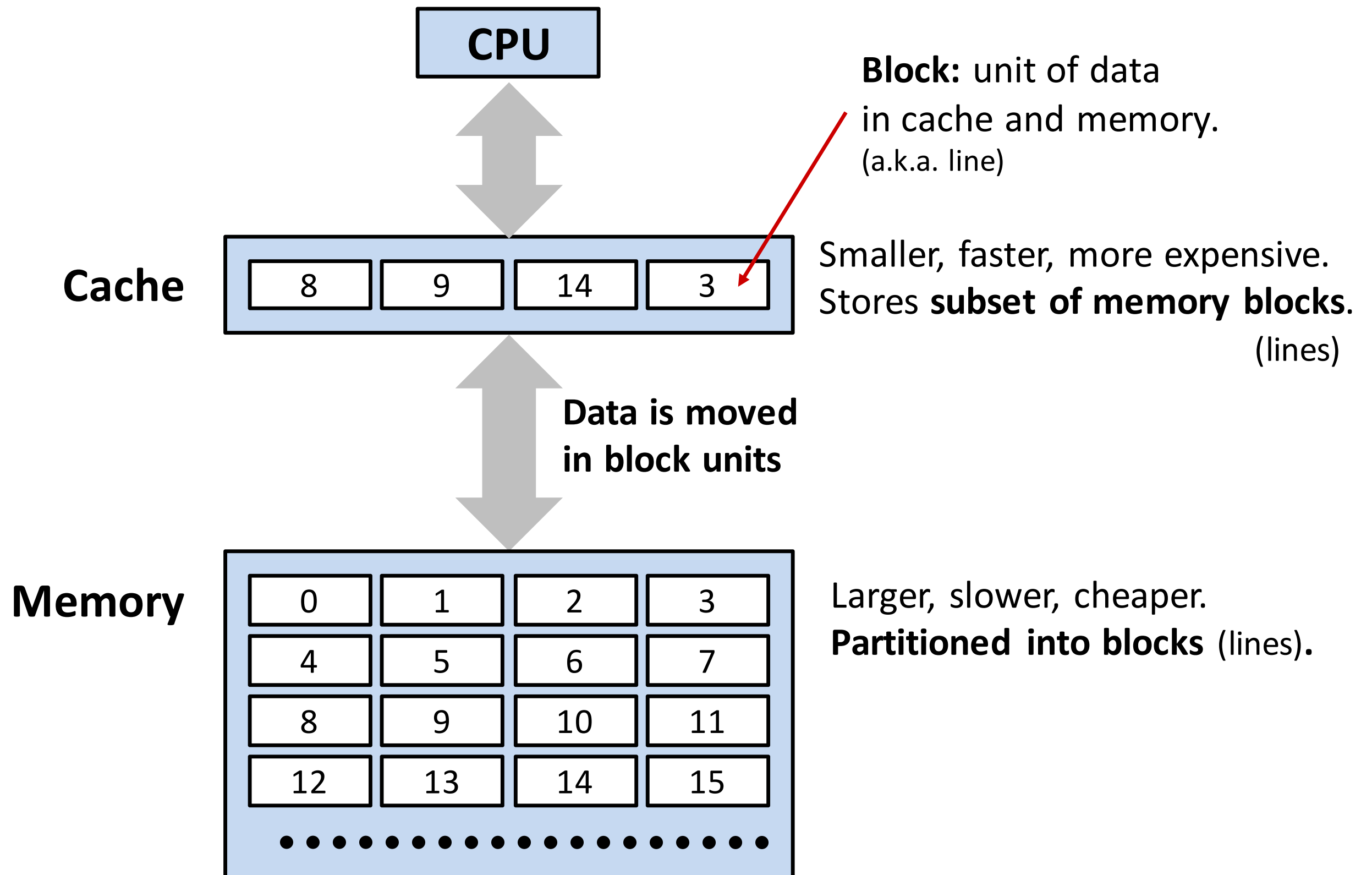


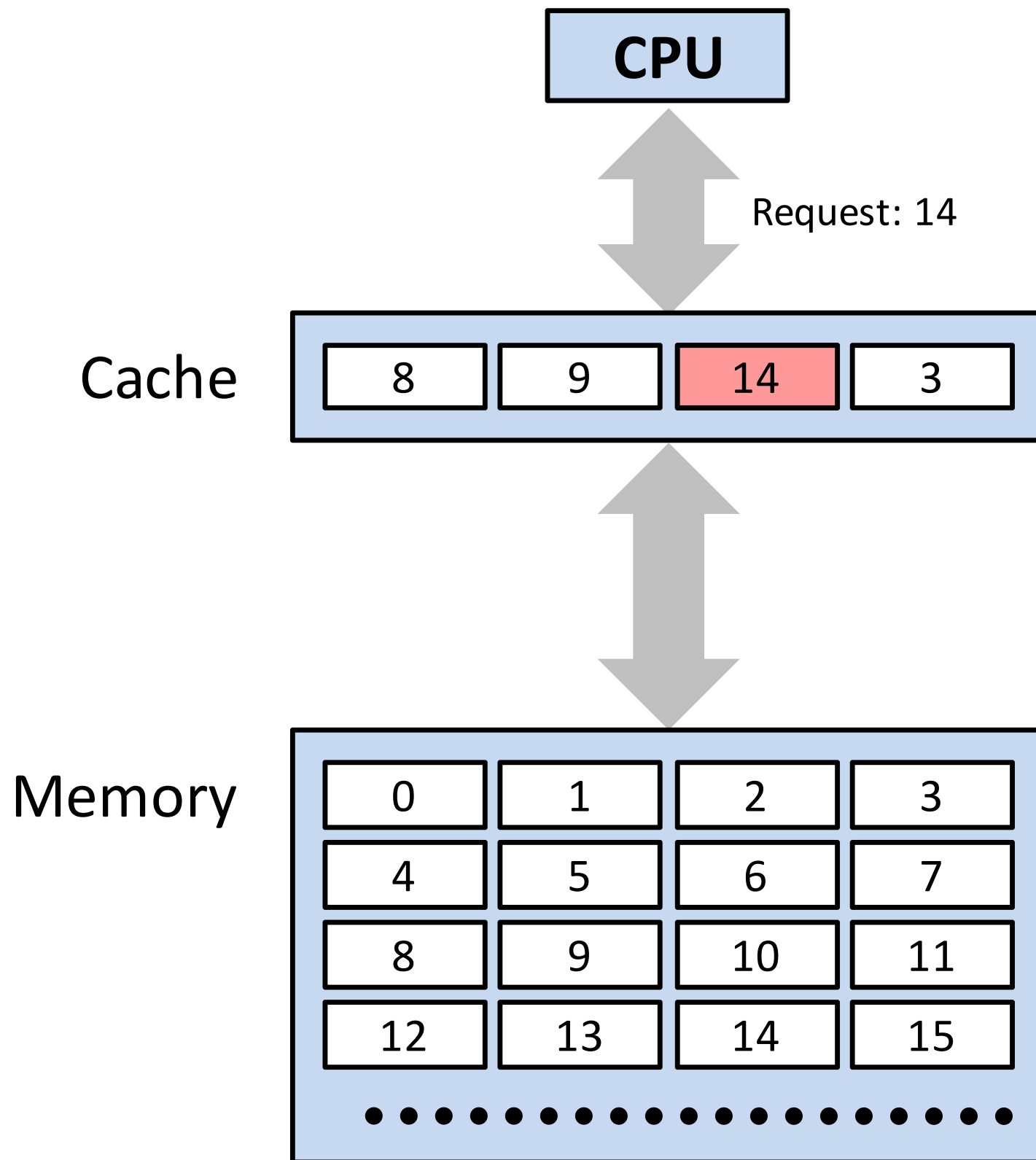
存储器层次结构

Part3: 虚拟存储器

Cache原理



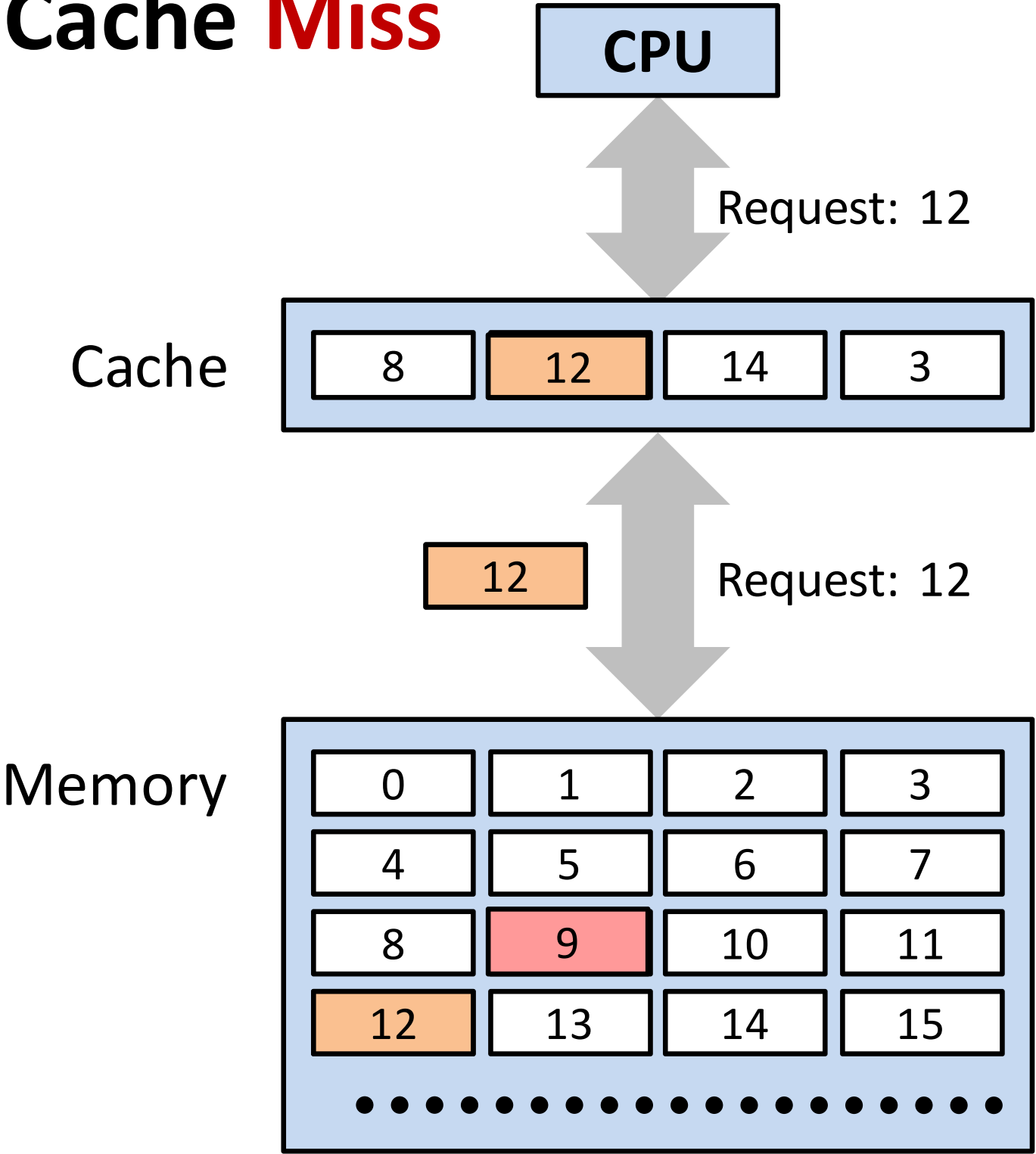
Cache Hit



1. Request data in block *b*.

2. *Cache hit:*
*Block *b* is in cache.*

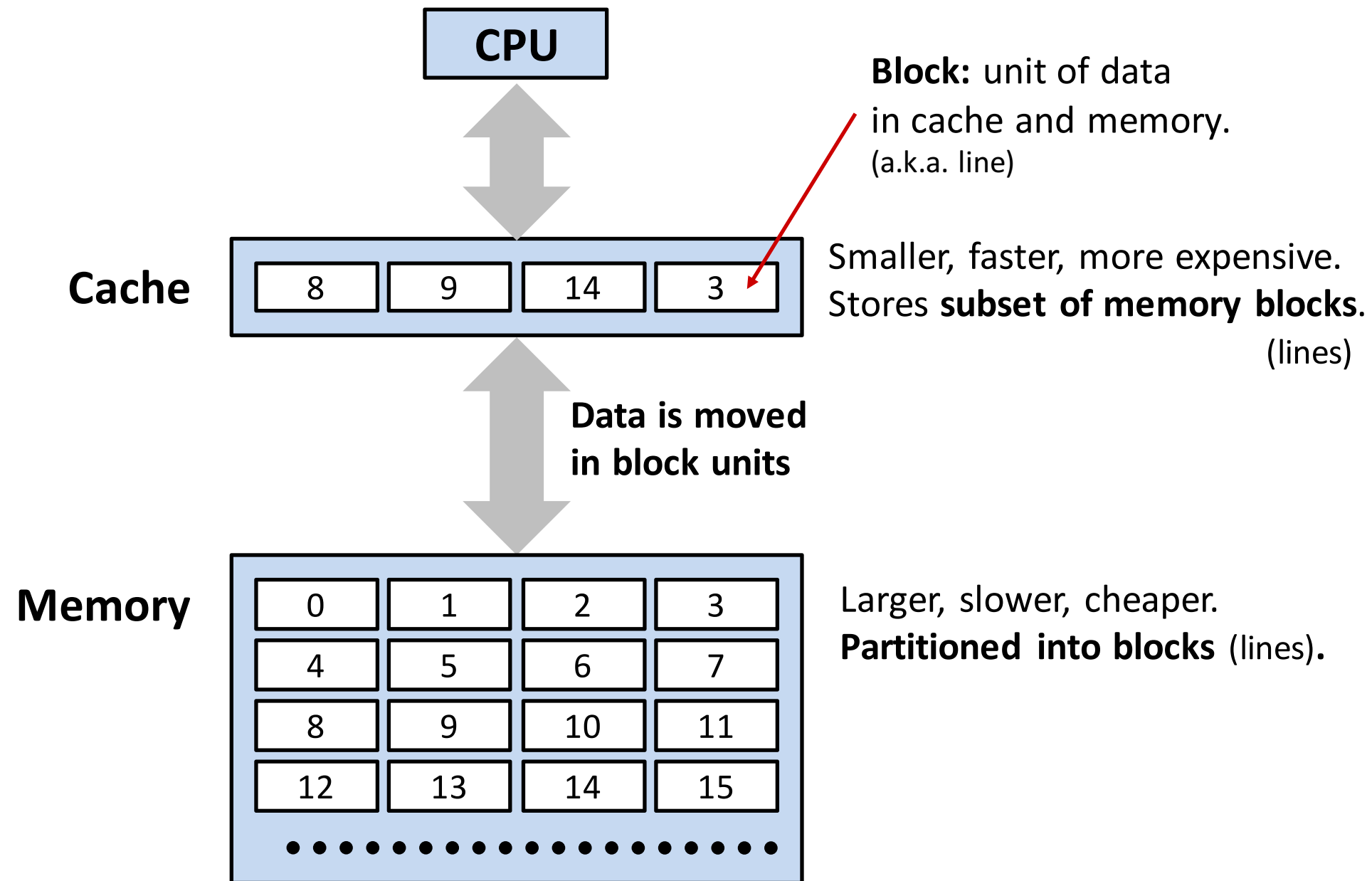
Cache Miss



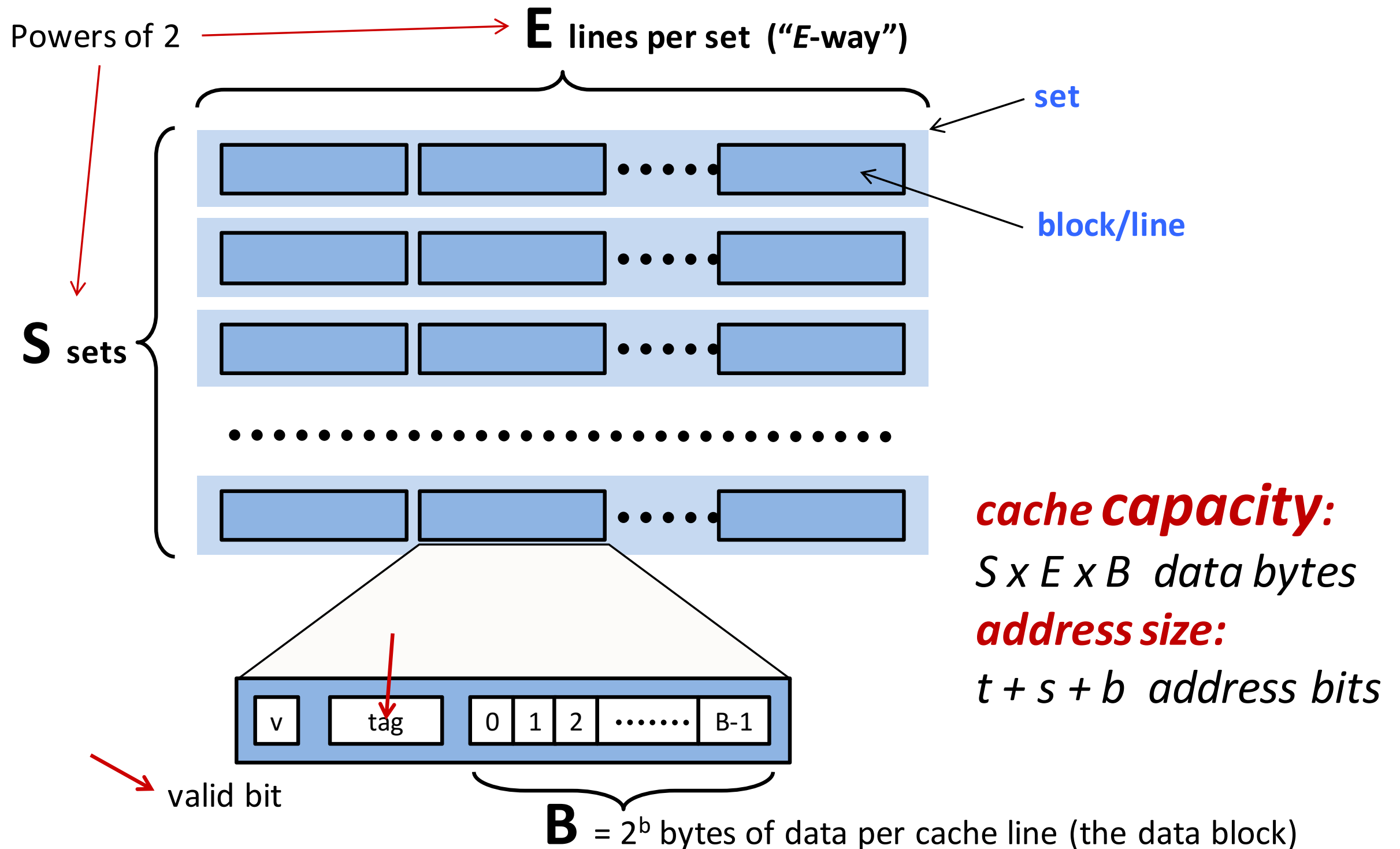
- 1. *Request data in block **b**.*
- 2. **Cache miss:**
*block is **not** in cache*
- 3. **Cache eviction:**
*Evict a block to make room,
maybe store to memory.*
- 4. **Cache fill:**
*Fetch block from memory,
store in cache.*

