虚拟内存

Virtual Memory

本章内容

Topic

- 地址空间

  Address Spaces
- 虚拟内存作为缓存的工具

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

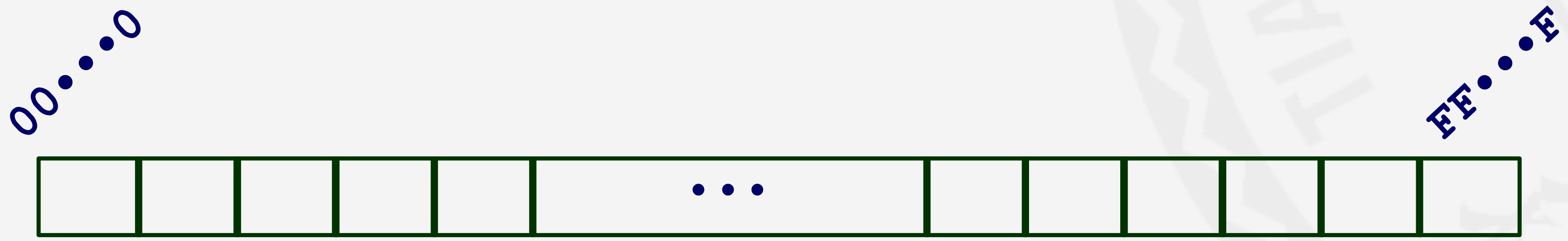
  VM as a tool for memory management
- 虚拟内存作为内存保护的工具

  VM as a tool for memory protection

地址空间

Address Spaces

回忆：面向字节的存储器组织
Recall: Byte-Oriented Memory Organization



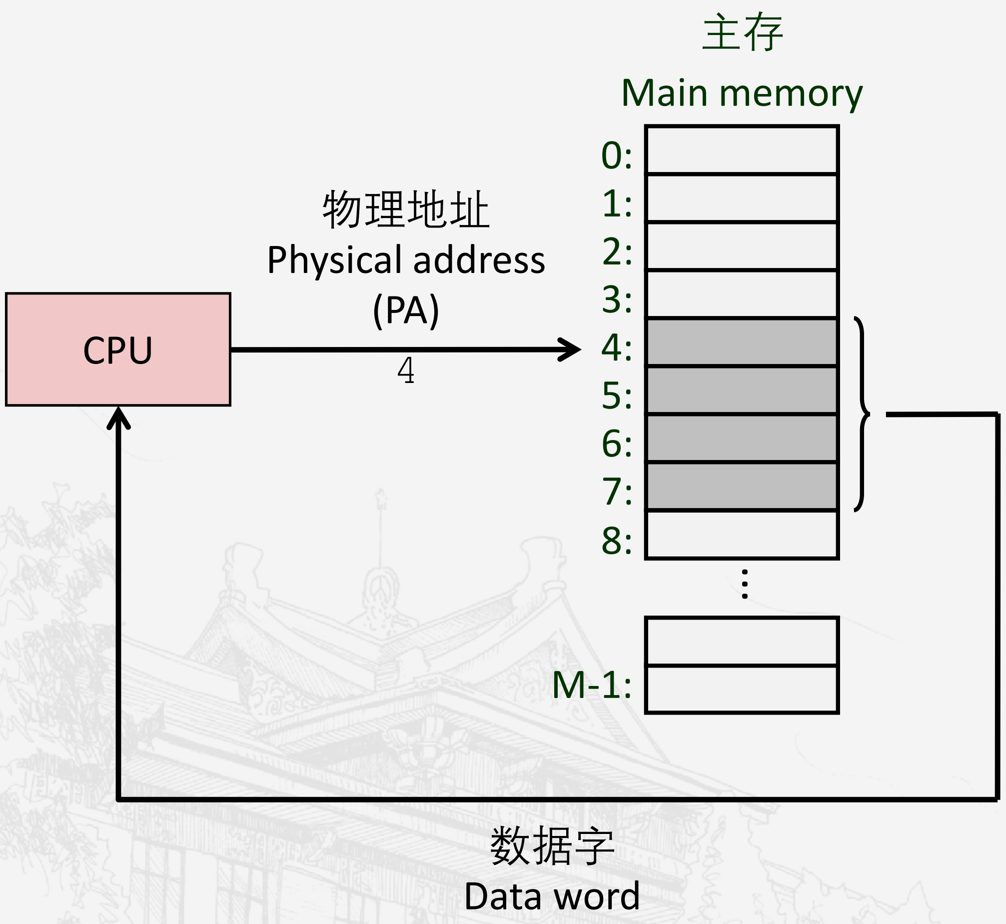
- 程序根据地址引用数据
  Programs refer to data by address
  - 概念上，可将其想象为一个非常大的字节数组
    Conceptually, envision it as a very large array of bytes
    - 事实上不是，但可以这么认为
      In reality, it’s not, but can think of it that way
  - 地址就像数组的下标
    An address is like an index into that array
    - 用指针来存储地址
      a pointer variable stores an address

- 系统为每个进程提供了一个私有地址空间
  Programs refer to data by address
  - 一个进程可以访问自身的数据，但不能破坏其他进程的数据
    A process can clobber its own data, but not that of others

地址空间

Address Spaces

一个使用物理寻址的系统
A System Using Physical Addressing

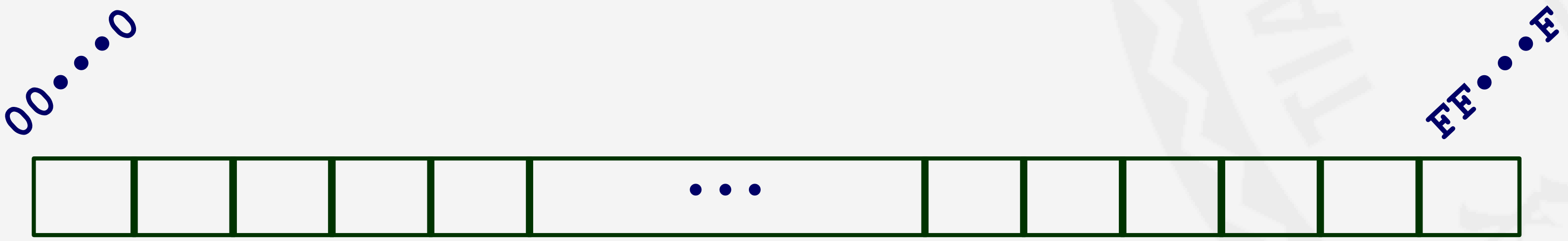


- 在一些简单的系统中应用这种方式进行寻址，如汽车、电梯和数码相框中的嵌入式微控制器
  Used in some “simple” systems, like embedded microcontrollers in cars, elevators, and digital picture frames

地址空间

Address Spaces

让我们考虑一下
Lets think about this, a bit



- 如何来决定你的程序到底使用哪些内存?
  How to decide which memory to use in your program?
- 如果另一个进程把数据存储到当前进程所使用的内存中怎么办?
  What if another process stores data into your memory?

