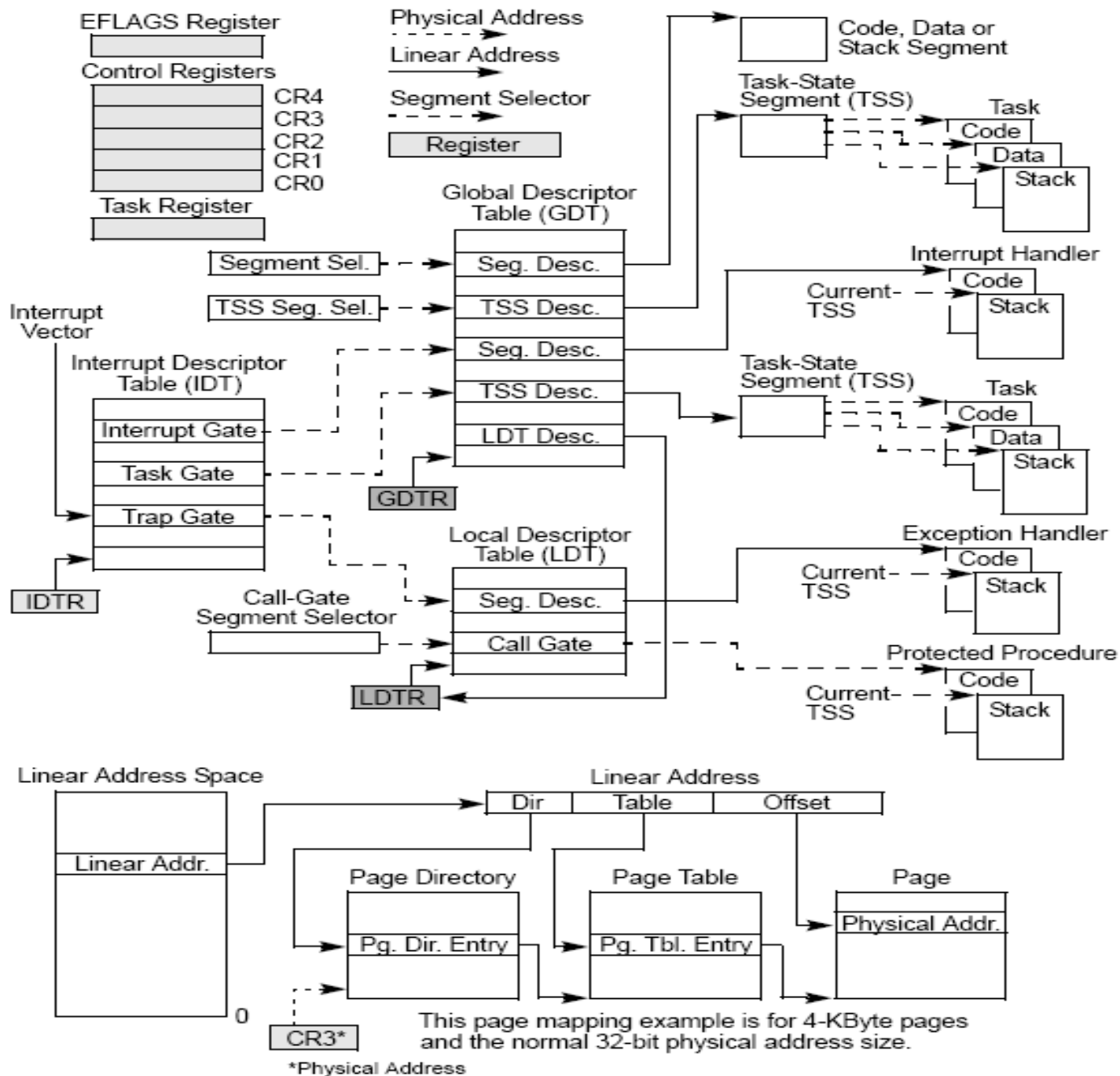


# Memory Addressing

- The address model discussed here pertains to the IA-32, 80x86 family of processors
- Three forms of address are considered
  - Logical address
  - Linear address (virtual address)
  - Physical address



# 80x86 System Level Registers



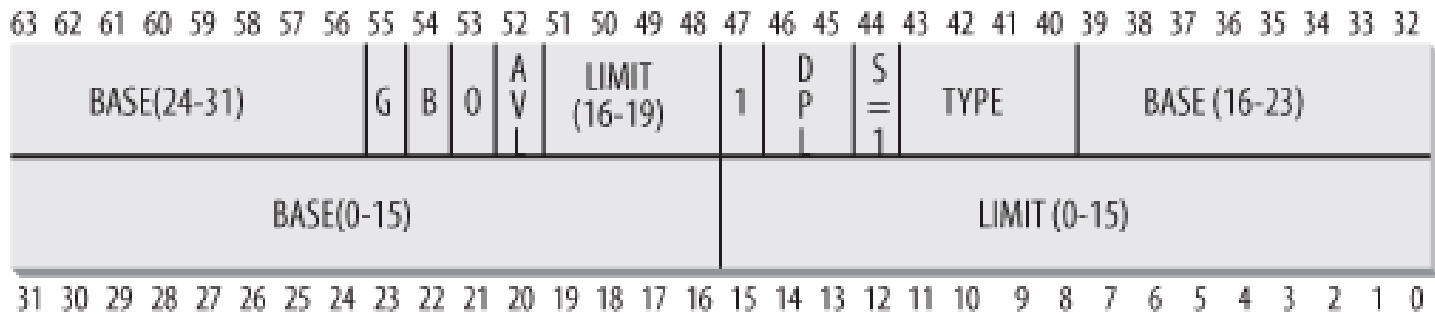
# Segment Selectors

- Six segment selectors (2 byte indexes) can be kept in registers to locate parts of a program by providing access to an 8 byte segment descriptor in a descriptor table (GDT or LDT)

