第四章 存储器管理

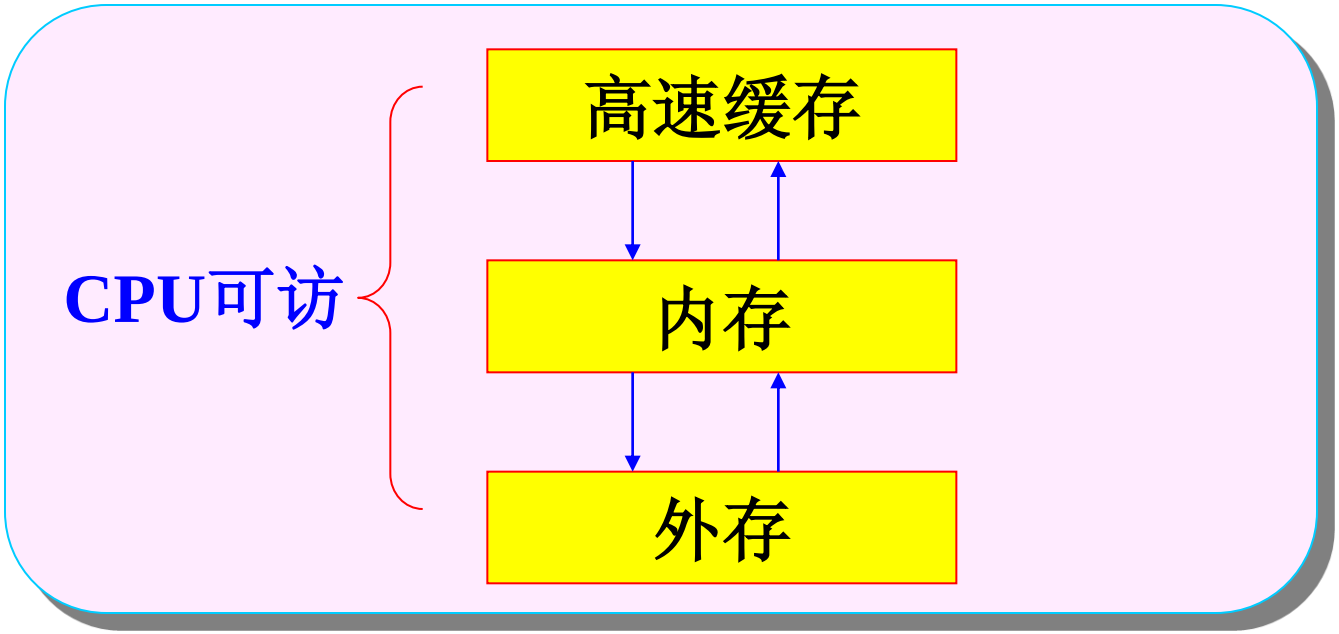
4.1 存储器的层次结构
4.2 程序的装入和链接
4.3 连续分配方式
4.4 对换
4.5 分页存储管理方式
4.6 分段存储管理方式

- 存储器作为计算机系统的重要组成部分，其容量虽然一直在不断扩大，但内存容量仍是计算机硬件资源中最关键而又最紧张的“瓶颈”，始终满足不了现代化软件发展的需要。
- 因此，存储器仍然是计算机系统中宝贵且紧俏的资源。
- 如何合理而有效地使用它，在很大程度上反映了 OS 的性能，并直接影响到整个计算机系统性能的发挥。

- 所以，存储器管理仍是目前研究 OS 的核心问题之一。
- 对外存的管理与对内存的管理有许多相似之处，只是两者用途不同，外存主要用来存储文件，故而对外存的管理放在文件管理一章中介绍，存储器管理讨论的对象是内存。
- 对内存的管理和有效使用是今天OS 十分重要的内容。
- OS之间最明显的区别特征之一，往往是所应用的内存管理方法不同。

4.1 存储器的层次结构

•为能更多的存放并更快地处理用户信息，目前许多计算机把存储器分为三级。



高速缓存Cache：K/M字节、高速、昂贵、易变的
内存RAM： M/G字节、中速、中等价格、易变的
外存Disk：G/T字节、低速、价廉、不易变的

- 为了便于对内存进行有效管理，通常把内存分成若干个区域。
- 即使在最简单的单用户OS中，至少也要把它分成两个区域

  ① 一个存放系统软件，如OS本身；

  ② 另一个存放用户任务。
- 在多用户OS中，为了提高系统利用率，需要将内存划分成更多区域，以便同时存放多个用户任务。
- 这就引起了内存分配问题及由此产生的其它问题。

存储器管理的主要目的和功能

1. 内存的分配和管理

① 记住每个存储区域的状态

- 哪些是已分配的，哪些是还未分配的

② 实施分配

- 在系统程序或用户提出申请时，按所需的数量进行分配，并修改相应的分配记录

③ 回收存储区域

- 接受系统或用户释放的存储区域，并相应地修改分配记录

2. 提高内存利用率

- 使多道程序能共享内存

3. “扩充”内存容量

- 利用虚拟存储器等技术为用户提供比内存更大的存储空间。

4. 存储保护

- 确保各进程都在所分配的存储区内运行，互不干扰。
- 以防止某进程由于发生错误而破坏其它进程，特别需要防止破坏系统进程。
- 这个问题必须由硬件提供保护功能，并由软件配合实现。

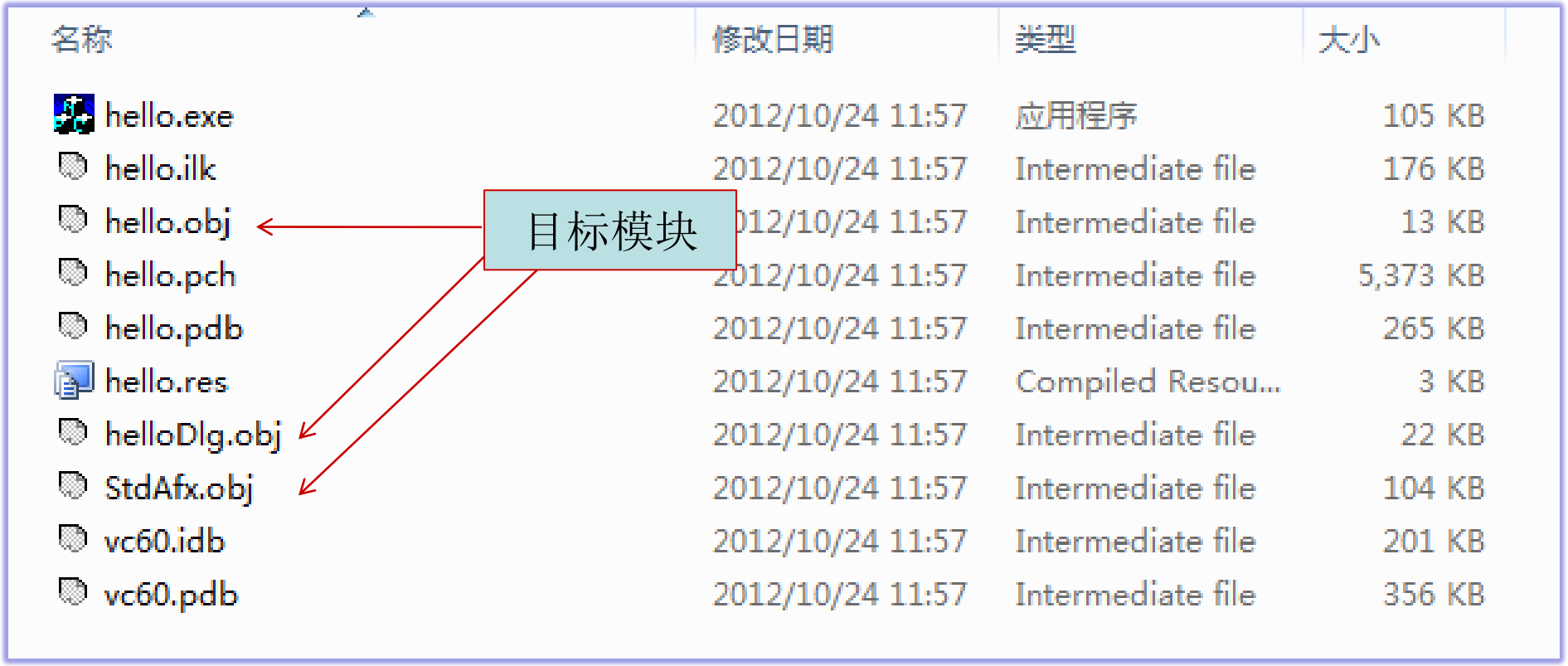
4.2 程序的装入和链接

- 在多道程序环境下，创建进程的第一件事情就是要把程序装入内存。
- 如何将一个源程序变为一个可执行程序，通常要经过下列几步：

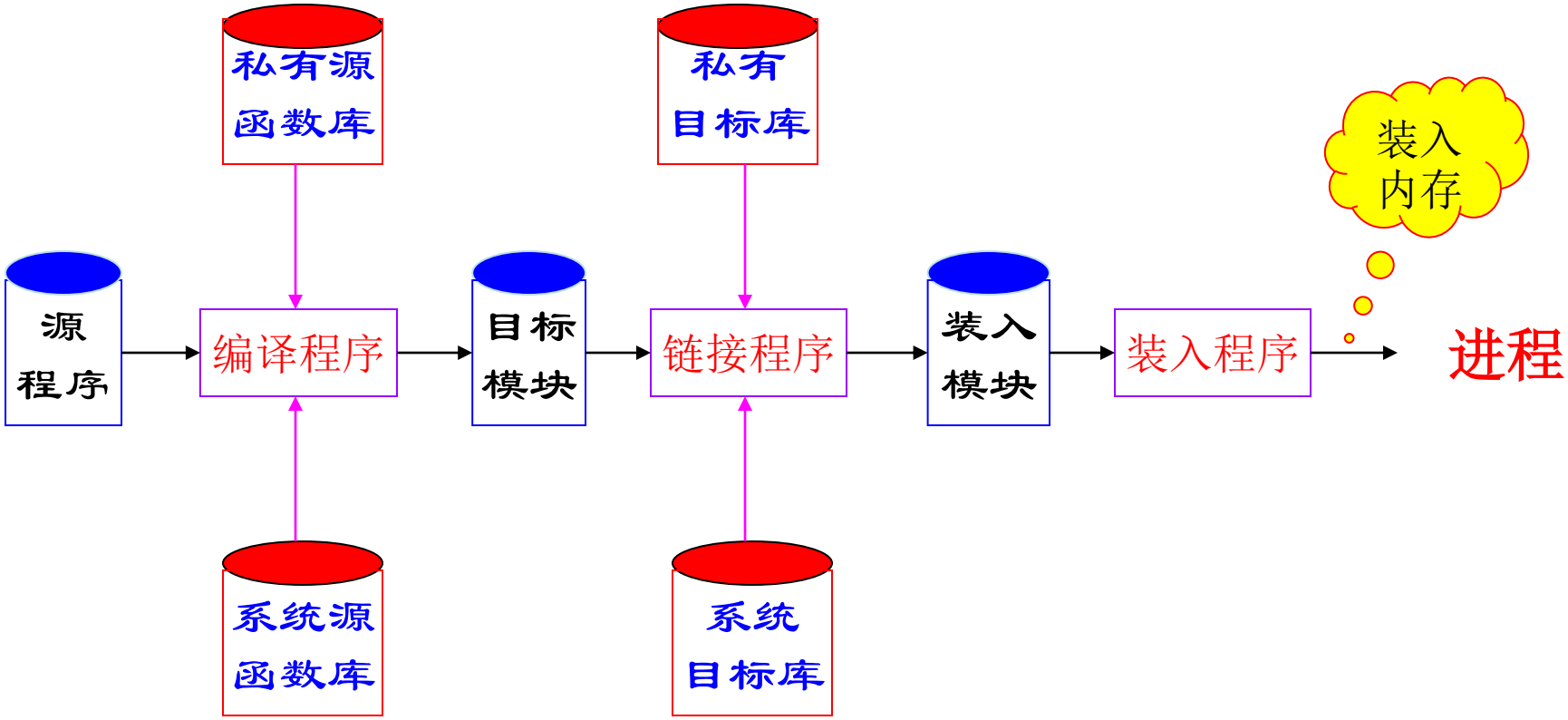
① 编译：编译程序将源程序翻译成机器代码。编译程序接受源程序，生成目标模块

② 链接：目标模块是纯二进制机器代码，但通常不能直接执行。首先必须将目标模块们链接成装入模块，才能执行

③ 装入：由装入程序将装入模块装入内存并执行


名称
修改日期
类型
大小
hello.exe 2012/10/24 11:57 应用程序 105 KB
hello.ilk 2012/10/24 11:57 Intermediate file 176 KB
hello.obj 012/10/24 11:57 Intermediate file 13 KB
hello.pch 2012/10/24 11:57 Intermediate file 5,373 KB
hello.pdb 2012/10/24 11:57 Intermediate file 265 KB
hello.res 2012/10/24 11:57 Compiled Resou... 3 KB
helloDlg.obj 2012/10/24 11:57 Intermediate file 22 KB
StdAfx.obj 2012/10/24 11:57 Intermediate file 104 KB
vc60.idb 2012/10/24 11:57 Intermediate file 201 KB
vc60.pdb 2012/10/24 11:57 Intermediate file 356 KB
目标模块

VC中的目标模块


私有源
函数库
私有
目标库
装入
内存
源
程序
编译程序
目标
模块
链接程序
装入
模块
装入程序
进程
系统源
函数库
系统
目标库

对用户程序的处理步骤

4.2.1 程序的装入

- 根据存储空间的分配方式，将一个装入模块装入内存时，可采用3种方式

① 绝对装入方式

② 可重定位装入方式

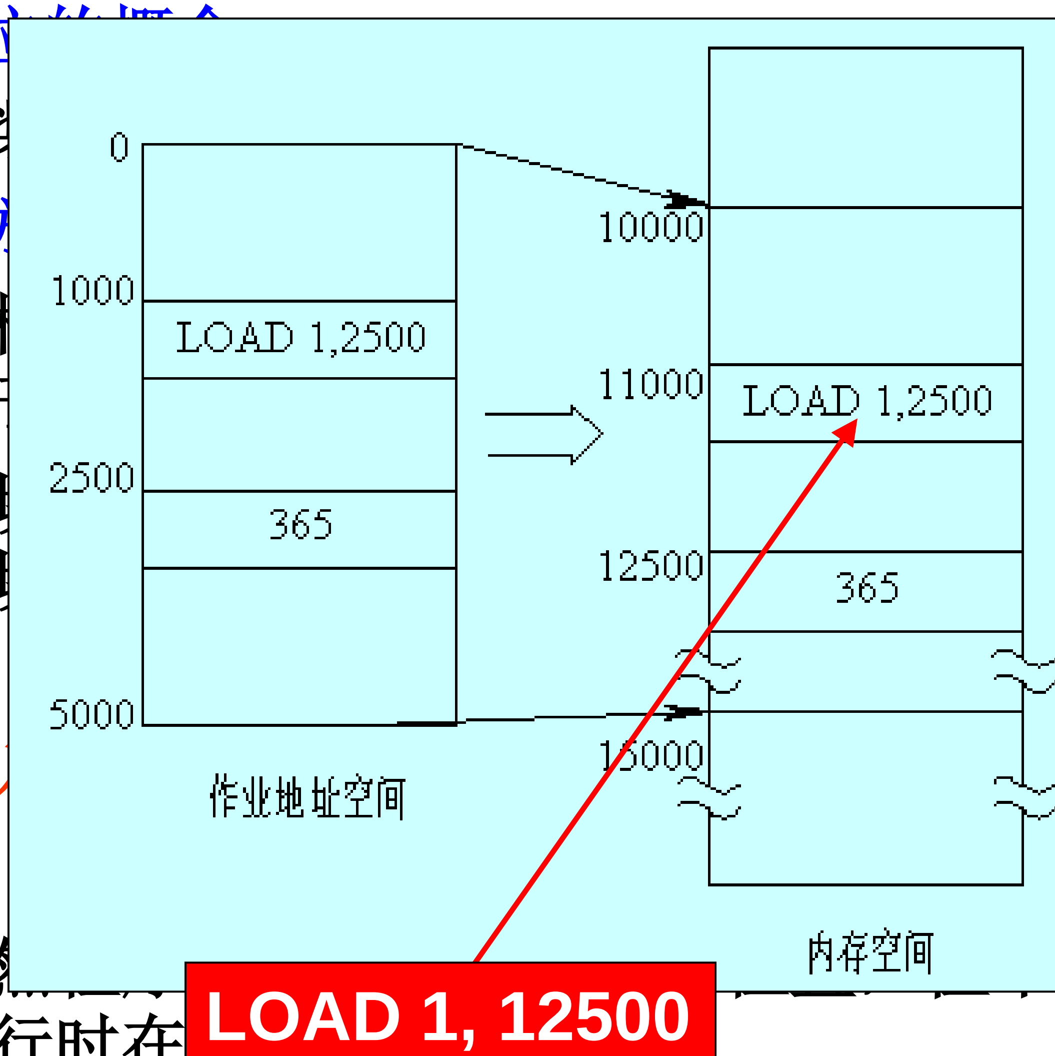
③ 动态运行时装入方式

① 绝对装入方式

- 程序中使用绝对地址，既可在编译时给出，也可由程序员直接赋予。
- 由程序员直接给出时，要求程序员熟悉内存的使用情况。一旦程序或数据被修改后，可能要改变程序中的所有地址。
- 因此，通常是在程序中采用符号地址(变量)，然后在编译时，再将这些符号地址转换为绝对地址。
- 绝对装入只能将目标模块装入指定的内存位置，多道程序下不可能知道它被放在什么地方执行，所以绝对装入只适用于单道程序环境下

