第 五 章

进程的存放空间~~~~存储管理

存储管理

存储管理是指存储器资源（内存并涉及外存）的管理

本章主要内存管理，主要包括：

- 存储管理的功能
- 分区存储管理
- 覆盖与交换技术
- 页式管理
- 段式与页式管理
- 局部性原理和抖动问题

5.1 存储管理功能

- 5.1.1 虚拟存储器
  - 虚拟地址（ virtual address ）
    - 用户的程序经过汇编或编译后形成目标代码，目标代码通常采用**相对地址**的形式。这种相对地址即为虚拟地址
  - 物理地址
    - 内存中**存储单元的地址**。
  - 虚拟存储器
    - **将进程中的目标代码、数据等的虚拟地址组成的虚拟空间**

5.1 存储管理功能

- 5.1.1 虚拟存储器基本原理
  - 在程序装入时，不必将其全部读入到内存，而只需将当前需要执行的部分页或段读入到内存，就可让程序开始执行。
  - 在程序执行过程中，如果需执行的指令或访问的数据尚未在内存（称为缺页或缺段），则由处理器通知操作系统将相应的页或段调入到内存，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页或段调出保存在外存上，从而腾出空间存放将要装入的程序以及将要调入的页或段—具有请求调入和置换功能，只需程序的一部分在内存就可执行，对于动态链接库也可以请求调入。

北京科技大学 计算机系

5.1 存储管理功能

- **5.1.1 虚拟存储器技术的好处**
  - 可在较小的可用内存中执行较大的用户程序
  - 可在内存中容纳更多程序并发执行
  - 不必考虑编程时的程序结构
  - 提供给用户可用的虚拟内存空间通常大于物理内存

5.1 存储管理功能

5.1.2 地址变换

- 程序在成为进程前的准备工作：
  - **编辑：形成源文件（符号地址）**
  - **编译：形成目标模块（模块内符号地址解析）**
  - **链接：由多个目标模块或程序库生成可执行文件（模块间符号地址解析）**
  - **装入：构造 PCB ，形成进程（使用物理地址）**

5.1 存储管理功能

- 5.1.2 地址变换
  - **地址映射（地址重定位）**：将用户程序中的虚拟地址转换为运行时由机器直接寻址的物理地址。
  - 实现的方法：
    - 静态地址重定位 (static address relocation) ：是在**虚拟空间程序执行之前由装配程序完成地址映射**工作。
    - 动态地址重定位 (Dynamic address relocation) ：是在程序执行过程中，在 **CPU 访问内存之前，将要访问的程序或数据地址转换成内存地址。**

5.1 存储管理功能

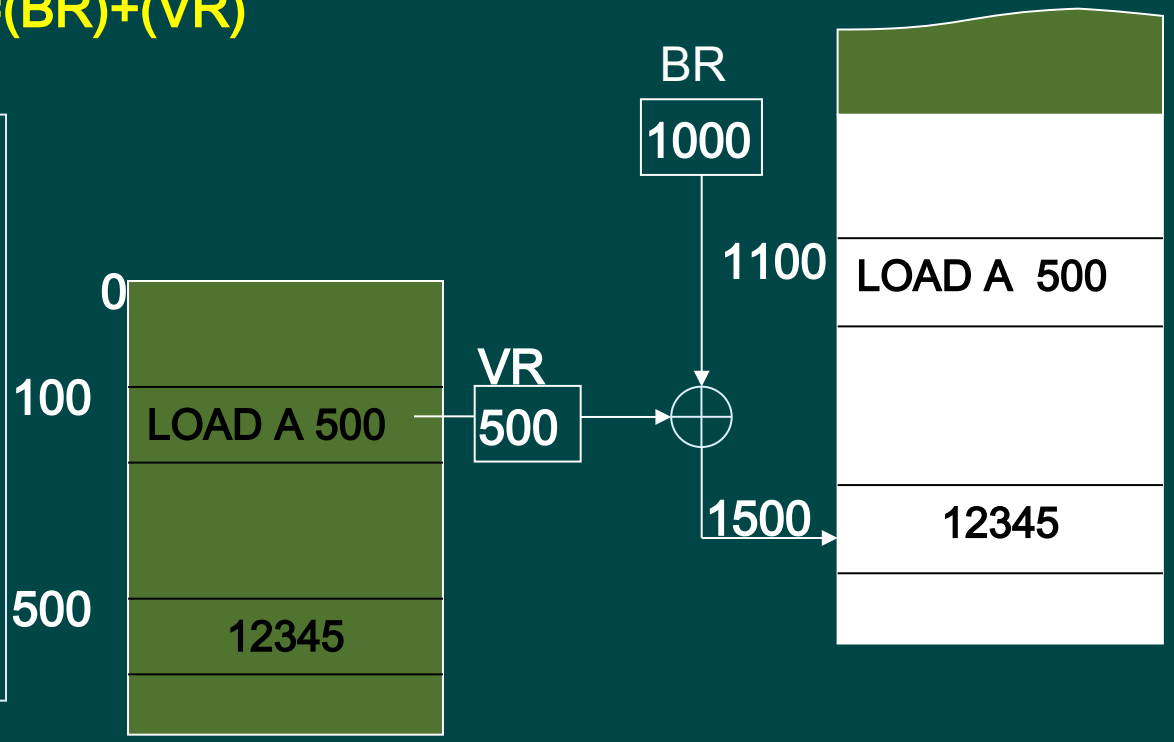
- 静态地址重定位
  - 优点：
    - 不需要硬件的支持
  - 缺点：
    - 使用静态重定位方法进行地址变换无法实现虚拟存储器。
    - 必须占用连续的内存空间
    - 难以做到程序和数据的共享

5.1 存储管理功能

- 动态地址重定位
  - 依靠硬件地址机构完成
  - 需要基地址寄存器 BR 和虚拟地址寄存器 VR
  - 物理内存地址 MA=(BR)+(VR)

具体过程：

1. 设置 BR ， VR
2. 将程序装入内存，并将内存首地址存入 BR 中
3. 将要访问的虚拟地址存入 VR
4. 地址变换机构把 VR 和 BR 内容相加，得到要访问的物理地址



5.1 存储管理功能

- 动态重定位主要优点：
  - 可以对内存进行非连续分配。
  - 动态重定位提供了实现虚拟存储器的基础。
  - 有利于程序段的共享。

5.1 存储管理功能

- 5.1.3 内外存数据传输的控制
  - 要实现内存扩充，内存和外存之间必须经常地交换数据。
    - 用户程序自己控制，如覆盖。
    - 操作系统控制
      - ✓ 交换方式（整体）：由操作系统把那些在内存中处于等待状态的进程换出内存，而把那些等待事件已经发生、处于就绪态的进程换入内存
      - ✓ 请求调入和预调入（页，段）

5.1 存储管理功能

5.1.4 内存分配与回收

- 为有效合理利用内存，分配与回收方法要考虑和确定以下 5 种策略和数据结构：
  - **分配结构**　登记内存使用情况
  - **放置策略**　确定调入内存的程序和数据的位置
  - **交换策略**　确定哪些程序段或数据段能调出内存，以便腾出足够的空间
  - **调入策略**　外存中的程序段和数据段什么时间按什么样的控制方式进入内存。
  - **回收策略**　回收策略包括：一是回收的时机，二是对所回收的内存空闲区和已存在的内存空闲区的调整。

5.1 存储管理功能

5.1.5 内存信息的共享与保护

- 在多道程序设计环境下，要限制各进程只在自己的存储区活动，**除了被允许共享的部分之外，各进程不能对别的进程的程序和数据段产生干扰和破坏。**因此须对内存中的程序和数据段采取保护措施

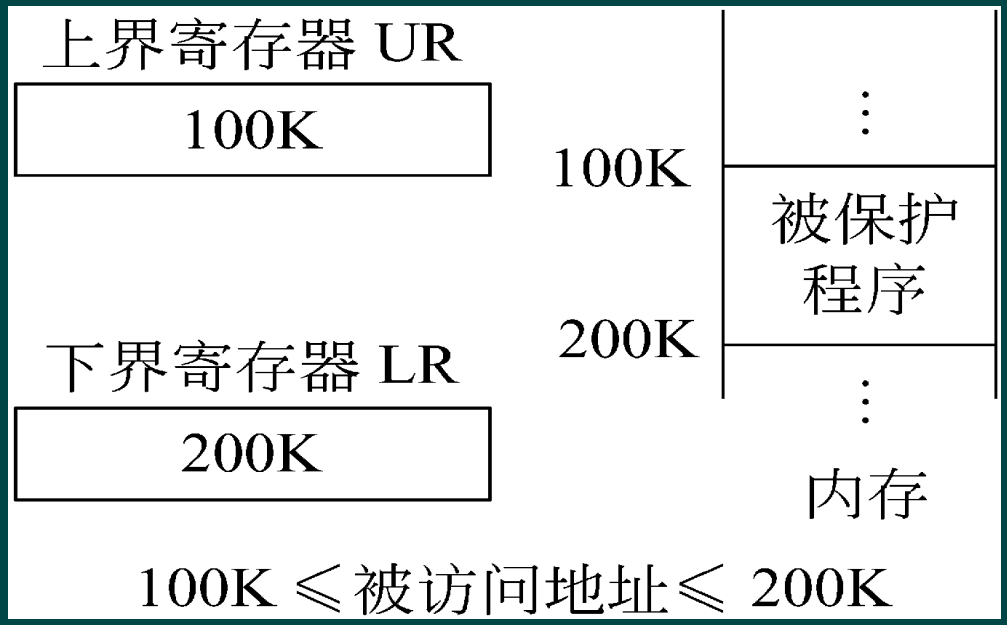
- 常用的内存信息保护方法有**硬件法、软件法**和**软硬件结合** 3 种方法。

