存储管理

主讲教师：赵霞

内存——“进程”的地盘

- 程序执行前加载到内存，OS如何给程序分配空间？
- 编程序时的代码段、数据段，如何对应到物理内存中的单元呢？
- OS管理内存的基本思路是什么？
- 如何评价内存管理的效率？
- 内存不够用的时候，怎么办？
- 虚拟内存是怎么实现的？这种技术为什么是有效的？

存储管理

- 目的与要求
  - 了解连续存储分配
  - 掌握页式存储管理
  - 了解段/段页式管理
  - 掌握虚存管理
  - 了解各种页面替换策略及实用的综合策略
- 重点与难点
  - 连续可变存储管理
  - 页式存储管理
  - 虚存管理系统内存访问过程
  - 固定驻留集算法和SWS等实用动态驻留集算法
- 作业
  - 5.17,　5.18
  - 制作本章思维导图

课后阅读与思考

- 教材
  - 第5章
- Modern Operating System (2nd edition)
- (现代操作系统)
  - Chapter4 Memory management

存储管理的目的

- 操作系统的重要使命
  - 提高资源利用率
  - 采用多道程序设计技术
- 存储管理
  - 合理利用内存，支持多道程序并发执行
  - 使得CPU和I/O设备利用率最高

存储管理的功能

- 尽可能方便用户使用
  - 自动装入用户程序
  - 用户程序中不必考虑硬件细节
- 解决程序空间比实际内存空间大的问题
- 程序在执行时可以动态扩缩
- 内存存取速度快
- 了解有关资源的使用状况

存储管理的任务

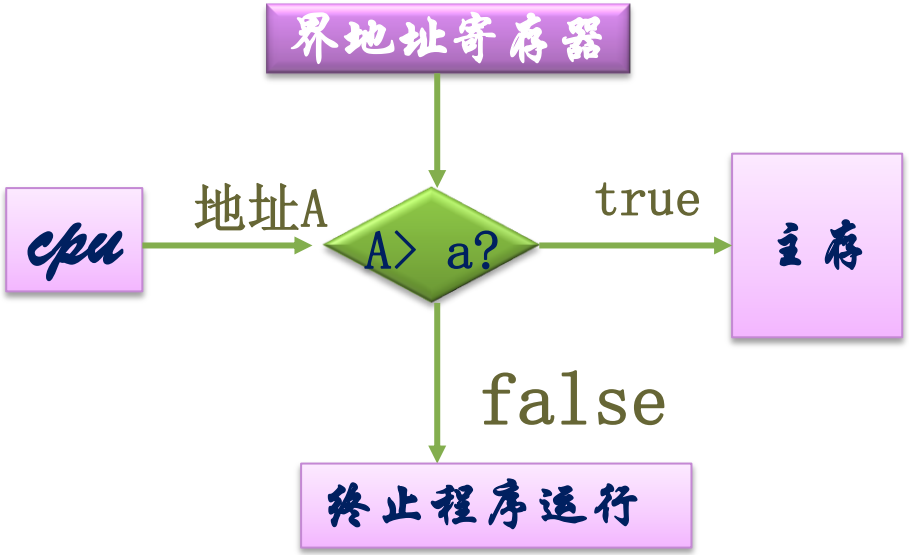
- 内存空间的管理、分配与回收
  - 记录内存的使用情况
  - 内存空间划分
    - 静态或动态，等长或不等长
- 存储共享
  - 两个或多个进程共用内存中相同区域
  - 节省内存空间，提高内存利用率
  - 实现进程通信（数据共享）
- 存储保护
  - 内存中的各道程序，只能访问它自己的区域，避免各道程序间相互干扰
- 内存扩充
  - 用户得到比实际内存容量大的多的内存空间

存储管理

- 连续空间分配
  - 单道连续分配
  - 多道固定划分法
  - 多道连续可变划分法
- 不连续空间分配
  - 页式管理
  - 段式管理
  - 段页式管理
- 虚存管理
  - 页式虚存的基本思想
  - 页式虚存管理的实现
  - 页面替换策略

单道连续分配

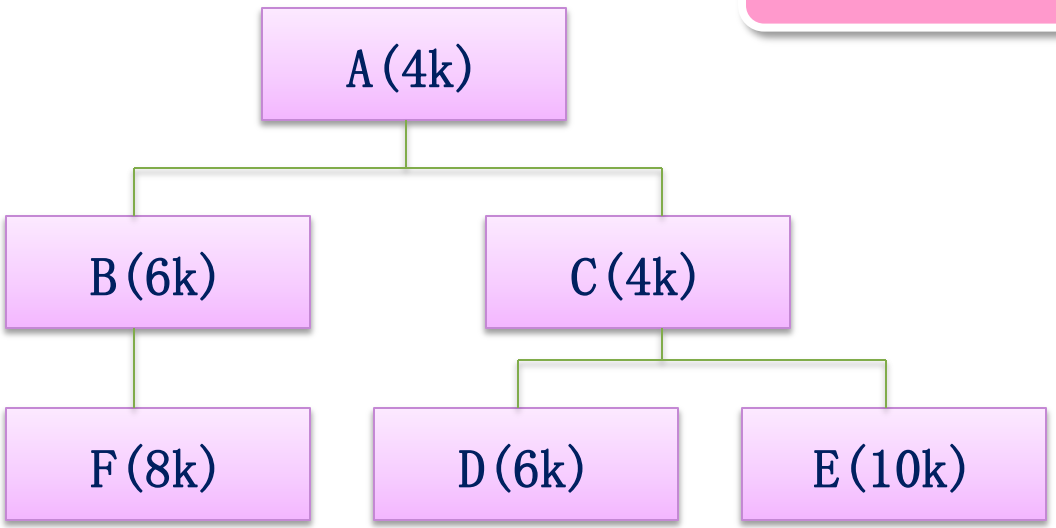
- 任一时刻内存只有一道作业，该作业连续存放于内存中
- 越界检查机构
  - 用户程序每访问一次主存，越界检查机构将访问的地址与界地址寄存器中的值比较
  - 若越界，则终止其执行



覆盖

- 因内存小于作业的程序空间而引入覆盖
- 将用户空间划分成一个固定区和多个覆盖区
- 主程序放固定区，依次调用的子程序则放在同一个覆盖区
- 操作系统提供覆盖系统调用函数，由用户编程序时决定覆盖策略

用户负担重



交换

- 在什么调度策略下，需要存储交换技术？
  - 采用时间片轮转调度或者可剥夺优先级调度

- 基本思想
  - 将处于等待状态(等I/O)或就绪(等CPU)状态的进程从主存换出到辅存——换出
  - 把将要执行的进程移入主存——换入
- 为了支持交换，必须在系统空间设立I/O缓冲区
  - 不影响等待I/O的作业被换出
- 交换花费时间与交换信息量成正比
  - 磁鼓平均延迟时间为8ms，传输速度250K字/秒
  - 用户空间为20K，则一次交换的时间

$$2\times\left(8+10^3\times\frac{20}{250}\right)=164ms$$

交换技术与覆盖技术的对比

- 共同点
  - 都是内存扩充技术
- 不同点

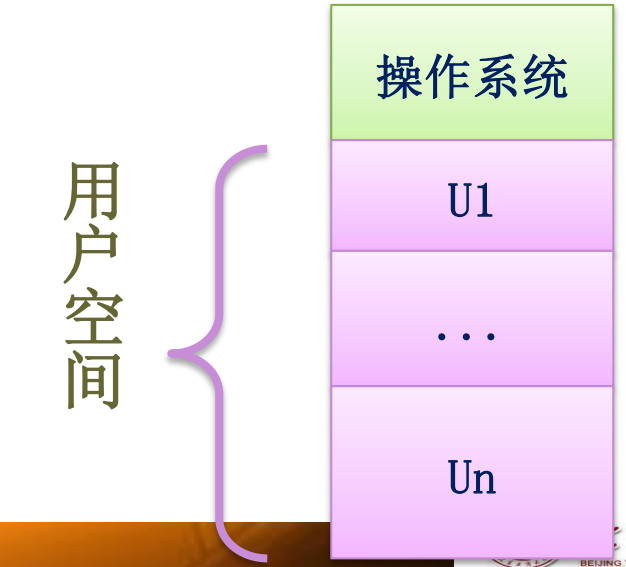
要点	覆盖技术	交换技术	虚拟存储技术
交换的粒度	进程或程序内部	整个进程空间	进程内的页面或段
对程序员的透明度	需要程序员给出程序段的逻辑覆盖结构	不需要程序员参与	不需要程序员参与

存储管理

- 连续空间分配
  - 单道连续分配
  - 多道固定划分法
  - 多道连续可变划分法
- 不连续空间分配
  - 页式管理
  - 段式管理
  - 段页式管理
- 虚存管理
  - 页式虚存的基本思想
  - 页式虚存管理的实现
  - 页面替换策略

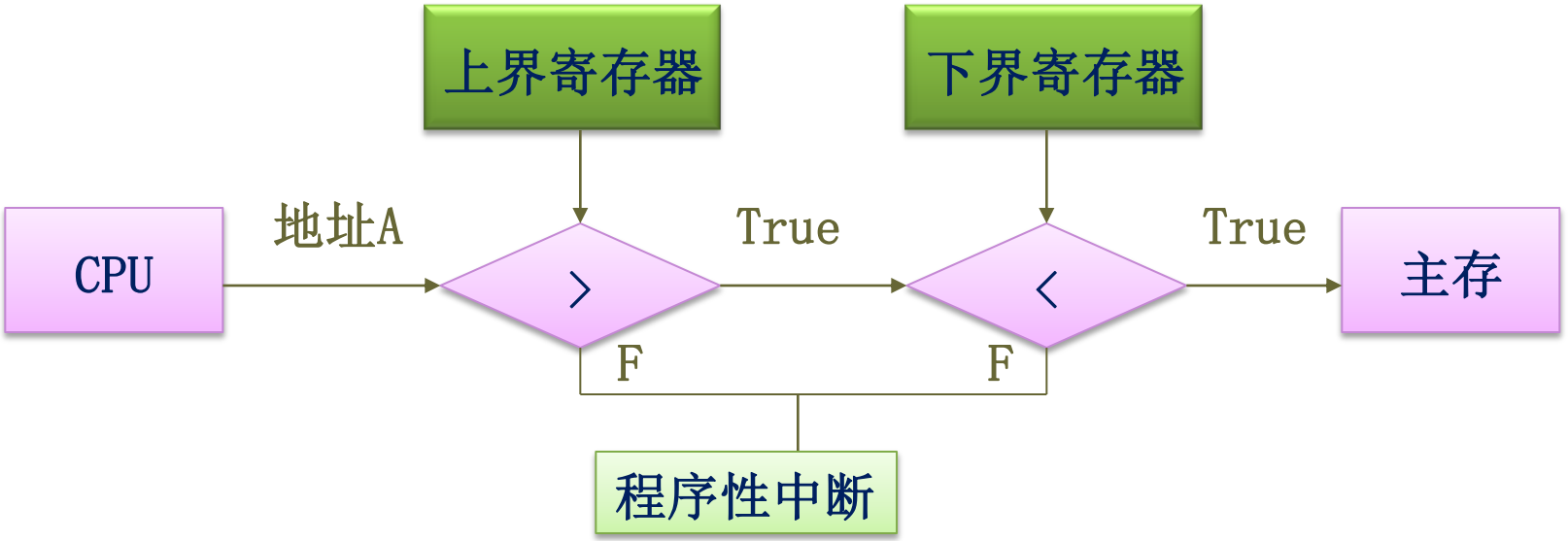
多道固定划分法

- 任一时刻内存可有多道作业，每道作业连续存放于内存
- 硬件提供界址寄存器和越界检查机构做存储保护
- 存储划分
  - 将用户内存空间分成长度固定的若干块
- 地址重定位
  - 目标代码中相对于0地址开始的所有 地址变换成其所在的主存物理地址
  - 静态重定位
    - 加载代码时变换
  - 动态重定位
    - 执行访存指令时



空间保护机制

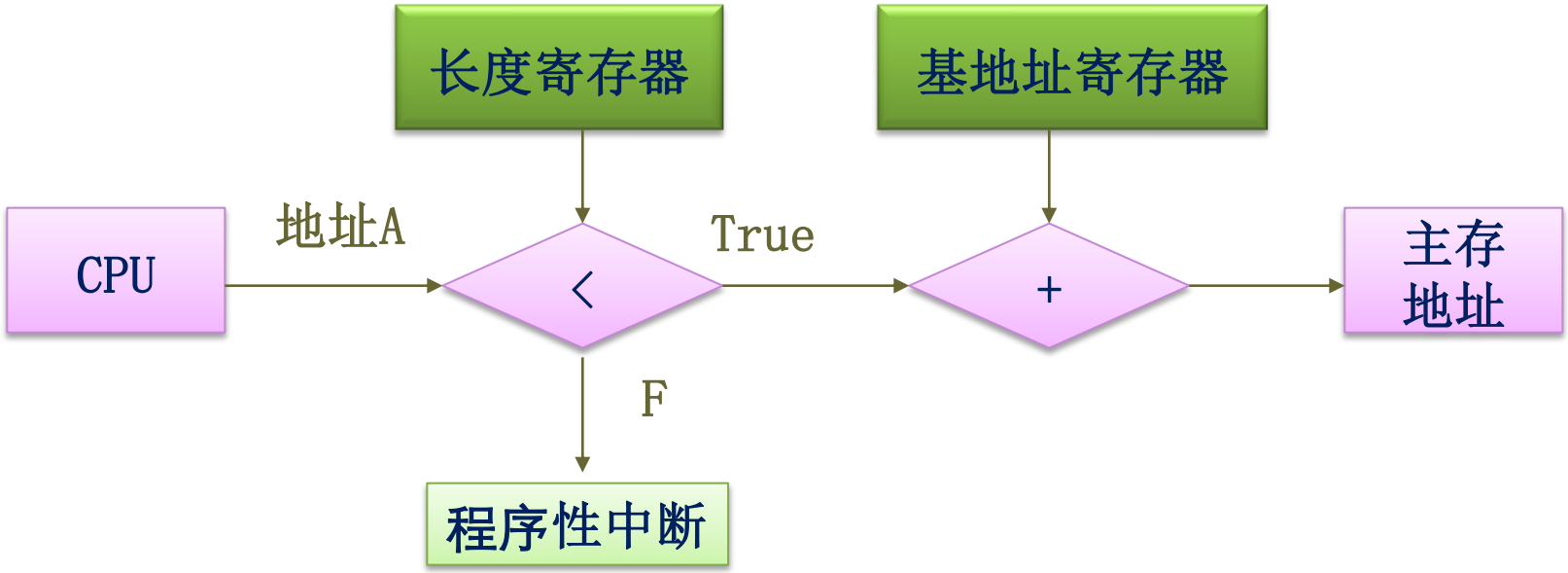
- 上下界寄存器与地址检查机构
  - 作业的上下界地址送上下界寄存器，每次内存访问时，地址检查机构作越界检查
  - 作业程序是绝对地址或静态可浮动



