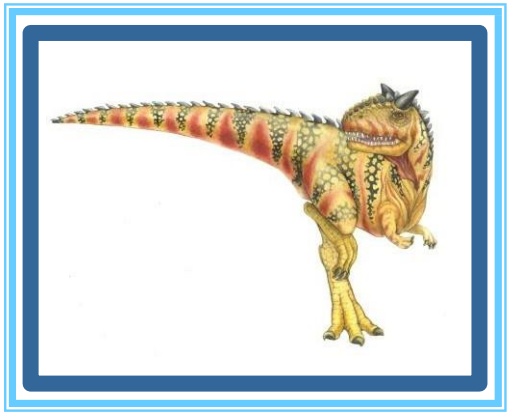
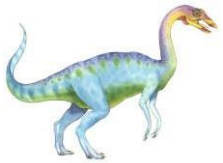
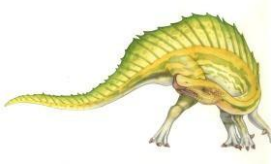
Main Memory : Part -2

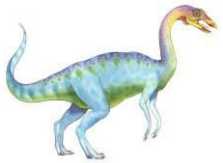




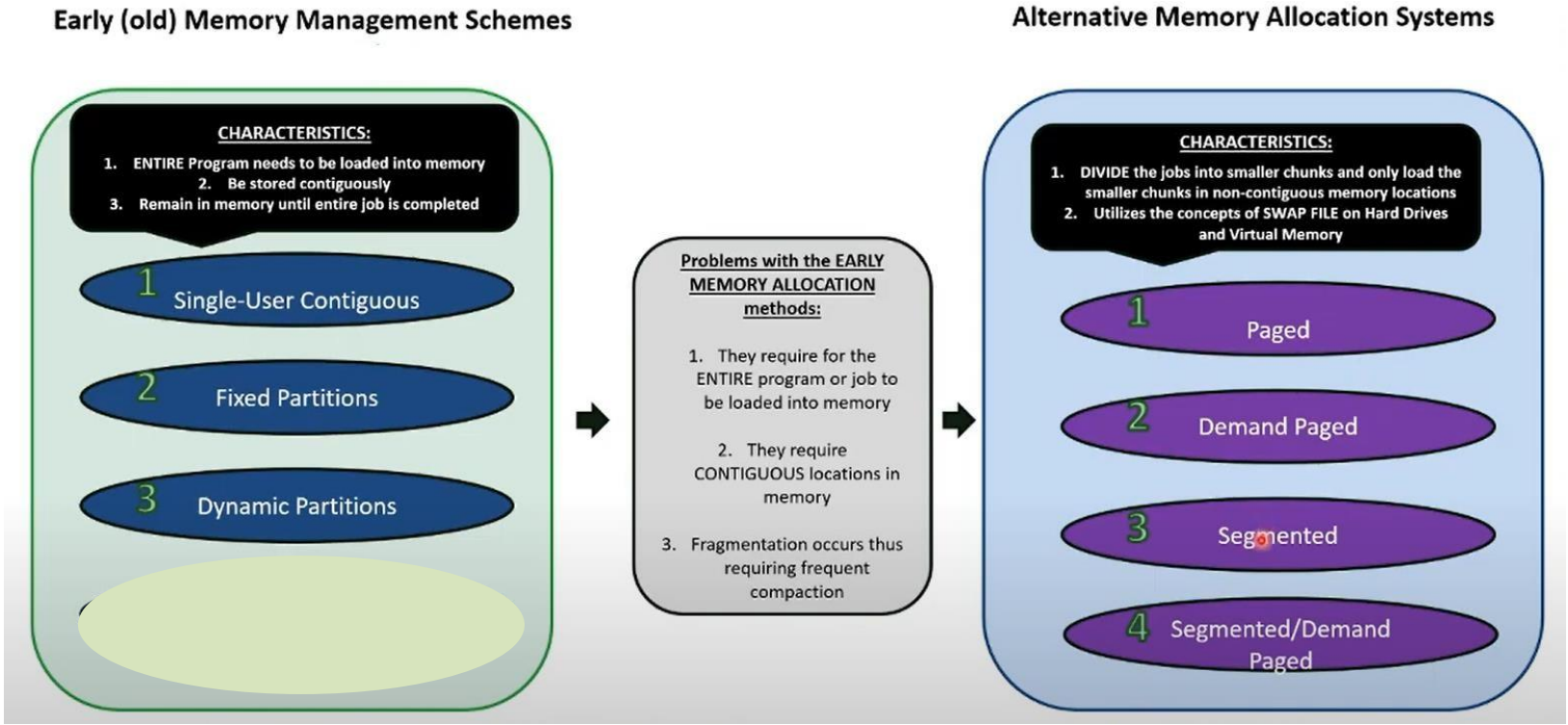
Today's Lecture

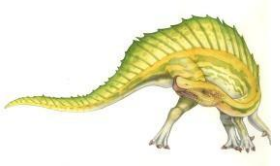
- Background
- Address binding
- Logical vs physical address
- Memory Management Schemes
- Advances Schemes
- Paging
- Structure of the Page Table
- Swapping
- Example: The Intel 32 and 64-bit Architectures
- Example: ARMv8 Architecture

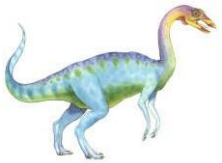




Recap


Early (old) Memory Management Schemes
CHARACTERISTICS:
1. ENTIRE Program needs to be loaded into memory
2. Be stored contiguously
3. Remain in memory until entire job is completed
1 Single-User Contiguous
2 Fixed Partitions
3 Dynamic Partitions
Problems with the EARLY MEMORY ALLOCATION methods:
1. They require for the ENTIRE program or job to be loaded into memory
2. They require CONTIGUOUS locations in memory
3. Fragmentation occurs thus requiring frequent compaction
Alternative Memory Allocation Systems
CHARACTERISTICS:
1. DIVIDE the jobs into smaller chunks and only load the smaller chunks in non-contiguous memory locations
2. Utilizes the concepts of SWAP FILE on Hard Drives and Virtual Memory
1 Paged
2 Demand Paged
3 Segmented
4 Segmented/Demand Paged





Important Definitions

1 – Programs are typically stored on disk (permanent storage)