地址空间

Address Spaces

所以，加入了一个间接层
So, we add a level of indirection

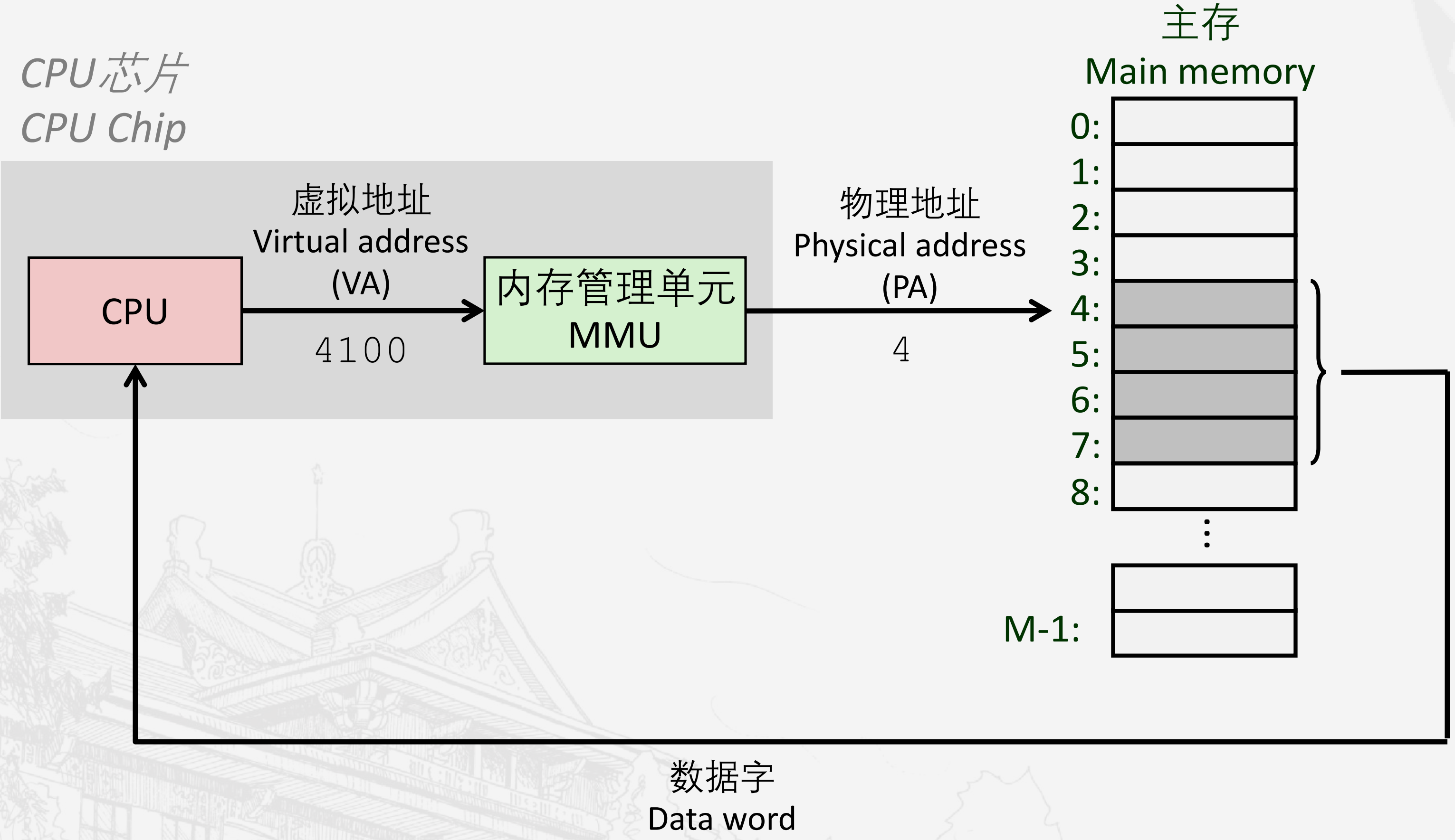
- 一个简单的技巧可以解决上面的问题:
  One simple trick solves all above problems:
- 每个进程具有自己私有的内存空间
  What if another process stores data into your memory?
  - 表现出是一个私有的、完整的内存空间
    Appears to be a full-sized private memory range
- 这解决了"如何选择内存"以及"其他人不会干扰当前进程内存"的问题
  This fixes "how to choose" and "others shouldn't mess with yours"

- 实现：透明的地址翻译过程
  Implementation: translate addresses transparently
  - 增加了一个映射函数
    Add a mapping function
    - 将（进程的）私有地址映射到物理地址
      to map private addresses to physical addresses
  - 在每一次内存加载和存储时都会发生
    do the mapping on every load or store
- 地址映射是虚拟内存的核心技术
  This mapping trick is the heart of virtual memory

地址空间

Address Spaces

一个使用虚拟寻址的系统
A System Using Virtual Addressing



- 在现代的服务器、桌面、笔记本电脑和绝大多数的智能手机中，都使用虚拟寻址系统
  Used in all modern servers, desktops, laptops and most smartphones
- 是计算机科学中的重要思想之一
  One of the great ideas in computer science

地址空间
Address Spaces

- **线性地址空间**：连续的非负整数地址的有序集合（为了简化，假设使用的是线性地址空间）
  Linear address space: Ordered set of contiguous non-negative integer addresses
  {0, 1, 2, 3 … }
- **虚拟地址空间**： N = 2^n 个虚拟地址所构成的集合
  Virtual address space: Set of N = 2n virtual addresses
  {0, 1, 2, 3, …, N-1}
- **物理地址空间**： M = 2^m 个物理地址所构成的集合
  Physical address space: Set of M = 2m physical addresses
  {0, 1, 2, 3, …, M-1}

- 请注意区分数据（字节）和它的属性（地址）
  Clean distinction between data (bytes) and their attributes (addresses)
- 现在，每个数据可能会有多个地址
  Each datum can now have multiple addresses
- 主存中的每个字节：一个物理地址，一个或多个虚拟地址
  Every byte in main memory: one physical address, one (or more) virtual addresses

地址空间

Address Spaces

为什么需要虚拟内存?
Why Virtual Memory?

- 更加有效率的利用主存
  Uses main memory efficiently
  - 将DRAM作为缓存，用于缓冲虚拟地址空间中的部分数据
    Use RAM as a cache for the parts of a virtual address space
- 简化内存管理
  Simplifies memory management
  - 每个进程都能够获得相同的、一致的、线性的地址空间
    Each process gets the same uniform linear address space
- 独立的地址空间
  Isolates address spaces
  - 一个进程不会影响另一个进程的内存
    One process can't interfere with another's memory
  - 用户程序不能够访问内核私有的数据和代码
    User program cannot access privileged kernel data and code

