



北京航空航天大学

Beijing University of Aeronautics and Astronautics



OS2019理论课-姜博



该二维码7天内(3月14日前)有效，重新进入将更新



北京航空航天大学

Beijing University of Aeronautics and Astronautics

三、存储器管理

2018年3月7日

内容提要

- § 存储管理基础
- § 页式内存管理
- § 段式内存管理
- § 虚拟存储管理
- § 存储管理实例

内容提要

§ 存储管理基础

- 存储器硬件发展
- 存储管理的功能
 - C程序实例分析(MIPS)
- 存储器分配方法
 - 单一连续
 - 分区管理：固定分区、可变分区
 - 覆盖、交换

§ 页式内存管理

§ 段式内存管理

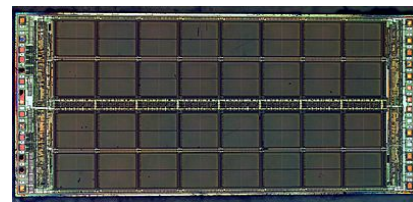
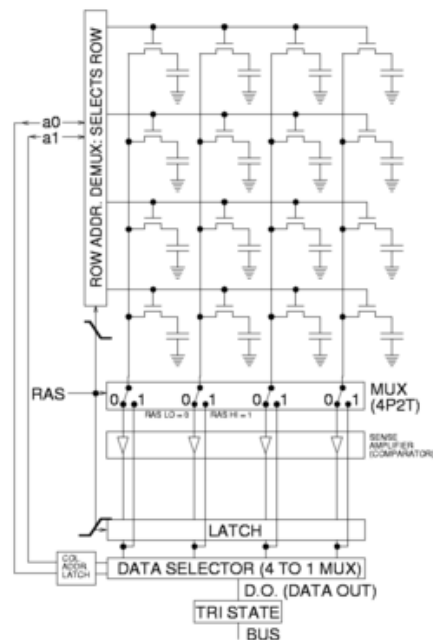
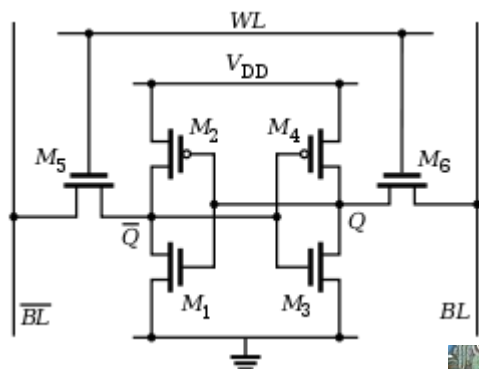
§ 虚拟存储管理

§ 存储管理实例

存储器硬件

§ 存储器的功能：保存数据，存储器的发展目标是高速、大容量和小体积。

- DDR4理论上每根DIMM模块能达到512GiB的容量
- DDR4-3200带宽可达51.2GB/s



存储器硬件

- § 静态存储器（SRAM）：读写速度快，生产成本低，多用于容量较小的高速缓冲存储器。- 用于做Cache
- § 动态存储器（DRAM）：读写速度较慢，集成度高，生产成本低，多用于容量较大的主存储器。

DRAM的一个重要特点是，数据以电荷的形式保存在电容，电容的放电使得电荷通常只能维持几个毫秒左右相当于1M个时钟周期左右，因此要定期进行刷新（读出后重新写），按行进行（所有芯片中的同一行一起进行，刷新操作所需时间通常只占1%~2%左右。

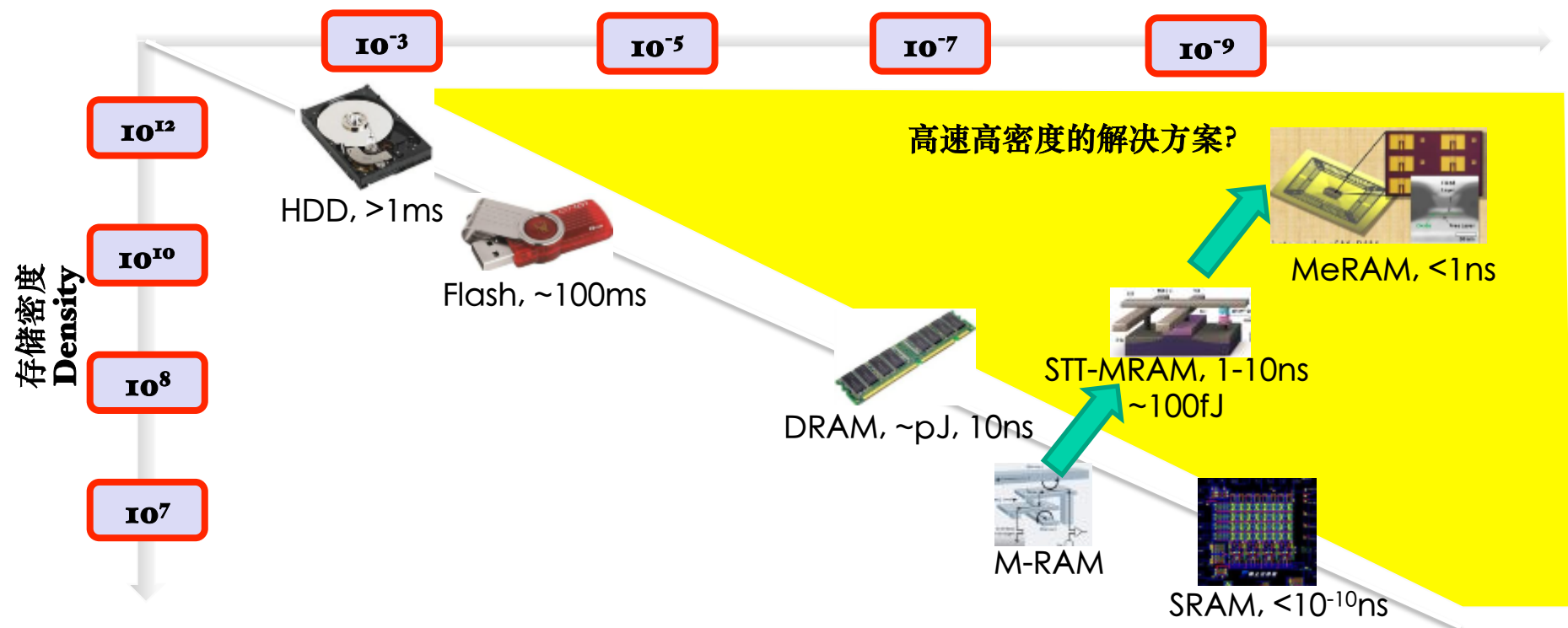
存储器硬件

	SRAM	DRAM
存储信息方式	触发器 (RS)	电容
破坏性读出	否	是 (读后再生)
定期刷新	不需要	需要
送地址方式	行列同时送	行列分两次送
运行速度	快	慢
发热量	大	小
存储成本	高	低
集成度	低	高

蕴含巨大的挑战

- 访问速度和存储密度不可兼得
- 非易失性存储器件 (NVM) 的发展是否有突破的机会?
- SSD (Flash)、PCM、忆阻器、自旋电子器件

访问速度 Speed (S)

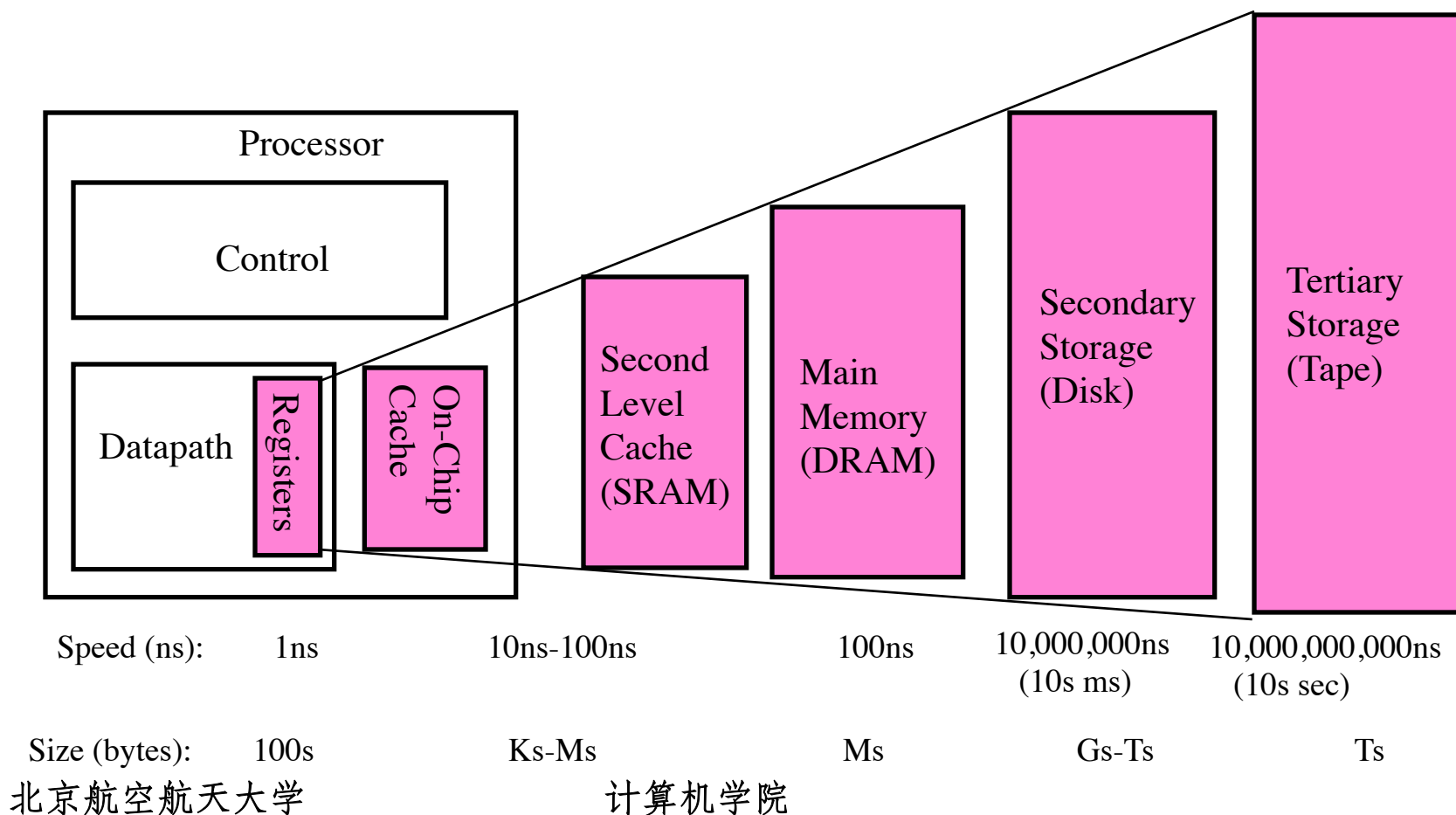


K.L.Wang, Spin orbit interaction engineering of magnetic memory for energy efficient electronics systems. **NVMTS 2015**

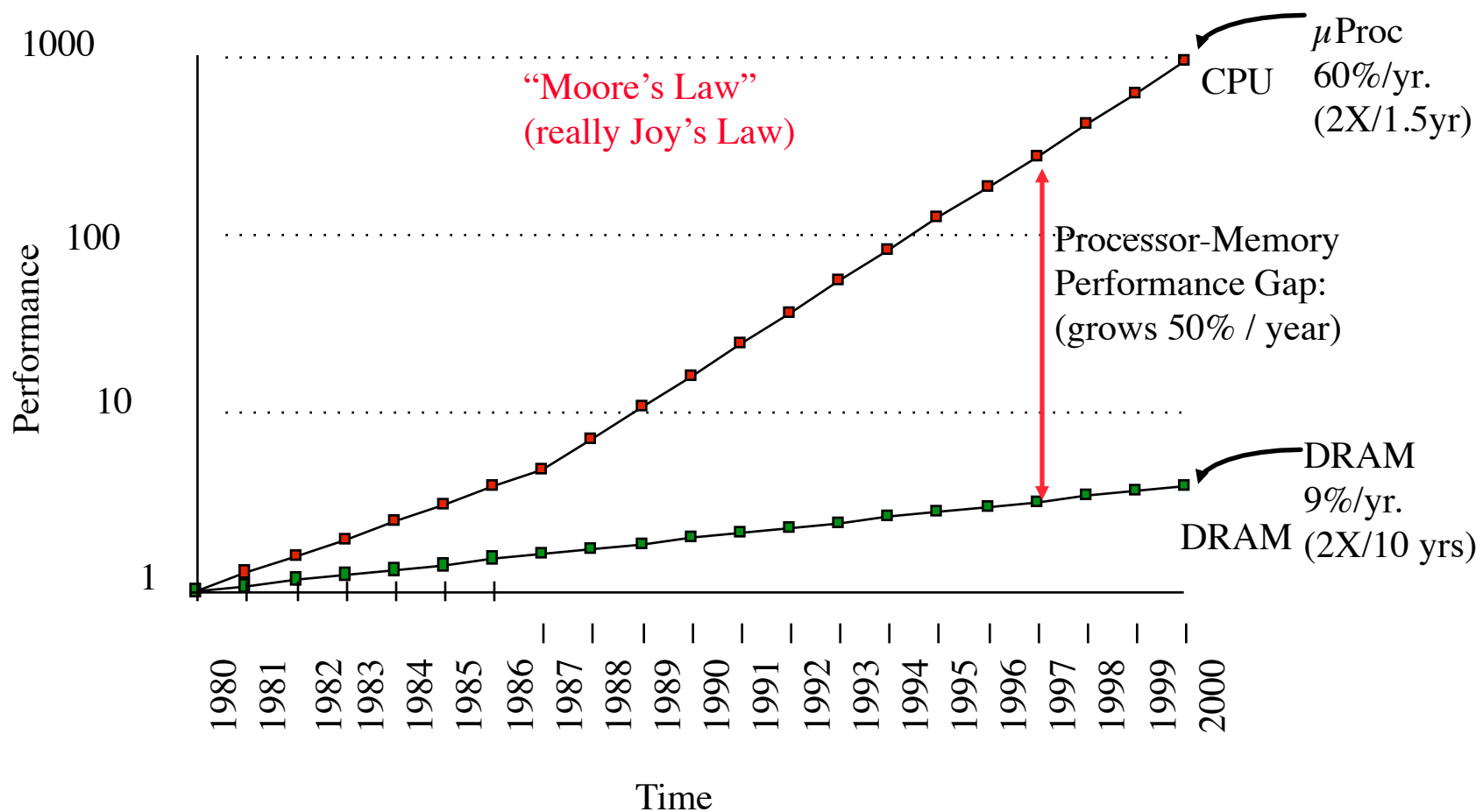
存储组织

- § 存储组织的功能是在存储技术和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”结构和“寄存器-缓存-内存-外存”结构；
- § 现在微机中的存储层次组织：访问速度越来越慢，容量越来越大，价格越来越便宜；
- § 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）；

存储层次结构



Processor-DRAM Memory Gap (latency)



两个基本概念

- § 1. 地址空间：源程序经过编译后得到的目标程序，存在于它所限定的逻辑地址范围内，这个范围称为地址空间。简言之，地址空间是逻辑地址的集合。
- § 2. 存储空间：存储空间是指主存中一系列存储信息的物理单元的集合，这些单元的编号称为物理地址或绝对地址。简言之，存储空间是物理地址的集合。

为什么进行存储管理

§ 重要的资源

§ 帕金森定律(Parkinson):

