第7周 IA-32/Linux中的地址转换

第1讲 IA-32的地址转换和寻址方式
第2讲 段选择符和段寄存器
第3讲 段描述符和段描述符表
第4讲 逻辑地址向线性地址的转换
第5讲 线性地址向物理地址的转换
第6讲 Intel Core i7/Linux存储系统

IA-32的存储管理

- 按字节编址（通用计算机大多是）
- 在保护模式下，IA-32采用段页式虚拟存储管理方式
- 存储地址采用逻辑地址、线性地址和物理地址来进行描述，其中，逻辑地址和线性地址是虚拟地址的两种不同表示形式，描述的都是4GB虚拟地址空间中的一个存储地址
  - ✓逻辑地址由48位组成，包含16位段选择符和32位段内偏移量（即有效地址）　　movw 8(%ebp,%edx,4), %ax
  - ✓线性地址32位（其位数由虚拟地址空间大小决定）
  - ✓物理地址32位（其位数由存储器总线中的地址线条数决定）
- 分段过程实现将逻辑地址转换为线性地址 ← 以下介绍分段机制
- 分页过程实现将线性地址转换为物理地址

IA-32处理器的寻址方式

IA-32指令举例：

movw 8(%ebp,%edx,4), %ax // R[ax]←M[R[ebp]+R[edx]×4+8]

32位有效地址

操作数的来源：

- 立即数(立即寻址)：直接来自指令
- 寄存器(寄存器寻址)：来自32位 / 16位 / 8位通用寄存器
- 存储单元(其他寻址)：需进行地址转换

逻辑地址 => 线性地址LA（=> 内存地址）

分段　　分页

即采用段页式！

指令中的信息：

(1) 段寄存器SR（隐含或显式给出）

(2) 8/16/32位偏移量A （显式给出）

(2) 基址寄存器B （明显给出，任意通用寄存器皆可）

(3) 变址寄存器I （明显给出，除ESP外的任意通用寄存器皆可。）

- 有比例变址和非比例变址
- 比例变址时要乘以比例因子S (1:8位 / 2:16位 / 4:32位 / 8:64位)

IA-32处理器寻址方式

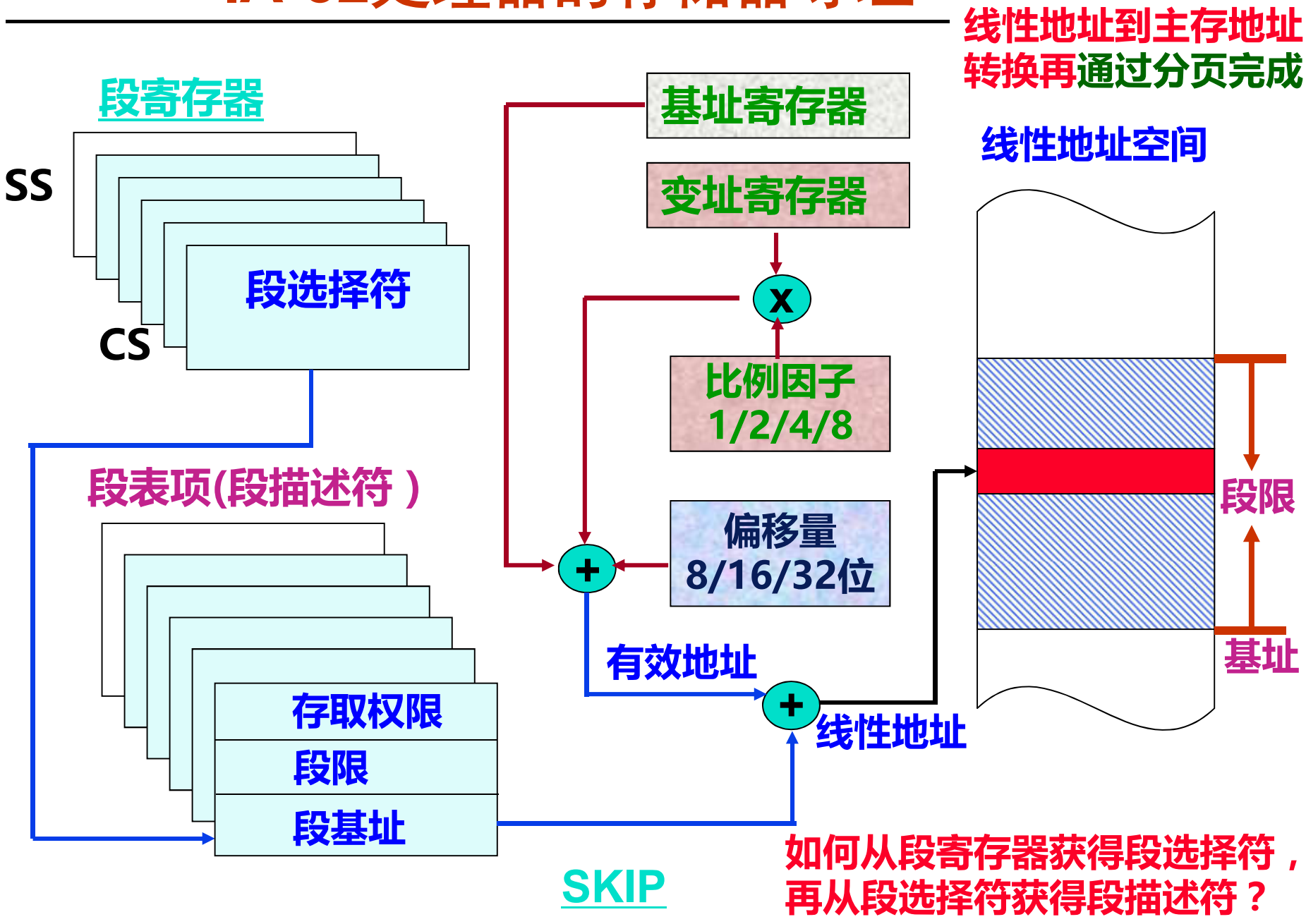
寻址方式	算法
立即(地址码A本身为操作数)	操作数=A
寄存器(通用寄存器的内容为操作数)	操作数= (R)
偏移量(地址码A给出8/16/32位偏移量)	LA=(SR)+A
基址(地址码B给出基址器编号)	LA=(SR)+(B)
基址带偏移量(一维表访问)	LA=(SR)+(B)+A
比例变址带偏移量(一维表访问)	LA=(SR)+ (I)xS+A
基址带变址和偏移量(二维表访问)	LA=(SR)+(B)+(I) +A
基址带比例变址和偏移量(二维表访问)	LA=(SR)+(B)+(I)xS+A
相对(给出下一指令的地址，转移控制)	转移地址=(PC)+A

有效地址EA

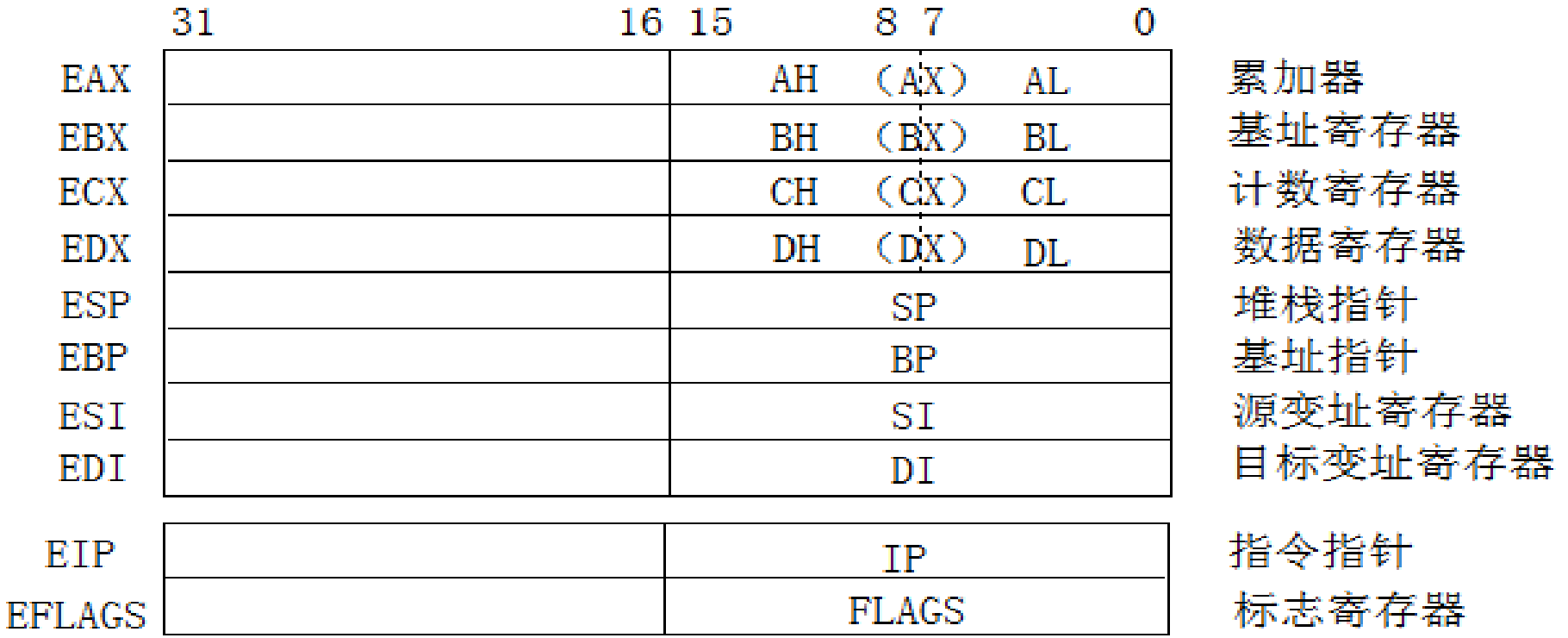
IA-32指令举例：

movw 8(%ebp,%edx,4), %ax // R[ax]←M[R[ebp]+R[edx]×4+8]

