华中农业大学
HUAZHONG AGRICULTURAL UNIVERSITY

计算机操作系统

倪福川
fcni_cn@mail.hzau.edu.cn

华中农业大学 信息学院

华中农业大学
HUAZHONG AGRICULTURAL UNIVERSITY

目 录

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 连续分配存储管理方式

4.4 对换

4.5 分页存储管理方式

4.5 分段存储管理方式

4.1 存储器的层次结构

4.1.1 多级存储器结构

4.1.2 主存储器与寄存器

4.1.3 高速缓存和磁盘缓存

4.1.1 多级存储器结构

存储层次至少应有三级：CPU寄存器、主存、辅存

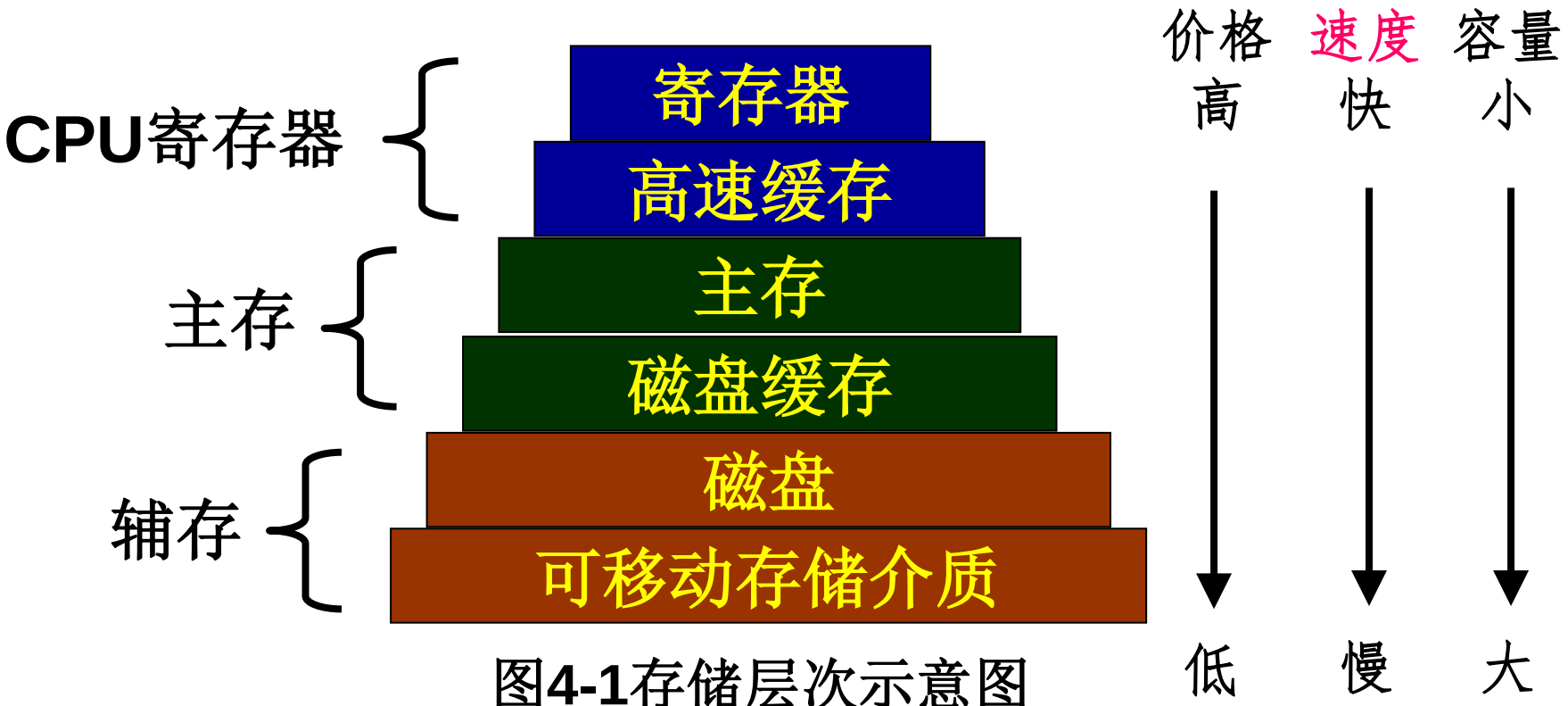


图4-1存储层次示意图

4.1.2 主存储器与寄存器

1、主存储器

主存也称可执行存储器。CPU可从其中取指令和数据，数据能从主存读取并装入到寄存器中，或从寄存器存入到主存。

2、寄存器

寄存器访问速度最快。其长度以字为单位。

4.1.3 高速缓存和磁盘缓存

1、高速缓存（cache）

容量大于寄存器，访问速度快于内存。

Cache分类：

一级cache紧靠内存，速度最高，容量最小。

二级cache容量稍大，速度也稍低。

2、磁盘缓存

本身并不是一种实际存储介质。

实质：利用主存中存储空间，暂存从磁盘中读出或写入的信息

4.2 程序的装入和链接

从源程序到程序执行

地址空间的概念

重定位的概念

程序的装入

程序的链接

1、从源程序到程序执行

编译：编译程序

将源代码编译成若干个目标模块。

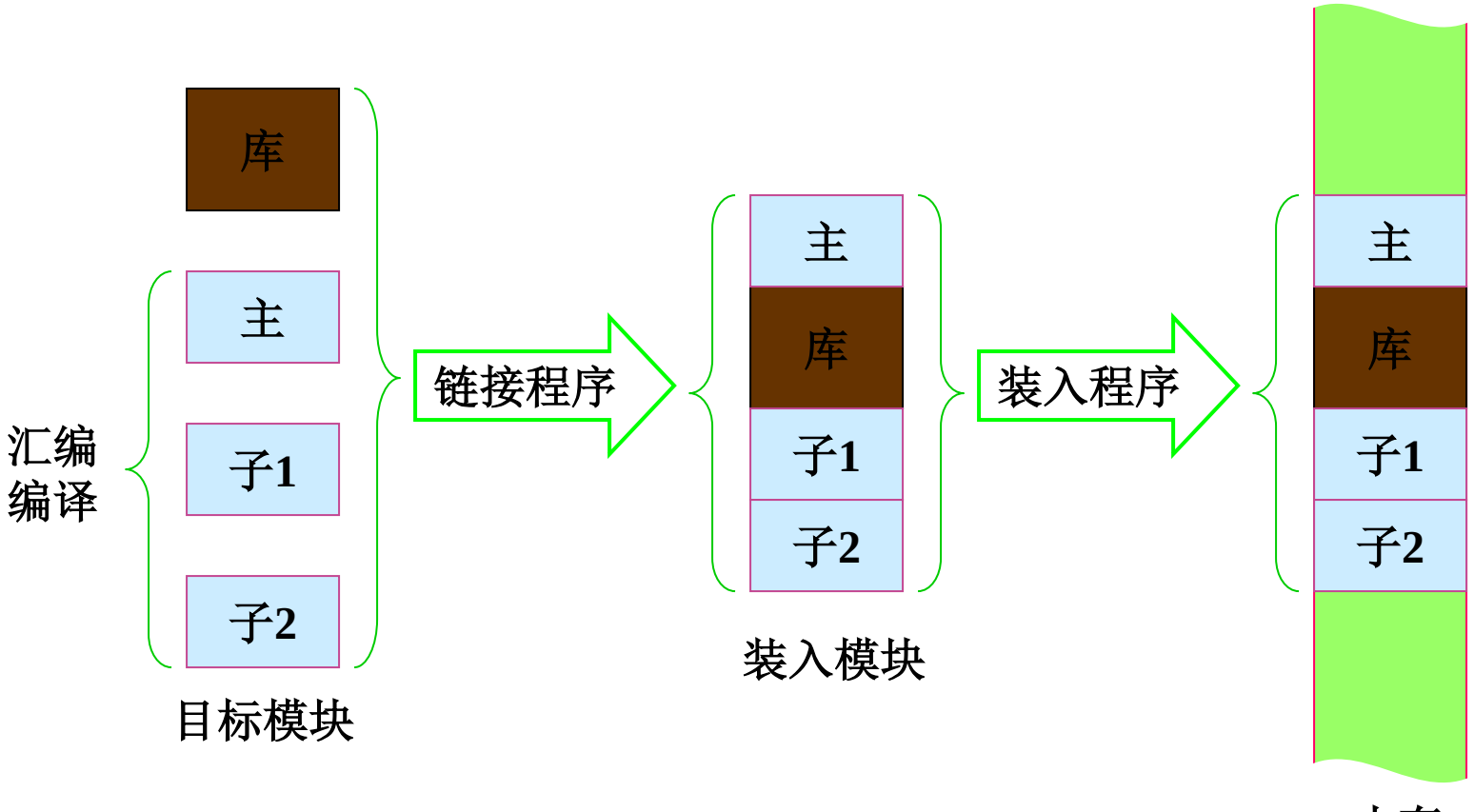
链接：链接程序

将编译后形成的一组目标模块，以及它们所需要的库函数链接在一起，形成装入模块。

装入：装入程序

将装入模块复制到内存中。

1、从源程序到程序执行



2、地址空间的概念

物理（绝对）地址——程序执行

每个内存单元的固定顺序地址（编号）。

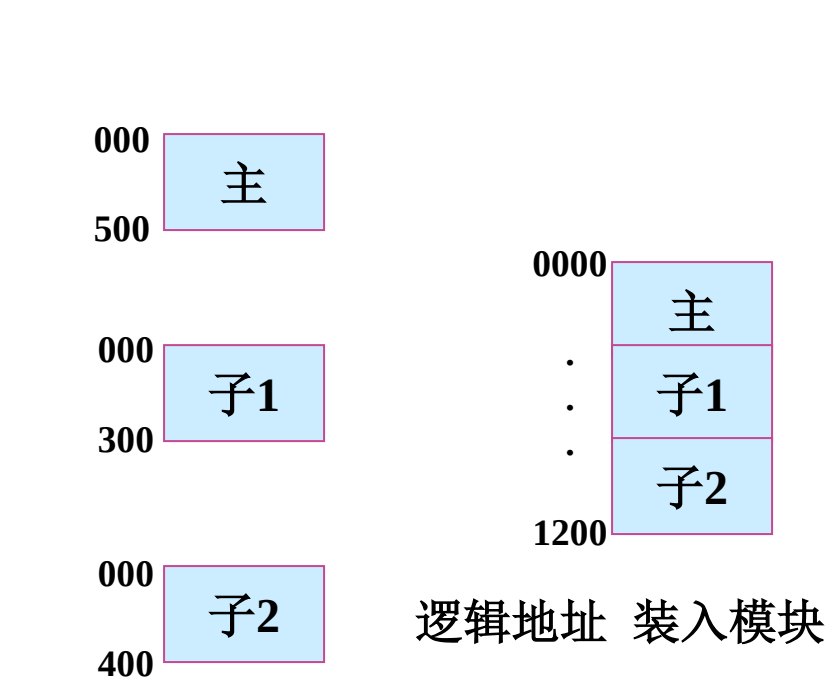
内存：由字或字节组成的一维线性地址空间

逻辑（相对）地址——装入（汇编编译）

被链接装配（或汇编、编译）后的目标模块所限定的地址的集合；

相对于某个基准量（通常为：0）的编址。

2、地址空间的概念


000
主
500
000
子1
300
000
子2
400
相对地址
源程序/单个目标模块
0000
主
子1
子2
1200
逻辑地址 装入模块
00000
00001
00002
01000
主
子1
子2
02200
物理地址 内存

3、重定位的概念

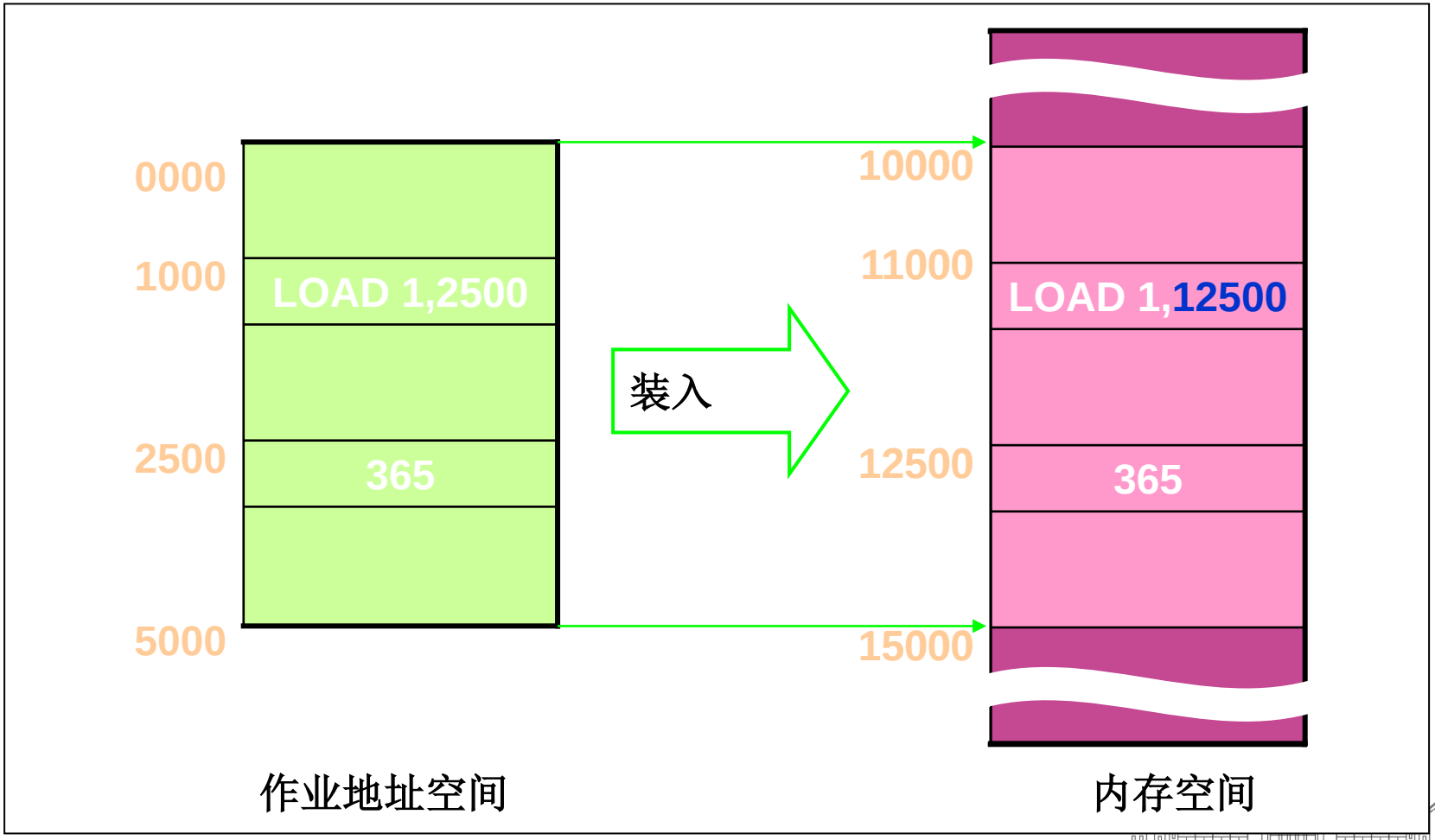
在装入时对目标程序中指令和数据的修改过程称为重定位。即逻辑地址变换为物理地址的过程。

