

第9章 虚拟内存: 基本概念

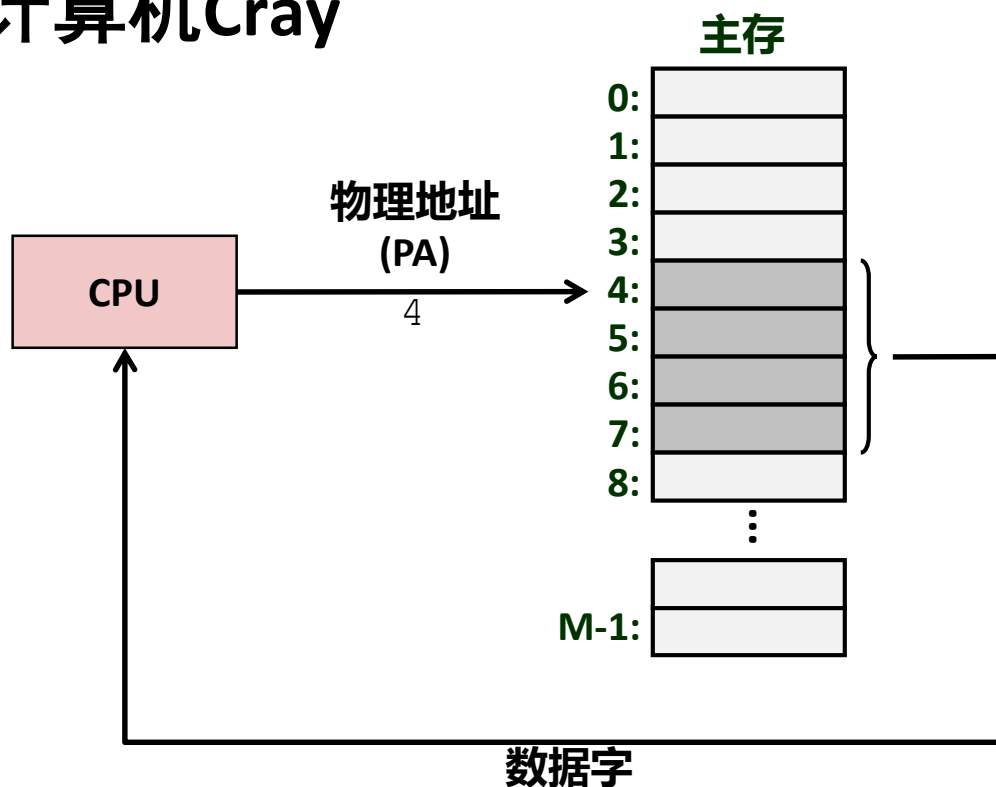
教 师: 郑贵滨
计算机科学与技术学院
哈尔滨工业大学

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

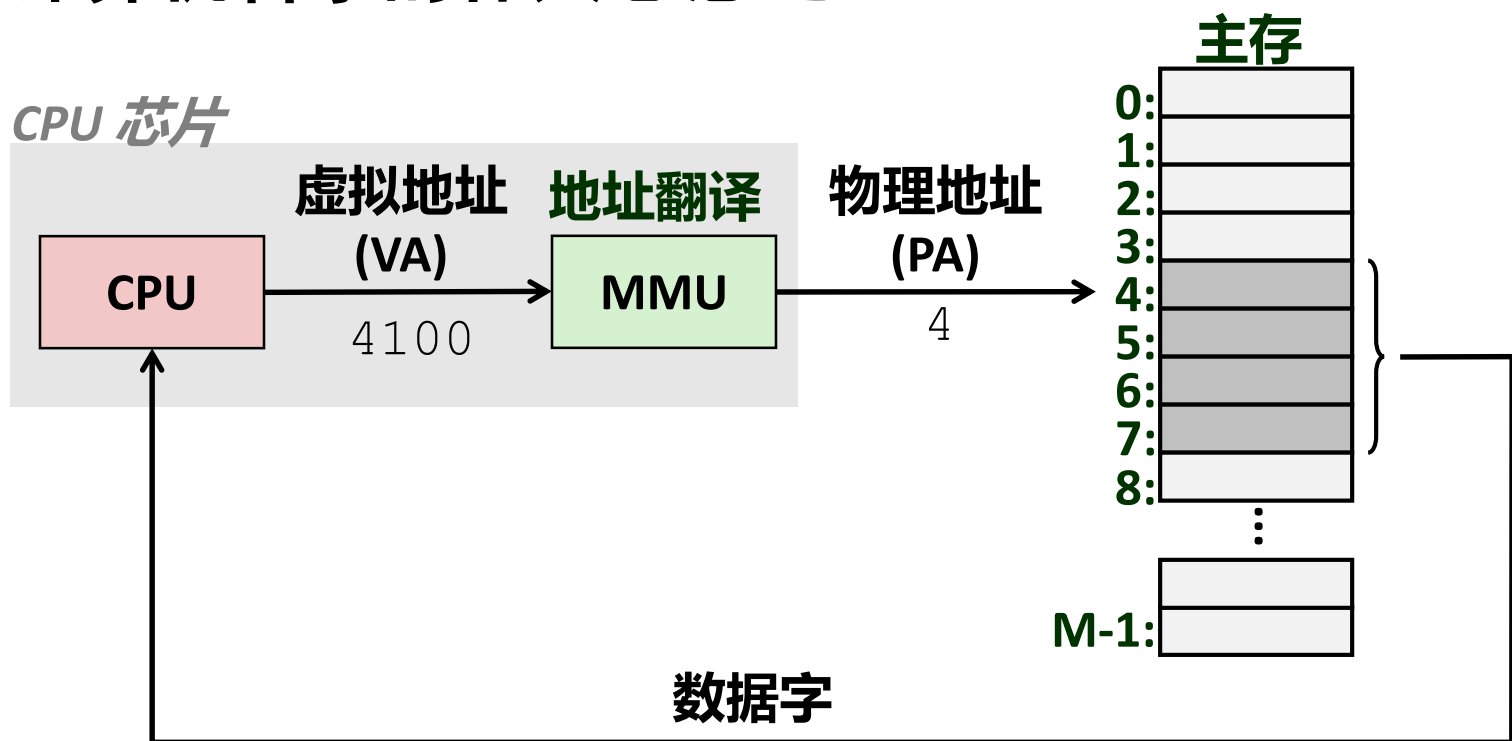
使用物理寻址的系统

- 使用嵌入式微控制器的“简单”系统：汽车、电梯、数字图像帧（digital picture frame）等
- 超级计算机Cray



使用虚拟寻址的系统

- 现代服务器、笔记本、智能电话等
- 计算机科学的伟大思想之一



MMU: 内存管理单元(Memory Management Unit)

地址空间

■ 地址空间 (address space)

非负整数地址的有序集合 $\{0, 1, 2, \dots\}$

■ 线性地址空间 (linear address space)

地址空间中的整数是连续的，则称为线性地址空间

■ 物理地址空间 (physical address sapce)

$M = 2^m$ 个物理地址的集合 $\{0, 1, 2, 3, \dots, M-1\}$

■ 虚拟地址空间 (virtual address space)

$N = 2^n$ 个虚拟地址的集合 $\{0, 1, 2, 3, \dots, N-1\}$

虚拟地址的思想： 允许每个数据对象有多个独立的地址，其中每个地址都选自一个不同的地址空间。

为什么要使用虚拟内存Virtual Memory (VM)?

■ 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

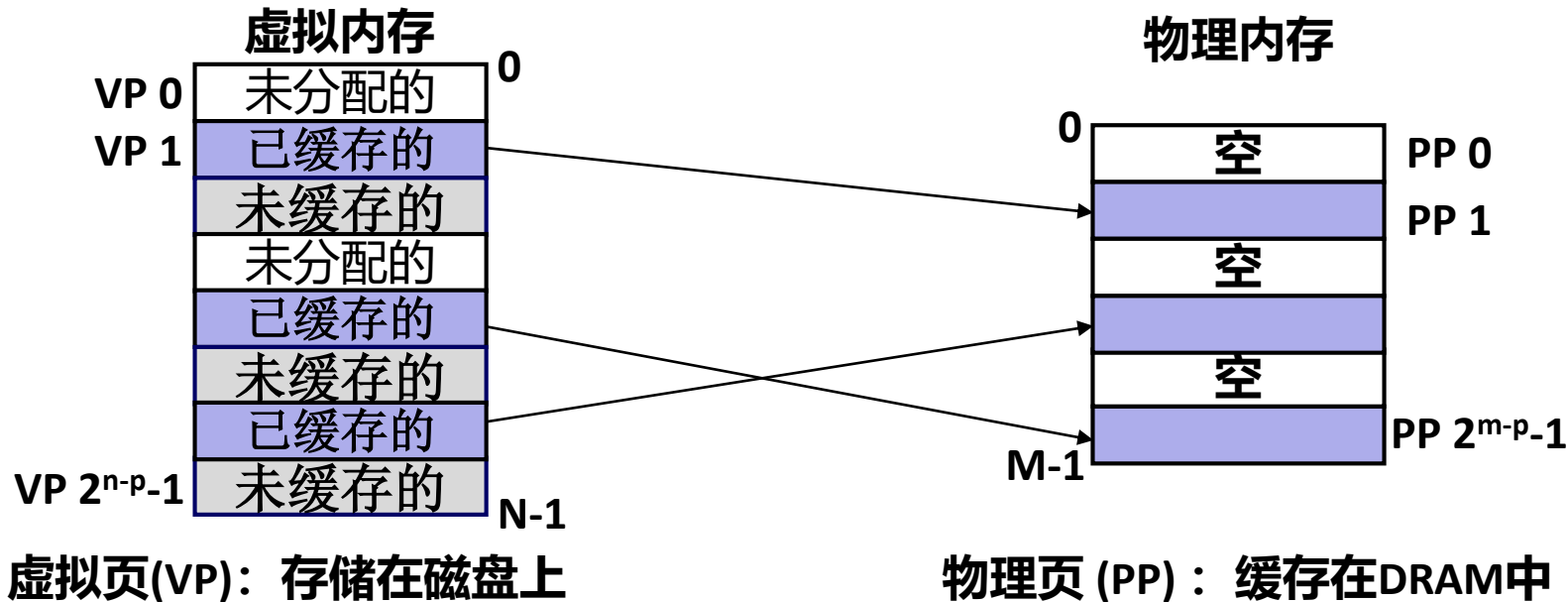
- 一个进程不能影响其他进程的内存
- 用户程序无法获取特权内核信息和代码

主要内容

- 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

- 虚拟内存：存放在**磁盘上**、有N个连续字节的数组。
- 磁盘上这个数组的内容被缓存在**物理内存中 (DRAM cache)**
 - 这些缓存块被称为页 (页面大小为 $P = 2^p$ 字节)