- 一个人做一件事所耗费的时间差别如此之大：他可以在10分钟内看完一份报纸，也可以看半天；一个忙人20分钟可以寄出一叠明信片，但一个无所事事的老太太为了给远方的外甥女寄张明信片，可以足足花一整天：找明信片一个钟头，寻眼镜一个钟头，查地址半个钟头，写问候的话一个钟头零一刻钟.....
- 工作会自动占满一个人所有可用的时间。如果一个人给自己安排了充裕的时间去完成一项工作，他就会放慢节奏或者增加其他项目以便用掉所有的时间。
- 工作膨胀出来的复杂性会使工作显得很重要，在这种时间弹性很大的环境中工作并不会感到轻松。相反会因为工作的拖沓、膨胀而苦闷。

为什么进行存储管理

§ 软件对存储的要求也类似：

- 存储器有多大，程序对存储的需要更大。
- 如果物理存储没有管理，软件会无节制的占用所有资源
- PC机的内存：256M-16G
- 手机的内存：256M-16G
- 还是不够用。。。。

存储管理的需求

§ 支持多道程序和多用户系统：

- 针对多个应用，将内存划分多个区域

§ 充分利用内存

§ 方便用户使用：OS自动加载程序，对用户透明

§ 较大的逻辑运行空间：不局限于物理内存大小

§ 存储保护与共享

存储管理的功能

- § 存储分配和回收：是存储管理的主要内容。讨论其算法和相应的数据结构。
- § 存储共享和保护：包括代码和数据共享，对地址空间的访问权限控制（读、写、执行）。
- § 存储器容量扩充：
 - 由应用程序控制的方案：覆盖；
 - 由OS控制的方案：交换（整个进程空间），请求调入和预调入（部分进程空间）

存储管理的功能-分配和回收

- § 当用户请求内存时，及时响应，分配内存
- § 当用户不需要时及时回收内存，让其他用户使用
- § 这就要求：
 - 用数据结构，记住内存分配状态
 - 分配：用户请求，分配内存，修改数据结构
 - 回收：用户释放内存，修改数据结构

存储管理的功能-扩充容量

§ OS要方便用户使用-用户编程不需要考虑内存的容量

- 需要扩充内存容量，用户可以用更大的空间

§ 解决方案

- 在MMU内存管理单元的控制下
 - 交换技术
 - 虚拟存储技术:把内存和外存统一使用，扩充内存

存储管理的功能-共享

§ 让多个进程共用内存中的相同区域

§ 代码共享

- 节省内存空间

§ 数据共享

- 实现进程的通讯

存储管理的功能-保护

§ 让OS，多道程序同时运行又不互相干扰

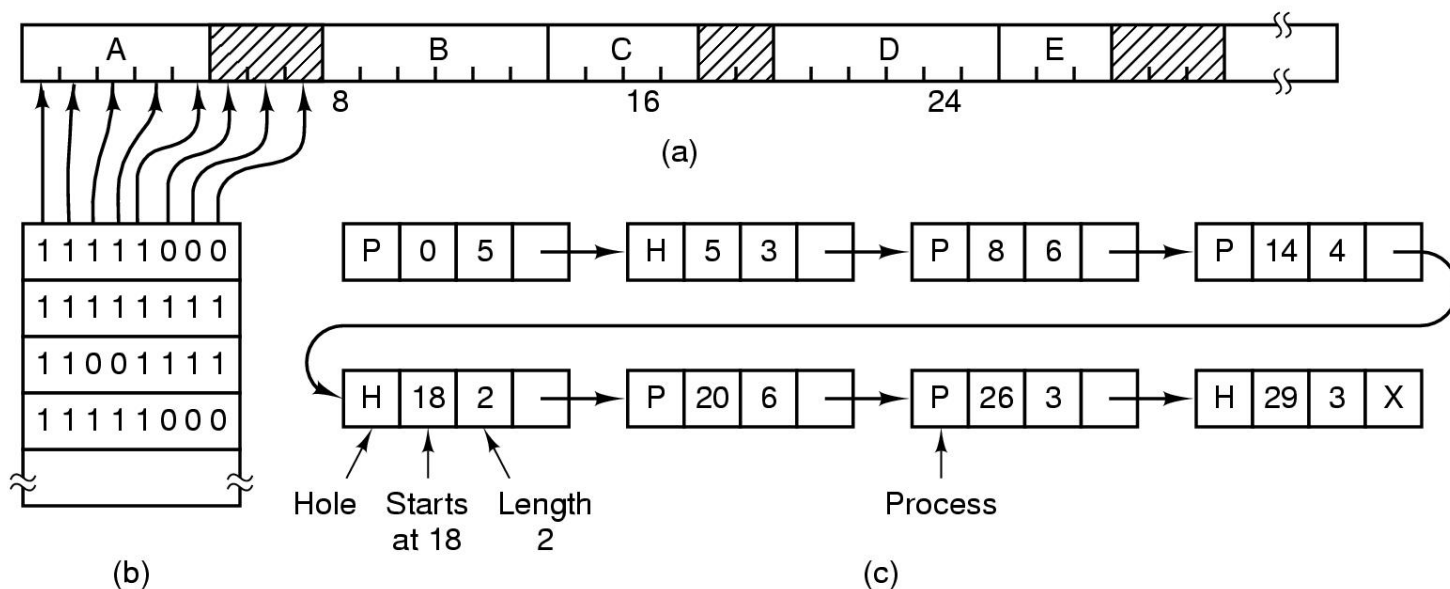
- 一个程序只能访问自己的区域
- 一个程序错误不影响其他程序，防止破坏系统

§ 存储保护的内容

- 地址越界保护
 - 进行地址越界可能会影响其他程序或者OS
 - 越界---产生中断，交给OS处理
- 权限保护
 - 共享区域，权限不同
 - 读写权限保护

闲置空间的管理-数据结构

§ 在管理内存的时候，OS需要知道内存空间有多少空闲？这就必须跟踪内存的使用，跟踪的办法有两种：位图表示法（分区表）和链表表示法（分区链表）



存储管理的数据结构

§ 链表

§ 位图

位图表示法

§ 给每个分配单元对应一位，用来记录该分配单元是否闲置。例如，位取值为0表示单元闲置，取值为1则表示已被占用，这种表示方法就是位图表示法。

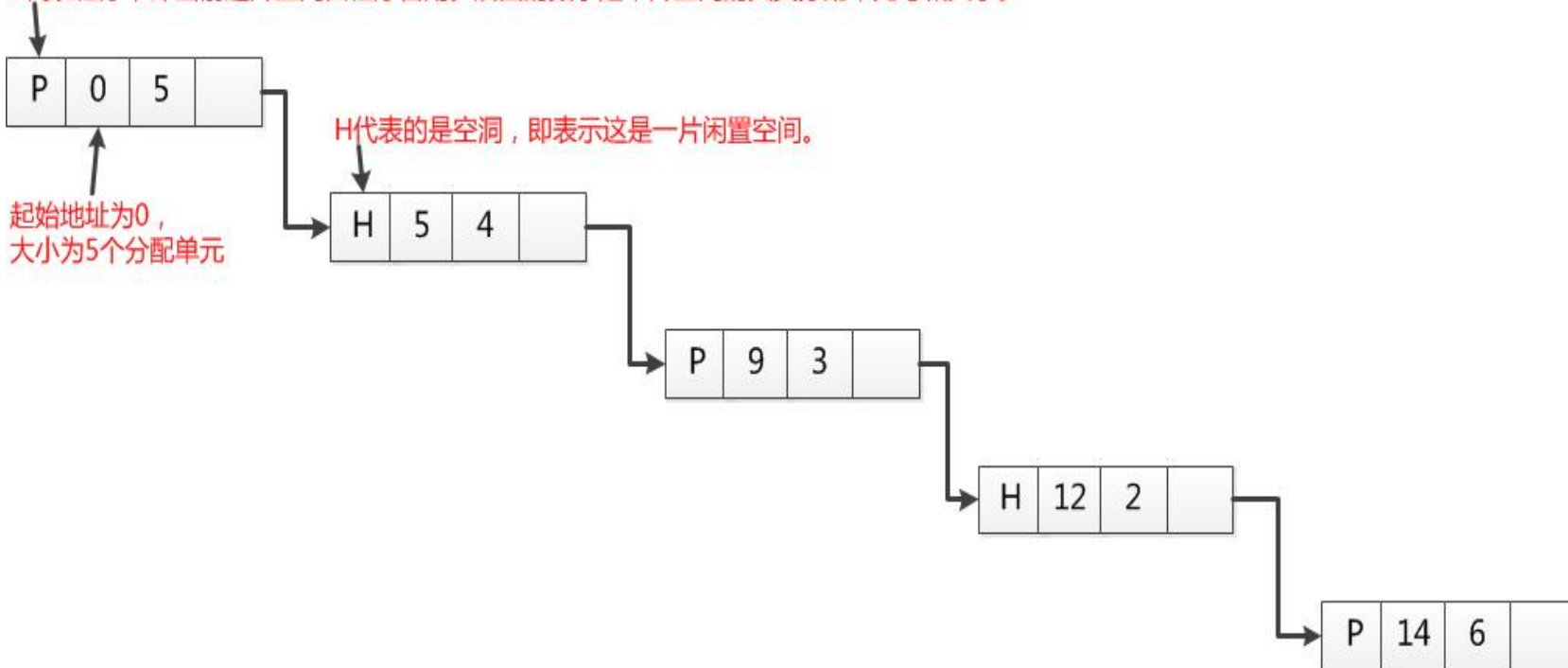
1	1	1	1	1	1	0	0	0	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---

内存分配位图表示

链表表示法

§ 将分配单元按照是否闲置链接起来，这种方法称为链表表示法。如上图所示的位图所表示的内存分配状态，使用链表来表示的话则会如下图所示

P代表程序，即当前这片空间由程序占用。后面的数字是本片空间的其实分配单元号和大小。



两种方法的特点

§ 位图表示法：

- 空间成本固定：不依赖于内存中的程序数量。
- 时间成本低：操作简单，直接修改其位图值即可。
- 没有容错能力：如果一个分配单元为1，不能肯定应该为1还是因错误变成1。

§ 链表表示法：

- 空间成本：取决于程序的数量。
- 时间成本：链表扫描通常速度较慢，还要进行链表项的插入、删除和修改。
- 有一定容错能力：因为链表有被占空间和闲置空间的表项，可以相互验证。

练习：比较位图和链表的存储

§ 128-MB的内存，以 n 个字节为单位进行分配。对于链表表示法，假设内存包含着交替的数据区和空闲区，每个区均为64K。假设链表的每个结点需要32位的内存地址，16位的长度和16位的指向下一个节点的指针。对于每种方法，分别需要多少存储空间？那种方法更好？

练习：比较位图和链表的存储

§ 对于位图来说，每个内存分配单元需要1位。128MB = 2^{27} ，n 个字节为一个内存分配单元，那么128M对应着 $2^{27} / n$ 个内存分配单元。所以位图需要 $2^{27} / n$ 位，等于 $2^{24} / n$ 字节。

练习：比较位图和链表的存储

- § 对于位图来说，每个内存分配单元需要1位。128MB = 2^{27} ，n 个字节为一个内存分配单元，那么128M对应着 $2^{27} / n$ 个内存分配单元。所以位图需要 $2^{27} / n$ 位，等于 $2^{24} / n$ 字节。
- § 对于链表来说，需要 $128\text{M} / 64\text{K} = 2^{27} / 2^{16} = 2^{11}$ 个节点。每个节点64位，也就是8个字节。所以需要 $2^{11} * 2^3 = 2^{14}$ 字节。

练习：比较位图和链表的存储

§ 对于位图来说，每个内存分配单元需要1位。128MB = 2^{27} ，n 个字节为一个内存分配单元，那么128M对应着 $2^{27} / n$ 个内存分配单元。所以位图需要 $2^{27} / n$ 位，等于 $2^{24} / n$ 字节。

§ 对于链表来说，需要 $128\text{M} / 64\text{K} = 2^{27} / 2^{16} = 2^{11}$ 个节点。每个节点64位，也就是8个字节。所以需要 $2^{11} * 2^3 = 2^{14}$ 字节。

§ 所以是比较 $2^{24} / n$ 和 2^{14}

§ 当n比较小，链表更加节省空间。当n比较大，位图更好。n取值的平衡点是 2^{10} ，也就是1 KB。所以，当 n 小于 1 KB，链表更好。反之，位图更好。同时， $n \leq 64 \text{ KB}$ ，因为数据和空闲区是 64

存储分配的三种方式

- § 1.直接指定方式：程序员在编程序时,或编译程序(汇编程序)对源程序进行编译(汇编)时,所用的是实际地址。
- § 2.静态分配(Static Allocation)：程序员编程时,或由编译程序产生的目的程序,均可从其地址空间的零地址开始；当加载(loader)程序对其进行链接装入时才确定它们在主存中的地址。
- § 3.动态分配(Dynamic Allocation)：编程采用逻辑地址空间，从0地址开始。作业在存储空间中的位置,在其装入时确定,在其执行过程中可动态申请或释放空间。

连续分配存储管理方式

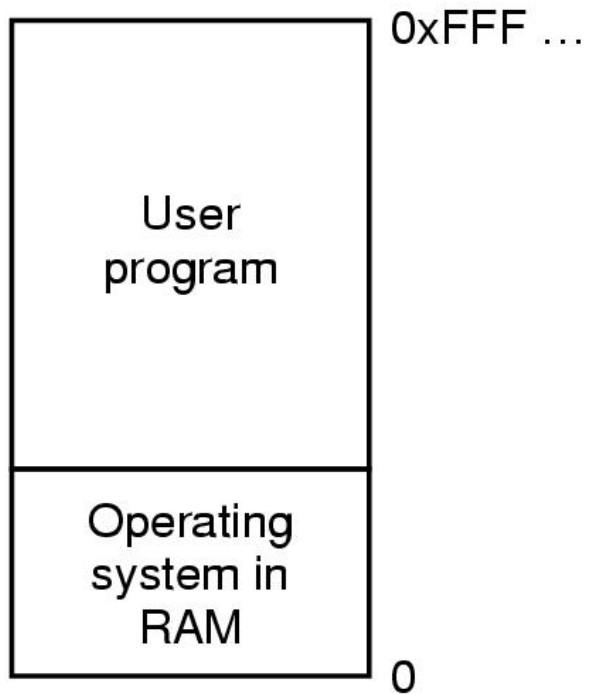
§ 单一连续分配方式

§ 分区分配方式

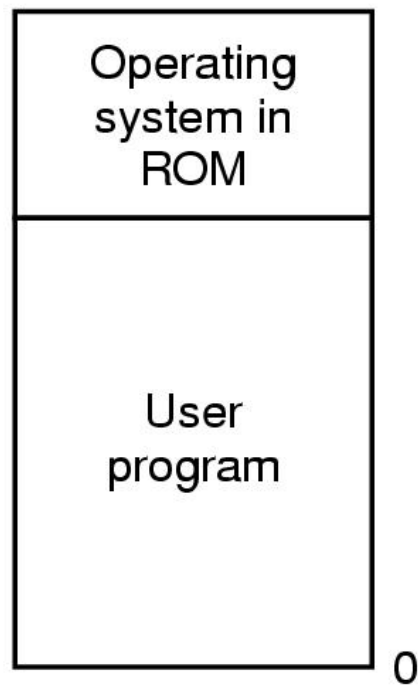
- 固定分区
- 动态分区

单一连续区存储管理

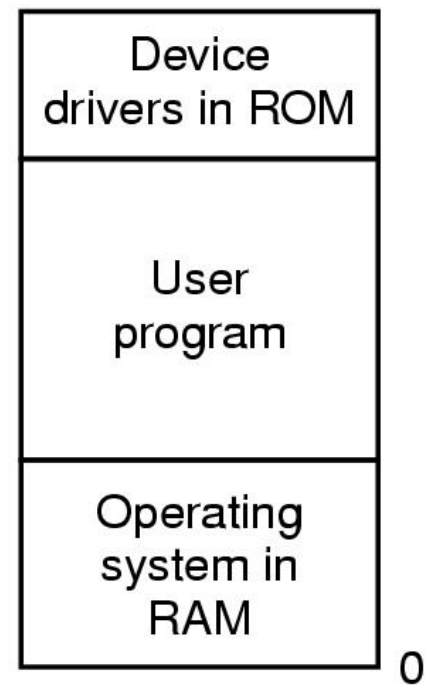
- § 内存分为两个区域：系统区，用户区。应用程序装入到用户区，可使用用户区全部空间。
- § 最简单，适用于单用户、单任务的OS。CP/M和DOS
- § 优点：易于管理。
- § 缺点：对要求内存空间少的程序，造成内存浪费；程序全部装入，很少使用的程序部分也占用内存。



