Computer

操作系统原理与实践

第三章 内存管理（1）

高等教育出版社

内存管理

- 目的与要求：掌握程序处理基本过程中内存管理相关环节的概念及内存管理的各种方法与技术。
- 重点与难点：重定位的基本概念，动态分区分配方式、分段和分页存储管理方式、虚拟存储器等主要内存管理方式的相关概念及关键技术。
- 作业：**2，3，5，6，7，8，10，11，15，17，18**

第三章　存储器管理

3.1 存储器的层次结构

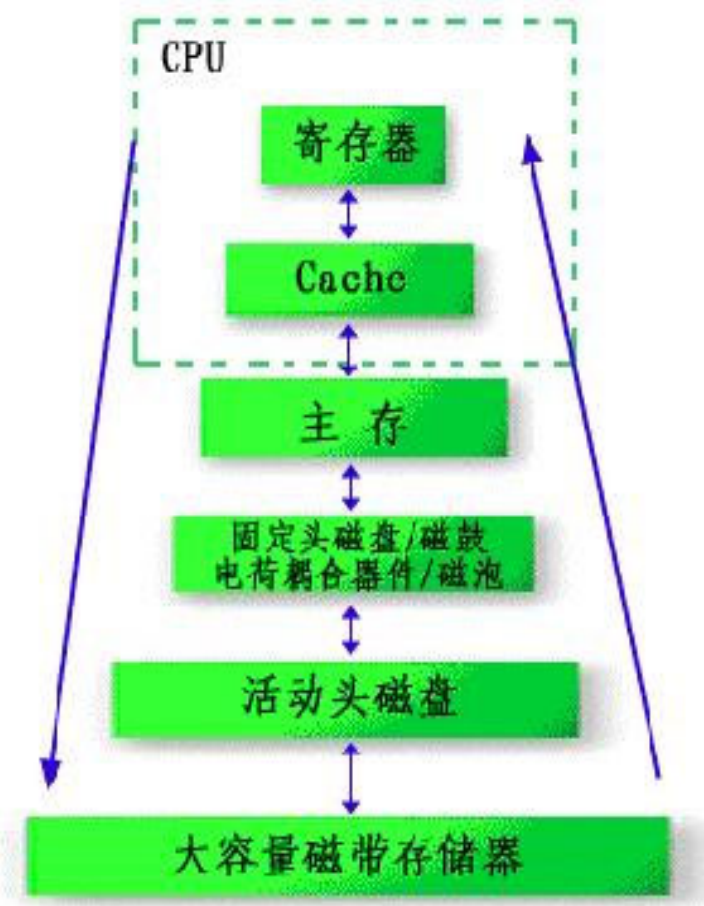
3.2　程序的装入和链接

3.3　连续分配方式

3.1.1 存储器的层次结构

1. **存储器的层次结构**

在现代计算机系统中，存储器是信息外理的来源与归宿，占据重要位置。但是，在现有技术条件下，任何一种存储装置，都无法同时从速度与容量两方面，满足用户的需求。实际上它们组成了一个速度由快到慢，容量由小到大的存储装置层次。



存储装置层次

2. 各种存储器

- 高速缓存Cache：
  - 少量的、非常快速、昂贵、易变的
- 内存RAM：
  - 若干兆字节、中等速度、中等价格、易变的
- 磁盘：
  - 数百兆或数千兆字节、低速、价廉、不易变的
- 由操作系统协调这些存储器的使用

3.1.2 存储管理的目的

- 内存分配
  - 使各得其所、提高利用率及适应动态增长要求
  - 连续分配/离散分配方式
- 地址映射
  - 逻辑地址转换为物理地址，与分配方式相关
- 内存保护
  - 基于地址的保护、存取访问控制保护
- 内存扩充
  - 对换技术、虚拟存储技术

3.1.3. 基本概念

1. 定位（存储分配）：为具体的程序和数据等分配存储单元或存储区工作。
2. 映射：把逻辑地址转换为相应的物理地址的过程。
3. 隔离：按存取权限把合法区与非法区分隔，实现存储保护。

4. 名空间

- 程序员在程序中定义的标识符
- 程序符号集合
- 由程序员自定义
- 没有地址的概念

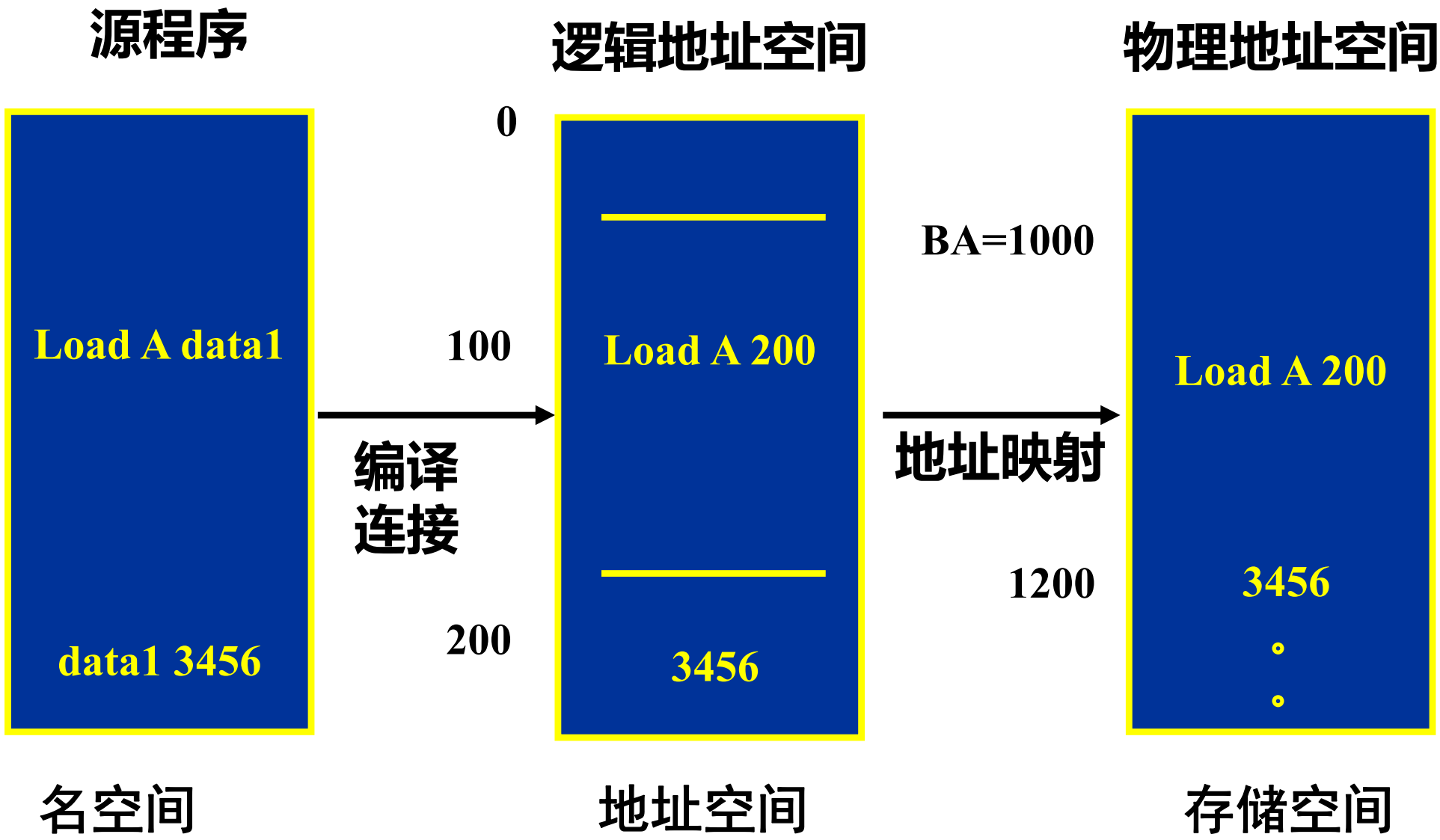
符号指令
数据说明
I/O说明

5. 地址空间

- 程序用来访问信息所用地址单元的集合
- 逻辑（相对）地址的集合
- 由编译程序生成

6. 存储空间

- 主存中物理单元的集合
- 物理（绝对）地址的集合
- 由装配程序等生成


源程序
逻辑地址空间
物理地址空间
0
100
200
Load A data1
data1 3456
编译
连接
Load A 200
3456
BA=1000
1200
Load A 200
3456
地址映射
名空间
地址空间
存储空间

7. 逻辑地址与物理地址

- 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，其首地址为0，其余指令中的地址都相对于首地址而编址。

  不能用逻辑地址在内存中读取信息

- 物理地址（绝对地址，实地址）

  内存中存储单元的地址，可直接寻址

8. 存储共享

- 内存共享：两个或多个进程共用内存中相同区域
- 目的：节省内存空间，提高内存利用率
- 实现进程通信（数据共享）
- 共享内容：

  代码共享，要求代码为纯代码

  数据共享

9. 存储保护与安全

保护目的：

为多个程序共享内存提供保障，使在内存中的各道程序，只能访问它自己的区域，避免各道程序间相互干扰，特别是当一道程序发生错误时，不致于影响其他程序的运行。通常由硬件完成保护功能，由软件辅助实现。（特权指令不能完成存储保护。）

1） 存储保护

- 保护系统程序区不被用户侵犯（有意或无意的）
- 不允许用户程序读写不属于自己地址空间的数据（系统区地址空间，其他用户程序的地址空间）

2）保护过程---防止地址越界

每个进程都有自己独立的进程空间，如果一个进程在运行时所产生的地址在其地址空间之外，则发生地址越界。即当程序要访问某个内存单元时，由硬件检查是否允许，如果允许则执行，否则产生地址越界中断，由操作系统进行相应处理。

