



# 课程主要内容

---

☞ 操作系统引论（第1章）

☞ 进程管理（第2-3章）

☞ 存储器管理（第4-5章）

☞ 设备管理（第6章）

☞ 文件管理（第7-8章）

☞ 操作系统接口（第9章）

☞ UNIX操作系统



## 第4章 存储器管理

---

存储器是计算机系统的重要组成部分，是计算机系统的一种宝贵而紧俏的资源。操作系统中的存储管理是指对内存的管理，它是操作系统的重要功能之一。

存储管理的主要任务是为多道程序的运行提供良好的环境，方便用户使用存储器，提高存储器的利用率以及从逻辑上扩充存储器。为此

**存储管理应具有以下功能：**

- 实现内存的分配和回收
- 地址变换
- “扩充”内存容量
- 进行存储保护



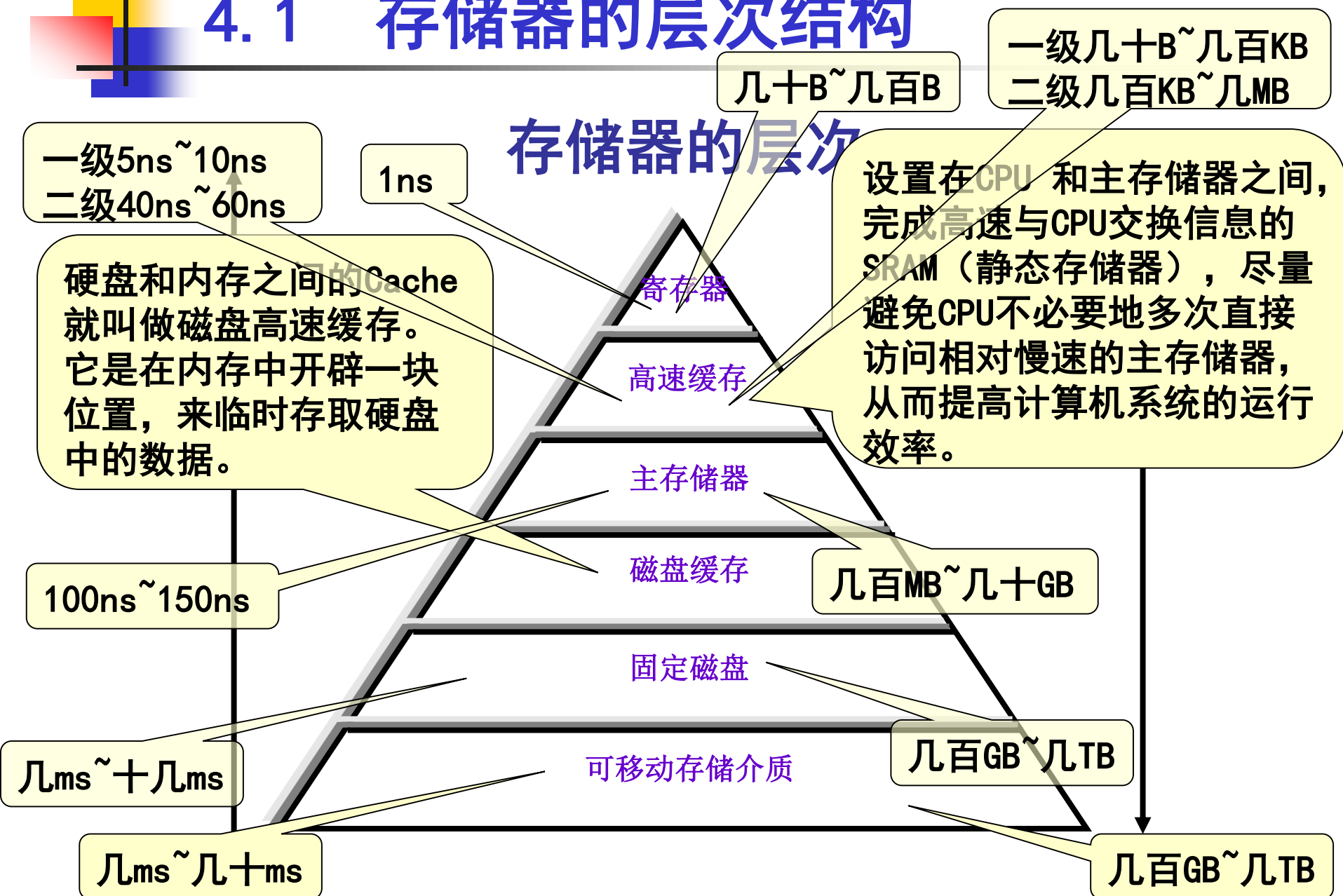
# 第4章 存储器管理主要内容

---

- ❖ 存储器的层次结构
- ❖ 程序的装入和链接
- ❖ 连续分配存储管理方式
- ❖ 基本分页存储管理方式
- ❖ 基本分段存储管理方式
- ❖ 虚拟存储器的基本概念
- ❖ 请求分页存储管理方式
- ❖ 页面置换算法
- ❖ 请求分段存储管理方式
- ❖ UNIX系统中存储器管理

## 4.1

# 存储器的层次结构



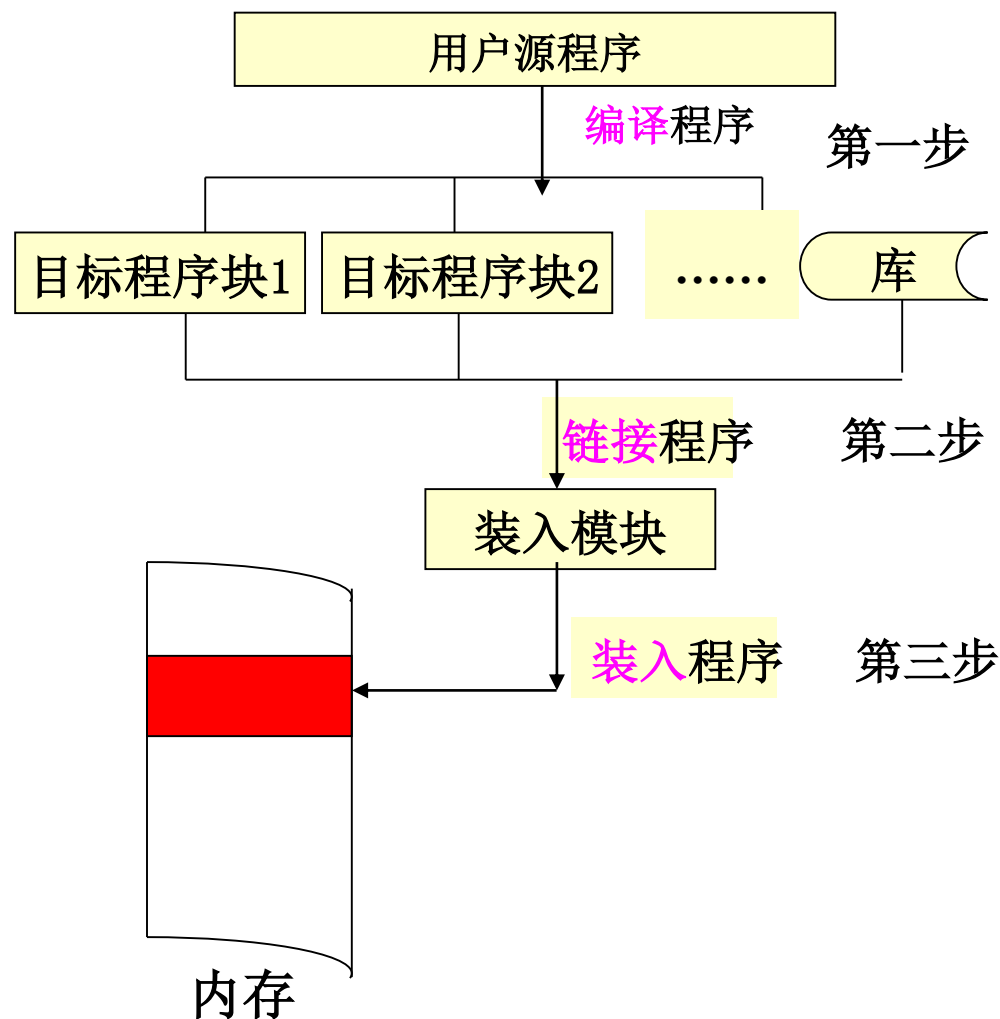
## 4.2 程序的装入和链接

在多道程序环境下，要使程序运行，必须创建进程，而创建进程第一件事就是将程序和数据装入内存。一个用户源程序要变为在内存中可执行的程序，通常要进行以下处理：

(1) **编译**：由编译程序将用户源程序编译成若干个目标模块

(2) **链接**：由链接程序将目标模块和相应的库函数链接成装入模块

(3) **装入**：由装入程序将装入模块装入内存





## 4.2 程序的装入和链接

---

### ❖ 程序的装入

- 绝对装入方式
- 可重定位装入方式
- 动态运行时装入方式

### ❖ 程序的链接

根据链接时间的不同，可将链接分成三种：

- 静态链接
- 装入时动态链接
- 运行时动态链接



# 物理地址和逻辑地址（1）

- ❖ 内存的结构：由若干**存储单元**组成，以字节为单位。
  - 存储最小单位：“二进制位”，包含信息为0或1
  - 最小编址单位：字节，一个字节包含八个二进制位
- ❖ 内存地址：为了便于CPU访问，给每个**存储单元**一个编号（第一个字节的地址是0，后面依次是1、2、3，等等），也称为物理地址或绝对地址。
- ❖ 内存地址空间（存储空间）：内存地址的集合，也称物理空间，它是一维线性空间，其编址为  
 $0, 1, 2, \dots, n-1$



## 物理地址和逻辑地址（2）

### ❖ 逻辑地址（程序地址，相对地址，虚地址）

用户编制的源程序，存在于程序员建立的符号名字空间内，经过汇编或编译后形成若干**目标代码**，这些目标代码连接后形成**可装入程序**，这些程序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。

### ❖ 作业地址空间（地址空间）：由逻辑地址组成的空间，也称为地址空间。





# 1、绝对装入方式

---

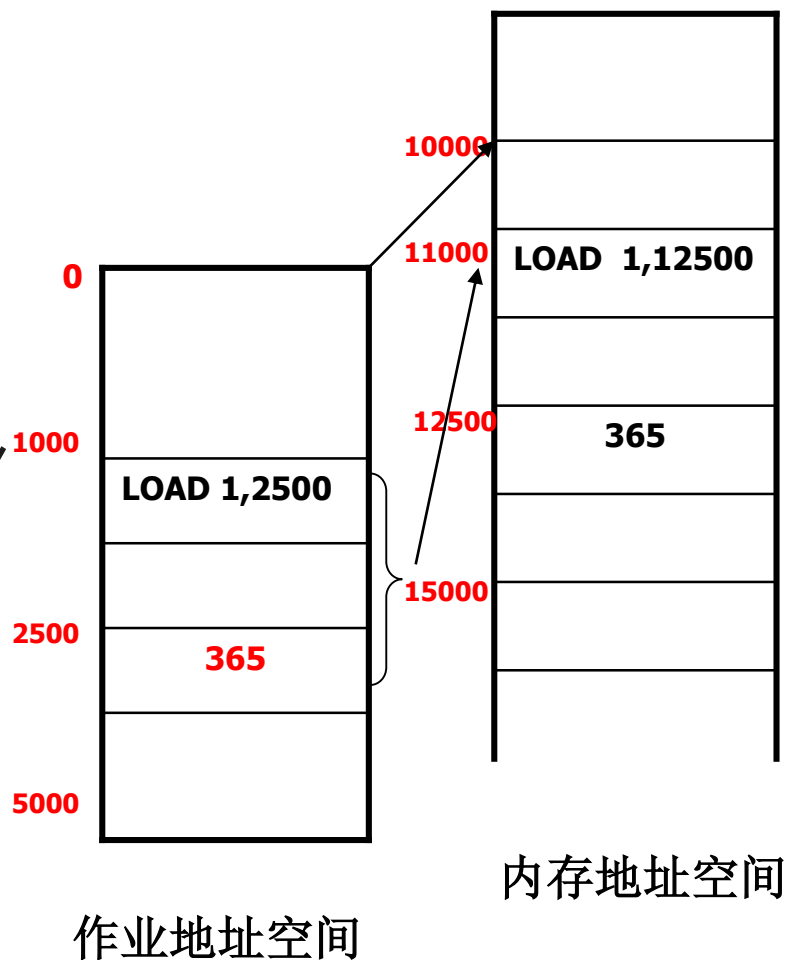
如果在编译时，事先知道用户程序在内存的驻留位置，则编译程序在编译时就产生绝对地址的目标代码。装入程序就直接把装入模块中的程序和数据装入到指定的位置（不需进行地址转换）。

这种装入方式只适用于单道程序环境。

## 2、可重定位装入方式（1）

❖ **重定位**：由于一个作业装入到与其地址空间不一致的存储空间所引起的需对其有关地址部分进行调整的过程就称为重定位（实质是一个地址变换过程/地址映射）。

❖ 根据地址变换进行的时间及采用技术手段不同，可分为**静态重定位**和**动态重定位**两类。



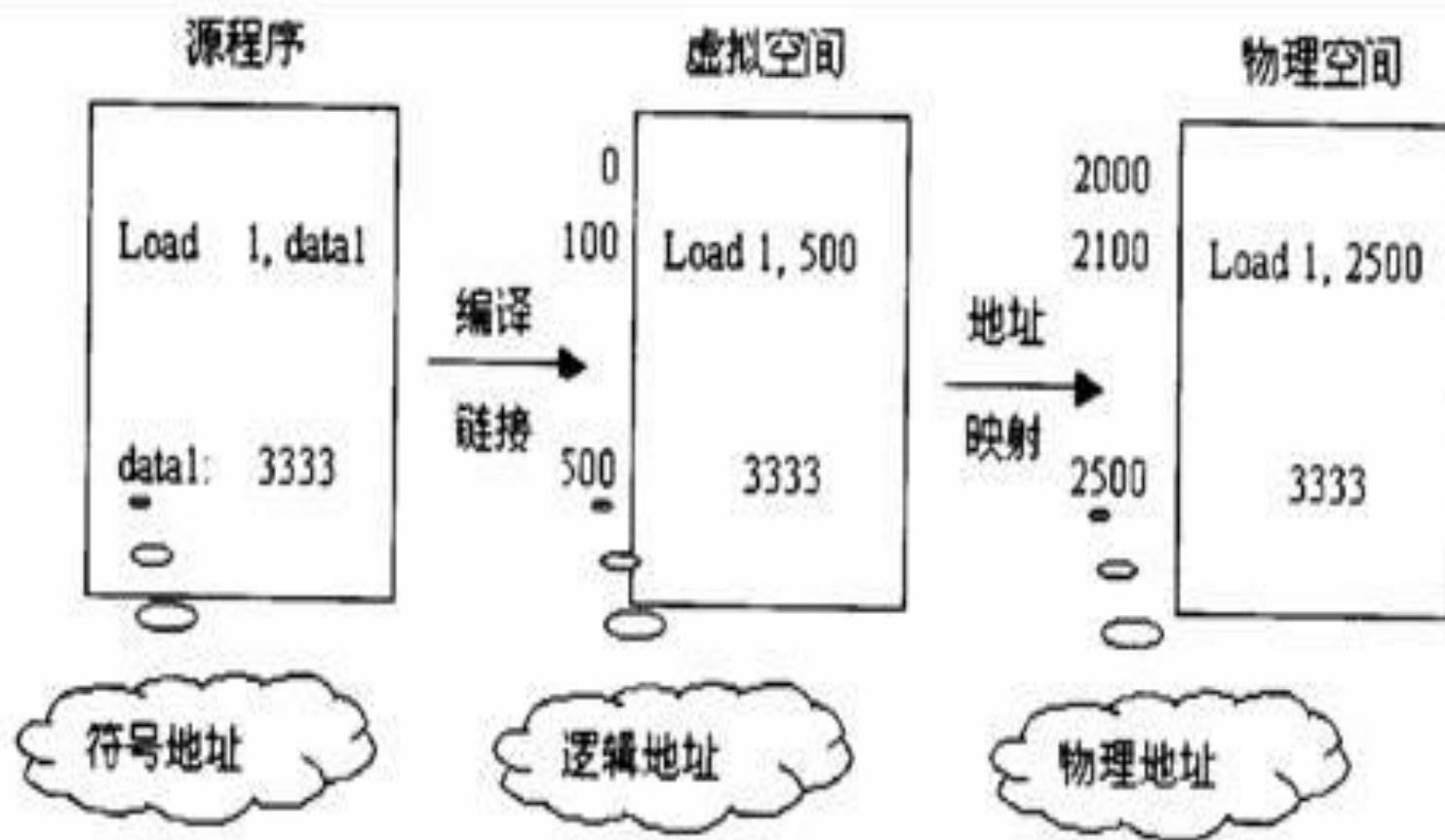


## 2、可重定位装入方式（2）

- ❖ **可重定位装入方式**：事先不知用户程序在内存的驻留位置，装入程序在装入时根据内存的实际情况把相对地址（逻辑地址）转换为绝对地址，装入到适当的位置（在装入时进行地址转换）。
- ❖ 地址变换在装入时一次完成，以后不再改变，称为**静态重定位**。

## 2、可重定位装入方式 (3)

### ❖ 用于多道程序环境



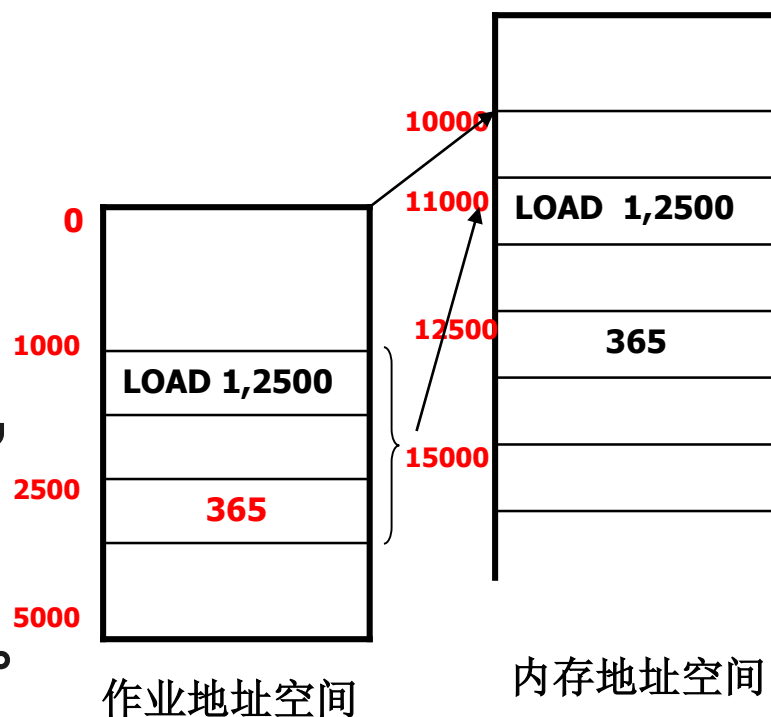
### 3、动态运行装入方式

如果事先不知用户程序在内存的驻留位置，为了保证程序在运行过程中，它在内存中的位置可经常改变，装入程序把装入模块装入内存后，并不立即把装入模块中相对地址转换为绝对地址，而是在程序运行时才进行。这种方式需一个重定位寄存器来支持。

（在程序运行过程中进行地址转换-**动态重定位**）

重定位寄存器

10000



## 二、程序的链接

### 1、静态链接方式

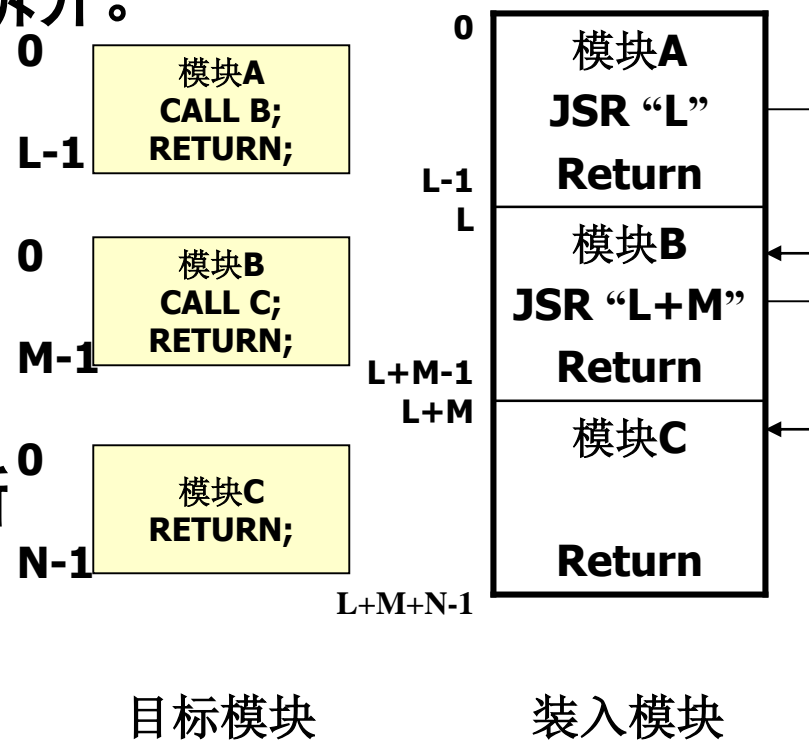
是一种事先链接方式，即在程序运行之前，先将各目标模块及它们所需的库函数，链接成一个完整的装入模块（执行文件），以后不再拆开。

实现静态链接应解决的问题：

- (1) 相对地址的修改
- (2) 变换外部调用符号

存在的问题：

- (1) 不便于对目标模块的修改和更新
- (2) 无法实现对目标模块的共享





## 二、程序的链接

### 2、装入时动态链接方式

指将一组目标模块在装入内存时，边装入边链接的方式。具有便于修改和更新、便于实现对目标模块的共享的优点。

**存在的问题：**

由于程序运行所有可能用的目标模块在装入时均全部链接在一起，所以将会把一些不会运行的目标模块也链接进去。如程序中的错误处理模块。

### 3、运行时动态链接方式

在程序运行中需要某些目标模块时，才对它们进行链接的方式。具有高效且节省内存空间的优点。



## 4.3 连续分配存储管理方式

- ❑ 连续/分区分配方式：指为一个用户程序分配一块连续的内存空间。
  - ❑ 单一连续分配方式
  - ❑ 固定分区分配方式
  - ❑ 动态分区分配方式
  - ❑ 动态重定位分区分配方式
- ❑ 分区的存储保护
- ❑ 对换

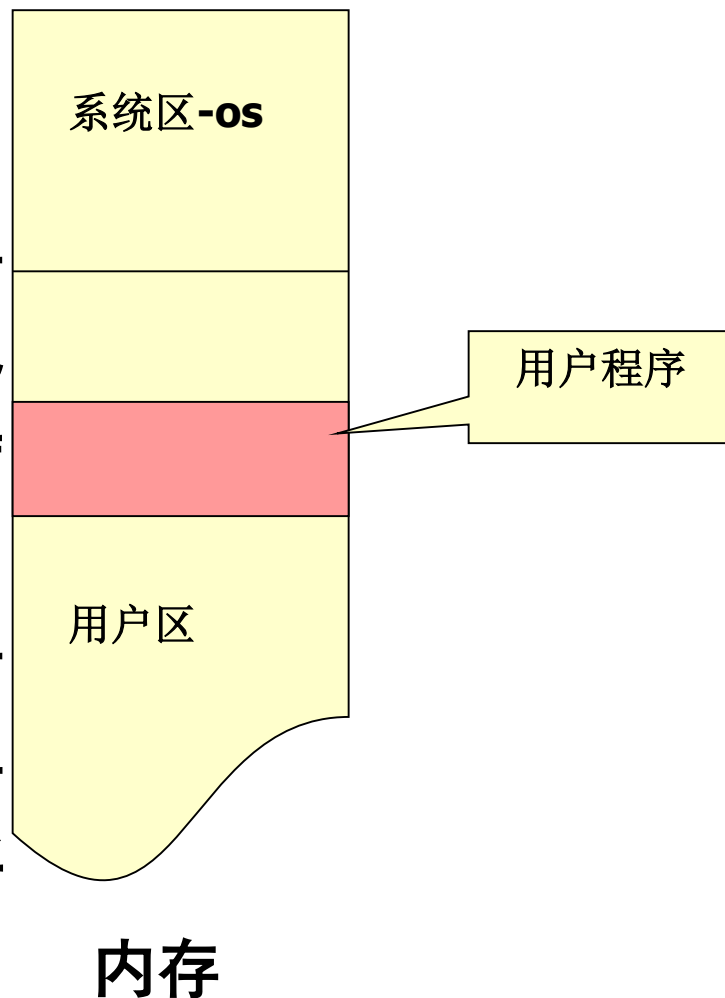


# 一、单一连续分配方式（单独分区分配）

最简单的一种存储管理方式，但只能用于单用户、单任务的OS中。

❖ **存储管理方法**：将内存分为系统区（内存低端，分配给OS用）和用户区（内存高端，分配给用户用）。采用静态分配方式，即作业一旦进入内存，就要等待它运行结束后才能释放内存。