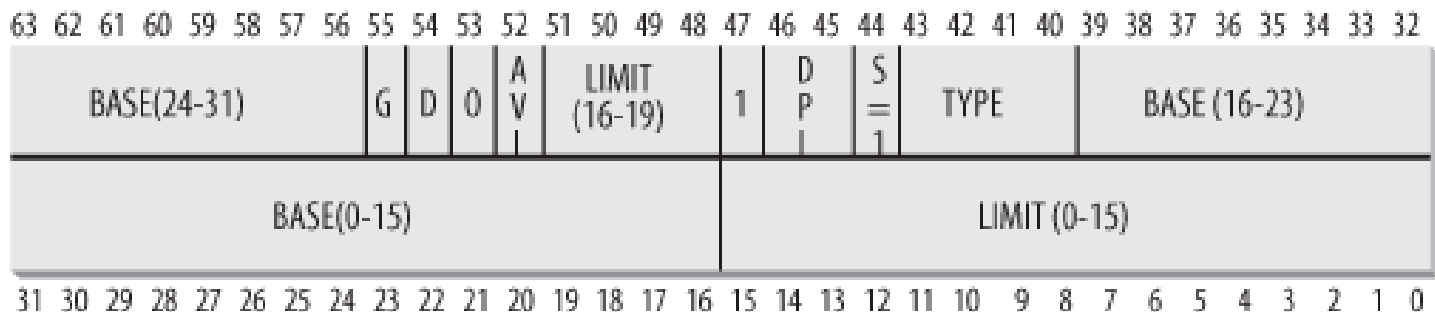


# 8 byte Descriptors

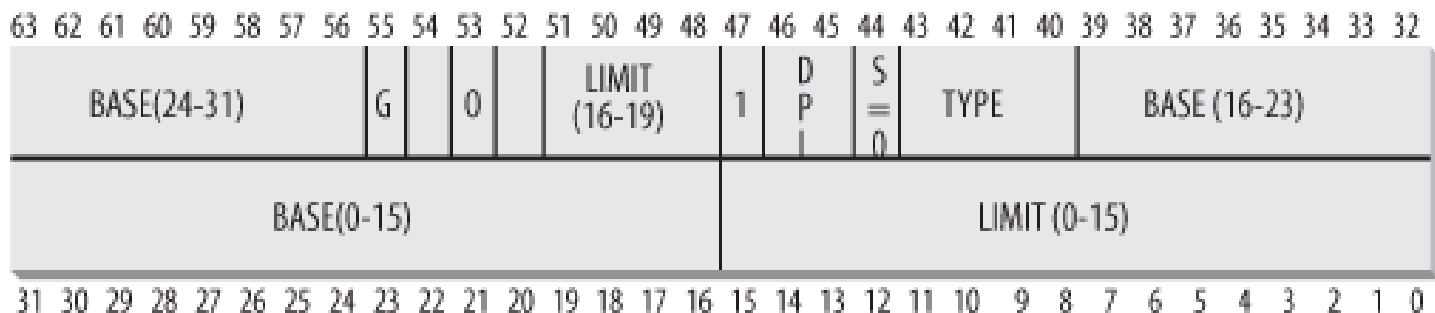
## Data Segment Descriptor



## Code Segment Descriptor



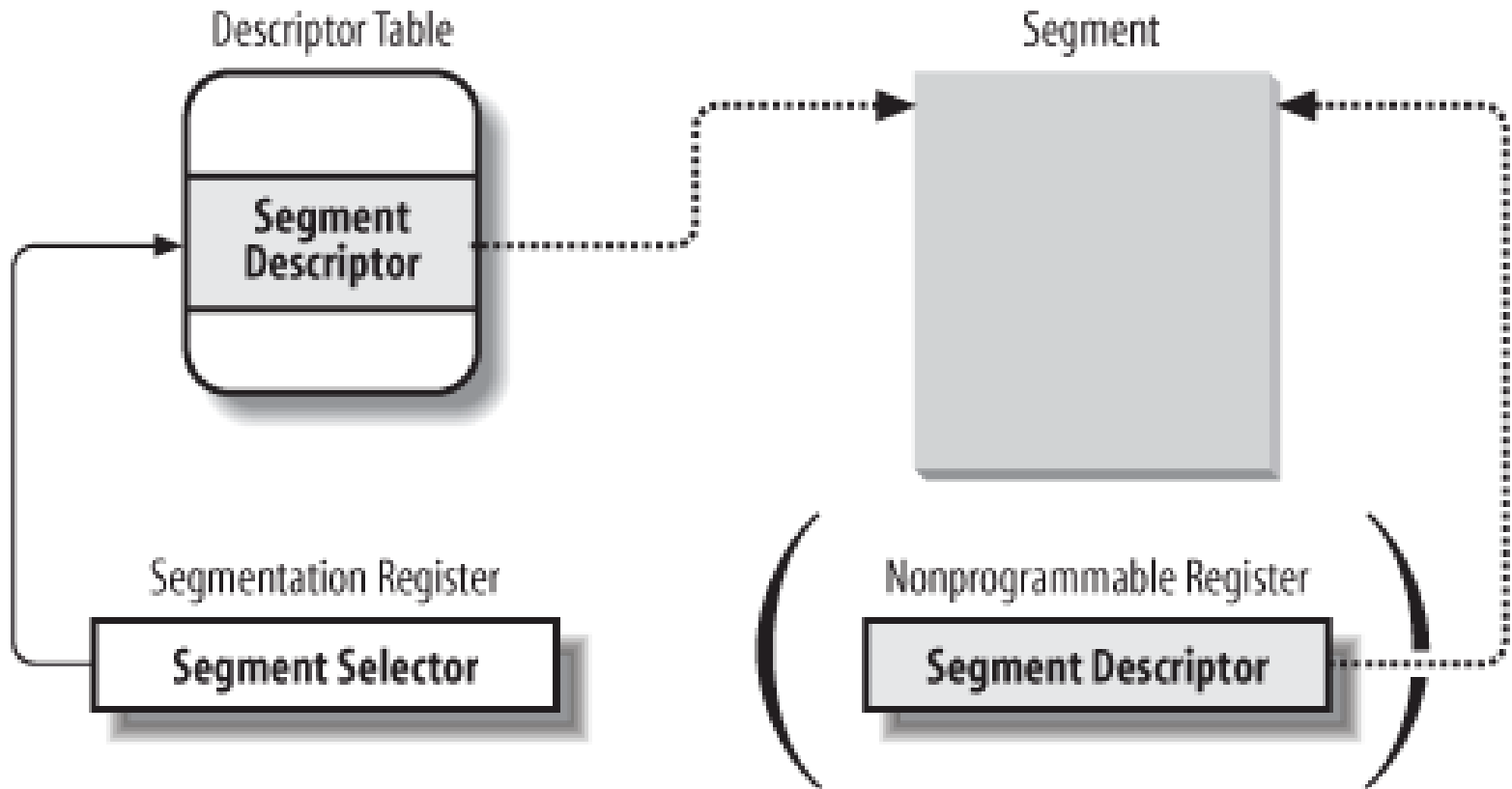
## System Segment Descriptor



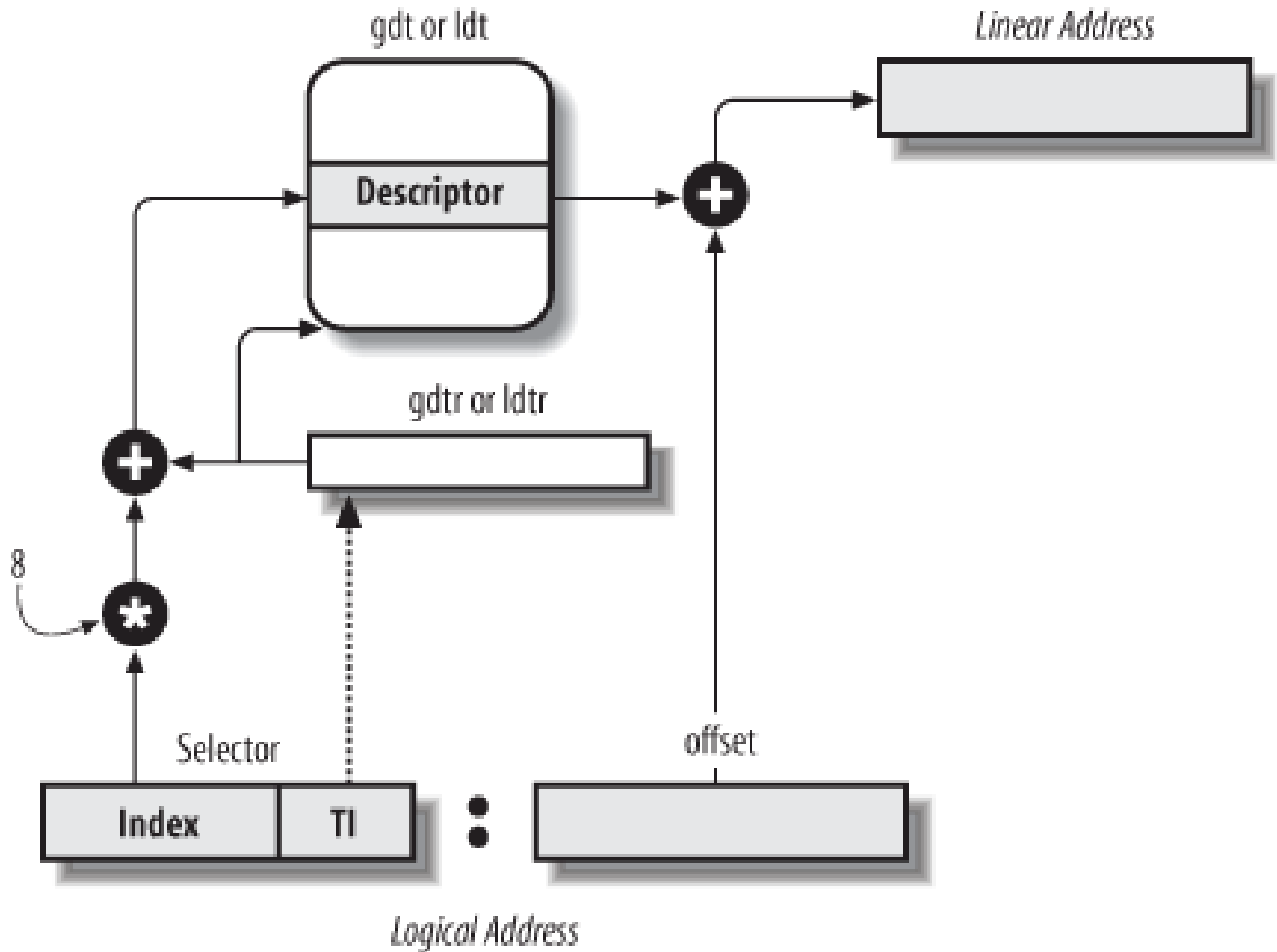
# Segment Descriptor fields

Field name	Description
<b>Base</b>	Contains the linear address of the first byte of the segment.
<b>G</b>	Granularity flag: if it is cleared (equal to 0), the segment size is expressed in bytes; otherwise, it is expressed in multiples of 4096 bytes.
<b>Limit</b>	Holds the offset of the last memory cell in the segment, thus binding the segment length. When G is set to 0, the size of a segment may vary between 1 byte and 1 MB; otherwise, it may vary between 4 KB and 4 GB.
<b>S</b>	System flag: if it is cleared, the segment is a system segment that stores critical data structures such as the Local Descriptor Table; otherwise, it is a normal code or data segment.
<b>Type</b>	Characterizes the segment type and its access rights.
<b>DPL</b>	Descriptor Privilege Level: used to restrict accesses to the segment. It represents the minimal CPU privilege level requested for accessing the segment. Therefore, a segment with its DPL set to 0 is accessible only when the CPL is 0 — that is, in Kernel Mode — while a segment with its DPL set to 3 is accessible with every CPL value.
<b>P</b>	Segment-Present flag : is equal to 0 if the segment is not stored currently in main memory. Linux always sets this flag (bit 47) to 1, because it never swaps out whole segments to disk.
<b>D or B</b>	Called D or B depending on whether the segment contains code or data. Its meaning is slightly different in the two cases, but it is basically set (equal to 1) if the addresses used as segment offsets are 32 bits long, and it is cleared if they are 16 bits long (see the Intel manual for further details).
<b>AVL</b>	May be used by the operating system, but it is ignored by Linux.

# Segment Selector and Descriptor Caching



# Translating a Logical Address



# Segmentation in Linux

- Linux uses segmentation only in a very limited way
- Since the 80x86 processor requires the CS and DS segments to be defined, Linux defines them both as starting at 0 and extending to the address limit 0xffffffff
- This is generally called the flat address space model

# The Linux GDT

- Each processor maintains a GDT with 18 entries
  - Kernel and user code and data
  - A Task State Segment (TSS)
  - The default Local Descriptor Table (LDT used by all processes)
  - Three Thread-Local Storage (TLS) segments
  - Three Advanced Power Management (APM) segments
  - Five segments related to Plug and Play (PnP)
  - A special “double fault” TSS segment

