操作系统
Operating Systems

Lec7 实验二 物理内存管理

清华大学计算机系

概 述

- x86 特权级（ privilege levels ）
- x86 内存管理单元（ Memory Management Unit ， MMU)

- 了解不同特权级的差别
- 了解当前 CPU 处在哪个特权级
- 了解特权级切换

X86 特权级

x86 特权级 – 简介

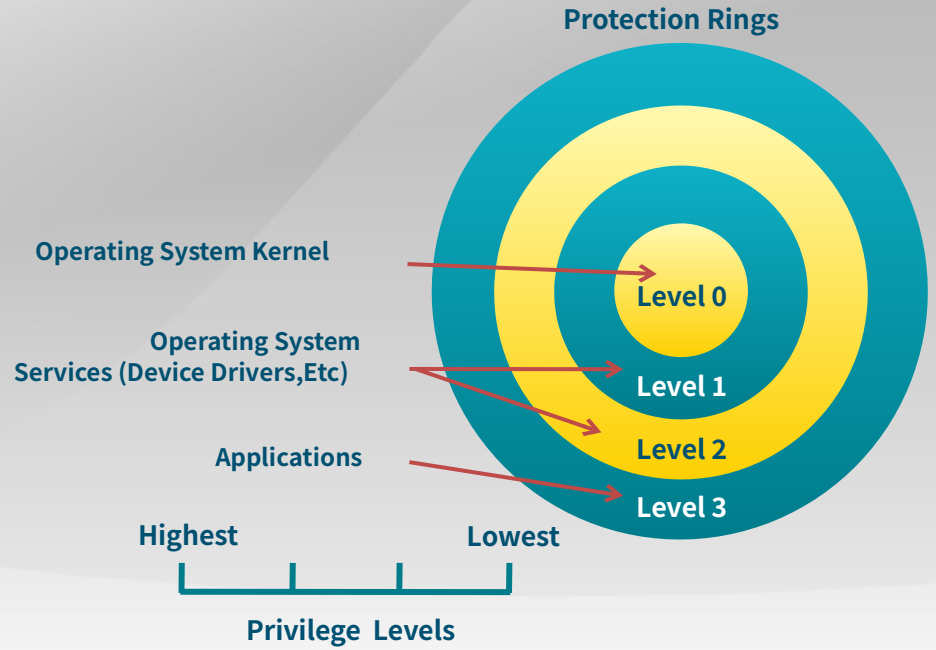


图 Protection Rings

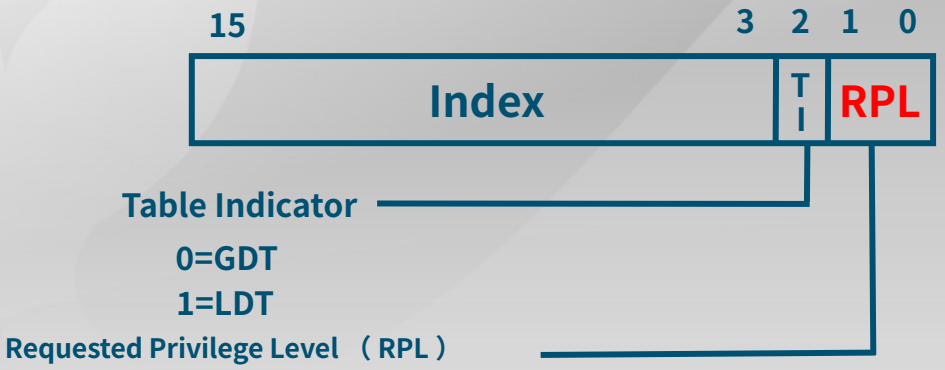
- Linux 和 uCore 只使用 ring 0 and ring 3

x86 特权级 –区别？

- 一些指令（比如特权指令）只能执行在 ring 0 (e.g. lgdt).
- CPU 在如下时刻会检查特权级
  - 访问数据段
  - 访问页
  - 进入中断服务例程（ ISRs ）
  - ……
- 如果检查失败会如何？

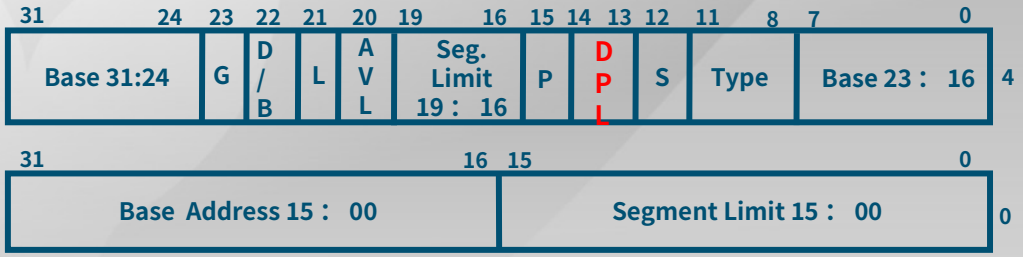
General Protection Fault!

x86 特权级 –段选择子



段选择子　Segment Selector

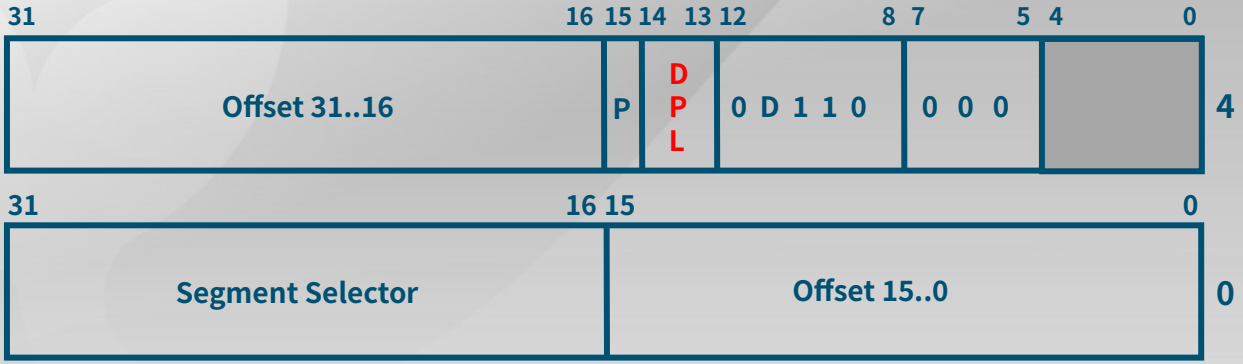
x86 特权级 –段描述符



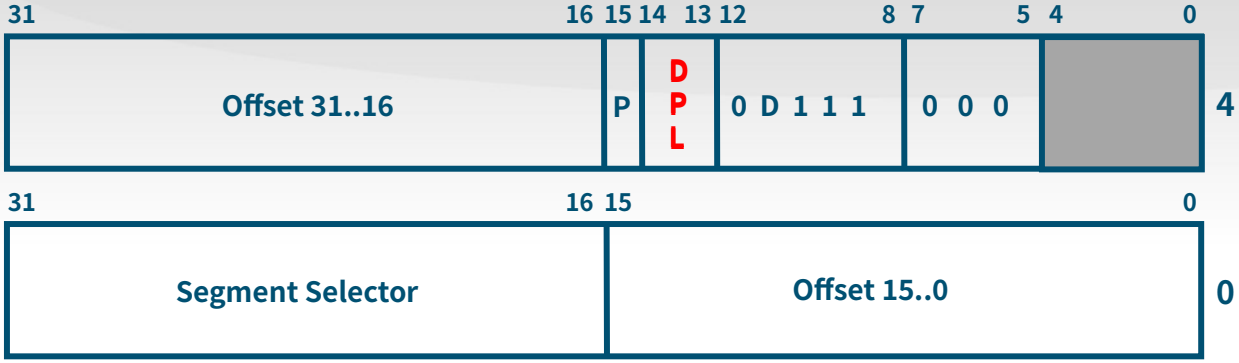
段描述符 Segment Descriptor

x86 特权级 –门描述符

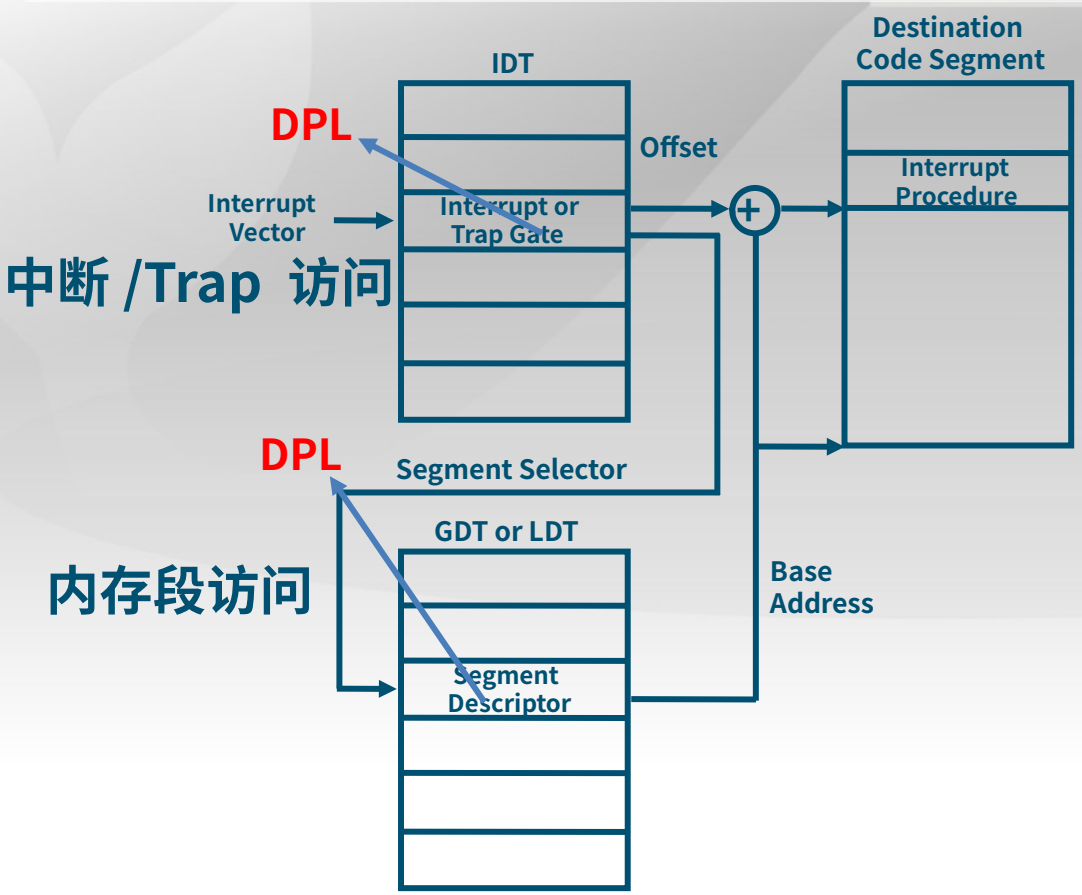
Interrupt Gate



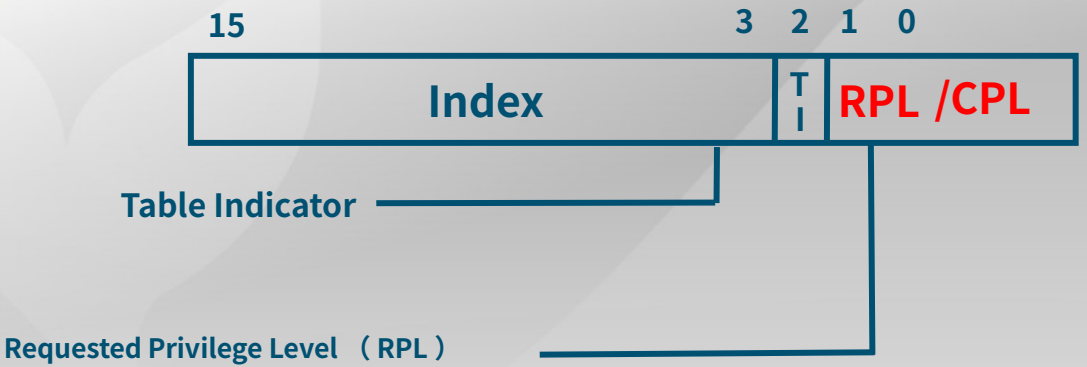
Trap Gate



x86 特权级 –特权转移


IDT
DPL
Interrupt
Vector
Interrupt or
Trap Gate
中断 /Trap 访问
Offset
Destination
Code Segment
Interrupt
Procedure
DPL
Segment Selector
GDT or LDT
内存段访问
Base
Address
Segment
Descriptor