(a)



(b)



(c)

多用户系统存储器管理---分区式分配

- § 把内存分为一些大小相等或不等的分区(partition)，每个应用程序占用一个或几个分区。操作系统占用其中一个分区。
- § 适用于多道程序系统和分时系统，支持多个程序并发执行，但难以进行内存分区的共享。
 - 进程属于不同分区

多用户系统存储器管理---分区式分配

§ 分区式分配的问题：内碎片和外碎片

- 内碎片：分区内部难以利用的空间
- 外碎片：分区之间难以利用的空间

§ 分区式管理的方法：

- 数据结构：分区表或者分区链表
- OS负责维护数据结构

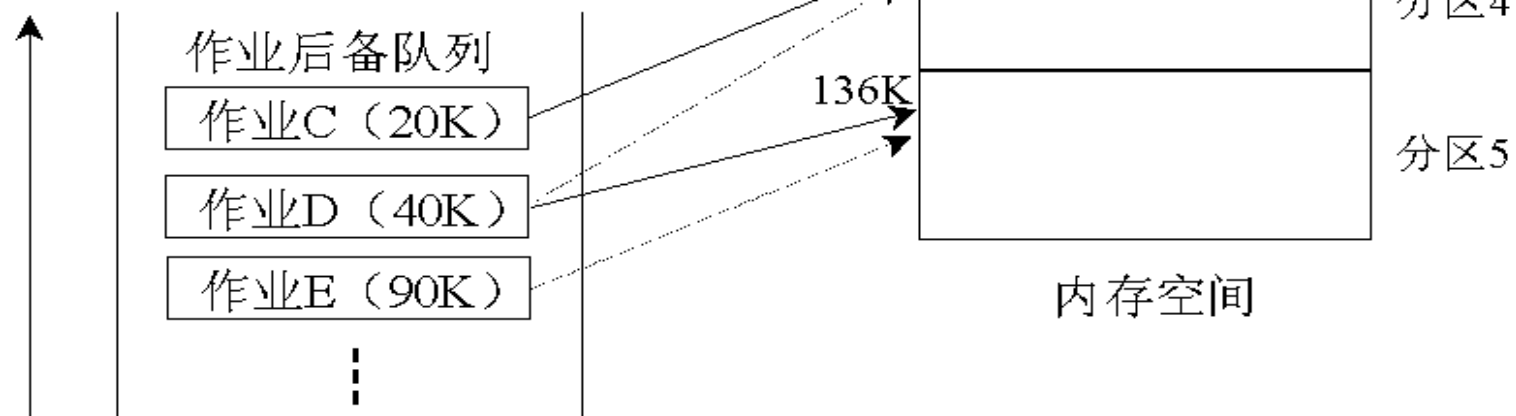
固定式分区

- § 固定式分区（静态存储区域）：当系统初始化时，把存储空间划分成若干个任意大小的固定区域；然后，把这些区域分配给每个用户作业。
- § 把内存划分为若干个固定大小的连续分区。
 - 分区大小相等：只适合于多个相似程序的并发执行（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。

固定式分区

分区说明表

区号	大小	起始地址	状态
1	8K	16K	占用
2	16K	24K	空闲
3	32K	40K	占用
4	64K	72K	空闲
5	120K	136K	占用

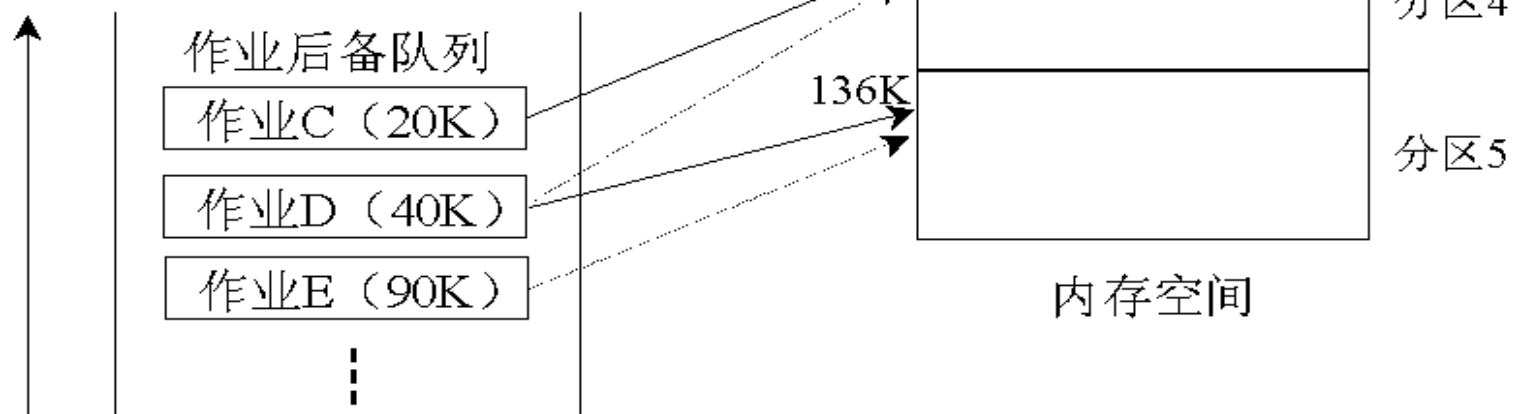


固定式分区

作业A小于分区1大小8K
产生内碎片

分区说明表

区号	大小	起始地址	状态
1	8K	16K	占用
2	16K	24K	空闲
3	32K	40K	占用
4	64K	72K	空闲
5	120K	136K	占用



§ 优点：易于实现，开销小。

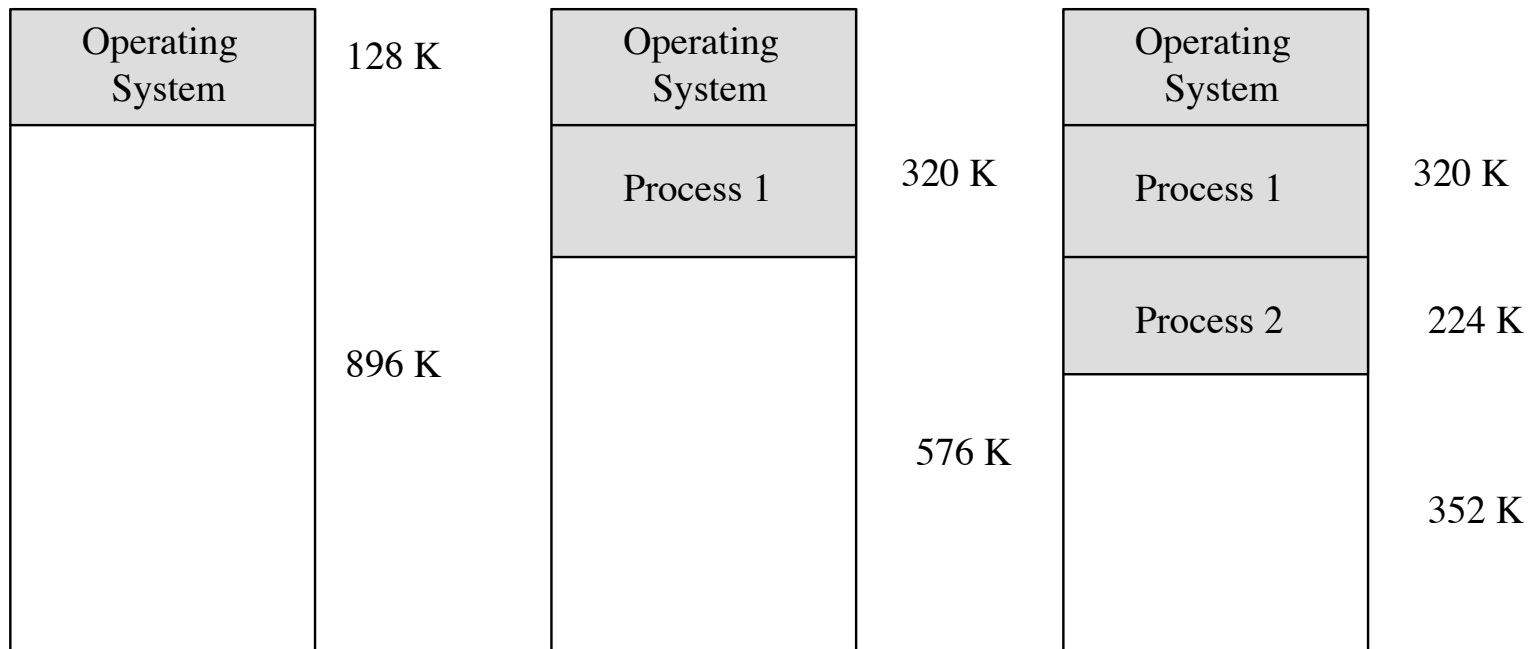
§ 缺点：

- 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；
- 灵活性差，接受程序大小受分区大小限制

§ 采用的数据结构：分区表——记录分区的大小和使用情况

可变式分区：

- § 可变式分区：分区的边界可以移动，即分区的大小可变。
- § 分区大小在程序装入时大小动态确定，量身定制
 - 也可以通过系统调用改变分区大小
- § 优点：没有内碎片。
- § 缺点：随着多次分配和回收，产生外碎片。



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

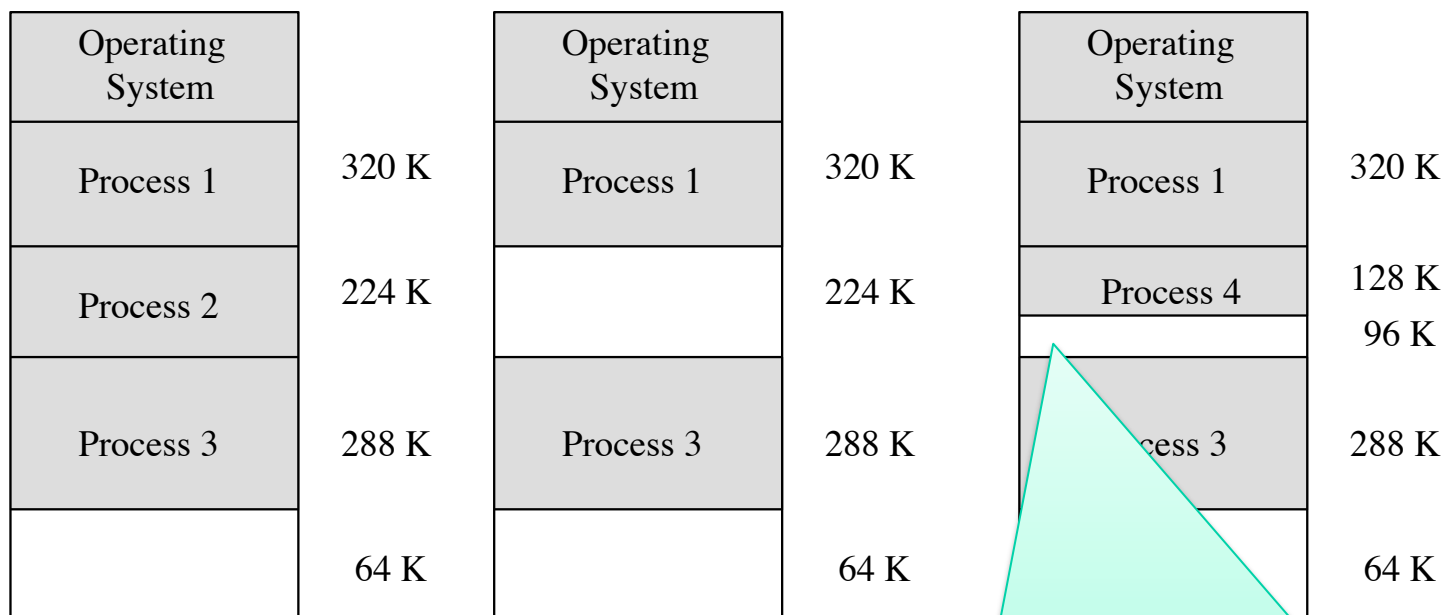
Operating System
Process 1
Process 4
Process 3

