操作系统
第十三讲

张涛

Review

进程、线程的概念

处理器调度

进程间同步与通信

死锁

第四章 存储管理

存储管理的基本概念

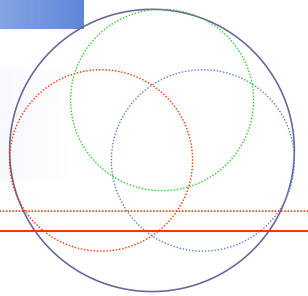
分区存储管理

分页存储管理

虚拟存储

请求分页存储管理

分段/段页式存储管理



Today we focus on …

存储管理的基本概念

存储系统的组织与四大功能

地址再定位

早期的存储管理

分区存储管理

4.1 存储管理的基本概念

- 重要性
  - 直接存取要求内存速度尽量快到与CPU取指速度相匹配，大到能装下当前运行的程序与数据，否则CPU执行速度就会受到内存速度和容量的影响而得不到充分发挥
  - 重要资源，“瓶颈”
- 帕金森定律
  - 内存多大，程序多长

- **内存：** 由存储单元（字节或字）组成的一维连续的地址空间，简称内存空间。用来存放当前正在运行程序的代码及数据，是程序中指令本身地址所指的、亦即程序计数器所指的存储器。
- **分为：**
  - 系统区：用于存放操作系统
  - 用户区：用于装入并存放用户程序和数据
- **存储管理的目的：**
  - 充分利用内存，为多道程序并发执行提供存储基础
  - 尽可能方便用户使用，如：自动装入用户程序，用户程序中不必考虑硬件细节
  - 解决程序空间比实际内存空间大的问题

4.1.1 存储系统的组织与四大功能

- 高速缓存Cache：少量的、非常快速、昂贵、易变
- 内存RAM：若干兆字节、中等速度、中等价格、易变
- 磁盘：数百兆或数千兆字节、低速、价廉、不易变

- 存储器的功能是保存数据，存储器的发展方向是高速、大容量和小体积。
  - 内存在访问速度方面的发展： SRAM 、DRAM、SDRAM等；
  - 硬盘技术在大容量方面的发展：接口标准、存储密度等；
- 存储组织是指在存储技术和CPU寻址技术许可的范围内组织合理的存储结构。
  - 其依据是访问速度匹配关系、容量要求和价格。
  - “寄存器-内存-外存”结构
  - “寄存器-缓存-内存-外存”结构；
- 微机中的存储层次组织：
  - 访问速度越慢，容量越大，价格越便宜；
  - 最佳状态应是各层次的存储器都处于均衡的繁忙状态（如：缓存命中率正好使主存读写保持繁忙）；

存储管理的四大功能

- （1）存储空间的管理、分配和回收
  - 记录内存的使用情况——设置相应的内存分配表（内存分配回收的依据）
  - 静态存储分配；动态存储分配
  - 分配和回收算法及相应的数据结构。
- （2）地址再定位（地址变换、地址映射）：
  - 可执行文件生成中的链接技术
  - 程序加载(装入)时的重定位技术
  - 进程运行时硬件和软件的地址变换技术和机构

- （3）存储共享和保护：两个或多个进程共用内存中相同区域
  - 代码和数据共享
  - 地址空间访问权限、基址—限长存储保护
    - 上、下界存储保护
  - 保护过程----防止地址越界、防止操作越权

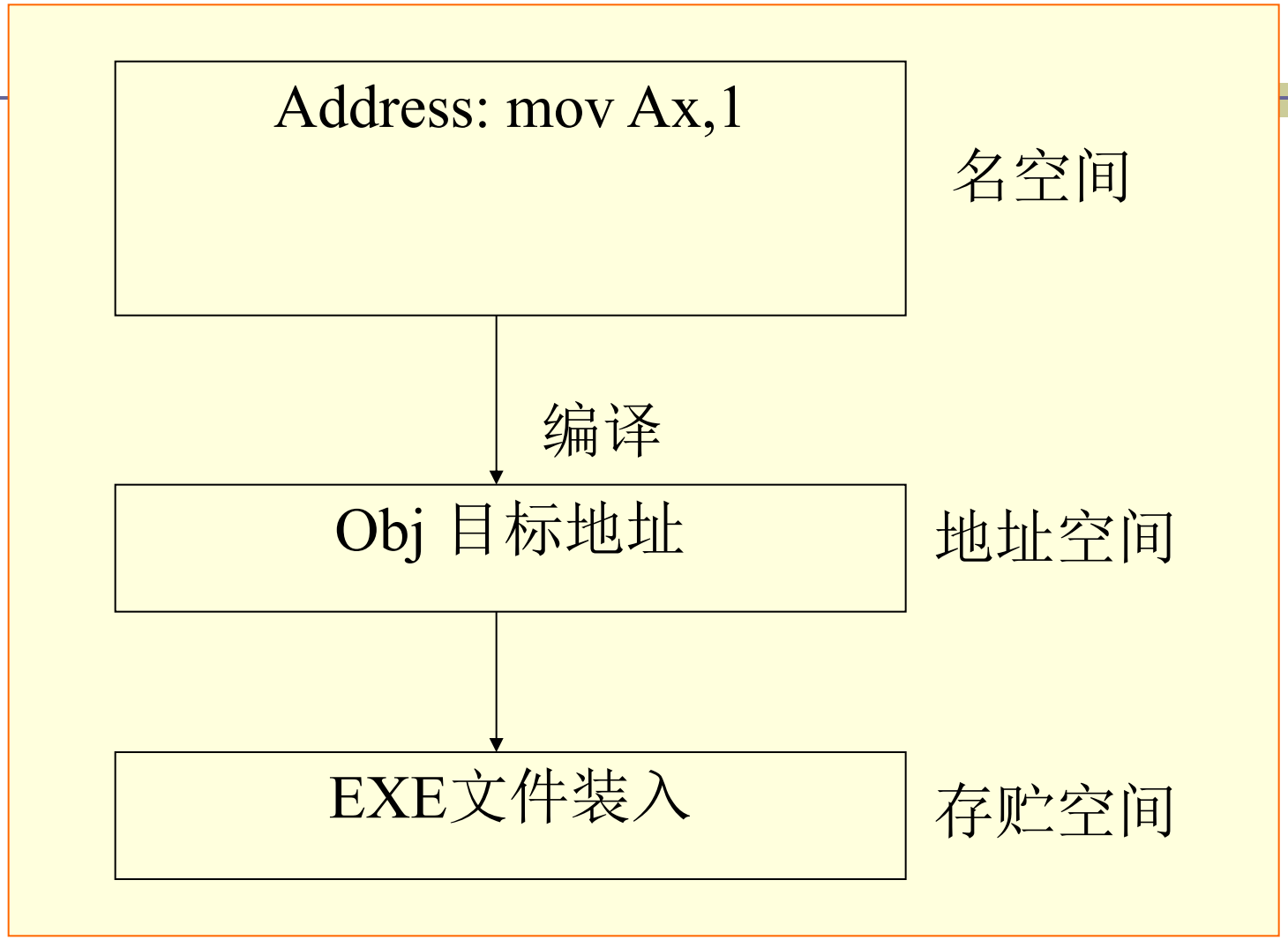
（a）上、下界保护

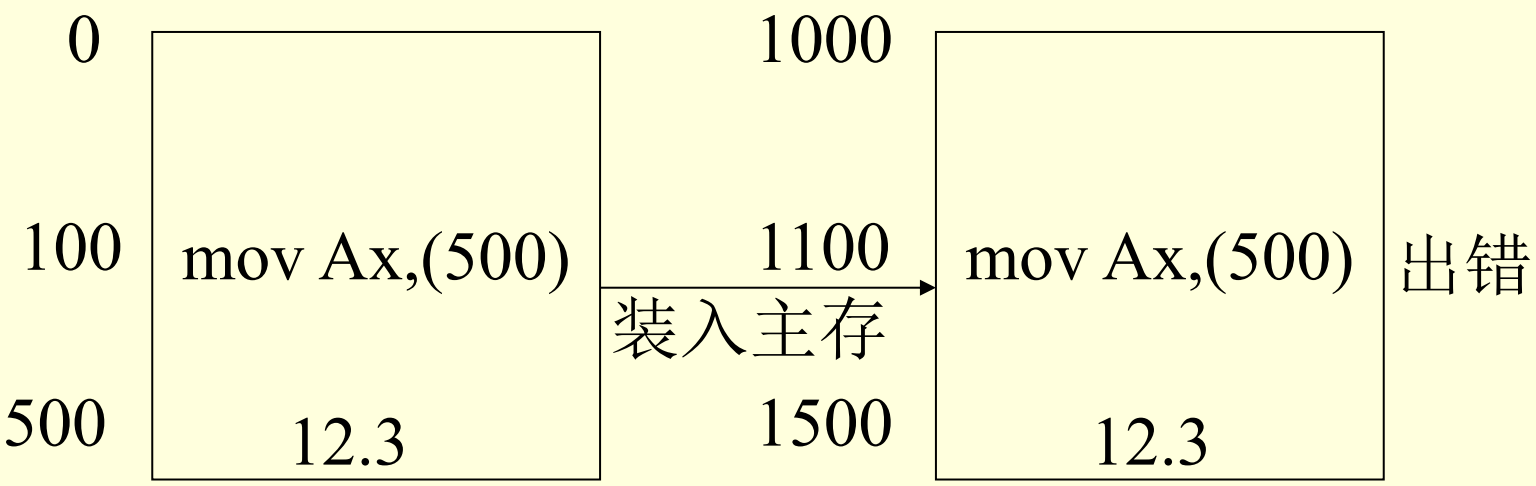
（b）基址——限长保护

- （4）存储器扩充：存储器的逻辑组织和物理组织，虚拟存储；
  - 由应用程序控制：覆盖；
  - 由OS控制
    - 交换（整个进程空间）
    - 虚拟存储的请求调入和预调入（部分进程空间）

