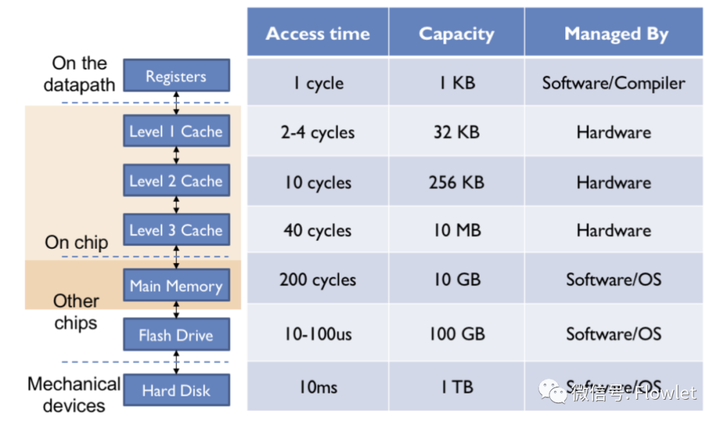
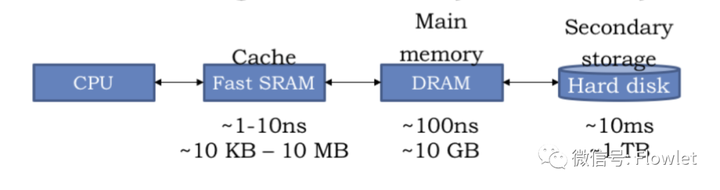
#### 1、存储器层次结构

分层存储（Tiered Storage）的理论基础是：局部性原理，其中包括时间局部性与空间局部性 2类：

* 时间局部性（Temporal locality）：如果某个信息这次被访问，那它有可能在不久的未来被多次访问（比如：循环）。
* 空间局部性（Spatial locality）：如果某个位置的信息被访问，那和它相邻的信息也很有可能被访问到（比如：数组）。





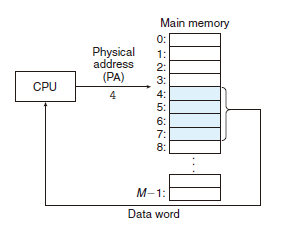
在计算机系统中：内存/DRAM 被称为主存（Primary Storage），磁盘/Disk 被称为辅助存储器/次级存储器（Secondary Storage）。

#### 2、物理内存与物理寻址

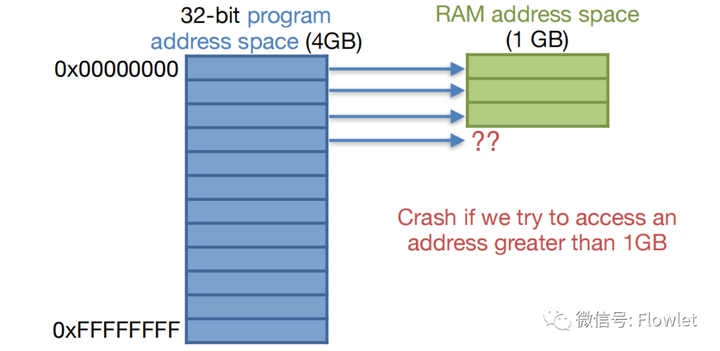
计算机系统的主存（Primary Storage）被组织成一个由 M个连续的字节（bytes）大小的单元组成的数组。每字节都有一个唯一的物理地址（Physical Address，PA）。CPU 访问内存的最自然的方式就是使用物理地址。我们把这种方式称为物理寻址（physical addressing）

##### Part 1：物理寻址的三个问题

（1）物理内存不足

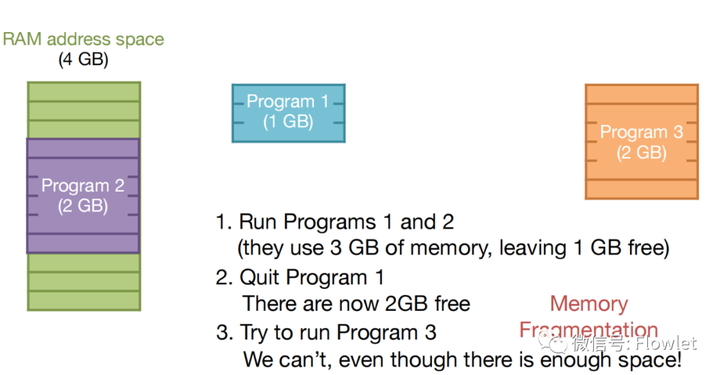


在一个 32-bit 的操作系统中，CPU 能够提供 的程序地址空间，如果该操作系统只有1GB DRAM物理内存。如果采用物理寻址，当应用程序的访问地址空间（4GB）超过物理内存的实际地址空间（1GB DRAM）范围时，就会导致程序崩溃。