2 – The Operating System is responsible for loading program into main memory

3 – CPU executes the instructions loaded into memory in previous step

Page

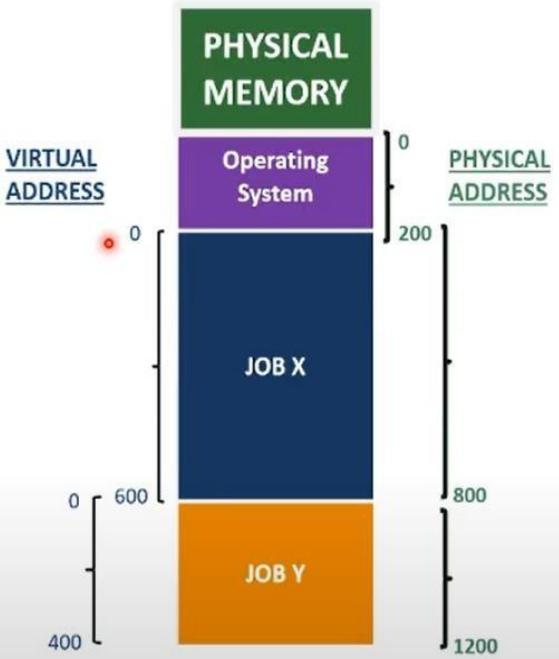
A fixed-size section of a user's job that corresponds in size to the page frames in physical memory

Page Frames

Equal-sized sections or chunks located in physical memory

Segment

A portion of memory assigned to a logical grouping of code in a job or process



Physical Address

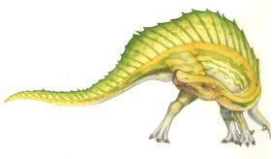
A real address in physical memory or RAM

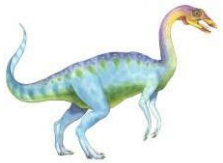
Virtual Address

A logical address relative to the start of a process's address space

Virtual Memory

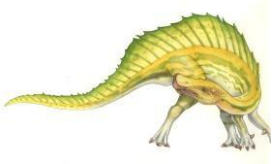
A technique that allows programs to be executed even though they are not stored entirely in physical memory

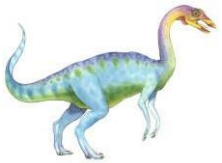




Paging

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a **page table** to translate logical to physical addresses
- Backing store likewise split into pages
- Still have Internal fragmentation





Paging

The role of the Operating System Memory Manager is critical in ensuring the appropriate allocation of memory to a Program.

FIRST … it identifies the number of pages that are part of a given Program

SECOND …identifies the same number of available or empty page frames in main memory for all pages to be loaded

THIRD .. It manages (moves, removes, reallocates) pages in memory

Microsoft Excel

Page 1
Page 2
Page 3

There is no need for contiguous memory space in physical memory

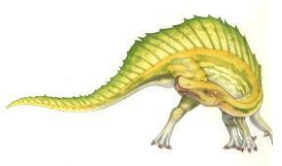
PHYSICAL MEMORY

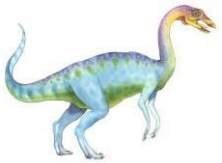
Page Frame 1

Page Frame 2

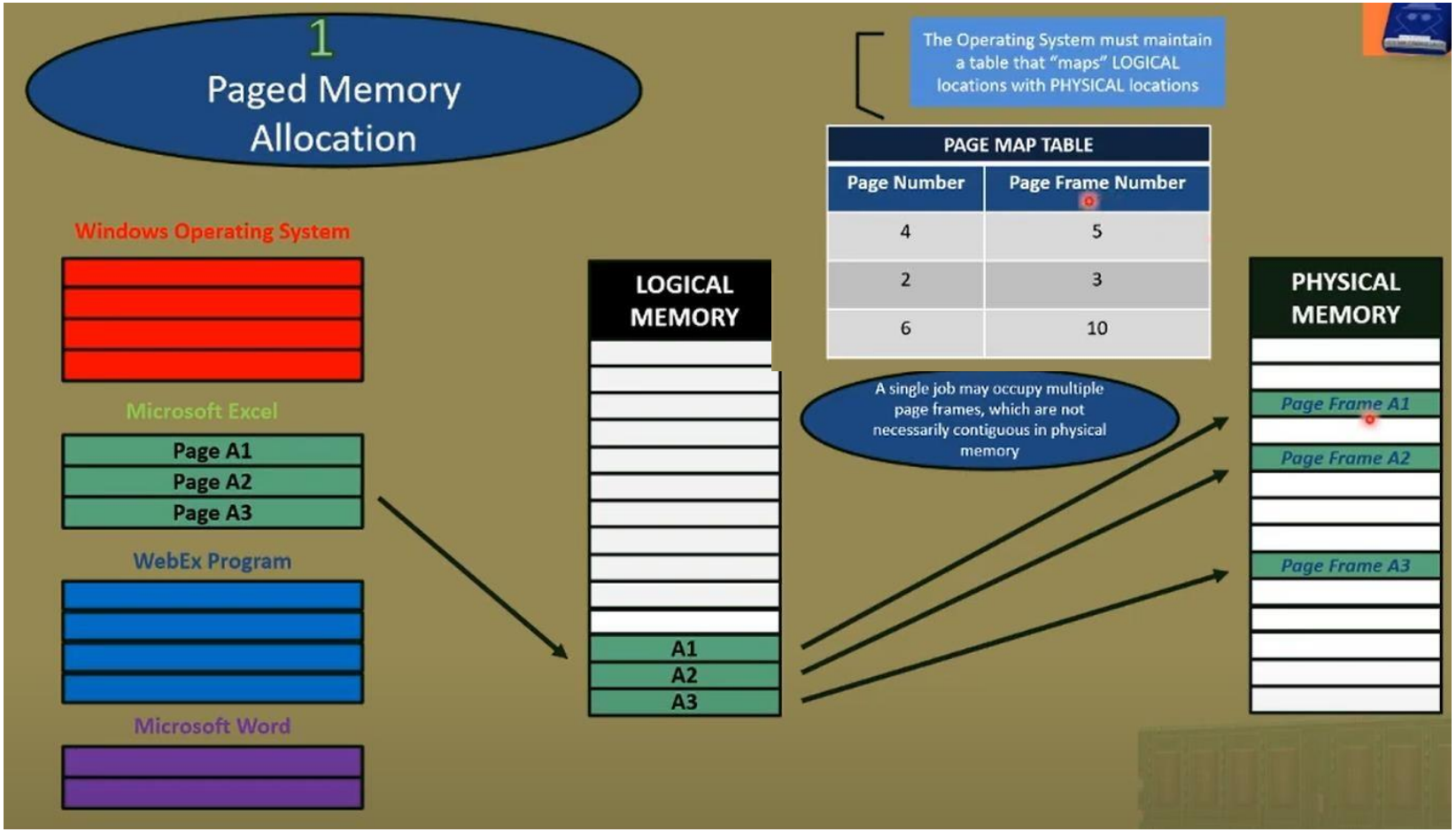
Page Frame 3

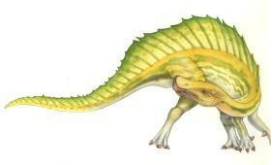
Memory is split up into small, equal-sized sections called **PAGE FRAMES**

