操作系统原理与设计

第8章 Main Memory3

陈香兰

中国科学技术大学计算机学院

2009年11月17日

提纲

Discrete Memory Allocation

- paging（分页）
  - internal fragmentation <one page
- segmentation（分段）
  - logical
- combined paging & segmentation（段页式）

Outline

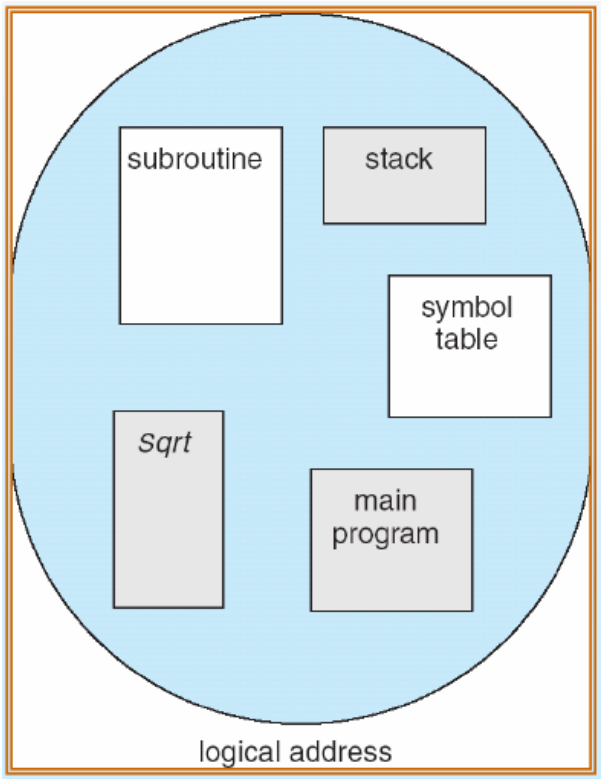
Segmentation

- Segmentation: supporting **user view** of memory

- A program is **a collection of segments**. **A segment is a logical unit** such as:

  main program, procedure,
  function, method,
  object, local variables,
  global variables, common block,
  stack, symbol table,
  arrays

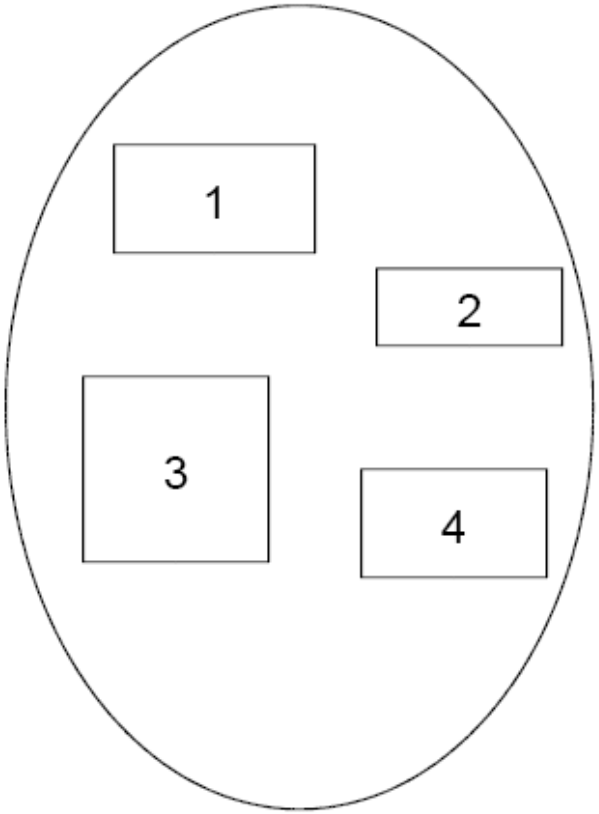
User's View of a Program



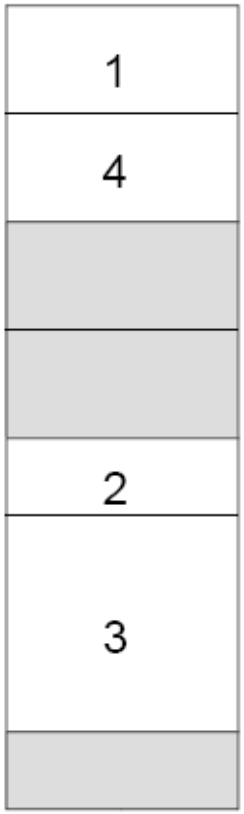
Logical address space

- A collection of segments
  - each segment $< name,\ length >$
  - 2-D address space
- A logical address consists of a two tuple
  - $< seg - name,\ offset >$, or
  - $< seg - num,\ offset >$
- Compiler automatically constructs segments reflecting the input program.
  - Pascal compiler
  - FORTRAN compiler
  - C compiler, such as gcc, ...

Logical View of Segmentation



user space



physical memory space

- Each segment is a logically integrated unit.
- Each segment is of variable length.
- Elements within one segment is addressed from the beginning of the segment.

