Computer

操作系统原理与实践

第三章 内存管理 (2)

高等教育出版社

主要内容

3.5 分页存储管理

3.6 分段存储管理

3.7 虚拟存储器

3.8 请求分页存储管理方式

3.5 分页存储管理

3.5.1 页式存储管理的引入

- 在动态分区的存储空间中， 存在“零头”问题。尽管采用“紧凑”技术可以解决这个问题，但要为移动大量信息花去不少的处理机时间，代价较高。

3.5.2 页面与页表

1. 页面和物理块

分页存储管理，是将一个进程的逻辑地址空间分成若干个大小相等的片，称为**页面或页**，并为各页加以编号。相应地，也把内存空间分成与页面相同大小的若干个存储块，称为(物理)**块或页框(frame)**。

1. 页面和物理块

- 在为进程分配内存时，以块为单位将进程中的若干个页分别装入到多个可以不相邻接的物理块中。由于进程的最后一页经常装不满一块而形成了不可利用的碎片，称之为“页内碎片”。

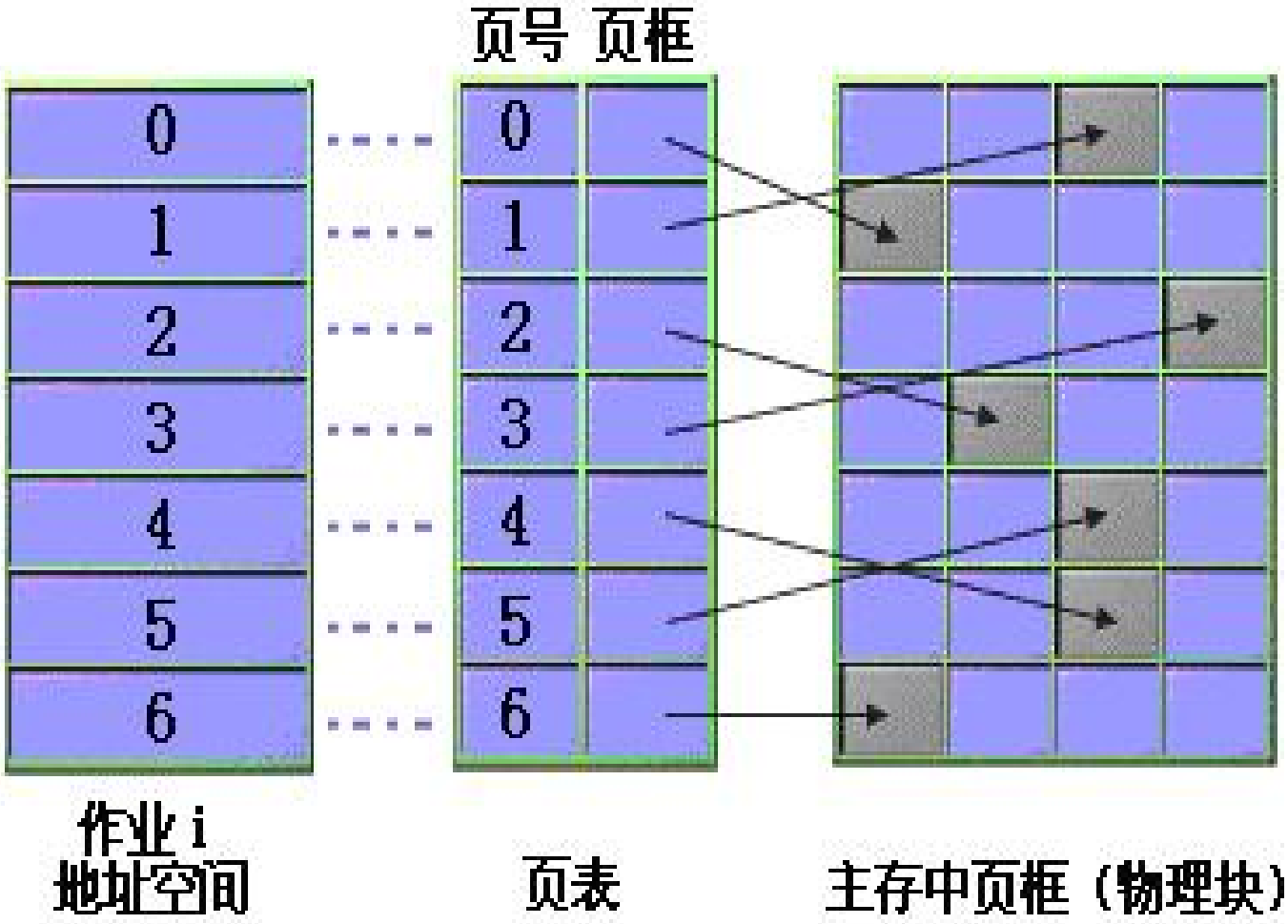
2. 页表

- 列出了作业的逻辑地址与其在主存中的物理地址间的对应关系。
- 页面大小：页面的大小应选择得适中，且页面大小应是2的幂，通常为512 B~8 KB。

2. 页表

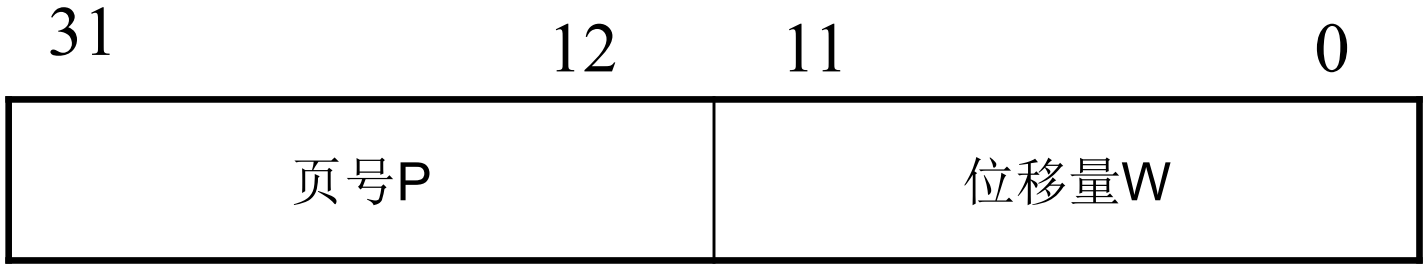
- 一个页表中包含若干个表目，表目的自然序号对应于用户程序中的页号，表目中的块号是该页对应的物理块号。
- 页表的每一个表目除了包含指向页框的指针外，还包括一个存取控制字段。
- 表目也称为页描述子。

分页管理中页与页框的对应关系示意图



3. 地址结构

分页地址中的地址结构如下：



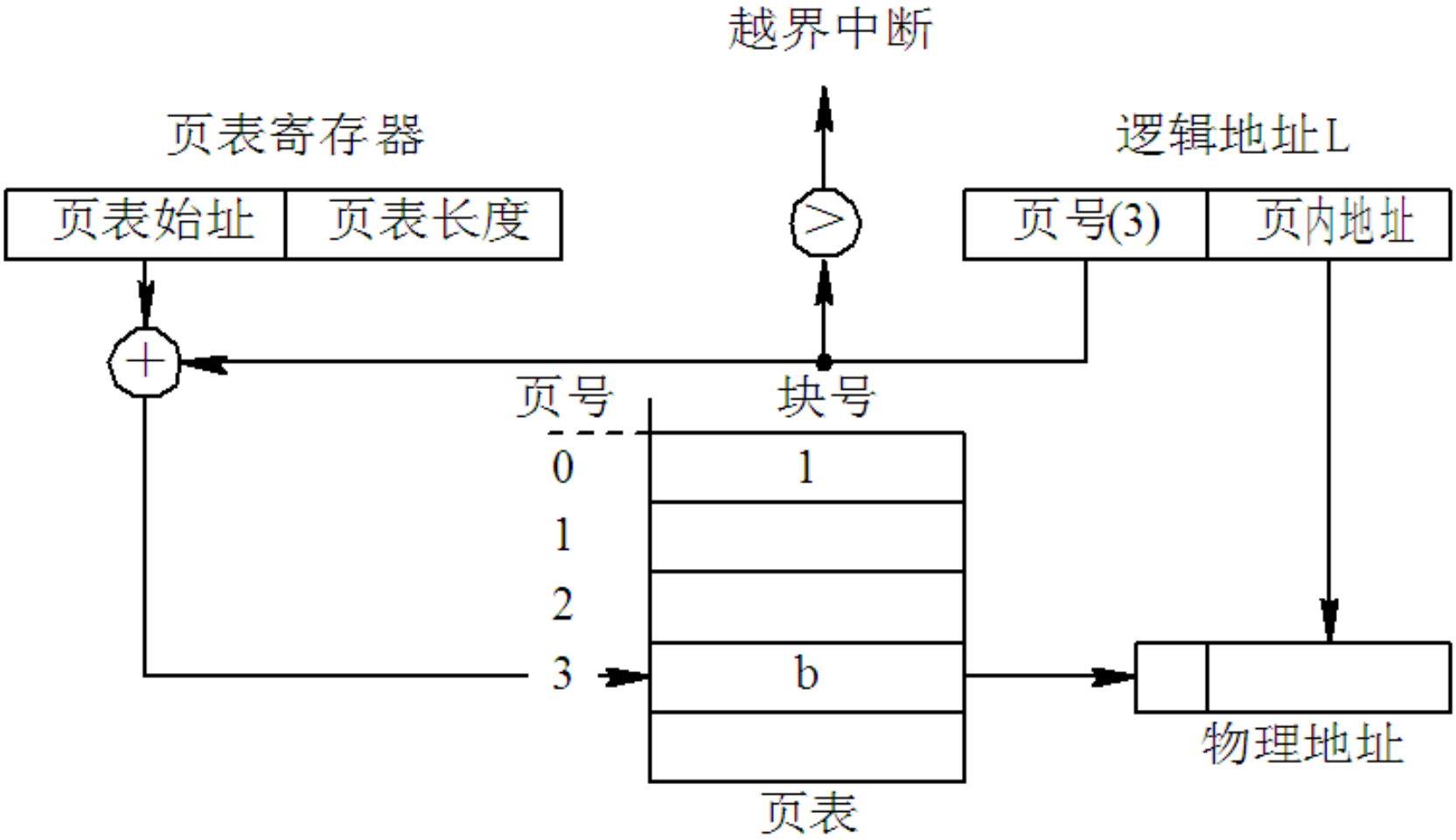
对某特定机器，其地址结构是一定的。若给定一个逻辑地址空间中的地址为**A**，页面的大小为**L**，则页号**P**和页内地址**d**可按下式求得：

$$P = INT\left[\frac{A}{L}\right]$$

$$d = [A]MODL$$

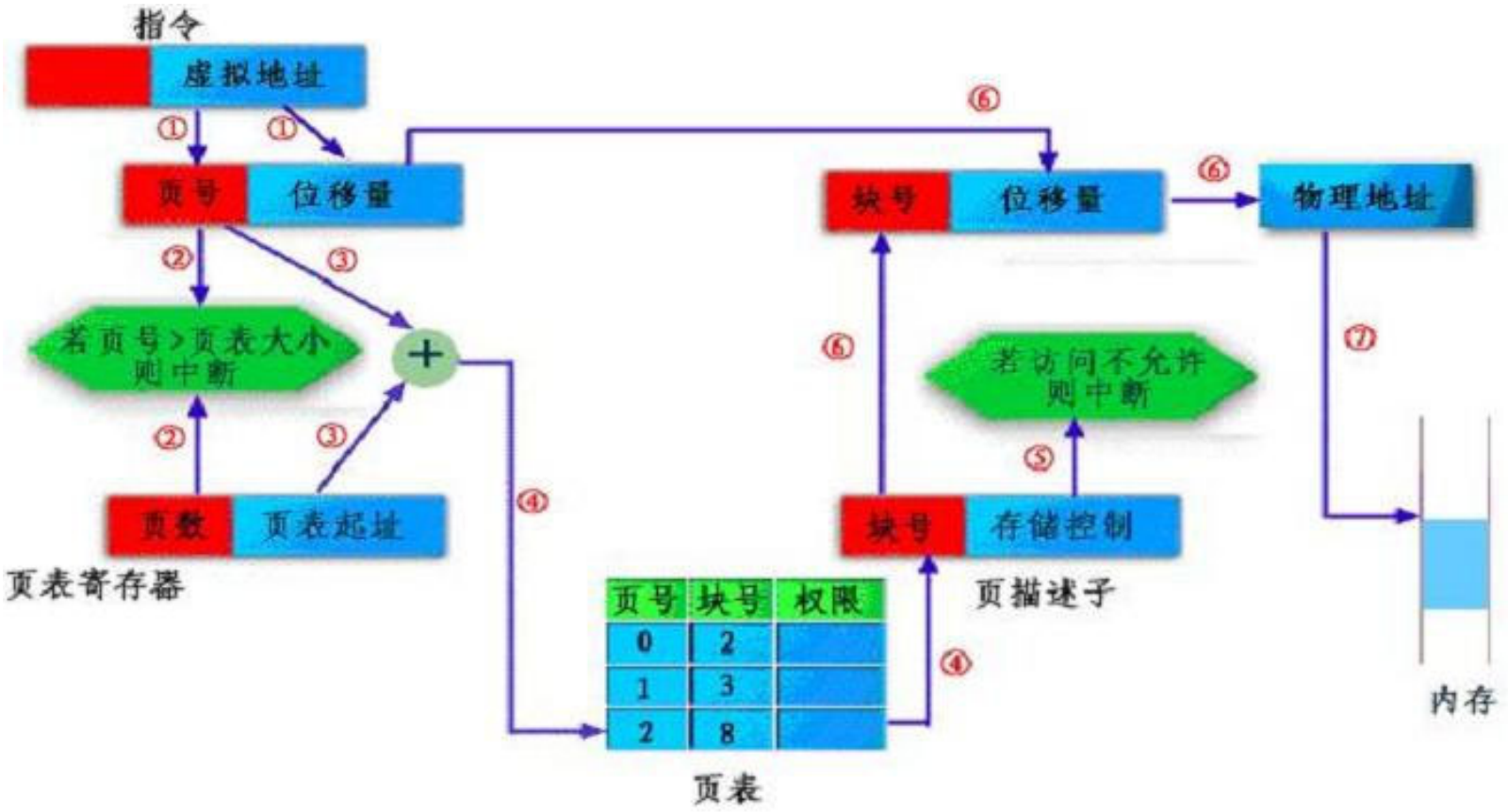
3.5.3 地址变换机构

1. 基本的地址变换机构



2. 地址变换过程

- 例如指令 LOAD 1,2500 的地址变换过程如下：



地址变换过程（续）

- 把虚拟地址2500转换成页号P=2，位移量W=452；
- 如果页号2大于页表大小，则中断；否则继续；
- 页号2与页表起址1000运算（1000+2*20，设页描述子大小为20）得到页描述子地址为1040；
- 从页描述子中读取块号8；
- 根据页描述子的“存取控制”判断该指令是否被允许访问内存，如果不允许，则中断；否则继续；
- 块号8与位移量452运算（8*1024+452=8644，1024为页面大小）得到物理地址8644；
- 把内存地址8644单元中数字写进寄存器1。

3. 管 理

1）页表

- 系统为每个进程建立一个页表，页表 给出逻辑页号和具体内存块号相应的关系

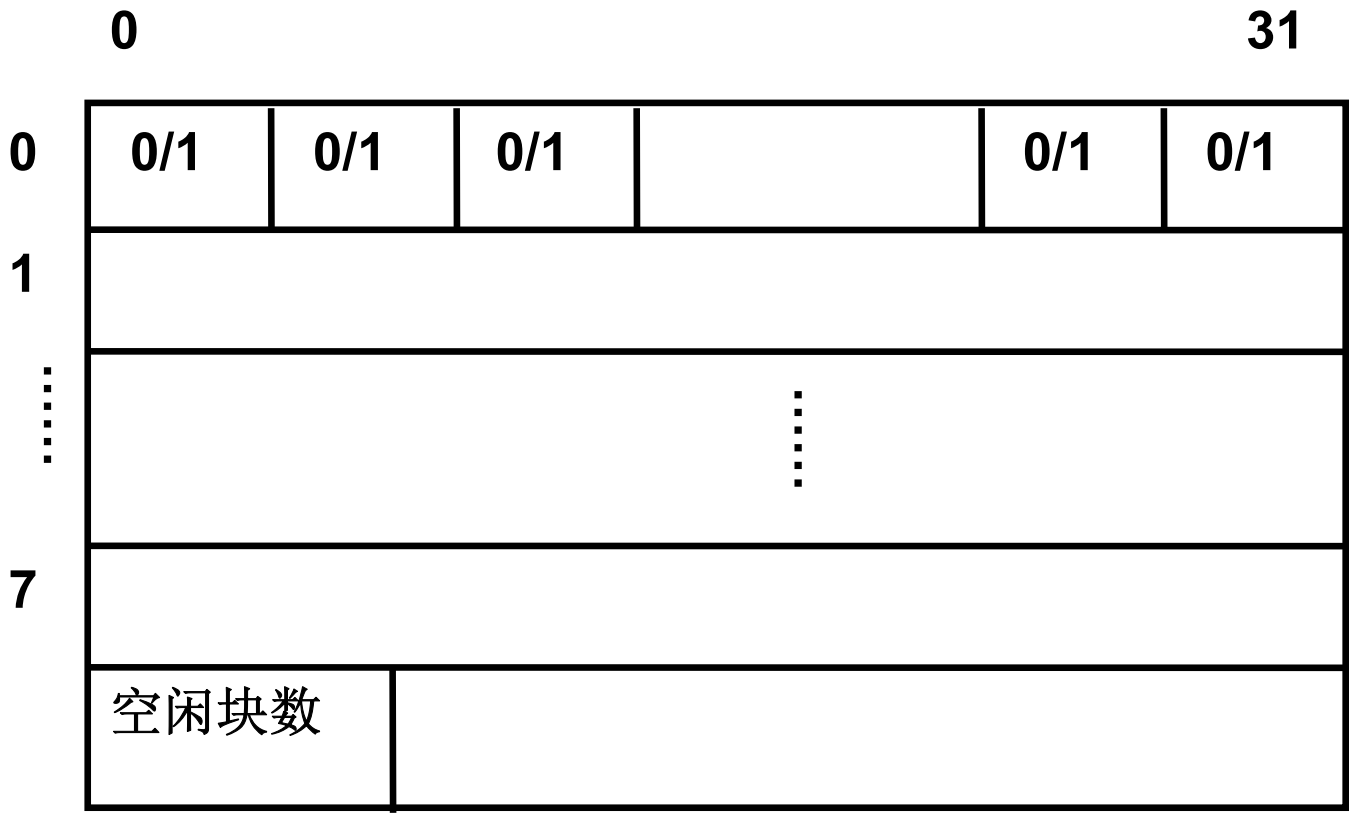
  页表放在内存，属于进程的现场信息

2） 空块管理

- 位示图
- 内存页表

3） 内存的分配与回收

空块管理——位示图



内存的分配

- 计算一个作业所需要的总块数N
- 查位示图，看看是否还有N个空闲块
- 如果有足够的空闲块，则页表长度设为N，可填入PCB中；申请页表区，把页表始址填入PCB
- 依次分配N个空闲块，将块号和页号填入页表
- 修改位示图