5.1 存储管理功能

5.1.5 内存信息的共享与保护

上下界保护法

- **一种常用的硬件保护法：为进程设置了上下界寄存器，访问的地址不能超出上下界规定的范围。**若在规定的范围之内，则访问是合法的；否则是非法的，并产生访址越界中断。

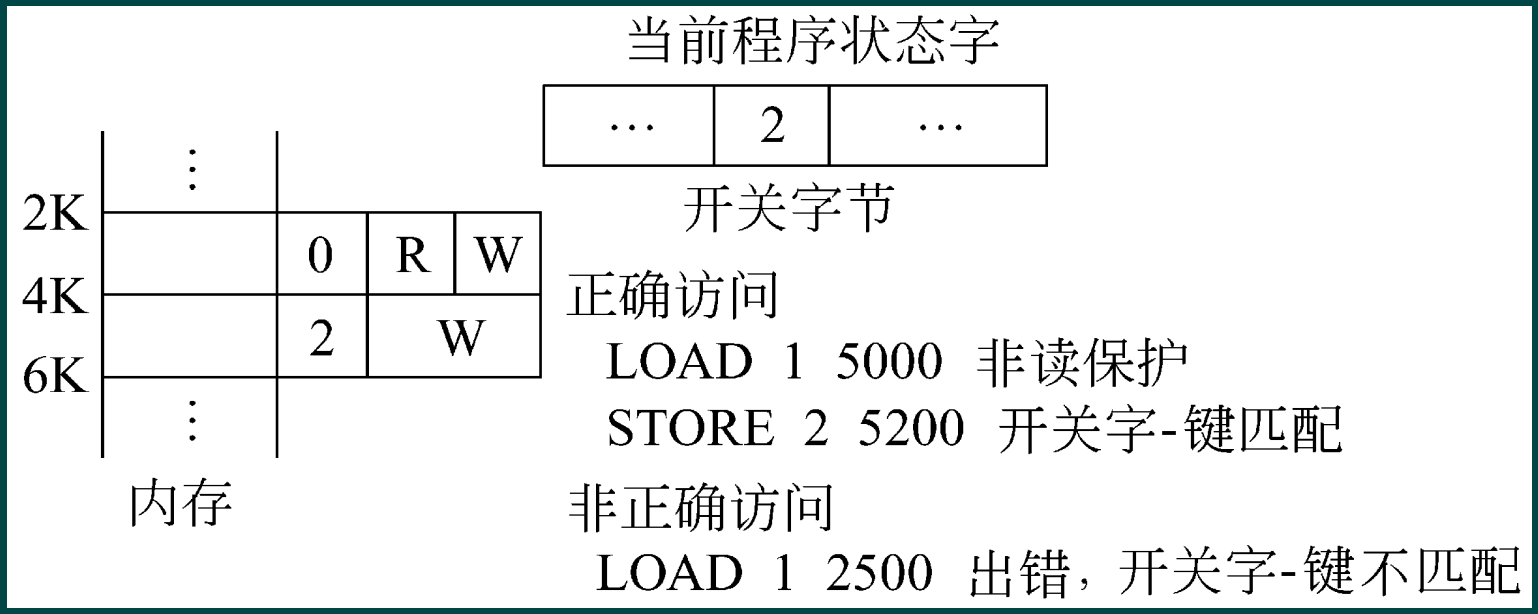


5.1 存储管理功能

◈ 保护键法

存储块分配一个单独的保护键。在程序状态字中则设置相应的保护键开关字段，对不同的进程赋予不同的开关代码以和被保护的存储块中的保护键匹配。

保护键可设置成：对读写同时保护（如 2k—4k) 或读、写单项保护（如 4k—6k)。**如果开关字与保护键匹配**，则访问该存储块是允许的，否则将产生访问出错中断。



5.1 存储管理功能

- 5.1.5 内存信息的共享与保护
  - **界限寄存器与 CPU 的用户态与核心态工作方式相结合的保护方式**
    - **在这种保护模式下，用户态进程只能访问那些在界限寄存器所规定范围内的内存部分，而核心态进程则可以访问整个内存地址空间。**


2006
GIFT CAMPAIGN

5.2 分区存储管理

- **(进程整体)分区管理原理：**
  给每一个内存中的进程划分一块适当大小的存储区，以连续存储各进程的程序和数据，使各进程得以并发执行。

- **按分区的时机，分区管理分类：**
  - **固定分区法**
  - **动态分区法**

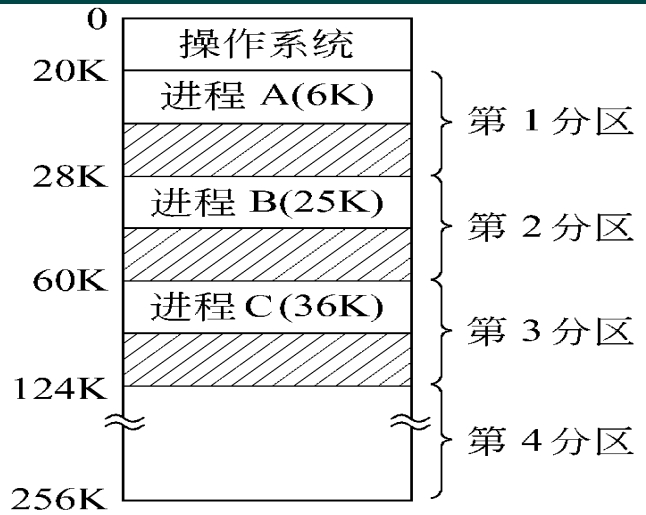
5.2 分区存储管理——固定分区法

固定分区法

- 把内存区固定地划分为若干个大小不等的区域。分区一旦划分结束，在整个执行过程中每个分区的长度和内存的总分区个数将保持不变。
- 对内存的管理和控制通过数据结构——分区表进行。

