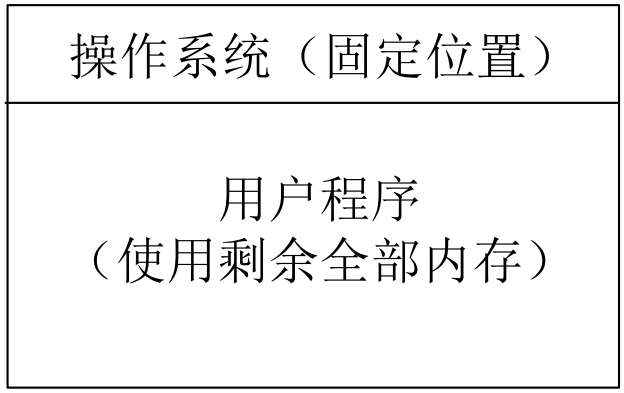
第7章 内存管理

- 本章内容
- 1. 内存管理的基本概念
- 2. 基本内存管理方案（分区、页式、段式、段页式管理方案）
- 3. 虚拟页式内存管理
- 4. 页面置换算法
- 5. 虚存性能的影响因素
- 6. 虚存相关技术

问题引入——最原始的内存使用

- 一个程序独占内存



程序直接使用物理地址

- 🤔 问题清单
  - 1. 只能运行一个程序 → CPU等待I/O时，内存闲置浪费
  - 2. 程序必须知道自己被加载到哪里 → 不同位置需要重新编译
  - 3. 没有安全保护 → 程序可以修改操作系统内存
  - 4. 程序大小 > 物理内存？ → 根本无法运行
- 💡 思考：如何改进？

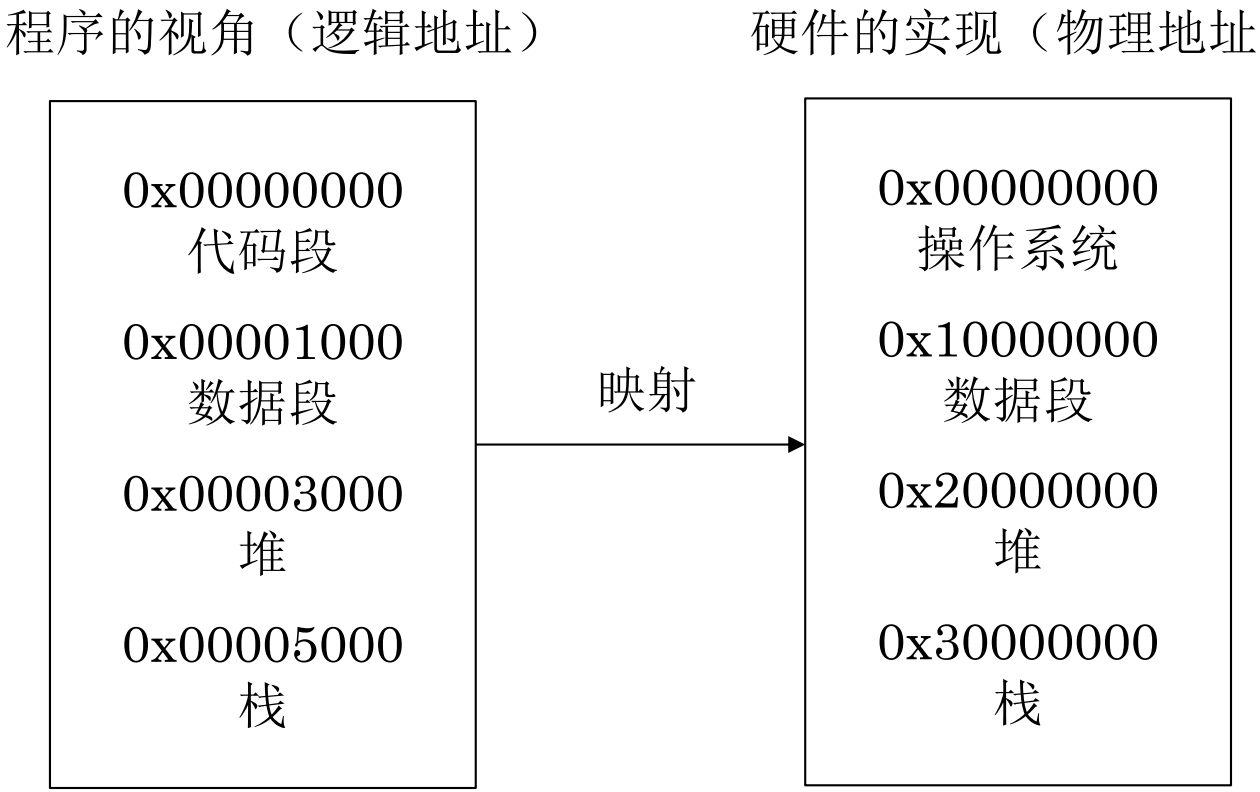
核心矛盾

- 内存管理要解决的根本矛盾
  - 有限的物理内存 vs 多个进程的内存需求

- 派生问题：
  - 如何分配？（空间管理）
  - 如何隔离？（安全保护）
  - 如何共享？（提高利用率）
  - 如何扩展？（突破物理限制）
- 解决思路：抽象 + 虚拟化 + 智能调度

概念1 - 地址空间

关键概念：地址空间抽象

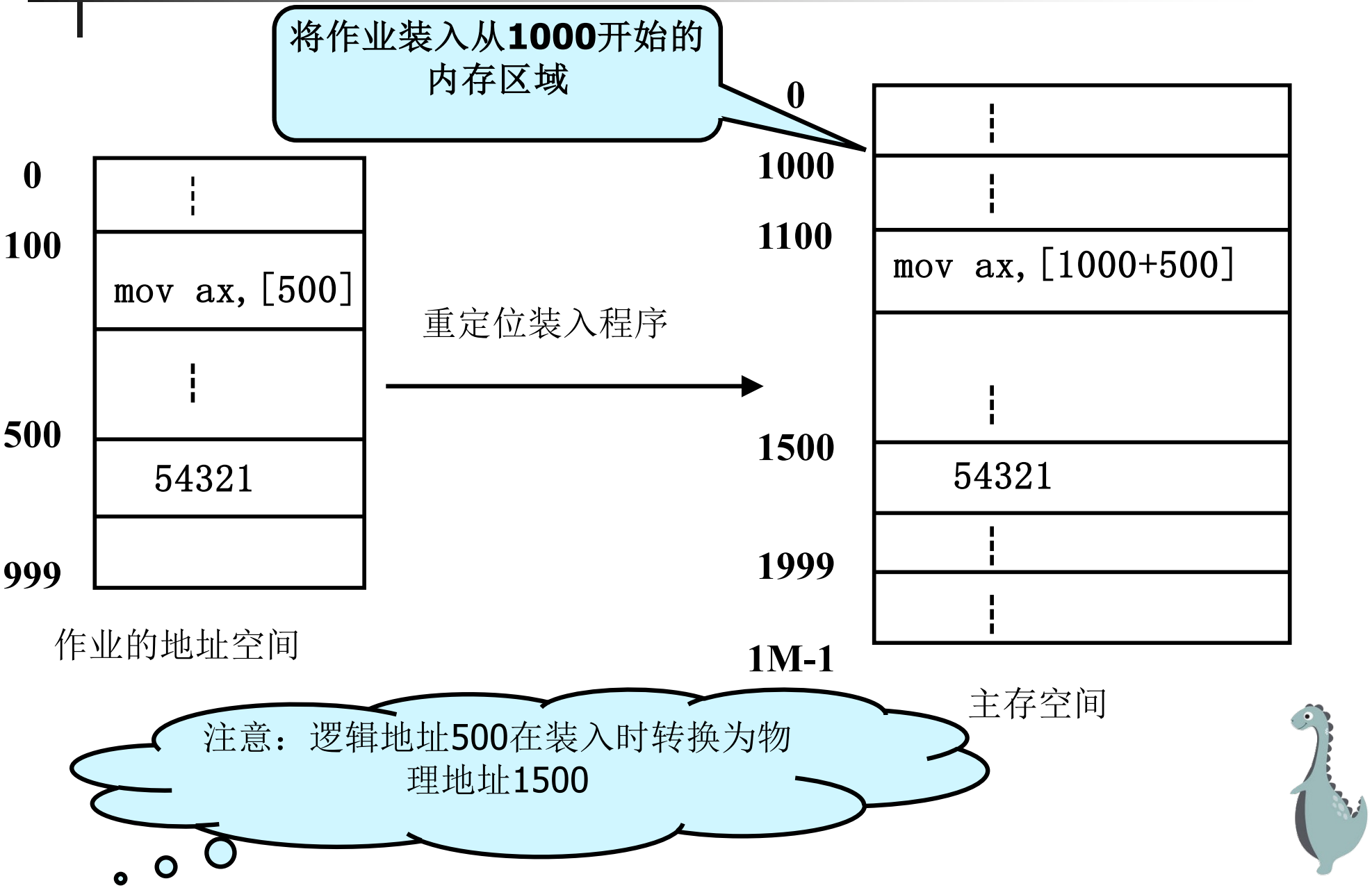


每个程序都"认为"自己从地址0开始，独占内存

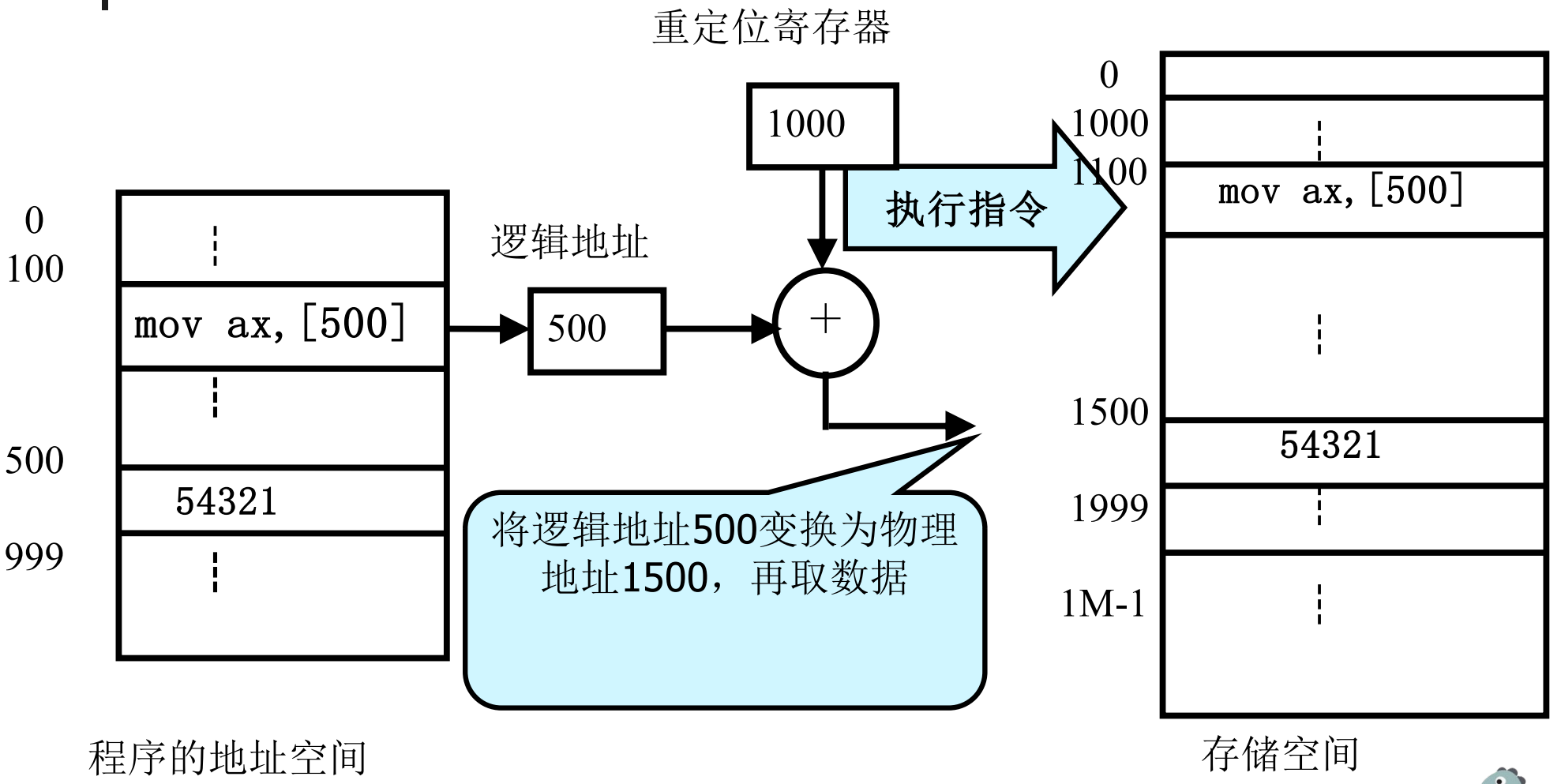
概念2 - 地址重定位

- 🔄 地址重定位：逻辑地址 → 物理地址
- 方案演进：
  - 1️⃣ 静态重定位（编译/加载时）
    - 编译器直接生成物理地址
    - ✗ 程序不能移动
    - ✗ 不支持多道程序
  - 2️⃣ 动态重定位（运行时）⭐
    - 使用基址寄存器
      - CPU发出逻辑地址: 1000 [+] 基址寄存器: 50000= 物理地址: 51000
    - ✓ 程序可加载到任意位置
    - ✓ 运行时可移动

加载时绑定示意图


将作业装入从1000开始的内存区域
0
100
500
999
mov ax,[500]
54321
作业的地址空间
重定位装入程序
0
1000
1100
1500
1999
1M-1
mov ax,[1000+500]
54321
主存空间
注意：逻辑地址500在装入时转换为物理地址1500

执行时绑定示意图

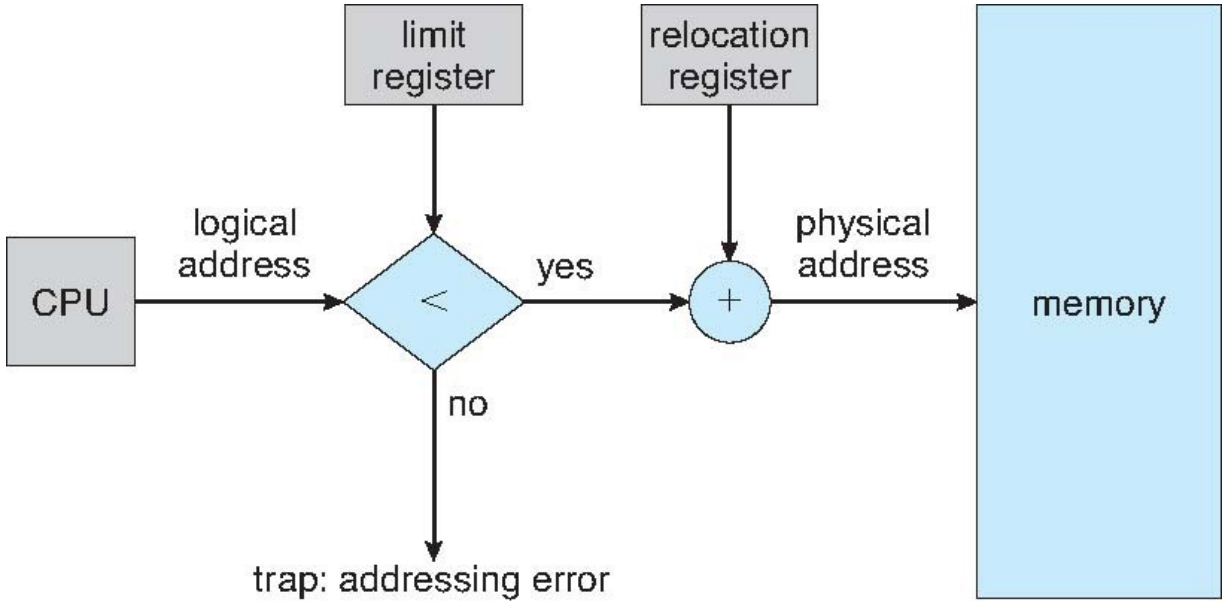

重定位寄存器
1000
执行指令
0
100
mov ax,[500]
500
54321
999
逻辑地址
500
+
将逻辑地址500变换为物理地址1500，再取数据
0
1000
1100
mov ax,[500]
1500
54321
1999
1M-1
程序的地址空间
存储空间

概念3 - 内存保护

- 内存保护：防止程序越界访问

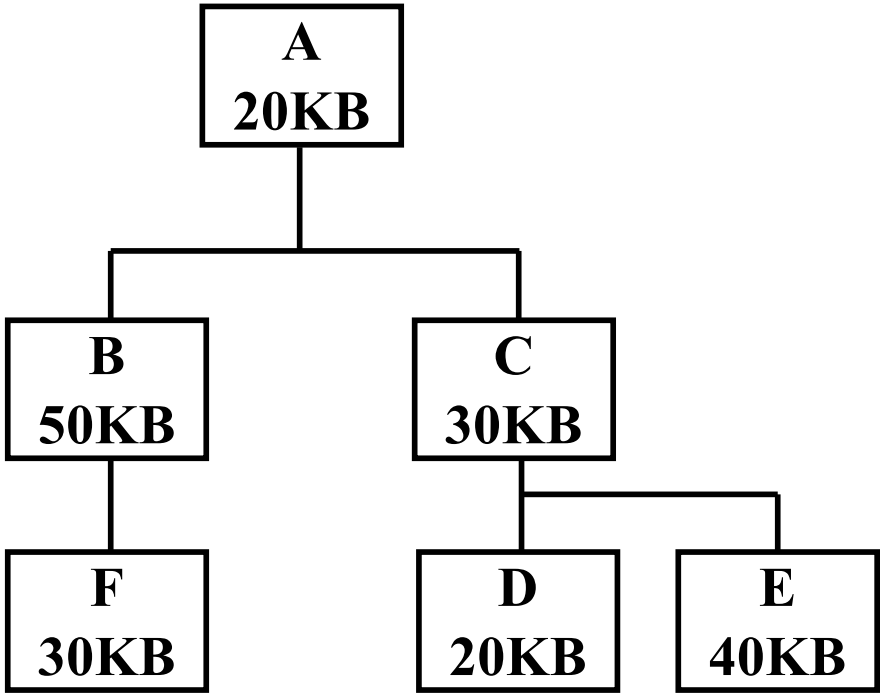
  硬件支持：基址 + 界限寄存器

基址寄存器 = 10000	← 程序起始地址
界限寄存器 = 5000	← 程序大小



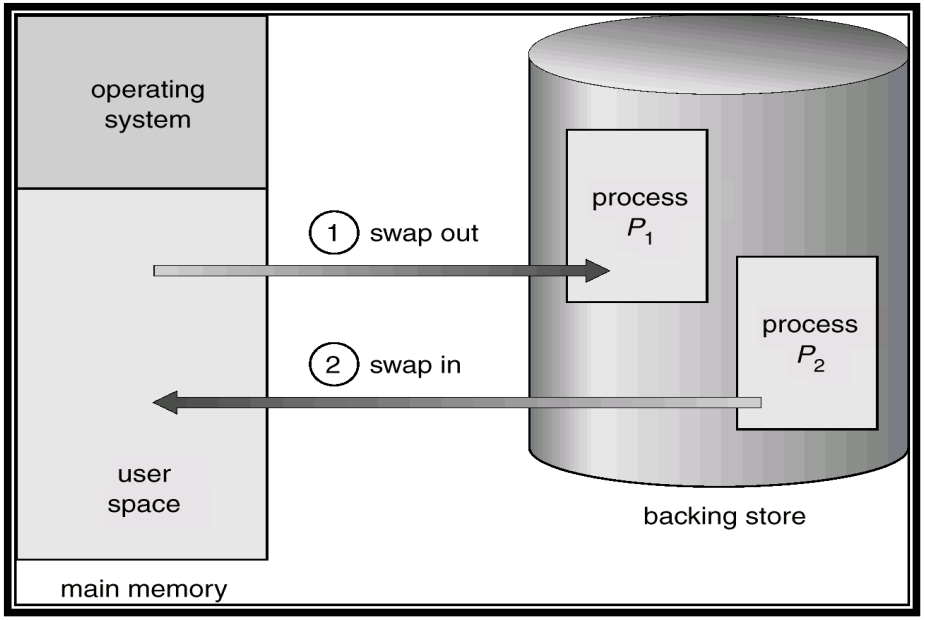
早期技术1：覆盖技术

- 覆盖技术 (Overlay)
  - 时代背景：1960年代，内存极其昂贵
  - 问题：程序 190KB，可用内存 110KB
  - 解决方案：手动将程序分成模块，轮流使用内存



早期技术2：交换技术

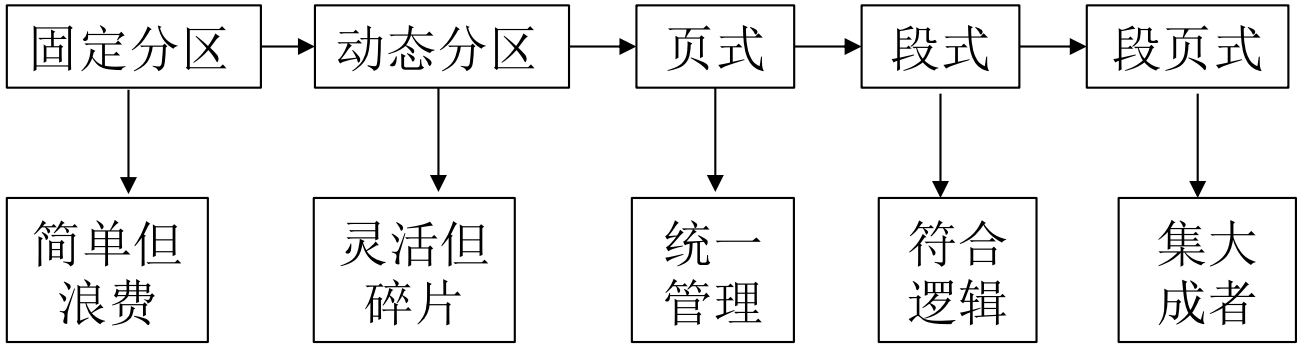
- 交换技术 (Swapping)
  - 让OS自动管理进程的换入/换出



  - 特点：
    - ✓ OS自动管理，程序员透明
    - ✓ 支持多道程序
    - ✗ 换出整个进程，开销大（10MB进程需200ms）
    - ✗ 进程必须完整换入才能运行

基本内存管理方案

- 核心问题：如何给多个程序分配内存？
- 方案演进路线：



每个方案都是为了解决前一个方案存在的问题！

方案1：固定分区

- 固定分区：预先分成几块

方案A：等大小

操作系统
100KB
100KB
100KB
100KB

方案B：不等大小

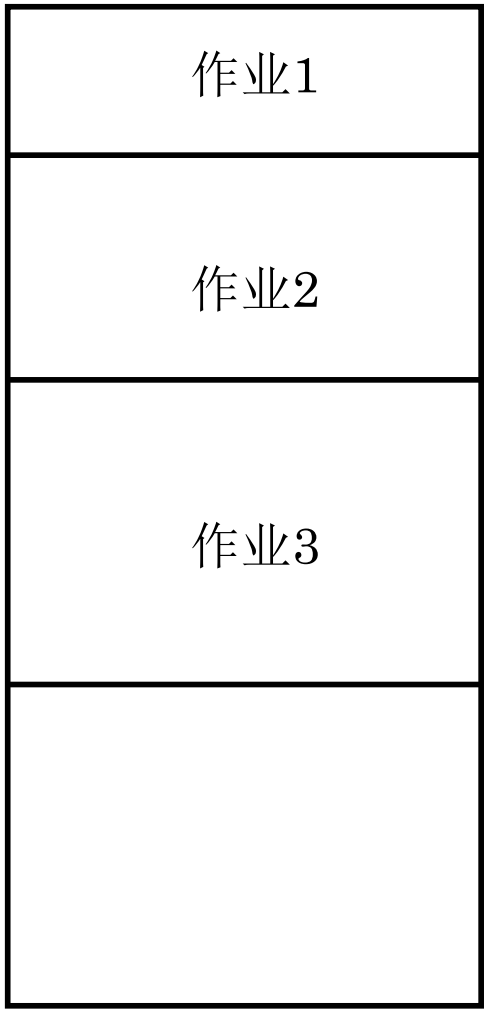
操作系统
50KB
50KB
150KB
250KB

- 缺点：
  - 内部碎片
  - 数量固定
  - 大小死板

⟹ 改进思路：动态分配
按需确定大小

方案2：动态分区

- 按程序需求分配，不预先划分分区大小
- 例：
  - 初始时，整个用户区是1个空闲块
  - 作业1进入，2个分区
  - 作业2进入，3个分区
  - 作业3进入，4个分区
  - 作业1结束，4个分区
  - 作业3结束，3个分区
  - 作业2结束，1个分区



用户区

空闲分区表示意图

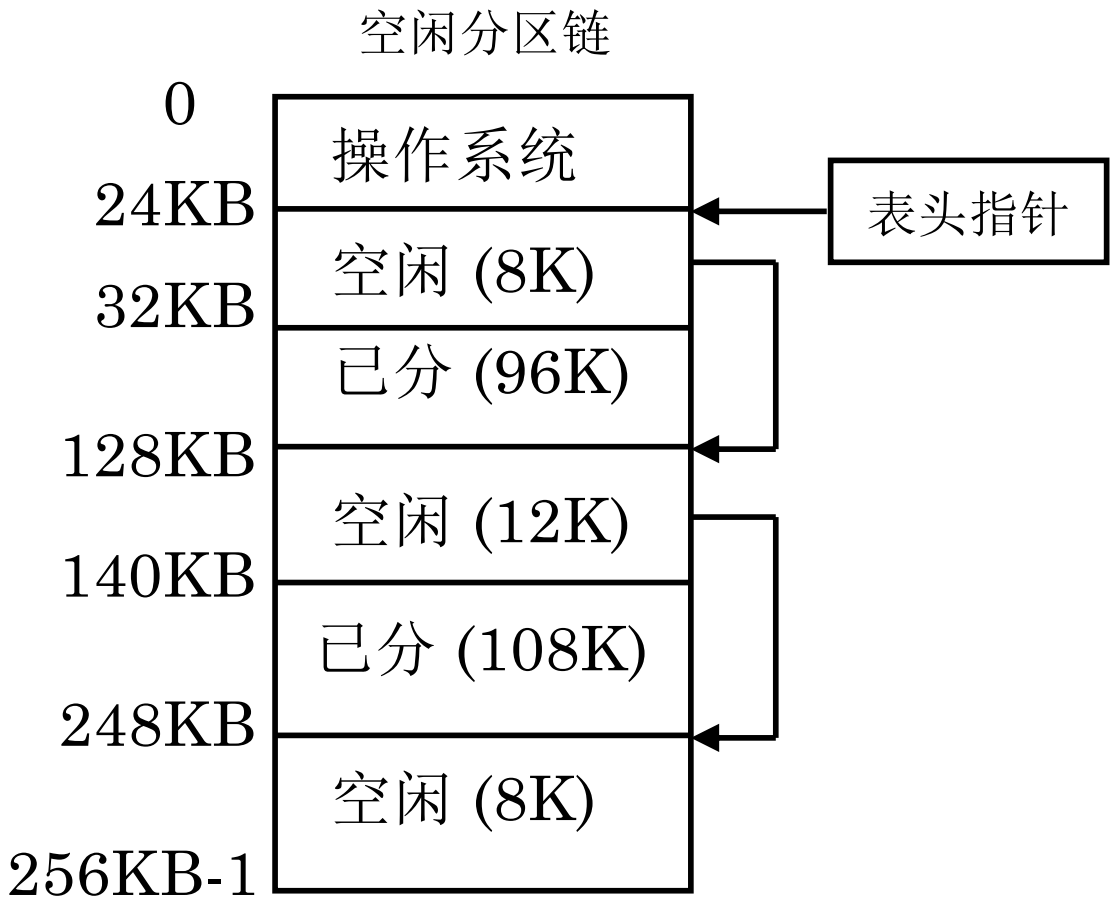
内存布局图

空闲分区表

分区号	大小	起始地址
1	8KB	24KB
2	12KB	128KB
3	8KB	248KB
4	…	…
5	…	…

空闲分区链示意图

内存布局图



方案2：动态分区

- 缺点：外部碎片问题
- 解决方案：
  - 紧凑（Compaction）- 移动所有进程 → 开销太大！
  - 改变分配策略 - 减轻碎片
    - 首次适应、最佳适应、最坏适应 → 只能缓解，无法根除

动态分区分配问题

- 如何从空闲分区中找到满足申请要求的分区？
  - 首次适应算法
  - 循环首次适应算法
  - 最佳适应算法
  - 最坏适应算法

首次适应算法 First-fit

- **首次适应算法**，该算法要求空闲分区按地址递增的次序排列。
- 分配时，从空闲分区表（或空闲分区链）首开始顺序查找，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，余下的空闲分区仍然留在空闲分区表（或空闲分区链）中。
- 特点：优先利用内存低地址端，高地址端有大空闲区。但低地址端有许多小空闲分区时会增加查找开销。

循环首次适应算法 Next-fit

- **循环首次适应算法**又称下次适应算法，它是首次适应算法的变形。
- 分配时，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，余下的空闲分区仍然留在空闲分区表（或空闲分区链）中。
- 特点：使存储空间的利用更加均衡，但会使系统缺乏大的空闲分区。

最佳适应算法 Best-fit

- **最佳适应算法**要求空闲分区按容量大小递增的次序排列。
- 分配时，从空闲分区表（或空闲分区链）首开始顺序查找，直到找到第一个能满足其大小要求的空闲分区为止。
- 如果该空闲分区大于作业的大小，则从该分区中划出一块内存空间分配给请求者，将剩余空闲区仍然留在空闲分区表（或空闲分区链）中。
- 特点：保留了大的空闲区。但分割后的剩余空闲区很小。

最坏适应算法 Worst-fit

- **最坏适应算法**要求空闲分区按容量大小递减的次序排列。
- 分配时，先检查空闲分区表（或空闲分区链）中的第一个空闲分区，若第一个空闲分区小于作业要求的大小，则分配失败；
- 否则从该空闲分区中划出与作业大小相等的一块内存空间分配给请求者，余下的空闲分区仍然留在空闲分区表（或空闲分区链）中。
- 特点：剩下的空闲区比较大，但当大作业到来时，其存储空间的申请往往得不到满足。

如何衡量分配算法的好坏

- 对于某一个作业序列来说，若某种分配算法能将该作业序列中所有作业安置完毕，则称该分配算法对这一作业序列合适，否则称为不合适。

示例

- 下表给出了某系统的空闲分区表，系统采用可变式分区存储管理策略。现有以下作业序列：96K、20K、200K。若用首次适应算法和最佳适应算法来处理这些作业序列，试问哪一种算法可以满足该作业序列的请求?

