重定位：程序在装入时或执行时将其中的指令和数据的相对地址转换成绝对地址的过程称为重定位。
- 1> 静态重定位：如果重定位是在程序装入时一次性完成，则称之为静态重定位。
- 2> 动态重定位：如果重定位是在程序装入后、真正执行时再进行，则称之为动态重定位。

4.2.1 程序的装入

* 可重定位（静态重定位）装入方式基本原理
  - 在多道环境下，编译器在编译时将源程序编译成使用相对地址（逻辑地址）的目标代码，装入时采用静态重定位方式将程序装入模块中的指令和数据的相对地址（逻辑地址）一次性调整成为相应内存单元的绝对地址，以后不再改变。
* 可重定位装入方式的特点
  - 可适用于多道系统。
  - 简单、但装入后不能在内存中再移动。

4.2.1 程序的装入

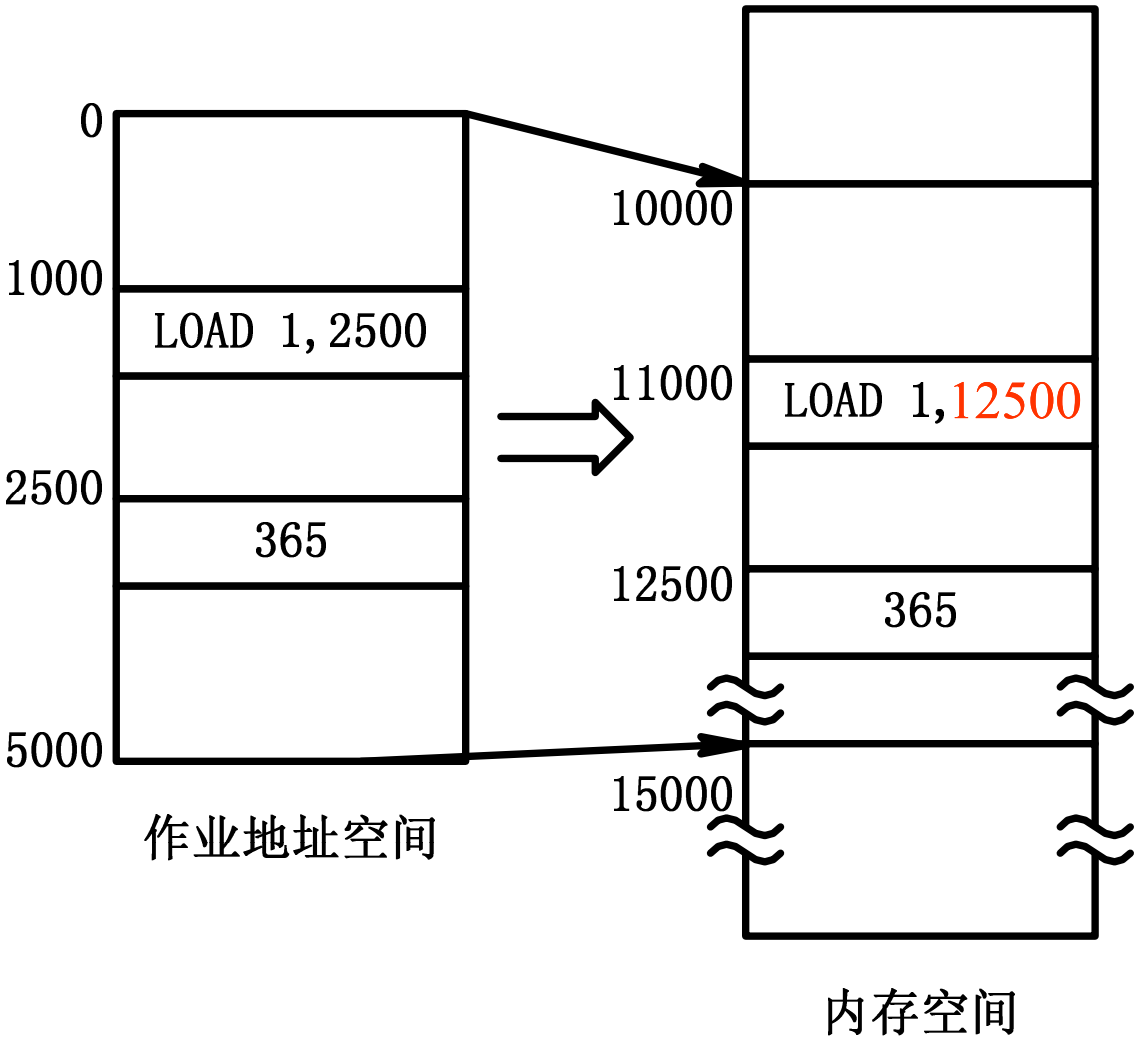


图 4-3 作业可重定位装入内存

4.2.1 程序的装入

❖ 3、动态运行时装入方式与动态重定位

* 基本原理

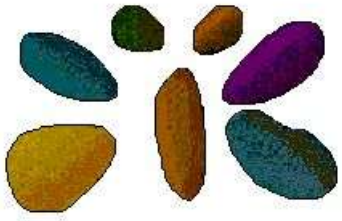
- 在多道环境下，编译器在编译时将源程序编译成使用相对地址（逻辑地址）的目标代码，装入内存后直到程序真正执行时才在重定位寄存器的支持下采用动态重定位方式完成相对地址到绝对地址的转换。

* 动态运行时装入的特点

- 可适用于多道系统。
- 支持程序在内存中移动。

4.2 程序的装入和链接

- 4.2.1 程序的装入
- 4.2.2 程序的链接



4.2.2 程序的链接

❖ 1、静态链接（Static Linking）

* 基本原理

- 在程序装入内存前，先将编译所得各目标模块及所需的库函数链接成一个完整的装入模块，以后不再拆开。

* 静态链接过程中须解决的两个问题

- （1）对相对地址进行修改
- （2）变换外部调用符号

4.2.2 程序的链接

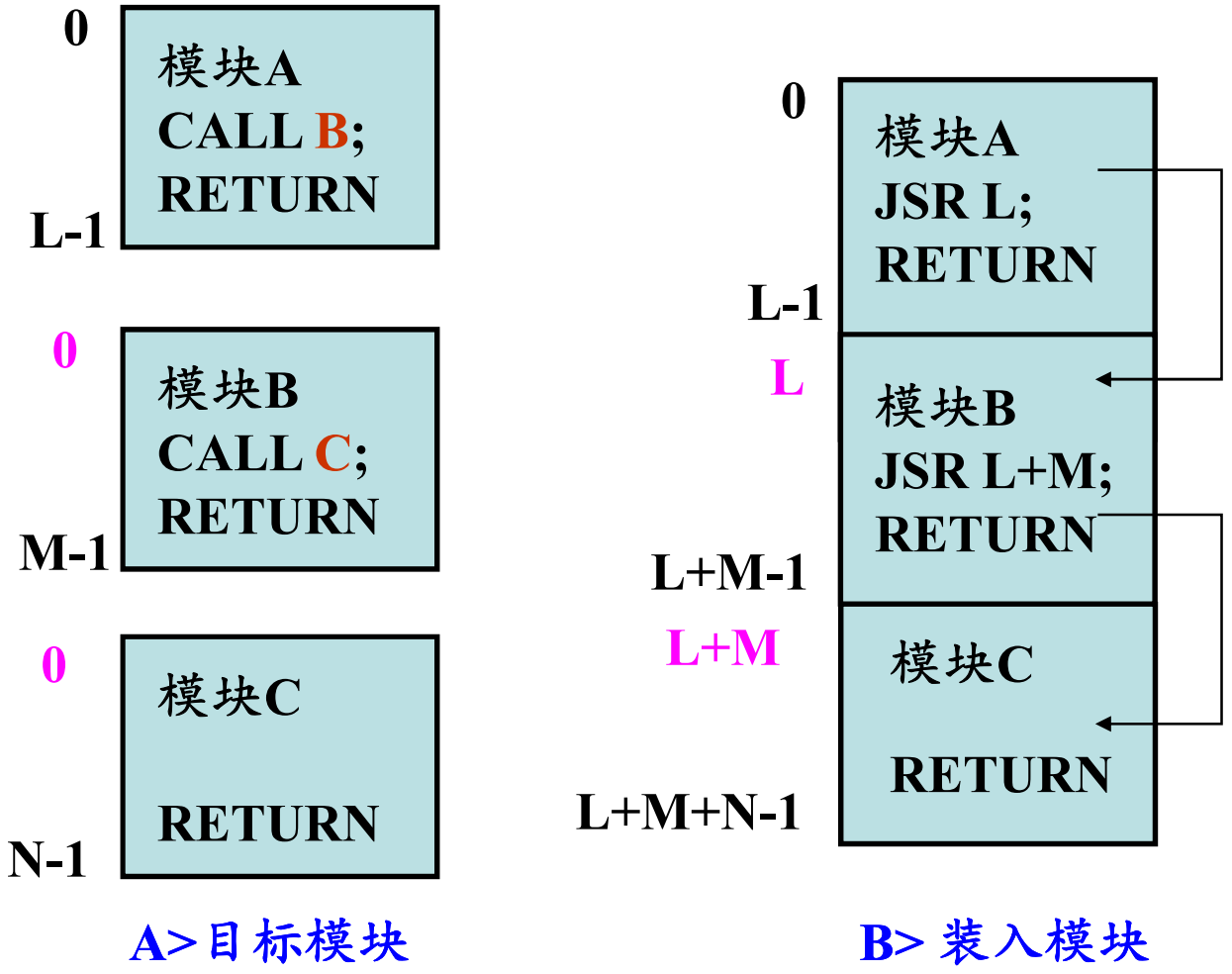


图 4-4 程序链接示意图

4.2.2 程序的链接

❖ 2、装入时动态链接（Load-time Dynamic Linking）

* 基本原理

- 源程序编译所得各模块在装入内存时(各目标模块在内存中是分开存放的)，边装入边链接。
- 在链接过程中同样需要对相对地址进行修改，并变换外部调用符号。

* 装入时动态链接的优点

- （1）便于修改和更新
- （2）便于实现对目标模块的共享

4.2.2 程序的链接

❖ 3、运行时动态链接（Run-time Dynamic Linking）

* 基本原理

- 运行时动态链接采用动态运行时装入方式，将对某些模块的链接推迟到程序执行时才去做，即在执行过程中，当发现一个被调用模块尚未装入内存时，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。

* 运行时动态链接的优点

- 是对装入时动态链接的一种改进，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调用者模块上，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。

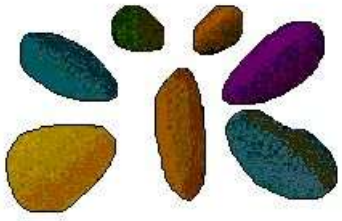
本章主要内容

- 4.1 存储器的层次结构
- 4.2 程序的装入和链接
- 4.3 连续分配存储管理方式
- 4.4 对换
- 4.5 基本分页存储管理方式
- 4.6 基本分段存储管理方式

连续分配方式是指为每个用户程序分配一个连续的内存空间。

4.3 连续分配方式

- 4.3.1 单一连续分配
- 4.3.2 固定分区分配
- 4.3.3 动态分区分配
- 4.3.4 基于顺序搜索的动态分区分配算法
- 4.3.5 基于索引搜索的动态分区分配算法
- 4.3.6 动态可重定位分区分配



4.3.1 单一连续分配

基本原理

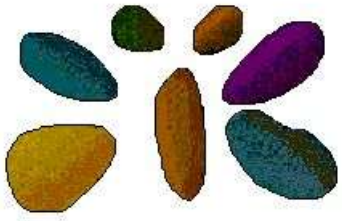
- 单一连续分配也称单用户存储管理，是最简单的一种存储管理方式，只能用于单用户单任务操作系统中。
- 单一连续分配将内存分为系统区和用户区两部分，系统区仅提供给OS使用，通常是放在内存的低址部分；用户区是指除系统区以外的全部内存空间，供用户程序使用，一次只能装入一个作业。
- 一般情况下无存储器保护机构（早期有）。

优缺点

- （1）简单易行，系统开销小
- （2）资源利用率低

4.3 连续分配方式

- 4.3.1 单一连续分配
- 4.3.2 固定分区分配
- 4.3.3 动态分区分配
- 4.3.4 基于顺序搜索的动态分区分配算法
- 4.3.5 基于索引搜索的动态分区分配算法
- 4.3.6 动态可重定位分区分配



4.3.2 固定分区分配

❖ 基本原理

* 固定分区分配是最早、最简单的多道程序存储管理方式。
* 固定分区分配将内存用户空间划分为若干个固定大小的区域，每个分区中只装入一道作业。

❖ 分区的划分方法

* 1> 分区大小相等 — 内存划分为多个大小相等的分区
  - 分区太大：浪费
  - 分区太小：有些程序不够用
* 2> 分区大小不等 — 内存划分为多个大、中、小分区
  - 根据程序大小来分配所合适的分区

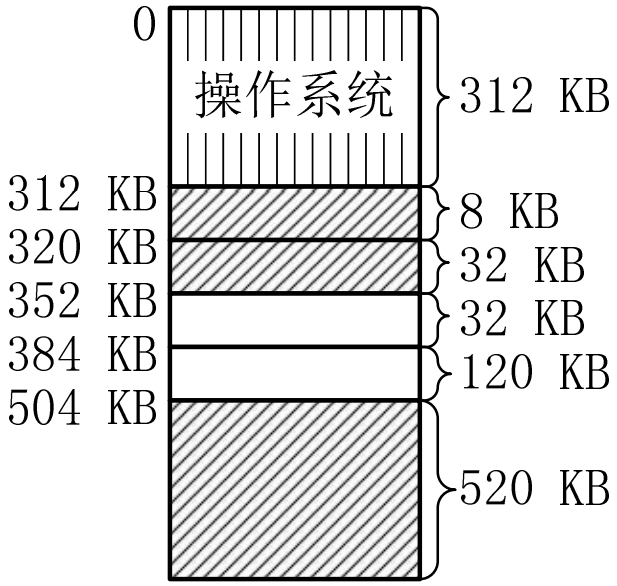
4.3.2 固定分区分配

❖ 内存分配

* 将分区按大小排序，建立分区使用表管理分配。

分区号	大小	始址	状态
1	8 KB	312 KB	在使用
2	32 KB	320 KB	在使用
3	32 KB	352 KB	未用
4	120 KB	384 KB	未用
5	520 KB	504 KB	在使用

(a)



(b)

图4-5 固定分区分配

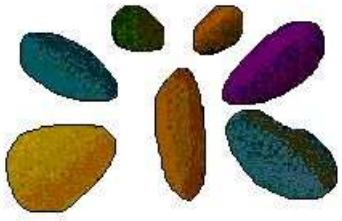
4.3.2 固定分区分配

❖ 管理特点

* 1> 一个作业只能装入一个分区，不能装入到两个或多个相邻的分区；一个分区只能装入一个作业，当分区大小不能满足作业的要求时，该作业暂时不能装入。
* 2> 通过对“分区使用表”的改写，来实现主存的分配与回收；作业在执行时，不会改变存储区域，所以采用静态地址重定位方式。此方法易于实现，系统开销小。
* 3>当分区较大作业较小时，仍然浪费许多主存空间；并且分区总数固定，限制了作业的并发数目。

4.3 连续分配方式

- 4.3.1 单一连续分配
- 4.3.2 固定分区分配
- 4.3.3 动态分区分配
- 4.3.4 基于顺序搜索的动态分区分配算法
- 4.3.5 基于索引搜索的动态分区分配算法
- 4.3.6 动态可重定位分区分配