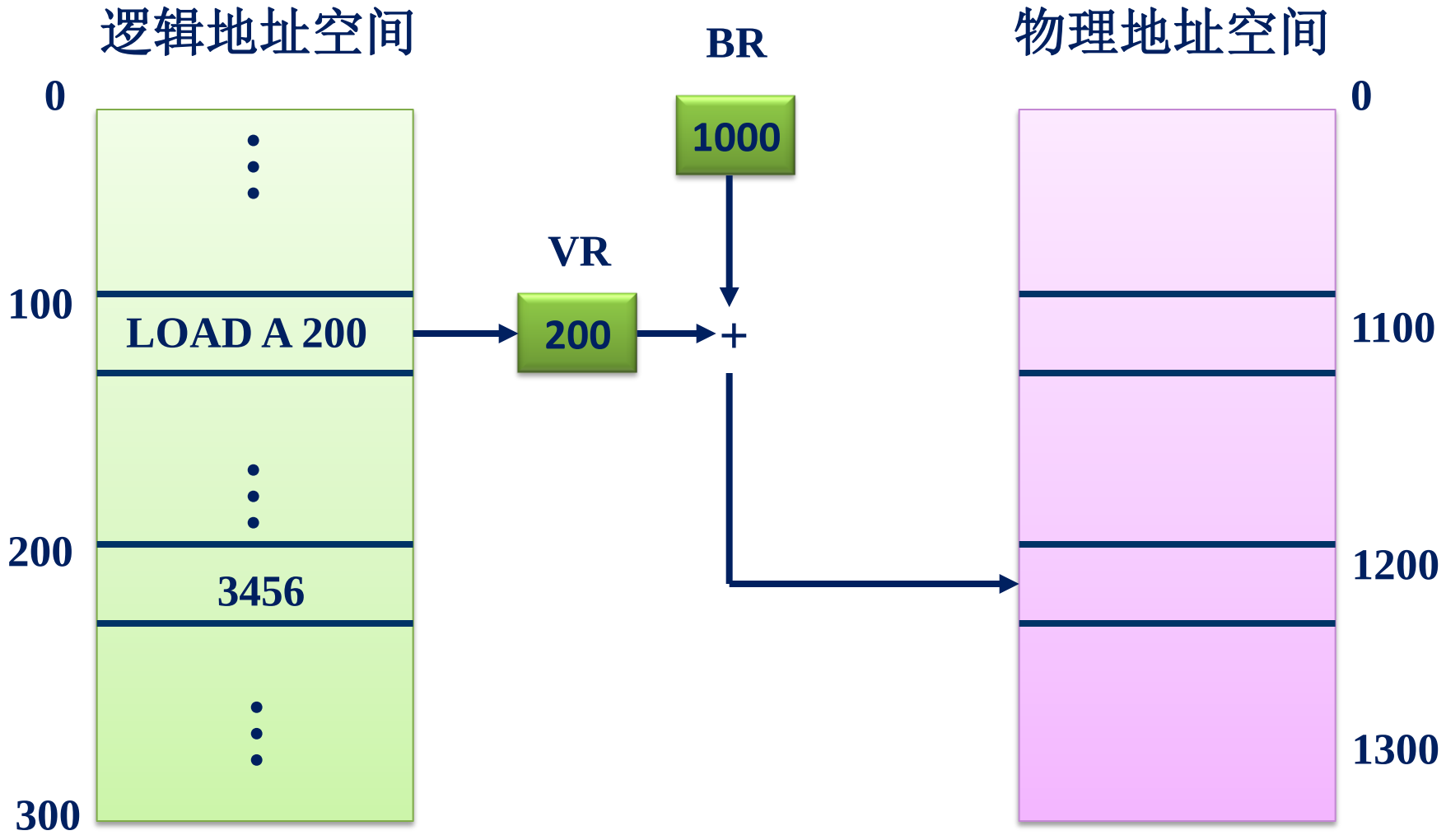
空间保护机制

基址寄存器

- 当作业被调度运行时，将作业所占内存基址及长度送基址、长度寄存器
- 访问内存时先看访问地址是否小于长度， 然后+基址进行访存
- 用户程序代码是可动态浮动的

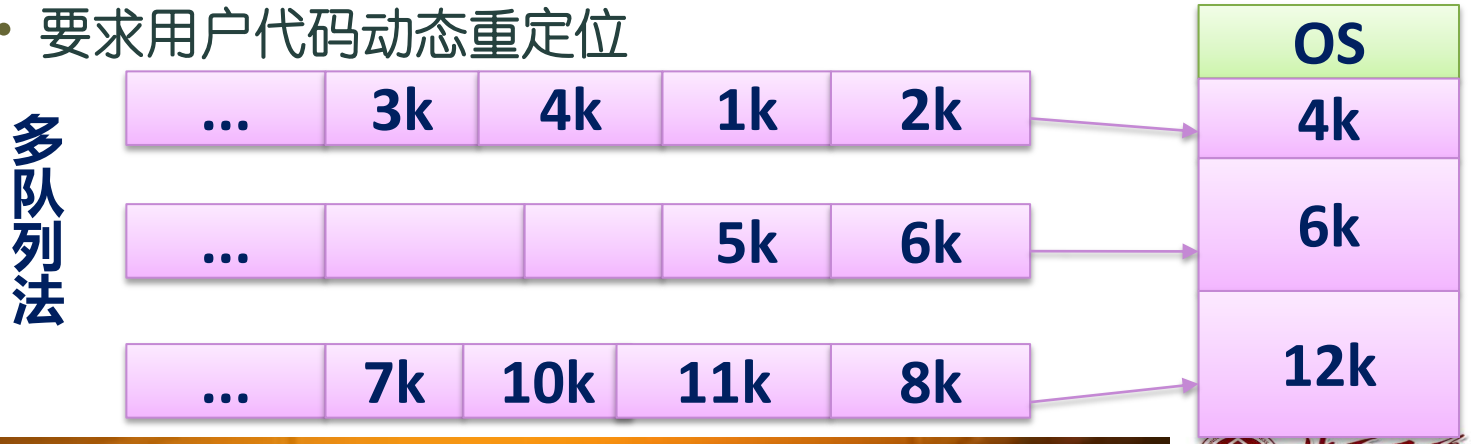


动态地址转换


逻辑地址空间
BR
物理地址空间
0
1000
0
100
VR
LOAD A 200
200
+
1100
200
3456
1200
1300
300

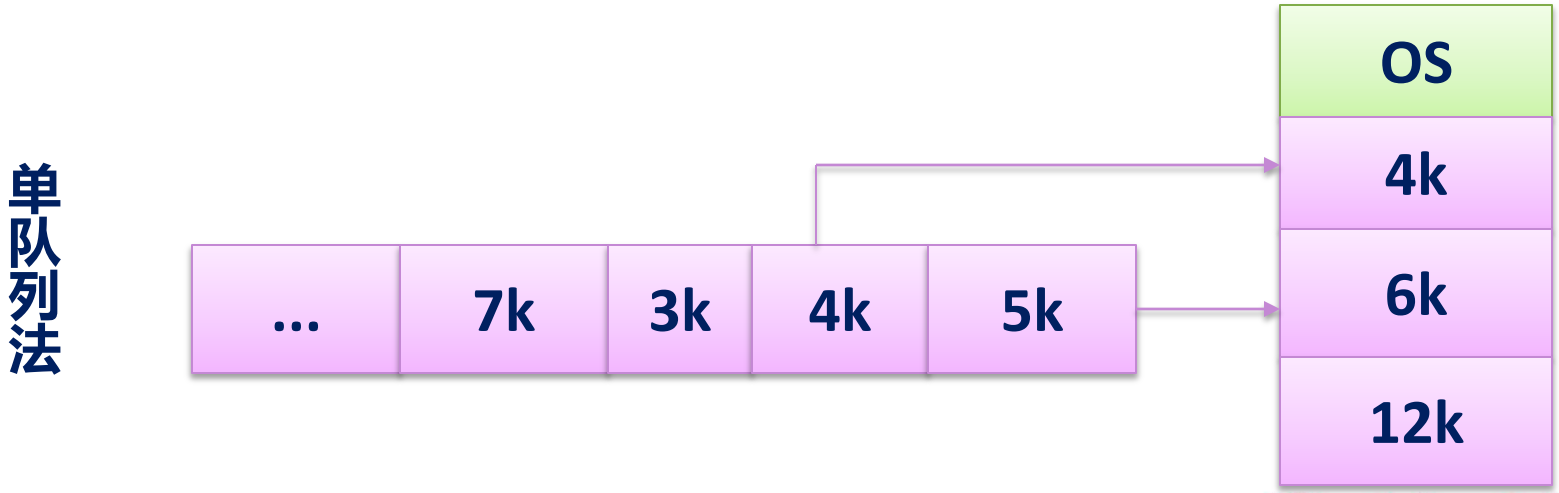
作业存储调度

- 多队列法
  - 每个存储区对应一个作业队列
  - 作业按照大小在对应队列排队
  - 各队列可以采用不同的调度方法
    - 空间利用率低
  - 各队列统一竞争所有内存
    - 要求用户代码静态重定位
  - 采用交换技术
    - 要求用户代码动态重定位



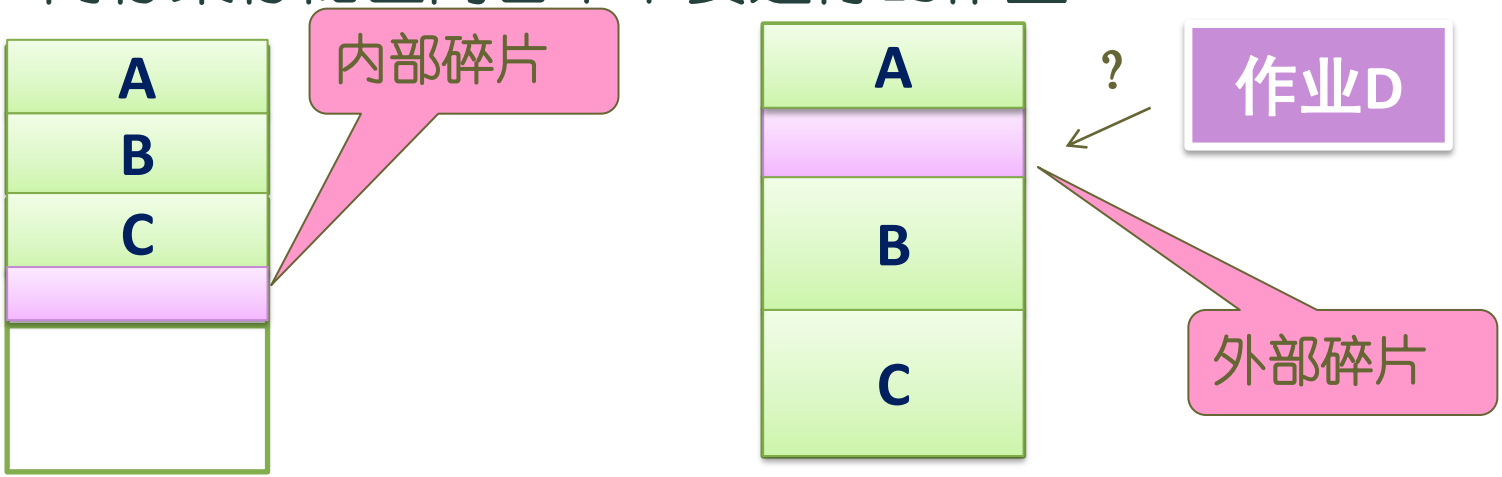
作业存储调度

- 单队列法
  - 系统中仅有一个队列
  - 作业按照先后到达顺序进入队列
  - 作业调度灵活，可采用交换技术



存储碎片

- Memory Fragmentation
  - 进入内存的作业长度短于存储块大小，存在未加利用的存储空间，形成浪费
- 内部碎片
  - 内存某存储区间大于其存放作业空间的部分
- 外部碎片
  - 内存某存储区间容不下要运行的作业



存储管理

- 连续空间分配
  - 单道连续分配
  - 多道固定划分法
  - 多道连续可变划分法
- 不连续空间分配
  - 页式管理
  - 段式管理
  - 段页式管理
- 虚存管理
  - 页式虚存的基本思想
  - 页式虚存管理的实现
  - 页面替换策略

多道连续可变划分法

- 对用户存储区实施动态分割，改善空间利用率
- 管理方法
  - 系统设置一个空闲块队列
  - 初始状态时队列中只有一个连续的空闲块
  - 作业到达后，以某种策略分配空间
  - 作业撤离时，将释放的空间加入空闲队列

举例

- 假设5个作业的同时到达，进入内存后的作业

作业到来次序	所需存储量	运行时间（ms）
1	60	10
2	100	5
3	30	20
4	70	8
5	50	15

0ms : J1进入内存
: J2进入
: J3进入
5ms : J2退出内存
: J4进入
10ms : J1退出
: J5进入
13ms : J4退出
20ms : J3退出
25ms : J5退出

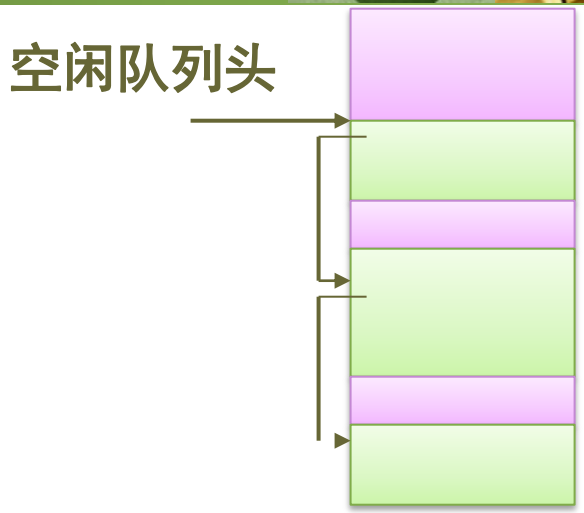
内存分配算法

- 分配策略
  - 找到合适空闲块，将作业所需大小空间分给作业，而剩余部分挂入空闲队列
- 分配策略
  - 首次满足法 (First Fit)
    - 按分区的地址先后次序，选择第一个满足条件的内存块
    - 最快
  - 最佳满足法 (Best Fit)
    - 按分区从小到大的顺序，选择最接近作业大小的内存块
    - 内存利用率高
  - 最大满足法 (Largest Fit)
    - 按分区从大到小的顺序，选择满足条件的最大的内存块
    - 极端情况

内存回收算法

- 回收：
  - 当作业结束时，收回作业所占空间，将此块链入空闲队列
  - 若空闲队列中原来有与此块的相邻块，则把这些块合并成一个大连续块

可用空间的管理



- 数据结构
  - 数组
    - 数组项=用户空间总量/基本分配单位
  - 链表
    - 可用块的低地址部分设置两个域
      - 指针域：指向下一块
      - 长度域：本块长度
- 处理外部碎片
  - 移动作业位置，将零散的空闲块连接成大块
  - 要求作业动态可浮动
  - 时间空间开销大