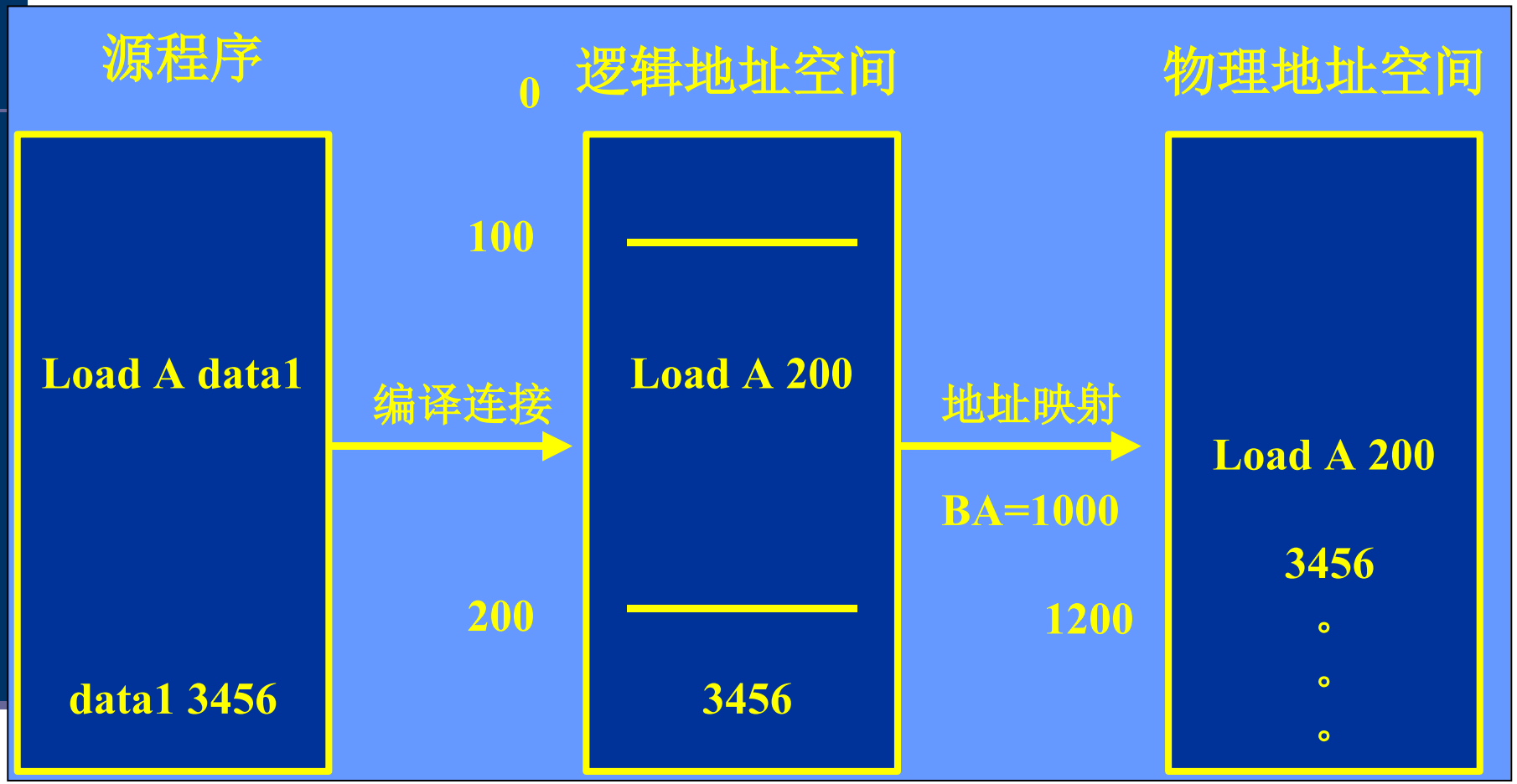
4.1.2 地址再定位

- 名空间：程序中由符号名组成的空间。
- 物理地址（绝对地址，实地址）：内存中存储单元的地址。物理地址可直接寻址。
- 逻辑地址（相对地址，虚地址）：用户的程序经过汇编或编译后形成目标代码，目标代码通常采用相对地址的形式。是指相对于某个基准量(通常用0)编址时所使用的地址。
  - 其首地址为0，其余指令中的地址都相对于首地址来编址。
  - 不能用逻辑地址在内存中读取信息。
- 逻辑地址空间通过地址再定位机构转换到绝对地址空间。


Address: mov Ax,1
名空间
编译
Obj 目标地址
地址空间
EXE文件装入
存贮空间



解决方法：重定位



逻辑地址、物理地址和地址映射

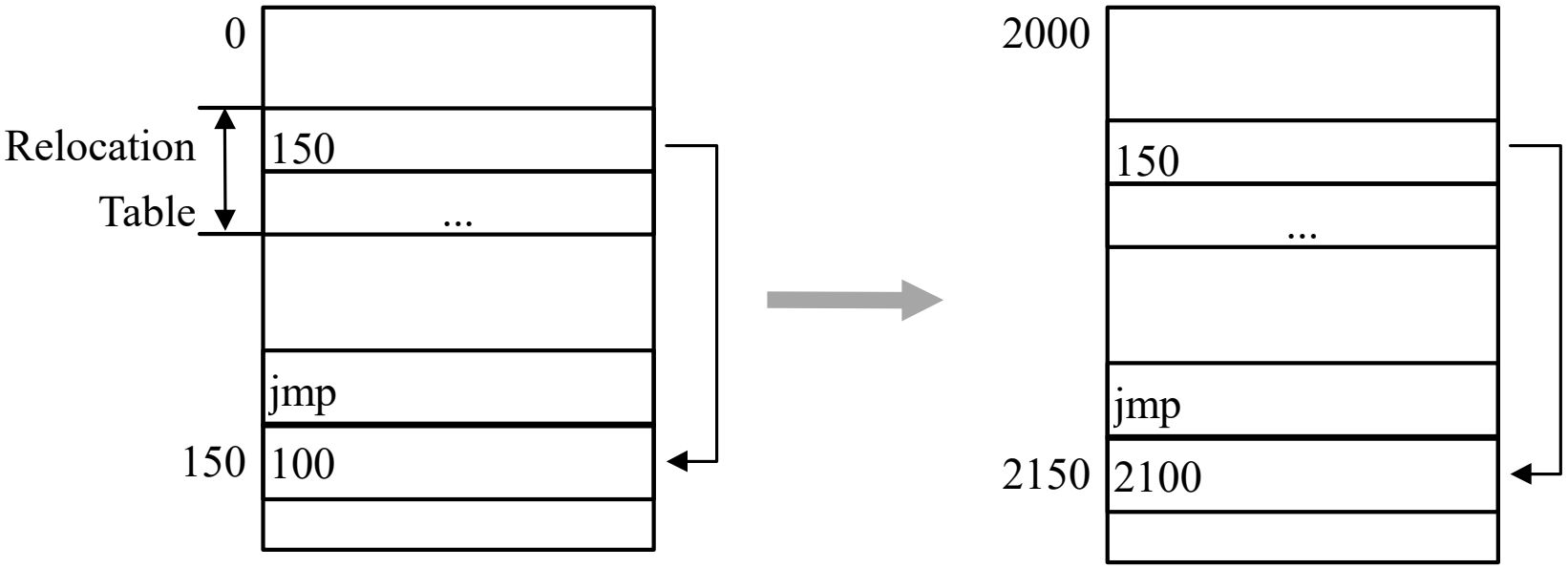
地址的再定位方法

- 将逻辑地址空间的程序装入到物理地址空间时，由于两个空间不一致，需要进行地址变换，所引起的对有关地址部分的调整过程称为地址再定位。
- 程序在成为进程前的准备工作
  - 编辑：形成源文件（符号地址）
  - 编译：形成目标模块（模块内符号地址解析）
  - 链接：由多个目标模块或程序库生成可执行文件（模块间符号地址解析）
  - 装入：构造PCB，形成进程（使用物理地址）
- 再定位方法
  - 静态再定位
  - 动态再定位

静态地址再定位

- 在程序执行之前进行地址再定位，由装配程序完成。
  - 在可执行文件中，列出各个需要重定位的地址单元和相对地址值。当用户程序被装入内存时，一次性实现逻辑地址到物理地址的转换，以后不再转换（一般在装入内存时由软件完成）。即：装入时根据所定位的内存地址去修改每个重定位地址项，添加相应偏移量。
- **优点：** 不需硬件支持，可以装入有限多道程序。
- **缺点：**
  - 程序装入内存后不能移动
  - 一个程序通常需要占用连续的内存空间
  - 不易实现共享

可执行文件在内存中的重定位

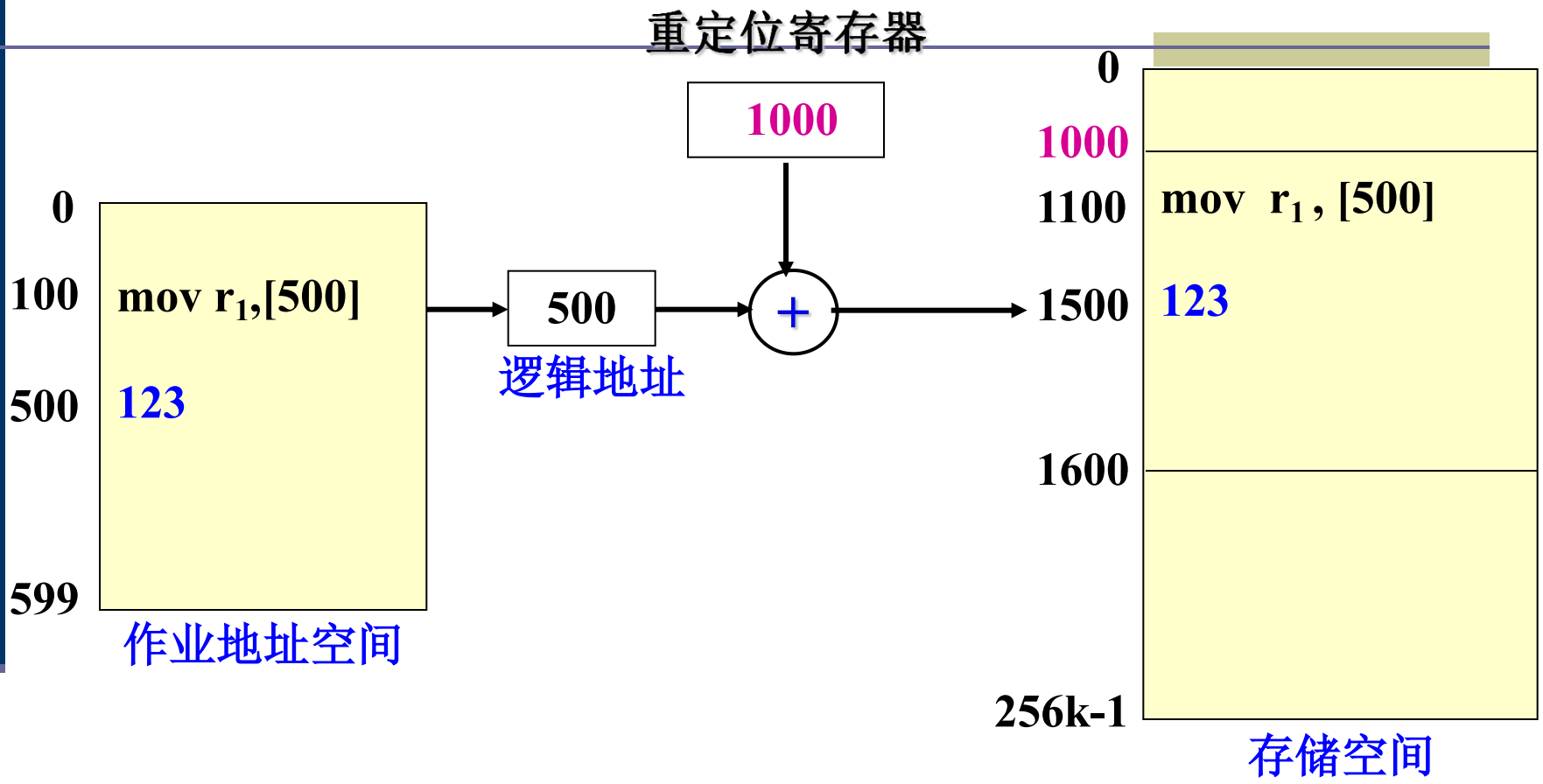


- 重定位修改：重定位表中的150->绝对地址2150(=2000+150)
- 内容修改：内容100变成2100(=100+2000))。

动态地址再定位

- 在执行寻址时重定位——在程序运行过程中要访问数据时再进行地址变换，即在逐条指令执行时完成地址映射。
- 一般为了提高效率，此工作由硬件地址映射机制来完成。硬件支持，软硬件结合完成
- 硬件上需要一对寄存器的支持：基址寄存器、变址寄存器。

