第4章　内存管理

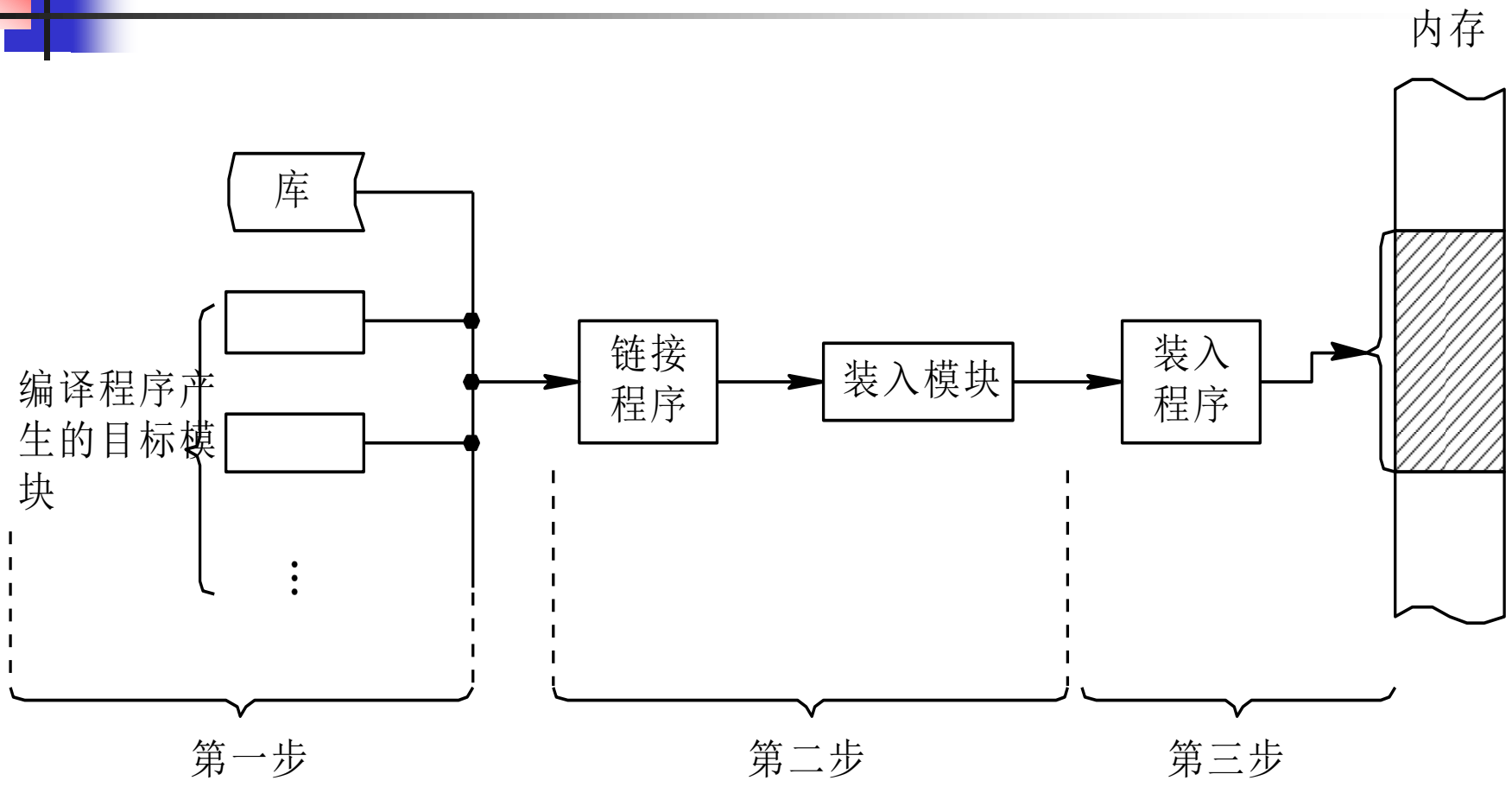
内存是计算机的工作场所，是除CPU之外的第二宝贵资源。内存能否高效利用，是衡量计算机系统效率高低的重要因素。本章主要介绍操作系统对内存资源的管理策略和方法。

教学要求

- 熟悉存储管理目的和功能，掌握地址重定位的概念。
- 熟悉单一连续分配、固定分区、可变分区实现原理，掌握可变分区分配的数据结构和分配回收算法。了解覆盖与交换的概念。
- 熟练掌握分页存储管理原理，熟练掌握基本的地址变换机构和具有快表的地址变换机构。掌握请求分页的页表机制、缺页中断机构和地址变换机构，掌握页面置换算法。
- 掌握虚拟存储器的理论基础和定义，熟悉虚拟存储器实现方式和特征。掌握分段、分页和段页式存储管理原理和地址变换机构。

教学内容

4.1程序的装入与链接



预备知识：三种地址概念

- 符号地址（出现在源程序中的地址概念）

  如：**start:mov ax,bx**

  jmp start

- 逻辑地址（源程序编译后生成的地址）

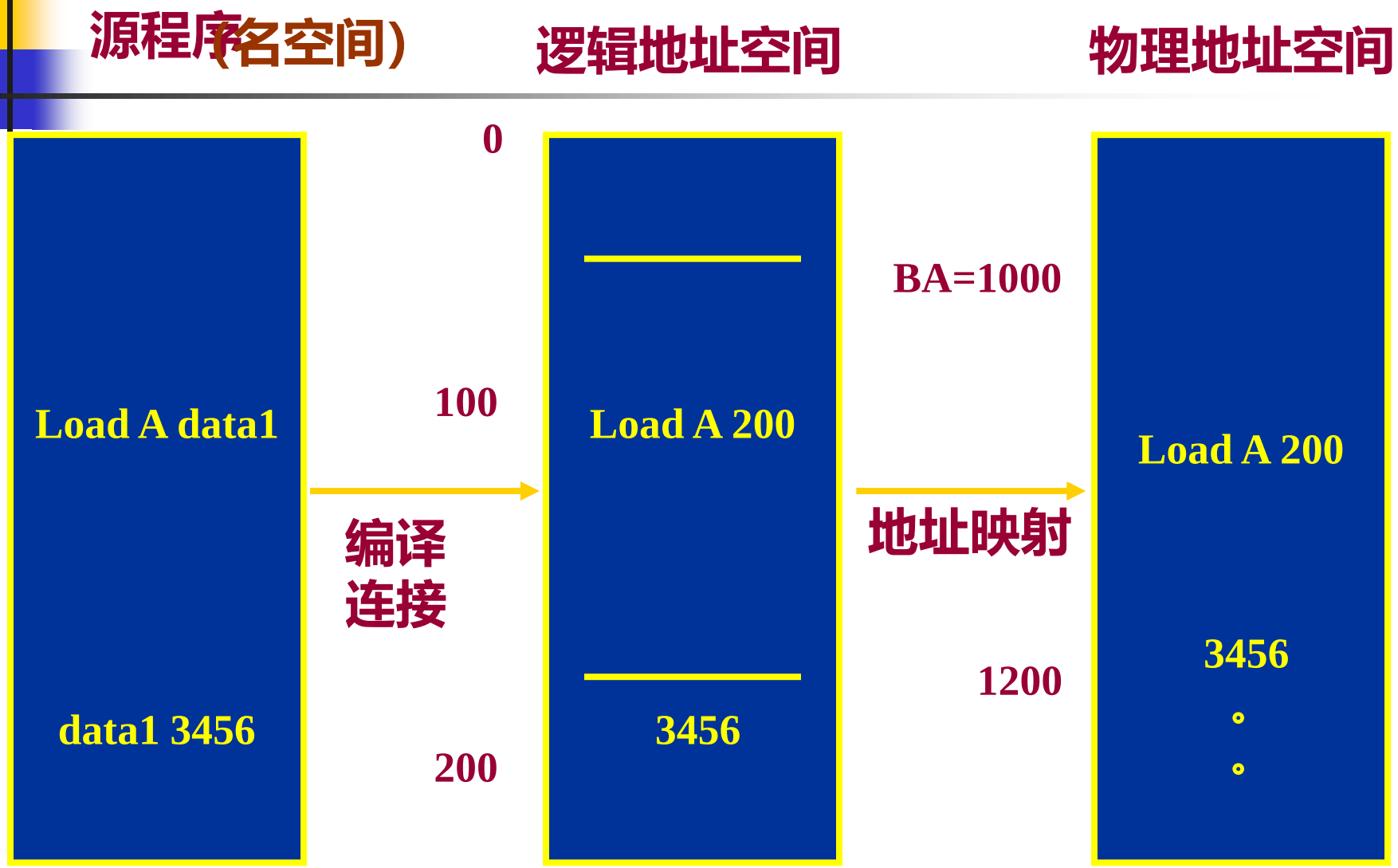
  是个相对地址概念，从**0**开始编址。

  如 **1200：mov ax,bx**

  jmp 1200

- 物理地址（也称内存地址、绝对地址）

三个地址空间


源程序(名空间)
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
地址映射
Load A 200
3456

地址转换（地址映射）

- **CPU**执行程序时，是按其逻辑地址来存取指令，而指令必须在内存中执行。为了能让**CPU**找到所执行的指令，必须将逻辑地址转换为内存地址的形式。

- 通常，逻辑地址与内存地址不存在一一对应的关系。

4.1.1 程序的装入

按装入时机的不同，可分为三种方式：

1.绝对装入方式

这种方式是指程序员采用物理内存地址编写程序。使用这种方式，必须事先划定内存的使用空间，这种方式下的内存利用率不高，用户使用较困难。因此，程序中通常使用符号地址来代替物理地址。

