操作系统
Operating System

汤臣薇、冯文韬

tangchenwei@scu.edu.cn, Wtfeng2021@scu.edu.cn

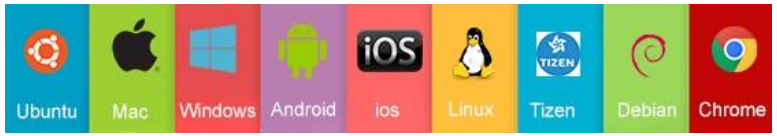
四川大学计算机学院（软件学院）

数据智能与计算艺术实验室

第四章

存储器管理

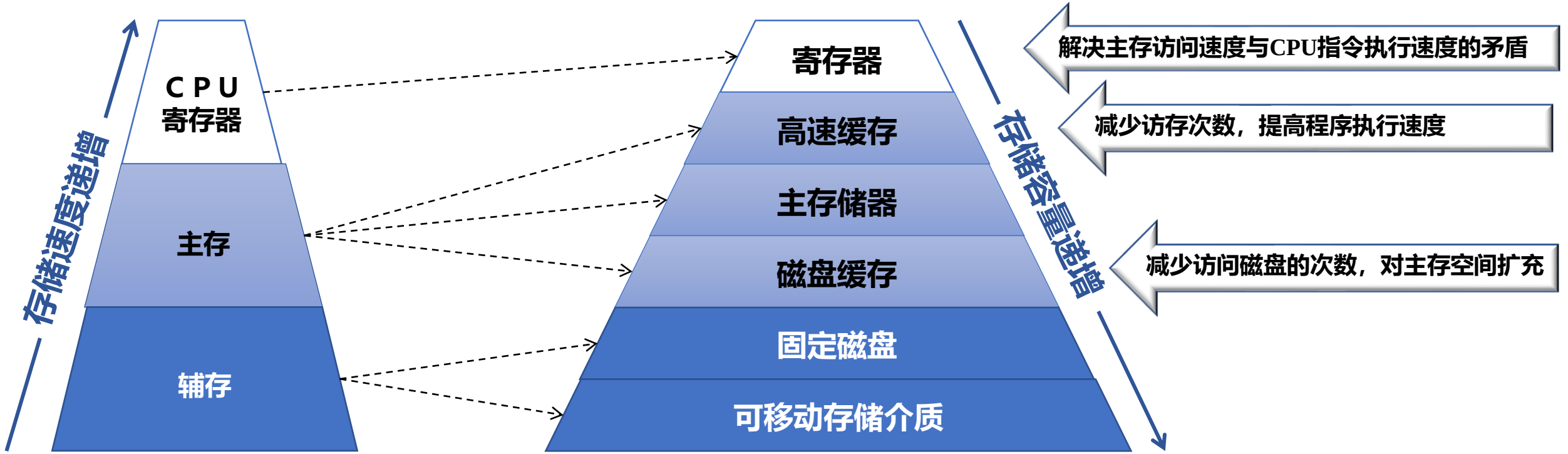
存储器管理



- 存储器的层次结构
- 程序的装入和链接
- 连续分配存储管理方式
- 对换（Swapping）
- 分页存储管理方式 
- 分段存储管理方式

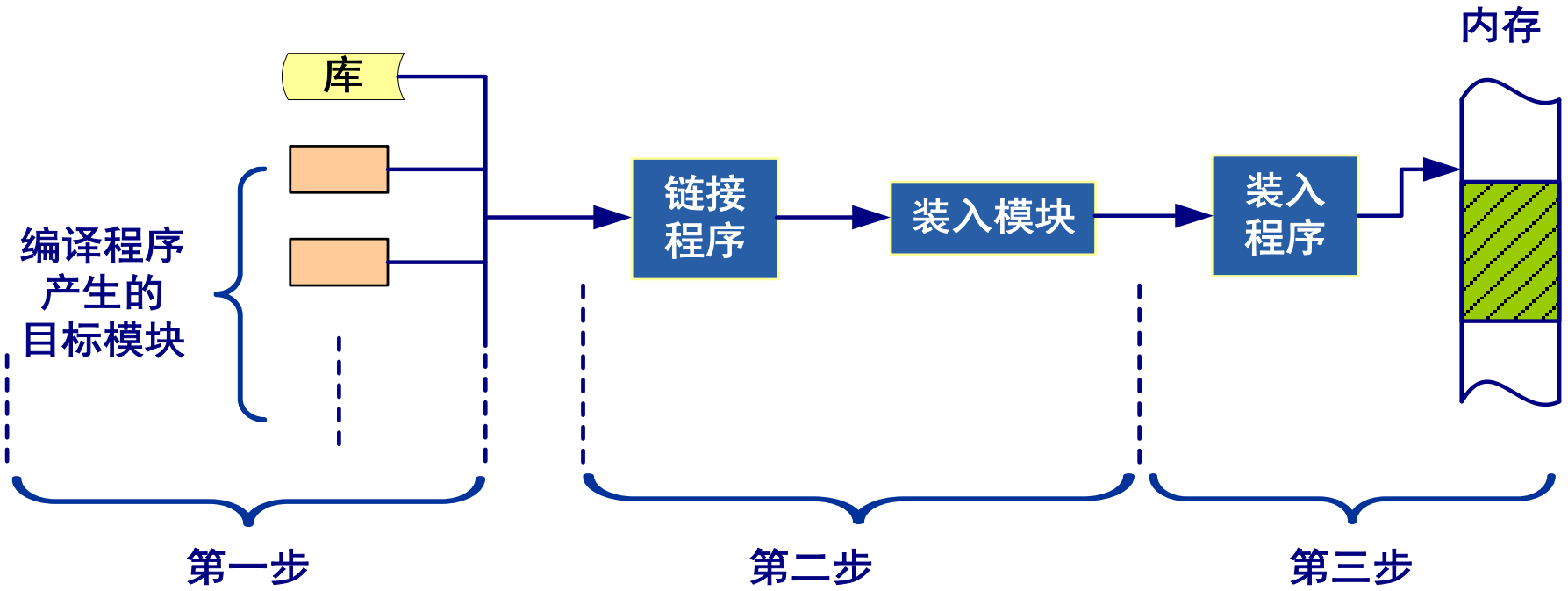
回顾——多层结构的存储器系统

- 对于通用计算机而言，存储层次至少应具有三级：最高层为ＣＰＵ寄存器，中间为主存，最底层是辅存
- 高档的计算机中，根据具体的功能细分为寄存器、高速缓存、主存储器、磁盘缓存、固定磁盘、可移动存储介质等６层
- 在存储器层次中，层次越高（越靠近ＣＰＵ），存储介质的访问速度越快，价格也越高，相对所配置的存储容量也越小。
- 可执行存储器：**寄存器 + 主存储器**
- 寄存器、高速缓存、主存储器和磁盘缓存均属于操作系统**存储管理**的管辖范畴，掉电后，它们中存储的信息不再存在。
- 低层的固定磁盘和可移动存储介质则属于**设备管理**的管理范畴，它们存储的信息将被长期保存。



回顾——程序的装入和链接

- **如何将一个用户源程序变成一个可以在内存中执行的程序?**
  - **编译：**由编译程序将用户源代码译成若干个**目标模块**。
  - **链接：**由链接程序将编译后的目标模块及所需要的库函数链接在一起，形成一个**装入模块**。
  - **装入：**由装入程序将装入模块装入内存。



回顾——程序的装入

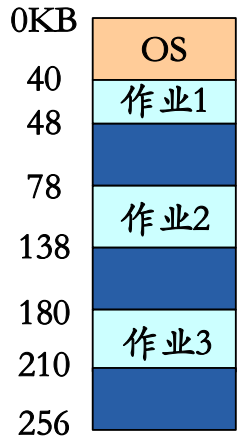
- **单个目标模块的装入过程可采用三种方式**
  - **绝对装入方式（Absolute Loading Mode）**
    - ✓绝对装入程序按照装入模块中的地址，将程序和数据装入内存
    - ✓装入模块被装入内存后，程序中的逻辑地址与实际内存地址完全相同，不须对程序和数据的地址进行修改
    - ✓适合单道程序环境
  - **可重定位方式**
  - **动态运行时的装入方式**

} **地址映射/地址重定位**

把用户程序装入内存时对有关指令的地址部分的修改，定义为从程序地址到内存地址的地址映射/重定位，有静态地址映射和动态地址映射

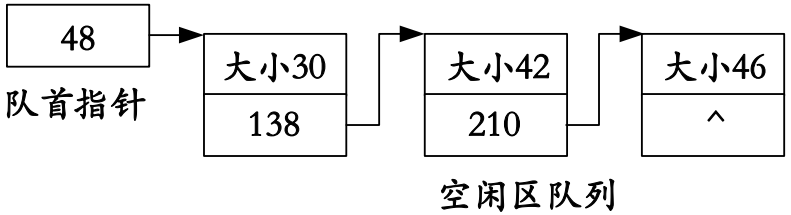
回顾——连续分配存储管理方式

- **连续分配：是指为一个用户程序分配一个连续的内存空间。**
- **连续分配方式可分为四类：**
  - **单一连续分配；**
  - **固定分区分配；**
  - **动态分区分配；**
    - **常用的有空闲分区表和空闲分区链**
      - **空闲分区表：每个空闲分区占一个表目，表目中包括分区序号、分区始址及分区的大小等数据项**
      - **空闲分区链：实现对空闲分区的分配和链接，在每个分区的起始部分，设置一些用于控制分区分配的信息，以及用于链接各分区所用的前向指针；在分区尾部则设置一后向指针，通过前、后向链接指针，可将所有的空闲分区链接成一个双向链**
  - **动态可重定位分区分配算法**



大小	位置
30	48
42	138
46	210
…	…

空闲区表

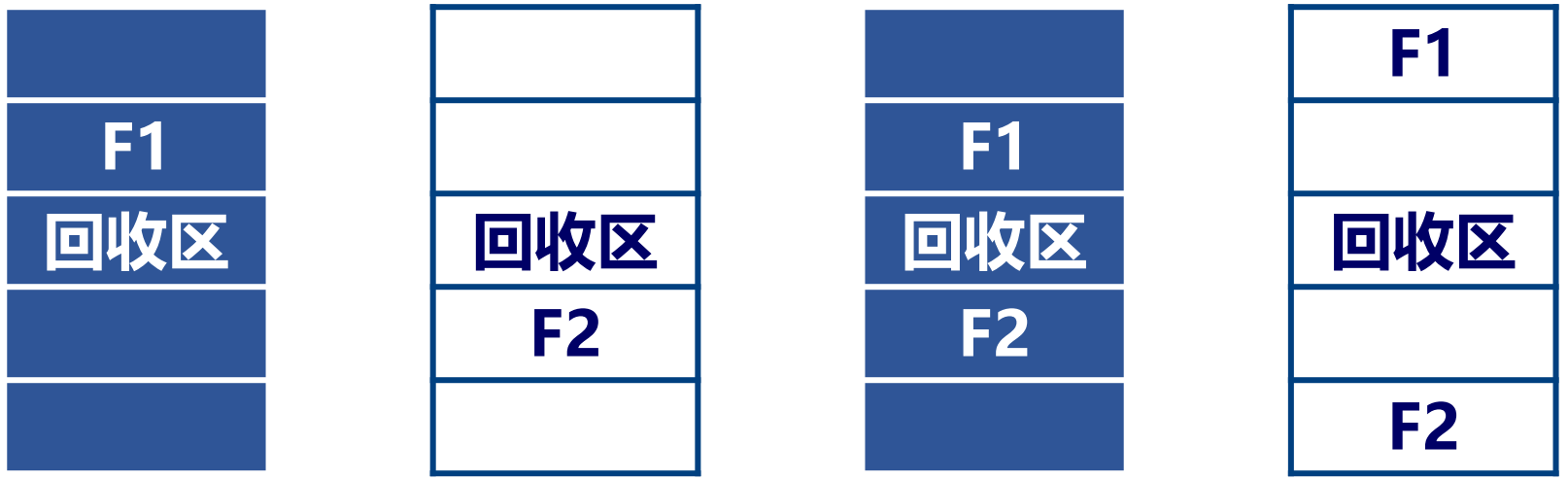


空闲区队列

回顾——动态/可变分区分配

■ 内存回收

- 回收区与插入点的前一个空闲分区 F1相邻接
- 回收分区与插入点的后一空闲分区 F2相邻接
- 回收区同时与插入点的前、后两个分区邻接
- 回收区既不与 F1邻接，又不与 F2邻接



回顾——动态/可变分区分配

- **动态分区分配算法**
  - **基于顺序搜索的动态分区分配算法**
    - ✓首次适应（first fit, FF）算法
    - ✓循环首次适应（next fit, NF）算法
    - ✓最佳适应（best fit, BF）算法
    - ✓最坏适应（worst fit, WF）算法
  - **基于索引搜索的动态分区分配算法**
    - ✓快速适应（quick fit）算法
    - ✓伙伴系统（buddy system）
    - ✓哈希算法

需要关注的问题：

1）算法的原理、优点、缺点；
2）空闲表/队列排列方式：
大小、地址；递增、递减

回顾——基于顺序搜索的动态分区分配算法

■ 首次适应（first fit, FF）算法

分配

- 要求空闲分区链以**地址递增**的次序链接
- 当进程申请大小为SIZE的内存时，系统从空闲区表的第一个表目开始查询，直到**首次找到等于或大于**SIZE的空闲区；
- 从该区中划出大小为SIZE的分区分配给进程；
- 余下的部分仍作为一个空闲区留在空闲区表中，但要**修改其首址和大小**。

回顾——基于顺序搜索的动态分区分配算法

■ 循环首次适应（next fit, NF）算法

分配

- 在为进程分配内存空间时，从上次找到的空闲分区的**下一个空闲分区**开始查找，直至找到一个能满足要求的空闲分区，从中划出一块与请求大小相等的内存空间分配给作业。
- 设置**起始查寻指针**，用于指示下一次起始查寻的空闲分区。
- 采用**循环查找**方式，即如果最后一个(链尾)空闲分区的大小仍不能满足要求，则应返回到第一个空闲分区，比较其大小是否满足要求。