# Linux GDT

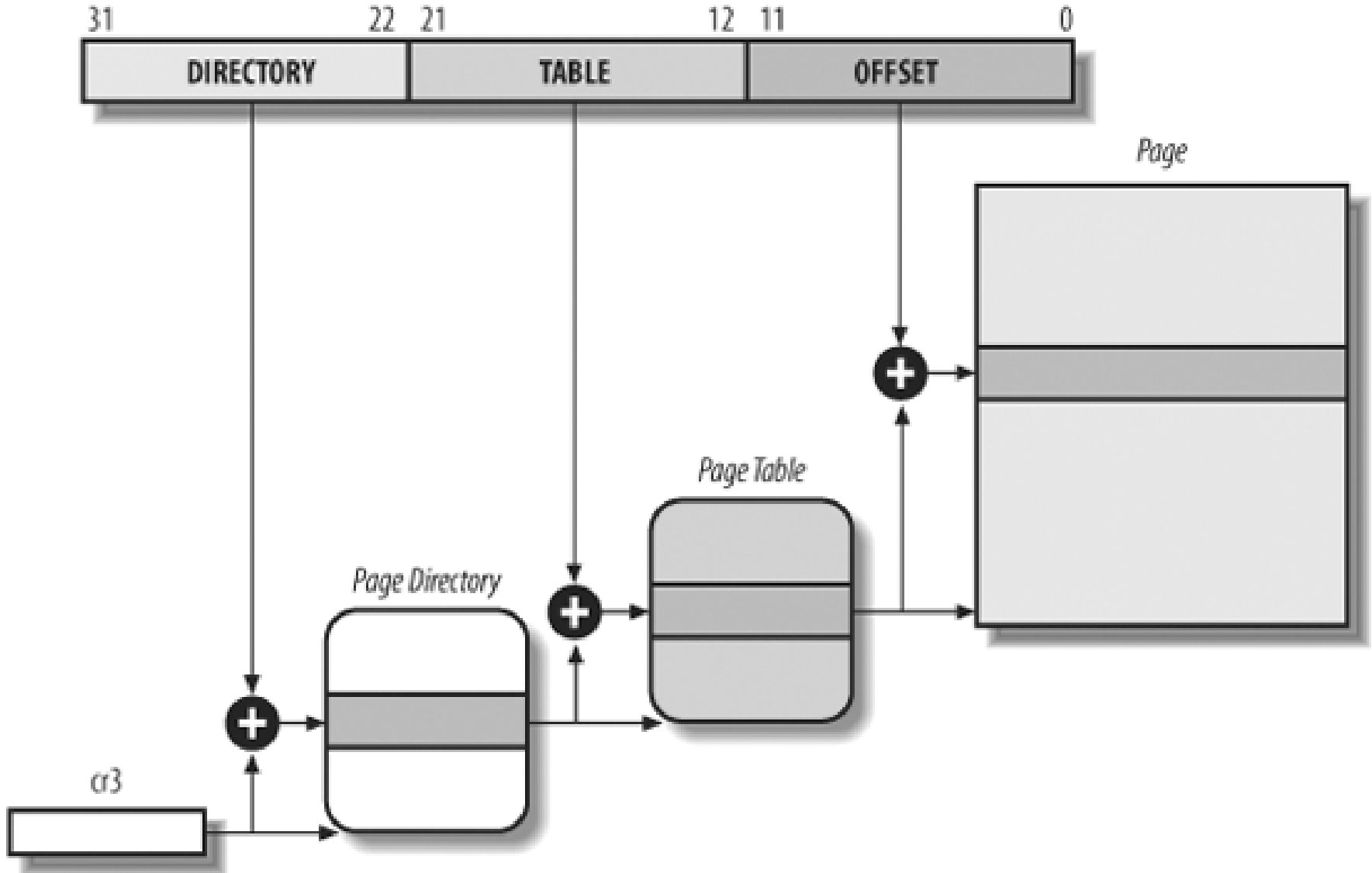
<i>Linux's GDT</i>	<i>Segment Selectors</i>	<i>Linux's GDT</i>	<i>Segment Selectors</i>
null	0x0	TSS	0x80
reserved		LDT	0x88
reserved		PNPBIOS 32-bit code	0x90
reserved		PNPBIOS 16-bit code	0x98
not used		PNPBIOS 16-bit data	0xa0
not used		PNPBIOS 16-bit data	0xa8
TLS #1	0x33	PNPBIOS 16-bit data	0xb0
TLS #2	0x3b	APMBIOS 32-bit code	0xb8
TLS #3	0x43	APMBIOS 16-bit code	0xc0
reserved		APMBIOS data	0xc8
reserved		not used	
reserved		not used	
kernel code	0x60 ( <code>__KERNEL_CS</code> )	not used	
kernel data	0x68 ( <code>__KERNEL_DS</code> )	not used	
user code	0x73 ( <code>__USER_CS</code> )	not used	
user data	0x7b ( <code>__USER_DS</code> )	double fault TSS	0xf8

# Paging

- Paging in the 80x86 processor is enabled by setting the PG bit in Control Register 0 (CR0)
- The Intel architecture specifies a 4KB page size
- A 32 bit Effective Address can be viewed as a 3 dimensional entity, comprised of a directory, table and offset component

# Intel Page Model

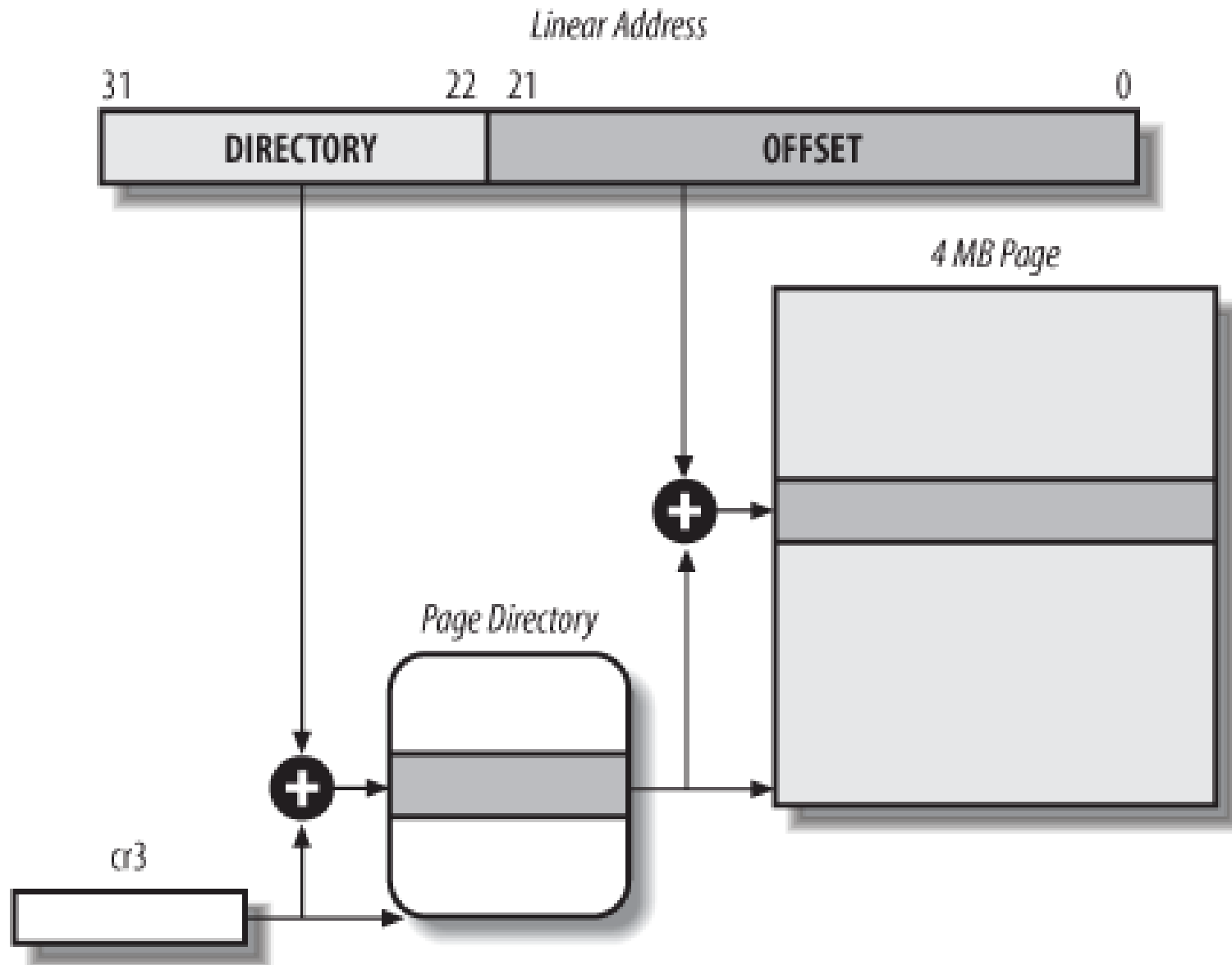
*Linear Address*



# Large Pages

- Intel support the use of extended paging, using a page size of 4MB instead of 4KB
- A directory table entry can be marked as a direct 4MB mapped location if the Page Size Flag is set
- Large pages require physically contiguous memory, but can save on TLB entries and generally make sense for mapping kernel components

