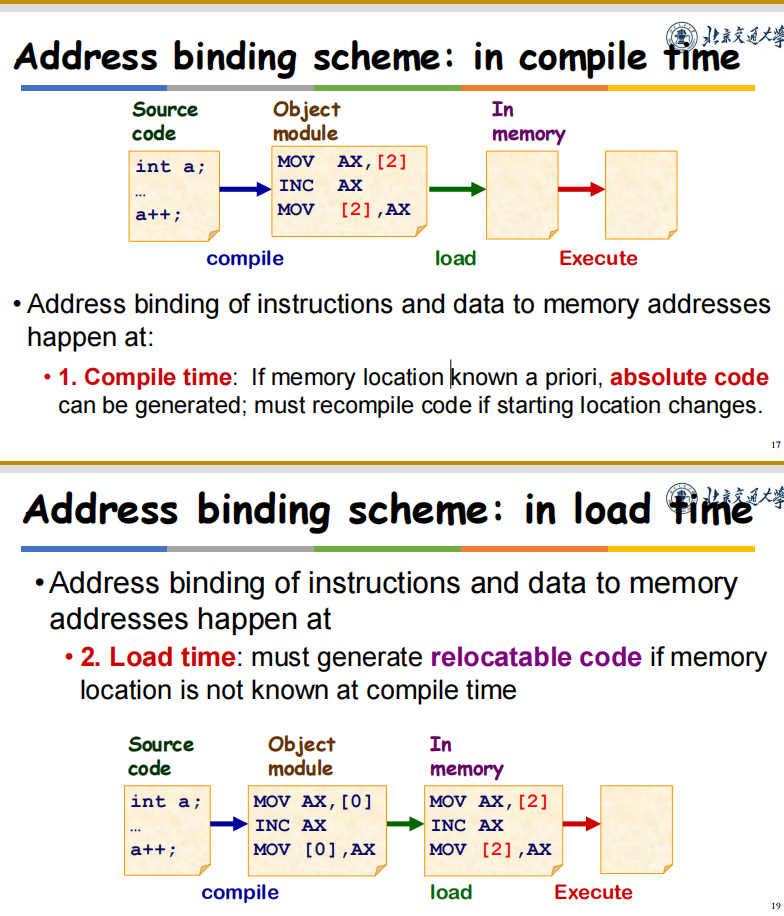
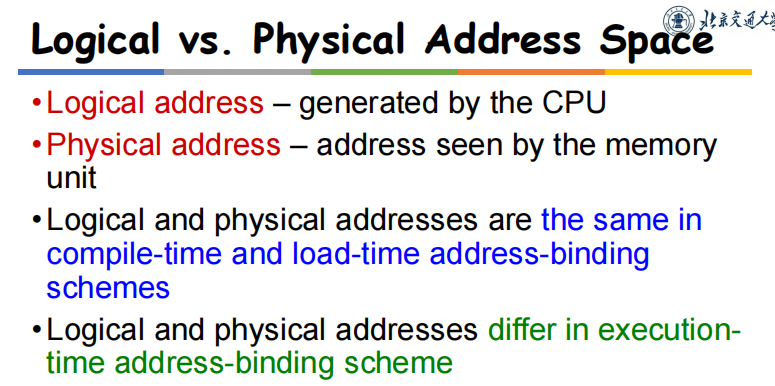
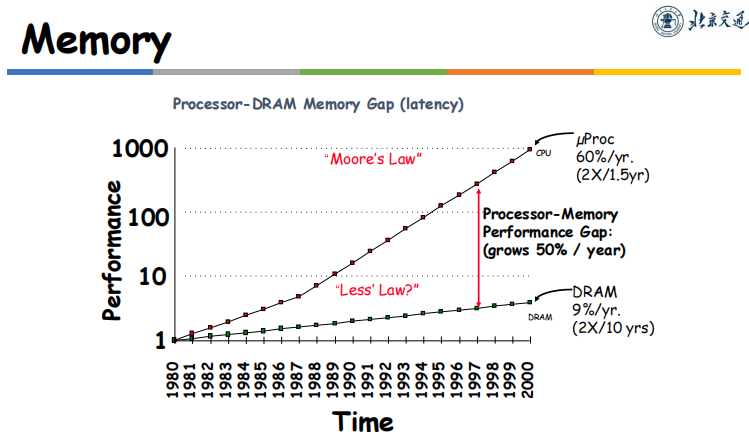
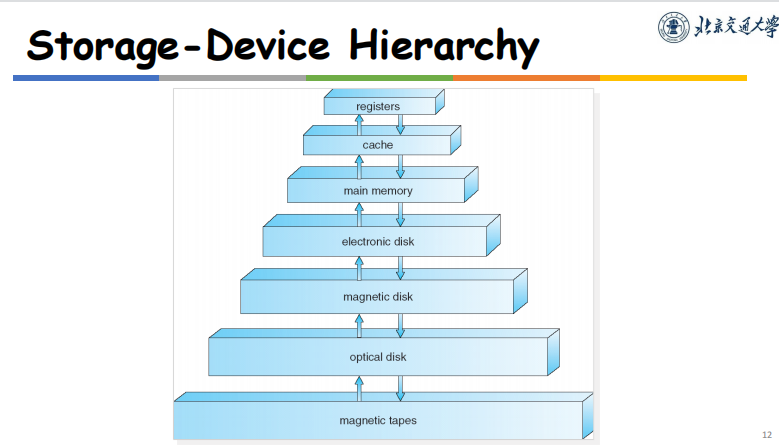
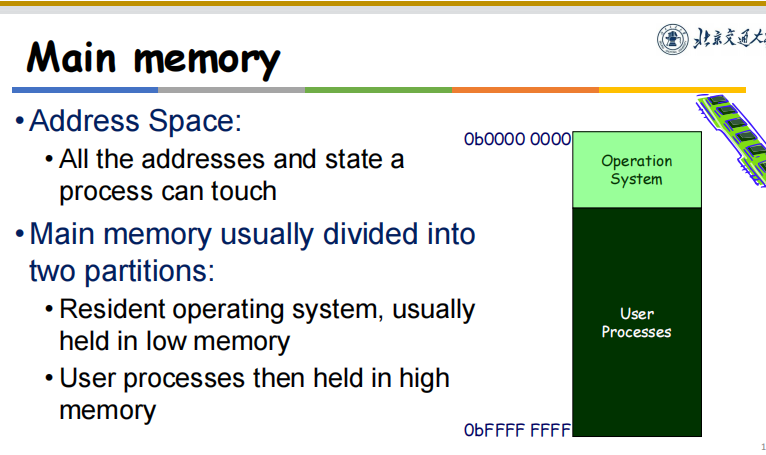
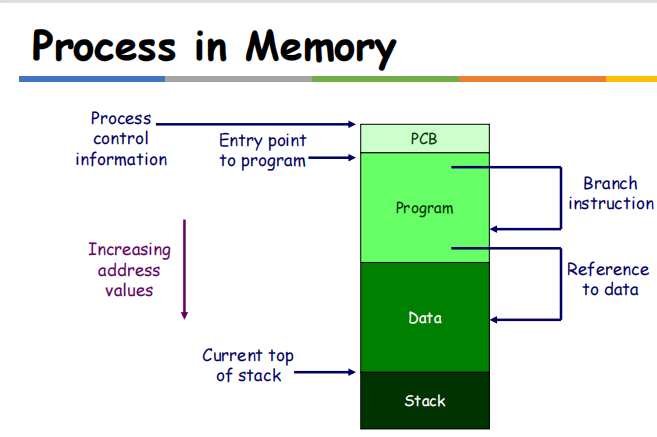
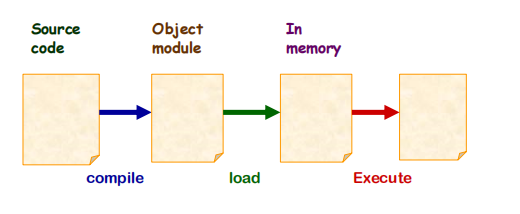
程序必须被将（从磁盘）带入到

