Virtual Memory: Concepts
虚拟内存:概念

100076202： 计算机系统导论

任课教师：

计卫星　　宿红毅　　张艳

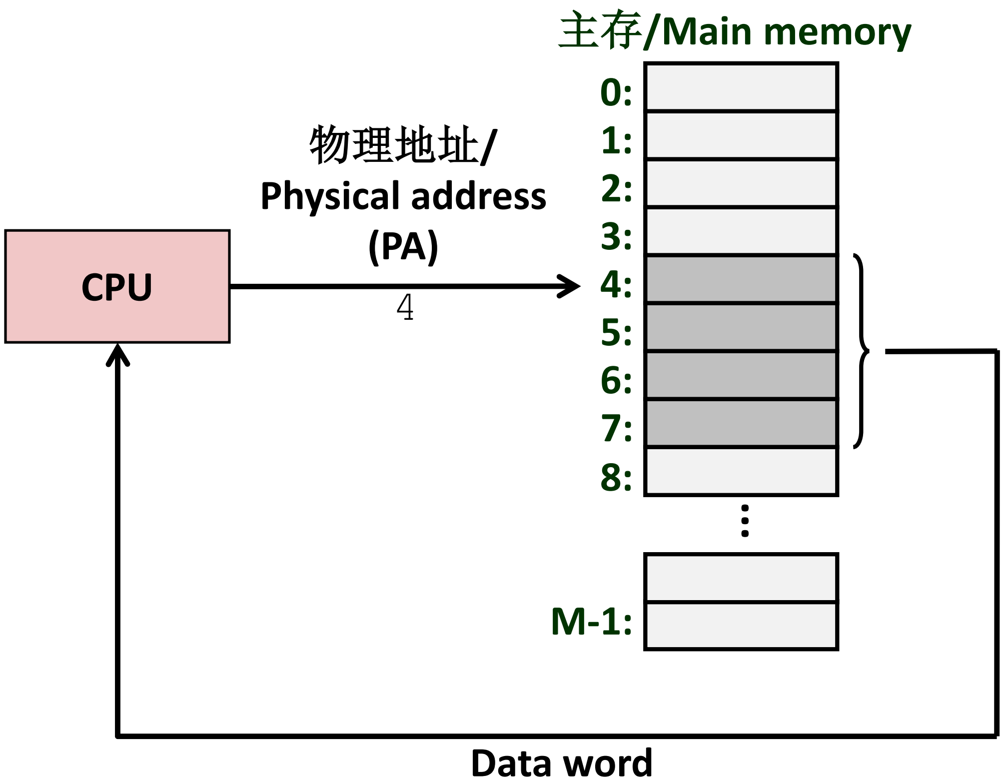
原作者：

Randal E. Bryant and David R. O’Hallaron

内容提纲/Today

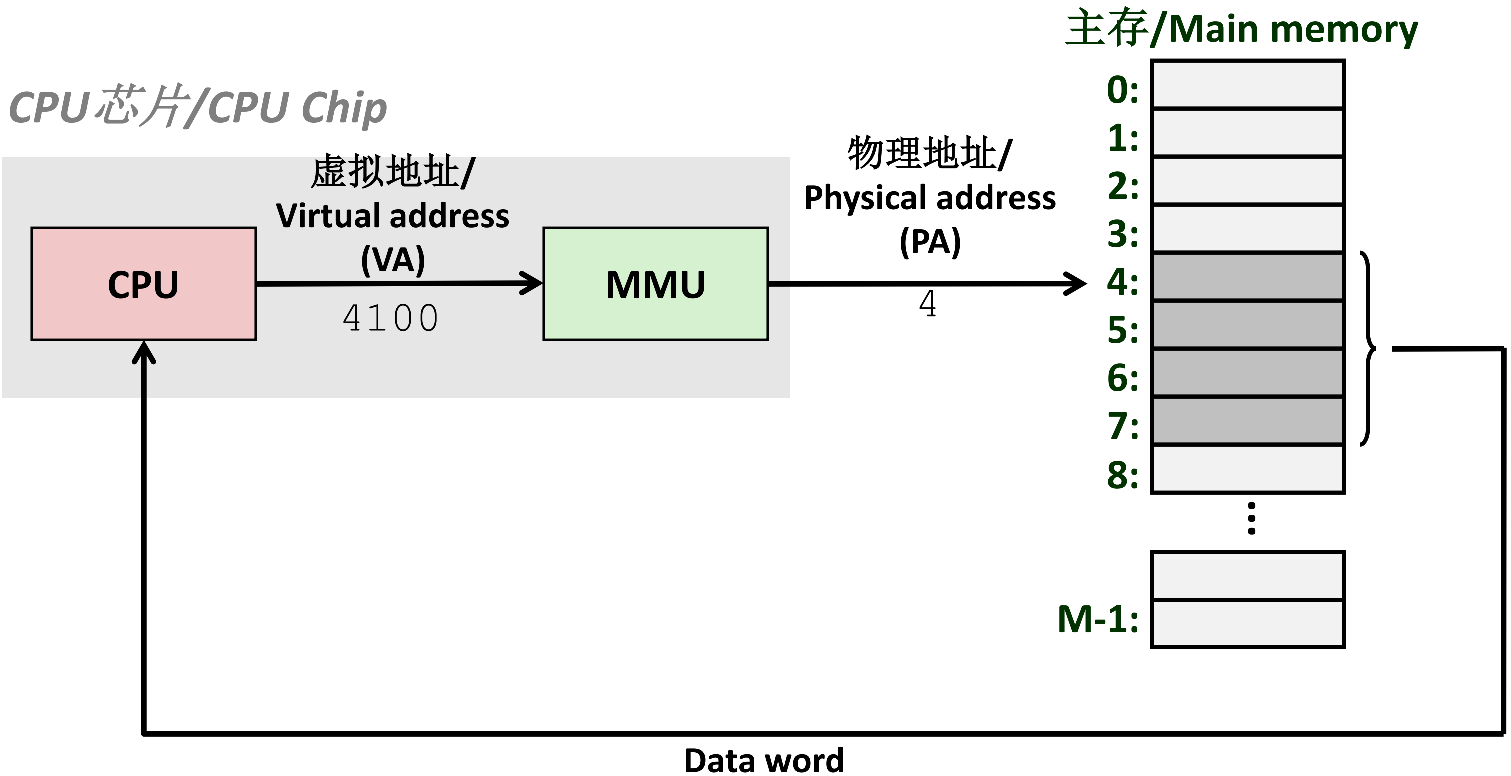
- **地址空间/Address spaces**
- 基于虚拟内存的缓存机制/VM as a tool for caching
- 基于虚拟内存的内存管理机制/VM as a tool for memory management
- 基于虚拟内存的内存保护机制/VM as a tool for memory protection
- 地址翻译/Address translation

使用物理寻址的系统/A System Using Physical Addressing



- 通常在车、电梯、数字相框等设备中简单系统的嵌入式微控制器使用/Used in “simple” systems like embedded microcontrollers in devices like cars, elevators, and digital picture frames

使用虚拟寻址的系统/A System Using Virtual Addressing



- 在服务器、笔记本和智能手机中使用/Used in all modern servers, laptops, and smart phones
- 计算机科学中的一个重要技术/One of the great ideas in computer science

地址空间/Address Spaces

- **线性地址空间/Linear address space:** 连续非负整型地址的有序集合/Ordered set of contiguous non-negative integer addresses:

  {0, 1, 2, 3 … }

- **虚拟地址空间/Virtual address space:** $N = 2^n$ 虚拟地址集合/Set of $N = 2^n$ virtual addresses

  {0, 1, 2, 3, …, N-1}

- **物理地址空间/Physical address space:** $M = 2^m$ 物理地址集合/Set of $M = 2^m$ physical addresses

  {0, 1, 2, 3, …, M-1}

为什么需要虚拟内存/Why Virtual Memory (VM)?

- **更高效地使用主存/Uses main memory efficiently**
  - 使用DRAM作为一部分虚拟地址空间的缓存/Use DRAM as a cache for parts of a virtual address space

- **简化内存管理/Simplifies memory management**
  - 每个进程都用同样的统一线性地址空间/Each process gets the same uniform linear address space

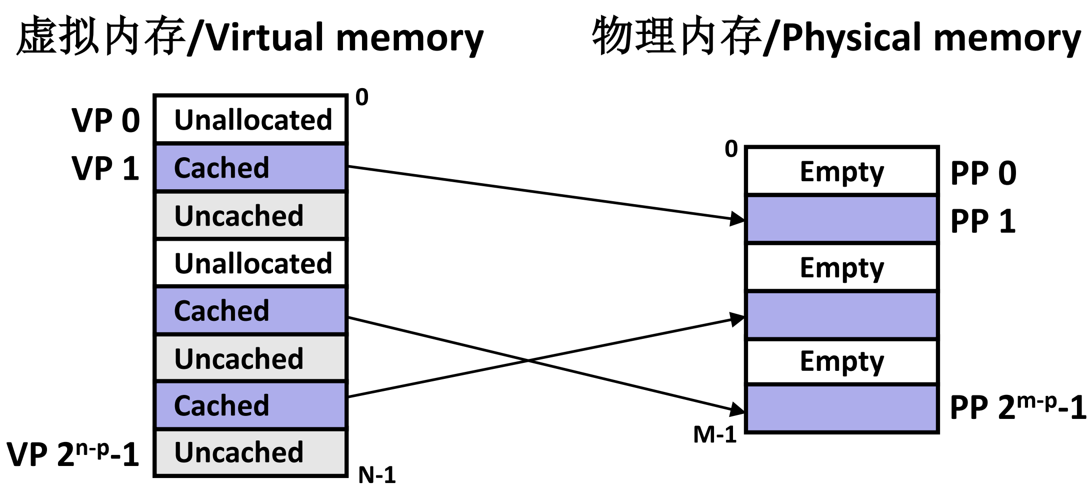
- **隔离的地址空间/Isolates address spaces**
  - 一个进程内存不会与另一个进程冲突/One process can’t interfere with another’s memory
  - 每个程序不能访问特权内核信息和代码/User program cannot access privileged kernel information and code

内容提纲/Today

- 地址空间/Address spaces
- 基于虚拟内存的缓存机制/VM as a tool for caching
- 基于虚拟内存的内存管理机制/VM as a tool for memory management
- 基于虚拟内存的内存保护机制/VM as a tool for memory protection
- 地址翻译/Address translation

基于虚拟内存的缓存机制/VM as a Tool for Caching

- 概念上来讲，虚拟内存就是N个连续地存储在磁盘上的字节/Conceptually, ***virtual memory*** **is an array of N contiguous bytes stored on disk.**
- 磁盘上的数组的内容是缓存在物理内存中的（DRAM）**The contents of the array on disk are cached in *physical memory* (*DRAM cache*)**
  - These cache blocks are called *pages* (size is $P = 2^p$ bytes)