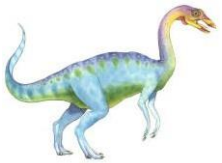




Paging


1
Paged Memory Allocation
The Operating System must maintain a table that "maps" LOGICAL locations with PHYSICAL locations
PAGE MAP TABLE
Page Number
Page Frame Number
4
5
2
3
6
10
Windows Operating System
Microsoft Excel
Page A1
Page A2
Page A3
WebEx Program
Microsoft Word
LOGICAL MEMORY
A1
A2
A3
A single job may occupy multiple page frames, which are not necessarily contiguous in physical memory
PHYSICAL MEMORY
Page Frame A1
Page Frame A2
Page Frame A3





CHARACTERISTICS:

1. Significant OVERHEAD. The Operating system must keep track of memory allocation … which means keeping track of the page frame in memory in which the page is located ….

2. The Memory Management module of the Operating System fulfills this responsibility by using three tables:

JOB TABLE

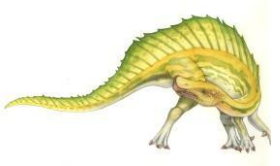
Job Size	Page Map Table Location
300	2180
100	2420
600	2084

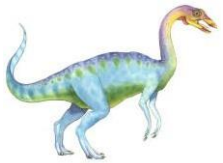
PAGE MAP TABLE

Page Number	Page Frame Number
4	5
2	3
6	10

MEMORY MAP TABLE

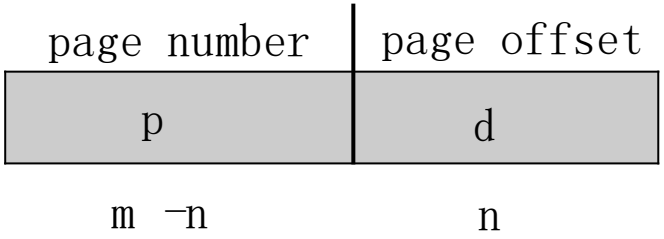
Page Frame	State
10	Busy
3	Busy
5	Free



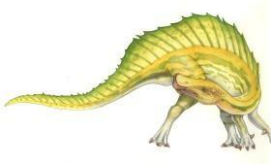


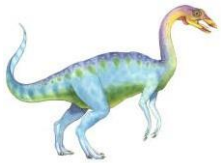
Address Translation Scheme

- 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

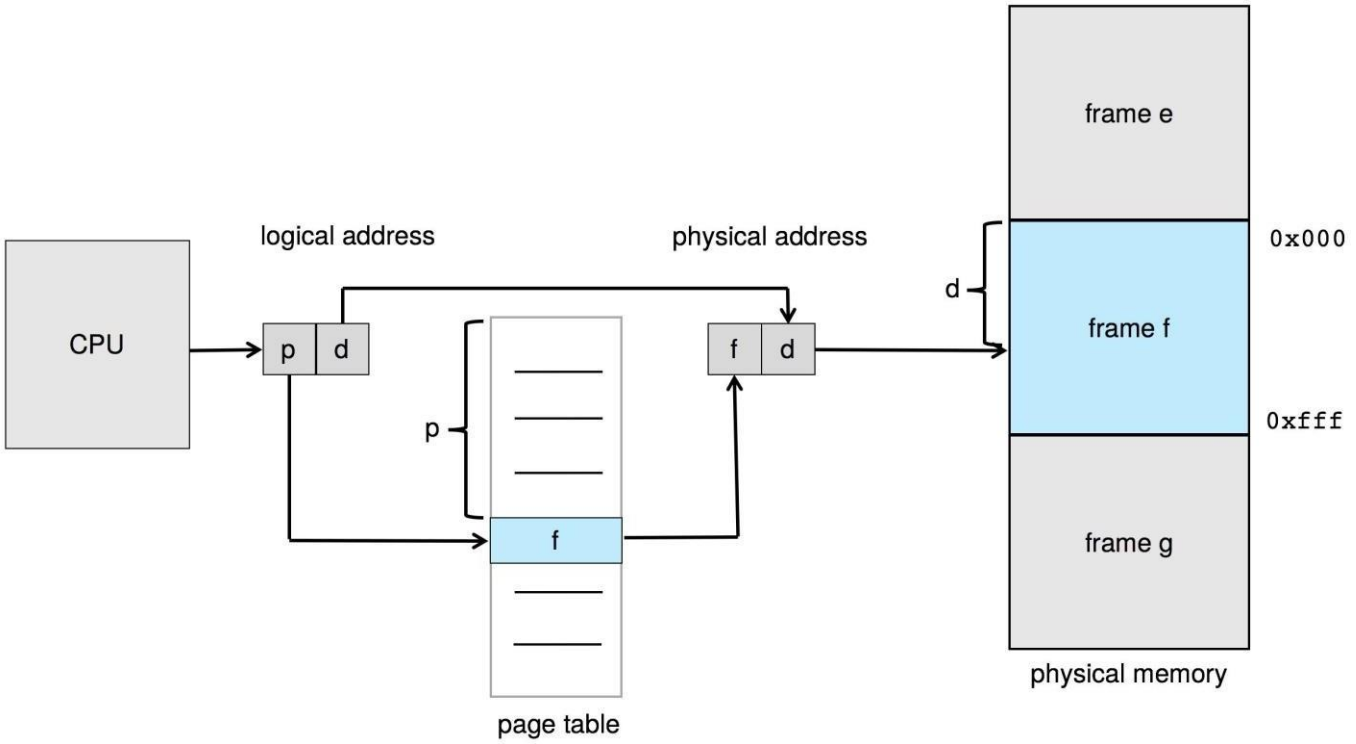


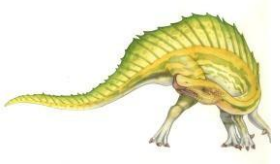
  - For given logical address space 2^m and page size 2^n

