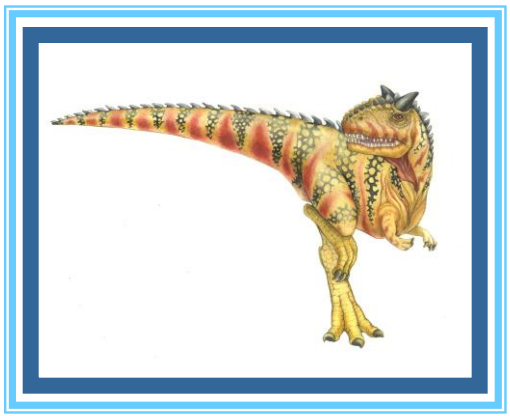
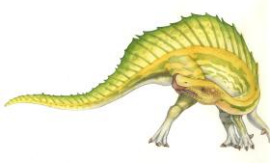
# Chapter 9: Main Memory





# Outline

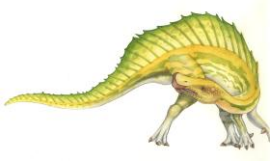
- Address binding, relocation, and protection
- Contiguous allocation and fragmentation
- Segmentation and paging
- TLBs and page-table structures
- Swapping and page-level reclaim
- IA-32, x86-64, and Armv8-A examples





# Objectives

- Explain virtual and physical addresses and how hardware translates between them.
- Compare contiguous allocation, segmentation, and paging.
- Calculate address fields, page-table sizes, internal fragmentation, and TLB access time.
- Distinguish a TLB miss from a page fault.
- Describe the key paging features of IA-32, x86-64, and AArch64.



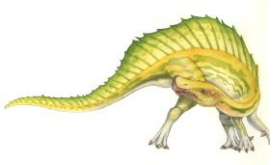


Troy University Resources



# Background

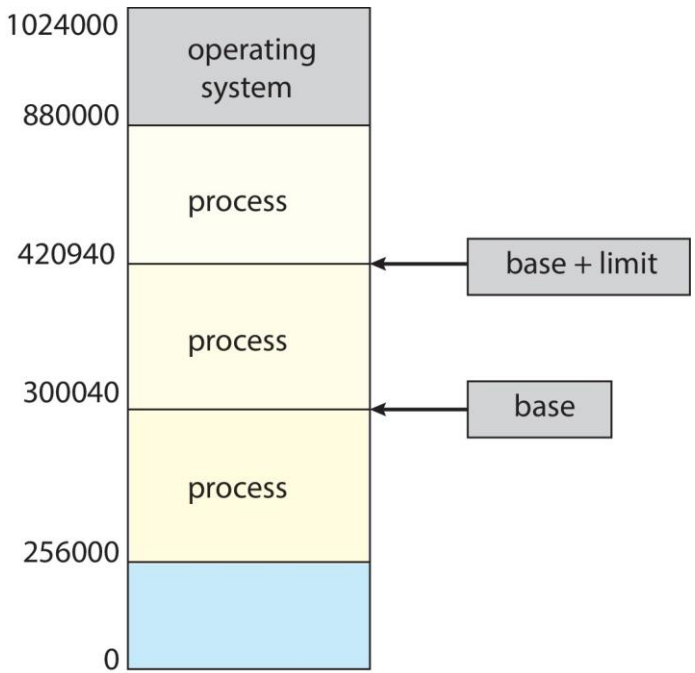
- A CPU executes instructions and accesses data through its memory hierarchy, including registers, caches, and main memory.
- Code and data must be available in memory when used. The entire program need not be resident at once.
- Register, cache, and DRAM access costs differ. A cache can satisfy an access without going to DRAM.
- Address translation and protection determine which memory a user process can access.

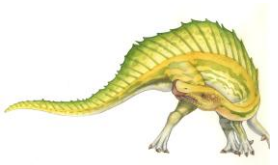




# Protection

- Need to ensure that a process can access only those addresses in its address space.
- We can provide this protection by using  a pair of **base** and **limit registers** define the logical address space of a process



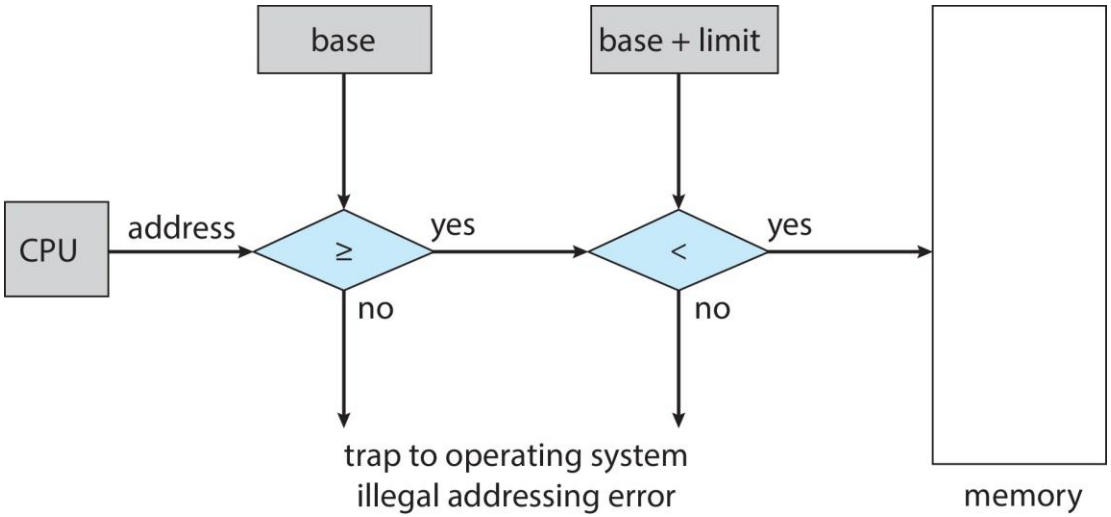


Troy University Resources

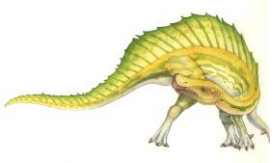


# Hardware Address Protection

- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user




- the instructions to loading the base and limit registers are privileged





# Address Binding

- Source code uses symbolic names for functions and variables.
- Compilers and assemblers produce object code with addresses and relocation information.
- Linkers combine objects and resolve references. Loaders map the resulting image into an address space.
- With execution-time translation, virtual addresses map to physical memory through the MMU.
- An image may be linked for a chosen address or relocated. There is no general requirement to load the first user program at physical address 0.



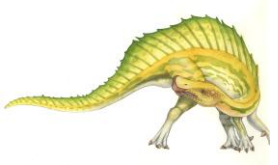


Troy University Resources



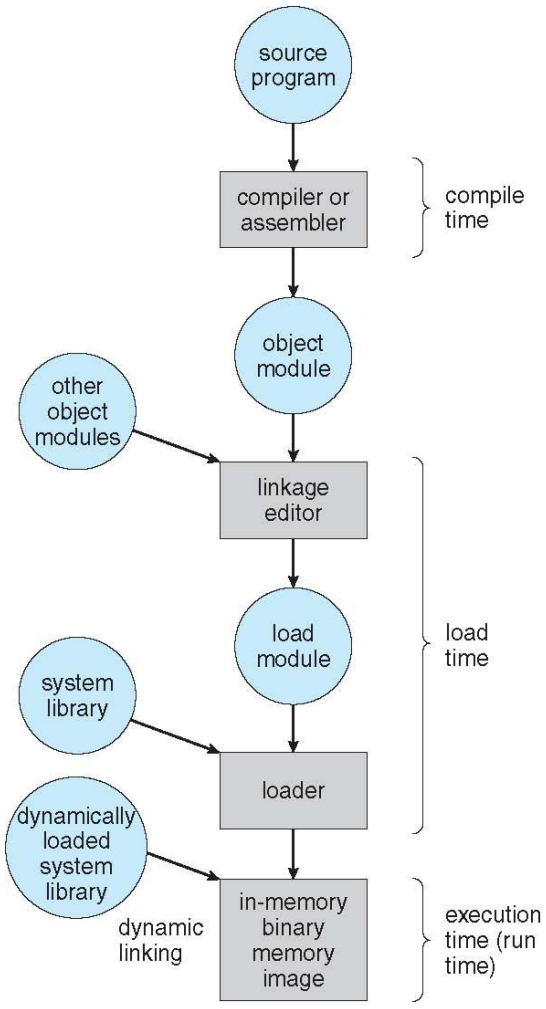
# Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can happen at three different stages
  - **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
  - **Load time**: Must generate **relocatable code** if memory location is not known at compile time
  - **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
    - Need hardware support for address maps (e.g., base and limit registers)





# Multistep Processing of a User Program





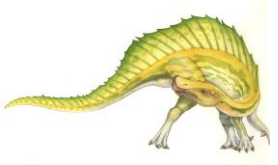



Troy University Resources



# Logical vs. Physical Address Space

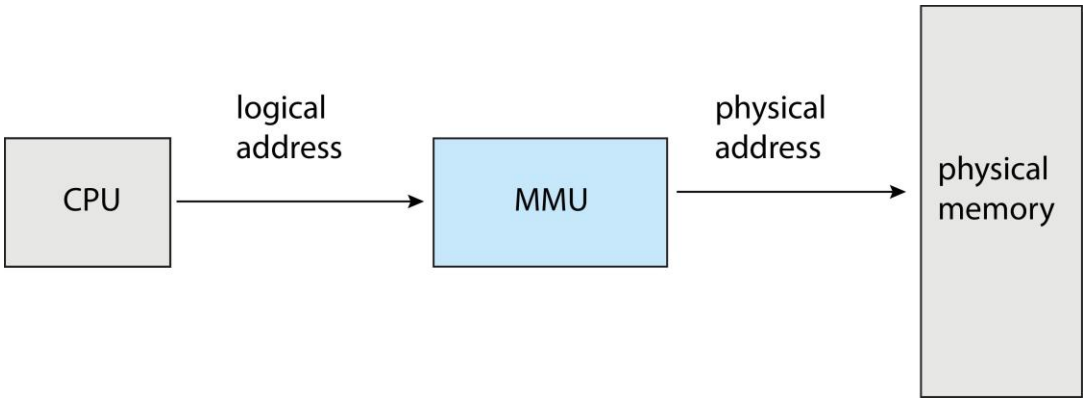
- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program





# Memory-Management Unit (MMU)

- Hardware device that at run time maps virtual to physical address



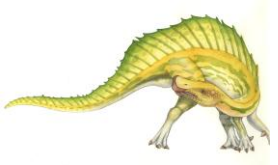

- Many methods possible, covered in the rest of this chapter





# Memory-Management Unit (Cont.)

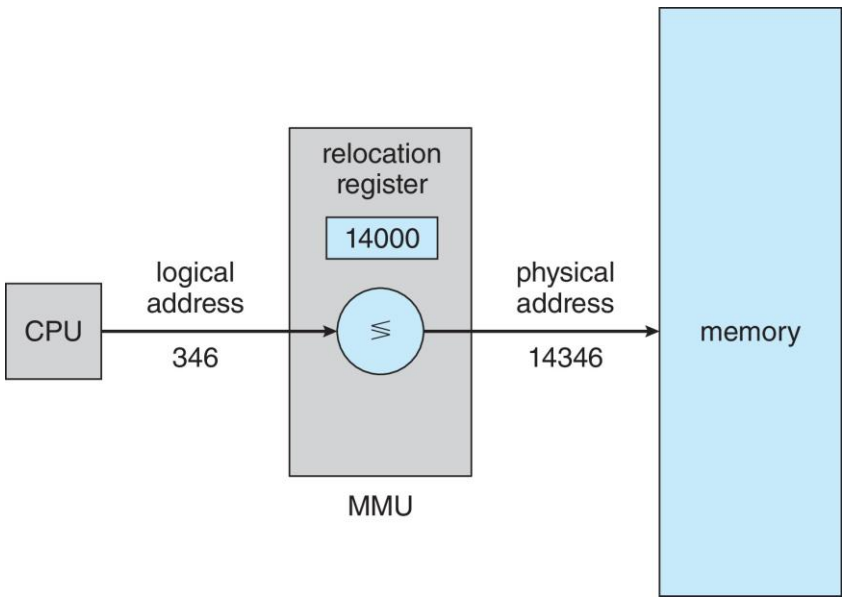
- Consider simple scheme. which is  a generalization of the base-register scheme.
- The base register now called **relocation register**
- The value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with *logical* addresses; it never sees the *real* physical addresses
  - Execution-time binding occurs when reference is made to location in memory
  - Logical address bound to physical addresses

