

Memory management

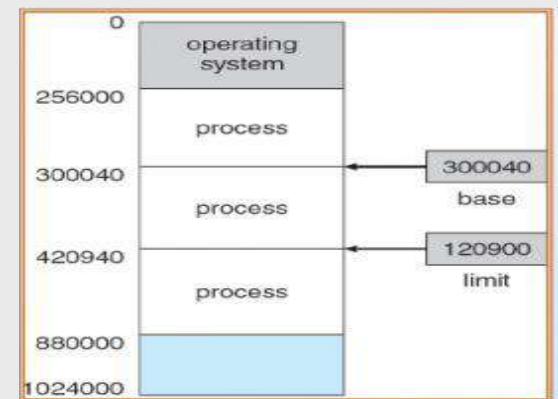
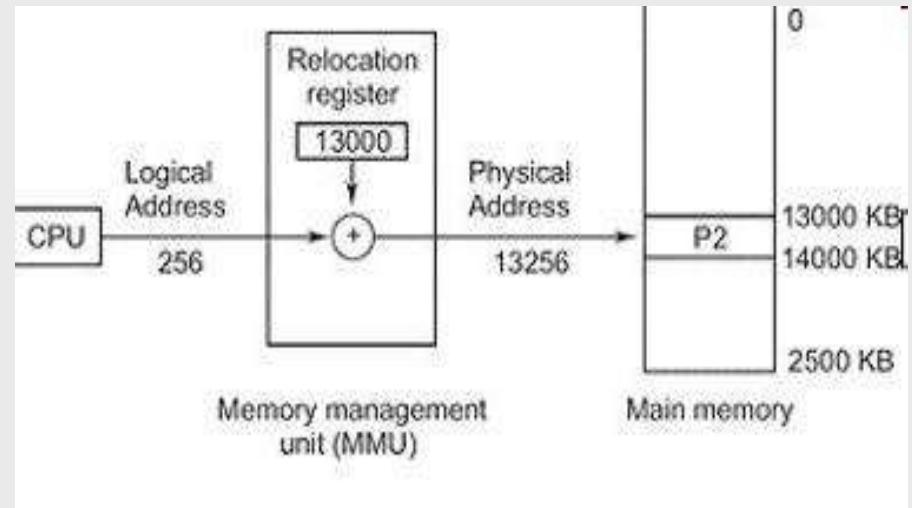
Mohammad S. Hasan
Staffordshire University, UK

Lecture Outline

- ◆ **Memory management definition**
 - **how the OS manages the allocation of memory to the various processes**
- ◆ **Address binding**
- ◆ **Logical vs. Physical Address Space**
- ◆ **Contiguous Allocation**
- ◆ **Paging**
- ◆ **Segmentation**

Address binding

- A process must be in the main memory to be executed.
- More than one process in the physical memory
- During execution processes make references to physical memory locations
 - i.e. locations in chips
- BUT - Program code represents locations symbolically
 - e.g. variable names

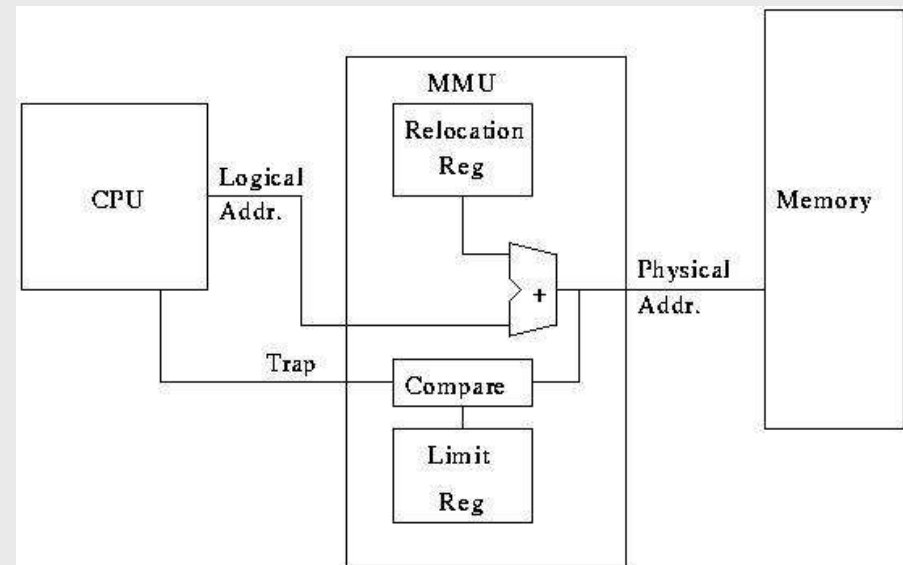


Address binding (Cont.)

- ◆ **Address binding**
 - **Mapping of one type of address to another**
- ◆ **Three address binding stages**
 - **Compile time binding**
 - **Link time binding**
 - **Execution time binding**

Compile time binding

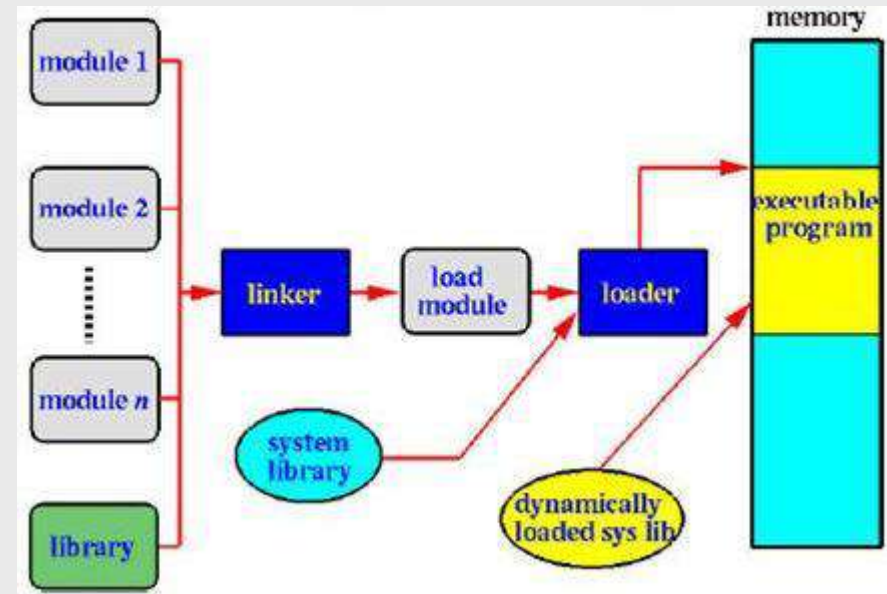
- If the process memory location is known when the program is compiled
 - compiler can replace symbolic references e.g. variables with actual addresses in memory.
- One must recompile code if location of process in memory is to change.



Link time binding

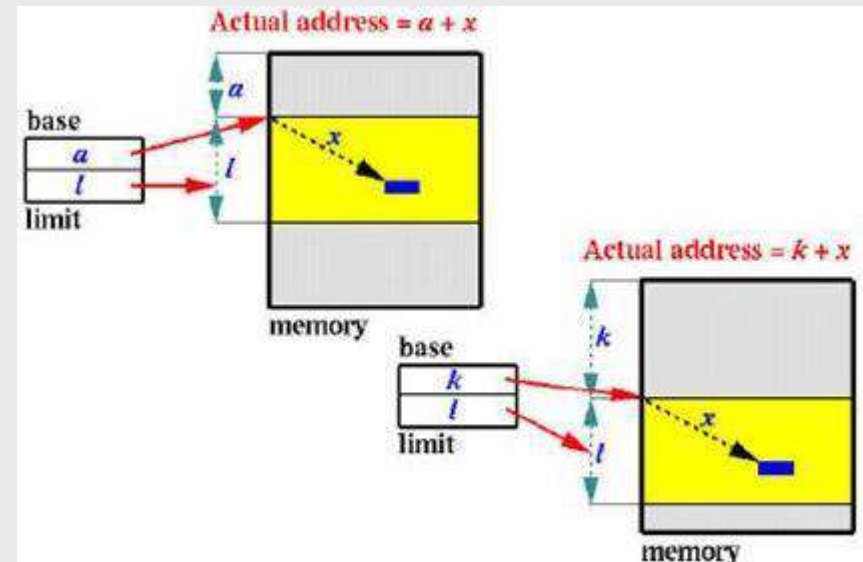
◆ If the memory location of the process is NOT known at compile time

- compiler replaces symbolic references to data with *relocatable addresses*.