动态地址映射过程示意图



动态地址映射的优缺点

- 优点：程序占用的内存空间是动态可变的，当程序从某个存储区移到另一个区域时，只需要修改相应的寄存器BR的内容即可。
  - 一个程序不一定要求占用一个连续的内存空间。
  - 可以部分地装入程序运行。
  - 便于多个进程共享同一个程序的代码。
- 动态地址重定位的代价：
  - 需要硬件的支持。
  - 实现存储管理的软件算法较为复杂。

IBM. PC的情况

源程序 —编译→ .obj —link→ .EXE —exezbin→ .com

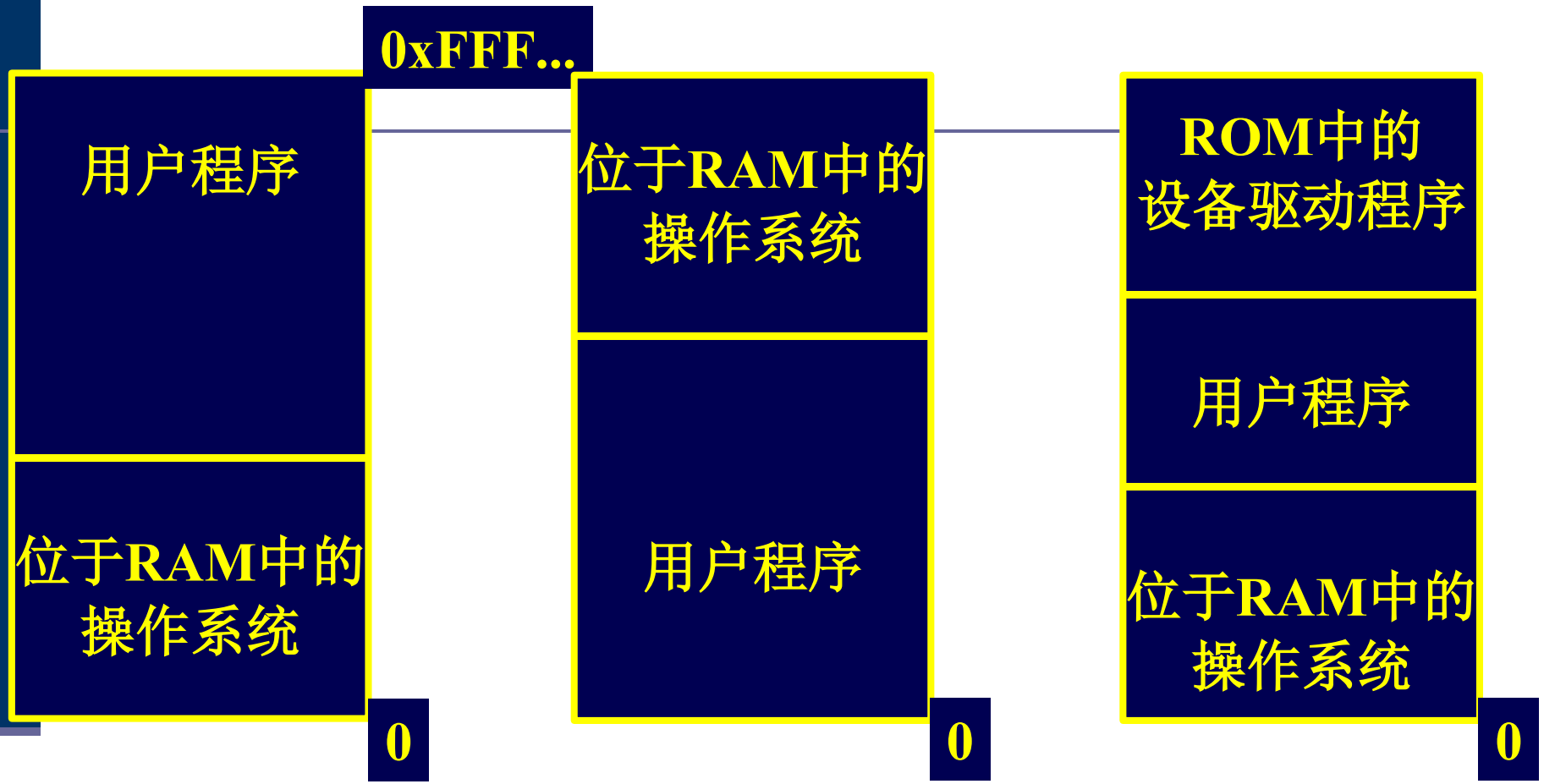
.com绝对地址：装入即可运行

.com文件<64k

.exe装入(重定位) .exe文件>64k(可以)

4.1.3 早期的存储管理

- 单一连续分区
  - 内存分为两个区域：系统区，用户区。应用程序装入到用户区，可使用用户区全部空间。最简单，适用于单用户、单任务的OS。
  - 优点：易于管理。
  - 缺点：对要求内存空间少的程序，造成内存浪费；程序全部装入，很少使用的程序部分也占用内存。


0xFFF...
用户程序
位于RAM中的
操作系统
0
位于RAM中的
操作系统
用户程序
0
ROM中的
设备驱动程序
用户程序
位于RAM中的
操作系统
0

单一连续区存储管理

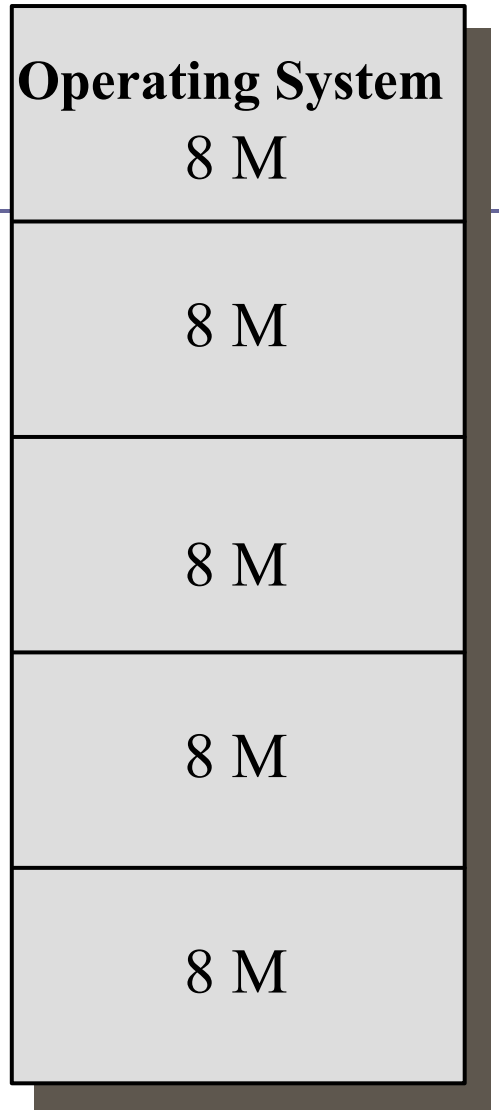
4.2 分区存储管理

- **原理：** 把内存分为一些大小相等或不等的分区，每个应用进程占用一个或几个分区。操作系统占用其中一个分区。
- **特点：** 适用于多道程序系统和分时系统
- **问题：** 可能存在内碎片和外碎片。
  - 内碎片：占用分区之内未被利用的空间
  - 外碎片：占用分区之间难以利用的空闲分区（通常是小空闲分区）。
- **分区方式：**
  - 固定分区(fixed partitioning)
  - 动态分区(dynamic partitioning)
  - 分区分配算法

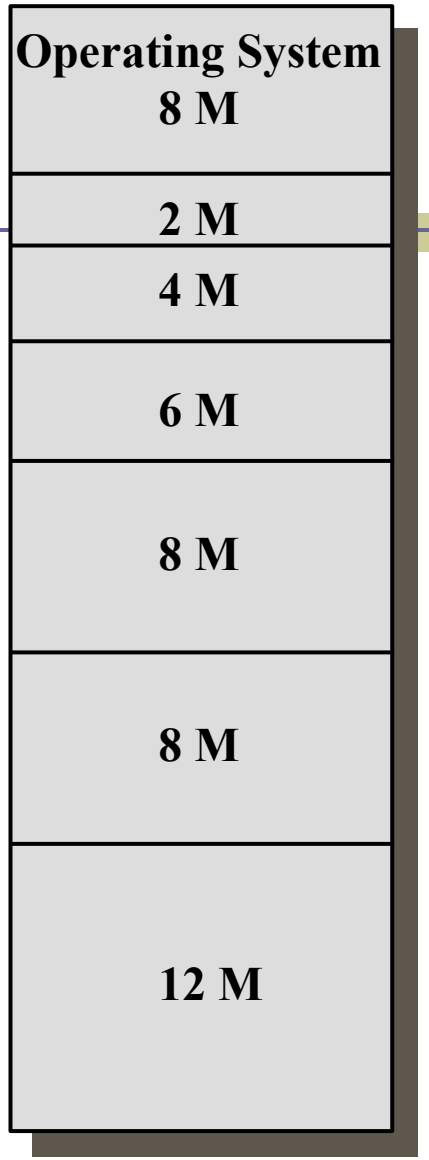
- 分区的数据结构：分区表，或分区链表
  - 只记录空闲分区，或同时记录空闲和占用分区
- 内存紧缩(compaction)：将各个占用分区向内存一端移动。使各个空闲分区聚集在另一端，然后将各个空闲分区合并成为一个空闲分区。
  - 对占用分区进行内存数据搬移占用CPU时间
  - 如果对占用分区中的程序进行"浮动"，则其重定位需要硬件支持。
  - 紧缩时机：每个分区释放后，或内存分配找不到满足条件的空闲分区时

4.2.1 固定分区(fixed partitioning)