4.1.1 程序的装入（续）

2.可重定位装入方式

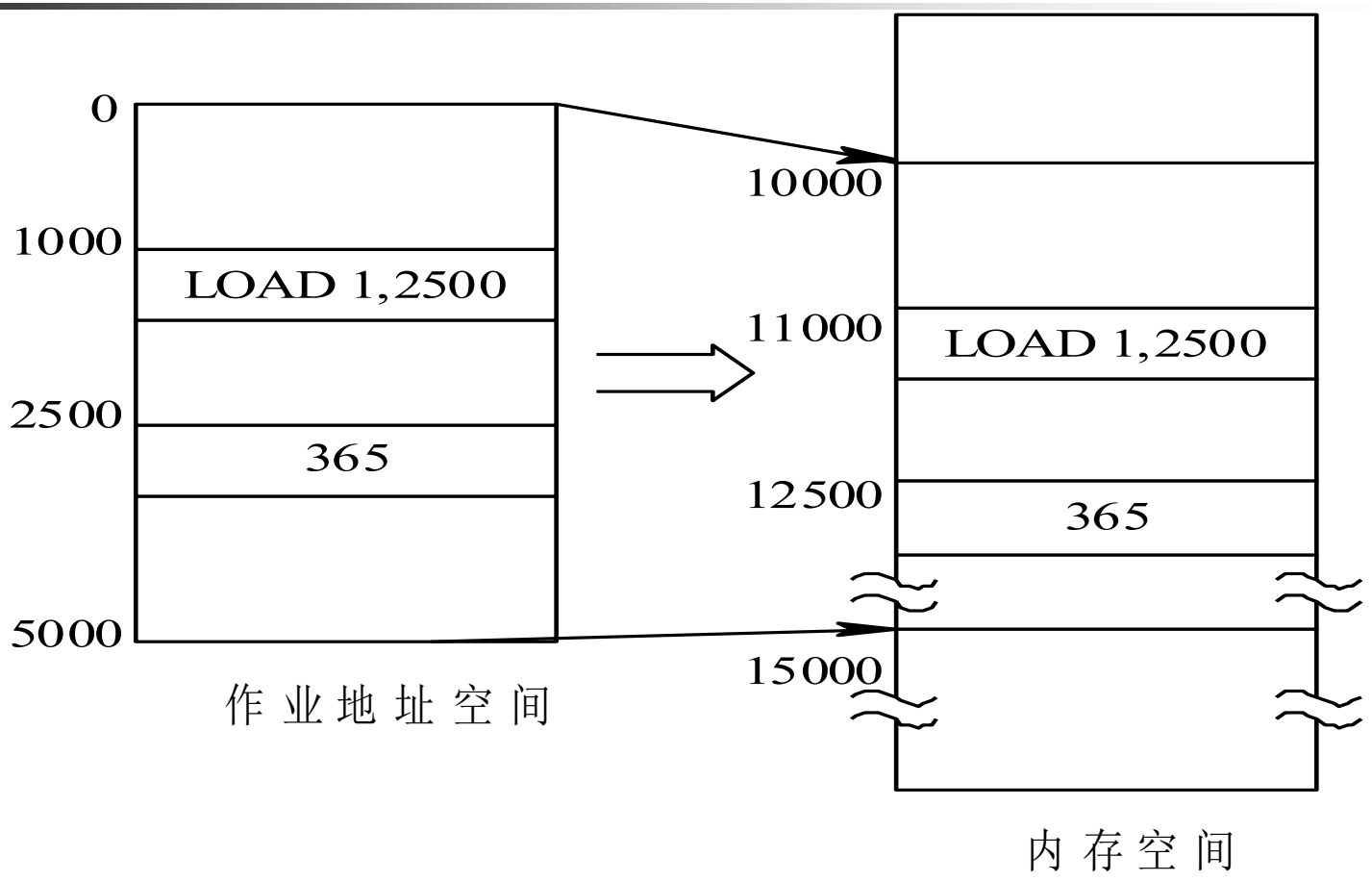
在程序装入内存时，由装入程序把装入模块中的所有逻辑地址转换为物理地址。又称静态重定位方式。

地址映射方法

- 假定程序装入内存的首地址为BR，程序地址为VR，内存地址为MR，则地址映射按下式进行：MR=BR+VR 。

- 例如，程序装入内存的首地址为10000，则装入程序就按MR=10000+VR对程序中所有地址部分进行修改，修改后指令Load A，2500就变为Load A，12500

指令和数据地址静态重定位



静态重定位的优缺点

优点：不需要硬件的支持；

缺点：程序必须占用连续的内存空间；一旦程序装入后不能移动；

4.1.1 程序的装入（续）

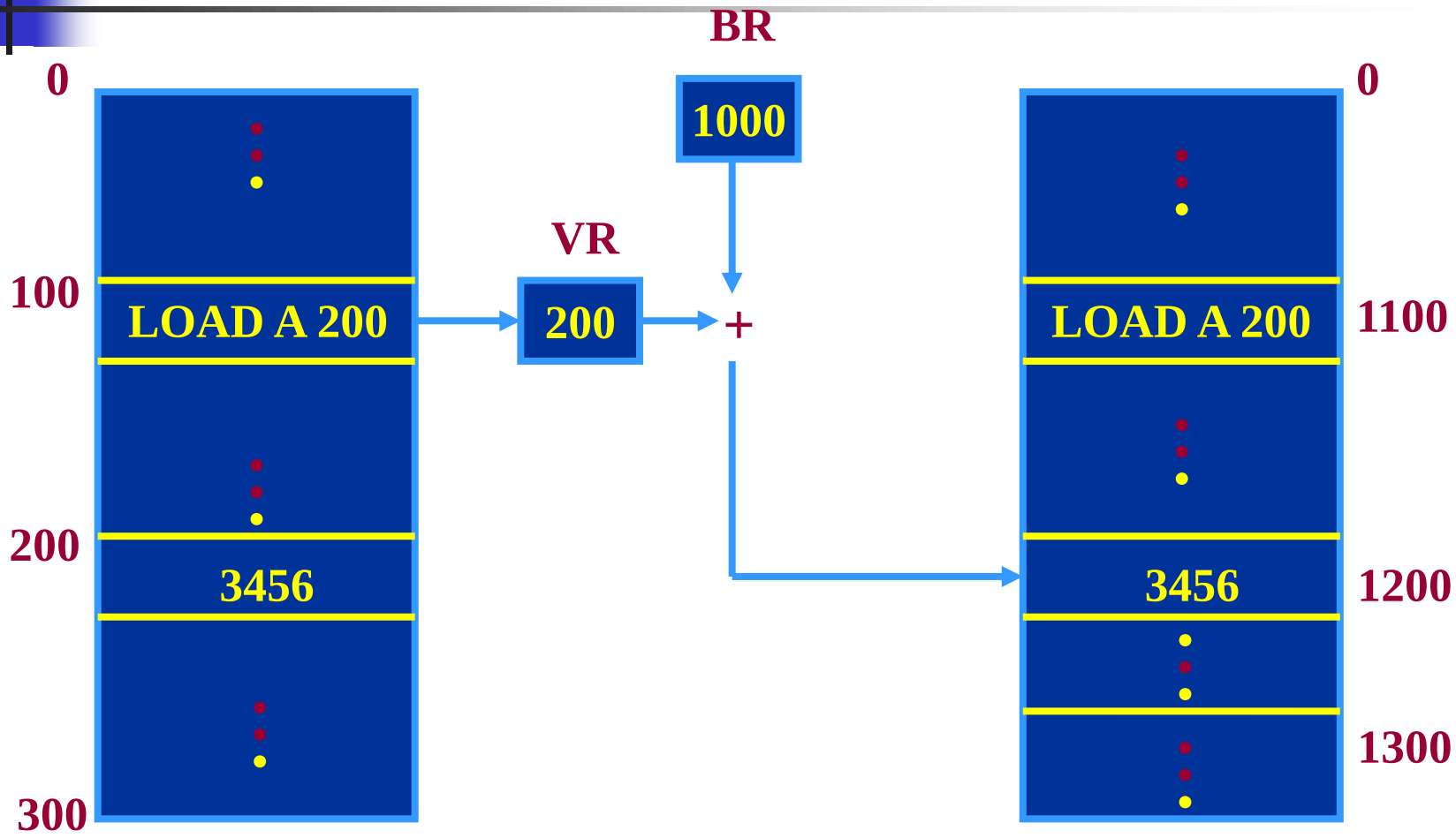
3.动态运行时装入方式

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。也称动态重定位方式。

地址映射方法

- 最简单的硬件机构是重定位寄存器。
- 在地址重定位机构中，有一个基地址寄存器BR和一个程序地址寄存器VR，一个内存地址寄存器MR。
- 地址映射：MR=BR+VR