（2）内存碎片化

在一个 32-bit 的操作系统中，如果该操作系统同时拥有4 GB的DRAM，首先运行程序1（占用1GB连续DRAM）与程序2（占用2GB连续DRAM），此时程序3无法正常运行（占用2GB连续DRAM）。此时若程序1退出，剩余2GB DRAM，程序3依然无法运行。因为剩余的2GB DRAM内存空间不是连续的。



（3）内存安全问题

程序都可以访问 4GB 的物理内存地址空间，如果多个程序访问相同的物理地址，会导致数据污染或崩溃。

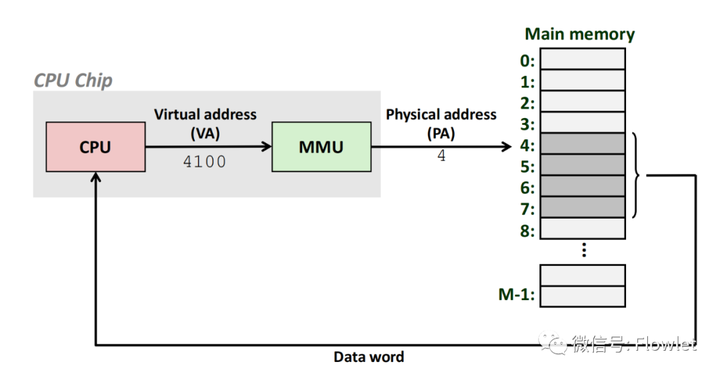
#### 3、虚拟内存

虚拟内存是物理内存和进程地址空间之间的中间层，它为进程隐藏了物理内存这一概念。虚拟内存提供了3个重要的能力：

* 简化内存管理：为每个进程提供了一致的地址空间（在一个32-bit的操作系统中，每个进程都拥有的程序地址空间），从而简化了链接、加载、内存共享等过程；
* 高效使用内存：将进程的地址空间存储在磁盘上，将主存看成是存储在磁盘上的地址空间的高速缓存，主存中保存热的数据，根据需要在磁盘和主存之间传送数据；
* 内存保护：保护每个进程的地址空间不被其他进程破坏。

##### Part 1：虚拟寻址

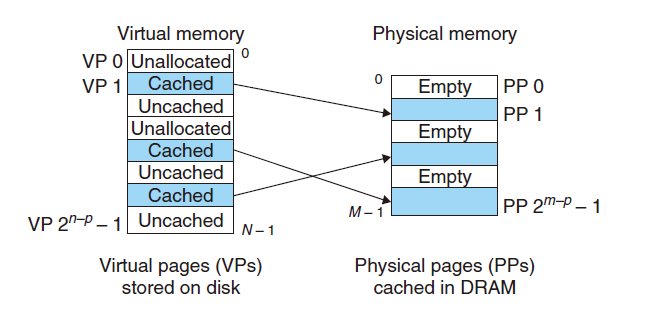
CPU 通过生成一个虚拟地址（Virtual Address，VA）来访问主存，这个虚拟地址在被送到主存之前需要先转换成适当的物理地址。将一个虚拟地址转换为物理地址的任务叫做地址翻译（address translation）。



这一过程需要CPU硬件和操作系统的合作，CPU芯片上叫做内存管理单元（memory Management Unit，MMU），利用操作系统管理的存放在主存中的查询表（Page Table）来动态的将虚拟地址翻译为物理地址

##### Part 2：虚拟内存使用主存作为缓存

虚拟内存被组织为一个由存放在磁盘上的N个连续的字节（Bytes）大小的单元组成的数组，每字节都有一个唯一的虚拟地址。磁盘上的数组的内容被缓存在主存中，根据需要在磁盘和主存之间传送数据。

通过查看DRAM时序，我们发现检索连续字节块的时间比访问单独一个字节的时间要短很多。对于磁盘来说，从一个扇区读取第一个字节的时间开销，要比读这个扇区内的连续字节要慢 100,000倍。因此虚拟内存想要获取比较高的性能，磁盘（较低层级）上的数据被分割成块，这些块作为磁盘和主存（较高层级）之间的传输单元。

