

虚拟内存：系统

主要内容

- 简单内存系统示例
- 案例研究: Core i7/Linux 内存系统
- Linux内存映射

符号回顾

■ 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
- VPO (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

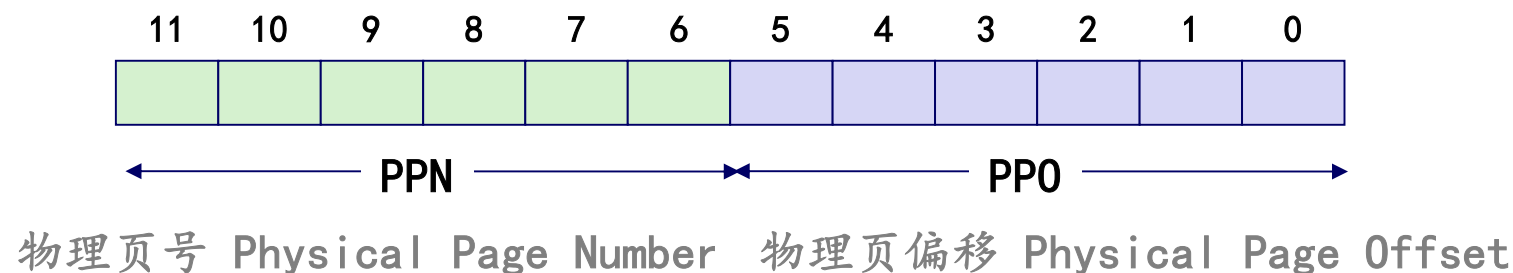
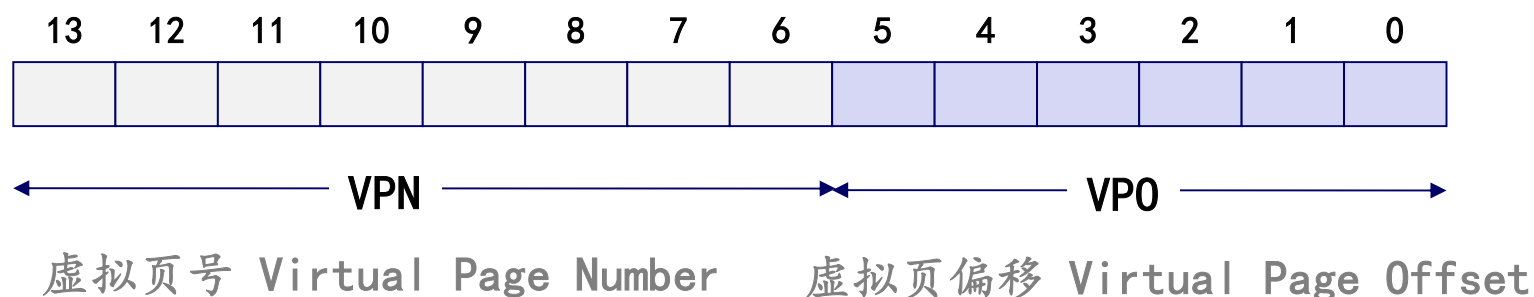
- PPO (Physical Page Offset) : 物理页偏移, 与虚拟页偏移 VPO 完全相同
- PPN (Physical Page Number) : 物理页号
- CO: Byte offset within cache line, 偏移
- CI: Cache index
- CT: Cache tag

端到端的地址翻译：一个简单的例子

- 这一节里，我们通过一个具体的端到端的地址翻译示例，来综合一下我们刚学过的这些内容，这个例子运行在有一个TLB和L1 d-cache的小系统上，为了保还可管理性，我们做出如下假设：
 - 内存是按字节寻址的。
 - 内存访问是针对1字节的字的(不是4字节的字)
 - 虚拟地址是14位长的($n=14$)。
 - 物理地址是12位长的($m=12$)。
 - 页面大小是64字节($P=64$)。
 - TLB是四路组相联的，总共有16个条目
 - L1 d-cache是物理寻址、直接映射的，行大小为4字节，而总共有16 个组。

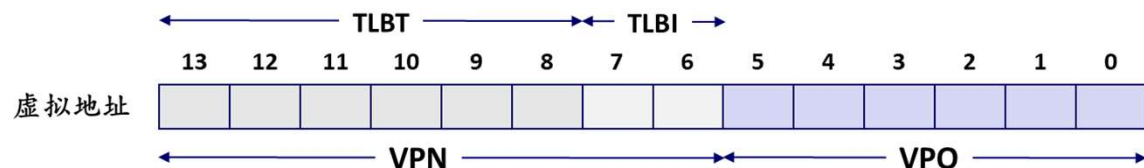
端到端的地址翻译: 一个简单的例子

- 这里展示了虚拟地址和物理地址的格式。
- 因为每个页面是 $2^6 = 64$ 字节，所以虚拟地址和物理地址的低6位分别作为VP0和PP0。虚拟地址的高8位作为VPN。物理地址的高6位作为PPN。



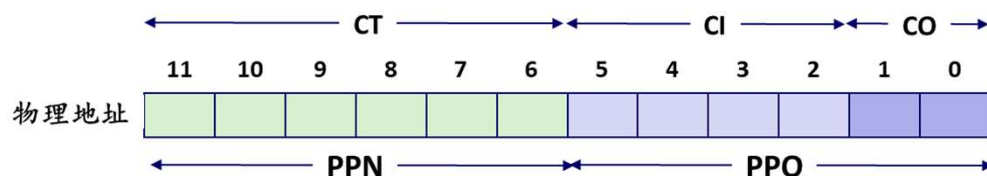
这个示例小内存系统的一个快照

- 包括TLB (a)、页表的一部分 (b)，高速缓存 (c)。
- 在TLB和高速缓存的图上面，我们还展示了访问这些设备时硬件是如何划分虚拟地址和物理地址的位的。



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

a) TLB: 四组，16个条目，四路组相连



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	-	-	-	-

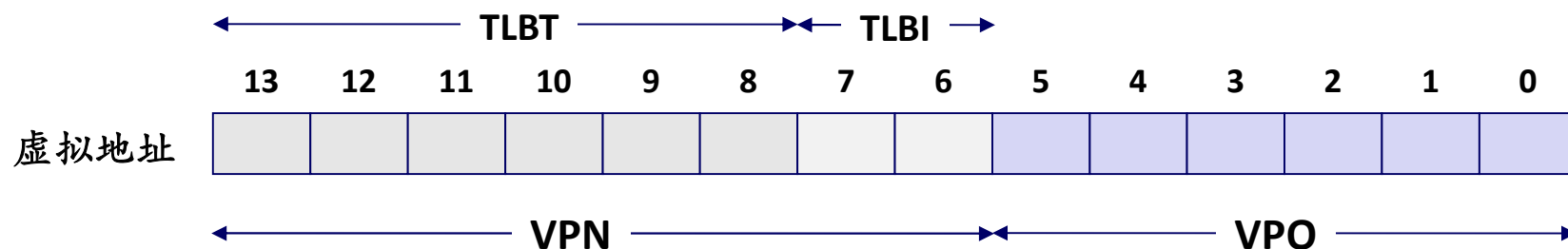
c) 高速缓存: 16个组，4字节的块，直接映射

VPN	PPN	Valid
00	28	1
01	-	0
02	33	1
03	02	1
04	-	0
05	16	1
06	-	0
07	-	0
08	13	1
09	17	1
0A	09	1
0B	-	0
0C	-	0
0D	2D	1
0E	11	1
0F	0D	1

b) 页表，只展示了前16个PTE

1. 示例小内存系统中的TLB

- TLB是利用 VPN 的位进行虚拟寻址的。
- TLB有4个组，所以 VPN 的低2位就作为组索引(TLBI)，VPN中剩下的高6位作为标记(TLBT)，用来区别可能映射到同一个 TLB 组的不同的VPN。



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

2. 示例小内存系统中的页表

- 这个页表是一个单级设计，一共有256个页表条目(PTE)。只列出了开头16个。
- 为了方便，我们用索引它的VPN来标识每个PTE;但是要记住VPN并不是页表的一部分，也不储存在内存中。
- 另外，每个无效PTE的PPN都用一个破折号来表示，以加强一个概念:无论刚好这里存储的是什么位值，都是没有任何意义的。

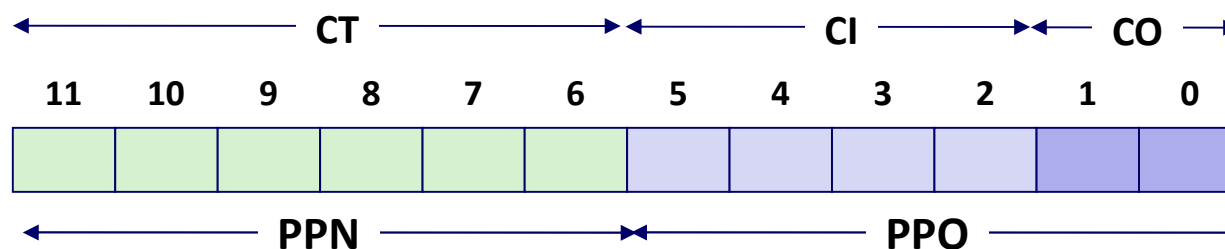
<i>VPN</i>	<i>PPN</i>	<i>Valid</i>
00	28	1
01	—	0
02	33	1
03	02	1
04	—	0
05	16	1
06	—	0
07	—	0

<i>VPN</i>	<i>PPN</i>	<i>Valid</i>
08	13	1
09	17	1
0A	09	1
0B	—	0
0C	—	0
0D	2D	1
0E	11	1
0F	0D	1

b) 页表，只展示了前
16个PTE

3. 示例小内存系统中的高速缓存(L1)

