

# 第9章 虚拟内存: 基本概念

教 师: 吴锐

计算机科学与技术学院

哈尔滨工业大学

# 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有
- **Address translation**地址翻译

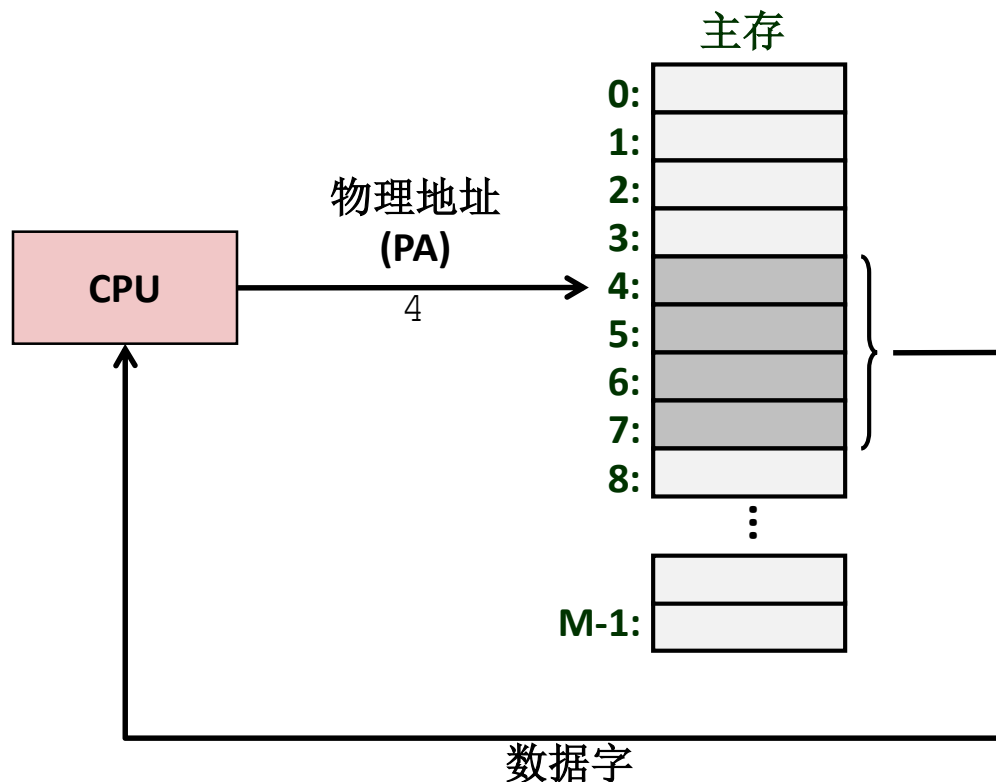
# Why Virtual Memory (VM)?

## 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

# A System Using Physical Addressing

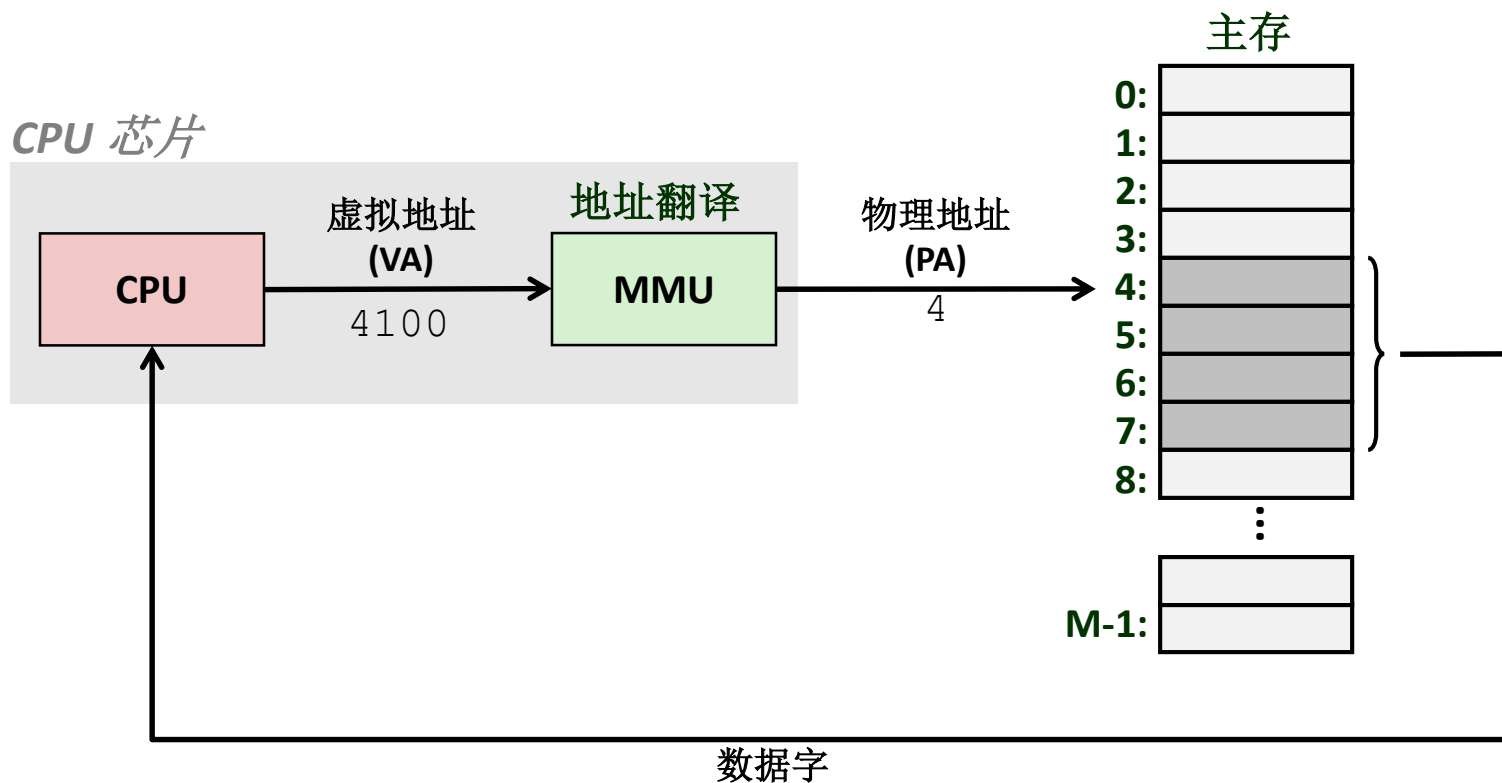
## 一个使用物理寻址的系统



- 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

# 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

# Address Spaces 地址空间

- **地址空间:** 非负整数地址的有序集合

$\{0, 1, 2, 3 \dots\}$

如果地址空间中的整数是连续的, 称为**线性地址空间**

- **虚拟地址空间:**  $N = 2^n$  个虚拟地址的集合

$\{0, 1, 2, 3, \dots, N-1\}$

- **物理地址空间:**  $M = 2^m$  个物理地址的集合

$\{0, 1, 2, 3, \dots, M-1\}$

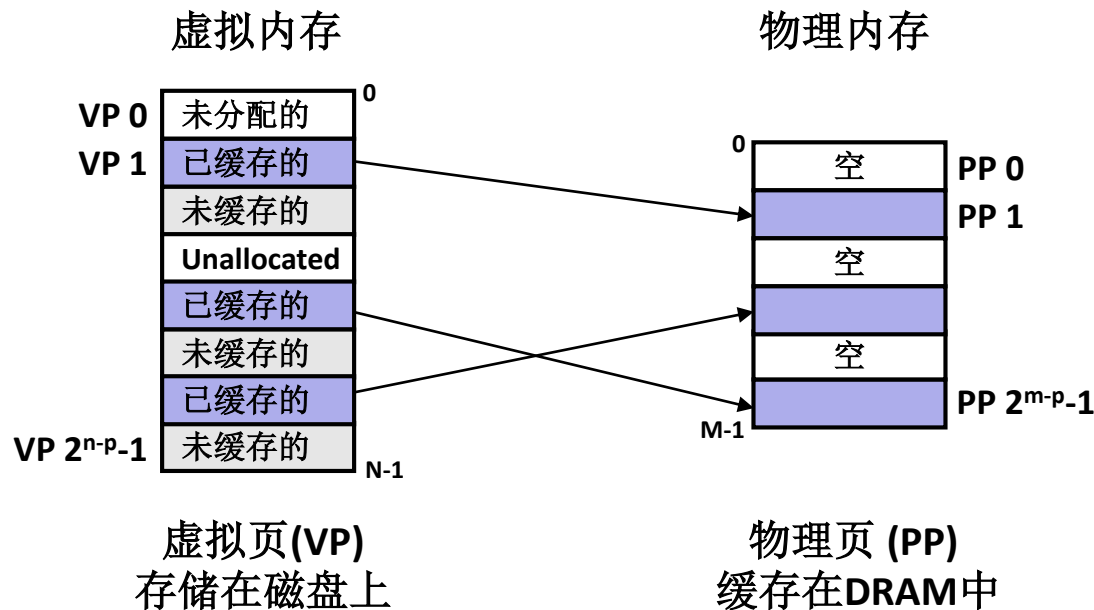
# 主要内容

- 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有
- 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)