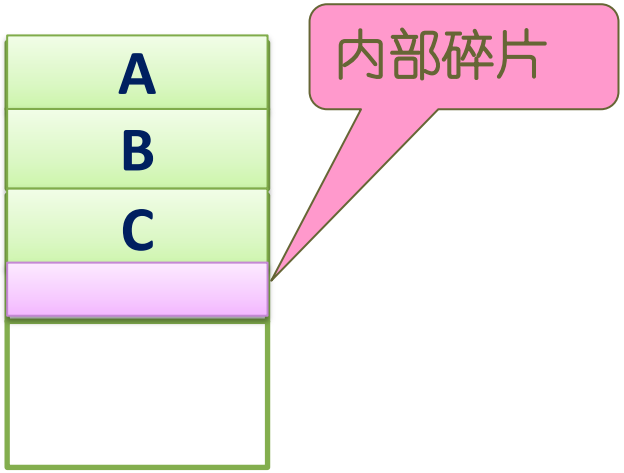
存储管理

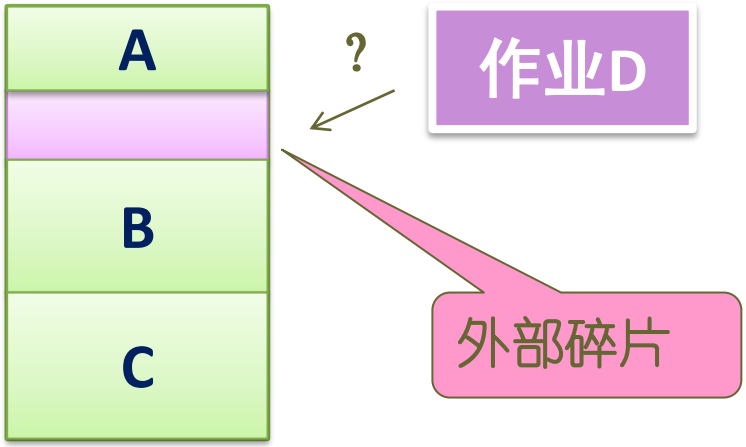
- 连续空间分配
  - 单道连续分配
  - 多道固定划分法
  - 多道连续可变划分法
- 不连续空间分配
  - 页式管理
  - 段式管理
  - 段页式管理
- 虚存管理
  - 页式虚存的基本思想
  - 页式虚存管理的实现
  - 页面替换策略

连续空间分配的问题

- 大量碎片
- 内存利用率低

如何减小/少碎片？
提高利用率？



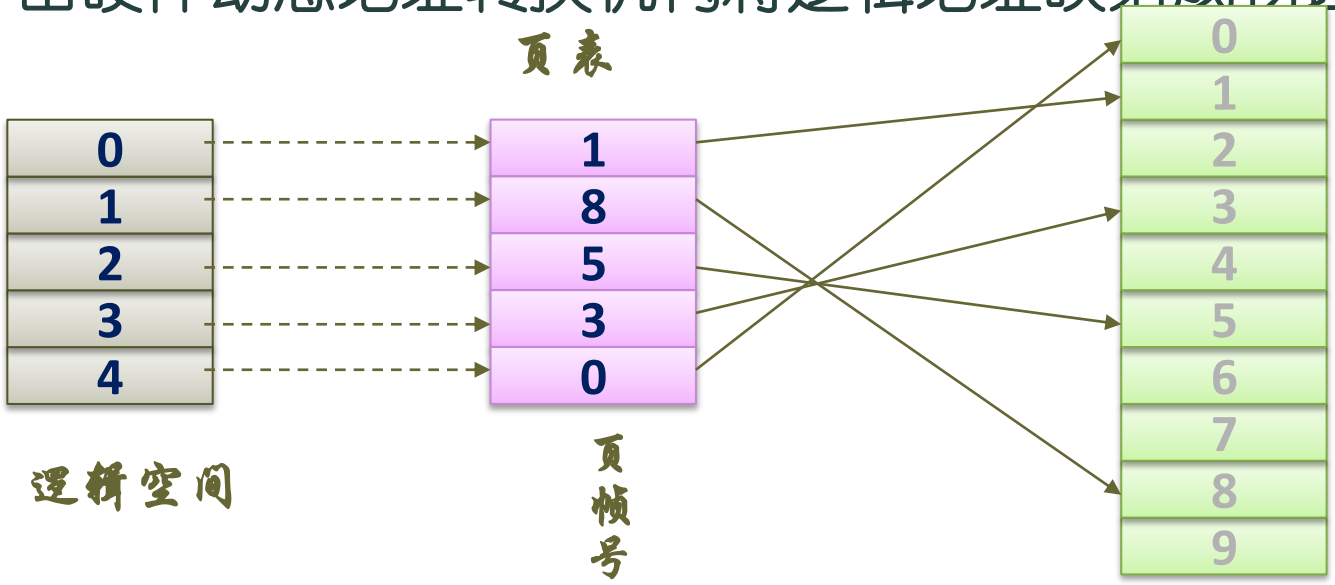


页式管理

- 基本思想
  - 作业(进程)分成页面
  - 内存也划分成页帧
  - 将作业(进程)页面不连续地分布到内存页面
- 空间安排
  - 逻辑空间(地址)
    - 进程空间(地址)
  - 物理空间(地址)
    - 内存空间(地址)
  - 页 (page)
    - 逻辑空间等分出的每个区域
  - 页帧 (page frame)
    - 物理内存空间等分出的区域

页式管理

- 页表
  - 放在系统空间的页表区
  - 存放逻辑页与物理页帧的对应关系
  - PCB表中有指针指向页表
- 地址映射
  - 由硬件动态地址转换机构将逻辑地址映射成物理地址



页式管理

- 分配
  - 初始时，所有页帧都在空闲队列中
  - 当用户进程被创建时，系统按需要量从空闲队列获得相应量的页帧
- 回收
  - 当进程结束时，系统回收它的所有物理页帧入空闲队列

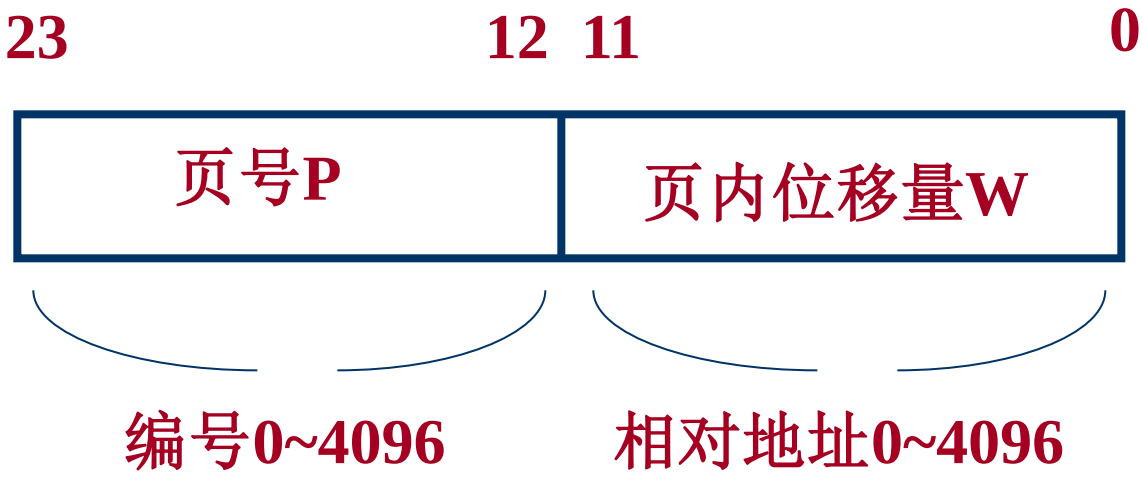
页式管理——页的地址编号

- 假定地址用m个二进制表示，其中页内地址部分占用n个二进制位
  - 每一块的长度是2^n，即每一页有2^n个字节
  - 页号部分为m-n位，则最大程序允许有2^{m-n}个页面
  - 通常，一页的大小为2的整数次幂，因此，地址的高位部分为页号，低位部分为页内地址
  - 逻辑地址连续
    - 第0页的地址范围：0:0 ~ 0: FF…F　(n/4 个F)
    - 第1页的地址范围：1:0 ~ 1: FF…F　(n/4 个F)
    - …

m - n	n

页式管理——页的大小

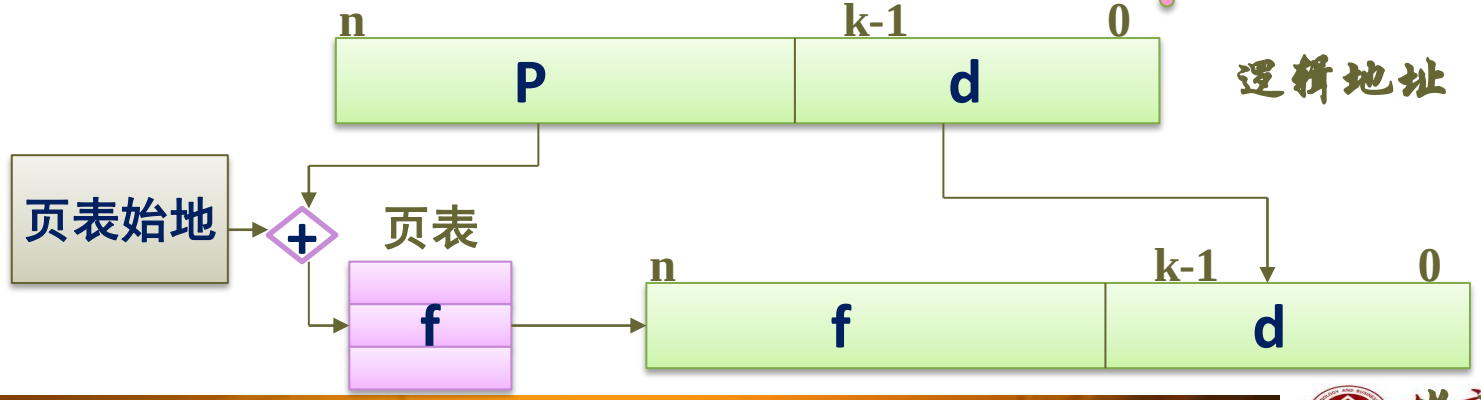
- 通常是：几KB到几十KB
  - 小->内碎片小；
  - 大->内碎片大，页表短，管理开销小，交换时对外存I/O效率高
  - 和目前计算机的物理内存大小有关



动态地址转换机制

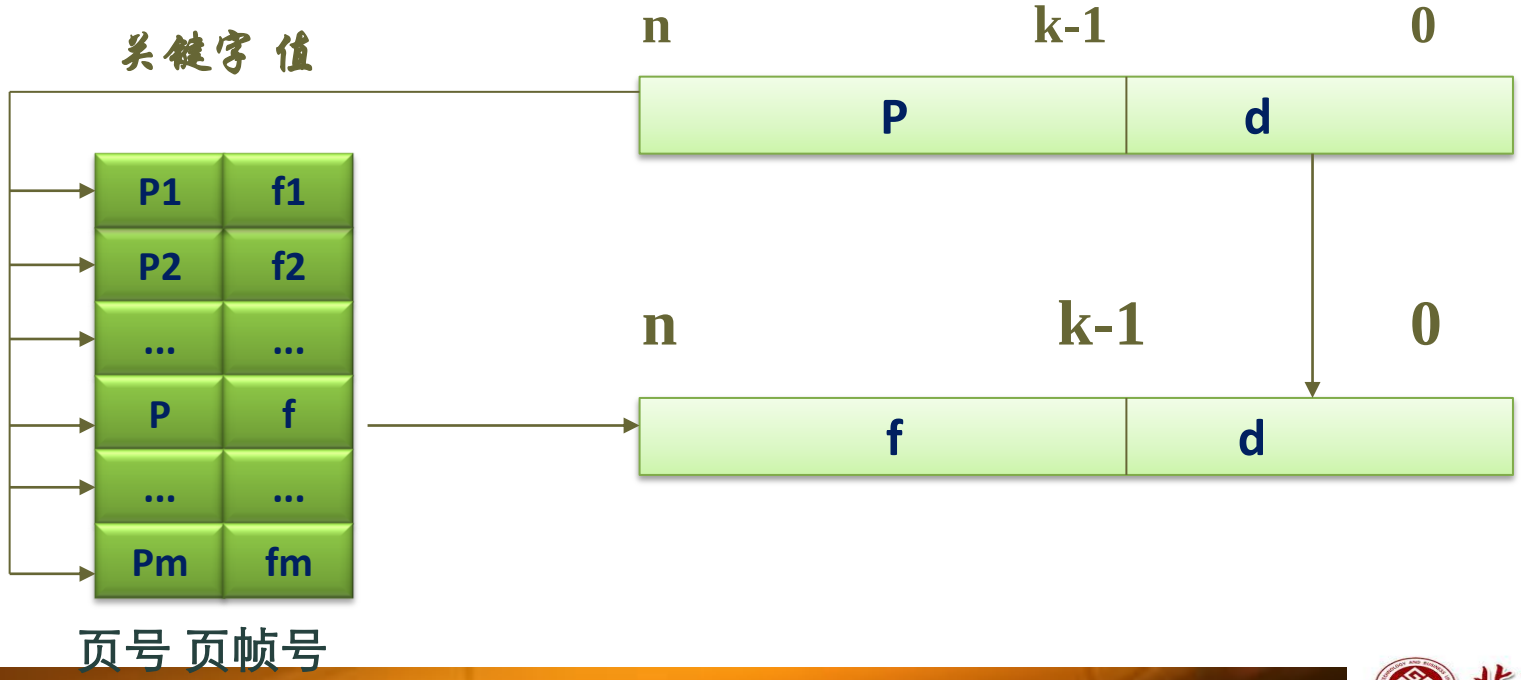
- 地址结构
  - 逻辑地址 = p(页号).d(页内位移)
  - 物理地址 = f(页帧号).d(同上)
  - p = 线性逻辑地址 / 页面大小
  - d = 线性逻辑地址 - p*页面大小
- 页面大小
  - 页面大小取成2的k次幂(k是正整数)
  - 获取p和d的除、乘法只要通过位移实现