动态重定位示意图

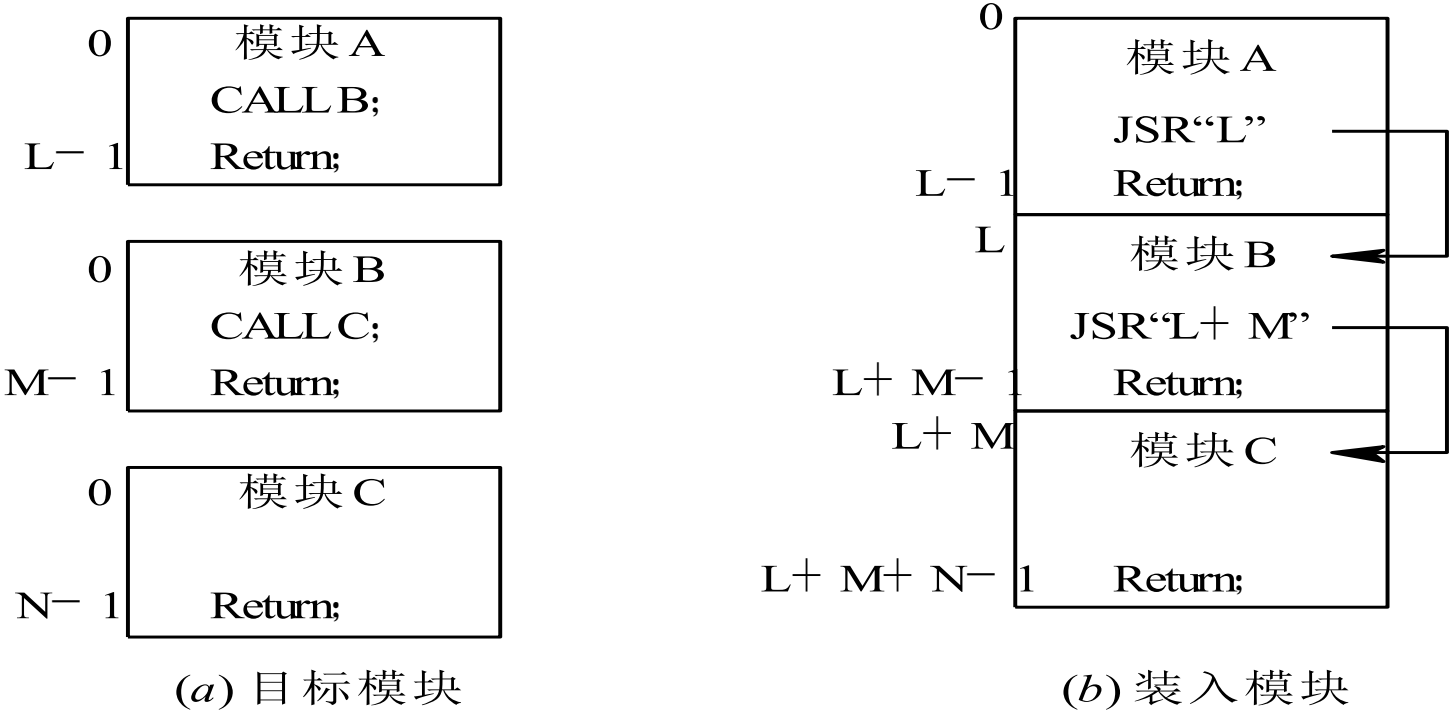

BR
0
1000
VR
100
LOAD A 200
200
+
0
LOAD A 200
1100
3456
200
3456
1200
1300
300

动态重定位的优缺点

- 优点：支持程序在内存中的移动，通过改变BR寄存器的值实现；
- 缺点：需要硬件的支持；存储管理的软件算法较为复杂；

4.1.2 程序的链接

1. 静态链接方式：装入内存前完成的链接



4.1.2 程序的链接（续）

2. 装入时动态链接

装入一个模块后，需调用另一模块时才进行链接。 优点：

(1) 便于修改和更新。

(2) 便于实现对目标模块的共享。

4.1.2 程序的链接（续）

3. 运行时动态链接

对某些模块的链接推迟到执行时才执行。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。

4. 2 分区管理

4.2.1 单分区管理

4.2.2 固定分区

4.2.3 可变分区

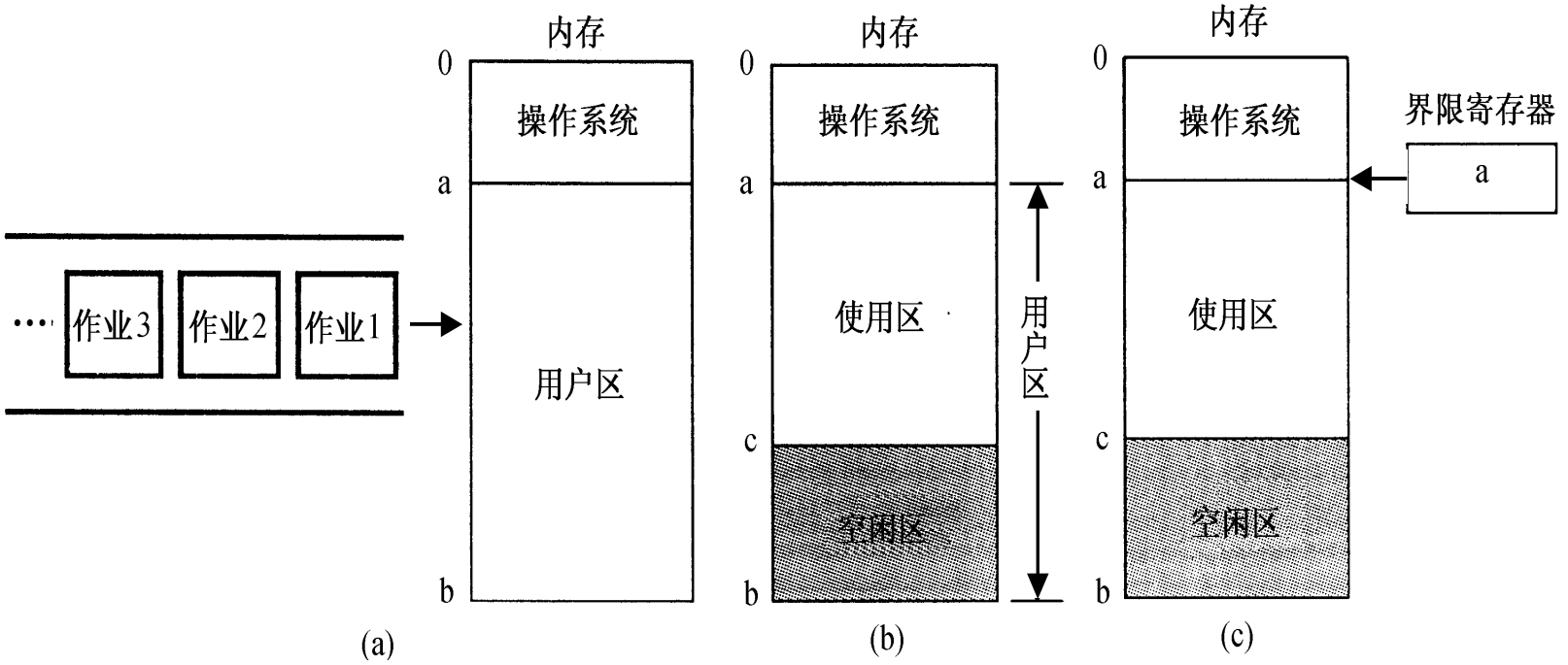
4.2.4 覆盖与交换

4.2.1 单分区管理

这是一种最简单的连续存储管理方式。但只能用于单用户、单任务的操作系统中。

- 系统区:仅提供给操作系统使用，通常设置在内存的低址部分;
- 用户区:指除系统区以外的全部内存空间，提供给用户使用。
- 空闲区:指剩余部分存储区。

单一连续分区分配示意图



4.2.2 固定分区

把可用空间划分成若干个固定大小的存储区，除操作系统占用一个区域外，其余区域为系统中多个用户共享，因为在系统运行期间，分区大小、数目都不变，所以固定分区也称为静态分区。

4.2.2 固定分区（续）

（a）

分区说明表

分区号	起始地址	分区大小	状态
1	40k	8k	0
2	48k	32k	1
3	80k	64k	0
4	144k	112k	1

（b）

4.2.3 可变分区

分区大小、数目可变，所以可变分区也称为动态分区。

可变分区内存使用情况示意图

（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

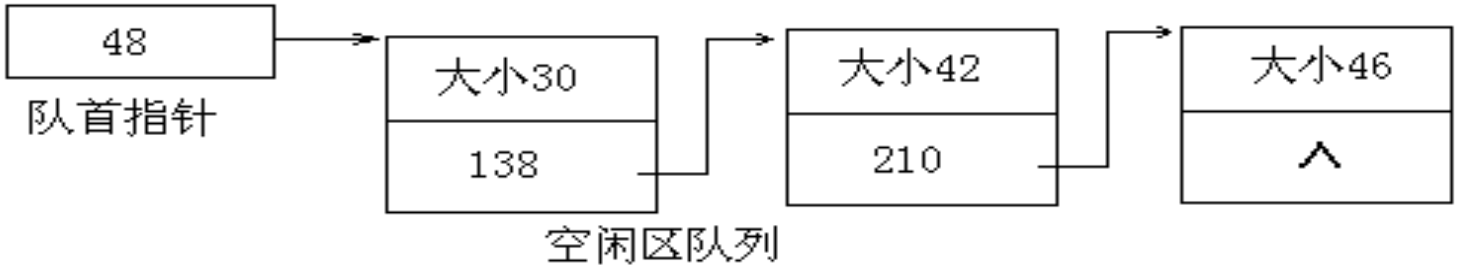
1. 可变分区中空闲区的组织形式

- 空闲区表
- 空闲区链（空闲区队列）

空闲区表、空闲区链（队列）示例

大小	位置
30	48
42	138
46	210
...	...

空闲区表



2. 内存的回收

- 检查回收区与内存中前后空闲区是否相邻，若相邻，则应进行合并，形成一个较大的空闲区，并对相应的链表指针进行修改；
- 若不相邻，应将空闲区插入到空闲区队列的适当位置。

3. 内存的分配算法

- 首次适应算法

  按空闲区地址递增的次序分配

- 最佳适应算法

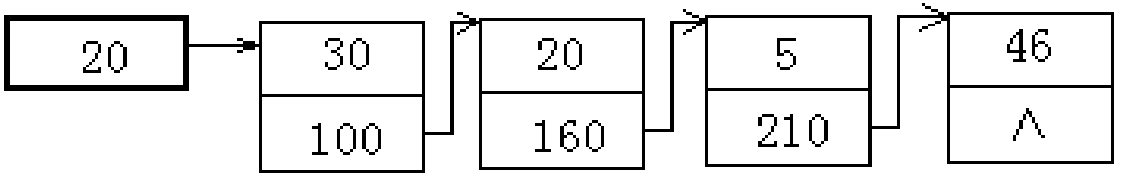
  按空闲区由小到大的次序分配

- 最坏适应算法

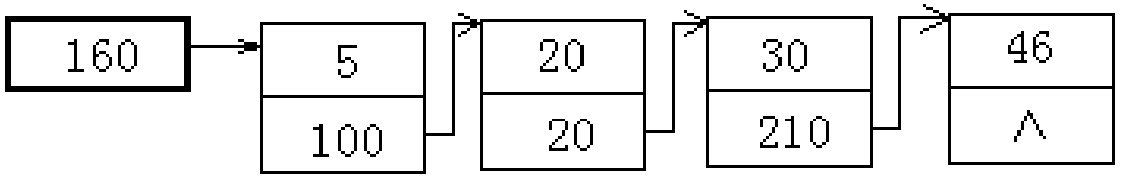
  按空闲区由大到小的次序分配

内存的分配算法（续）

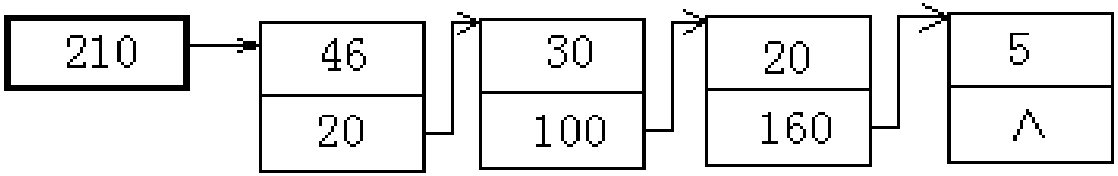
【例】有作业序列：作业A要求18K；作业B要求25K，作业C要求30K。系统中空闲区按三种算法组成的空闲区队列如下，请判断使用哪一种分配算法最合适。



首次适应法



最佳适应法



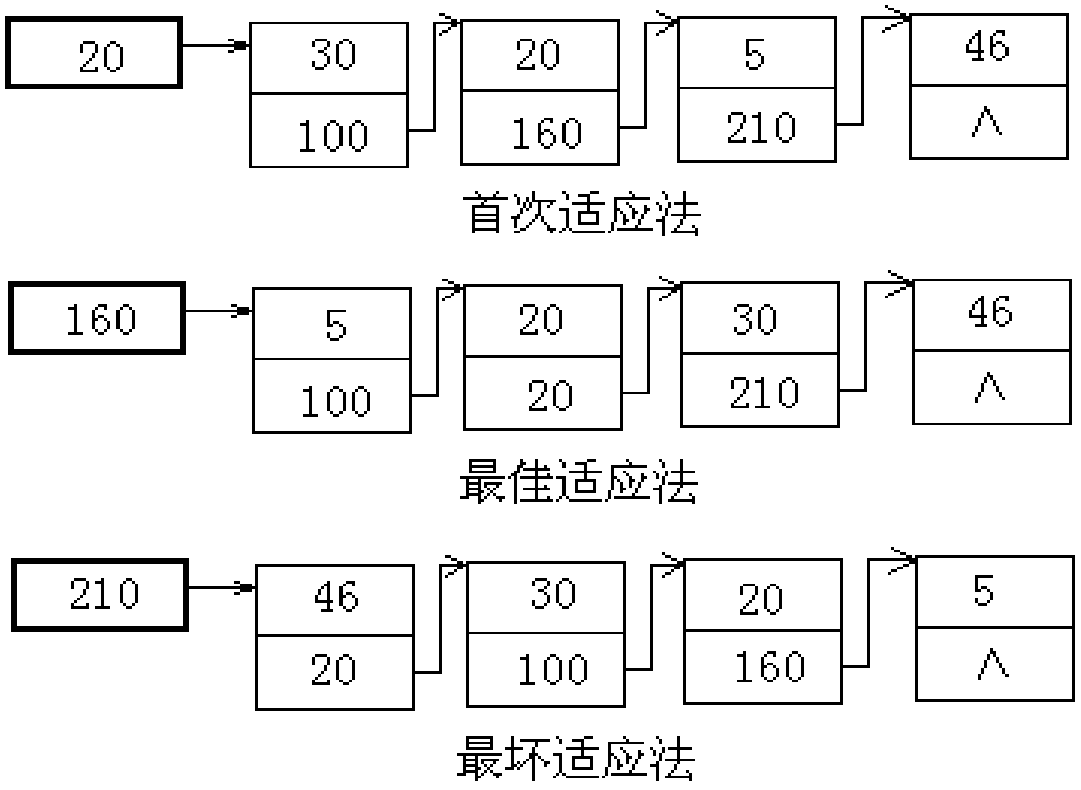
最坏适应法

【课堂练习】

有作业序列：作业A要求21K；作业B要求30K，作业C要求25K。

【解答】


os
20
30
50
作业
100
20
120
作业
160
5
165
作业
210
46
256


20
30
100
20
160
5
210
46
^
首次适应法
160
5
100
20
20
30
210
46
^
最佳适应法
210
46
20
30
100
20
160
5
^
最坏适应法