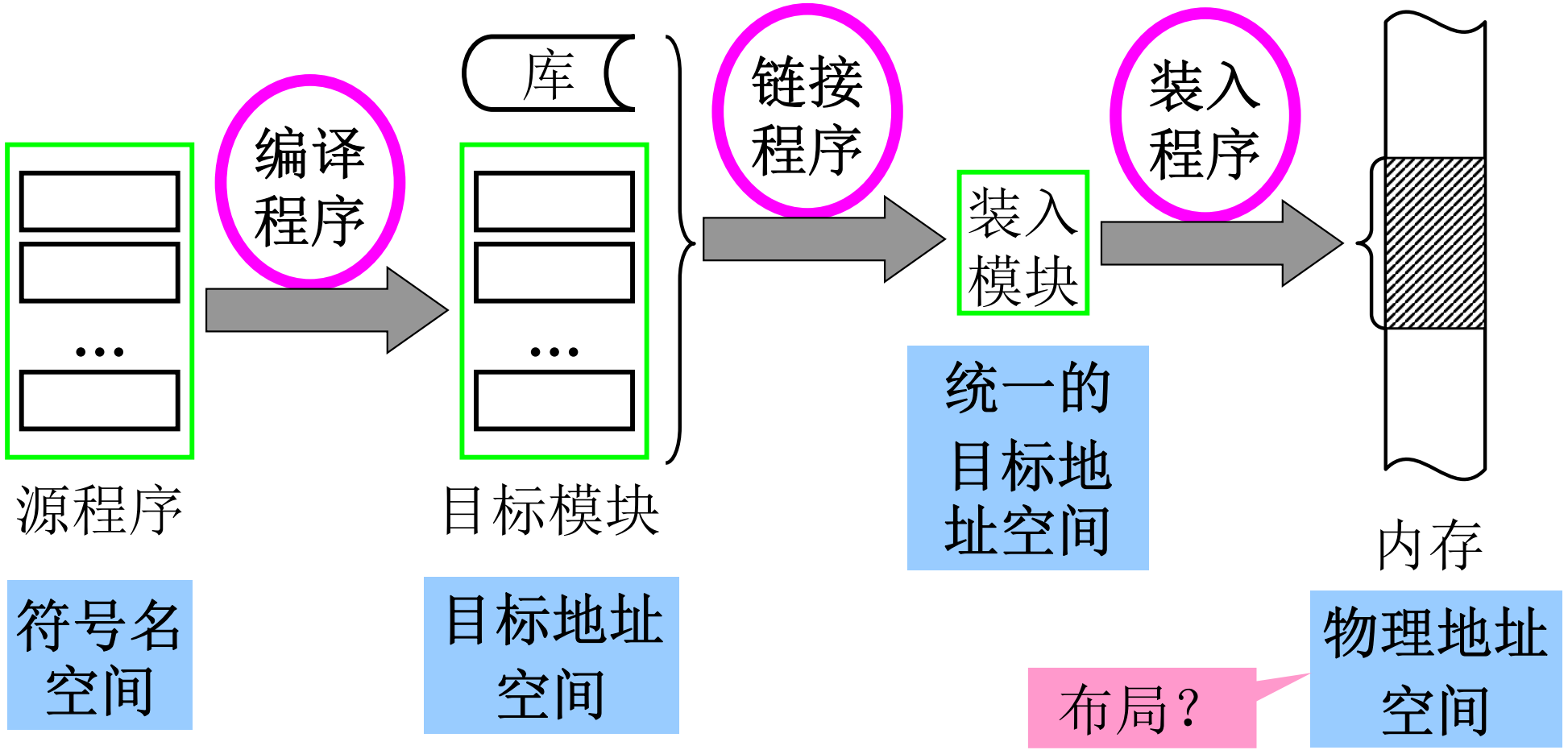
10. 内存“扩充”

通过虚拟存储技术实现

- 用户在编制程序时，不应该受内存容量限制，所以要采用一定技术来“扩充”内存的容量，使用户得到比实际内存容量大的多的内存空间
- 具体实现是在硬件支持下，软硬件相互协作，将内存和外存结合起来统一使用。通过这种方法把内存扩充，使用户在编制程序时不受内存限制

3.2 程序的装入和链接

程序处理基本过程

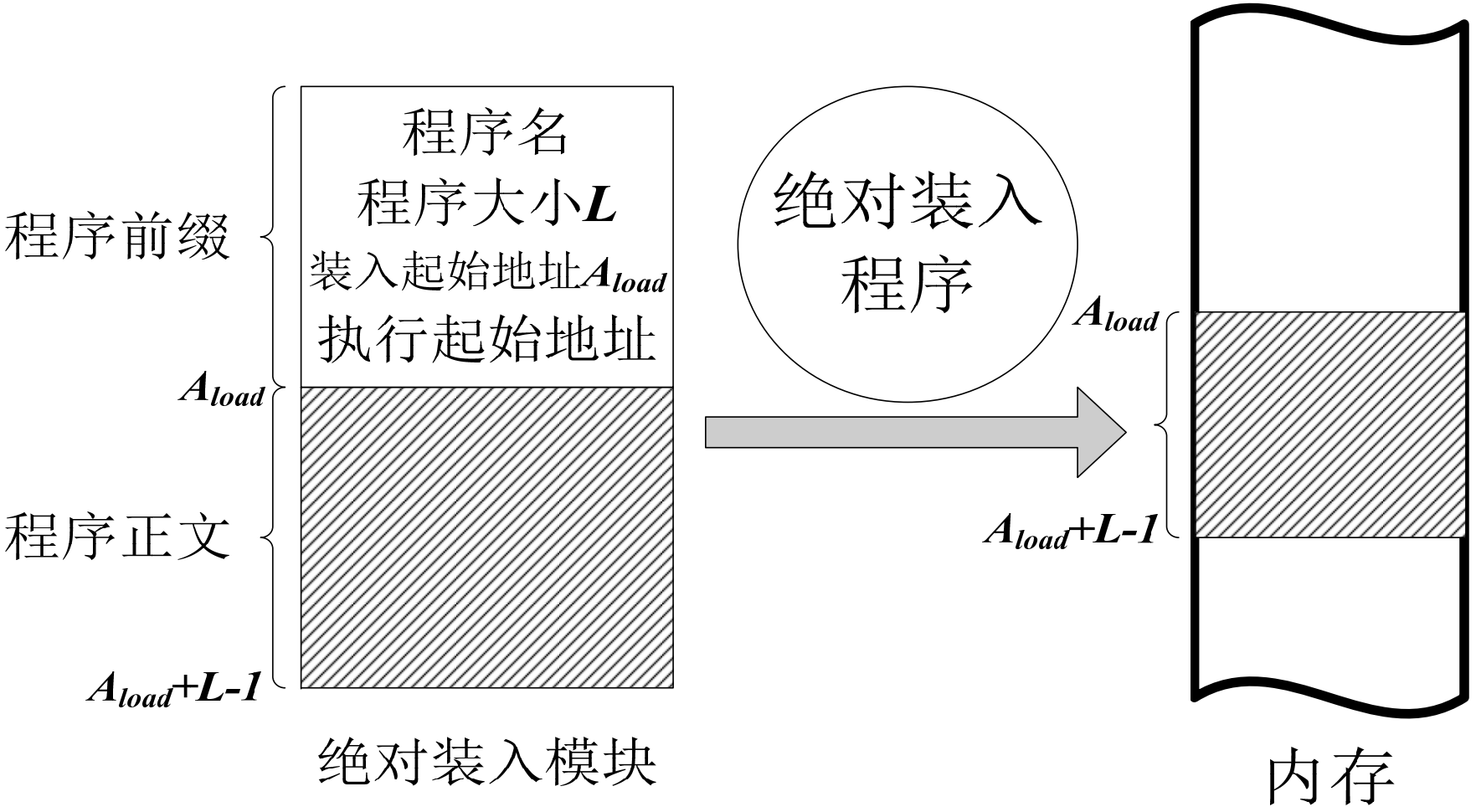


3.2.1 程序的装入

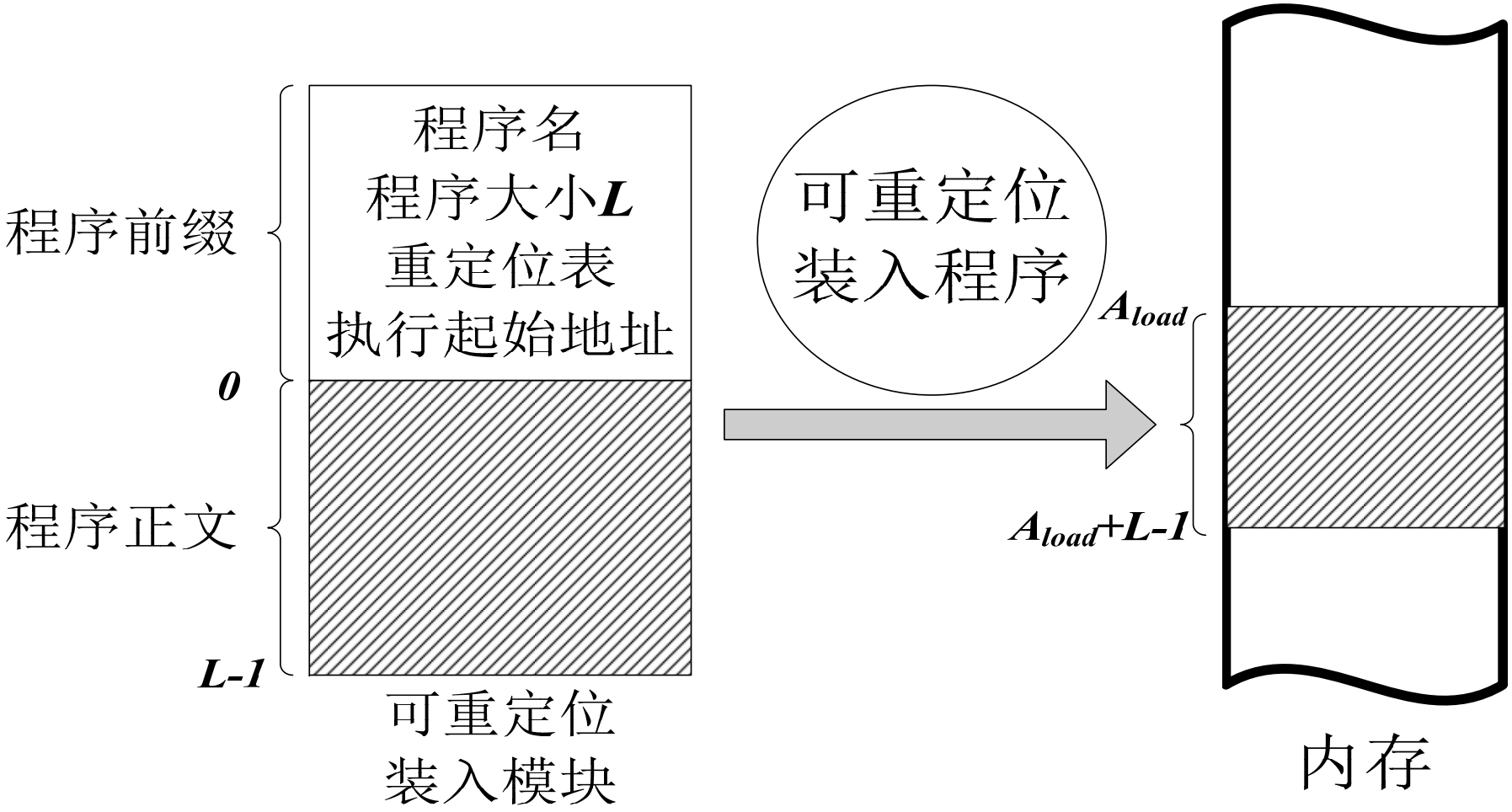
1. 绝对装入方式

程序中所使用的绝对地址，可在编译或汇编时给出， 也可由程序员直接赋予。 但在由程序员直接给出绝对地址时， 不仅要求程序员熟悉内存的使用情况，而且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，通常是宁可在程序中采用符号地址，然后在编译或汇编时，再将这些符号地址转换为绝对地址。