区号	分区长度	起始地址	状态
1	8K	20K	已分配
2	32K	28K	已分配
3	64K	60K	已分配
4	132K	124K	未分配

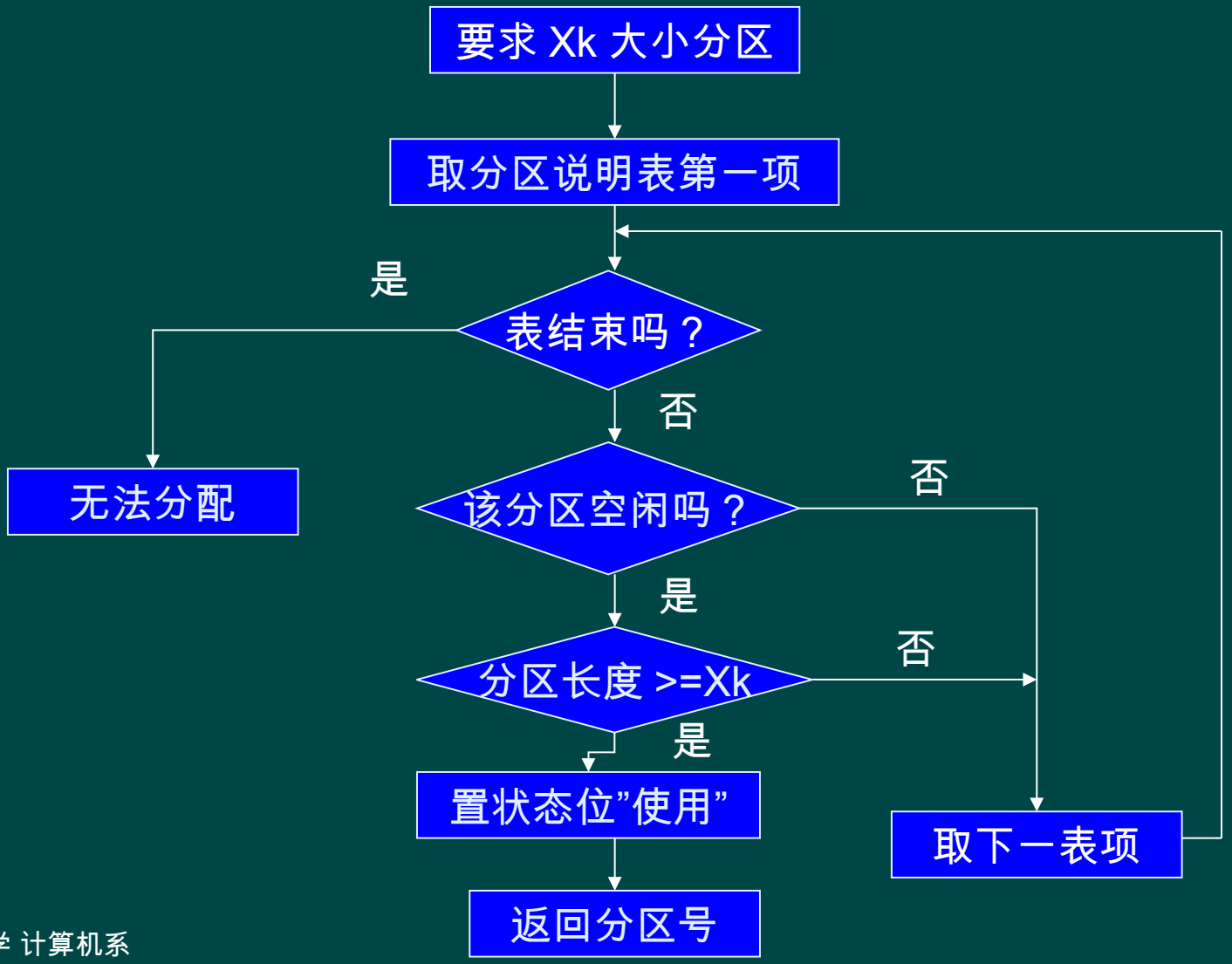
(a) 分区说明表



(b) 内存状态

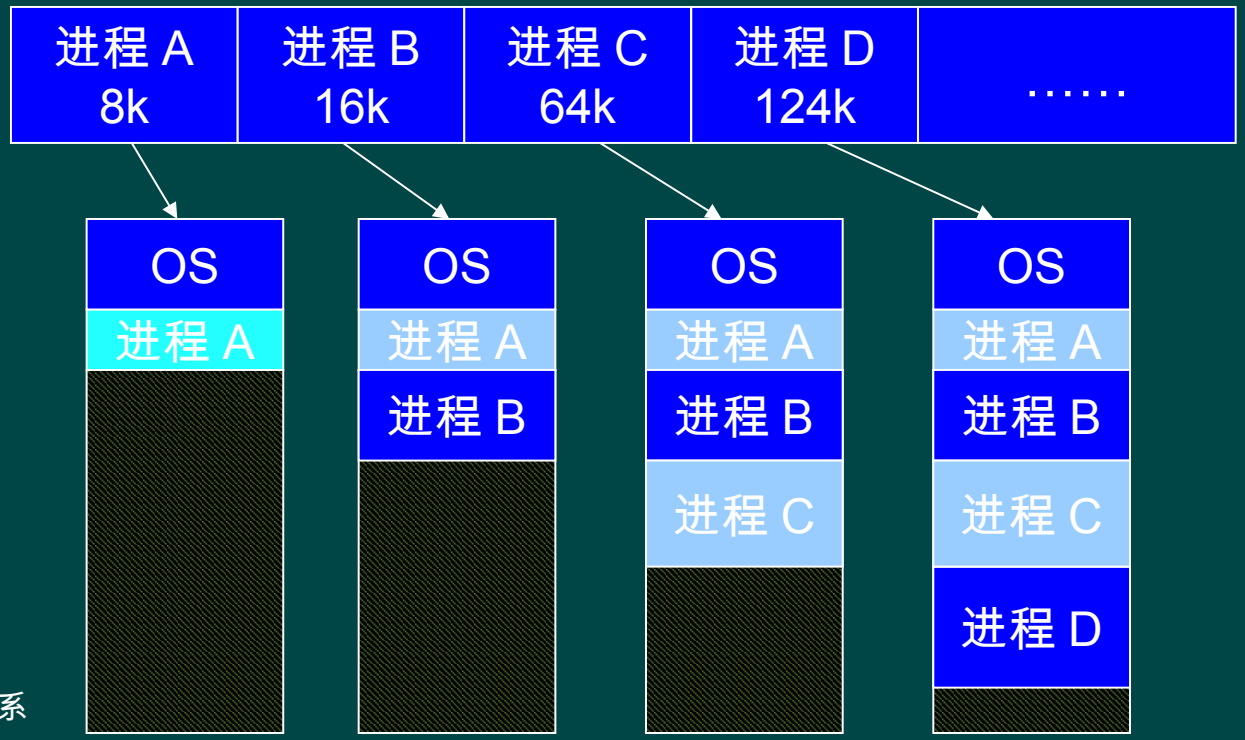
5.2 分区存储管理——固定分区法

- 分配



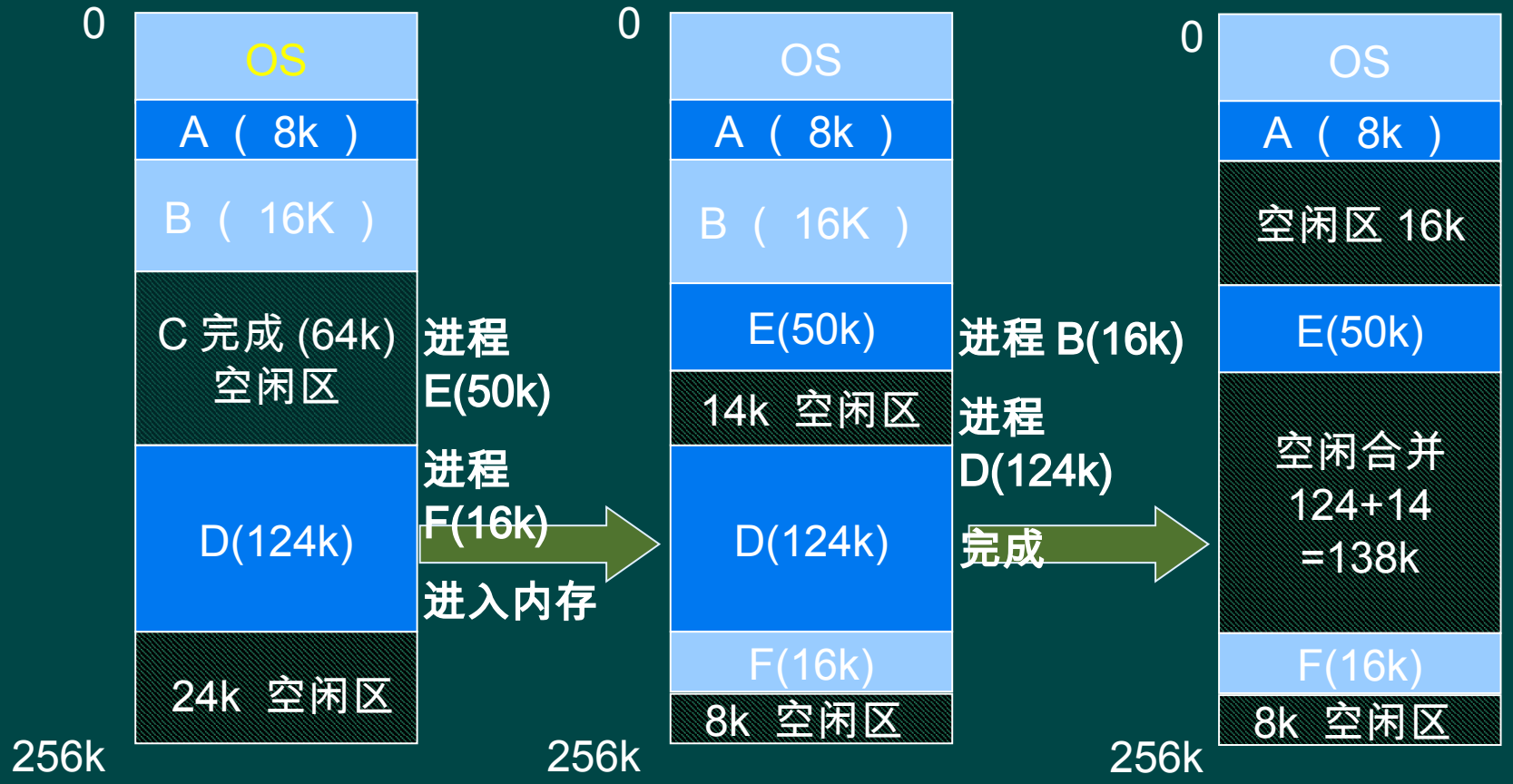
5.2 分区存储管理——动态分区法

- 动态分区法
  - **基本思路：在作业执行前并不直接建立分区，分区的建立是在作业的处理过程中进行的。且其大小可随作业或进程对内存的要求而改变。**



5.2 分区存储管理——动态分区法

- 内存分配变化过程

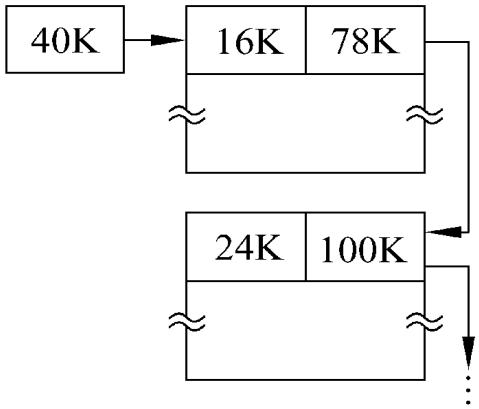


5.2 分区存储管理——动态分区法

- 动态分区法数据结构
  - 可用表：记录空闲区，包括区号，长度和起始地址
  - 自由链：把空闲区组成链表，空闲区的头几个单元存放本空闲区的大小及下个空闲区的起始地址。
  - 请求表：记录请求内存资源的作业或进程号和请求的内存大小

