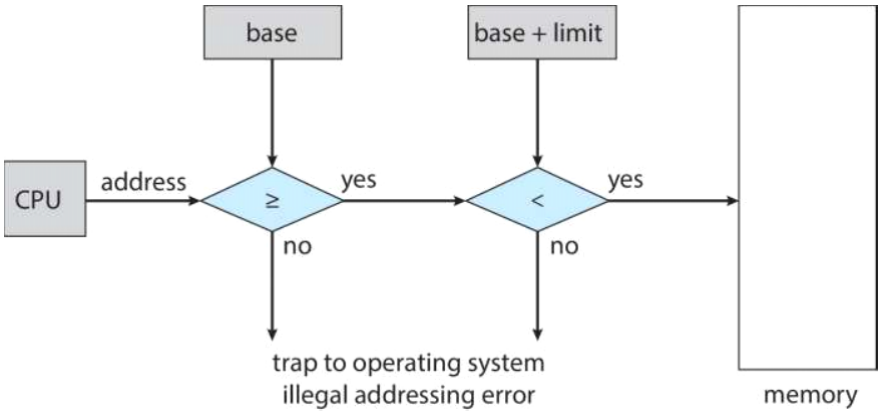
9 Memory Management

用**基址** (base address) 与**变址** (limit address) 表示进程的物理地址范围为 $[base, base + limit)$。用硬件访问保护机制来保证访问合法的空间。



为什么需要 **虚拟地址/逻辑地址 (virtual address / logical address)** 与 **物理地址 (physical address)** 两套地址体系?

- CPU register 空间有限，不能存储太大的地址;
- 每台机器硬件配置不同，物理地址大小不同;
- 用物理地址编程会给程序员开发带来较大难度。

对应的空间被称为 **逻辑地址空间** 和 **物理地址空间**。

内存管理单元 (Memory Management Unit) 管理从逻辑地址空间到物理地址空间的映射转换。

地址绑定 (address binding) 技术是指将逻辑地址与物理地址通过映射关联起来。

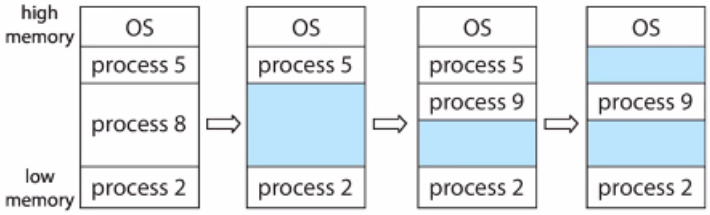
动态加载 (dynamic loading)：并不是在一开始将所有模块都加载进内存，而是用到了模块再动态加载进内存。

动态链接 (dynamic linking)：不同的文件可能共享同一个模块，可以仅加载一次，在其他文件用到时，采用动态链接，就不需要重新加载。经常被称为共享库 (shared library)。

连续内存空间分配 (Contiguous Allocation) 问题：

- 一部分区域分配给操作系统（带有中断向量表），可以在内存的低地址空间；但是现在一般操作系统习惯放在高地址空间。
- 一部份区域分配给用户进程，一般在内存的高地址空间;
- 希望每个进程的地址空间是一段连续的内存地址。

内存空洞/碎片 (hole / fragmentation) （图中蓝色区域）



50% 定理：碎片数量大约是分配的连续空间数量的 50%。

最先适应算法 (First-fit)：分配在第一个足够大的洞;

最优适应算法 (Best-fit)：分配在最小的足够大的洞（如果不对于块大小进行组织，需要搜索整个列表）；

最差适应算法 (Worst-fit)：分配在最大的足够大的洞（如果不对于块大小进行组织，需要搜索整个列表）。

外部碎片：进程释放后留下的不连续的空间，导致一个进程可能会分配到的一些不连续空间；

内部碎片：分配的空间可能略大于需要的资源，那么分配的空间的内部的碎片称为内部碎片。

页式管理 (paging)

- 物理内存会被分为很多个**页框 (帧)** (frame)，范围在 512Bytes ~ 16MBytes之间，默认为 4KBytes；比较大的页称为 **大页** (huge page)，更适合大的连续数据处理。
- 逻辑地址空间划分相同大小的数据块，称为**页** (page)。
- 运行一个 n 个页的程序，需要找到 n 个空的帧来存储程序。
- 保存帧和页的映射关系（从逻辑地址到物理地址）的表称为**页表**。
- **页表**在内存中的表示用 **PTBR (Page Table Base Register)** 和 **PTLR (Page Table Length Register)** 来表示，每个进程有一个页表。
- 在页表中加入 **Valid-invalid bit**（有效-无效位）表示页表的此位是否有效。
- 原因：连续访问地址的性能比随机访问的性能好很多 (*sequential accesses* > *random accesses*)

逻辑地址空间表示：设逻辑地址空间 2^m，页大小 2^n。

- **页号 (page number, p)**：$m-n$ 位二进制，在页表中查找基地址的索引；
- **偏移量 (page offset, d)**：n 位二进制，页内偏移地址。

逻辑地址转物理地址

- 偏移量不变，页号进行映射转换（查页表）。
- 对页号按顺序排列组织成页表，直接根据 p 的值查行数即可。
- 具体参见 CS359 Chap 3 Notes.

