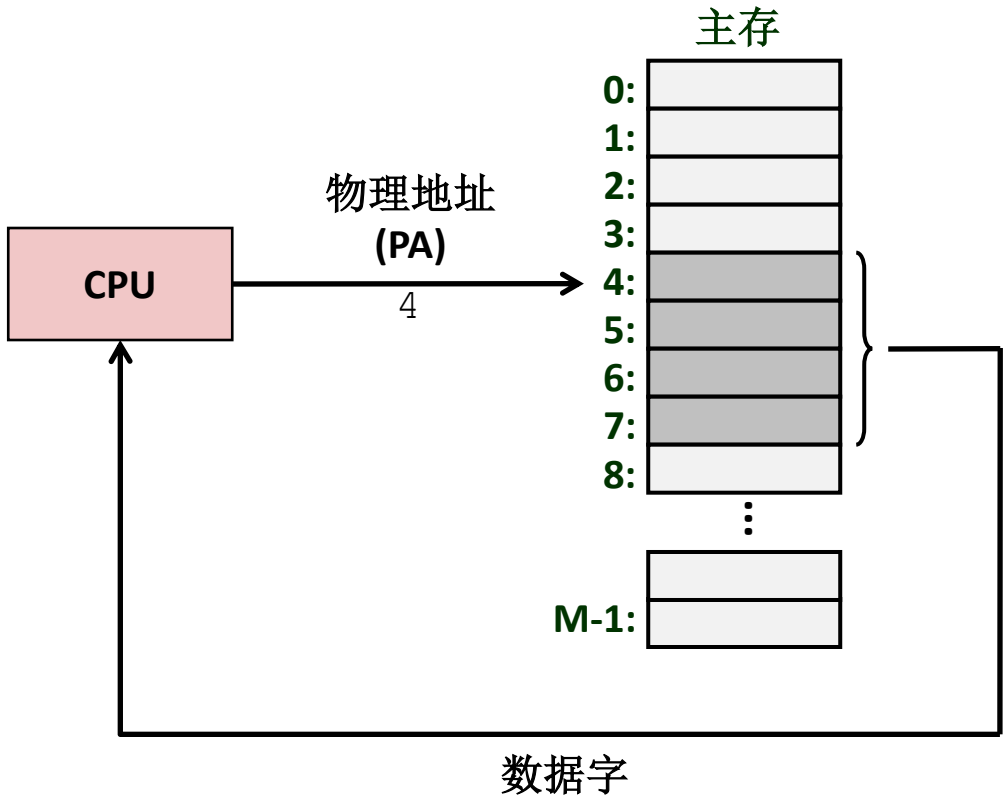
虚拟内存——概念
Virtual Memory: Concepts

课 程 名 ： 计算机系统
主 讲 人 ： 孟文龙

主要内容

- 地址空间
- 虚拟内存作为缓存的工具
- 虚拟内存作为内存管理的工具
- 虚拟内存作为内存保护的工具
- 地址翻译

采用物理寻址的系统



1. 简单直接

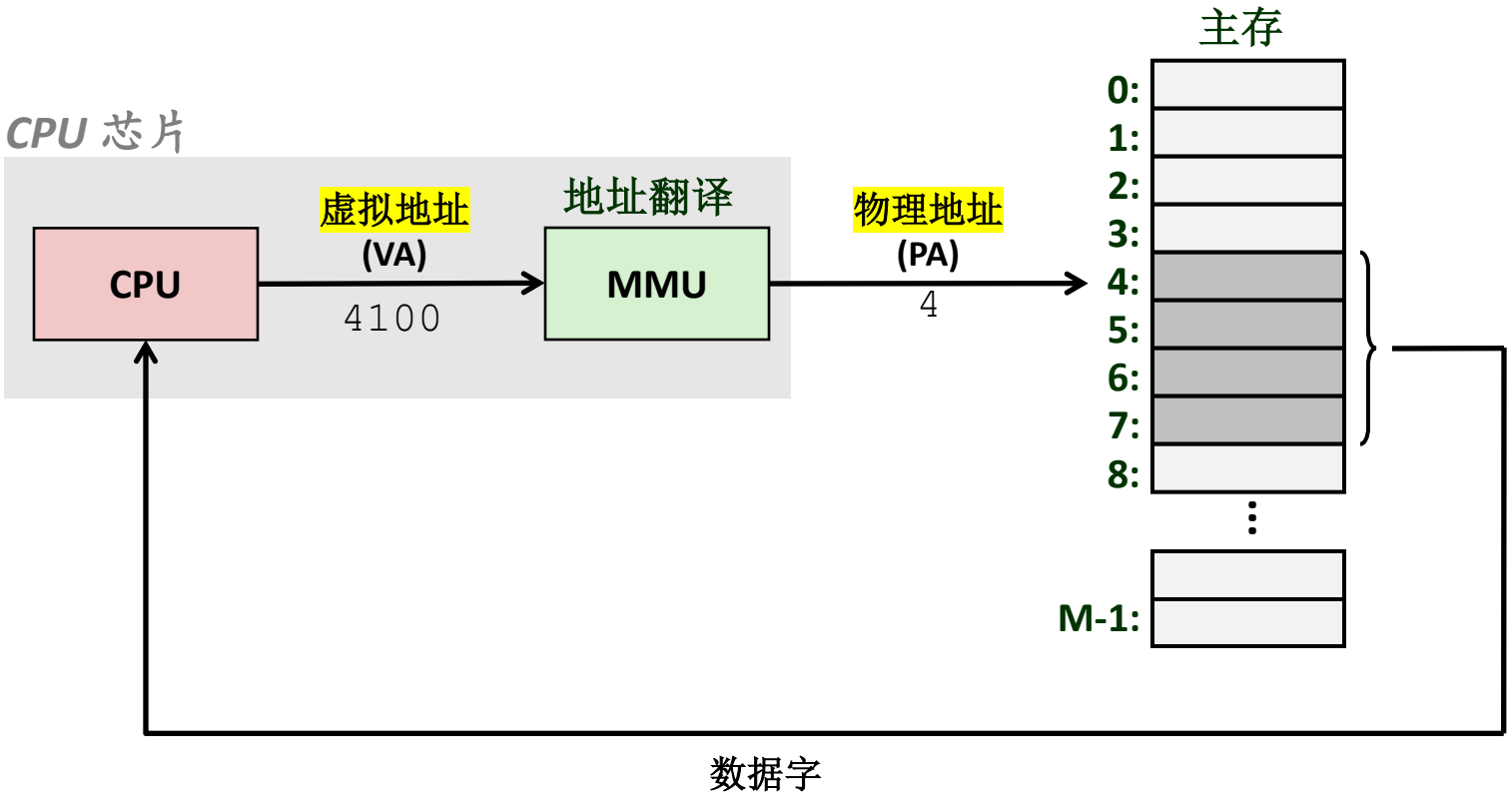
   地址空间小，硬件结构简单，访问速度快。

2. 无地址转换

   CPU生成的地址就是主存地址，无需地址映射或转换逻辑。

- 在诸如汽车电子、电梯等“简单”系统中作为嵌入式微控制器使用

采用虚拟寻址的系统



- 内存保护与隔离、地址空间大、内存管理灵活
- 用于所有现代服务器、笔记本电脑 、智能手机等

地址空间

- **线性地址空间：**连续的非负整数地址的有序集合：
  是一种中间地址，通常由CPU通过分段机制计算得到。在分页机制下，线性地址会被送入页表，映射为物理地址。
- **虚拟地址空间：**N = 2^n 个虚拟地址的集合：
  虚拟地址空间是操作系统**为每个进程提供的一个独立**、完整的地址空间。
  虚拟地址空间的大小由操作系统和硬件（CPU位数）决定
- **物理地址空间：**M = 2^m 个物理地址的集合：
  物理地址空间的大小由实际内存条的容量决定，比如8GB物理内存，物理地址空间就是2^{33}（约8G）个地址。

为什么要采用虚拟内存VM？

- 高效地使用主存

  虚拟内存允许操作系统把磁盘空间当作扩展内存，只有活跃使用的数据才会被加载到实际的物理内存（DRAM）中。可以让物理内存（主存）的利用效率更高

- 简化内存管理

  虚拟内存为每个进程提供了一个看似连续、完整的地址空间，即使物理内存是碎片化的。程序员不需要关心物理内存的分配细节，编程更简单。

- 地址空间隔离

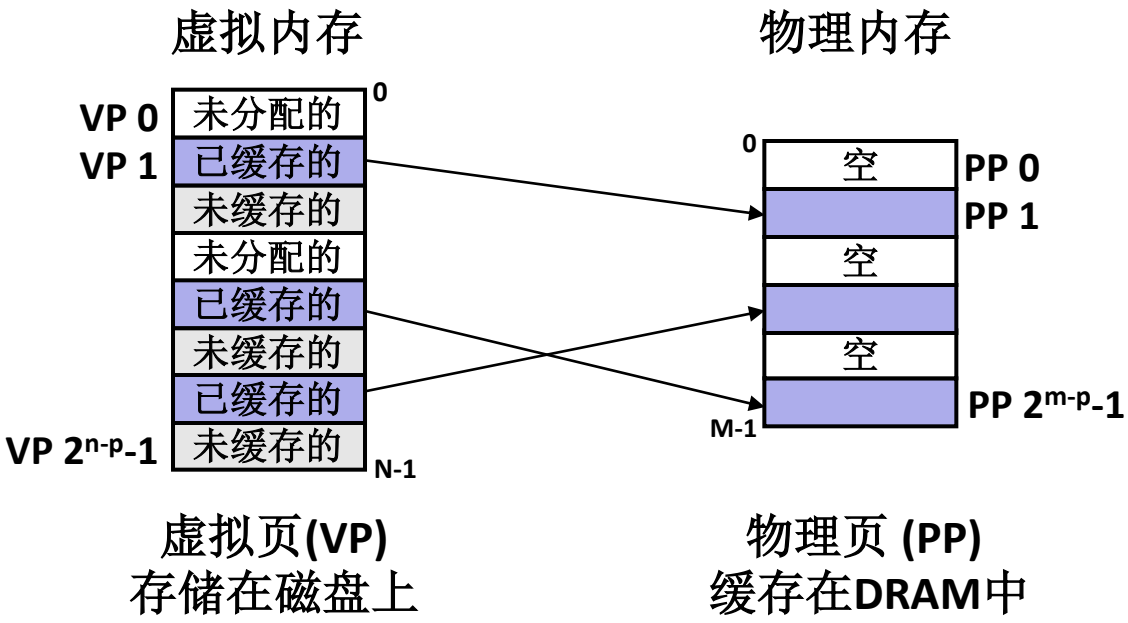
  虚拟内存机制为每个进程分配独立的虚拟地址空间，一个进程不能干涉其它进程的内存