# DRAM Cache Organization

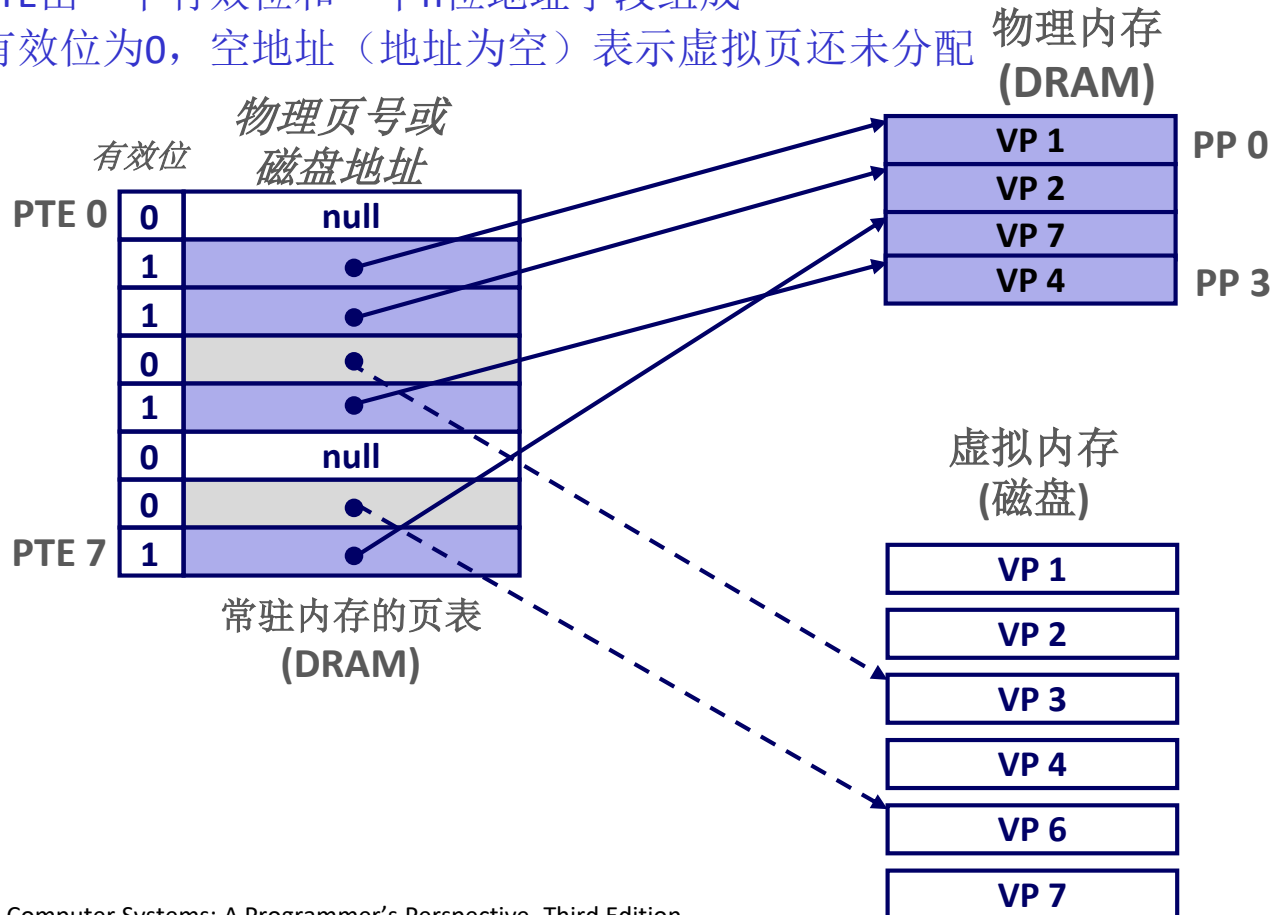
## DRAM缓存的组织结构

- **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 **100,000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

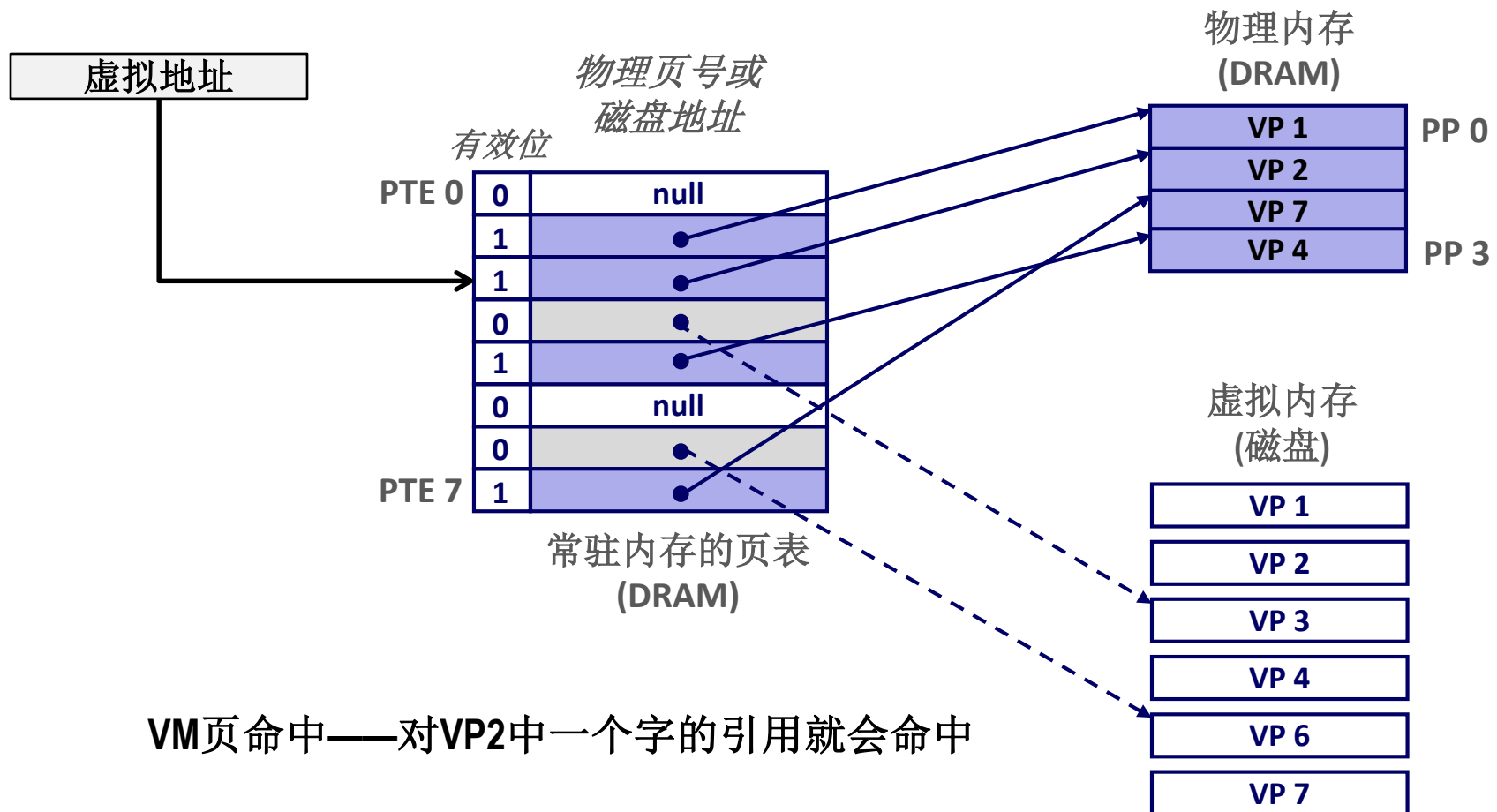
■ **页表**是一个页表条目 (Page Table Entry, PTE)的数组，将虚拟页地址映射到物理页地址。

- 虚拟地址空间中的每个页 (VP) 在页表固定位置有一个PTE
- PTE由一个有效位和一个n位地址字段组成
- 有效位为0，空地址（地址为空）表示虚拟页还未分配



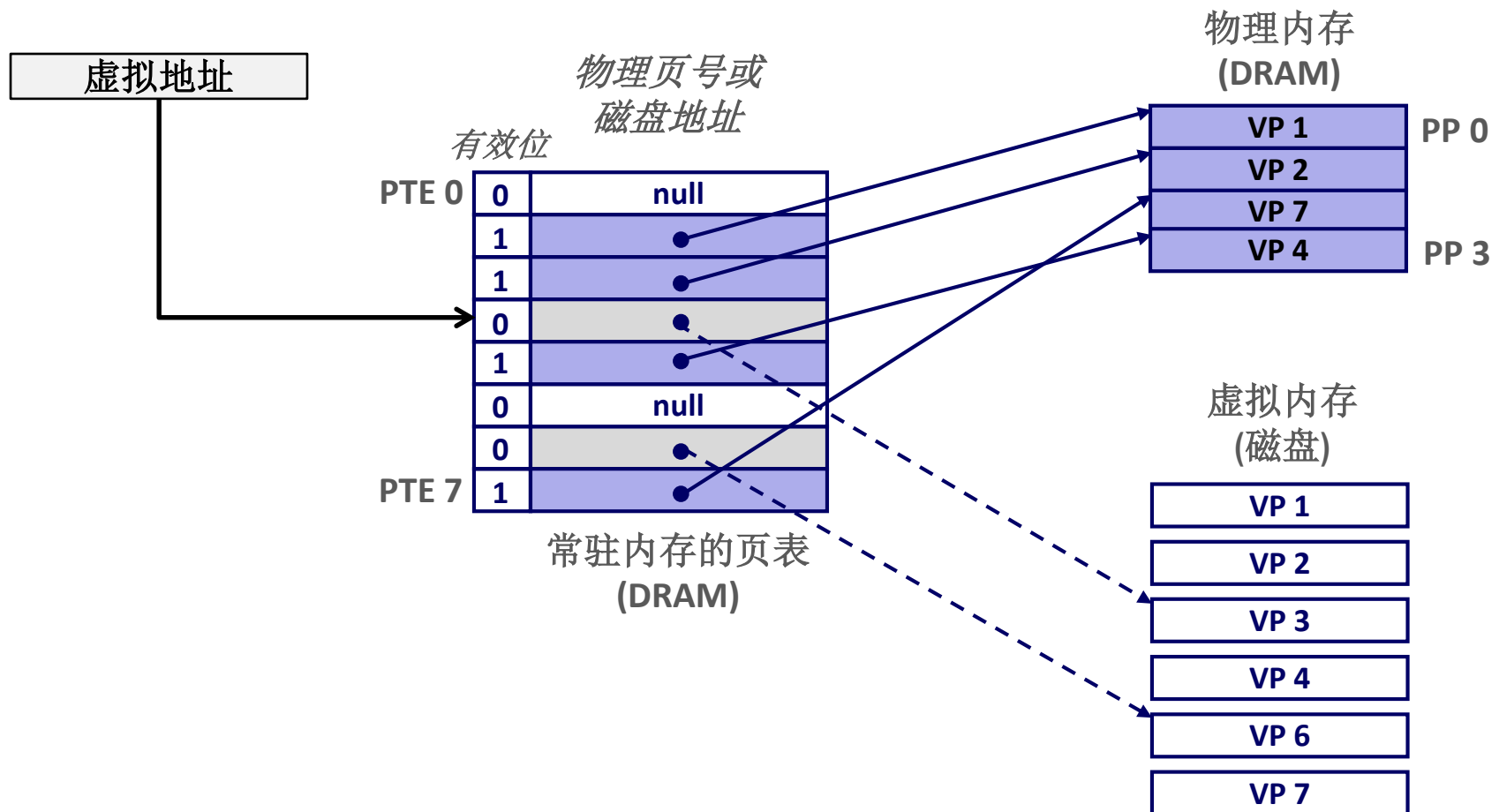
# Page Hit 页命中

- **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)