- 直接映射的缓存
- 通过物理地址中的字段来寻址
- 每个块都是4字节，所以物理地址底2位作为块偏移
- 一共16组，4位表示组索引 (CI)，6位表示标记 (CT)



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

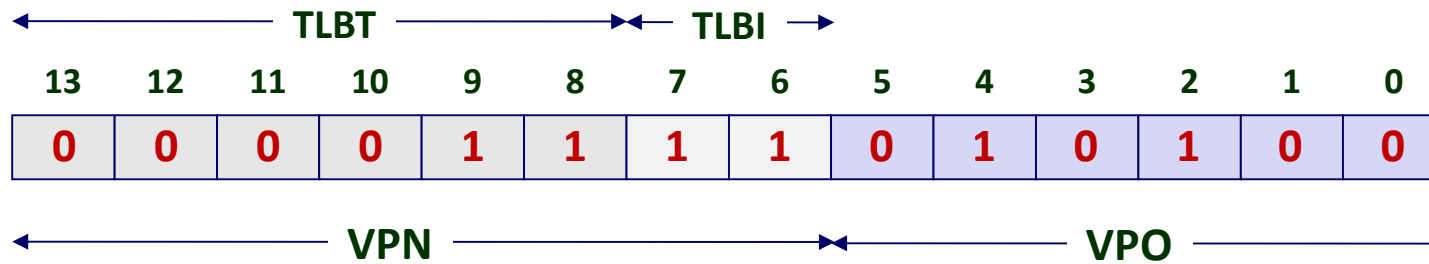
Idx	Tag	Valid	B0	B1	B2	B3
8	24	1	3A	00	51	89
9	2D	0	—	—	—	—
A	2D	1	93	15	DA	3B
B	0B	0	—	—	—	—
C	12	0	—	—	—	—
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	—	—	—	—

地址翻译例子#1

虚拟地址: 0x03D4:

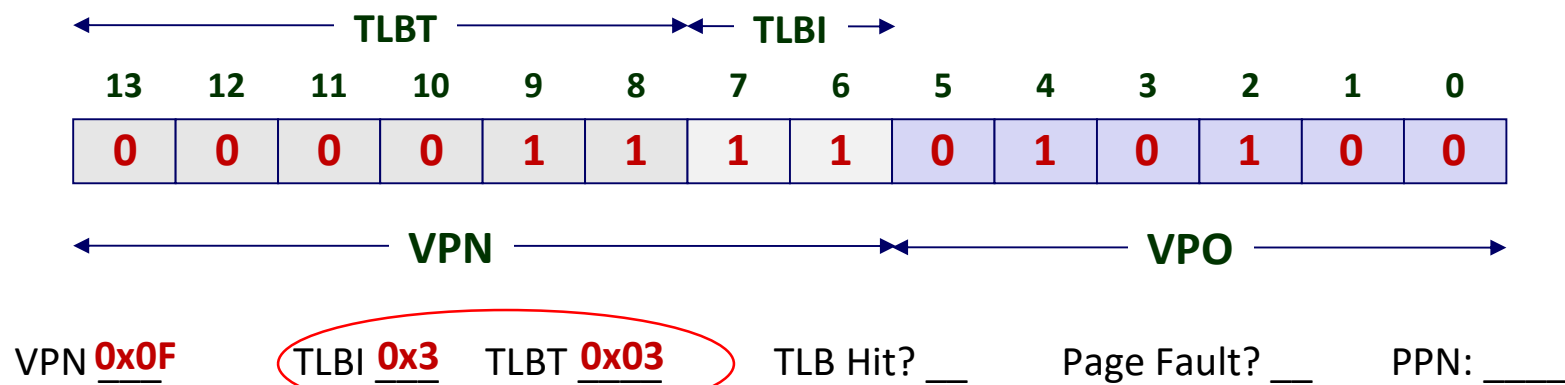
地址翻译例子#1

虚拟地址: 0x03D4:



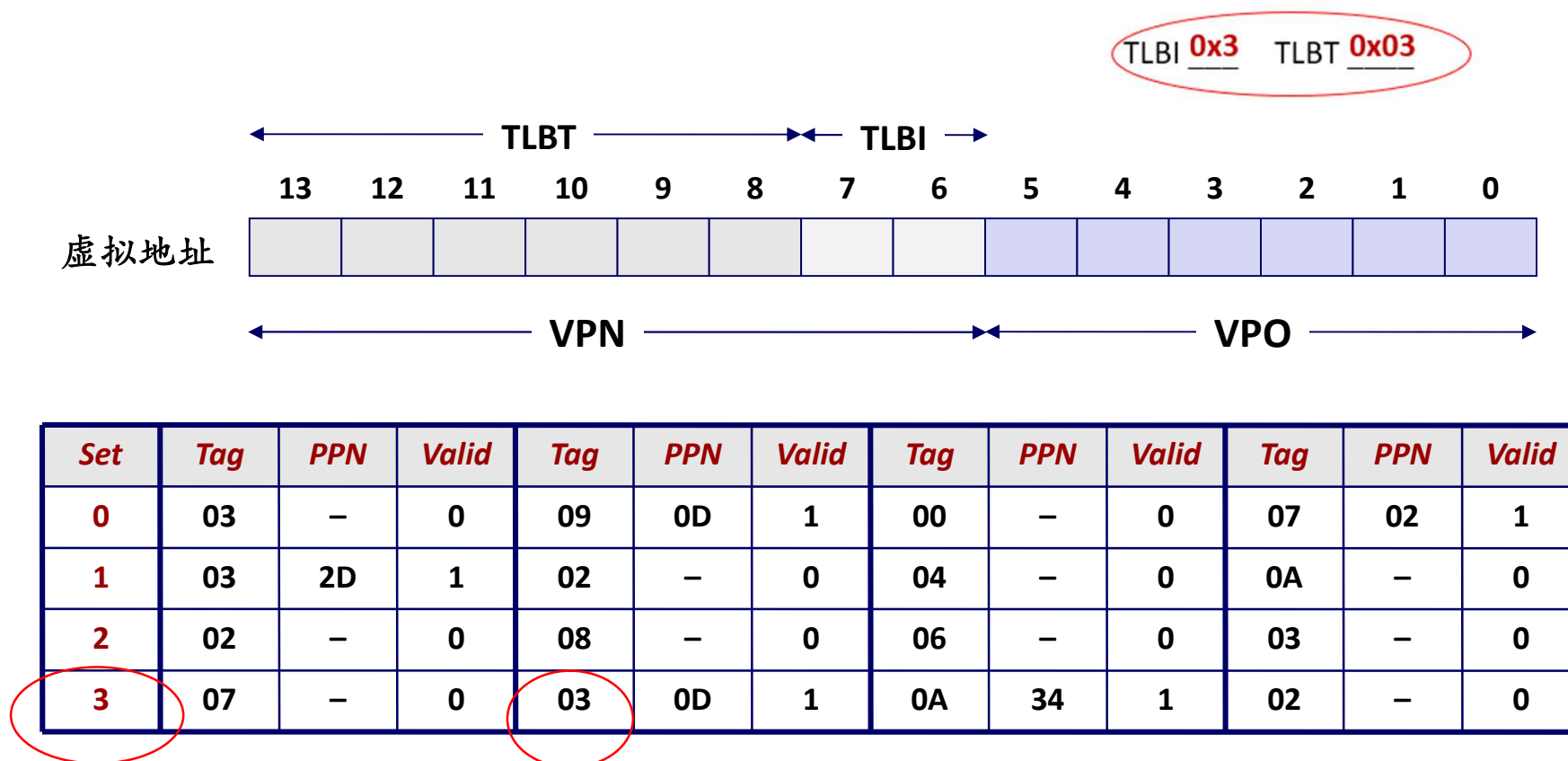
地址翻译例子#1

虚拟地址: 0x03D4:



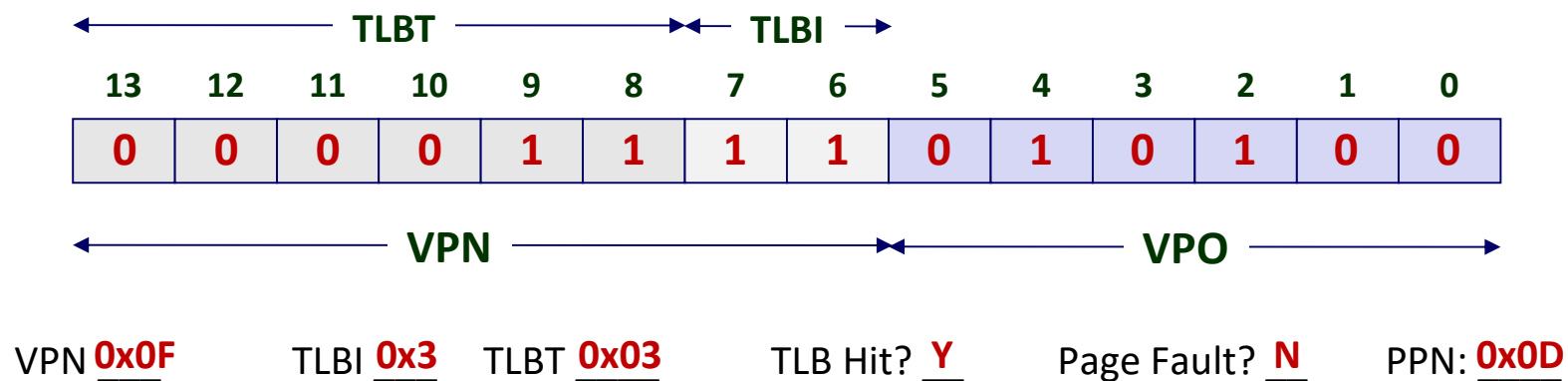
1. 示例小内存系统中的TLB

- TLB是利用 VPN 的位进行虚拟寻址的。
- TLB有4个组，所以 VPN 的低2位就作为组索引(TLBI)，VPN中剩下的高6位作为标记(TLBT)，用来区别可能映射到同一个 TLB 组的不同的VPN。