主要内容

- 地址空间
- **虚拟内存作为缓存的工具**
- 虚拟内存作为内存管理的工具
- 虚拟内存作为内存保护的工具
- 地址翻译

虚拟内存作为缓存的工具

- 概念上而言，虚拟内存可以视为一个由存放在磁盘上的N个连续的字节单元组成的数组
- 此数组的内容被从磁盘缓存至物理内存（DRAM缓存）
  - 这些缓存块被称为页（每页的大小为P = 2^p字节）

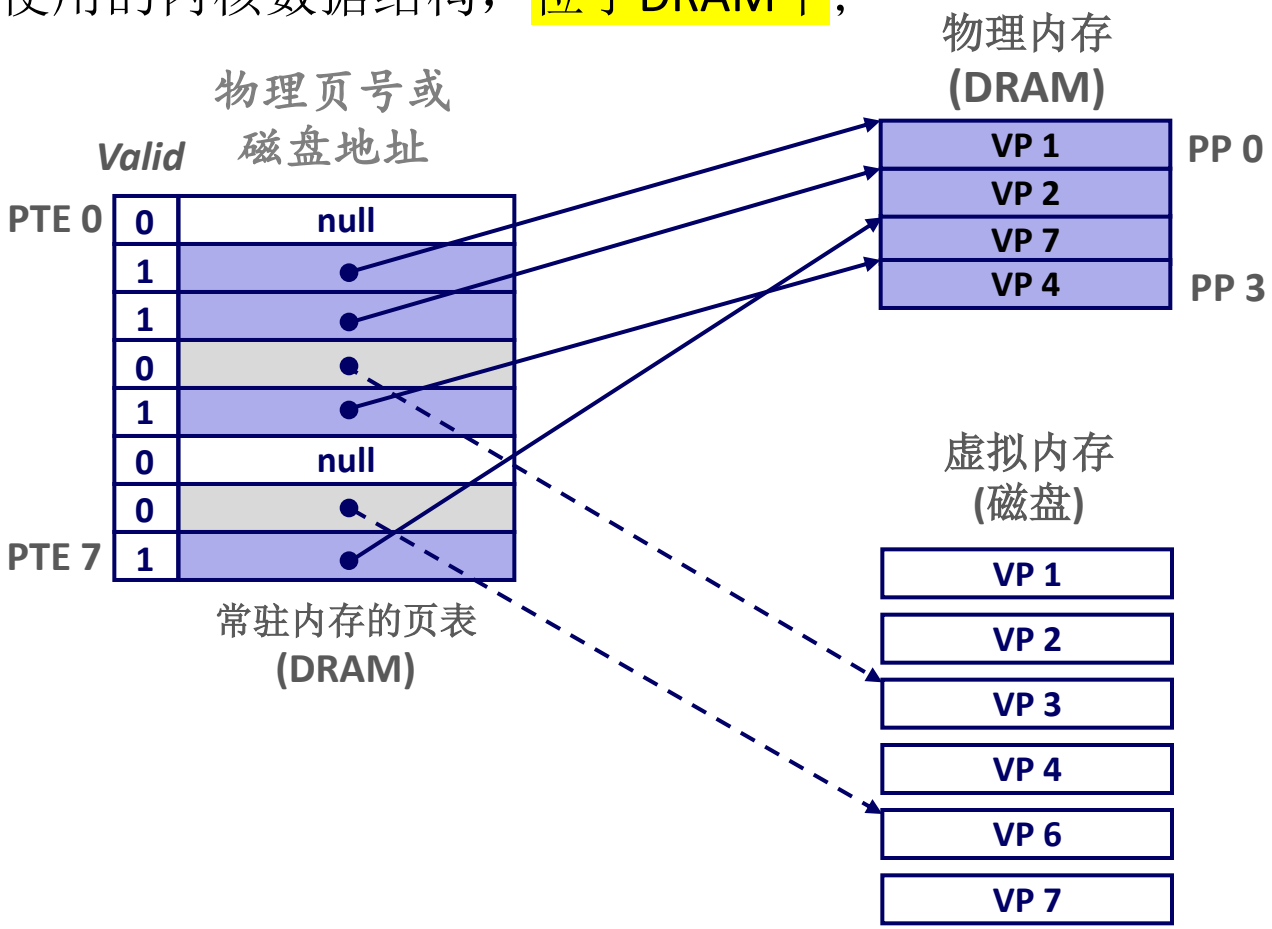


DRAM Cache的组织

- **DRAM 缓存的组织形式源自巨大的不命中开销：**
  - DRAM 比 SRAM 慢大约 **10倍**
  - 磁盘比 DRAM 慢大约 **10,000倍**
- **DRAM 缓存设计：**
  - 较大的页面尺寸：标准 4 KB，有时可达 4 MB
  - **全相联映像**
    - 任何虚拟页可以放置在任何物理页中
    - 需要一个“更大的”映像函数——不同于SRAM缓存
  - 更复杂精密的、花费更高的替换算法
    - 过于复杂和开放，以致无法在硬件上实现
  - DRAM缓存总是使用写回法，从不采用直写法