虚拟内存作为缓存的工具

■ 虚拟页面的三种状态：

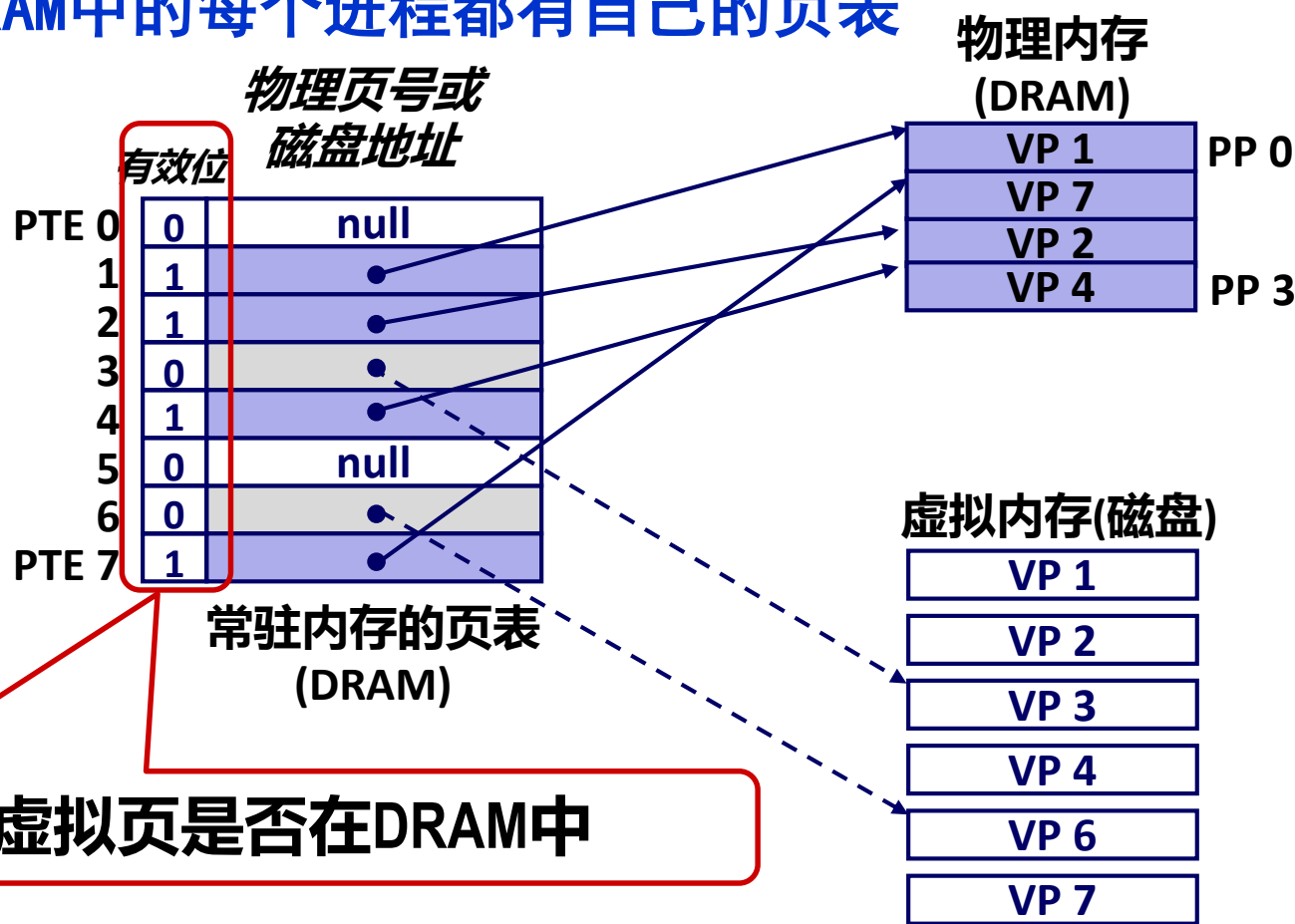
- (1) 缓存：已经缓存在物理内存中的已分配页。
- (2) 未缓存：未缓存在物理内存中的已分配页。
- (3) 未分配/创建：
 - 虚拟页也是根据需要创建的（节省资源）
 - VM系统还没有分配/创建的页，没有任何数据和它们关联，不占用任何磁盘空间。

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
 - 非常复杂、计算量大的替换算法
 - 太复、太开放, 不能在硬件上实现
 - DRAM缓存总是**使用写回, 而不是直写**

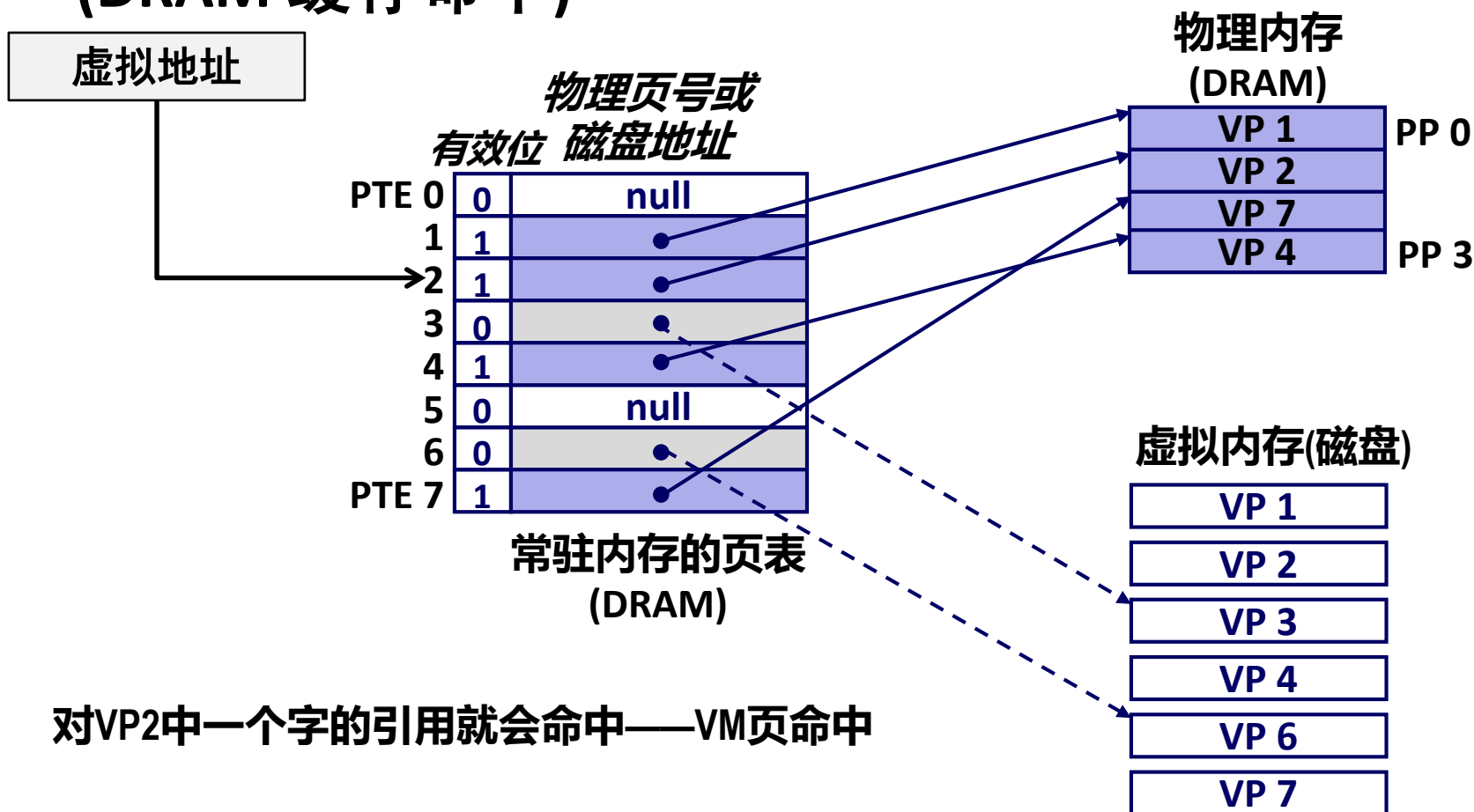
非常重要的数据结构：页表(Page Table)

- **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中的每个进程都有自己的页表**



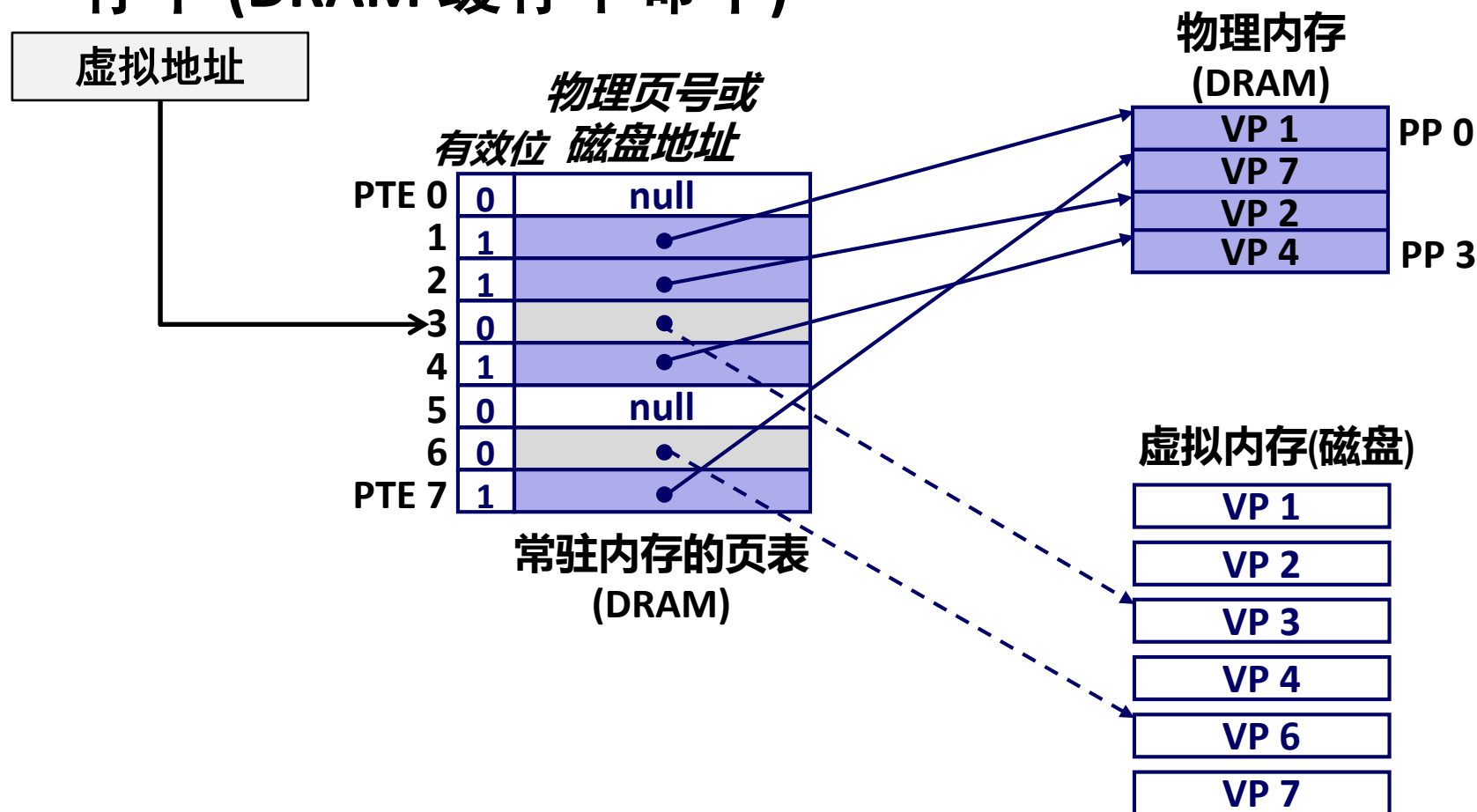
页命中(Page Hit)

- **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)