4.2.4 覆盖与交换

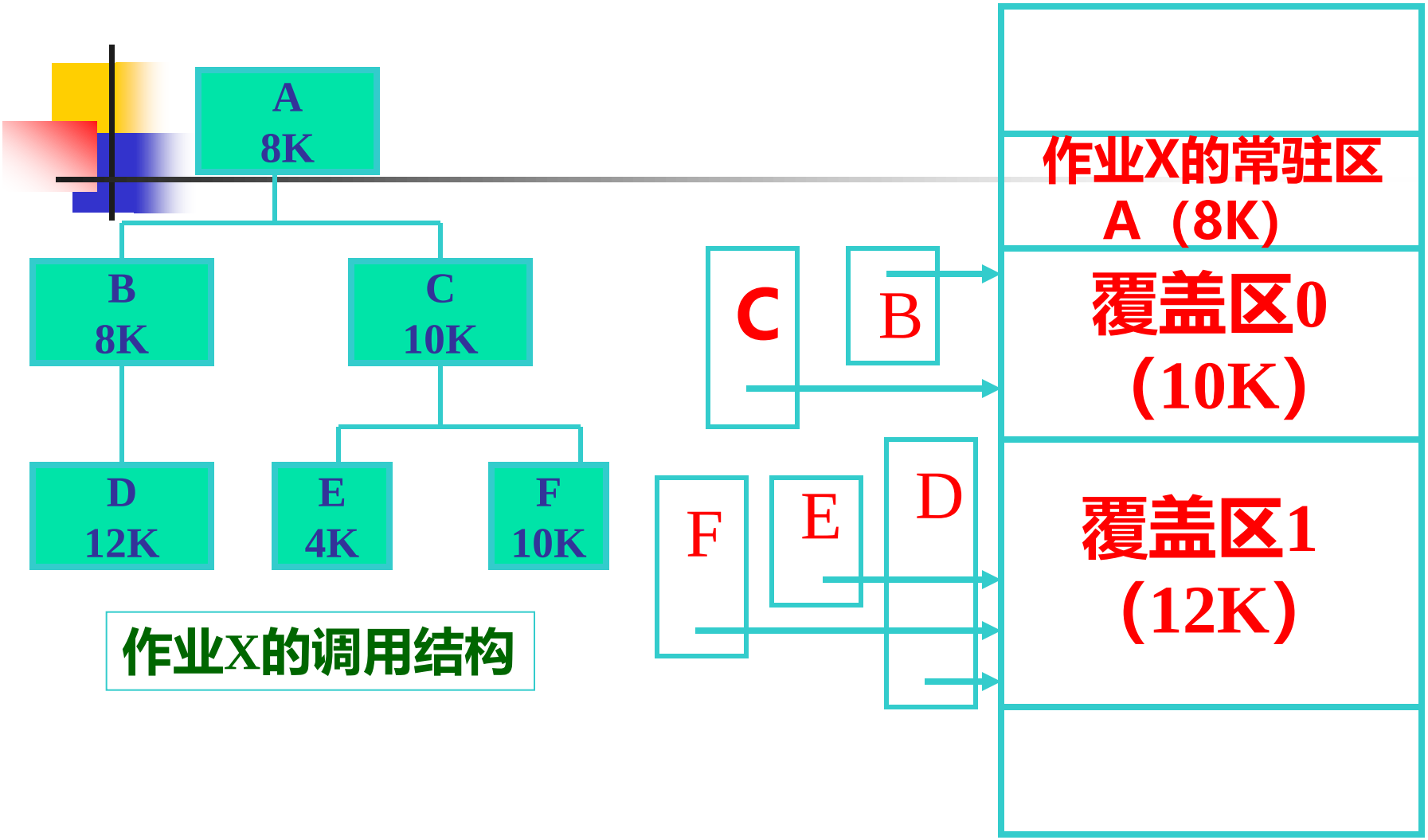
1.覆盖

- 覆盖：一个作业的若干程序段，或几个作业的某些部分共享某一段内存空间

- 覆盖的目的是解决小分区中运行大作业的问题

覆盖原理

- 把程序划分为若干个功能上相对独立的程序段，按照其自身的逻辑结构将那些不会同时执行的程序段共享同一块内存区域。
- 程序段先保存在磁盘上，当有关程序段的前一部分执行结束，把后续程序段调入内存，覆盖前面的程序段（内存“扩大”了）
- 一般要求作业各模块之间有明确的调用结构，程序员要向系统指明覆盖结构，然后由操作系统完成自动覆盖。


A
8K
B
8K
C
10K
D
12K
E
4K
F
10K
作业X的调用结构
作业X的常驻区
A（8K）
C
B
覆盖区0
（10K）
F
E
D
覆盖区1
（12K）

2. 交换

- 概念：系统将内存中某些进程暂时移到外存，把外存中某些进程换进内存，占据前者所占用的区域。

- 目的：在单分区中实现多任务

2.交换（续）

与覆盖技术相比，交换技术不要求用户给出程序段之间的逻辑覆盖结构；

- 交换发生在进程或作业之间，而覆盖发生在同一进程或作业内。
- 覆盖只能覆盖那些与覆盖段无关的程序段

分区管理的不足之处

- 会出现碎片问题，从而降低了内存的利用率。虽然采用压缩存储区的方法可以解决碎片问题，但系统开销太大；
- 作业大小受当前可用分区大小的限制，会出现“合起来够，但分开不够”的现象；
- 作业在内存中必须连续存放；

4.3 页式管理

4.3.1 页式管理概述

1.基本原理

- 将进程的地址空间划分成若干个大小相等的区域，称为页。
- 将内存空间划分成与页相同大小的若干个物理块，称为块或页帧。
- 在为进程分配内存时，将进程中若干页分别装入多个不相邻接的块中。

4.3.1 页式管理概述（续）

2. 逻辑地址结构：

【示例】

4.3.1 页式管理概述（续）

3. 页式管理类型：

- 静态分页

  实存分页模式

- 动态分页（请求分页）

  虚存分页模式

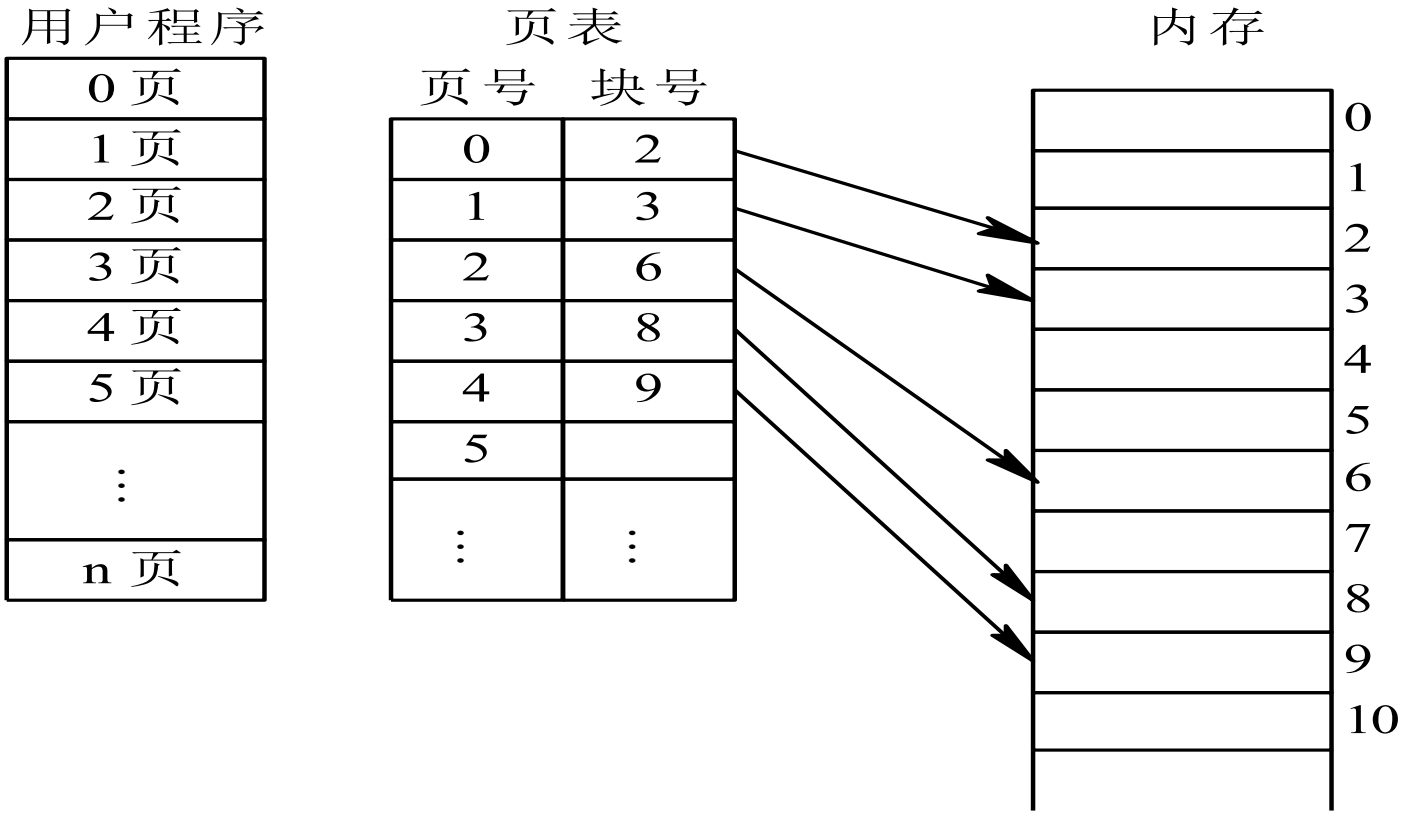
4.3.2 静态分页

1. 基本思想：必须装入作业的全部页面后才能执行该作业。

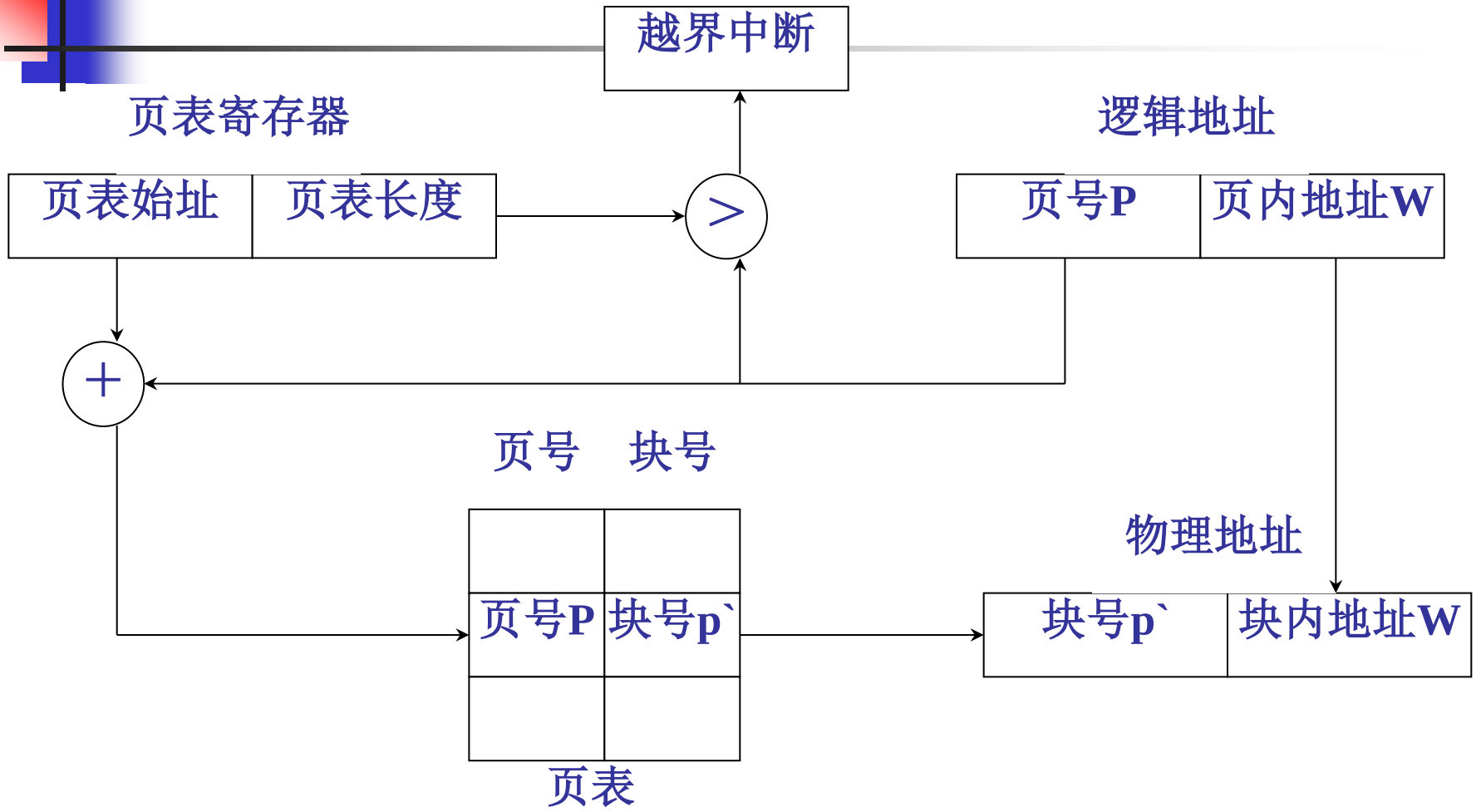
2. 地址转换：系统为每个进程建立了一张页面映射表，简称页表。每个页在页表中占一个表项，记录该页在内存中对应的物理块号。进程在执行时，通过查找页表，就可以找到每页所对应的物理块号。页表的作用是实现从页号到物理块号的地址映射。