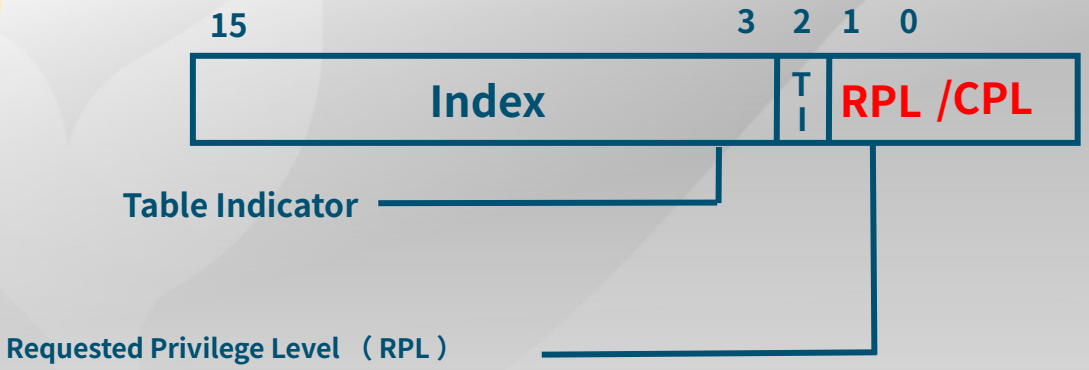
x86 特权级 –当前的特权级？



RPL 段寄存器 DS， ES， FS, GS

段选择子 Segment Selector

x86 特权级 –当前的特权级？

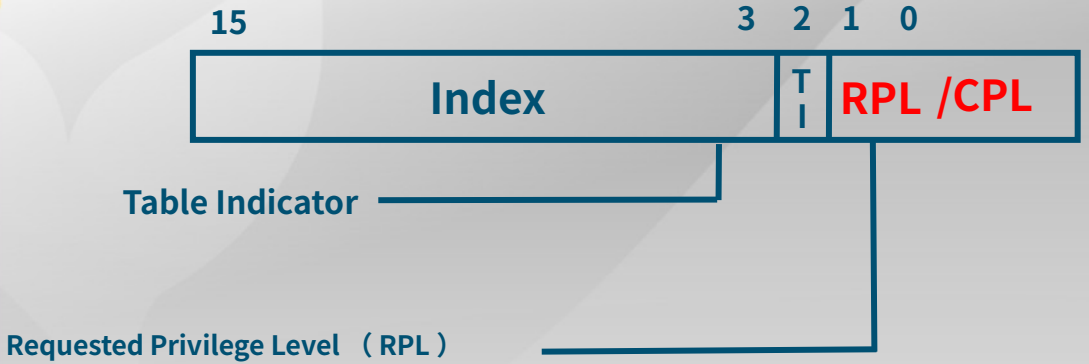


RPL 段寄存器 DS， ES， FS, GS

CPL 段寄存器 CS， SS

段选择子 Segment Selector

x86 特权级 –当前的特权级？

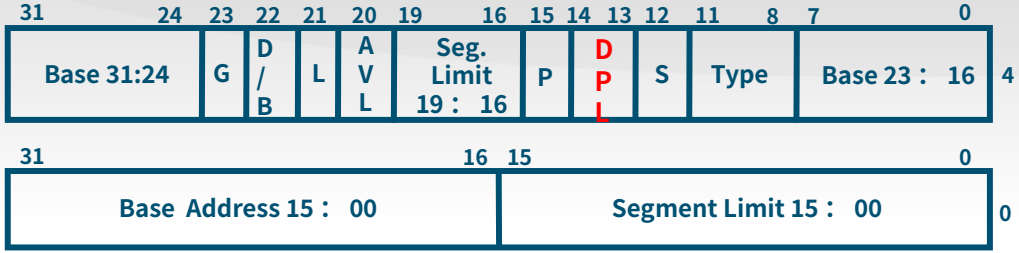


RPL 段寄存器 DS， ES， FS, GS

CPL 段寄存器 CS， SS

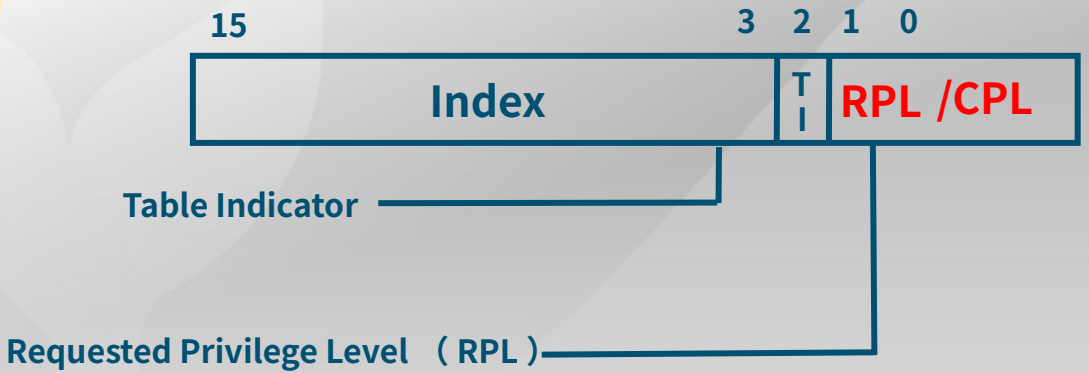
DPL 段描述符，门描述符

段选择子 Segment Selector



段 / 门描述符

x86 特权级 –当前的特权级？



RPL 段寄存器 DS， ES， FS, GS

CPL 段寄存器 CS， SS

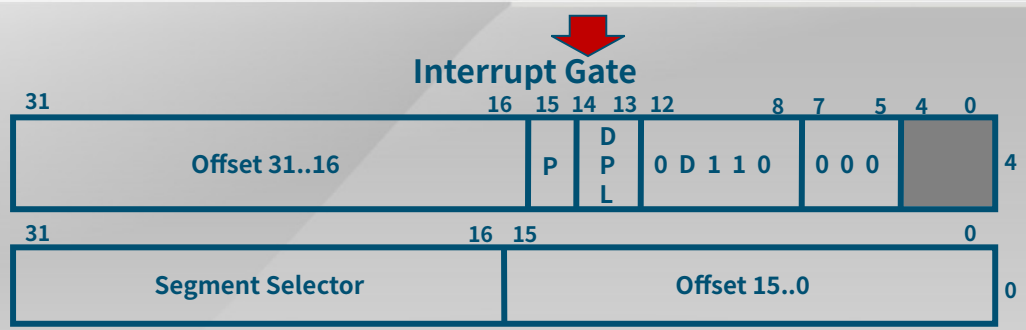
DPL 段描述符，门描述符

段选择子 Segment Selector

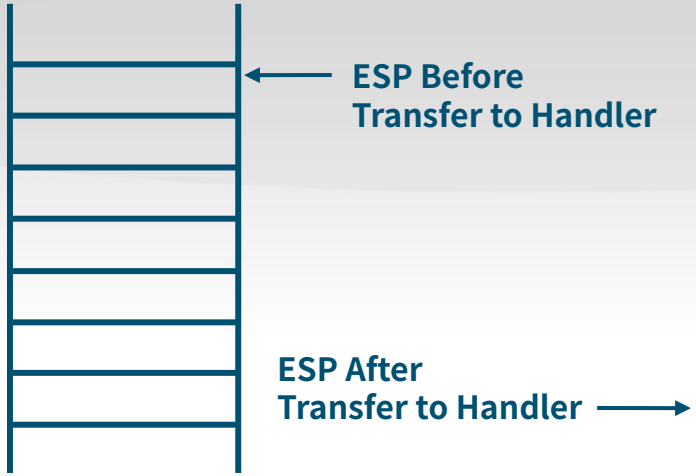
访问门时 CPL <= DPL[门] & CPL >= DPL[段]

访问段时 MAX(CPL, RPL) <= DPL[段]

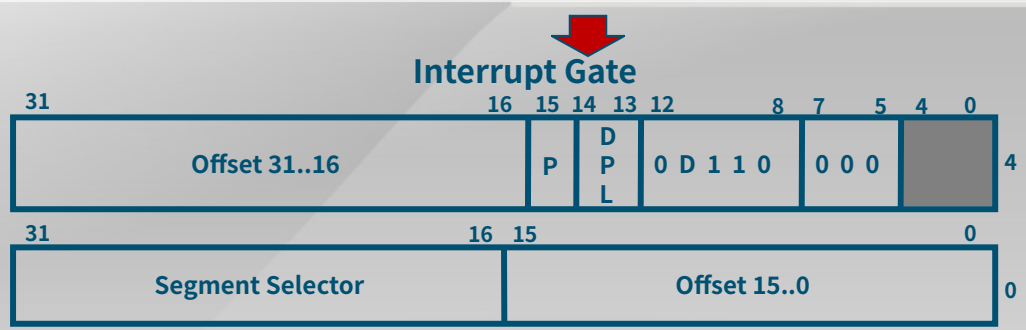
x86 特权级 –通过中断切换特权级



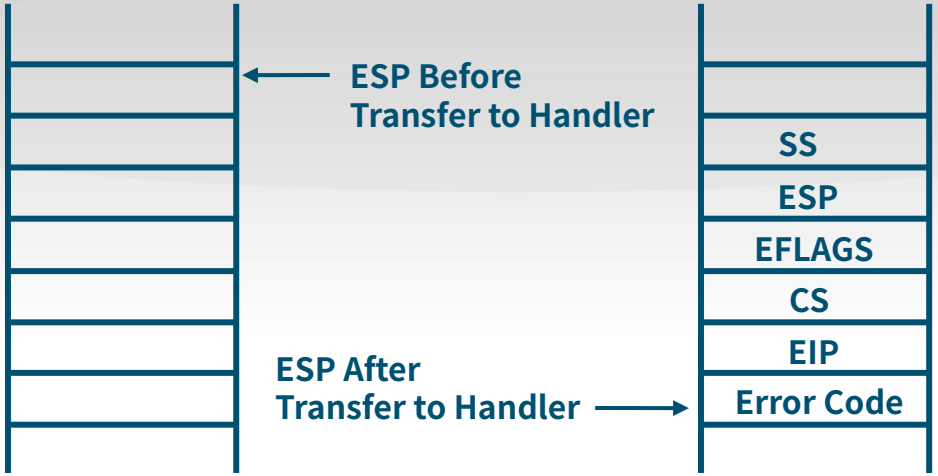
Interrupt Stack Usage with Privilege-Level Change