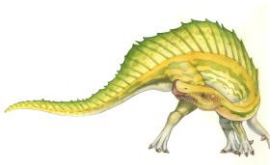




# Memory-Management Unit (Cont.)

- Consider simple scheme. which is  a generalization of the base-register scheme.
- The base register now called **relocation register**
- The value in the relocation register is added to every address generated by a user process at the time it is sent to memory



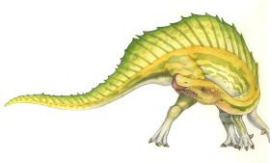


Troy University Resources



# Dynamic Loading

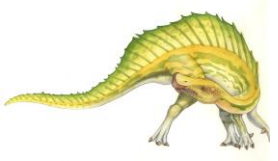
- The entire program does not need to be resident in memory to execute.
- A routine or module can be loaded when the program first needs it.
- Unused components need not occupy resident memory.
- A program can implement an explicit loading scheme, while OS loaders and runtime libraries provide practical support.
- Dynamic loading is different from demand paging, which transparently brings individual pages into memory.





# Dynamic Linking

- Static linking copies selected library code into an executable at link time.
- Dynamic linking resolves references to shared libraries at load time or during execution.
- A loader can use indirection tables or stubs to connect a call to the selected library routine.
- Read-only library code pages can be shared physically while processes have their own mappings and private data.
- Library versions and ABI compatibility govern whether a library update can be used safely.



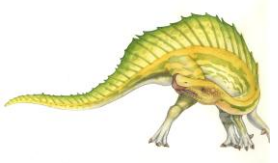


Troy University Resources



# Contiguous Allocation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two **partitions**:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
  - Each process contained in single contiguous section of memory





# Contiguous Allocation (Cont.)

- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically*
  - Can then allow actions such as kernel code being **transient** and kernel changing size

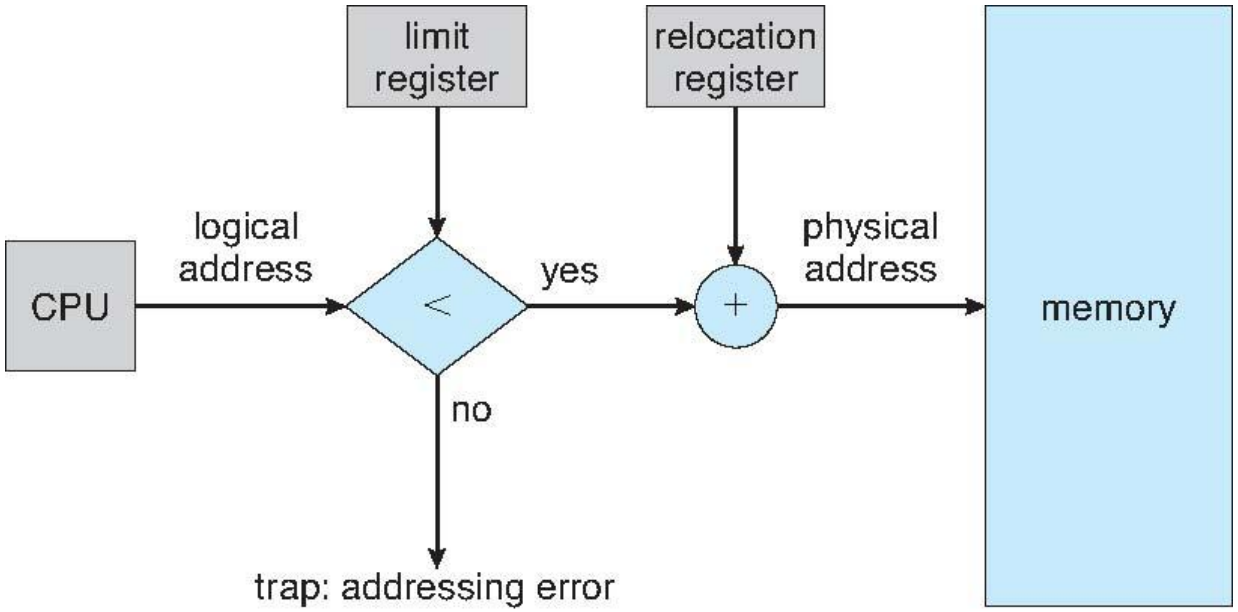




Troy University Resources



# Hardware Support for Relocation and Limit Registers

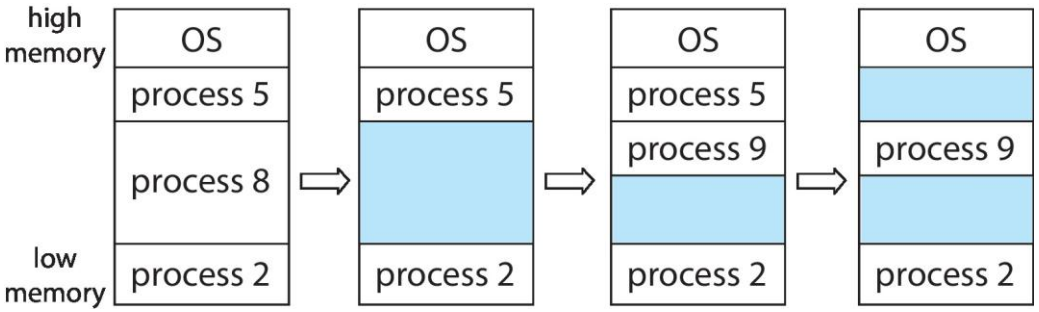


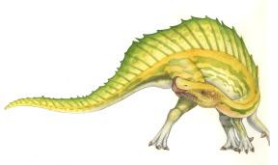





# Variable Partition

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
  - **Hole** – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    a) allocated partitions  b) free partitions (hole)



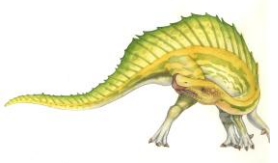


Troy University Resources



# Dynamic Storage Allocation

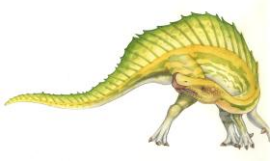
- First fit selects the first free hole large enough for a request.
- Best fit selects the smallest suitable hole. A size-ordered structure can help the search.
- Worst fit selects the largest hole.
- The leftover hole size, search cost, and future requests determine performance.
- There is no universal winner. First fit is simple and often effective, while best fit can leave many small unusable holes.





# Internal and External Fragmentation

- External fragmentation: enough free memory exists in total, but no single contiguous hole is large enough.
- Internal fragmentation: allocated space exceeds the space the object actually uses.
- The classical 50-percent rule estimates roughly one free hole per two allocated blocks under particular steady-state assumptions.
- That rule counts holes. It does not imply that exactly one third of memory bytes must be unusable.



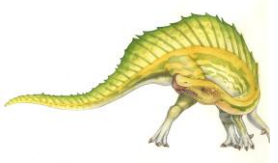


Troy University Resources



# Fragmentation (Cont.)

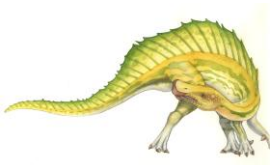
- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers
- Now consider that backing store has same fragmentation problems





# Segmentation

- Segmentation presents a program as logical, variable-size units.
- A segment can represent code, a procedure, an array, a stack, or shared data.
- A logical address identifies a segment and an offset within that segment.
- Segments may be placed separately in physical memory. Variable-size allocation can cause external fragmentation.





Troy University Resources



# A Segment Table and Address Translation

Translate ⟨s, d⟩ only if s is a valid segment and 0 ≤ d < limit[s]. Then physical = base[s] + d.

| Segment | Base | Limit (length) |
|---|---|---|
| 0: code | 1400 | 1000 |
| 1: data | 6300 | 400 |
| 2: stack | 4300 | 700 |

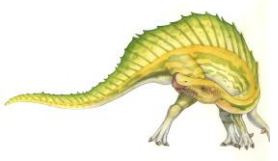
⟨1, 120⟩ maps to 6420. ⟨1, 400⟩ is outside the segment and traps.





# Segment Tables and Protection

- The segment-table base register (STBR) identifies the table. STLR gives the number of valid segment entries.
- The segment number must be within the table, and the offset must be within the selected segment.
- An entry can hold a presence or validation bit and read, write, or execute permissions.
- Several processes can share a segment while keeping other segments private.
- Segmentation describes logical variable-size units. Paging uses fixed-size pages and frames.

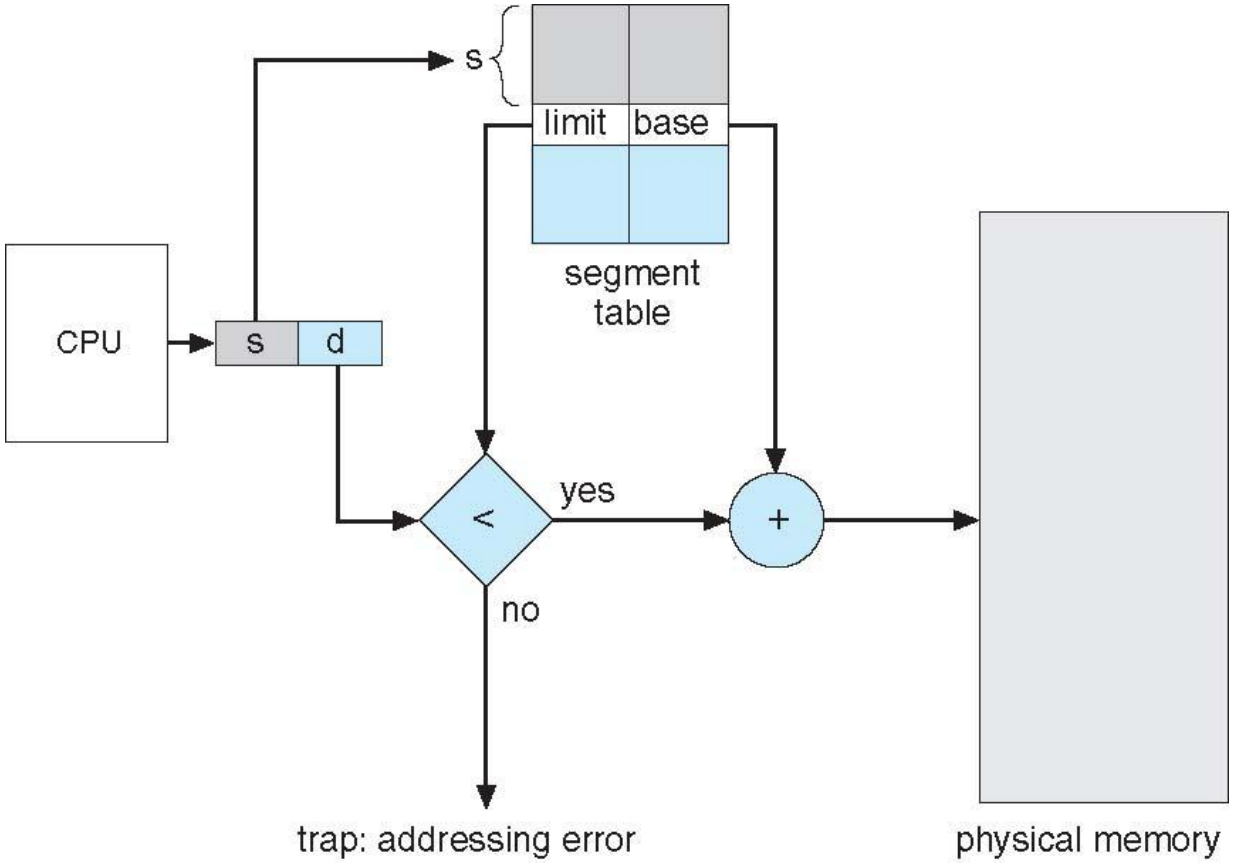


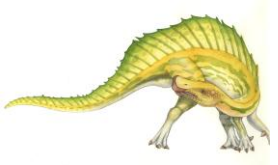


Troy University Resources



# Segmentation Hardware

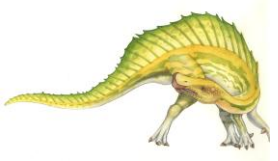




# Paging

- Divide virtual memory into fixed-size pages and physical memory into frames of the same size.
- A page table maps virtual page numbers to physical frame numbers.
- A process's frames can be scattered through physical memory, so a page need not join its neighbors physically.
- Paging avoids external fragmentation for allocation in base-page units. Unused space within an allocated page is internal fragmentation.
- In a fully resident model, all N pages need frames. Demand paging loads only the pages that are needed.