虚拟页存储在磁盘上/Virtual pages (VPs) stored on disk

物理页缓存在DRAM中/ Physical pages (PPs) cached in DRAM

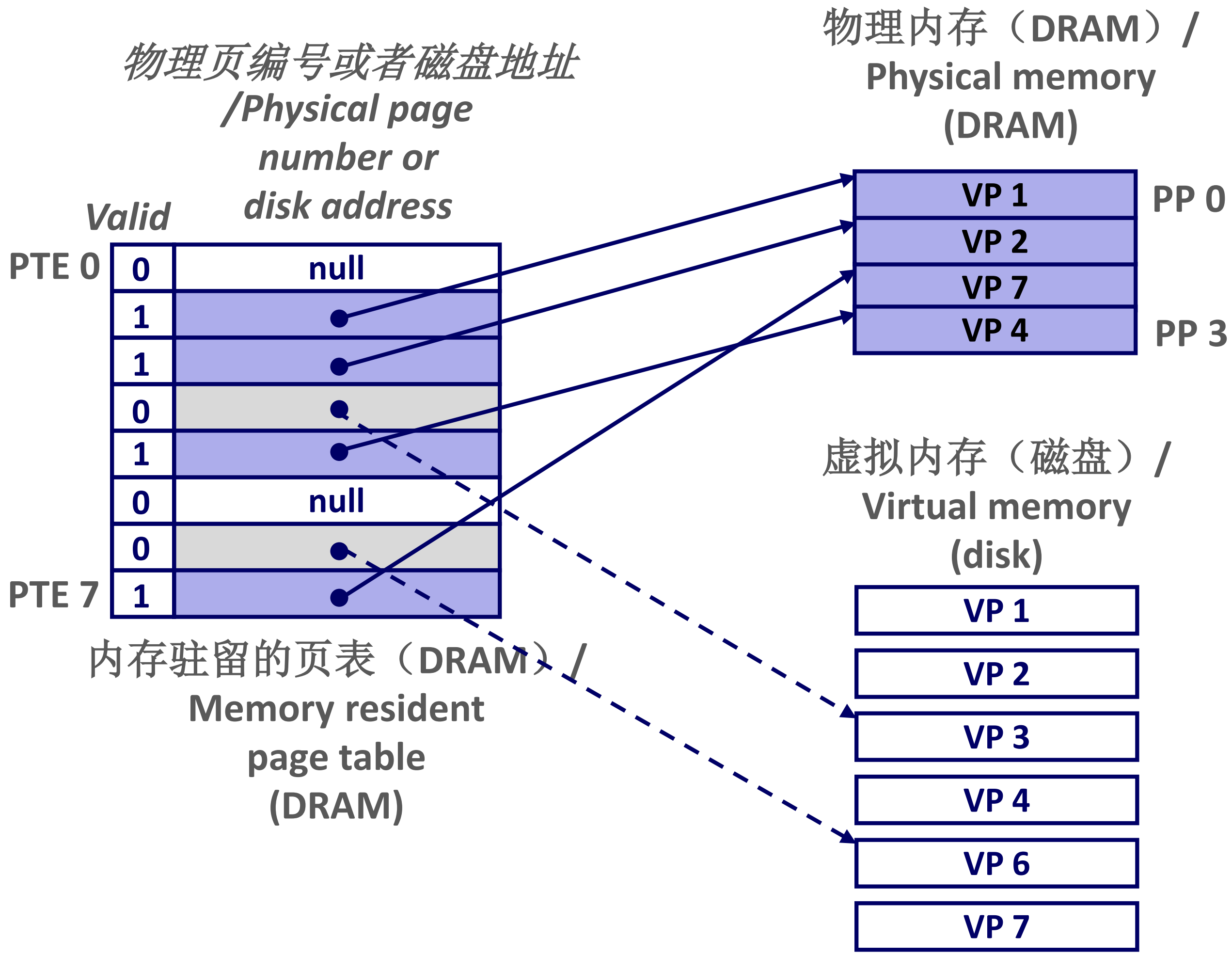
DRAM缓存组织/DRAM Cache Organization

- **DRAM缓存组织是受丢失后惩罚会很高这一因素影响的/DRAM cache organization driven by the enormous miss penalty**
  - DRAM大概比SRAM慢10倍左右/DRAM is about ***10x*** slower than SRAM
  - 磁盘大概DRAM慢10000倍/Disk is about ***10,000x*** slower than DRAM

- **因此/Consequences**
  - 比较大的页（块）：通常4 KB，有的4 MB/Large page (block) size: typically 4 KB, sometimes 4 MB
  - 全相联/Fully associative
    - 任意的虚拟页可以放在任意的物理页中/Any VP can be placed in any PP
    - 与Cache内存不同，需要一个更灵活的映射函数/Requires a “large” mapping function – different from cache memories
  - 高度复杂，替换算法开销比较大/Highly sophisticated, expensive replacement algorithms
    - 由于过于复杂和不确定性，无法在硬件中实现/Too complicated and open-ended to be implemented in hardware
  - 采用写回机制而不是写透机制/Write-back rather than write-through

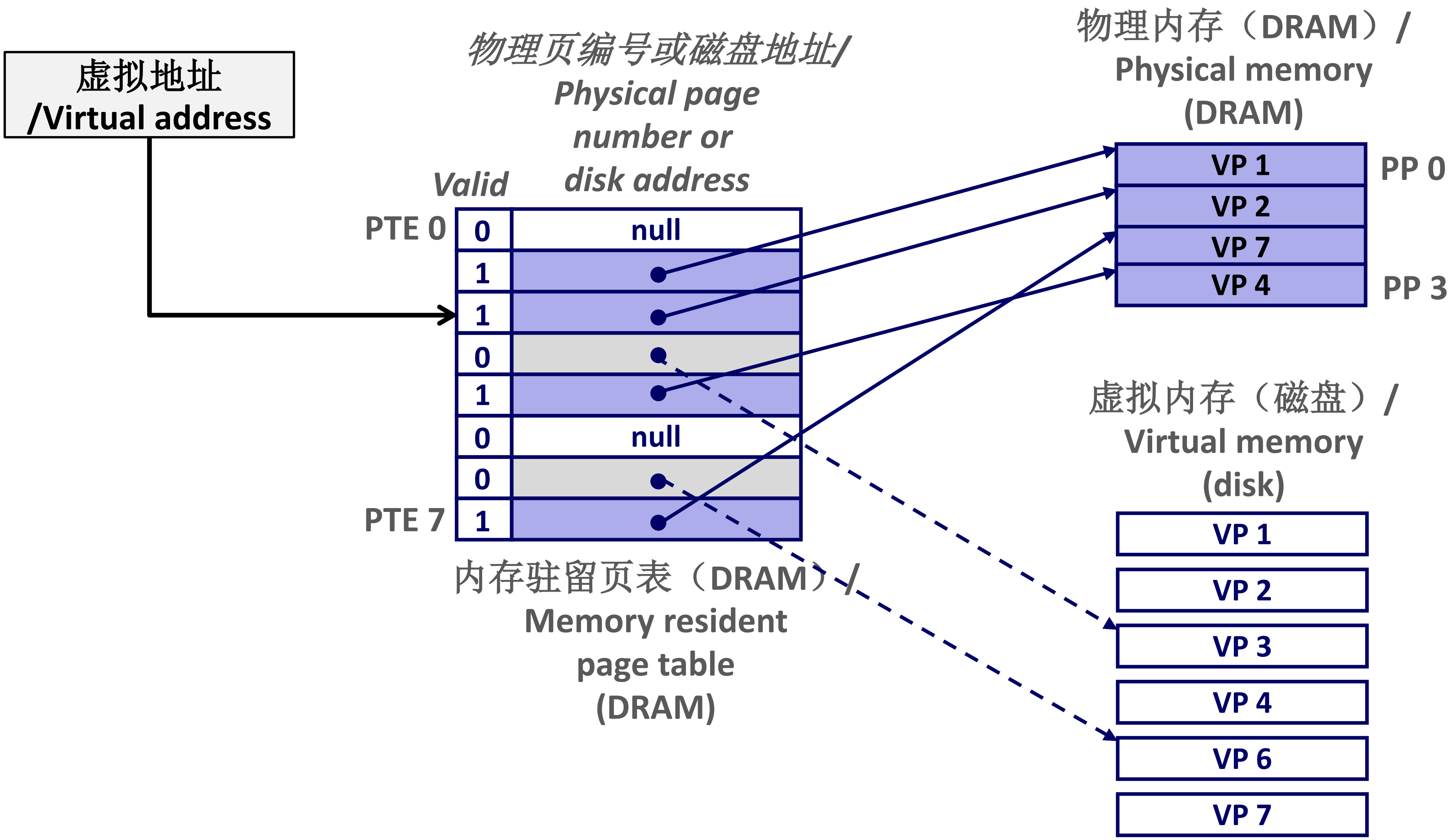
使能数据结构：页表/Enabling Data Structure: Page Table

- **一个页表实际上是将虚拟页映射物理页的页表条目构成的数组/A *page table* is an array of page table entries (PTEs) that maps virtual pages to physical pages.**
  - 每个集成在DRAM中的核心数据结构/Per-process kernel data structure in DRAM



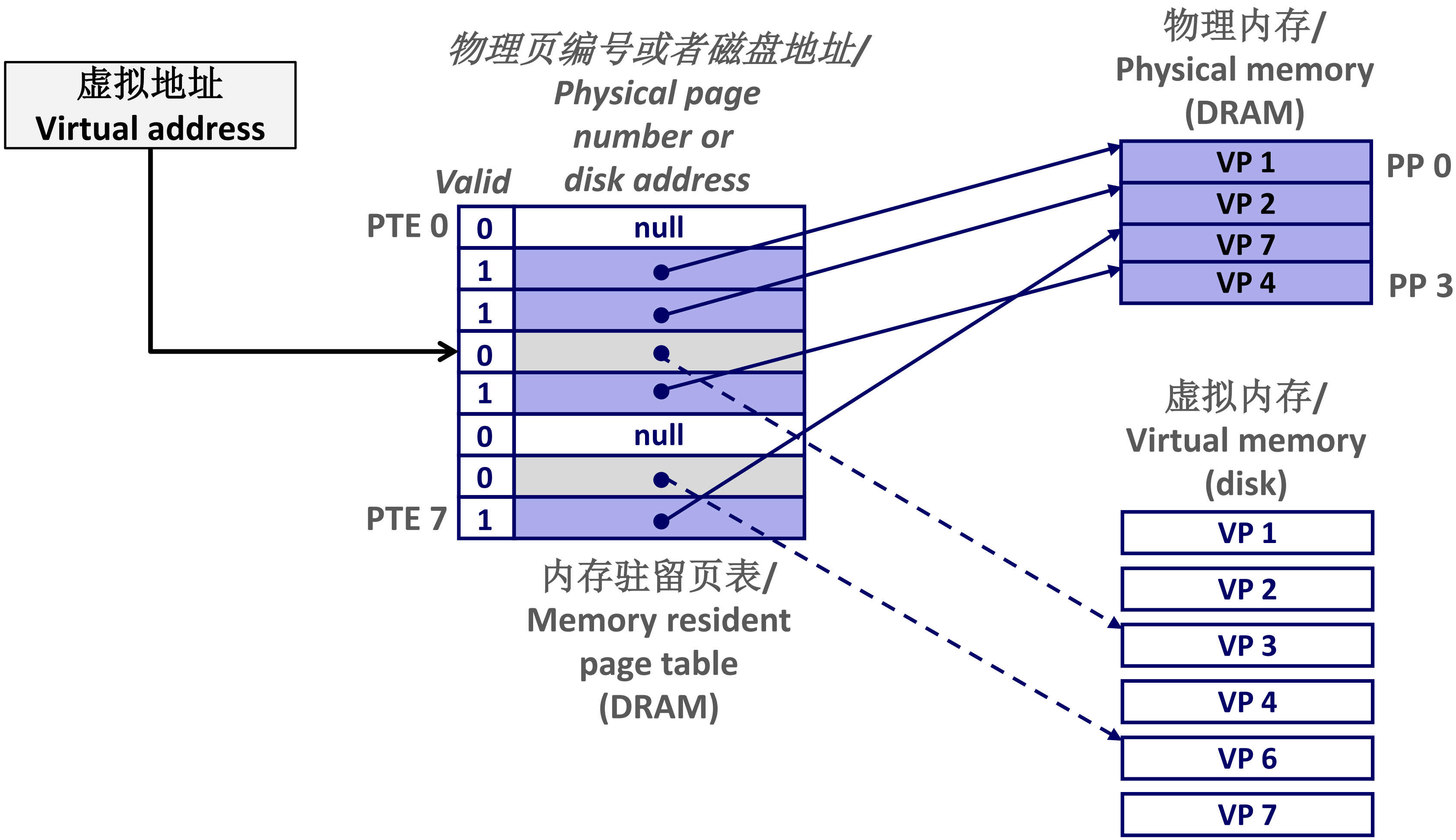
页命中/Page Hit

- *页命中：*引用的虚拟内存在在物理内存中（DRAM命中）*/Page hit:* reference to VM word that is in physical memory (DRAM cache hit)



