虚拟内存2回课

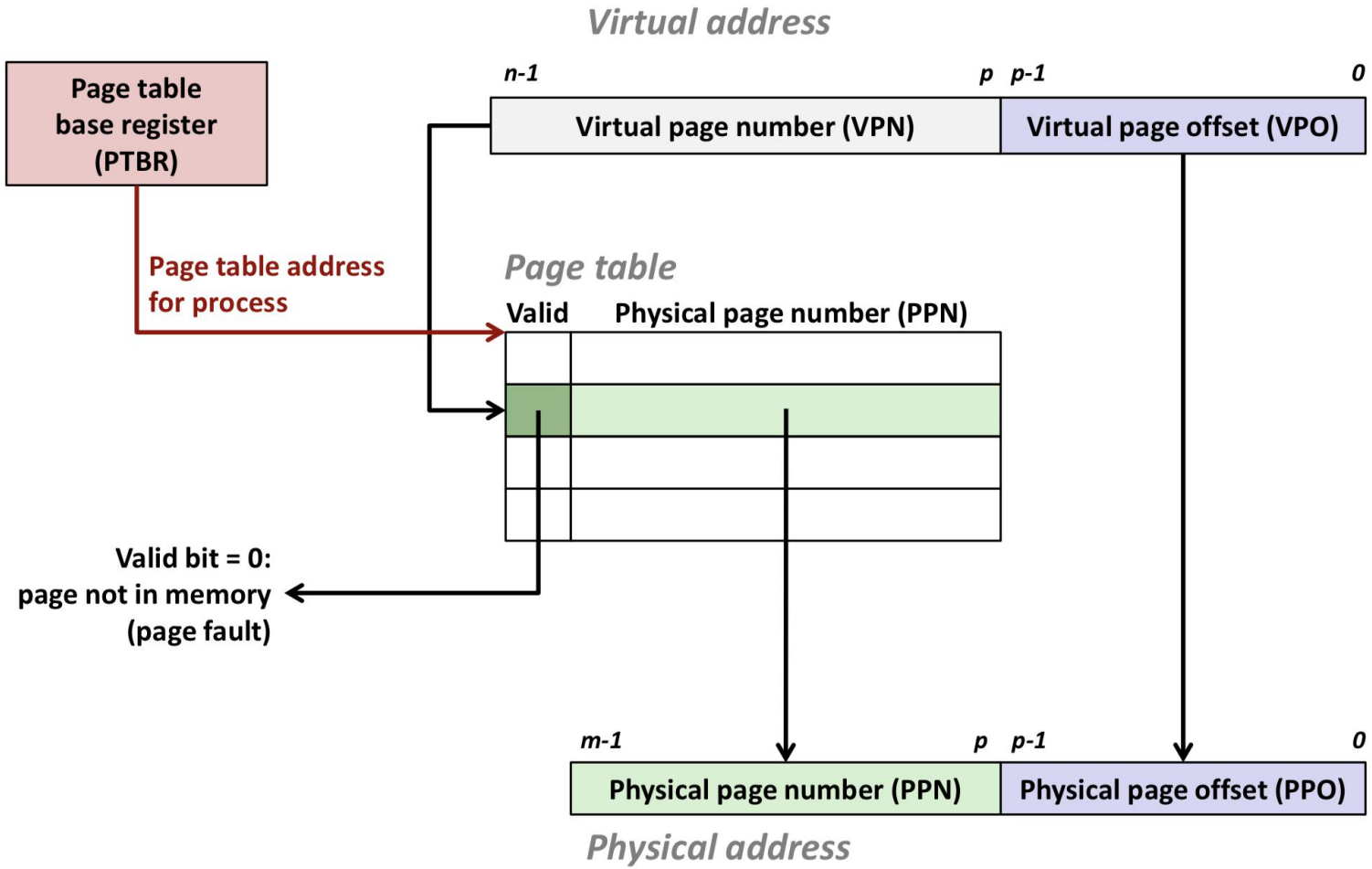
Introduction to Computer Systems
Nov. 28st