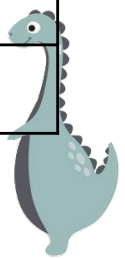
分区号	大小	起始地址
1	32K	100K
2	10K	150K
3	5K	200K
4	218K	220K
5	96K	530K

例--采用最佳适应算法分配1

- 申请96K，
- 选中5号分区，5号分区大小与申请空间大小一致，应从空闲分区表中删去该表项；

分区号	大小	起始地址
1	32K	100K
2	10K	150K
3	5K	200K
4	218K	220K
5	96K	530K

分区号	大小	起始地址
1	32K	100K
2	10K	150K
3	5K	200K
4	218K	220K



例--采用最佳适应算法分配2

- 申请20K，
- 选中1号分区，分配后1号分区还剩下12K；

分区号	大小	起始地址
1	32K	100K
2	10K	150K
3	5K	200K
4	218K	220K

分区号	大小	起始地址
1	12K	100K
2	10K	150K
3	5K	200K
4	218K	220K

例--采用最佳适应算法分配3

- 申请200K,
- 选中4号分区，分配后剩下18K。

分区号	大小	起始地址
1	12K	100K
2	10K	150K
3	5K	200K
4	218K	220K

分区号	大小	起始地址
1	12K	100K
2	10K	150K
3	5K	200K
4	18K	220K

例--采用首次适应算法分配1

- 申请96K，
- 选中4号分区，进行分配后4号分区还剩下122K；

分区号	大小	起始地址
1	32K	100K
2	10K	150K
3	5K	200K
4	218K	220K
5	96K	530K

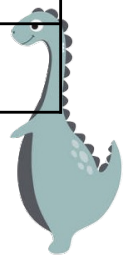
分区号	大小	起始地址
1	32K	100K
2	10K	150K
3	5K	200K
4	122K	220K
5	96K	530K

例--采用首次适应算法分配2

- 申请20K，
- 选中1号分区，分配后剩下12K；

分区号	大小	起始地址
1	32K	100K
2	10K	150K
3	5K	200K
4	122K	220K
5	96K	530K

分区号	大小	起始地址
1	12K	100K
2	10K	150K
3	5K	200K
4	122K	220K
5	96K	530K



例--采用首次适应算法分配3

- 申请200K，
- 现有的五个分区都无法满足要求，该作业等待。
- 显然采用首次适应算法进行内存分配，无法满足该作业序列的需求。

分区号	大小	起始地址
1	12K	100K
2	10K	150K
3	5K	200K
4	122K	220K
5	96K	530K

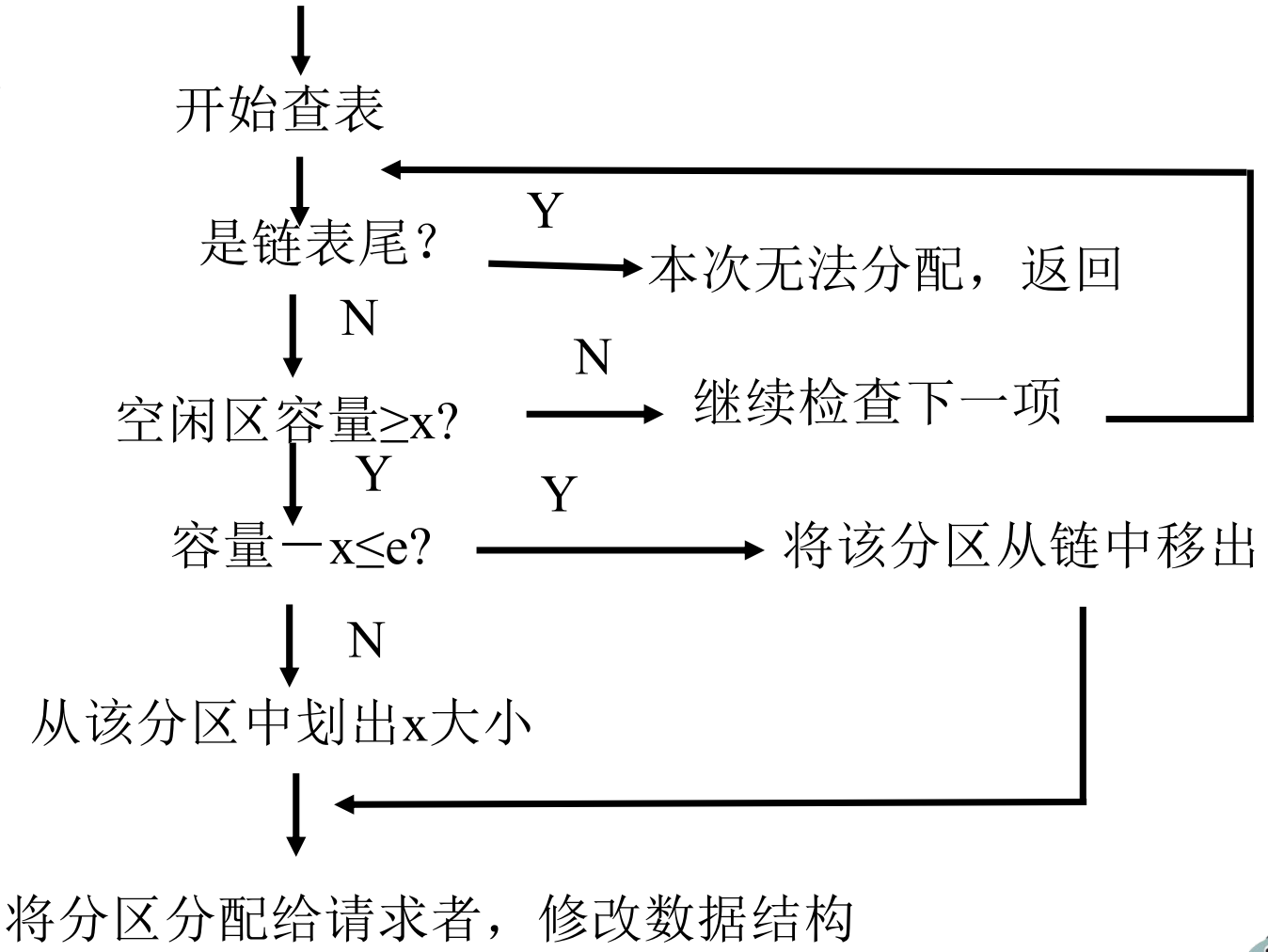
示例

- 下表给出了某系统的空闲分区表，系统采用可变式分区存储管理策略。现有以下作业序列：申请150kb，申请50kb，申请90kb，申请80kb
- 若用首次适应算法和最佳适应算法来处理这些作业序列，试问哪一种算法可以满足该作业序列的请求?

分区号	大小	起始地址
1	300K	100K
2	112K	500K

分区分配

- 以首次适应算法及空闲链表为例，申请分区大小为x，e是规定的不再分割的剩余区大小

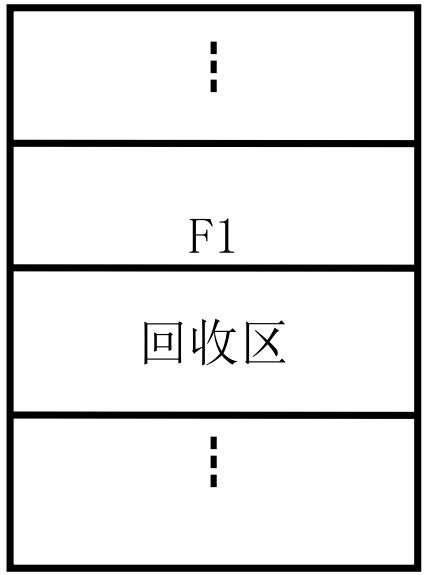


分区回收

- 回收分区时，应将空闲区插入适当位置，此时有以下四种：
  - 回收分区r上面邻接一个空闲分区
  - 回收分区r下面邻接一个空闲分区
  - 回收分区r上面、下面各邻接一个空闲分区
  - 回收分区r不与任何空闲分区相邻

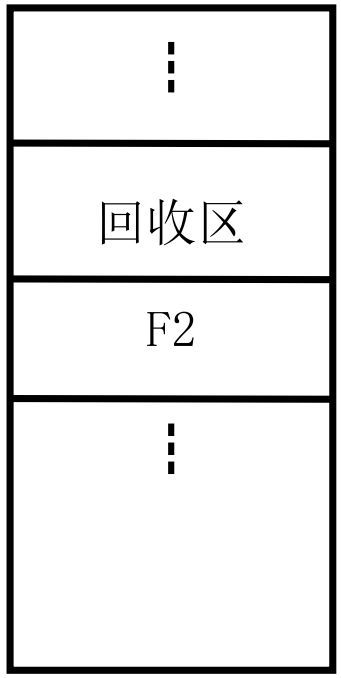
回收分区r上邻接一个空闲分区

- 此时应将回收区r与上邻接分区F1合并成一个连续的空闲区；
- 合并分区的首地址为空闲区F1的首地址，
- 其大小为二者之和



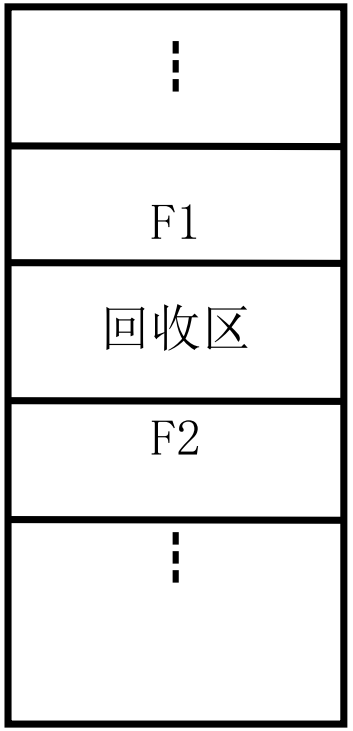
回收分区r下邻接一个空闲分区

- 此时应将回收区r与下邻接分区F2合并成一个连续的空闲区；
- 合并分区的首地址为回收分区r的首地址，
- 其大小为二者之和



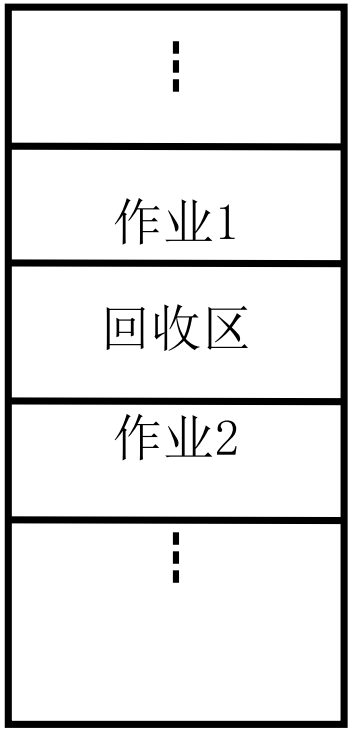
回收分区r上下邻接空闲分区

- 此时应将回收区r与上、下邻接分区合并成一个连续的空闲区；
- 合并分区的首地址为与r上邻接空闲区F1的首地址，
- 其大小为三者之和，
- 且应将与r下邻接的空闲区F2从空闲分区表(或空闲分区链)中删去



回收分区r不与任何空闲分区相邻

- 这时应为回收区单独建立一个新表项，填写分区大小及起始地址等信息，并将其加入到空闲分区表(或空闲分区链)中的适当位置。



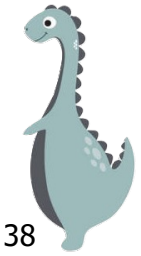
问题：空闲分区的个数在上述几种情况下如何变化？

碎片问题（Fragmentation）

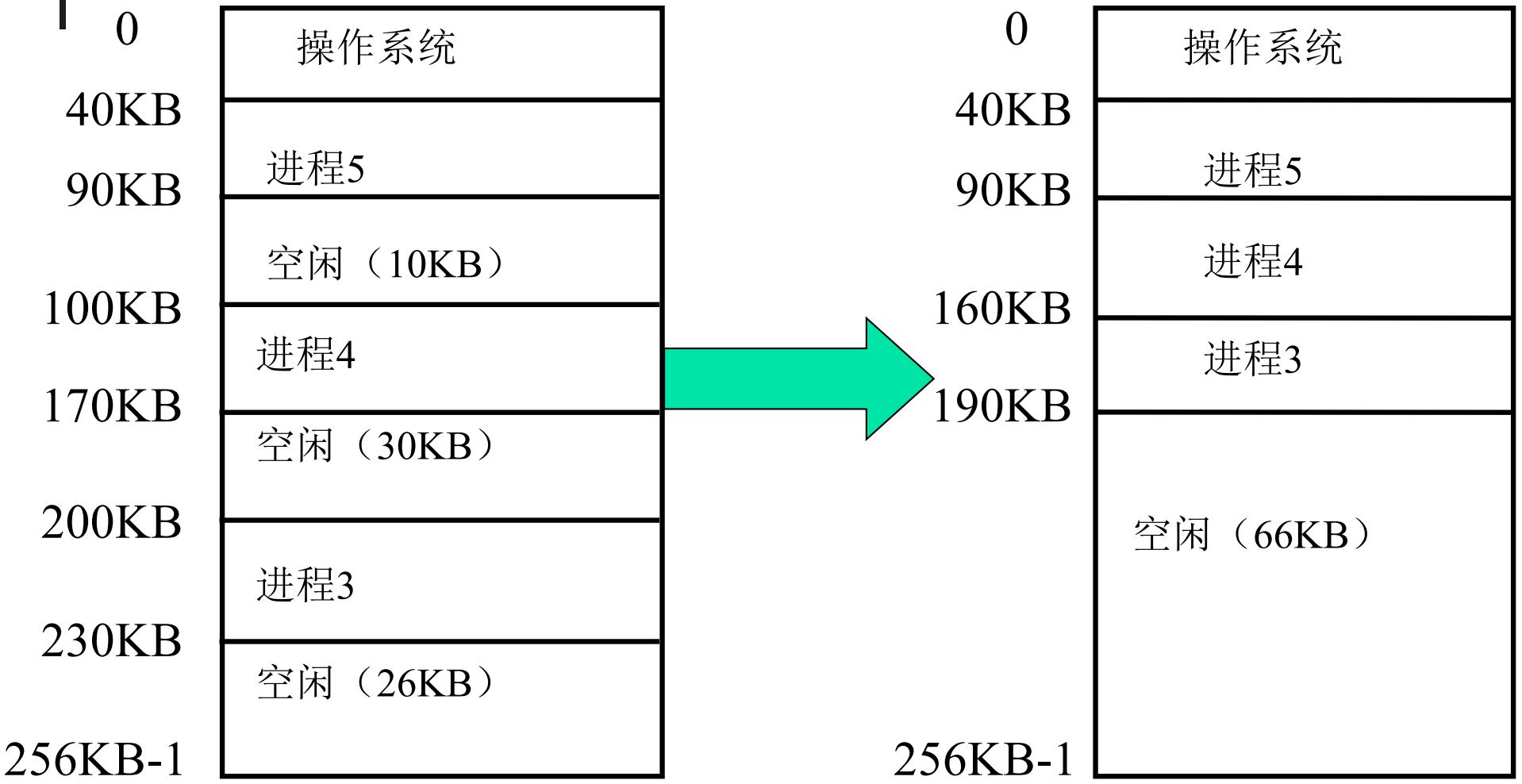
- 外部碎片——内存空间合起来可以满足请求，但不是连续的
- 内部碎片——已分配的内存可能略大于请求的内存；存在于分区内部，但未被使用
- 首次适应分析显示，在给定N个分配块的情况下，有0.5N个分区因碎片化而丢失。
  - 1/3可能无法使用 -> 50%规则
- 可以通过紧凑(compaction)来减少外部碎片

紧凑解决外部碎片问题

- 需移动内存内容，使所有空闲内拼成一个大的分区
- 需要运行时动态重定位的支持
- 若紧凑时进行了I/O操作，数据将到达错误的位置
- 解决I/O问题的方法：
  - 参与I/O操作的进程在内存中保持锁定状态
  - 仅将I/O操作执行到操作系统缓冲区中
- 磁盘上的存储具有相同的碎片问题
- 紧凑需要耗费大量处理机时间



拼接示意图

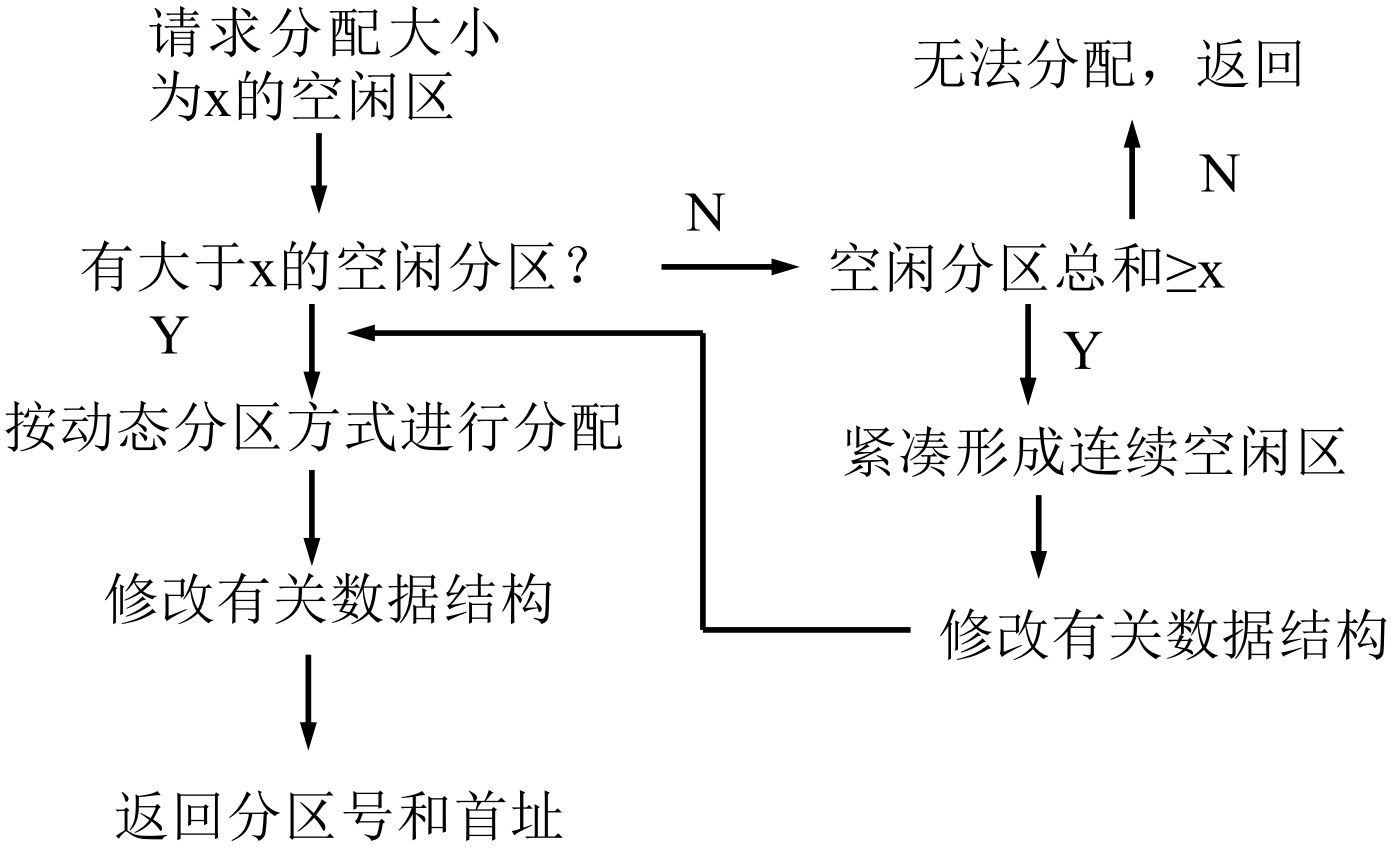

0
操作系统
40KB
进程5
90KB
空闲（10KB）
100KB
进程4
170KB
空闲（30KB）
200KB
进程3
230KB
空闲（26KB）
256KB-1
0
操作系统
40KB
进程5
90KB
进程4
160KB
进程3
190KB
空闲（66KB）
256KB-1

拼接需要的技术支持

- 拼接后程序在内存的位置发生变化，因此需要动态重定位技术支持。
- 空闲区放在何处：拼接后的空闲区放在何处不能一概而论，应根据移动信息量的多少来决定。
- 拼接的时机：
  - 回收分区时拼接：只有一个空闲区，但拼接频率过高增加系统开销。
  - 找不到足够大的空闲区且系统空闲空间总量能满足要求：拼接频率小于前者，空闲区管理稍复杂。也可以只拼接部分空闲区。

可重定位分区分配技术

- 可重定位分区分配算法与动态分区分配算法基本相同，差别仅在于：在这种分配算法中增加了拼接功能。



伙伴系统

- 固定分区存储管理限制了内存中的进程数，动态分区的拼接需要大量时间，而伙伴系统是一种较为实用的动态存储管理办法。
- **伙伴系统**采用伙伴算法对空闲内存进行管理。该方法通过不断对分大的空闲存储块来获得小的空闲存储块。当内存块释放时，应尽可能合并空闲块。

伙伴系统的内存分配

- 设系统初始时可供分配的空间为2^m个单元。
- 当进程申请大小为n的空间时，设$2^{i-1}<n\leq 2^i$，则为进程分配大小为2^i的空间。
- 如系统不存在大小为2^i的空闲块，则查找系统中是否存在大于2^i的空闲块，若找到则对其进行对半划分，直到产生大小为2^i的空闲块为止。

伙伴系统的内存回收

- 当一块被分成两个大小相等的块时，这两块称为伙伴。
- 当进程释放存储空间时，应检查释放块的伙伴是否空闲，若空闲则合并。这个较大的空闲块也可能存在空闲伙伴，此时也应合并。重复上述过程，直至没有可以合并的伙伴为止。

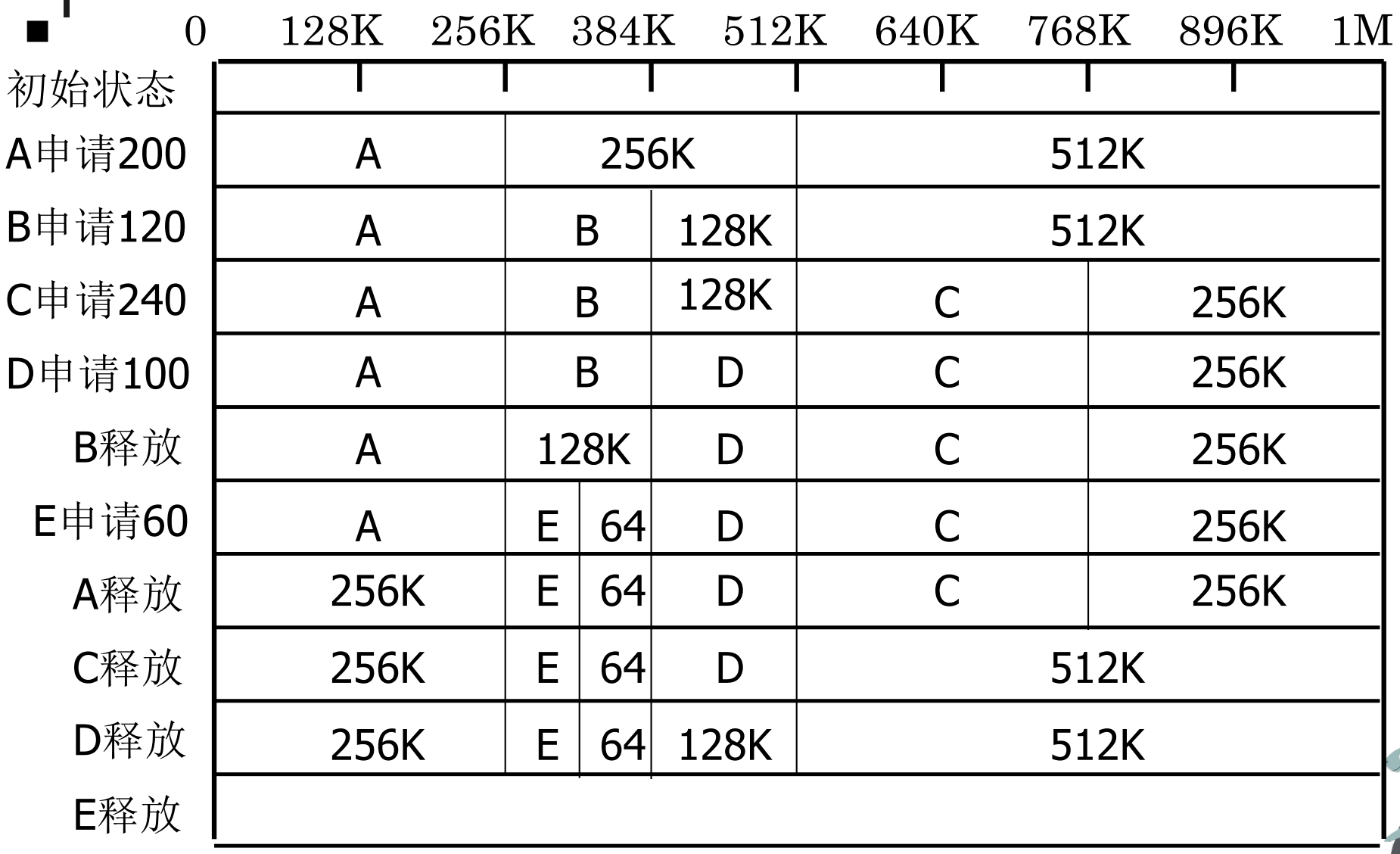
伙伴地址公式

- 设某空闲块的开始地址为d，长度为2^K，其伙伴的开始地址为：
  - Buddy（k，d）=d+2^k，若d % 2^{k+1}=0
  - =d-2^k，若d % 2^{k+1}= 2^k
- 如果参与分配的2^m个单元从a开始，则长度为2^K、开始地址为d的块，其伙伴的开始地址为：
  - Buddy（k，d）=d+2^k，若（d-a）% 2^{k+1}=0
  - =d-2^k，若（d-a）% 2^{k+1}= 2^k

伙伴系统分配及回收例

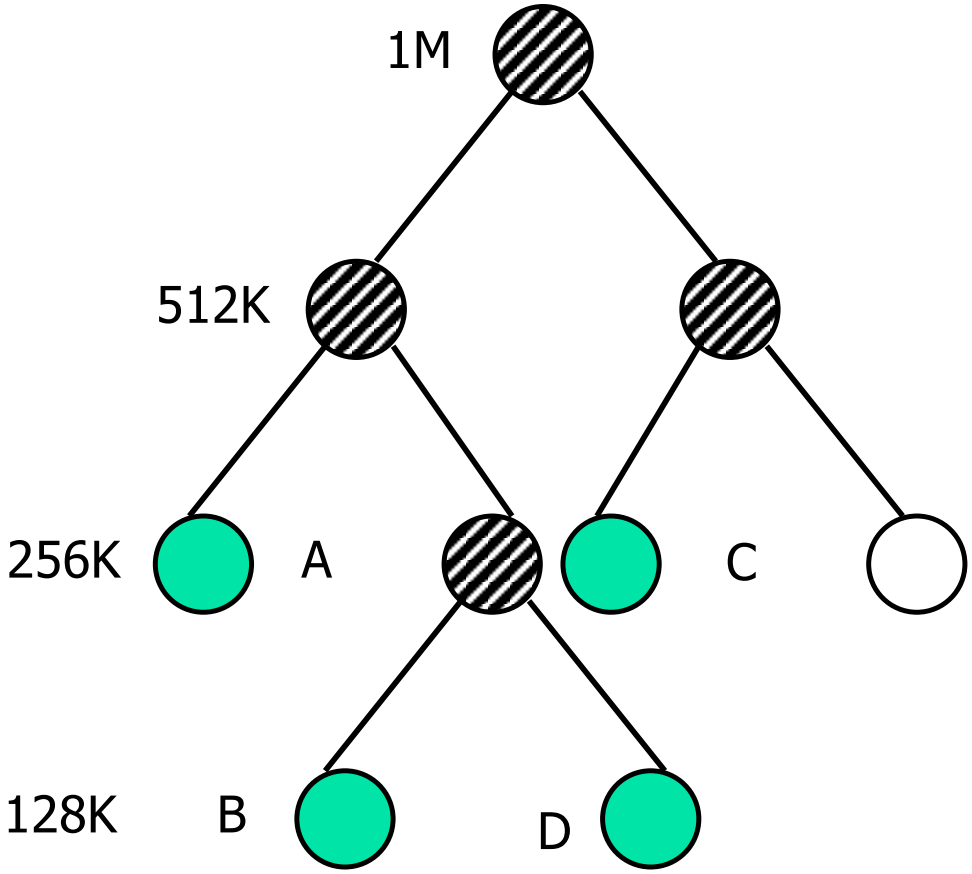
- 设系统中初始内存空间大小为1MB，进程请求和释放空间的操作序列为：
  - 进程A申请200KB；B申请120KB；C申请240KB； D申请100KB；
  - 进程B释放；E申请60KB；
  - 进程A、C释放；
  - 进程D释放；进程E释放。

分配过程示意图


0
128K
256K
384K
512K
640K
768K
896K
1M
初始状态
A申请200
A
256K
512K
B申请120
A
B
128K
512K
C申请240
A
B
128K
C
256K
D申请100
A
B
D
C
256K
B释放
A
128K
D
C
256K
E申请60
A
E
64
D
C
256K
A释放
256K
E
64
D
C
256K
C释放
256K
E
64
D
512K
D释放
256K
E
64
128K
512K
E释放

伙伴系统的二叉树表示

- 可以用二叉树表示内存分配情况。叶结点表示存储器中的当前分区，如果两个伙伴是叶子，则至少有一个被分配。
- 右图表示A（200）、B（120）、C（240）、D（100）分配之后的情况。

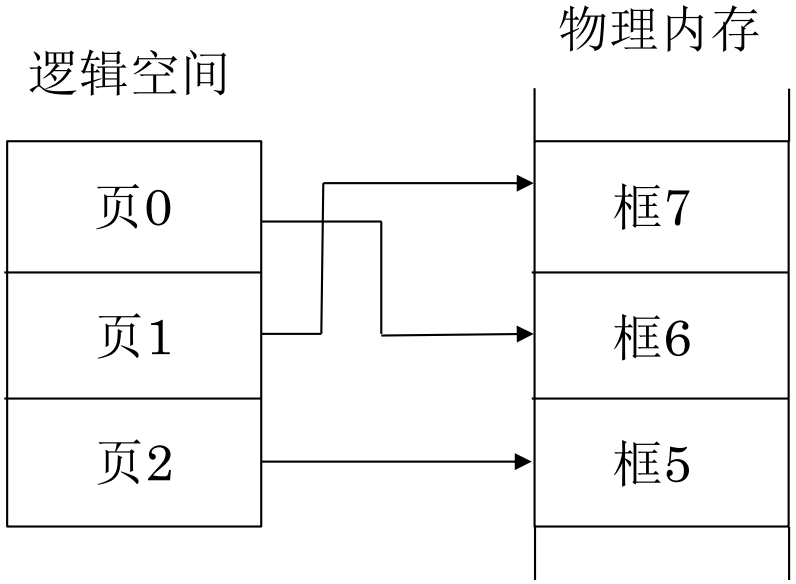


伙伴系统的不足

- 分配和回收时需要对伙伴进行分拆及合并
- 存储空间有浪费

方案3：页式管理

- 核心洞察："把内存和程序都切成相同大小的小块"
  - 逻辑地址空间 → 页 (Page)
  - 物理内存 → 页框/帧 (Frame)
  - 页大小 = 页框大小 (通常4KB)

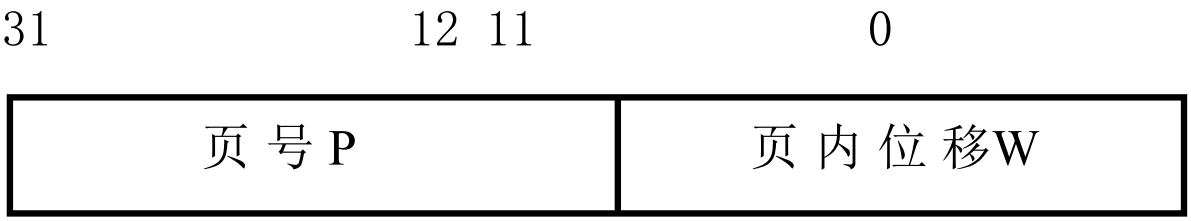


基本方法

- 实现分页的基本方法：将物理内存分为固定大小的块，称为**帧**（或称物理块、页框），将逻辑内存也分成同样大小的块，称为**页**。
- 在为进程分配存储空间时，总是以块为单位来分配，可以将进程中的某一页存放到主存的某一空闲块中。
- 分页系统中有碎片吗？

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(Page number)和页内位移(Page offset)组成。其结构如下所示：

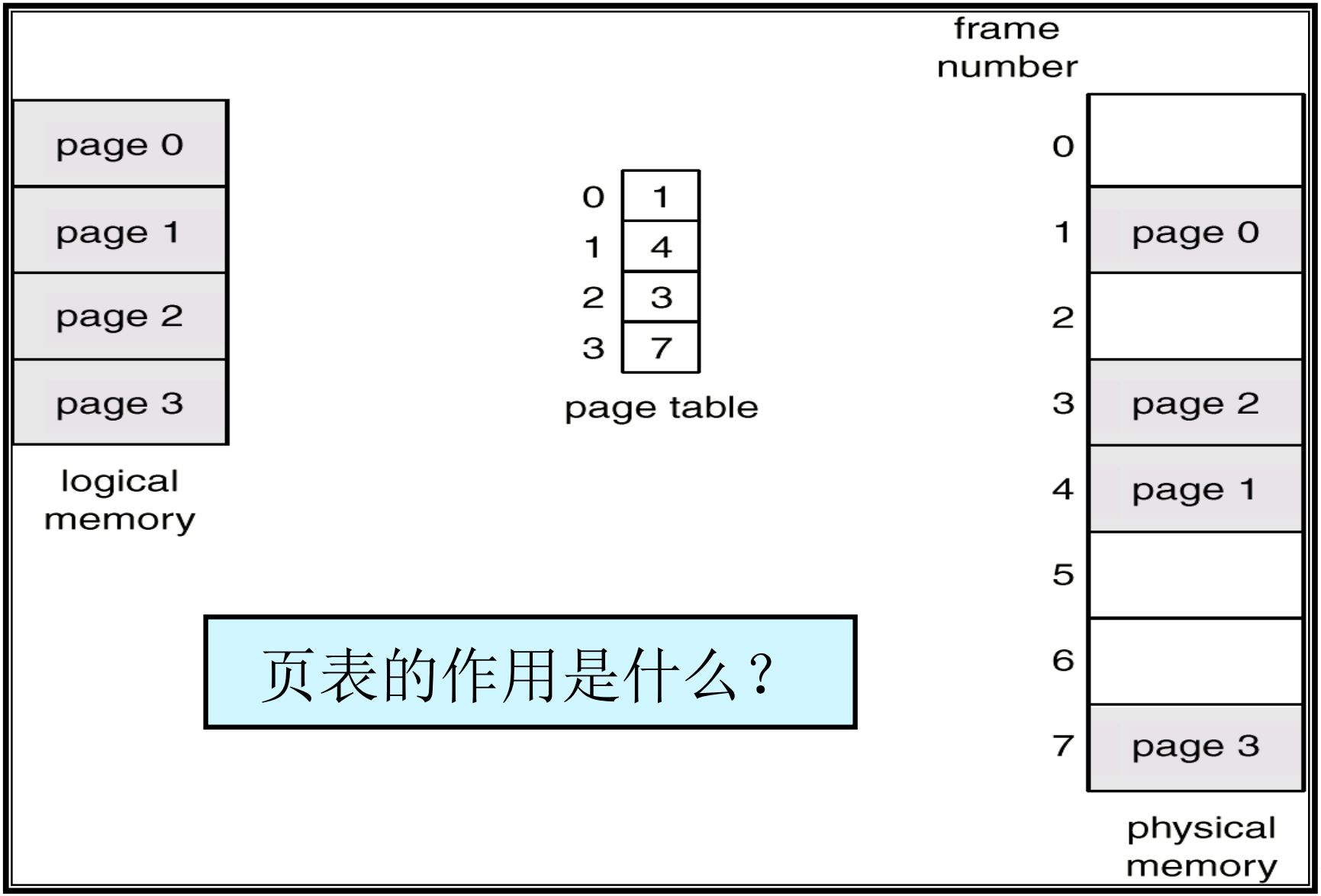


- 若A为逻辑地址，L为页面大小，则：
- 页号：　　P=int(A/L)
- 页内位移：W=A % L

页表 page table

- 为了在内存中找到进程的每个页面所对应的物理块，系统为每个进程建立一张页面映象表，简称页表。
- 页表：记录页面在内存中对应物理块的数据结构。
- 注意：Page frame英文词在目前国内操作系统教材中翻译的有物理块（块）、页框、页架、页帧和帧等，读者应加以注意，它表示的是与一个逻辑页相对应的一个物理块。