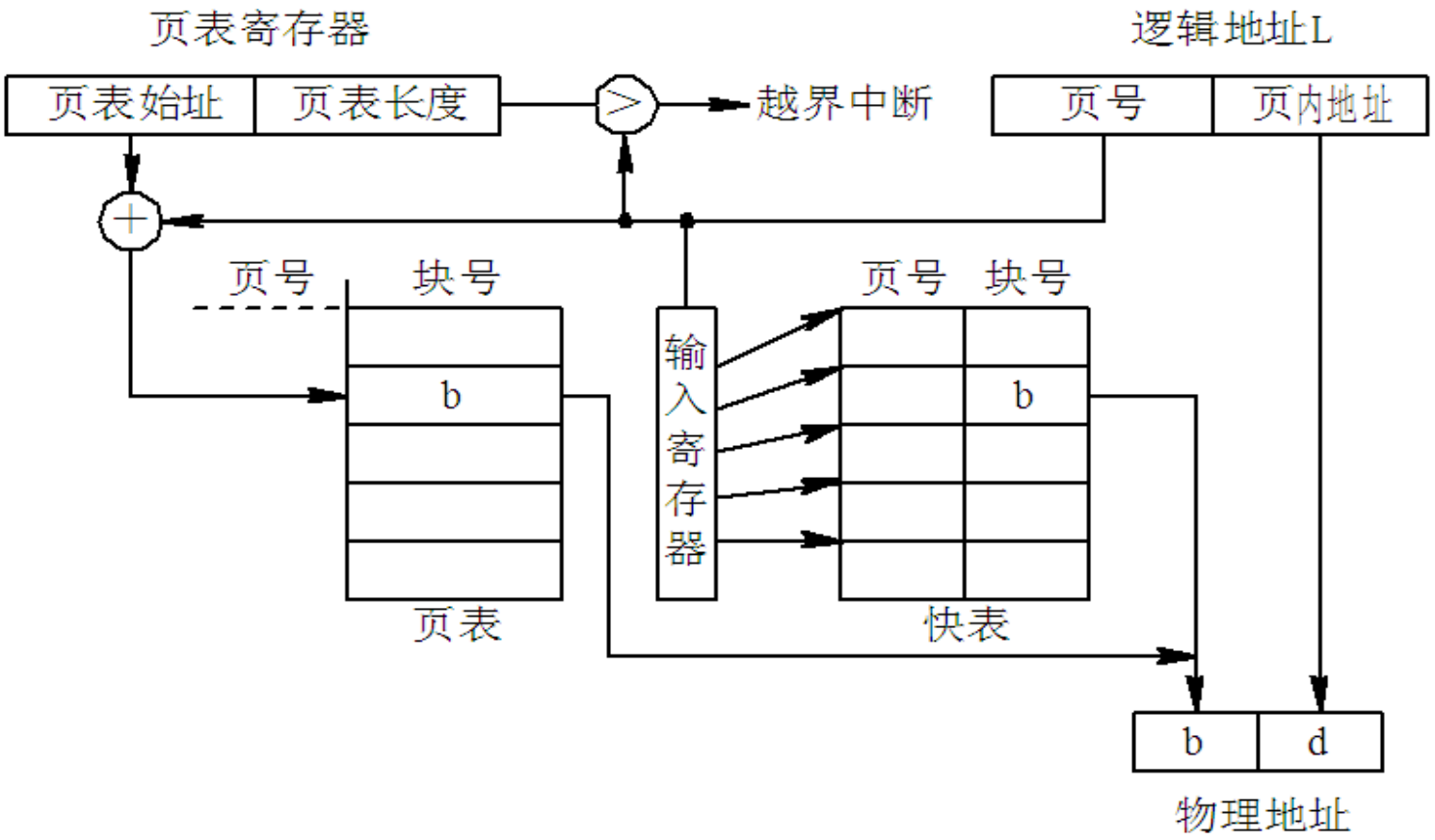
4. 快表

如果把页表放在主存中，无疑会影响系统的性能。这是因为每次访问主存，首先必须访问页表，读出页描述子，之后根据形成的实际地址再访问主存，这样使访问主存的次数加倍，因而使总的处理速度明显下降。

为了解决这个问题人们采用一组硬件寄存器，存放当前访问过的页的页描述子，

每次访问主存时，首先查找快表，若找到所需的页描述子，则快速形成物理地址。否则从页表中查找后形成物理地址，同时把页描述子写入快表。如果设计得当，快表的命中率可以很高。

具有快表的地址变换机构



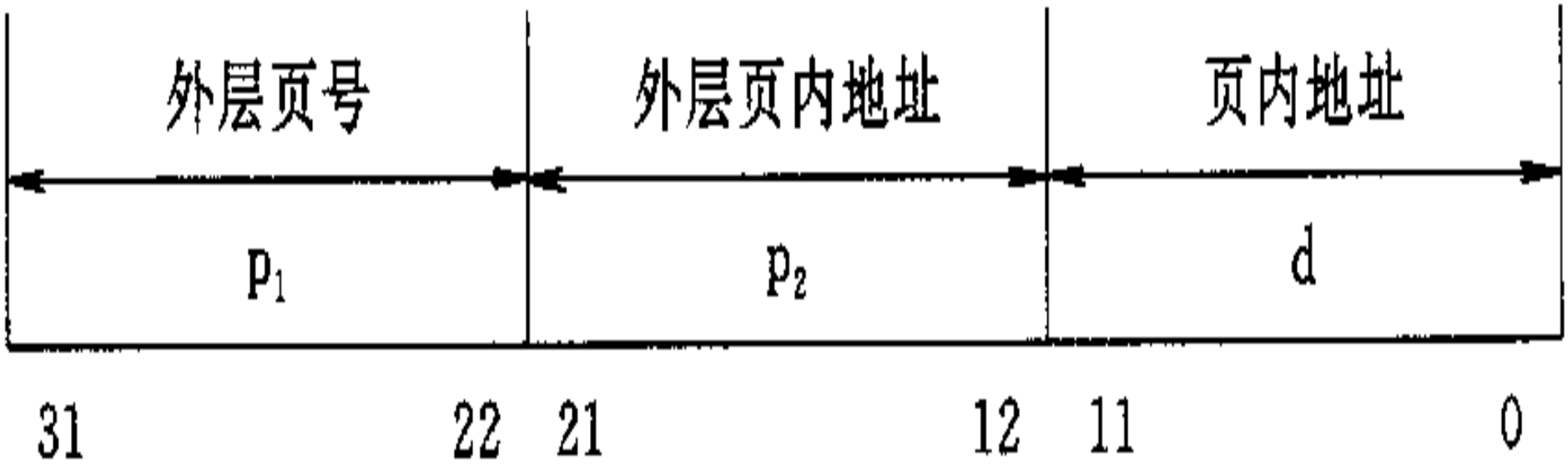
具有快表的地址变换机构

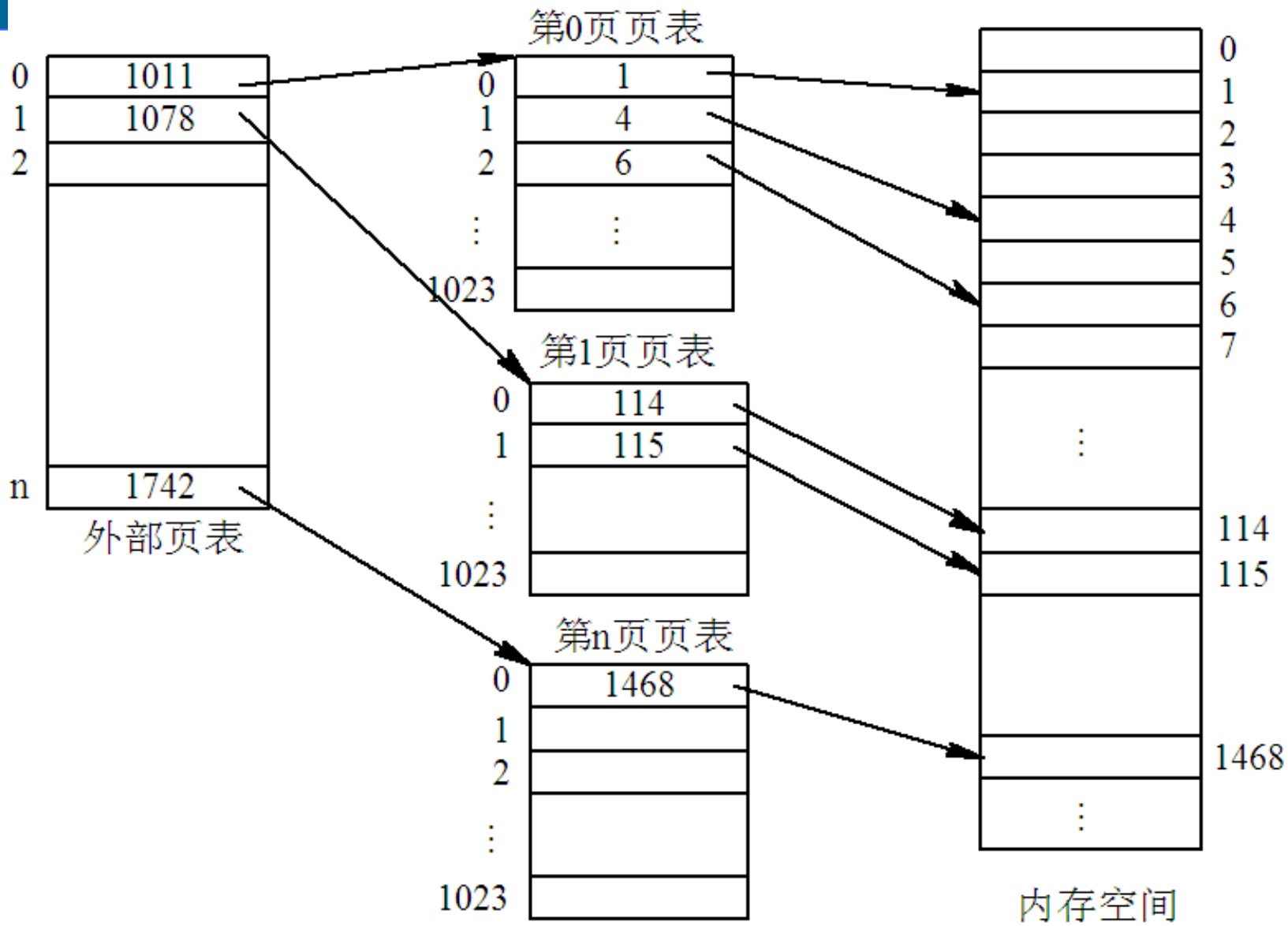
3.5.4 两级和多级页表

现代的大多数计算机系统，都支持非常大的逻辑地址空间(2^{32}~2^{64})。页表就变得非常大，要占用相当大的内存空间。可采用两个方法来解决这一问题：① 采用离散分配方式来解决难以找到一块连续的大内存空间的问题；② 只将当前需要的部分页表项调入内存，其余的页表项仍驻留在磁盘上，需要时再调入。

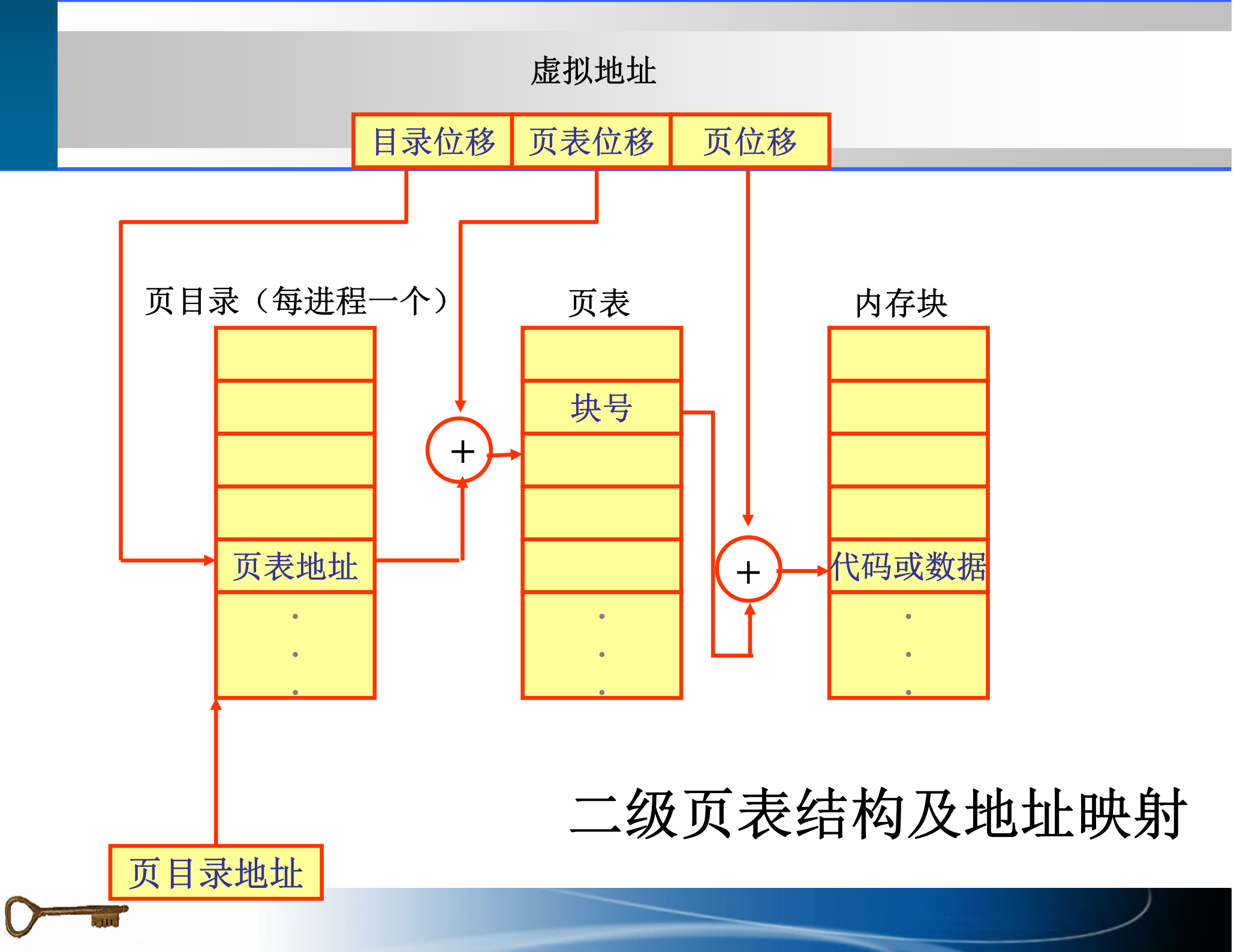
1. 两级页表

逻辑地址结构可描述如下：




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
1
4
6
114
115
1468
1023
0
1
2
3
4
5
6
7
114
115
1468