区号	分区长度	起始地址
1	16K	40K
3	24K	78K
5	9K	100K

(a) 可用表



(b) 自由链

作业(进程)号	请求长度
P_1	13K
P_2	20K
⋮	

(c) 请求表

5.2 分区存储管理——动态分区法

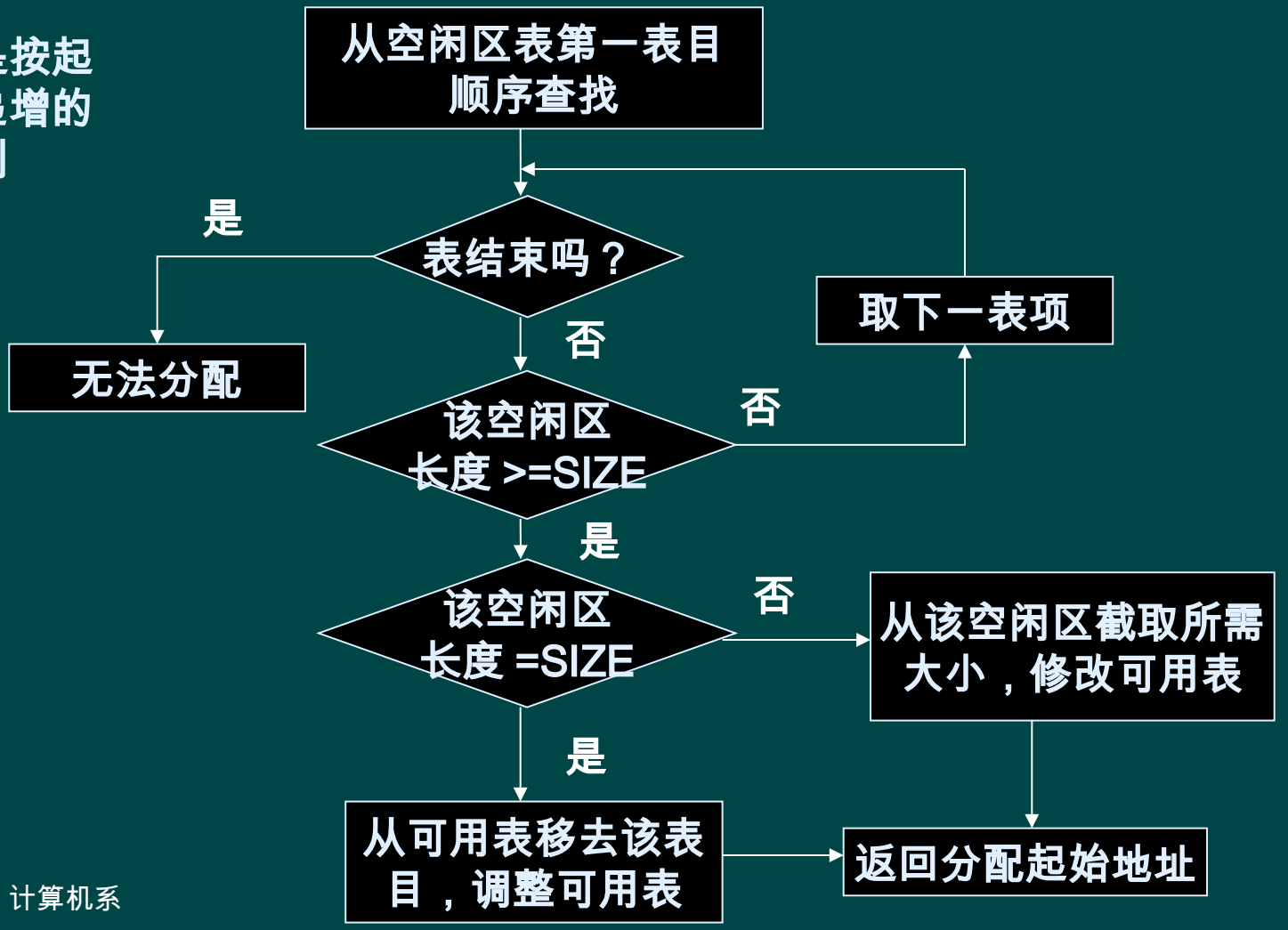
- 动态分区时的分配与回收主要解决 3 个问题：
  - **对于请求表中的要求内存长度，分配程序从可用表或自由链中寻找出合适的空闲区。**
  - **分配空闲区之后，更新可用表或自由链。**
  - **进程或作业释放内存资源时，和相邻的空闲区进行链接合并，更新可用表或自由链。**
- 空闲区分配算法：
  - **最先适应法（ first fit algorithm ）**
  - **最佳适应法（ best fit algorithm ）**
  - **最坏适应法（ worst fit algorithm ）**

5.2 分区存储管理——动态分区法

- 最先适应法

请求 SIZE 大小的分区

空闲区是按起始地址递增的次序排列



5.2 分区存储管理——动态分区法

- 最佳适应算法
  - **该算法要求按空闲区大小从小到大的次序组成空闲区可用表或自由链。**
- 最坏适应算法
  - **该算法要求空闲区按其大小递减的顺序组成空闲区可用表或自由链。**

5.2 分区存储管理——动态分区法

- 动态分区时的回收与拼接
  - **将一个新空闲可用区插入可用表或队列时，该空闲区和上下相邻区的关系是下述 4 种关系之一：**
    - **该空闲区的上下两相邻分区都是空闲区**
    - **该空闲区的上相邻区是空闲区**
    - **该空闲区的下相邻区是空闲区**
    - **两相邻区都不是空闲区**

5.2 分区存储管理——动态分区法

- 几种算法比较

从搜索速度上看

- **最先适应算法具有最佳性能。**

从释放速度来看

- **最先适应算法也是最佳的。**

从空间利用率来看

- **最佳适应法找到的空闲区是最佳的**

5.2 分区存储管理

- 分区存储管理的优缺点
  - 优点：
    - 实现了多个作业或进程对**内存的共享**。
    - 该方法要求的**硬件支持少**，**算法简单**，容易实现。
  - 缺点：
    - 内存**利用率仍然不高**。存储器中可能含有从未用过的信息。存在空闲区**碎片**不能利用的问题。
    - 作业或进程的大小受**分区大小控制**，除非配合采用覆盖和交换技术。
    - 难以实现各**分区间的信息**共享。

北京科技大学 计算机系

5.3 覆盖与交换技术

- 覆盖
  - 引入：其目的是在较小的可用内存中运行较大的程序。用于多道程序系统，与分区存储管理配合使用。
  - 原理：一个程序的几个代码段或数据段，按照时间先后来占用公共的内存空间。
    - 将程序的必要部分（常用功能）的代码和数据常驻内存；
    - 可选部分（不常用功能）在其他程序模块中实现，平时存在外存（覆盖文件），用到时才装入内存；
    - 不存在调用关系的模块不必同时装入，可以相互覆盖。
  - 缺点：编程时必须划分程序模块和确定程序模块之间的覆盖关系，增加编程复杂度。装入覆盖文件，以时间来换取空间。

5.3 覆盖与交换技术

- 交换技术
  - 引入
    - 多个程序并发执行，可以将暂时不执行的程序送到外存中，要执行的程序读入内存。
  - 原理
    - 暂停执行内存中的进程，将整个进程的地址空间保存到外存的交换区中（换出 swap out ），而将外存中由阻塞变为就绪的进程的地址空间读入到内存中，并将该进程送到就绪队列（换入 swap in ）。
  - 优点
    - 增加并发运行的程序数目；编写程序时不用考虑程序结构
  - 缺点
    - 程序整个地址空间都进行换入 / 换出增加了处理机开销。

5.4 页式管理

- 页式管理的基本原理
  - 将各进程的虚拟空间划分成若干个长度相等的页(page)，页式管理把内存空间按页的大小划分成片或者页面（ page frame ），然后把页式虚拟地址与内存地址建立一一对应页表，并用相应的硬件地址变换机构，来解决离散地址变换问题。页式管理采用请求调页或预调页技术实现了内外存存储器的统一管理。
- 优点
  - 没有外碎片，每个内碎片不超过页大小。
  - 一个程序不必连续存放。

