# 操作系统复习

第八章： 主存（内存管理）

第八章总纲：

* Background
* Swapping
* Contiguous Memory Allocation
* Segmentation
* Paging
* Structure of the Page Table
* Example: The Intel 32 and 64-bit Architectures
* Example: ARM Architecture

我们已经了解到

程序装载到内存才能运行（通常，程序以可执行文件格式保存在磁盘上）

多道程序设计模型（允许多个程序同时进入内存）

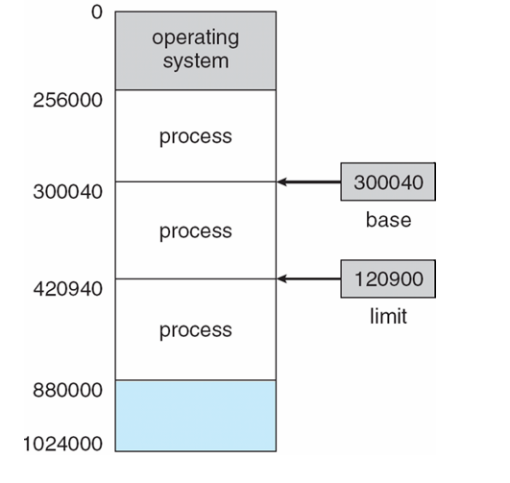
每个进程有自己的地址空间（一个进程执行时不能访问另一个进程的地址空间；进程不能执行不适合的操作）

