

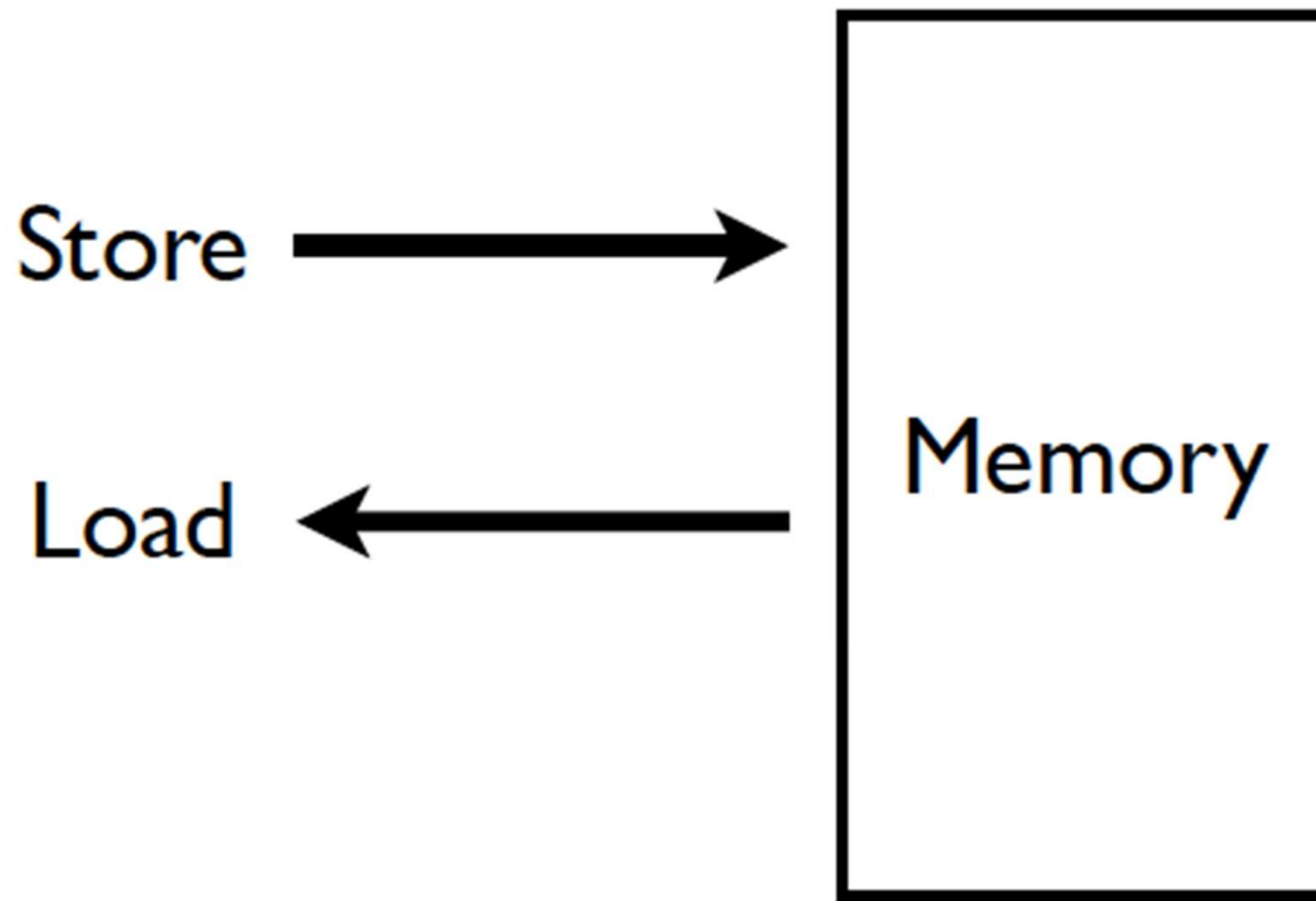
虚拟内存：概念

Sep. 08, 2025

主要内容

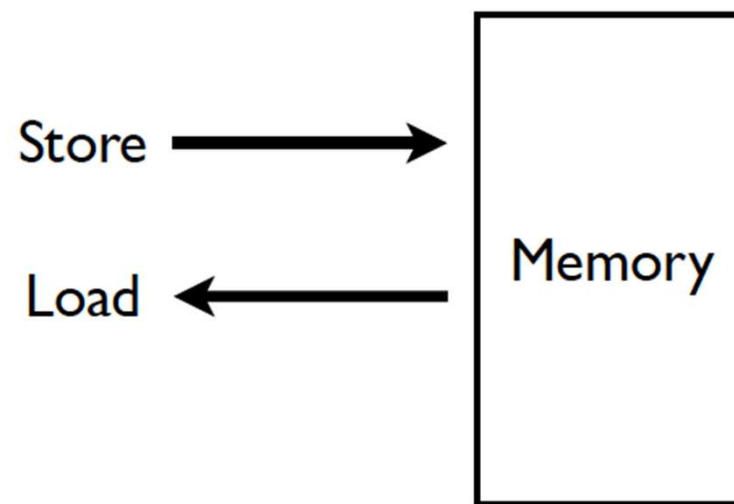
- 地址空间
- VM作为缓存工具
- VM作为内存管理工具
- VM作为内存保护工具
- 地址转换

内存（程序员的视角）



理想的内存

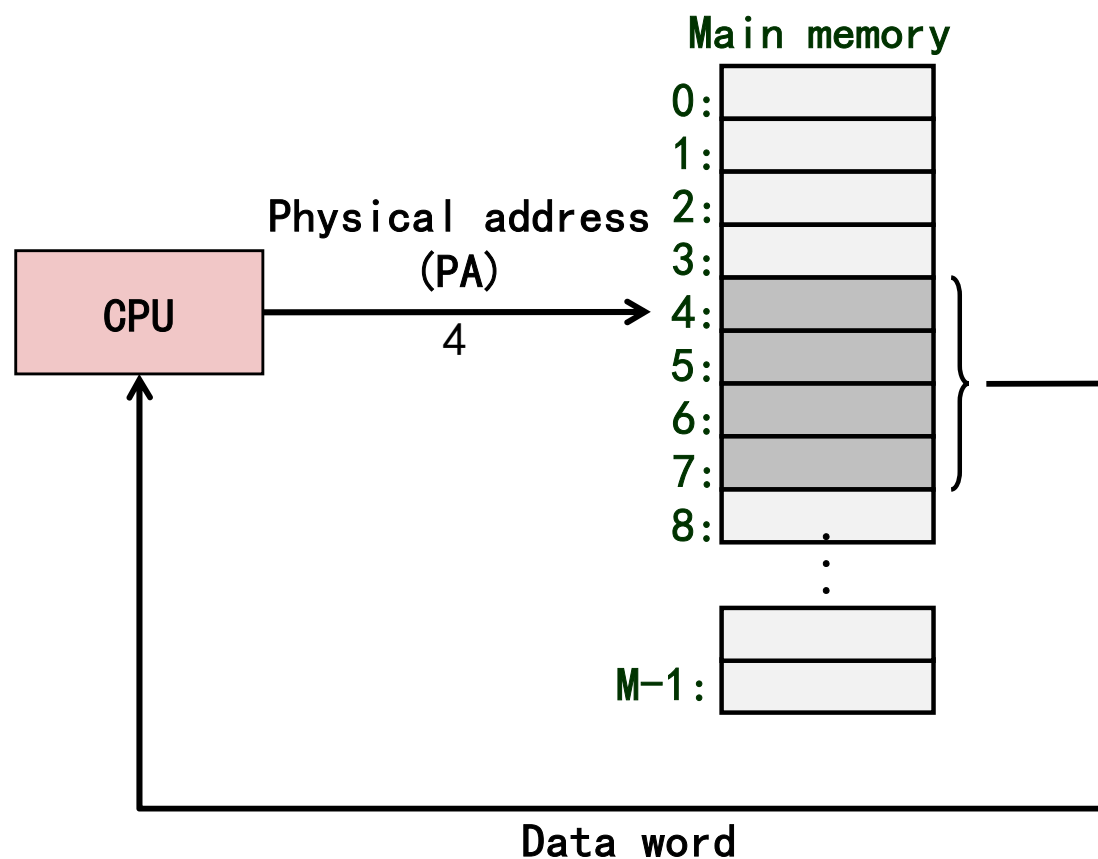
- 零访问时间（延迟）
- 无限容量
- 零成本
- 无限带宽（支持并行多路访问）



抽象：虚拟内存与物理内存

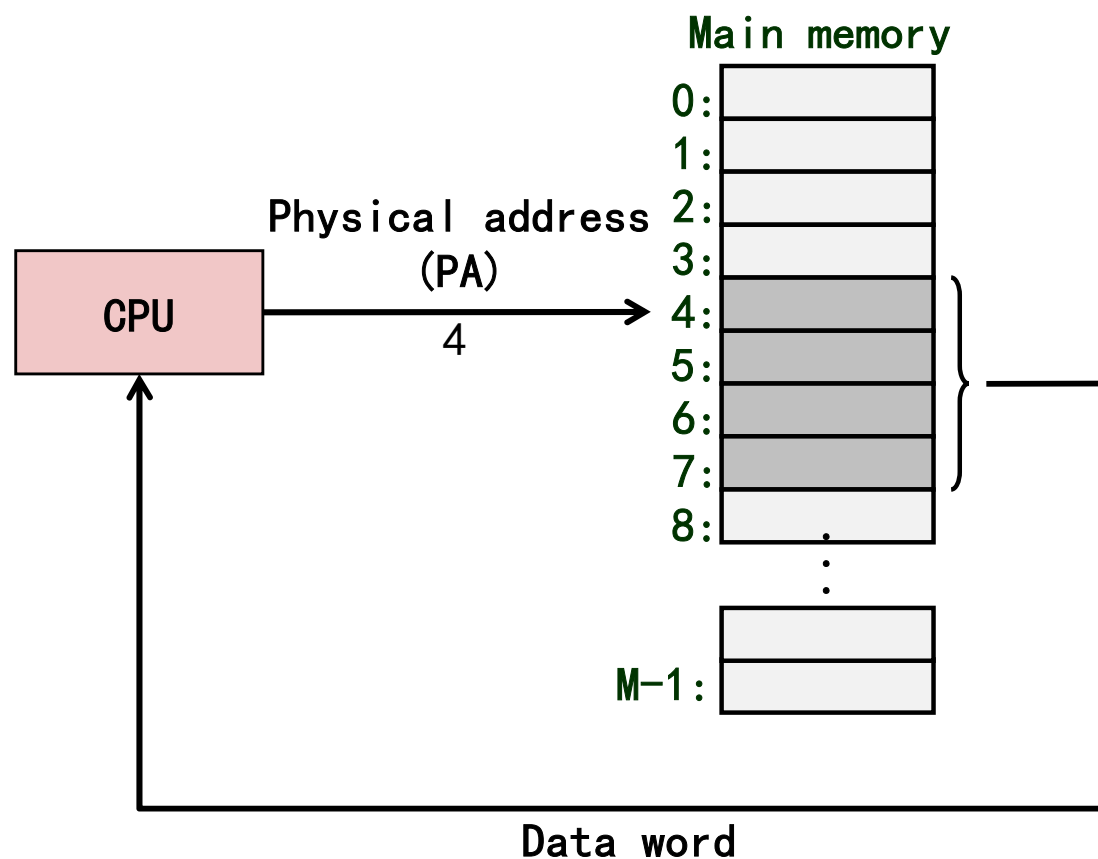
- 程序员看到虚拟内存
 - 可以假设内存是“无限的”
- 实际情况：物理内存大小远小于程序员的假设
- 系统（系统软件 + 硬件协同工作）将虚拟内存地址映射到物理内存
 - 系统自动管理物理内存空间，对程序员透明
- + 程序员无需了解内存的物理大小，也无需管理它
 - 较小的物理内存在程序员看来可能很大
 - 程序员的工作更轻松
- —— 更复杂的系统软件和架构

使用物理寻址的系统



- 用于“简单”系统，例如汽车、电梯和数码相框等设备中的嵌入式微控制器

使用物理寻址的系统



- 用于“简单”系统，例如汽车、电梯和数码相框等设备中的嵌入式微控制器

使用物理寻址的系统的问题

■ 物理内存大小有限（成本高）

- 如果需要更多内存怎么办？
- 程序员是否应该关注代码/数据块的大小是否适合物理内存？
- 程序员是否应该管理从磁盘到物理内存的数据移动？
- 程序员是否应该确保两个进程不使用相同的物理内存？