- 绝对装入模块及绝对装入方式



2. 可重定位装入方式


程序名
程序大小L
重定位表
执行起始地址
程序前缀
0
程序正文
L-1
可重定位
装入模块
可重定位
装入程序
Aload
Aload+L-1
内存

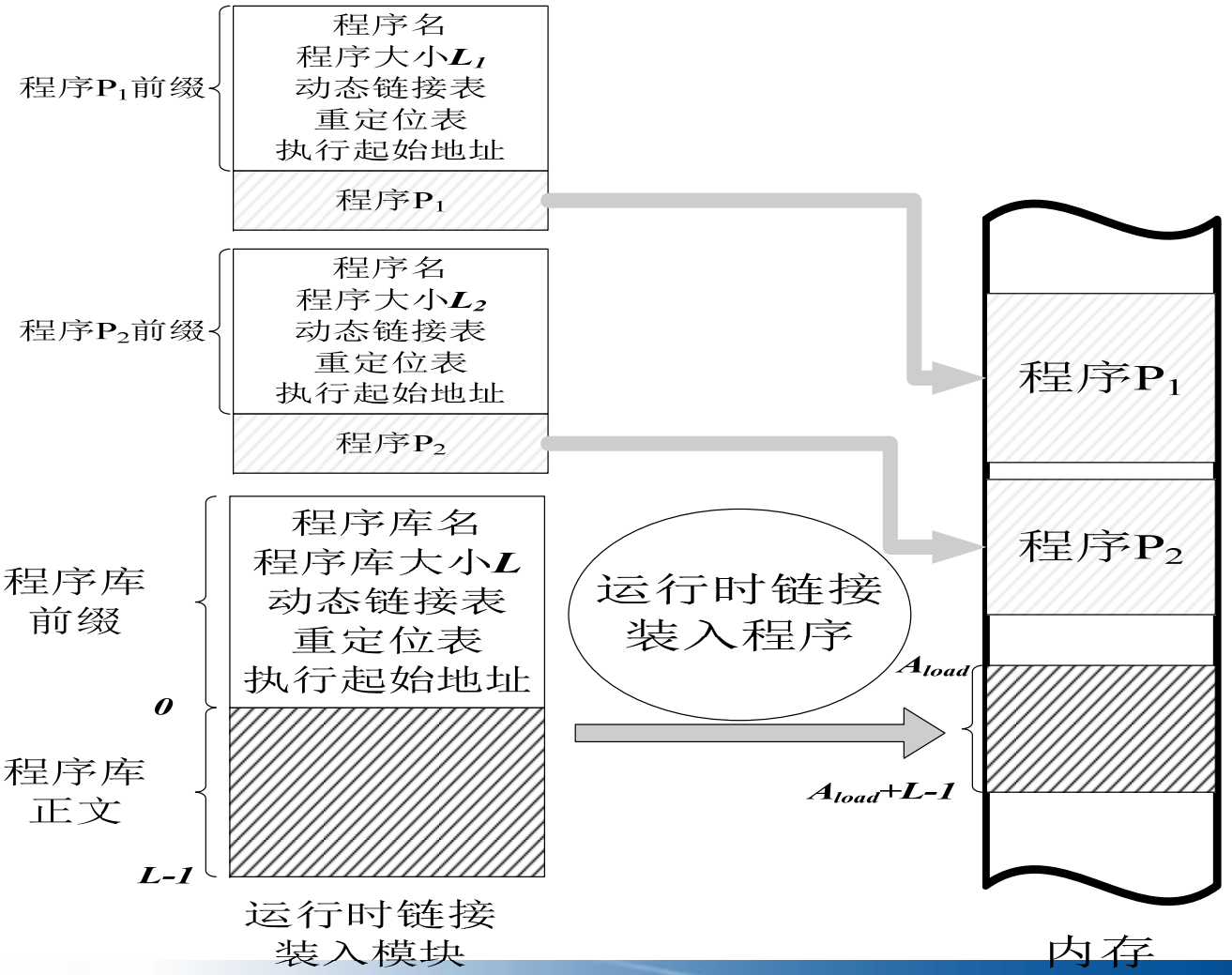
- 静态可重定位装入方式和动态可重定位装入方式
  - 如果在程序装入时一次性地完成程序中所有地址敏感指令及数据的地址修正且以后不再改变，则称对应的地址变换为静态重定位。
  - 如果在程序装入时并不进行由相对地址到绝对地址的转换过程，而是伴随程序执行进展来逐步进行程序中相关地址敏感指令及数据的地址修正，则称对应的地址变换为动态重定位。

- 静态可重定位装入方式并不允许程序在装入之后的运行过程中发生内存位置的移动
- 动态可重定位装入方式及动态重定位过程通常需要一定的硬件机构支持以使地址转换不影响指令执行速度

3. 动态运行时装入方式

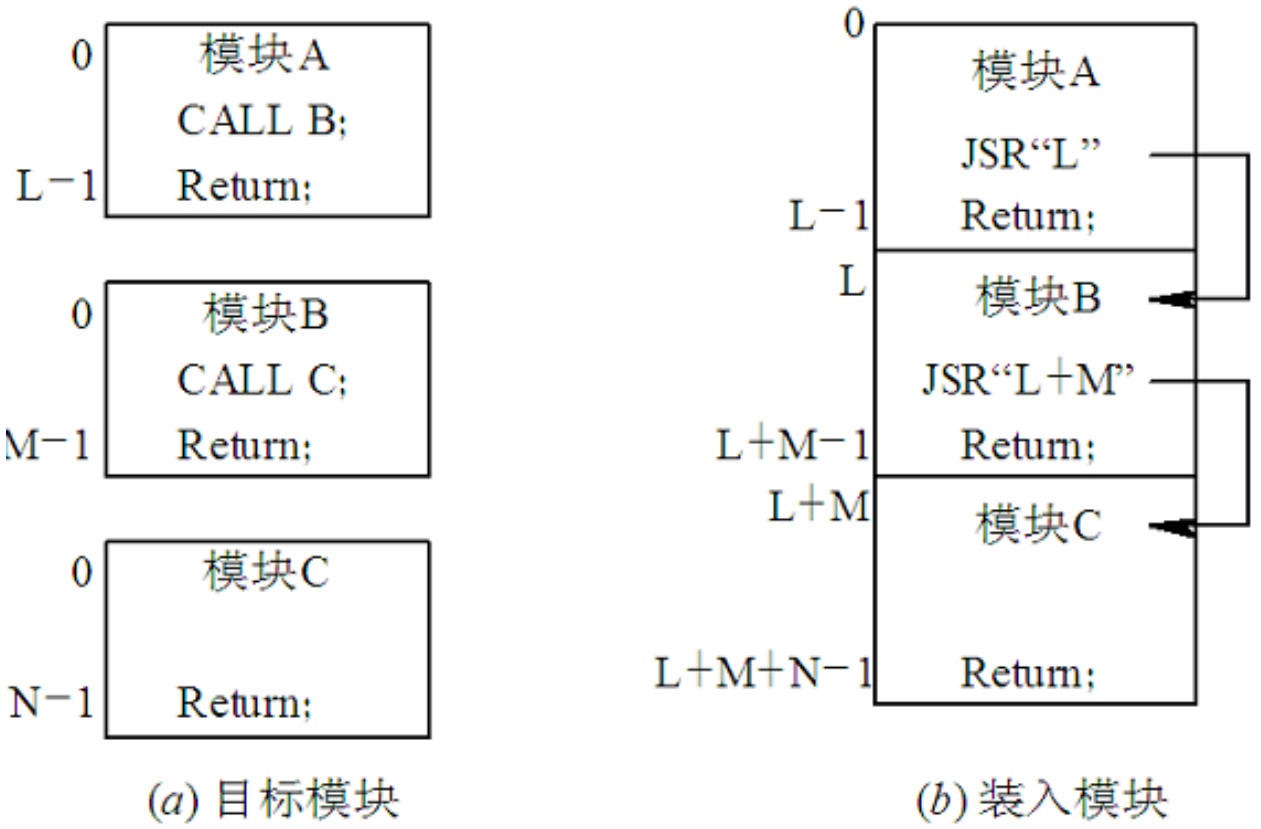
动态运行时的装入程序，在把装入模块装入内存后，并不立即把装入模块中的相对地址转换为绝对地址，而是把这种地址转换推迟到程序真正要执行时才进行。因此， 装入内存后的所有地址都仍是相对地址。

- 运行时链接装入模块及运行时链接装入方式



3.2.2 程序的链接

1. 静态链接方式



程序链接示意图

2. 装入时动态链接

- 装入时动态链接方式:是指在程序装入内存之前并未进行程序各目标模块的链接，而是在装入时，一边装入一边链接。
- 优点:

(1) 便于修改和更新。

(2) 便于实现对目标模块的共享。

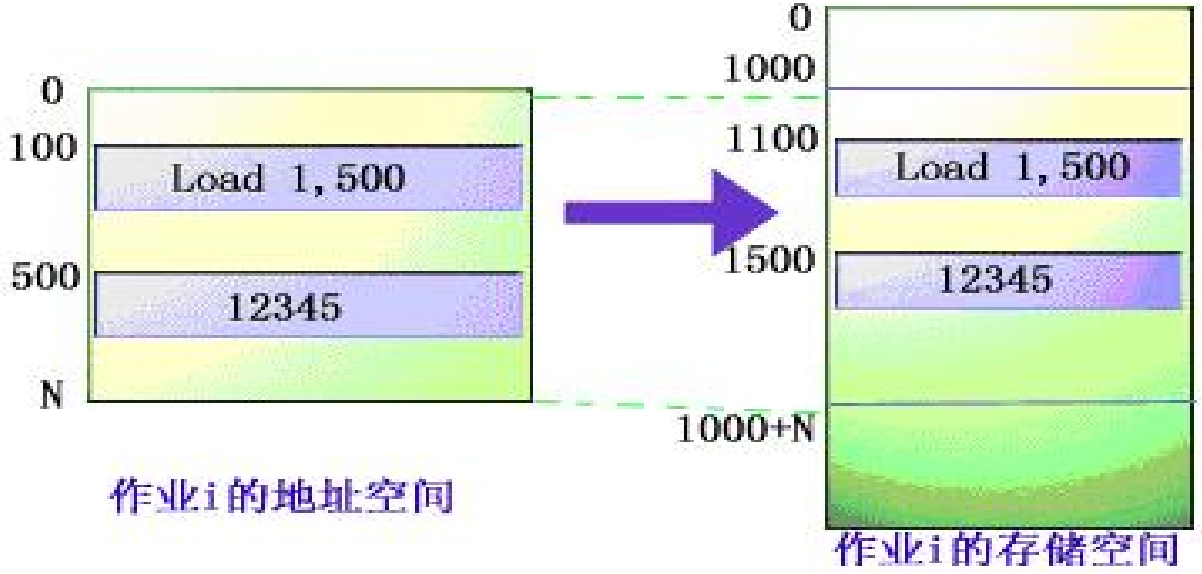
3. 运行时动态链接

这种链接方式是将对某些模块的链接推迟到执行时才执行，即，在执行过程中，当发现一个被调用模块尚未装入内存时，立即由**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。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。

3.2.3 重定位

把作业地址空间中使用的逻辑地址变换成内存空间中的物理地址的过程。又称地址映射。

如下图，作业i经过重定位，把地址集合映射到以**1000**为始址的内存中，作为作业i的存储空间。

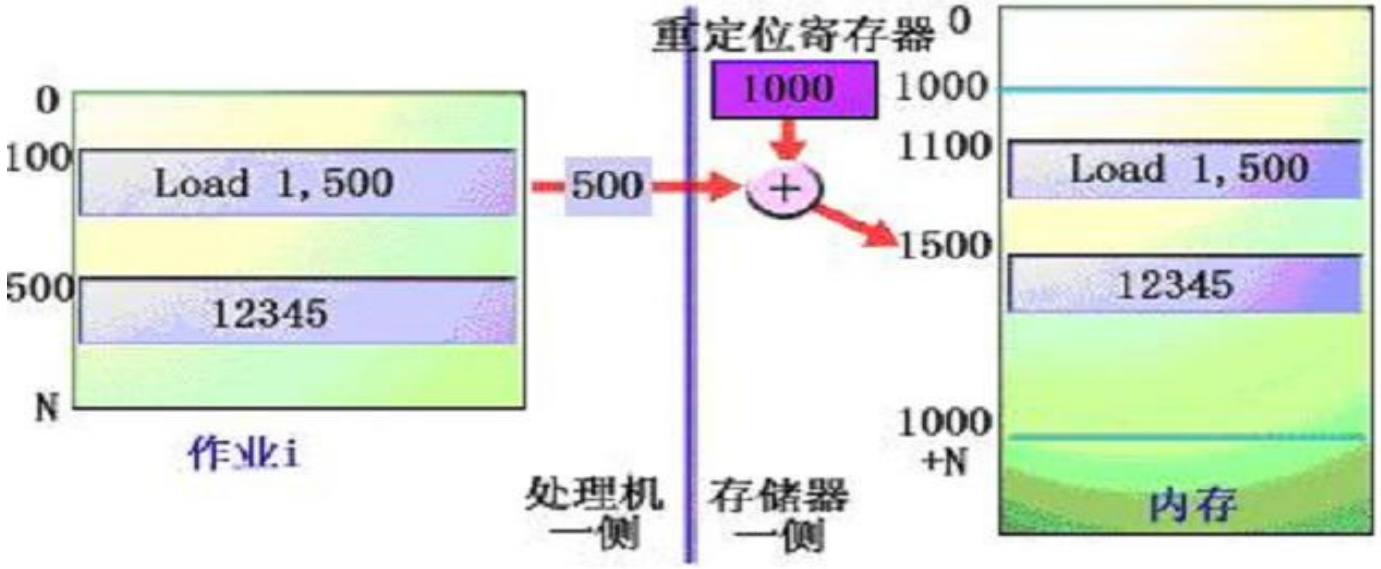


1. 重定位的类型

1)静态重定位: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）作业i在执行前一次变址，直到该作业完成退出内存为止。

2) 动态重定位

在程序运行过程中要访问数据时再进行地址变换。由地址变换机构进行的地址变换，硬件上需要重定位寄存器的支持。



2. 动态重定位的实现方式

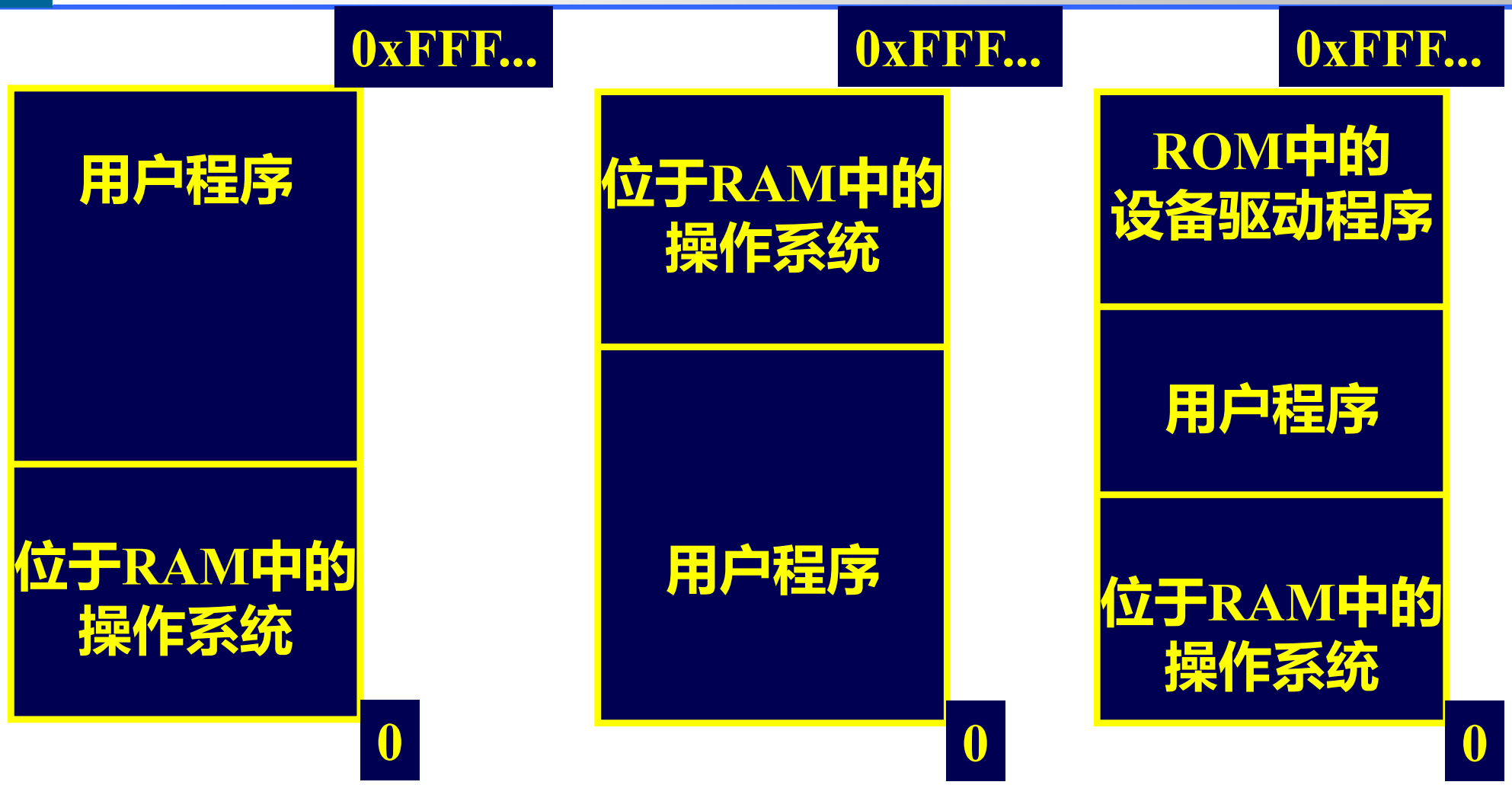
- 重定位寄存器：在执行一条指令取操作数时，要将指令给出的有效地址**(500)**与重定位寄存器中的内容（**1000**）相加，得访问地址（**1500**），从而实现了地址动态修改。
- 映象方式：采用页表来描述虚、实页面的对应关系 。

3.3 连续分配存储管理

连续内存分配：每个进程位于一个连续的内存空间。

3.3.1 单一连续内存管理

- 在单道环境下，不管是单用户系统还是单道批处理系统，进程（作业）执行时除了系统占用一部分主存外，剩下的主存区域全部归它占用。主存可以划分为三部分： 系统区、用户区、空闲区。用户占用区是一个连续的存储区所以又称单一连续区存储管理。
- 单用户系统在一段时间内，只有一个进程在内存，故内存分配管理十分简单，内存利用率低。内存分为两个区域，一个供操作系统使用，一个供用户使用