320 K

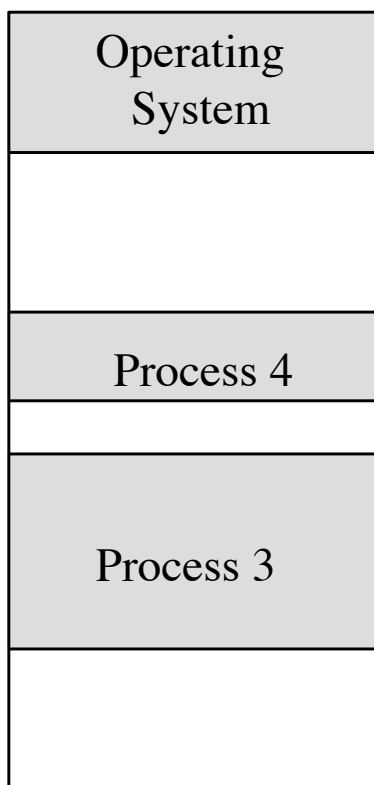
128 K
96 K

288 K

64 K



进程4小于进程2释放的空间，产生外碎片



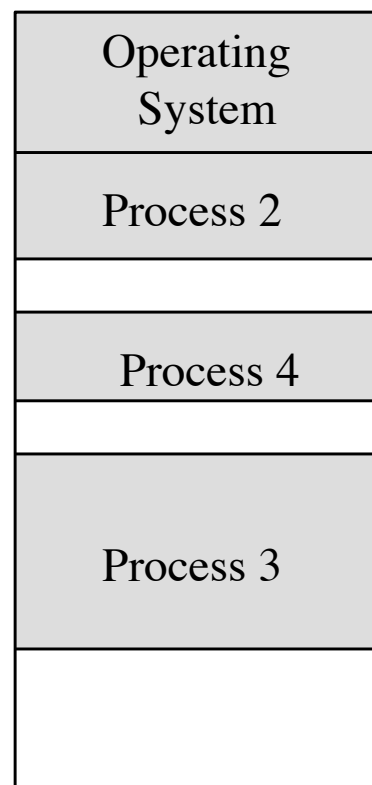
320 K

128 K

96 K

288 K

64 K



224 k

96 K

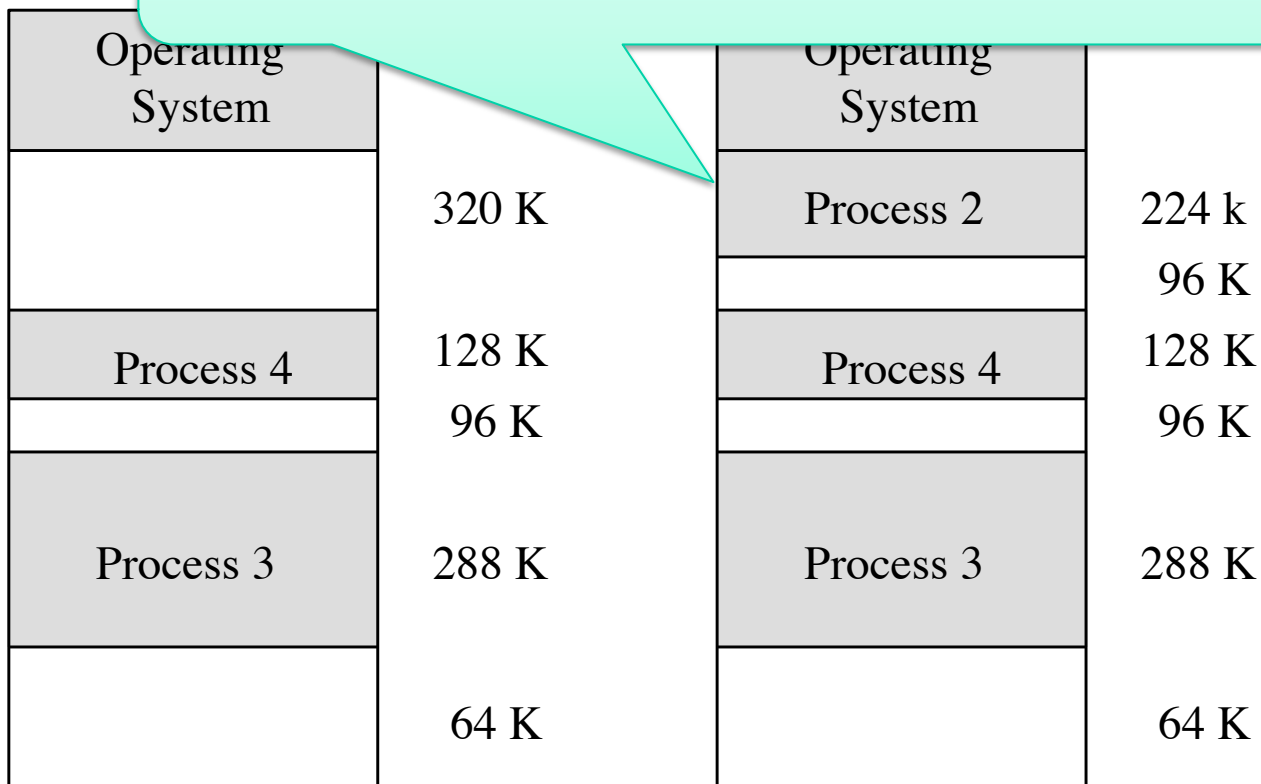
128 K

96 K

288 K

64 K

进程2小于进程1释放的空间，产生外碎片



可变分区管理的数据结构

§ 需要设置数据结构记录内存分配情况：

§ 内存分配表

- 已分配区表
- 空闲区表

动态分区的数据结构

起始地址	长度	标志
500	800	P1
1500	400	P2
		空
	⋮	

已分配区表

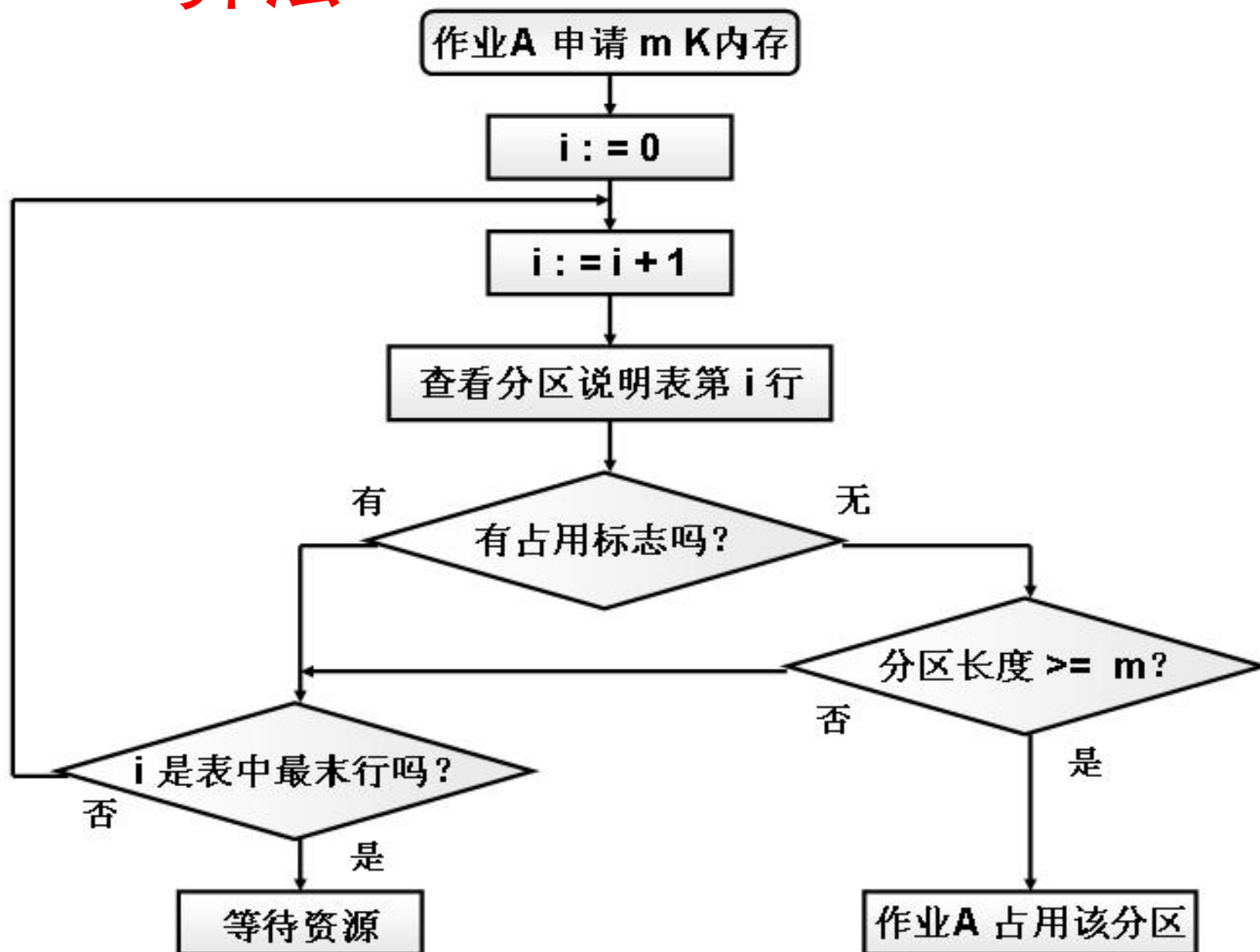
起始地址	长度	标志
1300	200	未分配
1900	650	未分配
		空
	⋮	

空闲区表

可变式分区的内存分配策略：

- § (1)最佳适应算法 (Best Fit)：为一个作业选择分区时，总是寻找其大小最接近于作业所要求的存储区域。
- § (2)最坏适应算法 (Worst Fit)：为作业选择存储区域时，总是寻找最大的空白区。
- § (3)首次适应算法 (First Fit)：每个空白区按其在存储空间中地址递增的顺序连在一起，在为作业分配存储区域时，从这个空白区域链的始端开始查找，选择第一个足以满足请求的空白块。
- § (4)下次适应算法 (Next Fit)：把存储空间中空白区构成一个循环链，每次为存储请求查找合适的分区时，总是从上次查找结束的地方开始，只要找到一个足够大的空白区，就将它划分后分配出去。

FirstFit算法



FirstFit算法

§ 优点：

- 分配和释放的时间性能较好
- 较大的空闲分区保留在内存的高端

§ 缺点：随着低端内存被不断分配，会产生很多小分区，开销会增大。

算法举例

§ 例：系统中的空闲分区表如下表示，现有三个作业分配申请内存空间 100K、30K 及 7K，给出按首次适应算法、下次适应算法、最佳适应算法和最坏适应算法的内存分配情况及分配后空闲分区表。

区号	大小	起始地址	状态
1	32K	20K	未分配
2	8K	52K	未分配
3	120K	60K	未分配
4	331K	180K	未分配

首次适应算法

§ 申请作业 100k，分配 3 号分区，剩下分区为 20k，起始地址 160K。

§ 100K、30K、7K

区号	大小	起始地址	状态
1	32K	20K	未分配
2	8K	52K	未分配
3	20K	160K	未分配
4	331K	180K	未分配

首次适应算法

§ 申请作业 30K，分配 1 号分区，剩下分区为 2K，起始地址 50K。

§ 100K、30K、7K

区号	大小	起始地址	状态
1	2K	50K	未分配
2	8K	52K	未分配
3	20K	160K	未分配
4	331K	180K	未分配

首次适应算法

§ 申请作业 7K，分配 2 号分区，剩下分区为 1K，起始地址 59K。

§ 100K、30K、7K

区号	大小	起始地址	状态
1	2K	50K	未分配
2	1K	59K	未分配
3	20K	160K	未分配
4	331K	180K	未分配

下次适应算法

§ 按下次适应算法，申请作业 100k，分配 3 号分区，剩下分区为 20k，起始地址 160K；

§ 100K、30K、7K

区号	大小	起始地址	状态
1	32K	20K	未分配
2	8K	52K	未分配
3	20K	160K	未分配
4	331K	180K	未分配

下次适应算法

§ 按下次适应算法，申请作业 30K，分配 4 号分区，
剩下分区为 301K；

§ 100K、30K、7K

区号	大小	起始地址	状态
1	32K	20K	未分配
2	8K	52K	未分配
3	20K	160K	未分配
4	301K	210K	未分配

下次适应算法

§ 按下次适应算法，申请作业 7k，分配 1 号分区

§ 100K、30K、7K

区号	大小	起始地址	状态
1	25K	27K	未分配
2	8K	52K	未分配
3	20K	160K	未分配
4	301K	210K	未分配

算法特点

- § 首次适应：优先利用内存低地址部分的空闲分区。但由于低地址部分不断被划分，留下许多难以利用的很小的空闲分区（碎片或零头），而每次查找又都是从低地址部分开始，增加了查找可用空闲分区的开销。
- § 下次适应：使存储空间的利用更加均衡，不致使小的空闲区集中在存储区的一端，但这会导致缺乏大的空闲分区。

最佳适应算法

按容量大小递增的次序排列

分配前的空闲分区表

区号	大小	起址
1	8k	52k
2	32k	20k
3	120k	60k
4	331k	180k

作业30K分配后

区号	大小	起址
2	2k	50k
1	8k	52k
3	20k	160k
4	331k	180k

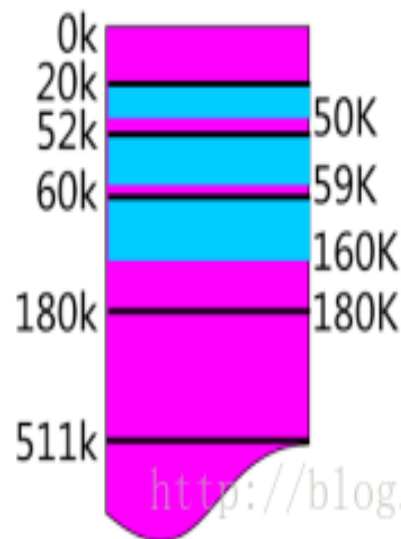
按容量递增的次序重新排列

作业100K分配后

区号	大小	起址
1	8k	52k
3	20k	160k
2	32k	20k
4	331k	180k

作业7K分配后

区号	大小	起址
1	1k	59k
2	2k	50k
3	20k	160k
4	331k	180k



区号	大小	起址
1	1k	59k
2	2k	50k
3	20k	160k
4	331k	180k

http://blog.csdn.net/qq_28602957

最坏适应算法

按容量大小递减的次序排列

分配前的空闲分区表

区号	大小	起址
1	331k	180k
2	120k	60k
3	32k	20k
4	8k	52k

作业30K分配后

区号	大小	起址
1	201k	310k
2	120k	60k
3	32k	20k
4	8k	52k

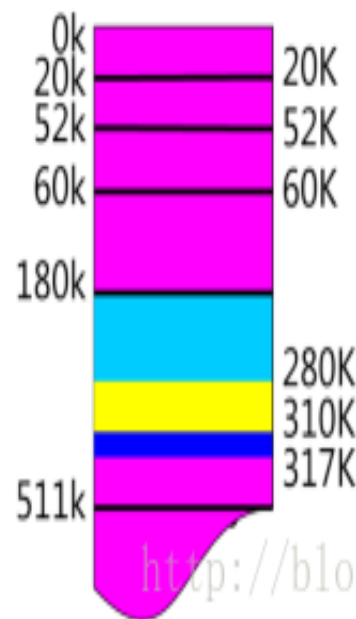
按容量递减的次序重新排列

作业100K分配后

区号	大小	起址
1	231k	280k
2	120k	60k
3	32k	20k
4	8k	52k

作业7K分配后

区号	大小	起址
1	194k	317k
2	120k	60k
3	32k	20k
4	8k	52k



区号	大小	起址
1	194k	317k
2	120k	60k
3	32k	20k
4	8k	52k

http://blog.csdn.net/qq_28602957

算法特点

- § 最佳适应：若存在与作业大小一致的空闲分区,则它必然被选中，若不存在与作业大小一致的空闲分区，则只划分比作业稍大的空闲分区，从而保留了大的空闲分区。最佳适应算法往往使剩下的空闲区非常小，从而在存储器中留下许多难以利用的小空闲区（碎片）。
- § 最坏适应算法的特点：总是挑选满足作业要求的最大的分区分配给作业。这样使分给作业后剩下的空闲分区也较大，可装下其它作业。由于最大的空闲分区总是因首先分配而划分，当有大作业到来时，其存储空间的申请往往会得不到满足。