② 可重定位装入方式（静态重定位）

- 重定位
  - 在装 … 改的过程
- 基本方
  - 目标 … 地址都是相对于
  - 模块 … 置，修改模块
- 特点
  - 装入
- 缺点
  - 虽 … 允许程序运行时在



③ 动态运行时装入方式

- 基本方法
  - 装入模块装入内存后，并不把其中的相对地址转换为绝对地址
  - 而是在执行代码时，再进行地址转换
  - 因此， 装入内存后的所有地址仍是相对地址
- 硬件支持
  - 由于需要经常修改地址，为不影响速度，需重定位寄存器的支持

4.2.2 程序的链接

- 功能
  - 将编译后所得到的目标模块以及它们所需要的库函数，装配成一个完整的装入模块。
- 链接的方法

① 静态链接

② 装入时动态链接

③ 运行时动态链接

① 静态链接

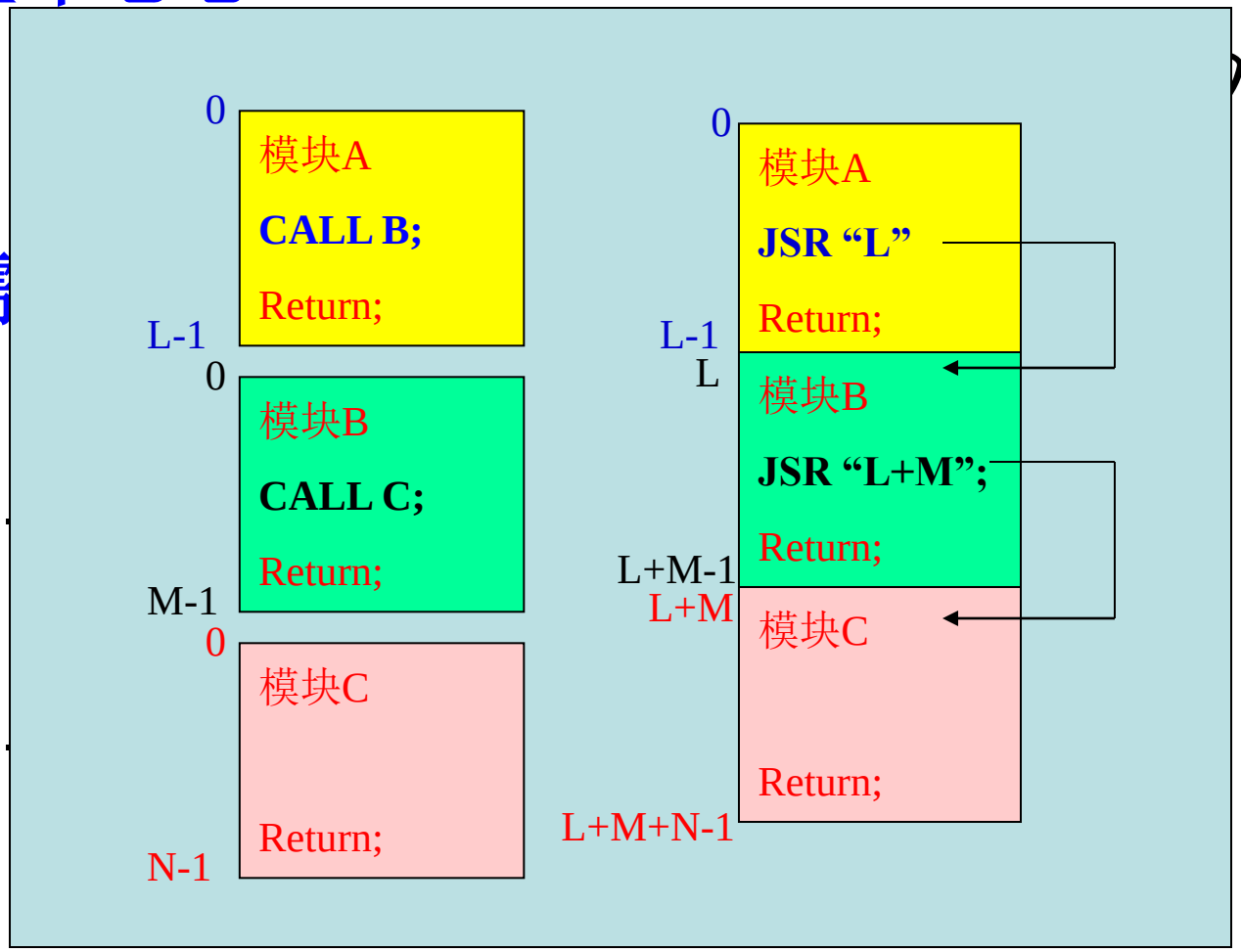
- 基本思想

  – 入模块，

- 需

  ①

  ②



② 装入时动态链接

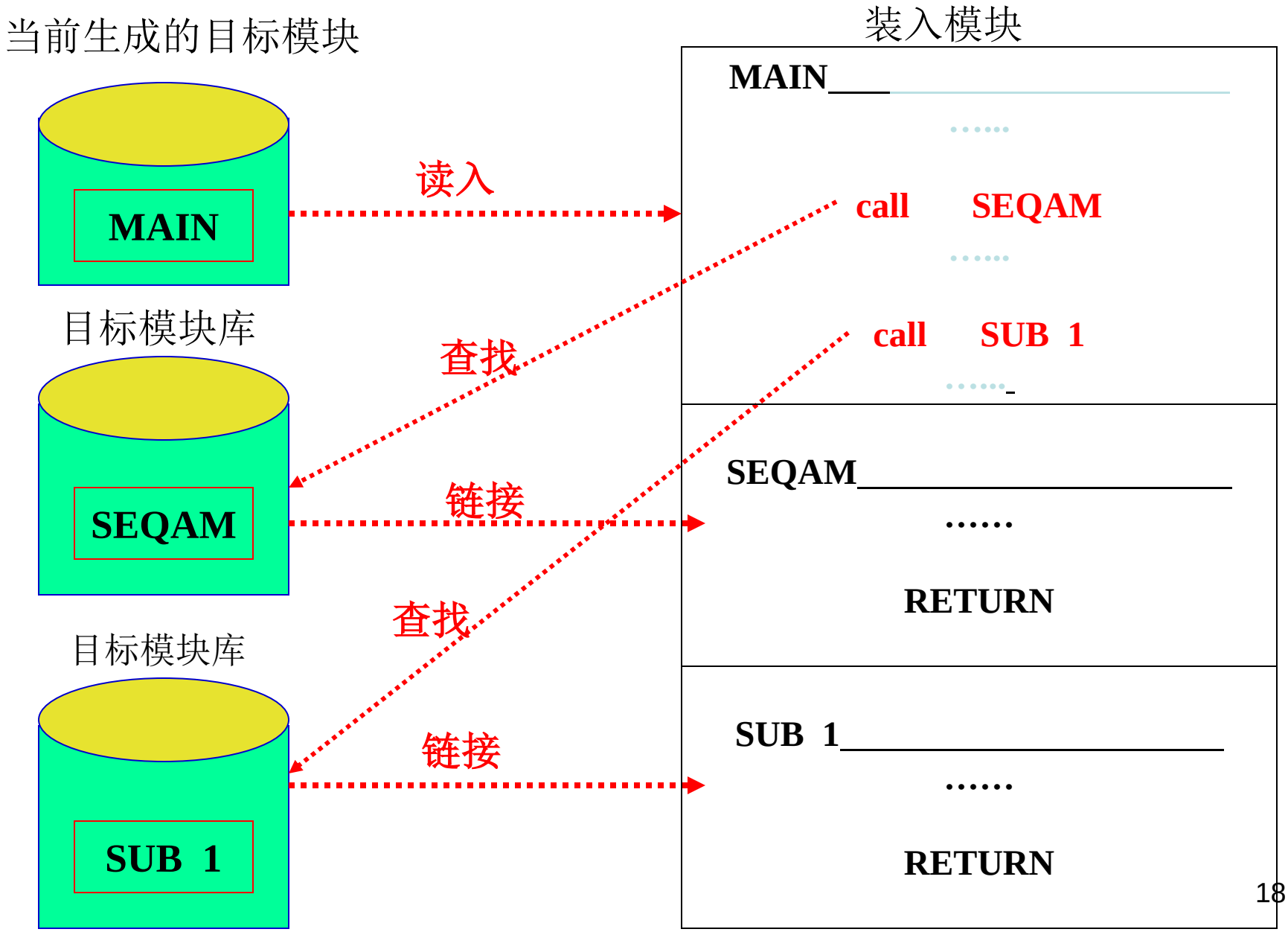
- 基本思想
  - 目标模块在装入内存时，边装入，边链接
  - 即在装入一个目标模块时，若发现需要另一个目标模块时，装入程序便去找出相应的目标模块，并将之装入内存
  - 同时，按照“静态链接”的方法，修改目标模块中的相对地址
- 优点

① 便于修改和更新

- 目标模块是分开的，修改时不必将装入模块拆开，非常方便

② 便于实现共享目标模块

- 一个目标模块可链接到多个装入模块中，从而实现多个程序对该模块的共享


当前生成的目标模块
装入模块
MAIN
读入
目标模块库
SEQAM
查找
链接
目标模块库
SUB 1
查找
链接
MAIN
call SEQAM
call SUB 1
SEQAM
RETURN
SUB 1
RETURN

– 缺点

① 装入模块的结构仍是静态的，表现在：

①进程（程序）在整个执行期间，装入模块是不改变的

②每次运行时的装入模块是相同的

② 事先无法知道本次要运行哪些模块，只能将所有可能要运行的模块全部链接在一起，而实际上往往有些目标模块根本不会运行

③ 运行时动态链接

- 近几年流行，针对“装入时动态链接”的一种改进
- 基本思想
  - 执行过程中，发现一个需要的目标模块尚未装入内存，立即由OS将它装入内存，并链接到调用者模块上
  - 因此，在执行过程中未被用到的目标模块，都不会被调入内存和链接
  - 这样，可加快装入过程，节省大量的内存空间

4.3 连续分配方式

- 连续分配的概念
  - 为 一个用户程序分配一个连续的内存空间
- 广泛应用于60—80年代的OS中，至今仍有使用
- 方式

① 单一连续分配

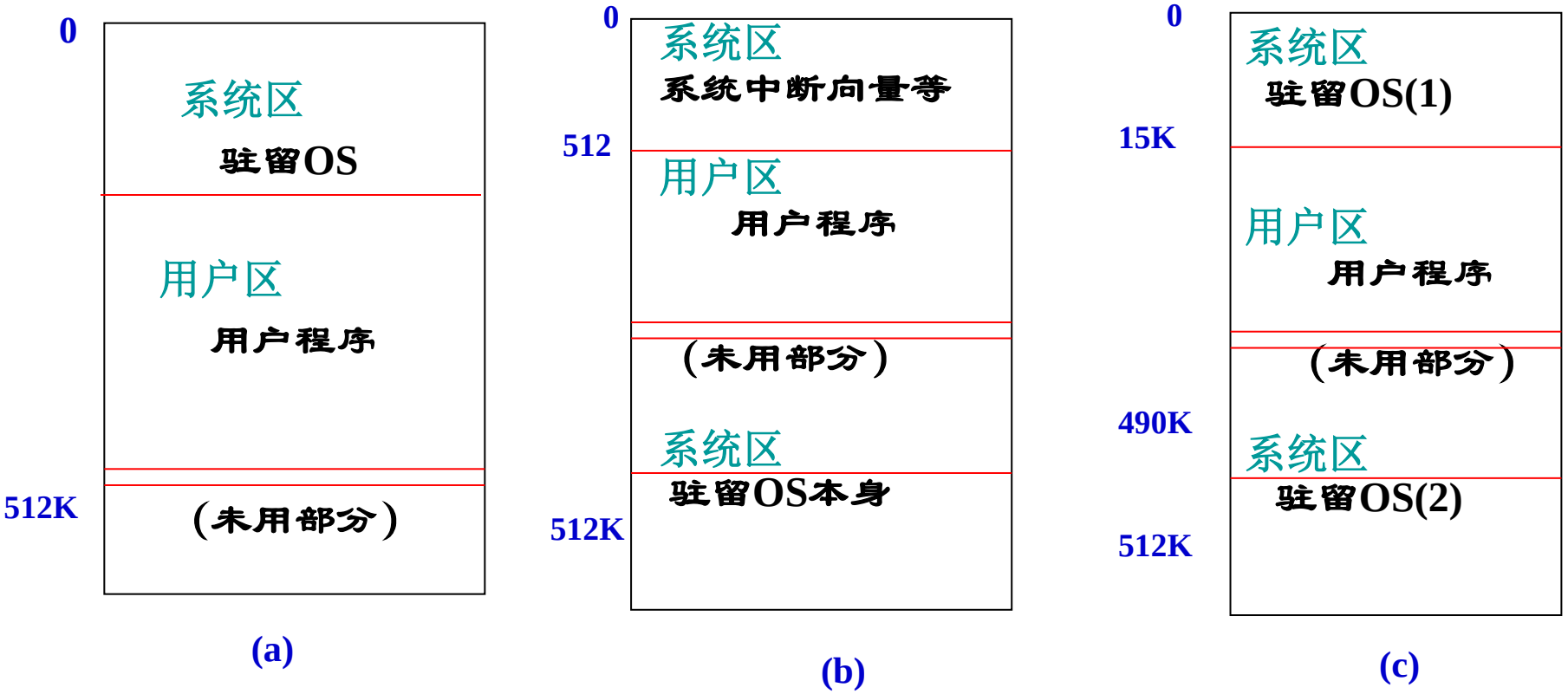
② 固定分区分配

③ 动态分区分配

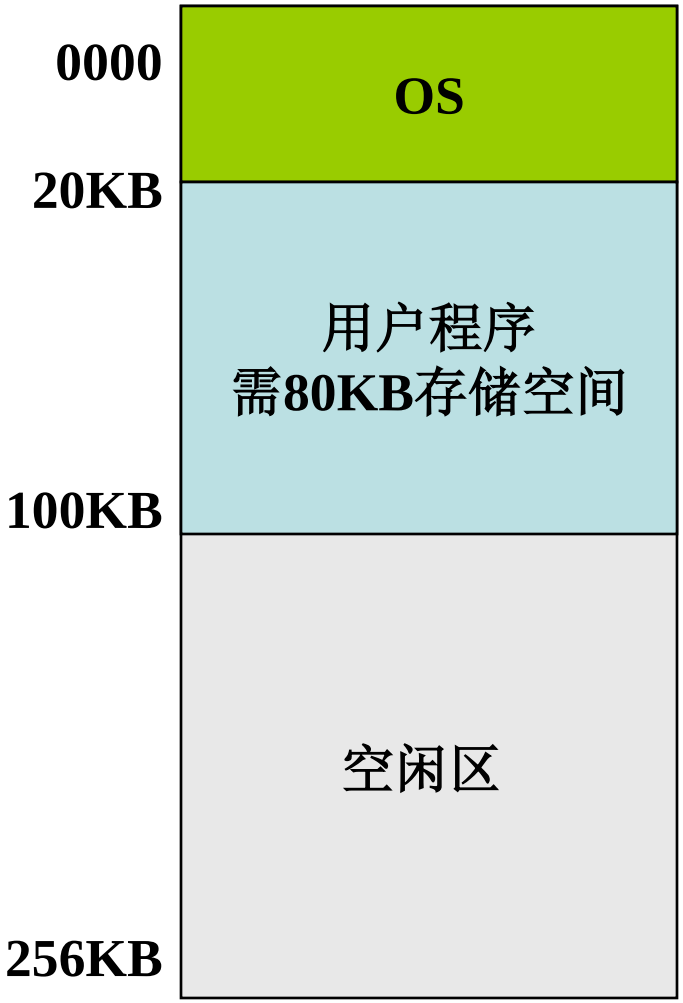
④ 动态可重定位分区分配

4.3.1 单一连续分配

- 基本思想
  - 把内存分为系统区和用户区两部分
    - ① 系统区
      - 仅提供给OS使用
    - ② 用户区
      - 除系统区以外的全部内存空间，供用户使用



- OS既可放在内存的低址部分，如图(a)；也可放在内存的高址部分，如图(b)；甚至可以放在内存的两端，如图(c)
- 通常放在低址部分



- 每次只允许装入一个程序

- **单一连续分配的特点**

  ① 把整个用户区分配给一个程序使用

  ② 实际上，用户区又被分为“使用区”和“空闲区”