#背景

CPU内置寄存器通常可以在一个CPU时钟周期内完成访问。

而内存访问则需要更多个时钟周期，CPU通常需要暂停（stall）。

由于内存访问很频繁，我们不能让CPU经常暂停，所以增加高速缓存（cache）。它用来协调速度差异。

 确保每个进程有自己的地址空间，因此需要基地址寄存器（Base register）和界限地址寄存器（Limit register）。

#地址绑定

进程中的地址不是最终地址。

在进程运行前无法计算出物理地址，因为：不能确定进程被加载到内存的什么地方。

\*\*\*\*\***地址绑定，就是将源程序中地址绑定在内存可重定位地址中的过程**

编译器通常将程序地址绑定到可重定位地址上，链接程序或加载程序再将这些可重定位的地址绑定成绝对地址。每次绑定都是从一个地址空间到另一个地址空间的**映射。**

**绑定可以发生在以下几种情况**

编译时：

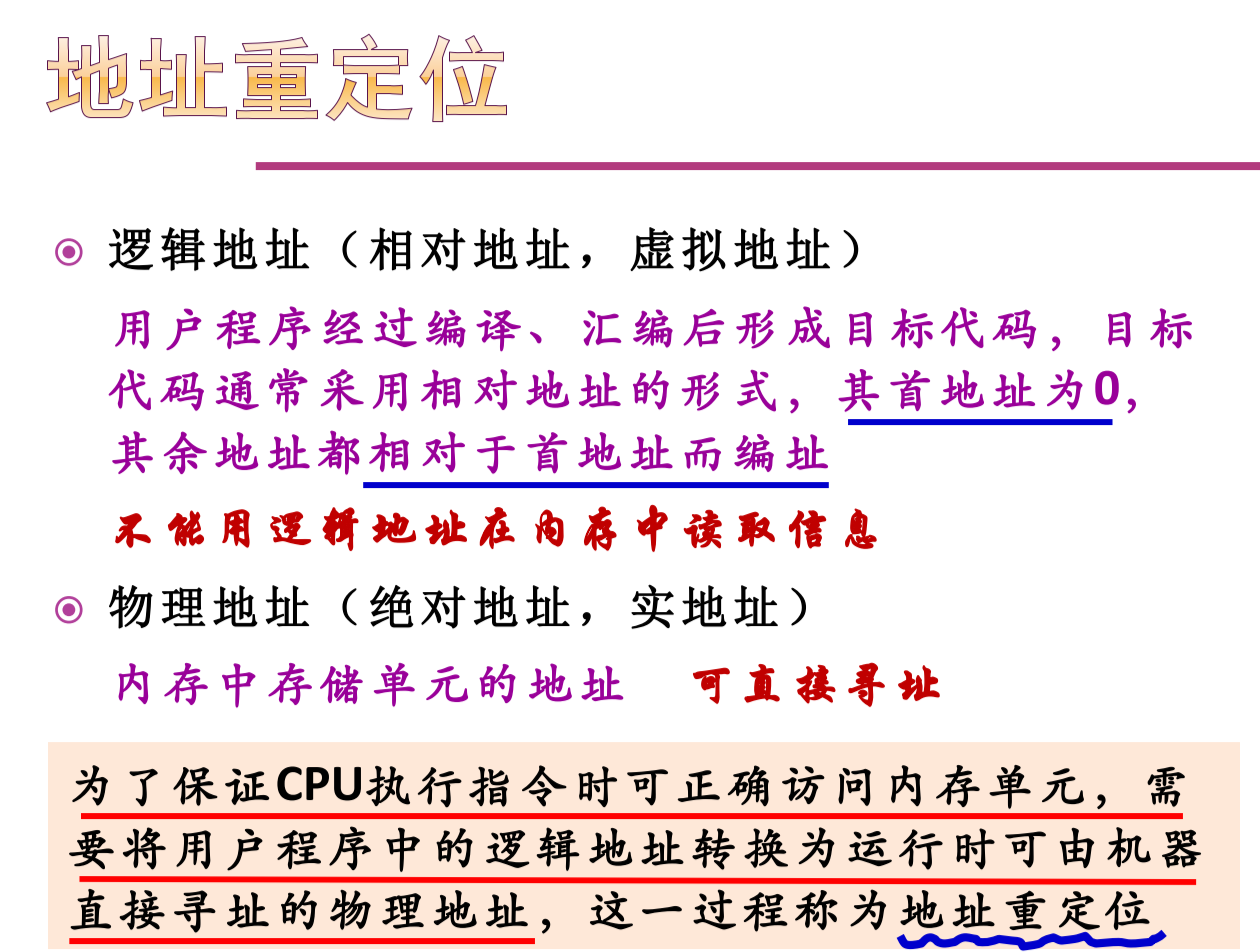
如果编译时就知道进程将在内存中的驻留地址，那么就可以生成**绝对代码**。

加载时：

如果编译时不知道进程将驻留在内存的什么地方，那么将生成**可重定位代码**。

执行时：

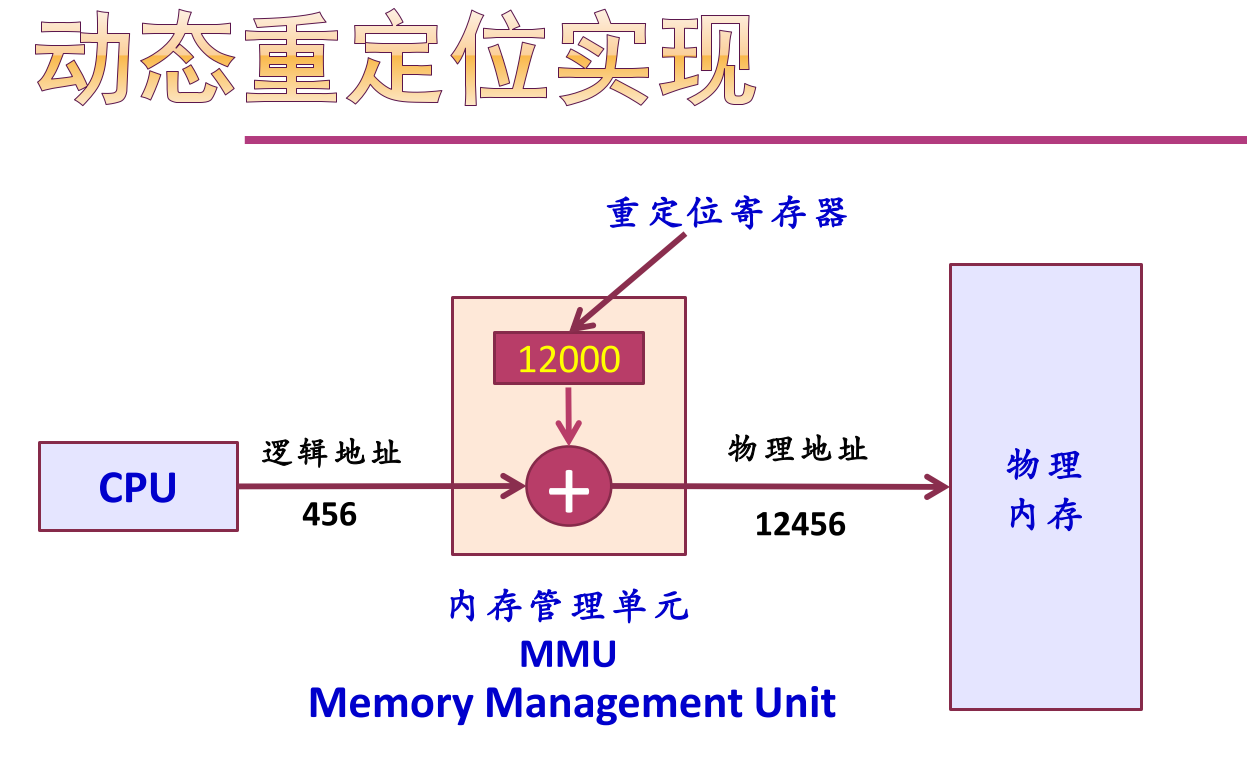
如果进程在执行时间可以从一个内存段移到另一个内存段，那么绑定必须延迟到执行时才进行。

