第8章 存储器管理

8.1 存储器管理的基本概念

8.2 单一连续分配

8.3 分区管理

8.4 伙伴系统

连续分配

8.5 分页存储管理

8.6 分段存储管理

8.7 段页式存储管理

非连续分配

8.5 分页存储管理

- 固定分区会限制作业道数，且有内部碎片
- 动态分区会产生外部碎片，碎片的拼接又会带来很大开销
- 若能取消作业对存储区的连续性要求，将内存的申请分配到多个不连续的片，则可避免拼接
  - 内存分配时的动态拼接
  - 逻辑上连续，物理上分散
- 分页存储管理就是基于这一思想提出的。

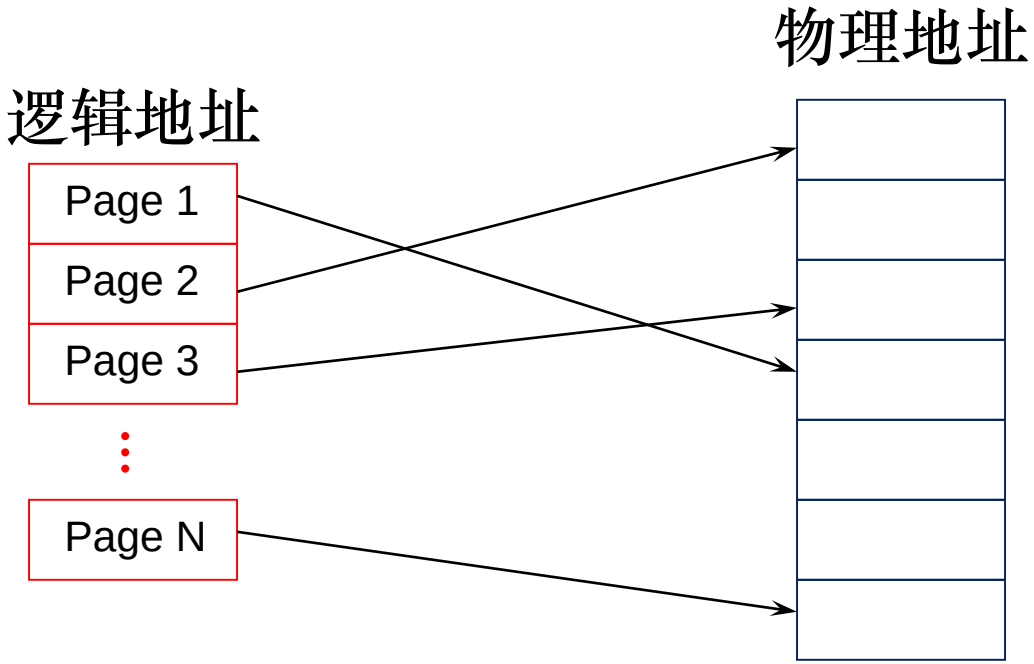
8.5.1分页存储管理的实现思想

- 在分页存储管理中，将逻辑地址空间划分成若干大小相等的页（术语page, 或称页面）
- 将物理主存空间也划分成与页大小相等的块（术语frame, 或称页帧、页框）
- 在分配内存时，总是以页为单位来分配，将逻辑地址中的一页映射到主存的某一空闲块上

分页系统中是否有碎片？

分页存储管理下的映射

- 物理地址
  - 分割成多个页帧
  - 页帧大小一致
  - 物理页帧是零散的
- 逻辑地址
  - 分割成多页
  - 页大小与物理页帧一致
  - 连续的地址空间



- 在大部分体系结构下，由MMU完成逻辑页与物理页帧的映射，对OS和用户进程不可见

分页的逻辑地址结构

- 分页存储管理系统中，逻辑地址由页号和页内位移组成。其结构如下所示：

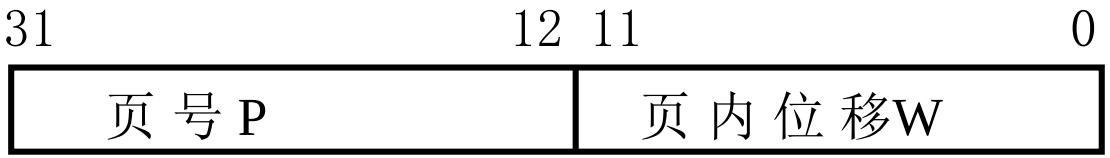
页 号 P	页 内 位 移W

- 若A为逻辑地址，L为页面大小，则：
  - 页号：　　P=int(A/L)
  - 页内位移：W=A % L

例如 A＝10000，L＝4096，则P＝2，W＝1808

分页的逻辑地址结构

- 如果逻辑地址空间为2^m，且页面大小为2^n，那么逻辑地址的高m-n位表示页号，低n位表示页内位移。
- 例如，逻辑地址空间为2^{32}，页面大小为4K，其逻辑地址结构如下所示：



8.5.2 页表结构

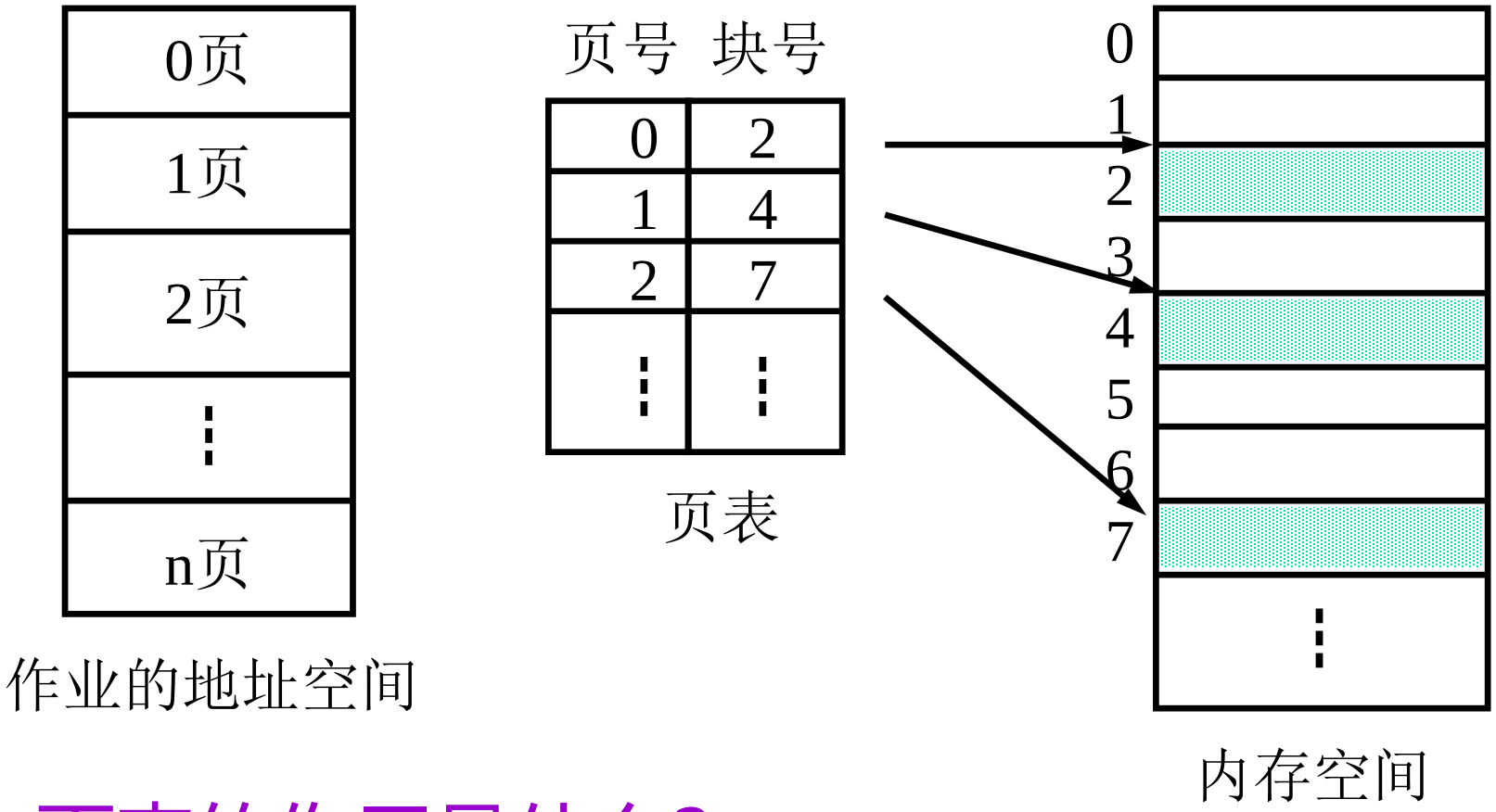
- 页表
  - 记录页面在内存中对应物理块的数据结构
  - 每个页面对应一个页表项
  - 页表长度：页表项的数量（页的数量）

- 页表项
  - 页表项的组成：物理块号、页表项标志位
    - 存在位、修改位、引用位
  - 页表项大小：所占字节数

- 系统为每个进程建立一张页表，随运行状态而变化

- 页表的实现
  - 进程创建时，初始化页表，并将页表基地址和长度保存在PCB
  - 进程运行时，上下文切换的过程中将页表的基地址和长度保存在页表寄存器中

基于页表的映射



- 页表的作用是什么?

8.5.3 地址变换

- 地址变换
  - 将逻辑地址映射为物理地址
- 分页系统地址变换的简单描述：
  - 逻辑地址中的页号转换为内存中的物理块号
  - 由块号计算得到物理块基地址
  - 物理地址=物理块基地址+页内偏移

  - 例如，页面大小为L，逻辑地址A的页号为P，页内位移为W，P对应的物理块号为F，那么A对应的物理地址 = F×L + W。

地址变换的例子

- 使用4B的页对32B的内存进行分页

0 1 2 3	A B C D
4 5 6 7	E F G H
8 9 10 11	I J K L
12 13 14 15	M N O P

逻辑地址

0	5
1	6
2	1
3	2

页表

0 …	
4 5 6 7	I J K L
8 9 10 11	M N O P
12 …	
16 …	
20 21 22 23	A B C D
24 25 26 27	E F G H
28 …	

物理内存

- 分别计算逻辑地址0，3，4，14对应的物理地址

计算过程如下：

- 逻辑地址0：
  - 逻辑地址0的页号为0，页内位移为0。根据页表可以查到，页0对应的物理块号为5。因此逻辑地址0变换为物理地址是20 (= (5*4)+0)。
- 逻辑地址3：
  - 逻辑地址3的页号为0，页内位移为3。页0对应的物理块号为5。因此逻辑地址3变换为物理地址是23 (= (5*4)+3)
- 逻辑地址4：
  - 逻辑地址4的页号为1，页内位移为0。页1对应的物理块号为6。因此逻辑地址4变换为物理地址是 24(= (6*4)+0)
- 逻辑地址13变换为物理地址是9 (= (2*4)+1)