Paging Example



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，同时页内碎片也较大。若页面太小，虽然可减少页内碎片，但会导致页表增长。
- 因此，页面大小应适中，通常为2的幂，大小范围从512B到16MB不等，通常为512B到8KB之间。还可以支持多种页大小。
- 页表一般存放在内存中。也可以在页表中设置存取控制字段，以实现存储保护。

存储分块表

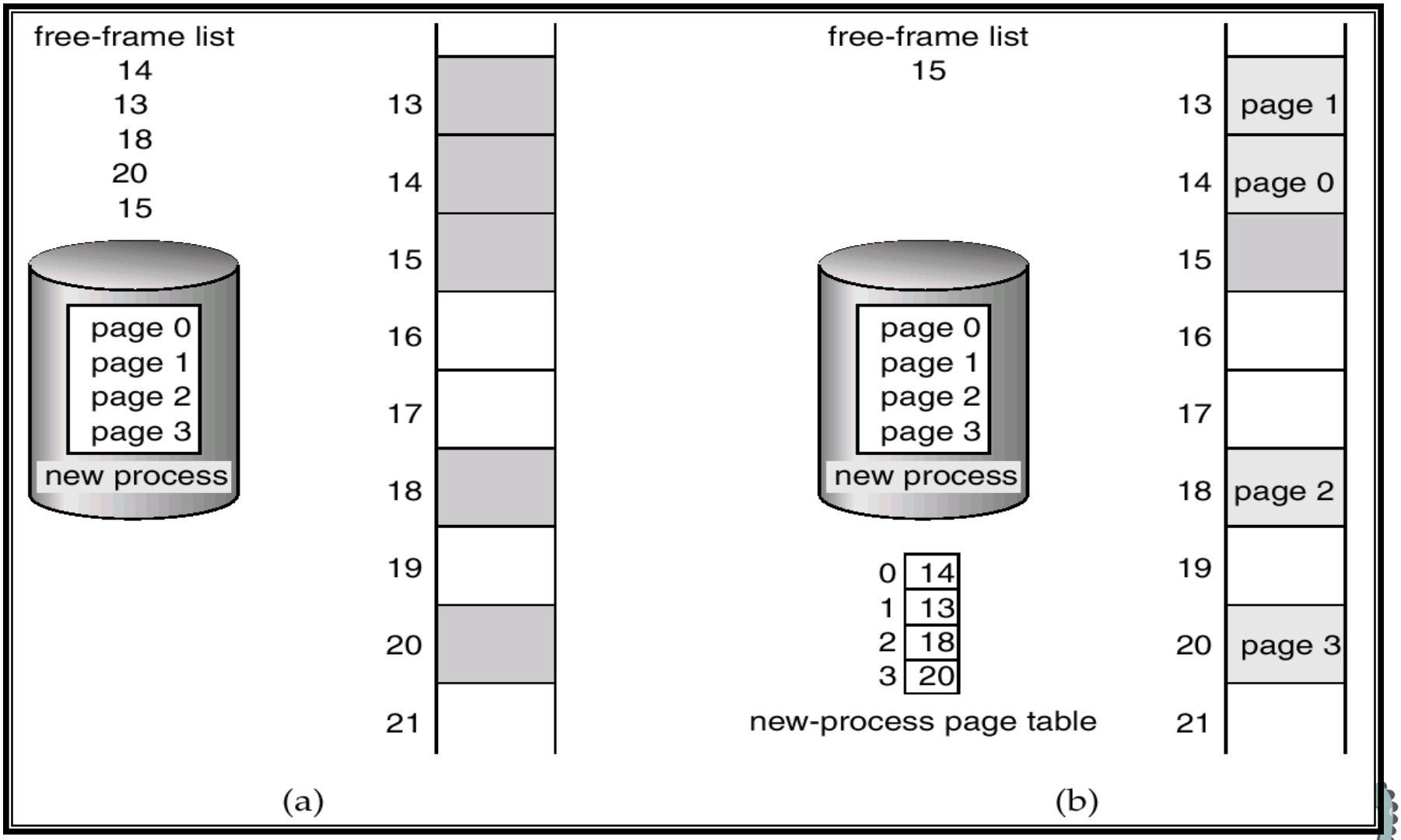
- 存储分块表用来记录内存中各物理块的使用情况及未分配物理块总数。也称为帧表（frame table）
- 存储分块表可用下述方式表示：
  - 位示图：利用二进制的一位表示一个物理块的状态，1表示已分配，0表示未分配。所有物理块状态位的集合构成位示图。
  - 空闲存储块链：将所有的空闲存储块用链表链接起来，利用空闲物理块中的单元存放指向下一个物理块的指针。

位示图例

	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	…															
⋮																

- 位示图占用的存储空间：物理块数/8（字节）

空闲帧 Free Frames



Before allocation

After allocation

存储空间的分配及回收

- 页面分配：计算进程所需页面数，然后在请求表中登记进程号、请求页面数等。如存储分块表中有足够的空闲块可供进程使用，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。
- 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，并释放页表，修改请求表中的页表始址及状态。

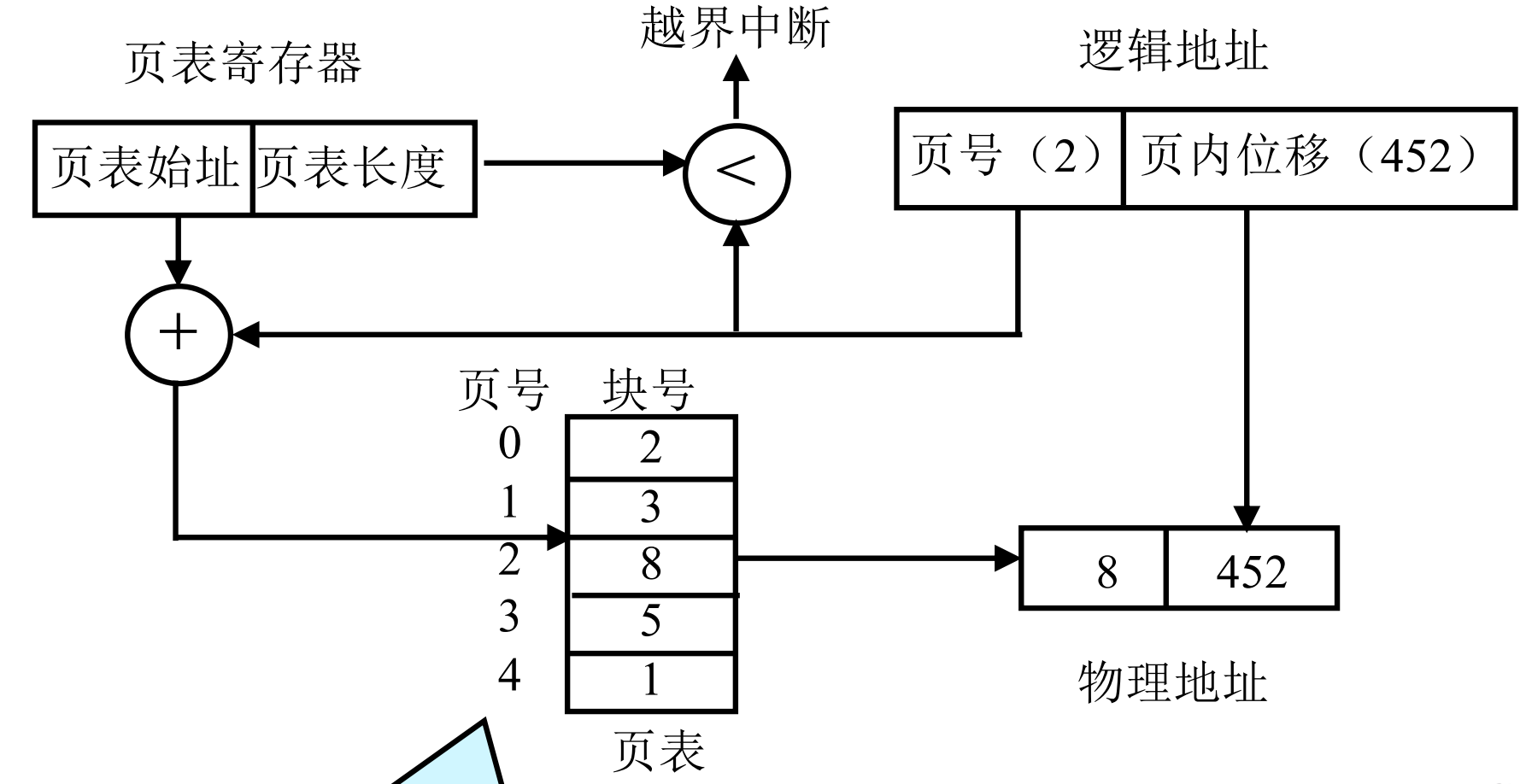
地址变换机构

- 地址变换机构的任务是实现逻辑地址到物理地址的变换，即将逻辑地址中的页号转换为内存中的物理块号
- 页表通常存放在内存中，为了实现方便，系统中设置了一个页表寄存器存放页表在内存的起始地址和页表的长度。
- 进程未执行时，页表的起始地址和长度存放在PCB中。当进程执行时，才将页表始址和长度存入页表寄存器中。

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。

分页系统的地址变换机构图



注意这里的页号字段？

分页地址变换示例

- 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2500的页号及页内地址为：
- 2500/1024=2（页号）；2500 %1024＝452（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＝8644。

地址变换示例

- 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，用十六进制表示块号为B，所以物理地址为：
- 1011　1001101111，即2E6FH。

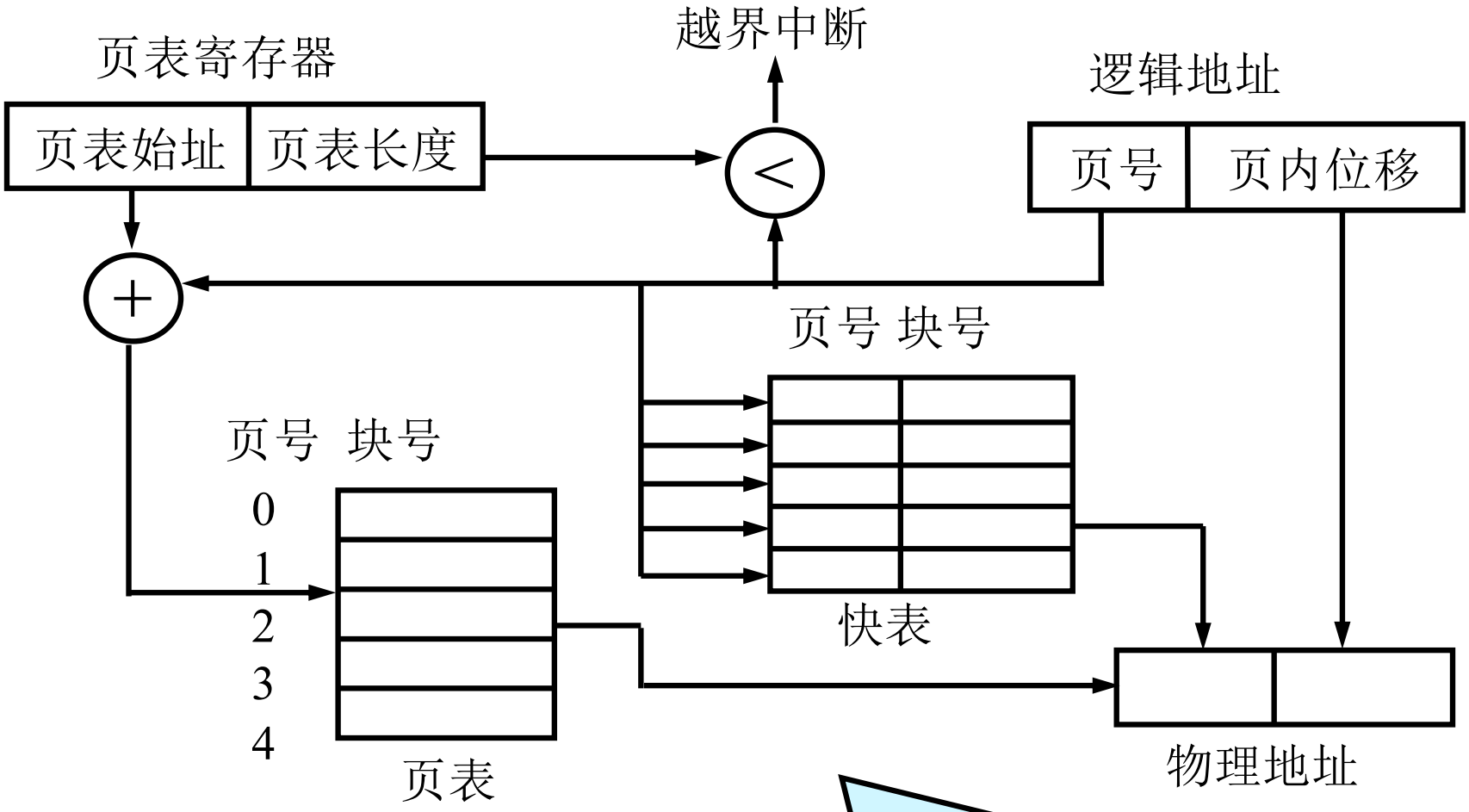
具有快表的地址变换机构

- 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(associative memory)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。

具有联想存储器的地址变换



此处页表与快表有何不同？

联想存储器的大小

- 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，联想存储器的命中率可达80%--90% 。

有效访问时间

- 假设内存一次存取时间是m，联想寄存器的查找时间是n，命中率为p
- 有效存取时间 （假定忽略快表更新时间）

  EAT=p*(n+m)+(1-p)(2m+n)

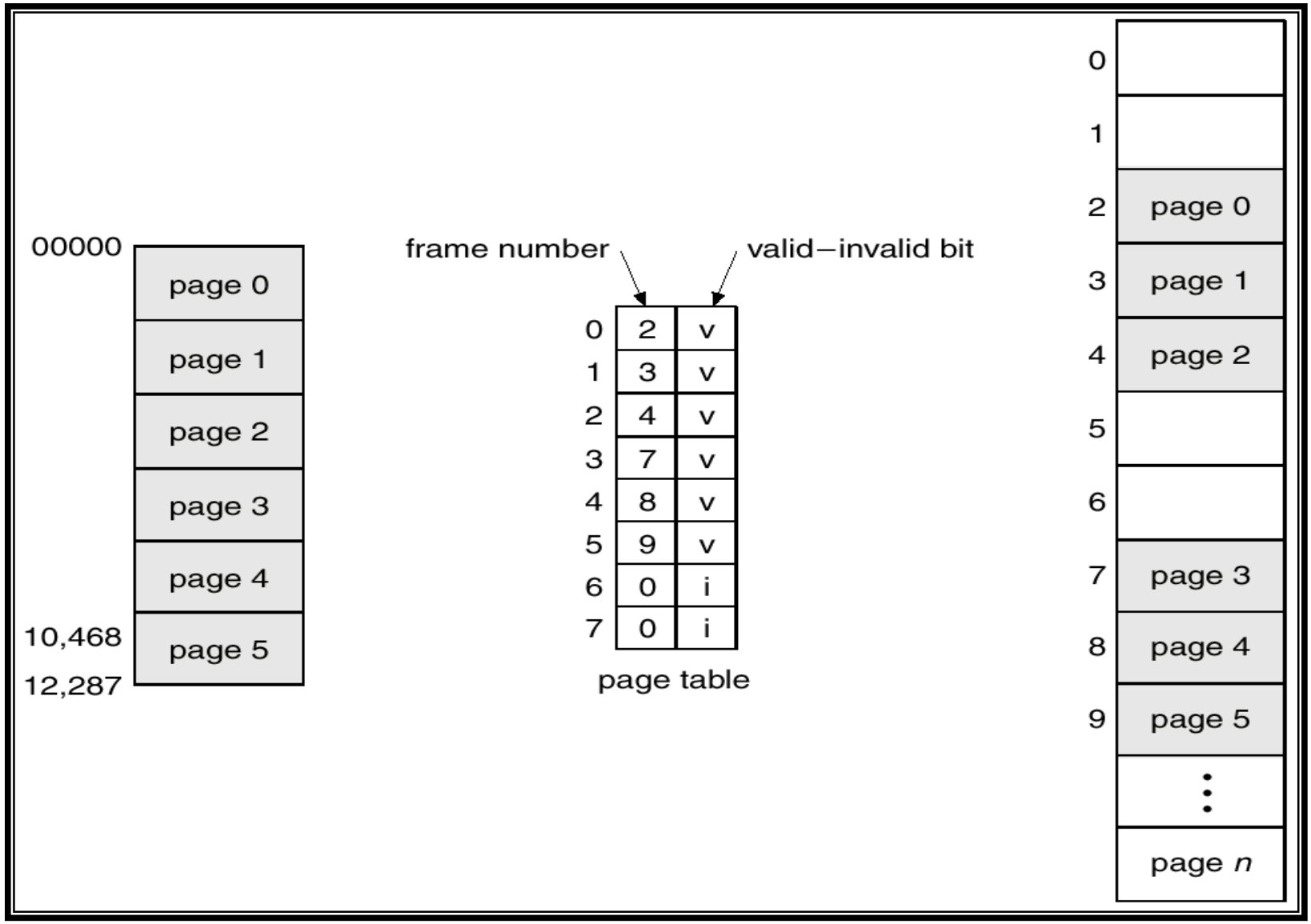
  若p=0.8，m=100ns，n=20ns

  则EAT=0.8*120+0.2*220=140ns

存储保护

- 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

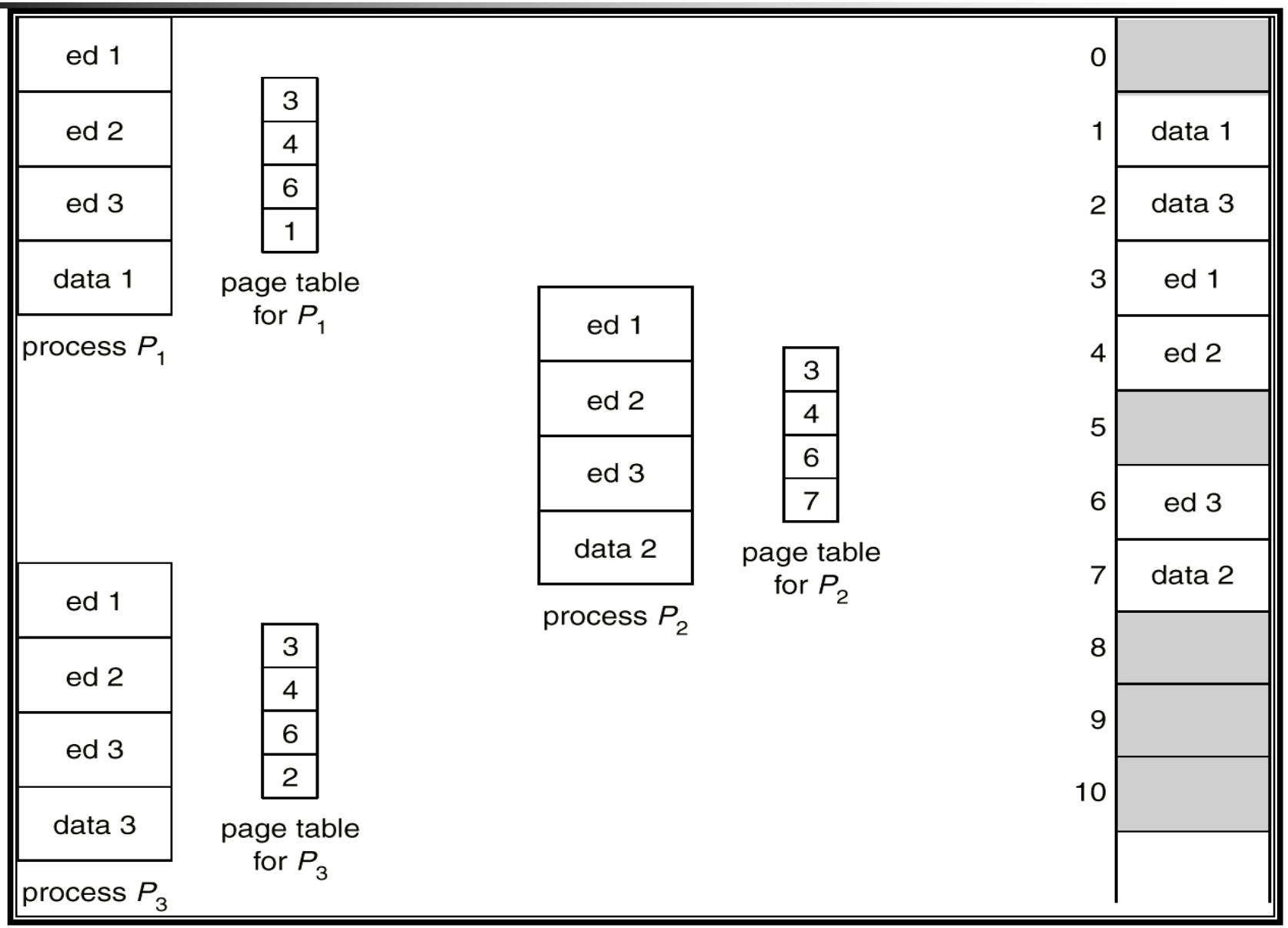
Valid (v) or Invalid (i) Bit In A Page Table



共享页 Shared Pages

- 分页的优点之一是可以共享公共代码
- 如果代码是可重入代码，则可以共享
  - 又称为纯代码，是不能自我修改的代码，它在执行期间不会改变
  - 因此两个或更多的进程可以在相同的时间执行相同的代码
- 信息的共享是通过使多个进程页表项指向同一个物理块来实现的

分页环境中的代码共享


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

页表结构

- 组织页表的常用技术
  - 分级页表 Hierarchical Paging
  - 哈希页表 Hashed Page Tables
  - 反向页表 Inverted Page Tables

分级页表

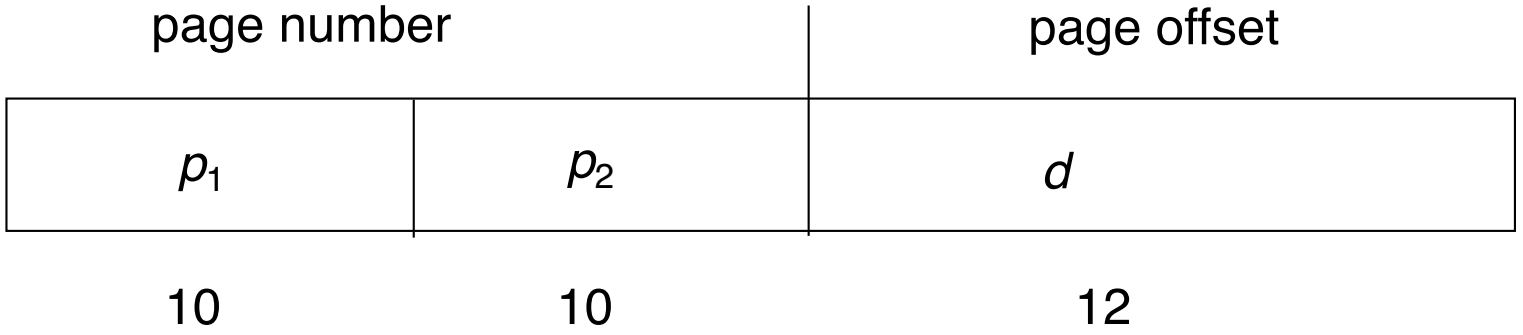
- 现代计算机系统都支持非常大的逻辑地址空间
- 在此情况下页表很大，显然不可能在内存中连续存放页表。
- 分级页表也称为多级页表，层次页表。

分级页表

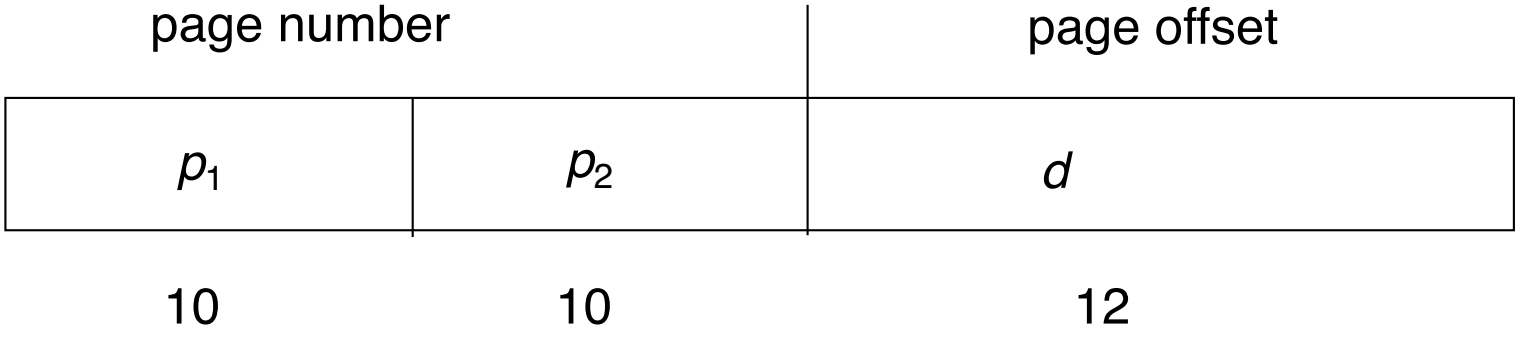
- 如具有32位逻辑地址空间的系统，页面大小4KB，则页表可以有1M项，若每个页表项占4字节，则页表共需要4MB内存空间。
-
- 解决方案：
  - 用离散方式存储页表
  - 仅将当前需要的部分页表项放在内存，其余放在磁盘上，需要时调入。

两级页表

- 将页表再分页，使每页与内存物理块大小相同，并为它们进行编号0、1、…，同时还为离散存放的页表建立一张页表。
- 例如：一个32位、4KB页面大小的逻辑地址可以划分为：

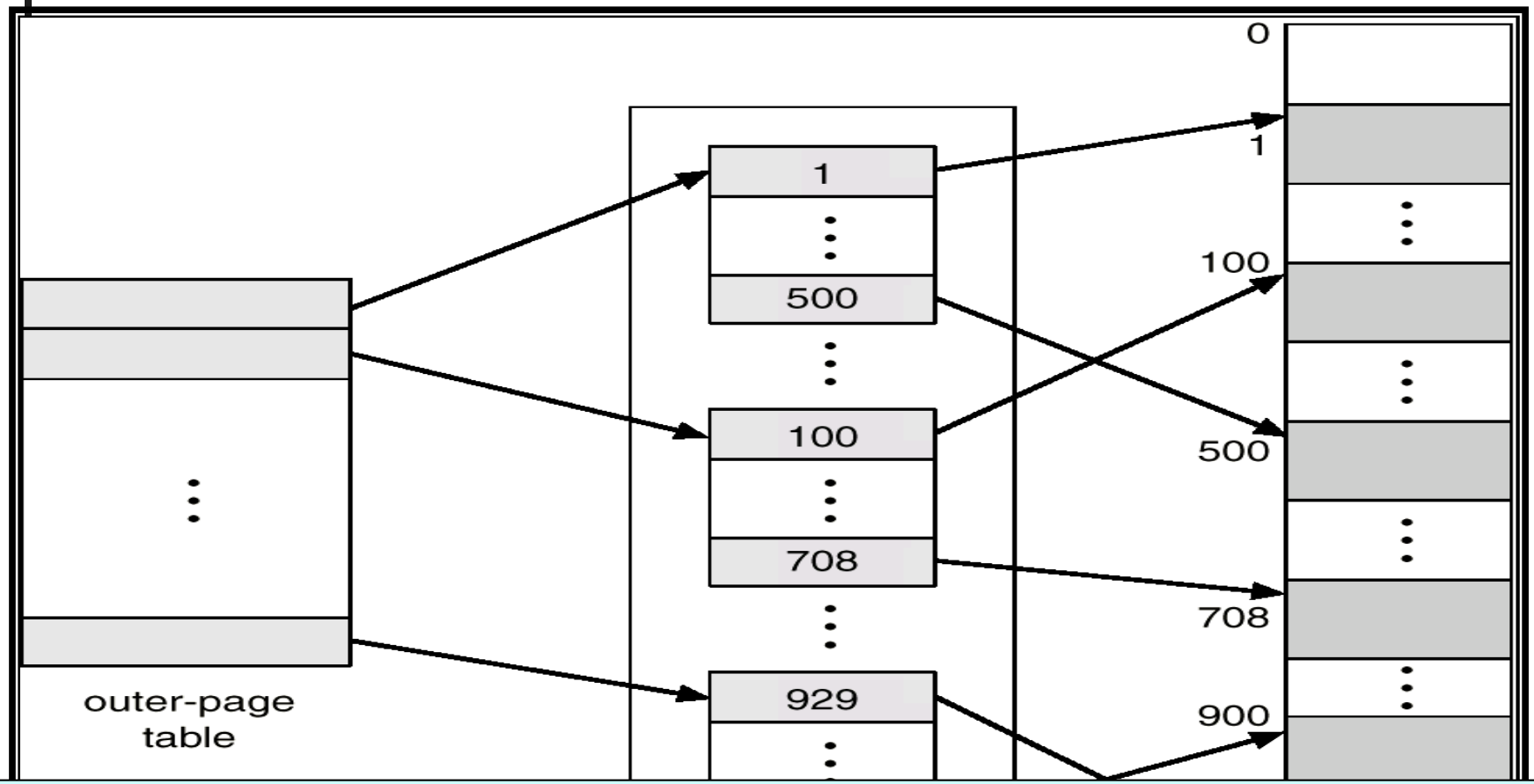


两级页表



- p1是访问外部页表的索引， p2是外部页表的偏移
- 也称p1是一级页号， p2是二级页号

两级页表结构

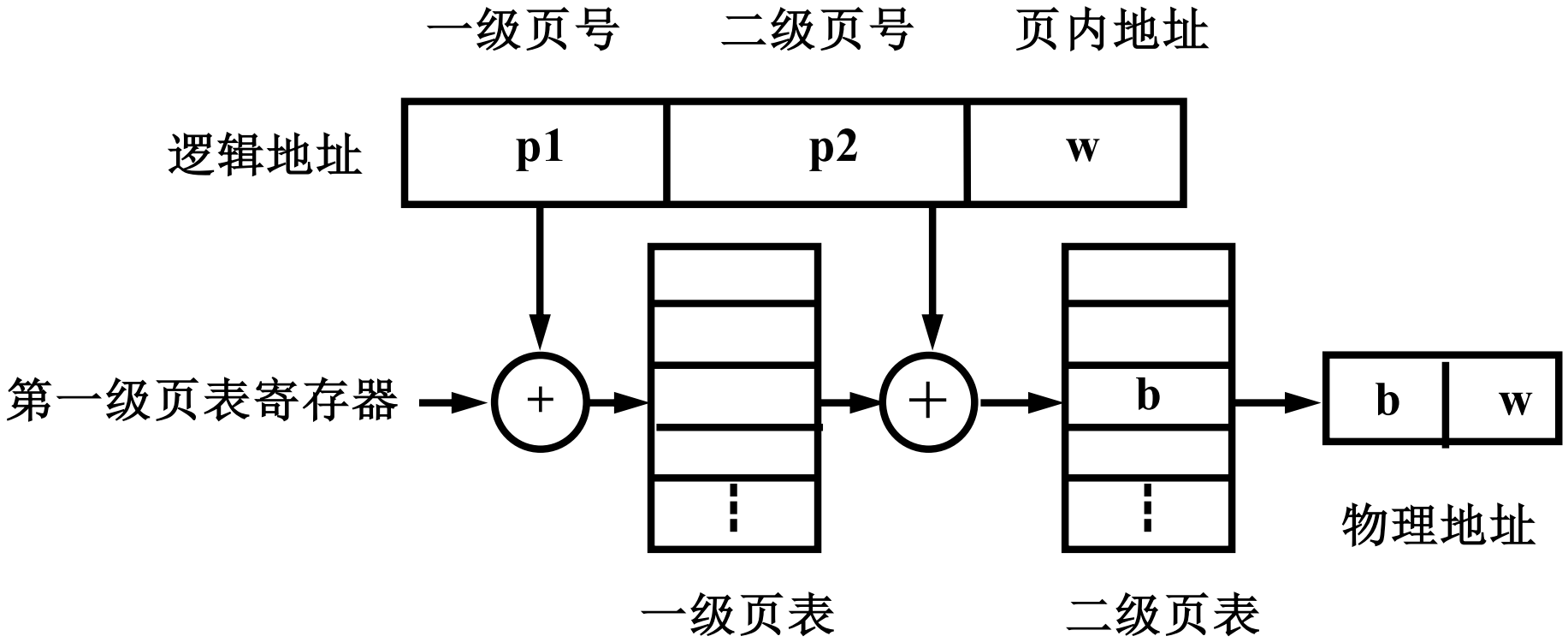


若页表项大小为**4B**，物理块大小为**4KB**，则两级页表结构可以存放下多少个页表项？

两级32位分页结构的地址转换机制

- 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。

两级32位分页结构的地址转换机制续



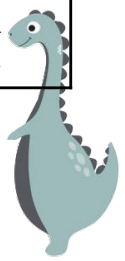
多级页表

- 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。
- 对于64位体系结构，分级页表通常并不合适

示例

- 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，则为了满足系统的分页管理至少应采用多少级页表？
- 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