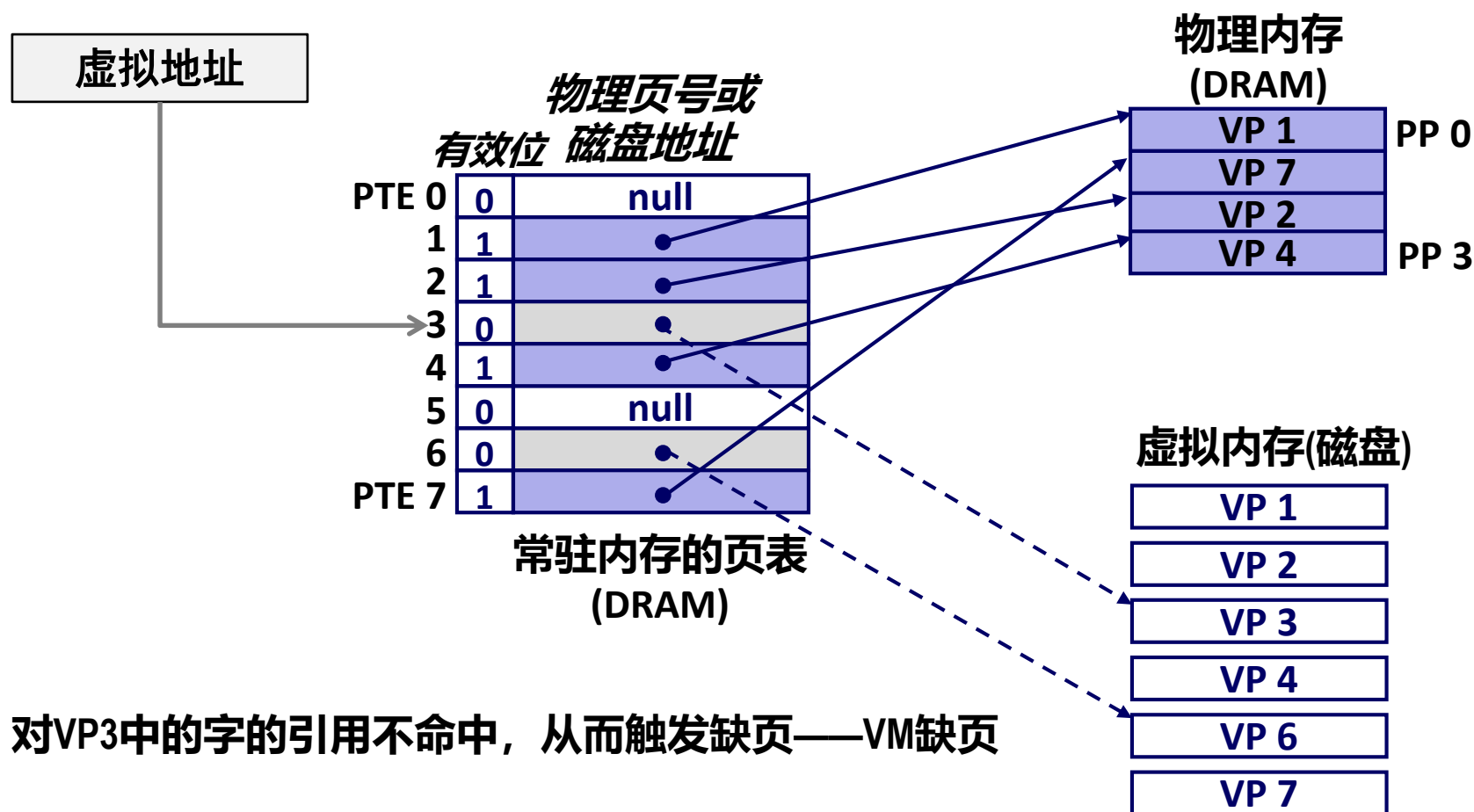
缺页(Page Fault)

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



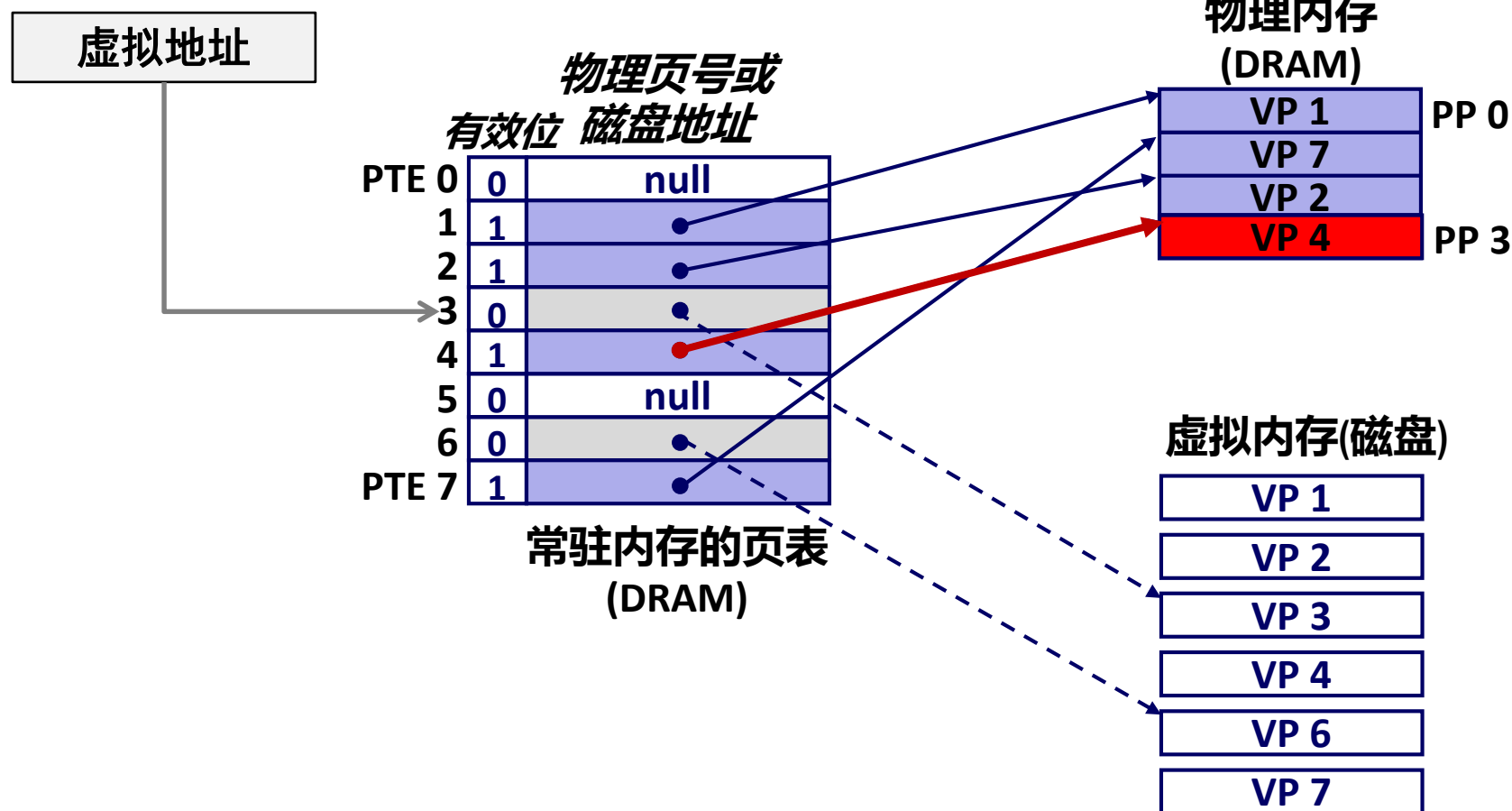
缺页的处理

- 页面不命中导致缺页(缺页异常)



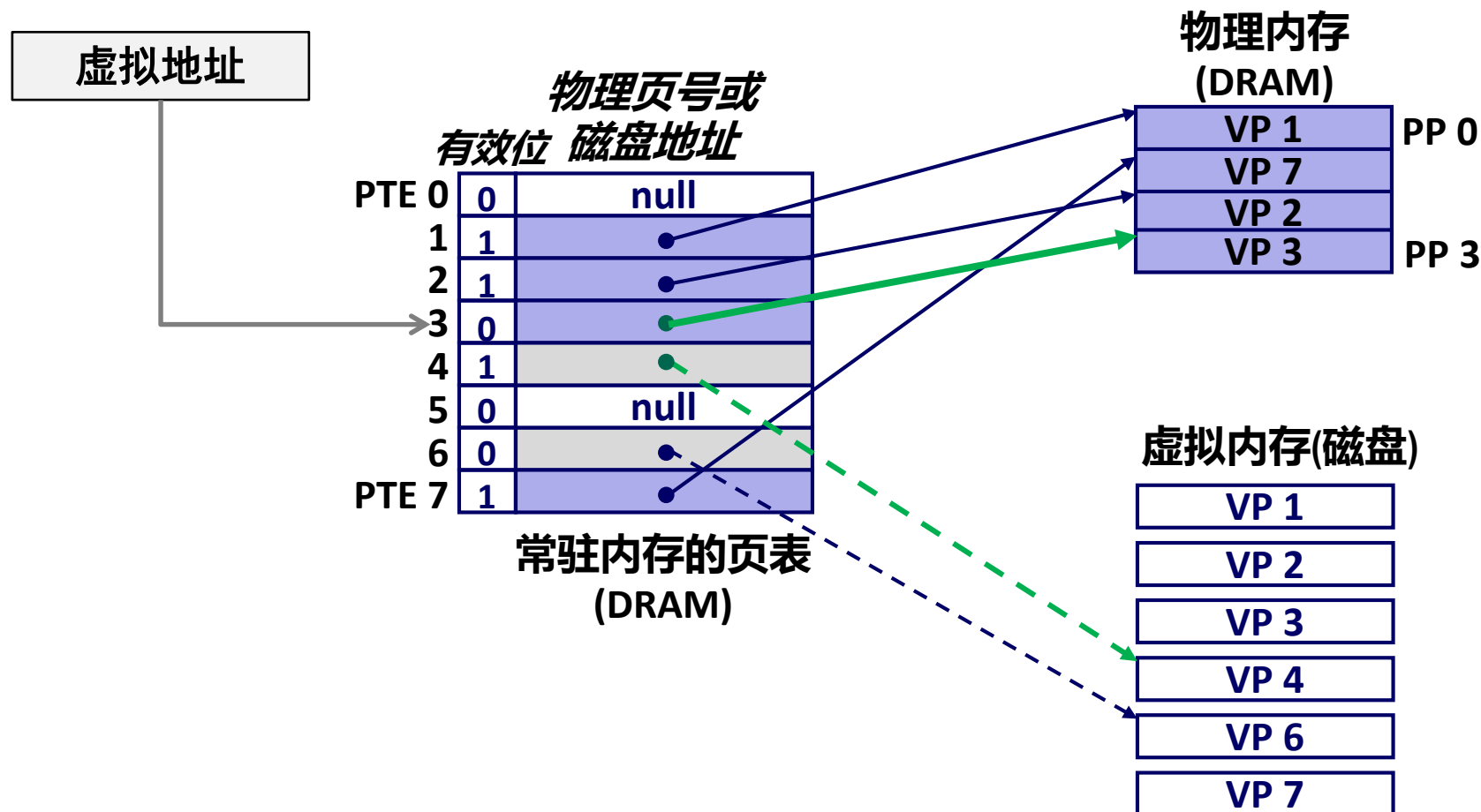
缺页的处理

- 页面不命中导致缺页(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)



