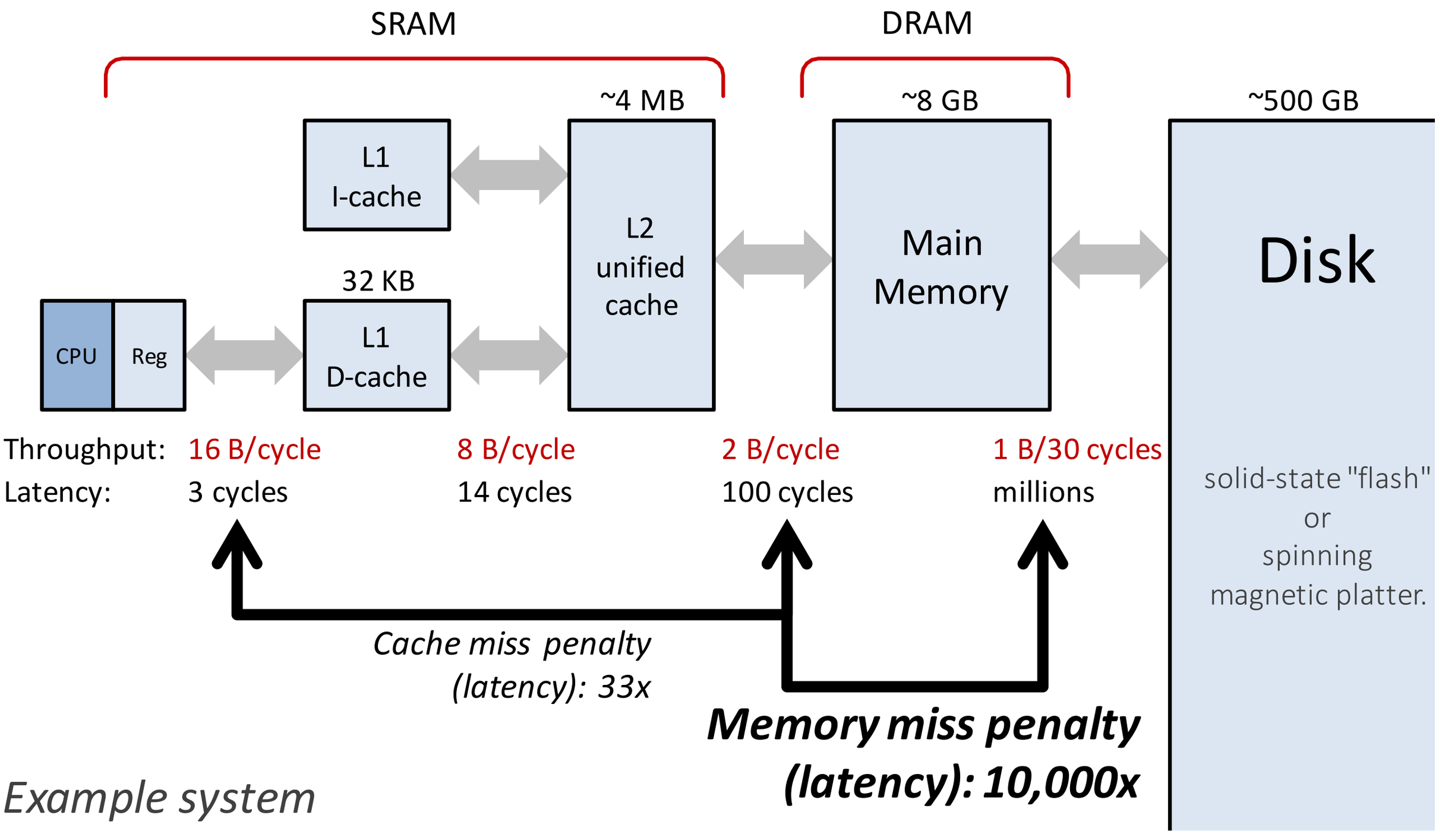
存储器层次结构

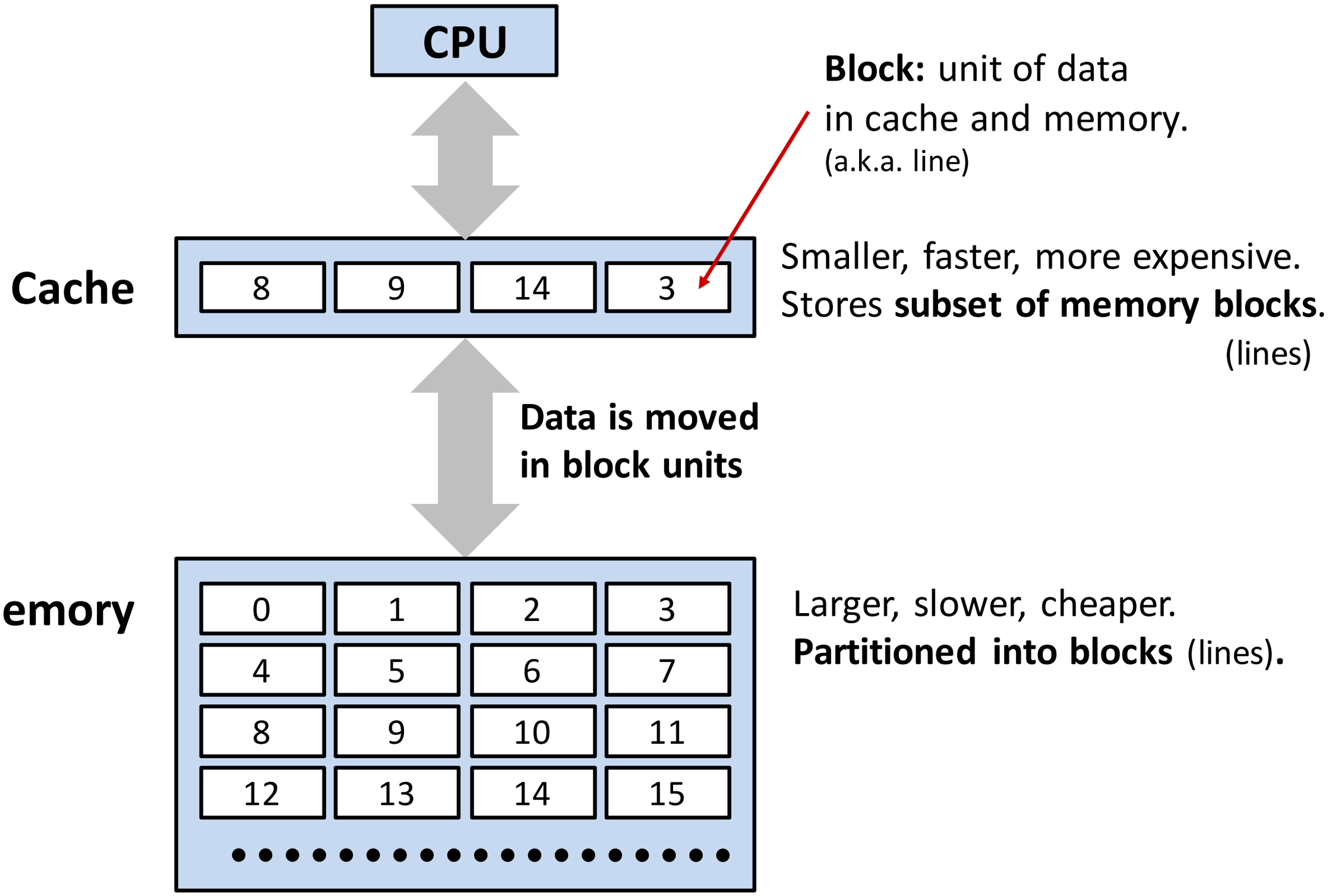
Part3：虚拟存储器

存储器层次结构

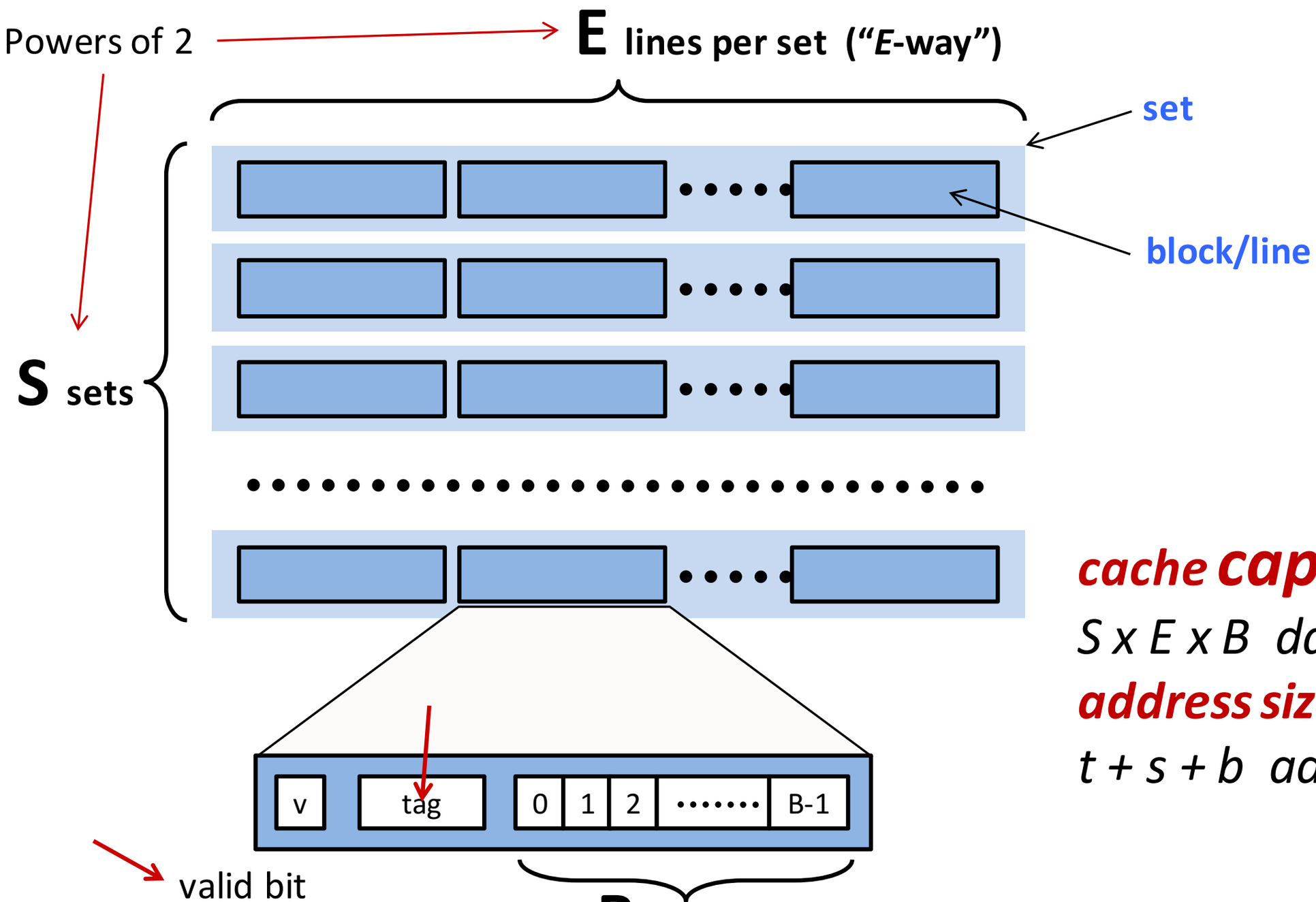


Example system

Cache原理


CPU
Block: unit of data in cache and memory. (a.k.a. line)
Cache
8
9
14
3
Smaller, faster, more expensive. Stores subset of memory blocks. (lines)
Data is moved in block units
Memory
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Larger, slower, cheaper. Partitioned into blocks (lines).

一般化的Cache组织形式



cache capacity:

S x E x B data bytes

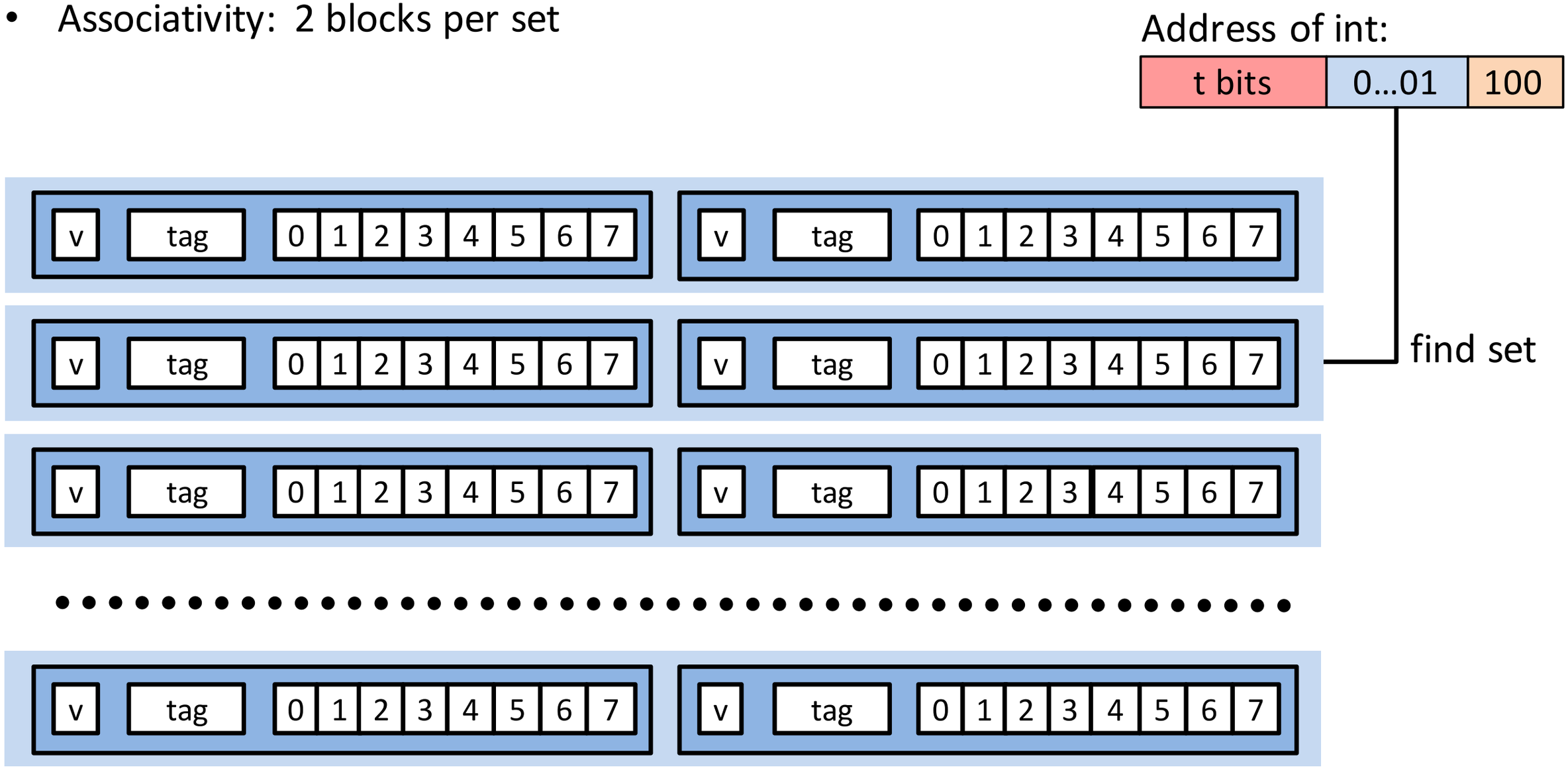
address size:

t + s + b address bits

读Cache：组相联映射

This cache:

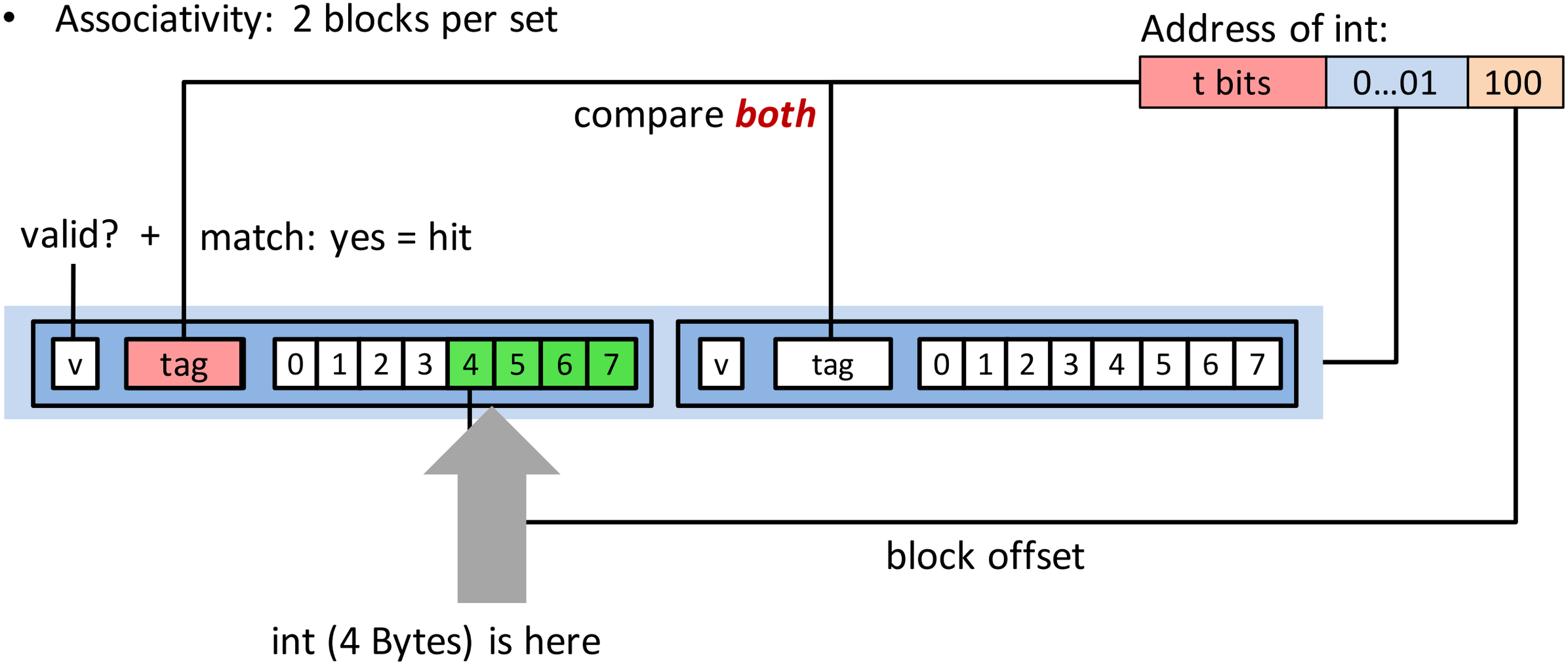
- Block size: 8 bytes
- Associativity: 2 blocks per set



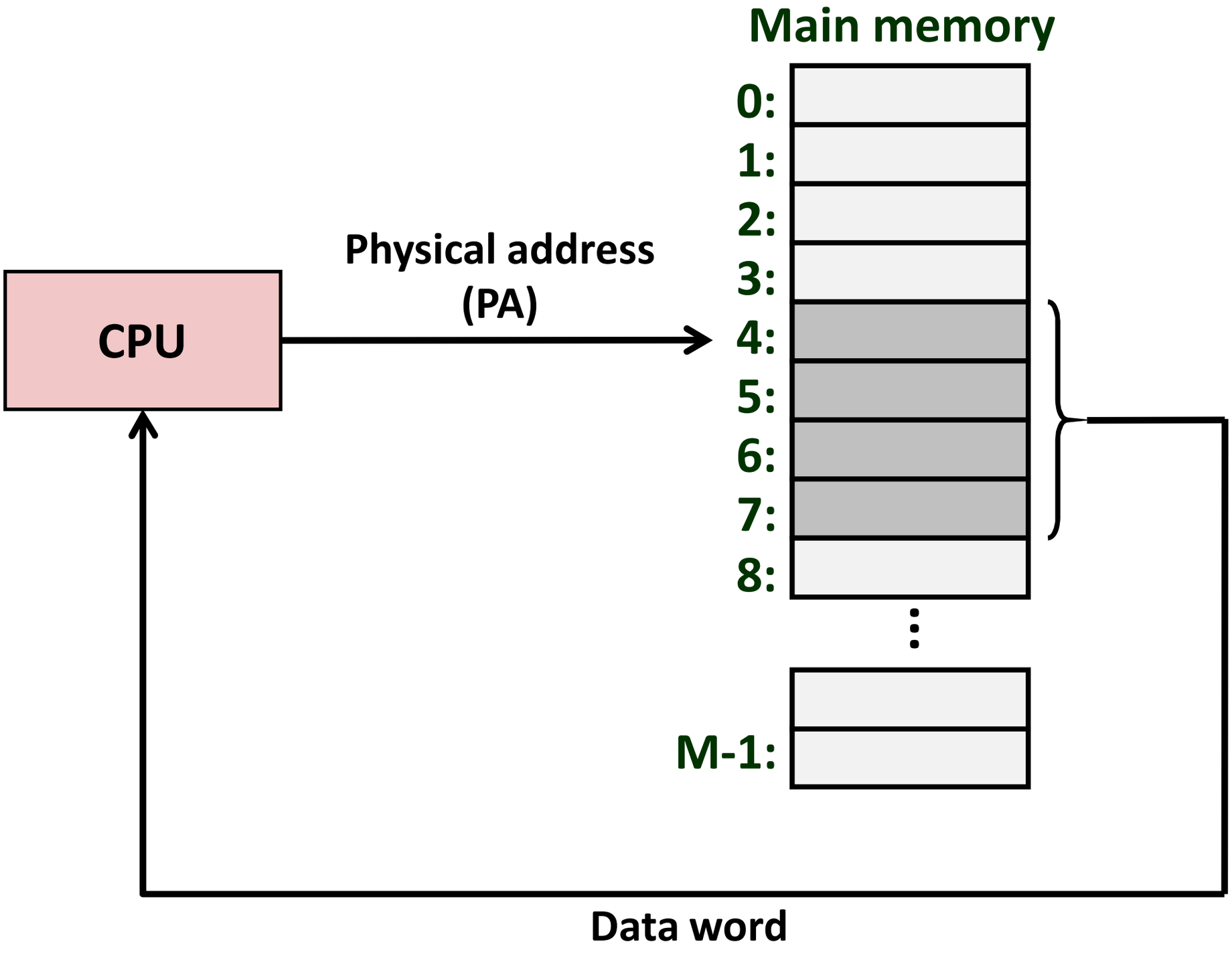
读Cache：组相联映射

This cache:

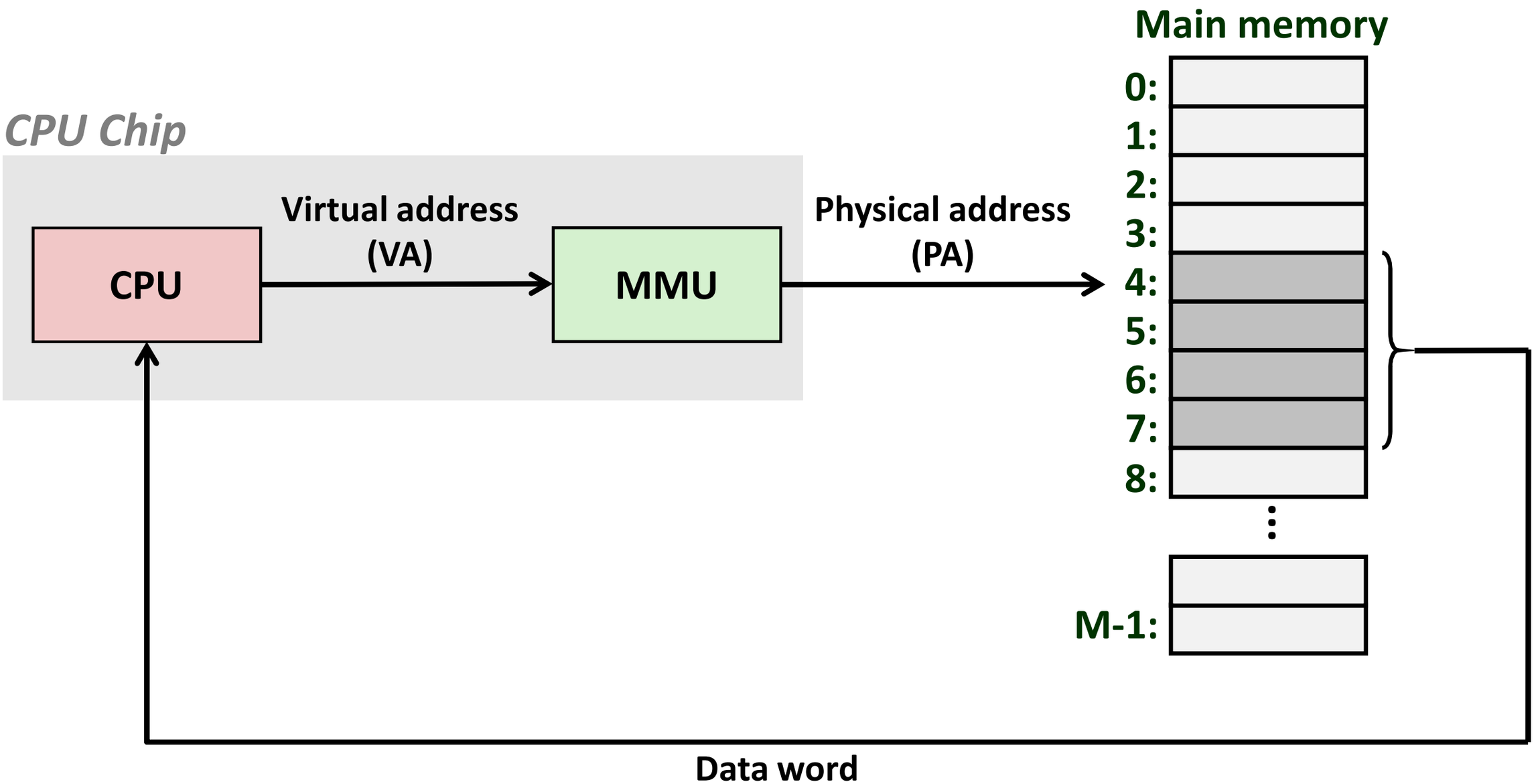
- Block size: 8 bytes
- Associativity: 2 blocks per set



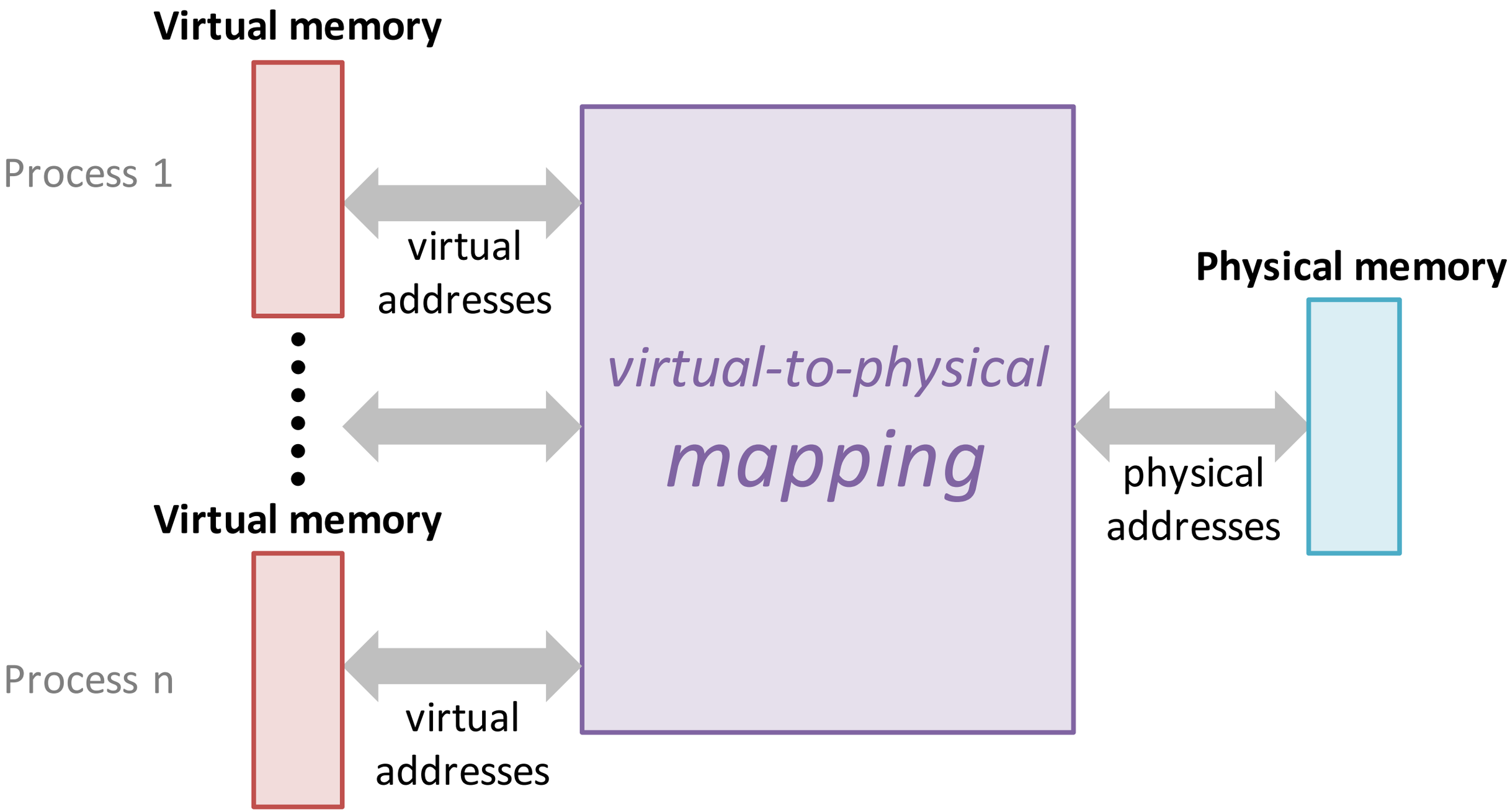
使用物理地址的计算机系统



使用虚拟存储器的计算机系统



虚拟存储器


Virtual memory
Process 1
virtual addresses
virtual-to-physical mapping
Physical memory
physical addresses
Virtual memory
Process n
virtual addresses