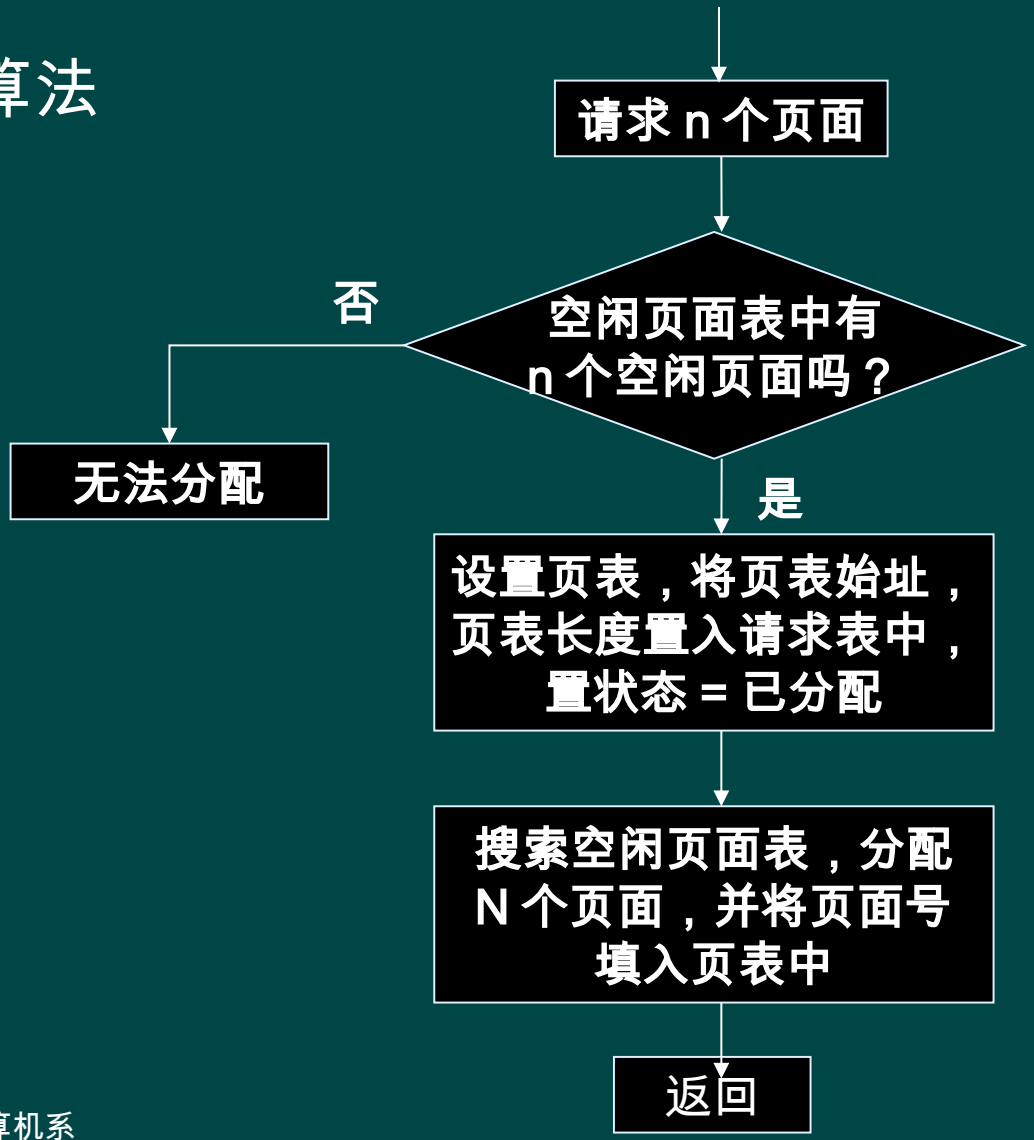
- 分页管理重点在于页划分后的地址变换以及页面的调入调出技术

5.4 页式管理—静态页式管理

- 内存页面分配与回收
  - 系统通过存储页面表、请求表以及页表来完成内存的分配工作。
    - 页表：内存中的一块固定存储区。页式管理时每个进程至少有一个页表。
    - 请求表：用来确定作业或进程的虚拟空间的各页在内存中的实际对应位置。
    - 存储页面表：整个系统有一个存储页面表，其描述了物理内存空间的分配使用状况。存储页面表有两种构成方法：①位示图法 ②空闲页面链表法。