Placement Policy:
where to put block in cache

Replacement Policy:
which block to evict



一般化的Cache组织形式

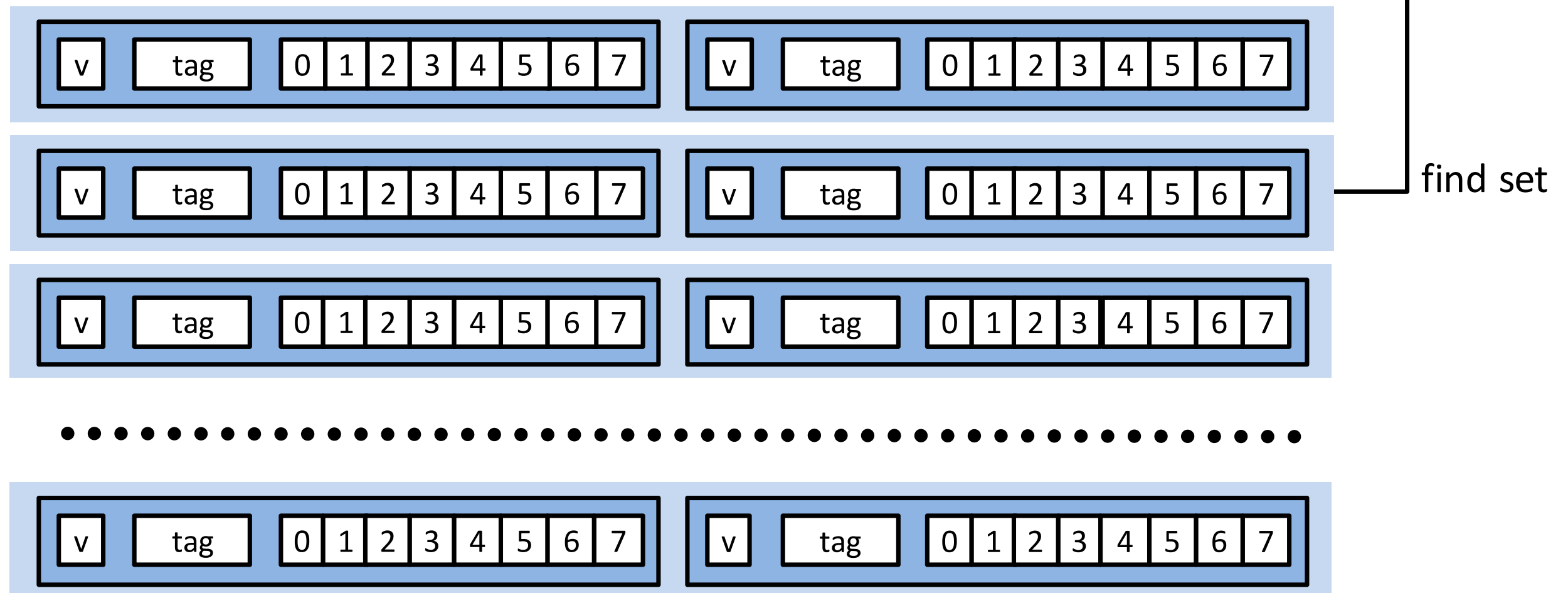
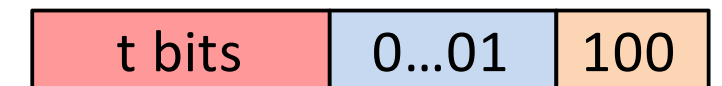


读Cache：组相联映射

This cache:

- Block size: 8 bytes
- Associativity: 2 blocks per set

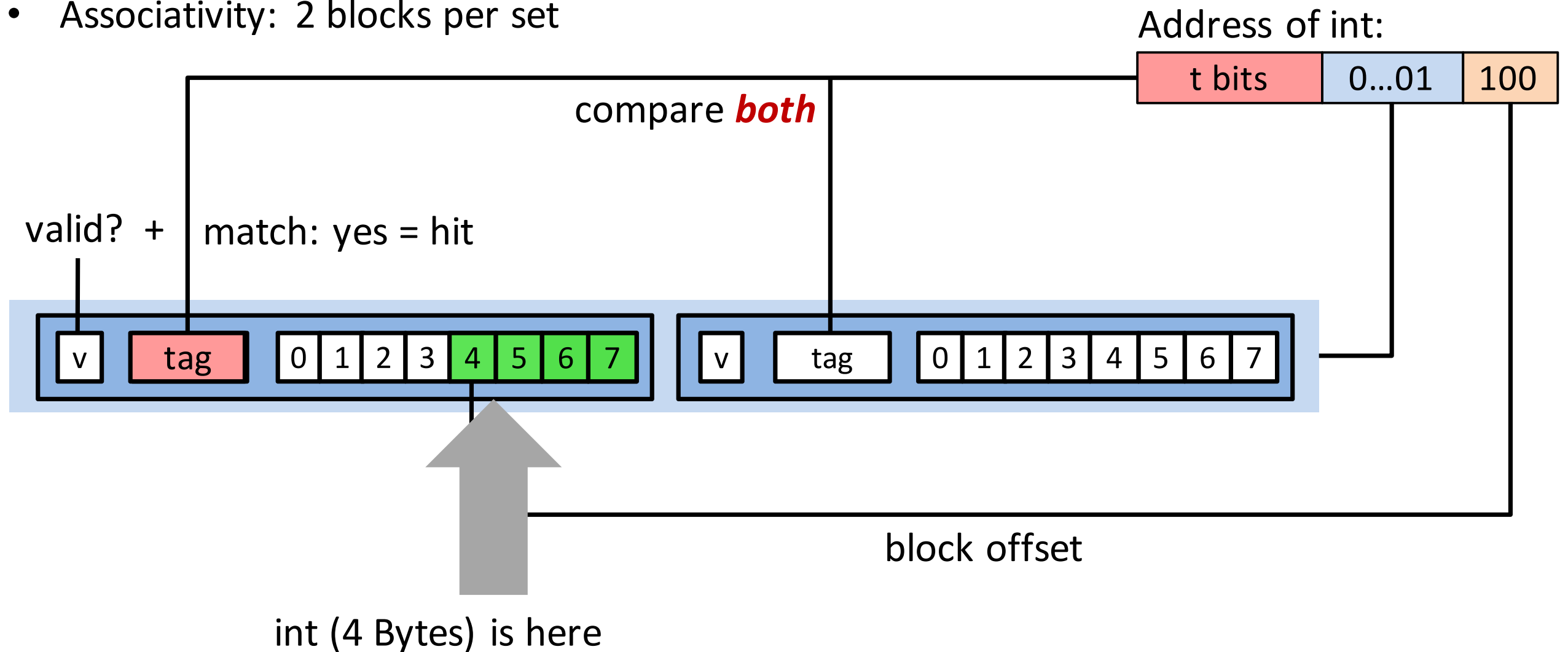
Address of int:



读Cache：组相联映射

This cache:

- Block size: 8 bytes
- Associativity: 2 blocks per set



本讲提纲

■ 虚拟存储器

- 虚拟存储器基本概念
- 虚拟地址到物理地址的转换
- 虚拟存储器的页式管理

存储容量需求

■ 应用需求

- 海量数据处理

- 天气预报、地震预测、石油勘探、搜索...

- 多媒体信息处理

- 语音、图形、图象...

■ 软件需求

- Nathan软件第一定律:软件是一种可以膨胀到充满整个容器的气体。

■ 技术需求

- 多进程、多道程序

■ 给程序员一个比实际内存大得多的可管理的编址空间

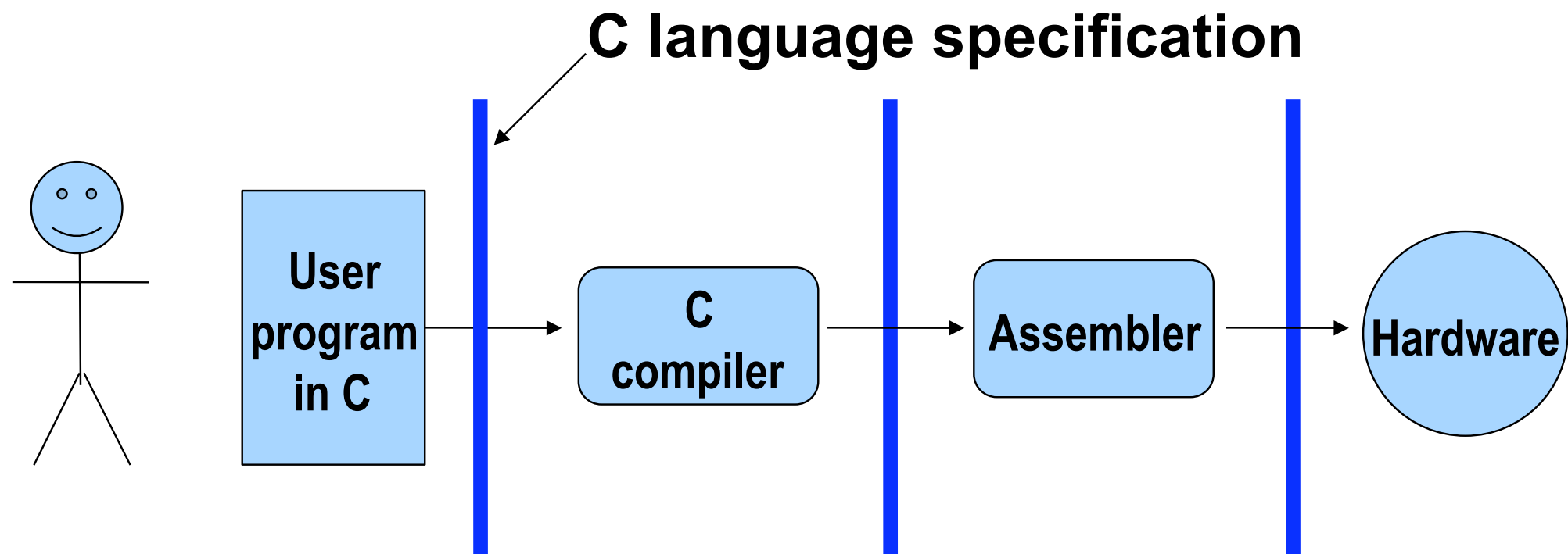
提高存储器容量

- 降低主存储器成本,在同样成本下,可以获得更大的主存容量
 - 主存的价格到今天也依然比较昂贵
 - 程序对主存的“胃口”的增加和主存价格的降低速度几乎同样的快
- 采用虚拟存储器
 - 只在确实需要的时候才把程序和数据装入到主存中

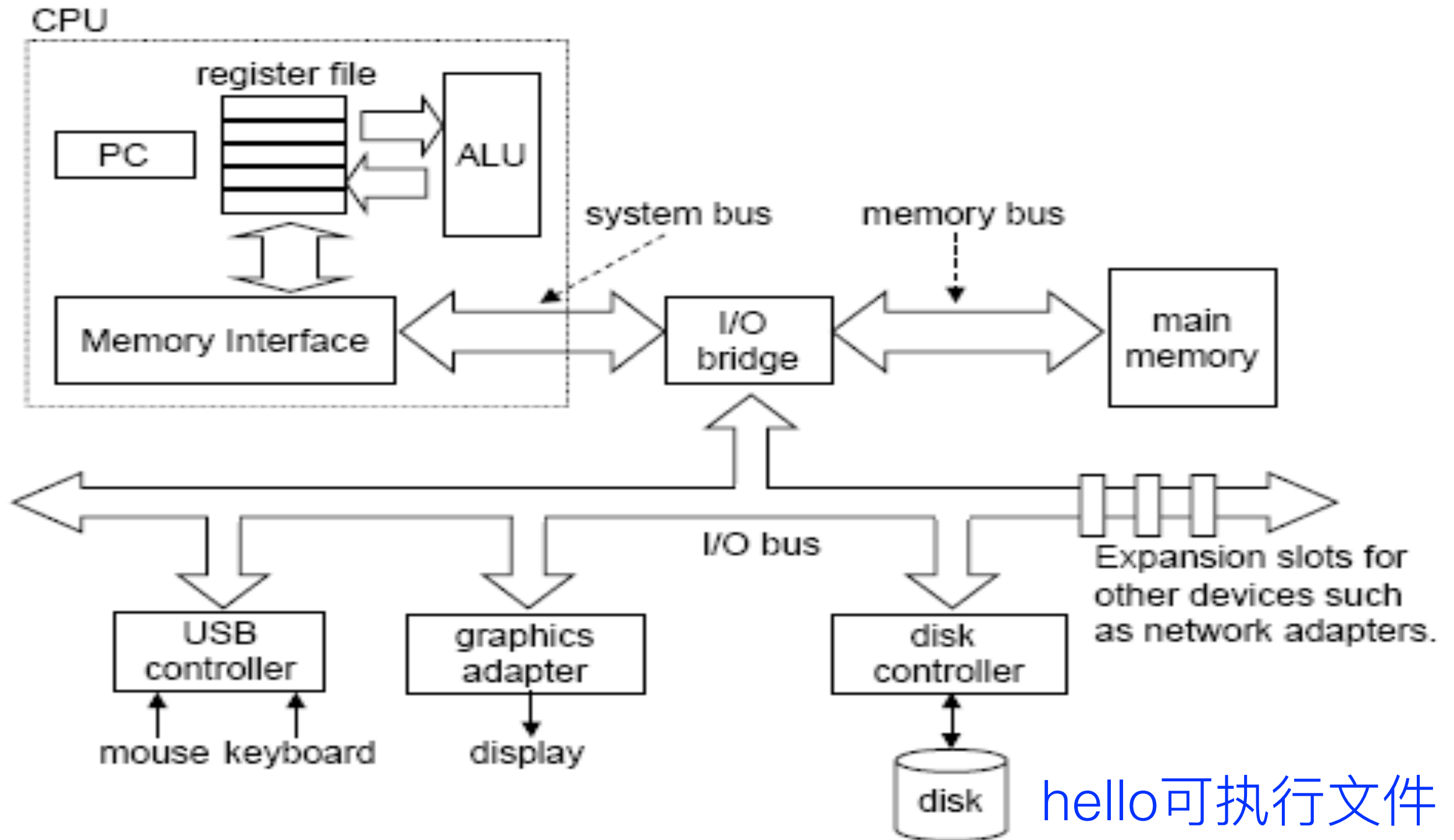
一个典型程序的转换处理过程

■ 经典的C-源程序

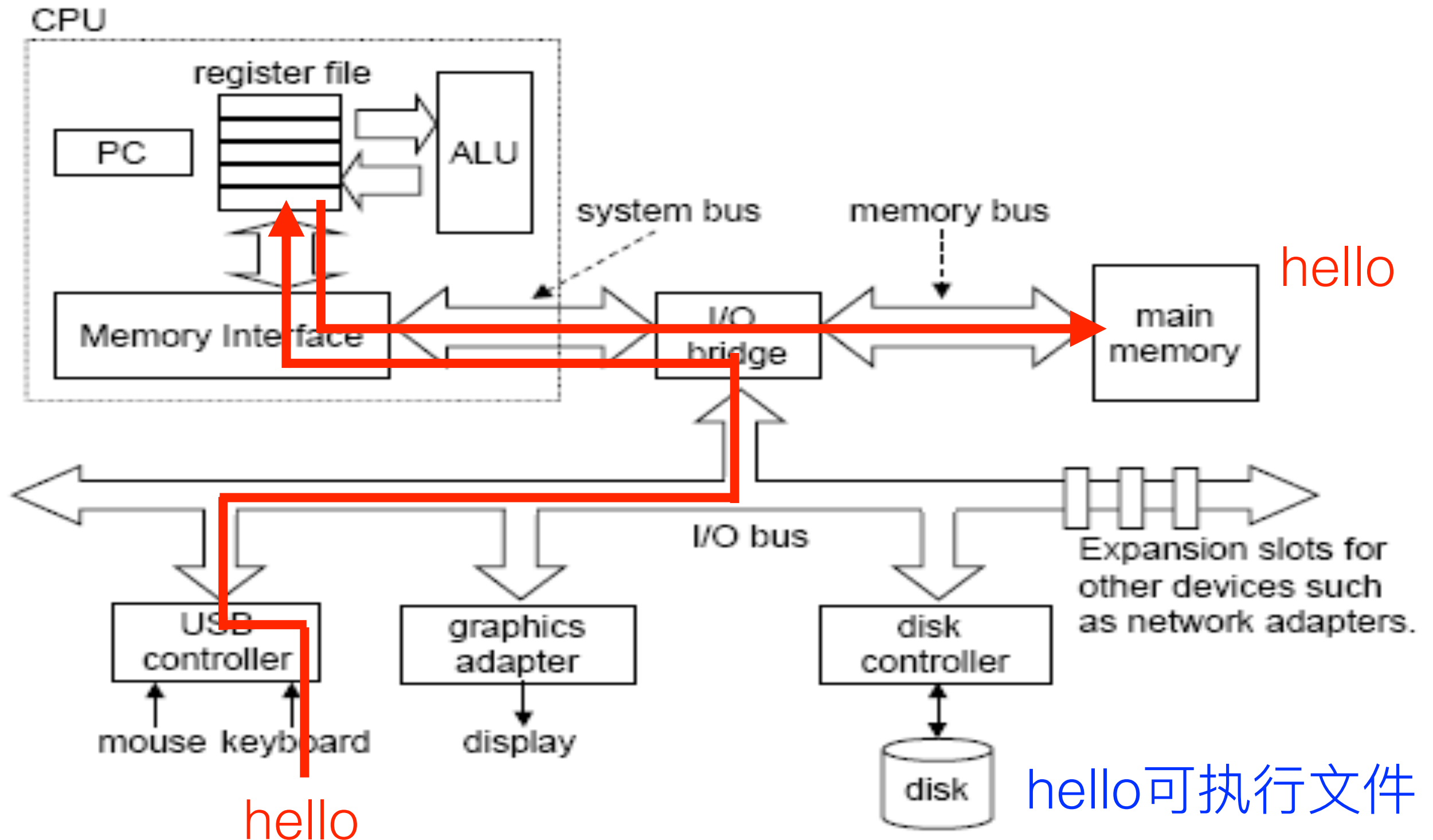
```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("hello, world\n");
6 }
```