回顾——基于顺序搜索的动态分区分配算法

■ 最佳适应（best fit, BF）算法

分配

- 当进程申请一个存储区时，系统从表头开始查找，当找到第一个满足要求的空闲区时，停止查找，并且这个空闲区是最佳的空闲区。
- 所谓“最佳”是指每次为作业分配内存时，总是把能满足要求、又是最小的空闲分区分配给作业，避免“大材小用”。

排序

- 将所有的空闲分区按其容量以从小到大的顺序形成一空闲分区链。

回顾——基于顺序搜索的动态分区分配算法

■ 最坏适应（worst fit, WF）算法

分配

- 扫描整个空闲分区表或链表，总是挑选一个**最大的空闲区**分割给作业使用
- 空闲分区按容量**从大到小**的顺序形成一空闲分区链
- 查找时只要检查空闲区表的**第一个空闲区**的大小是否大于或等于SIZE；若空闲区小于SIZE，则分配失败；否则从空闲区中分配SIZE的存储区给用户，修改和调整空闲区表。

回收

- 按释放区的首址，查询空闲区表（队列），若有与释放区相邻的空闲区，合并到相邻的空闲区中，修改该区的大小和首址；否则把释放区作为空闲区插入空闲区表（队列）；
- 分配和回收后要对空闲区表（队列）重新排序。

回顾——基于索引搜索的动态分区分配算法

■ 快速适应（quick fit）算法 = 分类搜索法

分配

- 将空闲分区根据其容量大小进行分类，对于每一类具有相同容量的所有空闲分区，单独设立一个空闲分区链表，系统中存在多个空闲分区链表
- 在内存中设立一张管理索引表，表的每一个表项对应了一种空闲分区类型， 并记录了该类型空闲分区链表表头的指针
- 空闲分区分类根据进程常用的空间大小进行划分

回顾——基于索引搜索的动态分区分配算法

■ 伙伴系统（buddy system）

分配

- 无论已分配分区或空闲分区，其大小均为 2 的 k 次幂，k 为整数，l≤k≤m。假设系统的可利用空间容量为 2^m个字，则系统开始运行时，整个内存区是一个大小为 2^m 的空闲分区。
- 在系统运行过程中，不断的划分会形成若干个不连续的空闲分区，将这些空闲分区根据分区的大小进行分类，对于每一类具有相同大小的所有空闲分区，单独设立一个空闲分区双向链表。
- 当需要为进程分配一个长度为 n 的存储空间时，首先计算一个 i 值，使 2^(i - 1)<n≤2^i，然后在空闲分区大小为 2i 的空闲分区链表中查找。若2i 的空闲分区已耗尽，则在2i + 1空闲分区链表中寻找。
- 若存在 2i + 1的空闲分区，则把该空闲分区分为相等的两个分区，这两个分区称为一对伙伴，其中的一个分区用于分配，而把另一个加入分区大小为 2i 的空闲分区链表中。

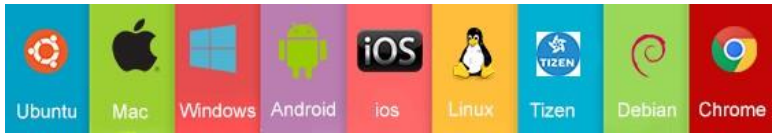
回顾——基于索引搜索的动态分区分配算法

■ 哈希算法

分配

- 哈希算法利用哈希快速查找的优点，以及空闲分区在可利用空闲区表中的分布规律，建立哈希函数，构造一张以空闲分区大小为关键字的哈希表，该表的每一个表项纪录了一个对应的空闲分区链表表头指针。
- 根据所需空闲分区大小，通过哈希函数计算得到在哈希表中的位置，从中得到相应的空闲分区链表，实现最佳分配策略。

存储器管理



- 存储器的层次结构
- 程序的装入和链接
- 连续分配存储管理方式
- 对换（Swapping）
- 分页存储管理方式 
- 分段存储管理方式

4.3.4 动态分区分配算法

■ 碎片问题

- 由于空闲区的大小与申请内存的大小相等的情况是很少的，绝大多数情况是从一个空闲区中切去一块，剩下的部分作为一个空闲区仍留在空闲区表中，随着时间的推移，空闲区的发展趋势是越来越小，直至不能满足任何用户要求。这种不能被任何用户使用的极小的空闲区称为**碎片**。碎片的出现造成了存储空间的浪费。

■ 解决策略

- **规定门限值**（由操作系统规定，如1K），分割空闲区时，若剩余部分小于门限值，则不再分割此空闲区。
- **定期压缩存储空间**，将所有空闲区集中到内存的一端，但这种方法的系统开销太大。

4.3.6 动态可重定位分区分配

■ 紧凑

- 在连续分配方式中，必须把一个系统或用户程序装入一连续内存空间。如果在系统中只有若干个小的分区，即使它们容量的总和大于要装入的程序，但由于分区不相邻接，无法把程序装入内存
- “零头”或“碎片”：不能被利用的小分区
- 将内存中的所有作业进行移动，使它们全都相邻接→把原来分散的多个小分区拼接成一个大分区
- “拼接”或“紧凑”：通过移动内存中作业的位置，以把原来多个分散的小分区拼接成一个大分区的方法
- 在每次“紧凑”后，都必须对移动了的程序或数据进行重定位

4.3.6 动态可重定位分区分配

■ 紧凑

➢ 通过移动内存中作业的位置，以把原来多个分散的小分区拼接成一个大分区的方法

4.3.6 动态可重定位分区分配

■ 动态重定位

- 在**动态运行时装入**的方式中，将**相对地址转换为物理地址**的工作，被**推迟到程序指令要真正执行时进行**。
- 为使地址的转换不会影响到指令的执行速度，在系统中增设一个**重定位寄存器**，用它来存放程序(数据)在内存中的**起始地址。**
- 程序在执行时，真正访问的内存地址是相对地址与重定位寄存器中的地址相加而形成的。
- **动态重定位：**地址变换过程在程序执行期间，随着对每条指令或数据的访问自动进行。当系统对内存进行了“紧凑”而使若干程序从内存的某处移至另一处时，**不需对程序做任何修改**，只要用该程序在内存的**新起始地址，去置换原来的起始地址**即可。

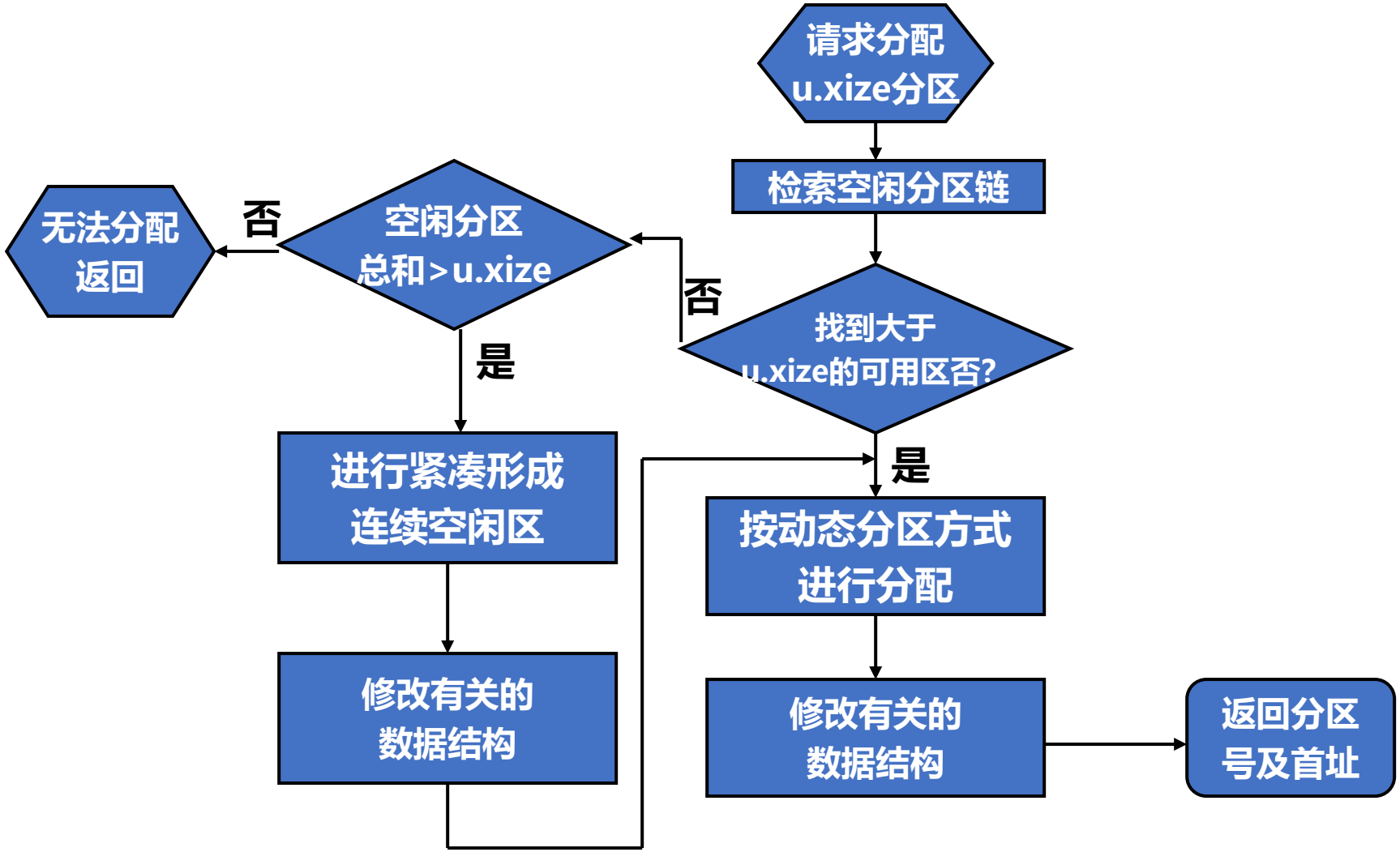
4.3.6 动态可重定位分区分配

■ 动态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**紧凑**的功能。
- 当该算法不能找到一个足够大的空闲分区以满足用户需求时，如果所有小的空闲分区的**容量总和大于用户的要求**，这是便须对内存进行“紧凑”，将经紧凑后多得到的大空闲分区分配给用户。
- 如果所有小空闲分区的容量总和仍小于用户的需求，则返回分配失败信息。

4.3.6 动态可重定位分区分配

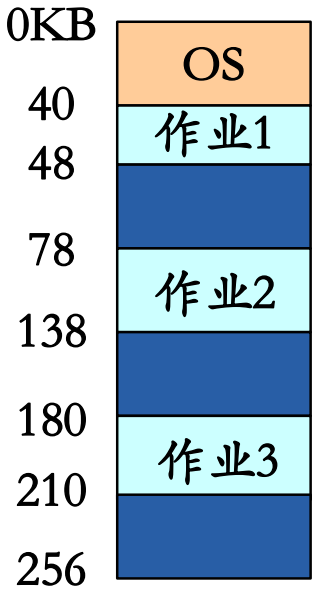
■ 动态重定位分区分配算法



早期大型机使用的内存管理方式

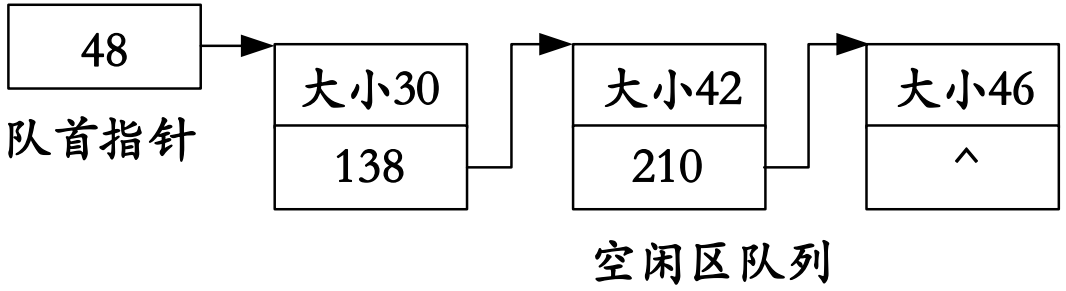
少数掌上电脑和嵌入式系统使用的内存管理方式

早期PC使用的内存管理方式（MS-DOS）



大小	位置
30	48
42	138
46	210
…	…

空闲区表



空闲区队列

紧凑前

紧凑后

2018 考研

3.某基于动态分区存储管理的计算机，其主存的容量为 55MB，这些空间在初始为空闲。采用最佳分配算法，分配和释放的顺序分别为: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?

A 7MB　　B 9MB　　C 10MB　　D 15MB

7.在动态分区存储系统中，空闲表的内容如下:

空闲块号	1	2	3	4
块大小	80	75	55	90
块的基址	60	150		350

此时，进程 P 请求 50KB 内存，系统从第 1 个空闲块开始查找，结果把第 4 个空闲块分配给了进程 P。请问系统是采用哪种分区分配算法实现这一方案?

250

A 首次适应法　　B 最佳适应法

C 最差适应法　　D 下次适应法

- 已知主存有256KB容量，其中OS占用低址20KB，有这样的一个作业序列：

作业1要求 80KB

作业2要求16KB

作业3要求140KB

作业1完成

作业3完成

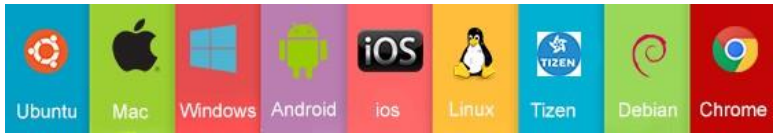
作业4要求 80KB

作业5要求120KB

试用首次适应算法和最佳适应算法分别处理上述作业序列(在存储分配时，从空白区高址处分割作为已分配区)，并完成以下各步：

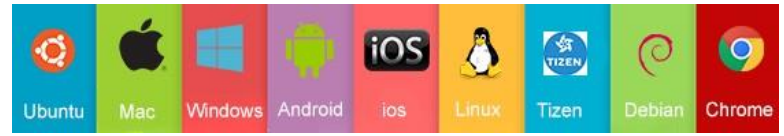
- 画出作业1、2、3进入主存后，主存的分配情况；
- 作业1、3完成后，画出主存分配情况；
- 画出两种算法中空白区的分区描述器信息(假定分区描述器所需占用字节数已包含在作业所要求的主存容量中)及空白区链接情况；
- 哪种算法对该作业序列而言是适合的?