4.3.3 动态分区分配

❖ 基本原理

* 根据进程的实际需要，动态地为之分配内存空间。

（a）可变式分区运行开始　（b）作业 1.2.3.4 进入内存　（c）作业 1.3 释放后内存

动态分区分配内存使用情况示意图

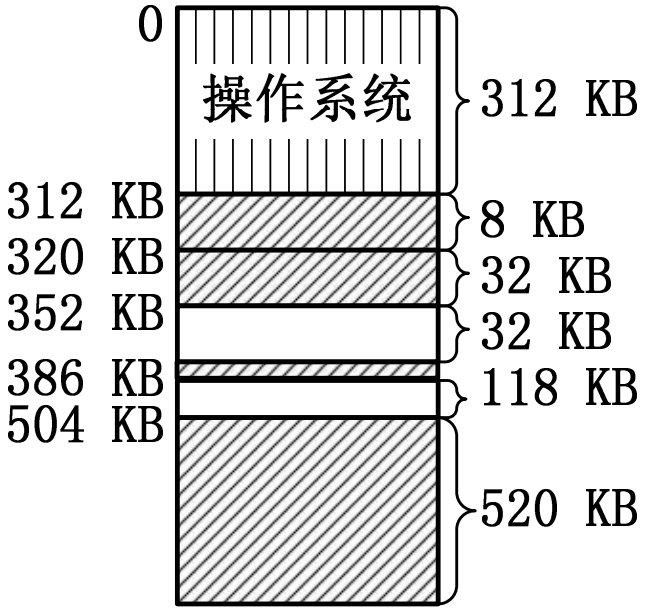
4.3.3 动态分区分配

❖ 1、动态分区分配中的两种数据结构

* 1> 空闲分区表：记录每个空闲分区的情况

序号	大小	始址	状态
1	32 KB	352 KB	空闲可用
2	118 KB	386 KB	空闲可用
3	…	…	…
4			
5			

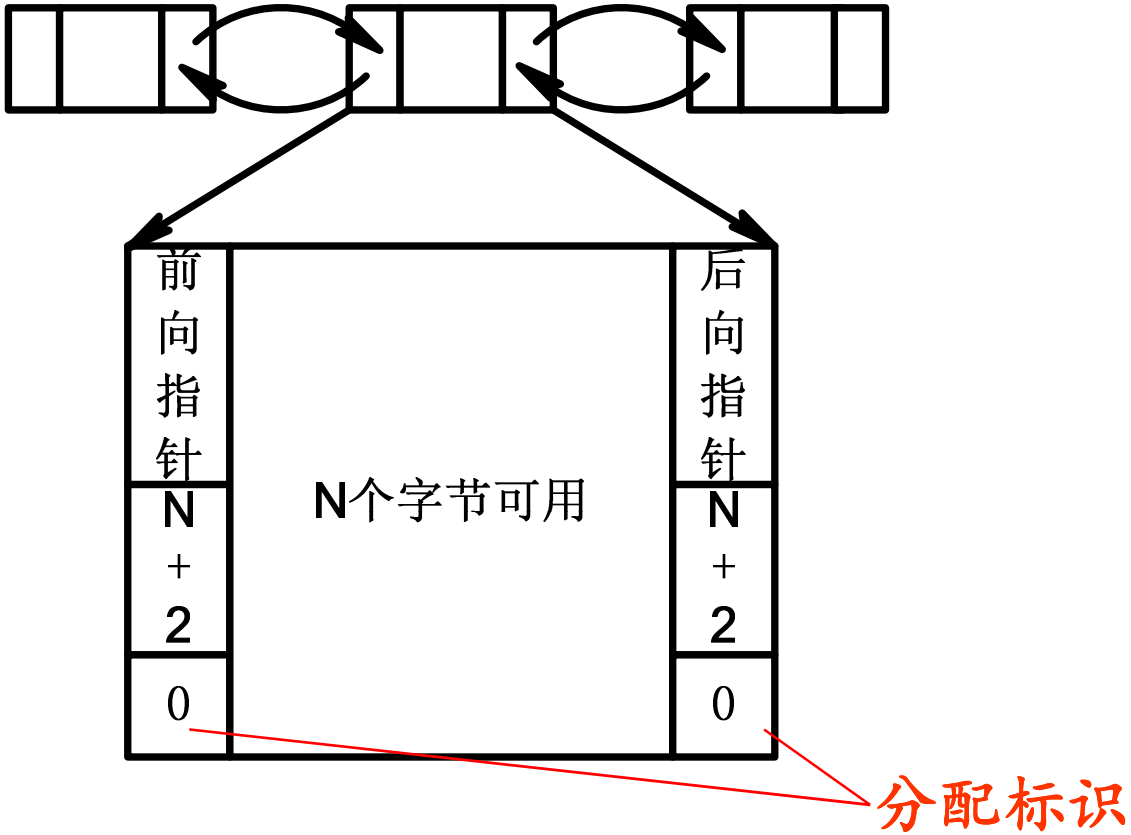
(a)



(b)

4.3.3 动态分区分配

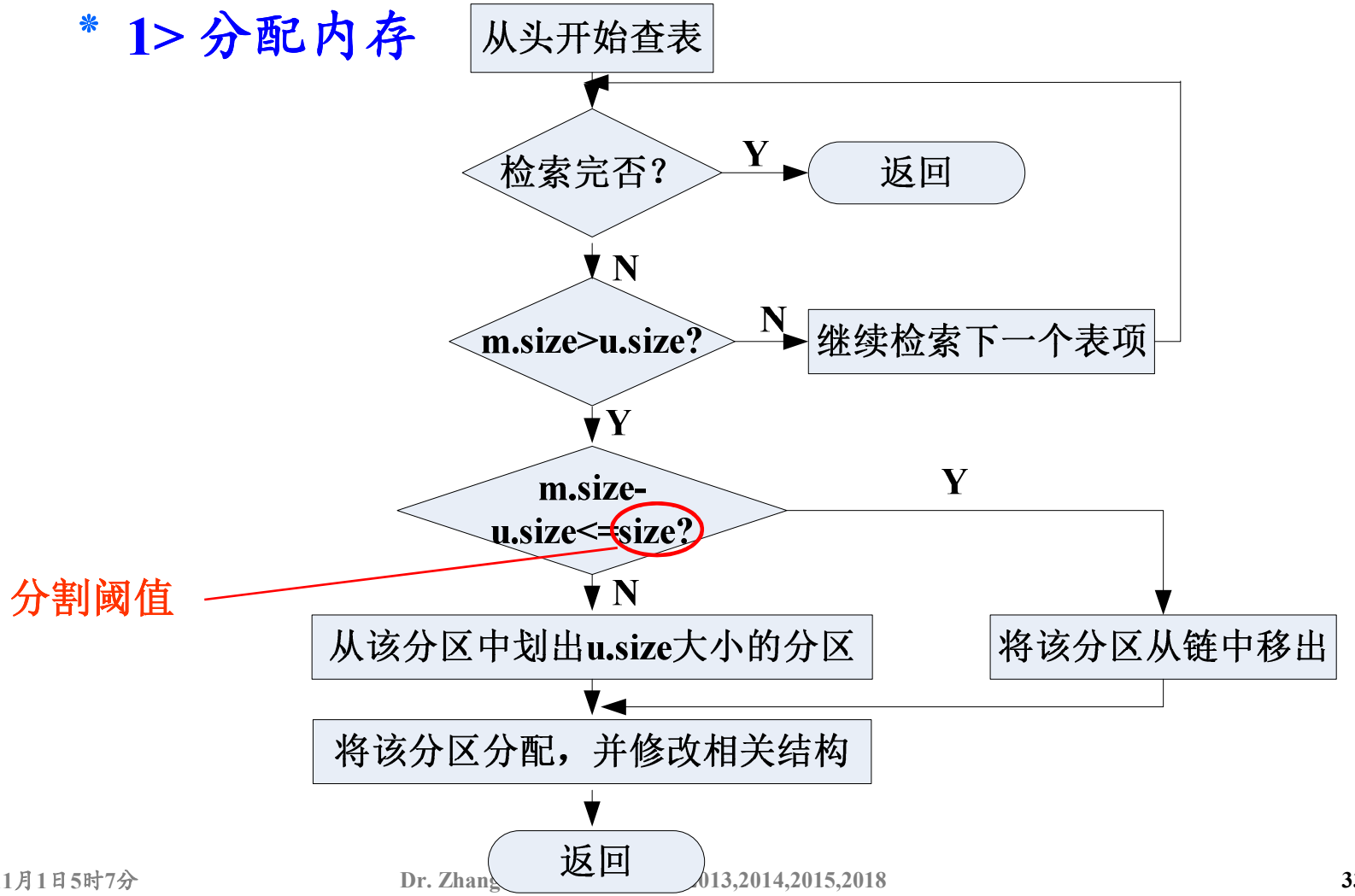
* 2> 空闲分区链：实现对空闲分区的链接和分配



4.3.3 动态分区分配

❖ 2、动态分区分配操作

* 1> 分配内存

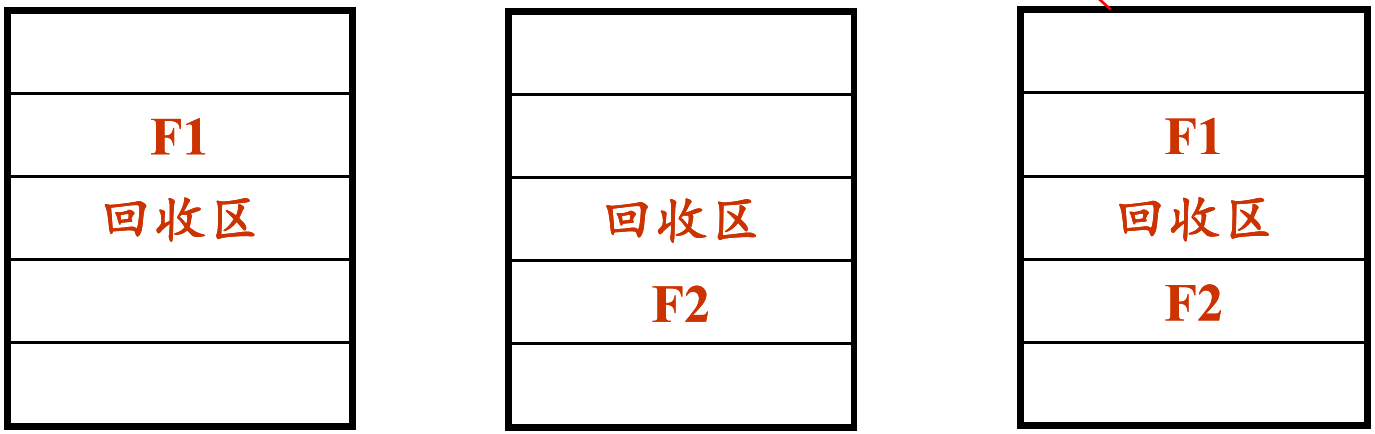


4.3.3 动态分区分配

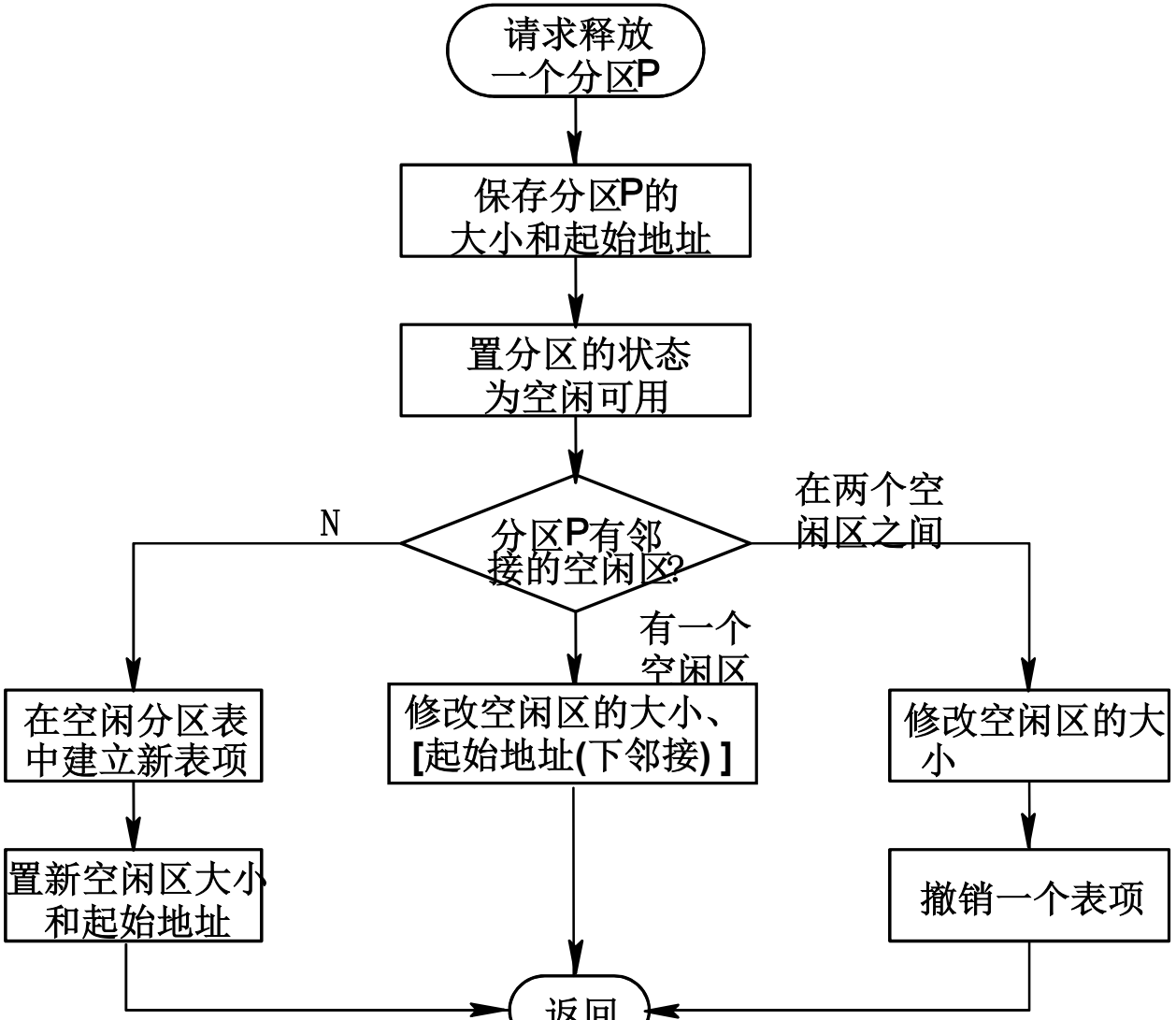
* 2> 回收内存

进程运行完释放所占用的分区时，系统将根据待回收分区的首地址将其插入到空闲分区链中：

- 1）上邻空闲区：合并，改大小
- 2）下邻空闲区：合并，改大小、首址。
- 3）上、下邻空闲区：合并，改大小。
- 4）不邻接，则建立一新空闲分区链表项。



动态分区内存回收流程图



4.3.3 动态分区分配

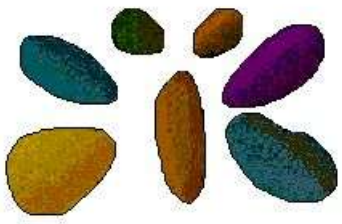
❖ 3、动态分区分配管理特点

* 1> 分区的长度不是预先固定的，而是按作业的实际需求来划分的；分区的个数也不是预先确定的，而是由装入的作业数决定的。
* 2> 分区的大小由作业的大小来定，提高了主存的使用效率。
* 3> 在主存分配过程中，会产生许多“外碎片”。“外碎片”是指因太小而无法使用的主存空间，从而使主存空间仍有一定的浪费。