叶炜宁

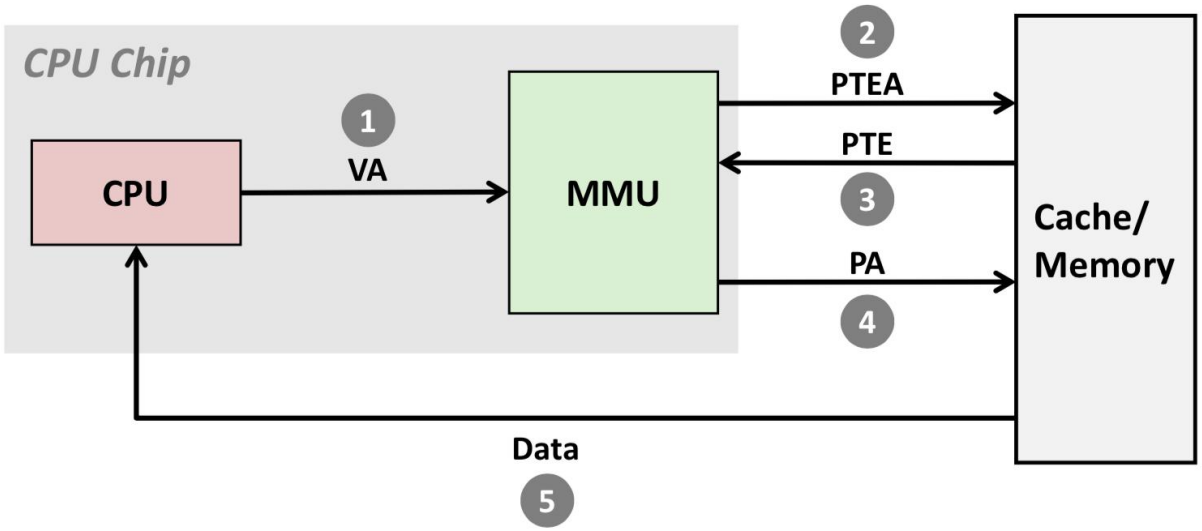
Today

- 地址翻译
- Core i7/Linux 内存系统
- 内存映射

地址翻译

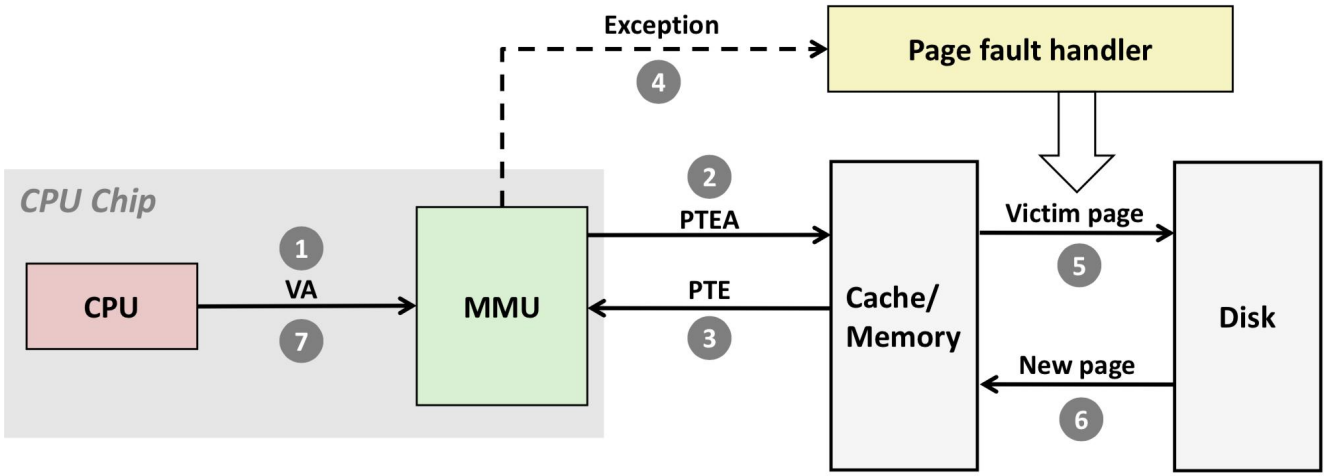

Virtual address
n-1
p
p-1
0
Virtual page number (VPN)
Virtual page offset (VPO)
Page table base register (PTBR)
Page table address for process
Page table
Valid
Physical page number (PPN)
Valid bit = 0: page not in memory (page fault)
m-1
p
p-1
0
Physical page number (PPN)
Physical page offset (PPO)
Physical address