虚拟内存系统通过将将虚拟内存分割为称为虚拟页（Virtual Page，VP）的大小固定的块来处理这个问题。每个虚拟页的大小为字节（虚拟页一般大小为4KB ~ 2MB）。类似的，物理内存被分割成物理页（Physical Page，PP），大小为 P 字节（物理页也被称为页帧，page frame）。

任意时刻，虚拟页面的集合分成3个不相交的子集：

* 未分配的：虚拟内存还未分配（或者创建）的页；
* 缓存的：已缓存在 DRAM 内存中的已分配页；
* 未缓存的：未缓存在 DRAM 内存中的已分配页；

上图中包含8个虚拟页的虚拟内存系统，VP0、VP3未分配，VP1、VP4、VP6分配并缓存在DRAM内存中，VP2、VP5、VP7被分配了，但是没有缓存在DRAM内存中。

##### Part 3：页表

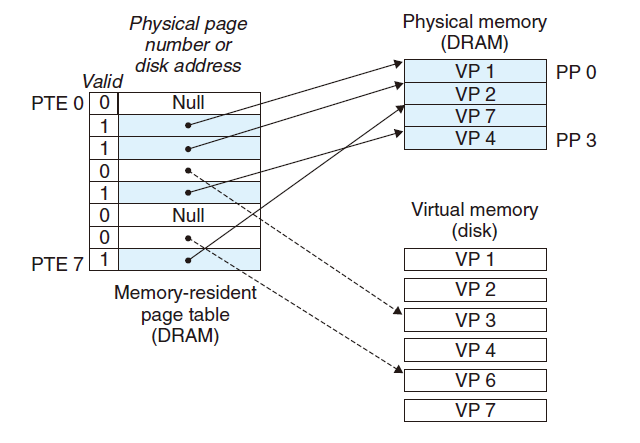
每一个进程都有一份页表（Page Table），作为其上下文的一部分。页表由一系列页表条目（PTE，Page Table Entry）组成，每个页表条目都包含着虚拟页和物理页的映射关系。PTE 由一个有效位（valid bit，表明该虚拟页是否被缓存在DRAM中）和一个n位地址字段组成。

页表（page table）的数据结构存放在物理内存（DRAM）中，操作系统负责维护页表结构，每次MMU（内存管理单元）中的地址翻译硬件将一个虚拟地址转换成物理地址时，都会读取页表。

MMU通过页表来确定一个虚拟页是否缓存在DRAM中：

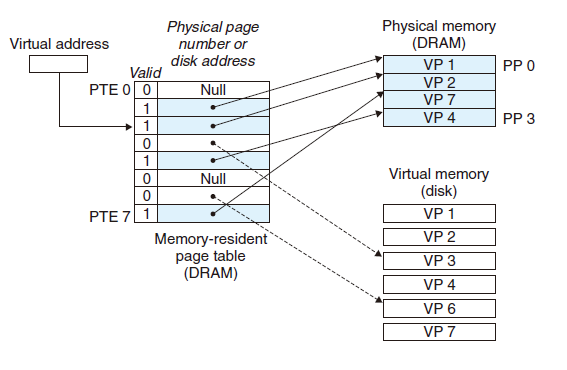
* 如果是（有效位为1），则该条目指向该虚拟页所存放在物理页的位置；
* 如果不是（有效位0），则该条目指向该虚拟页所存放在磁盘的位置，在物理内存（DRAM）中选择一个牺牲页，并将虚拟页从磁盘复制到DRAM中，替换这个牺牲页。

下图为包含8个虚拟页，4个物理页的系统的页表结构。



##### Part 5：页命中

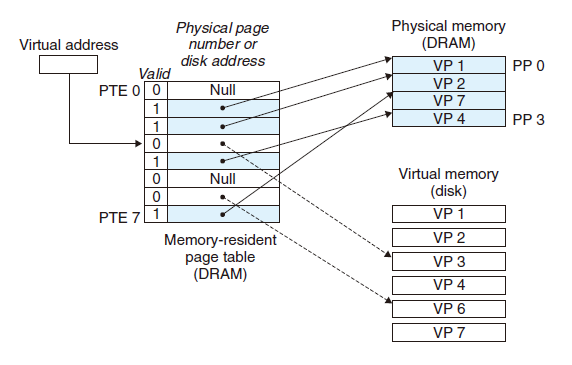
页命中指的是当 MMU需要根据虚拟地址输出物理地址时，这个地址所在的页已经被装载到物理内存中了（即对应的 PTE 的有效为 1）。