两次内存访问降低性能！



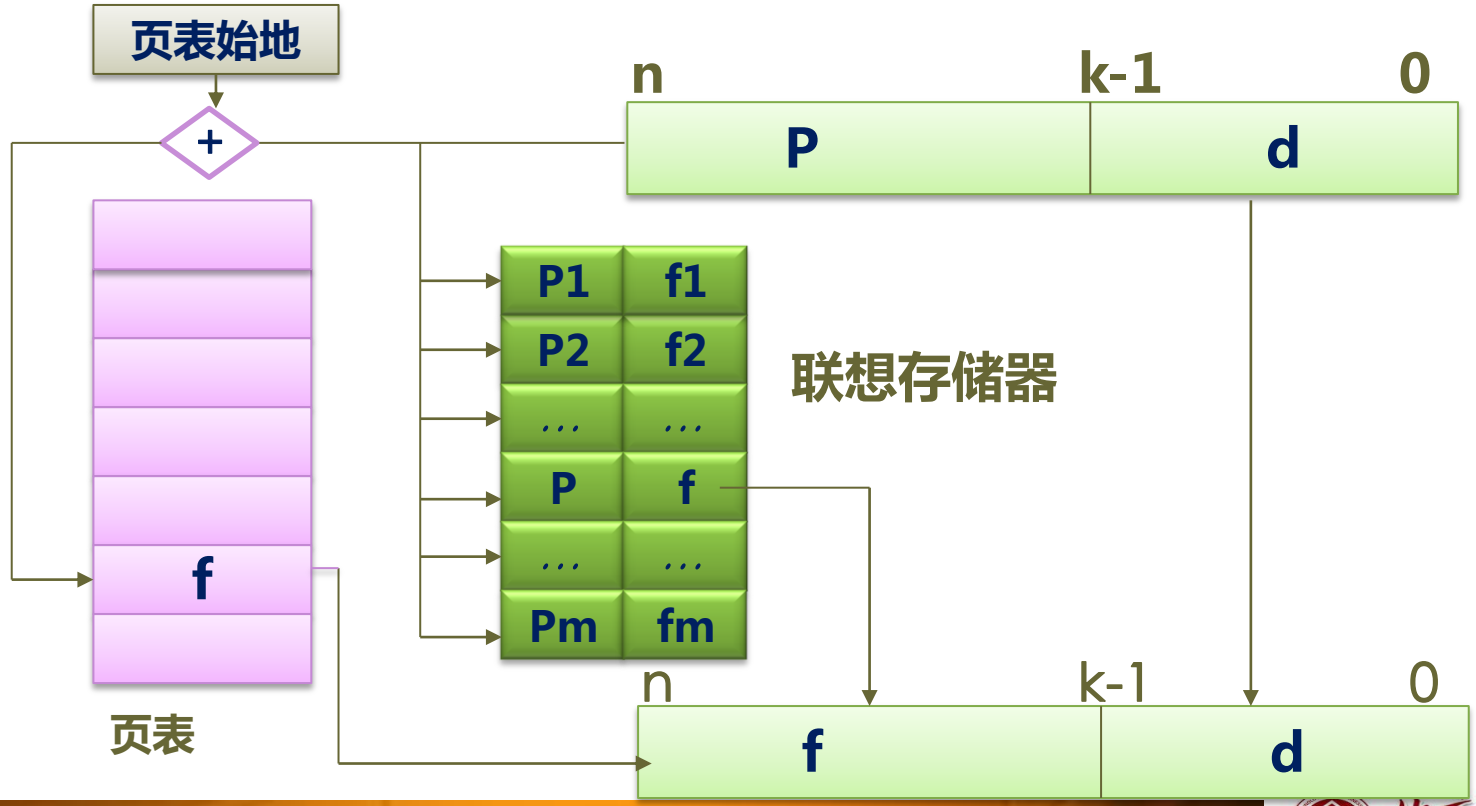
动态地址转换机制

- 联想存储器
  - CPU内有一个高速存储体（TLB）
  - 存放当前访问最频繁的页面的页号
  - 记录页号→页帧号的对应关系
- 工作原理
  - 用逻辑地址和页表项关键字匹配，直接获得页帧号



动态地址转换机制

- 地址转换的一般过程
  - 联想存储器可以看成是页表的cache
  - 同时匹配联系存储器和查找页表



动态地址转换机制

- 命中率
  - 选用8-12项组成的联想存储器，并采用适当的替换策略，在联想存储器中匹配成功的可能性可达80-90%
- 平均访存时间
  - 假设先查联想存储器再查页表，命中率为80%
  - 设访存时间为750ns，检索联想存储器的时间为50ns
  - 则 平均访存时间为：

$$80\% \times (750+50) + 20\% \times (750+50+750) = 950\text{ns}$$

- 比没有联想存储器提高性能：$\dfrac{750\times 2-950}{750\times 2}=36.7\%$
- 比连续存储管理降低性能：$\dfrac{950-750}{750}=26.7\%$

页式管理

- 可用空间管理
  - 用位图（bitmap）数组或空闲页帧链来管理可用页帧
- 存储保护
  - 越界保护
    - 设置页表长度寄存器，查页表前，先检查页号是否越界
  - 操作访问保护
    - 在每个页表项中增设一存储保护域
    - 说明对该页的访问权限
    - 每一个对该页存储的访问都首先要比照是否满足该页访问权限的说明
      - -满足则访问，否则报错

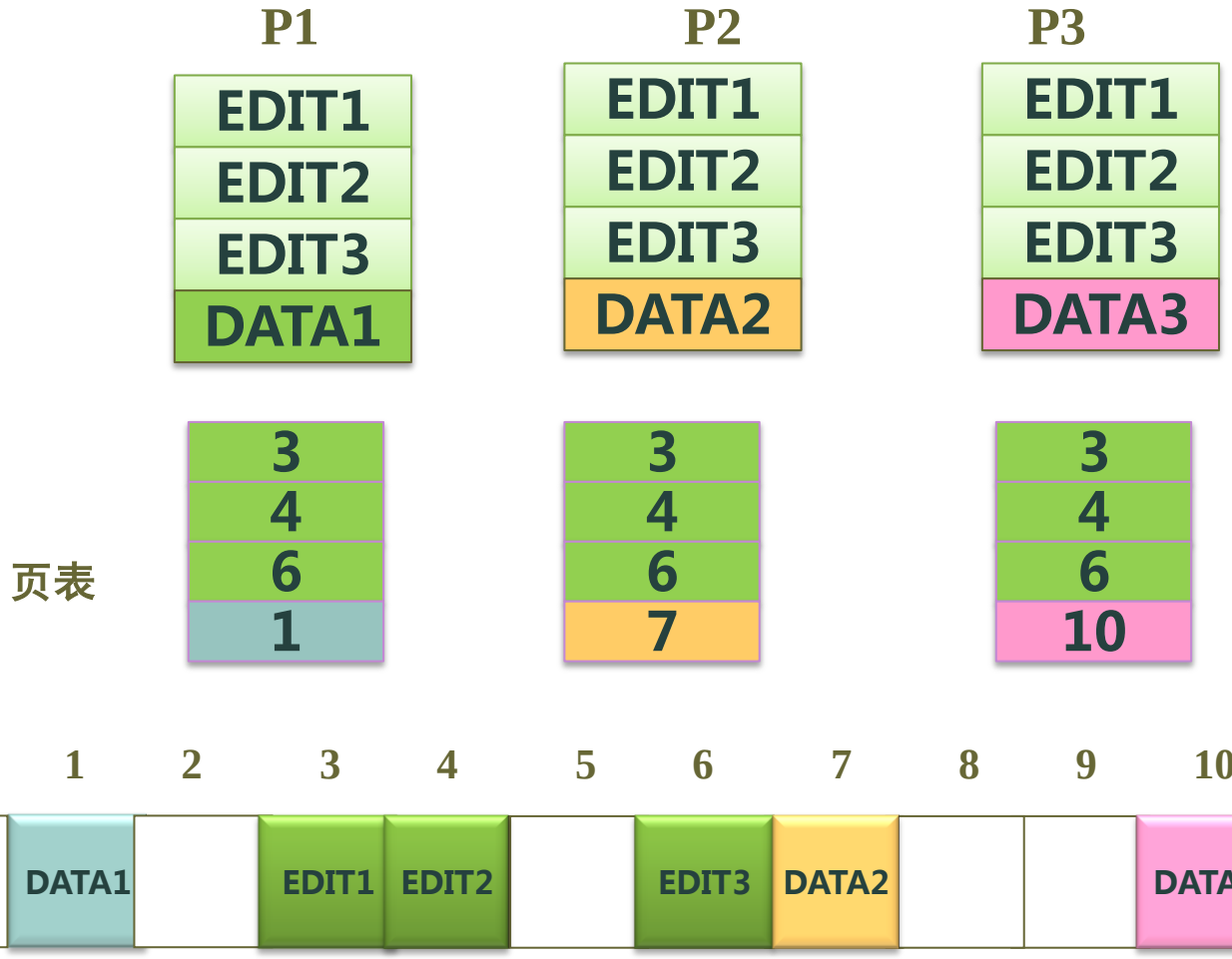
举例

- 设为每一页表项增加三位
  - R位表示读权限，W位表示写权限，E位表示执行权限

R	W	E	
0	0	0	不可进行任何操作
0	0	1	可以执行,不可以读写
0	1	0	只可以写
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

共享

- 可共享代码：动态链接库，编辑器，Shell
- 通过页表使几个逻辑空间指向同一个物理空间，实现程序共享



页式管理——优缺点

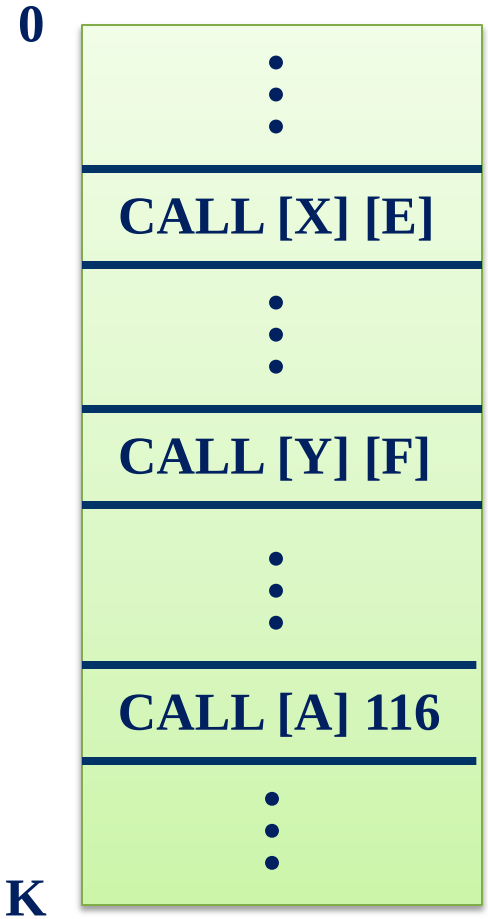
- 优点：
  - 没有外碎片，每个内碎片不超过页大小
  - 一个程序不必连续存放
  - 便于改变程序占用空间的大小，即随着程序运行而动态生成的数据增多，地址空间可相应增长
  - 便于管理
- 缺点：
  - 地址转换开销较大
  - 程序全部装入内存
  - 不易实现程序共享
  - 不便于动态连接

存储管理

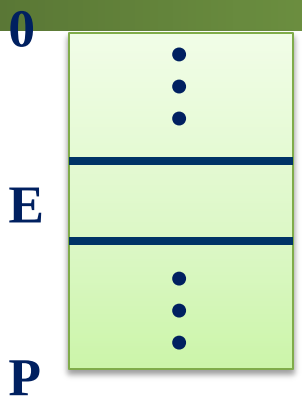
- 连续空间分配
  - 单道连续分配
  - 多道固定划分法
  - 多道连续可变划分法
- 不连续空间分配
  - 页式管理
  - 段式管理
  - 段页式管理
- 虚存管理
  - 页式虚存的基本思想
  - 页式虚存管理的实现
  - 页面替换策略

段式管理

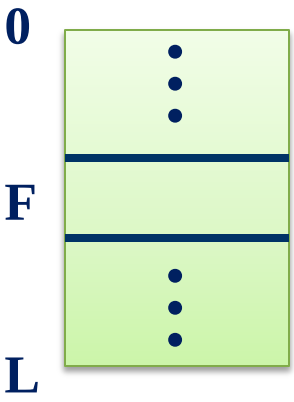
- 程序设计的需求
  - 程序在逻辑上由连续变成分离
  - 通过分段(segmentation)划分为多个模块，如代码段、数据段、共享段
    - 分别编写和编译
    - 针对不同类型的段采取不同的保护
    - 按段为单位来进行共享，包括通过动态链接进行代码共享



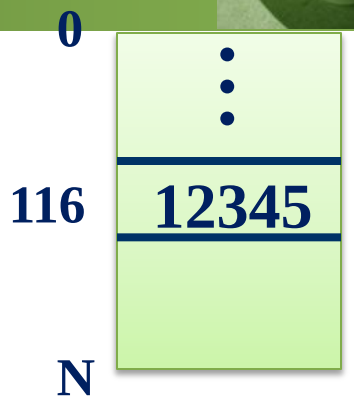
主程序段[M]



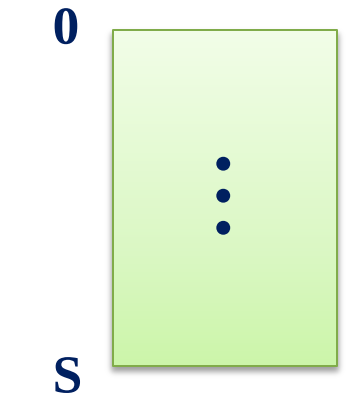
子程序段[X]



子程序段[Y]



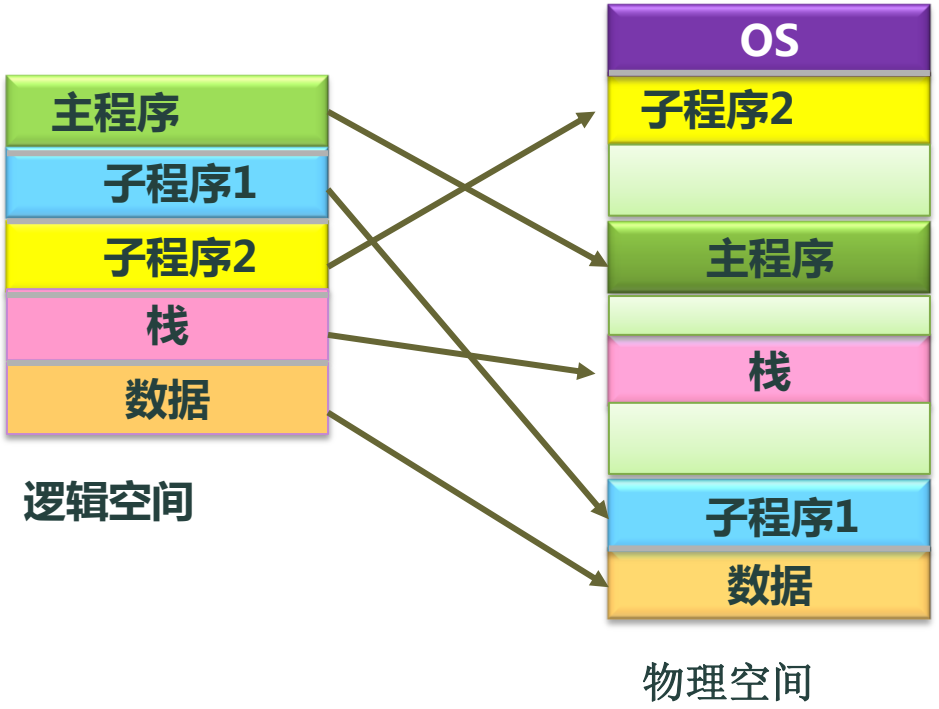
数组[A]