页命中



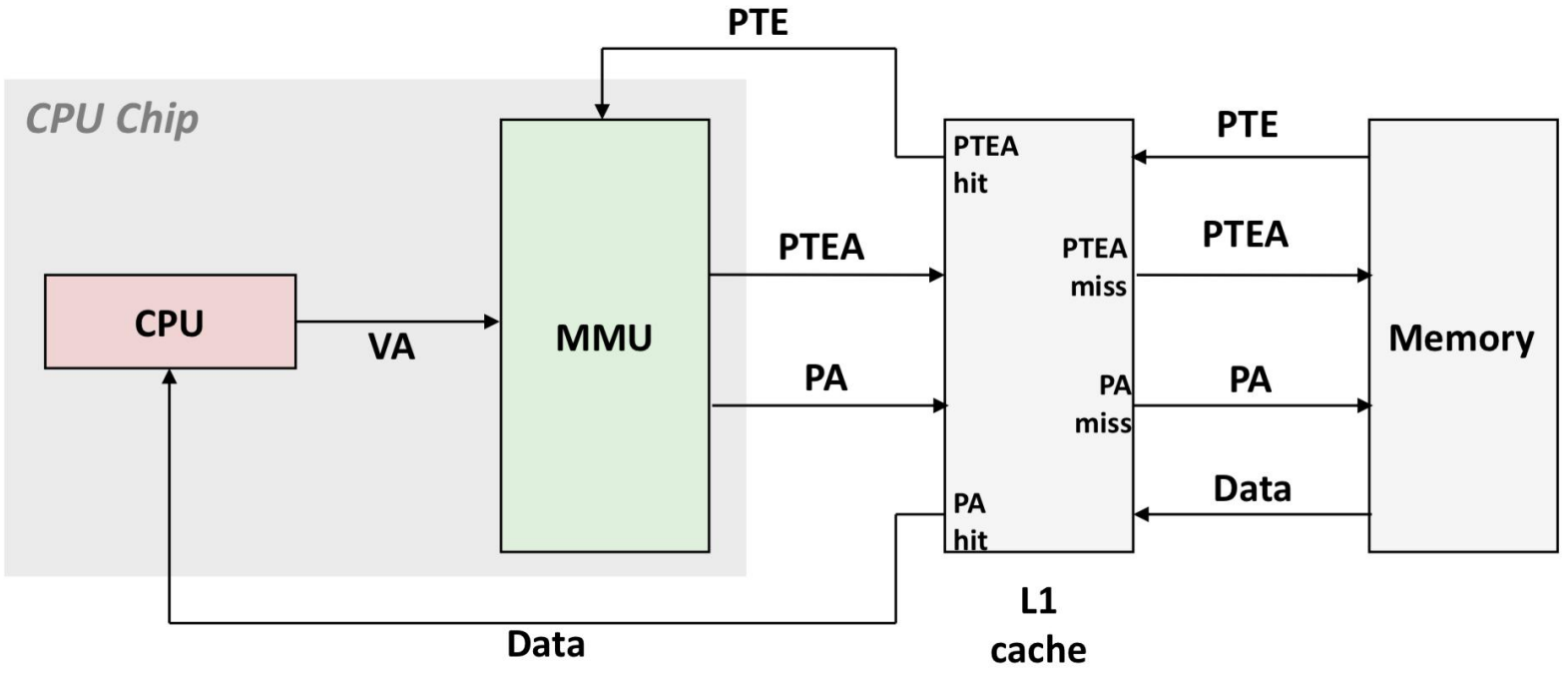
- 第1步：处理器生成一个虚拟地址，并将它传送给MMU。
- 第2步：MMU生成PTE地址，并从高速缓存/主存请求得到它。
- 第3步：高速缓存/主存向MMU返回PTE。
- 第4步：MMU构造物理地址，并将它传送给高速缓存/主存。
- 第5步：高速缓存/主存返回所请求的数据字给处理器。

缺页（页不命中）



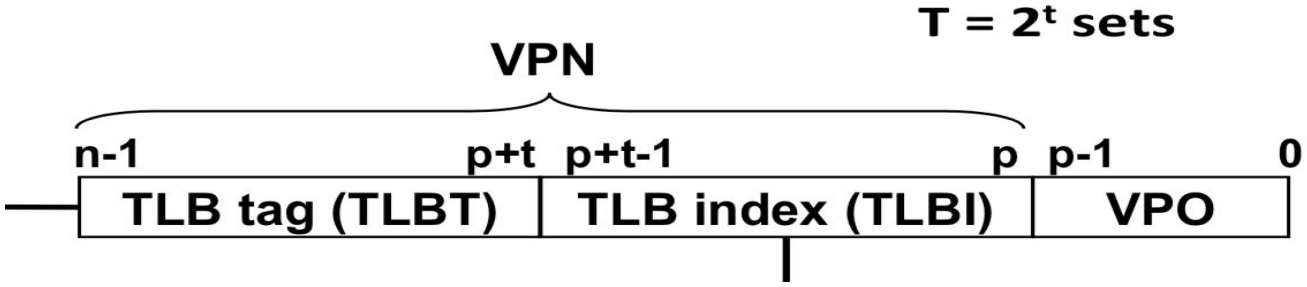
- 第1-3步，和页命中相同。
- 第4步，PTE中有效位为0，MMU触发异常，传递CPU中的控制到操作系统内核中的缺页一场处理程序。
- 第5步，缺页处理程序确定出，物理内存中的牺牲页，如果这个页面已经被修改了，则把它换出到磁盘。
- 第6步，缺页处理程序页面调入新的内面，更新内存中的PTE。
- 第7步，缺页处理程序返回到原来的进程，再次执行导致缺页的指令。此时已经页命中了，就依次执行页命中时的步骤。