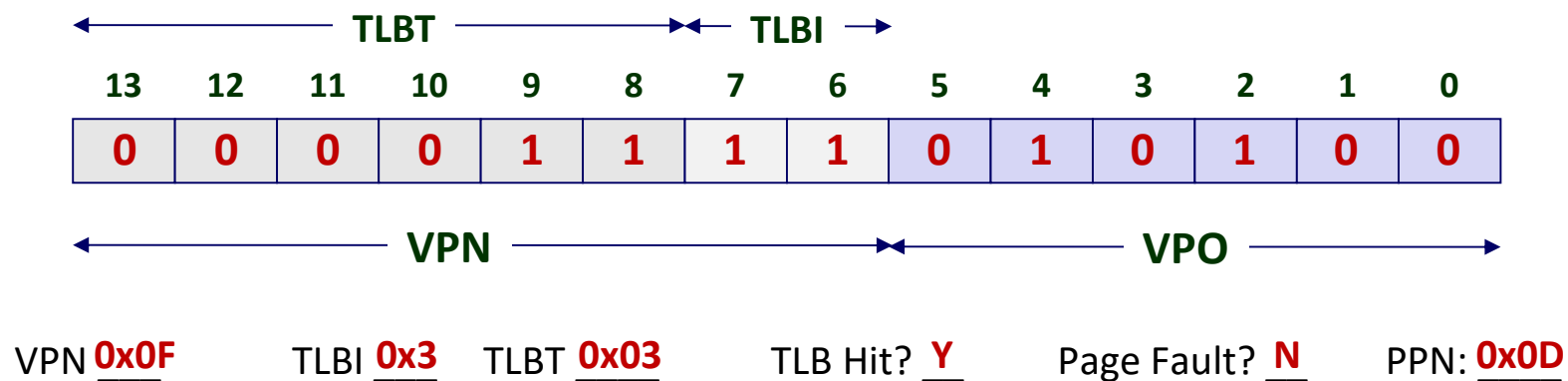
地址翻译例子#1

虚拟地址: 0x03D4:

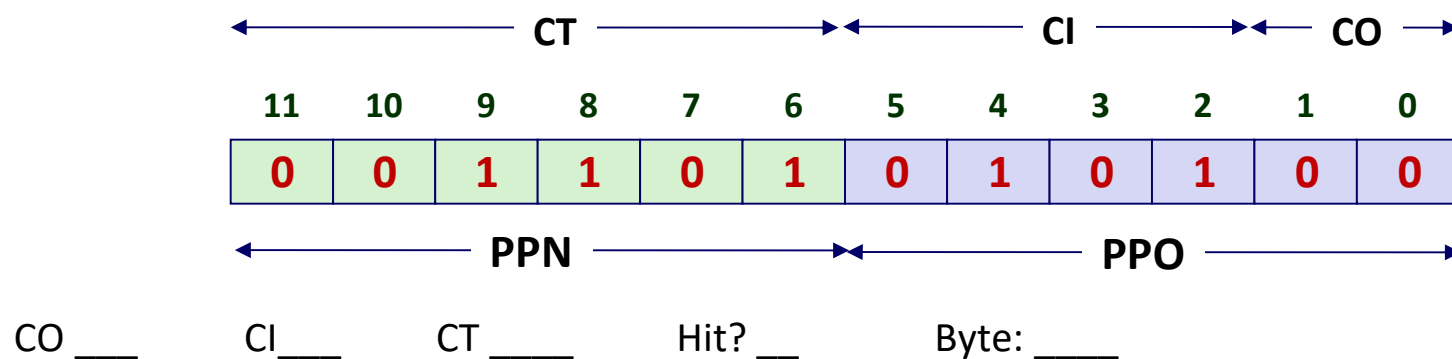


地址翻译例子#1

虚拟地址: 0x03D4:

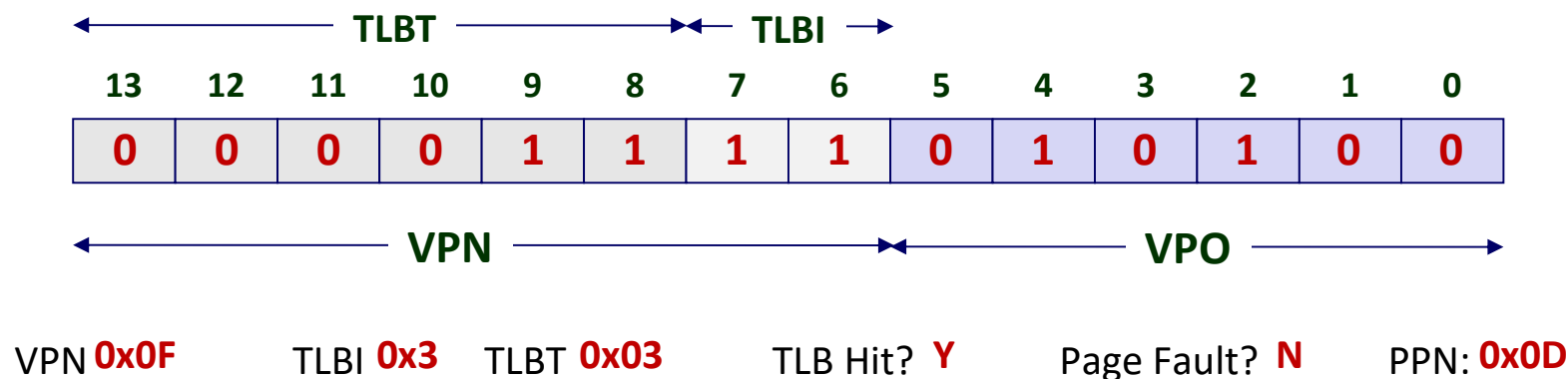


物理地址:

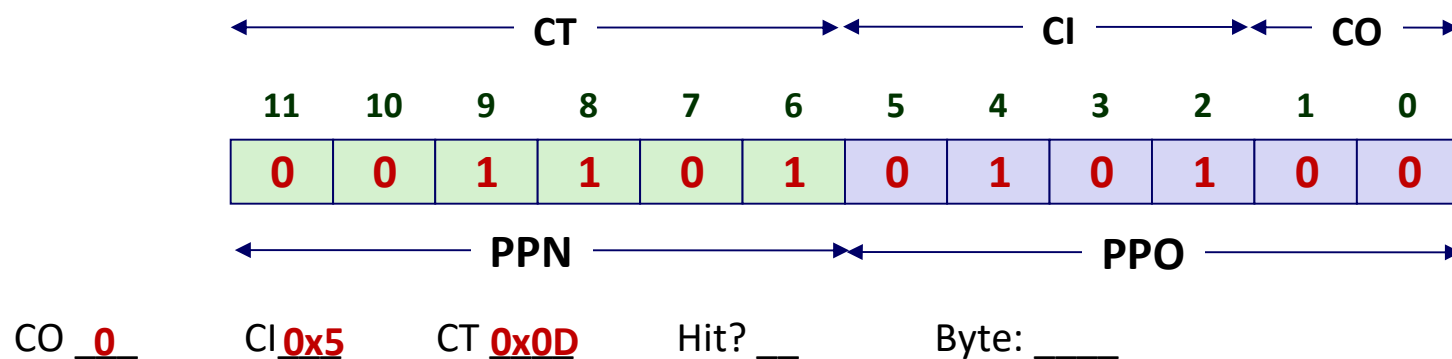


地址翻译例子#1

虚拟地址: 0x03D4:



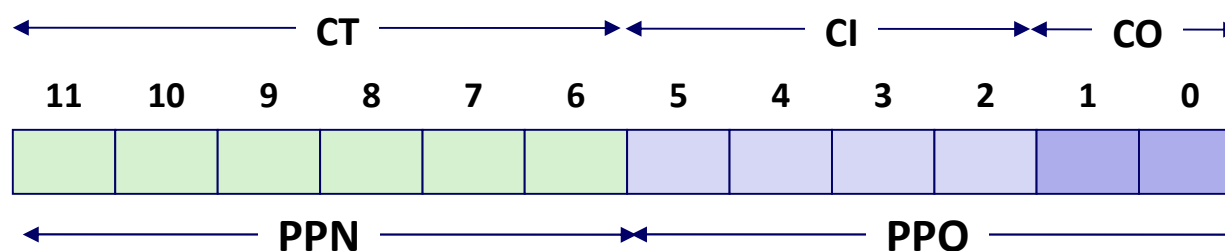
物理地址:



3. 示例的小内存系统中的高速缓存(L1)

- 直接映射的缓存
- 通过物理地址中的字段来寻址
- 每个块都是4字节，所以物理地址底2位作为块偏移
- 一共16组，4位表示组索引 (CI)，6位表示标记 (CT)