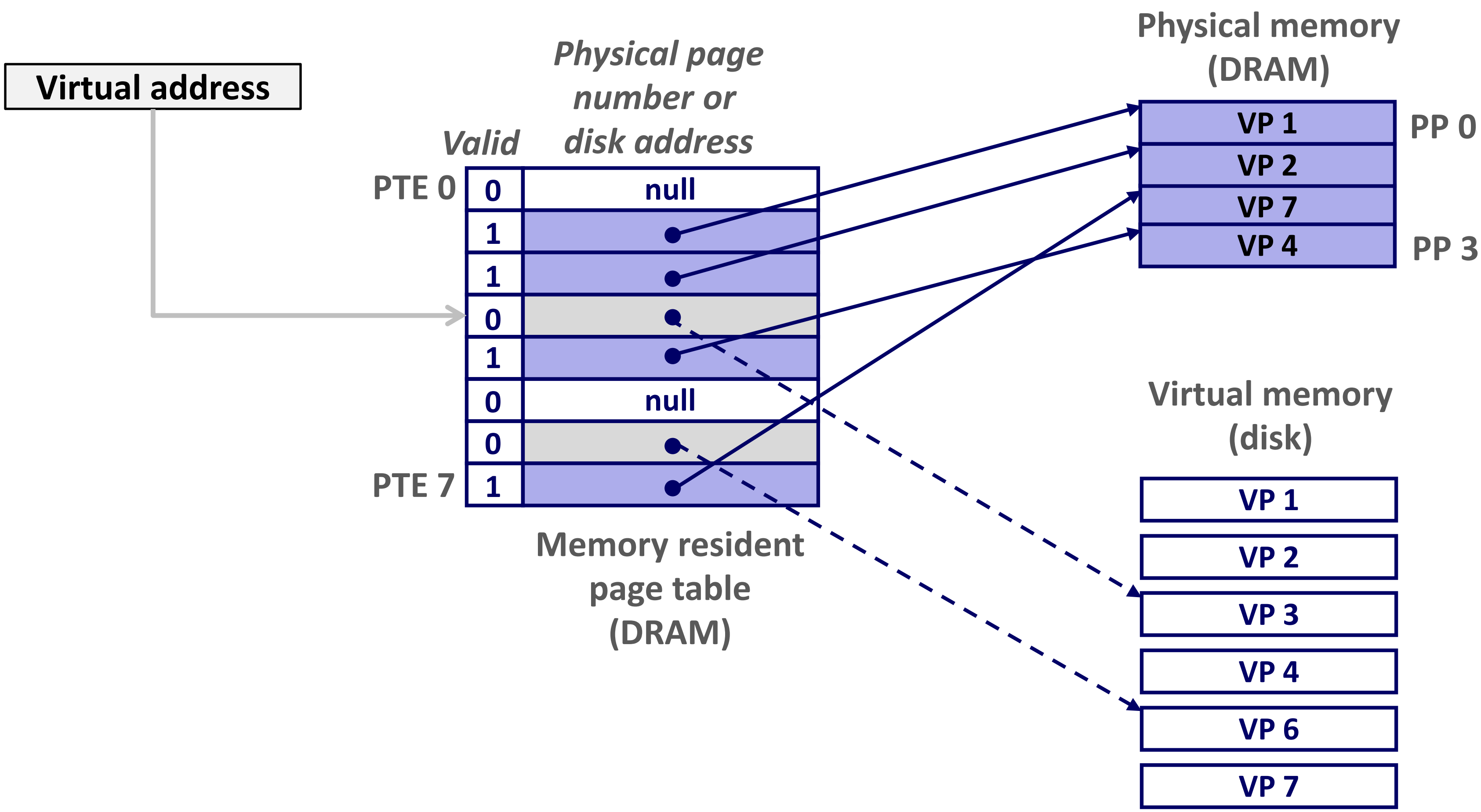
缺页中断/Page Fault

- *缺页中断*：引用的虚拟字不在物理内存中/***Page fault:*** **reference to VM word that is not in physical memory (DRAM cache miss)**



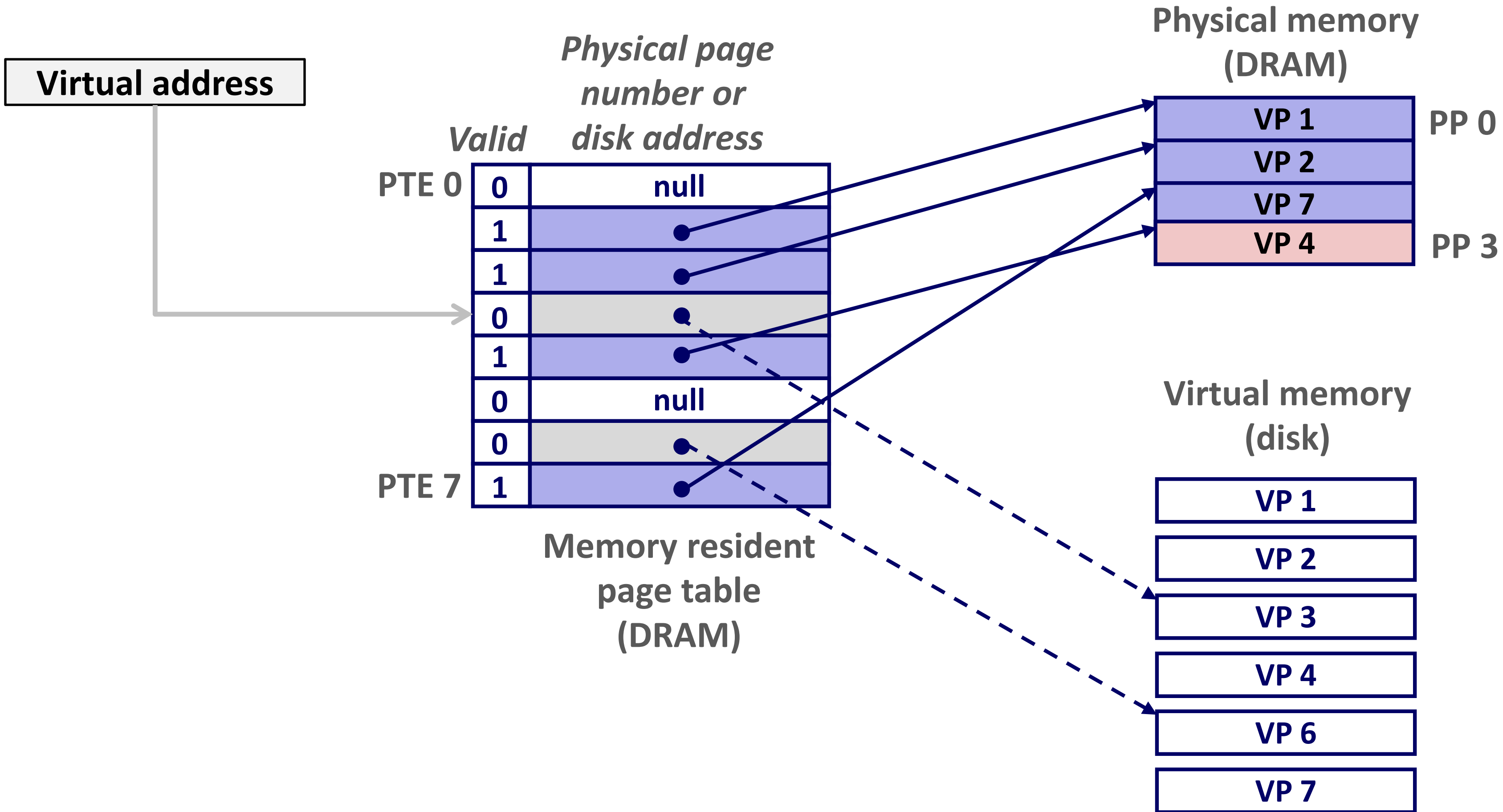
缺页中断处理/Handling Page Fault

- 页丢失导致缺页中断（异常的一种）/Page miss causes page fault (an exception)



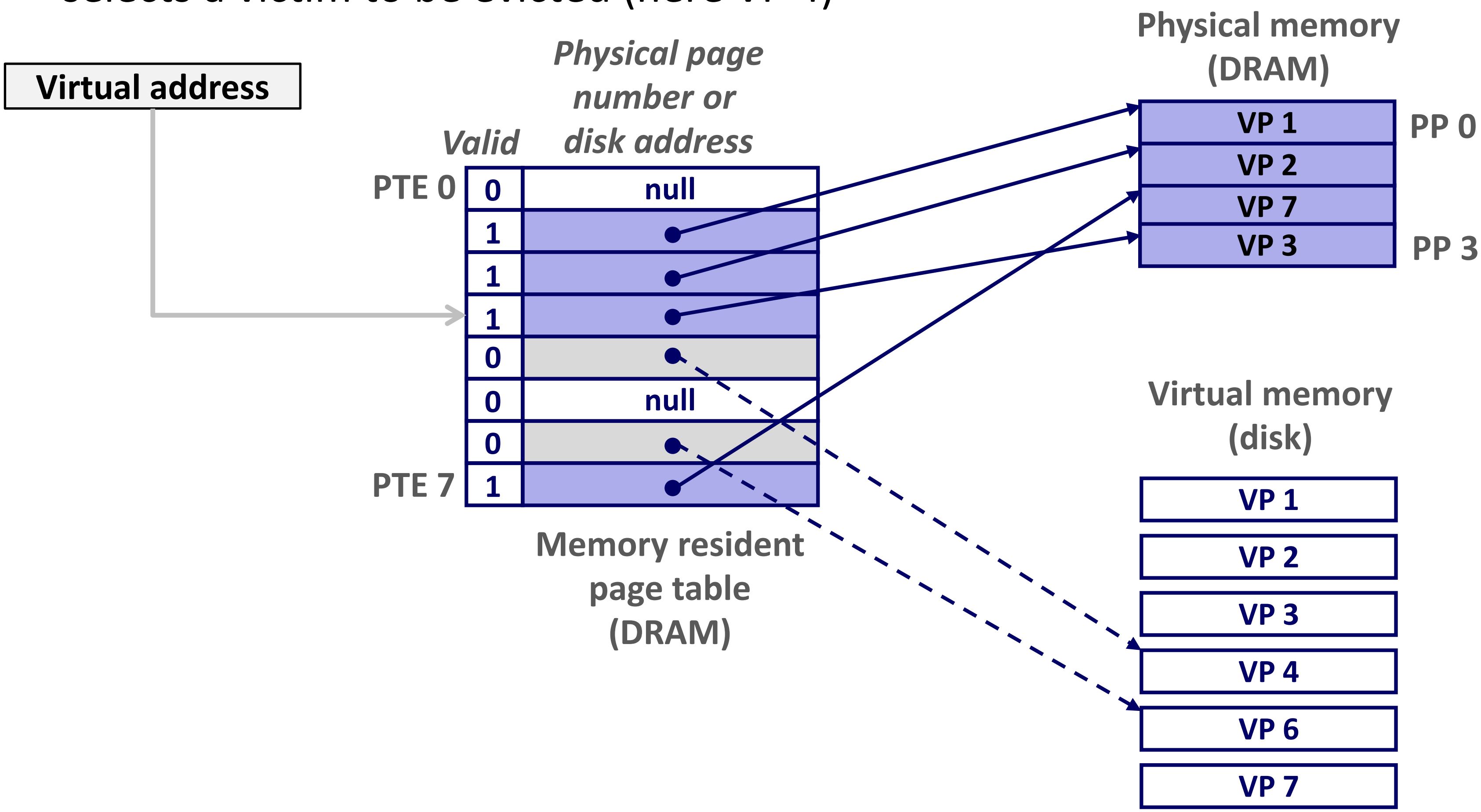
缺页中断处理/Handling Page Fault

- 页丢失导致缺页中断（异常的一种）/ Page miss causes page fault (an exception)
- 缺页中断处理程序选择一个条目换出（以VP 4为例）Page fault handler selects a victim to be evicted (here VP 4)