本章内容

Topic

- 地址空间

  Address Spaces

- 虚拟内存作为缓存的工具

  VM as a tool for caching

- 虚拟内存作为内存管理的工具

  VM as a tool for memory management

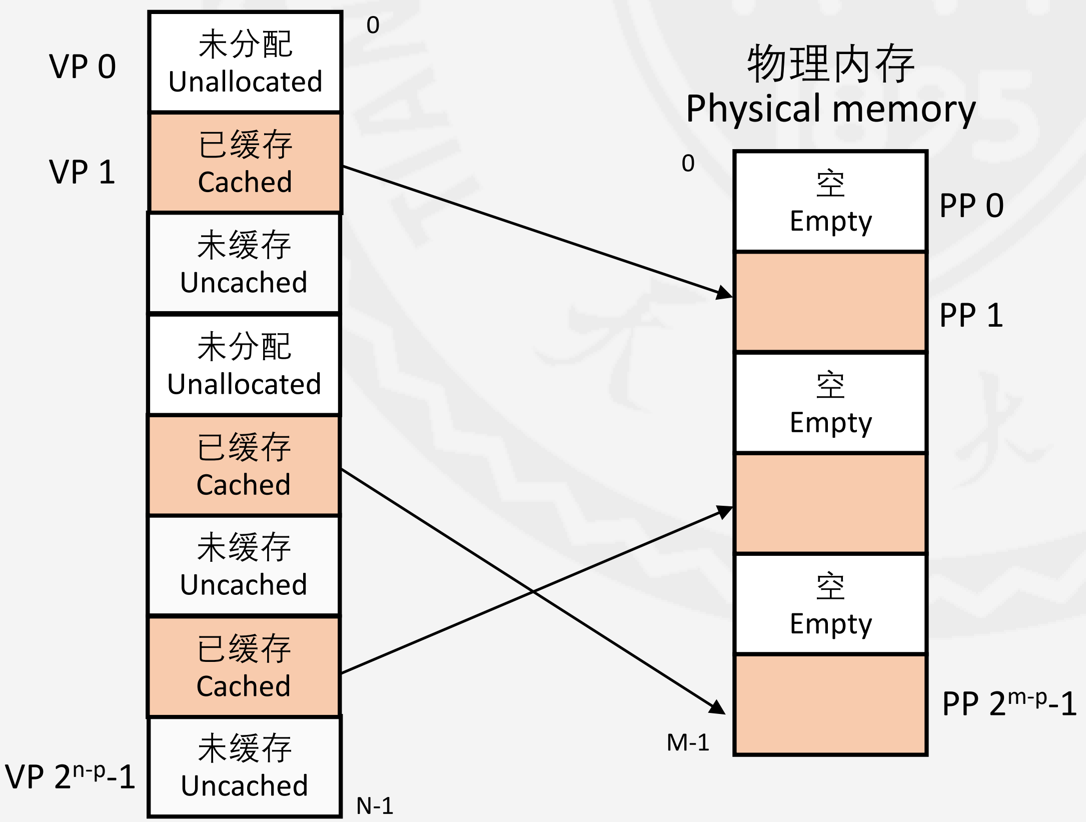
- 虚拟内存作为内存保护的工具

  VM as a tool for memory protection

虚拟内存作为缓存的工具

VM as a tool for caching

- 虚拟内存是一个包含N个连续字节的数组，可以被存储在磁盘上
  Virtual memory is an array of N contiguous bytes that may be stored on disk
- 磁盘上的数组被缓存在物理内存中（DRAM 缓存）
  The contents of the array on disk are cached in physical memory (DRAM cache)
  - 这些缓存块被称为“页”（大小为 $P = 2^p$ 字节）
    These cache blocks are called pages (size is $P = 2^p$ bytes)



虚拟页
存储在磁盘上
Virtual pages (VPs)
stored on disk

物理页
缓存在DRAM中
Physical pages (PPs)
cached in DRAM

虚拟内存作为缓存的工具

VM as a tool for caching

DRAM缓存的组织
DRAM Cache Organization

- DRAM缓存组织设计时主要的考虑因素是巨大的未命中惩罚
  DRAM cache organization driven by the enormous miss penalty
  - DRAM约比SRAM慢10倍
    DRAM is about 10x slower than SRAM
  - 磁盘约比DRAM慢100,000倍
    Disk is about 100,000x slower than DRAM

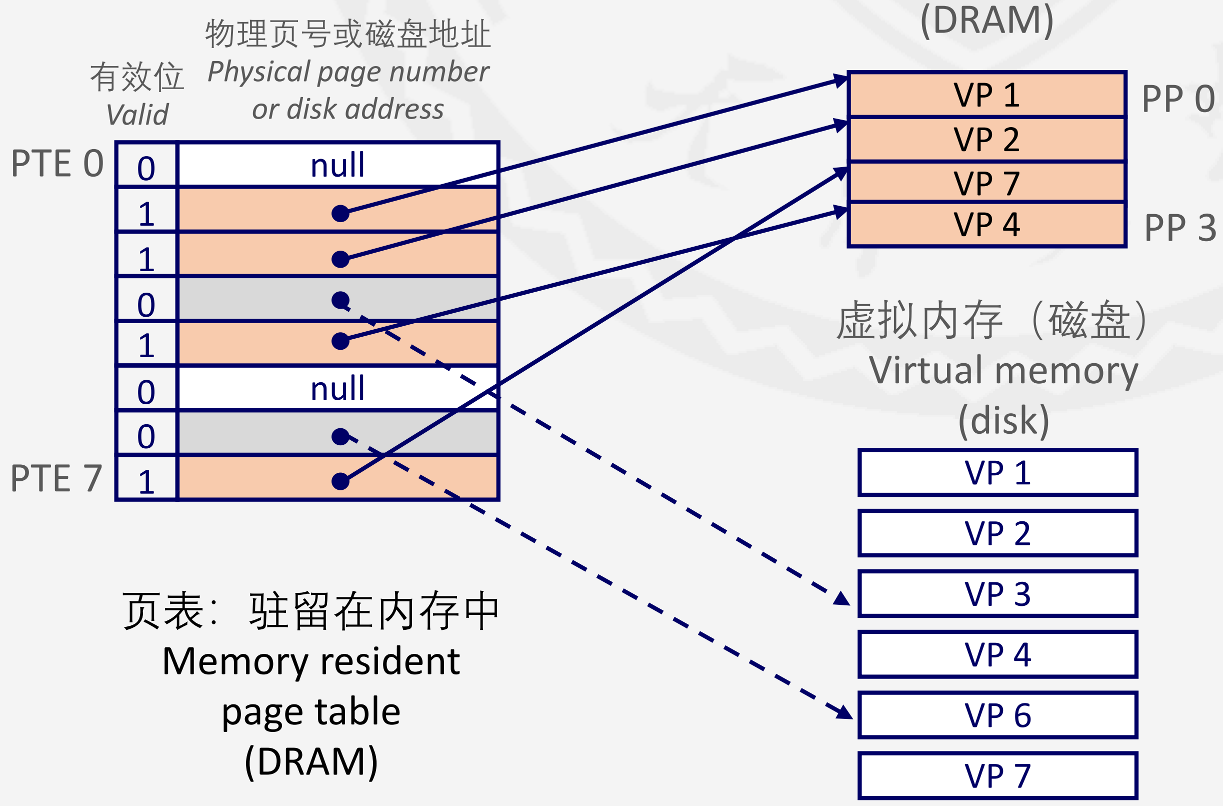
- 导致
  Consequences
  - 更大的页（块）大小：典型值4KB~2MB
    Large page (block) size: typically 4 KB ~ 2MB
  - 全相联
    Fully associative
    - 任意的虚拟页（VP）可以被缓存到任意的物理页（PP）中
      Any virtual page (VP) can be placed in any physical page (PP)
    - 与CPU的高速缓存不同，这需要一个大的映射函数
      Requires a “large” mapping function – different from CPU caches
  - 高度复杂的、代价更大的页替换算法
    Highly sophisticated, expensive replacement algorithms
    - 替换算法复杂且无限制，所以不能用硬件实现
      Too complicated and open-ended to be implemented in hardware
  - 采用回写策略，而不是直写策略
    Write-back rather than write-through

虚拟内存作为缓存的工具

VM as a tool for caching

启用一种新的数据结构：页表
Enabling data structure: Page Table

- **页表**是一个**页表项（PTE）**的数组，用于建立虚拟页到物理页的映射
  A **page table** is an array of **page table entries (PTEs)** that maps virtual pages to physical pages
  - 每个进程在**内核**中都会存在这样的一个数据结构，存储在DRAM中
    Per-process **kernel** data structure in DRAM

