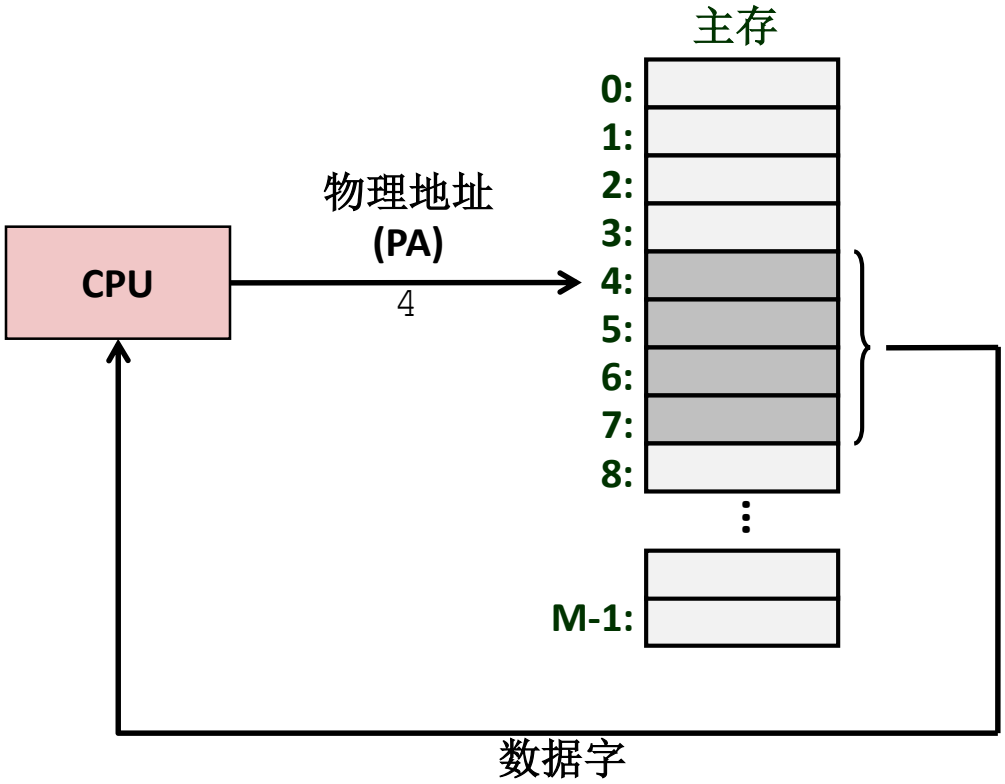
第9章 虚拟内存: 基本概念

主要内容

- **Address spaces 地址空间**
- **VM as a tool for caching 虚拟内存作为缓存的工具**
- **VM as a tool for memory management**
  虚拟内存作为内存管理的工具
- **VM as a tool for memory protection**
  虚拟内存作为内存保护的工具
- **Address translation地址翻译**

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



- 诸如汽车、电梯、数字图像帧（digital picture frame）等“简单”系统中作为嵌入式微控制器使用

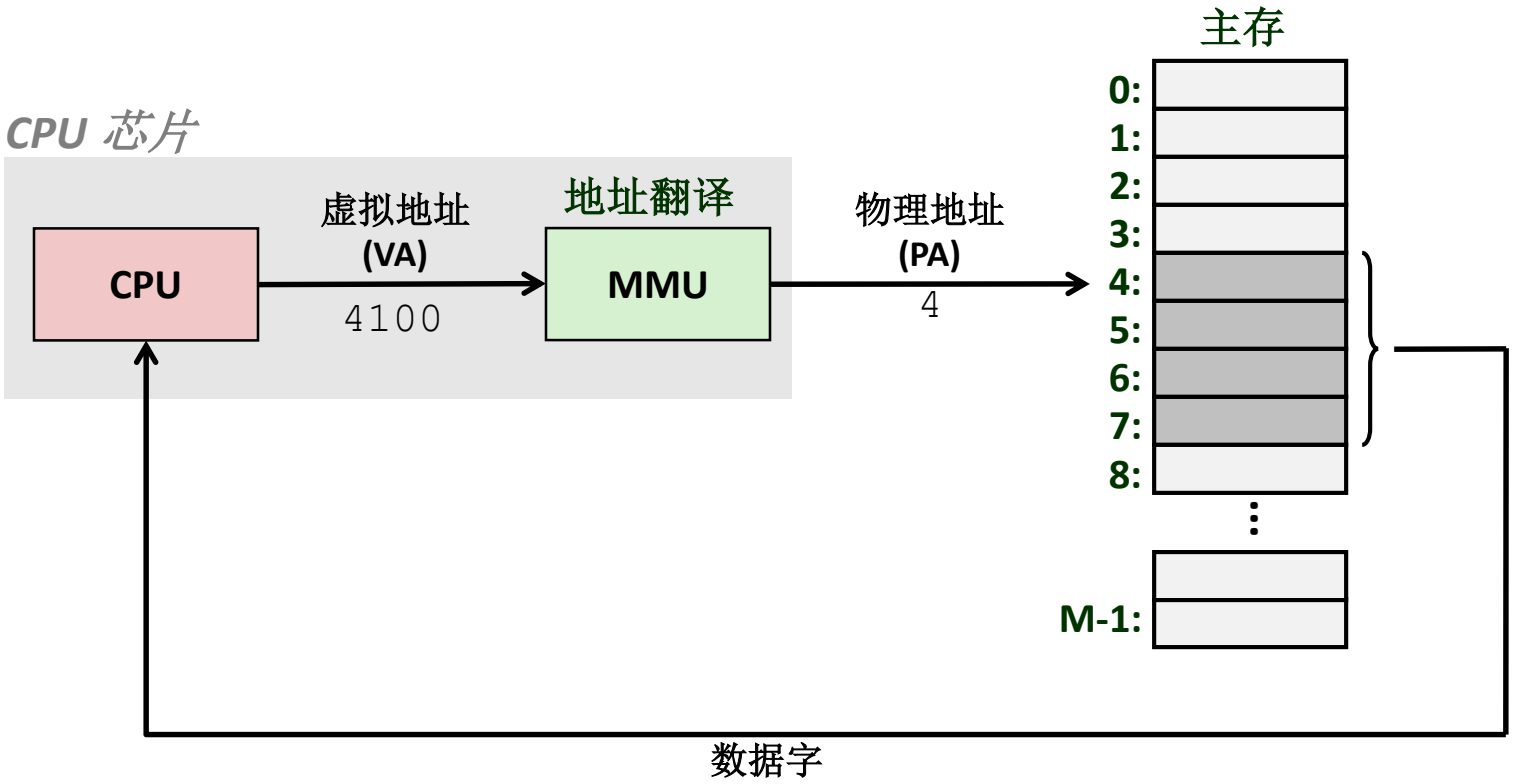
Address Spaces 地址空间

- **线性地址空间:** 非负整数地址的有序集合:
  {0, 1, 2, 3 … }
- **虚拟地址空间:** N = 2^n 个虚拟地址的集合 ===线性地址空间
  {0, 1, 2, 3, …, N-1}
- **物理地址空间:** M = 2^m 个物理（内存）地址的集合
  {0, 1, 2, 3, …, M-1}

主存中的每个字节都有一个选自虚拟地址空间的虚拟地址和一个选自物理地址空间的物理地址

A System Using Physical Addressing

一个使用物理寻址的系统



- 现在处理器使用，比如笔记本、智能电话等
- 计算机科学的伟大思想之一

Why Virtual Memory (VM)?

为什么要使用虚拟内存?

- **有效使用主存**
  - 使用DRAM作为部分虚拟地址空间的缓存
- **简化内存管理**
  - 每个进程都使用统一的线性地址空间
- **独立地址空间**
  - 一个进程不能影响其他进程的内存
  - 用户程序无法获取特权内核信息和代码

主要内容

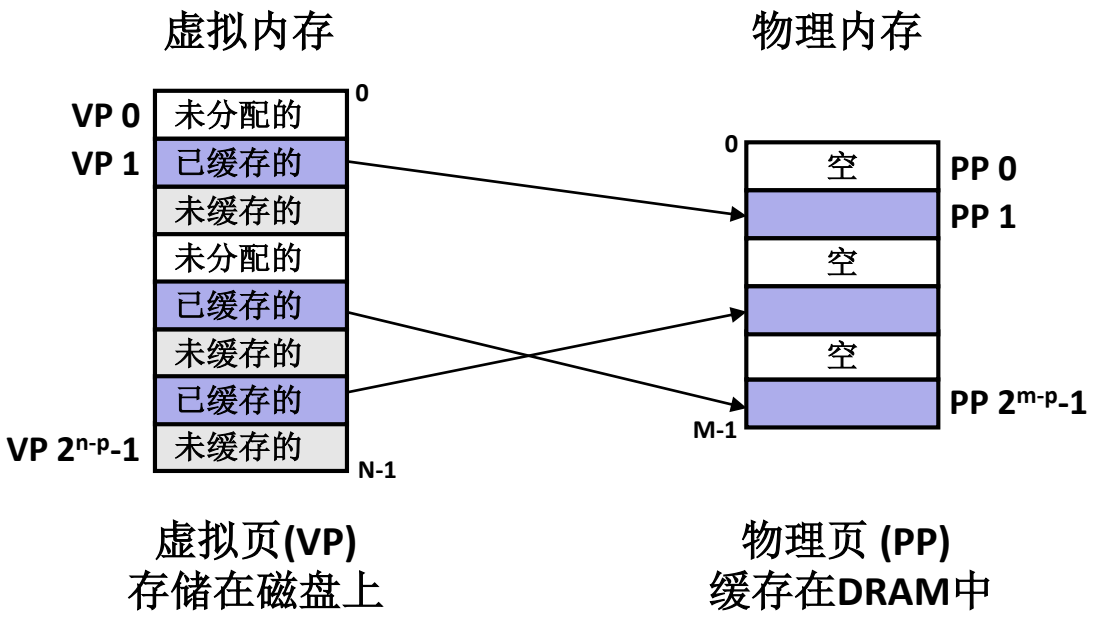
- Address spaces 地址空间
- **VM as a tool for caching 虚拟内存作为缓存的工具**
- VM as a tool for memory management
  虚拟内存作为内存管理的工具
- VM as a tool for memory protection
  虚拟内存作为内存保护的工具
- Address translation地址翻译

VM as a tool for caching

虚拟内存作为缓存的工具

- 概念上而言，虚拟内存被组织为一个由存放在磁盘上的N个连续的字节大小的单元组成的数组。
- 磁盘上数组的内容被缓存在*物理内存中* (***DRAM cache***)
  - 这些内存块被称为页 (每个页面的大小为P = 2^p字节)

任意时刻，虚拟页面的集合都被分为三个不相交的子集：
1、未分配的：VM系统还没分配/创建的页，不占用任何磁盘空间。
2、已缓存的：当前缓存在物理内存中的已分配页。
3、未缓存的：没有缓存在物理内存中的已分配页。