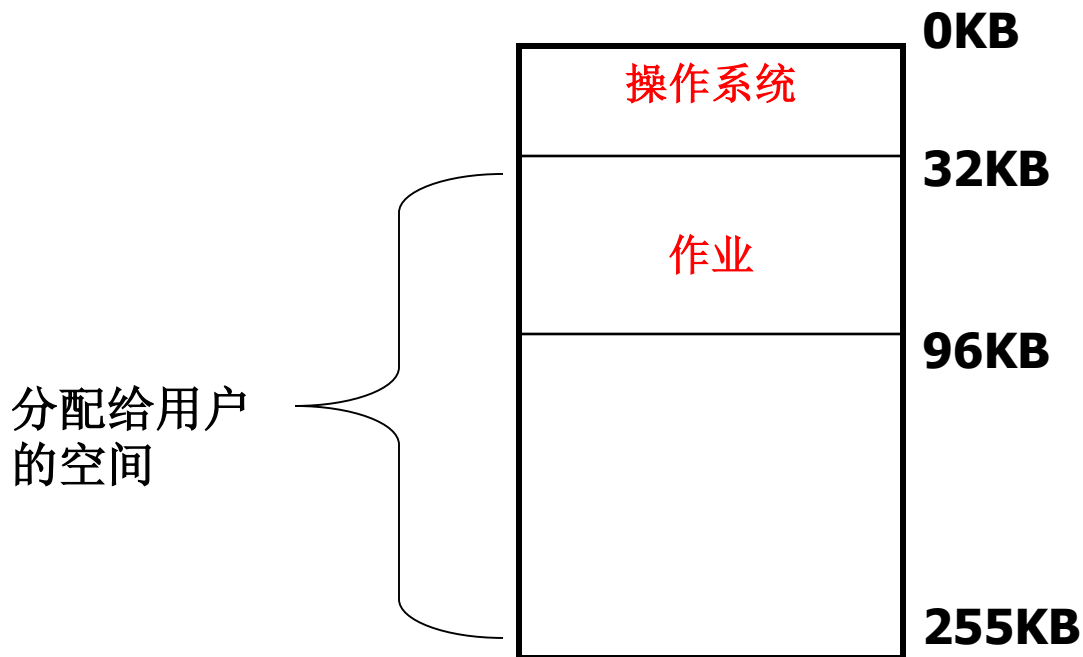
❖ **主要特点**：管理简单，只需少量的软件和硬件支持，便于用户了解和使用。但因内存中只装入一道作业运行，内存空间浪费大，各类资源的利用率也不高。





# 例：

- ❖ 一个容量为256KB的内存，操作系统占用32KB，剩下224KB全部分配给用户作业，如果一个作业仅需64KB，那么就有160KB的存储空间被浪费。





# 分区分配方式存储管理

---

分区分配方式是满足多道程序设计需要的一种最简单的存储管理方法。

## ❖ 存储管理方法

将内存分成若干个分区（大小相等/不相等），除OS占用一个分区外，其余的每一个分区容纳一个用户程序。按分区的变化情况，可将分区存储管理进一步分为：

- 固定分区存储管理
- 动态分区存储管理



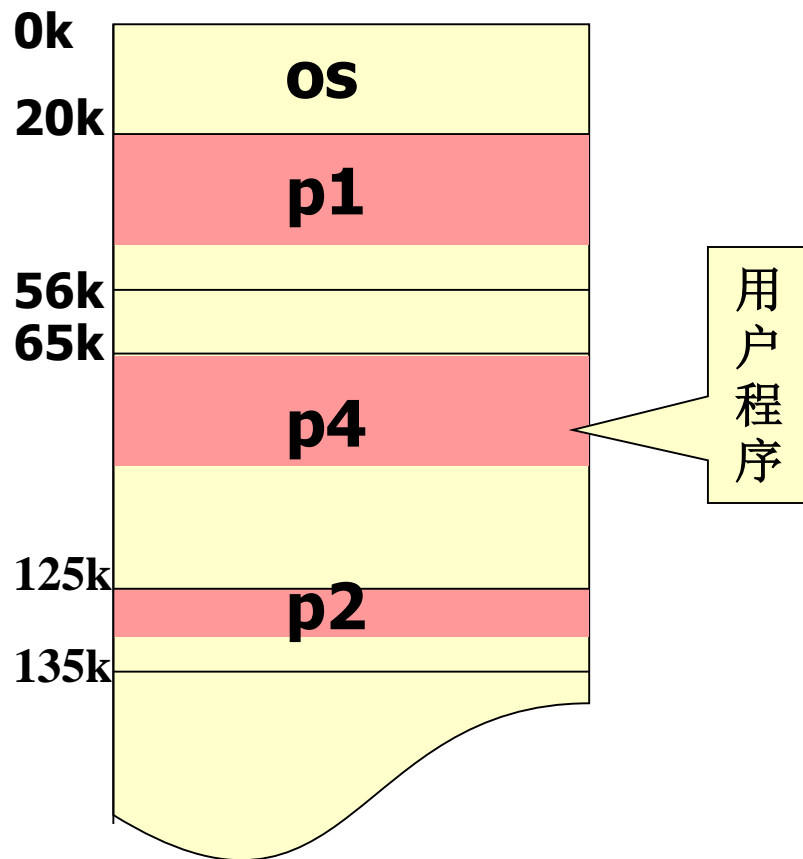
## 二、固定分区分配方式（固定分区存储管理）

是最早使用的一种可运行多道程序的存储管理方法。

### ❖ 存储管理方法

- **内存空间的划分**：将内存空间划分为若干个固定大小的分区，除OS占一分区外，其余的每一个分区装入一道程序。分区的大小可以相等，也可以不等，但事先必须确定，在运行时不能改变。即分区大小及边界在运行时不能改变。
- 系统需**建立一张分区说明表或使用表**，以记录分区号、分区大小、分区的起始地址及状态（已分配或未分配）。

# 固定分区分配方式示意图



区号	大小	起址	状态
<b>1</b>	<b>36k</b>	<b>20k</b>	已分配
<b>2</b>	<b>9k</b>	<b>56k</b>	未分配
<b>3</b>	<b>60k</b>	<b>65k</b>	已分配
<b>4</b>	<b>10k</b>	<b>125k</b>	已分配

分区说明表



## ❖ 内存分配

- 当某个用户程序要装入内存时，由内存分配程序检索分区说明表，从表中找出一个满足要求的尚未分配的分区分配给该程序，同时修改说明表中相应分区的状态；若找不到大小足够的分区，则拒绝为该程序分配内存。
- 当程序执行完毕，释放占用的分区，管理程序将修改说明表中相应分区的状态为未分配，实现内存资源的回收。

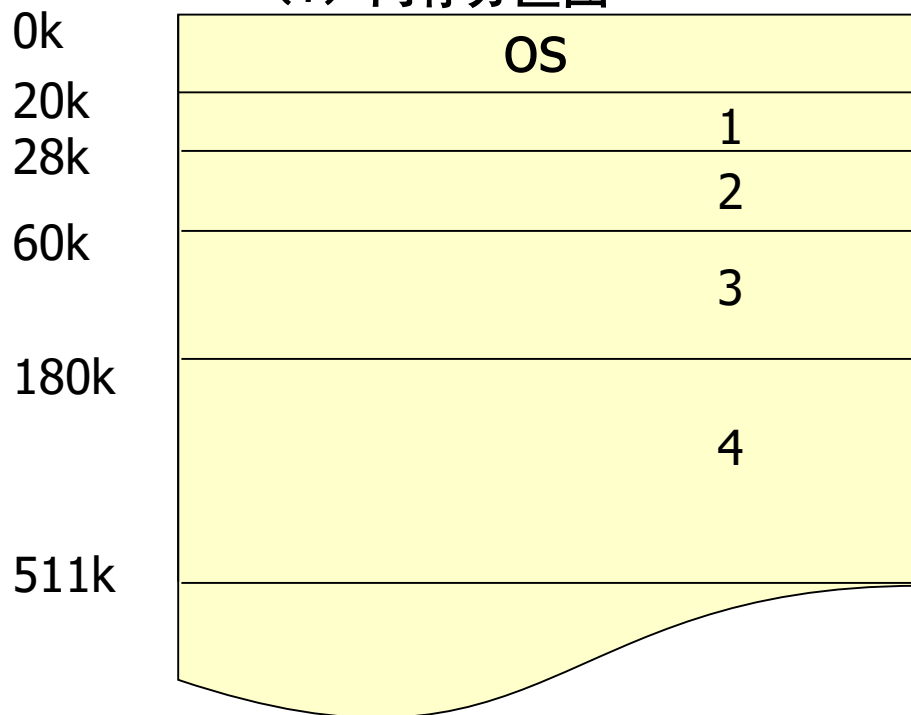
❖ **主要特点：**管理简单，但因作业的大小并不一定与某个分区大小相等，从而使一部分存储空间被浪费。所以主存的利用率不高。

❖ **例 题**



**例：**在某系统中，采用固定分区分配管理方式，内存分区（单位：字节）情况如图所示，现有大小为1KB、9KB、33KB、121KB的多个作业要求进入内存，试画出它们进入内存后的空间分配情况，并说明主存浪费多大？

(1) 内存分区图

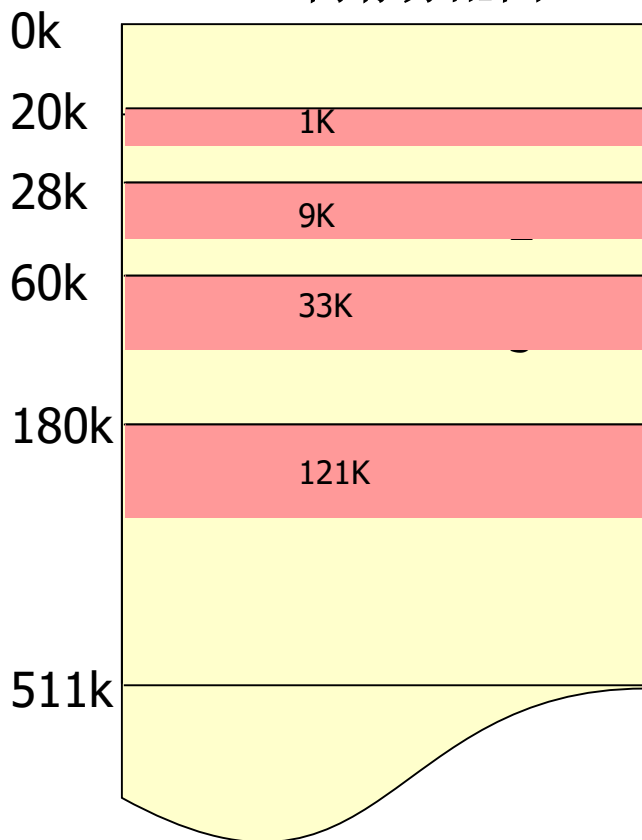


(2) 分区说明表

区号	大小	起址	状态
1	8k	20k	未分配
2	32k	28k	未分配
3	120k	60k	未分配
4	331k	180k	未分配

**解：**根据分区说明表，将4个分区依次分配给4个作业，同时修改分区说明表，其内存分配和分区说明表如下所示：

(1) 内存分配图



(2) 分区说明表

区号	大小	起址	状态
<b>1</b>	<b>8k</b>	<b>20k</b>	已分配
<b>2</b>	<b>32k</b>	<b>28k</b>	已分配
<b>3</b>	<b>120k</b>	<b>60k</b>	已分配
<b>4</b>	<b>331k</b>	<b>180k</b>	已分配

$$\begin{aligned} (3) \text{ 主存浪费空间} &= (8-1) + (32-9) + (120-33) + (331-121) \\ &= 7+23+87+210=327 (k) \end{aligned}$$





## 三、动态分区分配方式

动态分区分配又称为**可变式**分区分配，是一种动态划分存储器的分区方法。

### ❖ 存储管理方法

不事先将内存划分成一块块的分区，而是在作业进入内存时，根据作业的大小动态地建立分区，并使分区的大小正好适应作业的需要。因此系统中分区的大小是可变的，分区的数目也是可变的。

### ❖ 主要特点

管理简单，只需少量的软件和硬件支持，便于用户了解和使用。进程的大小与某个分区大小相等，从而主存的利用率有所提高。



# 1、分区分配中的数据结构（1）

## ❖ 空闲分区表

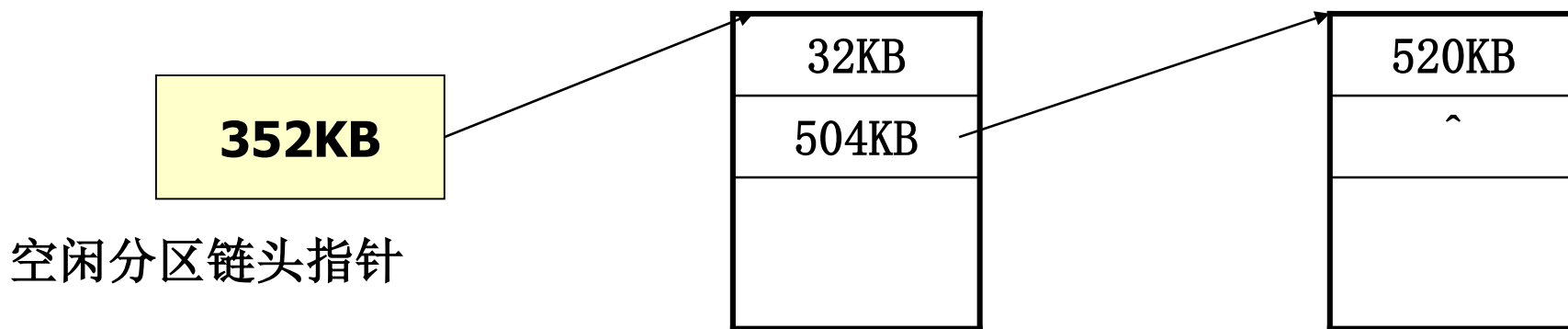
用来登记系统中的空闲分区（分区号、分区起始地址、分区大小及状态）。

分区号	大小KB	起始地址KB	状态
1	32	352	空闲
2	...	...	空表目
3	520	504	空闲
4	...	...	空表目
5	...	...	...

# 1、分区分配中的数据结构（2）

## ❖ 空闲分区链

用链头指针将系统中的空闲分区链接起来，构成空闲分区链。每个空闲分区的起始部分存放相应的控制信息（如大小，指向下一空闲分区的指针等）。





# 基于顺序搜索的动态分区分配算法

顺序搜索，是指依次搜索空闲分区链上的空闲分区，去寻找一个大小能满足要求的分区。常用的分配算法有以下四种：

- 首次适应算法 (First Fit)
- 循环首次适应算法 (Next Fit)
- 最佳适应算法 (Best Fit)
- 最坏适应算法 (Worst Fit)



# 首次适应算法（最先适应算法FF）

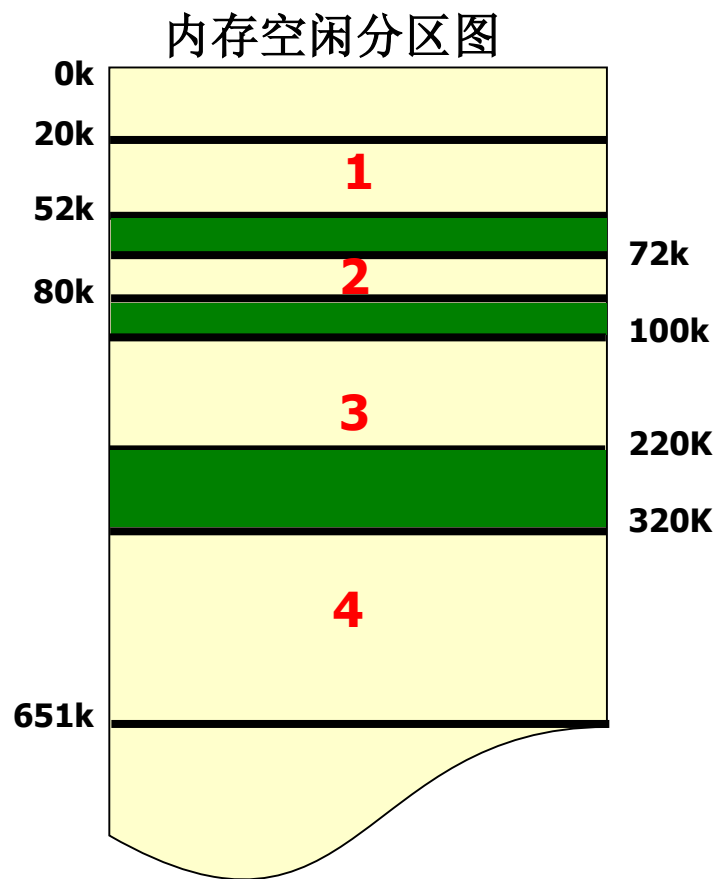
## ❖ 算法

- 空分区（链）按地址递增的次序排列。
- 在进行内存分配时，从空闲分区表/链首开始顺序查找，直到找到第一个满足其大小要求的空闲分区为止。
- 然后按照作业大小，从该分区中划出一块内存空间分配给请求者，余下的空闲分区仍按地址递增的次序保留在空闲分区表（链）中。

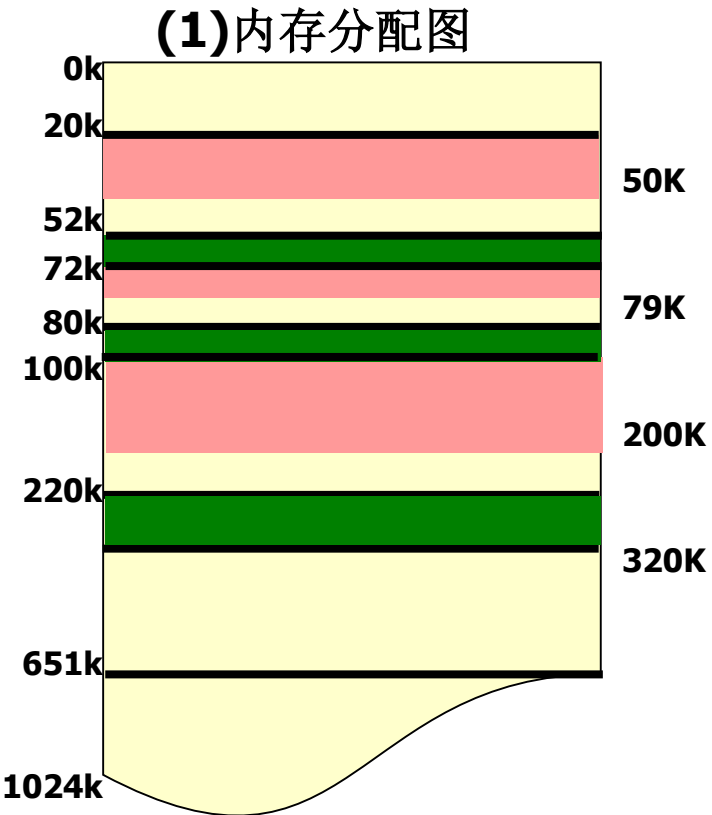
**例：**系统中的空闲分区表如下，现有三个作业申请分配内存空间100KB、30KB及7KB。给出按首次适应算法的内存分配情况及分配后空闲分区表。

空闲分区表

区号	大小	起址
1	32k	20k
2	8k	72k
3	120k	100k
4	331k	320k



**解：**按首次适应算法，  
 申请作业100k，分配3号分区，剩下分区为20k，起始地址200K ；  
 申请作业30k，分配1号分区，剩下分区为2k，起始地址50K ；  
 申请作业7k，分配2号分区，剩下分区为1k，起始地址79K ；  
 其内存分配图及分配后空闲分区表如下



**(2)该算法分配后的空闲分区表**

区号	大小	起址
1	2k	50k
2	1k	79k
3	20k	200k
4	331k	320k



## ❖ 首次适应算法的特点

优先利用内存低地址部分的空闲分区, 从而保留了高地址部分的大空闲区。

但由于低地址部分不断被划分, 致使低地址端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, 这增加了查找可用空闲分区的开销。





# 循环首次适应算法（NF）

## ❖ 算法要求

又称为下次适应算法，由首次适应算法演变而来。在为作业分配内存空间时，不再每次从空闲分区表/链首开始查找，而是从上次找到的空闲分区的下一个空闲分区开始查找，直到找到第一个能满足其大小要求的空闲分区为止。

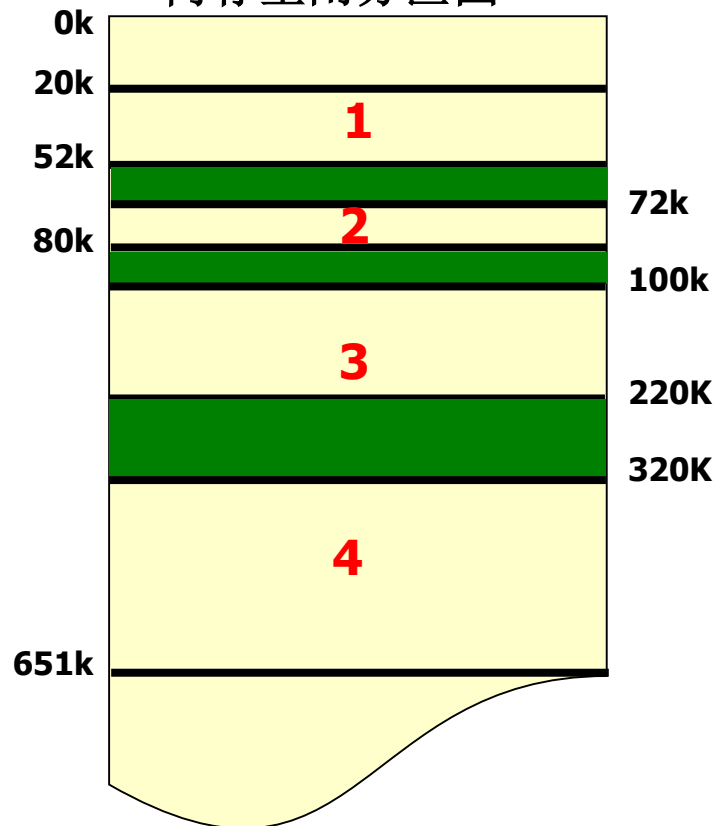
然后，再按照作业大小，从该分区中划出一块内存空间分配给请求者，余下的空闲分区仍按地址递增的次序保留在空闲分区表/链中。

**例：**系统中的空闲分区表如下，现有三个作业申请分配内存空间100KB、30KB及7KB。给出按循环首次适应算法的内存分配情况及分配后空闲分区表。

空闲分区表

区号	大小	起址
1	32k	20k
2	8k	72k
3	120k	100k
4	331k	320k

内存空闲分区图



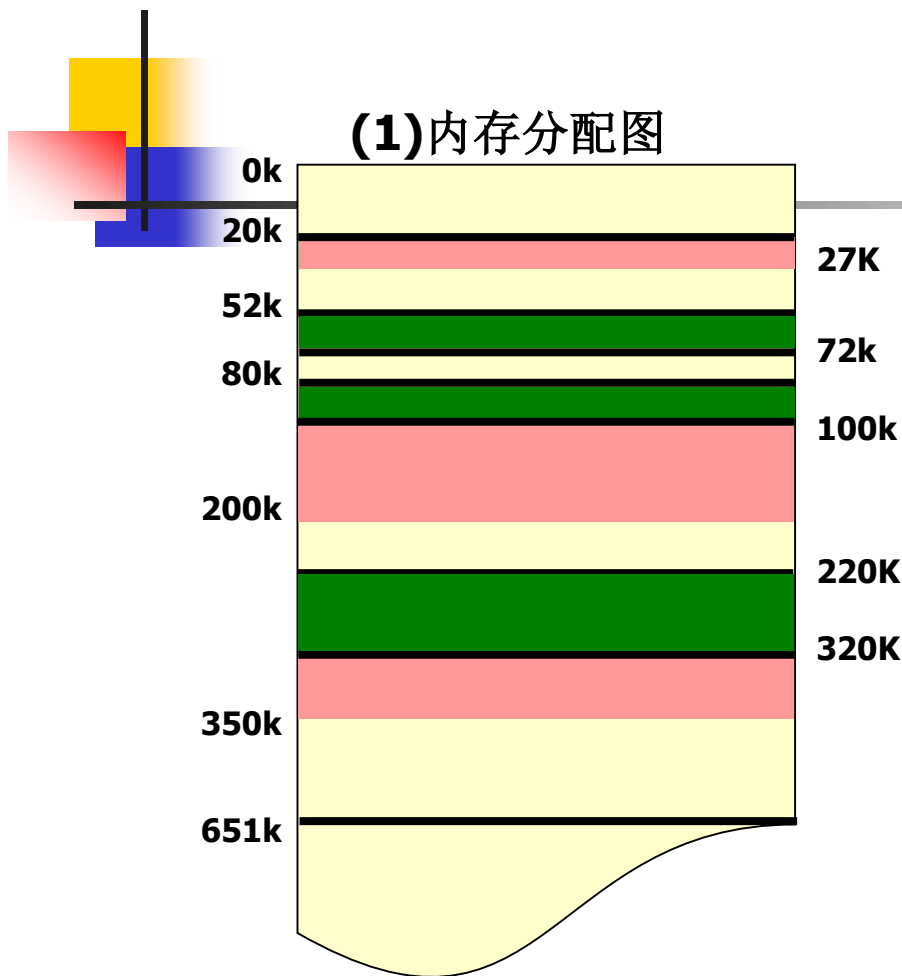
**解：**按循环首次适应算法，

申请作业100k，分配3号分区，剩下分区为20k，起始地址200K；

申请作业30k，分配4号分区，剩下分区为301k，起始地址350K；

申请作业7k，分配1号分区，剩下分区为25k，起始地址27K；

其内存分配图及分配后空闲分区表如下



**(2) 该算法分配后的空闲分区表**

区号	大小	起址
<b>1</b>	<b>25k</b>	<b>27k</b>
<b>2</b>	<b>8k</b>	<b>72k</b>
<b>3</b>	<b>20k</b>	<b>200k</b>
<b>4</b>	<b>301k</b>	<b>350k</b>