Link time binding (Cont.)

- ◆ **The base address**
 - The actual address of beginning of program code
- ◆ **Relocatable addresses are relative to base address**
 - first word has address 0
 - Address of other locations as offset from base address
- ◆ **The base address is added to all addresses, when program is loaded into memory.**

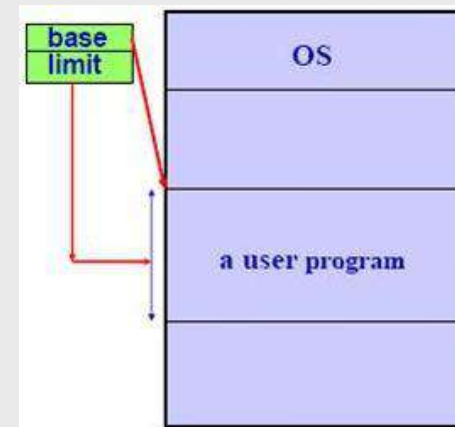
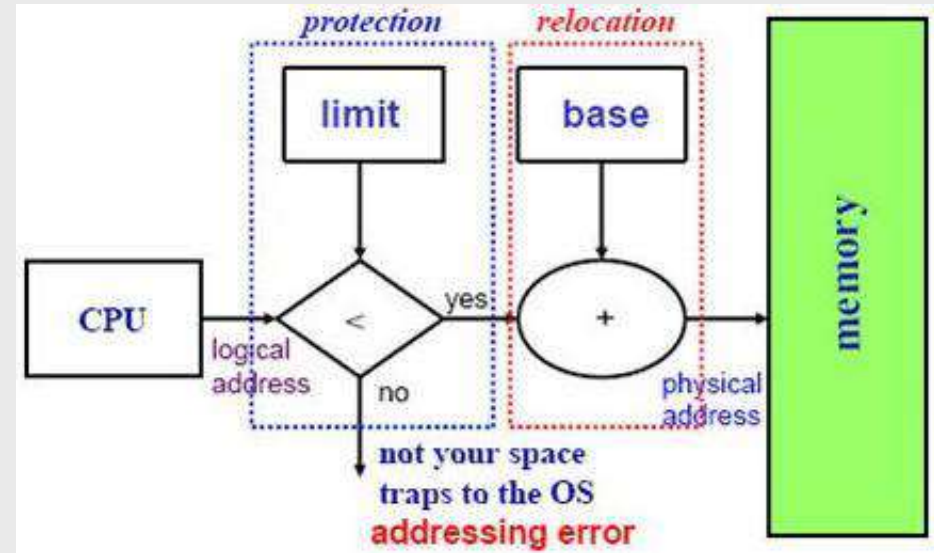


Execution time binding

- ◆ If the memory location of the process is only known when the process is to be executed
 - Also called run time binding
 - As before compiler replaces symbolic references to data with *relocatable addresses*.
 - C *malloc / calloc* function

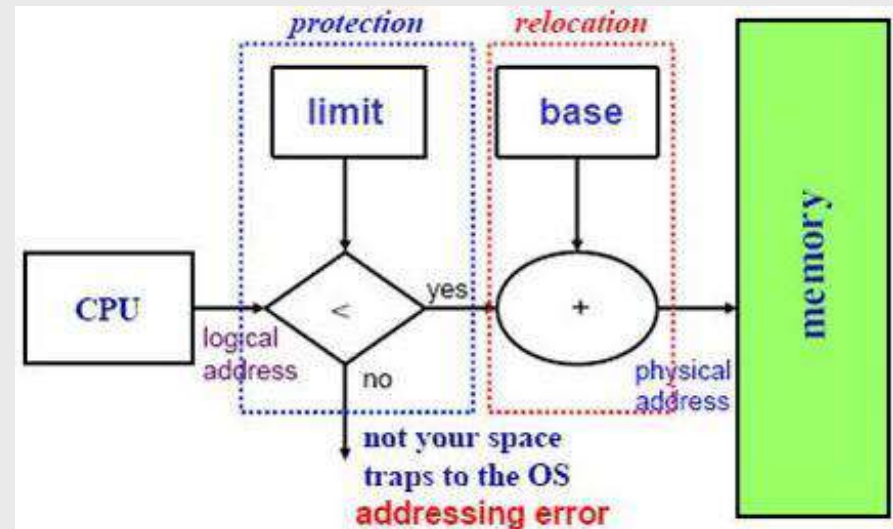
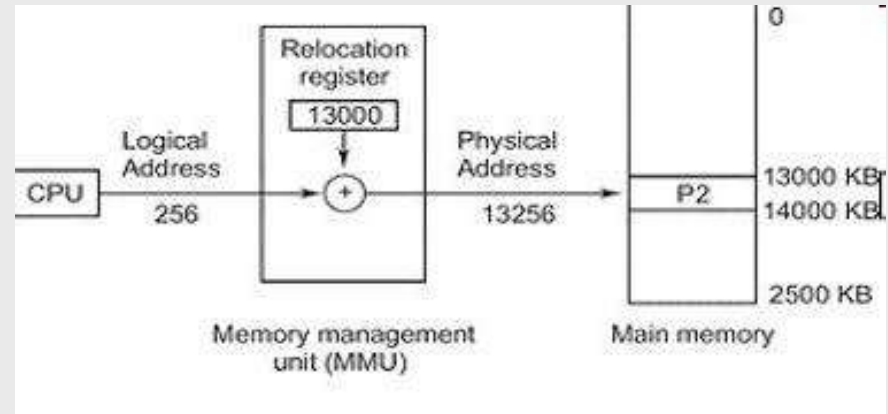
Execution time binding (Cont.)

- ◆ Process may be moved during its execution from one part of memory to another.
- ◆ Hardware (e.g. base and limit registers) generate actual / real address
 - Real address = address in some relocation register + relocatable address
- ◆ This is done for each memory reference.



Logical vs. Physical Address Space

- The address generated by the process, may be different to the actual physical location that holds the instruction or data

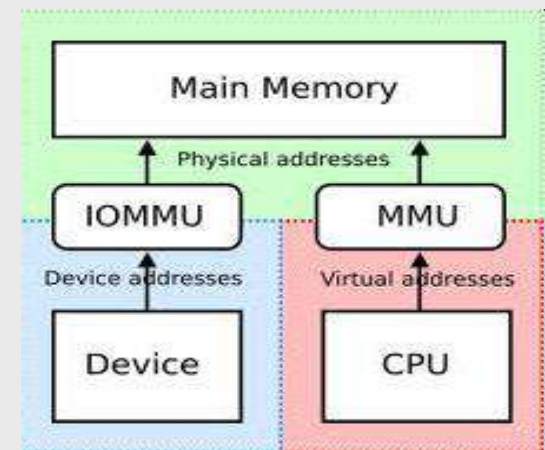
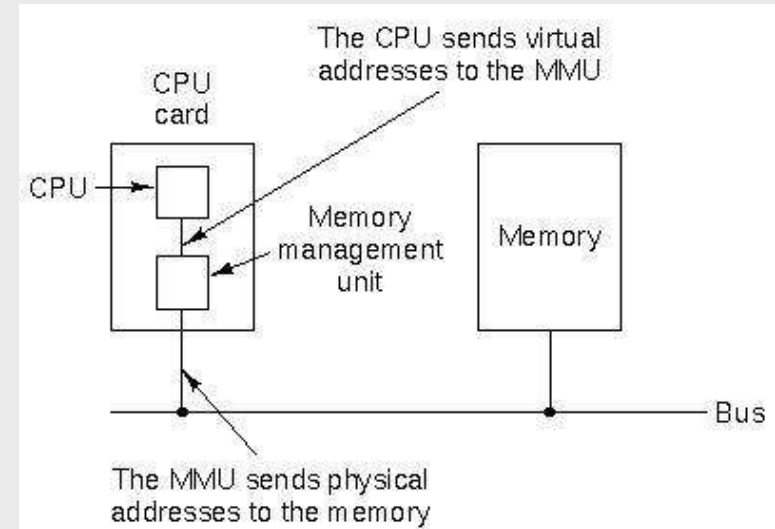


Logical vs. Physical Address Space (cont.)

