CSC 4320/6320: Operating Systems

Chapter 09: Main Memory – Last part

Spring 2025

Instructor: Dr. Md Mahfuzur Rahman
Department of Computer Science, GSU

Disclaimer

The slides to be used in this course have been created, modified, and adapted from multiple sources:

Chapter 9: Memory Management

- Background
- Contiguous Memory Allocation
- Paging
- Structure of the Page Table
- Swapping
- Example: The Intel 32/64-bit Architectures
- ~~Example: ARMv8 Architecture~~

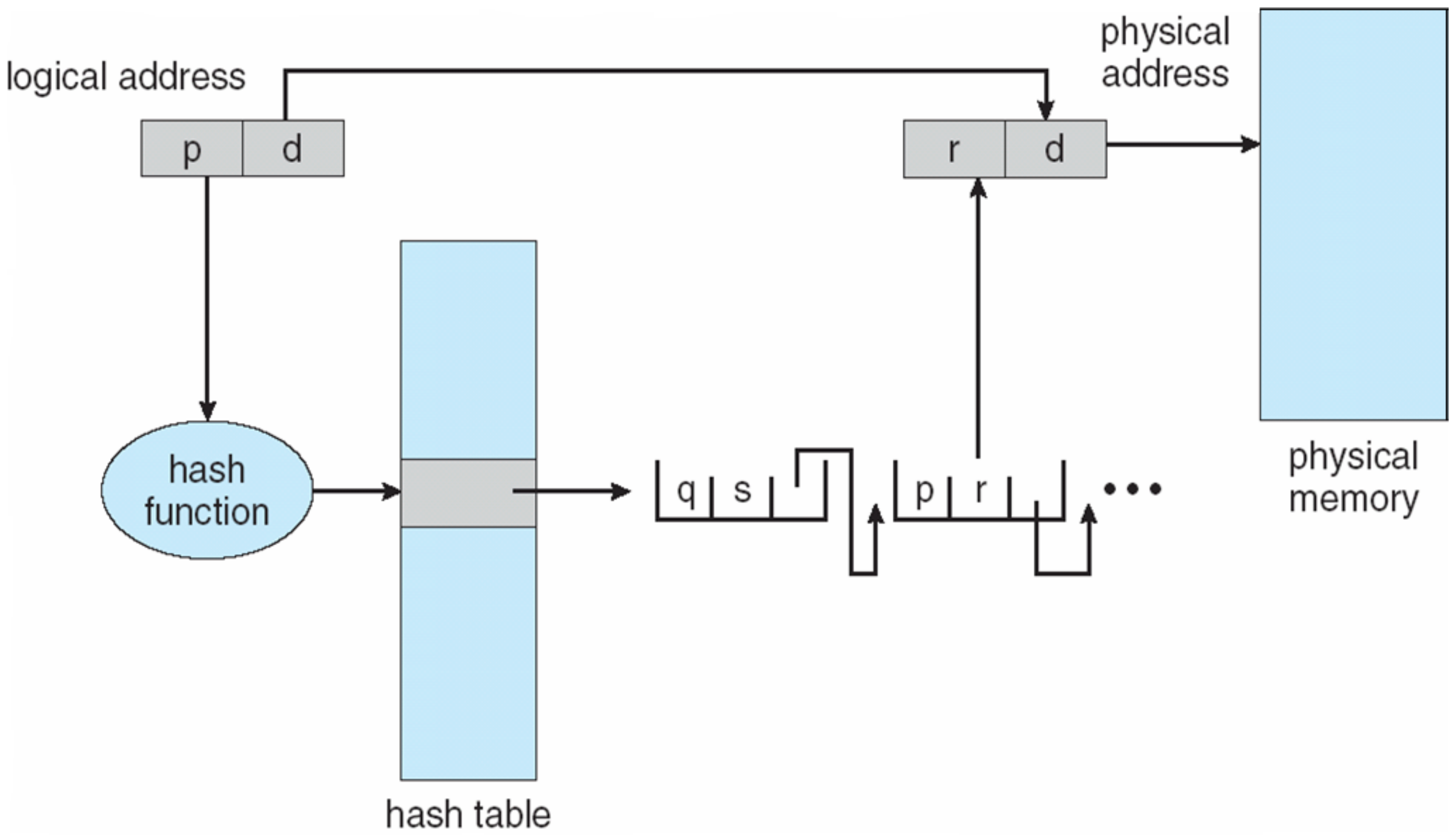
Objectives

- To provide a detailed description of various ways of organizing memory hardware
- To discuss various memory-management techniques,
- To provide a detailed description of the Intel Pentium, which supports both pure segmentation and segmentation with paging