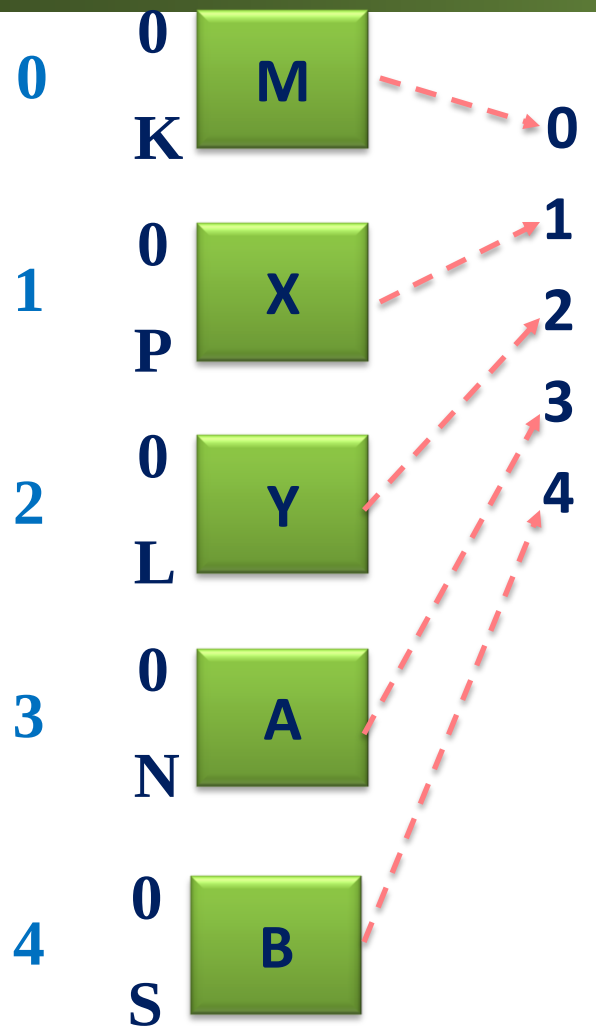
工作区段[B]

段式管理

- 空间安排
  - 用户作业逻辑空间为二维空间，由若干自然段组成
  - 逻辑地址：段号.段内偏移，记作s, d
  - 编译及装配时把所有地址记成(s, d)的形式
  - 物理内存空间管理：与多道可变划分法一样，系统以段为单位分配物理内存

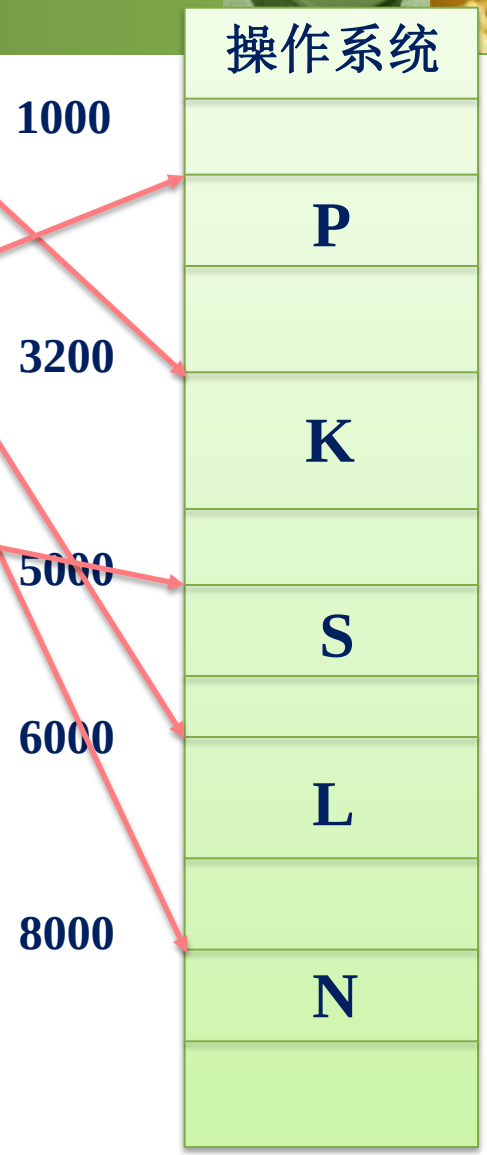


逻辑段号



作业1的逻辑地址空间

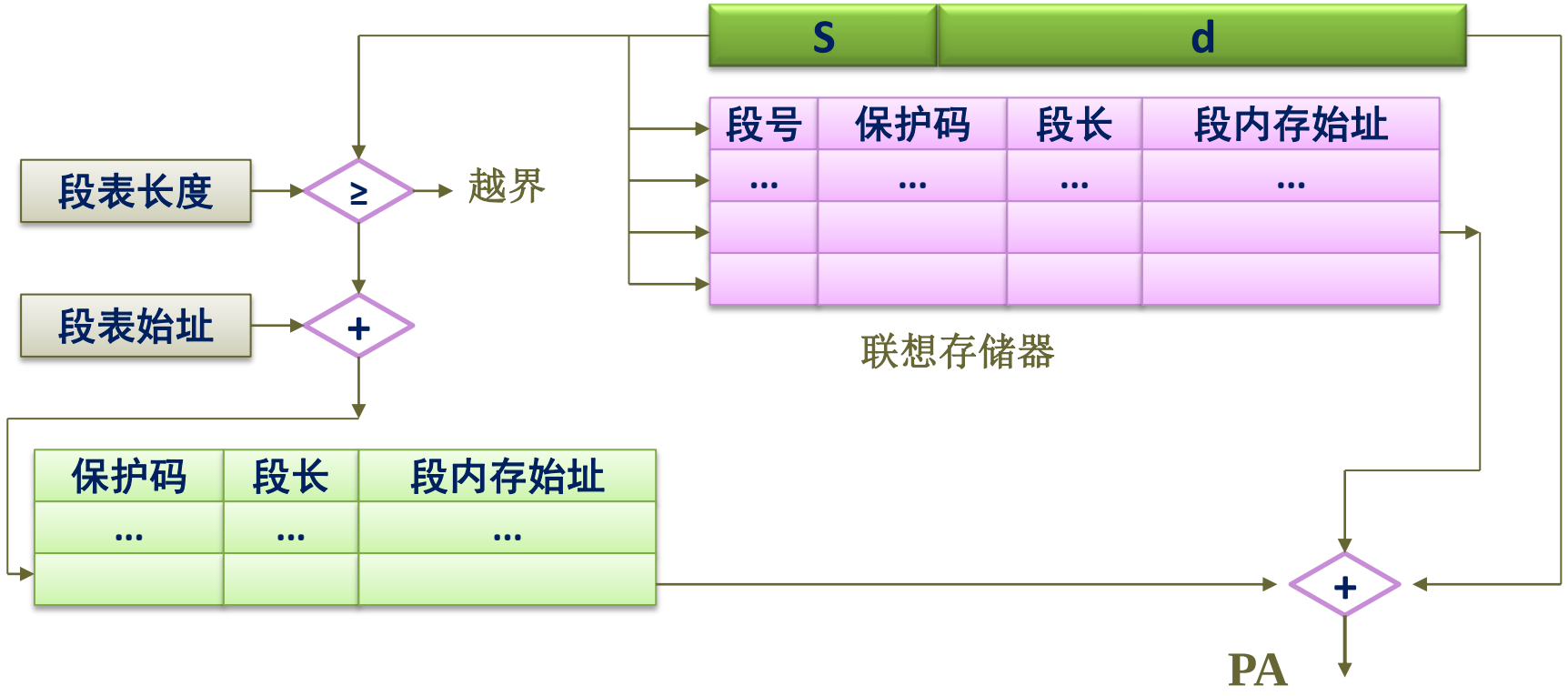
	长度	段地址
0	K	3200
1	P	1500
2	L	6000
3	N	8000
4	S	5000