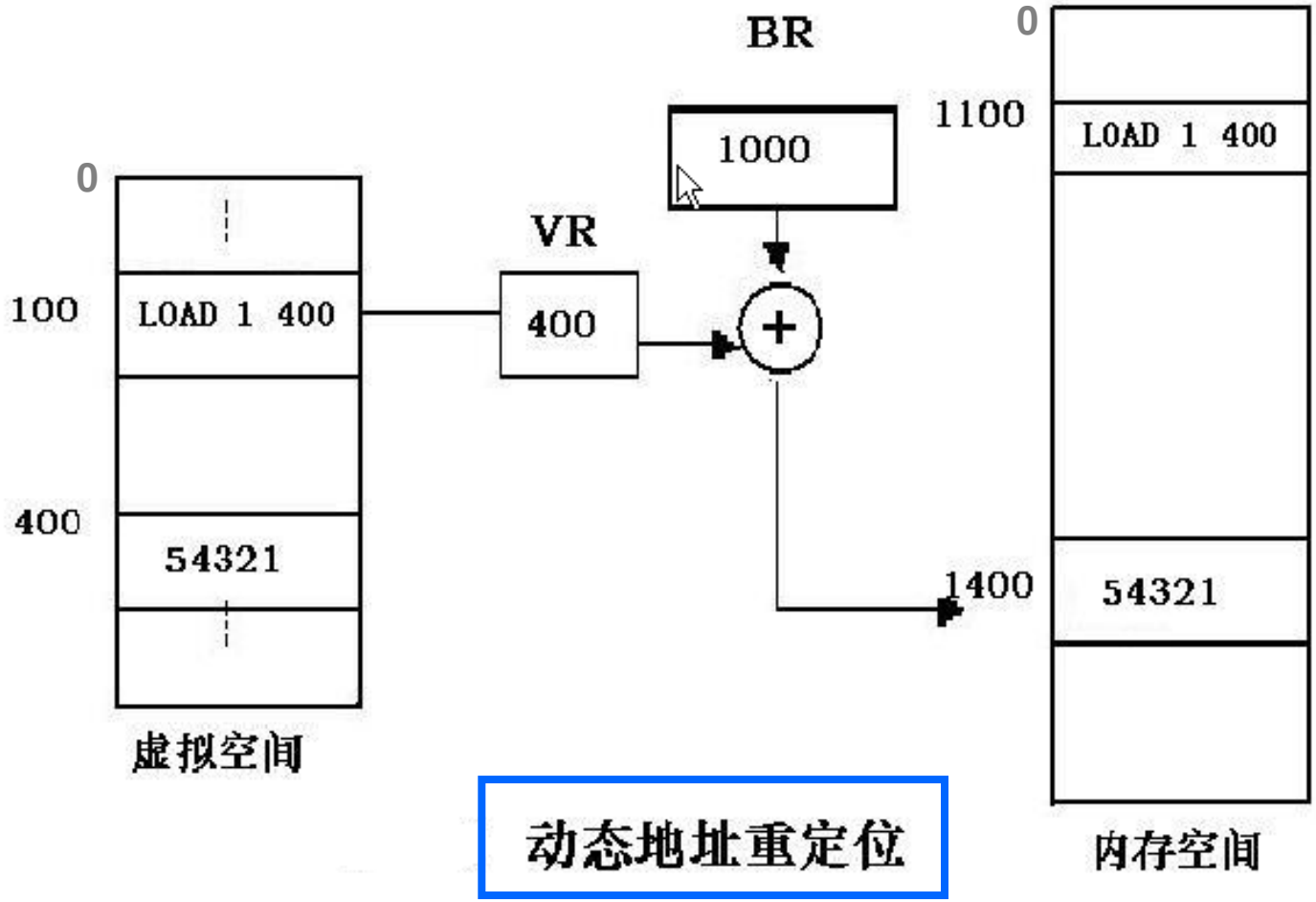
重定位的类型

静态重定位：地址变换是在装入时一次完成，以后不再改变。

动态重定位：地址变换是在程序指令执行时进的。

3、重定位的概念


0000
1000
2500
5000
LOAD 1,2500
365
装入
10000
11000
12500
15000
LOAD 1,12500
365
作业地址空间
内存空间



BR： 重定位寄存器　　**VR：** 变址寄存器

4、程序的链接

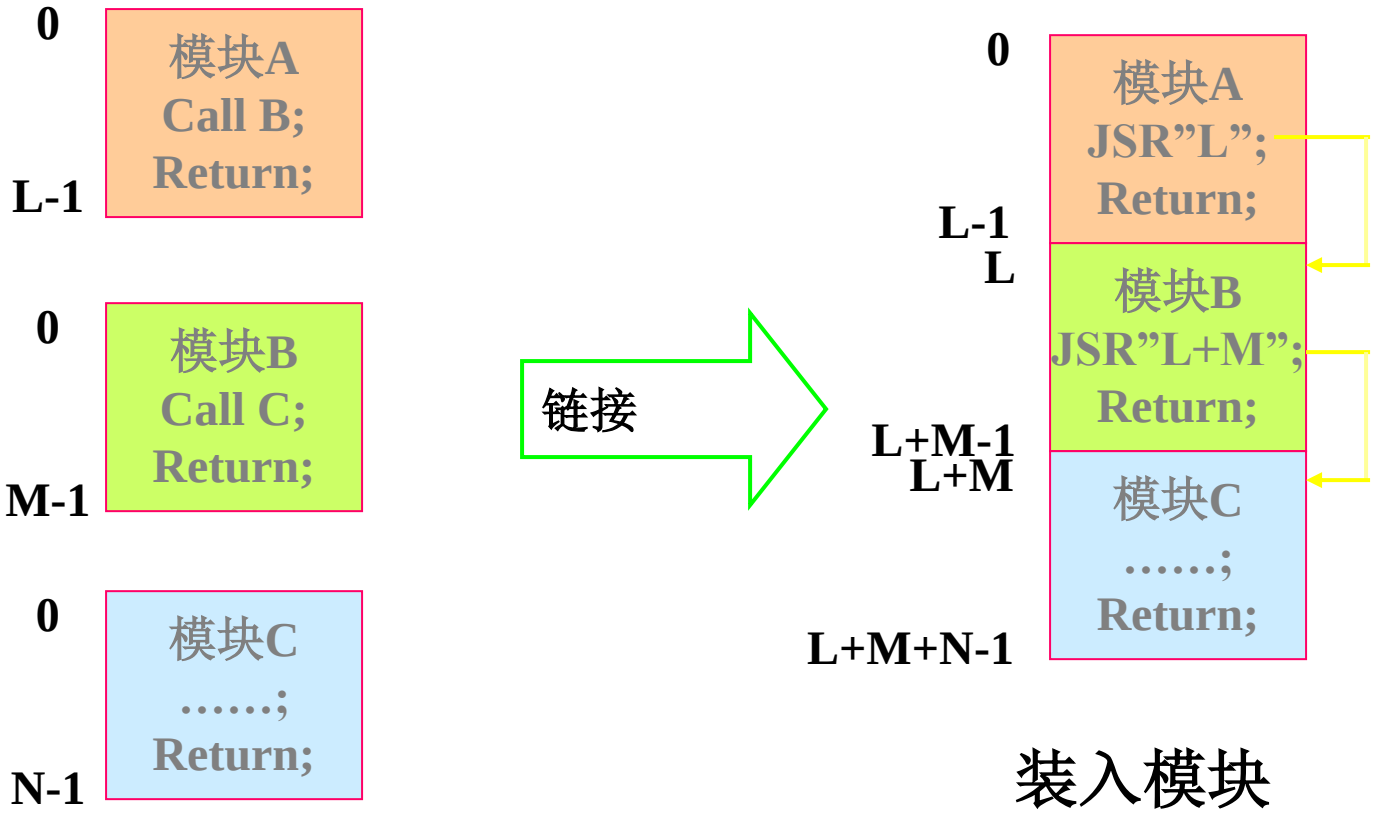
把程序相关的一组目标模块和系统调用模块（库函数）链接形成一个整体——装入模块的过程。

具体工作：对相对地址的修改；变换外部调用符号。

链接方式：

- **静态链接**
- **装入时动态链接**
- **运行时动态链接**

4、程序的链接


0
模块A
Call B;
Return;
L-1
0
模块B
Call C;
Return;
M-1
0
模块C
……;
Return;
N-1
链接
0
模块A
JSR"L";
Return;
L-1
L
模块B
JSR"L+M";
Return;
L+M-1
L+M
模块C
……;
Return;
L+M+N-1
装入模块

5、程序的装入

含义：把链接好的装入模块装入“内存”。

装入方式：

- 绝对装入
- 可重定位装入（静态重定位）
- 动态运行时装入（动态重定位）

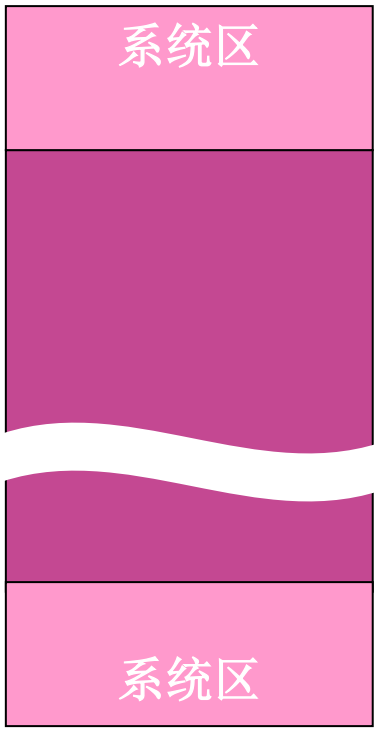
提示：通常链接、装入程序是一体的。

4.3 连续分配存储管理方式

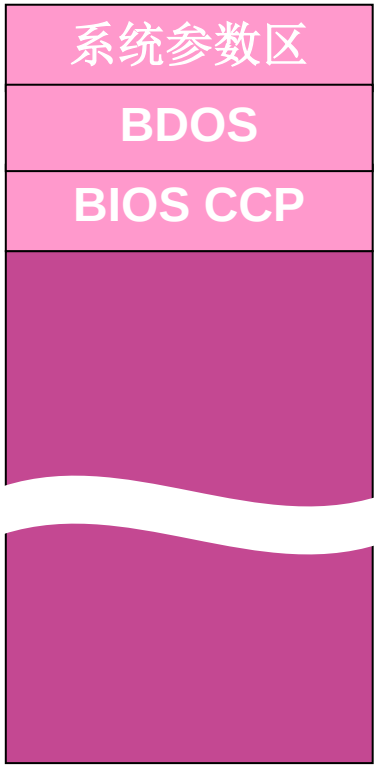
为用户程序分配一个连续的内存空间

- 单一连续分配
- 固定分区分配
- 动态分区分配
  - 基于顺序搜索的动态分区分配算法
  - 基于索引搜索的动态分区分配算法
- 动态可重定位分区分配

4.3.1 单一连续分配



DOS内存分区



CP/M内存分配

4.3.2 固定分区分配

最简单、可运行多道程序的存储管理方式。

1、划分分区的方法

分区大小相等： 缺乏灵活性，控制多个相同对象的系统

分区大小不等：多个小分区、适量中等分区、少量大分区

2、内存分配管理

建立分区使用表——起址、大小、状态

将分区按大小排队；装入时，分配程序检索分区使用表，找到符合要求的分区，并进行标记。

作业1进入，大小30K　作业2进入，大小500K
作业3进入，大小8K

分区号	大小	起址	状态
1	8 K	512K	已使用
2	32 K	520K	已使用
3	32 K	552K	未使用
4	128 K	584K	未使用
5	512 K	712K	已使用
…	…	…	…