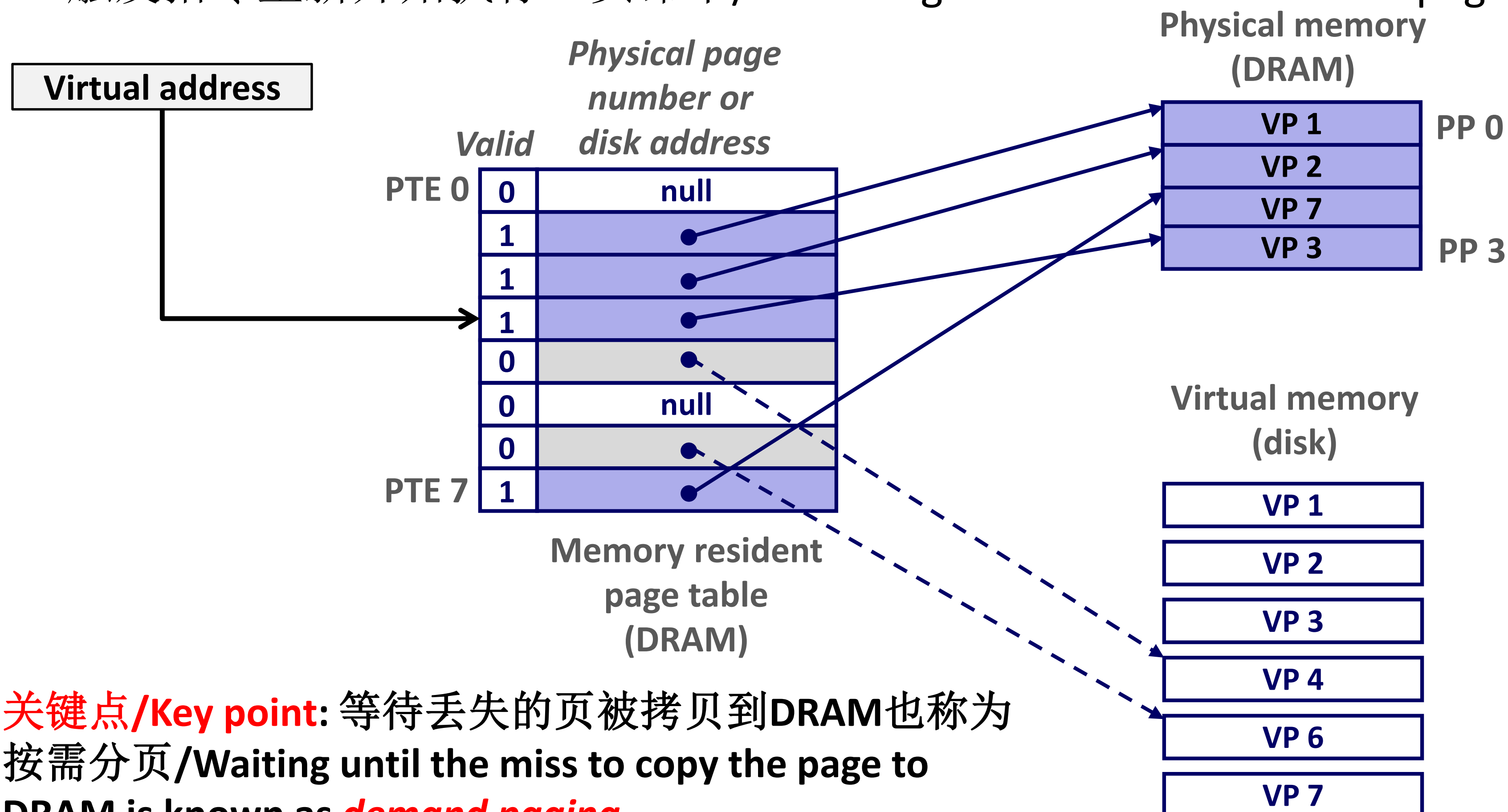
缺页中断处理/Handling Page Fault

- 页丢失导致缺页中断（异常的一种）/ Page miss causes page fault (an exception)
- 缺页中断处理程序选择一个条目换出（以VP 4为例）Page fault handler selects a victim to be evicted (here VP 4)



缺页中断处理/Handling Page Fault

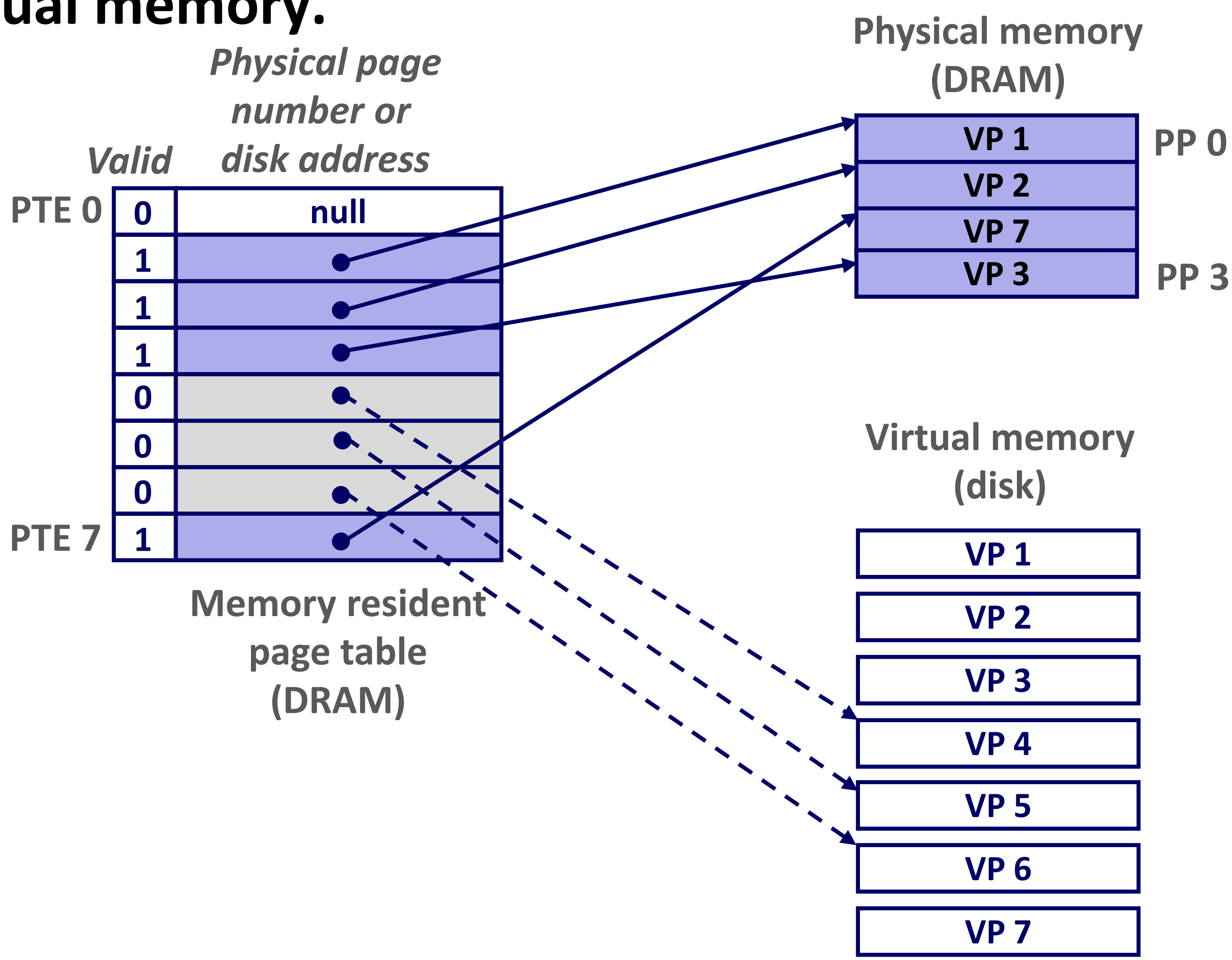
- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)
- 触发指令重新开始执行：页命中/Offending instruction is restarted: page hit!



关键点/Key point: 等待丢失的页被拷贝到DRAM也称为按需分页/Waiting until the miss to copy the page to DRAM is known as *demand paging*

页分配/Allocating Pages

- 分配虚拟内存的一个新页（VP 5）Allocating a new page (VP 5) of virtual memory.



局域性再次发挥作用/Locality to the Rescue Again!

- **虚拟内存看起来非常低效，能有效工作是因为局域性/Virtual memory seems terribly inefficient, but it works because of locality.**

- **在任何时间点，程序更倾向于只访问一个活跃的页集合，也称为工作集/At any point in time, programs tend to access a set of active virtual pages called the *working set***
  - Programs with better temporal locality will have smaller working sets

- **如果工作集的大小小于主存大小/If (working set size < main memory size)**
  - 每个进程在强制丢失后就会获得比较好的性能/Good performance for one process after compulsory misses

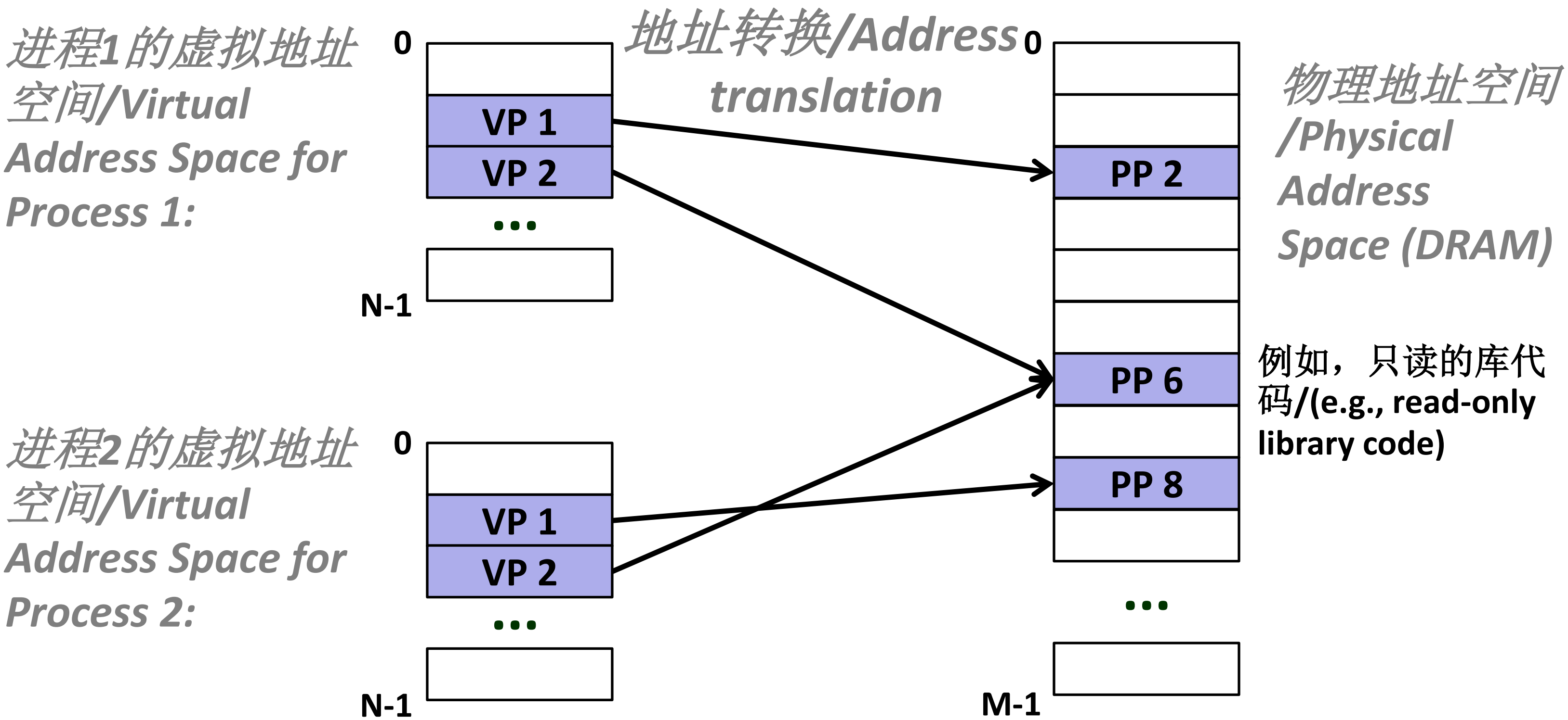
- **如果工作集的总大小大于主存大小/If (SUM(working set sizes) > main memory size)**
  - *抖动/Thrashing: 性能会由于持续的页面换入换出而变差*/Performance meltdown where pages are swapped (copied) in and out continuously

内容提纲/Today

- 地址空间/Address spaces
- 基于虚拟内存的缓存机制/VM as a tool for caching
- **基于虚拟内存的内存管理机制/VM as a tool for memory management**
- 基于虚拟内存的内存保护机制/VM as a tool for memory protection
- 地址翻译/Address translation