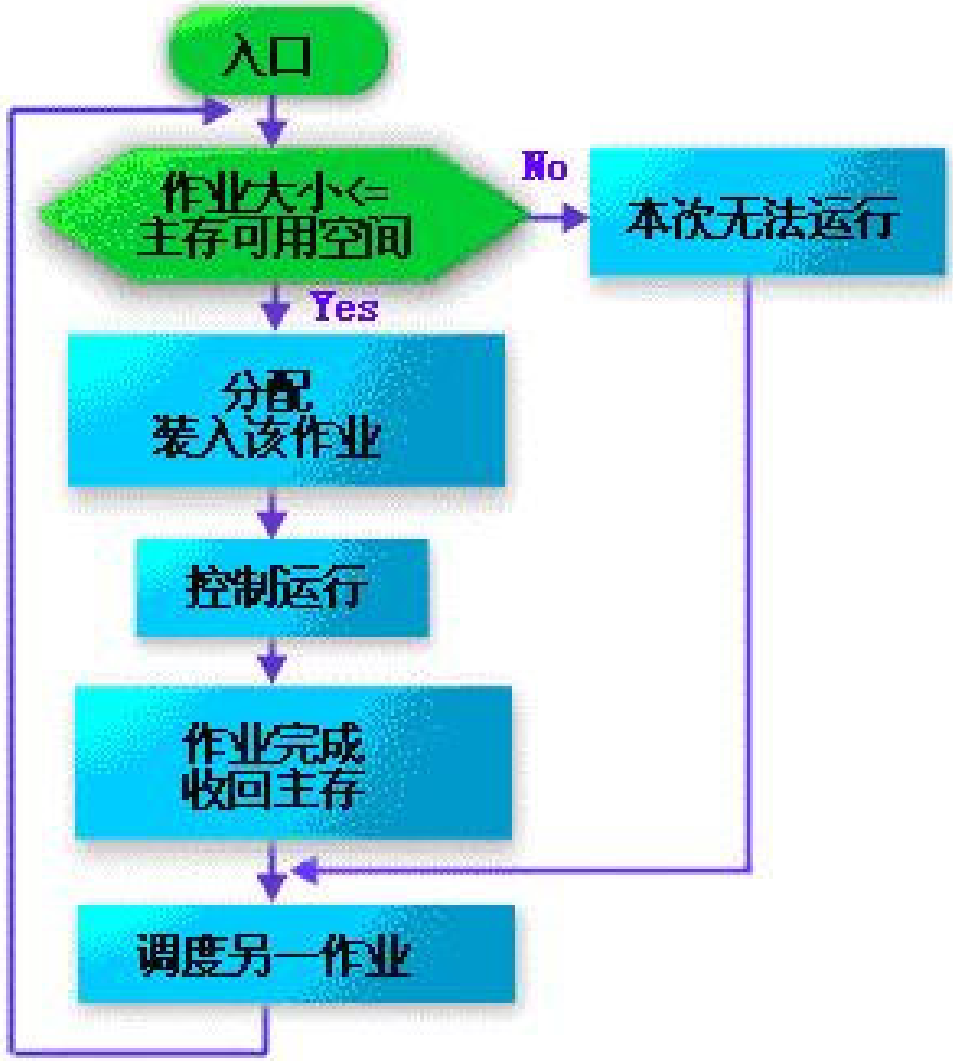


单一连续区存储分配示意图

- 工作流程

单一连续区分配采用静态分配和静态重定位方式，亦即作业或进程一旦进入主存，就一直等到它运行结束后才能释放主存。如下图所示的主存分配与回收法。并且由装入程序检查其绝对地址是否超越，即可达到保护系统的目的。

工作流程(续)



缺点

- 不支持多道程序。
- 主存利用率不高。
- 程序的运行受主存容量限制。

存储保护

- 自动地址修改　例如，存储器的地址空间为１２Ｋ，而操作系统位于低址端的４Ｋ内。对于这样的系统，我们给用户一个１３位的地址空间，并对其每个存储器访问自动加上４Ｋ。如果操作系统占用高址端的４Ｋ，则我们取每一个存储访问Ｒ，而实际上，其地址为（Ｒ　ｍｏｄ　８Ｋ）。从而实现了对操作系统的保护。

存储保护（续）

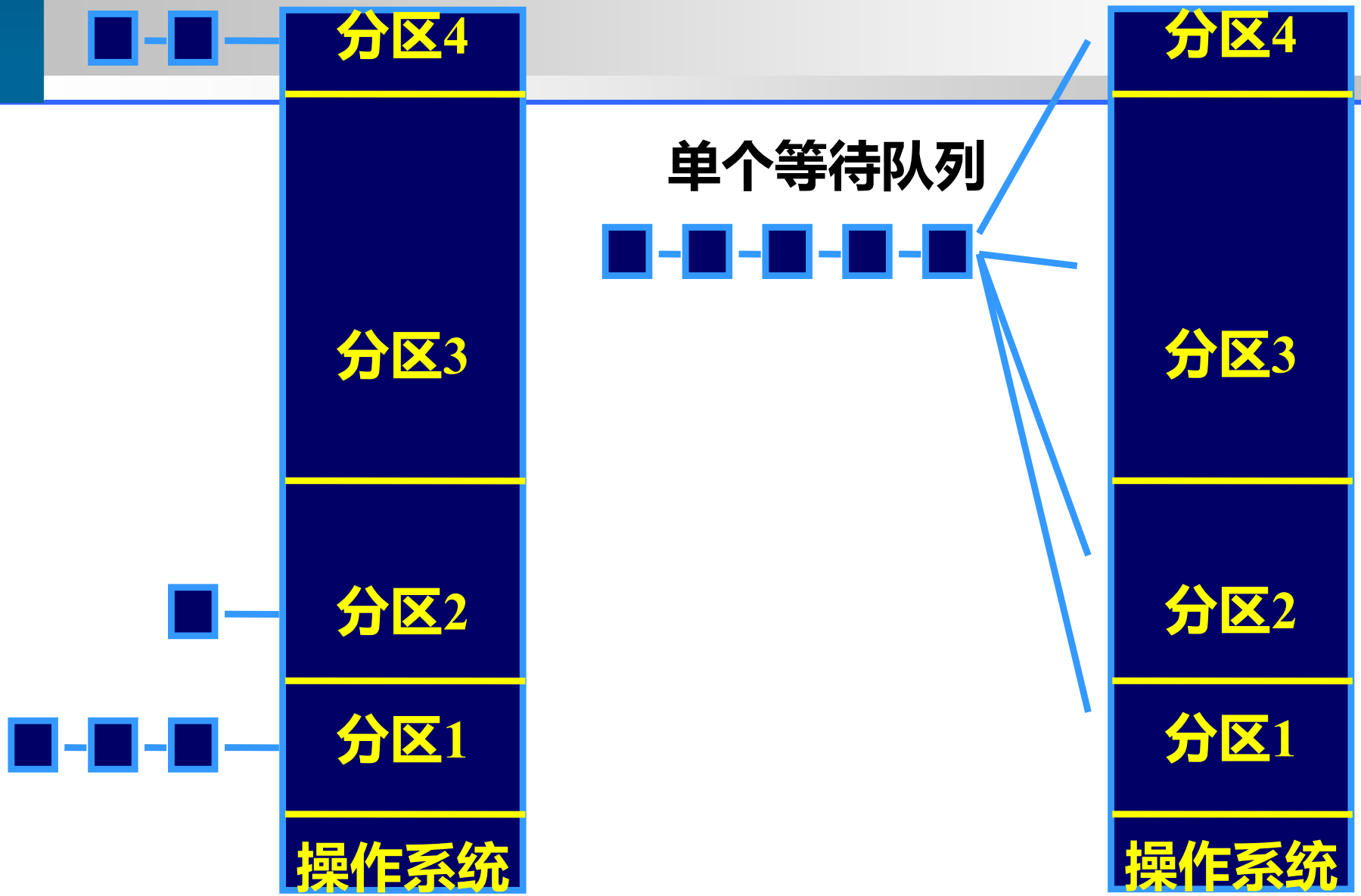
- 0 页、1 页寻址　通过对每个用户生成的地址左端拼接上一位 1 来实现 OS 区与用户区。把操作系统确定在 0 页，而把用户作业放在 1 页。
- 界限寄存器　通过增加界限寄存器，划分 OS 区与用户区。

3.3.2 固定分区分配

分区式管理是满足多道程序的最简单的存储管理方案。它的基本思想是将内存划分成若干个连续区域，称为分区。每个分区只能存储一个程序，而且程序也只能在它所驻留的分区中运行。

1. 固定分区

- 预先把可分配的主存储器空间分割成若干个连续区域，称为一个分区。每个分区的大小可以相同也可以不同，如图所示。但分区大小固定不变，每个分区装一个且只能装一个作业
- 存储分配：如果有一个空闲区，则分配给进程



固定分区示意图

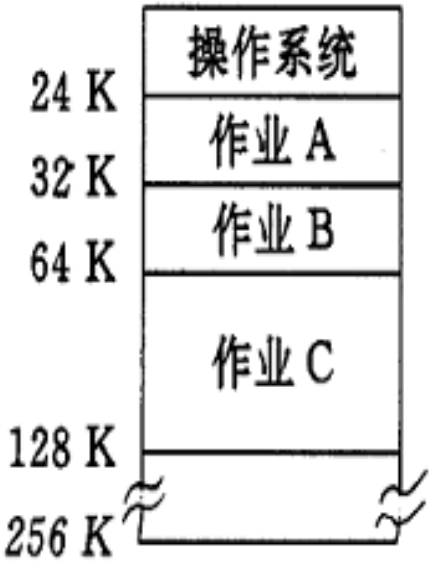
2. 内存分配管理

通过设置内存分配表，内存分配简单

缺点：内存利用率不高

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

固定分区使用表

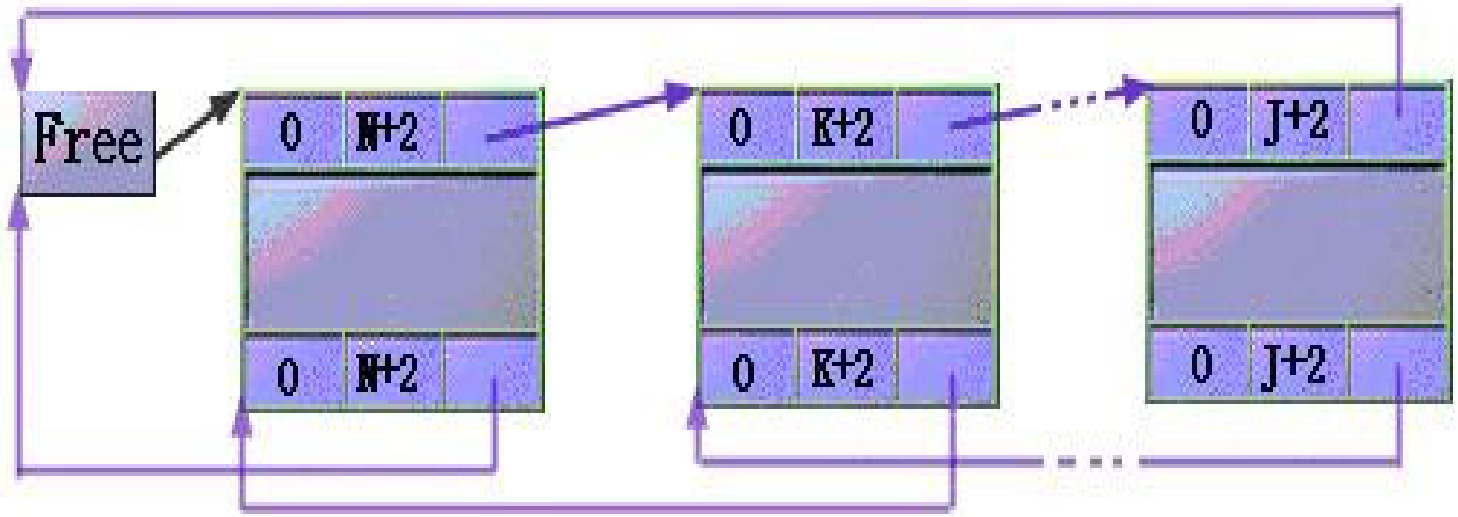
3.3.2 可变分区分配

- 基本思想：内存不是预先划分好的，而是当作业装入时，根据作业的需求和内存空间的使用情况来决定是否分配。若有足够的空间，则按需要分割一部分分区给该进程；否则令其等待主存空间
- 内存管理：设置内存空闲块表——记录了空闲区起始地址和长度
- 内存分配：动态分配
- 内存回收：当某一块归还后，前后空间合并，修改内存空闲块表