  ③ 由于任何时刻用户区中只有一个程序运行，因此只适用于单道OS

- **单一连续分配的缺点**

  ① 由于每次只能有一个程序进入内存，故整个系统的工作效率不高，资源利用率低

  ② 若程序比用户区大，那么它就无法运行。即大任务无法在小内存上运行

4.3.2 固定分区分配

- 实现多道系统存储器管理的一个最早的想法
- 基本思想
  - 把用户区划分成若干个大小固定的分区
  - 每个分区只放一个进程
  - 因此，有几个分区就允许几个进程并发
  - 当一个分区空闲时，可选择一个新的进程进入那里运行
  - 由于这些区域是在系统启动时划定的，在进程运行过程中，所获得的区域是不能改变的

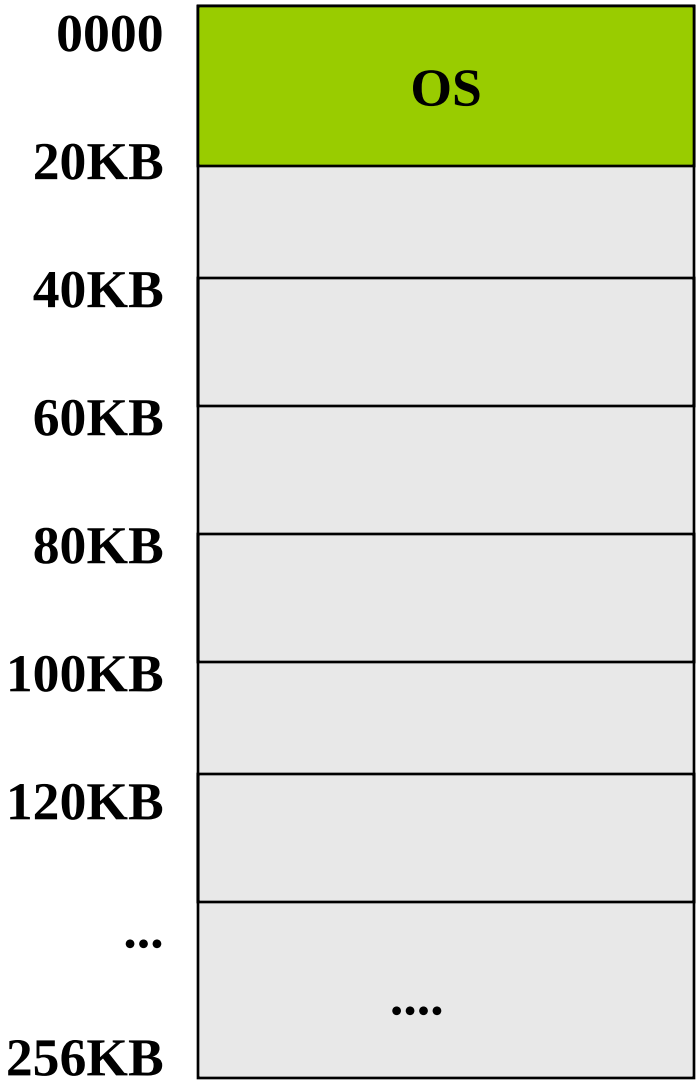
- **划分固定分区的方法**

① **分区大小相等**

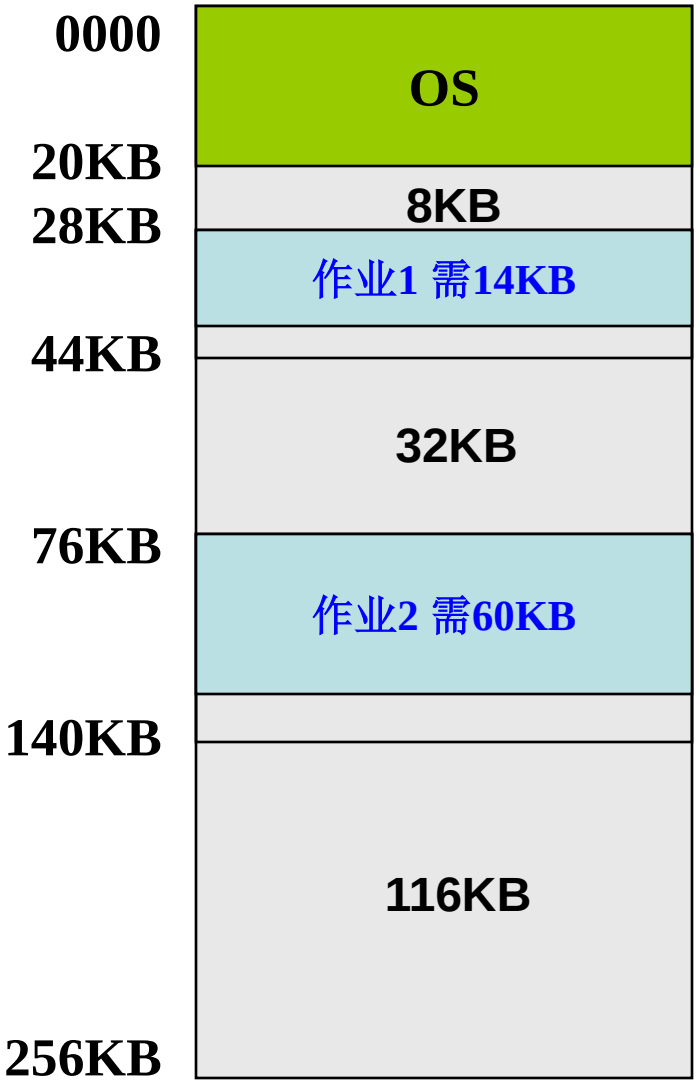
- **方法：**所有的分区都大小相等
- **缺点：**缺乏灵活性，程序太小时，内存浪费；太大时，又无法装入
- **应用：**控制多个相同的对象 (如工业控制机床、锅炉等)，因为它们所需的内存空间是大小相等的

② **分区大小不等**

- **目的：**克服“分区大小相等”的缺点
- **方法：**把内存划分成多个较小分区、适量中等分区和少量大分区



分区大小相等

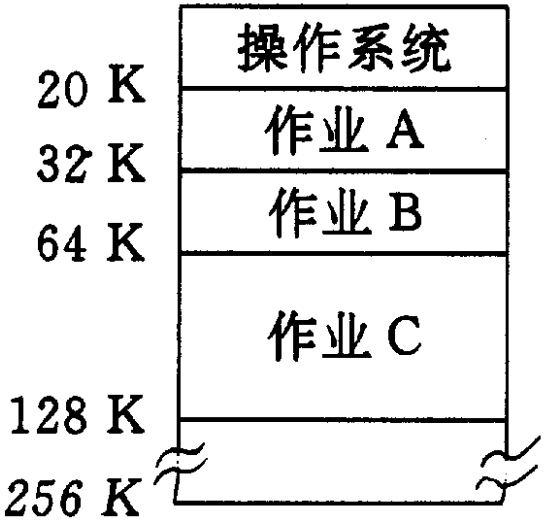


分区大小不等

- 为实现固定分区分配，通常将分区按大小排列，并在OS中为之建立一张分区说明表，指出可用于分配的分区数以及每个分区的大小、起始地址及状态。

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

4.3.3 动态分区分配

- 又称可变分区分配
- 根据进程的实际需要，动态分配内存空间
- 实现动态分区分配，必须解决3个问题

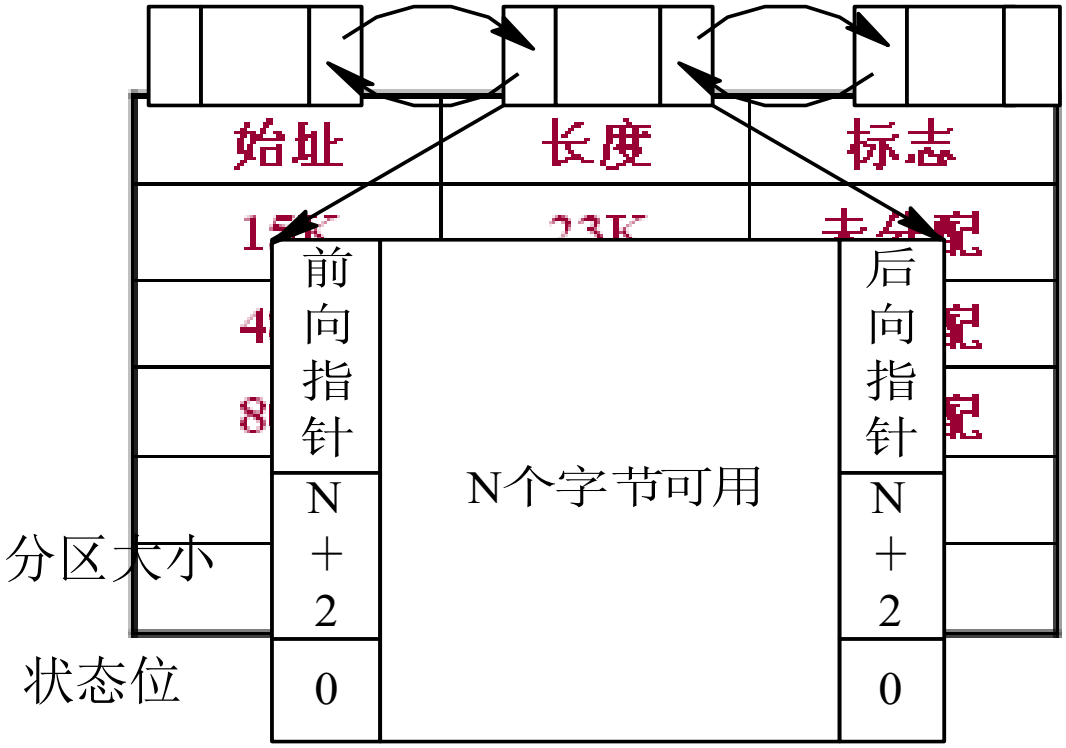
① 数据结构

② 分区分配算法

③ 分区的分配和回收

1. 数据结构

- 空闲分区表
  - 设置一张空闲分区表，用于记录每个空闲分区的情况
- 空闲分区链
  - 将所有空闲分区链接成一个双向链表



2. 分区分配算法

- 系统运行一段时间后，整个内存将出现许多大小不等的区域，有的被进程占用，有的则因进程退出系统而成为空闲分区
- 现在假设有一个新进程需进入内存，如何为其选择一个合适的分区？
- 基于顺序搜索的思想，有4种分配算法：

① 首次适应算法(FF)

② 循环首次适应算法(NF)

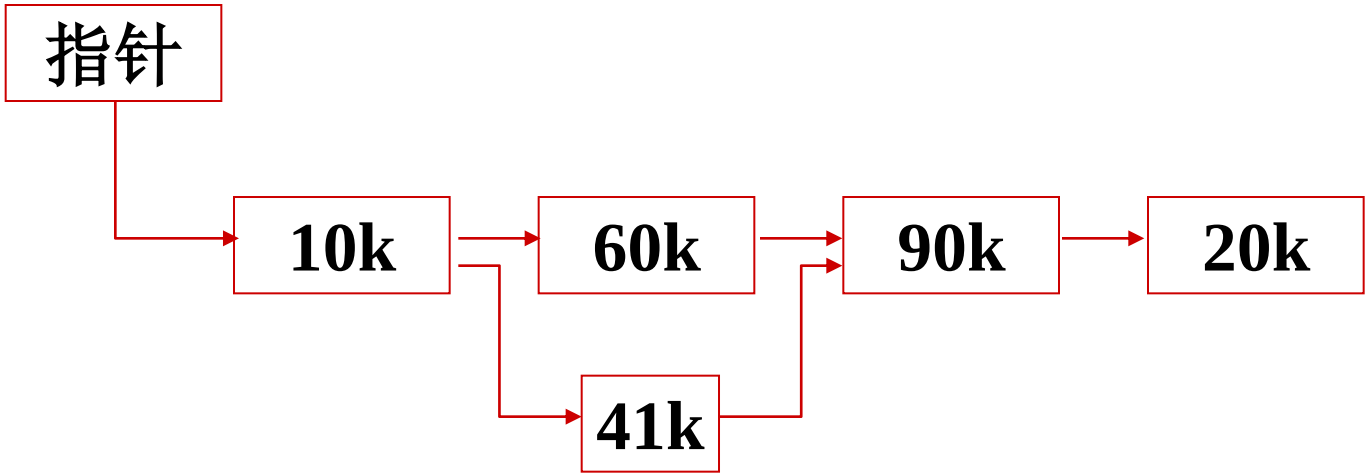
③ 最佳适应算法(BF)

④ 最坏适应算法(WF)

① 首次适应算法(FF)

- 基本思想（以空闲分区链为例）
  - 以地址递增次序链接空闲分区
  - 从链首开始查找，找到第一个满足请求大小的分区，从中划出请求大小的空间，余下的仍留在链中（一般从低地址开始划分）
- 优点
  - 保留了大分区
- 缺点
  - 低地址不断被划分，会留下很多难以利用的小空闲分区（碎片），下次查找会浪费时间

例：



- 有4块空闲分区(从低地址→高地址)
- 来了一个需要19K内存的新进程后的情况？

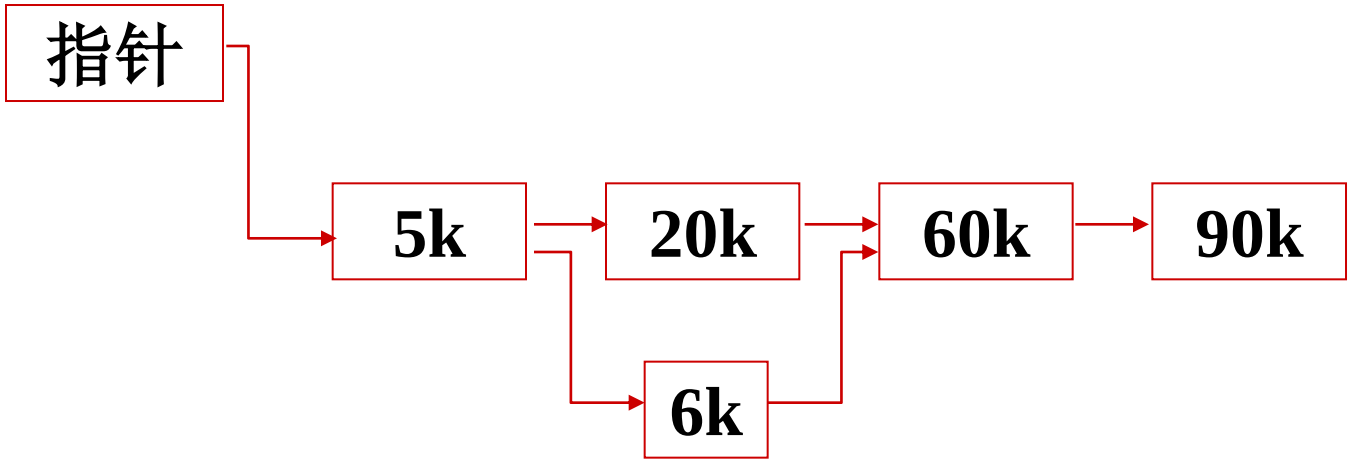
② 循环首次适应算法(NF)

- 由FF算法演变而成
- 基本思想
  - 从上次找到的空闲分区的下一个空闲分区查找
  - 需设置起始查找指针，以指示下次从哪开始查找
- 优点
  - 使空闲分区均匀分布
- 缺点
  - 缺乏大的空闲分区

③ 最佳适应算法(BF)

- 基本思想
  - 找能满足分配的最小空闲分区
  - 这要求分区按容量递增顺序排列
- 优点
  - 避免“大材小用”
- 缺点
  - 每次分割下的空闲分区总是最小的，难以利用（碎片）

例：



- 有4块空闲分区(从小分区→大分区)
- 来了一个需要14K内存的新进程后的情况？

④ 最坏适应算法(WF)

- 基本思想
  - 与最佳适应算法相反
  - 找能满足分配的最大空闲分区
  - 这要求分区按容量递减顺序排列
- 优点
  - 产生碎片的可能性最小，有利于中小进程
  - 查找效率高
- 缺点
  - 缺乏大空闲区