两级页表结构



二级页表结构及地址映射

2.多级页表

- 32位　进程页表多大？

  设：页面大小为4K

  　　用户空间 4GB

  则：一个进程 1M页

  设：每个内存块号 4字节

  则：进程页表 1K页

- 结论：一个进程的页表的各页之间可以不连续存放
- 解决：页表本身使用地址索引——页目录

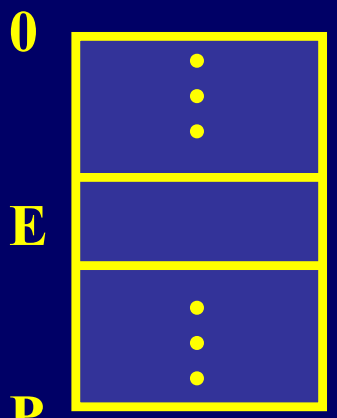
3.6 分段存储管理

3.6.1 分段式存储管理的引入

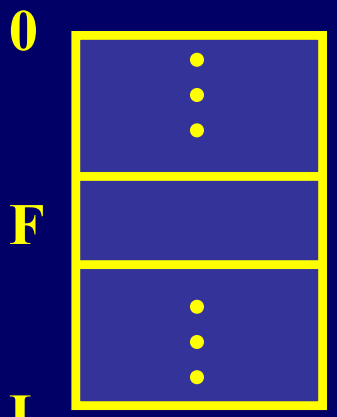
在分页存储系统中，作业的地址空间是一维线性的，这破坏了程序内部天然的逻辑结构，造成共享、保护的困难。引入分段存储管理方式，主要是为了满足用户和程序员的下述需要：

1) 方便编程
2) 信息共享
3) 信息保护
4) 动态增长
5) 动态链接

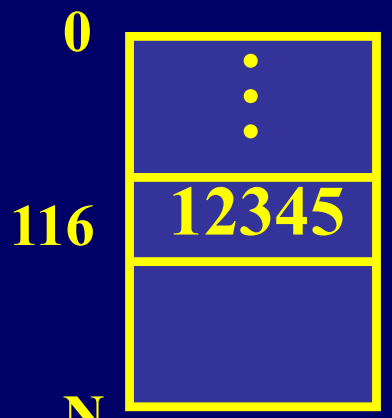
主程序段[M]



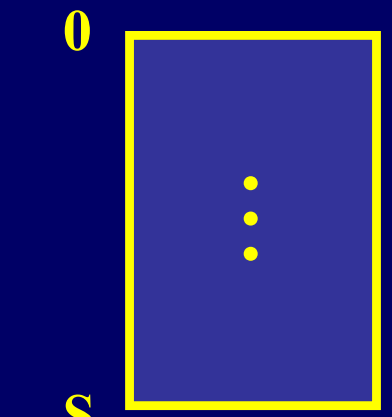
子程序段[X]



子程序段[Y]



数组[A]

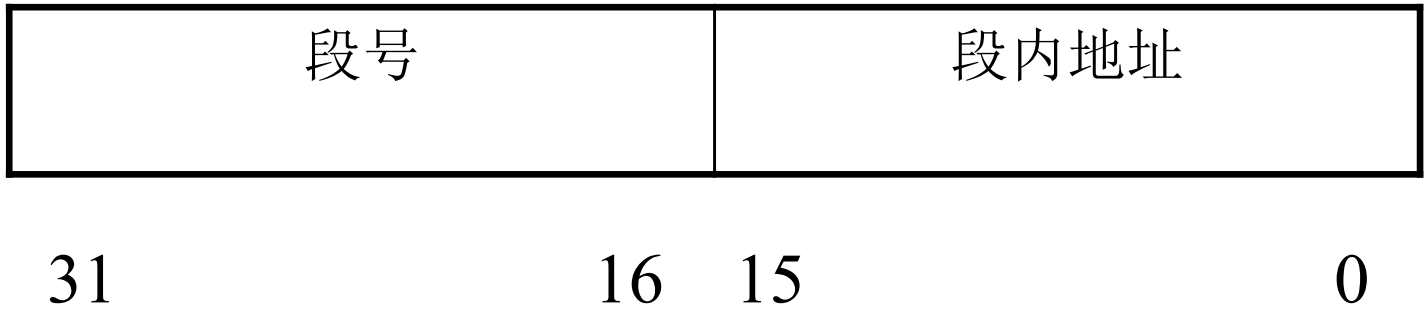


工作区段[B]