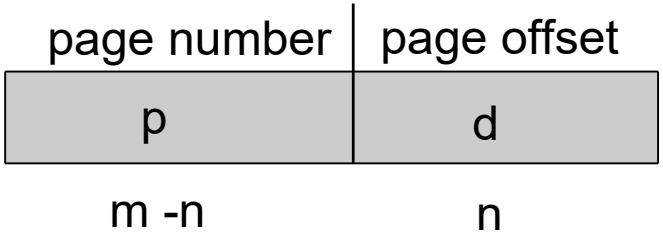


Troy University Resources

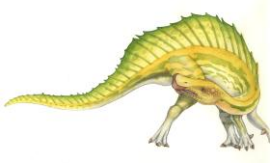


# Address Translation Scheme

- Address generated by CPU is divided into:
  - **Page number** (***p***) – used as an index into a **page table** which contains base address of each page in physical memory
  - **Page offset** (***d***) – combined with base address to define the physical memory address that is sent to the memory unit

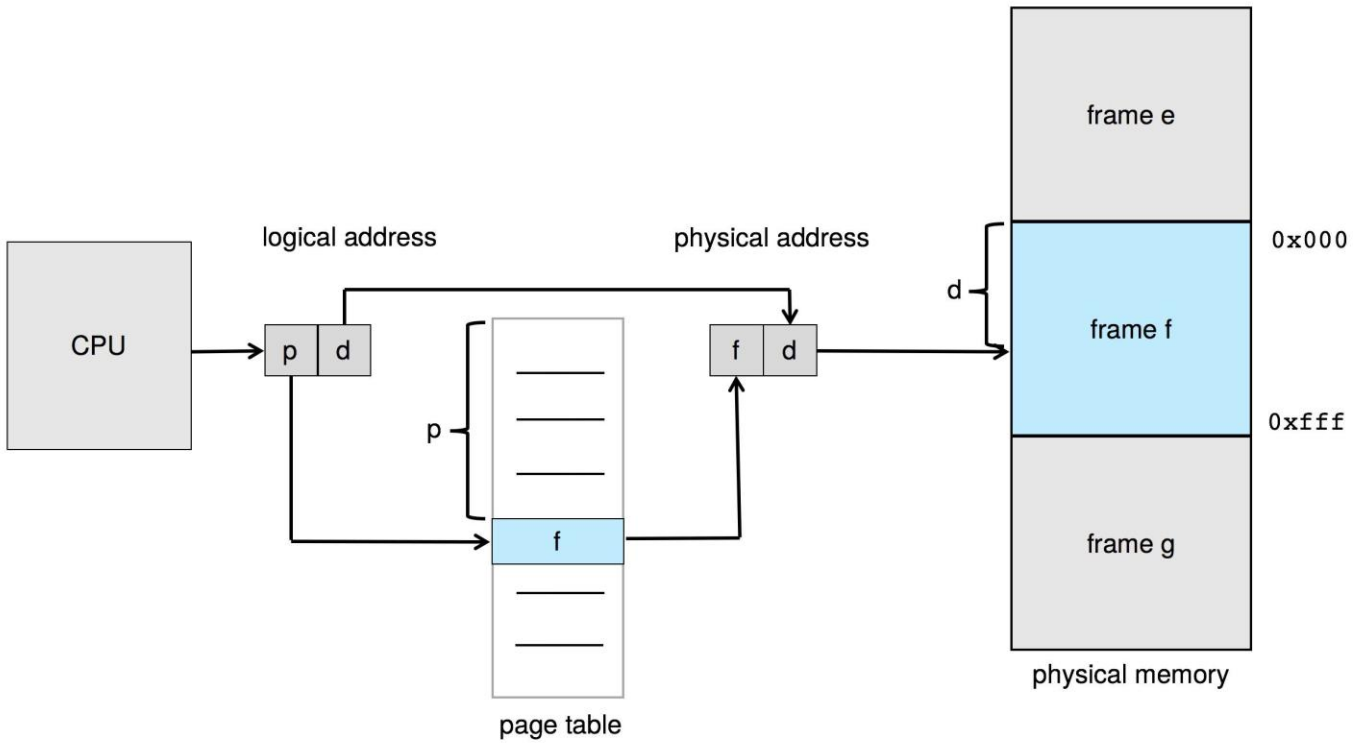



  - For given logical address space $2^m$ and page size $2^n$



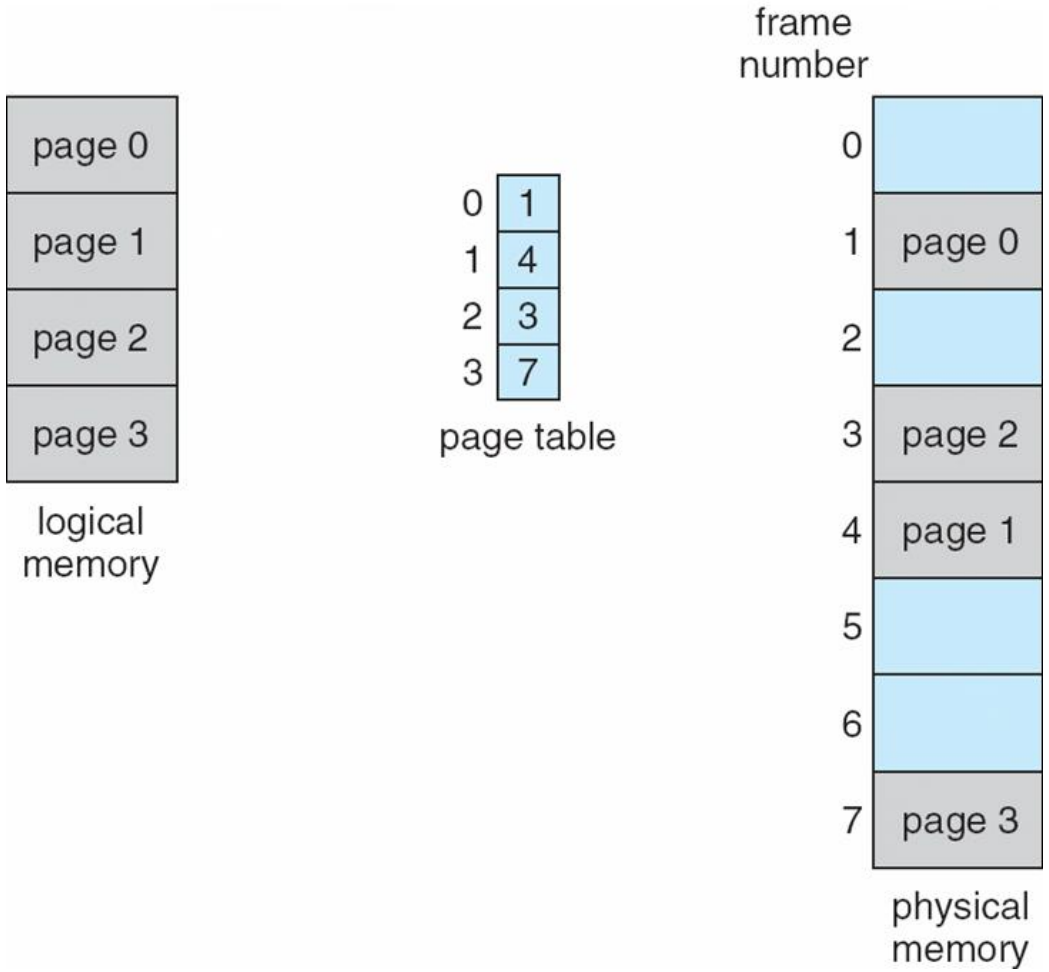


# Paging Hardware





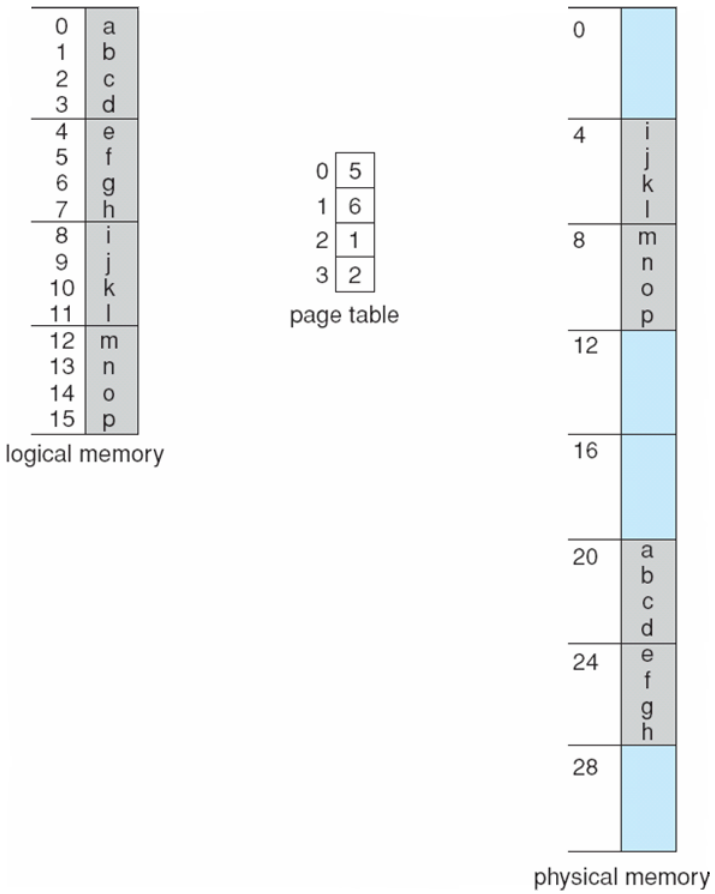

# Paging Model of Logical and Physical Memory





# Paging Example

- Virtual address width m = 4 bits, offset width n = 2 bits. Page size = 4 bytes; physical memory = 32 bytes = 8 frames.