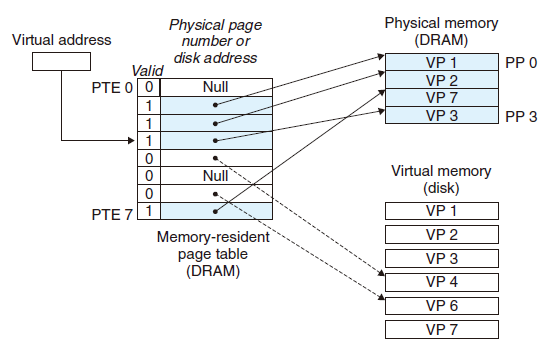


上图中，当MMU访问的虚拟地址对应到页表中VP 2时，地址翻译硬件发现该地址在页表当中有效位为1，即被缓存在DRAM当中（称为页命中），则使用页表当中PTE所对应的物理内存地址，来访问数据。

##### Part 5：缺页

下图中，当MMU访问的虚拟地址对应到页表中VP 3时，地址翻译硬件发现该地址在页表当中有效位为0，即未被缓存在DRAM当中，称为缺页（Page Fault），触发一个缺页异常。





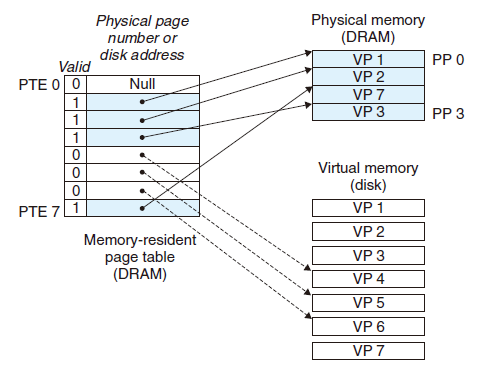
缺页异常的处理程序被启动，该程序会选择一个牺牲页，若是该牺牲页被标记为已经更改过，则内核会将其复制回磁盘，若是未更改过，调整牺牲页在页表中所对应的PTE。接着，内核从磁盘(虚拟内存)当中将内容复制到牺牲页(物理内存)上，再次更新其PTE，随后返回。

当缺页异常处理程序返回时，原进程会重新启动导致缺页异常的指令，该指令会将导致缺页的虚拟地址重发送到地址翻译硬件，这时就会进行页命中的相关流程了。

上图中，触发缺页异常后，缺页异常处理程序选择VP 4作为牺牲页，并从磁盘上用VP 3的副本取代它。在缺页异常处理程序重新启动导致缺页的指令之后，该指令将从内存中正常地读取字，而不会再产生缺页异常。

##### Part 6：分配新页

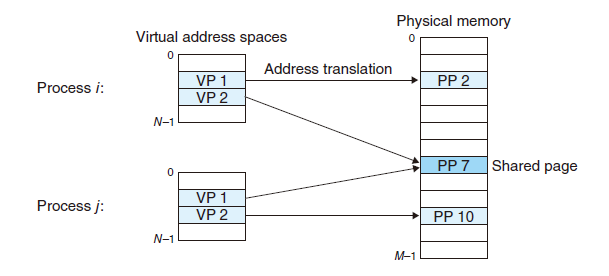
如下图，内核在磁盘上分配VP 5，并将PTE 5指向这个新的位置。



##### Part 7：虚拟内存作为内存管理的工具

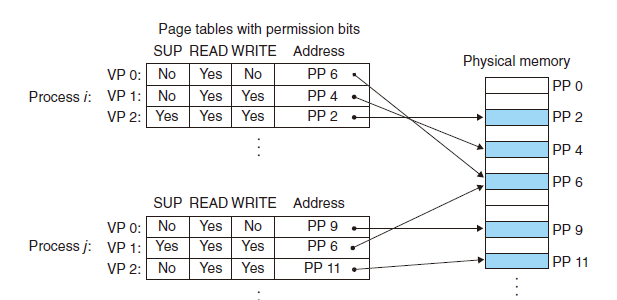
操作系统为每个进程提供了一个独立的页表，也就是每个进程独占一个独立的虚拟地址空间。这样做的好处：

