OPERATING SYSTEMS CS3500

PROF. SUKHENDU DAS DEPTT. OF COMPUTER SCIENCE AND ENGG., IIT MADRAS, CHENNAI – 600036.

Email: sdas@cse.iitm.ac.in
URL: http://www.cse.iitm.ac.in/~vplab/os.html

Oct . – 2022.

MAIN MEMORY - II

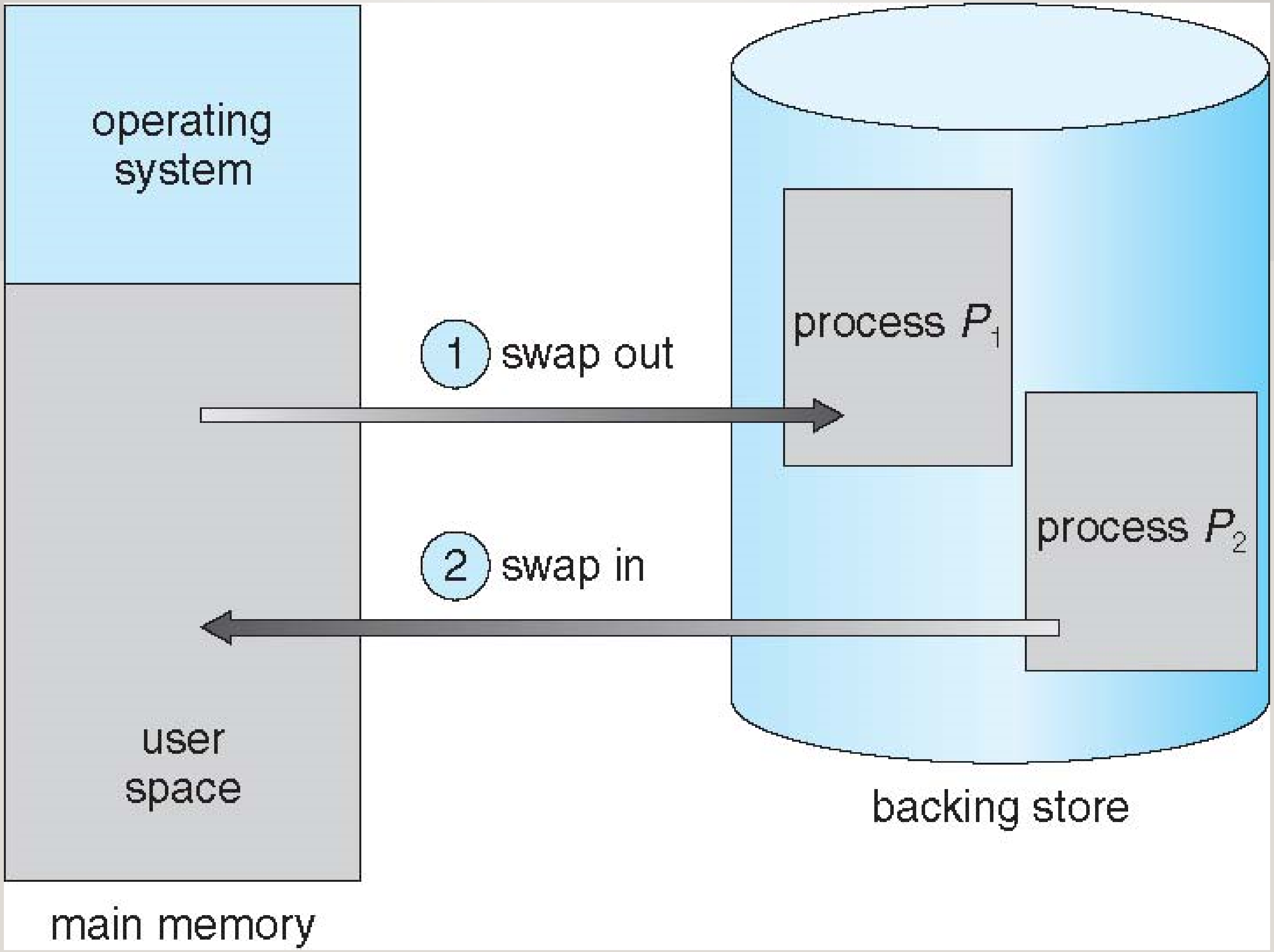
OUTLINE

- Swapping
- Example: The Intel 32 Architecture
- Example: 64-bit Architecture
- Example: The Intel IA-32 Architecture
- Example: ARMv8 Architecture

SWAPPING (CONT.)

- Does the swapped out process need to swap back in to same physical addresses?
- Depends on address binding method
  - Plus consider pending I/O to/from process memory space
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - Swapping normally disabled
  - Started if more than threshold amount of memory allocated
  - Disabled again once memory demand reduced below threshold

SCHEMATIC VIEW OF SWAPPING



CONTEXT SWITCH TIME INCLUDING SWAPPING

- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process
- Context switch time can then be very high
- 100MB process swapping to hard disk with transfer rate of 50MB/sec
  - Swap out time of [redacted] ms
  - Plus swap in of same sized process
  - Total context switch swapping component time of [redacted] seconds)
- Can reduce if reduce size of memory swapped – by knowing how much memory really being used
  - System calls to inform OS of memory use via `request_memory()` and `release_memory()`