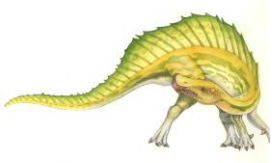



Troy University Resources



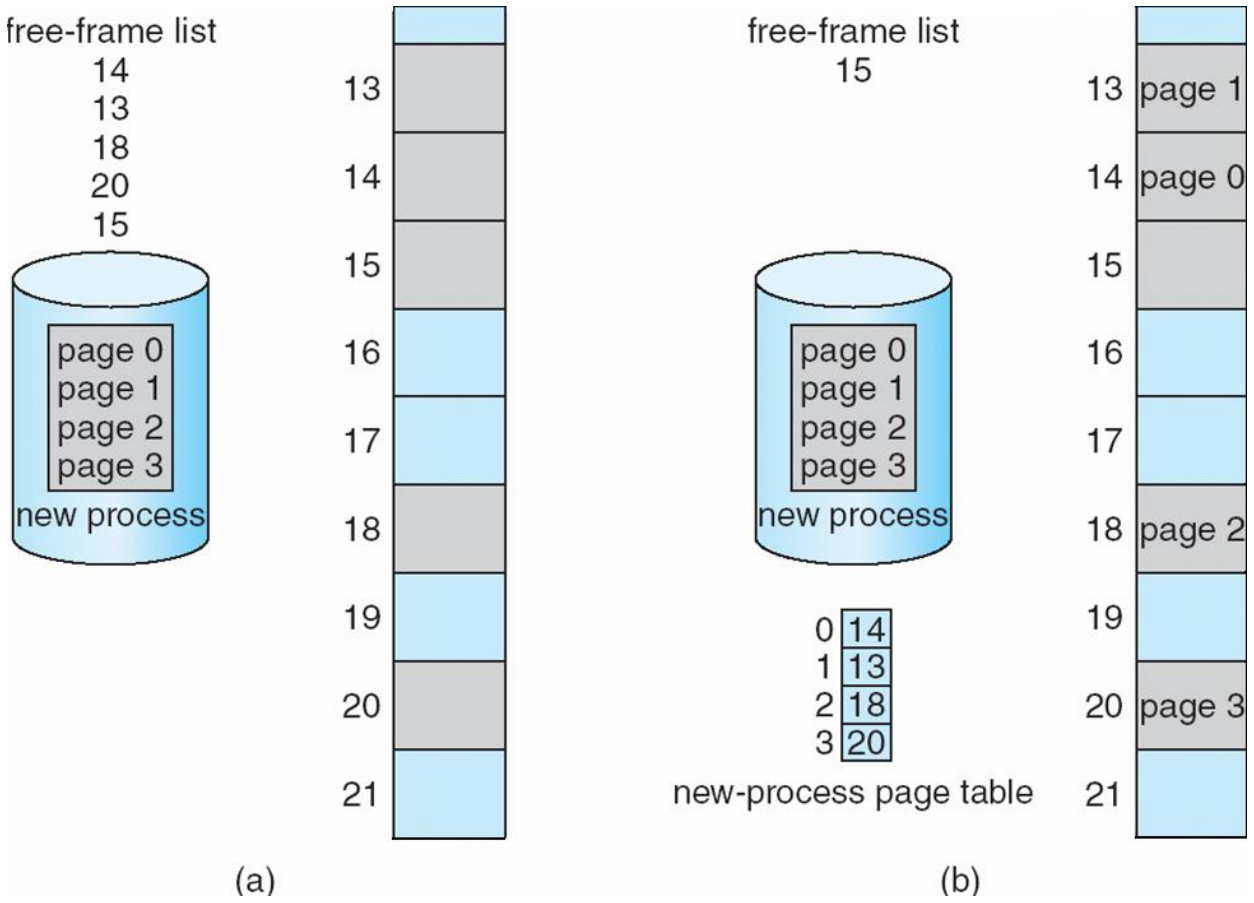
# Paging: Internal Fragmentation

- Page size = 2048 bytes. Process size = 72,766 bytes.
- 72,766 = 35 × 2048 + 1086, so the process needs 36 pages.
- The last page wastes 2048 - 1086 = 962 bytes.
- Worst-case waste in the last page is page size - 1 byte. With uniformly distributed remainders, average waste is about half a page.
- Smaller pages reduce this waste but require more page-table entries. Supported page sizes depend on the architecture.



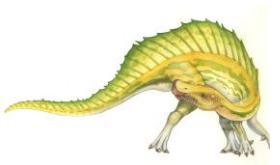


# Free Frames




Before allocation

After allocation



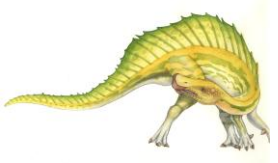


Troy University Resources



# Page Tables and the TLB

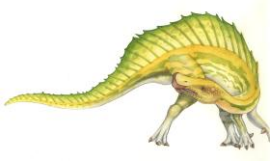
- Page-table entries usually reside in main memory. A hardware register identifies the root table.
- With a one-level table and no translation cache, a load needs a table access plus the data access.
- Multilevel walks may need more table accesses. Caches and page-walk caches can reduce their cost.
- A translation lookaside buffer (TLB) caches recently used translations and permissions.
- A TLB hit can avoid the page-table walk. It does not cache the page's application data.





# TLB Entries and Address-Space Tags

- A TLB caches virtual-page translations and access permissions.
- ASIDs or PCIDs identify address spaces and can preserve translations across context switches.
- Invalidate stale translations when mappings change or tags are reused.
- Without suitable tags, an address-space switch requires invalidating conflicting entries.
- A TLB miss causes translation work. It does not by itself imply a page fault.





Troy University Resources

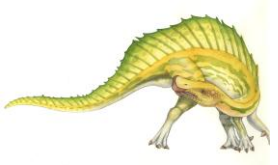


# Hardware

- Associative memory – parallel search

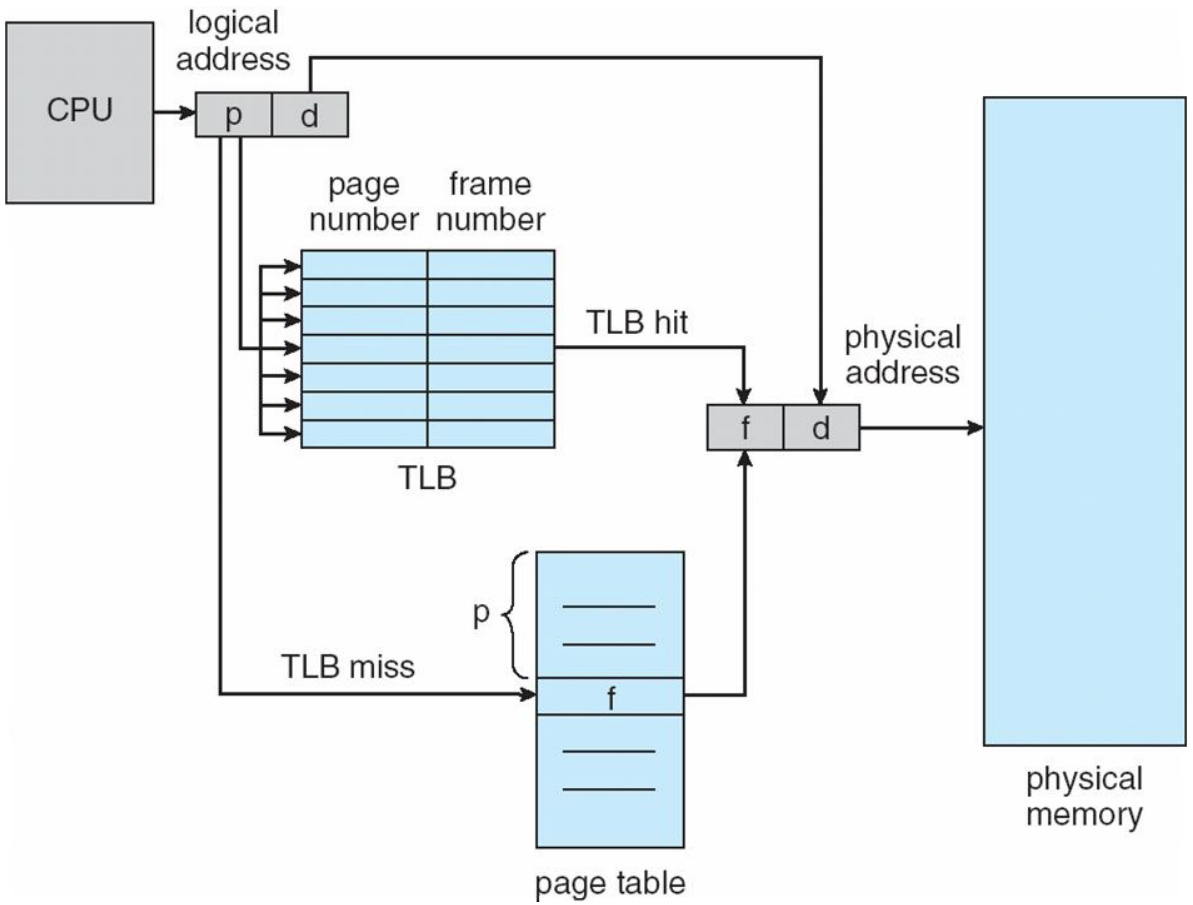
| Page # | Frame # |
|---|---|
| | |
| | |
| | |
| | |

- Address translation (p, d)
  - If p is in associative register, get frame # out
  - Otherwise get frame # from page table in memory





# Paging Hardware With TLB

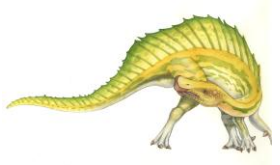







# Effective Access Time

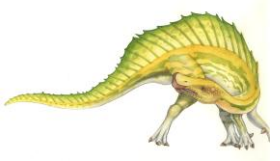
- Assume a one-level page table, 10 ns per memory access, negligible TLB lookup time, and no page faults.
- TLB hit: one memory access = 10 ns. TLB miss: table plus data = 20 ns.
- At hit ratio h = 0.80: EAT = 0.80 × 10 + 0.20 × 20 = 12 ns, a 20% increase.
- At h = 0.99: EAT = 0.99 × 10 + 0.01 × 20 = 10.1 ns, a 1% increase.
- With a serial TLB lookup cost ε, the model becomes EAT = ε + (2 - h) × 10 ns.





# Memory Protection

- Permissions belong to virtual mappings in page-table entries, such as readable, writable, user-accessible, and executable.
- A physical frame can be mapped by several processes with different permissions.
- In this chapter's simple model, a valid entry denotes a legal mapped page. An invalid entry rejects an address outside that mapping.
- Demand paging also needs to distinguish a legal but nonresident page from an invalid address.
- Hardware traps on a relevant not-present or protection failure so the kernel can handle it.