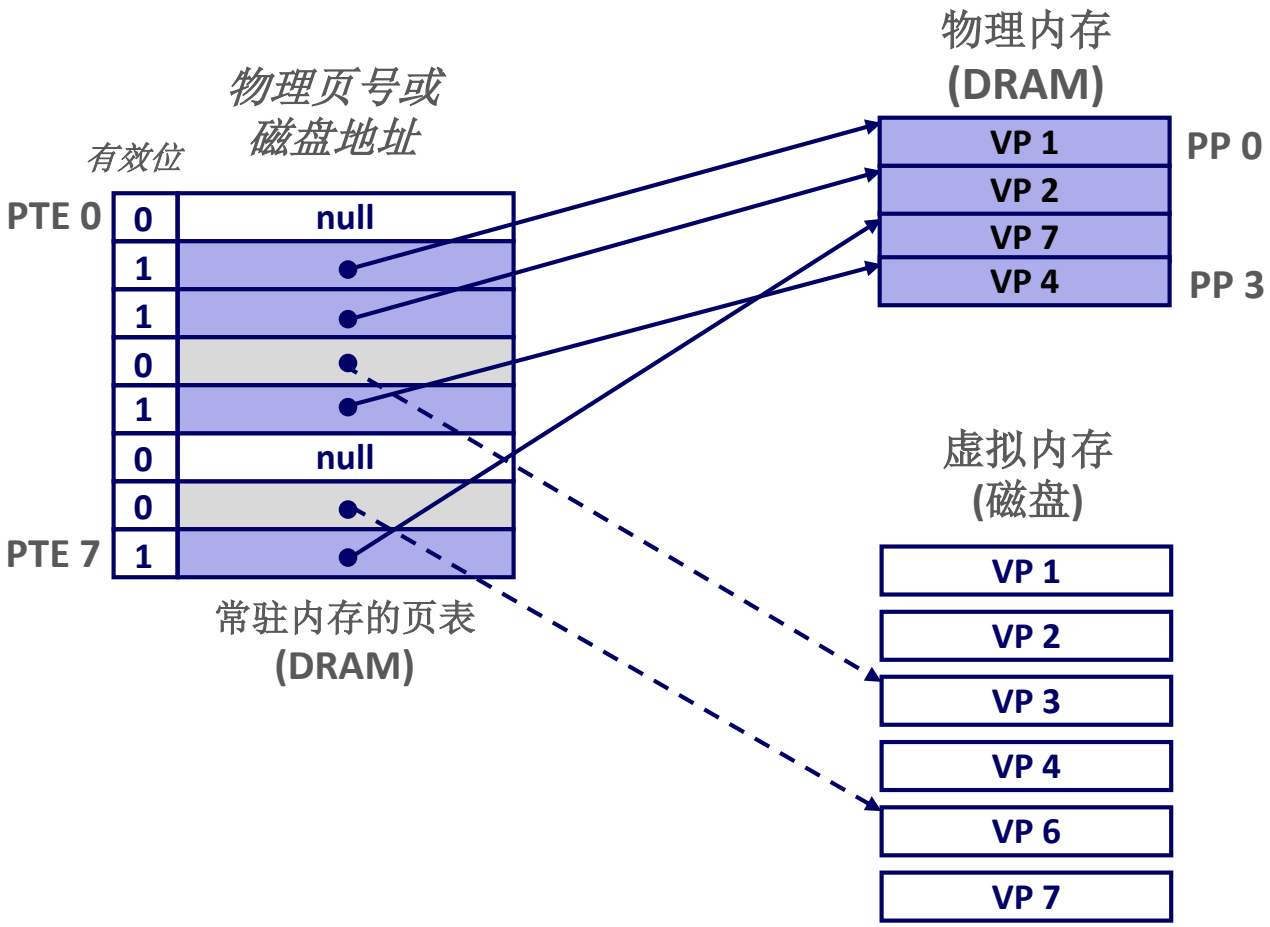


DRAM Cache Organization
DRAM缓存的组织结构

- **DRAM 缓存的组织结构完全是由巨大的不命中开销驱动的**
  - DRAM 比 SRAM 慢大约 *10倍*
  - 磁盘比 DRAM 慢大约 *10,000倍*
- **因此**
  - 虚拟页尺寸: 标准 4 KB, 有时可以达到 4 MB
  - DRAM缓存为全相联
    - 任何虚拟页都可以放置在任何物理页中
    - 需要一个更大的映射函数 – 不同于硬件对SRAM缓存
  - 更复杂精密的替换算法
    - 太复杂且无限制以致无法在硬件上实现
  - DRAM缓存总是使用写回，而不是直写

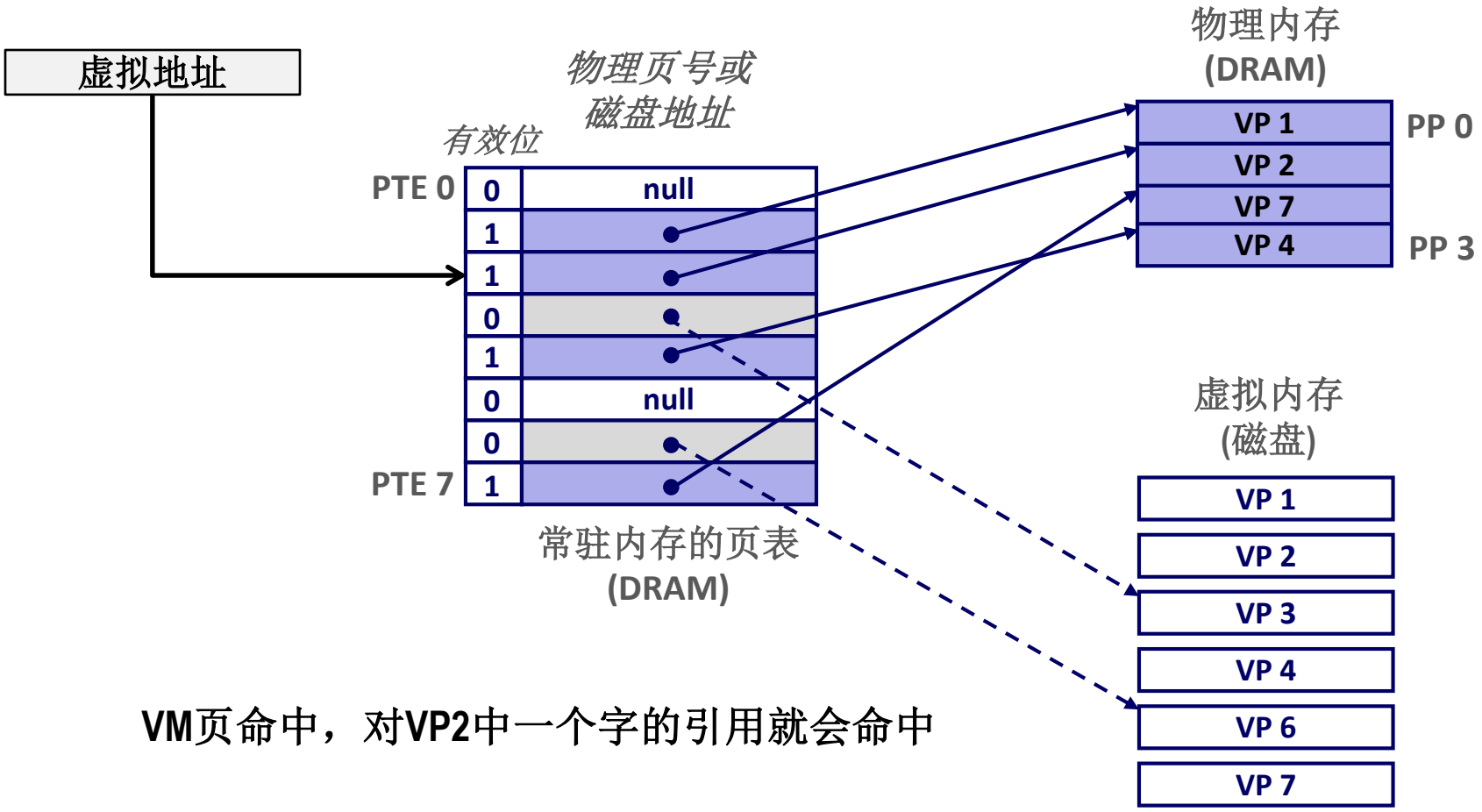
Enabling Data Structure: Page Table 页表

- *页表*是一个页表条目 (Page Table Entry, PTE)的数组，将虚拟页地址映射到物理页地址。
  - DRAM中的每个进程都使用的内核数据结构



Page Hit 页命中

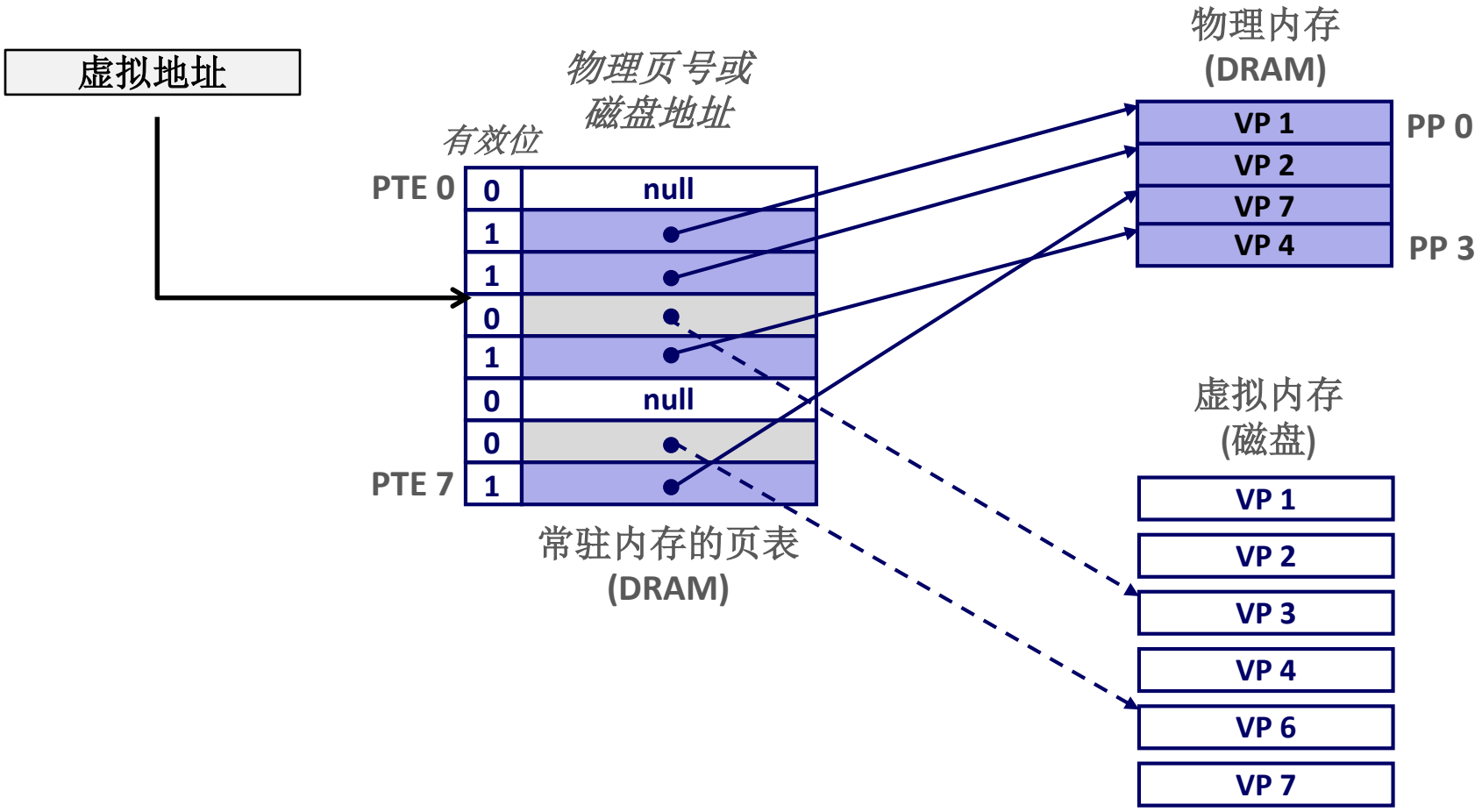
- ***Page hit页命中:*** 虚拟内存中的一个字存在于物理内存中，即(DRAM 缓存命中)



VM页命中，对VP2中一个字的引用就会命中

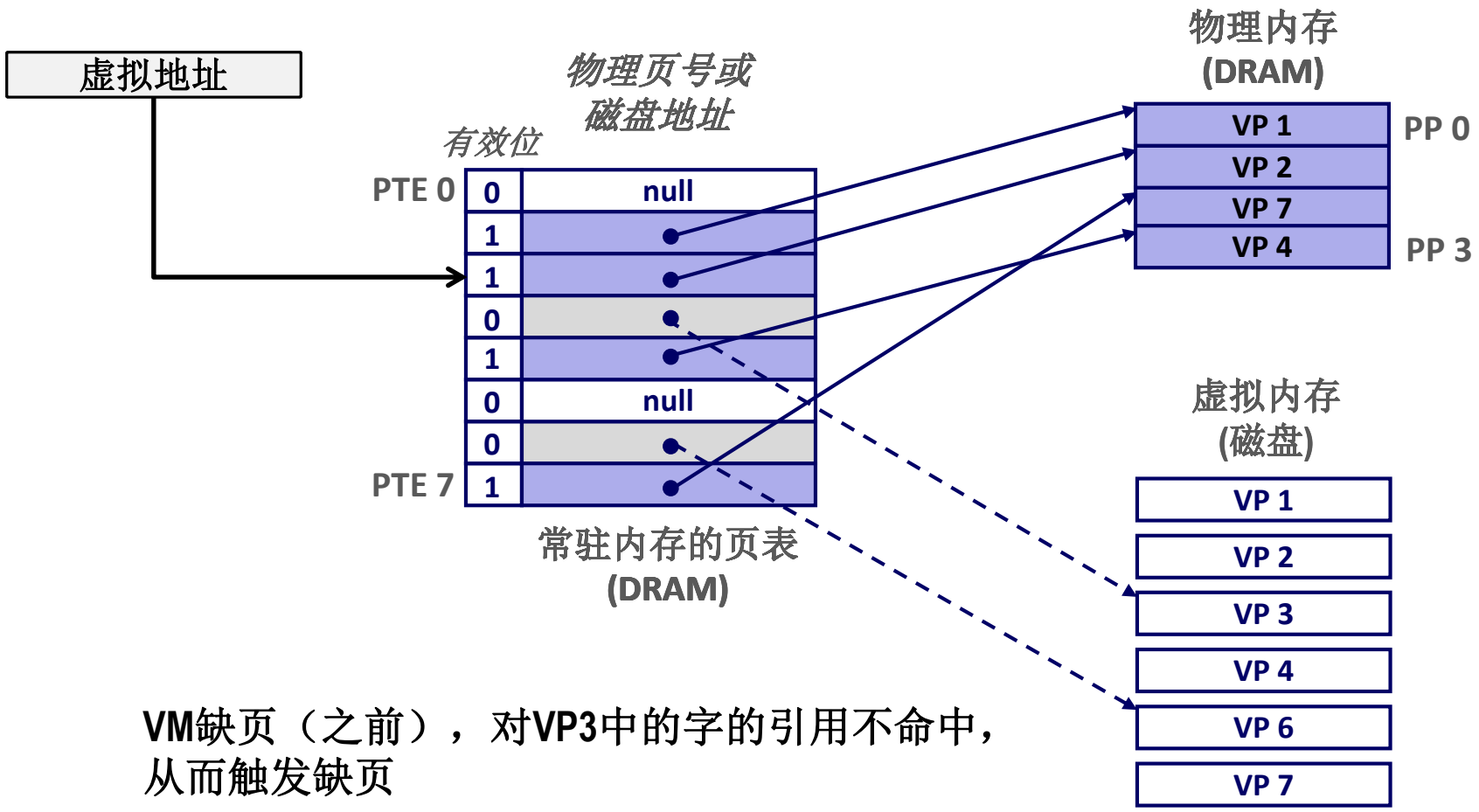
Page Fault 缺页

- ***Page fault缺页:*** 虚拟内存中的字不在物理内存中 (DRAM 缓存不命中)