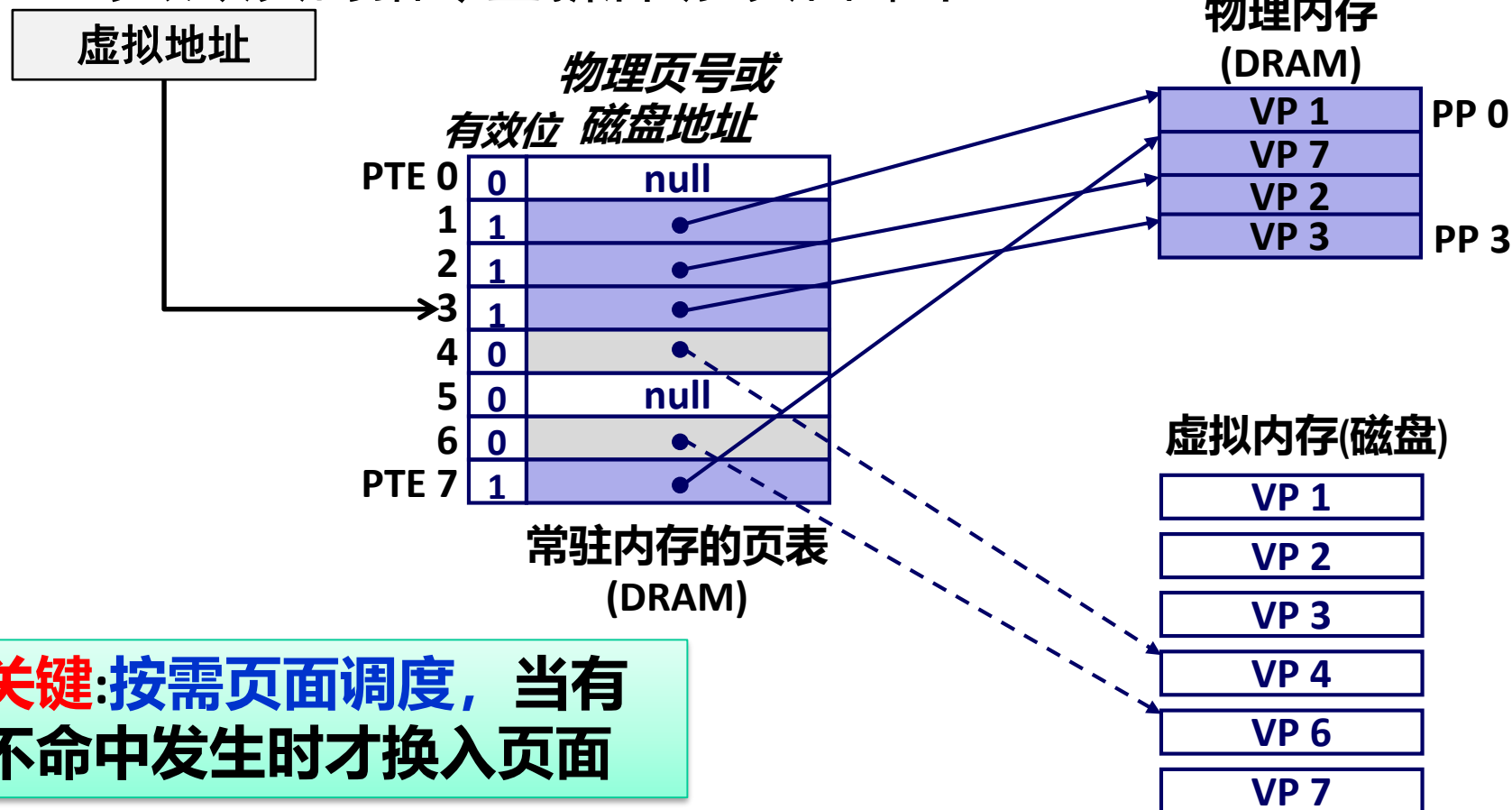
缺页的处理

- 页面不命中导致缺页(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)



缺页的处理

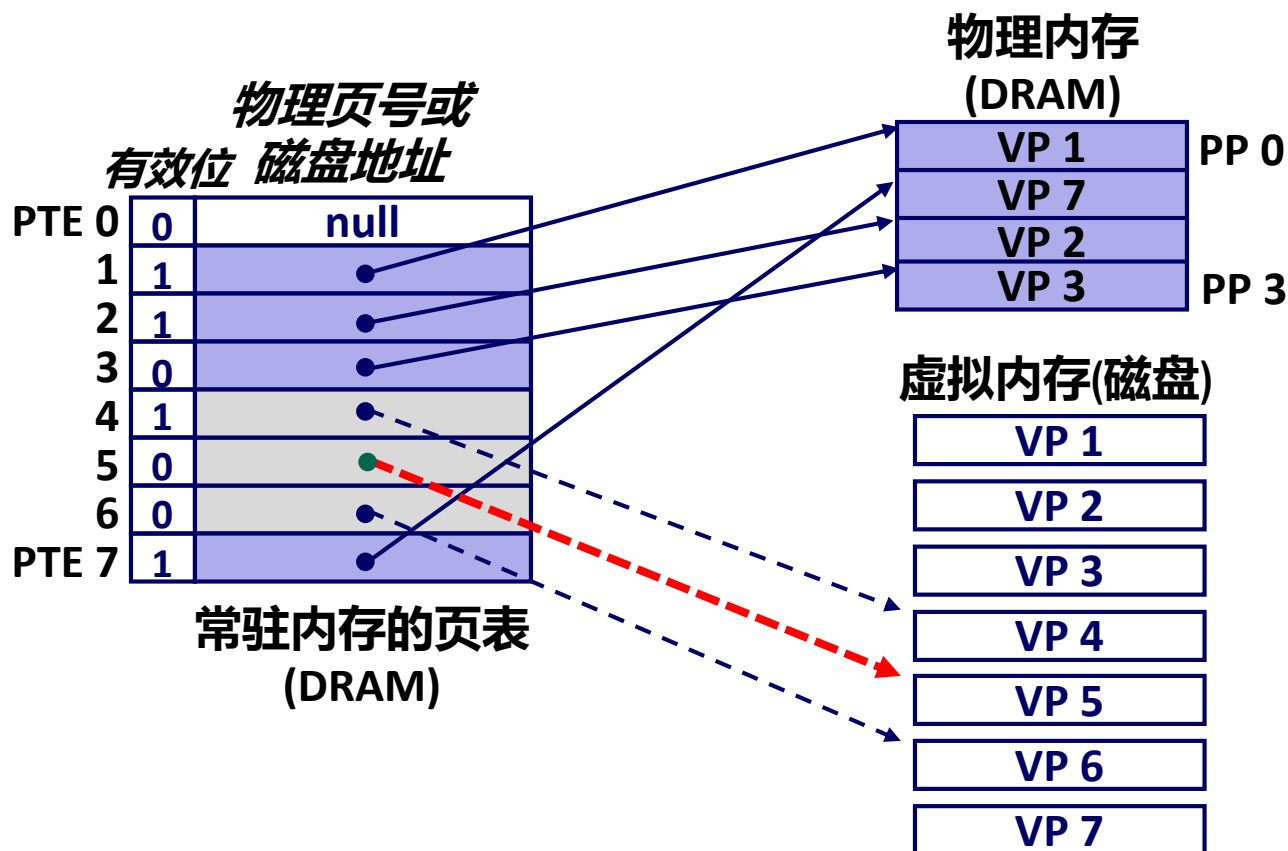
- 页面不命中导致缺页(缺页异常)
- 缺页异常处理程序选择一个牺牲页 (此例中就是 VP 4)
- 导致缺页的指令重新启动: 页面命中!



分配页面

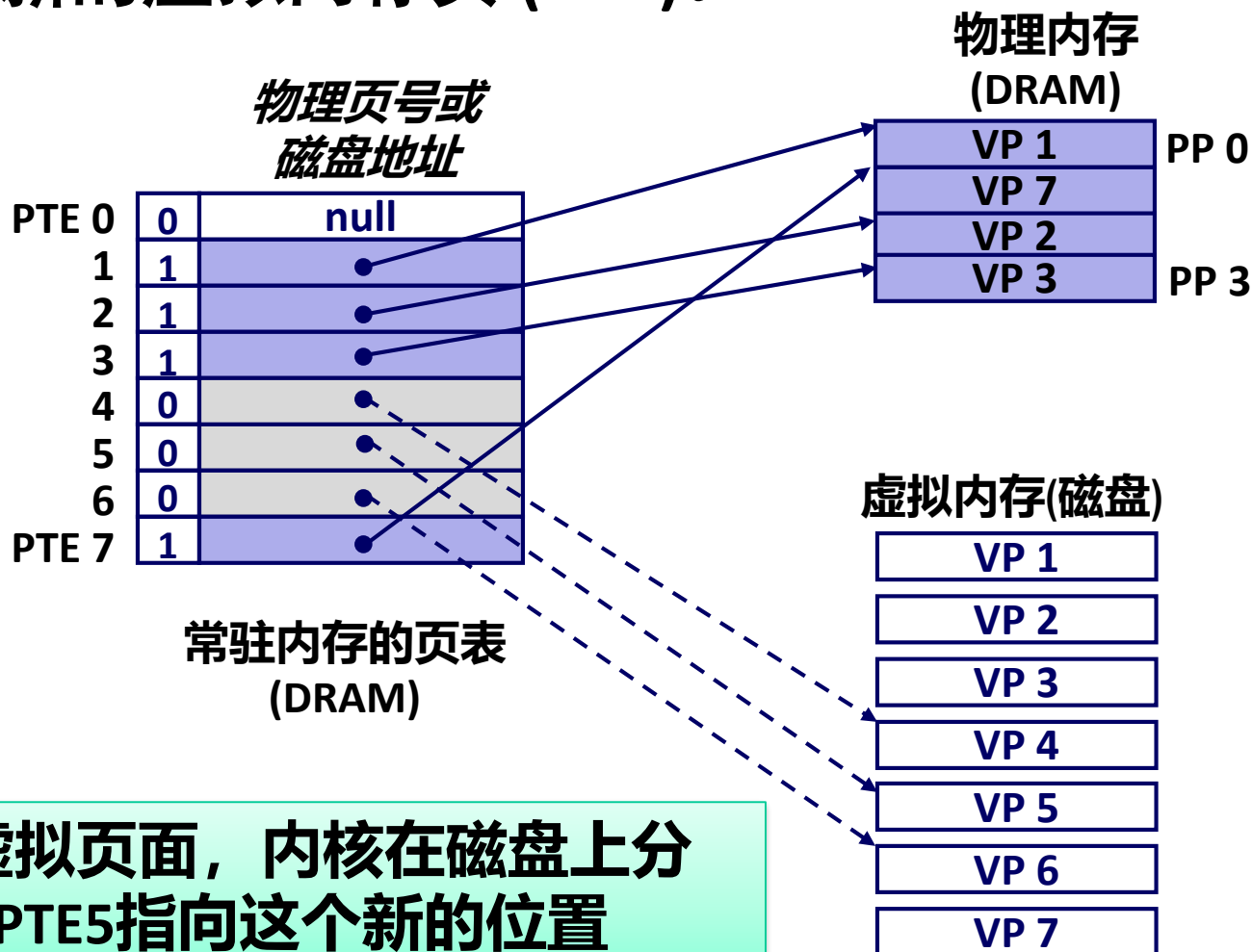
■ 分配一个新的虚拟内存页 (VP5)

- 内核在磁盘上分配VP5，并且将PTE5指向这个新的位置



分配页面

- 分配一个新的虚拟内存页 (VP5)。



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

又是局部性救了我们!

