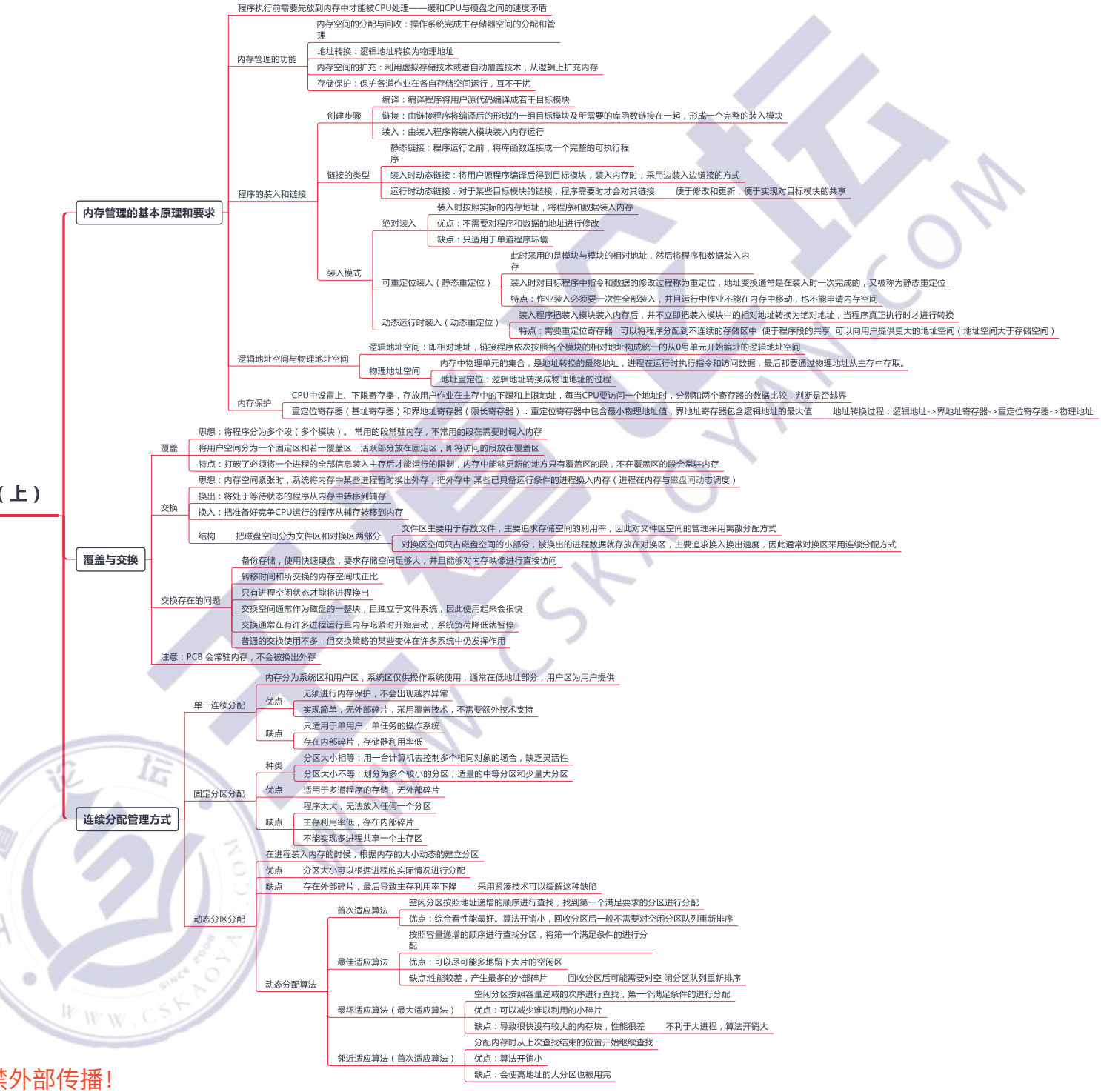
3.1 内存管理概念（上）



3.1 内存管理概念（中）

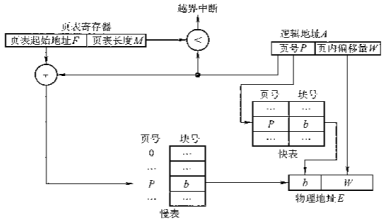
非连续分配管理方式

允许一个程序分散的装入不相邻的内存分区

基本分页存储管理方式

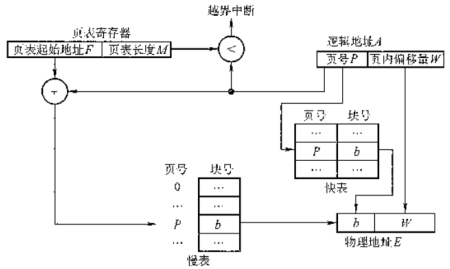
设计思想
- 将主存空间划分为大小相等且固定的块，块相对较小，作为主存的基本单位，进程以块为单位尽心空间申请
- 分页存储与固定分区技术很像，但是其分页相对于分区又很小，分页管理不会产生外部碎片，产生的内部碎片也非常的小

分页存储的基本概念
- 页面和页面大小
  - 进程中的块 = 页
  - 内存中的块 = 页框（页帧）
  - 进程申请主存空间，为每个页面分配主存中可用页框，即页与页框一 一对应
  - ***页面大小要适中***
    - 页面太小：进程页面数过多，页表过程，增加内存占用，降低硬件地址转换效率
    - 页面太大：页内碎片过多，降低内存利用率
- 地址结构　页号（有多少页的编号）+页内偏移（页内存了多少东西）
- 页表
  - 为了便于在内存中找到进程的每个页面对应的物理块，系统为每个进程建立一张页表，记录页面在内存中对应的物理块号，页表一般放在内存中
  - 页表项：页号+物理内存中的块号（不要与地址结构搞混）　页表项的物理内存块号+地址结构中的页内偏移 = 物理地址