结合高速缓存和虚拟内存

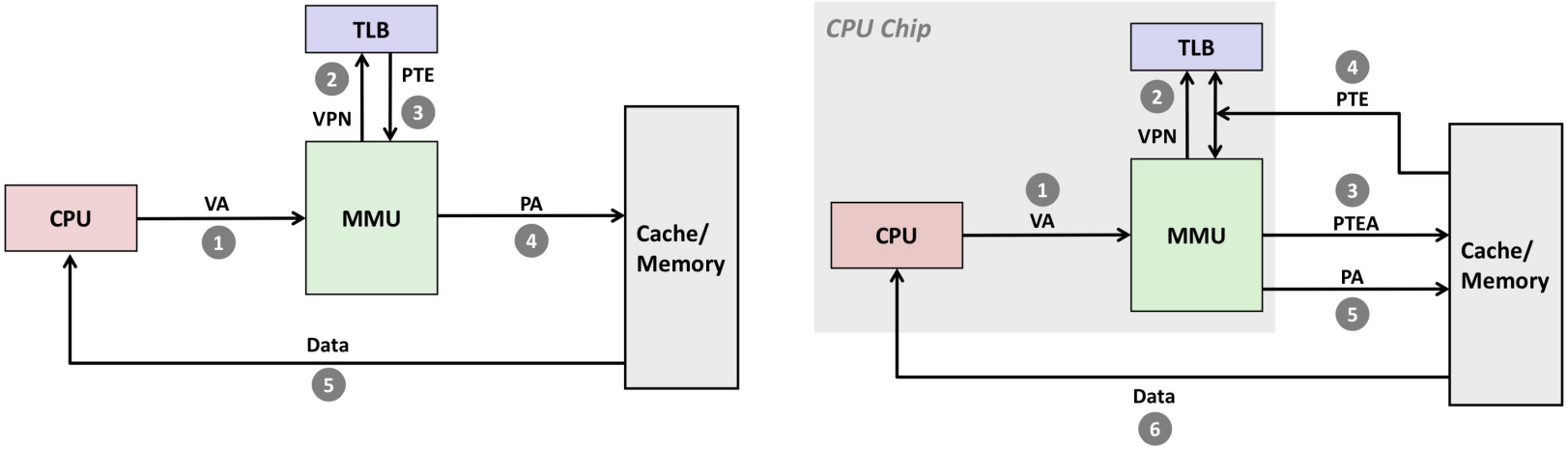


利用TLB加速地址翻译

- CPU每产生一个虚拟地址，MMU就必须查阅一个PTE，最坏情况下，会从内存中去读取一次数据，如果在L1中，开销就只有1个到2个周期。但许多系统依然想要消除这样的开销，引入一个PTE的缓存，TLB（翻译后备缓冲器）。
- TLB的每一行保存单个PTE组成的块。根据TLB索引找到组，然后根据TLB标记来找到那一行。
- 如果TLB有$T=2^t$个组，那么索引就是VPN的低t位，剩余的位就是标记。