内存和放置在一个过程中

被操纵

处理前后内存中的所有数据

内存中的所有指令，以便执行

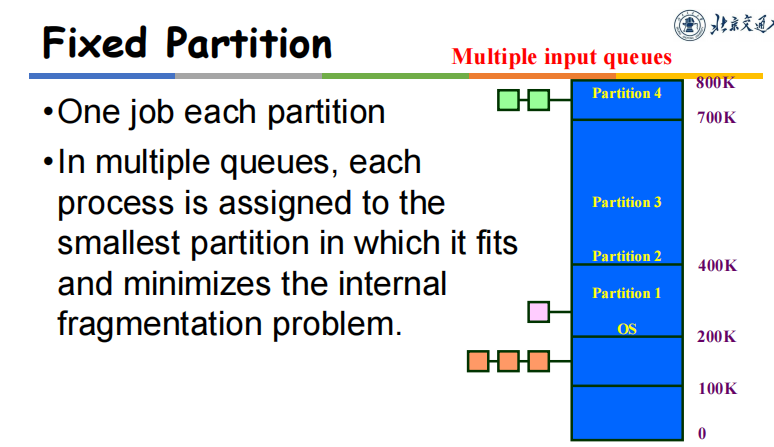
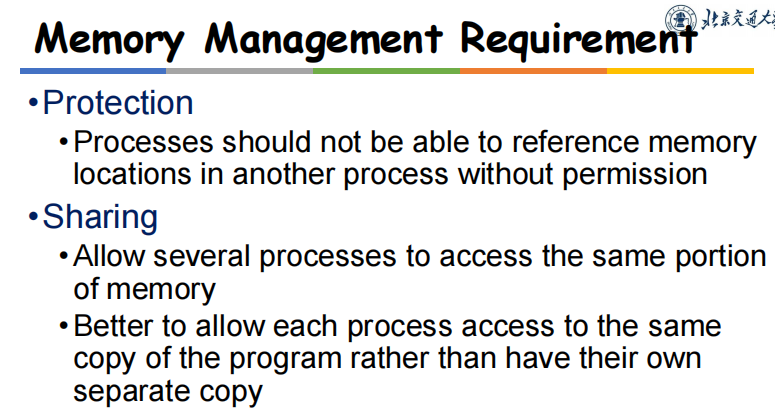
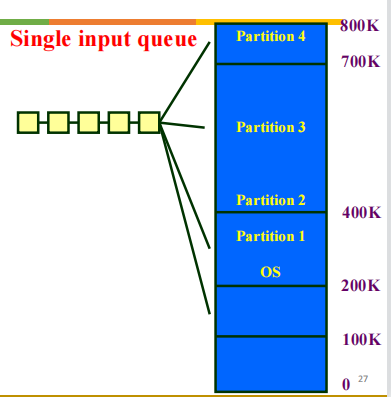