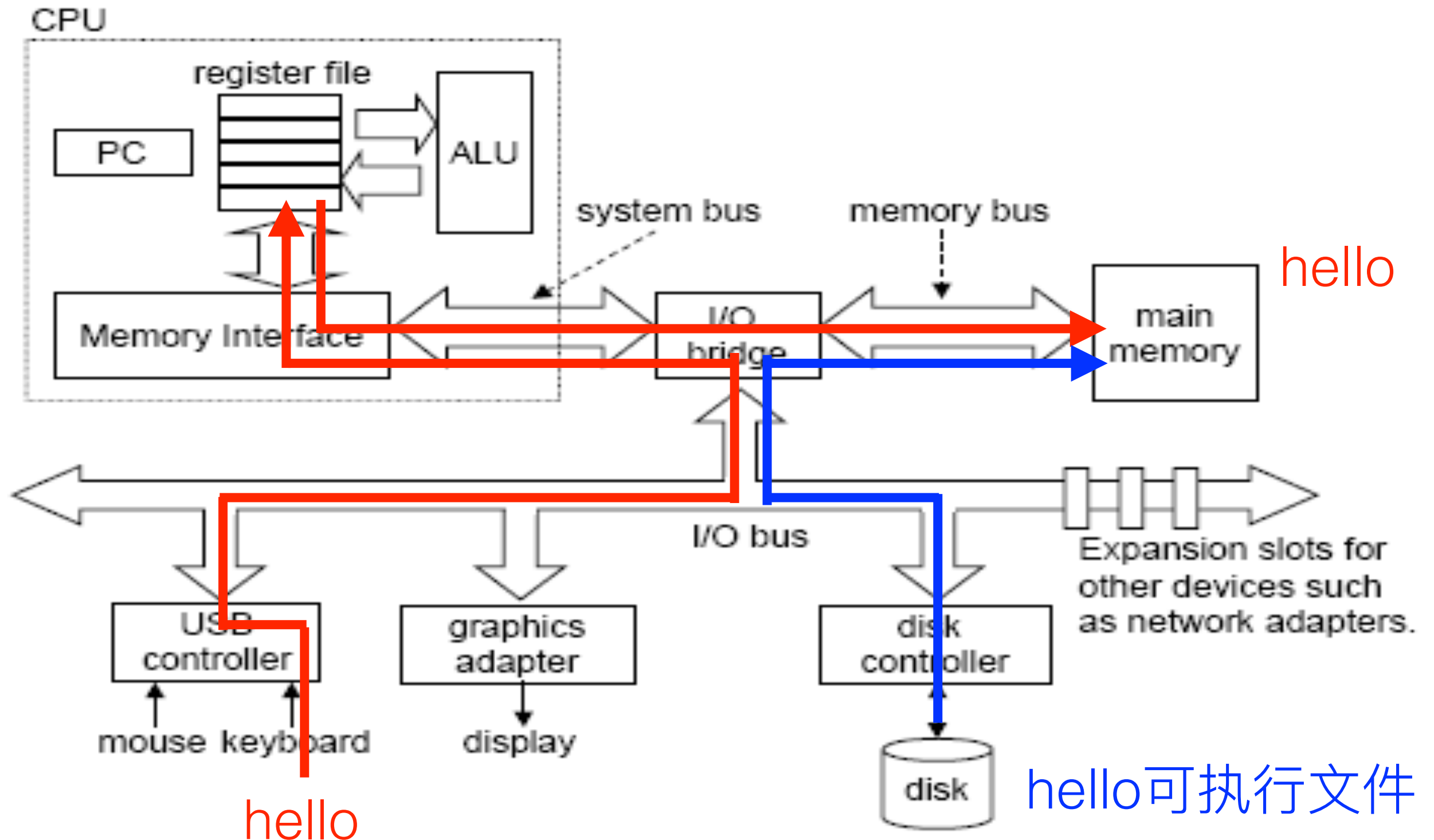
Hello程序的数据流动过程



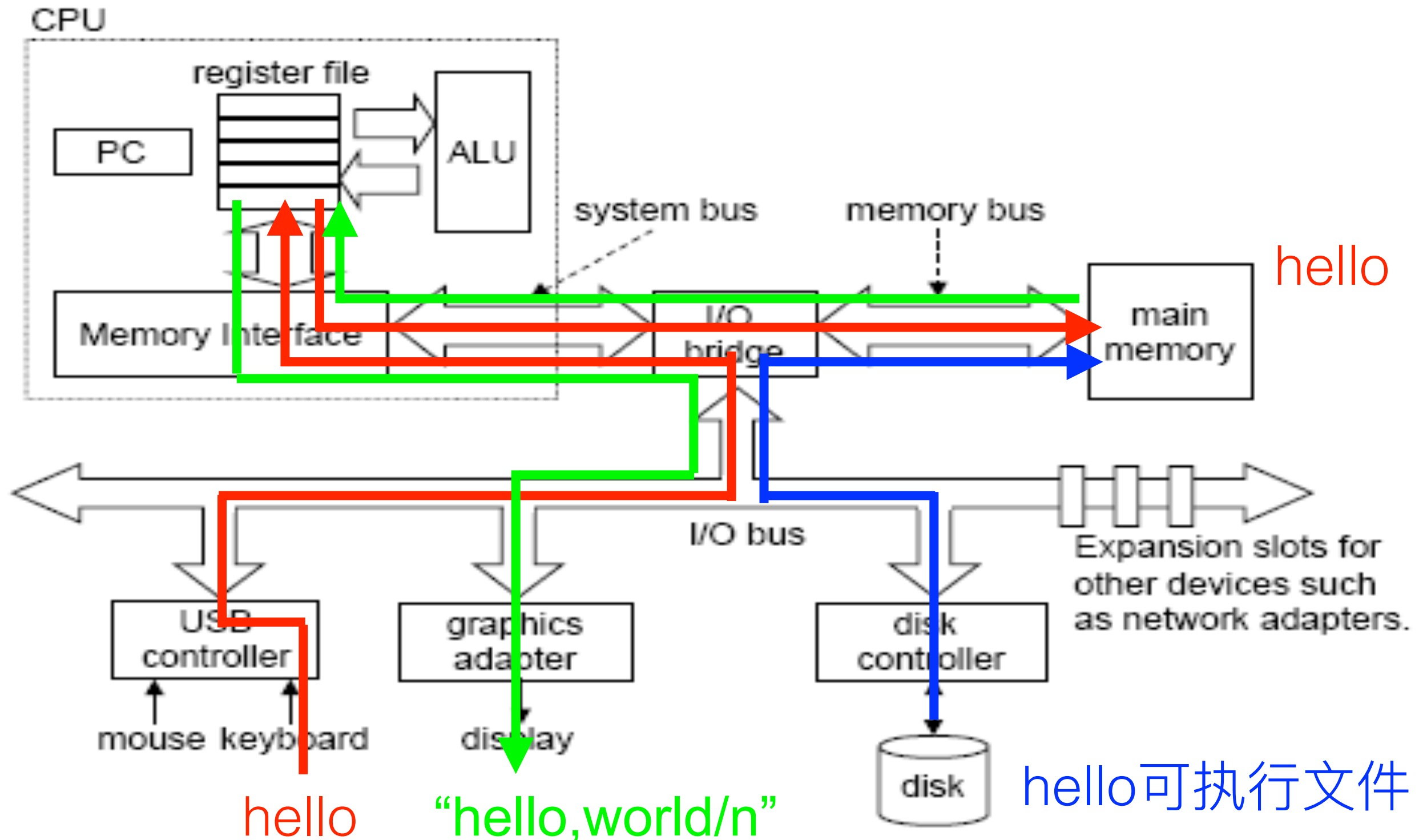
Hello程序的数据流动过程



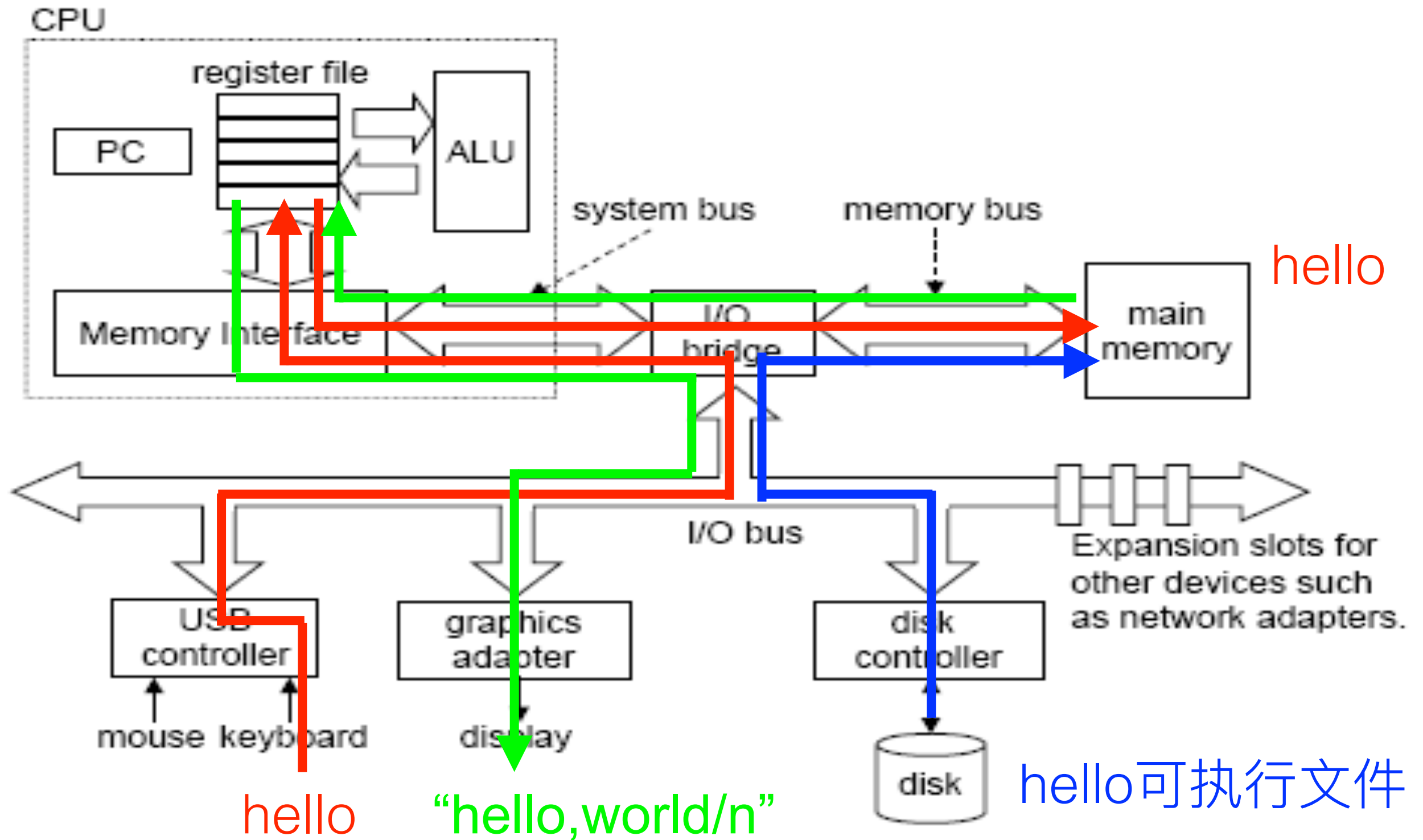
Hello程序的数据流动过程



Hello程序的数据流动过程



问题：hello程序何时被装？谁来装入？被谁启动？每次是否被装到相同的地方？Hello程序是否知道还有其他程序同时运行？是否能直接访问硬件资源？



操作系统在程序执行过程中的作用

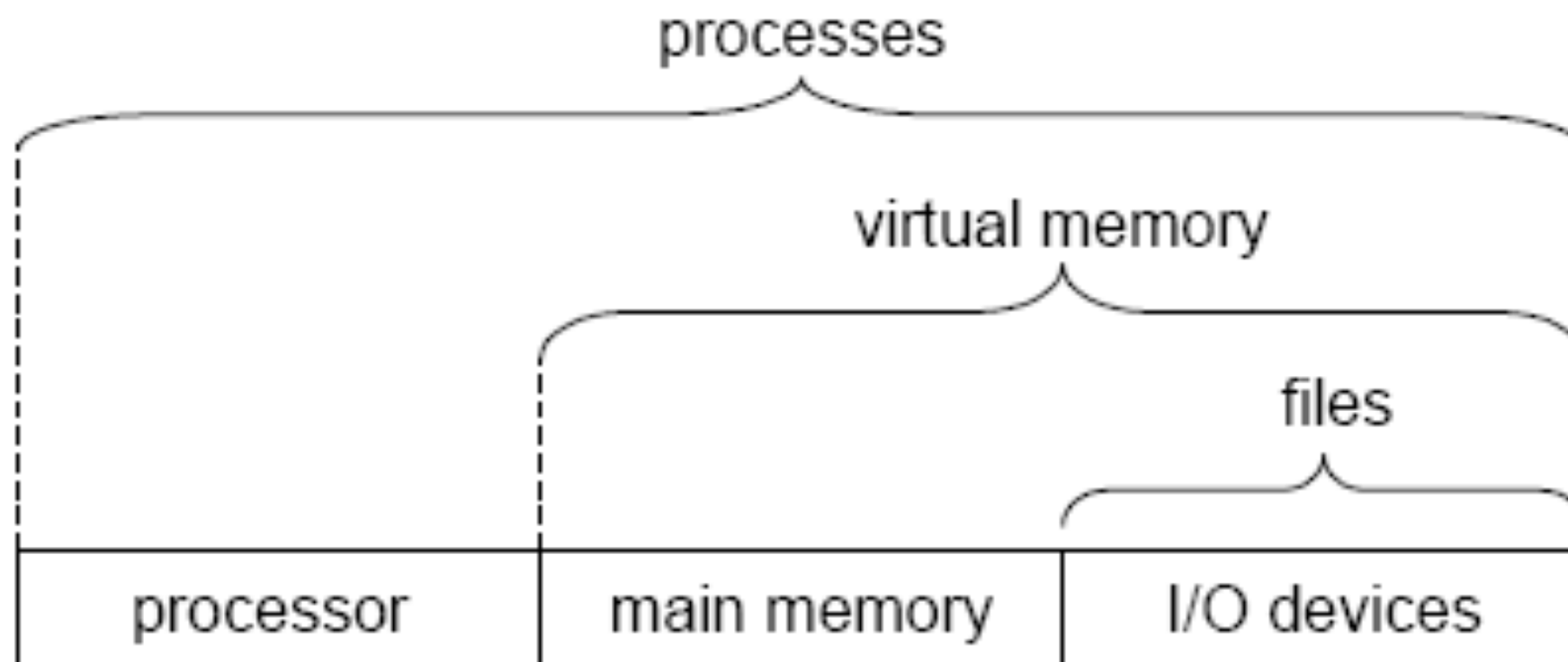
- 在Hello程序执行过程中，Hello程序本身没有直接访问键盘、显示器、磁盘和主存储器这些硬件资源，而是依靠**操作系统**提供的服务来间接访问。
- 操作系统是在应用程序和硬件之间插入的一个中间软件层。
- 操作系统的两个主要的作用：
 - **硬件资源管理**，以达到以下两个目的：
 - 统筹安排和调度硬件资源，以防止硬件资源被用户程序滥用
 - 对于广泛使用的复杂低级设备，为用户程序提供一个简单一致的使用接口
 - **为用户使用系统提供一个操作接口**

存储器管理

- 早期采用单道程序，系统的主存中包含：
 - 操作系统（常驻监控程序）
 - 正在执行的一个用户程序
 - 所以无需进行存储管理，即使有也很简单。
- 现在都采用多道程序，系统的存储器中包含：
 - 操作系统
 - 若干个用户程序
 - 如果在存储器中进程数很少，则由于进程花费很多时间来等待I/O，常使处理机处于空闲状态。因此，存储器需要进行合理分配，尽可能让更多进程进入存储器。
- 在多道程序系统中，存储器的“用户”部分须进一步划分以适应多个进程。划分的任务由OS动态执行，这被称为**存储器管理**(memory management)。

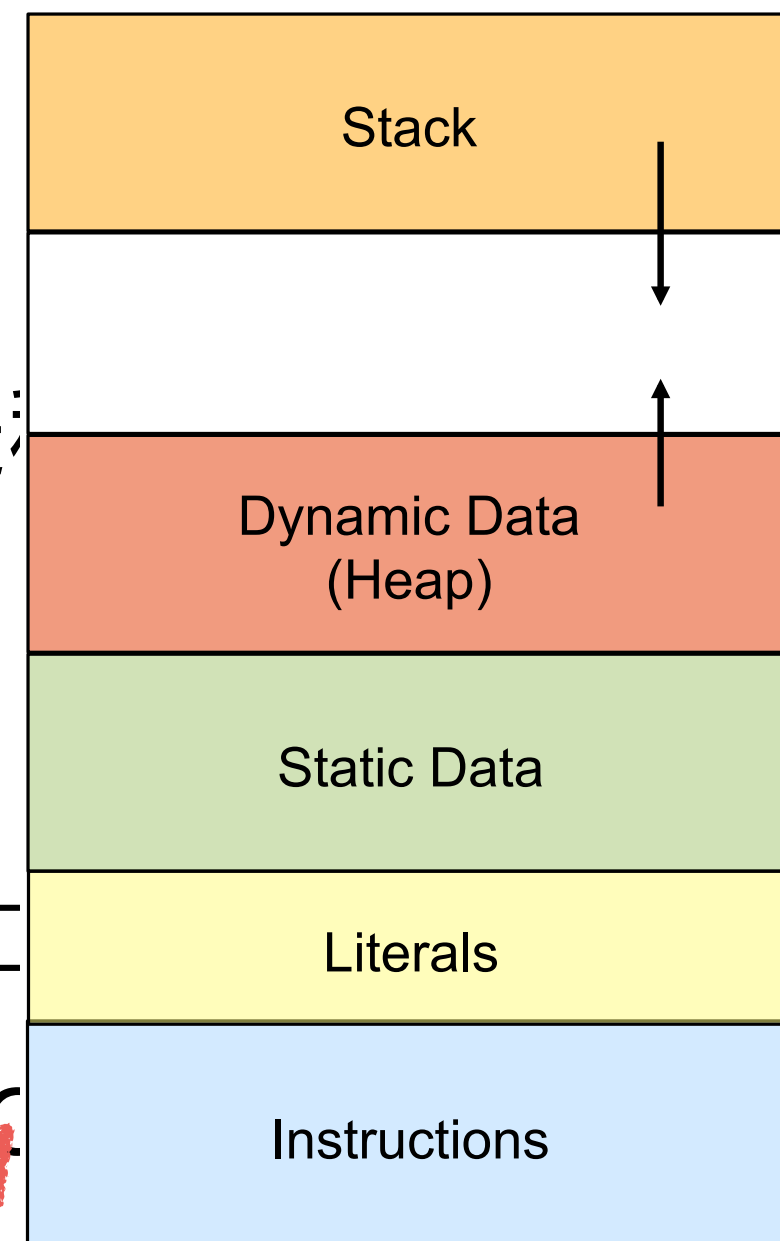
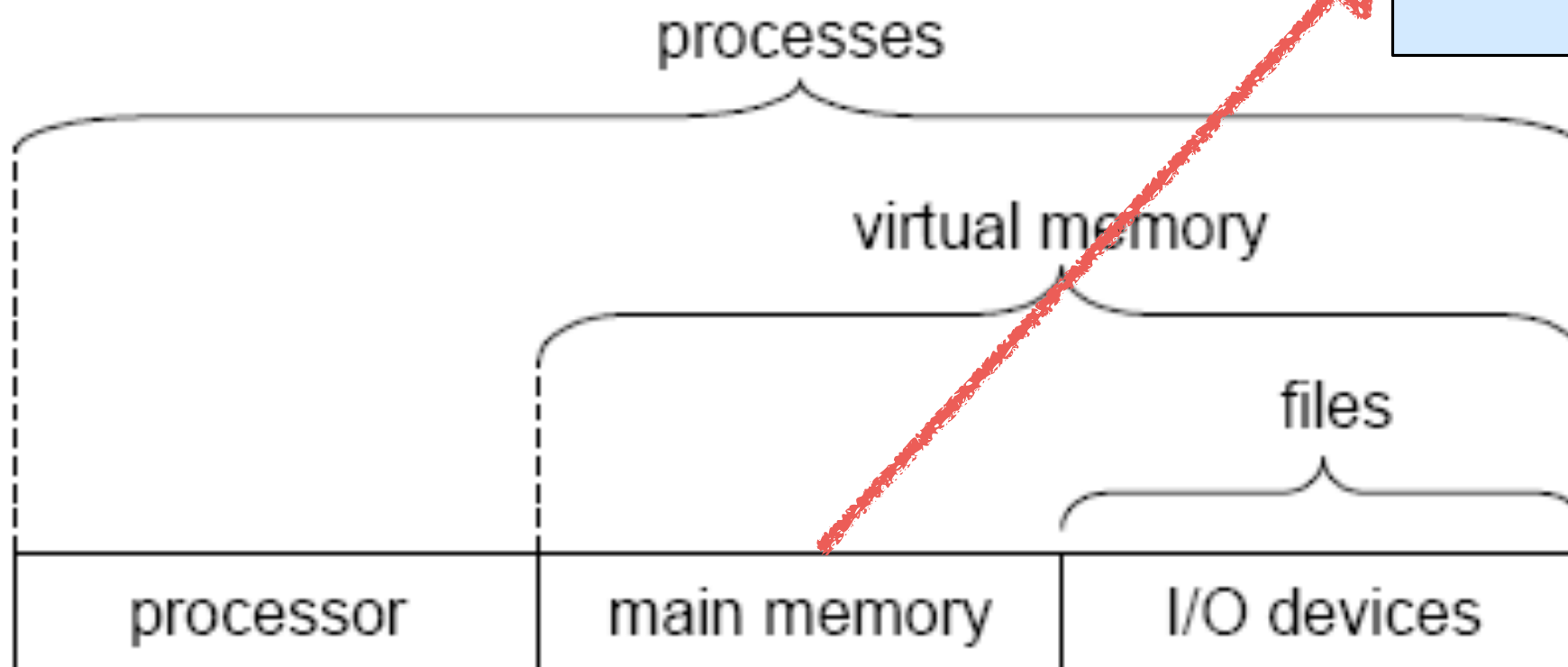
进程

- 操作系统通过三个基本的抽象概念（进程、虚拟存储器、文件）实现硬件资源管理
 - 文件（files）是对I/O设备的抽象
 - 虚拟存储器（Virtual Memory）是对主存和磁盘I/O的抽象
 - 进程（processes）是对处理器、主存和I/O设备的抽象



进程

- 操作系统通过三个基本的抽象概念（进程、设备、文件）实现硬件资源管理
 - 文件（files）是对I/O设备的抽象
 - 虚拟存储器（Virtual Memory）是对主存和I/O设备的抽象
 - 进程（processes）是对处理器、主存和I/O设备的抽象



进程

- Hello程序运行时，Hello程序会以为（错觉）：
 - 所有系统资源都被自己独占使用
 - 处理器始终在执行本程序的一条条指令
- 进程是操作系统对运行程序的一种抽象
 - 一个系统可同时运行很多进程，但每个进程都好像自己独占使用系统
 - 实际上，操作系统让处理器交替执行很多进程中的指令
 - 操作系统实现交替指令执行的机制称为“上下文切换 (context switching)”

