4-CPU寻址

搜狐焦点计算机基础学习系列课程

李少鹏 2019-03

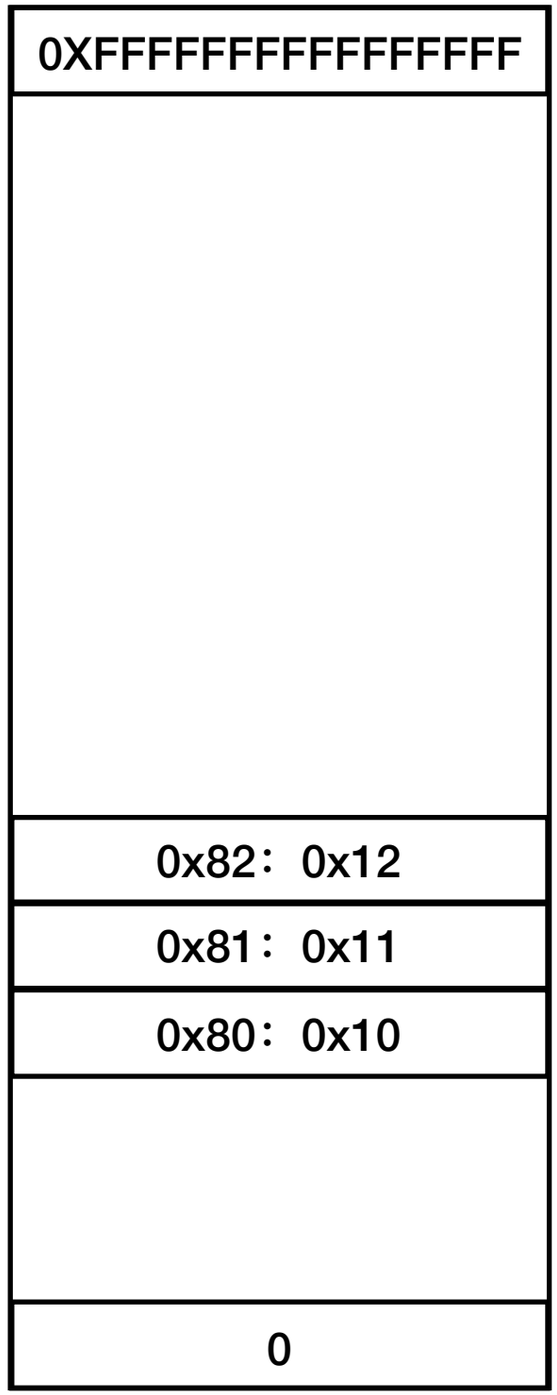
CPU寻址的定义：从存储器中找到数据

通用寄存器组：64位

RAX	RBX	RCX	RDX
RSP	RBP	RSI	RDI
R8	R9	R10	R11
R12	R13	R14	R15

RBX = 0x80 RCX = 0x1

找到寄存器或内存中的特定位置数据



CPU寻址的定义：汇编指令

寻址类型	数据位置	汇编格式	操作数值	示例	RAX存储值
立即数寻址	立即数	$Imm	Imm	movq $0x4f,%rax	0x4f
寄存器寻址	寄存器	%r	R[%r]	movq %rbx,%rax	0x80
绝对寻址	内存	Imm	M[Imm]	movq 0x80,%rax	0x10
间接寻址	内存	(%r)	M[R[%r]]	movq (%rbx),%rax	0x10
基址偏移量寻址	内存	Imm(%r)	M[Imm+R[%r]]	movq 2(%rbx),%rax	0x12
变址寻址	内存	(%r1,%r2)	M[R[%r1]+R[%r2]]	movq (%rbx,%rcx),%rax	0x11

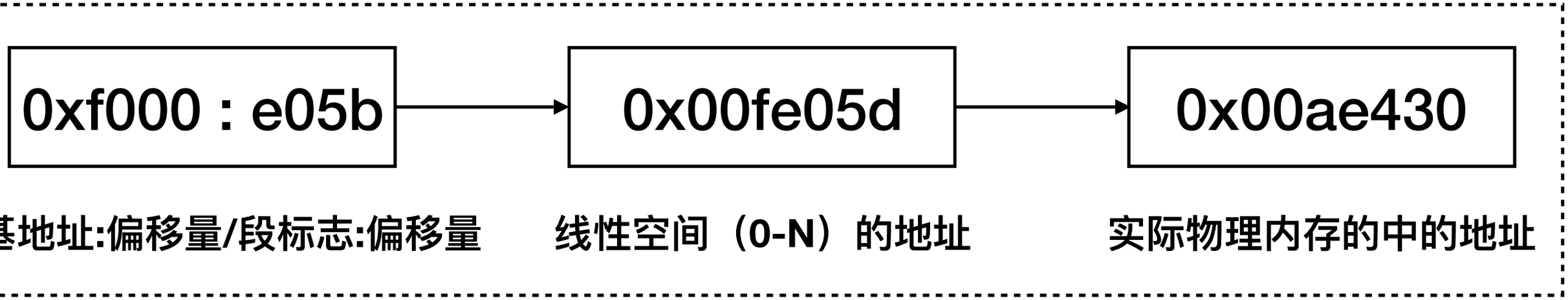
R[%r] = 取出寄存器r中存储的数值

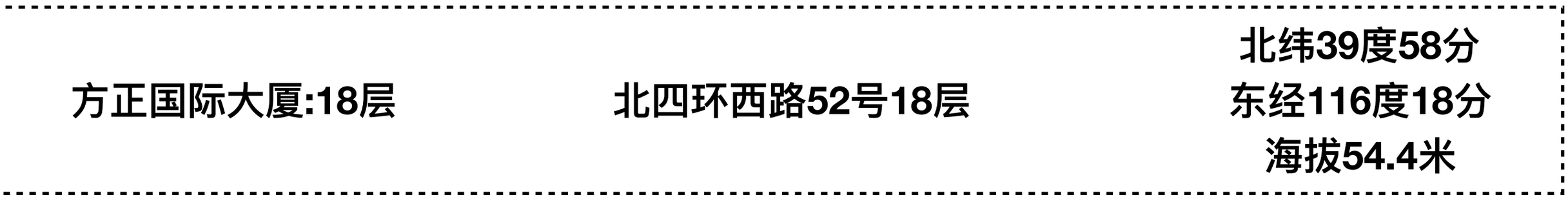
M[Imm] = 取出内存中物理地址Imm中的数值

X86-64汇编语言的几类寻址语法（ATT规则）

CPU寻址的基本机制


逻辑地址
实模式分段
保护模式分段
线性地址
相等
分页
物理地址


0xf000 : e05b
0x00fe05d
0x00ae430
基地址:偏移量/段标志:偏移量
线性空间（0-N）的地址
实际物理内存的中的地址


方正国际大厦:18层
北四环西路52号18层
北纬39度58分
东经116度18分
海拔54.4米

CPU寻址的目标：多任务系统

多任务系统的三个核心特征

权限分级	数据隔离	任务切换

CPU寻址功能是CPU提供给内核的主要接口
内核通过对CPU寻址功能的包装和调用来实现多任务系统

CPU寻址的目标：类比

如何做到有条不紊的同时和多个女朋友交往

1、给女朋友分级别（A>B=C）：
B的要求不能和A的要求冲突（巧妙获得A的授权）
如果A发现了B，则放弃B

2、B、C之间是隔离的，相互不能出现的重合

3、从和B相处切换到和C相处的时候：
把痕迹清理干净
把和B相处用到的东西藏好，以便下次拿出来继续用

目录

多模式机制

分段机制

权限分级的实现

分页机制

数据隔离：线性地址到物理地址的映射

中断机制

任务机制

任务数据结构以及任务的硬切换

多模式机制：X86-64CPU的5种操作模式

实地址模式（Real-address mode）
CPU接电即进入的模式（初始化之前的模式），只提供8086处理器的编程环境

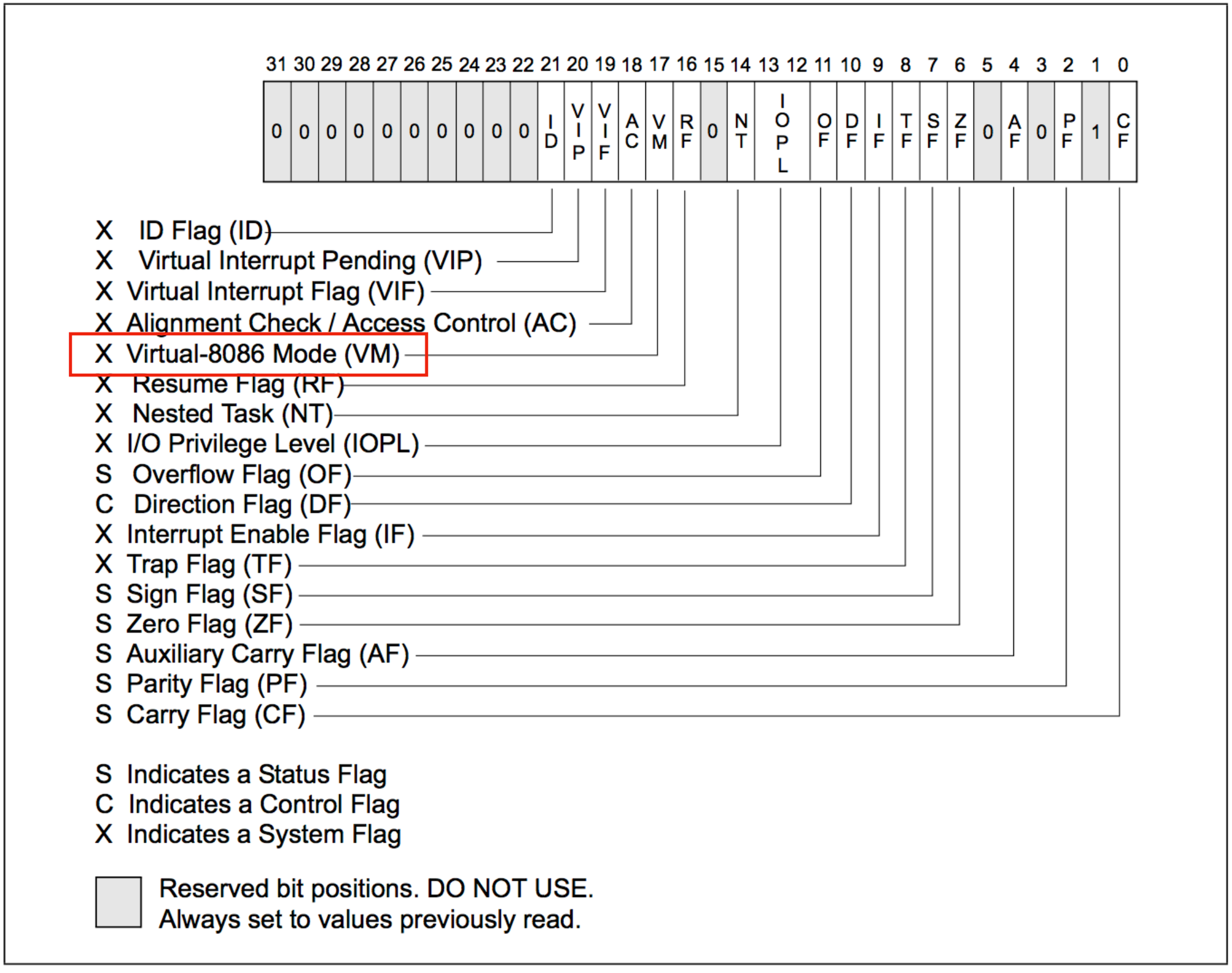
保护模式（Protected mode）
X86CPU的正常运行模式，提供了多任务运行环境