# Mapping 4MB Pages



# The Physical Address Extension (PAE) Mechanism

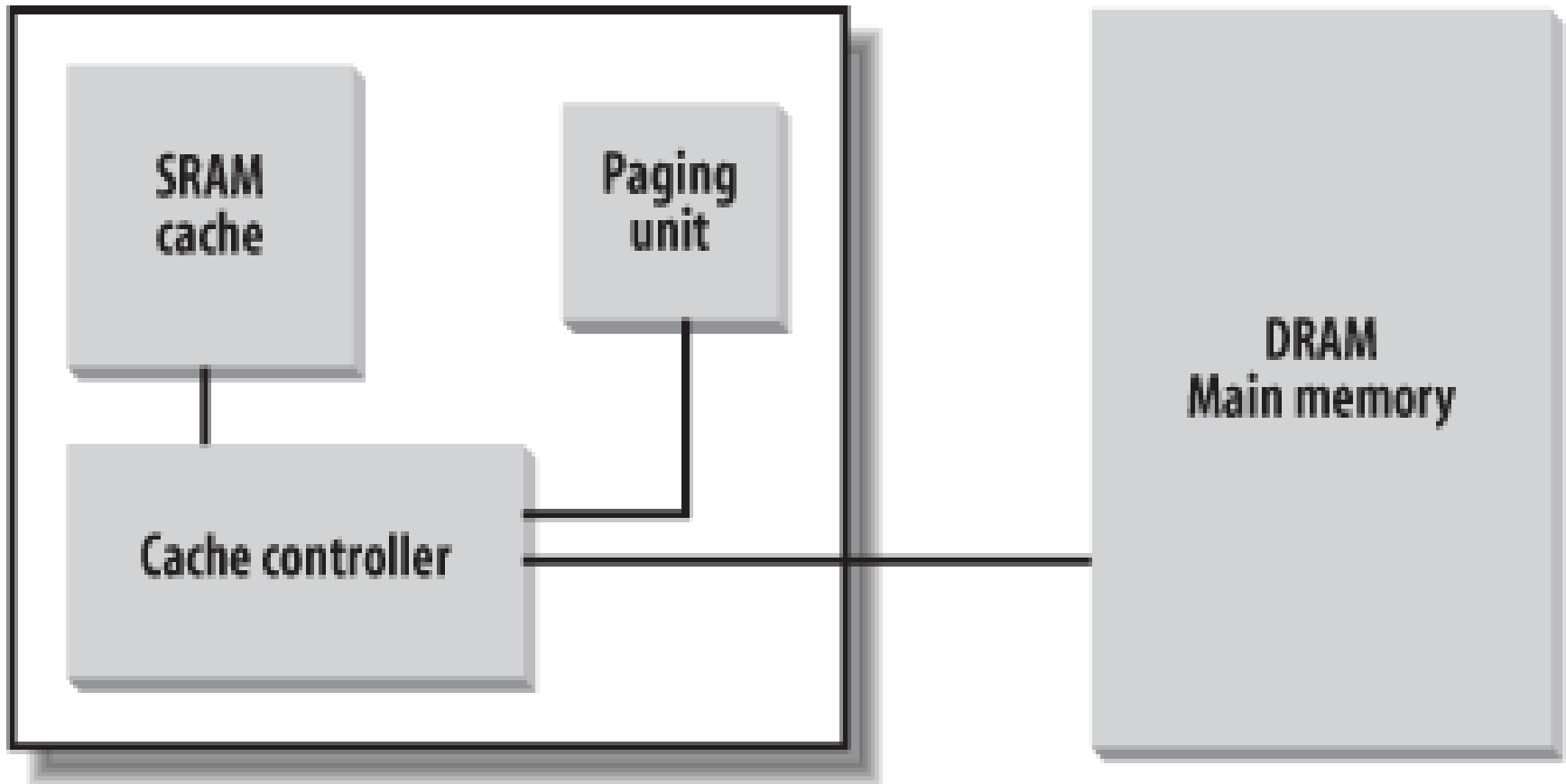
- As of the Pentium Pro (and including Pentium II, III and IV) the physical address space of the processor has been extended from 32 bits (4GB) to 36 bits (64GB)
- This feature requires a new page table organization
- Page Table Entries (PTEs) have been extended from 32 bits to 64 bits, reducing the entries per 4KB page from 1024 to 512