Logical address =
<segment#, offset>

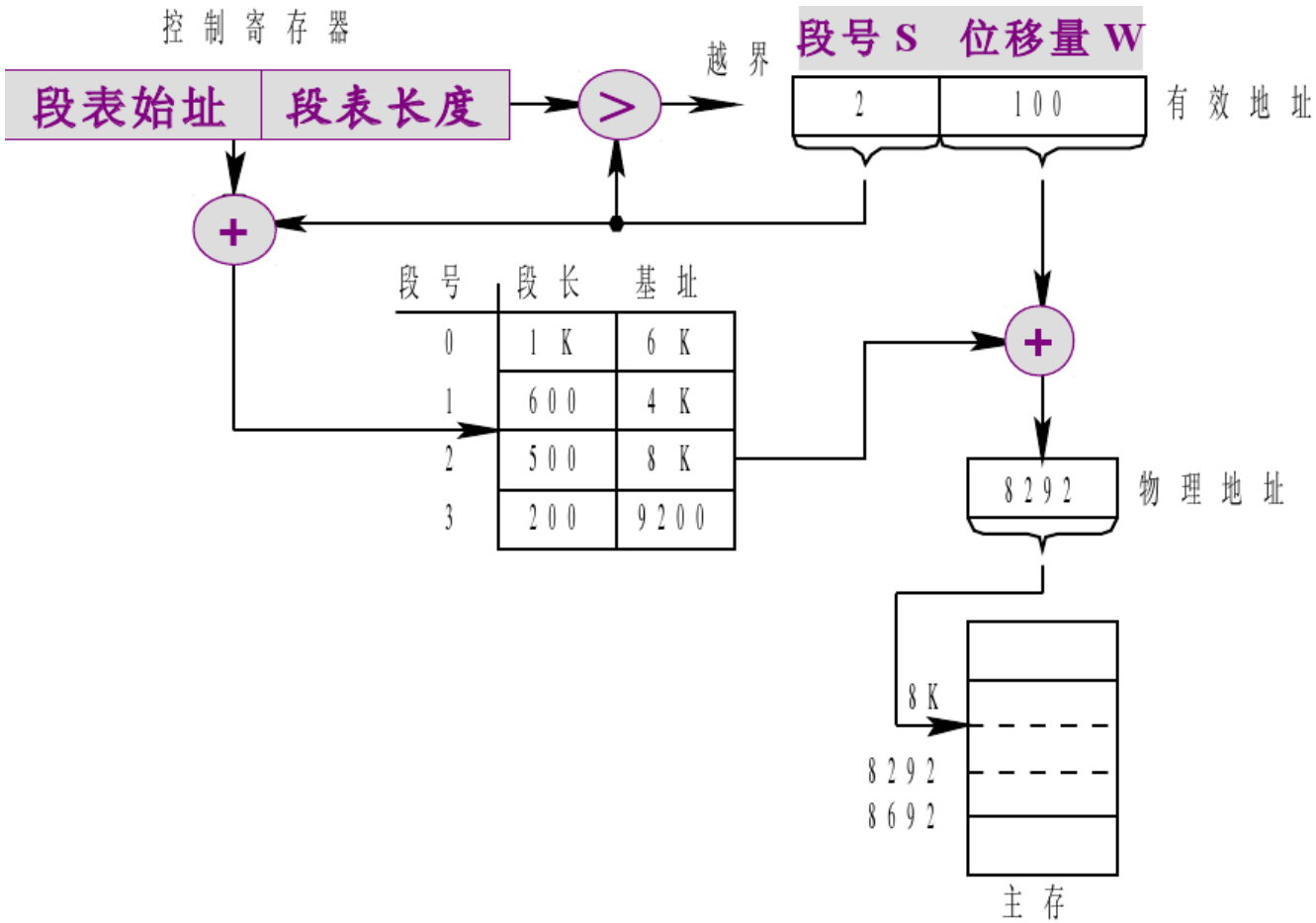
Outline

Segmentation Architecture I

- **Segment table** – maps **2-D LA** $\rightarrow$ **1-D PA**; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