- ◆ *Logical address or virtual address*
 - the address generated by the process running on CPU
- ◆ Logical address has to be translated into *physical address* of location in memory
- ◆ Notes
 - Logical and physical addresses are the same in compile-time and link-time address-binding,
 - But may be different in run-time binding

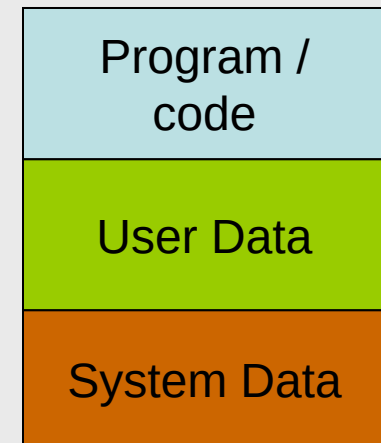
Memory-Management Unit (MMU)

- MMU hardware maps virtual / logical to physical address.
- MMU translates user process addresses before sending it to memory.
 - e.g. by adding a relocation register value to address
- Simple MMU
 - Base and limit registers



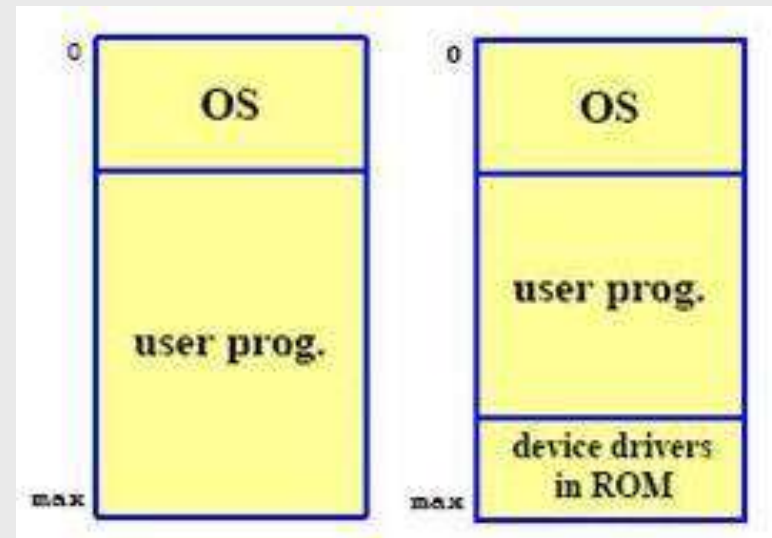
Memory allocation

- ◆ The scheme by which the OS decides what memory to allocate to a process for its
 - code, data and system data
- ◆ Three general schemes for memory allocation
 - contiguous allocation
 - paging
 - segmentation



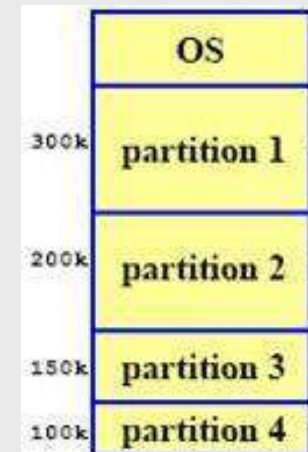
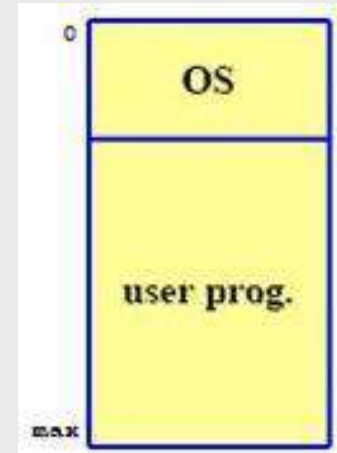
Contiguous Allocation

- ◆ The physical memory space of a process is held in one area of memory
 - i.e. all locations are contiguous (next to each other)
- ◆ Main memory divided into two parts
 - Resident OS
 - held in low memory.
 - User processes
 - held in high memory



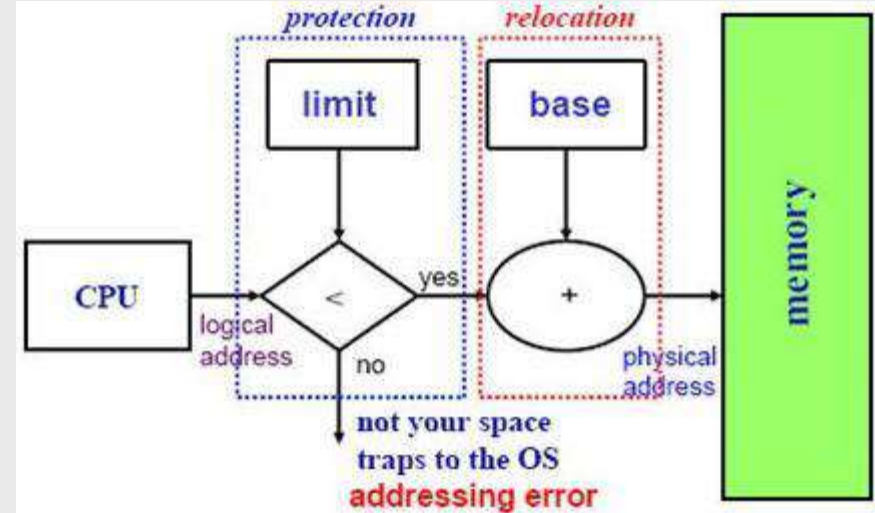
Contiguous Allocation (Cont.)

- ◆ Memory area for processes can be organised either
 - **Single partition allocation**
 - to hold the address space of a single process at a time
 - **Multiple partition allocation**
 - to hold the address space of several processes at the same time
 - one partition for each process

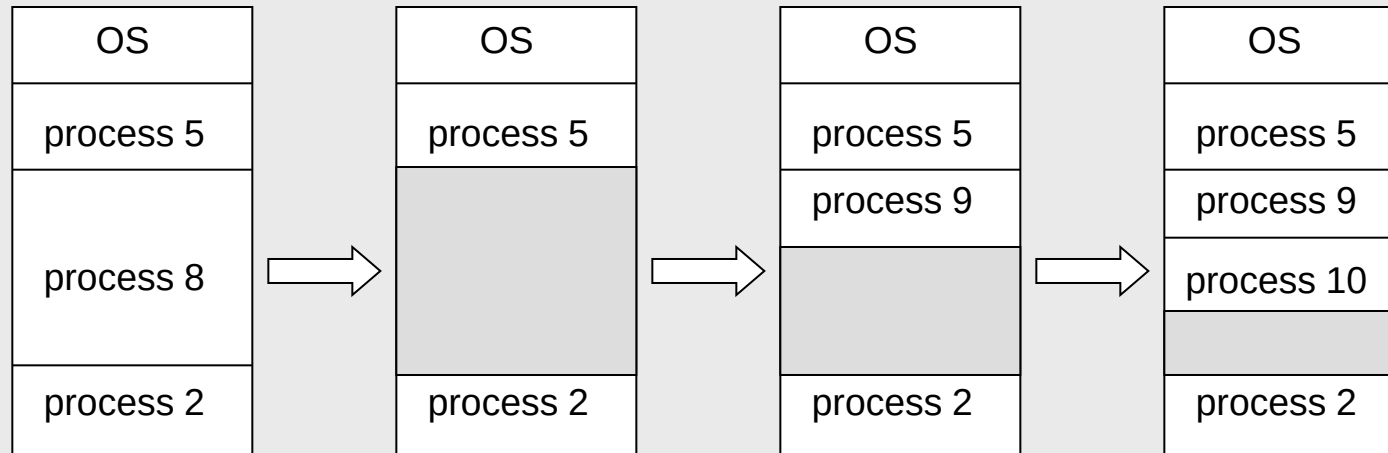


Single partition allocation

- ❖ Relocation register scheme used to protect OS code and data.
- ❖ Base register or relocation register contains the starting value of physical address.
- ❖ Limit register contains range of logical addresses
 - each logical address must be less than the limit register.