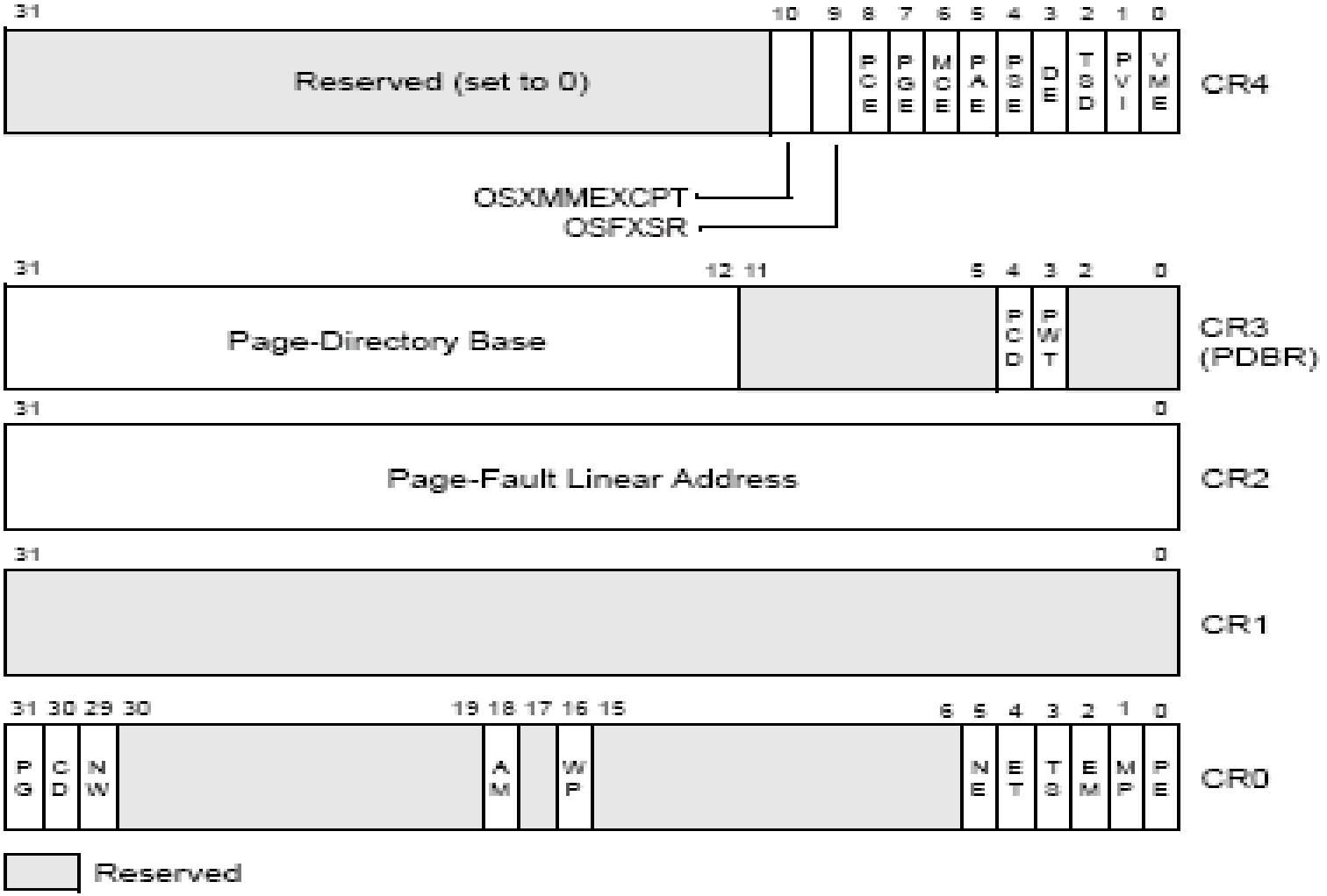
物理内存

动态地址转换

- 段表
  - 放于系统空间，由段表项组成，作业每段由一个段表项表示
  - 进程PCB表中存有段表始地址、段表长度
  - 段表始地址寄存器、段表长度寄存器

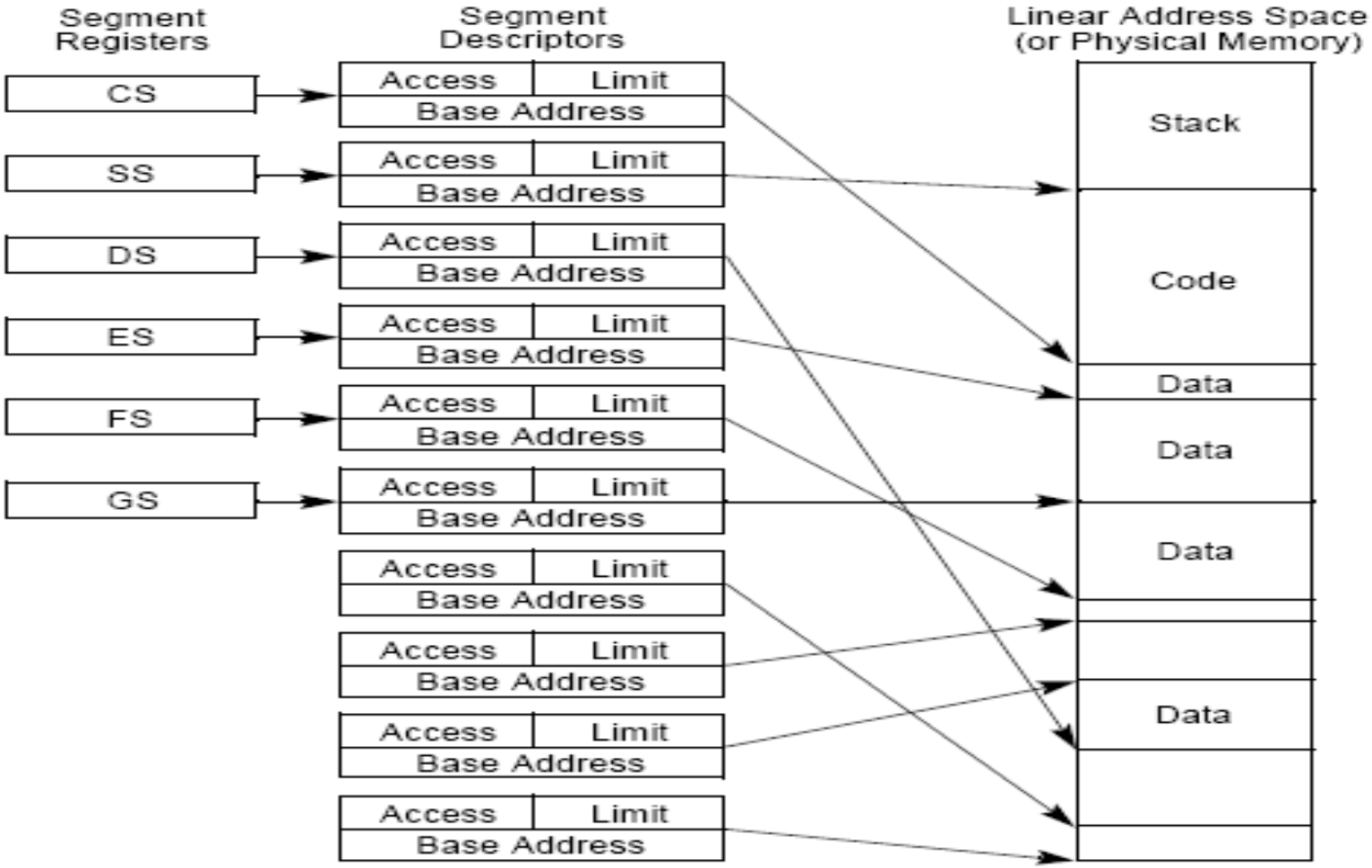


Intel体系结构的寄存器

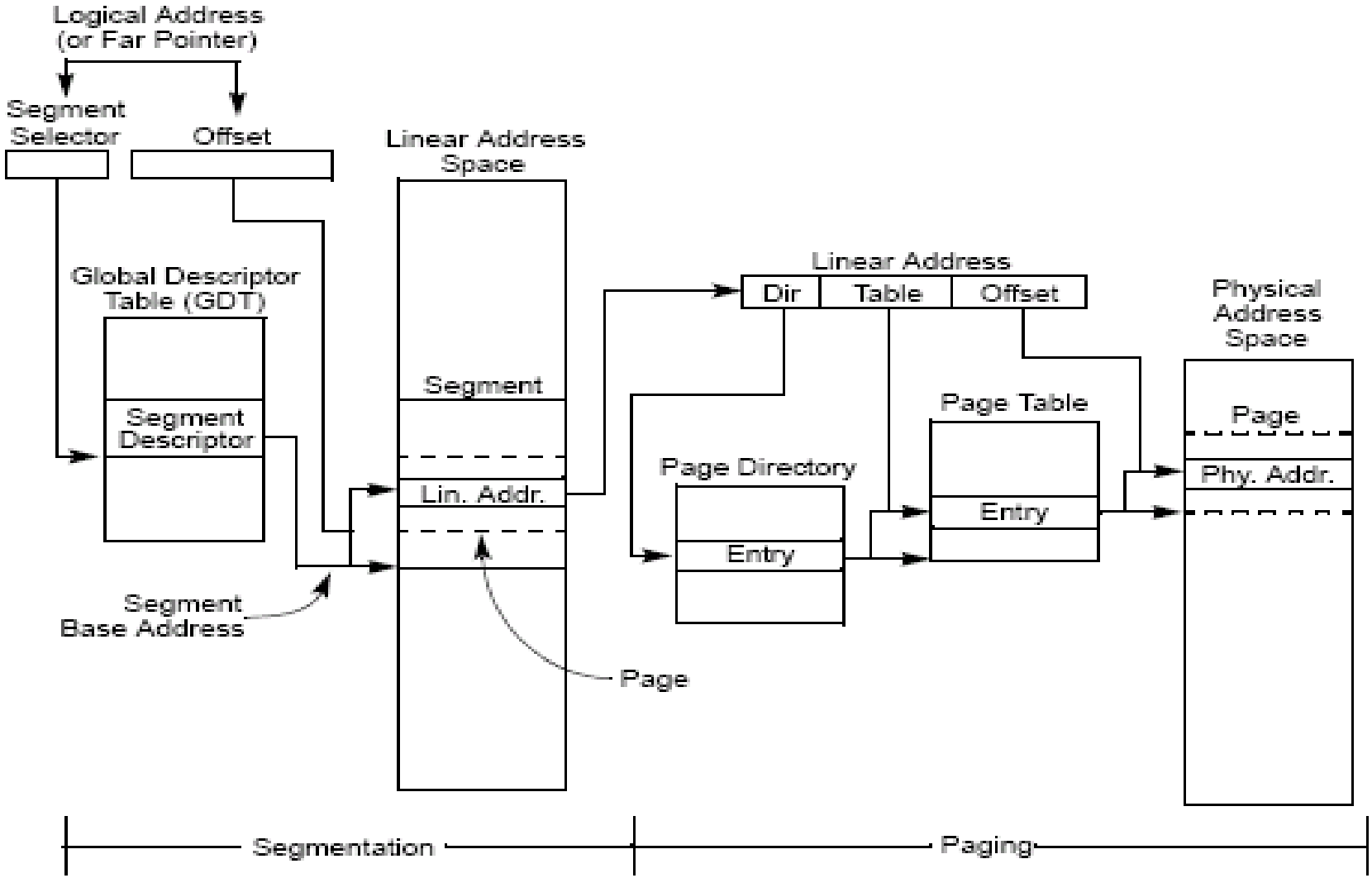


Intel体系结构分段与保护

- 多段模式：完全的硬件保护



Intel系统结构上的分段和二级页表



共享

- 与页式管理方法基本相同
- 多个作业段表中的某一项指向内存中的同一地址
- 若共享的段引用自身的某个地址，则各进程必须用同一段号来共享这一段

段式管理

- 优点
  - 管理和使用统一化
  - 便于程序共享
  - 便于动态链接
- 缺点
  - 产生碎片，段内连续

页式和段式的比较

- 分页是出于系统管理的需要，分段是出于用户应用的需要
  - 一条指令或一个操作数可能会跨越两个页的分界处，而不会跨越两个段的分界处
- 页大小是系统固定的，而段大小则通常不固定
- 逻辑地址表示：
  - 分页是一维的，各个模块在链接时必须组织成同一个地址空间
  - 分段是二维的，各个模块在链接时可以每个段组织成一个地址空间
- 通常段比页大，因而段表比页表短，可以缩短查找时间，提高访问速度

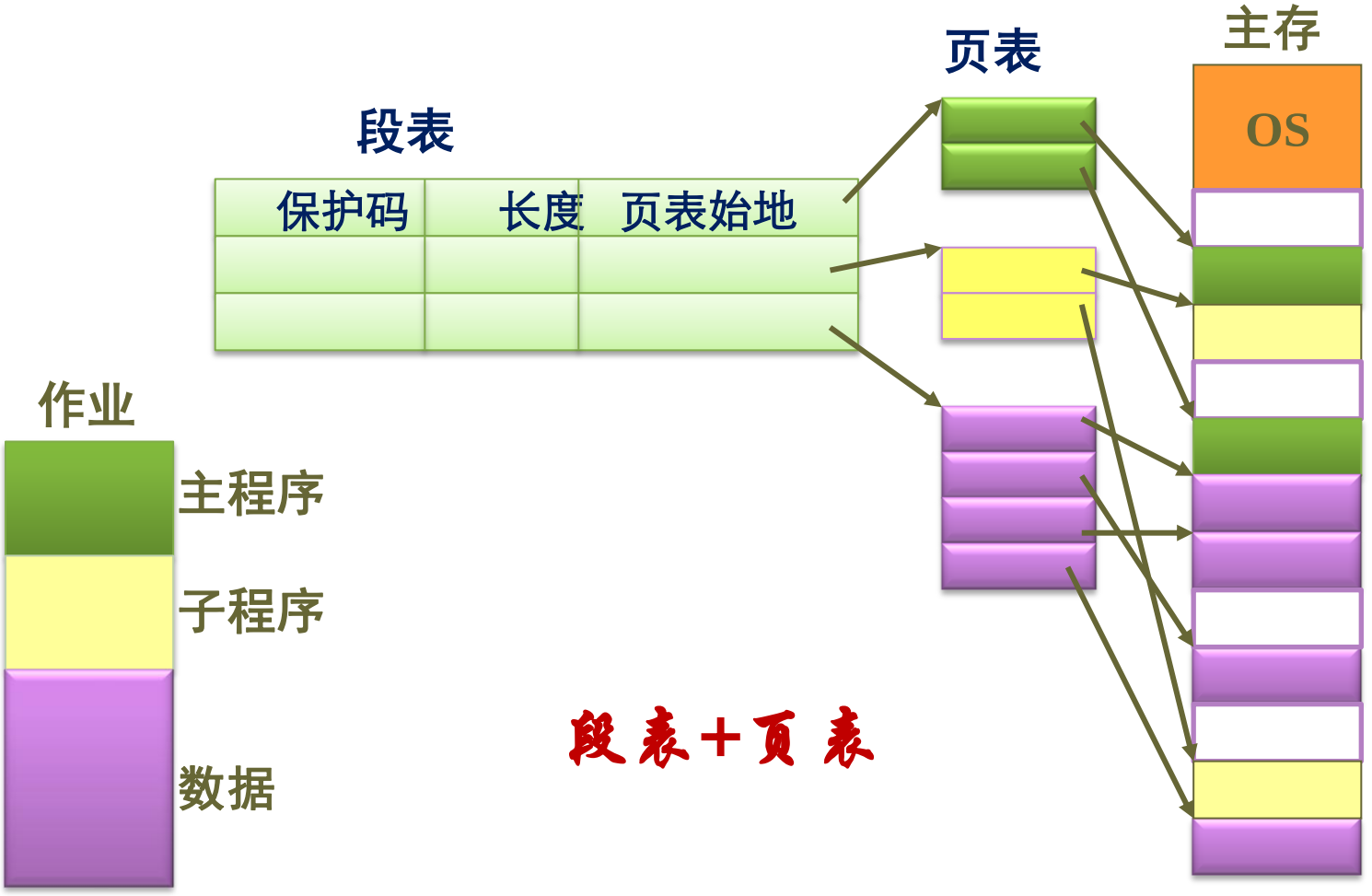
存储管理

- 连续空间分配
  - 单道连续分配
  - 多道固定划分法
  - 多道连续可变划分法
- 不连续空间分配
  - 页式管理
  - 段式管理
  - 段页式管理
- 虚存管理
  - 页式虚存的基本思想
  - 页式虚存管理的实现
  - 页面替换策略

段页式管理

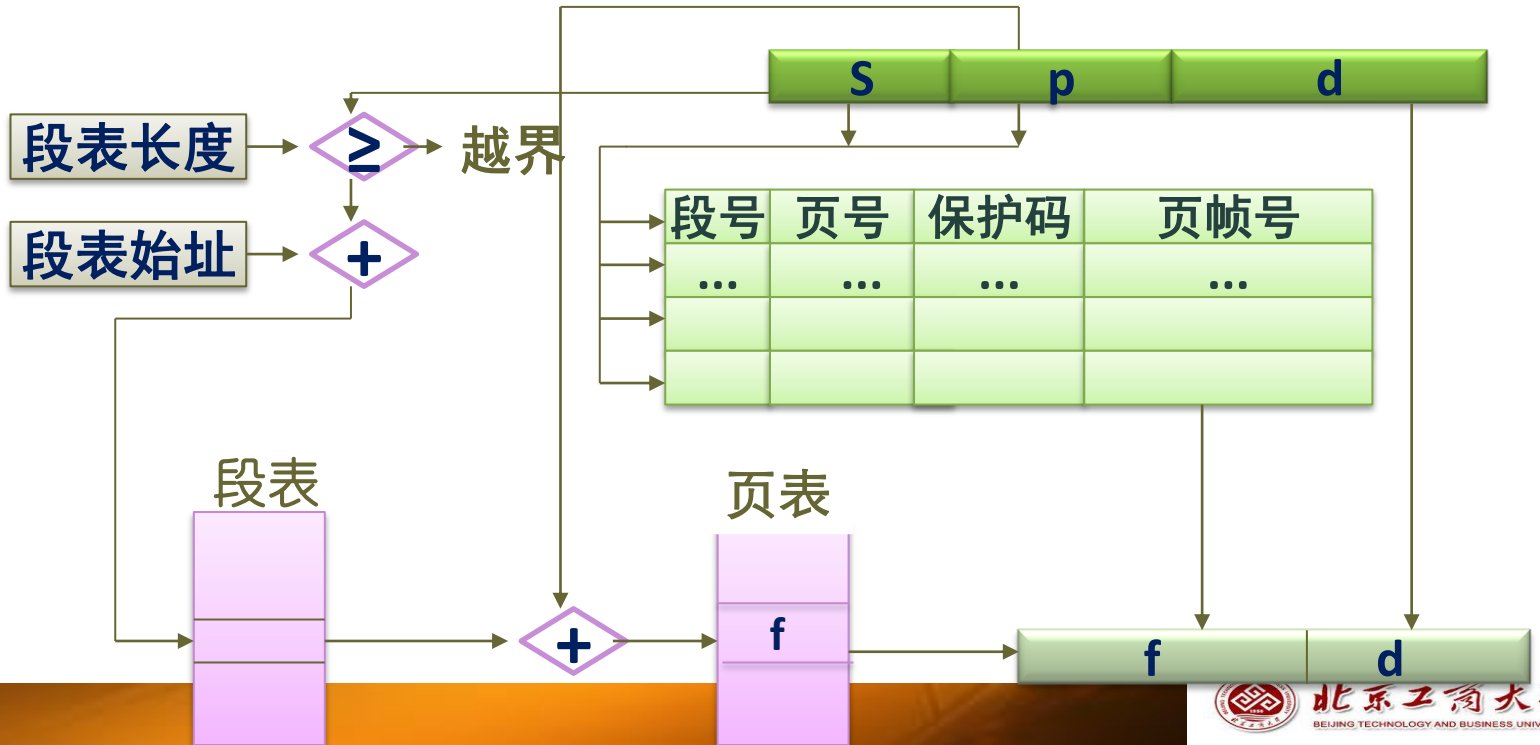
- 将作业分成若干段，每段用页式管理实现内存分配
- 空间安排
  - 对于用户而言
    - 段页式管理与段式相同，用户逻辑地址只涉及段号与段内位移
  - 对于物理内存管理而言
    - 与页式系统相同
  - 系统内的逻辑地址
    - 段号 ● 段内位移 --> 段号 ● 页号 ● 页内位移
    - 记作：S，P，d.

作业的段表和页表



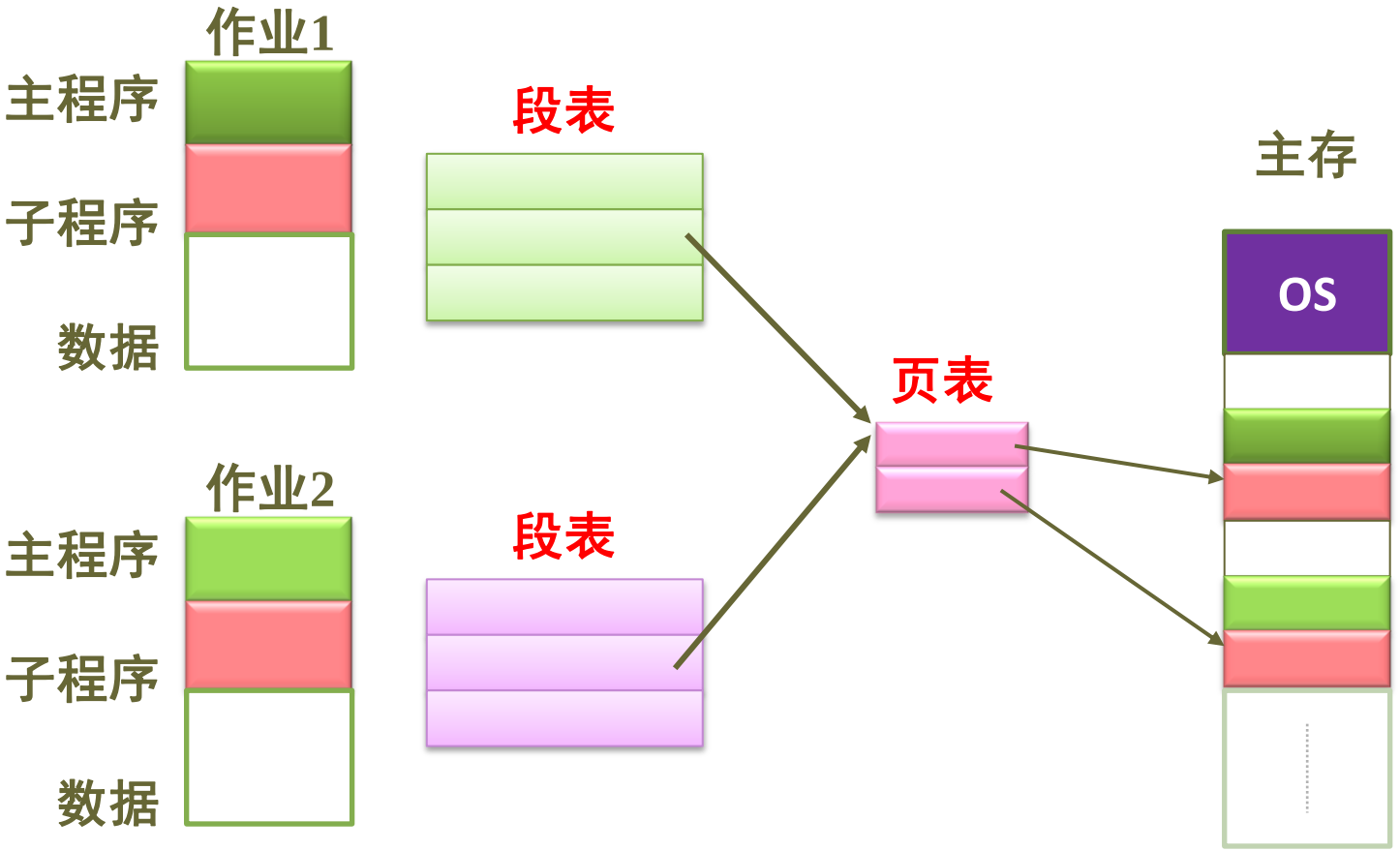
动态地址转换

- 平均访问时间
  - 设访存时间为750ns，搜索联想存储器的时间为50ns，命中率为95%，则
    - 95% *(750+50) + 5% * (50+750+750+750)) = 875ns
  - 比连续存储方式降低性能 ：$\frac{875-750}{750}=16.7\%$