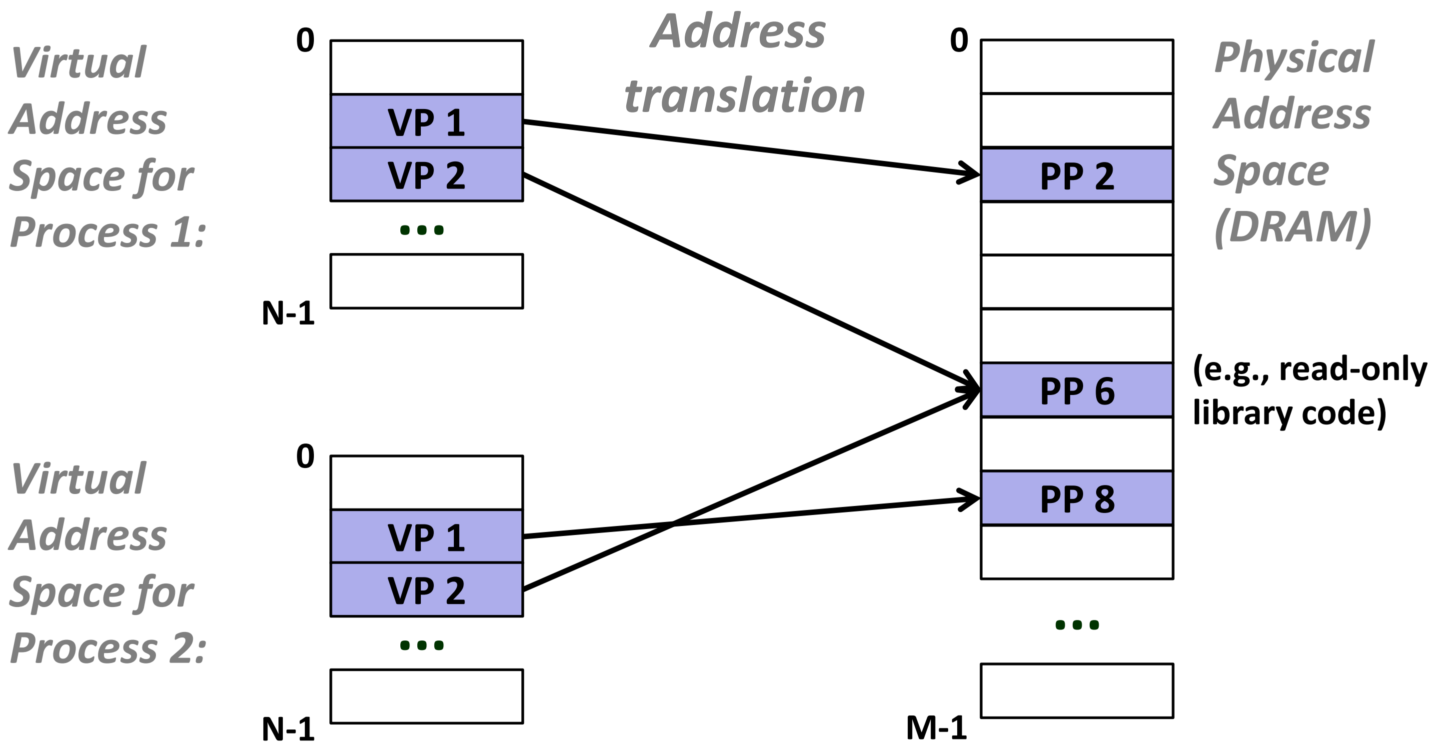
基于虚拟内存的内存管理机制/VM as a Tool for Memory Management

- **关键点：每个进程有自己的虚拟地址空间/Key idea: each process has its own virtual address space**
  - 将内存看做简单的线性数组/It can view memory as a simple linear array
  - 映射函数将地址分散到物理内存中/Mapping function scatters addresses through physical memory
    - 好的映射函数会提高局域性/Well-chosen mappings can improve locality



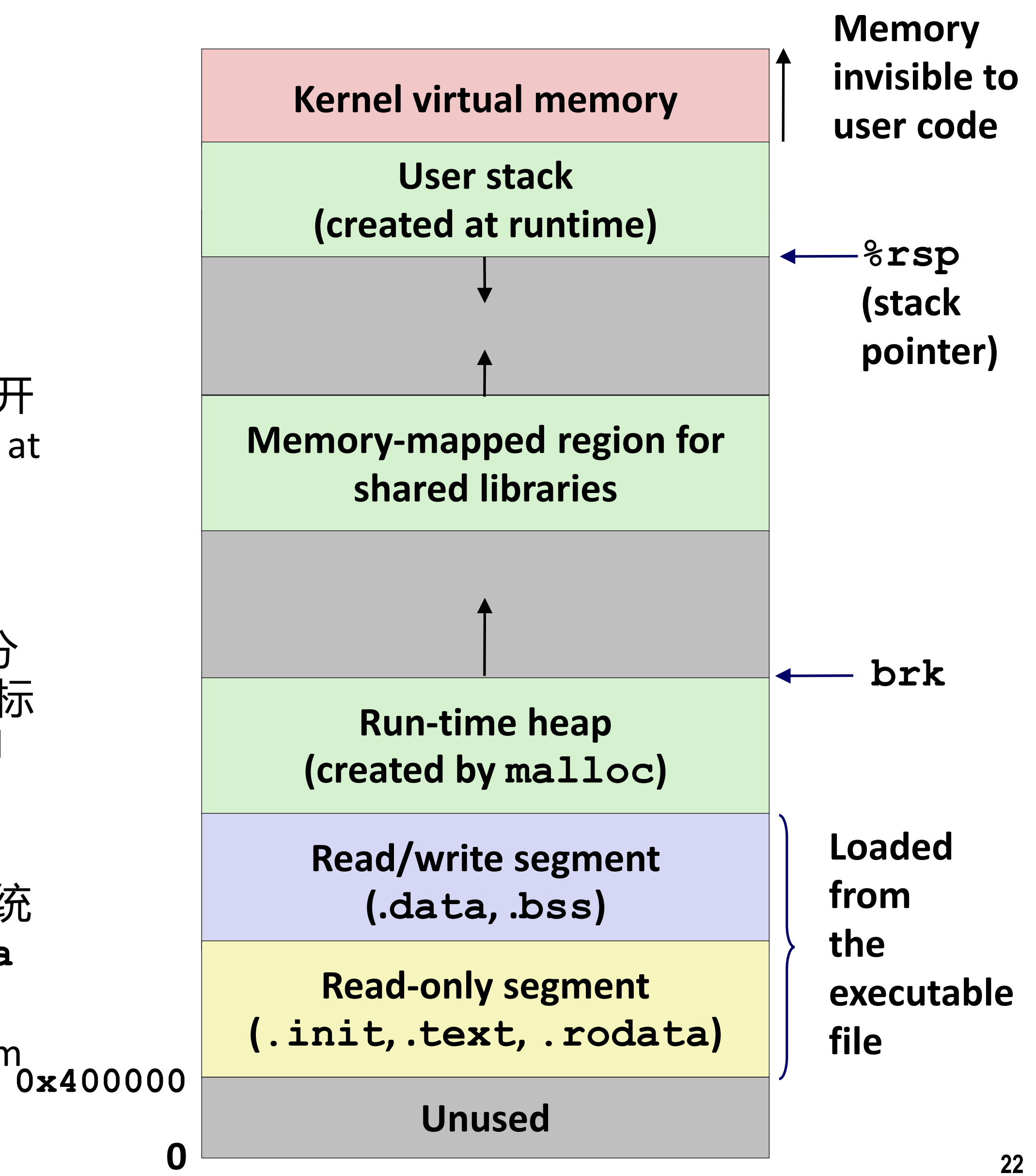
基于虚拟内存的内存管理机制/VM as a Tool for Memory Management

- 简化内存分配/**Simplifying memory allocation**
  - 每个虚拟页可以被映射到任意物理页/Each virtual page can be mapped to any physical page
  - 一个虚拟页可以在不同的时间点存储在不同的物理页中/A virtual page can be stored in different physical pages at different times
- 在进程间共享代码和数据/**Sharing code and data among processes**
  - 将虚拟页映射到同一个物理页/Map virtual pages to the same physical page (here: PP 6)



简化链接和加载/Simplifying Linking and Loading

- **链接/Linking**
  - 每个程序都有类似的虚拟地址空间/Each program has similar virtual address space
  - 代码、数据和堆总是从相同的地址开始/Code, data, and heap always start at the same addresses.
- **加载/Loading**
  - **`execve`**负责为`.text`和`.data`段分配虚拟也并创建页表记录，并将其标记位非法/**`execve`** allocates virtual pages for .text and .data sections & creates PTEs marked as invalid
  - .text和.data中的页是由虚拟内存系统按需拷贝的/The **`.text`** and **`.data`** sections are copied, page by page, on demand by the virtual memory system



内容提纲/Today

- 地址空间/Address spaces
- 基于虚拟内存的缓存机制/VM as a tool for caching
- 基于虚拟内存的内存管理机制/VM as a tool for memory management
- **基于虚拟内存的内存保护机制/VM as a tool for memory protection**
- 地址翻译/Address translation

基于虚拟内存的内存保护机制/VM as a Tool for Memory Protection

- 对页表记录进行扩展增加权限位/Extend PTEs with permission bits
- MMU在每个内存访问时检查/MMU checks these bits on each access

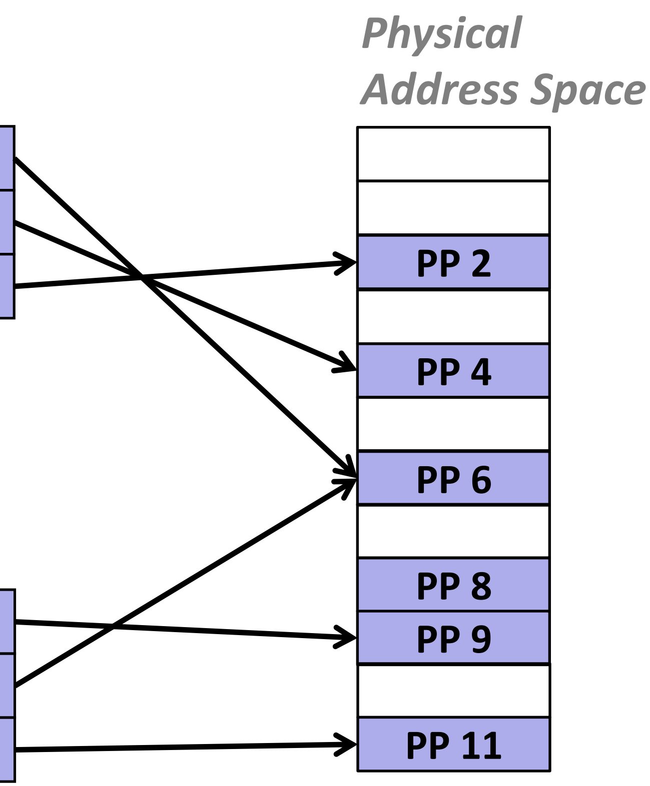
Process i:

	SUP	READ	WRITE	EXEC	Address
VP 0:	No	Yes	No	Yes	PP 6
VP 1:	No	Yes	Yes	Yes	PP 4
VP 2:	Yes	Yes	Yes	No	PP 2

⋮

Process j:

	SUP	READ	WRITE	EXEC	Address
VP 0:	No	Yes	No	Yes	PP 9
VP 1:	Yes	Yes	Yes	Yes	PP 6
VP 2:	No	Yes	Yes	Yes	PP 11



内容提纲/Today

- 地址空间/Address spaces
- 基于虚拟内存的缓存机制/VM as a tool for caching
- 基于虚拟内存的内存管理机制/VM as a tool for memory management
- 基于虚拟内存的内存保护机制/VM as a tool for memory protection
- 地址翻译/Address translation