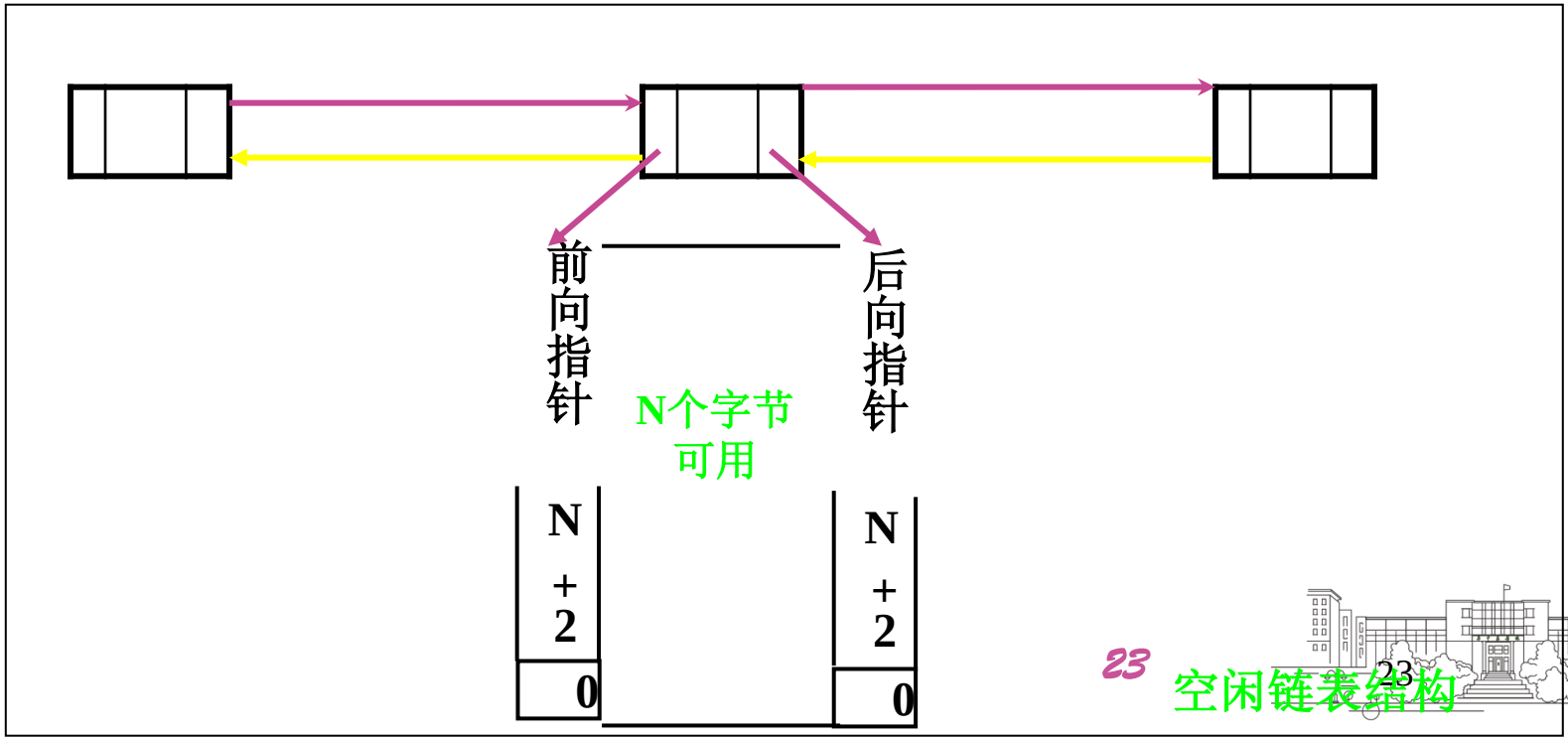
4.3.3、动态分区分配

根据进程的实际需要，动态分配内存空间

1、内存管理方式（数据结构）：

空闲分区表——序号、起址、大小等项

空闲分区链——双向链表



空闲链表结构

2、动态分区分配算法（4.3.4—4.3.5）

4.3.4基于顺序搜索的动态分区分配算法

首次适应算法： 空闲分区按起址递增次序排列，从头开始直至找到第一个满足要求的空闲分区。

特点：内存低端会留下小的空闲区，高端有大的空闲区；

循环首次应算法：从上次分配的位置之后开始查找。

特点：使内存的空闲分区均匀，但缺乏大的空闲分区；

最佳适应算法：空闲分区按大小递增的次序排列，从头开始找到第一个满足要求的空闲分区。

缺点：会留下大量小碎片。

最坏适应算法：空闲分区按大小递减的次序排列，最前面的最大的空闲分区就是找到的分区。

优点：分配后剩下的可用空间比较大

缺点：一段时间后就不能满足对于较大空闲区的分配要求。

4.3.5基于索引搜索的动态分区分配算法

1、快速适应算法：

- 空闲分区按容量大小分类，对每类具有相同容量的所有空闲分区，单独设立一个空闲分区链表。
- 在内存中设立一张管理索引表，每个表项对应一种空闲分区类型。

优点：查找效率高。保留大分区也不会产生碎片

缺点：分区归还主存时算法复杂。

2、伙伴系统：

分区（已分配和空闲）大小均为2的k次幂（1<=k<=m）,2^m 为可分配内存大小。

对不连续的空闲分区，按分区大小进行分类。对具有相同大小的所有空闲分区，单独设立一个空闲分区双向链表，即会存在k个空闲分区链表。

2、伙伴系统：

分配时，设需分配长度为n，找2^i分区链的分区，使$2^{i-1}<n<2^i$ 若无，找2^{i+1}且把它均分两块，称为伙伴。一个加入2^i分区链，一个分配；.....

回收时，若已存在2^i空闲分区，则将其于伙伴合并为2^{i+1}分区，.....

- 特点：性能取决于查找空闲分区的位置和分割、合并的时间。 时间上不及快速适应算法，但空闲分区的使用率高

回收算法 考虑互为伙伴的空闲区的归并

在伙伴系统中，对于一个大小为2^k，地址为x的内存块，其伙伴块的地址则用$buddy_k(x)$表示，其通式为：

$$buddy_k(x)=\begin{cases} x+2^k \text{(若 } x \text{MOD } 2^{k+1}=0) \\ x-2^k \text{(若 } x \text{MOD } 2^{k+1}=2^k) \end{cases}$$

3、哈希算法：

建立哈希函数，构造一张一空闲分区大小为关键字的哈希表，该表的每一个表项记录一个对应的空闲分区链表。

分配时，根据所需空闲分区大小，通过哈希函数计算，即得到在哈希表中的位置，再分配

利用哈希快速查找优点，以及空闲分区在可利用空闲区表中的分布规律，

3、分区分配操作（分配算法流程）

分配内存

表中空闲分区大小　　请求的分区大小

从空闲分区链（表）中找所需大小的分区。

判断条件：**M.Size - U.Size ≦ Size**　　下限值

剩余部分挂接到空闲分区链（表）上。

回收内存

回收区与插入点的前一个空闲分区相邻接；

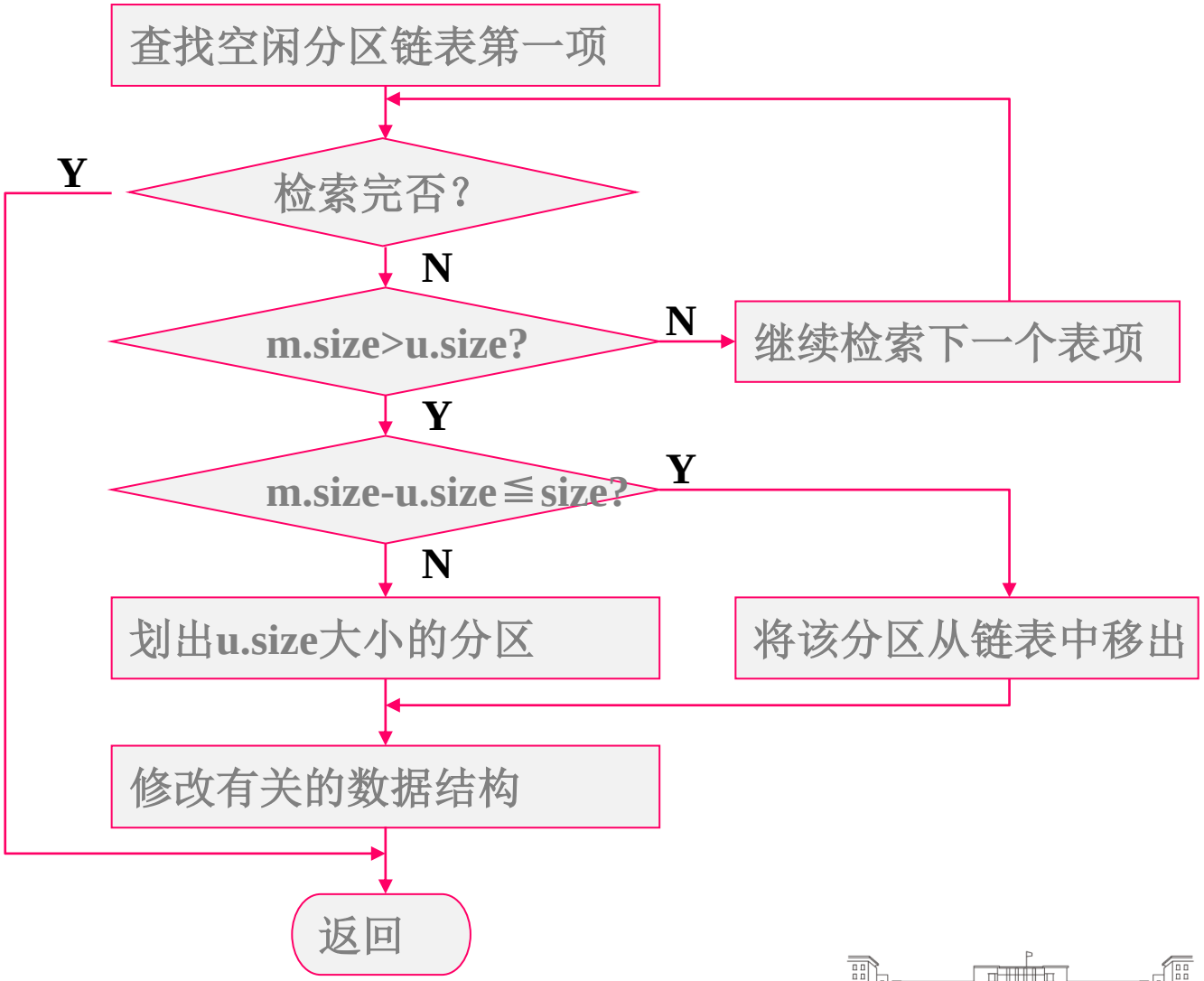
回收区与插入点的后一个空闲分区相邻接；

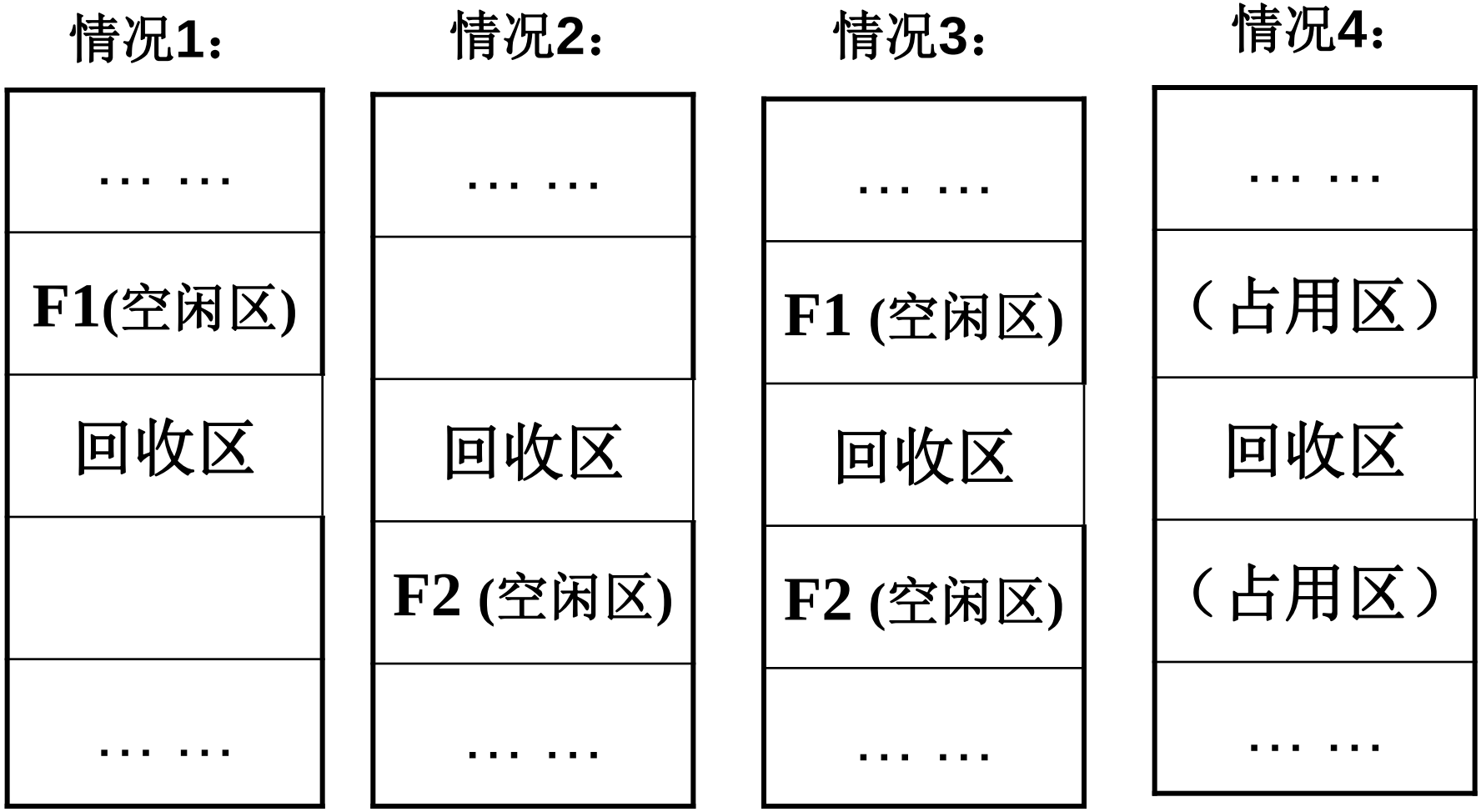
回收区与插入点的前后两个空闲分区相邻接；

回收区不与任何一个空闲分区相邻接；

优缺点： 管理复杂，总会有闲置的小分区——“碎片”。

内存分配流程


查找空闲分区链表第一项
检索完否?
Y
N
m.size>u.size?
N
继续检索下一个表项
Y
m.size-u.size≦size?
Y
N
划出u.size大小的分区
将该分区从链表中移出
修改有关的数据结构
返回


情况1：
… …
F1(空闲区)
回收区
… …
情况2：
… …
回收区
F2 (空闲区)
… …
情况3：
… …
F1 (空闲区)
回收区
F2 (空闲区)
… …
情况4：
… …
（占用区）
回收区
（占用区）
… …