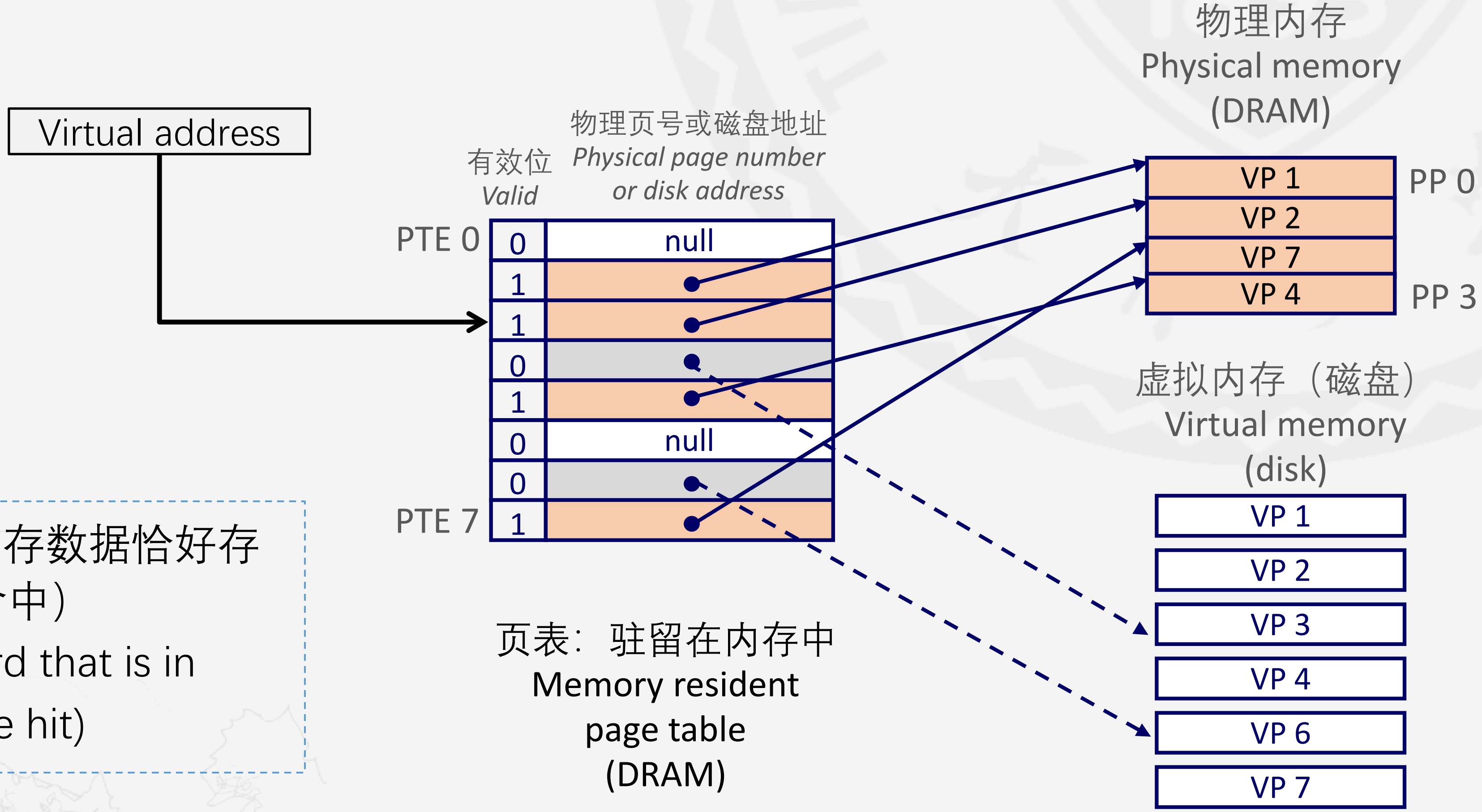


页表：驻留在内存中
Memory resident
page table
(DRAM)

虚拟内存作为缓存的工具

VM as a tool for caching

页命中
Page Hit

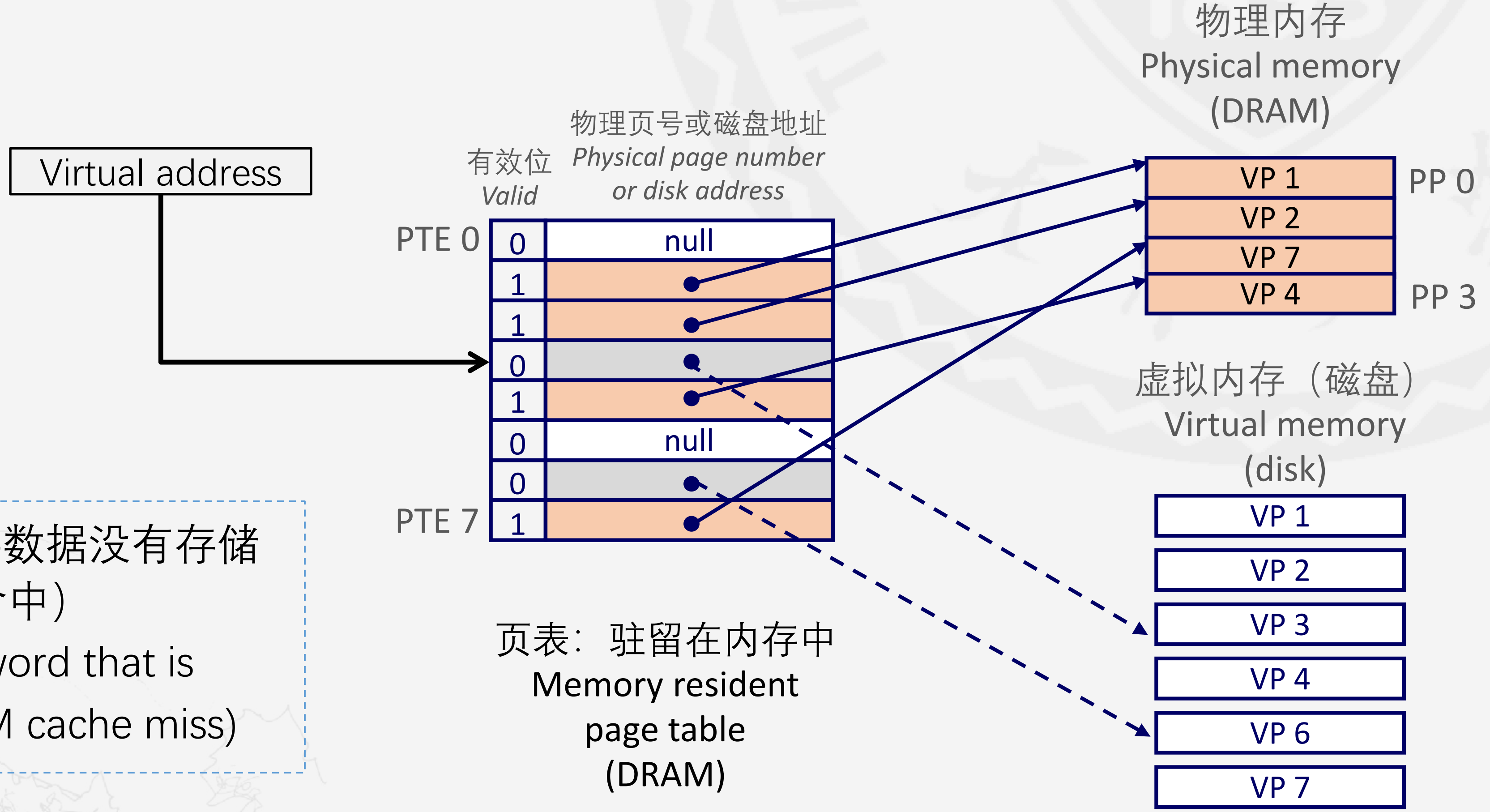


- **页命中**：此时所引用的虚拟内存数据恰好存储在物理内存中（DRAM缓存命中）
  Page hit: reference to VM word that is in physical memory (DRAM cache hit)