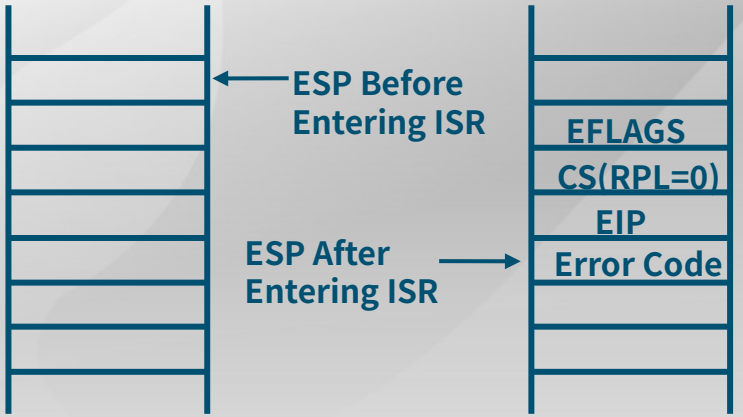
x86 特权级 –通过中断切换特权级



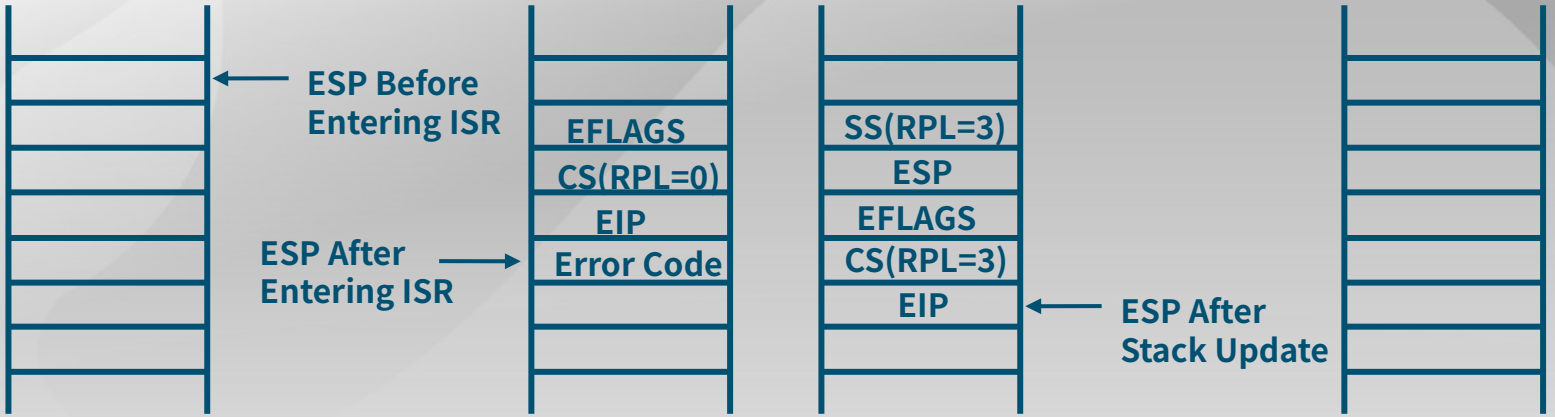
Interrupt Stack Usage with Privilege-Level Change



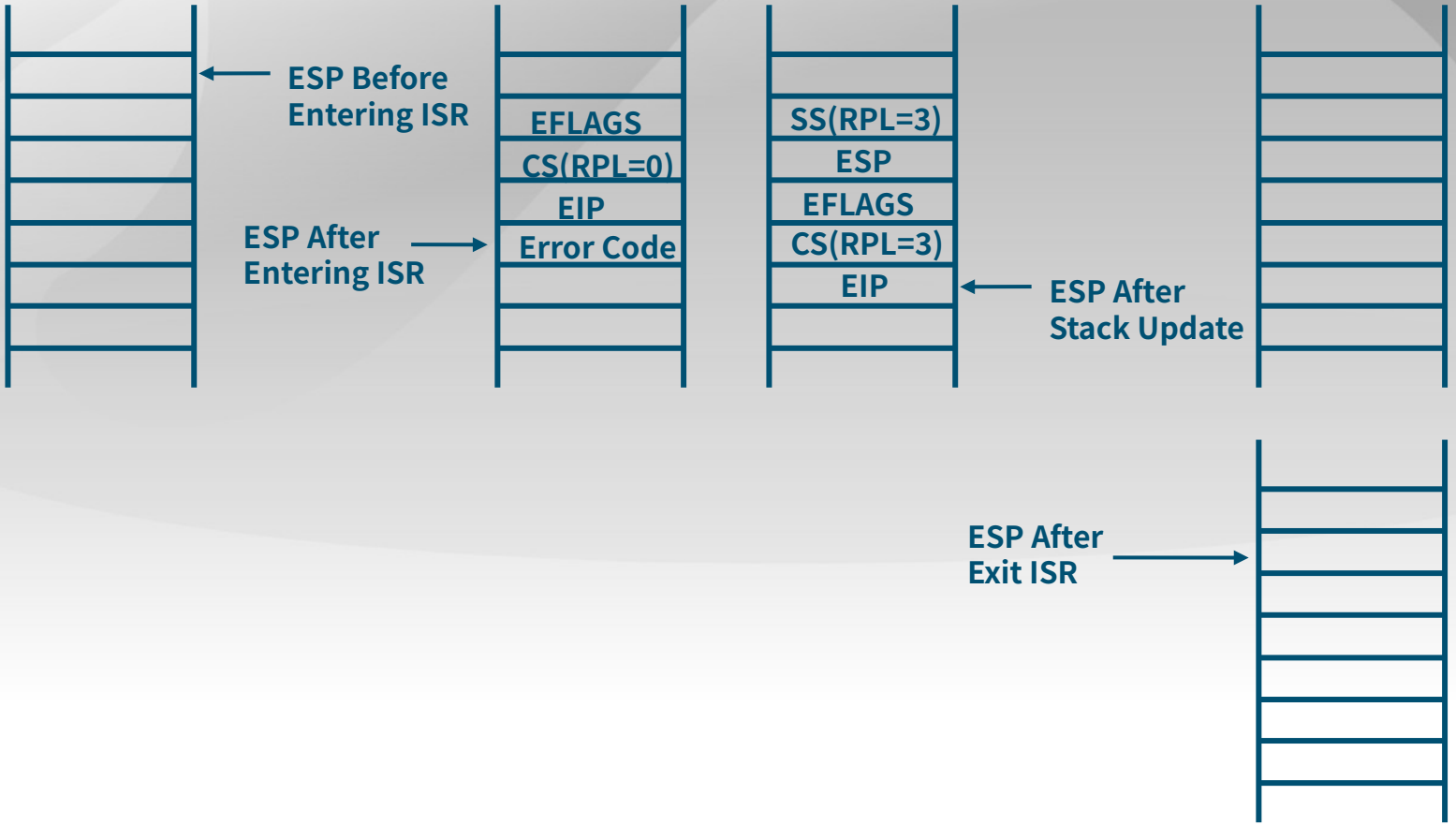
x86 特权级 –切换特权级（0 to 3)



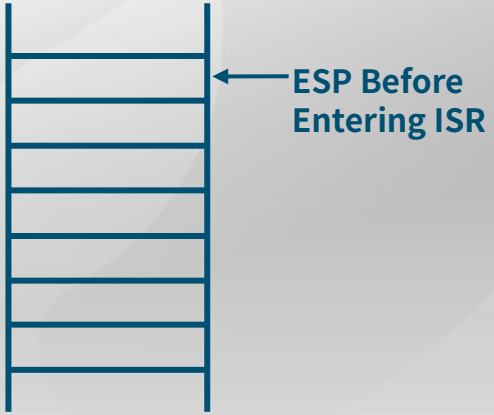
x86 特权级 –切换特权级（0 to 3)



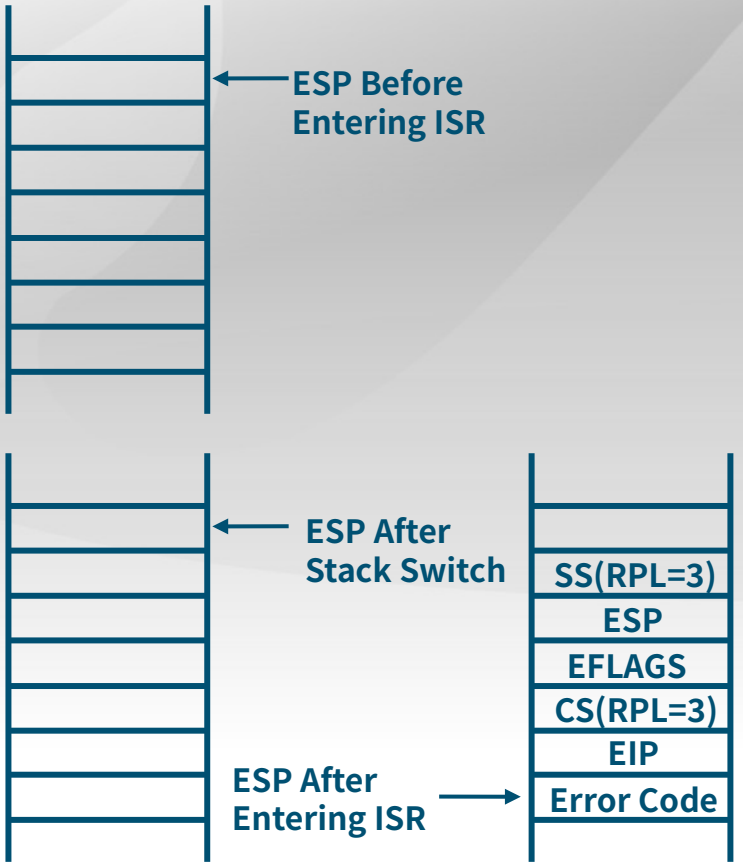
x86 特权级 –切换特权级（0 to 3)



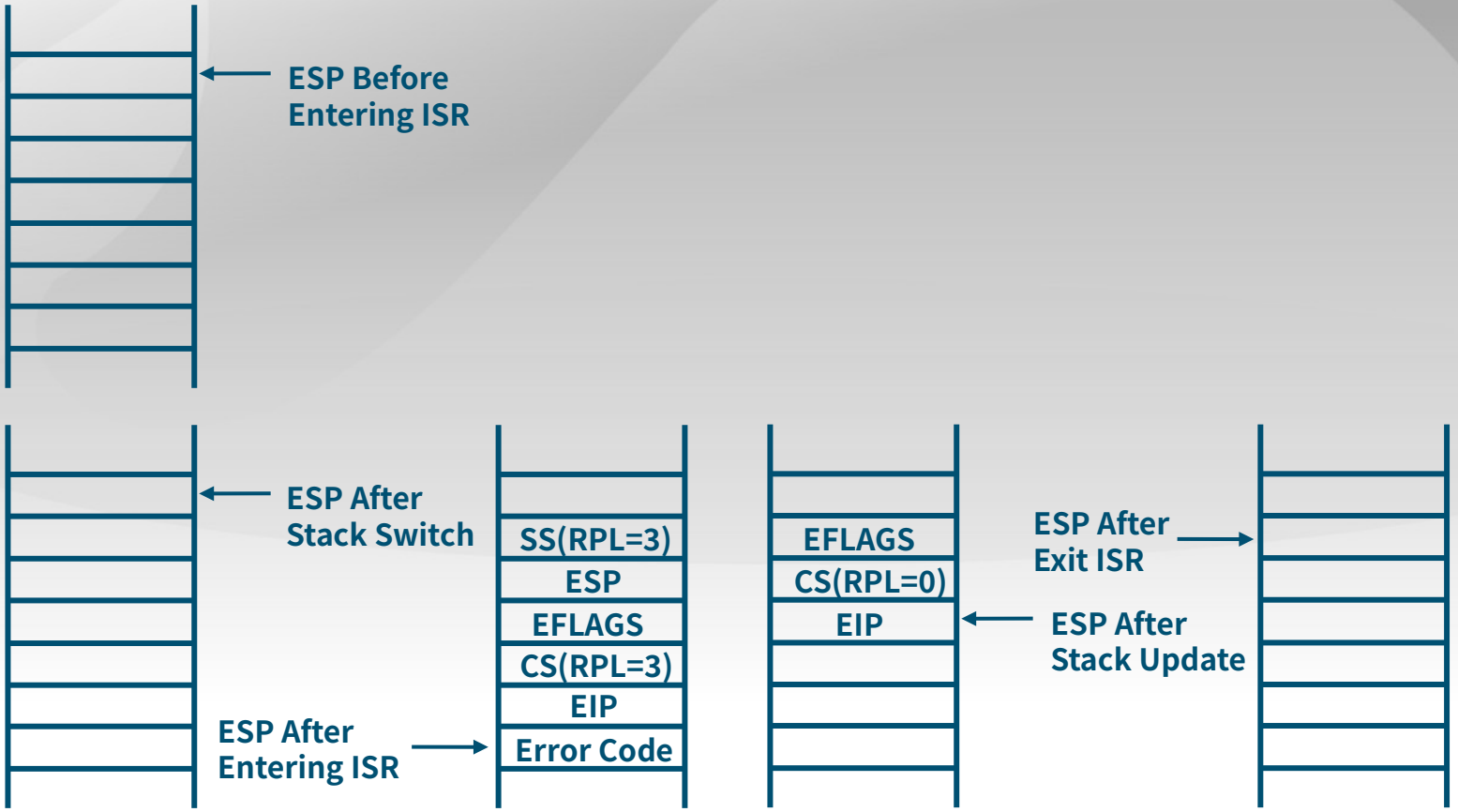
x86 特权级 –切换特权级（3 to 0)