空闲页框表（队列）：存储当前空闲的页框列表，用来分配给进程。

快表 (*Translation Look-aside Buffers*, *TLB*)：关于页表的 cache。

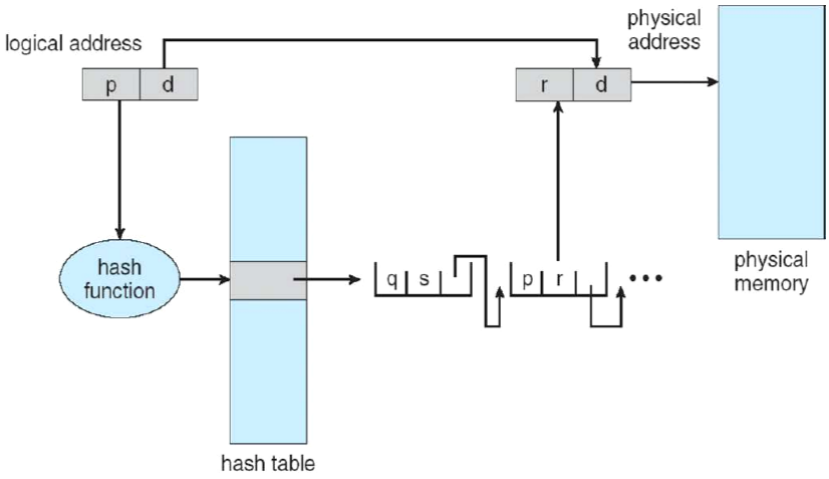
有效访问时间 (*Effective Access Time, EAT*)：执行 virtual address 到 physical address 映射的访问时间。

多级页表：如果存储一个完整的页表需要的内存太大，所以使用多级页表（某些级的页表如果全为空就不需要存储）。

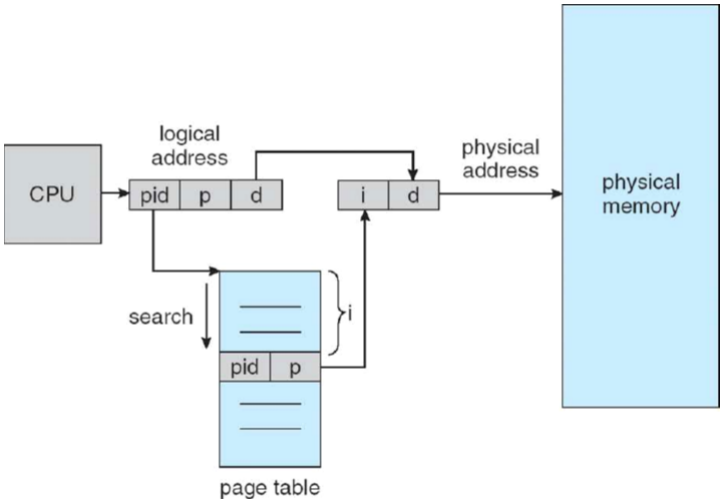
- **二级页表**：外层页表 (outer page table) 地址 p_1 + 内层 (inner page table) 页表地址 p_2 + 页内偏移量 d。
- 可以设计不同的页大小以存储不同大小的文件，因此需要设计许多不同的页表来处理不同的解码方式（如Intel IA-32）

其他页表：

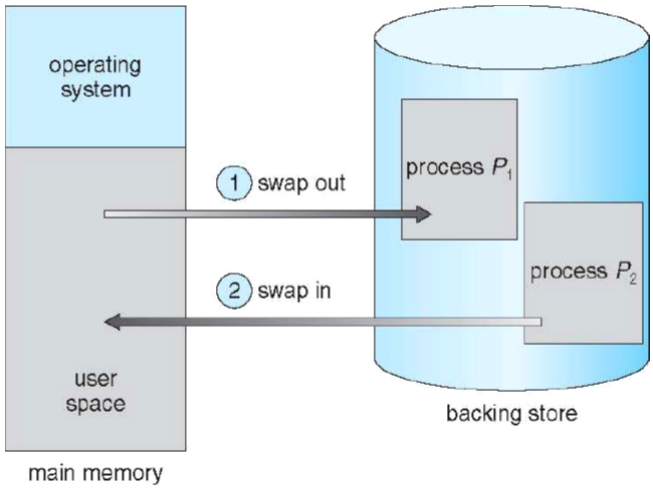
- **哈希页表** (hashed page table)：用哈希表组织映射；



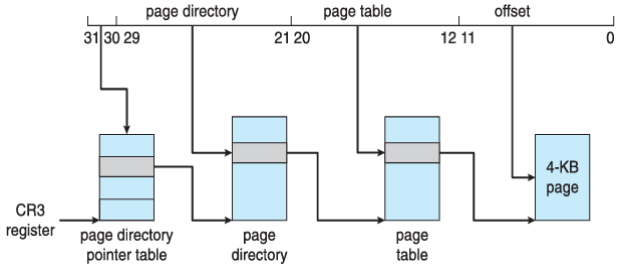
- **逆页表** (inverted page table)：本来是按照 page number 排序，现在按照 frame number 排序，进行查找。



后备区 (backing store)：硬盘上的划分的一块交由内存管理的空间。当页表需要交换时候，将被交换的页表放入后备区（硬盘）。



PAE (Page Address Extension) 技术，用 32-bit 的虚拟地址利用多于 4GB 的内存空间，也就是说，物理地址大于 32-bit。实际上，在任意一个时刻，程序可见的内存空间只有 4GB，但是可以动态调换程序可见的内存空间。



那么，为什么要设计成三级页表？因为物理地址大于32-bit，每一个 page table entry 只能设计成 64-bit（因为要映射到物理地址），从而数量减半。对于 page directory 和 page entry 都相同，数量减半后多出来的两位就需要第三级页表了。