分页地址变换例1

- 设页面大小为1K字节，作业的0、1、2页分别存放在第2、3、8块中。则逻辑地址2500D的页号及页内地址为:
- 2500/1024D=2（页号）；2500 %1024 = 452D（页内地址）；
- 查页表可知第2页对应的物理块号为8;
- 将块号8与页内地址452拼接得到物理地址为:8×1024 + 452 = 8644D。

地址变换例2

- 一分页系统中逻辑地址长度为16位，页面大小为1KB，且第0、1、2、3页依次存放在物理块3、7、11、10中。现有一逻辑地址0A6FH，其二进制表示如下：

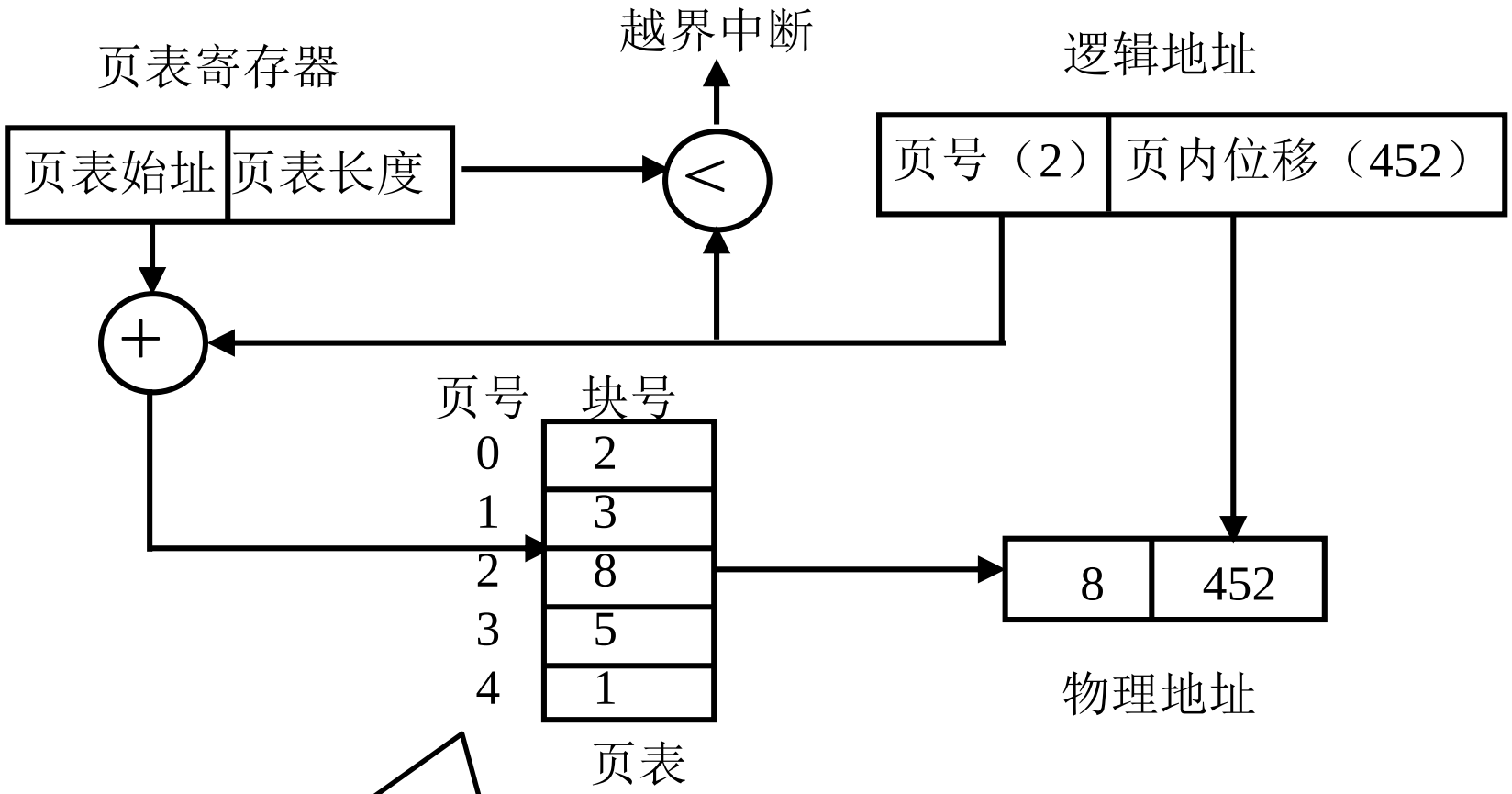
  页号　　页内地址

  000010　1001101111

- 由此可知逻辑地址0A6FH的页号为2，该页存放在第11号物理块中，用二进制表示块号为1011，所以物理地址为：

- 1011　1001101111，即2E6FH。

分页系统的地址变换机构图



注意这里的页号字段？

地址变换过程

- 分页地址变换机构自动地将逻辑地址分为页号和页内位移；
- 将页号与页表寄存器中的页表长度进行比较，如果页号超过了页表长度，则表示本次所访问的地址已超越进程的地址空间，系统产生地址越界中断；
- 若未出现越界，则由页表寄存器中的页表始址和页号计算出相应页表项的位置，从中得到该页的物理块号；
- 将物理块号与逻辑地址中的页内位移拼接在一起，就形成了访问主存的物理地址。

8.5.4 分页管理下的内存分配

- 存储分块表
  - 用来记录物理内存中各物理块的使用情况
- 存储分块表的表示方式
  - 位示图
    - 利用二进制的一位表示一个物理块的状态，1表示已分配，0表示未分配。所有物理块状态位的集合构成位示图。
  - 空闲存储块链
    - 将所有的空闲存储块用链表链接起来，利用空闲物理块中的单元存放指向下一个物理块的指针。

位示图例

- 位示图占用的存储空间：物理块数/8（字节）

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	1	1	0	1	1	1	0	1	1	1	1	1
1	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	1
2	1	1	1	1	1	1	0	1	1	1	1	0	0	0	0	0
3																
4	…															
⋮																

内存分配

- 页面分配
  - 计算进程所需页面数
  - 登记进程号、请求页面数等
  - 若存储分块表中有足够的空闲块，则在系统中取得页表始址，并在页表中登记页号及其对应的物理块号。否则无法分配。

内存回收

- 页面回收
  - 将存储分块表中相应的物理块改为未分配
  - 或将回收块加入到空闲存储块链中
  - 释放页表，修改请求表中的页表始址及状态

页面大小的选择

- 页面的大小应适中
  - 若页面太大，以至和一般进程大小相差无几，则页面分配退化为“分区分配”，碎片也较大。
  - 若页面太小，虽然可减少页内碎片，但会导致页表增长。
  - 页面大小通常为2的幂，一般在512B到64KB之间。

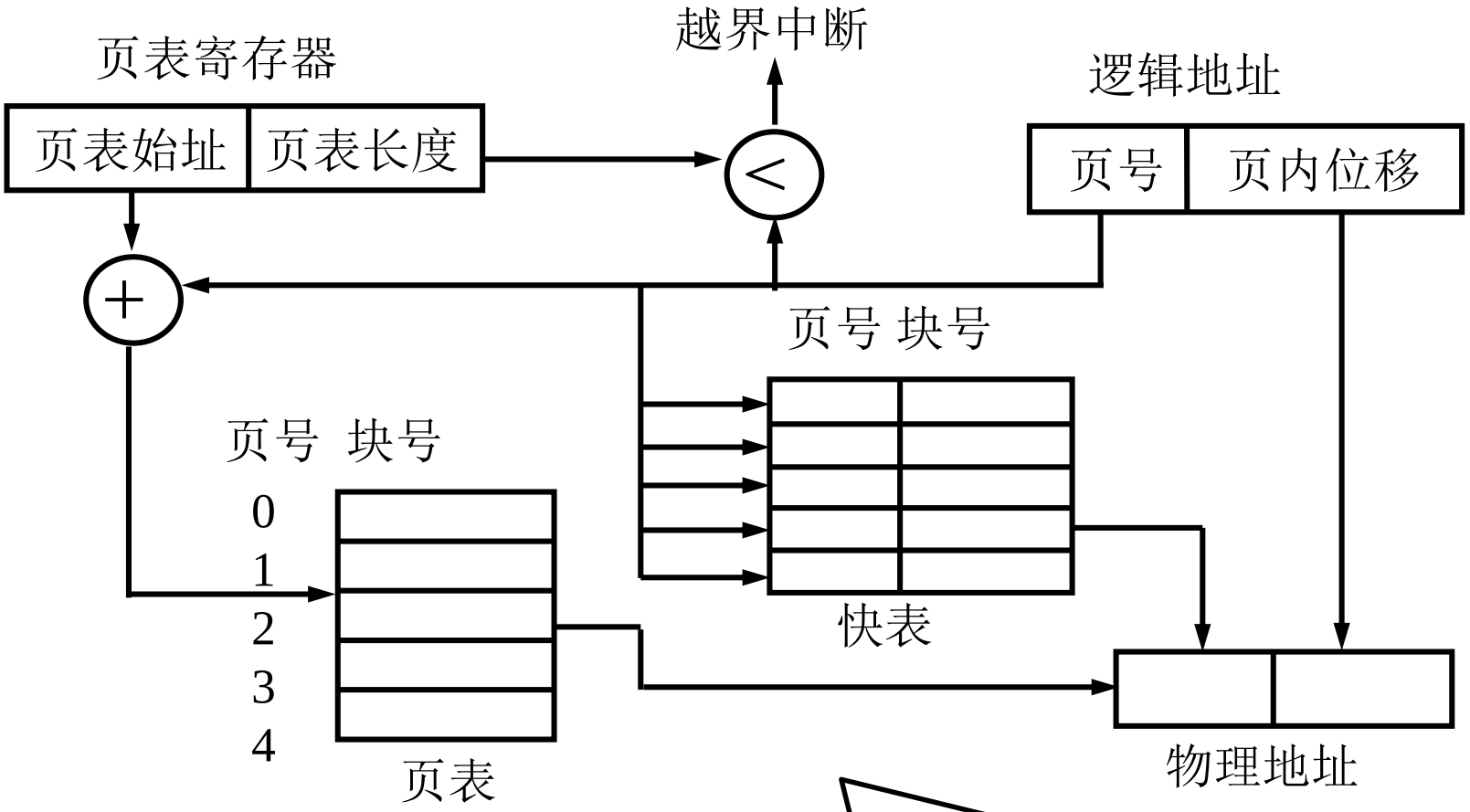
8.5.5 快表

- 因页表放在主存中，故存取数据时CPU至少要访问两次主存，降低了内存访问速度。
- 为了提高地址变换速度，可在地址变换机构中增设一个具有并行查找能力的高速缓冲存储器(又称联想存储器或快表)，用以存放当前访问的那些页表项。
  - TLB（translation look-aside buffer）：转换后备缓冲区，即快表。