5.4 页式管理——静态页式管理

- 分配算法



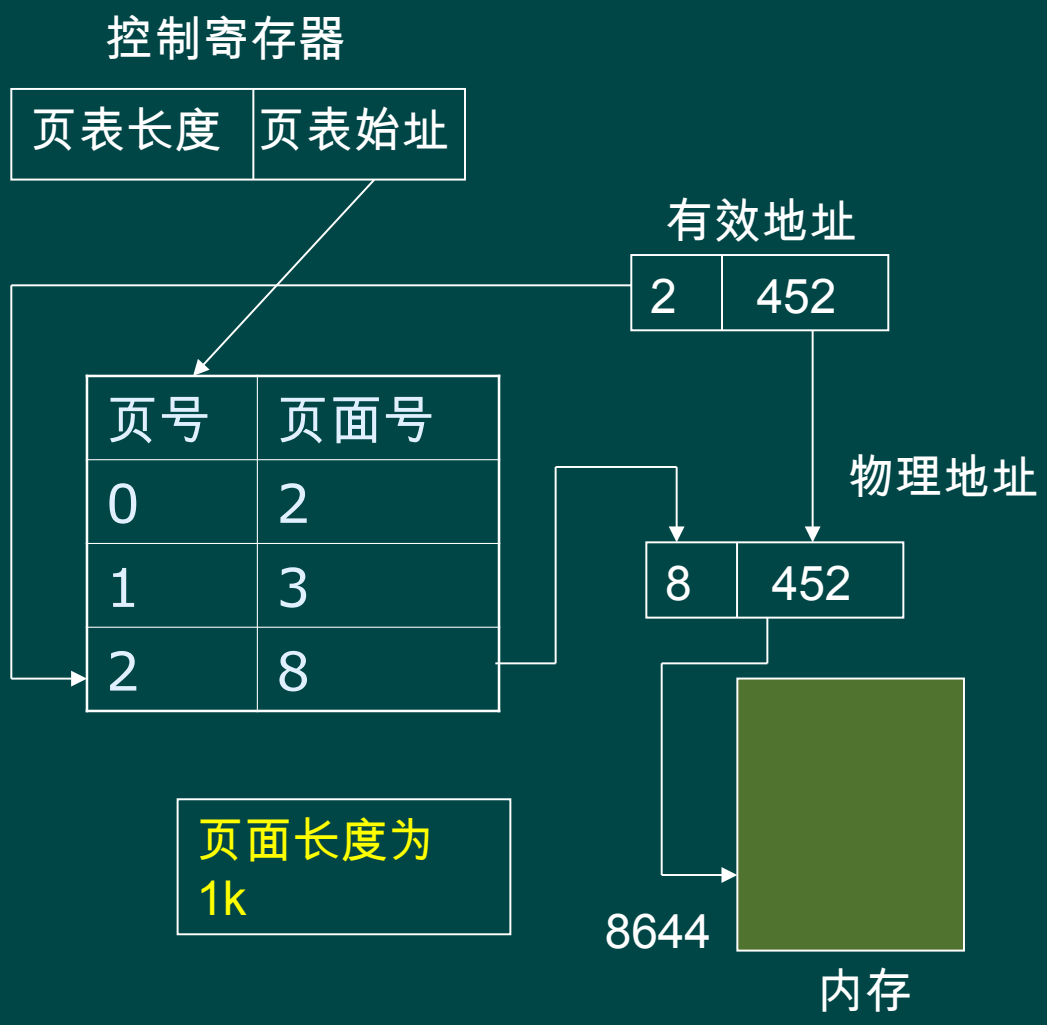
5.4 页式管理——静态页式管理

- 地址转换

1. 初始化页表始址和页表长度控制寄存器

2. 由控制寄存器页表始址可以找到页表所在位置

3. 根据页内偏移量，得到真实物理地址



5.4 页式管理——静态页式管理

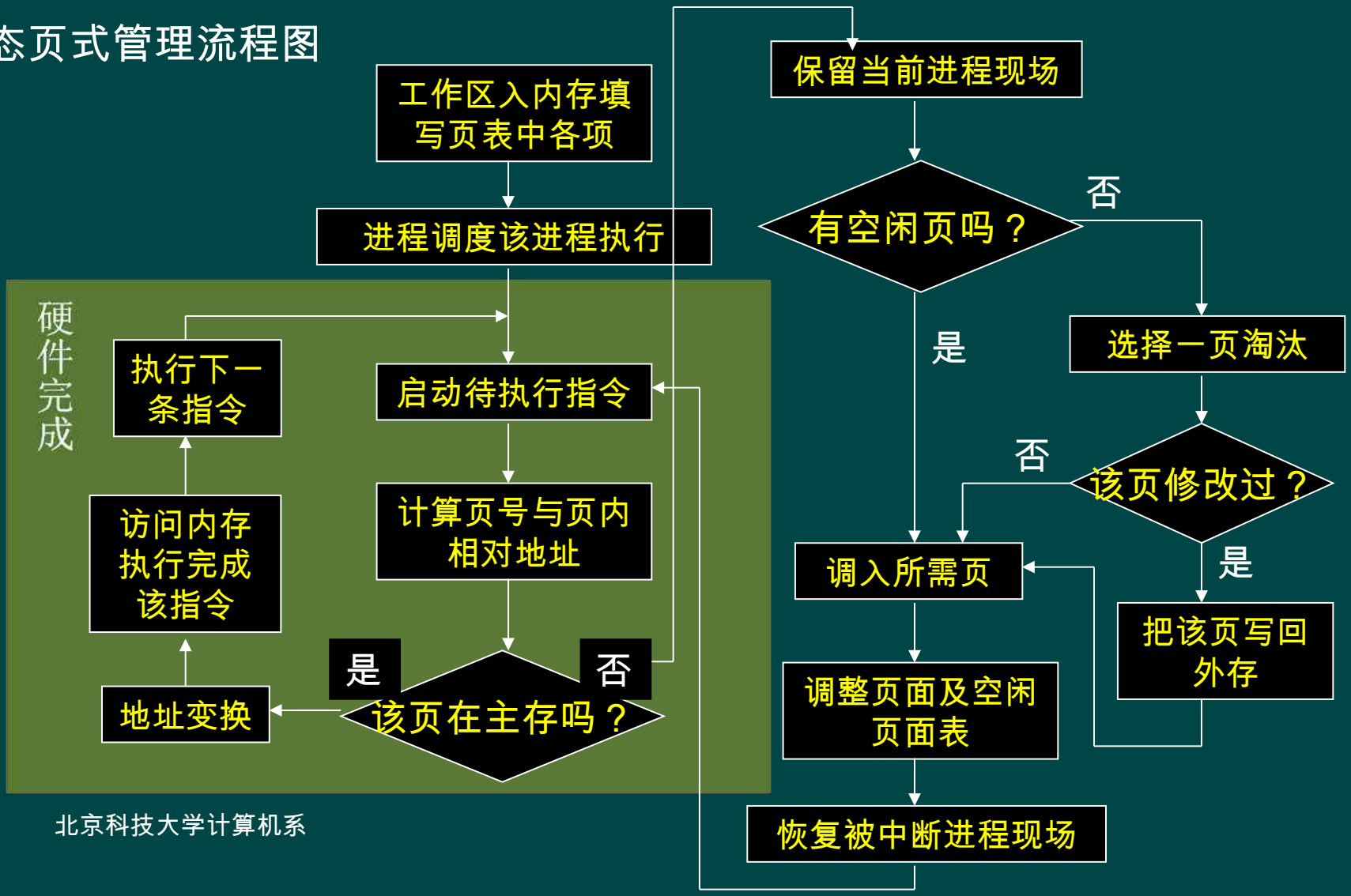
- 静态页面管理优缺点
  - **静态页式管理解决了分区管理时的碎片问题。**
  - **但是，由于静态页式管理要求进程或作业在执行前全部装入内存，如果可用页面数小于用户要求时，该作业或进程只好等待。而且作业和进程的大小仍受内存可用页面数的限制。**

5.4 页式管理——动态页式管理

- 动态页式管理
  - **动态页式管理是在静态页式管理的基础上发展起来的。它分为请求页式管理和预调入页式管理。**
  - **在作业或进程开始执行之前，都不把作业或进程的程序段和数据段一次性地全部装入内存，**
  - **当需要执行某条指令而又发现它不在内存时，从而发生缺页中断，系统将外存中相应的页面调入内存**

5.4 页式管理——动态页式管理

- 动态页式管理流程图



5.4 页式管理——动态页式管理

- 请求页式管理中的置换算法
  - 功能
    - **需要调入页面时，选择内存中哪个物理页面被置换。**
  - 出发点
    - **把未来不再使用的或短期内较少使用的页面调出**
  - 方法
    - **随机淘汰算法**
    - **轮转法和先进先出算法**
    - **最近最久未使用页面置换算法**
    - **理想型淘汰算法**

5.4 页式管理

- 存储保护
  - 页式管理为内存提供两种方式的保护：
    - 1. 地址越界保护

      由地址变换机构中的控制寄存器的值：页表长度和所要访问的虚地址相比较完成 .

    - 2. 通过页表控制对内存信息的操作方式以提供保护 .

5.4 页式管理

- 页式管理优缺点
  - 优点
    - 由于它不要求作业或进程的程序段和数据在内存中连续存放，从而有效地解决了碎片问题。
    - 动态页式管理提供了内存和外存统一管理的虚存实现方式，使用户可以利用的存储空间大大增加。这既提高了主存的利用率，又有利于组织多道程序执行。
  - 缺点
    - 有相应的硬件支持。
    - 增加了系统开销，例如缺页中断处理机，
    - 请求调页的算法如选择不当，有可能产生抖动现象。
    - 虽然消除了碎片，但每个作业或进程的最后一页（碎片）内总有一部分空要求间得不到利用果页面较大。

5.5 段式与段页式管理

- 段式管理的基本思想
  - 把程序按过程（函数）关系分成段，每段有自己的名字。一个用户进程所包含的段对应一个二维线形虚拟空间，也就是一个二维虚拟存储器。段式管理程序以段为单位分配内存，然后通过地址影射机构把段式虚拟地址转换为实际内存物理地址。
  - 程序通过分段 (segmentation) 划分为多个模块，如代码段、数据段、共享段。
    - 可以分别编写和编译
    - 可以针对不同类型的段采取不同的保护
    - 可以按段进行共享，包括通过动态链接进行代码共享.

5.5 段式与段页式管理

- 段式虚存空间
  - 段式管理把一个进程的虚地址空间设计成二维结构，即段号 s 与段内相对地址 w 。
    - 段号与段号之间无顺序关系
    - 段的长度不固定
    - 每个段定义一组逻辑上完整的程序或数据
    - 每个段是一个首地址为零的，连续的一维线性空间
    - 对段式虚地址空间的访问包括两个部分：段名和段内地址

段名	段内地址

5.5 段式与段页式管理

- 段式管理的内存分配与释放
  - **段式管理中以段为单位分配内存，每段分配一个连续的内存区。**分配与释放在进程的执行过程中动态进行 .
    - 当进程要求调入某一段时，内存中有足够的空闲区满足该段的内存要求：**采用动态分区模式空闲区管理方法 .**
    - 内存中没有足够的空闲区满足该段的内存要求：**根据给定的置换算法淘汰内存中在今后一段时间内不再被 CPU 访问的段 .**

5.5 段式与段页式管理

- **段式管理的地址变换**
  - **段式管理程序在进行初始内存分配之前，首先根据用户要求的内存大小为一个进程建立一个段表，以实现动态地址变换和缺段中断处理及存储保护等。段式管理通过段表来进行内存管理的。**

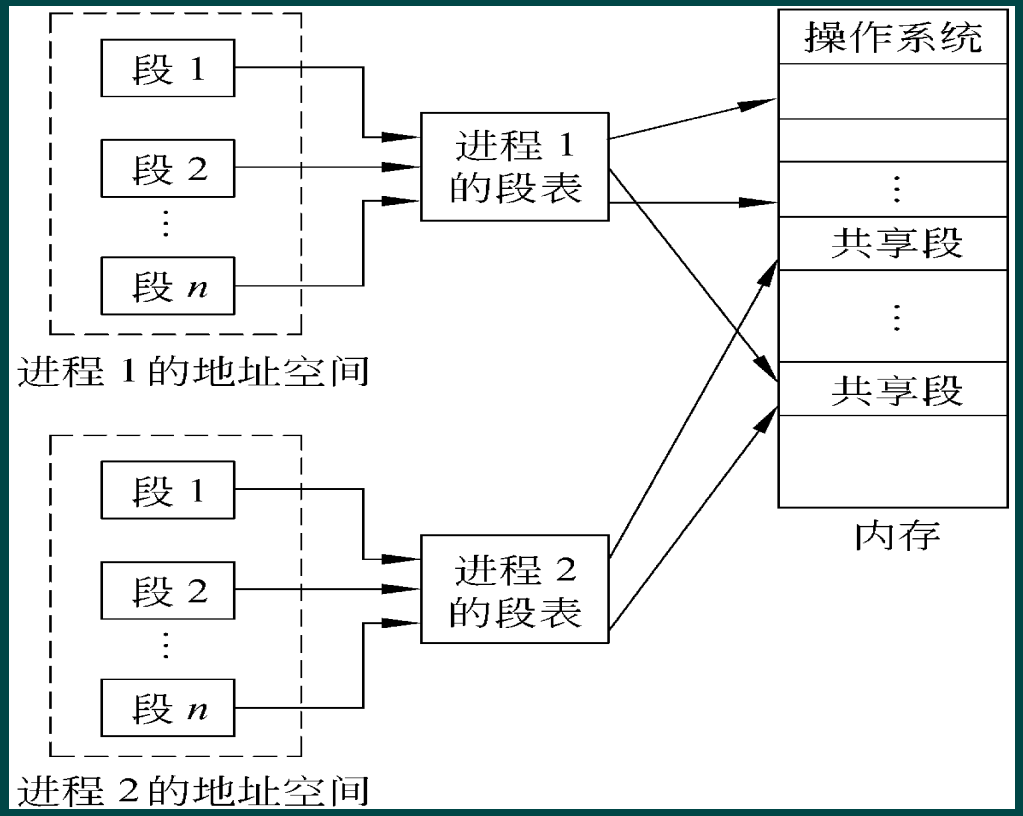
段号	始址	长度	存取方式	内外	访问位
与用户指定的段名一一对应	段在内存或外存的物理地址	段在内存或外存的实际长度	用来对该段进行存取保护的	段现在存储于外存还是内存中	根据淘汰算法的需要而设