存储器管理



- 存储器的层次结构
- 程序的装入和链接
- 连续分配存储管理方式
- 对换（Swapping）
- 分页存储管理方式 
- 分段存储管理方式

存储器管理



对换（Swapping）

- 多道程序环境下兑换技术
- 对换空间的管理
- 进程的换出与换入

4.4 对换（Swapping）

- 对换技术也称为交换技术，最早用于MIT的单用户分时系统CTSS中。
- 由于当时计算机的内存都非常小，为了使该系统能分时运行多个用户程序而引入了对换技术。
- 系统把所有的用户作业存放在磁盘上，每次只能调入一个作业进入内存，当该作业的一个时间片用完时，将它调至外存的后备队列上等待，再从后备队列上将另一个作业调入内存。
- 这就是最早出现的分时系统中所用的对换技术。现在已经很少使用。

4.4.1 多道程序环境下的兑换技术

■ 对换的引入

- 内存中的某些进程由于某事件尚未发生而被阻塞运行，但它却占用了大量的内存空间，甚至有时可能出现在内存中所有进程都被阻塞，而无可运行之进程，迫使CPU停止下来等待的情况；
- 同时，有许多作业，因内存空间不足，一直驻留在外存上，而不能进入内存运行。
- 系统资源的严重浪费，且使系统吞吐量下降。

■ 对换的类型：将一定数量的程序/数据换入换出内存

- 整体对换：对换以整个进程为单位
- 页面(分段)/部分对换：对换以页或段为单位

4.4.2 对换空间的管理

■ 对换空间管理的主要目标

- 在具有对换功能的OS中，把磁盘空间分为**文件区和对换区**
  - ✓ **文件区**：存放各类文件
  - ✓ **对换区**：存放从内存换出的进程
- 对**文件区**管理的主要目标
  - ✓ 提高文件存储空间的利用率 > 提高对文件的访问速度
- 对**对换区**管理的主要目标
  - ✓ 提高进程换入和换出的速度 > 提高文件存储空间的利用率

4.4.2 对换空间的管理

■ 对换区空闲盘块管理中的数据结构

- 实现对对换区中的空闲盘块的管理
- 用于记录外存对换区中的空闲盘块的使用情况
- 数据结构的形式与内存在动态分区分配方式中所用数据结构相似，可以用空闲分区表或空闲分区链。
- 在空闲分区表的每个表目中，应包含两项：对换区的首址及其大小，分别用盘块号和盘块数表示。

4.4.2 对换空间的管理

■ 对换空间的分配与回收

- 由于对换分区的分配采用的是连续分配方式，因而对换空间的分配与回收与动态分区方式时的内存分配与回收方法雷同。
- 分配算法：首次适应算法、循环首次适应算法或最佳适应算法等。
- 回收操作同样为四种情况：
  - ✓ 回收区与插入点的前一个空闲分区 F1相邻接
  - ✓ 回收分区与插入点的后一空闲分区 F2相邻接
  - ✓ 回收区同时与插入点的前、后两个分区邻接
  - ✓ 回收区既不与 F1邻接，又不与 F2邻接

4.4.3 进程的换出与换入

■ 进程的换出

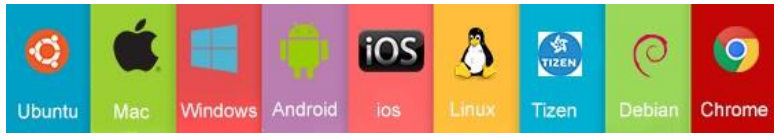
- 对换进程在实现进程换出时，是将内存中的某些进程调出至对换区，以便腾出内存空间。
- 换出过程可分为以下两步：
  - ✓ **选择被换出的进程：**状态——阻塞或睡眠；优先级最低；还需考虑在内存中的驻留时间；
  - ✓ **进程换出过程：**非共享的程序或数据直接换出；共享的程序或数据换出前必须判断引用计数减1后是否为0,不为0,则不能换出；可以换出时，在申请对换空间成功后换出，同时释放内存。

4.4.3 进程的换出与换入

■ 进程的换入

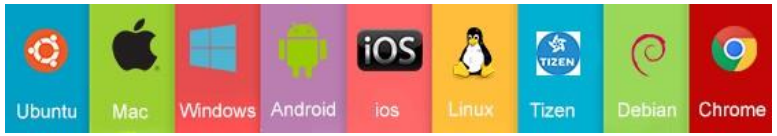
- 对换进程将定时执行换入操作
- 首先查看PCB集合中所有进程的状态，从中找出“就绪”状态但已换出的进程。
- 当有许多这样的进程时，选择其中已换出到磁盘上时间最久（必须大于规定时间，如2 s）的进程作为换入进程，为它申请内存。
- 如果申请成功，可直接将进程从外存调入内存；
- 如果失败，则需先将内存中的某些进程换出，腾出足够的内存空间后，再将进程调入。

存储器管理



- 存储器的层次结构
- 程序的装入和链接
- 连续分配存储管理方式
- 对换（Swapping）
- 分页存储管理方式 
- 分段存储管理方式

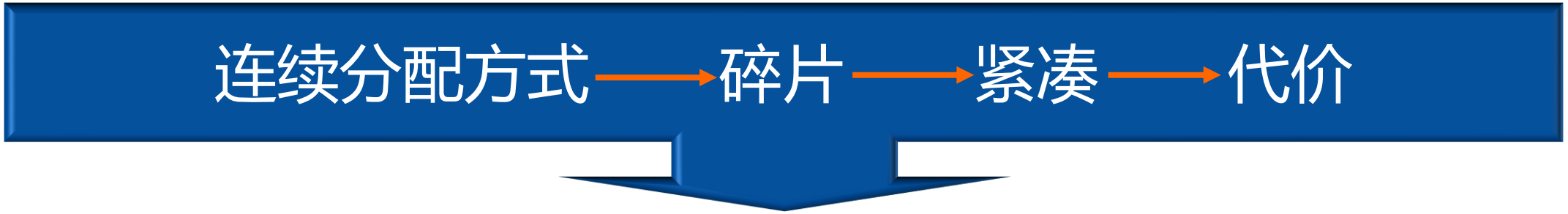
存储器管理



分页存储管理方式

- 分页存储管理的基本方法
- 地址变换机构
- 访问内存的有效时间
- 两级和多级页表
- 反置页表

4.5 分页存储管理方式



离散分配方式

分页　　分段　　段页式

基本的分页存储管理方式（纯分页存储管理方式）

分页技术是由曼彻斯特大学提出，于1960年前后在Atlas计算机上实现。

4.5.1 分页存储管理的基本方法

页面

- 分页存储管理将进程的逻辑地址空间分成若干个页，并为页面加以编号，从0开始编制页号，如第0页、第1页等，页内地址是相对于0编址。
- 相应地，把内存的物理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)，也同样为它们加以编号，如 0#块、1#块等。
- 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。进程最后一页装不满一块而形成不可利用的碎片——页内碎片
- 以页为单位进行内存分配，按作业的页数多少来分配；逻辑上相邻的页，物理上不一定相邻

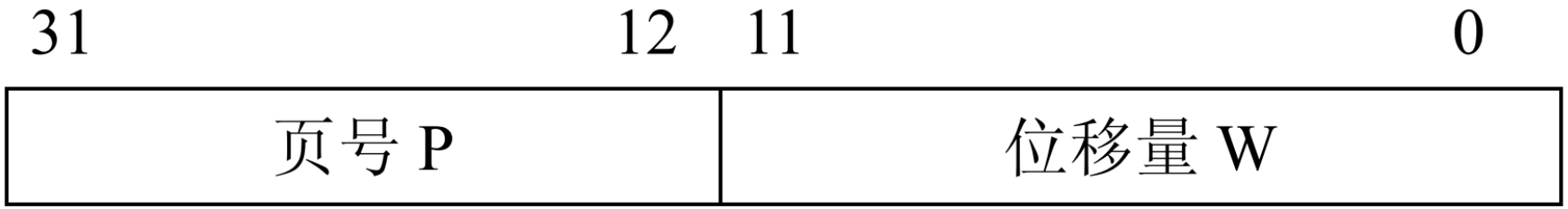
页面大小

- 在分页系统中的页面其大小应适中，且页面大小应是 2 的幂，通常为1 KB～8 KB
- 页面若太小，一方面虽然可使内存碎片减小，从而减少了内存碎片的总空间，有利于提高内存利用率，但另一方面也会使每个进程占用较多的页面，从而导致进程的页表过长，占用大量内存；此外，还会降低页面换进换出的效率。
- 页面较大，可以减少页表长度，提高页面换进换出速度，但又会使页内碎片增大。

4.5.1 分页存储管理的基本方法

■ 地址结构

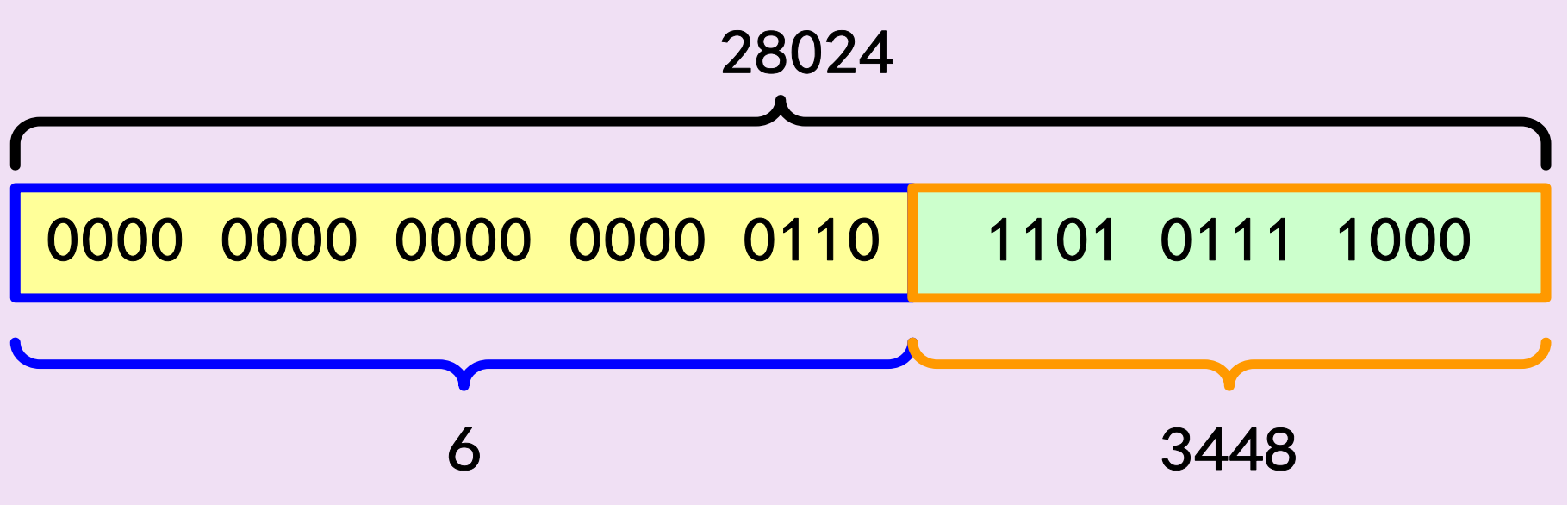
- 高位部分为**页号 P**，低位部分为**位移量 W(或称为页内地址)**。
- 图中的地址长度为 32 位，其中 0～11 位为页内地址，即每页的大小为 4 KB；12～31 位为页号，地址空间最多允许有 1 M 页。



4.5.1 分页存储管理的基本方法

■ 页地址映射

- **逻辑地址A；页大小L；页号：P = INT（A/L）页内地址：d = A mod L**
  - A=28024，L=4KB=4*1024=4096，28024/4096=6...3448，P = 6，d=3448.
- **28024的二进制用32位表示为：**0000 0000 0000 0000 0110 1101 0111 1000**；页面大小为4KB，则取低12位为页内地址，剩余高位是页号。**



4.5.1 分页存储管理的基本方法

■ 页表

- 在分页系统中，允许将进程的各个页离散地存储在内存不同的物理块中，但系统应能保证进程的正确运行，即能在内存中找到每个页面所对应的物理块。
- 系统为每个进程建立了一张页面映像表，简称页表。在进程地址空间内的所有页(0～n)，依次在页表中有一页表项，其中记录了相应页在内存中对应的物理块号
- 进程执行时，通过查找该表，即可找到每页在内存中的物理块号。
- 页表的作用是实现从页号到物理块号的地址映射。