  segment number s is legal if $s < \text{STLR}$

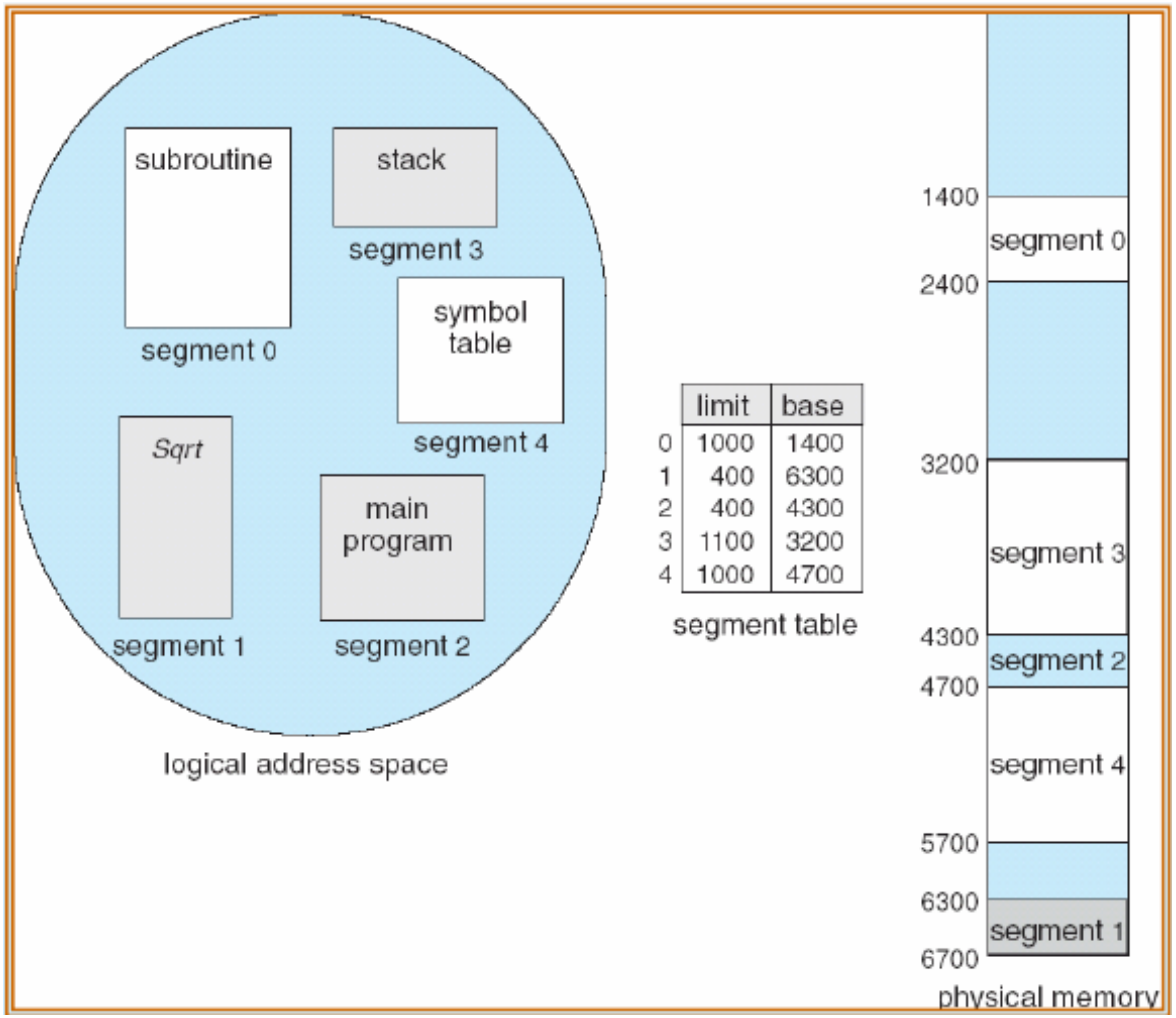
Segmentation Architecture II



Segmentation Architecture III

- Protection
  - With each entry in segment table associate:
    - validation bit $= 0 \Rightarrow$ illegal segment
    - read/write/execute privileges
  - Protection bits associated with segments; **code sharing** occurs at segment level
  - Since segments vary in length, memory allocation is a dynamic storage-allocation problem
    - First-fit, Best-fit, ...
    - Externel fragmentation, compaction, ...

Example of Segmentation



- $< segment2,\ 53 > \rightarrow$
  $4300 + 53 = 4353$
- $< setment3,\ 852 > \rightarrow$
  $3200 + 852 = 4052$
- What about
  $< segment1,\ 536 >$?