4.3 连续分配方式

- 4.3.1 单一连续分配
- 4.3.2 固定分区分配
- 4.3.3 动态分区分配
- 4.3.4 基于顺序搜索的动态分区分配算法
- 4.3.5 基于索引搜索的动态分区分配算法
- 4.3.6 动态可重定位分区分配

按某种顺序组织、搜索空闲分区链，直到找到一个合适的分区



4.3.4 基于顺序搜索的动态分区分配算法

❖ 1、 首次适应算法（First Fit，FF）

* 原理：空闲分区按地址递增次序进行链接；每次分配时均从链首开始查找，直到找到一个大小可满足的空闲分区，从中划出一块给请求者。
* 优点：1> 分配算法简单；2> 优先利用低址部分，保留了高址的大空闲区，为大作业装入提供条件。
* 缺点：每次从链首开始，增加了查找开销，并且留下许多难以利用的”碎片”（外碎片）。

4.3.4 基于顺序搜索的动态分区分配算法

❖ 2、循环首次适应算法（Next Fit，NF）

* 原理：由首次适应算法演变而成的；空闲分区按地址递增次序链接，链表为循环链表；每次分配时从上次找到空闲分区的下一个空闲区开始查找，直到找到一个大小可满足的空闲分区，从中划出一块给请求者。
* 优点：使空闲分区分布均匀，减少查找空闲分区开销。
* 缺点：会缺乏大的空闲分区。

4.3.4 基于顺序搜索的动态分区分配算法

❖ 3、 最佳适应算法（Best Fit，BF）

* 原理：每次分配时，把能满足要求、又是最小的分区分配给进程；空闲分区按容量大小递增次序链接，查找时从链首开始，在第一次找到满足要求的空闲分区中划出一块给进程。
* 优点：解决了大作业的分配问题。
* 缺点：每次分配后所余部分总是最小的，容易产生不可利用的空闲区（小碎片）；收回主存时，要按分区大小递增顺序插入到空闲区表中。

4.3.4 基于顺序搜索的动态分区分配算法

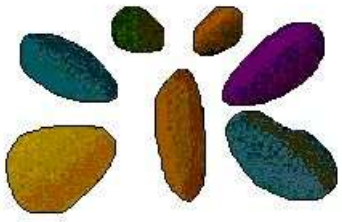
❖ 4、 最坏适应算法（Worst Fit，WF）

* 原理：每次分配时，总是挑选一个最大的空闲分区分割给作业使用；空闲分区按容量大小递减次序链接，查找时只要看第一个分区能否满足进程要求即可，如果满足要求则从中划出一块给进程。
* 优点：可使剩下的空闲分区不至于太小，产生碎片的几率最小，对中、小作业有利，同时该算法查找效率很高。
* 缺点：会使存储器中缺乏大的空闲分区，对大作业不利。

4.3 连续分配方式

- 4.3.1 单一连续分配
- 4.3.2 固定分区分配
- 4.3.3 动态分区分配
- 4.3.4 基于顺序搜索的动态分区分配算法
- 4.3.5 基于索引搜索的动态分区分配算法
- 4.3.6 动态可重定位分区分配

对大、中型OS适用



4.3.5 基于索引搜索的动态分区分配算法

❖ 1、快速适应算法（Quick Fit，QF）

* 原理：QF算法又称为分类搜索算法，该算法根据空闲分区容量大小将空闲分区组织成若干个空闲分区链，并设立一张索引表来对所有空闲分区链集中进行管理，分配时只要从某类满足要求的空闲分区链中取出一个空闲分区整个分配给进程。
* 说明：QF算法通常与FF算法配合使用，事先根据进程常用空间大小从一个大的空闲区中划分出若干类小的空闲分区并分类组织供QF算法使用，如果这些类型的分区大小不能满足进程的要求，就采用FF算法从内存中大的空闲分区划出一块分配给进程。
* 优点：查找效率高，无需分割。
* 缺点：分区归还内存时算法复杂，开销大。

4.3.5 基于索引搜索的动态分区分配算法

❖ 2、伙伴系统

* 分区大小：伙伴系统中内存块的大小为$2^k(l \le k \le m)$
* 分区分配：当进程申请n个字节时，若$2^{u-1}<n\le 2^u$，则在大小为2^u的空闲分区链中查找，如果找到，则分配；否则到大小2^{u+1}的空闲分区链中查找，如果找到，则取出基中一块将其划分为两个大小相等（均为2^u）的伙伴，将这两个伙伴中的一个分配给该进程，另一个加入到大小为2^u的空闲分区链中；如果在大小2^{u+1}的空闲分区也不存在，则到大小2^{u+2}的空闲分区链中查找，如找到，则将其划分成两个大小均为2^{u+1}的伙伴，拿出其中一个进一步划分成两个大小为2^u的伙伴，其中一个分配给该进程，剩余的大小为2^{u+1}、2^u的分区加入到相应大小的空闲分区链中，如果大小2^{u+2}的空闲分区也不存在，则需到2^{u+3}的空闲分区链中查找，依此类推。

4.3.5 基于索引搜索的动态分区分配算法

* 分区回收：当一个进程释放内存时，回收过程需要查找该分区的伙伴是否空闲，如果空闲，则与其伙伴合并成一个更大的分区。一次回收可能要进行多次合并。
* 管理特点：
  - 伙伴系统方式是固定分区方式和动态分区方式的折衷
  - 分配和回收的时间性能取决于查找空闲分区的位置和分割、合并空闲分区所花费的时间，其时间性能优于顺序搜索动态分区分配算法（即4.3.4节所授算法），差于快速适应算法；空间性能则优于快速适应算法，差于顺序适应算法。
  - 在多处理机系统中得以广泛使用。

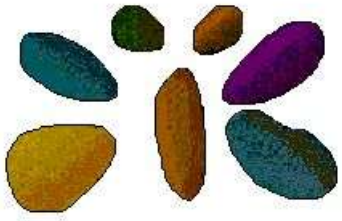
4.3.5 基于索引搜索的动态分区分配算法

❖ 3、哈希算法

* 快速适应算法及伙伴系统中每一类空闲分区均应设置一个空闲分区链表，因而存在大量的空闲分区链表，需要建立一个索引表进行管理。
* 哈希算法管理空闲分区就是根据空闲分区的分布规律建立哈希函数，构造一张以空闲分区大小为关键字的哈希表，以充分发挥哈希算法查找速度快的优点，来对空闲分区链表的查找进行管理。
* 注：哈希算法如果不熟悉的话，建议去看一下《数据结构》相关章节。

4.3 连续分配方式

- 4.3.1 单一连续分配
- 4.3.2 固定分区分配
- 4.3.3 动态分区分配
- 4.3.4 基于顺序搜索的动态分区分配算法
- 4.3.5 基于索引搜索的动态分区分配算法
- 4.3.6 动态可重定位分区分配