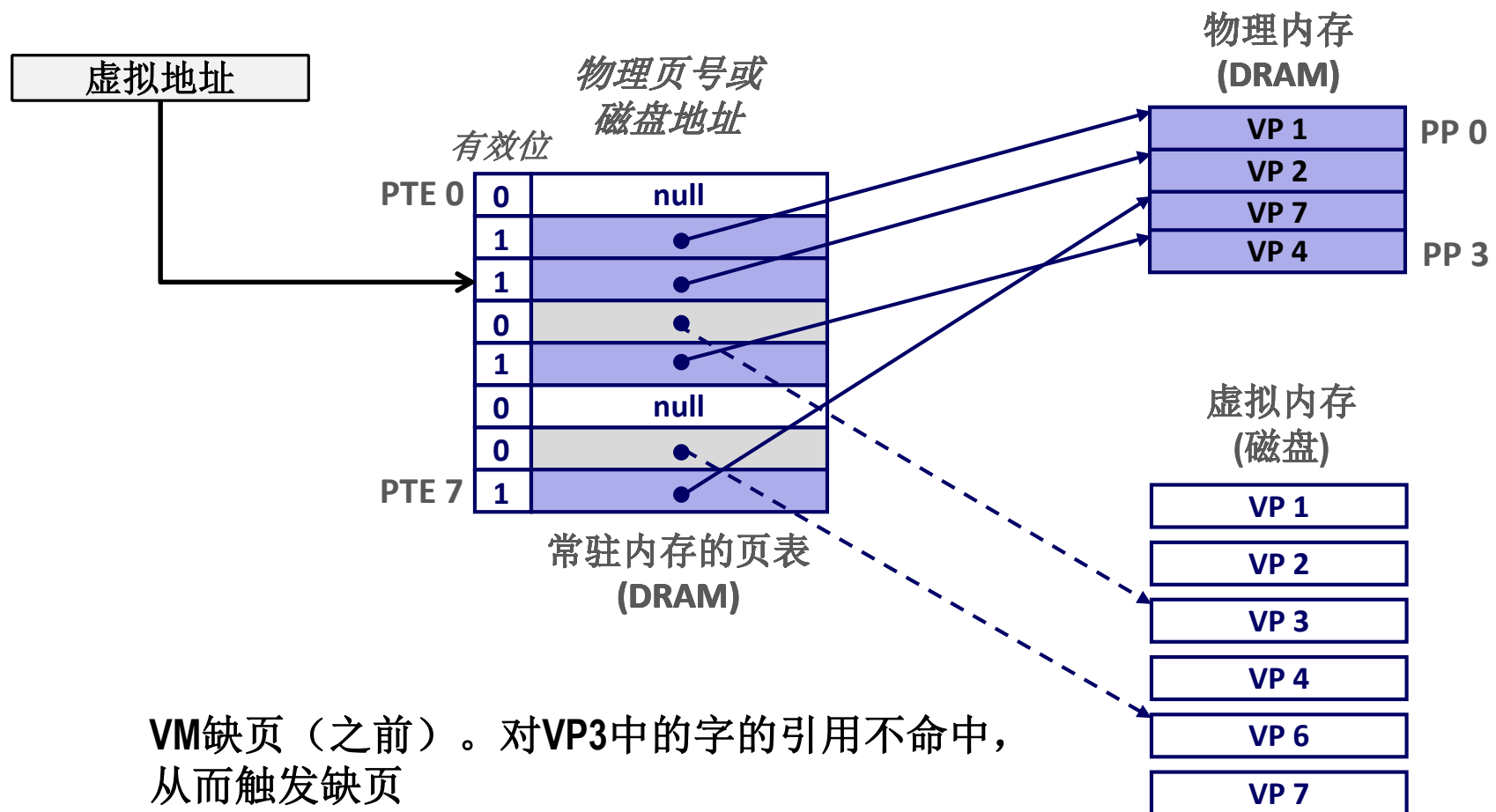
# Page Fault 缺页

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



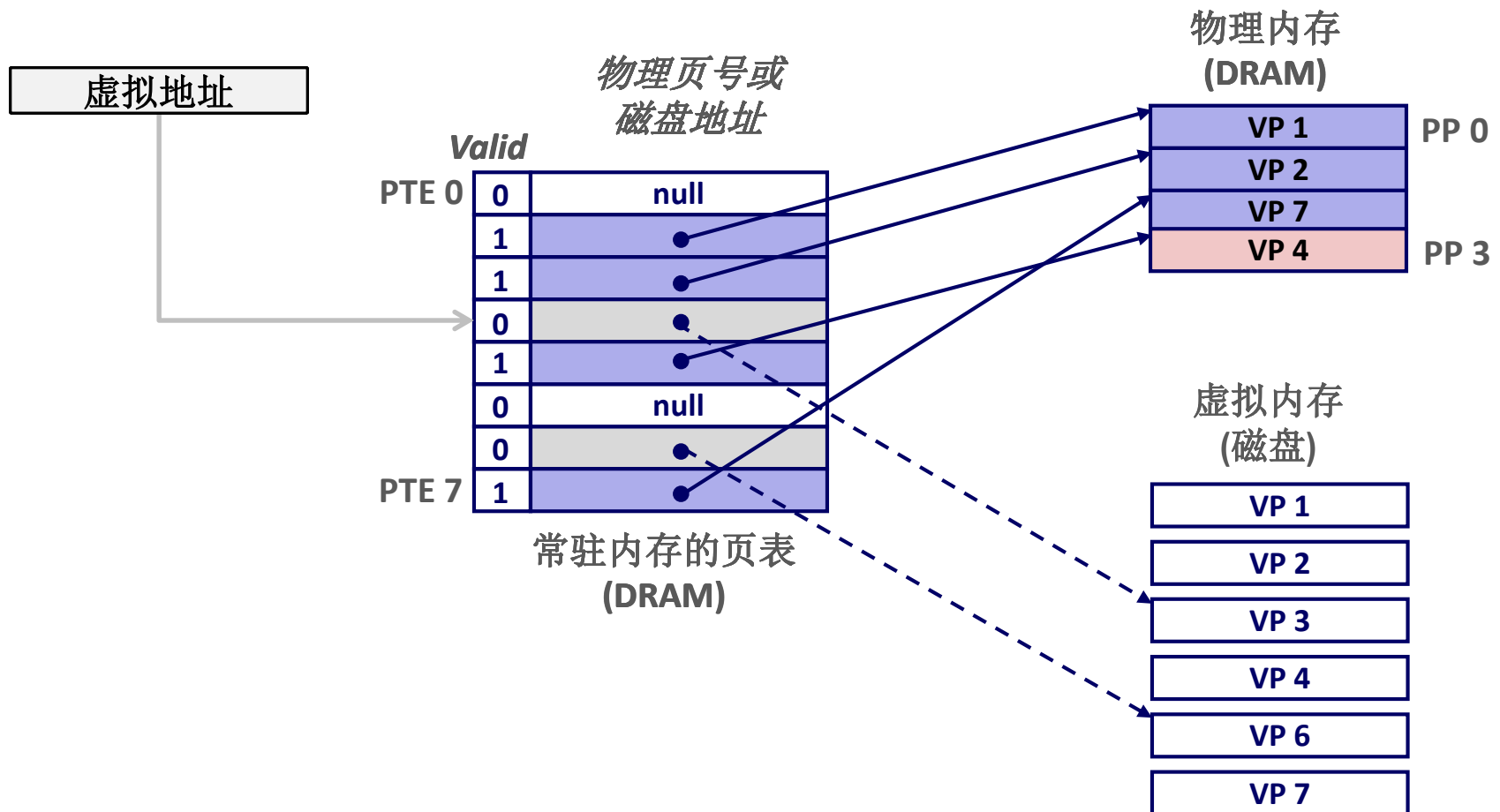
# Handling Page Fault 缺页的处理

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



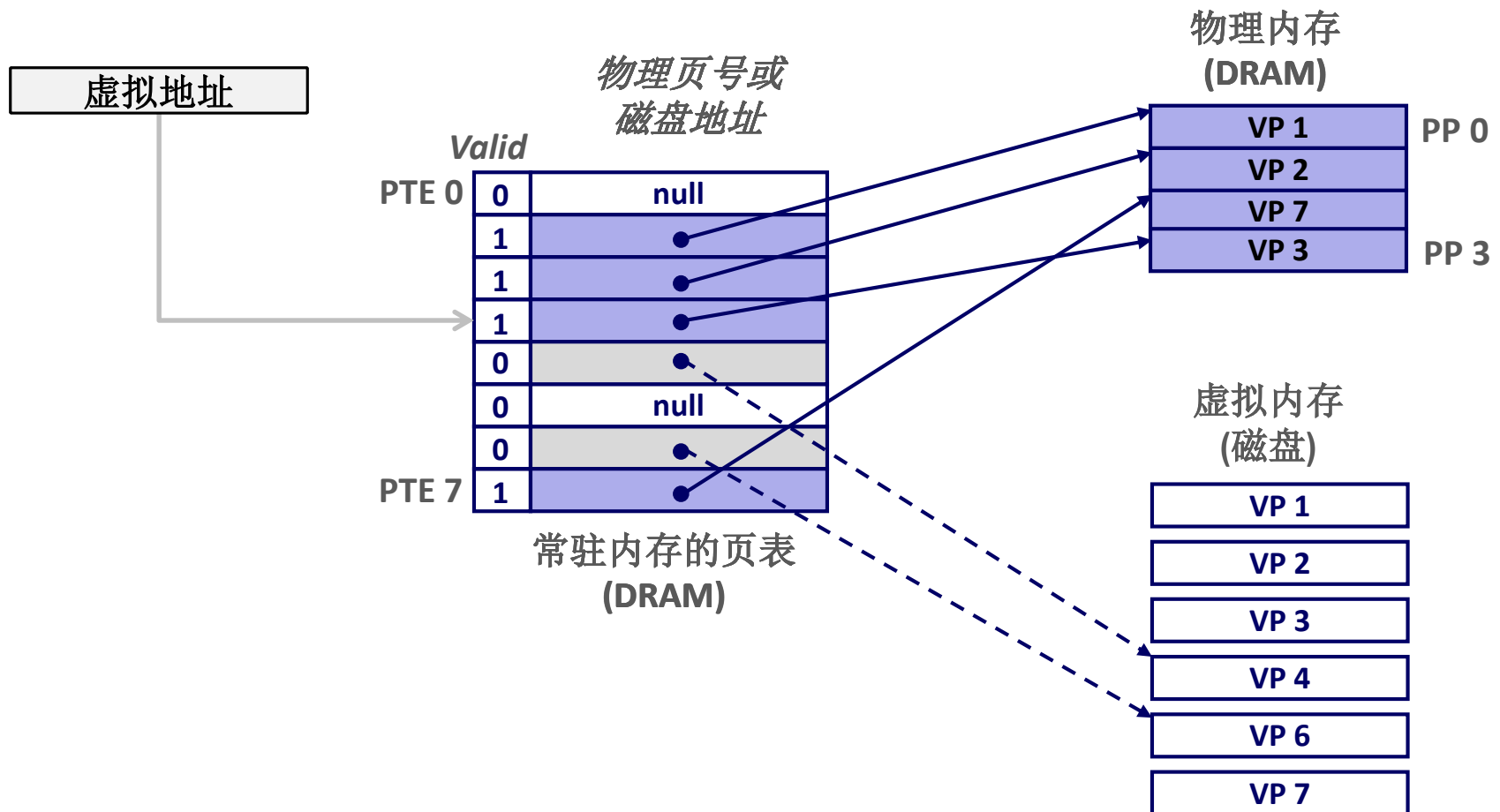
# Handling Page Fault 缺页的处理

- 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)