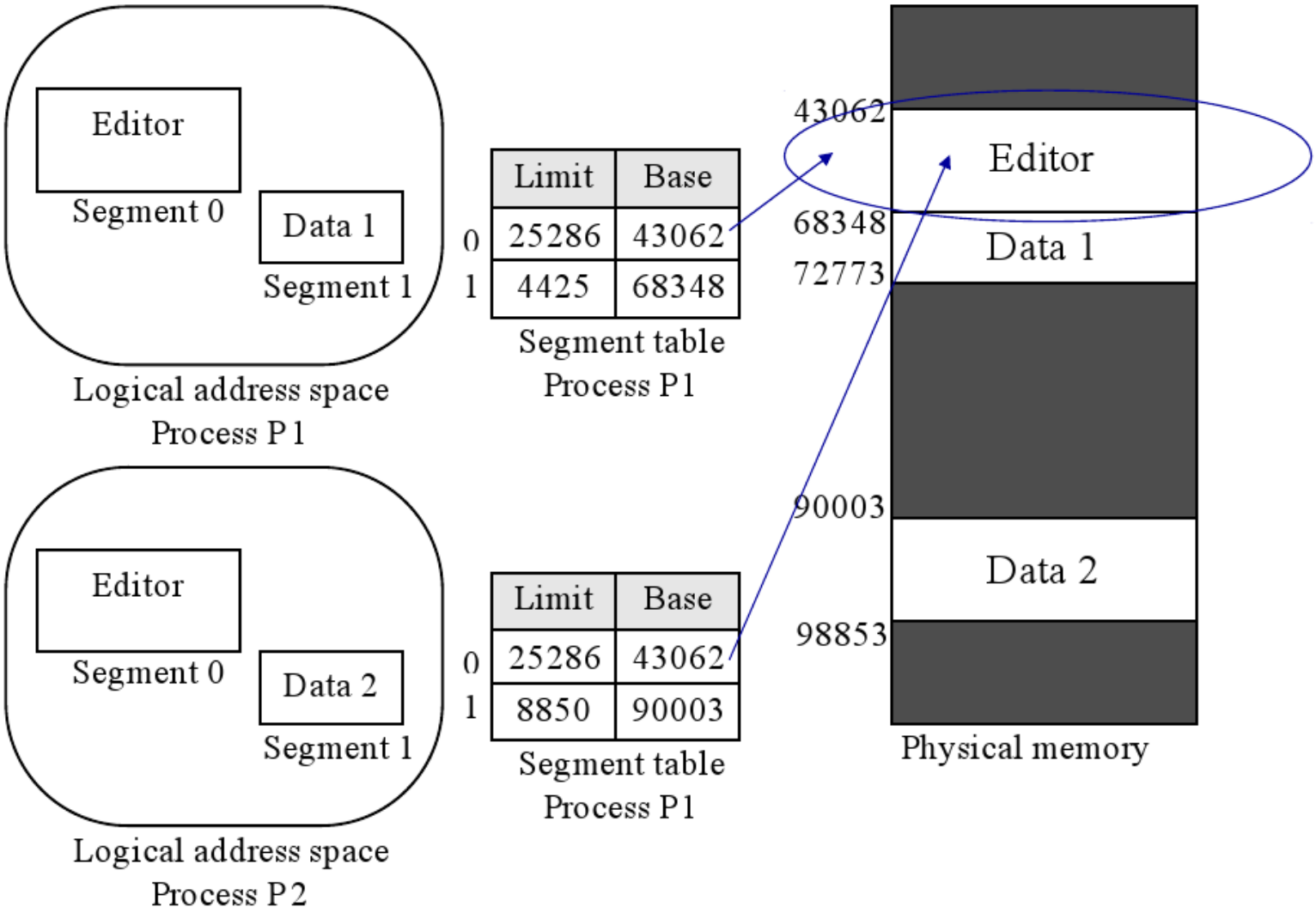
分页和分段的主要区别

- 角度和目的不同
  - 分页：面向系统，物理上离散，减少外部和内部碎片，提高内存利用率
    - 页是信息的物理单位
  - 分段：面向用户，逻辑上离散，满足用户的需求
    - 段是信息的逻辑单位，意义相对完整
- 大小不同
  - 分页：大小固定，由硬件决定
  - 分段：不固定，划分由程序决定，在编译时确定
- 地址空间的维数不同
  - 分页：1维
  - 分段：2维，段名(号)+段内偏移

分段的好处

- Relocation，可重定位，方便编程
  - Dynamic, by segment table
- Sharing
  - shared segments
  - same segment number
- Protection
  - Use segment table entry
  - Protection bit
    - Read-only, execute-only, read/write
    - Validation bit, 0 illegal segment
  - Protection & sharing at segment level
- 动态增长
- 动态链接

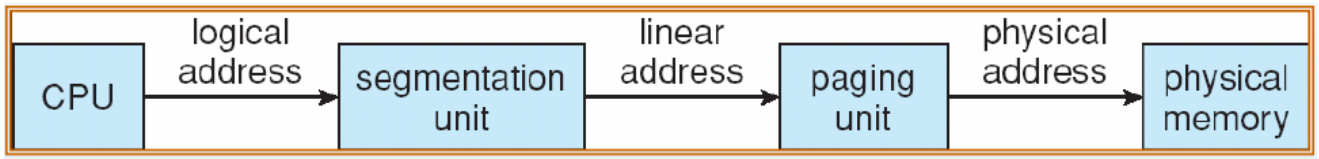
sharing


Editor
Segment 0
Data 1
Segment 1
Logical address space
Process P1
Limit
Base
0
25286
43062
1
4425
68348
Segment table
Process P1
Editor
Segment 0
Data 2
Segment 1
Logical address space
Process P2
Limit
Base
0
25286
43062
1
8850
90003
Segment table
Process P1
43062
Editor
68348
Data 1
72773
90003
Data 2
98853
Physical memory

Outline

Example: The Intel Pentium

- Supports both **pure segmentation** & **segmentation with paging**
- CPU generates logical address
  1. Logical address given to segmentation unit
     - Which produces linear addresses



  2. Linear address given to paging unit
     - Which generates physical address in main memory
     - Paging units form equivalent of MMU

page number		page offset
p_1	p_2	d
10	10	12

Intel Pentium Segmentation I

- I386体系结构采用分段机制
  - 逻辑地址=段：段内偏移
- 使用16位段寄存器来指明当前所使用的段
  - 有六个：cs, ss, ds, es, fs和gs
    CPU规定了3个寄存器的专门的用途
    - cs　　代码段寄存器，指向存放程序指令的段
    - ss　　堆栈段寄存器，指向存放当前堆栈的段
    - ds　　数据段寄存器，指向存放数据的段
- 自80386，Intel微处理器以两种不同的方式执行地址转换
  - 实模式（20位）
    - 16位段寄存器只记录段基址的高16位，因此段基址必须4位对齐（末4位为0）

Intel Pentium Segmentation II

- 物理地址=逻辑地址=段寄存器值*16+段内偏移

- 保护模式（32位）
  - 16位段寄存器无法直接记录段的信息，因此需要与全局描述符表GDT配合使用
  - GDT中记录了每个段的信息（段描述符），段寄存器只需记录段在GDT中的序号
  - **注意**：cs寄存器还有一个很重要的功能:它含有一个两位的域，用以指明CPU的当前特权级CPL (current privilege level)，值为0代表最高优先级，值为3代表最低优先级
  - 线性地址=段基地址+段内偏移
    其中，段基地址是根据段寄存器所指明的GDT中的段描述符中的信息得到的
  - 物理地址：根据页表对线性地址进行转换而得到