■ 此外，ISA 的地址空间可以大于物理内存大小

- 例如，具有字节寻址能力的 64 位地址空间
- 如果物理内存不足怎么办？

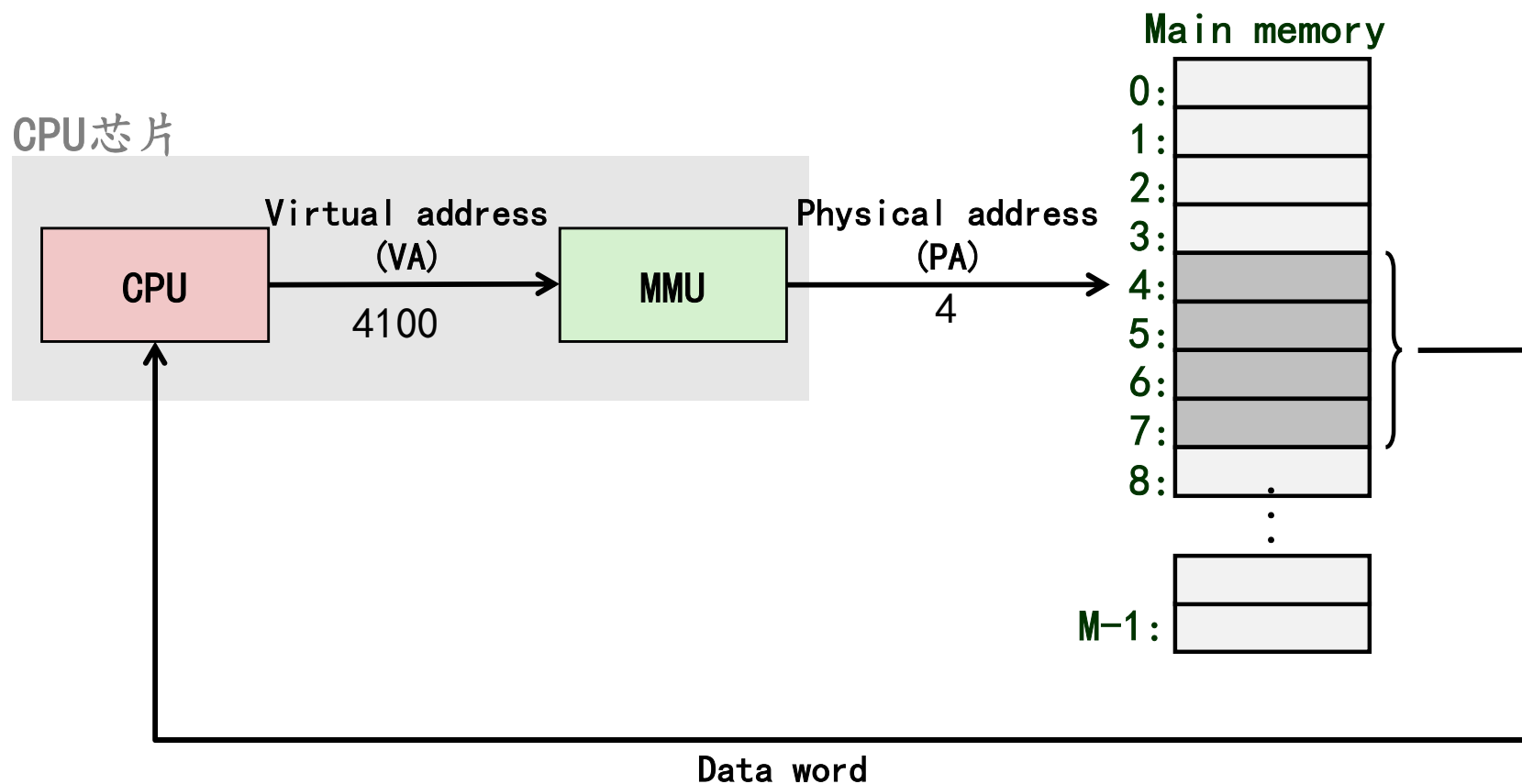
直接物理寻址的困难

- 程序员需要管理物理内存空间
 - 不方便且困难
 - 当有多个进程时，难度会更大
- 难以支持代码和数据重定位
- 难以支持多个进程
 - 多个进程之间的保护和隔离
 - 物理内存空间共享
- 难以支持跨进程的数据/代码共享

虚拟内存

- 理念：让程序员在拥有较小物理内存的情况下，产生地址空间较大的错觉。
 - 这样程序员就无需担心物理内存的管理。
- 程序员可以假设自己拥有“无限”的物理内存。
- 硬件和软件协同自动管理物理内存空间，从而提供这种错觉。
 - 每个独立进程都维持这种错觉。

使用虚拟寻址的系统



- 适用于所有现代服务器、笔记本电脑和智能手机
- 这是计算机科学领域的伟大理念之一

地址空间

- **线性地址空间 (Linear address space):** 连续非负整数地址的有序集合:

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

- **虚拟地址空间 (Virtual address space):** $N = 2^n$ 个虚拟地址集合:

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

- **物理地址空间 (Physical address space):** $M = 2^m$ 个物理地址集合:

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

为什么要使用虚拟内存（VM）？

■ 高效利用主内存

- 使用 DRAM 作为部分虚拟地址空间的缓存

■ 简化内存管理

- 每个进程获得相同的统一线性地址空间

■ 隔离地址空间

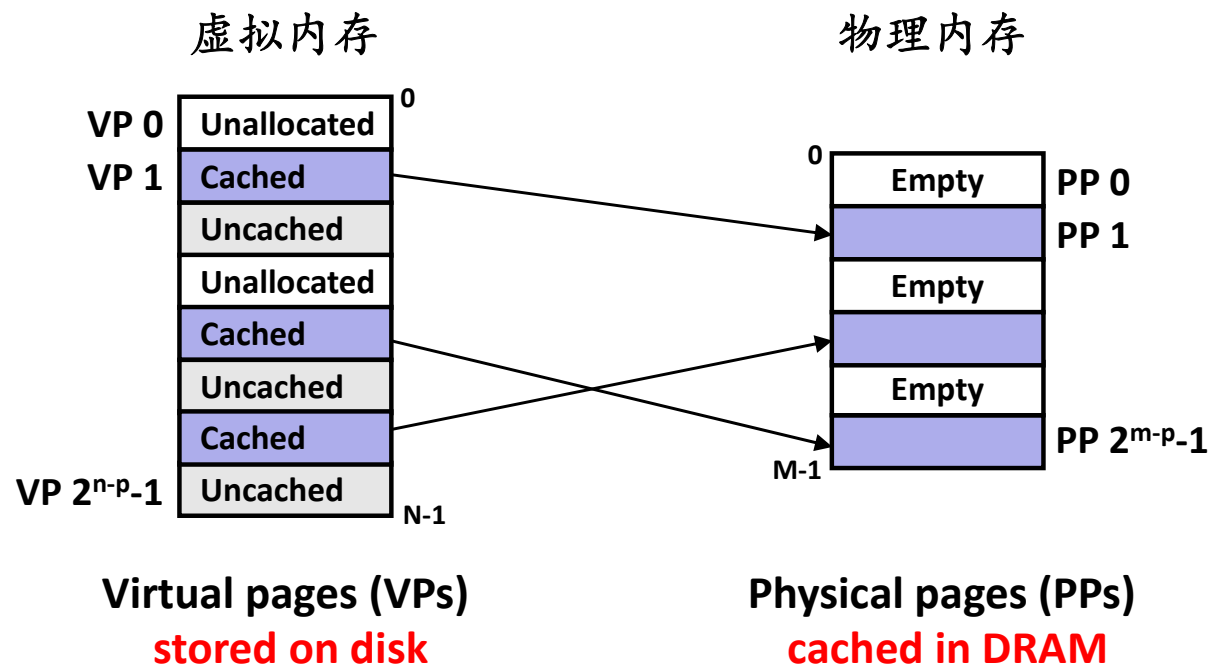
- 进程间的内存互不干扰
- 用户程序无法访问特权内核信息和代码

主要内容

- 地址空间
- VM作为缓存工具
- VM作为内存管理工具
- VM作为内存保护工具
- 地址转换

虚拟内存作为缓存工具

- 从概念上讲，虚拟内存是存储在磁盘上的 N 个连续字节的数组。
- 这些字节块的内容被**缓存到物理内存（DRAM缓存）**中。
 - 这些缓存块被称为页面（Pages），大小是 ($P = 2^p$ bytes)
 - 虚拟内存的每一页（VP）对应物理内存中的一个物理页（PP）



DRAM缓存是如何组织的

■ DRAM 缓存组织受巨大缺失惩罚驱动

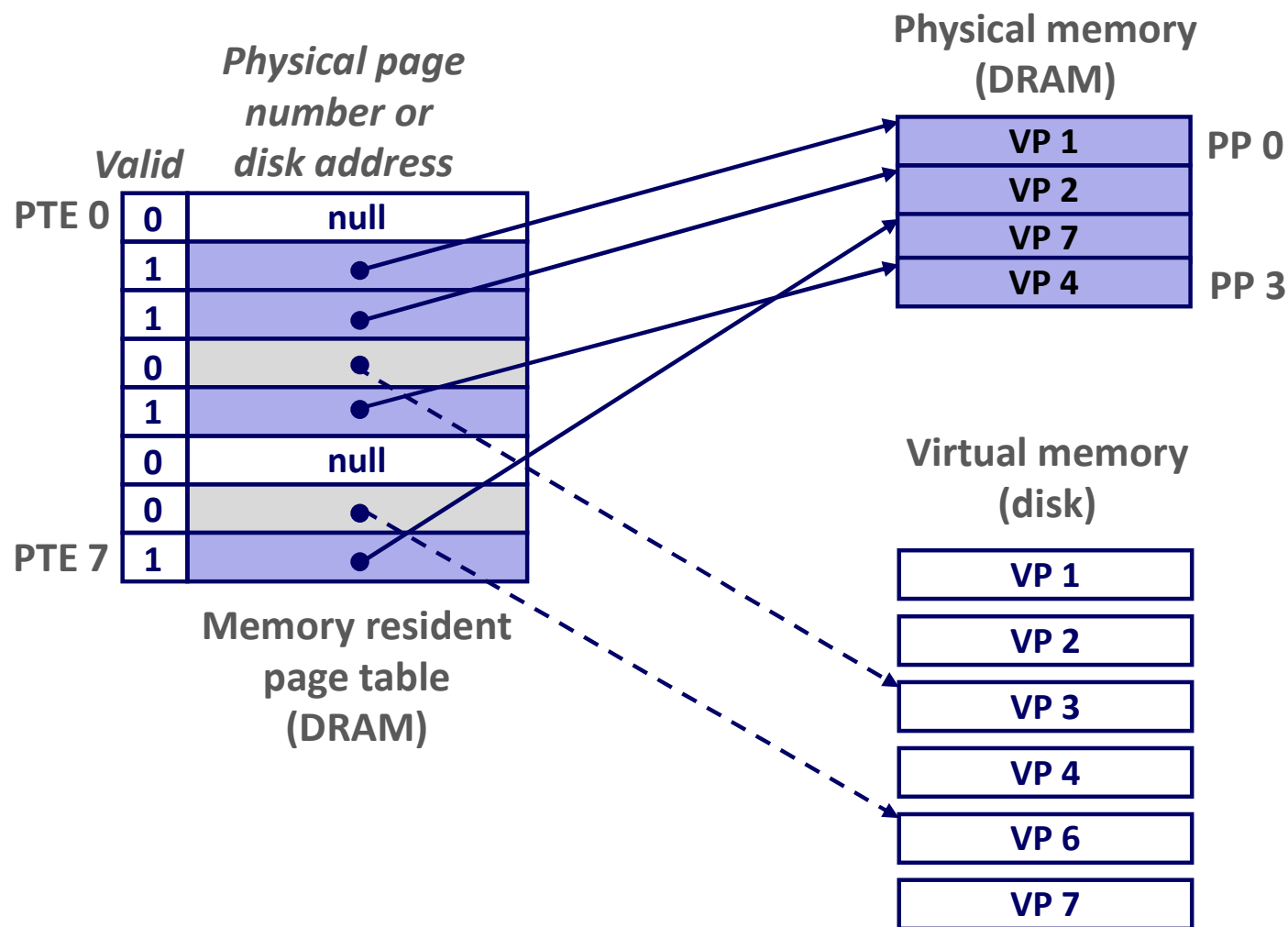
- DRAM 比 SRAM 慢 10 倍左右
- 磁盘比 DRAM 慢 10,000 倍左右

■ 后果

- 大页面（块）大小：通常为 4 KB，有时为 4 MB
- 完全关联（Fully associative）
 - 任何 VP（虚拟页面）都可以放置在任何 PP（物理页面）
 - 需要一个“大”映射函数——与缓存内存不同
- 高度复杂且昂贵的替换算法
 - 太复杂且开放式，无法在硬件中实现
- 写回而不是写穿透（直写）