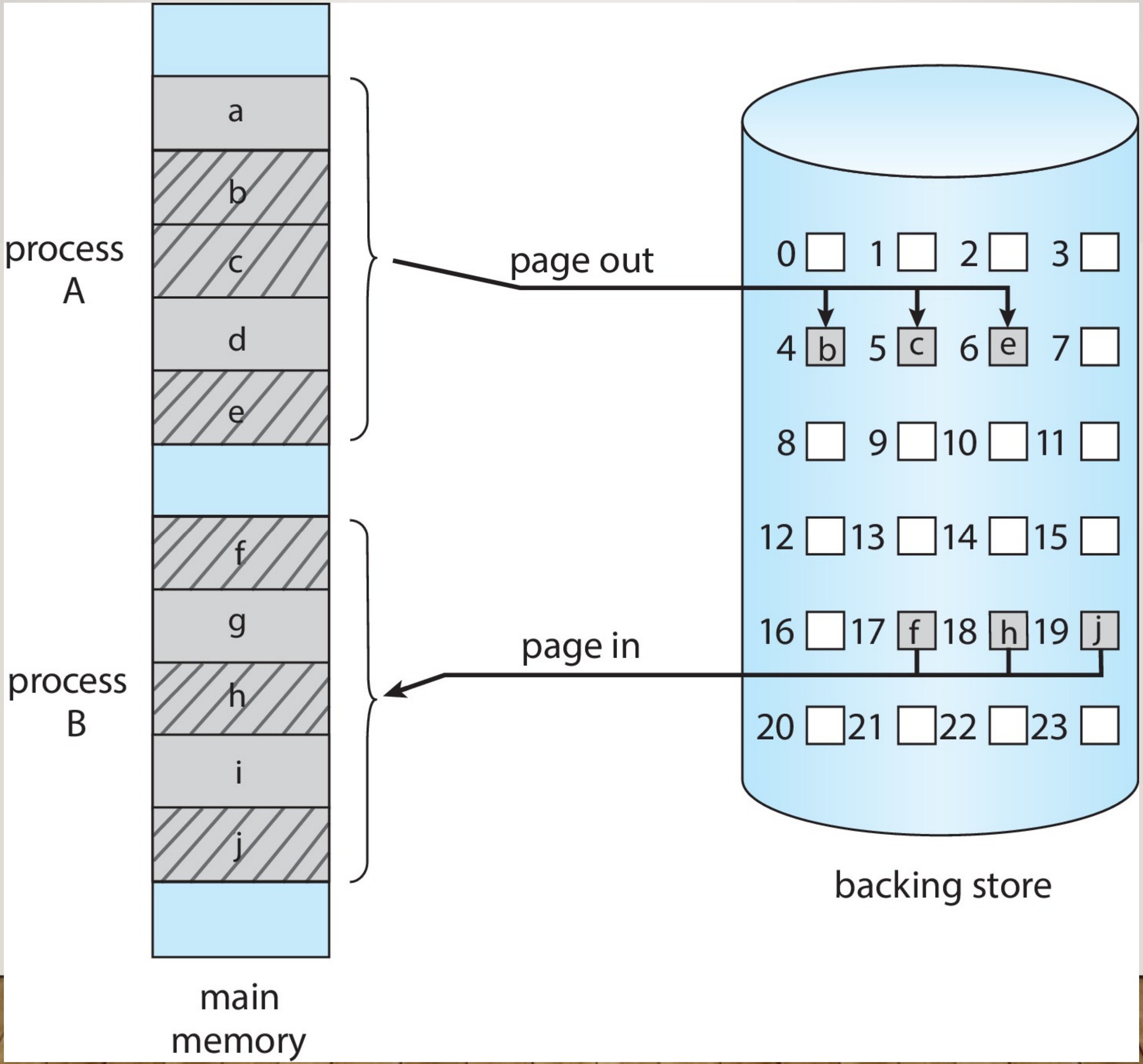
CONTEXT SWITCH TIME AND SWAPPING (CONT.)

- Other constraints as well on swapping
  - Pending I/O – can't swap out as I/O would occur to wrong process
  - Or always transfer I/O to kernel space, then to I/O device
    - Known as **double buffering**, adds overhead
- Standard swapping not used in modern operating systems
  - But modified version common
    - Swap only when free memory extremely low

SWAPPING ON MOBILE SYSTEMS

- Not typically supported
  - Flash memory based
    - Small amount of space
    - Limited number of write cycles
    - Poor throughput between flash memory and CPU on mobile platform
- Instead use other methods to free memory if low
  - iOS *asks* apps to voluntarily relinquish allocated memory
    - Read-only data thrown out and reloaded from flash if needed
    - Failure to free can result in termination
  - Android terminates apps if low free memory, but first writes application state to flash for fast restart