3.6.2　分段系统的基本原理

1. 分段

分段地址中的地址具有如下结构：



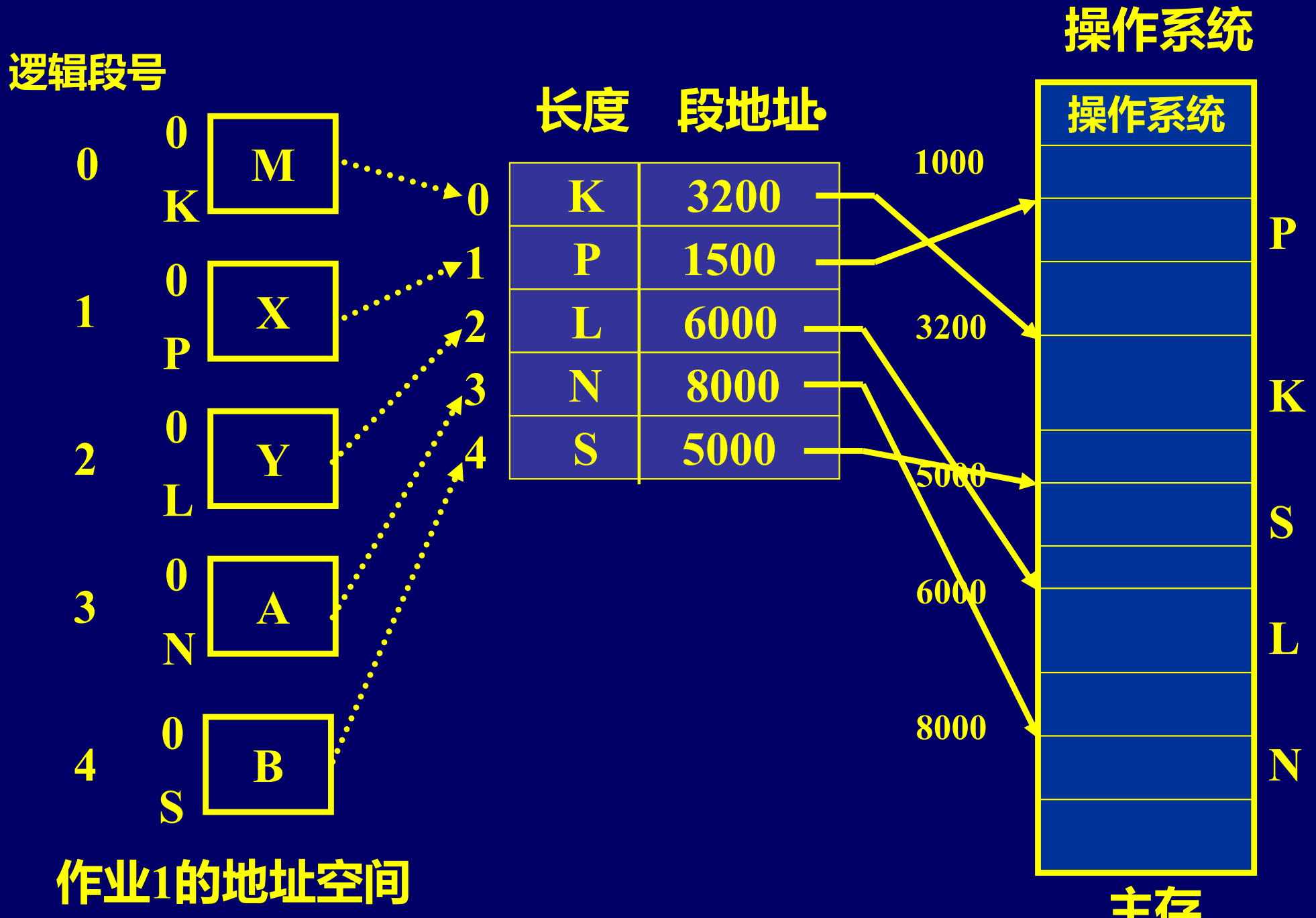
2. 段表

段号	段首址	段长度
0	58K	20K
1	100K	110K
2	260K	140K

它记录了段号，段的首（地）址和长度之间的关系

每一个程序设置一个段表，放在内存，属于进程的现场信息

分段管理中作业i与段表、存储空间的关系



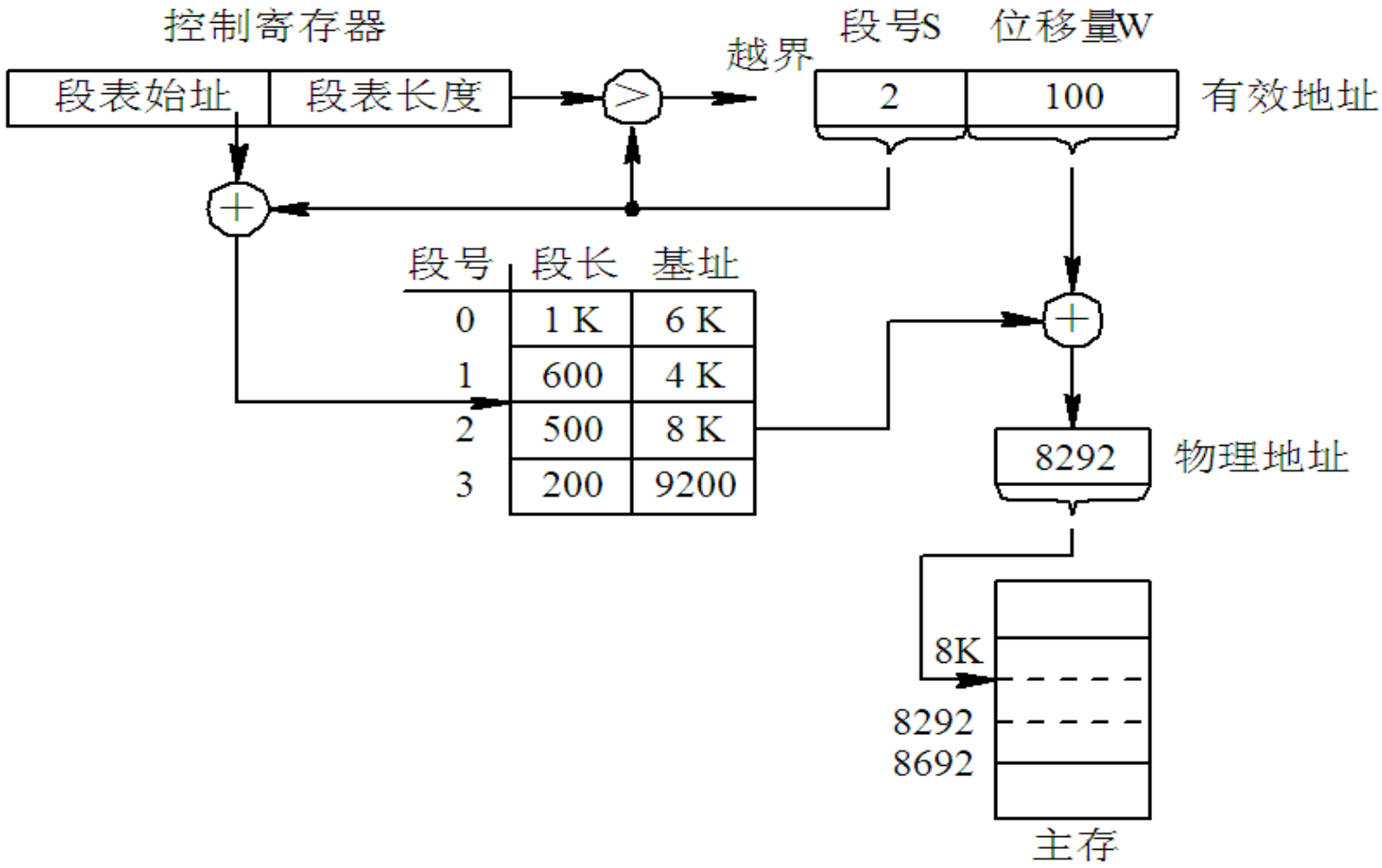
3. 硬件支持

- 系统设置一对寄存器
- 段表始址寄存器：

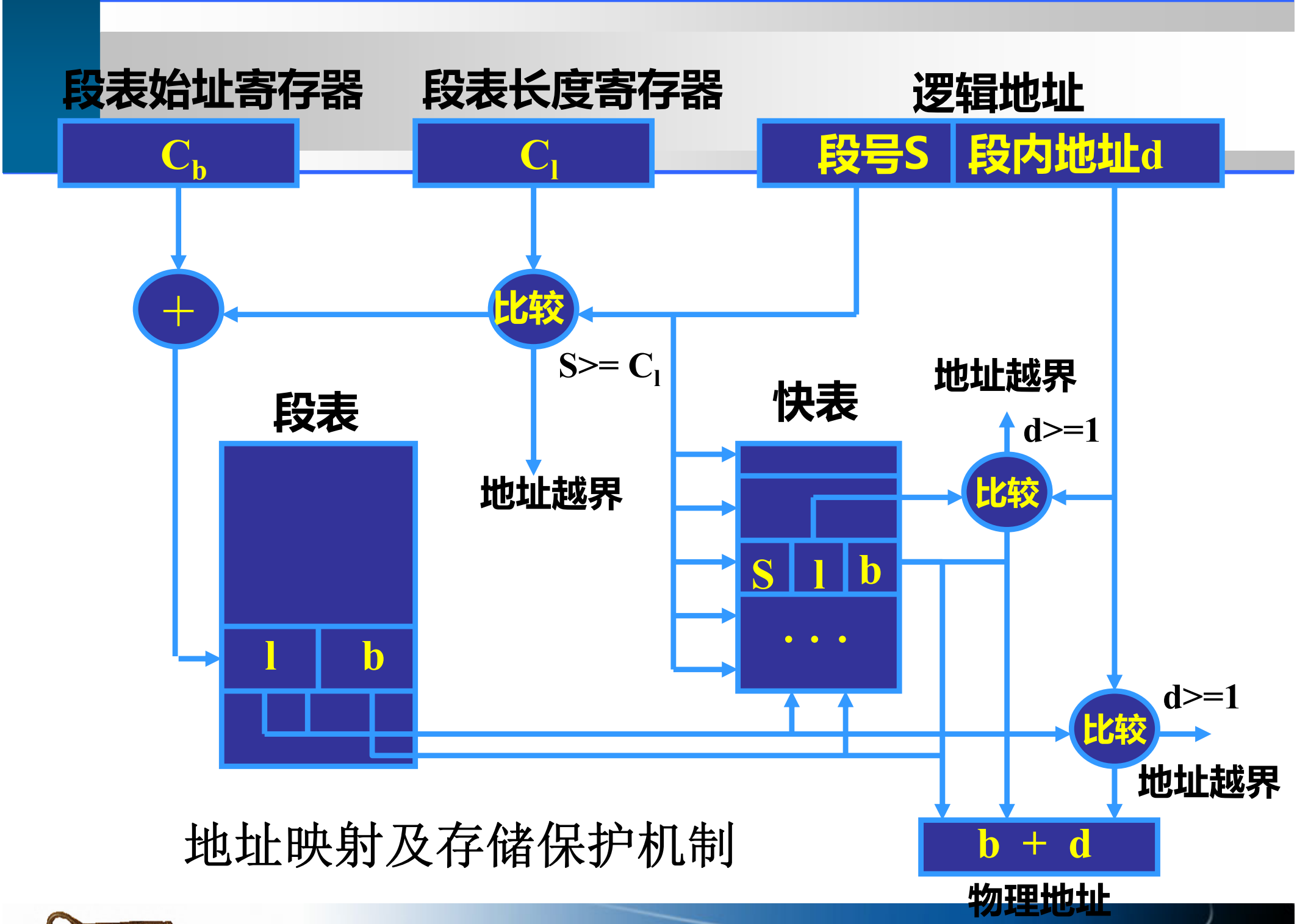
  用于保存正在运行进程的段表的始址
- 段表长度寄存器：

  用于保存正在运行进程的段表的长度（例如上图的段表长度为3）

4. 地址变换机构



分段系统的地址变换过程



地址映射及存储保护机制

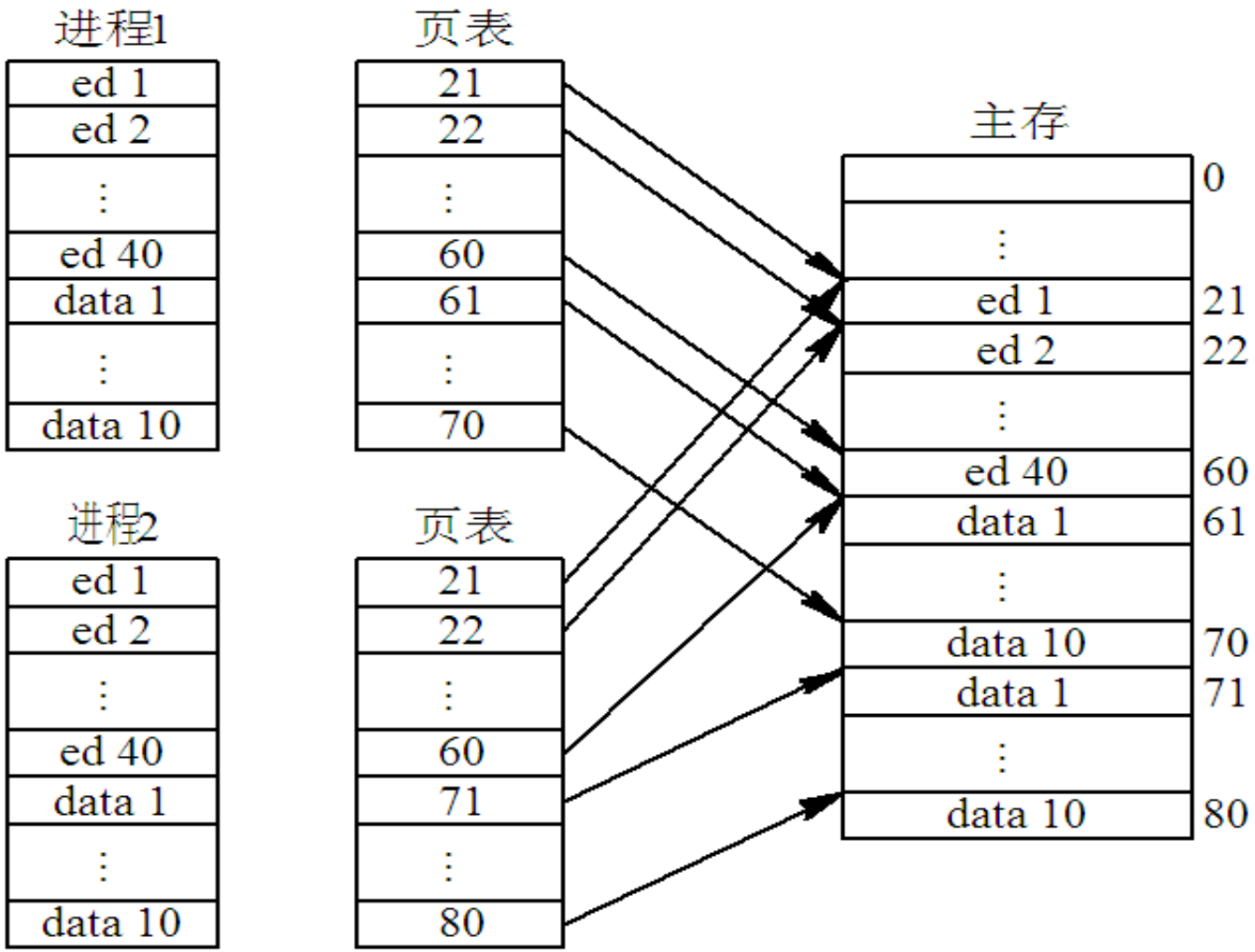
5. 分页和分段的主要区别

(1) 页是信息的物理单位，段则是信息的逻辑单位；

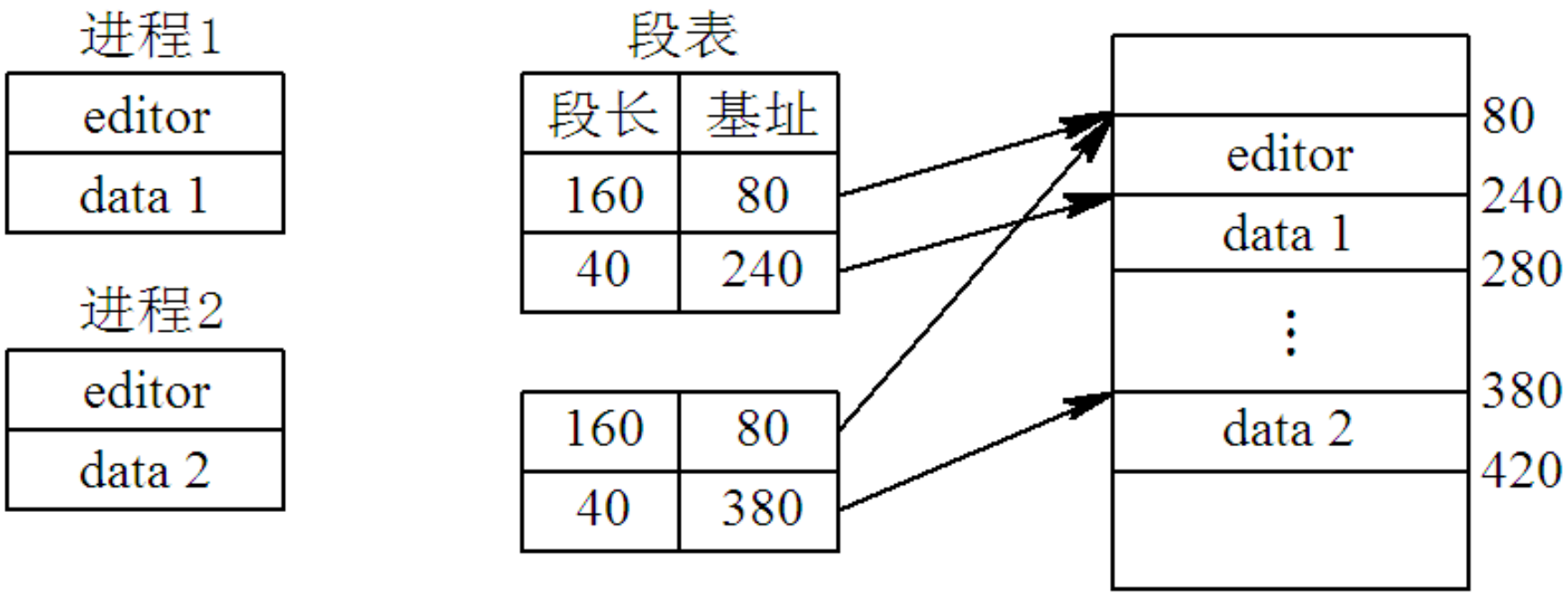
(2) 页的大小固定且由系统决定，而段的长度却不固定；

(3) 分页的作业地址空间是一维的，即单一的线性地址空间，分段的作业地址空间则是二维的。

3.6.3 信息共享



分页系统中共享editor的示意图



分段系统中共享**editor**的示意图

分段管理的优缺点

优点：

便于动态申请内存
管理和使用统一化
便于共享
便于动态链接

缺点：

产生碎片

思考：与可变分区存储管理方案的相同点与不同点？

3.7 段页存储管理

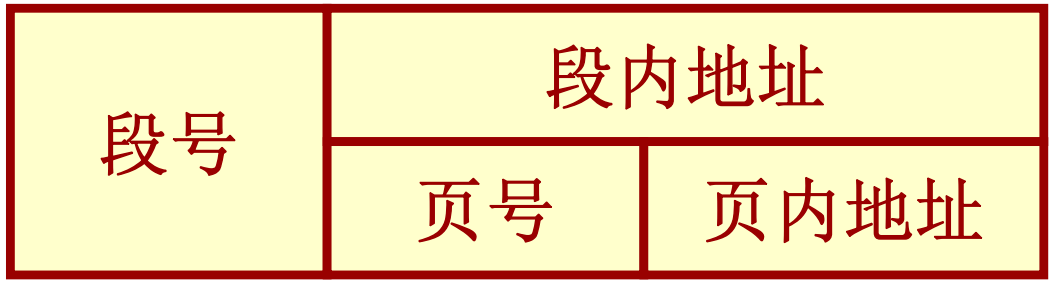
1.产生背景:

结合了段式与页式二者优点,克服了二者的缺点。

2. 基本思想

用户程序划分：按段式划分（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）

逻辑地址：



内存划分：按页式存储管理方案

内存分配：以页为单位进行分配

3. 管 理

1. 段表：记录了每一段的页表始址和页表长度
2. 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）
3. 空块管理：同页式管理
4. 分配：同页式管理

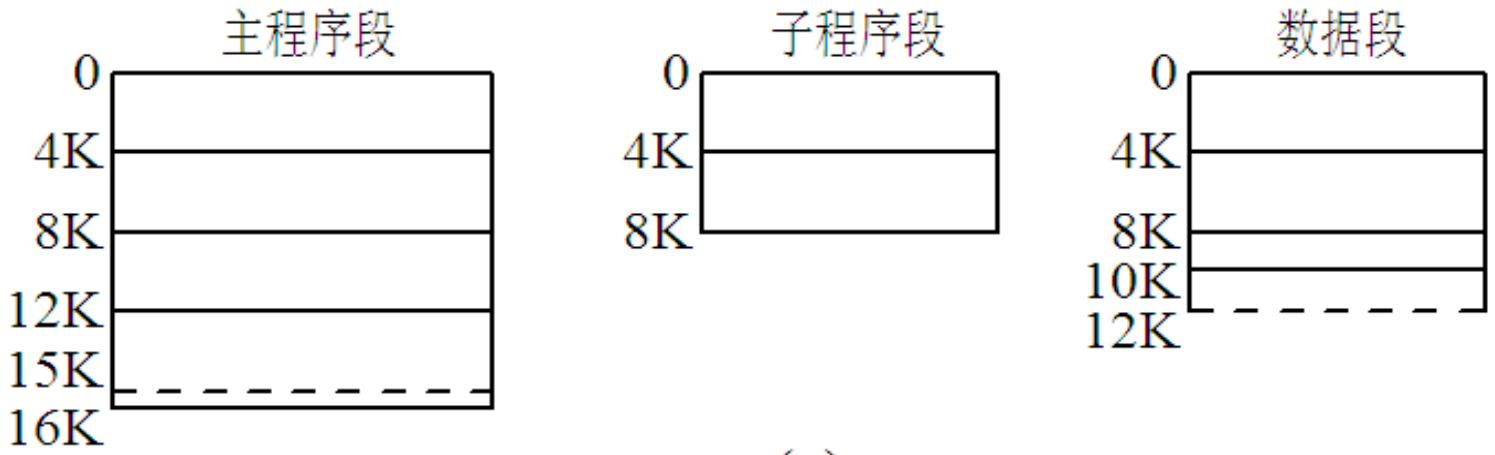
4. 地址空间和地址结构

一个程序首先被划分成若干程序段，每一段给予不同的分段标识符然后，对每一分段又分成若干个固定大小的页面。如下图(a)所示，系统中的一个作业的地址空间结构页面尺寸为4 K字节，该作业有三个分段，第一段为**15K**字节，占**4**页，最后一页只有**1K**未用；其它段同理。未足一页大小的补为一页。

4. 地址空间和地址结构

1. 作业地址空间:如图（a）所示

2. 地址结构如图（b）所示

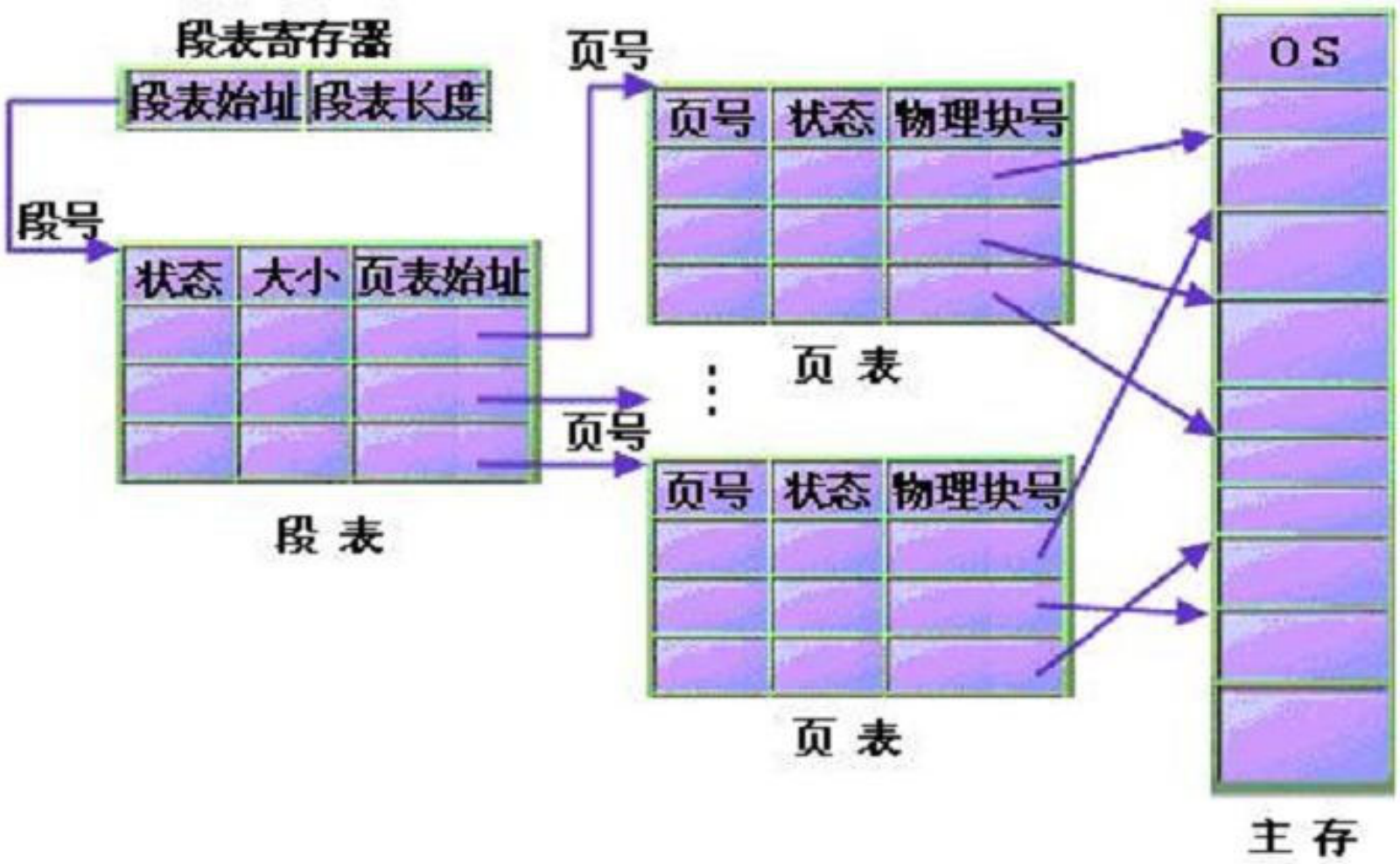


(*a*)

段号(S)	段内页号(P)	段内地址(W)

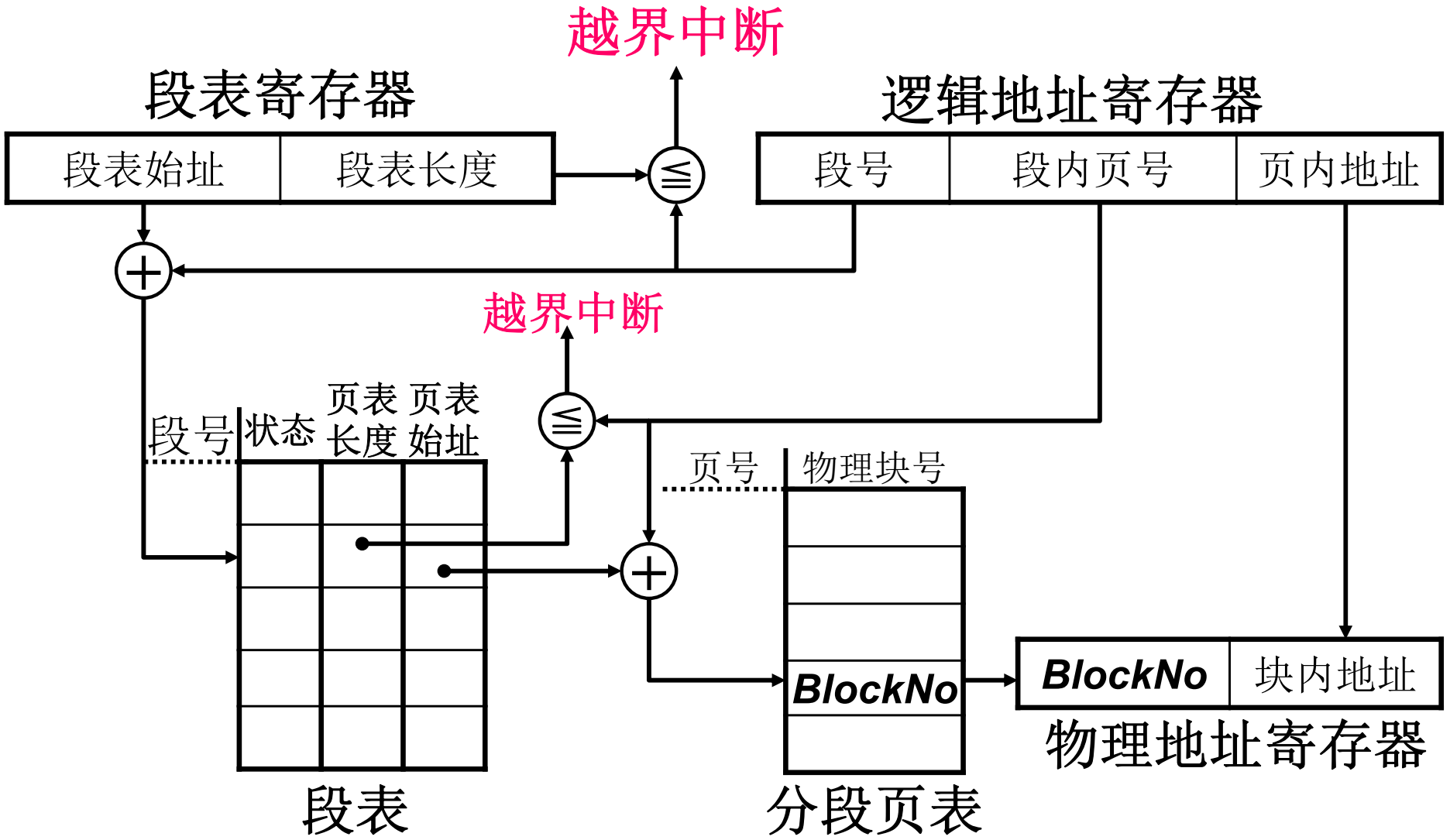
(*b*)