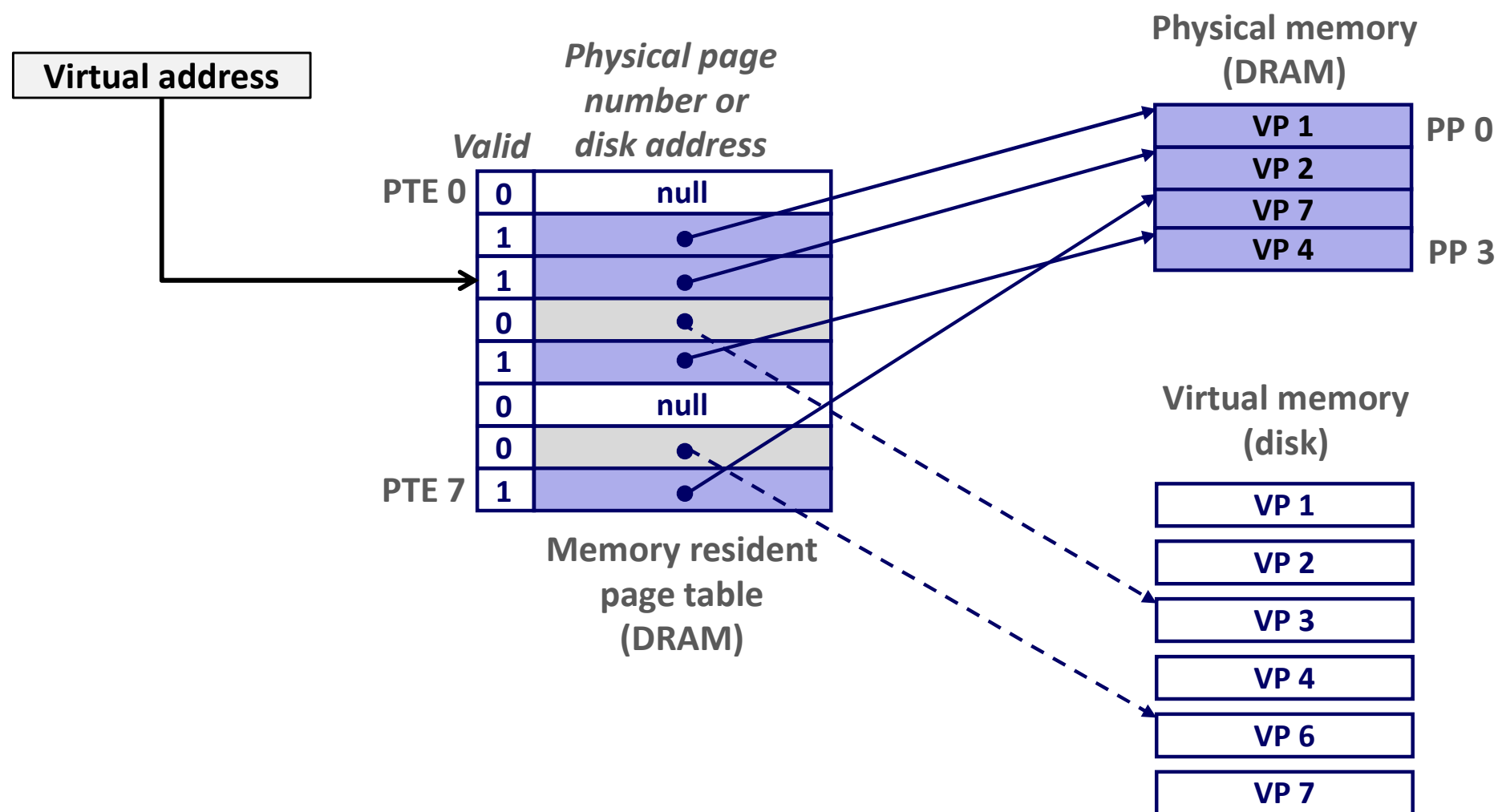
启用数据结构：页表

- 页表是一个页表项（PTE）的数组，用于将虚拟页映射到物理页。
- 每个进程的内核数据结构保存在 DRAM 中。



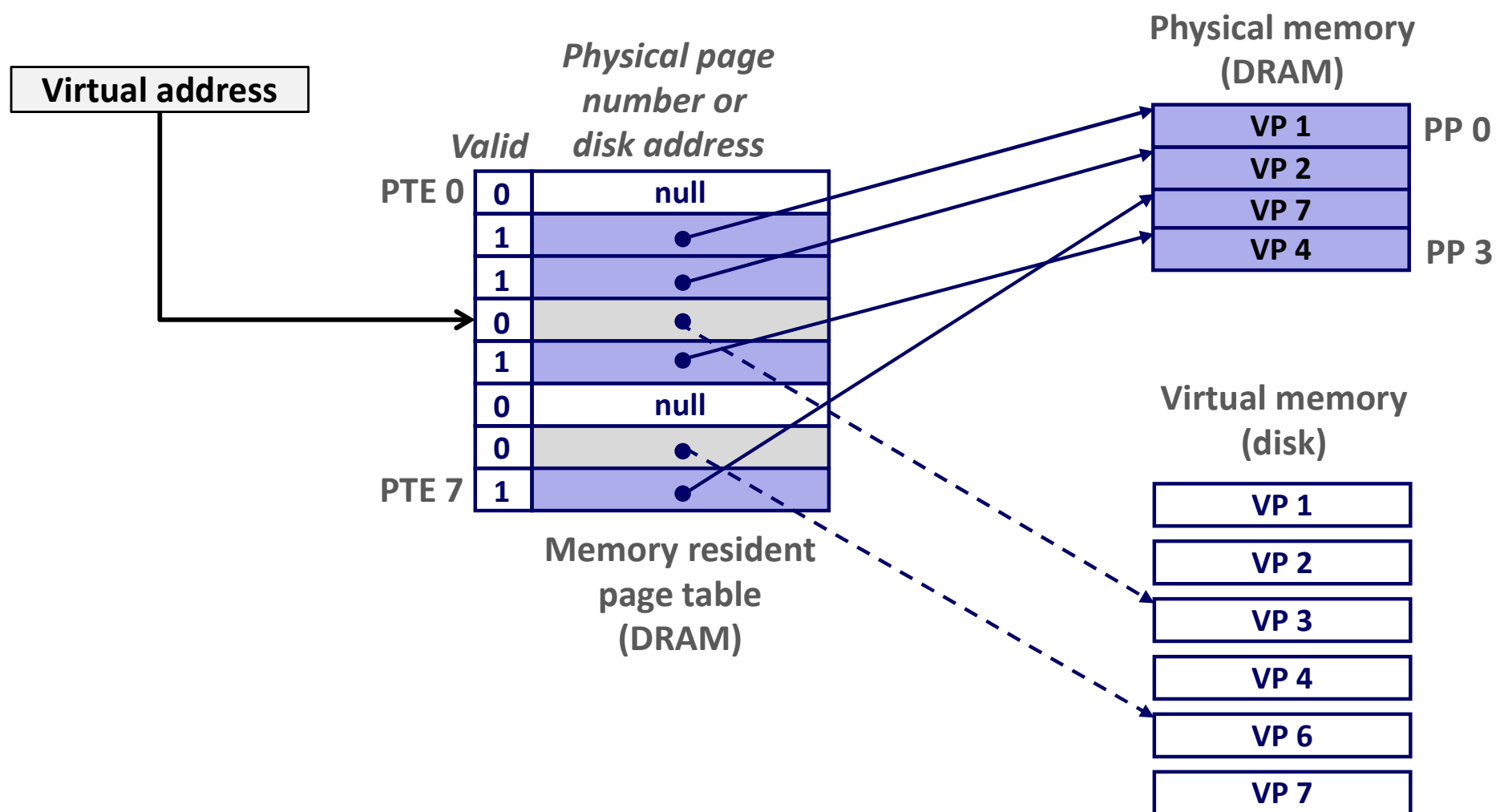
页命中 (Page Hit)

- 页命中：对在物理内存中的虚拟内存字的引用（DRAM 缓存命中）



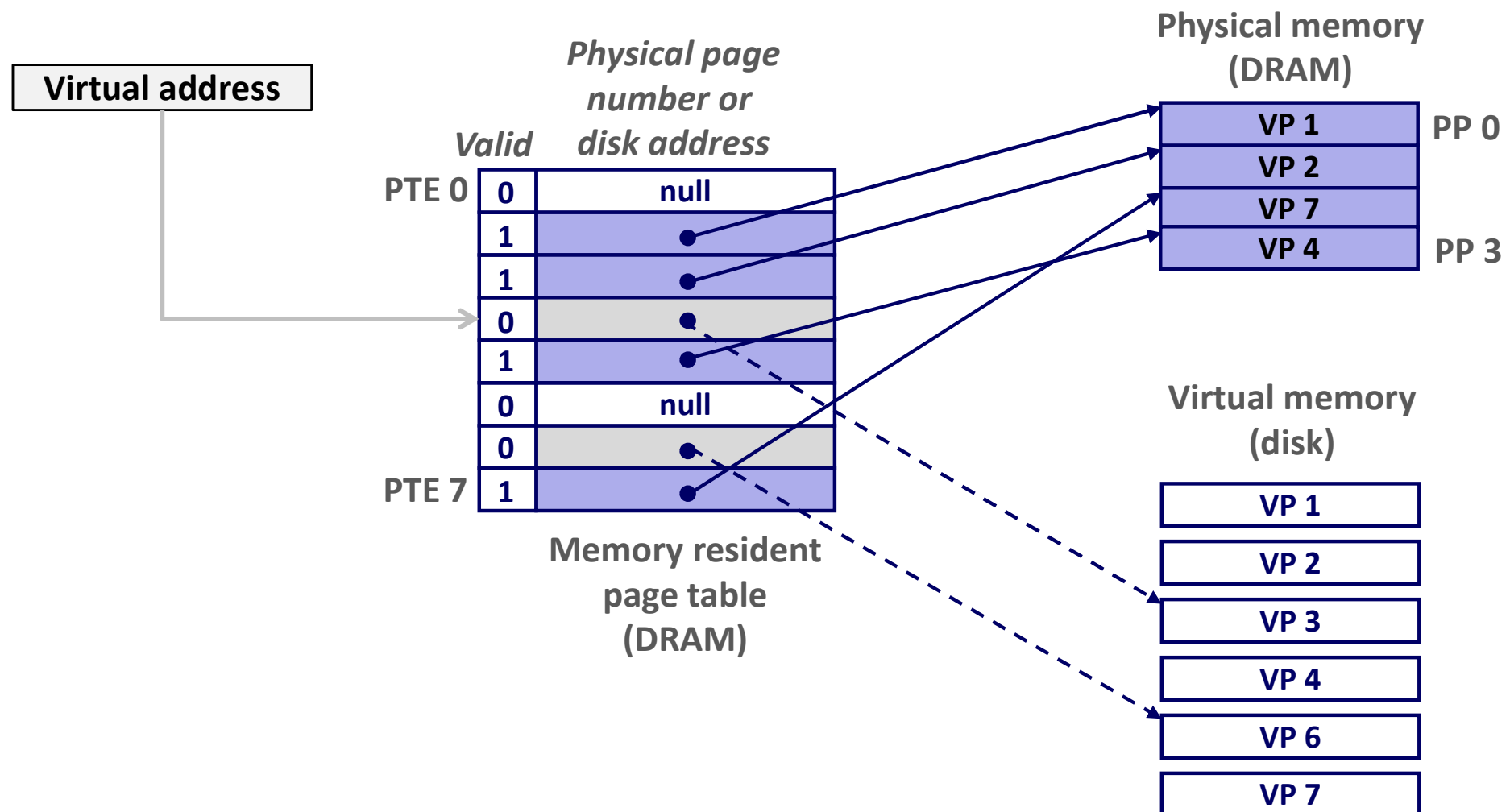
页面错误/缺页异常 (Page Fault)

- **页面错误**: 引用的虚拟内存 (VM) 中的字不在物理内存中 (即 DRAM 缓存未命中)



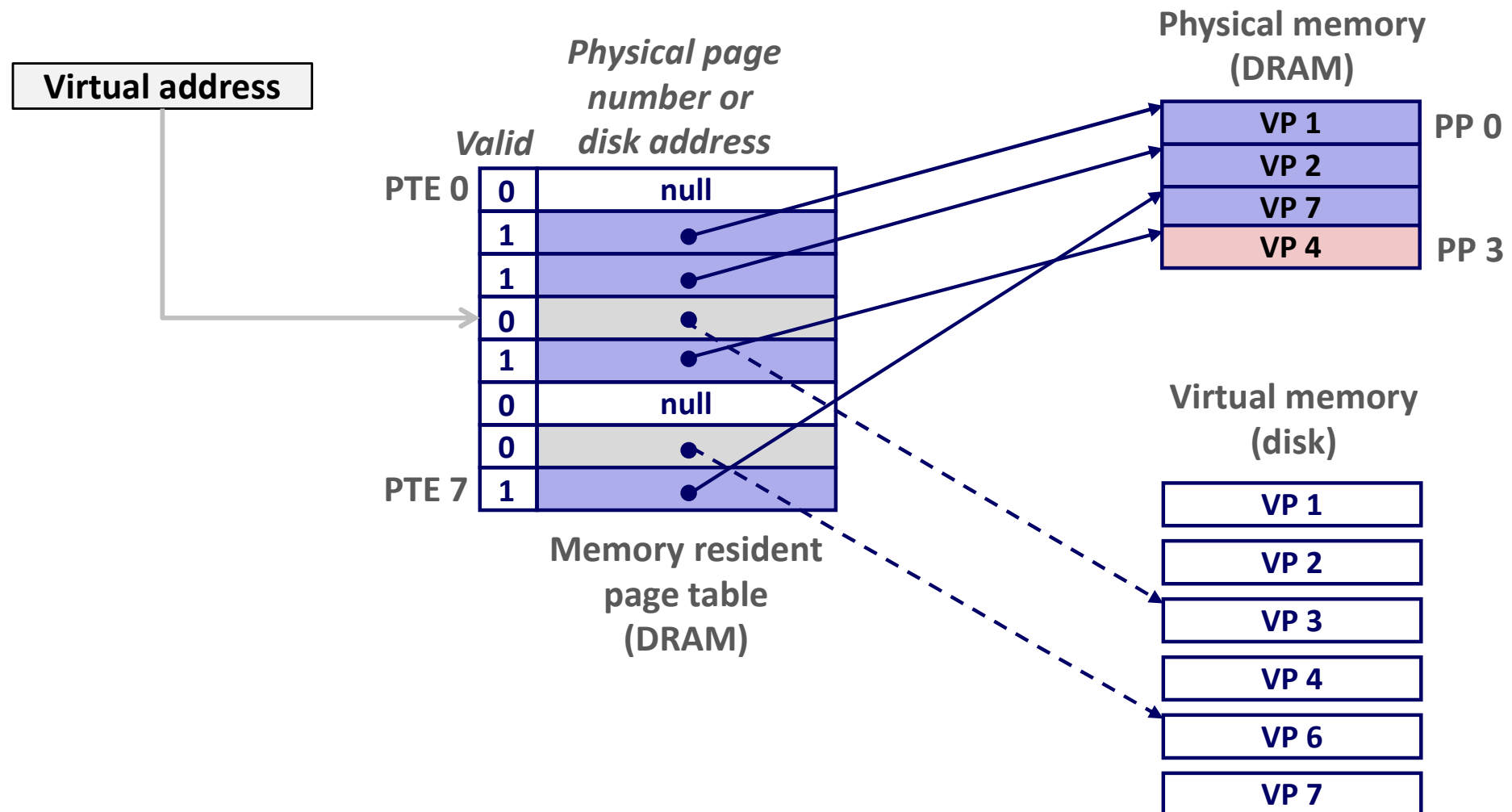
处理缺页异常 (Handling Page Fault)