- 把内存划分为若干个固定大小的连续分区。
  - **分区大小相等**：只适合于多个相同程序的并发执行（处理多个类型相同的对象）。
  - **分区大小不等**：多个小分区、适量的中等分区、少量的大分区。根据程序的大小，分配当前空闲的、适当大小的分区。
- 优点：易于实现，开销小。
- 缺点：
  - 内碎片造成浪费
  - 分区总数固定，限制了并发执行的程序数目。
- 采用的数据结构：分区表——记录分区的大小和使用情况



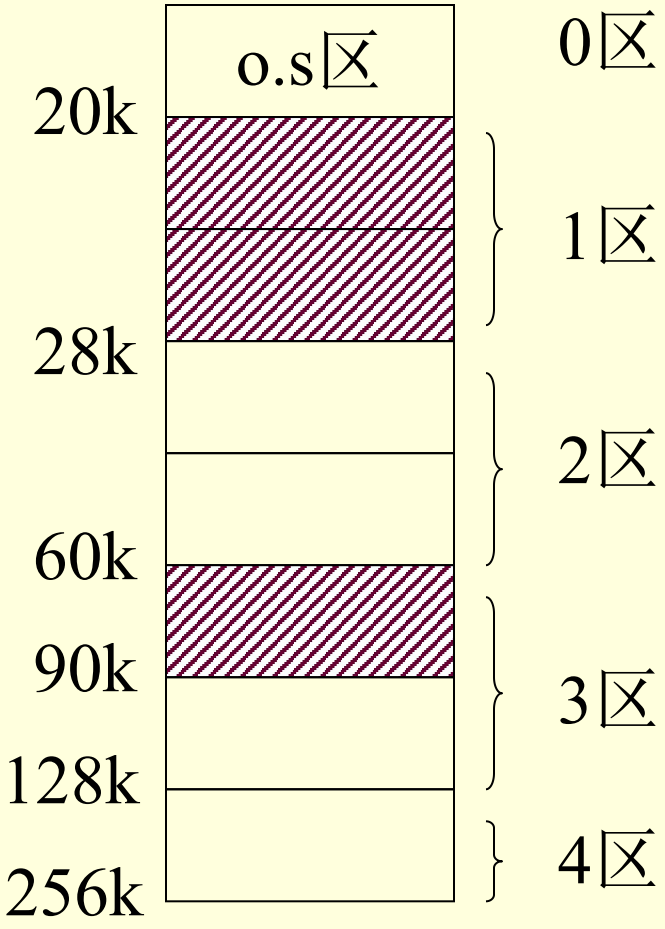
固定分区（大小相同）

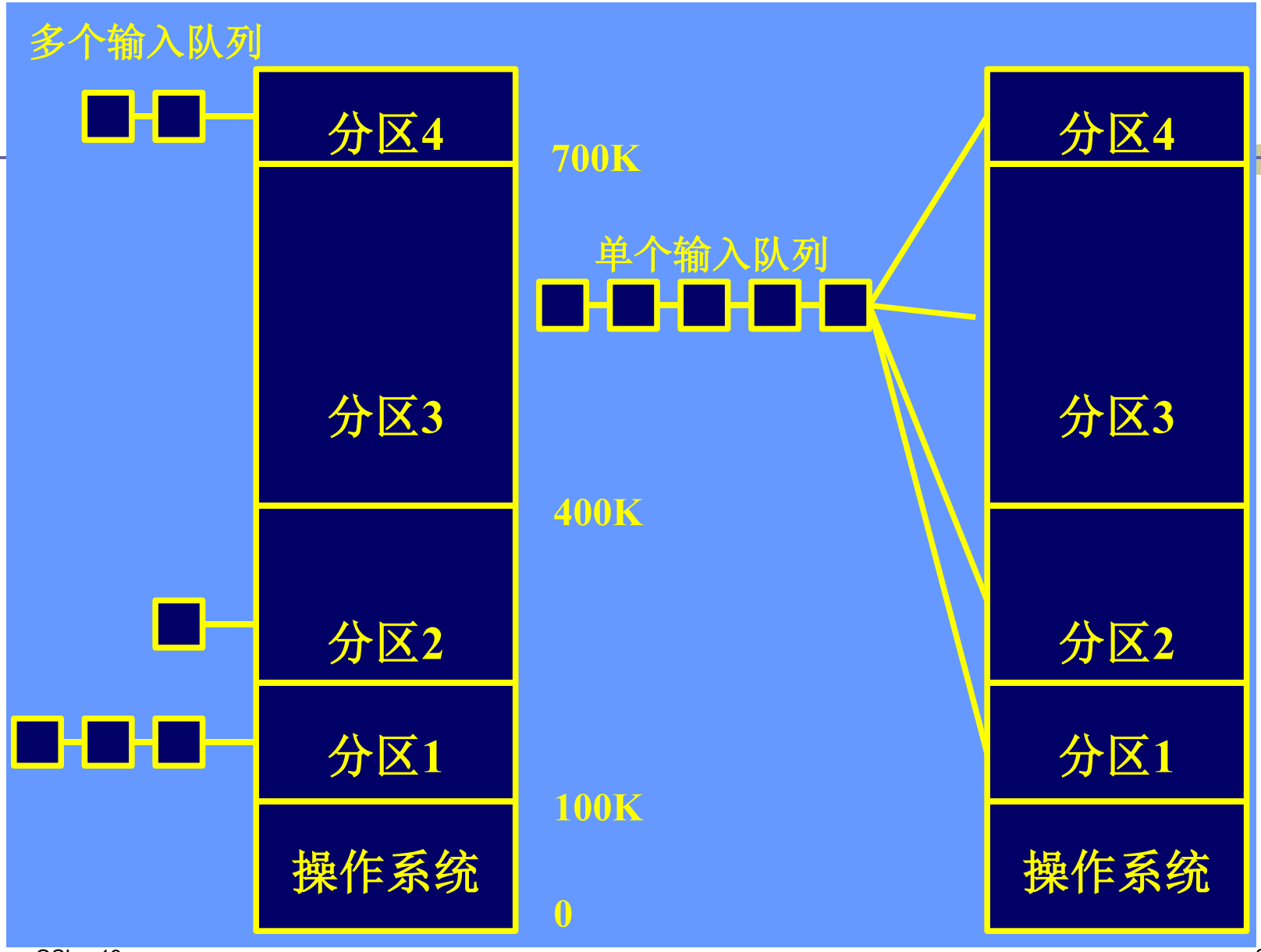


固定分区（多种大小）

区号	大小	起址	状态
1	8k	20k	in using
2	32k	28k	NuL
3	68k	60k	in using
4	128k	128k	NuL

(存贮分块表MBT)