## ❖ 算法特点

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 (BF)

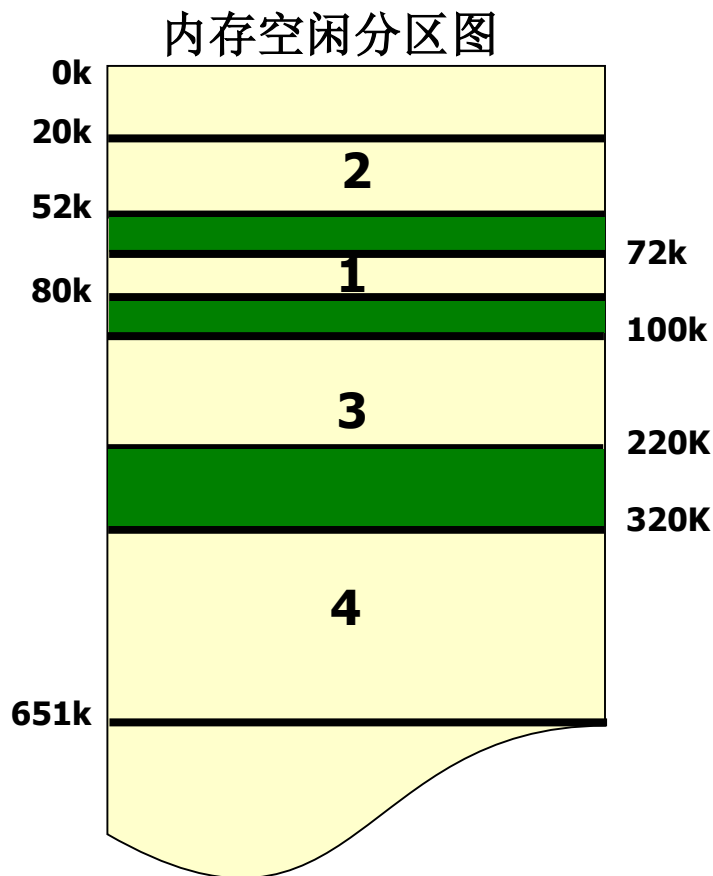
## ❖ 算法要求：

空闲分区表/链按容量大小递增的次序排列。

在进行内存分配时，从空闲分区表/链首开始顺序查找，直到找到第一个满足其大小要求的空闲分区为止。

按这种方式为作业分配内存，就能把既满足作业要求又与作业大小最接近的空闲分区分配给作业。如果该空闲分区大于作业的大小，则与首次适应算法相同，将剩余空闲分区仍按容量大小递增的次序保留在空闲分区表/链中。

**例：**系统中的空闲分区表如下，现有三个作业申请分配内存空间100KB、30KB及7KB。给出按最佳适应算法的内存分配情况及分配后空闲分区表。



分配前的空闲分区表

区号	大小	起址
1	8k	72k
2	32k	20k
3	120k	100k
4	331k	320k

**解：**按最佳适应算法，分配前的空闲分区表如上表。

申请作业100k，分配3号分区，剩下分区为20k，起始地址200K；

申请作业30k，分配2号分区，剩下分区为2k，起始地址50K；

申请作业7k，分配1号分区，剩下分区为1k，起始地址79K；

其内存分配图及分配后空闲分区表如下

作业**100K**分配后的空闲分区表

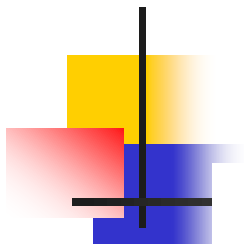
区号	大小	起址
<b>1</b>	<b>8k</b>	<b>72k</b>
<b>3</b>	<b>20k</b>	<b>200k</b>
<b>2</b>	<b>32k</b>	<b>20k</b>
<b>4</b>	<b>331k</b>	<b>320k</b>

作业**30K**分配后的空闲分区表

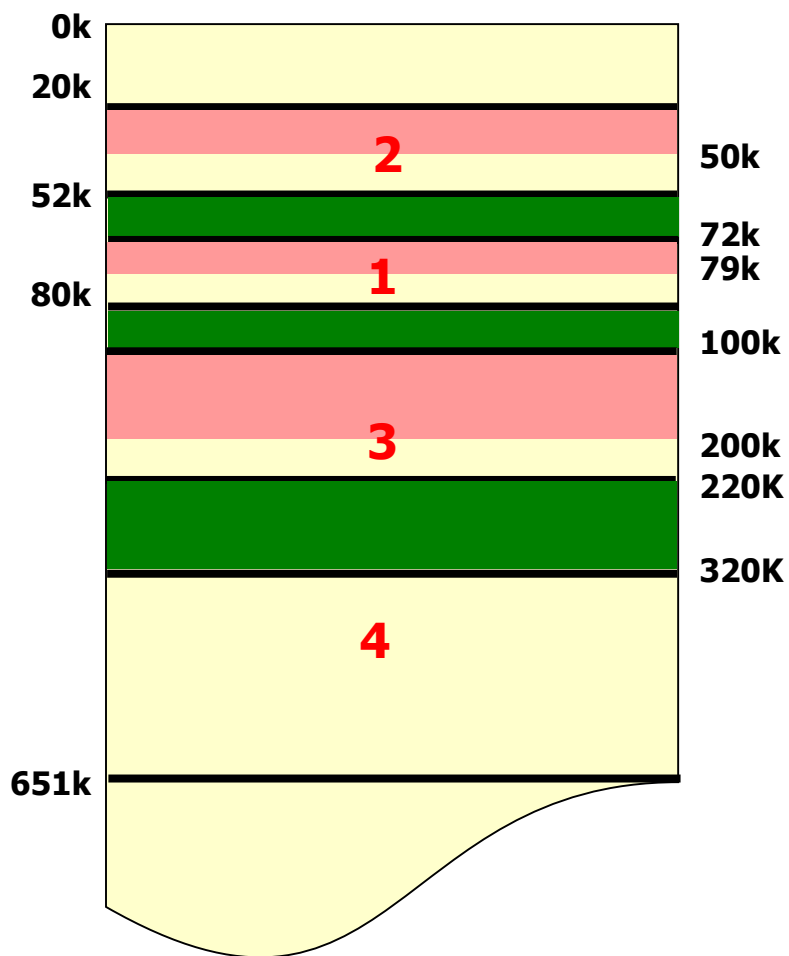
区号	大小	起址
<b>2</b>	<b>2k</b>	<b>50k</b>
<b>1</b>	<b>8k</b>	<b>72k</b>
<b>3</b>	<b>20k</b>	<b>200k</b>
<b>4</b>	<b>331k</b>	<b>320k</b>

作业**7K**分配后的空闲分区表

区号	大小	起址
<b>1</b>	<b>1k</b>	<b>79k</b>
<b>2</b>	<b>2k</b>	<b>50k</b>
<b>3</b>	<b>20k</b>	<b>200k</b>
<b>4</b>	<b>331k</b>	<b>320k</b>



(1) 内存分配示意图



(2) 该算法分配后的空闲分区表

区号	大小	起址
<b>1</b>	<b>1k</b>	<b>79k</b>
<b>2</b>	<b>2k</b>	<b>50k</b>
<b>3</b>	<b>20k</b>	<b>200k</b>
<b>4</b>	<b>331k</b>	<b>320k</b>



## ❖ 算法特点

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，但空闲区一般不可能正好和它申请的内存空间大小一样，因而将其分割成两部分时，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（外碎片或外零头）。





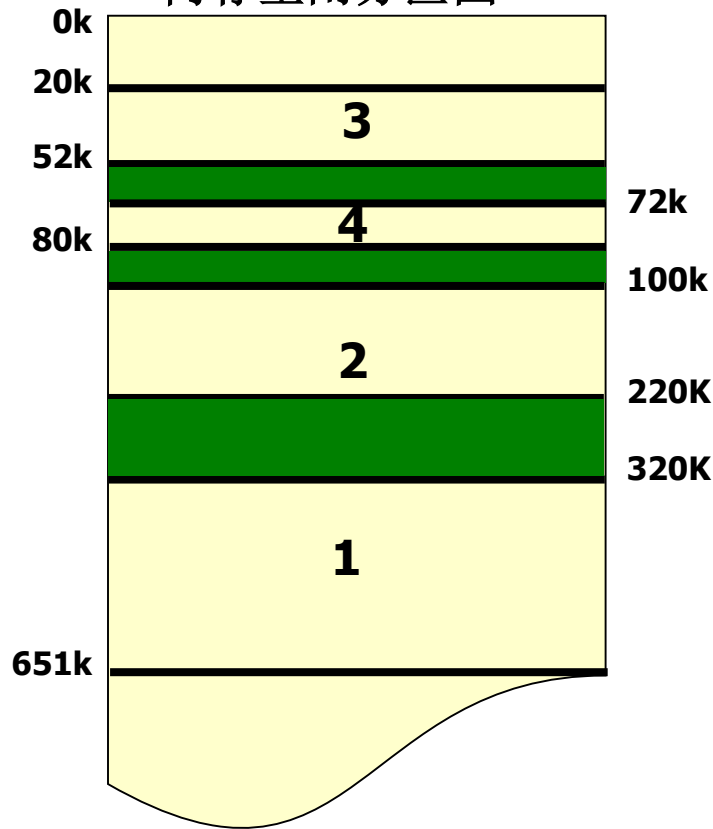
# 最坏适应算法 (WF)

## ❖ 算法要求

空闲分区表/链按容量大小递减的次序排列。在进行内存分配时，从空闲分区表/链首开始顺序查找，找到的第一个能满足作业要求的空闲分区，一定是个最大的空闲区。这样可保证每次分割后的剩下的空闲分区不至于太小（还可被分配使用，以减少“外碎片”），仍把它按从大到小的次序保留在空闲分区表/链中。

**例：**系统中的空闲分区表如下，现有三个作业申请分配内存空间100KB、30KB及7KB。给出按最坏适应算法的内存分配情况及分配后空闲分区表。

内存空闲分区图



空闲分区表

区号	大小	起址
1	331k	320k
2	120k	100k
3	32k	20k
4	8k	72k

**解：**按最坏适应算法，分配前的空闲分区表如上表。

申请作业100k，分配1号分区，剩下分区为231k，起始地址420K；

申请作业30k，分配1号分区，剩下分区为201k，起始地址450K；

申请作业7k，分配1号分区，剩下分区为194k，起始地址457K；

其内存分配图及分配后空闲分区表如下

作业**100K**分配后的空闲分区表

区号	大小	起址
<b>1</b>	<b>231k</b>	<b>420k</b>
<b>2</b>	<b>120k</b>	<b>100k</b>
<b>3</b>	<b>32k</b>	<b>20k</b>
<b>4</b>	<b>8k</b>	<b>72k</b>

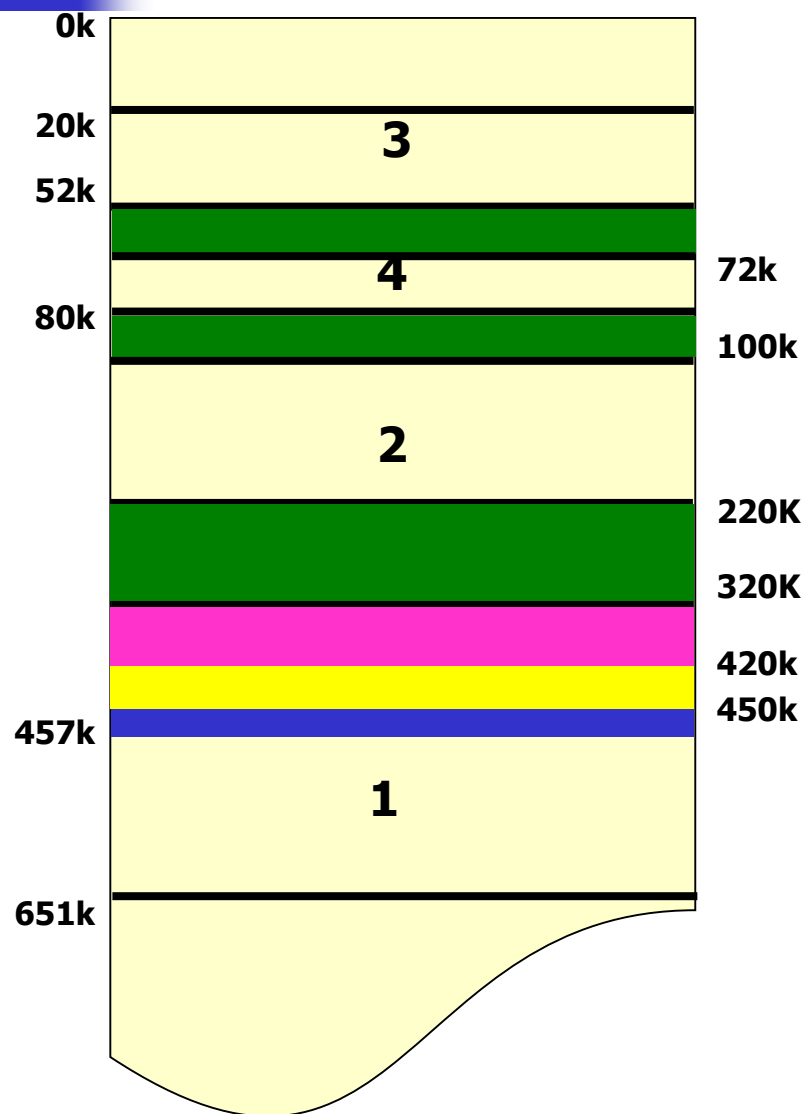
作业**30K**分配后的空闲分区表

区号	大小	起址
<b>1</b>	<b>201k</b>	<b>450k</b>
<b>2</b>	<b>120k</b>	<b>100k</b>
<b>3</b>	<b>32k</b>	<b>20k</b>
<b>4</b>	<b>8k</b>	<b>72k</b>

作业**7K**分配后的空闲分区表

区号	大小	起址
<b>1</b>	<b>194k</b>	<b>457k</b>
<b>2</b>	<b>120k</b>	<b>100k</b>
<b>3</b>	<b>32k</b>	<b>20k</b>
<b>4</b>	<b>8k</b>	<b>72k</b>

## (1)内存分配图



## (2) 该算法分配后的空闲分区表

区号	大小	起址
1	194k	457k
2	120k	100k
3	32k	20k
4	8k	72k



## ❖ 算法特点

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。

但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。



### 3、分区分配操作\_分配内存和回收内存

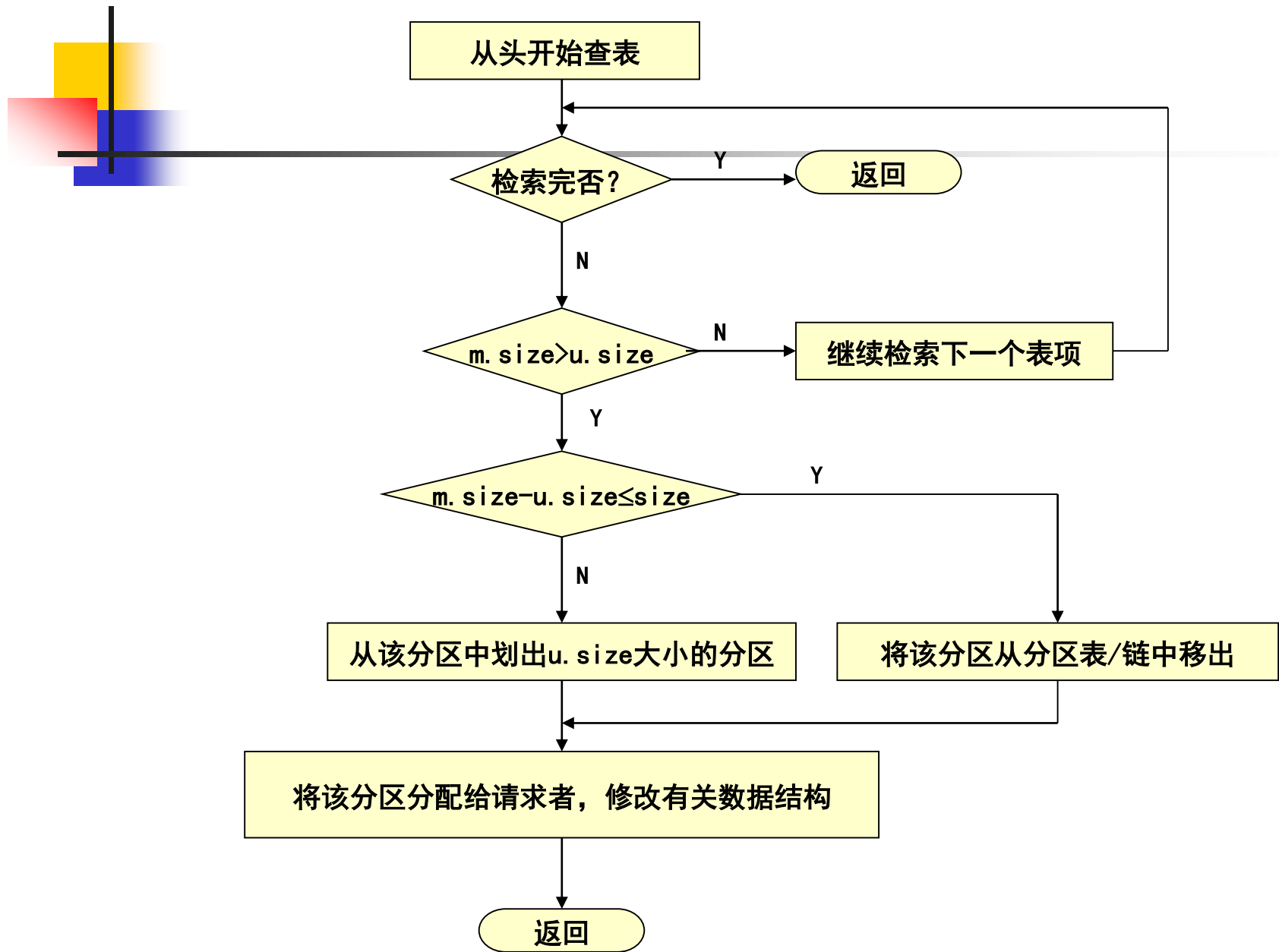
#### (1) 分配内存

系统利用某种分配算法，从空闲分区表/链中找到所需大小的分区。

#### 分区的切割：

设请求的分区大小为 $u.size$ ，空闲分区的大小为 $m.size$ ，若 $m.size - u.size \leq size$ （ $size$ 是事先规定的不再切割的剩余分区的大小），说明多余部分大小，可不再切割，将整个分区分配给请求者；否则，从该分区中按请求的大小划分出一块内存空间分配出去，余下的部分仍留在空闲分区表/链中，然后，将分配区的首址返回给调用者。

分配流程图如下



内存分配流程图



## (2) 回收内存

---

当作业执行结束时，释放所占有的内存空间，OS应回收已使用完毕的内存分区。

系统根据回收分区的大小及首地址，在空闲分区表中检查是否有邻接的空闲分区，如有，则合成为一个大的空闲分区，然后修改有关的分区状态信息。

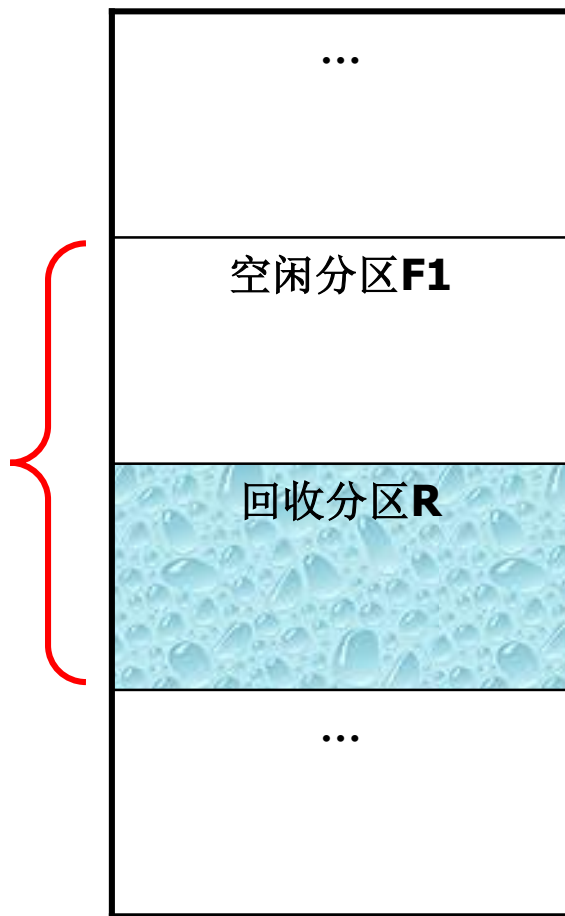
回收分区与已有空闲分区的相邻情况有以下四种：



## (2) 回收内存

- 1) 回收分区R上面邻接一个空闲分区F1，合并后首地址为空闲分区F1的首地址，大小为F1和R二者大小之和。

这种情况下，回收后空闲分区表中表项数不变。



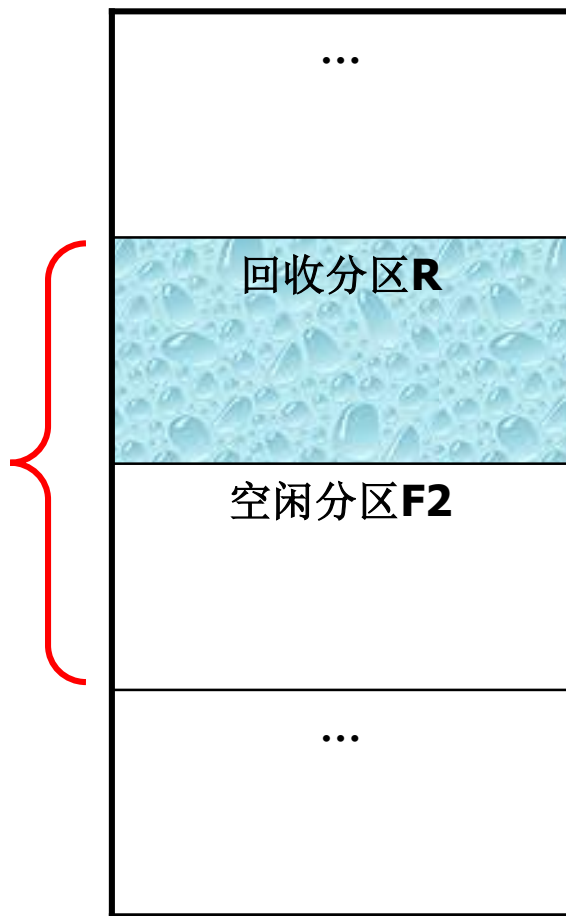
(a)

内存回收情况

## (2) 回收内存

2) 回收分区R下面邻接一个空闲分区F2，合并后首地址为回收分区R的首地址，大小为R和F2二者大小之和。

这种情况下，回收后空闲分区表中表项数不变。



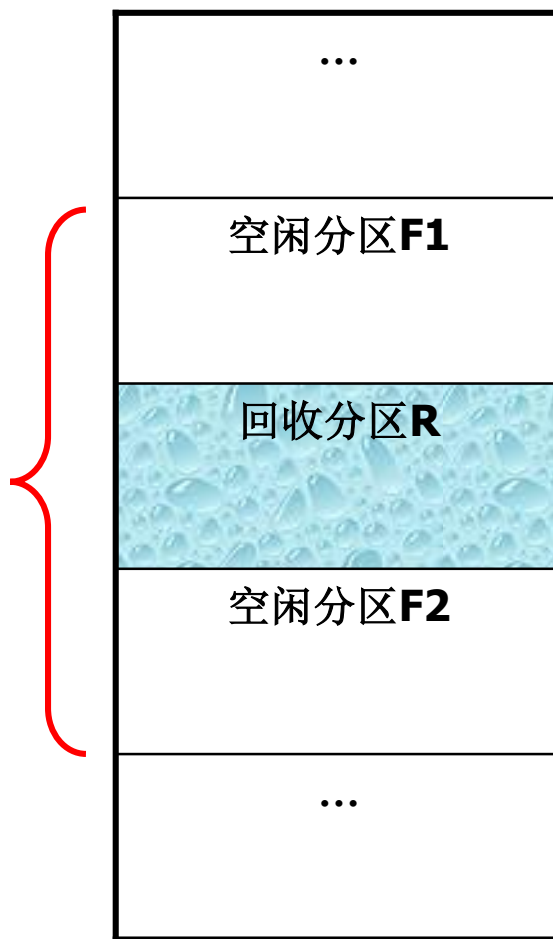
内存回收情况

(b)