内存回收时的情况

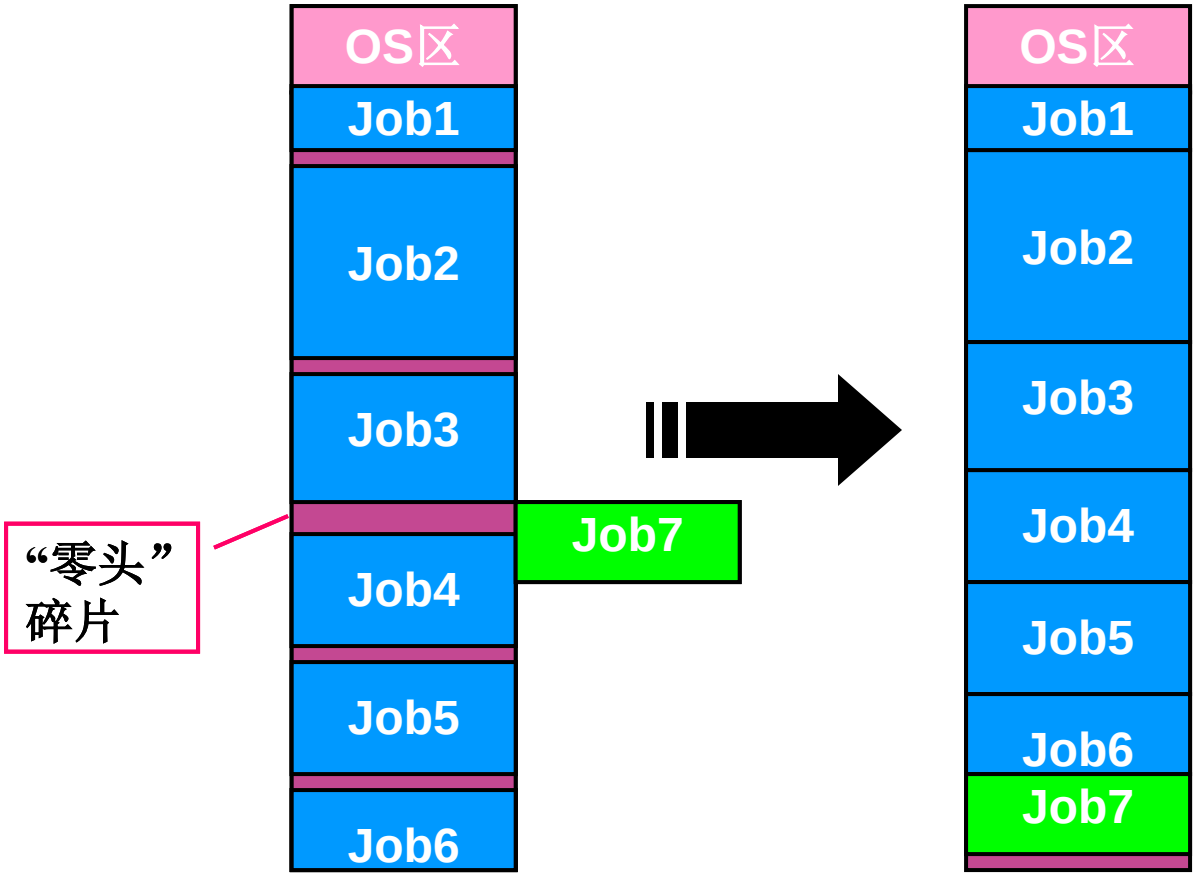
4.3.6、可重定位分区分配

1、动态重定位的引入

随系统接收作业增加，内存中连续大块分区不复存在，产生了大量的“碎片”。

新作业无法装入到“碎片”小分区上运行，但所有碎片总和可能大于需求。

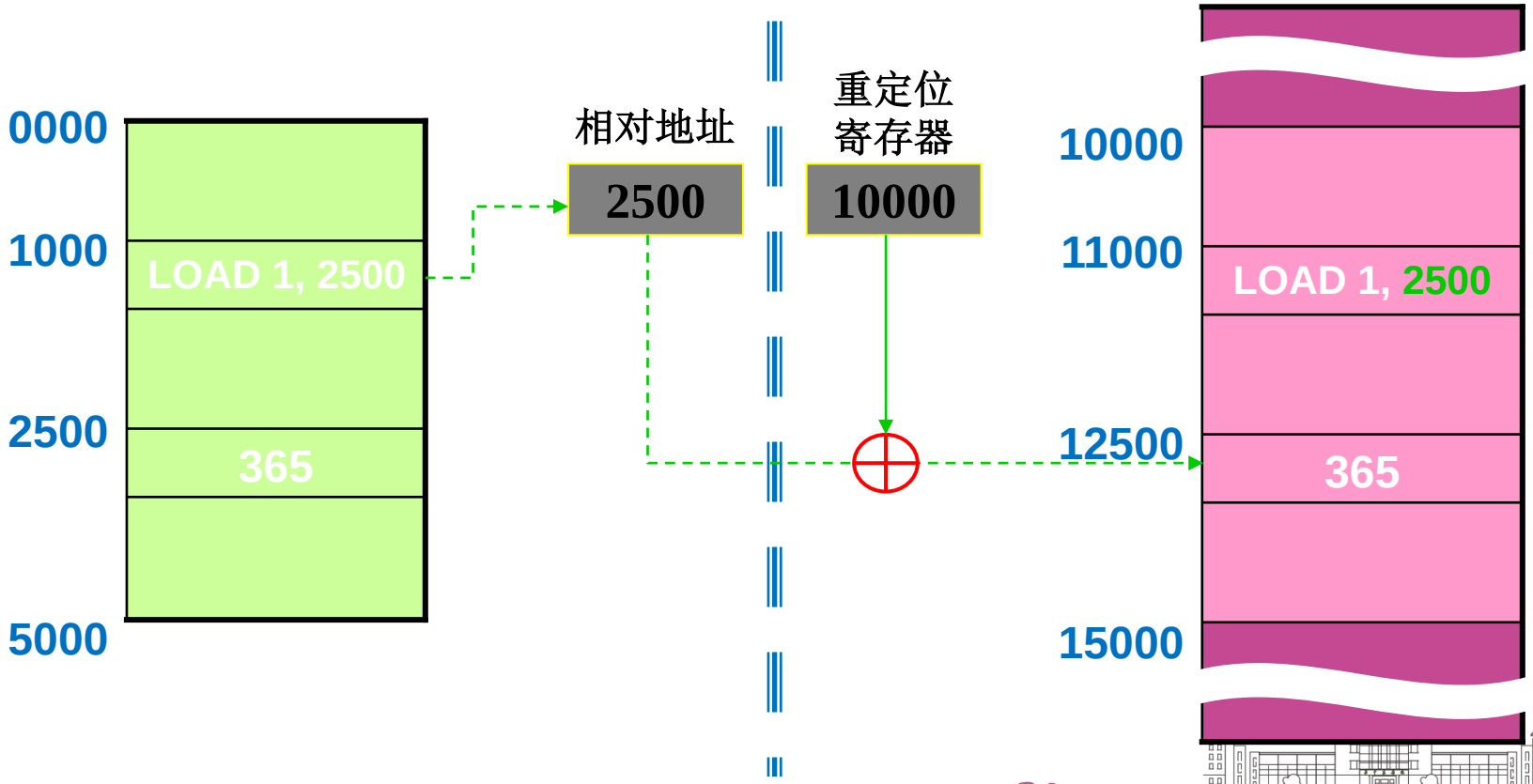
通过“拼接”或“紧凑” 来实现程序的浮动（动态重定位）。


OS区
Job1
Job2
Job3
Job7
Job4
Job5
Job6
“零头”碎片
OS区
Job1
Job2
Job3
Job4
Job5
Job6
Job7

2、动态重定位的实现

须由硬件地址变换机构支持

重定位寄存器：存放程序在内存中的起始地址。



3、可重定位分区分配算法

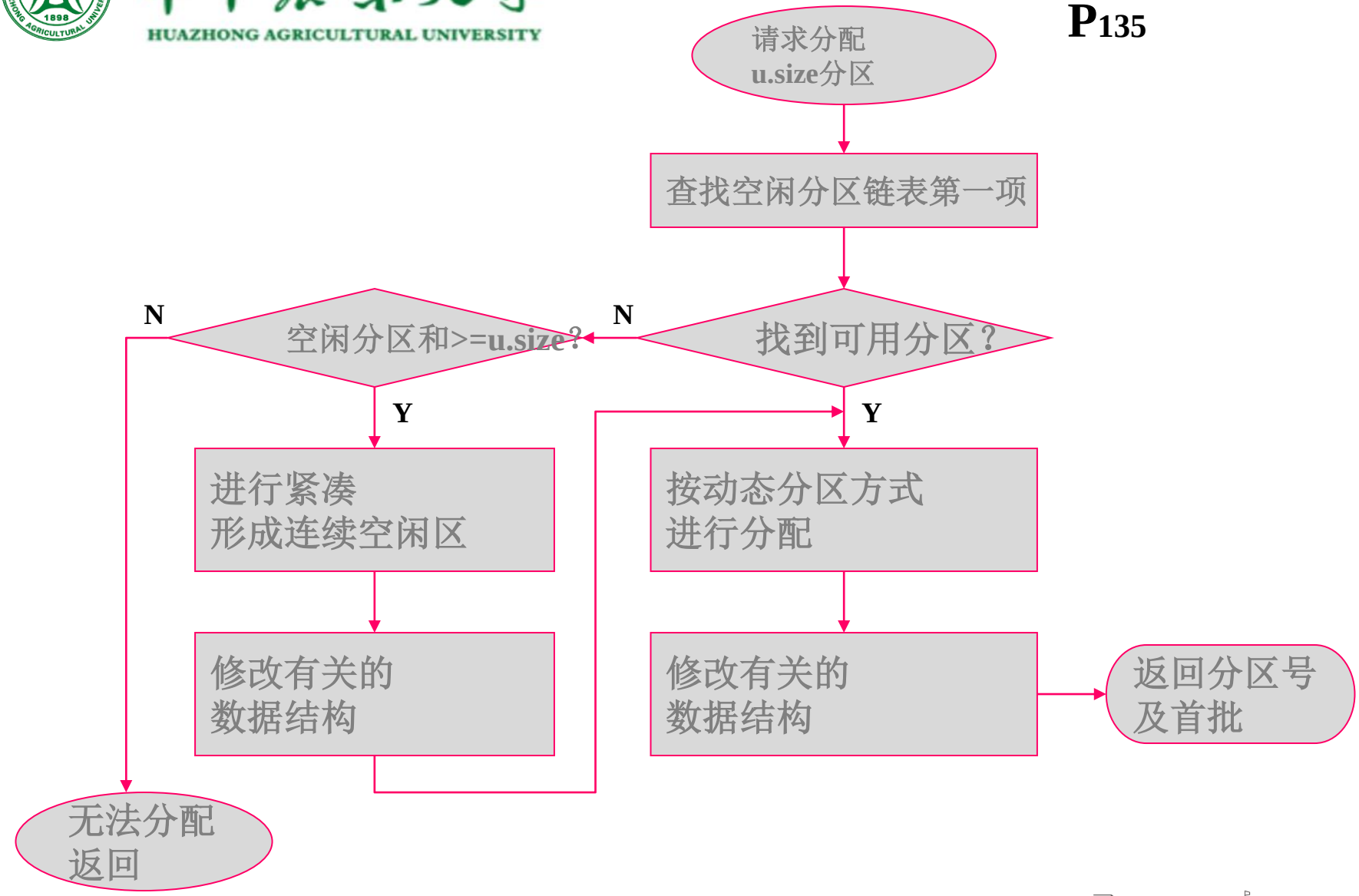
与动态分区分配算法基本相同，
但增加了紧凑功能

优缺点分析

优点：消除了“碎片”，提高了内存利用率，同时提高了系统效率。

缺点：需要动态重定位“硬件”机构支持，增加了系统成本，并轻度降低了程序执行速度，“紧凑”处理增加了系统开销。

P135


请求分配
u.size分区
查找空闲分区链表第一项
找到可用分区？
N
空闲分区和>=u.size?
N
Y
Y
进行紧凑
形成连续空闲区
按动态分区方式
进行分配
修改有关的
数据结构
修改有关的
数据结构
返回分区号
及首批
无法分配
返回

4.4、对换（Swapping） P135

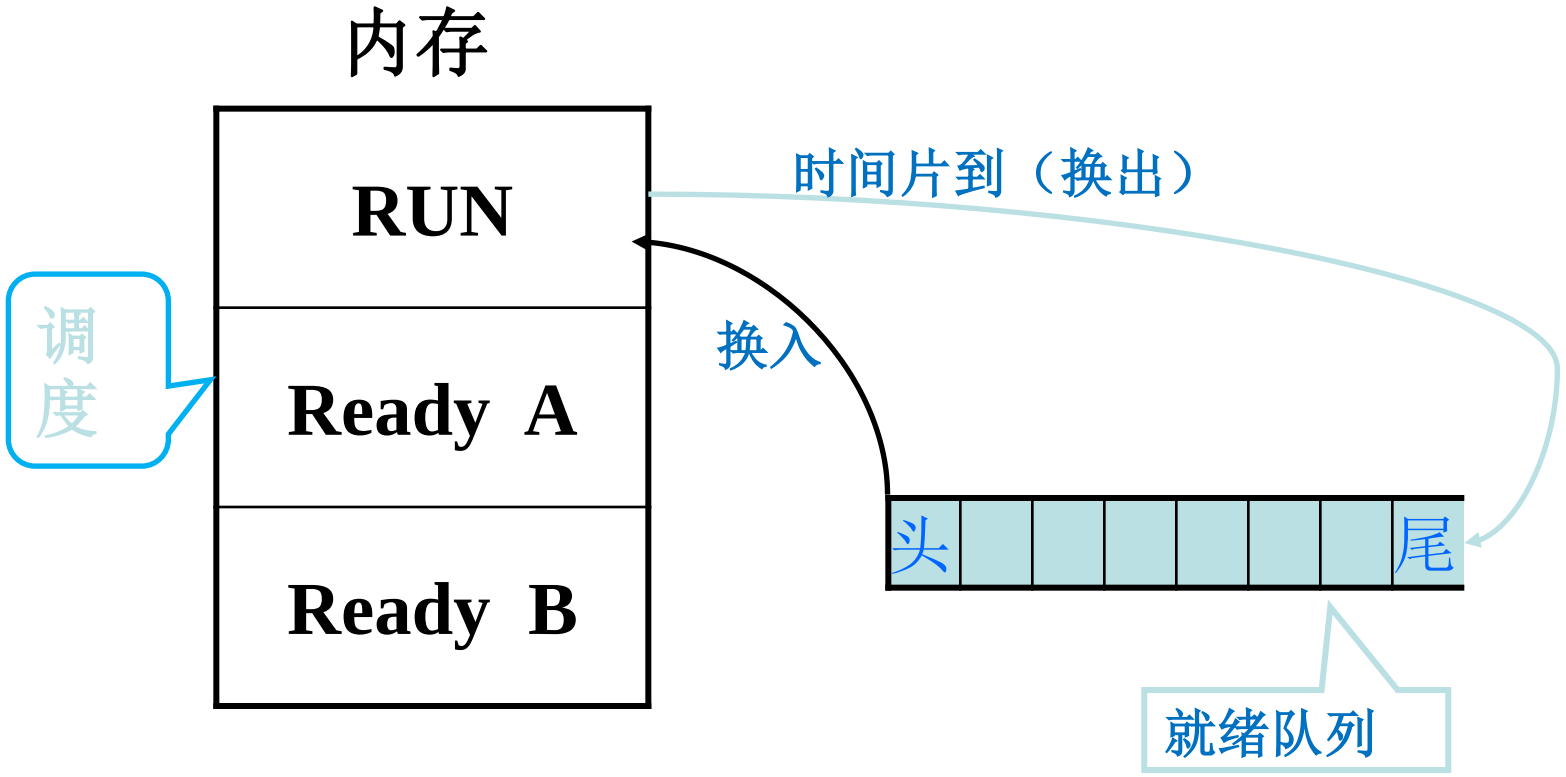
- 1、对换的引入

  解决内存不足的问题；

  - 定义
  - **对换类型：整体对换：**以进程为单位的对换

    部分对换：以“页”或“段”为单位的对换
- 2、对换空间的管理
  - **外存划分：**文件区、对换区
  - **管理方式：**空闲分区表、空闲分区链
  - **分配算法：**首次适应法、循环首次适应法、最佳适应法