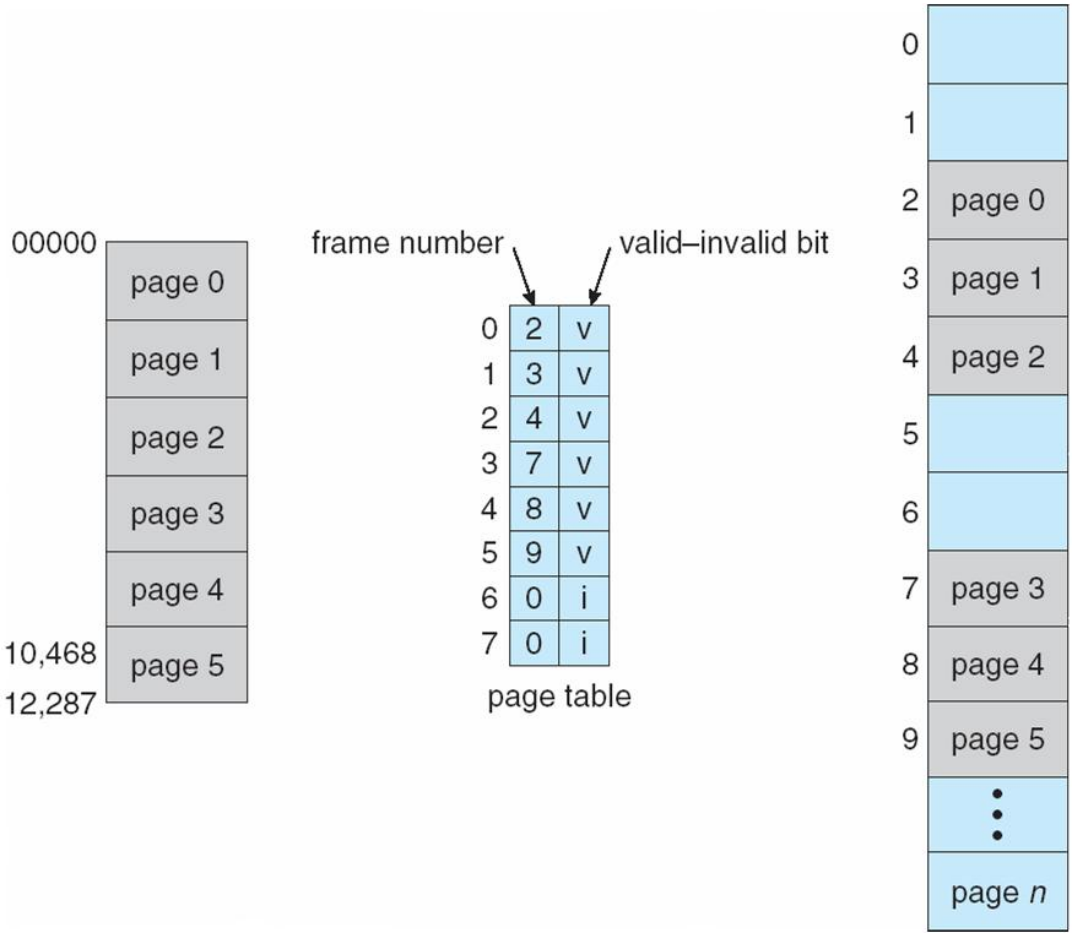




Troy University Resources



# Valid (v) or Invalid (i) Bit In A Page Table

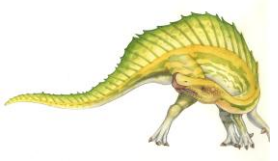







# Shared Pages

- **Shared code**
  - One copy of read-only (**reentrant**) code shared among processes (i.e., text editors, compilers, window systems)
  - Similar to multiple threads sharing the same process space
  - Also useful for interprocess communication if sharing of read-write pages is allowed
- **Private code and data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space

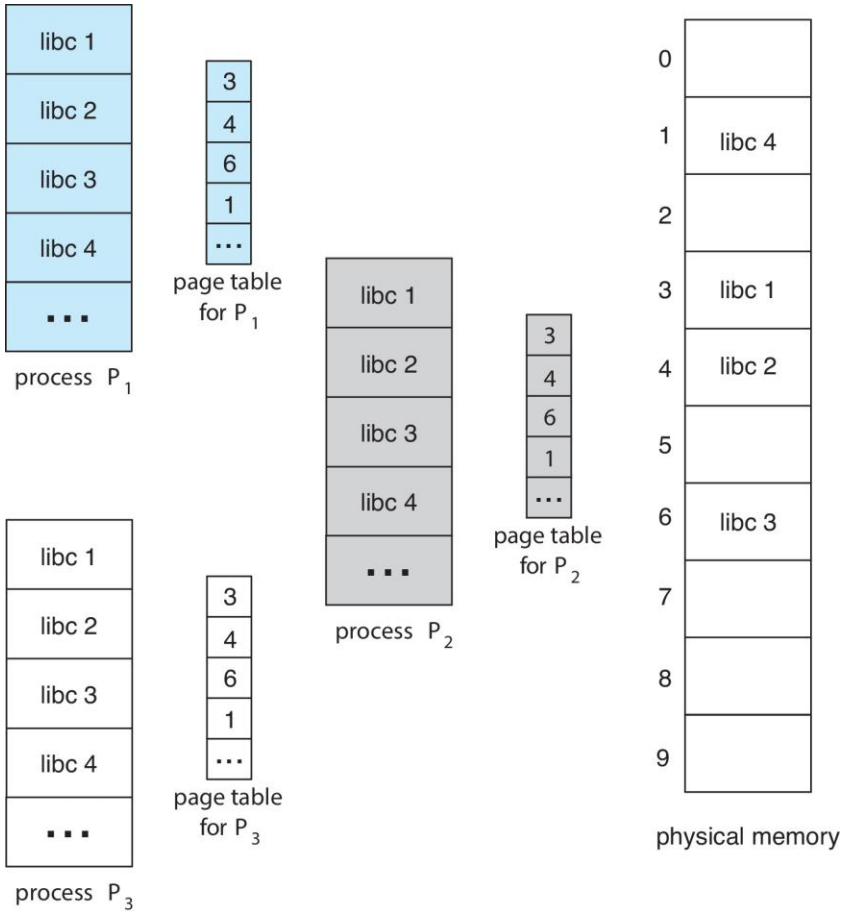




Troy University Resources



# Shared Pages Example

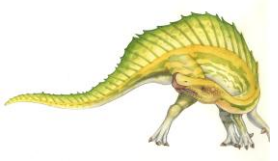







# Page-Table Size

- Example: 32-bit virtual addresses, 4 KiB pages, and 4-byte page-table entries.
- The offset uses 12 bits, leaving 20 bits for the virtual page number.
- A fully populated linear table has $2^{20}$ entries and occupies $2^{20} \times 4 = 4$ MiB per address space.
- Multilevel tables allocate lower-level tables only for used portions of a sparse address space.
- Hashed and inverted tables use different indexing strategies to reduce metadata or suit other architectures.



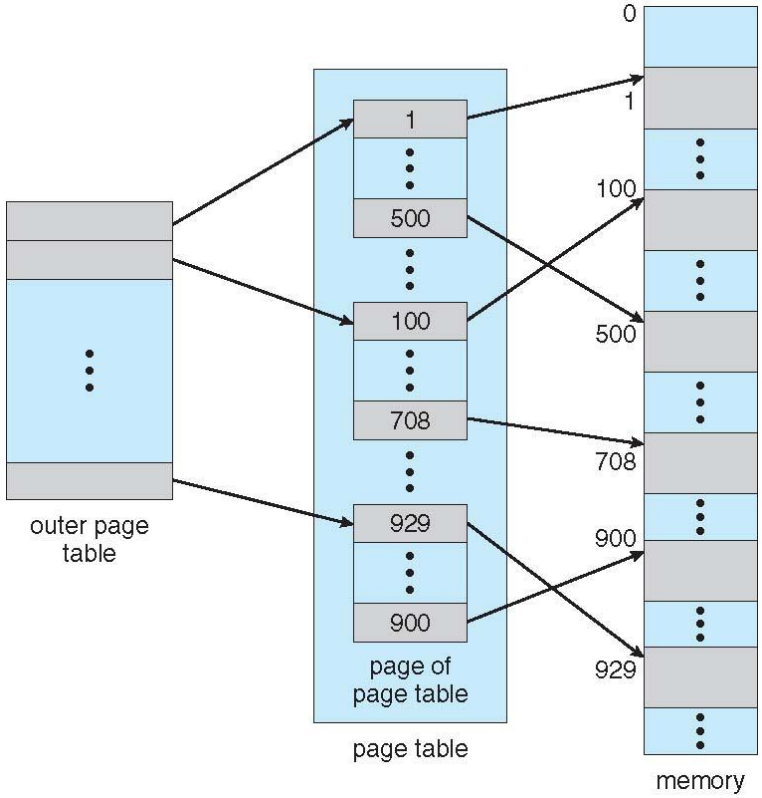


Troy University Resources



# Hierarchical Page Tables

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table
- We then page the page table

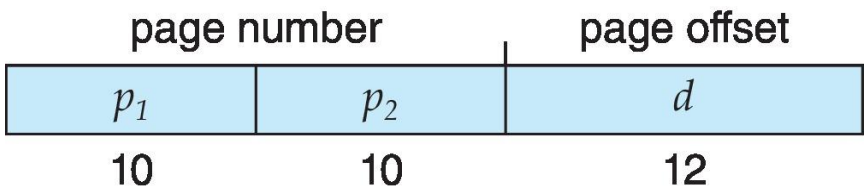




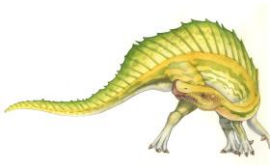



# Two-Level Paging Example

- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page number consisting of 20 bits
  - a page offset consisting of 12 bits
- Since the page table is paged, the page number is further divided into:
  - a 10-bit outer-table index
  - a 10-bit inner-table index
- Thus, a logical address is as follows:




- p1 indexes the outer table. p2 selects a page-table entry in the inner table; d is the byte offset in the final page.
- Known as **forward-mapped page table**

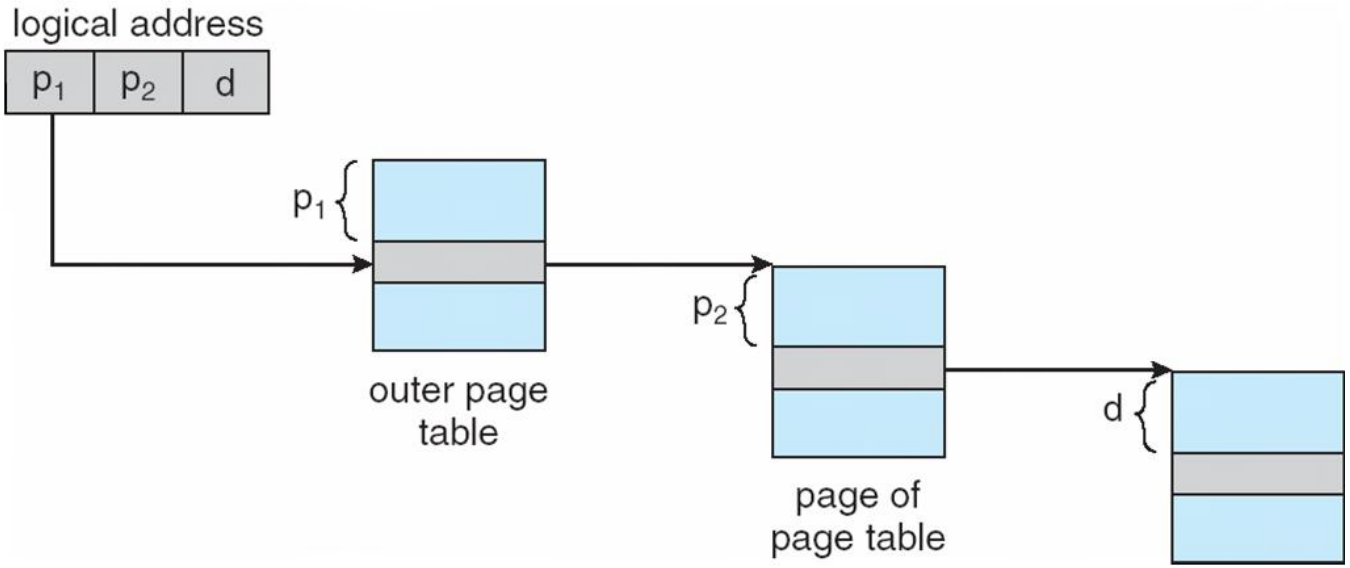


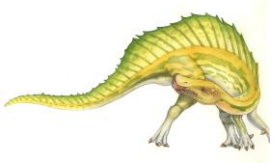


Troy University Resources



# Address-Translation Scheme

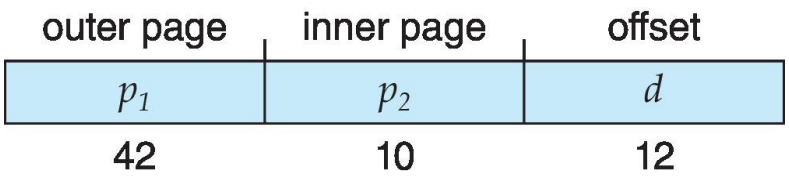


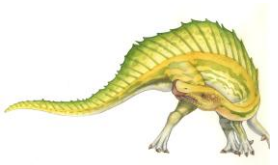


# A Theoretical 64-Bit Page Table

- With 4 KiB pages, a full 64-bit virtual address has 52 page-number bits.
- A flat page table therefore needs 2^52 entries.