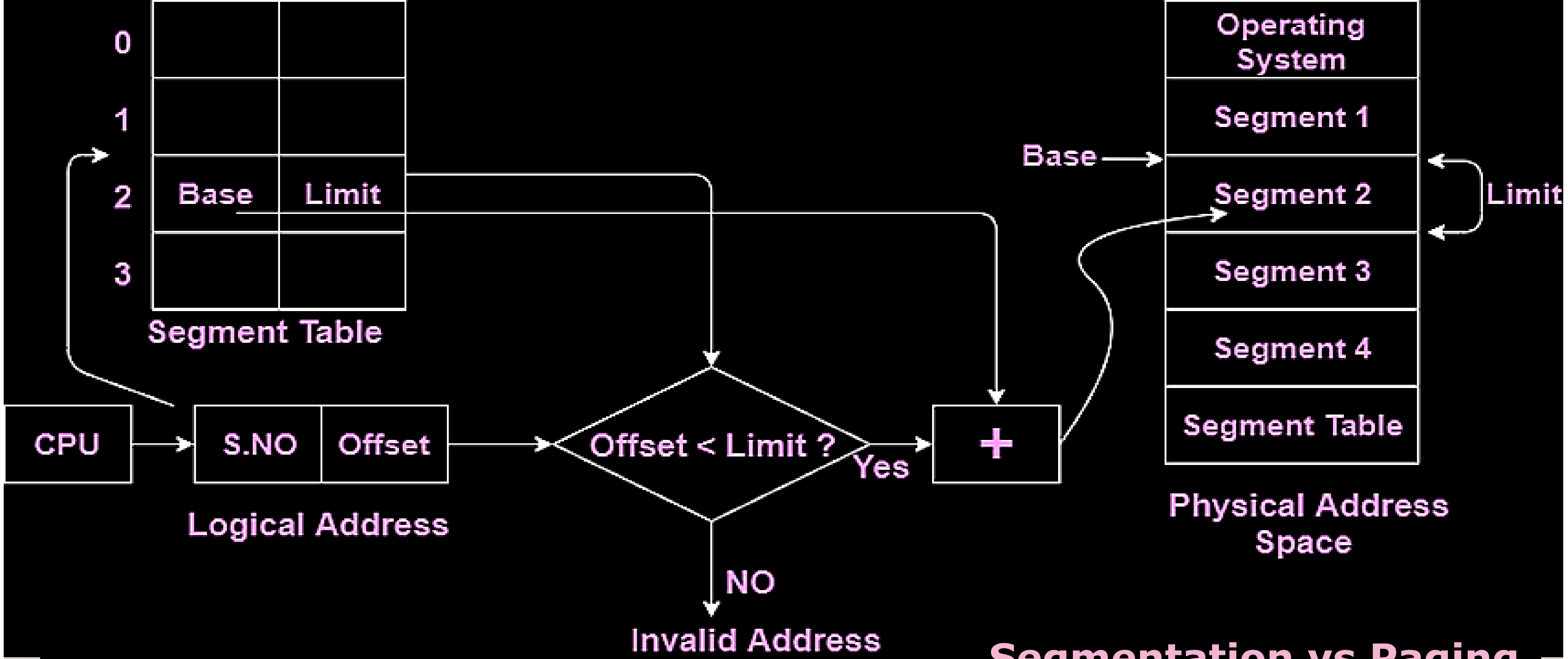
SWAPPING WITH PAGING



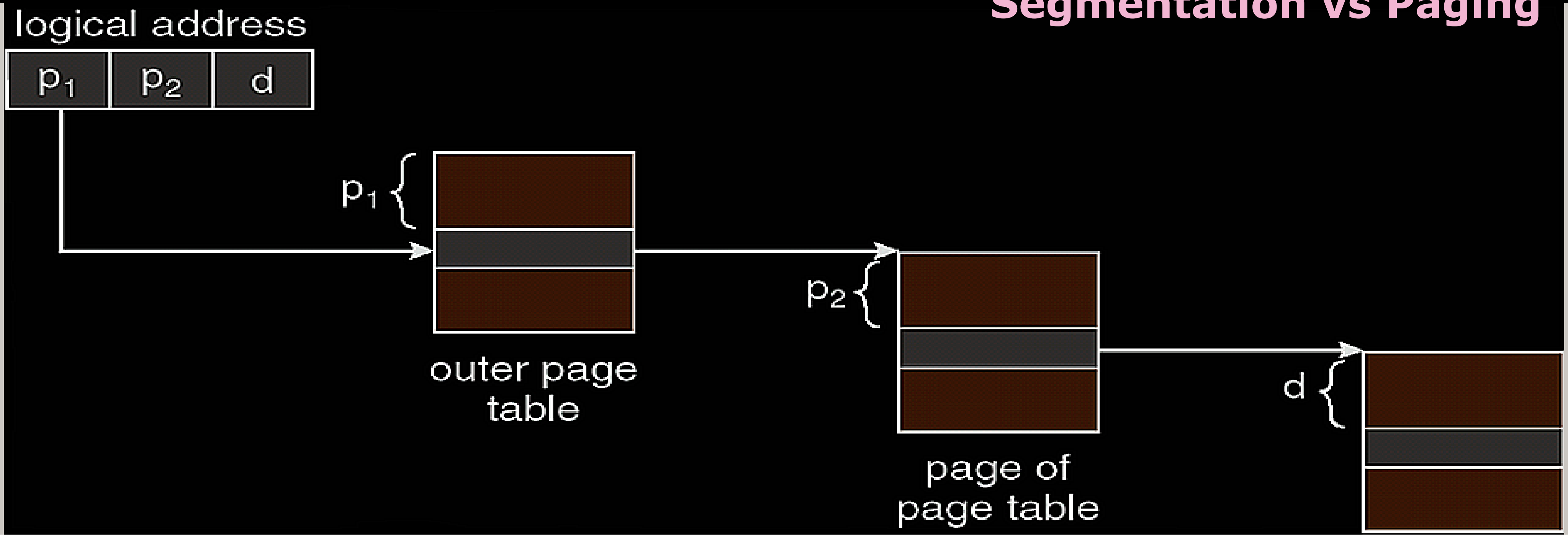
EXAMPLE: THE INTEL 32 AND 64-BIT ARCHITECTURES

- Dominant industry chips
- Pentium CPUs are 32-bit and called IA-32 architecture
- Current Intel CPUs are 64-bit and called IA-64 architecture
- Many variations in the chips, cover the main ideas here

IA-32 has two components— ***segmentation and paging***—and works as follows: The CPU generates *logical addresses*, which are given to the *segmentation unit*. The segmentation unit produces a *linear address* for each logical address. The linear address is then given to the *paging unit*, which in turn *generates the physical address* in main memory. Thus, the segmentation and paging units form the equivalent of the memory-management unit (MMU).



Segmentation vs Paging



EXAMPLE: THE INTEL IA-32 ARCHITECTURE

- Supports both segmentation and segmentation with paging
  - Each segment can be 4 GB
  - Up to 16 K segments per process
  - For each process, LA is divided into two partitions:
    - First partition of up to 8 K segments are private to process (kept in **local descriptor table** (**LDT**))
    - Second partition of up to 8K segments shared among all processes (kept in **global descriptor table** (**GDT**)

LDT/GDT entry: 8-byte segment descriptor with detailed information about a particular segment, including the base location and limit

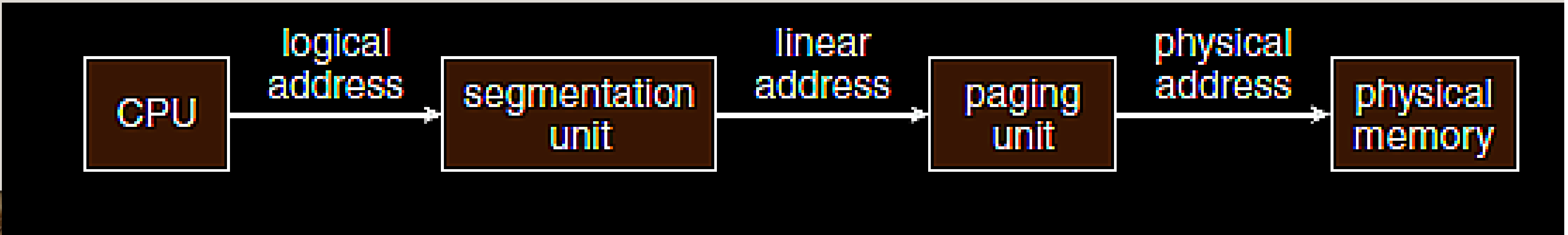
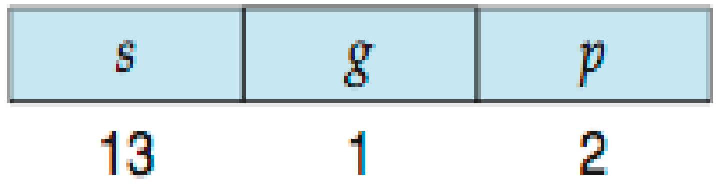


Figure 9.21 Logical to physical address translation in IA-32.

EXAMPLE: THE INTEL IA-32 ARCHITECTURE (CONT.)

- CPU generates logical address - a pair (selector, offset)
  - Selector (16-bit) given to segmentation unit
    - Which produces linear addresses