Process

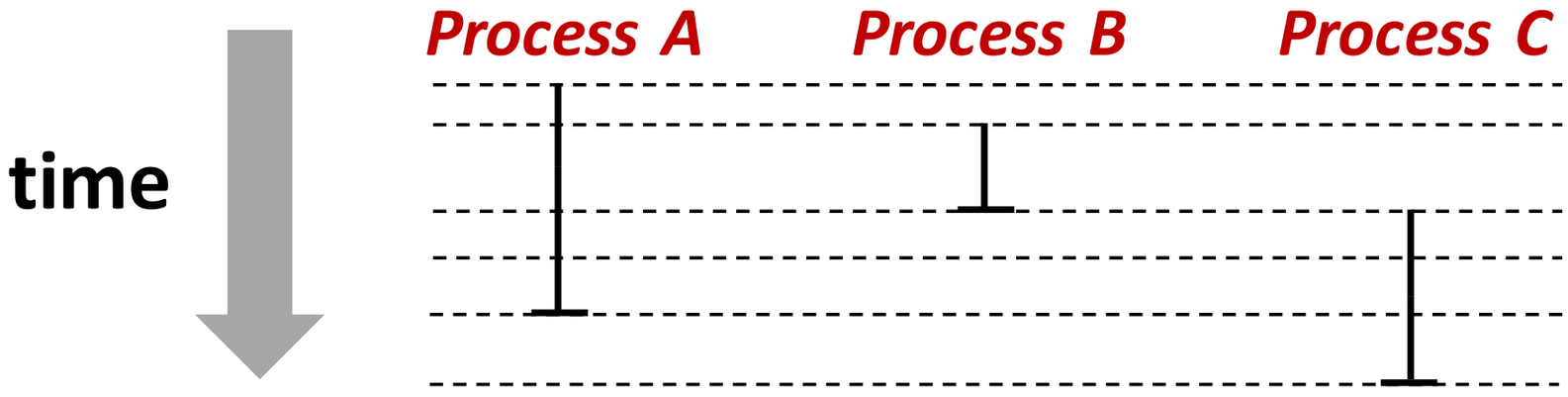
Program **= code (static)**

Process **= a running program instance (dynamic)**

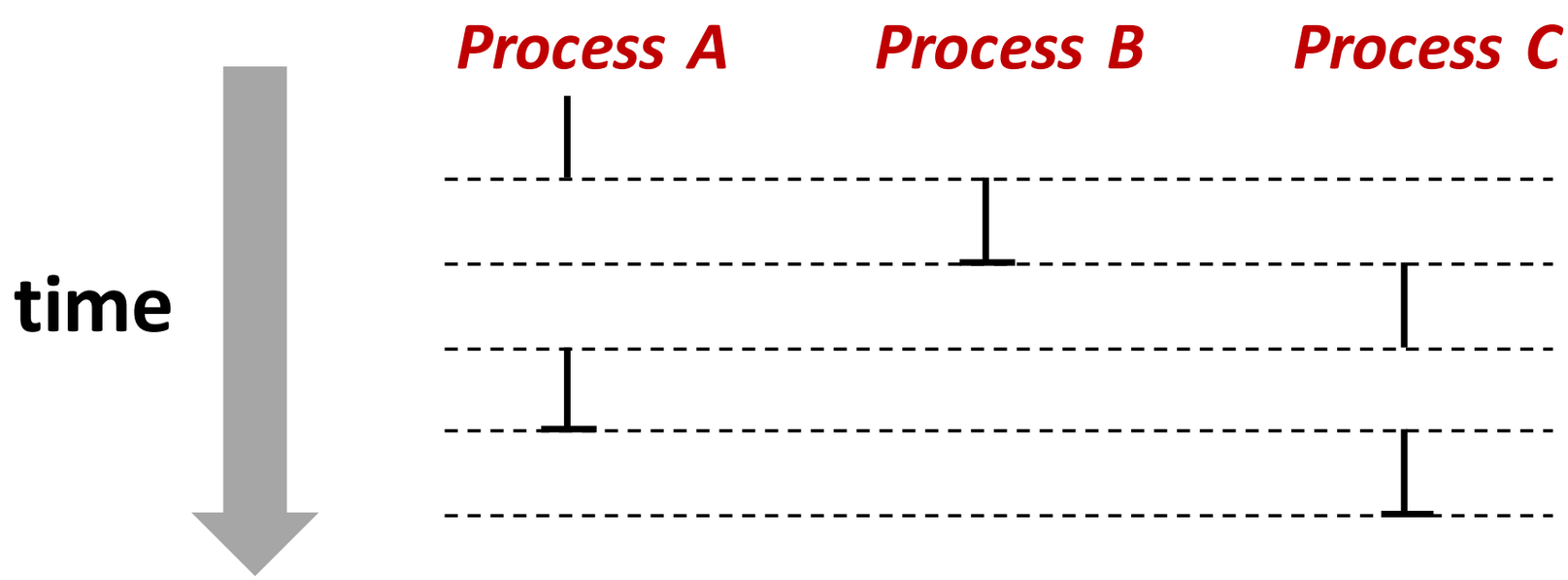
code + state

Process

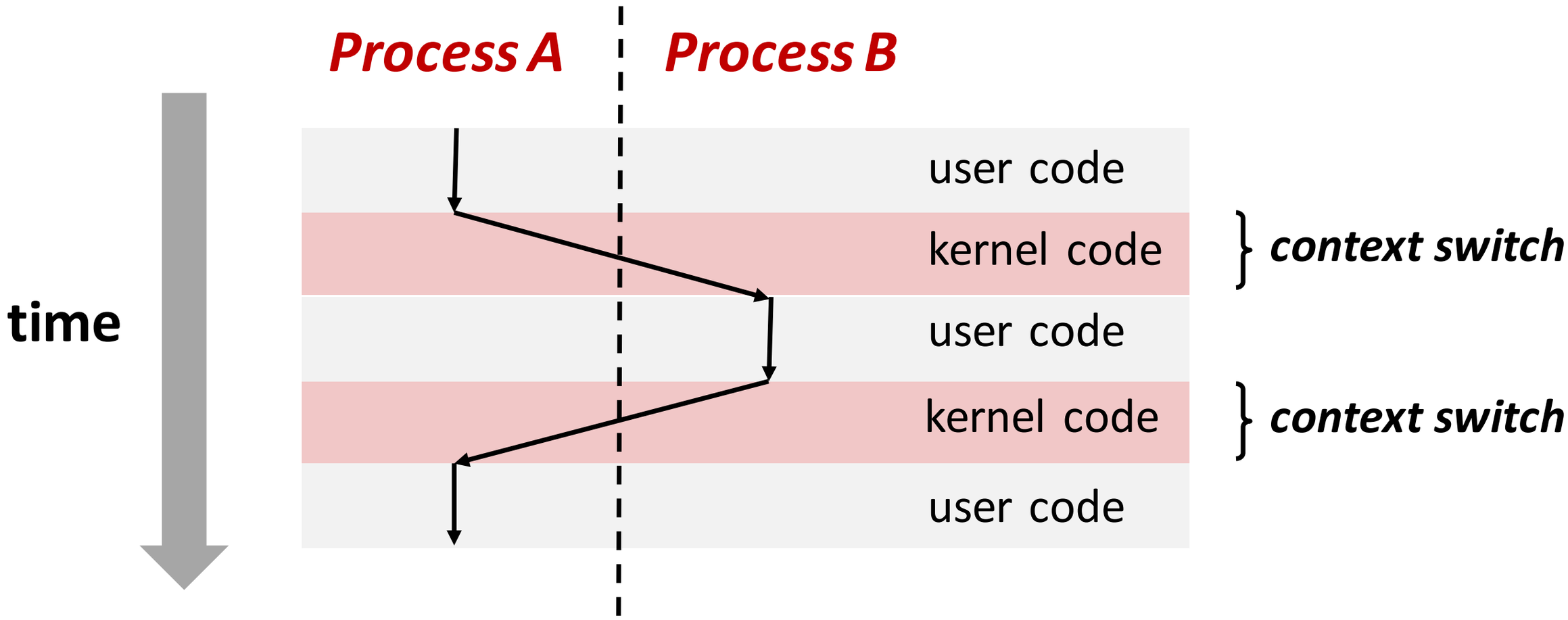
Abstraction: every process has full control over the CPU



Implementation: time-sharing



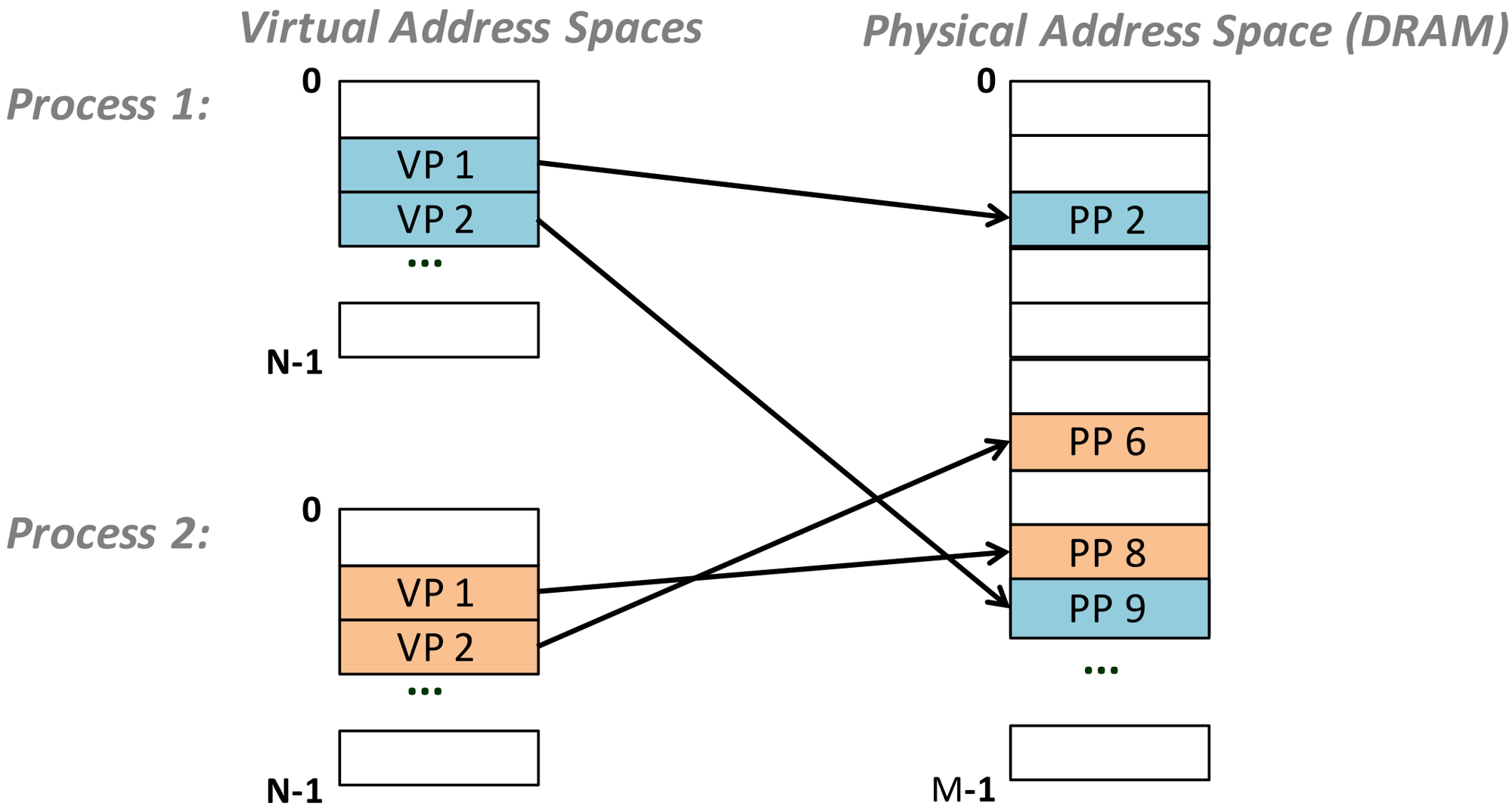
Process


Process A
Process B
time
user code
kernel code
user code
kernel code
user code
context switch
context switch

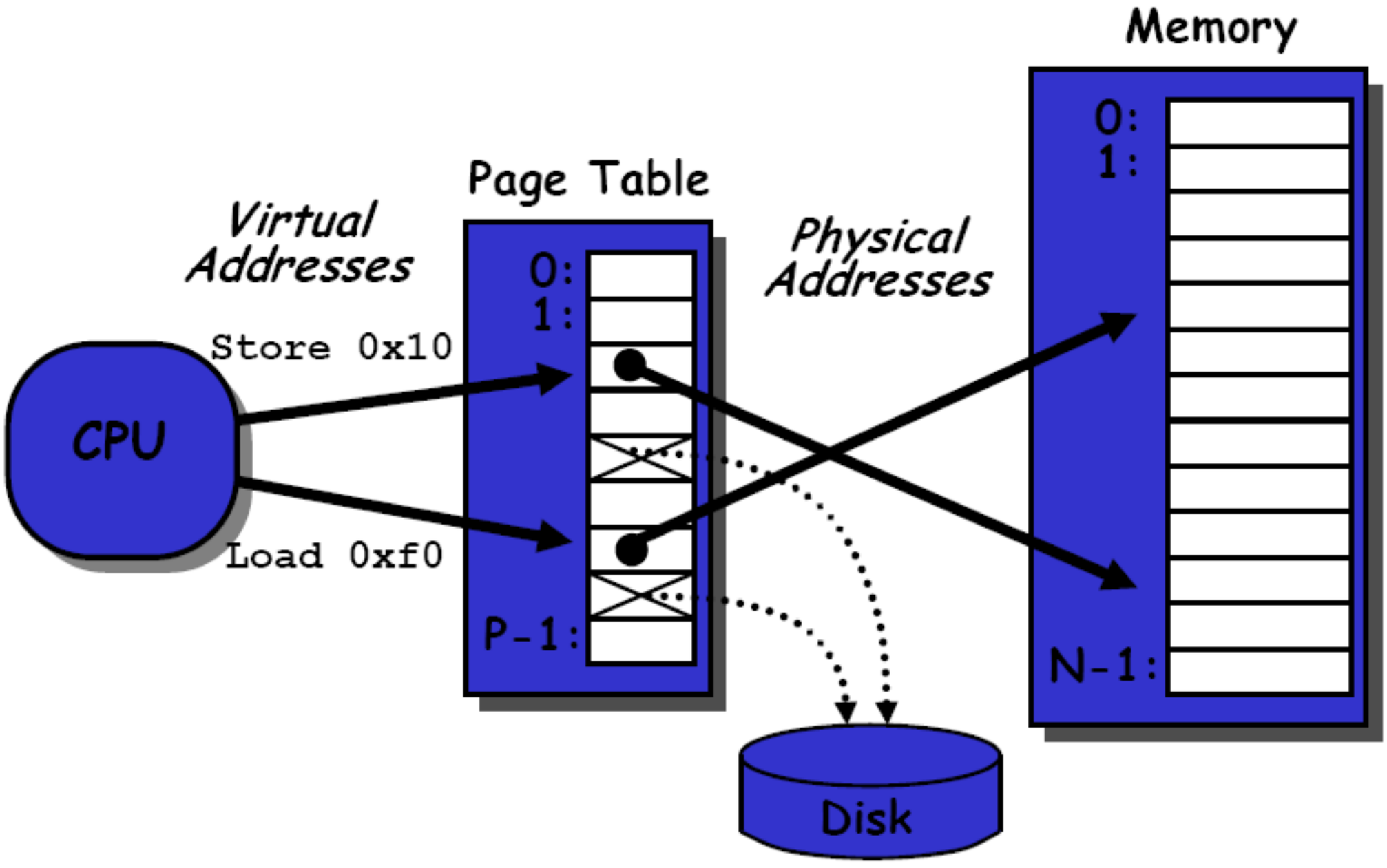
Process

Process needs private ***contiguous*** address space.