练习题

§ 在下列存储管理算法中，内存分配和释放平均时间之和为最大的是：

A 首次适应 B 下次适应 C 最佳适应 D 最差适应

练习题

§ 在下列存储管理算法中，内存分配和释放平均时间之和为最大的是：

A 首次适应 B 下次适应 C 最佳适应 D 最差适应

需要寻找满足需要且最小的空闲块，
释放要找到上下邻空闲区，修改插入链表

练习题

§ 可变分区又称为动态分区，它是在系统运行过程中——动态建立的：

- | | |
|---------|---------|
| A 作业未装入 | B 在作业装入 |
| C 在作业创建 | D 在作业完成 |

练习题

§ 可变分区又称为动态分区，它是在系统运行过程中——动态建立的：

- A 作业未装入
- B 在作业装入
- C 在作业创建
- D 在作业完成

分区大小在程序装入时大小动态确定，量身定制

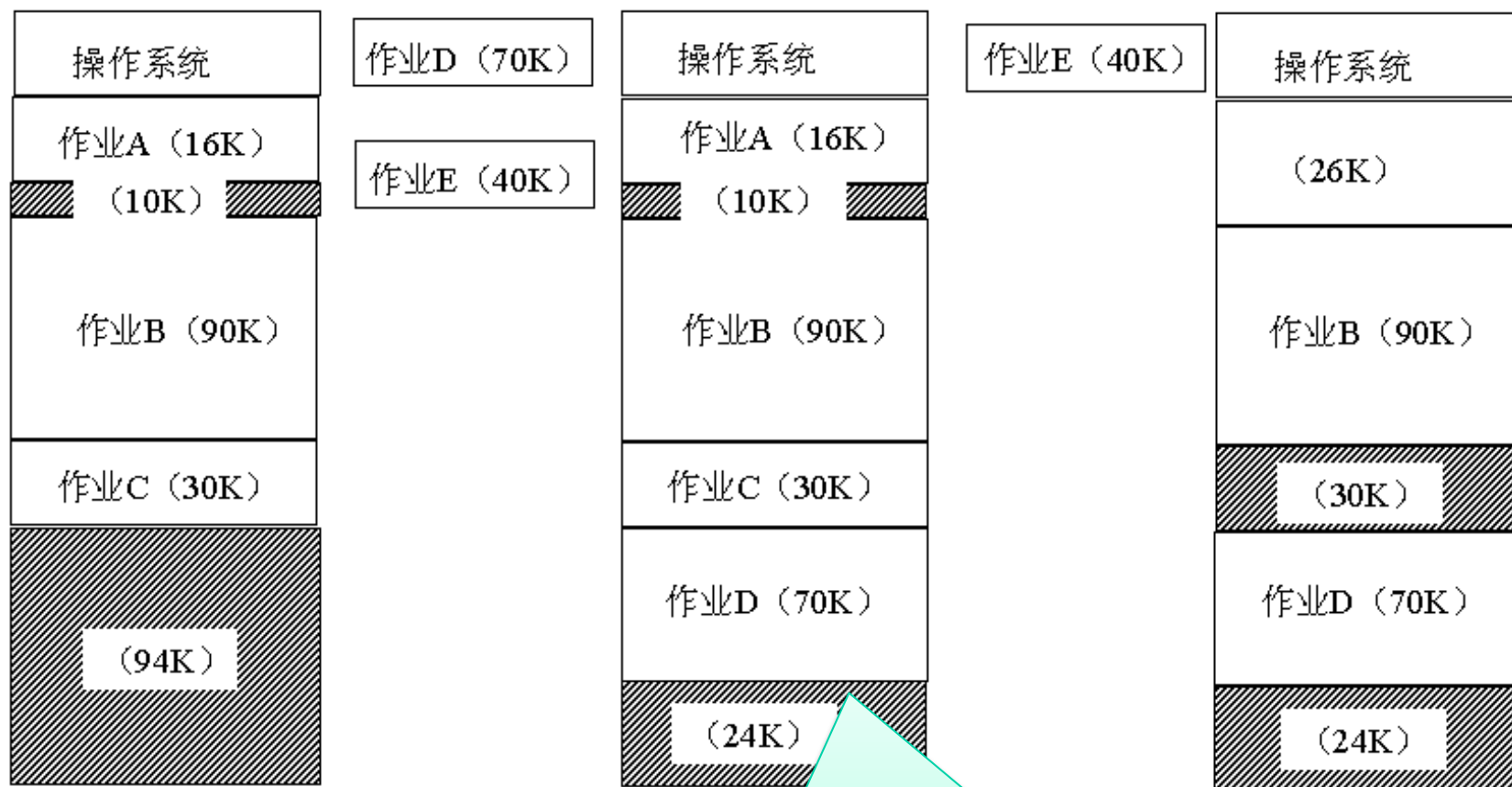
练习：可变分区的内存分配

- § 假设一个可变分区系统中内存按照顺序包含如下空闲区：10 KB, 4 KB, 20 KB, 18 KB, 7 KB, 9 KB, 12 KB, and 15 KB.
- § 假设后续连续内存分配请求是：(a) 12 KB (b) 10 KB (c) 9 KB
- § 对于首次适应那个空闲区会被分配？对于最佳适应，最差适应，和下次适应 又会分配那些空闲区？

练习：可变分区的内存分配

- § 首次适应 20 KB, 10 KB, 18 KB.
- § 最佳适应 12 KB, 10 KB, 9 KB.
- § 最差适应 20 KB, 18 KB, 15 KB.
- § 下次适应 20 KB, 18 KB, 9 KB.

可变分区分配和回收的例子

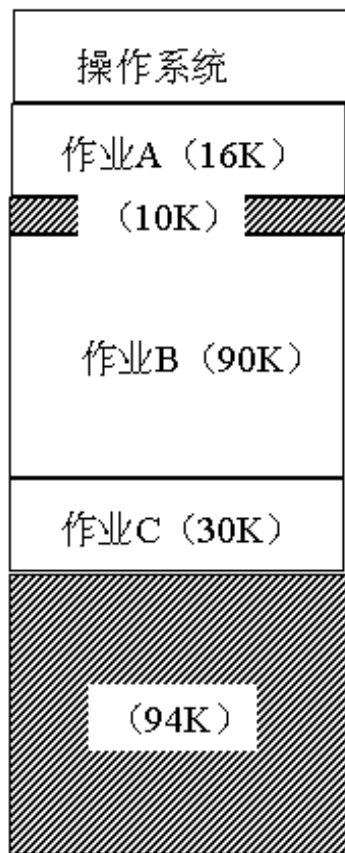


(a) 某时刻状态

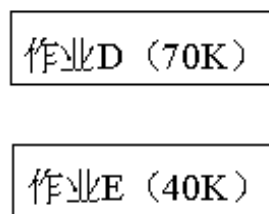
作业D分配，产生24K空闲区

可变分区分配

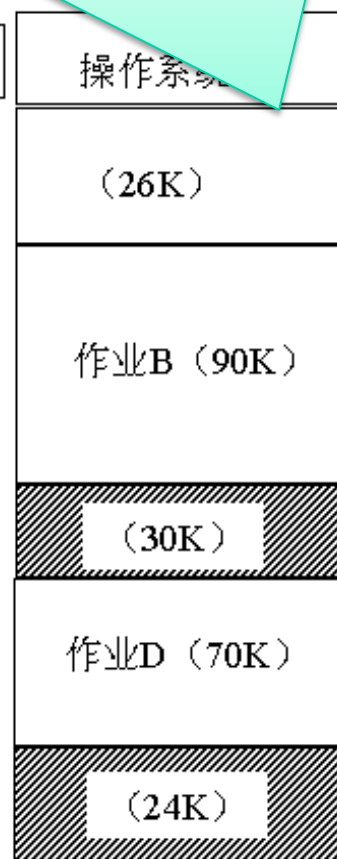
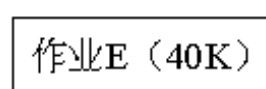
作业A撤销，产生16K空闲区，合并成26K