- 页面未命中会导致 缺页异常 (page fault, 属于一种异常)
- 缺页异常的处理由操作系统中的缺页处理程序 (Page Fault Handler) 完成。



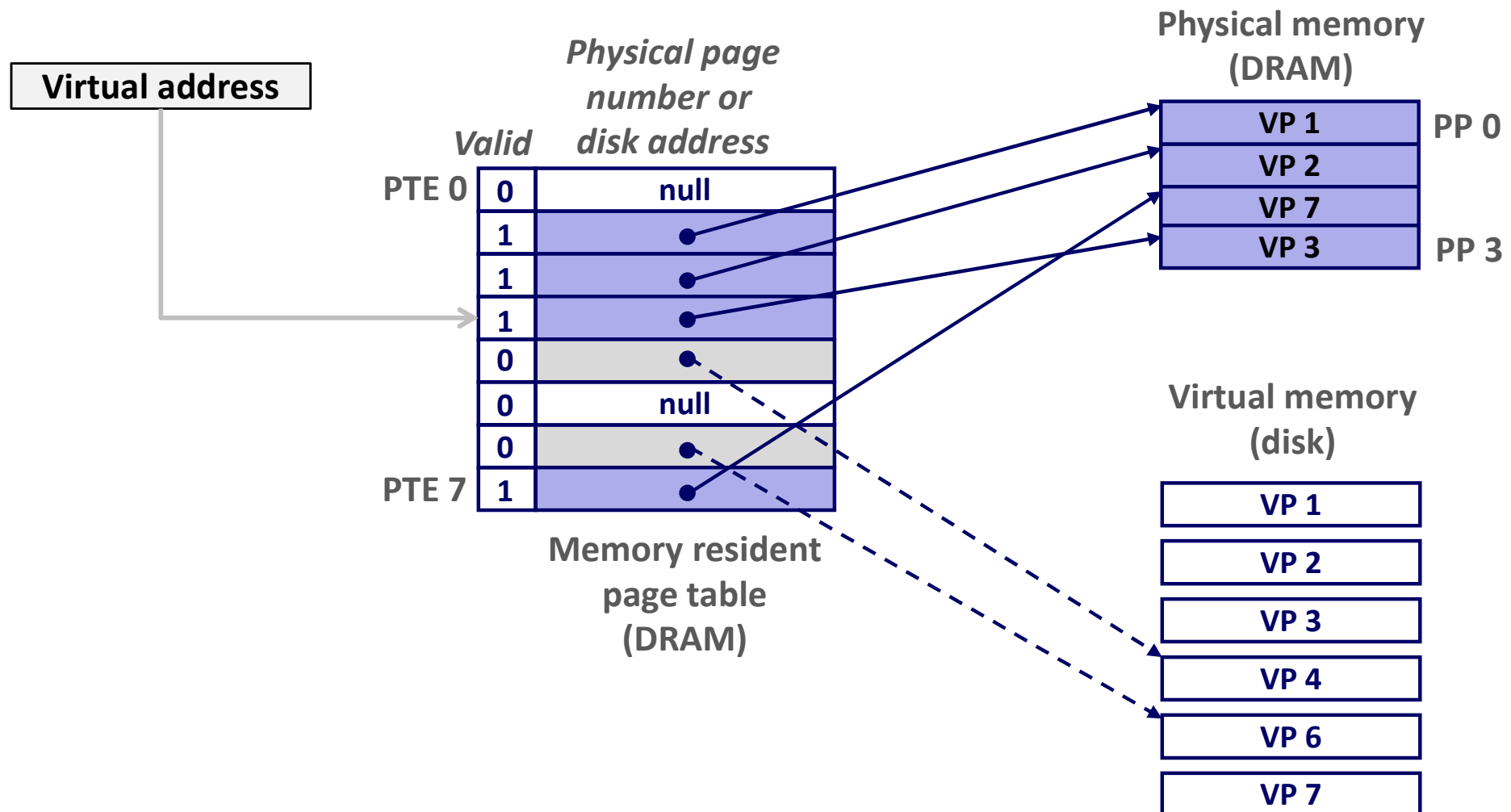
处理缺页异常 (Handling Page Fault)

- 页面未命中 会导致 缺页异常 (page fault, 属于一种异常)
- 缺页异常处理程序 (page fault handler) 会选择一个页 (victim page) 进行置换 (此处选择 VP4)



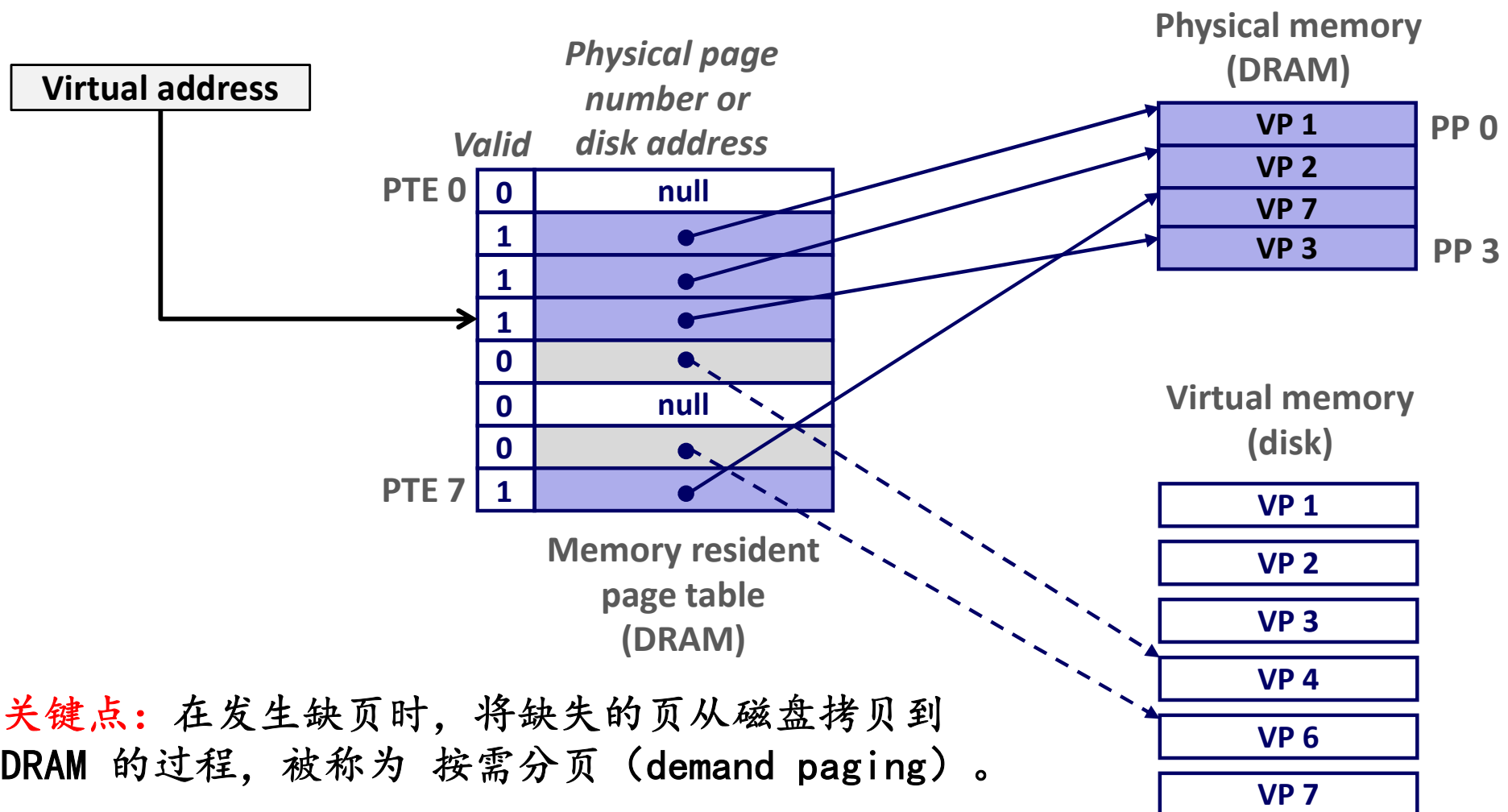
处理缺页异常 (Handling Page Fault)

- 页面未命中 会导致 缺页异常 (page fault, 属于一种异常)
- 缺页异常处理程序 (page fault handler) 会选择一个页 (victim page) 进行置换 (此处选择 VP4)



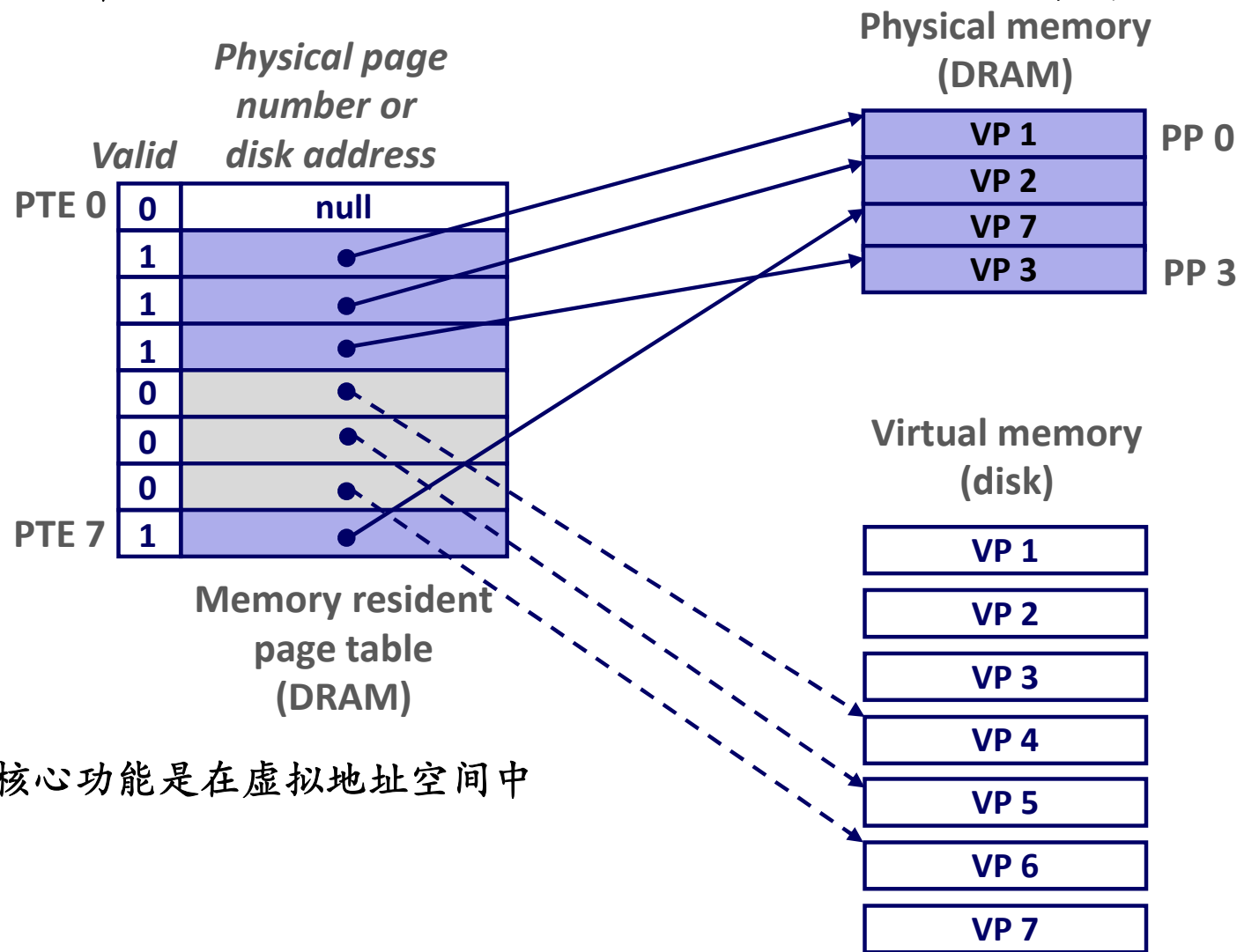
处理缺页异常 (Handling Page Fault)

- 页面未命中会导致缺页异常 (page fault)
- 缺页异常处理程序会选择一个页 (victim page) 进行置换 (这里选择 VP4)
- 导致异常的指令会重新执行, 页命中 (page hit)