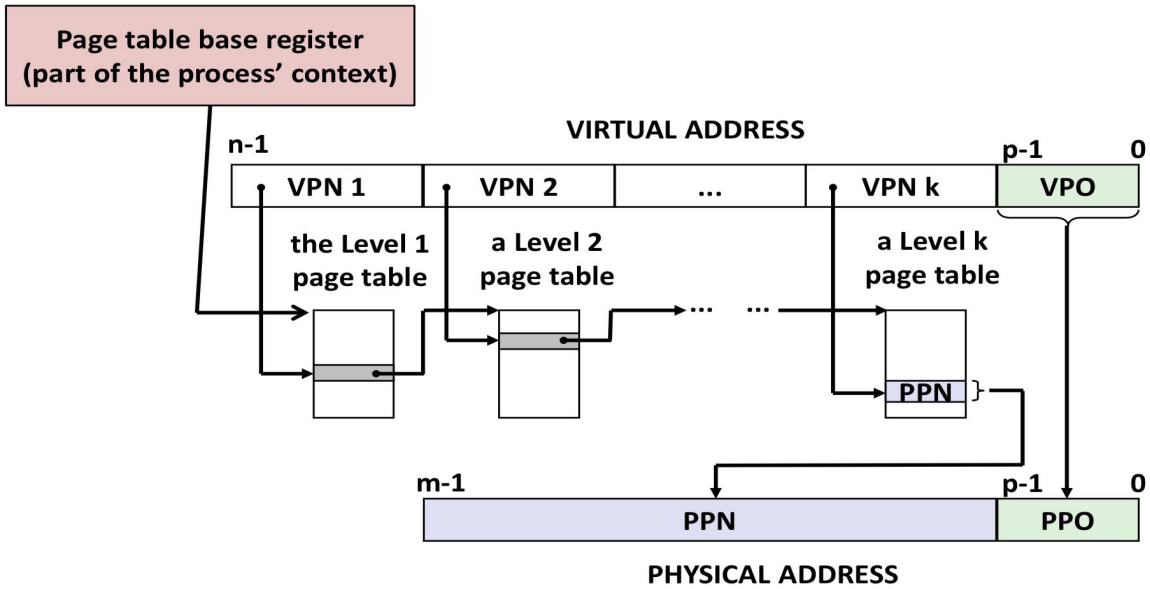


TLB的命中与不命中



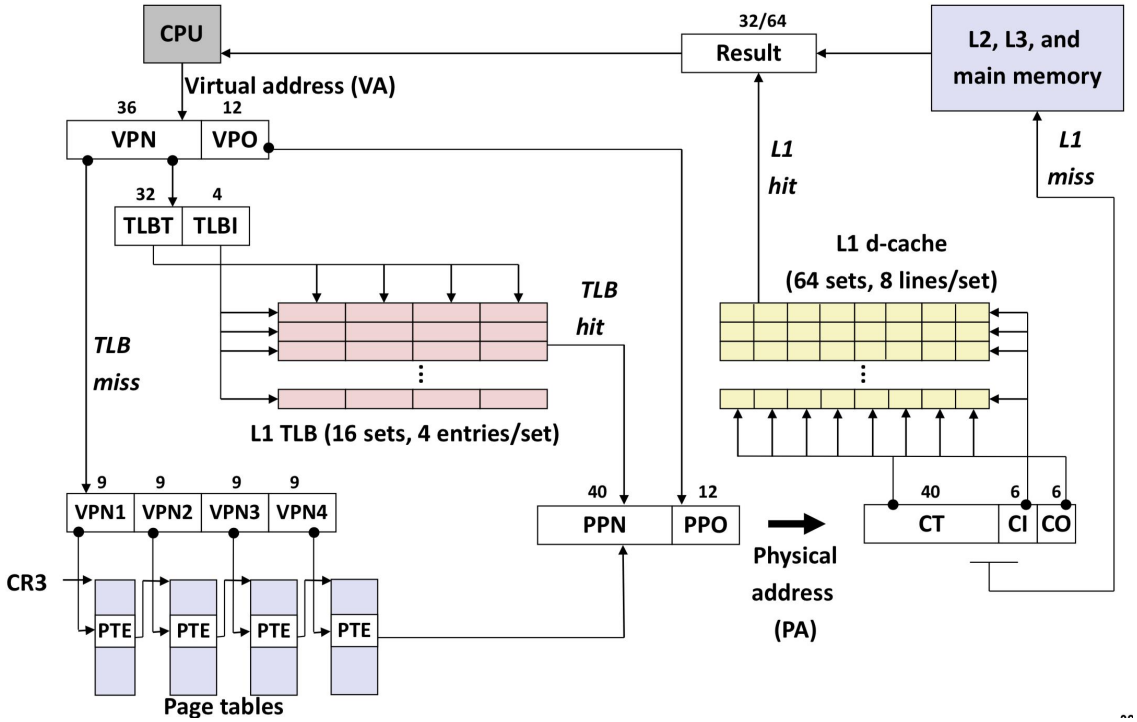
当TLB不命中时，MMU必须从L1缓存中取出对应的PTE，新取出的PTE存放在TLB中，可能会覆盖一个已经存在的条目。

多级页表



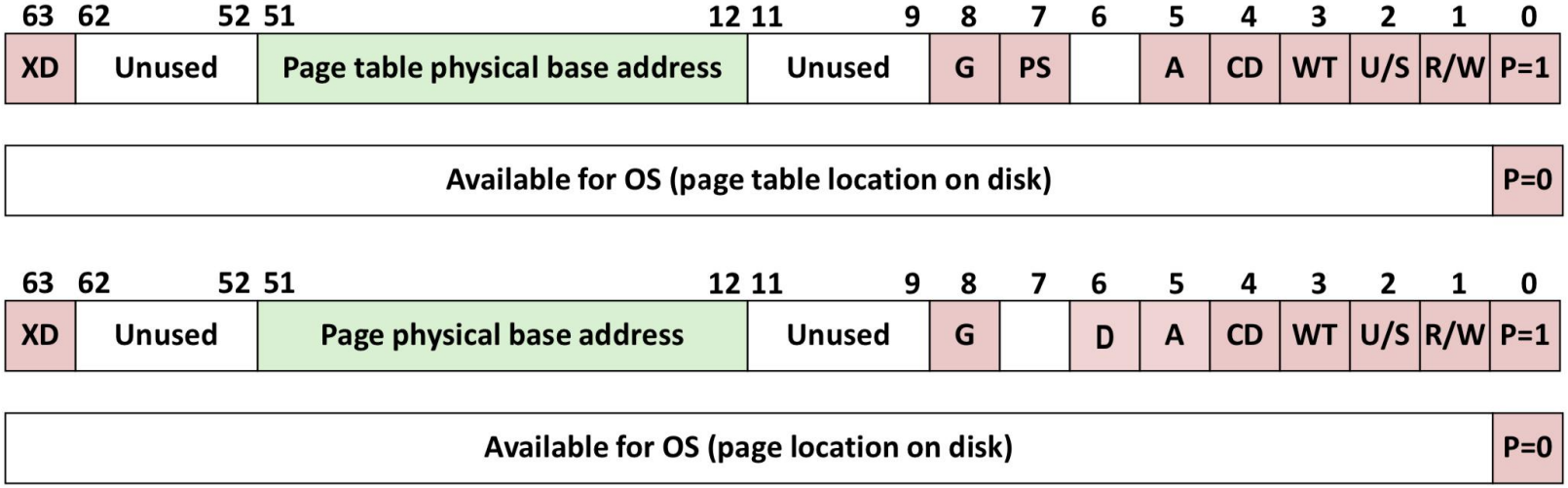
- 如果第i级页表中的一个PTE是空的，那么相应的之后几级页表不会存在。这是一种巨大的潜在节约。
- 只有一级页表需要总在主存中，虚拟内存可以在需要时创建、调入、调出二级页表。
- 一个k级页表，虚拟地址被划分为k个VPN和1个VPO，每个VPN i是到第i级页表的索引，前k-1级页表的PTE指向下一级某个页表的基址，第k级页表的PTE包含某个PPN或者磁盘块的地址。