Handling Page Fault 缺页的处理

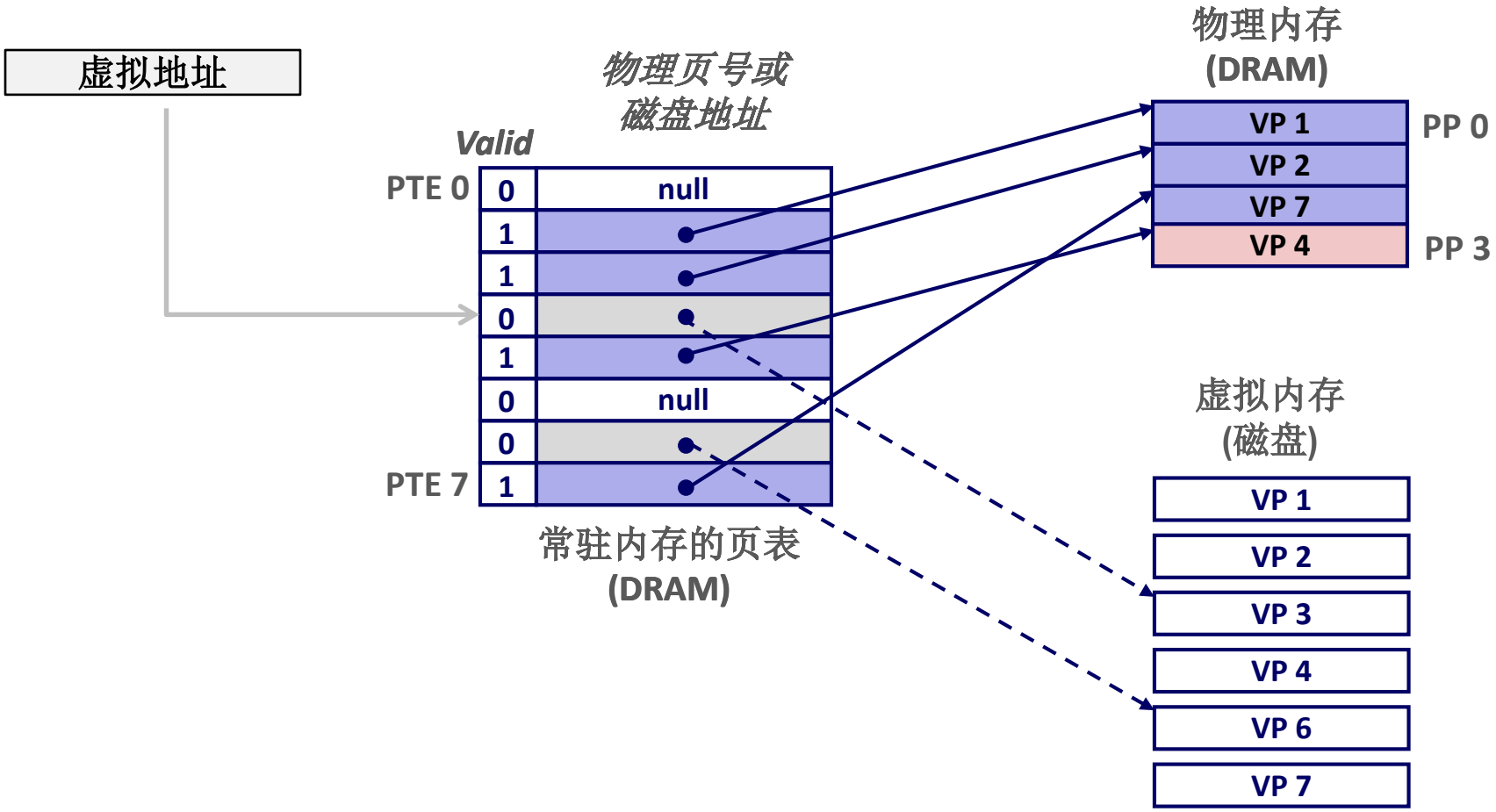
- Page miss causes page fault (an exception)
  缺页导致页面出错 (缺页异常)



VM缺页（之前），对VP3中的字的引用不命中，
从而触发缺页

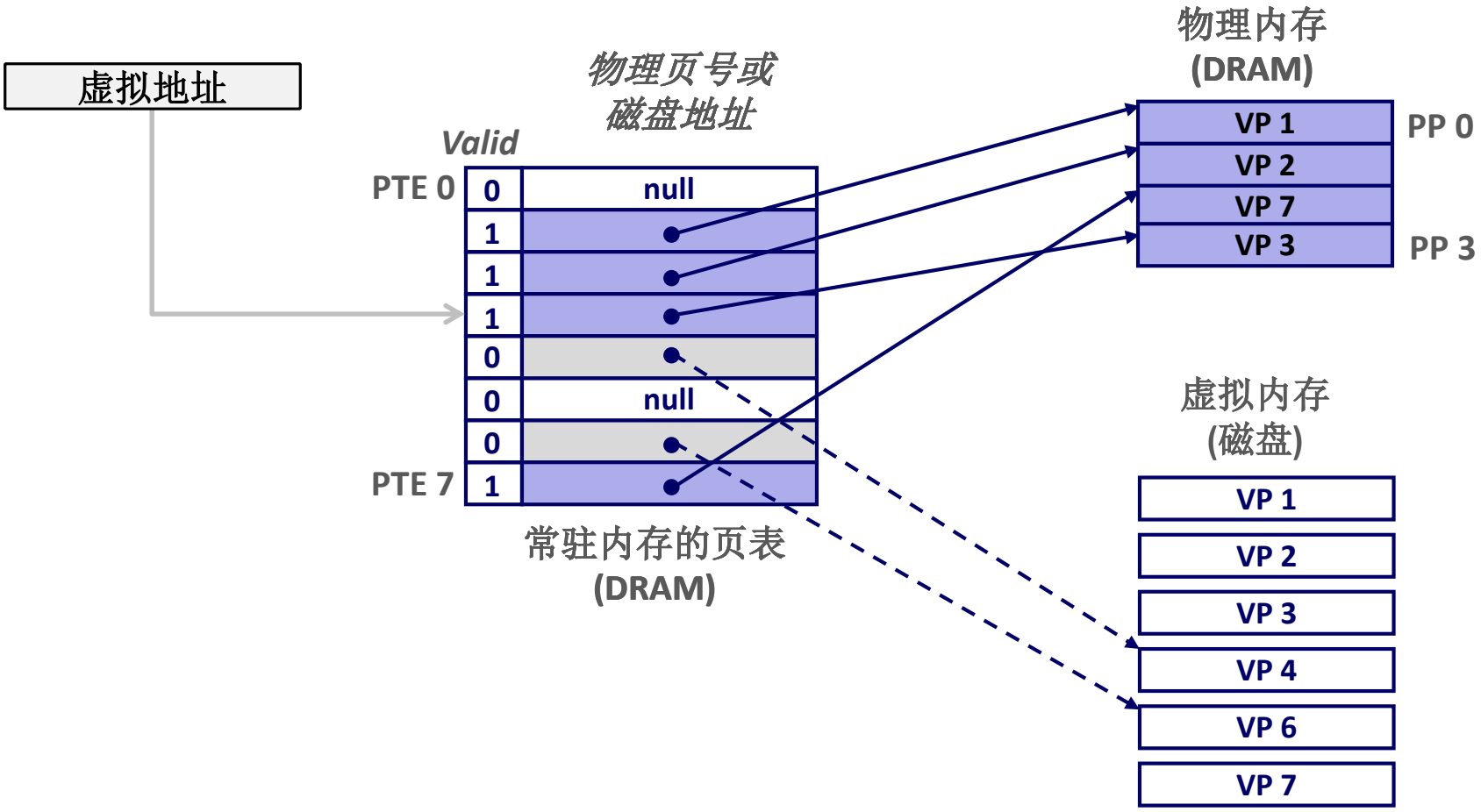
Handling Page Fault 缺页的处理

- 缺页导致页面出错 (缺页异常)
- 缺页异常处理程序选择一个牺牲页 (此例中就是 VP 4)



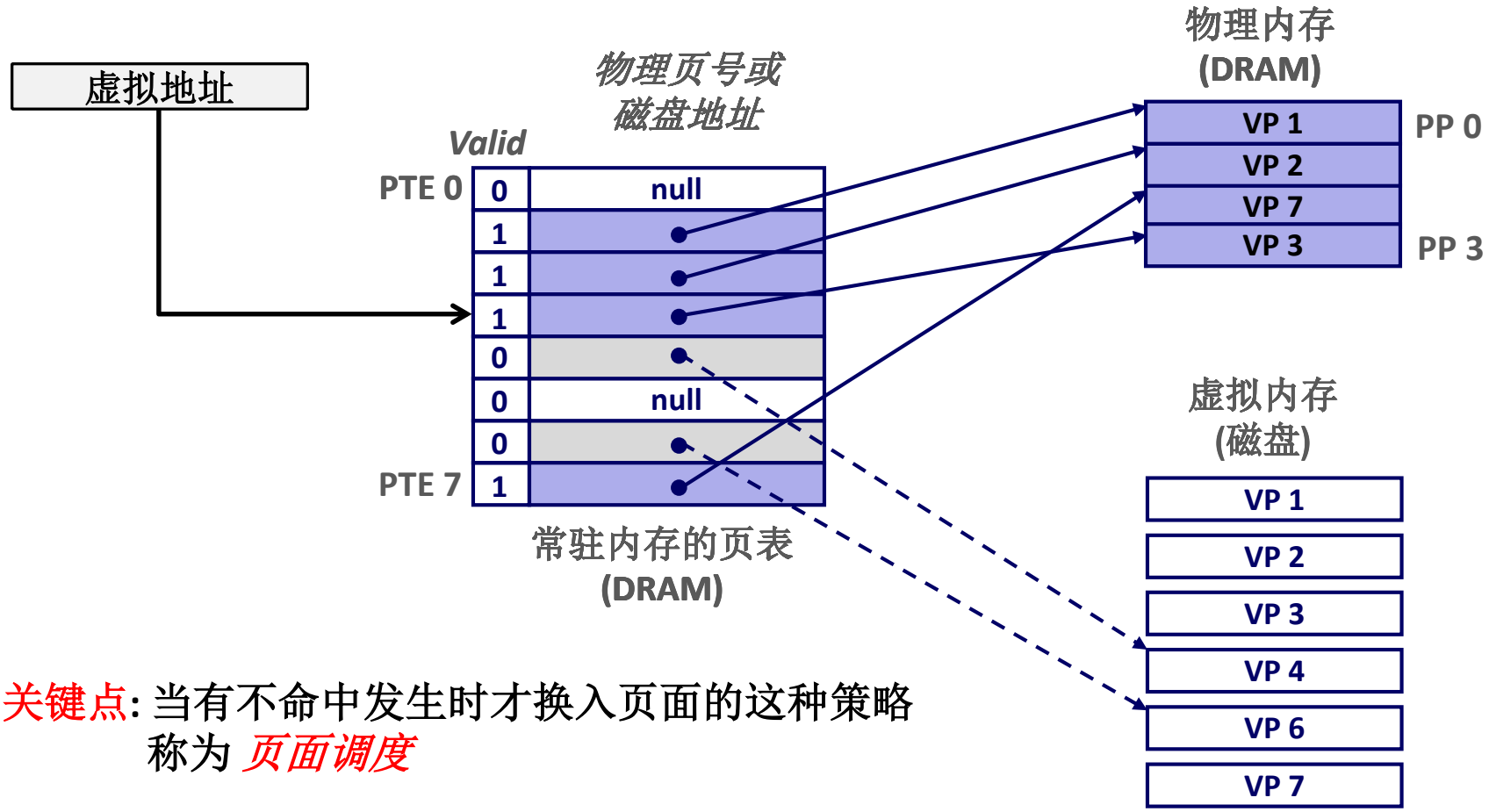
Handling Page Fault 缺页的处理

- 缺页导致页面出错 (缺页异常)
- 缺页异常处理程序选择一个牺牲页 (此例中就是 VP 4)