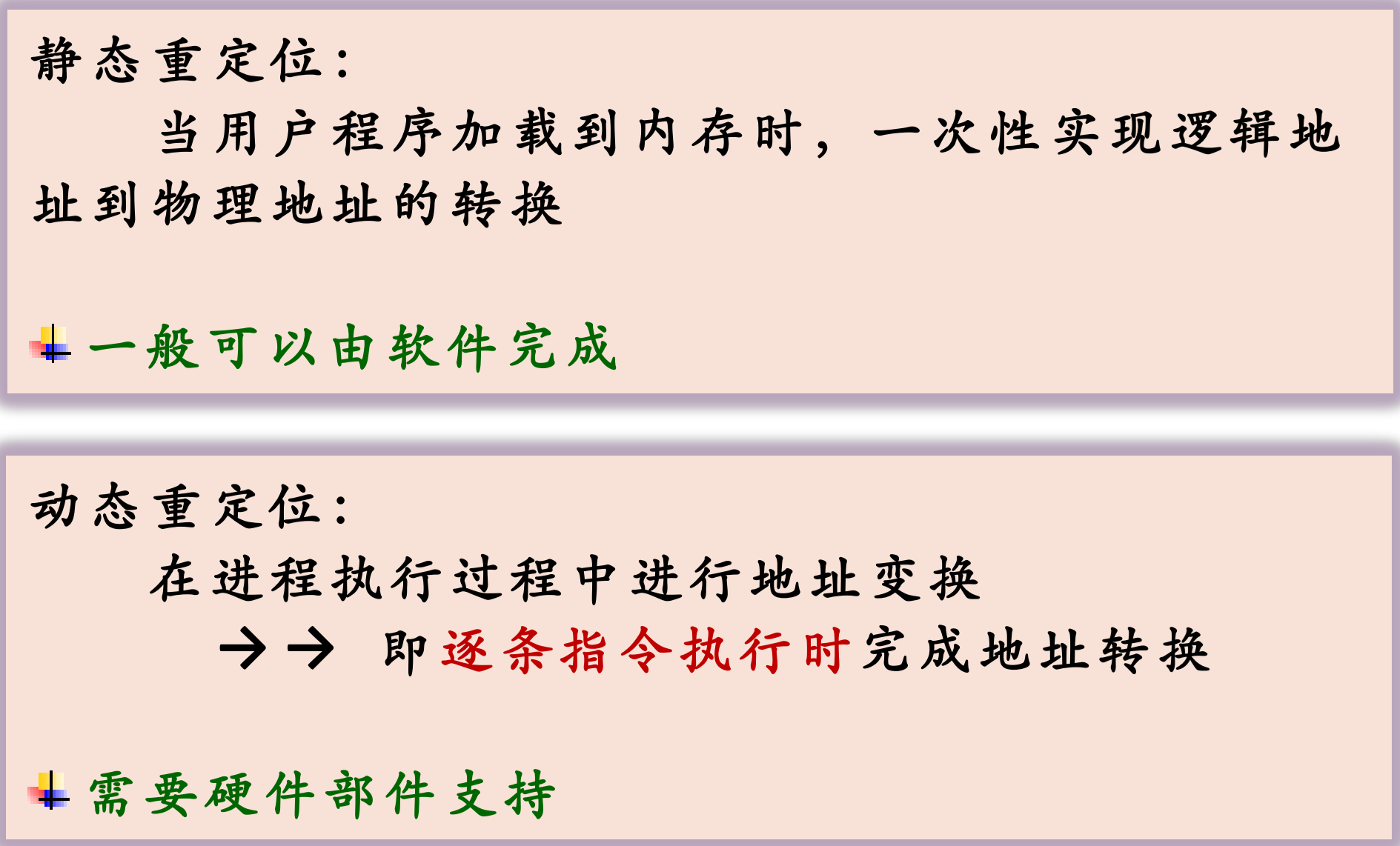


逻辑地址空间（logical address space）：由程序所生成的所有逻辑地址的集合称为逻辑地址空间；

物理地址空间（physical address space）：与这些逻辑地址相对应的所有物理地址称为物理地址空间。

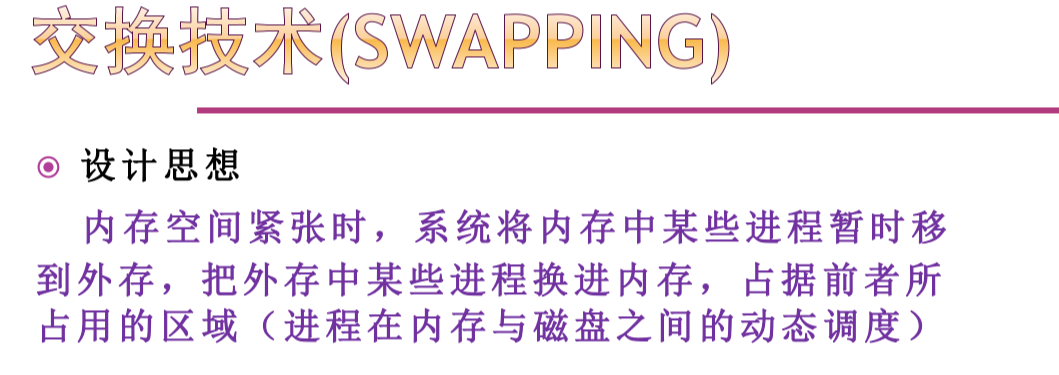
运行时，从逻辑地址到物理地址的映射是由MMU内存管理单元的硬件设备来完成的。





#交换

当内存不足时，如何管理？（即进程的地址空间大过内存）



进程可以暂时从内存中交换到备份存储上。

**Backing Store**—fast disk large enough to accommodate copies of all memory images for all users.

**Roll out roll in**—进程优先度不同，导致低优先度进程被高优先度进程交换出内存，当执行完高优先度进程后，低优先度进程被交换回内存。

**交换技术的局限：**

上下文交换时间过长：当下一个执行进程不在内存中时，需要从外存中交换进来，这需要大量的时间；

如果要换出进程，必须保证其完全在空闲状态；

在等待IO操作的进程不能被换出。

#连续内存分配

每个进程位于一个连续的内存区域。

#多分区方法

可变分区(variable-partition)：操作系统有一个表，用于记录哪些内存可用，哪些内存已被占用。

孔（hole）：可用的内存空间。

方法：

FIRST-FIT：从上往下，第一个可以放进去的孔，就放进去

BEST-FIT：找个最小可以放进去的孔放进去

WORST-FIT：找个最大可以放进去的孔放进去

#碎片问题

**外部碎片（External fragmentation）**

首次适应和最佳适应都有这个问题。随着进程的装入和移除，内存空间被分为小片段。**当可用的内存之和大于请求，但是并不连续，这就是外部碎片问题**。Total memory space exists to satisfy a request, but it is not contiguous