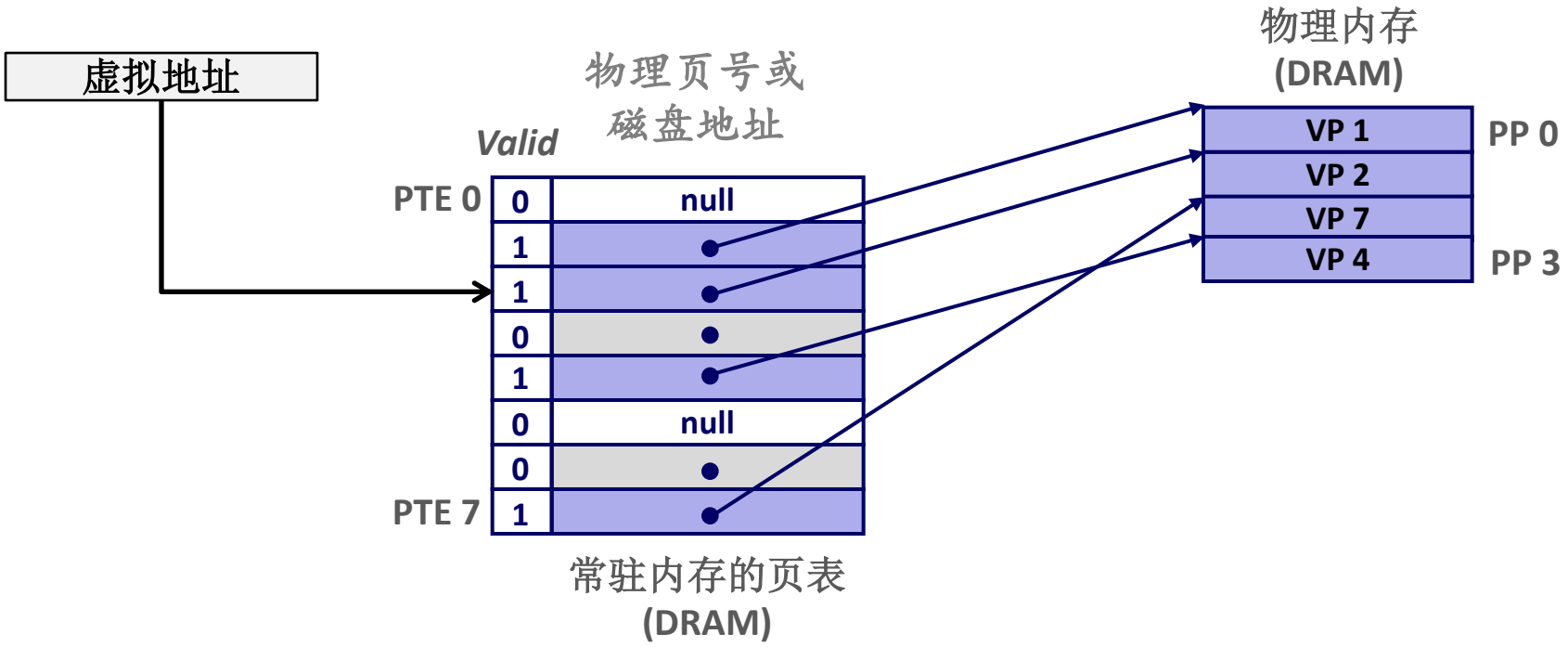
需要用到的数据结构：页表page table

- 页表是一个由页表条目PTE (Page Table Entry)构成的数组，将虚拟页映像到物理页。
  - 每个进程都使用的内核数据结构，位于DRAM中；



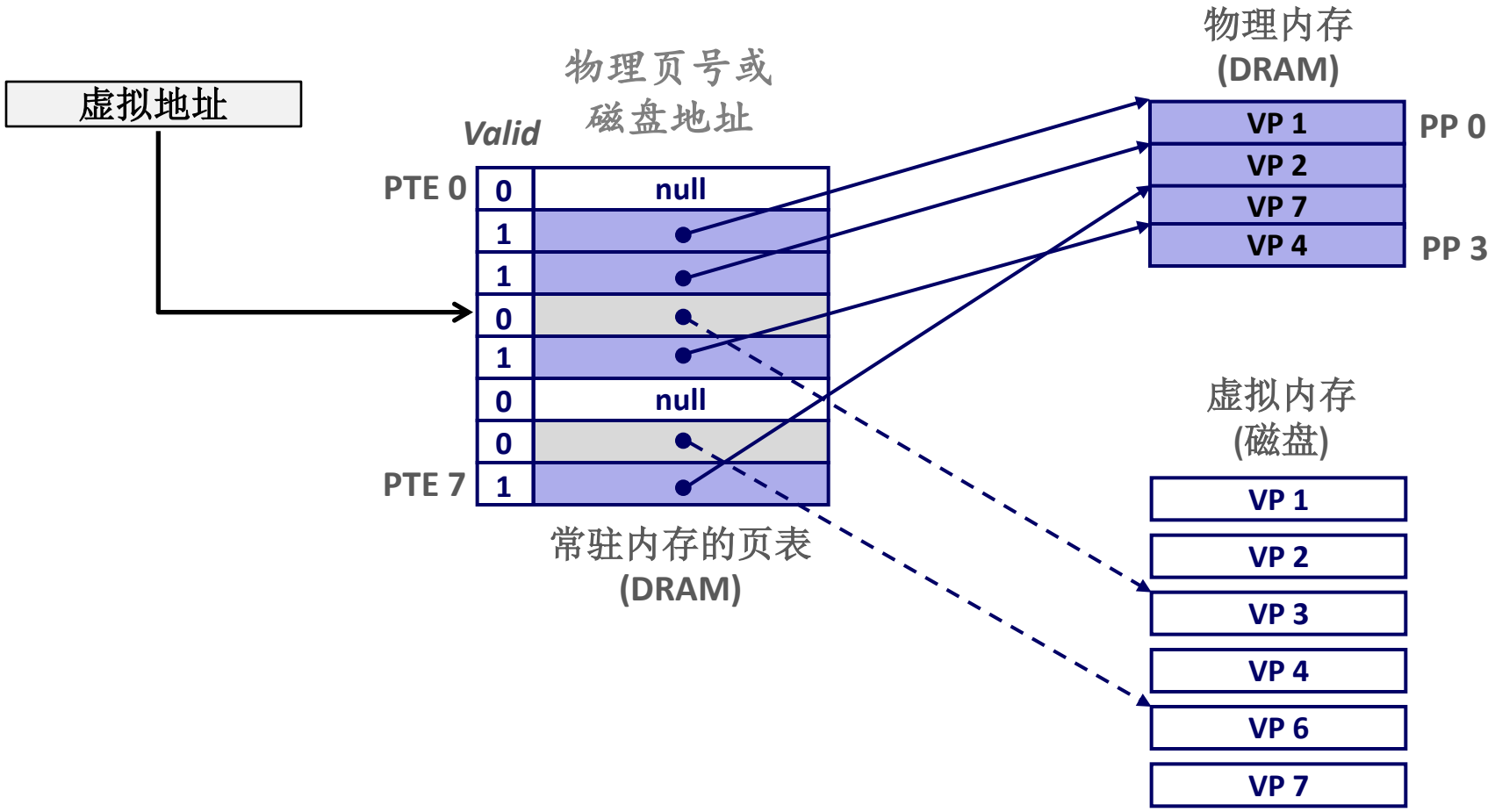
页命中

- **Page hit页命中**：虚拟内存中待访问的字存在于物理内存中（即DRAM缓存命中）



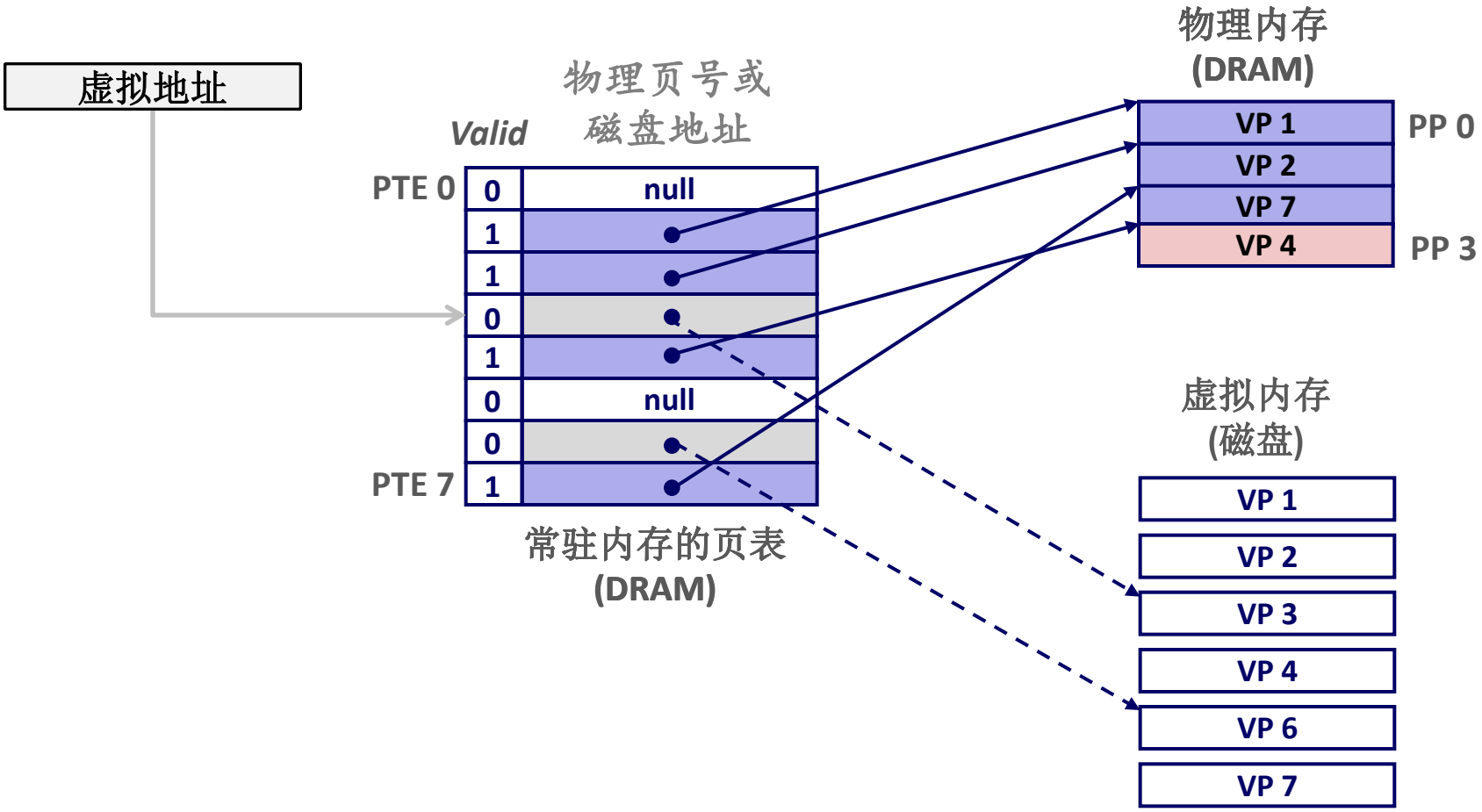
缺页

- **Page fault缺页：** 虚拟内存中待访问的字不在物理内存中（即DRAM缓存不命中）



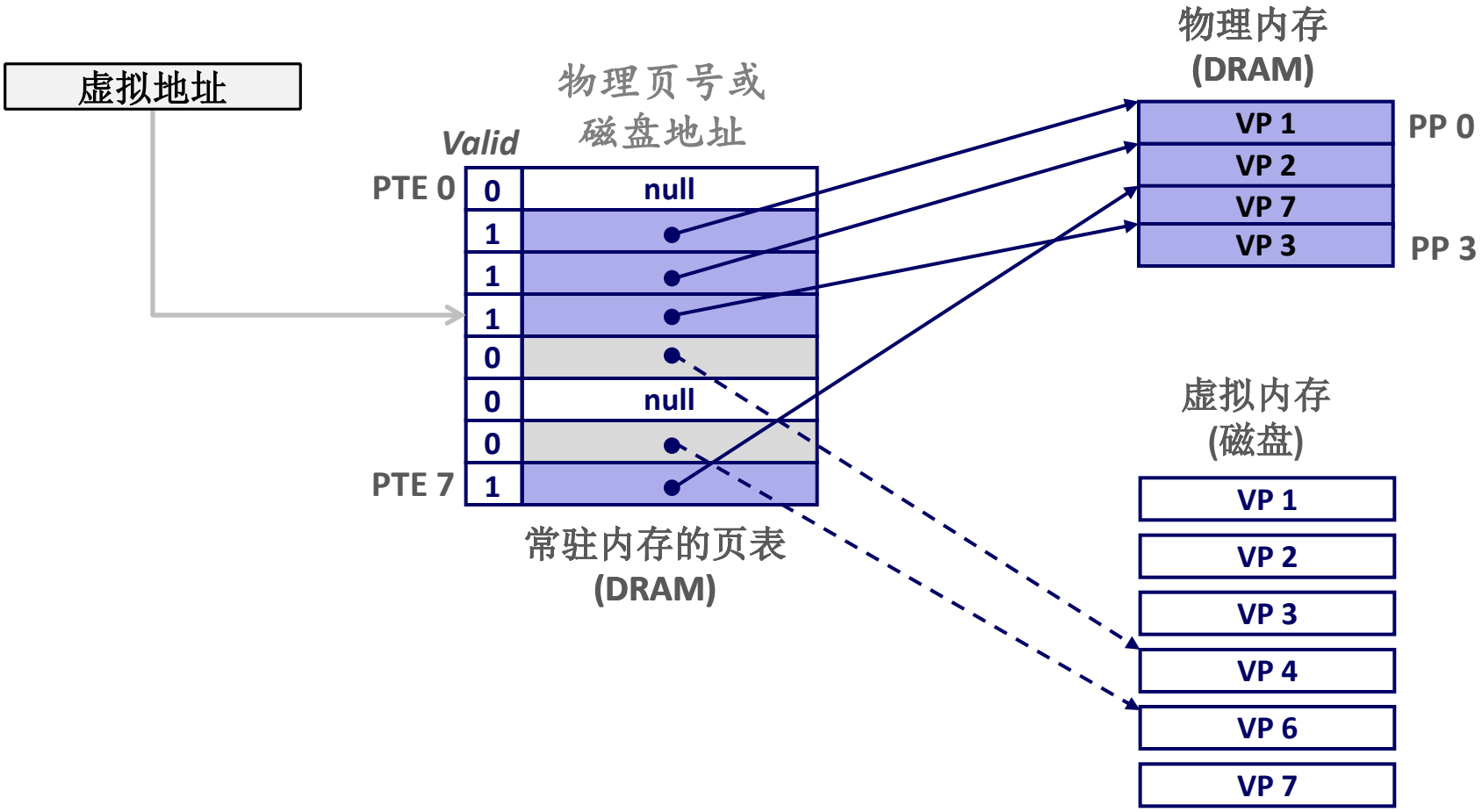
缺页的处理

- 缺页引发缺页故障（一种异常）
- 缺页异常处理程序选择一个牺牲页victim page（本例为VP 4）



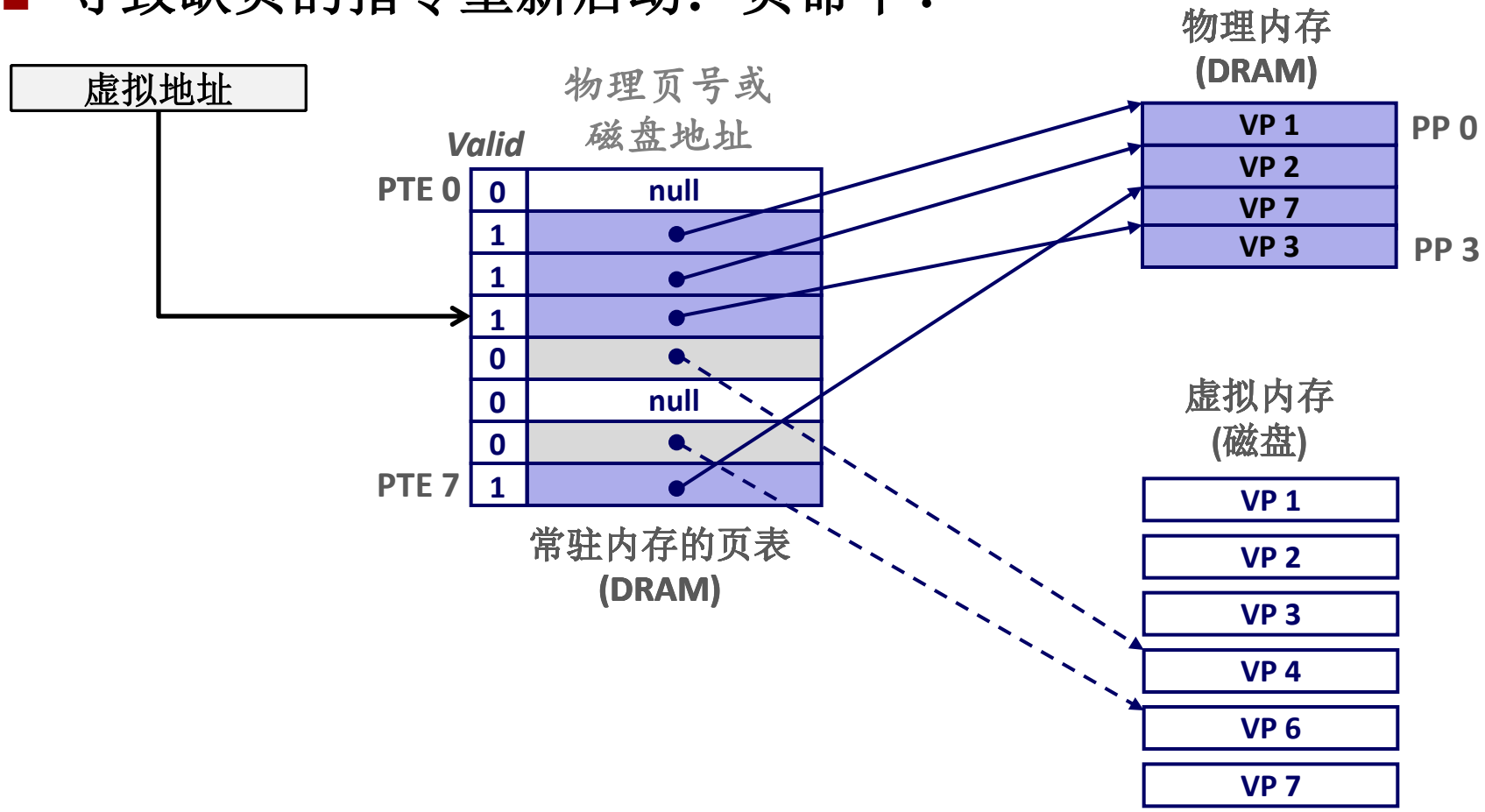
缺页的处理（续）

- 缺页引发缺页故障（一种异常）
- 缺页异常处理程序选择一个牺牲页victim page（本例为VP 4）