扩展64位模式（IA-32e mode）
X86-64CPU的正常运行模式，是X86的64位扩展

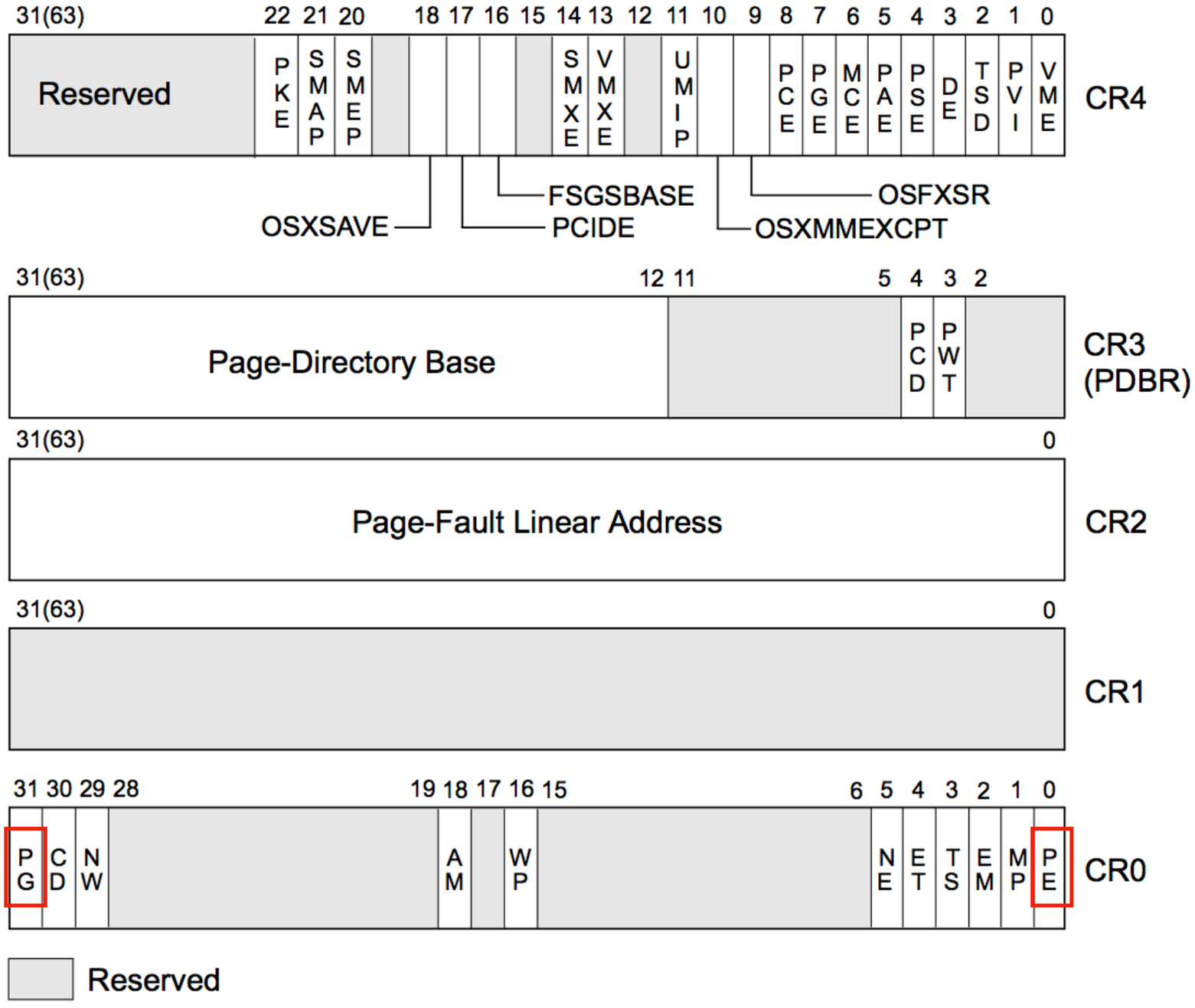
虚拟8086模式（Virtual-8086 mode）
允许CPU在多任务环境下执行8086体系下的软件