- GDT和段描述符 (segment descriptor)
  - 每个段由一个段描述符来表示，一个段描述符长度为8字节

Intel Pentium Segmentation III

- 全局描述符表GDT (global description table)：存放段描述符
  - GDT表也存放在RAM中，并使用一个专门的寄存器GDTR来指示GDT表在RAM中的位置（物理起始地址）
- 局部描述符表LDT（Local Description Table）
  - 根据x86，每个进程可以设置一个LDT
  - LDT表也存放在RAM中，使用LDTR来指示当前的LDT表

- 由于段的用途不一样，Intelx86提供下列几种段描述符
  - 数据段描述符（Data Segment Descriptor）
    - 可以描述各种用户数据段和堆栈段
  - 代码段描述符（Code Segment Descriptor）
    - 描述一个用户代码段
  - 任务状态段描述符（Task State Segment Descriptor）

Intel Pentium Segmentation IV

- 描述一个任务的状态段
- 局部描述符表描述符
  - 描述一个LDT段系统段描述符（System Segment Descriptor）

- 段描述符主要描述如下内容
  - 段的物理起始地址（base字段，32位）
  - 段长度（limit字段，20位）
  - 段长度的单位（粒度，G标志，1位） 0：字节为单位 1：4KB为单位
  - 是否系统段（S标志，1位） 0：系统段 1：普通的段
  - 类型字段（Type字段，4位）
    - 例如代码段、数据段、任务状态段、局部描述符段等等
  - 段的特权级描述字段（DPL字段，2位）
    - 00b：只能被CPL=00b的内核代码段访问

Intel Pentium Segmentation V

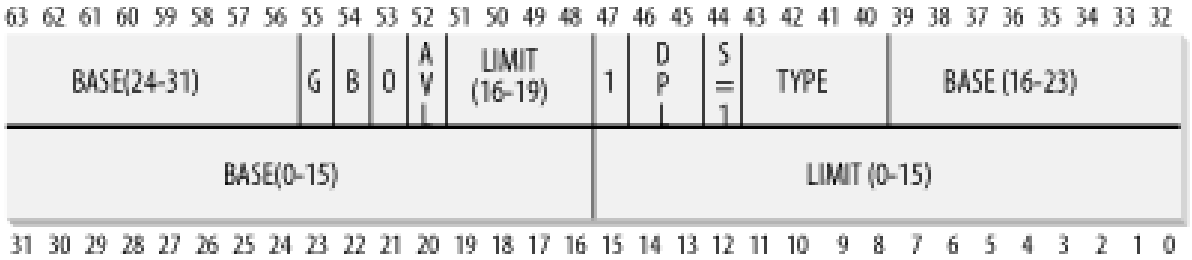
- ...
- 11b：可以被任意代码段访问

- 段存在标志（1位）
  - 0：该段当前不在内存中
  - 1：该段当前在内存中

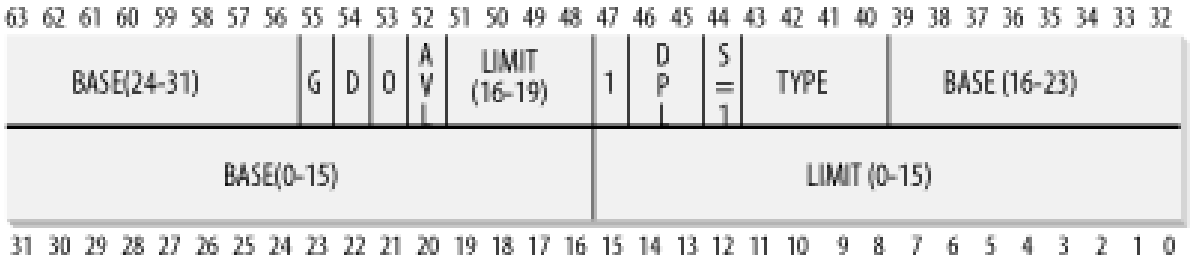
- ...