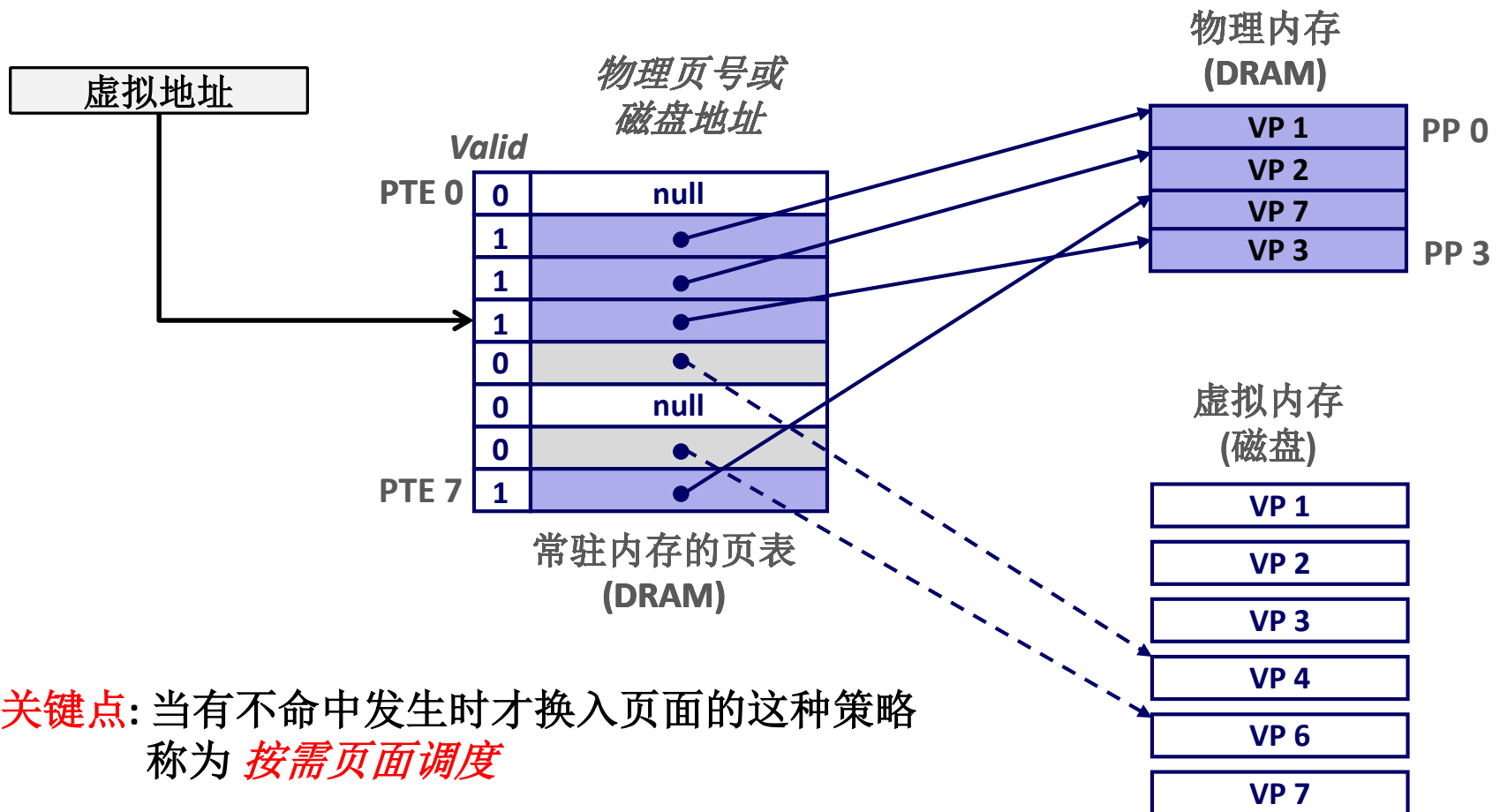
# Handling Page Fault 缺页的处理

- 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

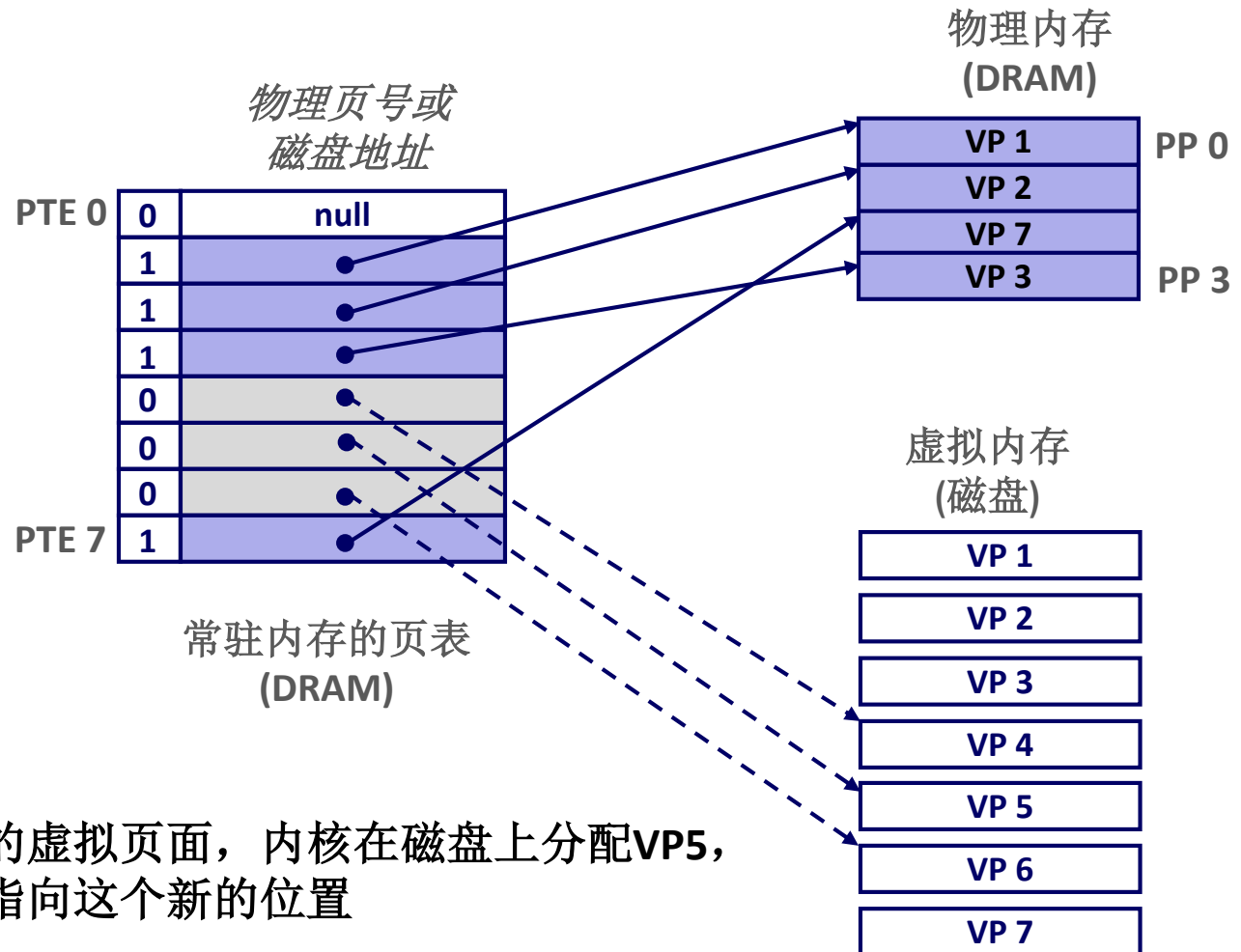
- 缺页导致页面出错 (缺页异常)
- 缺页异常处理程序选择一个牺牲页 (此例中就是 VP 4)
- 导致缺页的指令重新启动: 页面命中!





# 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。

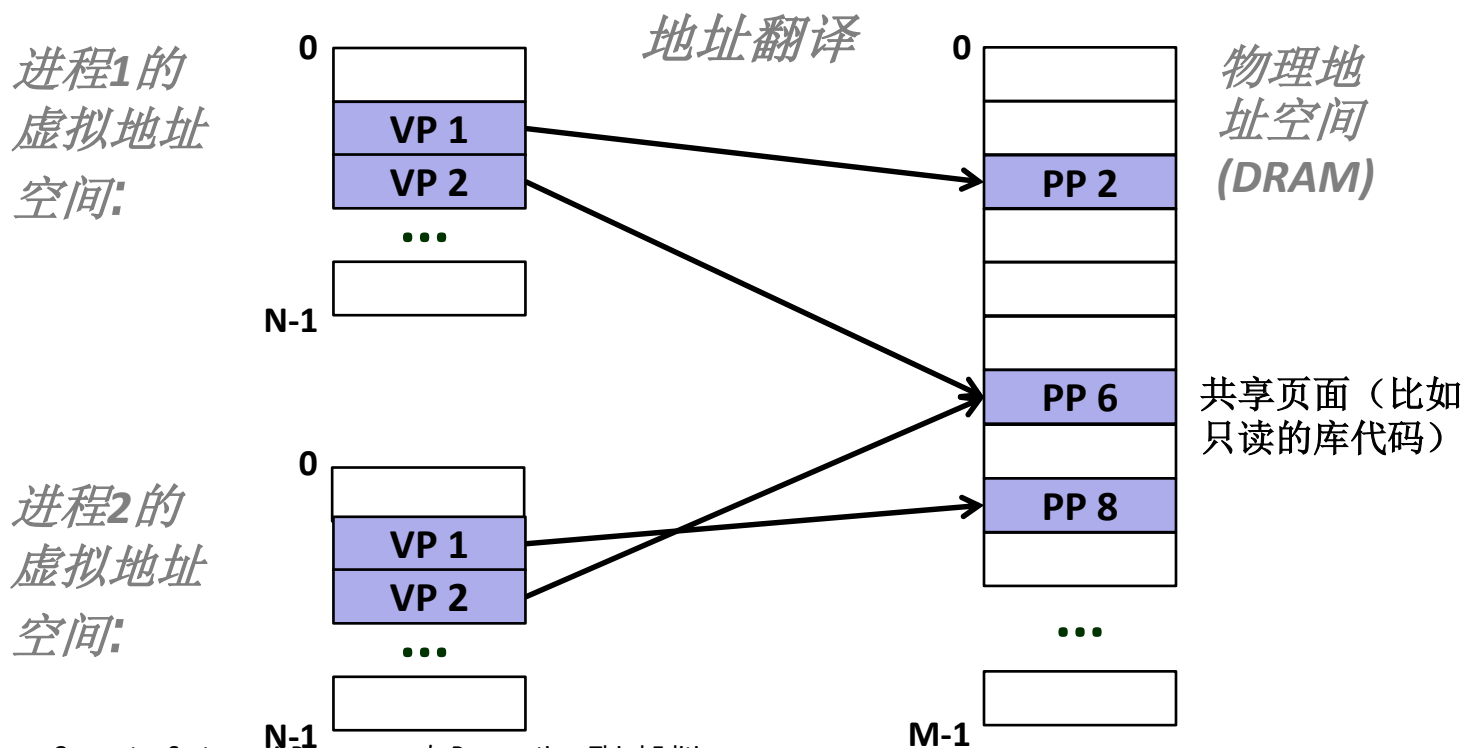
# 主要内容

- 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有
- 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