进程

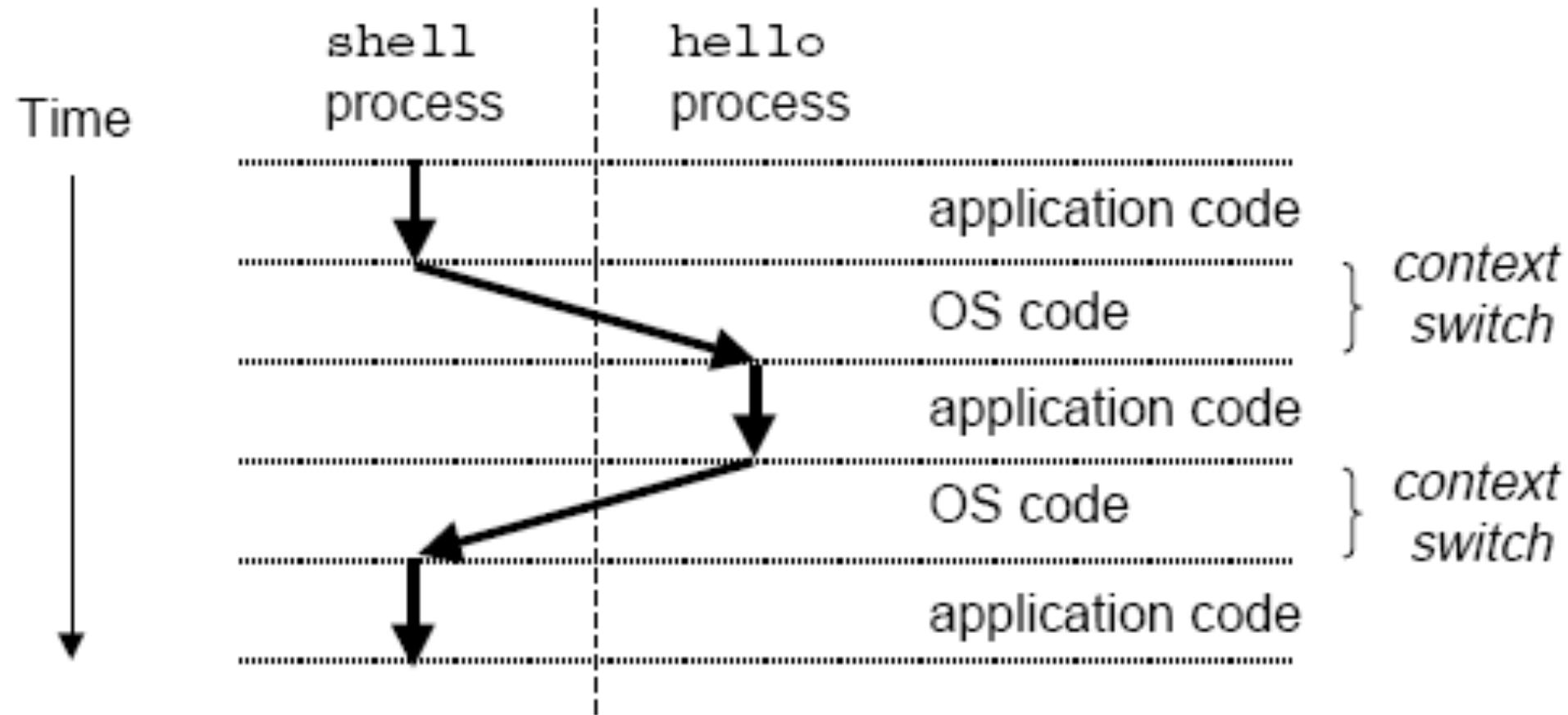
■ 进程的上下文

- 指进程运行所需的所有状态信息，例如：PC、寄存器堆的当前值、cache内容、段/页表等
- 系统中有一套状态单元，用以存放当前运行进程的上下文

■ 上下文切换过程（任何时刻，系统中只有一个进程正在运行）

- 上下文切换指把正在运行的进程换下，换一个新进程到处理器执行，上下文切换时，必须保存换下进程的上下文，恢复换上进程的上下文

进程



- 开始，Shell进程等待命令行输入
- 输入“Hello”后Shell进行系统调用
- OS保存shell上下文，创建并换入hello进程
- Hello进程中止时，进行系统调用
- OS恢复shell进程上下文，并换入shell进程

使系统中尽量多地存储用户程序的解决办法

- (1) 扩大主存(程序越来越长、主存贵，不是根本办法)
- (2) 采用交换(Exchange)方式和覆盖(Overlap)技术
 - 存储器中无处于就绪状态的进程（例如：某一时刻所有进程都在等待I/O）时，处理器将一些进程调出写回到磁盘，然后OS再调入其他进程执行，或新的作业直接覆盖老作业的存储区。
 - 分区(Partitioning)和分页(Paging)是交换的两种实现方式
 - “交换”和“覆盖”技术的缺点：对程序员不透明、空间利用率差

分区

■ 简单分区

- 主存分配：操作系统（固定）+用户区（分区）
- 使用长度不等的固定长分区(fixed-size partition)。
- 当一个进程调入主存时，分配给它一个能容纳它的最小的分区。

■ 简单分区方式的缺点：

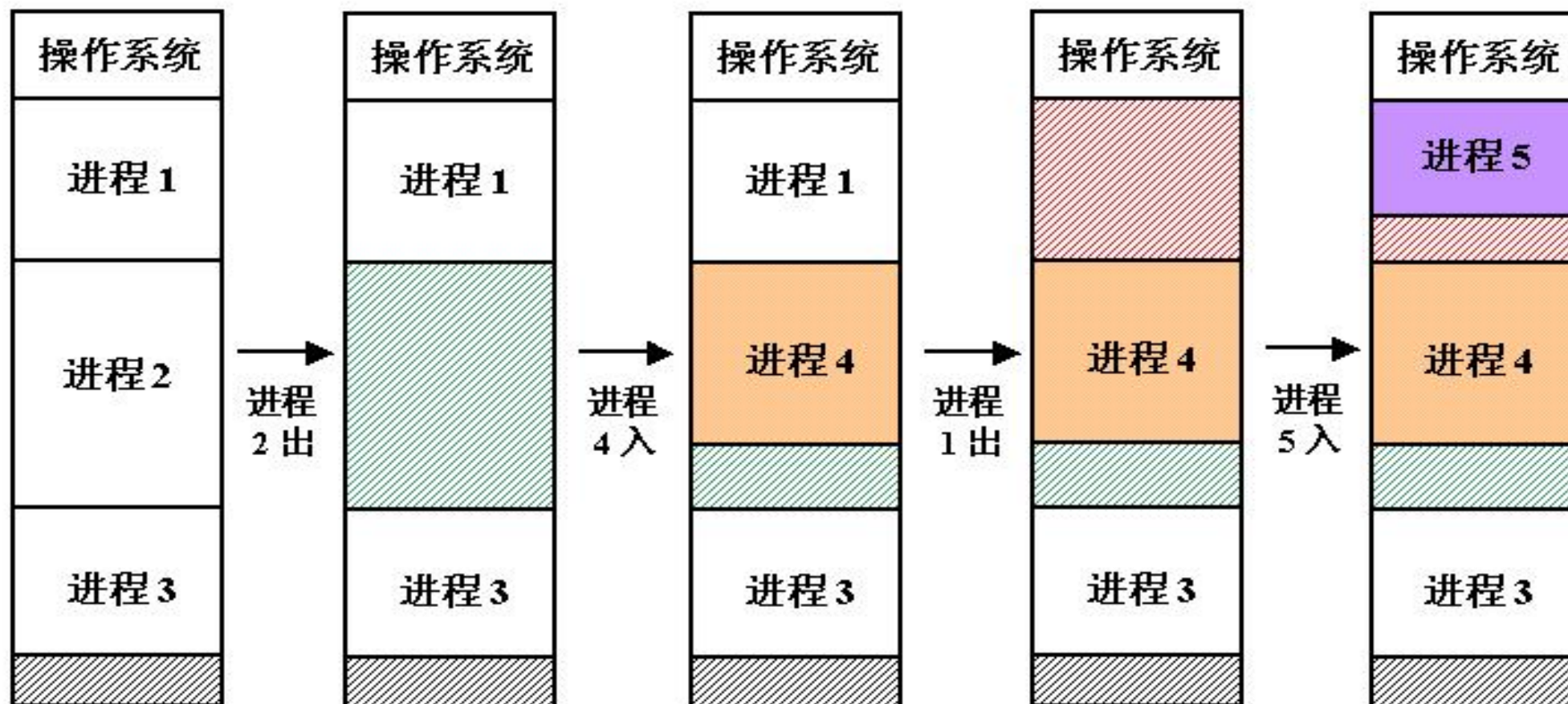
- 因为是固定长度的分区，故可能会浪费主存空间。多数情况下，进程对分区大小的需求不可能和提供的分区大小一样。

操作系统 128K
64K
192K
256K
384K

分区

■ 可变长分区

- 分配的分区大小与进程所需大小一样。
- 特点：开始较好，但到最后在存储器中会有许多小空块出现。时间越长，存储器中的碎片就会越来越多，因而存储器的利用率下降。



分页

■ 基本思想：

- 内存被分成固定长且比较小的存储块（页框、实页、物理页）
- 每个进程也被划分成固定长的程序块（页、虚页、逻辑页）
- 程序块可装到存储器中可用的存储块中
- 无需用连续页框来存放一个进程
- 操作系统为每个进程生成一个页表
- 通过页表(page table)实现逻辑地址向物理地址转换（Address Mapping）

■ 逻辑地址（Logical Address）：

- 程序中的指令所用的地址，也称为虚拟地址

■ 物理地址（physical或Memory Address）：

- 存放指令或数据的实际内存地址，也称为实地址、主存地址

使系统中尽量多地存储用户程序的解决办法

- (1) 扩大主存(程序越来越长、主存贵, 不是根本办法)
- (2) 采用交换(Exchange)方式和覆盖(Overlap)技术
 - 存储器中无处于就绪状态的进程 (例如: 某一时刻所有进程都在等待I/O) 时, 处理器将一些进程调出写回到磁盘, 然后OS再调入其他进程执行, 或新的作业直接覆盖老作业的存储区。
 - 分区(Partitioning)和分页(Paging)是交换的两种实现方式
 - “交换”和“覆盖”技术的缺点: 对程序员不透明、空间利用率差
- (3) 虚拟存储器(Virtual Memory)
 - 类似上述分页方式, 但不是把所有页面一起调到主存, 而是采用“按需调页Demand Paging”, 在外存和主存间以固定页面进行调度。
 - 虚拟存储器方式下, 引入了虚拟地址空间的概念。

虚拟存储器

- 提供一种容量非常大的存储器
 - 多个任务所需存储器的总和大于实际存储器空间
 - 单个程序的地址空间超过了实际存储器
- 使得珍贵的物理存储器得到更好地利用
- 简化对存储的管理

虚拟存储器基本概念

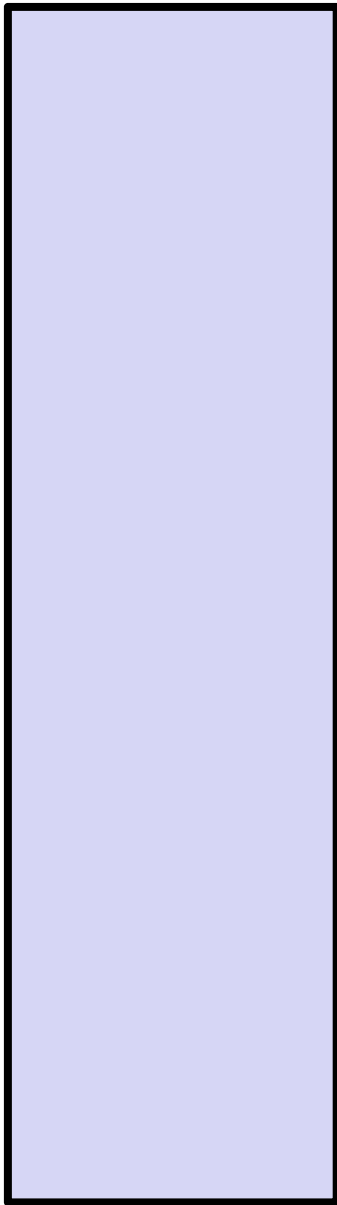
■ 虚拟存储技术的实质

- 程序员在比实际主存空间大得多的逻辑地址空间中编写程序
- 程序执行时，把当前需要的程序段和相应的数据块调入主存，其他暂不用的部分存放在磁盘上
- 指令执行时，通过硬件将逻辑地址（也称虚拟地址或虚地址）转化为物理地址（也称主存地址或实地址）
- 在发生程序或数据访问失效时，由操作系统进行主存和磁盘之间的信息交换

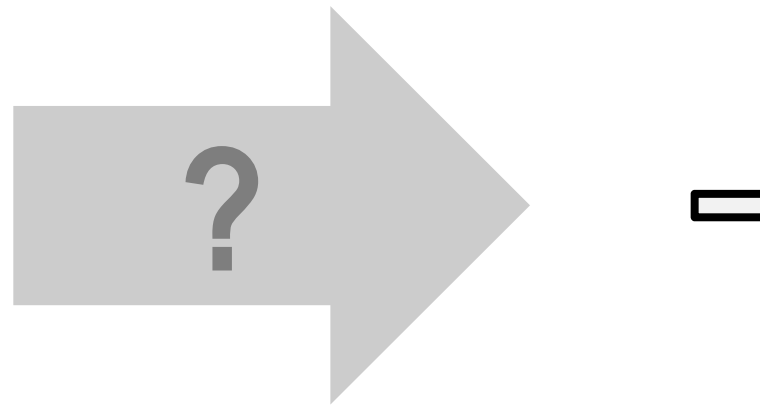
■ 虚拟存储器机制由硬件与操作系统共同协作实现，涉及到操作系统中的许多概念，如进程、进程的上下文切换、存储器分配、虚拟地址空间、缺页处理等。

如何将所需的数据都装入主存

**64-bit addresses:
16 Exabyte**



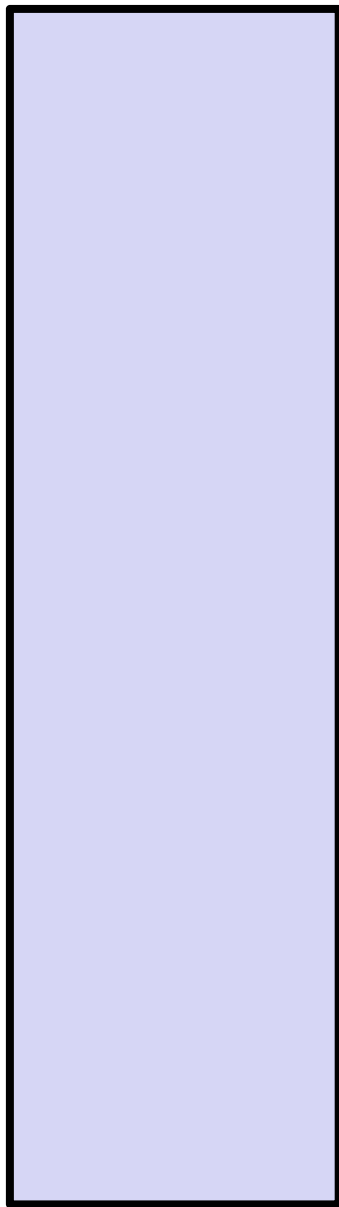
**Physical main memory:
Few Gigabytes**



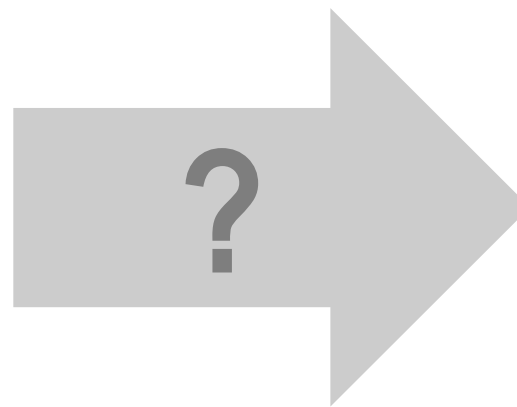
And there are many processes

如何将所需的数据都装入主存

64-bit addresses:
16 Exabyte



Physical main memory:
Few Gigabytes



=

And there are many processes

二进制前缀 (IEC 60027-2)

名字	缩写	次方
kibibyte	KiB	2^{10}
mebibyte	MiB	2^{20}
gibibyte	GiB	2^{30}
tebibyte	TiB	2^{40}
pebibyte	PiB	2^{50}
exbibyte	EiB	2^{60}
zebibyte	ZiB	2^{70}
yobibyte	YiB	2^{80}

如何有效地管理存储资源

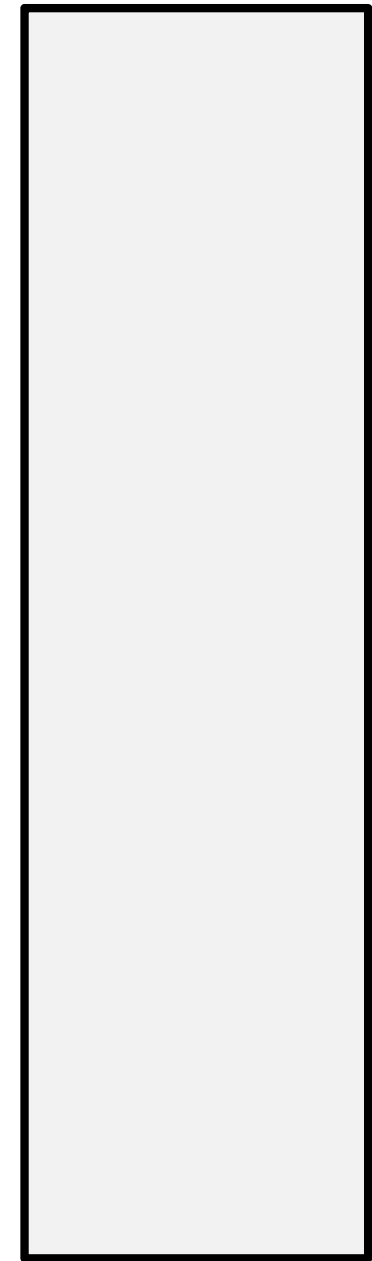
Process 1
Process 2
Process 3
...
Process n

X

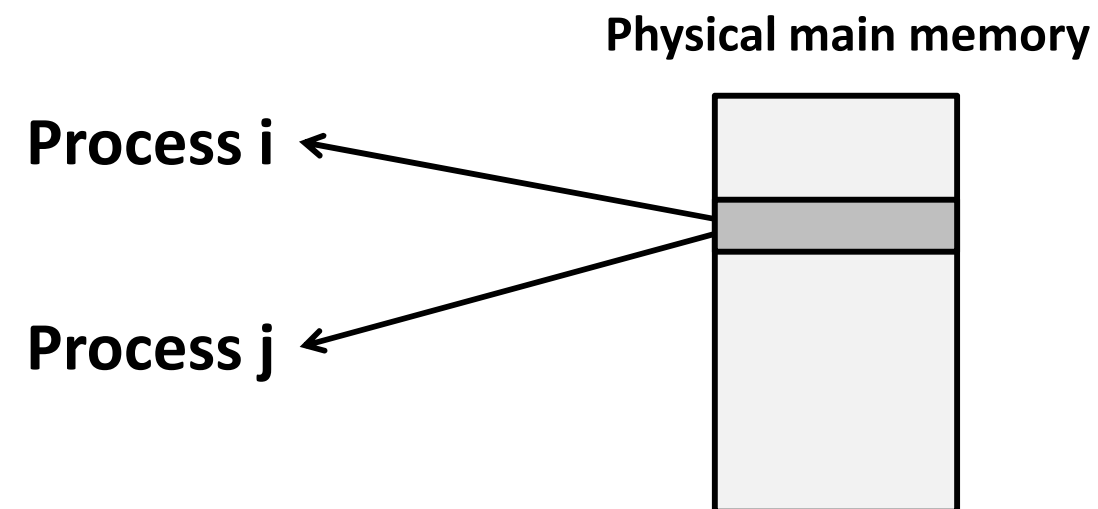
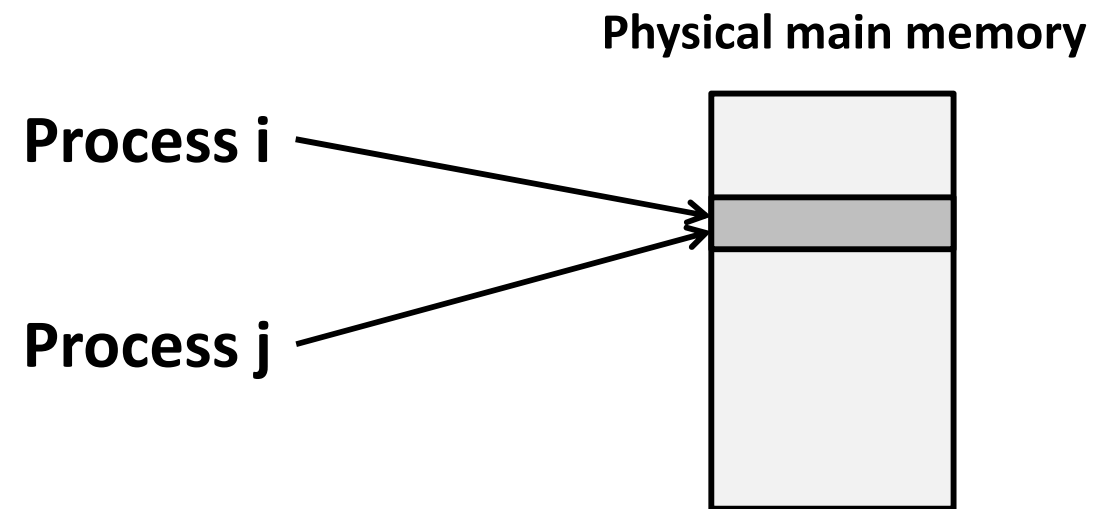
stack
heap
.text
.data
...