Hashed Page Tables

- Common in address spaces > 32 bits
- The virtual page number is hashed into a hash table
  - This hash table contains a chain of elements hashing to the same location
- Each element contains (1) the virtual page number (2) the value of the mapped page frame (3) a pointer to the next element
- Virtual page numbers are compared in this chain searching for a match
  - If a match is found, the corresponding physical frame is extracted
- Variation for 64-bit addresses is **clustered page tables**
  - Similar to hashed page tables but each entry refers to several pages (such as 16) rather than 1
  - Especially useful for **sparse** address spaces (where memory references are non-contiguous and scattered)

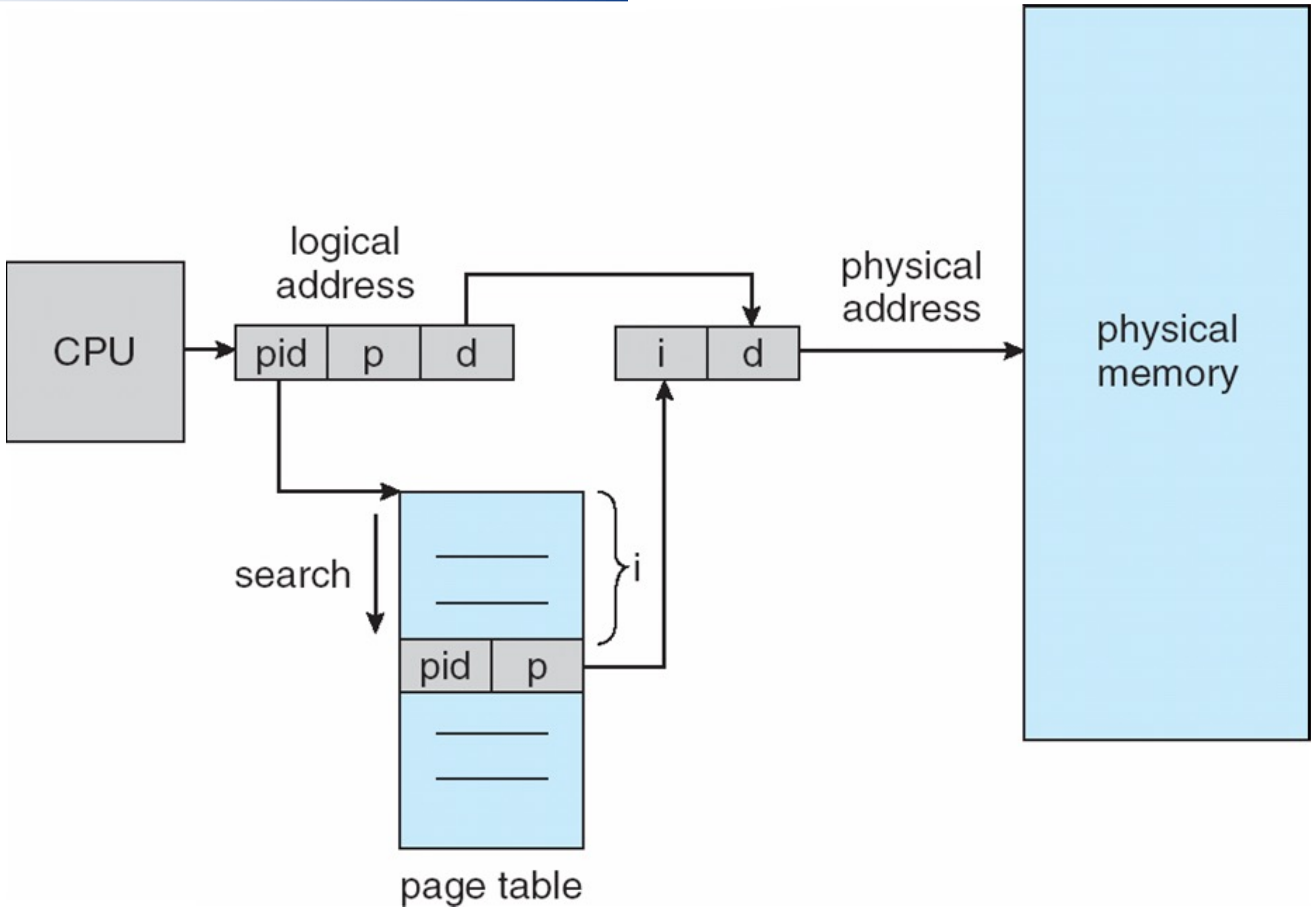
Hashed Page Table



Inverted Page Table

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages (frames)
- One entry for each real page of memory (frame)
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries
  - TLB can accelerate access
- But how to implement shared memory?
  - One mapping of a virtual address to the shared physical address

Inverted Page Table Architecture



When a memory reference occurs, part of the virtual address, consisting of <process-id, page-number>, is presented to the memory subsystem. The inverted page table is then searched for a match.

Questions

1. A(n) ____ page table has one page entry for each real page (or frame) of memory.
A) inverted ✓
B) clustered
C) forward-mapped
D) virtual
2. Consider a 32-bit address for a two-level paging system with an 8 KB page size. The outer page table has 1024 entries. How many bits are used to represent the second-level page table?
A) 10
B) 8
C) 12
D) 9 ✓
3. Which of the following statements are true with respect to hashed page tables?
A) They only work for sparse address spaces.
B) The virtual address is used to hash into the hash table.
C) A common approach for handling address spaces larger than 32 bits. ✓
D) Hash table collisions do not occur because of the importance of paging.

Question

Consider a computer system with a 32-bit logical address and 8-KB page size. The system supports up to 1 GB of physical memory. How many entries are there for inverted page table?

An **inverted page table** has one entry for each frame in physical memory.

- **Logical address size**: 32 bits
- **Page size/ frame size** : 8 KB (or 8×1024=8192 bytes)
- **Physical memory size**: 1 GB (or 1×1024×1024×1024=1,073,741,824 bytes)
- **How many frames?**
    - 1,073,741,824 /8192 = 131,072