Storage of virtual pages in physical pages is **fully associative.**

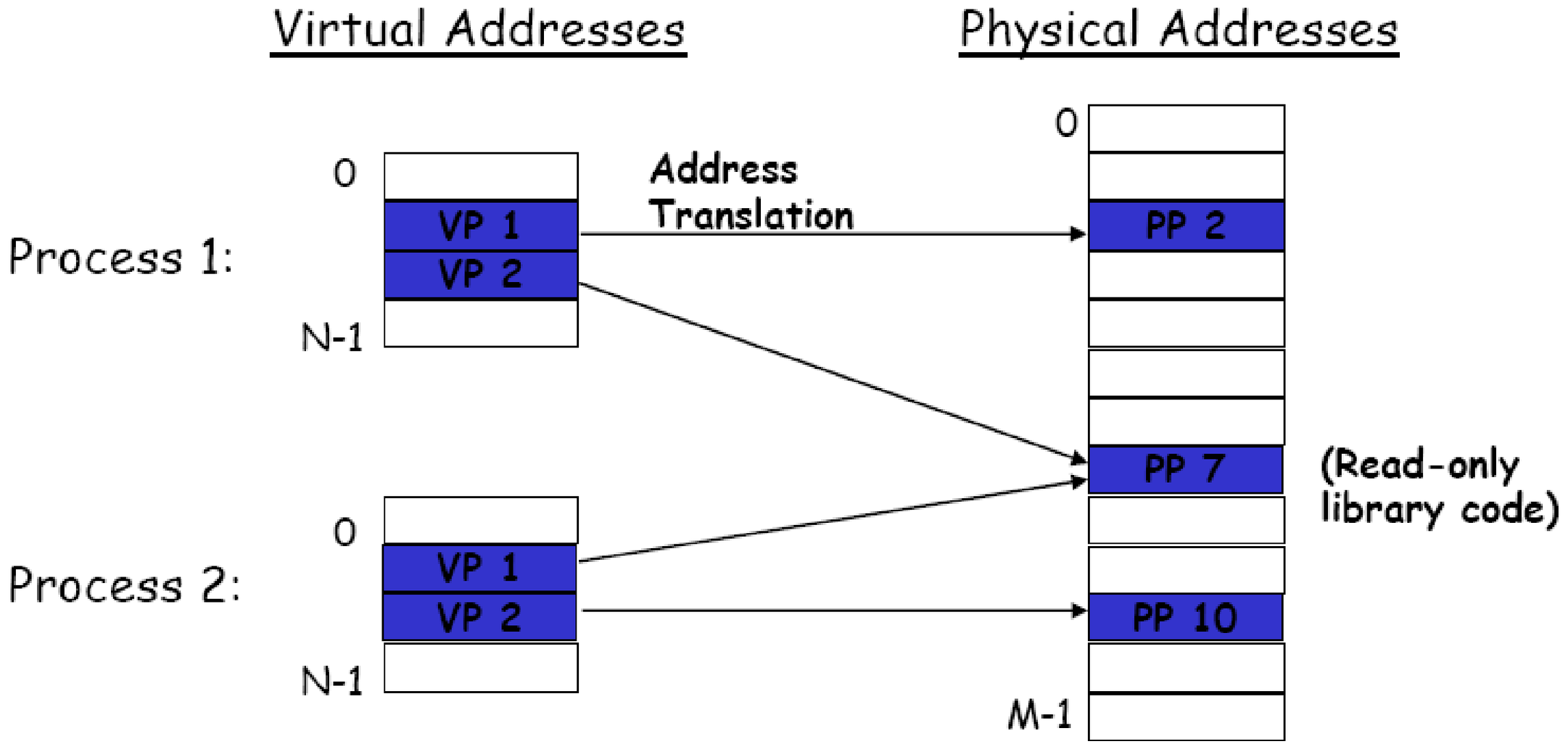


更大的逻辑地址空间



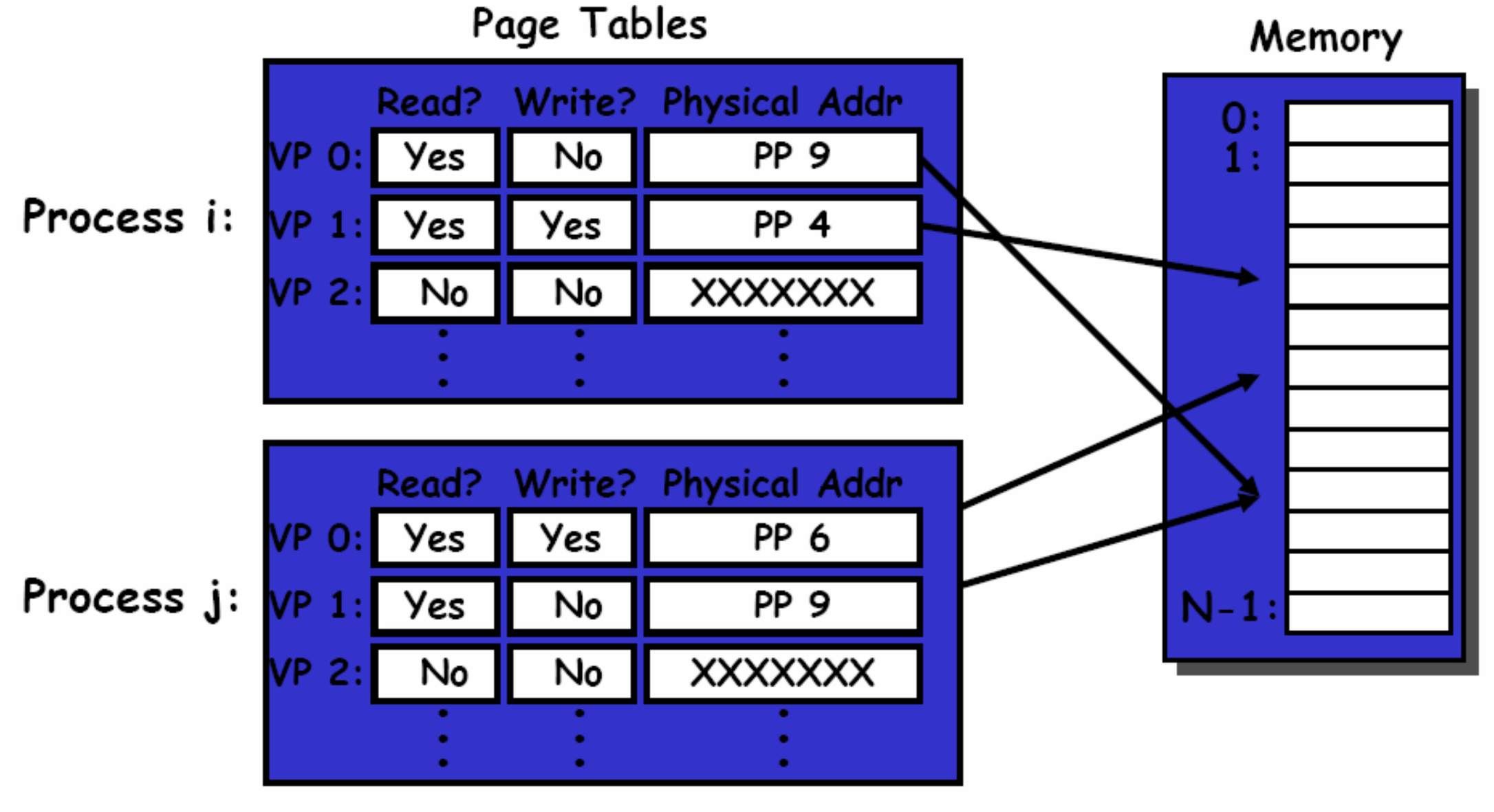
- 通过页表将虚地址转换为实地址

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

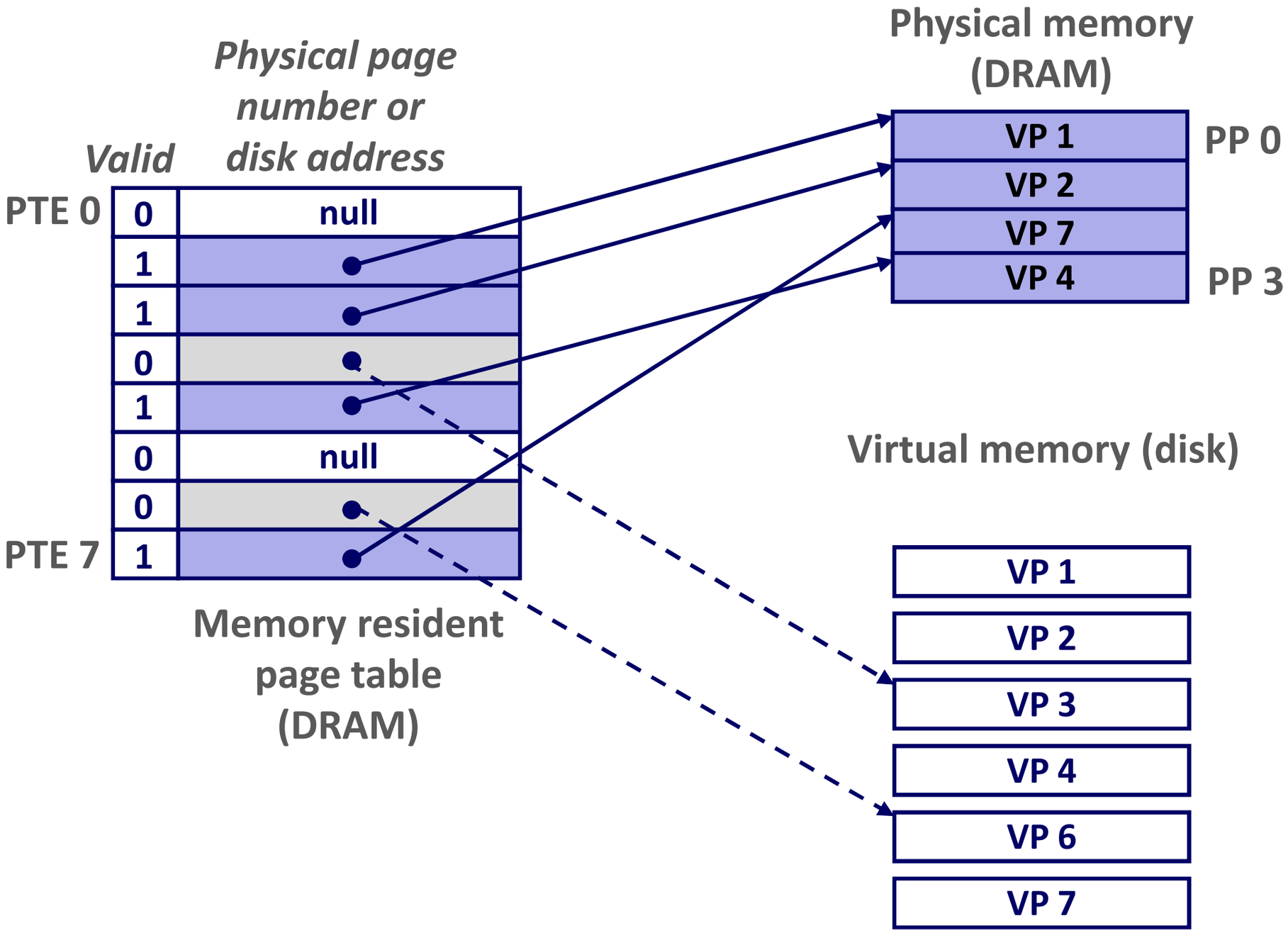
实现内存保护



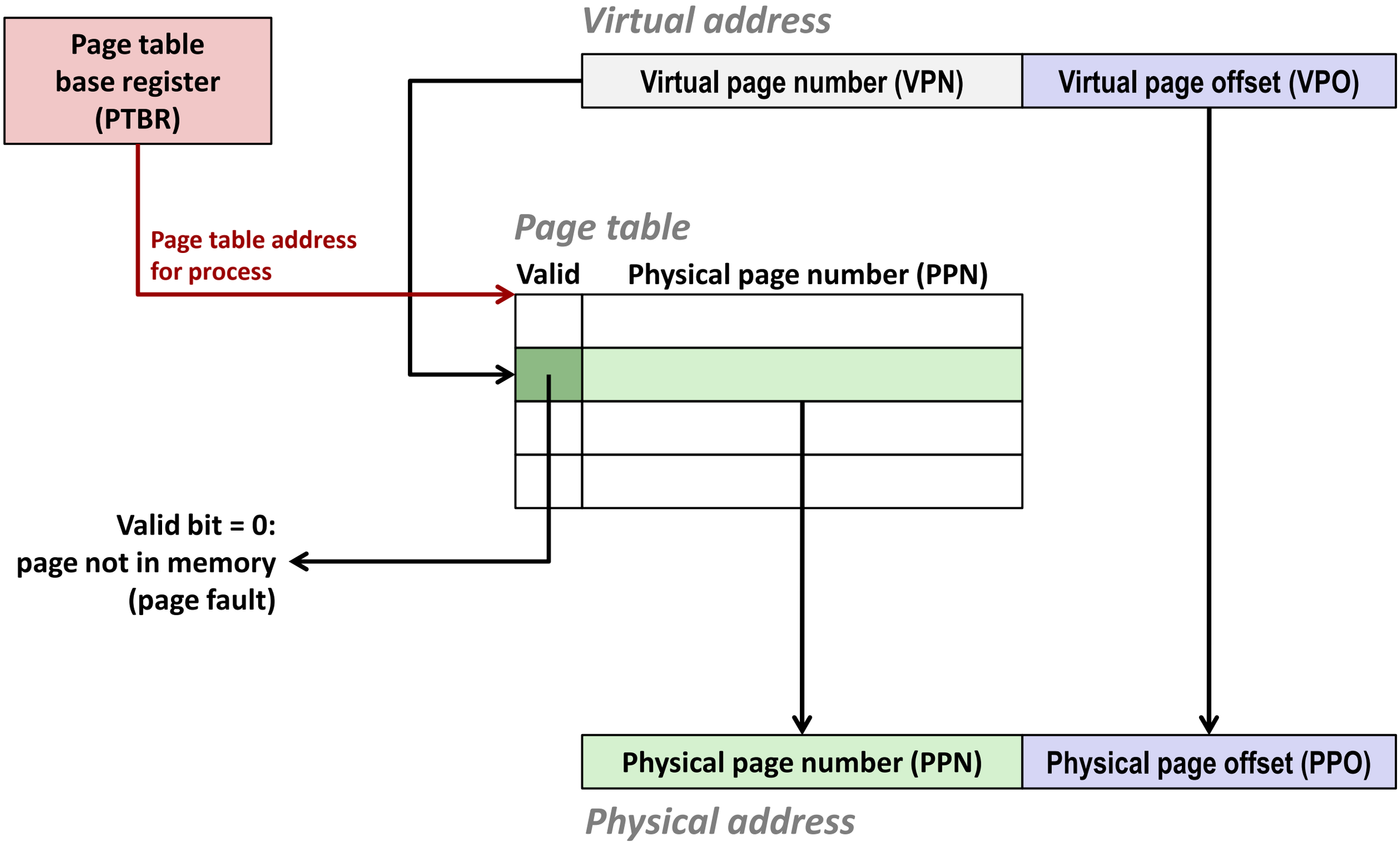
- 页表中存放有访问权限

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

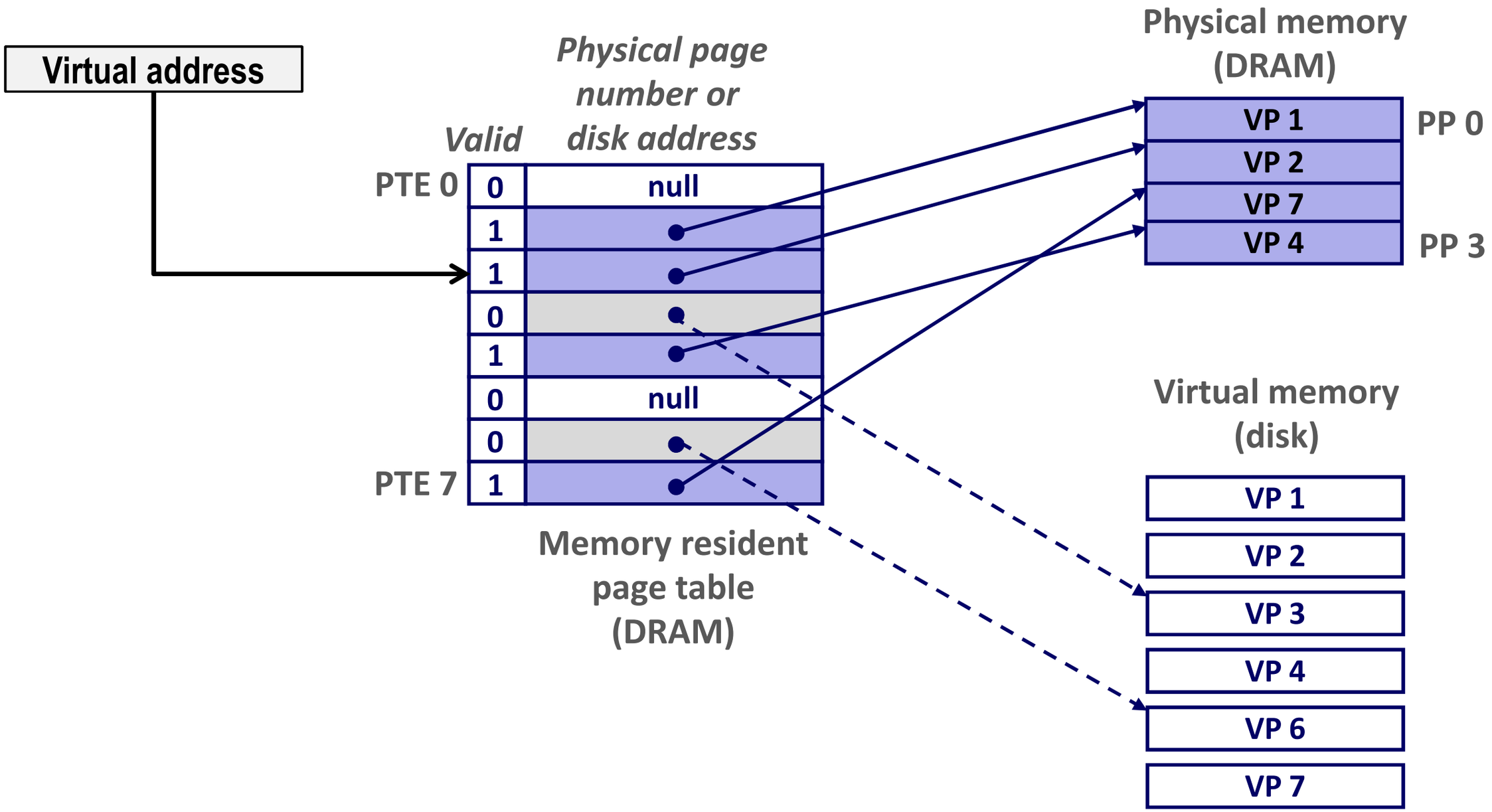


通过页表的地址转换


Page table base register (PTBR)
Virtual address
Virtual page number (VPN)
Virtual page offset (VPO)
Page table address for process
Page table
Valid
Physical page number (PPN)
Valid bit = 0: page not in memory (page fault)
Physical page number (PPN)
Physical page offset (PPO)
Physical address