引入快表后的地址变换过程

- 地址变换机构自动将页号与快表中的所有页号进行并行比较，若其中有与此匹配的页号，则取出该页对应的块号，与页内地址拼接形成物理地址。
- **若页号不在快表中**，则再到主存页表中取出物理块号，与页内地址拼接形成物理地址。
- 同时还应将这次所查到的页表项存入快表中，若快表已满，则必须按某种原则淘汰出一个表项以腾出位置。

具有快表的地址变换



此处页表与快表有何不同？

快表大小及命中率

- 局部性原理
  - CPU访问存储器时，无论是存取指令还是存取数据，所访问的存储单元都趋于聚集在一个较小的连续区域中。
  - 时间局部性(Temporal Locality)
  - 空间局部性(Spatial Locality)
  - 顺序局部性(Order Locality)：5：1
- 由于成本关系，快表大小一般由64—1024个表项组成。由于局部性原理，快表命中率可达80%--90%。

有效内存访问时间

- 设快表命中率为p，内存访问时间为m，快表访问时间为n，假定忽略快表更新时间
- 则有效内存访问时间=

$$p*（n+m）+（1-p）（2m+n）$$

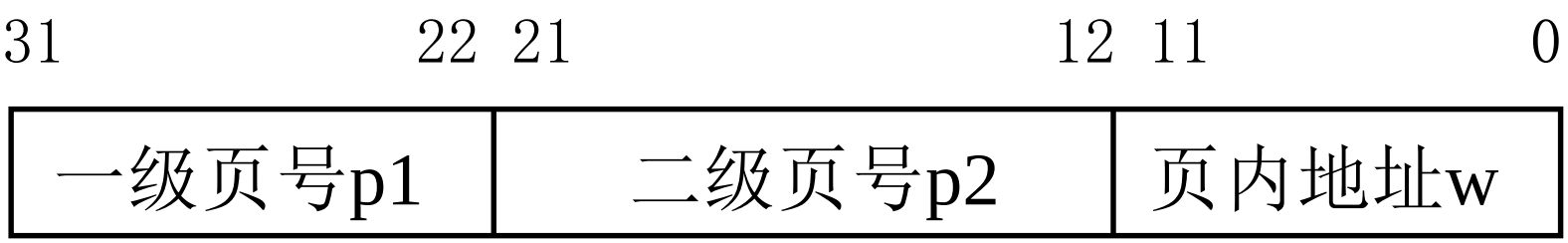
- 若p=0.8，m=100ns，n=20ns
- 则有效内存访问时间=0.8*120+0.2*220=140ns

8.5.6 多级页表及反向页表

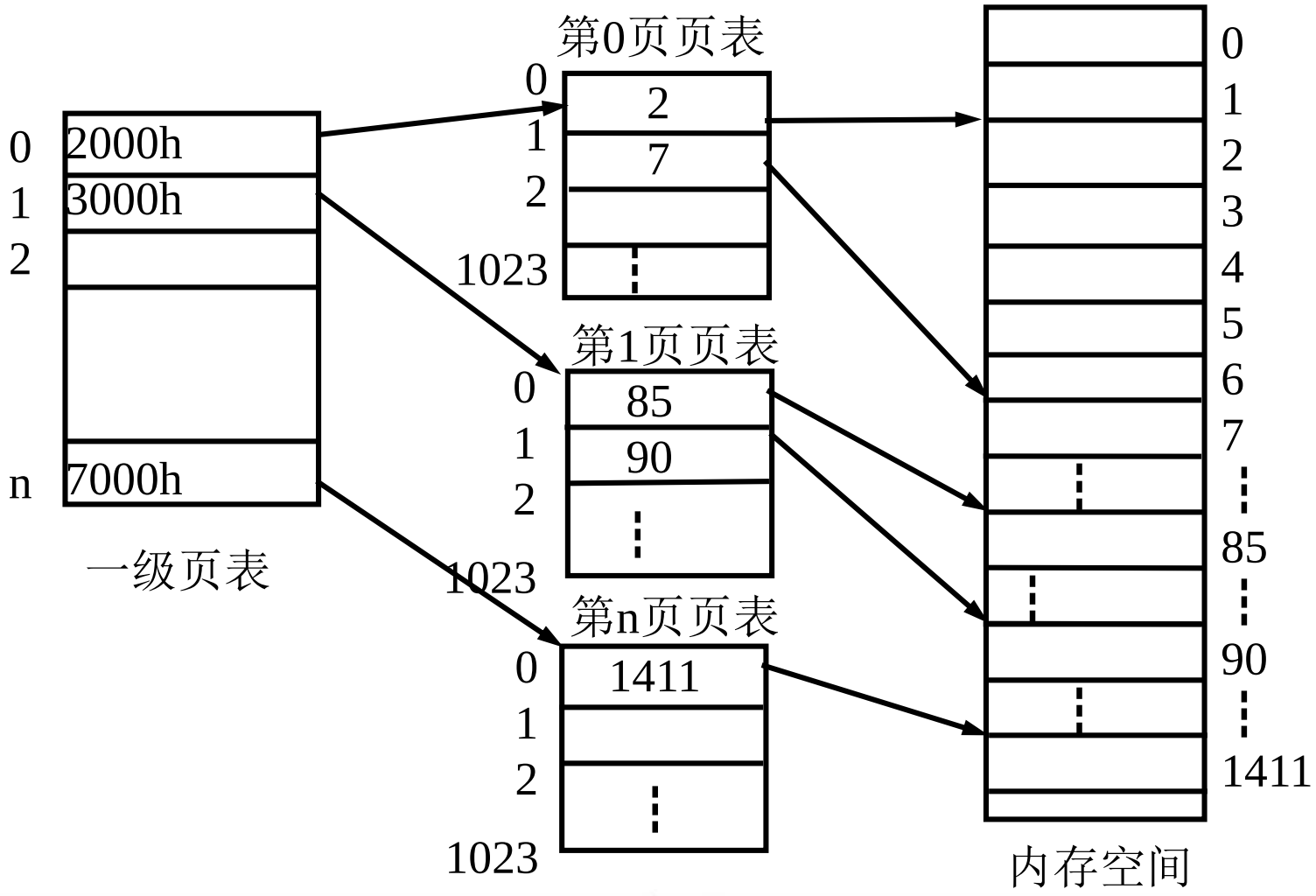
- 现代计算机系统都支持非常大的逻辑地址空间，致使页表很大，连续空间存放既浪费空间又不太灵活
  - 如Intel x86 CPU逻辑地址32位，页面大小4KB，则需页表项为1M个，若每个页表项占4字节，则页表共需要4MB的连续内存空间，**但这仅为一个进程所需**
- 解决方案：
  - 将页表页分级存储，多级索引
  - 反向查找（物理内存大小固定），时间换空间

1）两级页表及多级页表

- 将页表再分页，并为它们进行编号0、1、…，然后为离散存放的页表建立一张页表索引（页目录）
  - 典型情况下，一个页表的最大长度限定为一页
- 例如：32位地址可以划分为

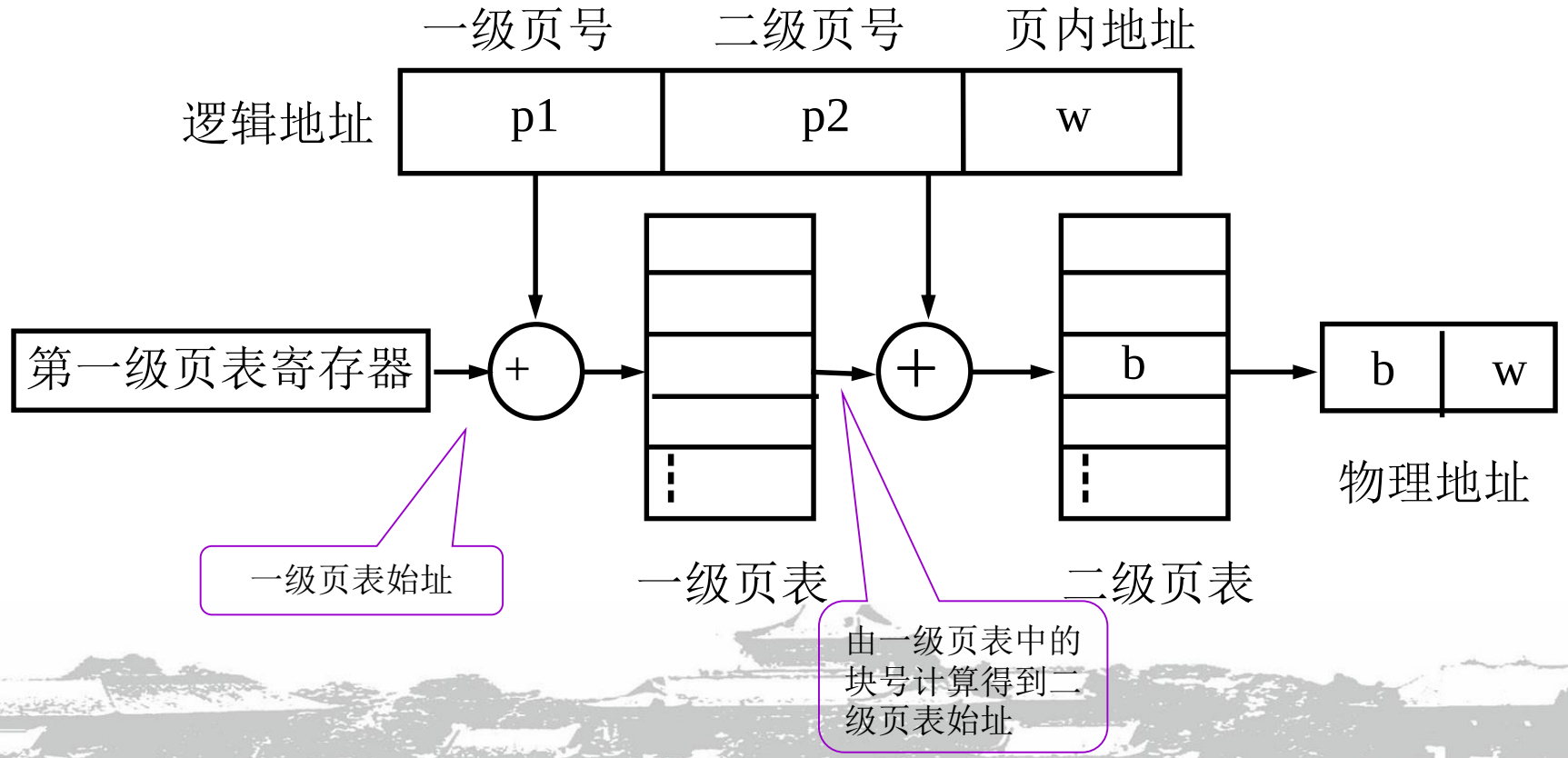


两级页表结构


第0页页表
第1页页表
第n页页表
0
1
2
n
2000h
3000h
7000h
一级页表
2
7
85
90
1411
1023
内存空间
0
1
2
3
4
5
6
7
85
90
1411