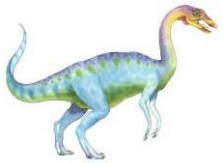




Paging Hardware







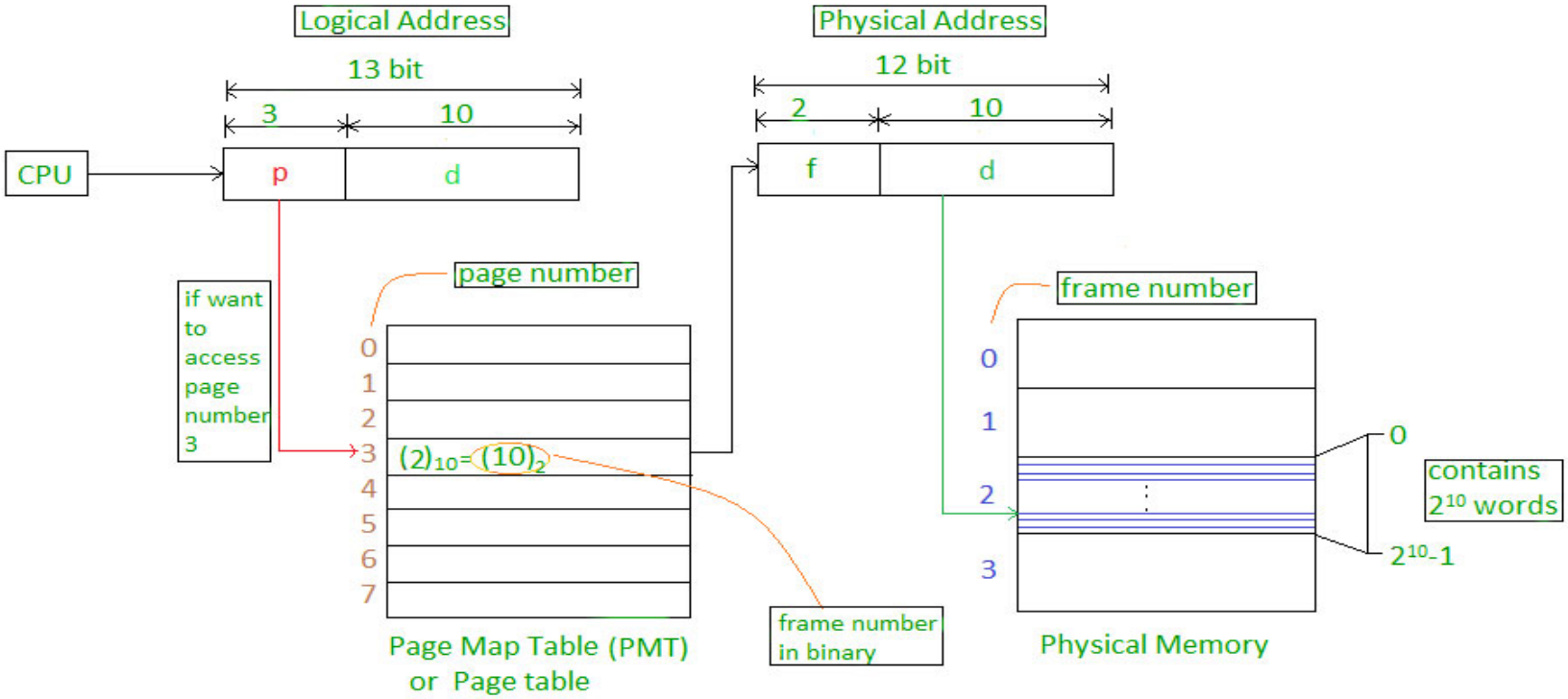
Example

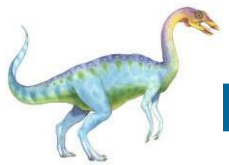
Let us consider an example:

- Physical Address = 12 bits, then Physical Address Space = 4 K words
- Logical Address = 13 bits, then Logical Address Space = 8 K words
- Page size = frame size = 1 K words (assumption)

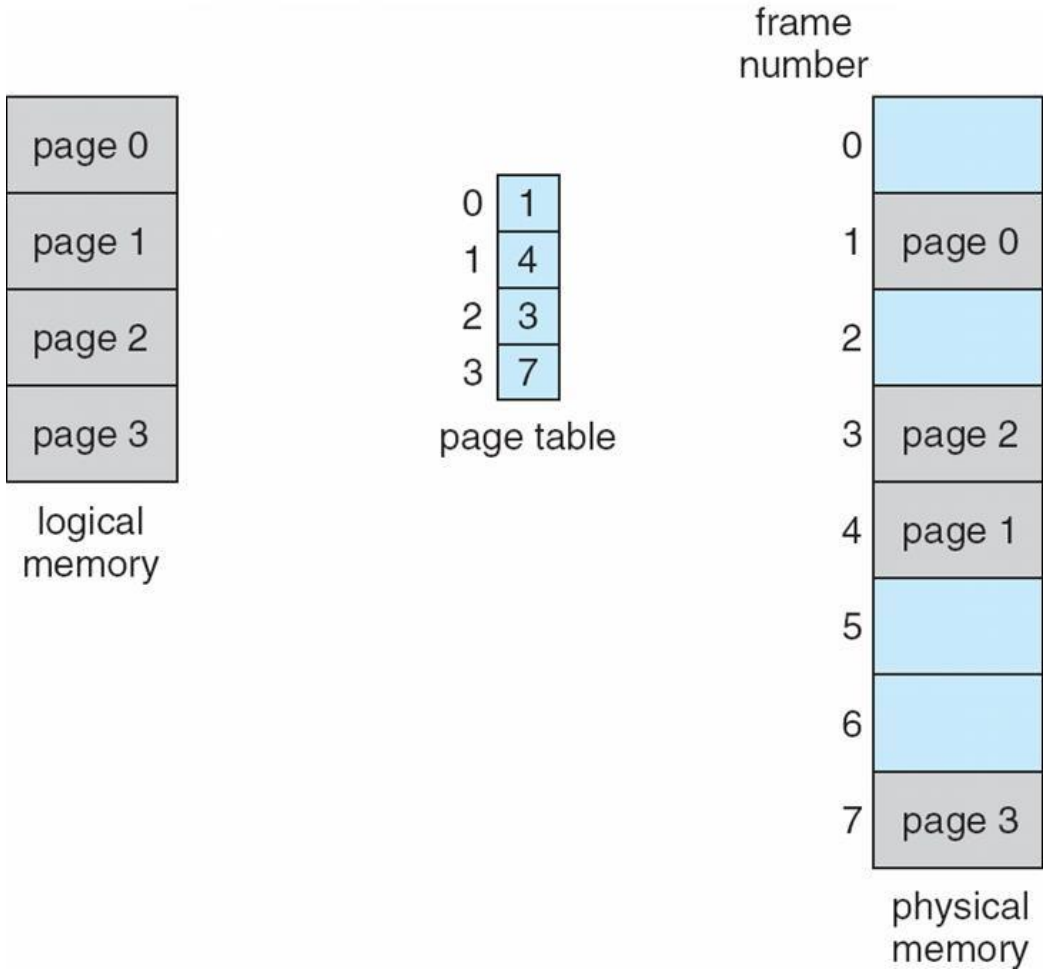
Number of frames = Physical Address Space / Frame size = 4 K / 1 K = 4 = 2^2