Intel Pentium Segmentation VI

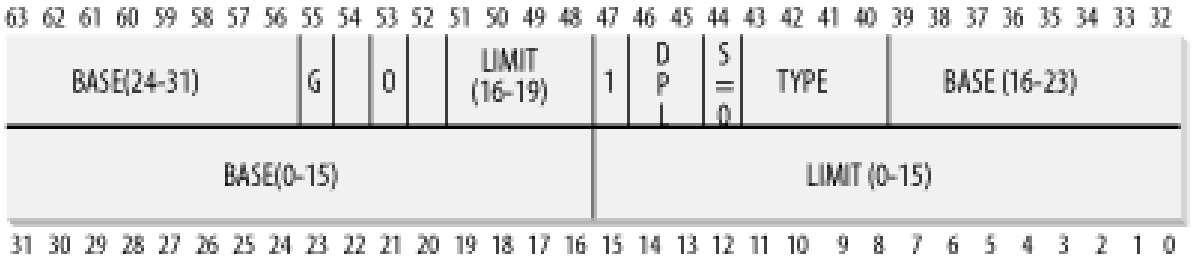
Data Segment Descriptor



Code Segment Descriptor

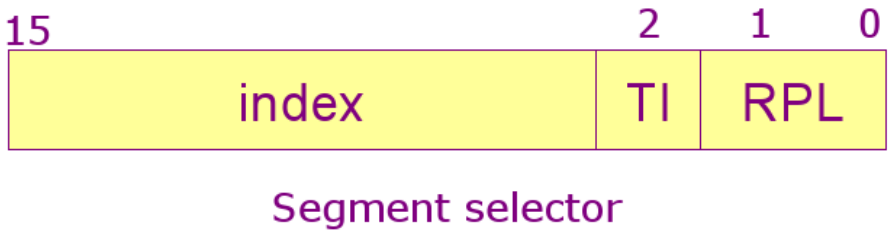


System Segment Descriptor



Intel Pentium Segmentation VII

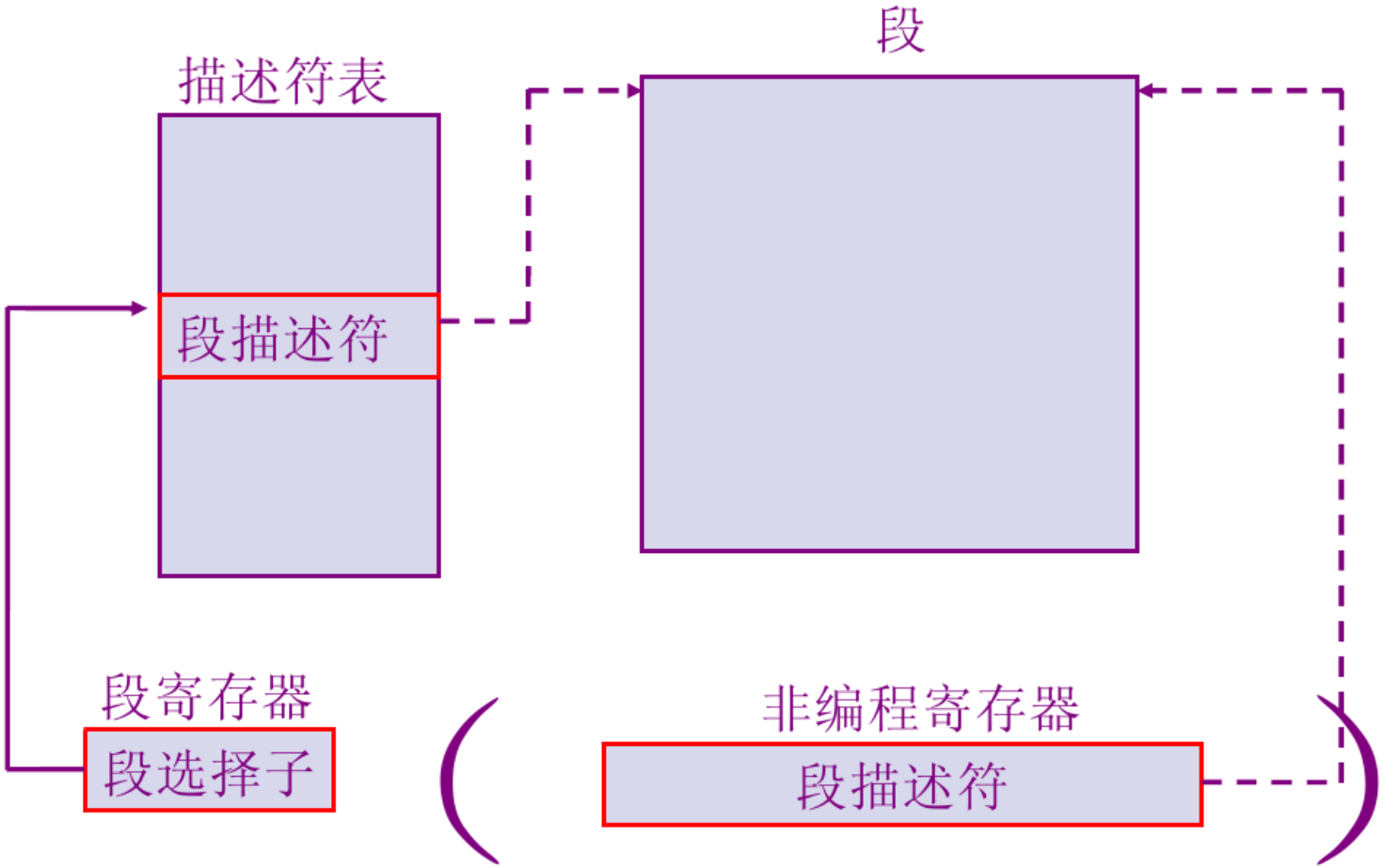
- 段选择子（Segment Selector）
  - 16位段寄存器与GDT或LDT配合起来对相应的段进行寻址
  - 段寄存器中的值称为段选择子，16位
    - 13位的索引，指定GDT表中的相应的段描述符
    - 1位的TI(Table Indicator)
      (跟LDT表有关，Linux中基本未使用）
    - 2位RPL(request privilege level)
      当相应的段选择符装入到cs寄存器中时，表明了CPU的当前特权级（用户/内核）