示例

- 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。

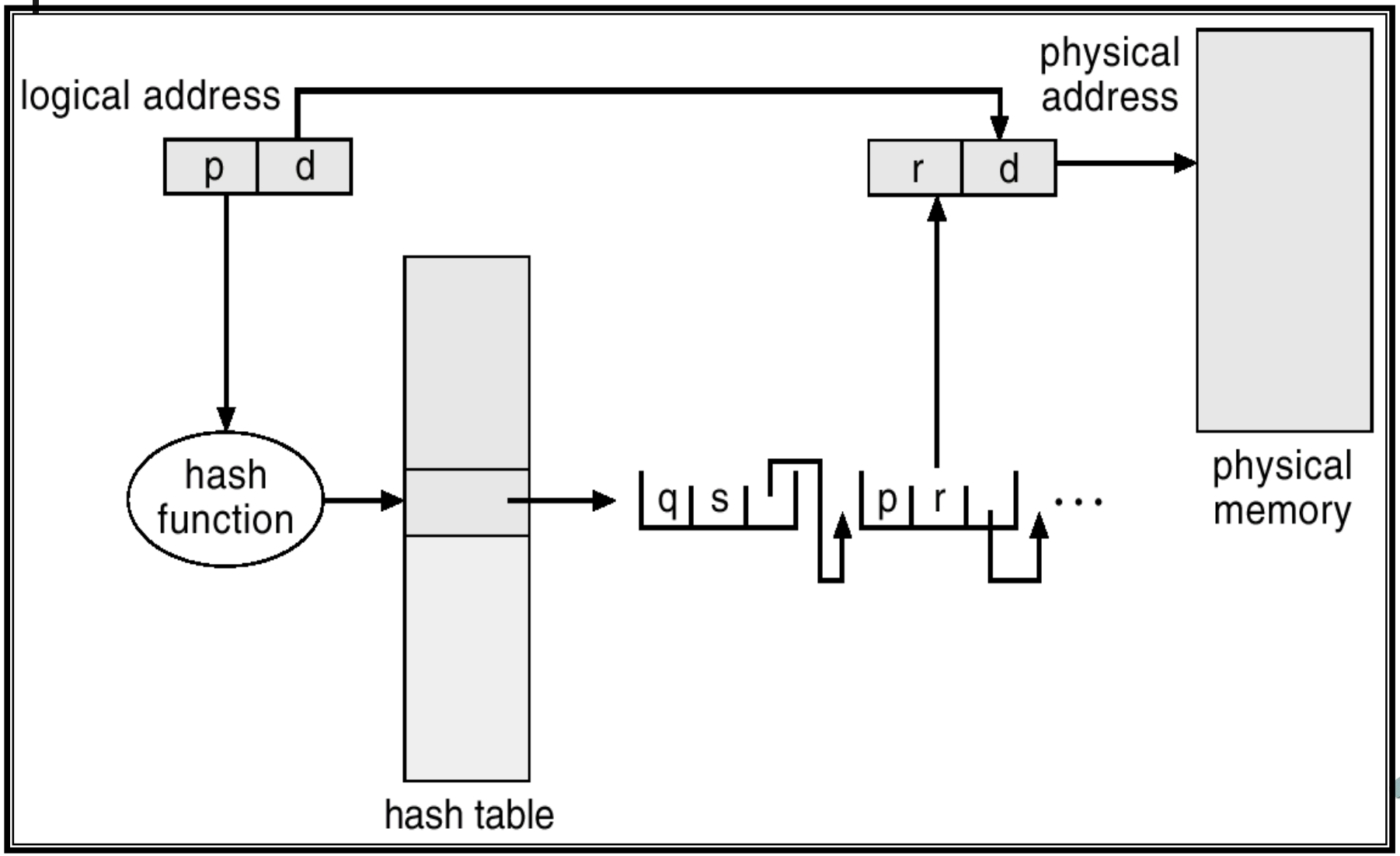
哈希页表

- 处理超过32位地址空间的常用方法是使用哈希页表
- 虚拟页号被散列到一个页表中，页表的每个表项包含散列到相同地址的链指针。
- 每个元素包含：虚拟页号、帧号、指针
-

哈希页表

- 算法按如下方式工作：用虚拟地址中的页号转换到哈希表中，用虚拟页号与链表中的每个元素的第一个域比较
- 如果匹配成功，相应的帧号用来形成物理地址，否则对下一节点进行比较以寻找匹配的页号。

Hashed Page Table



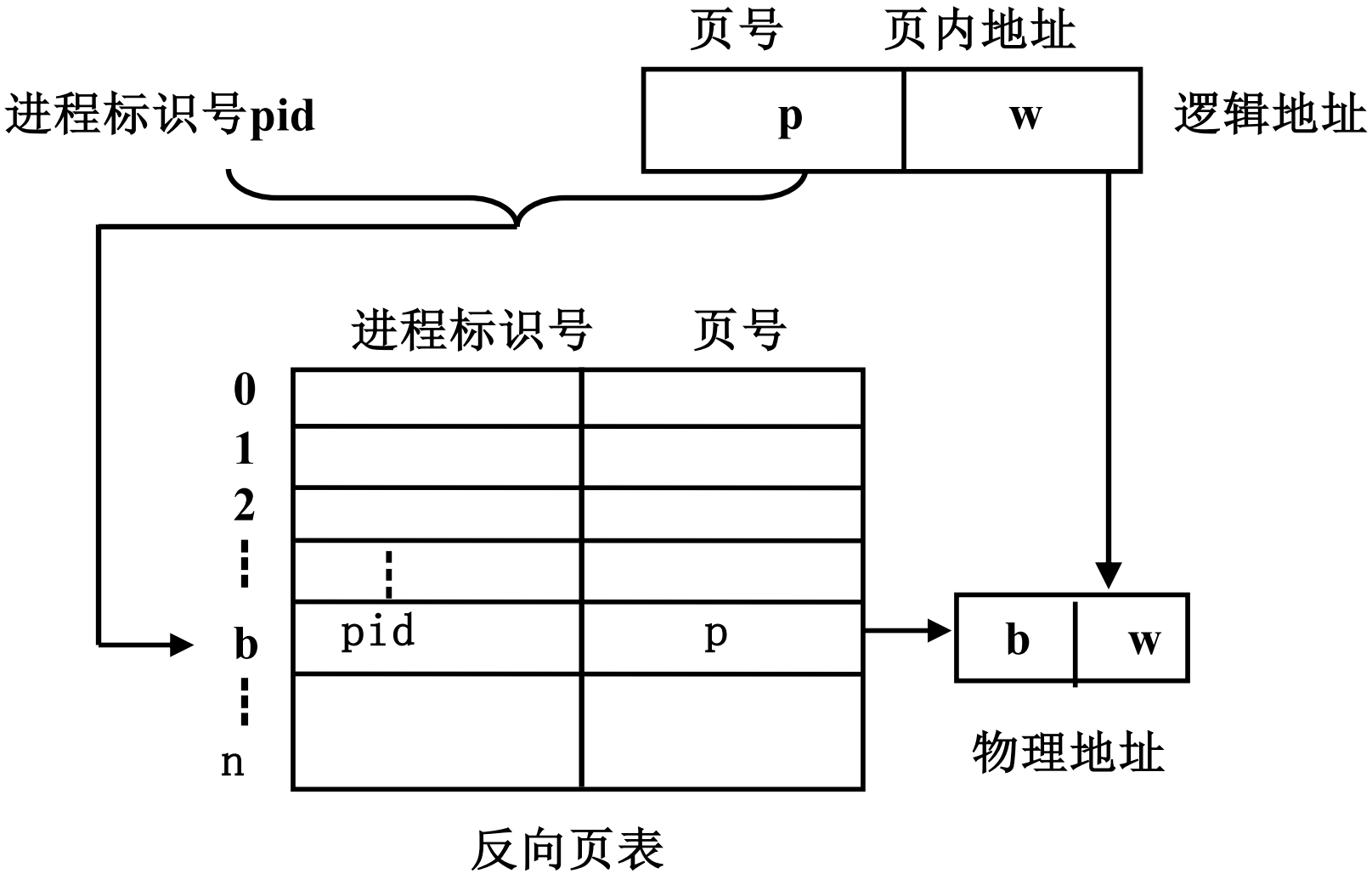
反向页表Inverted Page Table (IPT)

- 现代操作系统一般允许大逻辑地址空间，这使得页表太大，为解决页表占用大量存储空间的问题，引入了反向页表。
- 反向页表为每个物理块设置一个页表项，并将它们按物理块号大小排序，表项内容为页号及其隶属进程的标识号。

反向页表地址变换过程

- 利用进程标识号及页号检索反向页表，若找到相应的页表项，则将其物理块号与页内地址拼接；否则请求调入该进程相应页，在无调页功能的系统中则出错。
- 由于反向页表中没有存放进程中尚未调入页，因此必须为每个进程建立一张传统页表并存放在外存中，当所访问页不在内存时使用这张页表。页表中包含各页在外存的地址。

反向页表的地址变换



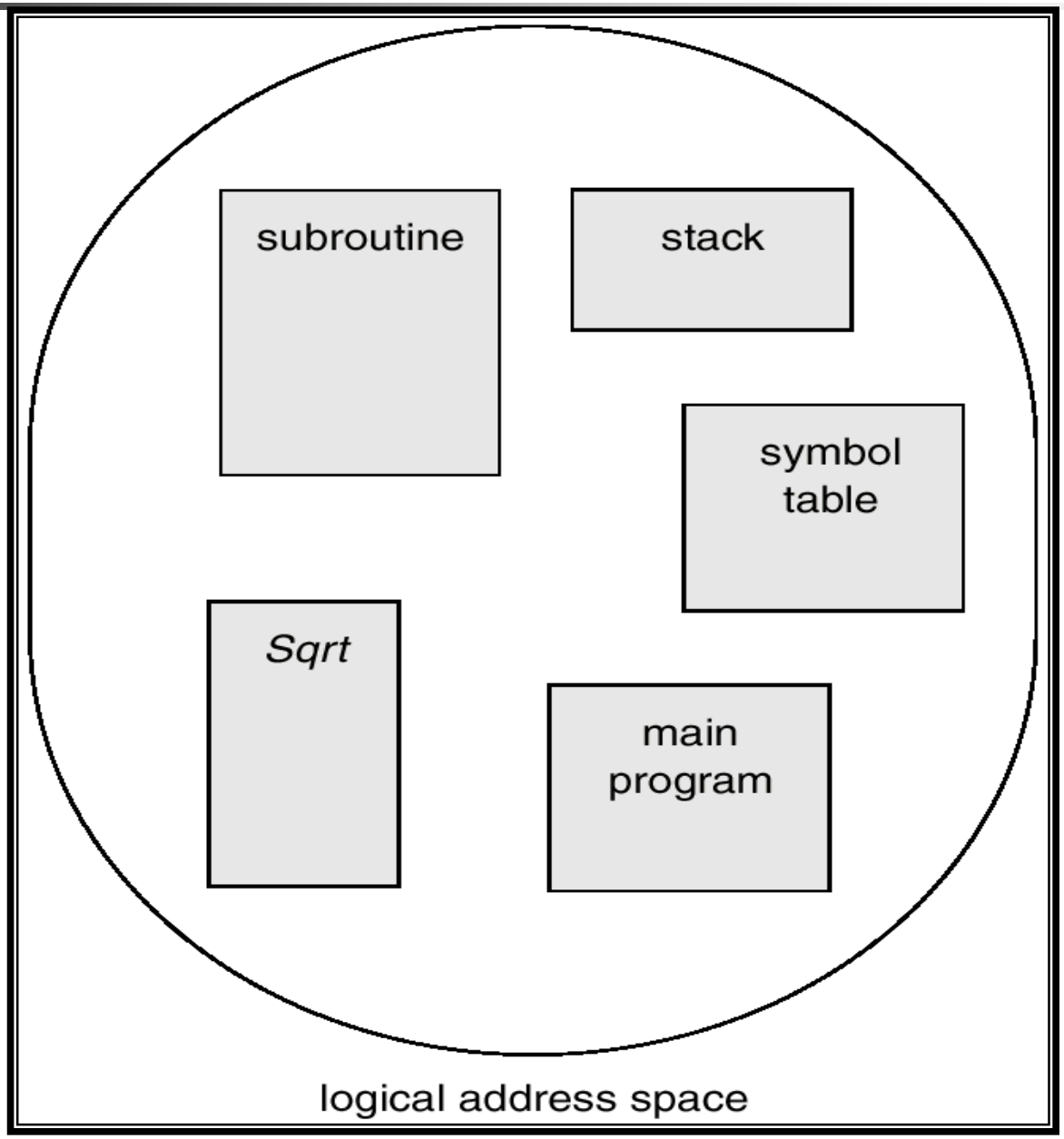
反向页表的不足

- 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
  - 用散列函数存放反向页表

方案4：段式管理

- 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。

User's View of a Program



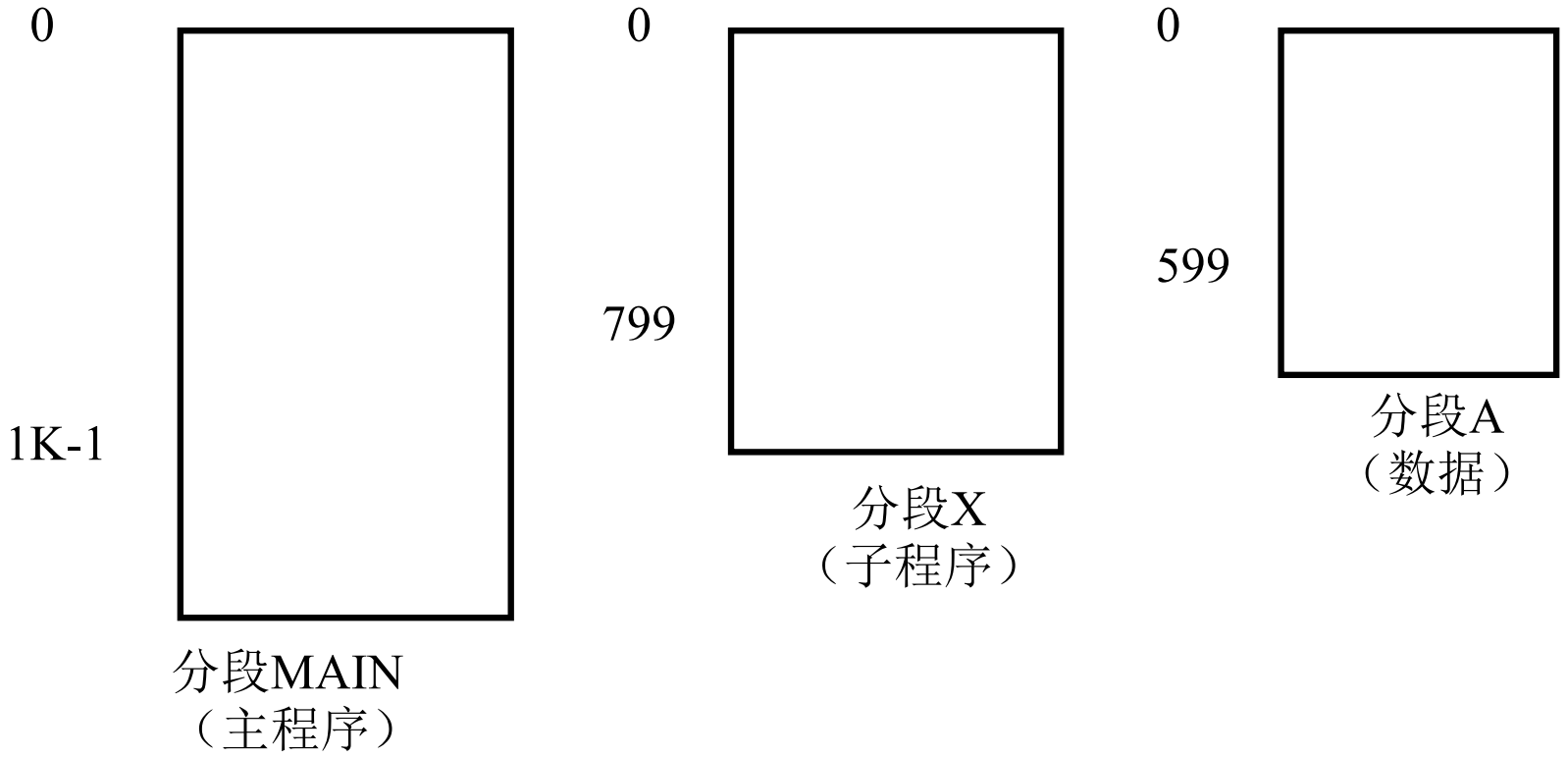
一个程序是一些段的集合，一个段是一个逻辑单位

分段管理的实现思想

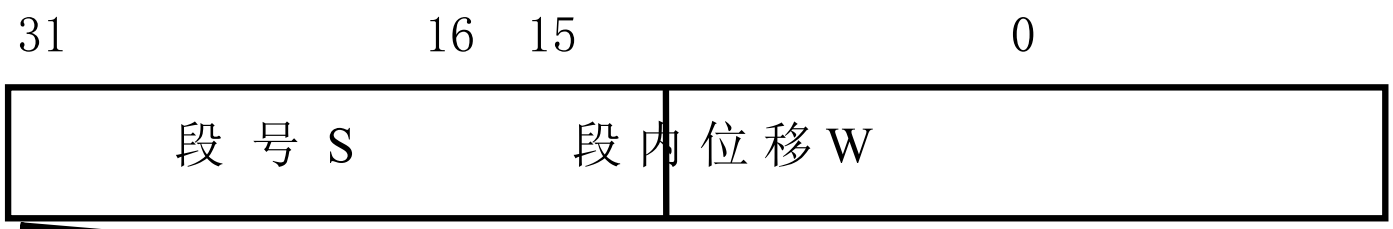
- 在分段存储管理系统中，作业的地址空间由若干个逻辑分段组成，每个分段是一组逻辑意义相对完整的信息集合，每个分段都有自己的名字，每个分段都从0开始编址并采用一段连续的地址空间。
- 在进行存储分配时，以段为单位分配内存，每段分配一个连续的内存区，但各段之间不要求连续。

作业的地址空间是二维的

- 作业的地址空间分为多段，每段都从0开始编址，故地址是二维的。



分段系统的逻辑地址结构



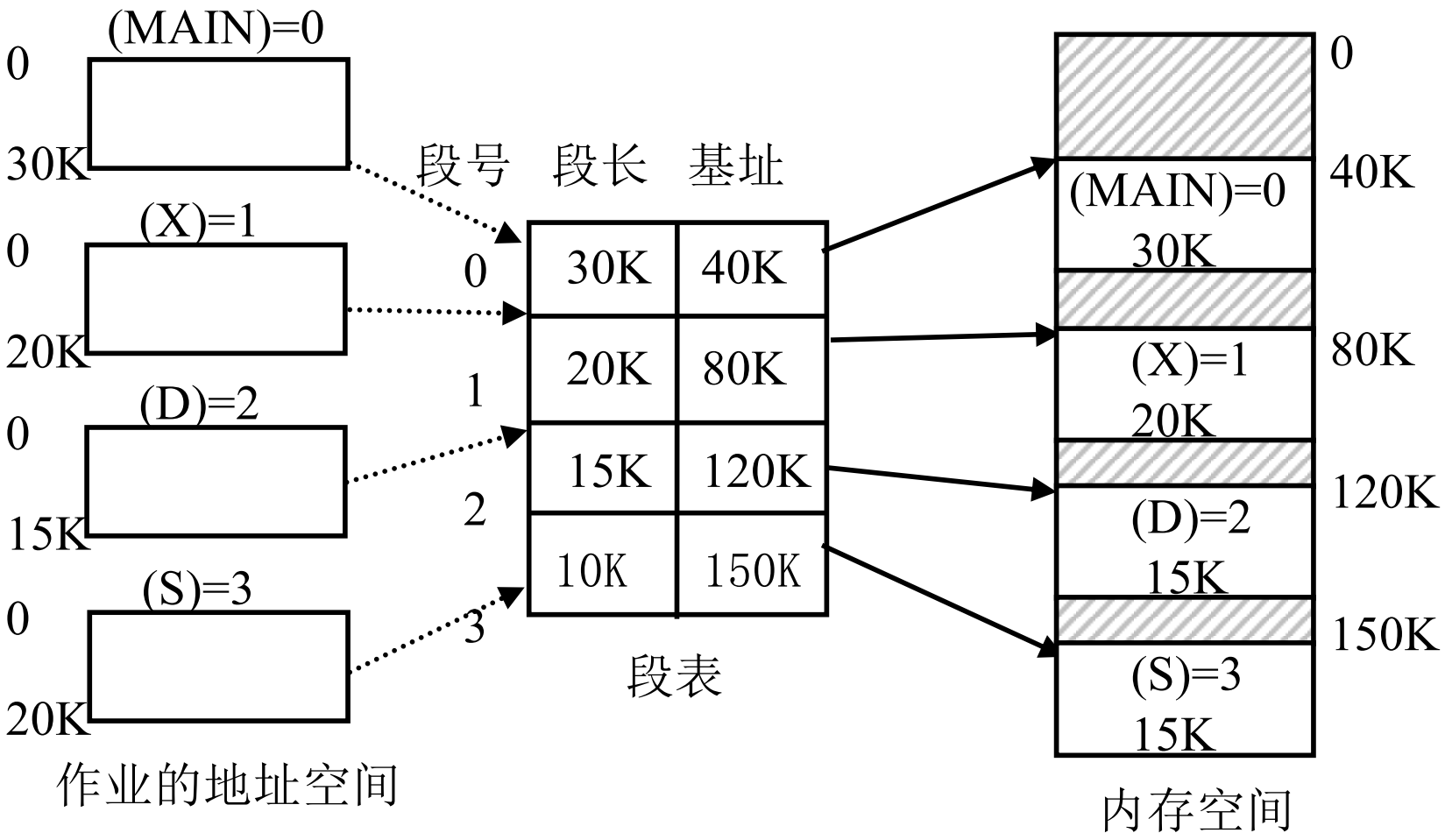
- 该地址结构最多允许多少分段?每段最大长度为多少?

- 该地址结构允许作业最多有64K个段，
- 每段的最大长度为64KB。

硬件

- 为了实现从逻辑地址到物理地址的变换，必须为每个进程建立一个段表，用来记录每段在内存的起始地址及相关信息。其中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
- 段表一般存放在内存。

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

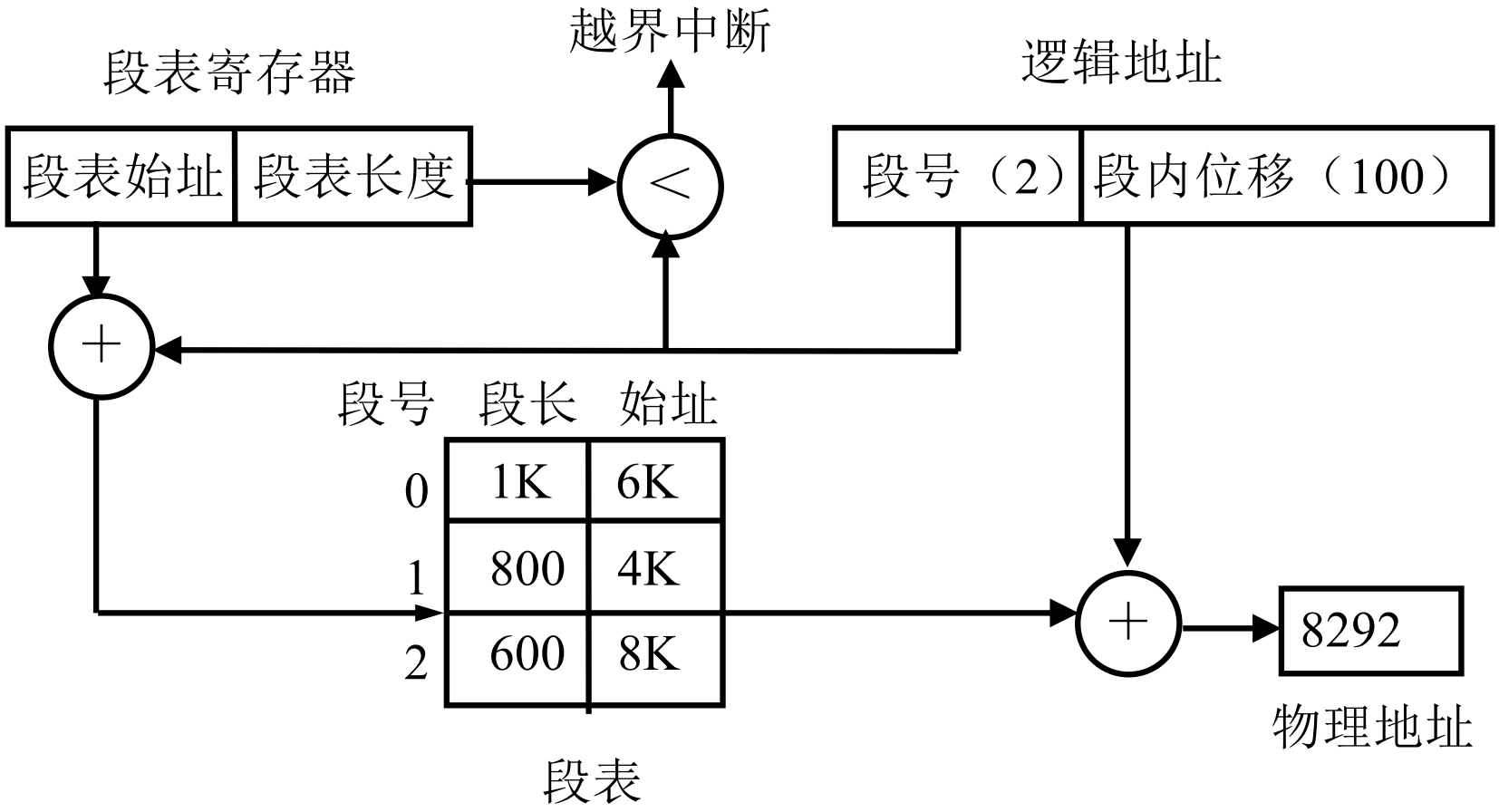
地址变换

- 为实现从逻辑地址到物理地址的转换，在系统中设置了段表寄存器，用于存放段表始址和段表长度。
- 为了提高内存的访问速度，也可以使用快表。

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
<
段号（2）
段内位移（100）
+
段号 段长 始址
0
1K
6K
1
800
4K
2
600
8K
+
8292
物理地址
段表

分段地址变换例

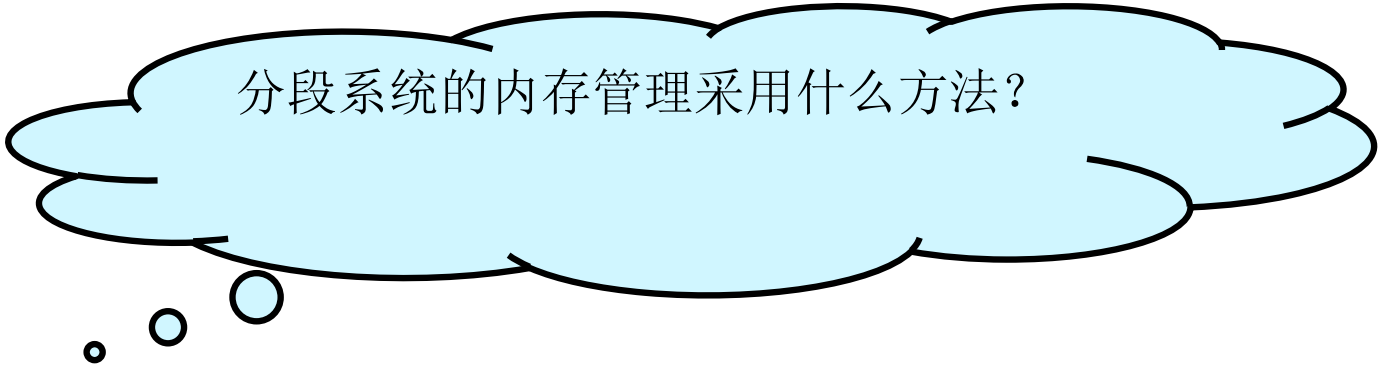
- 设作业分为3段，0、1、2段长度分别为1K、800、600，分别存放在内存6K、4K、8K开始的内存区域。
- 逻辑地址（2，100）的段号为2，段内位移为100。
- 查段表可知第2段在内内存的起始地址8K。
- 将起始地址与段内位移相加，8K＋100＝8292，物理地址为8292。

分段与分页的主要区别

- 分页管理与分段管理有许多相似之处，但两者在概念上也有很多区别，主要表现在：
  - 页是信息的物理单位，是为了减少内存碎片及提高内存利用率，是系统管理的需要。段是信息的逻辑单位，它含有一组意义相对完整的信息，分段的目的是为了更好地满足用户的需要。
  - 页的大小固定且由系统决定，由硬件把逻辑地址划分为页号和页内地址两部分。段的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。
  - 分页系统中作业的地址空间是一维的，分段系统中作业的地址空间是二维的。