x86 特权级 –切换特权级（3 to 0)



x86 特权级 –切换特权级（3 to 0)



x86 特权级 – TSS 格式

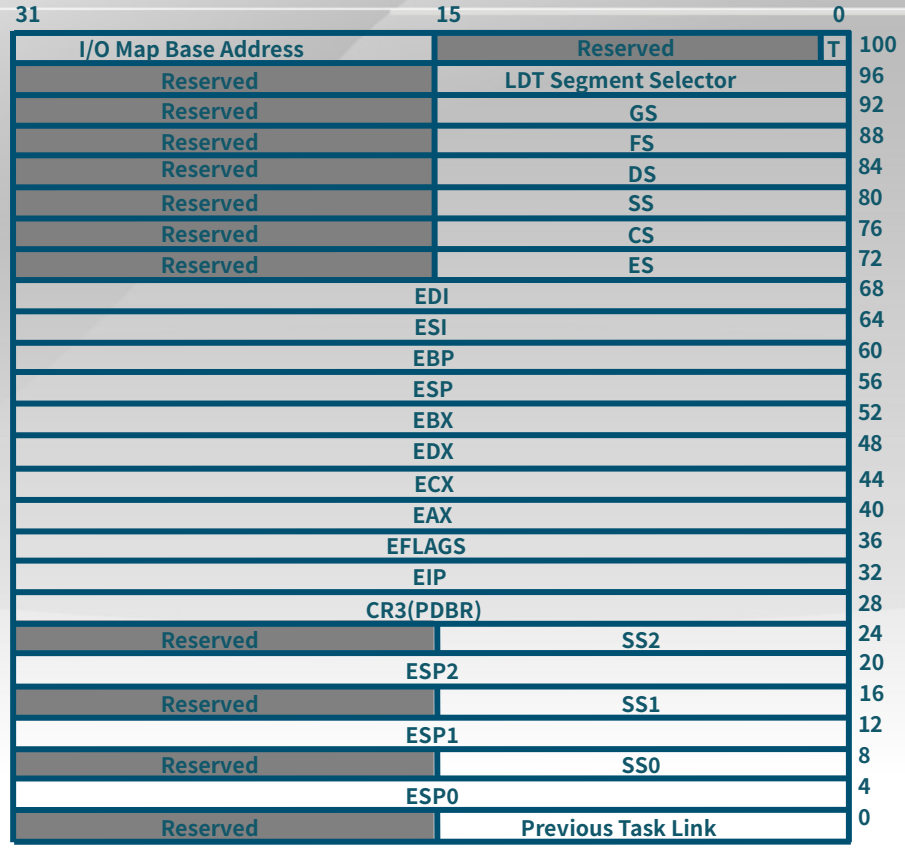


图　32-Bit Task-State Segment(TSS)

x86 特权级 – 栈切换中获取新的栈

- TSS = Task State Segment

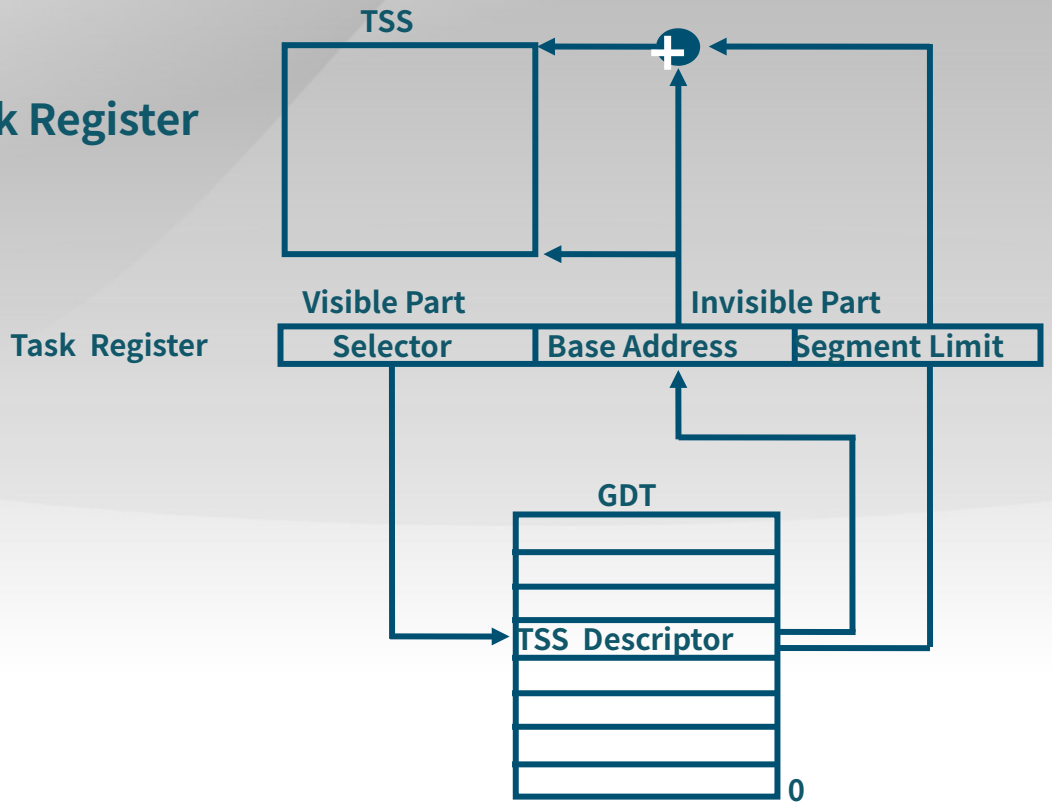
AVL	Available for use by system software
B	Busy flag
BASE	Segment Base Address
DPL	Descriptor privilege level
G	Granularity
LIMIT	Segment Limit
P	Segment Present
TYPE	Segment Type

图 Task State Segmemnt 描述符 TSS Descriptor

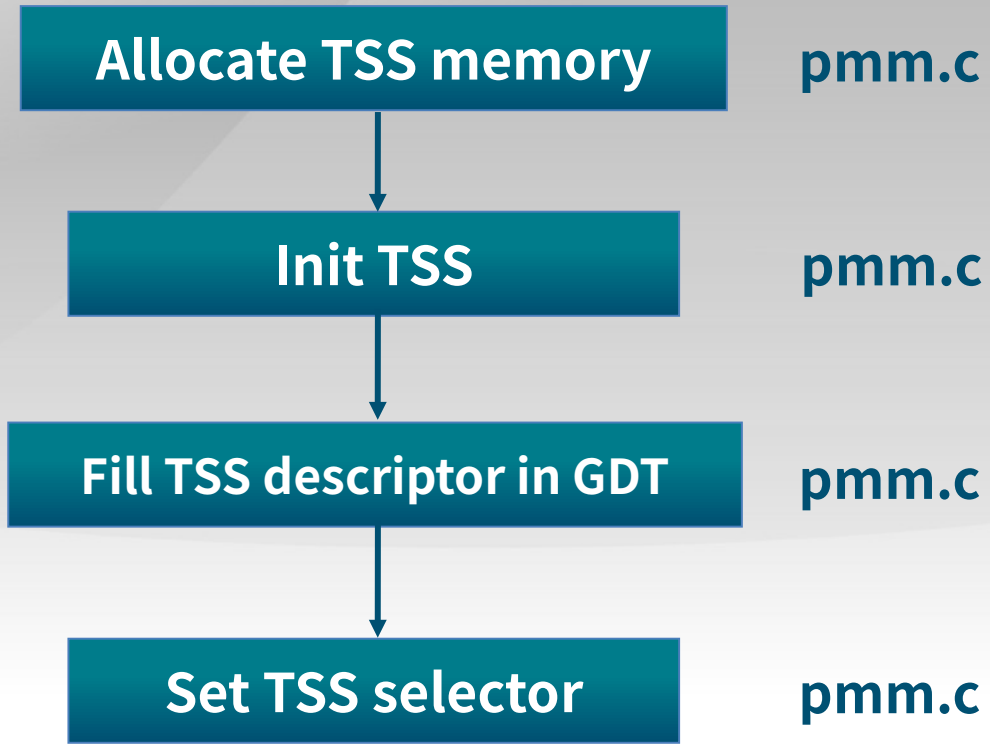
x86 特权级 – 栈切换中获取新的栈

- TSS = Task State Segment

图 Task Register



x86 特权级 –建立 TSS



x86 特权级 – 参考文献

- Chap. 6.3.5, Vol. 1, Intel® and IA-32 Architectures Software Developer’ s Manual
- Chap. 7, Vol. 3, Intel® and IA-32 Architectures Software Developer’ s Manual