Handling Page Fault 缺页的处理

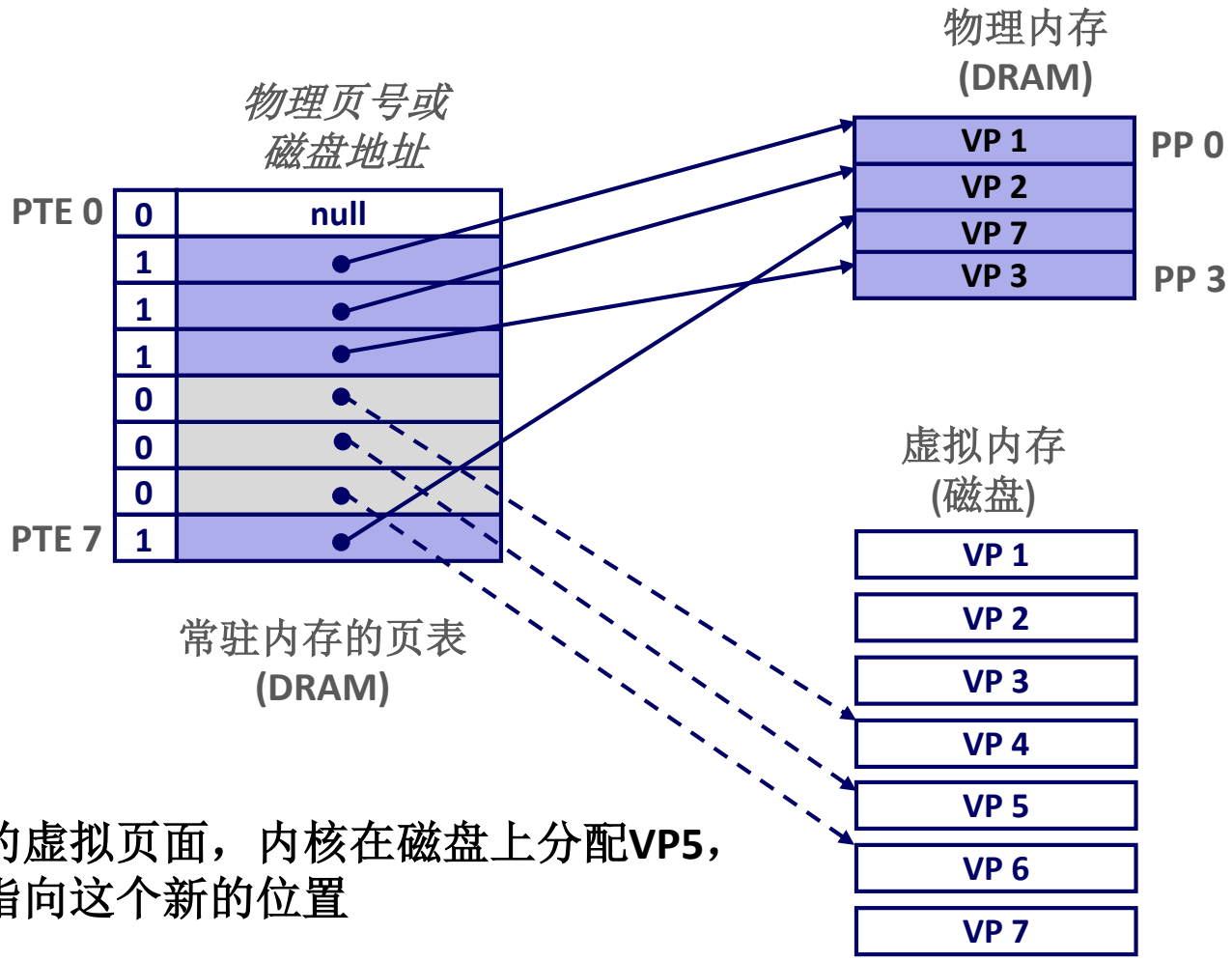
- 缺页导致页面出错 (缺页异常)
- 缺页异常处理程序选择一个牺牲页 (此例中就是 VP 4)
- 导致缺页的指令重新启动: 页面命中!



关键点: 当有不命中发生时才换入页面的这种策略称为 *页面调度*

Allocating Pages 分配页面

- 分配一个新的虚拟内存页 (VP 5).



分配一个新的虚拟页面，内核在磁盘上分配VP5，
并且将PTE5指向这个新的位置

Locality to the Rescue Again!

又是局部性救了我们!

- 虚拟内存看上去效率非常低, 但它工作得相当好，这都要归功于“局部性”.
- 在任意时间，程序将趋于在一个较小的活动页面集合上工作，这个集合叫做 *工作集 Working set*
  - 程序的时间局部性越好，工作集就会越小
- 如果 (工作集的大小 < 物理内存的大小)
  - 在初始开销后，对工作集的引用将导致命中。
- 如果 (工作集的大小) >物理内存的大小)
  - *Thrashing抖动:* 页面不断地换进换出，导致系统性能崩溃。

回顾页面置换算法

（1）FIFO页面置换

（2）OPT（最优）页面置换

（3）LRU页面置换

准确实现：计数器法、页码栈法

（4）近似LRU页面置换

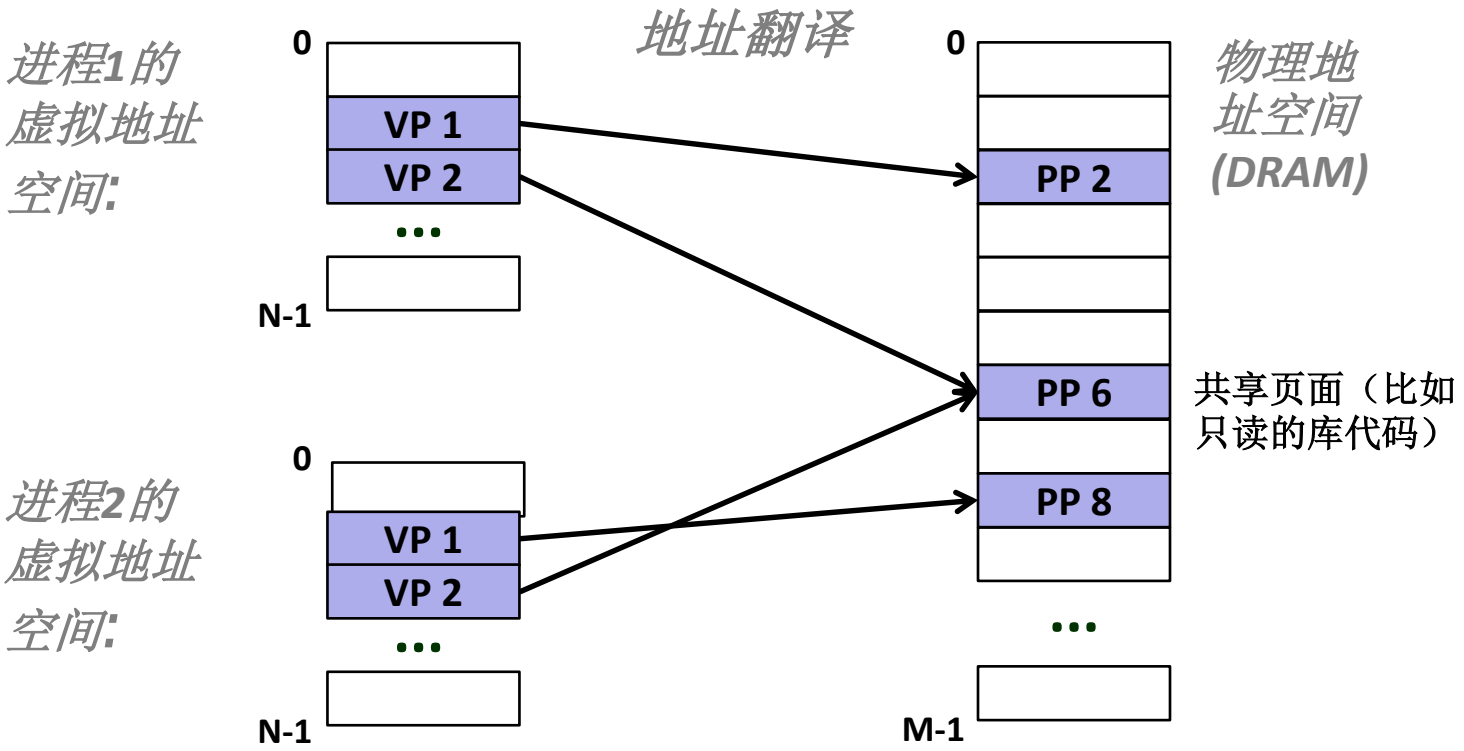
附加引用位法、时钟法

主要内容

- Address spaces 地址空间
- VM as a tool for caching 虚拟内存作为缓存的工具
- **VM as a tool for memory management**
  虚拟内存作为内存管理的工具
- VM as a tool for memory protection
  虚拟内存作为内存保护的工具
- Address translation地址翻译

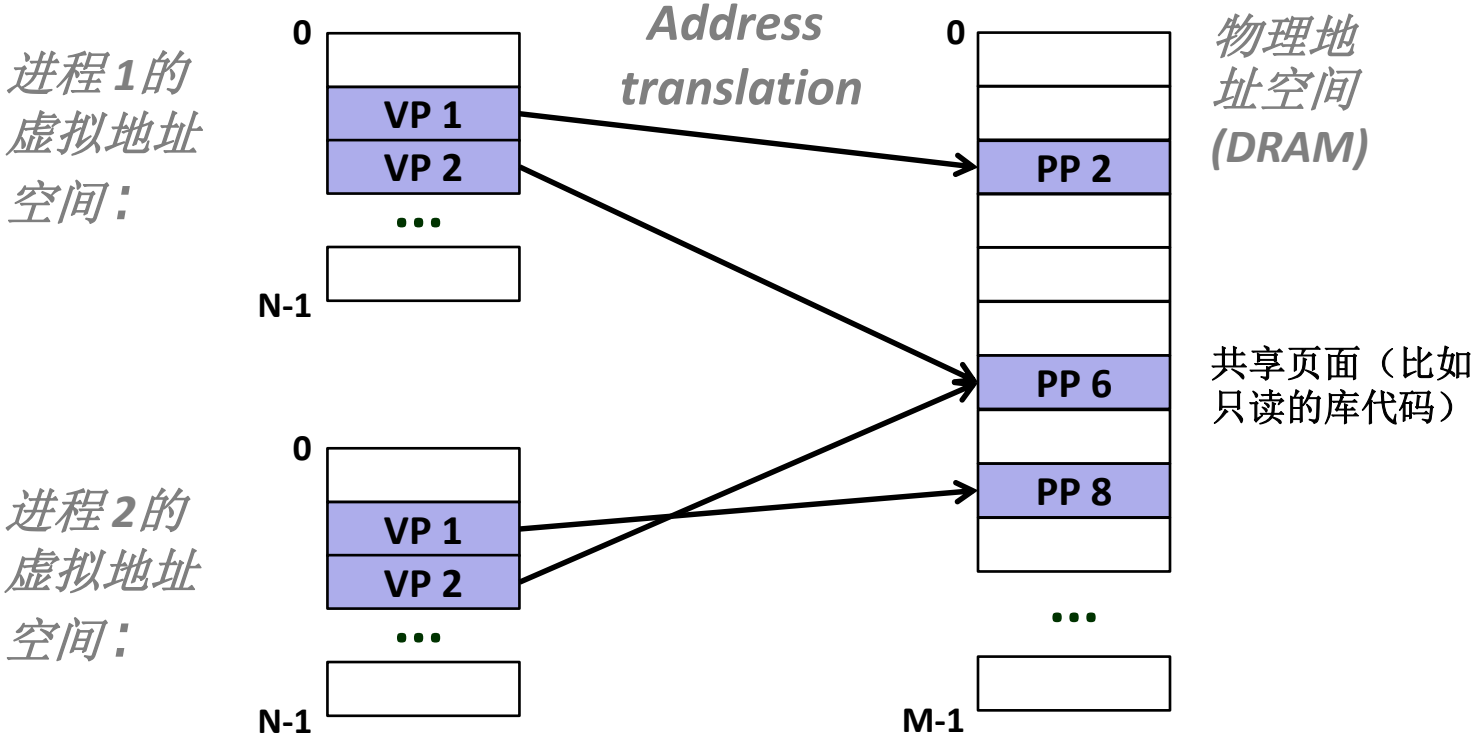
VM as a tool for memory management
虚拟内存作为内存管理的工具

- **Key idea核心观点: 每个进程都拥有一个独立的虚拟地址空间**
  - 把内存看作独立的简单线性数组
  - 映射函数通过物理内存来分散地址
    - 好的映射函数可以提高程序的局部性



VM as a tool for memory management
虚拟内存作为内存管理的工具

- **Simplifying memory allocation 简化内存分配**
  - 每个虚拟内存页面都要被映射到一个物理页面
  - 一个虚拟内存页面每次可以被分配到不同的物理页面
- **Sharing code and data among processes 简化代码和数据共享**
  - 不同的虚拟内存页面被映射到相同的物理页面 (此例中的 PP 6)