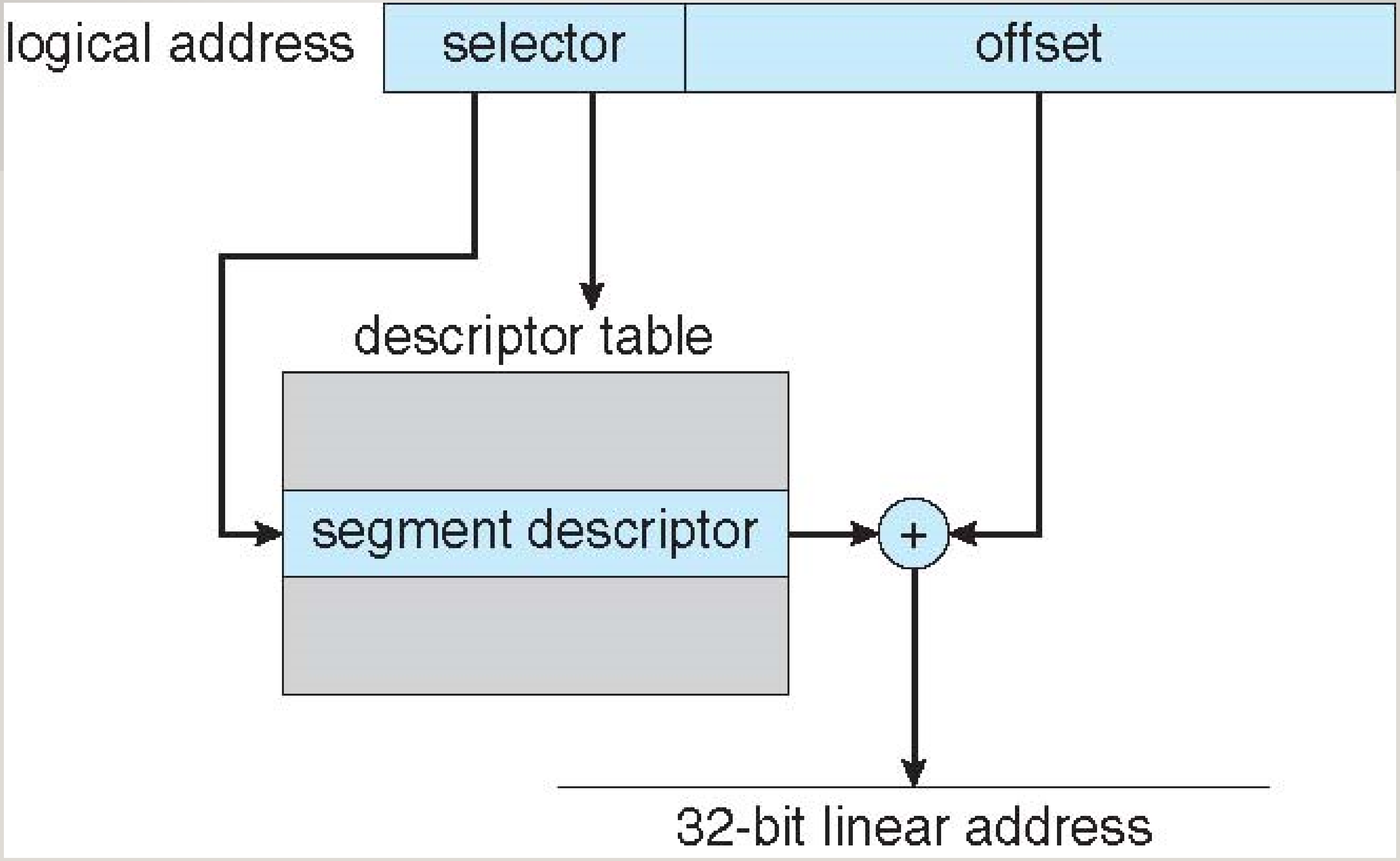


    - *S* designates the segment number, *g* indicates whether the segment is in the GDT or LDT, and ***p*** deals with protection. The offset is a 32-bit number specifying the location of the byte within the segment

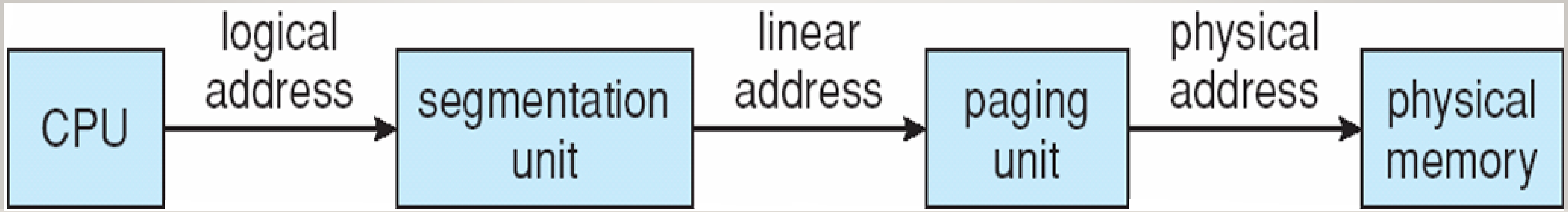
Machine has six segment registers, allowing six segments to be addressed at any one time by a process. It also has six 8-byte microprogram registers

  - Linear address (32-bit) given to paging unit
    - Which generates physical address in main memory
    - Paging units form equivalent of MMU
    - Pages sizes can be 4 KB or 4 MB