## (2) 回收内存

3) 回收分区R上下邻接空闲分区F1和F2，合并后首地址为上空闲分区F1的首地址，大小为F1、R和F2三者大小之和。

这种情况下，回收后空闲分区表中表项数不但没有增加，反而减少一项。



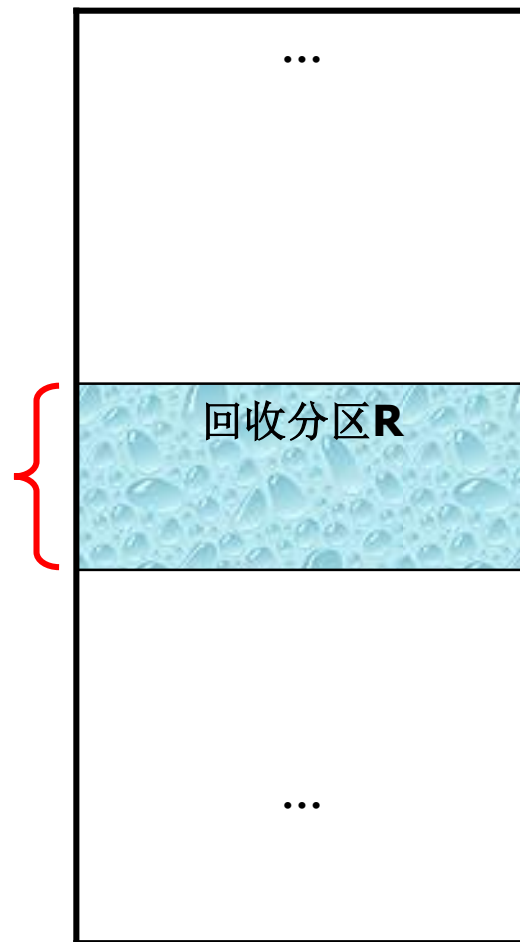
内存回收情况

(c)

## (2) 回收内存

4) 回收分区R不邻接空闲分区，这时在空闲分区表中新建一表项，并填写分区首地址、大小等信息。

这种情况下，回收后空闲分区表中表项数增加一项。



内存回收情况

(d)

## 四、动态可重定位分区分配方式

### 1、碎片问题

在分区存储管理方式中，必须把作业装入到一片连续的内存空间。如果系统中有若干个小分区，其总容量大于要装入的作业，但由于它们不相邻接，也将导致作业不能装入内存。

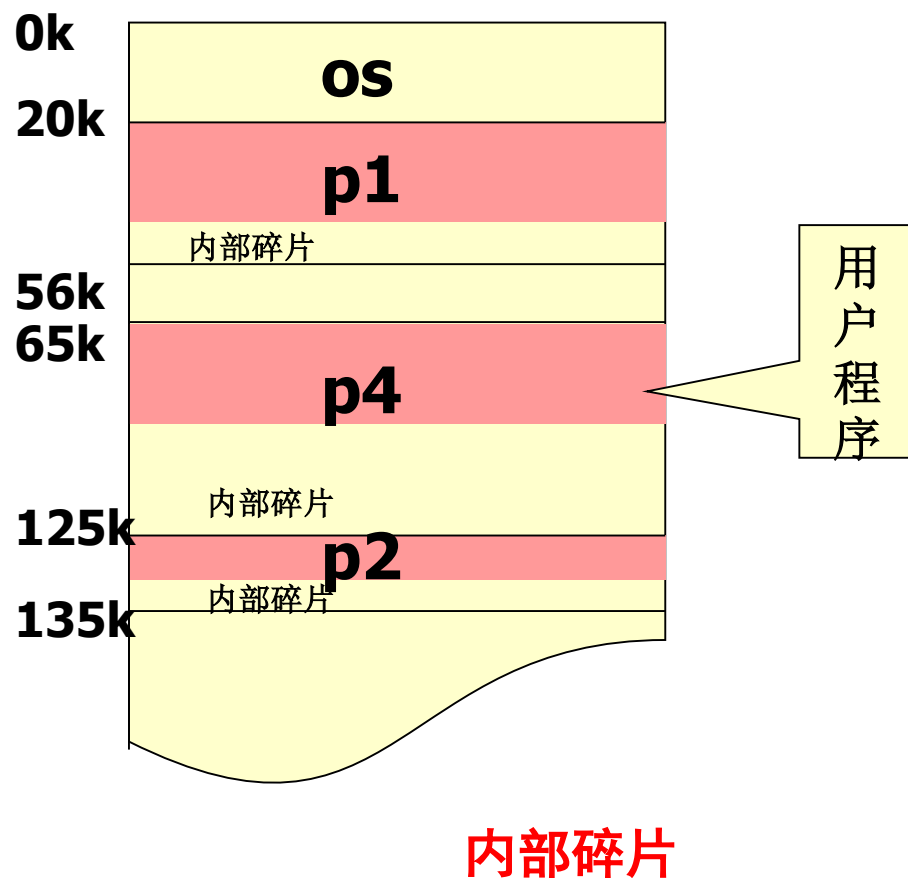
**例：**如图所示系统中有四个小空闲分区，不相邻，但总容量为90KB，如果现有一作业要求分配40KB的内存空间，由于系统中所有空闲分区的容量均小于40KB，故此作业无法装入内存。

操作系统
作业A
20KB
作业B
30KB
作业C
15KB
作业D
25KB

# 系统中的碎片（1）

❖ 这种内存中无法被利用的存储空间称为“零头”或“碎片”。根据碎片出现的情况分为以下两种：

■ **内部碎片**：指分配给作业的存储空间中未被利用的部分。如固定分区中存在的碎片。



## 系统中的碎片（2）

■ **外部碎片**：指系统中无法利用的小的空闲分区。如动态分区中存在的碎片。

操作系统	
作业A	
20KB	外部碎片
作业B	
30KB	外部碎片
作业C	
15KB	外部碎片
作业D	
25KB	外部碎片

**外部碎片**

## 2、碎片问题的解决方法

对系统中存在碎片，目前主要有两种技术（之一）：

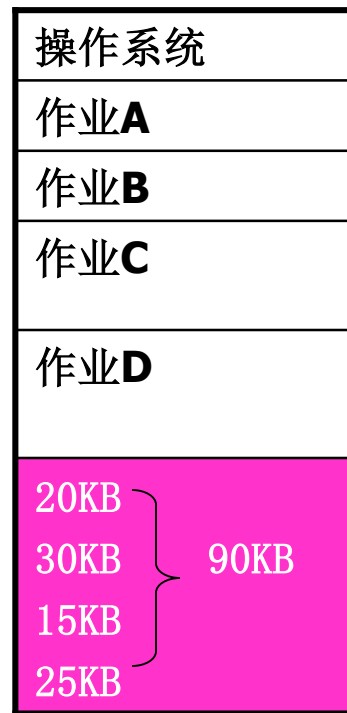
### ❖ 拼接或紧凑或紧缩技术

将内存中所有作业移到内存一端（作业在内存中的位置发生了变化，这就必须对其地址加以修改或变换即称为重定位），使本来分散的多个小空闲分区连成一个大的空闲区。如图所示。

这种通过移动作业从而把多个分散的小分区拼接成一个大分区的方法称为拼接或紧凑或紧缩。

拼接时机：分区回收时；

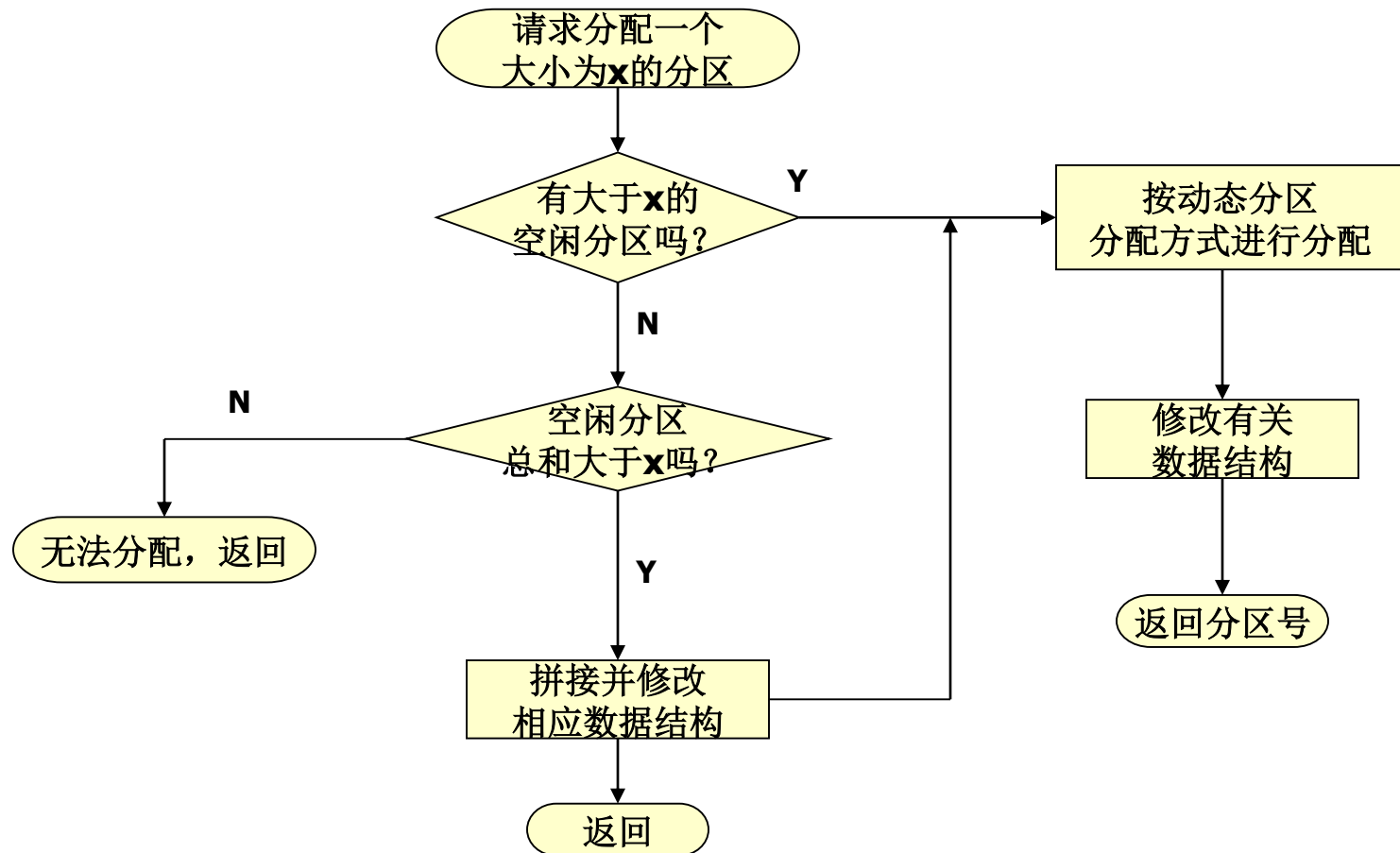
当找不到足够大的空闲分区且总空闲分区容量可以满足作业要求时。





## ❖ 动态重定位分区分配技术

在动态分区分配算法中增加拼接功能，在找不到足够大的空闲分区来满足作业要求，而系统中总空闲分区容量可以满足作业要求时，进行拼接。



动态重定位分区分配算法流程图



# 动态可重定位分区分配方式主要特点

---

可以充分利用存储区中的“零头/碎片”，提高主存的利用率。 但若 “零头/碎片” 太多，则拼接频率过高会使系统开销加大。



## 五、分区的存储保护（1）

---

存储保护是为了防止一个作业有意或无意地破坏操作系统或其它作业，常用的存储保护方法有：

### 1、界限寄存器方法

- **上下界寄存器方法**：用这两个寄存器分别存放作业的起始地址和结束地址。在作业运行过程中，将每一个访问内存的地址都同这两个寄存器的内容比较，如超出这个范围便产生保护性中断。



## 五、分区的存储保护（2）

存储保护是为了防止一个作业有意或无意地破坏操作系统或其它作业，常用的存储保护方法有：

### 1、界限寄存器方法

- **基址、限长寄存器方法**：用这两个寄存器分别存放作业的起始地址和作业的地址空间长度。当作业执行时，将每一访问内存的相对地址和限长寄存器比较，如果超过了限长寄存器的值，则发出越界中断信号，并停止作业的运行。



## 五、分区的存储保护（3）

### 2、存储保护键方法

给每个存储块（大小相同，一个分区为存储块的整数倍）分配一个单独的保护键，它相当于一把锁。

进入系统的每个作业也赋予一个保护键，它相当于一把钥匙。

当作业运行时，检查钥匙和锁是否匹配，如果不匹配，则系统发出保护性中断信号，停止作业运行。



## 六、对换(Swapping)

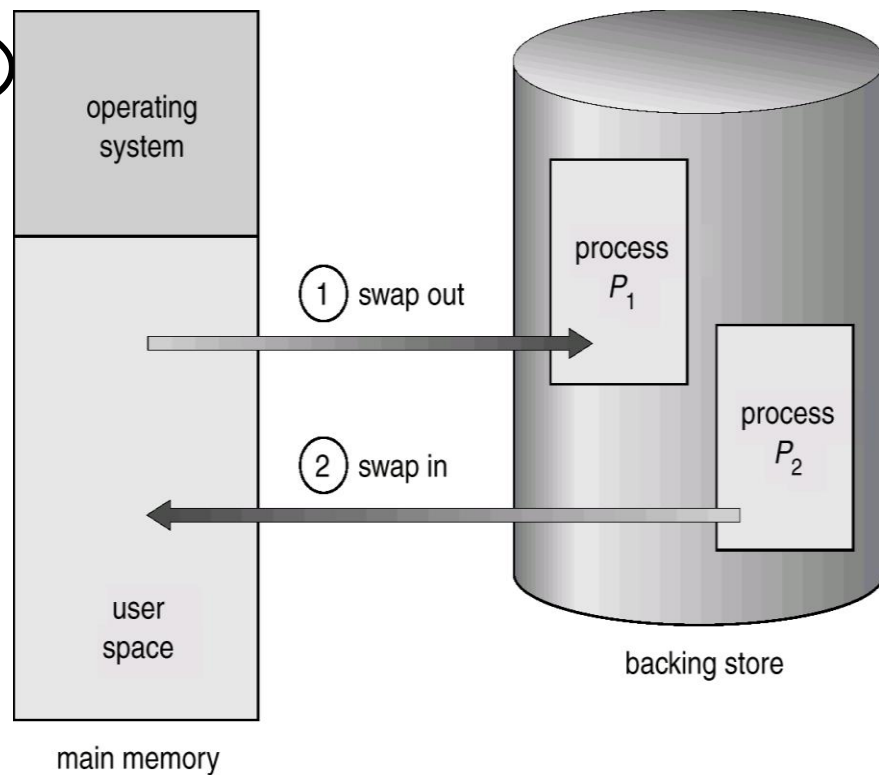
---

对换/交换技术是“扩充”内存容量和提高内存利用率的有效措施。现代OS中广泛采用。

最早用在MIT的兼容分时系统CTSS中，任何时刻系统中只有一个完整的用户作业，当运行一段时间后，因时间片用完或等待某事件发生，系统就把它交换到外存上，同时把另一作业调入内存让其运行，这样，可以在内存容量不大的小型机上分时运行，早期的一些分时系统多数采用这种交换技术。

## 六、对换(Swapping)

- ❖ **交换技术：**将暂时不用的某个进程及数据（首先是处于阻塞状态优先级最低的）部分（或全部）从内存移到外存（备份区或对换区，采用连续分配的动态存储管理方式）中去，让出内存空间，同时将某个需要的进程调入到内存，让其运行。交换到外存的进程需要时可以被再次交换回（选择换出时间最久的）内存中继续执行。





## 六、对换(Swapping)

---

### ❖ 整体对换（进程对换）

对换是以整个进程为单位的，广泛应用于分时系统中。

### ❖ 部分对换（页面对换或分段对换）

对换是以“页”或“段”为单位进行的，这种对换方法是实现后面要讲到的请求分页和请求分段式存储管理的基础，其目的是为了支持虚拟存储系统。





## 六、对换(Swapping)

为了实现进程对换，系统必须能实现三方面的功能：对换空间的管理、进程的换出、进程的换入。

### ❖ 对换空间的管理

- 把外存分为文件区和对换区。
- 文件区用于存放文件，由于通常的文件都是较长久地驻留在外存上，对文件区管理的主要目标，是提高存储空间的利用率，因此，对文件区采取离散分配方式。
- 对换区用于存放从内存换出的进程，进程在对换区中驻留的时间是短暂的，对换操作又较频繁，故对对换区管理的主要目标，是提高进程换入和换出的速度。因此，采取的是连续分配方式。



## 六、对换(Swapping)

为了实现进程对换，系统必须能实现三方面的功能：对换空间的管理、进程的换出、进程的换入。

### ❖ 进程的换出

- 每当一进程由于创建子进程而需要更多的内存空间，但又无足够的内存空间时，系统应将某进程换出。
- 系统首先选择处于阻塞状态且优先级最低的进程作为换出进程，然后启动磁盘，将该进程的程序和数据传送到磁盘的对换区上。若传送过程未出现错误，便可回收该进程所占用的内存空间，并对该进程的进程控制块做相应的修改。



## 六、对换(Swapping)

为了实现进程对换，系统必须能实现三方面的功能：  
对换空间的管理、进程的换出、进程的换入。

### ❖ 进程的换入

- 系统应定时地查看所有进程的状态，从中找出“就绪”状态但已换出的进程，将其中换出时间最久(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。



## 4.4 基本分页存储管理方式

### □ 连续分配存储管理方式产生的问题

在分区存储管理中，要求把进程放在一个**连续的存储区**中，因而会**产生许多碎片**。

### □ 碎片问题的解决方法

(1) **拼接/紧凑技术**——代价较高。

(2) **离散分配方式**——允许将作业/进程**离散**放到多个**不相邻接**的分区中，就可以避免拼接。基于这一思想产生了以下的离散分配方式：

- ❖ **分页式存储管理**：离散分配的基本单位是页
- ❖ **分段式存储管理**：离散分配的基本单位是段
- ❖ **段页式存储管理**：离散分配的基本单位是页



## 4.4 基本分页存储管理方式

---

在分页存储管理方式中，如不具备页面对换功能，不支持虚拟存储器功能，在调度作业运行时，必须将它的所有页面一次调入内存，若内存没有足够的块，则作业等待，这种存储管理方式称为纯分页或基本分页存储管理方式。

- 基本思想
- 页表
- 地址结构
- 地址变换机构
- 多级页表
- 页的共享与保护

## 一、基本思想 (1)

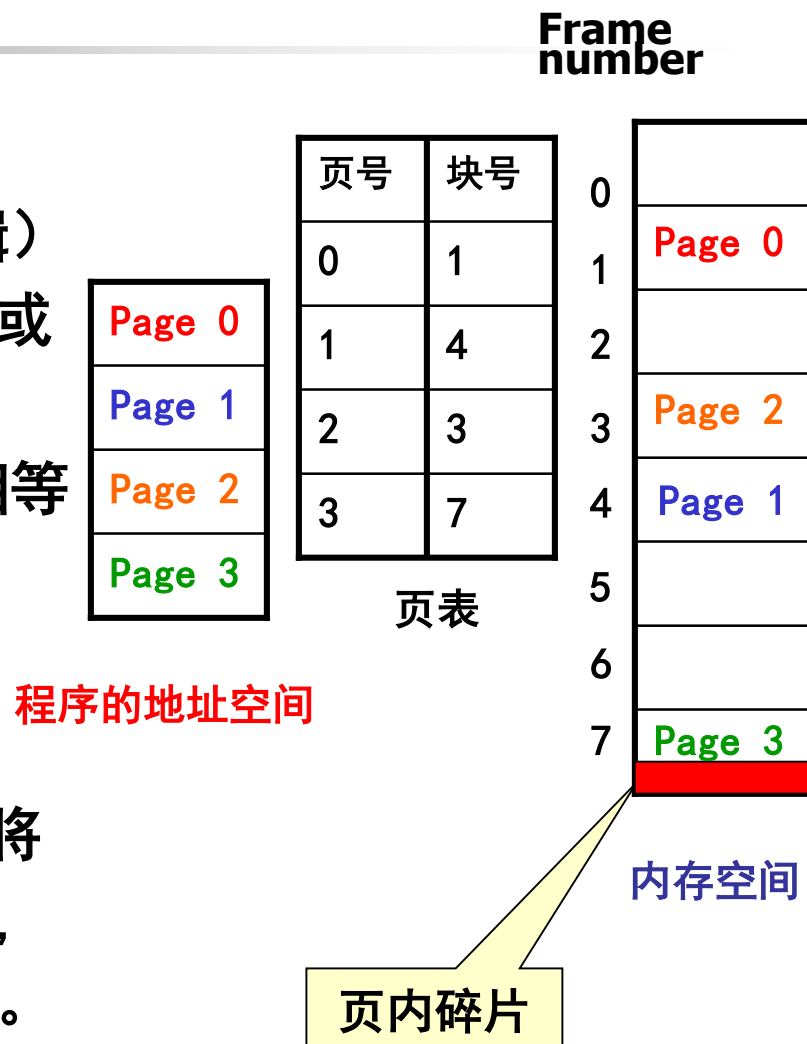
## ■ 空间划分

(1) 将一个用户进程的地址空间（逻辑）划分成若干个大小相等的区域，称为页或页面，并为各页从0开始编号。

(2) 内存空间也分成若干个与页大小相等的区域，称为（存储、物理）块或页框（frame），同样从0开始编号。

## ■ 内存分配

在为进程分配内存时,以块为单位,将进程中若干页装入到多个不相邻的块中,最后一页常装不满一块而出现**页内碎片**。



**注：需要CPU的硬件支持（地址变换机构）**

## 一、基本思想 (2)

页号	位移量 (页内地址)
----	------------

### ■ 页面大小——由地址结构决定

若页面较小：

- 减少页内碎片和内存碎片的总空间, 有利于提高内存利用率。
- 每个进程页面数增多, 从而使页表长度增加, 占用内存较大。
- 页面换进换出速度将降低。

若页面较大：

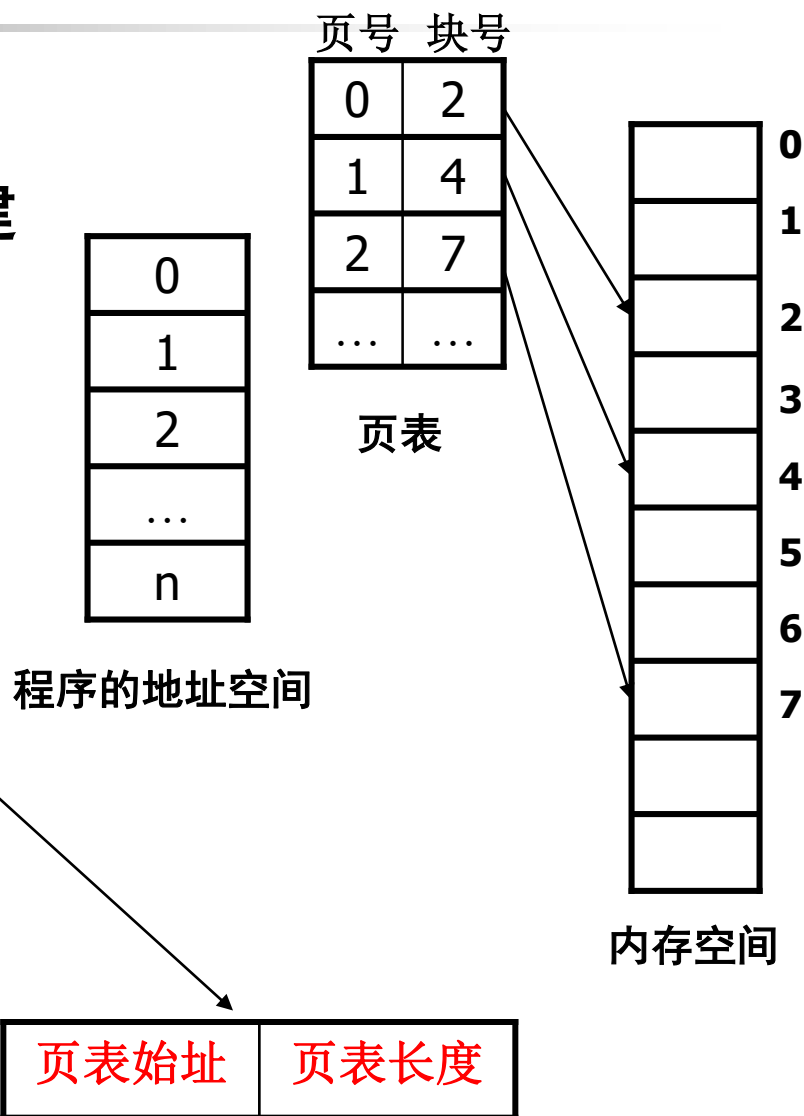
- 每个进程页面数减少, 页表长度减少, 占用内存就较小。
- 页面换进换出速度将提高。
- 会增加页内碎片, 不利于提高内存利用率。