例： 在分时系统中，一台主机，多台终端，每个用户得到的内存有限，因此可利用外存作为补充。



3、进程的换出与换入

进程的换出

选择处于阻塞状态且优先级最低的进程

将该进程的程序和数据传送道磁盘的对换区上

回收内存空间，修改该进程的PCB

进程的换入

定时查看进程状态

将处于就绪态的换出时间最久的进程换入内存

4.5 基本分页存储管理方式 P138

连续分配方式不足，产生了离散分配思想。

引入了“分页”分配管理方式。分为：

基本分页（纯分页）和支持虚存管理请求分页管理。

页面与页表

地址变换机构

两级和多级页表

基本分页的特点

4.5.1、页面与页表

- 1、页面
  - 页面和物理块（页框/架）：顺序编号，页内碎片
  - 页面大小：2^n Bytes 一般在：512B ~ 8/32K
- 2、地址结构

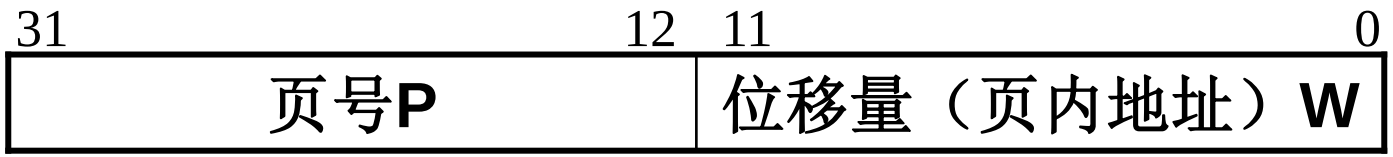
逻辑地址

页号P	位移量W

物理地址

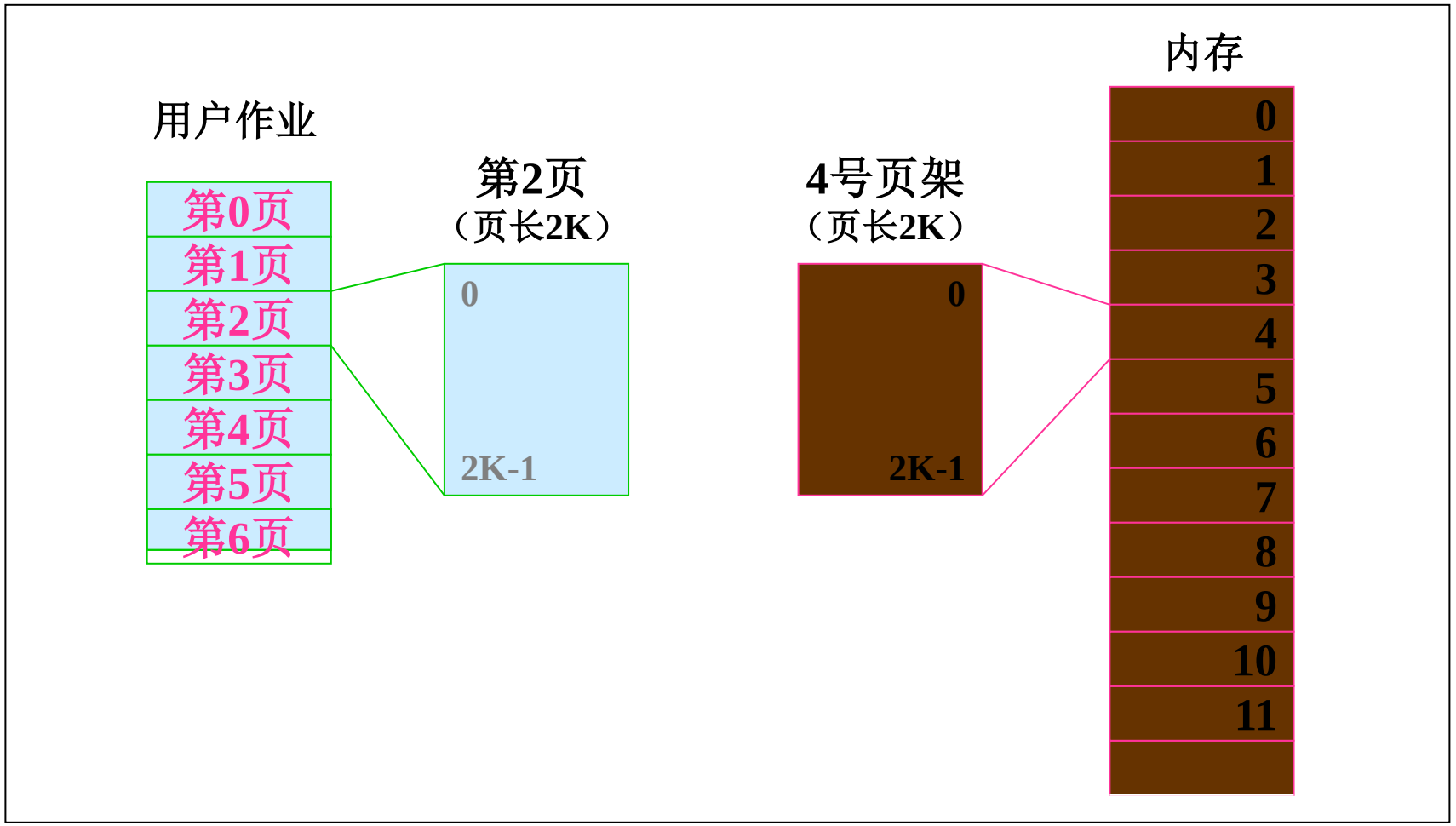
物理块号P	页内地址d

例：



每页大小为4KB，地址空间最多允许有1M页

4.5.1、页面与页表


用户作业
第0页
第1页
第2页
第3页
第4页
第5页
第6页
第2页
（页长2K）
0
2K-1
4号页架
（页长2K）
0
2K-1
内存
0
1
2
3
4
5
6
7
8
9
10
11