# PAE

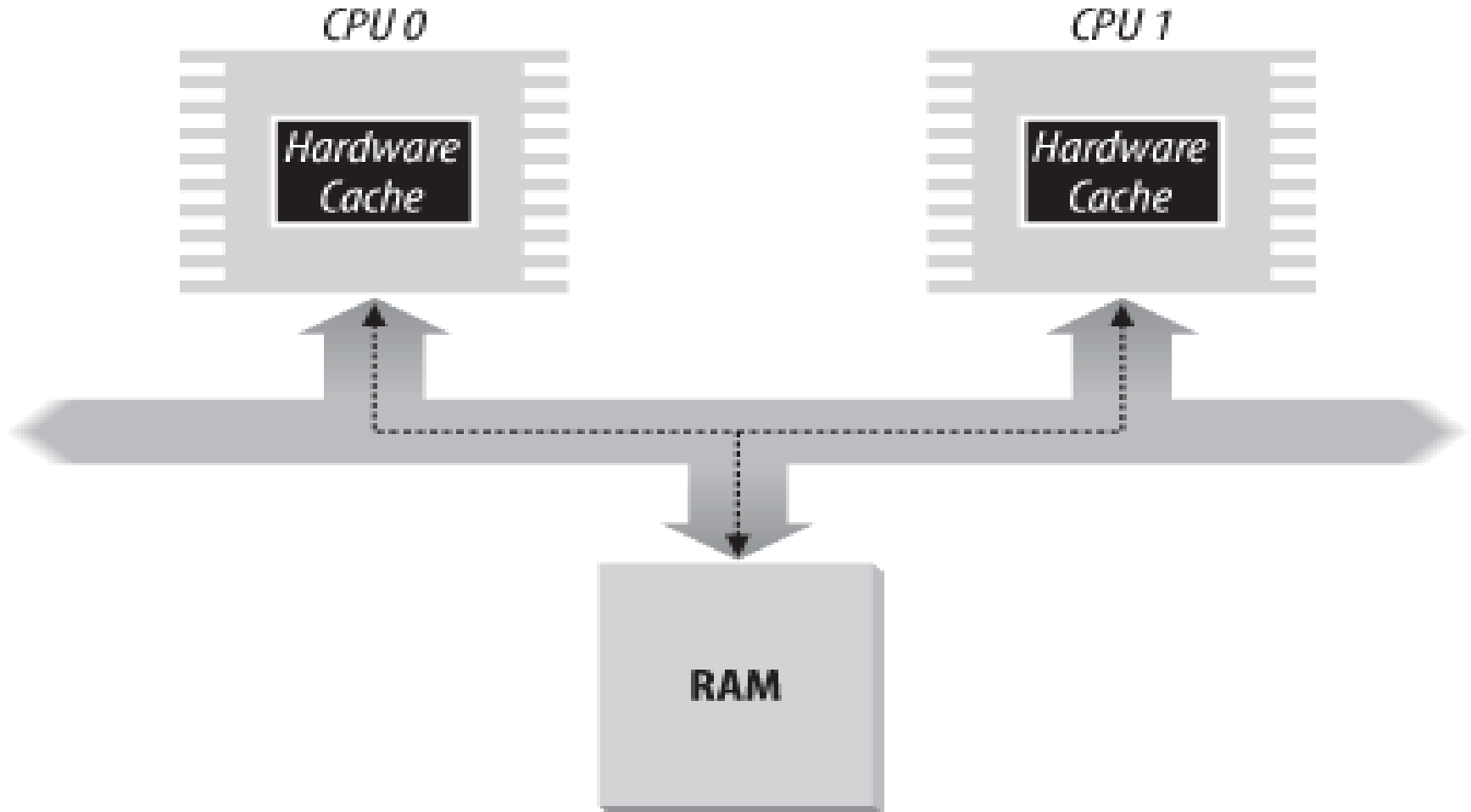
- If the PAE flag is set in CR4, the CR3 register now points to a new table (the Page Directory Pointer Table) that includes 4 - 64 bit PTEs, each pointing to a Directory Table
  - A 32 bit address now uses bits 30-31 to select the correct Directory Table
  - Bits 21-29 gets one of 512 Directory entries
  - Bits 12-20 get one of 512 Table entries
  - Bits 0-11 provide a 4KB offset
  - Extended pages are now 2MB (instead of 4MB)

# Hardware Cache

CPU



# Cache Coherence



# Page Table Entries (PTEs)

- For non PAE systems, a PTE includes 20 bits to reference one of a possible 1M page frames (4GB ISA space)
- The TLB uses the remaining 12 bits of the 32 bit entry for page properties
- The *present* bit indicates if the 20 bit target address is valid, or currently unmapped

# PTE Control Bits

- Present flag
  - If it is set, the referred-to page (or Page Table) is contained in main memory.
- Accessed flag
  - Set each time the paging unit addresses the corresponding page frame.
- Dirty flag
  - Applies only to the Page Table entries.
- Read/Write flag
  - Contains the access right (Read/Write or Read) of the page or of the Page Table
- User/Supervisor flag
  - Contains the privilege level required to access the page or Page Table
- PCD and PWT flags
  - Controls the way the page or Page Table is handled by the hardware cache
- Page Size flag
  - Applies only to Page Directory entries.
- Global flag
  - Applies only to Page Table entries.

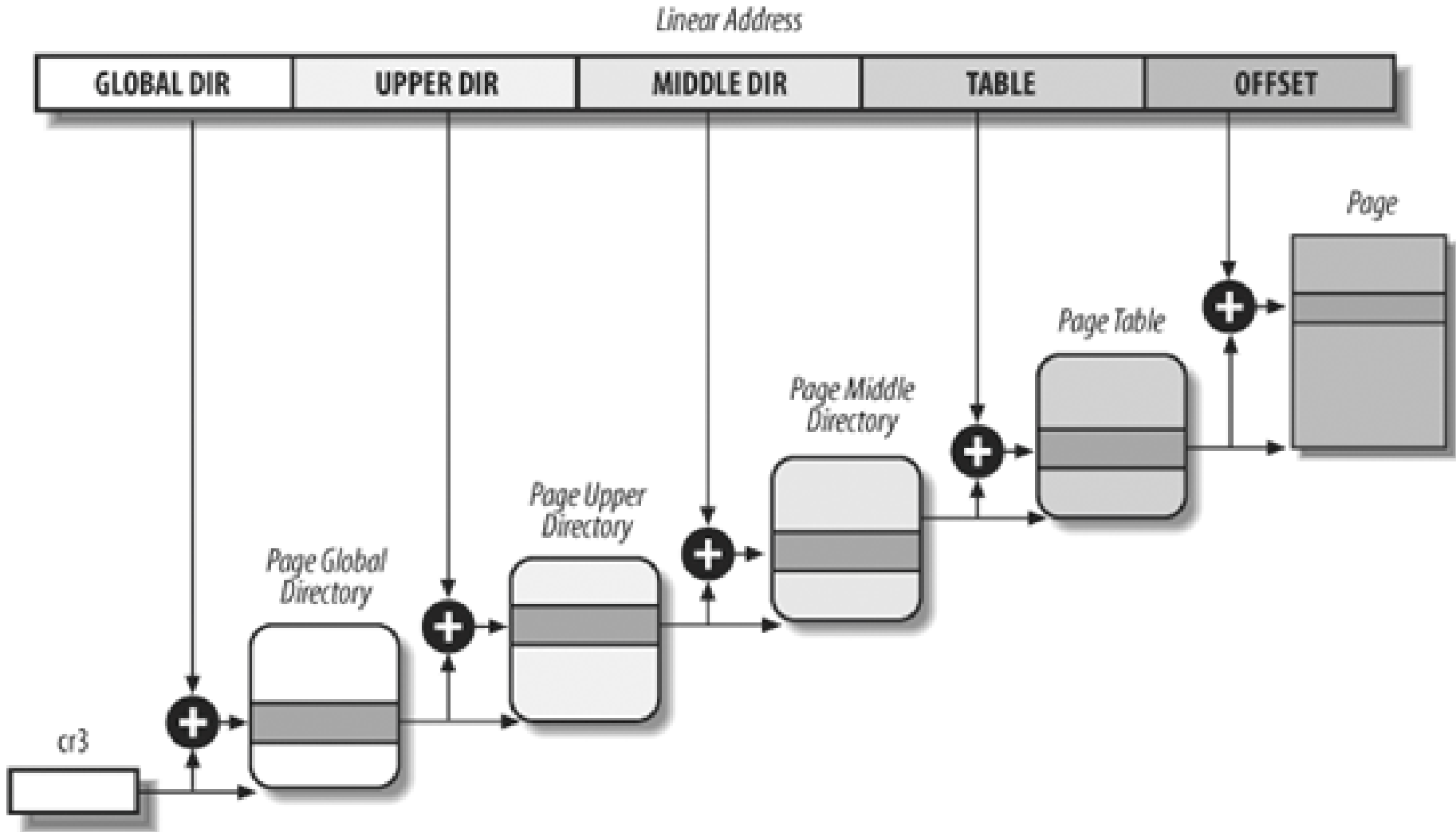
# Translation Lookaside Buffers TLBs

- The Memory Management Unit maintains a set of buffers used to capture recently translated virtual addresses (during table walks)
- TLB hit rates are critical for good performance
- When the cr3 register is reloaded (on a heavyweight context switch) the TLB cache is invalidated