4.5.1 分页存储管理的基本方法

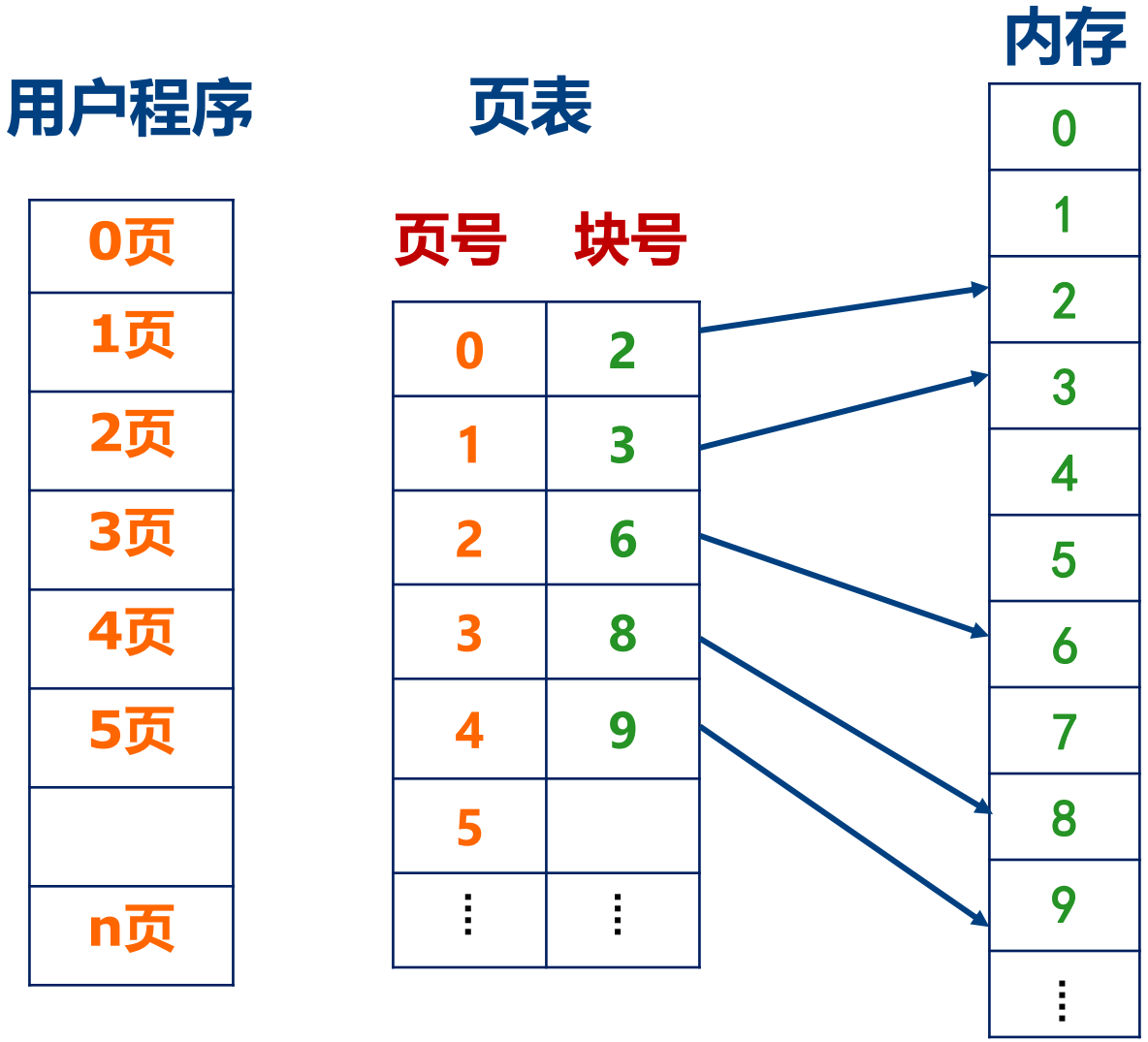
■ 页表

- 页表包含以下几个表项：
  - ✓ 页号：登记程序地址空间的页号
  - ✓ 块号：登记相应的页所对应的内存块号
  - ✓ 其它：登记与存储信息保护有关的信息

页号	块号	其它
0	5	…
1	65	…
2	13	…

4.5.1 分页存储管理的基本方法

- 页表



4.5.2 地址变换机构

■ 地址变换机构

- 将用户地址空间中的逻辑地址变换为内存空间中的物理地址
- 页内地址和物理地址是一一对应的(例如，对于页面大小是 1 KB 的页内地址是 0～1023，其相应的物理块内的地址也是 0～1023，无须再进行转换)
- 地址变换机构的任务：将逻辑地址中的页号，转换为内存中的物理块号。
- 页面映射表的作用就是用于实现从页号到物理块号的变换，因此，地址变换任务是借助于页表来完成的。

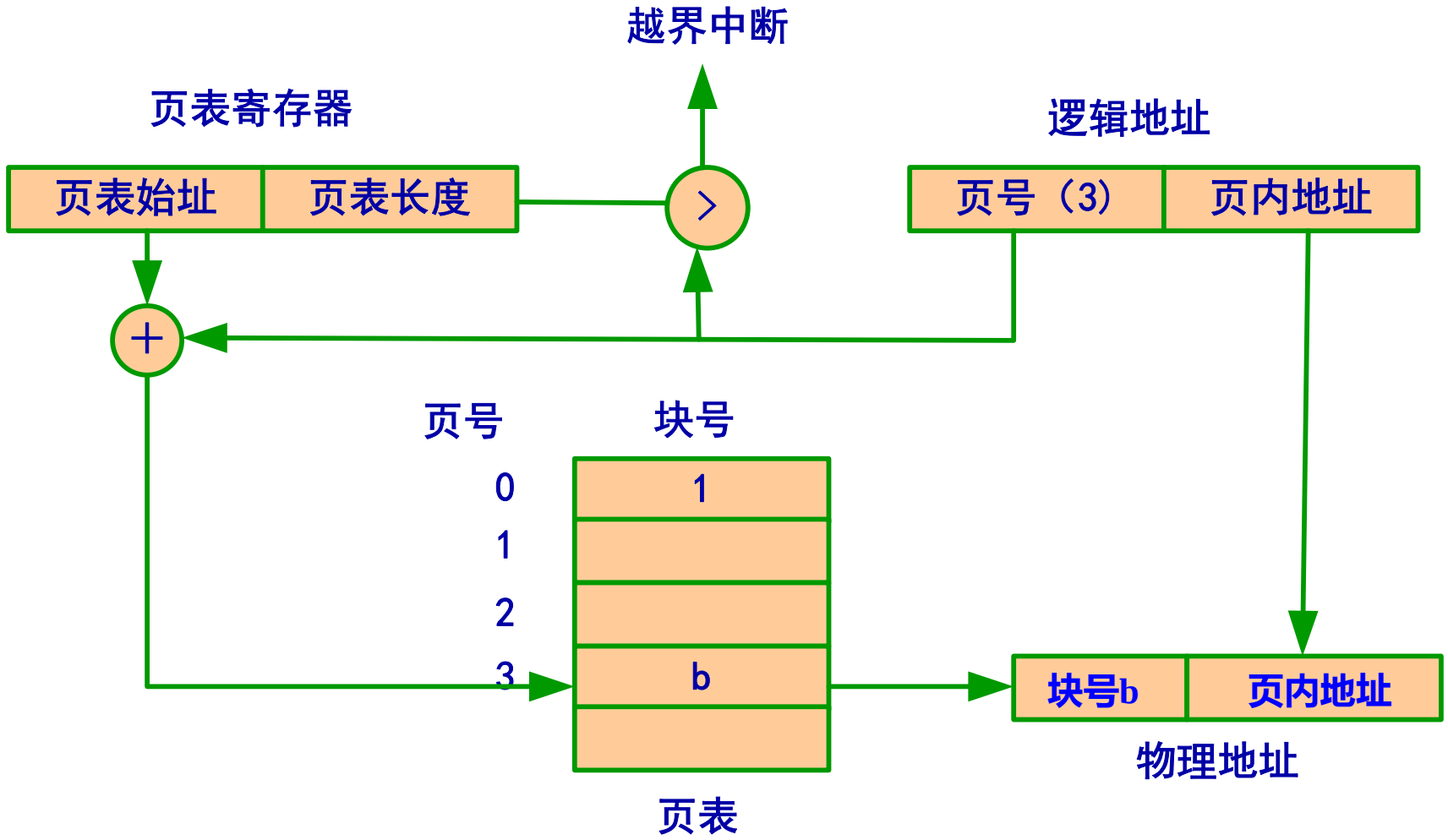
4.5.2 地址变换机构

■ 基本的地址变换机构

- 页表的功能可以由一组专门的寄存器来实现，一个页表项用一个寄存器。页表大多驻留在内存中，在系统中只设置一个页表寄存器 PTR(Page-Table Register)，在其中存放页表在内存的始址和页表的长度。当调度程序调度到某进程时，才将这两个数据装入页表寄存器中。在单处理机环境下，虽然系统中可以运行多个进程，但只需一个页表寄存器。
- 越界保护：在执行检索之前，先将页号与页表长度进行比较，如果页号大于或等于页表长度，则表示本次所访问的地址已超越进程的地址空间，产生地址越界中断。