- The figure shows two levels: 42 + 10 + 12 address bits.
- At 4 bytes per entry, that outer table uses 2^44 bytes (16 TiB).
- A further 32 + 10 split leaves a 2^34-byte (16 GiB) outermost table.





Troy University Resources



# Three-level Paging Scheme

| outer page | inner page | offset |
|---|---|---|
| $p_1$ | $p_2$ | $d$ |
| 42 | 10 | 12 |

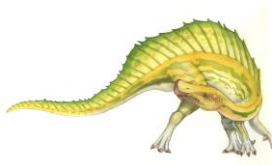
| 2nd outer page | outer page | inner page | offset |
|---|---|---|---|
| $p_1$ | $p_2$ | $p_3$ | $d$ |
| 32 | 10 | 10 | 12 |





# Hashed Page Tables

- Common in address spaces > 32 bits
- The virtual page number is hashed into a page table
  - This page 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 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)

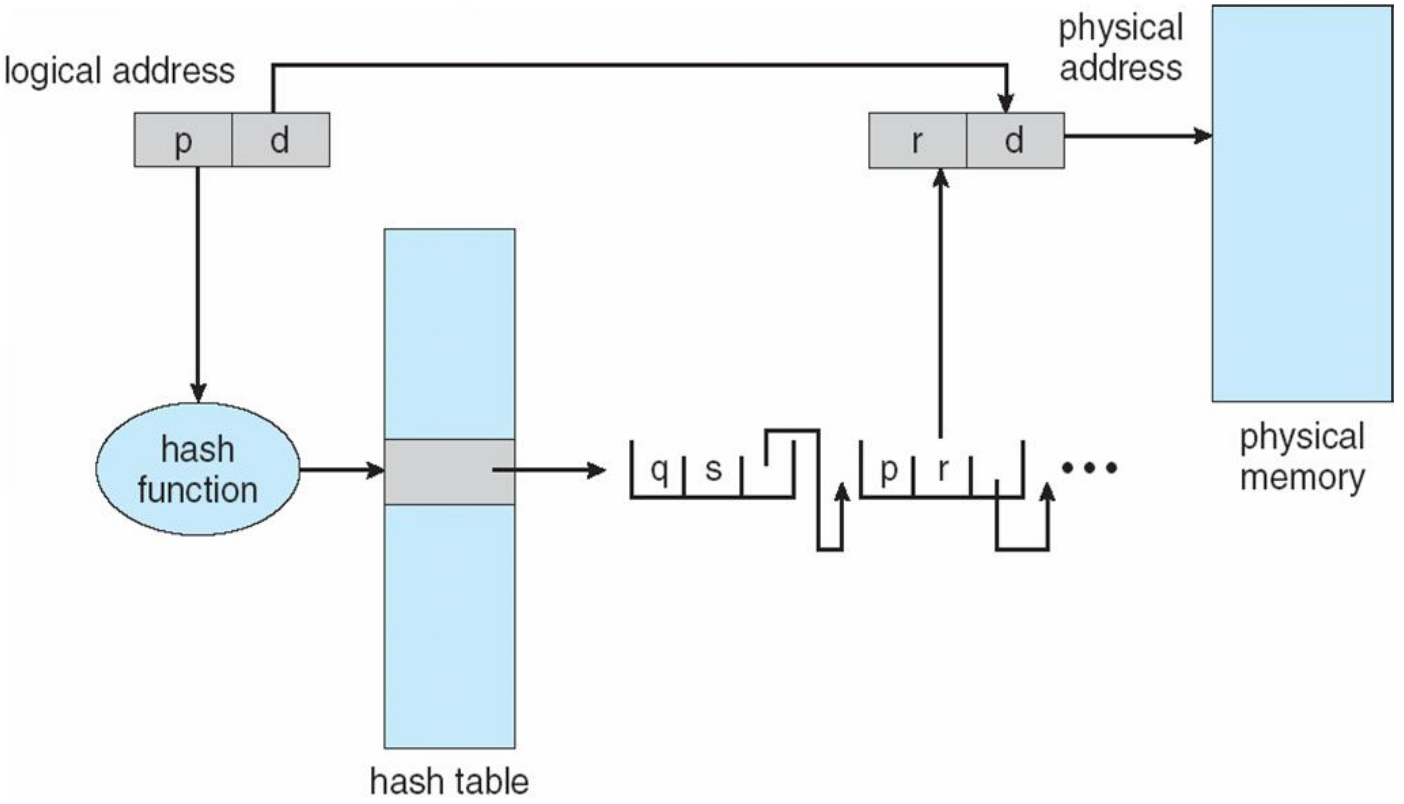


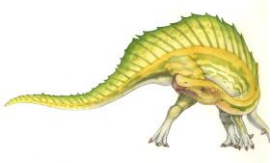


Troy University Resources



# Hashed Page Table





# Inverted Page Tables

- An inverted table indexes physical frames instead of allocating an entry for every possible virtual page of every process.
- A basic entry records a process or address-space identifier and the virtual page occupying that frame.
- A hash structure and the TLB can accelerate lookup by address space and virtual page.
- Shared frames can have multiple virtual aliases. The implementation needs extra information to represent those mappings.
- A simple one-entry-per-frame description alone is insufficient to express all aliases.

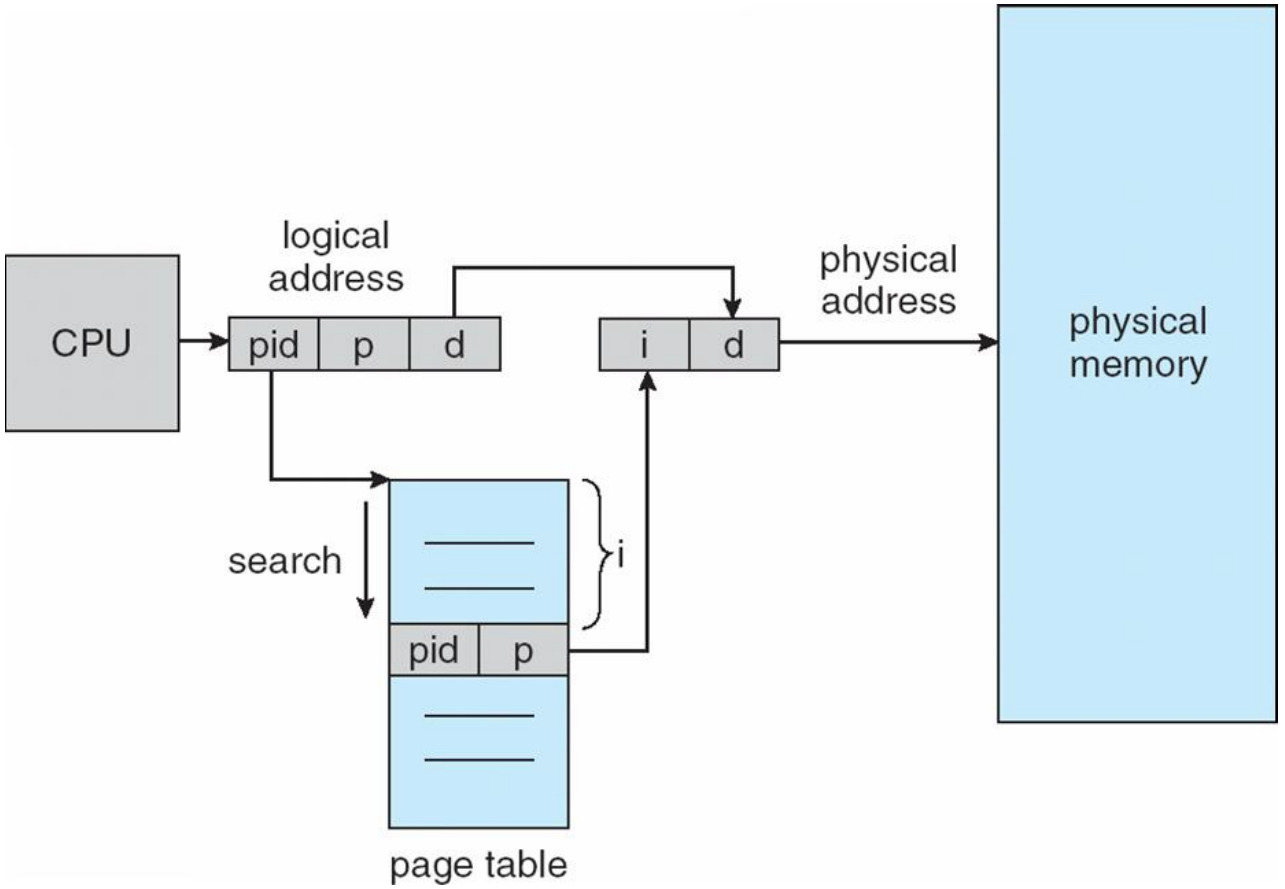


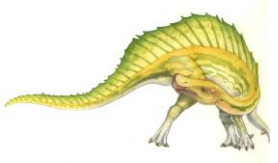


Troy University Resources



# Inverted Page Table Architecture





# Oracle SPARC Solaris

- Historical SPARC/Solaris example with tightly integrated hardware and software
  - Goals are efficiency, low overhead
- Based on hashing, but more complex
- Two hash tables
  - One kernel and one for all user processes
  - Each maps memory addresses from virtual to physical memory
  - Each entry represents a contiguous area of mapped virtual memory,
    - More efficient than having a separate hash-table entry for each page
  - Each entry has base address and span (indicating the number of pages the entry represents)



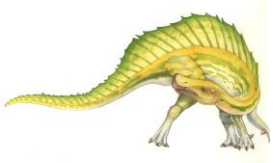


Troy University Resources



# SPARC/Solaris Translation Example

- The TLB caches translation table entries (TTEs) for fast lookups.
- A translation storage buffer (TSB) caches recently used TTEs in memory.
- On a TLB miss, hardware or kernel software consults the TSB, depending on the SPARC implementation.
- If no suitable cached TTE exists, the kernel finds the mapping in its address-space structures and refills translation caches.
- The original memory reference resumes after a valid translation is installed.





# Whole-Process Swapping

- In the classical model, the OS can move a process's memory image to backing storage and later bring it back.
- Transfer cost grows with the number of bytes moved, plus device and scheduling overhead.
- A swapped-out process is not immediately dispatchable until the required state is resident again.
- Modern virtual-memory systems commonly reclaim or swap individual pages instead of moving the complete process image.