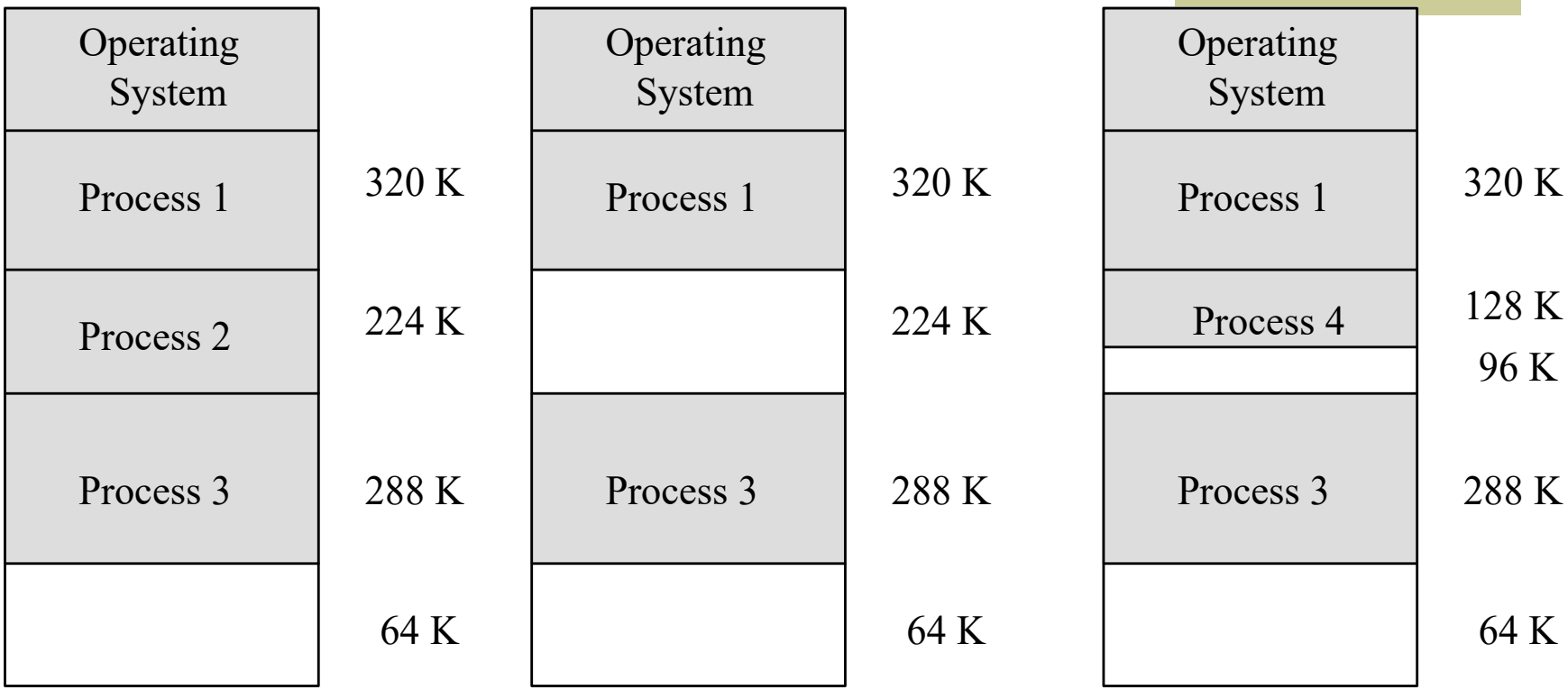


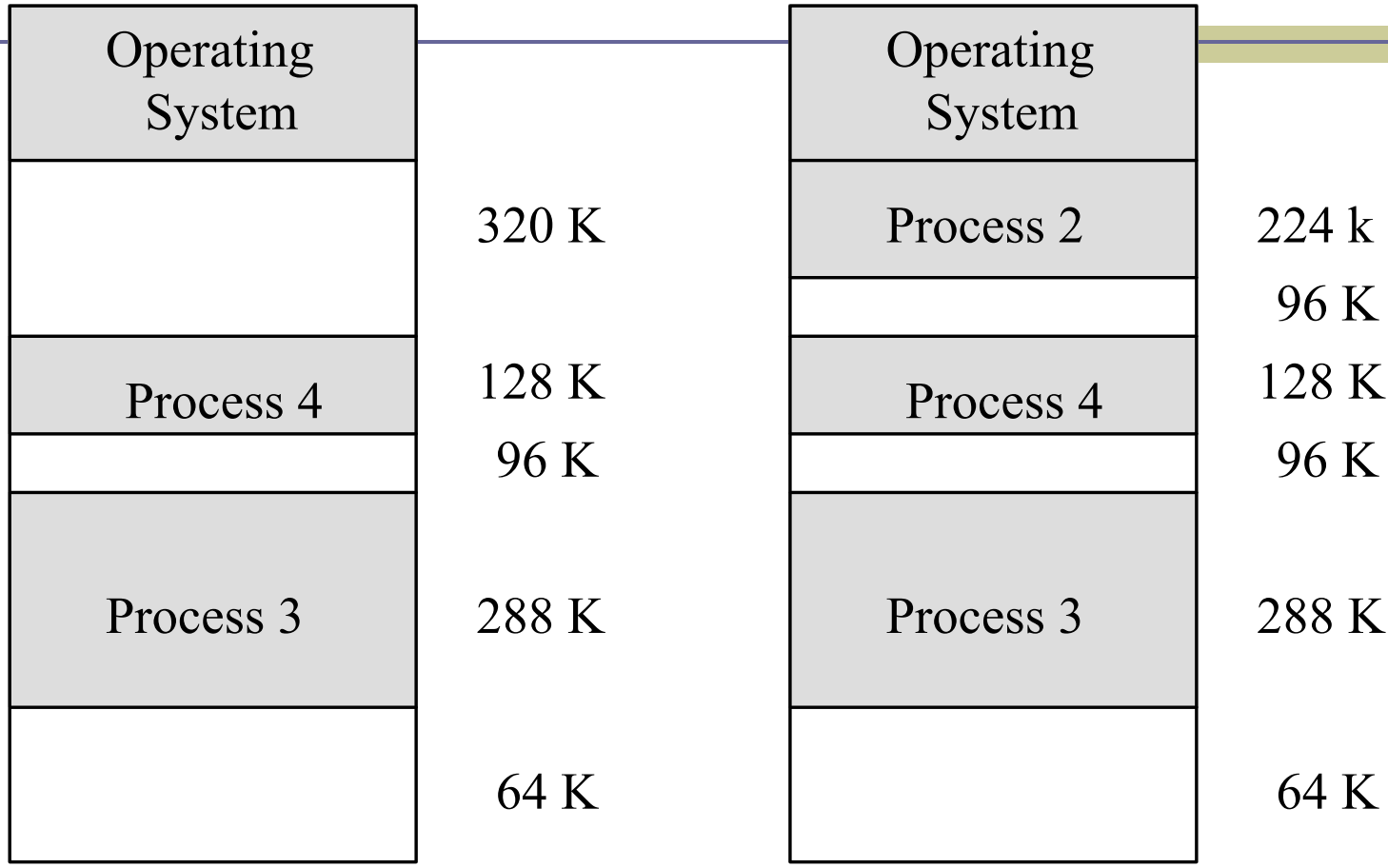

多个输入队列
分区4
分区3
分区2
分区1
操作系统
700K
400K
100K
0
单个输入队列
分区4
分区3
分区2
分区1
操作系统

4.2.2 动态分区(dynamic partitioning)

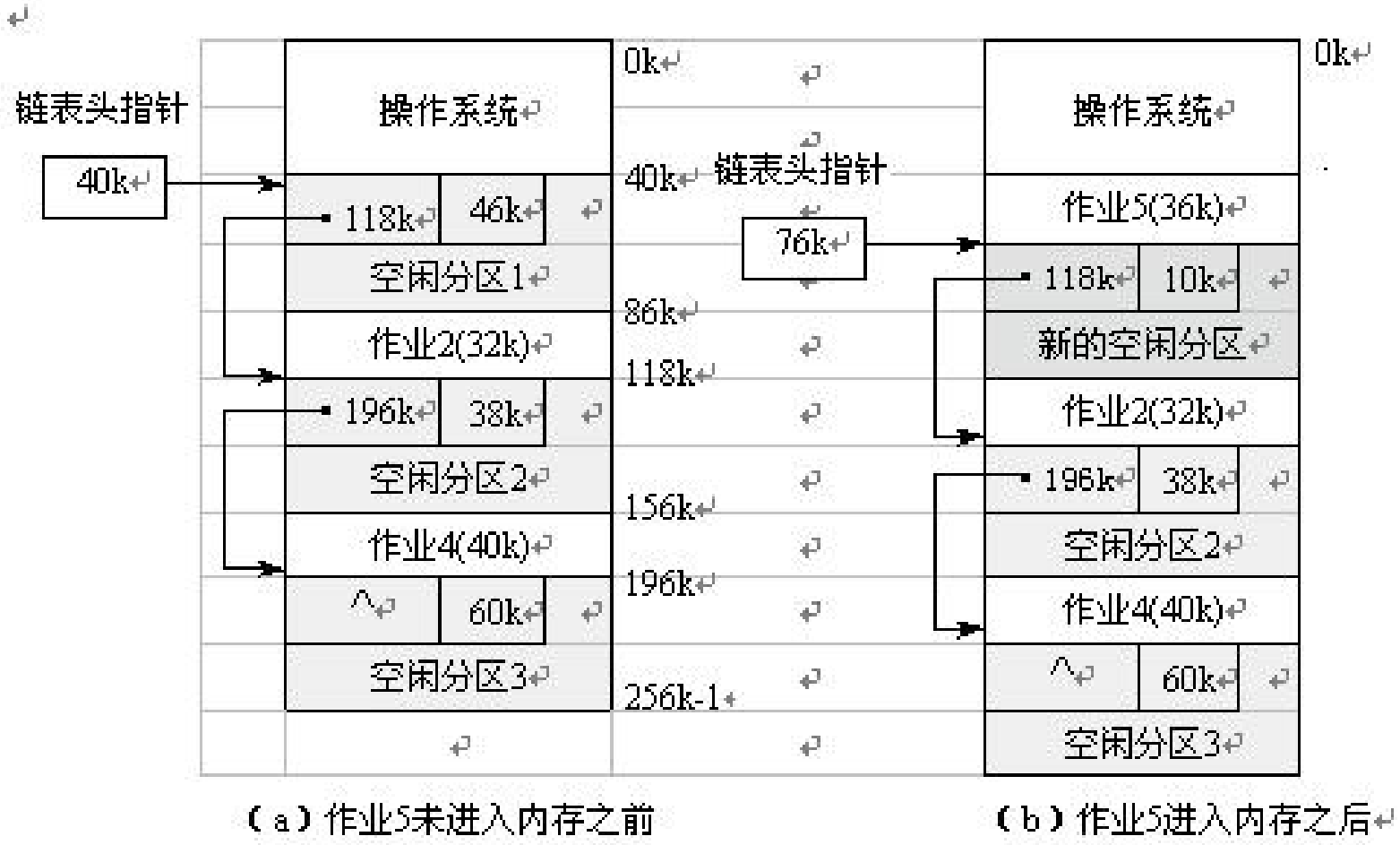
- 动态创建分区：在装入程序时按其初始要求分配，或在执行过程中通过系统调用进行分配或改变分区大小。
- 优点：没有内碎片。
- 缺点：有外碎片；如果大小不是任意的，也可能出现内碎片。

（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


Operating System
Process 1
Process 2
Process 3
320 K
224 K
288 K
64 K
Operating System
Process 1
Process 3
320 K
224 K
288 K
64 K
Operating System
Process 1
Process 4
Process 3
320 K
128 K
96 K
288 K
64 K


Operating System
320 K
Process 4
128 K
96 K
Process 3
288 K
64 K
Operating System
Process 2
224 k
96 K
Process 4
128 K
96 K
Process 3
288 K
64 K

空闲分区的组织形式



（a）作业5未进入内存之前　　（b）作业5进入内存之后

已分配分区表 UBT 表

序号	大小	起址	状态
1	8k	312k	已分配表目
2	32k	320k	已分配表目
3	—	—	空表目
4	120k	384k	已分配表目
5	—	—	空表目

自由分区表 FBT

大小	起址	状态
32k	352k	自由表目
—	—	空表目
520k	504k	自由表目
—	—	空表目
—	—	空表目

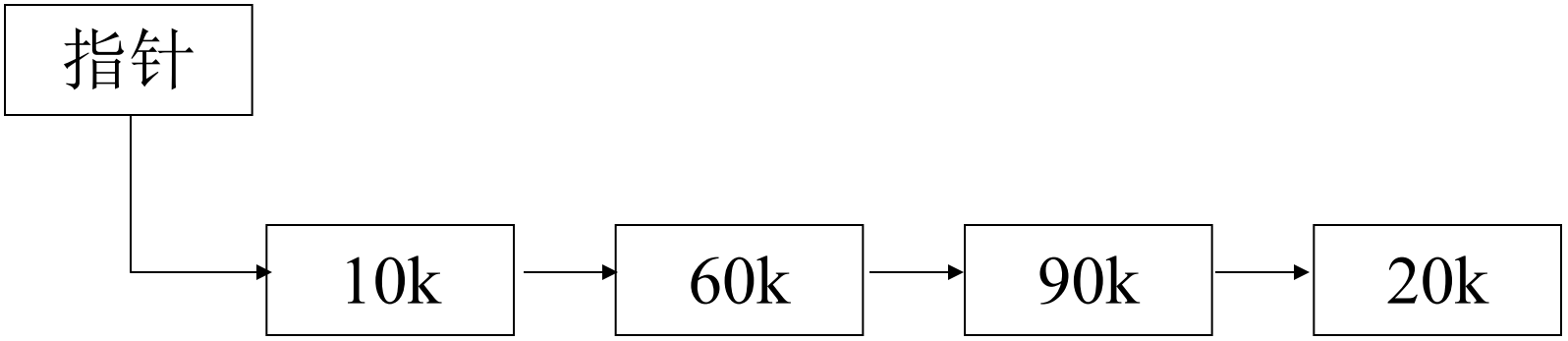
分区分配算法

- 分区分配算法：寻找某个空闲分区，其大小需大于或等于程序的要求。若是大于要求，则将该分区分割成两个分区，其中一个分区为要求的大小并标记为“占用”，而另一个分区为余下部分并标记为“空闲”。分区的先后次序通常是从内存低端到高端。
- 分区释放算法：需要将相邻的空闲分区合并成一个空闲分区。(这时要解决的问题是：合并条件的判断和合并时机的选择)

- **最先匹配法(first-fit)**：按分区的先后次序，从头查找，找到符合要求的第一个分区
  - 该算法的分配和释放的时间性能较好，较大的空闲分区可以被保留在内存高端。
  - 但随着低端分区不断划分而产生较多小分区，每次分配时查找时间开销会增大。
- **下次匹配法(next-fit)**：按分区的先后次序，从上次分配的分区起查找（到最后分区时再回到开头），找到符合要求的第一个分区
  - 该算法的分配和释放的时间性能较好，使空闲分区分布得更均匀，但较大的空闲分区不易保留。
- **最佳匹配法(best-fit)**：找到其大小与要求相差最小的空闲分区
  - 从个别来看，外碎片较小，但从整体来看，会形成较多外碎片。较大的空闲分区可以被保留。
- **最坏匹配法(worst-fit)**：找到最大的空闲分区
  - 基本不留下小空闲分区，但较大的空闲分区不被保留。