Core i7地址翻译



- Core i7采用四级页表层次结构，每个进程有自己私有的页表层次结构。
- CR3控制寄存器指向第一级页表的起始位置。CR3的值是每个进程上下文的一部分，每次上下文切换时，CR3的值都会被恢复。

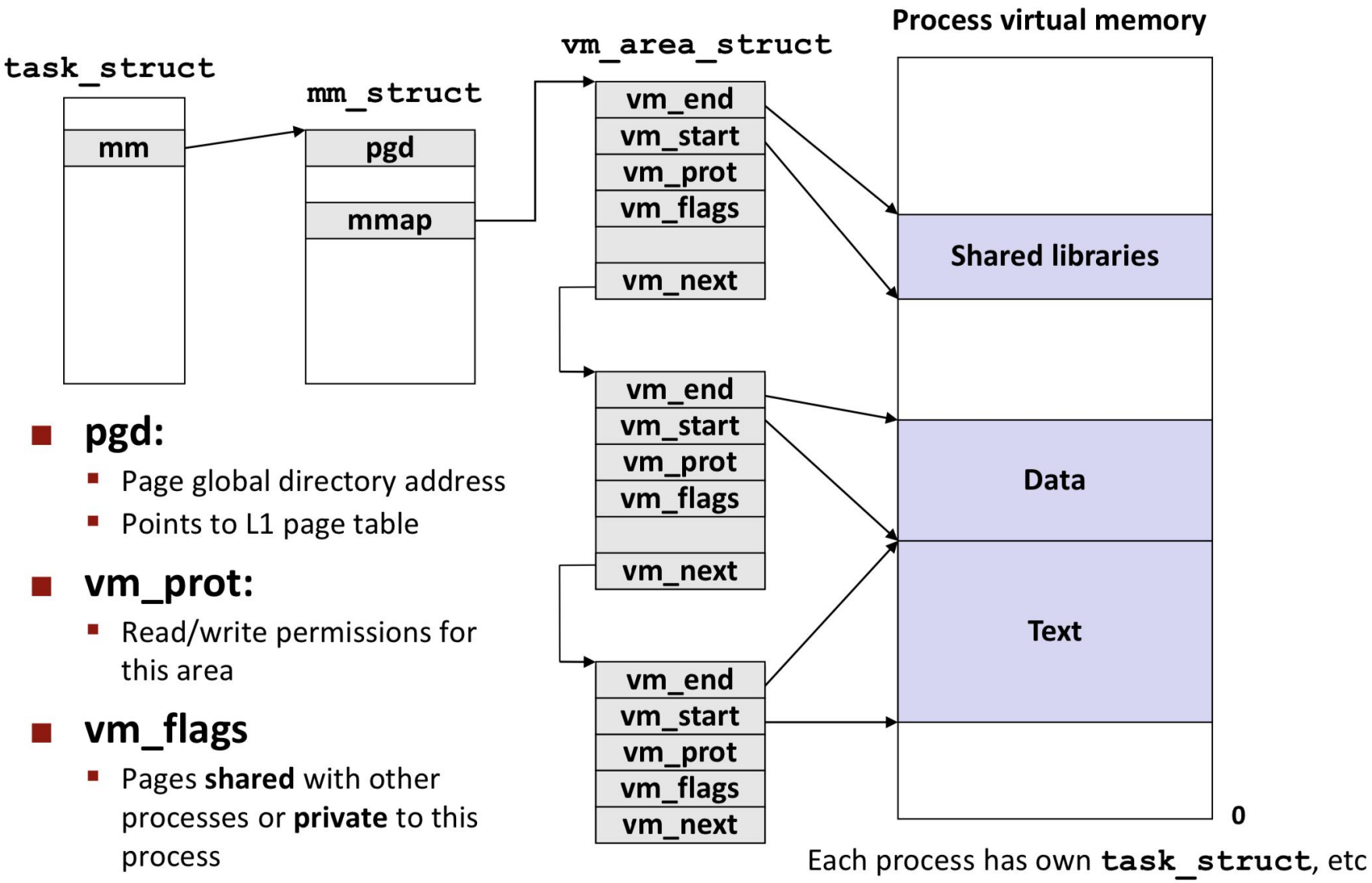
页表条目格式



- P是指子页表是否在物理内存中。
- R/W是读写访问权限。
- U/S是用户或内核模式访问权限。
- A是引用位，每访问一个页时，会设置它，内核可以用引用位来实现它的页替换算法。
- 第四级页表中还有一个D位，修改位，每次对一个页进行写之后会设置它，它告诉内核在复制替换页之前是否必须写回牺牲页。