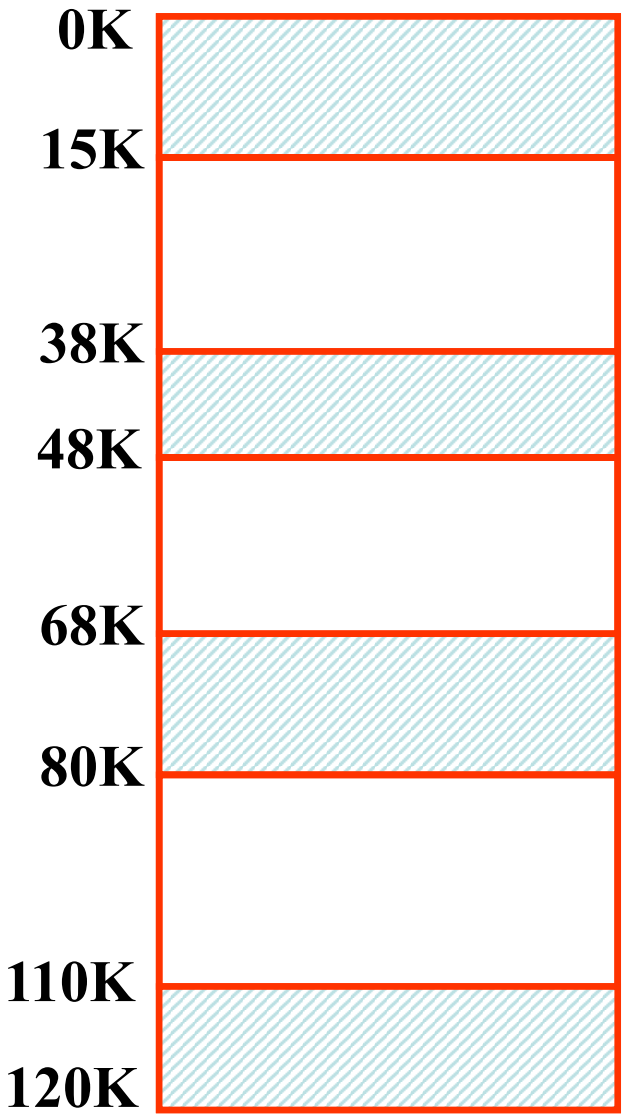
1. 分区分配中的数据结构

(1) 空闲分区链

系统设立空闲分区链表：每个空闲块的前后两个单元，放置必要的说明信息和指针。系统只要设立一个链首指针，指向第一个空闲块即可。分配程序可以依照自由块链表，来查找适合的空闲块进行分配。（如下图）



(2) 分区分配表



空闲区表

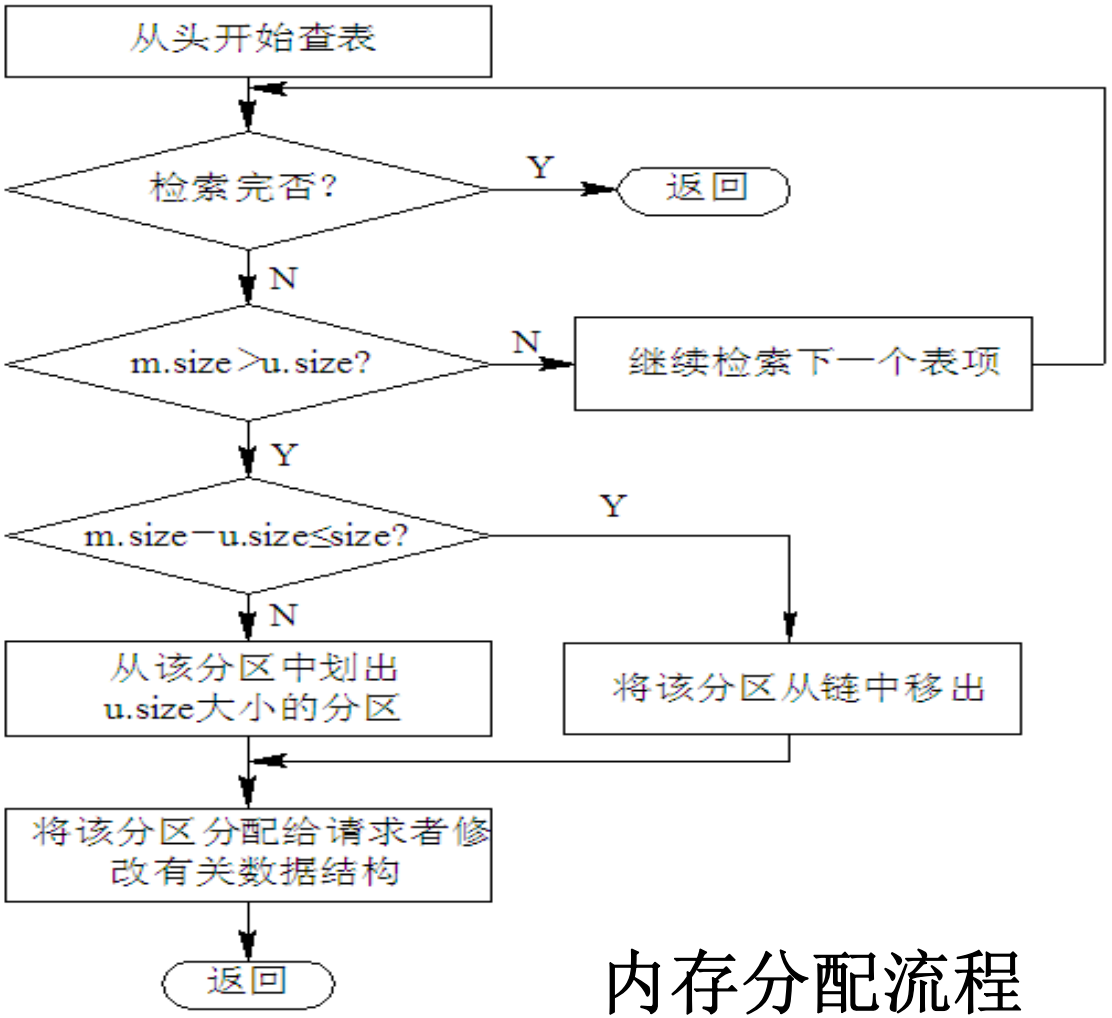
始址	长度	标志
15K	23K	未分配
48K	20K	未分配
80K	30K	未分配
		空
		空

已分配区表

始址	长度	标志
0K	15K	J1
38K	10K	J2
68K	12K	J3
110K	10K	J4
		空
		空

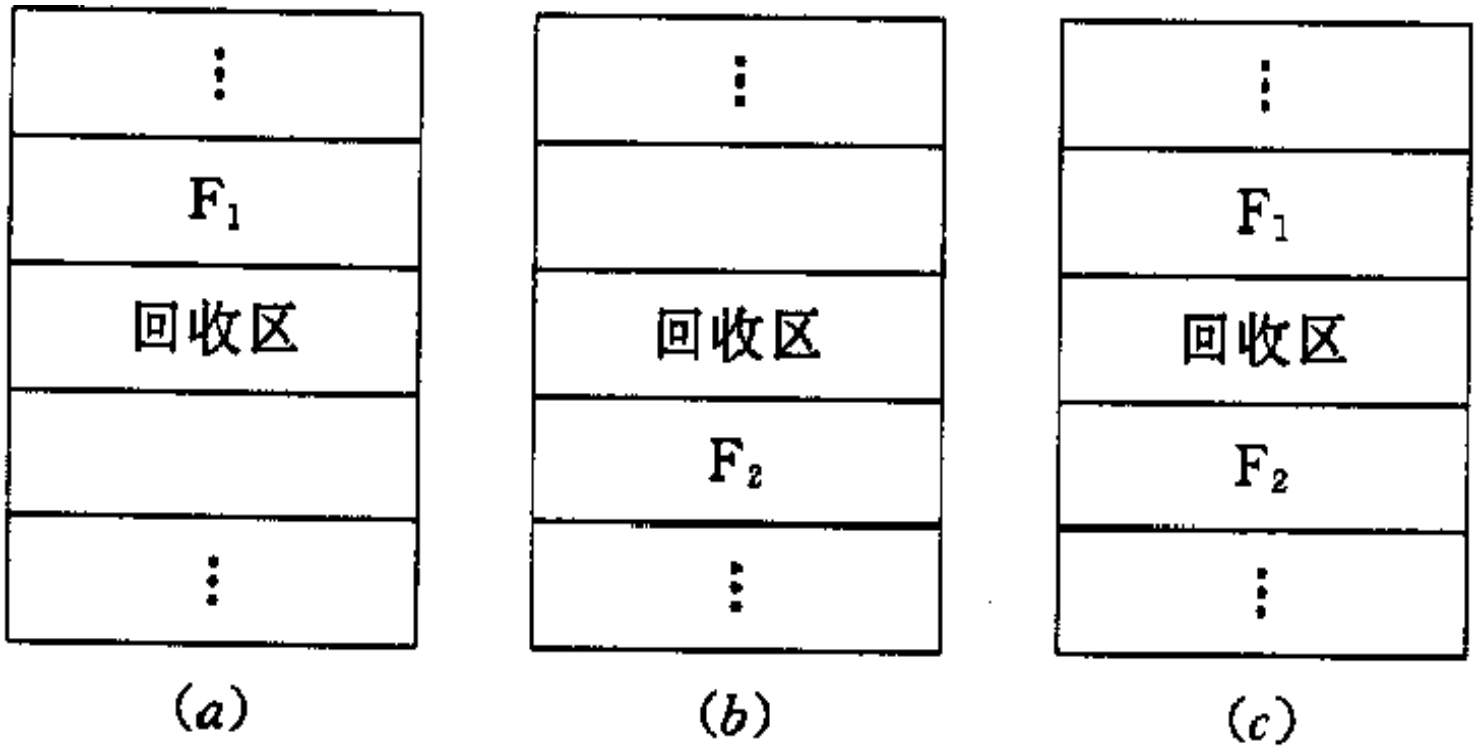
2. 分区分配操作

1) 分配内存



内存分配流程

2)　回收内存



内存回收时的情况

4. 分配算法

按空闲块链接的方式不同，可以有以下四种算法：

- 最佳适应法
- 最坏适应法
- 首次适应法
- 循环首次适应法

分配算法（续）

1）最佳适应算法

- 接到内存申请时，在空闲块表中找到一个不小于请求的最小空块进行分配
- 为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。
- 特点：用最小空间满足要求