分配页面 (Allocating Pages)

- 为虚拟内存分配一个新页面（此处示例为 VP5）
- 例如 C 语言中调用 `malloc()` 函数分配一大块内存



`sbrk()` 底层函数：核心功能是在虚拟地址空间中分配一块新的区域

局部性再次救场！

- 虚拟内存看起来似乎非常低效，但它能够工作，依赖的就是局部性原理（locality）。
- 在任意时间点，程序往往会频繁访问一组活跃的虚拟页，这组页被称为 工作集（working set）。
 - 具有更好时间局部性的程序，其工作集会更小。
- 如果（工作集大小 $<$ 主存大小）
 - 在经历强制性缺页（compulsory misses）之后，单个进程会表现出较好的性能。
- 如果（所有进程的工作集大小之和 $>$ 主存大小）
 - 抖动（Thrashing）：性能会急剧下降，系统会不断地在物理内存和磁盘之间来回交换页面，导致CPU几乎一直在等待数据。

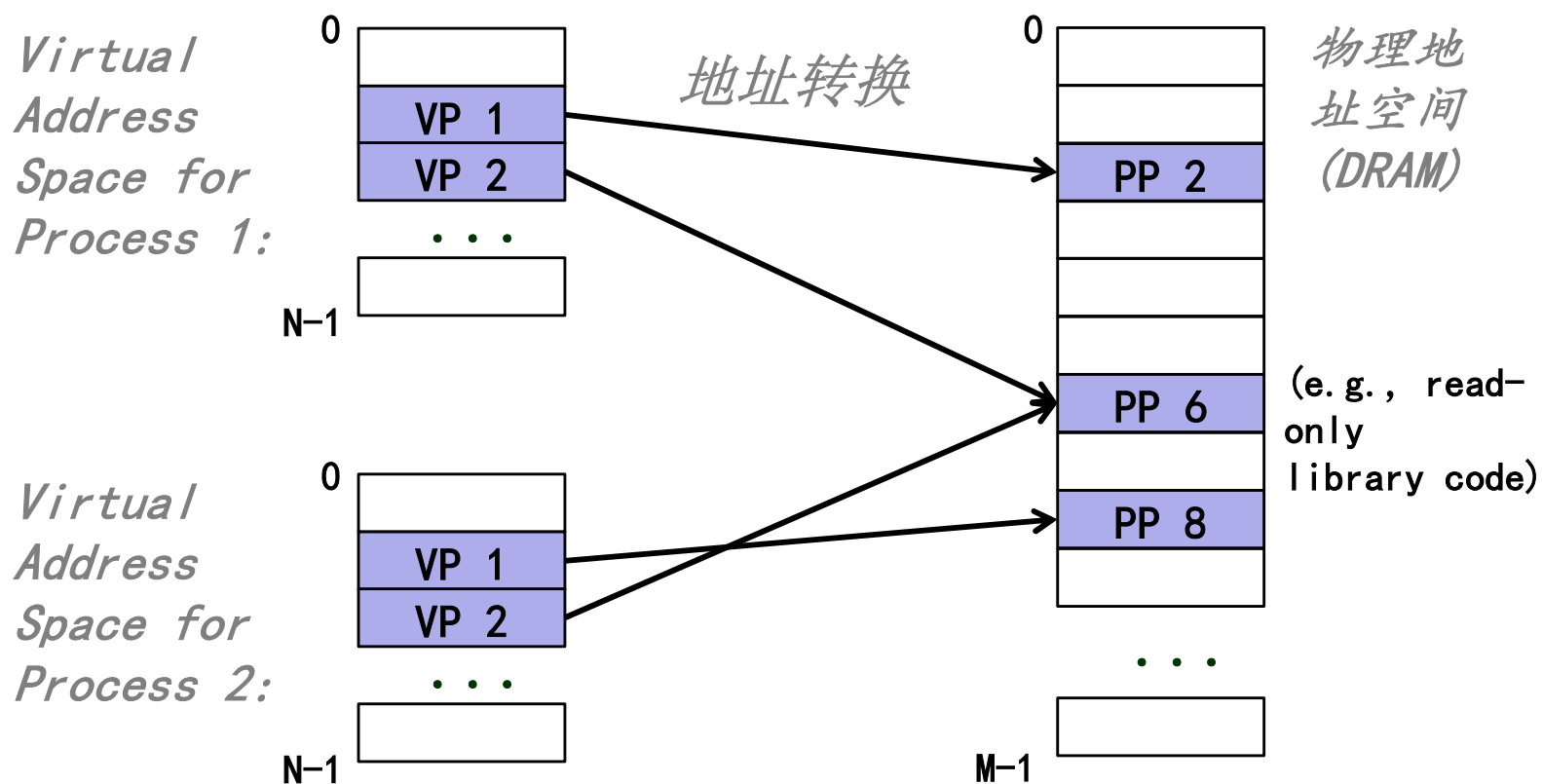
主要内容

- 地址空间
- VM作为缓存工具
- VM作为内存管理工具
- VM作为内存保护工具
- 地址转换

VM作为内存管理工具

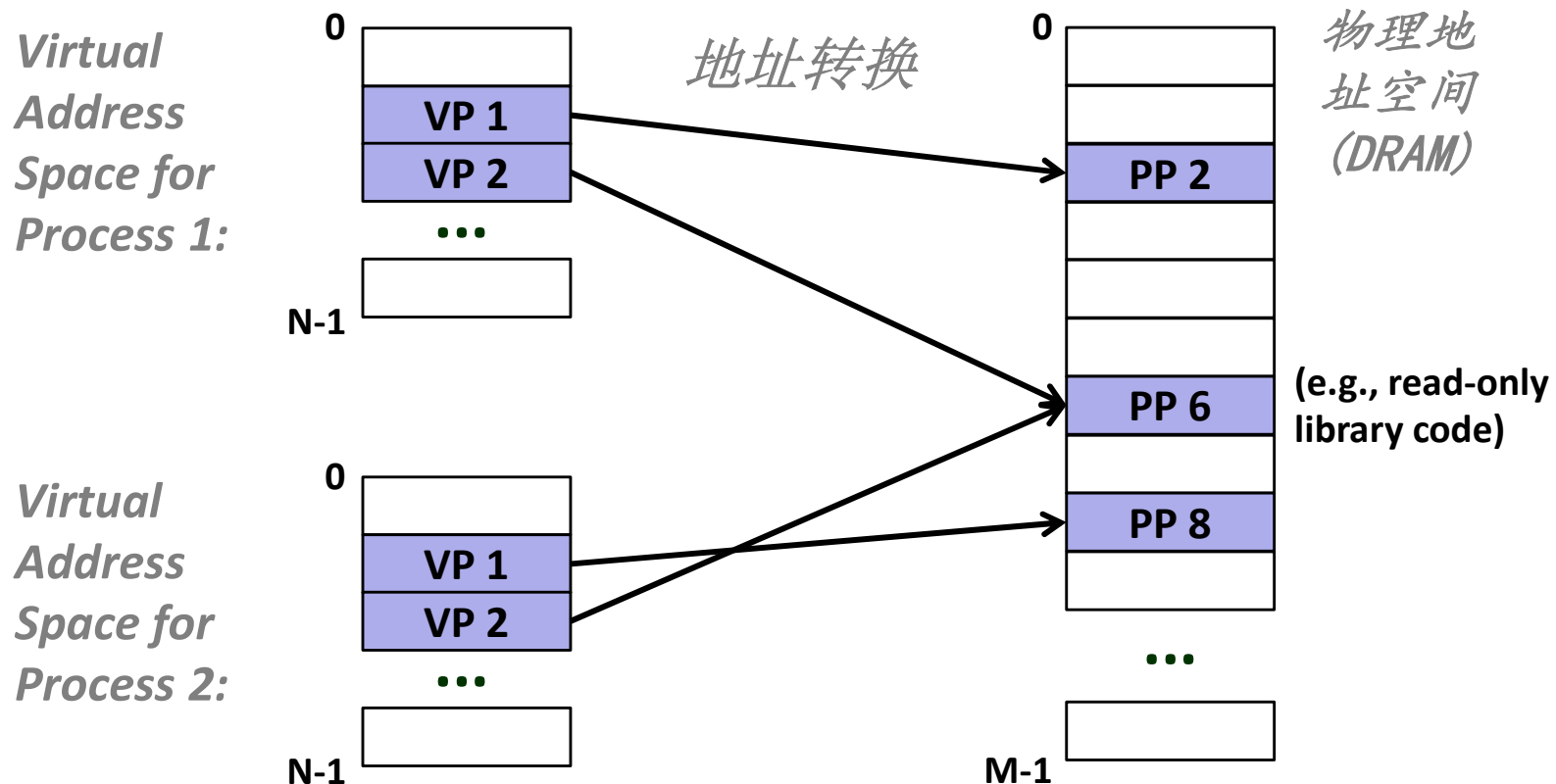
■ 关键思想：每个进程都有自己独立的虚拟地址空间

- 它可以将内存视为一个简单的线性数组
- 映射函数会将地址分散到物理内存中
 - 精心设计的映射可以提高局部性



VM作为内存管理工具

- 简化内存分配 (Simplifying memory allocation)
 - 每个虚拟页可以映射到任意物理页
 - 同一个虚拟页在不同时间可以存储在不同的物理页中
- 在进程之间共享代码和数据 (Sharing code and data among processes)
 - 将多个虚拟页映射到同一个物理页 (此处示例: PP6)

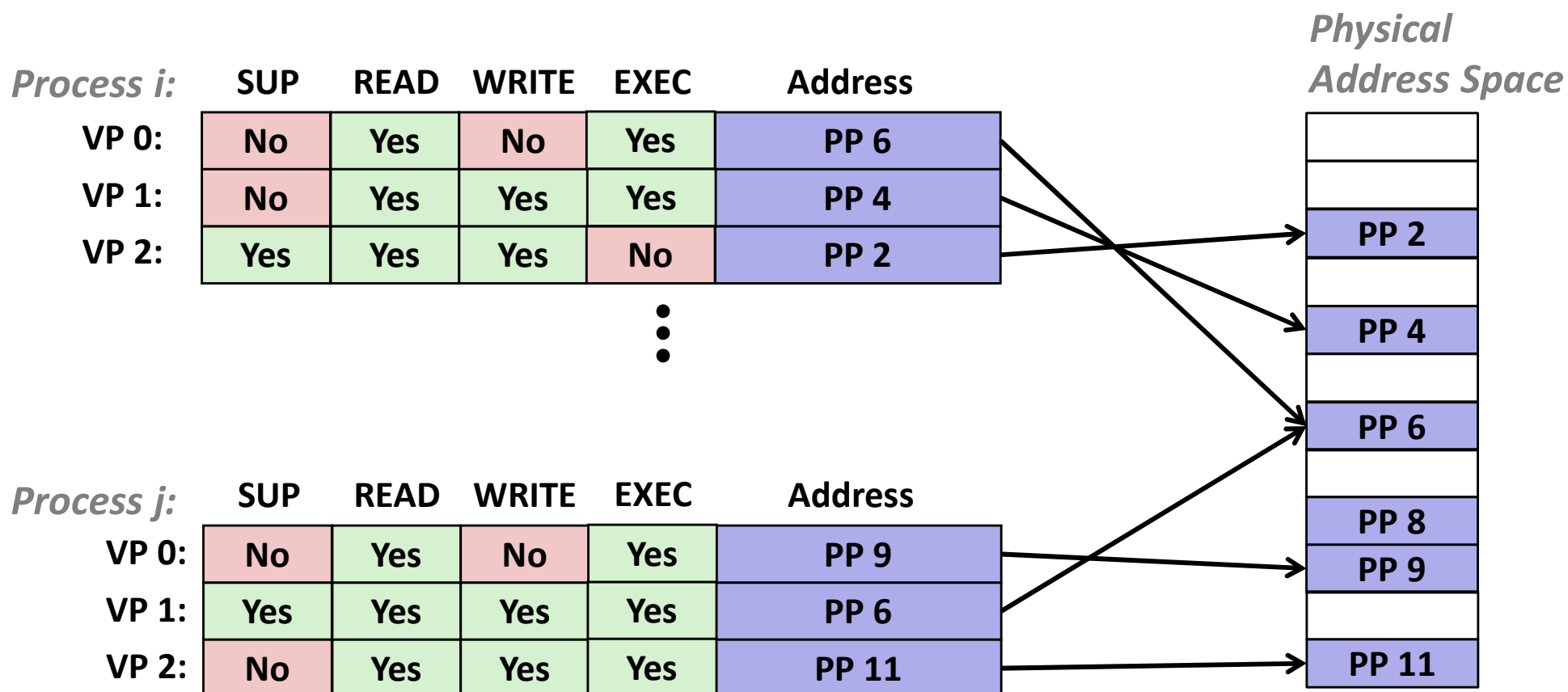


主要内容

- 地址空间
- VM作为缓存工具
- VM作为内存管理工具
- VM作为内存保护工具
- 地址转换

VM作为内存保护工具

- 扩展页表项（**PTEs**）增加权限位（**permission bits**）
- 内存管理单元 **MMU** 检查权限位（**MMU checks permission bits**）