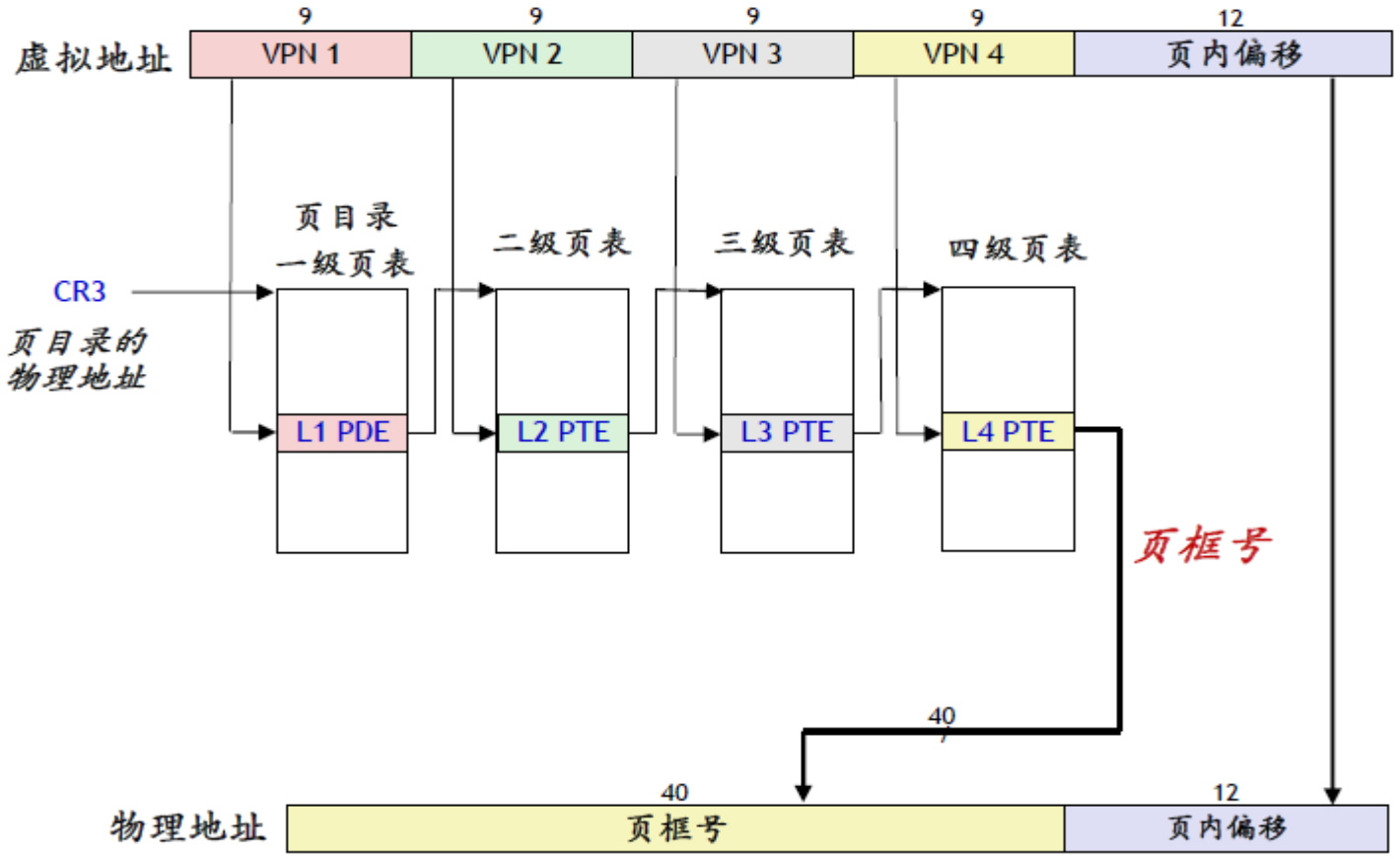
具有两级页表的地址变换过程

- 利用逻辑地址中的一级页号作为索引访问一级页表，找到第二级页表的起始地址
- 再利用第二级页号找到指定页表项，从中取出块号与页内地址拼接形成物理地址。



多级页表

- 对两级页表进行扩充，便可得到三级、四级或更多级的页表。多级页表的实现方式与两级页表类似。



CORE I7 页表结构

虚拟地址空间2^{48}

例

- 为满足2^{64}地址空间的作业运行，采用多级分页存储管理方式，假设页面大小为4KB，页表中每个页表项需占8字节，若每一级页表结构中的页表必须正好存放在一个物理页帧中，则为了满足系统的分页管理至少应采用多少级页表?

例续

- 解：页面大小=4KB=2^{12}B，每个页表项为8字节=2^3B，所以一个页面中可以存放$2^{12}/2^3=2^9$个页表项。
- 设有n层分页，则64位逻辑地址形式为：

第1层页号	第2层页号	…	第n层页号	页内偏移量

- 其中，页面大小为2^{12}字节，所以页内偏移量占12位。
- 由于最高层页表占一页，每页可以存放下2^9个表项，因此分页层数：52/9=6。
- 所以为了满足系统的分页管理至少应采用6级页表。

多级页表的不足

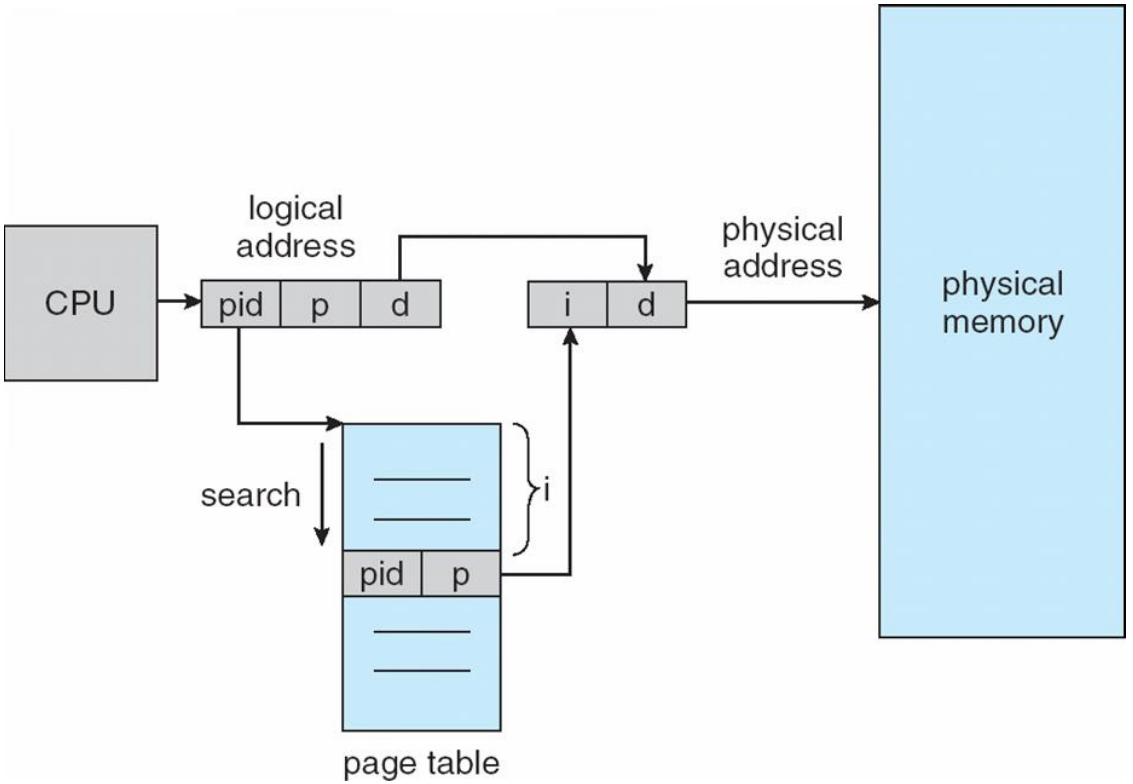
- 多级页表会影响效率。
  - 如二级页表地址变换需三次访问主存，一次访问一级页表、一次访问二级页表、一次访问指令或数据，访问时间加了两倍

2）反向页表

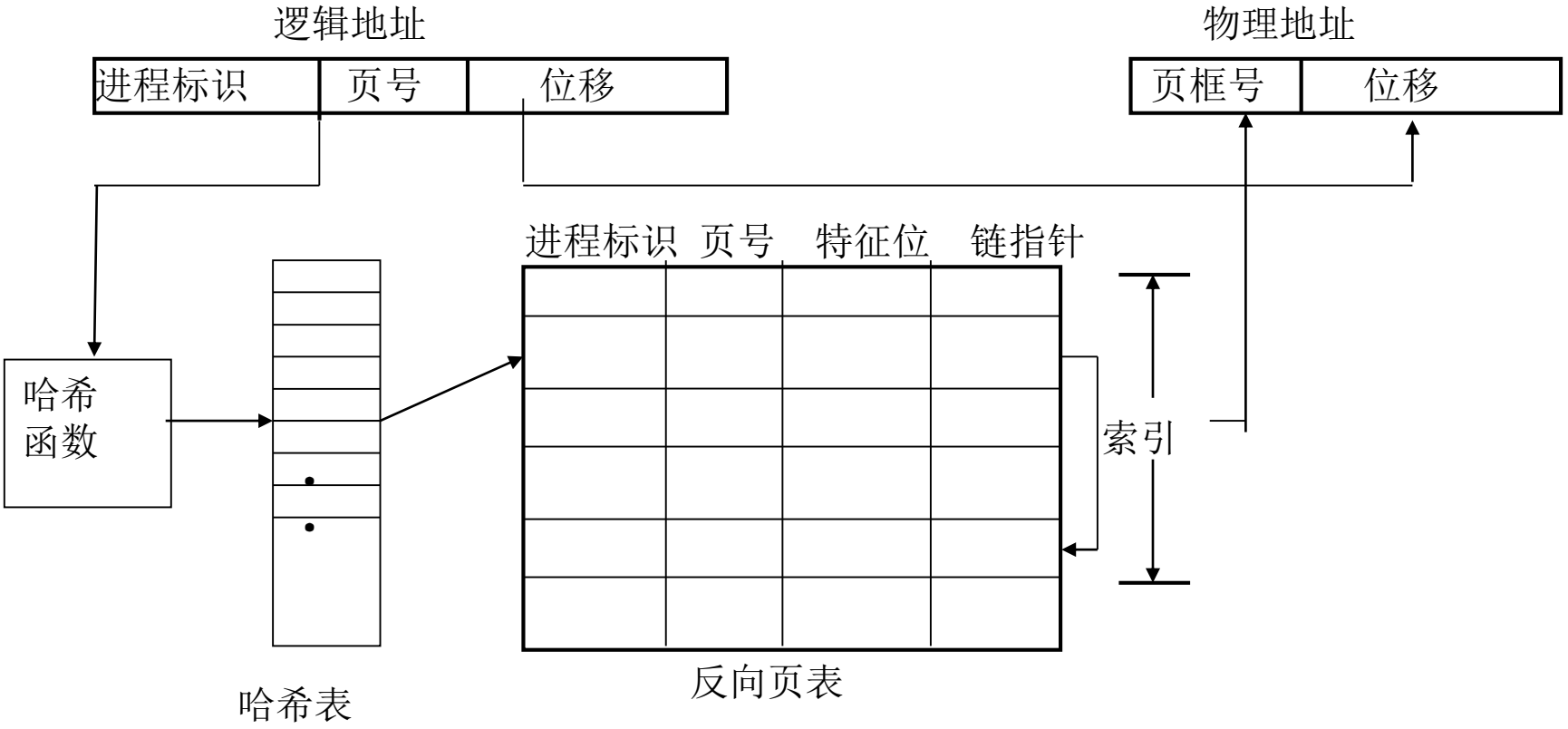
- 页表中的页表项数量取决于逻辑地址，现代操作系统一般允许大逻辑地址空间，如2^{32}，这使得页表太大
- 而采用多级页表会影响效率
- 反向页表 (Inverted Page Table, IPT)
  - 以物理内存的大小来计算页表项
  - 标记每个页帧的使用情况
  - 节省空间，但不可再使用页号进行按序查找