页面大小——选择适中, 通常为2的幂, 一般在512B-8KB之间。

## 二、页表

为了便于在内存找到进程的每个页面所对应的块，系统为每个进程建立一张页面映象，简称页表，如图。

- ❑ 记录了页面在内存中对应的块号
- ❑ 页表一般存放在内存中
- ❑ 页表的基址及长度由页表寄存器给出
- ❑ 访问一个数据/指令需访问内存2次(页表一次, 内存一次)，所以出现内存访问速度降低的问题。





### 三、地址结构（1）

□ 分页存储管理系统中的地址结构（逻辑）：



地址长为32位，其中0~11位为页内地址，即每页的大小为  
 $2^{12}=4\text{KB}$

12~31位为页号，地址空间最多允许有 $2^{20}=1\text{M}$ 页。

若给定一个逻辑地址空间中的地址为A，页面大小为L，则页号P和页内地址w可按下式求得：

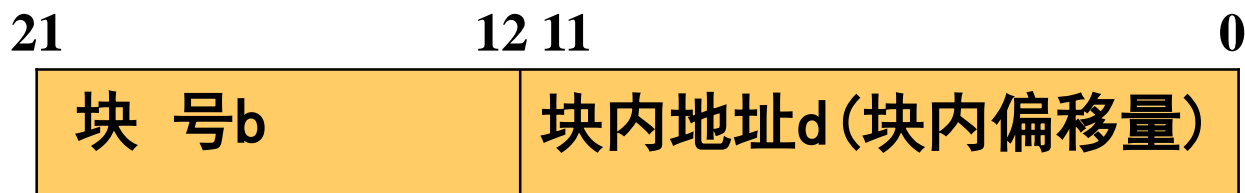
$$P = \text{INT}[A/L] \quad W = [A] \text{ MOD } L$$

其中，INT是整除函数，MOD是取余函数。

例：系统页面大小为 1 KB，设 $A=2170\text{B}$ ，则 $P=2$ ， $W=122$

## 三、地址结构（2）

□ 物理地址：



地址长为22位，其中0~11位为块内地址，即每块的大小为 $2^{12}=4\text{KB}$ ，与页相等；

12~21位为块号，内存地址空间最多允许有 $2^{10}=1\text{K}$ 块。

### 三、地址结构例题

设有一页式存储管理系统，向用户提供的逻辑地址空间最大为16页，每页2048B，内存总共有8个存储块，试问**逻辑地址**至少应为多少位？

**内存空间**有多大？

**解：**（1）页式存储管理系统的逻辑地址为：

页号p	页内位移量w
-----	--------

其中页内地址表每页的大小即  $2048\text{B}=2*1024\text{B}=2^{11}\text{B}$ ，所以页内地址为11位。

其中页号表最多允许的页数即  $16\text{页}=2^4\text{页}$ ，所以页号为4位。

故逻辑地址至少应为15位。

（2）物理地址为：

块号b	块内位移d
-----	-------

其中块内地址表每块的大小与页大小相等，所以块内地址也为11位。

其中块号表内存空间最多允许的块数即  $8\text{块}=2^3\text{块}$ ，所以块号为3位。

故内存空间至少应为14位，即  $2^{14}=16\text{KB}$



## 四、地址变换机构（1）

为了能将用户地址空间中的逻辑地址变换为内存空间中的物理地址，在系统中必须设置地址变换机构。分为基本的地址变换机构和具有快表的地址变换机构。

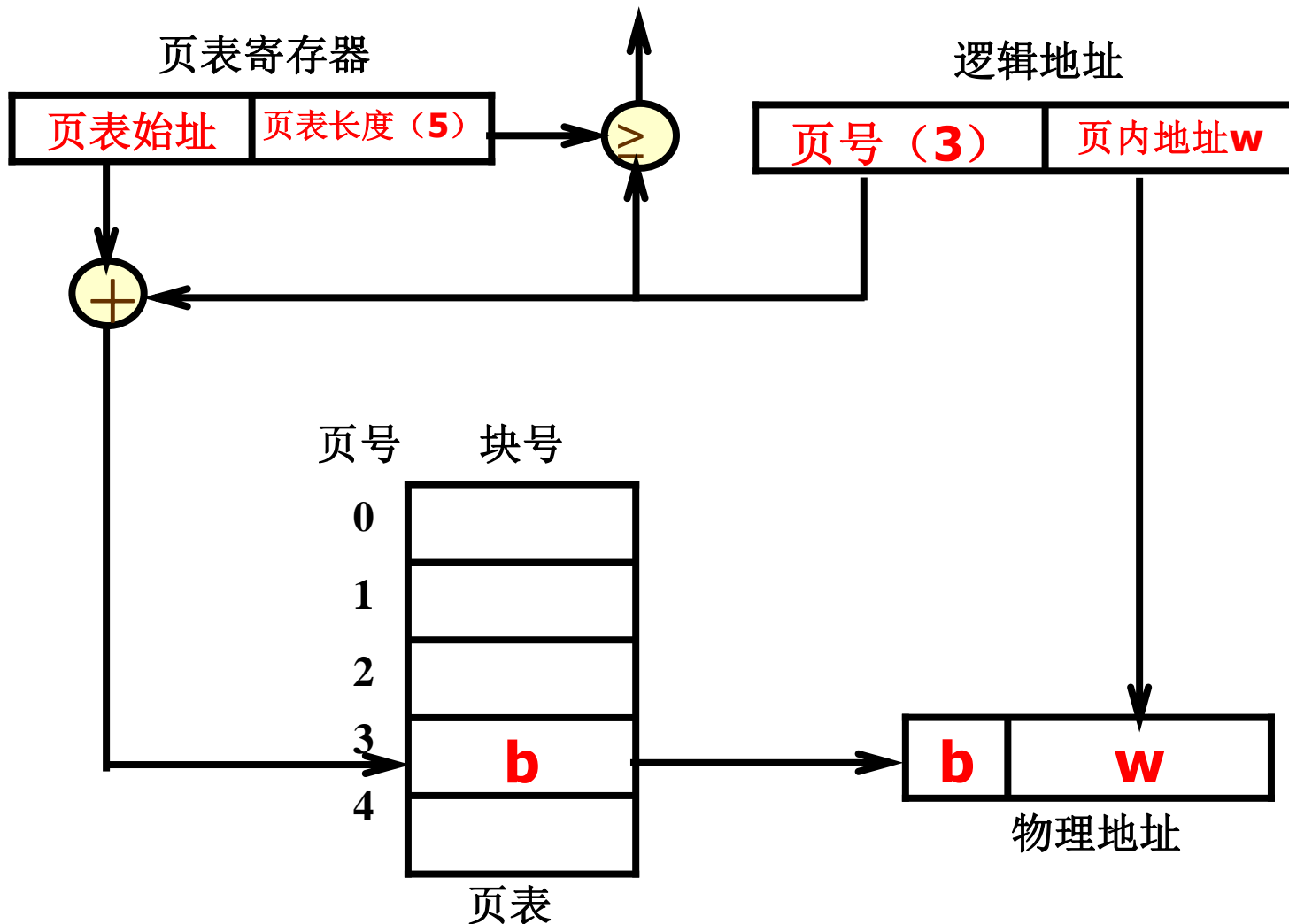
### □ 地址变换机构的基本任务

——实现逻辑地址向物理地址的转换（页号→块号）。

地址变换借助页表来完成。

## 四、地址变换机构（2）

□ 分页系统的基本地址变换机构如图所示：



# 地址变换例题

例1：若在一分页存储管理系统中，某作业的页表如表所示，已知页面大小为1024B，试将逻辑地址（十进制）1011，2148，5012转化为相应的物理地址，画出其地址转换图。

解：由题知逻辑地址为：

页号p(2位)

位移量w(10位)

物理地址为：

块号b(3位)

块内位移d(10位)

页号	块号
0	2
1	3
2	1
3	6

(1) 逻辑地址1011的二进制表示为 00 1111110011

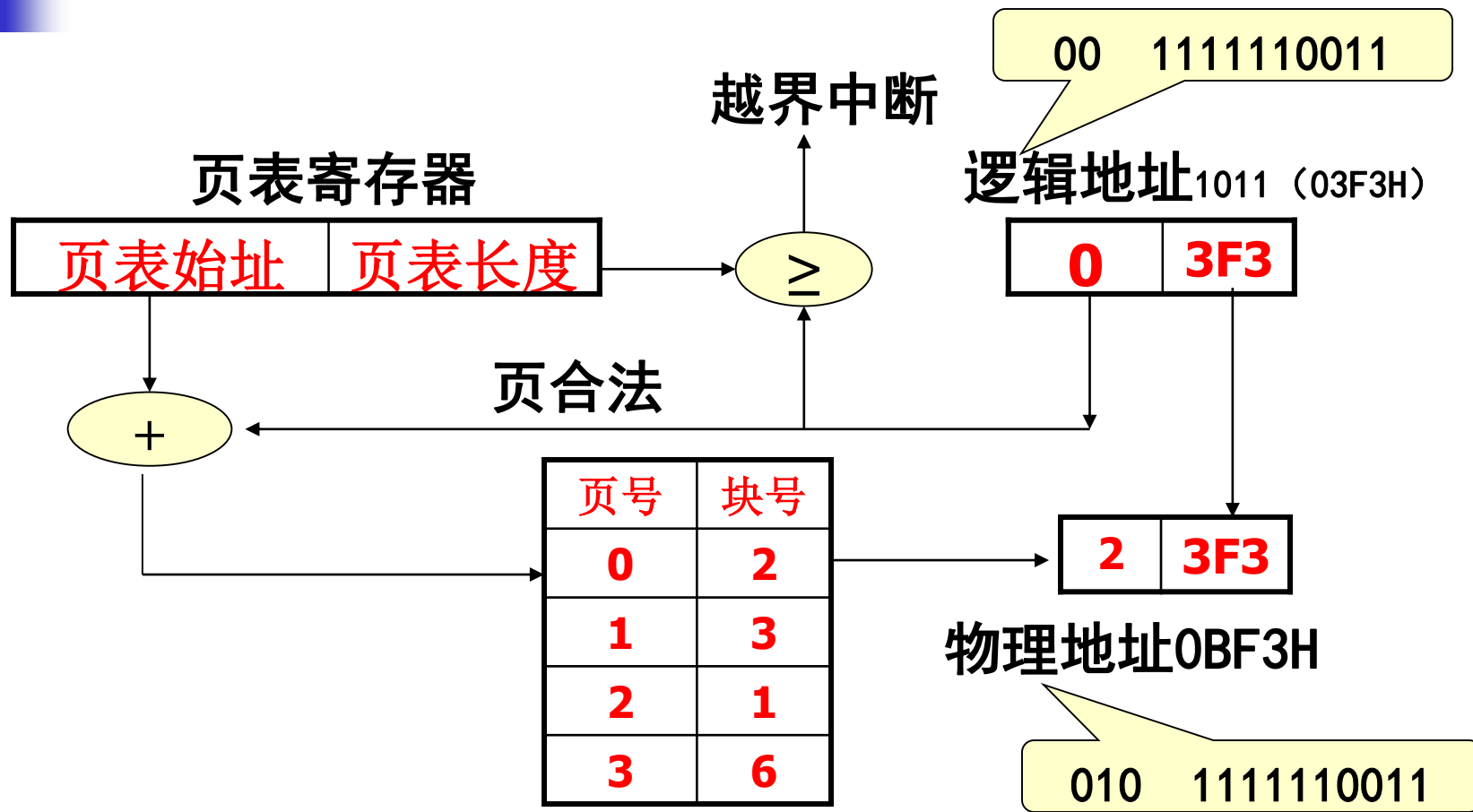
由此可知逻辑地址1011的页号0，查页表知该页放在第2物理块中，其物理地址的二进制表示为 010 1111110011

所以逻辑地址1011对应的物理地址为0BF3H. 其地址转换图如下页所示。

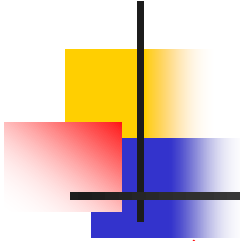
(2) 略

(3) 逻辑地址5012的二进制表示为：100 1110010100

可知该逻辑地址的页号为4，查页表知该页为不合法页，则产生越界中断。



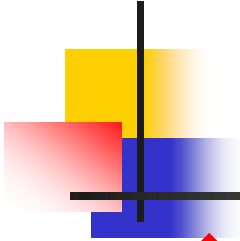
地址变换过程



❖例2 在一个页式存储管理系统中，页表内容如下所示，若页的大小为4KB，则地址转换机构将逻辑地址0转换成的物理地址为多少？

页号	块号
0	2
1	1
2	6
3	3
4	7

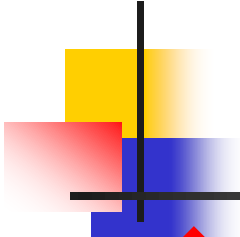




❖ 例3 某虚拟存储器的用户编程空间共32个页面，每页为1KB，内存为16KB。假定某时刻一用户程序中已调入内存的页面对应的物理块号如下表：

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：



❖例4 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4096字节，现有一逻辑地址为2F6AH，且第0、1、2页依次放在物理块10、12、14号中，问相应的物理地址为多少？

解答：

因逻辑地址长度为16位，页面大小4096字节，所以，前面的4位表示页号。

2F6AH的二进制表示：0010 1111 0110 1010

可知页号为2，根据已知条件：该页放在14号物理块中。

物理地址的十六进制表示为：EF6AH



## 四、地址变换机构

-----具有快表的地址变换机构(1)

### ❖ 基本的地址变换机构存在的问题

地址变换速度降低（因页表放于内存中，CPU访问一个数据需两次访问内存：一次访页表，以确定所取数据或指令的物理地址；另一次是根据物理地址取数据或指令。）

### ❖ 目的：

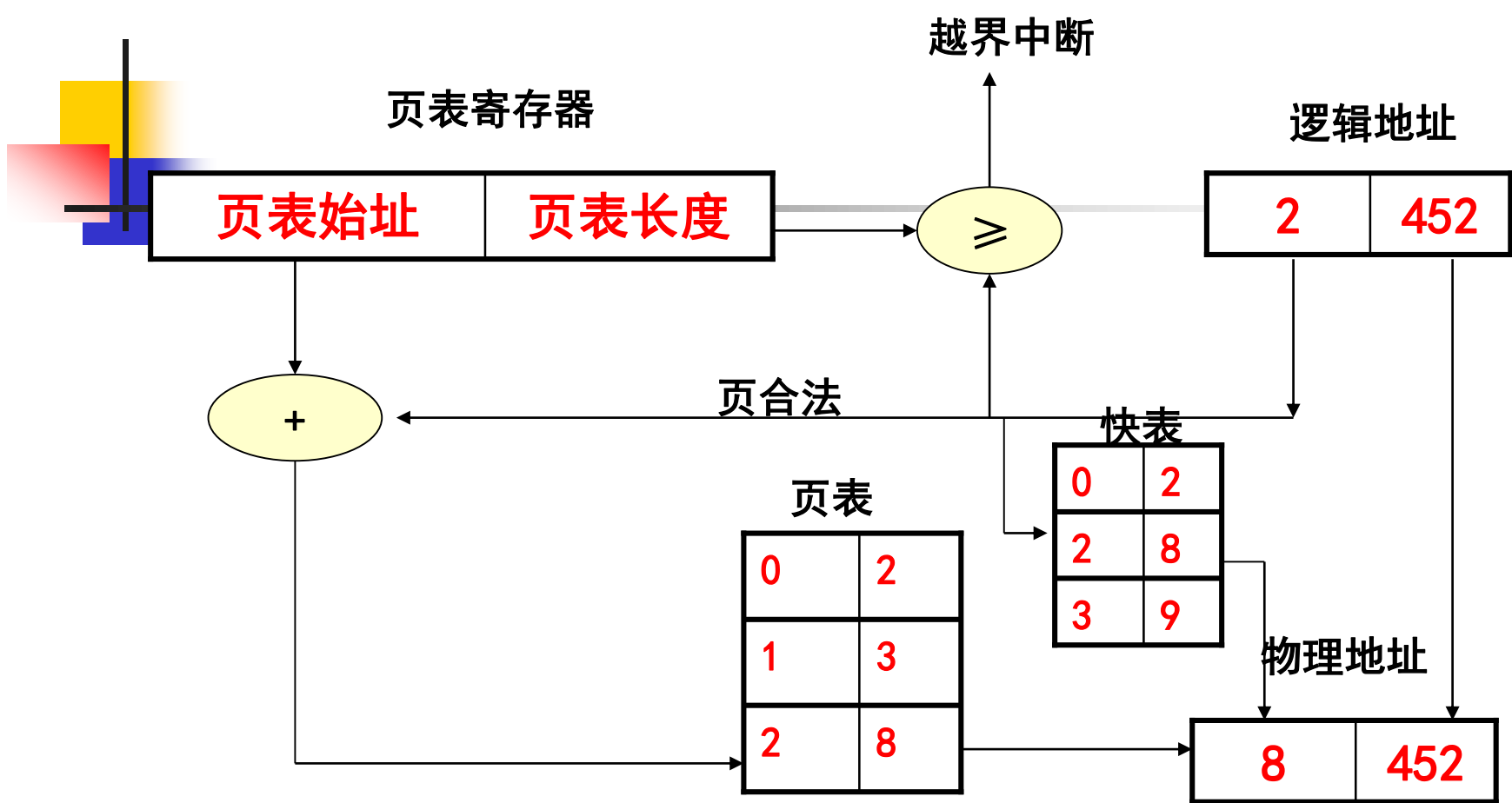
为了提高地址变换速度

## 四、地址变换机构

-----具有快表的地址变换机构(2)

### ❖ 快表（联想寄存器、联想存储器、TLB）

- ❑ 是一种特殊高速缓冲存储器。
- ❑ 内容--为页表中的一部分或全部
- ❑ CPU产生的逻辑地址的页首先在快表中寻找，若找到（命中），就找出其对应的物理块；若未找到（未命中），再到页表中找其对应的物理块，并将之复制到快表。
- ❑ 若快表中内容满，则按某种算法淘汰某些页。



## 具有快表的地址变换机构

有效访问内存的时间

$$T = P_{TLB} * (T_{TLB} + T_M) + (1 - P_{TLB}) * (T_{TLB} + 2T_M)$$

其中,  $P_{TLB}$  为快表的命中率,  $T_{TLB}$  为快表的访问时间,  $T_M$  为内存的访问时间



## □有效内存访问时间T例题

**例：** 有一页式系统，其页表存放在主存中。

(1) 如果对主存的一次存取需要100ns，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，对快表的一次存取需要20ns，若平均命中率为85%，试问此时的存取时间为多少？

解：(1) 页表放主存中，则实现一次页面访问需2次访问主存，一次是访问页表，确定所存取页面的物理块，从而得到其物理地址，一次根据物理地址存取页面数据。所以实现一次页面访问的存取时间为： $100\text{ns} \times 2 = 200\text{ns}$

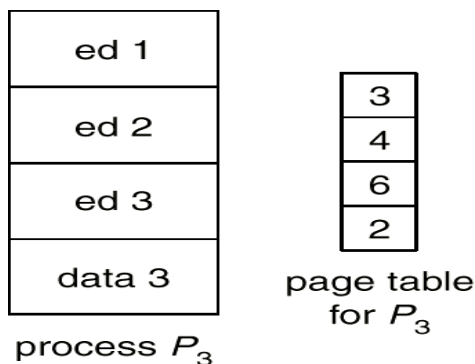
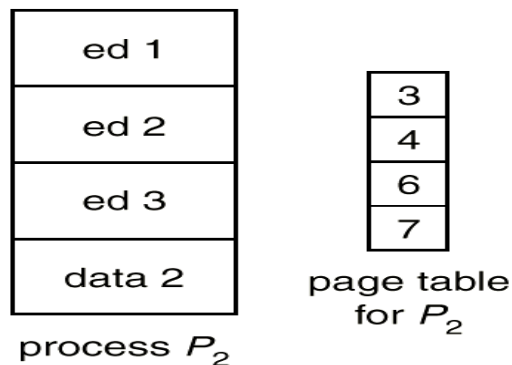
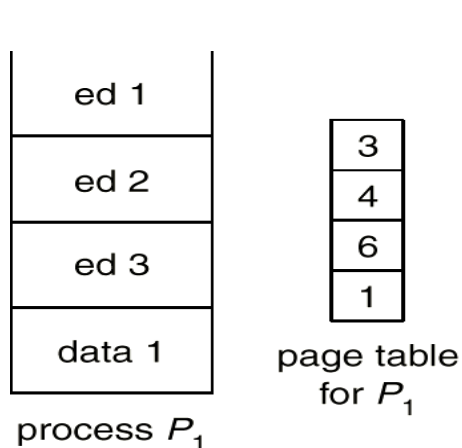
(2) 系统加有快表，则实现一次页面访问的存取时间为：

$$0.85 \times (20\text{ns} + 100\text{ns}) + (1 - 0.85) \times (20\text{ns} + 2 \times 100\text{ns}) = 135\text{ns}$$

## 四、页的共享与保护

### □ 共享代码（数据）的实现方法

由各进程共享的一段代码（数据），要求各进程相应的页存入内存相同物理块中。





## 四、页的共享与保护

### □ 带来的问题

若共享数据与不共享数据划在同一块中，则：

- ❖ 有些不共享的数据也被共享，不易保密
- ❖ 计算共享数据的页内位移较困难

实现数据共享的最好方法——段式存储管理。

### □ 页的保护

页式存储管理系统提供了两种方式：

- ❖ 地址越界保护
- ❖ 在页表中设置保护位

（定义操作权限：只读，读写，执行等）





## 4.5 基本分段存储管理方式

---

- ❖ 分段存储管理方式的引入
- ❖ 分段系统的基本原理
- ❖ 共享与保护
- ❖ 段页式存储管理方式



# 分段存储管理方式的引入-满足用户要求

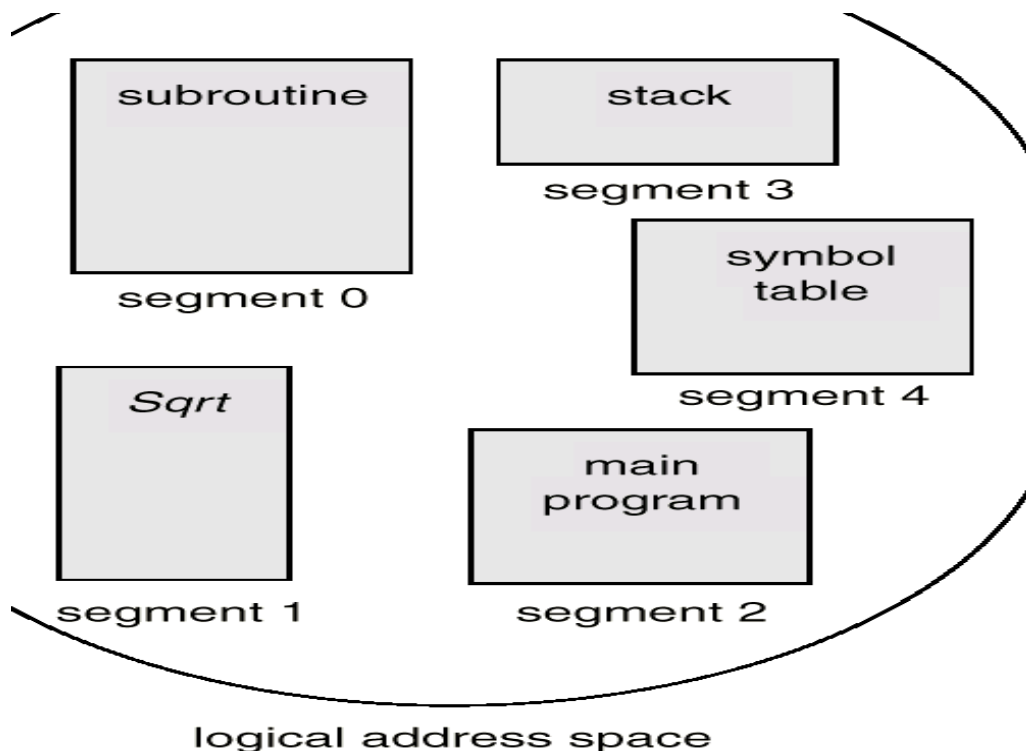
引入分段存储管理方式，主要是为了满足用户的一系列要求：

- ❖ **方便编程**：按逻辑关系分为若干个段，每个段从0编址，并有名字和长度，访问的逻辑地址由段名和段内偏移量决定。
- ❖ **信息共享**：共享是以信息为逻辑单位，页是存储信息的物理单位，段却是信息的逻辑单位。
- ❖ **信息保护**：保护也是对信息的逻辑单位进行保护的。
- ❖ **动态链接**：动态链接以段为单位。
- ❖ **动态增长**：实际应用中，某些段（数据段）会不断增长，前面的存储管理方法均难以实现。