**内部碎片问题（Internal Fragmentation）**

**当分配内存时，块比所需内存大一点，分配进去之后就会形成小孔，这就是内部碎片问题**。Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used

外部碎片问题可用紧缩（compaction）解决。

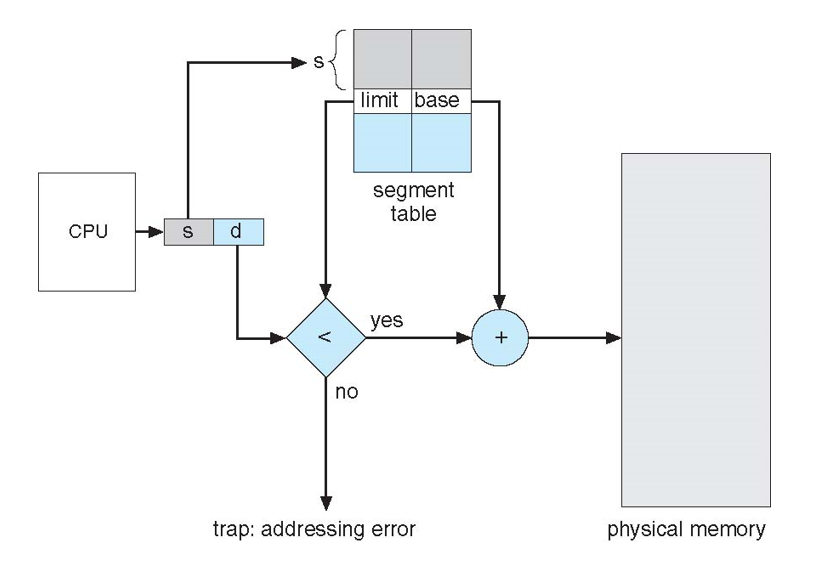
移动内存中的内容，以便所有空闲空间合并成一块。

！只用重定位是动态的并在运行时可采用。

#分段Segmentation

\*分段结构

《段号，偏移》

当知道一个分段结构的逻辑地址时。

先找到段表上对应的数据，然后查看，偏移量是否超过限制，如果超过则陷入trap，没有超过则让偏移量与其基地址相加。

#分页

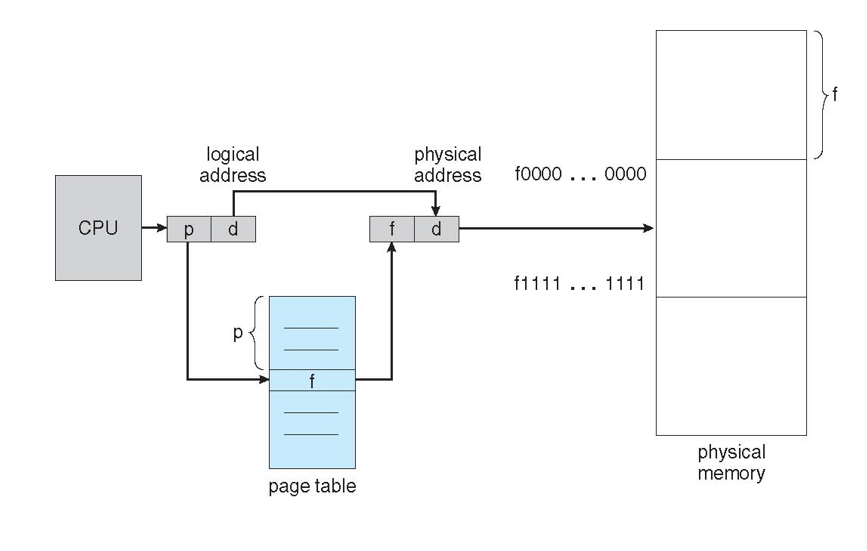