分配算法（续）

2) 最坏适应算法

- 接到内存申请时，在空闲块表中找到一个不小于请求的最大空块进行分配，与最佳适应法相反，它在作业选择存储块时，总是寻找最大的空白区。
- 特点：当分割后空闲块仍为较大空块

分配算法（续）

3) 首次适应法：

- 为作业选择分区时总是按地址从高到低搜索，只要找到可以容纳该作业的空白块，就把该空白块分配给该作业。

4) 循环首次适应法

- 类似首次适应法每次分区时，总是从上次查找结束的地方开始，使得内存空间利用更加均衡。

5. 碎片问题

- 经过一段时间的分配回收后，内存中存在很多很小的空闲块。它们每一个都很小，不足以满足分配要求；但其总和满足分配要求。这些空闲块被称为碎片
- 造成存储资源的浪费

碎片问题的解决

- 紧凑技术：通过在内存移动程序，将所有小的空闲区域合并为大的空闲区域

  （紧缩技术，紧致技术，浮动技术，搬家技术）
- 问题：开销大；移动时机

6. 分区式存储管理的优缺点

优点：　便于动态申请内存

便于共享内存

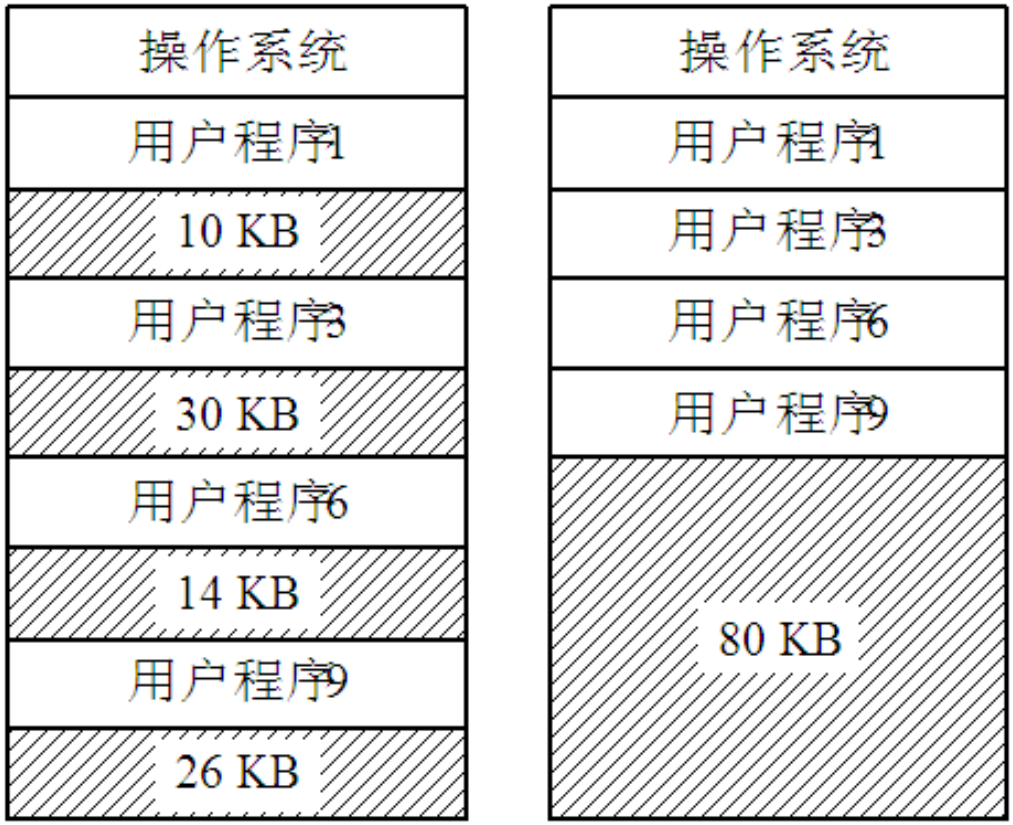
便于动态链接

缺点：

碎片问题(外碎片)，内存利用率不高，受实际内存容量限制

3.3.3 可重定位分区分配

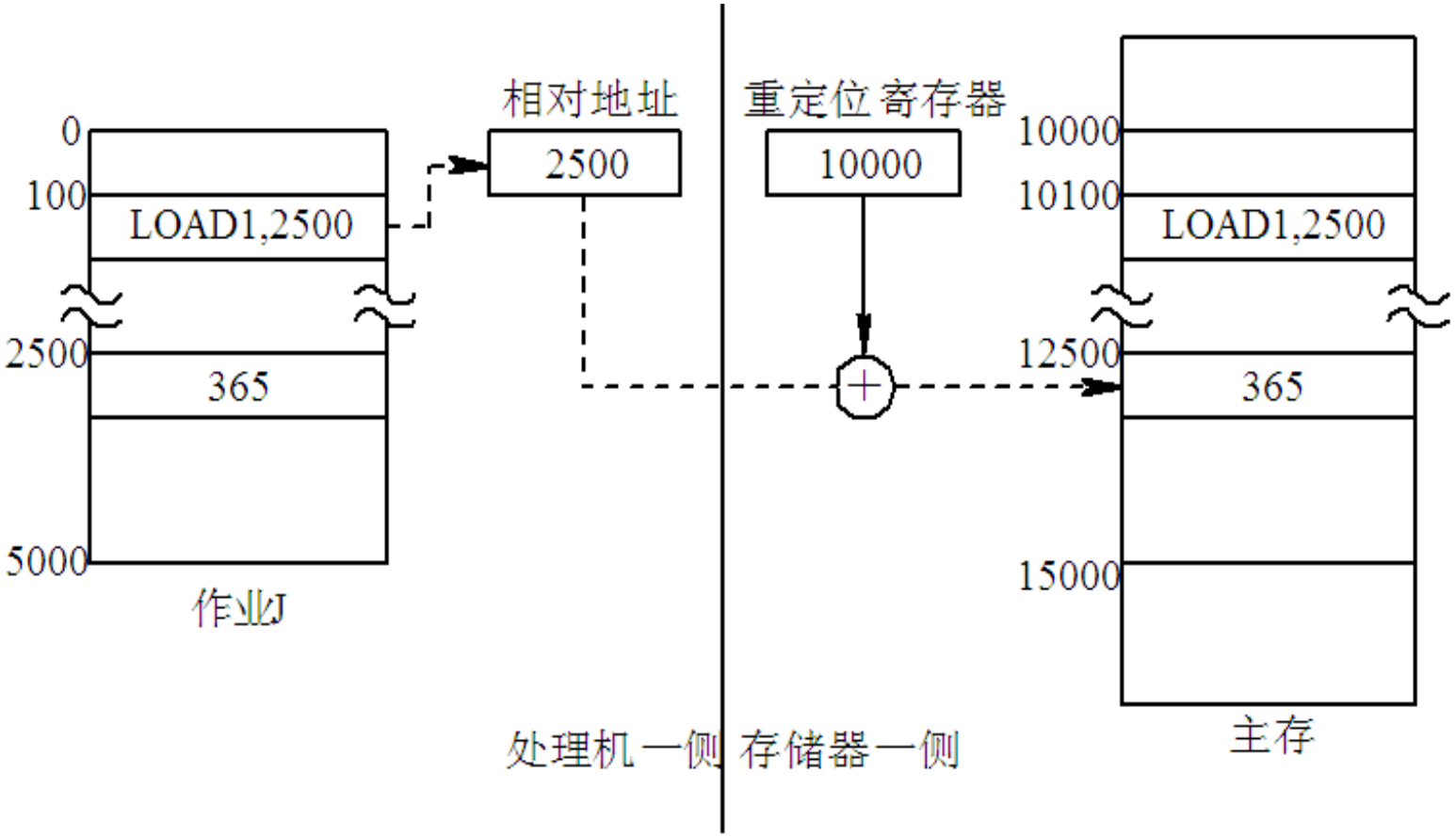
1. 动态重定位的引入



(a) 紧凑前　　(b) 紧凑后

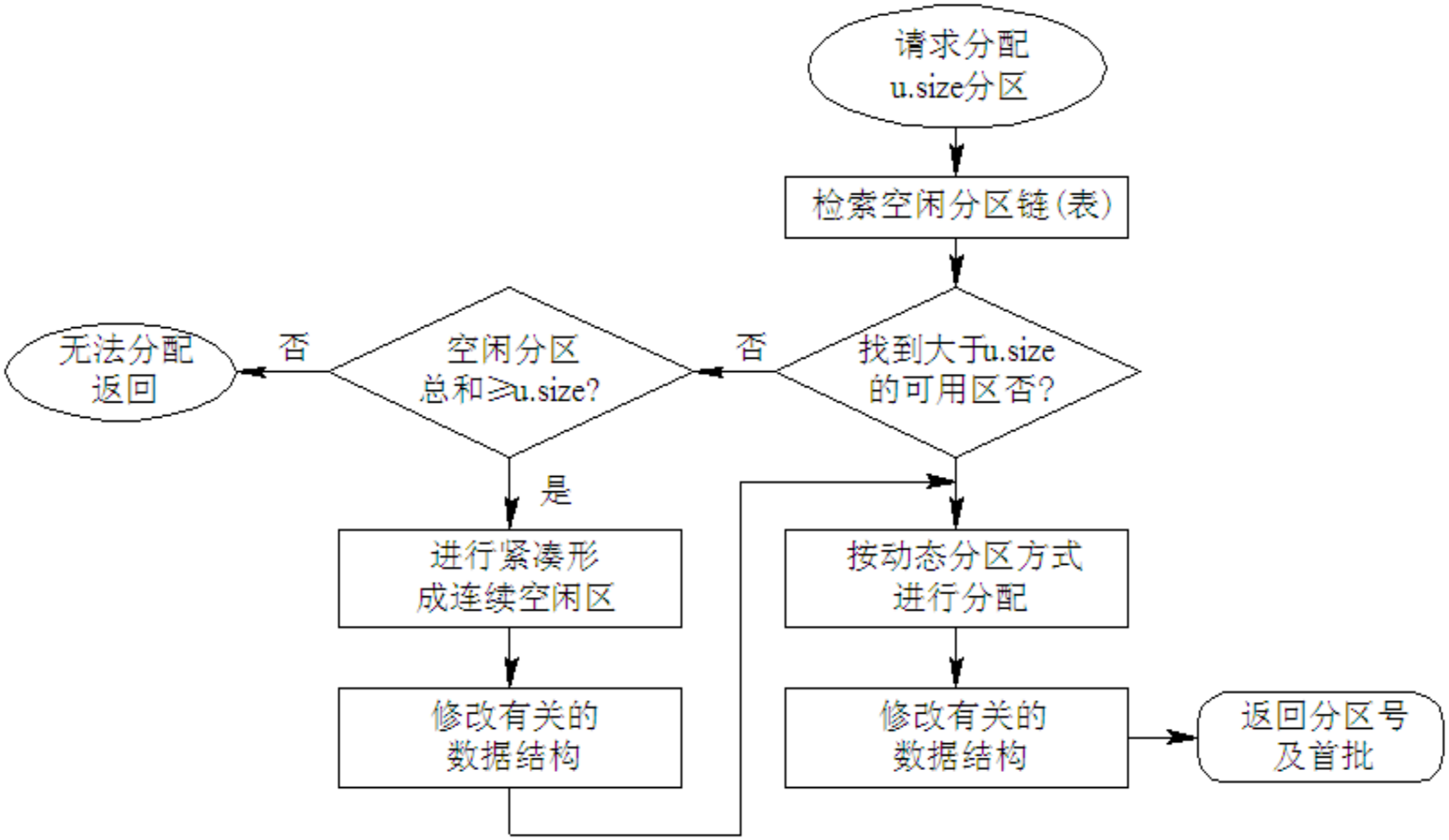
紧凑的示意

2. 动态重定位的实现



动态重定位示意图

3. 动态重定位分区分配算法



动态分区分配算法流程图

4. 可重定位分区的优缺点