页号P和页内地址d的计算公式

P＝INT［A/L］　　INT：整除函数

d＝[A] MOD L　　MOD：取余函数

（A：逻辑地址空间中的地址，L：页面大小）

例：某系统的页面大小为1KB，地址A=2170B，则求得P=2，d=122

3、页表——页面映像表

数据结构：页号、块号、存取控制项

页表作用：实现从页号到物理块号的地址映射。

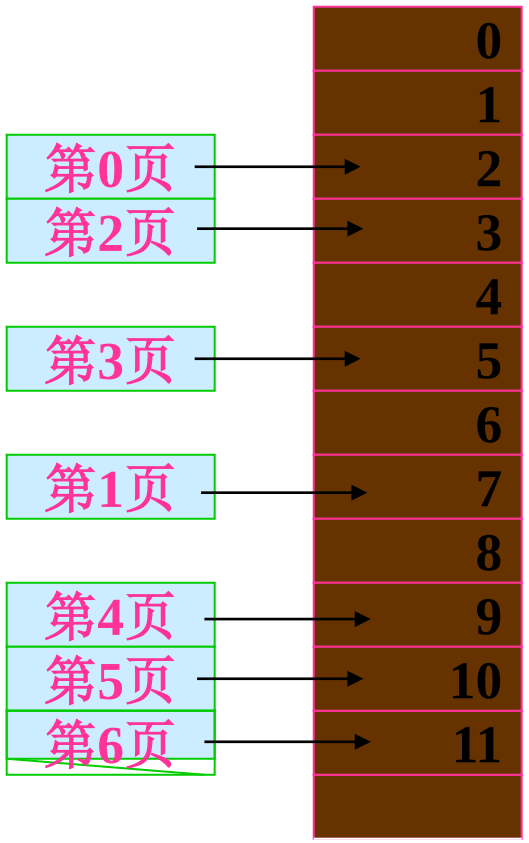
用户作业

第0页
第1页
第2页
第3页
第4页
第5页
第6页

页表

页号	块号
0	2
1	7
2	3
3	5
4	9
5	10
6	11

内存



0000,0000,0000,0000,0101 ,0001,0000,0000

第3页256字节的物理地址（页大小：4K）

4.5.2、地址变换机构 P140

1、基本的地址变换机构

任务：实现地址映射，即从逻辑地址到物理地址的变换过程。

页表存放在内存系统区的一个连续空间中；

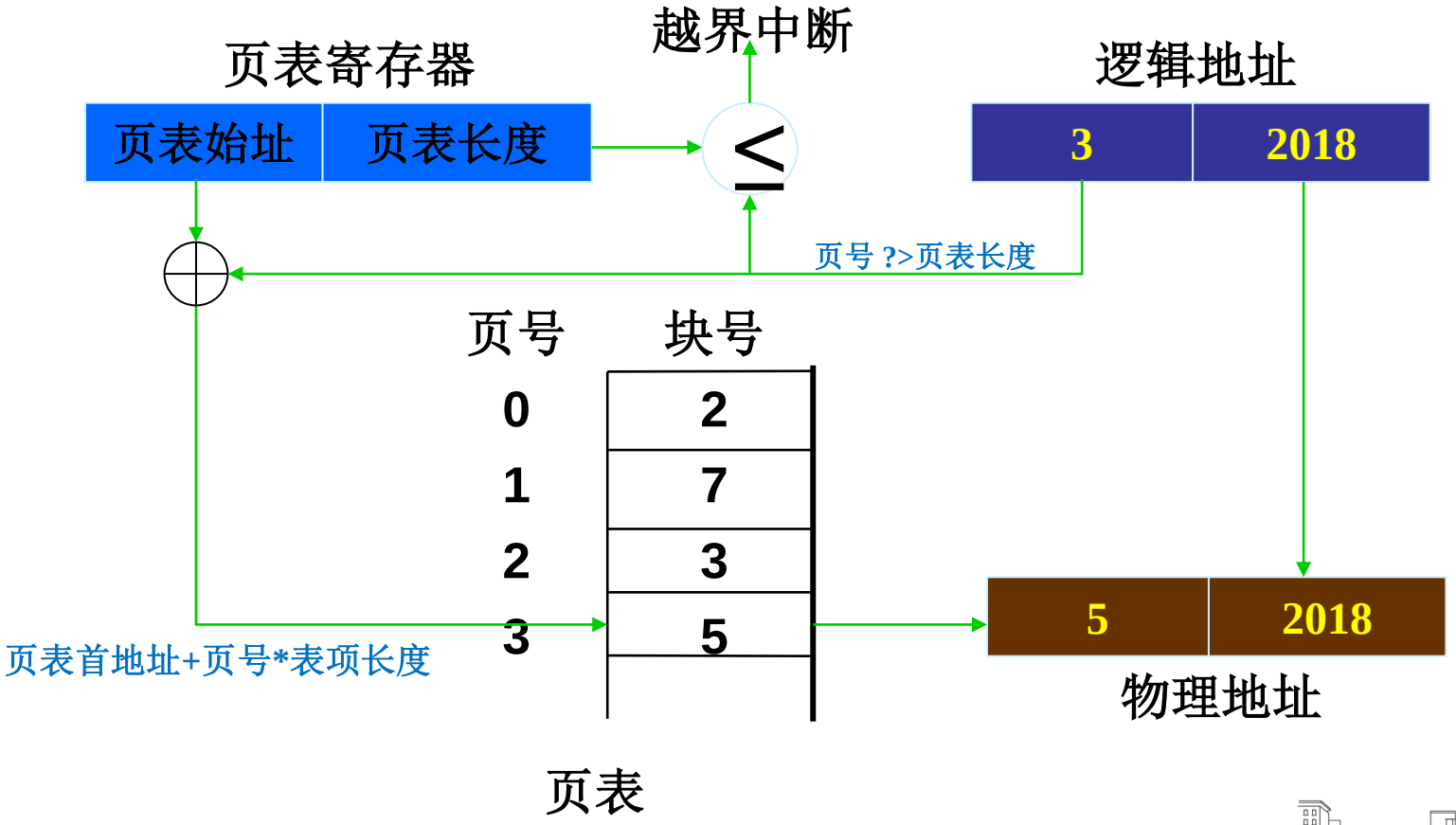
PCB和页表寄存器PTR中存有页表在内存的首地址和页表长度；

地址映射过程：

自动的将逻辑地址分为页号和页内地址

根据页号查询页表：页表首地址+页号*表项长度；

- 找到该页对应的物理块号，装入物理地址寄存器
- 将页内地址送入物理地址寄存器。 两次访问内存



- **2、具有快表的地址变换机构**

快表（联想寄存器）:具有并行查询能力的高速缓冲寄存器

空间大小：几K到几百K ，只含有部分页表项（16~512个）

快表与页表同时访问；

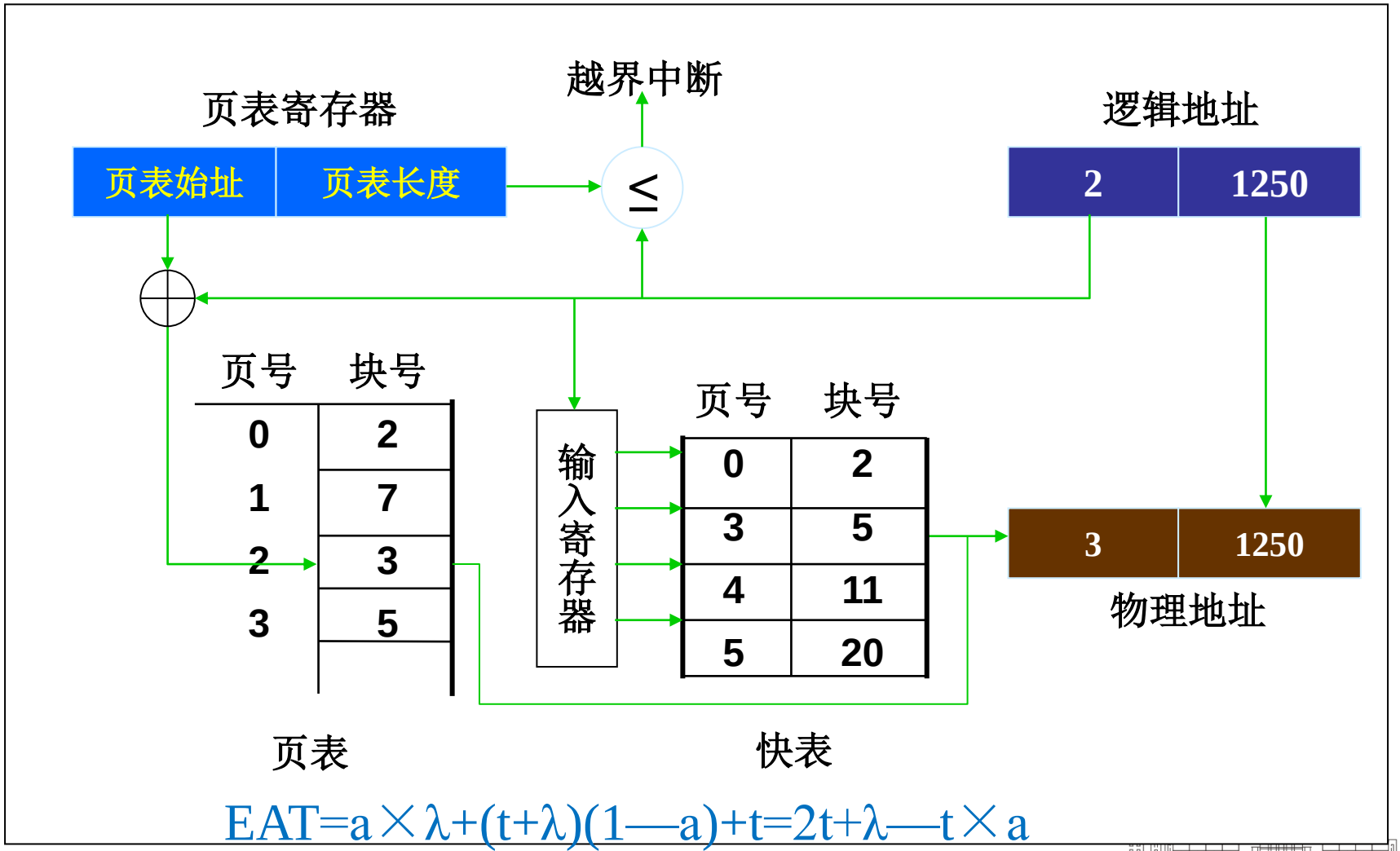
- **地址映射过程：**

将页号P送入快表，若有此页号，则读出该页对应的物理块号；若无，则访问页表

将物理块号送入地址寄存器，并将此页表项存入快表。若快表已满，则换出一个不再用的页表项

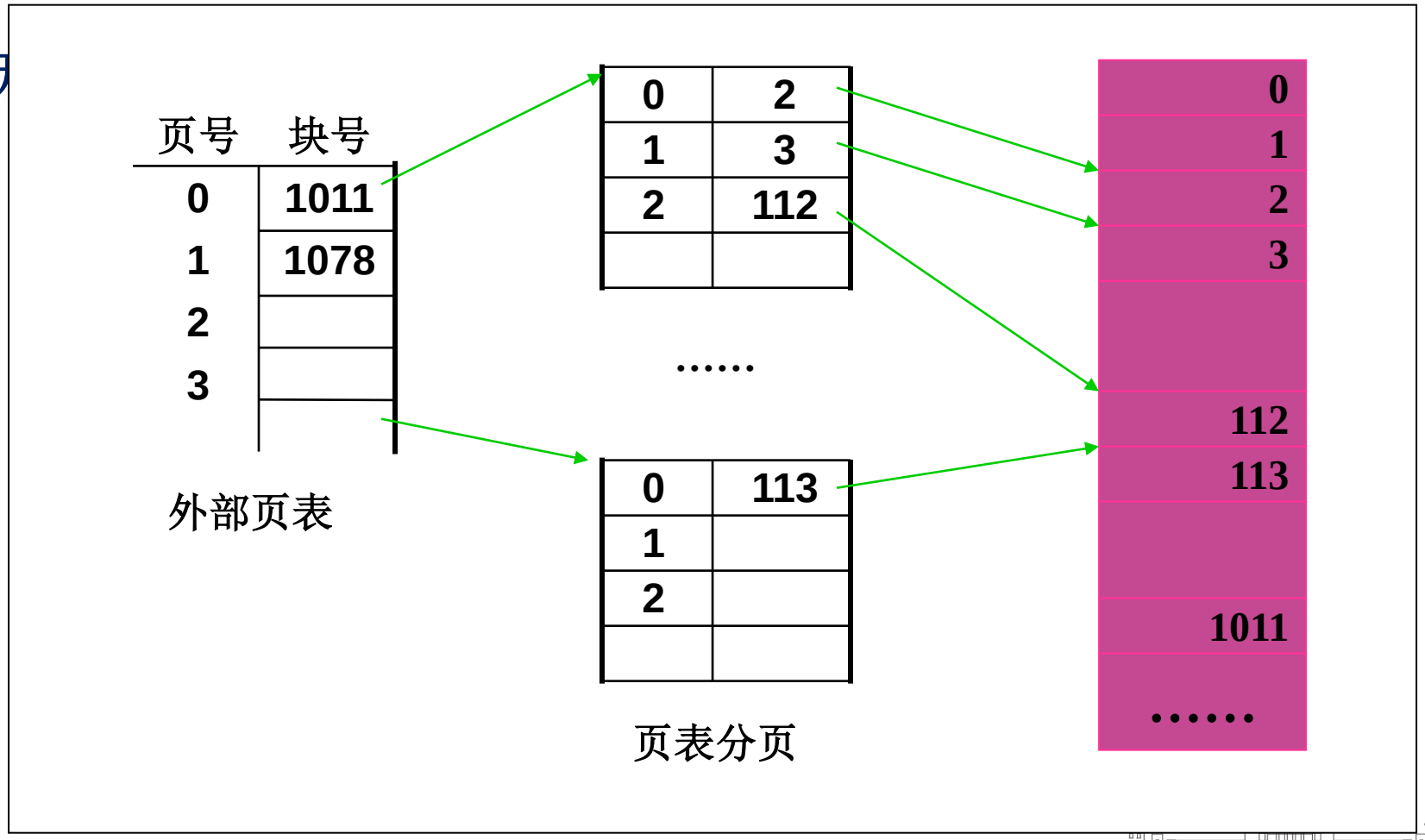
P141

2、具有快表的地址变换机构

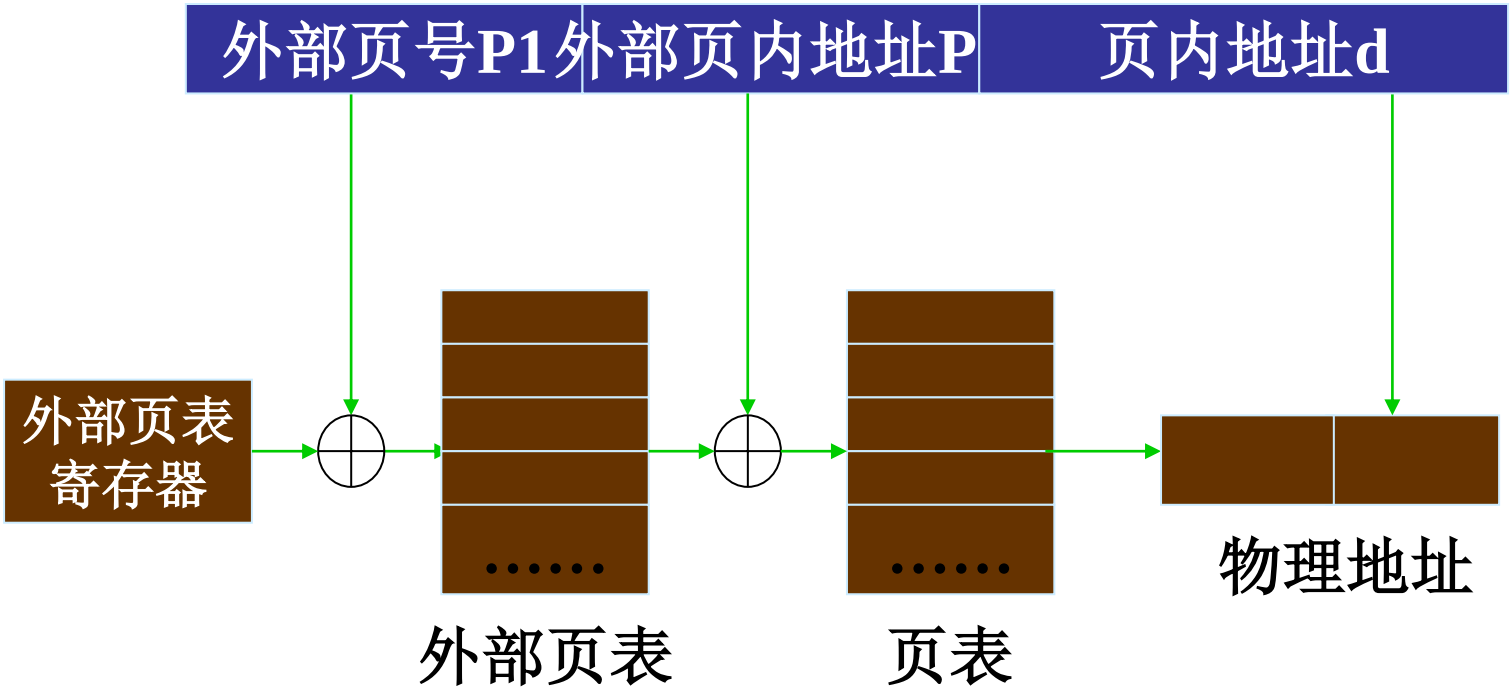


$$EAT=a\times\lambda+(t+\lambda)(1-a)+t=2t+\lambda-t\times a$$

4.5.3、两级和多级页表


页号
块号
0
1011
1
1078
2
3
外部页表
0
2
1
3
2
112
……
0
113
1
2
页表分页
0
1
2
3
112
113
1011
……

逻辑地址



具有两级页表的地址变换机构

基本分页的特点：

优点：

存在页内碎片，但碎片相对较小，内存利用率较高;实现了离散分配，消除了程序浮动;

缺点：

需要专门的硬件支持，尤其“快表”；

内存访问的效率下降。

4.6 分段存储管理方式

- 分段管理思想的引入
- 基本原理
- 地址变换
- 分段与分页的主要区别
- 段页式存储管理方式

4.6.1、分段管理思想的引入

P145

主要为了满足用户和程序员的下述需要：

方便编程：　　LOAD 1,[A]|<D>;