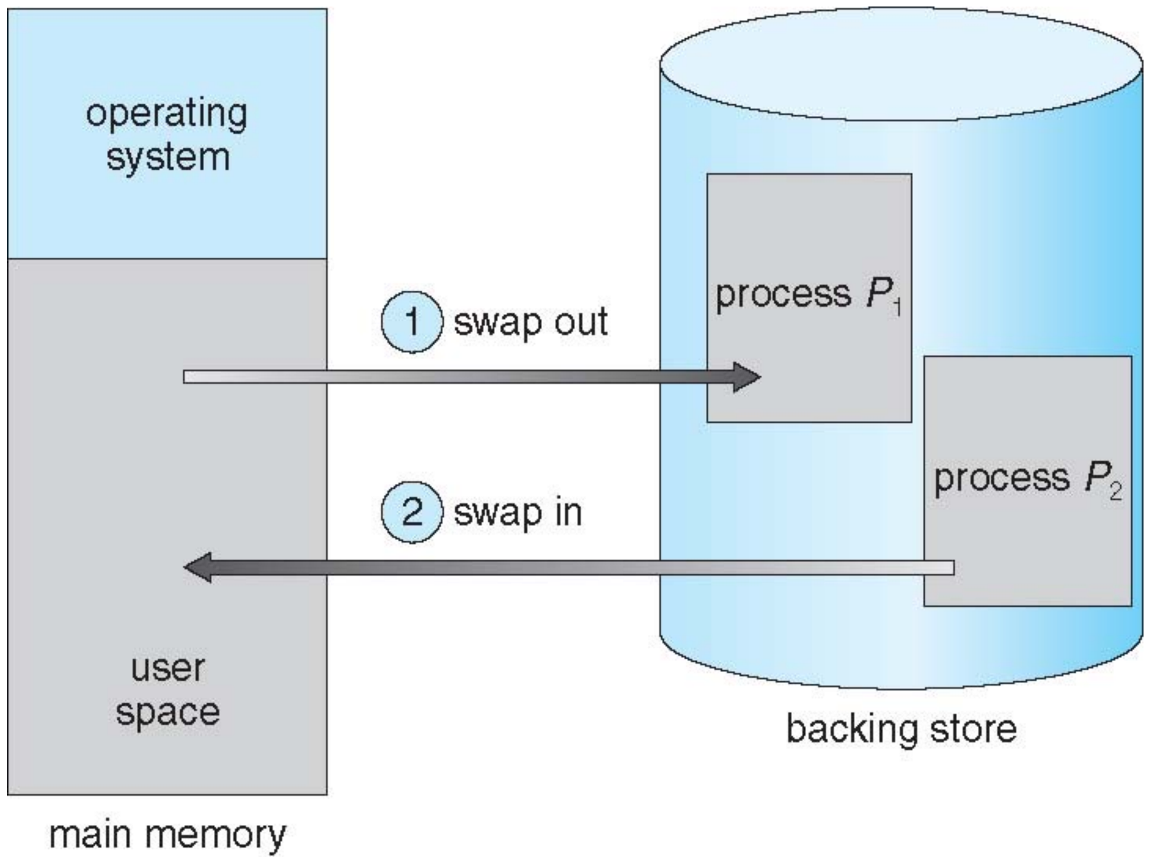
Swapping

- A process can be **swapped** temporarily out of memory to a backing store, and then brought **back** into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

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

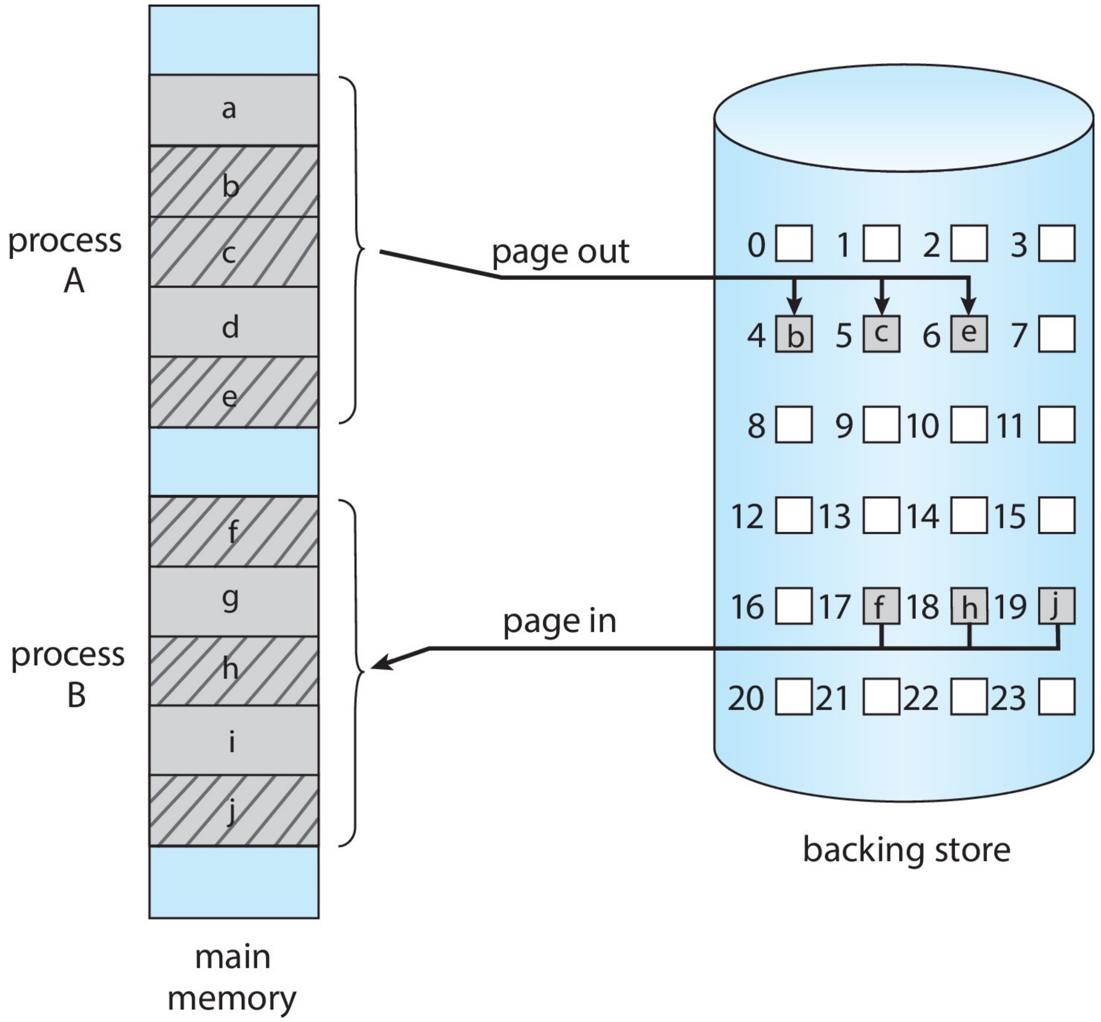
- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process
- Context switch time can then be very high
- 100MB process swapping to hard disk with transfer rate of 50MB/sec
  - Swap out time of 2000 ms
  - Plus swap in of same sized process
  - Total context switch swapping component time of 4000ms (4 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 (standard swapping always swaps out an existing process, regardless of memory situation)