4.3.6 动态可重定位分区分配

❖ 1、动态重定位的引入

* 连续分配存在的问题

- 内存中必须有足够大的连续空闲空间才能分配，即使存在多个总量大于进程所请求内存大小的不连续小空闲分区也不能分配。

* 解决办法

- 紧凑（拼接）：通过作业移动将原来分散的小分区拼接成一个大分区。

* 动态重定位

- 为了支持作业移动，需采用运行时动态重定位装入方式（作业已经装入，相对地址等到运行时再重新定位成绝对地址）。

4.3.6 可重定位分区分配

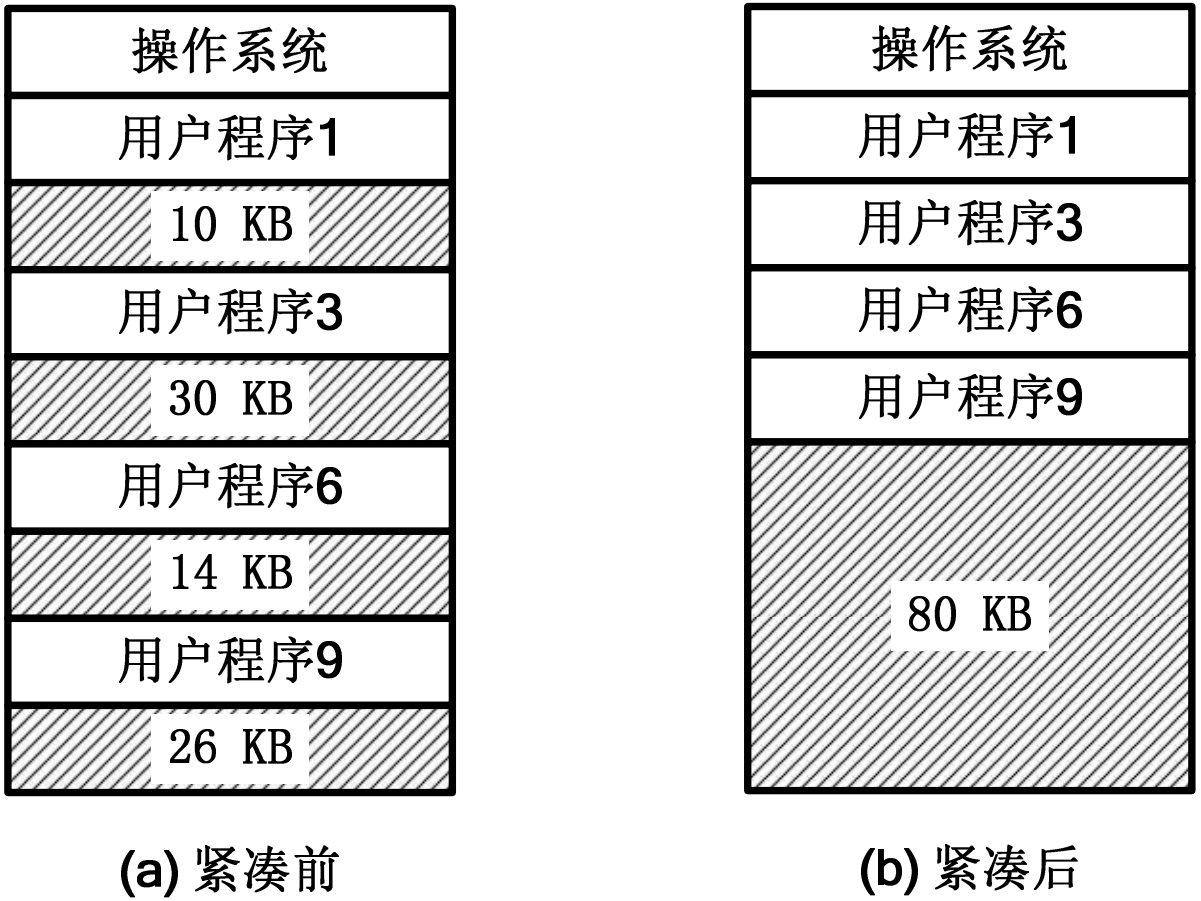


图 4-9　紧凑示意图

4.3.6 可重定位分区分配

❖ 2、动态重定位的实现

* 动态重定位：在程序执行期间将程序中的指令和数据的相对地址转换成物理地址（=相对地址+重定位寄存中存储的起始地址）。

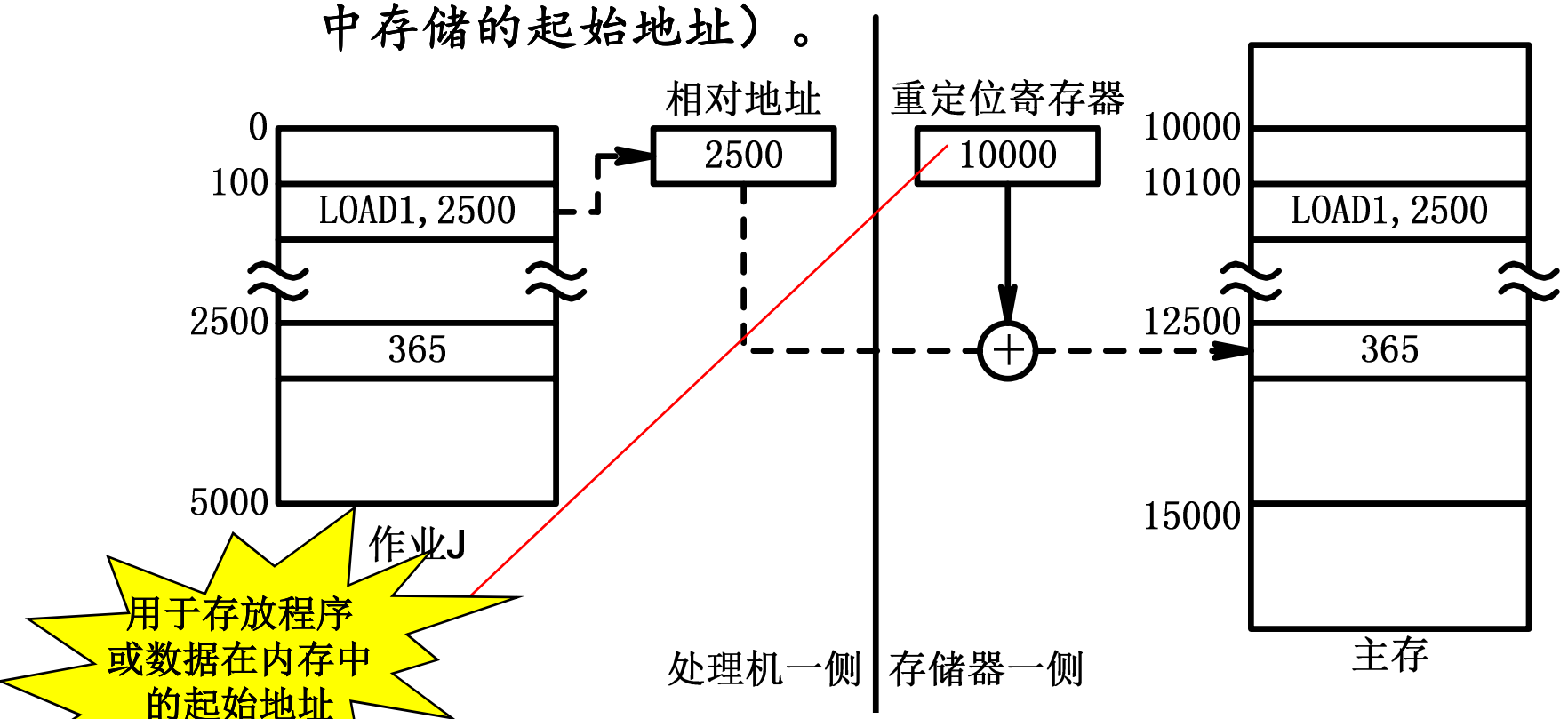


图 4-10 动态重定位示意图

4.3.6 可重定位分区分配

❖ 3、动态重定位分区分配算法

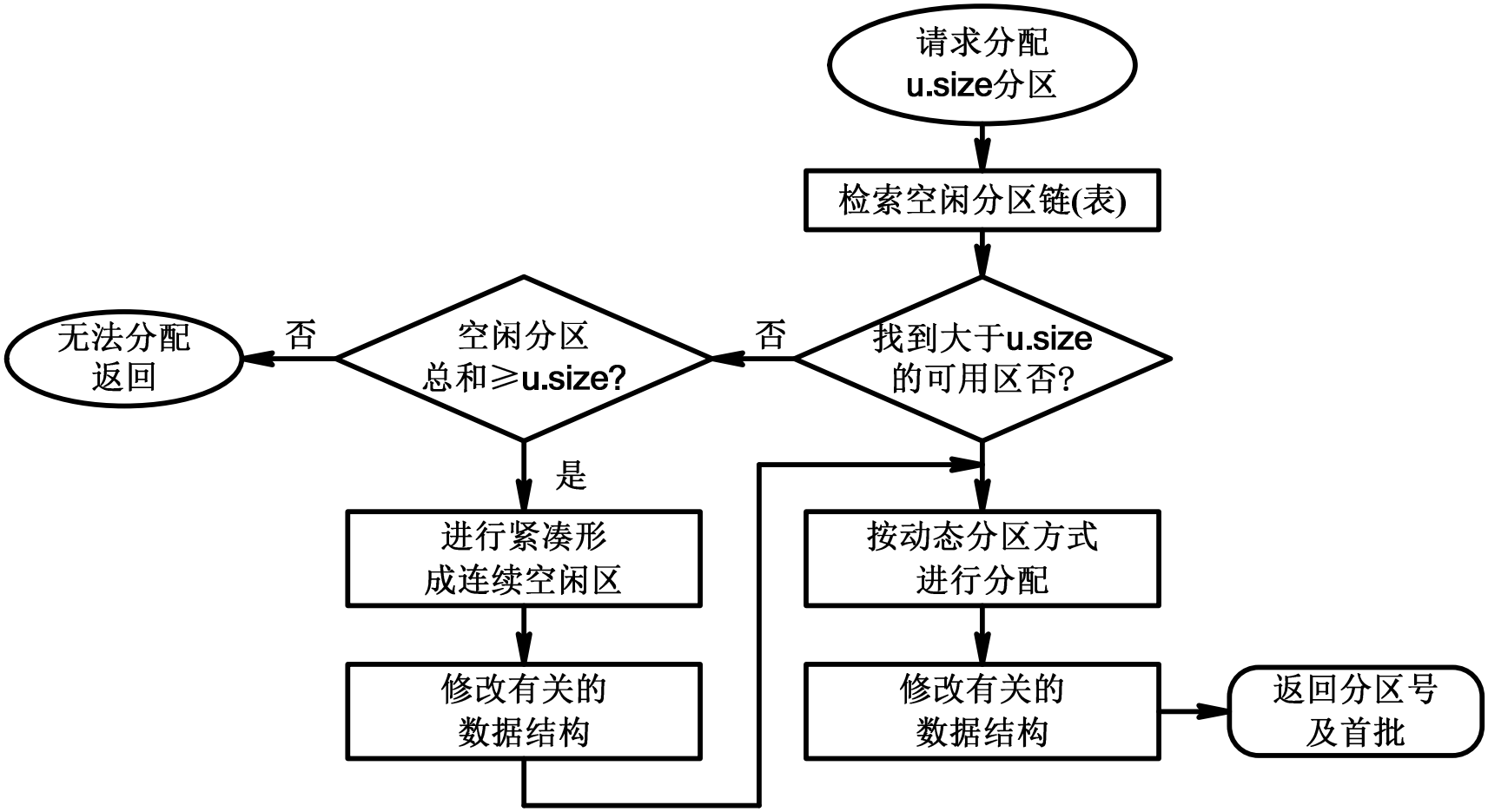


图4-11 动态重定位分区分配算法流程图

本章主要内容

- 4.1 存储器的层次结构
- 4.2 程序的装入和链接
- 4.3 连续分配存储管理方式
- 4.4 对换
- 4.5 基本分页存储管理方式
- 4.6 基本分段存储管理方式

4.4 对换

❖ 1、对换的引入

* 对换的定义

- 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，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，以提高内存利用率。

* 对换的类型

- 整体对换：进程对换，用来解决内存紧张问题
- 部分对换：页面对换、分段对换，用来提供虚拟存储器支持

4.4 对换

* 进程对换涉及的三大问题
  - 对换空间的管理
  - 进程的换入
  - 进程的换出

4.4 对换

❖ 2、对换空间的管理

* 外存的类型

- 文件区—存放文件，侧重提高外存的空间利用率，采用离散分配方式分配
- 对换区—存放从内存中置换出的进程、页或段，侧重于提高对换的速度，采用连续分配方式分配。

* 对换空间的管理目标

- 以提高进程的换入和换出速度为主要目标，然后才再考虑提高对换区的存储空间利用率。

* 对换空间的分配与回收

- 采用的数据结构、分配与回收的方法类似于前面讲的动态分区分配。

4.4 对换

❖ 3、进程的换出与换入

* 1> 进程的换出

- 选定拟换出进程，考虑因素：优先级高低、驻留时间长短、进程状态（阻塞态>就绪态）、进程占用内存长短等。
- 换出过程：将选定的换出进程置换到外存对换区上，再回收该进程占用的内存，并对该进程的PCB进行修改。

* 2> 进程的换入

- 选取拟换入进程，考虑因素：优先级高低、换出时间长短、进程状态、进程需要的内存大小等。
- 换入过程：申请内存，换入内存，再释放其占用的对换区。

本章主要内容

- 4.1 存储器的层次结构
- 4.2 程序的装入和链接
- 4.3 连续分配存储管理方式
- 4.4 对换
- 4.5 基本分页存储管理方式
- 4.6 基本分段存储管理方式

为用户程序分配离散的内存空间，提高内存利用率。

4.5 基本分页存储管理方式

❖ 离散分配方式

* 连续分配方式的不足：要求为每一个进程分配连续的内存空间（整体装入），形成了许多“碎片”，存在一定的浪费，“紧凑”操作虽然可解决这一问题，但付出的代价相当大。
* 如果允许一个进程直接分散地装入到许多不相邻接的内存分区中，称为离散分配方式。
* 离散分配方式类型
  - 分页存储管理：基本分页管理、请求分页管理
  - 分段存储管理：基本分段管理、请求分段管理
  - 段页存储管理

4.5 基本分页存储管理方式

❖ 基本分页存储管理

* 基本分页存储管理也称纯分页存储管理，是指不具备页面对换功能的分页存储管理方式，它不具备支持实现虚拟存储器的功能，要求把每个作业的全部装入内存后才能运行。

* 4.5.1 页面和页表
* 4.5.2 地址变换机构
* 4.5.3 访问内存的有效时间
* 4.5.4 两级和多级页表
* 4.5.5 反置页表

4.5.1 页面和页表

❖ 1、页面

* 1> 页面和物理块

- 分页存储管理：是指将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，从0开始为各页编号称为页号）、将内存空间分成与页面大小相同的多个存储块（称为物理块、块或页框frame，从0开始为各块编号，称为块号），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可以不相邻接的物理块中的一种存储管理方式。
- 在内存分配过程中，由于进程的最后一页经常装不满一个块，使得该块内部形成了不可利用的碎片，称之为“页内碎片”。

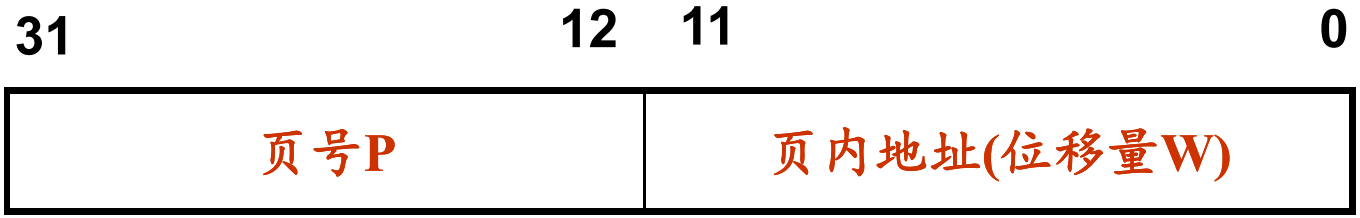
4.5.1 页面和页表

* 2> 页面大小
  - 页面大小由机器的地址结构决定，一般是2的幂，通常为1KB-8KB。
    - 页太大：页内碎片大。
    - 页太小：页表可能很长；换入/换出效率低。

4.5.1 页面和页表

❖ 2、地址结构

* 分页地址包括页号和页内地址（位移量、页内相对地址），大小为4KB、地址长度为32位的分页其地址结构如下：



* 对某特定机器，其地址结构是一定的。对逻辑地址空间中地址为A的逻辑地址而言，如果页面大小为L，则其对应的页号P和页内地址d可按下式求得：

$$P = INT[\frac{A}{L}] \qquad d = A \ Mod \ \ L$$

4.5.1 页面和页表

* 【例】系统页面大小为1KB，逻辑地址为2170B，求页号与页内地址。
  - 页号　　P=INT[2170/1024]=2
  - 页内地址　d=2170 mod 1024 =122
  - 第0页　　逻辑地址范围为：0~1023
  - 第1页　　逻辑地址范围为：1024~2047
  - 第2页　　逻辑地址范围为：2048~3071

4.5.1 页面和页表

❖ 3、页表

* 分页系统中，进程的每一页与内存的每一物理块中成一一映射关系，为了便于管理，需要为每个进程建立一张页面映像表，简称页表。
* 页表的作用：实现从页号到物理块号的地址映射。

4.5.1 页面和页表

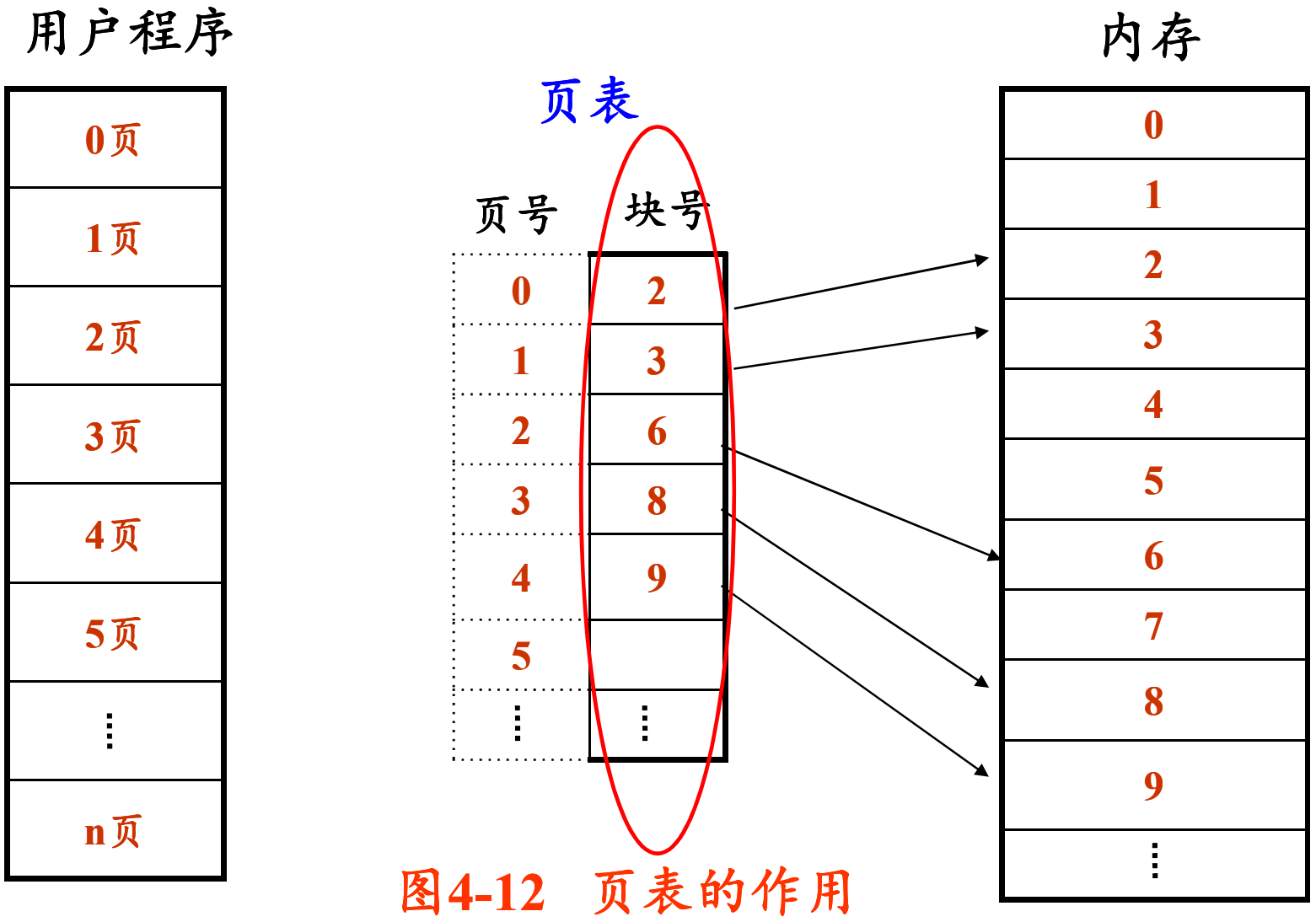
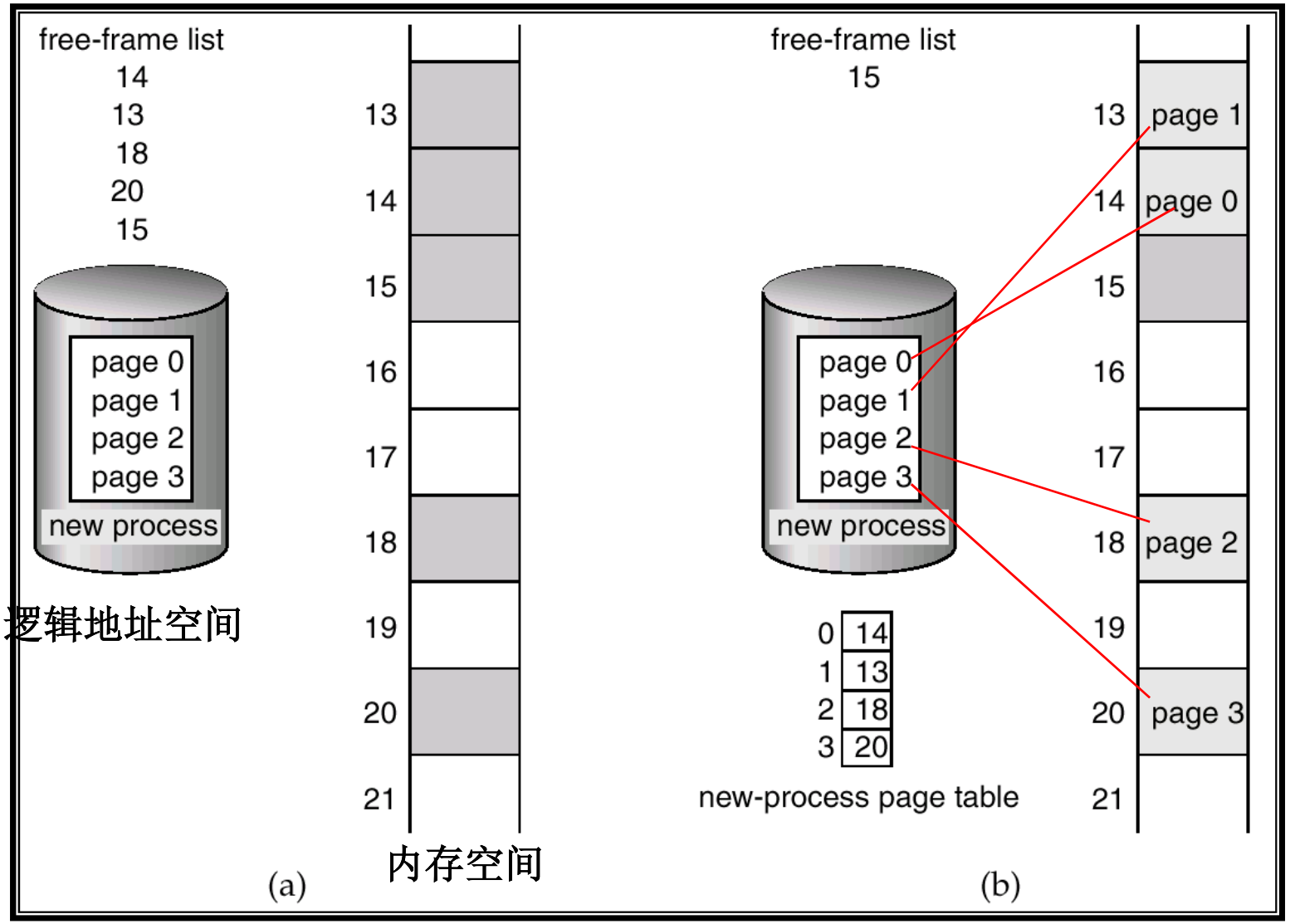