- 了解页表格式
- 了解如何建立段表 + 页表
- 了解如何操作页表项

X86 内存管理单元 MMU

x86 MMU – 段机制概述

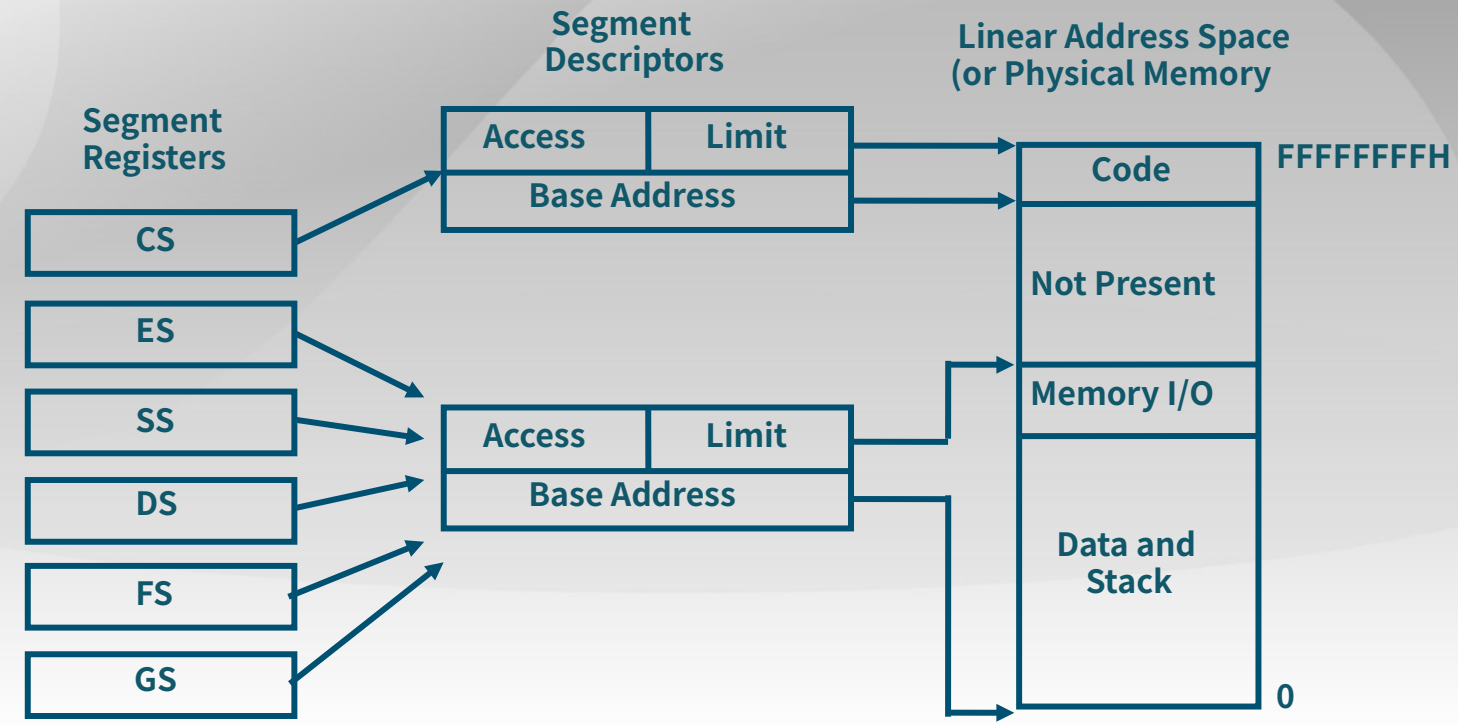


图　Protected Flat Model

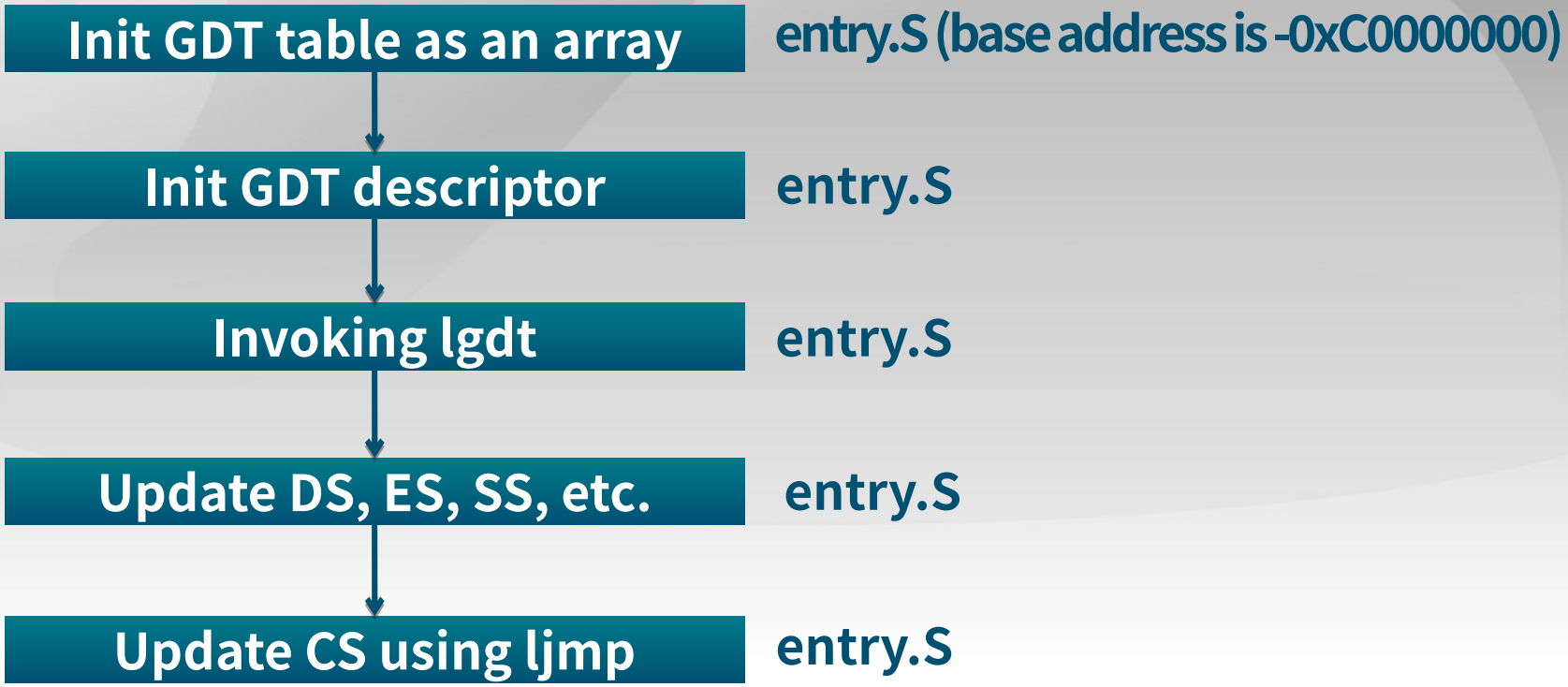
x86 MMU – 段选择子（segment selector）中的隐藏部分

Visible Part	Hidden Part	
Segment Selector	Base Address,Limit,Access Information	CS
		SS
		DS
		ES
		FS
		GS

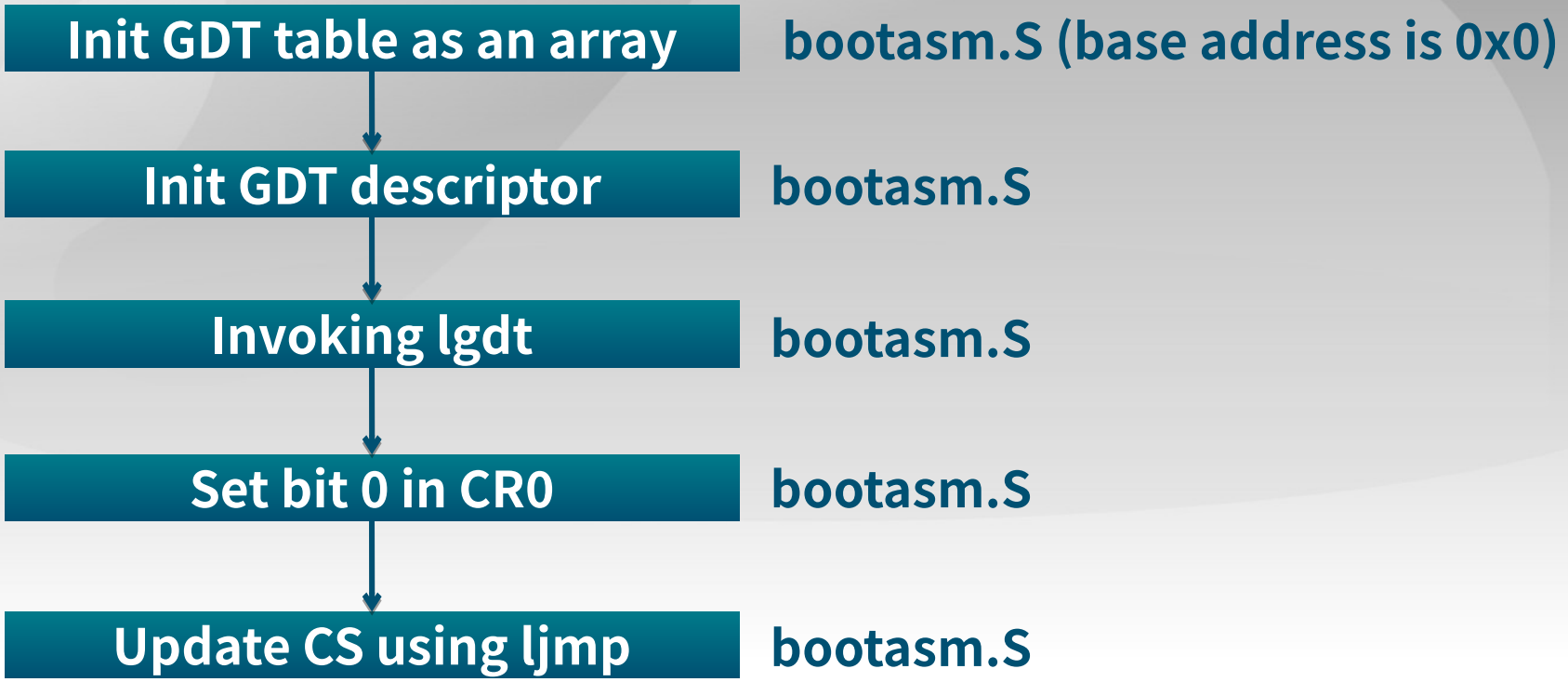
图 Segment Registers

- 基址（Base address）一直被存放在隐藏部分。直到选择子发生变化，才会更新基址内容（即新的段表项中的基址值）。

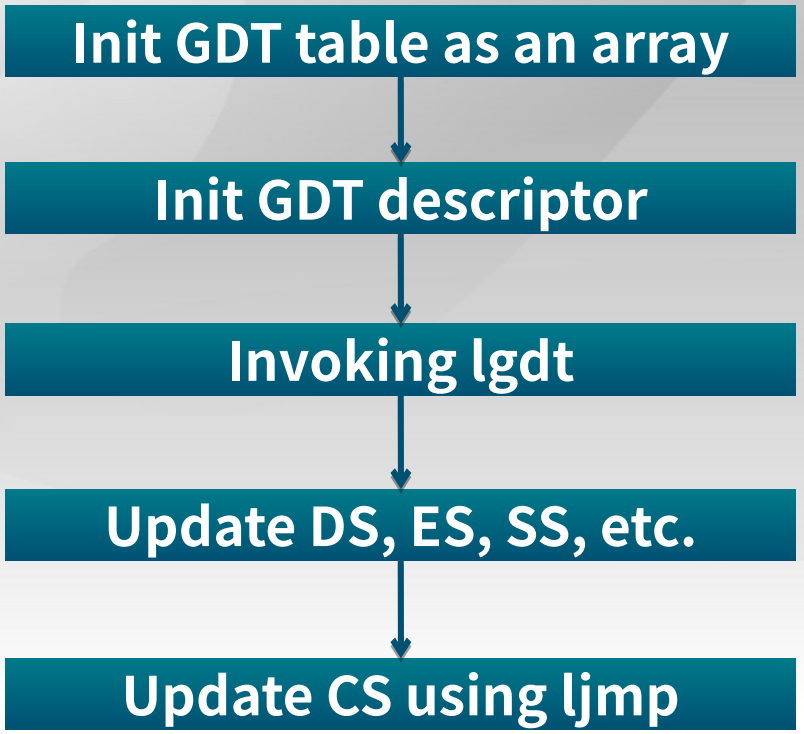
x86 hardware MMU – 建立 GDT tables (kernel init)



x86 MMU – 建立 GDT tables (bootloader)



x86 MMU – 建立 GDT tables (使能页机制 enable paging)



pmm.c (base address is 0x0)

pmm.c

pmm.c

pmm.c

pmm.c

Why bothering?!

x86 MMU – 页机制概述

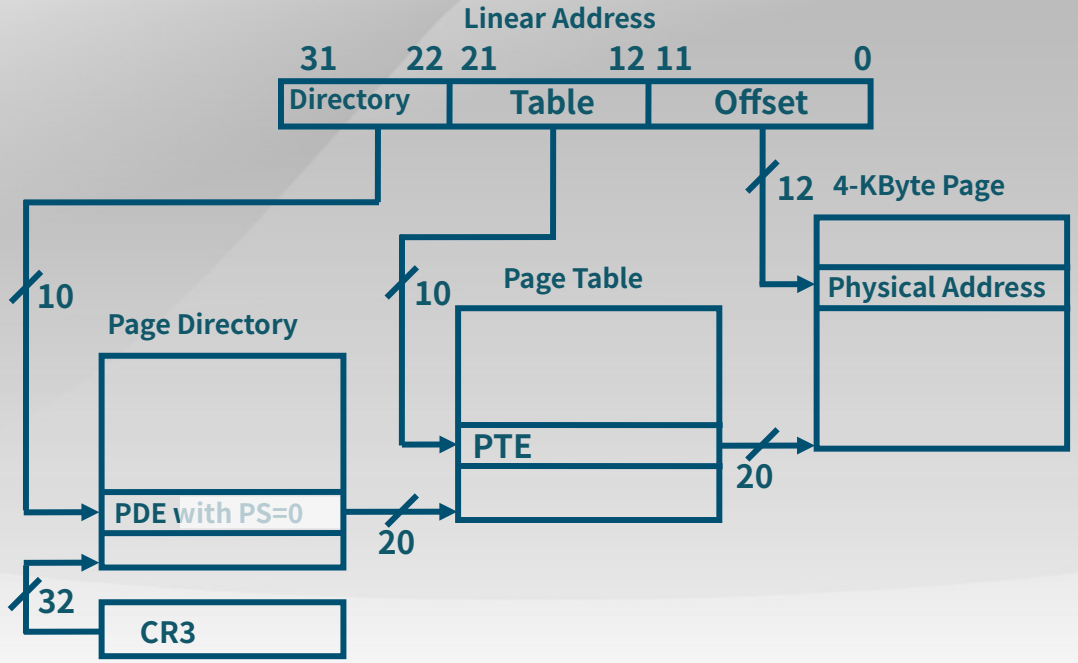
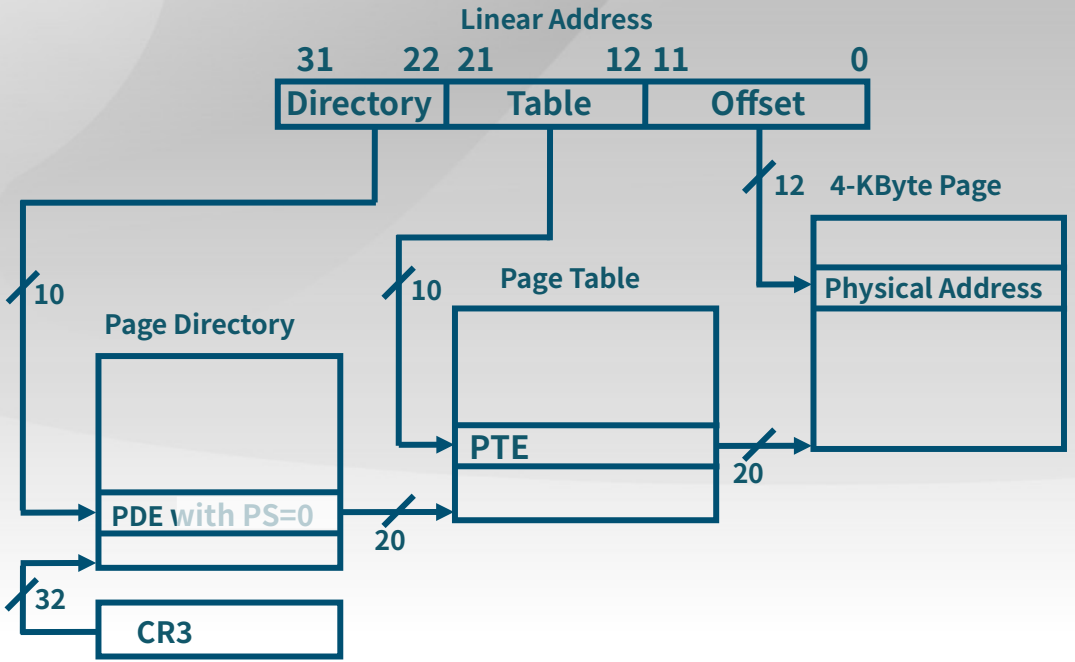