分段保护

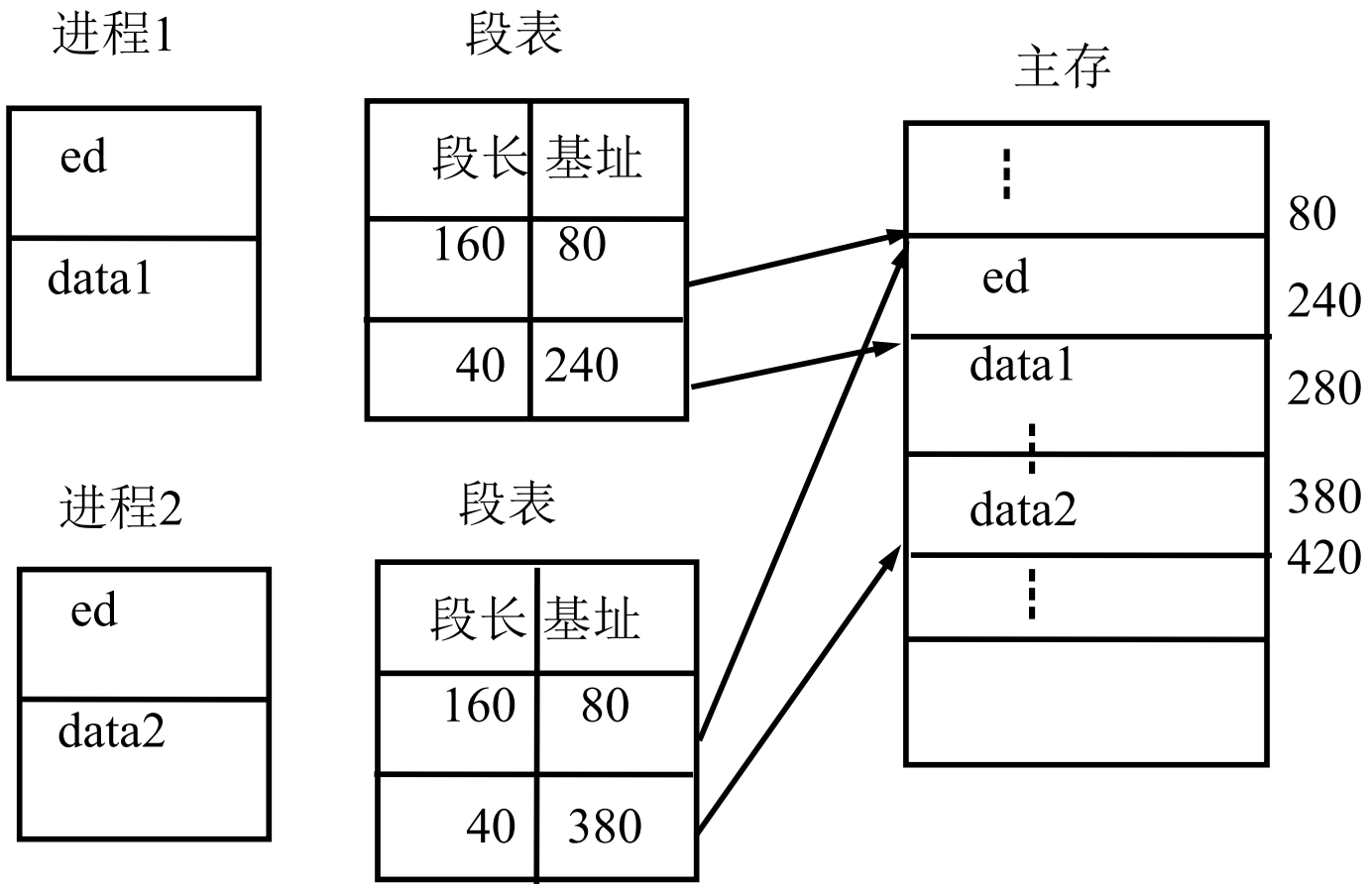
- 分段保护方法有：
  - 地址越界保护：段号与段表长度的比较，段内位移与段长的比较
  - 存取控制保护：设置存取权限，访问段时判断访问类型与存取权限是否相符



分段系统中的信息共享

- 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



方案5：段页式管理

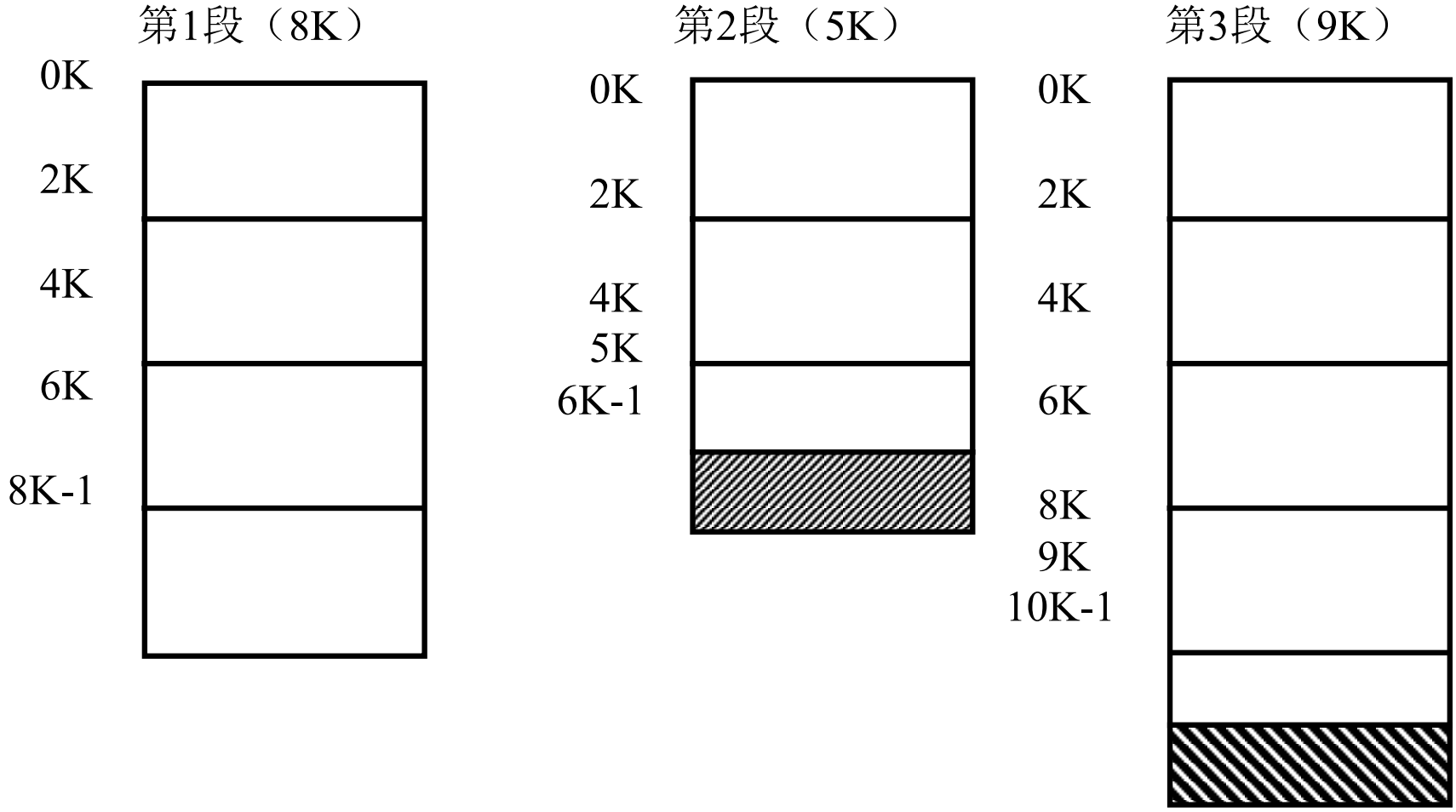
- 分页系统能有效地提高内存利用率，而分段系统能很好地反映用户要求。如果将这两种存储管理方式结合起来，就形成了段页式存储管理系统。

段页式存储管理的基本思想

- 在段页式存储管理系统中，作业的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。
- 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。



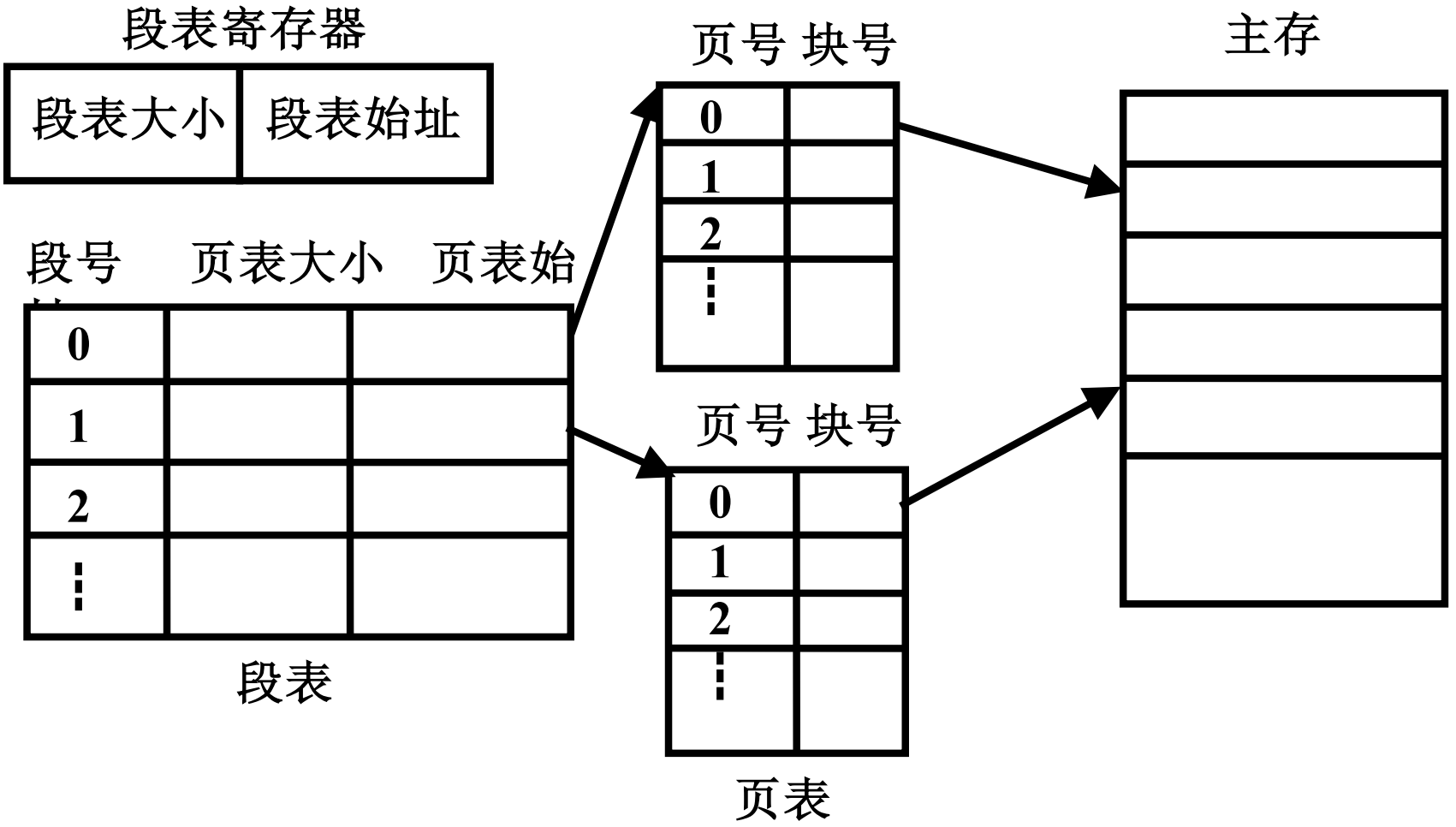
作业的逻辑地址结构

- 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。
- 此外，为了便于实现地址变换，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。

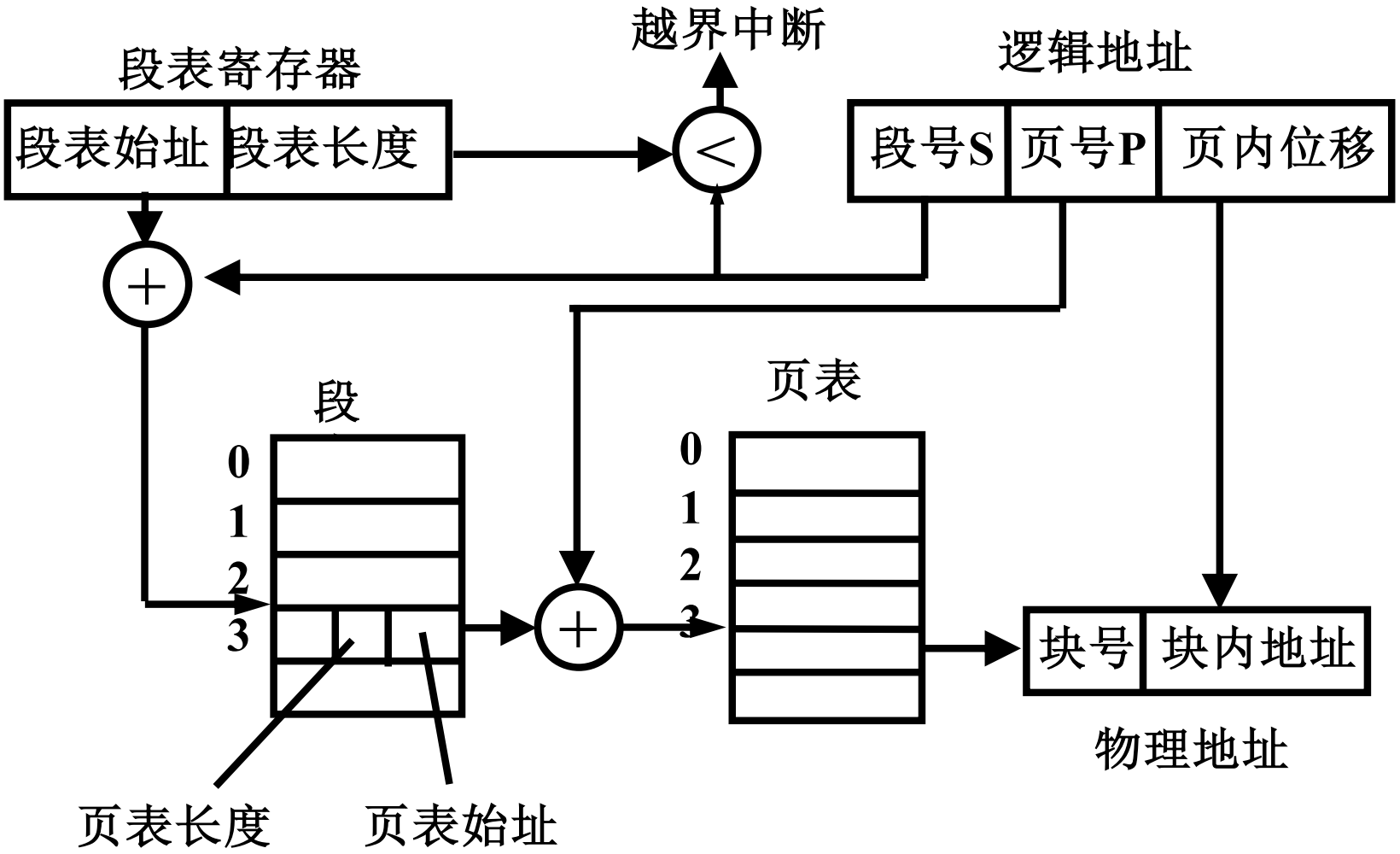
段表、页表及段表寄存器



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机构



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。

虚拟内存技术

- 常规存储器管理方式要求作业运行前全部装入内存，作业装入内存后一直驻留内存直至运行结束。
- 这种存储管理方式限制了大作业的运行。而物理扩充内存会增加成本，故应从逻辑上扩充内存。
- 虚拟内存技术允许执行进程不必完全在内存中。

背景

- **虚拟内存**：将用户逻辑内存与物理内存分开
  - 只有部分运行的程序需要在内存中
  - 逻辑地址空间能够比物理地址空间大
  - 允许若干个进程共享地址空间
  - 允许更多有效进程创建
- 虚拟存储器的理论基础是程序执行时的局部性原理
- **局部性原理**是指程序在执行过程中一个较短时间内，程序所执行的指令地址和操作数地址分别局限于一定区域内。
- 例如：
  - 除转移和过程调用外，程序主要是顺序执行。
  - 过程调用使程序从一部分区域转至另一部分区域
  - 循环结构

局部性的体现

- 局部性体现为：
  - 时间局部性：一条指令的一次执行和下次执行，一个数据的一次访问和下次访问，都集中在一个较短时间内。
  - 空间局部性：当前执行的指令和将要执行的指令，当前访问的数据和将要访问的数据，都集中在一个较小范围内。

虚拟存储器的基本原理

- 在程序运行之前，将程序的一部分放入内存后就启动程序执行。
- 在程序执行过程中，当所访问的信息不在内存时，由操作系统将所需要的部分调入内存，然后继续执行程序。
- 另一方面，操作系统将内存中暂时不使用的内容换出到外存上，从而腾出空间存放将要调入内存的信息。
- 从效果上看，这样的计算机系统好像为用户提供了一个存储容量比实际内存大得多的存储器，将这个存储器称为虚拟存储器。

虚拟存储器

- 虚拟存储器是指具有请求调入和置换功能，能从逻辑上对内存容量加以扩充的一种存储器系统。
- 虚拟存储器是一种以时间换空间的技术。

虚拟存储器的特征

- 离散性：不连续内存分配
- 多次性：一个作业分多次装入内存
- 对换性：允许运行中换进换出
- 虚拟性：逻辑上扩充内存

虚拟存储器的本质

- 虚拟存储器的本质是将程序的访问地址和内存的可用地址分离，为用户提供一个大于实际主存的虚拟存储器。
- 虚拟存储器的容量受限于：
  - 地址结构
  - 外存容量

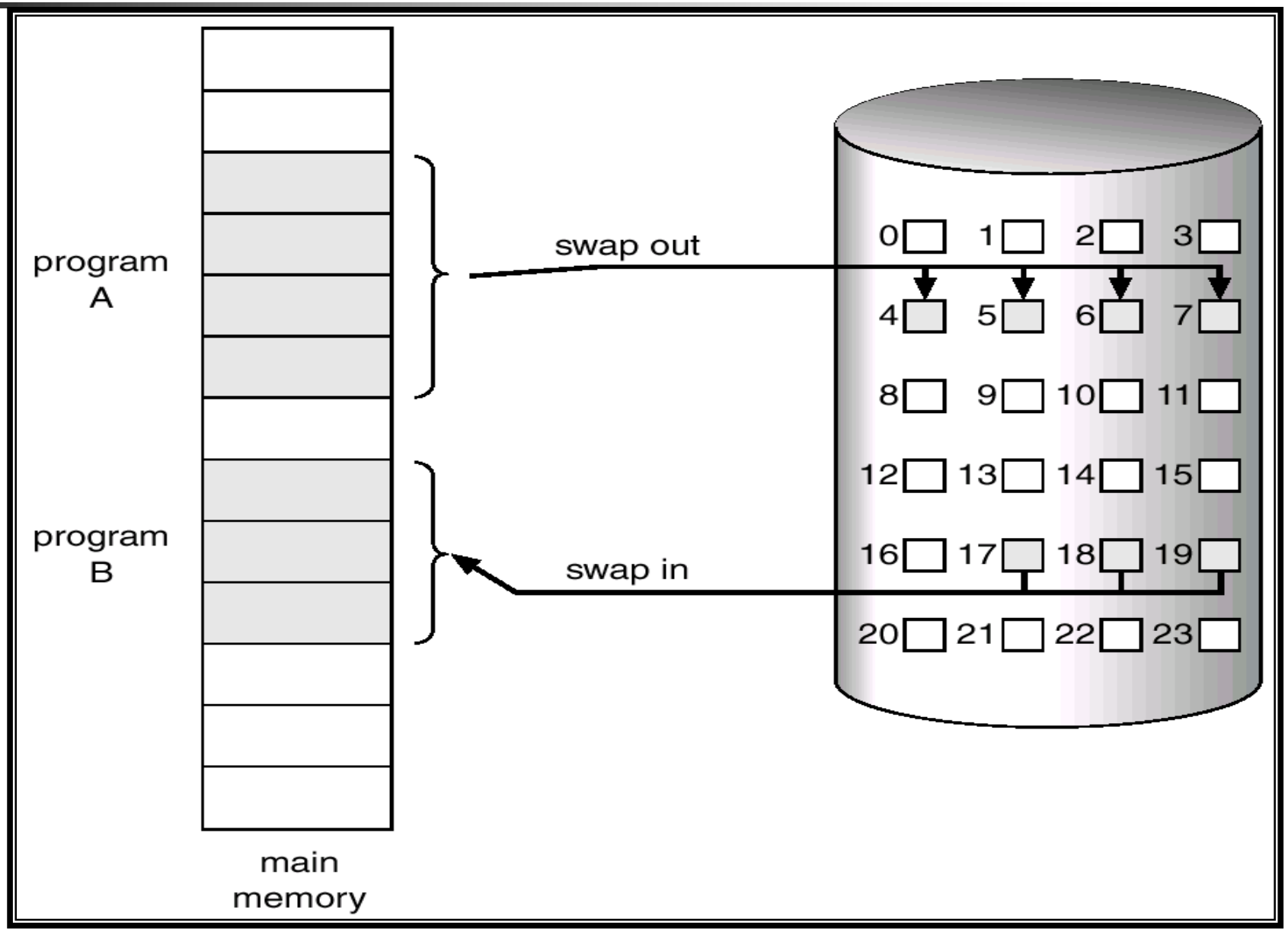
实现虚拟存储技术的物质基础

- 相当数量的外存：足以存放多个用户的程序。
- 一定容量的内存：在处理机上运行的程序必须有一部分信息存放在内存中。
- 地址变换机构：动态实现逻辑地址到物理地址的变换。

虚拟存储器的实现

- 虚拟内存通过以下方式实现：
  - 请求分页存储管理 Demand paging
    - 按需调页：只有在需要的时候才调入一个页
  - 请求分段存储管理 Demand segmentation
  - 请求调页的段页式存储管理

分页内存到连续磁盘空间的传送



请求分页存储管理实现思想

- 请求分页存储管理方法是在分页存储管理的基础上增加了请求调页和页面置换功能。
- 实现思想：在作业运行之前只装入当前需要的一部分页面便启动作业运行。在作业运行过程中，若发现所要访问的页面不在内存，便由硬件产生缺页中断，请求OS将缺页调入内存。若内存无空闲存储空间，则根据某种置换算法淘汰已在内存的某个页面，以腾出内存空间装入缺页。

请求分页系统中的支持机构

- 请求分页中的支持机构有：
  - 页表
  - 缺页中断机构
  - 地址变换机构
  - 请求调页和页面置换软件

页表

- 请求分页系统中使用的主要数据结构仍然是页表。
- 但由于每次只将作业的一部分调入内存，还有一部分内容存放在磁盘上，故需要在页表中增加若干项。
- 扩充后的页表项如下所示：

扩充后的页表项

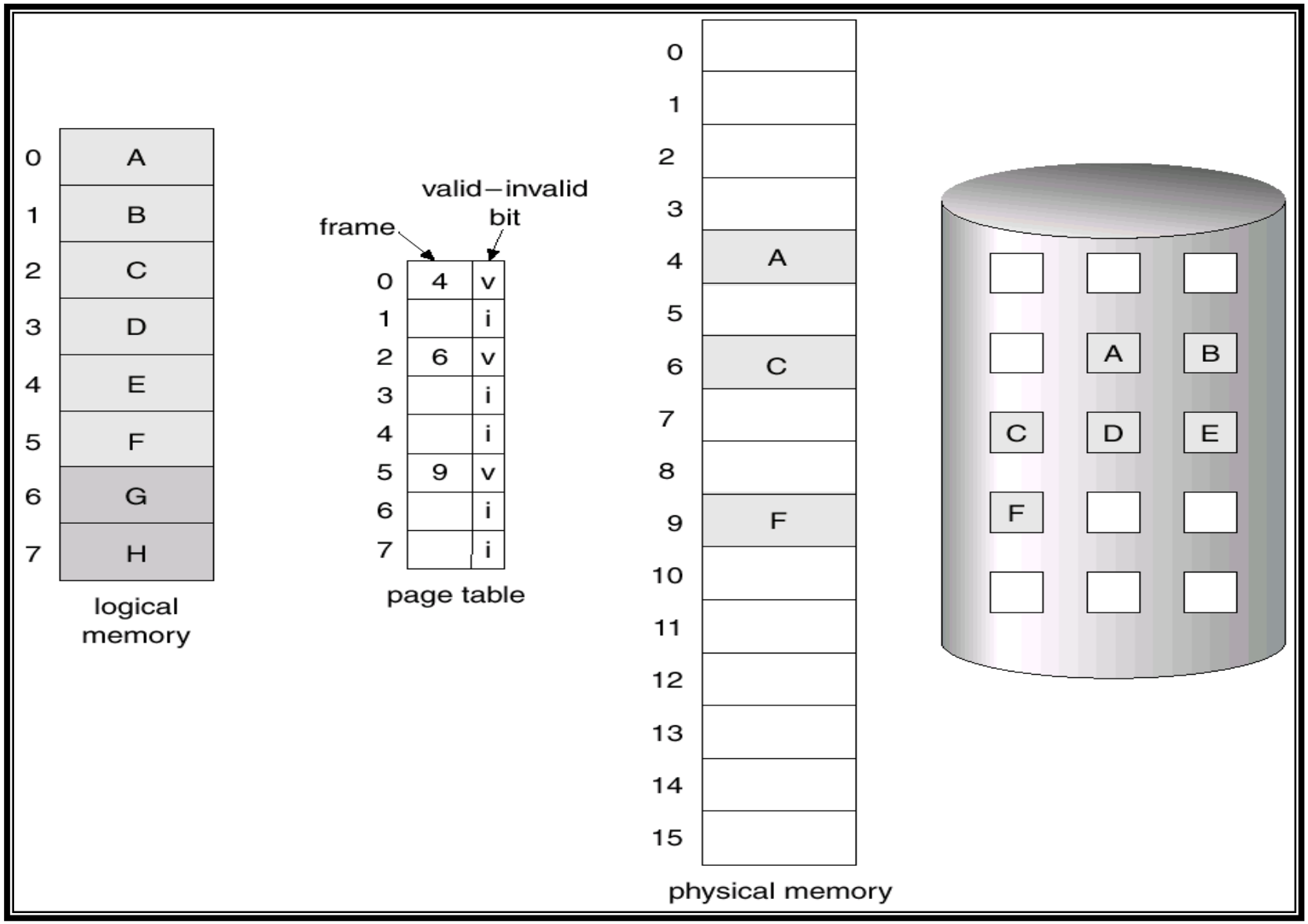
页号	物理块号	存在位	访问字段	修改位	外存地址

- 页号和物理块号：其定义同分页存储管理。
- 状态位：用于表示该页是否在主存中。
- 访问字段：用于记录本页在一段时间内被访问的次数，或最近已有多长时间未被访问。
- 修改位：用于表示该页调入内存后是否被修改过。
- 外存地址：用于指出该页在外存上的地址。

有效-无效位 Valid-Invalid Bit

- 可以使用有效-无效位来区分哪些页是在内存，哪些页在磁盘上
- 每个页表项关联一个有效- 无效位
  - 有效表示页面在内存且合法
  - 无效表示页面不合法或在外存

某些页不在内存时的页表



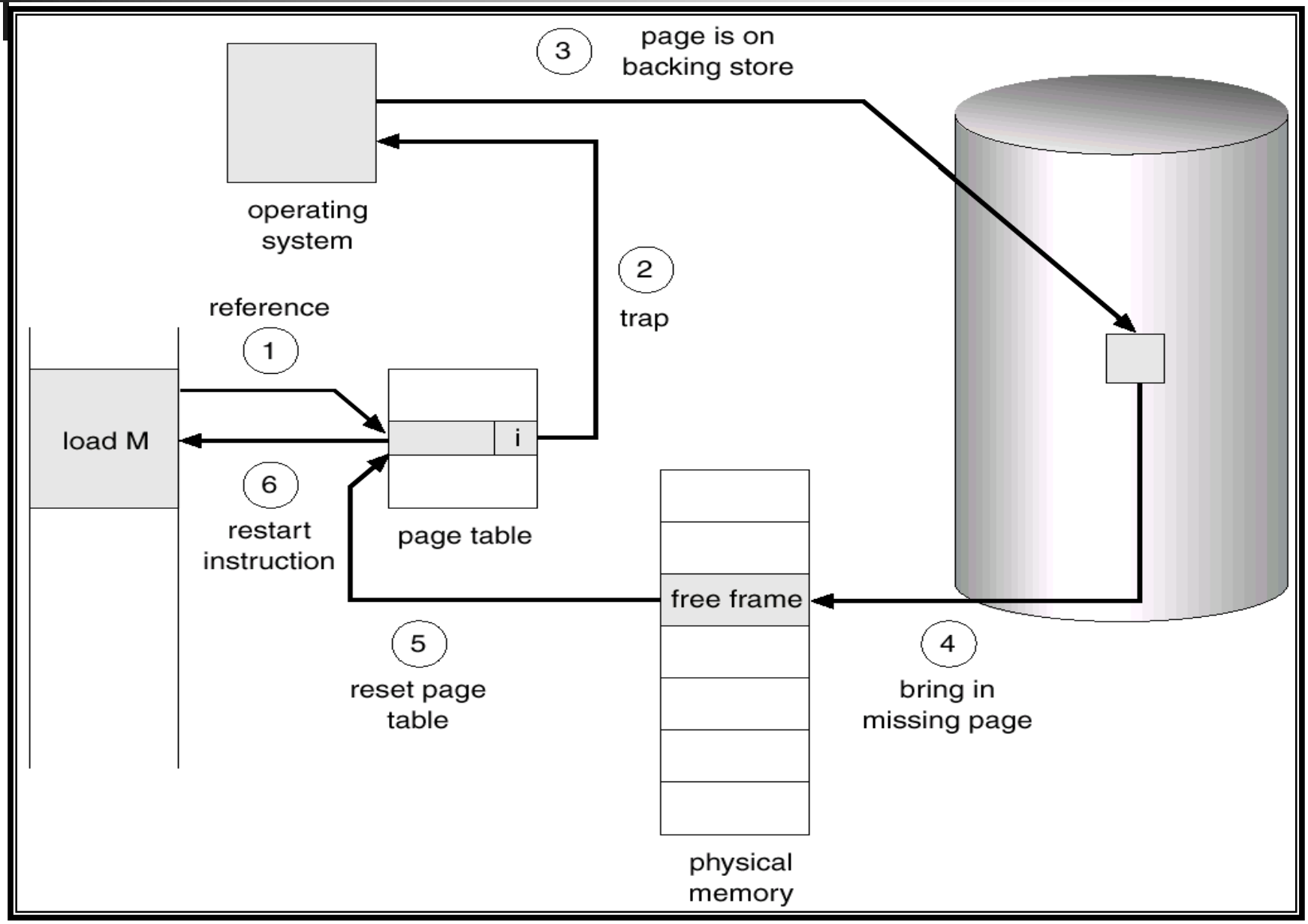
页错误 Page Fault

- 当访问无效页时，会产生页错误陷阱.
- 分页硬件在通过页表转换地址时，将发现已设置了无效位，会陷入操作系统。
- 即当所访问的页不在内存时，便产生缺页中断，请求OS将缺页调入内存。

页错误处理步骤

- 检查页表以确定引用是否合法
- 引用非法则终止，有效但不在内存则调入
- 找到一个空闲帧
- 把页换入页框
- 修改页表，使其有效.
- 重启指令

页错误处理步骤

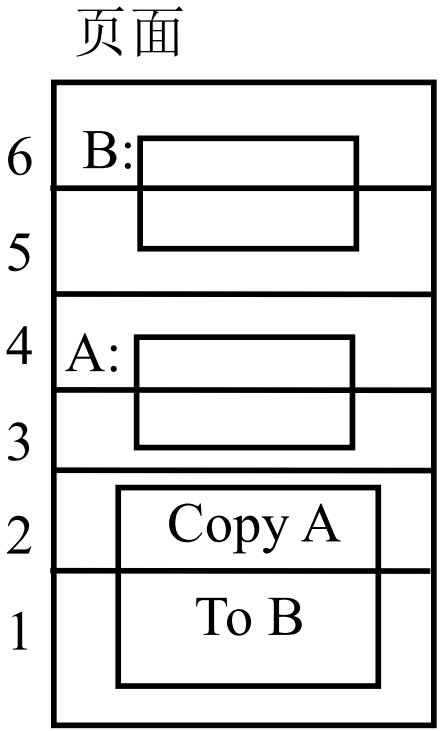

3
page is on
backing store
operating
system
2
trap
reference
1
load M
i
6
restart
instruction
page table
free frame
5
reset page
table
4
bring in
missing page
physical
memory

缺页中断

- 缺页中断处理程序根据该页在外存的地址把它调入内存。
- 在调页过程中，若内存有空闲空间，则缺页中断处理程序只需把缺页装入并修改页表中的相应项；
- 若内存中无空闲物理块，则需要先淘汰内存中的某些页，若淘汰页曾被修改过，则还要将其写回外存。