图4-12 页表的作用

4.5.1 页面和页表


free-frame list
14
13
18
20
15
page 0
page 1
page 2
page 3
new process
逻辑地址空间
13
14
15
16
17
18
19
20
21
内存空间
(a)
free-frame list
15
page 0
page 1
page 2
page 3
new process
0 14
1 13
2 18
3 20
new-process page table
13 page 1
14 page 0
15
16
17
18 page 2
19
20 page 3
21
(b)

4.5.1 页面和页表

- 【补充示例2】某系统采用页式存储管理策略，拥有逻辑空间32页，每页2KB，拥有物理空间1MB：
  - ① 写出逻辑地址的格式
  - ② 若不考虑访问权限等，进程的页表有多少项？每项至少有多少位（页表项长度为多少位）？
  - ③ 如果物理空间减少一半，页表结构应相应作怎样的改变？

4.5.1 页面和页表

❖ 解：① 该系统拥有逻辑空间32页，故逻辑地址中页号必须用5位描述；而每页为2KB，因此，页内地址必须用11位描述，格式如下：

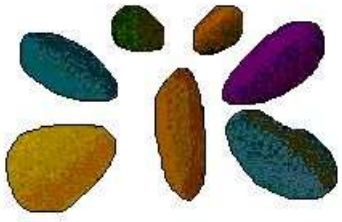
② 每个进程最多32个页面，因此进程的页表项最多为32项；页表项只需给出页所对应的物理块块号，1MB的物理空间可分为2^9个内存块，故每个页表项至少有9位。

为什么不是5 + 9=14？？？

③ 如果物理空间减少一半，则页表中页表项数不变，但每项的长度减少1位。

4.5 基本分页存储管理方式

- 4.5.1 页面和页表
- 4.5.2 地址变换机构
- 4.5.3 访问内存的有效时间
- 4.5.4 两级和多级页表
- 4.5.5 反置页表



4.5.2 地址变换机构

❖ 地址变换机构

* 地址变换机构的任务

- 用于实现从逻辑地址到物理地址的转换。由于页面大小与物理块大小相同，故地址变换机构的实际任务是将页面的页号转换为内存中的块号。

* 地址变换机构任务的实现

- 借助页表来完成，页表一般放在内存中。

* 地址变换机构的类型

- 1> 基本的地址变换机构
- 2> 具有快表的地址变换机构

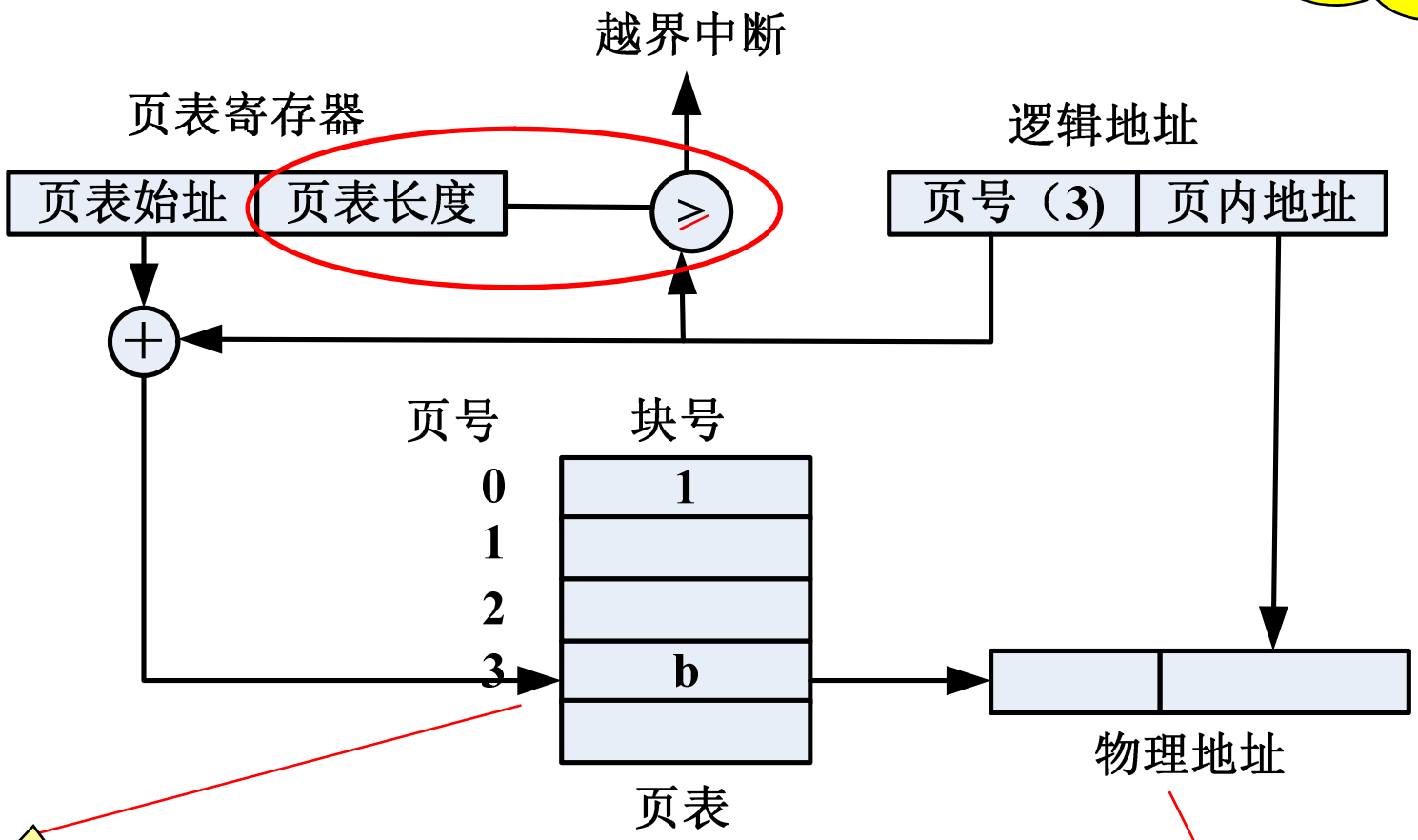
4.5.2 地址变换机构

❖ 1、基本的地址变换机构

* 页表寄存器：用来临时加载需要执行的进程的页表始址、页表长度。每个进程未执行时将其页表信息（如页表长度、始址）放在PCB中，执行时将它们装入页表寄存器。
* 地址变换机构可自动将需要转换的逻辑地址（由页号和页内地址组成）转成物理地址
* 基本地址变换流程（见图4-15、配套动画演示请自行到网络课程平台下载）

4.5.2 地址变换机构

存取一个数据需访问内存两次



页号对应的页表项的物理地址
=页表始址+页号*页表项长度

物理地址＝物理块号×页大小＋页内地址

4.5.2 地址变换机构

- 【补充示例3】已知某分页系统，主存容量为64KB，页面大小为1KB，对一个4页大的作业，其0、1、2、3页分别被分配到主存的2、4、6、7块中。
  - ①将十进制的逻辑地址1023、2500、3500、4500转换成物理地址？
  - ②以十进制的逻辑地址1023为例画出地址变换过程图？

4.5.2 地址变换机构

- 解：① 逻辑地址1023：1023/1024=0…1023，得页号为0，页内地址为1023，查页表找到对应的物理块号为2，故物理地址为2×1024+1023=3071;

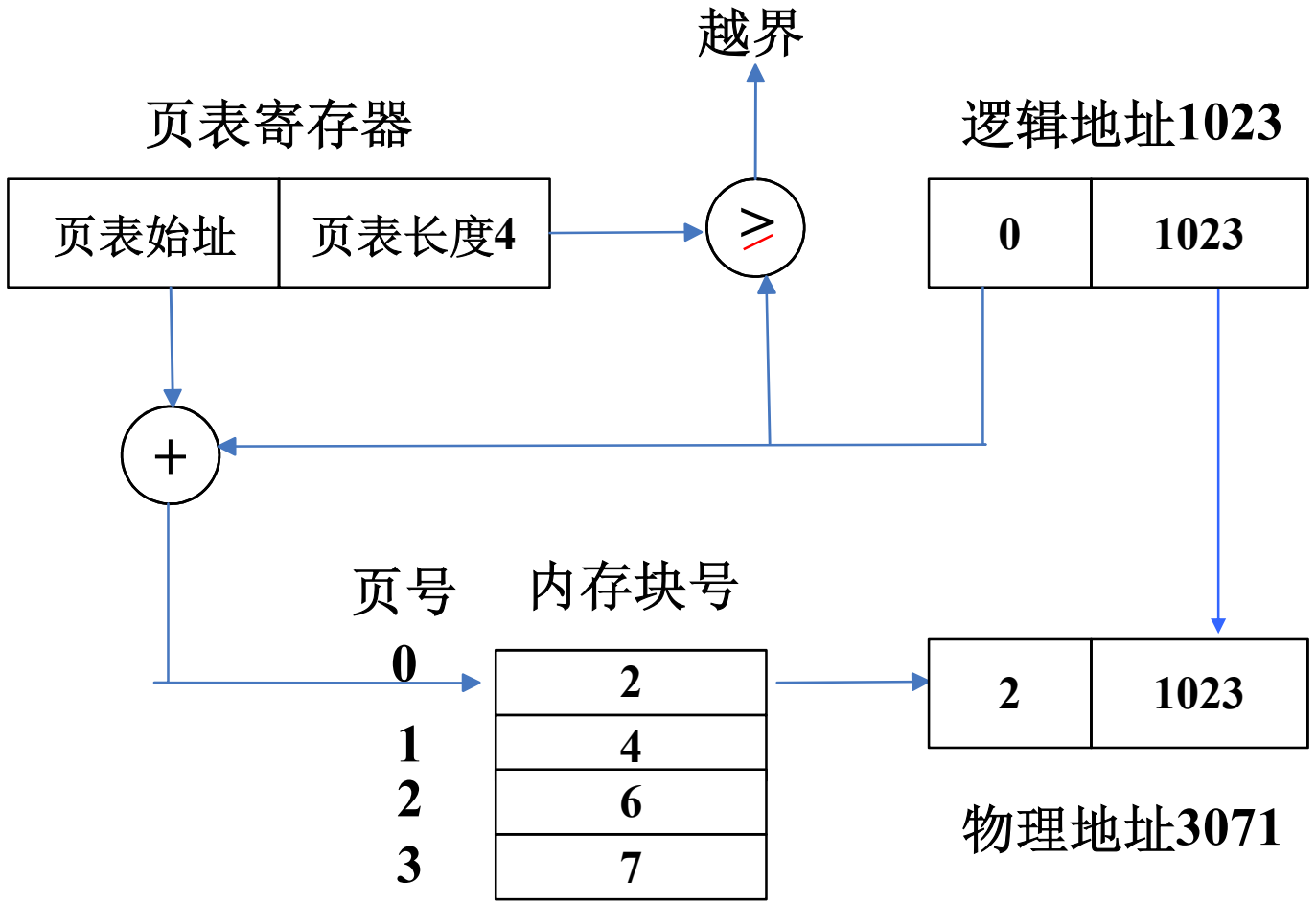
  逻辑地址2500：2500/1024=2…452，得页号为2，页内地址为452，查页表找到对应的物理块号为6，故物理地址为6×1024+452=6596

  逻辑地址3500：3500/1024=3…428，得页号为3，页内地址为428，查页表找到对应的物理块号为7，故物理地址为7×1024+428=7596

  逻辑地址4500：4500/1024=4…4，得页号为4，页内地址为404，因页号不小于页表长度，故产生越界中断。

4.5.2 地址变换机构

❖ 解：②



4.5.2 地址变换机构

❖ 【补充示例4】某存储器的用户编程空间共32个页面，每页为1KB，内存为16KB。假定某时刻一用户页表中已调入内存的页面对应的物理块号如下表：

页号	物理块号
0	5
1	10
2	4
3	7

则逻辑地址0A5C（H）所对应的物理地址为多少？

4.5.2 地址变换机构

❖ 解： 0A5C＝10*256+5*16+12=2652

2652/1024=2 …604

页号为2，对应块号为4，有：

物理地址：4*1024+604=1*16*16*16+2*16*16

+5*16+12

即：125C（H）

4.5.2 地址变换机构

❖ 2、具有快表的地址变换机构

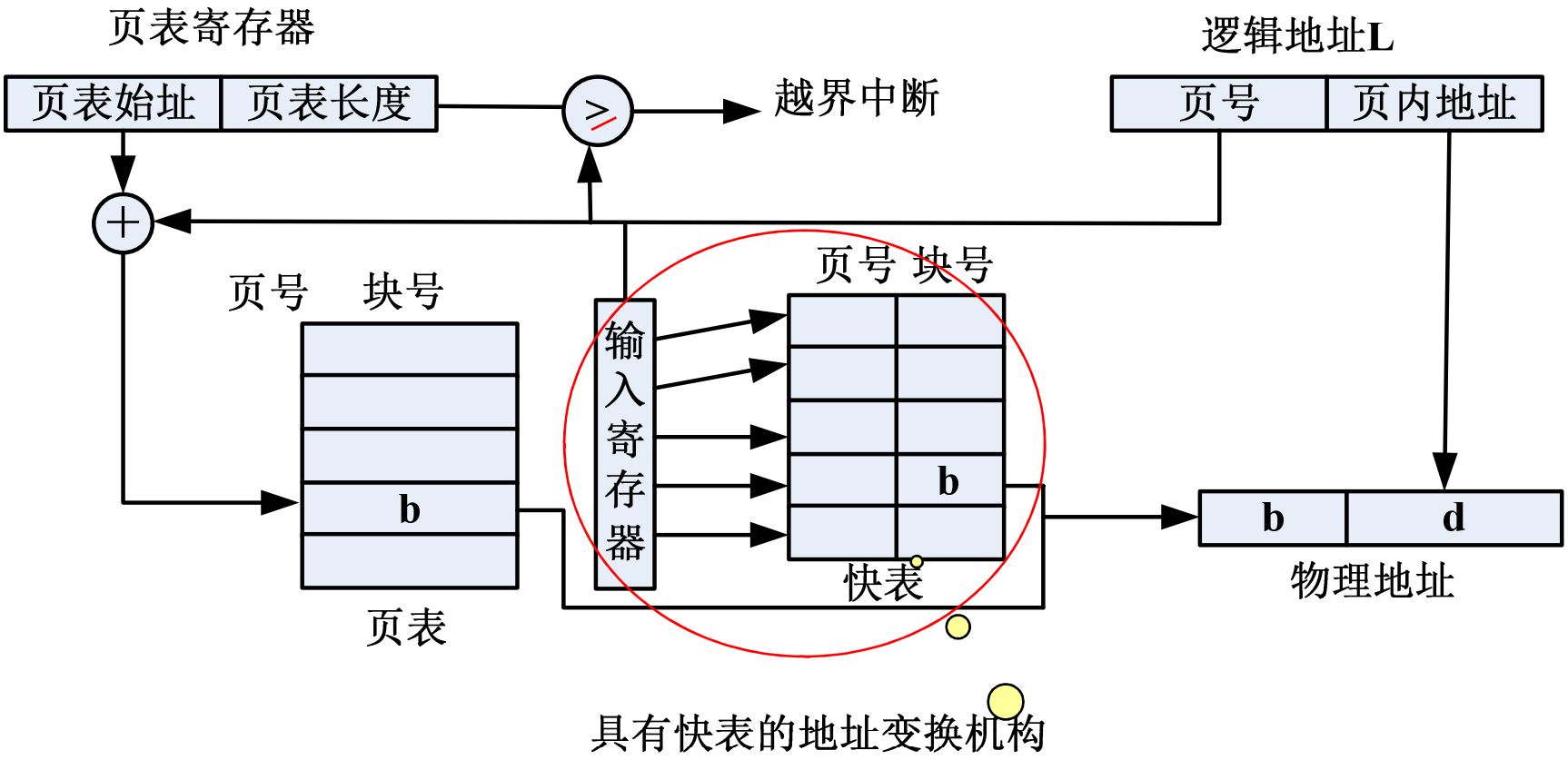
* 引入背景

- 基本地址变换机构存取一个数据需两次访问内存，即：1> 访问页表，将逻辑地址转换成物理地址；2> 根据物理地址来存取数据。

* 基本原理

- 在基本地址变换机构增设一个高速缓冲寄存器（联想寄存器、快表），用其缓存最近访问过的页表或部分页表（快表成本高，不能做得太大）来加快地址转换速度。地址转换时首先查找快表中有无对应的页号，如果没有，再访问内存中的页表，得出块号，并重新修改快表。