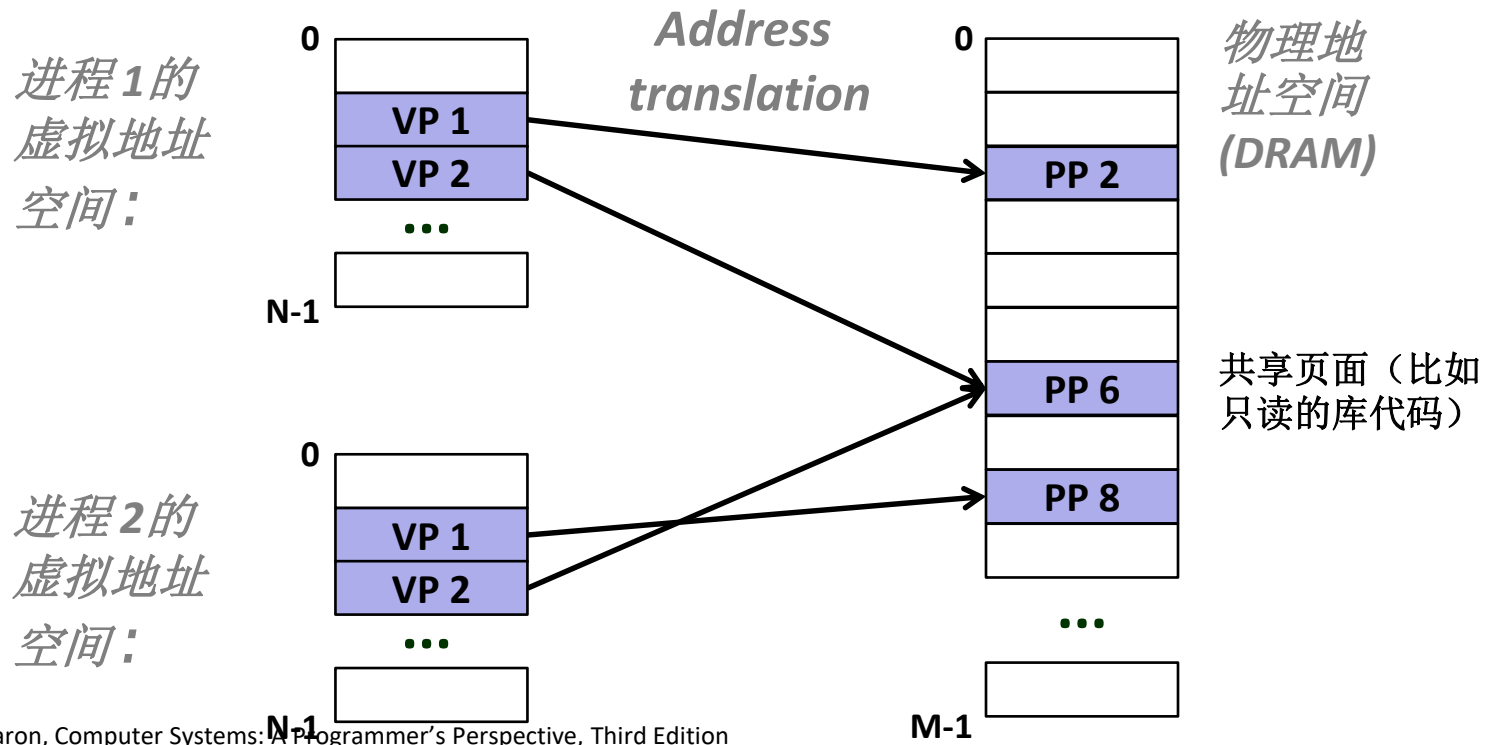
- **Key idea** 核心观点: 每个进程都拥有一个独立的虚拟地址空间
  - 把内存看作独立的简单线性数组
  - 操作系统为系统中每个进程都维护一个独立的页表
    - 也就是一个独立的虚拟地址空间



# 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
  - 一个虚拟内存页面可以被分配到不同的物理页面（可以不连续）
- **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)



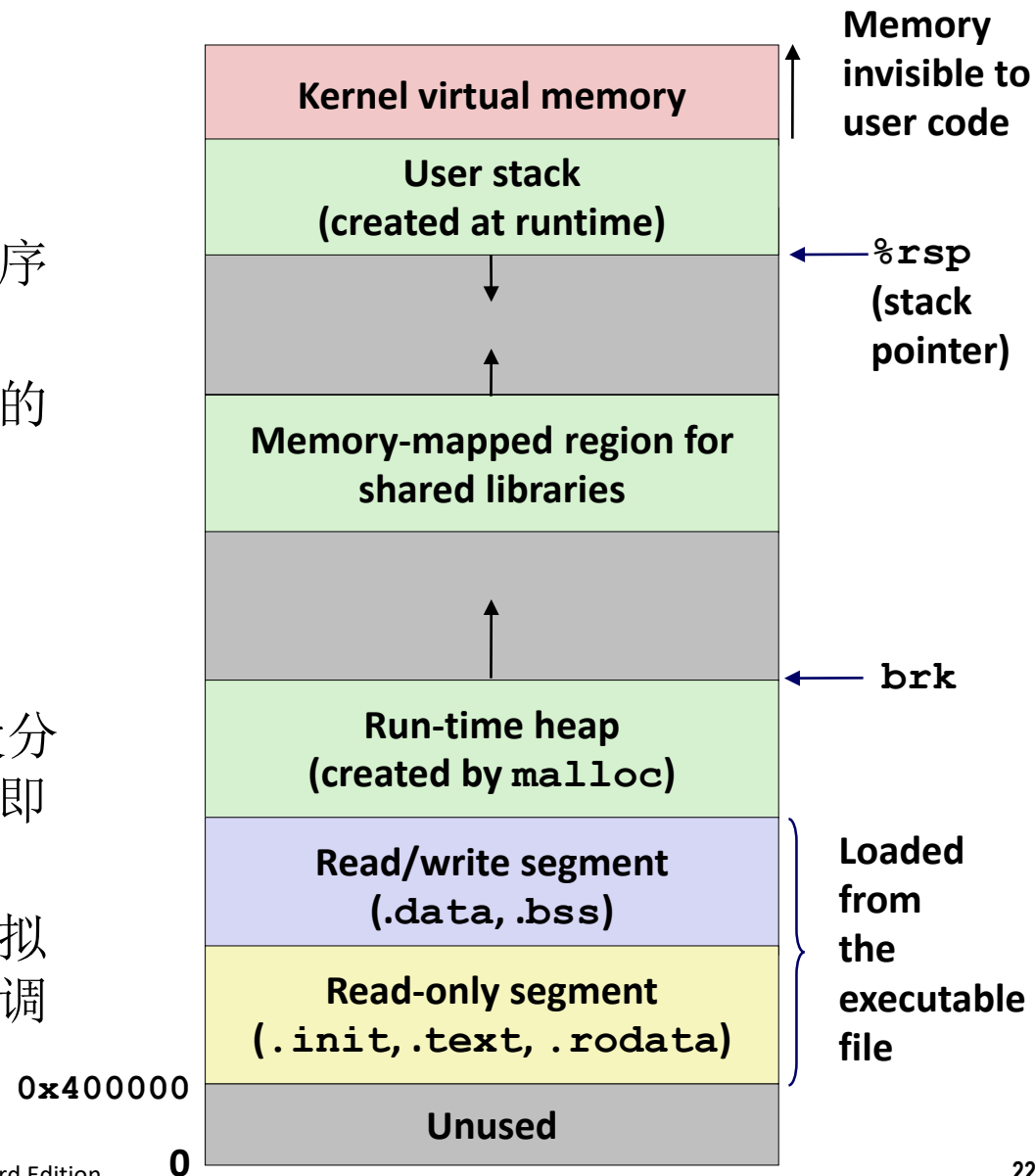
# Simplifying Linking and Loading 简化链接和加载

## ■ Linking 链接

- 独立的地址空间允许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

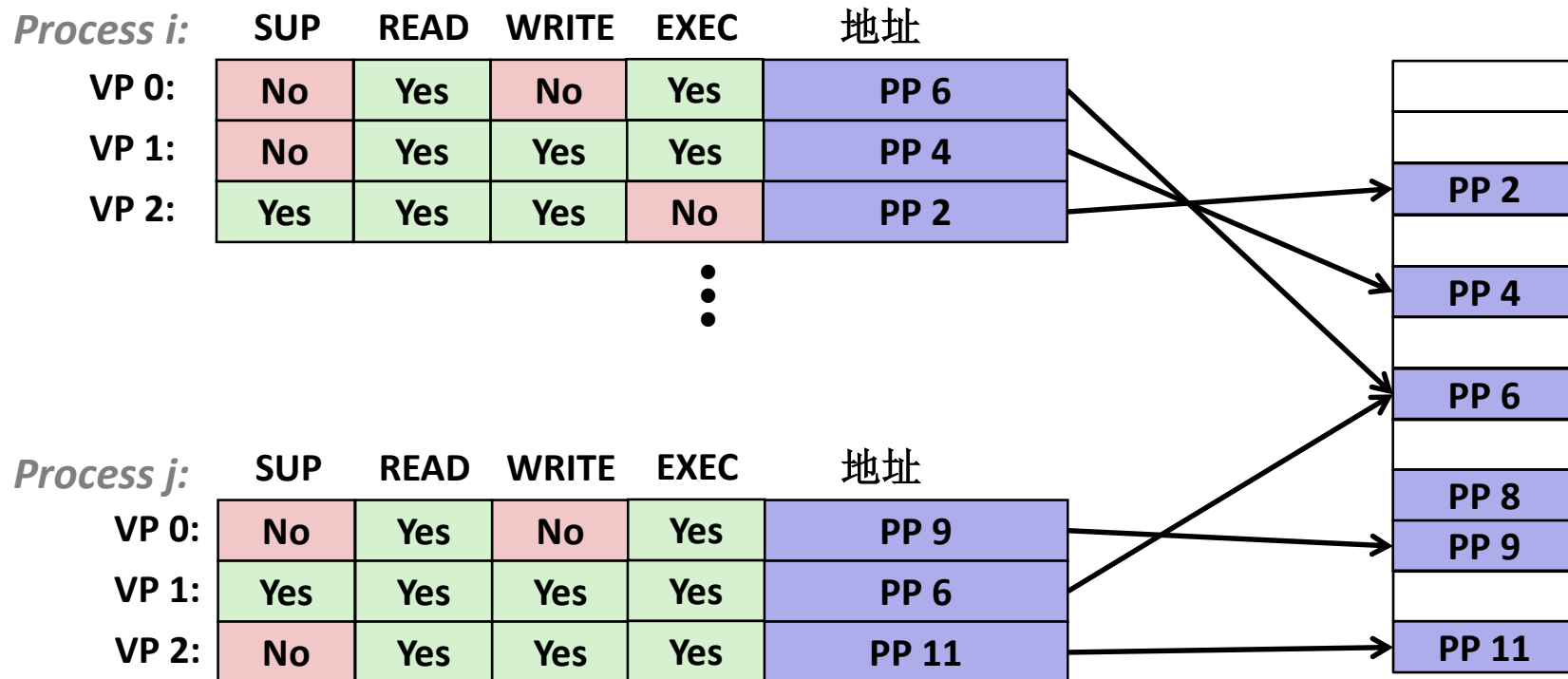
- Address spaces 地址空间
- VM as a tool for caching 虚拟内存作为缓存的工具
- VM as a tool for memory management  
虚拟内存作为内存管理的工具
- **VM as a tool for memory protection**  
虚拟内存作为内存保护的**工具**
- Address translation 地址翻译