主要内容

- 地址空间
- VM作为缓存工具
- VM作为内存管理工具
- VM作为内存保护工具
- 地址转换/翻译

VM地址翻译/转换

- 虚拟地址空间 (Virtual Address Space)
 - $V = \{0, 1, \dots, N-1\}$
- 物理地址空间 (Physical Address Space) P
 - $P = \{0, 1, \dots, M-1\}$
- 地址转换 (Address Translation)
- 存在一个映射函数：
 - $\text{MAP}: V \rightarrow P \cup \{\emptyset\}$
- 对于给定的虚拟地址 a :
 - 如果数据在物理内存中：
 - $\text{MAP}(a) = a'$, 说明虚拟地址 a 对应的数据存放在物理地址 a' 。
 - 如果数据不在物理内存中：
 - $\text{MAP}(a) = \emptyset$, 说明虚拟地址 a 的数据未加载到内存中, 可能:
 - 当前无效 (invalid)
 - 或者还在磁盘上 (需要缺页中断加载)

地址翻译符号总结

■ 基本参数 (Basic Parameters)

- $N = 2^n$: 虚拟地址空间中可寻址的地址数量
- $M = 2^m$: 物理地址空间中可寻址的地址数量
- $P = 2^p$: 页面大小 (bytes)

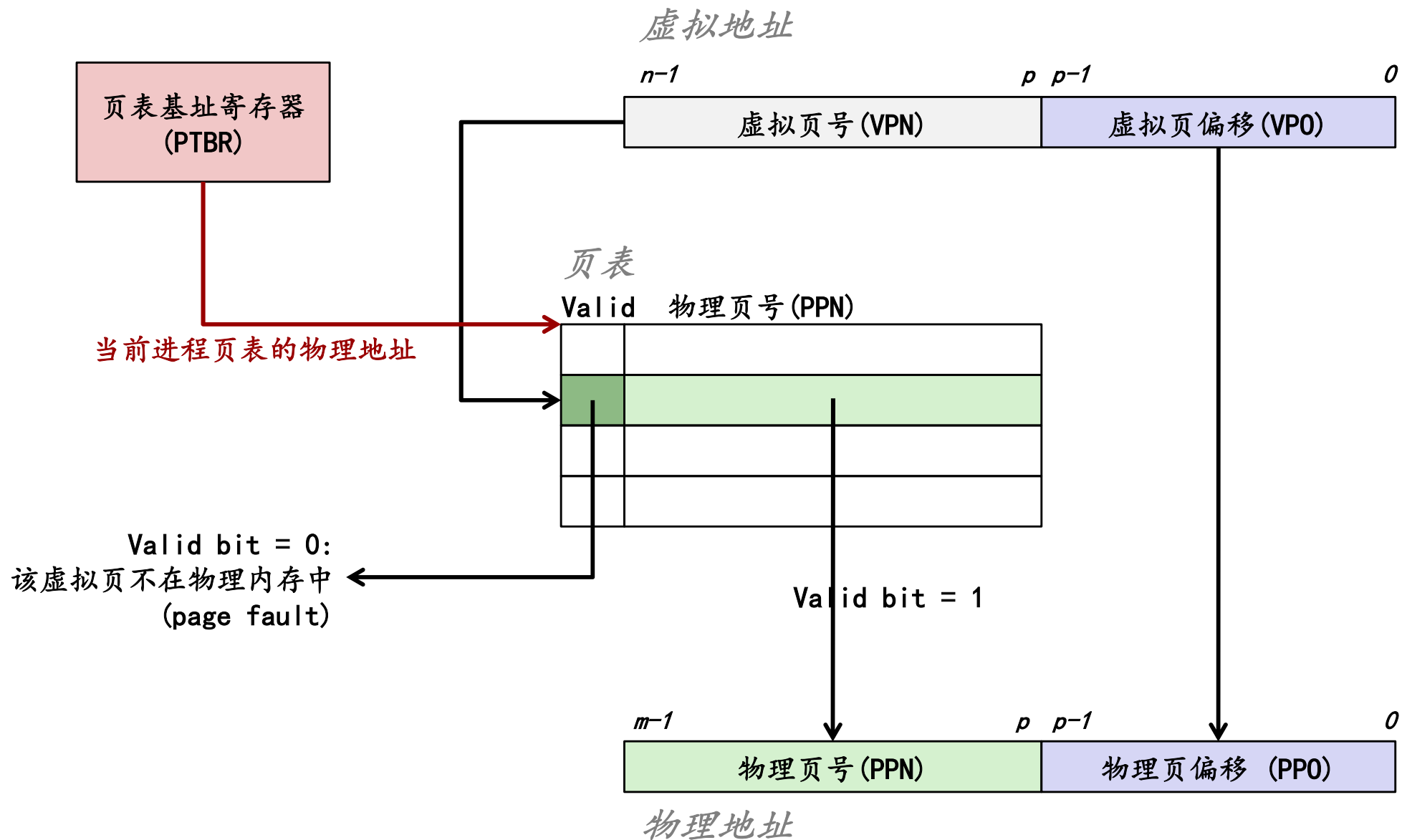
■ 虚拟地址 (VA, Virtual Address) 的组成

- TLBI (TLB Index) : TLB 索引
- TLBT (TLB Tag) : TLB 标签
- VP0 (Virtual Page Offset) : 虚拟页偏移
- VPN (Virtual Page Number) : 虚拟页号

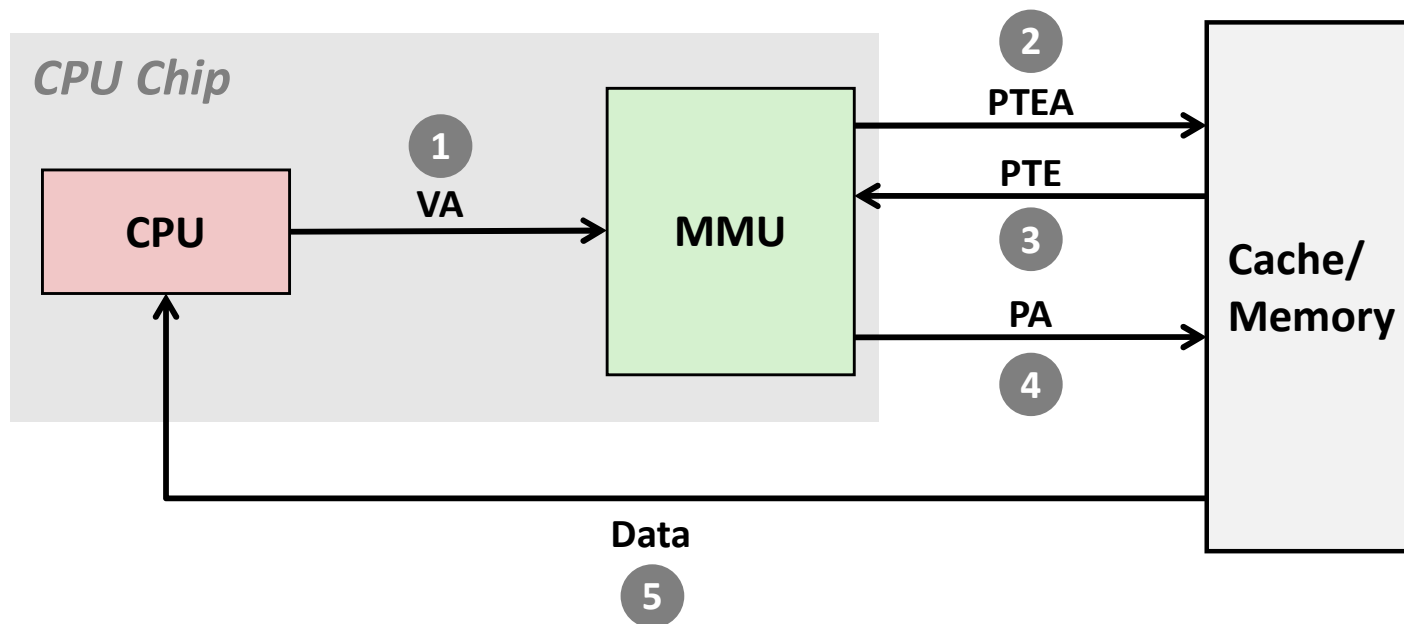
■ 物理地址 (PA, Physical Address) 的组成

- PP0 (Physical Page Offset) : 物理页偏移, 与虚拟页偏移 VP0 完全相同
- PPN (Physical Page Number) : 物理页号

通过页表进行地址翻译

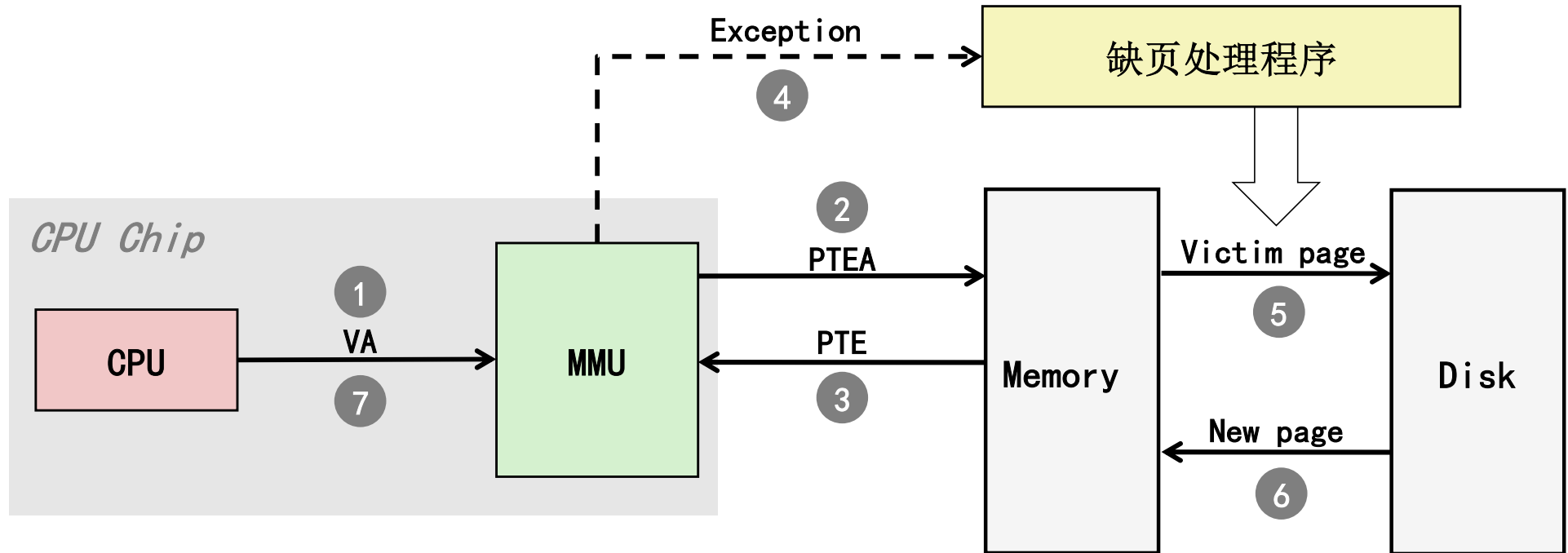


地址翻译：页命中 (Page Hit)



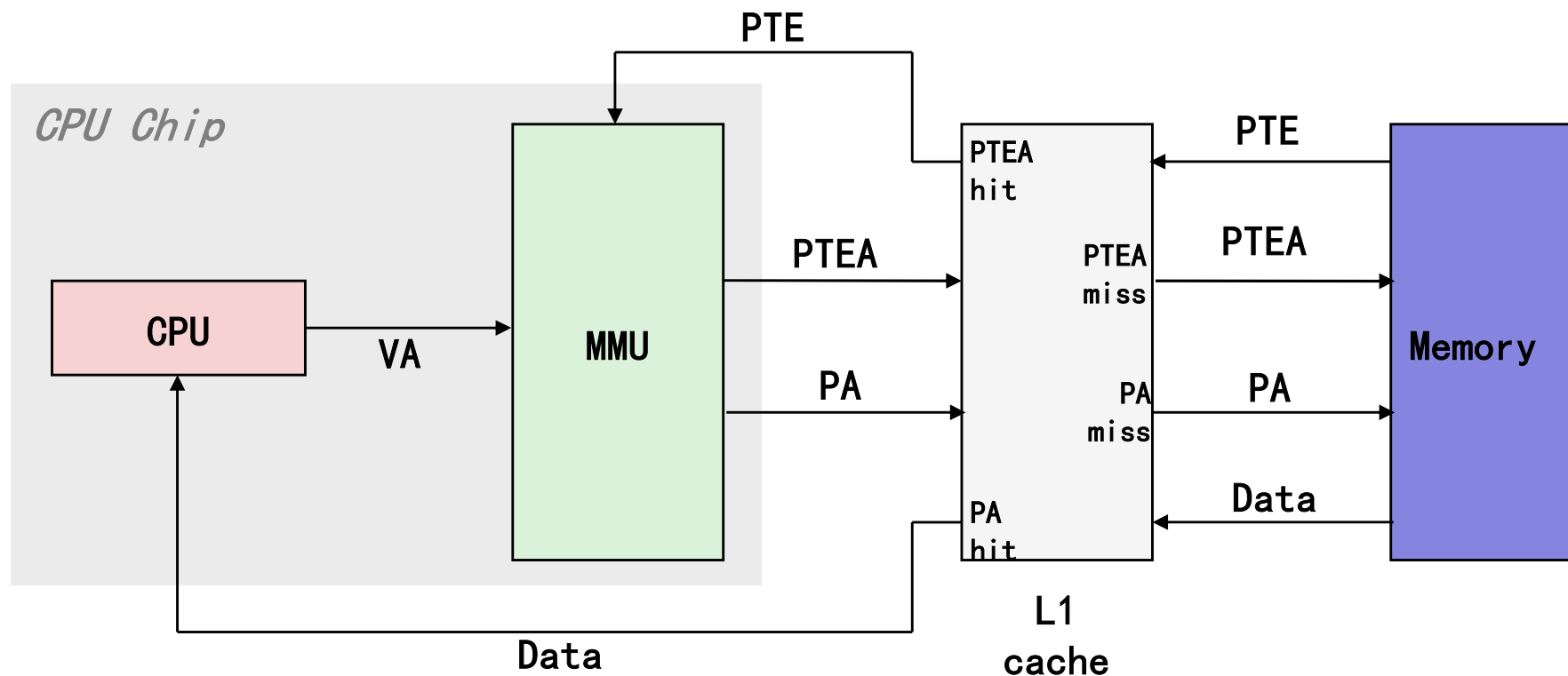
1. 处理器 (CPU) 将**虚拟地址 (VA)**发送给内存管理单元 (MMU)。
2. MMU 根据虚拟地址中的虚拟页号，到内存中的页表查找对应的页表项地址 (PTEA)。
3. MMU 取出页表项 (PTE)，得到对应的物理页号。
4. MMU 将拼接后的**物理地址 (PA)**发送到缓存/内存。
5. 缓存/内存根据物理地址找到数据，并把数据返回给处理器。