信息共享：共享的实现以是信息的逻辑单位为基础

信息保护：同样是对逻辑单位进行保护

动态增长：如数据段

动态链接：运行时调入内存并链接

4.6.2、基本原理

1、分段

作业（逻辑地址）空间分段，每个段都有名字：

主程序段、子程序段、数据段、栈段等

逻辑地址：二维地址（段号，段内地址）每个段装入内存中的一个连续的内存空间

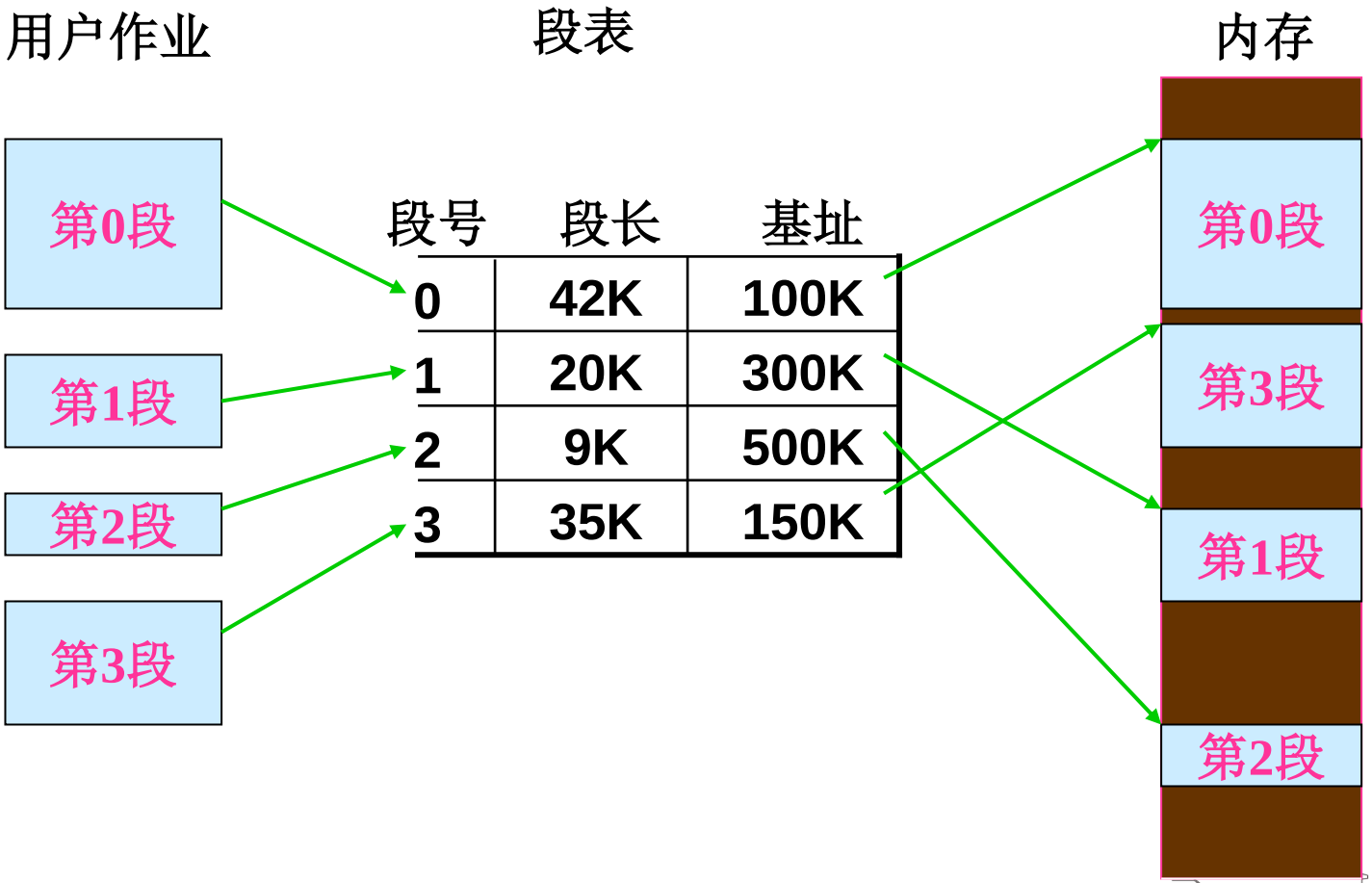
2、段表

段号、段基址、段长度等;每个段在段表中占一个表项

段表寄存器：存放段表的起址和长度

利用段表实现地址映射

4.6.2、基本原理


用户作业
段表
内存
第0段
第1段
第2段
第3段
段号
段长
基址
0
42K
100K
1
20K
300K
2
9K
500K
3
35K
150K
第0段
第3段
第1段
第2段

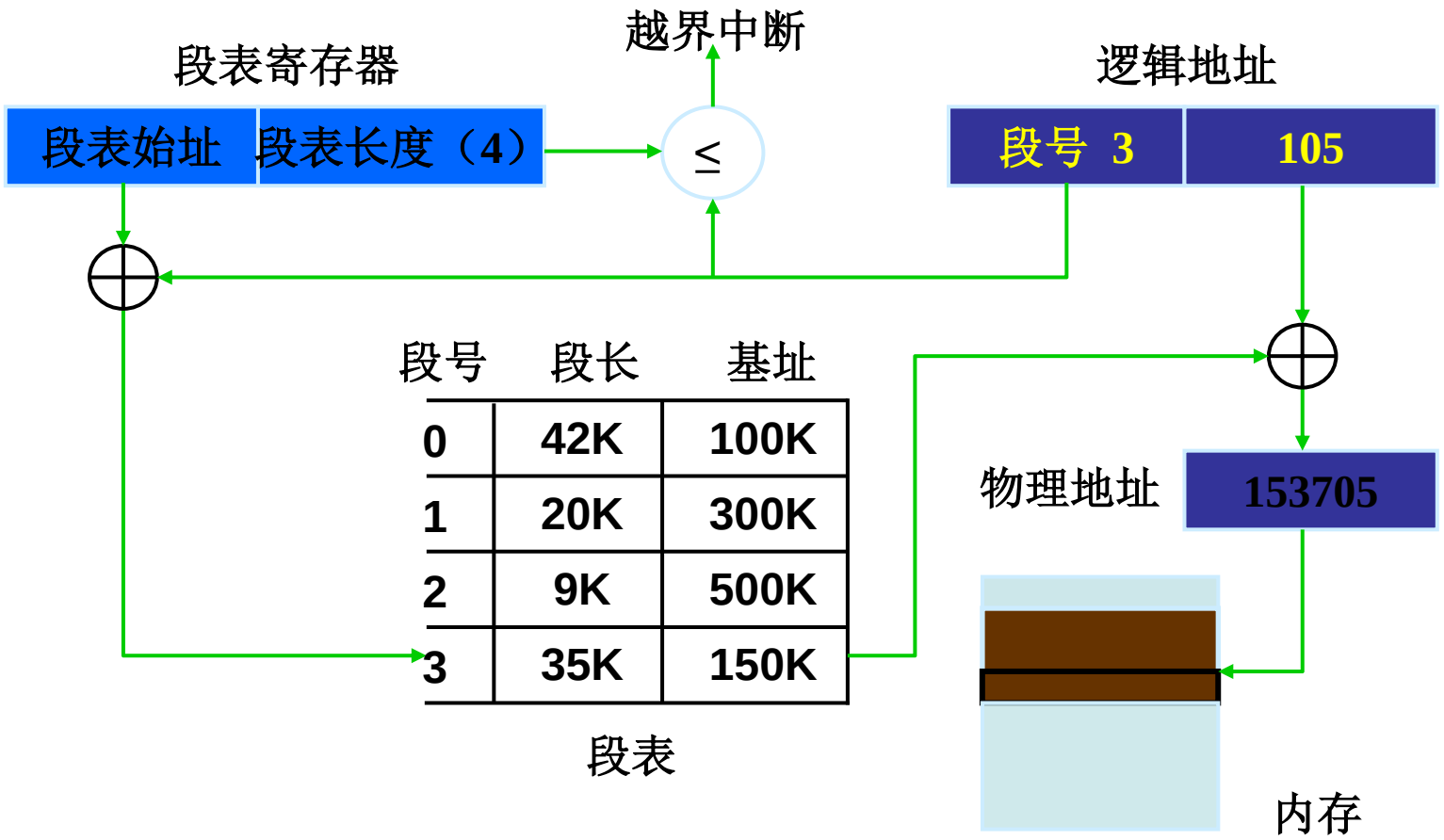
3、基本分段管理的地址变换

与基本分页管理的变换机构和过程类似。

段表寄存器

存放段表的起始地址和段表长度；

- **越界访问控制**
  - *逻辑地址的段号与段表长度比较；*
  - *段内地址与段表中保存的段长比较；*



150*1024=153600

153600+105=153705

4、分段与分页的主要区别

页是信息的物理单位，

段是信息的逻辑单位；

页的大小固定，

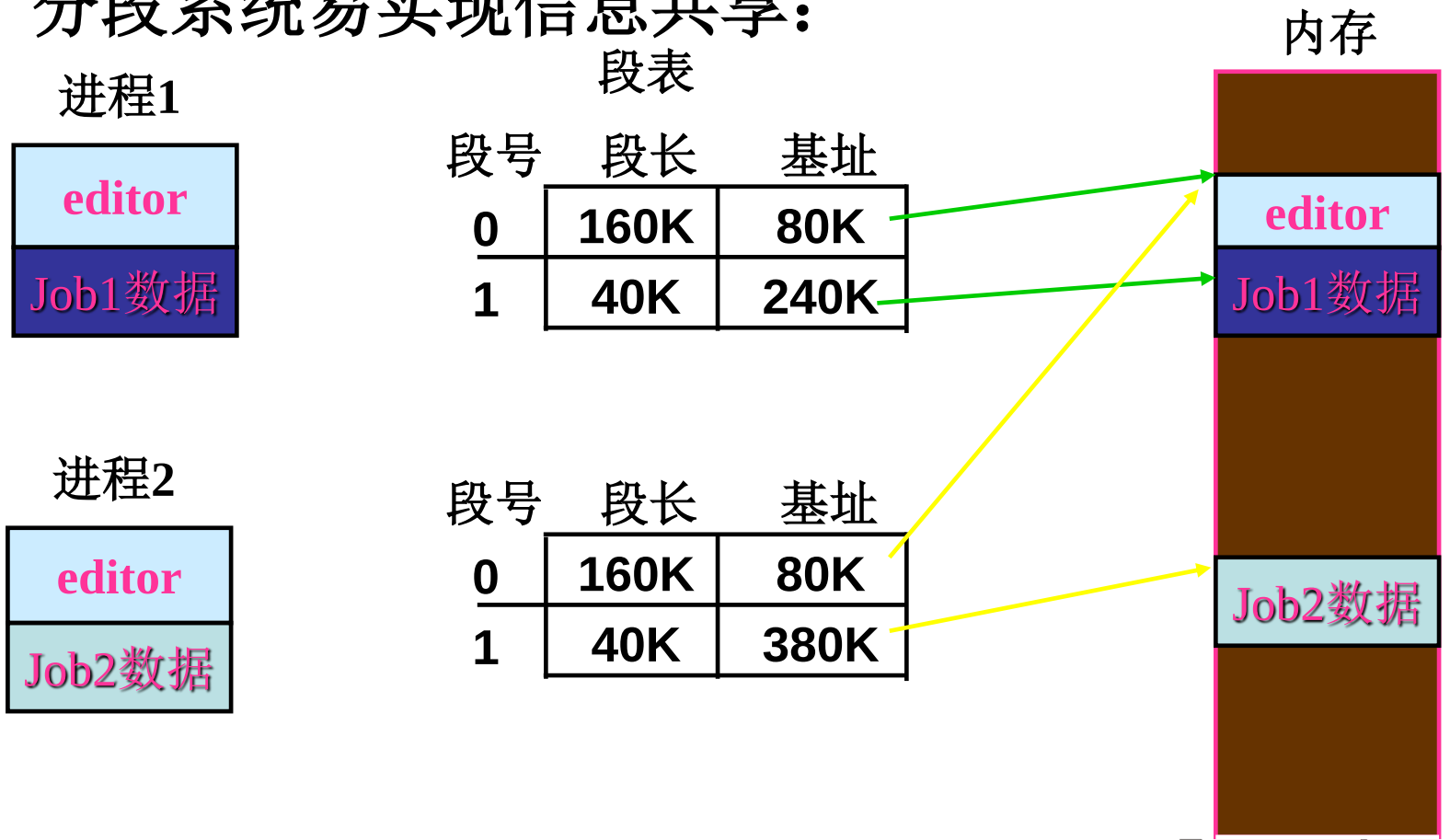
段的大小动态变化；

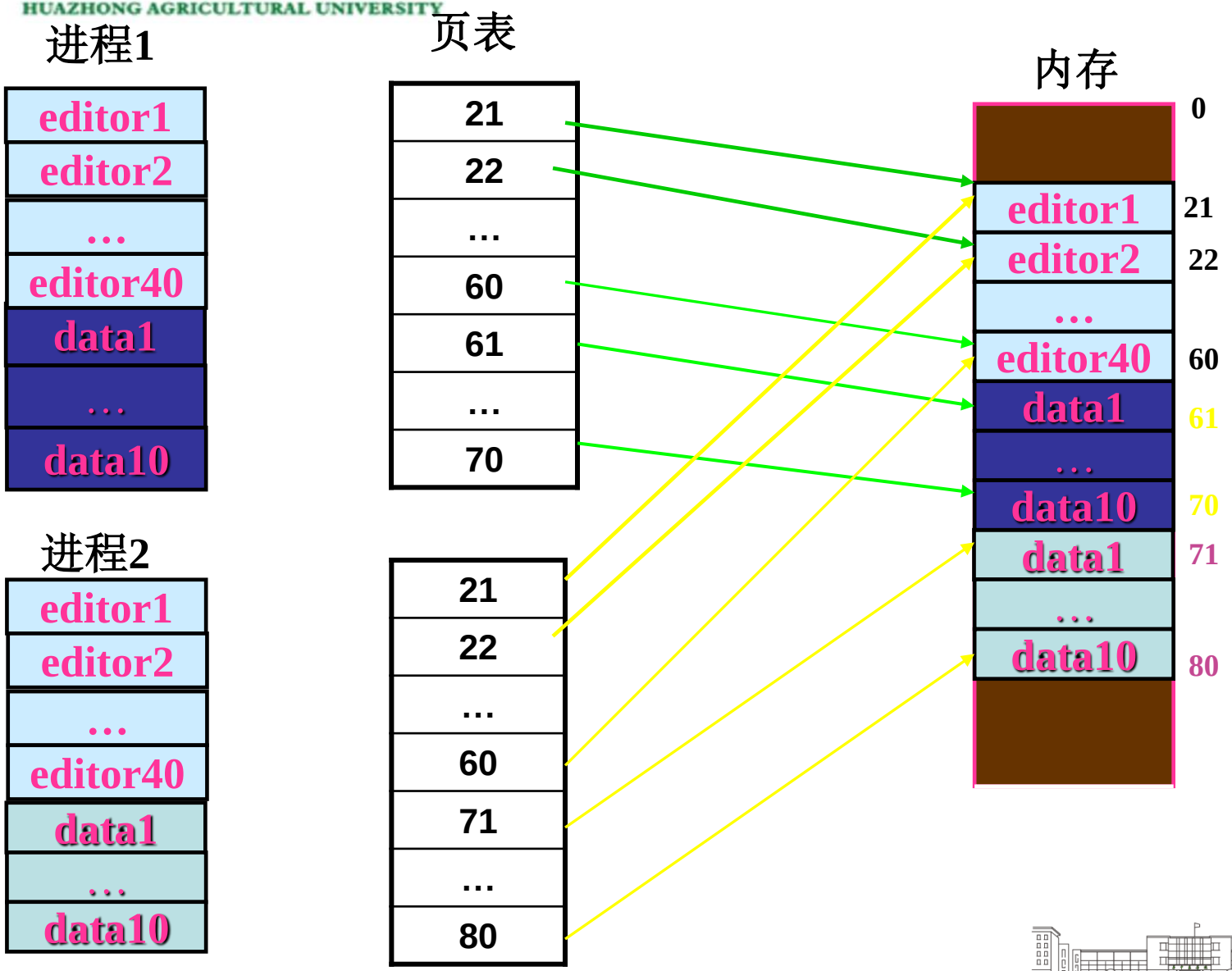
分页系统中的逻辑地址空间是一维的，

分段系统中的是二维的。

4.6.3、信息共享

分段系统易实现信息共享：




进程1
editor1
editor2
…
editor40
data1
…
data10
进程2
editor1
editor2
…
editor40
data1
…
data10
页表
21
22
…
60
61
…
70
21
22
…
60
71
…
80
内存
0
editor1
21
editor2
22
…
editor40
60
data1
61
…
data10
70
data1
71
…
data10
80