(a) 某时刻状态

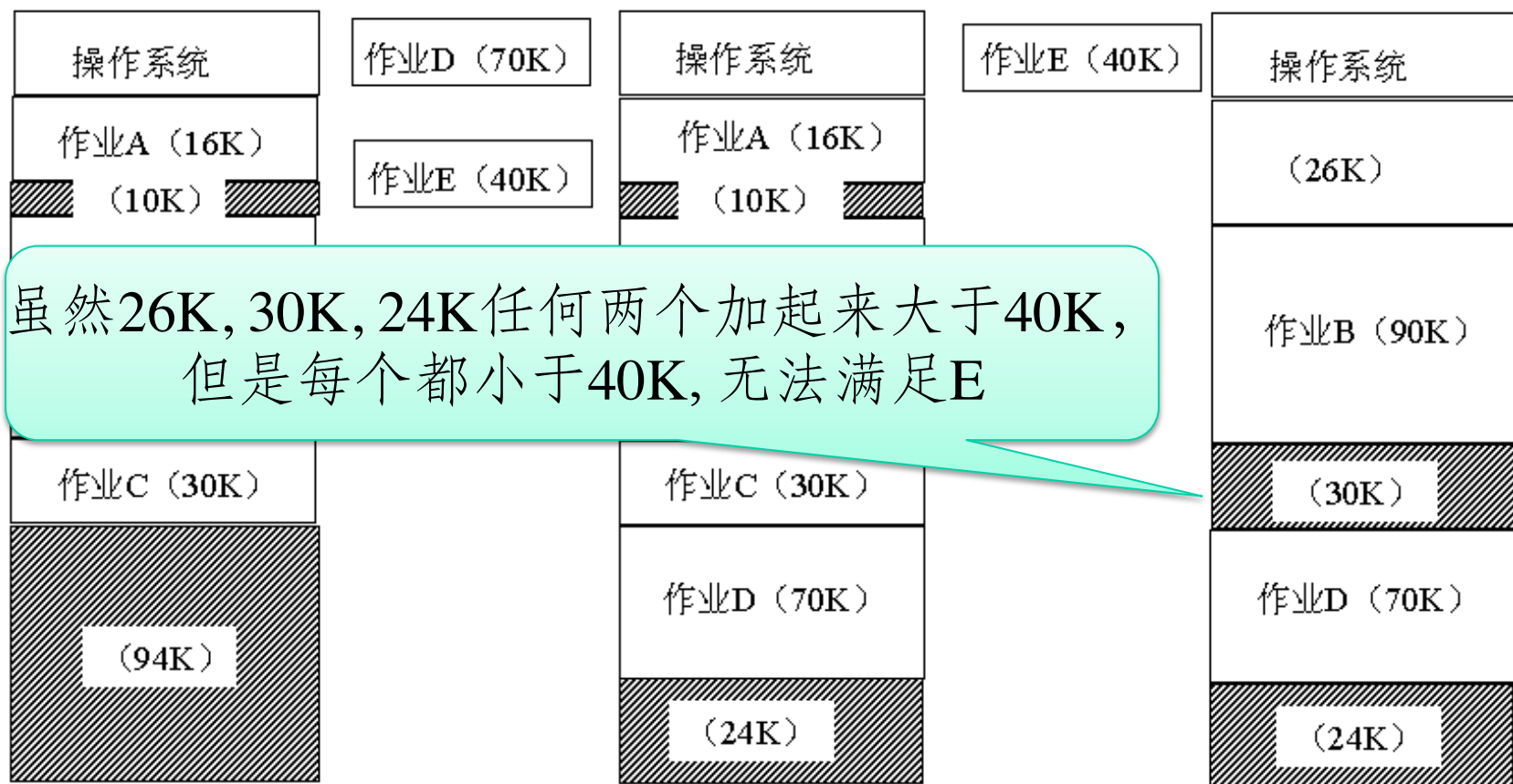


(b) 加入作业D



(c) 撤消作业A、C

可变分区分配和回收的例子

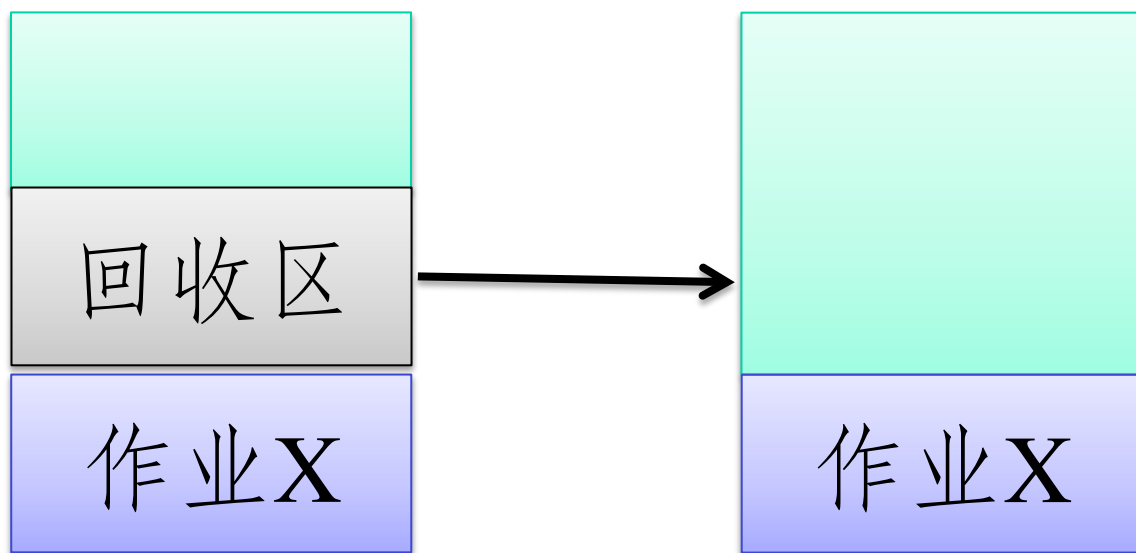


(a) 某时刻状态

(b) 加入作业D

(c) 撤消作业A、C

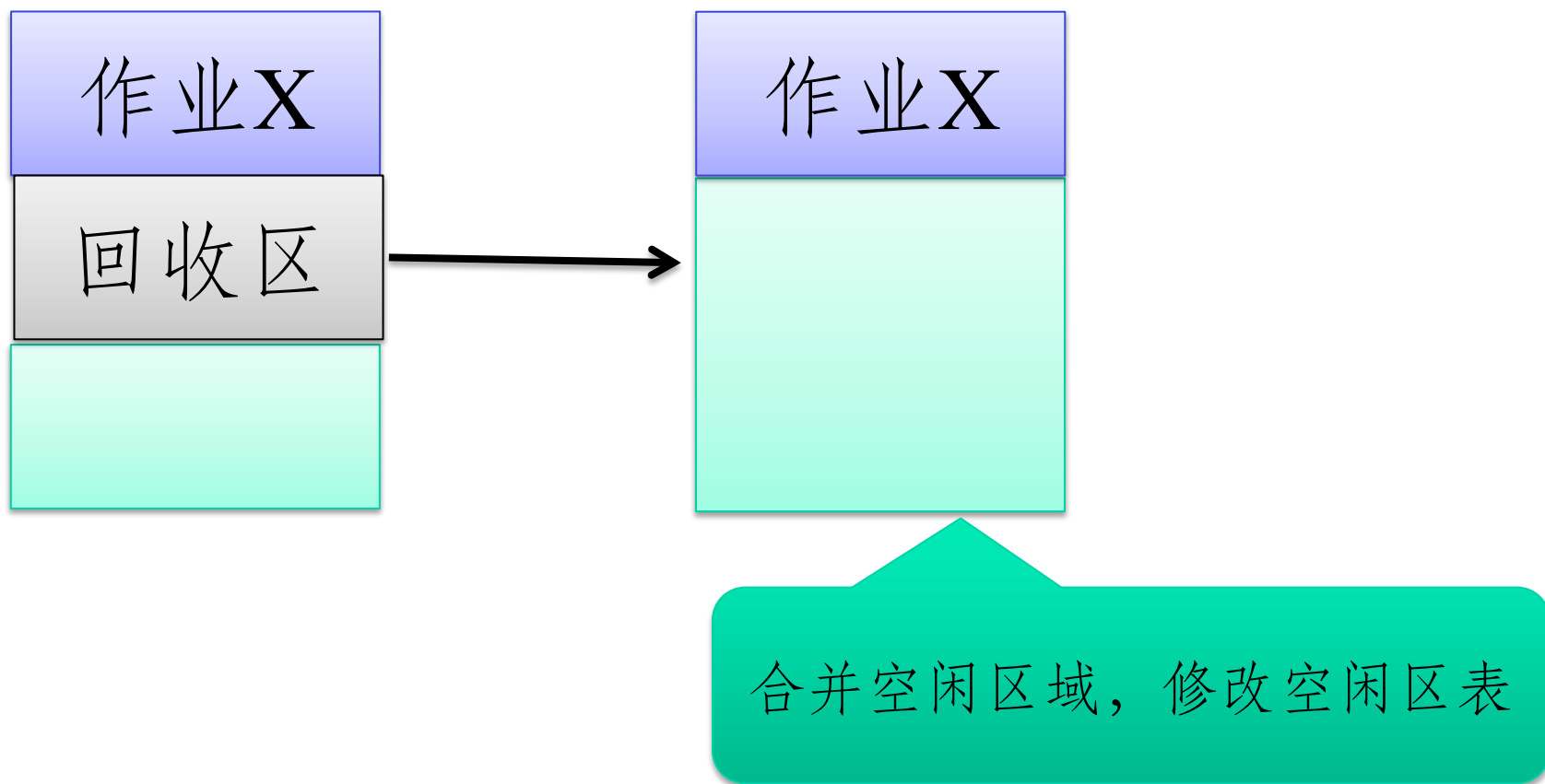
回收区域空白区邻接的四种情况



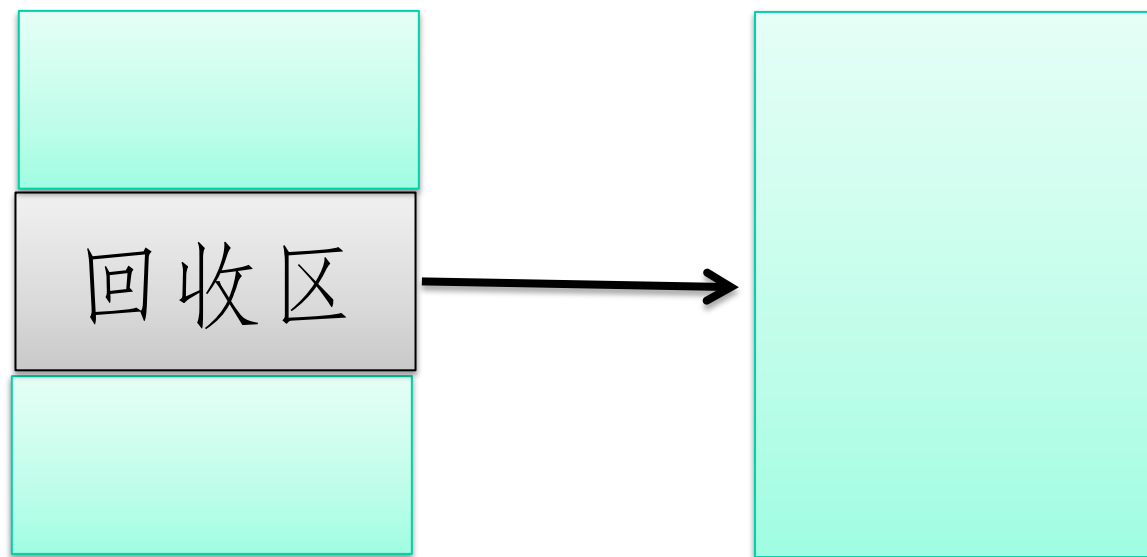
绿色表示空闲区

合并空闲区域，修改空闲区表

回收区域空白区邻接的四种情况

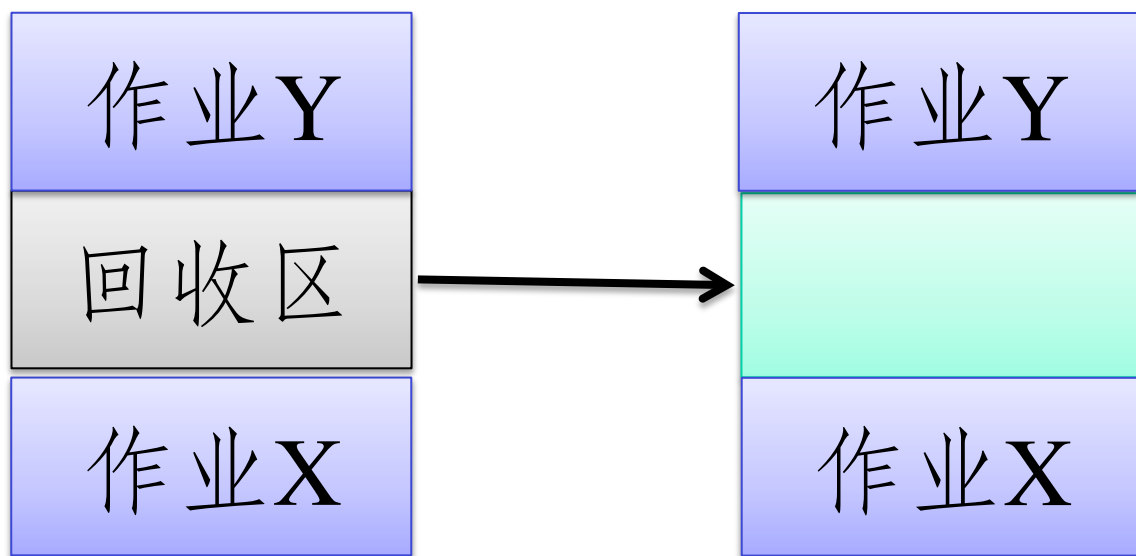


回收区域空白区邻接的四种情况



合并空闲区域，修改空闲区表

回收区域空白区邻接的四种情况



绿色表示空闲区

直接修改空闲区表

练习题

§ 在可变分区存储管理中，当释放和回收一个空闲区时，造成空闲表项区数减一的情况是：

- A 无上邻空闲区，也无下邻空闲区
- 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 有上邻空闲区，也有下邻空闲区

练习题

§ 在可变分区存储管理中，当释放和回收一个空闲区时，造成空闲表项区数减一的情况是：

A 无上邻空闲区，也无下邻空闲区

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 有上邻空闲区，也有下邻空闲区

请求回收分区R

Size ← 分区R的大小
Loc ← 分区的起始地址

已分配区说明表中
置R的状态 = 未填表项

否 分区R与F₂邻接?

是

Size ← Size + F₂的大
小

是

分区R与F₁邻接?

否

否

分区R与F₁邻接?

是

在空闲分区表中
找一个未填表项

置新空闲分区
的大小=Size
始址=Loc
状态=空闲

在空闲分区表中置F₂为未填表项

置空闲分区F₁的大小
=Size + F₁的大小

置空闲分区F₂
的大小=Size
始址=Loc

返回

已分配分区说明			
序号P	大小	起址	状态
1	8K	20K	已分配
2	32K	28K	已分配
3	—	—	未填表项
4	120K	92K	已分配
5	—	—	未填表项
...

空闲分区说明表			
序号F	大小	起址	状态
1	32K	60K	空闲
2	300K	212K	空闲
3	—	—	未填表项
4	—	—	未填表项
5	—	—	未填表项
...

空闲分区回收算法

练习题

§ 一下分配方案中，____不适合多道系统：

- A 单一连续区管理 B 固定分区管理
C 可变分区管理 D 页式存储管理

练习题

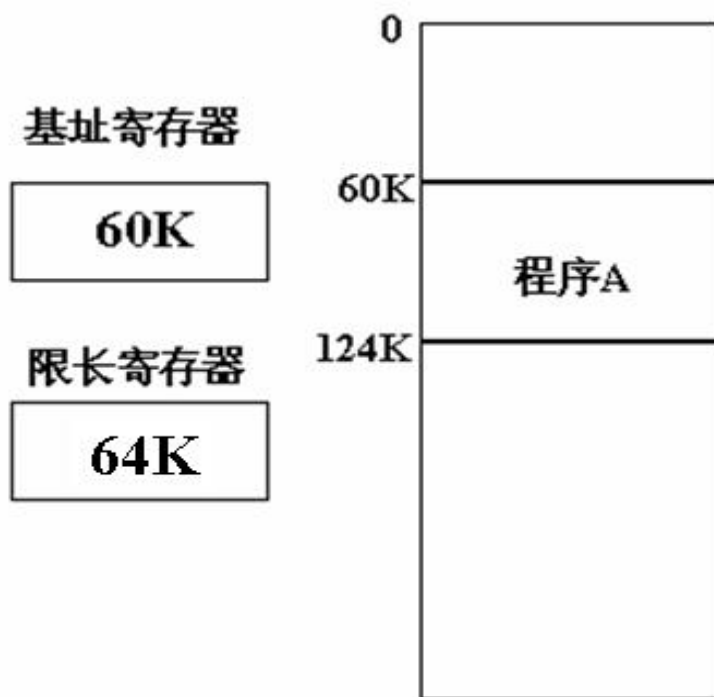
§ 一下分配方案中，____不适合多道系统：

- A 单一连续区管理 B 固定分区管理
C 可变分区管理 D 页式存储管理

可变分区的保护-界限寄存器保护

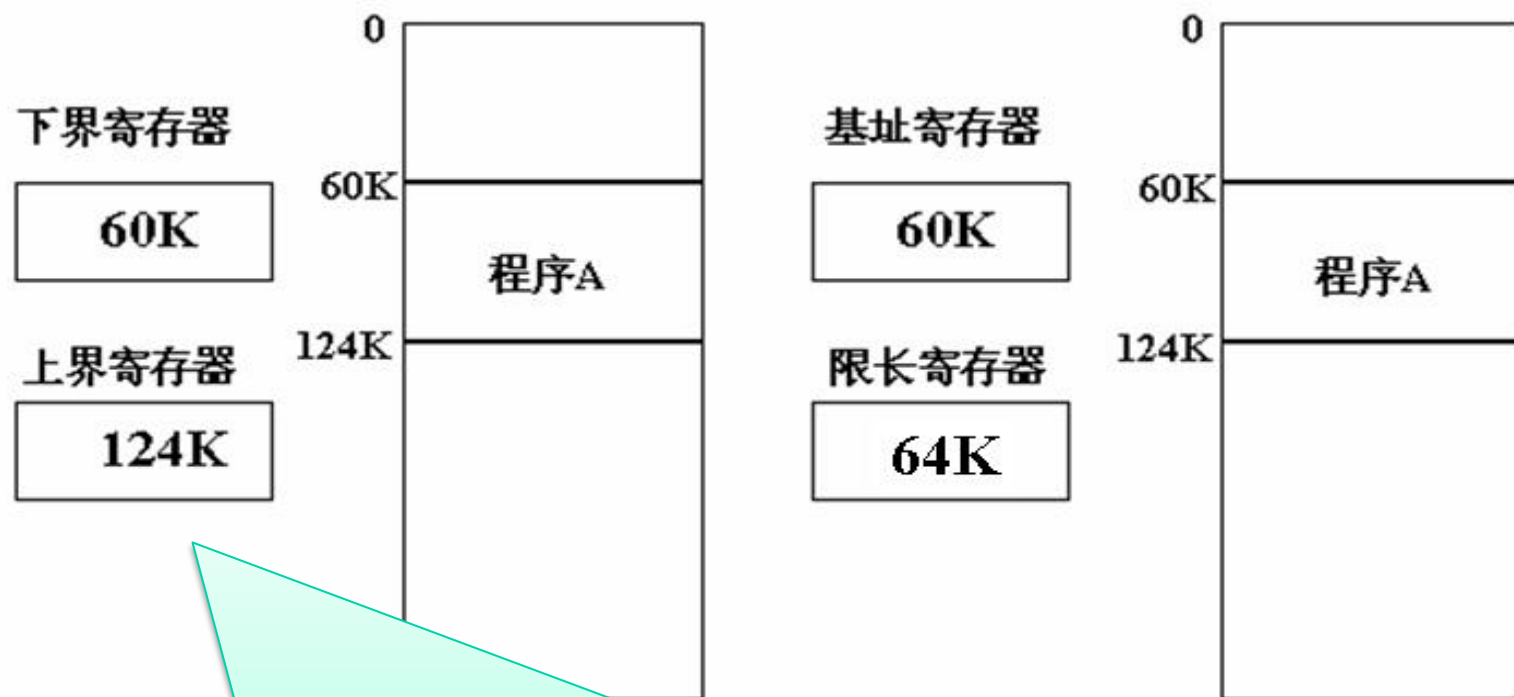


(a)



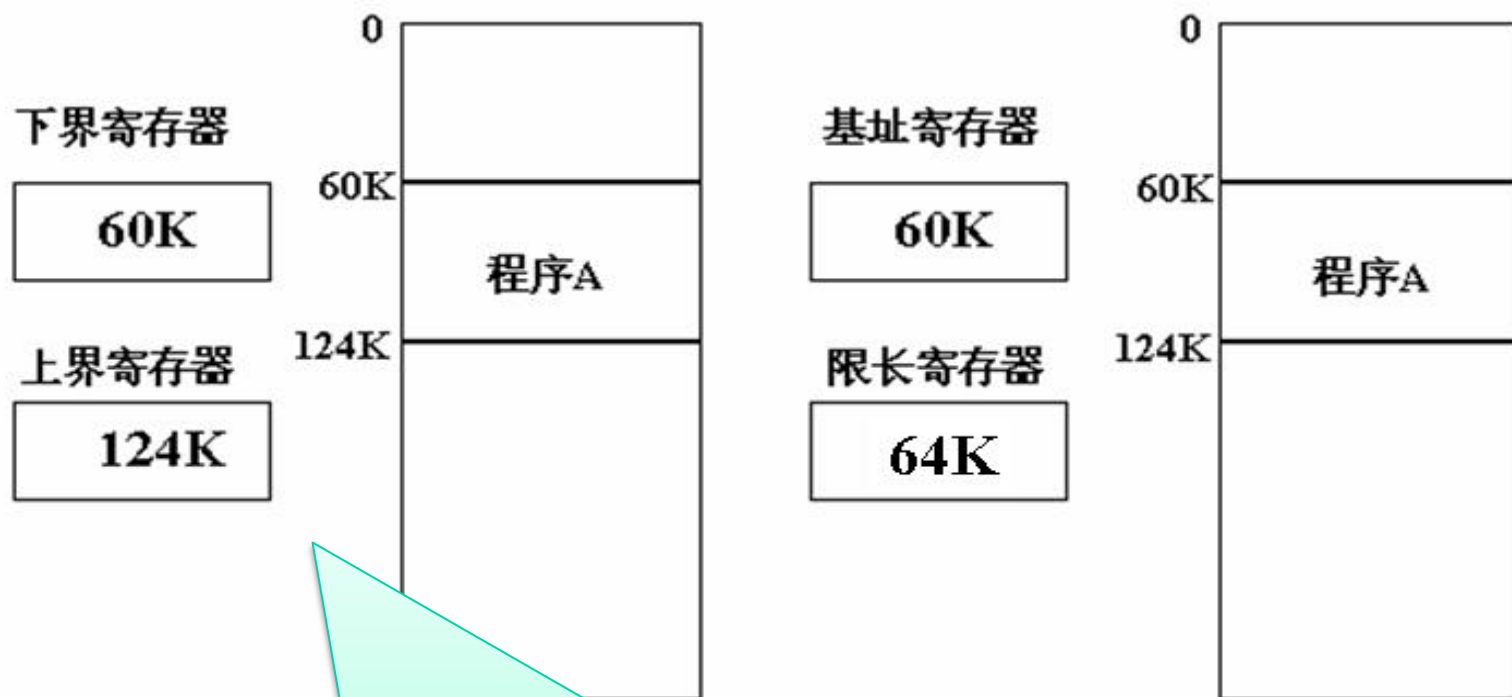
(b)