分页允许进程的物理地址空间可以是非连续的。

实现分页的基本方法涉及将物理内存分为固定大小的块，称为帧（frame）；

而将逻辑内存也分为同样大小的块，称为页（page）。

当需要执行进程时，其页从备份存储中调入到可用的内存帧中。备份存储也分为固定大小的块，其大小与内存帧一样。

《页号，偏移》CPU产生的地址是这个。（逻辑地址）



逻辑地址通过页号对应页表中的页号，来找到物理地址的帧号，然后通过偏移来计算具体位置。

\*页表的实现

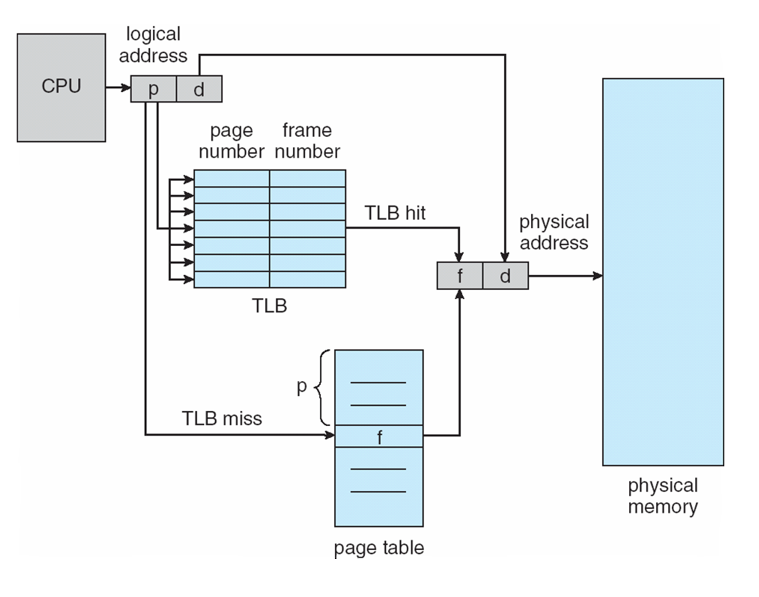
页表保存在主存中

页表基寄存器（PTBR）：指向页表

页表长度寄存器（PTLB）：包含页表的大小。

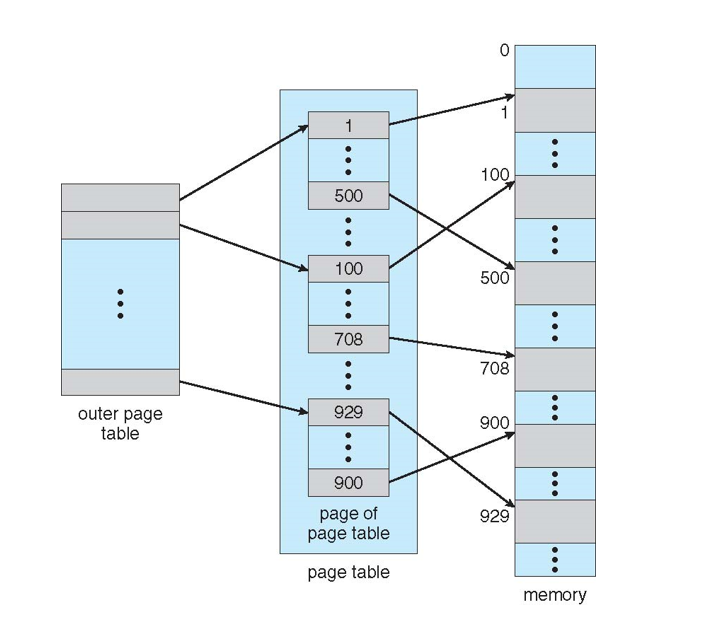
转换表缓冲区（Translation look-aside buffer TLB）