4.5.2 地址变换机构

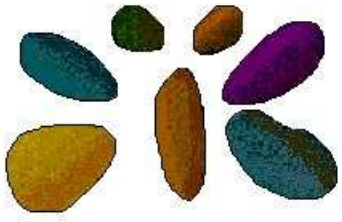


具有快表的地址变换机构

快表通常可存放16-512个页表项,命中率可达90%以上

4.5 基本分页存储管理方式

- 4.5.1 页面和页表
- 4.5.2 地址变换机构
- 4.5.3 访问内存的有效时间
- 4.5.4 两级和多级页表
- 4.5.5 反置页表



4.5.3 访问内存的有效时间

❖ 有效访问时间（EAT）

* 定义

- 从进程发出指定逻辑地址的访问需求，经过地址变换，到在内存中找到对应的实际物理地址单元并取出数据，花费的总时间称为内存的有效访问时间。

* 无快表有效访问时间

- **EAT = 2t**　　(t为访问一次内存的时间)

* 有快表有效访问时间

- **EAT = P*k + (1-P)(k+t) + t = 2t + k – P*t**

其中：P、k 为快表的命中概率、该问时间

4.5.3 访问内存的有效时间

- 【补充示例5】已知某分页系统，其页表存在主存中：
  - ① 如果对主存的一次存取需要1.5μs，试问实现一次页面访问的存取时间是多少？
  - ② 如果系统加有快表，平均命中率为85%,当页表项在快表中时,其查找时间忽略为0， 试问此时平均存取时间是多少？

4.5.3 访问内存的有效时间

- 解：若页表存放在主存中，则要实现一次页面访问需两次访问主存：一次是访问页表，确定所存取页面的物理地址（称为定位）；第二次才根据该地址存取页面数据，故：

① 页表在主存的存取访问时间

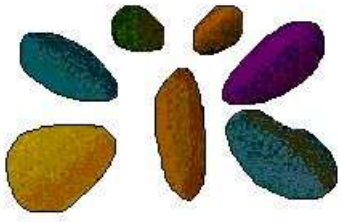
=1.5*2=3(μs)

② 增加快表后的平均存取访问时间

=0.85*1.5+(1-0.85)*2*1.5=1.725(μs)

4.5 基本分页存储管理方式

- 4.5.1 页面和页表
- 4.5.2 地址变换机构
- 4.5.3 访问内存的有效时间
- 4.5.4 两级和多级页表
- 4.5.5 反置页表



4.5.4 两级和多级页表

❖ 1、为什么要引入多级页表？

* 问题

- 对32位或64位计算机来说，支持的逻辑地址空间为2^{32}或2^{64}，如果此时页面大小为4KB，则页表中的页表项数量最大可达1M或2^{52}，数量非常大，如果每个页表项占1个Byte，则整个页表需占用的空间高达1MB或4096TB，此时整个页表很难在内存中连续存储，如何解决这一问题？

* 解决办法

- 1> 对页表分页，采用离散分配方式存储页表；
- 2> 设置状态位，只将当前需要的小部分页表项调入内存，其余大部分页表仍驻留在磁盘上。

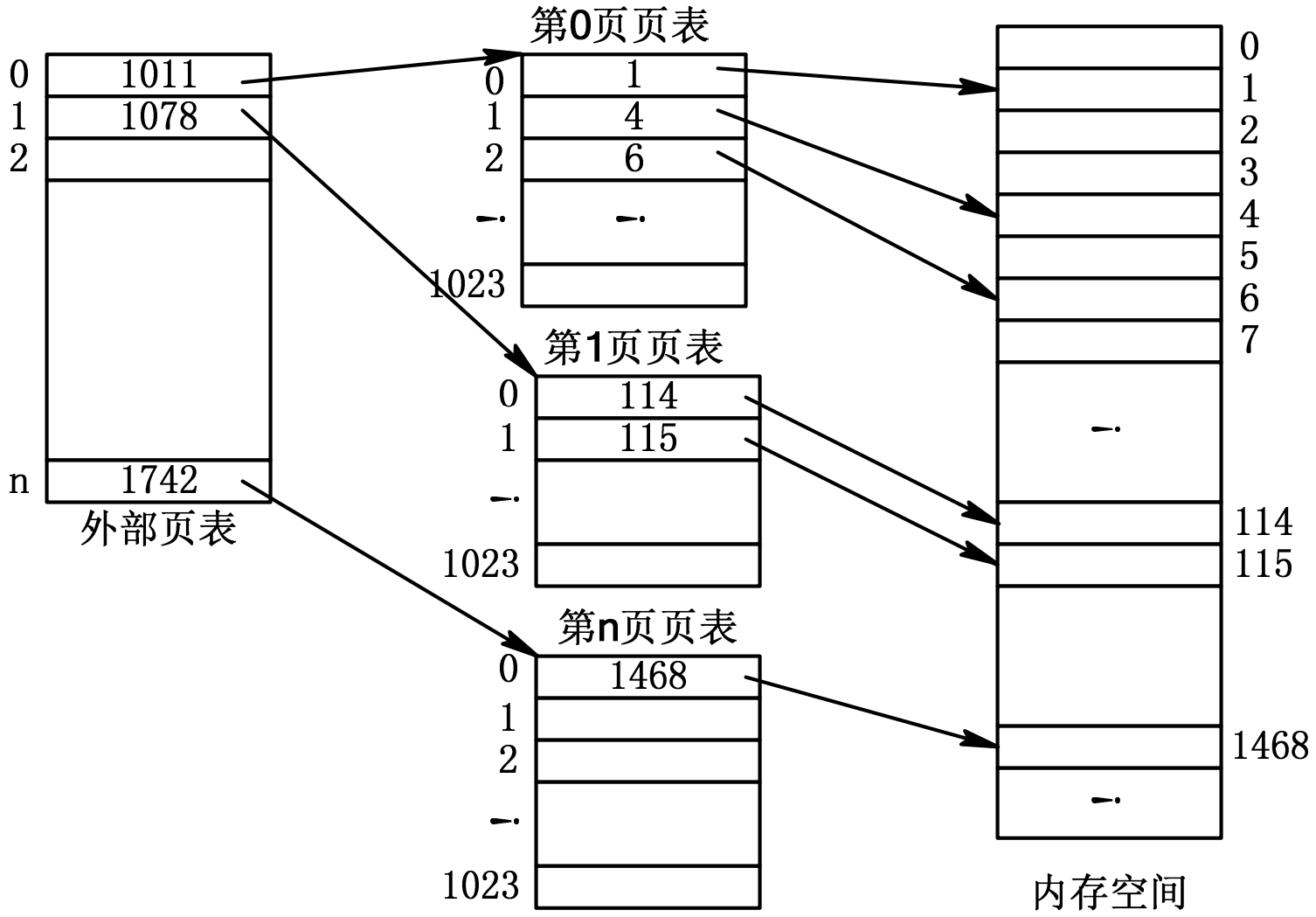
4.5.4 两级和多级页表

❖ 2、两级页表

* 基本原理

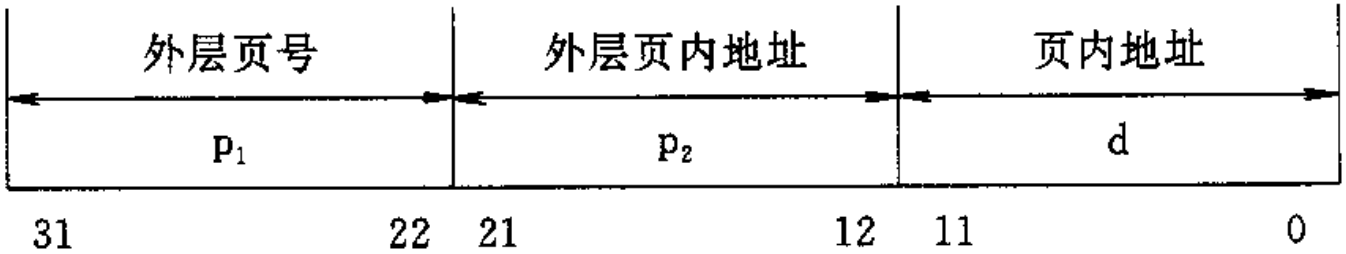
- 将页表分页，离散地将各个页面分别存放在不同的物理块中，并为离散分配的页表（内层页表）再建立一张页表（称为外层页表，Outer Page Table），其每个页表项中记录了内层页表的物理块号。

图 4-18 两级页表结构


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

4.5.4 两级和多级页表

* 两级分页逻辑地址结构



- 例：假设当前逻辑地址长度为32位，页面大小4KB，页表分页后每页含2^{10}个页表项
- 对逻辑地址空间中地址为A的逻辑地址而言，如果页面大小为L，页表项的长度为N，则其对应的外层页号P1和外层页内地址P2、页内地址d可按下式求得：

$$P1=INT[INT[A/L]/N]$$

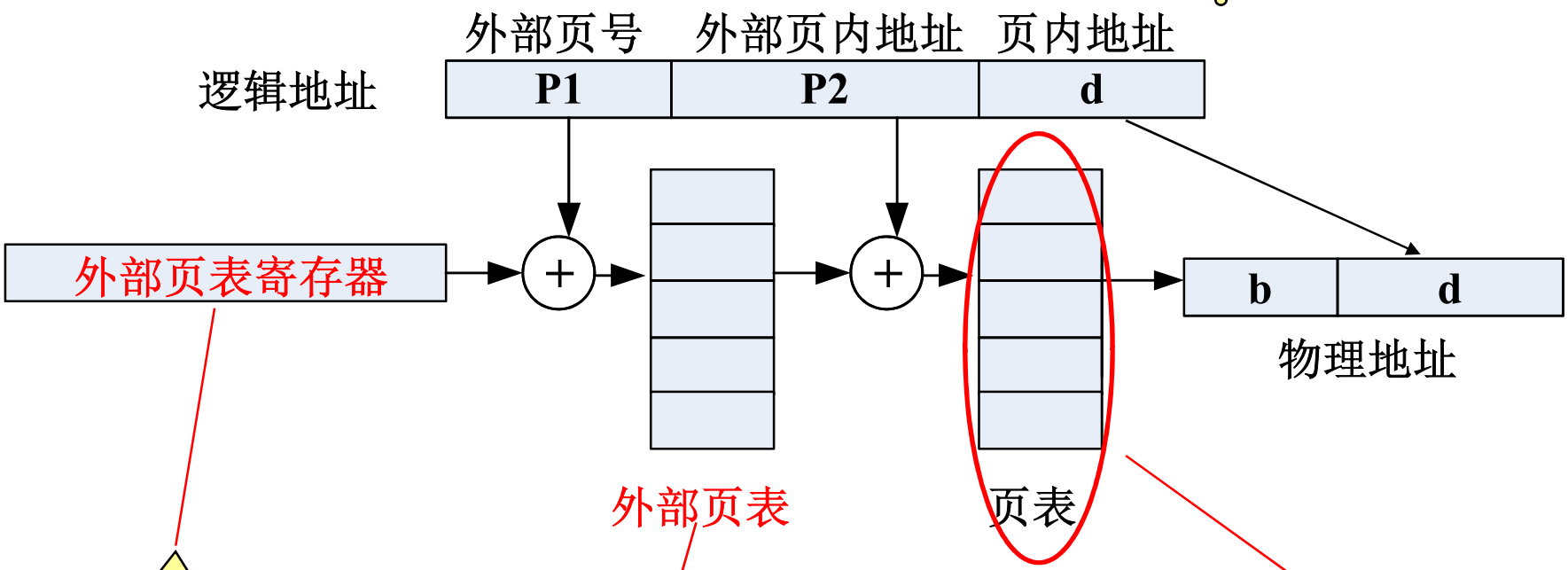
$$P2=INT[A/L]\ Mod\ N$$

$$d=A\ Mod\ L$$

4.5.4 两级和多级页表

需访问几次内存？

* 具有两级页表的地址变换机构



具有两级页表的地址变换机构

存放外部页表的始址

一般应常驻内存，各外部页表项中设置有页表状态位

按需调入内存

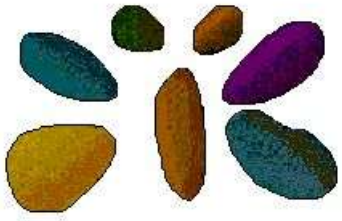
4.5.4 两级和多级页表

❖ 3、多级页表

* 对于32位的机器，采用两级页表结构是合适的。
* 对于64位的机器，采用两级页表结构时，外层页表仍然大太，需要再进行分级。
* 64位机器页表一般>3级，最外层页表常驻内存。

4.5 基本分页存储管理方式

- 4.5.1 页面和页表
- 4.5.2 地址变换机构
- 4.5.3 访问内存的有效时间
- 4.5.4 两级和多级页表
- 4.5.5 反置页表



4.5.5 反置页表

❖ 1、反置页表的引入

* 在分页系统中，一个进程一般配一张页表。传统页表是按页号排序的，页表内容是物理块号，如果进程的逻辑地址空间特别大，则页表也变得非常大。为了减少页表占用的内存空间，反置页表按物理块号排序，页表项中存储的是页号。

❖ 2、地址变换

* 首先根据pid和页号查找反置页表，若找到则该页表项的序号就是对应的物理块号，若没找到则转向外部页表查找并请求将该页调入内存，接下来与传统页表地址变换相同。

注：反置页表只记录了已调入内存的页面，未入内存的页面另设一张外部页表记录。

本章主要内容

- 4.1 存储器的层次结构
- 4.2 程序的装入和链接
- 4.3 连续分配存储管理方式
- 4.4 对换
- 4.5 基本分页存储管理方式
- 4.6 基本分段存储管理方式

从逻辑上对作业进行划分，为用户程序分配离散的内存空间，实现信息共享。

4.6 基本分段存储管理方式

❖ 分段存储管理

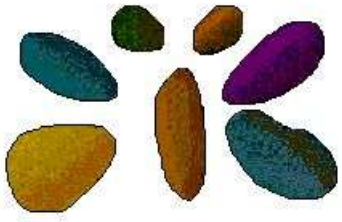
* 在分段存储管理中，按信息的逻辑关系将作业的地址空间划分为若干个段(各段大小不一定一样)。
* 它以段为单位分配主存，每段分配一个连续的主存空间，但各段之间不要求连续。由于各段的长度不一样，所以分配的内存空间大小也不一样。
* 供用户使用的逻辑地址为段号(段名)+段内地址。在装入作业时，用一张段表记录每个分段在主存中的起始地址和长度。若待装入的某段信息找不到足够大的空闲区，可采用“紧凑”技术，合并分散的空闲区。