Linux 虚拟内存系统

- Linux将虚拟内存组织称一些区域的集合。一个区域就是已经存在着的虚拟内存的连续片。
- 内核为系统中的每个进程维护一个单独的任务结构(task_struct)，任务结构中的元素包含或指向内核运行该进程所需要的所有信息。其中一个指向mm_struct，它描述了虚拟内存的当前状态。
- pgd指向第一级页表的基址，当内核运行这个进程时，就把它存放在CR3控制寄存器中。
- mmap指向一个vm_area_structs(区域结构)的链表，每个区域结构描述了一个区域。
  - vm_prot 描述这个区域的所有页的读写许可权限。
  - vm_flags 描述这个区域是与其他进程共享的，还是私有的。
  - vm_next 指向链表中下一个区域结构

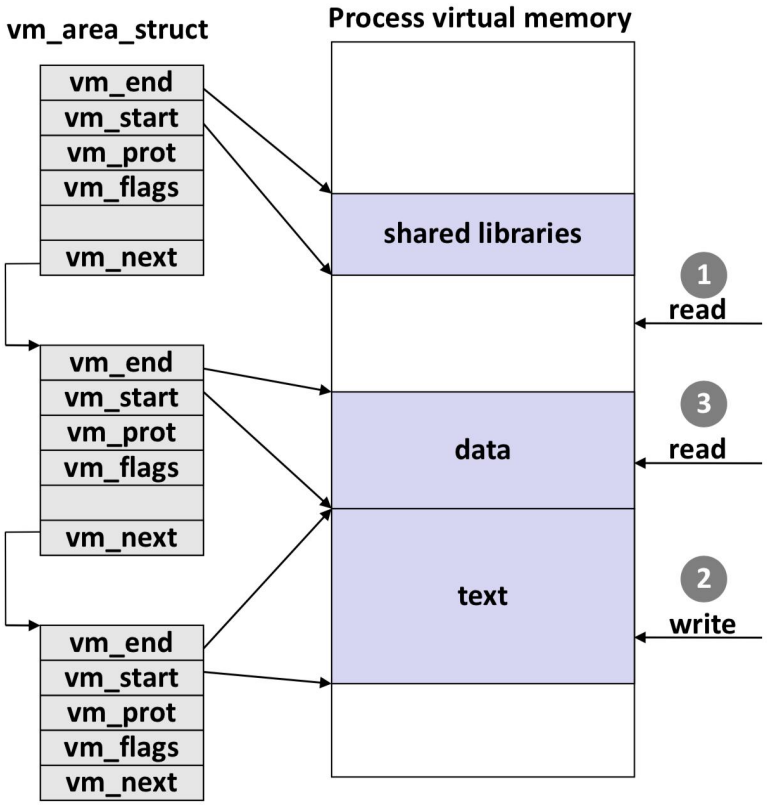


- **pgd:**
  - Page global directory address
  - Points to L1 page table
- **vm_prot:**
  - Read/write permissions for this area
- **vm_flags**
  - Pages **shared** with other processes or **private** to this process