Multiple partition allocation



◆ Hole

- this is a block of memory that is currently available (free) to be allocated to a process.

- ◆ Holes of various sizes are scattered throughout memory.

Multiple partition allocation (Cont.)

- ◆ **When a process arrives**
 - it is allocated memory from a hole large enough to accommodate it.
- ◆ **OS maintains information about**
 - a) allocated partitions
 - b) free partitions (holes)

Dynamic Storage - Allocation Problem

◆ Three methods for hole allocation

● First-fit

- Allocate the first hole from list that is big enough.

● Best-fit

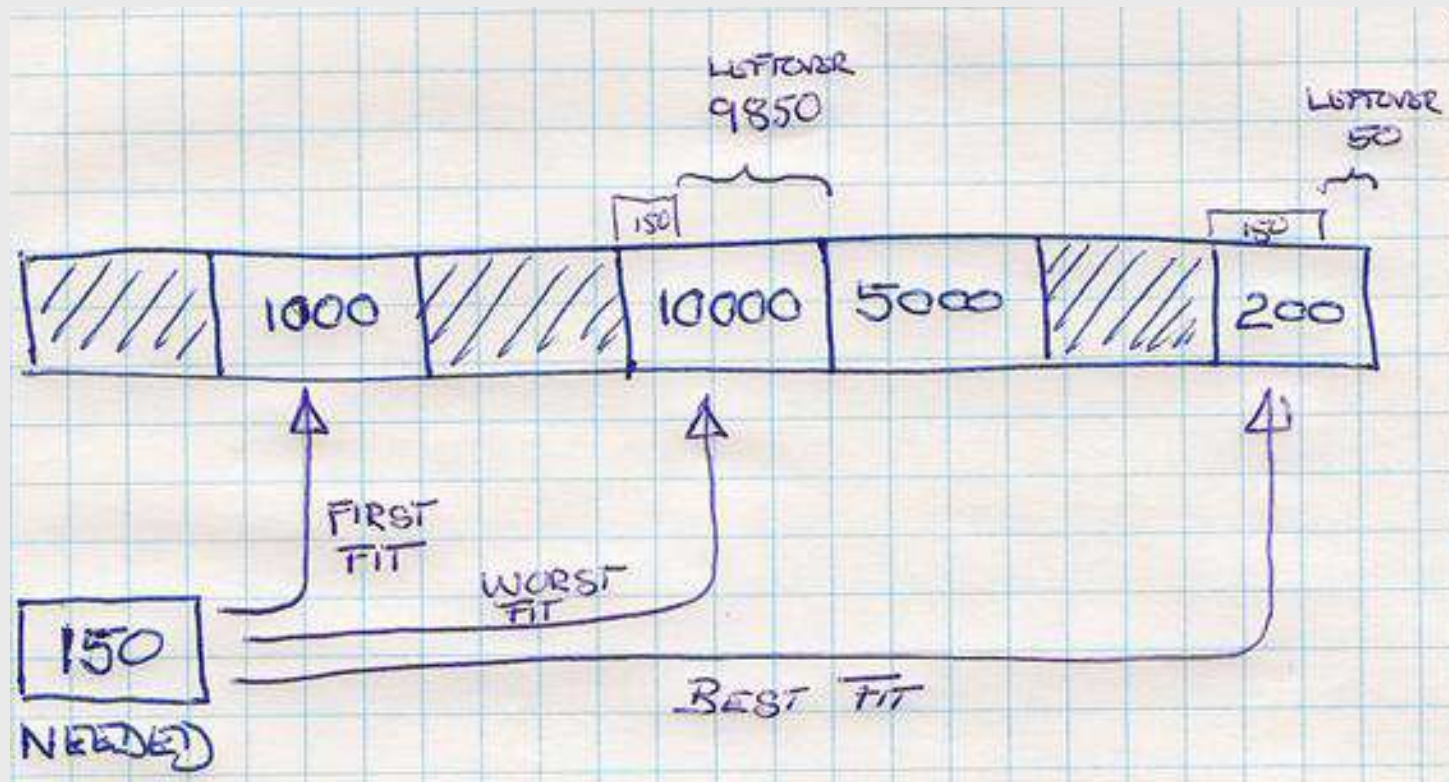
- Allocate the smallest hole that is big enough
- must search entire list, unless ordered by size.
- Produces the smallest leftover hole.

● Worst-fit

- Allocate the largest hole
- Must search entire list.
- Produces the largest leftover hole.



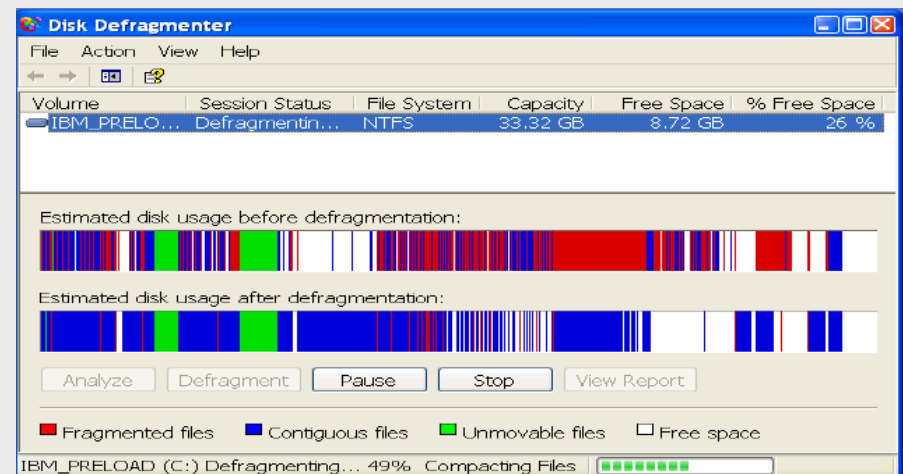
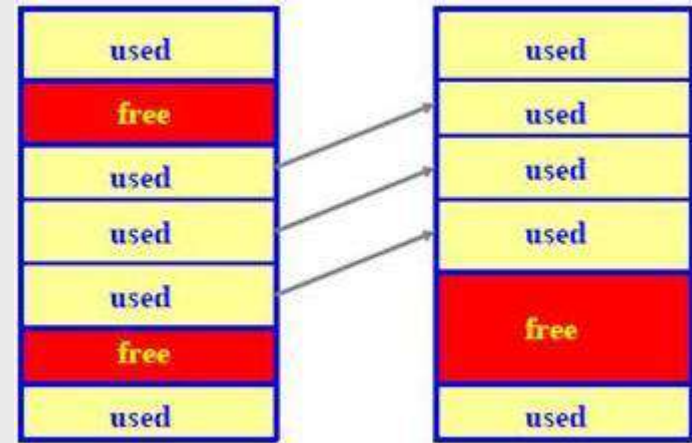
Dynamic Storage - Allocation Problem (Cont.)