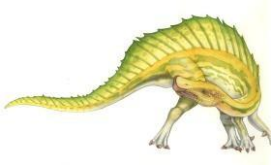
Number of pages = Logical Address Space / Page size = 8 K / 1 K = 8 = 2^3





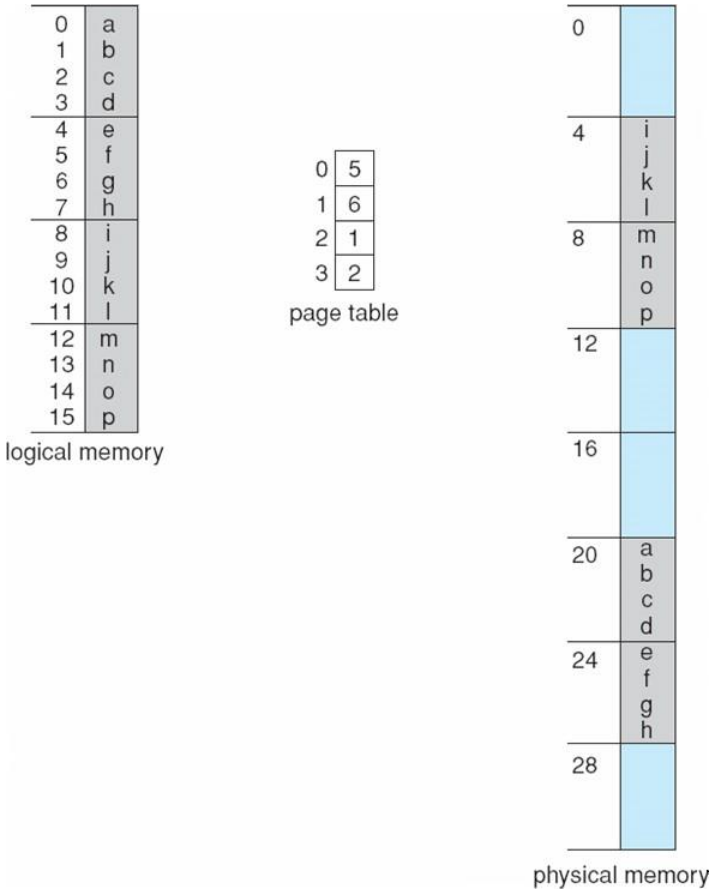
Paging Model of Logical and Physical Memory



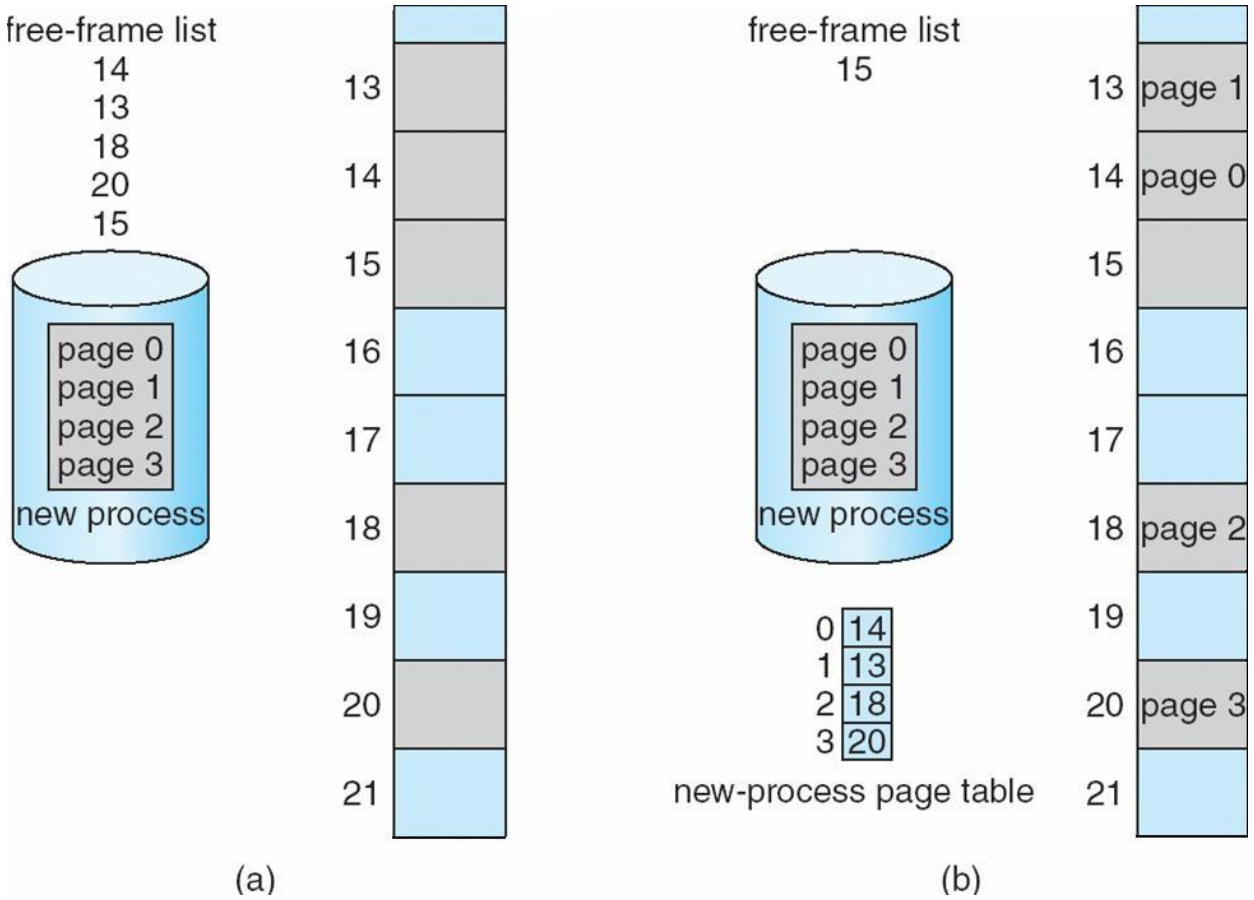


Paging Example

- Logical address: n = 2 and m = 4. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages)