IA-32处理器的存储器寻址



IA-32的寄存器组织



8个通用寄存器

两个专用寄存器

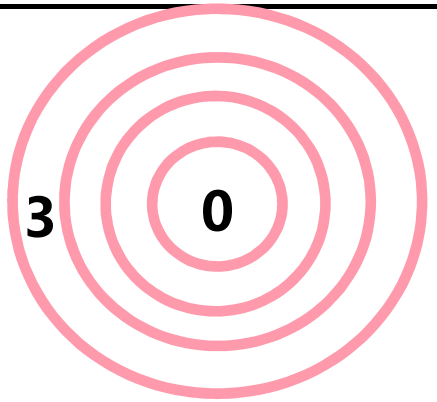
6个段寄存器

CS	代码段
SS	堆栈段
DS	数据段
ES	附加段
FS	附加段
GS	附加段

BACK

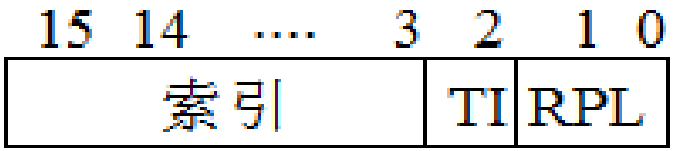
段选择符和段寄存器

° 段寄存器（16位），用于存放段选择符
- CS(代码段)：程序代码所在段
- SS(栈段)：栈区所在段
- DS(数据段)：全局静态数据区所在段
- 其他3个段寄存器ES、GS和FS可指向任意数据段



环保护：内核工作在0环，用户工作在3环，中间环留给中间软件用。Linux仅用第0和第3环。

°段选择符各字段含义：



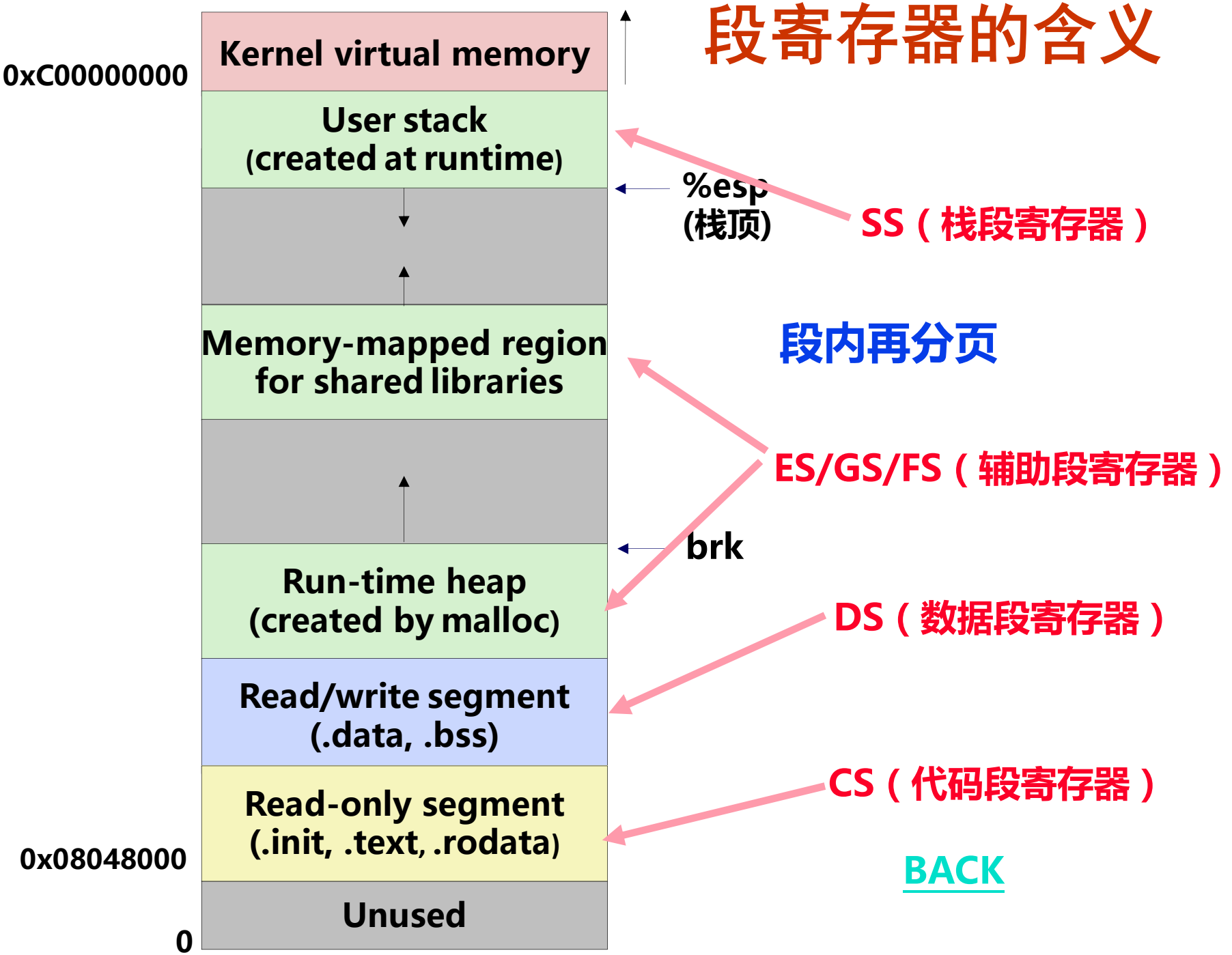
CS寄存器中的RPL字段表示CPU的当前特权级（Current Privilege Level，CPL）

- TI=0，选择全局描述符表(GDT)，TI=1，选择局部描述符表(LDT)
- RPL=00，为第0级，位于最高级的内核态，RPL=11，为第3级，位于最低级的用户态，第0级高于第3级
- 高13位索引用来确定当前使用的段描述符在描述表中的位置