# 分段系统的基本原理

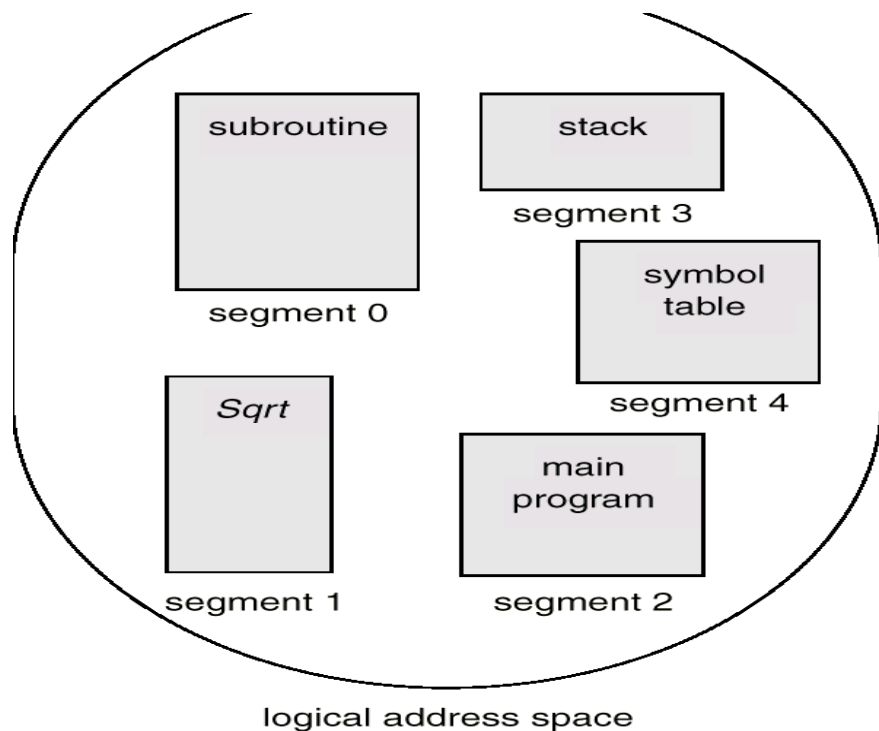
## 空间划分(分段)

将用户作业的逻辑地址空间划分成若干个大小不等的段（由用户根据逻辑信息的相对完整来划分）。各段有段名（常用段号代替），首地址为0



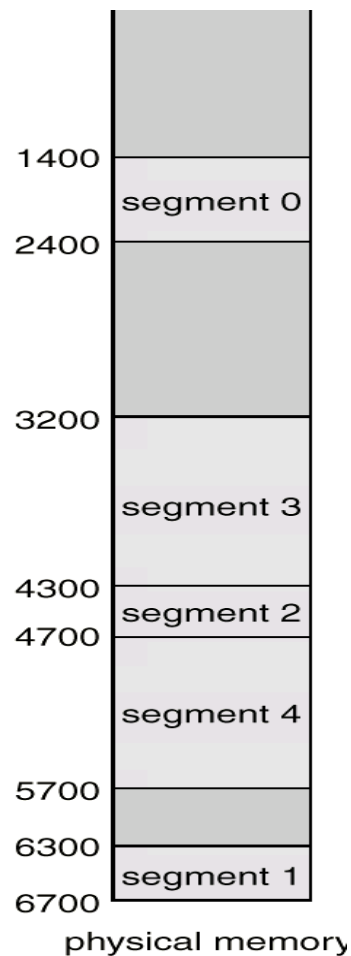
## ■ 内存分配

在为作业分配内存时，以段为单位，分配一段连续的物理地址空间；段间不必连续。



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



**注：**1、整个作业的逻辑地址空间是二维的，其逻辑地址由段号和段内地址组成；物理地址空间是一维的。

2、需要CPU的硬件支持（地址变换机构）

# 分段系统的基本原理——段表

段号	段长	基址
0	30k	40k
1	20k	80k
2	15k	120k
3	10k	150k

❑ 记录了段与内存位置的对应关系

❑ 段表常保存在内存中

❑ 段表的基址及长度由段表寄存器给出

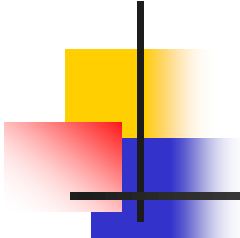
段表始址	段表长度
------	------

❑ 访问一个数据/指令需访问内存2次(段表一次, 内存一次), 所以也出现内存访问速度降低的问题。

❑ 二维的逻辑地址:

段号	段内地址
----	------

❑ 许多编译程序支持分段方式, 自动根据源程序的情况产生若干个段



例：采用段式存储管理的系统中，若地址用24位表示，其中8位表示段号，则允许段的最大长度是（ ），一个作业最多可有（ ）个段。

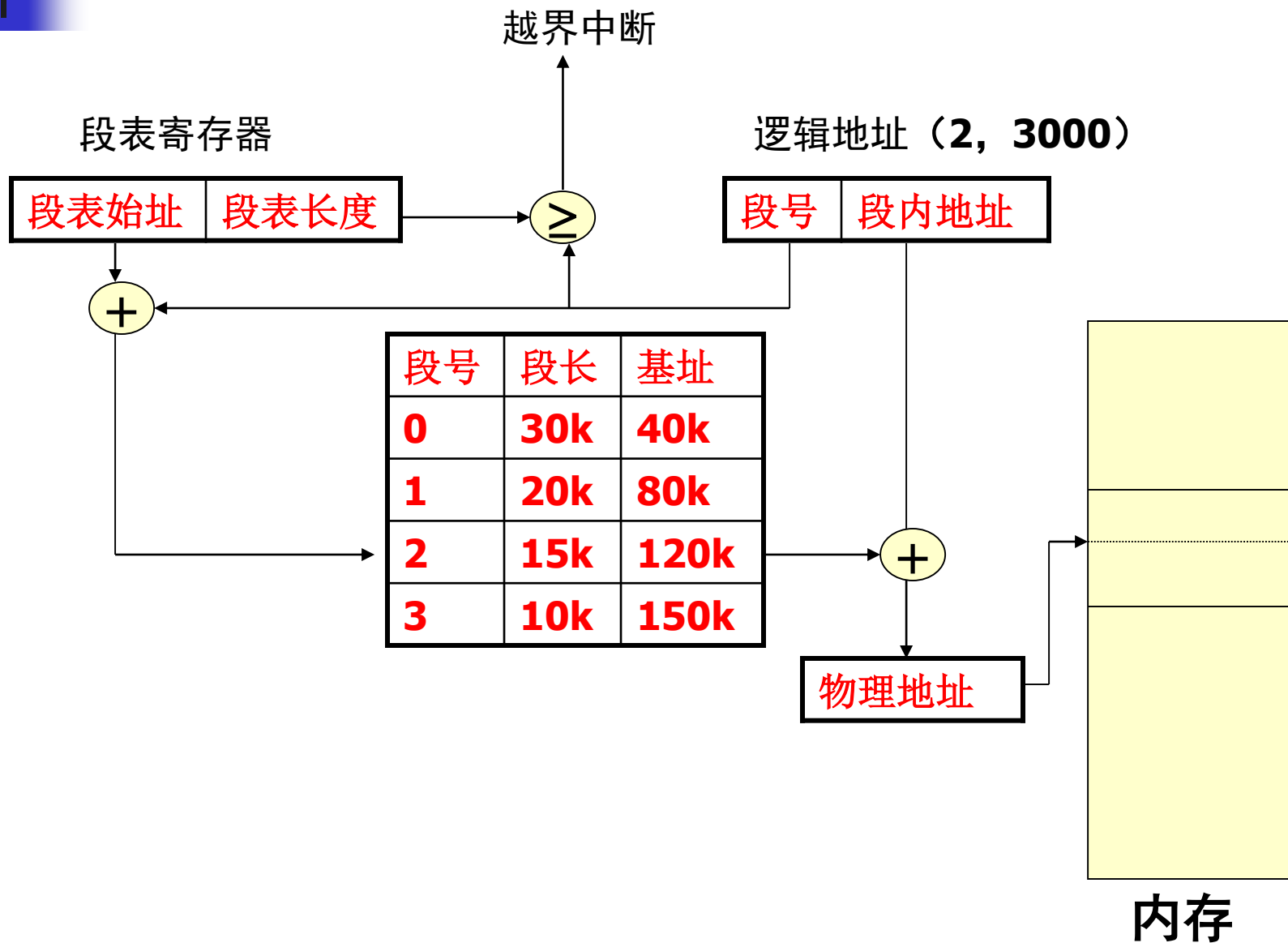
A.  $2^{24}$

B.  $2^{16}$

C.  $2^8$

D.  $2^{32}$

# 地址变换机构-实现逻辑地址向物理地址的变换





例：

1、某段表的内容如下：

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

一逻辑地址为 (2, 154)，它对应的物理地址为多少？

解：逻辑地址为：

段号	段内地址
----	------

逻辑地址 (2, 154) 的段号为2，查段表知其对应的物理地址为：  
 $480K + 154$



2、在一个段式存储管理系统中，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

试求表中逻辑地址对应的物理地址是什么？

解：逻辑地址为：

段号	段内地址
----	------

0	430
2	120

逻辑地址

0	430
---	-----

对应的物理地址为： $210+430=640$

逻辑地址

2	120
---	-----

因为段内地址 $120 >$ 段长 $90$ ，所以该段为非法段。



# 分页和分段的主要区别

	页式存储管理	段式存储管理
目的	实现非连续分配, 解决碎片问题	更好满足用户需求
信息单位	页（物理单位）	段（逻辑单位）
大小	固定（由系统定）	不定（由用户程序定）
内存分配单位	页	段
作业地址空间	一维	二维
优点	有效解决了碎片问题 有效提高内存的利用率	更好地实现数据共享与保护 段长可动态增长 便于动态链接

二者优点的结合——段页式存储管理



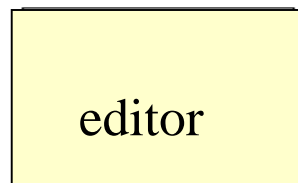
# 段的保护与共享

---

- 分段易于实现段的共享，即允许若干个进程共享一个或多个分段
- 段的共享，是通过不同作业段表中的项指向同一个段基址来实现。
- 几道作业共享的例行程序就可放在一个段中，只要让各道作业的共享部分有相同的基址/限长值。
- 对共享段的信息必须进行保护

# 段的共享

## □ 共享代码/数据

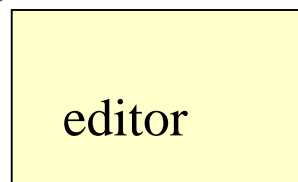


segment 0

data 1

segment 1

logical memory  
process  $P_1$



segment 0

data 2

segment 1

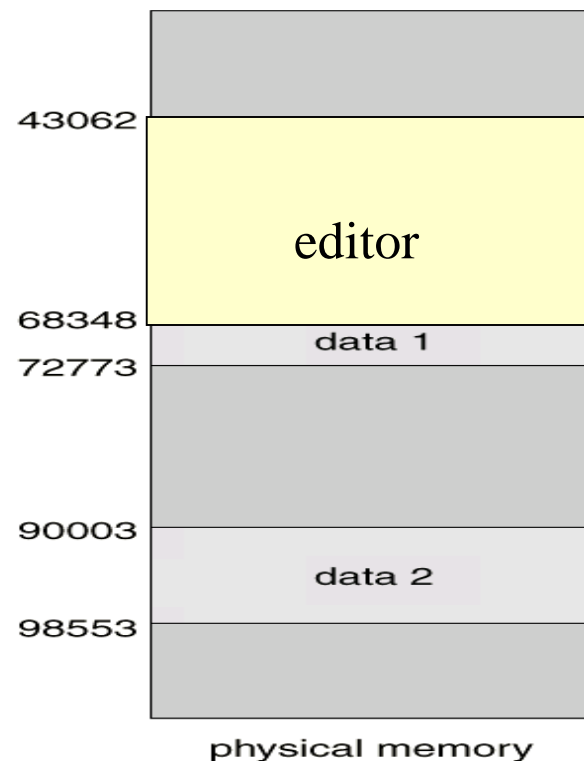
logical memory  
process  $P_2$

	limit	base	
0	25286	43062	□
1	4425	68348	

segment table  
process  $P_1$

	limit	base	
0	25286	43062	□
1	8850	90003	

segment table  
process  $P_2$



43062

editor

68348

data 1

72773

90003

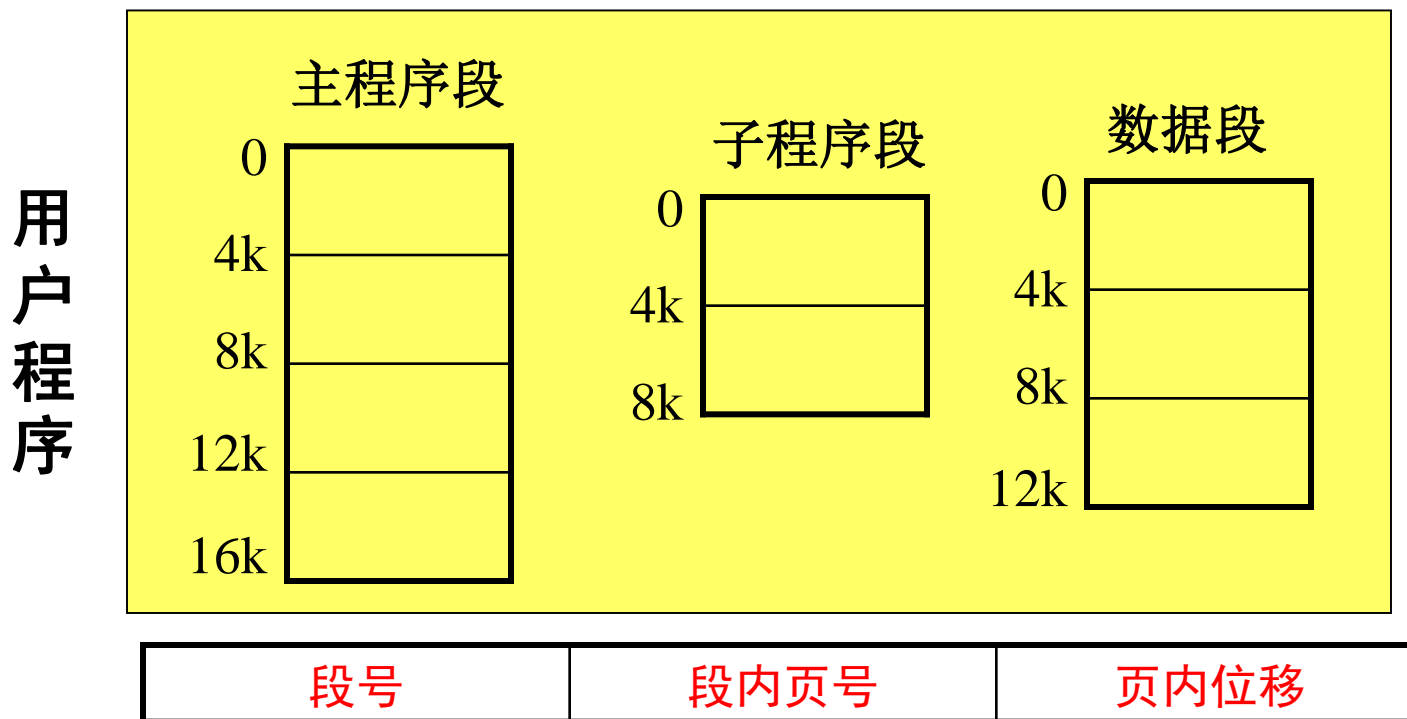
data 2

98553

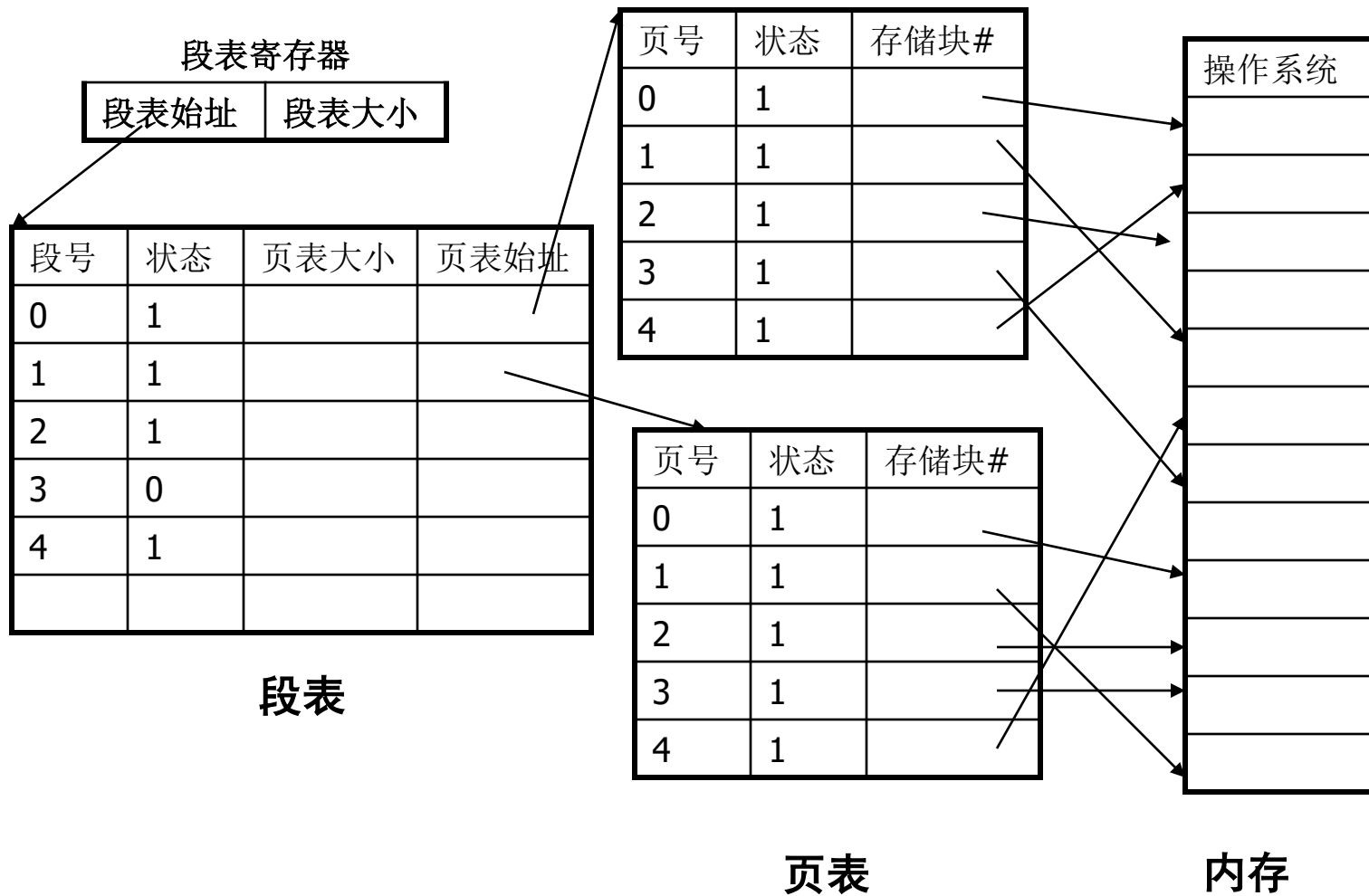
physical memory

# 段页式存储管理——基本原理

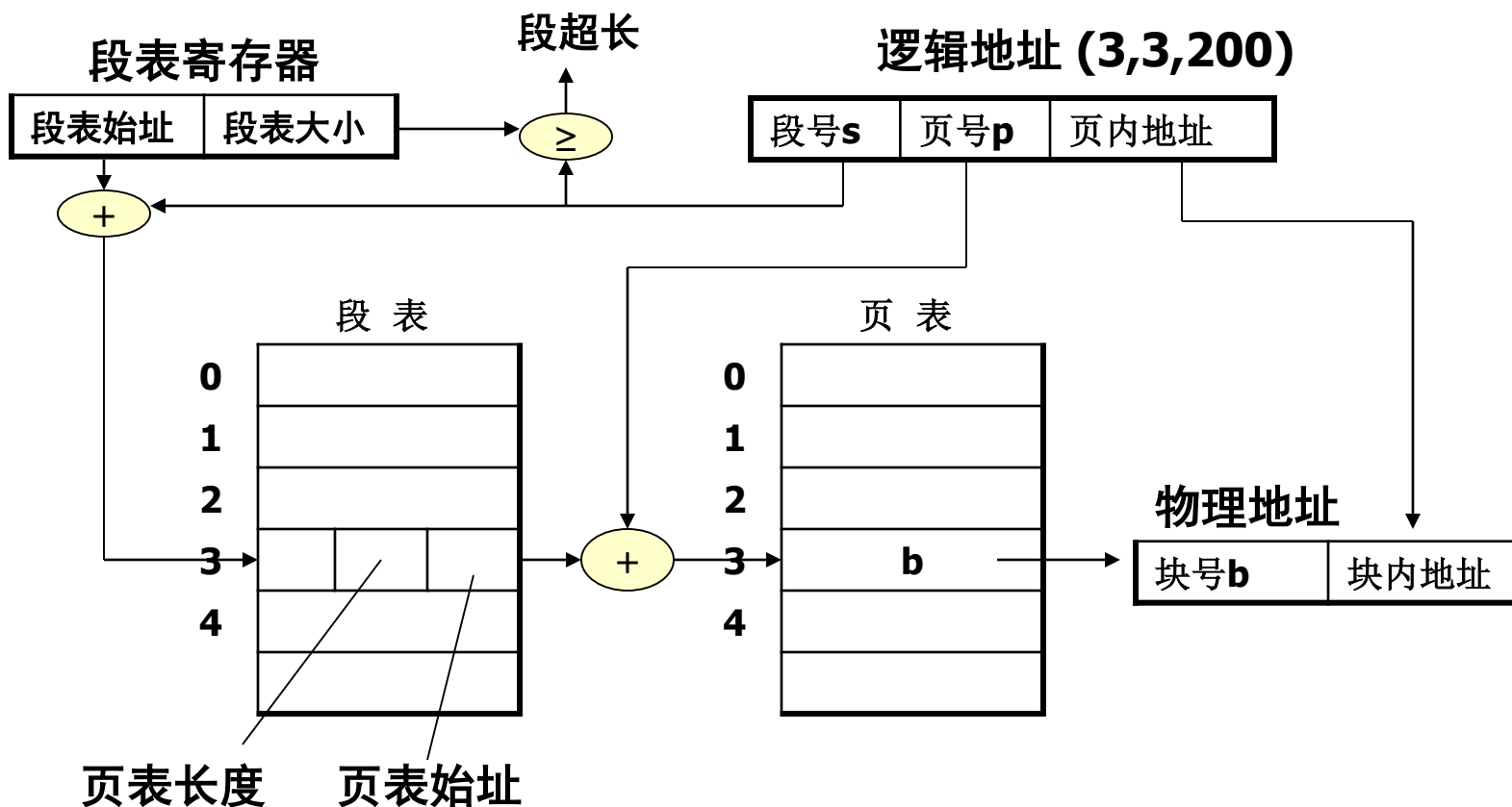
段页式存储管理是分段和分页原理的结合，即先将用户程序分成若干个段（段式），并为每一个段赋一个段名，再把每个段分成若干个页（页式）。其地址结构由段号、段内页号、及页内位移三部分组成。



# 利用段表和页表实现地址映射



# 段页式系统中的地址变换机构





# 段页式系统中的地址变换

- ❖ 系统中设段表和页表, 均存放于内存中. CPU访问一个指令或数据须访问内存三次。为提高执行速度可增设高速缓冲寄存器。
- ❖ 每个进程一张段表, 每个段一张页表。
- ❖ 段表含段号, 页表始址和页表长度; 页表含页号和块号。
- ❖ 进行地址变换:

先用段号与段寄存器中的段长进行比较, 若小于段长则利用段表始址和段号找出该段页表的始址, (否则越界中断), 再用逻辑地址中的段内页号在页表中找到相应的块号, 最后与页内位移形成物理地址。





## 4.6 虚拟存储器的基本概念

### □ 常规存储管理方式的共同点：

要求一个作业全部装入内存后方能运行。

### □ 问题：

(1) 有的作业很大, 所需内存空间大于内存总容量, 使作业无法运行。

(2) 有大量作业要求运行, 但内存容量不足以容纳下所有作业, 只能让一部分先运行, 其它在外存等待。

### □ 解决方法 (1) 增加内存容量。

(2) 从逻辑上扩充内存容量

——对换

——虚拟存储器



# 一、虚拟存储器的引入(1)

---

## ❖ 常规存储器管理方式的特征

(1) 一次性:

作业在运行前需一次性地全部装入内存。将导致上述两问题。

(2) 驻留性:

作业装入内存后, 便一直驻留内存, 直至作业运行结束。



# 一、虚拟存储器的引入(2)

## ❖ 局部性原理

指程序在执行时呈现出局部性规律，即在一较短时间内，程序的执行仅限于某个部分，相应地，它所访问的存储空间也局限于某个区域。

局部性又表现为**时间局部性**（由于大量的循环操作，某指令或数据被访问后，则不久可能会被再次访问）和**空间局部性**（如顺序执行，指程序在一段时间内访问的地址，可能集中在一定的范围之内）。



# 虚拟存储器的概念（1）

---

- ◆ 基于局部性原理，程序在运行之前，没有必要全部装入内存，仅须将当前要运行的页（段）装入内存即可。
- ◆ 运行时，如访问的页（段）在内存中，则继续执行，如访问的页（段）未在内存中（缺页或缺段），则利用OS的请求调页（段）功能，将该页（段）调入内存。
- ◆ 如内存已满，则利用OS的页（段）置换功能，按某种置换算法将内存中的某页（段）调至外存，从而调入需访问的页（段）。