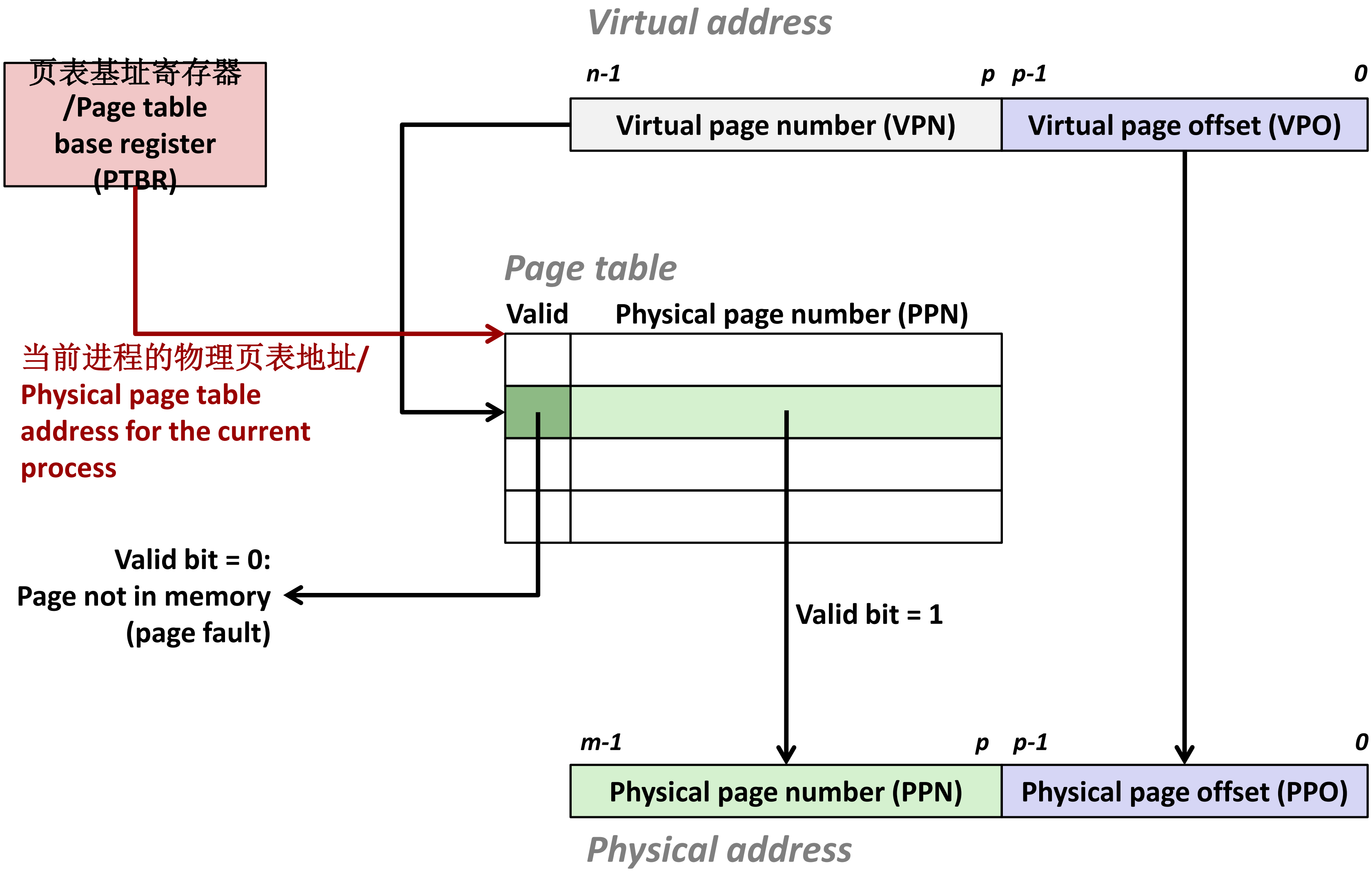
虚拟地址翻译/VM Address Translation

- **虚拟地址/Virtual Address Space**
  - $V = \{0, 1, \ldots, N-1\}$
- **物理地址/Physical Address Space**
  - $P = \{0, 1, \ldots, M-1\}$
- **地址翻译/Address Translation**
  - ***映射/MAP:*** $V \rightarrow P \cup \{\varnothing\}$
  - 对于虚拟地址/For virtual address ***a***:
    - ***MAP(a) = a'*** if data at virtual address ***a*** is at physical address ***a'*** in ***P*/如果数据虚拟地址a在p中的物理地址a'**
    - ***MAP(a) =*** $\varnothing$ if data at virtual address ***a*** is not in physical memory/如果数据虚拟地址a不在物理内存
      - 非法的或者在磁盘上/Either invalid or stored on disk

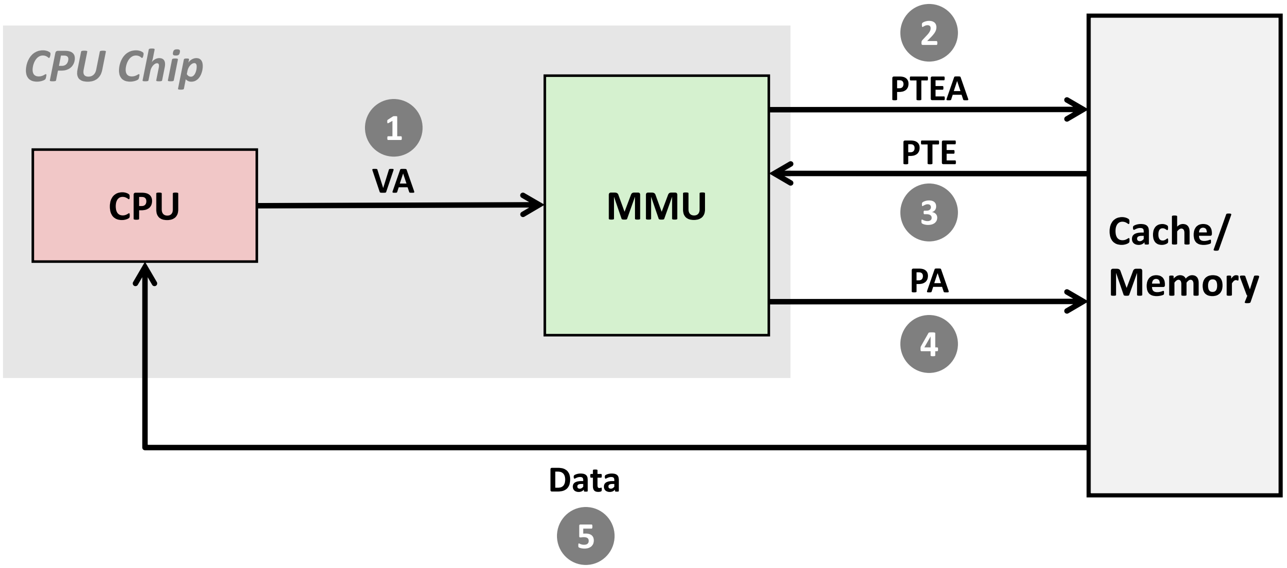
地址翻译符号总结/Summary of Address Translation Symbols

- 基本参数/Basic Parameters
  - **N = 2^n** : Number of addresses in virtual address space/虚拟地址空间的地址个数
  - **M = 2^m** : Number of addresses in physical address space/物理地址空间的地址个数
  - **P = 2^p** : Page size (bytes)/页大小（字节）
- 虚拟地址划分/Components of the virtual address (VA)
  - **TLBI**: TLB index/TLB索引
  - **TLBT**: TLB tag/TLB标记
  - **VPO**: Virtual page offset /虚拟地址页偏移
  - **VPN**: Virtual page number /虚拟地址页号
- 物理地址划分/Components of the physical address (PA)
  - **PPO**: Physical page offset (same as VPO)/物理页偏移量（同VPO）
  - **PPN:** Physical page number/物理页编号

基于页表的地址翻译/Address Translation With a Page Table



地址翻译：页命中/Address Translation: Page Hit



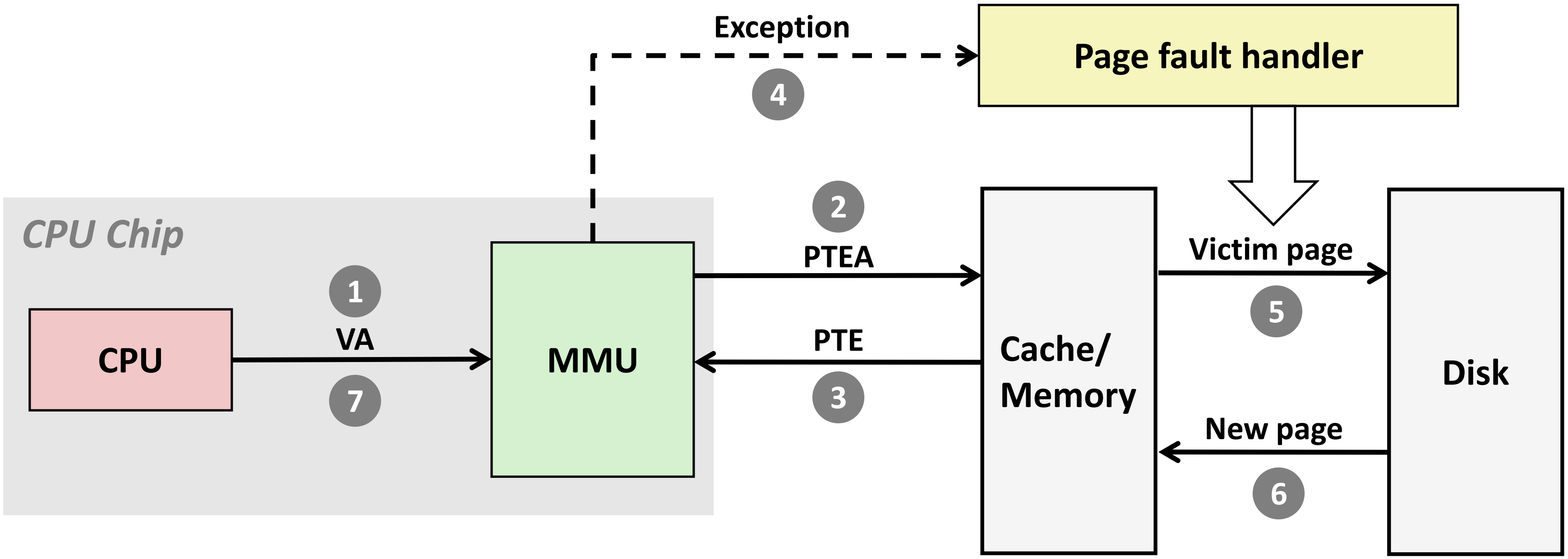
1) 处理器将虚拟地址发送给MMU/Processor sends virtual address to MMU

2-3) MMU将内存中页面里面的页表记录取出来/MMU fetches PTE from page table in memory

4) MMU将物理地址发给Cache或者主存/MMU sends physical address to cache/memory

5) Cache或者主存将数据发送给处理器/Cache/memory sends data word to processor

地址翻译：缺页中断/Address Translation: Page Fault



1) 处理器将虚拟地址发给MMU/Processor sends virtual address to MMU

2-3)MMU从内存中的页表取出页表记录/ MMU fetches PTE from page table in memory

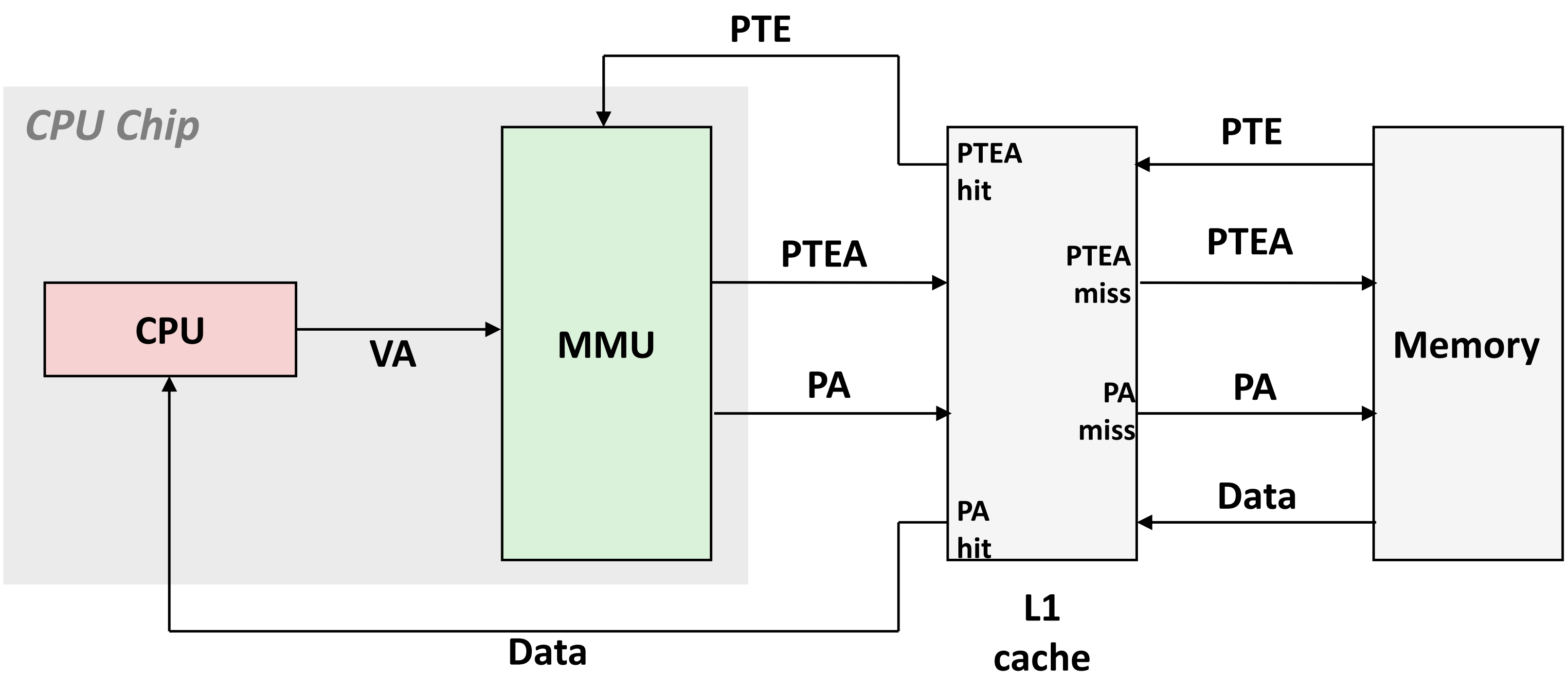
4) 当合法位为0时MMU触发缺页中断异常/Valid bit is zero, so MMU triggers page fault exception