CO 0 CI 0x5 CT 0x0D

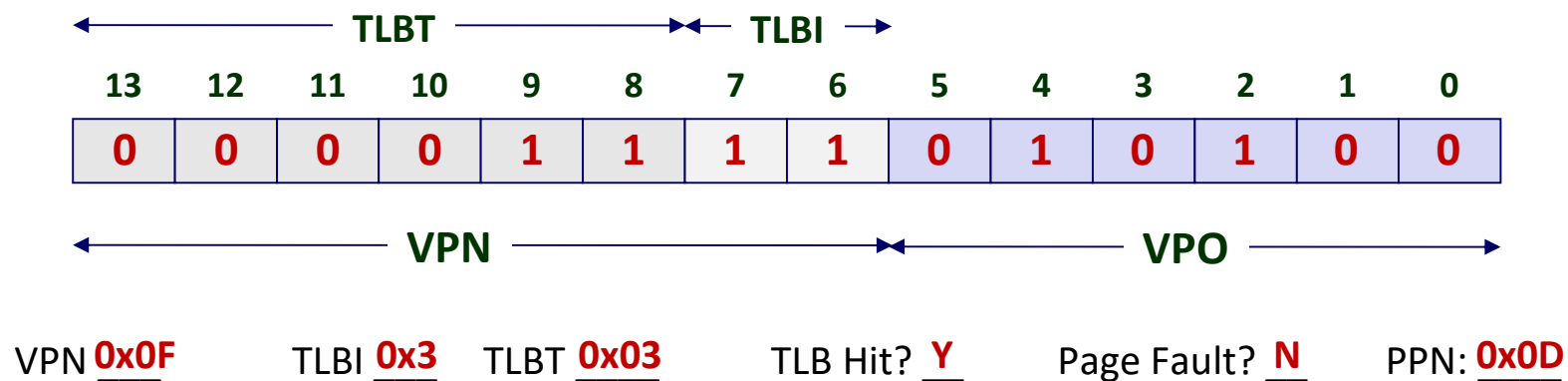


Idx	Tag	Valid	B0	B1	B2	B3
0	19	1	99	11	23	11
1	15	0	—	—	—	—
2	1B	1	00	02	04	08
3	36	0	—	—	—	—
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	—	—	—	—
7	16	1	11	C2	DF	03

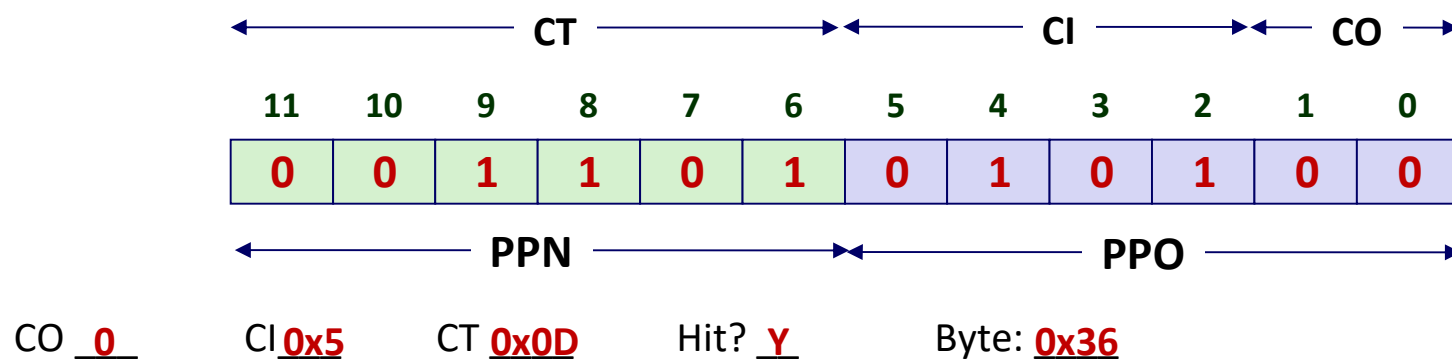
Idx	Tag	Valid	B0	B1	B2	B3
8	24	1	3A	00	51	89
9	2D	0	—	—	—	—
A	2D	1	93	15	DA	3B
B	0B	0	—	—	—	—
C	12	0	—	—	—	—
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	—	—	—	—

地址翻译例子#1

虚拟地址: 0x03D4:



物理地址:

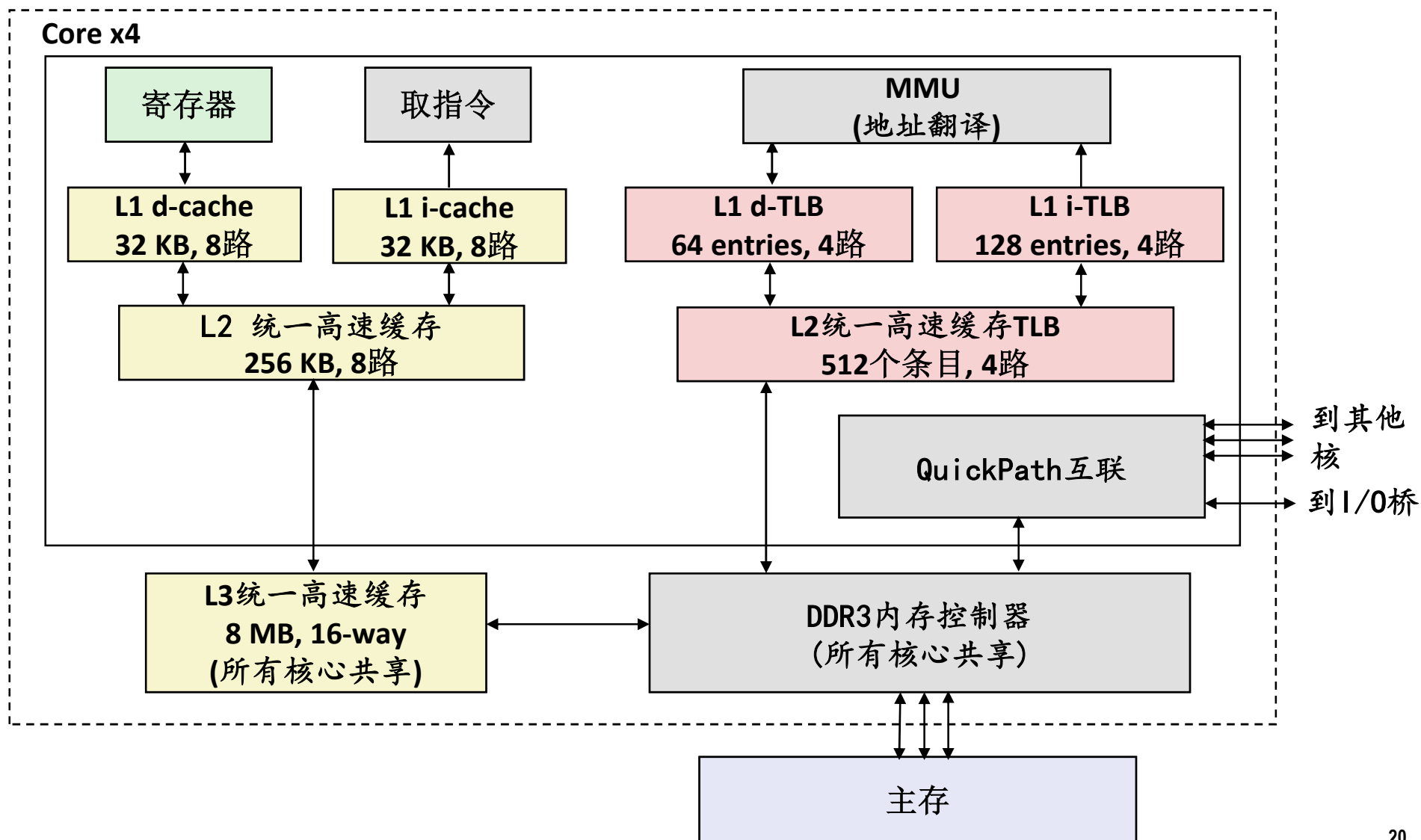


主要内容

- 简单内存系统示例
- 案例研究：Core i7/Linux 内存系统
- Linux内存映射

Intel Core i7 内存系统

处理器封装



符号回顾

■ 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)