保护与共享

- 保护与段式管理相同
- 共享则以页为单位，或者共享页表



存储管理

- 连续空间分配
  - 单道连续分配
  - 多道固定划分法
  - 多道连续可变划分法
- 不连续空间分配
  - 页式管理
  - 段式管理
  - 段页式管理
- 虚存管理
  - 页式虚存的基本思想
  - 页式虚存管理的实现
  - 页面替换策略

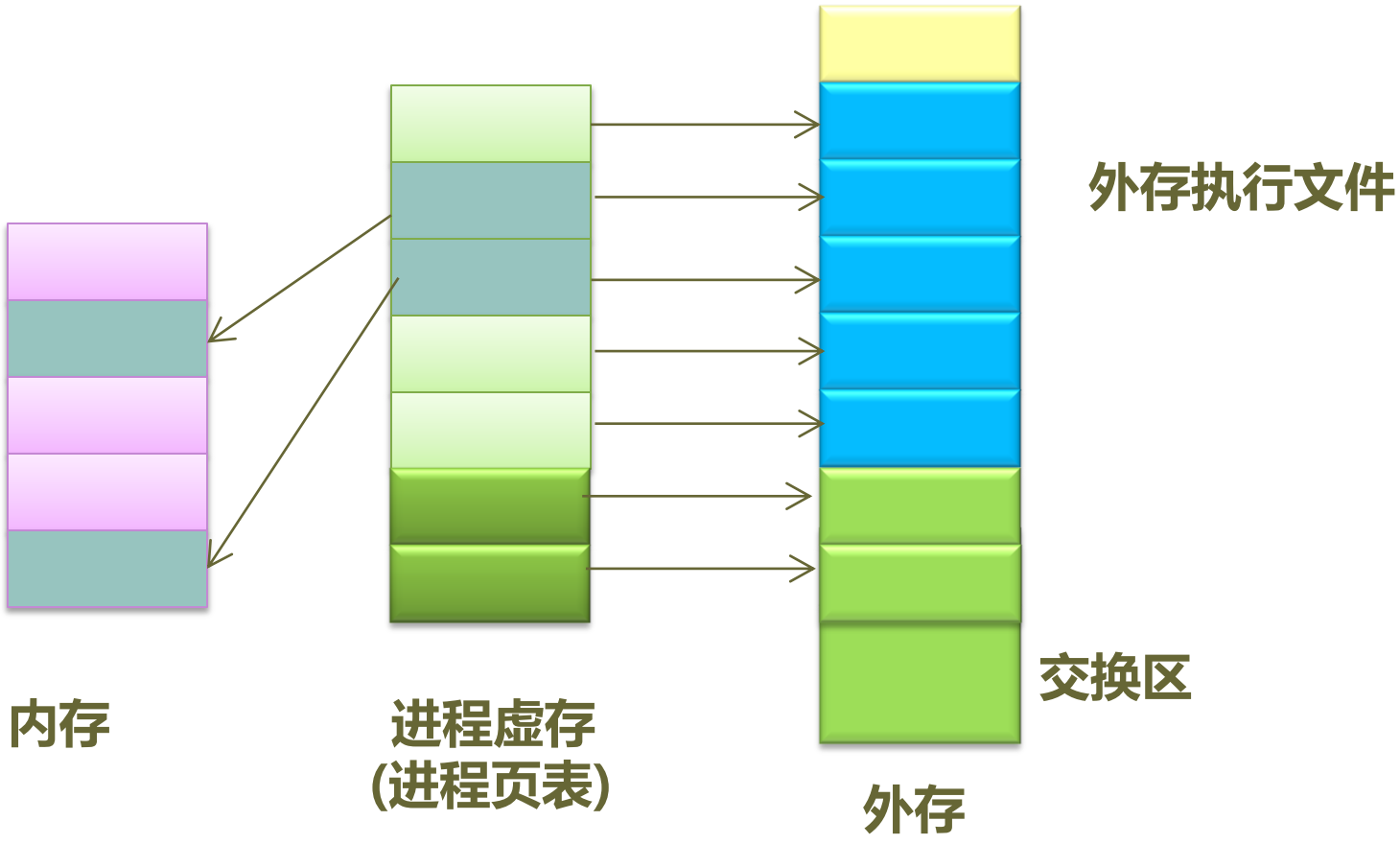
虚存管理

- 问题的提出：
  - 程序大于内存
  - 程序暂时不执行是否还要占用内存?
- 基本思想
  - 程序、数据、堆栈的大小可以超过内存的大小
  - 操作系统把程序当前使用的部分保留在内存，而把其它部分保存在磁盘上
  - 在需要时在内存和磁盘之间动态交换
  - 支持多道程序设计技术
- 分类
  - 虚拟页式
  - 虚拟段式
  - 虚拟段页式

页式虚存的基本思想

- 基本工作原理
  - 在进程开始运行之前，装入一个或零个页面
  - 之后根据进程运行的需要，动态装入其它页面
  - 当内存空间已满，而又需要装入新的页面时，则根据某种算法淘汰某个页面，以便装入新的页面
- 在简单页式存储管理的基础上，增加请求调页和页面置换功能

进程页式虚空间与内、外存关系



存储管理

- 连续空间分配
  - 单道连续分配
  - 多道固定划分法
  - 多道连续可变划分法
- 不连续空间分配
  - 页式管理
  - 段式管理
  - 段页式管理
- 虚存管理
  - 页式虚存的基本思想
  - 页式虚存管理的实现
  - 页面替换策略

页式虚存管理

页表项结构

合法位	修改位	页类型	保护码	外存块号	页帧号

- 合法位：1表示该页在内存
- 修改位：1表示该页被修改过，在释放或淘汰时应写回外存
- 页类型：
  - 零页：表示该页在分配物理页帧时应清0页帧空间
  - 回写swap区页:表示回写swap区
- 保护码：R、W、E保护说明
- 外存块号：该页所在外存的块号
- 页 帧 号：当合法位为1时，代表该页所在内存的帧号

页表建立

- 在进程创建时建立页表，页表项在初始时，合法位、修改位及页帧号都置0
- 初始化页表方法
  - 部分复制父进程页表（如UNIX的fork()）
    - 分配pid给子进程，分配PCB空间；
    - 初始化PCB（进程标识，调度信息）；
    - 分配子进程页表空间；
    - 拷贝父进程的程序区页表项，使程序共享；
    - 申请空闲页帧，复制父进程的数据区和栈区，复制数据区和栈区页表项内容后，修改页帧号；
    - 继承父进程对其他资源的访问现场；
    - 用父进程PCB中现场区初始化子进程的现场区，且保证子进程恢复现场运行后返回值为零；
    - 将子进程挂到就绪队列；
    - 把子进程pid作为返回值

页表建立

- 用一个可执行的文件来初始化页表
  - 为执行程序页面建页表项，保护码为可执行，外存块号即该页所在的文件的外存块号。（不必回写）
  - 为所有初始数据页建页表项，保护码为可读写，页类型说明成回写swap页，外存块号即该页所在文件的物理块号
    - 待该页回写时，再分配swap区空间，改外存块号栏，修改页类型为正常页
  - 为所有临时数据页建页表项，保护码为可读写，页类型说明成零页，外存块号栏空
    - 当第一次访问该页时，分配页帧并清0页帧
    - 回写时，再分配swap区空间，填外存块号栏，页类型变为正常页

硬件动态地址转换

- 在执行虚存访问指令时，由硬件合成物理地址
  - 若能在联想存储器中获得该虚页的物理页帧号，则访问之
  - 若要查当前进程页表，须先检查该页页表项的合法位
    - 若置1，则从页表项中获得页帧号
    - 否则要发一个页故障(page fault)或叫缺页中断（例外）
    - 当缺页中断处理完后，重新执行访存指令.
  - 联想存储器中的页表项都是合法页的页表项