4.5.2 地址变换机构

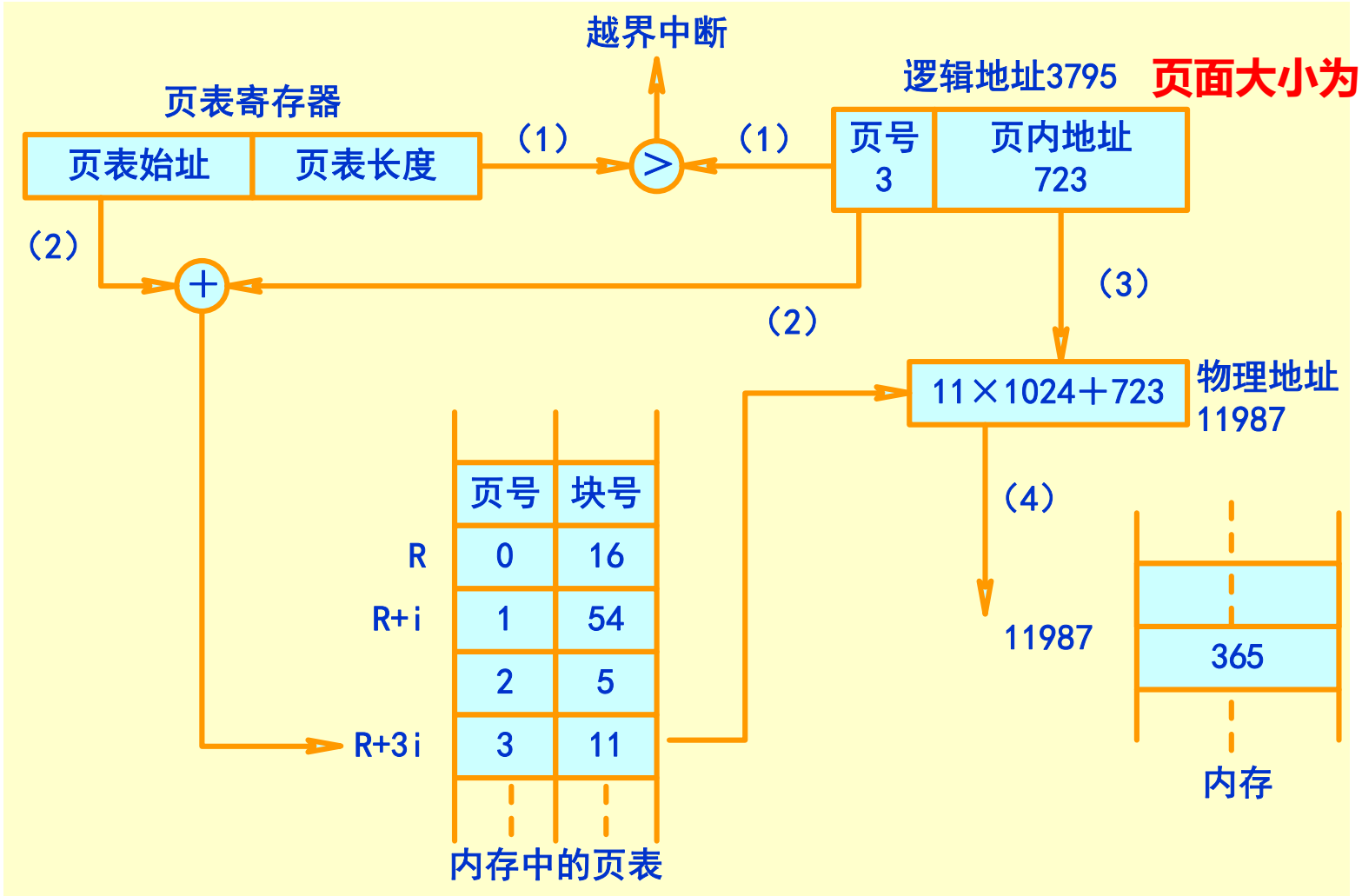
- **基本的地址变换机构**



分页系统地址变换机构

4.5.2 地址变换机构

- 访问内存11987单元，得到需要的数据365



4.5 分页式存储管理-练习

- 某虚拟存储器的用户编程空间共32个页面，每页为1KB，内存为16KB。假定某时刻一用户页表中已调入内存的页面对应的物理块号如下表：

问题：逻辑地址0A5C（H）
所对应的物理地址为 _____？

页号	物理块号
0	5
1	10
2	4
3	7

4.5 分页式存储管理-练习

- 某虚拟存储器的用户编程空间共32个页面，每页为1KB，内存为16KB。假定某时刻一用户页表中已调入内存的页面对应的物理块号如下表：

0A5C＝0000, 1010, 0101, 1100

＝0000, 1010, 0101, 1100

页号为2，对应块号为4，有：

物理地址：0001, 0010, 0101, 1100

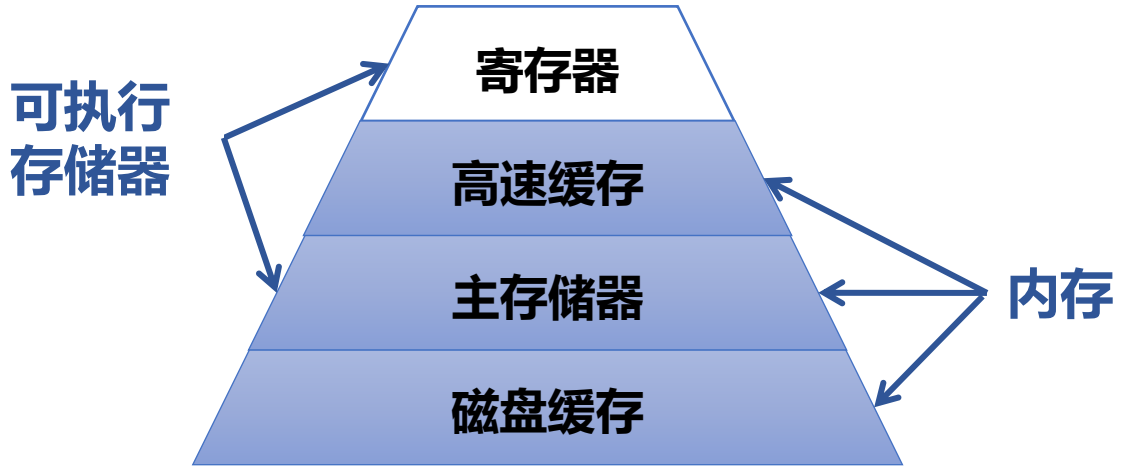
即：125C

页号	物理块号
0	5
1	10
2	4
3	7

4.5.2 地址变换机构

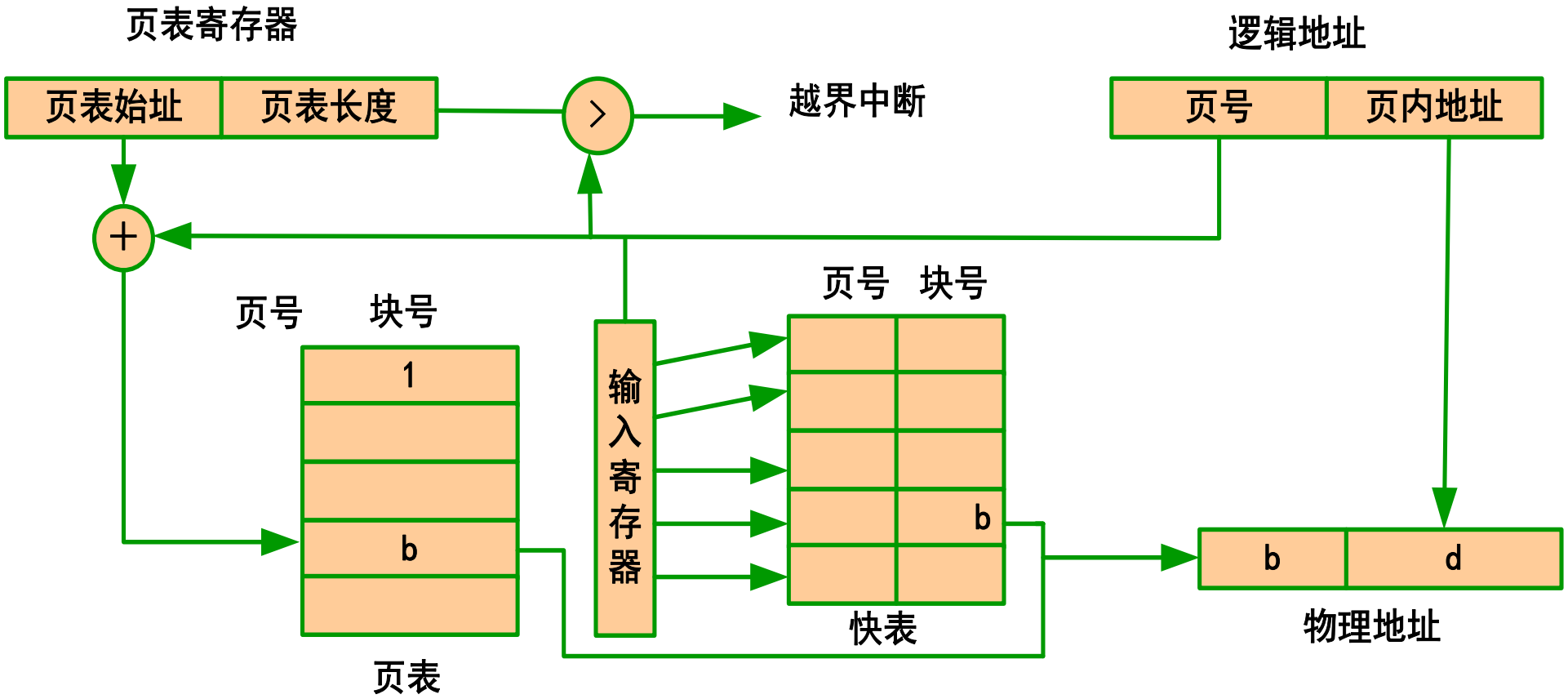
■ 具有快表的地址变换机构

- 由于页表是存放在内存中的，这使 CPU 在每存取一个数据时，都要两次访问内存：第一次是访问页表从中找到指定页的物理块号，再将块号与页内偏移量 W 拼接，以形成物理地址；第二次访问时，才是从第一次所得地址中获得所需数据(或向此地址中写入数据)。
- 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寻能力的特殊高速缓冲寄存器，又称为联想寄存器/快表，用以存放当前访问的那些页表项
- 如果没有快表，则需两次访问内存：第一次：访问页表; 第二次：得到绝对地址内容
- 有快表，速度提高；快表贵，不能太多。



4.5.2 地址变换机构

■ 具有快表的地址变换机构



具有快表的地址变换机构

4.5.3 访问内存的有效时间

■ 内存的有效访问时间

- 进程发出指定逻辑地址的访问请求，经过地址变换，到在内存中找到对应的实际物理地址单元并取出数据，所需花费总时间
- 在引入块表的分页存储管理方式中，可减少一次内存访问时间，但存在找到所需表项的命中率
- 令m为查找快表所需时间，a为命中率，t为内存访问时间，其有效访问时间：

$$EAT = ?$$

4.5.3 访问内存的有效时间

■ 内存的有效访问时间

- 进程发出指定逻辑地址的访问请求，经过地址变换，到在内存中找到对应的实际物理地址单元并取出数据，所需花费总时间
- 在引入块表的分页存储管理方式中，可减少一次内存访问时间，但存在找到所需表项的命中率
- 令m为查找块表所需时间，a为命中率，t为内存访问时加，其有效访问时间：

$$EAT = a*m+(t+m)*(1-a)+t = 2t + m – t * a$$

4.5.3 访问内存的有效时间

■ 内存的有效访问时间

- 举例：有一页式系统，其页表存放在主存中，如果对主存的一次存取需要1.5μs，试问实现一次页面访问的存取时间是多少? 如果系统加有快表，平均命中率为85%，当页表项在快表中时，其查找时间忽略为0，试问此时的存取时间是多少?

- 页表在主存的存取访问时间：

  $=1.5*2=3(\mu s)$

- 增加快表后的存取访问时间：

  $= 2t + m – t * a = 2 * 1.5 + 0 – 1.5 *0.85 = 1.725\ \mu s$

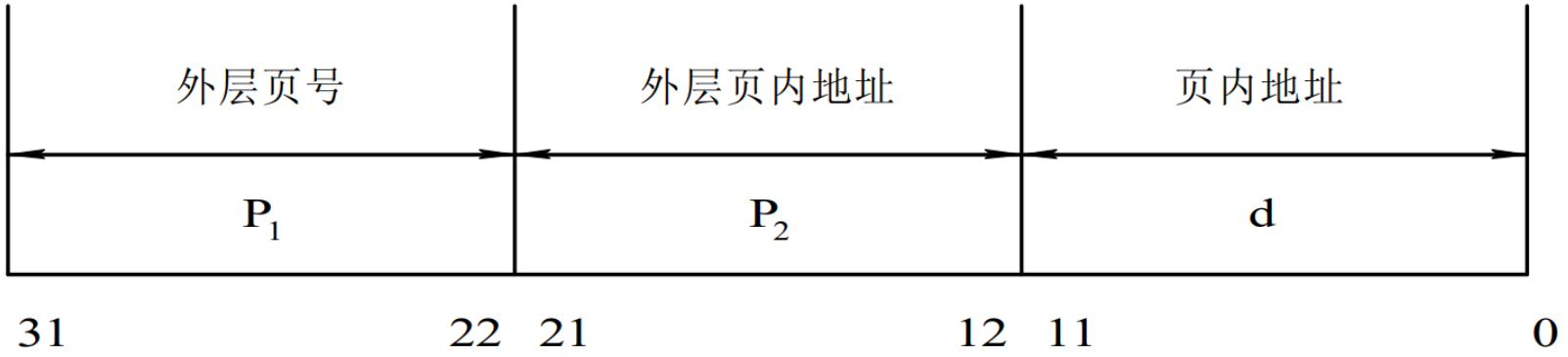
4.5.4 两级和多级页表

- 现代的大多数计算机系统支持非常大的逻辑地址空间(2^32～2^64)。
- 页表要占用相当大的内存空间：如对于一个具有 32 位逻辑地址空间的分页系统，规定页面大小为 4 KB 即 2^12 B，则在每个进程页表中的页表项可达 1 兆多（2^32-2^12=2^20=1MB）。
- 每个页表项占用一个字节，故每个进程仅仅其页表就要占用 1 MB 的内存空间，而且还要求是连续的。
- 可以采用下述两个方法来解决这一问题：
  - 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；
  - 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。

4.5.4 两级和多级页表

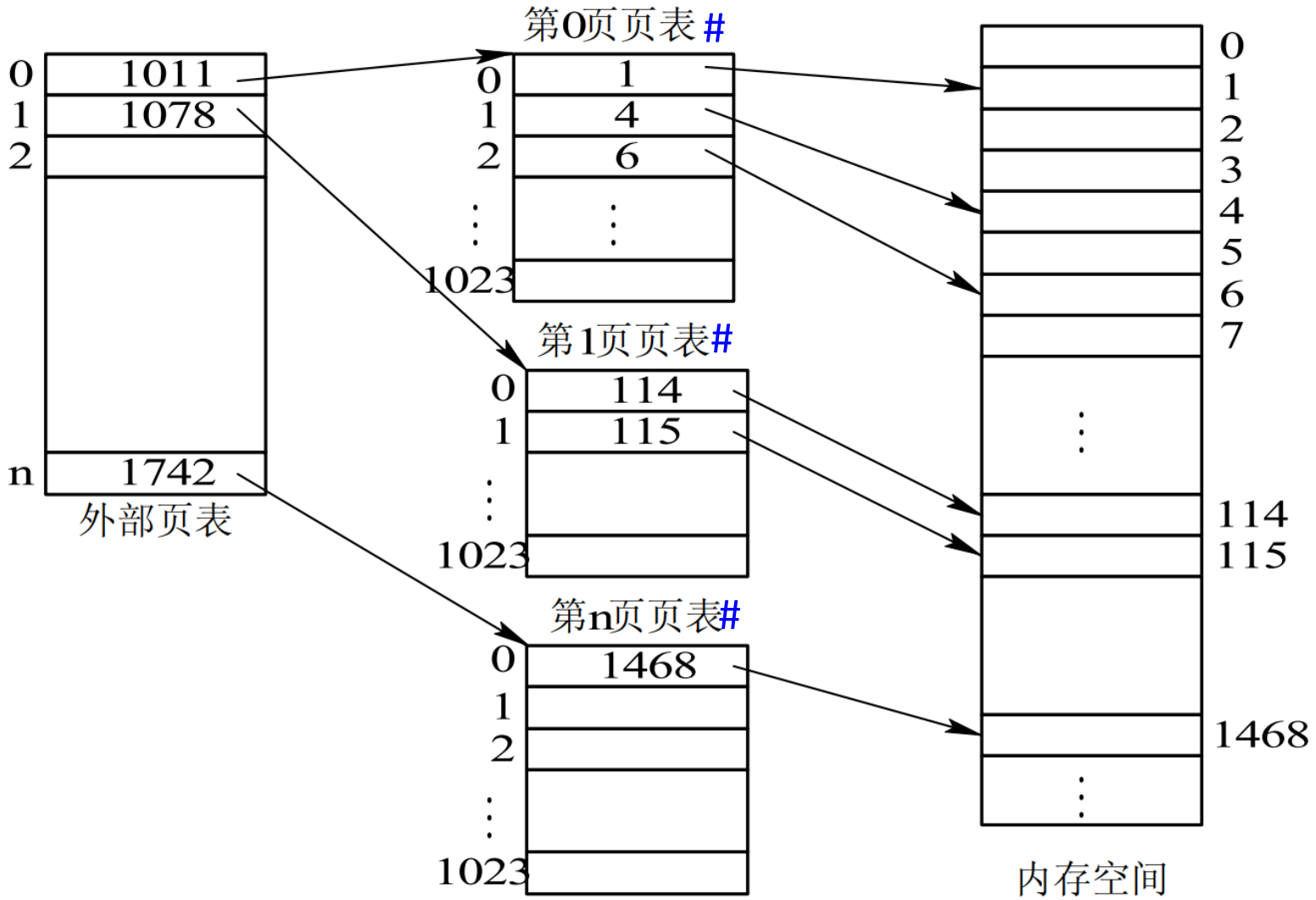
■ 两级页表 (Two-Level Page Table)

- 将**页表进行分页并离散**地将各个页面分别存放在不同的物理块中的办法来加以解决，同样也要为离散分配的页表再建立一张页表——**外层页表**(Outer Page Table)，在每个页表项中记录了页表页面的物理块号。
- 当页面大小为 4 KB 时，若采用一级页表结构应有 20 位的页号，即页表项应有 1 兆个；在采用两级页表结构时，再对页表进行分页，即外层页表中的外层页内地址 P2为 10 位，外层页号 P1也为 10 位。此时的逻辑地址结构：



4.5.4 两级和多级页表

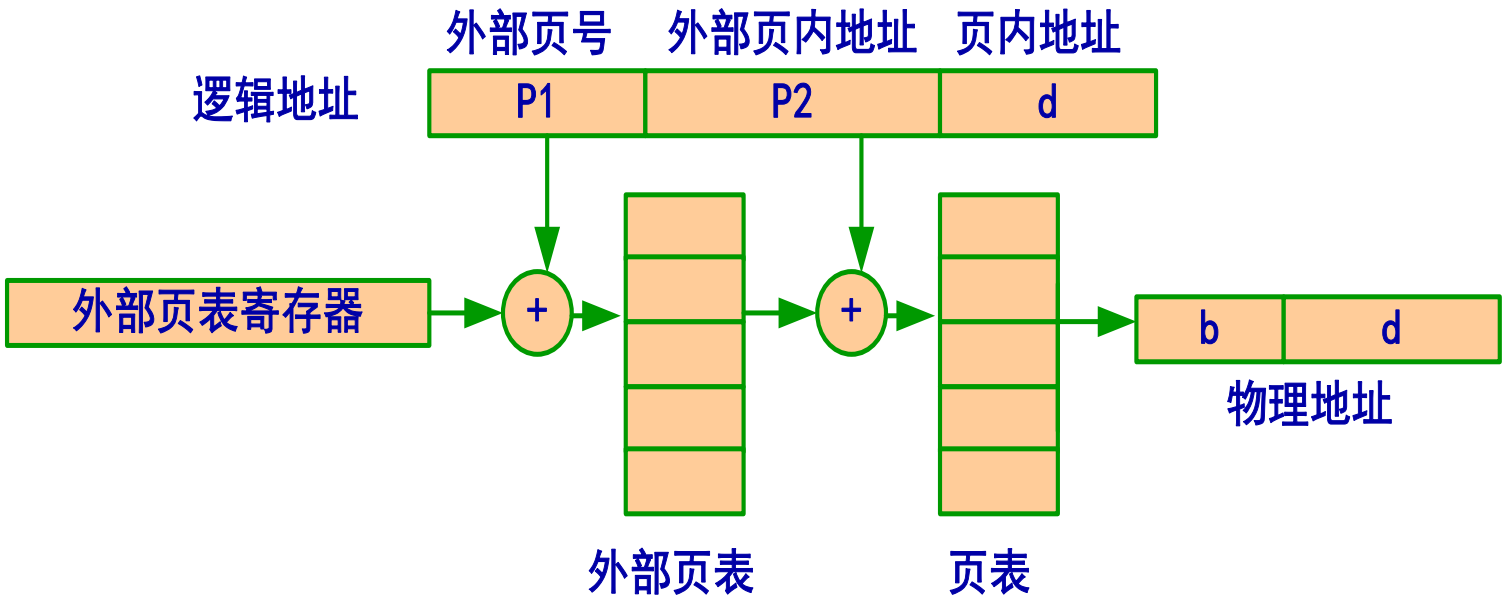
- **两级页表 (Two-Level Page Table)**



4.5.4 两级和多级页表

■ 两级页表地址变换机构

➢ 增设一个外层页表寄存器，用于存放外层页表的始址，并利用逻辑地址中的外层页号，作为外层页表的索引，从中找到指定页表分页的始址，再利用 P2 作为指定页表分页的索引，找到指定的页表项，其中即含有该页在内存的物理块号，用该块号和页内地址 d 即可构成访问的内存物理地址。