例1：已知某分页系统，主存容量为64k，页面大小为1k，对一个4页大的作业，第0、1、2、3页被分配到内存的2、4、6、7块中。
求：将十进制的逻辑地址1023、2500、4500转换成物理地址。

解：(1) 1023/1K，得到页号为0，页内地址1023。
又 对应的物理块号为2，故物理地址为2*1k+1023=3071

(2) 2500/1K，得到页号为2，页内地址452。
又 对应的物理块号为6，故物理地址为6*1k+452=6596

(3) 4500/1K，得到页号为4，页内地址404。
因为页号不小于页表长度，故产生越界中断。

例2：对于如下所示的段表，请将逻辑地址（0，137），（1，4000），（2，3600），（5，230）转换成物理地址。

段号	内存始址	段长
0	50k	10k
1	60k	3k
2	70k	5k
3	120k	8k
4	150k	4k

解：(4) 段号5 ＝ 段表长，故段号不合法。产生越界中断。

4.6.4、段页式存储管理方式

P150

基本思想

结合分页和分段技术；
分页：有效提高内存利用率
分段：很好的满足用户需求

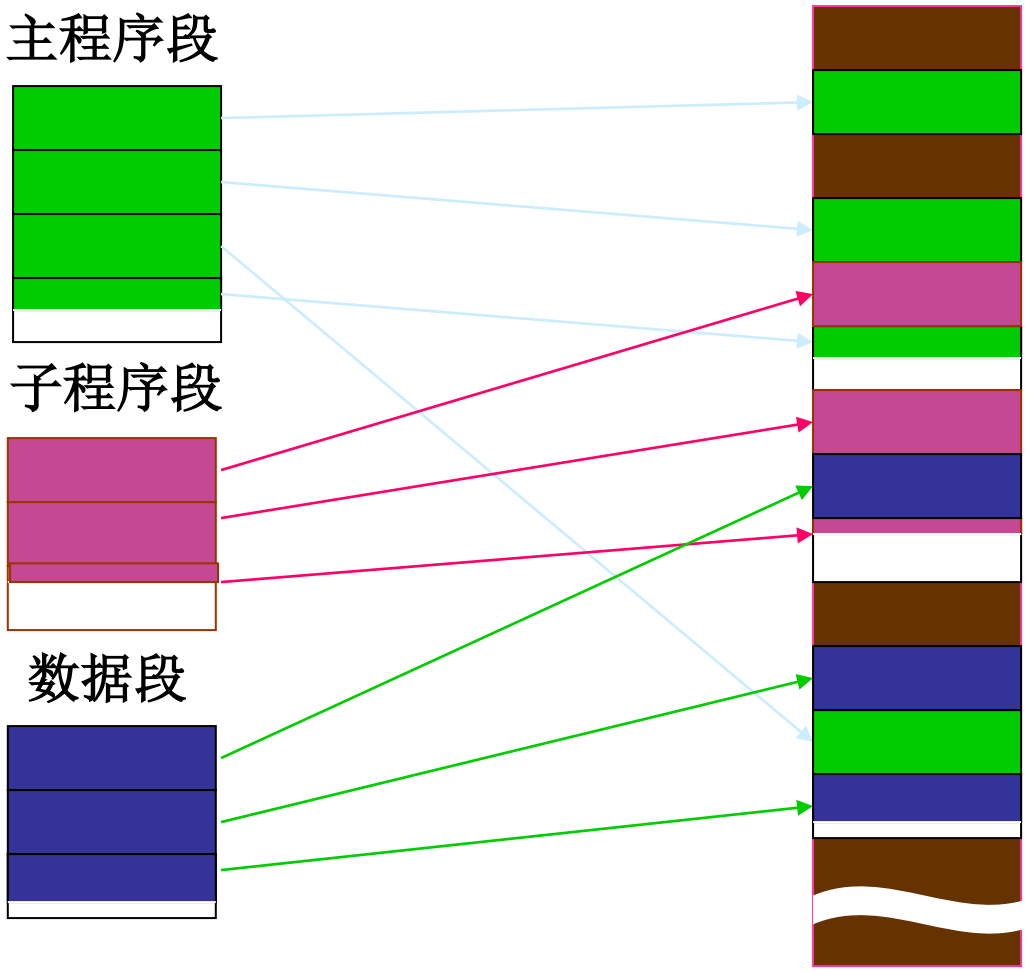
原理

对内存进行分页（物理块/页架）；
对用户作业先分段，各段再分页。

地址结构与地址变换

段号、段内页号、页内地址三部分
段表和页表

段号S	段内页号P	页内地址W


主程序段
子程序段
数据段

段表寄存器

段表始址	段表长度

段表

段号	状态	页表长度	页表始址
0	1		
1	1		
2	0		
3	1		

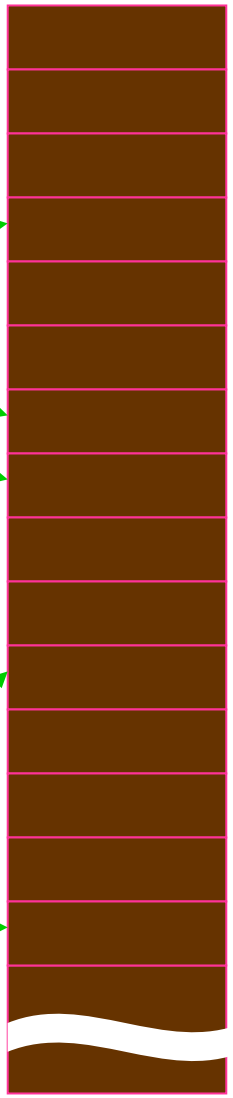
页表

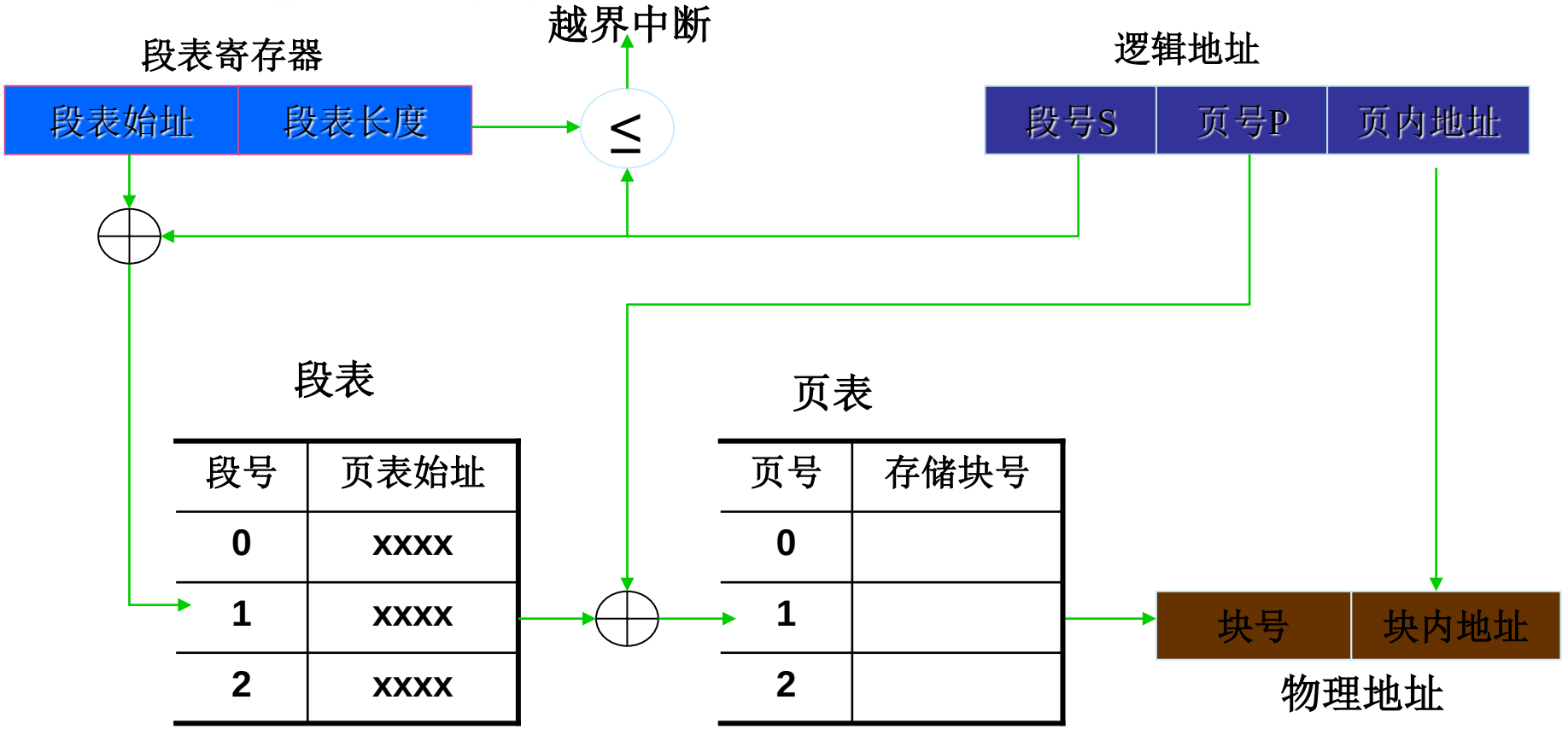
页号	状态	存储块号
0	1	
1	1	
2	1	
3	0	

页表

页号	状态	存储块号
0	1	
1	1	

……



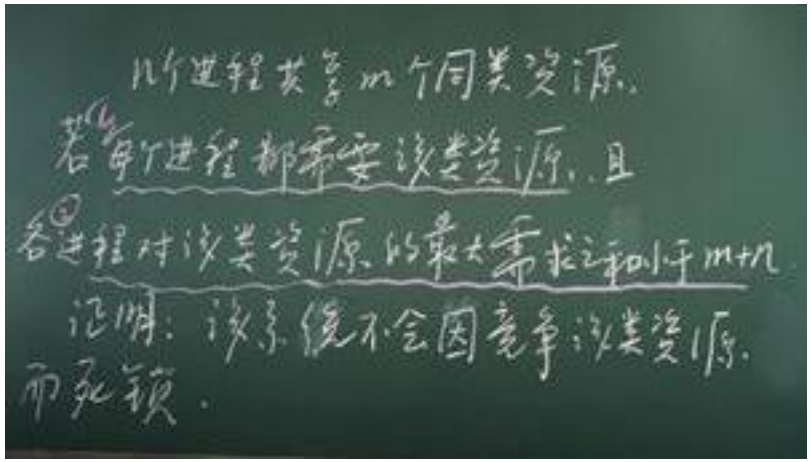

越界中断
段表寄存器
段表始址
段表长度
≤
逻辑地址
段号S
页号P
页内地址
段表
段号
页表始址
0
xxxx
1
xxxx
2
xxxx
页表
页号
存储块号
0
1
2
块号
块内地址
物理地址

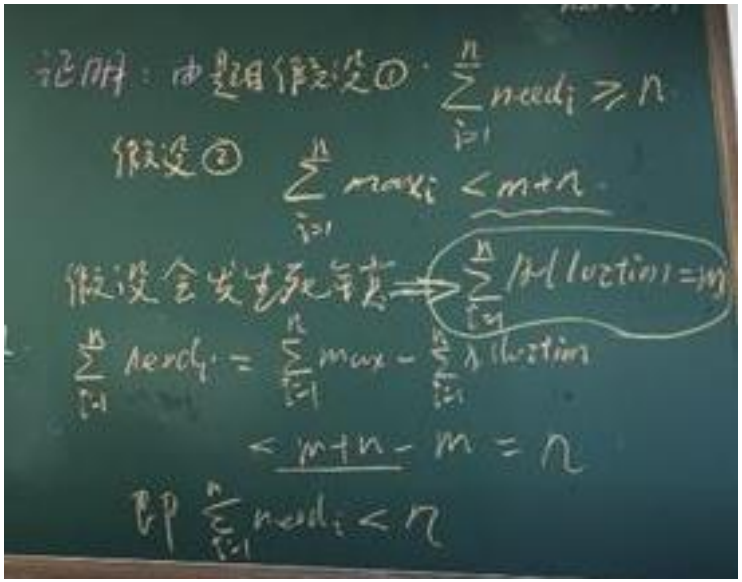
段页式存储管理的优缺点

同时具备分段和分页管理的优点：

分散存储，内存利用率较高；便于代码或数据共享，支持动态链接等。

访问效率下降：一次访问转换成了三次访问。


n个进程共享m个同类资源，
若每个进程都需要该类资源，且
各进程对该类资源的最大需求之和小于m+n
证明：该系统不会因竞争该类资源
而死锁。


证明：由题目假设①
假设②
假设会发生死锁

主观题 10分

设置

HUAZHONG AGRICULTURAL UNIVERSITY

在本课堂上，你想学习那些内容？
你有那些建议？

作答

正常使用主观题需2.0以上版本雨课堂

投票(匿名) 最多可选5项

设置

HUAZHONG AGRICULTURAL UNIVERSITY

在线听课评价:

A 教师准备充分,讲述条理清晰

B 能调动学习的积极性

C 教学方式单一、缺乏互动

D 学生的收获少，获得感不强

E 内容枯燥、听不懂

提交