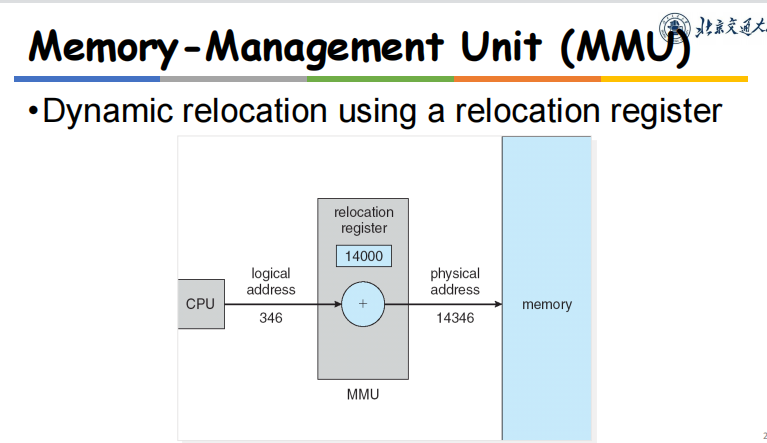
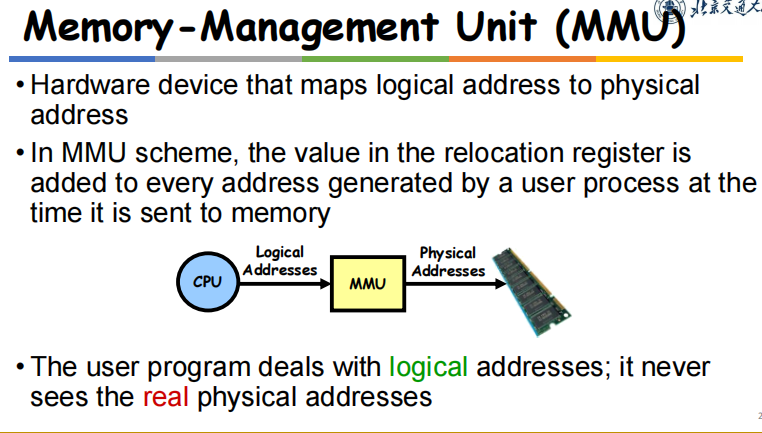
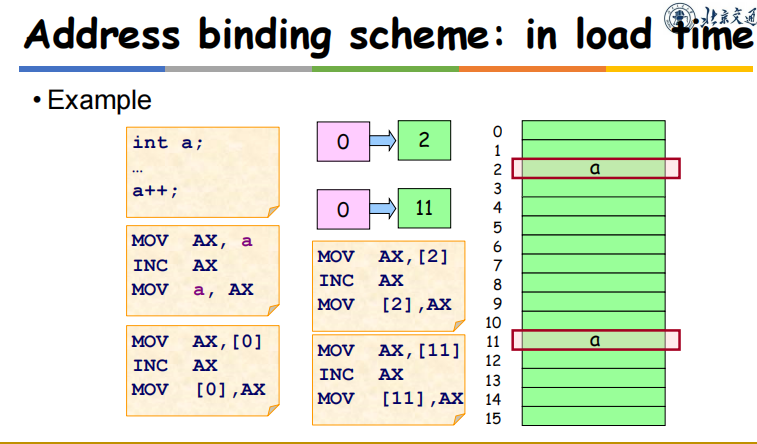
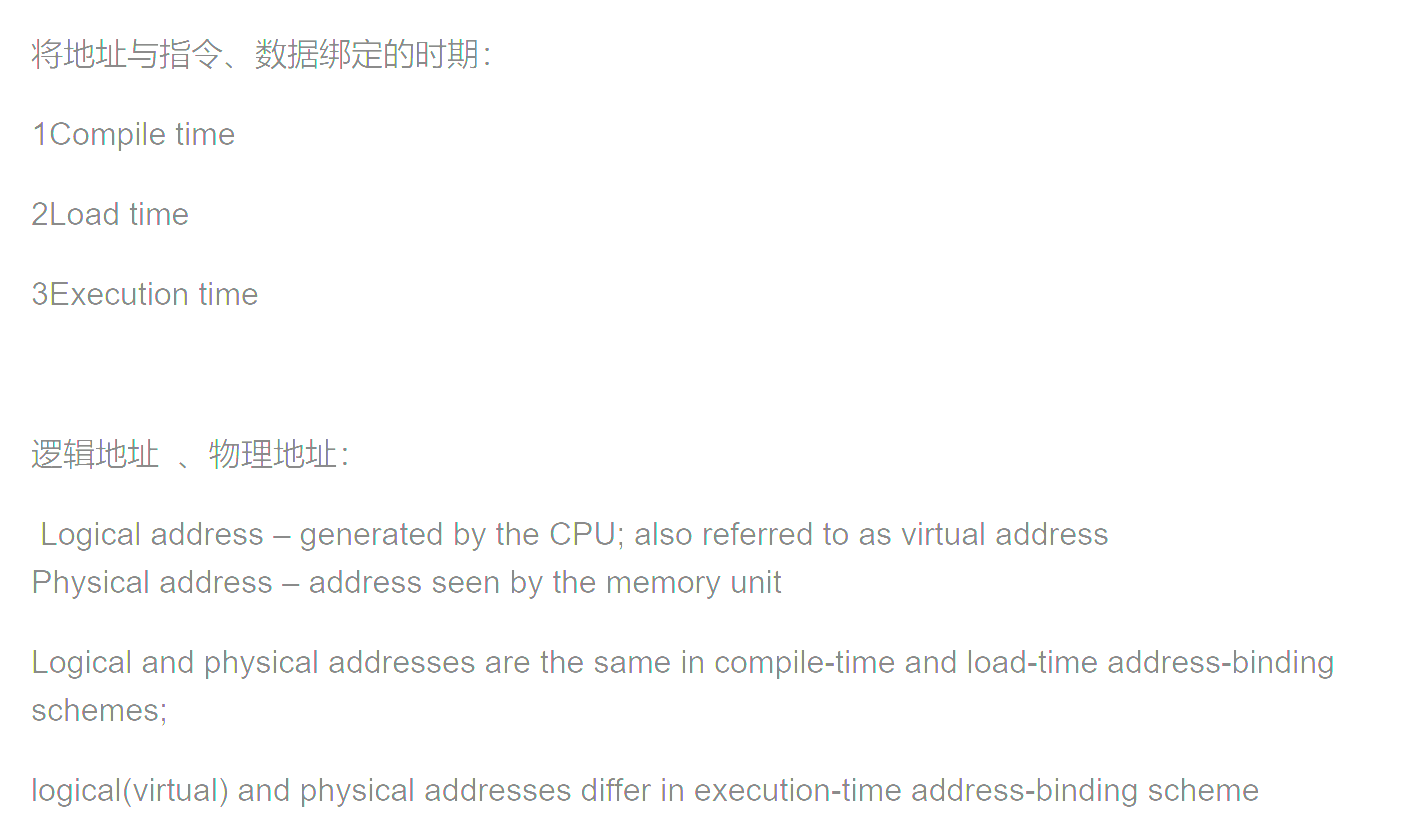