5.5 段式与段页式管理

- **动态地址变换**

5.5 段式与段页式管理

- 段的共享与保护
  - **共享：内存中只保留一个副本，供多个用户使用**



5.5 段式与段页式管理

- 段的共享与保护
  - **保护**
    - **地址越界保护法：利用段表中的段长项与虚拟地址中的段内相对地址比较进行。若段内相对地址大于段长，系统就会产生保护中断。**
    - **存取方式控制保护法**

5.5 段式与段页式管理

- 段式管理的优缺点
  - **优点：**
    - **段式管理提供了内外存统一管理的虚存实现**
    - **段长可以根据需要动态增长**
    - **便于对具体逻辑功能的信息段进行共享**
    - **便于实现动态链接**
  - **缺点**
    - **要求较高的硬件支持**
    - **碎片问题**

5.5 段式与段页式管理

- 段页式管理基本思想
  - 段式管理为用户提供了一个二维的虚地址空间，反映了程序的逻辑结构，有利于段的动态增长以及共享和内存保护等，这大大地方便了用户。
  - 而分页系统则有效地克服了碎片，提高了存储器的利用率。
  - 那么两者结合呢？

5.5 段式与段页式管理

- 段页式管理的实现原理
  - 虚拟地址的构成
    - 段页式管理时的进程的虚拟地址空间中的虚拟地址由三部分组成：即段号 s ，页号 P 和页内相对地址 d

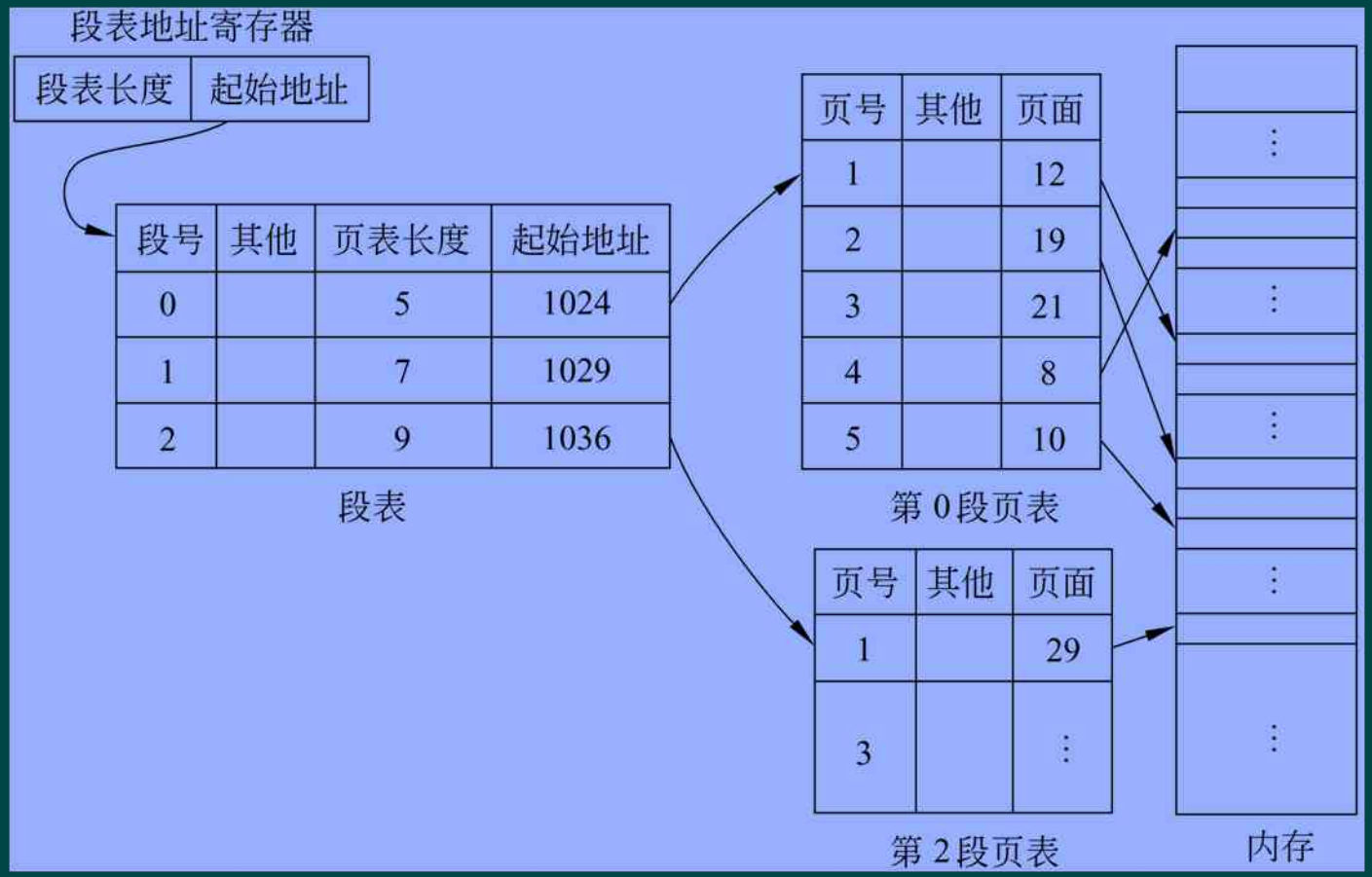
    - 虚拟空间的最小单位是页而不是段

5.5 段式与段页式管理

- 段页式管理的实现原理
  - 段表和页表
    - **为了实现段页式管理，系统必须为每个进程建立一张段表以管理内存分配与释放、缺段处理、存储保护相地址变换等。**
    - **由于一个段又被划分成了若干页，每个段又必须建立一张页表以把段中的虚页变换成内存中的实际页面。**
    - **段表中应有专项指出该段所对应页表的页表始址和页表长度。**