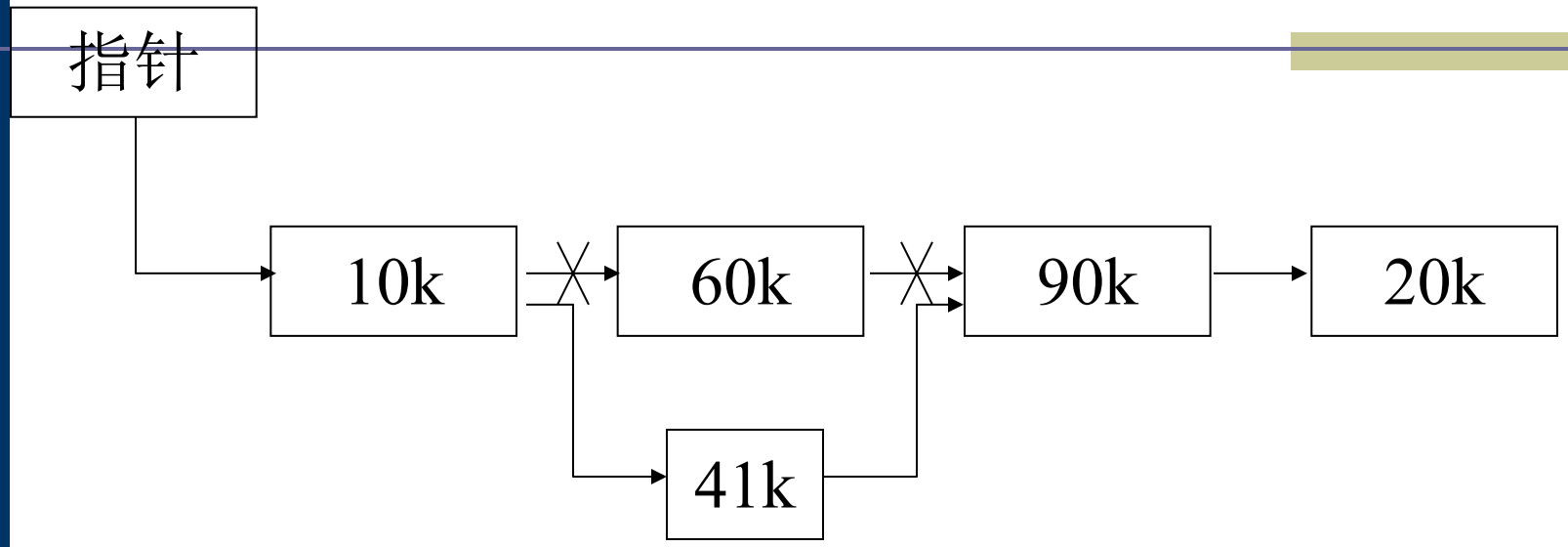
(i)首次适应算法(First Fit: FF)

例：



有四块空白区(从低地址⌫高地址)，来了一个作业需分配19k内存。

解：

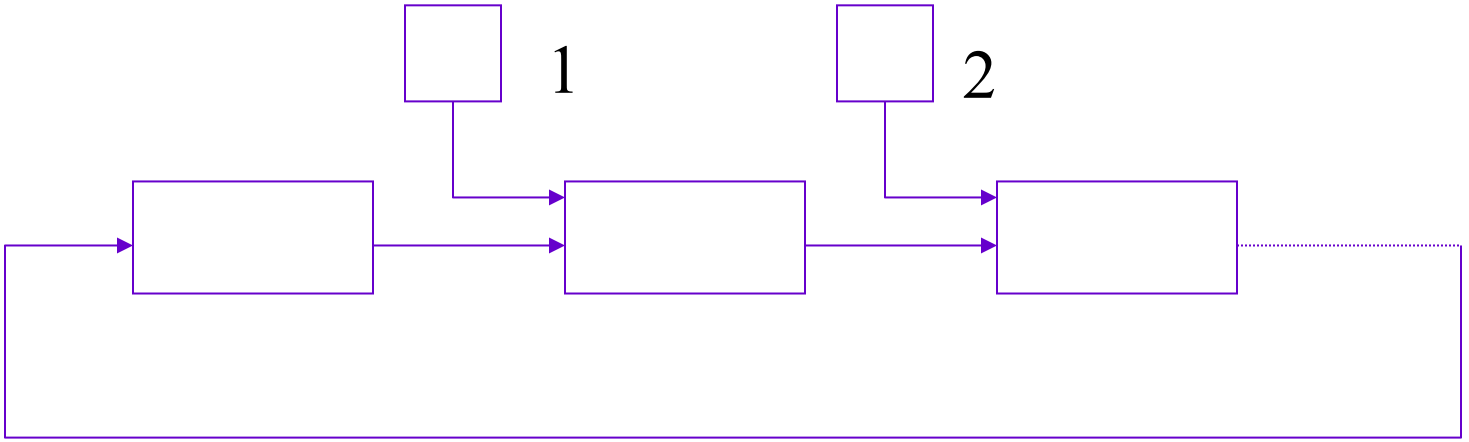


FF特点：

在高地址空白区中保持较大空白区(每次从10k开始分配寻找)。

(ii) 循环首次适应(Next fit: NF)

将空白区组成环状队列，按循环顺序寻找空白区。(与FF区别，头指针从低地址开始向高地址循环移动)



指针移动

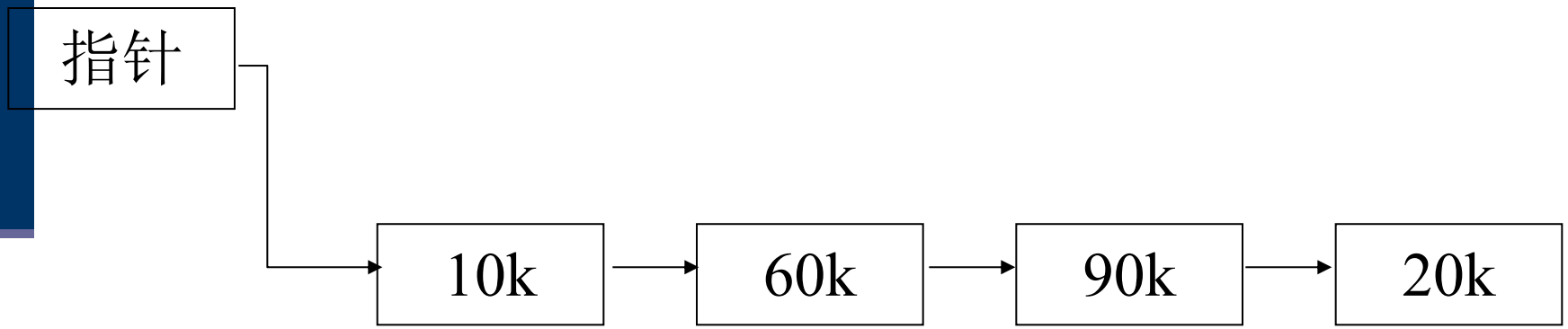
NF特点：

使得小空白区均匀分布，易于与其它空白区合并。

(iii) 最佳适应算法(Best fit: BF)

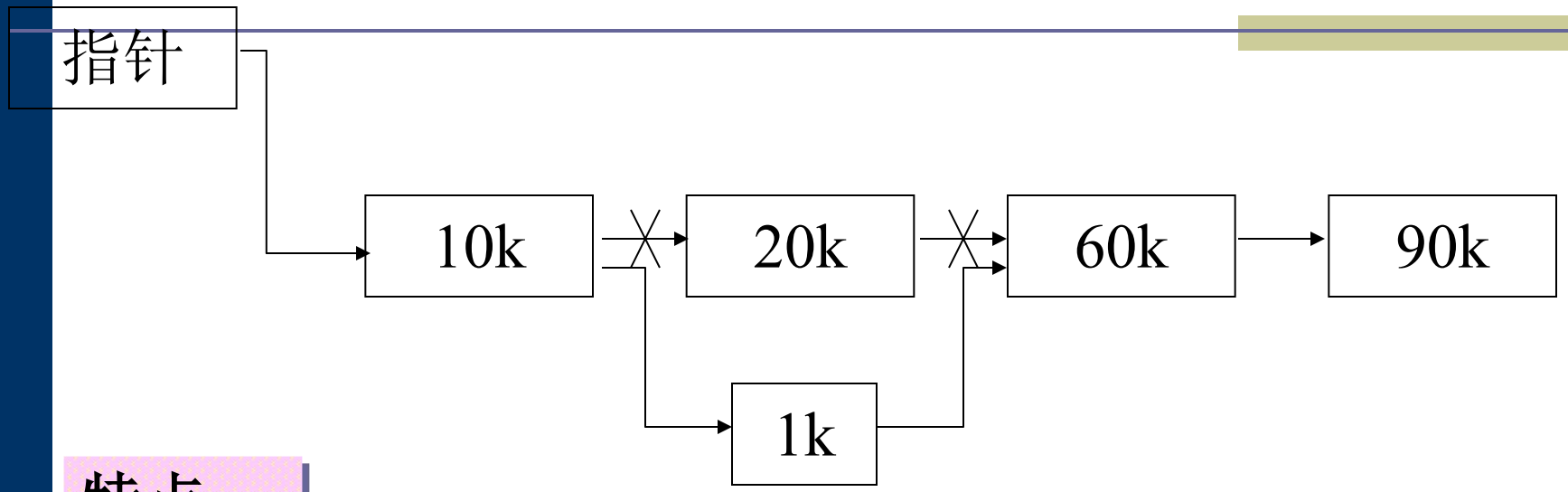
★ 将空白区按大小排成队列，寻找时总是以最小的空白区开始，找到第一个合适的分区

例：



来一个19k的作业

解：



特点：

- ✶ 最佳地利用分区；
- ✸ 开销比较大，并不是最好算法。

(iv) 最坏适应算法(Worst fit: WF)