页表

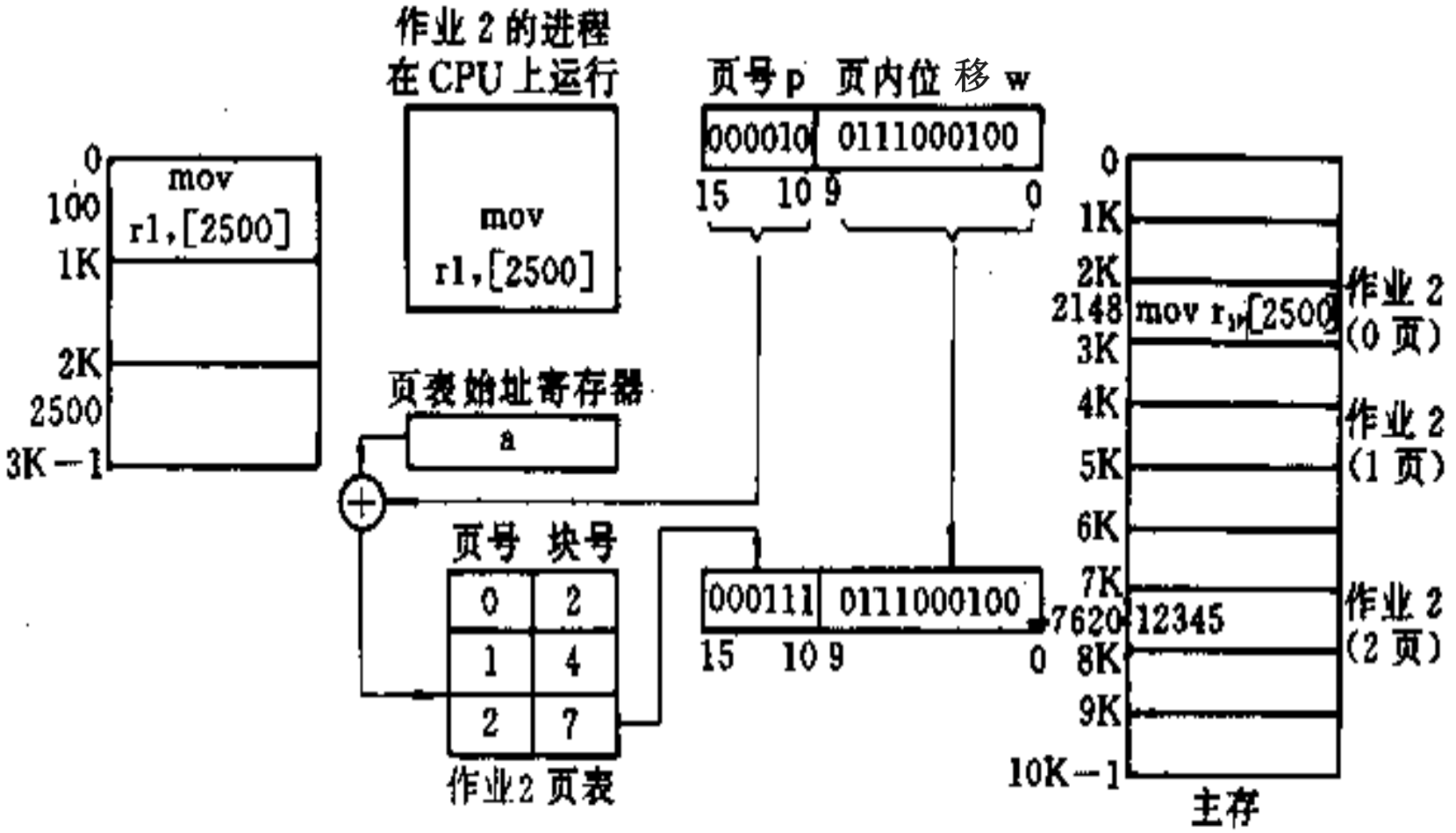
页号	块号
0	2
1	3
2	8


用户程序
0 页
1 页
2 页
3 页
4 页
5 页
n 页
页表
页号 块号
0 2
1 3
2 6
3 8
4 9
5
内存
0
1
2
3
4
5
6
7
8
9
10

分页中的地址转换机构


越界中断
页表寄存器
逻辑地址
页表始址
页表长度
>
页号P
页内地址W
+
页号
块号
物理地址
页号P
块号p`
块号p`
块内地址W
页表

【例】设页长为1K，程序地址字长为16位，作业2空间和页表如下图所示。

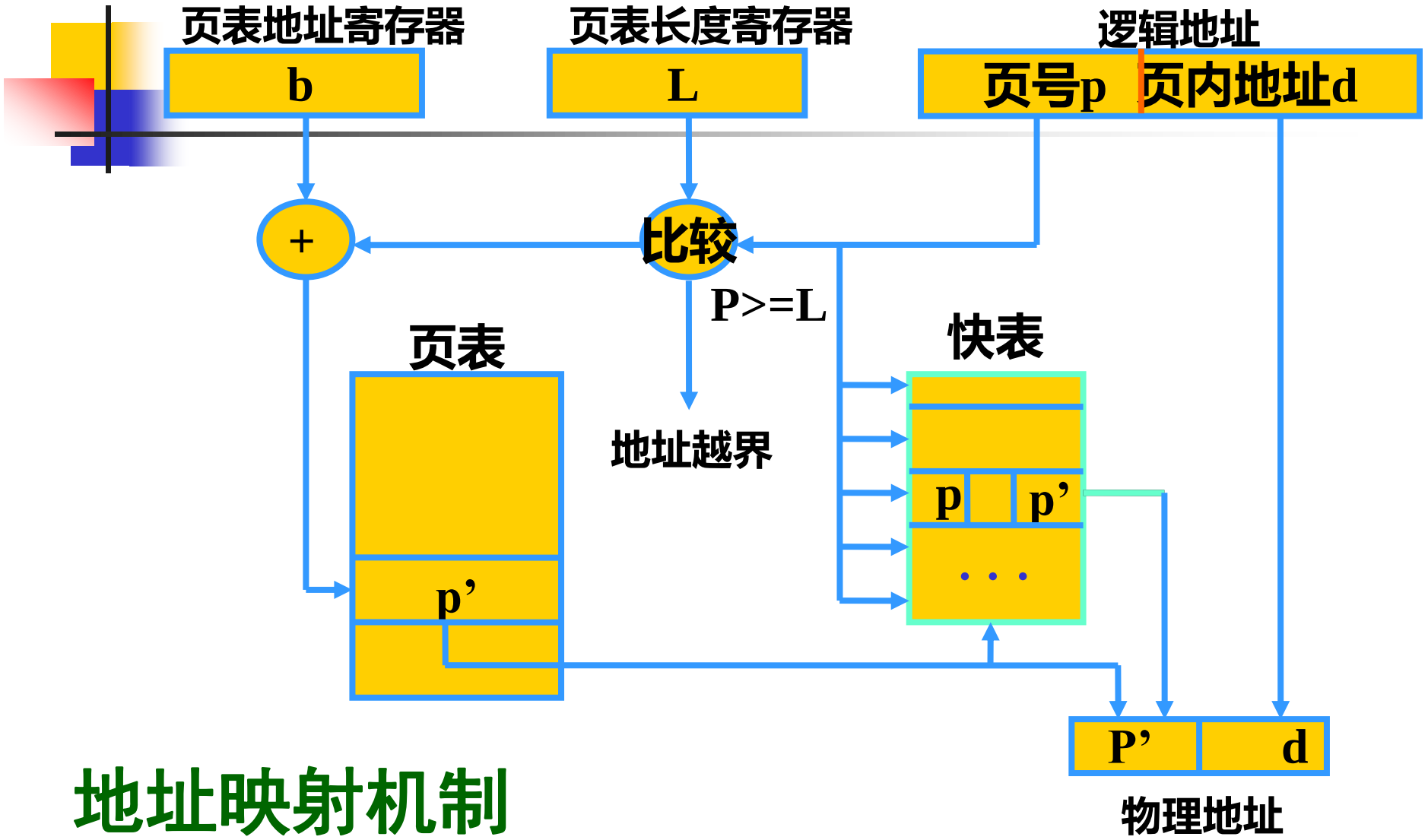


3. 快表与联想存储器

- 在前述的页地址变换过程中有一个严重的问题，那就是每一次对内存的访问都要访问页表，页表是放在内存中的，也就是说每一次访问内存的指令至少要访问两次内存，运行速度要下降一半。

- 若不解决这一问题是不能令人忍受的。

- 解决这个问题的一种方法是把页表放在一组快速存储器（也称为联想存储器）中，从而加快访问内存的速度。这种快速存储器中的页表称为快表，把存放在内存中的页表称为慢表。



地址映射机制

快表的规模多大才合适？

- 实际上我们并不需要一个很大的联想存储器，有一个能存放16个页表表目的联想存储器就够了。
- 硬件根据需要将页表中当前需要的少量表目读入快表，其它表目仍留在内存的页表中，当需要时读入新的表目，并淘汰适当的表目。