系统管理模式（System management mode）
在此模式下可以进行电源管理和一些功能特性的设置

多模式机制：FLAGS寄存器



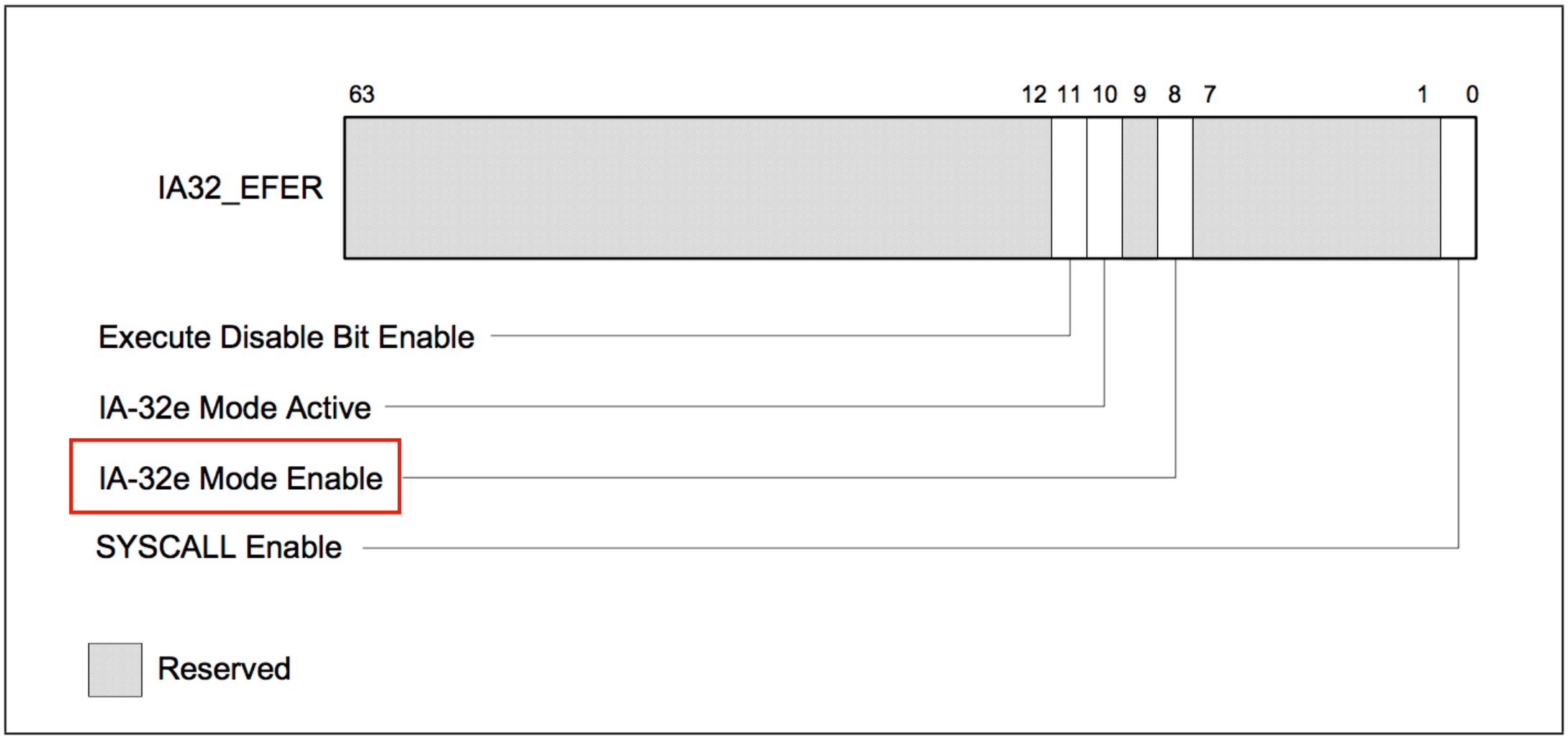
多模式机制：控制寄存器（CR0-CR4）



CR0.PG : Paging

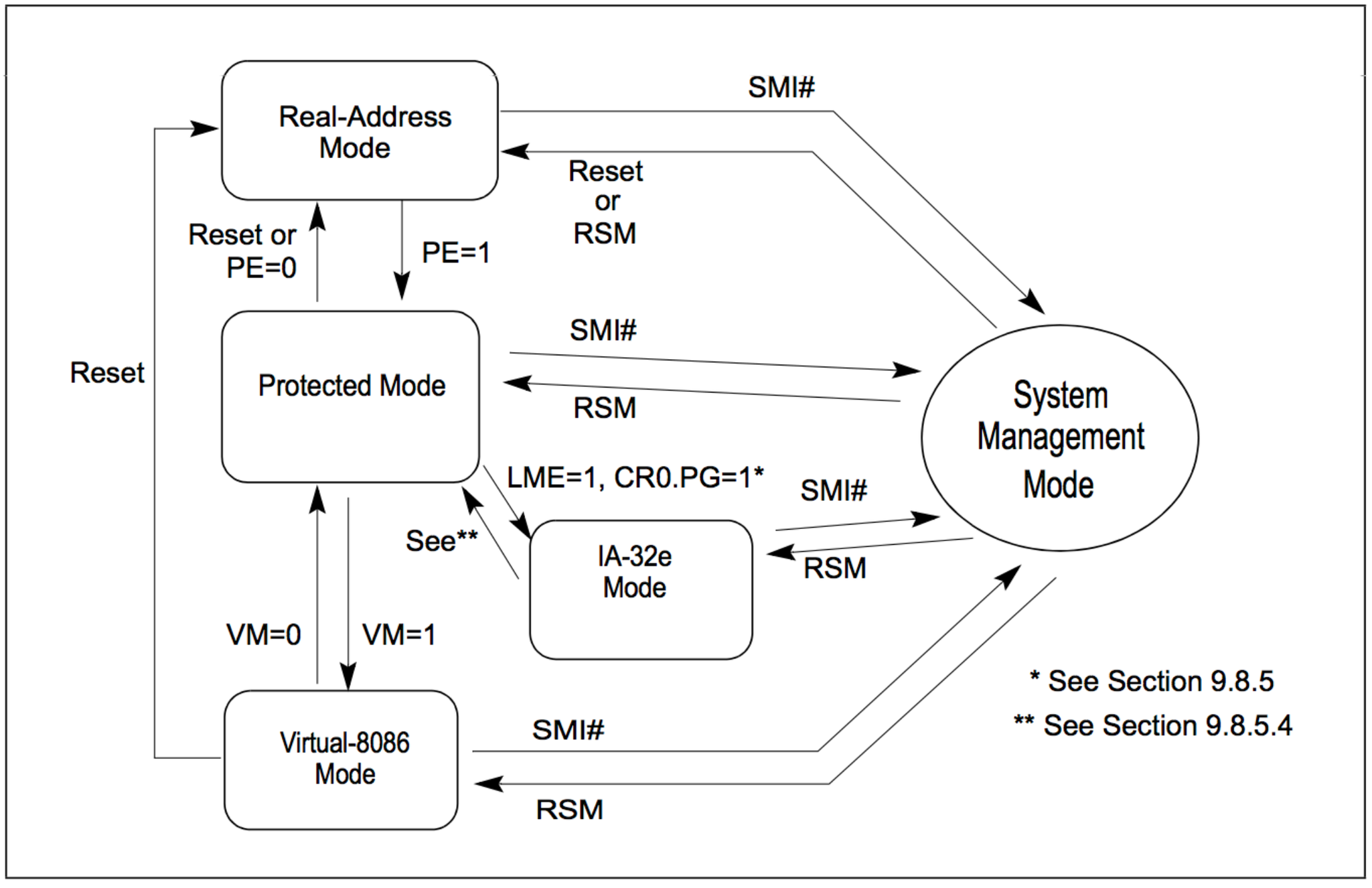
CR0.PE : Protection Enable

多模式机制：控制寄存器（EFER）



EFER.lme : IA-32e Mode Enable

多模式机制：5种模式之间的切换



多模式机制：核心模式之间的差异

	保护启用 CR0.PE=1?	分页启用 CR0.PG=1?	扩展模式启用 EFER.LME=1?	线性地址空间	物理地址空间
实地址模式	否	否	否	1M	1M
保护模式	是	是/否	否	4G（2^32）	64G（2^36）
扩展64位模式	是	是	是	4096P（2^64）	64T（2^46）

1P = 2^10T = 2^20G = 2^30M = 2^40K = 2^50

多模式机制：核心模式之间的差异

分段机制：实模式分段和保护模式分段

	实模式分段	保护模式分段
目标	用16位总线寻址20位空间	将线性空间分为不同的段 区分读写操作权限
段寄存器用途	基地址	段选择符（段描述符的索引） 标志读写权限级别
适用模式	实地址模式	保护模式 扩展64位模式
段大小	64KB	1B - 1MB 或者 4KB - 4GB