* 简化共享内存：操作系统通过将不同进程中适当的虚拟页面映射到相同的物理页面，从而安排多个进程共享下共同代码的副本。
* 简化链接：独立的地址空间允许每个进程的内存布局使用相同的基本格式，而不管代码和数据实际存放在物理内存的何处。链接器可以假设每个程序都加载到相同的位置，然后它可以重定位这些引用。
* 简化加载：execve 查看 ELF文件，它知道文件中的代码和数据段有多大，它从固定的地址为代码和数据分配虚拟内存。
* 简化内存分配：虚拟内存为向用户进程提供一个简单的分配额外内存的机制。当一个运行在用户进程中的程序要求额外的堆空间时（如调用malloc），操作系统分配一个适当数字（eg：k）个连续的虚拟内存页面，并且将他们映射到物理内存中任意位置的k个任意的物理页面。



##### Part 8：虚拟内存作为内存保护的工具

虚拟内存通过在PTE上添加一些额外的许可位来控制对一个虚拟页面的访问权限，通过虚拟内存可以提供页面级的内存保护。SUP：监管者模式（Supervisor Mode）

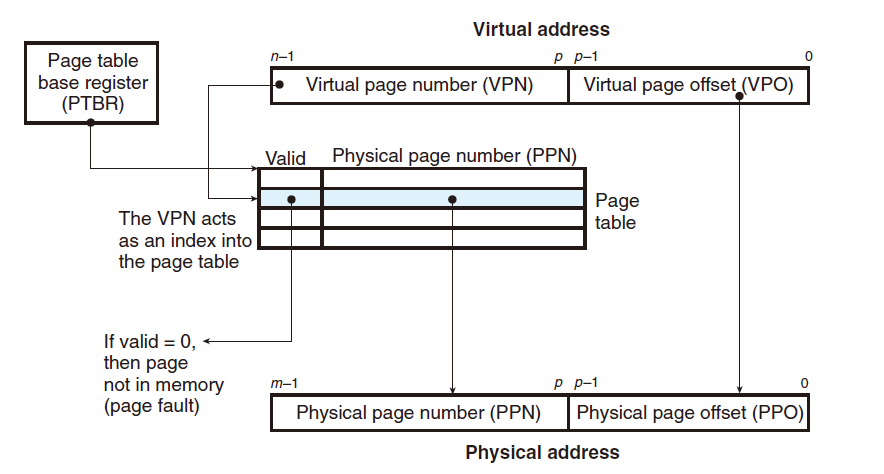


#### 4、地址翻译

##### Part 1：地址翻译术语

##### Part 2：使用页表的地址翻译

在CPU中地址翻译由一个叫做MMU（Memory Management Unit，内存管理单元）的硬件完成。MMU接收一个虚拟地址，并且输出一个物理地址。如果这个虚拟地址在物理内存中存在，那么就叫做页命中。如果这个虚拟地址在物理内存中不存在，那么MMU将产生一个缺页错误。



n位的虚拟地址包括两个部分：一个 p位的虚拟页面偏移（Virtual Page Offset，VPO），和一个n-p位的虚拟页号（Virtual Page Number，VPN），MMU利用VPN来选择适当的PTE。

因为物理和虚拟页面都是P字节的，所以物理页面偏移（Physical Page Offset，PPO）和 虚拟页面偏移（Virtual Page Offset，VPO）是相同的，因此将PTE中的物理页号（Physical Page Number， PPN）与VPO串联起来，就得到了相应的物理地址。

（1）页命中时地址翻译

1、处理器生成一个虚拟地址，并把它传送给MMU。

2、MMU生成根据虚拟地址生成VPN，然后请求高速缓存/主存，获取PTE的数据。

3、高速缓存/主存向MMU返回PTE的数据。

4、从PTE获取对应的物理页号PPN。用物理页的基址加上页偏移PPO（假设页大小为 4KB，那么页偏移就是虚拟地址的低12位，物理页的页偏移和虚拟页的页偏移相同），获取对应的物理地址。