Scan Direction

Fragmentation

- **External fragmentation**
 - enough total memory space for a process, but it is NOT contiguous.
- The free memory is divided into small holes
- thus process cannot be loaded.
- **Solution – compaction**
 - Defragmentation of HDD

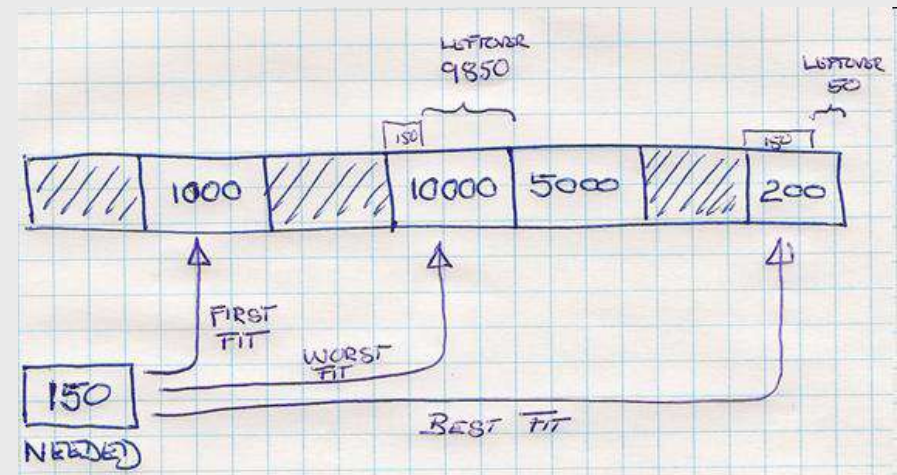
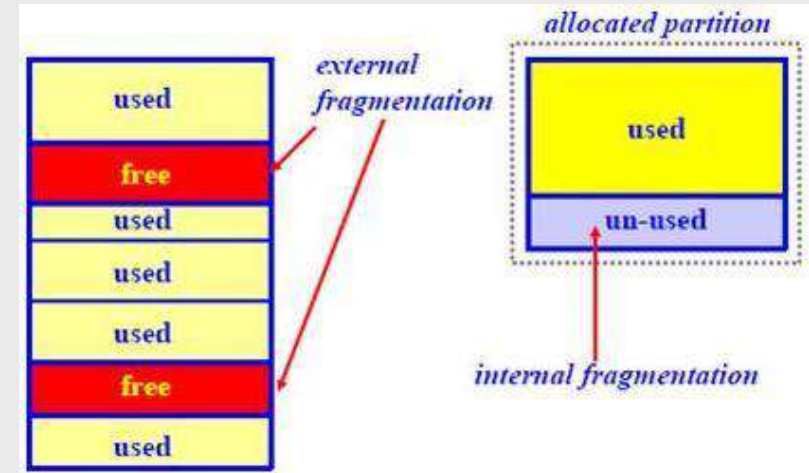


Fragmentation (Cont.)

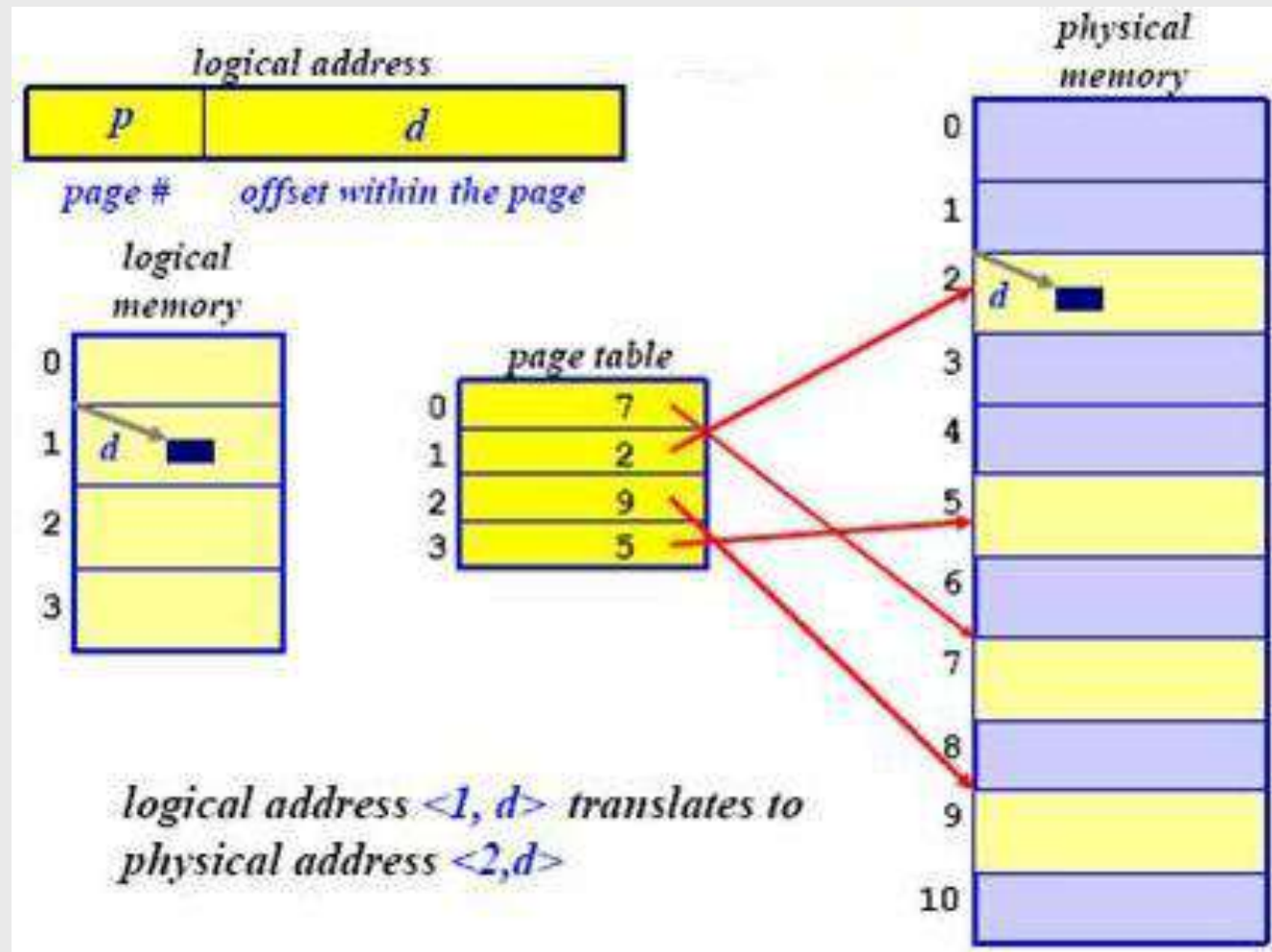
- ◆ **The unusable memory space exists outside of the memory partitions allocated to the running processes**
 - **Hence called external fragmentation.**
- ◆ **Reduce external fragmentation by compaction**
 - **Bring all holes together to form one large free block.**
 - **Compaction is possible only if relocation is dynamic and is done at execution time.**

Fragmentation (Cont.)

- ◆ **Internal fragmentation**
 - memory allocated to a process may be slightly larger than amount of memory requested.
- ◆ **The unused area of memory is internal to a partition**
 - Hence called internal fragmentation

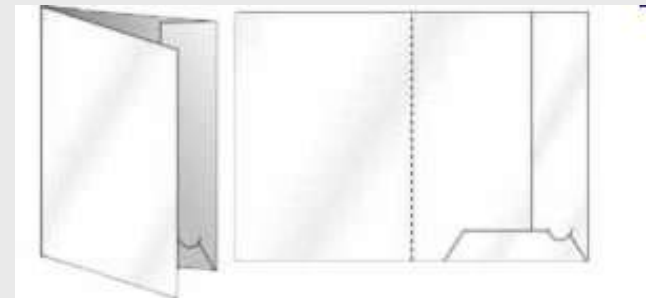


Paging



Paging

- ◆ **A process can have non-contiguous physical address space.**
- ◆ **Page**
 - **Logical memory of a process is also divided into blocks called pages.**
 - **In other words, the process is divided into a number of separate equal sized blocks**
 - **normally between 0.5K and 8K in size except may be the last page.**



Paging (Cont.)

- ◆ **Page Frame or simply “Frame”**
 - The physical memory is also divided into page-sized containers
 - normally between 0.5K and 8K in size.
- ◆ The physical address of a page is the frame address in which it is loaded.
- ◆ As a result a process can be allocated physical memory wherever the OS has an available frame.