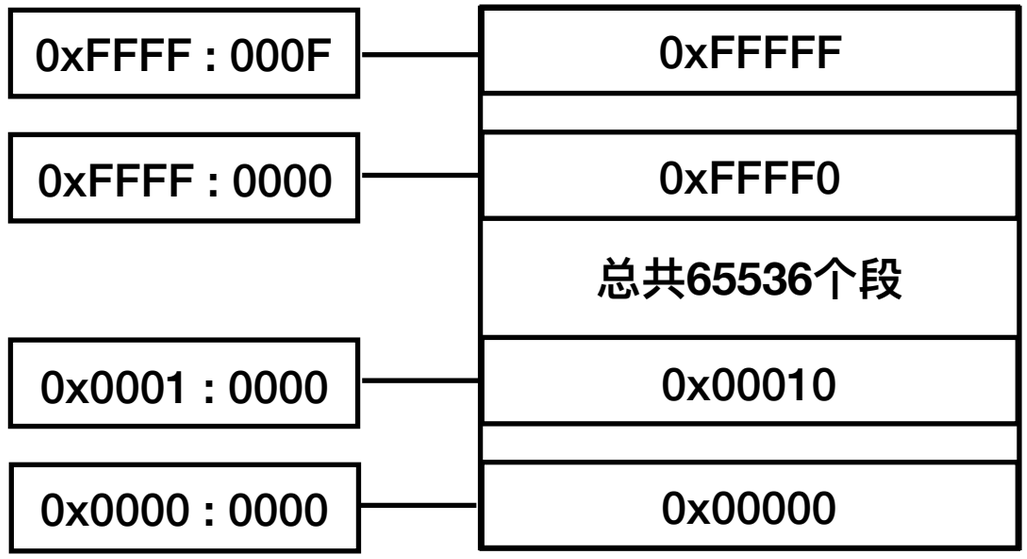
分段机制：实模式分段

定义：物理地址（20位） = 基地址（16位）* 16 + 偏移量（16位）

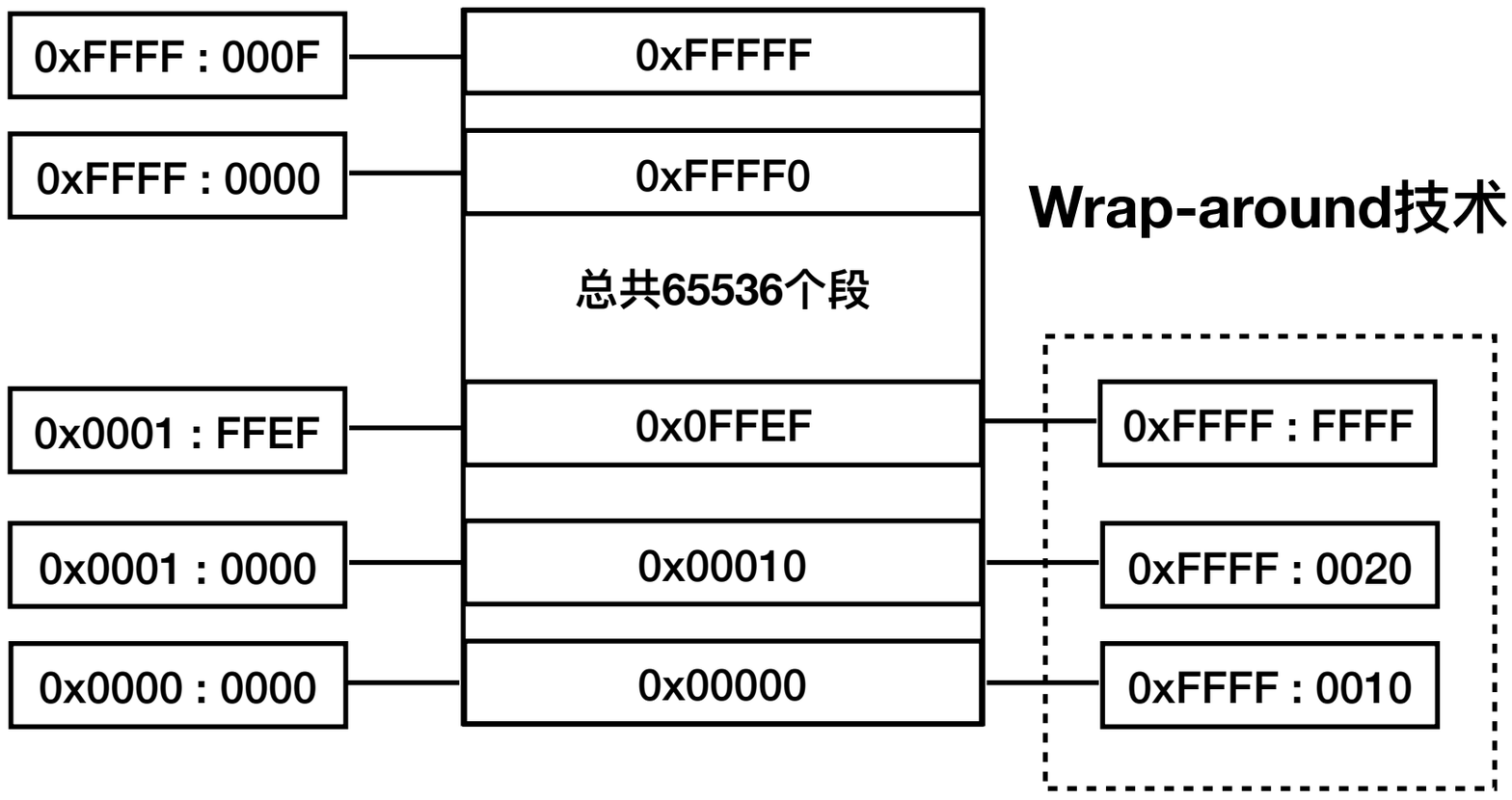
举例：0xf000 : e05b = 0xfe05b

指令举例：JMPF 0xf000 : e05b（指令跳转到物理地址0xfe05b处）

指令举例：MOV byte ptr ds : 0x04b0, bl（把bl的内容放到物理地址ds*16+0x04b0处）



分段机制：实模式分段



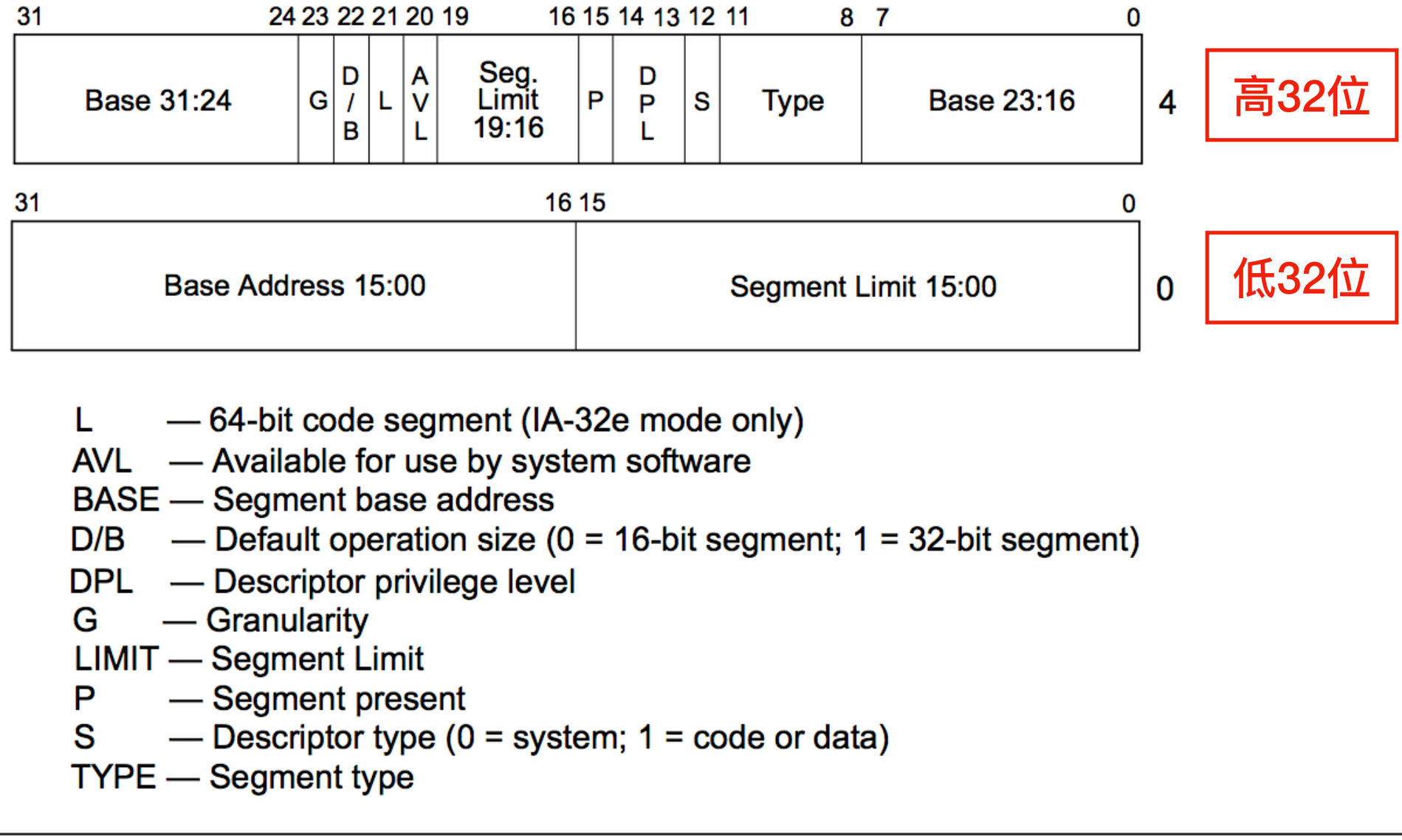
用两个16位（64K）数（基地址&偏移量），在20位（1M）位物理空间中寻址

空间可分为2^16个段（65536），每段含有2^16个Byte（65536）

分段机制：保护模式分段-核心概念

段描述符	Segment Descriptor，32模式下位64位，64位下128位，定义了线性地址空间中的一个连续区域，以及这个区域的属性
段选择符	Segment Selector，16位，定了指向一个段描述符的索引，以及这个索引的属性
全局描述符表	GDT（Global Descriptor Table），由多个段描述符首位相接排列成的表，给所有任务用；头部地址在GDTR寄存器中
局部描述符表	LDT（Local Descriptor Table），由多个段描述符首位相接排列成的表，给单个任务用；段选择符在LDTR寄存器中

分段机制：保护模式分段-段描述符含义



IA-32描述符为64位，IA-32e描述符为128位

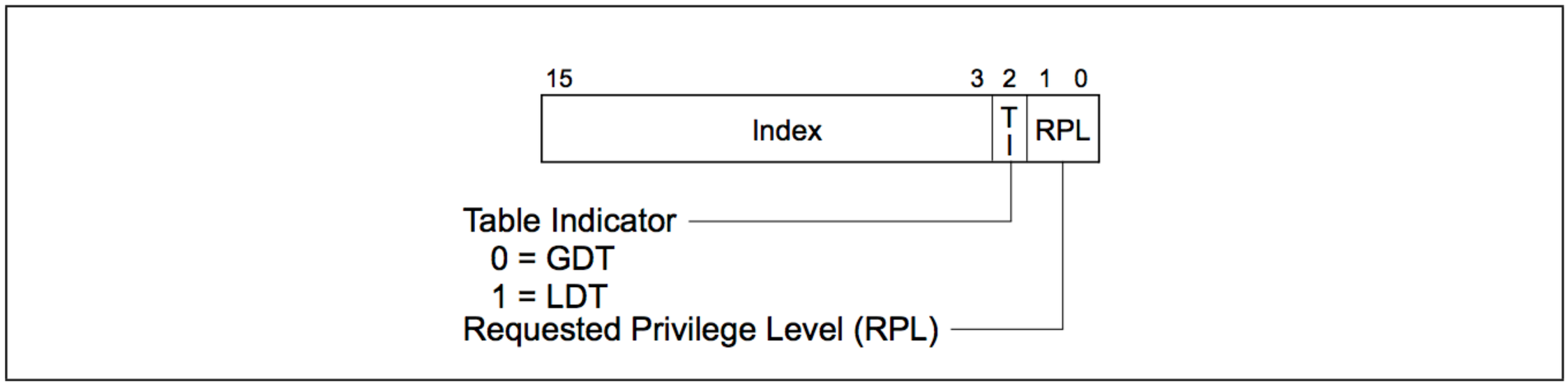
分段机制：保护模式分段-段描述符含义

标志	大小	含义
BASE	32	在线性地址空间中，段的起始地址
LIMIT	20	段的大小；如果G=0则在1B-1MB之间，如果G=1则在4KB-4GB之间
TYPE	4	段的类型（数据/代码）；增长方向（上/下）；是否可读可写可执行
S	1	S=1普通段（数据/代码）；S=0系统段
DPL	2	访问该段所需要权限级别（0-4）
P	1	P=1（段在物理内存中）；P=0（段不在物理内存中）
AVL	1	保留字段（给操作系统）
L	1	在非IA-32e模式或者非代码段中，L=0；否则，1表示Intel64模式，0兼容模式
D/B	1	使用32位偏移地址（1：寻址4G）还是16位偏移地址（0：寻址64K）
G	1	G=0，LIMIT单位为1B；G=1，LIMIT最小单位为4KB

分段机制：保护模式分段-段分类

名称	类型	S取值	TYPE取值	说明
代码段	段描述符	1	1???	用于存储可以执行的代码
数据段	段描述符	1	0???	用于存放数据，堆数据（DS），栈数据（SS）
LDT段	段描述符	0	0010	存储任务的LDT表
任务段	段描述符	0	?0?1	存储任务的状态数据