具有两级页表的地址变换机构

4.5.4 两级和多级页表

- **多级页表**
  - 对于 32 位的机器，采用两级页表结构是合适的；对于 64 位的机器，则采用多级页表，将外层页表再进行分页，也就是将各分页离散地装入到不相邻接的物理块中，再利用第 2 级的外层页表来映射它们之间的关系。
  - 对于 64 位的计算机，如果要求它能支持 2^64 B(= 1 844 744 TB)规模的物理存储空间，则即使是采用三级页表结构也是难以办到的；而在当前的实际应用中也无此必要。故在近两年推出的 64 位 OS 中，把可直接寻址的存储器空间减少为 45 位长度(即 2^45)左右，这样便可利用三级页表结构来实现分页存储管理。

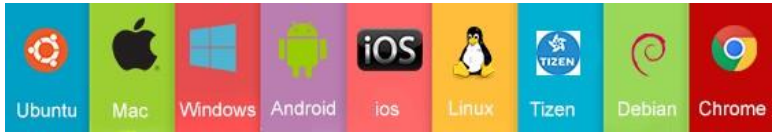
4.5.5 反置页表

- **反置页表的引入**
  - 在分页系统中，为每个进程配置了一张页表，进程逻辑地址空间中的每一页，在页表中都对应有一个页表项。在现代计算机系统中，通常允许一个进程的逻辑地址空间非常大，因此就需要有许多的页表项，而因此也会占用大量的内存空间。
  - 为了减少页表占用的内存空间，引入了反置页表：为每一个物理块设置一个页表项，并将它们按物理块的编号排序，其中的内容则是页号和其所隶属进程的标识符。
- **地址变换**
  - 根据进程标识符和页号，去检索反置页表。如果检索到与之匹配的页表项，则该页表项(中)的序号i便是该页所在的物理块号，可用该块号与页内地址一起构成物理地址送内存地址寄存器。
  - 若检索了整个反置页表仍未找到匹配的页表项，则表明此页尚未装入内存。

4.5 分页式存储管理

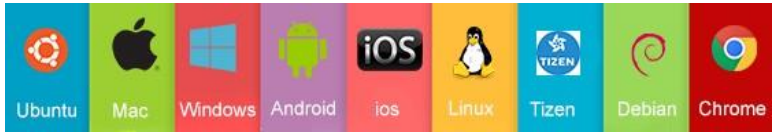
- **优点：**
- **缺点1：**
- **缺点2：**
- **缺点3：**
- **缺点4：**
  - **存储扩充问题仍未得到解决，当没有足够空间装下整个作业地址空间时，该作业仍无法运行。**

存储器管理



- 存储器的层次结构
- 程序的装入和链接
- 连续分配存储管理方式
- 对换（Swapping）
- 分页存储管理方式 
- 分段存储管理方式

存储器管理



分段存储管理方式

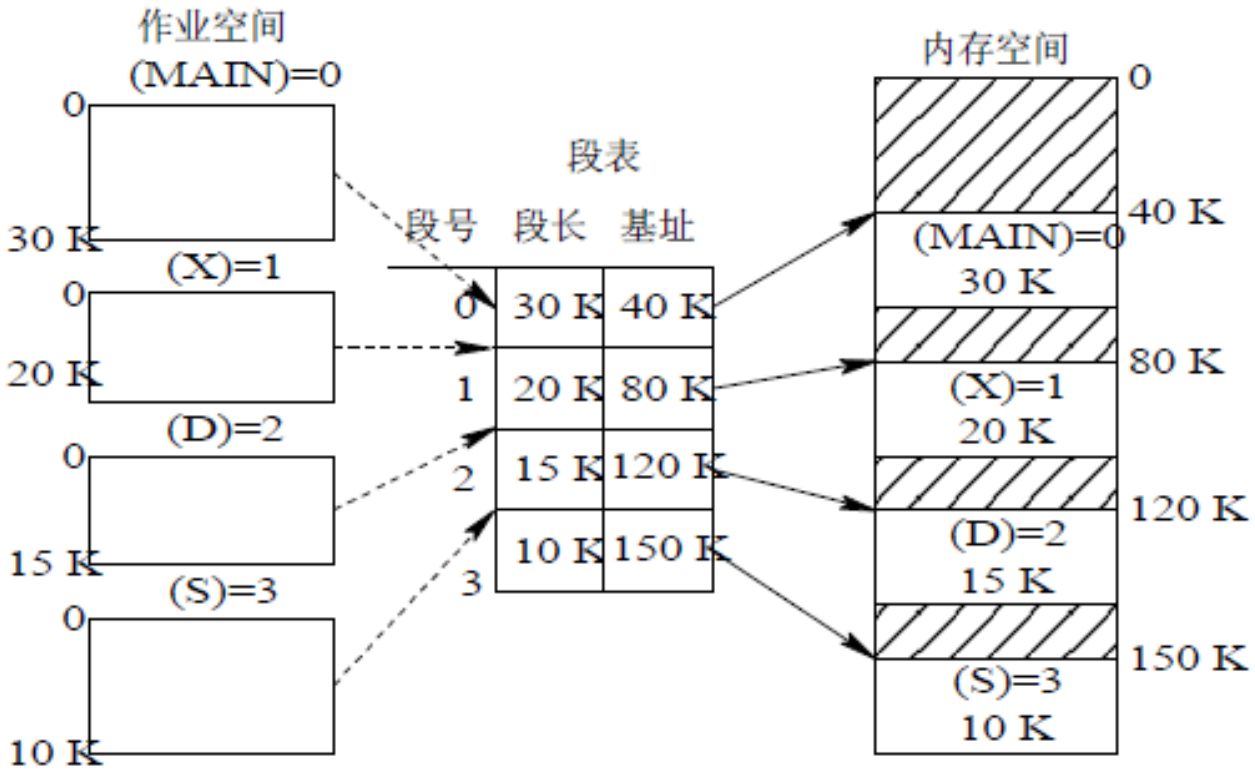
- 分段存储管理方式的引入
- 分段系统的基本原理
- 信息共享
- 段页式存储管理方式

4.6.1 分段存储管理方式的引入

- **引入分段存储管理方式的目的**
  - 满足用户(程序员)在编程和使用上多方面的要求
- **每个段可有其逻辑意义及功能：**
  - 方便编程：程序更直观，更具可读性
  - 分段信息共享：段可以是信息的逻辑单位，可简化共享的实现
  - 分段信息保护：一般以一个进程、函数或文件为基本单位进行保护，更有效方便
  - 动态增长：如数据段的动态增长
  - 动态链接：动态链接要求以目标程序（即段）作为链接基本单位

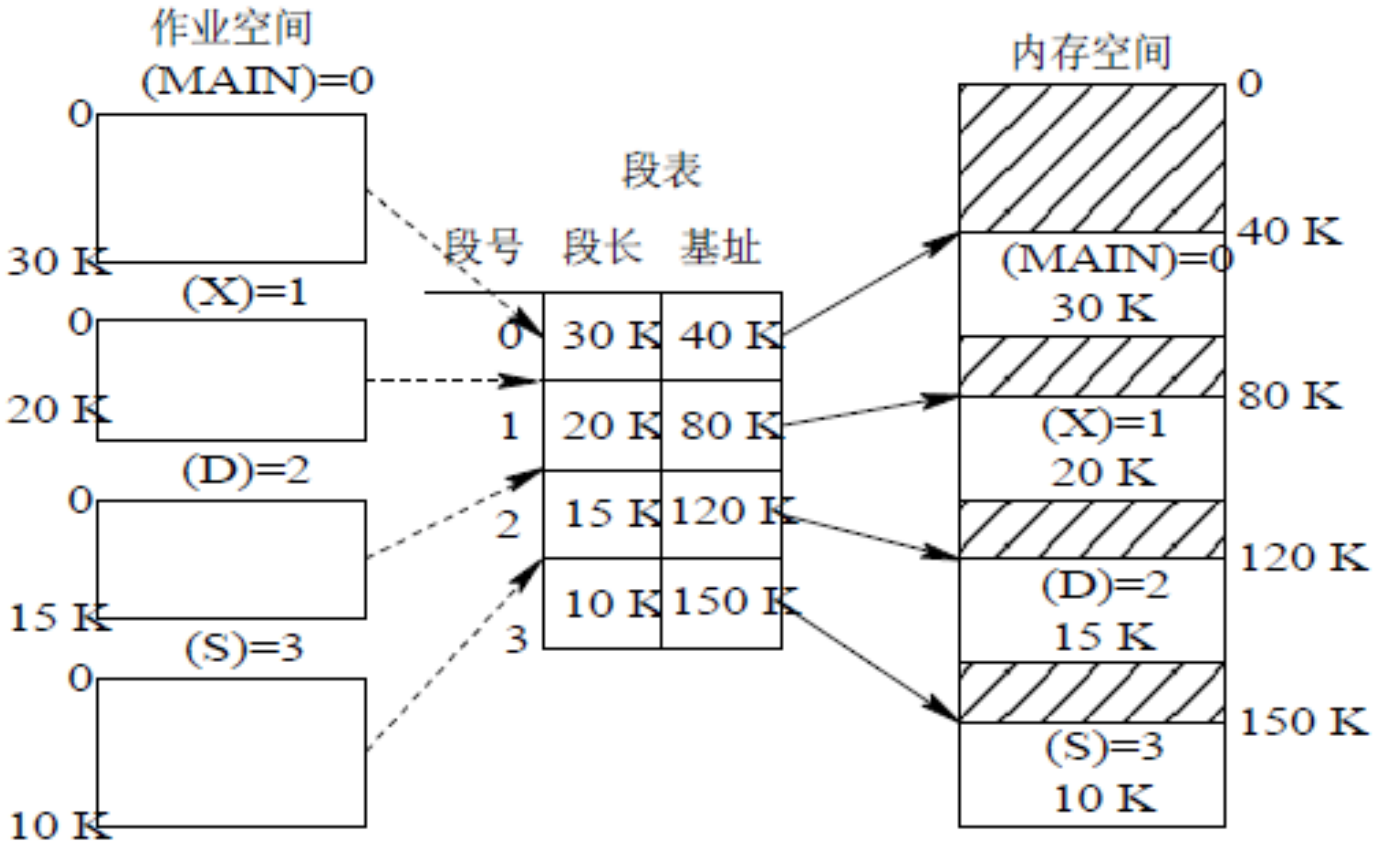
4.6.2 分段系统的基本原理

- **分段**
  - 作业的地址空间被划分为若干个段，每个段定义了一组**逻辑信息**，并有自己的**名字**和对应的**段号**。每个都从0开始编址，采用一段**连续的地址空间**。段的长度由相应的逻辑信息组的长度决定。整个作业的地址空间由于是分成多个段，因而是**二维**的，亦即，其逻辑地址由**段号(段名)和段内地址**所组成。



4.6.2 分段系统的基本原理

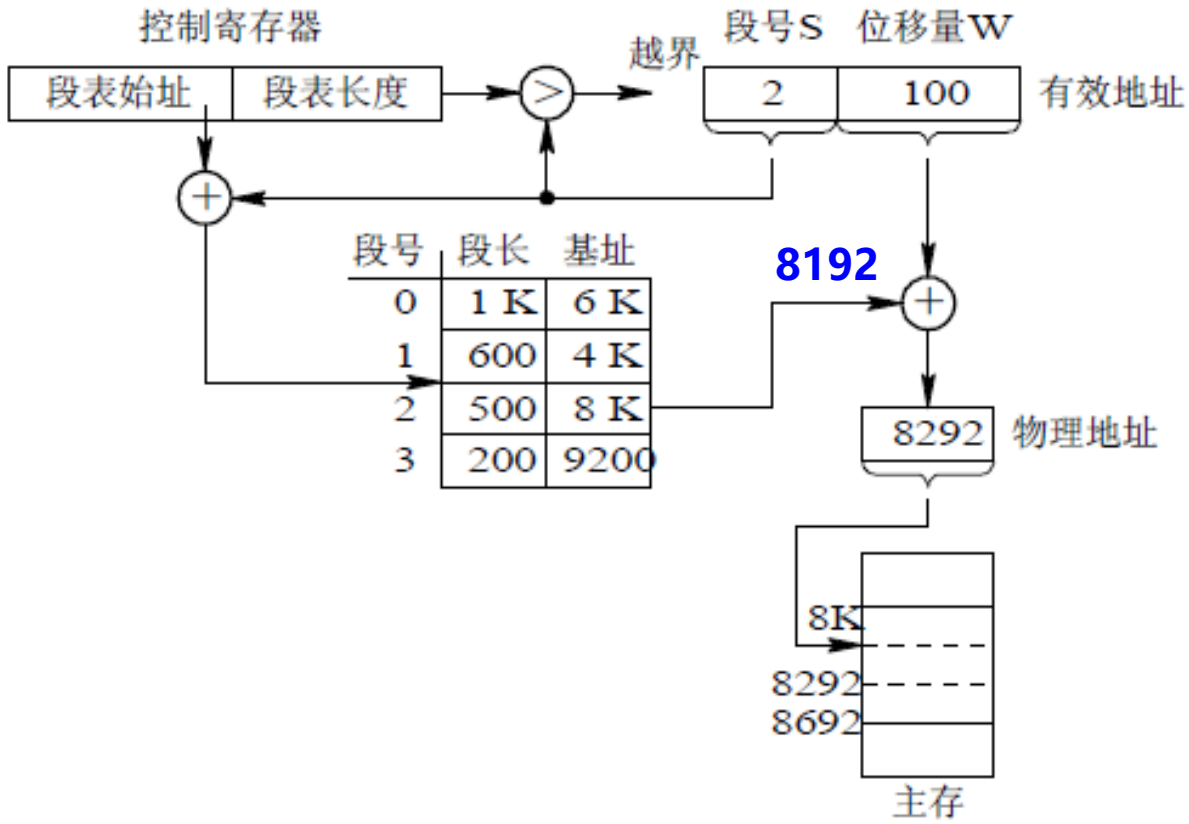
- **段表：实现从逻辑段到物理内存区的映射**
  - **为每个分段分配一个连续的分区，进程中的各个段可以离散地移入内存中不同的分区中。为每个进程建立一张 “段表”。每个段在表中占一个表项，记录了该段在内存中的起始地址(基址)和段的长度**