INTEL IA-32 SEGMENTATION

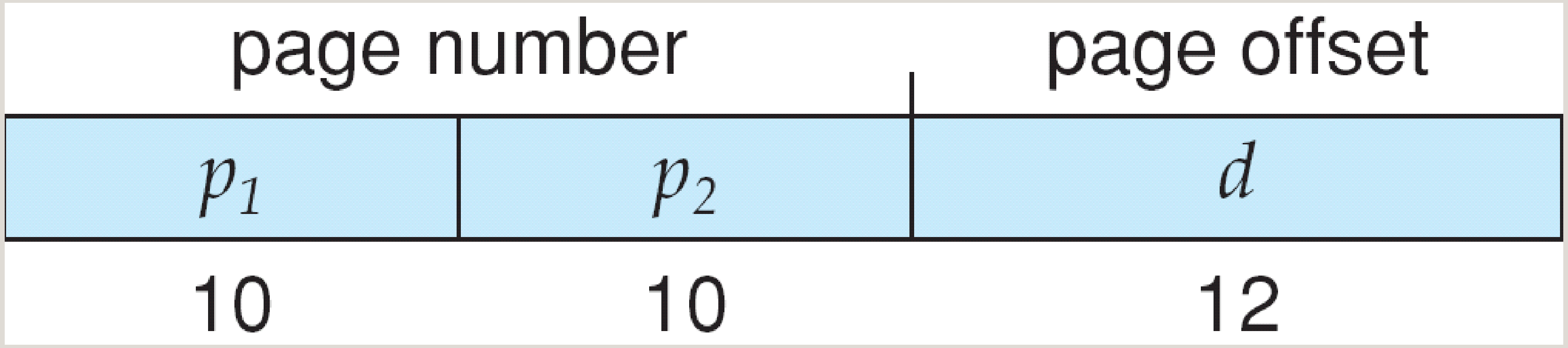


LOGICAL TO PHYSICAL ADDRESS TRANSLATION IN IA-32

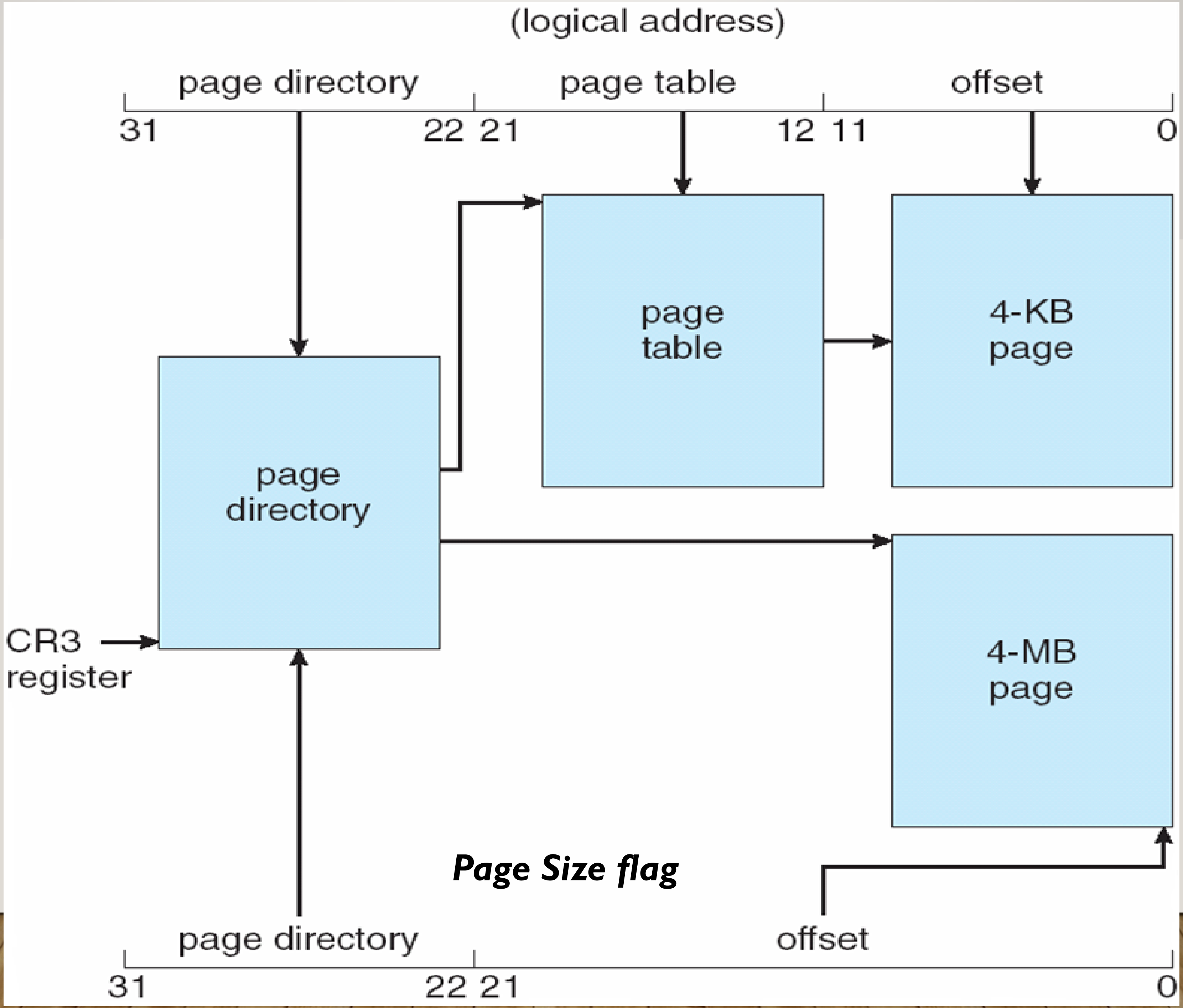


The IA-32 architecture allows a page size of either 4 KB **or** 4 MB.

For 4-KB pages, IA-32 uses a two-level paging scheme in which the division of the 32-bit linear address is as follows:



INTEL IA-32 PAGING ARCHITECTURE



INTEL IA-32 PAGE ADDRESS EXTENSIONS

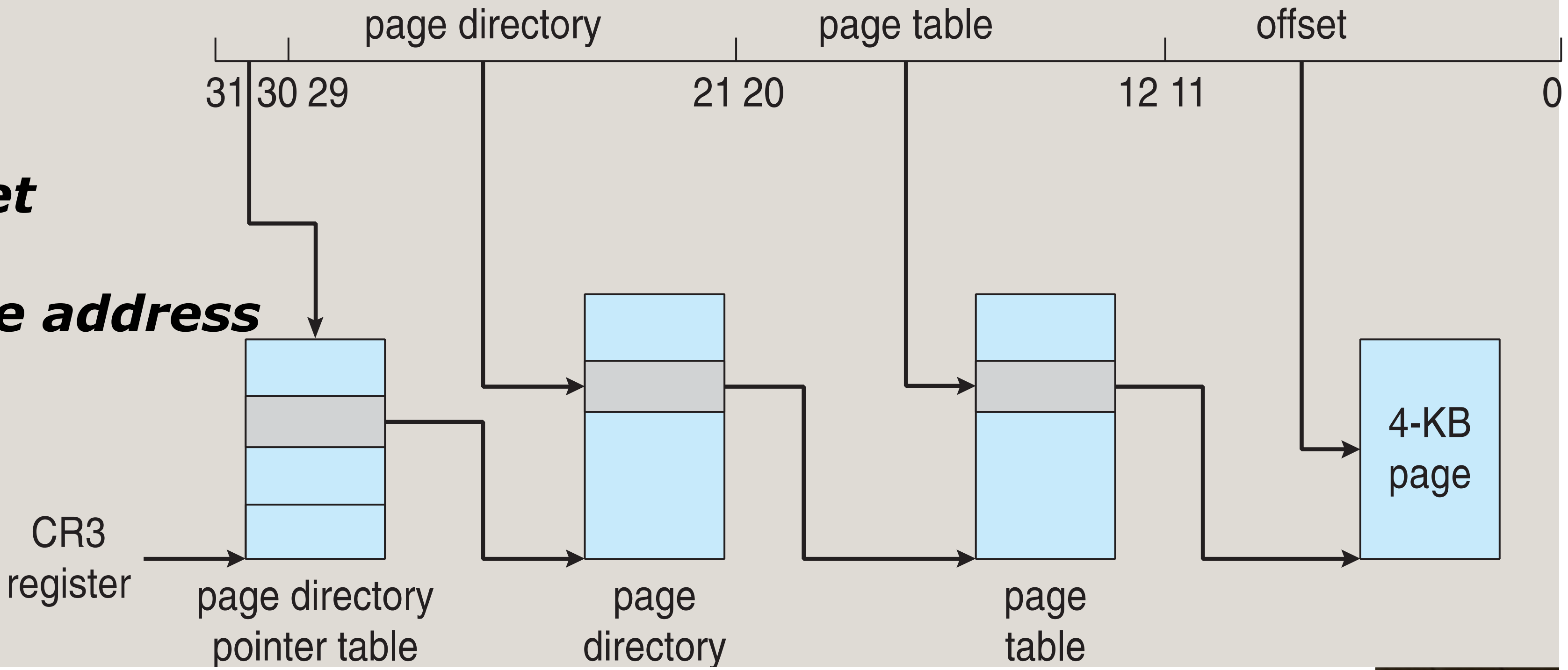
- 32-bit address limits led Intel to create **page address extension** (**PAE**), allowing 32-bit apps access to more than 4GB of memory space
  - Paging went to a 3-level scheme
  - Top two bits refer to a **page directory pointer table**
  - Page-directory and page-table entries moved to 64-bits in size – more bits for physical page address, or "page frame number" field,
  - Net effect is increasing address space to 36 bits – 64GB of physical memory