基本地址变换机构
- 计算方式
  - 页号P=A/L，页内偏移量W=A%L
  - 比较页号P和页表长度M，若P>=M产生越界中断
  - 页表中页号P对应的页表项地址=页表始址F+页号P*页表项长度　取出该页表项内容b
  - 计算E=b*L+W　　使用E去访问内存
- 页表项大小的设计应当尽量一页正好能装下所有的页表项
- 分页管理存在的问题
  - 地址变换过程必须足够快，否则访存速率会降低
  - 页表不能太大，否则会降低内存利用率
- 组成
  - 设置一个页表寄存器（PTR），存放页表在内存中的起始地址F 和页表长度M
  - 页表的始址和页表长度放在进程控制块（PCB）中

具有快表的地址变换机构
- 可优化方向：如果页表放在内存中，取地址访问一次内存，按照地址取出数据访问一次内存，共需要两次访问内存
- 优化：地址变换机构中增加一个具有并性查找能力的高速缓冲寄存器（快表），又称为相联存储器（TLB）　相联存储器既可以按照地址查找也可以按照内容查找
- 访问一个逻辑地址的访存次数
  - 基本地址变换机构　两次访存
  - 具有快表的地址变换机构
    - 快表命中，只需一次访存
    - 快表未命中，需要两次访存
- 变换过程
  - CPU给出逻辑地址后，先查询快表中是否命中
  - 若快表命中，直接从快表中该页对应的页框好，与页内偏移量拼接成物理地址
  - 若快表不命中，再按照正常方式从页表中查询相应页表项，并将该页表项存入快表中（按照一定策略）
  - ***查找图示***