4.6.2 分段系统的基本原理

■ 地址变换机构

➢ 系统中设置了段表寄存器，用于存放段表始址和段表长度TL。在进行地址变换时，系统将逻辑地址中的段号与段表长度TL进行比较。若S>TL，表示段号太大，是访问越界，于是产生越界中断信号；若未越界，则根据段表的始址和该段的段号，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d是否超过该段的段长SL。若超过，即d>SL，同样发出越界中断信号；若未越界，则将该段的基址d 与段内地址相加，即可得到要访问的内存物理地址。



4.6.2 分段系统的基本原理

■ 分页和分段的主要区别

- **页是信息的物理单位**，分页是为实现离散分配方式，以消减内存的外零头，提高内存的利用率。或者说，分页仅仅是由于系统管理的需要而不是用户的需要。
- **段则是信息的逻辑单位**，它含有一组其意义相对完整的信息。分段的目的是为了能更好地满足用户的需要。
- **页的大小固定且由系统决定**，由系统把逻辑地址划分为页号和页内地址两部分，是由机器硬件实现的，因而在系统中只能有一种大小的页面；
- 而**段的长度却不固定**，决定于用户所编写的程序，通常由编译程序在对源程序进行编译时，根据信息的性质来划分。
- **分页的作业地址空间是一维的**，即单一的线性地址空间，程序员只需利用一个记忆符，即可表示一个地址；
- **分段的作业地址空间则是二维的**，程序员在标识一个地址时，既需给出段名，又需给出段内地址。

4.6.3 信息共享

- **分页和分段的主要区别**
  - **分段系统的一个突出优点，是易于实现段的共享，即允许若干个进程共享一个或多个分段，且对段的保护也十分简单易行。**

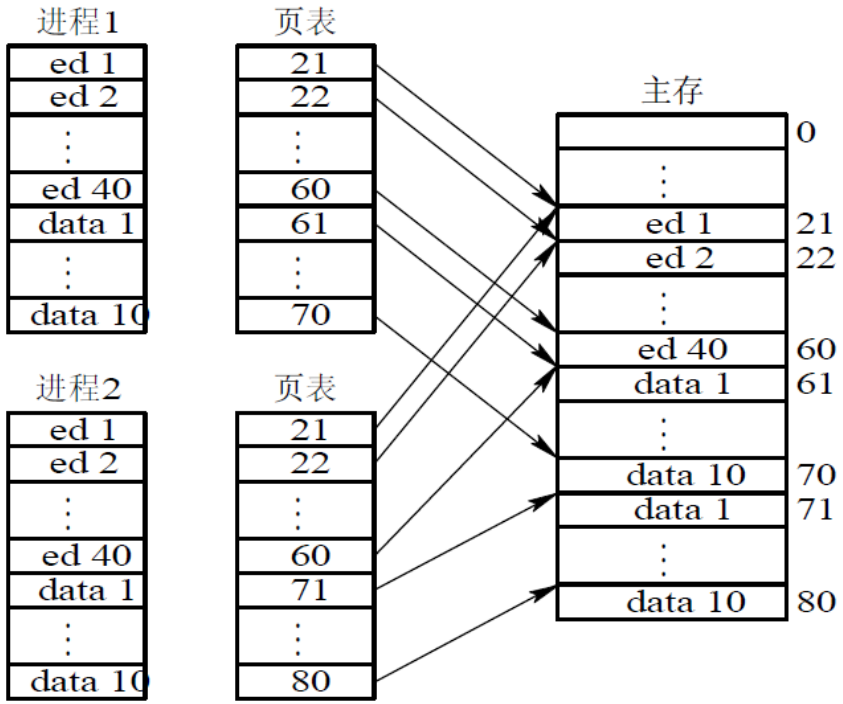


图 4-19　分页系统中共享 editor 的示意图

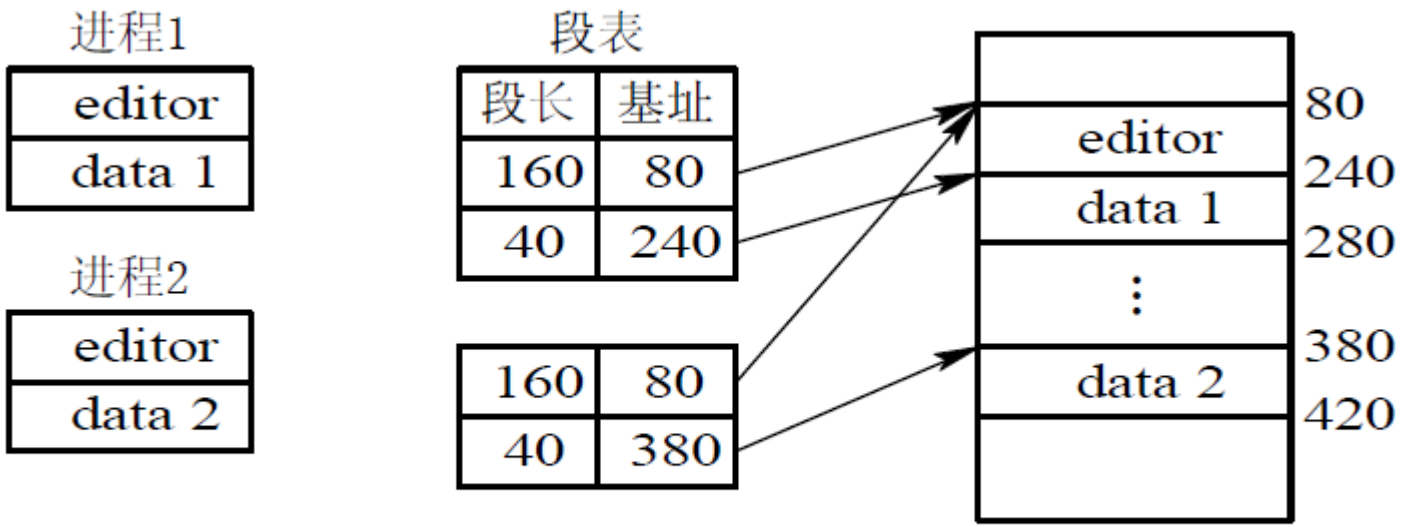


图 4-20　分段系统中共享 editor 的示意图

4.6.4 段页式存储管理方式

- **基本原理**
  - **分段和分页原理的结合，即先将用户程序分成若干个段，再把每个段分成若干个页，并为每一个段赋予一个段名**

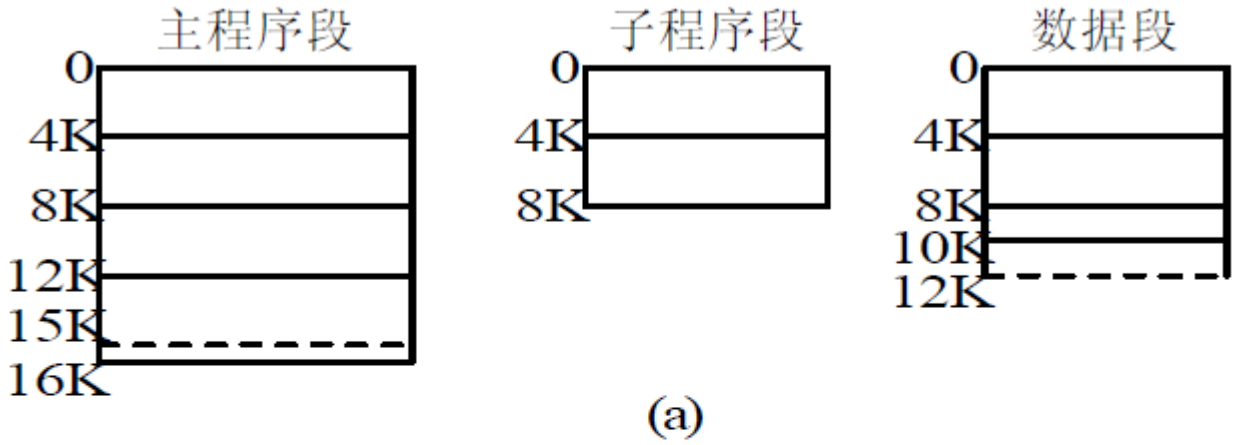
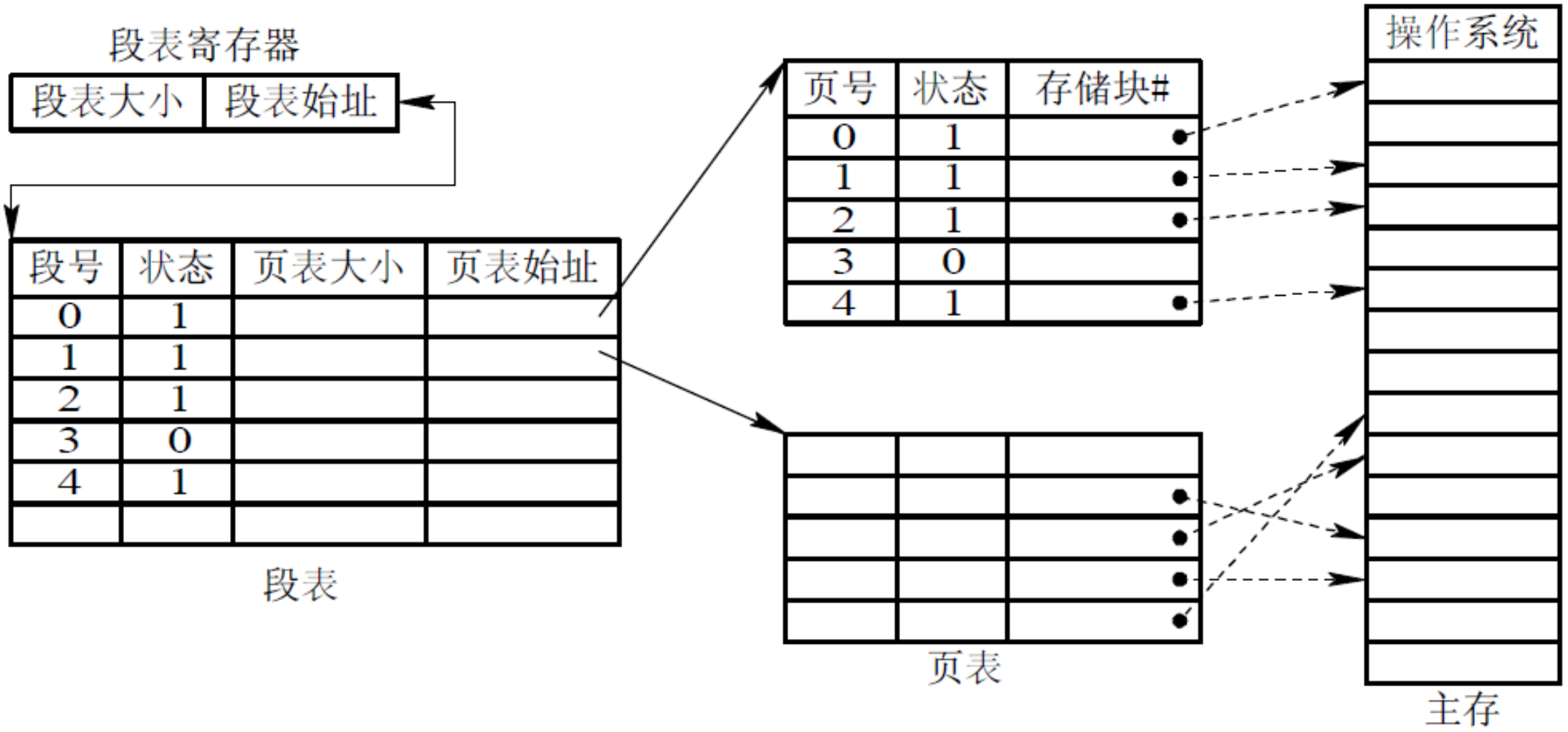