编译时间：如果内存位置已知，绝对代码

可以生成；如果启动位置发生更改，则必须重新编译代码。

2.加载时间：如果有内存，则必须生成可重新定位的代码

在编译时，位置是未知的

在单个队列中，该进程

会被分配给最小的一个吗

可用分区和

多编程的级别是

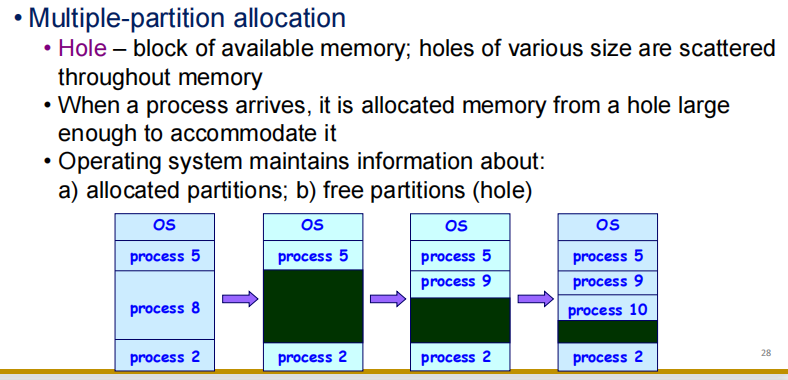
增加的

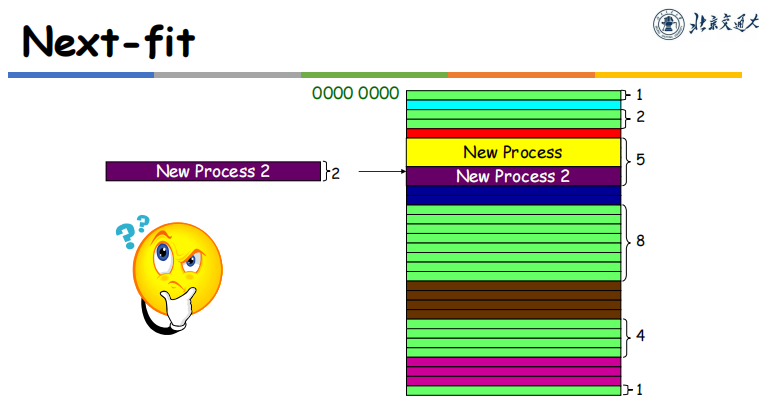
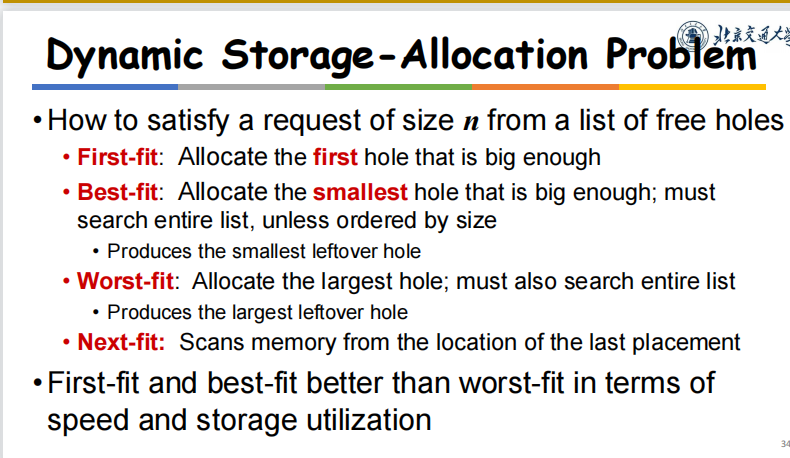
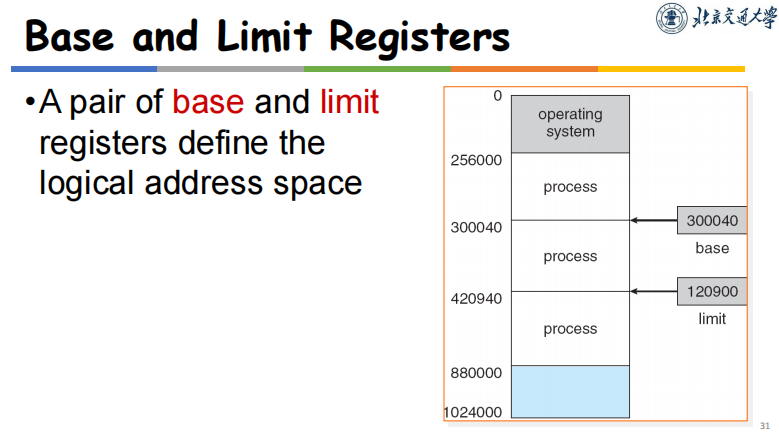
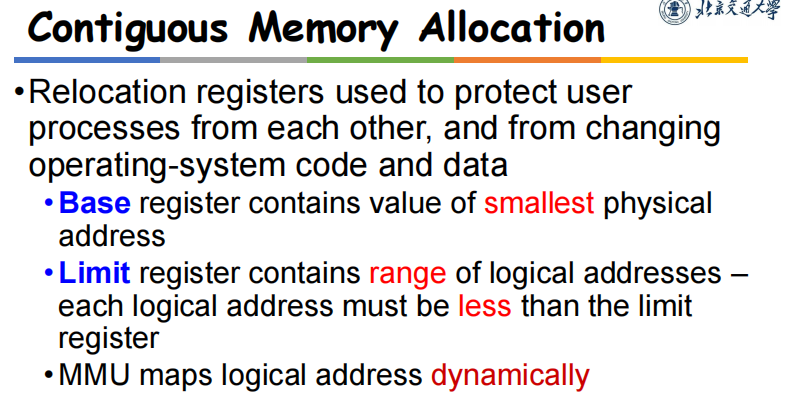
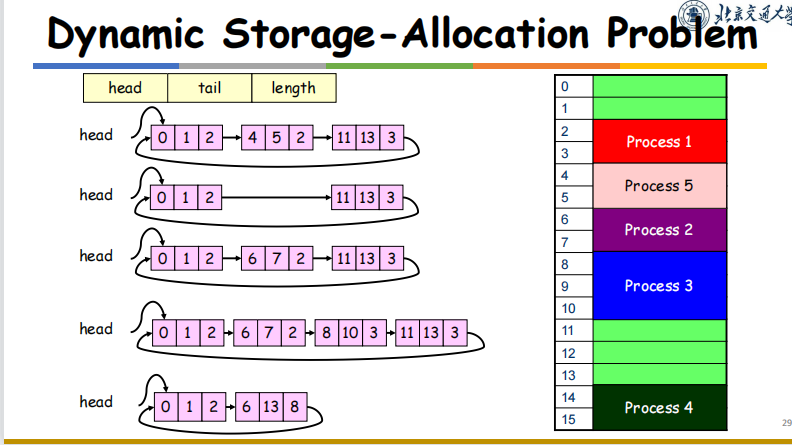
主内存的使用是