虚拟内存作为缓存的工具

VM as a tool for caching

缺页
Page Fault



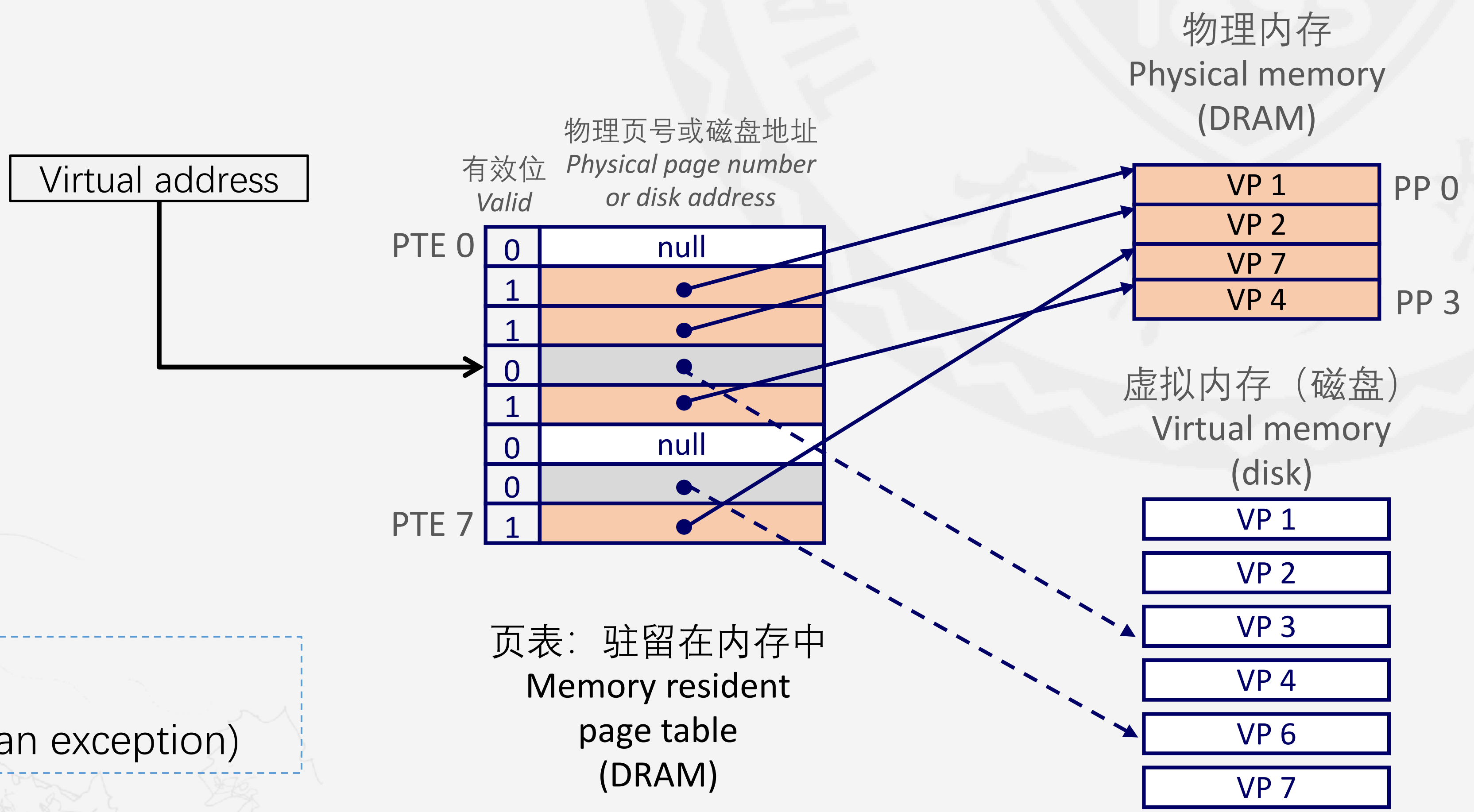
缺页：此时所引用的虚拟内存数据没有存储在物理内存中（DRAM缓存未命中）

Page Fault: reference to VM word that is not in physical memory (DRAM cache miss)

虚拟内存作为缓存的工具

VM as a tool for caching

缺页处理
Handling Page Fault

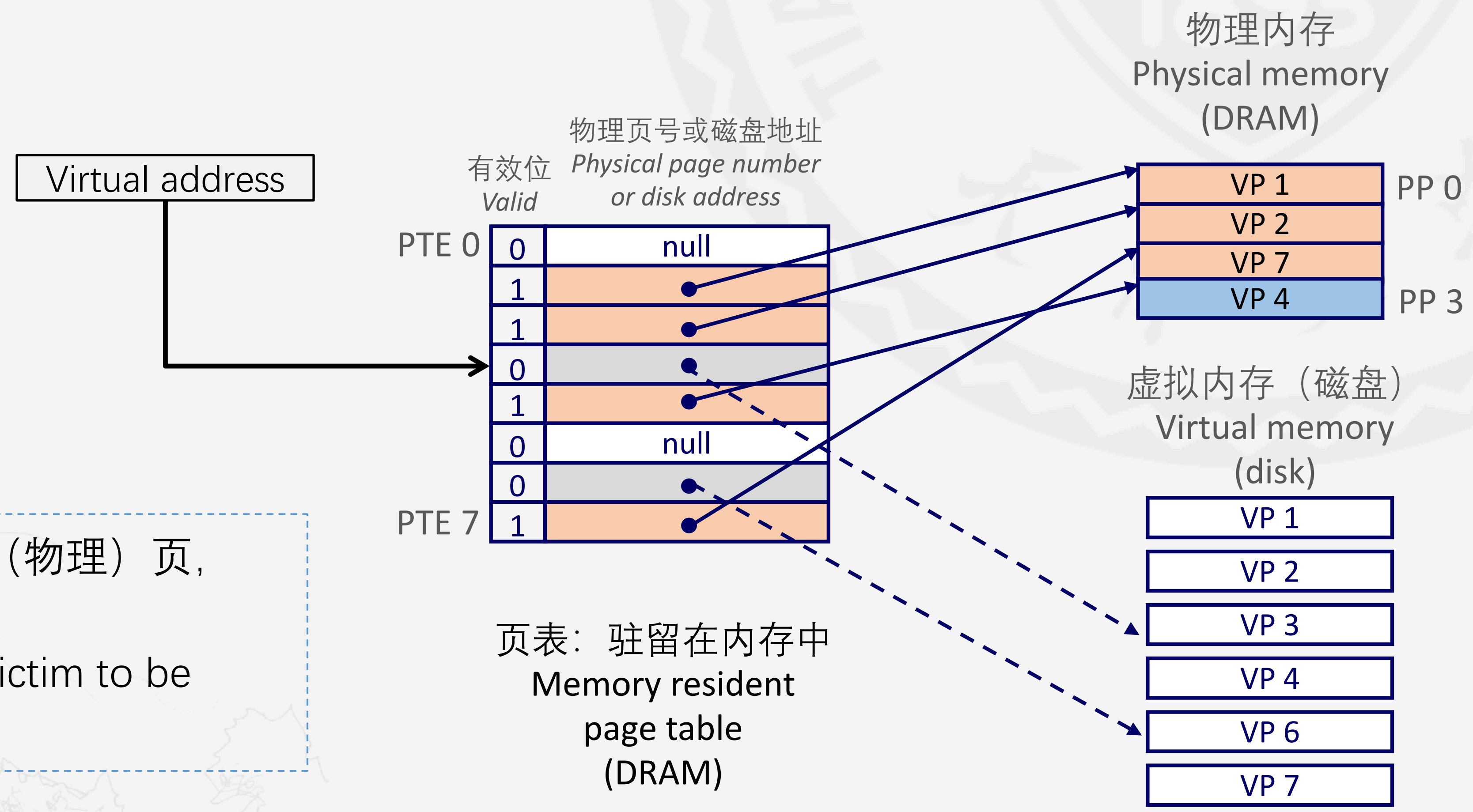


- 页未命中会引发**缺页故障异常**
  Page miss causes **page fault** (an exception)