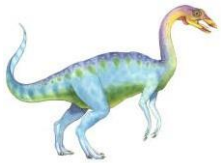


Free Frames



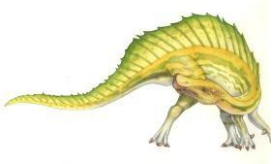
Before allocation

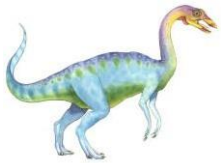
After allocation



Implementation of Page Table

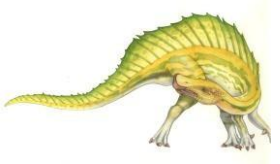
- Page table is kept in main memory
  - **Page-table base register** (**PTBR**) points to the page table
  - **Page-table length register** (**PTLR**) indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses
  - One for the page table and one for the data / instruction
- The two-memory access problem can be solved by the use of a special fast-lookup hardware cache called **translation look-aside buffers** (**TLBs**) (also called **associative memory**).

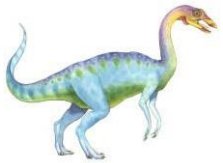




Translation Look-Aside Buffer

- Some TLBs store **address-space identifiers** (**ASIDs**) in each TLB entry – uniquely identifies each process to provide address-space protection for that process
  - Otherwise need to flush at every context switch
- TLBs typically small (64 to 1,024 entries)
- On a TLB miss, value is loaded into the TLB for faster access next time
  - Replacement policies must be considered
  - Some entries can be **wired down** for permanent fast access



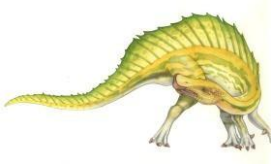


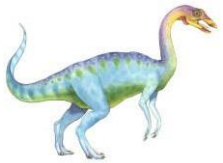
Hardware

- Associative memory – parallel search

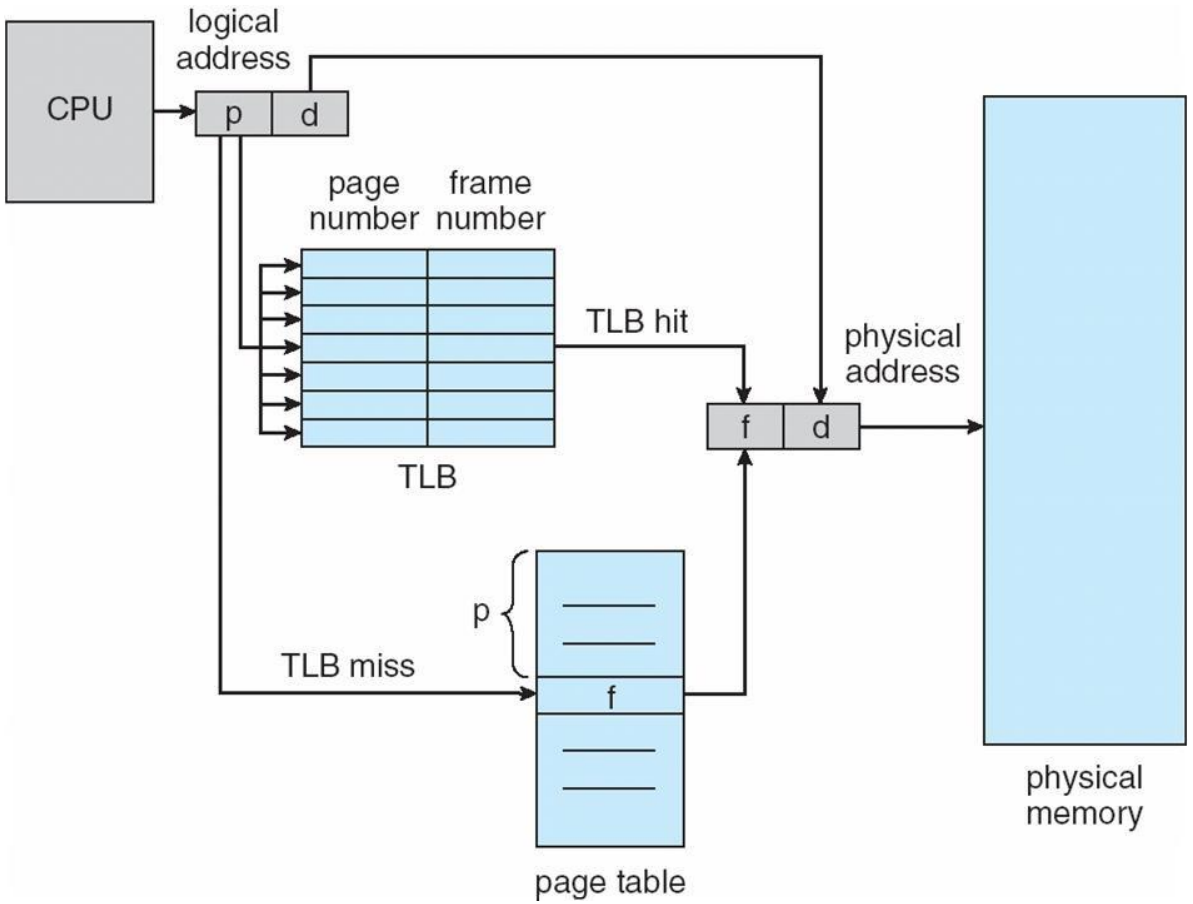
Page #	Frame #

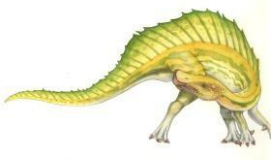
- Address translation (p, d)
  - If p is in associative register, get frame # out
  - Otherwise get frame # from page table in memory

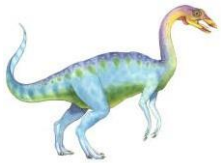




Paging Hardware With TLB

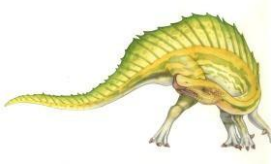


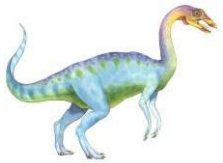




Effective Access Time

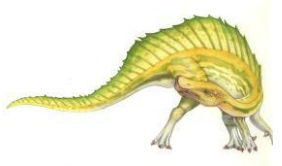
- Hit ratio – percentage of times that a page number is found in the TLB
- An 80% hit ratio means that we find the desired page number in the TLB 80% of the time.
- Suppose that 10 nanoseconds to access memory.
  - If we find the desired page in TLB then a mapped-memory access take 10 ns
  - Otherwise we need two memory access so it is 20 ns