3. 分区的分配和回收

- 涉及动态分区的主要操作有分配内存和回收内存

① 分配内存

设进程请求的分区大小： **u.size**

每个空闲分区大小： **m.size**

最小不再分割大小： **size**

if m.size – u.size <= size

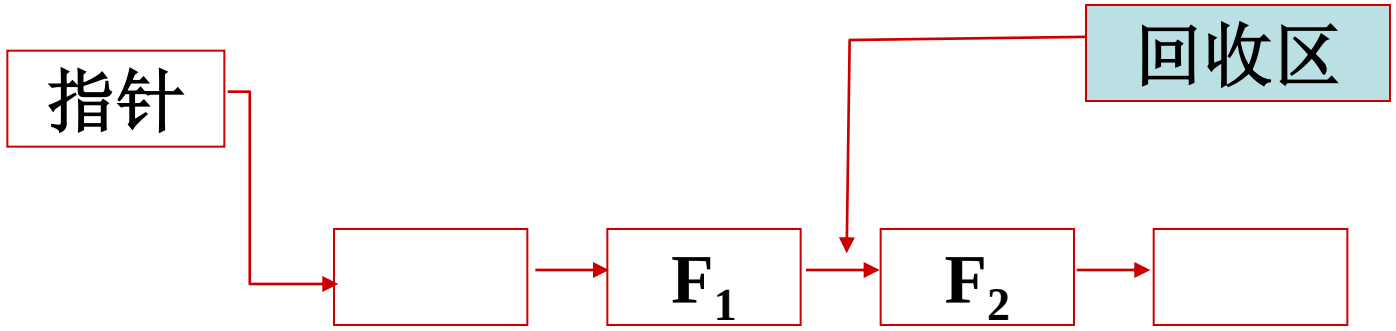
说明剩下的太小了，可不再分割，把整个分区都分配

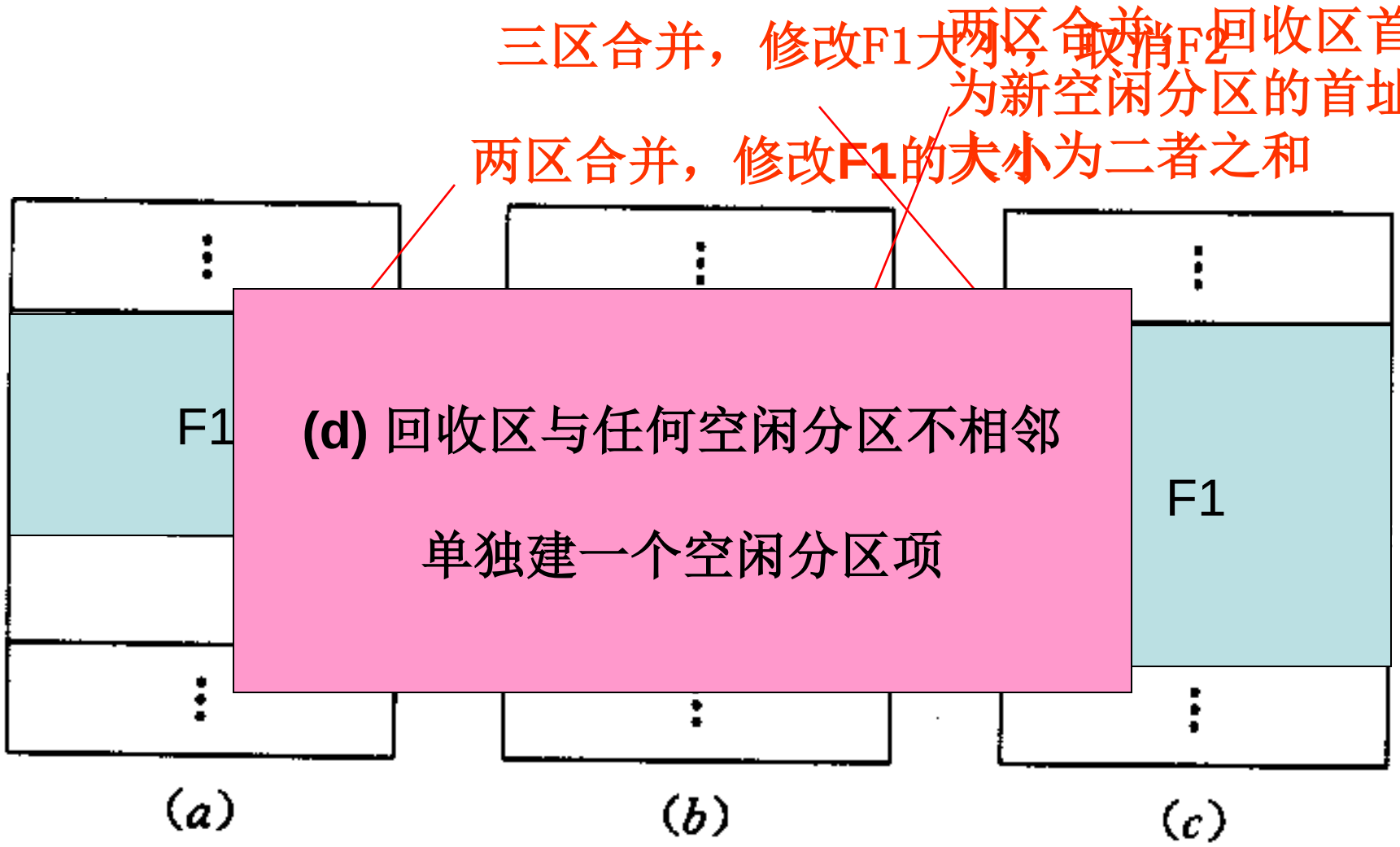
else

从空闲分区中把**u.size**大小的空间分配出去，余下的仍留在空闲分区表(链)里

② 回收内存

- 当一个进程终止时，OS必须回收它所占用且未释放的所有空间，以便供其他请求者使用
- 一个回收的分区在插入到空闲链 (表)中时，与其他空闲分区存在四种邻接情况
  - ① 回收区与插入点的前一个空闲分区F_1相邻接
  - ② 回收区与插入点的后一个空闲分区F_2相邻接
  - ③ 回收区同时与插入点的前、后两个分区相邻接
  - ④ 回收区既不与F_1邻接，也不与F_2邻接

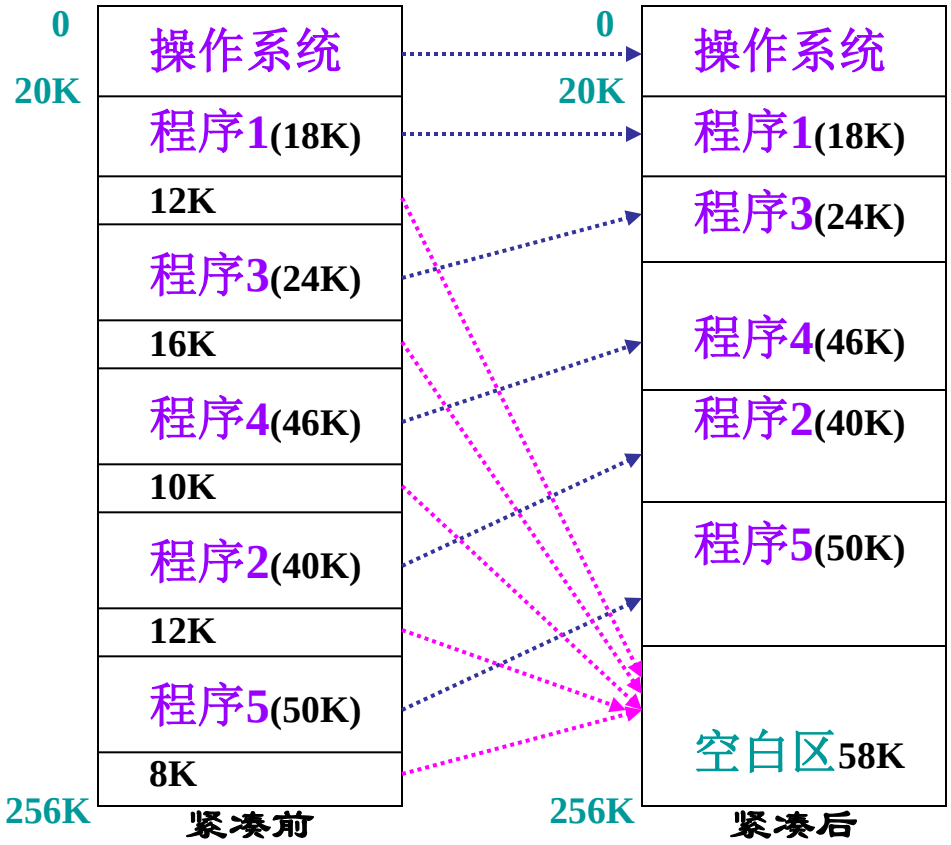




内存回收时，如何修改空闲链(表)中的信息？

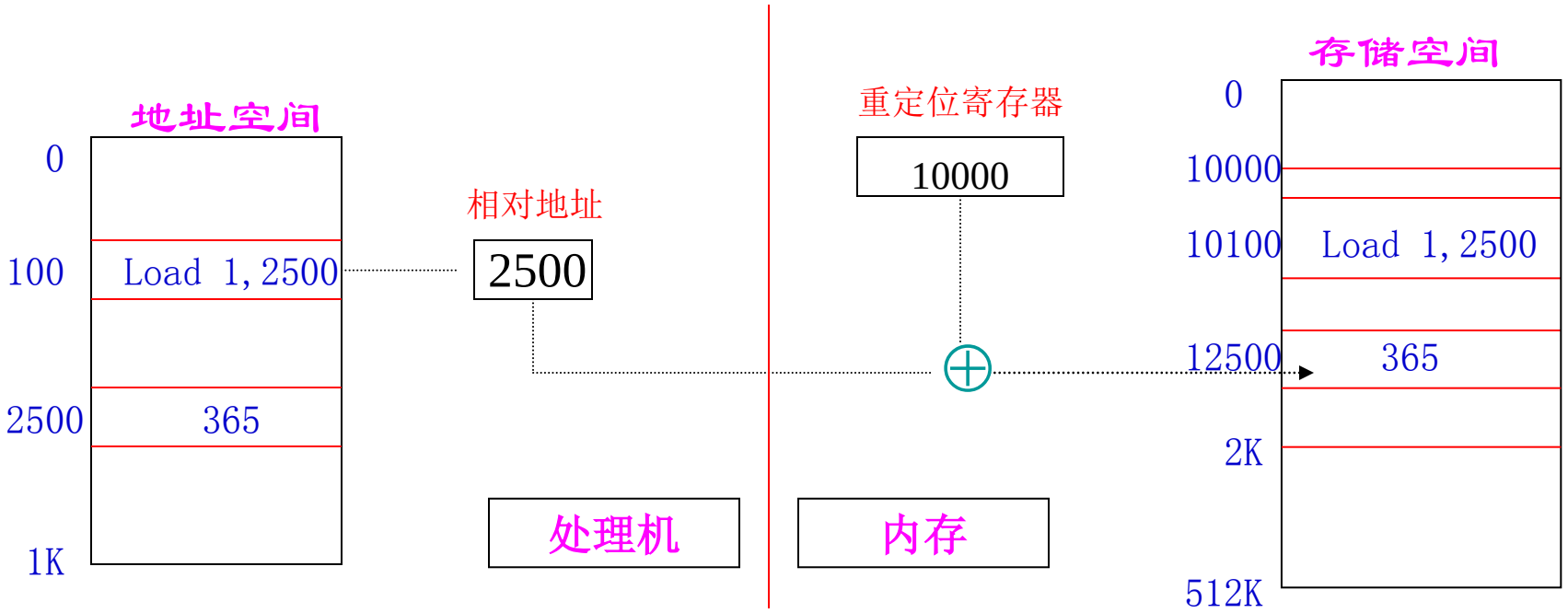
4.3.6 动态可重定位分区分配

- “连续分配方式”存在的主要问题
  - 程序必须装入连续的空间中，如果系统只剩若干小分区，即使其总和大于程序，也无法分配
  - 这些难以利用的小分区称为碎片
- 如何解决这个问题？
  - 一种解决方法：紧凑(拼接)
  - 原理：将各程序移动位置并拼凑到一起，从而使小分区组成一个大分区，供新程序使用



- 把一个程序从一个存储区域移动到另一个存储区域所发生的问题，正如把一个程序装入到与它地址空间不 一致的存储空间所引起的问题一样，
- 需要对程序中涉及的地址进行修改

- 动态重定位技术
  - 一个较实用且可行的办法是采用动态重定位技术
  - 程序在内存中仍使用相对地址，通过重定位寄存器计算绝对地址
  - 程序移动后，只要改变重定位寄存器中的内容即可



4.4 对换

- 在早期，为了克服内存不足而采取的另一种措施，称为对换技术。
- 提高内存利用率的有效措施
- 早期的UNIX、Windows中都采用这种技术
- 广义地说，所谓对换就是
  - 把暂时不能运行的进程或暂时不用的程序和数据移到外存上，以便腾出必要的空间；
  - 接着把已具备运行条件的进程或进程所需的程序和数据换入内存

- 整体对换(进程对换)
  - 以整个进程为单位
  - 广泛应用于多道程序系统，并作为中级调度
  - 目的
    - 解决内存紧张问题，提高内存利用率和系统吞吐量
- 部分对换(页面对换、分段对换)
  - 以“页”、“段”为单位
  - 目的
    - 支持虚拟存储

- 为了实现进程对换，OS需提供以下三个方面的功能：

  ① 对换空间的管理

  ② 进程的换出

  ③ 进程的换入

① 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。
- 前者用于存放文件，由于通常的文件都是较长久地驻留在外存上，故对文件区管理的主要目标，是提高文件存储空间的利用率。为此，系统采取离散分配方式。
- 后者用于存放从内存换出的进程，由于这些进程在对换区中驻留的时间是短暂的，而对换操作又较频繁，
- 故对换空间管理的主要目标，则是提高进程的换入、换出速度。为此，所应采取的管理策略是用连续分配方式，较少考虑外存中的碎片问题。

- 数据结构
  - 为了能对对换区中的空闲盘块进行管理，在OS中应设置相应的数据结构，以记录外存的使用情况。
    - 可以设置空闲分区表或空闲分区链(类似内存的动态分区分配方式)
    - 在空闲分区表中，每个表项应包含对换区首址及其大小，其单位是盘块号和盘块数。
- 分配和回收
  - 对换区采用连续分配方式，因而也类似内存的动态分区方式

② 进程的换出

- 一般选择处于**阻塞状态**且**优先级最低**的进程作为**换出进程**，将其程序和数据存到对换区上
- 若传送过程未出现错误，便可回收它所占用的内存，并对它的PCB作相应的修改

③ 进程的换入

- OS定时查看所有进程的状态，从中找出**“就绪”但已换出的进程**，将**换出时间最久**(换出到磁盘上) 的进程作为**换入进程**，将之换入
- 直至已无可换入的进程或无可换出的进程为止

如何克服连续分配存在的问题

- 主要问题
  - 会出现很多难以利用的碎片
  - 虽然可通过紧凑来解决，但系统开销太大
- 解决方法
  - 如果允许进程在不连续的空间里运行，就无需紧凑了
  - 所以，引入“离散分配方式”
- 根据分配时所用的基本单位的不同，它分为：

① 分页存储管理方式

② 分段存储管理方式

③ 段页式存储管理方式

4.5 分页存储管理方式

4.5.1 基本方法

1. 页面和物理块（存储块）

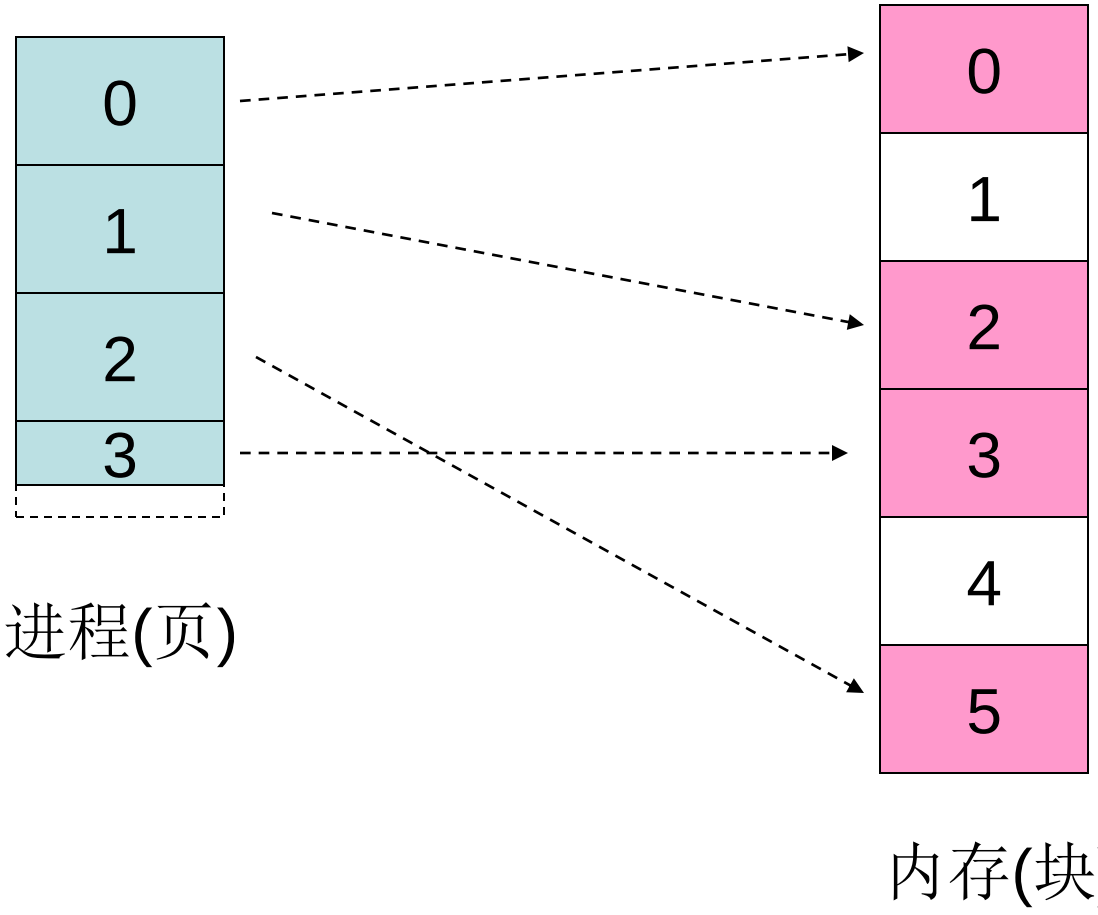
- 把进程的逻辑地址空间分成若干大小相等的片，称之为页面或页，编号从0开始
- 把内存的存储空间也分成与页相同大小的片，这些片称为(物理)块或页框，编号从0开始
- 在为进程分配空间时，总是以块为单位