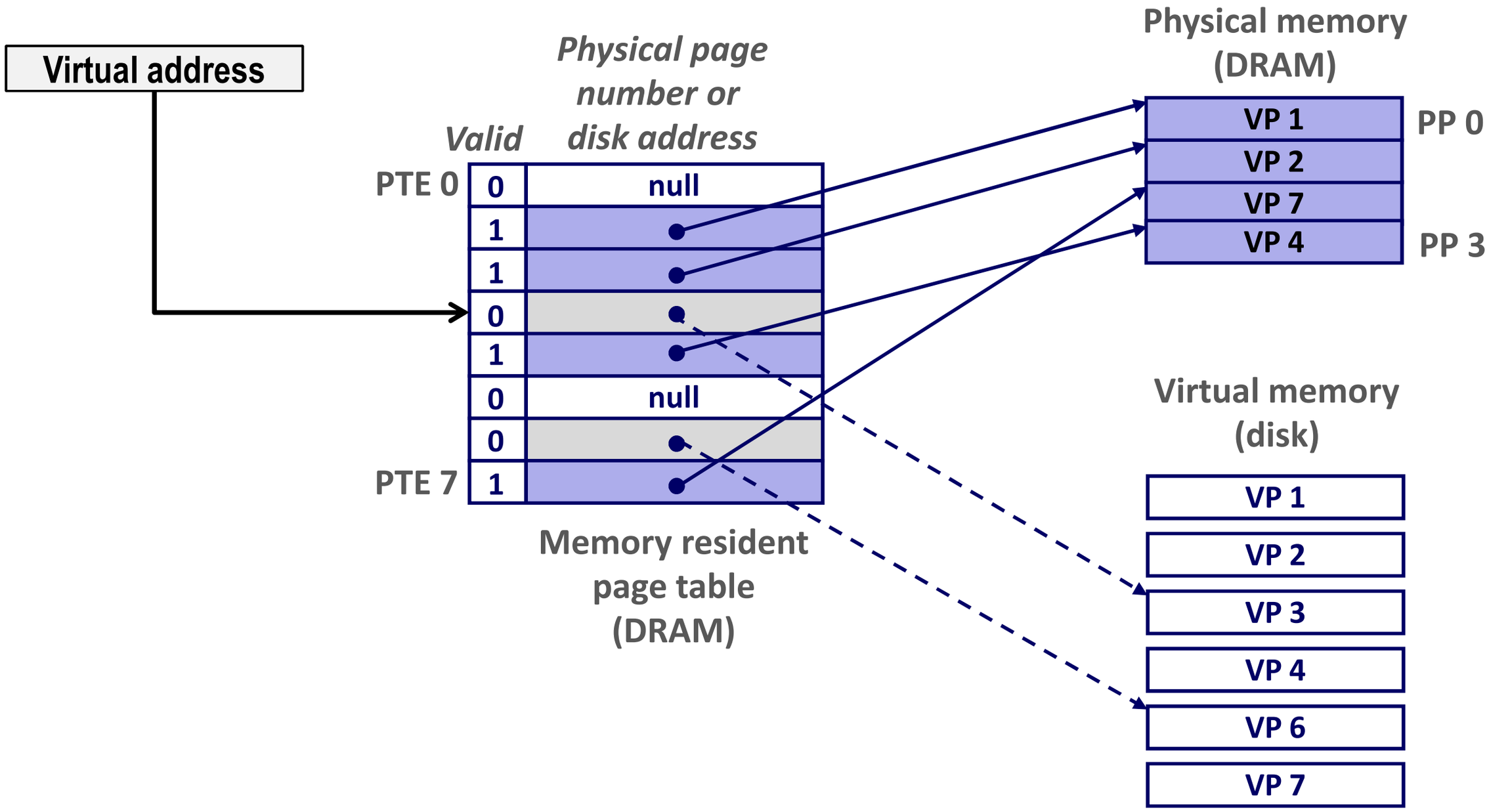
页命中

- 需要访问的虚地址中的字在主存中（有效位为1）



缺页

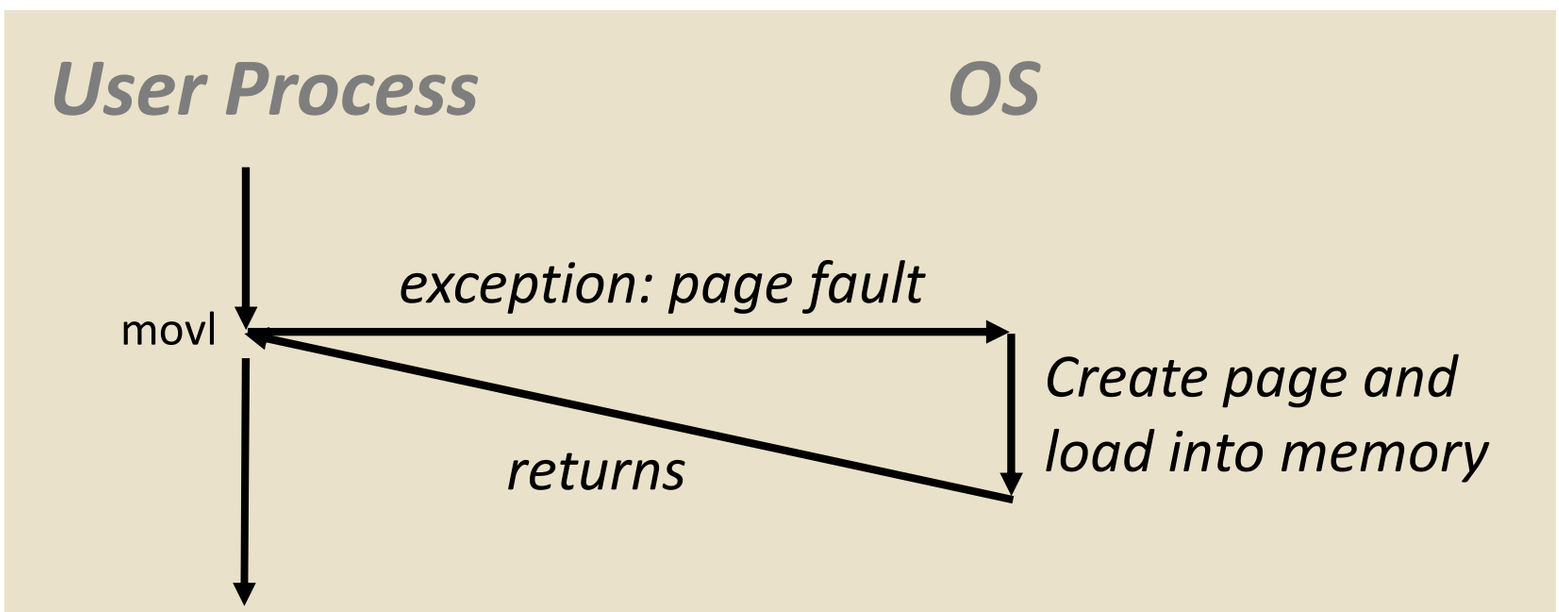
- 需要访问的虚地址中的字在不主存中（有效位为0）



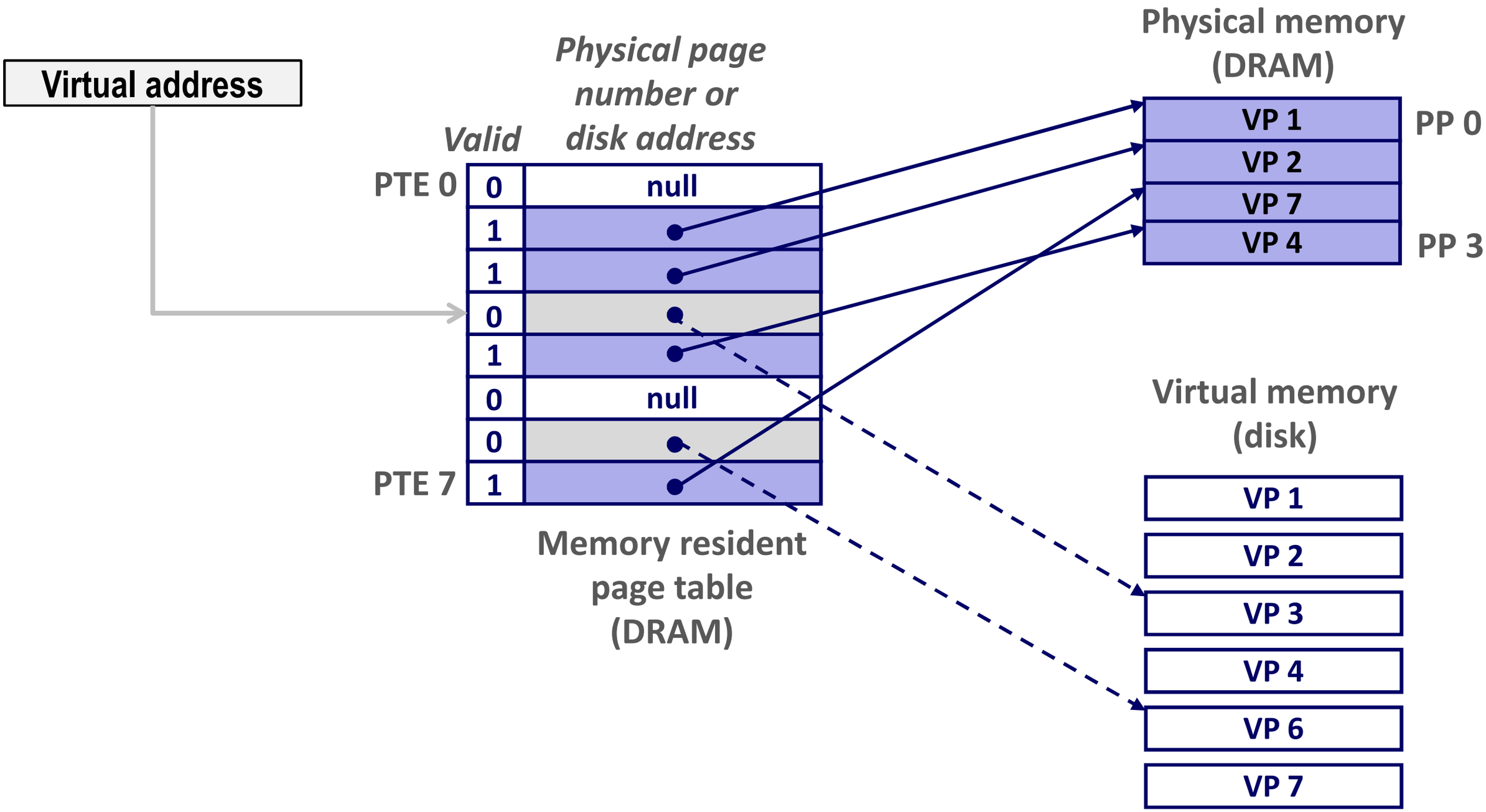
发生缺页

- 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

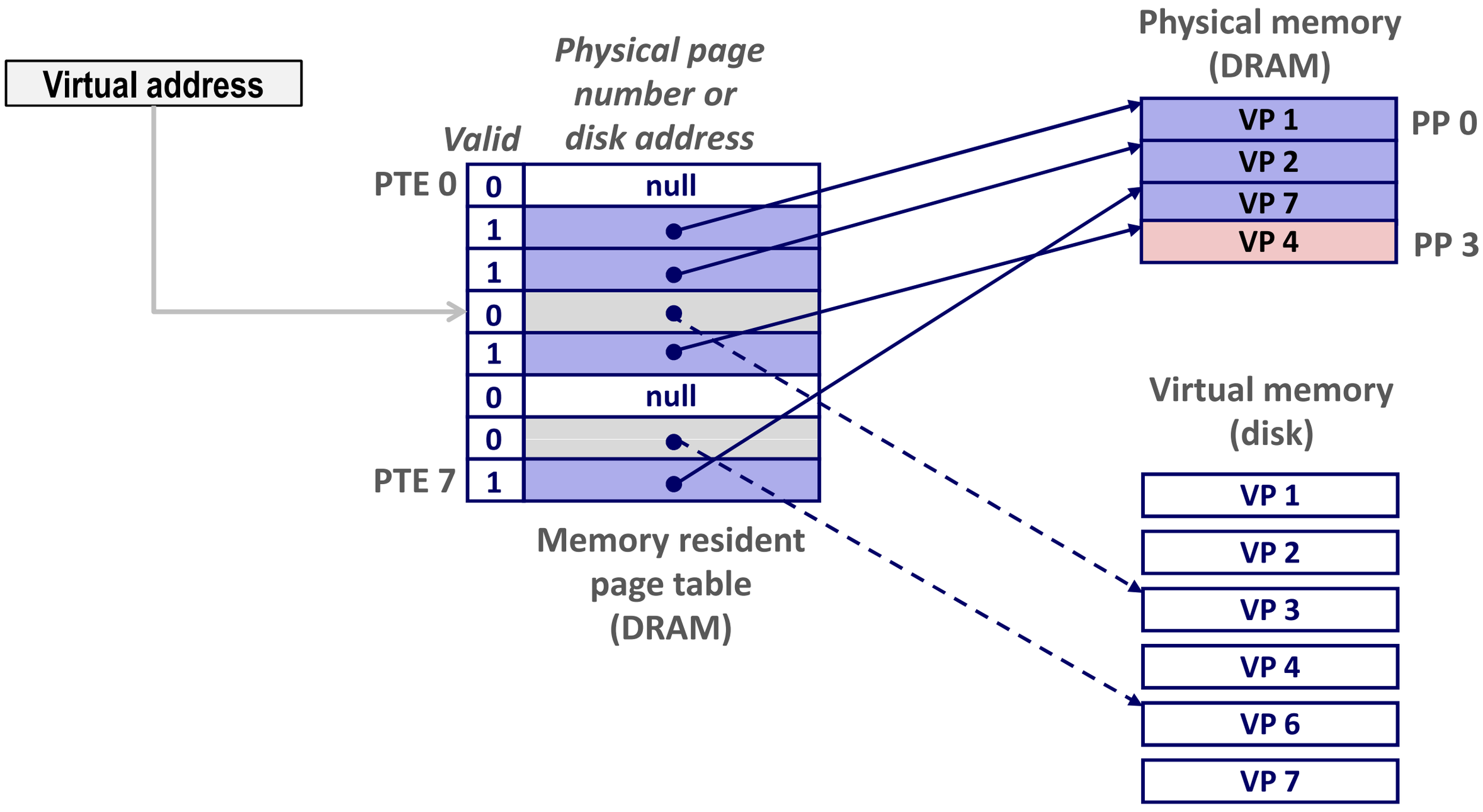
```
int a[1000];
main ()
{
    a[500] = 13;
}
```



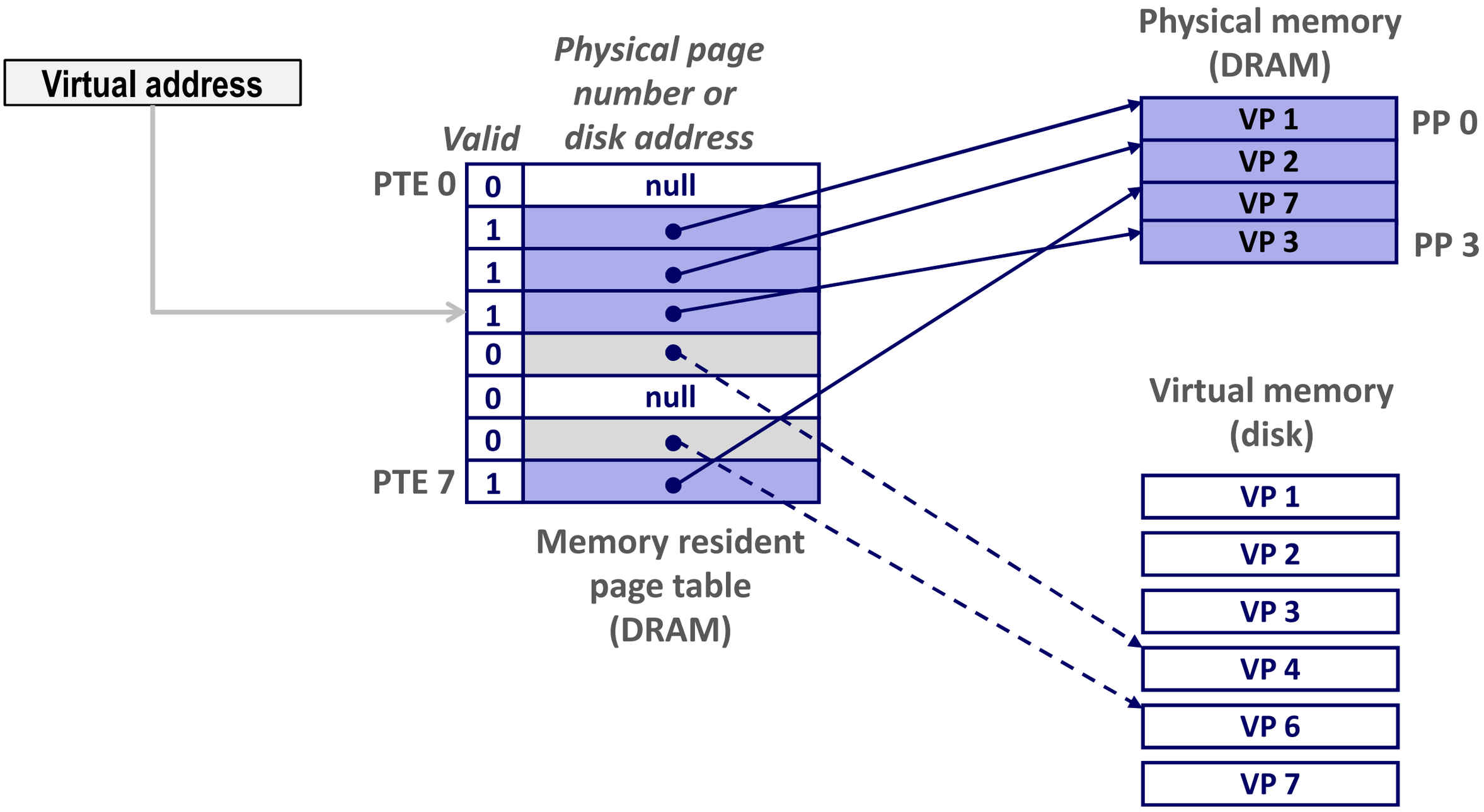
缺页处理


Virtual address
Physical page number or disk address
Valid
PTE 0
0
null
1
1
0
1
0
null
0
PTE 7
1
Memory resident page table (DRAM)
Physical memory (DRAM)
VP 1
VP 2
VP 7
VP 4
PP 0
PP 3
Virtual memory (disk)
VP 1
VP 2
VP 3
VP 4
VP 6
VP 7

缺页处理