图　基于页机制的地址转换

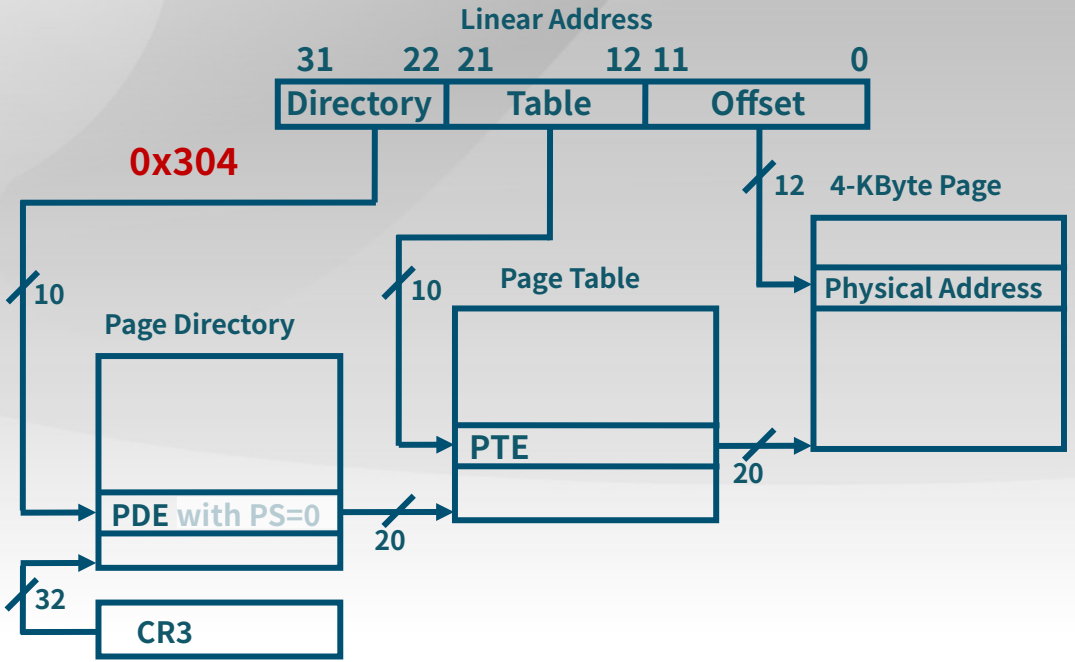
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



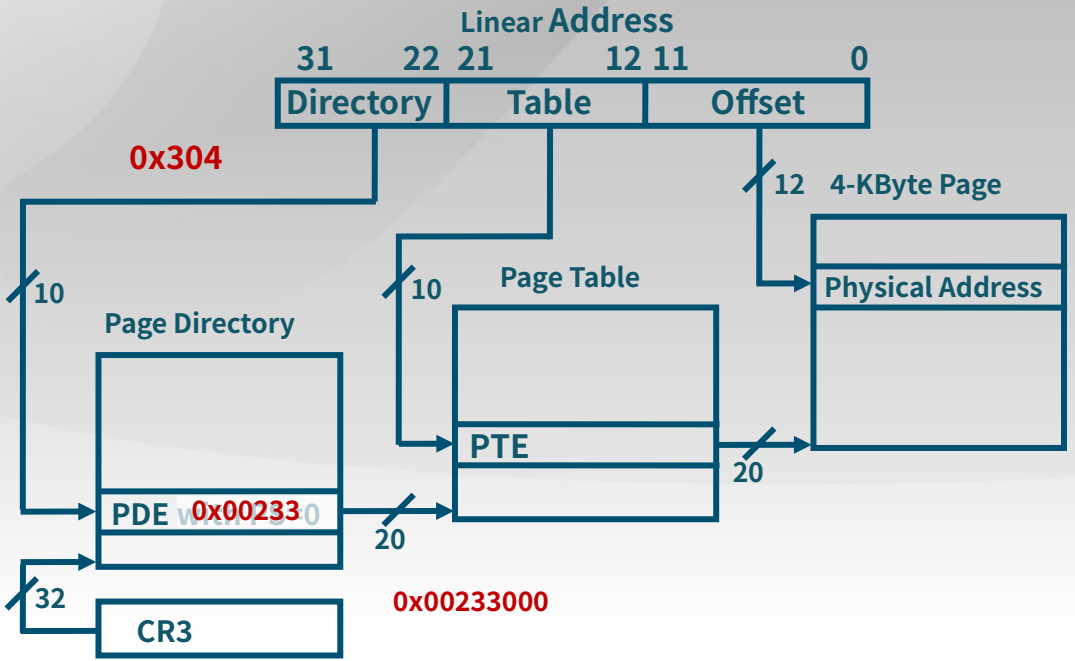
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



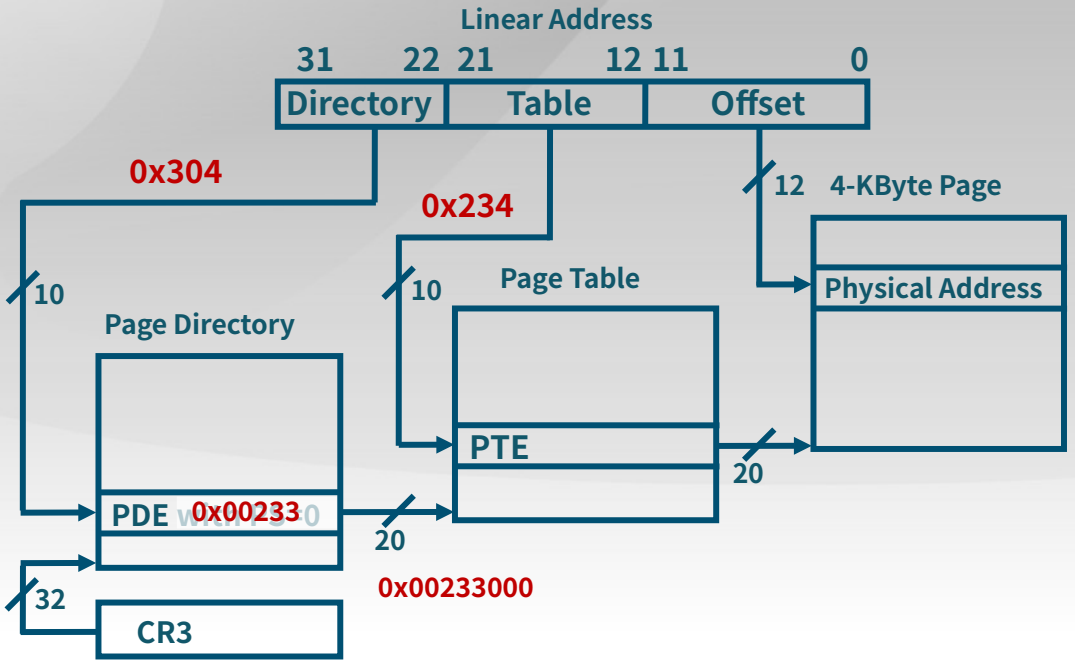
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



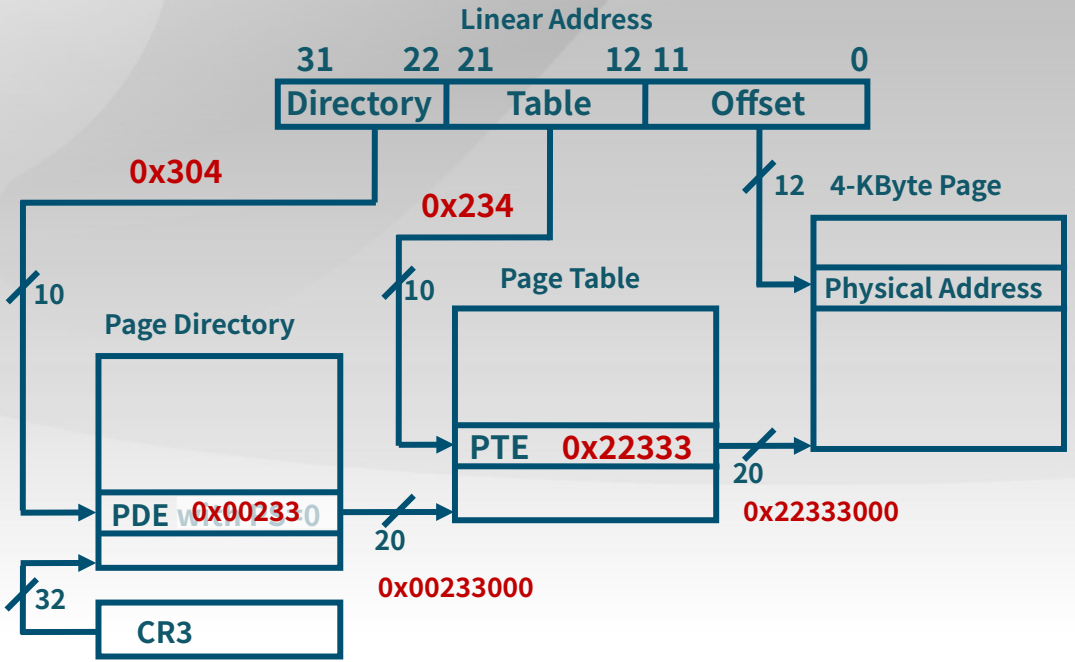
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



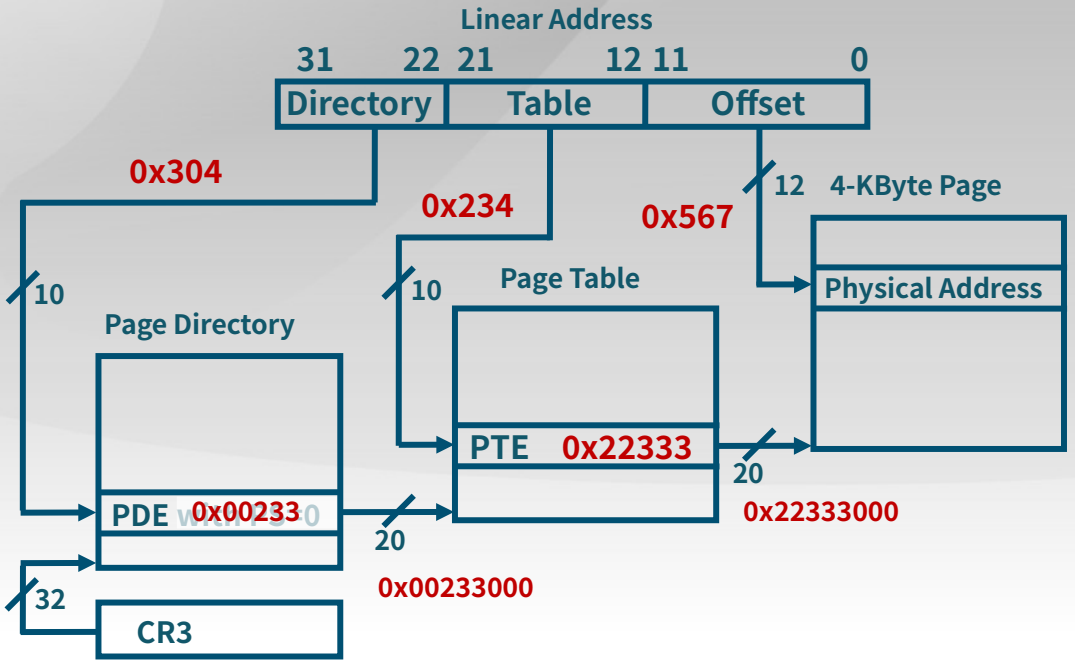
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