4.6 基本分段存储管理方式

* 主存的分配与回收类似于动态分区分配，采用动态重定位。
* 基本分段存储管理也称纯分段存储管理，是指不具备对换功能的分段存储管理方式，它不具备支持实现虚拟存储器的功能，要求把每个作业的全部装入内存后才能运行。

4.6 基本分段存储管理方式

- 4.6.1 分段存储管理方式的引入
- 4.6.2 分段系统的基本原理
- 4.6.3 信息共享
- 4.6.4 段页式存储管理方式



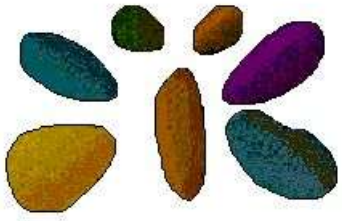
4.6.1 分段存储管理方式的引入

- 引入目的
  - （1）方便编程
  - （2）信息共享
    - 分页系统：页是信息的“物理”单位
    - 分段系统：段是信息的逻辑单位
  - （3）信息保护
  - （4）动态链接
  - （5）动态增长
    - 数据段在使用过程中可能不断地增长

基于信息的逻辑关系来划分段

4.6 基本分段存储管理方式

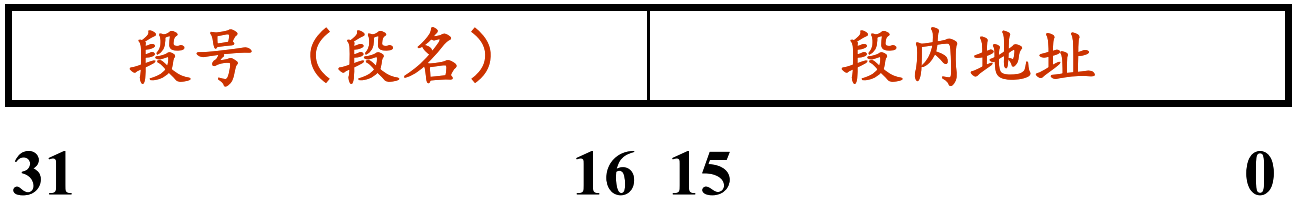
- 4.6.1 分段存储管理方式的引入
- 4.6.2 分段系统的基本原理
- 4.6.3 信息共享
- 4.6.4 段页式存储管理方式



4.6.2 分段系统的基本原理

❖ 1、分段

* 分段存储管理方式中，作业的地址空间被分成若干个段(segment)，每个段定义了一组逻辑信息。
* 分段地址中的地址结构



* 分段方式已得到许多编译程序的支持

4.6.2 分段系统的基本原理

❖ 2、段表

* 在分段存储管理系统中，为每个分段分配一个连续的分区，而进程中的各个段可以离散地移入内存中的不同的分区中。
* 系统为每个进程建立一张段映射表，简称为“段表”
* 每个段在段表中占一个表项，其中记录了该段在内存中的起始地址(又称为“基址”)和段的长度。
* 段表实现从逻辑段到物理内存区的映射。

4.6.2 分段系统的基本原理

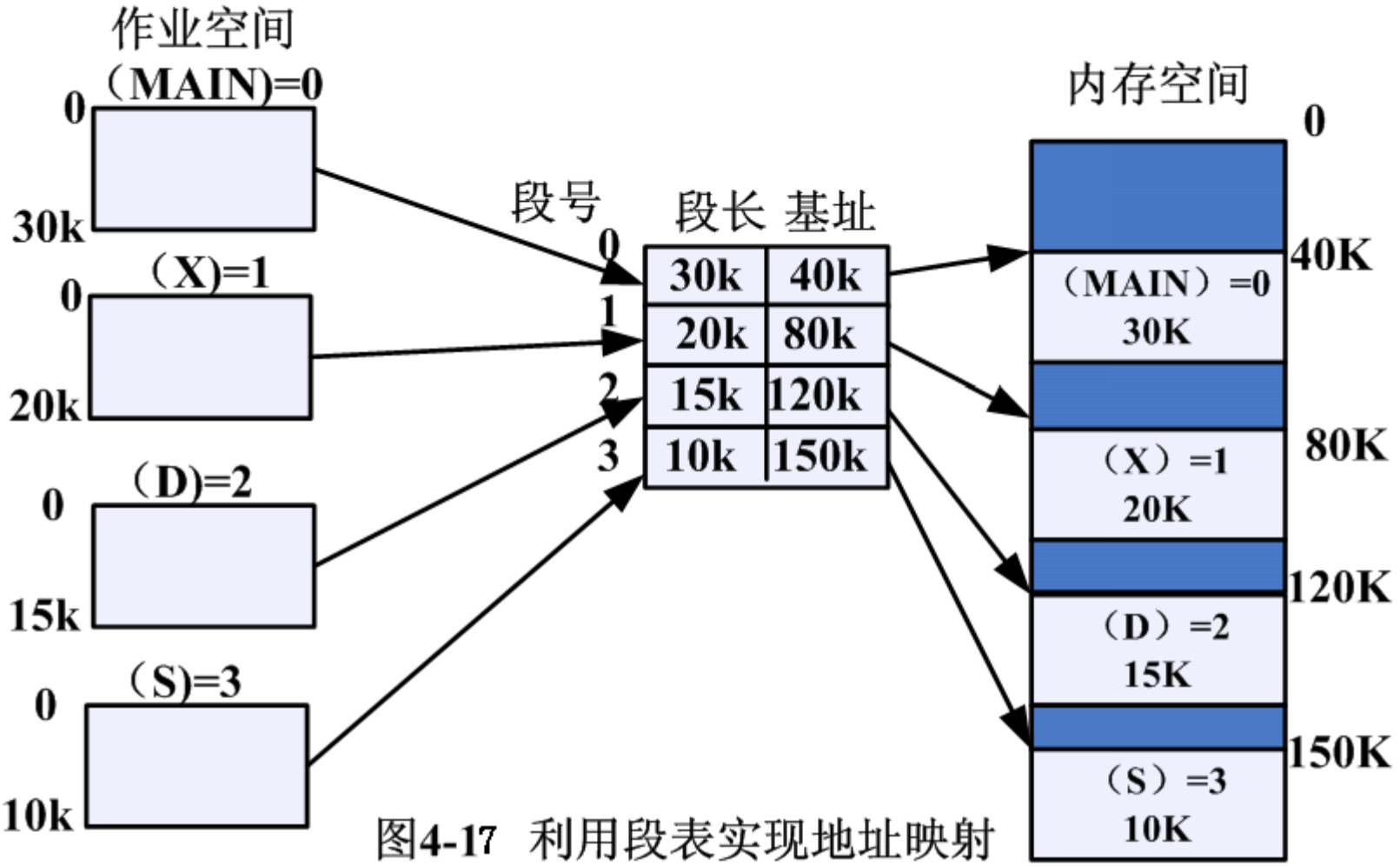
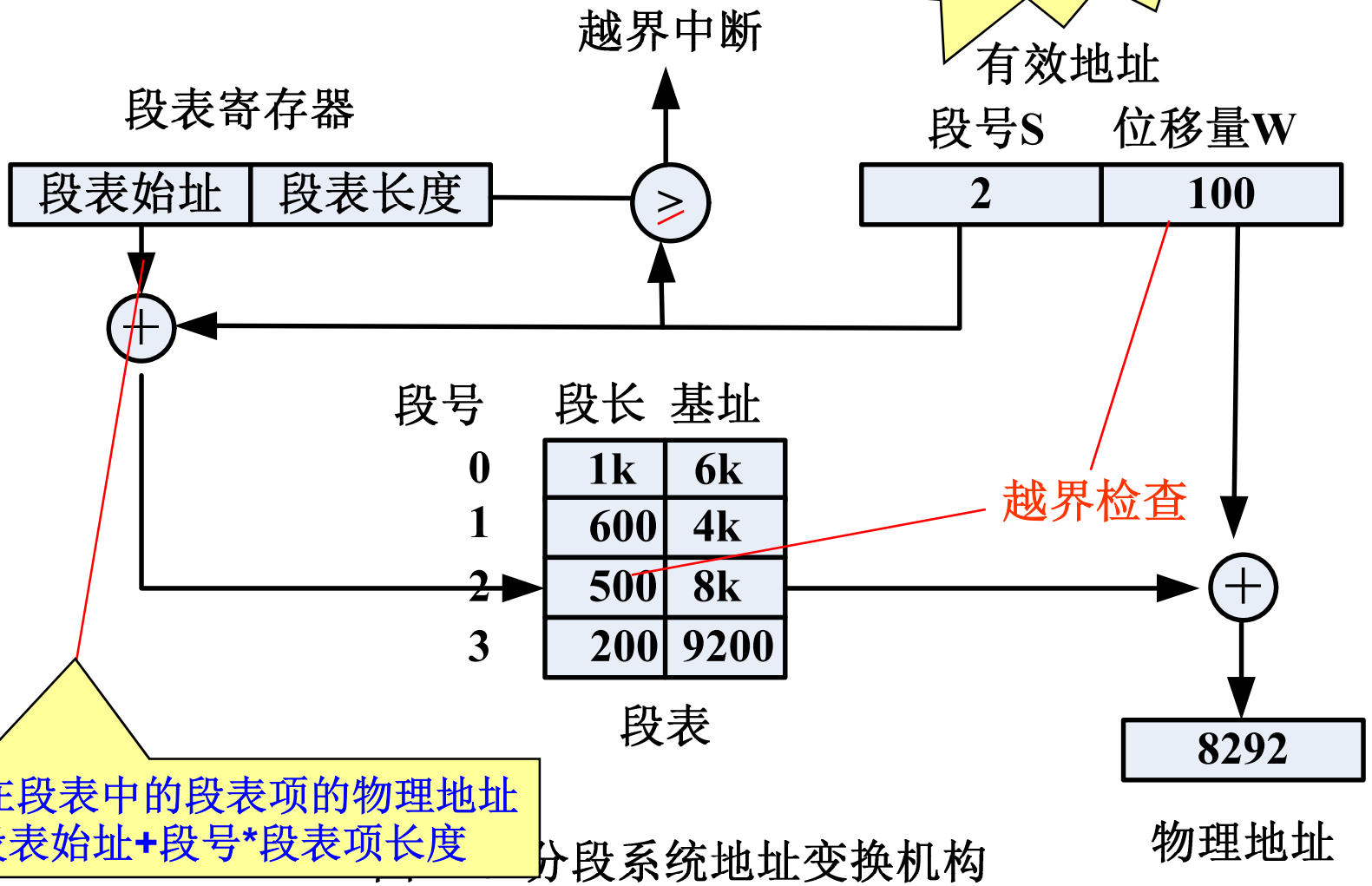


图4-17 利用段表实现地址映射

4.6.2 分段系统的基

可引入快表保存最近使用段表项进行改进!

❖ 3、基本的地址变换机构



分段系统地址变换机构

4.6.2 分段系统的基本原理

- 【补充示例6】对于下面的段表，请将逻辑地址（0，137），（1，4000），(2，3600)，(5，230)转换成物理地址。

段号	基址	段长
0	50K	10K
1	60K	3K
2	70K	5K
3	120K	8K
4	150K	4K

4.6.2 分段系统的基本原理

❖ 解：1> (0，137) 物理地址为：50K+137=51337

2> (1，4000) 段内地址超过段长3K，产生越界中断。

3> (2，3600)物理地址为：70K+3600=75280

4> (5，230)段号等于段表长，段号不合法，产生越界中断。

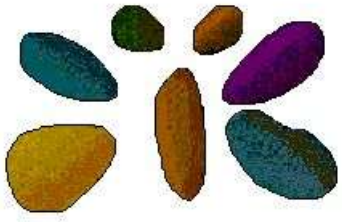
4.6.2 分段系统的基本原理

❖ 4、分页和分段的主要区别（重要！！！）

* （1）页是信息的“物理”单位；段是逻辑单位；
* （2）页长度固定（由机器决定）；段长度不固定（由用户指定）
* （3）分页的作业地址空间是一维地址空间（逻辑地址空间）；段是二维地址空间（段名+段内地址）。

4.6 基本分段存储管理方式

- 4.6.1 分段存储管理方式的引入
- 4.6.2 分段系统的基本原理
- 4.6.3 信息共享
- 4.6.4 段页式存储管理方式



4.6.3 信息共享

❖ 信息共享

* 分段存储的一个优点是易于实现段的共享，即允许若干个进程共享一个或多个分段。
* 分页系统中虽然也能实现程序和数据的共享，但远不如分段系统方便。
* 可重入代码又称纯代码，是一种允许多个进程同时访问的代码，且不允许任何进程对它进行修改。
* 一般代码如何转换成可重入码？
  - (1)各个进程应保留局部数据区，将执行时可能改变的部分，拷贝到该局部数据区，所剩代码就成了“纯代码”。
  - (2) 进程执行时，只对其局部数据区(属于该进程私有)中的内容进行修改，而不去改变共享的代码。