Simplifying Linking and Loading

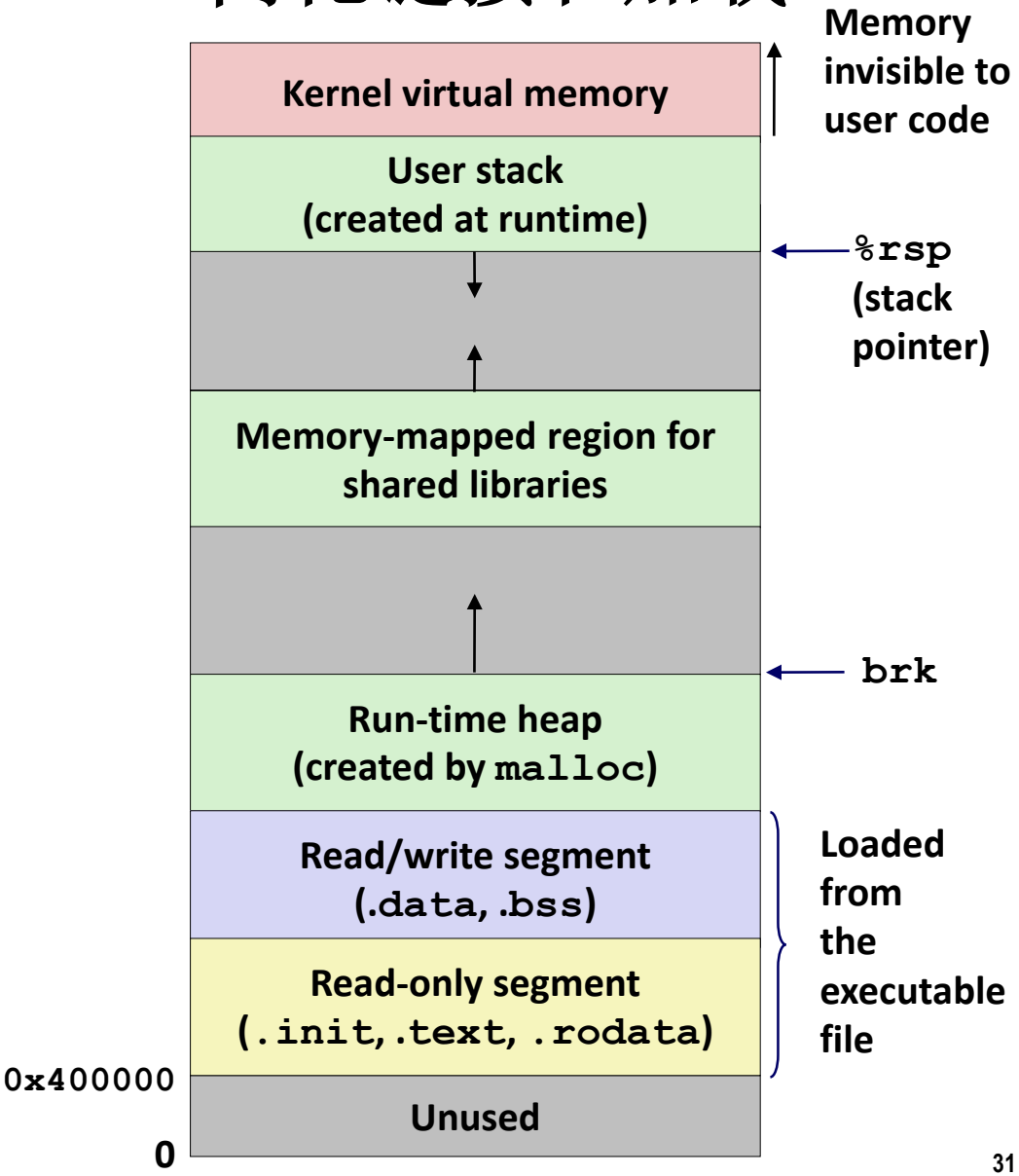
简化链接和加载

Linking 链接

- 每个程序使用相似的虚拟地址空间
- 代码、数据和堆都使用相同的起始地址.

Loading 加载

- `execve` 为代码段和数据段分配虚拟页，并标记为无效（即未被缓存）
- 每个页面被初次引用时，虚拟内存系统会按照需要自动地调入数据页。

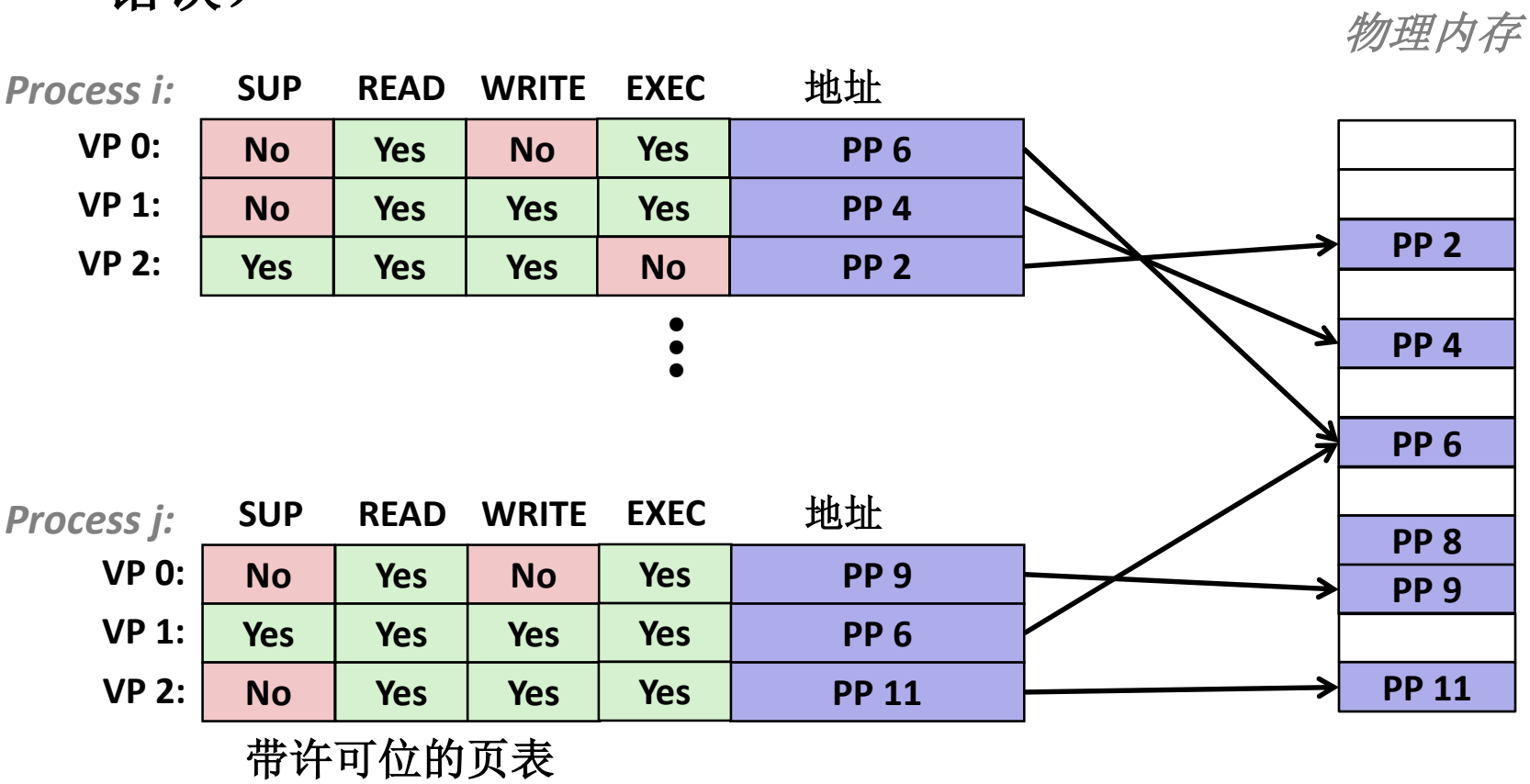


主要内容

- Address spaces 地址空间
- VM as a tool for caching 虚拟内存作为缓存的工具
- VM as a tool for memory management
  虚拟内存作为内存管理的工具
- **VM as a tool for memory protection**
  虚拟内存作为内存保护的工具
- Address translation地址翻译

VM as a Tool for Memory Protection
虚拟内存作为内存保护的工具

- 在 PTE 上扩展许可位以提供更好的访问控制
- 内存管理单元（MMU）每次访问数据都要检查许可位（段错误）



带许可位的页表

主要内容

- Address spaces 地址空间
- VM as a tool for caching 虚拟内存作为缓存的工具
- VM as a tool for memory management
  虚拟内存作为内存管理的工具
- VM as a tool for memory protection
  虚拟内存作为内存保护的工具
- **Address translation地址翻译**

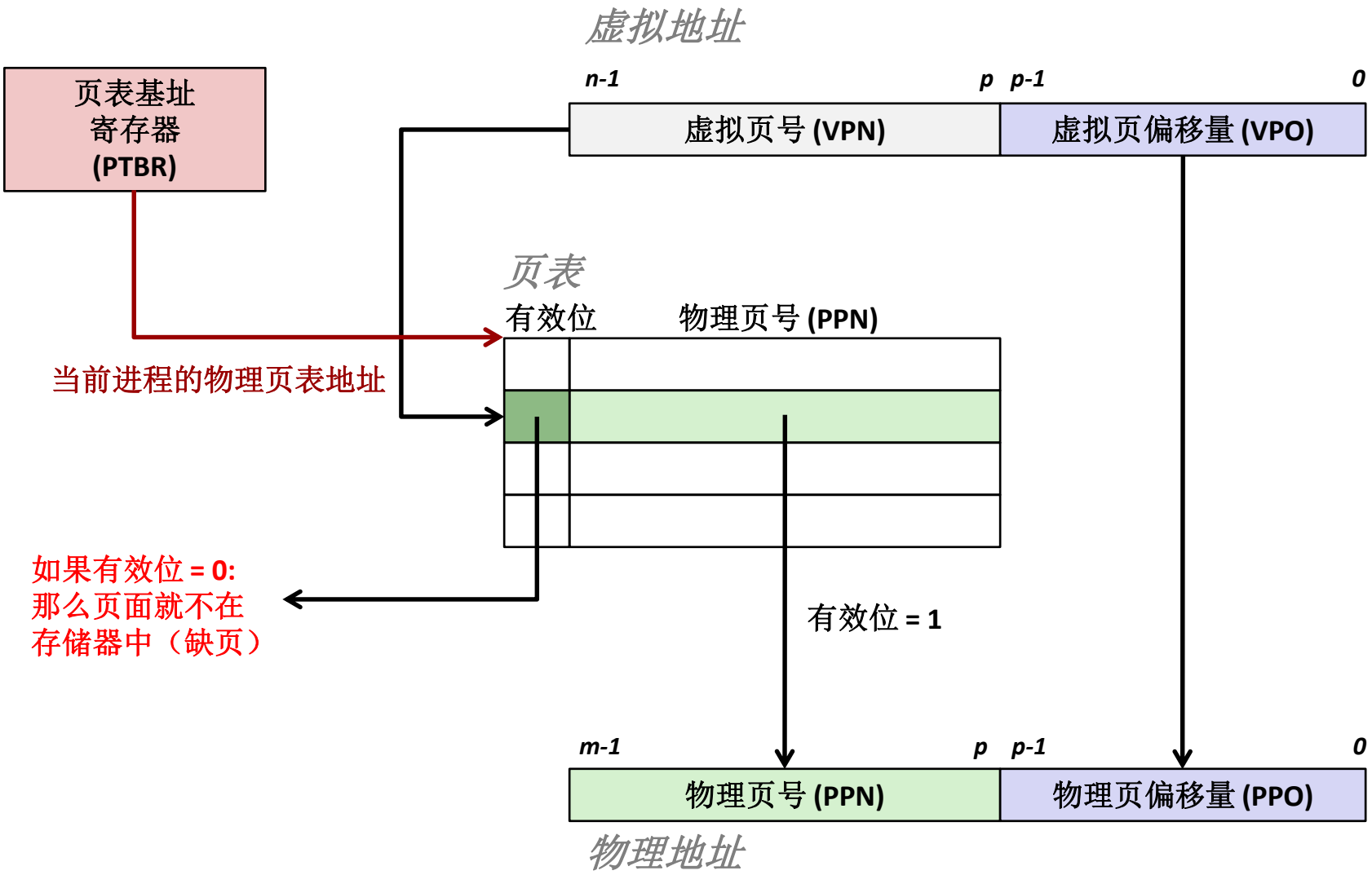
VM Address Translation 地址翻译

- **Virtual Address Space 虚拟地址空间**
  - $V = \{0, 1, \ldots, N-1\}$
- **Physical Address Space 物理地址空间**
  - $P = \{0, 1, \ldots, M-1\}$
- **Address Translation 地址翻译**
  - $MAP: V \rightarrow P \ U \ \{\varnothing\}$
  - For virtual address *a*:
    - *MAP(a) = a'* 如果虚拟地址 *a*处的数据在*p*的物理地址 *a'* 处
    - *MAP(a)* = ∅ 如果虚拟地址 *a*处的数据不在物理内存中
      - 不论无效地址还是存储在磁盘上