分段机制：保护模式分段-段选择符



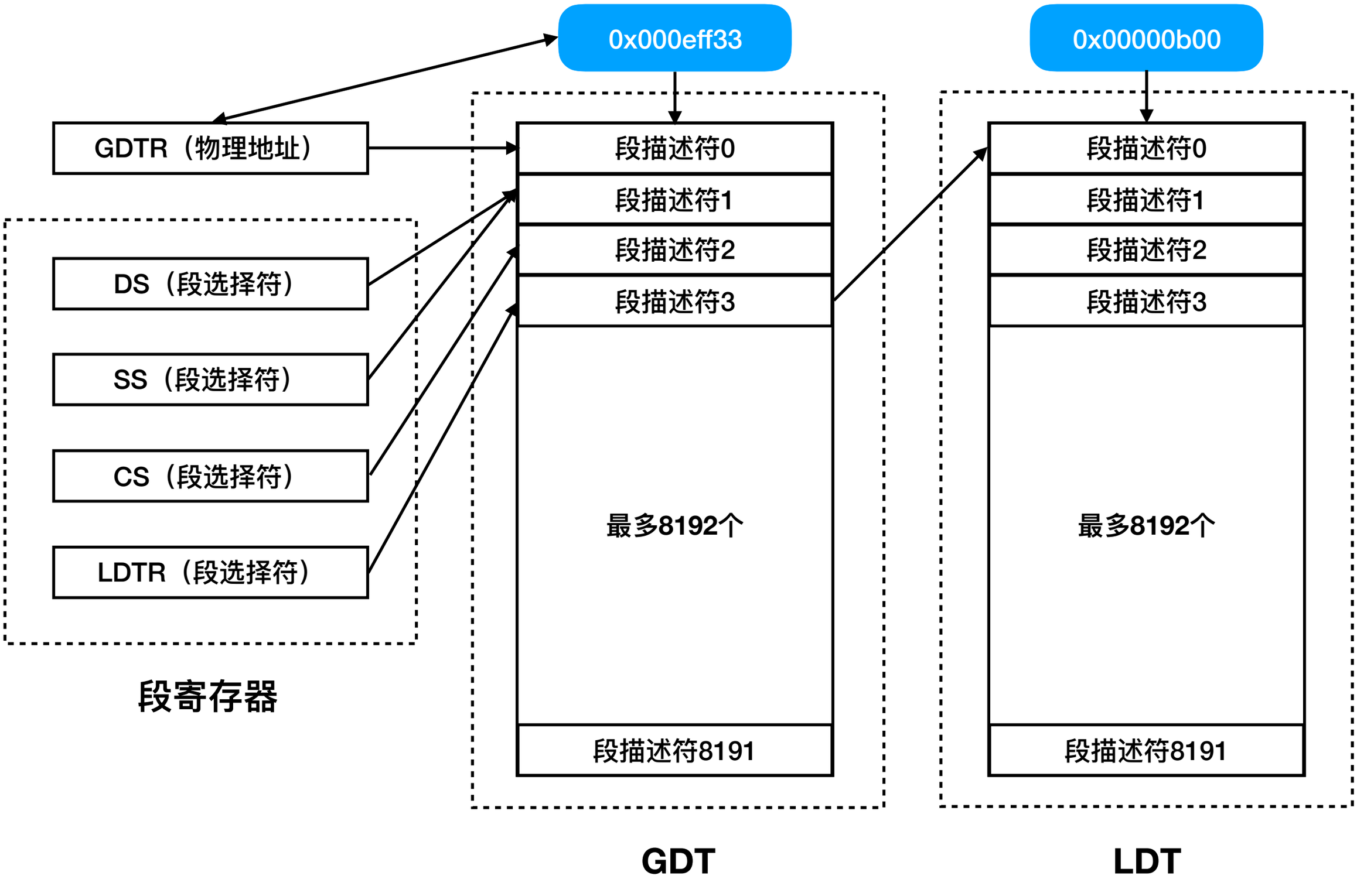
段寄存器组：16位

ES	CS	SS	DS
FS	GS	LDTR	TR

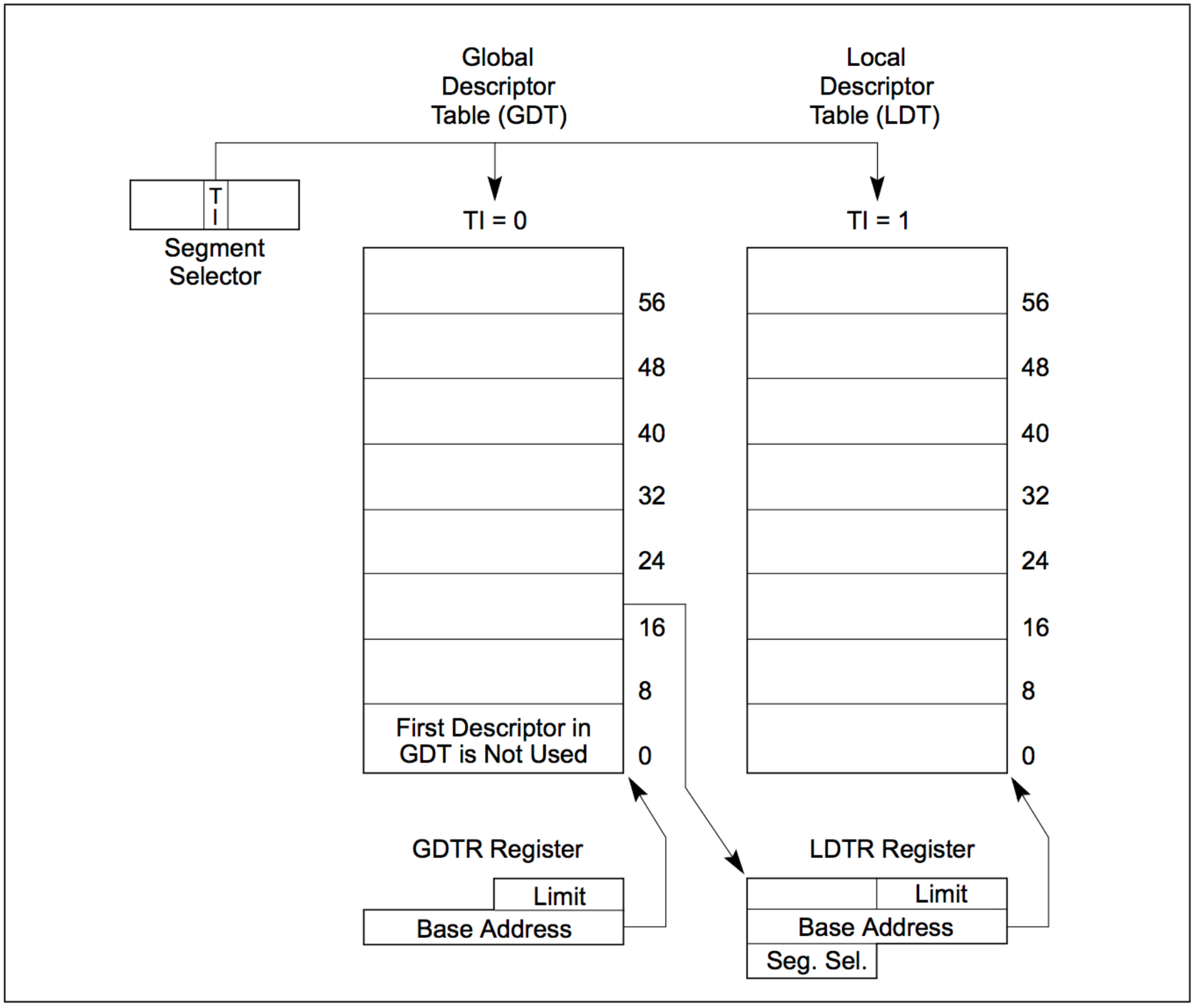
段选择符（Segment Selector）

总数：2^13 = 8192个

分段机制：保护模式分段-GDT和LDT



分段机制：保护模式分段-GDT和LDT



分段机制：保护模式分段-bochs演示

```
<bochs:7> sreg
es:0x0018, dh=0x00cf9300, dl=0x0000ffff, valid=1
        Data segment, base=0x00000000, limit=0xffffffff, Read/Write, Accessed
cs:0x0010, dh=0x00cf9b00, dl=0x0000ffff, valid=1
        Code segment, base=0x00000000, limit=0xffffffff, Execute/Read, Non-Cor
ss:0x0018, dh=0x00cf9300, dl=0x0000ffff, valid=31
        Data segment, base=0x00000000, limit=0xffffffff, Read/Write, Accessed
ds:0x0018, dh=0x00cf9300, dl=0x0000ffff, valid=31
        Data segment, base=0x00000000, limit=0xffffffff, Read/Write, Accessed
fs:0x0018, dh=0x00cf9300, dl=0x0000ffff, valid=1
        Data segment, base=0x00000000, limit=0xffffffff, Read/Write, Accessed
gs:0x0018, dh=0x00cf9300, dl=0x0000ffff, valid=1
        Data segment, base=0x00000000, limit=0xffffffff, Read/Write, Accessed
ldtr:0x0068, dh=0xc000822b, dl=0x67c40007, valid=1
tr:0x0060, dh=0xc0008b2d, dl=0xa1e400eb, valid=1
gdtr:base=0x00000000c02b37c0, limit=0x205f
idtr:base=0x00000000c0305000, limit=0x7ff
```

CS：0010，index=2，TI=0，RPL=0

DS：0018，index=3，TI=0，RPL=0

分段机制：保护模式分段-bochs演示

```
<bochs:12> x /100x 0xc02b37c0
[bochs]:
0x00000000c02b37c0 <bogus+       0>:    0x00000000      0x00000000      0x00000000      0x00000000
0x00000000c02b37d0 <bogus+      16>:    0x0000ffff      0x00cf9b00      0x0000ffff      0x00cf9300
0x00000000c02b37e0 <bogus+      32>:    0x0000ffff      0x00cffb00      0x0000ffff      0x00cff300
0x00000000c02b37f0 <bogus+      48>:    0x00000000      0x00000000      0x00000000      0x00000000
0x00000000c02b3800 <bogus+      64>:    0x00000000      0x00409200      0x00000000      0x00409a00
0x00000000c02b3810 <bogus+      80>:    0x00000000      0x00009a00      0x00000000      0x00409200
0x00000000c02b3820 <bogus+      96>:    0xa1e400eb      0xc0008b2d      0x67c40007      0xc000822b
```

CS指向的段描述符

高32：0x00cf9b00

低32：0x0000ffff

BASE：0x00000000
LIMIT：0xfffff
P：1（在内存中）
G：1（limit*4K）
DPL：00
D/B：1（32位指令）
S：1（普通段）
TYPE：1011（代码段）

DS指向的段描述符

高32：0x00cf9300

低32：0x0000ffff

BASE：0x00000000
LIMIT：0xfffff
P：1（在内存中）
G：1（limit*4K）
DPL：00
D/B：1（32位指令）
S：1（普通段）
TYPE：0011（数据段）

分段机制：保护模式分段-bochs演示

```
<bochs:35> sreg
es:0x002b, dh=0x00cff300, dl=0x0000ffff, valid=31
        Data segment, base=0x00000000, limit=0xffffffff, Read/Write, Accessed
cs:0x0023, dh=0x00cffb00, dl=0x0000ffff, valid=1
        Code segment, base=0x00000000, limit=0xffffffff, Execute/Read, Non-Con
ss:0x002b, dh=0x00cff300, dl=0x0000ffff, valid=31
        Data segment, base=0x00000000, limit=0xffffffff, Read/Write, Accessed
ds:0x002b, dh=0x00cff300, dl=0x0000ffff, valid=31
        Data segment, base=0x00000000, limit=0xffffffff, Read/Write, Accessed
fs:0x0000, dh=0x00001000, dl=0x00000000, valid=0
gs:0x0000, dh=0x00001000, dl=0x00000000, valid=0
ldtr:0x0068, dh=0xc000822b, dl=0x67c40007, valid=1
tr:0x0180, dh=0xdd008b53, dl=0x21e400eb, valid=1
gdtr:base=0x00000000c02b37c0, limit=0x205f
idtr:base=0x00000000c0305000, limit=0x7ff
```