Virtual address
Physical page number or disk address
Valid
PTE 0
0
null
1
1
0
1
0
null
0
PTE 7
1
Memory resident page table (DRAM)
Physical memory (DRAM)
VP 1
VP 2
VP 7
VP 4
PP 0
PP 3
Virtual memory (disk)
VP 1
VP 2
VP 3
VP 4
VP 6
VP 7

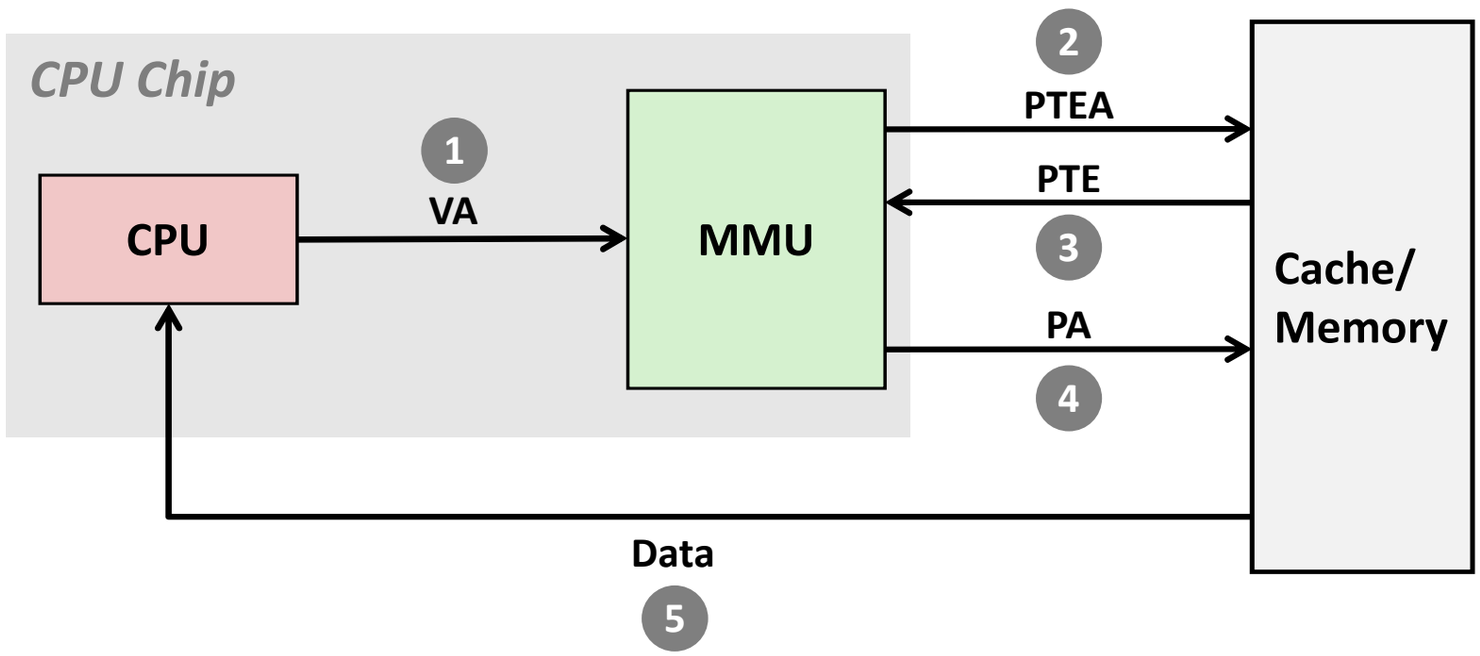
缺页处理


Virtual address
Physical page number or disk address
Valid
PTE 0
0
null
1
1
1
0
0
null
0
PTE 7
1
Memory resident page table (DRAM)
Physical memory (DRAM)
VP 1
VP 2
VP 7
VP 3
PP 0
PP 3
Virtual memory (disk)
VP 1
VP 2
VP 3
VP 4
VP 6
VP 7

页面替换算法

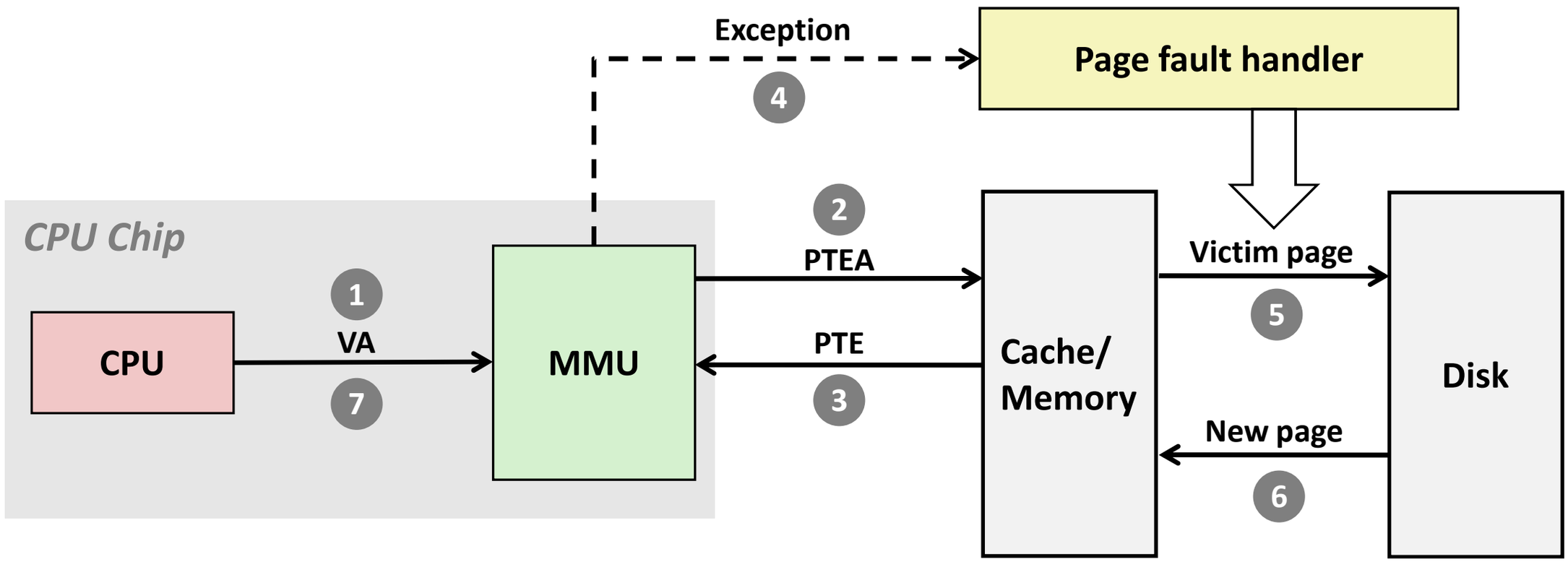
- 最近最少使用(LRU)
  - 将页帧按照最近最多使用到最近最少使用进行排序,再次访问一个页帧时,将该页帧移到表头,替换时将表尾的页帧换出