缺页中断与一般中断的区别

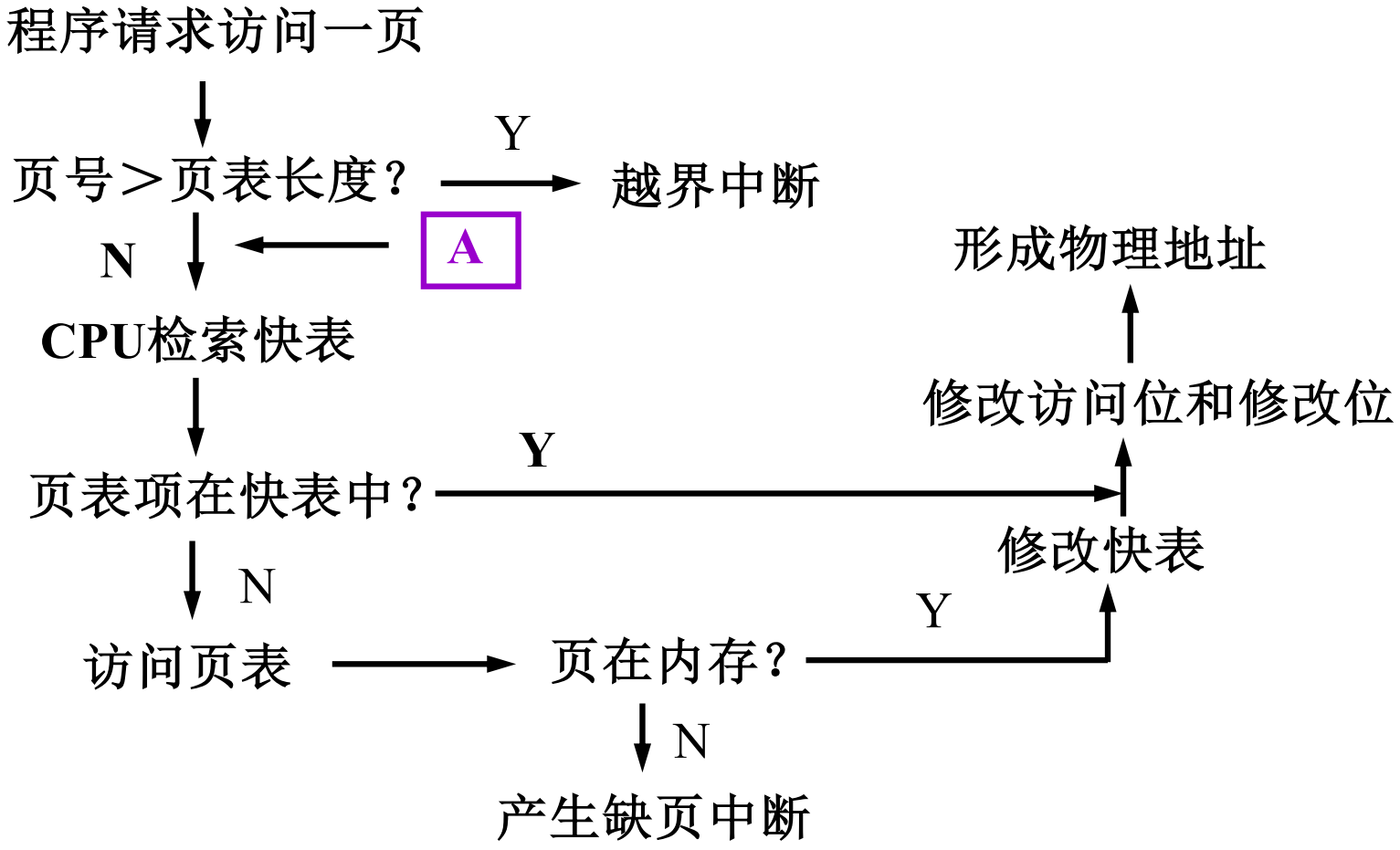
- 缺页中断与一般中断的区别主要有：
  - 在指令的执行期间产生和处理缺页中断。
  - 一条指令可以产生多个缺页中断。如：执行一条复制指令copy A to B
  - 缺页中断返回时执行产生中断的指令，一般中断返回时执行下条指令



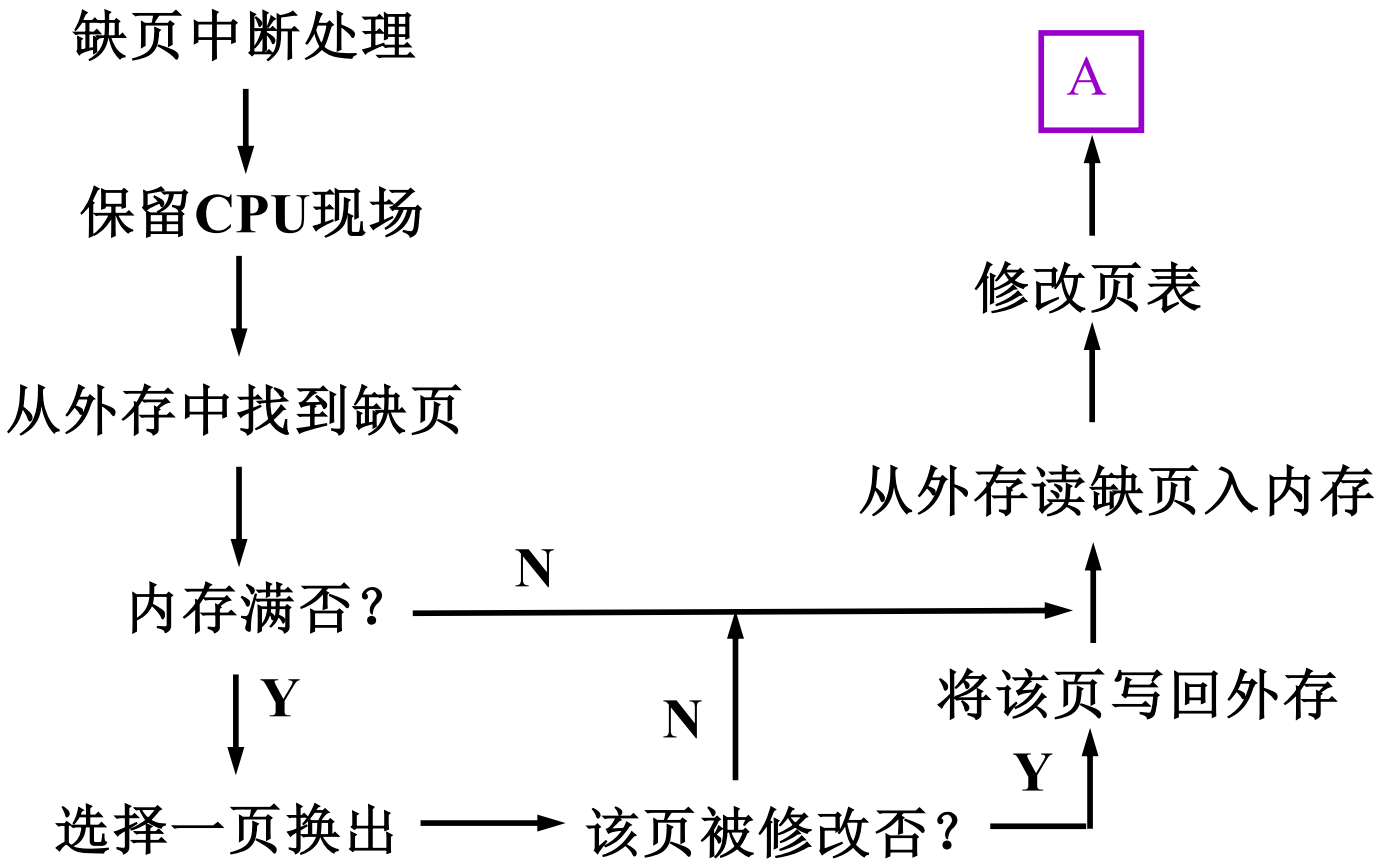
地址变换

- 请求分页存储管理系统的地址变换过程类似于分页存储管理，但当被访问页不在内存时应进行缺页中断处理。

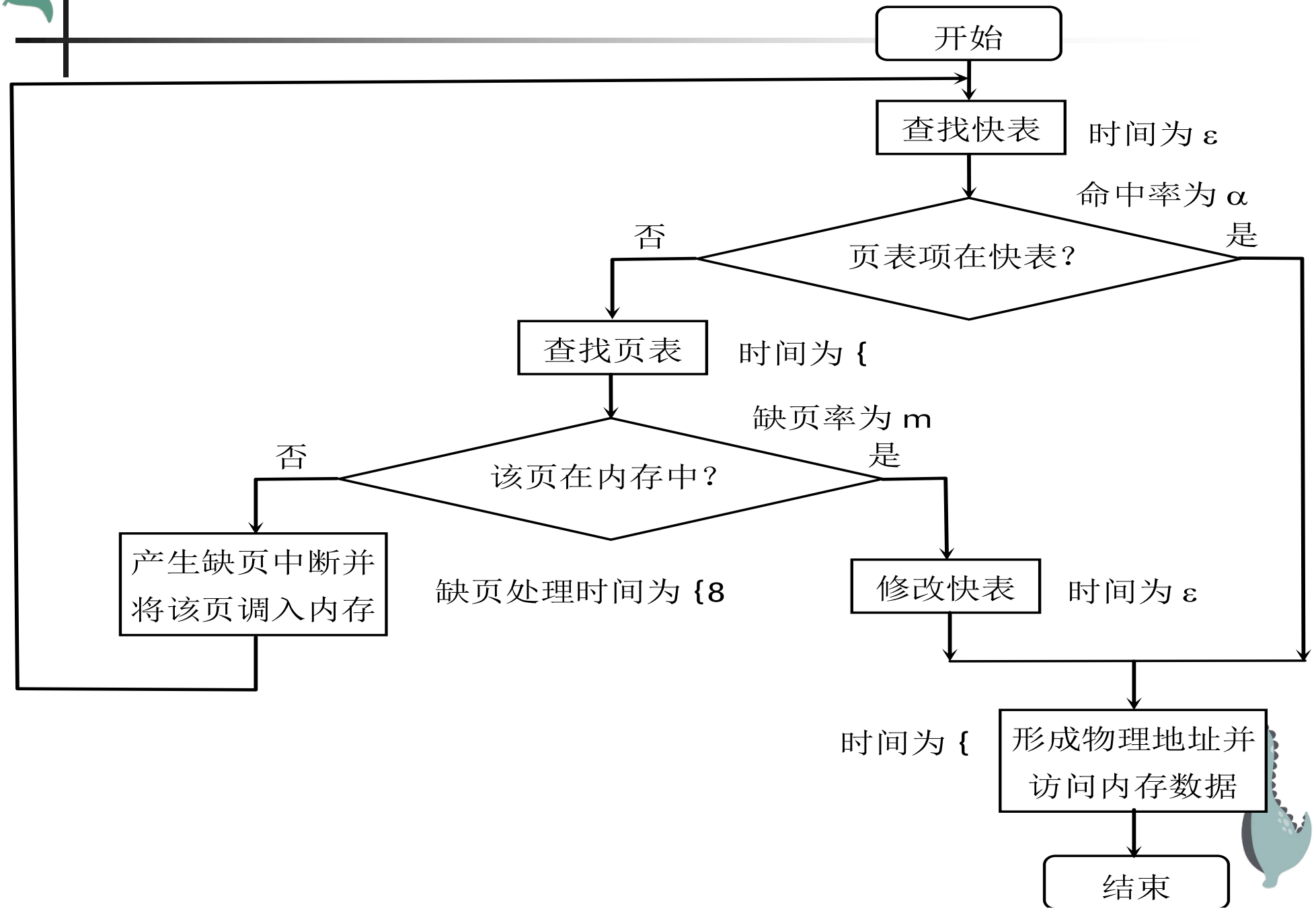
地址变换流程


程序请求访问一页
页号＞页表长度？
Y
越界中断
N
A
CPU检索快表
页表项在快表中？
Y
N
访问页表
页在内存？
Y
N
修改快表
修改访问位和修改位
形成物理地址
产生缺页中断

缺页处理流程


缺页中断处理
保留CPU现场
从外存中找到缺页
内存满否？
N
Y
选择一页换出
该页被修改否？
N
Y
将该页写回外存
从外存读缺页入内存
修改页表
A

请求页式系统的内存访问流程



访内存操作的三种情况

- 页在主存中且页表项在快表中：

  访问时间=查快表时间+访问内存时间=ε+t。

- 页在主存中且页表项不在快表中：

  访问时间=查快表时间+查页表时间+修改快表时间+访问内存时间=ε+t+ε+t=2(ε+t)

- 页不在主存中，设处理缺页中断的时间为t1（包含读入缺页、页表更新、快表更新时间）：

  访问时间=查快表时间+查页表时间+处理缺页中断时间t1+查快表时间+访问内存时间
  =ε+t+t1+ε+t=t1+2(ε+t)

有效访问时间

- 内存的读写周期为t，缺页中断服务时间为t1（包含读入缺页、页表更新、快表更新时间），快表的命中率为α，缺页中断率为f，快表访问时间为ε，则有效存取时间可表示为：
  - EAT= α *(ε +t)+(1- α)* [(1-f) *2 (ε +t) +f*(t1+ 2 (ε+t))]

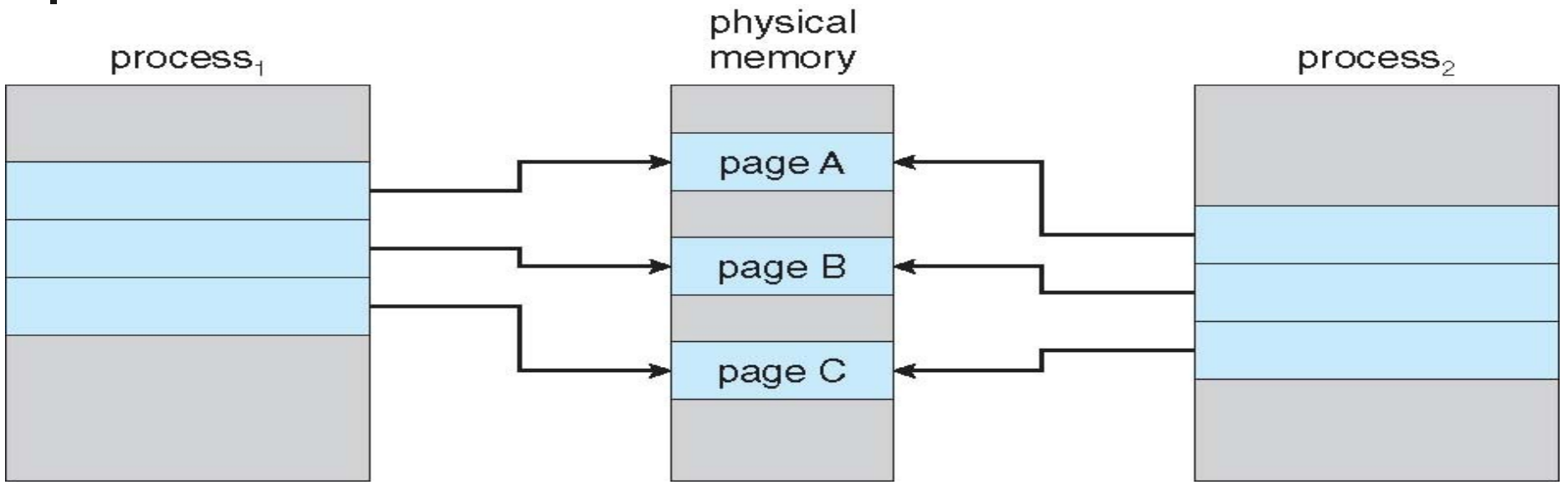
缺页中断处理时间

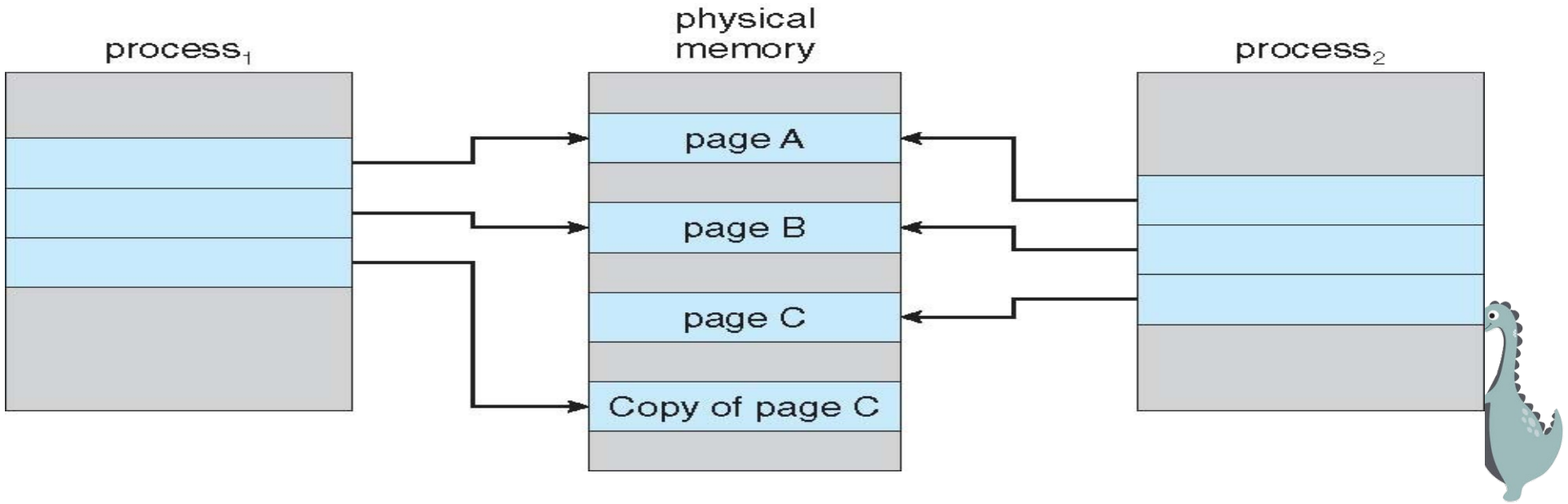
- 缺页中断处理时间由三部分组成：
  - 缺页中断服务时间
  - 页面传送时间：包括读缺页和写置换页的时间
  - 重新启动进程时间
- 由于缺页中断服务时间及重新启动进程时间较短，所以主要考虑页面传送时间。

写时拷贝

- 写时拷贝允许父子进程在内存中共享页面
- 如果任一进程需要对页面进行写，那么就创建一个共享页的副本
- 只有可能修改的页才需要标记为写时拷贝

进程1修改页面C前后

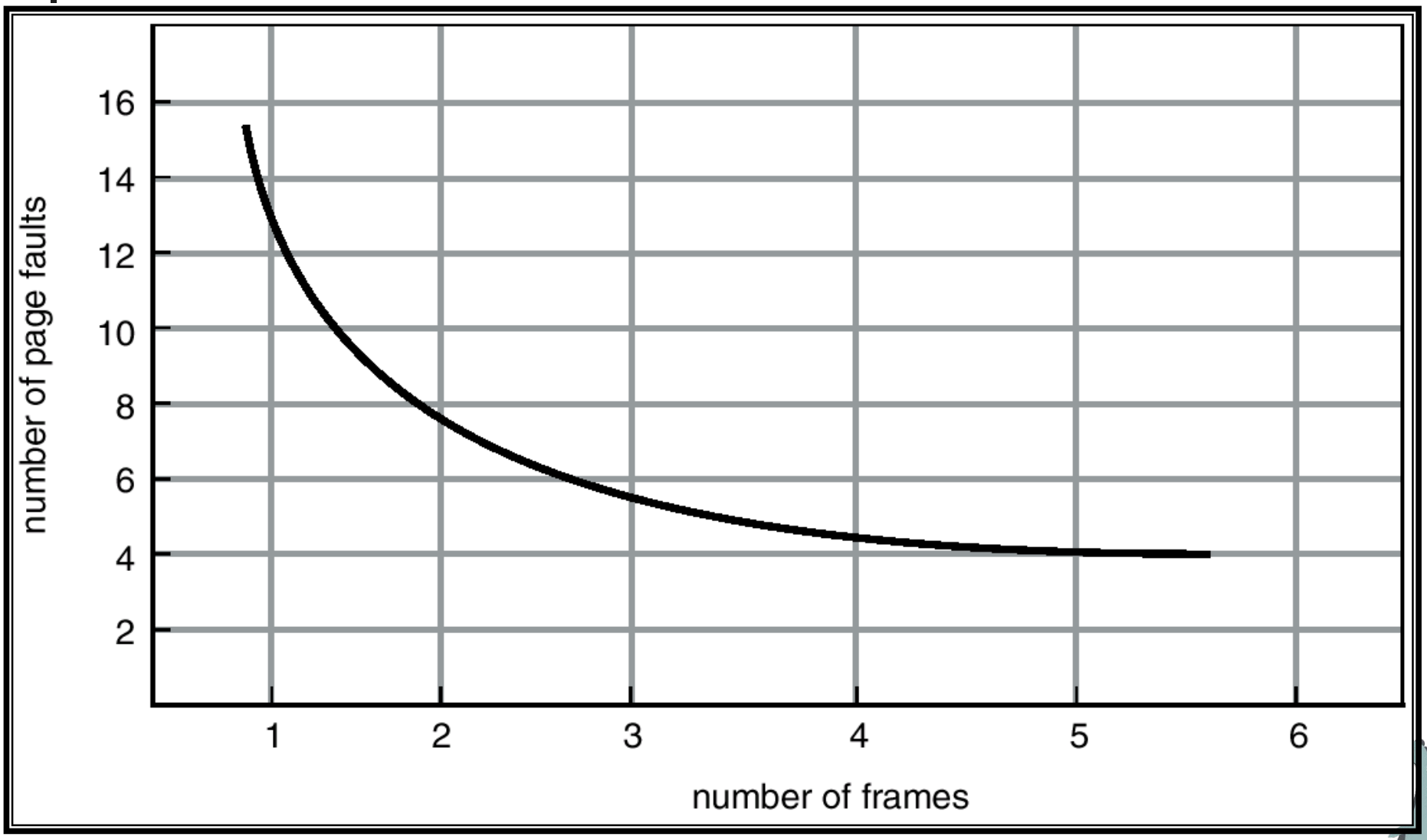

process1
physical memory
process2
page A
page B
page C


process1
physical memory
process2
page A
page B
page C
Copy of page C

页面置换

- 页面置换算法又称为页面淘汰算法，是用来选择换出页面的算法。
- 为了实现按需调页，要解决两个问题：
  - 帧分配算法：决定为每个进程分配多少帧
  - 页置换算法：选择要置换的页

页错误与帧数量图


number of page faults
16
14
12
10
8
6
4
2
1
2
3
4
5
6
number of frames

先进先出置换算法(FIFO)

- **先进先出算法**：选择调入主存时间最长的页面予以淘汰。
- 特点：该算法实现比较简单，对按线性顺序访问的程序比较合适，但可能产生异常现象。很少使用纯粹的FIFO。
- Belady异常：在某些情况下，分配给进程的页面数增多，缺页次数反而增加。

示例

- 引用串：

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- 3个帧开始为空
- 采用先进先出置换算法时的页面置换情况如下所示：

采用先进先出算法的页面置换情况

走向	1	2	3	4	1	2	5	1	2	3	4	5
块1	1	1	1	4	4	4	5			5	5	
块2		2	2	2	1	1	1			3	3	
块3			3	3	3	2	2			2	4	
	缺	缺	缺	缺	缺	缺	缺			缺	缺	

- 从上表中可以看出，共发生了9次缺页中断。其缺页率为9/12＝75%。

为进程分配4个物理块

走向	1	2	3	4	1	2	5	1	2	3	4	5
块1	1	1	1	1			5	5	5	5	4	4
块2		2	2	2			2	1	1	1	1	5
块3			3	3			3	3	2	2	2	2
块4				4			4	4	4	3	3	3
	缺	缺	缺	缺			缺	缺	缺	缺	缺	缺

- 从上表中可以看出，共发生了10次缺页中断。其缺页率为10/12＝83.3%。

无异常现象的页面序列

- 假定系统为某进程分配了3个物理块，页面访问序列为：7、0、1、2、0、3、0、4、2、3、0、3，开始时3个物理块均为空闲，采用先进先出置换算法时的页面置换情况如下所示：

页面置换情况

走向	7	0	1	2	0	3	0	4	2	3	0	3
块1	7	7	7	2		2	2	4	4	4	0	
块2		0	0	0		3	3	3	2	2	2	
块3			1	1		1	0	0	0	3	3	
	缺	缺	缺	缺		缺	缺	缺	缺	缺	缺	

- 从上表中可以看出，共发生了10次缺页中断。其缺页率为10/12＝83.3%。

为进程分配4个物理块

走向	7	0	1	2	0	3	0	4	2	3	0	3
块1	7	7	7	7		3		3			3	
块2		0	0	0		0		4			4	
块3			1	1		1		1			0	
块4				2		2		2			2	
	缺	缺	缺	缺		缺		缺			缺	

- 从上表中可以看出，共发生了7次缺页中断。其缺页率为7/12＝58.3%。

最佳置换算法（OPT）

- 最佳算法：也称最优置换算法。从内存中选择最长时间不会使用的页面予以淘汰。
- 特点：因页面访问的未来顺序很难精确预测，该算法具有理论意义，可以用来评价其他算法的优劣。
- 如与最佳相比最坏不差12.3%，平均不差4.7%，就是可用算法。

最佳置换算法例

- 假定系统为某进程分配了3个物理块，页面访问序列为：1、2、3、4、1、2、5、1、2、3、4、5，开始时3个物理块均为空闲，采用最佳置换算法时的页面置换情况如下所示：

采用最佳置换算法的页面置换情况

走向	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	缺	缺	缺			缺			缺	缺	

- 从上表中可以看出，共发生了7次缺页，其缺页率为7/12＝58.3%。

最近最久未使用置换算法(LRU)

- **最近最久未使用算法**：选择最近一段时间内最长时间没有被访问过的页面予以淘汰。
- 为此，应赋予每个页面一个访问字段，用于记录页面自上次访问以来所经历的时间。
- LRU是Least Recently Used，也称为最近最少使用

LRU算法示例

- 假定系统为某进程分配了3个物理块，页面访问序列为：1、2、3、4、1、2、5、1、2、3、4、5，开始时3个物理块均为空闲，采用LRU置换算法时的页面置换情况如下所示：

采用LRU算法的页面置换情况

走向	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	缺	缺	缺	缺	缺	缺			缺	缺	缺

- 从上表中可以看出，共发生了10次缺页，其缺页率为10/12＝83.3%。

为进程分配4个物理块

走向	1	2	3	4	1	2	5	1	2	3	4	5
块1	1	1	1	1			1			1	1	5
块2		2	2	2			2			2	2	2
块3			3	3			5			5	4	4
块4				4			4			3	3	3
	缺	缺	缺	缺			缺			缺	缺	缺

- 从上表中可以看出，共发生了8次缺页中断。其缺页率为8/12＝66.7%。

LRU算法的实现

- LRU算法实现时需要较多的硬件支持，以记录进程中各页面自上次访问以来有多长时间未被访问。下面介绍几种实现方法。
  - 计数器法
  - 特殊栈

计数器实现

- 每个页表项有一个计数器，每次访问页时，把时间拷贝到计数器中。
- 置换计数器最小值的页
- 为每个页面配置一个计数器，其初值为0。当进程访问某页时，将计数器的最高位置1，定时器每隔一定时间将计数器右移1位，则数值最小的页是最近最久未使用的页面。

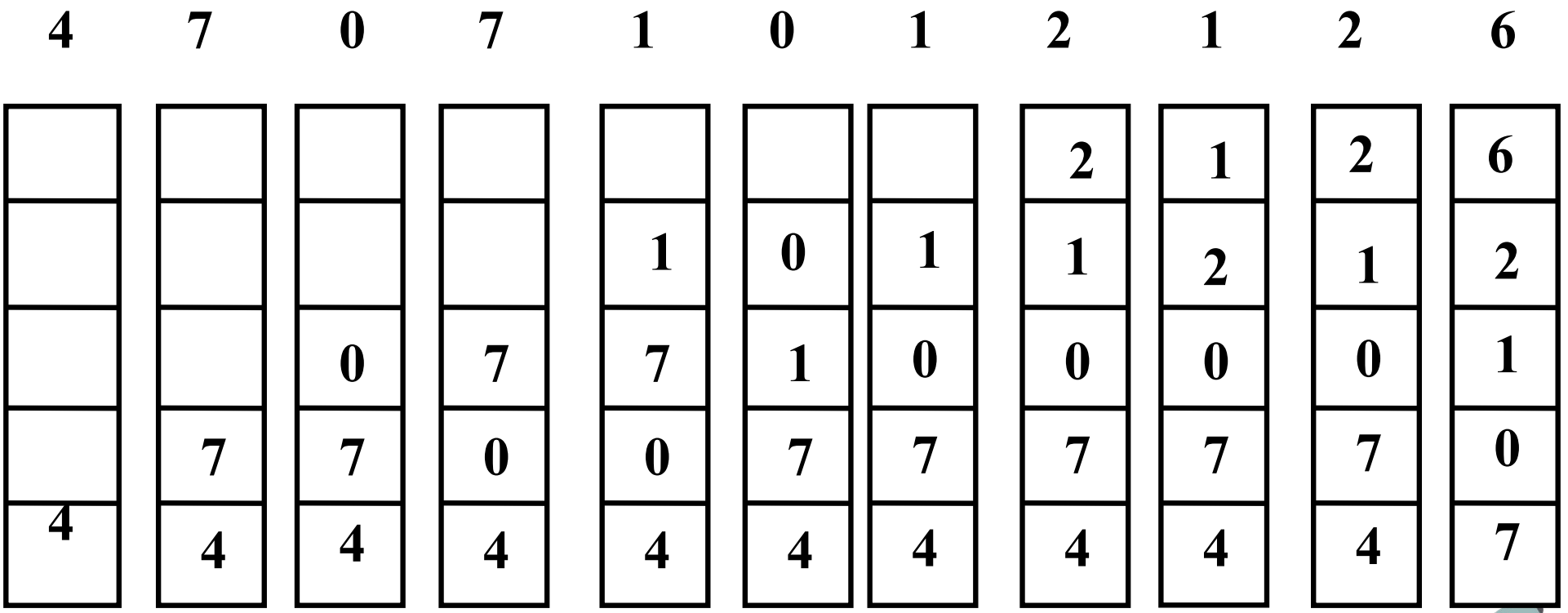
计数器实现方法

页面 \ R	R7	R6	R5	R4	R3	R2	R1	R0
1	0	1	0	1	0	0	1	0
2	1	0	1	0	1	1	0	0
3	1	0	0	0	0	0	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

最近最久未使用的页是？

特殊栈（Stack）

- 利用一个特殊栈保存当前使用各页的页面号。当进程访问一个页面时，将该页面号从栈中移出压到栈顶。栈底即最近最久未使用的页面号。



栈算法

- 最佳置换和LRU都没有belady异常，这类算法称为栈算法。
- 对于栈算法：帧数为n的内存页集合是帧数为n+1的页集合的子集。

LRU近似置换算法

- 近似LRU算法的基础：引用位（也称访问位）
  - 每个页表项都关联一个引用位，初始值为0
  - 当页被访问时将引用位设为1
  - 如果存在引用位为0的页，则置换它

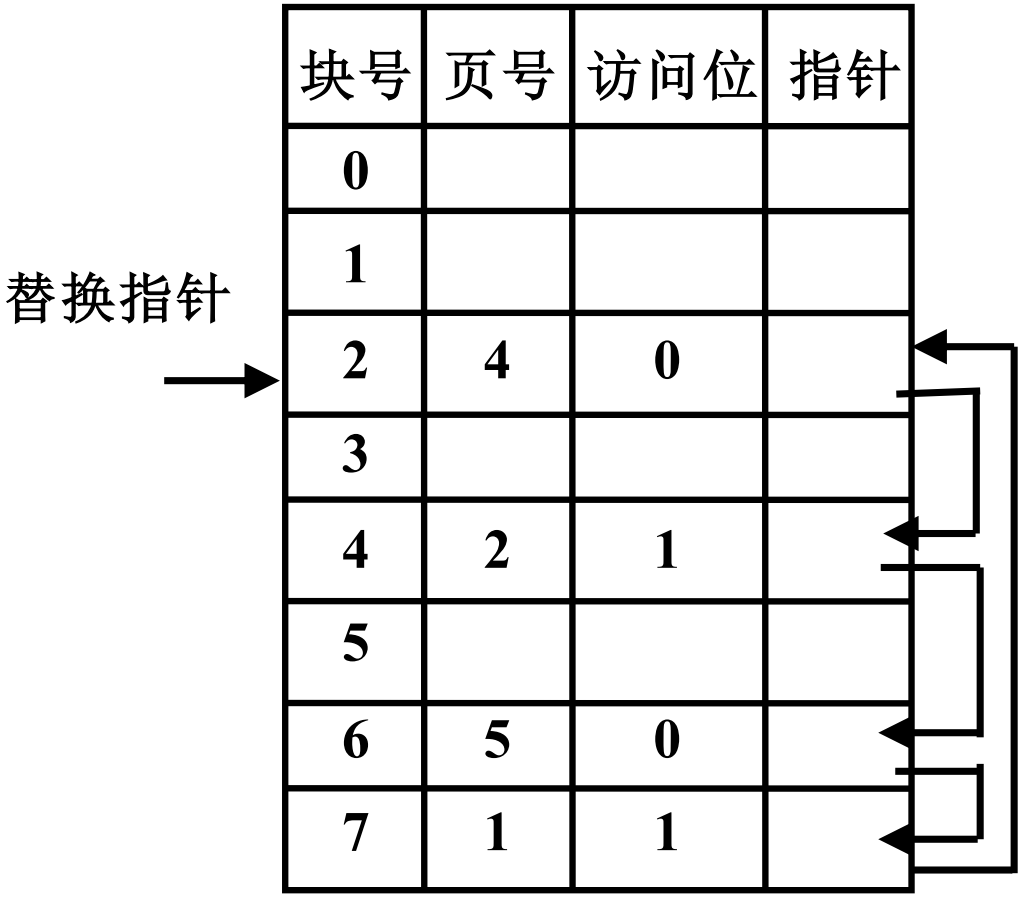
LRU近似置换算法

- 1.附加引用位算法
  - 与前面的计数器实现方法类似
- 2.二次机会算法：是FIFO算法的改进，以避免将经常使用的页面淘汰掉。
  - 算法思想：使用FIFO算法选择一页淘汰时，先检查该页的引用位，如果是0就立即淘汰该页，如果是1就给它第二次机会，将其引用位清0，并将它放入页面链的末尾，将其装入时间置为当前时间，然后选择下一个页面。