# VM as a Tool for Memory Protection

## 虚拟内存作为内存保护的工具体

- 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

物理内存



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有
- Address translation地址翻译

# VM Address Translation 地址翻译

- Virtual Address Space 虚拟地址空间
  - $V = \{0, 1, \dots, N-1\}$
- Physical Address Space 物理地址空间
  - $P = \{0, 1, \dots, M-1\}$
- Address Translation 地址翻译
  - $MAP: V \rightarrow P \cup \emptyset$
  - 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
    - $MAP(a) = \emptyset$  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

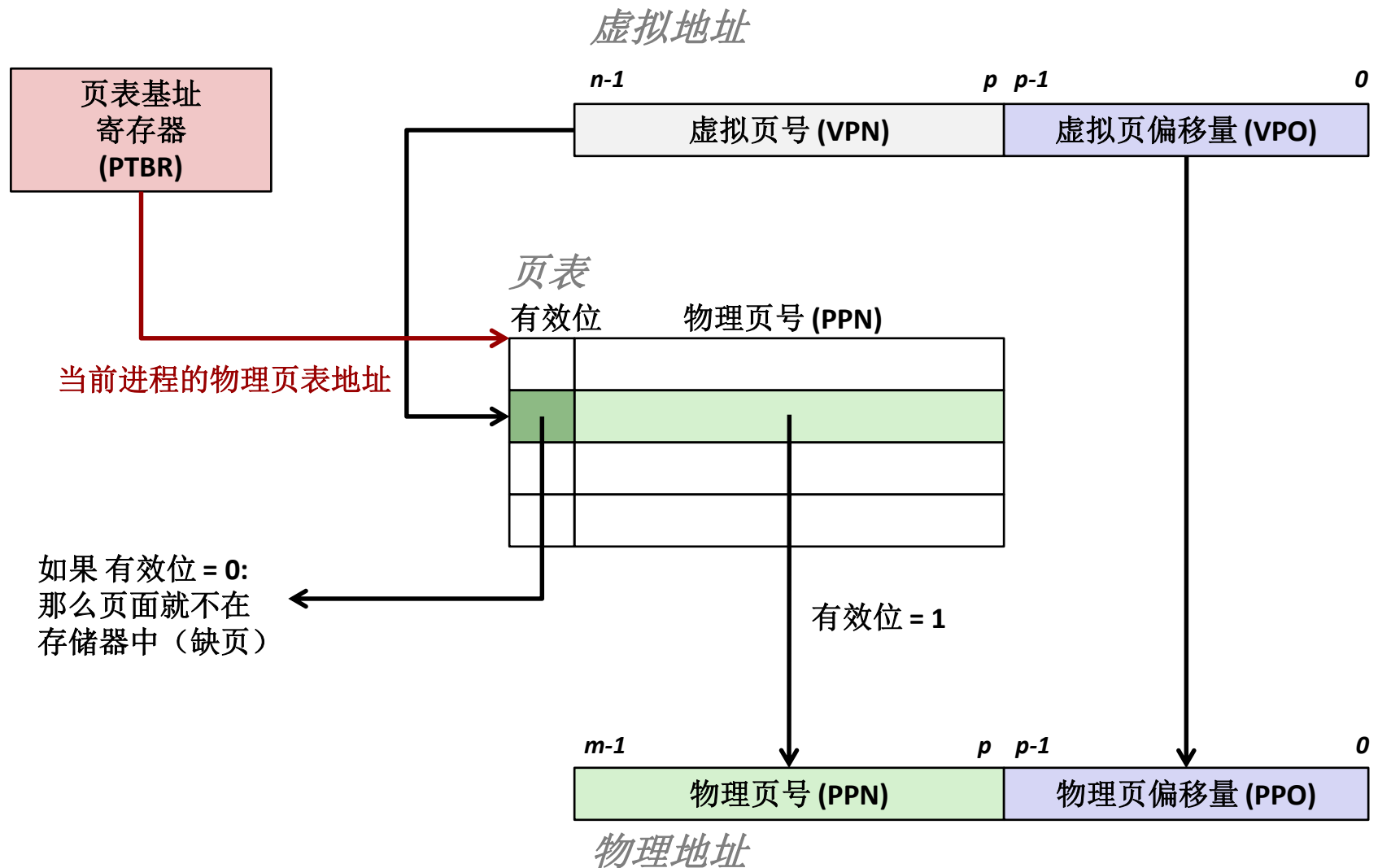
- TLBI: TLB index----TLB索引
- TLBT: TLB tag----TLB标记
- VPO: Virtual page offset----虚拟页面偏移量 (字节)
- VPN: Virtual page number----虚拟页号

## ■ 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

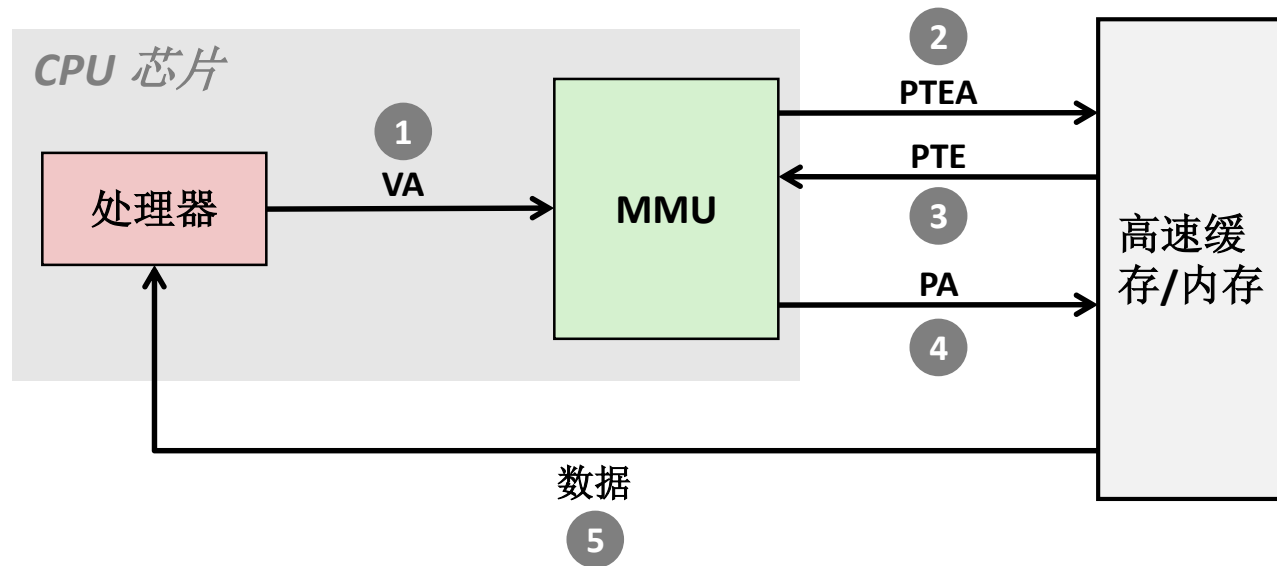
# Address Translation With a Page Table

## 基于页表的地址翻译



# Address Translation: Page Hit

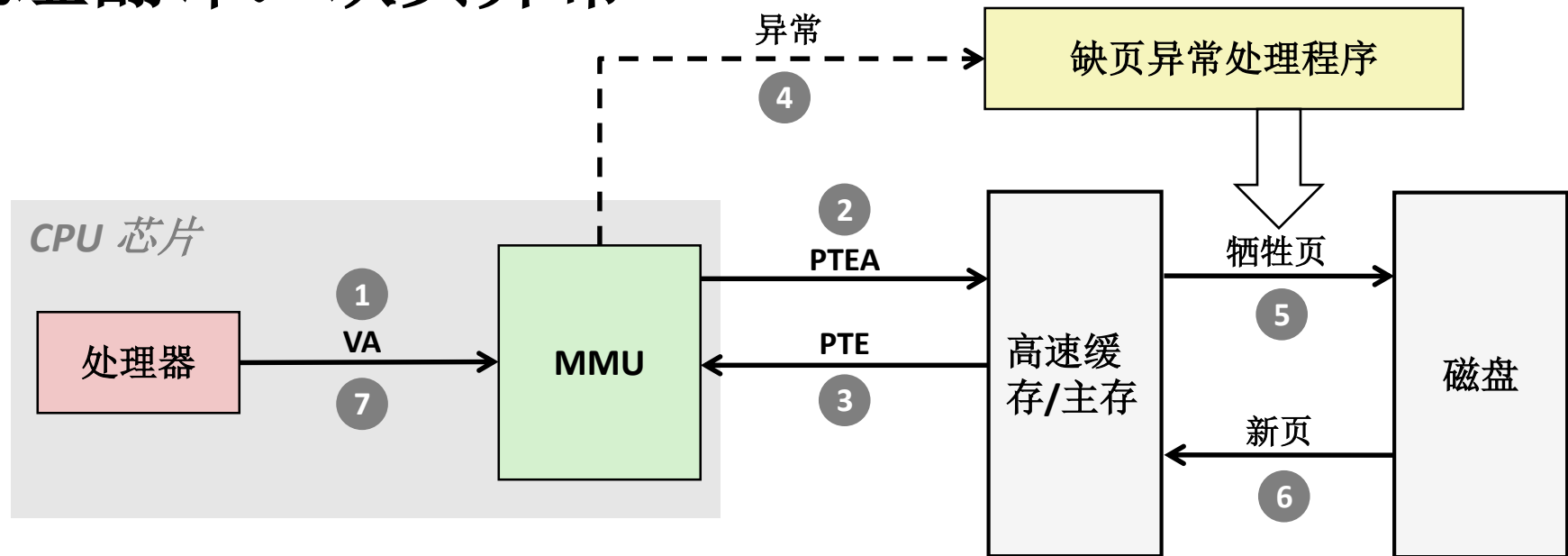
## 地址翻译：页面命中



- 1) 处理器生成一个虚拟地址，并将其传送给MMU
- 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