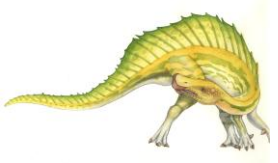


Troy University Resources



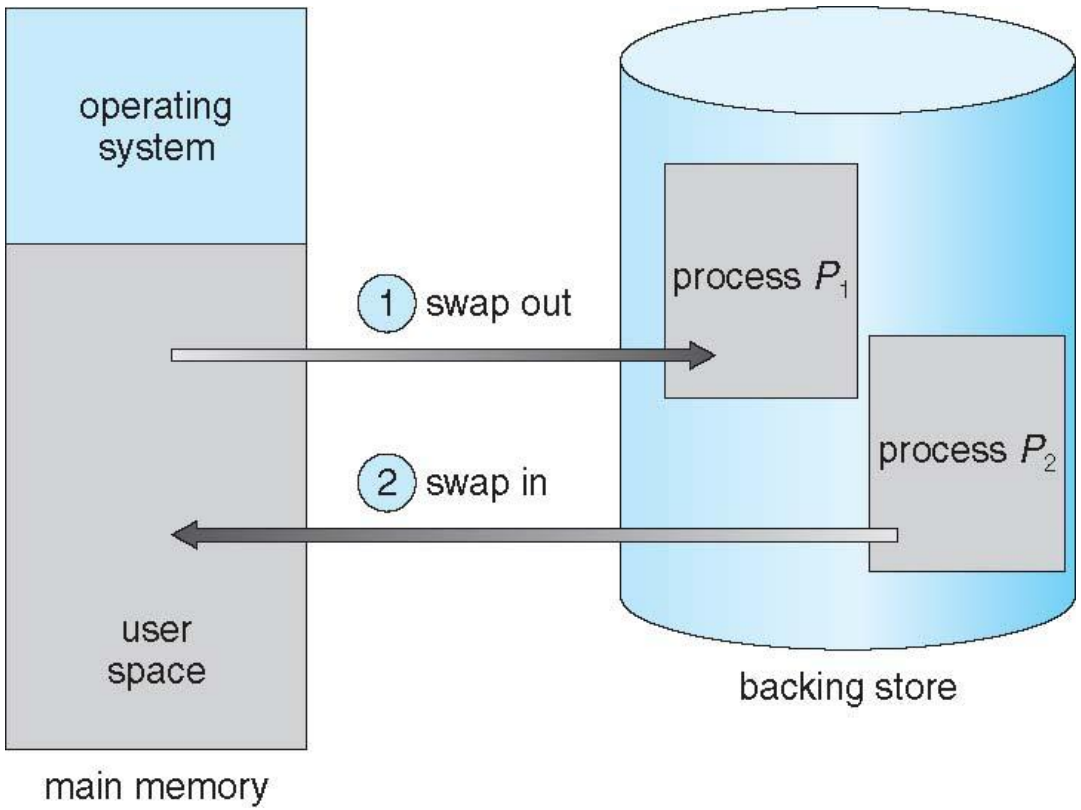
# Swapping and Relocation

- A process can return to different physical addresses if its relocation or translation mechanism supports the new mapping.
- Pinned pages and outstanding I/O constrain which physical memory can be moved or reclaimed.
- Modern systems balance page reclaim, writeback, compression, and swapping according to policy and memory pressure.
- There is no universal rule that swapping stays disabled until one fixed threshold is crossed.





# Schematic View of Swapping

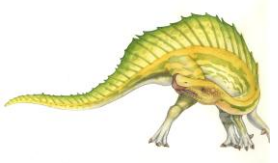







# Illustrative Whole-Process Swap Cost

- Assume a 100 MB image and a sustained transfer rate of 50 MB/s.
- Swapping out the full image takes 100 / 50 = 2 seconds.
- Swapping in another equally large image adds 2 seconds. Transfer time alone is 4 seconds.
- This excludes queueing, device latency, and software overhead.
- Ordinary CPU context switches do not inherently copy a process's complete memory image.





# Swapping and I/O

- A device performing DMA must continue accessing the intended physical pages until I/O completes.
- An OS can pin the required pages, use kernel buffers, or use another supported mapping mechanism.
- Page-level reclaim avoids the cost of whole-process swapping when only some pages are cold.
- The timing of reclaim and swap is an OS policy, rather than a fixed rule for all modern systems.



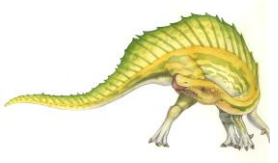


Troy University Resources



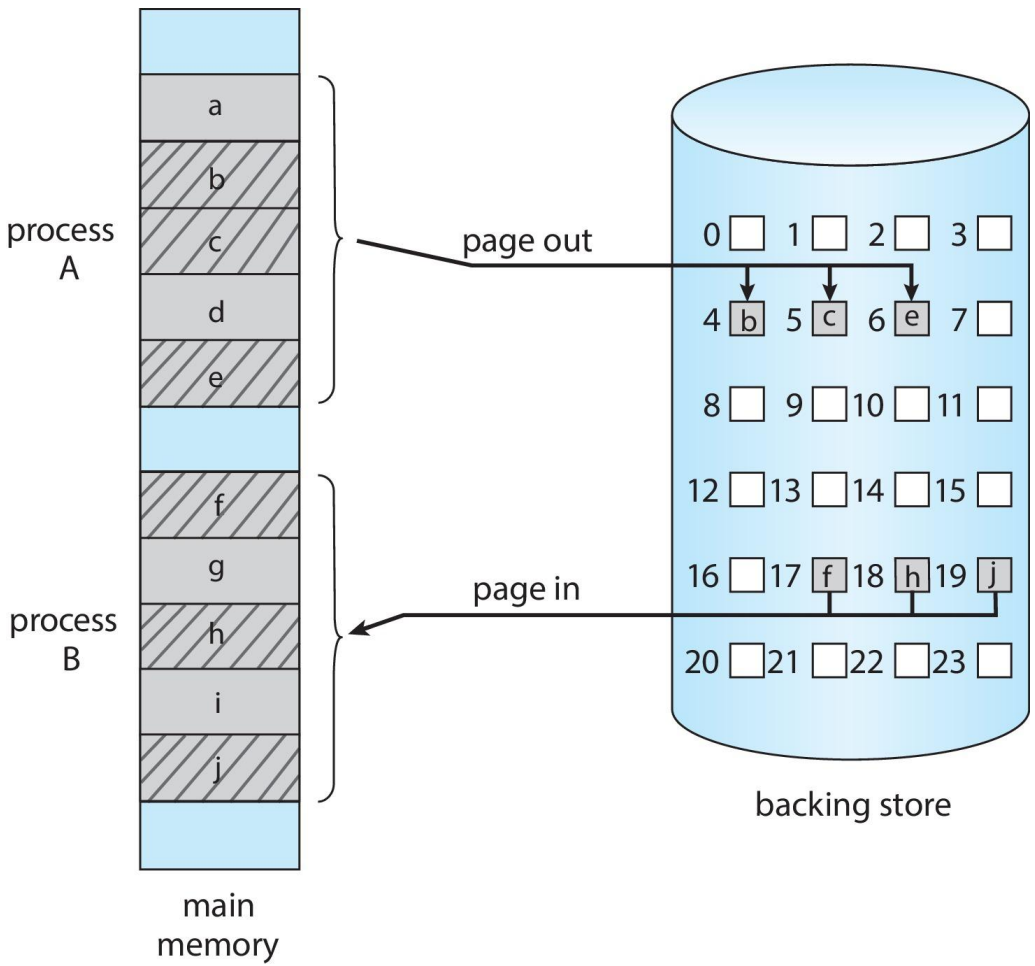
# Memory Pressure on Mobile Systems

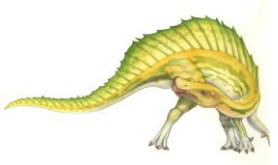
- Mobile systems reclaim clean file-backed pages and can reload them from storage later.
- Android can compress anonymous and private dirty pages into zRAM, a RAM-backed swap device.
- If reclaim is insufficient, Android can terminate processes. Applications must manage their own saved state; termination does not guarantee an automatic state snapshot.
- iOS uses memory-pressure notifications and can terminate applications when necessary.
- Exact policies depend on the OS version and device. “Mobile systems never swap” is too broad.





# Swapping with Paging



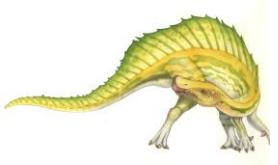


Troy University Resources



# IA-32, Intel 64, and IA-64

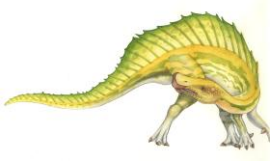
- IA-32 is the 32-bit x86 architecture used by the Pentium family and compatible processors.
- Intel 64 and AMD64 are implementations of x86-64, which extends x86 with 64-bit operation.
- IA-64 is the separate Itanium architecture. It is not the name of 64-bit x86.
- The next slides first illustrate IA-32 segmentation and paging, then x86-64 address translation.





# 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
  - 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**))



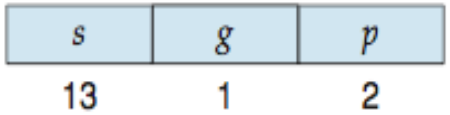


Troy University Resources

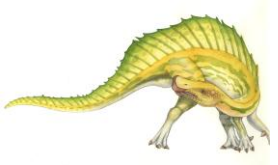


# Example: The Intel IA-32 Architecture (Cont.)

- CPU generates logical address
  - Selector given to segmentation unit
    - Which produces linear addresses

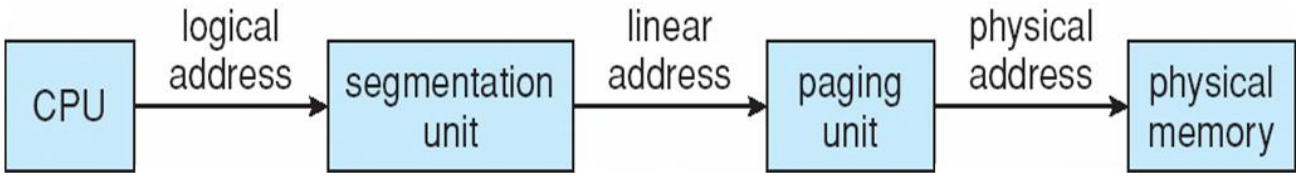


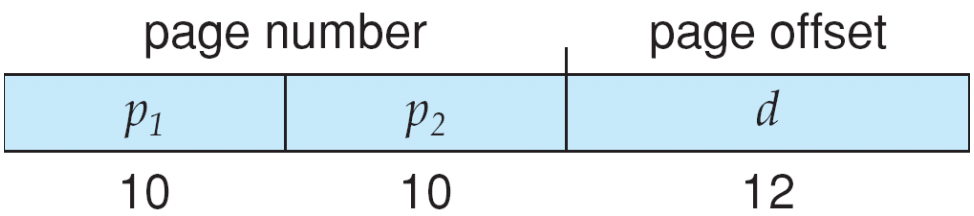

  - Linear address given to paging unit
    - Which generates physical address in main memory
    - Paging units form equivalent of MMU
    - Pages sizes can be 4 KB or 4 MB