反向页表的地址变换

- 逻辑地址给出进程标识和页号,用它们去比较IPT
- 如果未能找到匹配的页表项,说明该页不在主存,产生请页中断,请求操作系统调入;
- 否则，该表项的序号便是页框号,块号加上位移,便形成物理地址。



反向页表的地址变换



- 引入hash表，可以加速匹配速度

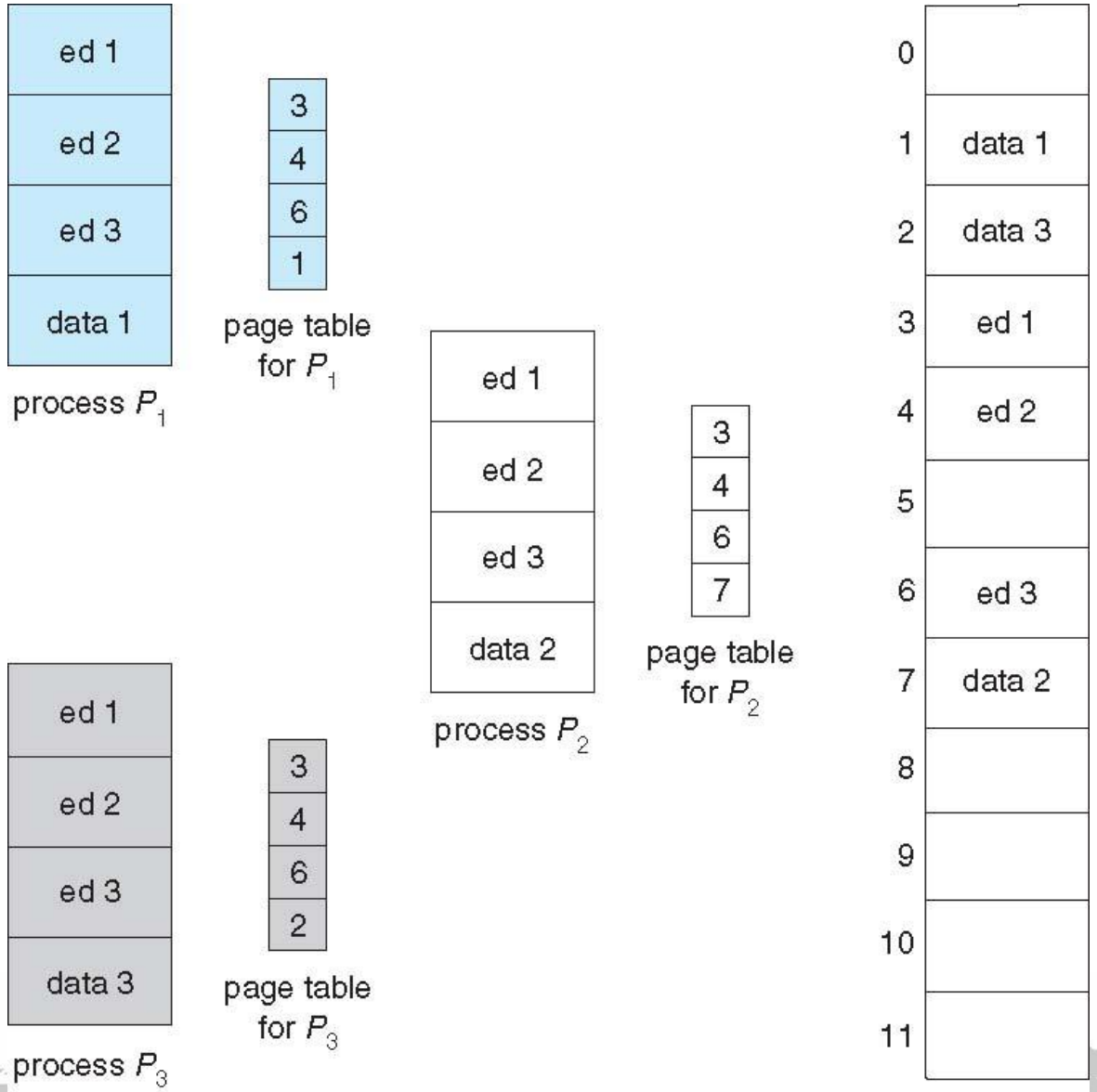
反向页表的不足

- 反向页表查找慢：因为进程号及页号不能作为索引，查找时必须在整个反向页表中进行。
  - 解决办法：将常用页表项存入快表
- 没有调入的页面不在IPT中，仍然需要保存传统页表
  - 解决办法：存放到磁盘中

8.5.7 分页存储管理的保护与共享

- 分页存储管理采用两种方式保护内存：
  - 地址越界保护：页表长度与逻辑地址中的页号比较
  - 存取控制保护：在页表中增加保护位
- 分页存储管理的共享方式
  - 数据共享：不同进程的页表将页号映射到共享的页帧上，允许不同进程对共享数据页，。
  - 代码共享：由于共享代码页面内包含地址，代码内的逻辑地址必须在不同的逻辑空间保持一致，因此不同进程的页表中必须给代码页赋予相同的页号

共享页面例子


ed 1
ed 2
ed 3
data 1
process P_1
3
4
6
1
page table for P_1
ed 1
ed 2
ed 3
data 2
process P_2
3
4
6
7
page table for P_2
ed 1
ed 2
ed 3
data 3
process P_3
3
4
6
2
page table for P_3
0
1 data 1
2 data 3
3 ed 1
4 ed 2
5
6 ed 3
7 data 2
8
9
10
11

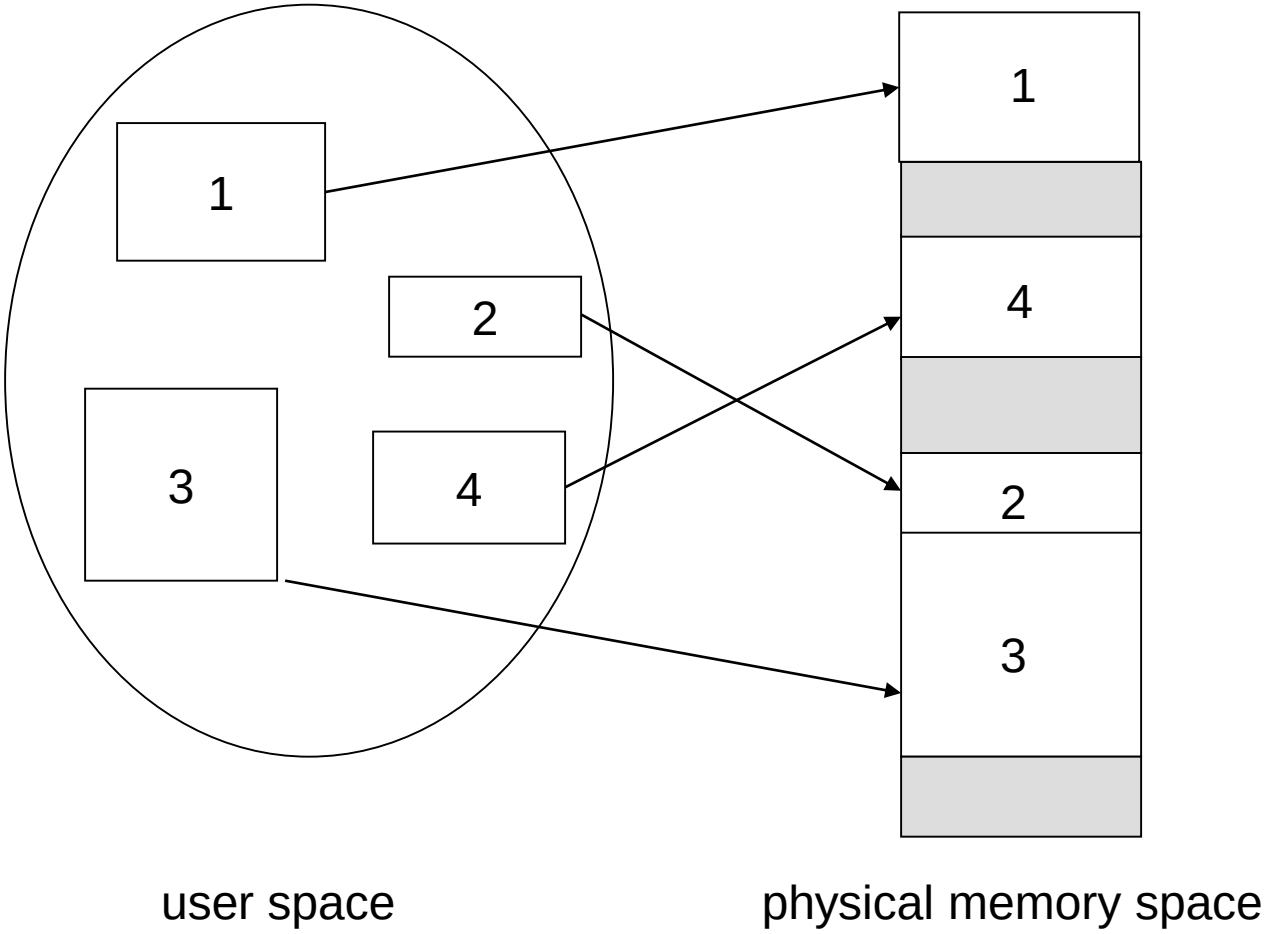
8.6 分段存储管理

- 由于分页按物理单位进行，没有考虑程序段的逻辑完整性，给程序段的共享和保护带来不便，另外动态链接及段的动态增长也要求以逻辑上完整的程序段为单位管理。
- 一个典型的C编译器可能会创建如下段
  - 代码
  - 全局变量
  - 堆
  - 标准的C库函数