# Address Translation: Page Fault

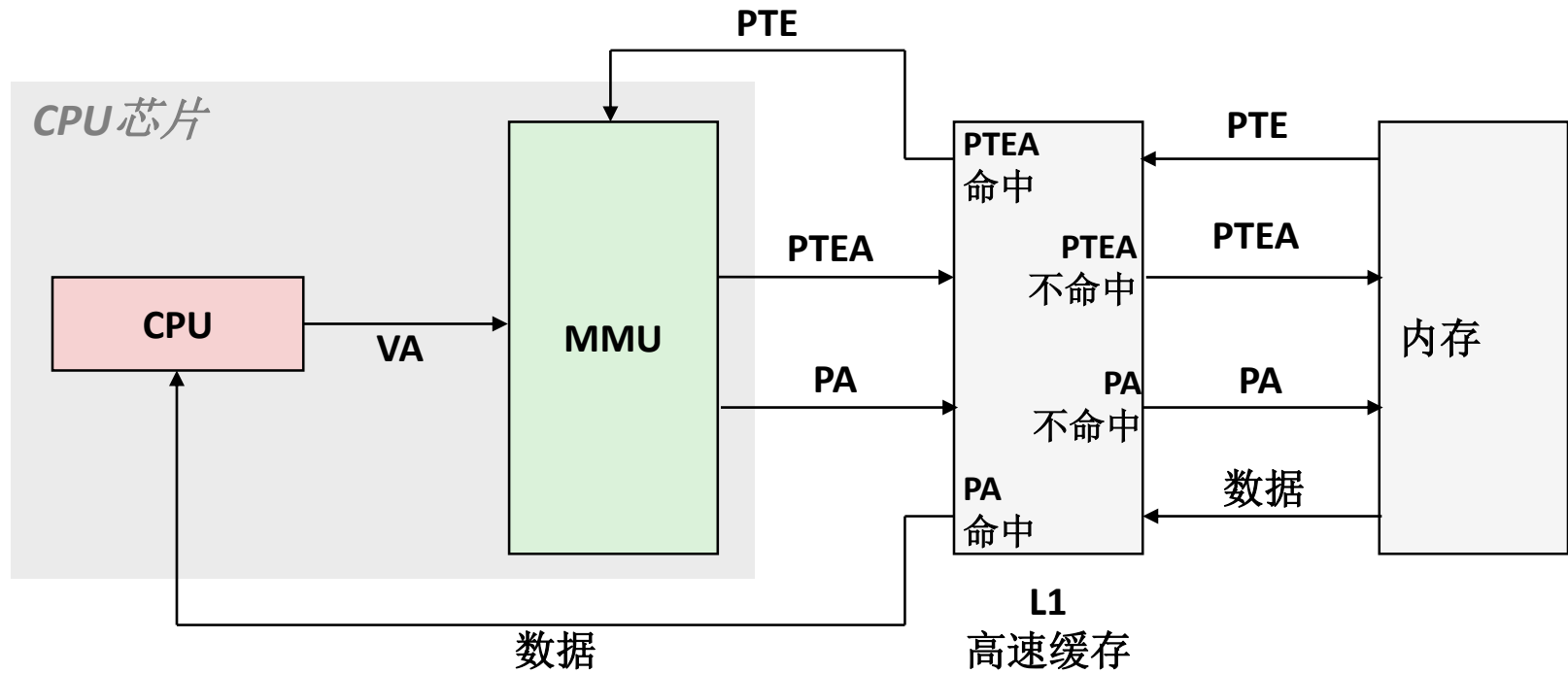
## 地址翻译：缺页异常



- 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牺牲页 (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

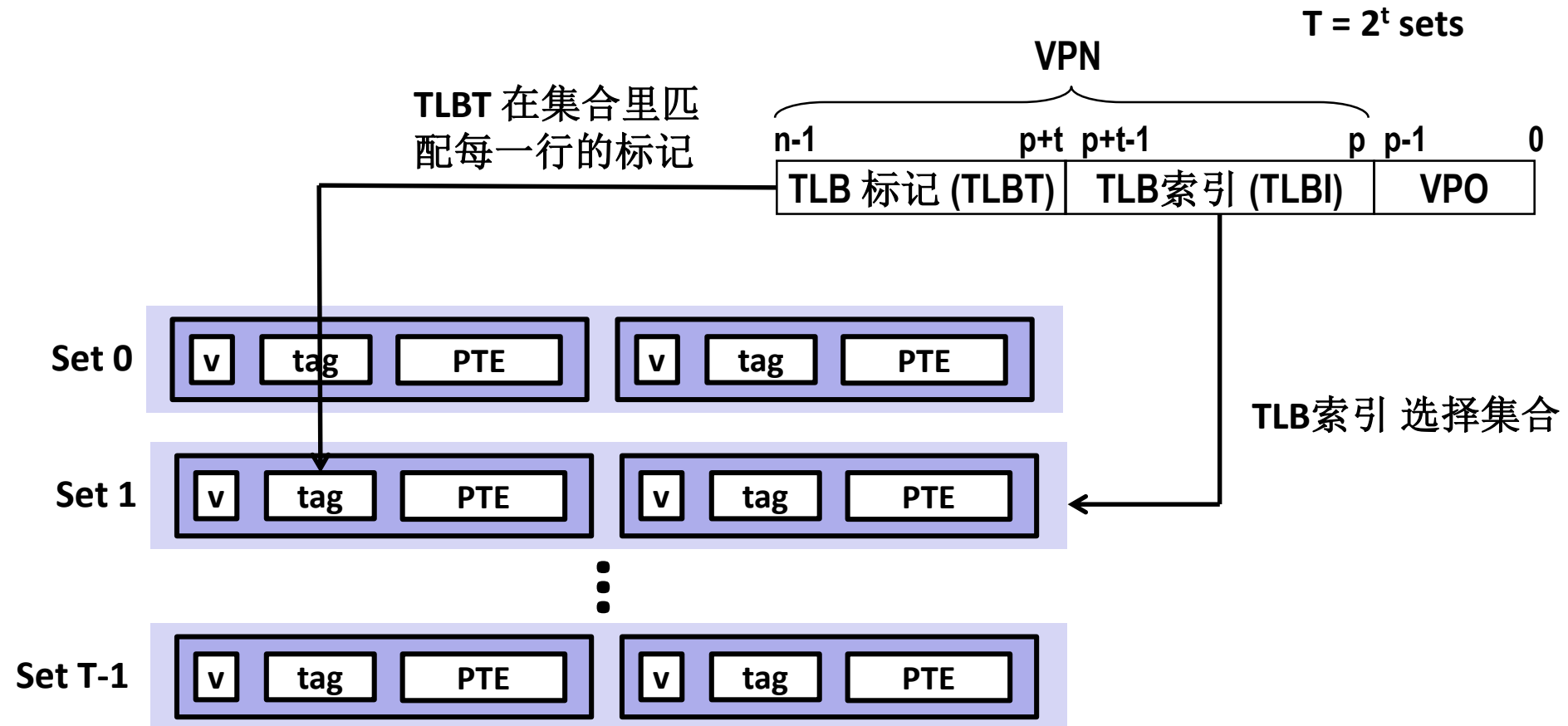
## 利用TLB加速地址翻译

- 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
  - 对于页码数很少的页表可以完全包含在TLB中

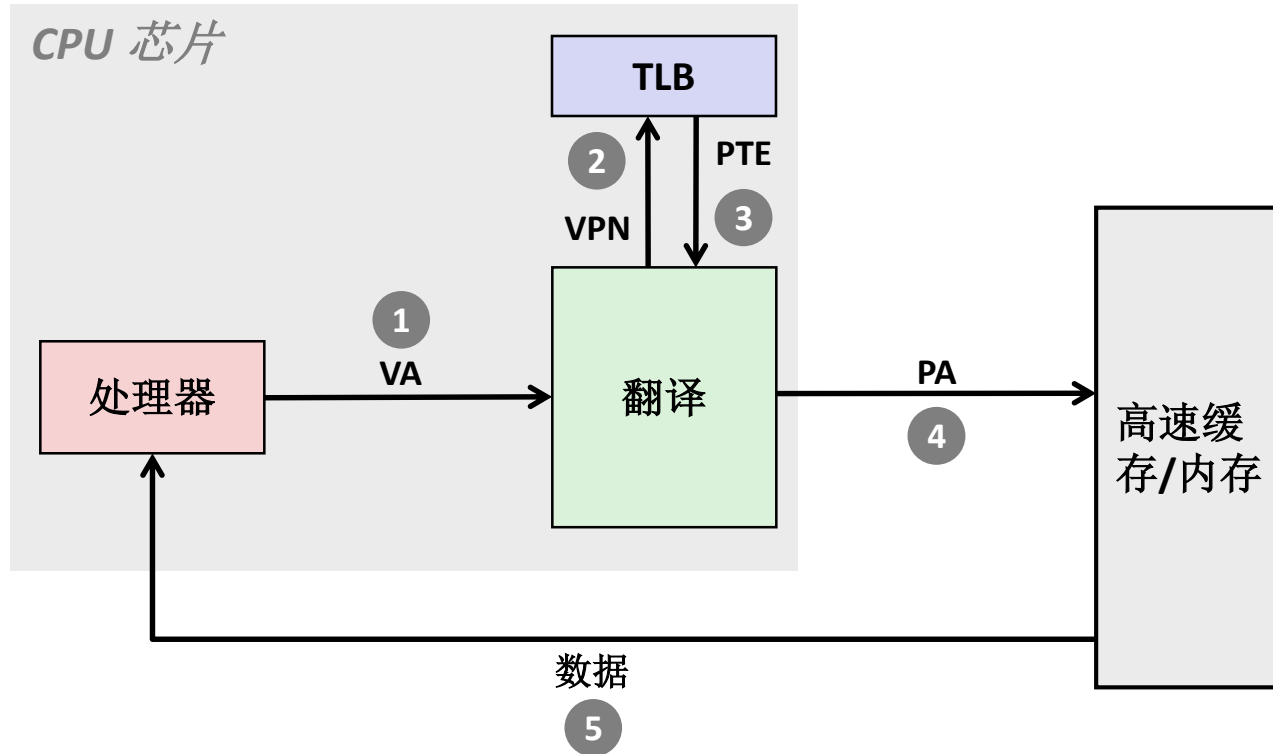


# Accessing the TLB 访问TLB

- MMU 使用虚拟地址的 VPN 部分来访问TLB:

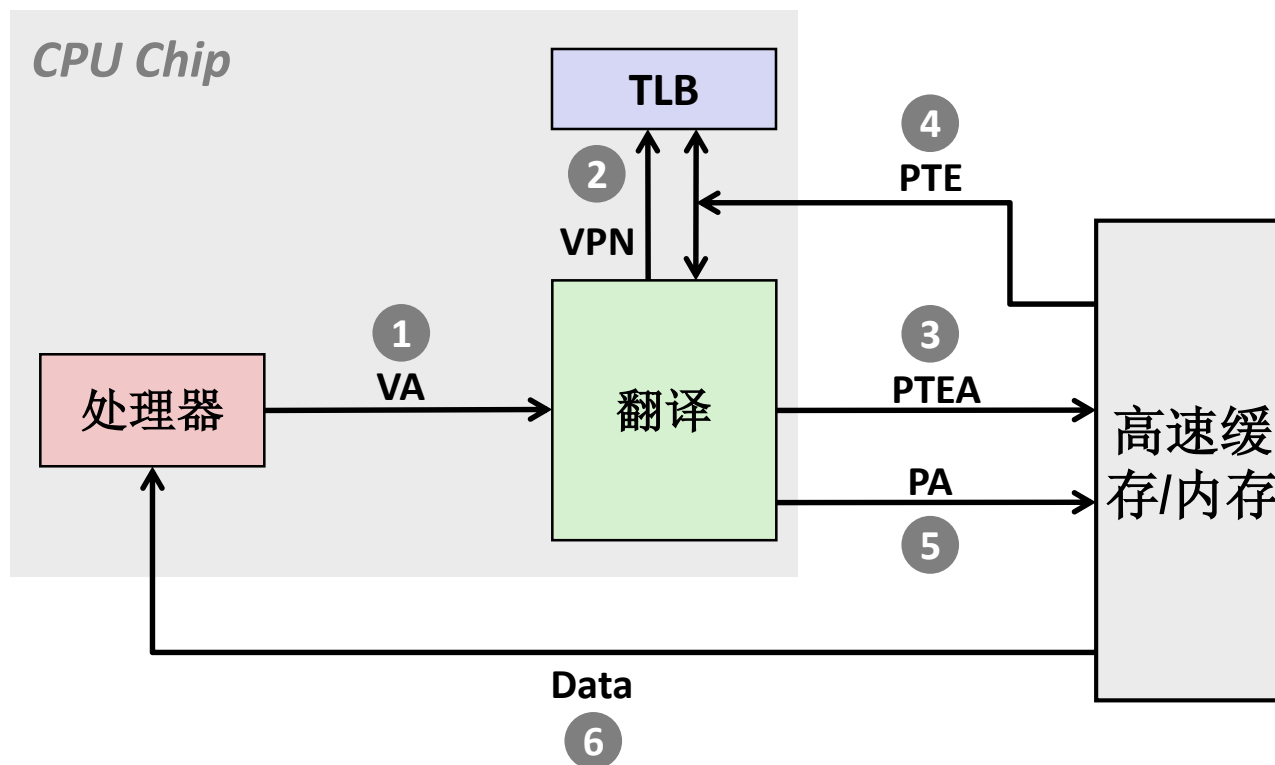


# TLB Hit TLB命中



TLB 命中减少内存访问

# TLB 不命中

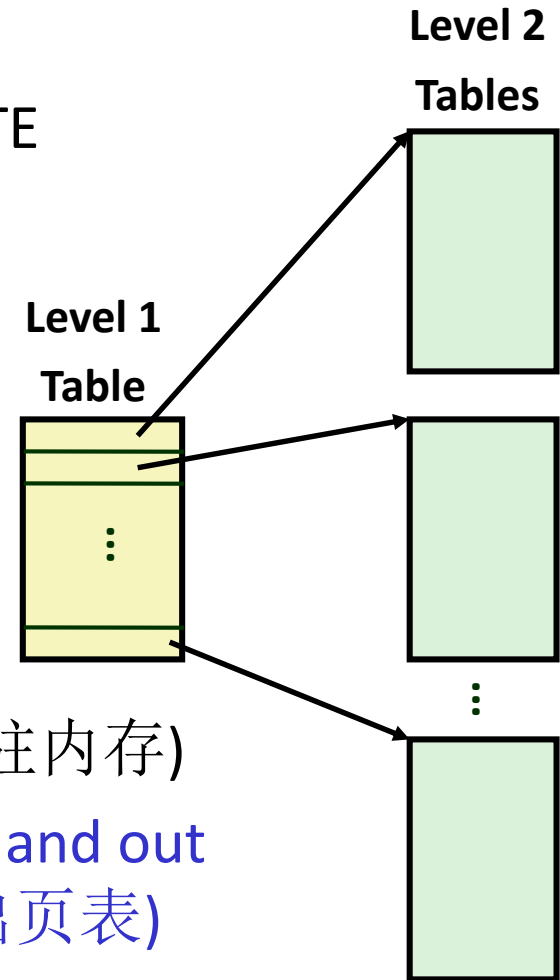


**TLB 不命中引发了额外的内存访问**

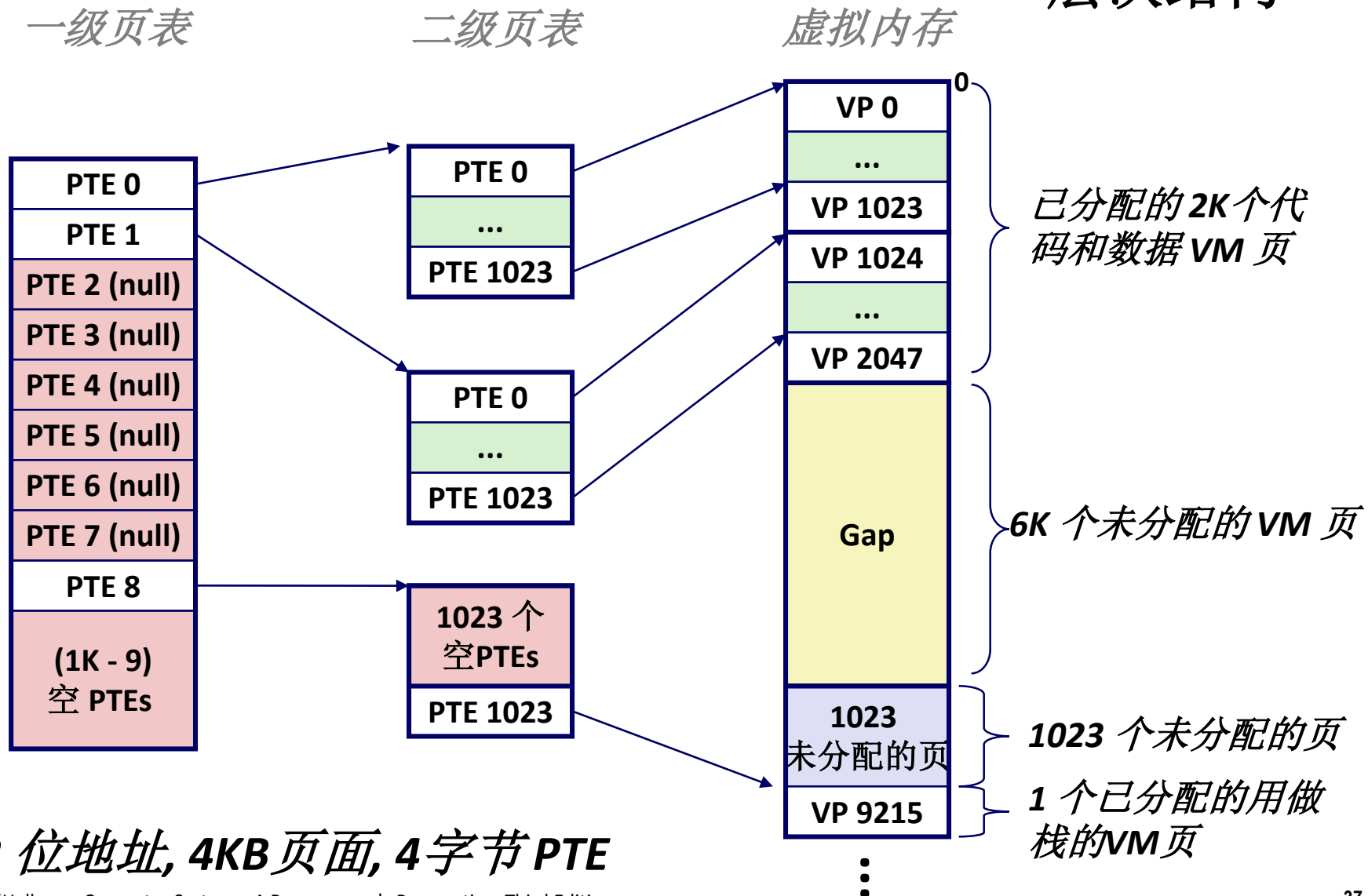
万幸的是, TLB 不命中很少发生。

# 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:
  - 将需要一个大小为 512 GB 的页表!
    - $2^{48} * 2^{-12} * 2^3 = 2^{39}$  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)

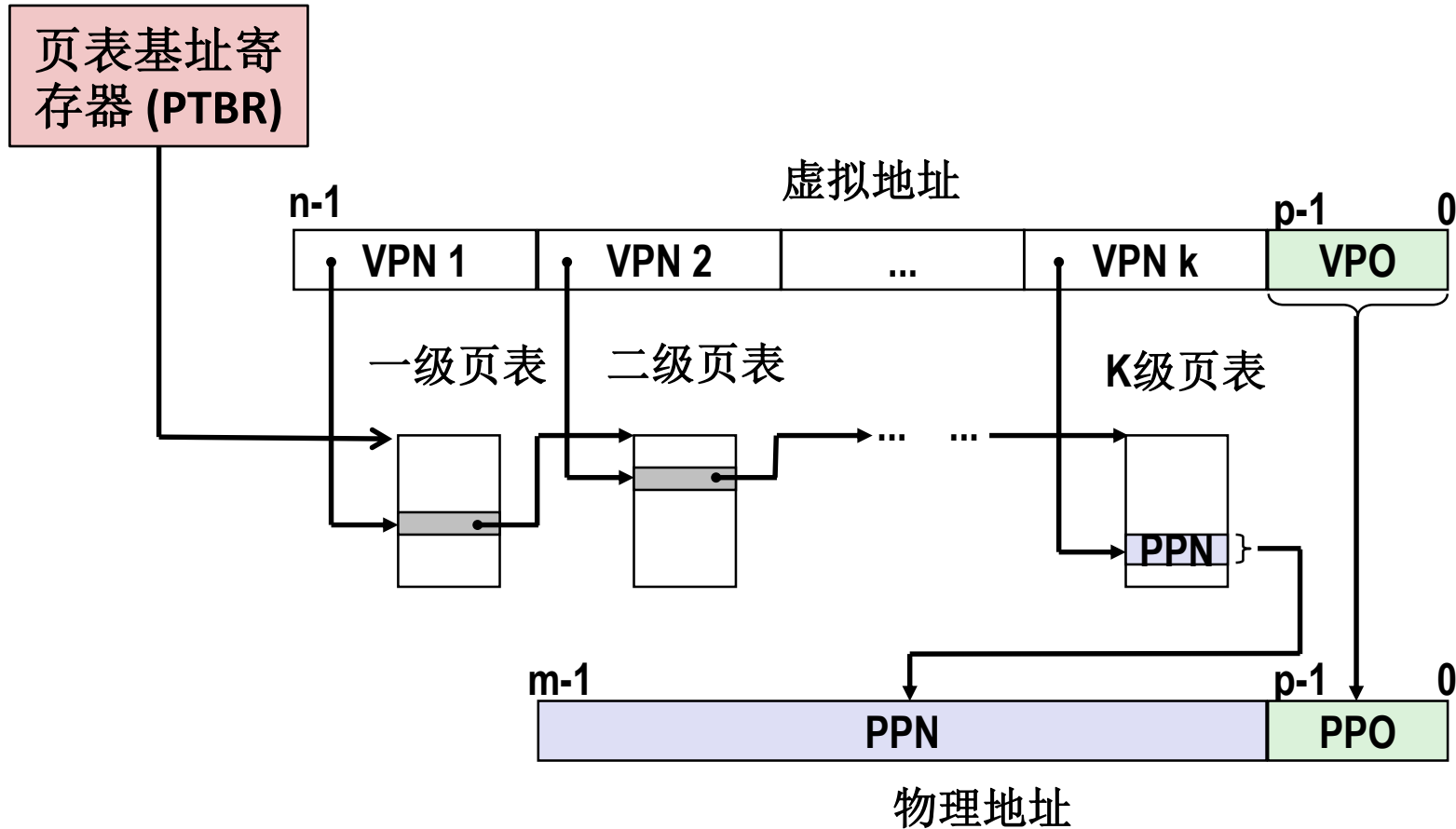


# A Two-Level Page Table Hierarchy 二级页表的层次结构



# Translating with a k-level Page Table

## 使用K级页表的地址翻译



# 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