# Paging in Linux

- Linux has attempted to accommodate both 32 bit and 64 bit systems, and so has used a paging model with more than 2 levels
- The model up to version 2.6.10 used 3 table levels
- The model from 2.6.11 forward now uses a 4 table model

# Linux 4 Table Model



# Page Table Handling

- **pte\_t**, **pmd\_t**, **pud\_t**, and **pgd\_t** describe the format of, respectively, a Page Table, a Page Middle Directory, a Page Upper Directory, and a Page Global Directory entry.
- They are 64-bit data types when PAE is enabled and 32-bit data types otherwise.
- **pgprot\_t** is another 64-bit (PAE enabled) or 32-bit (PAE disabled) data type that represents the protection flags associated with a single entry.

# Support to Read or Modify PTEs

- The kernel also provides several macros and functions to read or modify page table entries:
- **pte\_none**, **pmd\_none**, **pud\_none**, and **pgd\_none** yield the value 1 if the corresponding entry has the value 0; otherwise, they yield the value 0.
- **pte\_clear**, **pmd\_clear**, **pud\_clear**, and **pgd\_clear** clear an entry of the corresponding page table, thus forbidding a process to use the linear addresses mapped by the page table entry. The **ptep\_get\_and\_clear( )** function clears a Page Table entry and returns the previous value.
- **set\_pte**, **set\_pmd**, **set\_pud**, and **set\_pgd** write a given value into a page table entry; **set\_pte\_atomic** is identical to **set\_pte**, but when PAE is enabled it also ensures that the 64-bit value is written atomically.
- **pte\_same(a,b)** returns 1 if two Page Table entries a and b refer to the same page and specify the same access privileges, 0 otherwise.
- **pmd\_large(e)** returns 1 if the Page Middle Directory entry  $e_{25}$  refers to a large page (2 MB or 4 MB), 0 otherwise.

# Page flag reading functions

<b>Function name</b>	<b>Description</b>
<code>pte_user( )</code>	Reads the User/Supervisor flag
<code>pte_read( )</code>	Reads the User/Supervisor flag (pages on the 80 x 86 processor cannot be protected against reading)
<code>pte_write( )</code>	Reads the Read/Write flag
<code>pte_exec( )</code>	Reads the User/Supervisor flag (pages on the 80 x 86 processor cannot be protected against code execution)
<code>pte_dirty( )</code>	Reads the Dirty flag
<code>pte_young( )</code>	Reads the Accessed flag
<code>pte_file( )</code>	Reads the Dirty flag (when the Present flag is cleared and the Dirty flag is set, the page belongs to a non-linear disk file mapping ... see: <code>remap_file_pages()</code> )

# Additional VM Support

- The kernel supplies several other functions and macros for manipulating tables and PTEs
  - Page flag setting functions such as `pte_wrprotect( )`
  - Macros acting on Page Table entries such as `mk_pte(p,prot)`
  - Page allocation functions such as `pte_free(pte)`
  - See Tables 2.5 through 2.8 in the text

# Physical Memory Layout

- During system boot and initialization the kernel builds a physical address map
- For 80x86 processors the kernel considers each 4KB frame as either available or reserved
  - Frames that do not correspond to RAM are reserved
  - Frames that contain kernel code and initialized data structures are reserved
- Reserved frames are not pagable (wired)

# PM Layout (cont'd)

- As a general rule, the Linux kernel is installed in RAM starting from the physical address 0x00100000 (just past 1MB)
- Page frame 0 is used by BIOS to store the system hardware configuration detected during the *Power-On Self-Test(POST)*
- Physical addresses ranging from 0x000a0000 to 0x000fffff are usually reserved to BIOS (640KB -1MB hole)

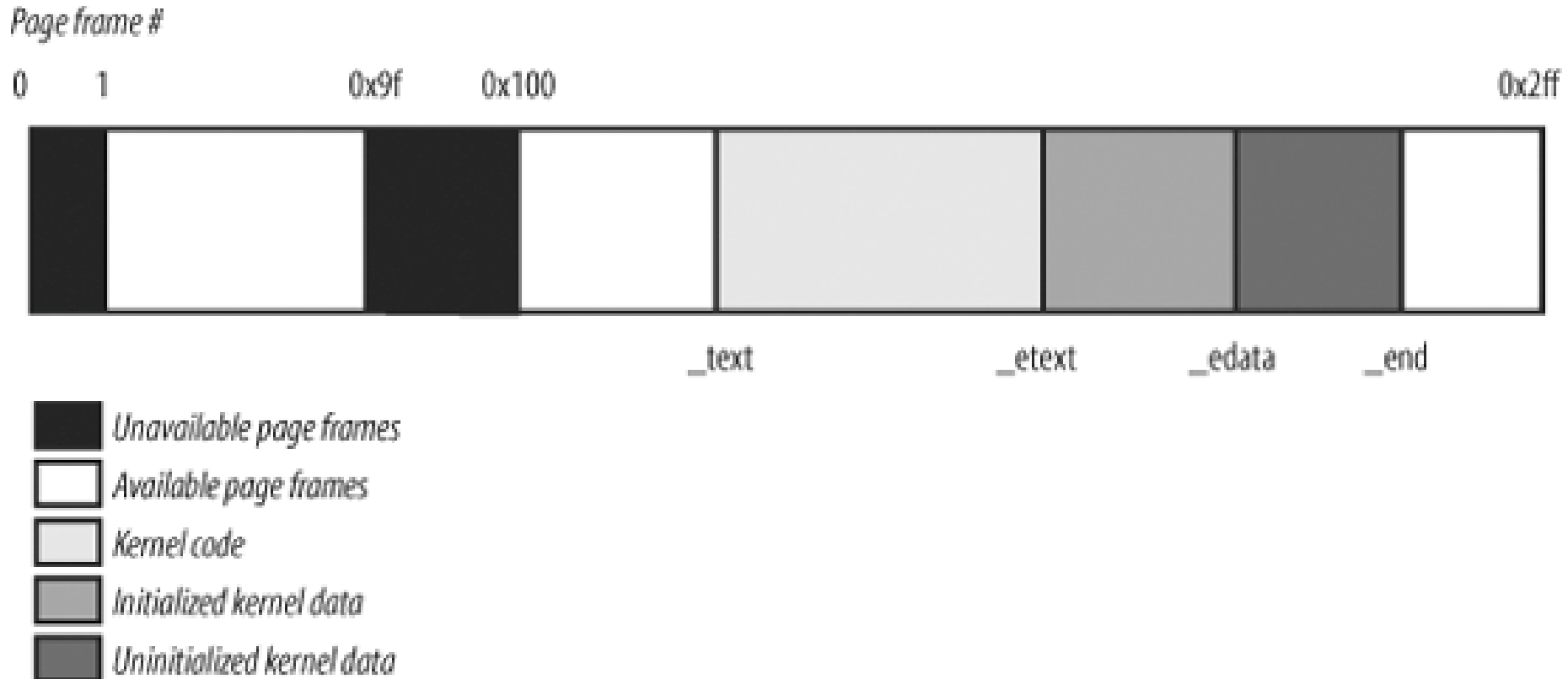
# Example of BIOS-provided PM

<b>Start</b>	<b>End</b>	<b>Type</b>
0x00000000	0x0009ffff	Usable
0x000f0000	0x000ffffff	Reserved
0x00100000	0x07feffff	Usable
0x07ff0000	0x07ff2fff	ACPI data
0x07ff3000	0x07fffffff	ACPI NVS
0xfffff0000	0xffffffff	Reserved