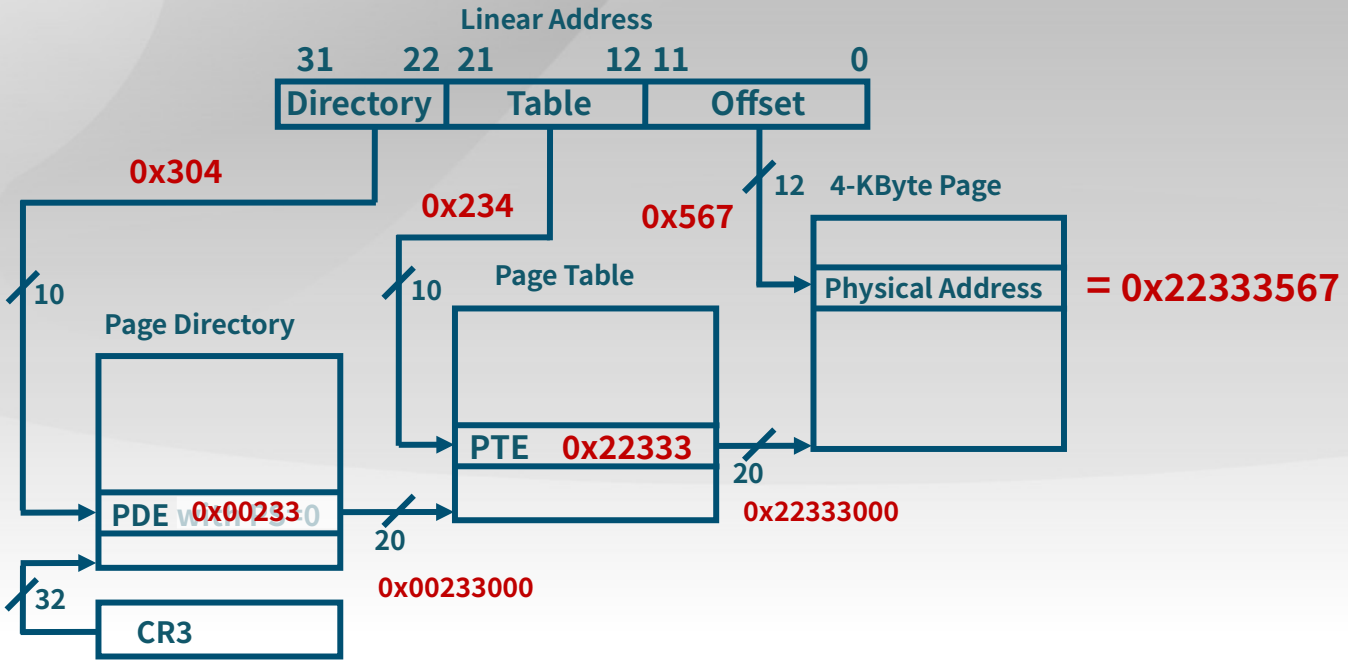
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



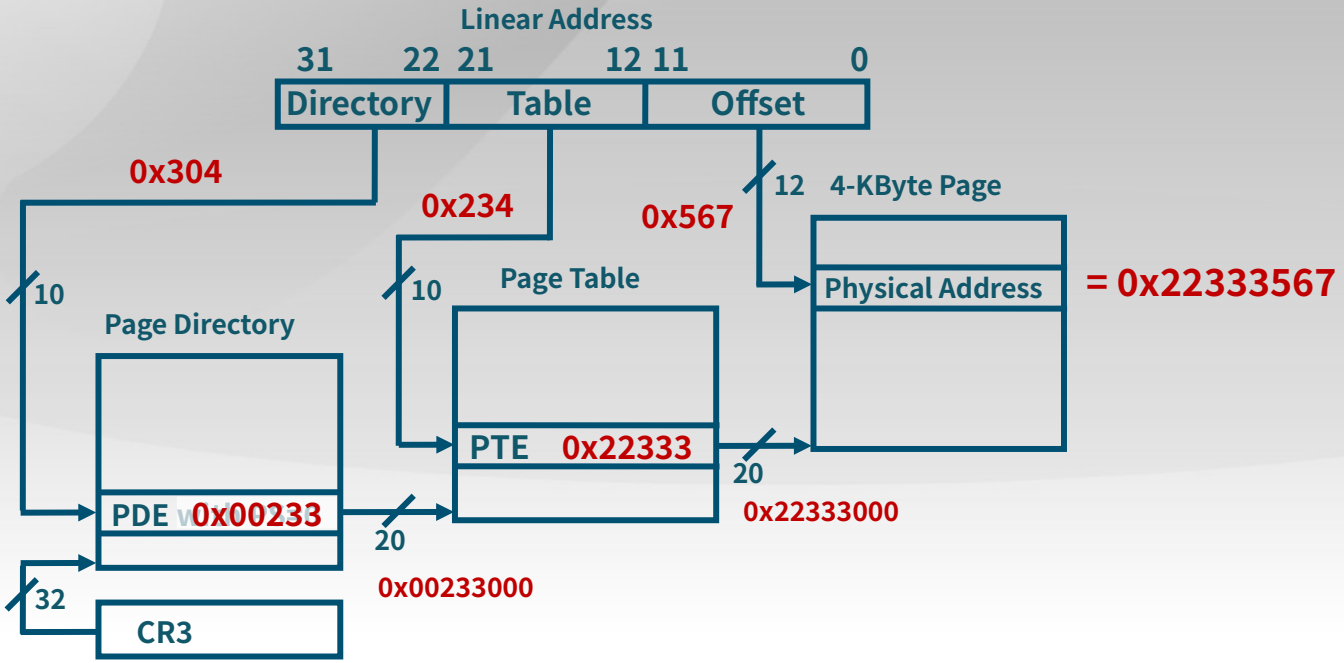
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



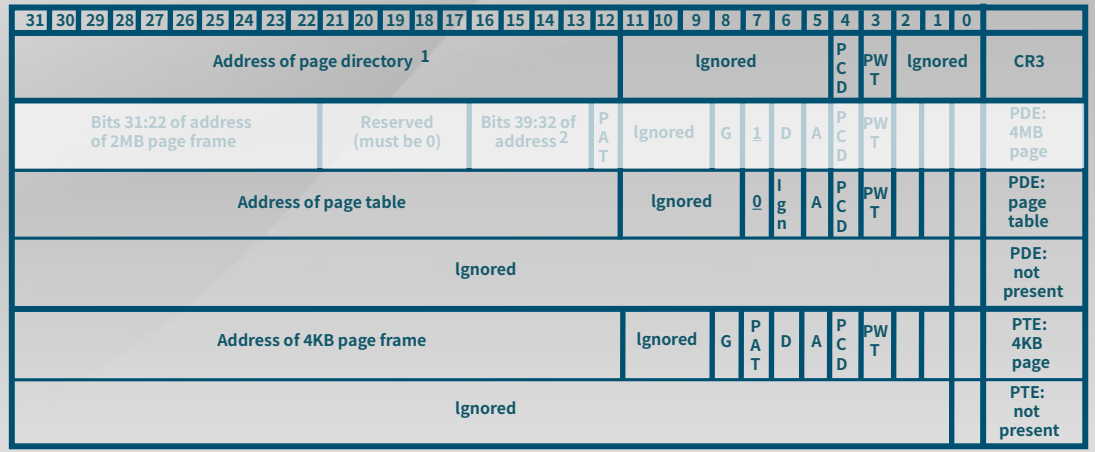
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



在页表项中存放的地址内容是线性地址（ linear addresses ）！

x86 MMU – 页表项（page table entries）



- R/W: 1 if this page is writable
- U/S: 1 if this page is accessible in ring 3
- A: 1 if this page has been accessed
- You may ignore others for now

图 页目录项和页表项的结构

x86 MMU – 使能页机制（enable paging）

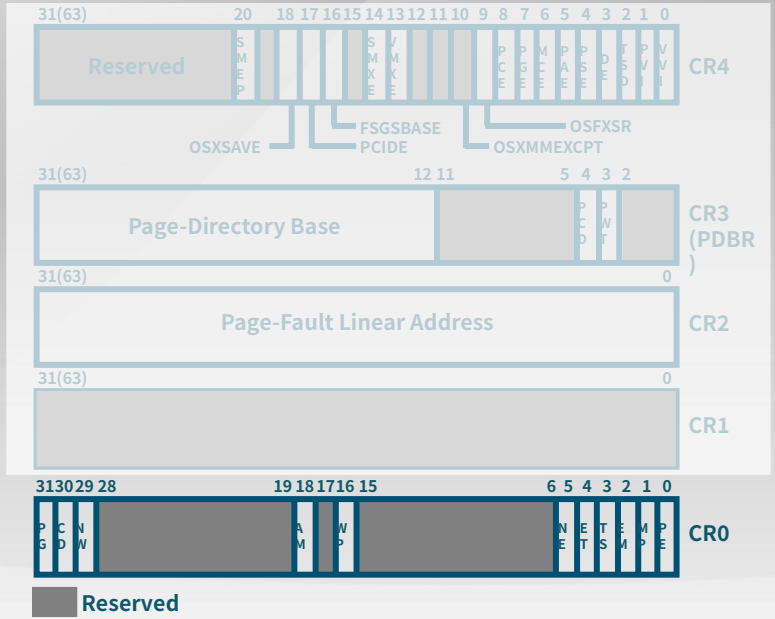
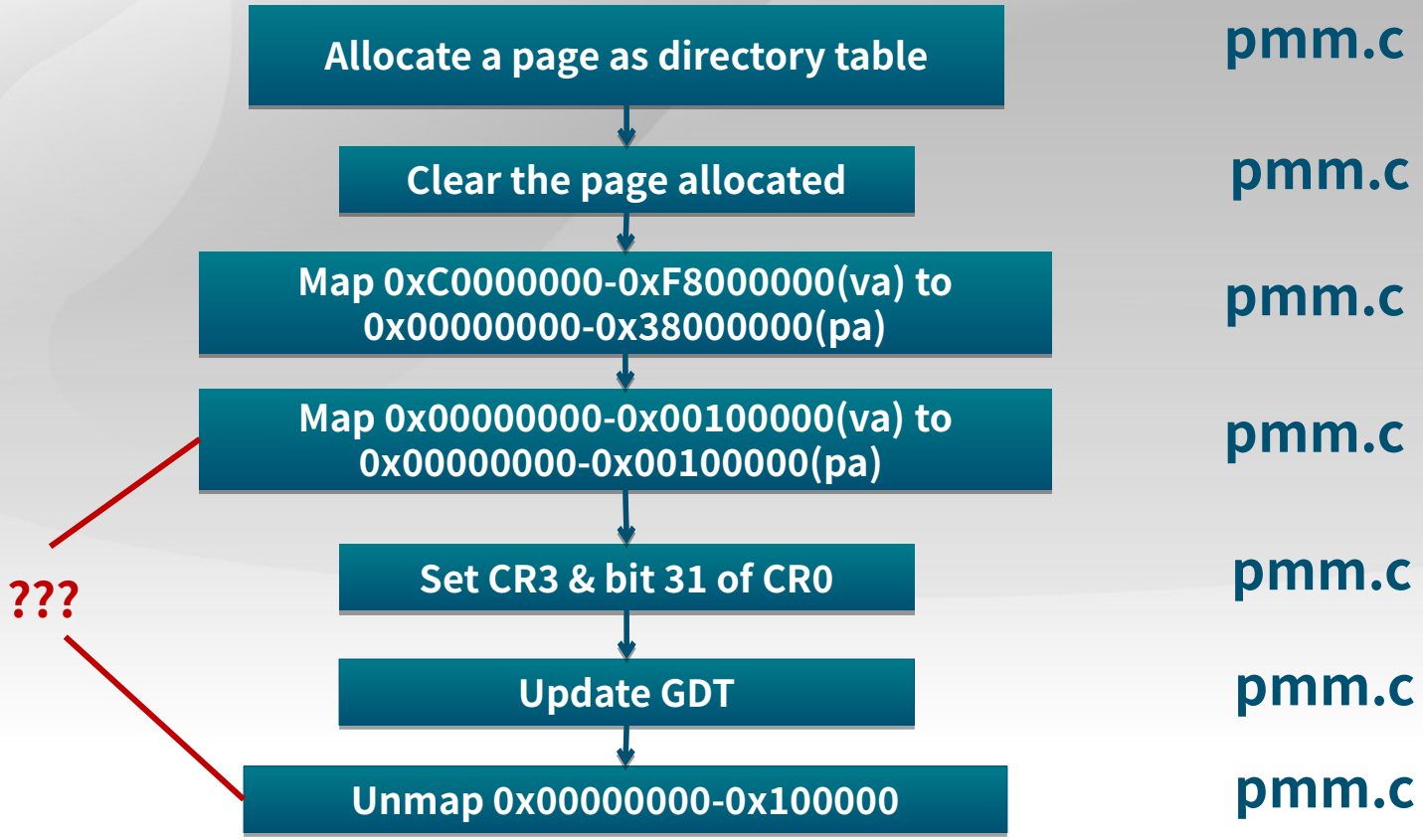