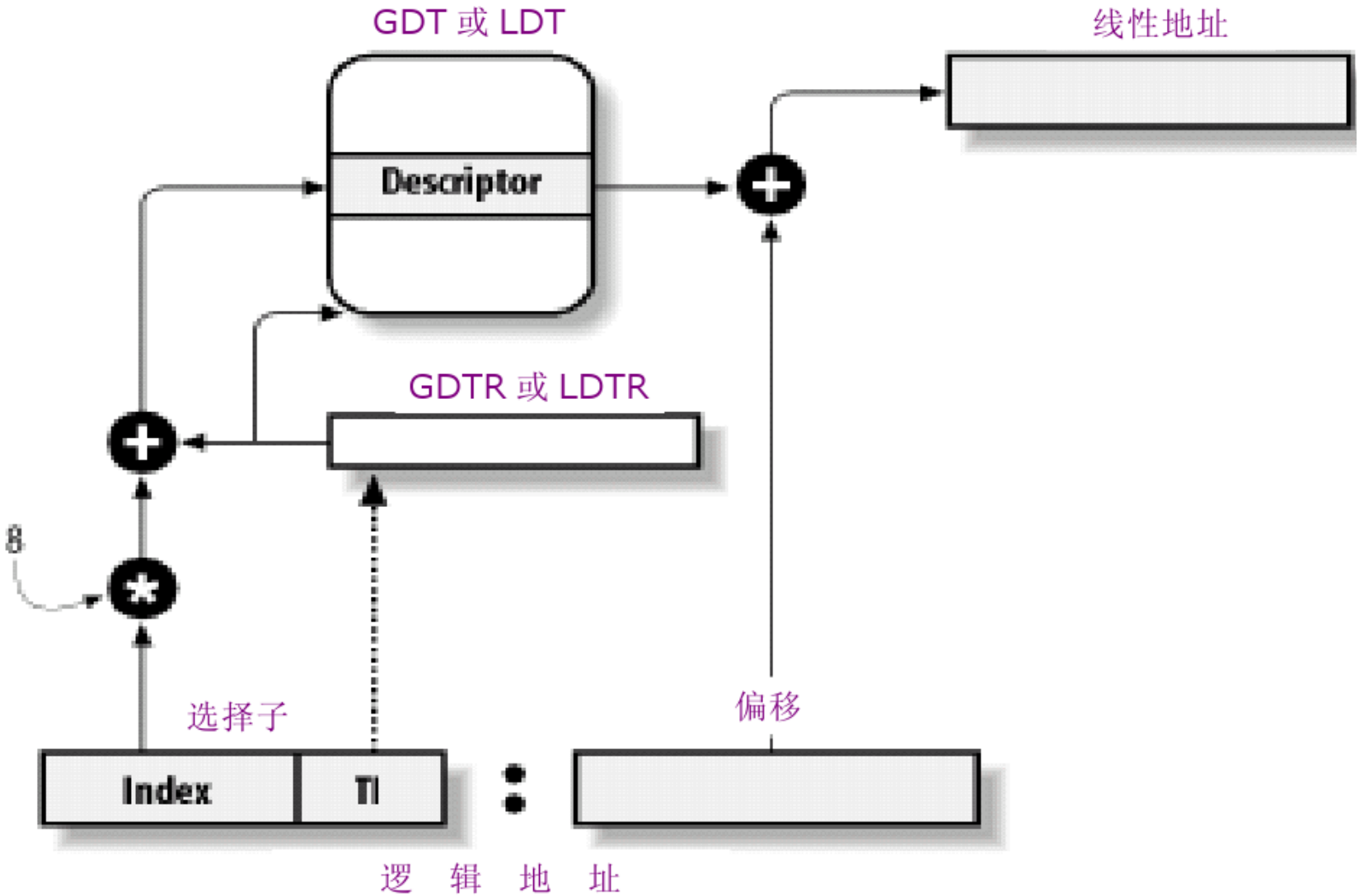
Segment selector

Intel Pentium Segmentation VIII

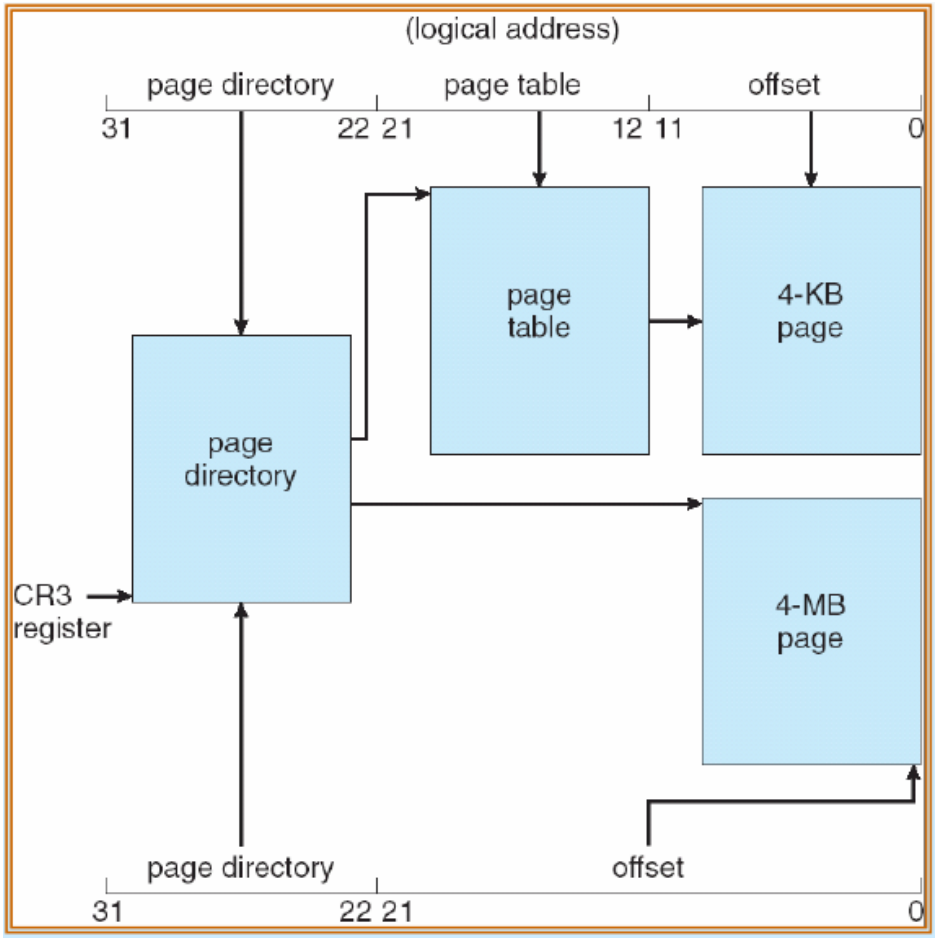
- 段选择子的使用和段描述符的快速访问



Intel Pentium Segmentation IX



Pentium Paging Architecture



Linux on Pentium Systems

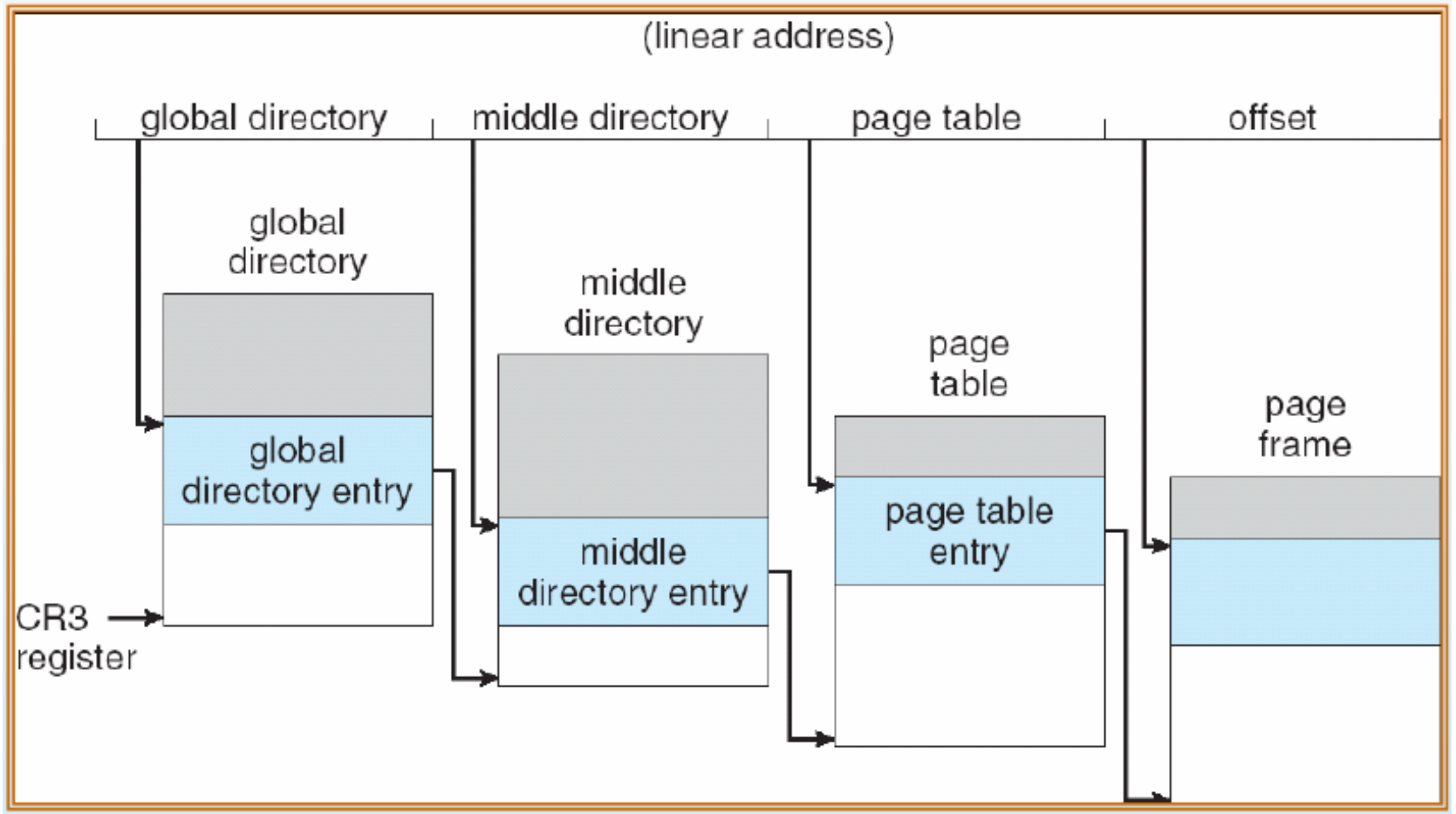
- Linux does not rely on segmentation and uses it minimally.
- only 6 segments
  - __KERNEL_CS, __KERNEL_DS
  - __USER_CS, __USER_DS
    - shared by all processes
    - all processes use the same logical address
  - A Task-state segment (TSS)
  - A default LDT segment, shared by all processes, usually not used

  (allow processes to create its own LDT replacing the default LDT)

Linear Address in Linux

- Broken into four parts:

Three-level Paging in Linux



小结

作业

- 非华夏班：9.1，9.2，9.5，9.8，9.10，9.16
- 华夏班：8.1，8.3，8.5，8.9，8.12
  - 说明基地址寄存器与可重定位寄存器有什么不同？

谢谢！