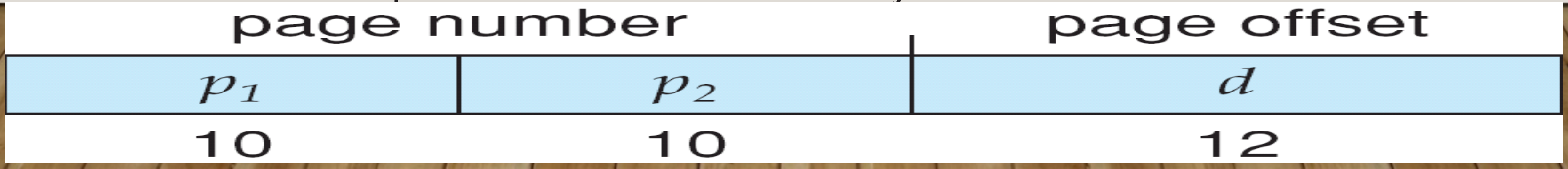
How 36 ?

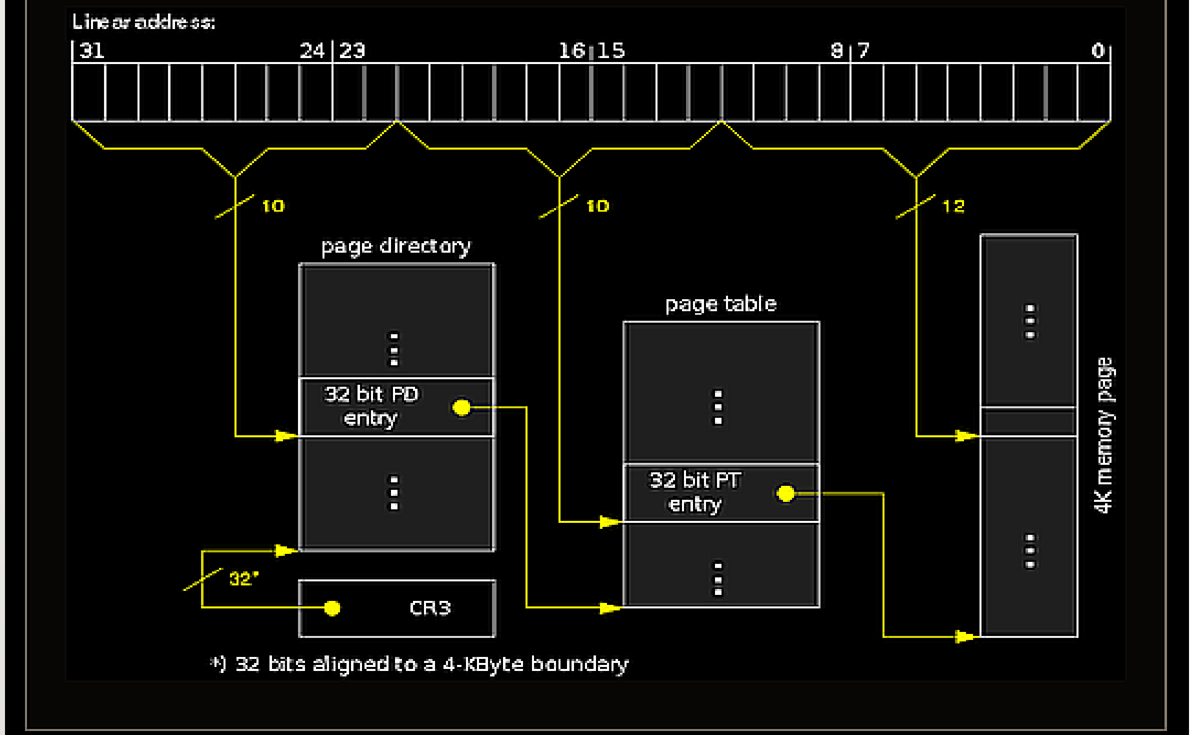
12-bit offset

+

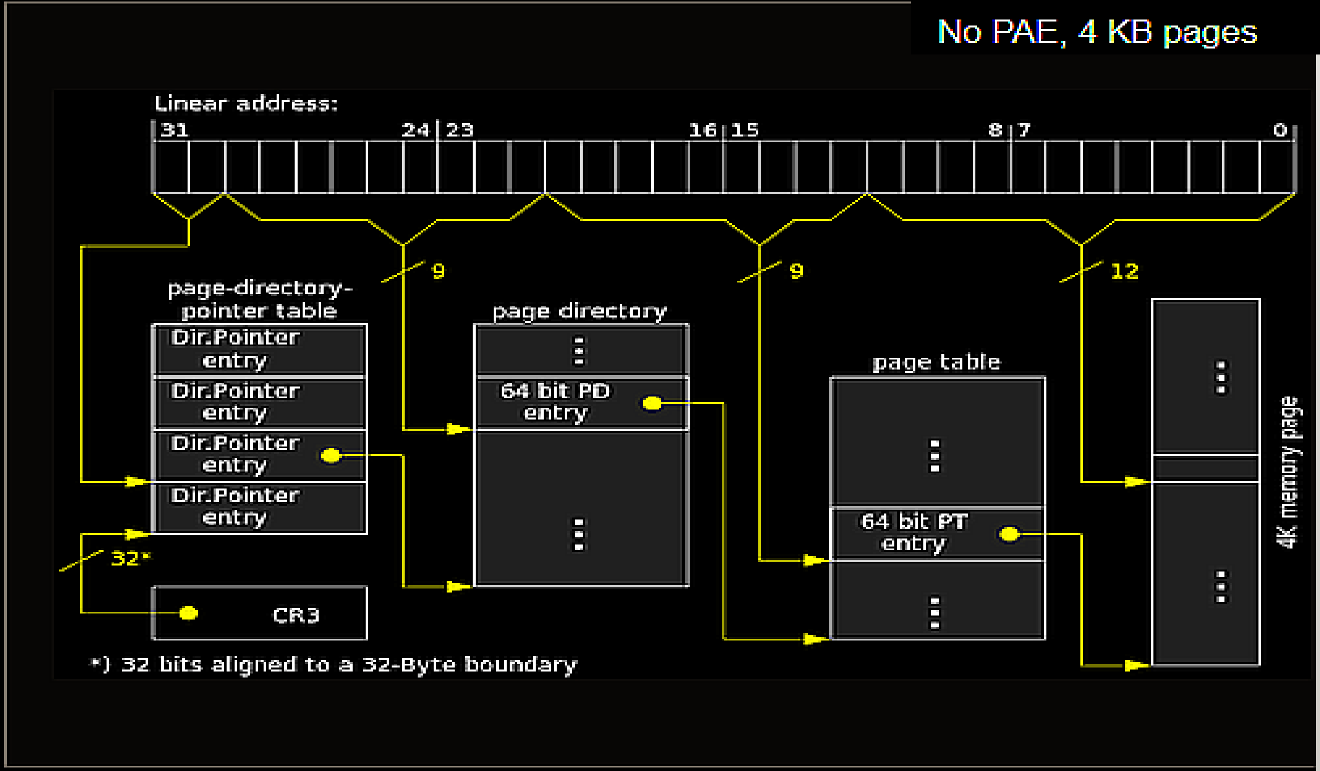
(*) 24-bit base address of page table/frames