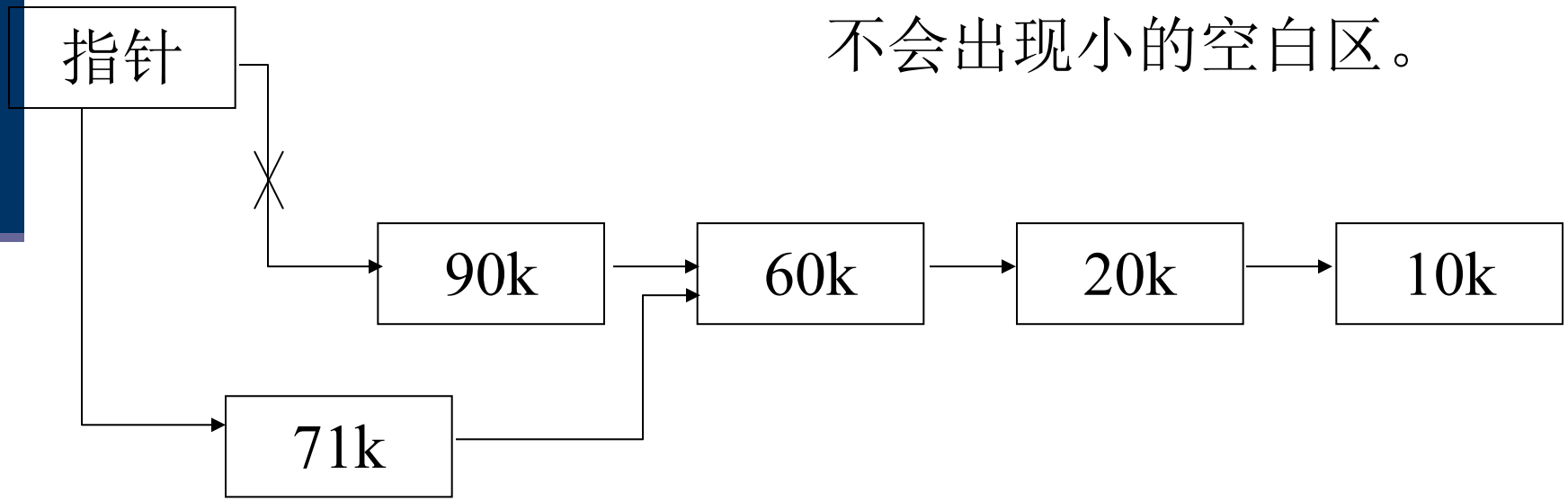
★ 将空白区排序(按从大到小)