CS：0023，index=4，TI=0，RPL=3

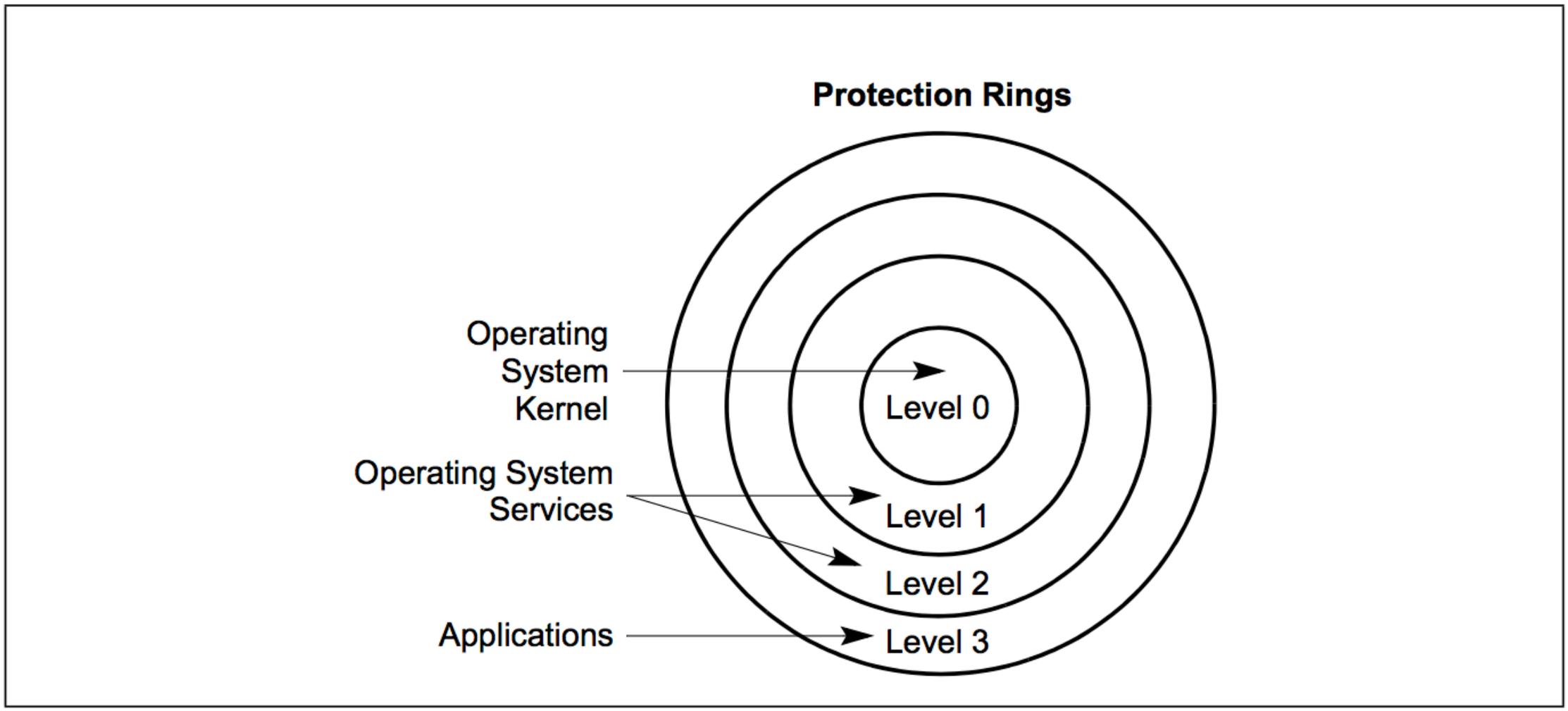
DS：002b，index=5，TI=0，RPL=3

分段机制：保护模式分段-权限类型

每条指令执行的时候
存在以段位单位的四种权限类型

读写内存	读写IO
指令执行	指令转移

分段机制：保护模式分段-权限级别



CPU位应用执行设计了4层权限级别

分段机制：保护模式分段-权限令牌

CPL

Current Privilege Level （拥有的权限）
CPL = CS.RPL
CS为当前执行代码所在的代码段的段选择符

DPL

Descriptor Privilege Level （访问数据所需要的权限）
DPL = SegmentDescriptor.DPL
段描述符为所访问数据段的段描述符