【思考题1】

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要**3us**，问实现一次页面访问要多长时间。

(2)如系统有快表，平均命中率为**97%**，假设访问快表的时间忽略为**0**，问此时一次页面访问要多长时间。

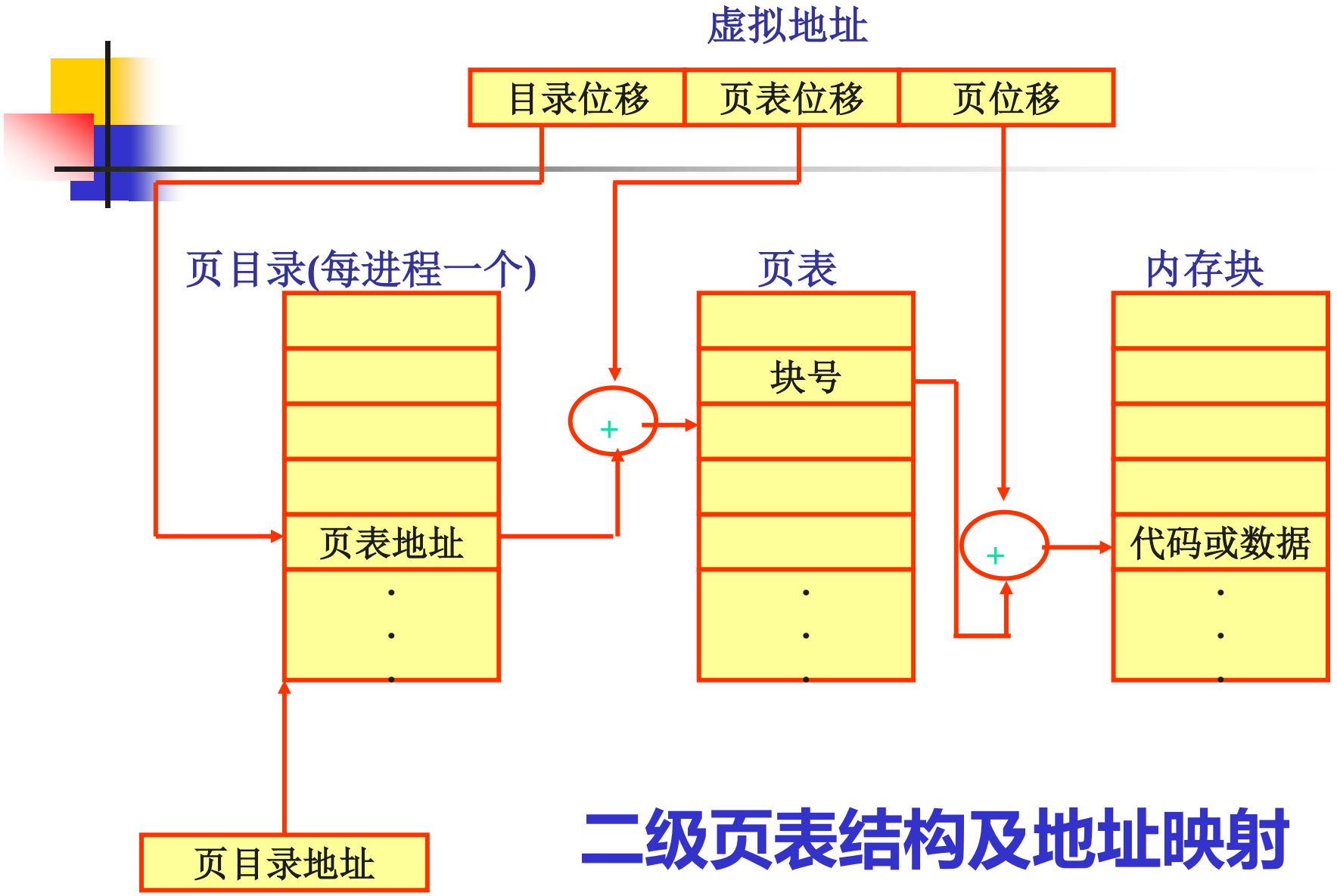
【答案】

1、2*3=6us

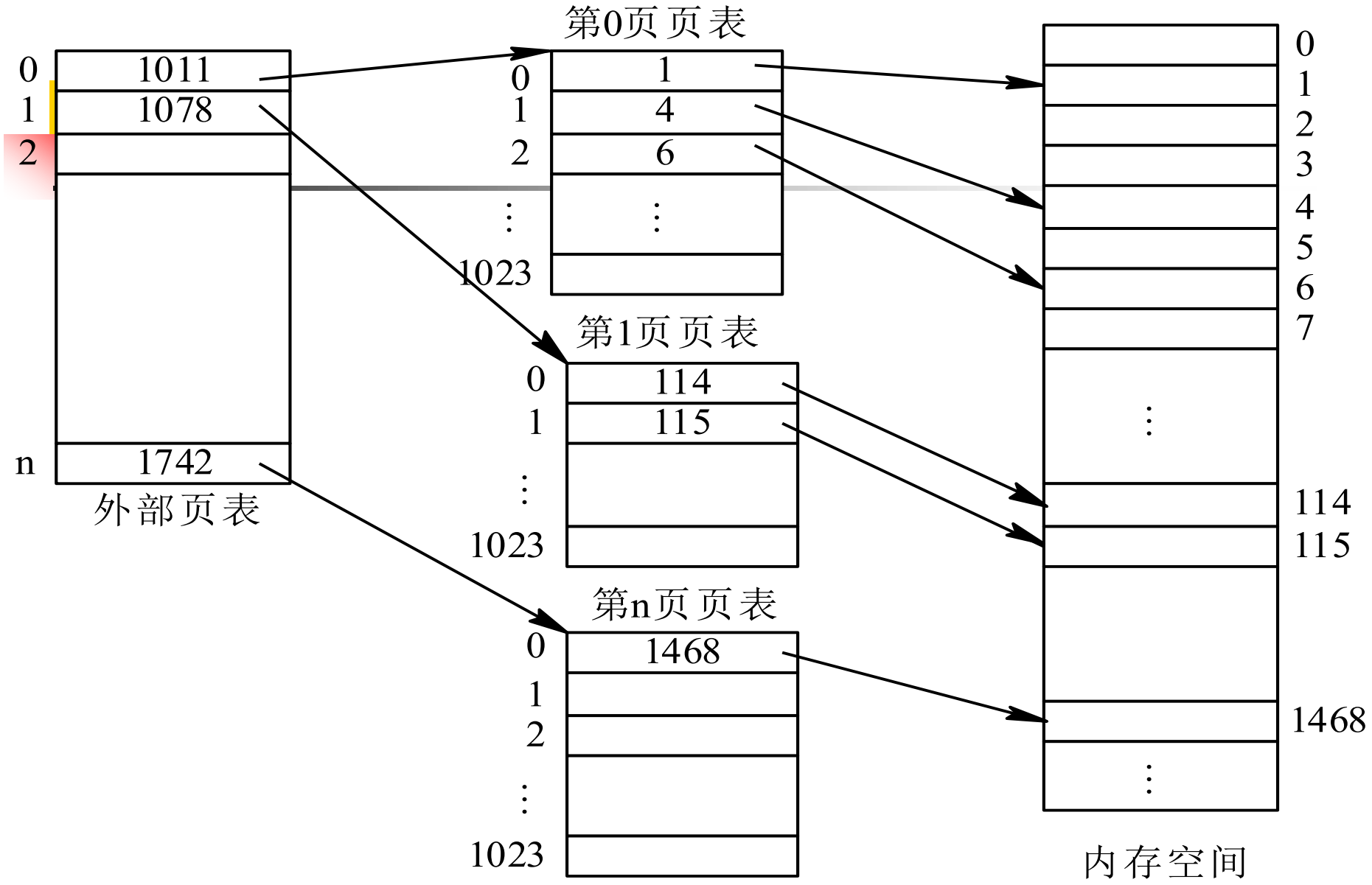
2、0.97*3+0.03*6=3.09us

4. 二级页表

- 当页表项很多时，仅采用一级页表需要大片连续空间，可将页表也分页，并对页表所占的空间进行索引形成外层页表（也称页目录表）。由此构成二级页表。
- 更进一步可形成多级页表。



二级页表结构及地址映射


第0页页表
第1页页表
第n页页表
外部页表
内存空间
0
1
2
n
1011
1078
1742
1023
1
4
6
114
115
1468
7
1468

4.3.3 虚拟存储器

- 虚拟存储器是具有请求调入功能和置换功能，能从逻辑上对内存容量进行扩充的存储器系统。其逻辑容量由内存和外存之和所决定。其运行速度接近于内存，但每位的成本接近于外存。
- 理论基础是程序的局部性原理。包括时间局部性和空间局部性。

局部性原理

- 程序执行时， 除了少部分的转移和过程调用指令外，在大多数情况下仍是顺序执行的。
- 过程调用将会使程序的执行轨迹由一部分区域转至另一部分区域，但经研究看出，过程调用的深度在大多数情况下都不超过**5**。
- 程序中存在许多循环结构，这些虽然只由少数指令构成， 但是它们将多次执行。
- 程序中还包括许多对数据结构的处理，如对数组进行操作， 它们往往都局限于很小的范围内

虚拟存储器的特征

- 多次性
- 对换性
- 虚拟性

4.3.4 动态分页

1. 基本思想：采用虚拟存储技术，只需装入作业的部分页面就能启动作业的运行。以后再通过调页功能和页面置换功能，陆续把将要运行的页面调入内存，同时把暂不运行的页面置换到外存上，置换时以页面为单位。

2.页表结构

页号	块号	状态位	访问位	修改位	外存地址
0	10	1	1	0	120
1	14	1	0	0	123
2	15	1	2	0	134
3	—	0	—	—	135

3. 缺页中断

当所要访问的页面不在内存时，便要产生缺页中断，请求OS将所缺页调入内存。与一般中断的主要区别在于：

- 缺页中断在指令执行期间产生和处理中断信号，而一般中断在一条指令执行完后检查和处理中断信号。
- 缺页中断返回到该指令的开始重新执行该指令，而一般中断返回到该指令的下一条指令执行。
- 一条指令在执行期间，可能产生多次缺页中断。

【思考题2】

内存分配一页，初始时矩阵数据均不在内存；页面大小为**128**个整数；矩阵**A128X128**按行存放。

这两个程序执行时分别会产生多少次缺页中断？

- 程序编制方法1：

```
for j:=1 to 128
  for i:=1 to 128
    A[i,j]:=0;
```

- 次数：128*128

程序编制方法2：

```
for i:=1 to 128
  for j:=1 to 128
    A[i,j]:=0;
```

次数：128

【思考题3】

- 一个有快表的请求页式虚存系统，内存访问周期为1微妙，内外存传送一个页面平均时间为5毫秒，如果快表命中率为75%,缺页中断率为10%。忽略快表访问时间，试求内存的有效存取时间。