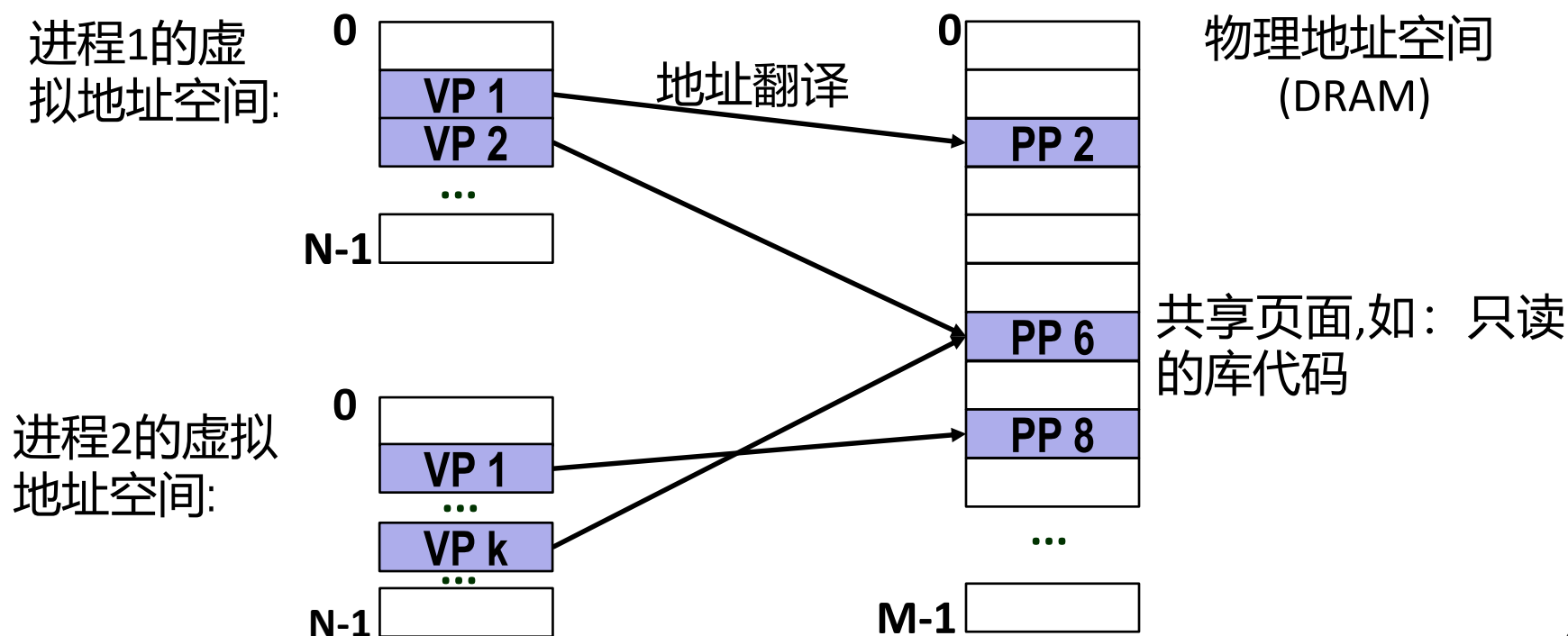
- 虚拟内存看上去效率非常低, 但它工作得相当好, 这都要归功于“局部性”。
- 在任意时间, 程序将趋于在一个较小的活动页面集合上工作, 这个集合叫做 **工作集** *Working set*
 - 程序的时间局部性越好, 工作集就会越小
- 如果 (工作集的大小 $<$ 物理内存的大小)
 - 在初始开销后, 对工作集的引用将导致命中, 不会产生额外磁盘流量。
- 如果 (工作集的大小 $>$ 物理内存的大小)
 - **Thrashing 抖动**: 页面不断地换进换出, 导致系统性能暴跌。

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

- **核心思想: 每个进程都拥有一个独立的虚拟地址空间**
 - 把内存看作简单的线性数组
 - 页表(映射函数)将虚拟地址映射到物理地址（物理内存）
 - 好映射可以提高程序的局部性



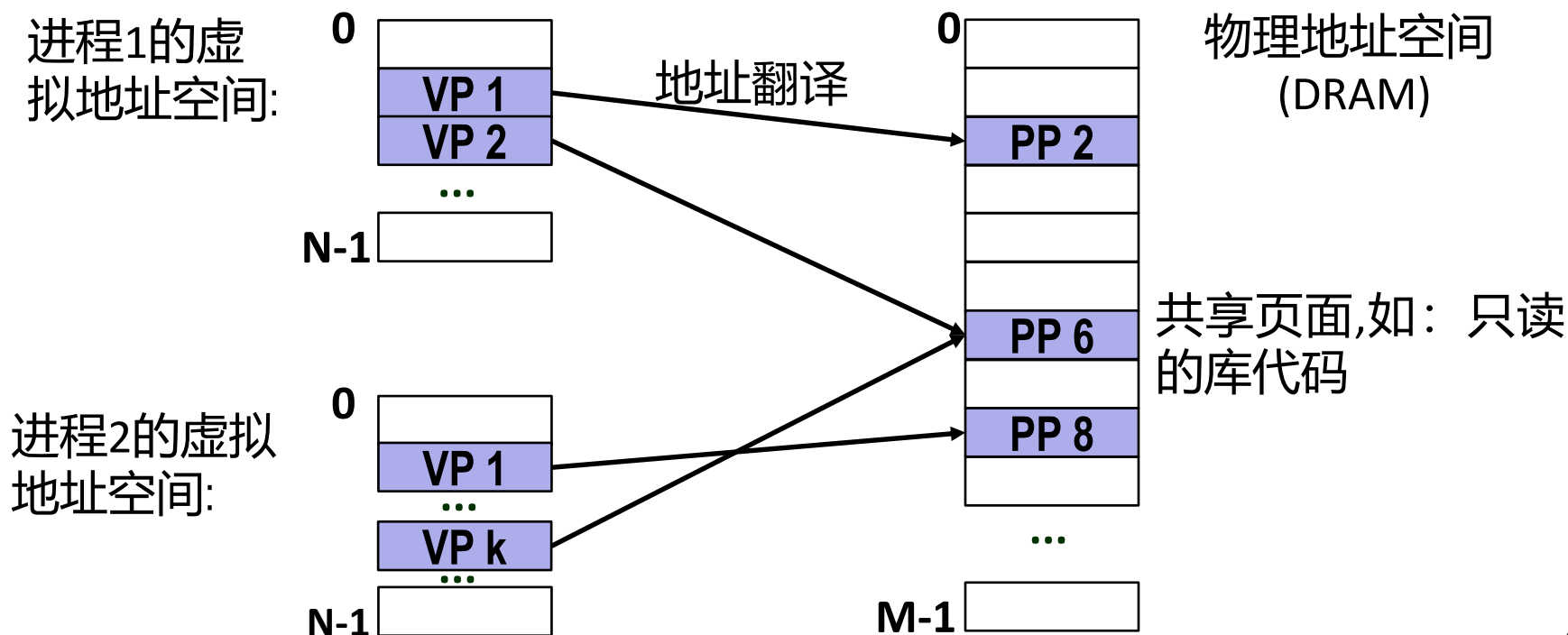
虚拟内存作为内存管理的工具

■ 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

■ 简化共享——进程间共享代码和数据

- 不同的虚拟页面映射到相同的物理页面 (如 PP 6)



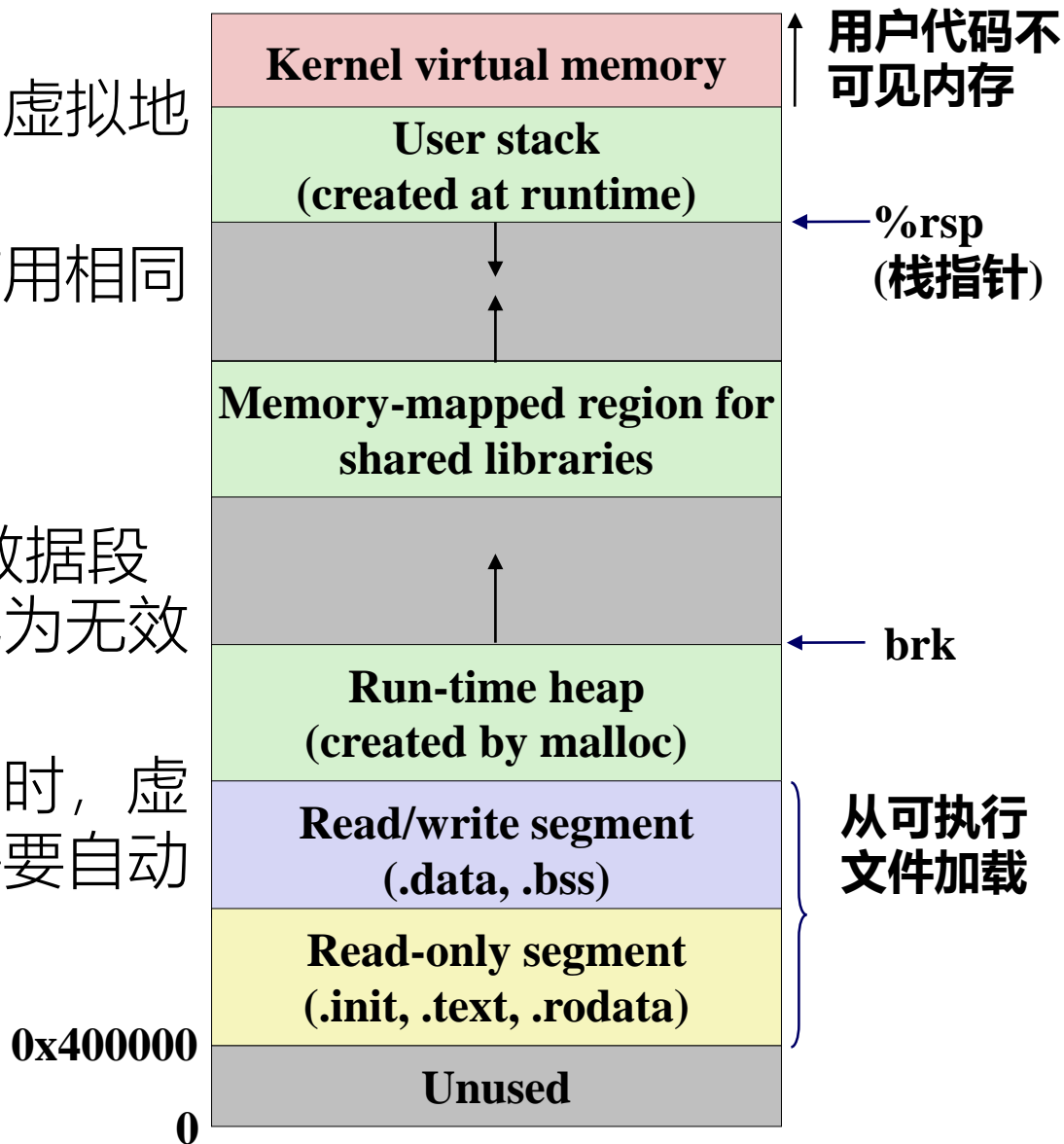
简化链接和加载

■ 简化链接

- 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

■ 简化加载

- `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

- 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县

- 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**

SUP: 进程运行在内核(超级用户)模式下才能访问

进程 i:

	SUP	READ	WRITE	EXEC	地址
VP0:	No	Yes	No	Yes	PP 6
VP1:	No	Yes	Yes	Yes	PP 4
VP2:	Yes	Yes	Yes	No	PP 2

⋮

进程 j:

	SUP	READ	WRITE	EXEC	地址
VP0:	No	Yes	No	Yes	PP 9
VP1:	Yes	Yes	Yes	Yes	PP 6
VP2:	No	Yes	Yes	Yes	PP 11

物理内存

PP 2
PP 4
PP 6
PP 8
PP 9
PP 11

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

虚拟地址翻译

■ 虚拟地址空间

- $V = \{0, 1, \dots, N-1\}$

■ 物理地址空间

- $P = \{0, 1, \dots, M-1\}$

■ 地址翻译

- $MAP: VAS \rightarrow PAS \cup \emptyset$

- 对于虚拟地址 A :

- $MAP(A) = A'$: 虚拟地址 A 处的数据在 PAS 的物理地址 A' 处

- $MAP(A) = \emptyset$: 虚拟地址 A 处的数据不在物理内存中

- ✓ 或者是无效地址、或者存储在磁盘上

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
- $P = 2^p$: Page size (bytes)