地址翻译使用到的所有符号

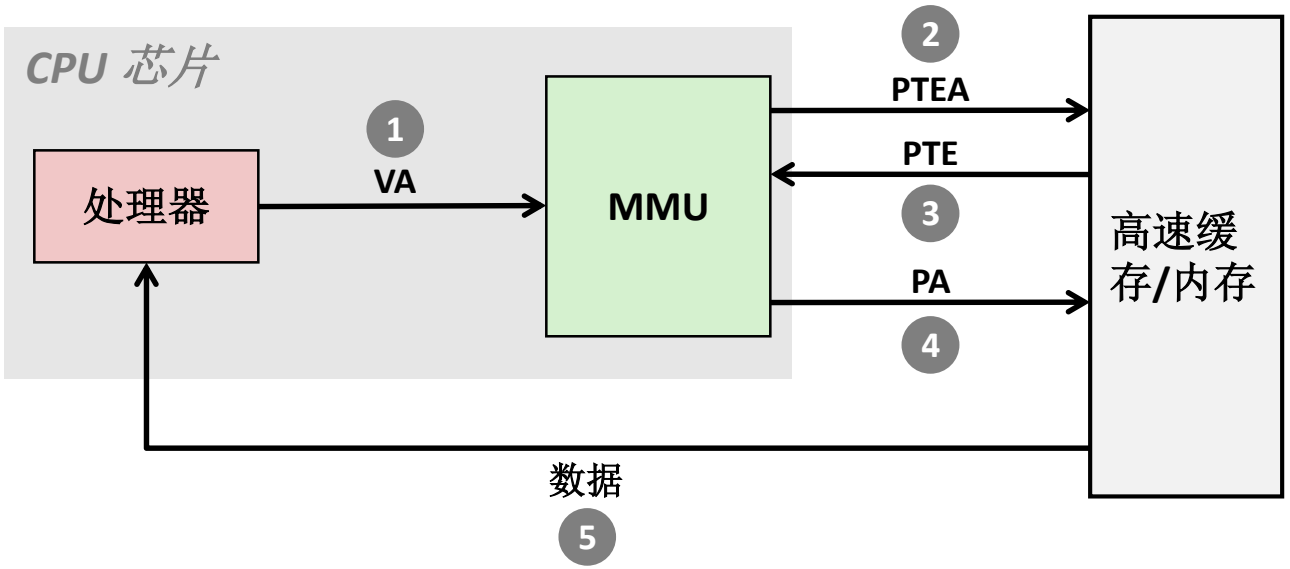
- **Basic Parameters 基本参数**
  - $N = 2^n$: 虚拟地址空间中的地址数量
  - $M = 2^m$: 物理地址空间中的地址数量
  - $P = 2^p$ ：页的大小 (bytes)
- **Components of the virtual address (VA) 虚拟地址组成部分**
  - TLBI: TLB index----TLB索引
  - TLBT: TLB tag----TLB标记
  - VPO: Virtual page offset----虚拟页面偏移量（字节）
  - VPN: Virtual page number----虚拟页号
- **Components of the physical address (PA)物理地址组成部分**
  - PPO: Physical page offset (same as VPO)----物理页面偏移量
  - PPN: Physical page number----物理页号

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



Address Translation: Page Hit

地址翻译：页面命中



1) 处理器生成一个虚拟地址，并将其传送给MMU（Memory Management Unit）

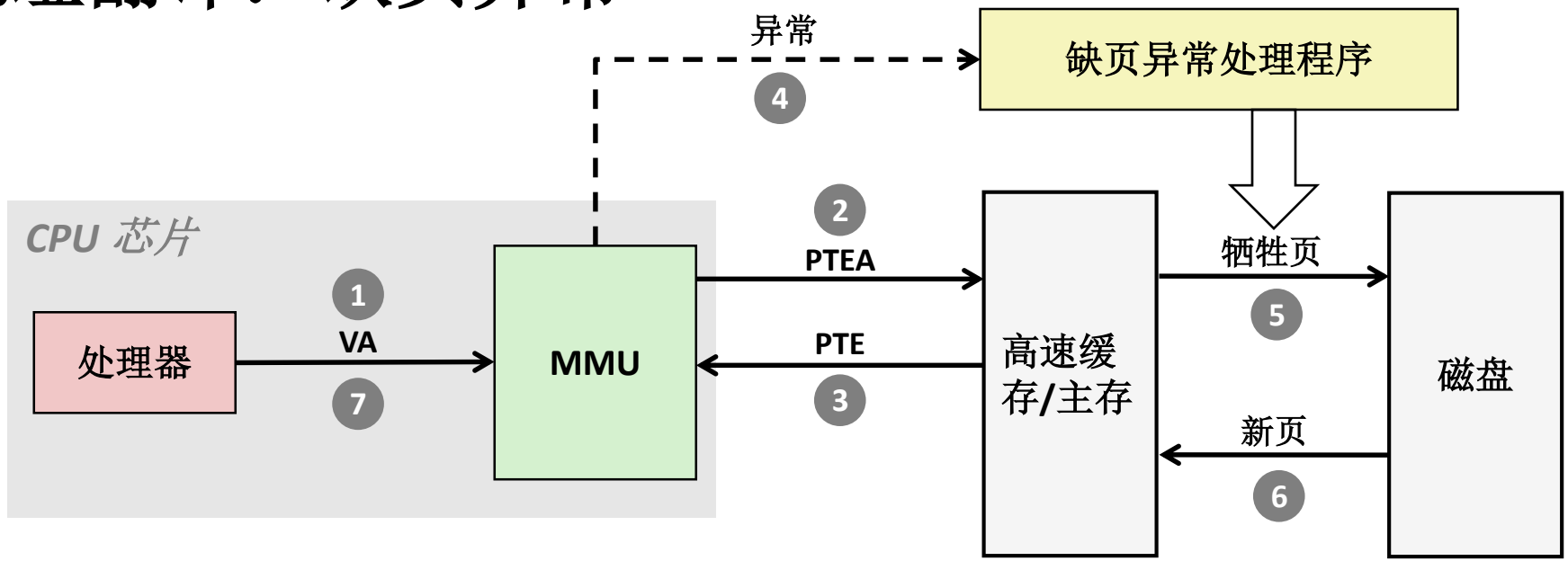
2-3) MMU 使用内存中的页表生成PTE地址

4) MMU 将物理地址传送给高速缓存/主存

5) 高速缓存/主存返回所请求的数据字给处理器

PTEA: page table entry adress
PTE: page table entry
PA: physical adress

Address Translation: Page Fault
地址翻译：缺页异常



1) 处理器将虚拟地址发送给 MMU

2-3) MMU 使用内存中的页表生成PTE地址

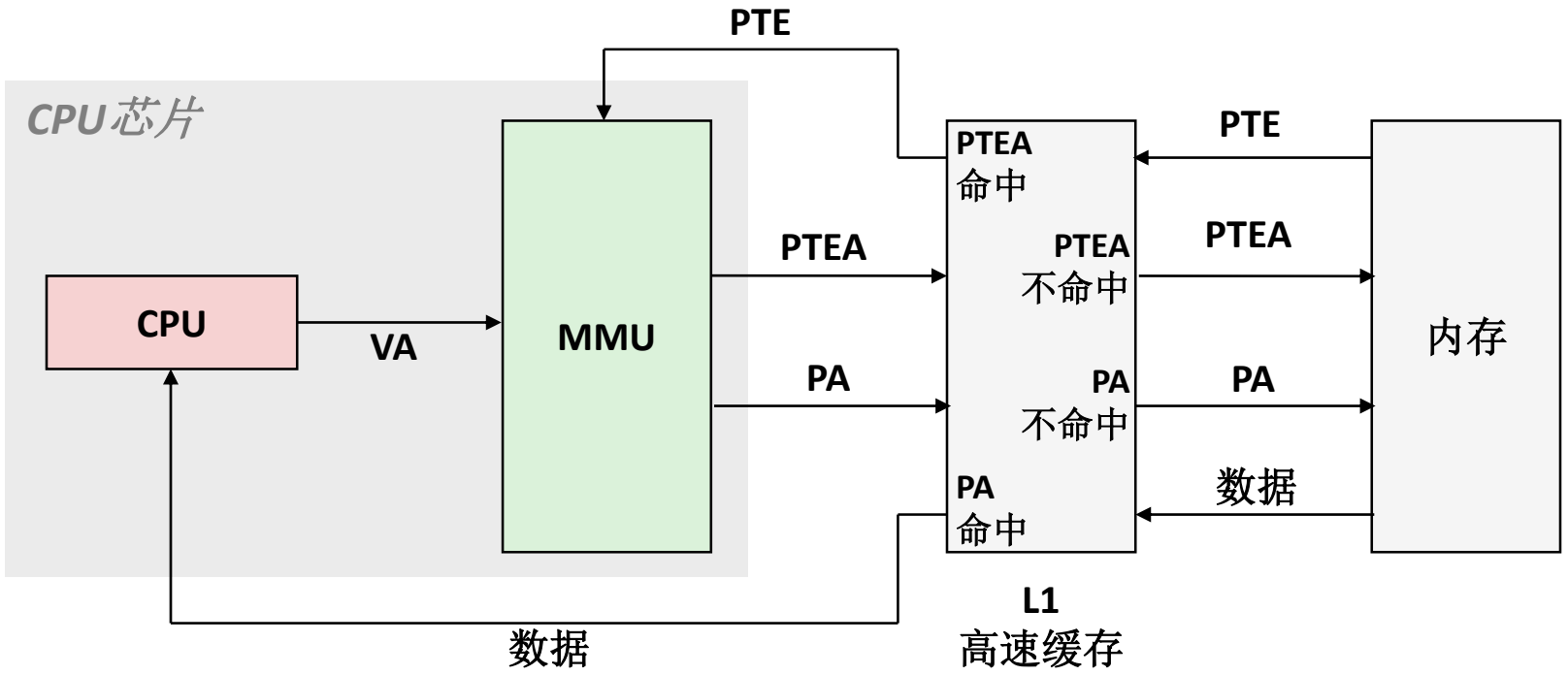
4) 有效位为零, 因此 MMU 触发缺页异常

5) 缺页处理程序确定物理内存中替换页 (若页面被修改，则换出到磁盘)

6) 缺页处理程序调入新的页面，并更新内存中的PTE

7) 缺页处理程序返回到原来进程，再次执行缺页的指令

Integrating VM and Cache
结合高速缓存和虚拟内存



VA: virtual address虚拟地址, PA: physical address物理地址,

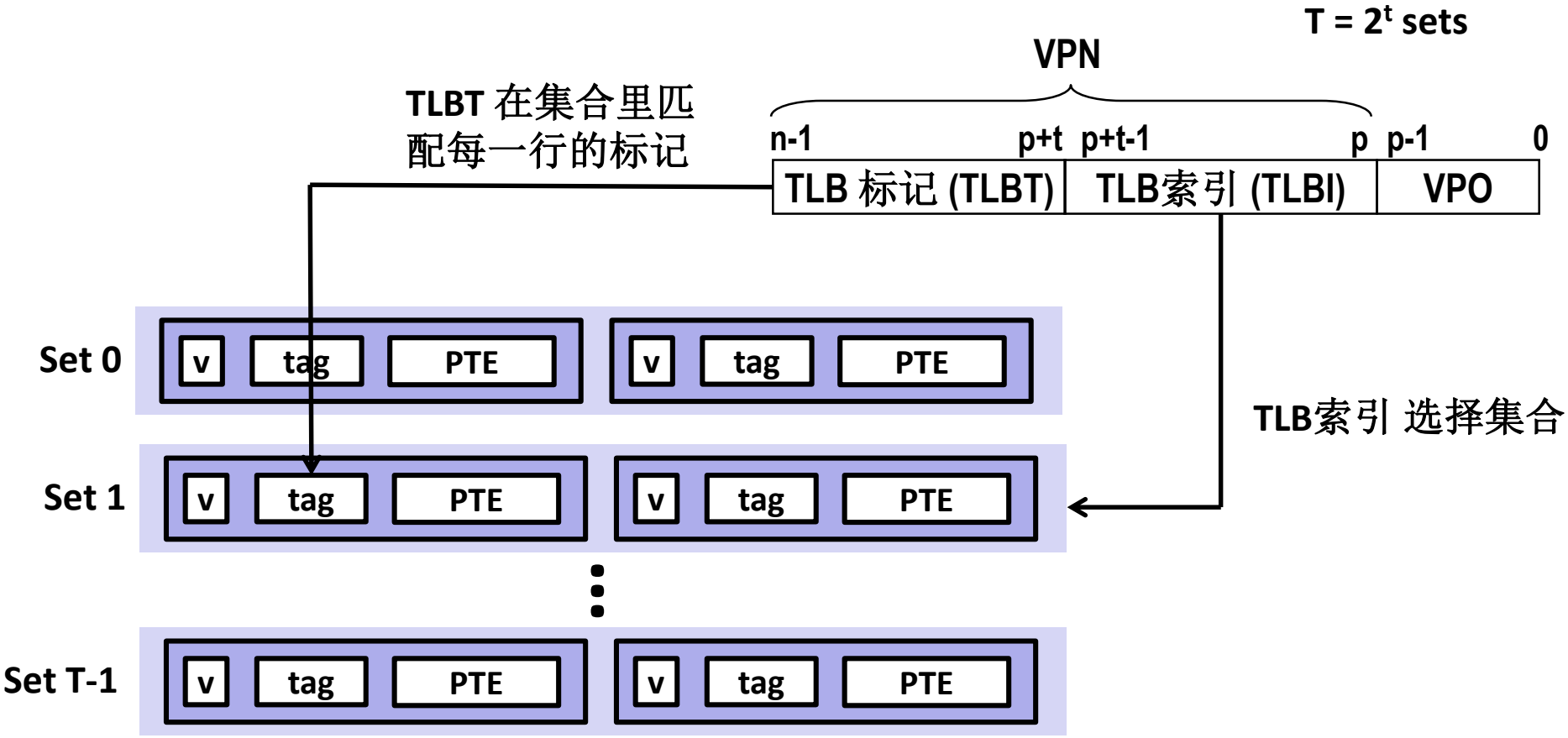
PTE:page table entry页表条目, PTEA = PTE address页表条目地址