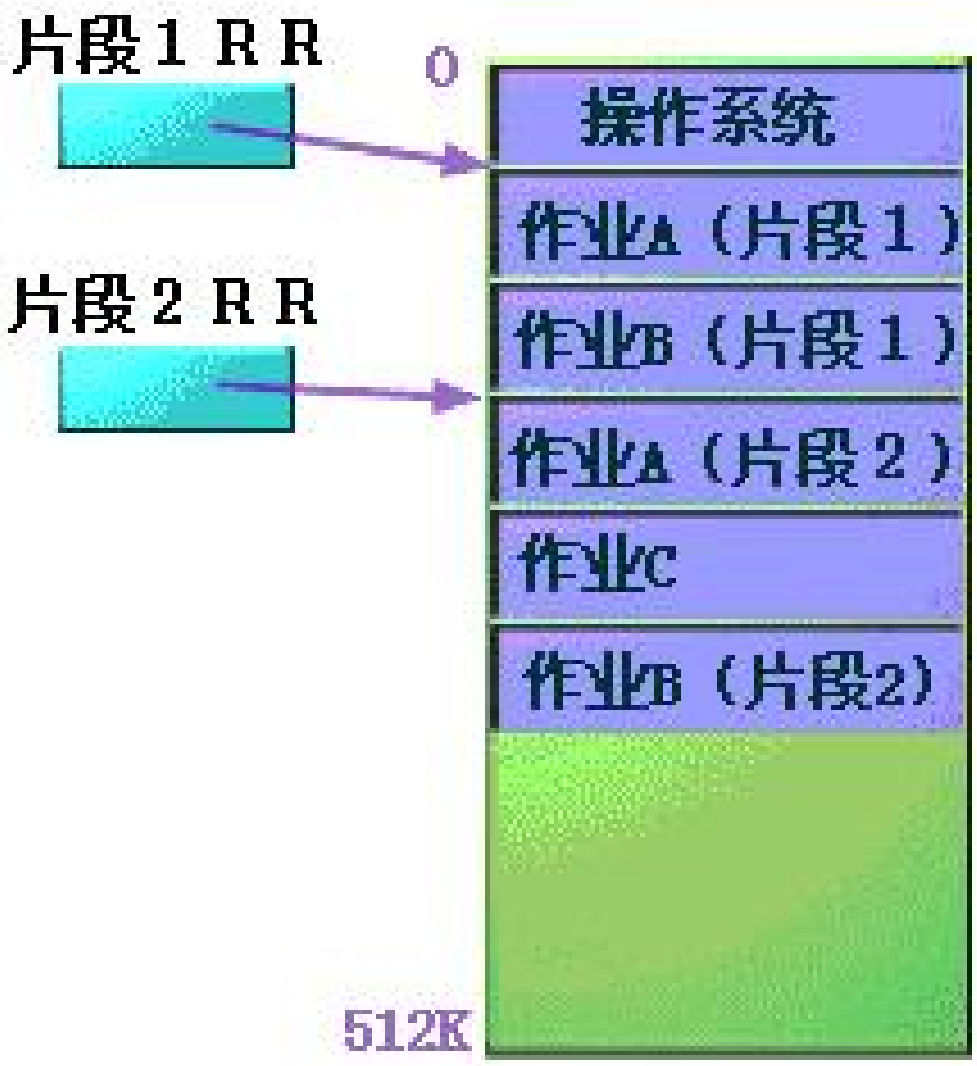
- 优点:解决了可变分区分配所引入的“外零头”问题。消除内存碎片，提高内存利用率。
- 缺点:提高硬件成本，紧凑时花费ＣＰＵ时间。

5. 多重分区

以上讨论都是基于一个作业在主存中占据的是一个连续分区的假定。为了支持结构化程序设计，操作系统往往把一道作业分成若干片段如子程序、主程序、数据组等）。这样，片段之间就不需要连续了。只要增加一些重定位寄存器，就可以有效地控制一道作业片段之间的调用。

如下图所示，作业A、B分别被分成两个片段放进互不相连的存储区域中。由两个变址寄存器实现控制。

多重分区分配



6. 分区的保护

为了防止一首作业有意或无意地破坏操作系统或其它作业。一般说来，没有硬件支持，实现有效的存储保护是困难的。通常采取：

- 界限寄存器方式
- 保护键方式
- 两种措施，或二者兼而有之。

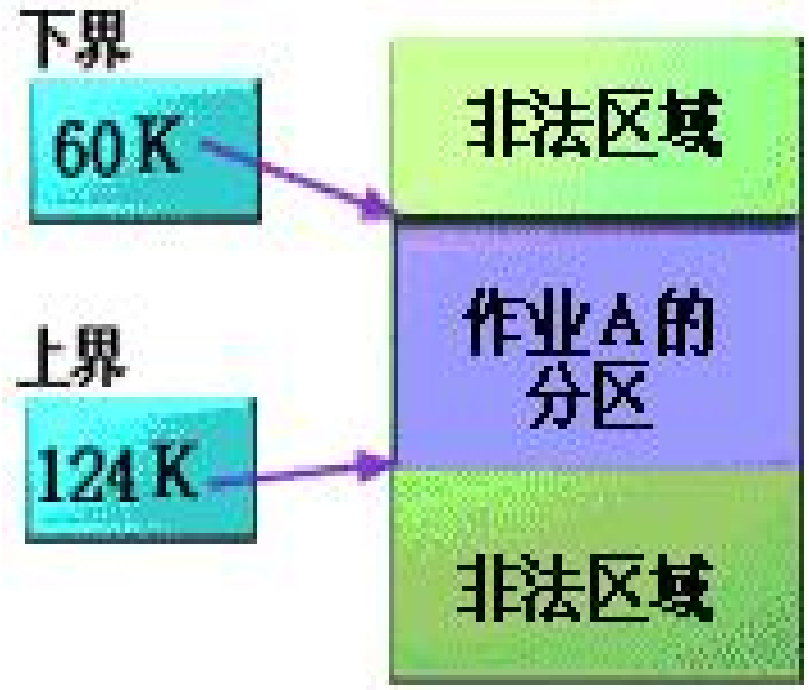
保护过程----防止地址越界

- 一般由硬件提供一对寄存器：

基址寄存器：存放起始地址
限长寄存器：存放长度
（上界寄存器/下界寄存器）

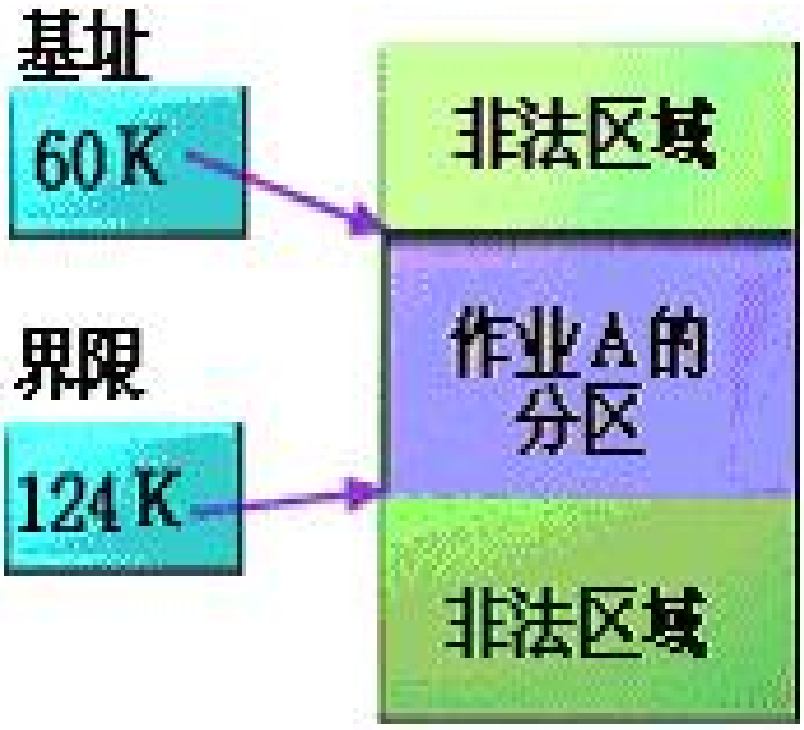
1) 界限寄存器保护

- **60K > 访问地址 > =124K**
- 则产生访问地址界中断



2) 基址、限长寄存器保护

- 相对地址 > 限长寄存器的值
- 则产生访问地址界中断



防止操作越权

对于允许多个进程共享的存储区域，每个进程都有自己的访问权限。如果一个进程对共享区域的访问违反了权限规定，则发生操作越权

即读写保护

3) 保护键方式

The End