✸ 找最大空白区

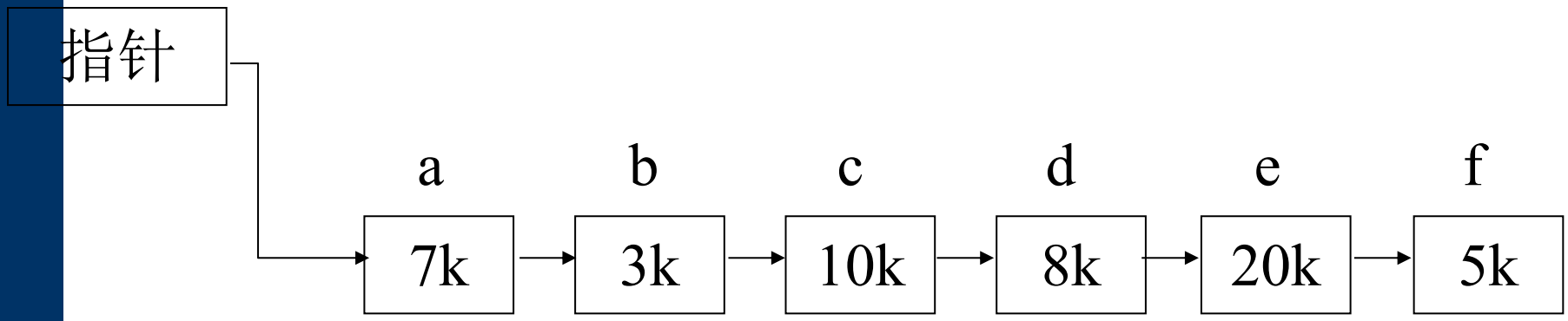
特点：

不会出现小的空白区。

如上例：

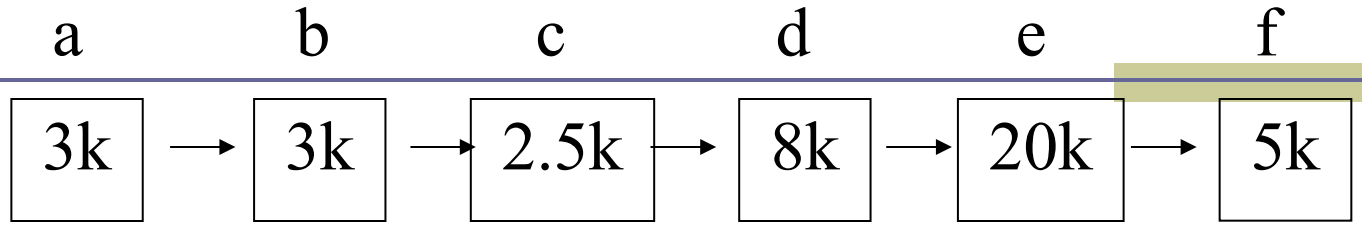


例：设系统空白链表为

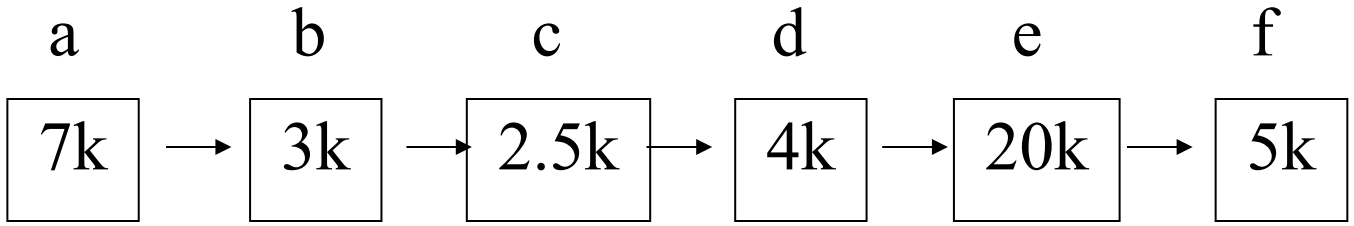


用户先后申请7.5k，4k试用四种算法，试求出分配块。

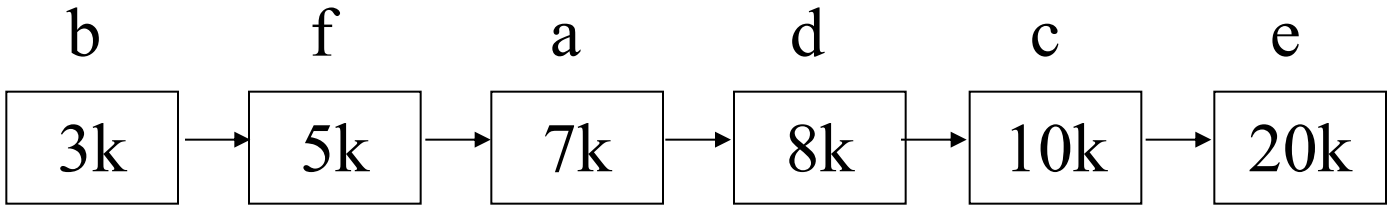
FF: c,a

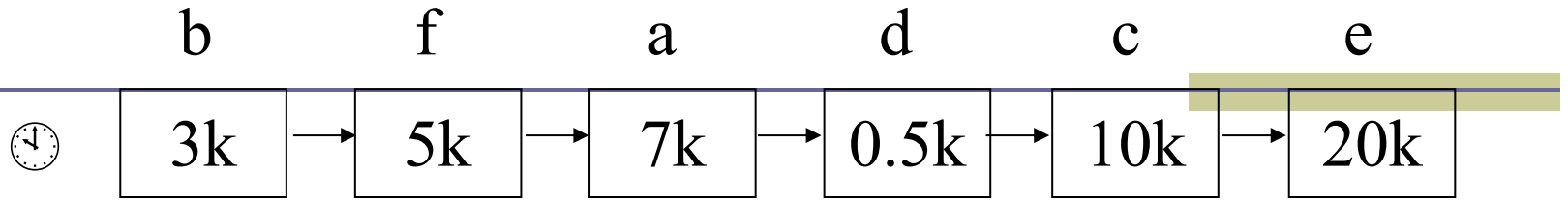


NF: c,d

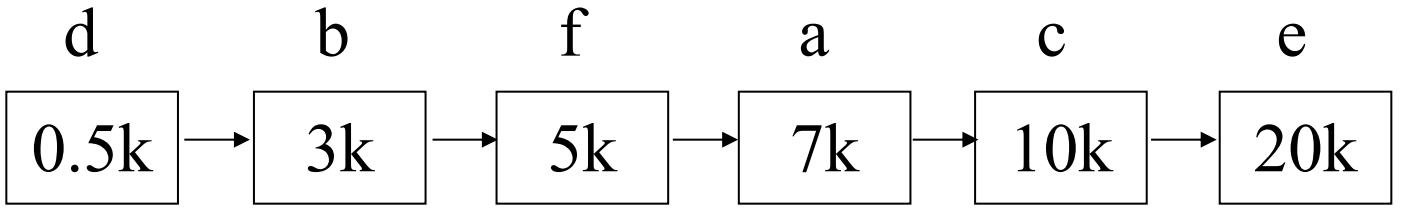


BF:　首先从小到大排序

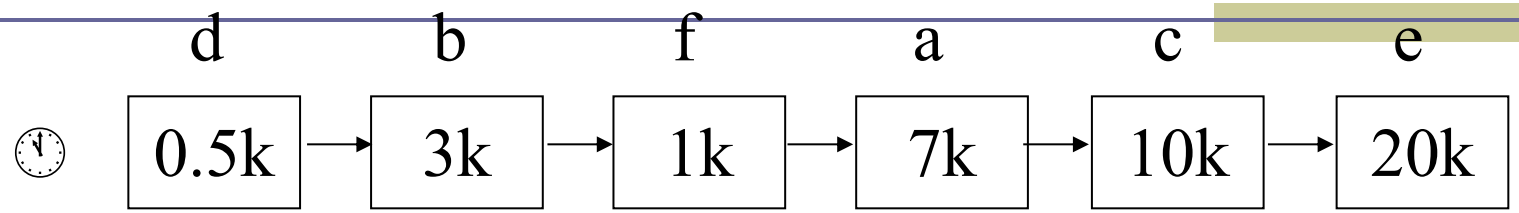




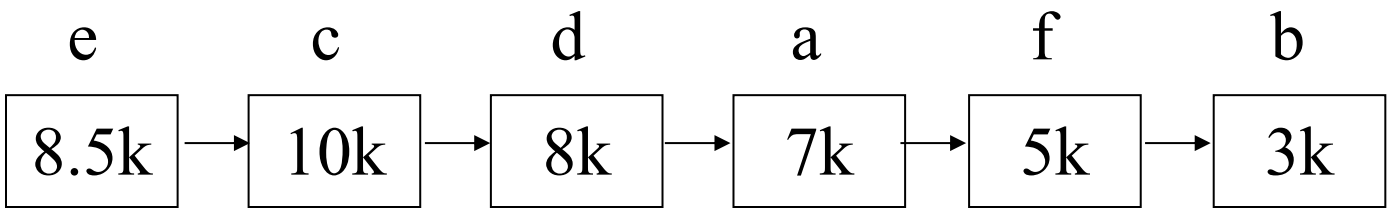
再排序从小到大



∴ 分配块为d, f



WF: e,e



碎片(零头)问题

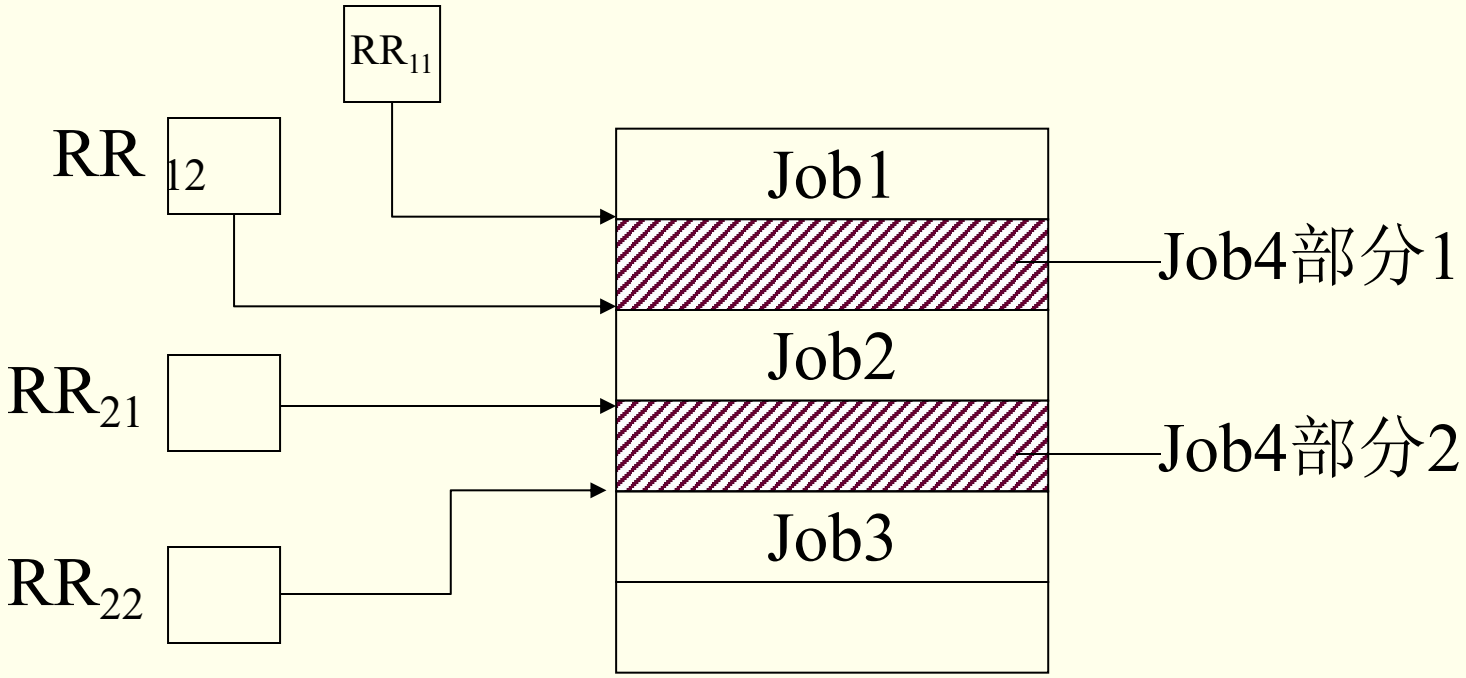
- 存在于已分配的分区之间的一些不能充分利用的空白区

(i) 原因：请求⌫释放⇒使存区分割

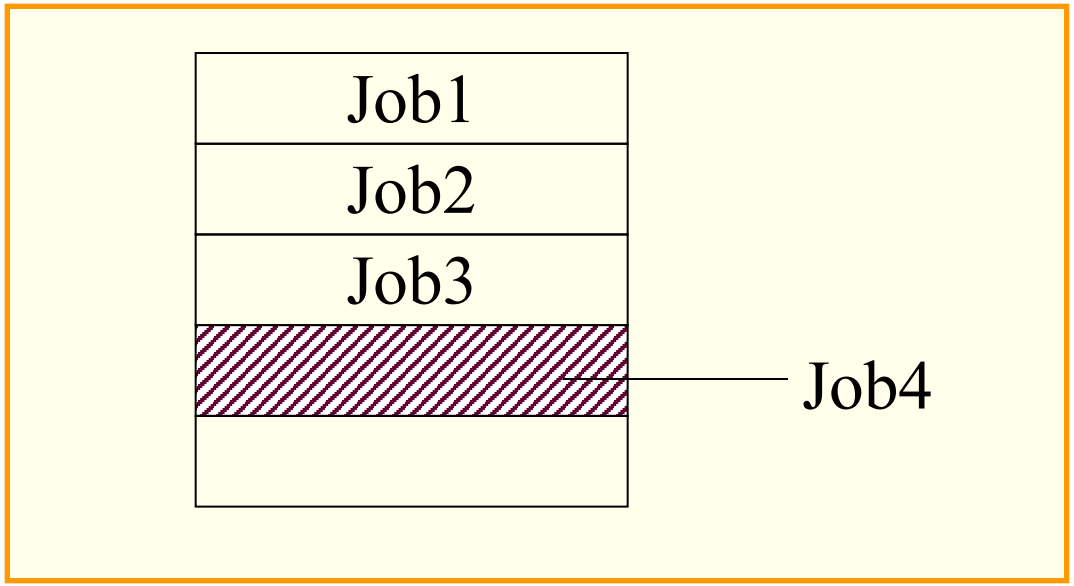
(ii) 碎片总和>nk，但不能装入nk作业

解决的方法：

(I) 将程序装入分散存区中 —— 多重分区

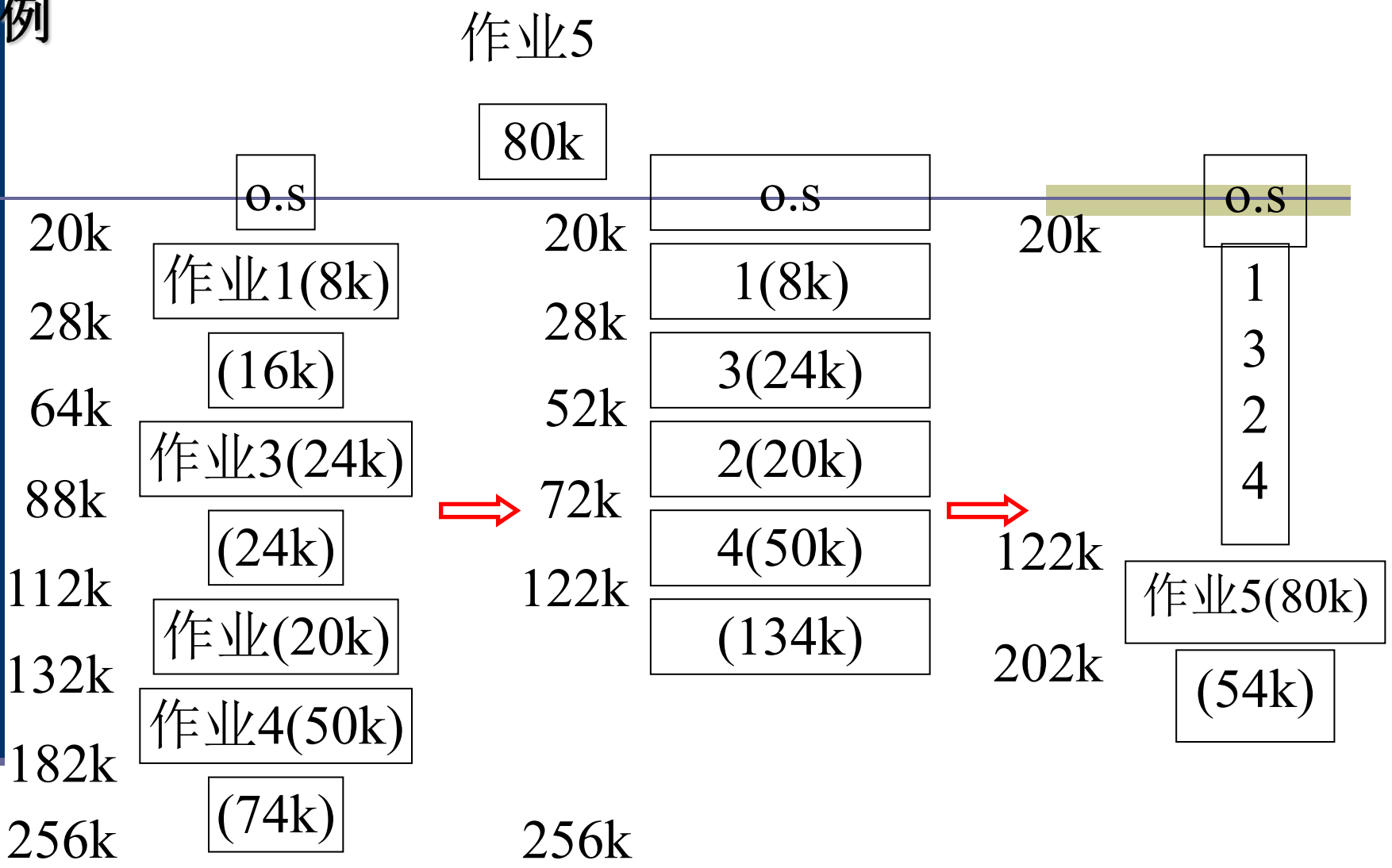


(II) 将碎片集中(紧凑或拼接) —— 可重定位分配



- 移动内存已分配区的信息，使得所有分配区靠在一起使空白区连成一片，采用浮动方法。

例



(a)初始状态

(b)移动之后(即浮动)

(c)分配作业5之后

两大类分区分配类

- 单一连续区
- 固定分区
- 可变分区
- 多重分区
- 可重定位分区

固定分区、可变分区、多重分区、可重定位分区：多用户

多重分区、可重定位分区：解决零头

MS DOS中的分区存储管理

- DOS提供动态分区管理；
- 通过DOS功能调用int 21h，支持分区的创建(48h)、释放(49h)和改变分区大小(4Ah)
- 设置或查询分区的分配策略(58h)：
  - 最先匹配法、
  - 最佳匹配法、
  - 最后匹配法（last-fit，从内存高端向低端查找）

数据结构为单向链表。每个分区以一个MCB(Memory Control Block)结构开始，MCB占16个字节，按段边界对齐（起始地址可被16整除）

PSP(Program Segment Prefix)：起进程控制块的作用，其起始地址可作为进程ID

```
typedef struct {
  BYTE type;          /* 'M' = in chain; 'Z' = at end  */
  WORD owner;         /* PSP of the owner process, 0 = free */
  WORD size;          /* in 16-byte paragraphs */
  BYTE unused[3];
  BYTE dos4[8];       /* program name(DOS 4.x)    */
} MCB;
```

What you need to do?

- 复习课本4.1、4.2节的内容

See you next time!