无效率的任何程序，没有

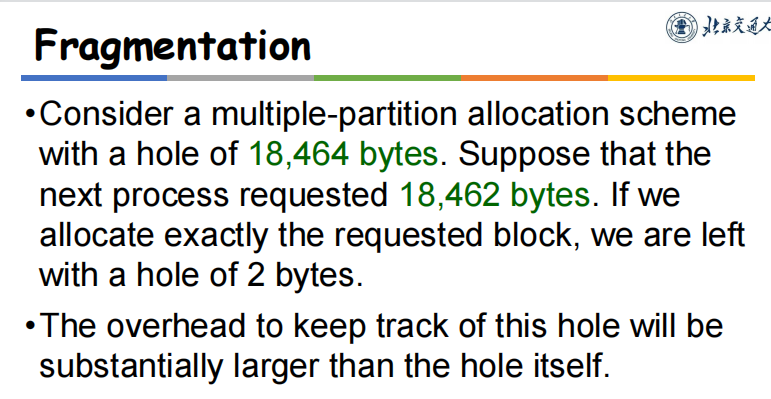
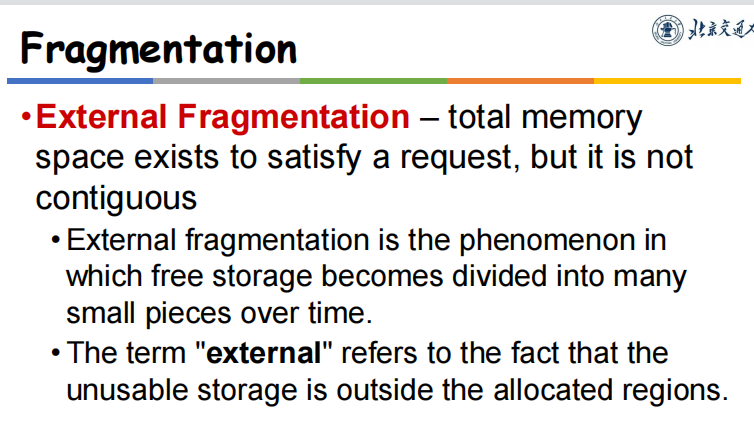
无论多么小，占据