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

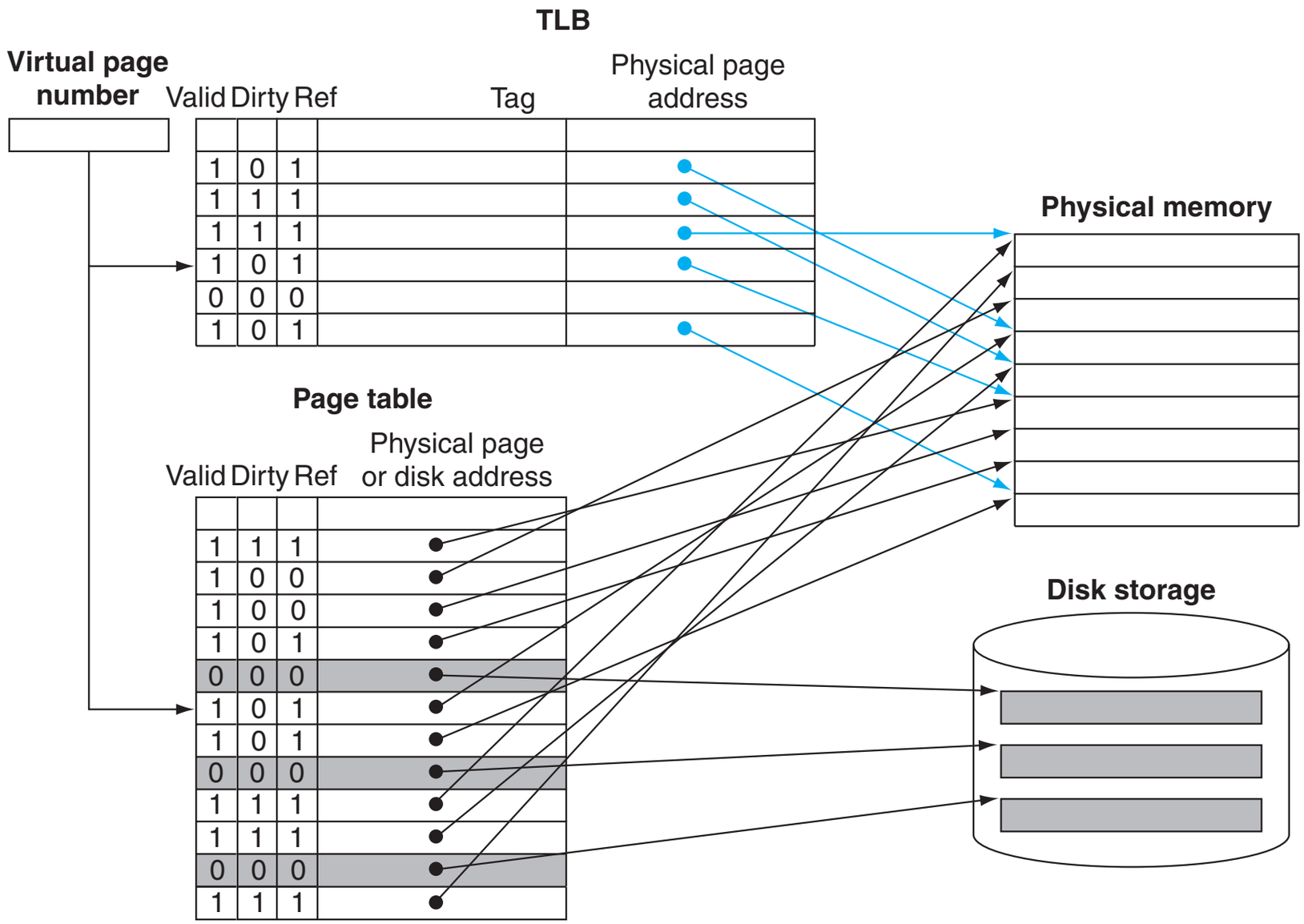


- (1)CPU向MMU发送虚拟地址
- (2-3)MMU从页表中获取页表项
- (4)有效位为0，MMU触发缺页异常
- (5)选择被替换的页
- (6)在主存中插入新的页
- (7)返回原来的进程，重新执行刚才发生异常的操作

地址转换的效率问题

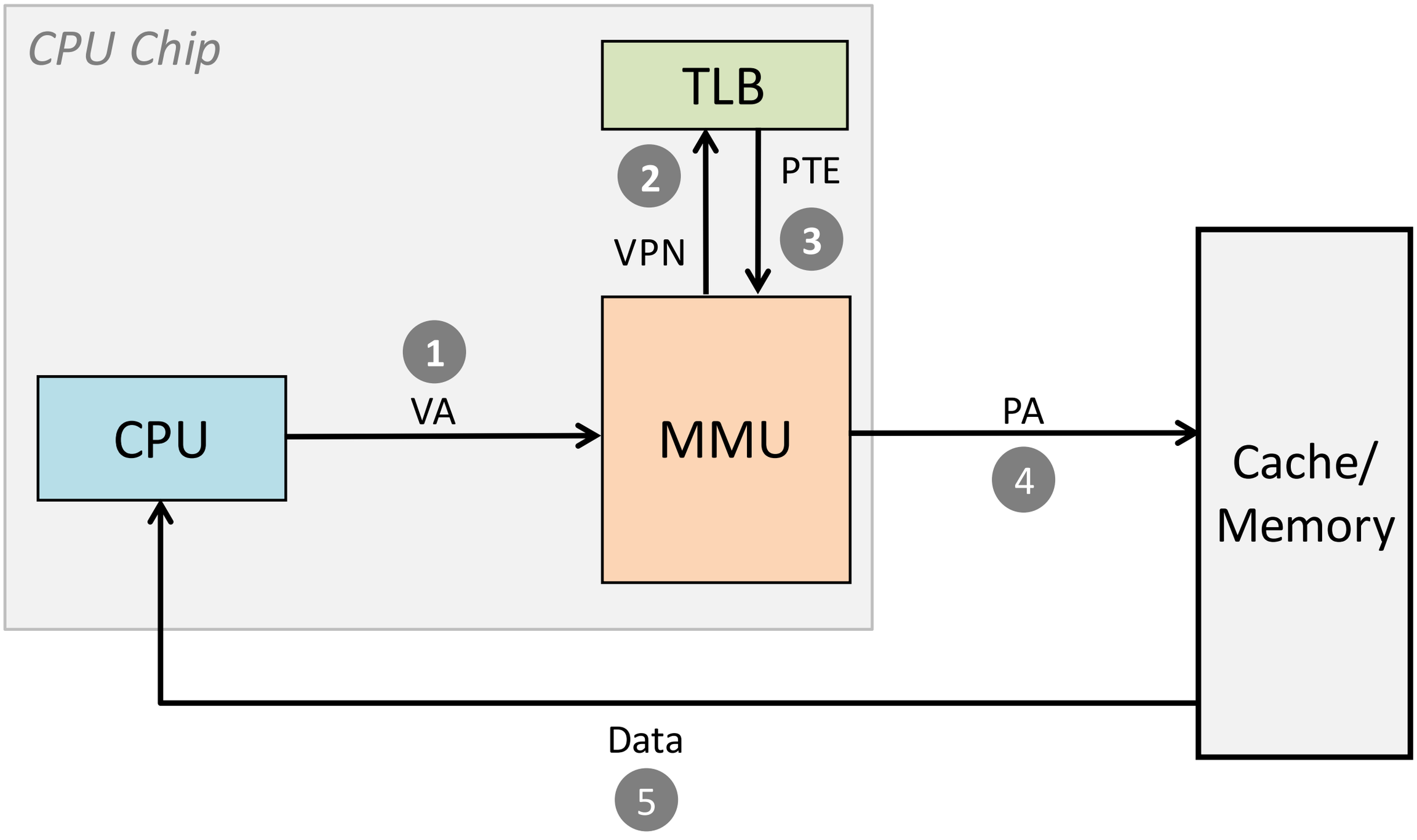
- 由于页表存放在主存中，因此程序每次访存至少需要两次
  - 一次访存获取物理地址
  - 第二次访存获取数据
- 提高地址转换速度的方法
  - 利用页表的访问局部性，即当一个转换的虚拟页号被使用时，它可能在不久的将来再次被使用；
  - 解决方案：转换旁路缓冲TLB
    - 包含在处理器中的一个特殊的cache，用以追踪最近使用过的地址变换