可变分区的保护-界限寄存器保护



每次访问内存的地址，硬件都自动将其与界限寄存器比较，
若发生地址越界，便产生保护性地址越界中断

可变分区的保护-界限寄存器保护



程序从60K开始存放，到124K为止

可变分区的保护-只访问自己的区域

§ 每个分区分一个保护键-锁

§ 每个进程分一个保护键-钥匙

- 存在程序状态字中

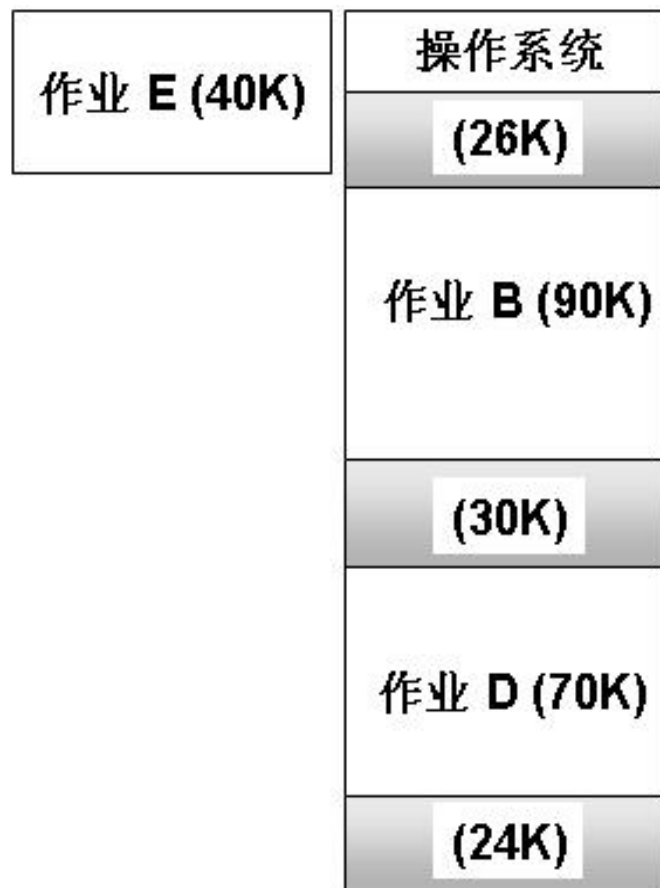
§ 访问内存的时候，检查锁和钥匙是否匹配

- 不匹配---触发中断

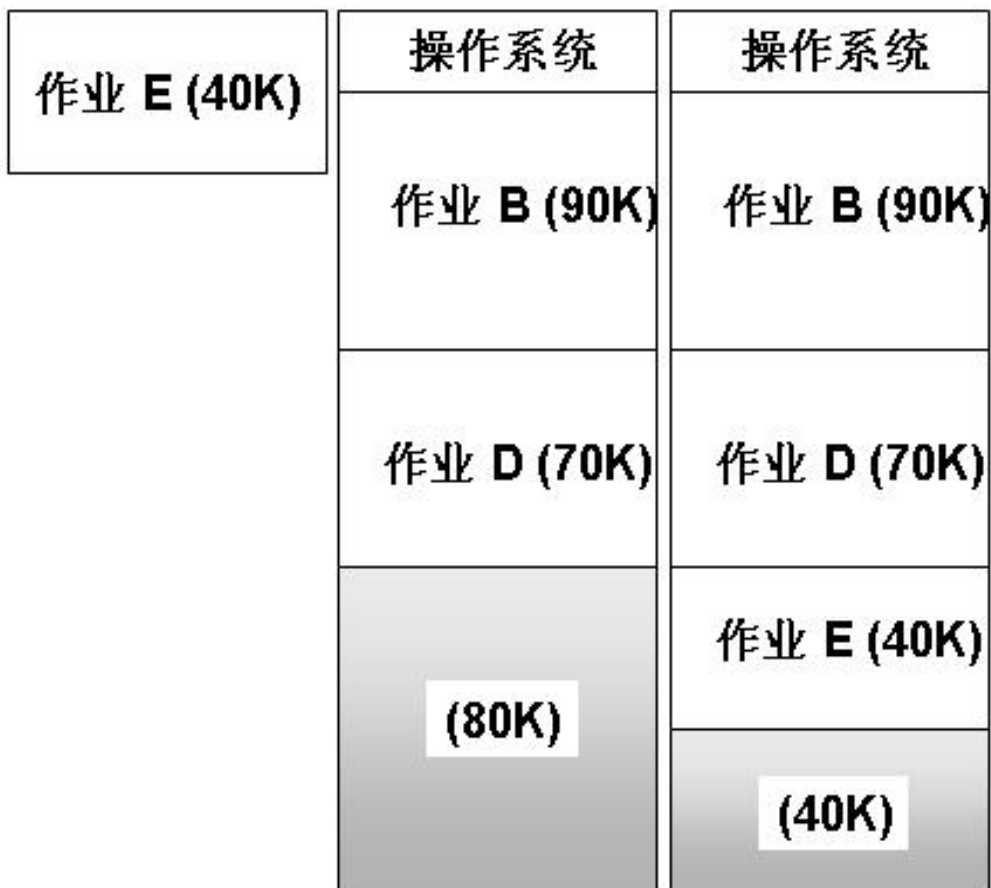
如何处理外碎片？-紧缩 (compaction)

§ 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。

如何处理外碎片？-紧缩



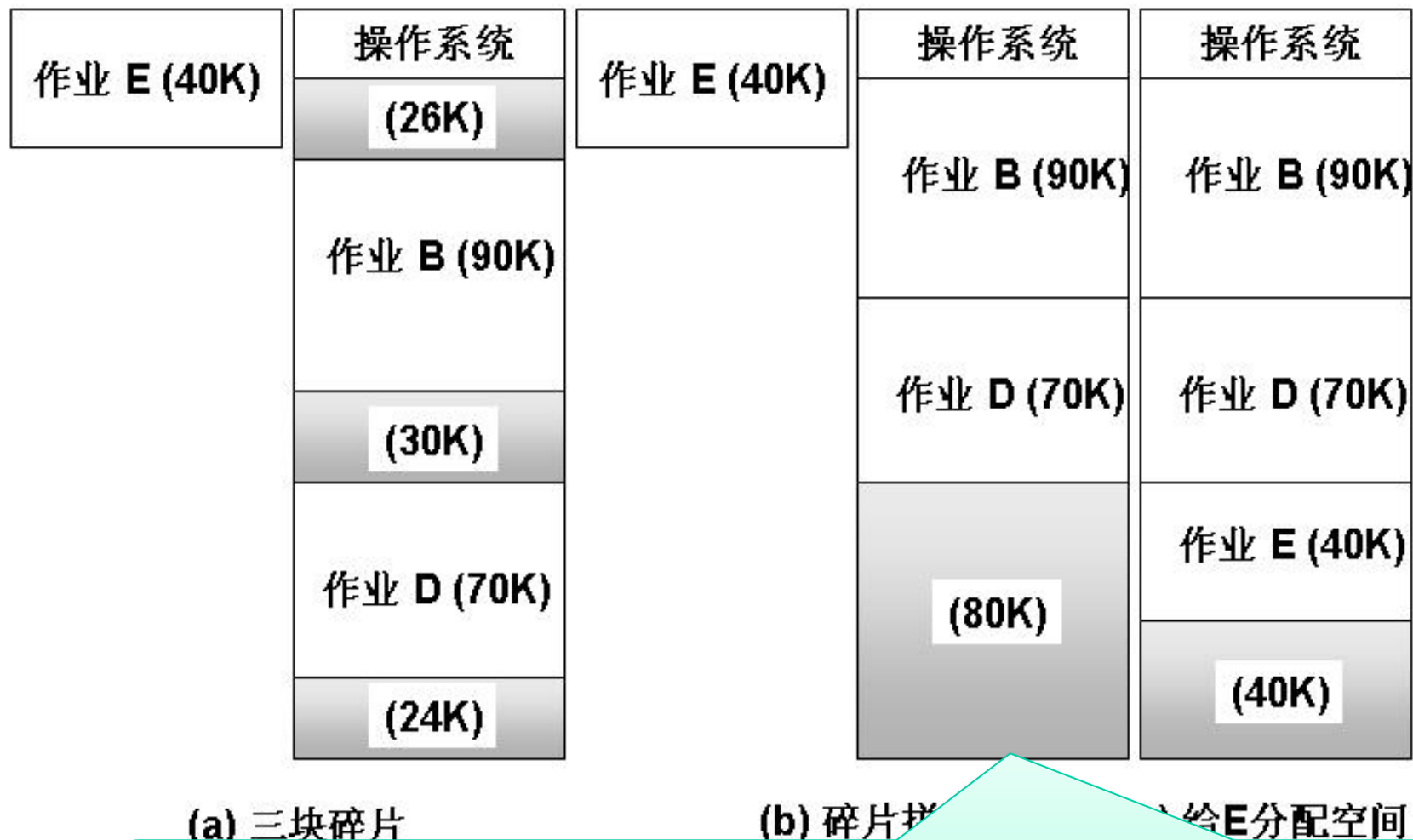
(a) 三块碎片
作业E无法分配空间



(b) 碎片拼接

(c) 给E分配空间

如何处理外碎片？-紧缩



在内存中的作业B和作业D被移动到内存的一端，三块碎片被拼接为一个大的空闲区，其容量为80K

处理外碎片-紧缩技术的问题

- § 进行数据搬移，占用CPU时间
- § 进行程序搬移，需要重定位和硬件支持
- § 效率比较低，不适合频繁进行

内存的管理-扩充容量

§ 扩充容量的思路

- 把一部分程序和数据放在外存上，把当前执行要使用的程序和数据放在内存

§ 覆盖技术

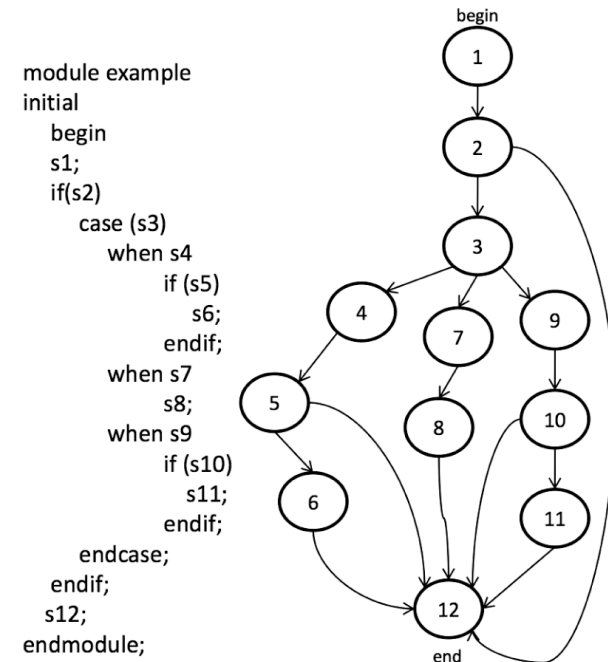
- 让程序和数据对内存空间时分复用

§ 交换技术

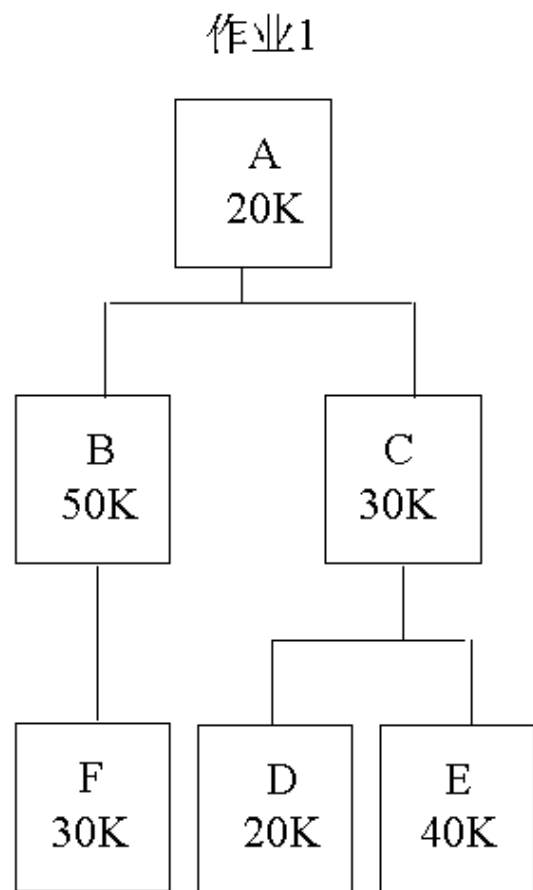
覆盖

§ 覆盖：一个程序的代码段和数据段按照时间先后占用内存空间

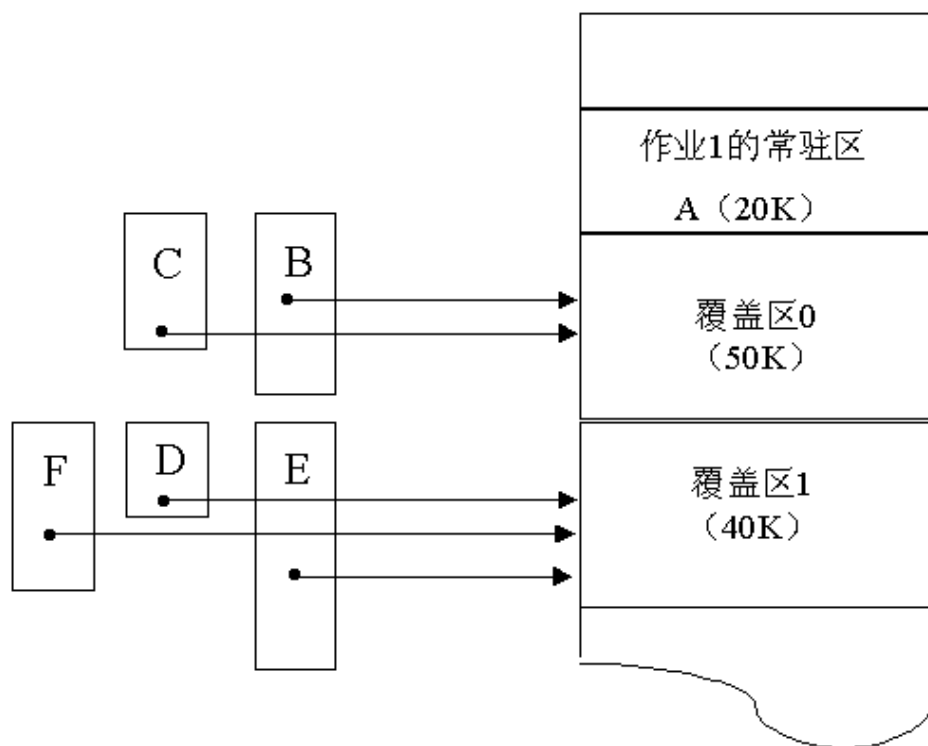
- 必要的部分占据内存
- 可选的部分放外存
- 不存在调用关系的模块不必同时在内存，可以互相覆盖
- 不同的执行路径不需要同时在内存