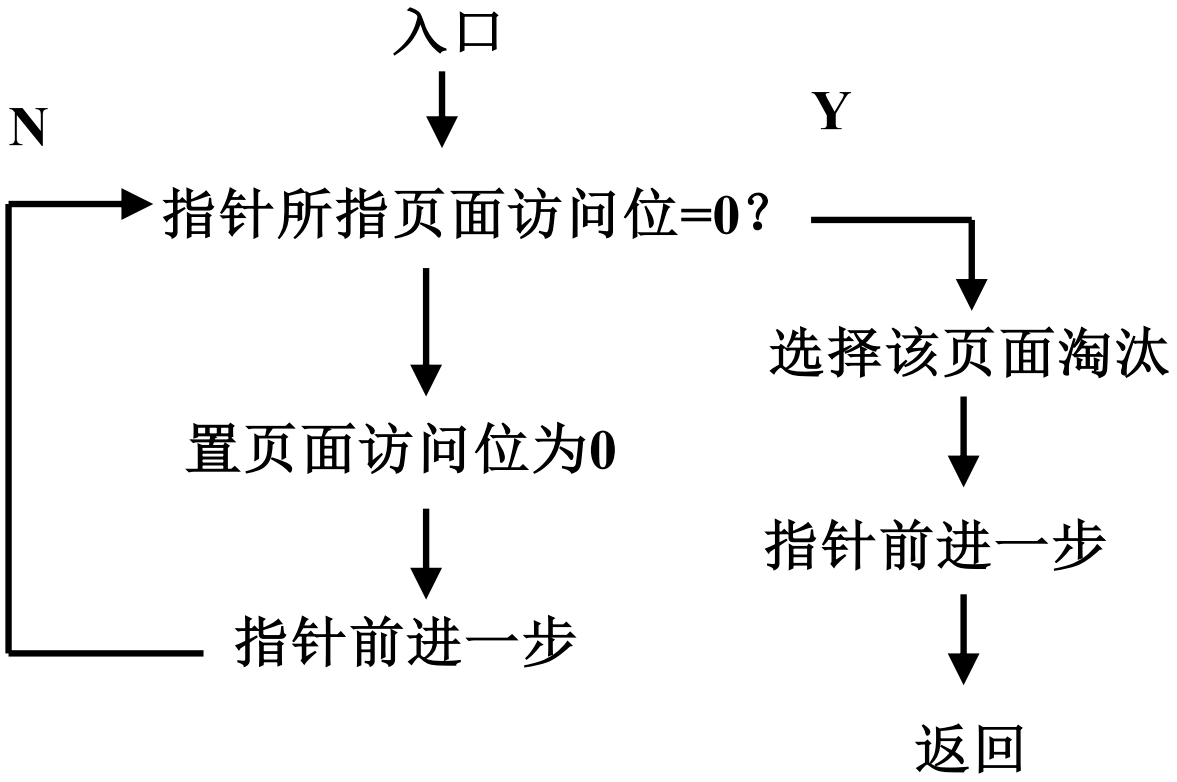
简单时钟（clock）算法

- **简单时钟置换算法**既是对二次机会算法的改进，也是对LRU算法的近似。该算法也要求为每页设置一个访问位。
- 实现思想：将页面排成一个循环队列，类似于时钟表面，并使用一个置换指针。当发生缺页时，检查指针指向的页面，若其访问位为0则淘汰该页，否则将该页的访问位清0，指针前移并重复上述过程，直到找到访问位为0的淘汰页为止。最后指针停留在被置换页的下一页上；
- 该算法也称为最近未使用（Not Recently Used，NRU）算法。

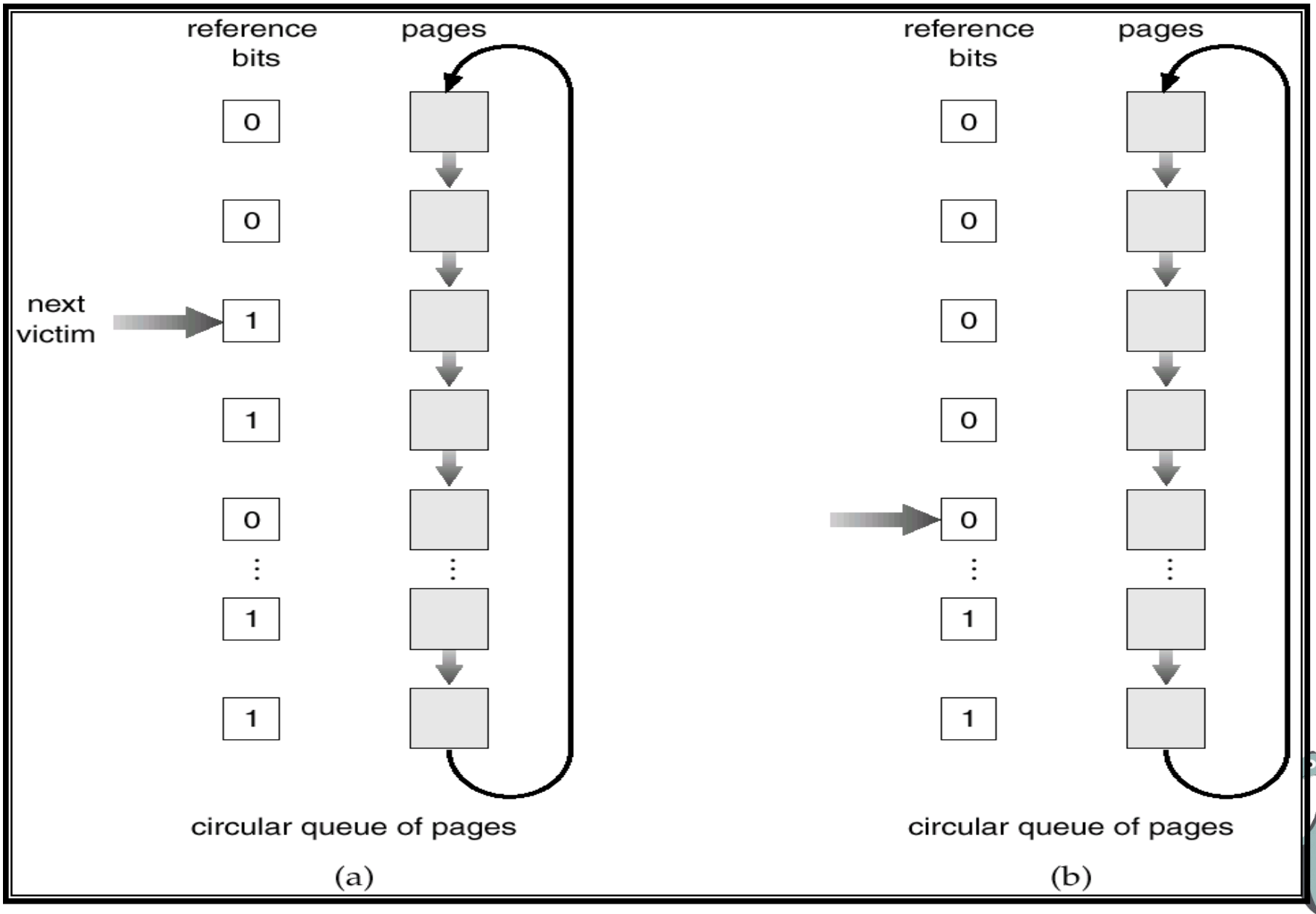
简单时钟置换算法页面链



简单时钟置换算法流程



Clock Page-Replacement Algorithm



页面置换情况

走向	7	0	1	2	0	3	0	4	2	3	0	3
块1	7*	7*	7*	2*	2*	2*	2*	4*	4*	4*	4	3*
块2		0*	0*	0	0*	0	0*	0	2*	2*	2	2
块3			1*	1	1	3*	3*	3	3	3*	0*	0*
	缺	缺	缺	缺		缺		缺	缺		缺	缺

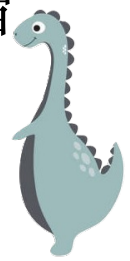
- 从上表中可以看出，共发生了9次缺页中断。其缺页率为9/12＝75% 。

改进的时钟算法

- 将一个修改过的页面换出需要写磁盘，其开销大于未修改页面。为此在改进型时钟算法中应考虑页面修改情况。设R为访问位，U为修改位，将页面分为以下4种类型：
  - 1类(R=0,U=0)：未被访问又未被修改
  - 2类(R=0,U=1)：未被访问但已被修改
  - 3类(R=1,U=0)：已被访问但未被修改
  - 4类(R=1,U=1)：已被访问且已被修改

改进型时钟算法描述

- 从指针当前位置开始扫描循环队列，寻找R=0,U=0的页面，将满足条件的第一个页面作为淘汰页。
- 若第1步失败，则开始第2轮扫描，寻找R=0,U=1的页面，将满足条件的第一个页面作为淘汰页，并将所有经历过页面的访问位置0。
- 若第2步失败，则将指针返回到开始位置，然后重复第1步，若仍失败则必须重复第2步。此时一定能找到淘汰页面。
- 特点：减少了磁盘I/O次数，但算法本身开销增加。



基于计数的页面置换

- **最不经常使用算法（LFU）**：选择到当前时间为止访问次数最少的页淘汰。
- 该算法要求为每页设置一个访问计数器，每当页被访问时，该页的访问计数器加1。发生缺页中断时，淘汰计数值最小的页面，并将所有计数器清零。
- 最常使用算法：

页面缓冲算法

- 页面缓冲算法是FIFO算法的发展。它在系统中保存一个空闲帧缓冲池。
- 页面缓冲置换算法采用FIFO选择被置换页面，选择出的页面不是立即换出，而是放入两个链表之一。

页面缓冲算法描述

- 算法采用FIFO选择淘汰页，规定将淘汰页放入两个链表之一：
  - 空闲链：页面未修改则放入空闲链末尾
  - 修改链：页面已修改则放入修改链末尾
- 当需要访问一个页面时，若该页在上述两个队列中，则只要将该页面从队列中移出，否则用空闲链的第一个物理块装入该页。当修改链中的页面数达到一定值时，再将它们一次写回磁盘，然后将它们加入空闲页面链表中。

页面分配

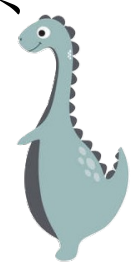
- 在为进程分配物理块时应确定:
  - 进程需要的最少物理块数
  - 进程的物理块数是固定还是可变
  - 按什么原则为进程分配物理块数

最小物理块数

- **最小物理块数**是指能保证进程正常运行所需的最少物理块数，若小于此值进程无法运行。
- 一般情况下：
  - 单地址指令且采用直接寻址，则所需最少物理块数为2。
  - 若单地址指令且允许采用间接寻址，则所需最少物理块数为3。
  - 某些功能较强的机器，指令及源地址和目标地址均跨两个页面，则最少需要6个物理块。

页面分配算法

- 页面分配是指按什么原则给进程分配物理块数，通常有以下几种分配算法：
  - **平均分配**（Equal allocation）：将系统中所有可供分配物理块平均分配给每个进程。
  - **按比例分配**（Proportional allocation）：根据进程大小按比例分配物理块。
  - **按优先级分配**(priority allocation):有两种方案
    - 将可供分配的物理块分为两部分，一部分按比例分配给每个进程，另一部分根据进程的优先级适当增加物理块。
    - 先按比例为进程分配物理块，当高优先级进程发生缺页时，允许它从低优先级进程处获得物理块。



全局分配与局部分配

- 页面分配常采用固定分配和可变分配两种策略。
- 页面置换也有全局置换和局部置换两种策略。
- 将分配策略和置换范围组合可得如下三种策略：
  - 固定分配局部置换
  - 可变分配全局置换
  - 可变分配局部置换

固定分配局部置换

- 固定分配局部置换：基于某种原则为每个进程分配固定数量的物理块，当进程运行中出现缺页时，只能从自己的页面中选择一页换出，然后再调入缺页。
- 实现这种策略的困难在于：难以确定应为每个进程分配多少个物理块。

可变分配全局置换

- 可变分配全局置换：先为系统中的每个进程分配一定数量的物理块，操作系统本身也保持一个空闲物理块队列。当某进程发生缺页时，由系统从空闲物理块队列中取出一个物理块分配给该进程；当空闲物理块队列中的物理块用完时，操作系统才从内存中选择一页调出，该页可以是系统中任何一个进程的页面。
- 这是一种容易实现的页面分配和置换策略，目前已用于若干操作系统中。

可变分配局部置换

- 可变分配局部置换：根据某种原则为每个进程分配一定数量的物理块，当进程发生缺页中断时，只允许从该进程的页面中选出一页换出。如果某进程的缺页率较高，则系统再为该进程分配若干物理块；反之若某进程的缺页率特别低，则系统可适当减少分配给该进程的物理块数。
- 该算法比较复杂，但性能较好。

抖动

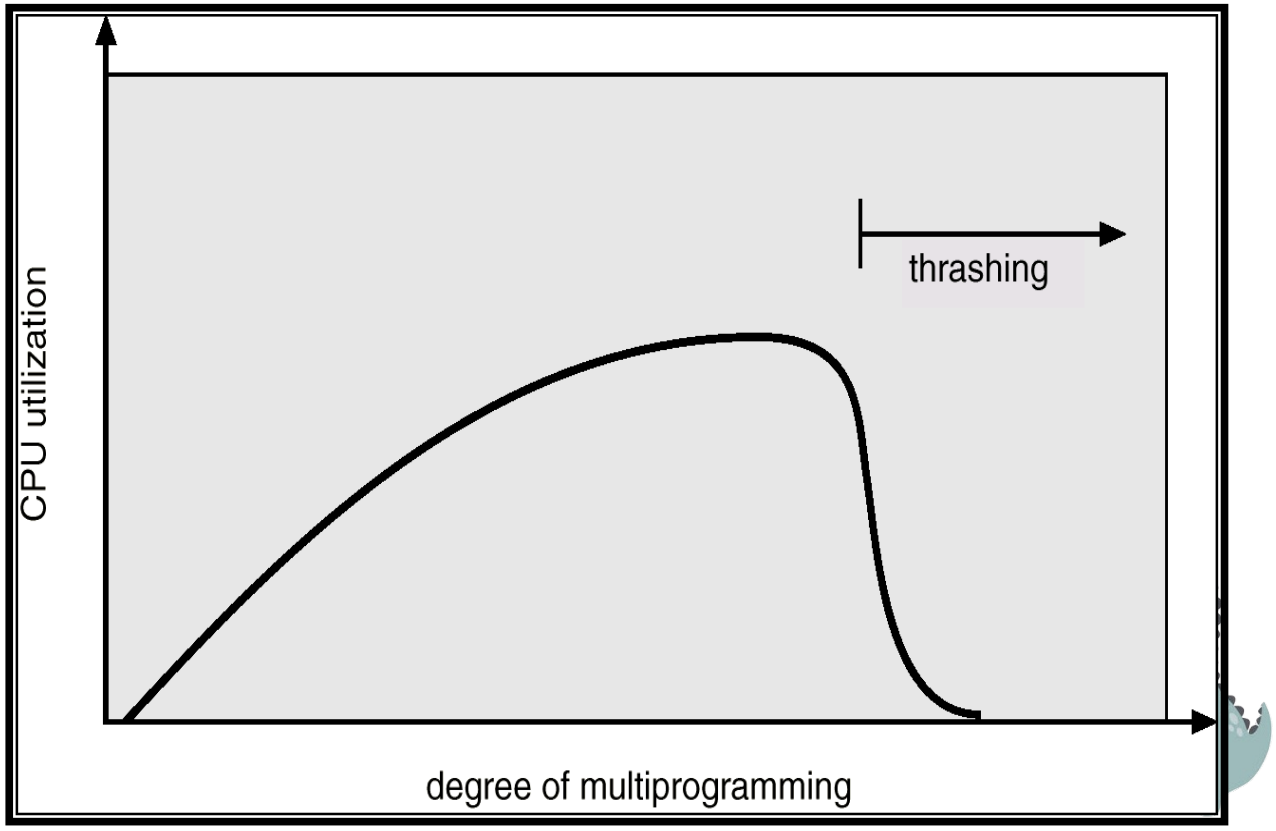
- **抖动**：如果运行进程的大部分时间都用于页面的换入/换出，而几乎不能完成任何有效的工作，则称此进程处于抖动状态。抖动又称为颠簸、颤动。
- 抖动分为：
  - 局部抖动
  - 全局抖动

抖动的原因

- 如果一个进程没有足够的页，那么缺页率将很高，这将导致：
  - CPU利用率低下 low CPU utilization.
  - 操作系统认为需要增加多道程序设计的道数、
  - 系统中将加入一个新的进程

CPU利用率与程序道数的关系

- 开始时CPU利用率会随着程序道数的增加而增加，当程序道数增加到一定数量以后，程序道数的增加会使CPU的利用率急剧下降。



抖动的原因

- 抖动产生的原因有：
  - 进程分配的物理块太少
  - 置换算法选择不当
  - 全局置换使抖动传播

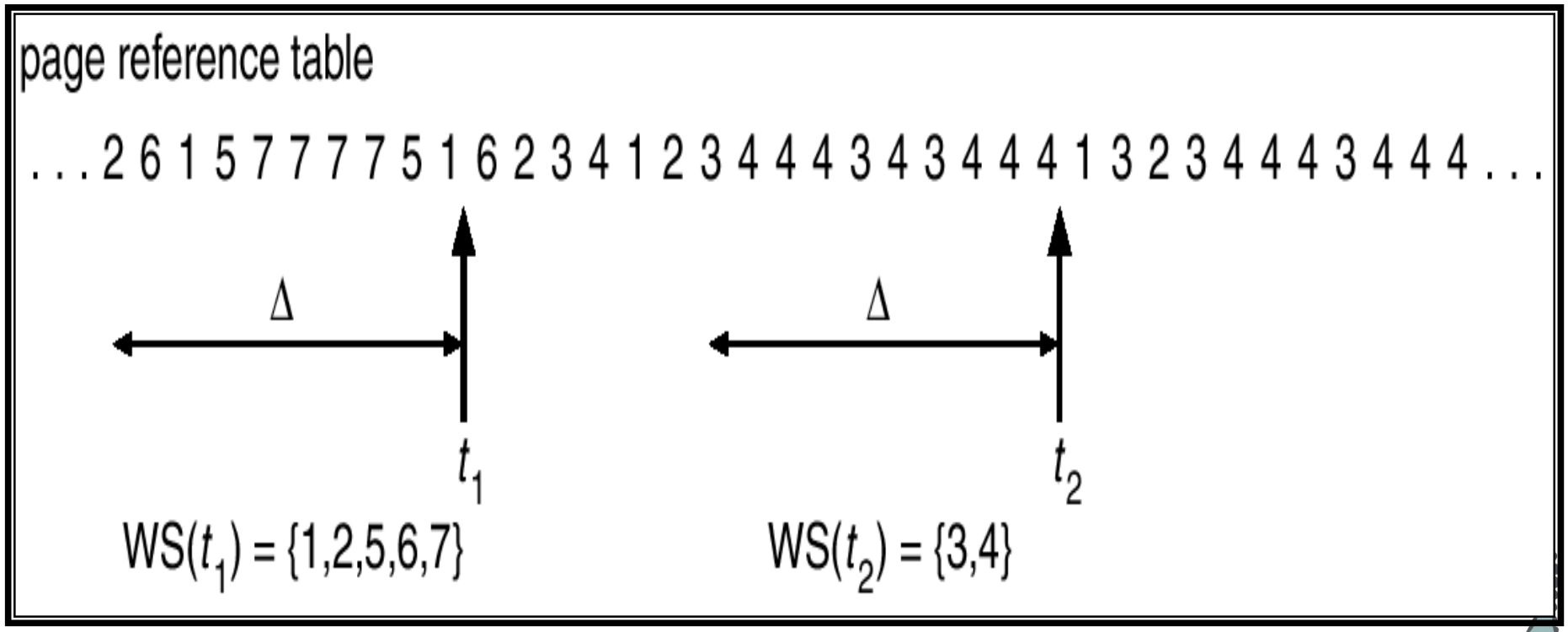
工作集模型

- 工作集窗口Δ：固定数量的页面引用。
- Example：10,000 instruction
- 进程Pi的工作集WSSi ： 最近Δ个页面引用的集合
-

例

- Δ为10个内存引用

  Δ=10 memory references



工作集模型

- 工作集精确度与Δ的选择有关：
  - 如果Δ太小，那么不能包含整个局部
  - 如果Δ太大，那么它可能包含多个局部
  - 如果Δ无穷大，那么包含整个程序

工作集模型

- 操作系统跟踪每个进程的工作集，并为进程分配大于其工作集的帧数。
- 如果还有空闲帧，那么可启动另一进程
- 如果所有工作集之和的增加超过了可用帧的总数，则选择暂停一个进程。
-

抖动的发现

- 抖动发生前会出现一些征兆，可利用这些征兆发现抖动并加以防范。这些技术有：
  - 全局范围技术
  - L=S准则
  - 利用缺页率发现抖动
  - 平均缺页频率

全局范围技术

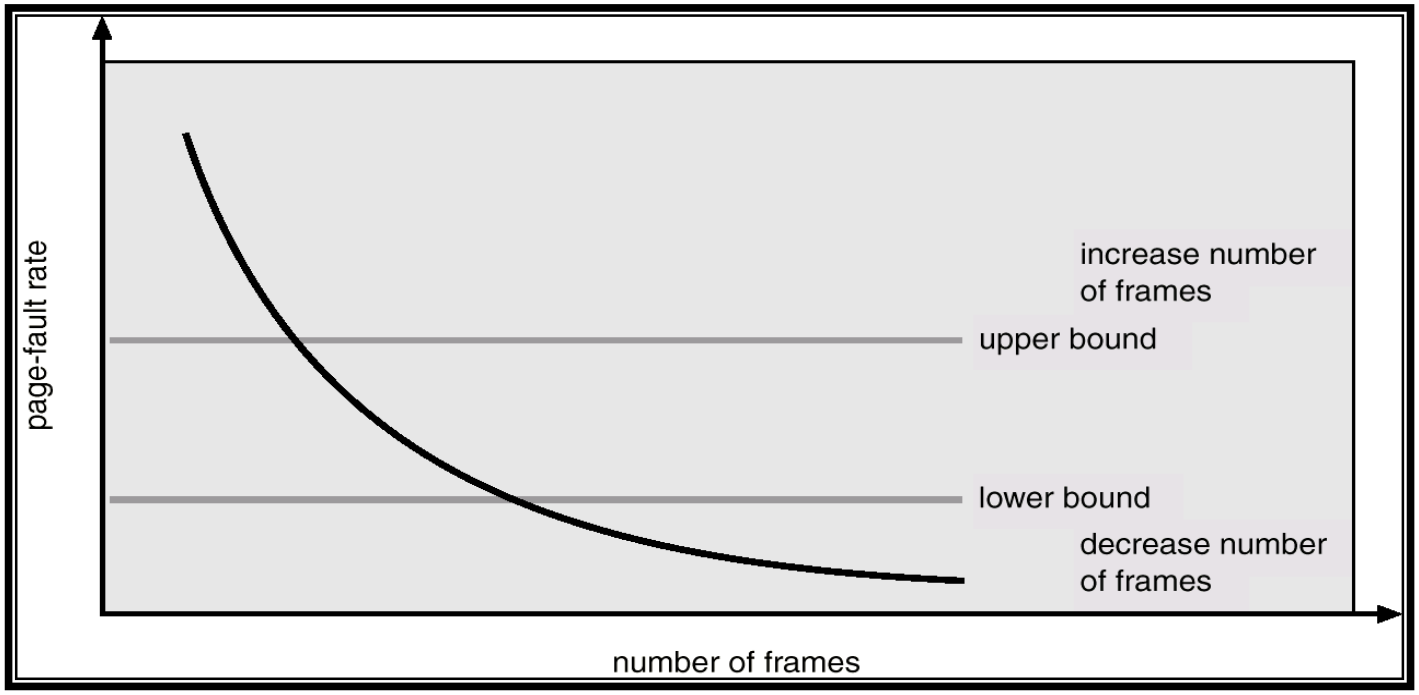
- 全局范围技术采用时钟置换算法，用一个计数器C记录搜索指针扫描页面缓冲的速度。
  - 若C的值大于给定的上限值，说明缺页率太高（可能抖动）或找不到可供置换的页面，这时应减少程序道数。
  - 若C小于给定的下限值，表明缺页率小或存在较多可供置换的页面，这时应增加程序的道数。

L=S准则

- 实际证明，产生缺页的平均时间L等于系统处理缺页的平均时间S时，CPU的利用率达到最大。
- 当L<S时，表明系统频繁缺页，CPU利用率低，会导致系统抖动。

利用缺页率发现抖动

- 下图是缺页率与进程分得物理块数之间的关系。当缺页率超过上限时会引起抖动，因此应增加分配给进程的物理块；此时每增加一个物理块，其缺页率明显降低；当进程缺页率达到下限值时，物理块的进一步增加对进程缺页率的影响不大。



缺页率算法

- 缺页率算法是一种直接的控制抖动方法。该方法要求为每页设一个使用位，当该页被访问时，相应位置1。同时设计一个计数器，记录自上次进程产生缺页以来进程执行的时间。
- 方法1：设置一个阈值F，如最近两次缺页时间间隔小于F，则分配一个物理块给该进程；否则淘汰使用位为0的页，并减少该进程的物理块数，同时将该进程的剩余页使用位重置为0。

缺页率算法

- 方法2：设置两个阈值，当缺页率达到上限值时为进程增加物理块，当缺页率达到下限值时减少进程的物理块。
- 缺页率算法的缺点：当进程由一个局部转移到另一个局部时，在原局部中的页面未移出内存之前，连续的缺页会导致该进程在内存的页面迅速增加，产生对内存请求的高峰。

平均缺页频率

- 设t_i为两次缺页之间的间隔时间，f_i为其缺页频率，则有： $f_i = 1/t_i$
- 设F为平均缺页频率，则有：
  - $F=（f_1+f_2+\ldots+f_n）/n$
- 当F大于系统中规定的允许缺页频率时，则说明系统中缺页率过高，有可能引起抖动。

抖动的预防及解除

- 采用局部置换策略可以防止抖动传播
- 利用工作集模型防止抖动：给进程分配工作集所需的物理块。
- 挂起进程来解决抖动。
- 选择挂起进程的条件
  - 优先级最低：符合进程调度原则
  - 发生缺页中断的进程：内存不含工作集，缺页时应阻塞
  - 最后被激活的进程：工作集可能不在内存
  - 最大的进程：可释放较多空间

其他考虑

- 置换算法及分配策略的选择是调页系统的主要问题
- 然而还有其他一些问题要考虑

预调页

- 预调页：将需要的所有页一起调入内存。
  - 减少进程启动时发生的大量页面错误
  - 在引用之前预取进程需要的所有或部分页面
  - 但是，如果预取的页面未被使用，I/O和内存就会被浪费

页面尺寸

- 没有单一的最佳页大小存在，许多因素会影响页大小。
  - 页表大小，应选大页面。
  - 页内碎片，应选小页面。
  - 页读写时间，应选大页面。
  - 局部性 locality。采用小页面导致更少的I/O和更少的总的分配内存；但为了降低页错误数量，需要大页。矛盾！

页面大小选择的分析

- 设系统内每个进程的平均长度为s，页面大小为p，每个页表项需e个字节。
  - 每个进程页表长度：s/p
  - 页表占用空间：se/p
  - 每进程碎片的平均大小：p/2
  - 每进程浪费的空间为：se/p+p/2
  - 对p求导数：$=1/2-se/p^2$
  - 令导数为0求最小值：$1/2s-e/p^2=0$
  - 则页面大小为：$p=\sqrt{2es}$时，f最小。

页面大小选择的分析

- 页的大小总为2的幂。
- 页面尺寸多在512B到16MB之间。

TLB范围

- TLB范围：从 TLB 可访问的存储器数量。
- TLB Reach = (TLB Size) X (Page Size)
- 理想地，每进程的工作集存储在 TLB中。否则有很高的缺页率

TLB范围

- 增加TLB范围的方法：
  - 增加页大小：这可能导致碎片的增加。
  - 提供多种页大小：允许需要大页面的应用程序使用大页面而不增加碎片。

程序结构对缺页率影响例

- 设页面大小为128字节，考虑一个程序，其功能是将128×128的二维数组清零

- 程序1

```
short int a[128][128];
for (j=0;j<=127;j++)
for (i=0;i<=127;i++)
a[i][j]=0;
```

程序2

```
for (i=0;i<=127;i++)
for (j=0;j<=127;j++)
a[i][j]=0;
```

程序1的缺页次数

■程序1

```
short int a[128][128];
for (j=0;j<=127;j++)
  for (i=0;i<=127;i++)
    a[i][j]=0;
```

- 因数组以行为主存放，页面大小为128字节，故每行占一个页面。
- 程序1的内层循环将每行中的指定列置为0，故产生128次中断。
- 外层循环128次，总缺页次数为128×128。

程序2的缺页次数

■程序2

```
short int a[128][128];
for (i=0;i<=127;i++)
  for (j=0;j<=127;j++)
    a[i][j]=0;
```

- 因数组以行为主存放，页面大小为128字节，故每行占一个页面。
- 程序2的内层循环将每行的所有列置为0，故产生1次中断。
- 外层循环128次，总缺页次数为128。

请求分段存储管理

- 请求分段存储管理是另一种实现虚拟存储器的方法。
- 分段有如下优点：
  - 有利于动态增长
  - 允许按段进行编译
  - 有利于共享
  - 有利于保护

请求分段的实现思想

- 请求分段存储管理系统基于分段存储管理，是在分段存储管理系统的基础上，增加了请求调段、分段置换功能所形成的一种虚拟存储系统。
- 在请求分段存储管理中，作业运行之前，只要求将当前需要的若干个分段装入主存，便可启动作业运行。在作业运行过程中，若所要访问的分段不在主存，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上，以便腾出内存空间。

请求分段的支持机构

- 请求分段的支持机构有：
  - 段表
  - 缺段中断机构
  - 地址变换机构
  - 请求调段和分段置换软件

段表结构

- 请求分段系统中使用的主要数据结构仍然是段表。
- 由于每次只将作业的一部分调入内存，还有一部分内容存放在磁盘上，故需扩充段表表项，扩充后的段表项如下所示：

段表项中各字段的说明

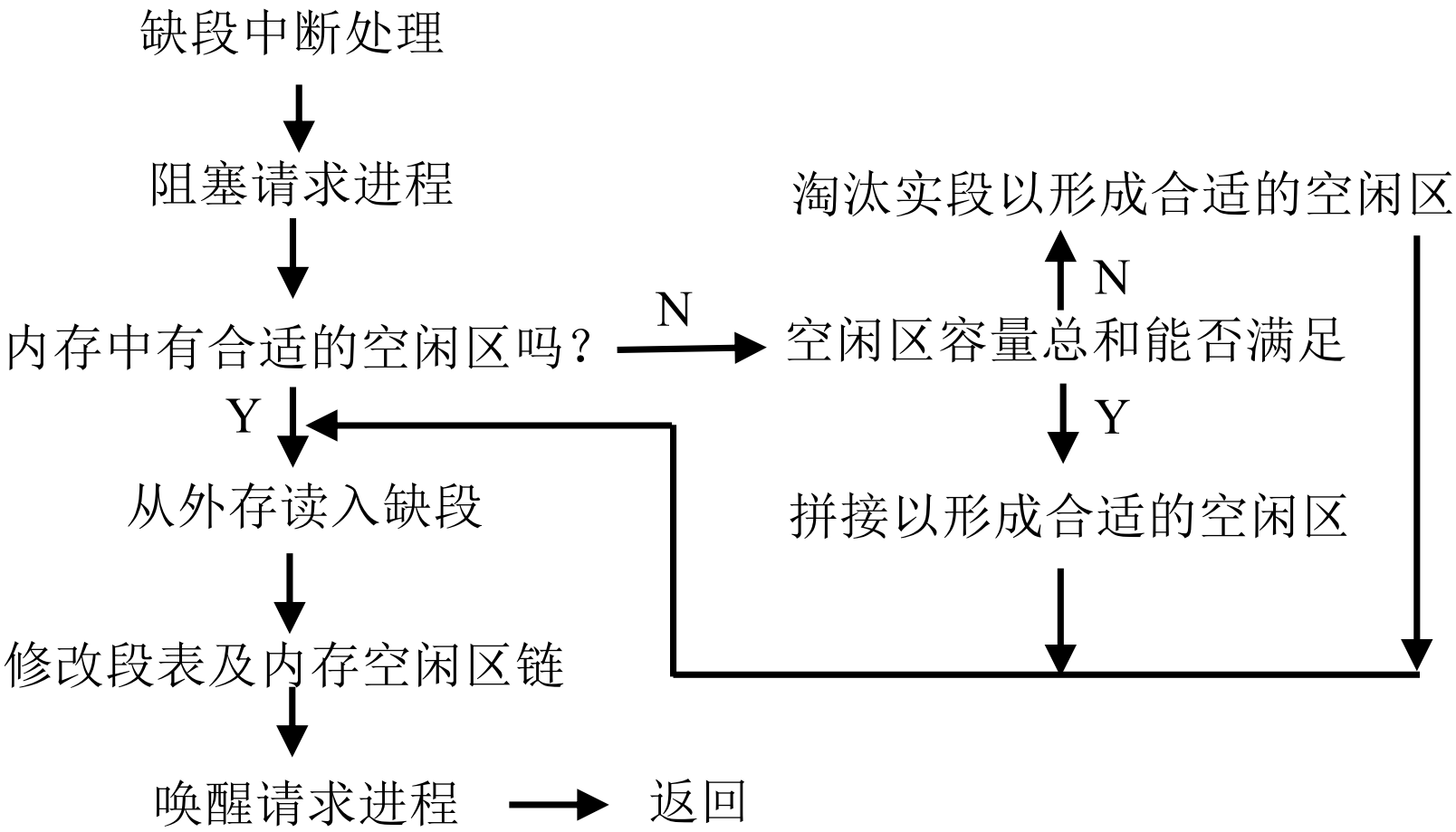
段号	段长	段基址	存取方式	访问字段	修改位	存在位	增补位	外存地址

- 段号、段长和段基址：其定义同分段存储管理。
- 存取方式：标识该分段的存取方式：读、写、执行。
- 访问字段：记录该段在一段时间内被访问的次数。
- 修改位：表示该段调入内存后是否被修改过。
- 存在位：表示该段是否在主存中。
- 增补位：指出该段在运行过程中，是否作过动态增长。
- 外存地址：指出该段在外存上的地址。