Speeding up Translation with a TLB
利用TLB加速地址翻译

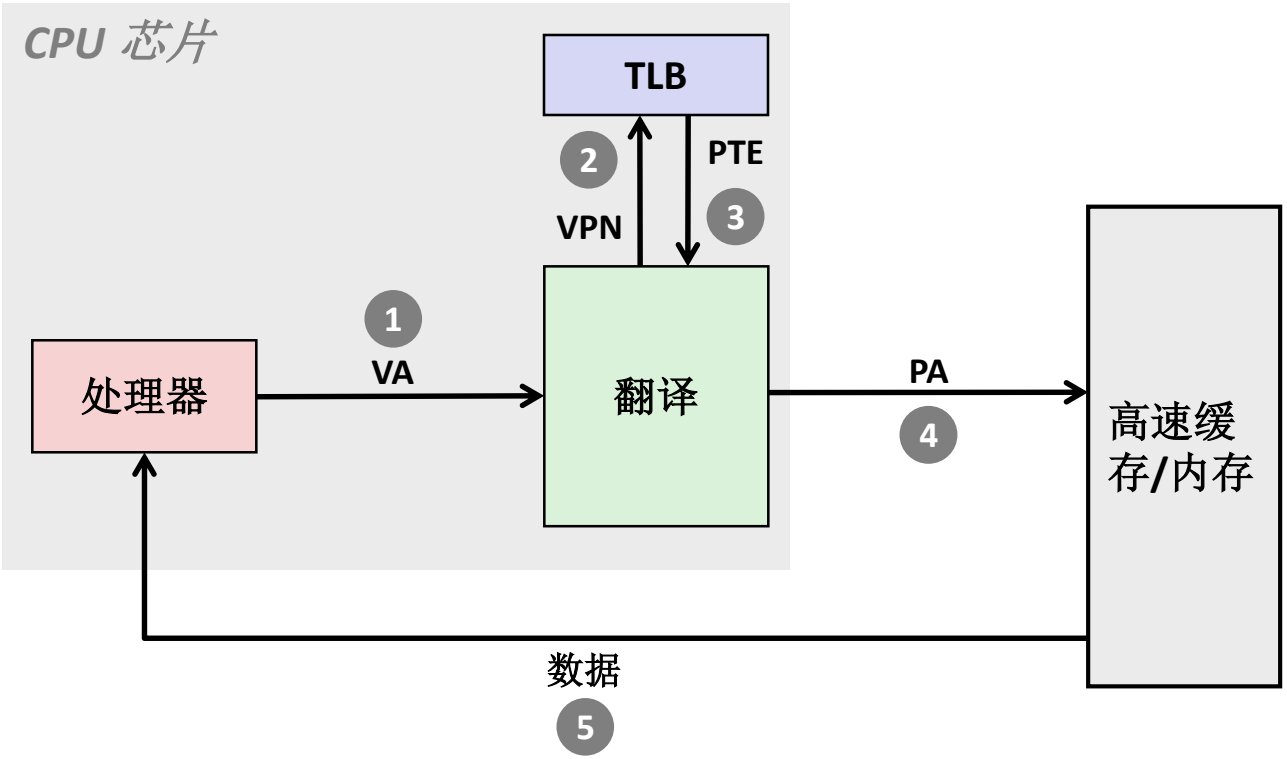
- 页表条目 (PTEs) 恰巧缓存在 L1
  - PTE 可能被其他数据引用所驱逐
  - PTE 命中仍然需要1-2周期的延迟
- 解决办法: ***Translation Lookaside Buffer*** (TLB)翻译后备缓冲器
  - MMU中一个小的具有高相联度的集合
  - 实现虚拟页码向物理页码的映射
  - 对于页表项很少的页表可以完全包含在TLB中

Accessing the TLB 访问TLB（组相联例子）

- MMU 使用虚拟地址的 VPN 部分来访问TLB:

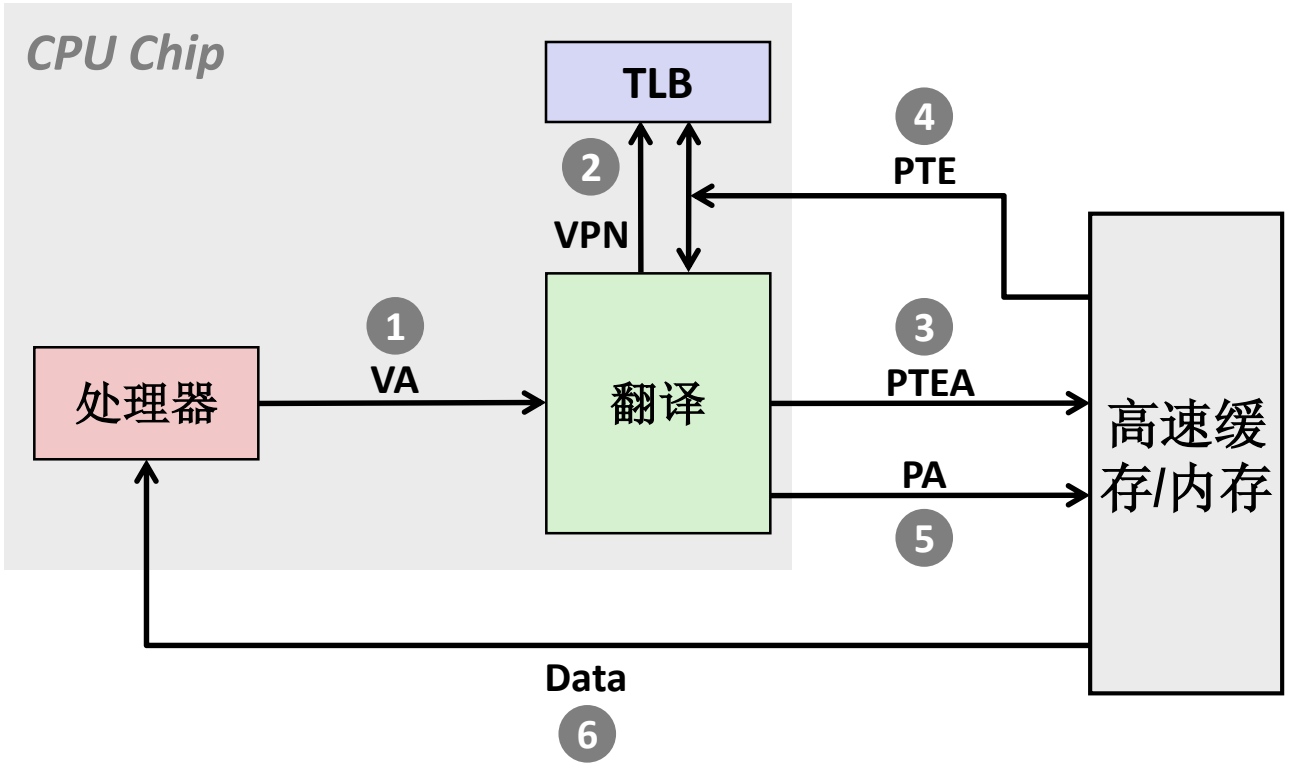


TLB Hit TLB命中



TLB 命中减少内存访问

TLB 不命中



TLB 不命中引发了额外的内存访问

万幸的是, TLB 不命中很少发生。这是为什么呢？--局部性

TLB得以发挥作用分析

- TLB命中时效率会很高，未命中效率会降低，平均后仍表现良好。 用数字来说明:

有效访问时间 = HitR×(TLB+MA) + (1-HitR)×(TLB+2MA)

命中率!

内存访问时间!
假设100ns

TLB时间!
假设20ns

有效访问时间=80%×(20ns+100ns) + 20%×(20ns+200ns)=140ns

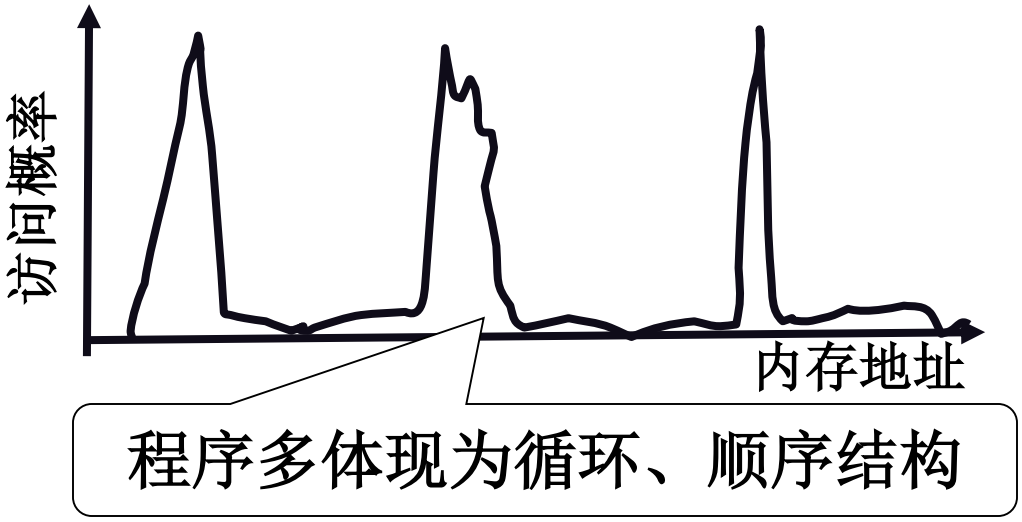
有效访问时间=98%×(20ns+100ns) + 2%×(20ns+200ns)=122ns

平均加快了13%!

- TLB要想发挥作用，命中率应尽量高
- TLB越大越好，但TLB价格昂贵，通常TLB条数在 [64, 1024] 范围

为什么TLB条目数在64-1024之间?

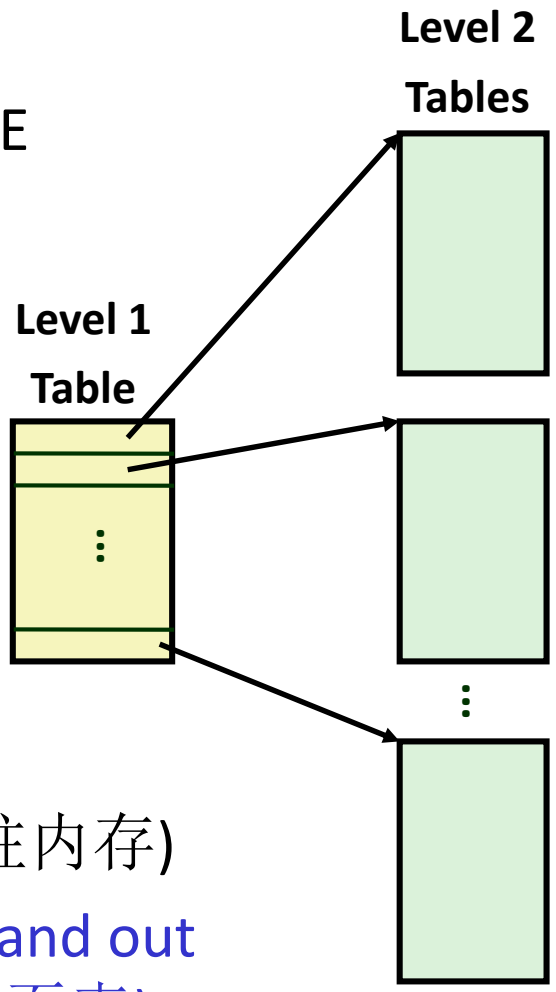
- 相比2^{20}个页，64很小，为什么TLB就能起作用?
  - 程序的地址访问存在局部性
  - 空间局部性(Locality in Space)