TLB与页表一起使用。它只包含页表中的一小部分条目。当CPU产生逻辑地址后，其页号提交给TLB如果找到页号，那么就得到了帧号，并用来访问内存，如果不在TLB中（TLB失效），则还是访问页表，将该没找到的页信息存储在TLB中，用以下次访问。

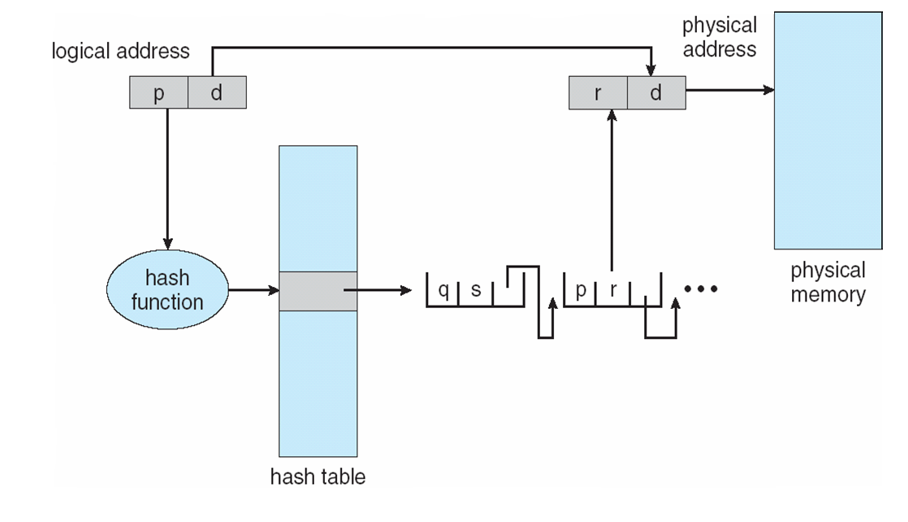


#页表结构

层次页表

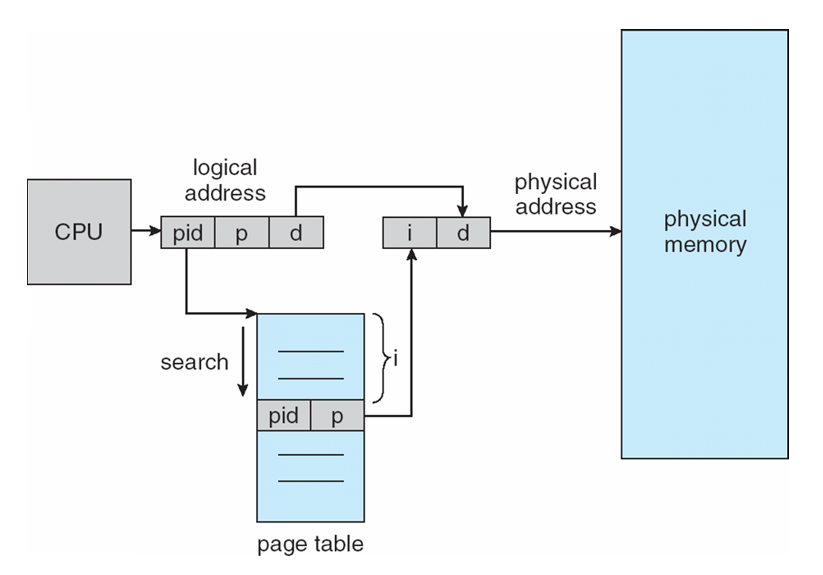


哈希页表



用于稀疏地址

倒排页表



倒排页表的建立：从物理地址空间出发，系统建立一张页表，页表项记录进程i的某虚拟地址与帧的映射关系。