虚拟内存作为缓存的工具

VM as a tool for caching

缺页处理
Handling Page Fault

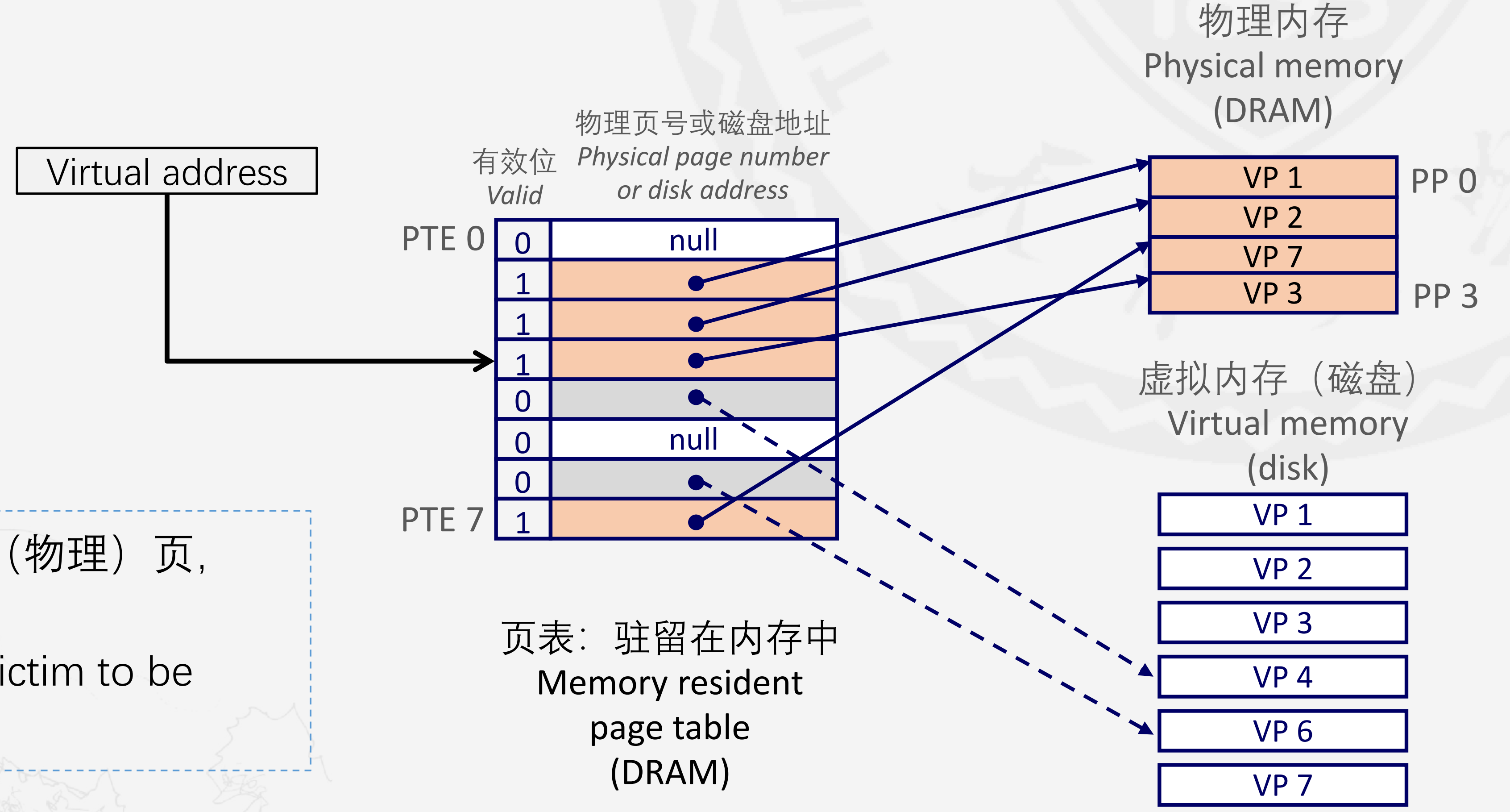


- 缺页故障处理程序选择一个（物理）页，换出内存　（这里是VP4）
  Page fault handler selects a victim to be evicted (here VP 4)

虚拟内存作为缓存的工具

VM as a tool for caching

缺页处理
Handling Page Fault

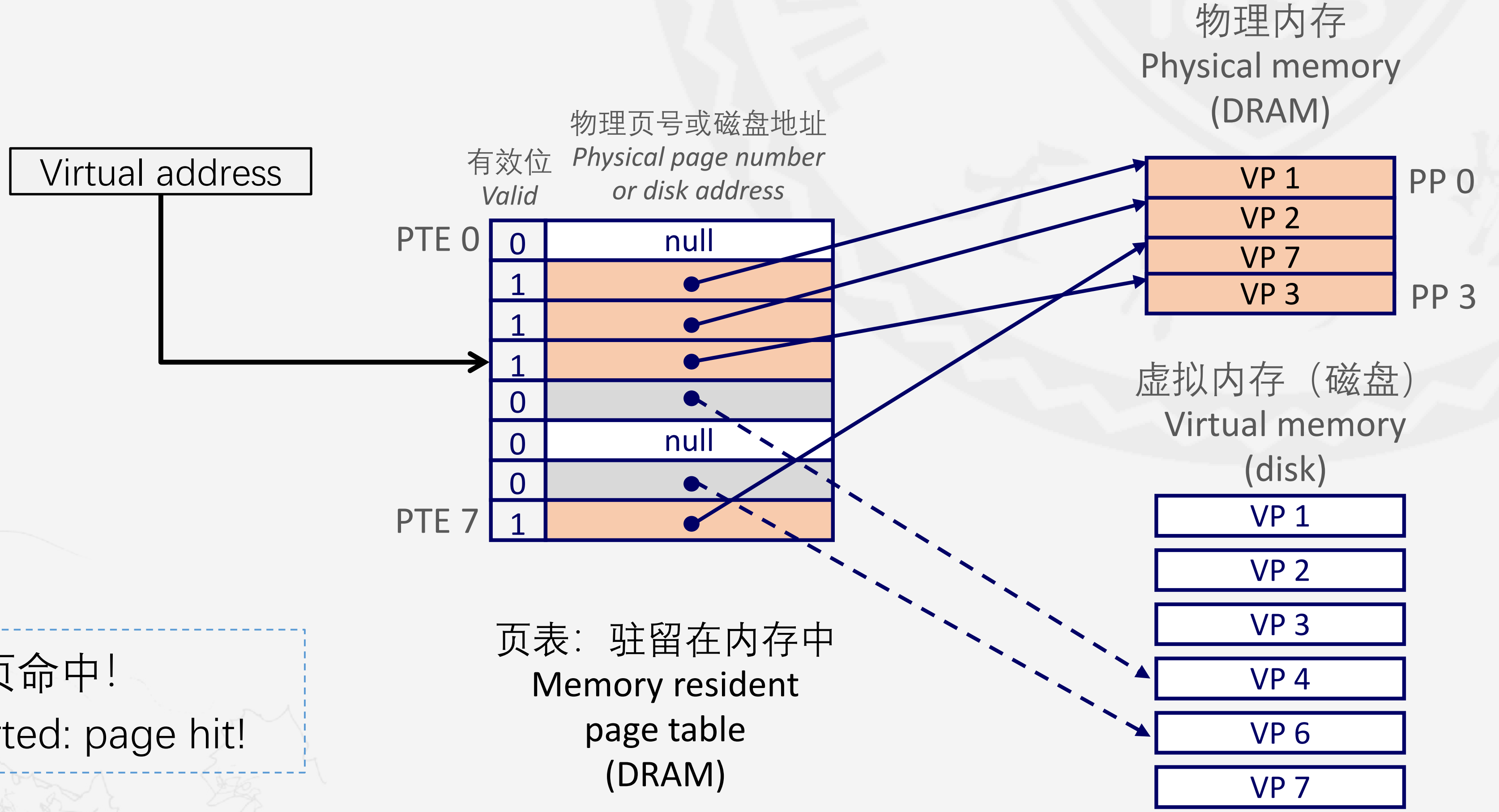


- 缺页故障处理程序选择一个（物理）页，换出内存　（这里是VP4）
  Page fault handler selects a victim to be evicted (here VP 4)

虚拟内存作为缓存的工具

VM as a tool for caching

缺页处理
Handling Page Fault



- 重新执行引发异常的指令：页命中！
  Offending instruction is restarted: page hit!

虚拟内存作为缓存的工具

VM as a tool for caching

又是局部性救了我们！
Locality to the Rescue Again!