5) 异常处理程序找到一个换出页（如果是脏页则要写回磁盘）/Handler identifies victim (and, if dirty, pages it out to disk)

6) 异常处理程序拷贝页并更新页表记录/Handler pages in new page and updates PTE in memory

7)异常处理程序返回原进程中断的指令重新执行/Handler returns to original process, restarting faulting instruction

整合虚拟地址和Cache/Integrating VM and Cache



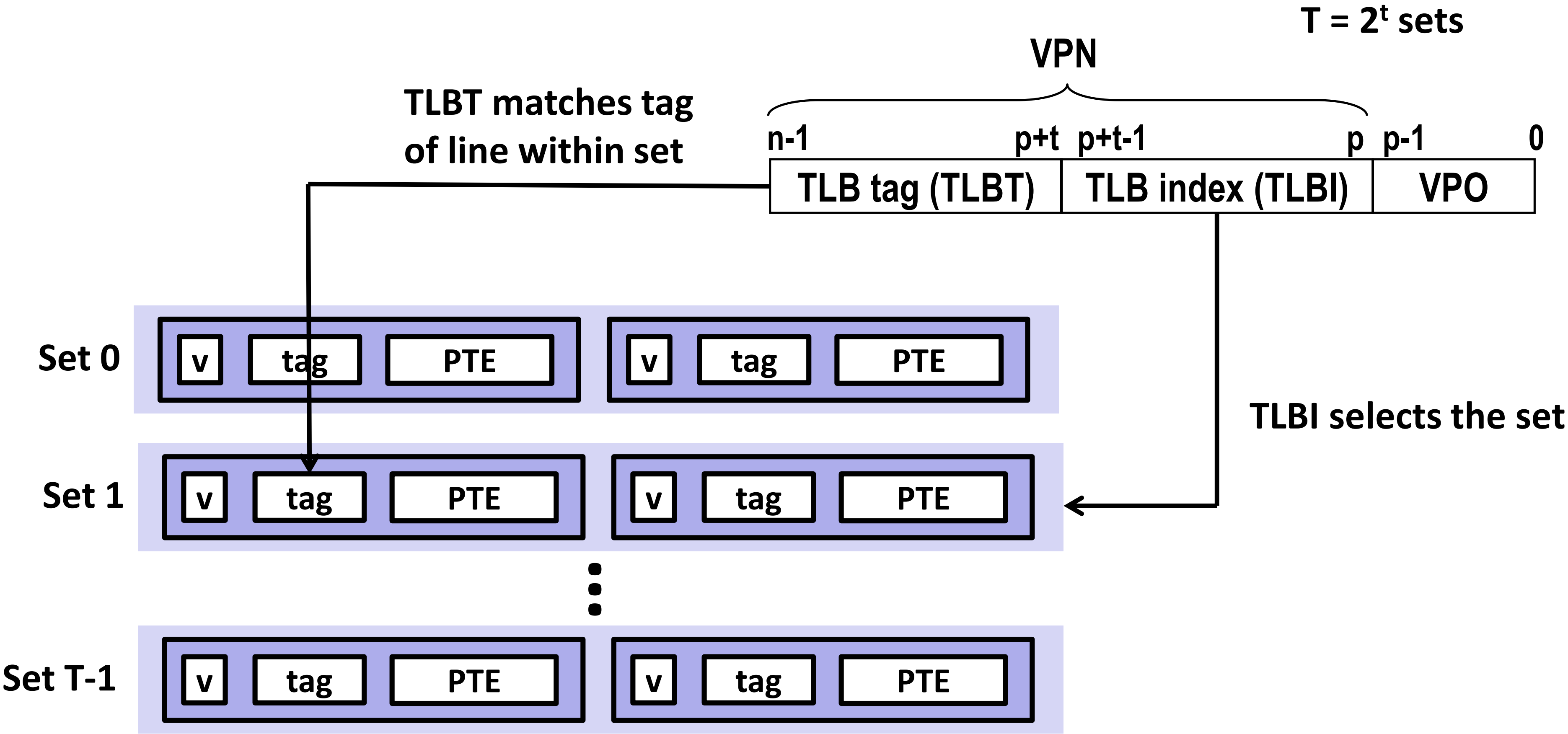
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

使用TLB加速地址翻译/Speeding up Translation with a TLB

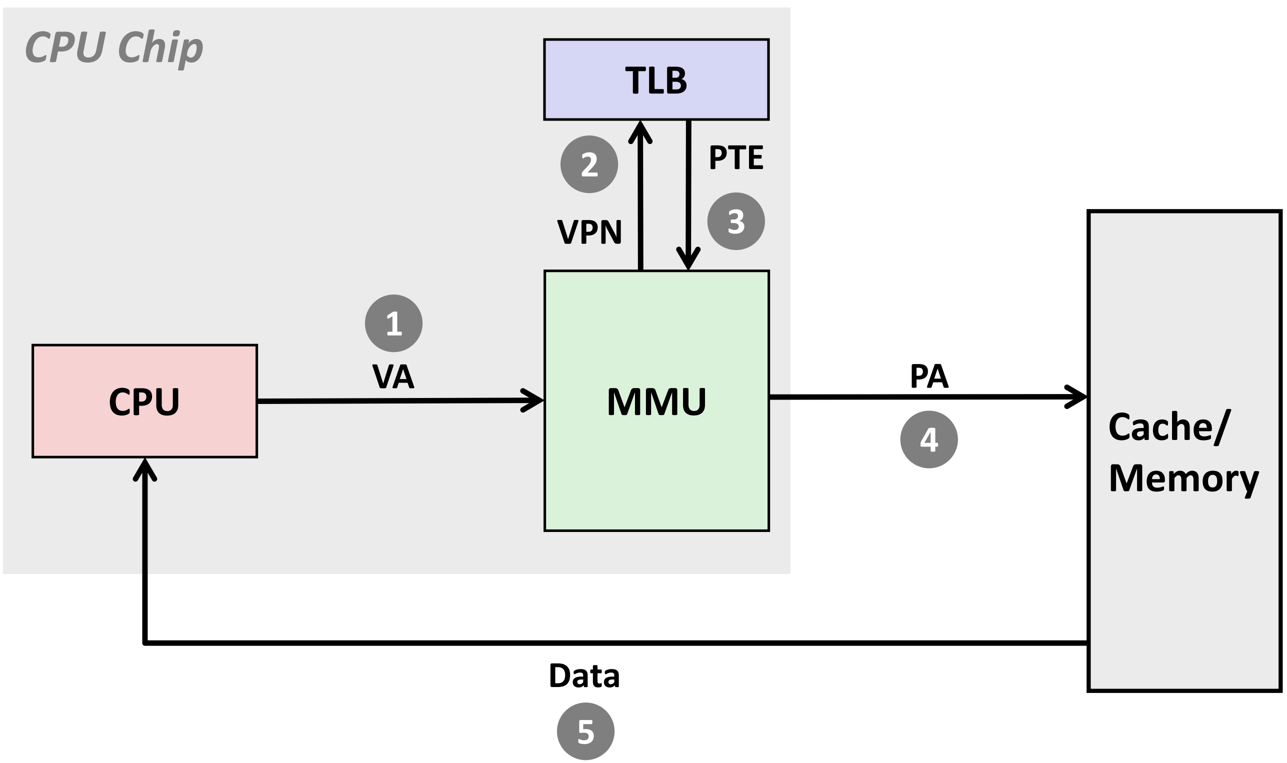
- **TLB记录像其他数据一样缓存在L1中/Page table entries (PTEs) are cached in L1 like any other memory word**
  - 由于其他数据访问PTE可能会被驱逐出内存/PTEs may be evicted by other data references
  - PTE命中则延迟较小/PTE hit still requires a small L1 delay
- **解决方案：/Solution: *Translation Lookaside Buffer* (TLB)**
  - 在MMU中的全相联硬件缓存/Small set-associative hardware cache in MMU
  - 将虚拟页编号映射为物理页编号/Maps virtual page numbers to physical page numbers
  - 包含了一少部分页表记录的全部信息/Contains complete page table entries for small number of pages

访问TLB/Accessing the TLB

- **MMU使用虚拟地址的VPN部分访问TLB/MMU uses the VPN portion of the virtual address to access the TLB:**