【解答】

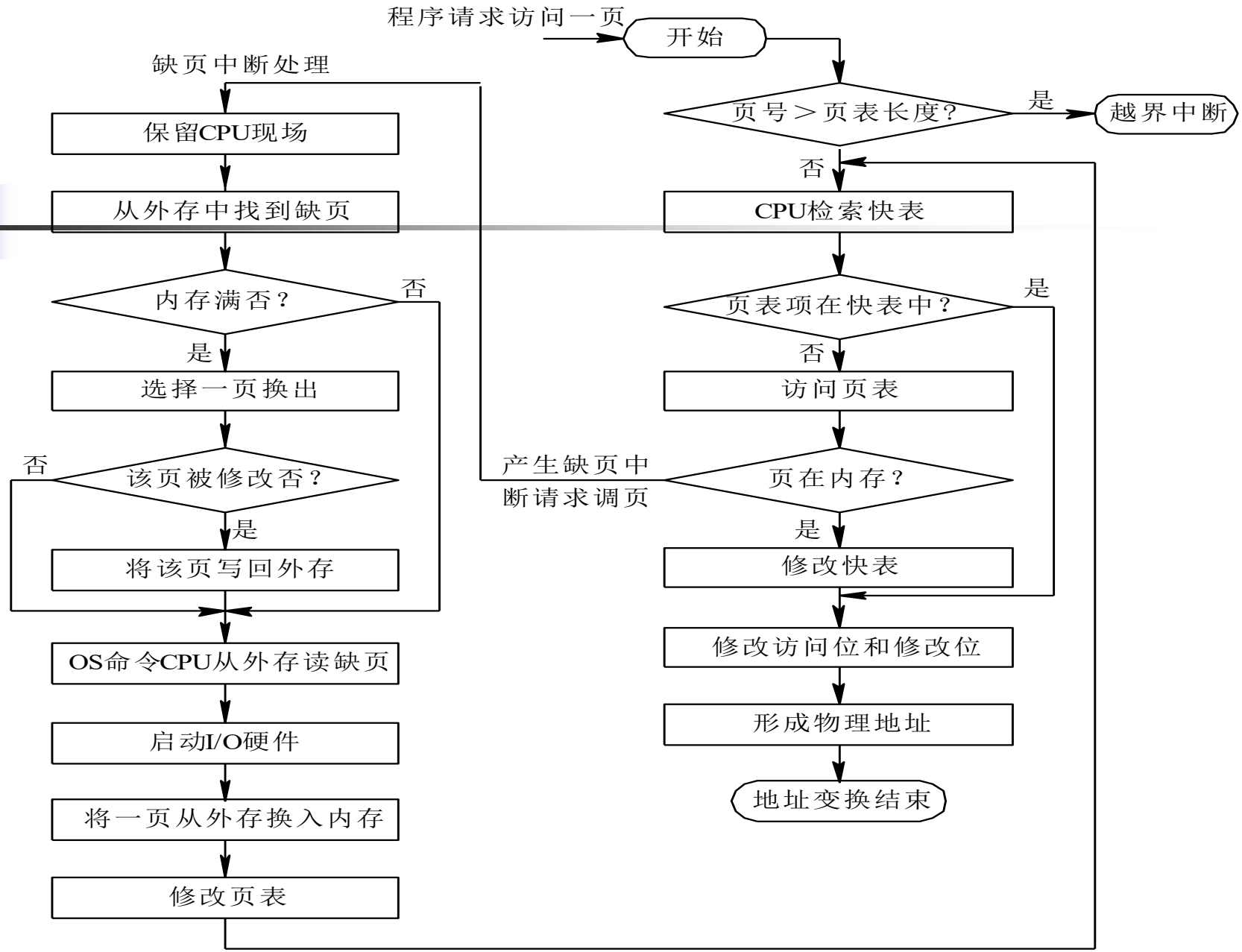
- 内存命中率为**1-75%-10%=15%**

- 内存有效存取时间：

1*75%+2*15%+（5000+2）*10%

4. 地址变换机构

请求分页系统中的地址变换机构，是在分页系统的地址变换机构的基础上，再为实现虚拟存储器而增加了某些功能所形成的，如产生和处理缺页中断，以及从内存中换出一页的功能等等。


程序请求访问一页
开始
页号＞页表长度？
是
越界中断
否
CPU检索快表
页表项在快表中？
是
否
访问页表
产生缺页中
断请求调页
页在内存？
是
修改快表
修改访问位和修改位
形成物理地址
地址变换结束
缺页中断处理
保留CPU现场
从外存中找到缺页
内存满否？
否
是
选择一页换出
否
该页被修改否？
是
将该页写回外存
OS命令CPU从外存读缺页
启动I/O硬件
将一页从外存换入内存
修改页表

5. 页面置换算法

◆最优置换算法（OPT算法）

从内存中移出以后不再使用的页面；或选择以后最长时间内不需要访问的页。这种算法本身不是一种实际的方法，因为页面访问的顺序是很难预知的。

页面置换算法（续）

◆先进先出算法（FIFO算法）

总是先淘汰那些驻留在内存时间最长的页面，即先进入内存的页面先被置换掉。

理由是：最先进入内存的页面不再被访问的可能性最大。

页面置换算法（续）

◆最近最少使用算法（LRU算法）

当需要置换一页时，选择在最近一段时间内最久没有使用过的页面予以淘汰。故也称最近最久未使用算法。

近似算法：

最近未使用算法（NRU）或Clock算法

OPT算法性能分析（M=3）

时刻	0	1	2	3	4	5	6	7	8	9	10	11	12
页面		1	2	3	4	1	2	5	1	2	3	4	5
块1		1	1	1	1			1			3	3	
块2			2	2	2			2			2	4	
块3				3	4			5			5	5	
缺页		√	√	√	√			√			√	√	

缺页率＝7/12

FIFO算法性能分析（M=3）

时刻	0	1	2	3	4	5	6	7	8	9	10	11	12
页面		1	2	3	4	1	2	5	1	2	3	4	5
块1		1	2	3	4	1	2	5			3	4	
块2			1	2	3	4	1	2			5	3	
块3				1	2	3	4	1			2	5	
缺页		√	√	√	√	√	√	√			√	√	

缺页率＝9/12

LRU算法性能分析（M=3）

时刻	0	1	2	3	4	5	6	7	8	9	10	11	12
页面		1	2	3	4	1	2	5	1	2	3	4	5
块1		1	1	1	4	4	4	5			3	3	3
块2			2	2	2	1	1	1			1	4	4
块3				3	3	3	2	2			2	2	5
缺页		√	√	√	√	√	√	√			√	√	√