你认为什么是段描述符？ 段表项—段的描述信息

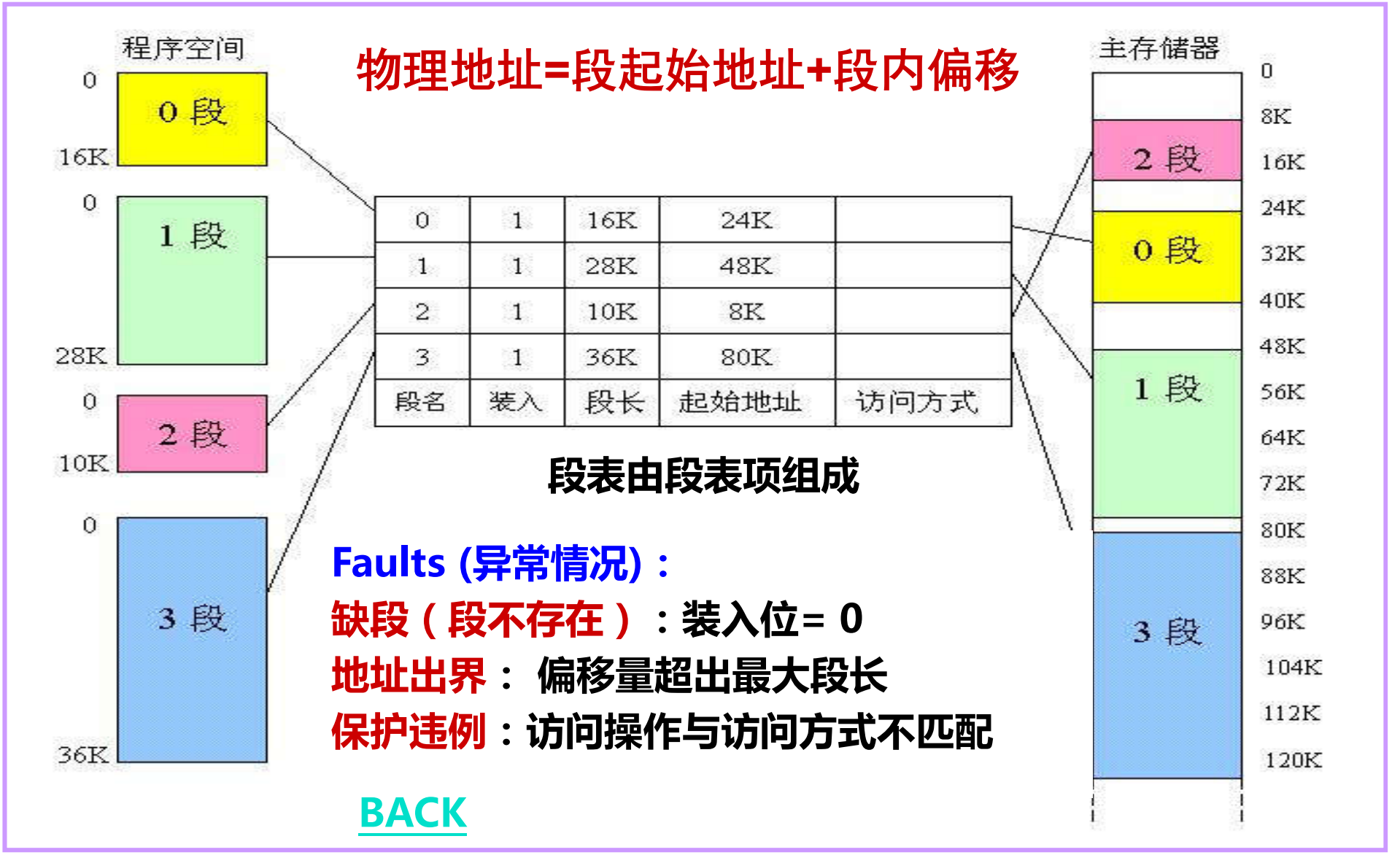
SKIP

段寄存器的含义



BACK

段式虚拟存储器的地址映像

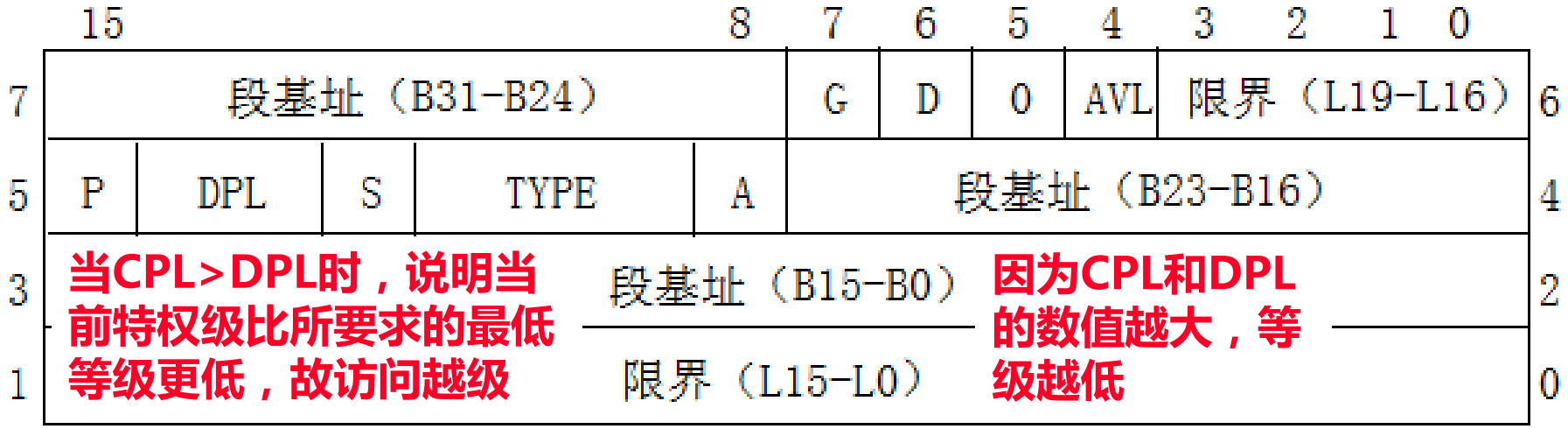


段描述符和段描述符表

- 段描述符是一种数据结构，实际上就是段表项，分两类：
  - 普通段：用户/内核的代码段和数据段描述符
  - 系统控制段描述符，又分两种：
    - 特殊系统控制段描述符，包括：局部描述符表（LDT）描述符和任务状态段（TSS）描述符
    - 控制转移类描述符，包括：调用门描述符、任务门描述符、中断门描述符和陷阱门描述符
- 描述符表实际上就是段表，由段描述符(段表项)组成。有三种类型：
  - 全局描述符表GDT：只有一个，用来存放系统内每个任务共用的描述符，例如，内核代码段、内核数据段、用户代码段、用户数据段以及TSS（任务状态段）等都属于GDT中描述的段
  - 局部描述符表LDT：存放某任务（即用户进程）专用的描述符
  - 中断描述符表IDT：包含256个中断门、陷阱门和任务门描述符

IDT将在第7章介绍

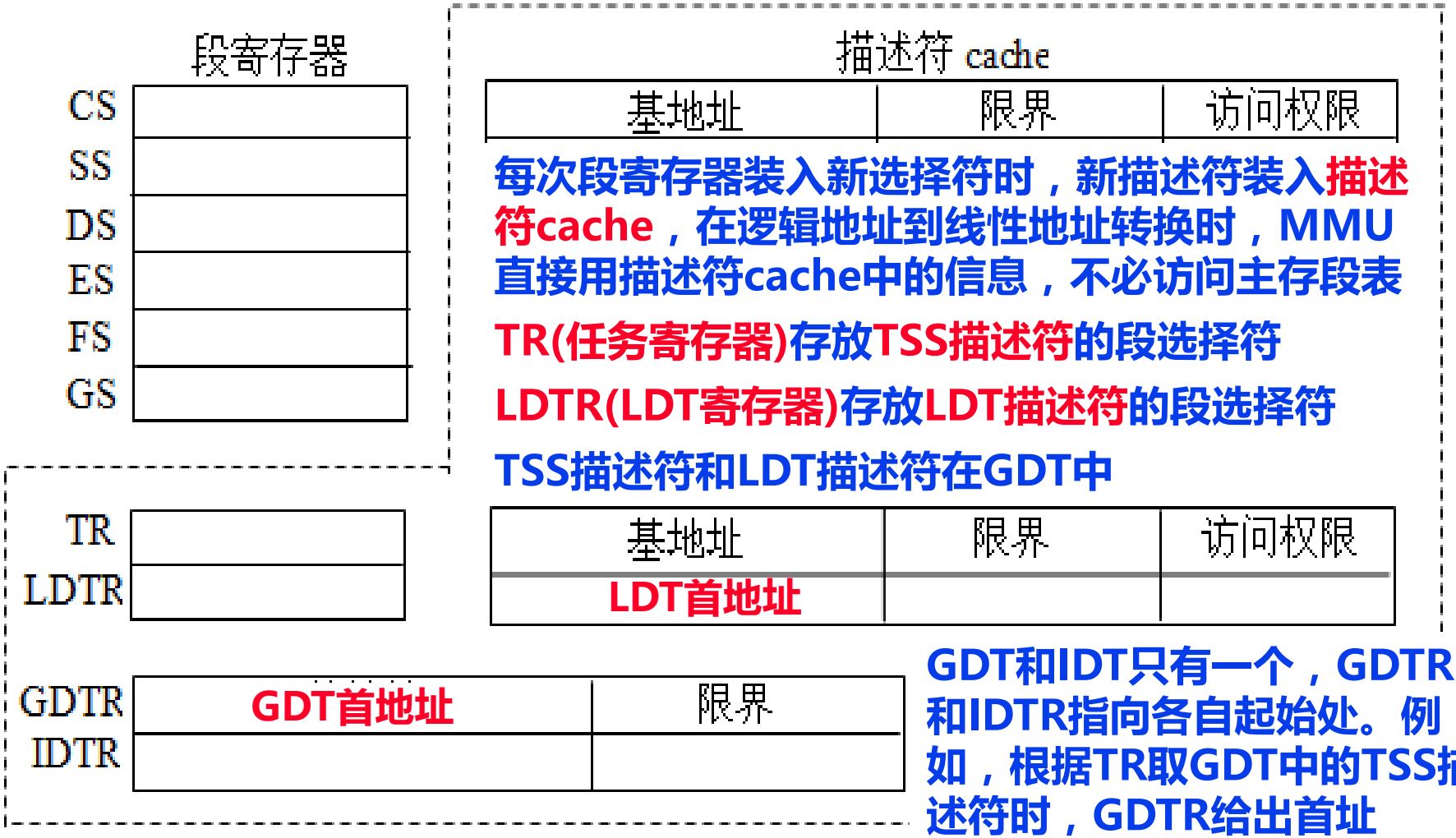
段描述符（8B）的定义



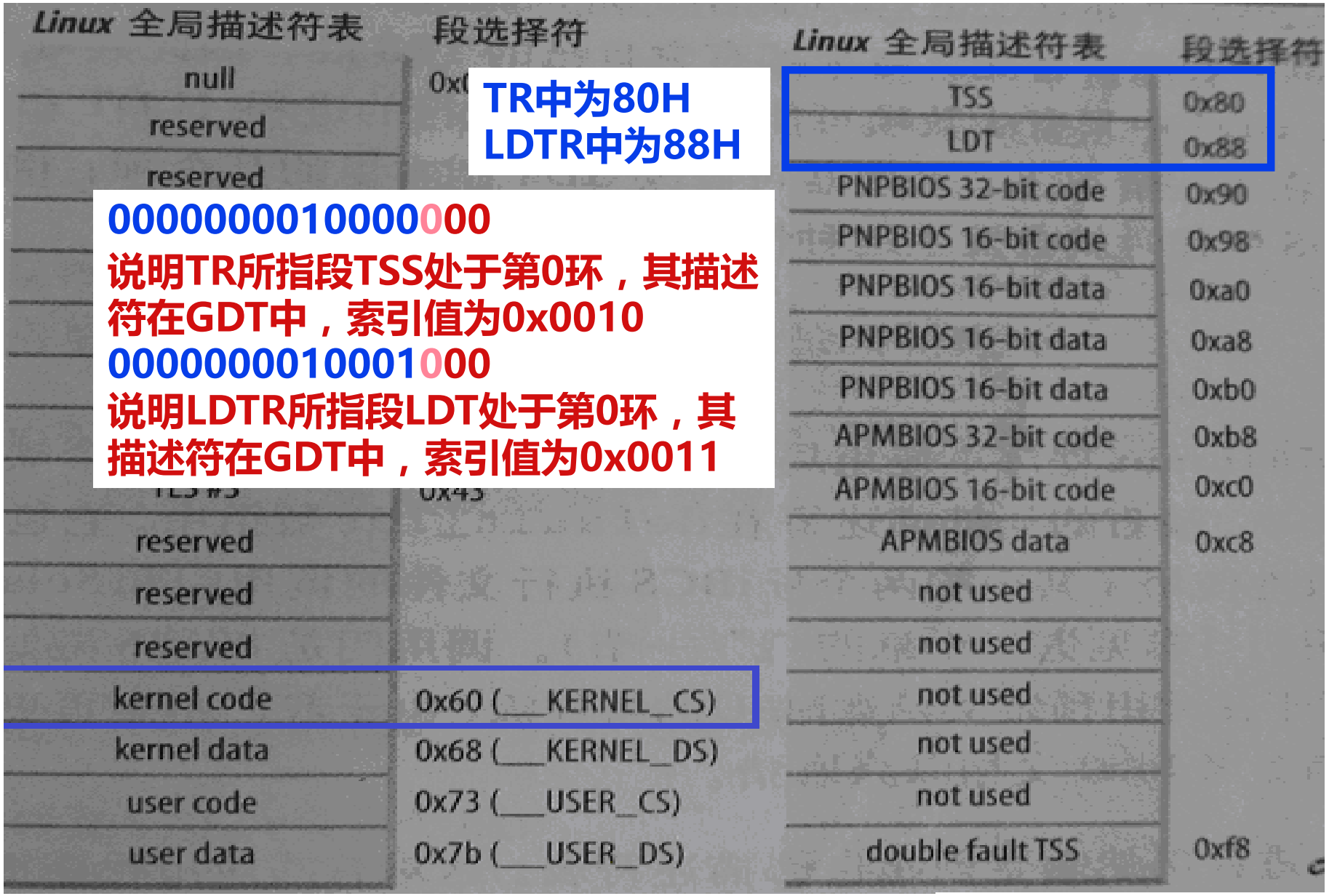
- B31~B0： 32位基地址； L19~L0：20位限界，表示段中最大页号
- G：粒度。G=1以页（4KB）为单位；G=0以字节为单位。因为界限为20位，故当G=0时最大的段为1MB；当G=1时，最大段为4KB×2^{20} =4GB
- D：D=1表示段内偏移量为32位宽，D=0表示段内偏移量为16位宽
- P：P=1表示存在，P=0表示不存在。Linux总把P置1，不会以段为单位淘汰
- DPL：访问段时对当前特权级的最低等级要求。因此，只有CPL为0（内核态）时才可访问DPL为0的段，任何进程都可访问DPL为3的段（0最高、3最低）
- S：S=0系统控制描述符，S=1普通的代码段或数据段描述符
- TYPE：段的访问权限或系统控制描述符类型
- A：A=1已被访问过，A=0未被访问过。（通常A包含在TYPE字段中）