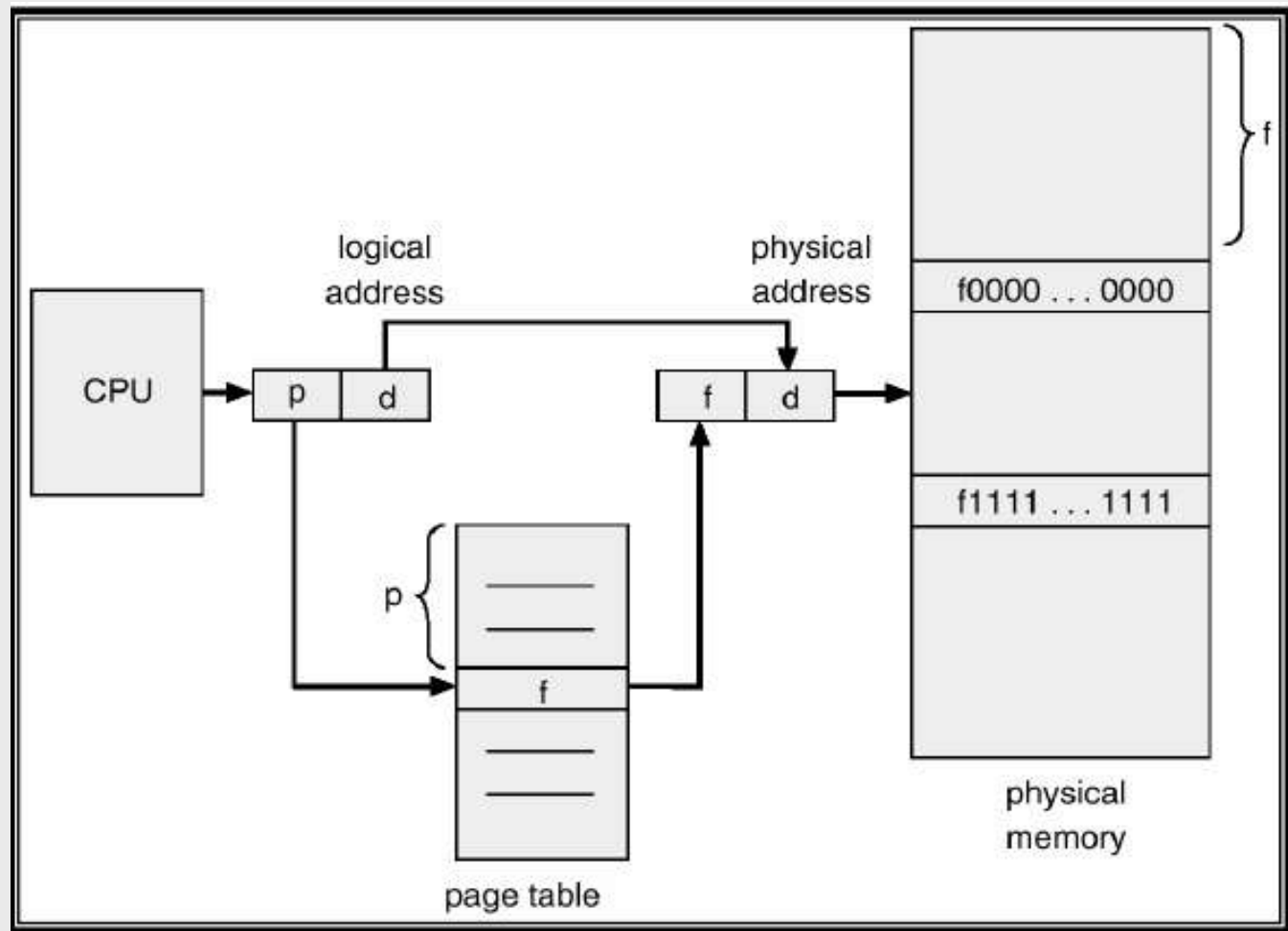
Paging (Cont.)

- ◆ OS keeps track of all free frames.
- ◆ To run a program of size n pages, OS needs to find n free frames and load program into them.
- ◆ Then set up a page table to translate logical to physical addresses.
- ◆ Separate page table for each process.

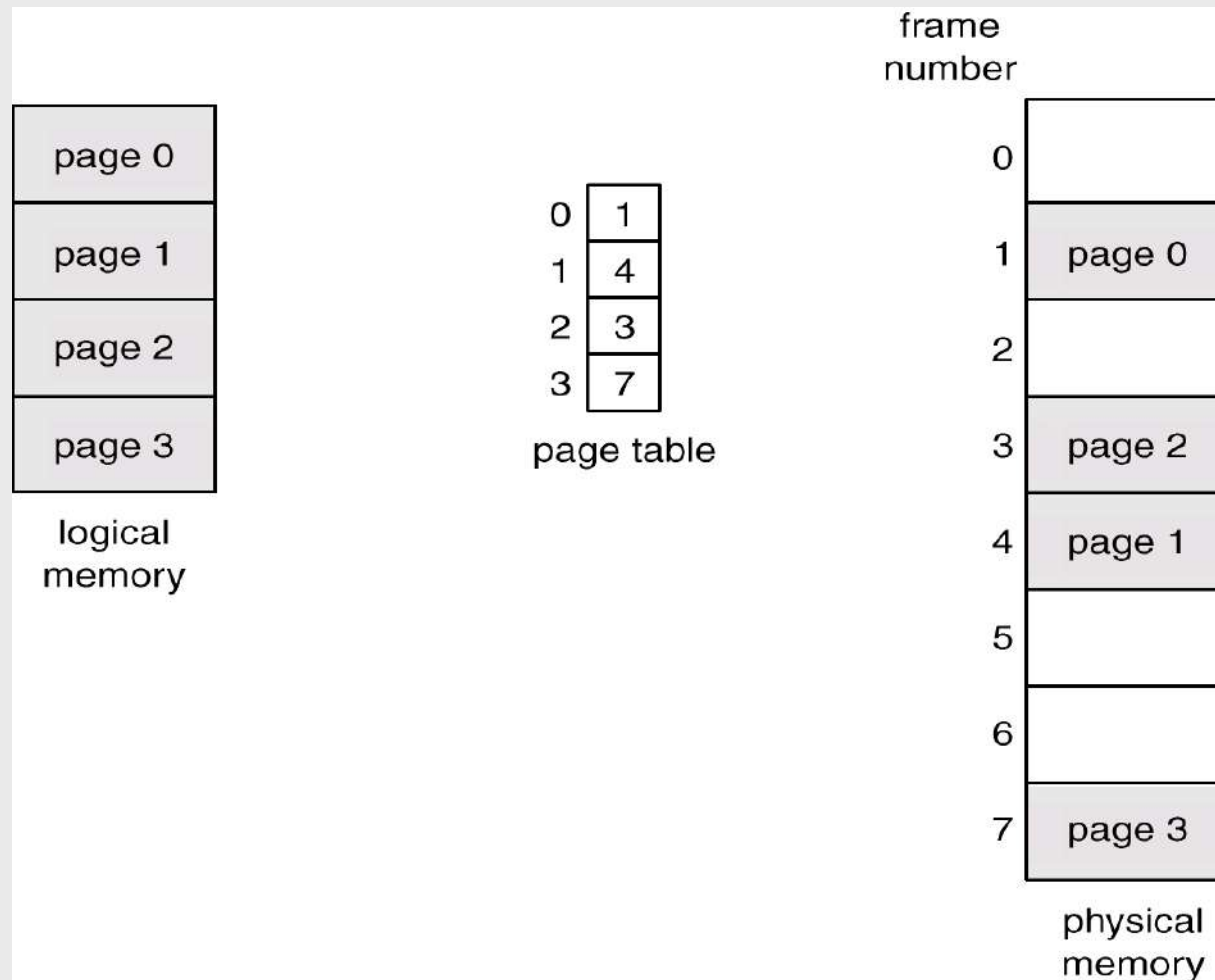
Address Translation Scheme

- ◆ 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.
- ◆ Paging eliminates external fragmentation, but internal fragmentation can still exist.

Address Translation Architecture



Paging Example



Page Table implementation – Main Memory

- ◆ Page table can be kept in main memory.
- ◆ *Page-table base register (PTBR)*
 - points to the page table.
- ◆ *Page-table length register (PRLR)*
 - indicates size of the page table.
- ◆ Every data/instruction access requires two memory accesses.
 - one to get the page frame address from the page table
 - one to actually get the page (data / instruction)



Implementation of Page Table (Cont.)