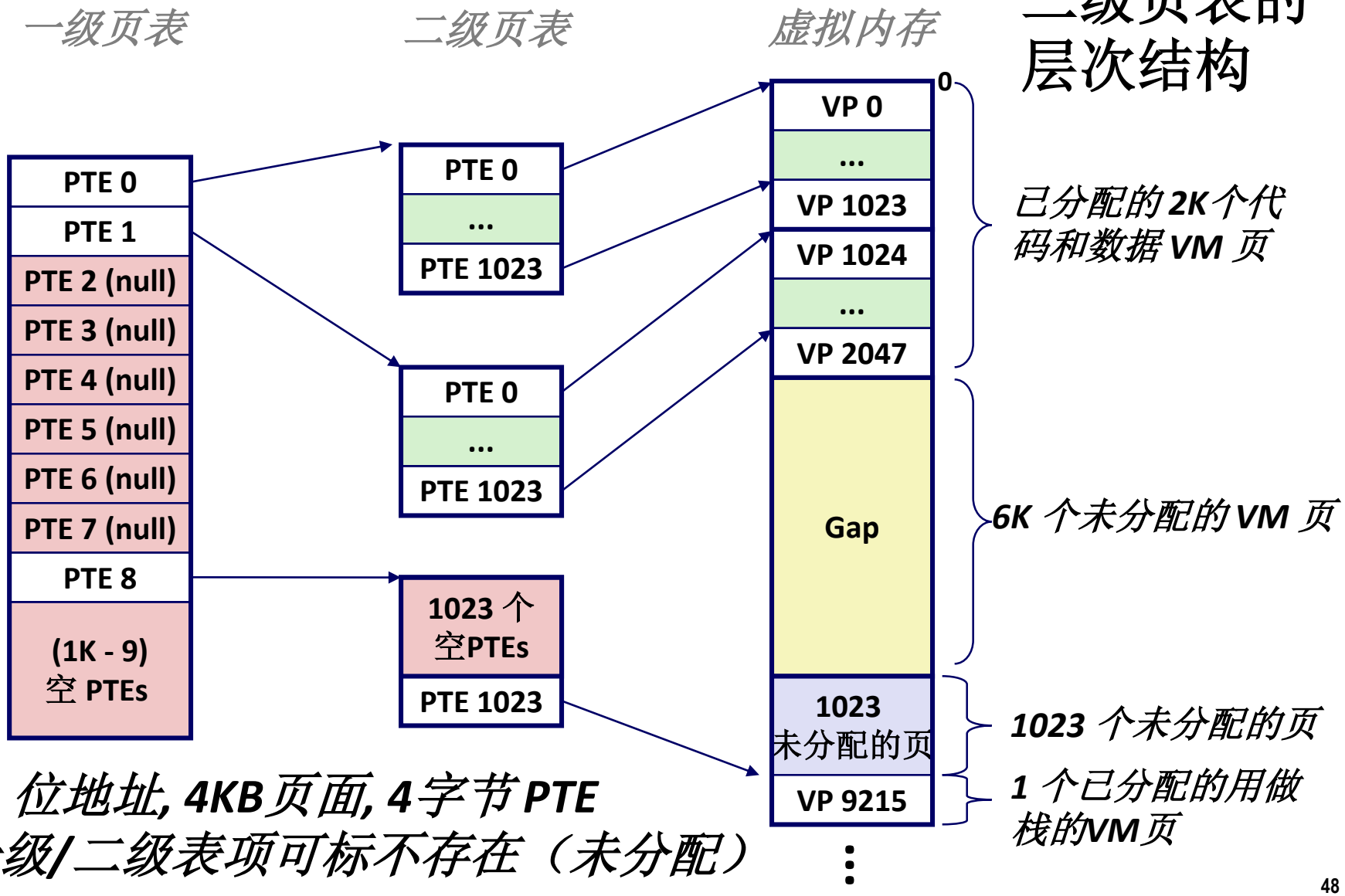
局部性又是计算机
的一个基本特征

Multi-Level Page Tables 多级页表

- 假设:
  - 4KB (2^{12}) 页面, 48位地址空间, 8字节 PTE
- 问题:
  - 耗费页表项大小 512GB!
    - $2^{48} * 2^{-12}$个页表项
    - $2^{48} * 2^{-12} * 2^{3} = 2^{39}$ bytes
- 常用解决办法: 多级页表
- 以二级页表为例:
  - 一级页表: 每个 PTE 指向一个页表 (常驻内存)
  - 二级页表: 每个 PTE 指向一页(paged in and out like any other data页面可以调入或调出页表)



A Two-Level Page Table Hierarchy

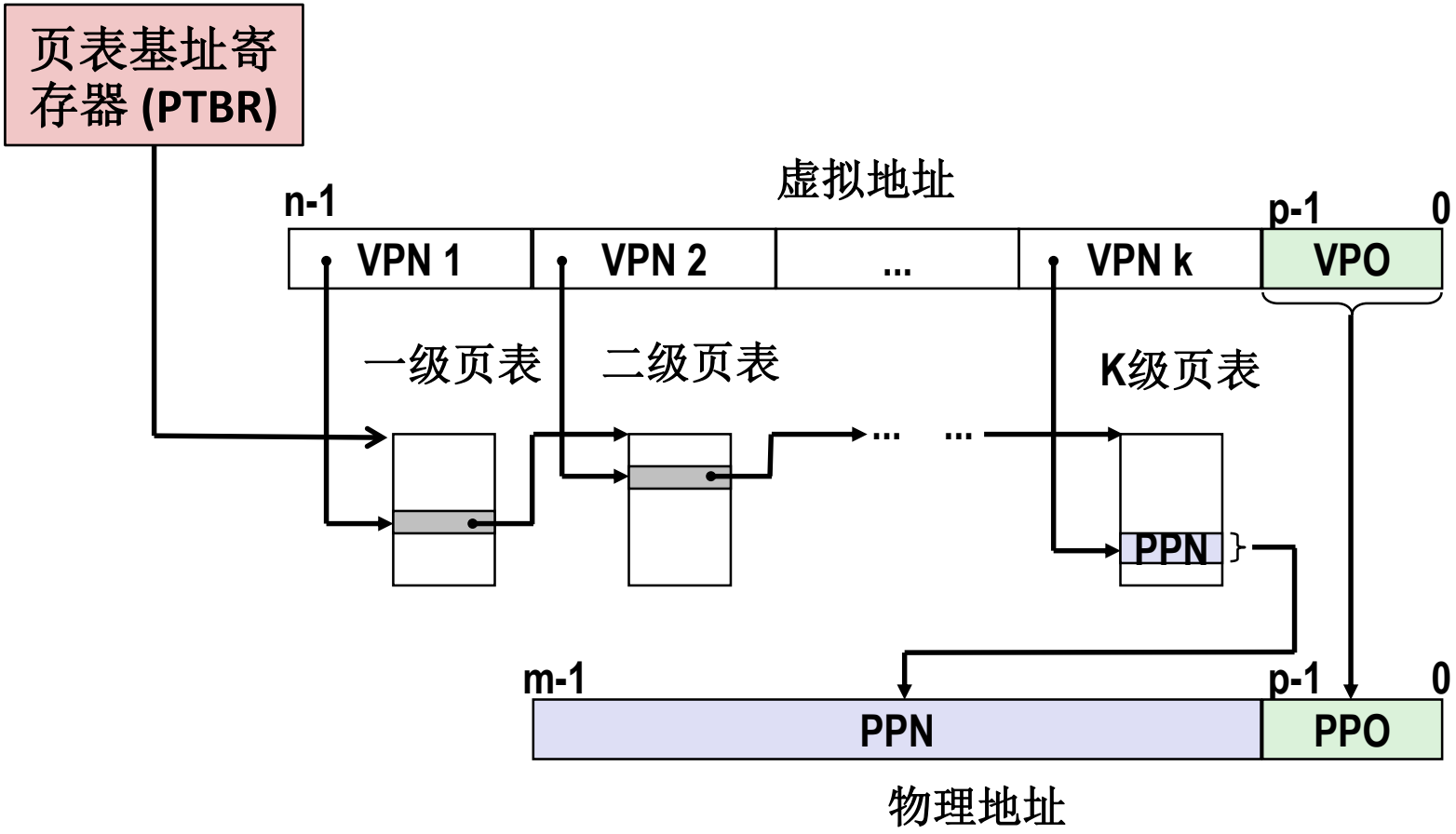


32 位地址, 4KB页面, 4字节 PTE

一级/二级表项可标不存在（未分配）

Translating with a k-level Page Table

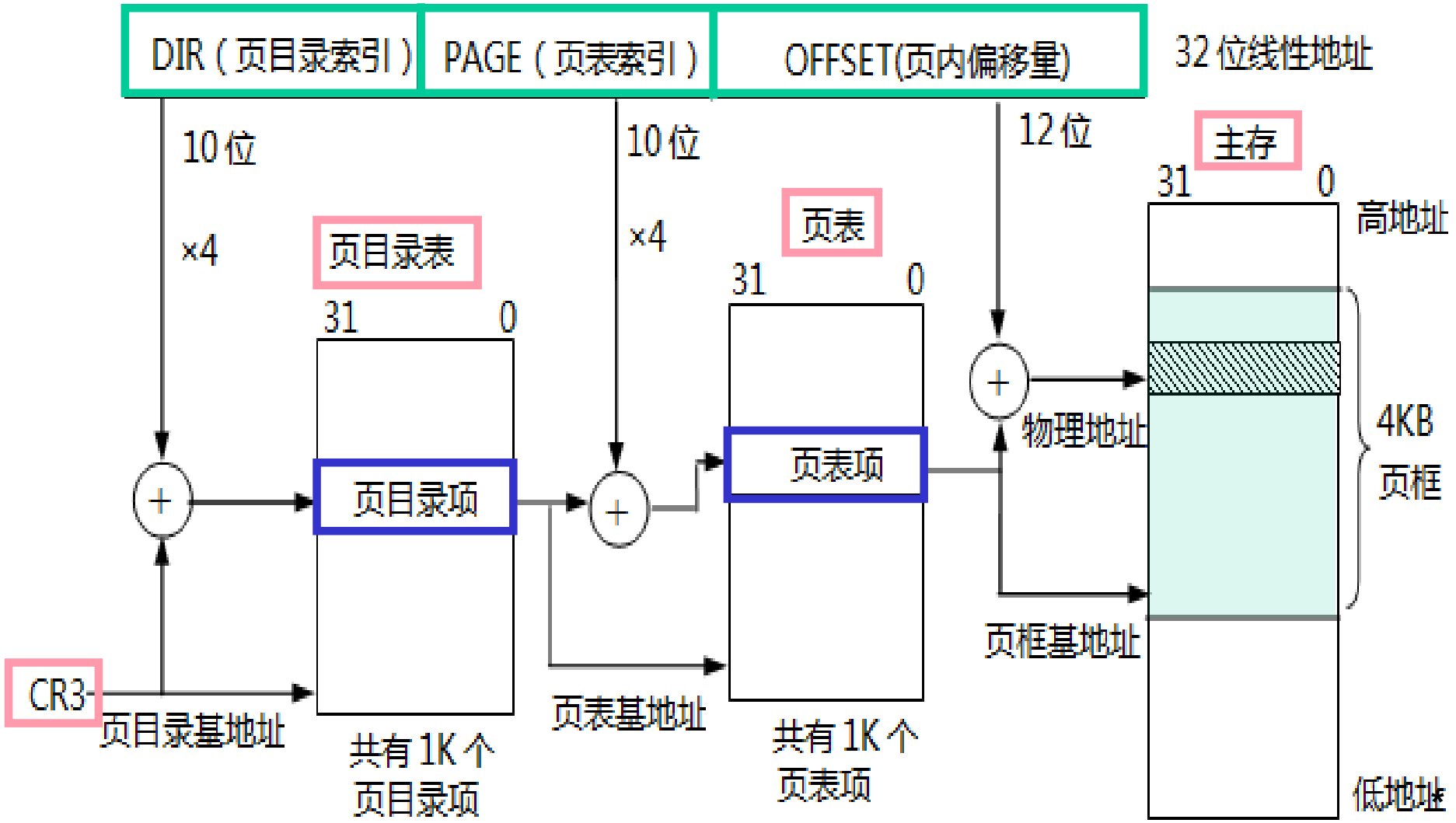
使用K级页表的地址翻译



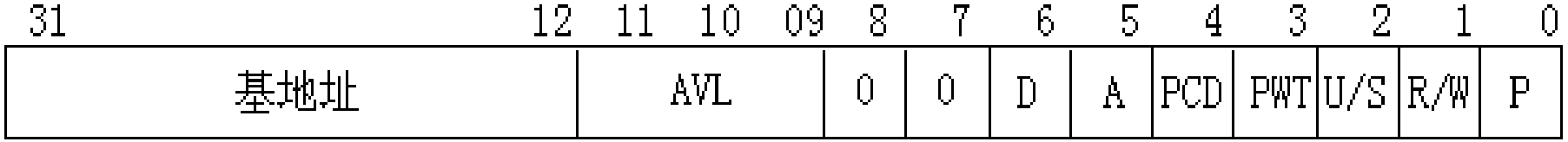
IA32线性地址向物理地址转换

线性地址空间划分：4GB=1K个子空间 * 1K个页面/子空间 * 4KB/页

- **页目录项和页表项格式一样，都是32位（4B）**



IA-32的页目录项和页表项



- P：1表示页表或页在主存中；P=0表示页表或页不在主存，即缺页，此时需将页故障线性地址保存到CR2。
- R/W：0表示页表或页只能读不能写；1表示可读可写。
- U/S：0表示用户进程不能访问；1表示允许访问。
- PWT：控制页表或页的cache写策略是全写还是回写（Write Back）。
- PCD：控制页表或页能否被缓存到cache中。
- A：1表示指定页表或页被访问过，初始化时OS将其清0。利用该标志，OS可清楚了解哪些页表或页正在使用，一般选择长期未用的页或近来最少使用的页调出主存。由MMU在进行地址转换时将该位置1。
- D：修改位(脏位dirty bit)。页目录项中无意义，只在页表项中有意义。初始化时OS将其清0，由MMU在进行写操作的地址转换时将该位置1。
- 高20位是页表或页在主存中的首地址对应的页框号，即首地址的高20位。每个页表的起始位置都按4KB对齐。

总结

- **程序员的角度看待虚拟内存**
  - 每个进程拥有自己私有的线性地址空间
  - 不允许被其他进程干扰
- **系统的角度看待虚拟内存**
  - 通过获取虚拟内存页面来有效使用内存
    - 有效只因为“局部性”的原因
  - 简化编程和内存管理
  - Simplifies protection by providing a convenient interpositioning point to check permissions
    提供方便的标志位来检查权限以简化内存保护

Hope you enjoyed the CSAPP course!