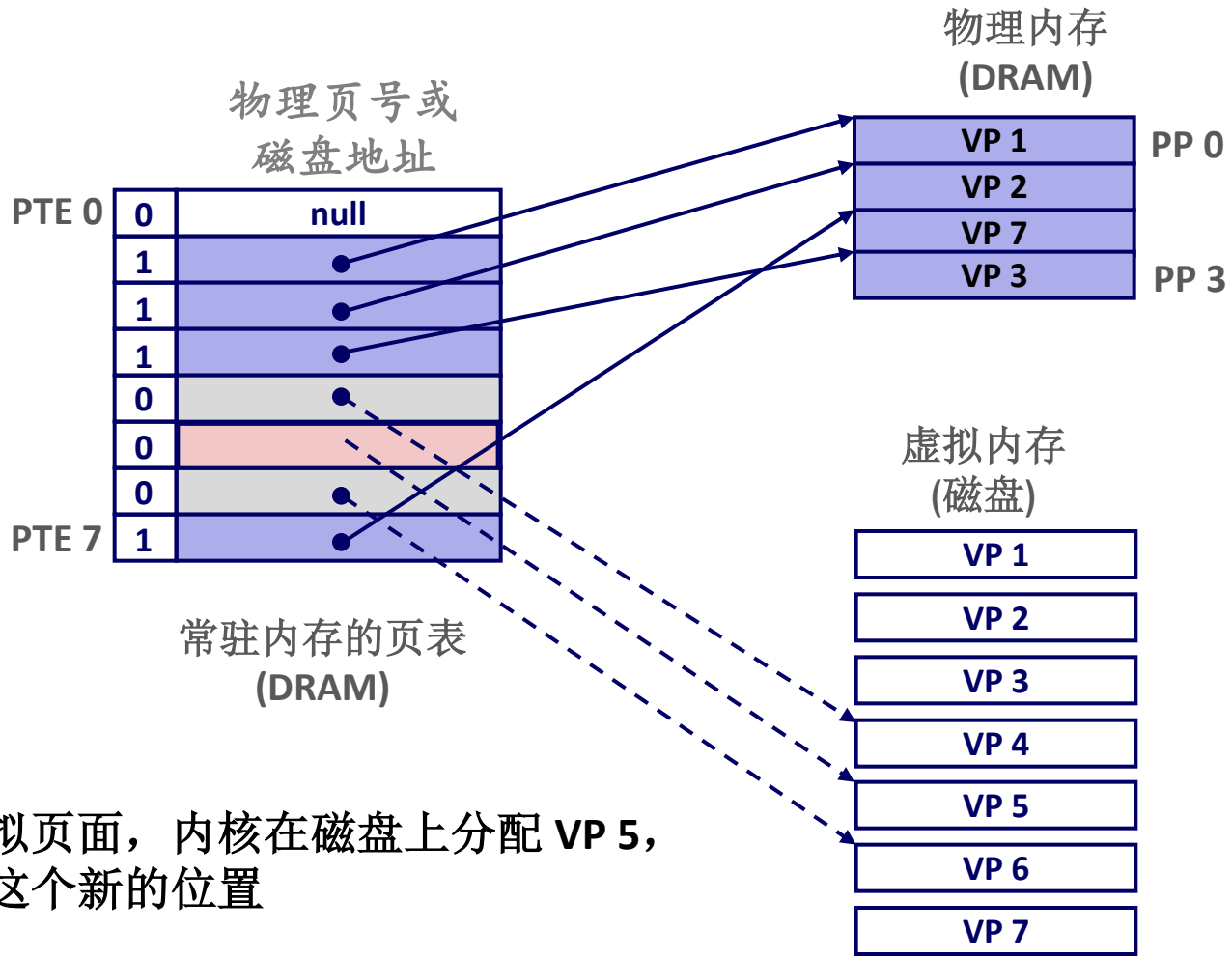
缺页的处理（续）

- 缺页引发缺页故障（一种异常）
- 缺页异常处理程序选择一个牺牲页victim page（本例为VP 4）
- 导致缺页的指令重新启动：页命中！



页面的分配

- 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 VP 5，
并且将PTE5指向这个新的位置

Locality to the Rescue Again!

局部性再一次成为救世主！

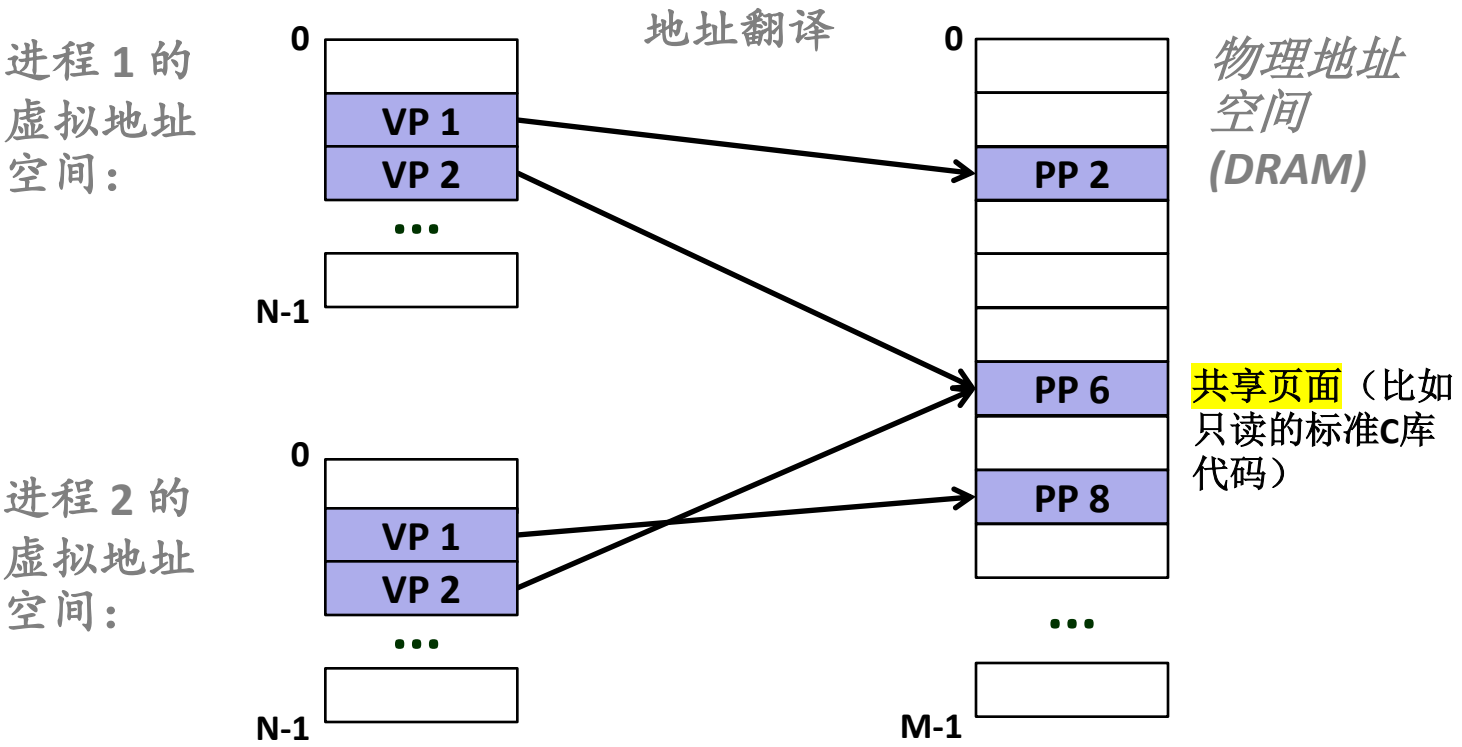
- 虚拟内存机制看上去非常低效，但事实上它运行得相当好，这都要归功于“局部性”
- 在任意时间，程序总是趋向于在一个较小的活动页面集合上执行，这个集合叫做工作集working set
  - 程序的时间局部性越好，工作集就会越小
- 如果工作集的大小 < 物理内存容量
  - 在强制不命中开销过后，对工作集的引用将导致命中
- 如果SUM(各工作集的大小) > 物理内存容量
  - 抖动thrashing：页面不断地换进换出，导致系统性能崩溃