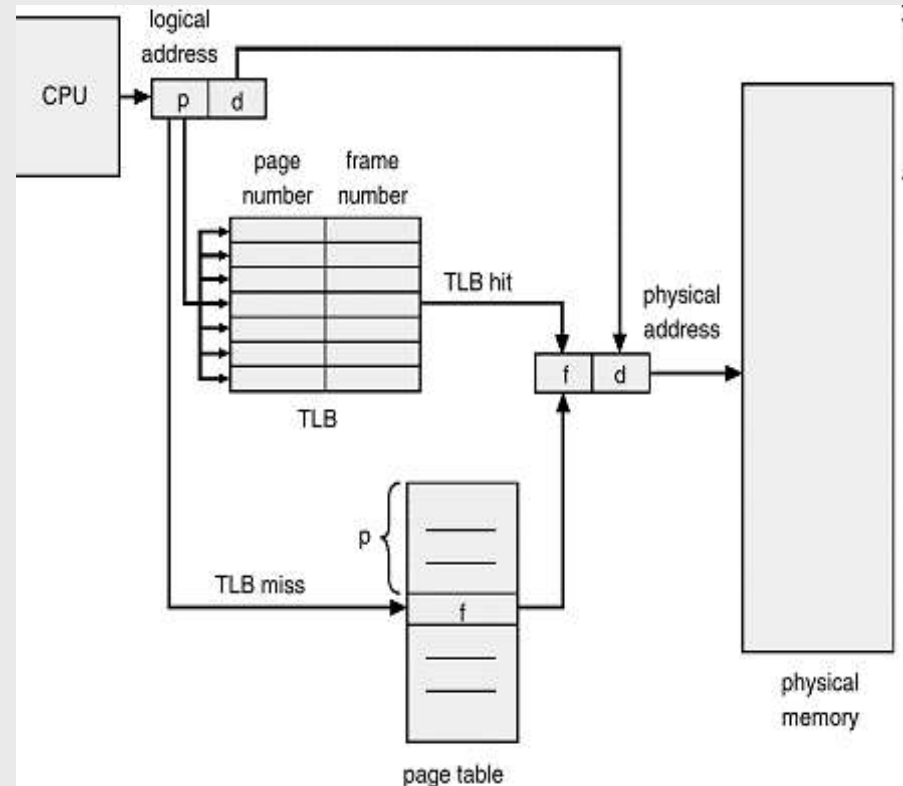
Page table in main memory problem

- two memory access – slower.

Solution

- *Associative Registers or Translation Look-aside Buffers (TLBs).*

- TLBs are special fast-lookup hardware cache.
- Stores the entire or partial page table



Page Table implementation - Associative Registers or TLBs

◆ Associative registers

- allow parallel search

◆ If page number is in associative register

- get frame number out.

◆ Else

- get frame number from page table in memory and
- add it to associative registers

Page #	Frame #



Effective Access Time

- ◆ **Associative Lookup = T_a time units**
 - **Processor speed: 3.2 GHz – T_a is nanosecond range**
- ◆ **Assume memory cycle time is 1 μ s**
- ◆ **Hit ratio**
 - **percentage of times that a page number is found in the associative registers;**
 - **ratio related to number of associative registers.**
- ◆ **If hit ratio = α , then Effective Access Time (EAT) can be found as follows:**

Effective Access Time (Cont.)

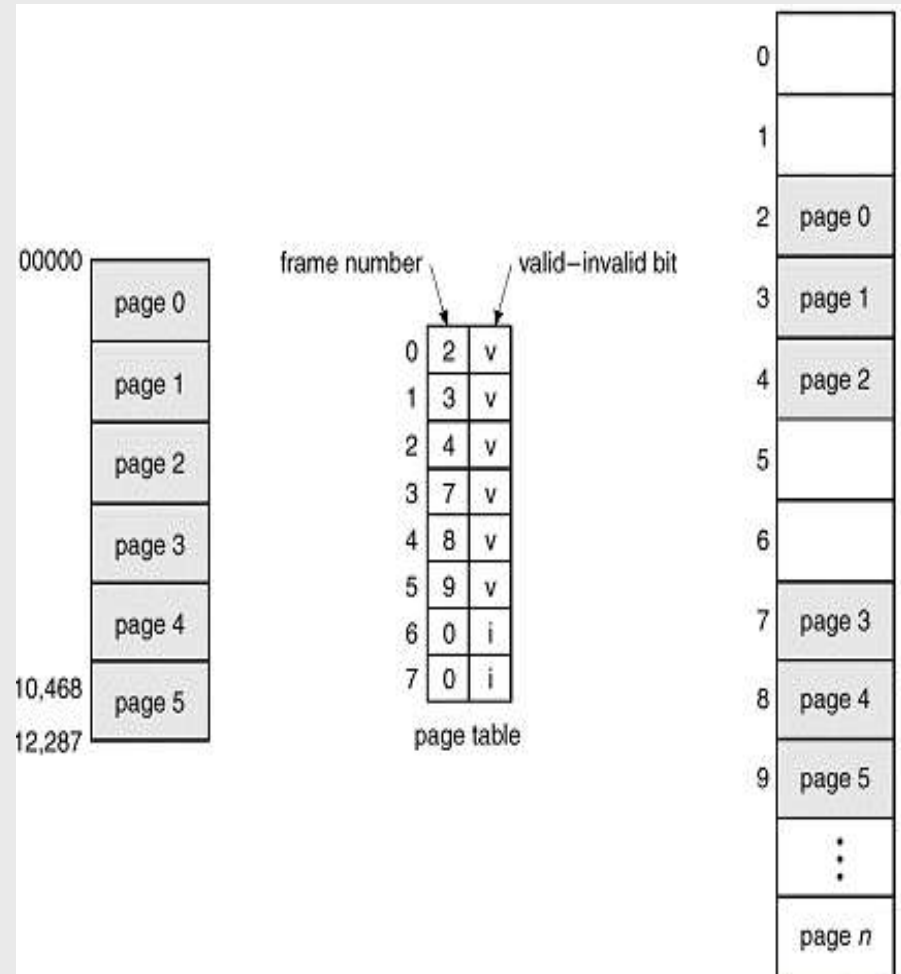
- ◆ If page no. in associative regs: α fraction of the time)
 - T_a time units for register lookup + $1 \mu s$ to access memory for data
- ◆ Else i.e. if page no. not in associative regs: $1 - \alpha$ fraction of the time
 - T_a time units for register lookup + $1 \mu s$ for page frame address lookup + $1 \mu s$ to access memory for data
- ◆ Overall formula
 - $EAT = \alpha(1 + T_a) + (1 - \alpha)(2 + T_a) = 2 + T_a - \alpha$



Memory Protection

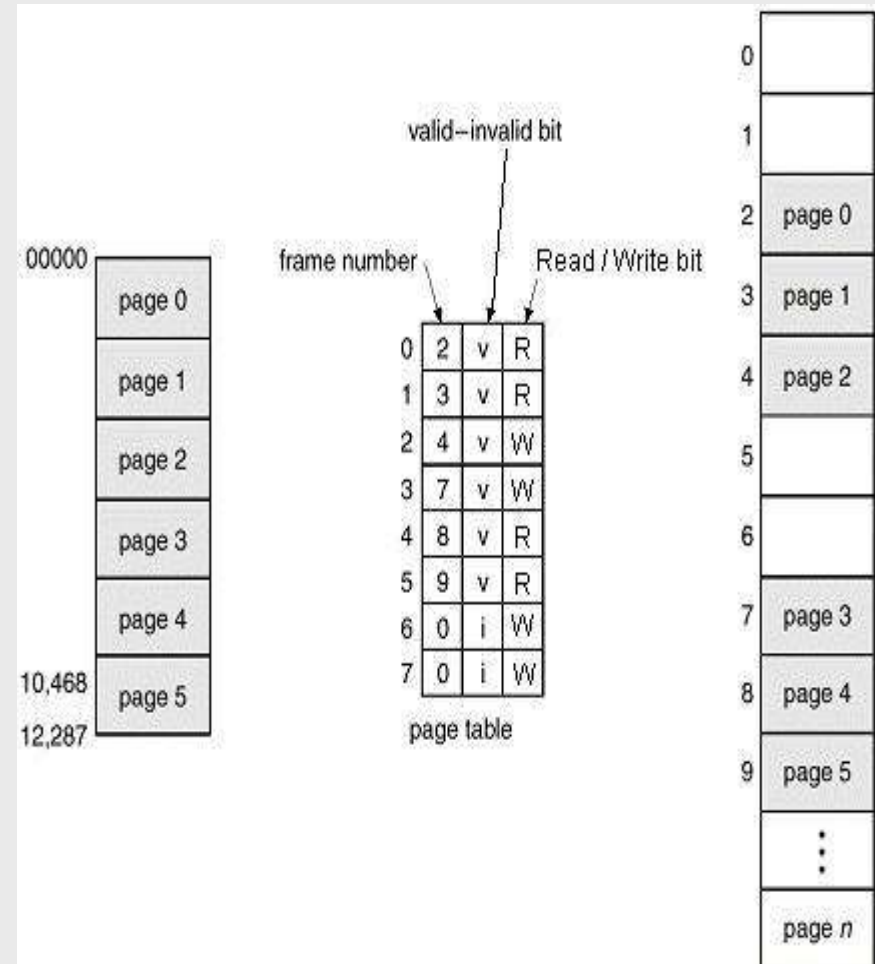
◆ **A Valid-invalid bit attached to each entry in the page table for process context switch**