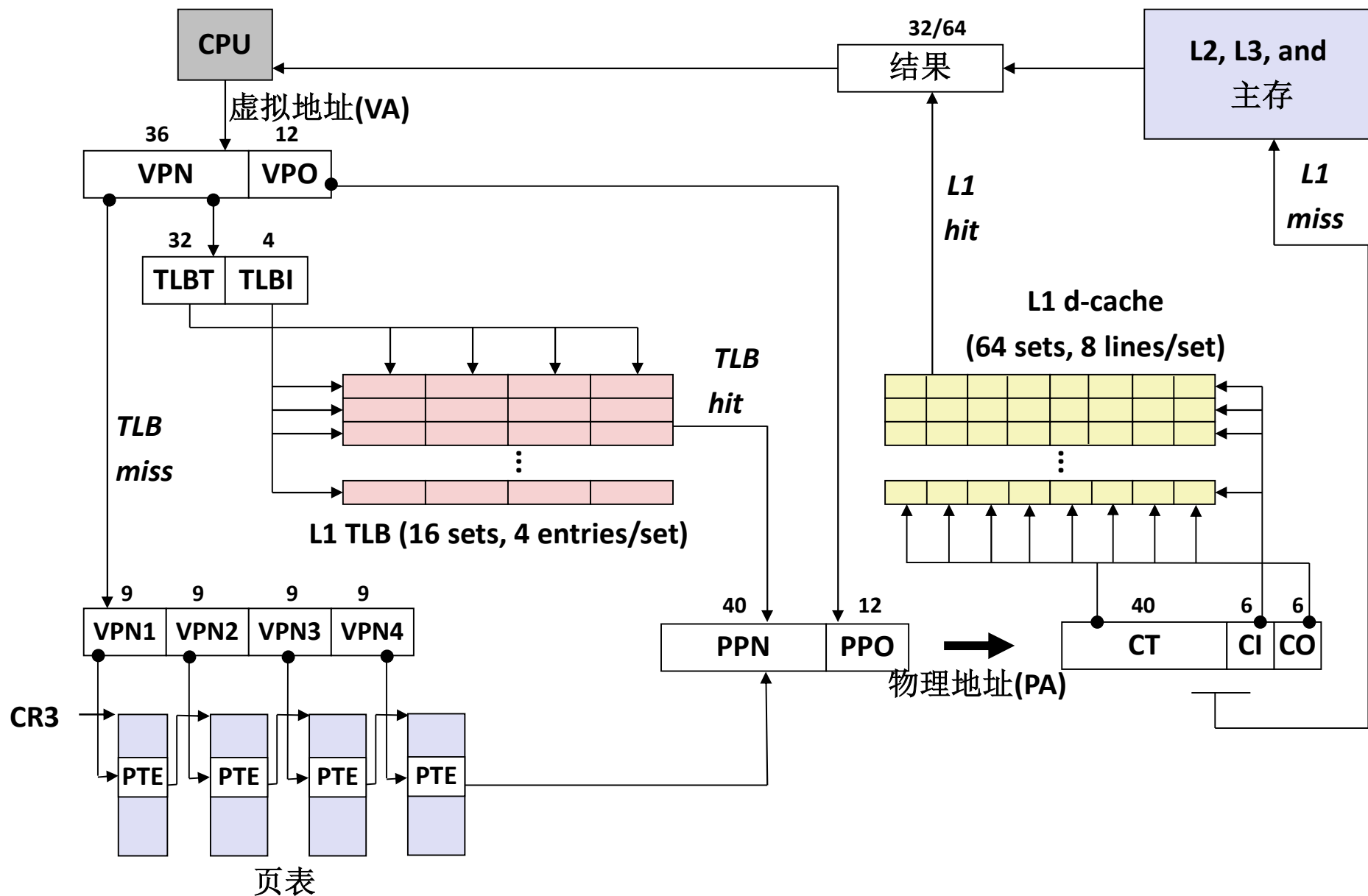
■ 虚拟地址 (VA, Virtual Address) 的组成

- TLBI (TLB Index) : TLB 索引
- TLBT (TLB Tag) : TLB 标签
- VPO (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

- PPO (Physical Page Offset) : 物理页偏移, 与虚拟页偏移 VPO 完全相同
- PPN (Physical Page Number) : 物理页号
- CO: Byte offset within cache line, 偏移
- CI: Cache index
- CT: Cache tag

Core i7 地址翻译简化概况



Core i7 第1-3级页表条目格式

63	62	52	51	12	11	9	8	7	6	5	4	3	2	1	0
XD	未使用	页表物理基地址				未使用	G	PS		A	CD	WT	U/S	R/W	P=1
OS可用 (磁盘上的页表位置)															P=0

每个条目引用一个 4K 子页表。主要字段说明如下：

- P: 指示子页表是否存在于物理内存中。1 表示存在，0 表示不存在。
- R/W: 为所有可访问页面设置读写权限。只读或读写。
- U/S: 为所有可访问页面设置用户模式或特权（内核）模式的访问权限。
- WT: 子页表的缓存策略，选择直写（Write-through）或回写（Write-back）。
- A: 引用位（Reference bit），由 MMU 在读写操作时自动设置，软件可清零。
- PS: 页面大小，4KB 或 4MB（仅在一级页表项中定义）。
- 页表物理基址: 物理页表地址的最高 40 位，强制页表按 4KB 对齐。
- XD: 禁止或允许从该页表项可访问的所有页面中取指令。

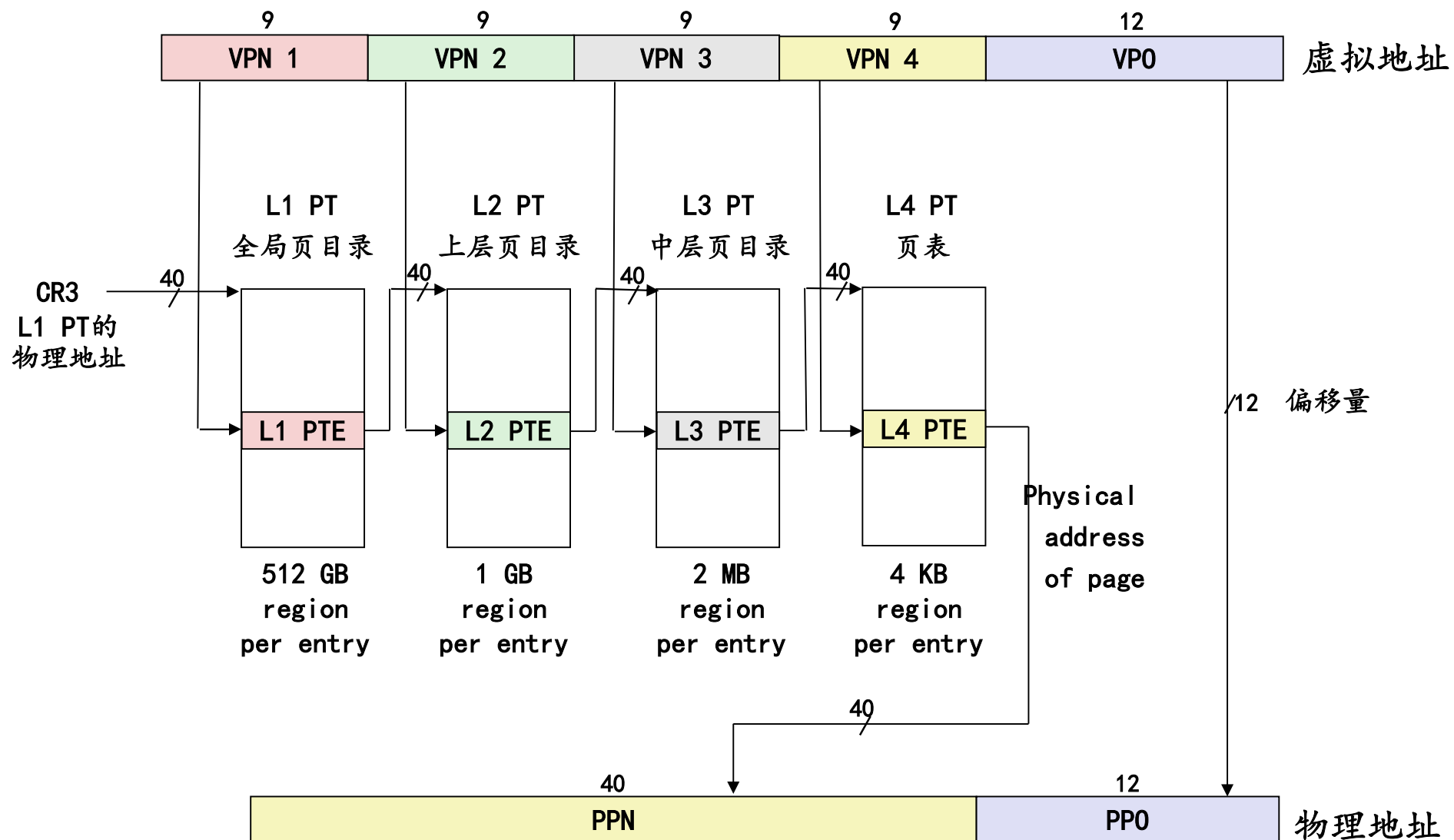
Core i7 第4级页表条目格式

63	62	52	51	12	11	9	8	7	6	5	4	3	2	1	0
XD	未使用	页表物理基地址				未使用	G		D	A	CD	WT	U/S	R/W	P=1
OS可用 (磁盘上的页表位置)															P=0

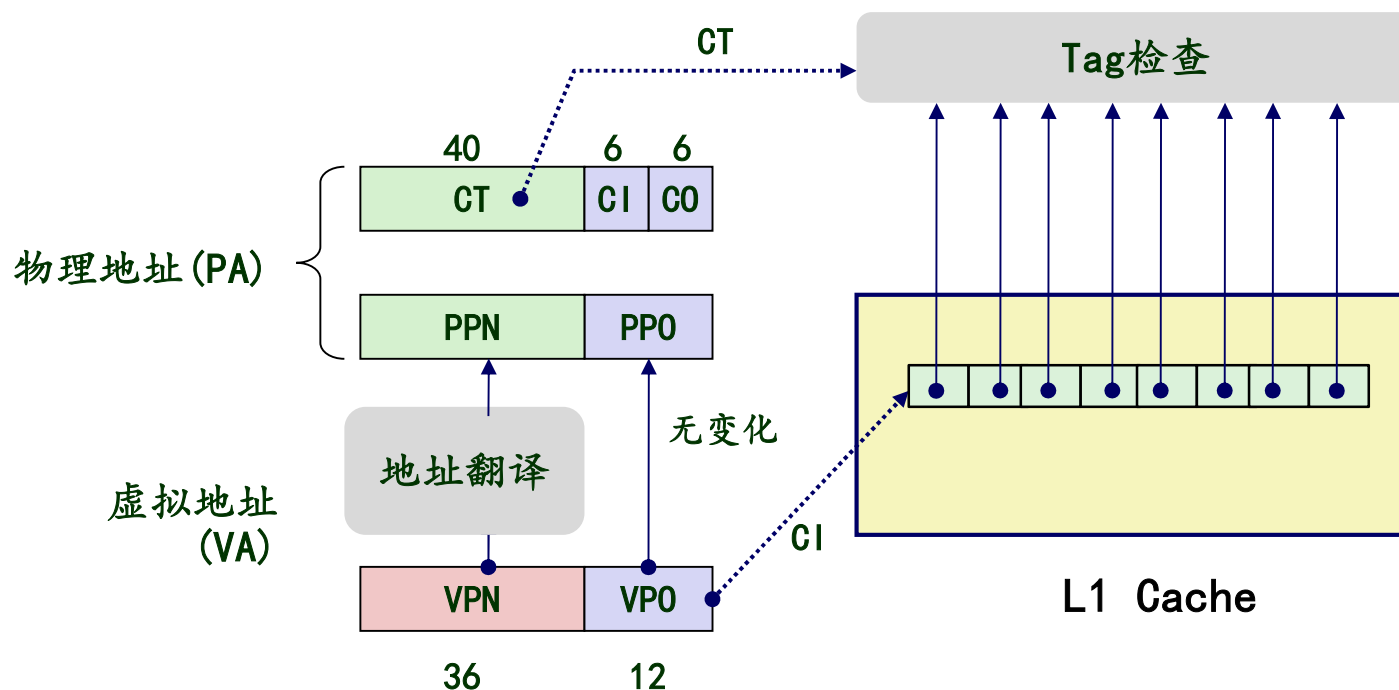
每个条目引用一个 4K 子页面。主要字段说明如下：

- P: 指示子页面是否存在于物理内存中。1 表示存在，0 表示不存在。
- R/W: 对子页面的读写权限控制，可设为只读或读写。
- U/S: 用户模式或特权（内核）模式的访问权限控制。
- WT: 此页面的缓存策略，选择直写（Write-through）或回写（Write-back）。
- A: 引用位（Reference bit），由 MMU 在读写操作时自动设置，软件可清零。
- D: 脏位（Dirty bit），由 MMU 在写操作时自动设置，软件可清零。
- 页面物理基址: 物理页面地址的最高 40 位，强制页面按 4KB 对齐。
- XD: 禁止或允许从该页面中取指令。