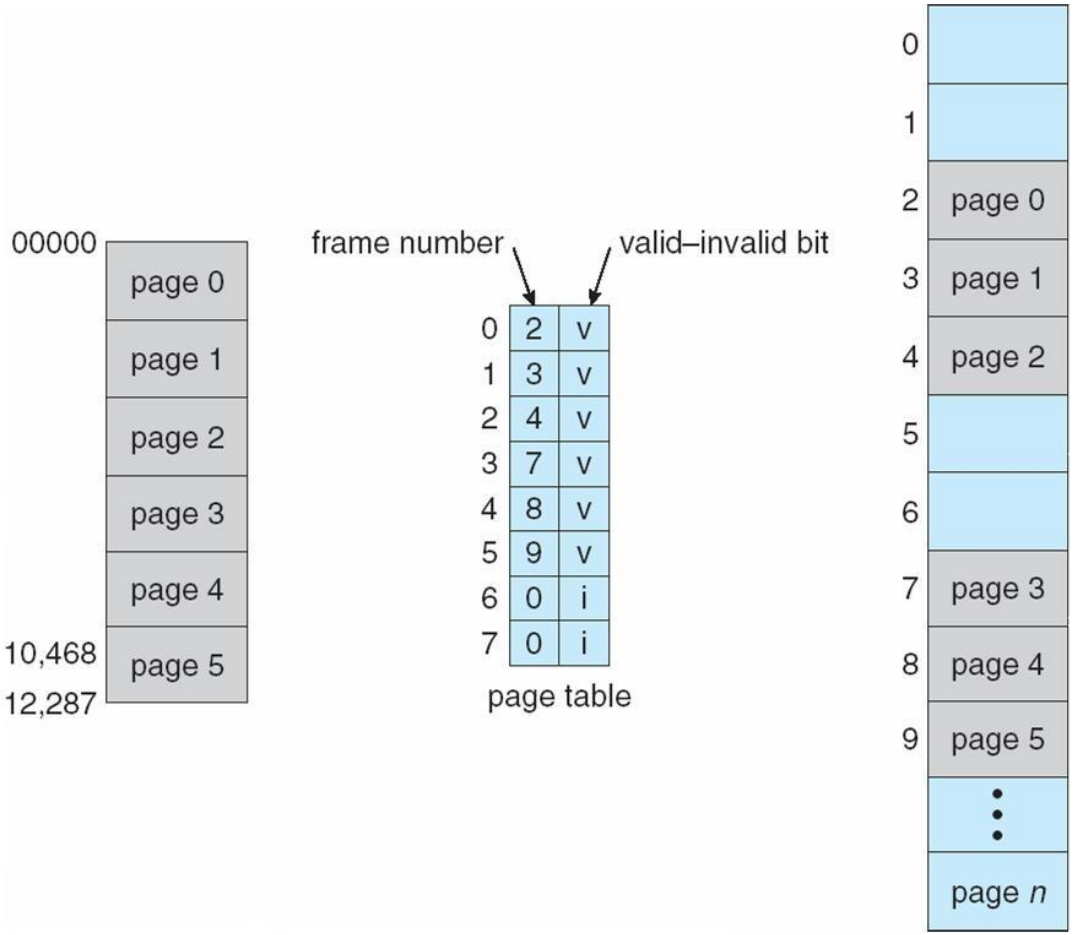


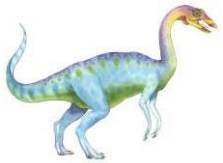
Memory Protection

- Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed
  - Can also add more bits to indicate page execute-only, and so on
- **Valid-invalid** bit attached to each entry in the page table:
  - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
  - "invalid" indicates that the page is not in the process' logical address space
  - Or use **page-table length register** (**PTLR**)
- Any violations result in a trap to the kernel



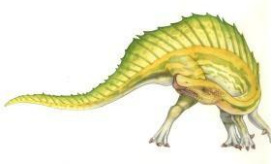
Valid (v) or Invalid (i) Bit In A Page Table

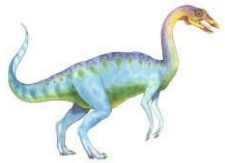




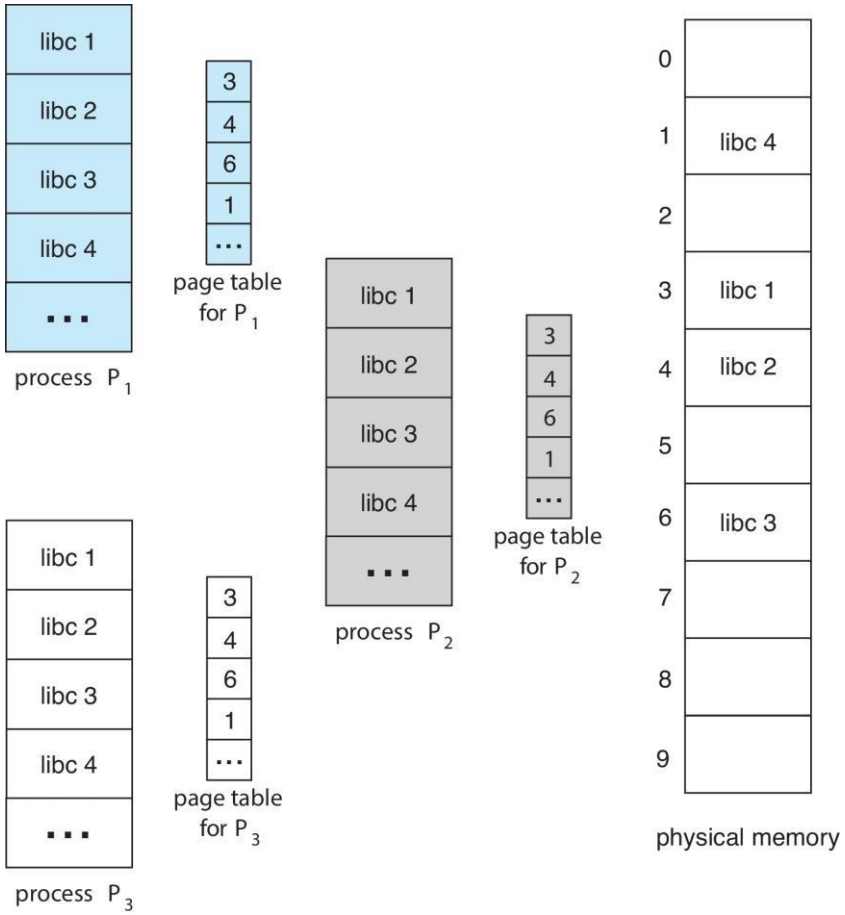
Shared Pages

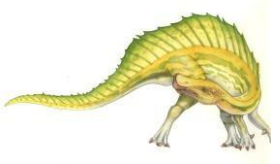
- **Shared code**
  - One copy of read-only (**reentrant**) code shared among processes (i.e., text editors, compilers, window systems)
  - Similar to multiple threads sharing the same process space
  - Also useful for interprocess communication if sharing of read-write pages is allowed
- **Private code 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