Swapping with Paging

a subset of pages for processes A and B are being **paged-out** and **paged-in** respectively



Questions

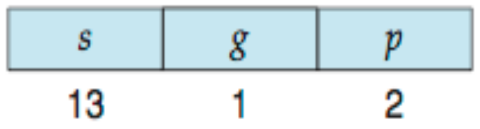
1. Which of the following is not a reason explaining why mobile devices generally do not support swapping?
A) Limited space constraints of flash memory.
B) Small size of mobile applications do not require use of swap space. ✓
C) Limited number of writes of flash memory.
D) Poor throughput between main memory and flash memory.

2. Standard swapping is generally not used in contemporary operating systems, because
A) some processes are so large that they can't fit in backing store.
B) memory in contemporary systems is large enough to store all processes.
C) the amount of time required to move entire processes between memory and the backing store is prohibitive. ✓
D) contemporary operating systems do not oversubscribe memory.

3. Standard swapping involves swapping in pages of processes instead of entire processes.
True
False ✓

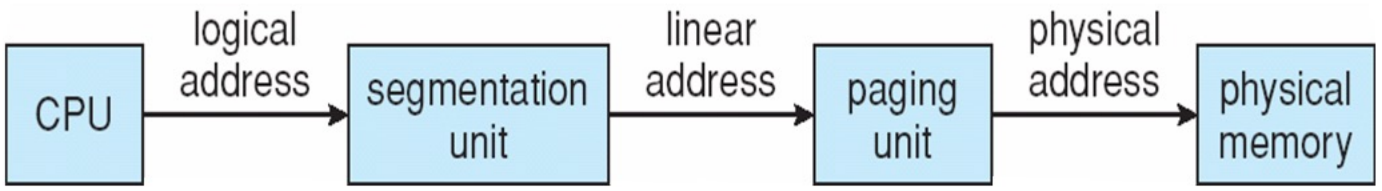
Example: The Intel IA-32 Architecture (Cont.)