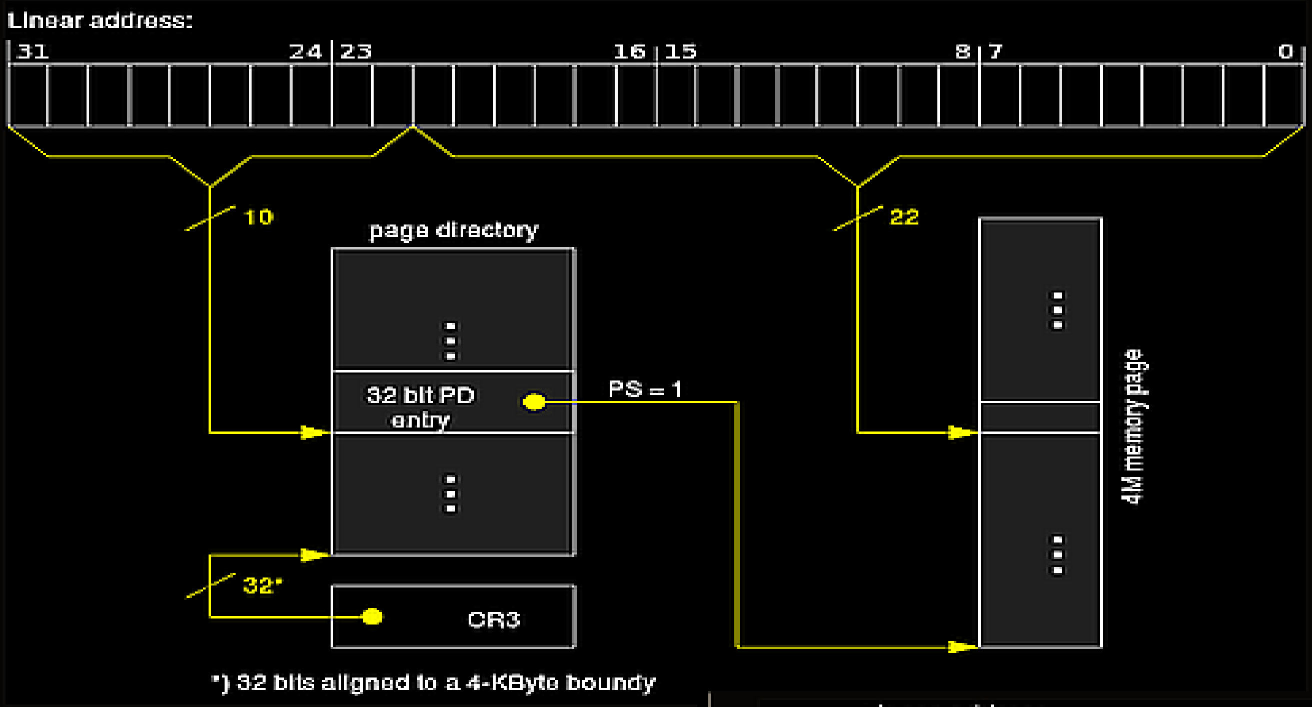




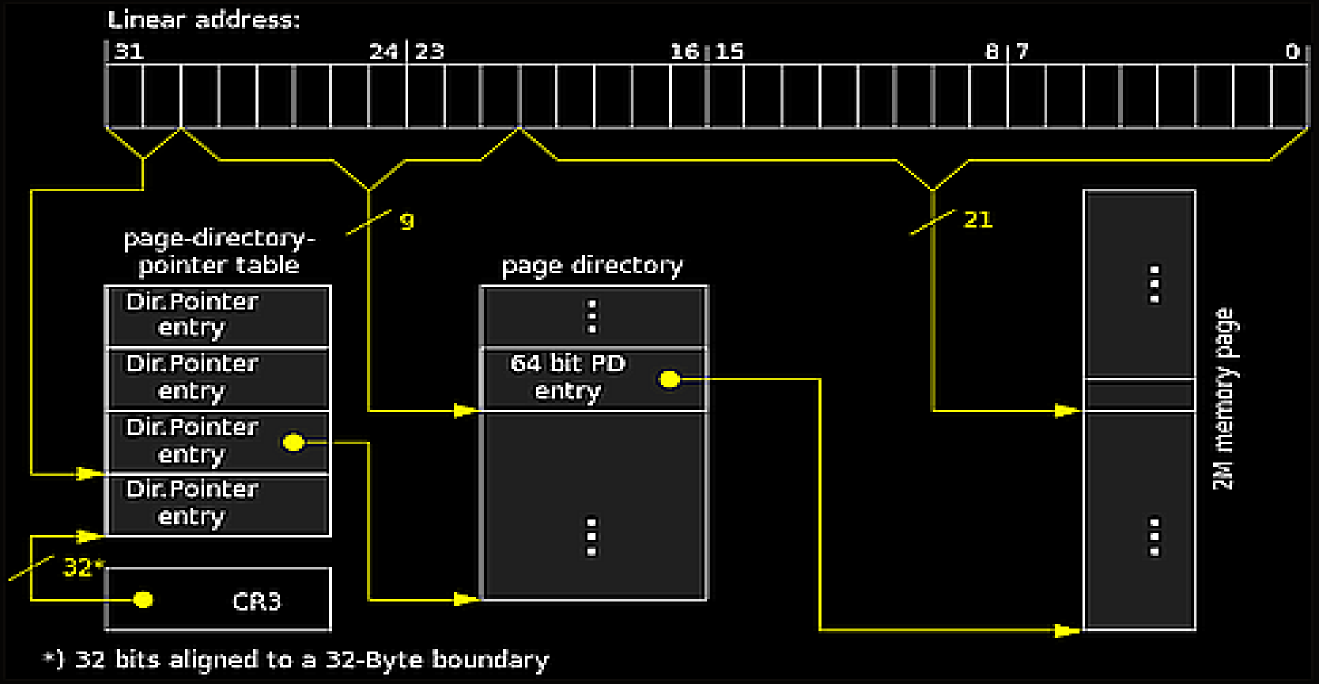
No PAE, 4 KB pages



With PAE; 4 KB pages



No PAE, 4 MB pages



With PAE; 2 MB pages

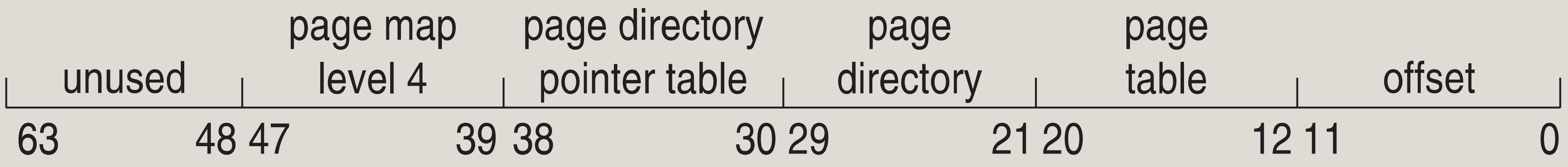
In all page table formats supported by x86 and x86-64, the 12 least significant bits of the page table entry are either interpreted by the memory management unit or are reserved for operating system use. In processors that implement the "no-execute" or "execution disable" feature, the most significant bit (**bit 63**) is the NX bit. The next eleven most significant bits (**bits 52 through 62**) are reserved for operating system use by both Intel and AMD's architecture specifications. Thus, from 64 bits in the page table entry, 12 low-order and 12 high-order bits have other uses, leaving **40 bits (bits 12 though 51**) for the physical page number. Combined with **12 bits of "offset within page**" from the linear address, a maximum **of 52 bits** are available to address physical memory. This allows a maximum RAM configuration of 2^{52} bytes, or **4 petabytes (about 4.5×10^{15} bytes).**