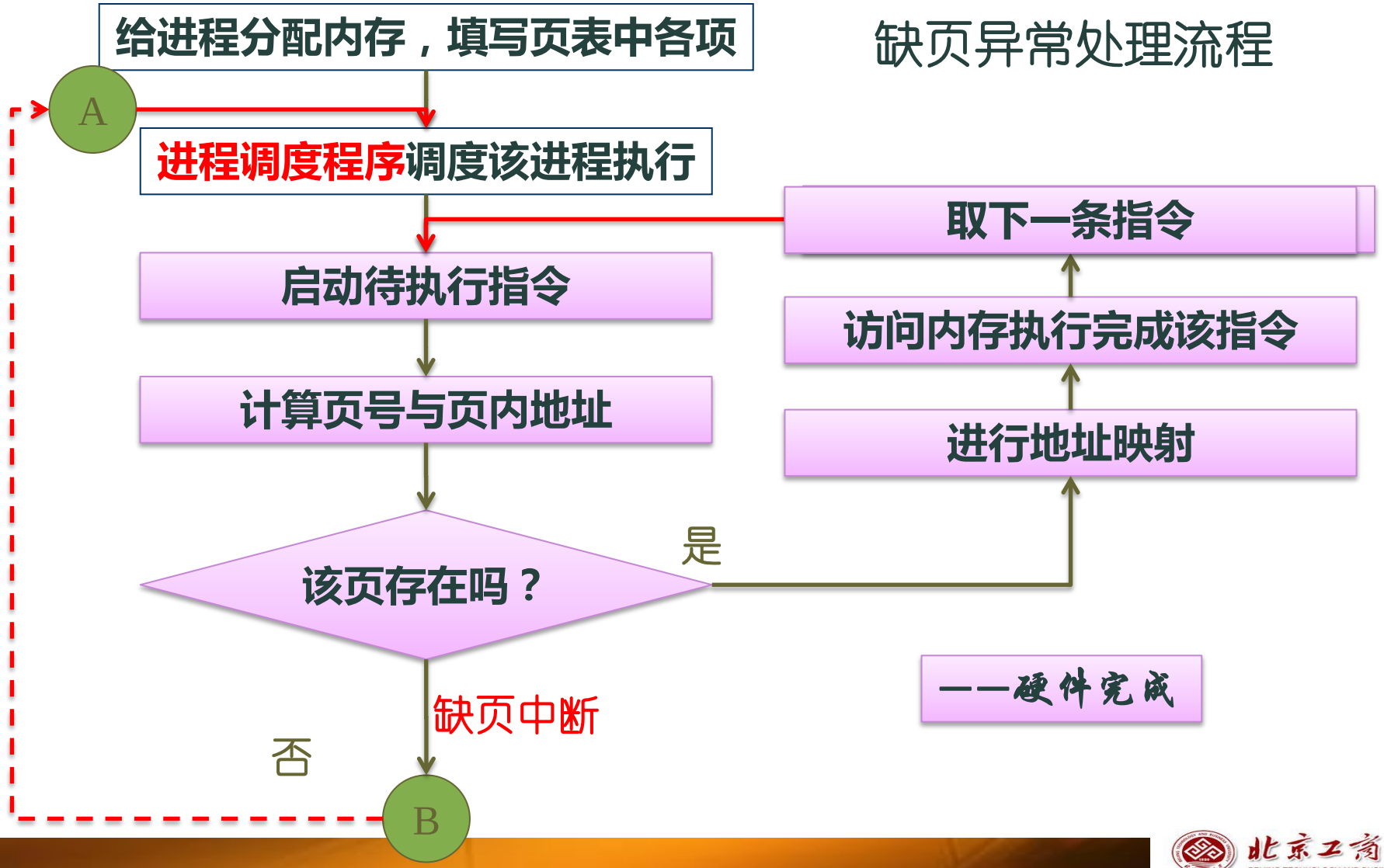
缺页处理

- 在地址映射过程中，在页表中发现所要访问的页不在内存，则产生缺页异常
- 操作系统接到此异常信号后，就调缺页中断处理程序

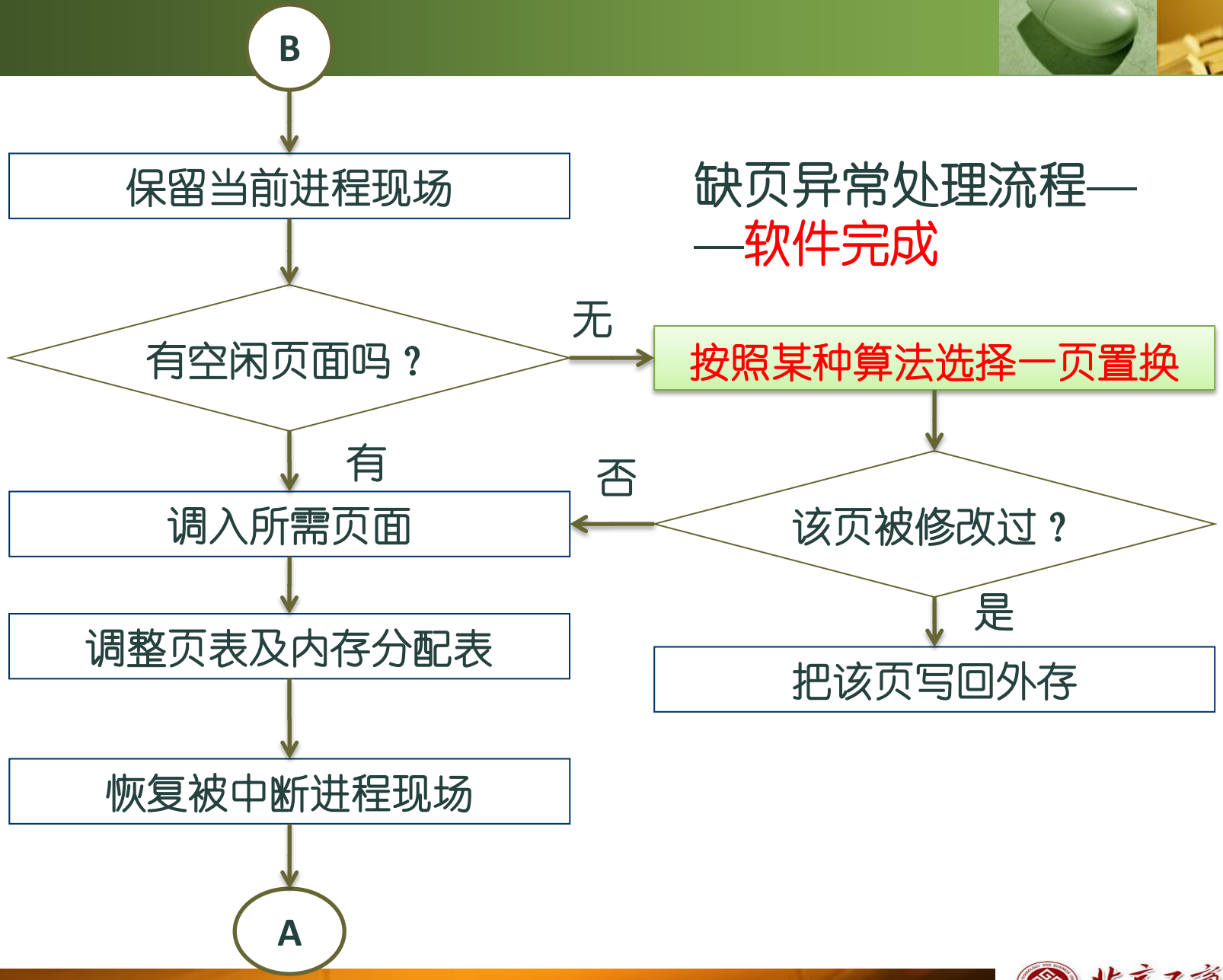
缺页处理

- 根据发生页故障的虚地址得到页表项
- 申请一个可用的页帧
  - 根据所采用的替换策略可能需要引起淘汰某一页
- 检查页类型
  - 若为零页，则将页帧清0，将页帧号填入页表项的页帧号一栏，置合法位为1
  - 若非零页，则调用I/O子系统将外存块号所指的数据读到可用页帧，将页帧号填入页表项中，合法位置1
- 结束.

缺页异常处理流程


给进程分配内存，填写页表中各项
A
进程调度程序调度该进程执行
启动待执行指令
计算页号与页内地址
该页存在吗？
是
否
缺页中断
B
取下一条指令
访问内存执行完成该指令
进行地址映射
——硬件完成

缺页异常处理流程——软件完成



页淘汰(页置换)

- 页淘汰可以发生在申请页帧时，而现代OS一般都定时进行页淘汰
- 如何选取被淘汰的页是由页面替换策略决定的
- 若已决定淘汰页P，则淘汰一页的主要工作有
  - 查P页表项的修改位
    - 若未修改，则清0合法位，将页帧送回空闲页帧队列
    - 若已修改
      - 则检查类型栏
        - » 若是零页或回写swap区页，则申请 一块swap区空间，记下P的外存块号
      - 调用I/O子系统将页帧上的数据写到外存块号所指的外存空间
      - 清0合法位
  - 将页帧送回空闲页帧队列

存储管理

- 连续空间分配
- 不连续空间分配
- 虚存管理
  - 页式虚存的基本思想
  - 页式虚存管理的实现
  - 页面替换策略

页面替换策略

- 功能
  - 需要调入页面时，选择内存中哪个物理页面被置换（replacement policy）
- 目标
  - 把未来不再使用的或短期内较少使用的页面调出
  - 通常在局部性原理指导下依据过去的统计数据进行预测

页面替换策略中基本概念

- 驻留集(工作集)
  - 进程的合法页的集合
- 访问串
  - 进程访问虚空间的地址踪迹
- 举例
  - 某进程依次依次访问如下地址，0100，0432，0101，0612，0102，0103，…
  - 页式虚存管理以页为基本单位，只需页号即可
  - 设页面大小为100，上述访问串可简化为1，4，1，6，1，1，…

页面替换策略

- 分类
  - 驻留集大小固定的替换策略
  - 驻留集大小可变的替换策略

驻留集大小固定的替换策略

- 设驻留集大小为m，s(t)为t时刻的驻留集，r(t)为t时刻访问的页号。t取0, 1, …, t，指访存指令执行时刻
- 驻留集与换入/出的关系
  - 进程刚创建时，驻留集为空，即s(t)=NULL
  - 若t+1时刻访问的页在s(t)中时，访问之
    - 即若r(t+1)∈s(t)，则s(t+1)= s(t)
  - 若t+1时刻访问的页不在s(t)中时，且驻留 集 小于m，则换入
    - 即若 r(t+1)!∈s(t)，且|s(t)|<m，则 s(t+1)=s(t)+{r(t+1)}
  - 若t+1时刻访问的页不在s(t)中时，且驻留集等于m，则先换出页y，再 换入 r(t+1)页
    - 即s(t+1)=s(t)-{y}+{r(t+1)}

先进先出算法(FIFO)

- 选择最早进入内存的页面被置换
  - 可以通过链表来表示各页的建立时间先后
- 性能较差
  - 较早调入的页往往是经常被访问的页，这些页在FIFO算法下被反复调入和调出
- 有Belady奇异
  - 指替换策略不满足“随着驻留集的增大，页故障数一定减少”的规律

Belady奇异的例子

进程P有5页程序访问页的顺序为：1，2，3，4，1，2，5，1，2，3，4，5；

如果在内存中分配3个页面，则缺页情况如下：12次访问中有缺页9次；

FIFO	1	2	3	4	1	2	5	1	2	3	4	5
页 0	1	2	3	4	1	2	5	5	5	3	4	4
页 1		1	2	3	4	1	2	2	2	5	3	3
页 2			1	2	3	4	1	1	1	2	5	5
缺页	x	x	x	x	x	x	x	√	√	x	x	√

如果在内存中分配4个页面，则缺页情况如下：12次访问中有缺页10次；

FIFO	1	2	3	4	1	2	5	1	2	3	4	5
页 0	1	2	3	4	4	4	5	1	2	3	4	5
页 1		1	2	3	3	3	4	5	1	2	3	4
页 2			1	2	2	2	3	4	5	1	2	3
页 3				1	1	1	2	3	4	5	1	2
缺页	x	x	x	x	√	√	x	x	x	x	x	x

Belady奇异：分配的页面多了，反而出现更多的缺页

——〉很少单独使用FIFO算法

理想页面置换算法(OPT)

- 选择“未来不再使用的”或“在离当前最远位置上出现的”页面被置换
- 这是一种理想情况，是实际执行中无法预知的，因而不能实现
- 可用作性能评价的依据

举例：驻留集大小为3，访问串为

7，0，1，2，0，3，0，4，2，3，0，3，2..

7	7	7	2	2	2	2	2	2	2	2	2	2
	0	0	0	0	0	0	4	4	4	0	0	0
		1	1	1	3	3	3	3	3	3	3	3
o	o	o	o		o		o			o		

LRU(Least Recently Used)

- 选择内存中最久未使用的页面被置换
- 局部性原理的合理近似，性能接近最佳算法

举例：驻留集大小为3，访问串为

7，0，1，2，0，3，0，4，2，3，0，3，2..

7	7	7	2	2	2	2	4	4	4	0	0	0
	0	0	0	0	0	0	0	0	3	3	3	3
		1	1	1	3	3	3	2	2	2	2	2
o	o	o	o		o		o	o	o	o		

LRU(Least Recently Used)

- LRU策略是一种栈算法
  - 满足S（m，t）属于 S（m+1，t）的替换算法被称为栈算法
- 分析
  - LRU策略中，当驻留集大小为m时，S（m，t）中保持着最近使用过的m个页帧
  - 当驻留集大小为m+1时，S（m+1，t）中保持着最近使用过的m+1个页帧
  - 故S（m，t）属于 S（m+1，t），LRU策略是栈算法

LRU(Least Recently Used)

- 栈算法没有Belady奇异
  - 设n>m，对于栈算法有S(m，t)属于 S(n，t)，
  - 任取r（t)，若r（t）!∈ S(n，t)，则r（t）!∈ S(m，t)
  - 因此，驻留集为n 时出现的页故障一定会出现在驻留集为m 时， 即 P(m)>=P(n) （ P(i) 为驻留集为i的页故障数)
  - 所以,LRU没有Belady奇异
- LRU策略的特点
  - 要硬件配合，实现费用高，但效果适中

LRU(Least Recently Used)

- 实现方法1
  - 给每个页帧设一个计数器
  - 每访问一页，对应页帧计数清0，其余页帧计数加1
  - 淘汰计数最大的页帧
- 实现方法2
  - 类似于栈的结构，栈长等于驻留集m
  - 每访问一页
    - 若在驻留集内，移出该页号，压入栈顶
    - 若不在驻留集内，从辅存载入该页，页号压入栈顶
    - 淘汰栈底页号

实用方法（兼顾FIFO和LRU策略）

- 为页帧在页表项中增加一位使用位
- 硬件每访存一次即将对应页的使用位置1
- 操作系统页面管理程序定时将所有使用位清0
- 淘汰时任选一个使用位为0的页
- 操作系统选择淘汰页时
  - 尽量避免选被修改过的页
  - 首先选择使用和修改位都为0的页
  - 若没有，再选修改位为1，使用位为0
  - 再选使用位为1，修改位为0的页
  - 最后按FIFO选两者均为1的页

驻留集大小可变的替换策略

- 程序行态
  - 指程序访存布局特性和行为特性
  - 访问串具有的性质
- 局部性行态（Locality）
  - 一段时间内程序访存有局部性
  - 将程序执行时间分段，在每一段时间内程序只引用页集合中的一个小子集
- 阶段转换行态
  - 从一个局部集向另一个局部集过渡是突然的
  - 局部集大小一般不超过程序总页数的20%
  - 因此：驻留集应比最大的局部集稍大一些
- 引入可变大小的驻留集
  - 驻留集小于局部集时引起抖动，驻留集大于局部集又是浪费
  - 因此，随着程序访问虚存的局部集变化而改变驻留集

WS(working set)

- 若驻留集中的某页有△个访问间隔没被访问则将其淘汰
- 举例：取△=5，访问串为

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1
	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
			1	1	1	1	1	4	4	4	4	4	2	2	2	2	1
				2	2	2	2	2	2	2	2	2		1	1	1	
驻留集	1	2	3	4	4	4	4	4	4	4	4	4	3	4	4	4	3

发生7次页故障，驻留集平均大小为3.5个页帧

WS(working set)

- 特点
  - 每一页面设一计数器
  - 每访存一次，将进程所有页面计数器加1，所访存的页面计数器清0
  - 淘汰计数器值等于△的页面
- 特点：
  - 开销太大，没有实用

SWS(Sampled Working Set)

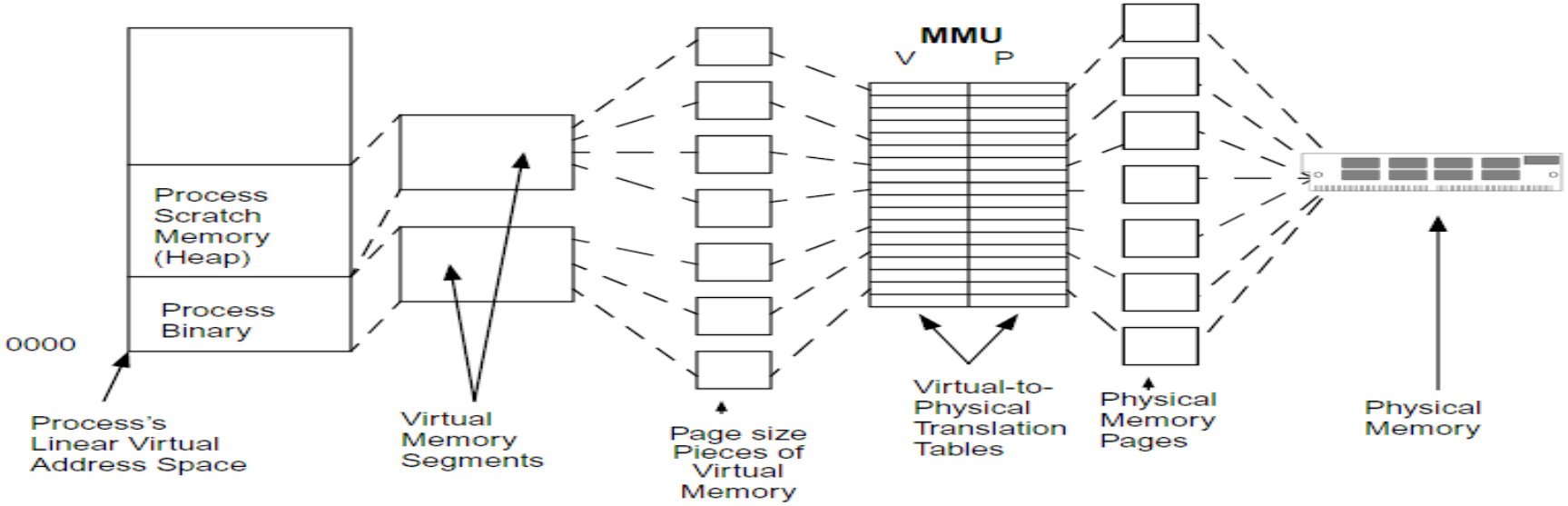
- 思路
  - 定时检查计数器，淘汰计数器值大于等于△的页面
  - 硬件消耗仍很大
  - 每访问一页，将当前硬时钟值记录在页表项中，操作系统定时（ 10us）检查驻留集页表项的时钟值
  - 若：（当前时钟值 - 页表项中时钟值 ）>△，则淘汰之

替换策略选择

- 实用操作系统选择动态驻留集FIFO(SWS)的变种
- 系统设立两个队列：自由链表和修改链表
- 定时作页预淘汰
  - 淘汰时不立即删去页中数据，根据页面修改否挂入自由链/修改链，修改链过长时，回写页面后改挂到自由链中
  - 在自由链/修改链中的页面再次被访问时，则将该页从链中摘除
    - 该页又能通过页表项访问到
    - 从预淘汰回到被使用状态
  - 换入 要用空页时，选自由链的第一页帧，改变该页帧原页面页表项相关信息
    - 这时页中数据被覆盖（真正被淘汰）

小结

- 进程的逻辑空间到物理空间的映射



- 物理内存页帧按需换入和换出
- 进程之间的内存共享和保护

小结

- 连续可变存储管理
  - 几个基本概念：逻辑地址、物理地址、基地址、动态/静态地址变换
- 页式存储管理
  - 基本原理和地址变换过程
- 段/段页式管理
  - 共享和保护
- 虚存管理系统内存访问过程
  - 页故障，软硬件分工
- 各种页面替换策略及实用的综合策略
  - 替换策略分析及对比
- 固定驻留集算法和SWS等实用动态驻留集算法