*What goes
where?*

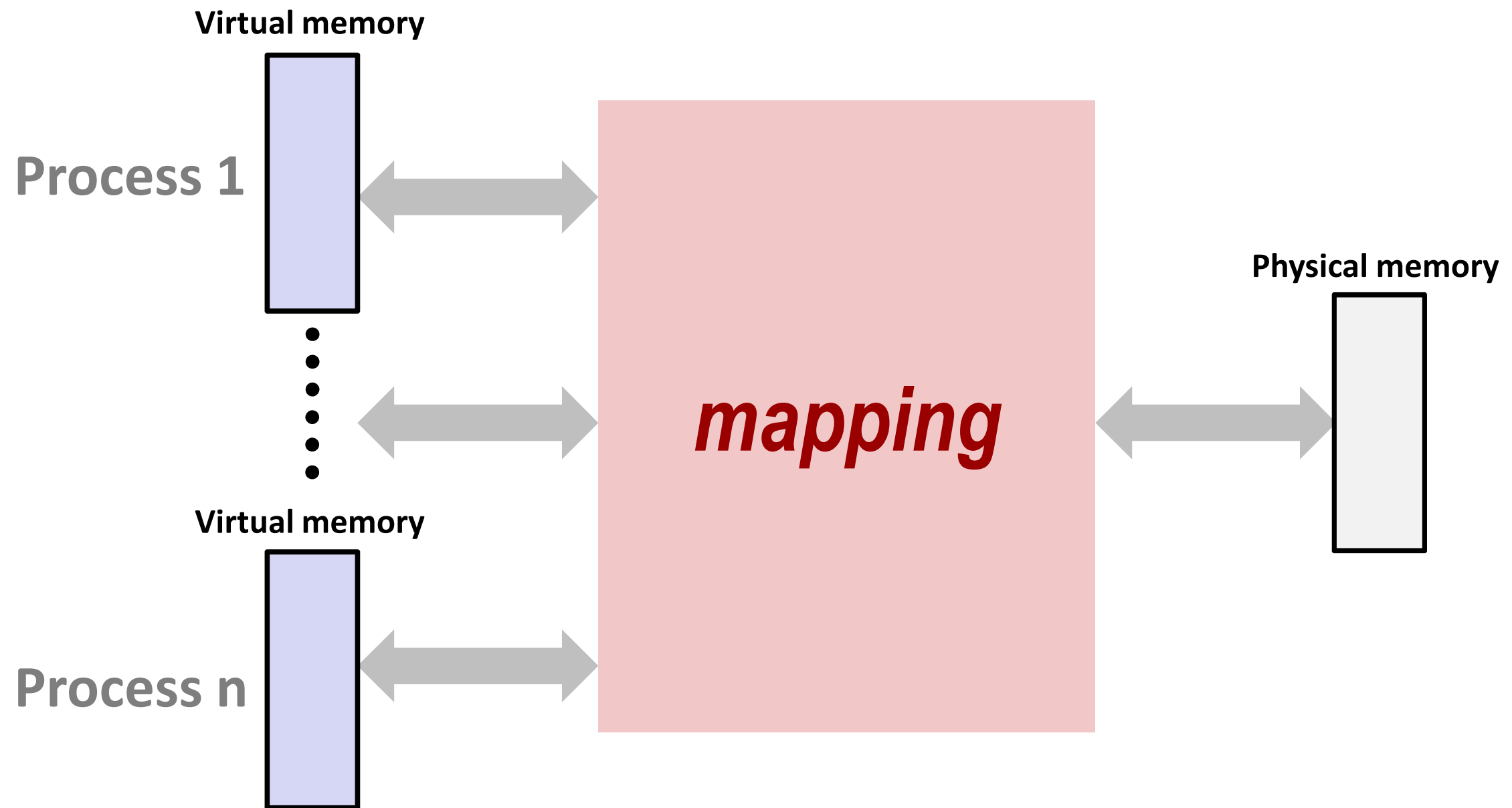
Physical main memory



如何内存共享和保护？

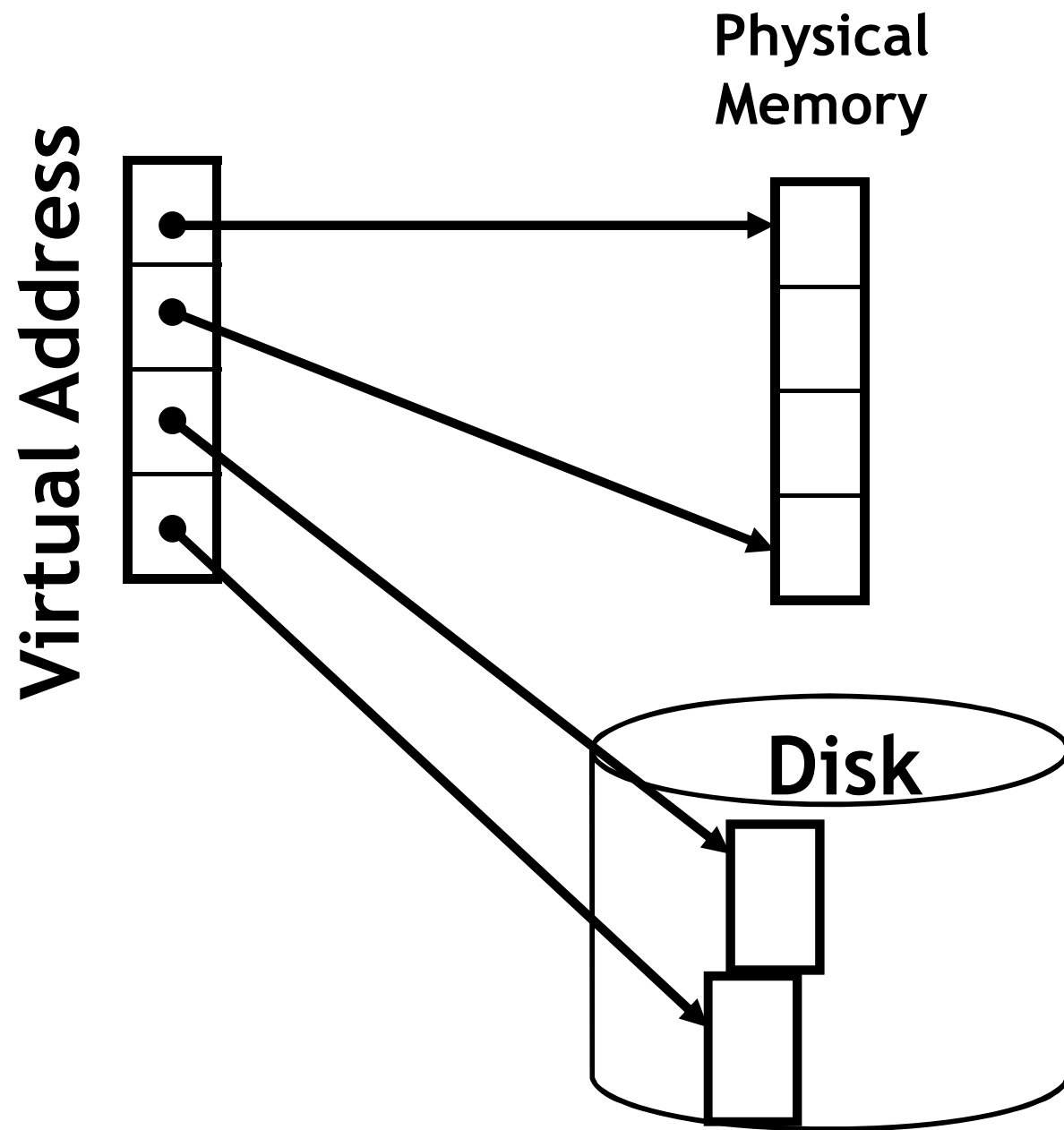


虚拟存储器



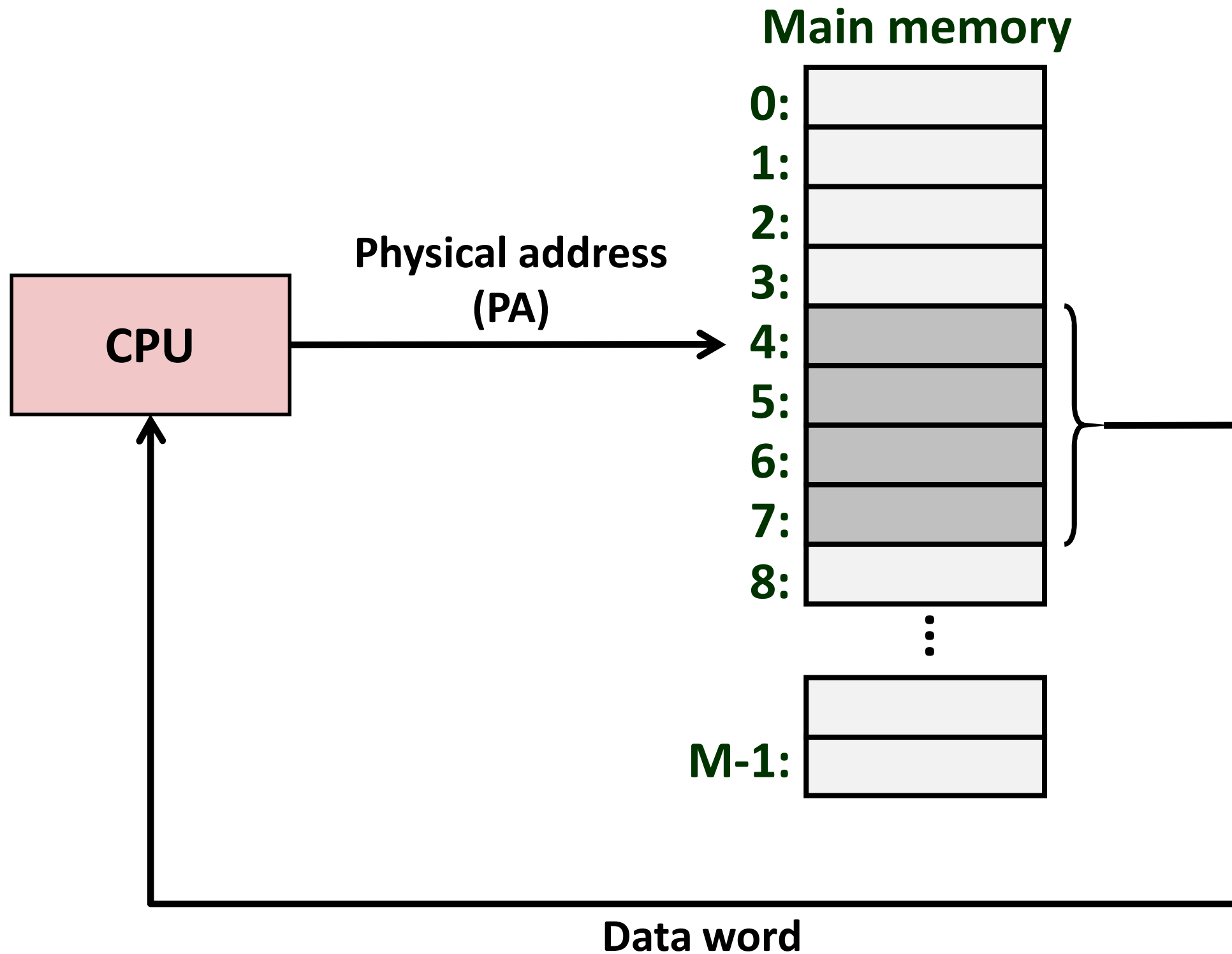
每个进程拥有自己的虚拟地址空间

地址映射



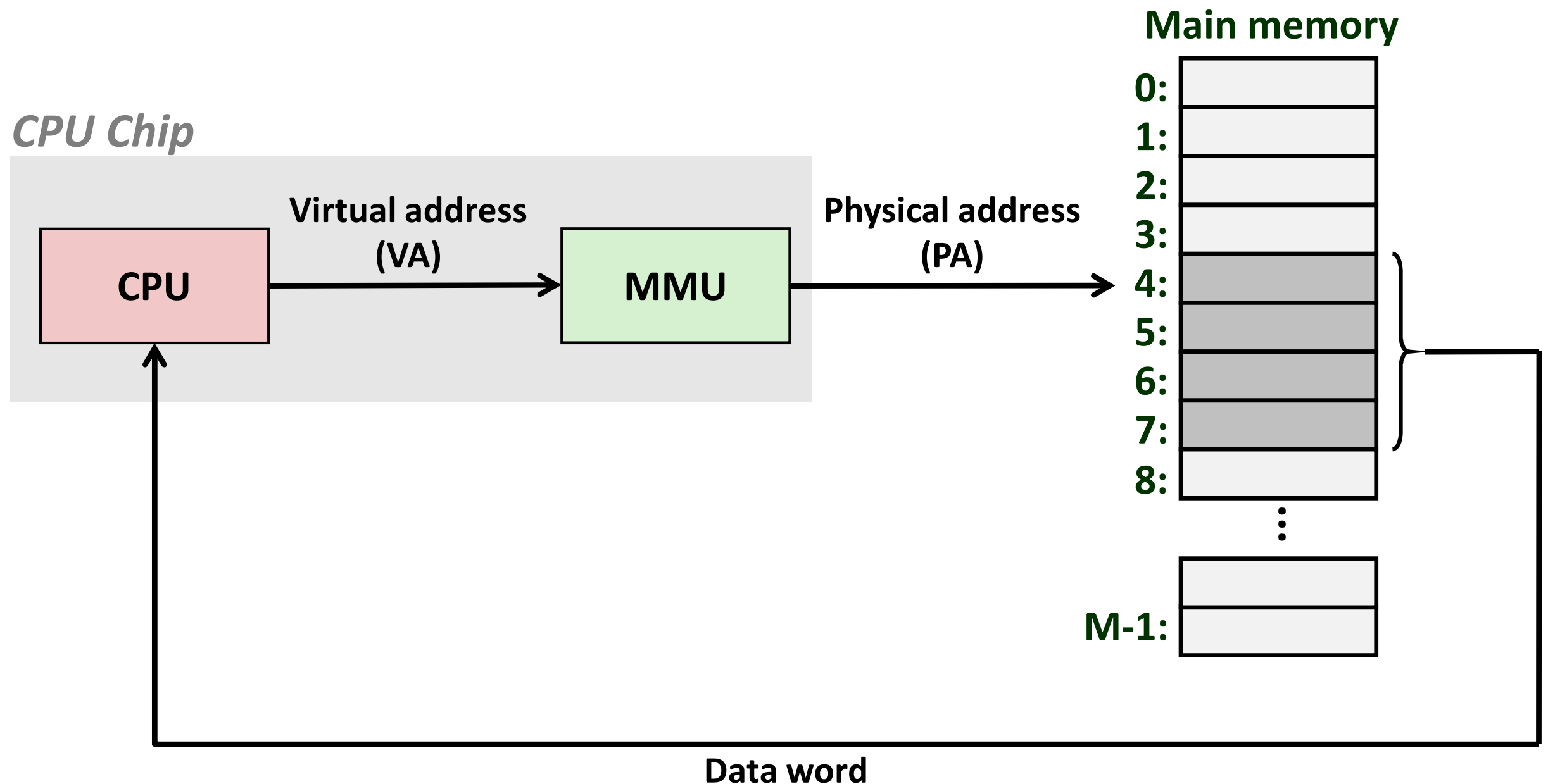
A virtual address can be mapped to either physical memory or disk.