用户不可见寄存器

- 为支持分段机制，CPU中有多个用户不可访问的内部寄存器，操作系统通过特权指令可对寄存器TR、LDTR、GDTR和IDTR进行读写

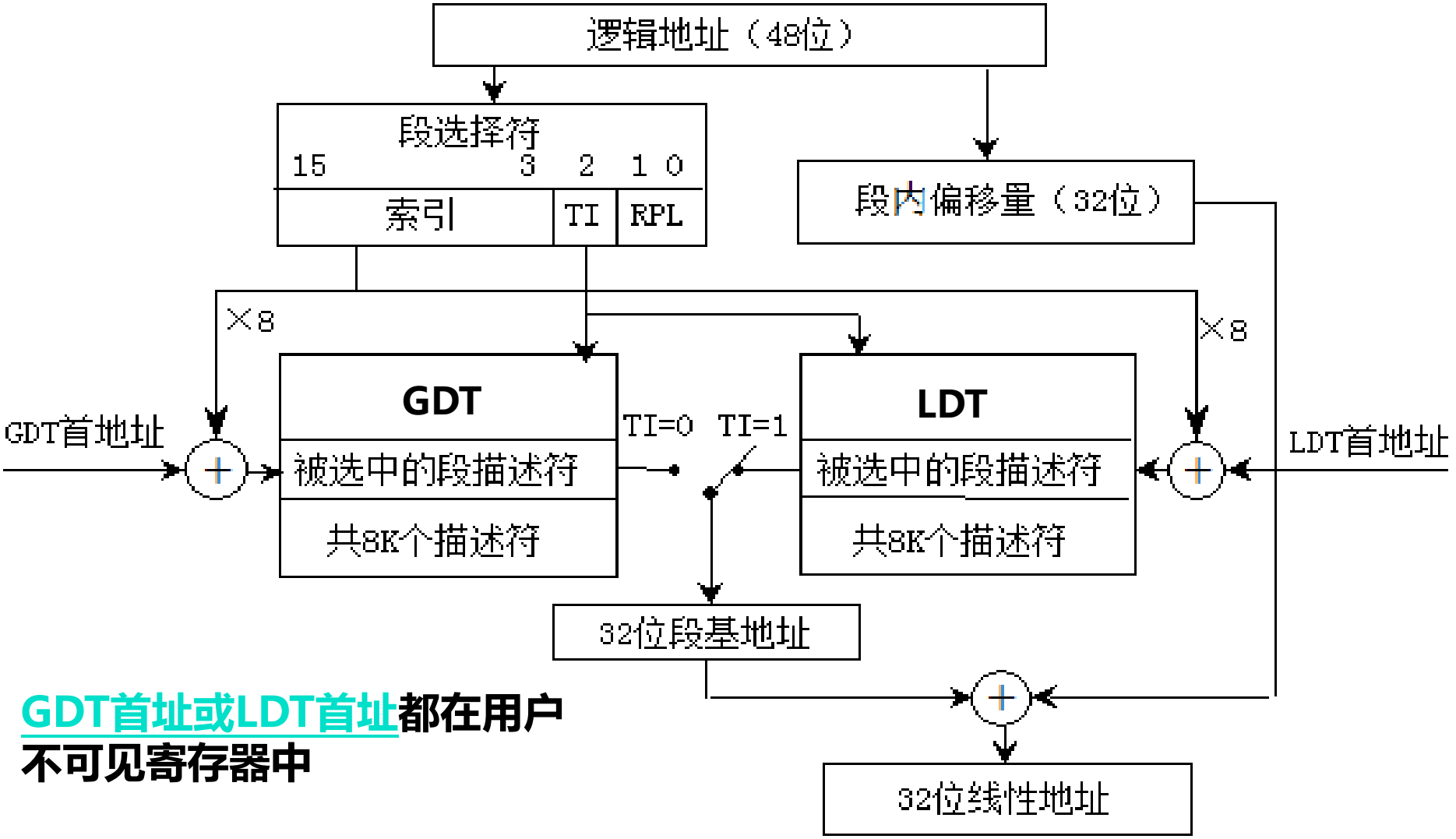


Linux的全局描述符表（GDT）



逻辑地址向线性地址转换

- 被选中的段描述符先被送至描述符cache，每次从描述符cache中取32位段基址，与32位段内偏移量（有效地址）相加得到线性地址



GDT首址或LDT首址都在用户不可见寄存器中

IA-32/Linux中的分段机制

- 为使能移植到绝大多数流行处理器平台， Linux简化了分段机制
- RISC对分段支持非常有限，因此Linux仅使用IA-32的分页机制，而对于分段，则通过在初始化时将所有段描述符的基址设为0来简化
- 若把运行在用户态的所有Linux进程（何意？）的代码段和数据段分别称为用户代码段和用户数据段；把运行在内核态的内核代码段和数据段分别称为内核代码段和内核数据段，则Linux初始化时，将上述4个段的段描述符中各字段设置成下表中的信息：

段	基地址	G	限界	S	TYPE	DPL	D	P
用户代码段	0x0000 0000	1	0xFFFFF	1	10	3	1	1
用户数据段	0x0000 0000	1	0xFFFFF	1	2	3	1	1
内核代码段	0x0000 0000	1	0xFFFFF	1	10	0	1	1
内核数据段	0x0000 0000	1	0xFFFFF	1	2	0	1	1

每个段都被初始化在0~4GB的线性地址空间中

初始化时，上述4个段描述符被存放在GDT中

Linux的全局描述符表（GDT）BACK

0000000001100000
说明内核代码段处于第0环，其描述符在GDT中，索引值为0x000C
0000000001101000
说明内核数据段处于第0环，其描述符在GDT中，索引值为0x000D
0000000001110011
说明用户代码段处于第3环，其描述符在GDT中，索引值为0x000E
0000000001111011
说明用户数据段处于第3环，其描述符在GDT中，索引值为0x000F

Linux 全局描述符表 段选择符

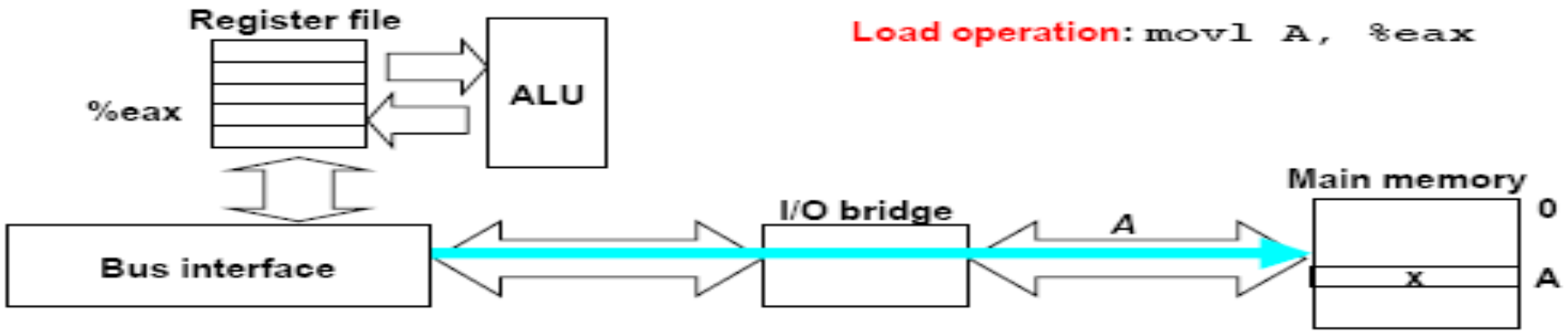
reserved	
kernel code	0x60 (__KERNEL_CS)
kernel data	0x68 (__KERNEL_DS)
user code	0x73 (__USER_CS)
user data	0x7b (__USER_DS)

Linux 全局描述符表 段选择符

TSS	0x80
LDT	0x88
PNPBIOS 32-bit code	0x90
PNPBIOS 16-bit code	0x98
PNPBIOS 16-bit data	0xa0
PNPBIOS 16-bit data	0xa8
PNPBIOS 16-bit data	0xb0
APMBIOS 32-bit code	0xb8
APMBIOS 16-bit code	0xc0
APMBIOS data	0xc8
not used	
not used	
not used	
not used	
not used	
double fault TSS	0xf8

回顾：指令“movl 8(%ebp), %eax”操作过程

由8(%ebp)得到主存地址A的过程较复杂，涉及MMU、TLB、页表等许多重要概念！



- IA-32/Linux中，执行“movl 8(%ebp), %eax”时，源操作数的逻辑地址向线性地址转换的过程如下：
  - 计算有效地址EA=R[ebp]+0×0+8
  - 取出段寄存器DS对应的描述符cache中的段基址（Linux中段基址为0）
  - 线性地址LA=段基址+EA=EA

逻辑地址向线性地址转换举例

- 已知变量y和数组a都是int型，a的首地址为0x8048a00。假设编译器将a的首地址分配在ECX中，数组的下标变量i分配在EDX中，y分配在EAX中，C语言赋值语句“y=a[i];”被编译为指令“movl (%ecx, %edx, 4), %eax”。若在IA-32/Linux环境下执行指令地址为0x80483c8的该指令时，CS段寄存器对应的描述符cache中存放的是表6.2中所示的用户代码段信息且CPL=3，DS段寄存器对应的描述符cache中存放的是表6.2中所示的用户数据段信息，则当i=100时，取指令操作过程中MMU得到的指令的线性地址是多少？取数操作过程中MMU得到的操作数的线性地址是多少？

```
int func(int a[ ], int c)
{
  int i , y = 0;
  for(i = 0; i<c; i++) {
        y= a[i];
        …
  }
  ……
}
```

400=511-111=511-(64+32+15)

=1 1111 1111B-(0110 1111B)

=1 1001 0000B = 190H

80483c8: movl (%ecx, %edx, 4), %eax

…代码和数据段DPL都为3，即CPL最低应为3，而CPL=3，故访问未越级

指令的线性地址：代码段基地址+EA=0+0x80483c8=0x80483c8

操作数的线性地址：数据段基地址+EA=0+R[ecx]+R[edx]×4

0x8048b90 =0x8048a00+100×4=0x8048e00 对吗？

IA-32的存储管理

- 按字节编址（通用计算机大都是）
- 在保护模式下，IA-32采用段页式虚拟存储管理方式
- 存储地址采用逻辑地址、线性地址和物理地址来进行描述，其中，逻辑地址和线性地址是虚拟地址的两种不同表示形式，描述的都是4GB虚拟地址空间中的一个存储地址
  - ✓逻辑地址由48位组成，包含16位段选择符和32位段内偏移量（即有效地址）
  - ✓线性地址32位（其位数由虚拟地址空间大小决定）
  - ✓物理地址32位（其位数由存储器总线中的地址线条数决定）
- 分段过程实现将逻辑地址转换为线性地址
- 分页过程实现将线性地址转换为物理地址 ⟵ 以下介绍分页机制