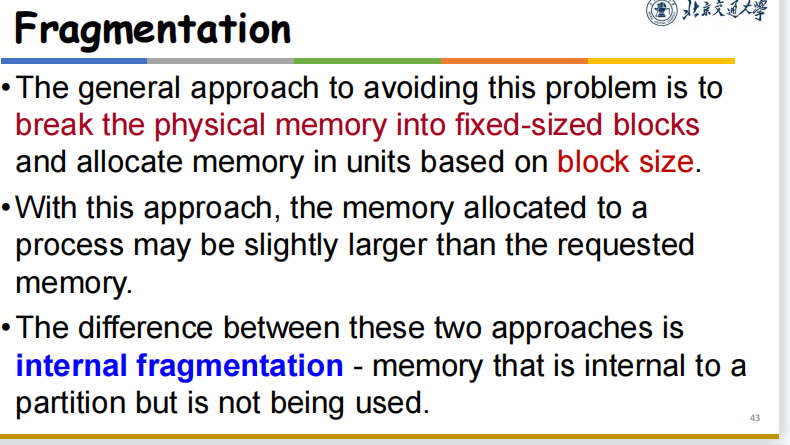
整个分区。





碎片

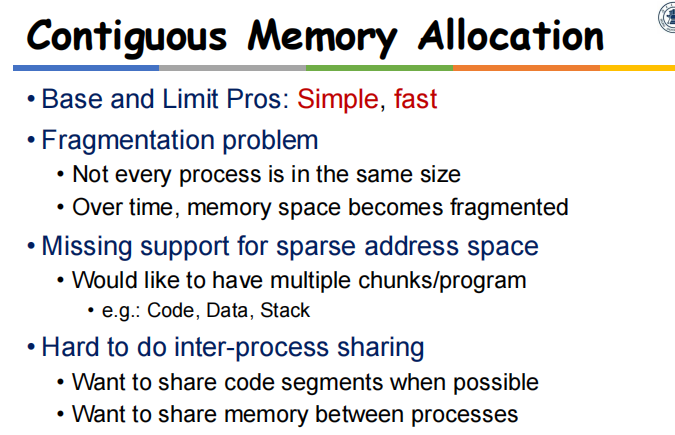
跟踪这个洞的开销将比洞本身大得多。



避免这个问题的一般方法是将物理内存分解成固定大小的块，并根据块的大小按单位分配内存。

使用这种方法，分配给进程的内存可能略大于所请求的内存。

这两种方法之间的区别是内部碎片化内存，它位于分区的内部，但没有被使用。



基础和限制的优点：简单，快速

沙痂碎裂问题

并非每个流程的大小都相同的

随着时间的推移，内存空间会变得碎片化

缺少对稀疏地址空间的支持

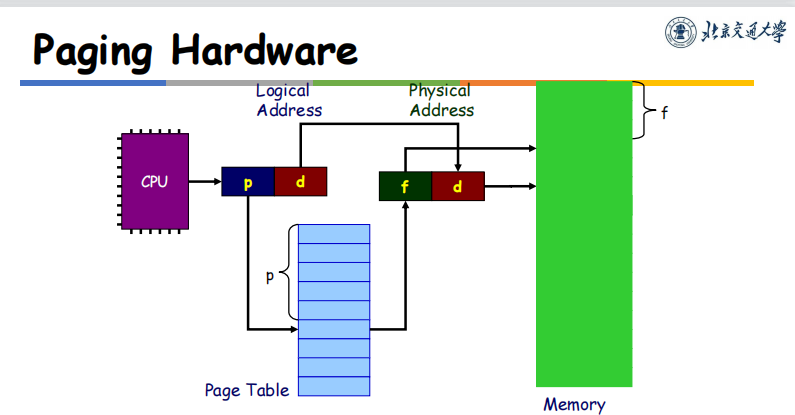
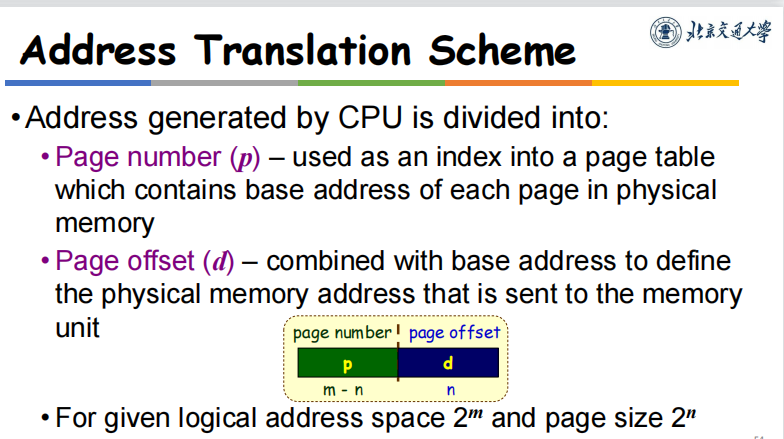
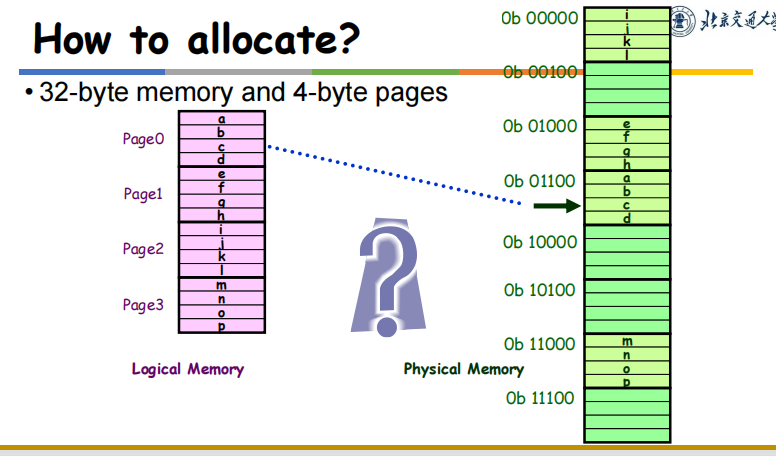
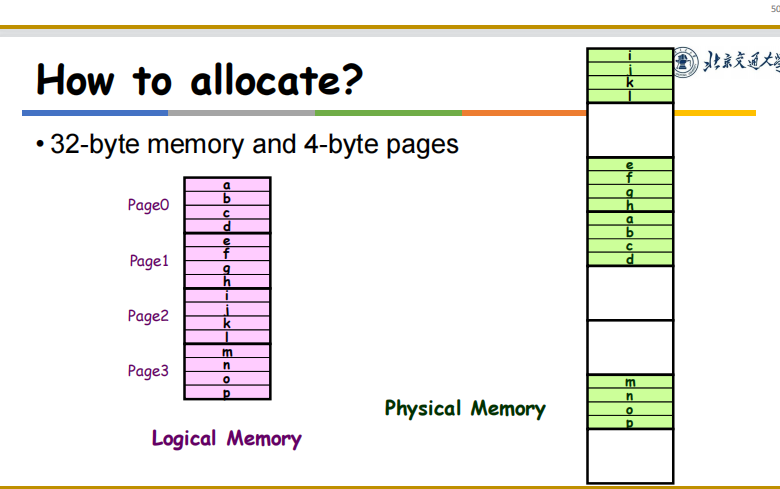
希望有多个数据块/程序