- CPU generates logical address, which is a pair (selector, offset)
  - Selector 16 bits (as shown), Offset 32 bits (specifies location within 4GB segment)



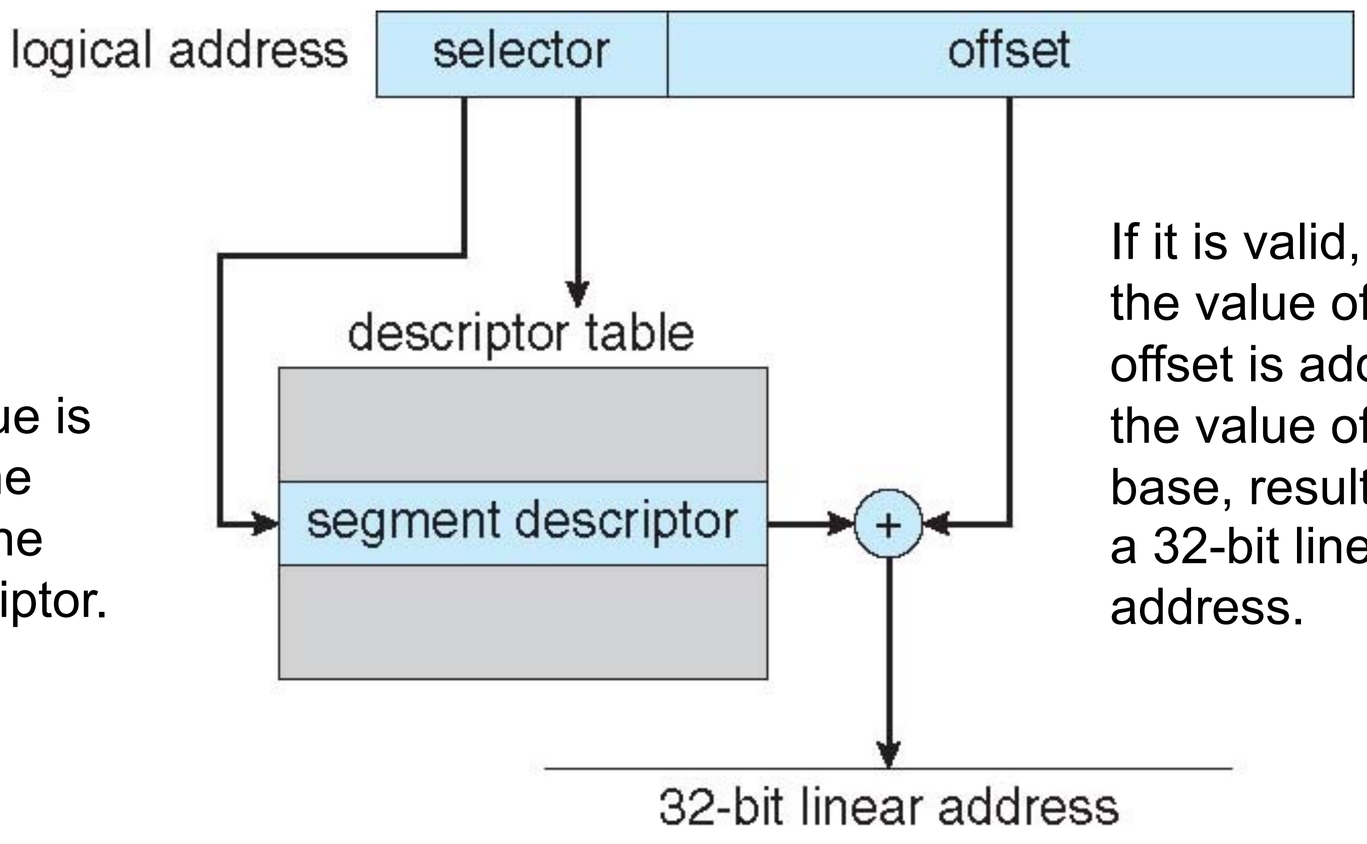
    - **s** designates the segment number,
    - **g** indicates whether the segment is in the global/local descriptor table (GDT/LDT)
    - **p** deals with protection.
  - Selector given to segmentation unit
    - Which produces linear addresses
  - Linear address given to paging unit
    - Which generates physical address in main memory
    - Pages sizes can be 4 KB or 4 MB
  - Segmentation and paging units form the equivalent of MMU