On x86-64 processors in native long mode, the address translation scheme uses PAE but adds a fourth table, the **512-entry *page-map*** *level 4* table, and extends the page directory pointer table to 512 entries instead of the original 4 entries it has in protected mode. Currently **48 bits of virtual page number** are translated, giving a **virtual address space of up to 256 TB**. In the page table entries, in the original specification, 40 bits of physical page number are implemented.

https://en.wikipedia.org/wiki/Physical_Address_Extension

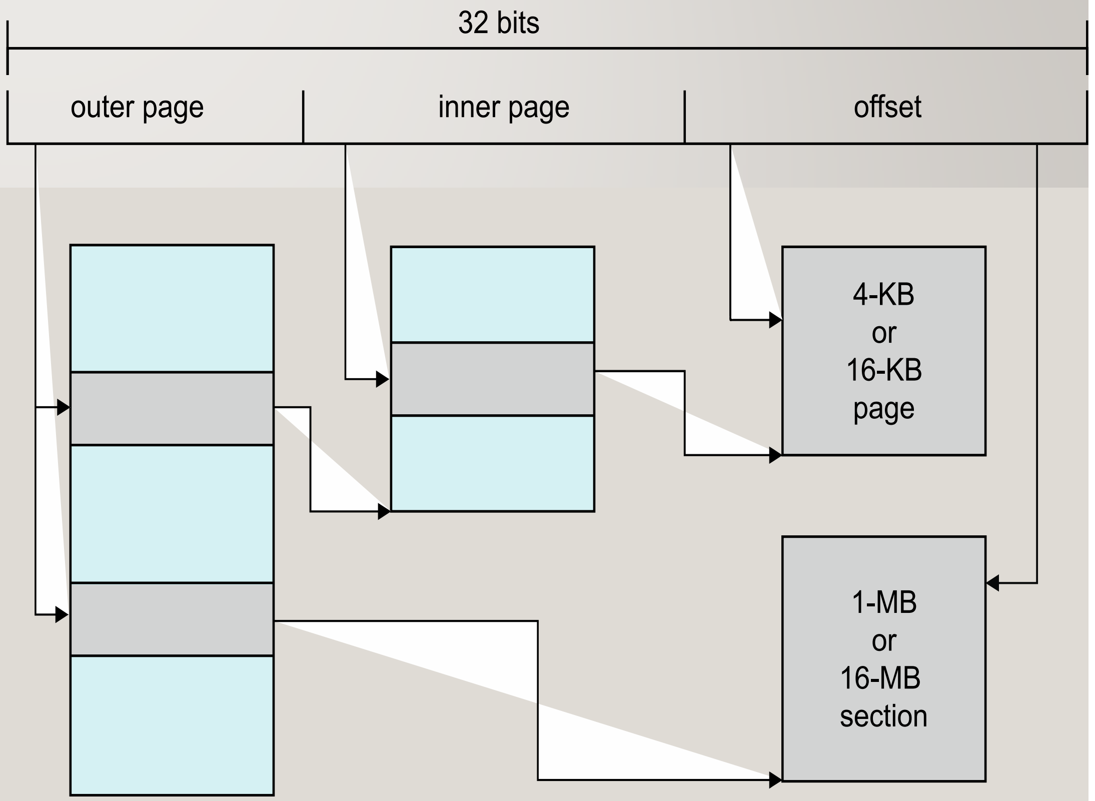
INTEL X86-64

- Current generation Intel x86 architecture
- 64 bits is ginormous (> 16 exabytes)
- In practice only implement 48 bit addressing
    - Page sizes of 4 KB, 2 MB, 1 GB
    - Four levels of paging hierarchy
- Can also use PAE so virtual addresses are 48 bits and physical addresses are 52 bits



EXAMPLE: ARM ARCHITECTURE

- Dominant mobile platform chip (Apple iOS and Google Android devices for example)
- Modern, energy efficient, 32-bit CPU
- 4 KB and 16 KB pages
- 1 MB and 16 MB pages (termed **sections**)
- One-level paging for sections, two-level for smaller pages
- Two levels of TLBs
  - Outer level has two micro TLBs (one data, one instruction)
  - Inner is single main TLB
  - First inner is checked, on miss outers are checked, and on miss page table walk performed by CPU



32 bits
outer page
inner page
offset
4-KB or 16-KB page
1-MB or 16-MB section