覆盖

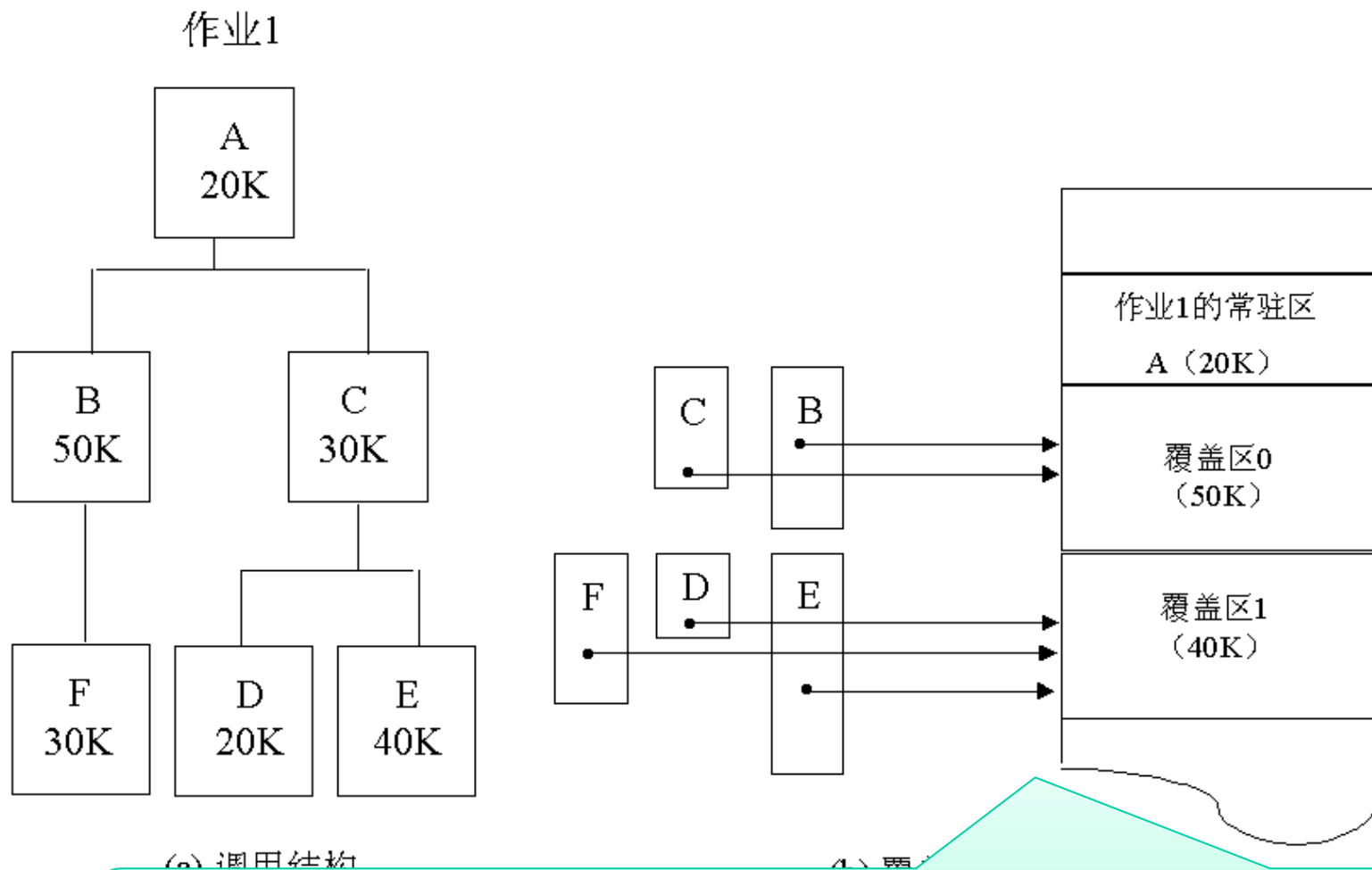


(a) 调用结构



(b) 覆盖结构及内存分配

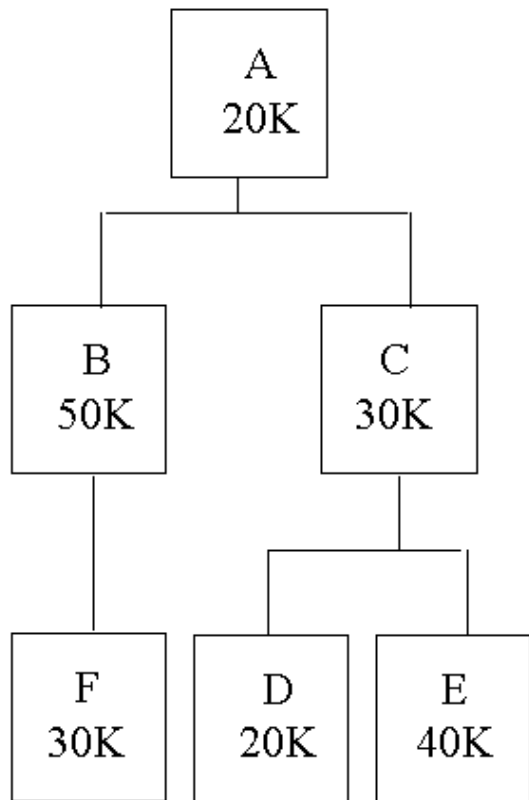
覆盖



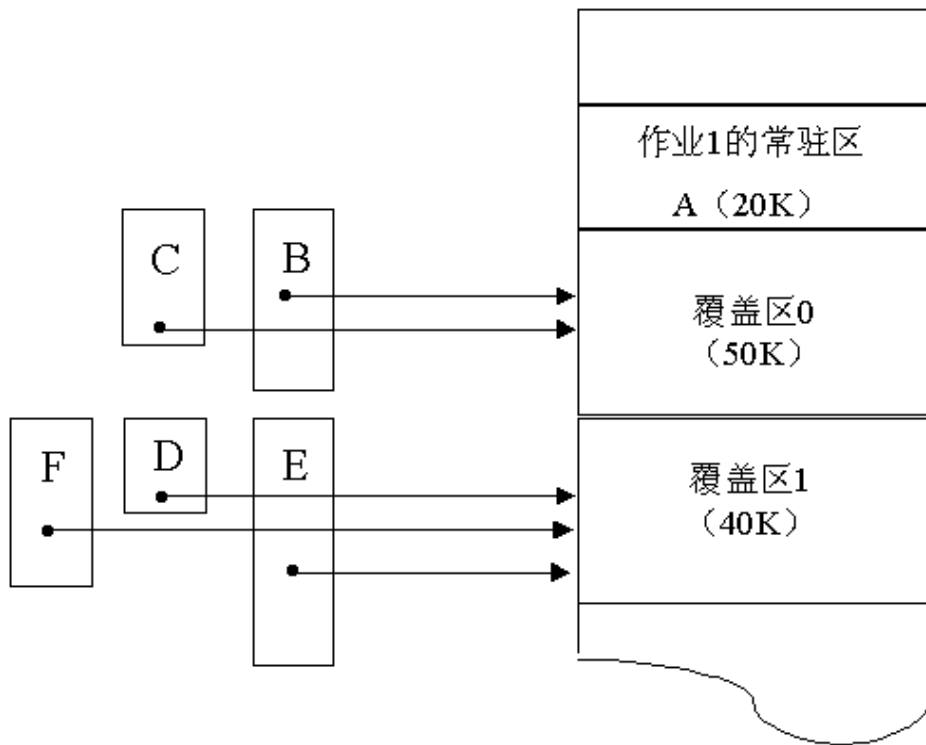
B和C不会互相调用，因此B和C就无需同时在内存中
同理，F、D、E无需同时在内存

覆盖

作业1



(a) 调用结构



(b) 覆盖结构及内存分配

覆盖前， $ABCDEF = 190K$

覆盖后：110K，即容量需求最大的路径

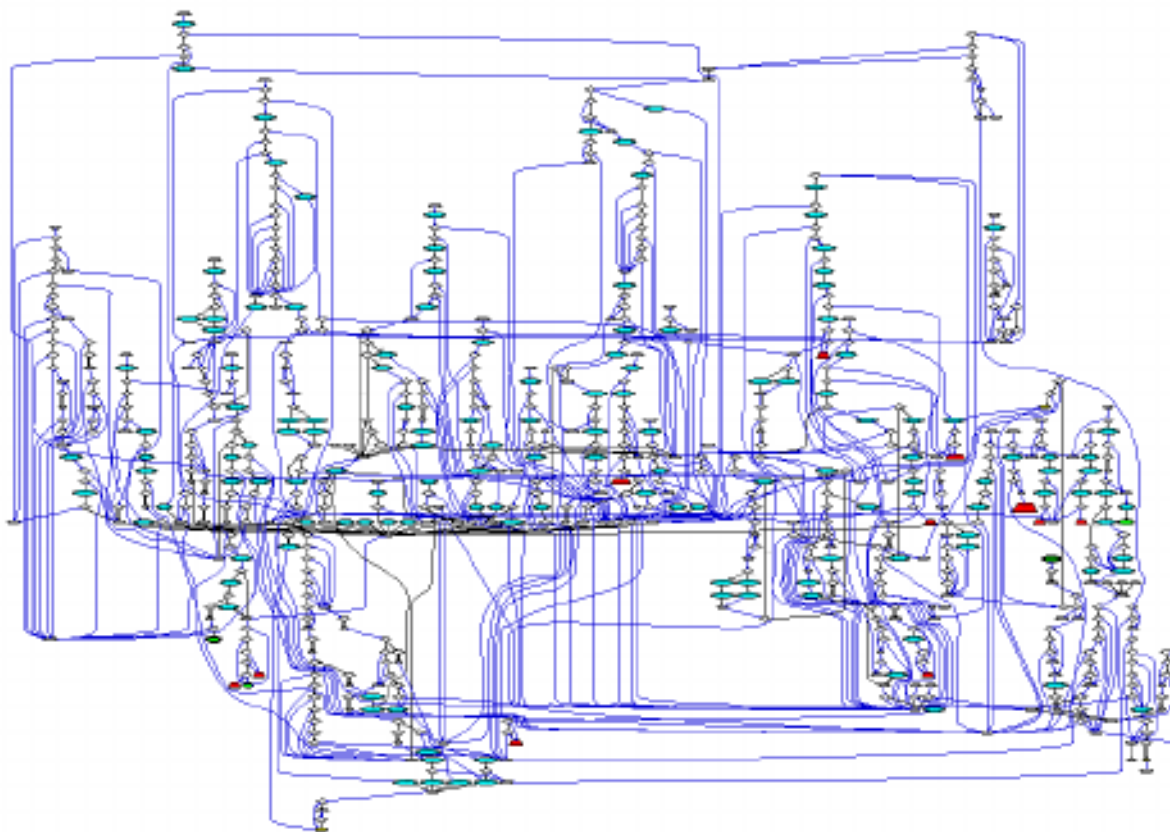
覆盖

§ 缺点

- 编程时必须划分程序模块和确定程序模块之间的覆盖关系，了解调用次序，增加编程复杂度。
- 从外存装入覆盖文件，以时间延长来换取空间节省。
- 程序的最大长度仍然受限内存

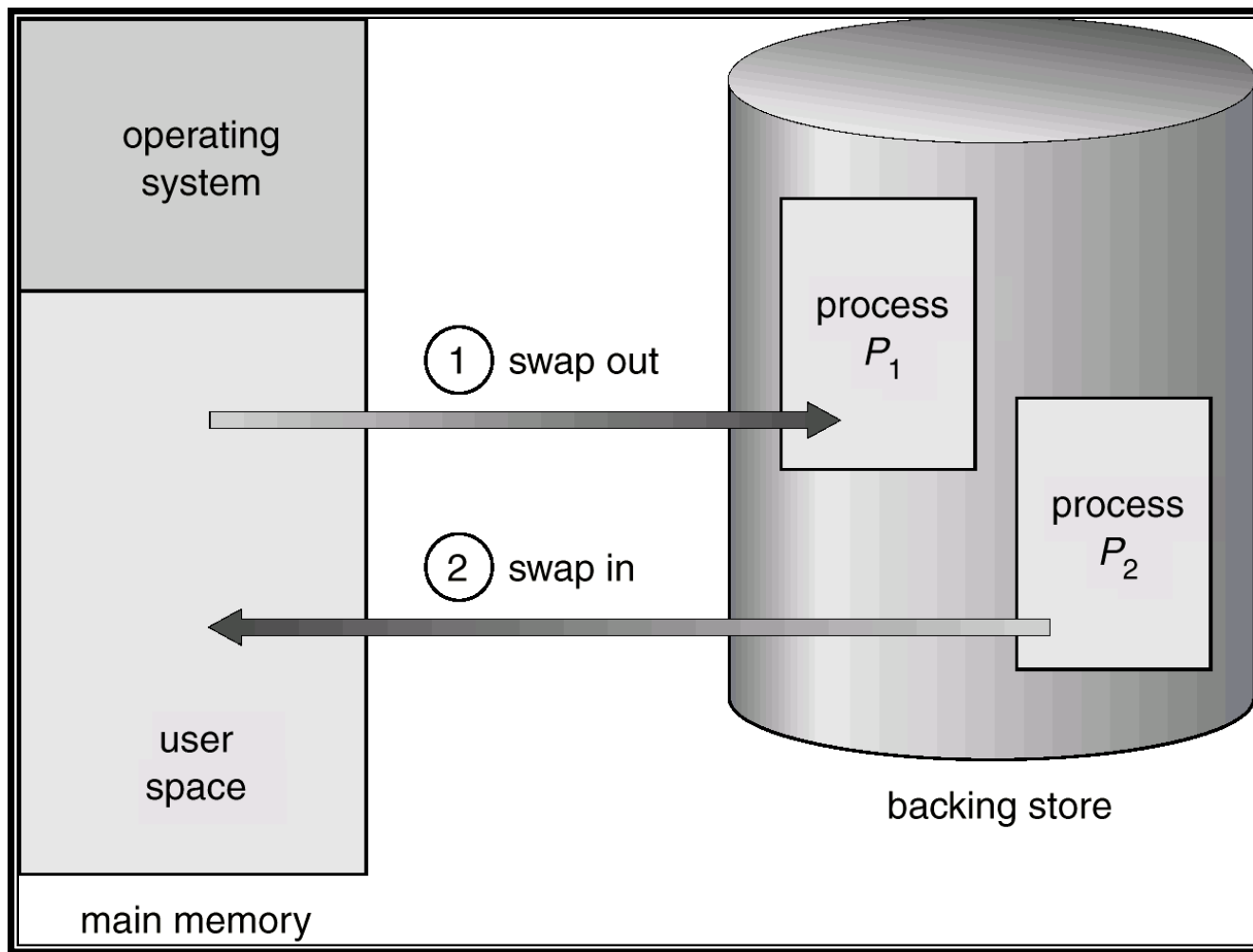
覆盖

§ 一个普通Android应用的控制流图



交换

- § 交换：广义的说，所谓交换就是把暂时不用
的某个（或某些）程序及其数据的部分或全
部从主存移到辅存中去，以便腾出必要的存
储空间；接着把指定程序或数据从辅存读到
相应的主存中，并将控制转给它，让其在系
统上运行。
- § 优点：增加并发运行的程序数目，并且给用
户提供适当的响应时间；编写程序时不影响
程序结构
- § 缺点：对换入和换出的控制增加处理机开销
；程序整个地址空间都进行传送，没有考虑
执行过程中地址访问的统计特性。



交换技术的几个问题

§ 选择原则，即将哪个进程换出/内存？

- 系统在选择换出程序时，希望换出的程序是短时间内不会立刻投入运行。
- 等待I/O的进程

§ 交换时机的确定，何时需发生交换？

- 只要不用就换出（很少再用）；
- 只在内存空间不够或有不够的危险时换出

交换技术的几个问题

§ 交换空间的分配和管理？

- 当程序被换出时，必须为它分配磁盘空间
- 固定空间v.s.可变空间

§ 换回内存时位置的确定

- 如果包含物理地址引用：原位置
- 如果是相对地址：可以换位置

覆盖与交换技术的区别

- § 覆盖可减少一个程序运行所需的空間。
交换可让整个程序暂存于外存中，让出内存空间。
- § 覆盖是由程序员实现的，操作系统根据程序员提供的覆盖结构来完成程序段之间的覆盖。交换技术对程序员透明。
- § 覆盖技术主要对同一个作业或程序进行。交换换主要在作业或程序间之间进行。