图4-21 分页系统中共享editor的示意图

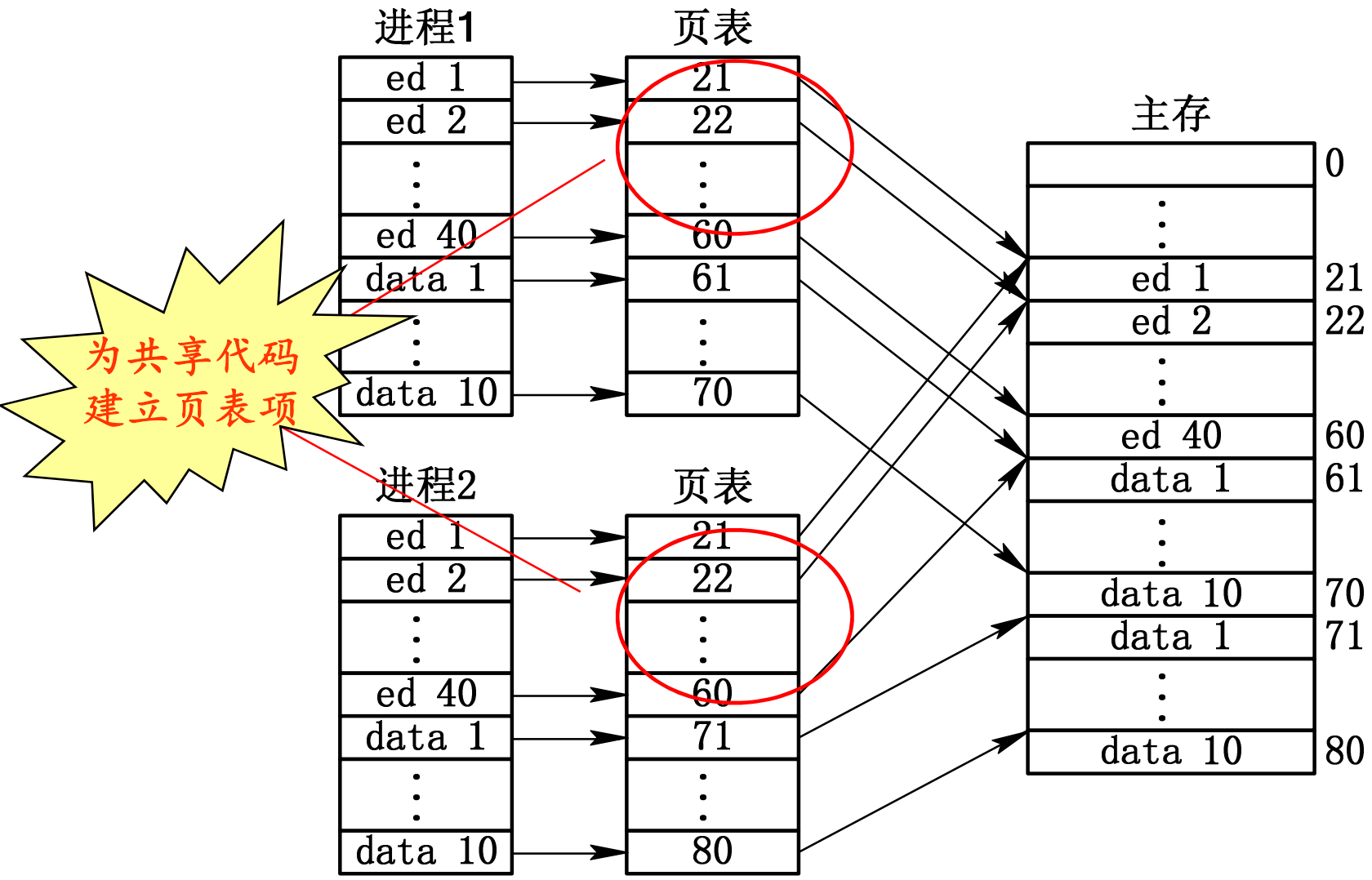
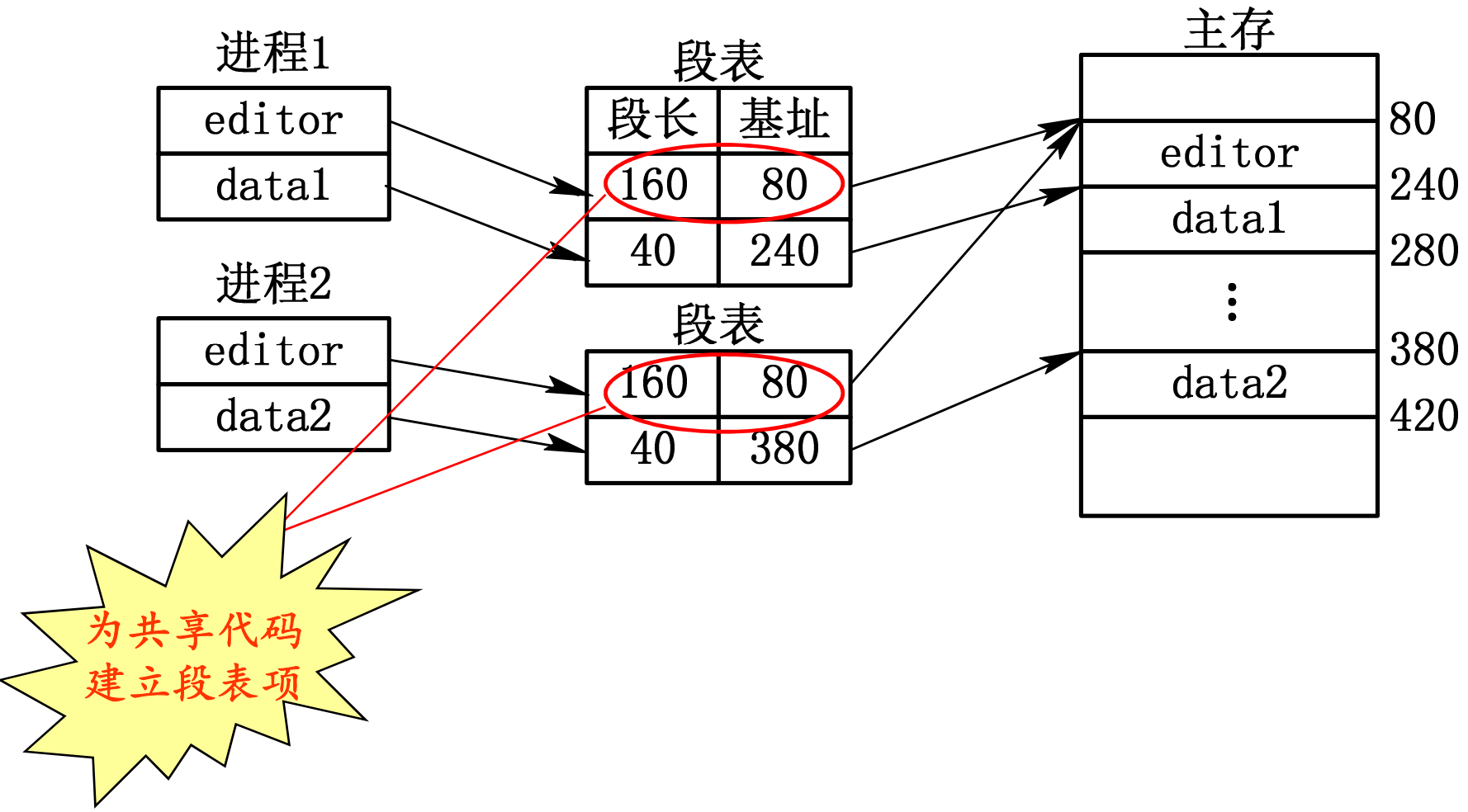
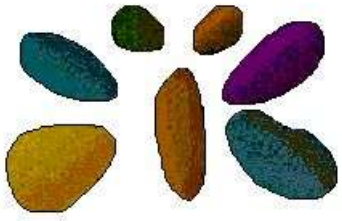


图4-20 分段系统中共享editor的示意图



4.6 基本分段存储管理方式

- 4.6.1 分段存储管理方式的引入
- 4.6.2 分段系统的基本原理
- 4.6.3 信息共享
- 4.6.4 段页式存储管理方式



4.6.4 段页式系统存储管理方式

❖ 1、基本原理

* 段页式系统是分段和分页式系统的结合，将用户程序分成若干个段，并为每一段赋予一个段名，再把每一段分成若干个页，把主存分成与页大小相同的物理块，每段分配与其页数相同的物理块，物理块可以连续，也可以不连续。
* 段页式逻辑地址结构由段号、段内页号及页内地址三部分所组成。

段号(S)	段内页号(P)	页内地址(W)

  - 注意：对用户而言，仍然是二维编址；对系统而言，则是三维编址

4.6.4 段页式系统存储管理方式

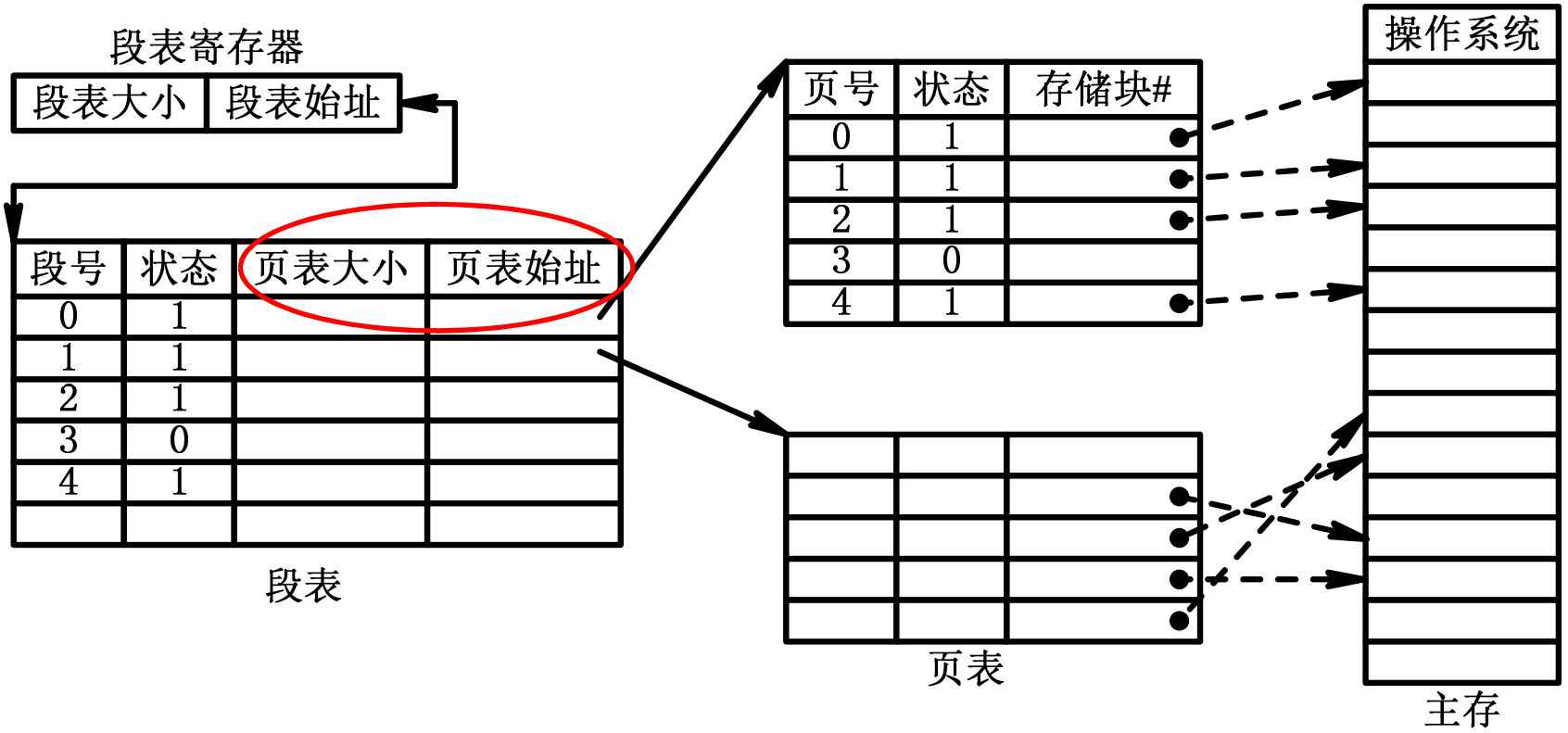
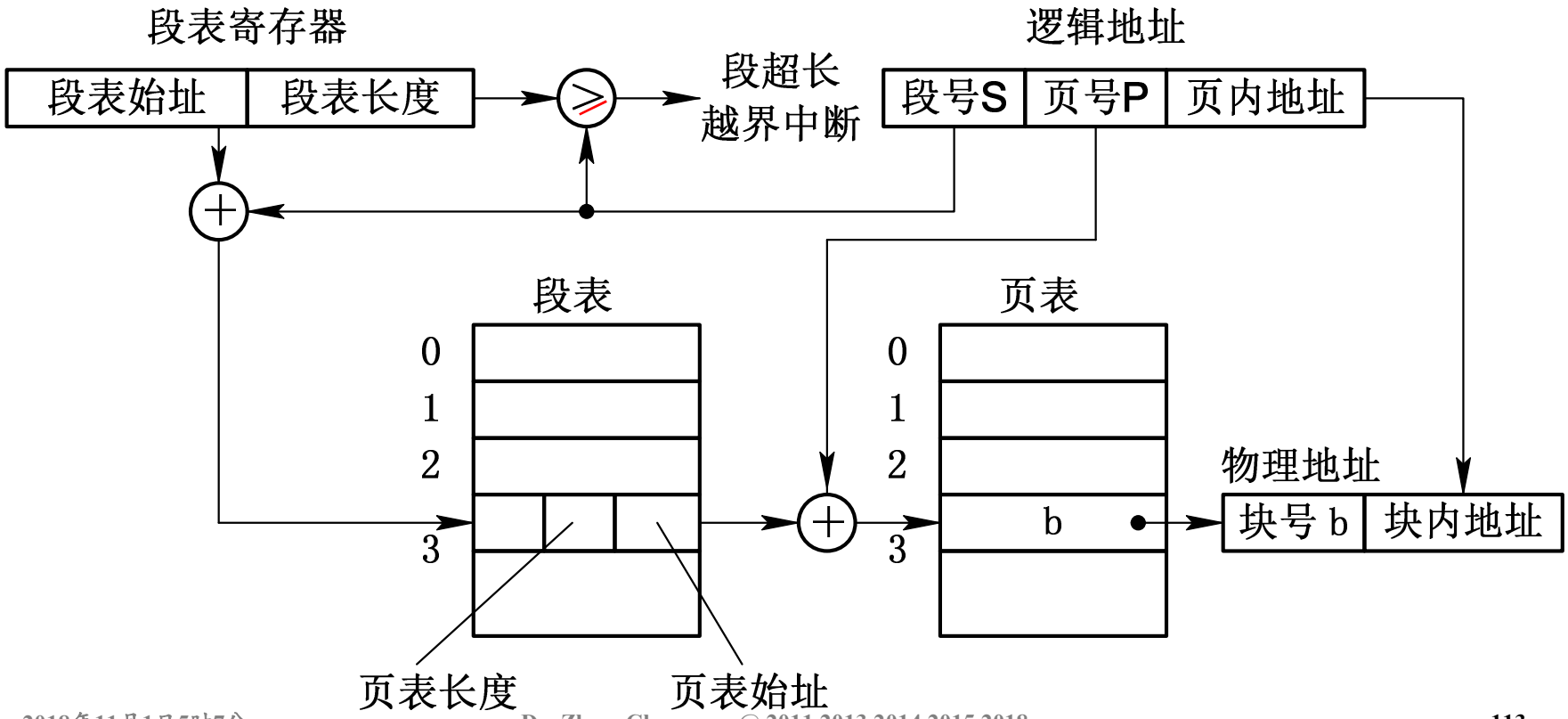


图4-22 利用段表和页表实现地址映射

4.6.4 段页式系统存储管理方式

❖ 2、地址变换机构

* 需三次访问内存，为提高速度，应在地址变换机构中再增设一高速缓冲寄存器（Cache）即快表。



4.6.4 段页式系统存储管理方式

❖ 3、段页式管理特点

* 根据程序情况把程序分成若干段，再根据页面大小把每一段分成若干页，内存仍然分成与页大小相等的块。分配主存时，把程序的每一段的页分配到物理块中。
* 这种分配方式既照顾到了用户共享和使用方便的需求，又考虑到了内存的利用率，提高了系统的性能。
* 这种分配方式比分页管理的空间浪费要多，因为程序各段的最后一页都有可能浪费一部分空间。另外段表和页表占用空间，都比分页和分段多一些，这样就增加了硬件成本、系统的复杂性和开销。

本章小结

- 存储器六级层次结构
- 程序的三种链接方式和三种装入方式*
- 连续存储管理方式（分区）**
  - 四种基于顺序搜索的动态分区分配算法**
  - 两种基于索引搜索的动态分区分配算法*
  - 动态重定位分区分配算法 **
- 离散存储管理方式（分页、分段）**
  - 纯分页存储管理**
  - 纯分段存储管理**
  - 段页式存储管理**

本章小结

❖ 重要概念

* 逻辑地址、物理地址、地址转换、静态重定位、动态重定位、内碎片、外碎片、对换、连续分配、离散分配、页表、段表、分页、分段、分区等

本章作业

- 要求：
  - 一定要做在作业本上
- 交作业日期：
  - 下周课上提交
- 作业内容：
  - 操作系统第4章网络在线测试
  - 课后习题P152 第2-5题、第27题 及补充题2道

本章作业

- 补充题1：某存储器的用户编程空间共32个页面，每页为1KB，内存为16KB。假定某时刻一用户页表中已调入内存的页面对应的物理块号如下表：

页号	物理块号
0	5
1	10
2	4
3	7

而该用户作业的长度为6页，试将十六进制的逻辑地址0A5C、103C、1A5C转换成物理地址。

本章作业

- 补充题2：某系统采用动态分区分配方式管理内存，内存空间为640K，高端40K用来存放操作系统。在内存分配时，系统优先使用空闲区低端的空间。对下列的请求序列：作业1申请130K、作业2申请60K、作业3申请100K、作业2释放60K、作业4申请200K、作业3释放100K、作业1释放130K、作业5申请140K、作业6申请60K、作业7申请50K、作业6释放60K，请分别画图表示出使用首次适应算法和最佳适应算法进行内存分配和回收后内存的实际使用情况。

本章课程结束！谢谢大家！