5. 地址映射


段表寄存器
段表始址
段表长度
段号
状态
大小
页表始址
段 表
页号
页号
状态
物理块号
页 表
页号
页号
状态
物理块号
页 表
OS
主 存

6. 地址变换

- 段页式系统的地址变换结构



地址变换（续）

- 从控制寄存器读取段表始址，找到段表；
- 段号＋段表始址 得到段描述子地址；
- 从段描述子读取页表始址，找到页表；
- 页号＋页表始址 得到页描述子地址；
- 从页描述子读取物理块号；
- 物理块号＋页内位移量 得到物理地址。

上述的地址变换至少要访问主存三次，这将使执行程序的速度大大降低。为了解决上述问题，可以采取前边讲过的“快表”技术。

3.8交换与覆盖

1. 交换技术与覆盖技术

- 在多道环境下扩充内存的方法，用以解决在较小的存储空间中运行较大程序时遇到的矛盾
- 覆盖技术主要用在早期的操作系统中
- 交换技术被广泛用于小型分时系统中，交换技术的发展导致了虚存技术的出现

2. 交换与覆盖异同点

- 共同点：

  进程的程序和数据主要放在外存，当前需要执行的部分放在内存，内外存之间进行信息交换

- 不同点：如何控制交换？

3. 覆盖技术

- 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）
- 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个存储空间
- 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，然后由由操作系统完成自动覆盖

4. 覆盖技术的缺点

- 对用户不透明，增加了用户负担
- 目前这一技术用于小型系统中的系统程序的内存管理上
- MS-DOS的启动过程中，多次使用覆盖技术；启动之后，用户程序区TPA的高端部分与COMMAND.COM暂驻模块也是一种覆盖结构

5. 交换技术

当内存空间紧张时，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，这种技术是进程在内存与外存之间的动态调度。多用于分时系统中

6. 交换技术实现中的几个问题

1. 选择原则:将哪个进程换出/内存?
2. 交换时机的确定
3. 交换时需要做哪些工作?
4. 换入回内存时位置的确定

7. 选择原则

- 将哪个进程换出/内存？
- 例子：分时系统，时间片轮转法或基于优先数的调度算法，在选择换出进程时，要确定换出的进程是要长时间等待的
- 需要特殊考虑的是：任何等待I/O进程中存在的问题
- 解决：从不换出处于等待I/O状态的进程

  有些I/O进程因DMA而不能换出内存或换出前需要操作系统的特殊帮助

8. 交换时机的确定

何时需发生交换？

例子：

- 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

9. 交换时需要做哪些工作

需要一个盘交换区：必须足够大以存放所有用户程序的所有内存映像的拷贝；必须对这些内存映像的直接存取

10. 换入回内存时位置的确定

- 换出后再换入的内存位置一定要在换出前的原来位置上吗？
- 受地址“绑定”技术的影响，即绝对地址产生时机的限制

11. 覆盖与交换的比较

- 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；而且，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。此外，覆盖只能覆盖那些与覆盖段无关的程序段

3.9 虚拟存储器

1. 概 述

问题的提出 ：

- 程序大于内存
- 程序暂时不执行或运行完是否还要占用内存

虚拟存储器的基本思想是：程序、数据、堆栈的大小可以超过内存的大小，操作系统把程序当前使用的部分保留在内存，而把其它部分保存在磁盘上，并在需要时在内存和磁盘之间动态交换

虚拟存储器支持多道程序设计技术

2. 局部性原理

程序局部性原理

在一段时间内一个程序的执行往往呈现出高度的局部性，表现在时间与空间两方面

- 时间局部性：

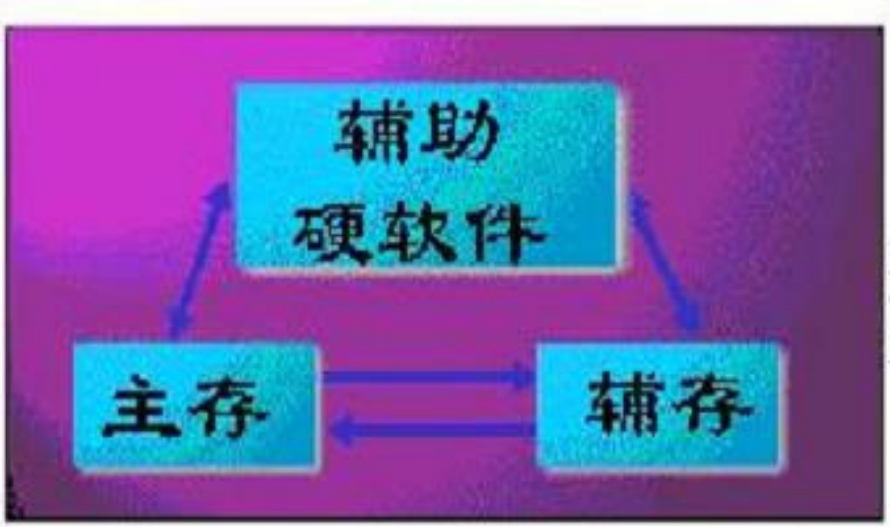
  一条指令被执行了，则在不久的将来它可能再被执行。

- 空间局部性：

  若某一存储单元被使用，则在一定时间内，与该存储单元相邻的单元可能被使用。

3. 虚拟存储器

- 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。虚拟存储器就是一个地址空间，且具有比实存大得多的容量。



从整体看
速度是主存
容量是辅存

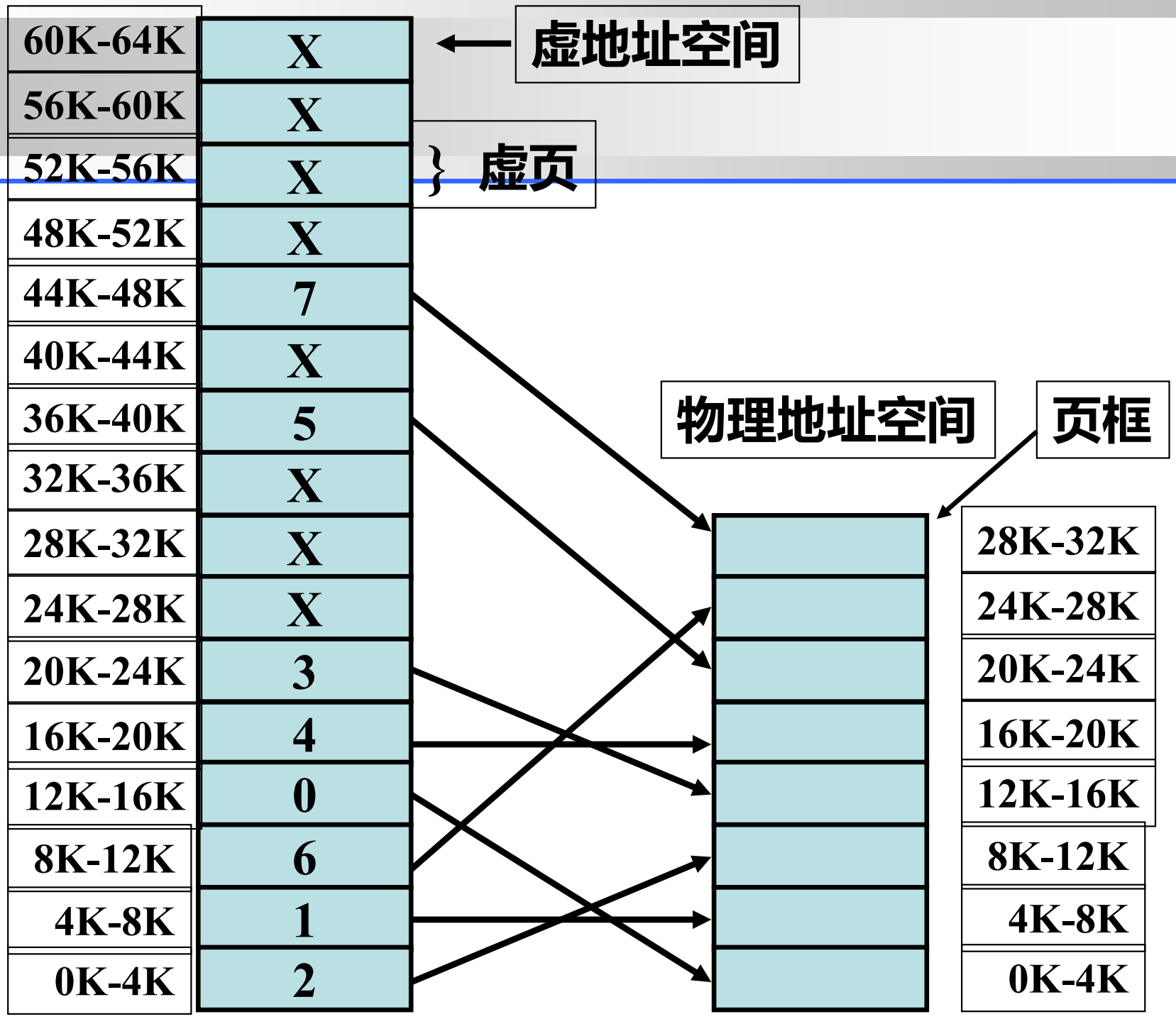
3. 虚拟存储器

连续性	离散性
驻留性	交换性
一次性	多次性

以CPU时间和外存空间换取昂贵内存空间，这是操作系统中的资源转换技术

3. 虚拟存储器

- 对用户：指令地址部分所限定的比实存大得多的地址实间。
- 对系统：借助于各种表格机构，体现虚拟实间。


60K-64K
56K-60K
52K-56K
48K-52K
44K-48K
40K-44K
36K-40K
32K-36K
28K-32K
24K-28K
20K-24K
16K-20K
12K-16K
8K-12K
4K-8K
0K-4K
X
X
X
X
7
X
5
X
X
X
3
4
0
6
1
2
虚地址空间
虚页
物理地址空间
页框
28K-32K
24K-28K
20K-24K
16K-20K
12K-16K
8K-12K
4K-8K
0K-4K

4. 虚拟存储器的容量

- 一个虚拟存储器的最大容量是由计算机的地址结构确定的。如：若**CPU**的有效地址长度为**32**位，则程序可以寻址范围是**0～(2^32)-1**，即虚存容量为 **4GB**。
- 虚拟存储器的容量与主存的实际大小没有直接的关系，而是由主存与辅存的容量之和所确定。

5. 虚拟存储技术

虚存：把内存与外存有机的结合起来使用，从而得到一个容量很大的“内存”，这就是虚存。

实现思想：当进程运行时，先将一部分程序装入内存，另一部分暂时留在外存，当要执行的指令不在内存时，由系统自动完成将它们从外存调入内存工作。

目的：提高内存利用率。

3.10 请求式分页存储管理

请求式分页也称虚拟页式存储管理

与纯分页存储管理不同，请求式分页管理系统在进程开始运行之前，不是装入全部页面，而是装入一个或零个页面，之后根据进程运行的需要，动态装入其它页面；当内存空间已满，而又需要装入新的页面时，则根据某种算法淘汰某个页面，以便装入新的页面

1. 需要解决的问题

系统需要解决下面三个问题：

- 系统如何获知进程当前所需页面不在主存。
- 当发现缺页时，如何把所缺页面调入主存。
- 当主存中没有空闲的页框时，为了要接受一个新页，需要把老的一页淘汰出去，根据什么策略选择欲淘汰的页面。

2. 页描述子的扩充

页号	中断位	内存块号	外存地址	访问位	修改位

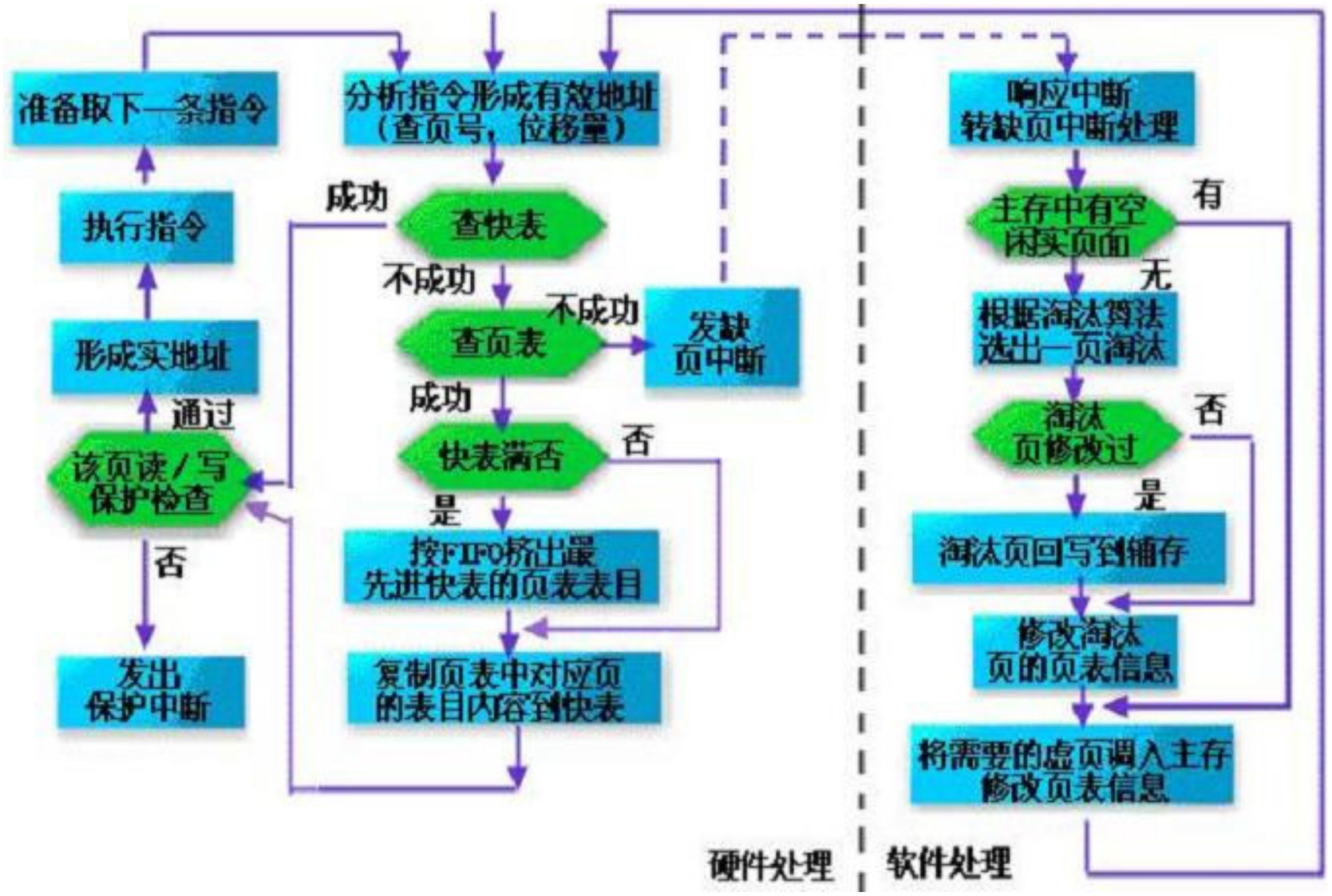
页号、驻留位、内存块号、外存地址、访问位、修改位、（存取控制、辅存地址）

- 中断位：表示该页是在内存还是在外存
- 访问位：表示该页最近被访问过，根据访问位来决定淘汰哪页
- 修改位：查看此页是否在内存中被修改过

3. 地址变换与缺页中断

查页表时，当存在位指示该页不在主存时，则引起一个缺页中断发生，相应的中断处理程序把控制转向缺页中断子程序。执行此子程序，即把所缺页面装入主存。然后处理机重新执行缺页时打断的指令。这时，就将顺利形成物理地址。如下图所示。