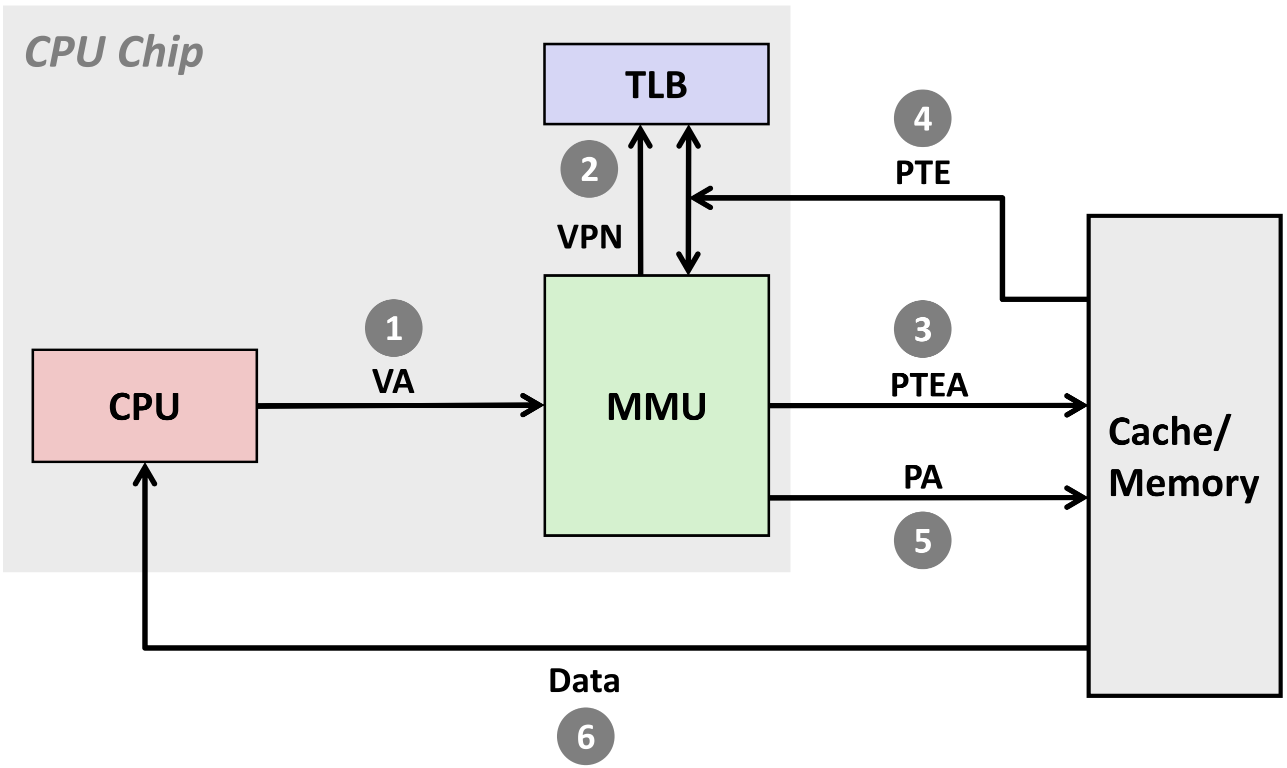


TLB命中/TLB Hit



TLB命中会减少一个内存访问/A TLB hit eliminates a memory access

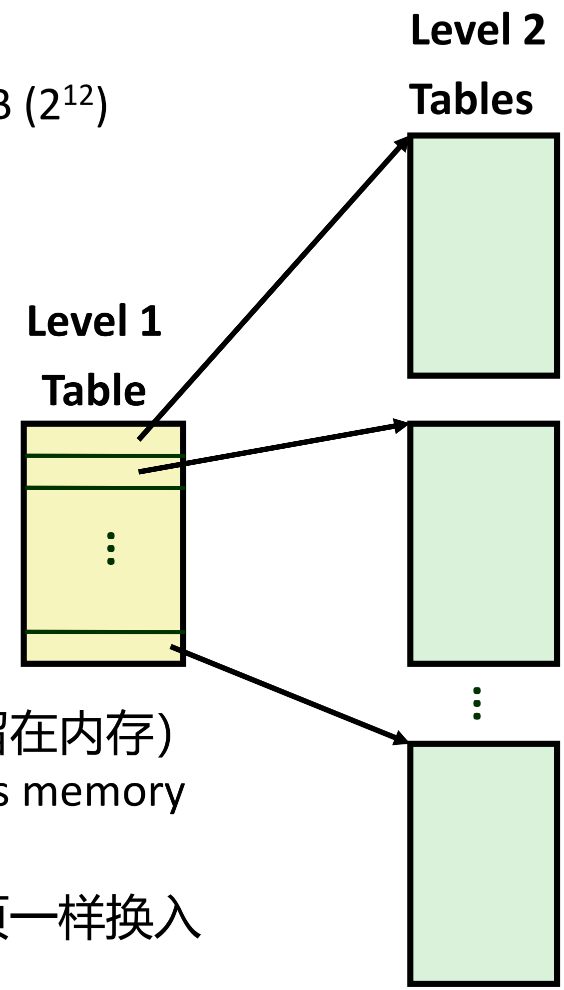
TLB丢失/TLB Miss



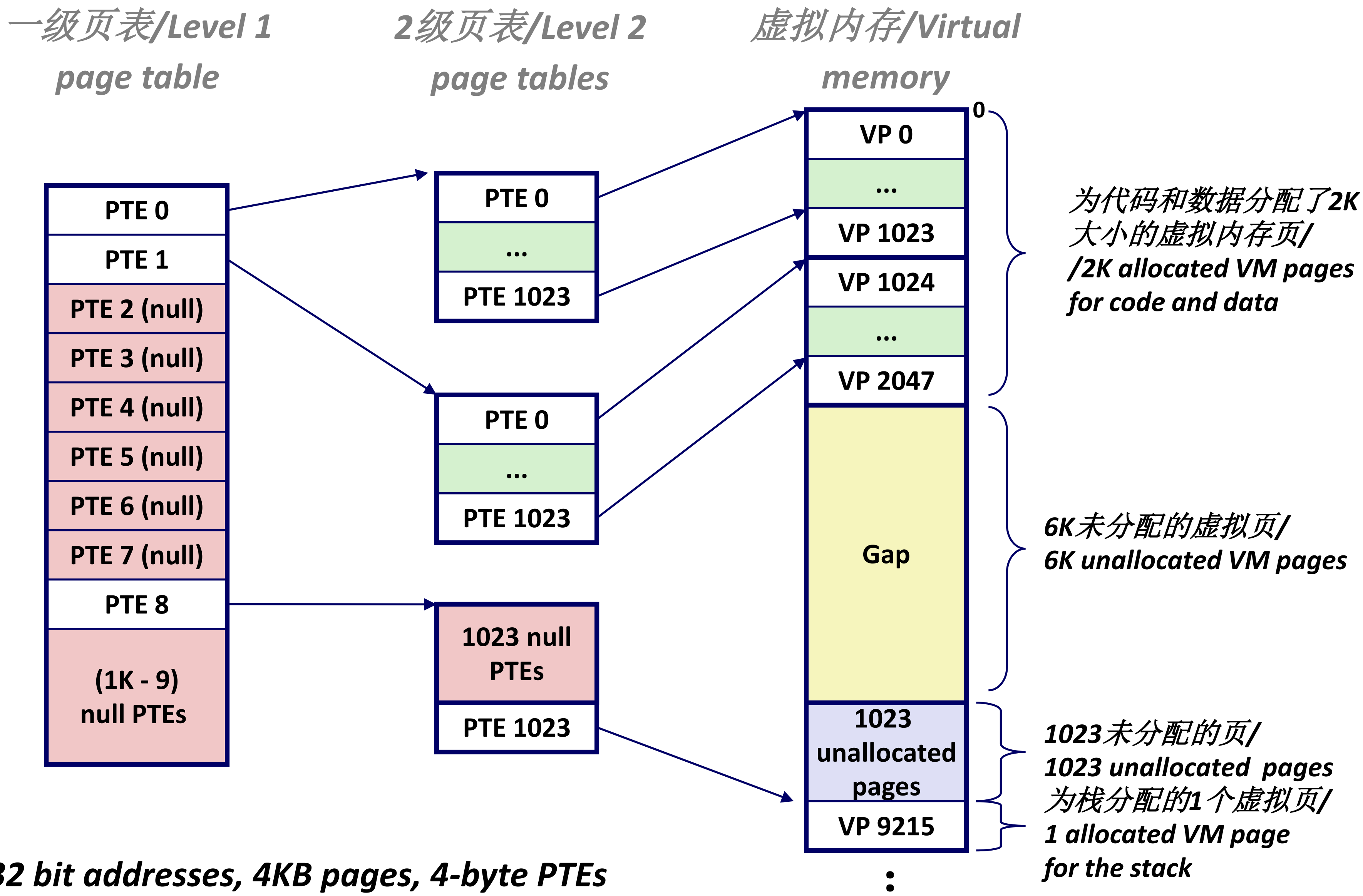
TLB丢失会导致一个额外的内存访问，幸运的是，TLB丢失很少发生
/A TLB miss incurs an additional memory access (the PTE)
Fortunately, TLB misses are rare. Why?

多级页表/Multi-Level Page Tables

- **假设/Suppose:**
  - 4KB大小页表，48位地址空间，8字节页表记录/4KB (2^{12}) page size, 48-bit address space, 8-byte PTE
- **问题/Problem:**
  - 页表占用的空间将高达512GB/
  - Would need a 512 GB page table!
    - $2^{48} * 2^{-12} * 2^{3} = 2^{39}$ bytes
- **常见方法：多级页表Common solution: Multi-level page table**
- **例如：2级页表/Example: 2-level page table**
  - 一级页表：每个页表记录指向一个页表（总是驻留在内存）/Level 1 table: each PTE points to a page table (always memory resident)
  - 二级页表：每个页表记录指向一个页表（像其他页一样换入换出）/Level 2 table: each PTE points to a page (paged in and out like any other data)

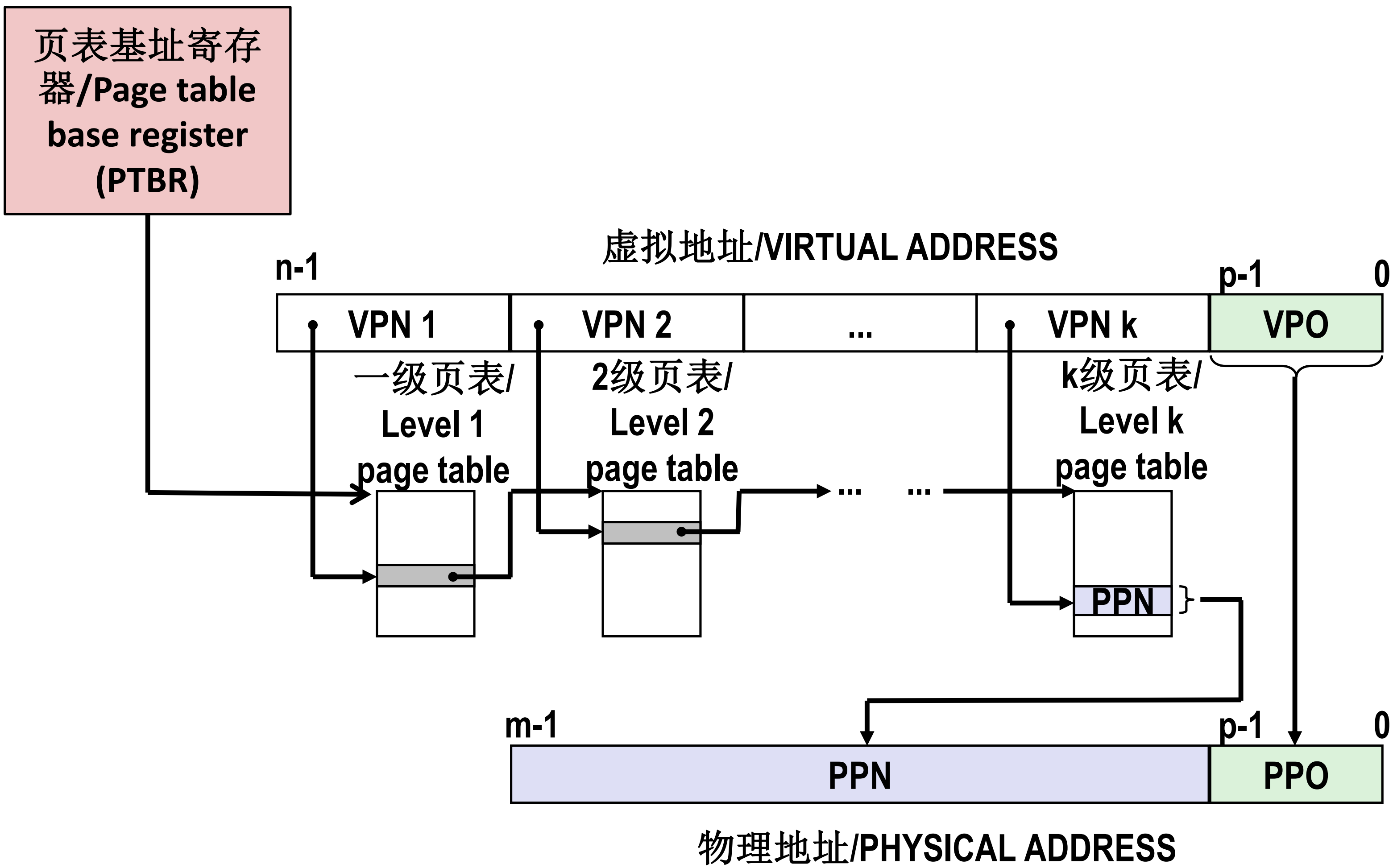


二级页表结构/A Two-Level Page Table Hierarchy



32 bit addresses, 4KB pages, 4-byte PTEs

k级页表的地址翻译/Translating with a k-level Page Table



总结/Summary

- **程序员眼中的虚拟内存/Programmer's view of virtual memory**
  - 每个进程都有各自私有的线性地址空间/Each process has its own private linear address space
  - 不能被其他进程破坏/Cannot be corrupted by other processes
- **系统眼中的虚拟内存/System view of virtual memory**
  - 通过缓存虚拟内存页高效地使用内存/Uses memory efficiently by caching virtual memory pages
    - 高效是因为局域性/Efficient only because of locality
  - 简化内存管理和编程/Simplifies memory management and programming
  - 通过提供方便的插入点来检查权限，简化了保护/Simplifies protection by providing a convenient interpositioning point to check permissions