- 说明

① 每一页的页内地址相对于0开始

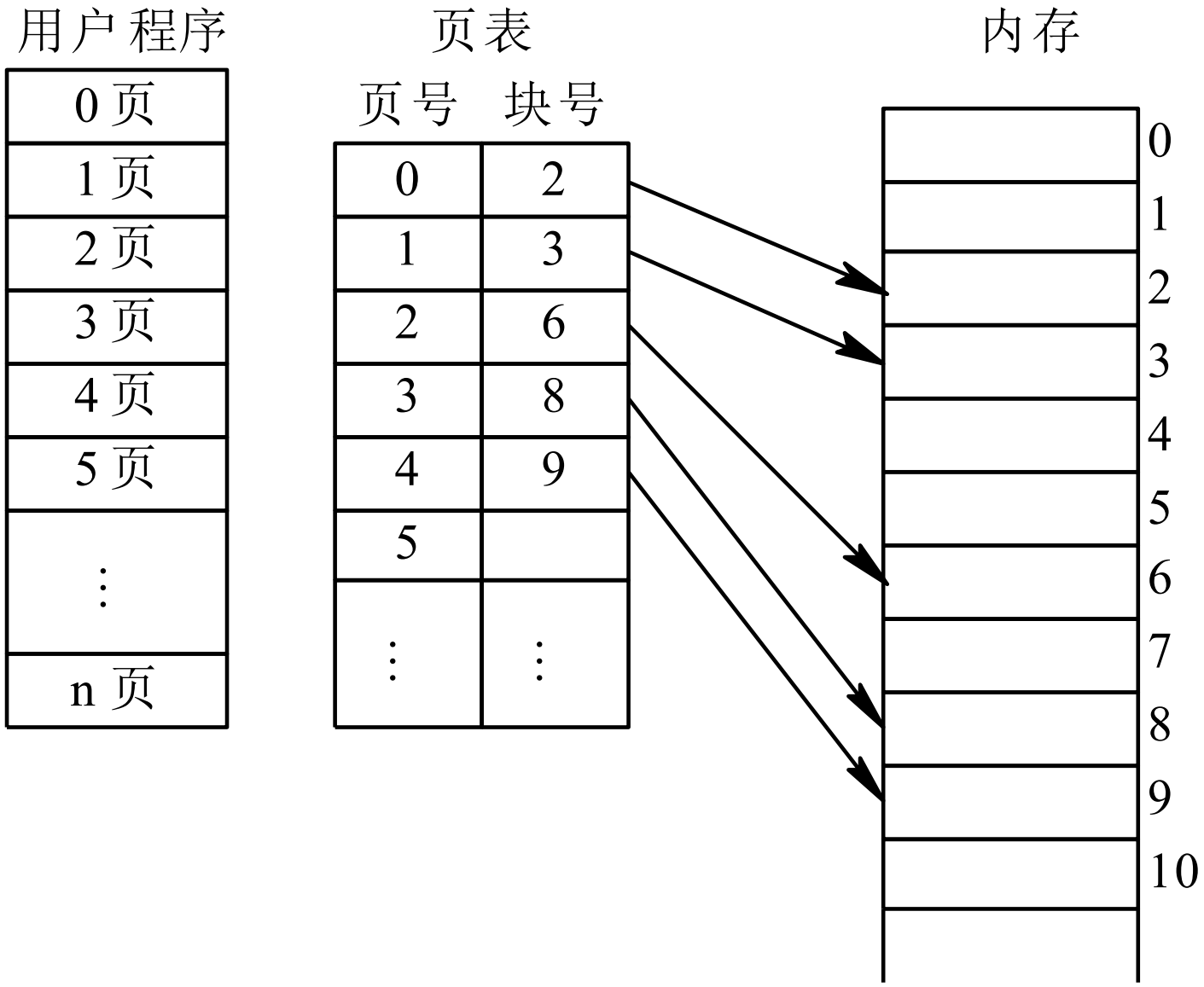
② 调度时，必须把进程的所有页一次性装入到内存的块中

- 分配内存时，以块为单位，可以将进程的若干页分别装入到不相邻的块中
- 进程的最后一页往往装不满一块，形成了不可利用的碎片，称之为“页内碎片”



2. 页表

- 在分页系统里，OS怎么知道程序的哪一页放在内存的哪一块里？
  - OS为每个进程建立一张页表
  - 页表的作用是保存页号和块号的对应关系
  - 页表在进程装入内存时，由OS建立



页表的作用：实现从页号到块号的地址映射

3. 页面大小

- 页面大小是由机器的地址结构所决定的
- 对于某一机器只能采用一种大小的页面
- 对页面的大小应选择的适当。因为：

  ① 如果页面较小，虽然可使内存碎片小，从而减少了碎片的总空间，有利于提高内存利用率；但也会使每个进程要求较多的页面，导致过长的页表占用大量内存

  ② 如果页面较大，虽然可减少页表长度，但又会使页内碎片增大

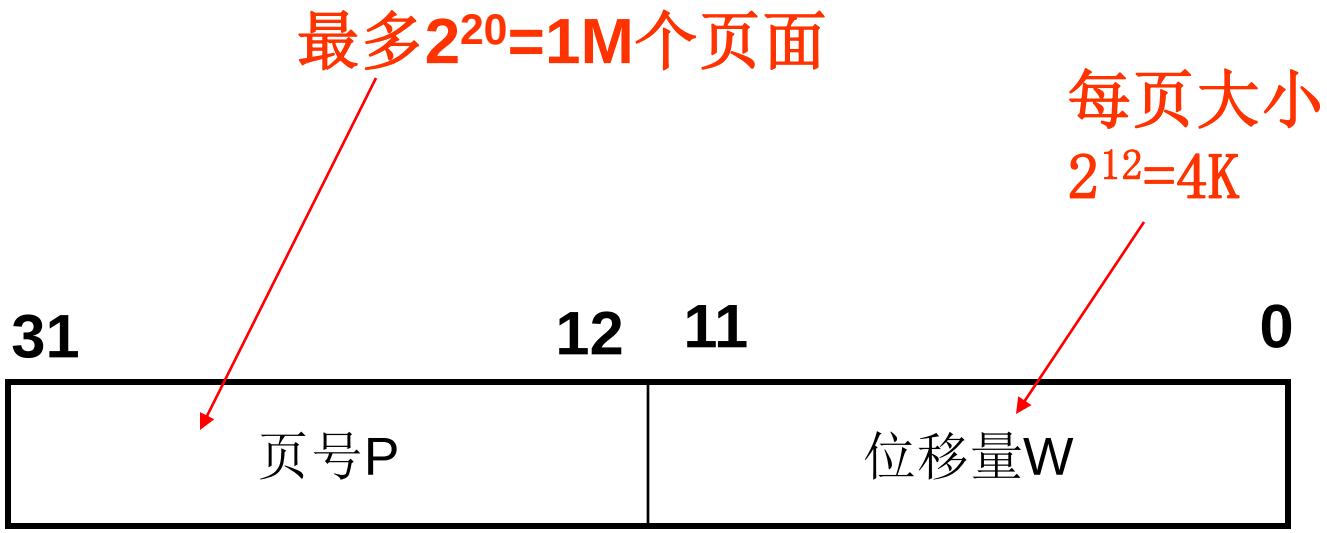
- 页面的大小通常在**1KB～8KB**，但**总是2的幂**

4. 地址结构

- 在分页系统中，由于每一页内的地址都是从0开始的，程序中的地址便由“页号”和“页内地址(位移量)”两部分组成

- 因此，在程序访问数据时，程序就要指出：
  - 该数据位于第几页，以及从该页开头算起的第几个字节的位置上

- 页号、位移量的划分由OS自动完成，对用户是透明的
- 例如
  - 某分页地址中的地址结构如下(32位地址)：

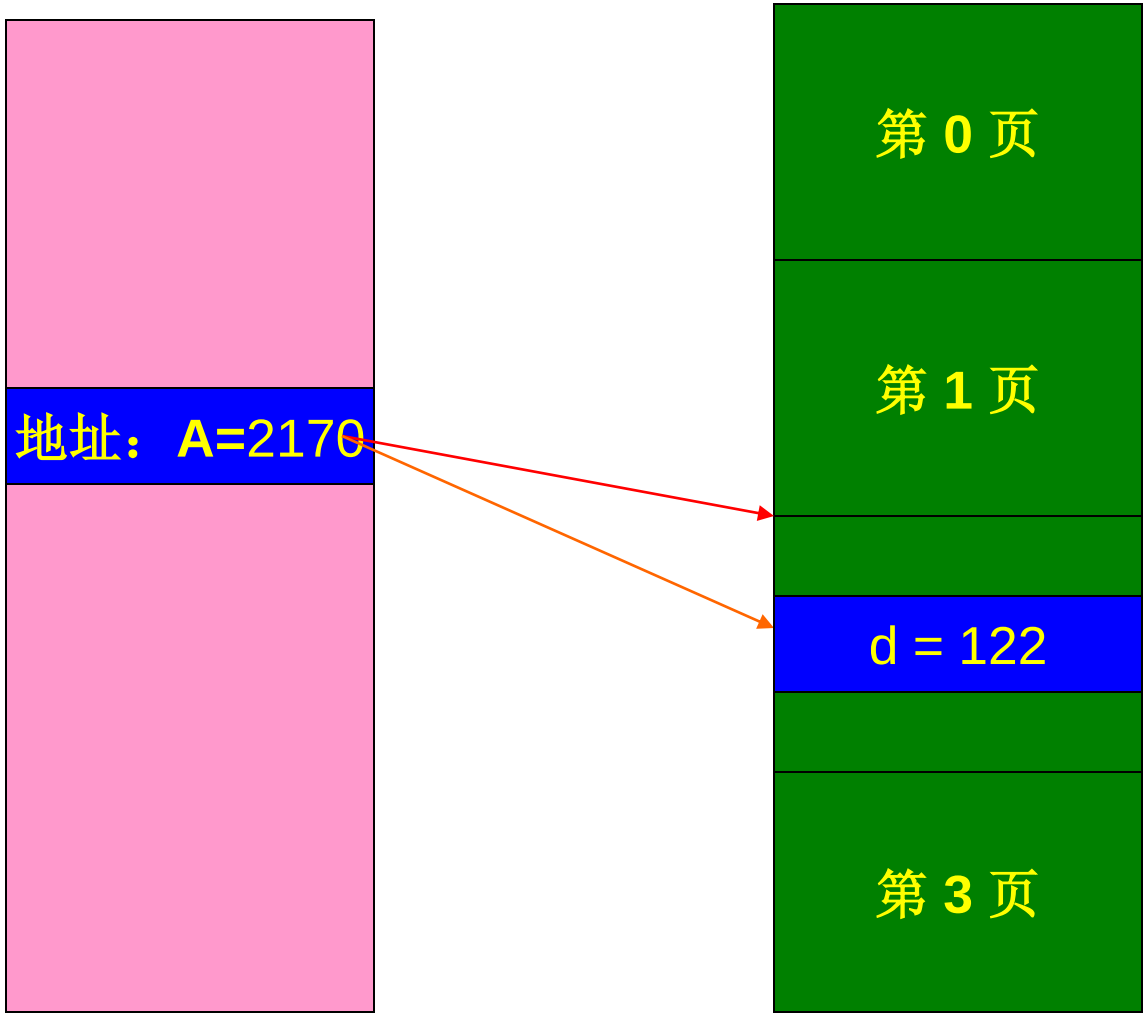


- 对某种特定机器，其地址结构是一定的
- 若给定一个逻辑地址空间中的地址**A**，页面的大小为**L**，则页号**P**和页内地址**d**可按下式求得：

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

$$d = [A]MODL$$

页大小 L=1K



在哪2页的哪个位置？

d = [A] MOD L
= 2170 MOD 1K
= 2170 MOD 1024
= 122

• 例：设每一页大小为1K字节，逻辑地址为$(1DE41)_{16}$，问该地址对应的页号和偏移量各是多少

页的大小：$1K=2^{10}$

所以，页内偏移量占10个比特

$(1DE41)_{16}$ ＝

1D 1110 41

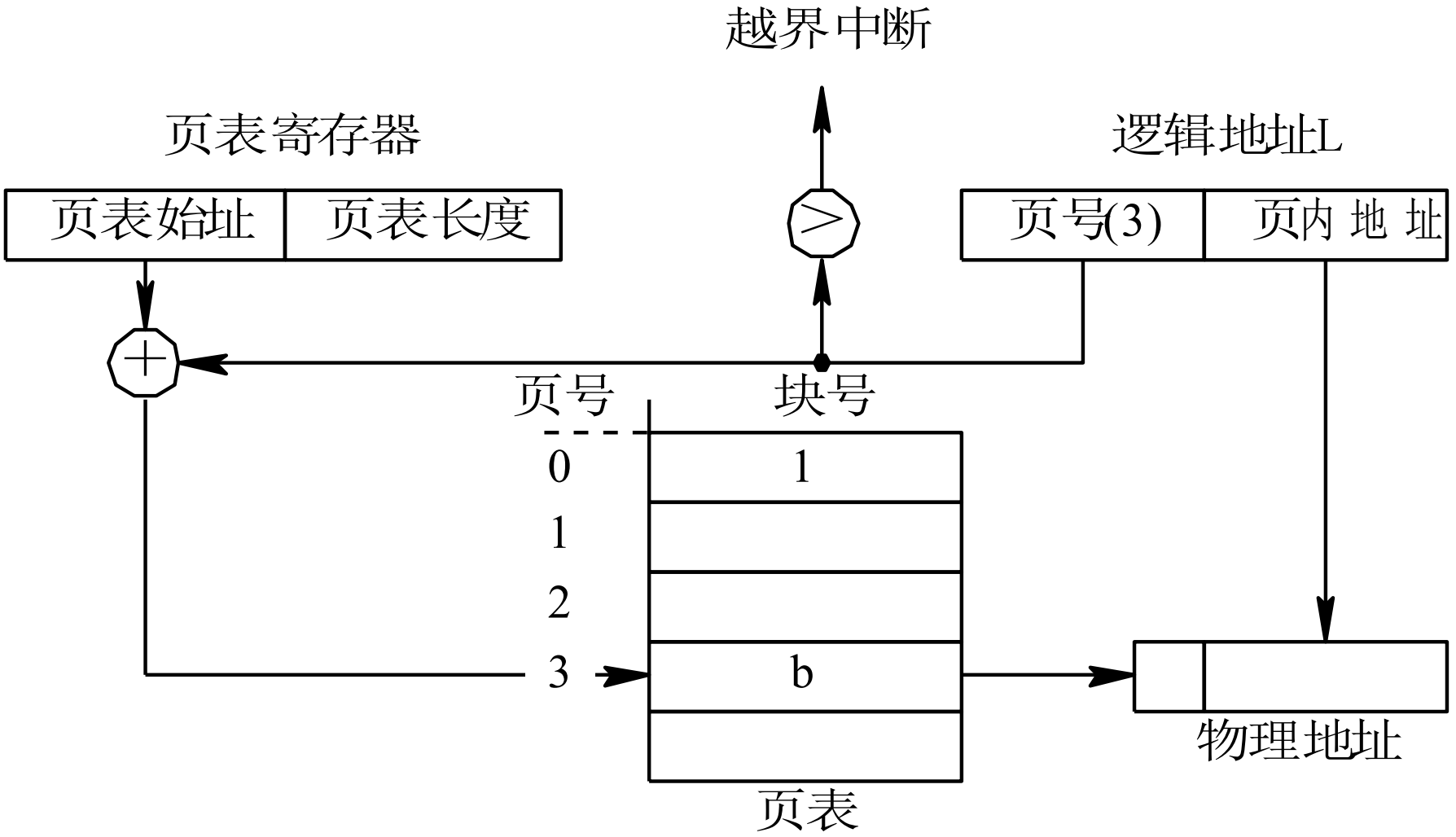
$(111,0111)_2$ $(10,0100,0001)_2$

所以，页号为$(77)_{16}$， 位移量为$(241)_{16}$

4.5.2 地址变换机构

1. 基本的地址变换机构

- 用于解决“如何将逻辑地址转变为物理地址”的问题
- 逻辑地址转变为物理地址称为 “重定位”
- 基本任务
  - 借助页表，将页号转变为物理块号
- 页表的功能本应由一组专门的寄存器实现，但造价太高
- 所以，页表通常保存在系统区里
- 页表的首地址和长度(即有多少个表项)则保存在进程的PCB中
- 进程被调度执行时，这两项便被取出，并调入页表寄存器(PTR)中