• e.g.：代码、数据、堆栈

很难进行进程间的共享

希望尽可能地共享代码段

希望在进程之间共享内存



该页表在主存中是连续的

页表的基本寄存器（PTBR）指向页表

页表长度寄存器（PRLR）表示页表的大小

在这个方案中，每个数据/指令的访问

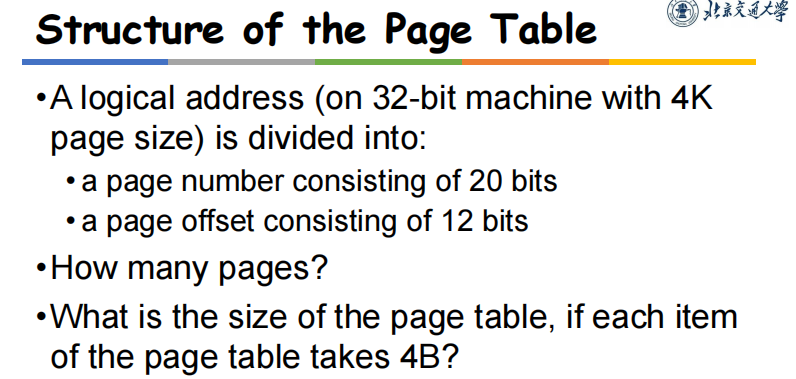
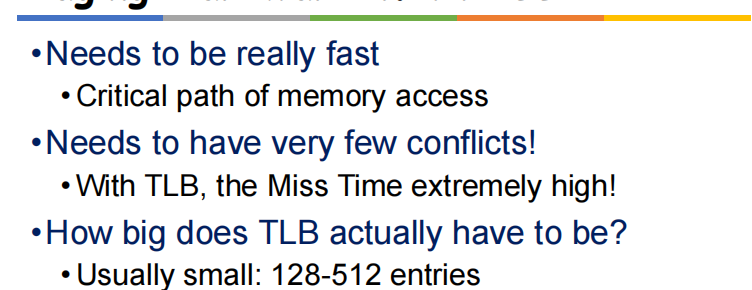
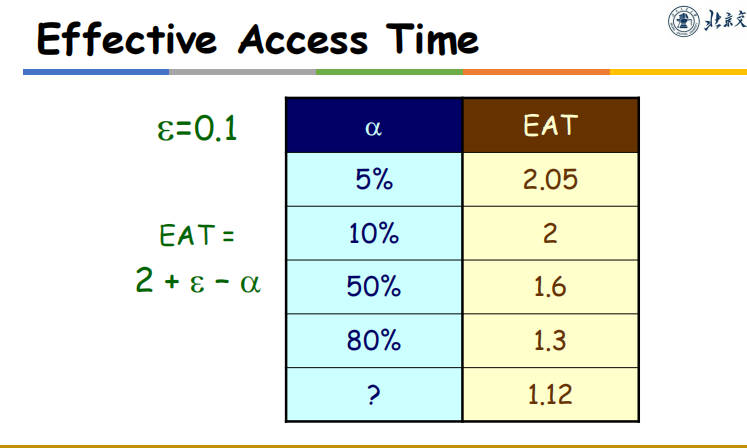
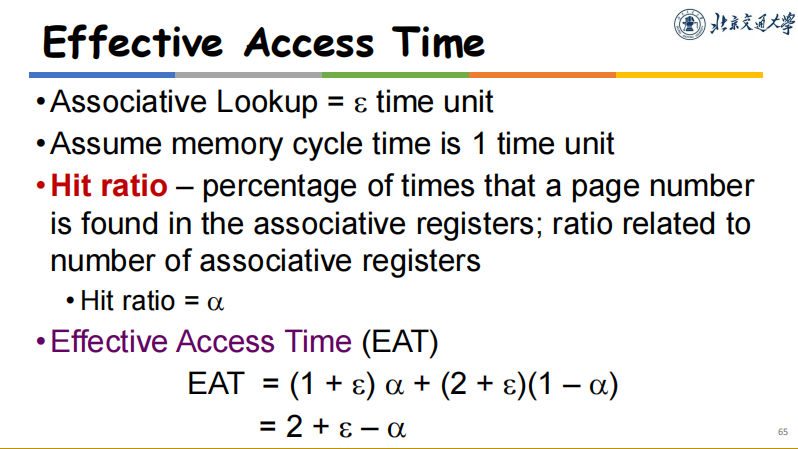
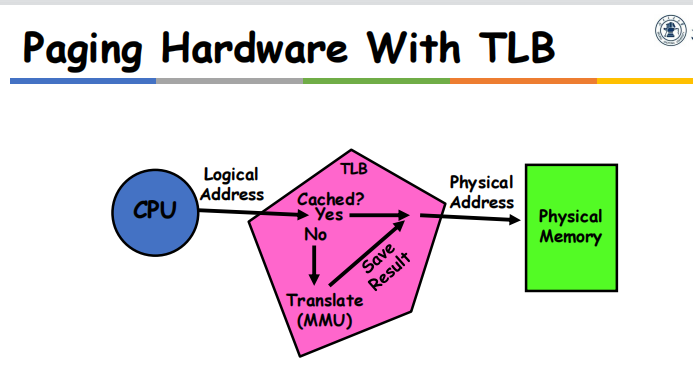
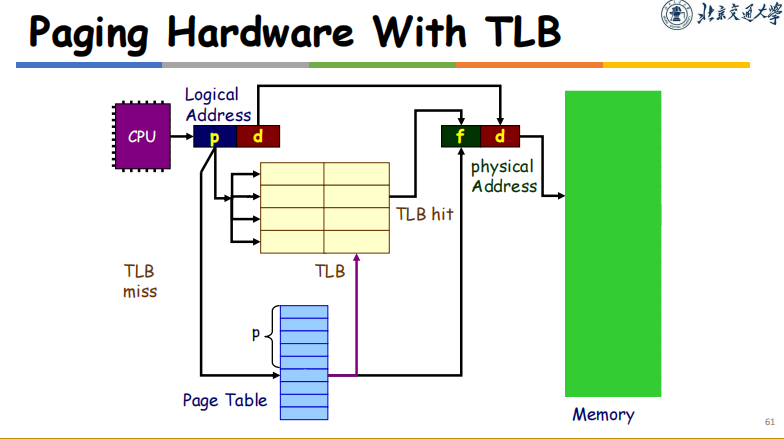
需要两个内存访问。