若页大小为4KB，每个页表项占4B，则理论上一个页表有多大？

因为$2^{32}/2^{12}=2^{20}$，故页表大小为4MB，比页还大！故采用多级页表方式

IA-32中的控制寄存器

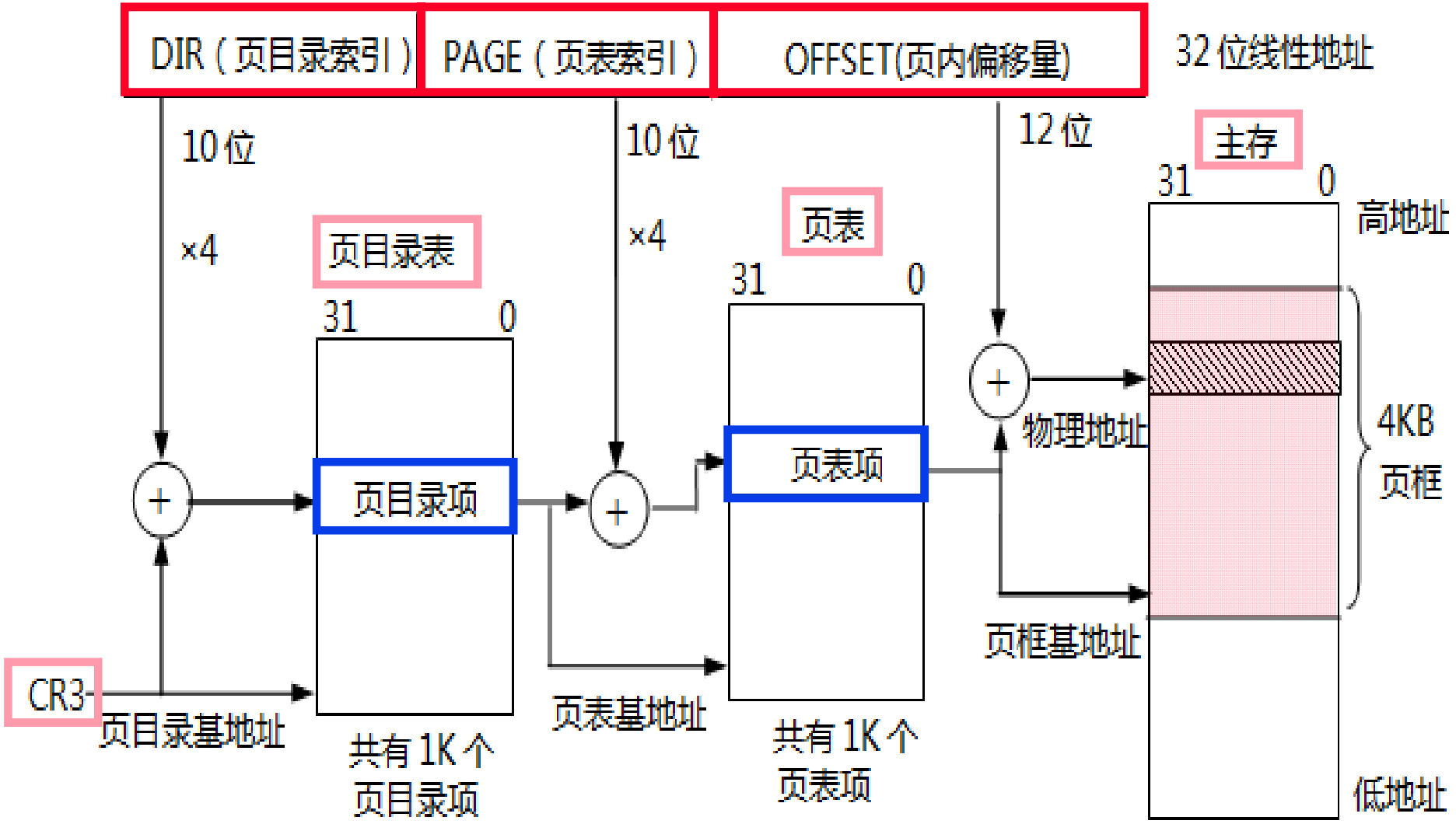
- 控制寄存器保存机器的各种控制和状态信息，它们将影响系统所有任务的运行，操作系统进行任务控制或存储管理时使用这些控制和状态信息。
  - CR0：控制寄存器
    - ① PE: 1为保护模式。一旦在保护模式，不能再将PE清0，只能重启系统以回到实模式。② PG：1-启用分页；0-禁止分页，此时线性地址被直接作为物理地址使用。若要启用分页机制，则PE和PG都要置1。③ 任务切换位TS：任务切换时将其置1，切换完毕则清0，可用CLTS指令将其清0。④ 对齐屏蔽位AM。⑤ cache功能控制位NW（（Not Write-through）和CD（Cache Disable）。只有当NW和CD均为0时，cache才能工作。
  - CR2：页故障（page fault）线性地址寄存器
    - 存放引起页故障的线性地址。只有在CR0中的PG=1时，CR2才有效。
  - CR3：页目录基址寄存器
    - 保存页目录表的起始地址。只有当CR0中的PG=1时，CR3才有效。

IA-32采用两级页表方式，第一级页表称为页目录表

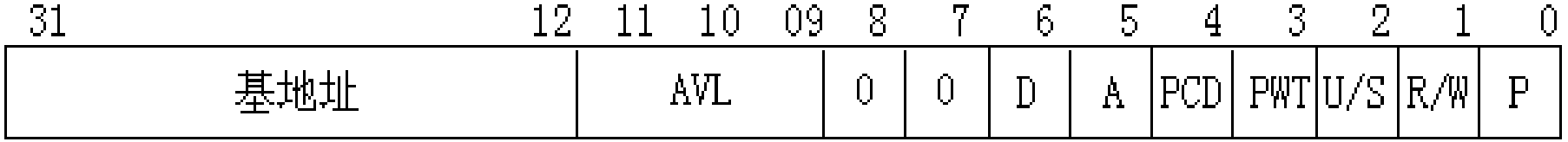
线性地址向物理地址转换

线性地址空间划分：4GB=1K个子空间 * 1K个页面/子空间 * 4KB/页

- **页目录项和页表项格式一样，有32位（4B）**



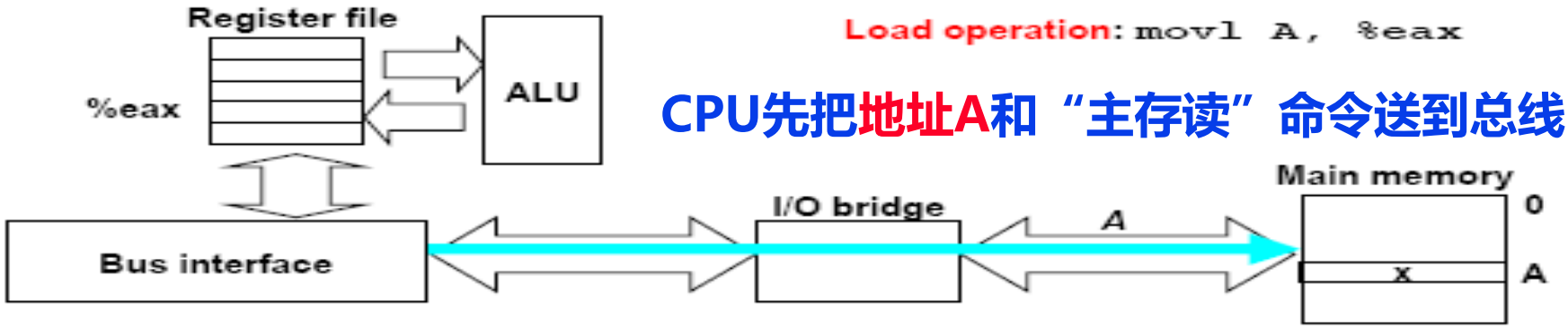
IA-32的页目录项和页表项



- P：1表示页表或页在主存中；P=0表示页表或页不在主存，即缺页，此时需将页故障线性地址保存到CR2。
- R/W：0表示页表或页只能读不能写；1表示可读可写。
- U/S：0表示用户进程不能访问；1表示允许访问。
- PWT：控制页表或页的cache写策略是全写还是回写（Write Back）。
- PCD：控制页表或页能否被缓存到cache中。
- A：1表示指定页表或页被访问过，初始化时OS将其清0。利用该标志，OS可清楚了解哪些页表或页正在使用，一般选择长期未用的页或近来最少使用的页调出主存。由MMU在进行地址转换时将该位置1。
- D：修改位(脏位dirty bit)。页目录项中无意义，只在页表项中有意义。初始化时OS将其清0，由MMU在进行写操作的地址转换时将该位置1。
- 高20位是页表或页在主存中的首地址对应的页框号，即首地址的高20位。每个页表的起始位置都按4KB对齐。