使用物理地址的计算机系统



嵌入式系统：汽车、电梯、电子相框等

使用虚拟存储器的计算机系统

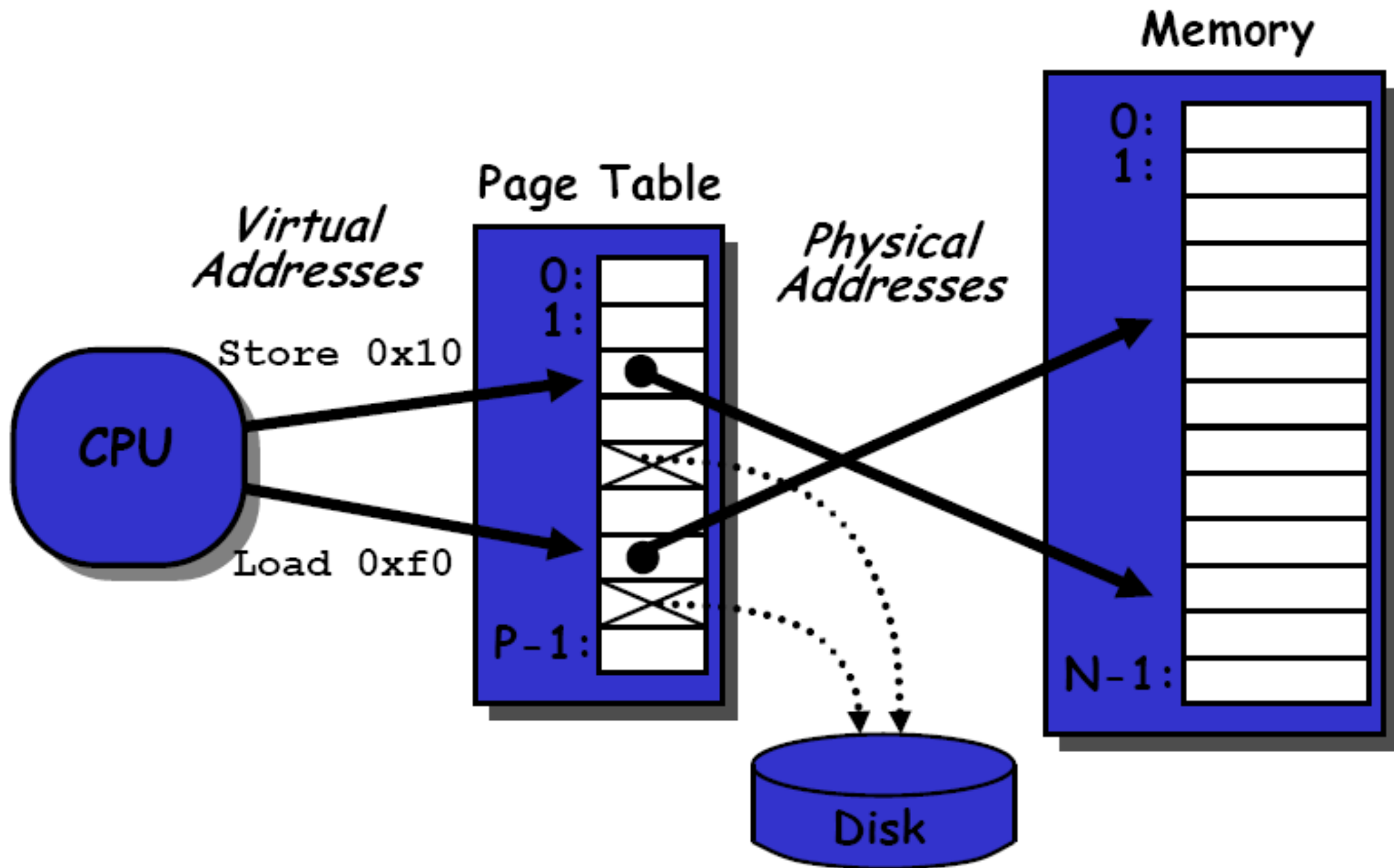


所有现代计算机，包括个人电脑、工作站等
计算机科学中的一个重要创新

虚拟存储器的作用

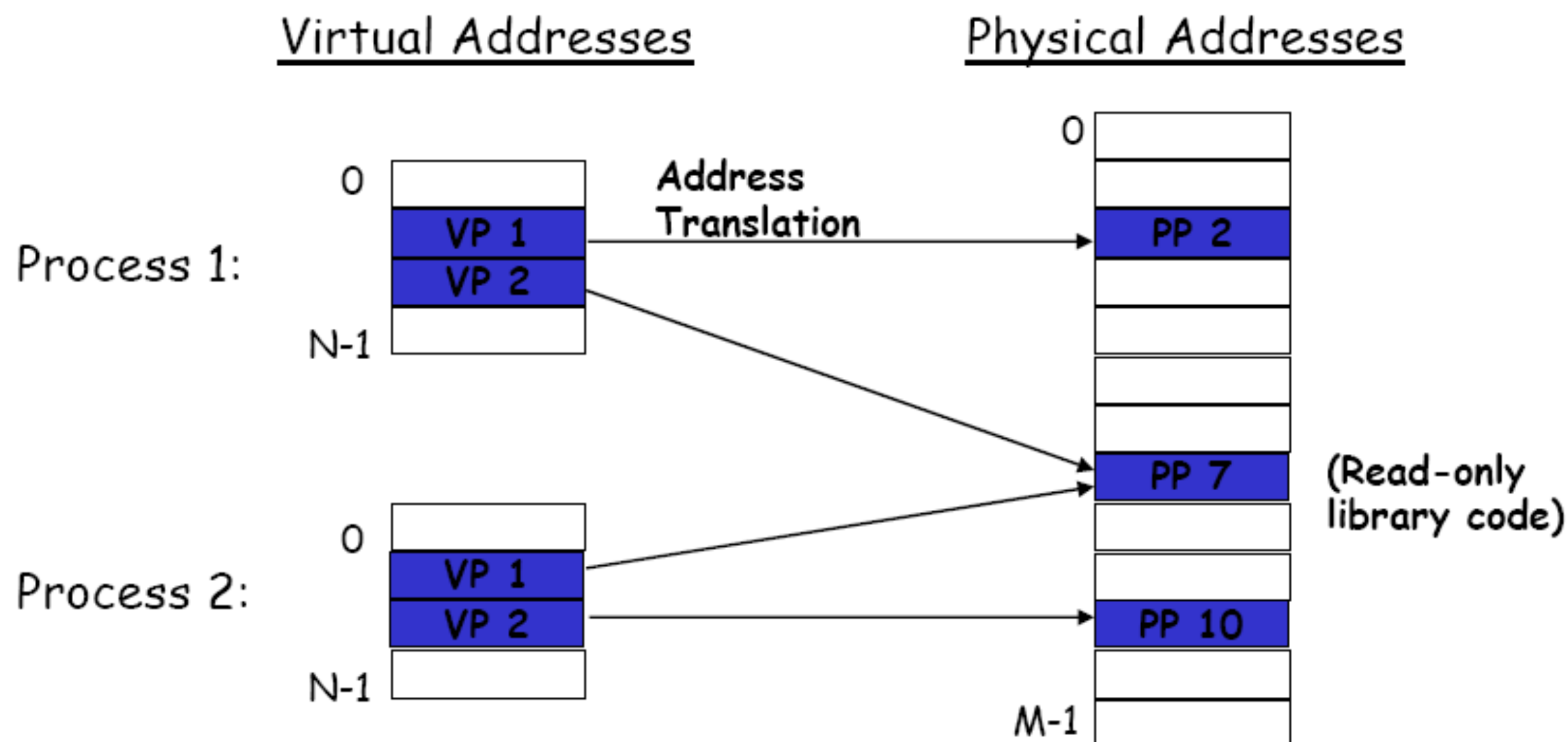
- 获得更大的逻辑空间
- 实现内存共享
- 实现内存保护

更大的逻辑地址空间



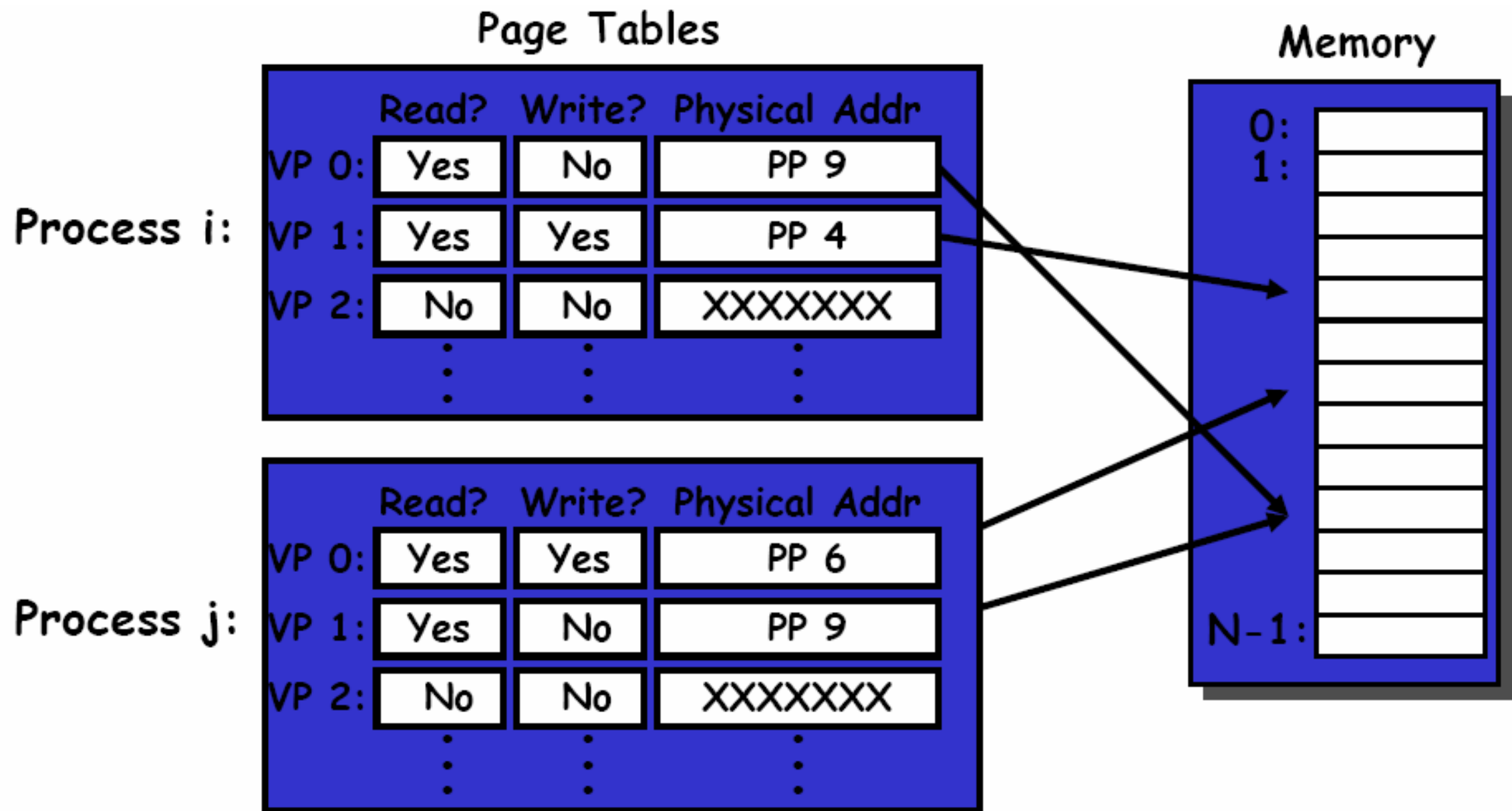
- 通过页表将虚地址转换为实地址

实现内存共享



- 每个进程有独立的逻辑地址空间
- 操作系统来控制逻辑地址和物理地址的转换

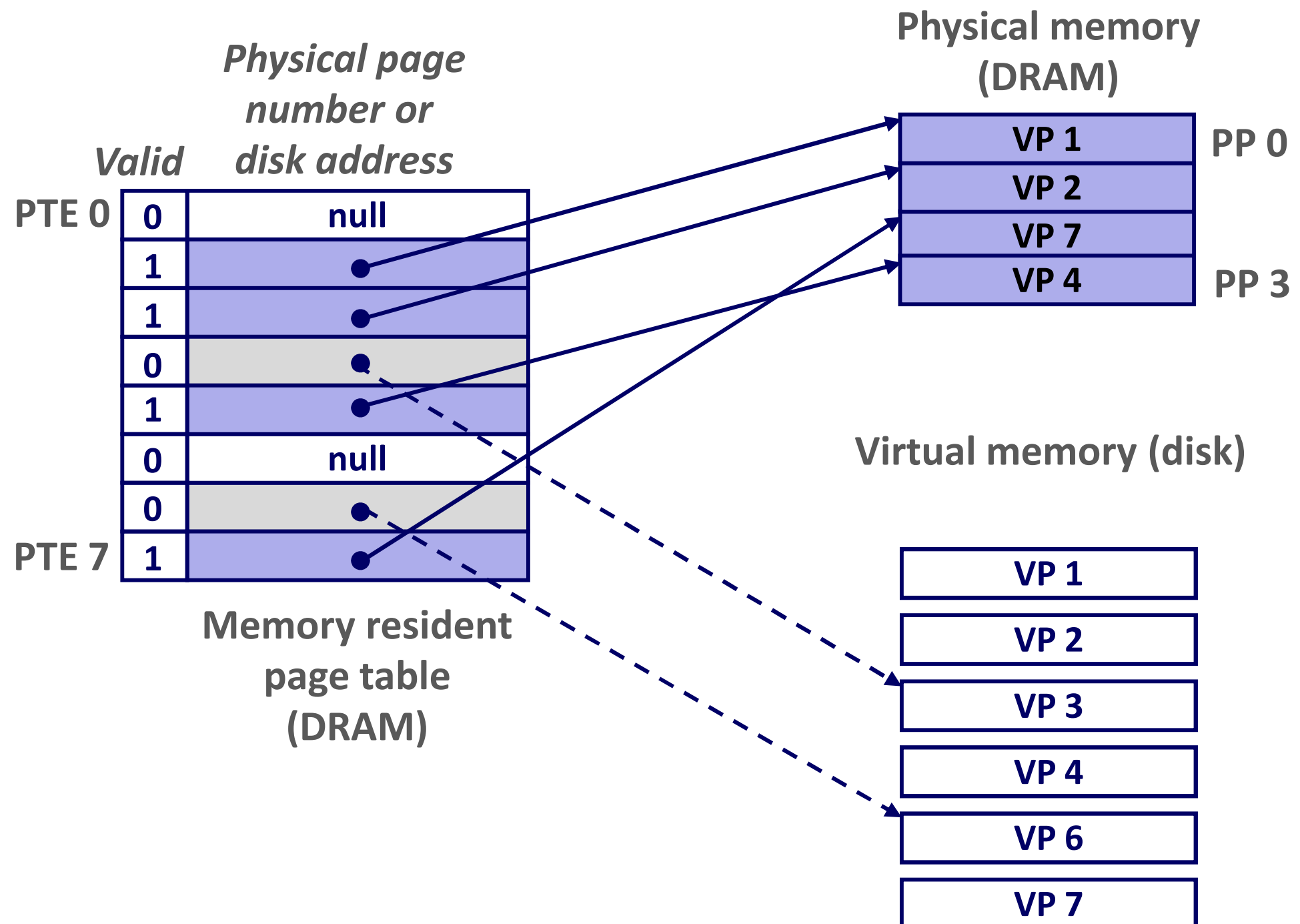
实现内存保护



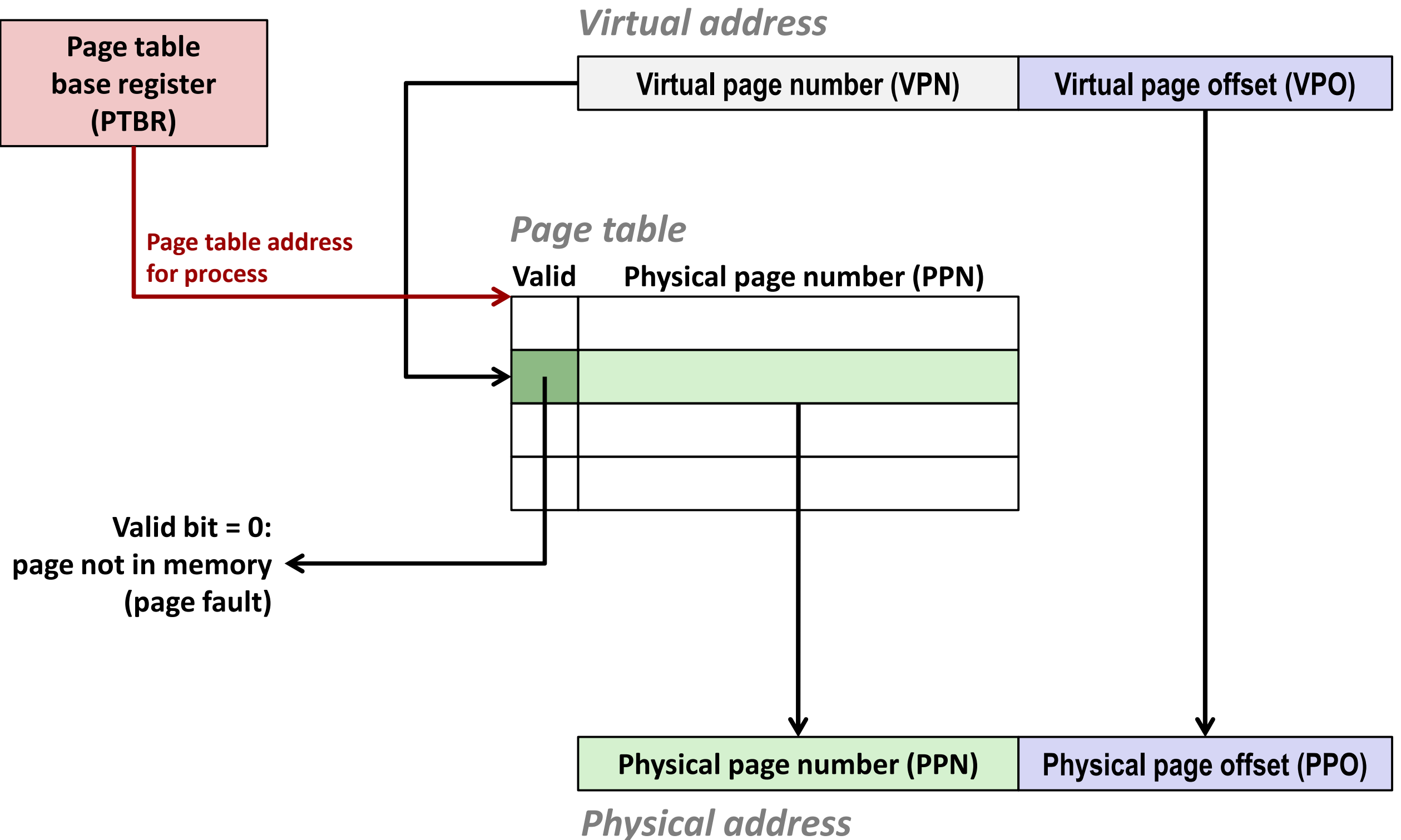
- 页表中存放有访问权限

通过页表的地址转换

- 页表记录了一组由虚拟页向物理页的映射

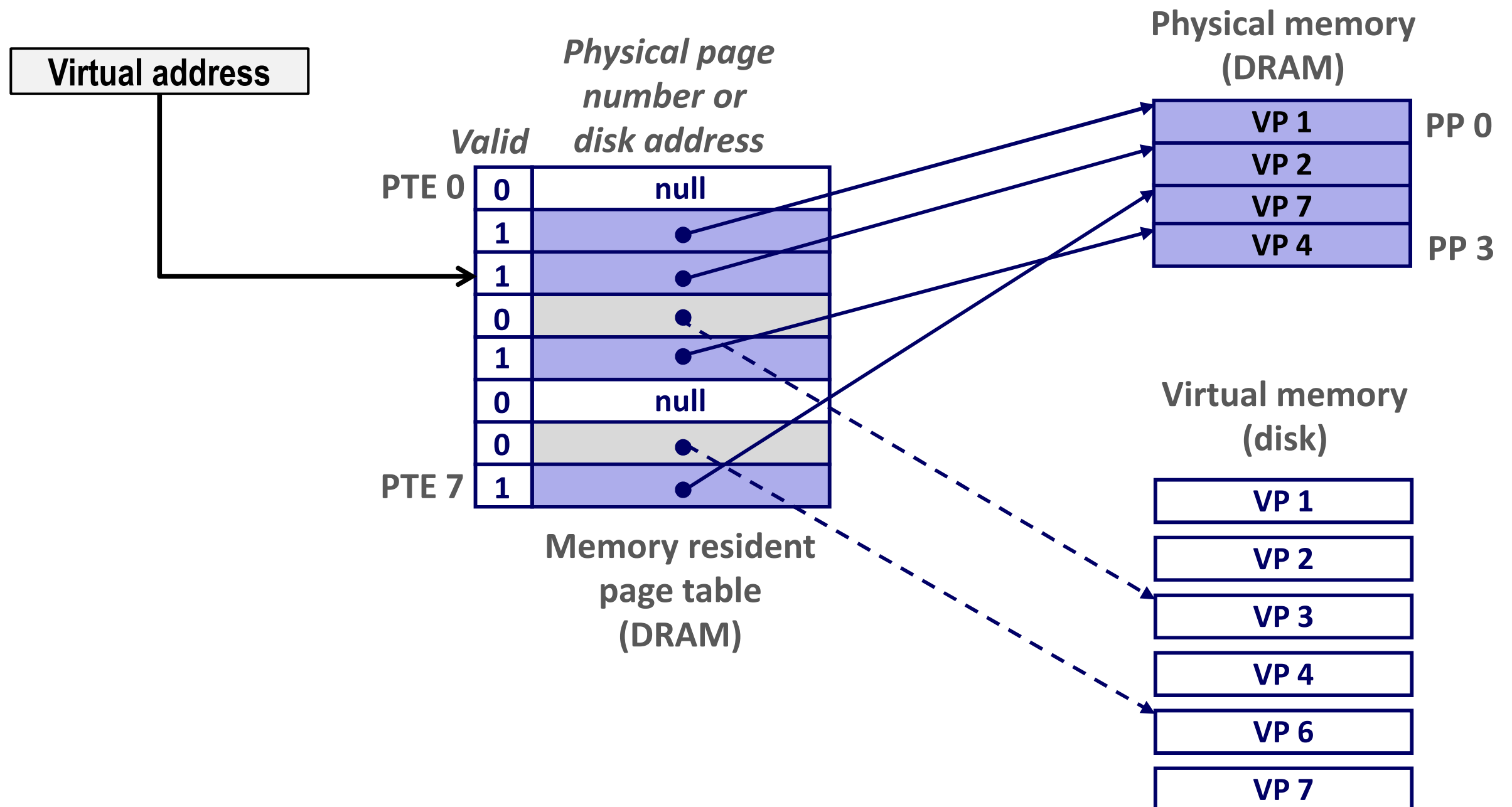


通过页表的地址转换



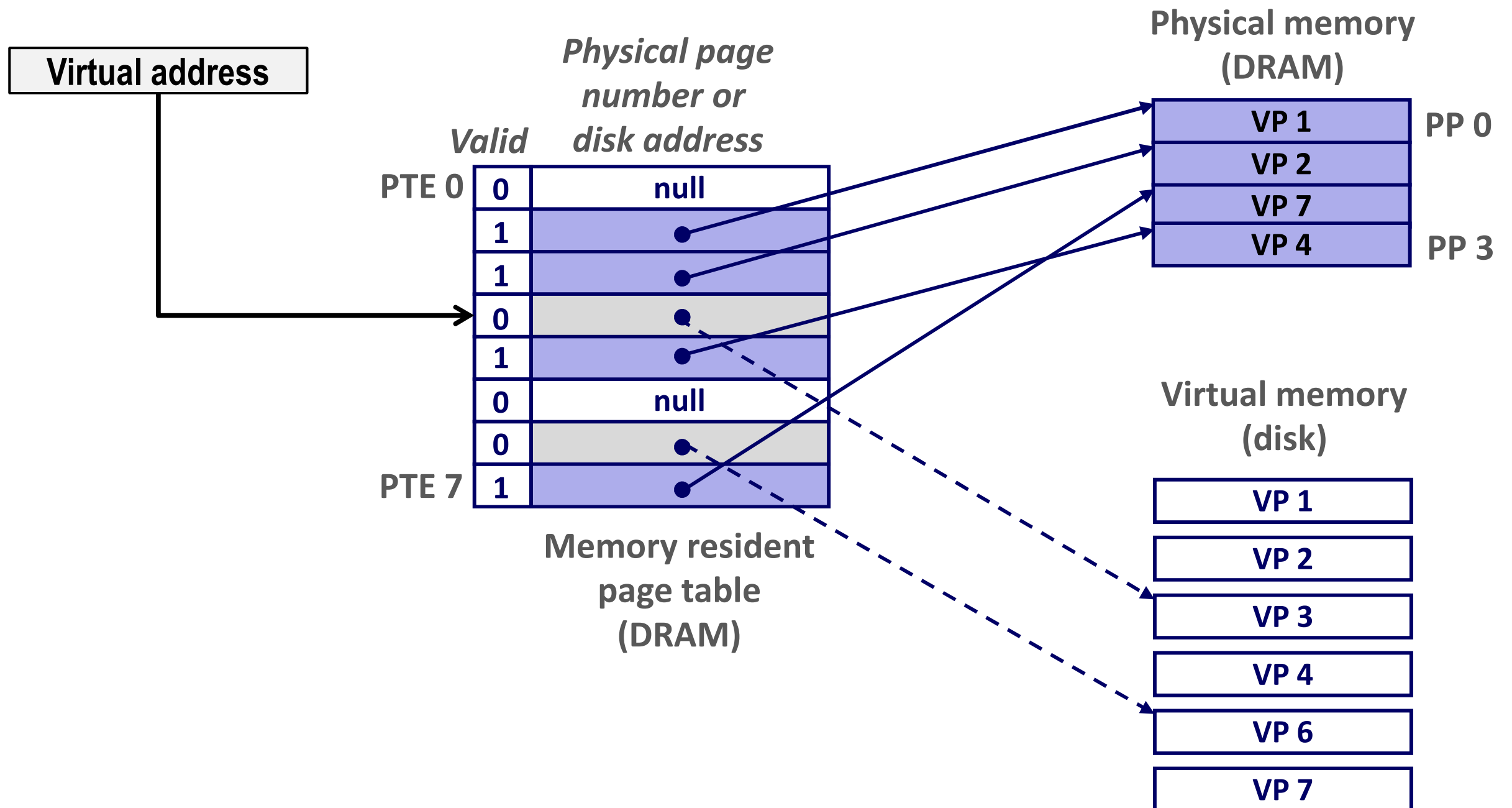
页命中

- 需要访问的虚地址中的字在主存中（有效位为1）



缺页

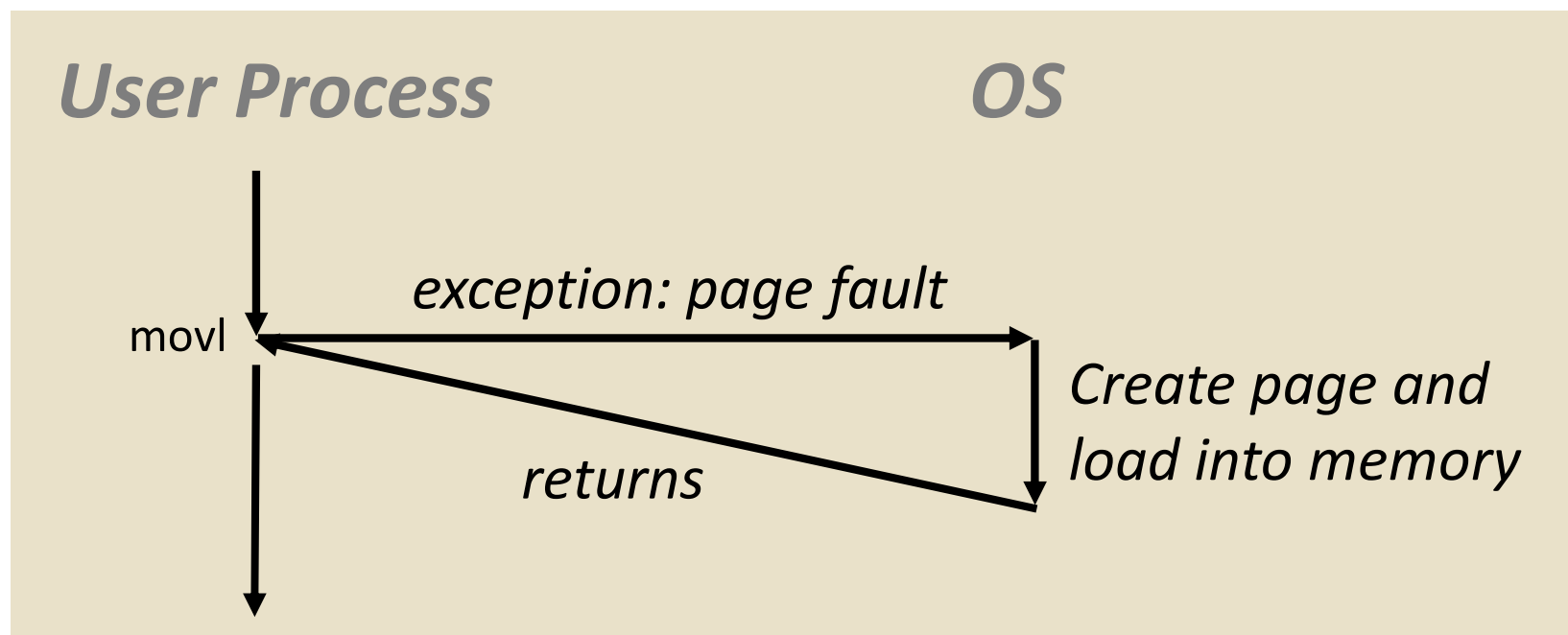