缺段中断处理

- 在请求分段系统中，每当所访问的段不在内存时，便产生一个缺段中断信号，请求OS将所缺分段调入内存。
- 缺段中断与缺页中断类似，也是在指令执行期间产生和处理。

缺段中断处理流程


缺段中断处理
阻塞请求进程
内存中有合适的空闲区吗？
N
Y
从外存读入缺段
修改段表及内存空闲区链
唤醒请求进程
返回
空闲区容量总和能否满足
N
Y
拼接以形成合适的空闲区
淘汰实段以形成合适的空闲区

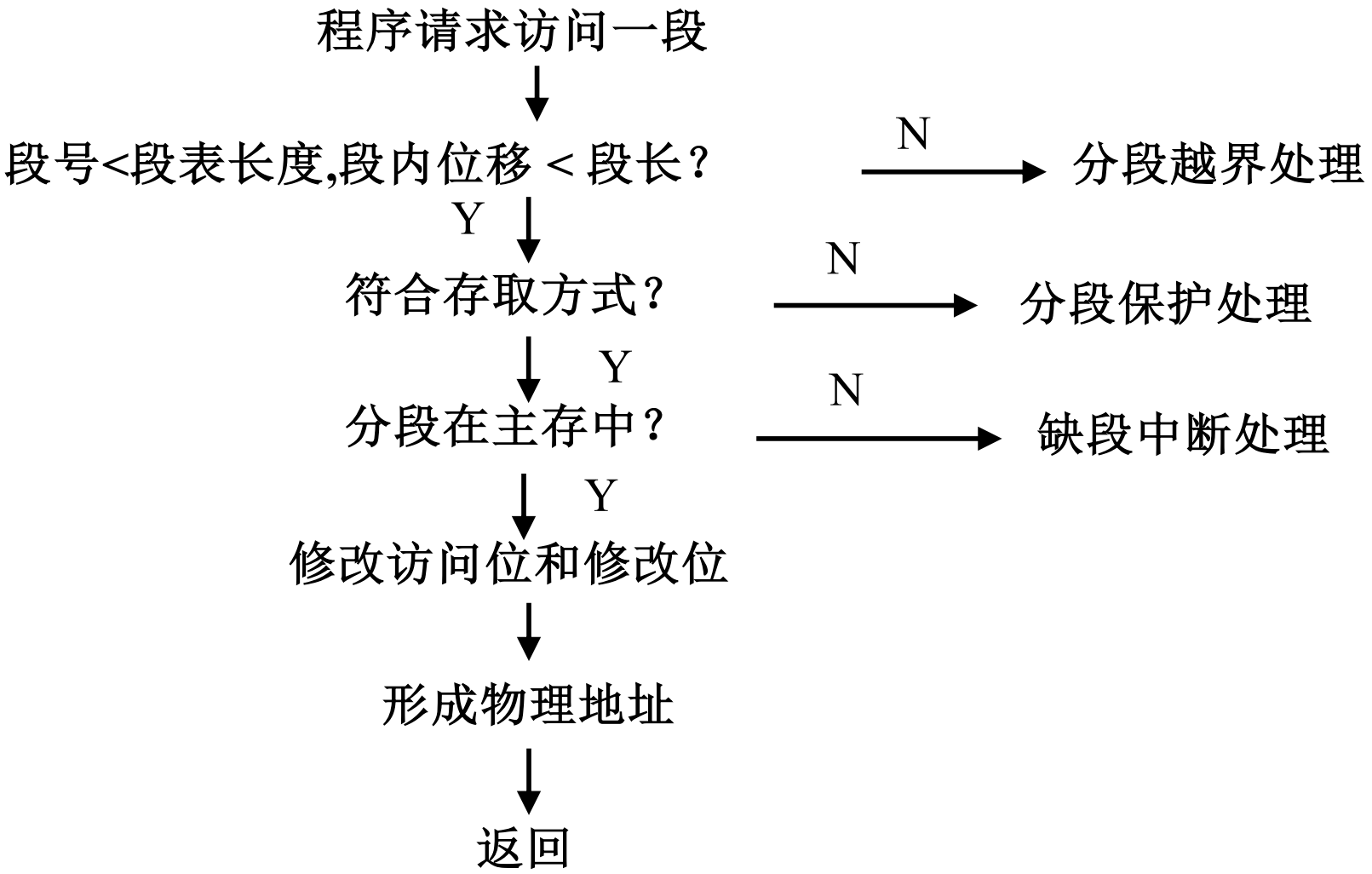
缺段中断涉及的几个问题

- 内存管理：请求分段的内存管理与与分区管理类似，采用连续内存管理方式。
- 段的置换：调入缺段时，若有足够的空闲分区则直接调入，否则查看空闲区之和能否满足要求，若能则进行拼接，否则进行置换。置换时可能要淘汰多段。

动态地址变换

- 请求分段系统的地址变换是在分段系统地址变换的基础上形成的。
- 在地址变换过程中，若段在内存则判断其存取权限是否合法，若合法则从段表中取出该段在内存的起始地址，与段内位移相加形成访问内存的物理地址。
- 若段不在内存，则产生缺段中断信号，请求OS将缺段调入内存，然后修改段表，再利用段表进行地址变换。

请求分段的地址变换过程



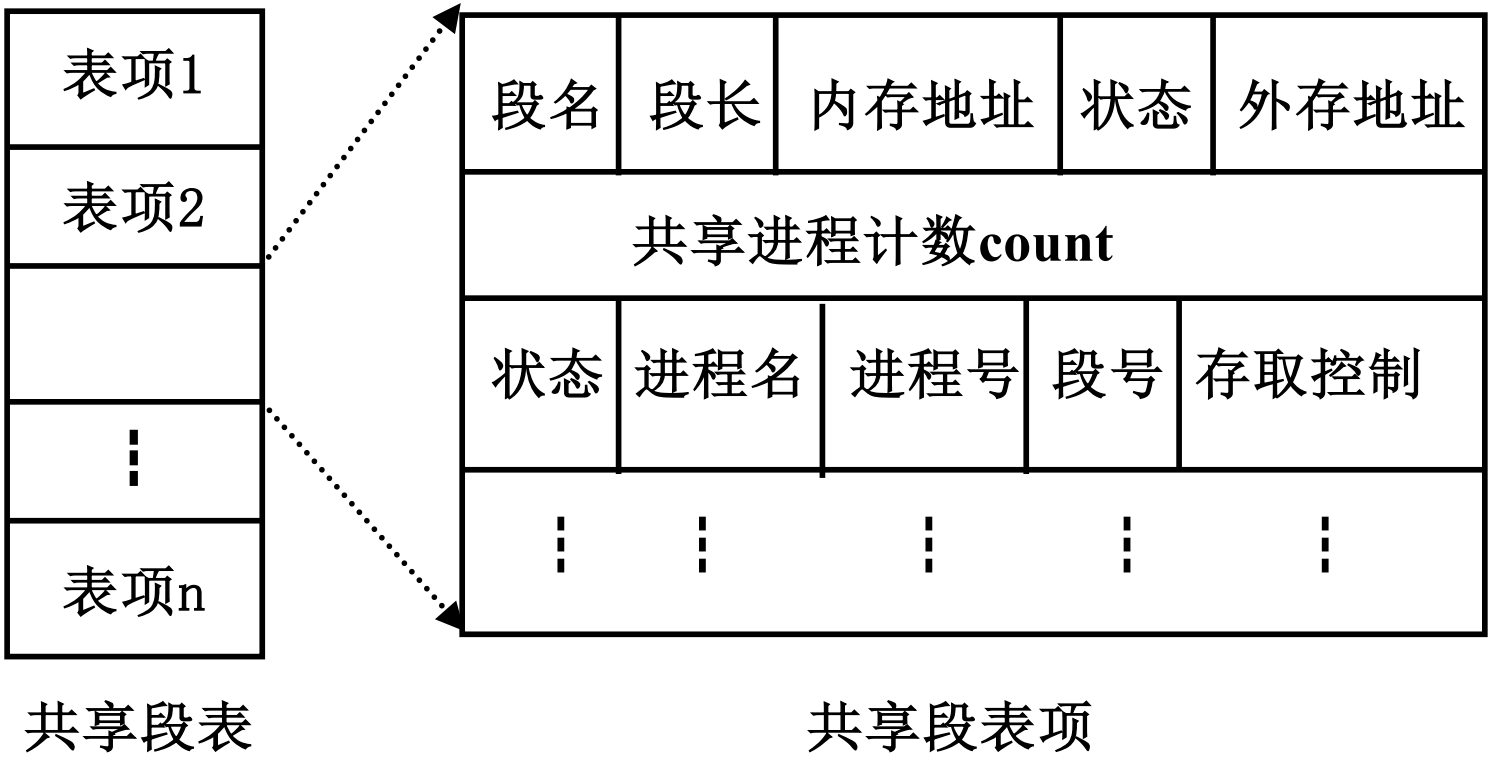
引入快表提高访问速度

- 与请求分页管理一样，请求分段的地址变换过程必须访问内存两次以上。
- 为提高访问速度，也可以使用快表。

段的共享与保护

- 为了实现分段共享，可在系统中设置一张共享段表，所有共享分段都在其中占有一表项。共享段表的格式如下：

共享段表及表项



- 共享进程计数：记录有多少进程共享该段。
- 存取控制字段：对同一共享段，不同进程有不同的操作权限。
- 段号：共享段在不同进程中有不同的段号。

共享段的分配

- 当为共享段分配内存时，对第一个请求使用该段的进程，系统为该共享段分配一个内存区，再将该共享段调入，同时将该区的始址填入请求进程的段表，还需在共享段表中增加一项，填写有关数据并将count置1。
- 此后，当又有其他进程申请使用该共享段时，只需将count加1并填写有关数据结构。

共享段的回收

- 当某进程不再需要使用共享段时，应释放该段。此时应将该进程段表中共享段的对应表项撤消，并将count减1，若结果为0则需要系统回收共享段的物理内存及有关表项，否则仅取消该进程在共享段中的记录。

虚拟段页式存储管理系统

- 虚拟段页式存储管理系统是虚拟页式和虚拟段式存储管理的结合。

练习

- 9.2 Why are page sizes always powers of 2?
- 9.4 Consider a logical address space of 64 pages of 1,024 words each, mapped onto a physical memory of 32 frames. a. How many bits are there in the logical address? b. How many bits are there in the physical address?
- 9.22 The MPV operating system is designed for embedded systems and has a 24-bit virtual address, a 20-bit physical address, and a 4-KB page size. How many entries are there in each of the following? a. A conventional, single-level page table b. An inverted page table What is the maximum amount of physical memory in the MPV operating system?
- 9.25 Consider a paging system with the page table stored in memory. a. If a memory reference takes 50 nanoseconds, how long does a paged memory reference take? b. If we add TLBs, and if 75 percent of all page-table references are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes 2 nanoseconds, if the entry is present.)

练习

- 10.7 Consider the two-dimensional array A:

  int A[][] = new int[100][100];

Where A[0][0] is at location 200 in a paged memory system with pages of size 200. A small process that manipulates the matrix resides in page 0(locations 0 to 199). Thus, every instruction fetch will be from page 0. For three page-frames, how many page faults are generated by the following array-initialization loops? Use LRU replacement, and assume that page frame 1 contains the process and the other two are initially empty.

a.
```
for (int j = 0; j < 100; j++)
    for (int i = 0; i < 100; i++)
        A[i][j] = 0;
```

b.
```
for (int i = 0; i < 100; i++)
    for (int j = 0; j < 100; j++)
        A[i][j] = 0;
```

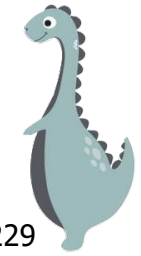
练习

- 10.9 Consider the following page reference string:

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0 , 1.

Assuming demand paging with three frames, how many page faults would occur for the following replacement algorithms?

- LRU replacement
- FIFO replacement
- Optimal replacement



练习

- 10.13 Consider a demand-paged computer system where the degree of multiprogramming is currently fixed at four. The system was recently measured to determine utilization of the CPU and the paging disk. Three alternative results are shown below. For each case, what is happening? Can the degree of multiprogramming be increased to increase the CPU utilization?

  a. CPU utilization 13 percent; disk utilization 97 percent

  b. CPU utilization 87 percent; disk utilization 3 percent

  c. CPU utilization 13 percent; disk utilization 3 percent

选择题1

- 首次适应算法的空白区是 _____ 。

  A. 按大小递减顺序连在一起

  B. 按地址由大到小排列

  C. 按地址由小到大排列

  D. 按大小递增顺序连在一起

- 在分区存储管理中的拼接技术可以 _____ 。

  A. 缩短访问周期　　B. 增加内存容量

  C.集中空闲区　　D. 加速地址转换

选择题2

- 采用 _____ 不会产生内部碎片。

  A.分页存储管理

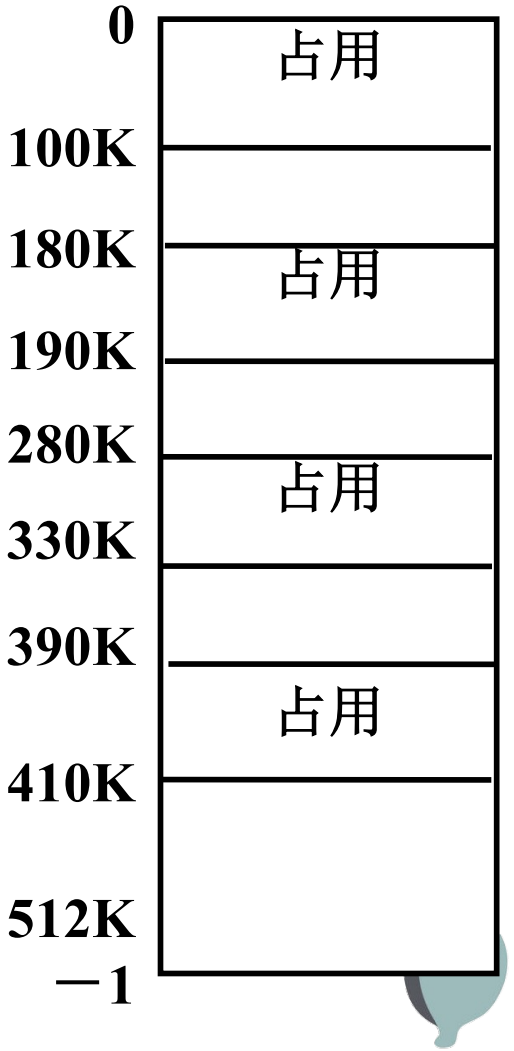
  B.固定分区存储管理

  C.分段存储管理

  D.段页式存储管理

- 设内存分配情况如右图所示。若要申请一块40K字节的内存空间，采用最佳适应算法，则所得到的分区首址为 _____ 。

  A. 190K B. 100K C. 410K D. 330K



选择题3

- 采用分段存储管理的系统中，若地址用24位表示，其中8位表示段号，则允许每段的最大长度是_____。

  A. 2^{16}　B. 2^{32}　C. 2^{24}　D. 2^{8}

- 在固定分区分配中，每个分区的大小是_____。

  A.可以不同但预先固定

  B.相同

  C.随作业长度变化

  D.可以不同但根据作业长度固定

选择题4

- 把作业地址空间使用的逻辑地址变成内存的物理地址称为 _____ 。

  A.加载　　B.重定位

  C.物理化　　D.逻辑化

- 在以下存储管理方案中，不适用于多道程序设计系统的是 _____ 。

  A.固定式分区分配　　B.单一连续分配

  C.可变式分区分配　　D.页式存储管理

选择题5

- 在可变式分区分配方案中，某一作业完成后，系统收回其内存空间并与相邻空闲区合并，为此需修改空闲区表，造成空闲区数减1的情况是_____。

  A.有下邻空闲区但无上邻空闲区

  B.有上邻空闲区但无下邻空闲区

  C.有上邻空闲区也有下邻空闲区

  D.无上邻空闲区也无下邻空闲区

- 采用两级页表的页式存储管理中，按给定的逻辑地址进行读写时，通常需访问主存的次数是_____。

  A. 1次　　B. 2次　　C. 3次　　D. 4次

选择题6

- 分页系统中的页面是_____。

  A．用户感知的　　B．操作系统感知的

  C．编译程序感知的 D．链接装配程序感知的

- 下述内存分配算法中，_____ 更易产生无法利用的小碎片。

  A．首次适应算法　B．循环首次适应算法

  C．最佳适应算法　D．最坏适应算法

选择题7

- 实现虚拟存储器的目的是 _____ 。

  A. 实现存储保护　　B. 实现程序浮动

  C. 扩充辅存容量　　D. 扩充内存容量

- 页式虚拟存储管理的主要特点是 _____ 。

  A. 不要求将作业装入到内存的连续区域

  B. 不要求将作业同时全部装入到内存的连续区域

  C. 不要求进行缺页中断处理

  D. 不要求进行页面置换

选择题8

- 作业在执行中发生了缺页中断，经操作系统处理后，应让其执行 _____ 指令。

  A. 被中断的前一条　　B. 被中断的那条

  C. 被中断的后一条　　D. 启动时的第一条

- 虚拟存储管理系统的基础是程序的 _____ 理论。

  A. 局部性　　B. 全局性

  C. 动态性　　D. 虚拟性

选择题9

- 在以下存储管理方案中，属于虚拟存储器管理的是 _____ 。

  A. 可重定位分区分配　　B. 分段存储管理

  C. 请求分页存储管理　　D. 段页式存储管理

- 由于实现_____ 页面置换算法的成本高，通常使用一种近似的页面置换算法_____算法。

  A.Optimal　LRU　　B.LRU　Clock

  C.FCFS　Clock　　D.Clock　改进的Clock

选择题10

- 会产生Belady异常现象的页面置换算法是_____。

  A．最佳页面置换算法

  B．先进先出页面置换算法

  C．最近最久未使用置换算法

  D．最少使用页面置换算法

- 在请求分页存储管理系统中，下述_____策略是不适用的

  A．固定分配局部置换　　B．固定分配全局置换

  C．可变分配全局置换　　D．可变分配局部置换

选择题11

- 二次机会置换算法与简单时钟置换算法在决定淘汰哪一页时，都用到了_____。

  A．快表　　B．引用位

  C．修改位　　D．存在位

- 请求段页式系统_____。

  A．是以页为单位管理用户的虚空间，以段为单位管理内存空间。

  B．是以段为单位管理用户的虚空间，以页为单位管理内存空间。

  C．是以连续的内存区存放每个段。

  D．为提高内存利用率，允许用户使用大小不同的页。

填空题1

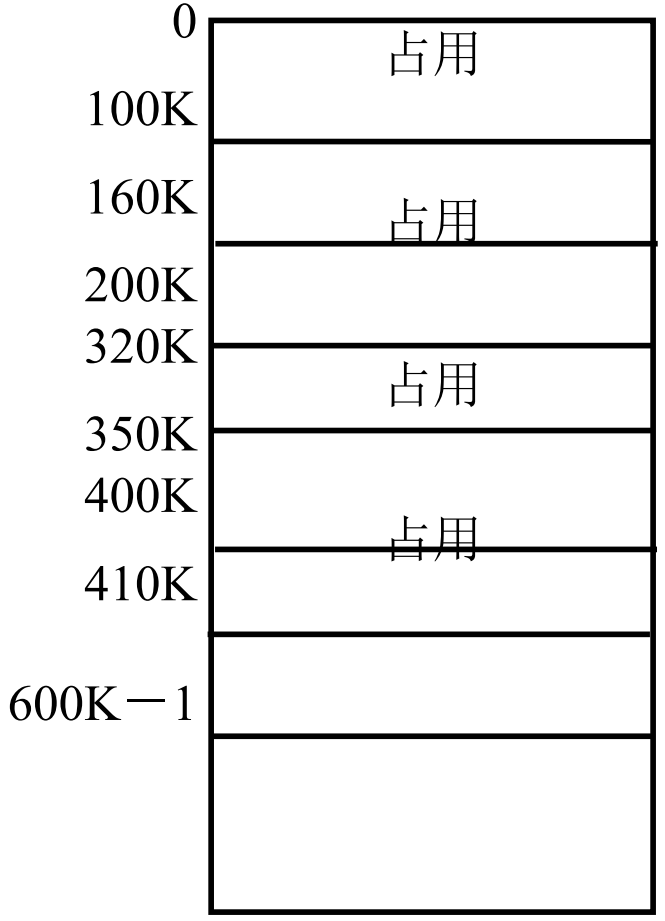
- 在分区分配算法中，首次适应算法倾向于优先利用内存中的 ① 部分的空闲分区，从而保留了 ② 部分的大空闲区。
- 段页式存储管理中，是先将作业分 ① ，② 内分 ③ 。分配以 ④ 为单位。在不考虑使用联想存储器的情况下，执行程序时需要 ⑤ 次访问内存，其中第 ⑥ 次是查作业的页表。

填空题2

- 把作业装入内存中随即进行地址变换的方式称为 ①，而在作业执行期间，当访问到指令或数据时才进行地址变换的方式称为 ② 。
- 三种不连续内存管理方式是：①、②和③。
- 分区存储管理可以分为：①分区和②分区。

填空题3

- 对右图所示的内存分配情况，若要申请30K的存储空间，使首地址最大的分配策略是_____。



填空题4

- 在请求页式存储管理系统中，常用的页面淘汰算法有：__①__，选择淘汰不再使用或最远的将来才使用的页；__②__，选择淘汰在内存驻留时间最长的页。
- 程序运行时的局部性表现为：__①__和__②__。
- 虚拟存储器的特点是__①__、__②__、__③__、__④__。

填空题5

- 所谓虚拟存储器是指具有 ① 和 ② 功能，能从逻辑上对内存容量进行扩充的一种存储器系统。
- 虚拟存储器的实现方法有三种 ① 、 ② 和 ③ 。
- 在请求页式系统中，当访问的页不在主存时，由 ① 将该页调入内存；当主存无空闲块时，必须 ② 一页。

考研题1

- 分区分配内存管理方式的主要保护措施是（）。09

  A、界地址保护　　B、程序代码保护

  C、数据保护　　D、栈保护

- 一个分段存储管理系统中，地址长度为32位，其中段号占8位，则最大段长为（）。09

  A、2^8字节　　B、2^{16}字节

  C、2^{24}字节　　D、2^{32}字节

考研题2

- 某基于动态分区存储管理的计算机，其主存容量为55MB（初始为空），采用最佳适配算法，分配和释放的顺序为：分配15MB，分配30MB，释放15MB，分配8MB，分配6MB，此时主存中最大空闲分区的大小是____。10

A.7MB　　B.9MB　　C.10MB　　D.15MB

考研题3

- 某计算机采用二级页表的分页存储管理方式，按字节编址，页大小为2^{10}字节，页表项大小为2字节，逻辑地址结构为：

页目录号	页号	页内偏移量

逻辑地址空间大小为2^{16}页，则表示整个逻辑地址空间的页目录表中包含表项的个数至少是____。10

A.64　　B.128　　C.256　D.512

考研题4-1

- 请求分页管理系统中，假设某进程的页表内容如下表所示：

页号	页框号	有效位
0	101H	1
1	-	0
2	254H	1

- 页面大小为4KB，一次内存的访问时间是100ns，一次快表（TLB）的访问时间是10ns，处理一次缺页的平均时间是10^8ns（已含更新TLB表和页表的时间），进程的驻留集大小固定为2，采用最近最少使用置换算法（LRU）和局部淘汰策略。假设（1）TLB初始为空

考研题4-2

- （2）地址转换时先访问TLB，若TLB未命中，再访问页表（忽略访问页表之后的TLB更新时间）；（3）有效位为0表示页面不在内存，产生缺页中断，缺页中断处理后，返回到产生缺页中断的指令处重新执行。设有虚地址访问序列2362H，1565H，25A5H，请问
- （1）依次访问上述三个虚地址，各需多少时间？给出计算过程。
- （2）基于上述访问序列，虚地址1565H的物理地址是多少？请说明理由　09

考研题4-3

- （1）

  2362H的访问时间为10+100+100=210ns

  1565H的访问时间为10+100+100000000

+10+100=100000220ns

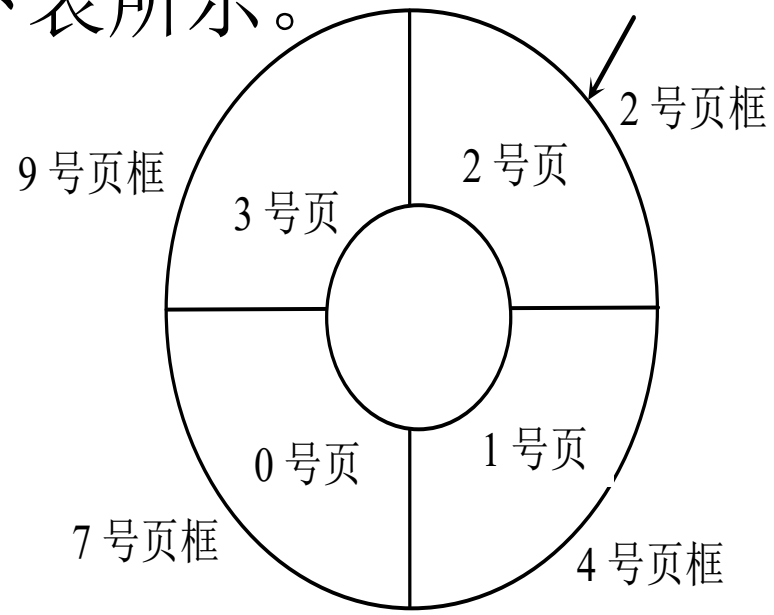
  25A5H的访问时间为10+100=110ns

- （2）1565H的物理地址是：101565H，因为2号页面刚被访问，不会被置换，因此用101H页框，物理地址为101565H

考研题5-1

- 设某计算机的逻辑地址空间和物理地址空间均为64KB，按字节编址。若某进程最多需要6页数据存储空间，页的大小为1KB。操作系统采用固定分配局部置换策略为此进程分配4个页框（Page frame），如下表所示。

页号	页框号	装入时刻	访问位
0	7	130	1
1	4	230	1
2	2	200	1
3	9	160	1



考研题5-2

- 当该进程执行到260时刻时，要访问逻辑地址为17CAH的数据，请回答下列问题：
- （1）该逻辑地址对应的页号是多少？
- （2）若采用先进先出（FIFO）置换算法，该逻辑地址对应的物理地址是多少？要求给出计算过程。
- （3）若采用时钟（CLOCK）置换算法，该逻辑地址对应的物理地址是多少？要求给出计算过程。（设搜索下一页的指针沿顺时针方向移动，且当前指向2号页框，如图所示）。
- 注：本题为2010年全国考研题

考研题5 -3

- 解：（1）因为17ACH=(0001 0111 1100 1010)$_2$，由于采用固定分配局部置换策略，所以该进程只能占用4个页框。页大小为1KB=2^{10}B，所以页内偏移量为10位，于是前6位为页号，对应的页号为5（2分）。

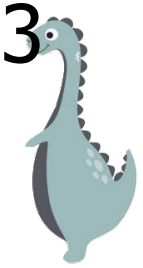
考研题5-4

- （2）页面走向是：0，3，2，1，5。采用FIFO置换算法时的页面置换情况如下表（需要替换装入时间最早的页面），从中看到被置换的页面所在的页框为7，所以17ACH对应的物理地址为(000111 11 1100 1010)2=1FCAH（3分）。

页面走向	0	3	2	1	5
物理块2			2	2	2
物理块4				1	1
物理块7	0	0	0	0	5
物理块9		3	3	3	3
缺页否	√	√	√	√	√

考研题5-5

- 根据CLOCK算法，如果当前指针所指页框的使用位为0，则替换该页；否则将使用位清零，并将指针指向下一个页框，继续查找。
- 根据题设和示意图，将从2号页框开始，前4次查找页框的顺序为2→4→7→9，并将对应页框的使用位清零。在第5次查找中，指针指向2号页框，因2号页框的使用位为0，故淘汰2号页框对应的2号页面，把5号页面装入2号页框中，并将对应使用位设置为1，所以对应的物理地址为00010 11 1100101 0B，换算成十六进制为0BCAH（3分）。



考研题6

- 在缺页处理过程中，操作系统执行的操作可能是_____。11

  I、修改页表 II、磁盘I/O III、分配页框

  A、仅I、II　　B、仅II

  C、仅III　　D、I、II和III

- 当系统发生抖动时，可以采取的有效措施有_____。11

  I、撤销部分进程 II、增加磁盘交换区的容量

  III、提高用户进程的优先级

  A、仅I　　B、仅II

  C、仅III　　D、仅I、II

考研题7

- 在虚拟内存管理中，地址变换机构将逻辑地址变换为物理地址，形成该逻辑地址的阶段是_____。11

A、编辑　B、编译　C、链接　D、装载

- 下列关于虚存的叙述中，正确的是___。12

A. 虚存只能基于连续分配技术

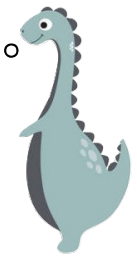
B. 虚存只能基于非连续分配技术

C. 虚存容量只受外存容量的限制

D. 虚存容量只受内存容量的限制

考研题8-1

- 某请求分页系统的页面置换策略如下：从0时刻开始扫描，每隔5个时间单位扫描一轮驻留集（扫描时间忽略不计）且在本轮没有被访问过的页框将被系统回收，并放入到空闲页框链尾，其中内容暂时不清空，当发生缺页时，如果该页曾被使用过且还在空闲页框链表中，则将其重新放回进程的驻留集中；否则，从空闲页框链表头部取出一个页框。
- 忽略其他进程的影响和系统开销。初始时进程驻留集为空。目前系统空闲页的页框号依次为32、15、21、41。进程P依次访问的<虚拟页号,访问时刻>为<1,1>、<3,2>、<0,4>、<0,6>、<1,11>、<0,13>、<2,14>。请回答以下问题：



考研题8-2

- （1）当虚拟页为<0,4>时，对应的页框号是什么？
- （2）当虚拟页为<1,11>时，对应的页框号是什么？说明理由。
- （3）当虚拟页为<2,14>时，对应的页框号是什么？说明理由。
- （4）这种方法是否适合于时间局部性好的程序？说明理由。

考研题8-3

- （1）页框号为21。因为起始驻留集为空，而0页对应的页框为空闲链表中的第三个空闲页框（21），其对应的页框号为21。
- （2）页框号为32。因为11>10故发生第三轮扫描，页号为1的页框在第二轮已处于空闲页框链表中，此刻该页又被重新访问，因此应被重新放回到驻留集中，其页框号为32。
- （3）页框号为41。因为第2页从来没有被访问过，它不在驻留集中，因此从空闲链表中取出链表头的页框41，页框号为41。
- （4）适合。如果程序的时间局部性越好，从空闲页框链表中重新取回的机会越大，该策略的优势越明显。