请求分页存储管理地址变换流程

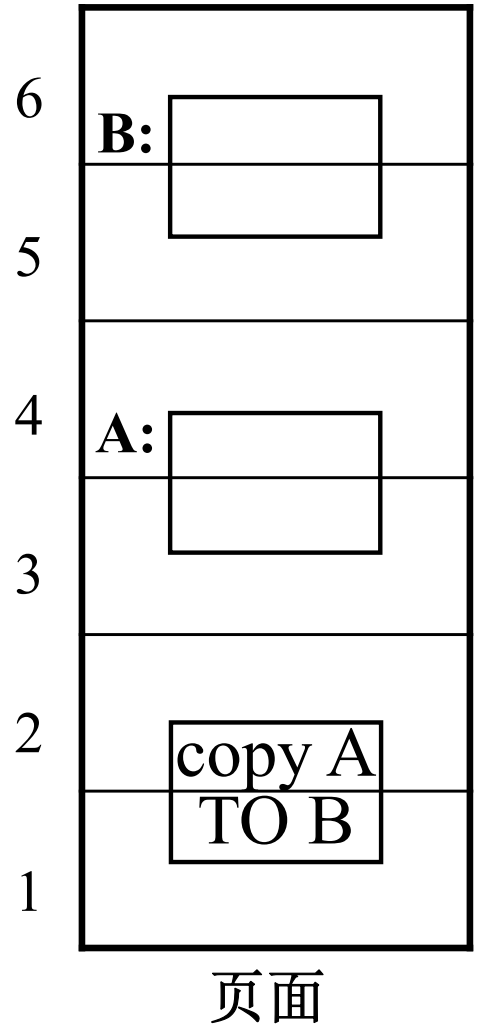


缺页中断（Page Fault）

- 在地址映射过程中，在页表中发现所要访问的页不在内存，则产生缺页中断。操作系统接到此中断信号后，就调出缺页中断处理程序，根据页表中给出的外存地址，将该页调入内存，使作业继续运行下去
- 如果内存中有空闲块，则分配一页，将新调入页装入内存，并修改页表中相应页表项目的驻留位及相应的内存块号
- 若此时内存中没有空闲块，则要淘汰某页，若该页在内存期间被修改过，则要将其写回外存

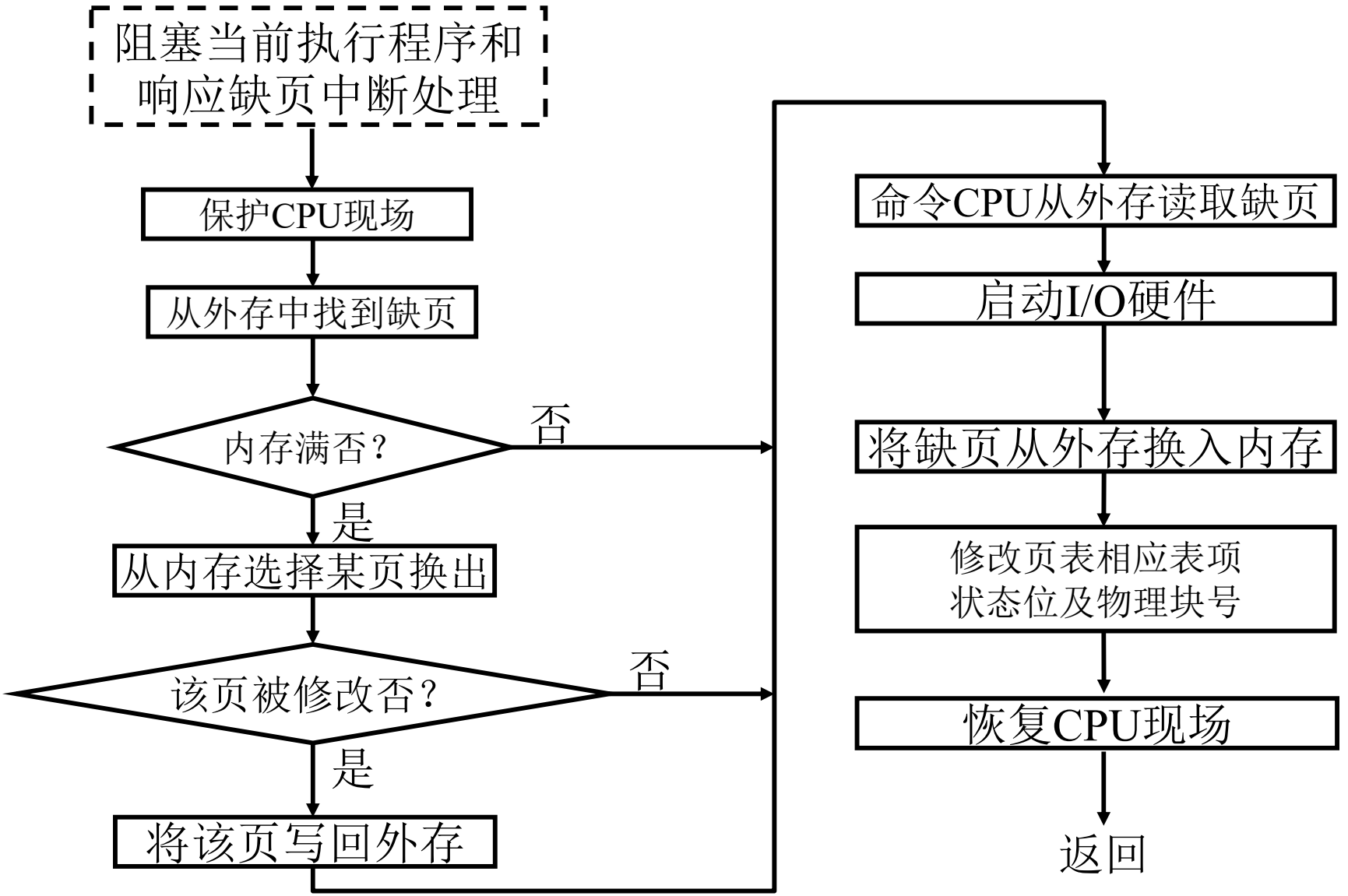
缺页中断

- 缺页中断机构
  - 缺页中断之中断特征
    - 保护**CPU**现场
    - 分析中断原因
    - 转入缺页中断处理程序
    - 恢复**CPU**现场
  - 缺页中断之特殊性
    - 在指令执行期间产生和处理中断信号
    - 一条指令执行期间，可能产生多次缺页中断



缺页中断

- 缺页中断处理算法流程



4. 页面置换算法

当要放一页面到全满的主存块时，系统需淘汰一页。用来选取淘汰哪一页的规则，叫置换算法。

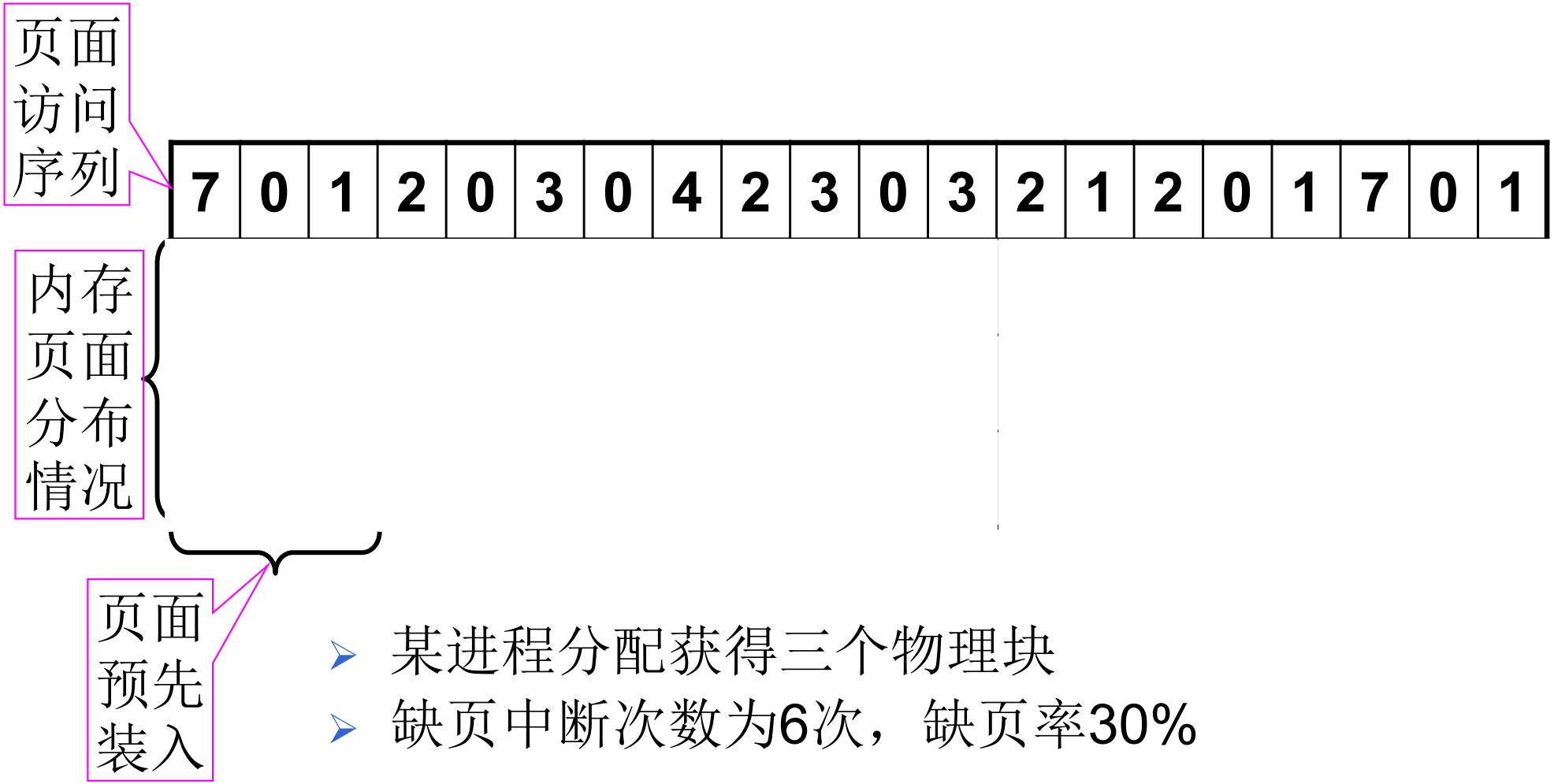
- 最佳置换算法
- 先进先出置换算法
- 最近最久未用置换算法
- 近似的LRU算法（NRU算法）

（1）最佳置换算法

最佳置换算法是由Belady于1966年提出的一种理论上的算法。 其所选择的被淘汰页面，将是以后永不使用的， 或许是在最长(未来)时间内不再被访问的页面。采用最佳置换算法，通常可保证获得最低的缺页率。

最佳置换算法

- 最佳置换算法举例说明



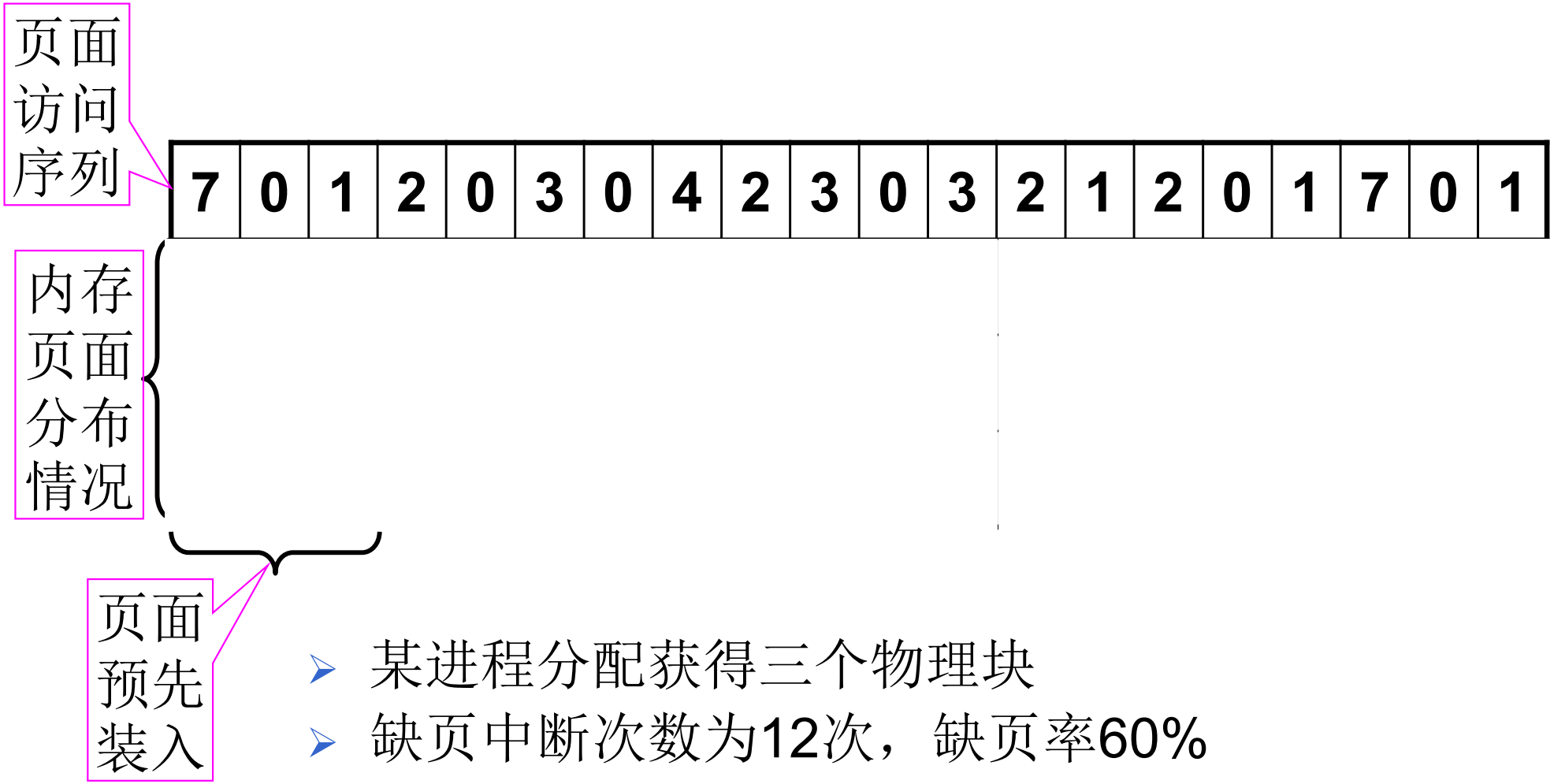
- 某进程分配获得三个物理块
- 缺页中断次数为6次，缺页率30%

（2） 先进先出(FIFO)页面置换算法

- 先进先出置换算法
  - 基本思想
    - 选择最先进入内存即在内存驻留时间最久的页面换出到外存
    - 进程已调入内存的页面按进入先后次序链接成一个队列，并设置替换指针以指向最老页面
  - 评价
    - 简单直观，但不符合进程实际运行规律，性能较差，故实际应用极少

先进先出(FIFO)页面置换算法

- 先进先出置换算法举例说明



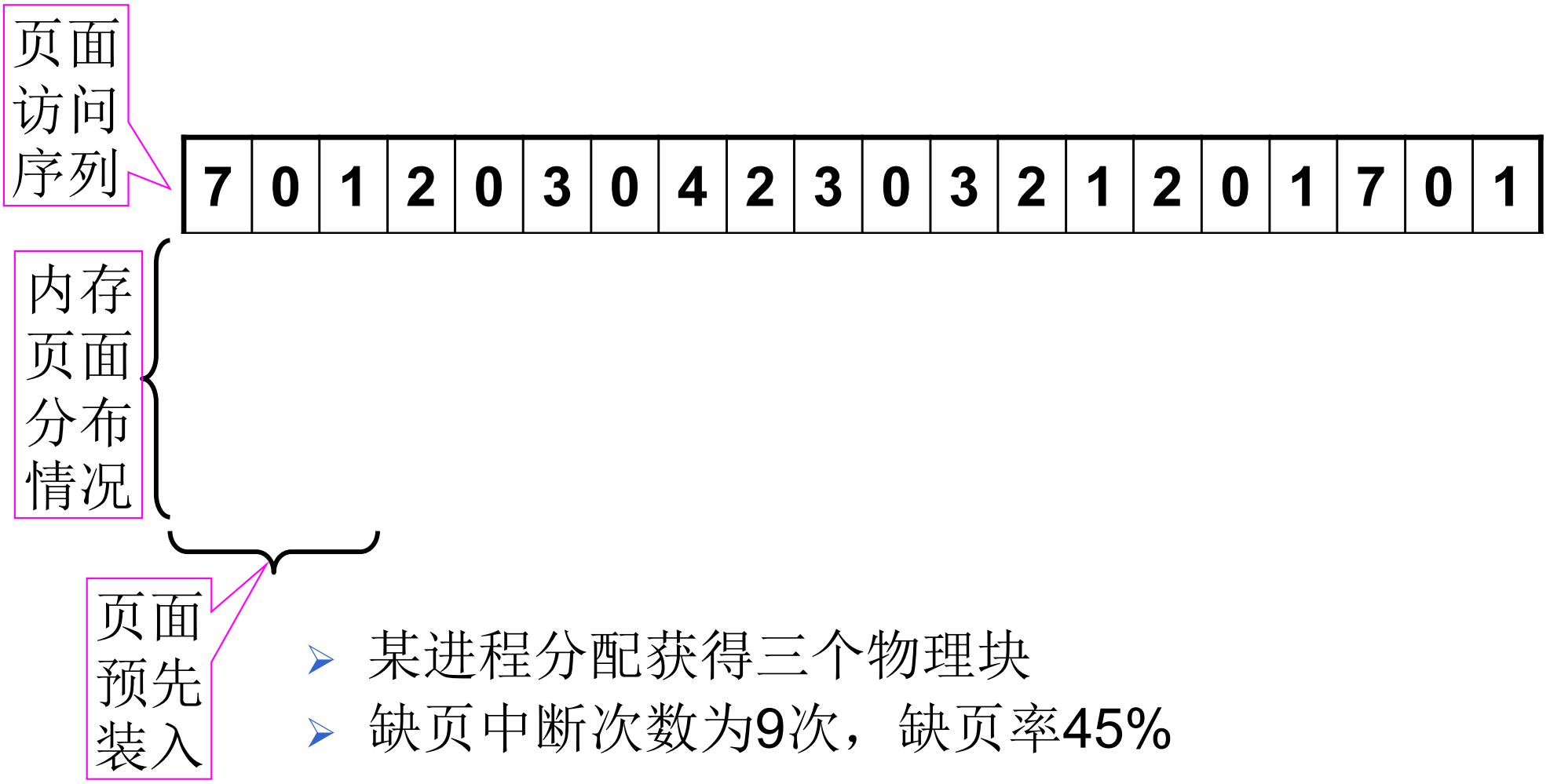
- 某进程分配获得三个物理块
- 缺页中断次数为12次，缺页率60%

(3) 最近最久未使用(LRU)置换算法

- 最近最久未使用置换算法LRU
  - 基本思想
    - 以“最近的过去”作为“最近的将来”的近似，选择最近一段时间最长时间未被访问的页面淘汰出内存
  - 评价
    - 适用于各种类型的程序，性能较好，但需要较多的硬件支持