图 Control Registers

为了在保护模式下使能页机制，OS 需要置 CR0 寄存器中的 bit 31 (PG)

x86 MMU – 建立页表（page tables）



pmm.c

pmm.c

pmm.c

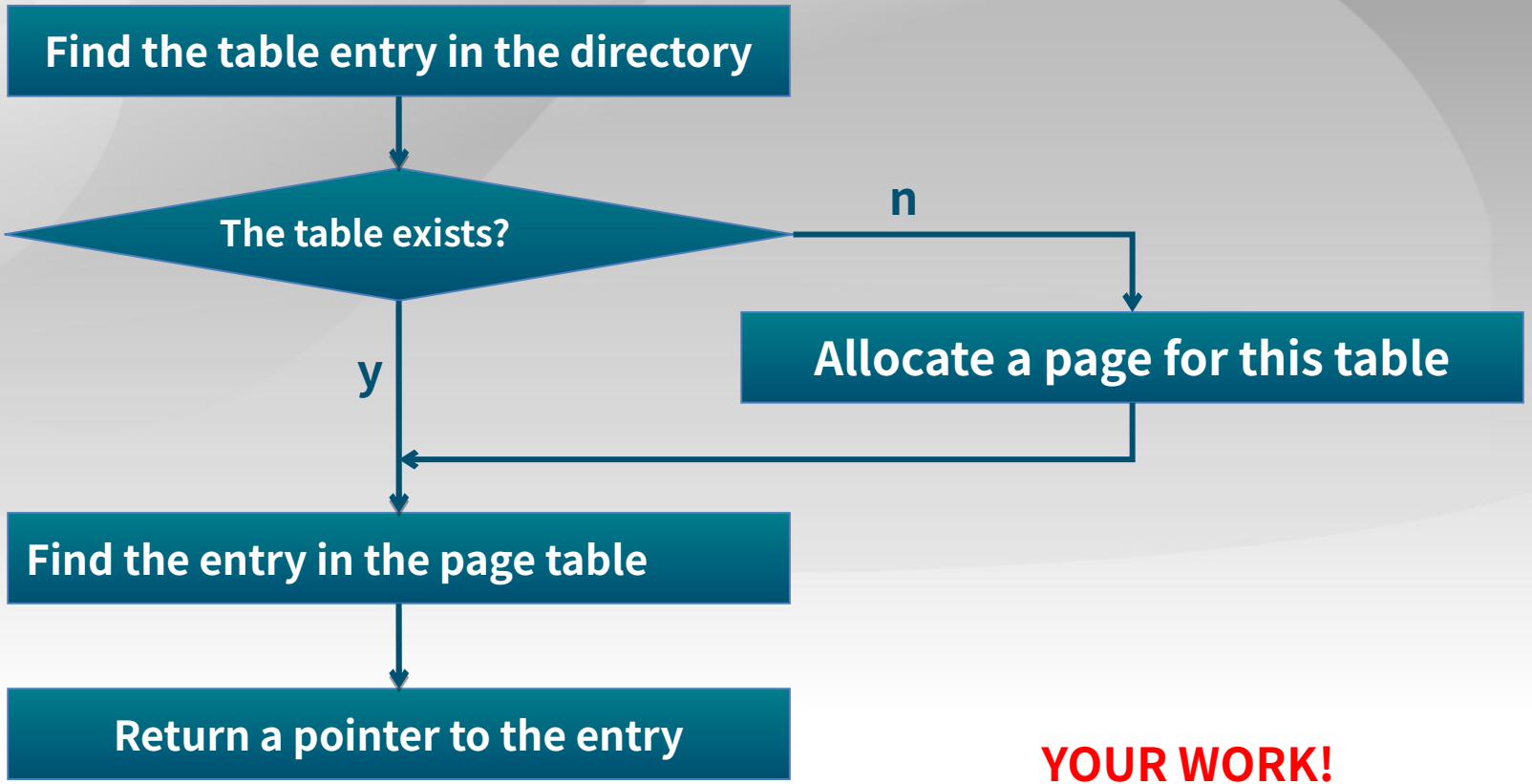
pmm.c

pmm.c

pmm.c

pmm.c

x86 MMU – 在页表中建立页的映射关系



YOUR WORK!

x86 MMU – 合并段机制 + 页机制（segmentation + paging）

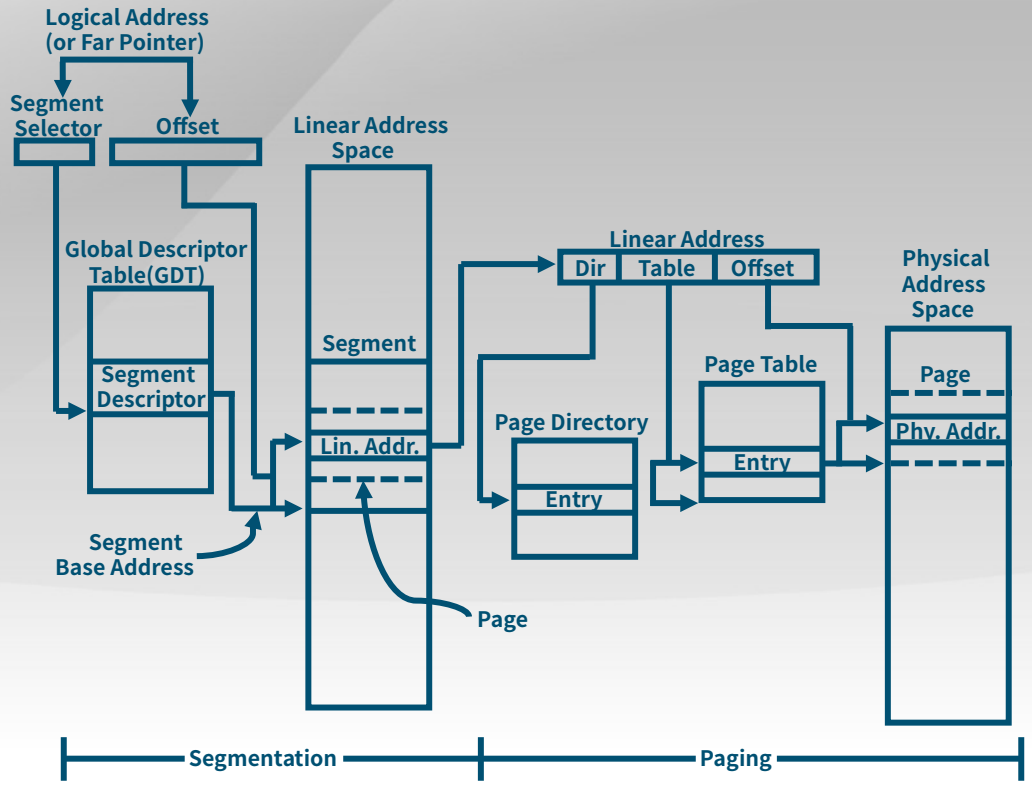
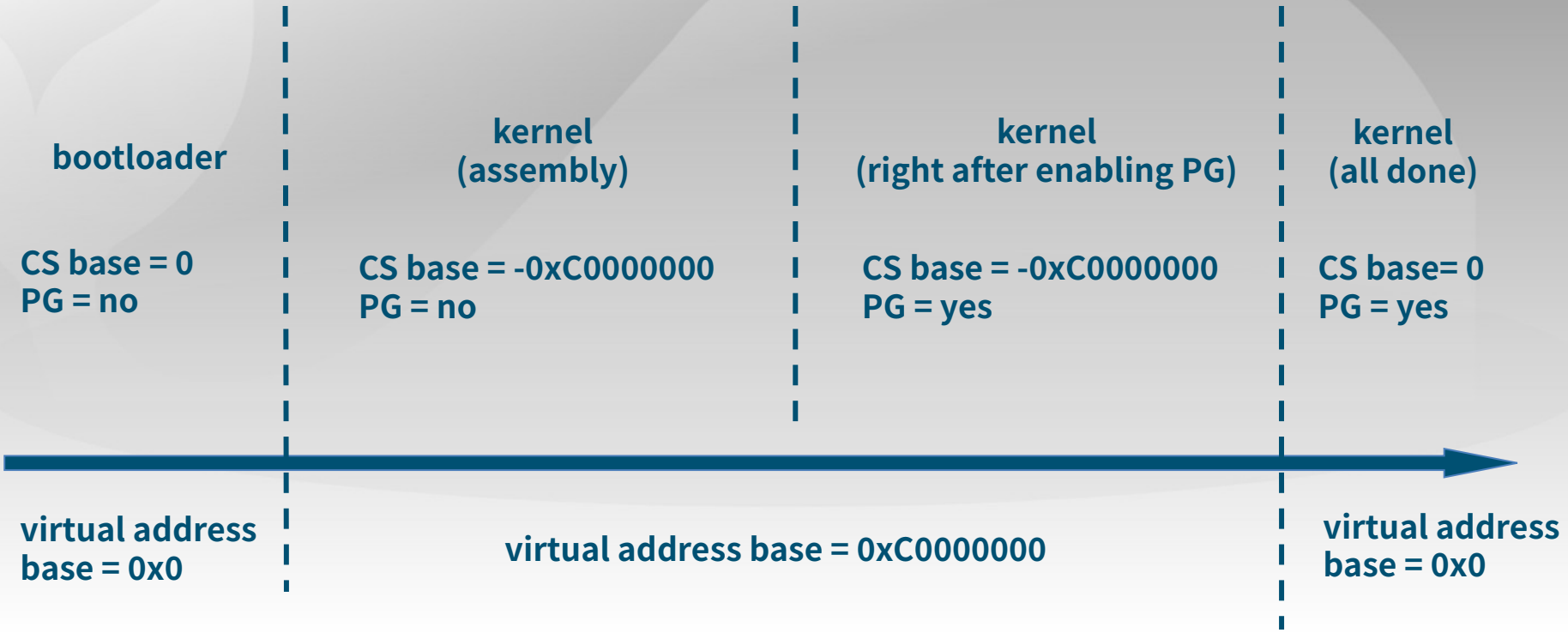


图 段页式内存映射机制 Segmentation and Paging

x86 MMU – uCore 内存管理初始化



x86 MMU – 参考资料

- Chap. 3 & 4, Vol. 3, Intel® and IA-32 Architectures Software Developer’ s Manual