5.6.1 分段管理的实现思想

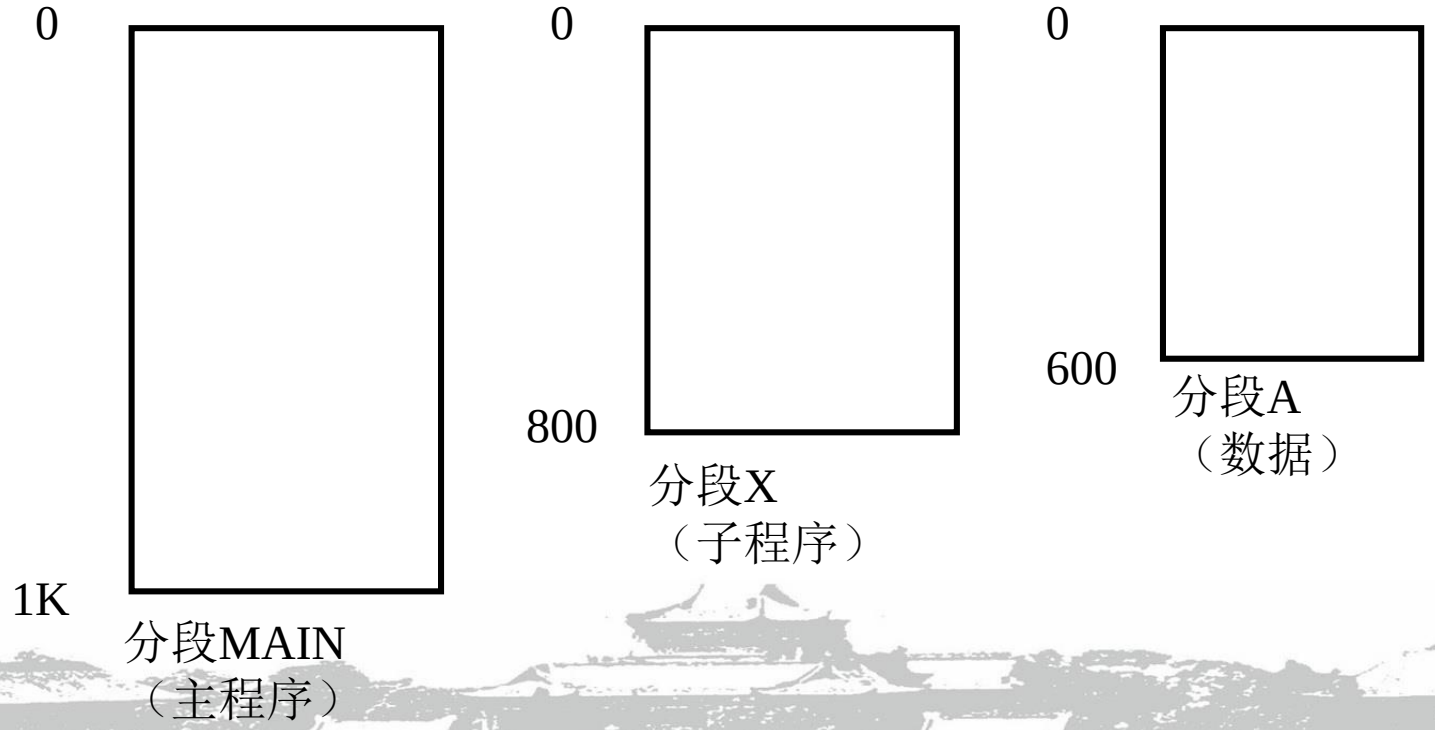
- 在分段存储管理系统中，作业的地址空间由若干个逻辑分段组成，每个分段是一组逻辑意义相对完整的信息集合，每个分段都有自己的名字，每个分段都从0开始编址并采用一段连续的地址空间。
  - 用户视角的内存管理

- 在进行存储分配时，以段为单位分配内存，每段分配一个连续的内存区，但各段之间不要求连续。
  - 可视为动态分区的一种扩展
  - 有碎片吗?

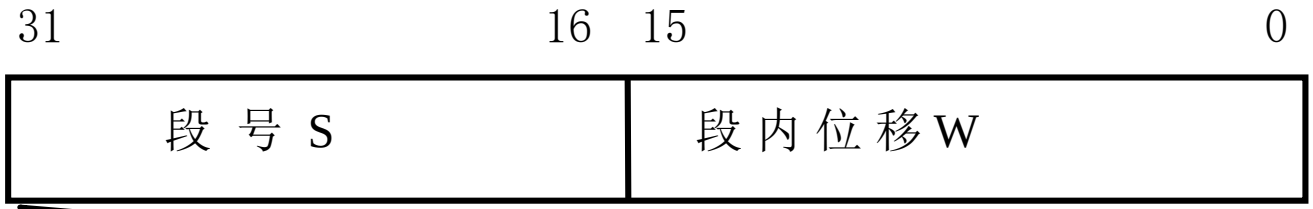

1
2
3
4
1
4
2
3
user space
physical memory space

8.6.1 分段管理的逻辑地址

- 作业的地址空间分为多段，每段都从0开始编址。
- 每个分段都有名称和长度，为了实现的简便，赋予段号，通过段号来引用
- 故地址是二维的，包括段号和段内偏移。



逻辑地址结构



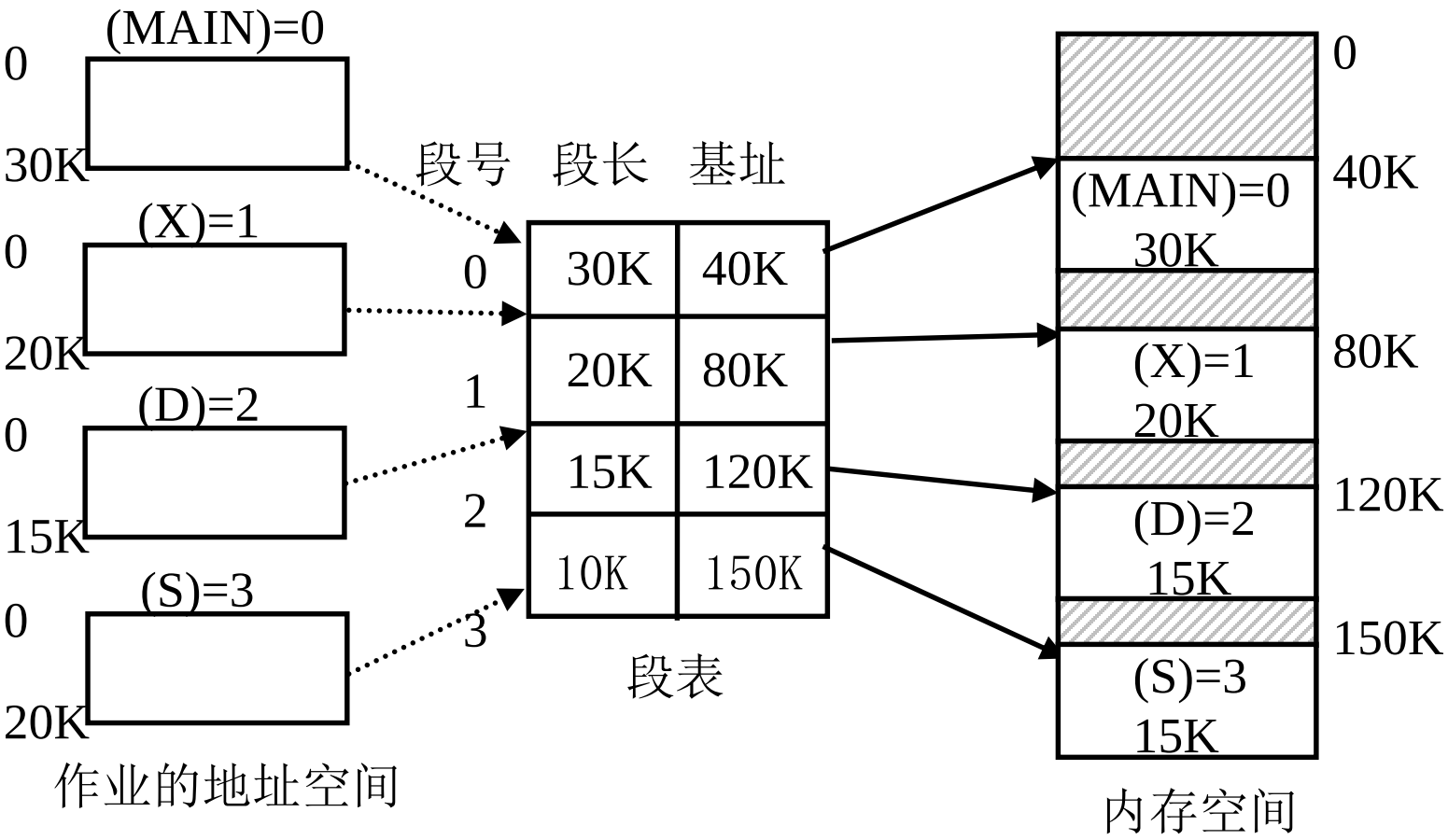
该地址结构最多允许多少分段?每段最大长度为多少?

该地址结构允许作业最多有64K个段，
每段的最大长度为64KB。

8.6.2 段表及地址变换

- 段表：用来记录每段在内存的起始地址及相关信息。
- 每个表项描述一个分段的信息，至少包含：
  - 段号
  - 段长
  - 段在内存的起始地址
  - 其他信息
- 段表在内存中，段表始址和段表长度保存在段表寄存器中。
- 为了提高内存的访问速度，也可以使用快表。