分页系统的地址变换机构

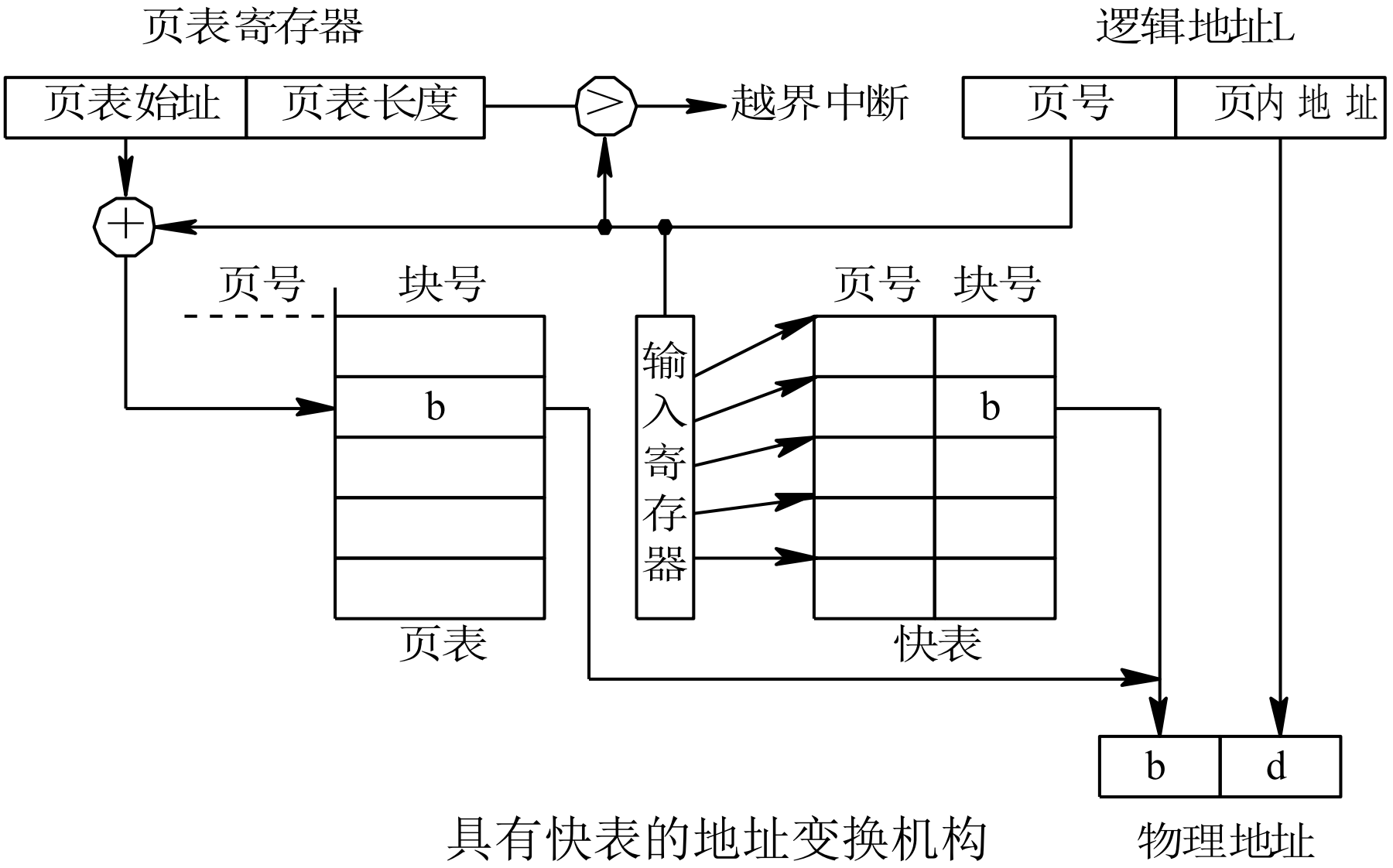
2. 具有快表的地址变换机构

- 因为页表存储在内存中，这使得每次访问一个数据，必须访问两次内存

  ① 一次是访问页表：确定数据所在的物理地址

  ② 另一次才是访问真正的数据

- 这使得访问内存的次数加倍，导致系统总的处理速度明显下降
- 解决的方法
  - 增设一个具有并行查寻能力的特殊高速缓冲寄存器，用于存放当前访问的那些表项
  - 这个高速缓存在IBM系统中取名为快表(TLB)，又称为联想存储器



具有快表的地址变换机构

4.5.4 两级和多级页表
4.5.5 反置页表

- 大家自己回去看！

练习题

1、最佳适应算法的空闲区按(B)排列

A. 大小递减顺序　　B. 大小递增顺序

C. 地址大小递增顺序　D. 地址大小递减顺序

2、首次适应算法的空闲区是(A)

A. 按地址递增顺序连在一起

B. 按大小递增顺序连在一起

C. 从最小空闲分区开始查找

D. 从最大空闲分区开始查找

3、把逻辑地址变换为物理地址，称为(B)

A. 加载　B. 重定位　C. 物理化　D. 逻辑化

4、在固定分区分配中，分区大小（ C ）

A.相同　　B. 随作业长度变化

C.可以不同但预先固定

D. 可以不同但据作业长度固定

5、动态分区分配中，系统回收一块内存，需要修改空闲分区表，造成空闲区数减1是哪种情况（ D ）

A. 无上、下相邻空闲区

B. 有上邻空闲区、但无下邻空闲区

C. 无上邻空闲区、但有下邻空闲区

D. 有上、下相邻空闲区

6、动态重定位技术依赖于(B)

A. 重定位装入程序　B. 重定位寄存器

C. 地址机构　D. 目标程序

7、在动态分区存储管理中的紧凑技术可以(A)

A. 集中空闲区　B. 增加主存容量

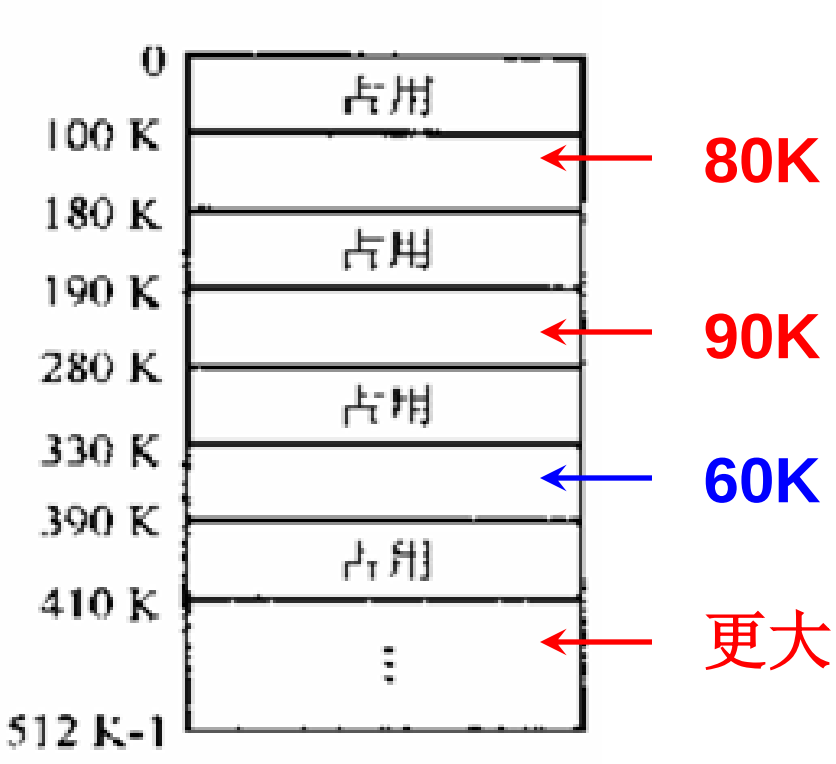
C. 缩短访问周期　D. 加速地址转换

8、在分页系统环境下，程序员编制的程序，其地址空间是连续的，分页是由(　D　)完成的

A. 程序员　B. 编译地址　C. 用户　D.系统

9、设内存的分配情况如下图，要申请一块40K的内存空间，若采用最佳适应算法，则所得的分区首址是(C)

A. 100K　B.190K　C.330K　D.410K



10、在一个分页存储管理系统中，页表内容如图。若页的大小为4K，则地址变换机构将逻辑地址0变换为物理地址(　A　)

A. 8192　B. 4096　C. 2048　D. 1024

页号	块号
0	2
1	1
2	6
3	3
4	7

- 逻辑地址0 => 页号0
  偏移量0
- 物理地址=对应块号*页大小
  + 偏移量
  = 2 * 4K + 0
  = 8K
  = 8*1024
  = 8192

11、某 OS 采用动态分区分配存储管理方法，用户区为512K，且始址为0。若分配时采用分配空闲区低地址部分的方案，且初始时用户的512K空间空闲，对下述申请序列：

申请300K，申请100K，释放300K，申请150K，申请30K，申请40K，申请60K，释放30K

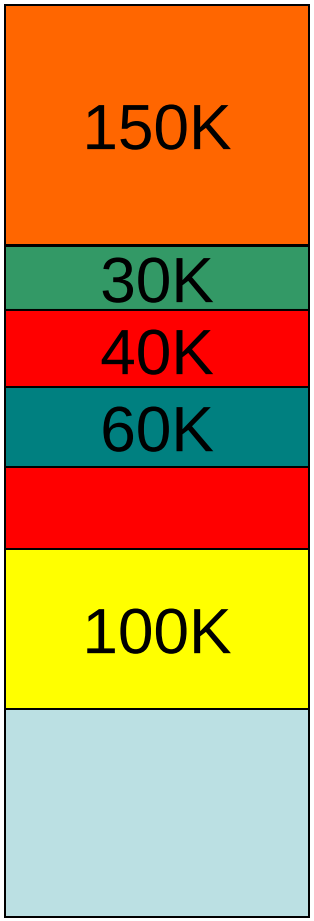
回答：

（1）采用首次适应算法，空闲分区中有哪些空块（给出始址、大小）？

（2）采用最佳适应算法，空闲分区中有哪些空块（给出始址、大小）？

（3）如再申请100K，针对（1）和（2）各有什么结果？

首次适应算法：



0

512K

申请 300K

申请 100K

释放 300K

申请 150K

申请 30K

申请 40K

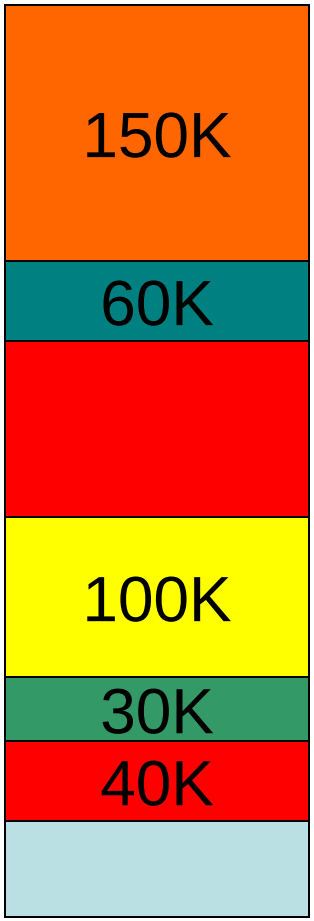
申请 60K

释放 30K

空闲区：

始址	大小
150K	**30K**
280K	**20K**
400K	**112K**

最佳适应算法：



0

512K

申请 300K

申请 100K

释放 300K

申请 150K

申请 30K

申请 40K

申请 60K

释放 30K

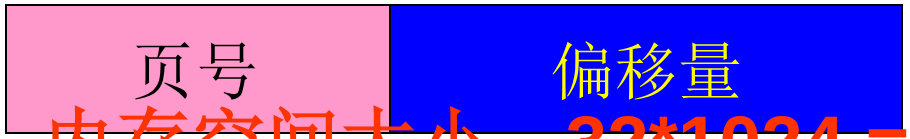
空闲区：始址　　大小

210K　　90K

400K　　30K

470K　　42K

12、设有一分页存储管理系统，向用户提供的逻辑地址空间最大为64页，每页1024B，内存总共有32个存储块，试问逻辑地址至少应为多少位？内存空间有多大？



内存空间大小：32*1024 = 32K

- 64=2^6，所以页号用6位表示
- 1024 = 2^{10}，所以偏移量用10位表示
- 逻辑地址应为6+10=16位

13、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4096B，现有一逻辑地址为2F6AH，且第0、1、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？

- 页面大小=4096B=4K=2^{12}
- 逻辑地址是16位，故页号占4位,偏移量占12位
- 2F6AH→页号是2H，偏移量是F6AH
- 所以它在11号物理块中(11的十六进制为BH)
- 相应地址为 B000H + F6AH =BF6AH