Core i7 页表地址翻译



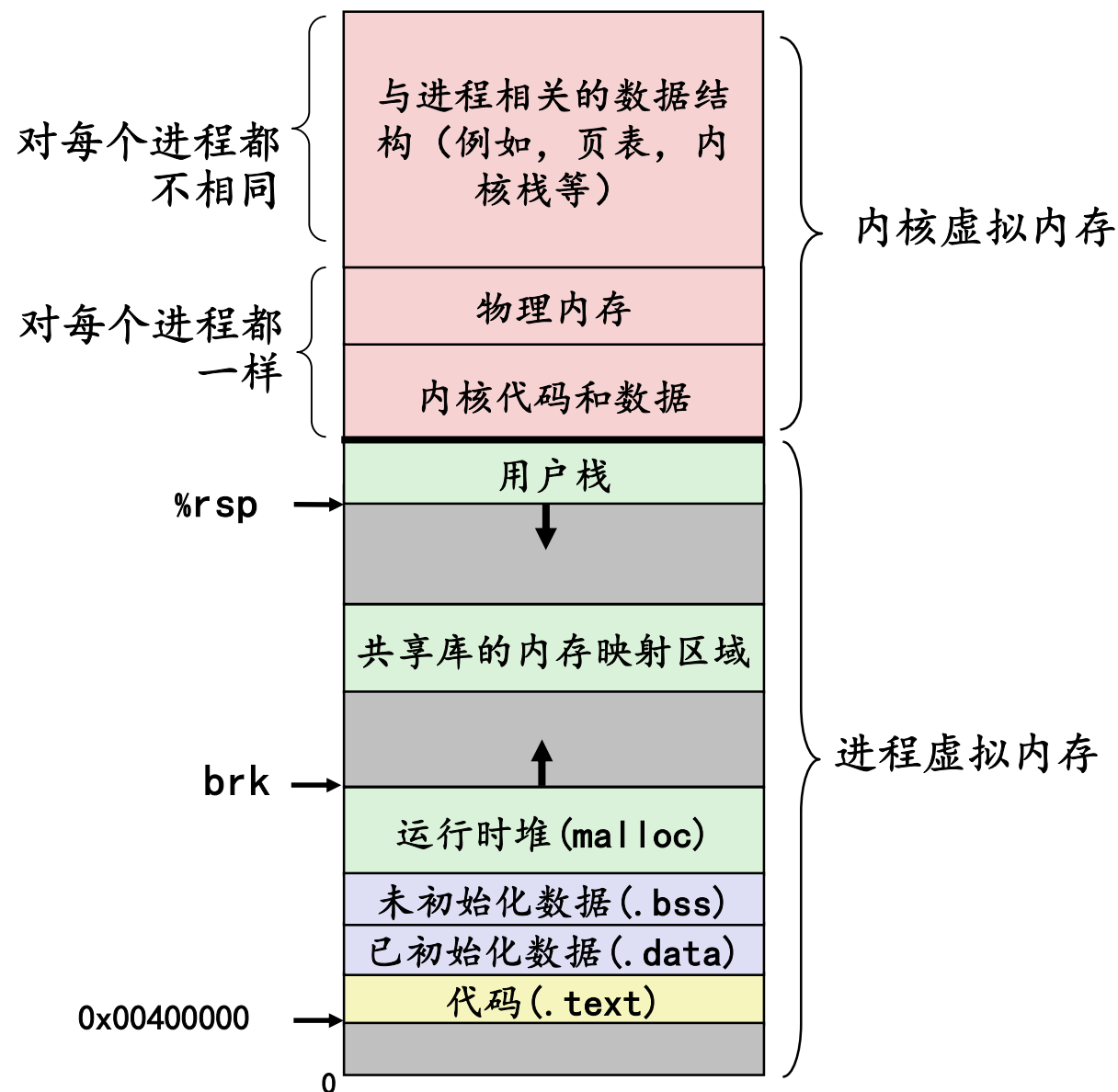
加速 L1 访问的技巧



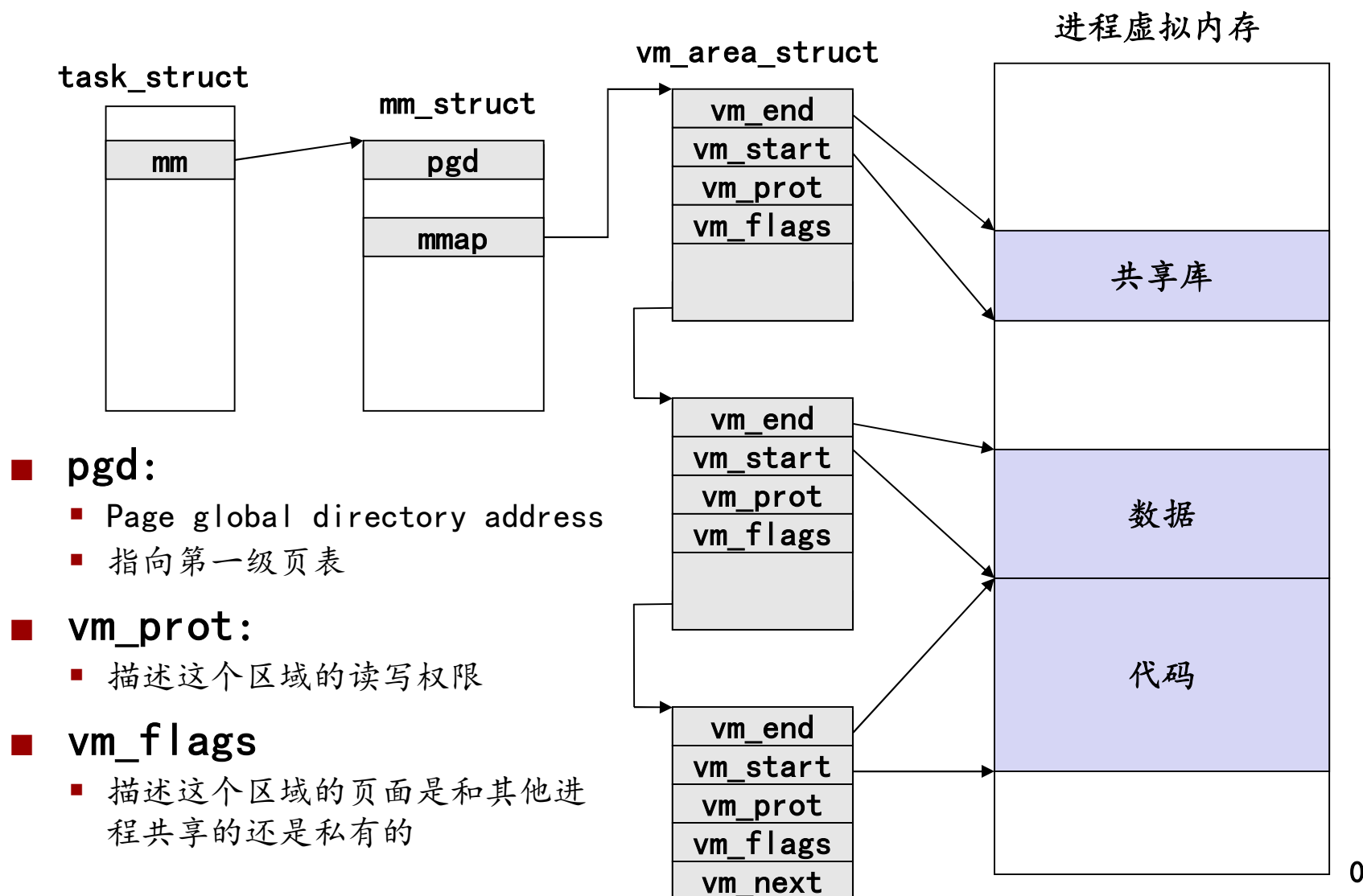
■ 观察:

- 确定在虚拟地址和物理地址中相同的缓存索引 (CI) 位
- 在地址转换的同时就可以用这些位索引缓存
- 通常会命中TLB, 因此 物理页号 (PPN) 中的高位 (即 CT bits) 很快可得。
- “虚拟索引, 物理标记” 缓存
- 缓存需要精心设计大小, 以确保上述方法可行。