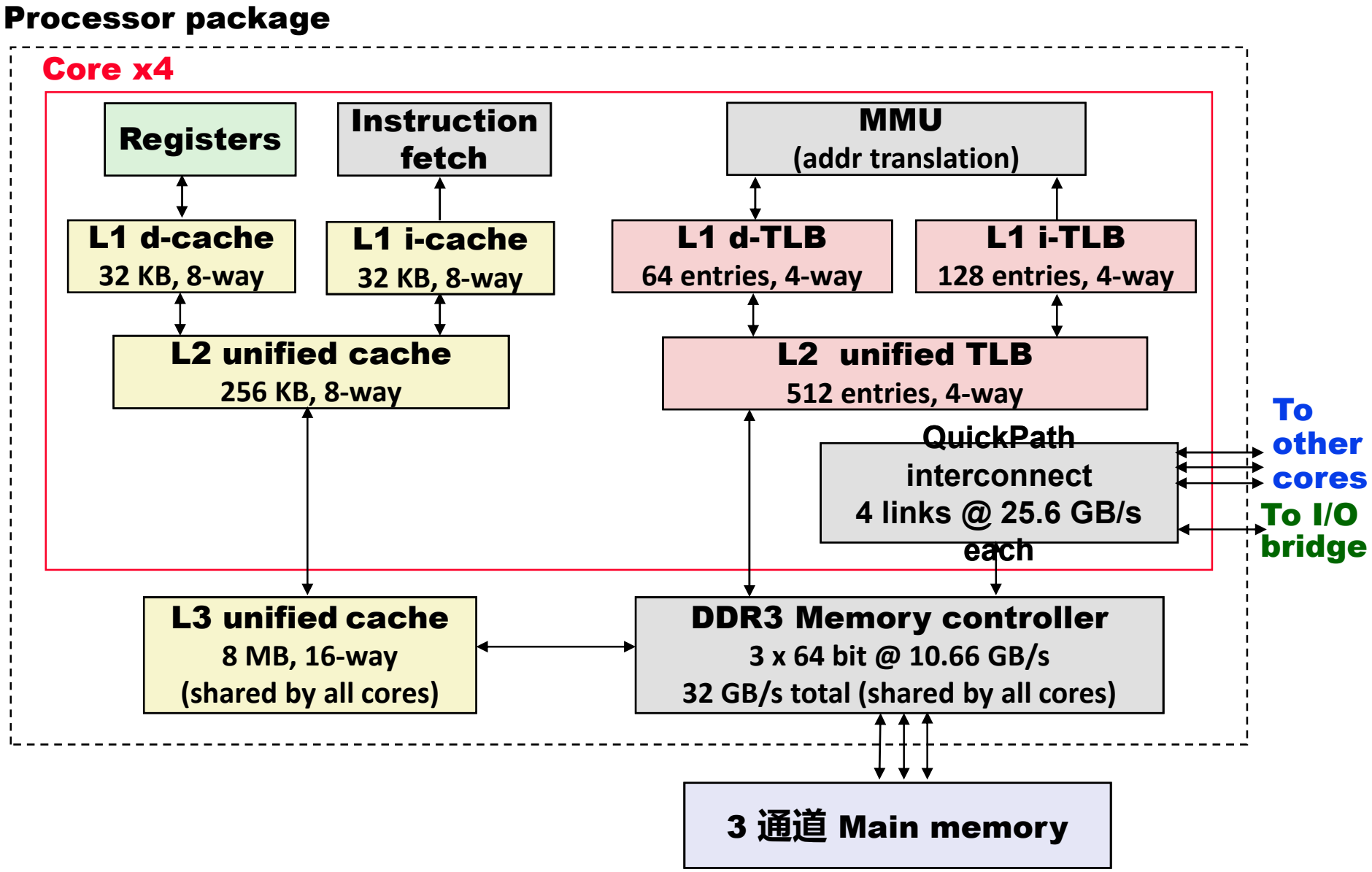
回顾：指令“movl 8(%ebp), %eax”操作过程

由8(%ebp)得到主存地址A的过程较复杂，涉及MMU、TLB、页表等许多重要概念！

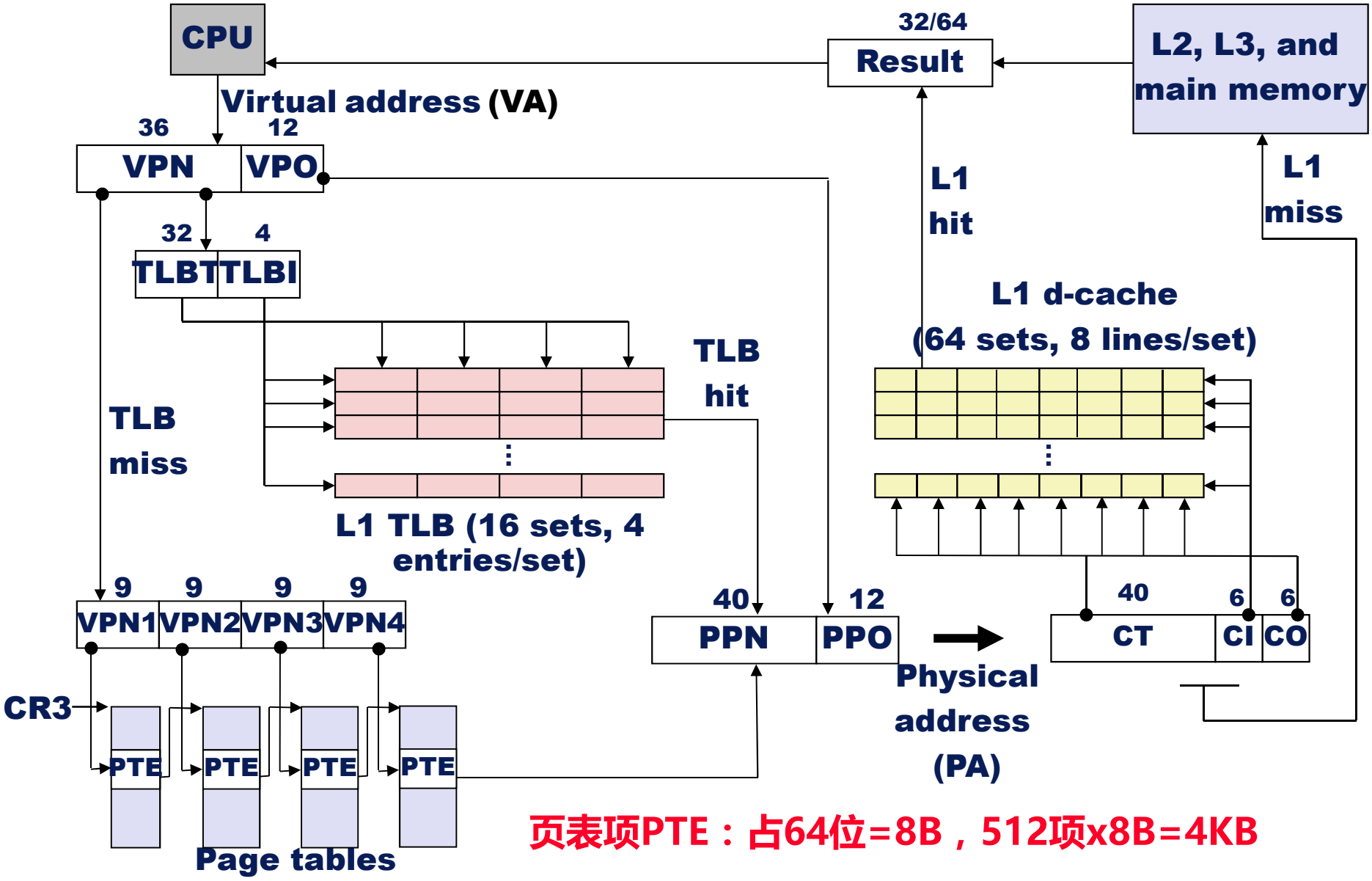


- IA-32中，执行“movl 8(%ebp), %eax” 中取数操作的大致过程如下：
  - 若CPL>DPL则越级，否则计算有效地址EA=R[ebp]+0×0+8
  - 通过段寄存器找到段描述符以获得段基址，线性地址LA=段基址+EA
  - 若“LA>段限”则越界，否则将LA转换为主存地址A
    - 若访问TLB命中则地址转换得到A；否则处理TLB缺失（硬件/OS）
    - 若缺页或越权(R/W不符)则调出OS内核；否则地址转换得到A
    - 根据A先到Cache中找，若命中则取出A在Cache中的副本
    - 若Cache不命中，则再到主存取A所在主存块送对应Cache行

补充：Intel Core i7 Memory System

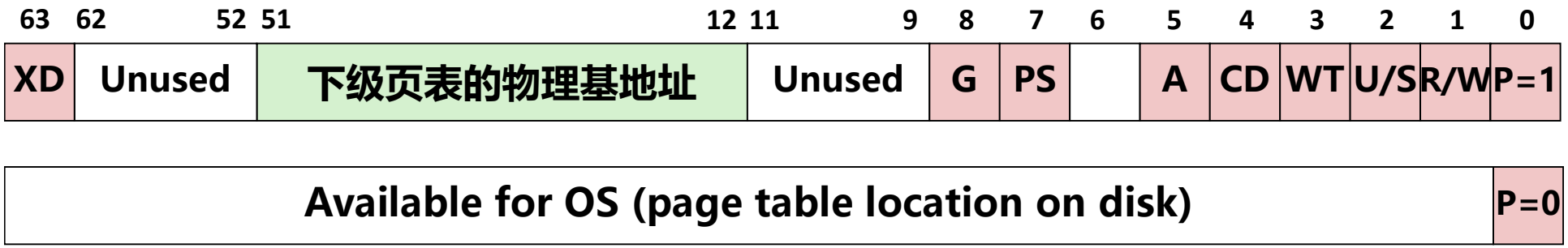


End-to-end Core i7 Address Translation



页表项PTE：占64位=8B，512项x8B=4KB

Core i7 Level 1-3 Page Table Entries



Each entry references a 4KB child page table

P: Child page table present in physical memory (1) or not (0).

R/W: Read-only or read-write access permission for all reachable pages.

U/S: user or supervisor (kernel) mode access permission for all reachable pages.

WT: Write-through or write-back cache policy for the child page table.

CD: Caching disabled or enabled for the child page table.

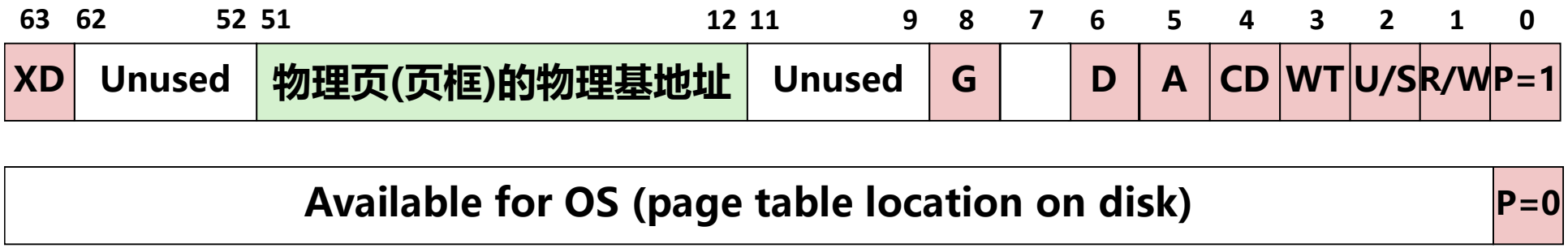
A: Reference bit (set by MMU on reads and writes, cleared by software).

PS: Page size either 4 KB or 4 MB (defined for Level 1 PTEs only).

G: Global page (don't evict from TLB on task switch)

Page table physical base address: 40 most significant bits of physical page table address (forces page tables to be 4KB aligned)

Core i7 Level 4 Page Table Entries



Each entry references a 4KB child page

P: Child page table present in physical memory (1) or not (0).