- 虚拟内存机制能够有效工作是因为局部性原理
  Virtual memory works because of locality
- 在任意时刻，程序总是会频繁访问一个由活跃的虚拟页组成的集合，这个集合称为**工作集**
  At any point in time, programs tend to access a set of active virtual pages called the **working set**
  - 如果程序有很好的的时间局部性特征，那么它的工作集会更小
    Programs with better temporal locality will have smaller working sets
- 如果：工作集大小<主存容量
  If (working set size < main memory size)
  - 在一个进程经历了一系列必要的缺页后，会获得比较好的性能
    Good performance for one process after compulsory misses
- 如果：所有进程的工作集的总大小 > 主存容量
  If (SUM(working set sizes) > main memory size)
  - **抖动**：页面可能会出现连续的换入和换出，这会导致性能的下降
    Thrashing: Performance meltdown where pages are moved (copied) in and out continuously

本章内容

Topic

- 地址空间

  Address Spaces

- 虚拟内存作为缓存的工具

  VM as a tool for caching

- 虚拟内存作为内存管理的工具

  VM as a tool for memory management

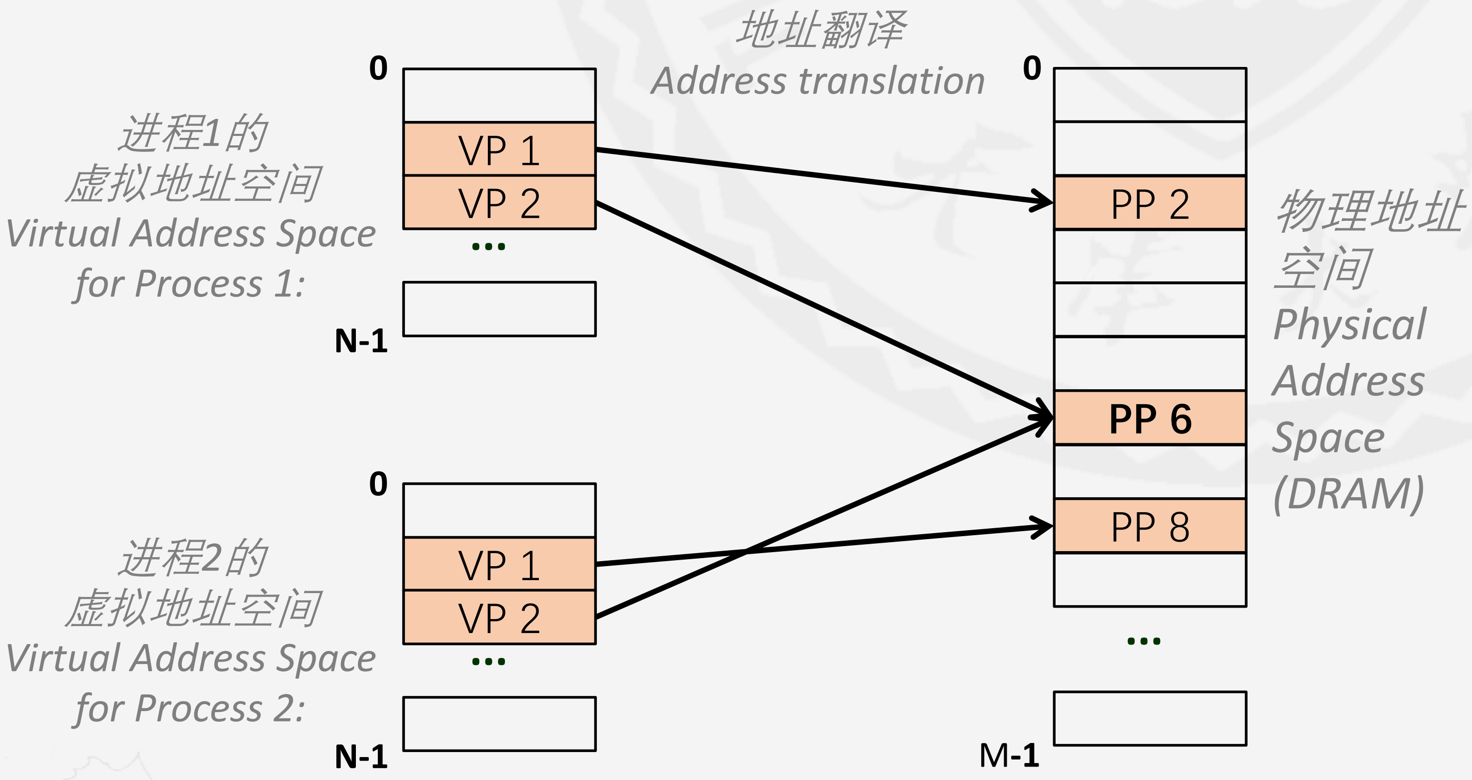
- 虚拟内存作为内存保护的工具

  VM as a tool for memory protection

虚拟内存作为内存管理的工具

VM as a tool for memory management

- **核心思想：**每个进程都拥有属于它们自己的虚拟地址空间
  Key idea: each process has its own virtual address space
  - 可以将内存看作是一个简单的、线性的数组
    It can view memory as a simple linear array
  - 映射函数可以将地址分散地映射到物理内存上（以页为基本单位）
    Mapping function scatters addresses through physical memory
    - 一种经过优秀设计的映射方法可以简化内存的分配和管理
      Well chosen mappings simplify memory allocation and management

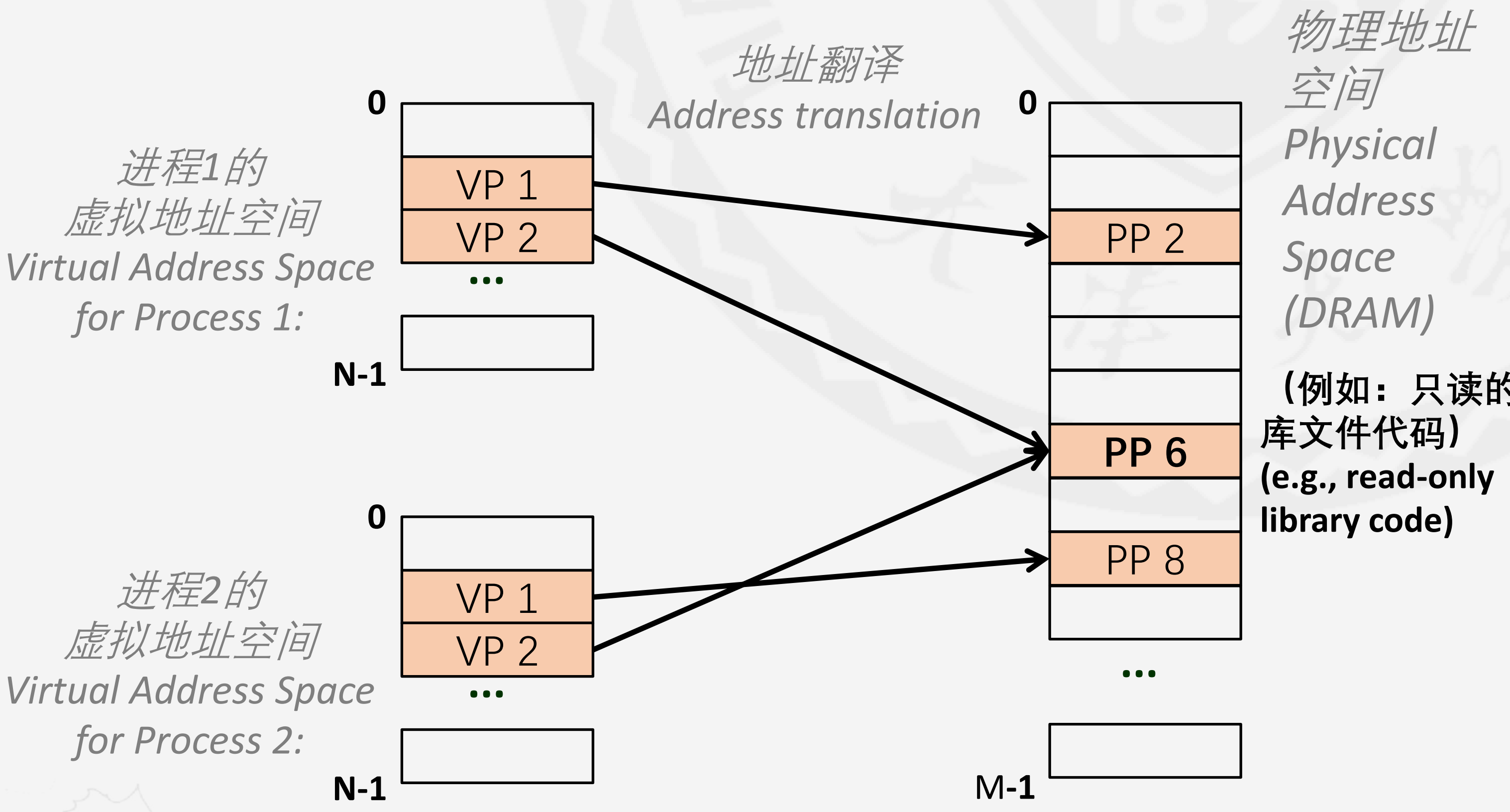


虚拟内存作为内存管理的工具

VM as a tool for memory management

简化分配和共享
Simplifying allocation and sharing

- **内存分配**
  Memory allocation
  - 每一个虚拟页可以映射到任意的物理页
    Each virtual page can be mapped to any physical page
  - 一个虚拟页，可能在不同的时间点，被存储在不同的物理页中
    A virtual page can be stored in different physical pages at different times
- **共享进程间的代码和数据**
  Sharing code and data among processes
  - 多个虚拟页映射到同一个物理页中（PP6）
    Map multiple virtual pages to the same physical page (here: PP 6)