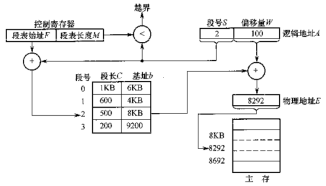


两级页表
- 如果页数过多，就会导致页表也过多，那么我们可以考虑设置一个用来储存页表的页表(套娃）
- 逻辑地址空间格式 = 一级页号 + 二级页号 + 页内偏移
- 设计多级页表的时候，最后一定要保证顶级页表一定只有一个
- 建立多级页表的目的在于建立索引，不必浪费主存空间去储存无用的页表项，也不用盲目式的查询页表项

3.1 内存管理概念（下）

基本分段存储管理方式

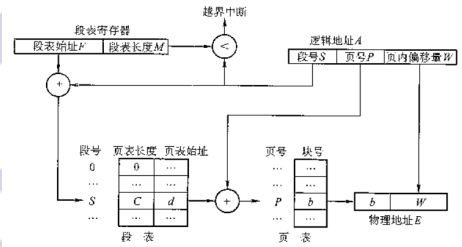
- 出发点
  - 分页是从计算机角度考虑设计的，目的是为了内存的利用率，提高计算机性能，分页通过硬件机制实现，对用户完全透明
  - 分段是从用户和程序员的角度提出，满足方便编程，信息保护和共享，动态增长及动态链接等多方面的需要
- 分段
  - 按照用户进程中的自然段划分逻辑空间
  - 地址结构 = 段号S+段内偏移量W
  - 页式系统中，页号和页内偏移对用户透明　　段式系统中 段号和段内偏移量必须由用户显示的提供
- 段表
  - 每个进程都有一张逻辑空间与内存空间映射的段白，这个段表项对应进程的一段，段表项记录该段在内存中的始址和长度
  - 段表内容 = 段号 + 段长 + 本段在主存中的地址
- 地址变换机构（*变换过程*）
  - 逻辑地址A中取出段号S和段内偏移量W
  - 比较段号S和段表长度M，若S>=M,则产生越界中断，否则继续执行
  - 段号S对应的段表项地址 = 段表始址F+段号S*段表项长度，从该段表项中取出段长C，比较段内偏移量与C的大小判断是否出现越界
  - 取出段表项中该段的始址b,计算E = b+W，用得到的物理地址E去访问内存



- 段的共享与保护
  - 共享：两个作业的段表中响应表项指向被共享段的同一个物理副本来实现的　　纯代码或者可重入代码以及不可修改的数据可以被共享
  - 保护机制
    - 存取控制保护
    - 地址越界保护