- 需要访问的虚地址中的字在不主存中（有效位为0）



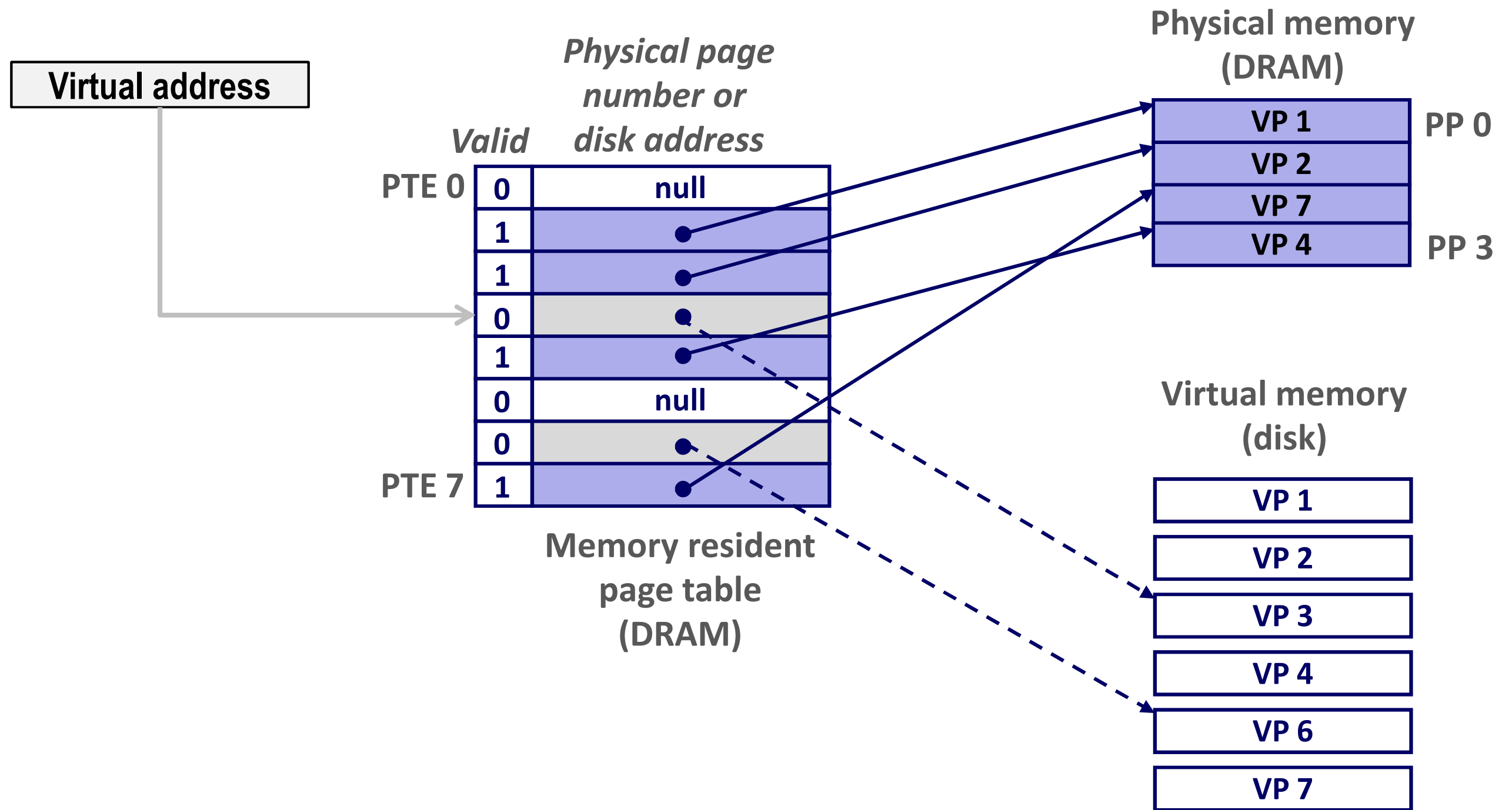
发生缺页

- 发生缺页后，操作系统将获得控制权（异常机制），其在下一级存储层次（通常是磁盘）中找到该页，然后将请求页放到主存的某个位置

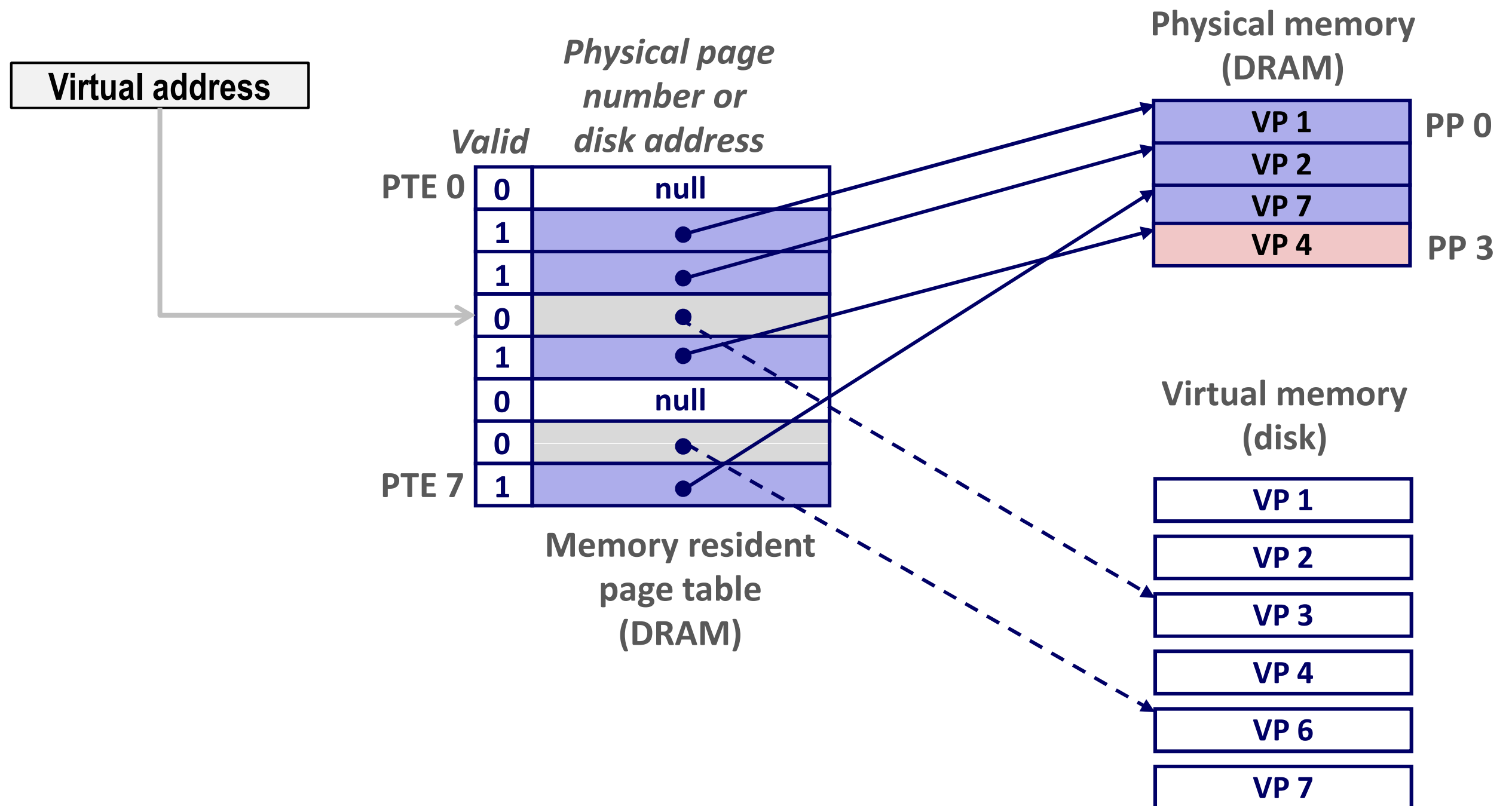
```
int a[1000];  
main ()  
{  
    a[500] = 13;  
}
```



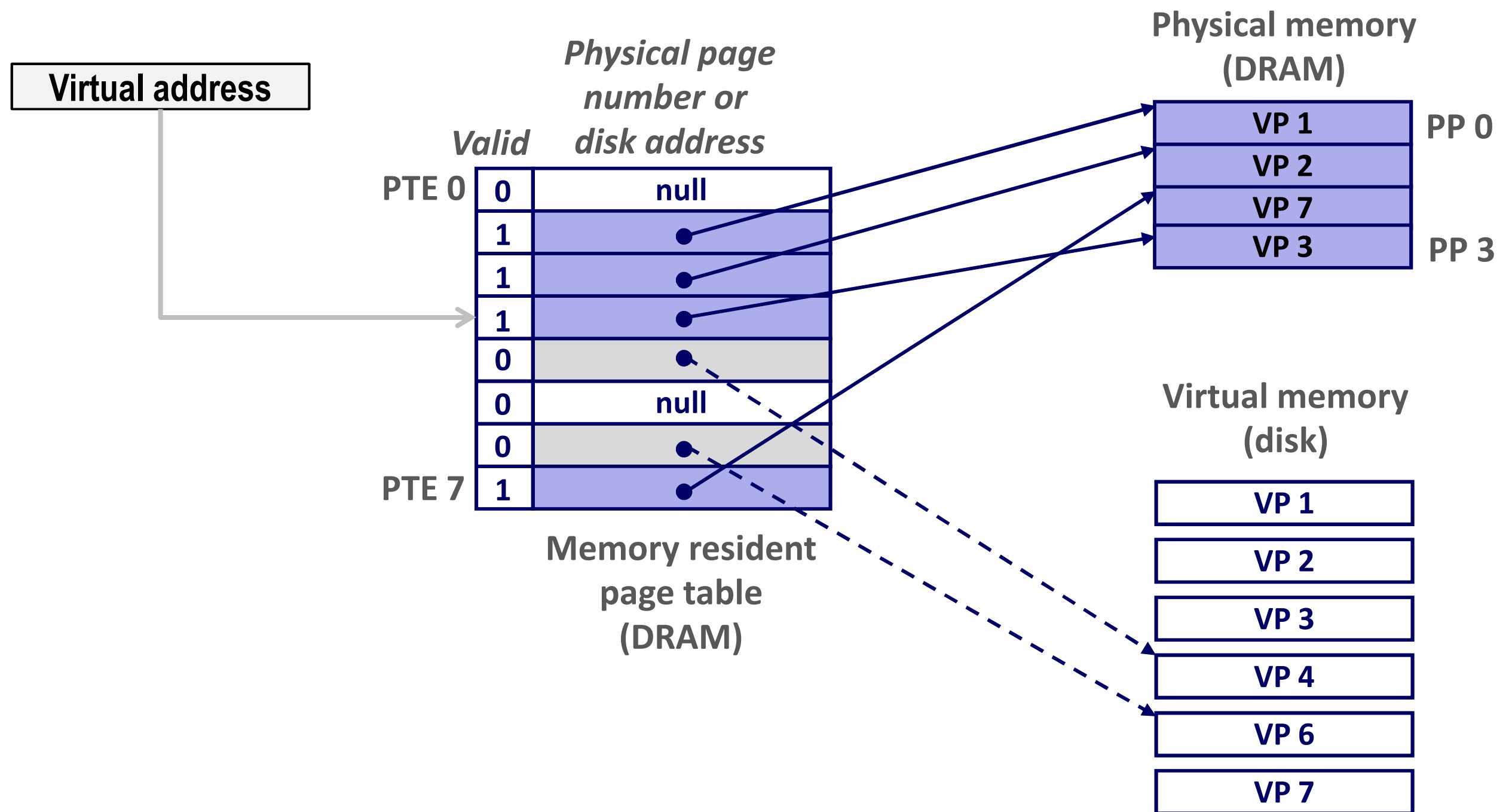
缺页处理



缺页处理



缺页处理



页面替换算法

■ 最近最少使用(LRU)

- 将页帧按照最近最多使用到最近最少使用进行排序,再次访问一个页帧时,将该页帧移到表头,替换时将表尾的页帧换出

虚拟存储器目的

■ 根本目的

- 获得运行比物理存储器更大空间程序的能力

■ 保护

- 操作系统可以对虚拟存储空间进行特定的保护...

■ 灵活

- 程序的某部分可以装入主存的任意位置

■ 提高存储效率

- 只在主存储器中保留最重要的部分

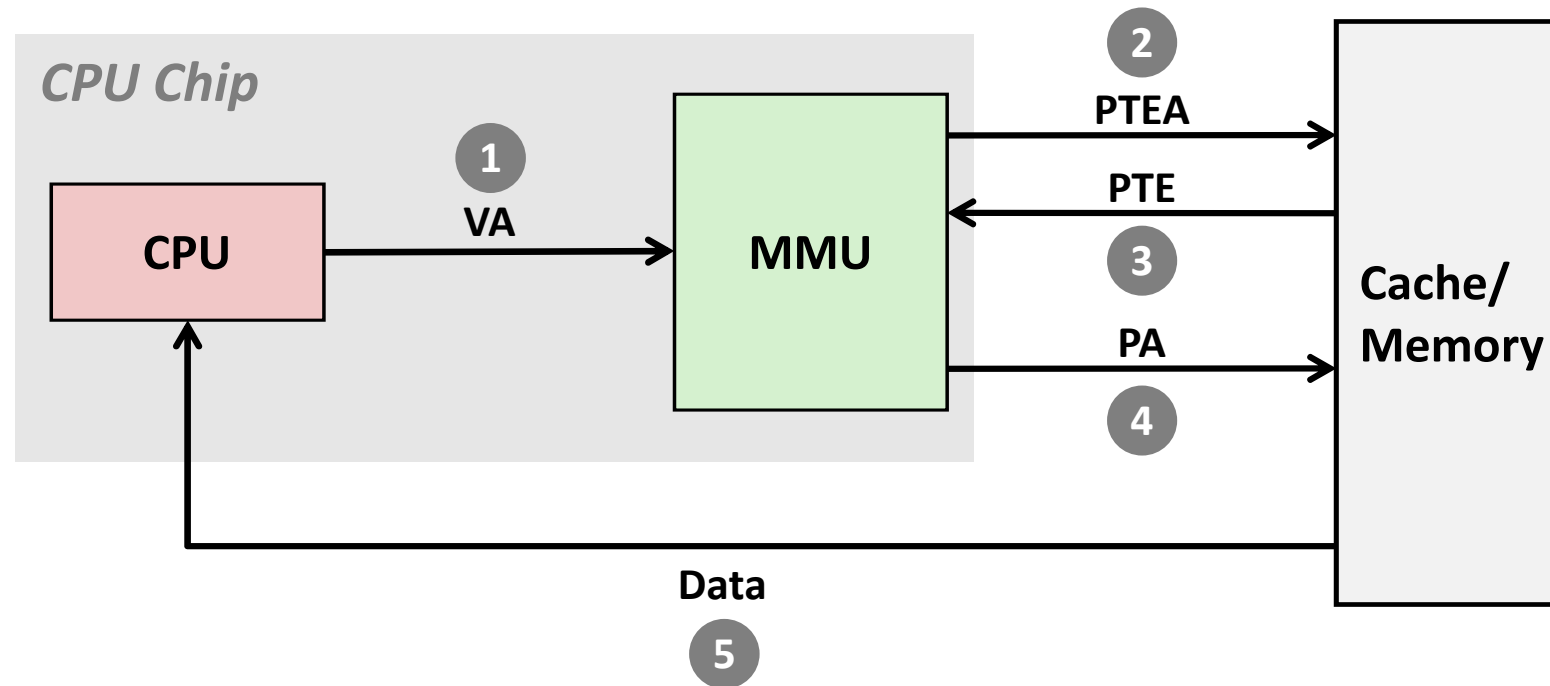
■ 提高并行度

- 在进行段页替换的同时可以执行其它进程

■ 可扩展

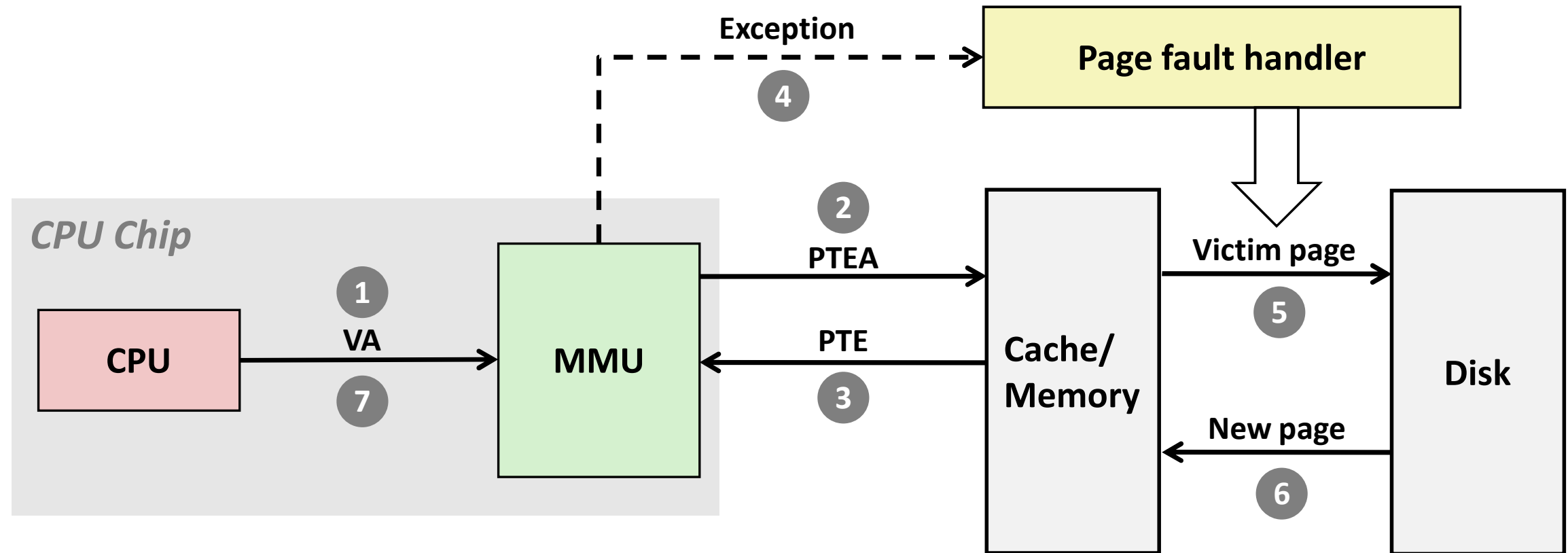
- 为对象提供了扩展空间的能力.