图 4-21 作业地址空间和地址结构

4.6.4 段页式存储管理方式

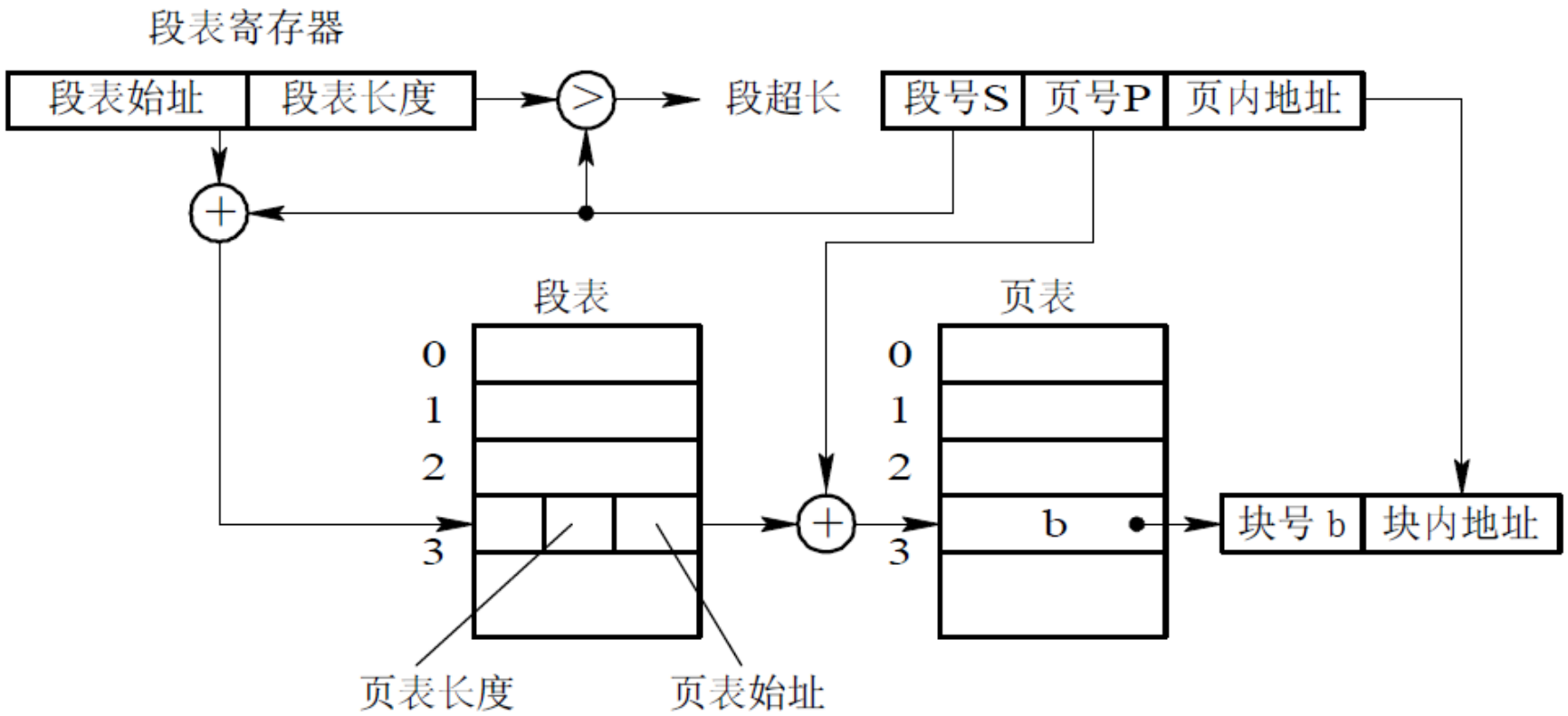
- **基本原理**



4.6.4 段页式存储管理方式

■ 地址变换过程

➢ 配置一个段表寄存器，存放段表始址和段表长TL。首先用段号S与段表长TL进行比较。若S<TL，利用段表始址和段号来求出该段所对应的段表项在段表中的位置，得到该段的页表始址，并利用逻辑地址中的段内页号P 来获得对应页的页表项位置，得到物理块号b，再用块号b和页内地址来构成物理地址



THEORY

PRACTICAL

6. 某系统使用 32 位逻辑地址，页大小为 4kbytes，以及 36 位物理地址。那么该系统中的页表大小为： A

A) 2^20 个页表项(2^(32-12)).

B) 2^24 个页表项(2^(36-12)).

C) 2^4 个页表项 (2^(36-32)).

D) 2^12 个页表项

15. 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，则段长最大 C

A. 2 的 8 次方字节　　B. 2 的 16 次方字节　　C. 2 的 24 次方字节　　D. 2 的 32 次方字节

2. 已知某系统页面长 4K 字节，页表项 4 字节，采用多层分页策略映射 64 位虚拟地址空间。若限定最高层页表占 1 页。问它可以采用几层分页策略。（15 分）

该系统虚拟地址空间为字节，页面长4K字节，页表项每项4字节，即每页可放页表项的个数为2^10，第二级页表同样可存 2^10 个页表项，所以可以采用 [64-12 / 10] = 6 级页表

如果一个分页系统的页表存放在内存。

（1）若对内存的一次存取需要1.2s，请问一次页面访问的存取需要花多少时间?

（2）若系统配置了联想寄存器，对快表的命中率为70%，假如查询联想寄存器的时间忽略不计，请问实现一次页面访问的存取时间是多少?

如果一个分页系统的页表存放在内存。

（1）若对内存的一次存取需要1.2s，请问一次页面访问的存取需要花多少时间?

1.2 * 2 = 2.4s

（2）若系统配置了联想寄存器，对快表的命中率为70%，假如查询联想寄存器的时间忽略不计，请问实现一次页面访问的存取时间是多少?

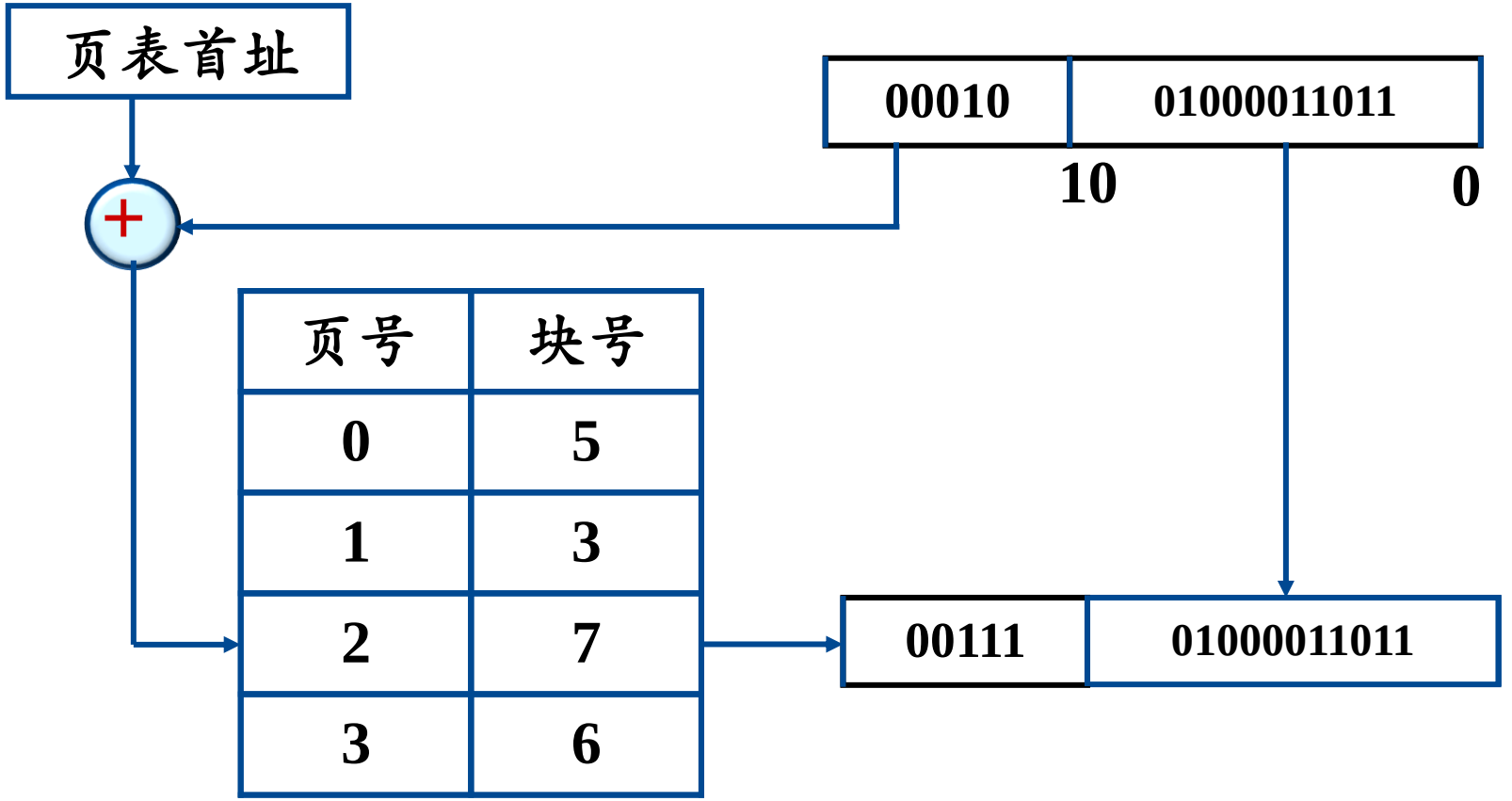
0.7 * 0 + 0.3 * 1.2 + 1.2 = 1.56s

本章例题

在分页存储管理系统中，有一作业大小为4页，页长为2K，页表如下，试借助地址变换图（即要求画出地址变换图）求出逻辑地址4635所对应的物理地址。

页号	块号
0	5
1	3
2	7
3	6

本章例题



物理地址为：14875

某分页系统的逻辑地址结构采用16位，其中高6位用于页号，10位用于页内地址，问：这样的地址结构下一页有多少字节？逻辑地址可有多少页？一个作业最大空间是多少？ 有一个程序，访问的逻辑地址分别是2058，3072和1023，请问它们的页号是多少？页内地址是多少？

某分页系统的逻辑地址结构采用16位，其中高6位用于页号，10位用于页内地址，问：这样的地址结构下一页有多少字节？逻辑地址可有多少页？一个作业最大空间是多少？ 有一个程序，访问的逻辑地址分别是2058，3072和1023，请问它们的页号是多少？页内地址是多少？

① 一页含有 $2^{10} = 1024$ 个字节

② 逻辑地址有 $2^6 = 64$ (页)

③ 一个作业最大空间是 $64 \times 1K = 64KB$

2058 ⇒ 1000 0000 1010
页号 页内地址

页号为2，页内地址为10

3072 ⇒ 1100 0000 0000
页号 页内地址

页号为3，页内地址为0

1023 ⇒ 0011 1111 1111
页号 页内地址

本章例题

4.某计算机系统按字节编址，采用二级页表的分页存储管理方式，虚拟地址格式如下所示:

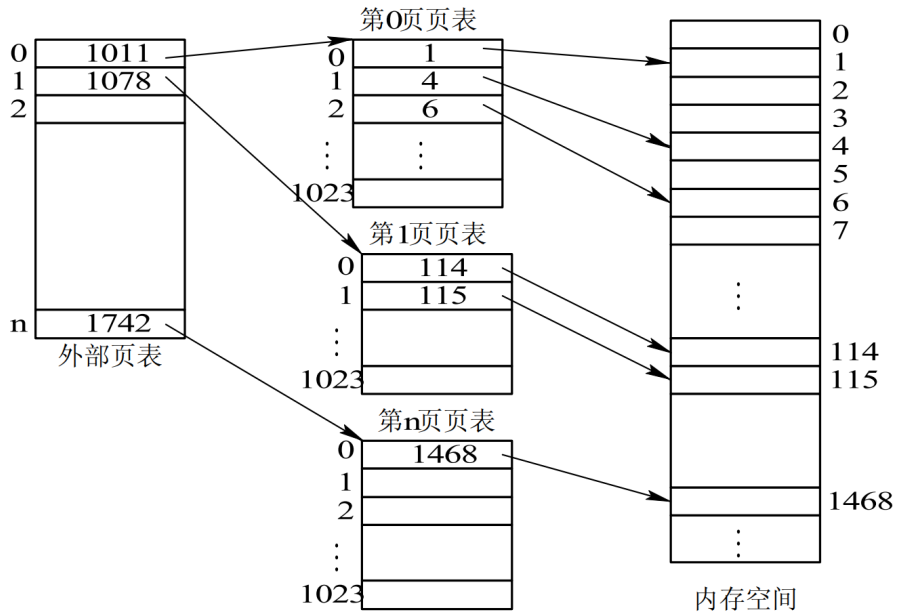
10位	10位	12位
页目录号	页表索引	页内偏移量

请回答下列问题:

1)页和页框的大小各为多少字节？进程的虚拟地址空间大小为多少页？

2)若页目录项和页表项均占4B,则进程的页目录和页表共占多少页?写出计算过程。

3)若某指令周期内访问的虚拟地址为0100 0000H和0111 2048H,则进行地址转换时共访问多少个二级页表?说明理由。



本章例题

4.某计算机系统按字节编址，采用二级页表的分页存储管理方式，虚拟地址格式如下所示:

10位	10位	12位
页目录号	页表索引	页内偏移量

请回答下列问题:

1)页和页框的大小各为多少字节？进程的虚拟地址空间大小为多少页？

2)若页目录项和页表项均占4B,则进程的页目录和页表共占多少页?写出计算过程。

3)若某指令周期内访问的虚拟地址为0100 0000H和0111 2048H,则进行地址转换时共访问多少个二级页表?说明理由。

1）页和页框大小均为 4KB。进程的虚拟地址空间大小为 $2^{32}/2^{12}=2^{20}$ 页。

2）$(2^{10}\times4)/2^{12}$（页目录所占页数）+ $(2^{20}\times4)/2^{12}$（页表所占页数）= 1025 页。

3）需要访问一个二级页表。因为虚拟地址 0100 0000H 和 0111 2048H 的最高 10 位的值都是 4，访问的是同一个二级页表。

本章例题

如果一个系统的段表为：
求下列逻辑地址相应的物理地址。如果越界请指明。
{0,380}、{1,20}、{1,24}、{2,200}、{3,500}、{4,120}。

段号	始址	段长
0	200	510
1	900	30
2	100	80
3	1200	500
4	1800	80

本章例题

如果一个系统的段表为：
求下列逻辑地址相应的物理地址。如果越界请指明。
{0,380}、{1,20}、{1,24}、{2,200}、{3,500}、{4,120}。

段号	始址	段长
0	200	510
1	900	30
2	100	80
3	1200	500
4	1800	80

{0,380}：200+380=580

{1,20}：900+20=920

{1,24}：900+24=924

{2,200}：200>80 越界

{3,500}：1200+500=1700

{4,120}：120>80 越界