5、主存/高速缓存将数据返回给CPU。

（2）缺页时的地址翻译

1、处理器生成一个虚拟地址，并把它传送给MMU。

2、MMU根据虚拟地址生成VPN，然后请求高速缓存/主存，获取PTE的数据。

3、高速缓存/主存向MMU返回PTE的数据。

4、由于判断出PTE的有效位是0，所以CPU将触发一次异常，将控制权转移给内核中的缺页异常处理程序。

5、缺页异常处理程序确定出物理内存中的牺牲页，如果这个页面被修改过了（D标志位为 1），那么将牺牲页换出到磁盘。

6、缺页处理程序从磁盘中调入新的页面到主存中，并且更新PTE。

7、缺页处理程序将控制权返回给原来的进程，再次执行导致缺页的指令。再次执行后，就会产生页命中时的情况了。

##### Part 3：地址翻译加速

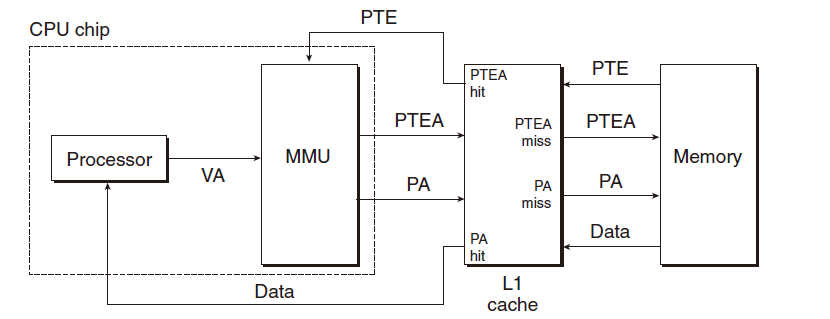
（1）将 Cache 与虚拟内存整合在一起

VA：虚拟地址

PA：物理地址

PTE：页表条目

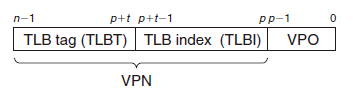
PTEA：页表条目地址

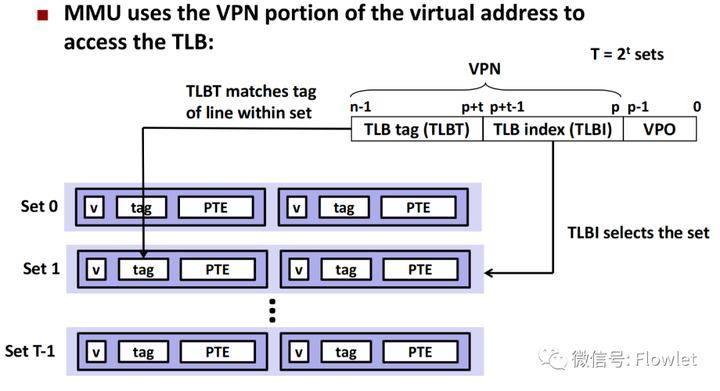
从页命中的流程图中可以看出，CPU每次需要请求一个虚拟地址，MMU就需要从内存/Cache中获取PTE，然后再根据PTE的内容去从物理内存中加载数据。如果在PTE在Cache 中未命中，就需要从内存中获取PTE。这部分由于Cache Miss造成的开销是巨大的。如果我们将PTE存储到一个专门的L1 Cache中，那么将会减少这部分开销，这个专门存放PTE 的L1 Cache就是TLB（Translation Lookaside Buffer，翻译后备缓冲器）。

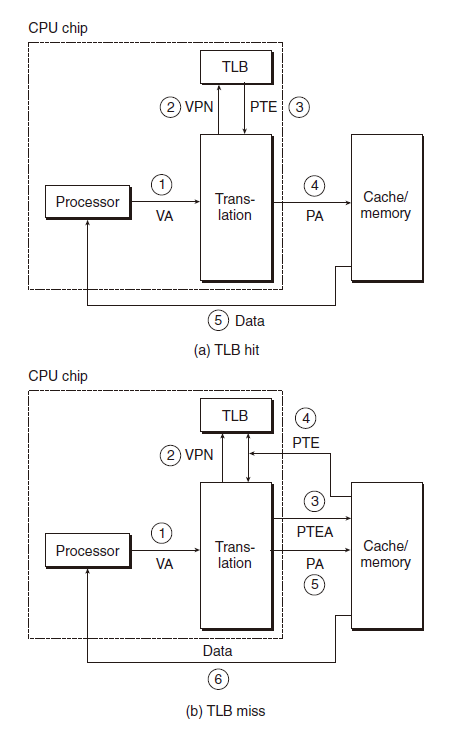
（2）利用TLB加速地址翻译

许多 MMU 包含一个名叫TLB（Translation Lookaside Buffer，翻译后备缓冲器）的 Cache。

TLB 将虚拟内存的VPN视为由组索引和行匹配组成，索引部分（TLBI）用来定位 TLB 中的缓存数据项，标记部分（TLBT）用来校验存储的数据项是否为指定的VPN对应的数据。