■ 虚拟地址VA的组成部分

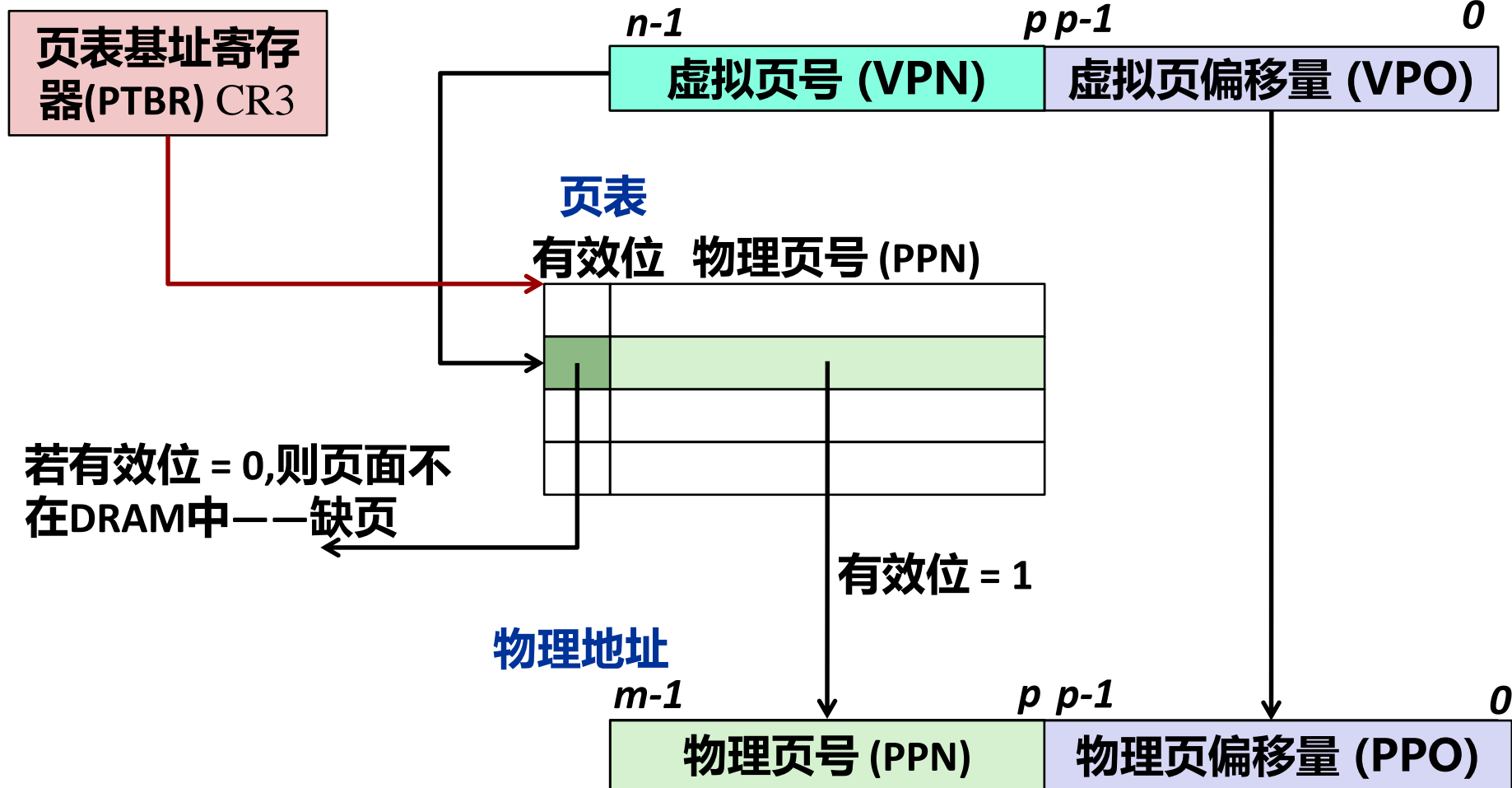
- TLBI: TLB index----TLB索引
- TLBT: TLB tag----TLB标记
- VPO: 虚拟页面偏移量（字节） ----Virtual page offset
- VPN: 虚拟页号----Virtual page number

■ 物理地址PA的组成部分

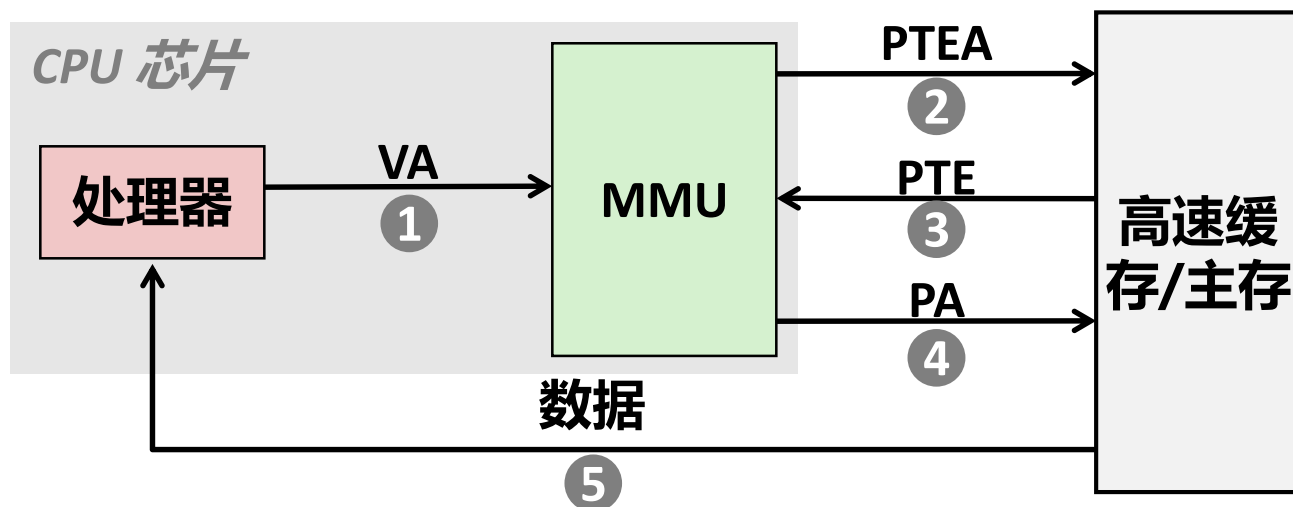
- PPO: 物理页面偏移量(Physical page offset ,same as VPO)
- PPN: 物理页号(Physical page number)

基于页表的地址翻译

当前进程页表的物理地址



地址翻译：页面命中

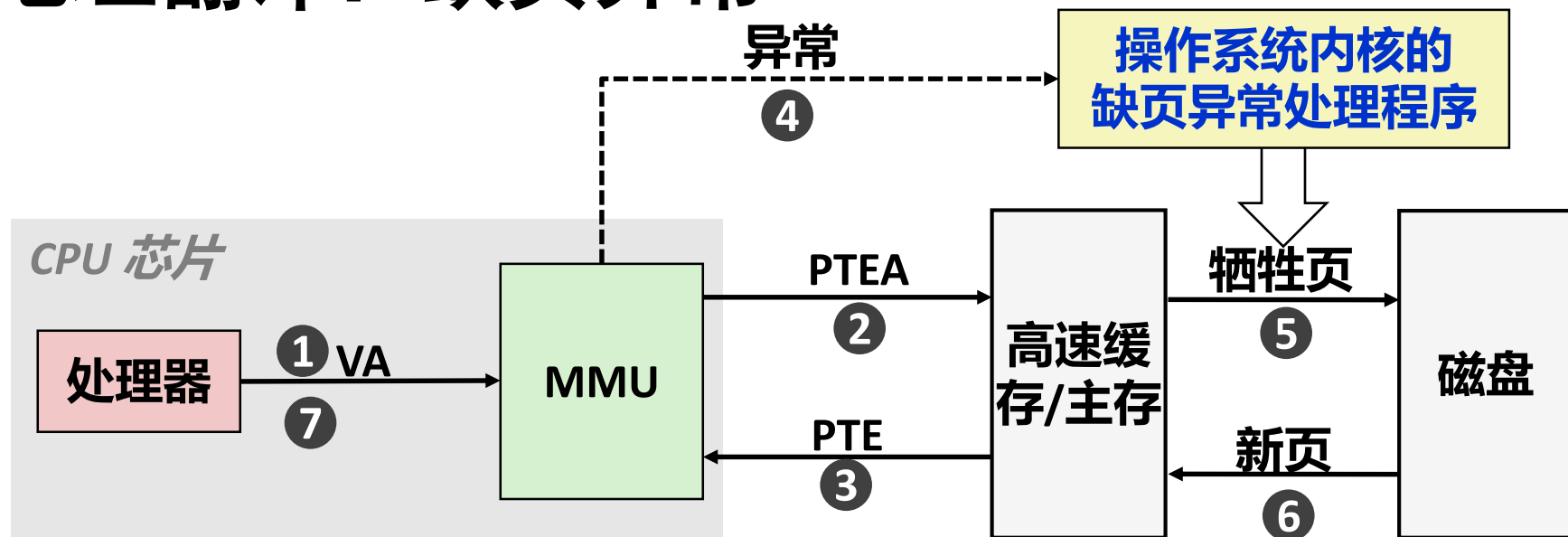


- 1) 处理器生成一个虚拟地址，并将其传送给MMU
- 2) MMU生成PTE地址(PTEA)，并从高速缓存/主存请求得到PTE
- 3) 高速缓存/主存向MMU返回PTE
- 4) MMU 将物理地址传送给高速缓存/主存
- 5) 高速缓存/主存返回所请求的数据字给处理器

整个过程完全由硬件处理

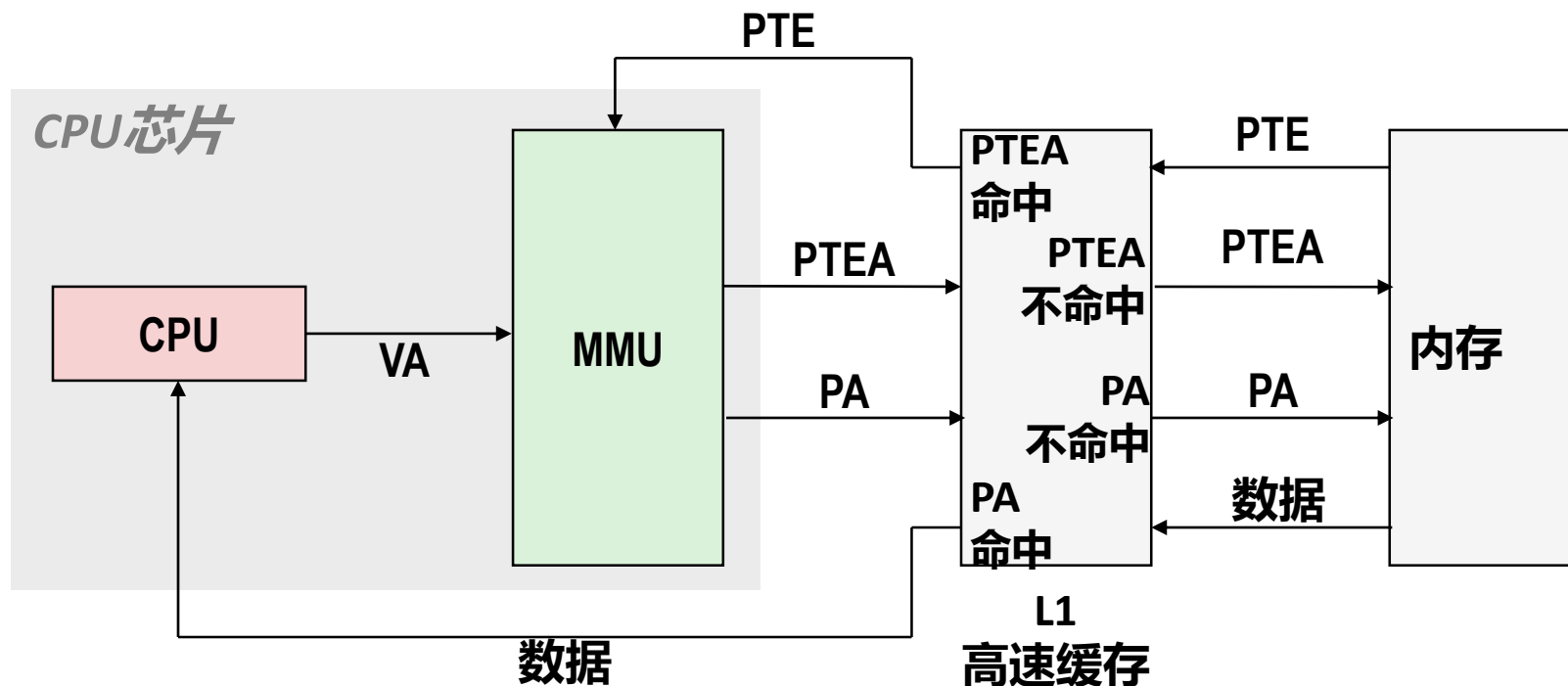
地址翻译：缺页异常

由硬件、OS内核协作完成



- 1) 处理器生成一个虚拟地址，并将其传送给MMU
- 2) MMU生成PTE地址(PTEA)，并从高速缓存/主存请求得到PTE
- 3) 高速缓存/主存向MMU返回PTE
- 4) PTE的有效位为零，因此MMU触发缺页异常
- 5) 缺页处理程序确定物理内存中的牺牲页（若页面被修改，则换出到磁盘——写回策略）
- 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，再次执行导致缺页的指令