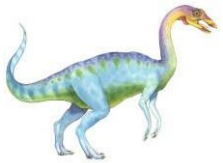




Shared Pages Example

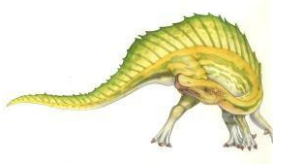


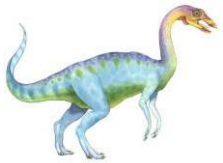




Structure of the Page Table

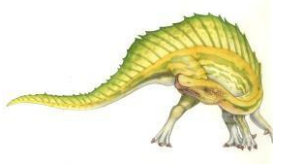
- Memory structures for paging can get huge using straight-forward methods
  - Consider a 32-bit logical address space as on modern computers
  - Page size of 4 KB (2^{12})
  - Page table would have 1 million entries ($2^{32} / 2^{12}$)
  - If each entry is 4 bytes ➔ each process 4 MB of physical address space for the page table alone
    - Don't want to allocate that contiguously in main memory
  - One simple solution is to divide the page table into smaller units
    - Hierarchical Paging
    - Hashed Page Tables
    - Inverted Page Tables

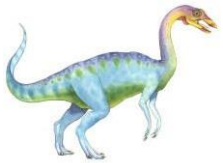




Swapping

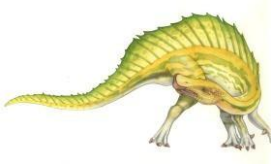
- A process can be **swapped** temporarily out of memory to a backing store, and then brought **back** into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for 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

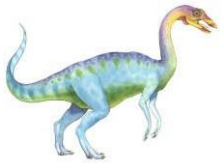




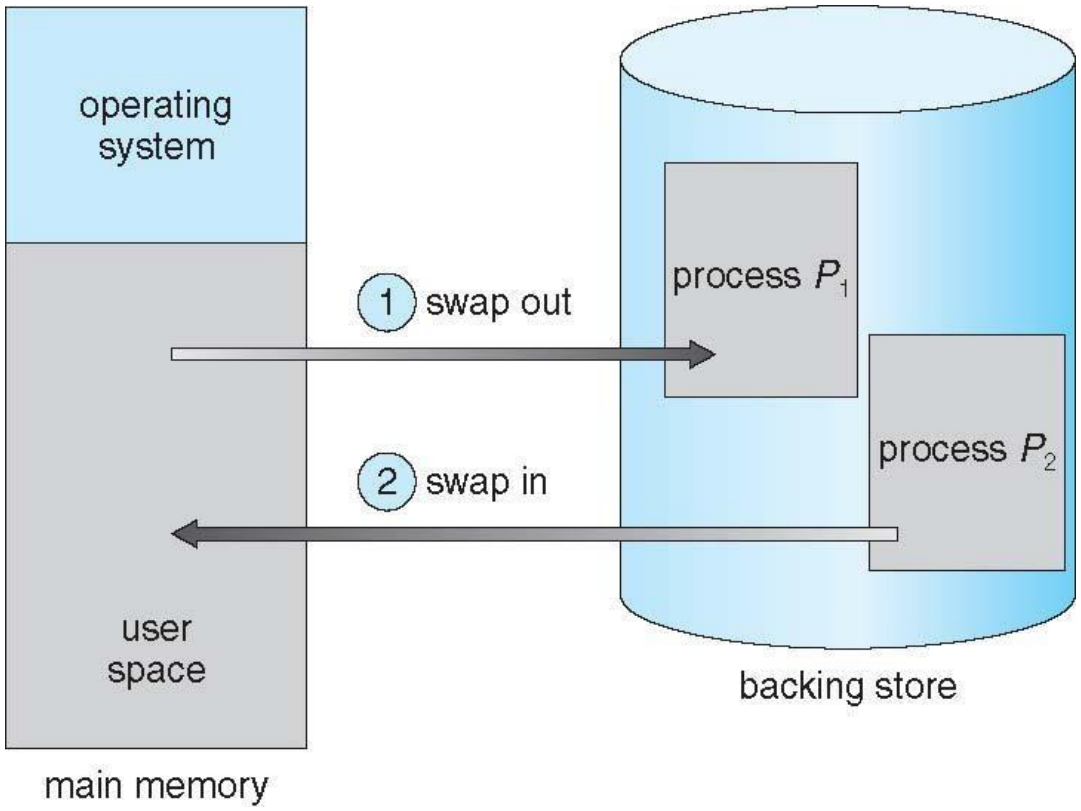
Swapping (Cont.)

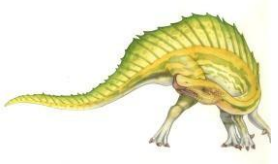
- Does the swapped out process need to swap back in to same physical addresses?
- Depends on address binding method
  - Plus consider pending I/O to / from process memory space
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - Swapping normally disabled
 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





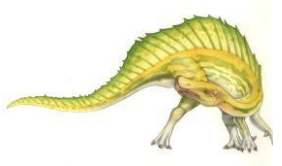
Schematic View of Swapping





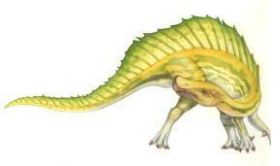
Context Switch Time including Swapping

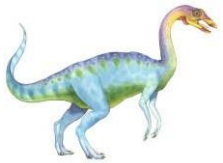
- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process
- Context switch time can then be very high
- 100MB process swapping to hard disk with transfer rate of 50MB/sec
  - Swap out time of 2000 ms
  - Plus swap in of same sized process
  - Total context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

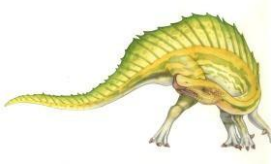
- Other constraints as well on swapping
  - Pending I/O – can't swap out as I/O would occur to wrong process
  - Or always transfer I/O to kernel space, then to I/O device
    - Known as **double buffering**, adds overhead
- Standard swapping not used in modern operating systems
  - But modified version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





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
  - Both OSes support paging as discussed below



Swapping with Paging