R/W: Read-only or read-write access permission for all reachable pages.

U/S: user or supervisor (kernel) mode access permission for all reachable pages.

WT: Write-through or write-back cache policy for the child page table.

CD: Caching disabled or enabled for the child page table.

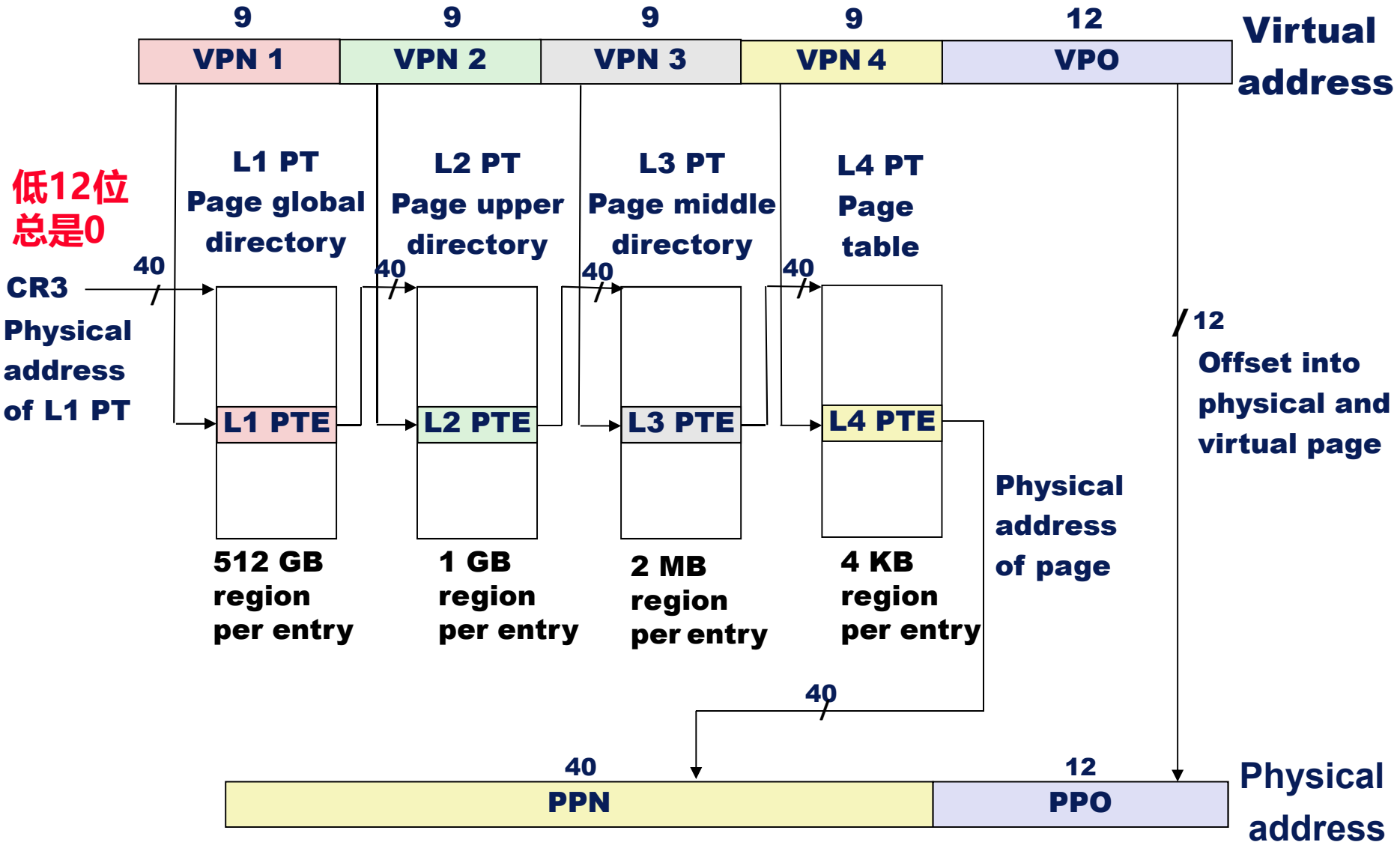
A: Reference bit (set by MMU on reads and writes, cleared by software).

D: Dirty bit (set by MMU on writes, cleared by software)

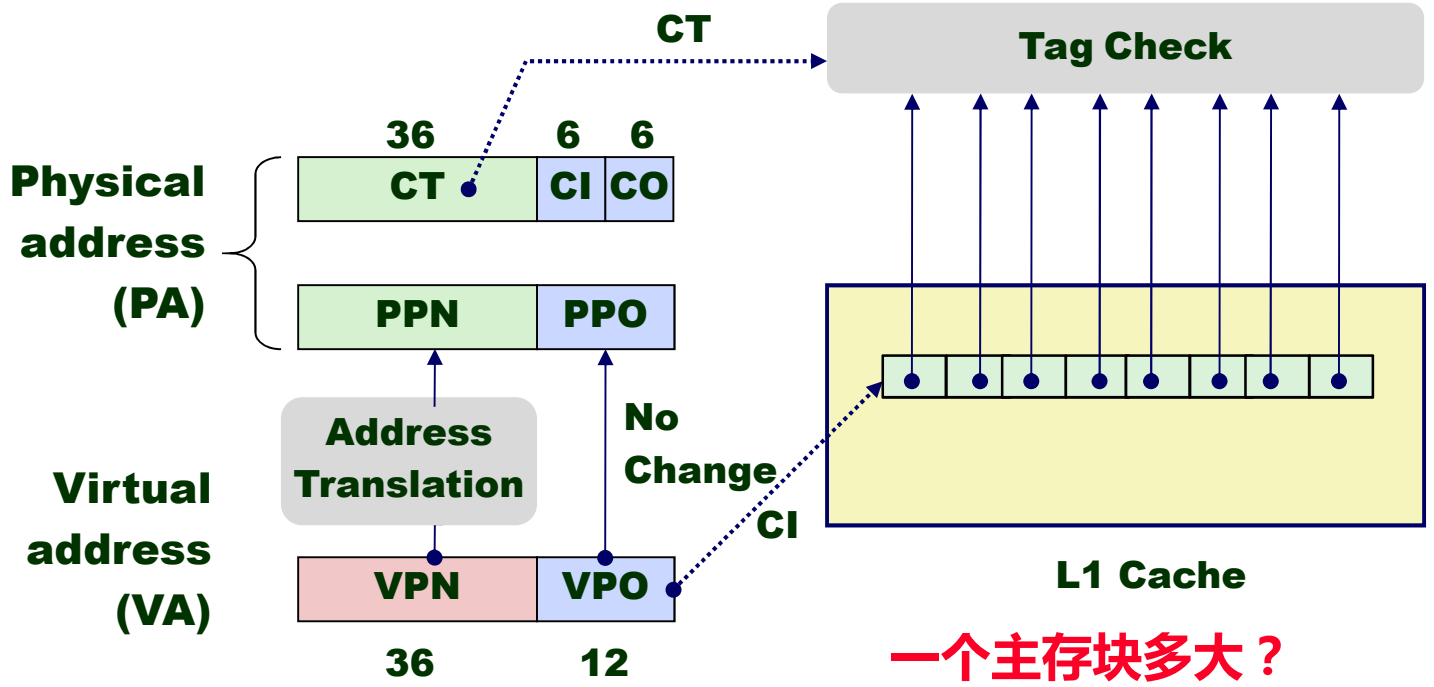
G: Global page (don't evict from TLB on task switch)

Page physical base address: 40 most significant bits of physical page address (forces pages to be 4KB aligned)