地址翻译：缺页中断（Page Fault）



1. 处理器将虚拟地址（VA）发送给 MMU（内存管理单元）。
2. MMU 访问页表项地址（PTEA）
3. MMU 根据虚拟页号，从内存中的页表中获取页表项（PTE）。
4. Valid Bit=0，MMU 会触发缺页异常（Page Fault Exception）。
5. 缺页处理程序（Page Fault Handler）选择一个物理页腾出来（称为 Victim Page），如果 Victim Page 是“脏”的，需要先写回磁盘
6. 缺页处理程序读取新页面，同时更新页表项（PTE）。
7. 缺页处理完成后，处理器重新执行指令，继续运行程序。

额外：虚拟内存与缓存（Cache）的集成



VA (Virtual Address) : 虚拟地址
 PA (Physical Address) : 物理地址
 PTE (Page Table Entry) : 页表项
 PTEA (PTE Address) : 页表项的地址
 L1 Cache: 一级缓存

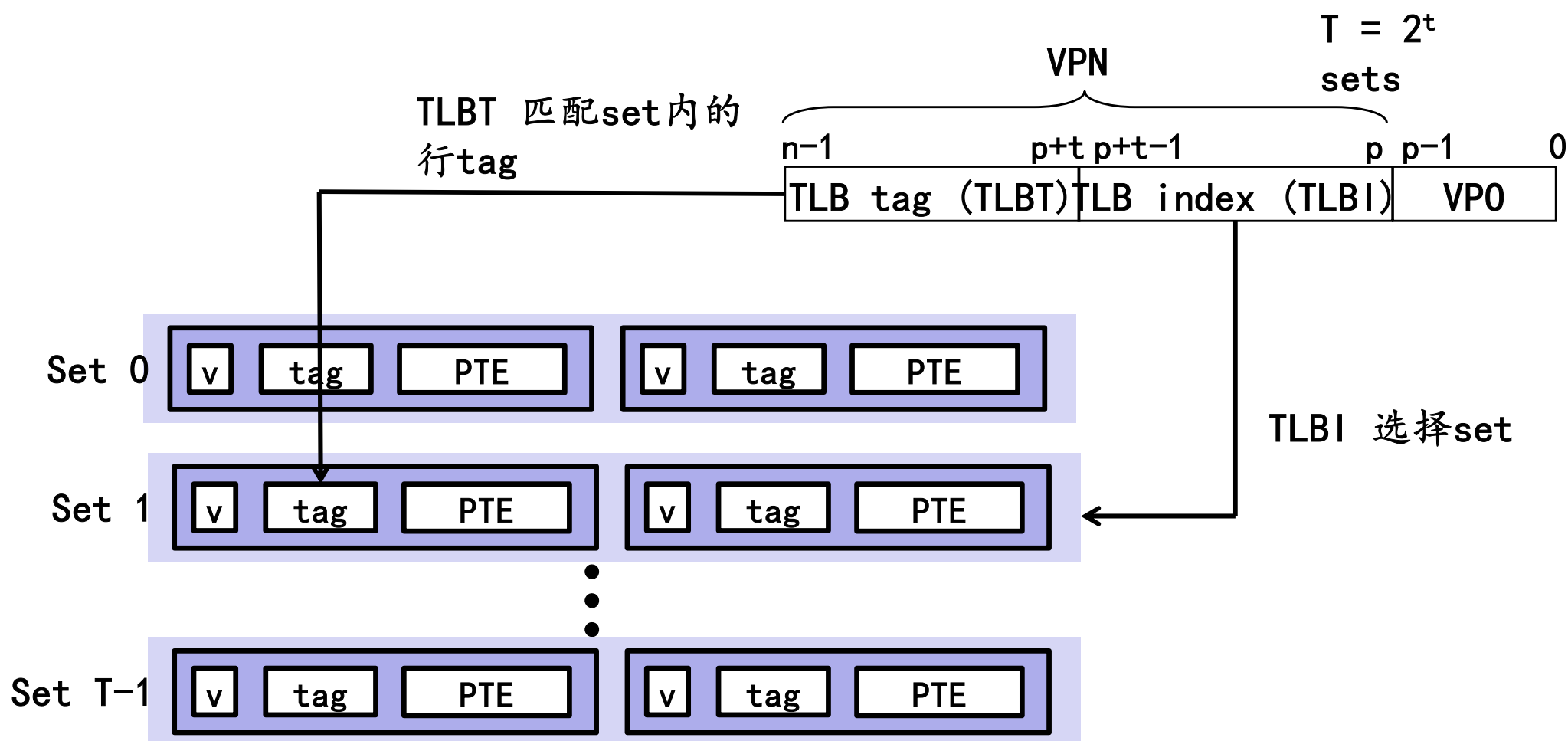
第一步: CPU → MMU, 发送虚拟地址 VA。
 第二步: MMU 根据 VA 提取虚拟页号 (VPN), 查找页表项 (PTE)。如果 PTEA 命中缓存, 直接获得 PTE; 如果 PTEA 未命中, 从主存加载 PTE。
 第三步: 得到物理地址 PA 后, 查询 L1 缓存: 如果 PA 命中缓存 → 直接返回数据; 如果 PA 未命中 → 去主存取数据, 同时把数据写回缓存。第四步: 将最终数据返回给 CPU。

通过TLB加速地址转换

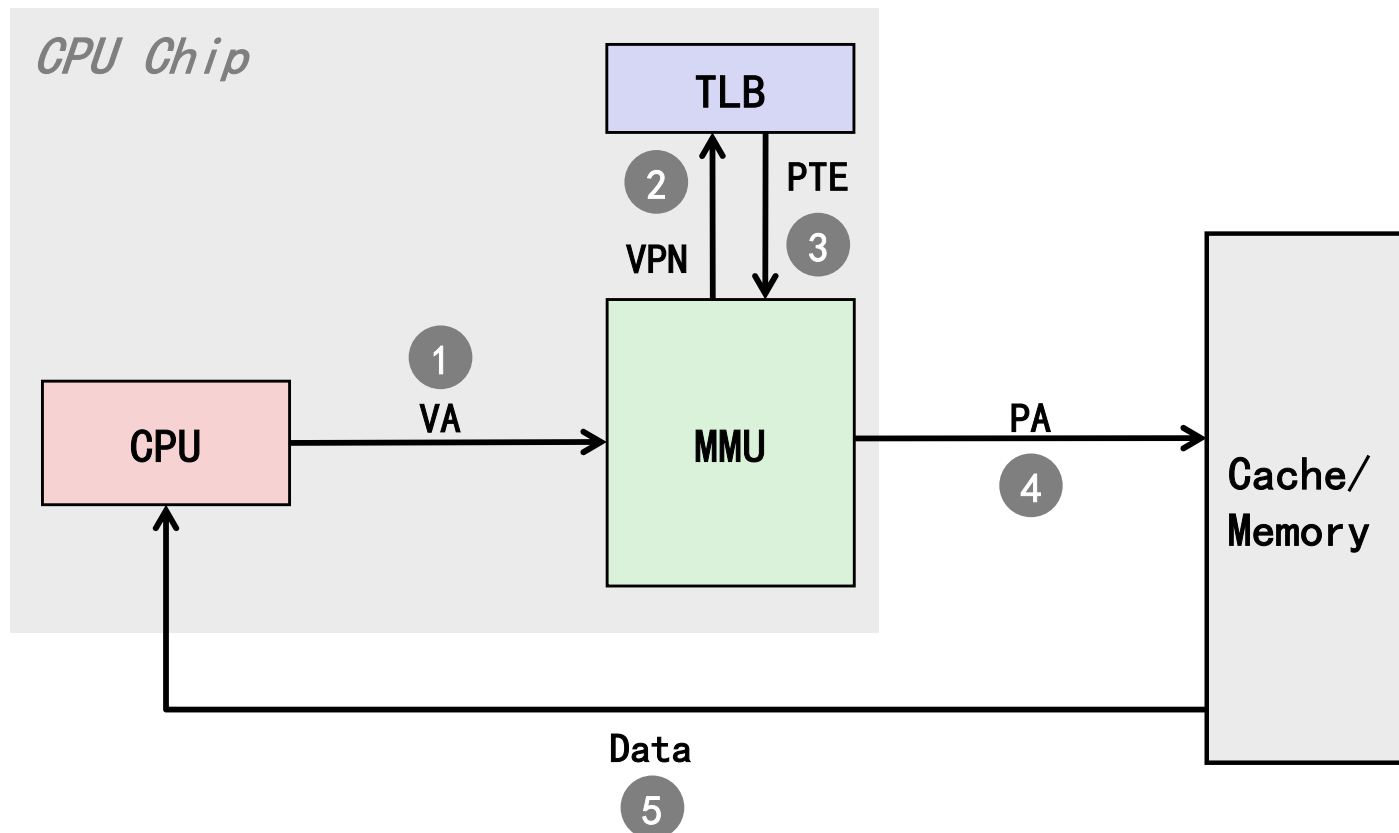
- 页表项（PTE, Page Table Entry）和普通数据一样，被缓存在L1 Cache中
 - PTE可能被逐出，如果L1缓存被其他数据占用
 - 即便PTE命中L1缓存仍然需要一次L1访问延迟
- 解决方案: *Translation Lookaside Buffer* (TLB)
 - TLB是一种专门用于加速虚拟地址到物理地址映射的小型硬件缓存，特点是：
 - TLB通常设计在MMU（内存管理单元）中，采用组相联（set-associative）结构，比L1缓存更高效。
 - 直接存储映射关系，它直接把虚拟页号（VPN）映射到物理页号（PPN），无需查找内存中的页表。
 - 存储少量完整页表项
虽然TLB的容量小，但它能缓存最常用的一部分页表项，从而极大提高转换速度。

访问TLB

- MMU（内存管理单元）使用**虚拟地址（VA）中的虚拟页号（VPN）**部分来访问TLB:

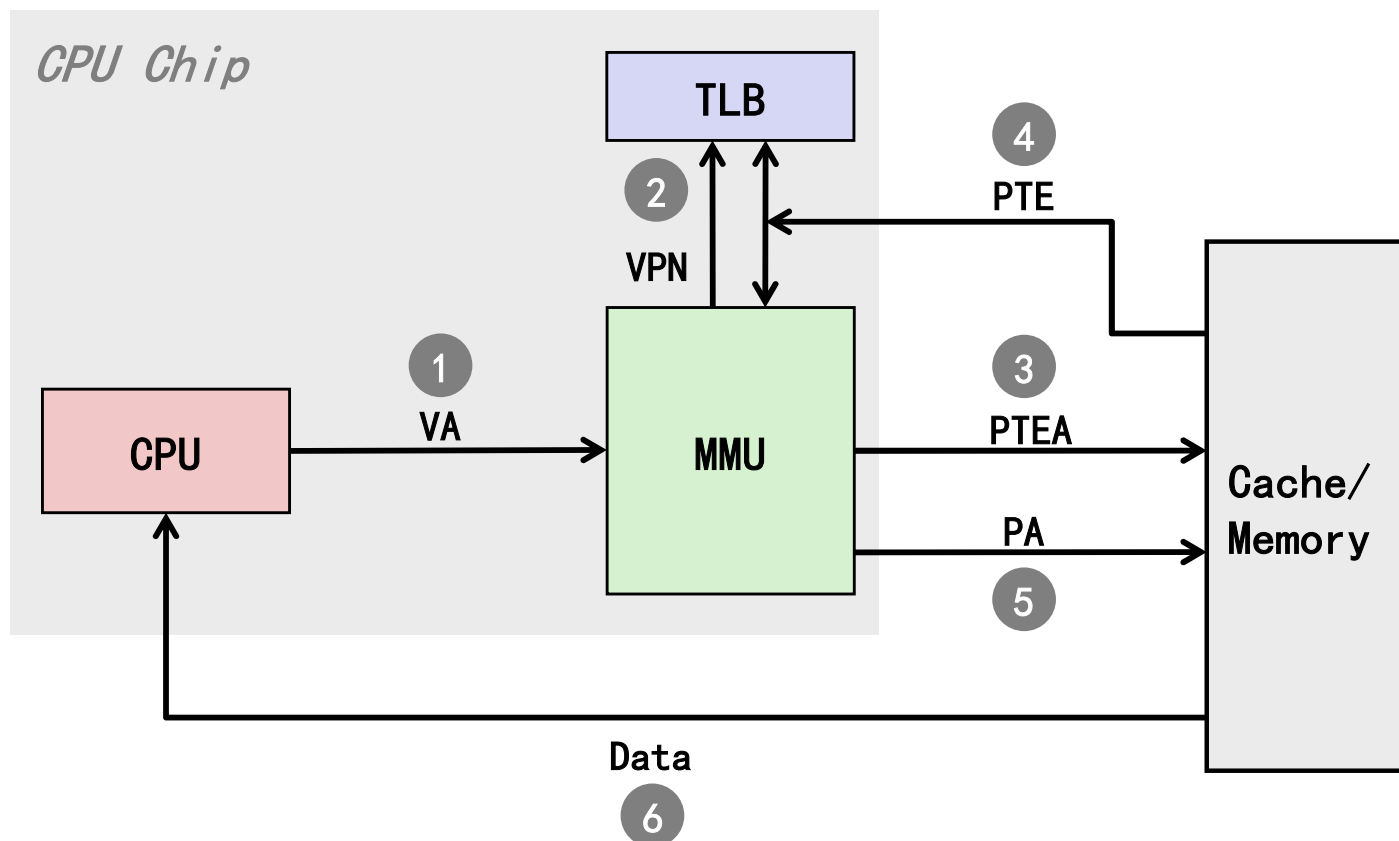


TLB 命中



TLB 命中消除了内存访问

TLB Miss



TLB 未命中会导致额外的内存访问（PTE）。

多级页表 (Multi-Level Page Tables)