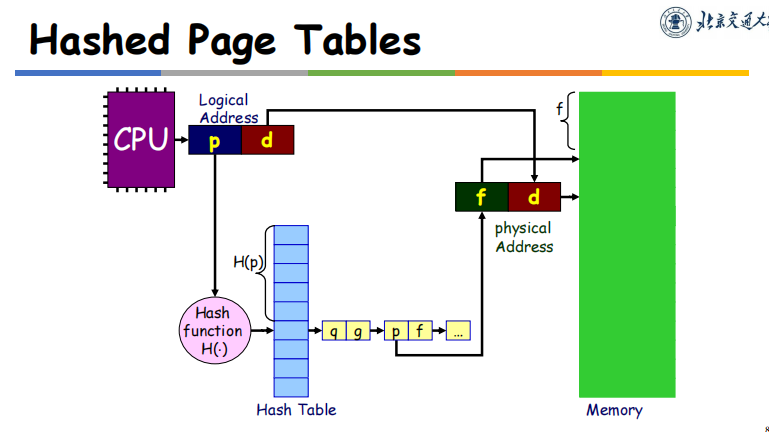
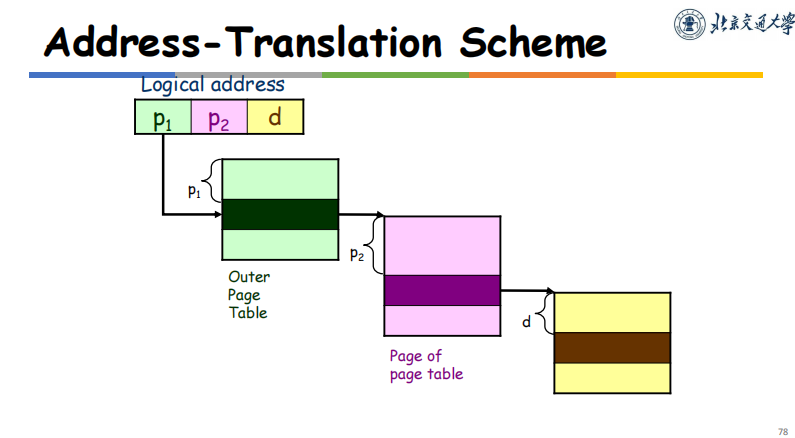
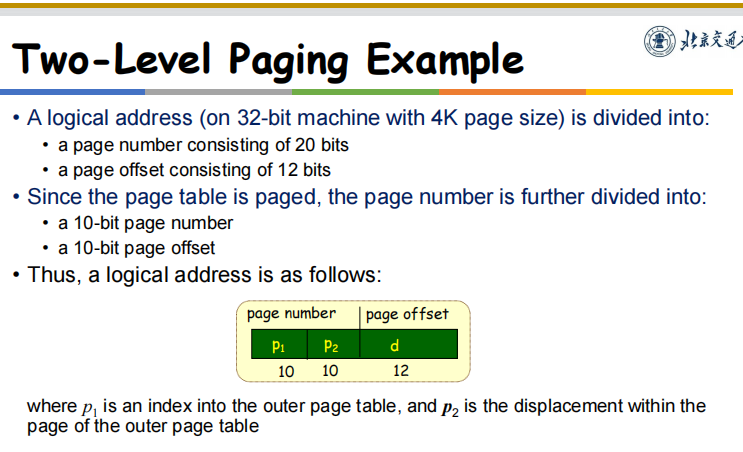
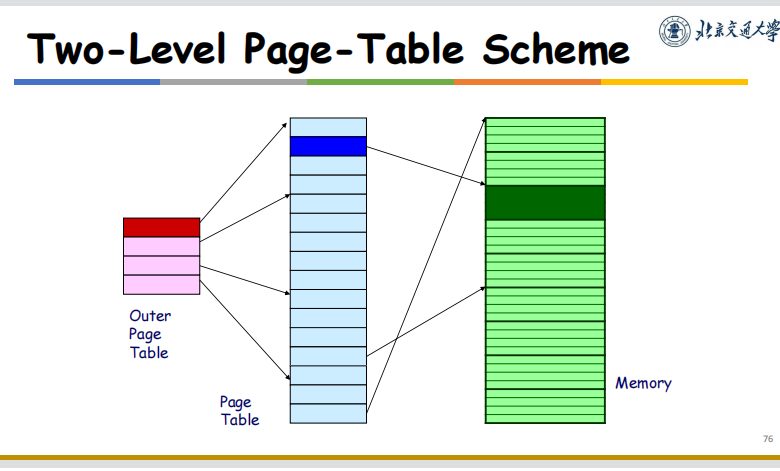
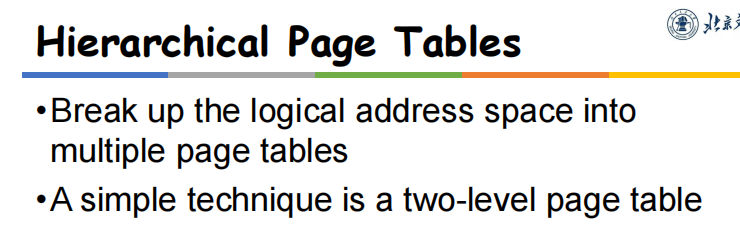
这两个内存访问的问题可能是

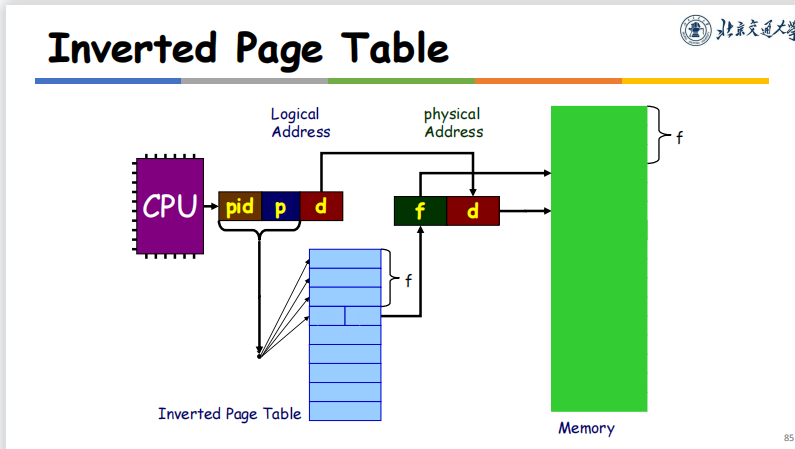
通过使用一种特殊的快速查找来解决

硬件高速缓存称为关联内存

或转换备用缓冲区（TLB）







减少了存储每个页表所需的内存，但增加了在发生页面引用时搜索该表所需的时间