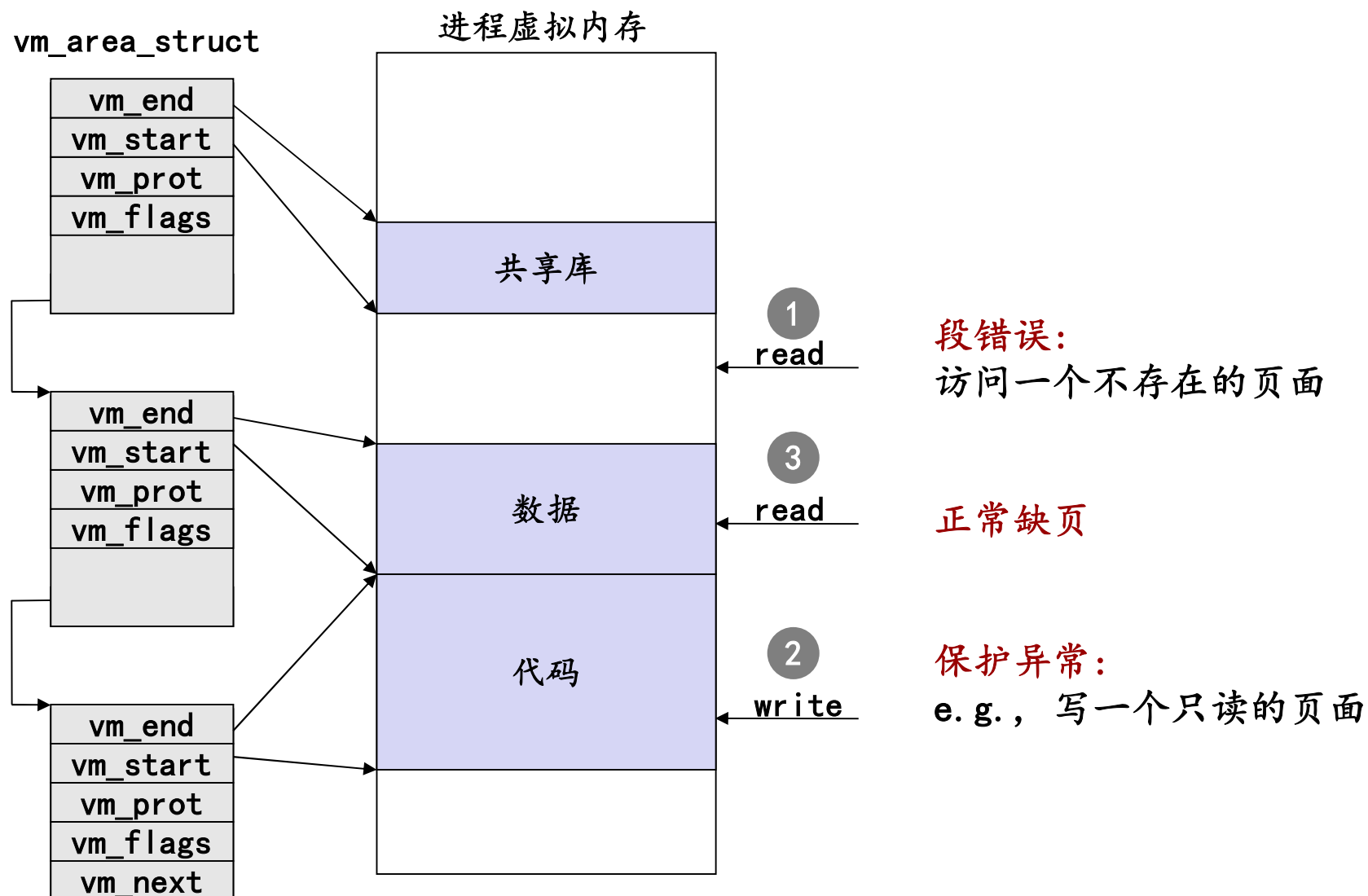
一个Linux进程的虚拟内存



Linux将虚拟内存组织为区域的集合



Linux缺页处理



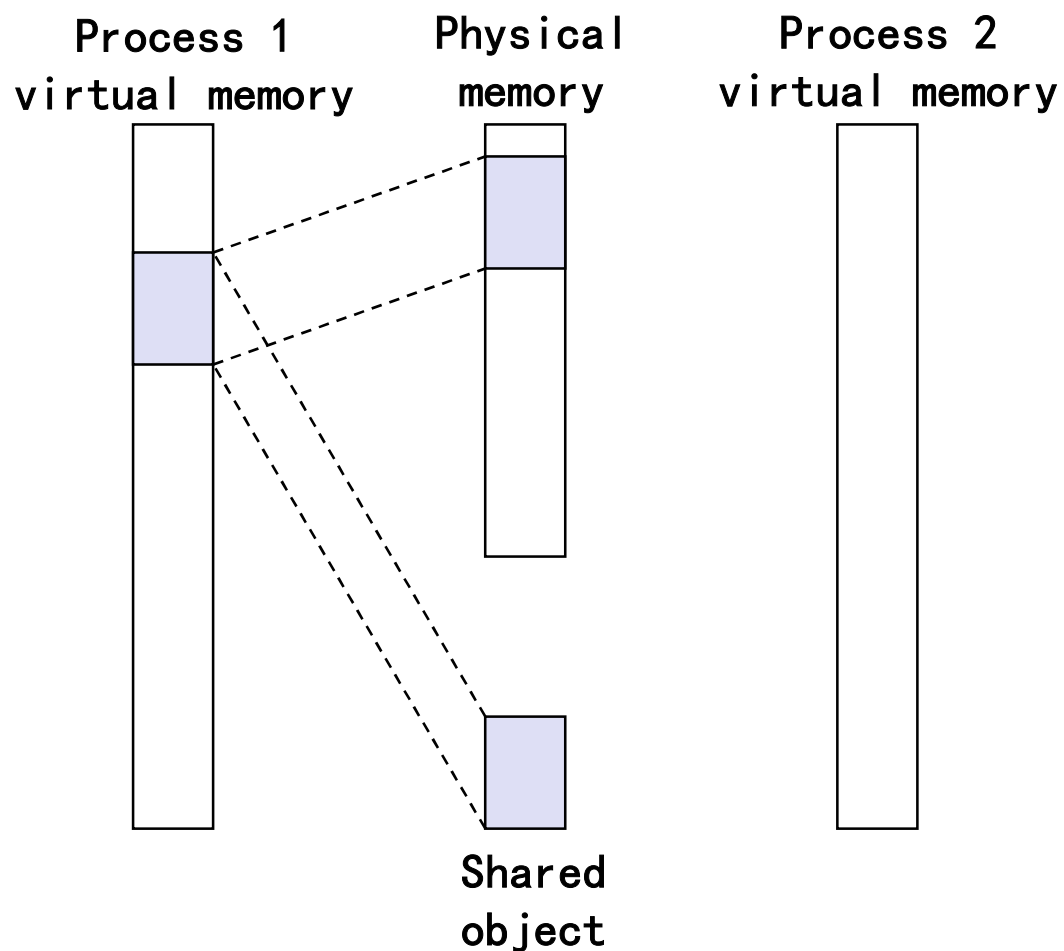
主要内容

- 简单内存系统示例
- 案例研究：Core i7/Linux 内存系统
- Linux内存映射

Linux 内存映射 (Memory Mapping)

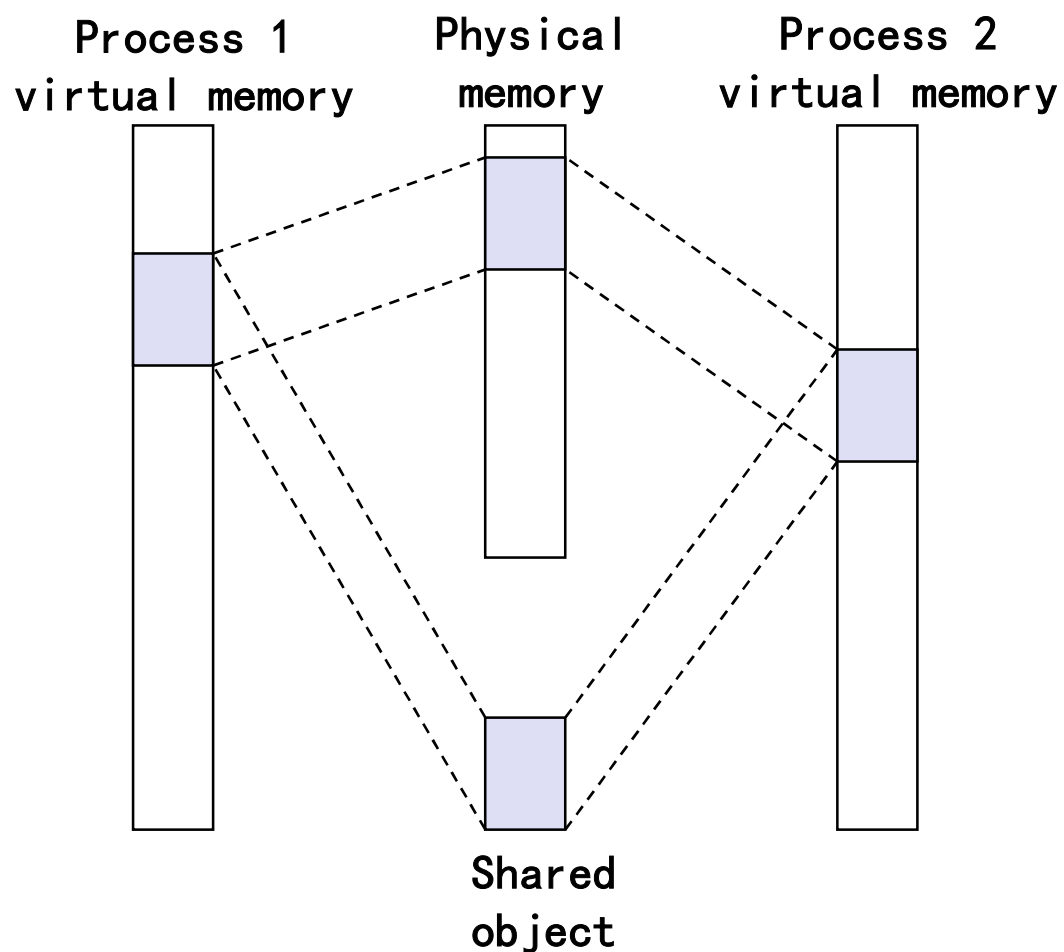
- 虚拟内存区域 (VM areas) 通过与磁盘对象关联来初始化。
 - 这个过程称为 内存映射 (memory mapping) 。
- 内存区域可以由以下来源支持 (backed by, 也就是初始值来自哪里) :
 - 普通文件 (Regular file) : 例如磁盘上的可执行文件。
 - 初始页的内容来自文件的某一部分。
 - 匿名文件 (Anonymous file) : 例如什么也没有。
 - 第一次缺页 (page fault) 时, 系统会分配一个全是 0 的物理页 (称为 需求零页 demand-zero page) 。
 - 一旦该页被写入数据 (变脏, dirtied) , 它就和普通页面一样。
- 脏页 (Dirty pages) 会在内存与一个特殊的交换文件 (swap file) 之间来回拷贝。