14、某采用分页存储管理的系统中，物理地址占20位，逻辑地址中页号占6位，页面大小为1KB，问：

①该系统的内存空间大小为多少？每个块的大小为多少？逻辑地址共几位？每个程序的最大长度为多少？

②若第0、1、2页分别放在第3、7、9块中，则逻辑地址0420H对应的物理地址是多少？

- **解答：**

① 物理地址占20位，所以该系统的内存空间大小为：

2^{20}=1MB

块的大小与页面大小相同，而页面大小为1KB，因此存储块的大小也为1KB

由于页面大小为1K=2^{10}，占10位，而页号占6位，因此逻辑地址共16位

该系统中的每个程序最大为：2^{16}=64KB

② 因为页号占6位，偏移量占10位

逻辑地址0420H的前6位为页号，后10位为偏移量

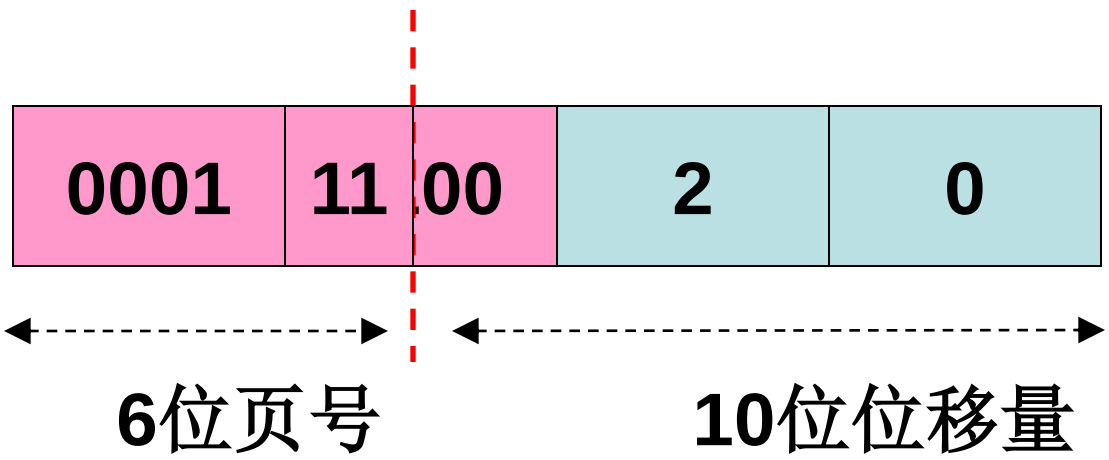
其前6位的二进制表示为$(0000,01)_2$，故在1号页内

偏移量为20H

因为，1号页面对应7号块

所以，物理地址可写为 $(0001,1100,0010,0000)_2$

所以，物理地址可写为1C20H



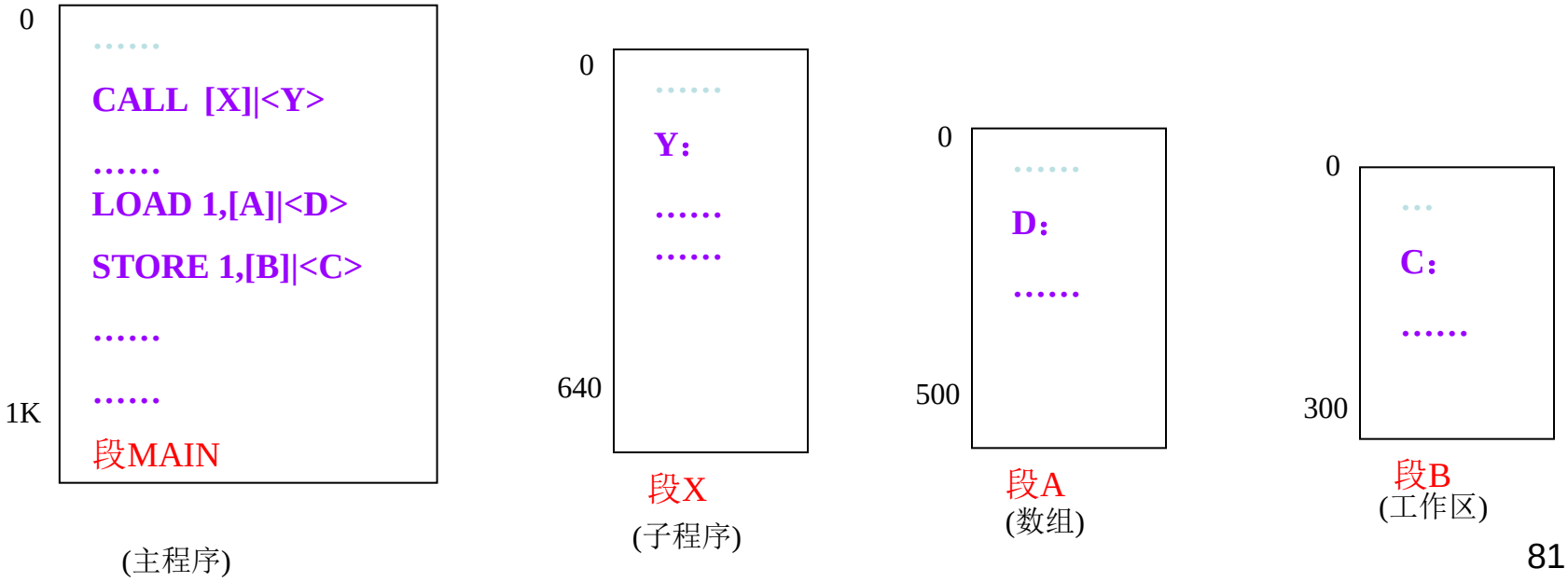
4.6 分段存储管理方式

4.6.1 分段存储管理方式的引入

- 引入目的：为了满足用户和程序员的需要

① 方便编程

– 程序员把程序划分成若干逻辑段，逻辑地址由段名、段内偏移量组成



② 信息共享

- 实现程序、数据的共享，是以信息的逻辑单位为基础的。页不是逻辑单位，所以难以实现信息共享

③ 信息保护

- 信息保护是对逻辑单位进行保护

④ 动态增长

- 数据段会随着运行不断增长，段式存储可以有效地应付这种情况

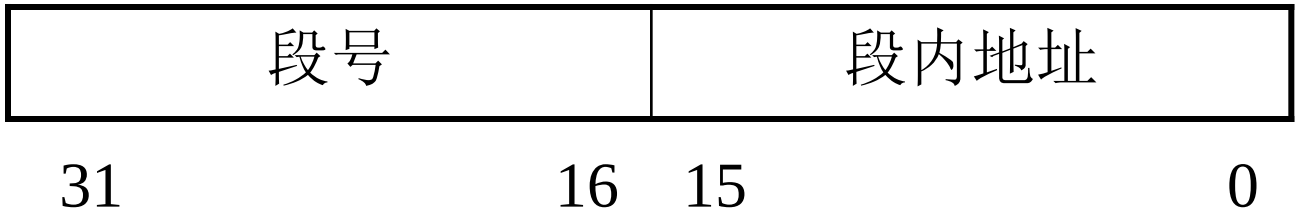
⑤ 动态链接

- 动态链接也是以逻辑单位为基础的

4.6.2 分段系统的基本原理

1. 分段

- 在分段管理系统中，程序经编译链接后，指令和数据的地址均由两部分构成：
  ① 表示段名的段号S
  ② 段内位移量W，即段内地址
- 所以，其地址结构有如下形式：

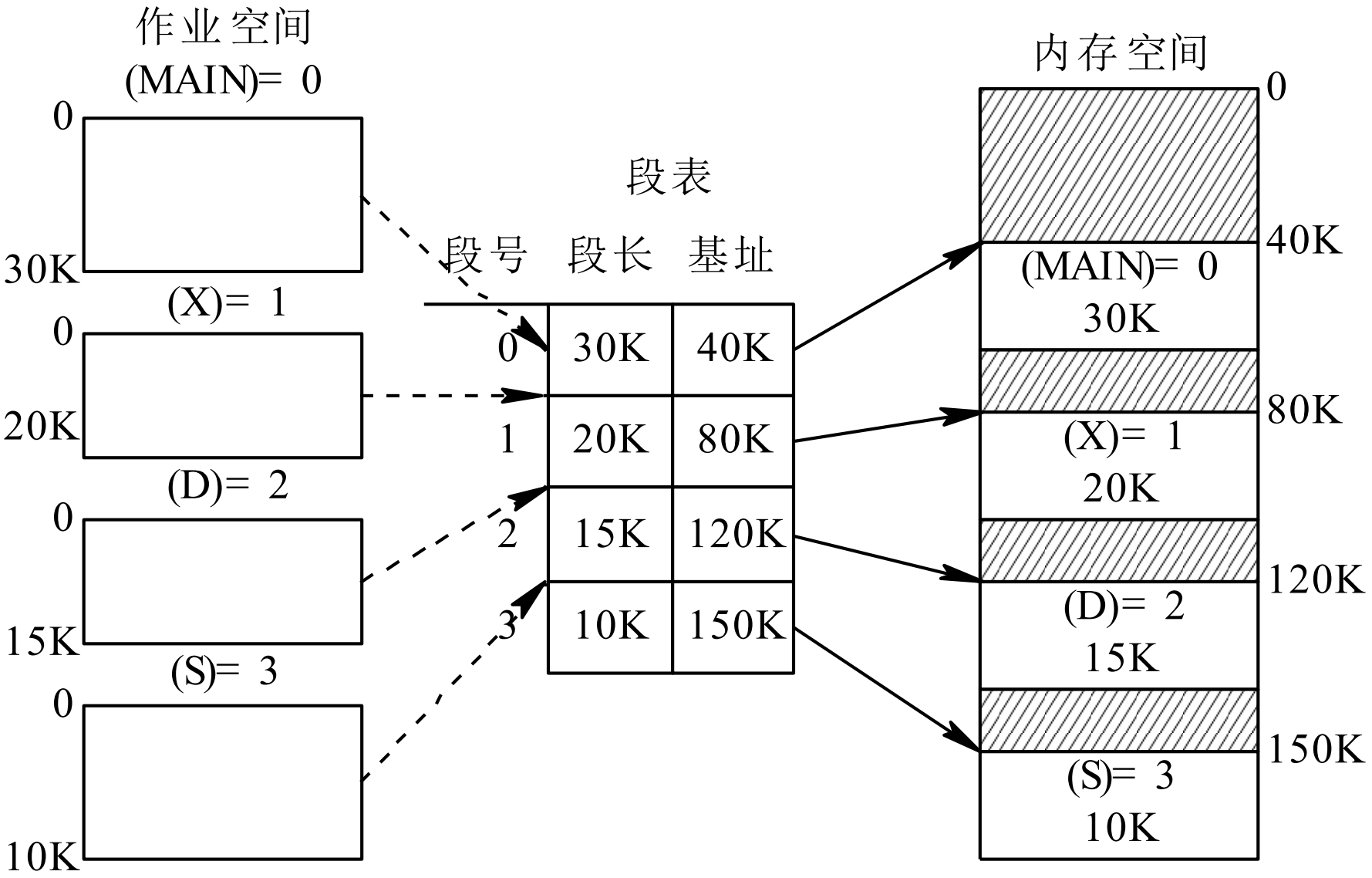


该地址结构中，最多可有64K个段，每个段最大64K

- 所谓分段管理，就是管理由若干段组成的程序，且按段进行存储分配。
- 实现分段管理的关键在于
  - 如何把逻辑地址变换成物理地址
  - 和分页管理一样，可采用动态重定位技术
  - 即，通过类似的地址变换机构来实现

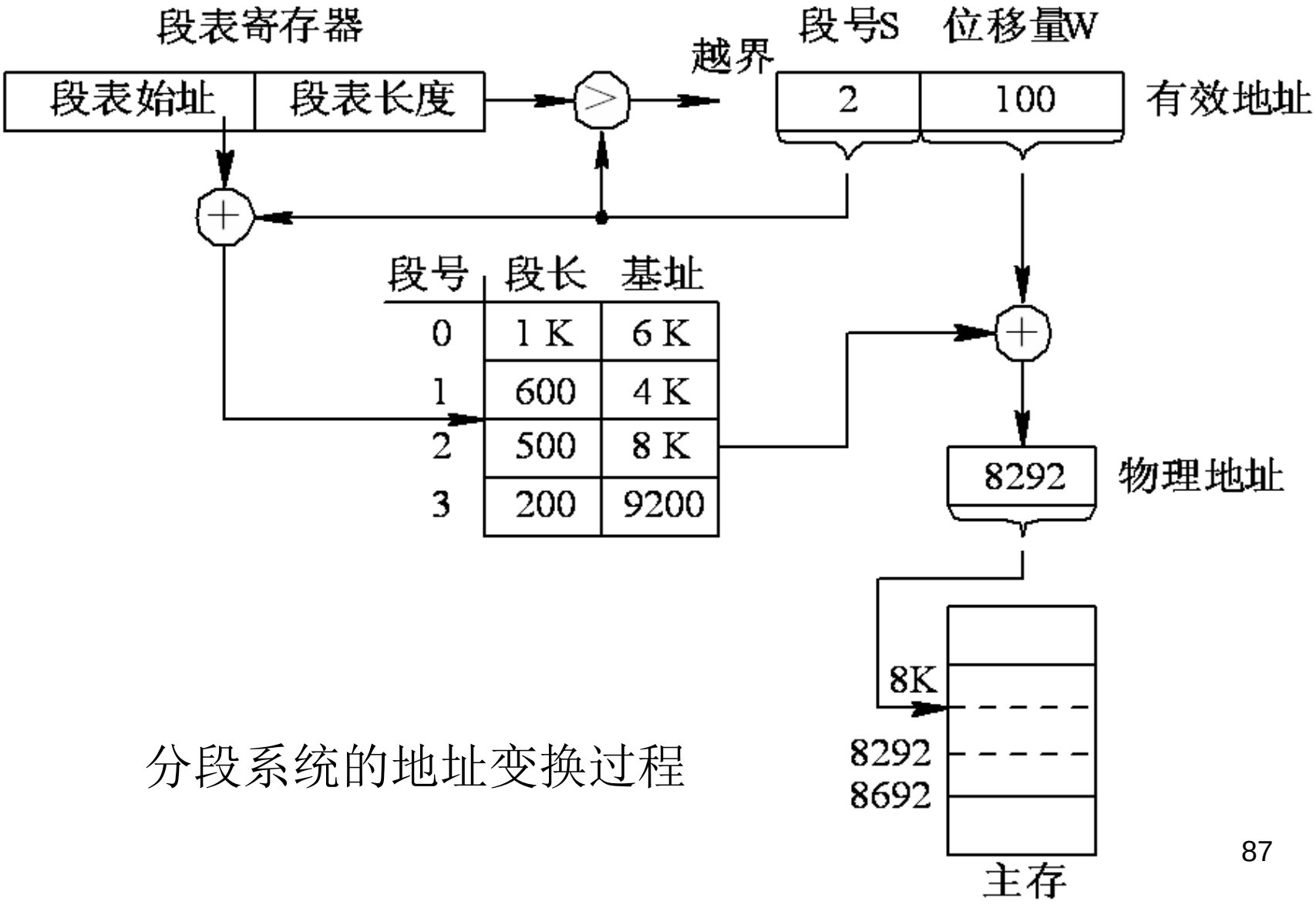
2. 段表

- OS在系统区为每个进程建立一张段表
- 每个段在段表中都占有一项，记录段的
  ① 段长
  ② 基址


作业空间
(MAIN)= 0
0
30K
(X)= 1
0
20K
(D)= 2
0
15K
(S)= 3
0
10K
段表
段号 段长 基址
0 30K 40K
1 20K 80K
2 15K 120K
3 10K 150K
内存空间
0
40K
(MAIN)= 0
30K
80K
(X)= 1
20K
120K
(D)= 2
15K
150K
(S)= 3
10K

利用段表实现地址映射

3. 地址变换机构



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，分页是为消减内存的碎片，提高内存的利用率。

分页仅是由于系统管理的需要。

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。

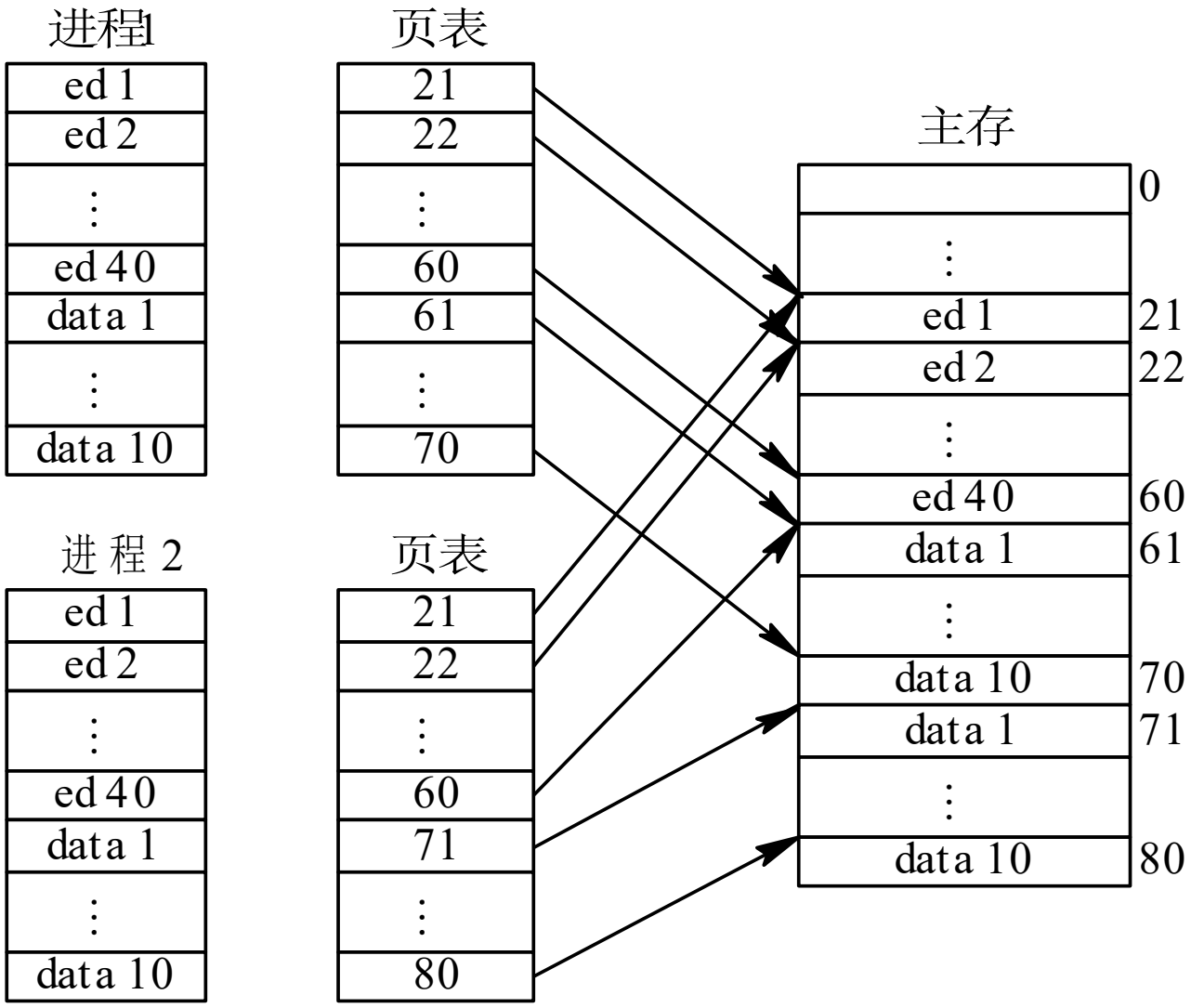
② 页的大小固定，由系统决定，由系统把逻辑地址划分为页号和页内地址两部分，是由机器硬件实现的，因而在系统中只能有一种大小的页面；