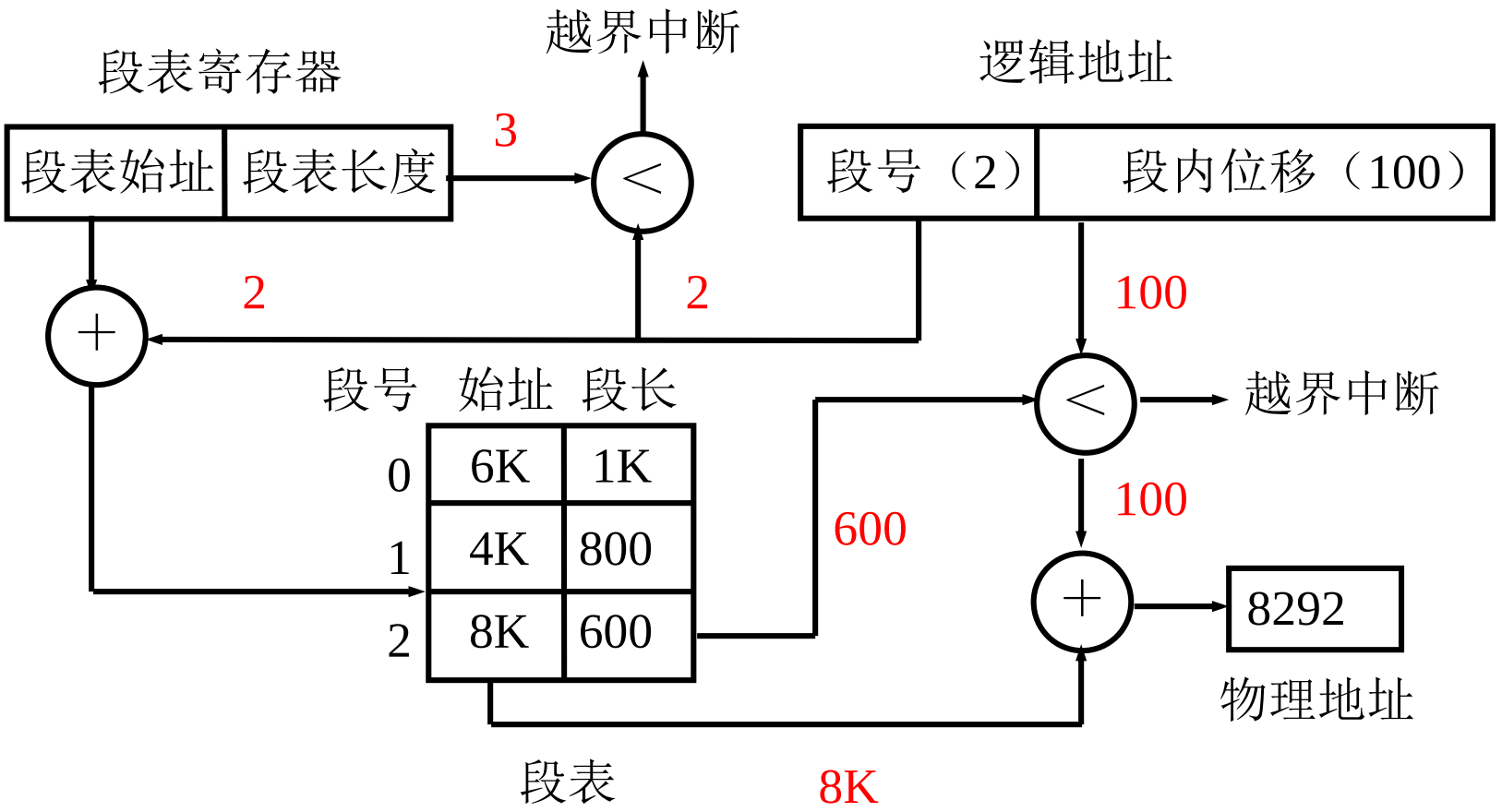
段表的作用


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

地址变换过程

- 进行地扯变换时，系统将逻辑地址中的段号S与段表长度进行比较，若段号超过了段表长度则产生越界中断；
- 否则根据段表始址和段号计算出该段对应段表项的位置，从中读出该段在内存的起始地址，
- 然后再检查段内地址是否超过该段的段长，若超过则同样发出越界中断信号；
- 若未越界，则将该段的起始地址与段内位移相加，从而得到了要访问的物理地址。

地址变换机构图


越界中断
段表寄存器
逻辑地址
段表始址
段表长度
3
段号（2）
段内位移（100）
2
2
100
段号 始址 段长
越界中断
0 6K 1K
1 4K 800
2 8K 600
600
100
8292
物理地址
段表
8K

练习：分段地址变换

- 设作业分为3段，0、1、2段长度分别为1K、800、600，分别存放在内存6K、4K、8K开始的内存区域。
- 逻辑地址（1，100）的段号为1，段内位移为100。如何计算其对应的物理地址?

- 查段表可知第1段在内存的起始地址4K。
- 将起始地址与段内位移相加，4K + 100 = 4196，即物理地址为4196。

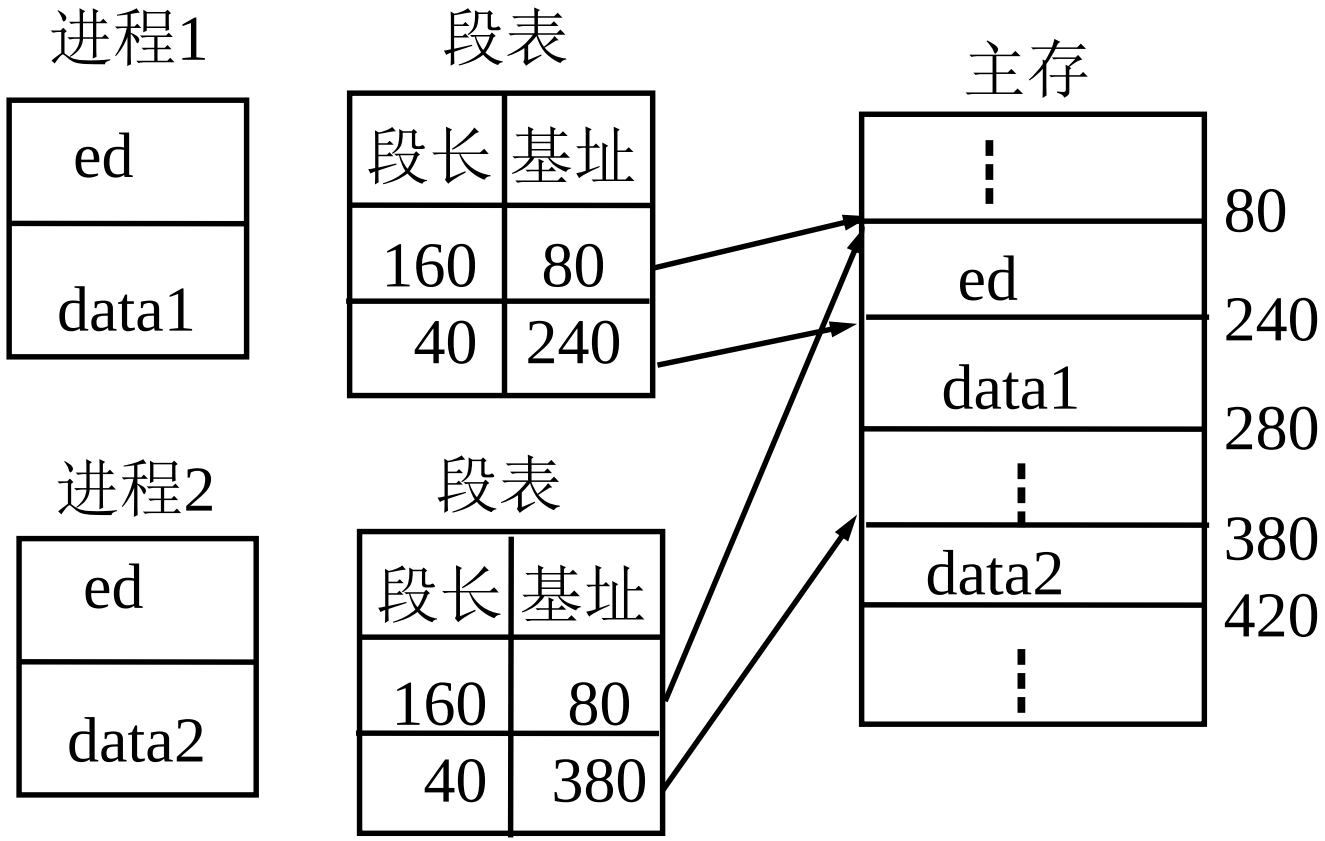
8.6.3 分段保护和共享

- 分段保护方法有：
  - 地址越界保护：段号与段表长度的比较，段内位移与段长的比较
  - 存取控制保护：设置存取权限，访问段时判断访问类型与存取权限是否相符

分段系统中的信息共享

- 分段是信息的逻辑单位，因而实现共享比分页系统方便。
- 在分段存储管理系统中，信息的共享是通过使多个进程的段表项指向同一内存区域实现的。

分段系统中共享信息示意图



分段与分页的比较

- 分段
  - 段是逻辑单位，方便组织程序和数据
  - 段长度不固定且由用户所编写的程序决定，通常由编译系统在对源程序进行编译时根据信息的性质来划分
  - 段基址通常由程序员或编译器显示给出
- 分页
  - 页帧是物理内存管理单位，减少碎片，提高内存使用率
  - 页的大小固定且由系统决定
  - 硬件负责解析逻辑地址为页号和页内地址两部分
- 分段和分页可以有效结合
  - 可以将段按页面大小分割成多页

8.7 段页式存储管理

- 分页系统能有效地提高内存利用率，而分段系统能很好地反映用户要求。

- 如果将这两种存储管理方式结合起来，就形成了段页式存储管理系统。

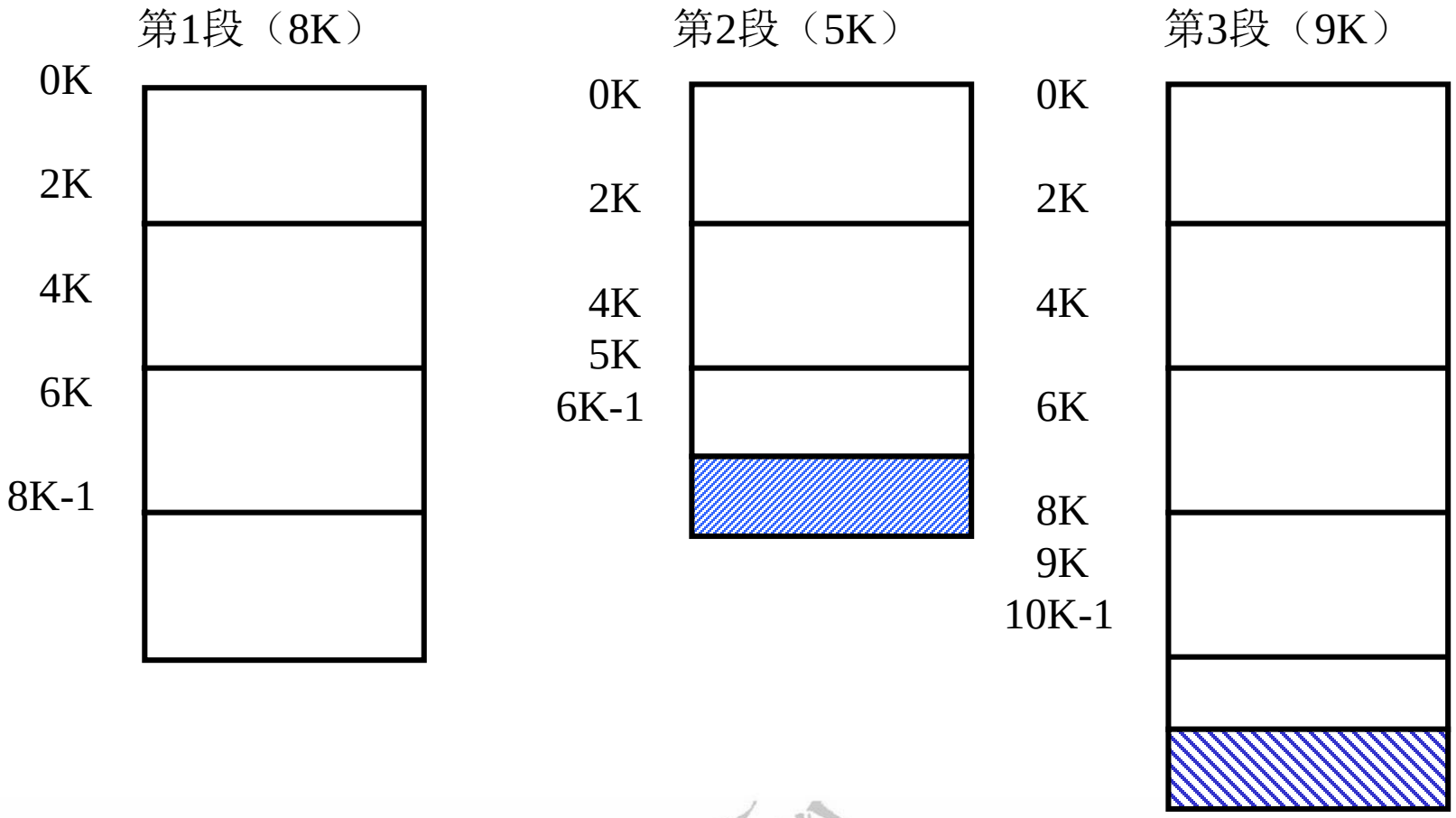
段页式存储管理的基本思想

- 逻辑地址空间的划分
  - 在段页式存储管理系统中，作业的逻辑地址空间**首先**被分成若干个**逻辑分段**；
  - **然后**再将每一段分成若干个大小**固定的页面**。