缺页率＝10/12

页式管理小结

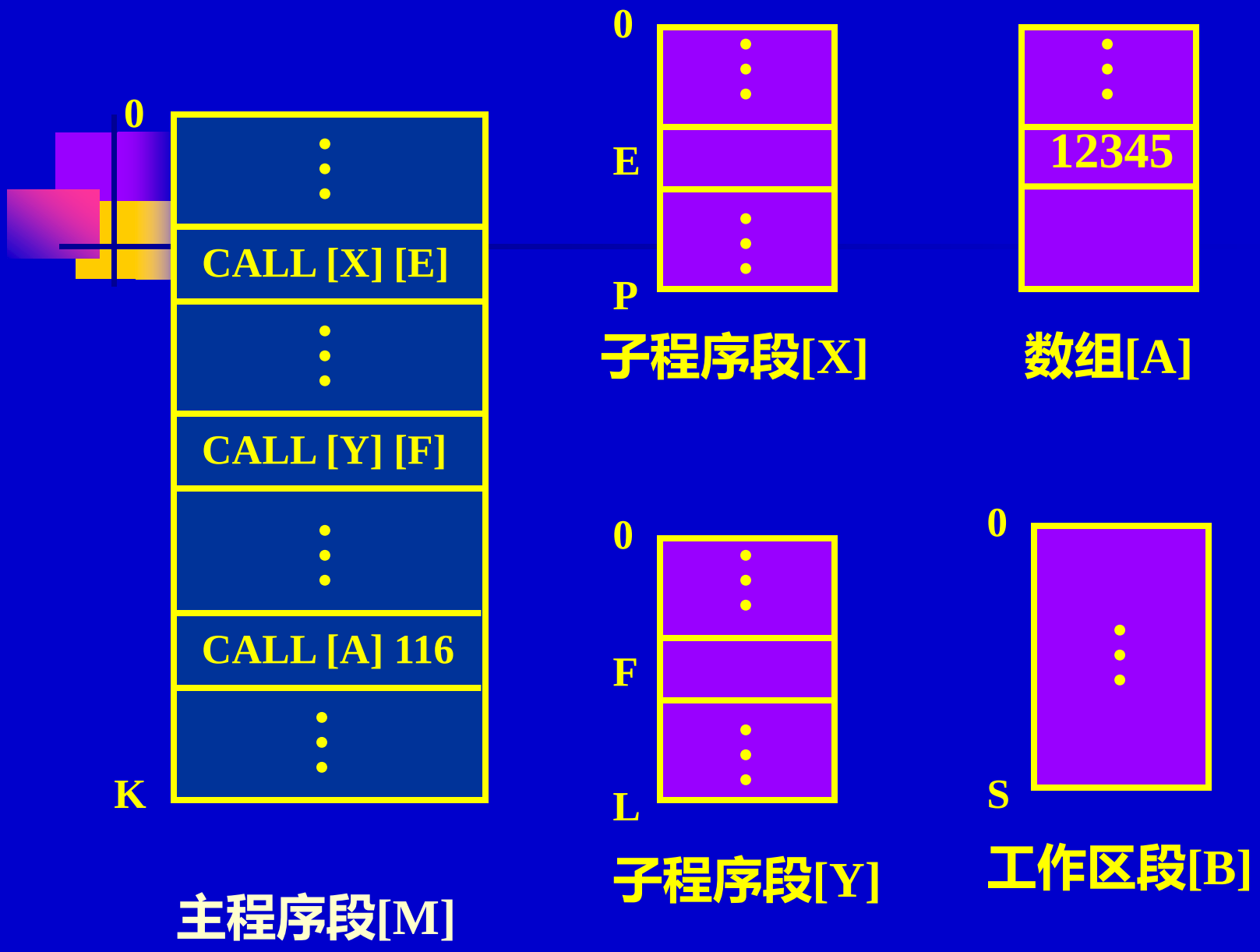
- 优点：解决了碎片问题，便于管理
- 缺点：不易实现共享，不便于动态链接

4.4 段式管理

4.4.1 段式管理基本思想

1、用户程序划分

按程序自身的逻辑关系划分为若干个程序段，每个程序段都有一个段名，且有一个段号。段号从0开始，每一段段内也从0开始编址，段内地址是连续的


0
CALL [X] [E]
CALL [Y] [F]
CALL [A] 116
K
主程序段[M]
0
E
P
子程序段[X]
12345
数组[A]
0
F
L
子程序段[Y]
0
S
工作区段[B]

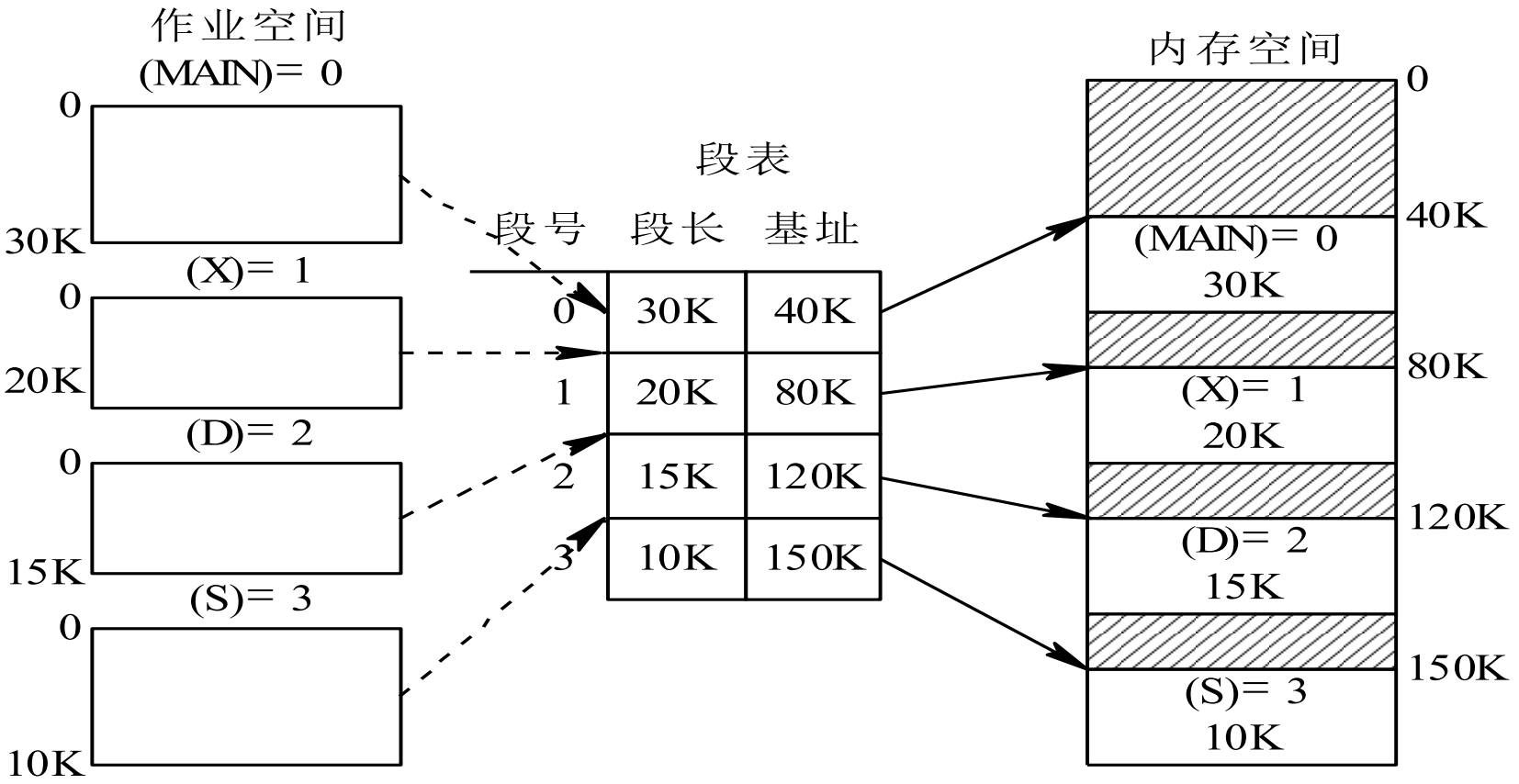
4.4 段式管理（续）

2、逻辑地址结构

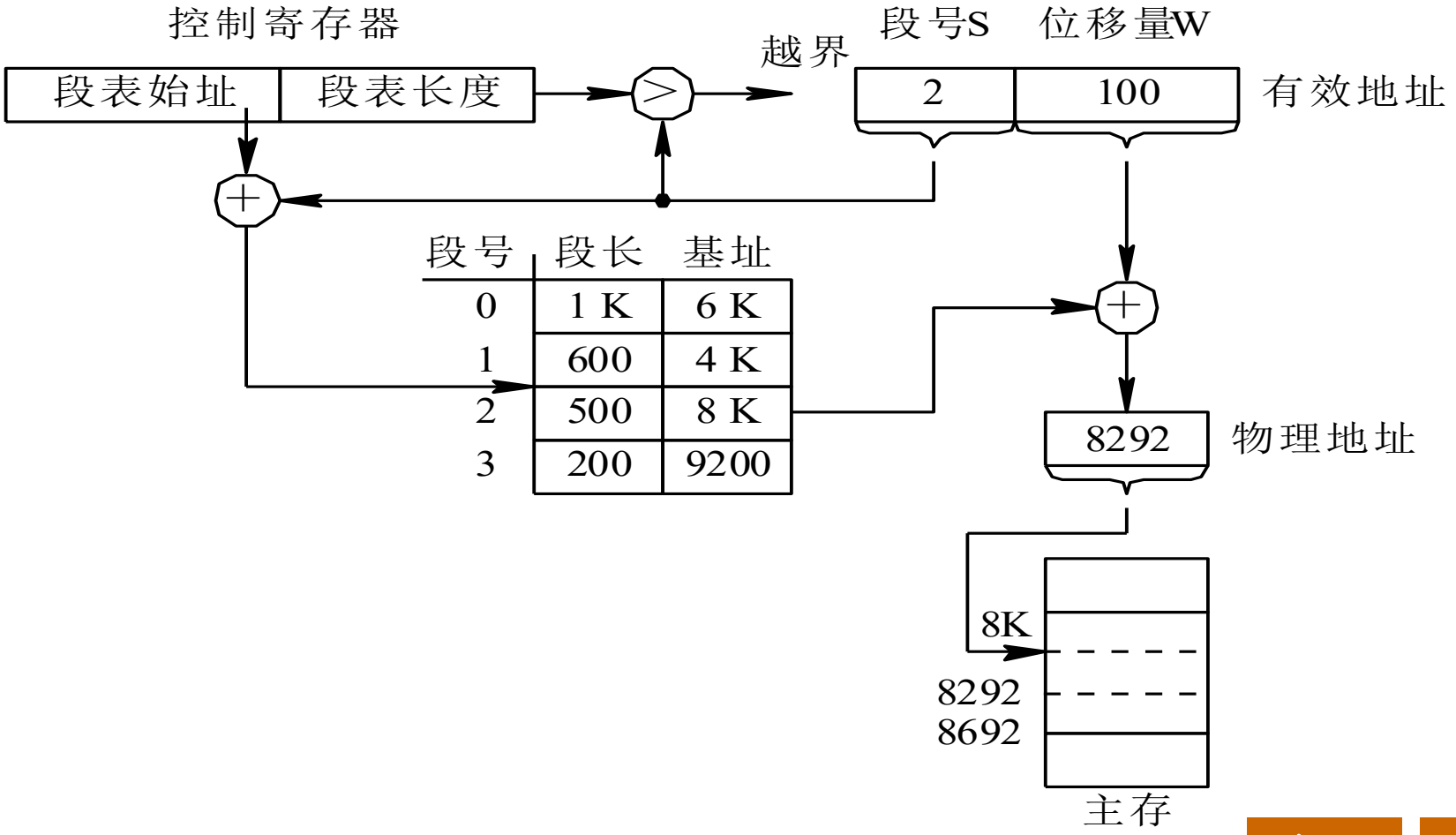
3、内存分配

以段为单位分配内存，每一个段在内存中占据连续空间（内存随机分割，需要多少分配多少），但各段之间可以不连续存放

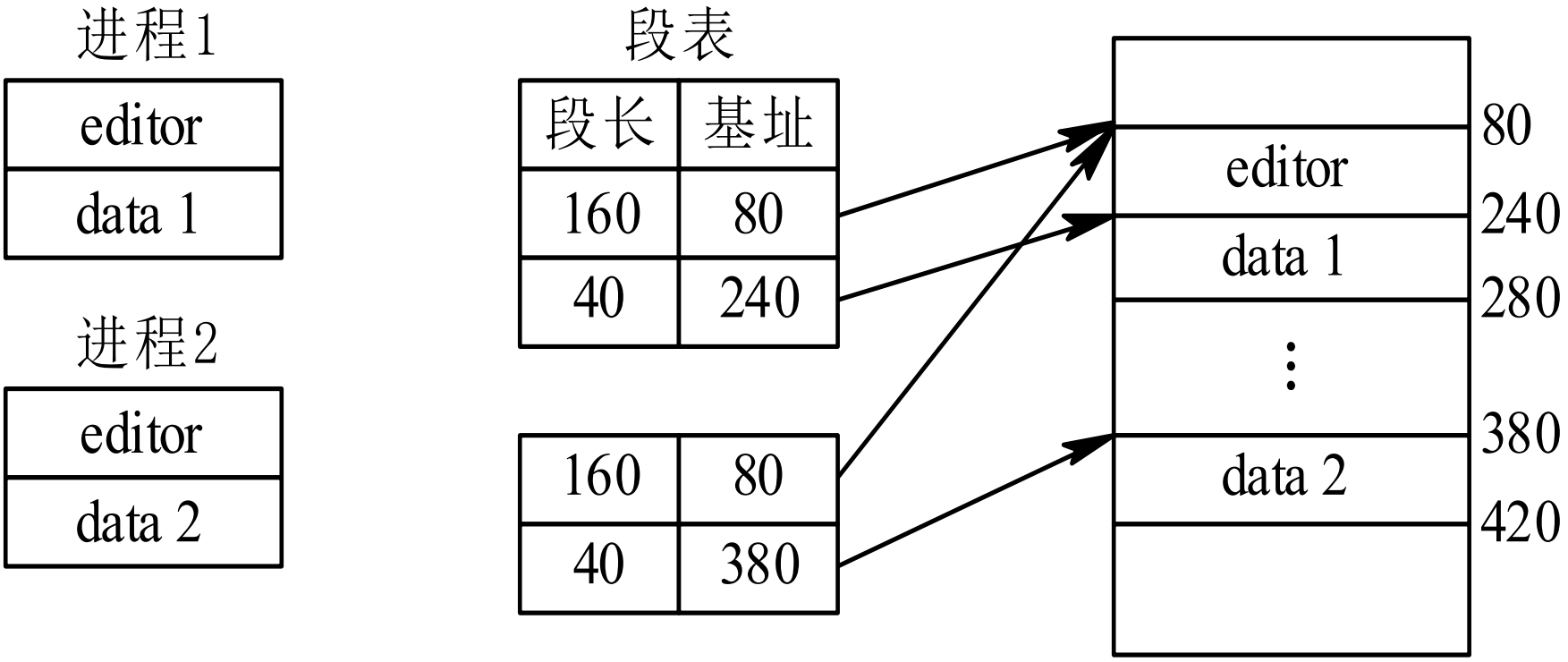
4.4.2 段表


作业空间
(MAIN)= 0
0
30K
(X)= 1
0
20K
(D)= 2
0
15K
(S)= 3
0
10K
段表
段号 段长 基址
0 30K 40K
1 20K 80K
2 15K 120K
3 10K 150K
内存空间
0
40K
(MAIN)= 0
30K
80K
(X)= 1
20K
120K
(D)= 2
15K
150K
(S)= 3
10K

4.4.3 地址转换过程


控制寄存器
段表始址
段表长度
越界
段号S
位移量W
2
100
有效地址
段号
段长
基址
0
1 K
6 K
1
600
4 K
2
500
8 K
3
200
9200
8292
物理地址
8K
8292
8692
主存

4.4.4 段的共享


进程1
editor
data 1
进程2
editor
data 2
段表
段长
基址
160
80
40
240
160
80
40
380
editor
data 1
data 2
80
240
280
380
420

4.4.5 分段与分页的区别

(1) 页是信息的物理单位，分页是为消减内存的外碎片，提高内存的利用率。分页是由于系统管理的需要而不是用户的需要；而段是信息的逻辑单位，它含有一组其意义相对完整的信息。 分段的目的是为了能更好地满足用户的需要。

4.4.5 分段与分页的区别（续）

(2) 页的大小固定且由系统决定，由系统把逻辑地址划分为页号和页内地址两部分，是由机器硬件实现的；而段的长度却不固定， 决定于用户所编写的程序，通常由编译程序在对源程序进行编译时，根据信息的性质来划分。

4.4.5 分段与分页的区别（续）

(3) 分页的作业地址空间是一维的，即单一的线性地址空间； 而分段的作业地址空间则是二维的，程序员在标识一个地址时，既需给出段名，又需给出段内地址。

4.5 段页式管理

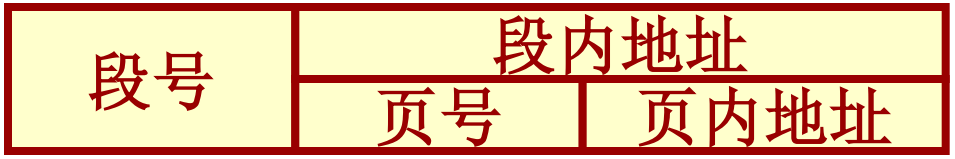
4.5.1 基本原理

1、用户程序划分

先分段，再分页（对用户来讲，按段的逻辑关系进行划分；对系统讲，按页划分每一段）

4.5.1 基本原理（续）

2、逻辑地址结构



3、内存划分：按页式管理进行

4、内存分配：以页为单位进行

4.5.2 地址转换

通过段表和页表实现

- 段表：记录了每一段的页表始址和页表长度
- 页表：记录了逻辑页号与内存块号的对应关系（每一段有一个，一个程序可能有多个页表）

4.5.2 地址转换（续）