5.5 段式与段页式管理

- 段页式管理中段表、页表与内存的关系

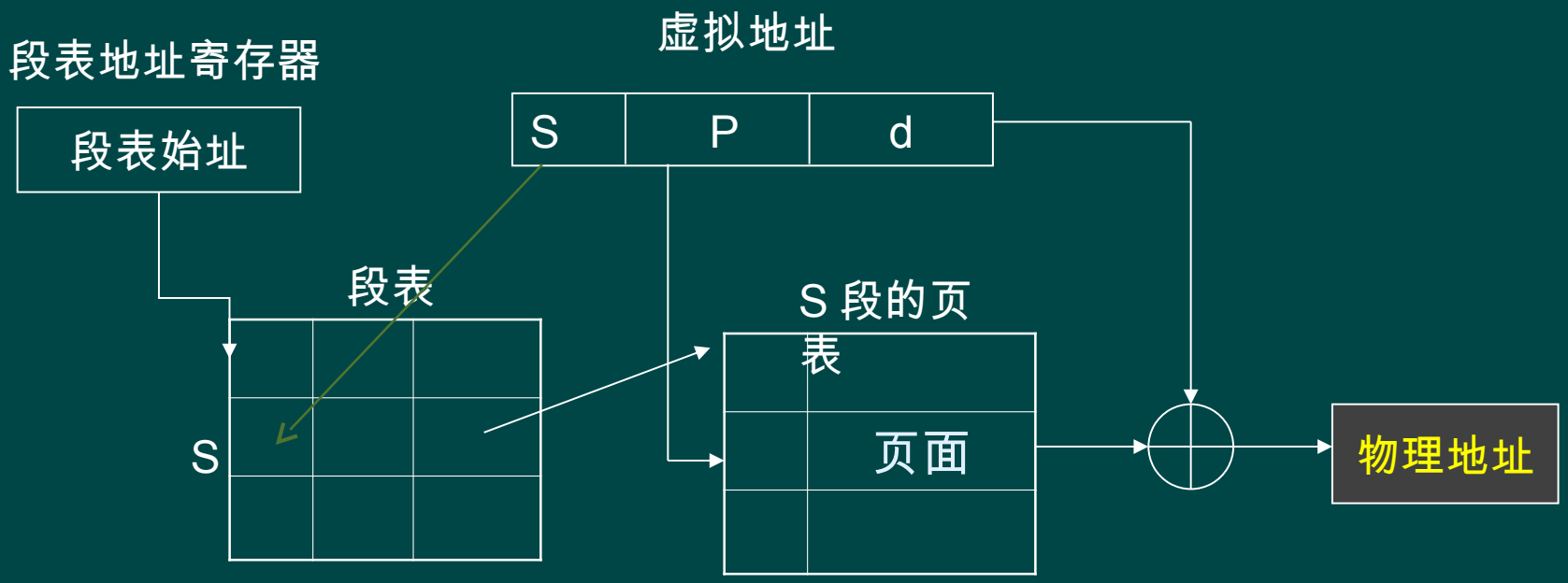


5.5 段式与段页式管理

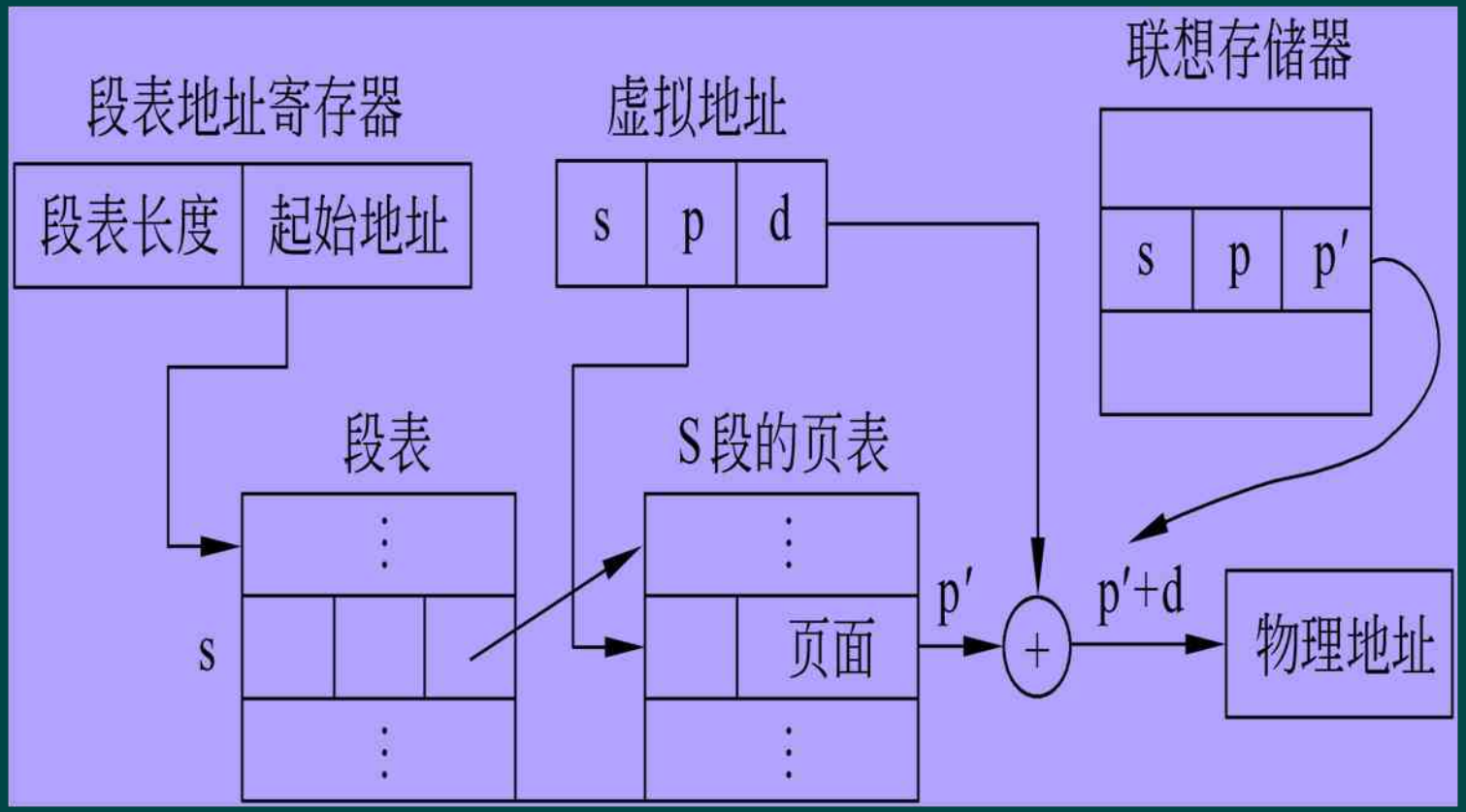
- 动态地址转换
  - 在段页式管理系统中，要对内存中指令或数据进行一次存取的话，至少需要访问三次以上的内存：
    - 第一次是由段表地址寄存器得段表始址后访问段表，由此取出对应段的页表在内存中的地址。
    - 第二次则是访问页表得到所要访问的物理地址。
    - 第三次才能访问真正需要访问的物理单元。

5.5 段式与段页式管理

- 动态地址转换



5.5 段式与段页式管理



5.5 段式与段页式管理

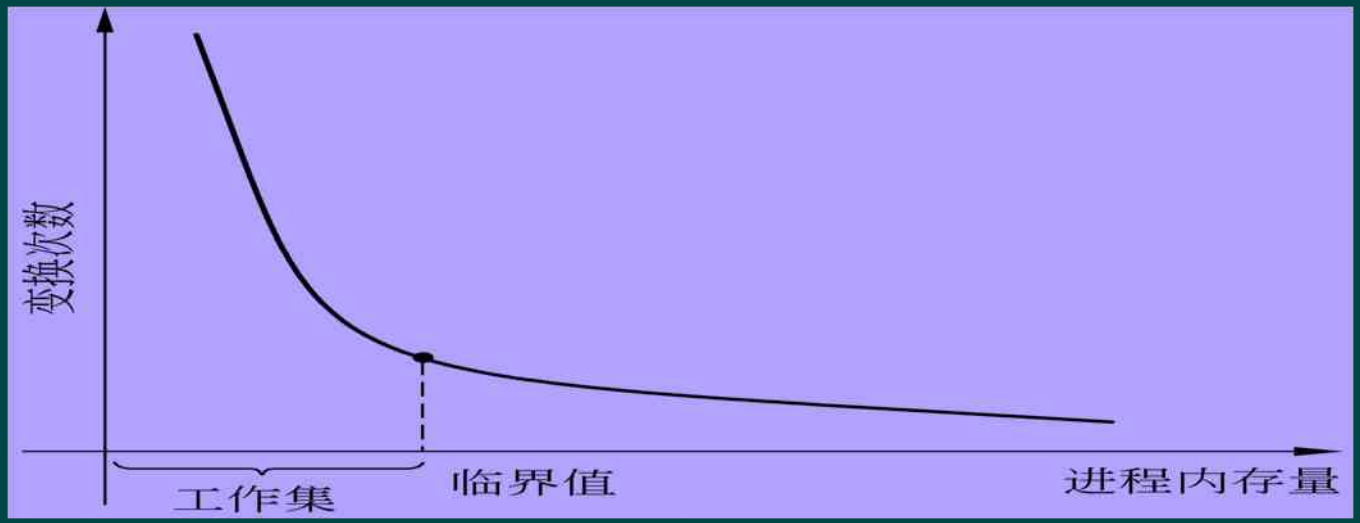
- 段页式管理是段式管理的页式管理方案结合而成的，所以具有它们二者的优点。
- 由于管理软件的增加，复杂性和开销也就随之增加了。
- 需要的硬件以及占用的内存也有所增加。更重要的是，如果不采用联想寄存器的方式提高 CPU 的访内速度，将会使得执行速度大大下降。

5.6 局部性原理和抖动问题

- 局部性原理
  - **指程序在执行过程中的一个较短时期，所执行的指令地址和指令的操作数地址，分别局限于一定区域。还可以表现为：**
    - **时间局部性**
    - **空间局部性**

5.6 局部性原理和抖动问题

- 工作集
  - 任何程序在局部性放入时，都有一个临界值要求。当内存分配小于这个临界值时，内存和外存之间的交换频率将会急剧增加，而内存分配大于这个临界值时，再增加内存分配也不能显著减少交换次数。这个内存要求的临界值被称为工作集。



5.6 局部性原理和抖动问题

- 抖动问题
  - 在虚存中，页面在内存与外存之间频繁调度，以至于调度页面所需时间比进程实际运行的时间还多，此时系统效率急剧下降，甚至导致系统崩溃。这种现象称为颠簸或抖动。
  - 原因：
    - 页面淘汰算法不合理
    - 分配给进程的物理页面数太少

小结

方法 / 功能	单一连续区	分区式		页式		段式	段页式
		固定分区	可变分区	静态	动态		
适用环境	单道	多道		多道		多道	多道
虚拟空间	一维	一维		一维		二维	二维
重定位方式	静态	静态 动态		动态		动态	动态
分配方式	静态分配连续区	静态动态分配连续区		静态或动态页为单位非连续		动态分配段为单位非连续	动态分配页为单位非连续
释放	执行完成后全部释放	执行完成后全部释放	分区释放	执行完成后释放	淘汰与执行完后释放	淘汰与执行完成后释放	淘汰与执行完成后释放
保护	越界保护或没有	越界保护与保护键		越界保护与控制权保护		同左	同左
内存扩充	覆盖与交换技术	同左		同左	外存、内存统一管理的虚存	同左	同左
共享	不能	不能		较难		方便	方便
硬件支持	保护用寄存器	保护用寄存器 同左 加重定位机构		地址变换机构 中断机构 保护机构		段式地址变换机，保护与中断，动态连接机构	同左