主要内容

- 地址空间
- 虚拟内存作为缓存的工具
- **虚拟内存作为内存管理的工具**
- 虚拟内存作为内存保护的工具
- 地址翻译

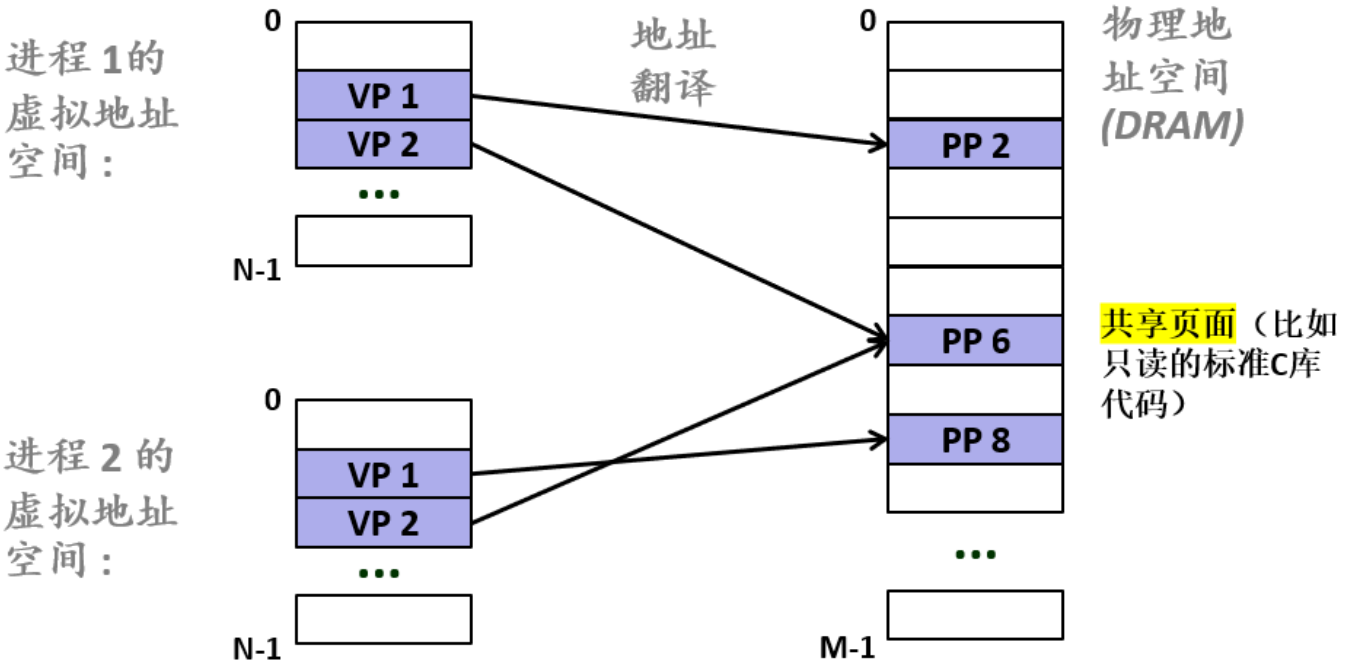
虚拟内存作为内存管理的工具

- 关键思想：每个进程都拥有一个独立的虚拟地址空间
  - 各进程可以将内存视为一个简单的线性数组；
  - 映射函数将众多虚拟地址分散在物理内存的各处；



虚拟内存作为内存管理的工具

- 简化内存分配
  - 任意一个虚拟内存页可以被映像至任意一个物理内存页
  - 同一个虚拟内存页每次被分配到的物理内存页也可以不同
- 在进程之间共享代码和数据
  - 不同的虚拟内存页面被映像到同一个物理内存页（此例为 PP 6）。

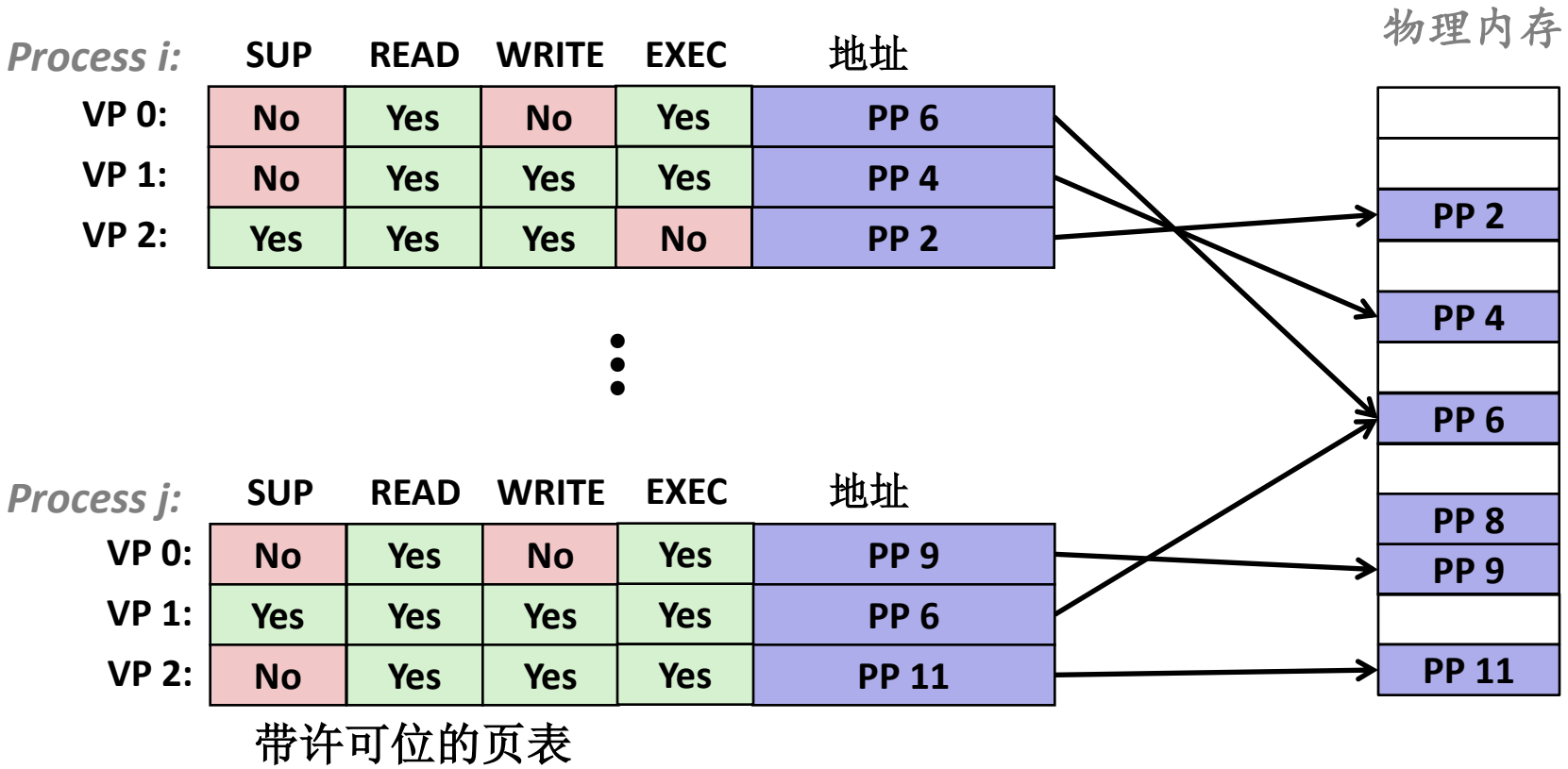


主要内容

- 地址空间
- 虚拟内存作为缓存的工具
- 虚拟内存作为内存管理的工具
- **虚拟内存作为内存保护的工具**
- 地址翻译

虚拟内存作为内存保护的工具

- 向PTE中添加若干许可位，以提供更好的访问控制
- 内存管理单元每次访问数据都要检查权限位
- 越权访问导致内核发送SIGSEGV，Linux报segmentation fault。



带许可位的页表

主要内容

- 地址空间
- 虚拟内存作为缓存的工具
- 虚拟内存作为内存管理的工具
- 虚拟内存作为内存保护的工具
- **地址翻译**

VM地址翻译

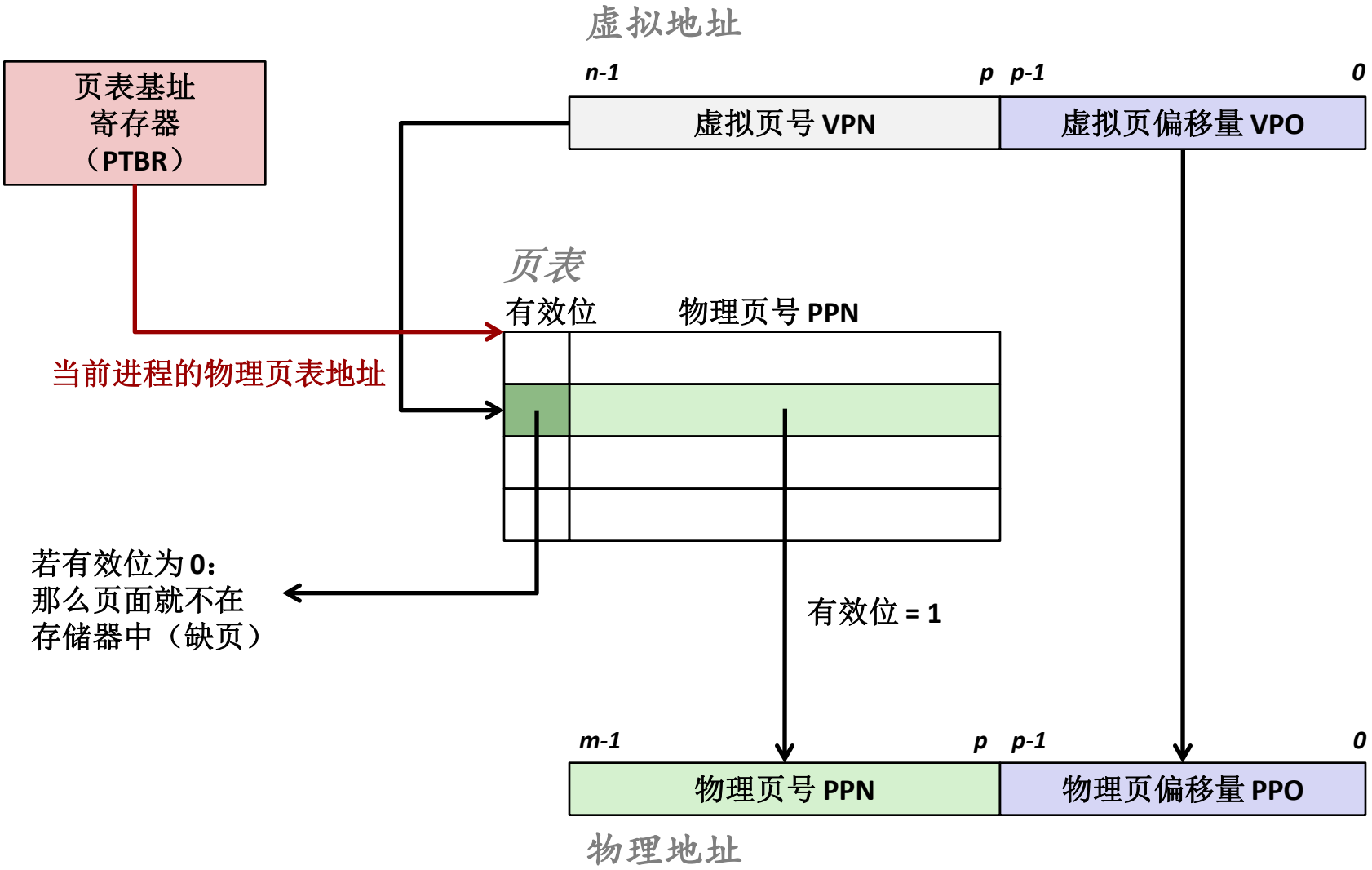
- 虚拟地址空间
  - *V = {0, 1, …, N–1}*
- 物理地址空间
  - *P = {0, 1, …, M–1}*
- 地址翻译
  - ***MAP: V → P U {∅}***
  - 对于某一虚拟地址a：
    - ***MAP(a) = a'*** 如果虚拟地址 ***a***处的数据在***p***的物理地址 ***a'*** 处；
    - ***MAP(a) = ∅*** 如果虚拟地址 ***a***处的数据不在物理内存中；
      - 或者为无效地址，或者仍存储在磁盘上。