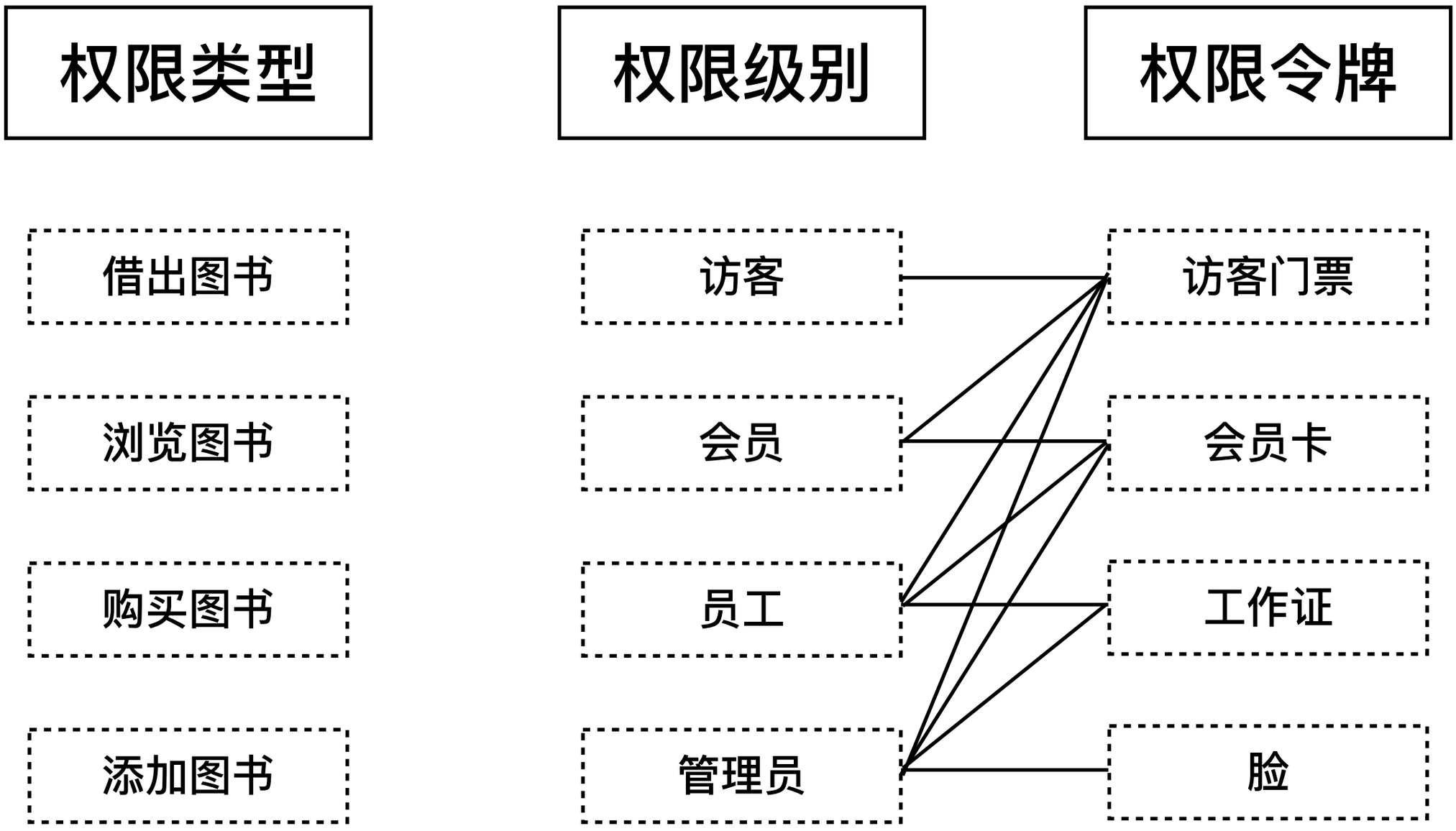
IOPL

IO Privilege Level （访问IO所需要的权限）
IOPL = EFLAGS.IOPL

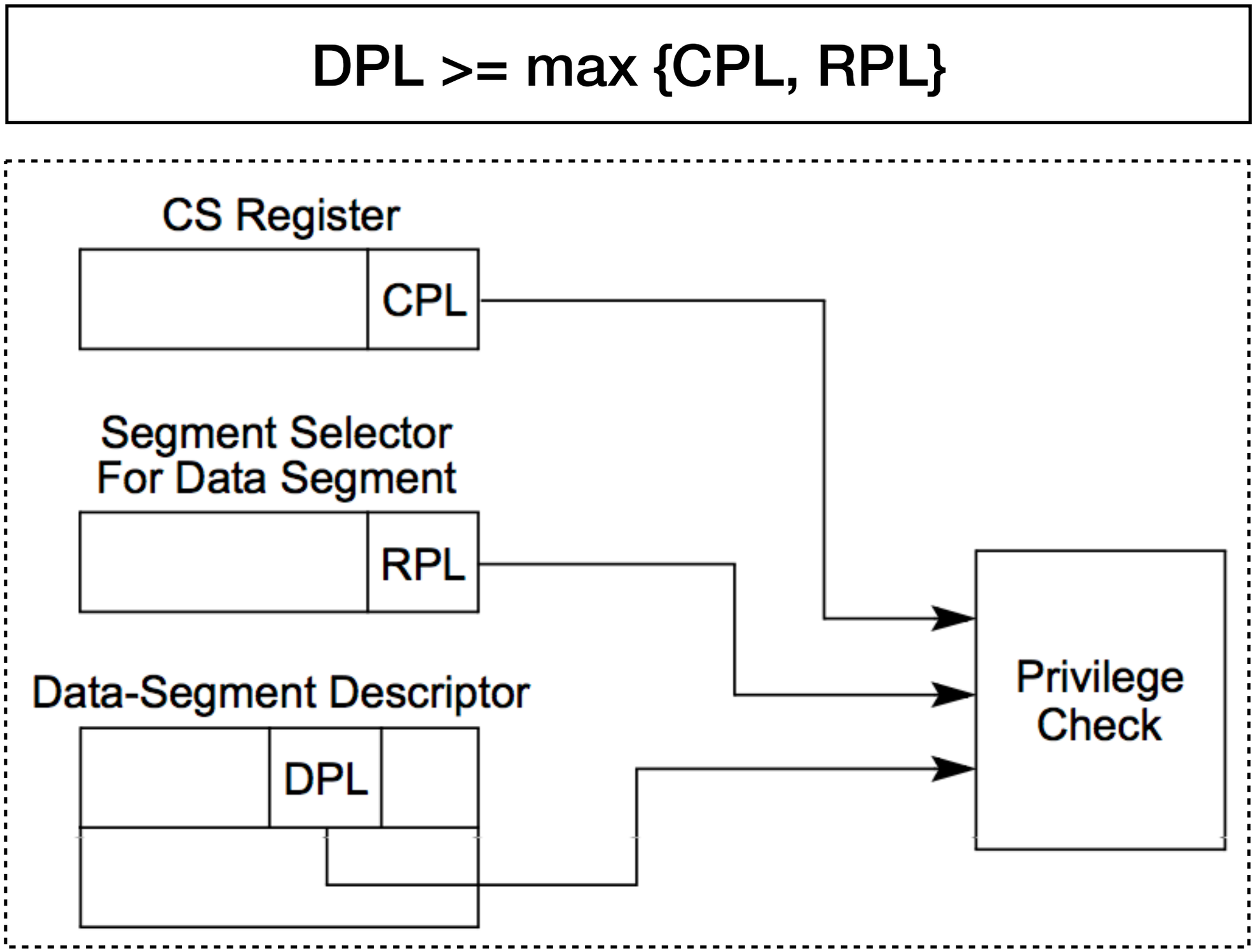
RPL

Request Privilege Level （请求使用的权限）
RPL = SegmentSelector.RPL
段选择符为所访问数据段的段选择符

分段机制：保护模式分段-权限类比

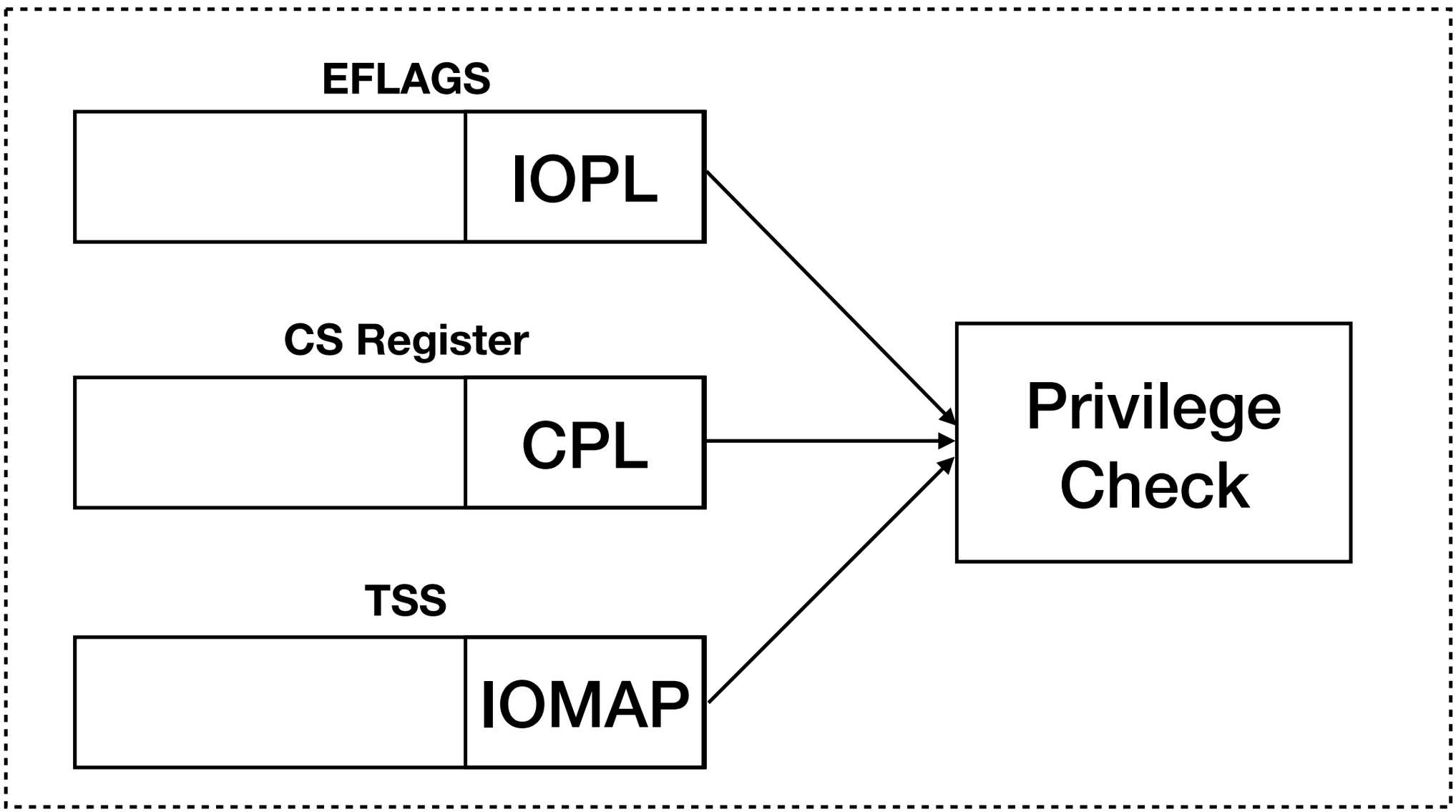


分段机制：保护模式分段-读写内存权限



分段机制：保护模式分段-读写IO权限

读写IO：IOPL >= CPL



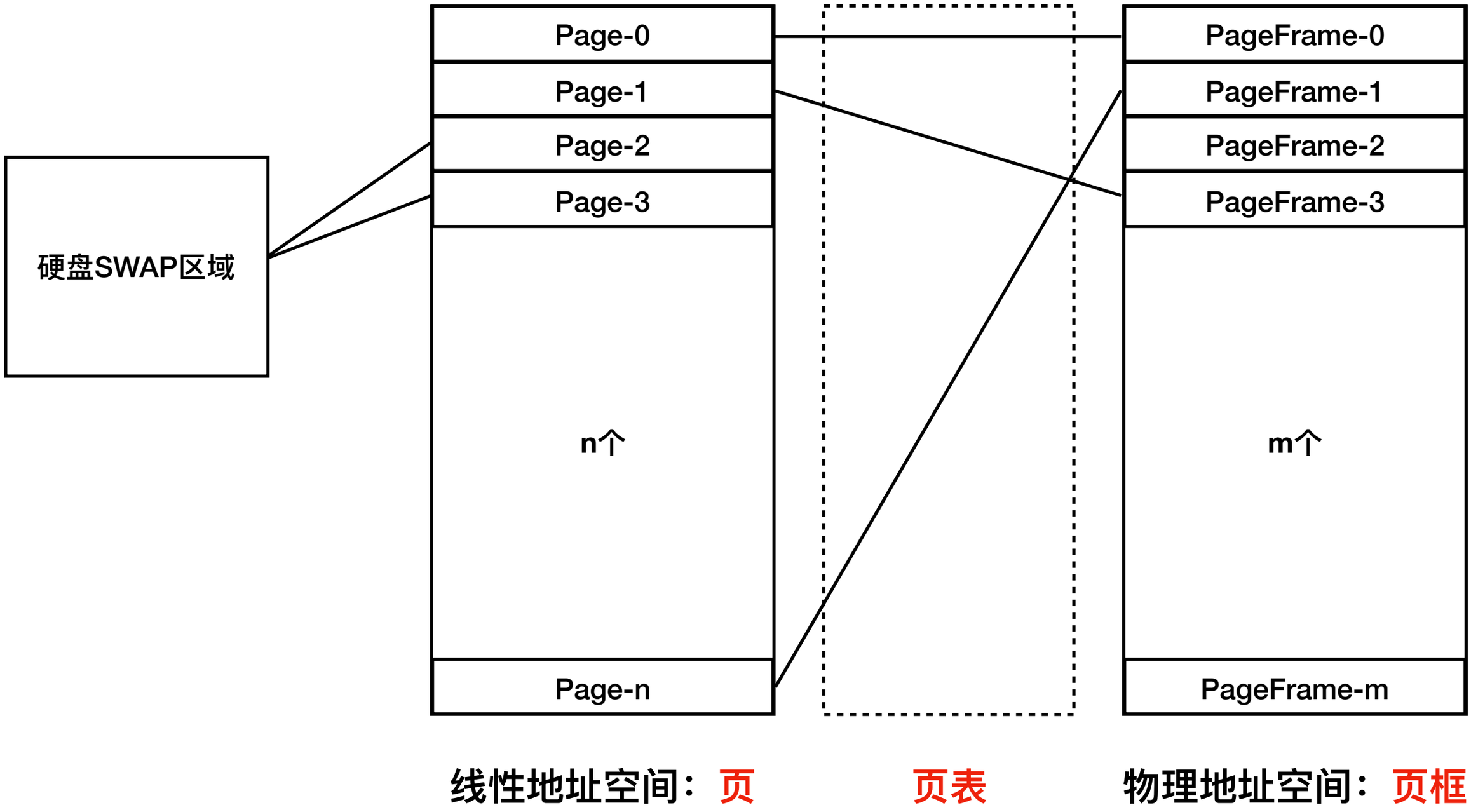
分段机制：保护模式分段-指令执行权限

指令执行：是否特权指令

LGDT：写GDTR寄存器
LLDT：写LDTR寄存器
LTR：写TR寄存器
LIDT：写IDTR寄存器
MOV（到CR寄存器）：读写控制寄存器
MOV（到Debug寄存器）：读写调试寄存器
HLT：挂起CPU