如果TLB命中了，那么所有的地址翻译步骤都是在MMU中执行的，所以非常快。

* TLB命中

1、CPU生成1个虚拟地址；

2、MMU向 TLB请求PTE；

3、TLB返回PTE到MMU；

4、MMU将这个虚拟地址翻译成一个物理地址，并且把它发送到高速缓存/主存；

5、高速缓存将所请求的数据字返回给CPU；

* TLB未命中

1、CPU生成1个虚拟地址；

2、MMU向TLB请求PTE，TLB未命中；

3、MMU从高速缓存/内存中获取相应的PTE；

4、MMU将新取出来的PTE放在TLB中；

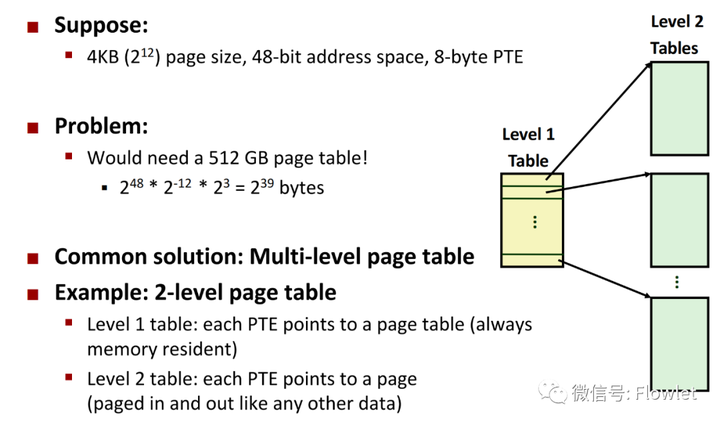
5、MMU将通过PTE这个虚拟地址翻译成一个物理地址，并且把它发送到高速缓存/主存；

6、高速缓存将所请求的数据字返回给CPU；

理解TLB需要注意的是，因为不同进程的页表内容是不同的，因此在进程上下文切换时，会重置TLB。

##### Part 4：多级页表

PTE的数量由虚拟地址空间的大小和页大小决定。也就是：X=N/P。那如果我们有一个 32 位的物理地址空间、4KB的页面和一个4字节的PTE。即使程序只使用了一小部分虚拟地址空间，也总是需要一个4MB（）的页表常驻主存。对于64位的系统来说，情况将变得更加复杂。因此使用层次结构的页表来压缩页表。



（1）二级页表

下图展示了一个两级页表的层次结构。二级页表中的每个PTE项都负责一个4KB页面，而一级页表中的每个PTE负责1024个二级页表项。

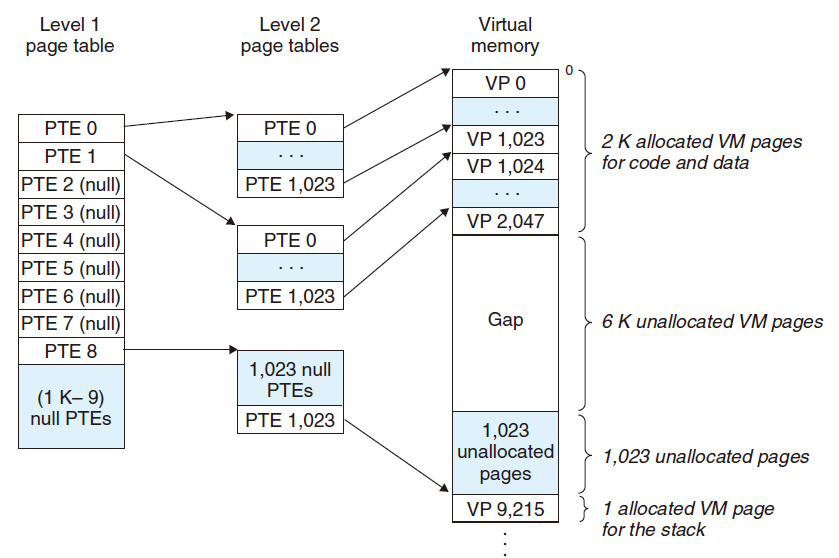
只有一级页表才需要存放在主存/TLB，虚拟内存系统可以在需要时创建、调入或调出二级页表；且常用的二级页表才需要缓存在主存/TLB。

如果一个一级页表是空的，那么二级页表也不会存在。这是一个很大的节约，因为一个典型程序4G的虚拟地址空间的大部分都是未分配的。如果页表层级太多，则增加缓存未命中的概率，一般层级是4。