结合高速缓存和虚拟内存



VA : virtual address 虚拟地址

PTE: page table entry 页表条目

PA: physical address 物理地址

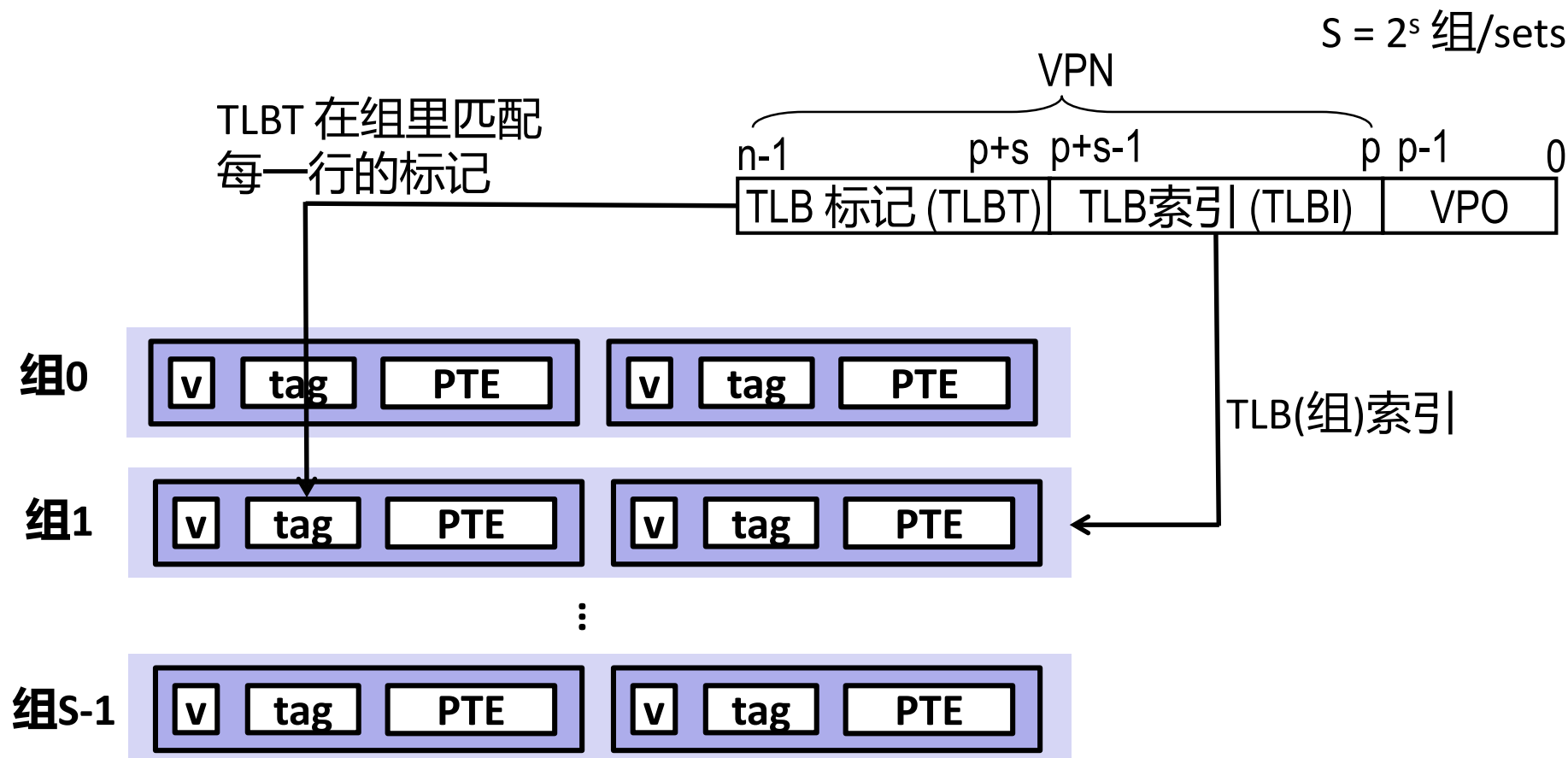
PTEA: PTE address 页表条目地址

利用TLB加速地址翻译

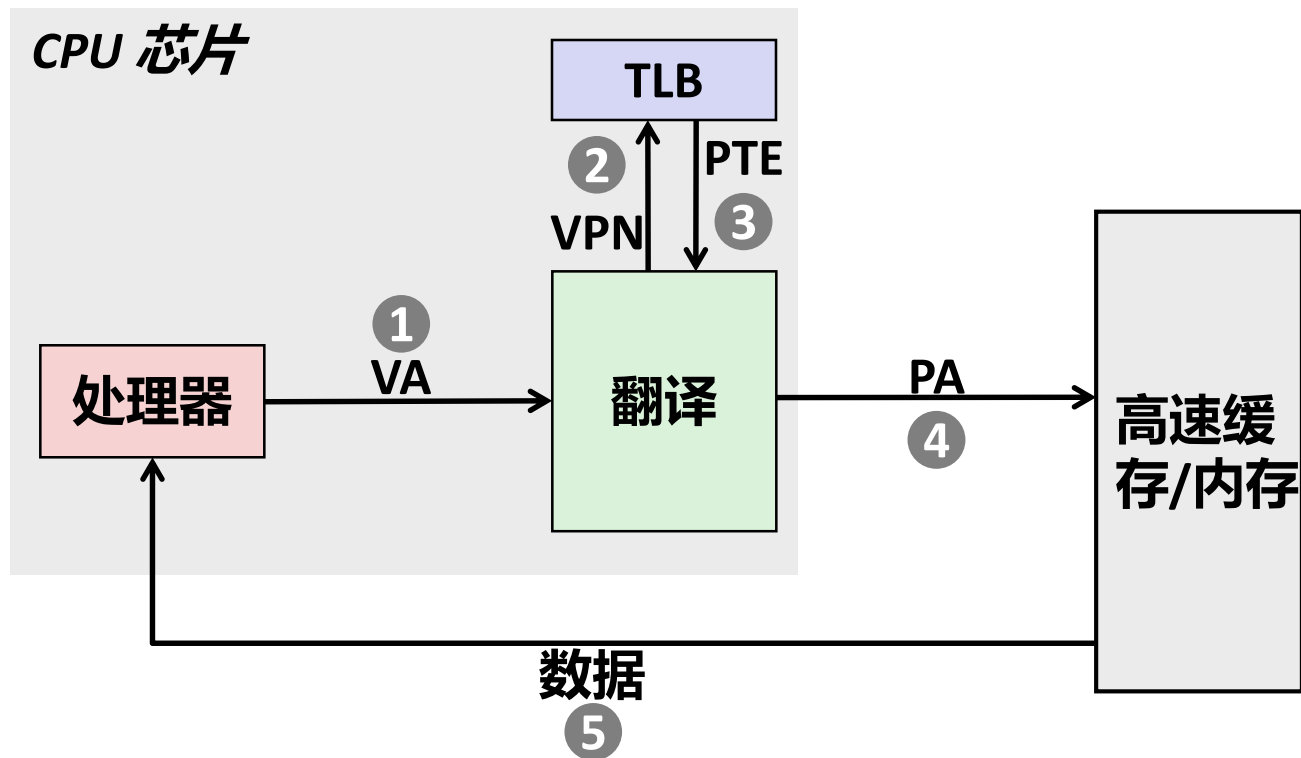
- 页表条目 (PTEs) 与其他内存数据字一样缓存在 L1 中
 - PTE 可能被其他数据引用所替换/驱逐，导致不命中
 - PTE 即便 cache 命中，也仍然需要与 L1 相当的延迟 (1-2 周期)
- 解决办法：翻译后备缓冲器 (*Translation Lookaside Buffer*, TLB), *更快*
 - MMU 中一个小的具有高相联度的集合
 - 实现虚拟页号向物理页号的映射
 - 页数很少的页表可以完全放在 TLB 中

访问TLB

■ MMU 使用虚拟地址的 VPN 部分来访问TLB:



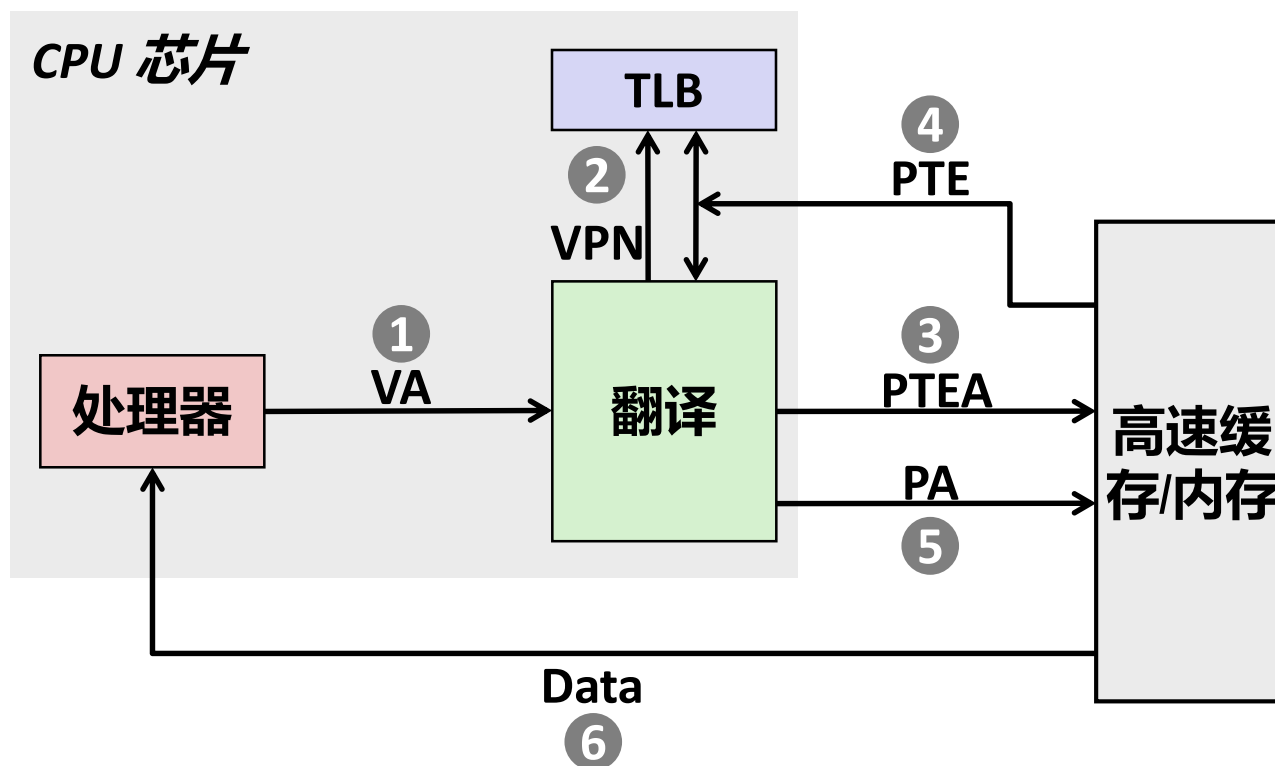
TLB命中



TLB 命中减少内存访问

地址翻译都在芯片上的MMU中完成, **极快!** **极快!**

TLB 不命中



TLB 不命中会引发额外的内存访问

万幸：TLB 不命中很少发生，为什么？

多级页表

■ 假设:

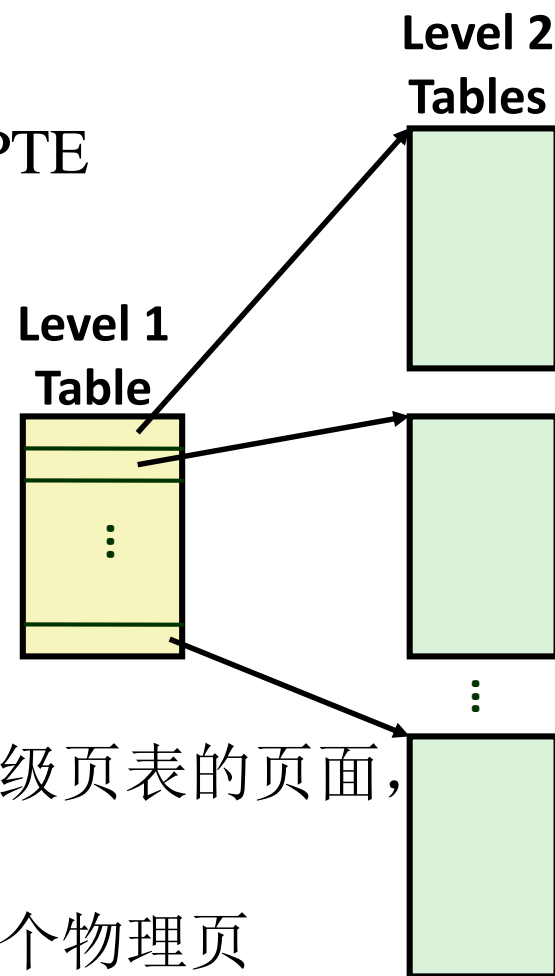
- 4KB (2^{12}) 页面, 48位地址空间, 8字节 PTE

■ 问题:

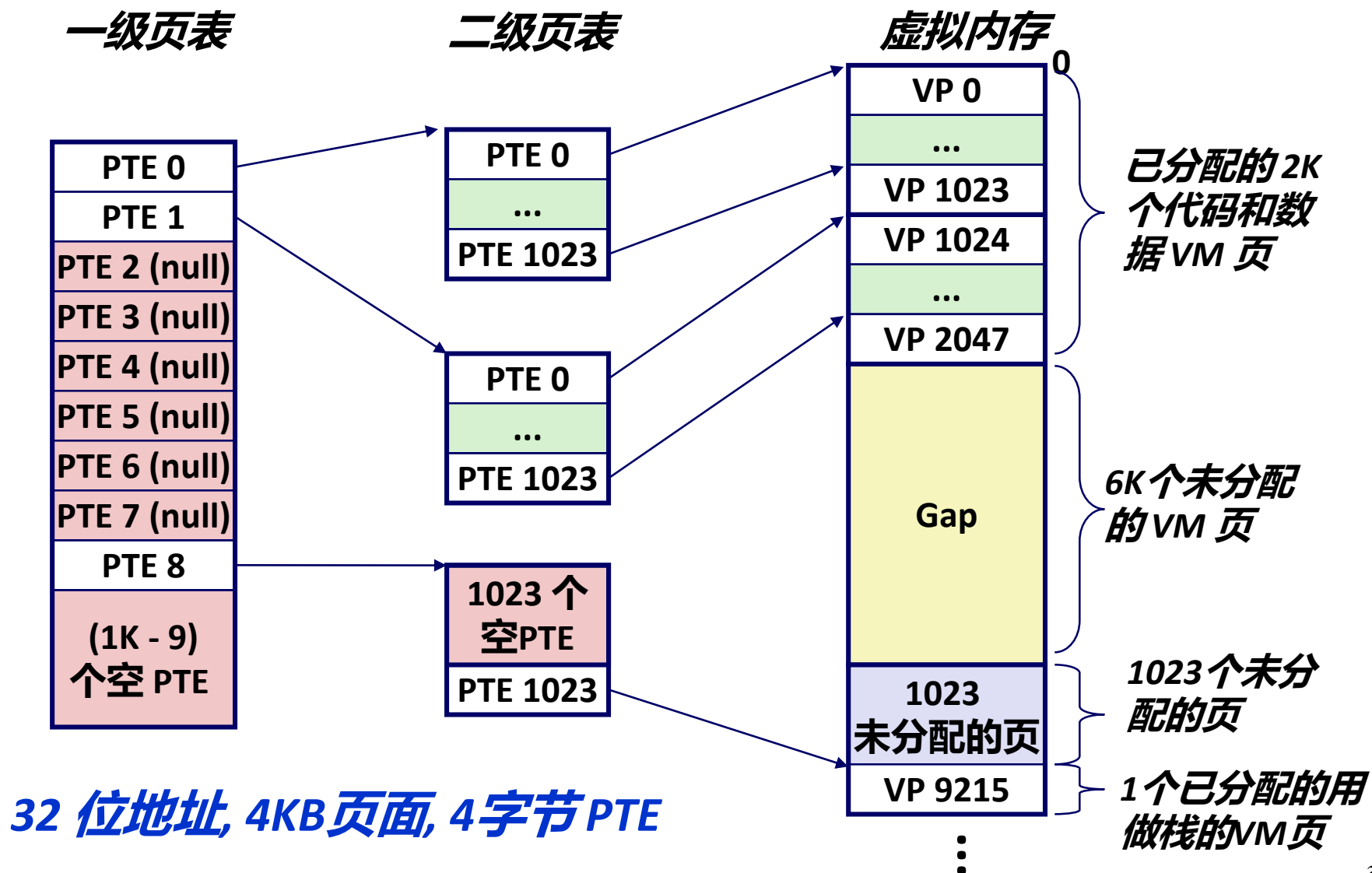
- 将需要一个大小为 512 GB 的页表!
 - $2^{48} \times 2^{-12} \times 2^3 = 2^{30+9}$ bytes

■ 常用解决办法: 多级页表

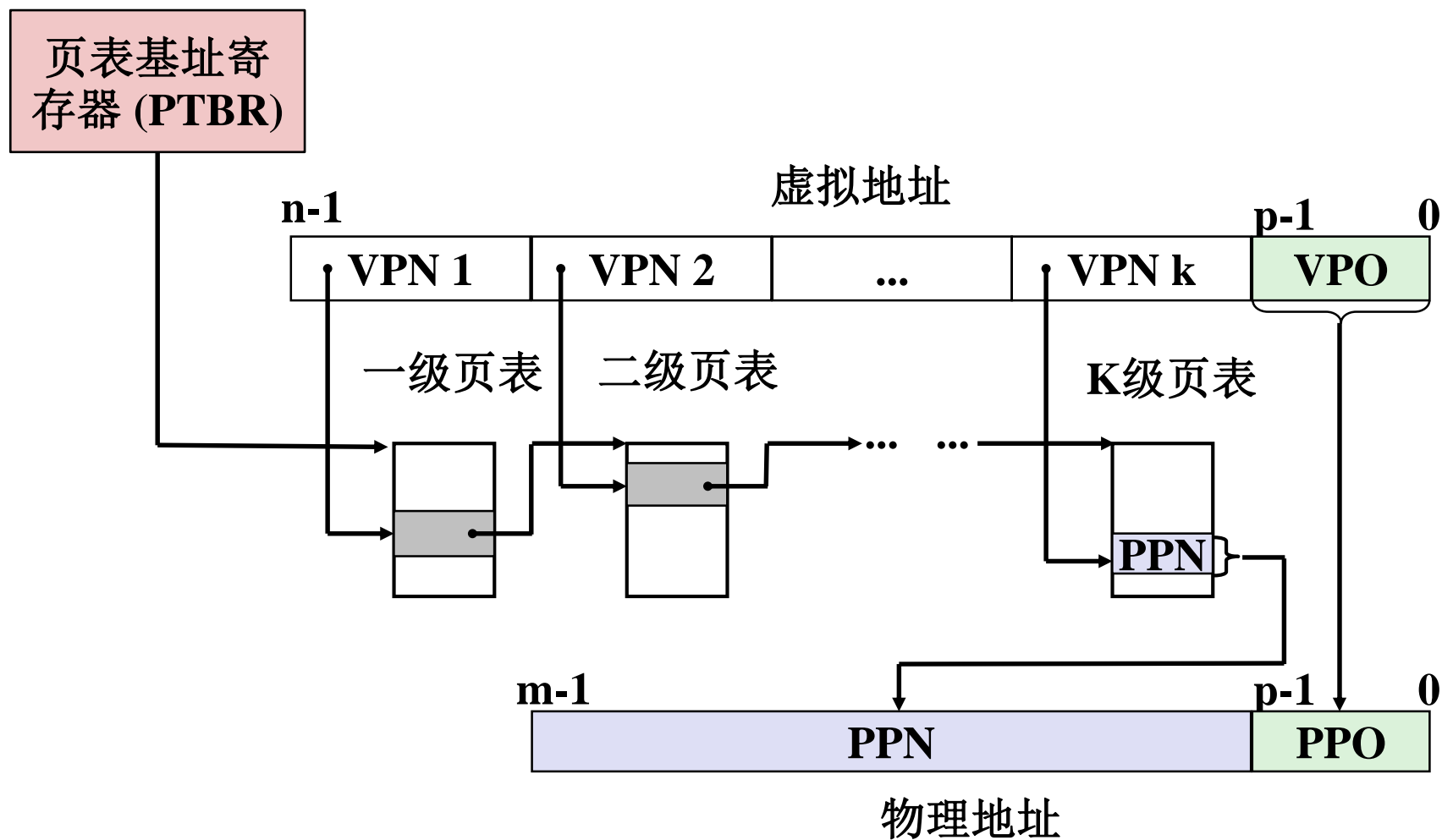
- 以二级页表为例:
 - 一级页表: 1页, 每个PTE指向1个2级页表的页面, 常驻内存
 - 二级页表: 多页, 每个 PTE 指向1个物理页
像普通数据一样, 可以调入或调出页表



二级页表的层次结构



使用k级页表的地址翻译



二级页表的优点

■ 节省内存

- 若一级页表项为空，对应的二级页表就不存在(不用保存)
- 只有一级页表需要常驻在主存中
- 二级页表：根据需要，创建、页面调入或调出。
 - 最经常使用的二级页表，才会在主存中。

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
- 不会被其他进程破坏

■ 系统的角度看待虚拟内存

- 通过缓存虚拟内存页面来有效使用内存(内存作为虚拟内存的cache)
 - 有效，仅仅是因为程序的“局部性”
- 简化了内存管理和编程
- 通过提供方便的插入点(标志位)来检查权限许可，简化内存保护