地址翻译使用到的所有符号

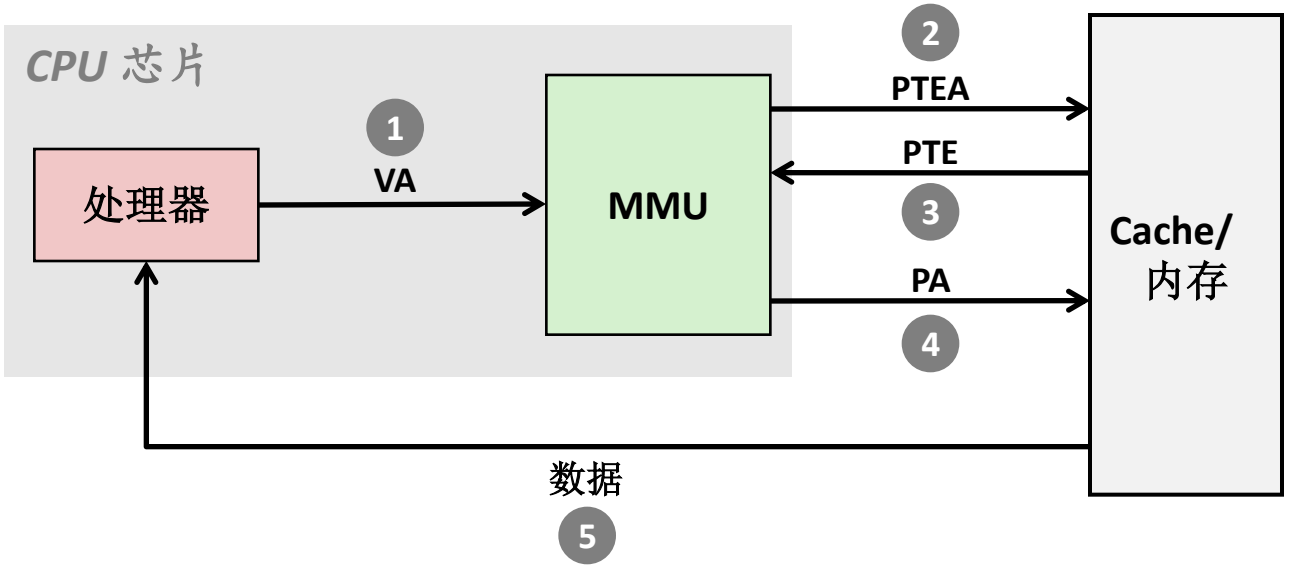
- 基本参数
  - **N = 2^n**：虚拟地址空间中的地址数量
  - **M = 2^m**：物理地址空间中的地址数量
  - **P = 2^p** ：页的大小（bytes)
- 虚拟地址**VA**的组成部分
  - **TLBI**：TLB索引 （用于查找TLB中的某一项)
  - **TLBT**：TLB标记 （用于TLB命中判断)
  - **VPO**：虚拟页面偏移量（字节）
  - **VPN**：虚拟页号
- 物理地址**PA**的组成部分
  - **PPO**：物理页面偏移量（同VPO）
  - **PPN**：物理页号

虚拟地址 (VA)	虚拟页号 (VPN)	页内偏移 (VPO)	物理页号 (PPN)	物理地址 (PA)
0x12345ABC	0x12345	0xABC	0x01A3	0x01A3ABC

基于页表的地址翻译


虚拟地址
n-1
p
p-1
0
虚拟页号 VPN
虚拟页偏移量 VPO
页表基址
寄存器
（PTBR）
页表
有效位
物理页号 PPN
当前进程的物理页表地址
若有效位为 0：
那么页面就不在
存储器中（缺页）
有效位 = 1
m-1
p
p-1
0
物理页号 PPN
物理页偏移量 PPO
物理地址

地址翻译：页面命中



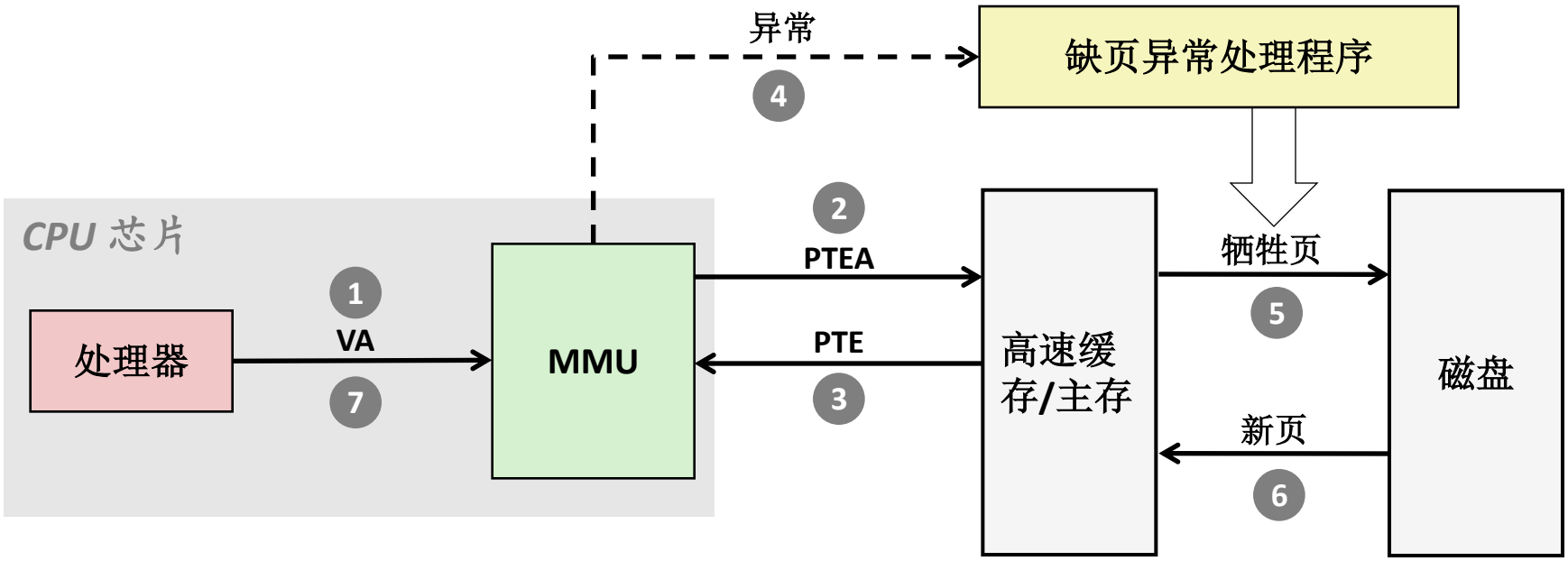
1) 处理器生成一个虚拟地址，并将其传送至**内存管理单元**MMU

2-3) MMU生成PTE地址，进而从位于Cache/内存的页表当中获取PTE

4) MMU 据此建立物理地址，再将物理地址发给Cache/主存

5) Cache/主存返回所请求的数据字给处理器。

地址翻译：缺页故障



1)处理器生成一个虚拟地址，并将其传送至MMU

2-3) MMU生成PTE地址，进而从位于Cache/内存的页表当中获取PTE

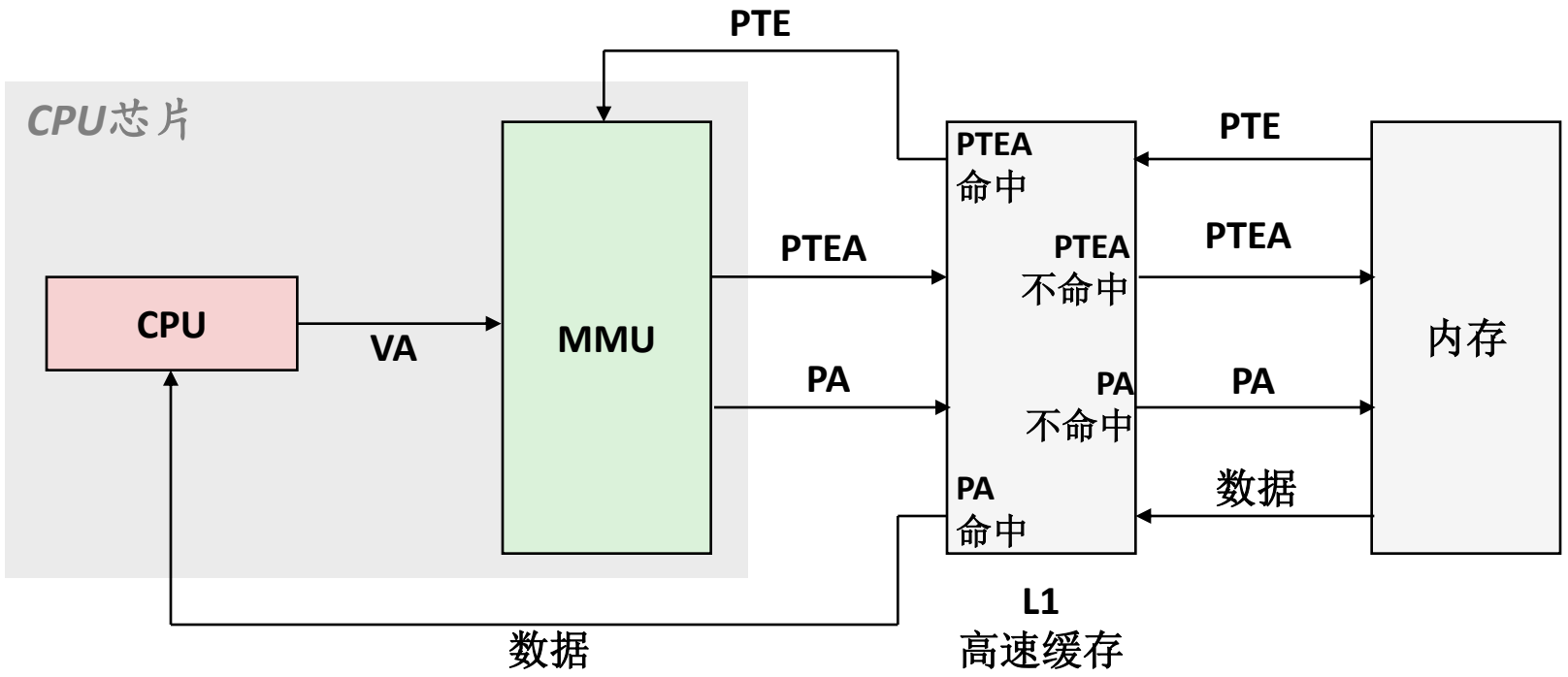
4) MMU发现有效位为零，便触发缺页故障异常

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) 缺页处理程序将控制流返回至原进程，重新执行缺页的指令