最近最久未使用(LRU)置换算法

- 最近最久未使用置换算法举例说明



LRU置换算法的硬件支持

1) 寄存器

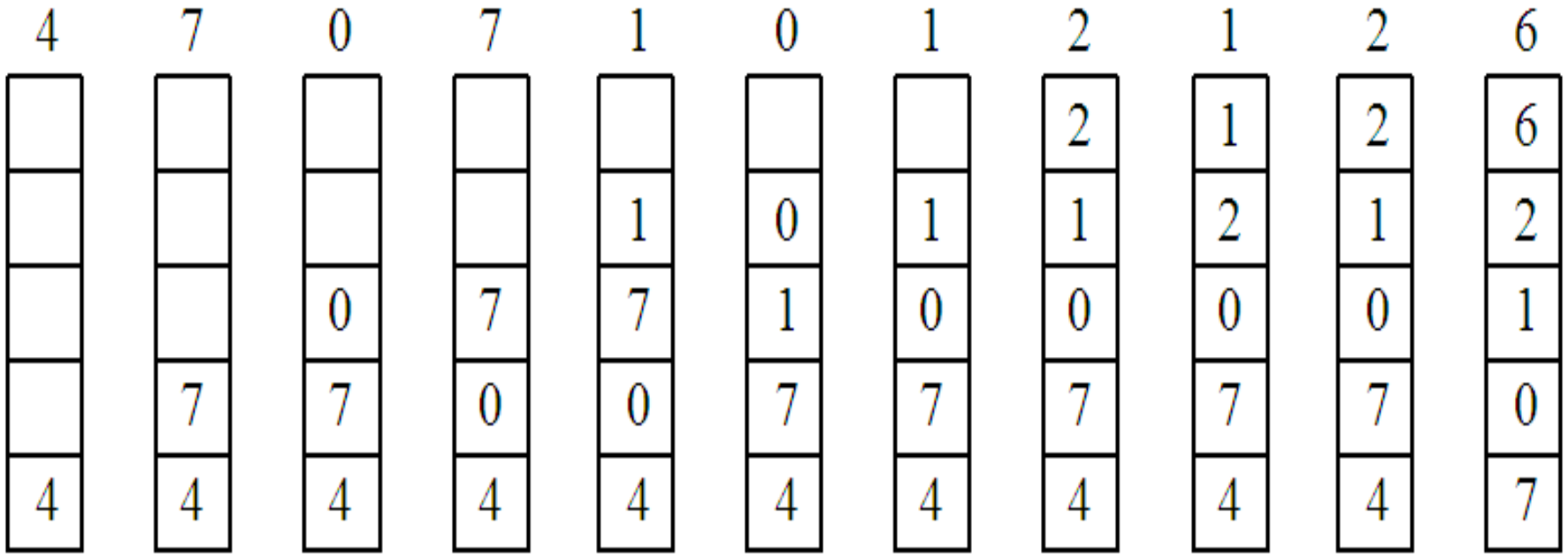
为了记录某进程在内存中各页的使用情况，须为每个在内存中的页面配置一个移位寄存器，可表示为 ：$R=R_{n-1}R_{n-2}R_{n-3}\cdots R_2R_1R_0$

实页 \ R	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	0	1	0	1	0	0	1	0
2	1	0	1	0	1	1	0	0
3	0	0	0	0	0	1	0	0
4	0	1	1	0	1	0	1	1
5	1	1	0	1	0	1	1	0
6	0	0	1	0	1	0	1	1
7	0	0	0	0	0	1	1	1
8	0	1	1	0	1	1	0	1

某进程具有8个页面时的LRU访问情况

LRU置换算法的硬件支持

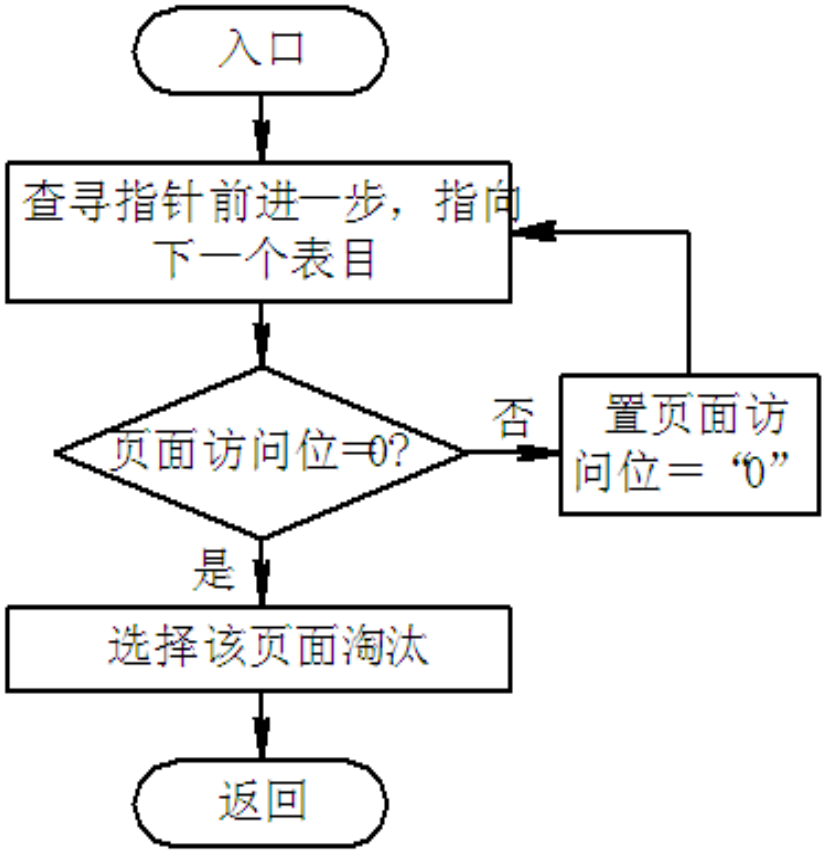
2）栈



用栈保存当前使用页面时栈的变化情况

(4) Clock置换算法

1) 简单的Clock置换算法(近似的LRU算法）



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

简单Clock置换算法的流程和示例

2）改进型Clock置换算法

由访问位A和修改位M可以组合成下面四种类型的页面：

1类(A=0, M=0)：表示该页最近既未被访问， 又未被修改，是最佳淘汰页。

2类(A=0, M=1)： 表示该页最近未被访问， 但已被修改，并不是很好的淘汰页。

3类(A=1, M=0)： 最近已被访问， 但未被修改， 该页有可能再被访问。

4类(A=1, M=1): 最近已被访问且被修改， 该页可能再被访问。

2) 改进型Clock置换算法

其执行过程可分成以下三步：

(1) 从指针所指示的当前位置开始， 扫描循环队列， 寻找A=0且M=0的第一类页面，将所遇到的第一个页面作为所选中的淘汰页。在第一次扫描期间不改变访问位A。

2) 改进型Clock置换算法

（2）如果第一步失败，即查找一周后未遇到第一类页面， 则开始第二轮扫描，寻找A=0且M=1的第二类页面，将所遇到的第一个这类页面作为淘汰页。在第二轮扫描期间，将所有扫描过的页面的访问位都置0。

2) 改进型Clock置换算法

(3) 如果第二步也失败，亦即未找到第二类页面，则将指针返回到开始的位置，并将所有的访问位复0。 然后重复第一步，如果仍失败，必要时再重复第二步，此时就一定能找到被淘汰的页。

(5)最少使用置换算法LFU

- 最少使用置换算法**LFU**
  - 基本思想
    - 为内存各页设置一移位寄存器用于记录对应被访频率，并选择在最近时期使用次数最少的页面淘汰
  - 评价
    - 鉴于仅用移位寄存器有限各位来记录页面使用会导致访问一次与访问多次的等效性，本算法并不能真实全面地反映页面使用情况

(6)页面缓冲算法PBA

- 页面缓冲算法**PBA**
  - 基本思想
    - 设立空闲页面链表和已修改页面链表
    - 采用可变分配和基于先进先出的局部置换策略，并规定被淘汰页先不做物理移动，而是依据是否修改分别挂到空闲页面链表或已修改页面链表的末尾
    - 空闲页面链表同时用于物理块分配
    - 当已修改页面链表达到一定长度如**64**个页面时，一起将所有已修改页面写回磁盘，故可显著减少磁盘**I/O**操作次数

请求分页虚拟存储系统软硬件支持

- 请求分页虚拟存储系统
  - 技术组成：分页 + 请求调页 + 页面置换
  - 硬件支持：请求分页的页表机制 + 缺页中断机构 + 地址变换机构
  - 软件支持：请求调页功能模块 + 页面置换功能模块

影响缺页次数的因素

(1) 分配给进程的物理页面数

(2) 页面本身的大小

(3) 程序的编制方法

(4) 页面淘汰算法

性能问题

颠簸（抖动）：

在虚存中，页面在内存与外存之间频繁调度，以至于调度页面所需时间比进程实际运行的时间还多，此时系统效率急剧下降，甚至导致系统崩溃。这种现象称为颠簸或抖动

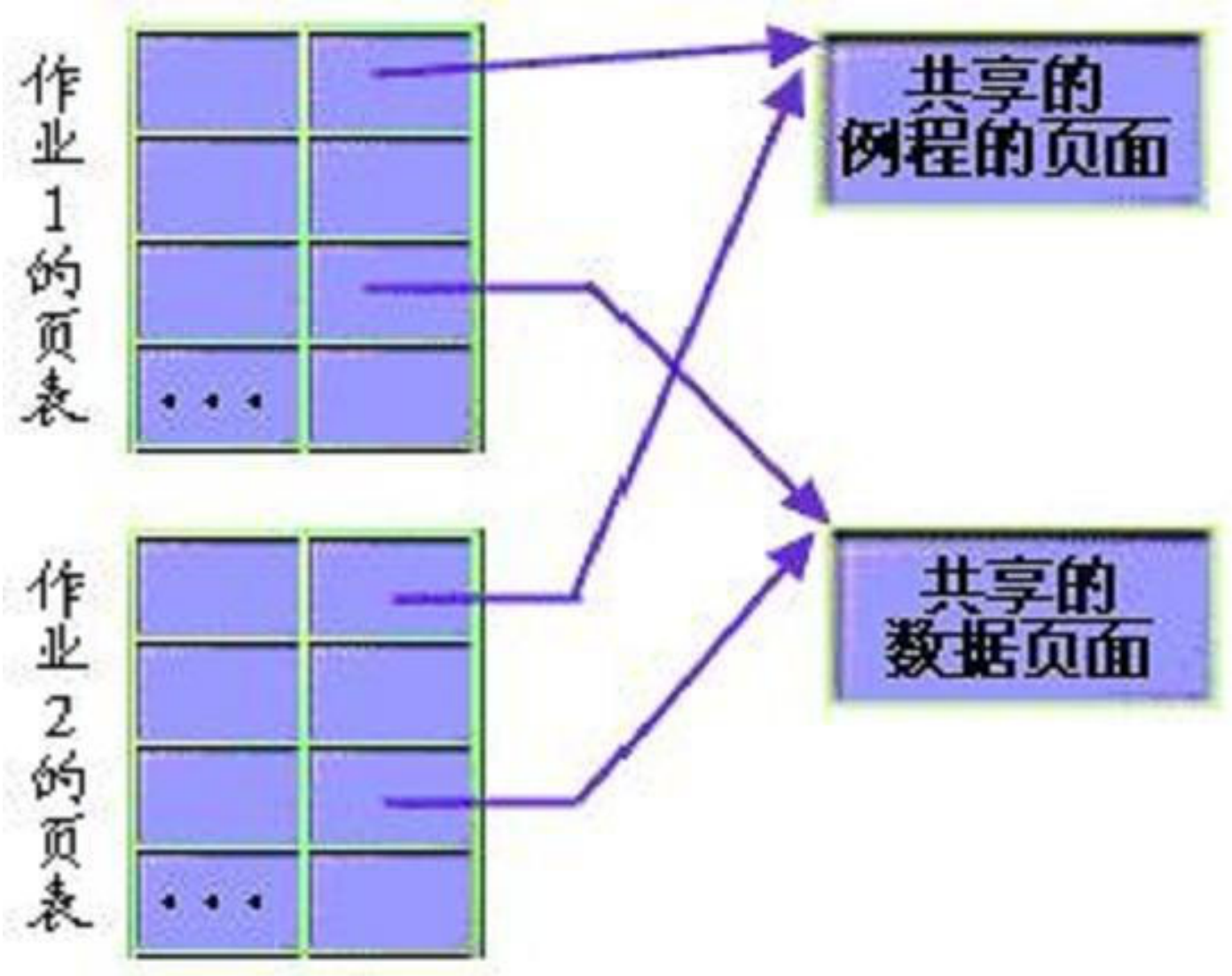
原因：

- 页面淘汰算法不合理
- 分配给进程的物理页面数太少

页面共享

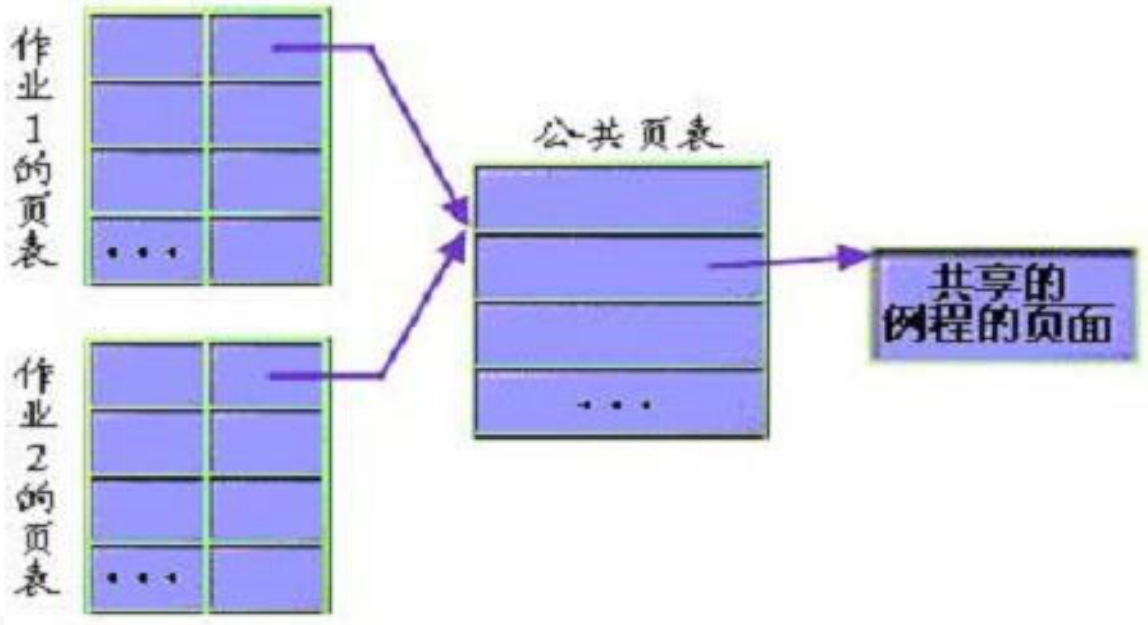
对于数据页面共享，可以安排在诸作业地址空间中的任何一个页面上。至于库例程的共享则不然，它必须把共享的例程，安排到所有共享它的作业地址空间中相同的页面中，即共享例程所在的地址空间必须重叠。如下图所示。

页面共享



实现页面共享的问题

实现页面共享引起的主要问题是，当共享页从主存中淘汰或被重新装入主存时，共享此页的所有作业页表中的相应“页面存在”位都必须更新。具体实现时采用如下技术：



实现页面共享的问题（续）

为那些被共享的页面独立设置一张页表——公共页表，当某个被共享的页面与辅存之间交换时，中需对这个公共页表的表目加以更新，至于有共享页面的作业之页表，其相应表目设有一指针，指向该页在公共页表的表目即可。