Logical to Physical Address Translation in IA-32



Intel IA-32 Segmentation

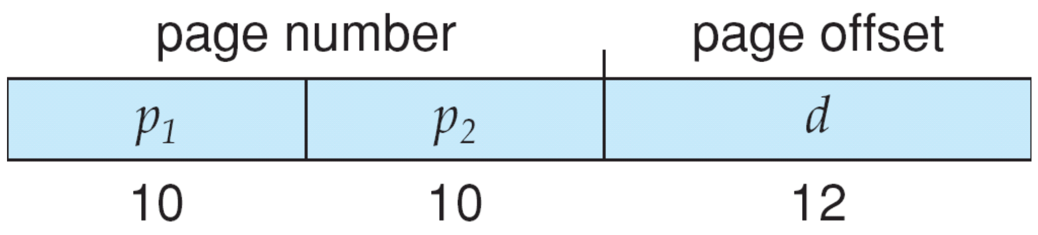
The **offset** value is compared to the **limit** value in the segment descriptor.



If it is valid, then the value of the offset is added to the value of the base, resulting in a 32-bit linear address.

Each entry in the LDT and GDT consists of an 8-byte segment descriptor with detailed information about a particular segment, including the base location and limit of that segment.

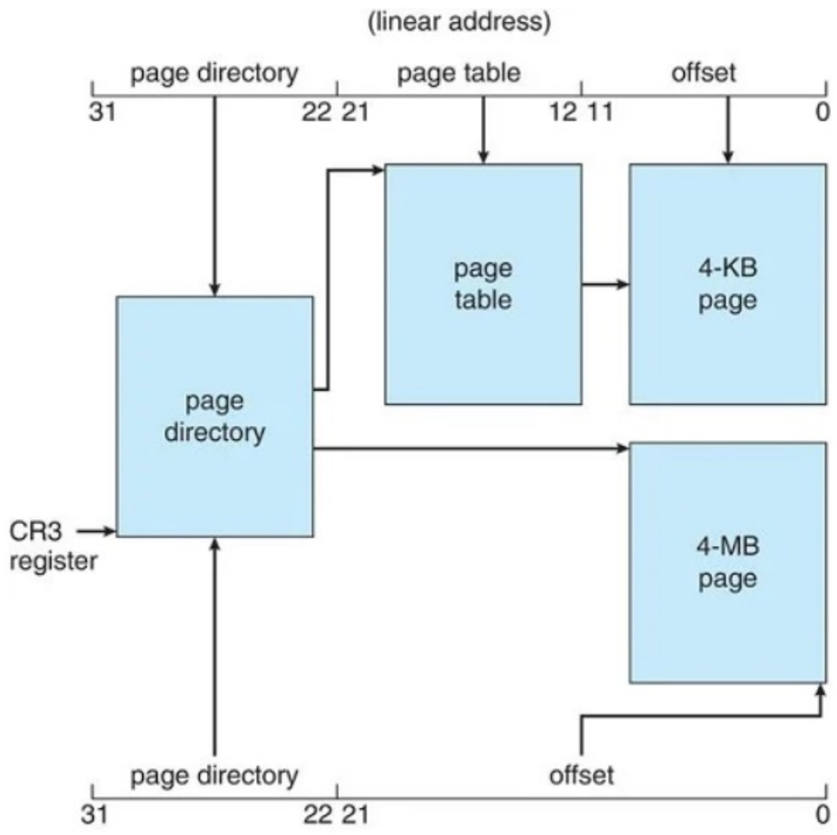
Linear Address to Physical Address Translation in IA-32



- Linear address is fed into paging unit for physical address generation.
- Both a page size of either 4 KB or 4 MB is possible.
- For 4-KB pages, IA-32 uses a two-level paging scheme in which the division of the 32-bit linear address is shown above

Intel IA-32 Paging Architecture

- The 10 high-order bits reference an entry in the outermost page table (page directory)
- The page directory entry points to an inner page table
- The contents of the innermost 10 bits in the linear address work as indices for the page table.
- the low-order bits 0–11 refer to the offset in the 4-KB page pointed to in the page table.
- If the size of the page frame is 4 MB, page directory directly points to the page frame.
  - 22 bits offset in that case (see bottom side of the figure)



Exercise

- Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):
  - 3085
  - 42095

- Consider a computer system with a 32-bit logical address and 4-KB page size. The system supports up to 512 MB of physical memory. How many entries are there in each of the following?

a. A conventional, single-level page table
b. An inverted page table