每一个一级页表中的PTE映射到了一个4MB的虚拟地址空间块，即二级页表所在（，每一个PTE 映射4KB的地址空间）。每一个块包含了1024个连续的页。例如，L1中的PTE 0映射到了第0个块，PTE 1映射到了第1个块。如果物理地址空间是4GB，那么1024个PTE就足够映射整个空间（）。

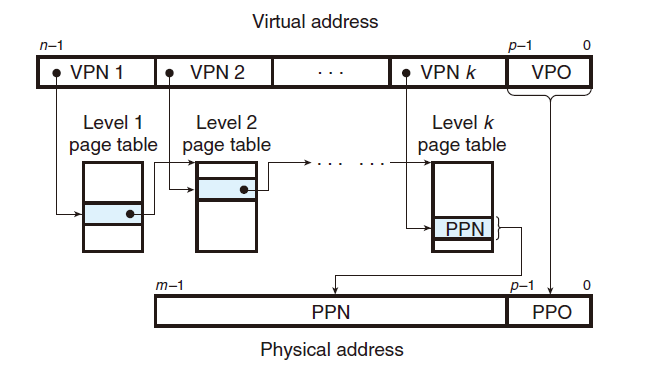
如果第个块中的页都未分配，那么一级页表中的PTE 就为空。例如，图中块2~7都未分配。但是，只要块中有一个页被分配，那么一级页表中的PTE 就指向了这个块（二级页表）的基地址。例如，图中块0、1、8中的所有或部分页被分配，所以它们的一级页表中的PTE指向了对应的二级页表。

如同单页表一样，二级页表中的每一个PTE映射到了一个4KB的虚拟内存空间。注意到所有PTE都是4Byte，所以每一个一级页表和二级页表的大小都是4KB，和一个页的大小相同，这带来了很大的便利。



（2）K级页表

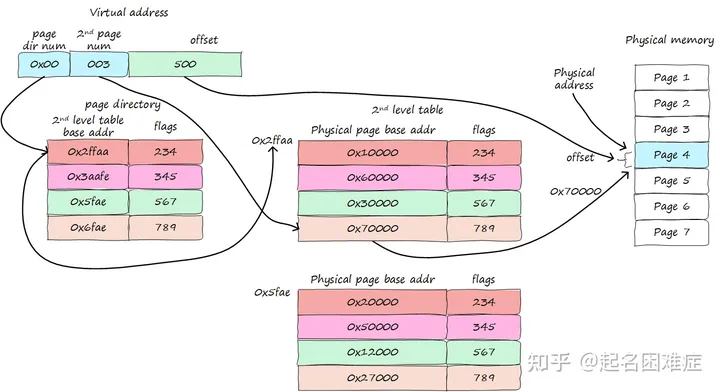
下图展示的是一个K级层次页表的结构图，其实就是将VPN部分划分为多个段，每个段都代表某一级页表。而每一级中的PTE的 Base addr为下一级提供入口地址，最后一级的 Base addr则表示最终物理地址的PPN。



（3）多级页表优化实例

在两级页表中，虚拟地址的页号部分被分为页目录号（Page Directory Number）和二级页号，页表也被分成了页目录（Page Directory）和二级页表。每一个页目录项（Page Directory Entry, PDE）包含页表物理基地址和各种状态标志位。其中的页表物理基地址指向了对应的二级页表首地址。在地址转换时，首先根据页目录号找到对应的页目录项，得到其中的页表物理基地址，然后根据页表物理基地址找打二级页表，然后使用二级页表号找到对应的页表项，然后将页表项中的物理页基地址和页内偏移相加得到物理地址。

结合下图，更加形象地理解一下：虚拟地址为0x00003500，其中高10位被作为页目录号，中间10位被作为二级页号，低12位被作为页内偏移。根据页目录号，我们可以索引到页目录中的第0x00个PDE。该PDE中存储的二级页表物理机地址时0x2ffaa，然后根据此地址，我们找到对应的二级页表。然后，根据页号，我们可以索引到二级页表中的第003个页表项。该页表项中存储的物理页基地址为0x70000，该基地址对应到物理内存中的Page 4。在Page 4中，根据页内偏移500可以定位到具体物理地址。



二级页表能节省多少空间呢？还是以32位系统为例，虚拟地址空间是4GB，我们只需要为4GB空间分配页目录。每个PDE大小32位，则整个页目录需要4\*2^10=4KB内存。然后，假设程序实际只使用了100M虚拟内存，那么我们只需要为这100M空间分配二级页表，大概为100MB/4KB\*4B=100KB。总共，需要100KB+4KB=104KB内存，相比于单级页表的4MB，减少了很多。