## 虚拟存储器的概念（2）

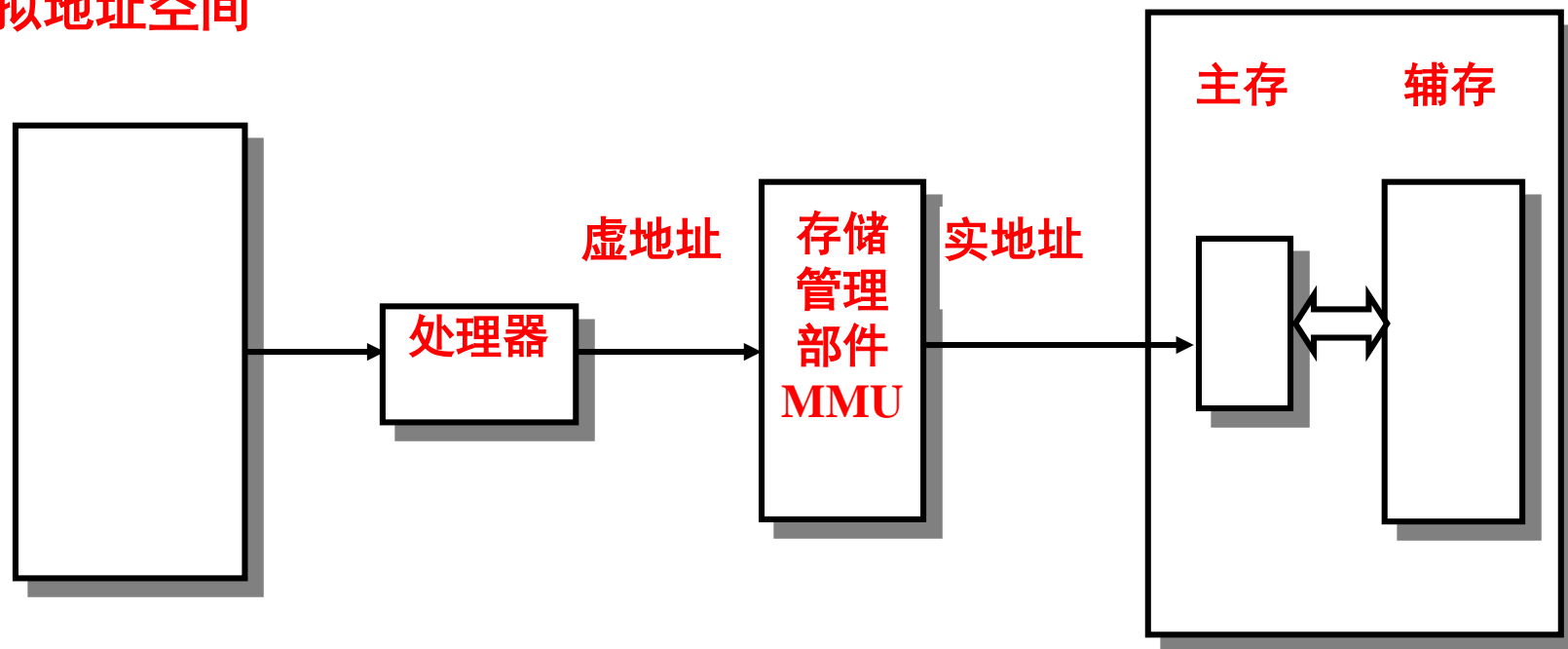
**虚拟存储器**是指仅把作业的一部分装入内存便可运行作业的存储管理系统，它具有请求调页功能和页面置换功能，能从逻辑上对内存容量进行扩充，其**逻辑容量**由外存容量和内存容量之和决定，其**最大容量**由计算机的地址结构决定，其运行速度接近于内存，成本接近于外存。

对于一台内存为256MB的32bit x86主机来说，它的虚拟地址空间范围是 $0 \sim 0xFFFFFFFF$ （4GB），而物理地址空间范围是 $0 \sim 0x0FFFFFFF$ （256MB）。对于64位的CPU，它的虚拟地址空间范围是 $0 \sim 0xFFFFFFFFFFFFFFFF$ （ $2^{64}B$ ）。

# 虚拟存储器的概念图

虚拟地址空间

物理地址空间



MMU是用来管理虚拟内存系统的器件，通常是CPU的一部分，本身有少量存储空间存放从虚拟地址到物理地址的匹配表（TLB）。所有数据请求都送往MMU，由MMU决定数据是在RAM内还是在辅存。如果数据不在内存，MMU将产生页面错误中断。



## 二、虚拟存储器的特征

### 1、多次性

多次性是虚拟存储器最重要的特征。指一个作业被分成多次调入内存运行。

### 2、对换性

对换性指允许在作业运行过程中进行换进、换出。换进、换出可提高内存利用率。

### 3、虚拟性

虚拟性是指能够从逻辑上扩充内存容量，使用户所看到的内存容量远大于实际内存容量。虚拟性是虚拟存储器所表现出来的最重要的特征，也是实现虚拟存储器最重要的目标。

**注：**虚拟性以多次性和对换性为基础，而多次性和对换性又是以**离散**分配为基础。



## 三、虚拟存储器的实现方法

---

### ◆ 实现虚拟存储器必须解决好以下有关问题：

- ◆ 主存辅存统一管理问题
- ◆ 逻辑地址到物理地址的转换问题
- ◆ 部分装入和部分对换问题

### ◆ 虚拟存储管理主要采用以下技术实现：

- ◆ 请求分页存储管理
- ◆ 请求分段存储管理
- ◆ 请求段页式存储管理





## 三、虚拟存储器的实现方法

### 1、请求分页系统

在分页系统的基础上，增加了请求调页功能、页面置换功能所形成的页式虚拟存储器系统。

它允许只装入若干页的用户程序和数据，便可启动运行，以后在硬件支持下通过调页功能和页面置换功能，陆续将要运行的页面调入内存，同时把暂不运行的页面换到外存上，置换时以页面为单位。

**系统须设置相应的硬件支持和软件：**

(1) 硬件支持：请求分页的页表机制、缺页中断机构和地址变换机构。

(2) 软件：请求调页功能和页面置换功能的软件。



# 三、虚拟存储器的实现方法

## 2、请求分段系统

在分段系统的基础上，增加了请求调段功能及分段置换功能，所形成的段式虚拟存储器系统。

它允许只装入若干段的用户程序和数据，便可启动运行，以后在硬件支持下通过请求调段功能和分段置换功能，陆续将要运行的段调入内存，同时把暂不运行的段换到外存上，置换时以段为单位。

**系统须设置相应的硬件支持和软件：**

(1) 硬件支持：请求分段的段表机制、缺段中断机构和地址变换机构

(2) 软件：请求调段功能和段置换功能的软件

## 4.7 请求分页存储管理方式

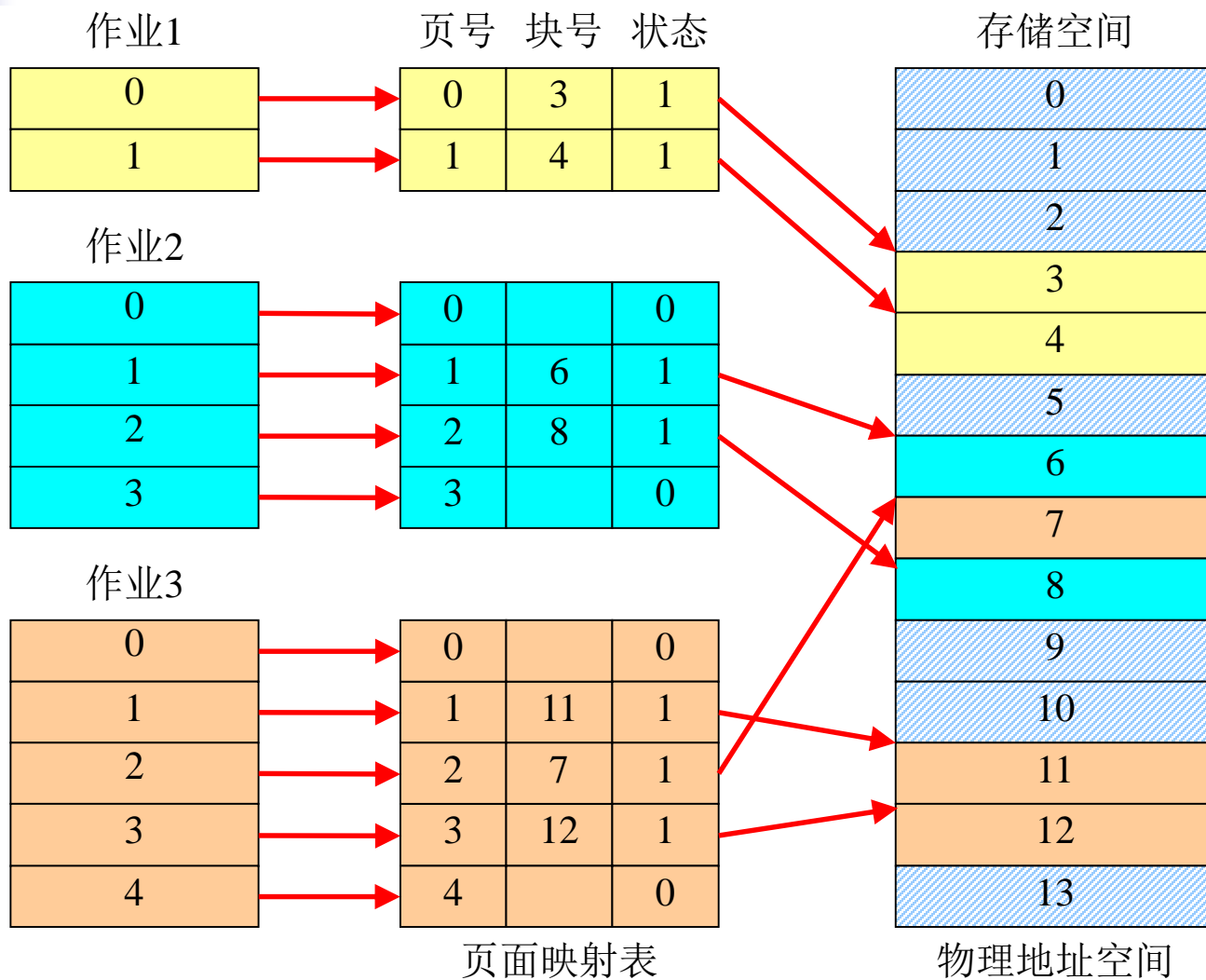
### ❖ 虚拟存储器的实现方式

	请求分页系统	请求分段系统
基本单位	页	段
长度	固定	可变
分配方式	固定分配	动态
复杂性	简单	较复杂

❖ **原理**——地址空间的划分与页式存储管理相同；装入页时，装入作业的一部分(即运行所需的)页即可运行。

- 请求分页中的硬件支持
- 请求分页中的内存分配策略和分配算法
- 请求分页中的页面调入策略

# 请求分页存储管理示意图





# 一、请求分页中的硬件支持

## 1、页表机制

页号	块号	状态位	访问字段	修改位	外存地址
----	----	-----	------	-----	------

- (1) 状态位P：指示该页是否已调入内存。
- (2) 访问字段A：记录本页在一段时间内被访问的次数或最近未被访问的时间。
- (3) 修改位M：表示该页在调入内存后是否被修改过。若修改过，则换出时需重写至外存。
- (4) 外存地址：指出该页在外存上的地址。



# 一、请求分页中的硬件支持

---

## 2、缺页中断机构

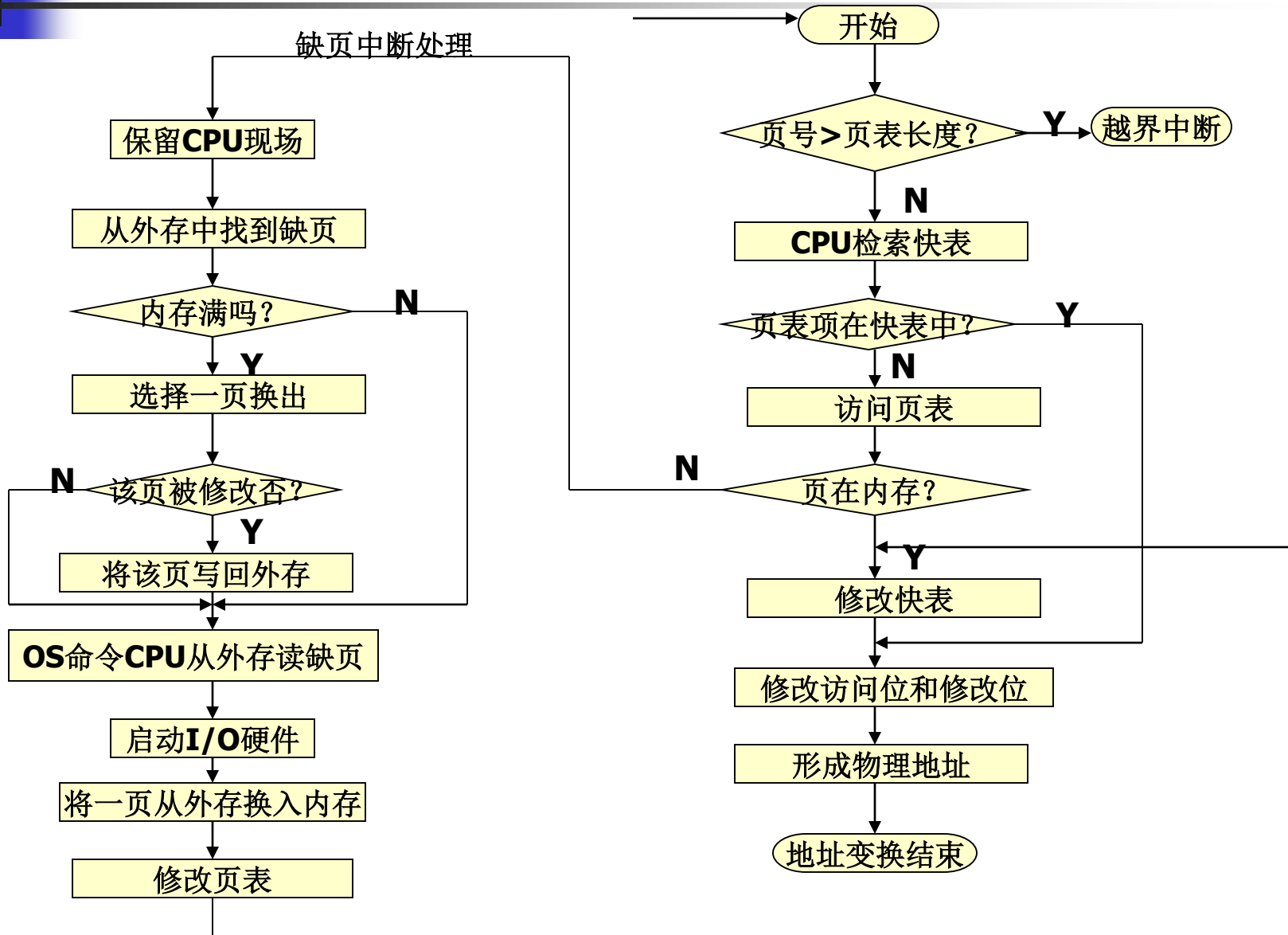
在请求分页系统中，当访问的页不在内存，便产生一缺页中断，请求OS将所缺页调入内存空闲块，若无空闲块，则需置换某一页，同时修改相应页表表目。

### 缺页中断与一般中断的区别：

- (1) 在指令执行期间产生和处理中断信号
- (2) 一条指令在执行期间，可能产生多次缺页中断

# 3、地址变换机构

程序请求访问一页



# 地址变换例题

❖ 某虚拟存储器的用户空间共有32个页面, 每页1KB, 主存16KB. 假定某时刻系统为用户的第0、1、2、3页分别分配的物理块号为5、10、4、7, 试将虚拟地址0A5C和093C变换为物理地址。

解:

虚拟地址为: 页号 ( $2^5=32$ ) 5位    页内位移 ( $2^{10}=1024$ ) 10位

物理地址为: 物理块号 ( $2^4=16$ ) 4位    块内位移 ( $2^{10}=1024$ ) 10位

虚拟地址0A5C对应的二进制为:

00010    1001011100

即虚拟地址0A5C的页号为2, 页内位移为1001011100

由题意知对应的物理地址为: 0100    1001011100 即125C

同理求093C。略



## 二、请求分页中的内存分配策略和分配算法

在请求分页系统中，为进程分配内存时，将涉及以下三个问题：

### 1、最小物理块数的确定

最小物理块数指能保证进程正常运行所需的最小的物理块数，与计算机的硬件结构有关，取决于指令的格式、功能和寻址方式。

### 2、物理块的分配策略和页面置换策略

### 3、物理块分配算法



## 2、物理块的分配策略和页面置换策略

(1) 固定分配局部置换：为每个进程分配**固定数目 $n$ 的物理块**，在整个运行中都不改变。如出现缺页，则从中置换一页。

(2) 可变分配全局置换：分配**固定数目的物理块**，但OS自留一空闲块队列，若发现缺页，则从空闲块队列中分配一空闲块给该进程，并调入缺页于其中。当空闲块队列用完时，OS才从内存中选择一页置换。

(3) 可变分配局部置换：分配一定数目的物理块，若发现缺页，则从**该进程**的页面中置换一页，根据该进程缺页率高低，则**可增加或减少物理块**。



### 3、物理块分配算法

---

在采用固定分配策略时，将系统中可供分配的所有物理块分配给各个进程，可采用以下几种算法：

- (1) 平均分配算法：平均分配给各个进程。
- (2) 按比例分配算法：根据进程的大小按比例分配给各个进程。
- (3) 考虑优先权的分配算法：将系统提供的物理块一部分根据进程大小先按比例分配给各个进程，另一部分再根据各进程的优先权适当增加物理块数。



### 三、请求分页中的页面调入策略

调入策略决定什么时候将一个页面由外存调入内存，从何处将页面调入内存，如何调入。

#### □何时调入页面（1）

❖预调页策略：将那些**预计**在不久便被访问的页预先调入内存。这种调入策略提高了调页的效率，减少了I/O次数。但由于这是一种基于局部性原理的预测，若预调入的页面在以后很少被访问，则造成浪费，故这种方式常用于程序的首次调入。



## 三、请求分页中的页面调入策略

### □何时调入页面（2）

❖**请求调页策略**：当进程运行中访问的页不在内存时，则发出缺页中断，提出请求调页，由OS将所需页调入内存。这种策略实现简单，应用于目前的虚拟存储器中，但易产生较多的缺页中断，且每次调一页，系统开销较大，容易产生抖动现象。



### 三、请求分页中的页面调入策略

#### □ 从何处调入页面（1）

在请求分页系统中，通常将外存分成了文件区和对换区，文件区按离散分配方式存放文件，对换区按连续分配方式存放对换页。

❖ **对换区**：系统有足够的对换区空间，运行前可将与进程相关的文件从文件区复制至对换区，以后缺页时，**全部从对换区调页**。



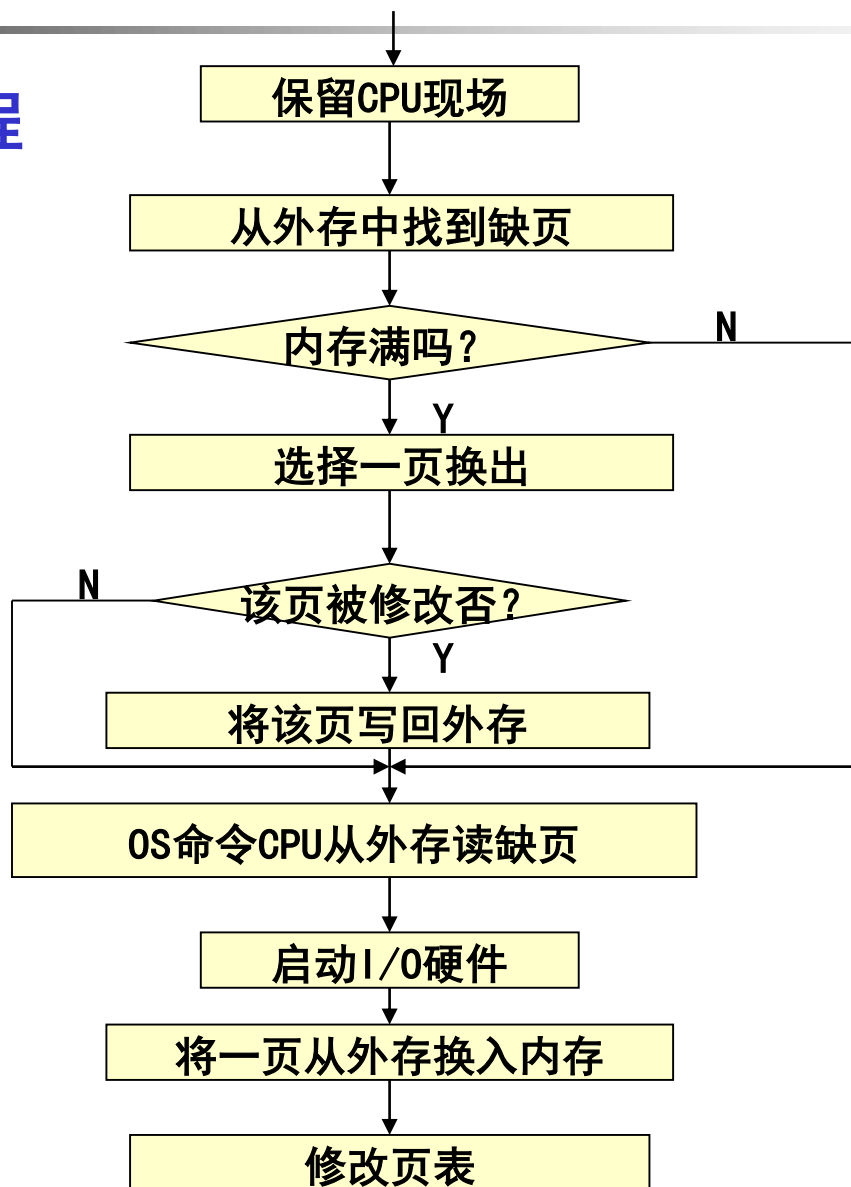
## 三、请求分页中的页面调入策略

### □ 从何处调入页面（2）

- ❖ **文件区**：系统没有足够的对换区空间，凡是不会被修改的文件，每次都直接从文件区调页，换出时不必换出。
- ❖ **文件区、对换区**：系统没有足够的对换区空间，对可能会修改的文件第一次调页直接从文件区，换出时换至对换区，以后从对换区调页。
- ❖ **UNIX方式**：凡未运行过的页面均从文件区调页，运行过的页面和换出的页面均从对换区调页。

### 三、请求分页中的页面调入策略

#### □ 页面调入过程







## 三、请求分页中的页面调入策略

### ❑ 缺页率

❖ **缺页率**：假设一个进程的逻辑空间为 $n$ 页，系统为其分配的内存物理块数为 $m$  ( $m \leq n$ )，如果在进程的运行过程中，访问页面成功的次数为 $S$ ，访问页面失败的次数为 $F$ ，则该进程总的页面访问次数为 $A=S+F$ ，那么进程在运行过程中的缺页率为：

$$f=F/A$$

❖ **影响缺页率的因素**：

- ❖ 页面大小
- ❖ 进程所分配物理块的数目
- ❖ 页面置换算法
- ❖ 程序固有属性



## 4.8 请求分页中的页面置换算法

页面置换算法也称为页面淘汰算法，是用来选择换出页面的算法。页面置换算法的优劣直接影响到系统的效率，若选择不合适，可能会出现以下现象：

刚被淘汰出内存的页面，过后不久又要访问它，需要再次将其调入，而该页调入内存后不久又再次被淘汰出内存，然后又要访问它，如此反复，使得系统把大部分时间用在了页面的调进换出上，而几乎不能完成任何有效的工作，这种现象称为**抖动**（又称**颠簸**）。



## 4.8 请求分页中的页面置换算法

常用的页面置换算法：

- ❖ 最佳置换算法：选择永远不再需要的页面或最长时间以后才需要访问的页面予以淘汰。
- ❖ 先进先出置换算法FIFO：选择先进入内存的页面予以淘汰。
- ❖ 最近最久未使用置换算法LRU：选择最近一段时间最长时间没有被访问过的页面予以淘汰。
- ❖ Clock置换算法
- ❖ \*其它算法

# 最佳置换算法例

假定系统为某进程分配了3个物理块，进程运行时的页面走向为 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5，开始时3个物理块均为空，计算采用**最佳置换**页面淘汰算法时的缺页率？

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	1			1			3	3	
物理块2		2	2	2			2			2	4	
物理块3			3	4			5			5	5	
缺页	缺	缺	缺	缺			缺			缺	缺	

缺页率=7/12

**注：**实际上这种算法无法实现，因页面访问的未来顺序很难精确预测，但可用该算法评价其它算法的优劣。



# 先进先出置换算法例题

1、假定系统为某进程分配了3个物理块，进程运行时的页面走向为 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5，开始时3个物理块均为空，计算采用先进先出页面淘汰算法时的缺页率？

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	4	4	4	5			5	5	
物理块2		2	2	2	1	1	1			3	3	
物理块3			3	3	3	2	2			2	4	
缺页	缺	缺	缺	缺	缺	缺	缺			缺	缺	