■ 假设场景

- 页大小：4KB (2^{12} 字节)，虚拟地址空间：48位，每个页表项 (PTE) 大小：8字节

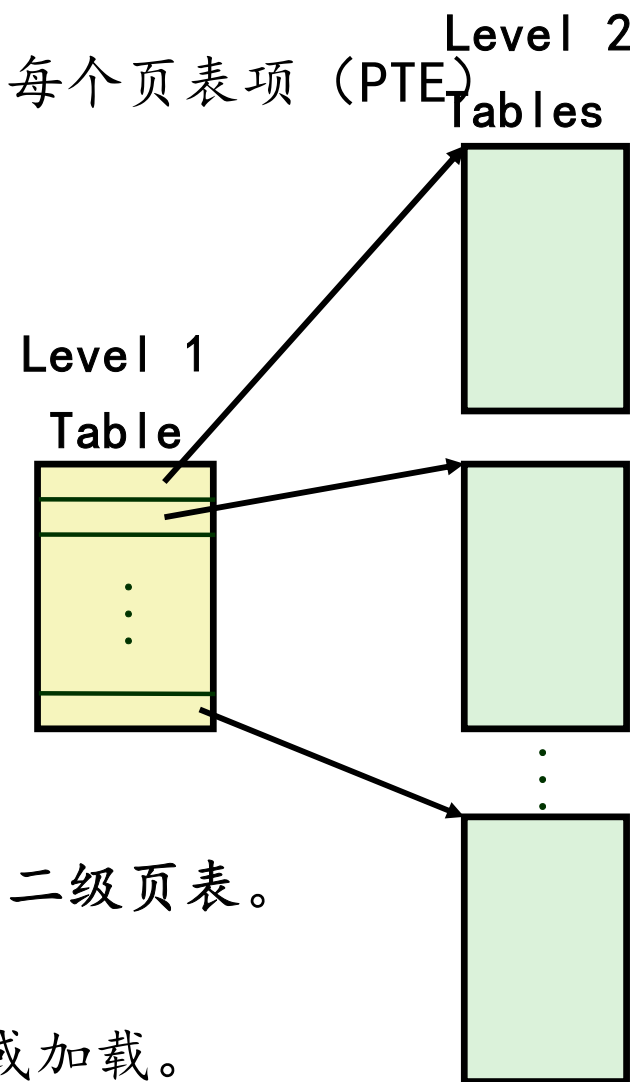
■ 问题：

- 仅仅页表本身就需要512GB内存！
 - $2^{48} * 2^{-12} * 2^3 = 2^{39}$ bytes

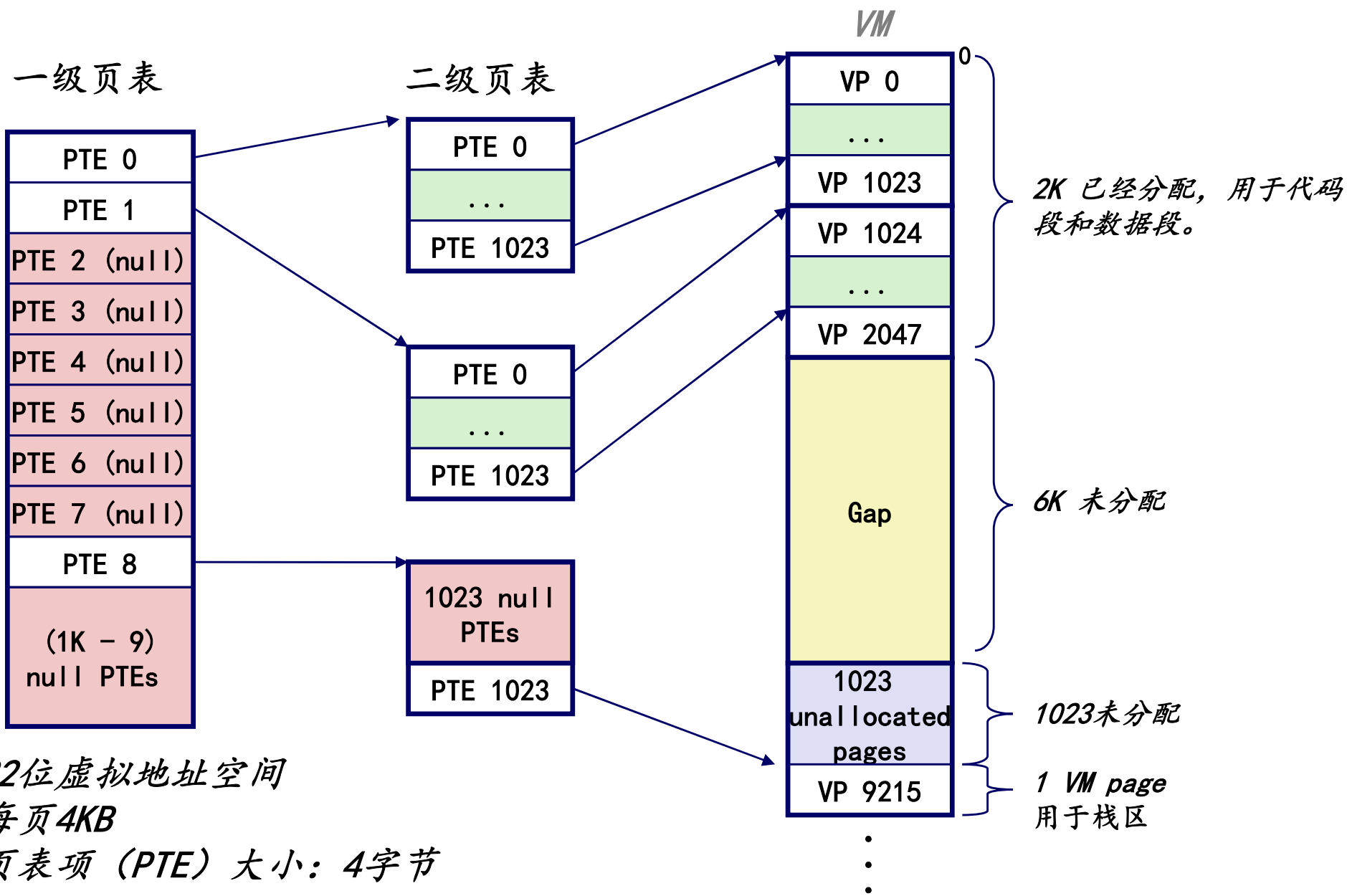
■ 常见解决方案：多级页表

■ 两级页表示例

- 第一级页表 (Level 1 Table)
 - 存放在内存中，始终常驻。
 - 每一项PTE不直接指向物理页面，而是指向第二级页表。
- 第二级页表 (Level 2 Table)
 - 只有当对应虚拟页范围被访问时，才会创建或加载。
 - 每一项PTE指向具体的物理页。
 - 当不需要时，二级页表本身也可以被换出，就像普通数据一样。

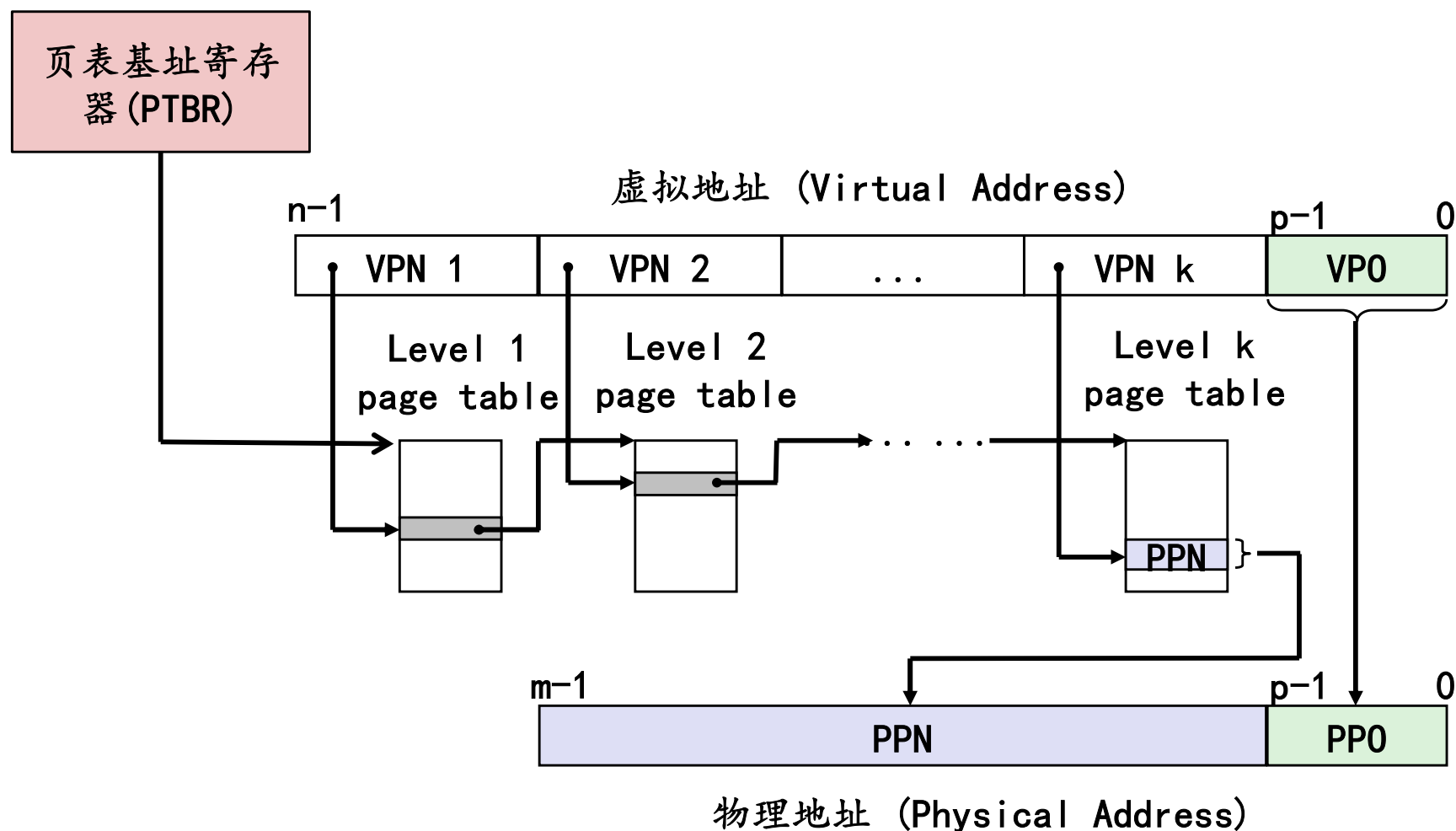


两级页表层次结构



- 32位虚拟地址空间
- 每页4KB
- 页表项 (PTE) 大小: 4字节

使用 k 级页表进行地址转换



总结

- 1. 程序员视角下的虚拟内存 (Programmer's view of virtual memory)
 - 每个进程拥有自己独立的线性地址空间
 - 程序员看到的是一片连续、完整、属于自己的内存。
 - 进程之间相互隔离
 - 一个进程的地址空间不会被其他进程破坏。
- 2. 系统视角下的虚拟内存 (System view of virtual memory)
 - 高效利用内存
 - 系统通过缓存虚拟内存页面来减少频繁的磁盘访问。
 - 效率高的原因是局部性原理 (locality)：程序大多数时间只访问一小部分数据。
 - 简化内存管理和编程
 - 操作系统帮程序员管理物理内存和地址映射，程序员不需要直接操心。
 - 简化内存保护
 - 虚拟内存提供了方便的检查点，操作系统能在这里验证访问权限，从而保护数据安全。