# Kernel's PM Variables

<b>Variable name</b>	<b>Description</b>
num_physpage	Page frame number of the highest usable page frame
totalram_pages	Total number of usable page frames
min_low_pfn	Page frame number of the first usable page frame after the kernel image in RAM
max_pfn	Page frame number of the last usable page frame
max_low_pfn	Page frame number of the last page frame directly mapped by the kernel (low memory)
totalhigh_pages	Total number of page frames not directly mapped by the kernel (high memory)
highstart_pfn	Page frame number of the first page frame not directly mapped by the kernel
highend_pfn	Page frame number of the last page frame not directly mapped by the kernel

# The first 768 page frames (3 MB)



# Process Page Tables

- The linear address space of a process is divided into two parts
  - Linear addresses from 0x00000000 to 0xbfffffff can be addressed when the process runs in either User or Kernel Mode.
  - Linear addresses from 0xc0000000 to 0xffffffff can be addressed only when the process runs in Kernel Mode.
- This represents the 3GB user 1GB kernel model

# Kernel Page Tables

- The kernel must initialize its own page table as a reference for system processes
- The kernel's page table will map the deepest 1GB of virtual space that all processes will share
- The kernel will also map all of physical memory if RAM is  $< 896$  pages
- A 128MB region of the KVM is left unmapped for fine-grained dynamic alloc

# TLB Management

- The 80x86 processors carry out invalidation operations on the non-global TLB entries whenever the CR3 register is modified
  - The kernel avoids this flush when a context switch between two processes sharing the same page tables occurs
  - The kernel also avoids this when switching to a kernel thread

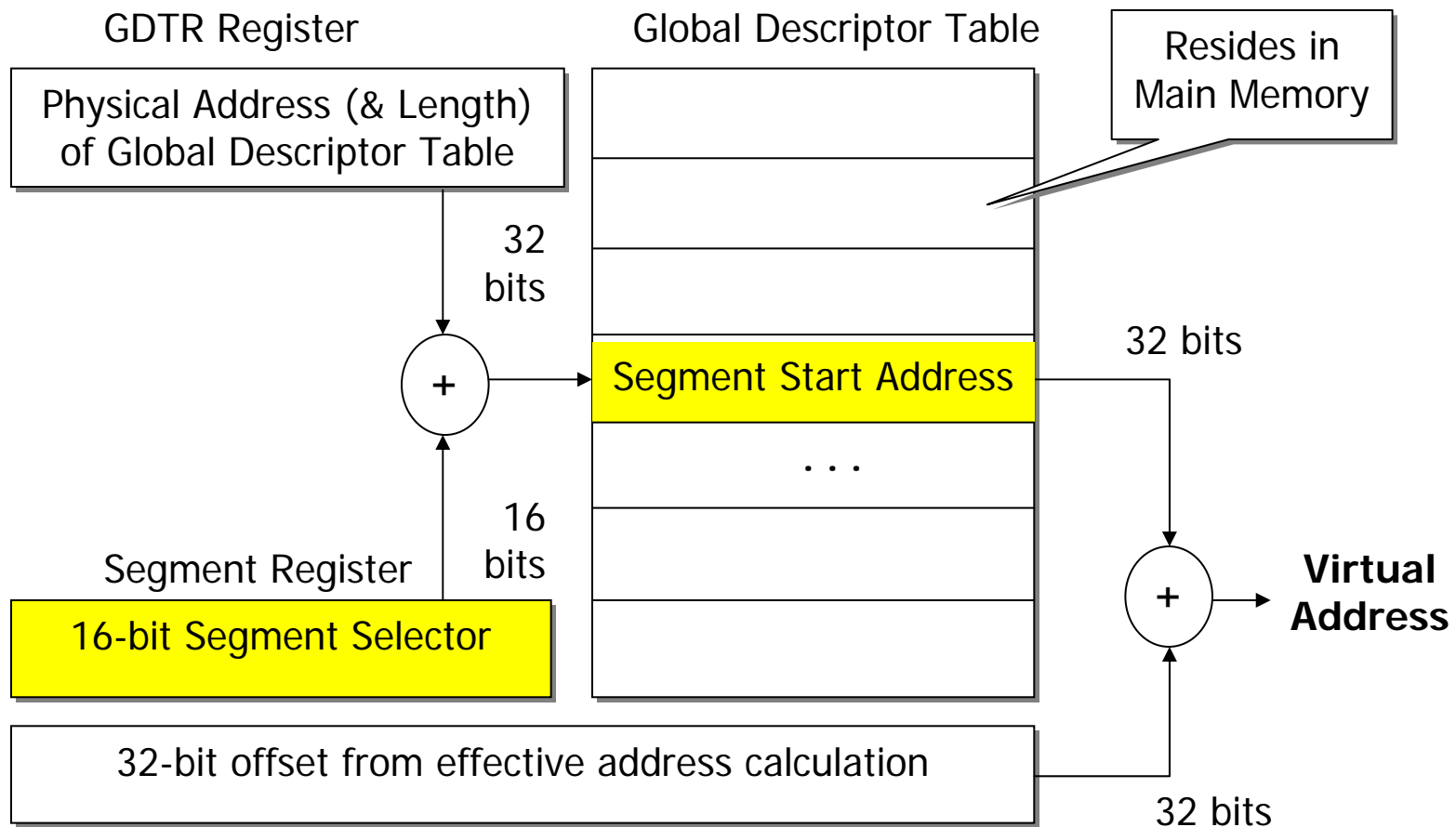
# Memory Management Summary

- The Intel 80x86 architecture uses segmentation as an integral part of address translation
- An assembly instruction like:  
    MOV EAX,[x]  
    implicitly depends on the DS selector (base) to locate the linear address of the operand x (offset)
- Fetching this instruction required the CS selector (base) and the IP register (offset) to build the linear address of the instruction

# Virtual Memory

- If CR 0 has the 0 bit on (protected mode) and the 31 bit on (paging mode), then linear addresses are not considered to be physical, but virtual
- These virtual addresses require further translation with the help of Directory and Page tables to become physical
- Linux must manage these tables on a per process basis

# How Segment Registers are Used



# 80x86 System Level Registers