段的长度不固定，决定于用户所编写的程序，由编译程序编译源程序时，根据信息的性质来划分。

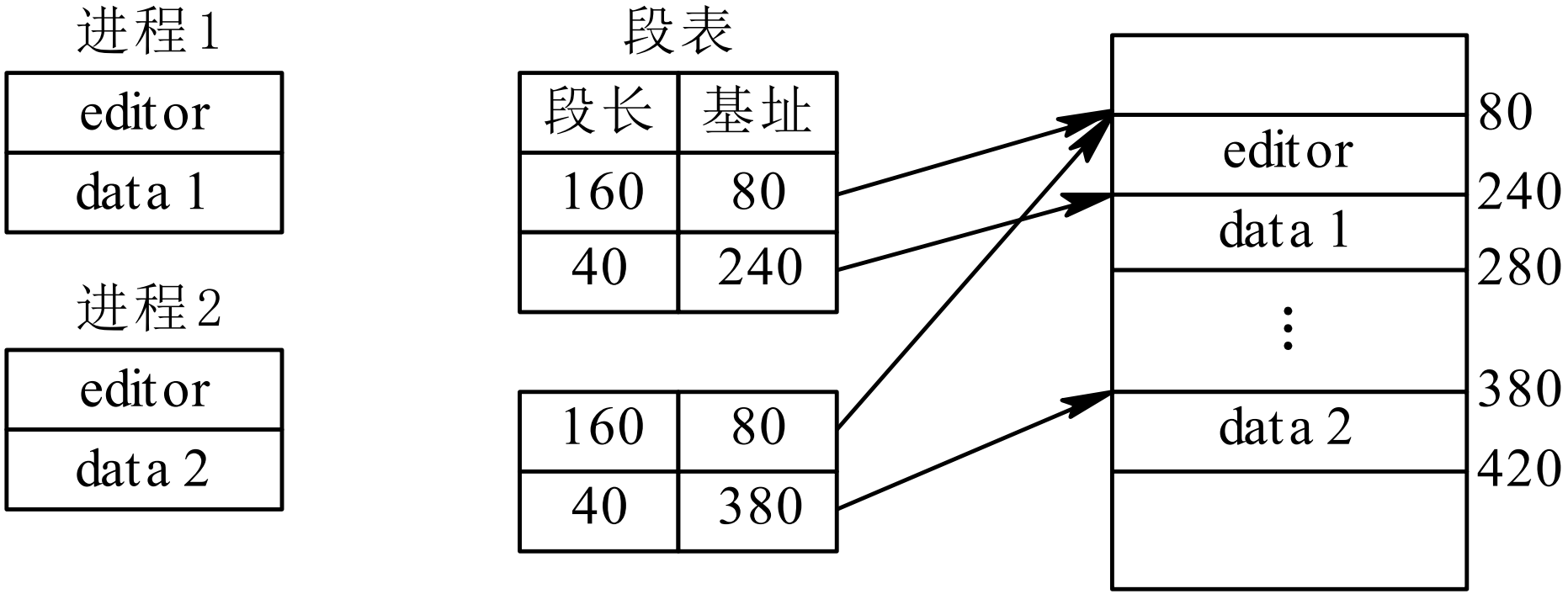
③分页的作业地址空间是一维的，即单一的线性地址空间，程序员只需利用一个记忆符，即可表示一个地址；

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



分页系统中共享editor的示意图



分段系统中共享editor的示意图

4.6.4 段页式存储管理方式

- 分页、分段两种管理方式各有优缺点

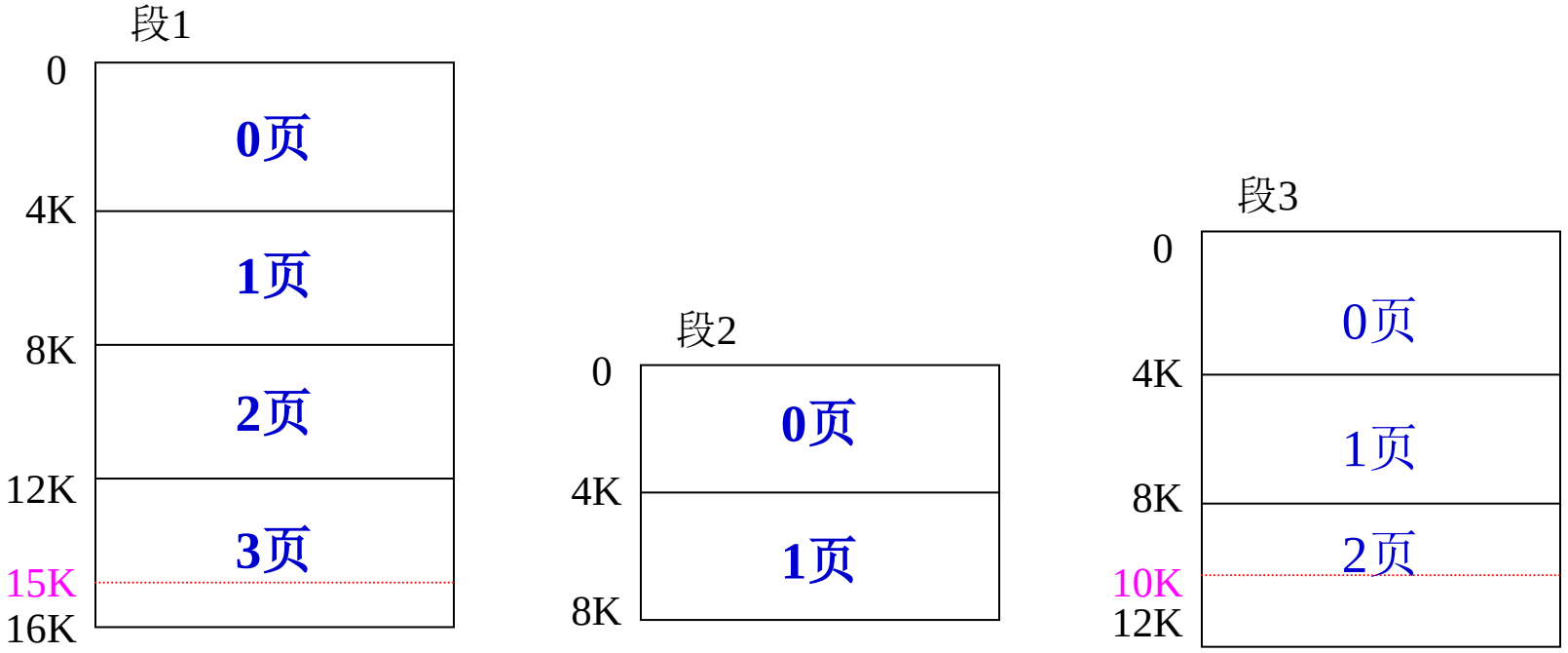
  ① 分页系统能有效提高内存利用率

  ② 分段系统能更好地满足用户的需要

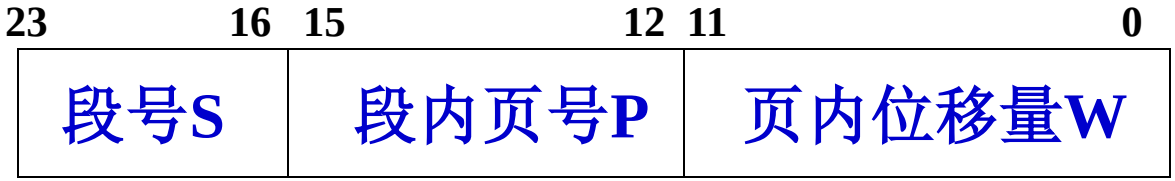
- 为了获得分段在逻辑上的优点和分页在管理存储空间方面的优点，结合分段和分页，形成段页式存储管理。

1. 基本原理

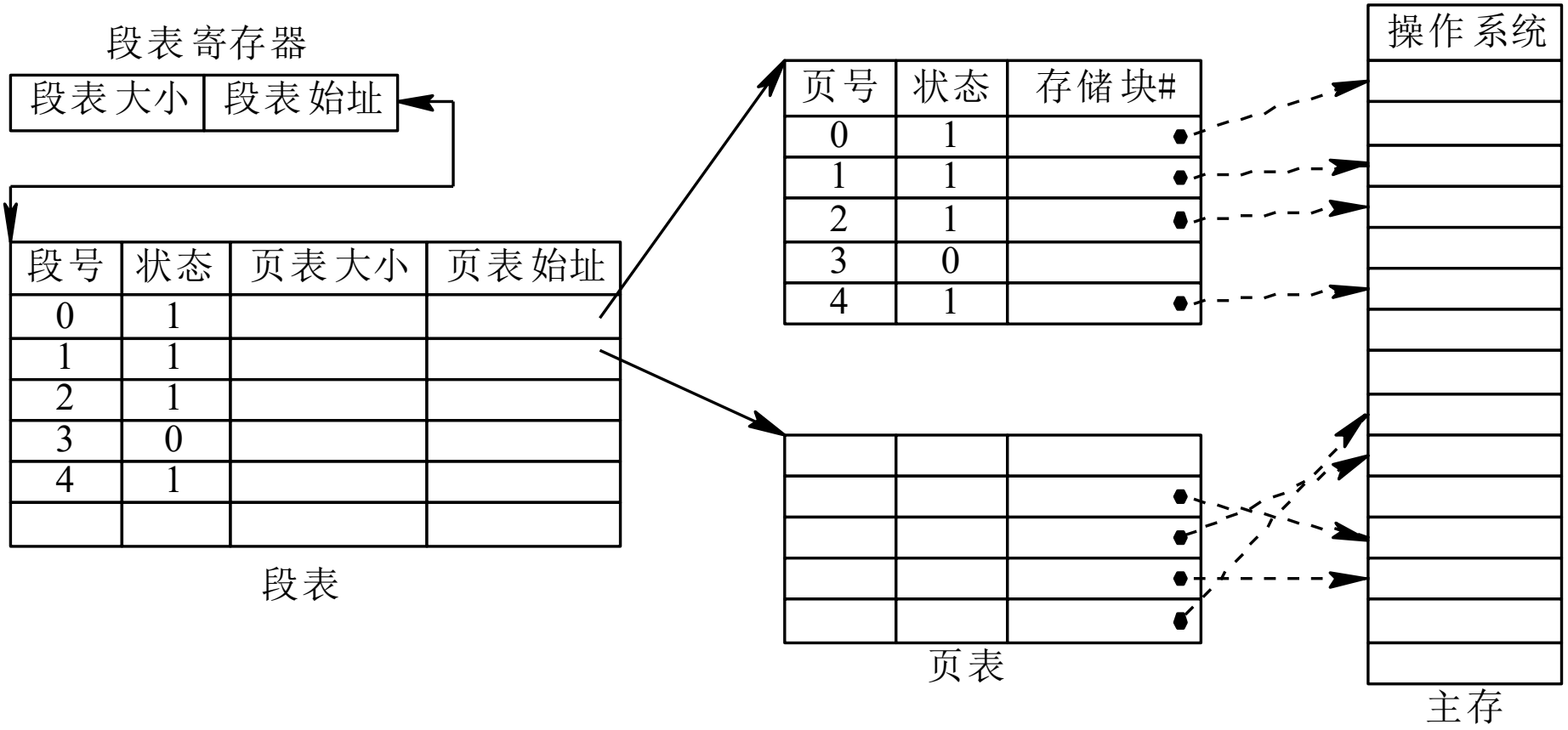
- 一个程序首先被分成若干段，每一个段赋予不同的段名(段号)
- 每一个段又分成若干个页面
- 由于段页式系统给逻辑地址空间增加了一级结构，现在地址空间便由段号S、段内页号P 和页内地址(位移量)W构成：



- 该图是段页式系统中一个程序的逻辑地址空间，页面大小4KB
- 该程序有3个段
  - 段1为15KB，占4页，最后一页有1KB未用；
  - 段2为8KB，恰好占满2页；
  - 段3为10KB，占3页，最后一页有2KB未用。
- 和分页系统一样，这些未写满的页依然存在页内碎片问题

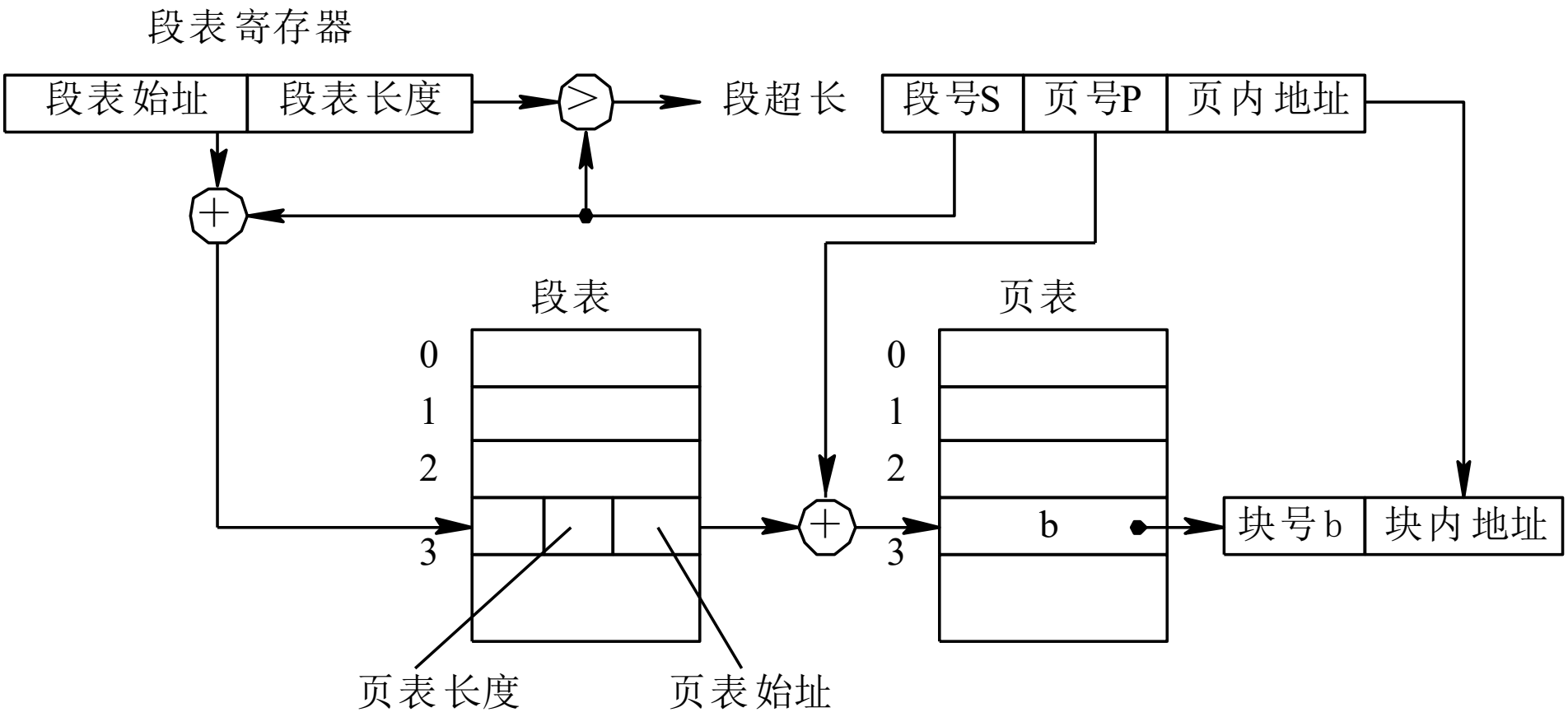


- 地址结构确定了一个程序可用地址空间的范围，限定了一个程序最多能有多少段，每段多少页，以及页面的大小
- 上图的结构可允许每个程序有256段，每段16页、页面的大小为4K字节
- 注意：程序的分段可由程序员或编译程序根据信息的逻辑结构来划分；而分页则与程序员无关，是由系统自动进行的
- 也即，程序中使用的地址形式仍然是二维的，即段号S和段内地址；只是由地址变换机构把段内地址 解析成页号P和页内地址的 (位移量)W形式



利用段表和页表实现地址映射

2. 地址变换过程



段页式系统中的地址变换机构

练习题

1、采用段式管理的系统里，若地址用24位表示，其中8位表示段号，则允许每段的最大长度是(B)

A. 2^{24} B. 2^{16} C. 2^{8} D. 2^{32}

2. 很好解决了“碎片”问题的存储管理方法是(A)

A. 分页存储管理　B．分段存储管理

C. 多重分区管理　D．可变式分区管理

3、在段页式存储管理系统中，内存等分成（ A ），程序按逻辑模块划分成若干（ D ）

A. 块　B．基址　C. 分区

D．段　E．页号　F．段长

4、 在一个分段存储管理系统中，其段表为：

段号	内存起始地址	段长
0	210	500
1	2350	20
2	100	90
3	1350	590
4	1938	95

1	10
4	32

试求表中逻辑地址对应的物理地址是什么？

2350 + 10 = 2360

1938 + 32 = 1970

5、某段表的内容如下，逻辑地址(2/154)，其中2为段号，154为段内地址，它对应的物理地址为（ B ）

A．120K+2　　B．480K+154

C．30K+154　　D．2+480K

段号	段首址	段长度
0	120 K	40 K
1	760 K	30 K
2	480 K	20 K
3	370 K	20 K

6. 在分页存储管理方案中，采用 (A) 实现地址变换

A. 页表 B. 段表 C. 段表和页表 D. 空闲区表

7. 在段页式存储管理系统中，每道程序都有一个(段)表和一组(页)表

小结

- 连续分配方式(主要是：固定分区、动态分区)
- 分页、分段存储管理方式
- 段页式存储管理技术