- 物理地址空间的划分
  - 将主存空间分成若干个和页面大小相同的物理块，对主存的分配以物理块为单位。

作业的分段及分页示意图

- 设作业包含分为三段，页面大小为2K。



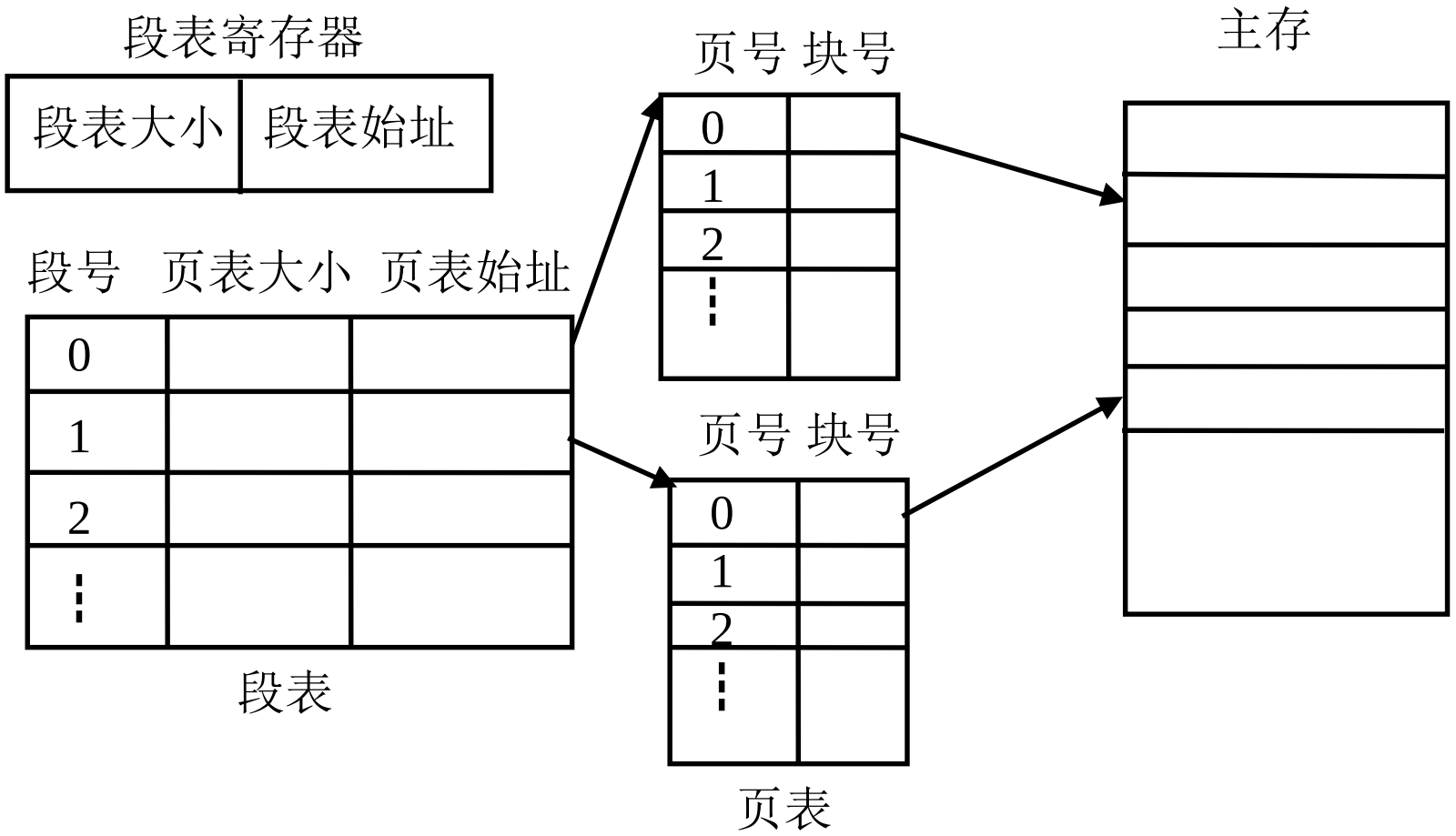
作业的逻辑地址结构

- 作业的逻辑地址结构:

段号S	段内页号P	页内位移D

- 为了实现地址变换，系统中需要设立段表及页表。
  - 一个作业对应一个段表、多个页表
- 此外，还需配置一个段表寄存器，其中存放作业的段表起始地址和段表长度。
  - 各页表起始地址和页表长度保存在段表中

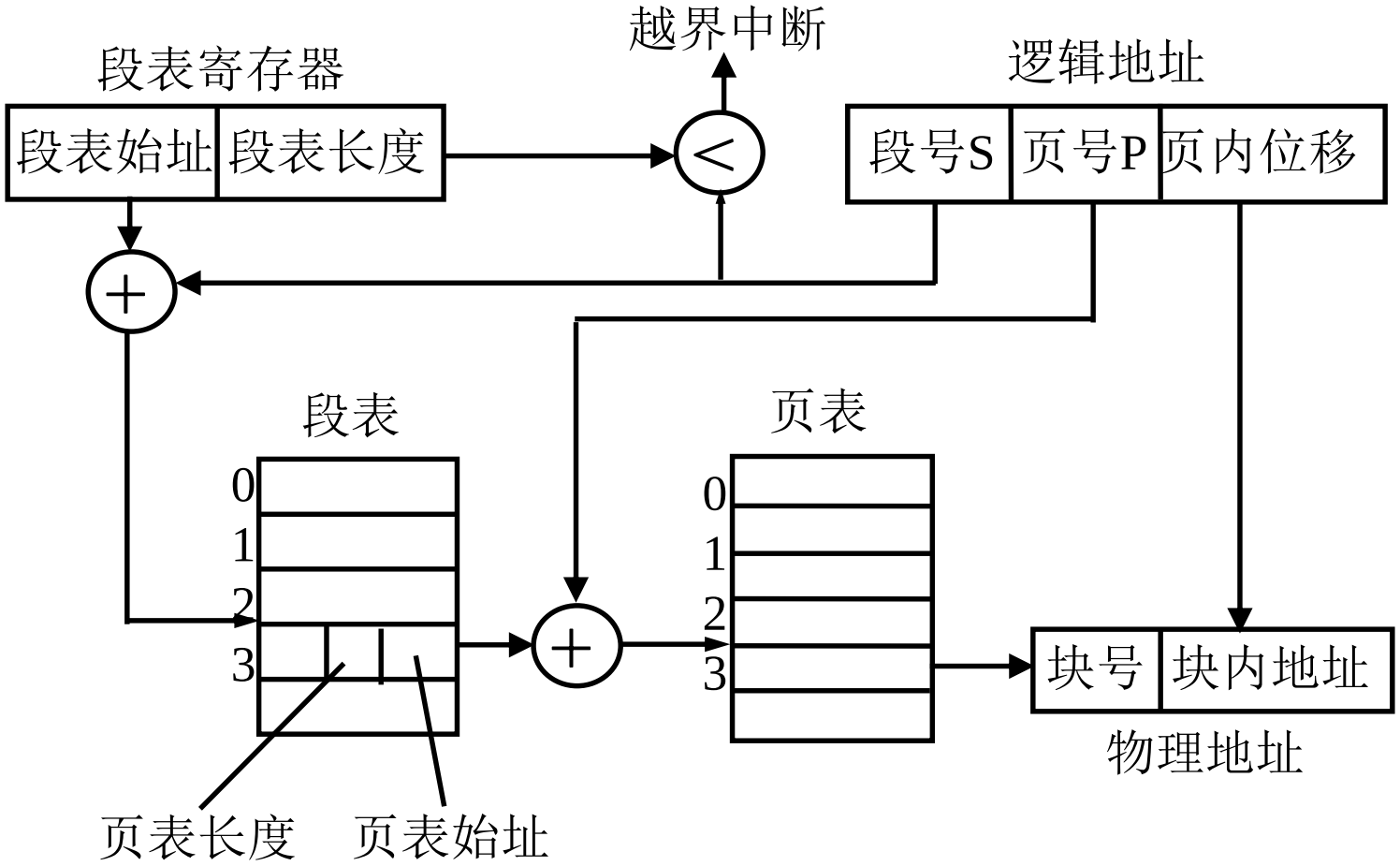
段表、页表及段表寄存器



地址变换过程

- 在进行地址变换时，首先将段号S与段表寄存器中的段表长度进行比较，若小于段表长度则表示未越界，
- 利用段表始址和段号求出该段对应段表项的位置，从中得到该段的页表始址，
- 再利用逻辑地址中的段内页号P获得对应页表项的位置，从中读出该页所在的物理块号，再与页内地址拼接成物理地址。

段页式系统中的地址变换



使用快表提高内存访问速度

- 在段页式系统中，要想存取访问信息，需要三次访问内存：
  - 第一次访问段表
  - 第二次访问页表
  - 第三次访问信息
- 为了提高访问主存的速度，应考虑使用联想寄存器。

练习题

- 假设一个任务被划分成4个大小相等的段，每段有一个大小为8项的页表，若页面大小为2KB。试问段页式存储系统中：
  - (a)每段最大尺寸是多少?
  - (b)该任务的逻辑地址空间最大为多少?
  - (c)若该任务访问到逻辑地址空间5ABCH中的一个数据，试给出逻辑地址的格式（段号、页号、页内偏移）。

作业

V9 教材 P261-262

1. 8.12
2. 8.15
3. 8.17
4. 8.20