Cache和虚拟内存的结合



VA：虚拟地址　　*PA*：物理地址

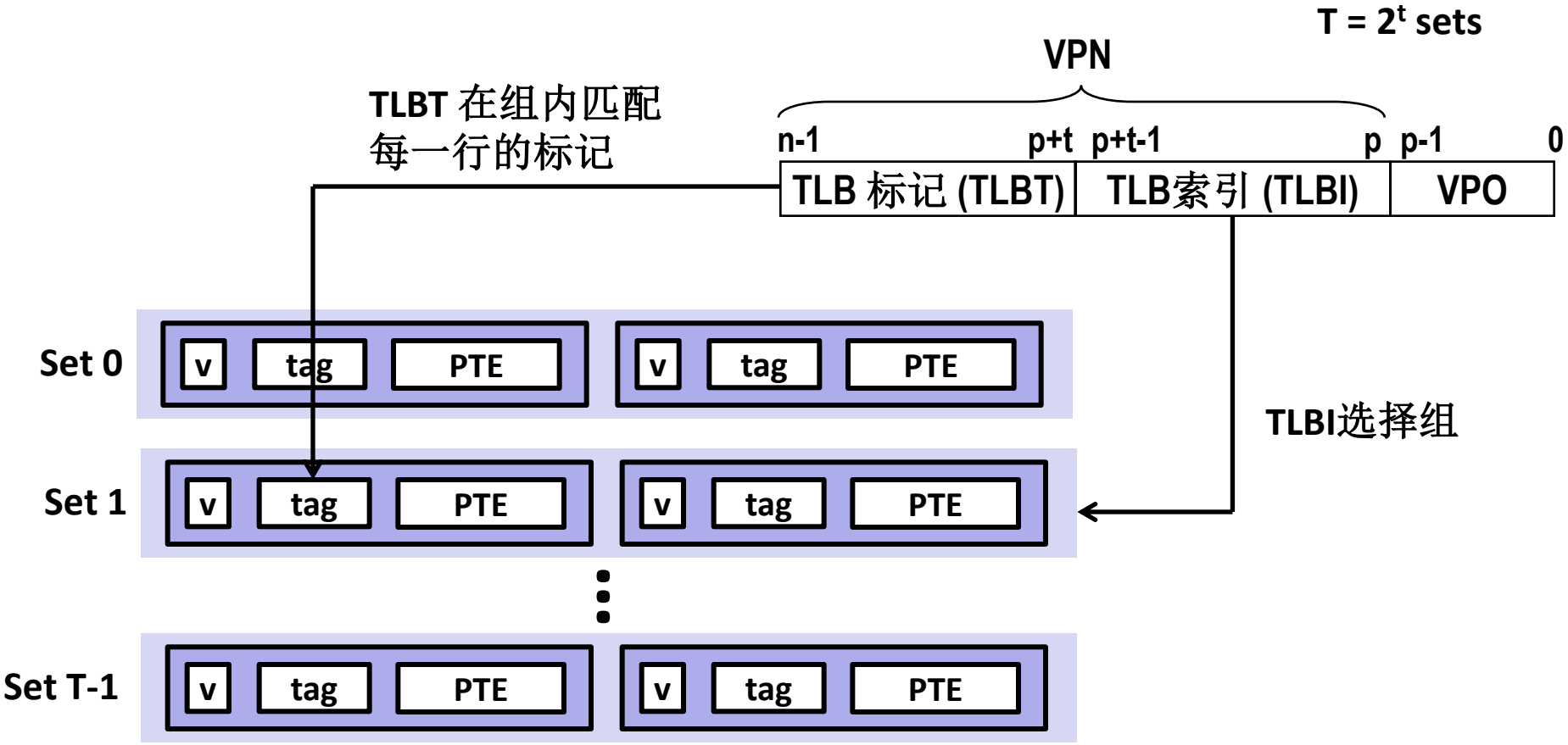
PTE：页表条目　　*PTEA*：页表条目地址

利用TLB加速地址翻译

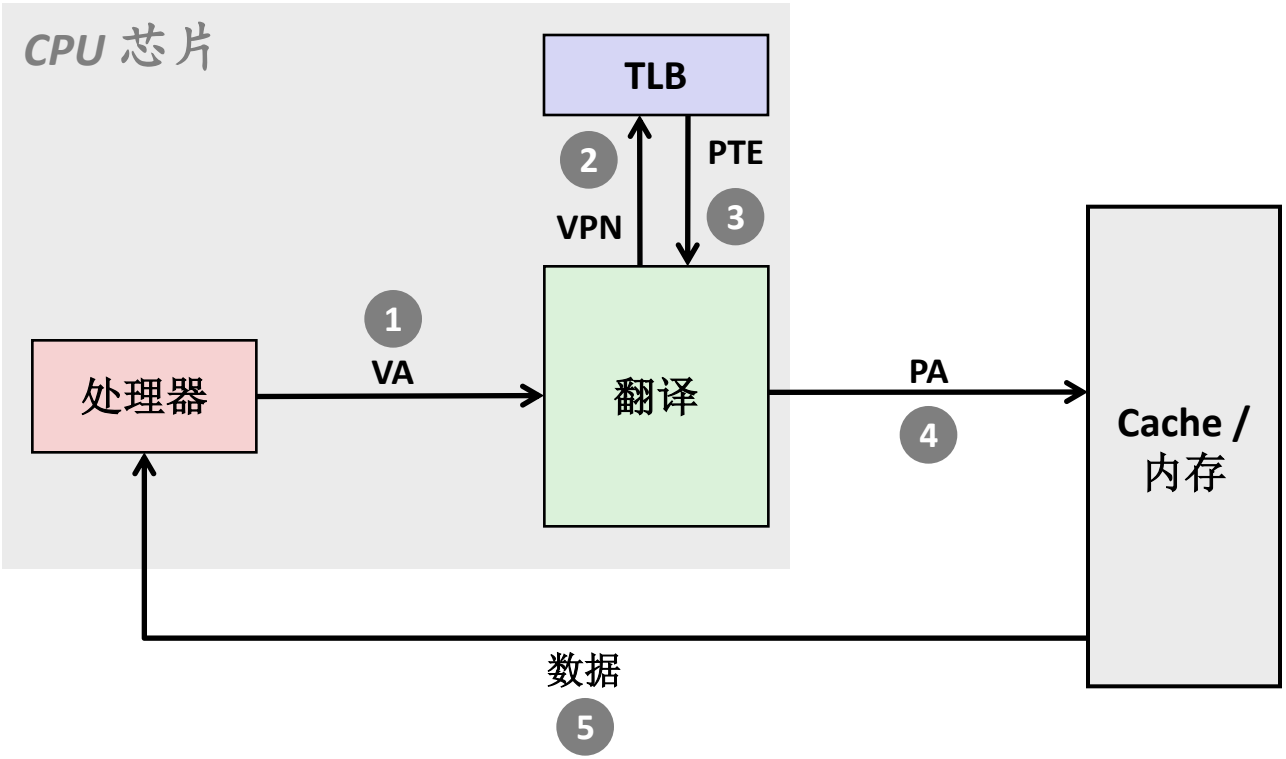
- 各页表项PTE最差在内存，较好在L1
  - 但PTE仍可能被其它数据引用驱逐出Cache
- 解决方法：TLB
  - Translation Lookaside Buffer翻译后备缓冲器
  - 又称快表（内存中的普通页表相应称作慢表）
  - MMU中的一个小容量、高相联度、零延迟的Cache
  - 实现**虚拟页号**向**物理页号**的映射
  - 对于页码数很少的页表可以完全包含在TLB中