总之，实现页面共享，必须使共享的页面具有相同的页号，或者要在地址之间重叠，这将带来麻烦。因此分页不利于共享。这可以在分段存储管理中得到改进。

快表TLB

- 为加快地址映射速度，改善系统性能
- 采用联想映射技术同时查找
- 置于MMU（Memory Management Unit）中
- 大小：
  - Intel 80x86/Pentium：32项
  - SGI MIPS R4000：48项
- 命中率（在TLB中发现一个页面的百分比）
- TLB的两种访问方式：
  - 软件访问
  - 硬件访问

其他有关设计问题

- 页面尺寸
- TLB的大小、位置、访问方式
- 局部与全局分配策略
- 装入策略　与　清除策略
- 负载控制
- 多级页表

局部与全局分配策略

- 分配给一个进程多少页面？
  - 固定数目分配　与　可变数目分配
- 置换范围
  - 全局　与　局部

三种组合：固定　+　局部

可变　+　全局

可变　+　局部

页面装入策略与清除策略

- 装入方式
  - 请求调页
  - 预先调页
- 清除方式
  - 请求
  - 预先

3.11 请求分段存储管理

（虚拟段式存储管理）

3.11请求分段存储管理

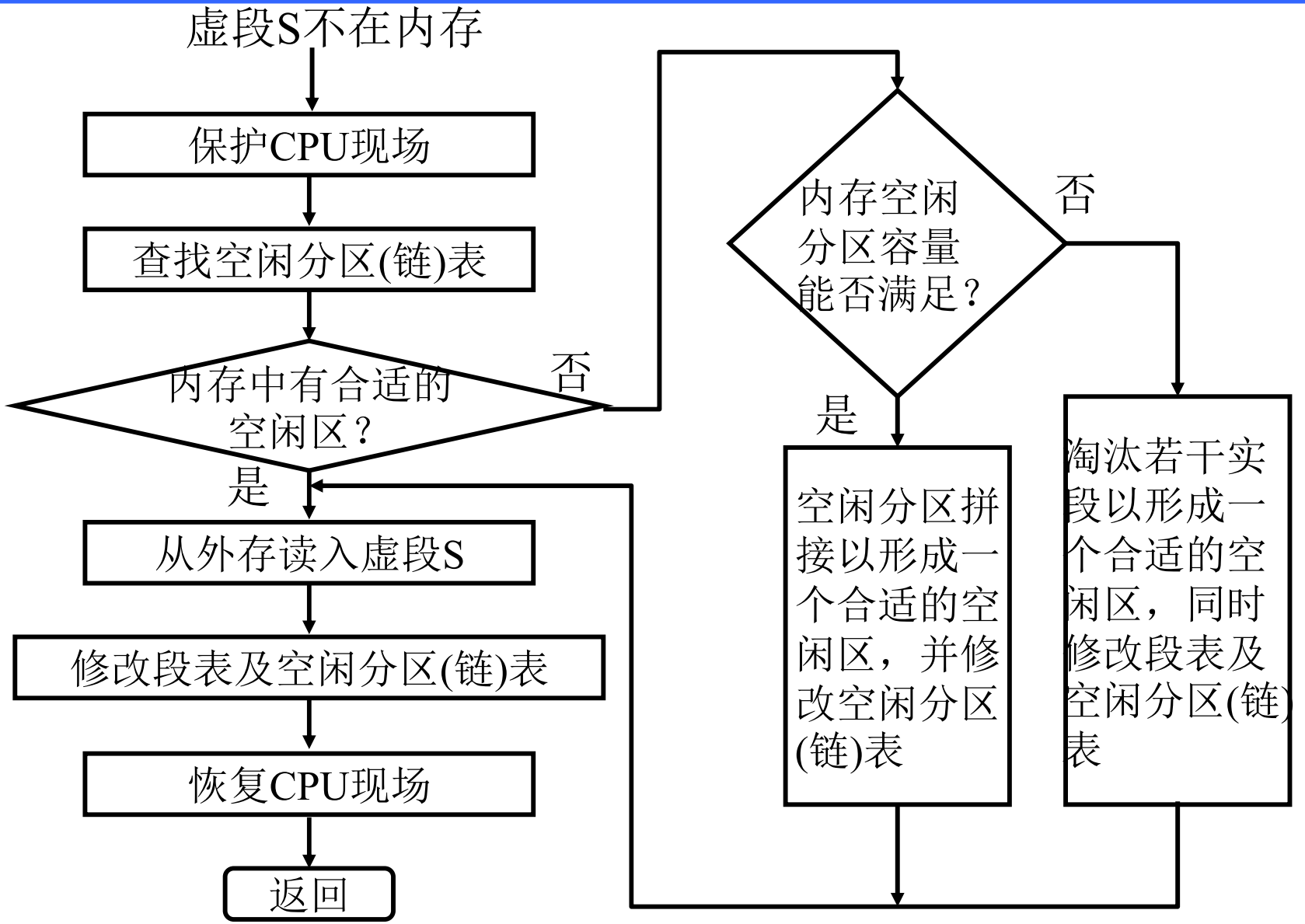
- 段表机制
  - 段表项的扩充

段名	段长	分段基址	存取方式	访问字段	修改位	状态位	增补位	外存地址

起始盘块号

3.11 请求分段存储管理

- 缺段中断机构及处理过程

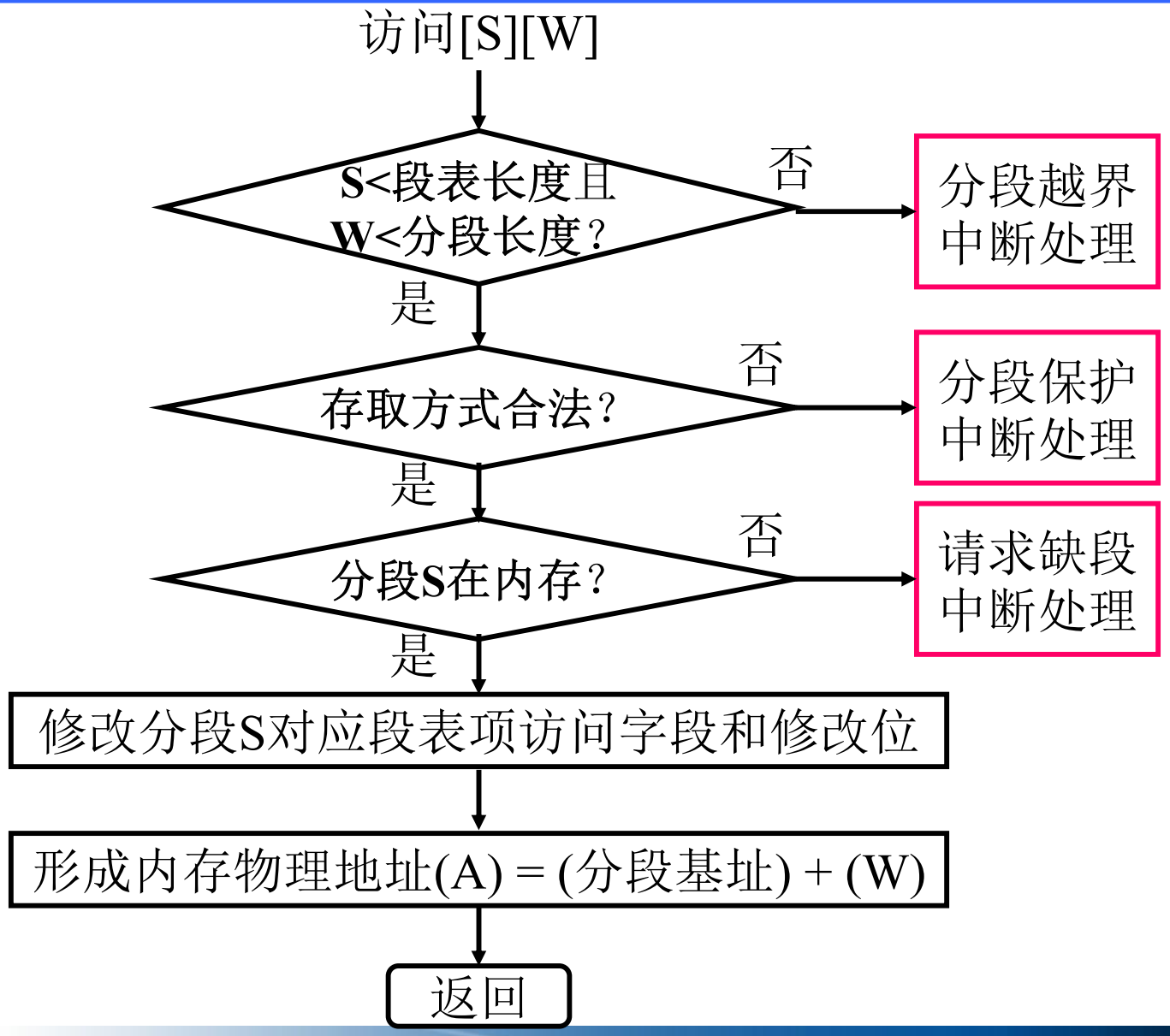


3.11 请求分段存储管理

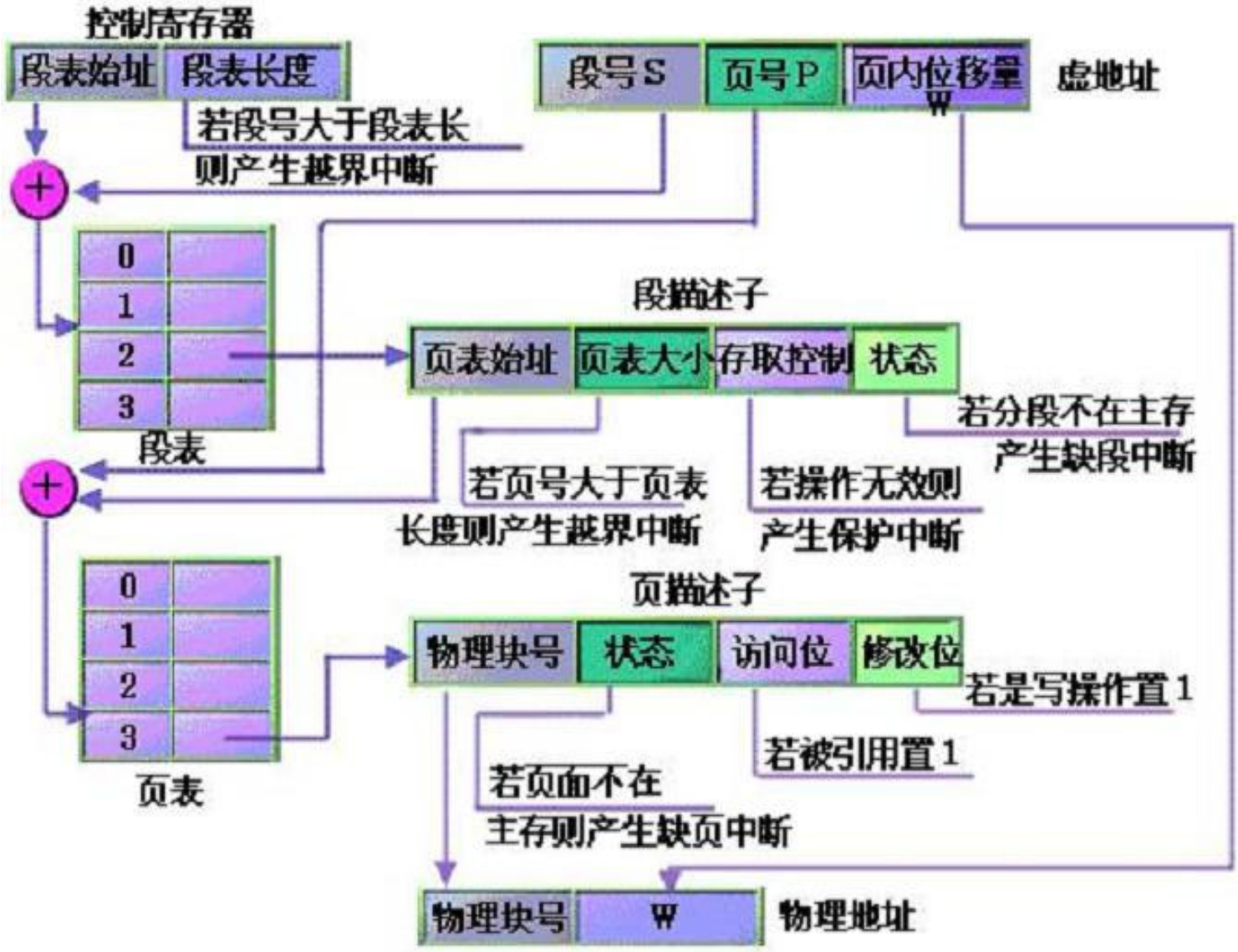
- 地址变换机构
  - 在分段系统的地址变换机构的基础上，增加缺段中断产生和处理并分段置换功能而构成
  - 地址变换过程要领
    - 从段表找到对应分段的段表项获悉该段尚未调入内存时，应产生缺段中断，请求操作系统从外存把该段调入内存
    - 关于快表和段表的检索及表项修改

3.11 请求分段存储管理

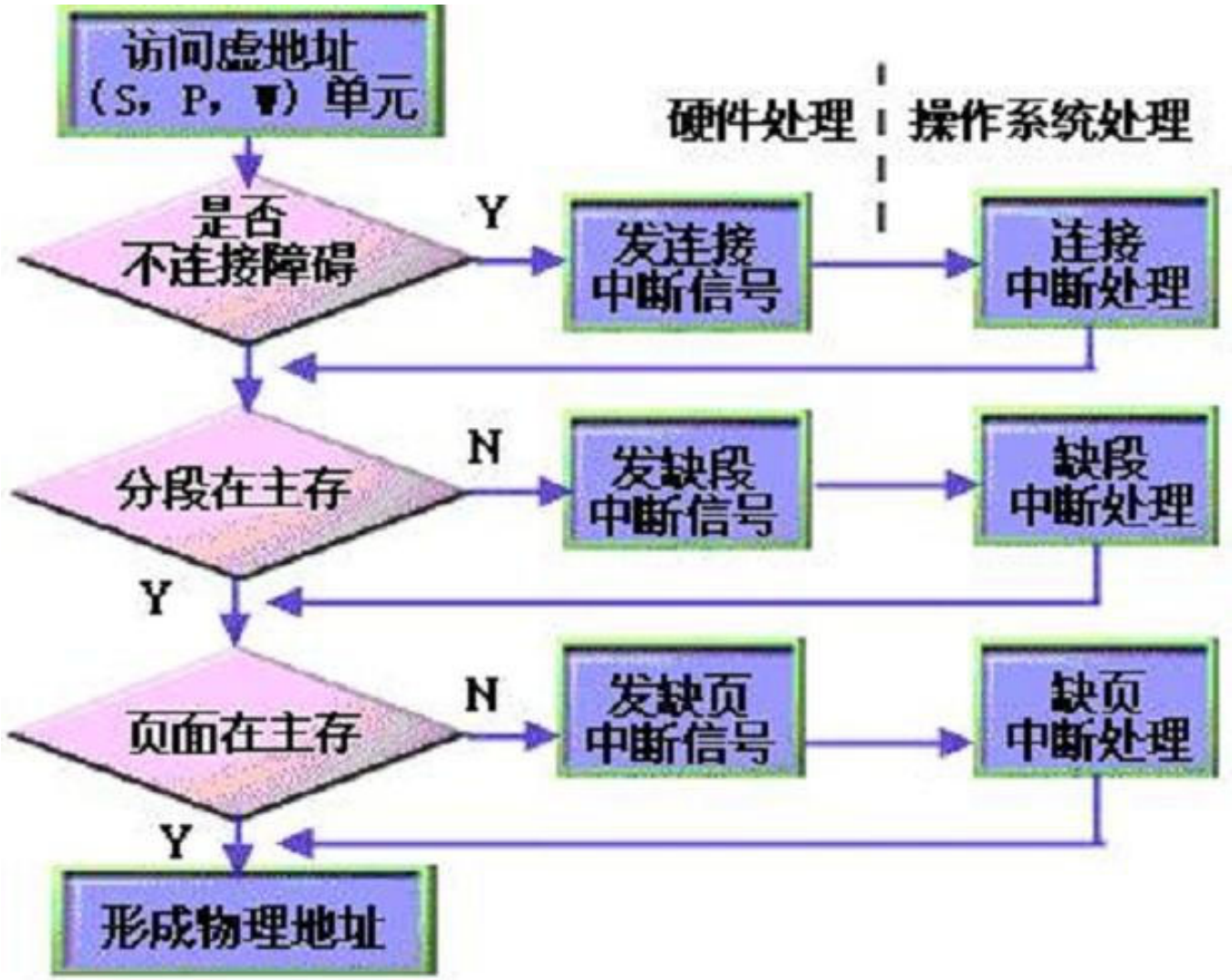
- 请求分段系统地址变换



3.10 请求段页式存储管理



3.10 请求段页式存储管理



The End