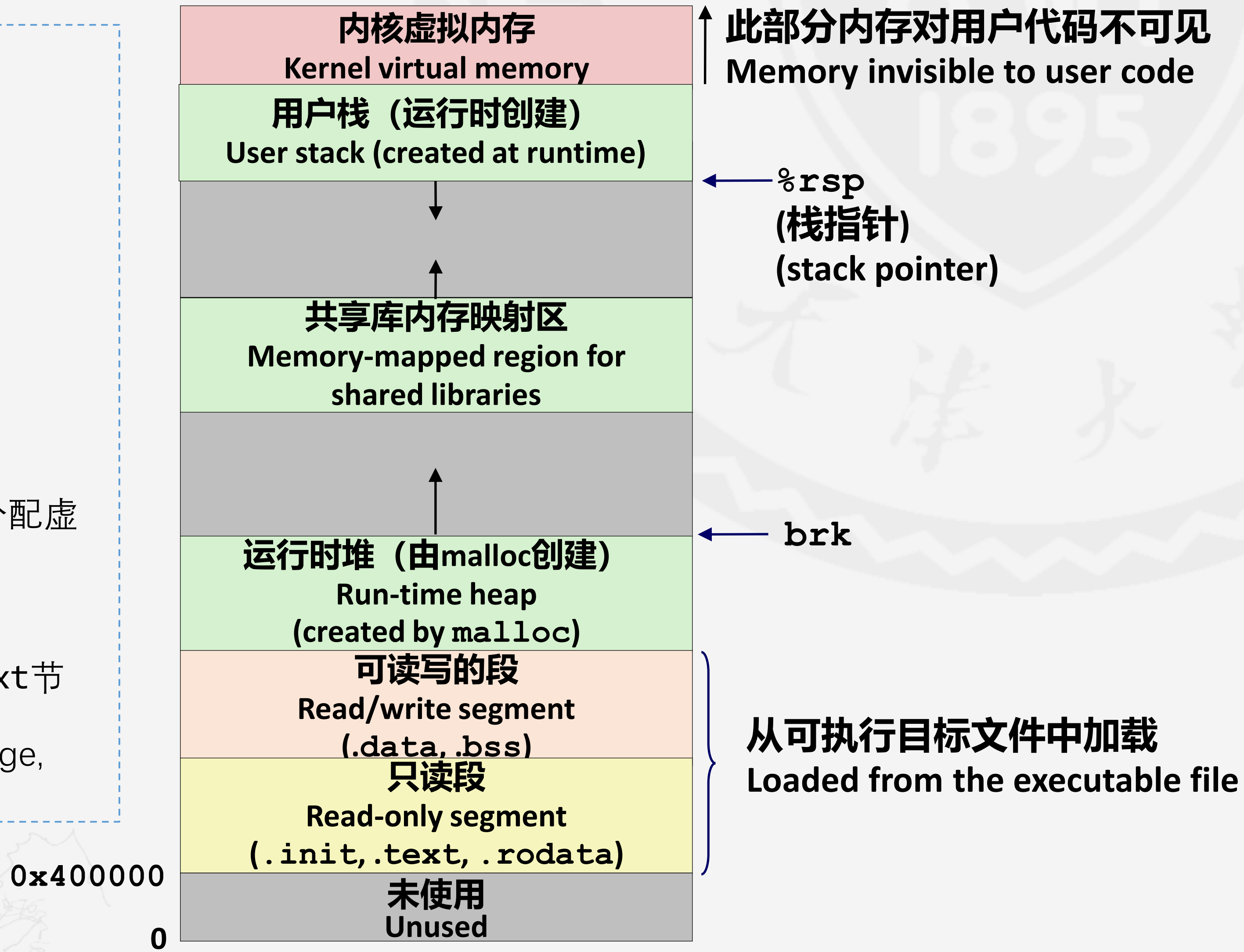
虚拟内存作为内存管理的工具

VM as a tool for memory management

简化链接和加载
Simplifying Linking and Loading

- **链接**
  Linking
  - 每一个进程都具有相似的虚拟地址空间
    Each program has similar virtual address space
  - 代码、栈和共享库总是从相同的位置开始
    Code, stack, and shared libraries always start at the same address

- **加载**
  Loading
  - `execve` （Linux内核加载器）为`.text`和`.data`分配虚拟页（创建了页表项并标记该项为无效）
    execve() allocates virtual pages for .text and .data sections = creates PTEs marked as invalid
  - 当虚拟内存系统需要使用它们的时候，对应的`.text`节和`.data`节才会被以页为单位复制到内存中
    The .text and .data sections are copied, page by page, on demand by the virtual memory system



本章内容

Topic

- 地址空间

  Address Spaces

- 虚拟内存作为缓存的工具

  VM as a tool for caching

- 虚拟内存作为内存管理的工具

  VM as a tool for memory management

- 虚拟内存作为内存保护的工具

  VM as a tool for memory protection

虚拟内存作为内存保护的工具

VM as a tool for memory protection

- 页表项中增加了权限位
  Extend PTEs with permission bits
- 在每一次内存访问时MMU都会检查这些权限位
  MMU checks these bits on each access

进程 i 页表
Page table for process i:

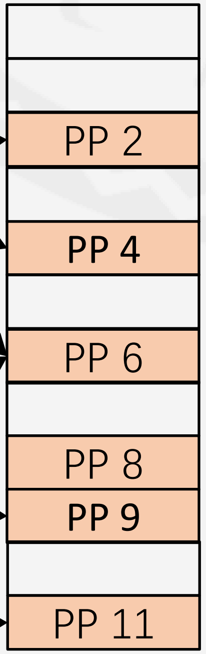
	SUP	READ	WRITE	EXEC	Address
VP 0:	No	Yes	No	Yes	PP 6
VP 1:	No	Yes	Yes	Yes	PP 4
VP 2:	Yes	Yes	Yes	No	PP 2

⋮

进程 j 页表
Page table for process j:

	SUP	READ	WRITE	EXEC	Address
VP 0:	No	Yes	No	Yes	PP 9
VP 1:	Yes	Yes	Yes	Yes	PP 6
VP 2:	No	Yes	Yes	Yes	PP 11

物理地址空间
Physical Address Space



小结
Summary

- **程序员视角中的虚拟内存**
  Programmer’s view of virtual memory
  - 每个进程有属于它们自己的、私有的、线性地址空间
    Each process has its own private linear address space
  - （这个地址空间）不能被其他进程所破坏
    Cannot be corrupted by other processes
- **系统视角中的虚拟内存**
  System view of virtual memory
  - 通过缓存虚拟页，可以有效地提高内存的使用效率 （局部性原理）
    Uses memory efficiently by caching virtual memory pages (locality)
  - 简化了内存管理和编程
    Simplifies memory management and programming
  - 提供了一个方便的插入点实现访问权限的检查，以简化内存保护机制
    Simplifies protection by providing a convenient interpositioning point to check permissions