对TLB的访问

- MMU使用虚拟地址的VPN部分访问TLB：

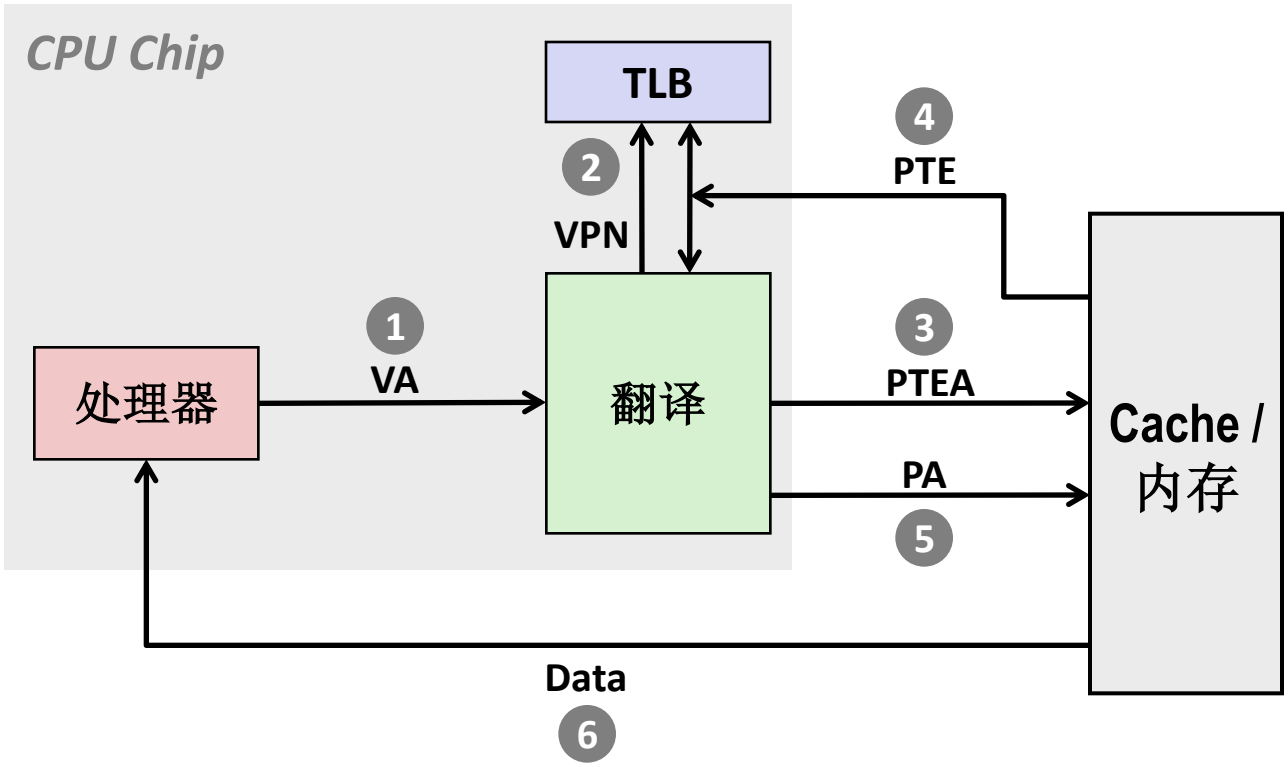


TLB命中



TLB 命中省去一次Cache/内存访问

TLB 不命中

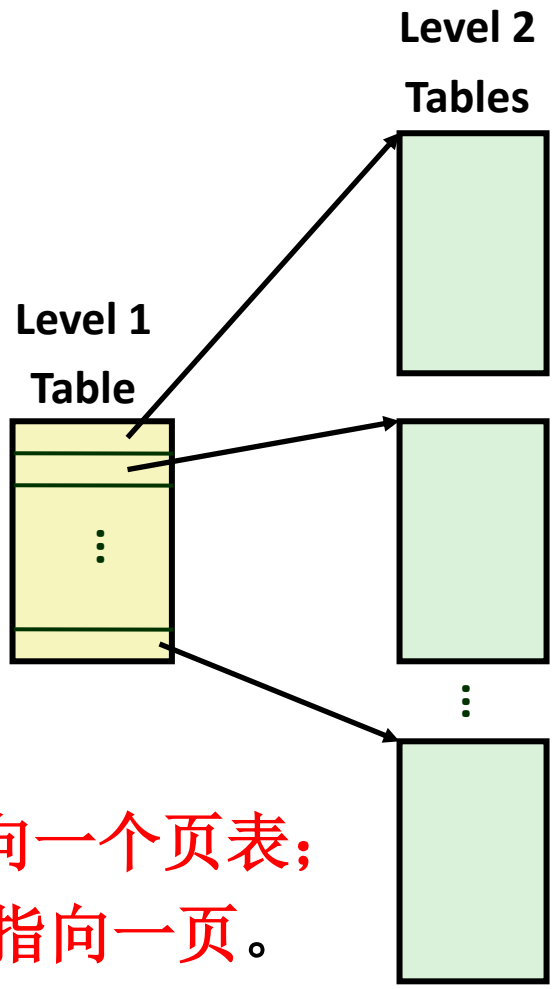


TLB不命中仍需进行Cache/内存访问

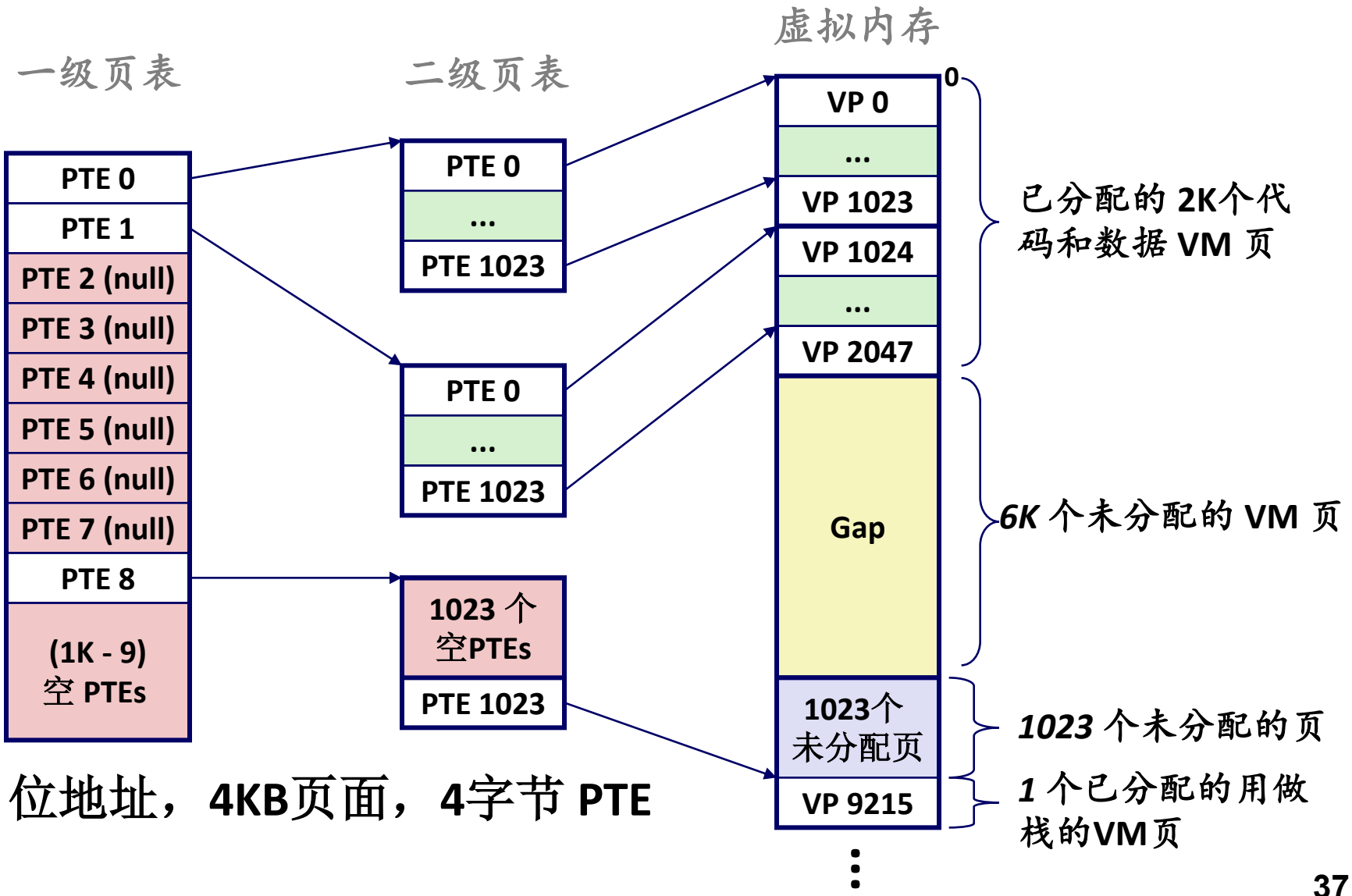
所幸TLB 不命中很少发生。原理是？

多级页表

- 假设：
  - 页面大小为4KB (2^{12})；
  - 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!
    - $2^{48} * 2^{-12} * 2^{3} = 2^{39}$ B
- 常用解决办法：多级页表
- 以两级页表为例：
  - 一级页表（常驻内存）：每个 PTE 指向一个页表；
  - 二级页表（可调入调出）：每个 PTE 指向一页。

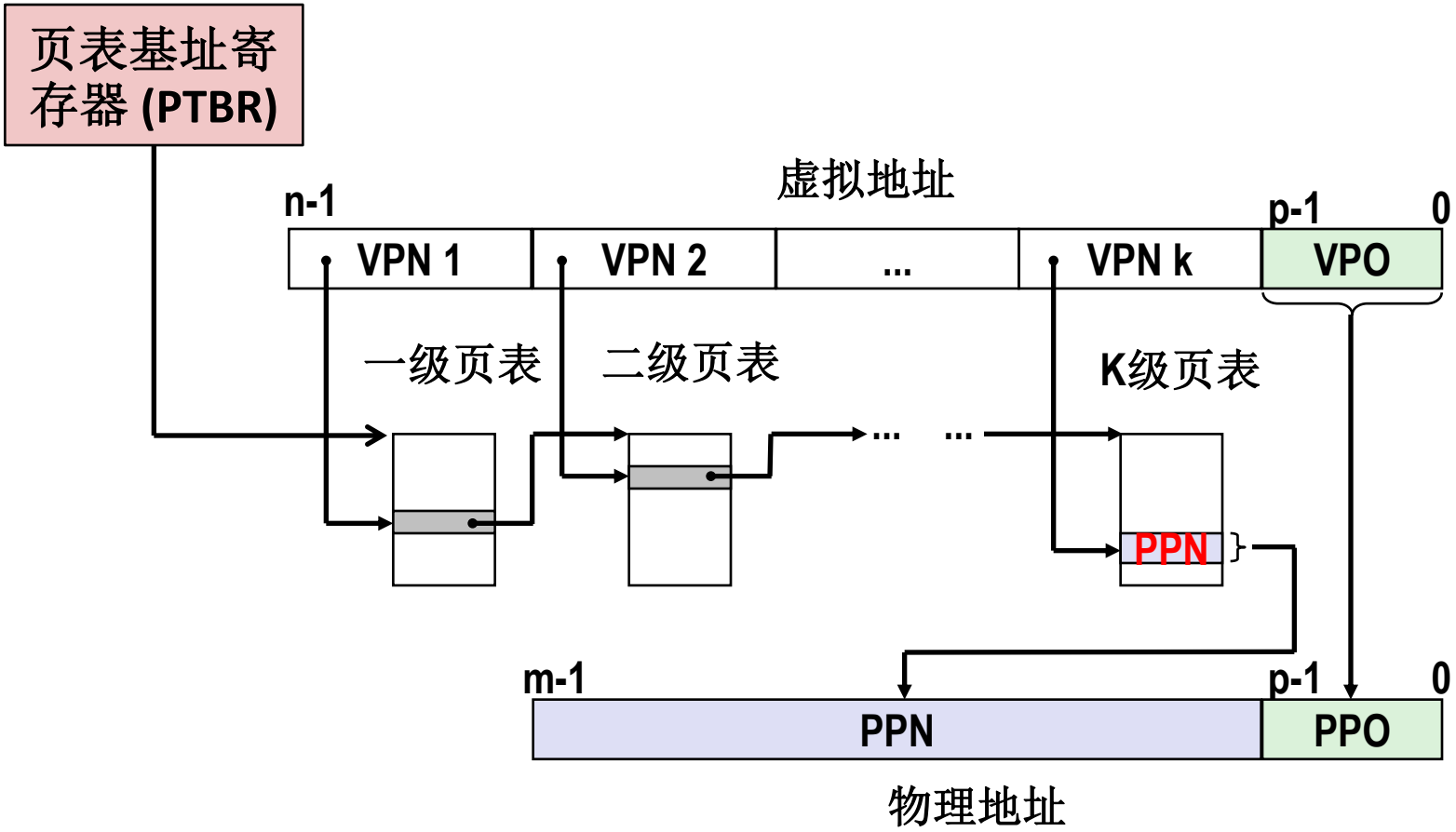


二级页表层次结构



32 位地址，4KB页面，4字节 PTE

使用K级页表的地址翻译



总结

- 以程序员的视角看虚拟内存：
  - 每个进程拥有自己的私有线性地址空间
  - 不允许被其它进程干扰
- 从系统的角度看待虚拟内存：
  - 通过缓存虚拟内存页面，从而高效利用内存
    - 高效率源自“局部性”