转换旁路缓冲TLB

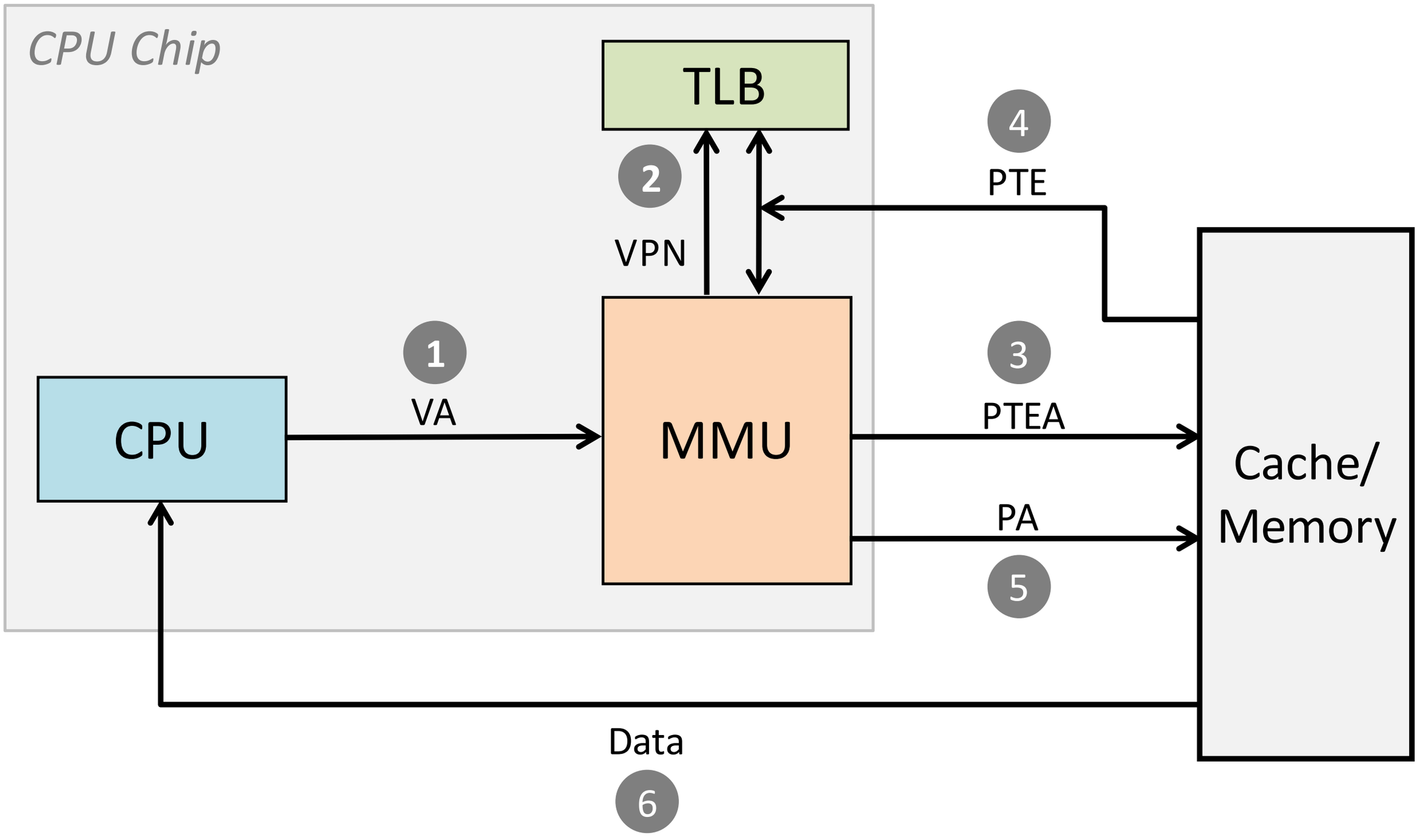


- 两种缺失
  - TLB缺失：TLB中没有访问的虚页号，但是该页在主存中
  - 缺页：访问的页不在主存中

TLB 命中

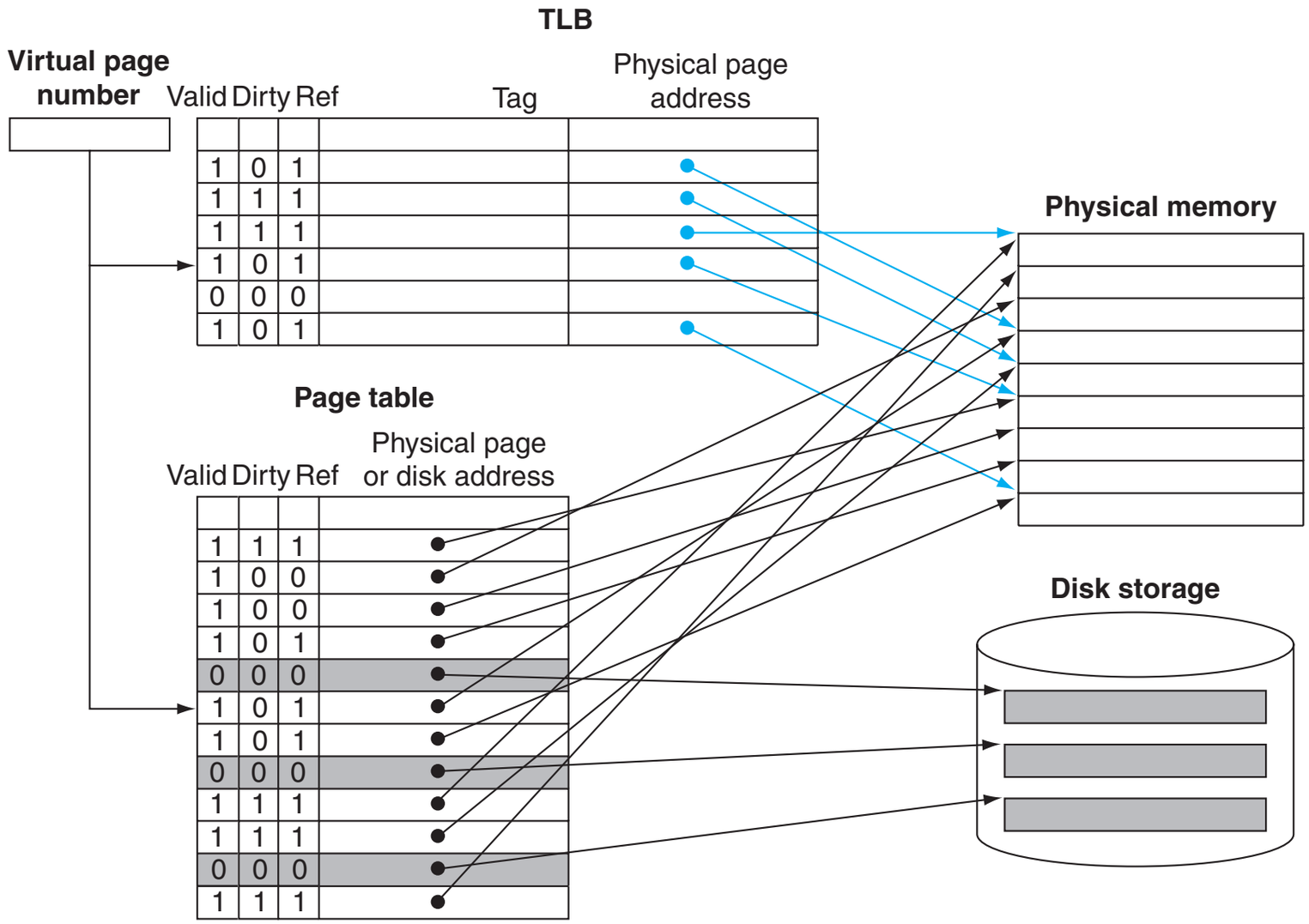


TLB缺失


CPU Chip
TLB
2
VPN
4
PTE
1
VA
CPU
MMU
3
PTEA
PA
5
Cache/
Memory
Data
6

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



存储器系统举例

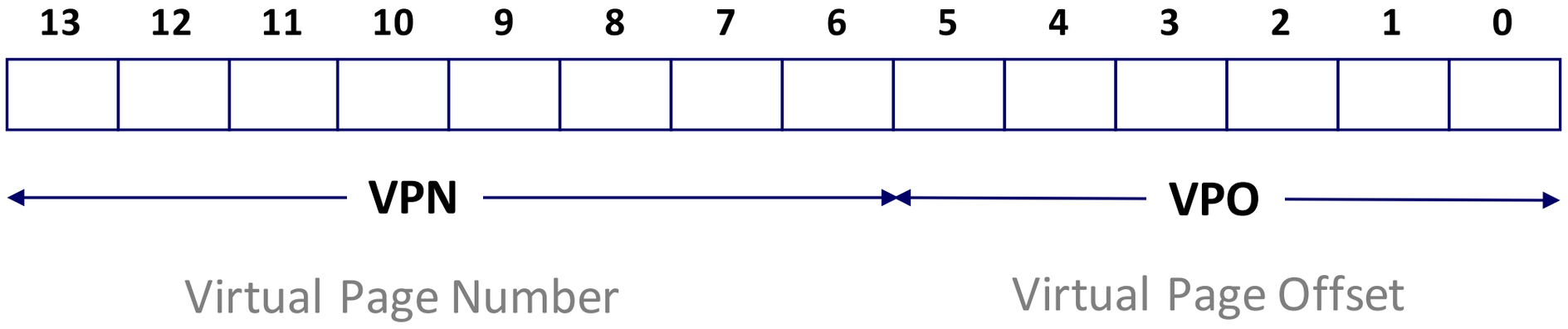
Addressing

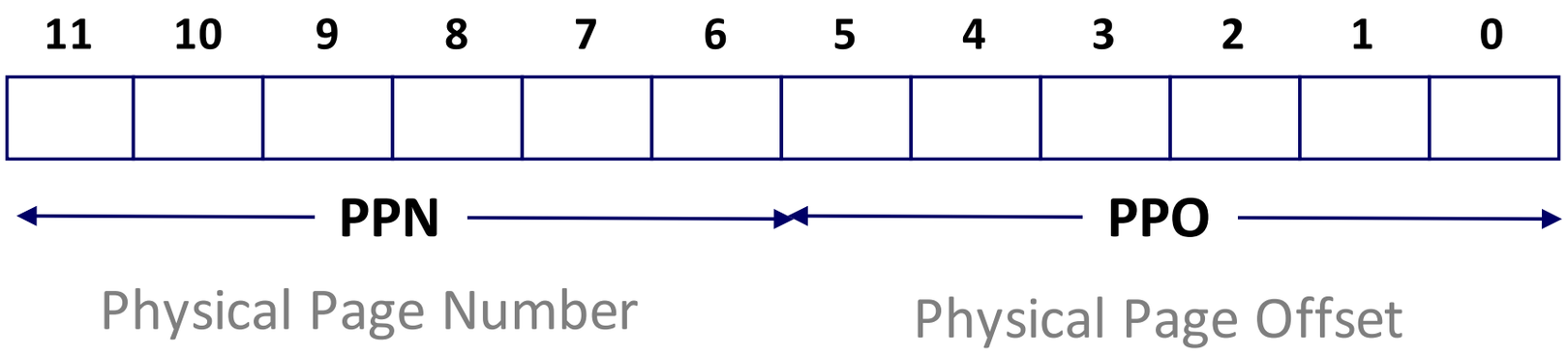
14-bit virtual addresses

12-bit physical address

Page size = 64 bytes

Simulate accessing these virtual addresses on the system: **`0x03D4, 0x0B8F, 0x0020`**





页表

Only showing first 16 entries (out of 256 = 2^8)

virtual page #___ TLB index___ TLB tag ____ TLB Hit? __ Page Fault? __ physical page #: ____

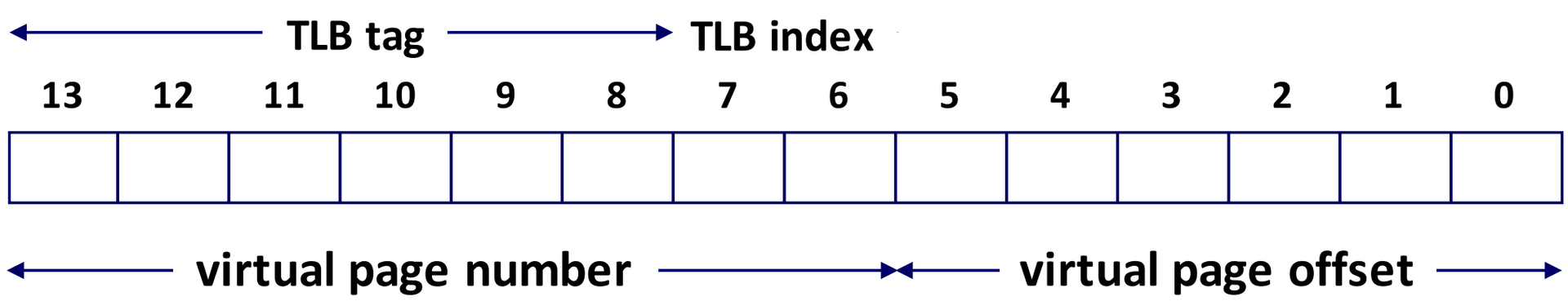
VPN	*PPN*	*Valid*
00	28	1
01	–	0
02	33	1
03	02	1
04	–	0
05	16	1
06	–	0
07	–	0

VPN	*PPN*	*Valid*
08	13	1
09	17	1
0A	09	1
0B	–	0
0C	–	0
0D	2D	1
0E	11	1
0F	0D	1

TLB

16 entries

4-way associative



virtual page #___ TLB index___ TLB tag ____ TLB Hit? __ Page Fault? __ physical page #: ____

Set	*Tag*	*PPN*	*Valid*	*Tag*	*PPN*	*Valid*	*Tag*	*PPN*	*Valid*	*Tag*	*PPN*	*Valid*
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

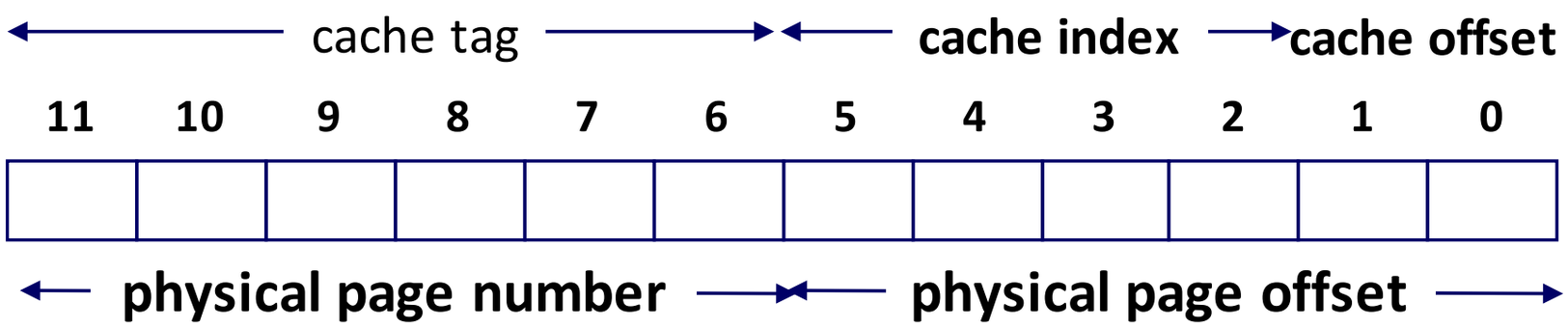
Cache

16 lines

4-byte block size

Physically addressed

Direct mapped



cache offset___ cache index___ cache tag____ Hit? __ Byte: ____

Idx	*Tag*	*Valid*	*B0*	*B1*	*B2*	*B3*
0	19	1	99	11	23	11
1	15	0	–	–	–	–
2	1B	1	00	02	04	08
3	36	0	–	–	–	–
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	–	–	–	–
7	16	1	11	C2	DF	03

Idx	*Tag*	*Valid*	*B0*	*B1*	*B2*	*B3*
8	24	1	3A	00	51	89
9	2D	0	–	–	–	–
A	2D	1	93	15	DA	3B
B	0B	0	–	–	–	–
C	12	0	–	–	–	–
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	–	–	–	–

虚拟存储器与Cache比较

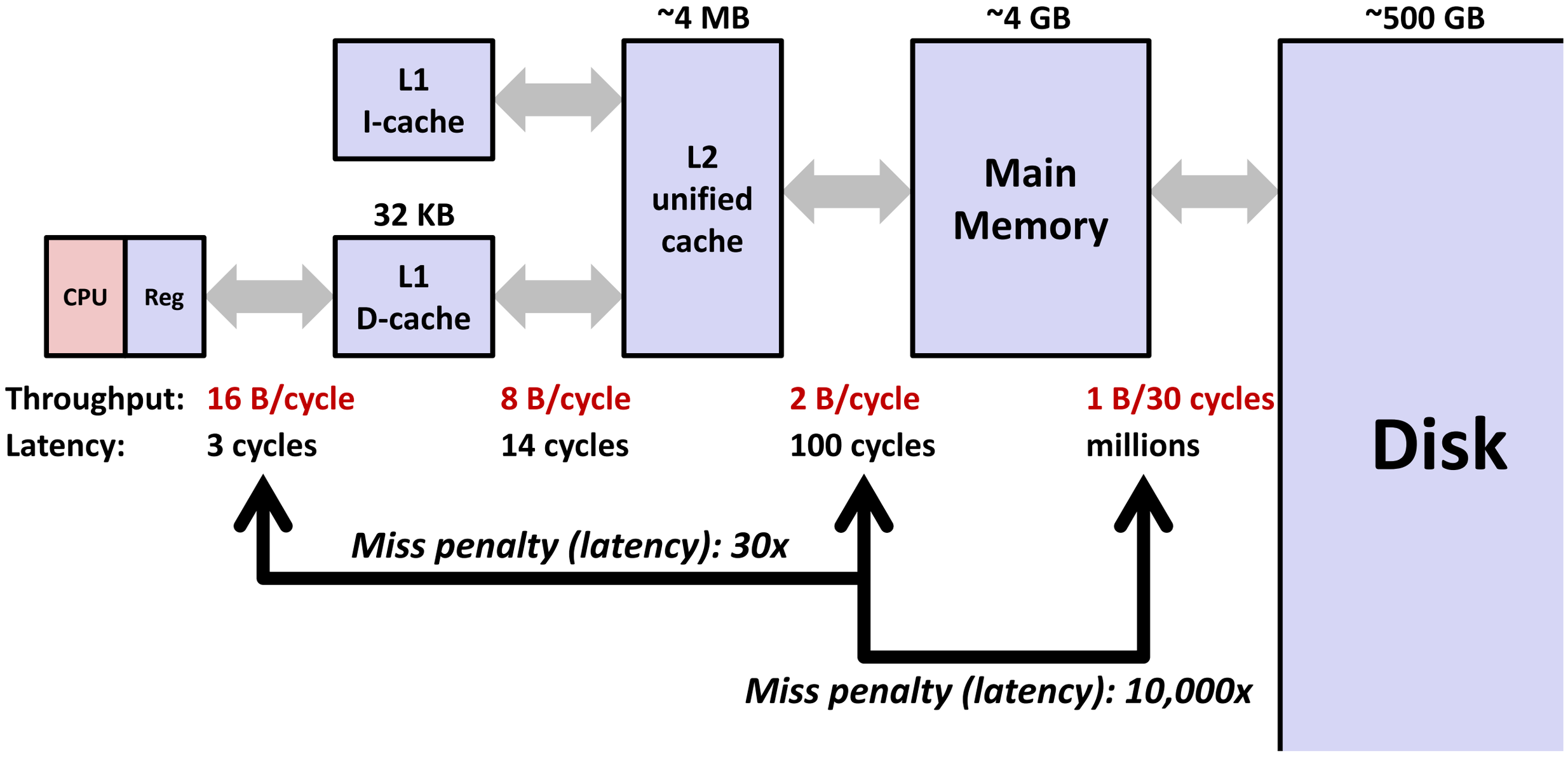
- 虚存
  - “主存——辅存层次”,主要目的是解决存储容量的问题。
  - 单位时间内数据交换次数较少,但每次交换的数据量大,达几十至几千字节。
- Cache
  - Cache主要目的是解 决存储速度问题,使存储器的访问速度不 太影响CPU的运行速度。
  - 单位时间内数据交换的次数较多,每次交换的数据量较小,只有几个到几十个字节。

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)