Core i7 Page Table Translation



Cute Trick for Speeding Up L1 Access

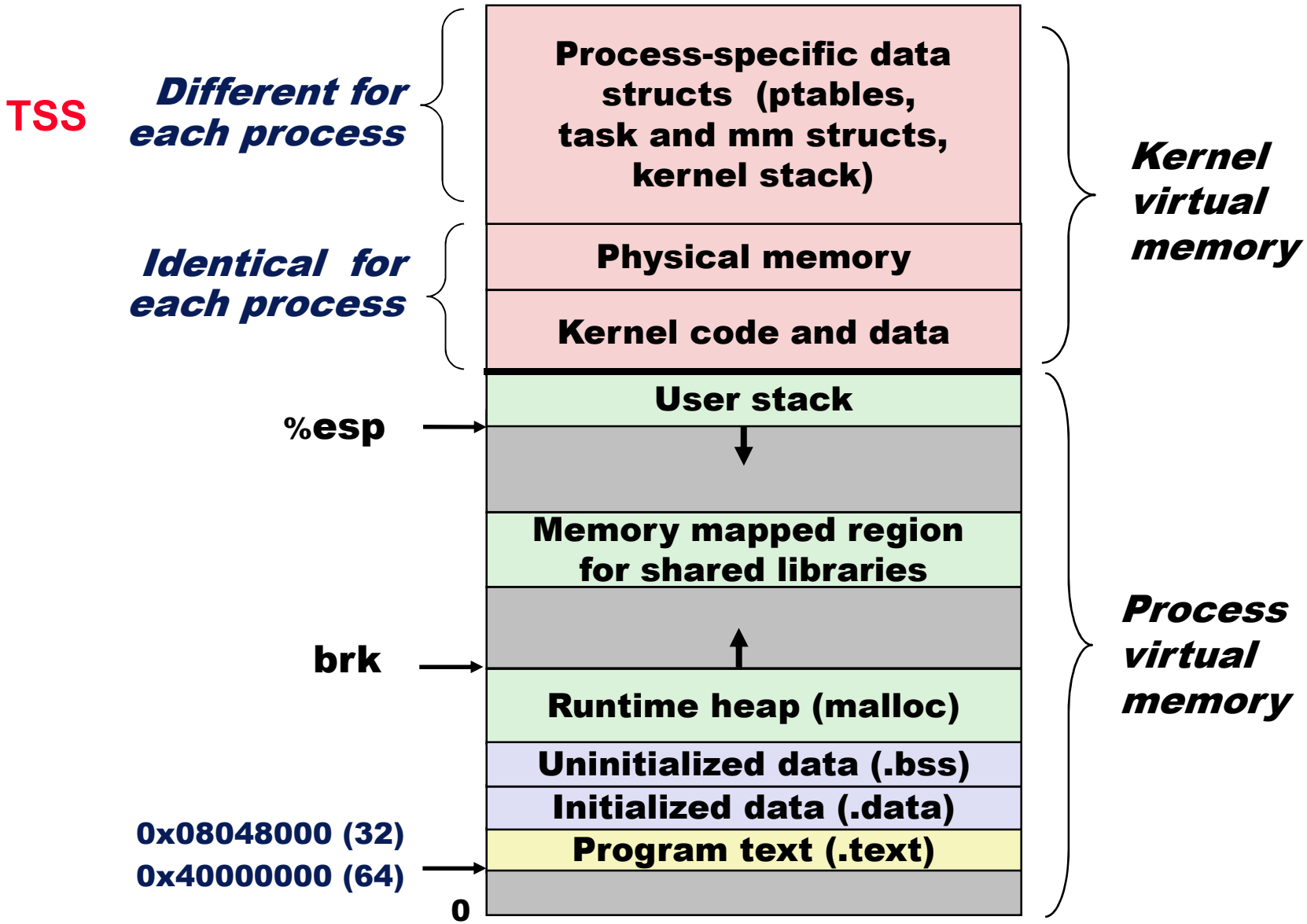


一个主存块多大？

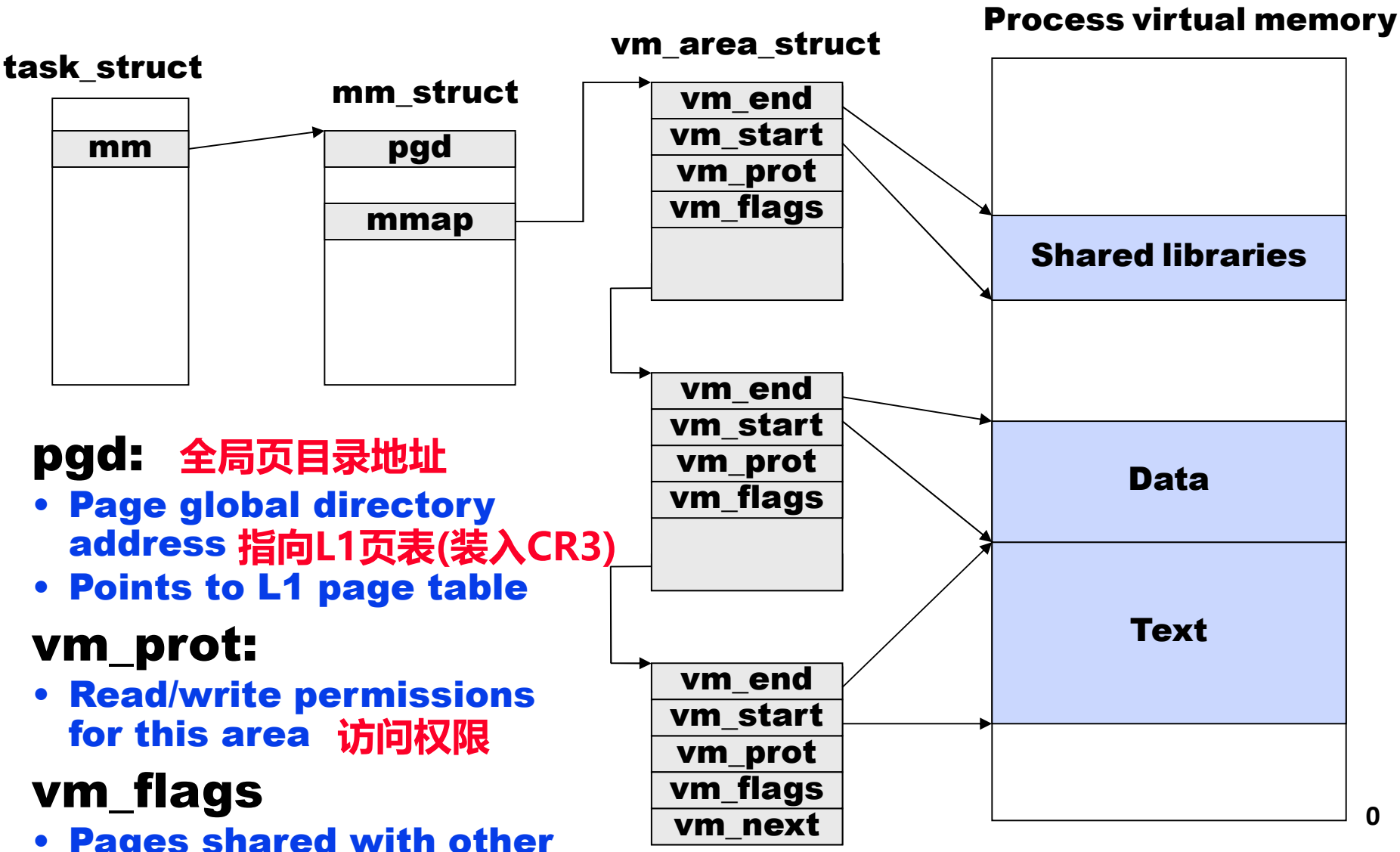
64B

- **Observation**
  - **Bits that determine CI identical in virtual and physical address**
  - **Can index into cache while address translation taking place**
  - **Generally we hit in TLB, so PPN bits (CT bits) available next**
  - **“Virtually indexed, physically tagged”**
  - **Cache carefully sized to make this possible**

Virtual Memory of a Linux Process

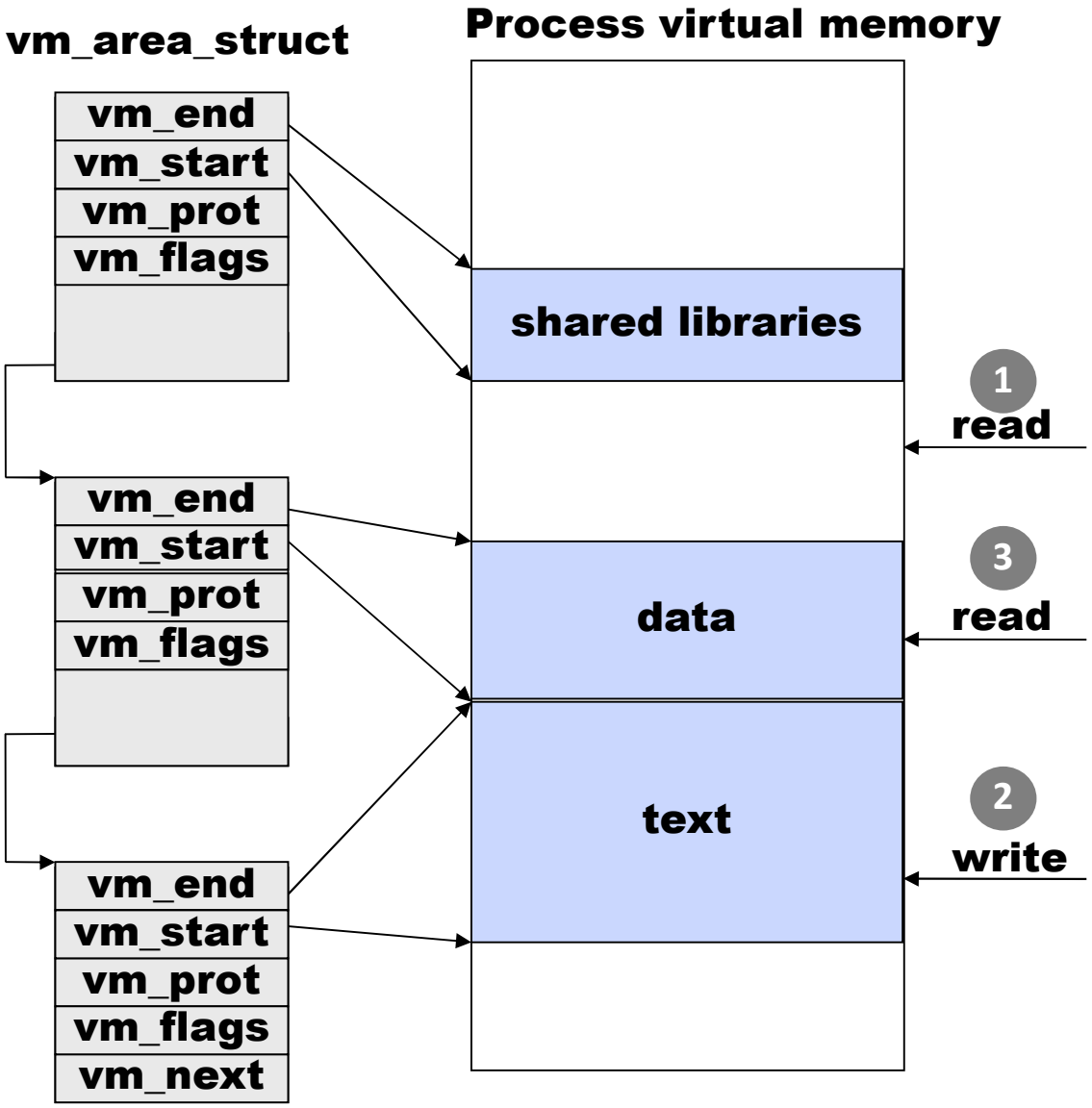


Linux将虚存空间组织成“区域(area)”



- **pgd:** 全局页目录地址
  - Page global directory address 指向L1页表(装入CR3)
  - Points to L1 page table
- **vm_prot:**
  - Read/write permissions for this area 访问权限
- **vm_flags**
  - Pages shared with other processes or private to this process 是共享还是本进程私有

Linux Page Fault Handling



Linux页故障类型

Segmentation fault:
accessing a non-existing page

Normal page fault
not in memory

Protection exception:
e.g., violating permission by writing to a read-only page (Linux reports as Segmentation fault)

本周小结

- IA-32的寻址方式中，除立即、寄存器寻址外，其他都是存储器寻址方式。
- IA-32指令给出逻辑地址：16位段选择符+32位有效地址（段内偏移量）
- 16位段选择符在CS、DS等段寄存器中，高13位为段描述表索引，低两位为特权级（00-内核级、11-用户级）
- 段描述符表（GDT或LDT）中给出每个段的描述信息，包括段基址等
- Linux简化了段描述信息：
  - 内核代码段、内核数据段、用户代码段、用户数据段的基地址都为0；段内最大地址都是FFFFFFFFH
  - 内核代码和数据段、用户代码和数据的访问特权级不同
- 线性地址= 段基址+有效地址（段内偏移量）
- 线性地址（虚拟地址）到物理地址的转换采用分页方式
- 通常情况下，页内地址为12位，每页占4KB，IA-32的虚拟地址为32位，采用两级页表，Core i7的虚拟地址占48位，采用4级页表