共享对象 (Shared Objects)



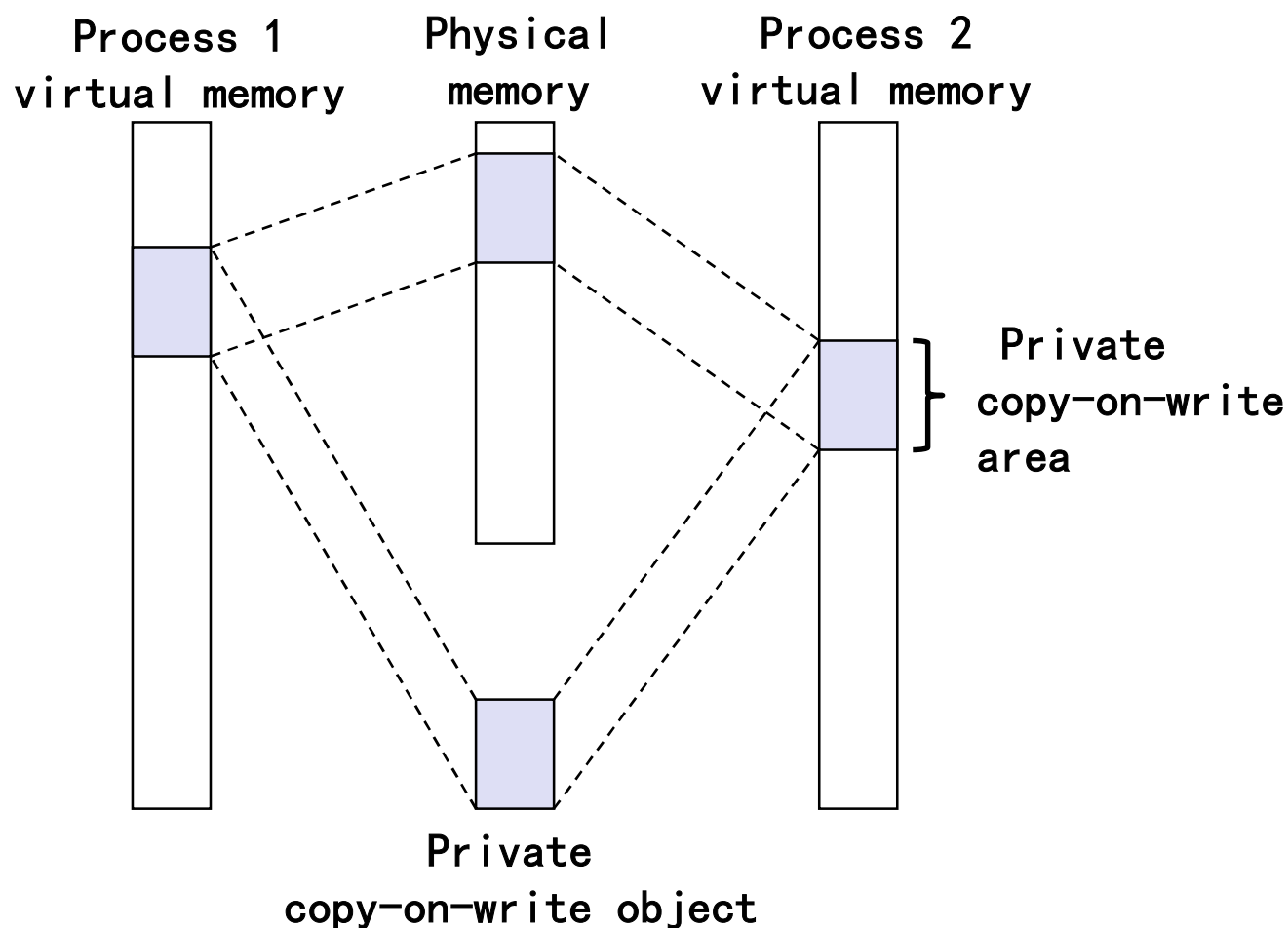
- 进程1通过虚拟内存映射到一块存放了共享对象的物理内存。

共享对象 (Shared Objects)



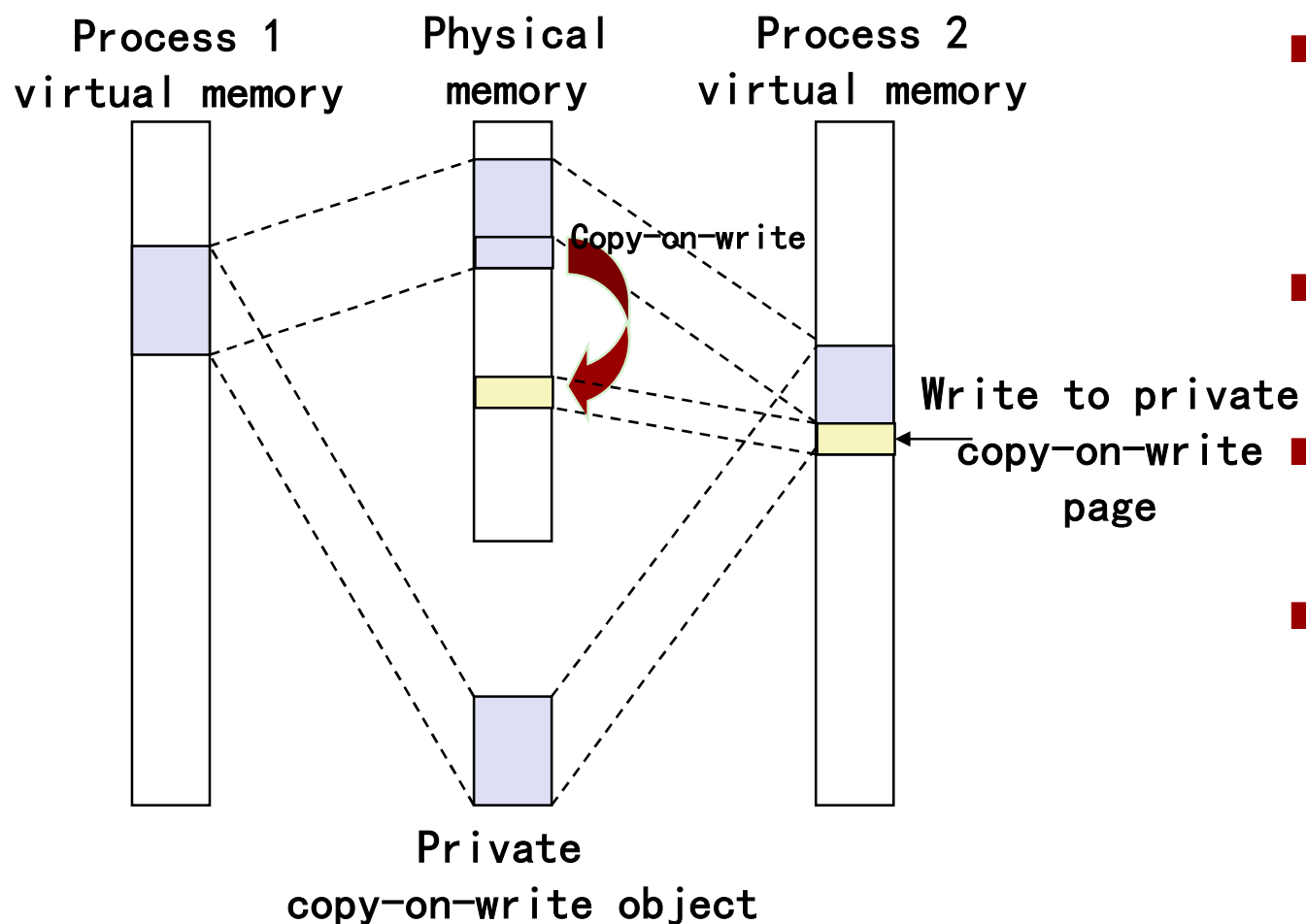
- 进程2也映射到了同一个共享对象。
- 注意：进程1和进程2虽然在虚拟地址空间中访问的是不同的虚拟地址，但实际上它们指向同一块物理内存。

写时复制 (Copy-On-Write, COW)



- 两个进程映射一个私有写时复制 (COW) 对象。
- 标记为私有写时复制的区域
- 私有区域中的 PTE 被标记为只读

写时复制 (Copy-On-Write, COW)



- 写入私有页面的指令触发保护错误。
- 处理程序创建新的读/写页面。
- 处理程序返回后，指令重新启动。
- 复制操作尽可能延迟！

这个示例小内存系统的一个快照

- 包括TLB、页表的一部分，高速缓存。
- 在TLB和高速缓存的图上面，我们还展示了访问这些设备时硬件是如何划分虚拟地址和物理地址的位的。