Each process has own `task_struct`, etc

Linux 缺页异常处理

- 1 会产生段错误，访问一个不存在的页面。
- 3 会产生正常缺页问题
- 2 会有保护异常，比如写一个只读的页面



内存映射

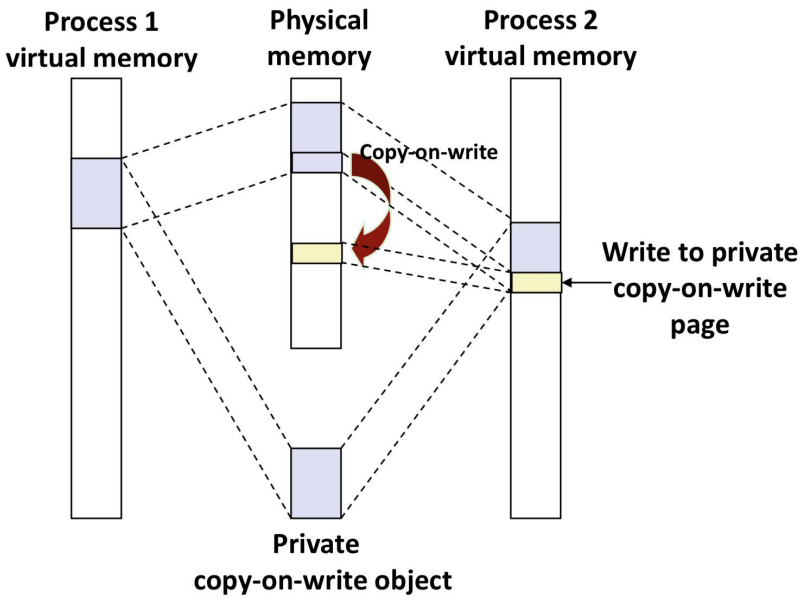
- 内存映射：Linux通过将一个虚拟内存与一个磁盘上的对象关联起来，以初始化这个虚拟内存区域的内容，这个过程叫内存映射。
- 对象类型
  - 1）普通文件 。一个区域可以映射到一个普通磁盘文件的连续部分。第一次引用之前没有实际交换进入物理内存。
  - 2）匿名文件。 匿名文件由内核创建，包含的全是二进制0，第一次引用时，内核在物理内存中找到一个合适的牺牲页面，如果该页面修改过，就换出来，然后用二进制0覆盖这个牺牲页面并更新页表。映射到匿名文件区域中的页也叫请求二进制零的页。
- 一旦虚拟页面被初始化了，它就在一个有内核维护的专门的交换文件之间换来换去。交换文件的大小限制当前运行着的进程能够分配的虚拟页面的总数。

共享对象

- 如果一个进程将一个共享对象映射到它的虚拟地址空间的一个区域内，那么这个进程对这个区域的任何写操作，对于那些也把这个共享对象映射到他们虚拟内存的其他进程而言是可见的。而且这些变化也会反映在磁盘上的原始对象中。
- 另一方面，对于一个映射到私有对象的区域做的改变，对于其他进程来说是不可见的，并且进程对这个区域所做的任何写操作都不会反应在磁盘上的对象上。

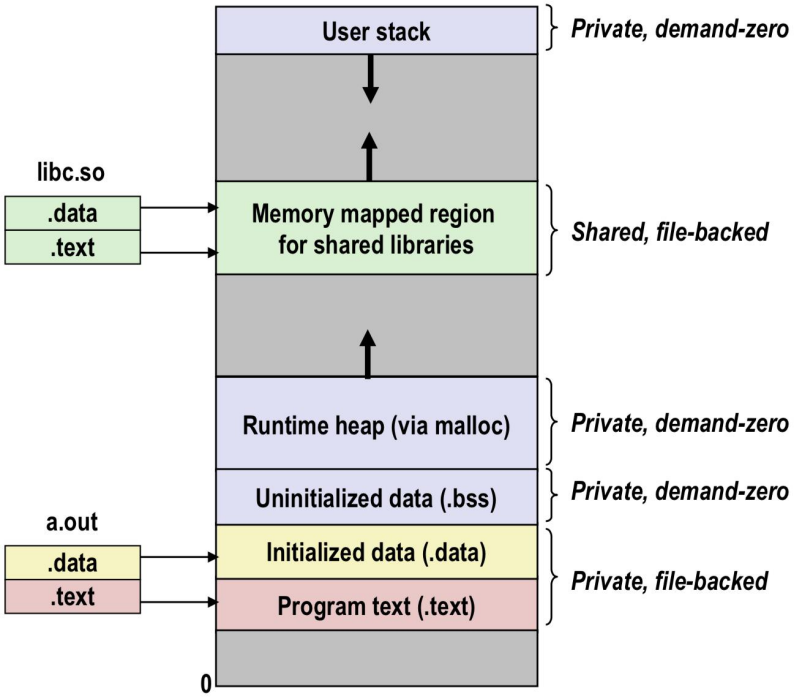
写时复制

- 对于每个映射私有对象的进程，相应私有区域的页表条目都被标记为只读，区域结构被标记为私有的写时复制。
- 当一个进程试图写私有的写时复制区域的一个页面，故障处理程序，会在物理内存中创建这个页面的一个新副本，更新页表条目指向这个新的副本，然后恢复这个页面的可写权限，然后重新执行这个写操作，就可以正常执行了。



再看fork和execve函数

- fork调用时，内核创建了当前进程的mm_struct、区域结构和页表的原样副本，然后页面都标记可读，区域结构都标记私有的写时复制。
- execve在加载并运行可执行文件时，需要做
  - 删除已存在的用户区域
  - 映射私有区域
  - 映射共享区域
  - 设置程序计数器



mmap 函数

- mmap可以创建新的虚拟内存区域，并将对象映射到这些区域中。
- 从地址start开始的一个区域，并将文件描述符fd指定对象offset偏移位置的连续length字节映射到这个新区域。
- prot包含描述访问权限位，即对应vm_prot
- flags由描述被映射对象类型的位组成。
  - MAP_ANON 就是一个匿名对象。
  - MAP_PRIVATE 就是一个私有的、写时复制的对象。
  - MAP_SHARED 就是一个共享对象。

谢谢!