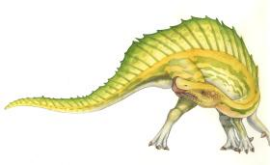




# Logical to Physical Address Translation in IA-32



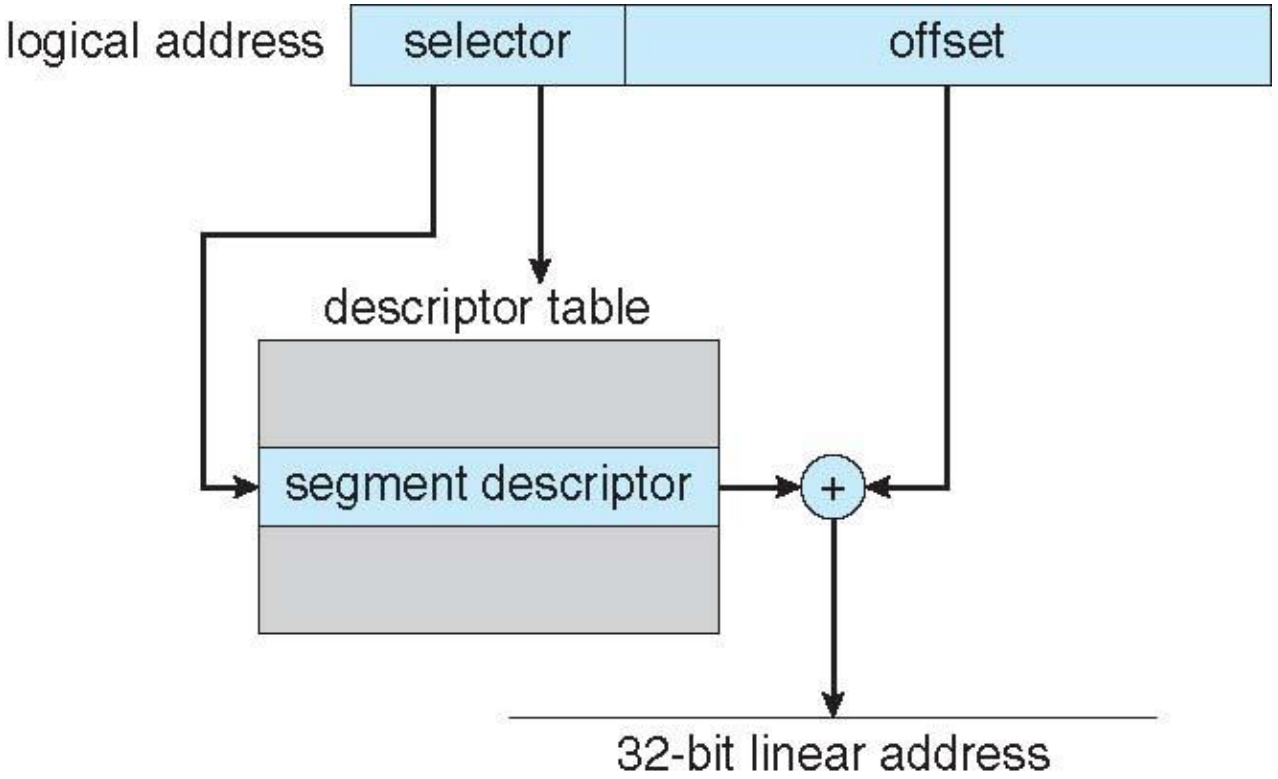



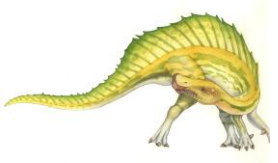


Troy University Resources



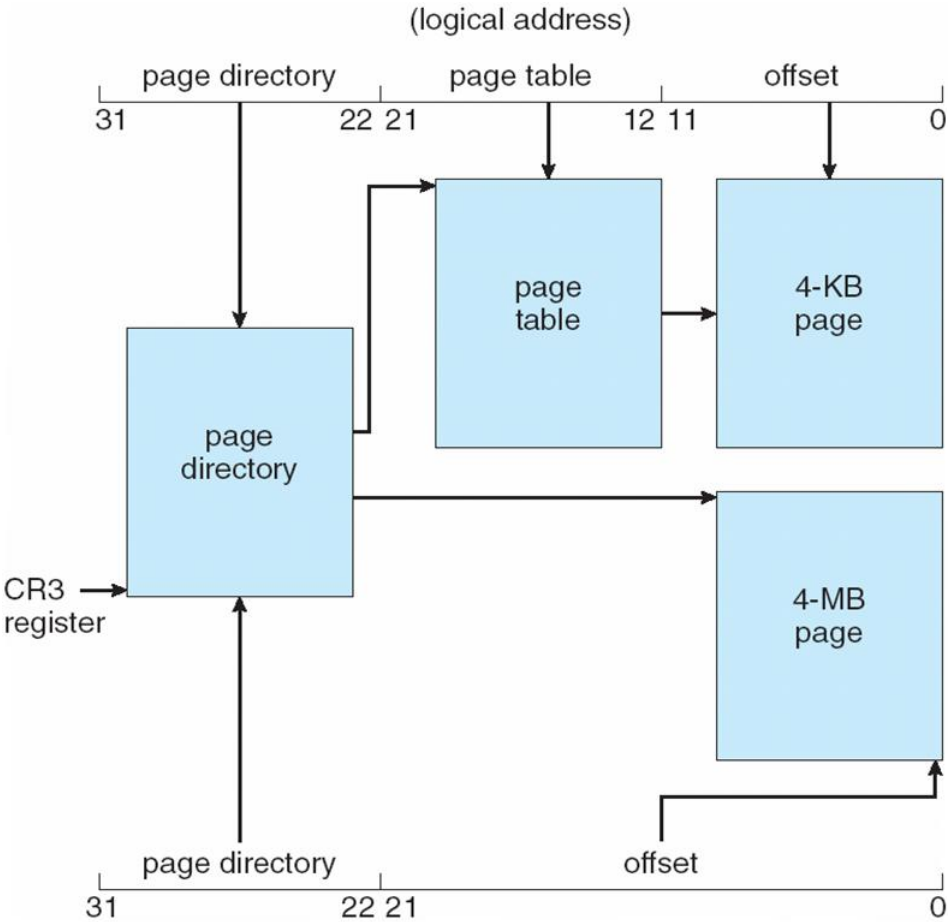
# Intel IA-32 Segmentation

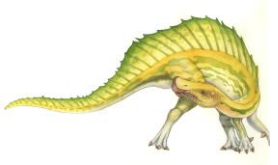




# Intel IA-32 Paging Architecture

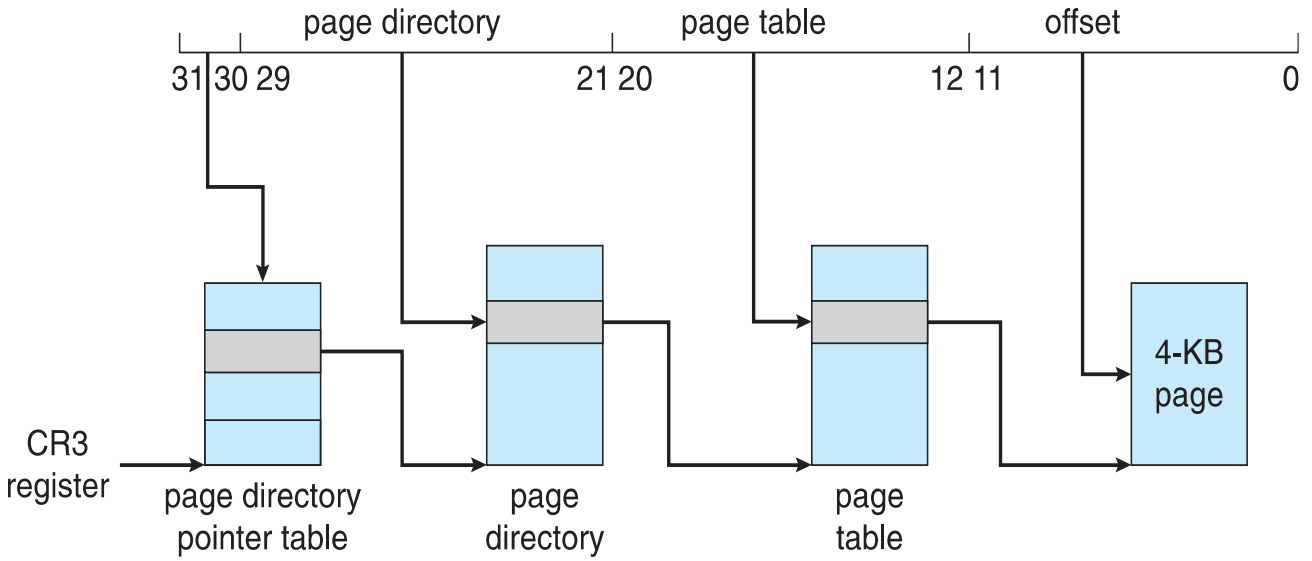


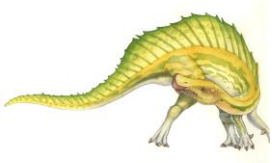





# IA-32 Physical Address Extension

- PAE extends physical addressing beyond 4 GiB.
- A 32-bit process still has a 32-bit virtual address space.
- With 4 KiB pages, translation uses a 2-bit PDPT index, two 9-bit indices, and a 12-bit offset.
- Page-directory and page-table entries are 64 bits. The classic physical limit is 36 bits, or 64 GiB.

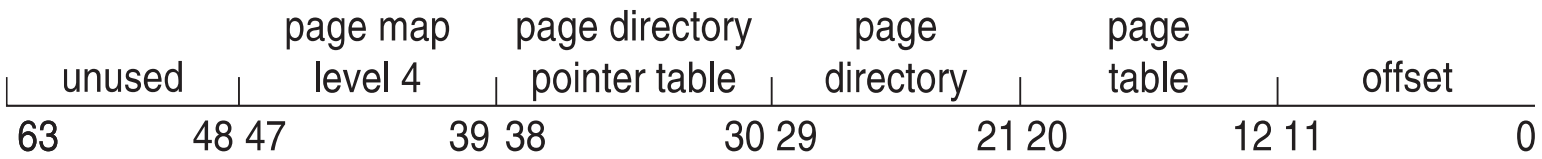


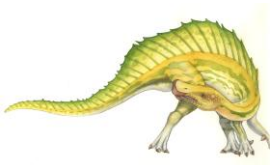


# x86-64 Paging

- 64-bit registers do not imply 64 implemented virtual-address bits.
- Four-level paging commonly uses 48 virtual-address bits. Five-level paging can use 57 bits when supported and enabled.
- The figure shows the four-level split: 9 + 9 + 9 + 9 + 12 bits.
- Large pages can be 2 MiB or 1 GiB where supported. Physical-address width is a separate hardware property.




Page-table entries are 64 bits. x86-64 paging is distinct from IA-32 PAE mode.



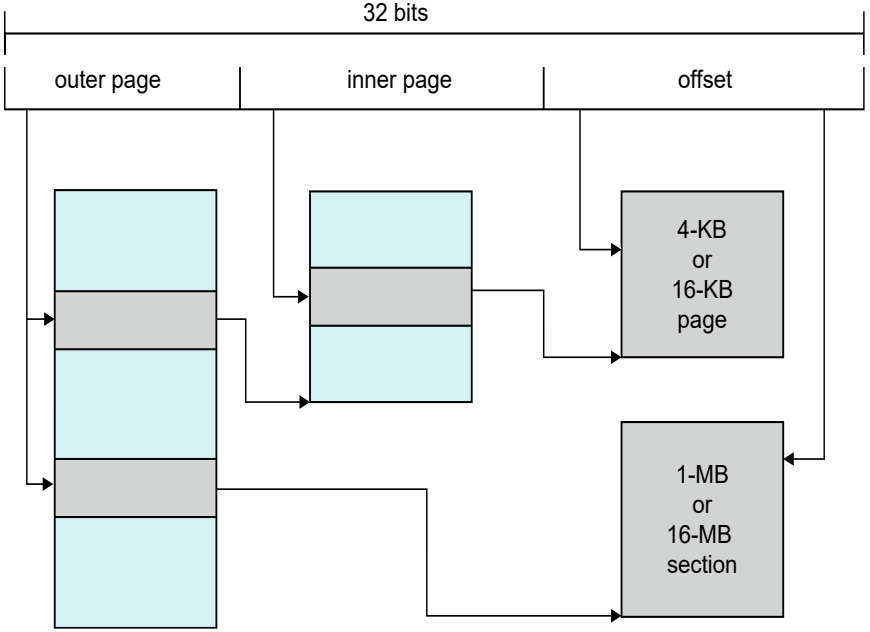


Troy University Resources



# ARM Paging: AArch32 Example

- The figure illustrates a classic 32-bit ARM translation scheme.
- Small pages use two translation levels. Sections map larger regions.
- Page sizes, sections, and TLB structure depend on the ARM architecture version.




AArch64 uses a different translation format. Its supported granules can include 4 KiB, 16 KiB, and 64 KiB, depending on the implementation.



# End of Chapter 9