段页式管理方式

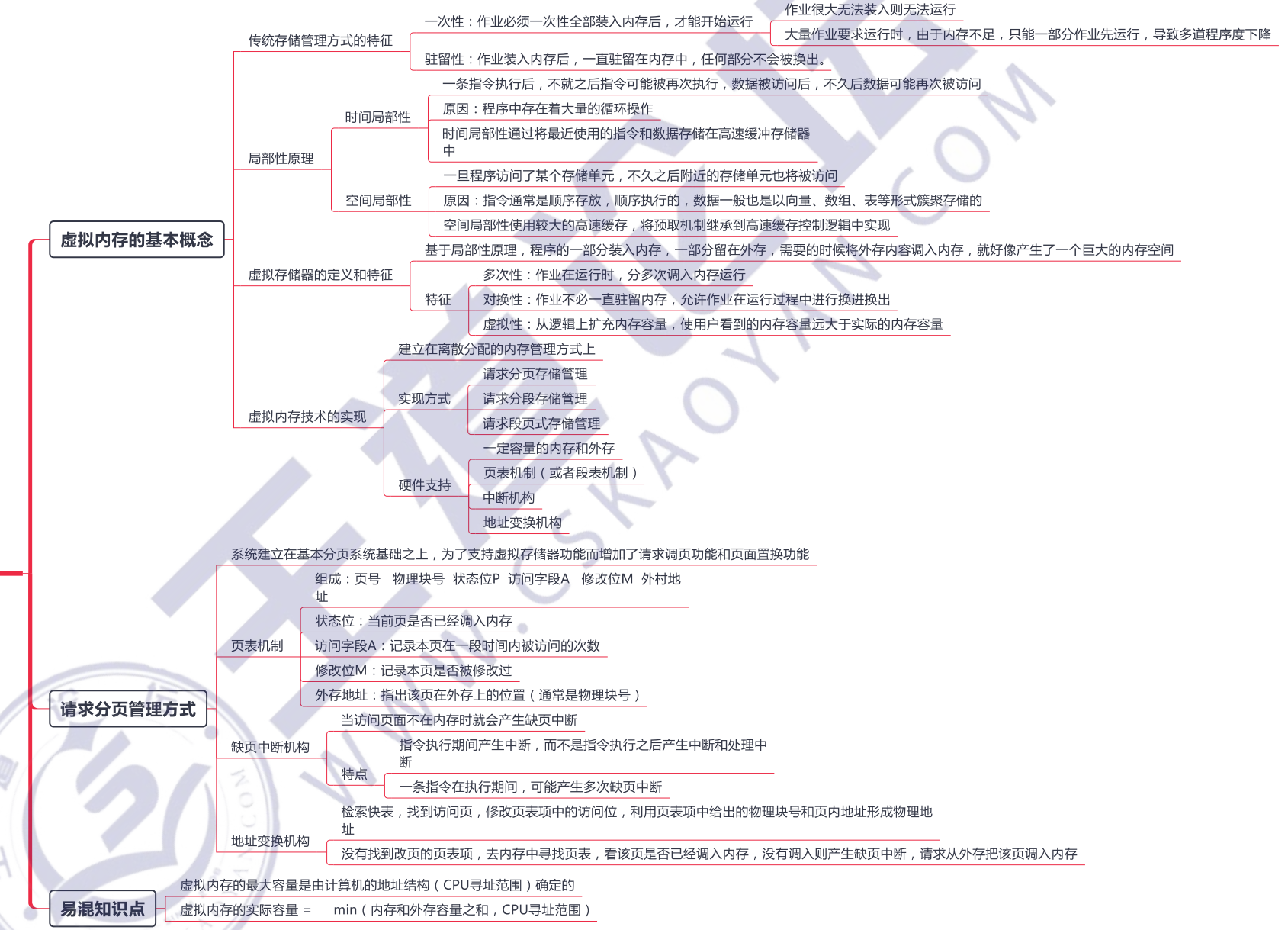
- 页式存储有效的提高内存利用率，分段存储能反映程序的逻辑结构并有利于段的分享，将这两种方式结合一下　　这种二者结合的方法经常在计算机理论中遇到
- 思想
  - 作业的地址空间首先被分成若干逻辑段，每段有自己的段号
  - 每个段分成若干大小固定的页
  - 对内存空间的管理仍然和分页存储管理一样
- 地址结构　　段号S+页号P+页内偏移量W
- 为了实现地址变换，系统为每个进程建立了一张段表，每个分段有一个页表　　一个进程中，段表只能有一个，页表可以有多个
- 地址变换方式



补充

- 不能被修改的代码称为纯代码或可重入代码（不属于临界资源）
- 分段与分页的区别
  - 分页对用户不可见，分段对用户可见
  - 分页的地址空间是一维的，分段的地址空间是二维的
  - 分页（单级页表）、分段访问一个逻辑地址都需要两次访存，分段存储中也可以引入快表机构
  - 分段更容易实现信息的共享和保护（纯代码问重入代码可以共享）
- 分段与分页优缺点
  - 分页管理
    - 优点：内存空间利用率高，不会产生外部碎片，只会有少量的页内碎片
    - 缺点：不方便按照逻辑模块实现信息的共享和保护
  - 分段管理
    - 优点　　很方便按照逻辑模块实现信息的共享和保护
    - 缺点
      - 如果段长过大，为其分配很大的连续空间会很不方便
      - 段式管理会产生外部碎片

3.2虚拟内存管理（上）



3.2虚拟内存管理（下）