缺页率=9/12

## 先进先出置换算法例题

2、假定系统为某进程分配了4个物理块，进程运行时的页面走向为 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5，开始时4个物理块均为空，计算采用先进先出页面淘汰算法时的缺页率？

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	1			5	5	5	5	4	4
物理块2		2	2	2			2	1	1	1	1	5
物理块3			3	3			3	3	2	2	2	2
物理块4				4			4	4	4	3	3	3
缺页	缺	缺	缺	缺			缺	缺	缺	缺	缺	缺

缺页率=10/12

## 先进先出置换算法例题

3、假定系统为某进程分配了5个物理块，进程运行时的页面走向为 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5，开始时5个物理块均为空，计算采用先进先出页面淘汰算法时的缺页率？

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	1			1					
物理块2		2	2	2			2					
物理块3			3	3			3					
物理块4				4			4					
物理块5							5					
缺页	缺	缺	缺	缺			缺					

缺页率=5/12

# 先进先出置换算法\_注（1）：

- 1、该算法的出发点是最早调入内存的页面不再被访问的可能性会大一些。
- 2、该算法实现比较简单，对具有线性顺序访问的程序比较合适，而对其他情况效率不高。因为经常被访问的页面，往往在内存中停留最久，结果这些常用的页面却因变老而被淘汰。

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	4	4	4	5			5	5	
物理块2		2	2	2	1	1	1			3	3	
物理块3			3	3	3	2	2			2	4	
缺页	缺	缺	缺	缺	缺	缺	缺			缺	缺	





## 先进先出置换算法\_注（2）：

**3、**先进先出算法存在一种异常现象，即在某些情况下会出现分配给的进程物理块数增多，缺页次数有时增加，有时减少的奇怪现象，这种现象称为**Belady现象**。如上几例：

物理块数	3	4	5
缺页次数	9	10	5



## 最近最久未使用算法例

假定系统为某进程分配了3个物理块，进程运行时的页面走向为 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5，开始时3个物理块均为空，计算采用最近最久未使用页面淘汰算法时的缺页率？

页面走向	1	2	3	4	1	2	5	1	2	3	4	5
物理块1	1	1	1	4	4	4	5			3	3	3
物理块2		2	2	2	1	1	1			1	4	4
物理块3			3	3	3	2	2			2	2	5
缺页	缺	缺	缺	缺	缺	缺	缺			缺	缺	缺

缺页率=10/12



## 最近最久未使用算法\_注 (1)

❖ **该算法的出发点**：如果某个页面被访问了，则它可能马上还要被访问。反之，如果很长时间未被访问，则它在最近一段时间也不会被访问。

❖ **该算法的性能接近于最佳算法，但实现起来较困难**。因为要找出最近最久未使用的页面，必须为每一页设置相关记录项，用于记录页面的访问情况，并且每访问一次页面都须更新该信息。这将使系统的开销加大，所以在实际系统中**往往使用该算法的近似算法**。



## 最近最久未使用算法\_注 (2)

### ❖ 该算法的近似算法实现：

方法1：利用一特殊的栈保存当前使用的页号，每当进程访问某页面时，把被访问页面移到栈顶，于是栈底的页面就是最久未使用的页面。

方法2：为每个页面设立一个寄存器记录页面的访问情况。每当进程访问某页面时，将该页面对应寄存器的最高位置1，系统定期将寄存器右移一位并将最高位补0，于是寄存器数值最小的页面是最久未使用的页面。



# Clock置换算法

## (1) 简单Clock置换算法

该算法是LRU和FIFO的折衷。该算法要求为每页设置一个访问位，并将内存中的所有页链接成一个循环队列。

当某页被访问时，系统将其访问位设置为1。

置换时采用一个指针，从当前指针位置开始按序检查各页，若访问位为0则选择该页换出，若访问位为1则将其设置为0，最后指针停留在被置换页的下一页上。

块号	页号	访问位	指针
0			
1			
2	4	0	←
3			
4	2	1	←
5			
6	5	0	←
7	1	1	←

替换指针



# Clock置换算法

## (1) 简单Clock置换算法

- (1) 一个页面首次装入内存时，将其“访问位”置为1；
- (2) 在内存中的任何一个页面被访问时，将其“访问位”置为1；
- (3) 淘汰页面时，存储管理从指针当前指向的页面开始扫描循环队列，把所遇到的“访问位”是1的页面的“访问位”清0，并跳过这个页面，把所遇到的“访问位”是0的页面淘汰掉，指针前进一步；



# Clock置换算法

## (1) 简单Clock置换算法

(4) 扫描循环队列时，如果遇到的所有页面的“访问位”为1，指针会遍历整个循环队列一圈，把遇到的所有页面的“访问位”清0，指针停在起始位置，淘汰掉该页，然后指针前进一步。

## (2) 改进型Clock置换算法

该算法要求除须考虑页面的使用情况外，还须再增加一个因素，即置换代价，这样，选择页面换出时，既要是未使用过的页面，又要是未被修改过的页面。把同时满足这两个条件的页面作为首选淘汰的页面。由页表中的访问位A和修改位M组合成4种类型的页面：

$A=0, M=0$  （最佳淘汰页）

$A=0, M=1$  （不是很好的淘汰页）

$A=1, M=0$  （有可能再被访问）

$A=1, M=1$  （可能再被访问）



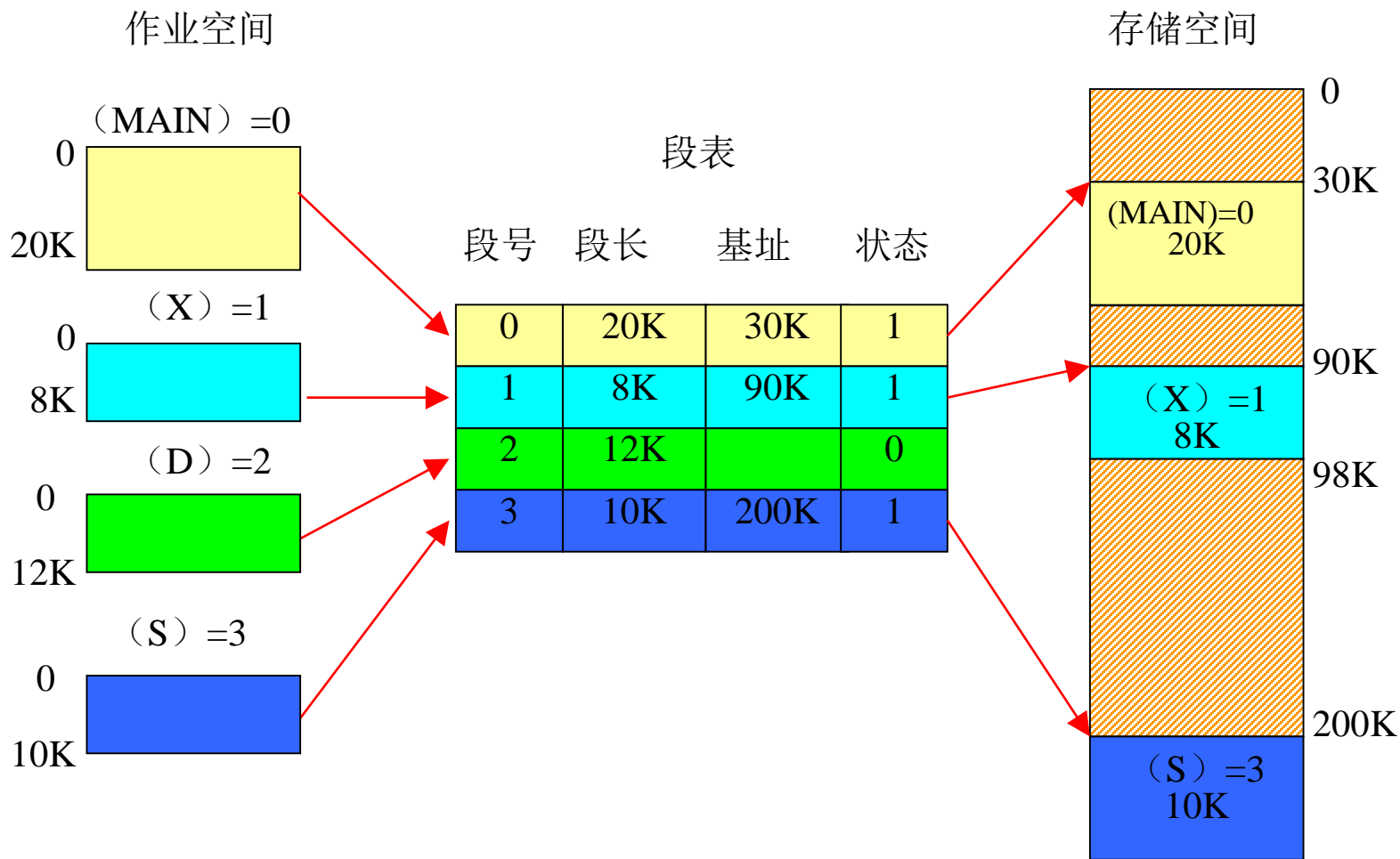




## 4.9 请求分段式存储管理方式

- ❖ 请求分段存储管理系统也与请求分页存储管理系统一样，为用户提供了一个比内存空间大得多的**虚拟存储器**。虚拟存储器的最大容量由计算机的地址结构确定。
- ❖ 在请求分段存储管理系统中，作业运行之前，只要求将当前需要的若干个分段装入内存，便可启动作业运行。在作业运行过程中，如果要访问的分段不在内存中，则通过**调段功能**将其调入，同时还可以通过**置换功能**将暂时不用的分段换出到外存，以便腾出内存空间。

# 请求分段存储管理系统的示意图





## 4.9 请求分段式存储管理方式

---

### ❖ 请求分段中的硬件支持

- 段表机制
- 缺段中断机构
- 地址变换机构

### ❖ 分段共享与保护

- 共享段表
- 共享段的分配与回收
- 分段保护（越界检查、存取控制检查、环境保护机构）



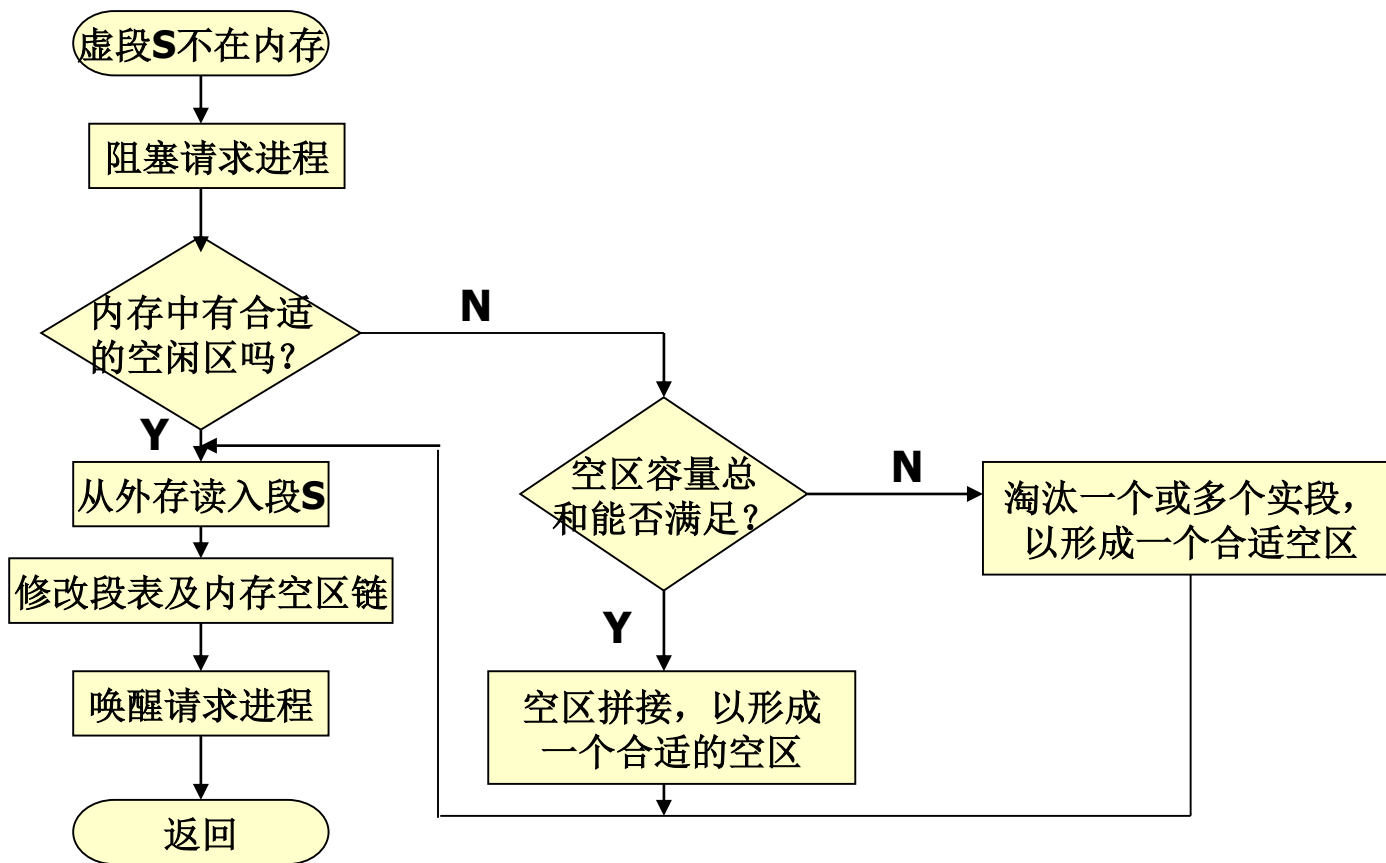
# 段表机制

段名	段长	段的基址	存取方式	访问字段 <b>A</b>	修改位 <b>M</b>	存在位 <b>P</b>	增补位	外存地址
----	----	------	------	------------------	-----------------	-----------------	-----	------

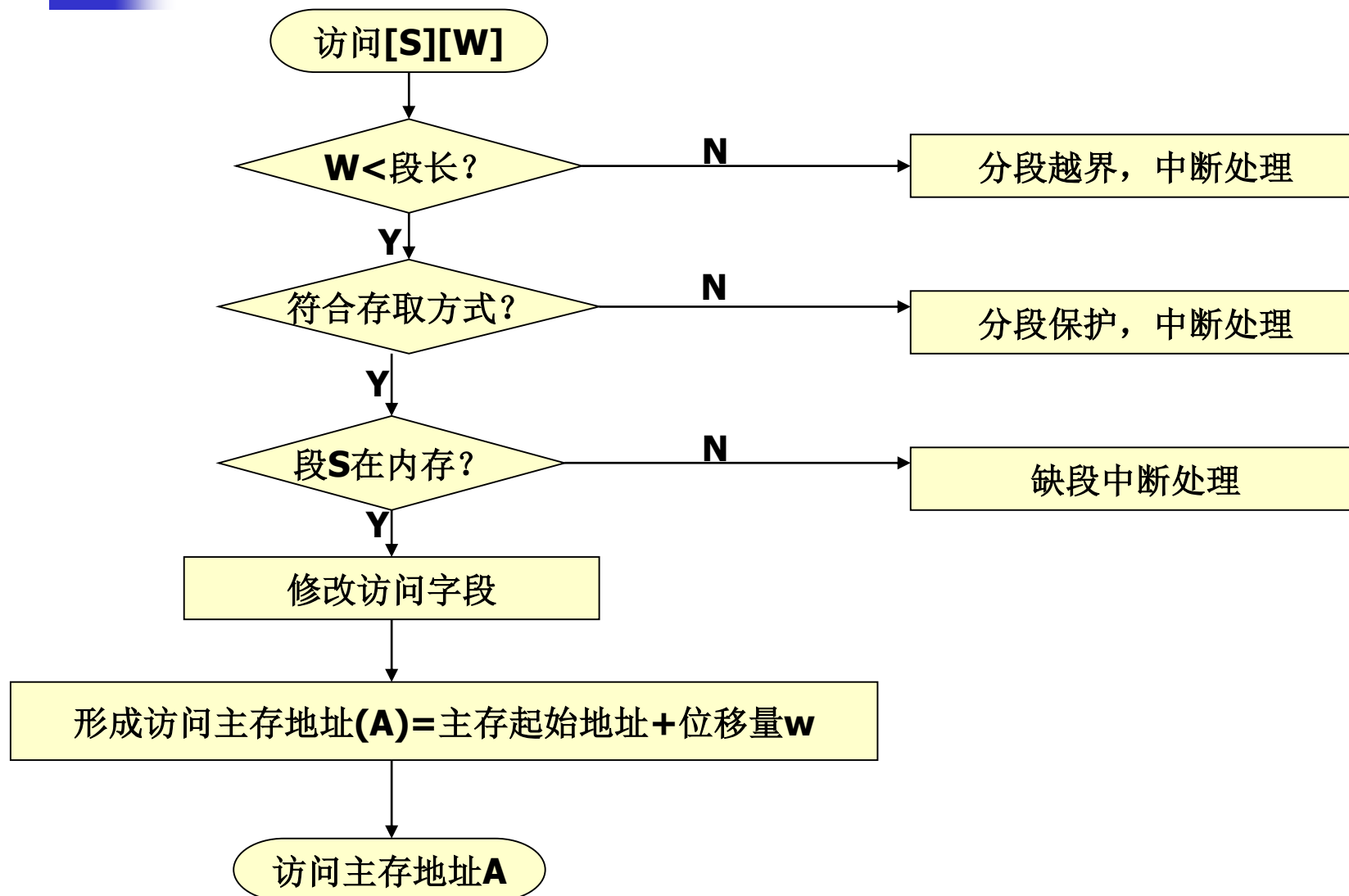
- ❖ 存取方式：存取属性（执行、只读、允许读/写）
- ❖ 访问字段A：记录该段被访问的频繁程度
- ❖ 修改位M：表示该段在进入内存后，是否被修改过。
- ❖ 存在位P：表示该段是否在内存中。
- ❖ 增补位：表示在运行过程中，该段是否做过动态增长。
- ❖ 外存地址：表示该段在外存中的起始地址。

# 缺段中断机构

❖ 当被访问的段不在内存中时，将产生一缺段中断信号。缺段中断的处理过程如图：



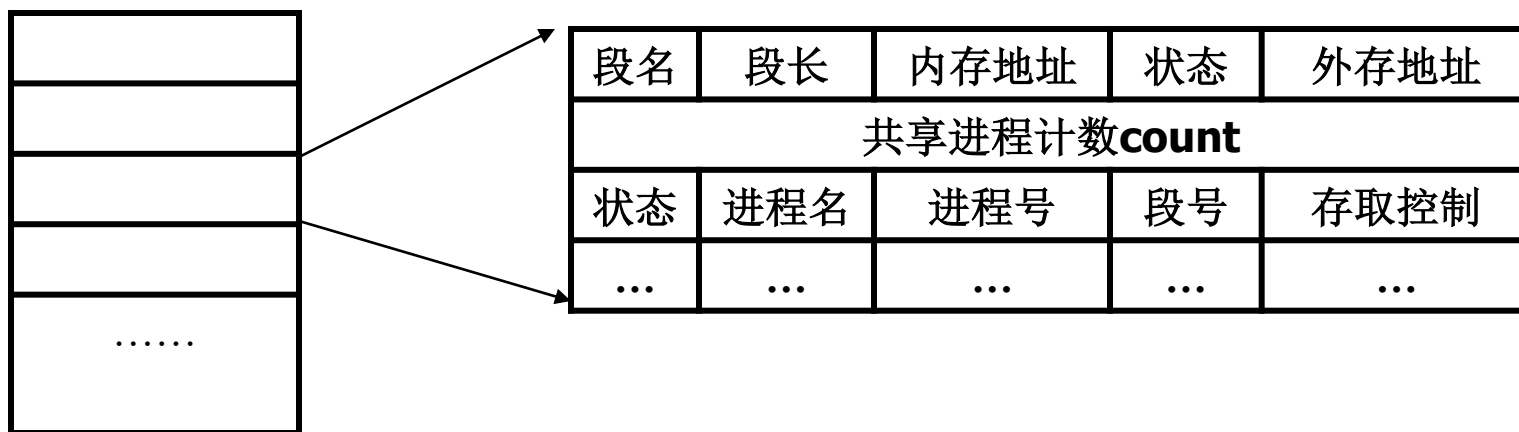
# 地址变换机构



# 分段的共享与保护

为实现分段的共享，应配置相应的数据结构：

## ❖ 共享段表

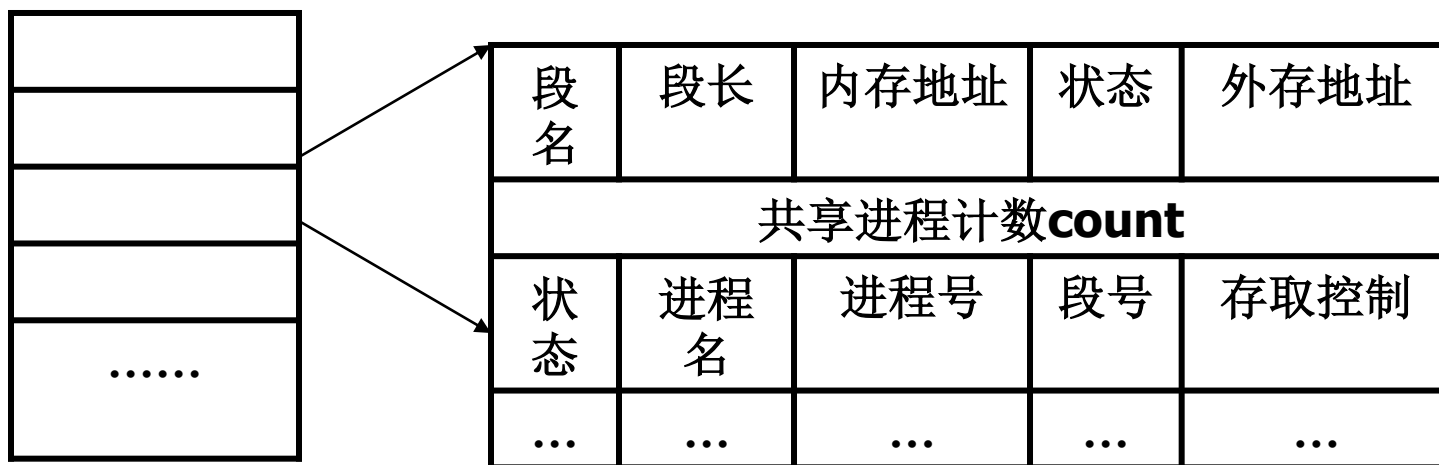


共享段表

# 共享段的分配与回收 (1)

## ❖ 共享段的分配

当第一个使用共享段的进程提出请求时，由系统为该共享段分配一物理区，并调入该共享段，同时修改相应的段表（该段的内存地址）和共享段表。当其它进程需要调用此段时，不需调入，只需修改相应的段表和共享段表。



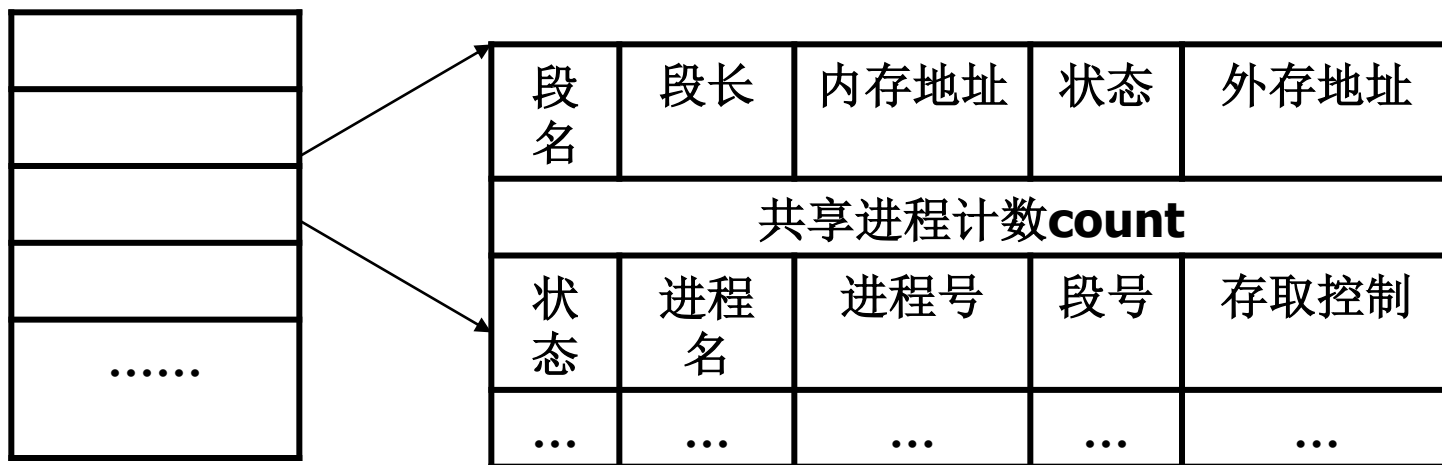
共享段表



## 共享段的分配与回收（2）

### ❖ 共享段的回收

当共享共享段的某进程不再使用该共享段时，修改相应的段表和共享段表。当最后一个共享此段的进程也不再需要此段时，则系统回收此共享段的物理区，同时修改共享段表（删除该表项）。



共享段表