地址转换： 页命中



- (1)CPU向MMU发送虚拟地址
- (2-3)MMU从页表中获取页表项
- (4)MMU向主存或cache发送物理地址
- (5)主存或cache向CPU发送数据字

地址转换：缺页

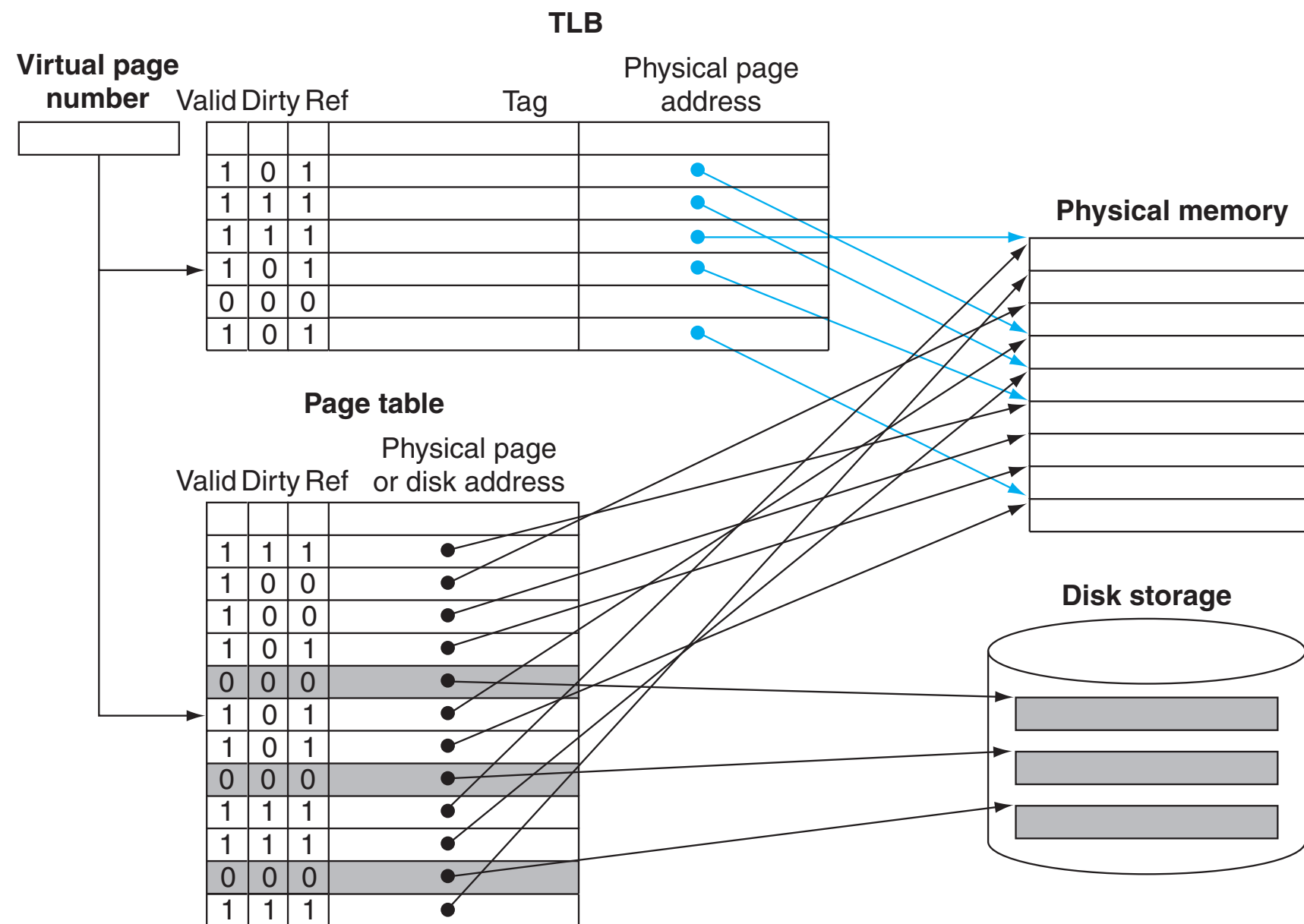


- (1) CPU向MMU发送虚拟地址
- (2-3) MMU从页表中获取页表项
- (4) 有效位为0，MMU触发缺页异常
- (5) 选择被替换的页
- (6) 在主存中插入新的页
- (7) 返回原来的进程，重新执行刚才发生异常的操作

地址转换的效率问题

- 由于页表存放在主存中，因此程序每次访存至少需要两次：
 - 一次访存获取物理地址
 - 第二次访存获取数据
- 提高地址转换速度的方法
 - 利用页表的访问局部性，即当一个转换的虚拟页号被使用时，它可能在不久的将来再次被使用；
 - 解决方案：转换旁路缓冲TLB
 - 包含在处理器中的一个特殊的cache，用以追踪最近使用过的地址变换

转换旁路缓冲TLB



■ 两种缺失

- TLB缺失：TLB中没有访问的虚页号，但是该页在主存中
- 缺页：访问的页不在主存中

转换旁路缓冲TLB

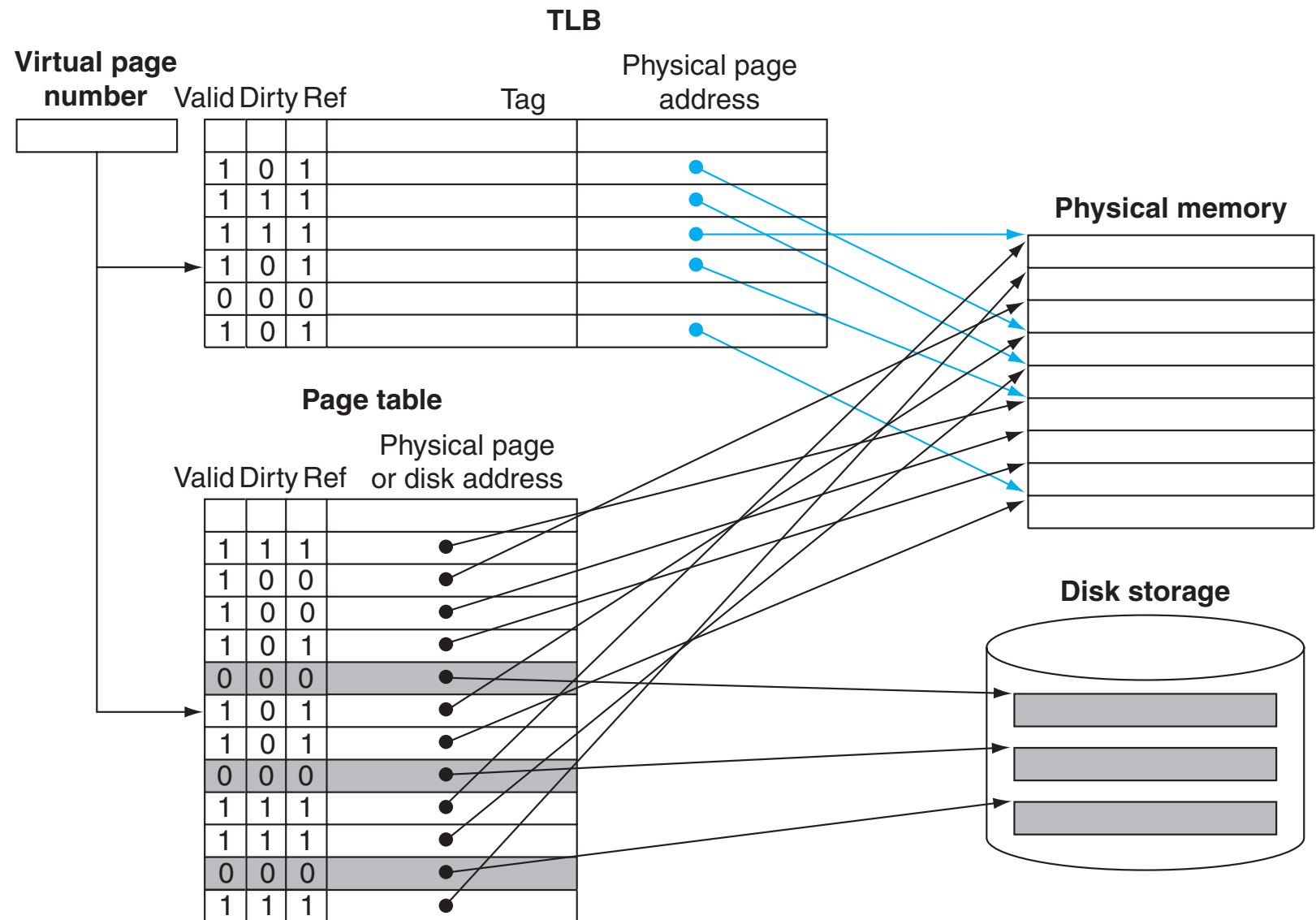
■ 访问频繁:速度第一

■ TLB 缺失将造成:

- 流水线停止
- 通知操作系统
- 读页表
- 将表项写入TLB
- 返回到用户程序
- 重新访问

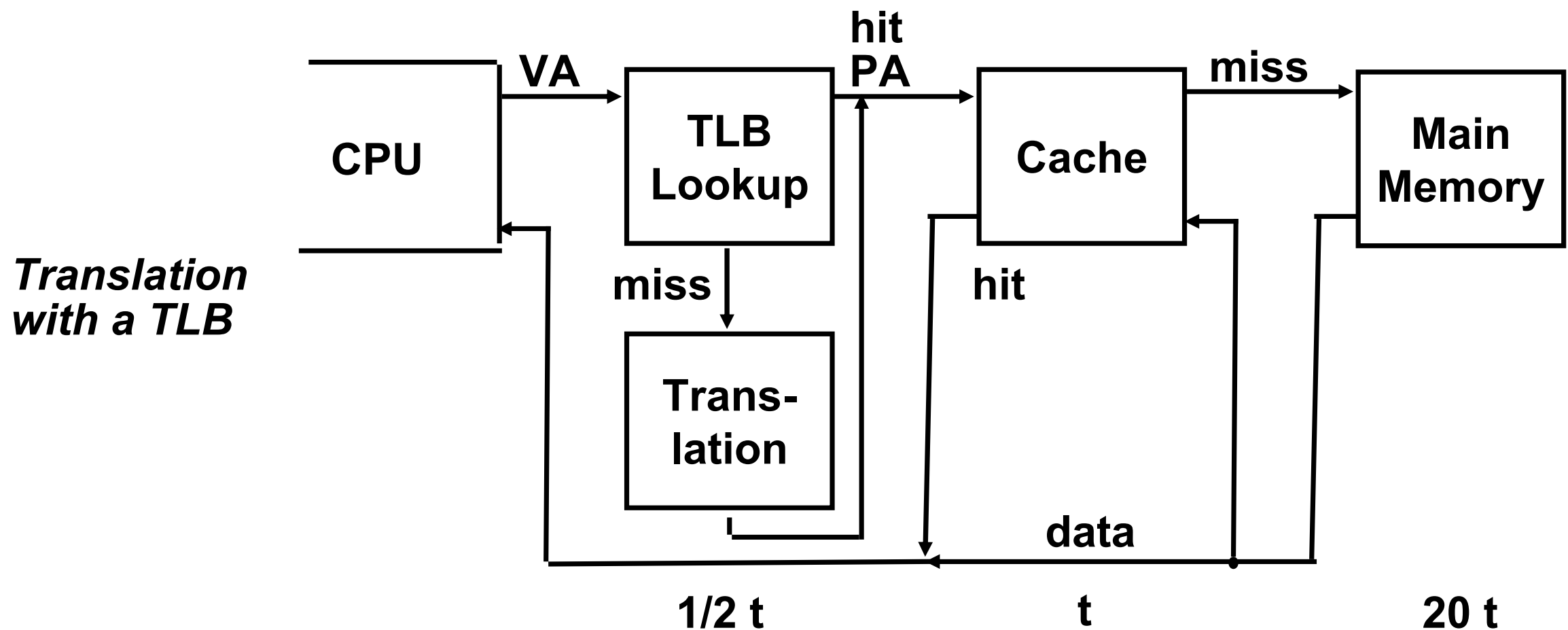
■ 因此,应尽量减少缺失:

- 多路组相连
- 再尽量提高TLB的容量



TLB

- 为页表设置的专用Cache,实现虚页号到实页号的转换
- 多路组相连、全相连
- 容量较小,128~256个表项



虚拟存储器与Cache比较

■ 虚存

- “主存——辅存层次”,主要目的是解决存储容量的问题。
- 单位时间内数据交换次数较少,但每次交换的数据量大,达几十至几千字节。

■ Cache

- Cache主要目的是解决存储速度问题,使存储器的访问速度不太影响CPU的运行速度。
- 单位时间内数据交换的次数较多,每次交换的数据量较小,只有几个到几十个字节。

虚拟存储器与Cache的不同

■ 虚拟存储器

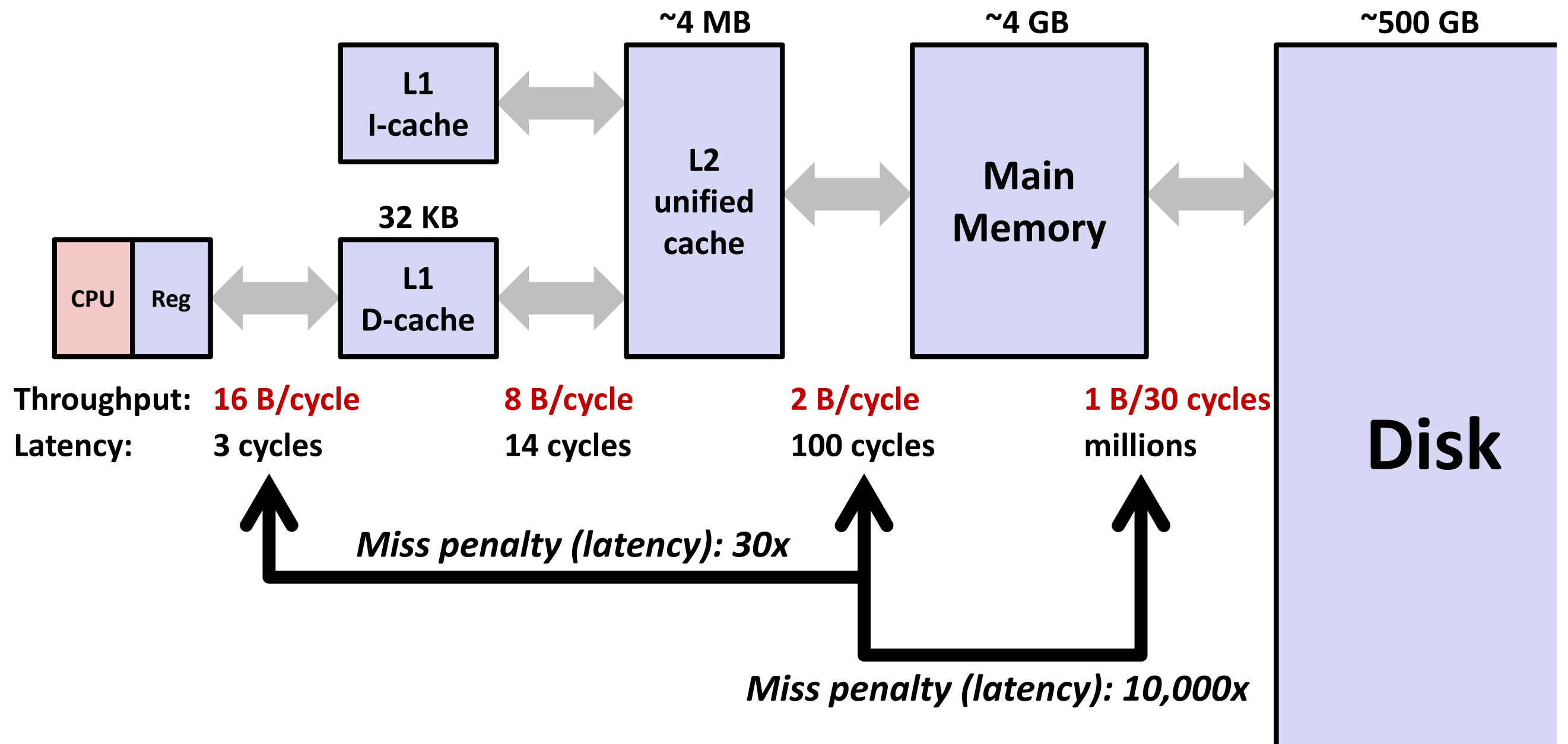
- 克服存储容量的不足
- 获得对主存储器管理的便利
- 由操作系统管理

■ 高速缓冲存储器

- 解决主存储器与CPU 性能的差距
- 获得最小粒度的访问
- 由硬件实现

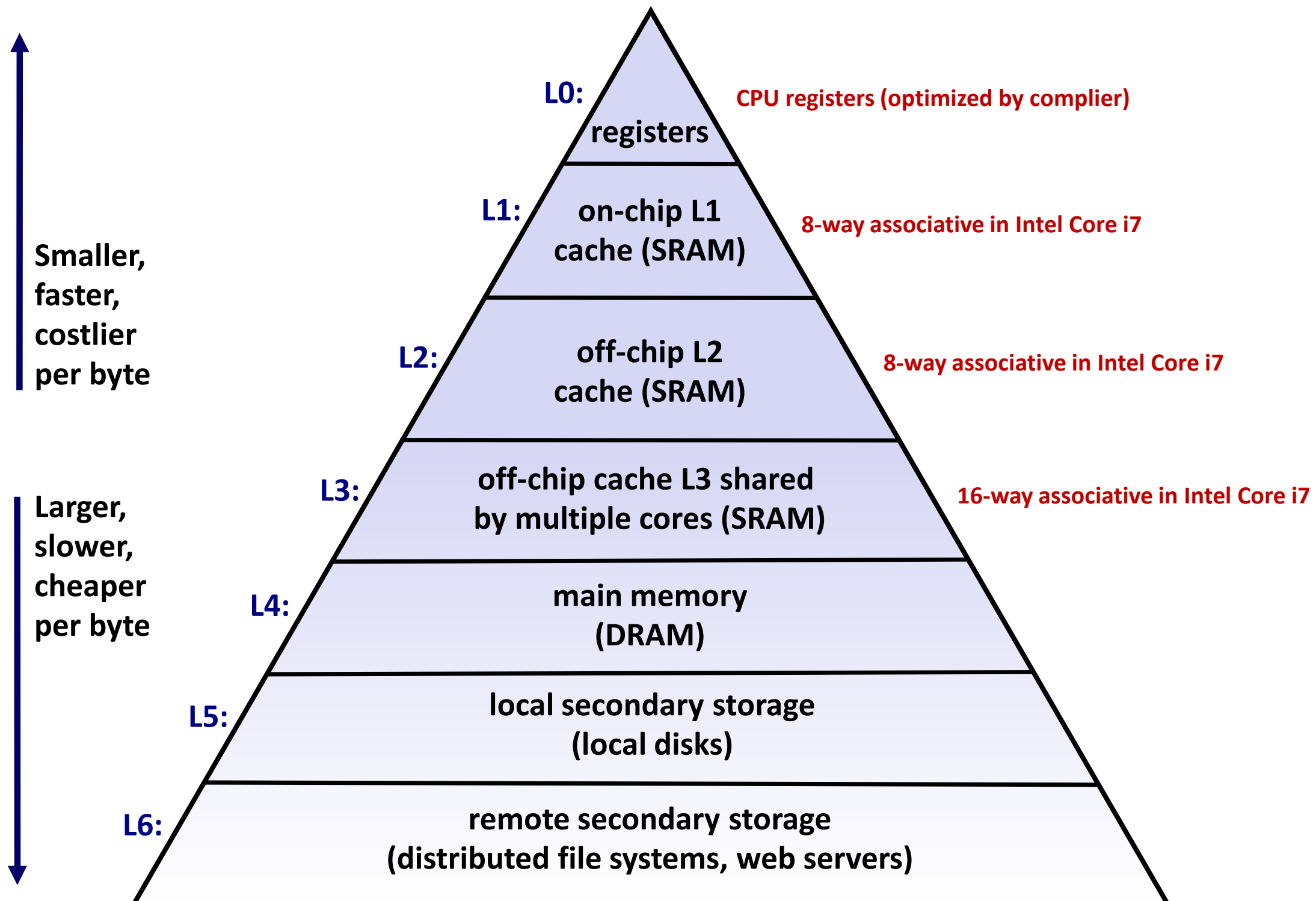
Core 2 Duo的层次存储系统

L1/L2 cache: 64 B blocks



存储器层次结构

Typical Memory Hierarchy (Intel Core i7)



本讲小结

■ 虚拟存储器

- 虚拟存储器基本概念
- 虚拟地址到物理地址的转换
- 虚拟存储器的页式管理