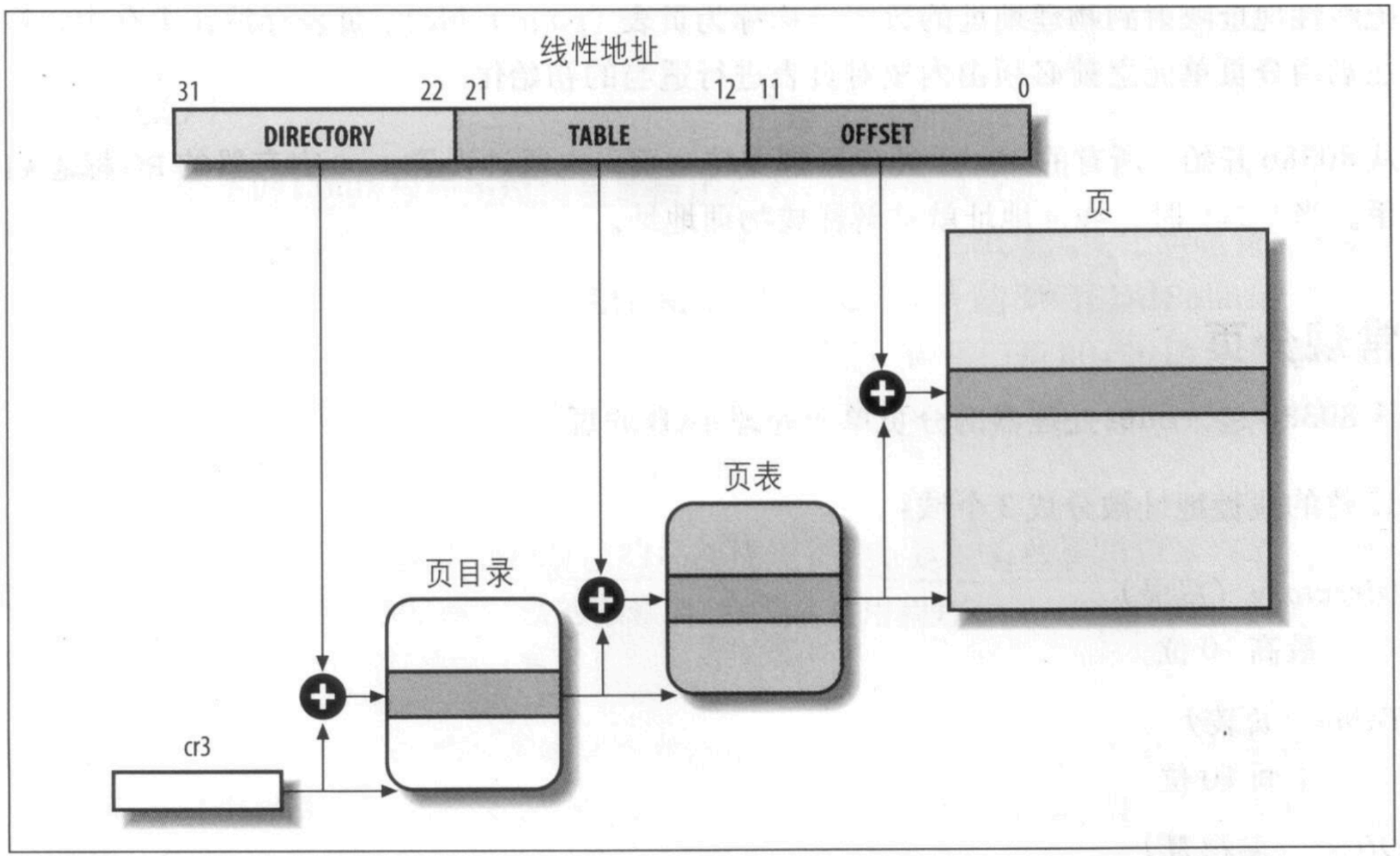
特权指令只有在CPL=0时才可以执行

分页机制：基本原理



线性地址空间：页　　页表　　物理地址空间：页框

分页机制：页表结构

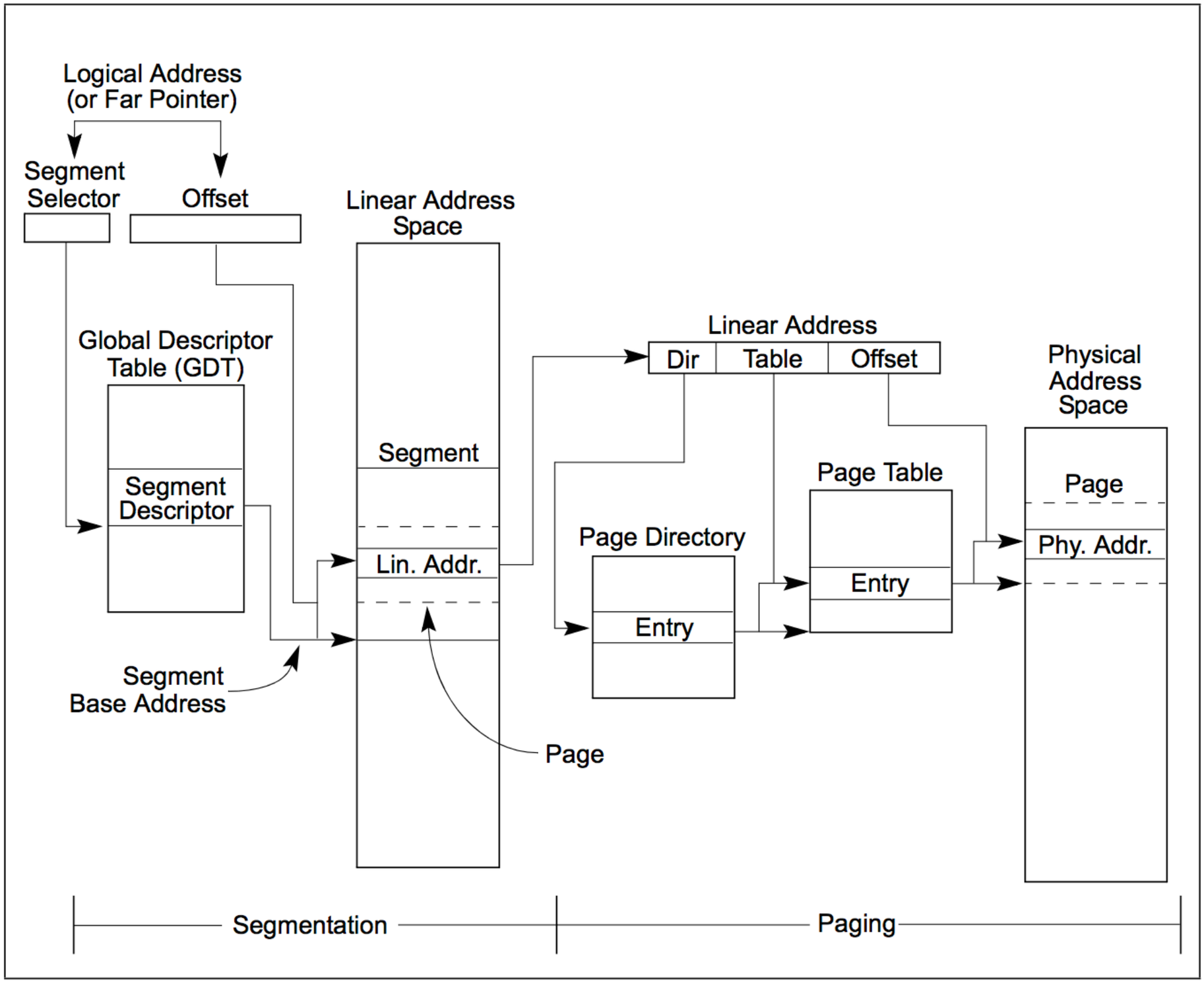


不同任务对应自己不同的页表，实现数据隔离

分页机制：bochs演示页表切换

```
247078799489: address space switched. CR3: 0x000000101000
247348297782: address space switched. CR3: 0x00001fe6b000
247348303792: address space switched. CR3: 0x000000101000
247477299847: address space switched. CR3: 0x00001d677000
247477309248: address space switched. CR3: 0x000000101000
247643802493: address space switched. CR3: 0x00001fe6b000
247643808505: address space switched. CR3: 0x000000101000
247841305780: address space switched. CR3: 0x00001f877000
247841308725: address space switched. CR3: 0x000000101000
247932807118: address space switched. CR3: 0x00001fe6b000
247932813128: address space switched. CR3: 0x000000101000
247987963726: address space switched. CR3: 0x00001f97b000
247988025403: address space switched. CR3: 0x00001d539000
247988046076: address space switched. CR3: 0x000000101000
247988070467: address space switched. CR3: 0x00001d539000
248209811523: address space switched. CR3: 0x00001fe6b000
248209817553: address space switched. CR3: 0x00001d539000
248244812053: address space switched. CR3: 0x000000101000
248244820024: address space switched. CR3: 0x00001d539000
^CNext at t=248363018571
```

保护模式下的完整寻址流程



寻址机制中CPU和内核的分工

CPU提供的功能

5种模式的进入规则与实现

段描述符的格式定义

段选择符的格式定义

GDT表的定义和规则

CS、SS寄存器的特殊校验规则

基于CPL/DPL/APL/IOPL的权限验证

分页的定义与实现

内核要干的事情

初始化EFLAGS寄存器

初始化GDT表和GDTR

初始化LDT表和LDTR

初始化页表和CR寄存器组

初始化CS寄存器

CPU运行模式切换控制

切换任务时控制页表的切换

CPU寻址的评价：以IntelX86/64为例

	Intel的想法	别人的想法
命名	IA-32/IA-32e	所有人 : X86/X86-64
汇编指令	mov rbx, rax	ATT : movq %rax, %rbx
内存模型	segmented model	LINUX : flat model
任务切换	硬切换（多个TSS）	LINUX : 软切换（单一TSS）
权限级别	4级	LINUX : 2级
任务数据	用LDT	LINUX：不用LDT

Intel：全世界都在和我过不去！

Intel生产了一件华丽的西装，大家都用来当抹布

CPU寻址的评价：以IntelX86/64为例

还不是因为你长的不好看

为什么会被狮子男拒绝呢?
还不是因为你长的不好看!

为什么会被双鱼男拒绝呢?
还不是因为你长的不好看!

为什么会被天蝎男拒绝呢?
还不是因为你长的不好看!

为什么这么久都没有对象呢?
还不是因为你长的不好看!

你以为他很在意姑娘是否有内涵么?
不你错了他只喜欢姑娘很好看!

你以为你长的挺好看就没事了么?
不你错了他还想找个更好看的!

还不是为了向上兼容

为什么EFLAGS的设计这么混乱呢?
还不是为了向上兼容!

为什么段寄存器的功能这么诡异呢?
还不是为了向上兼容!

为什么权限检查这么复杂呢?
还不是为了向上兼容!

为什么段描述符这么反人类呢?
还不是为了向上兼容?

你以为工程师很在意你设计合理么?
不你错了他们只在意你向上兼容

你以为你向上兼容就够了么?
不你错了他们还想你兼容别的CPU!

推荐

CPU仿真工具：Bochs 下载地址 http://http://bochs.sourceforge.net/

《深入学习linux内核》第三版：第二章、第四章