- **“valid” indicates that the associated page is legal or valid page for the process.**
- **“invalid” indicates that the page is not in the process’ logical address space and should thus generate an error**



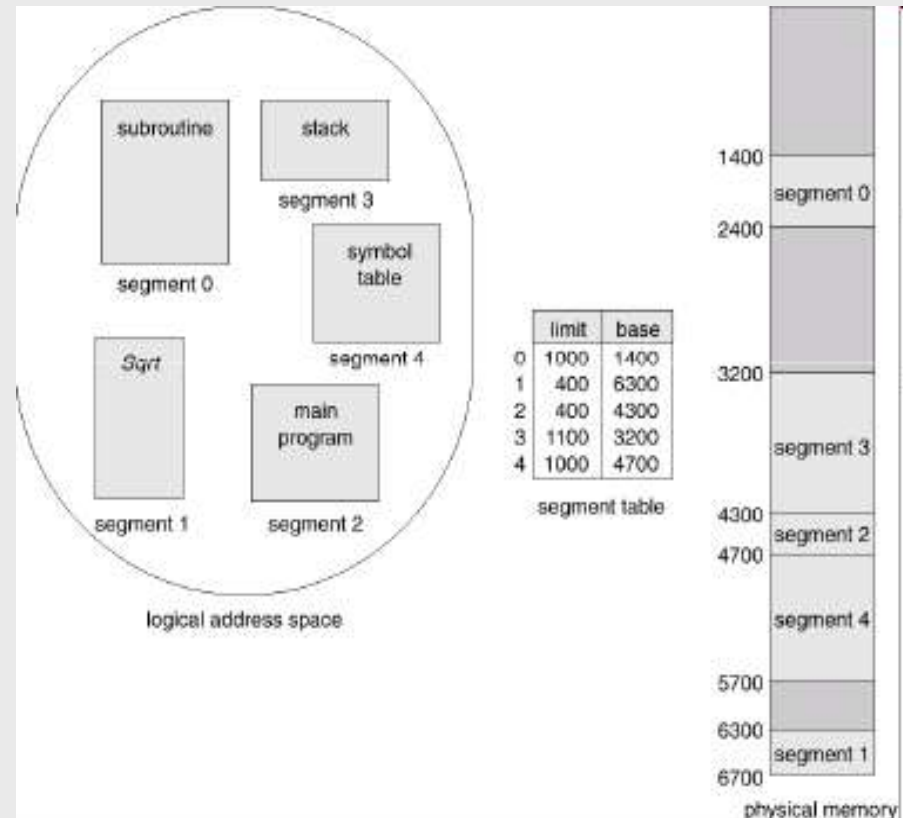
Memory Protection (Cont.)

- ◆ **Read / Write protection**
 - a protection bit with the page table entry for each frame
 - the bit specifies whether frame is read/write or read only



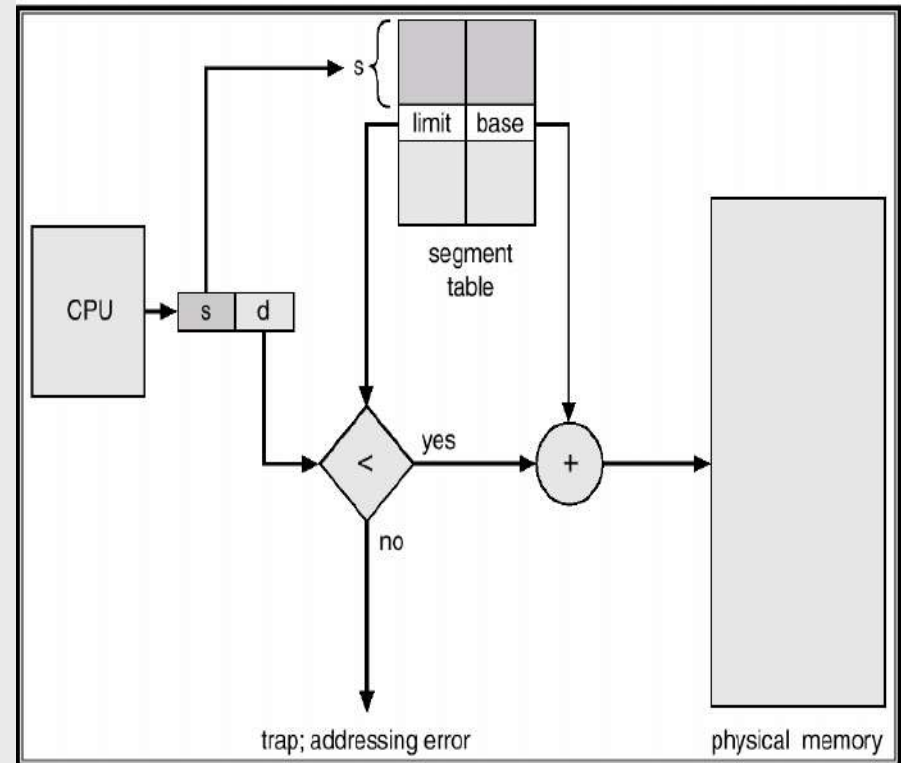
Segmentation

- ◆ Segmentation divides up processes' address space into a number of blocks or logical units called segments
- ◆ A program is treated as a collection of segments.
 - main part of program,
 - a group of functions,
 - the set of global data values, arrays, a stack,
 - in OO languages may be individual objects

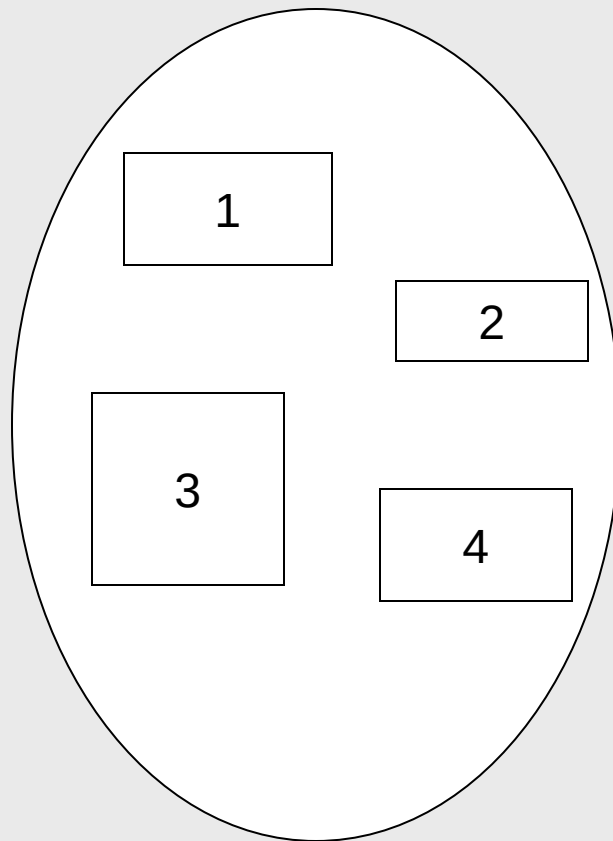


Segmentation (Cont.)

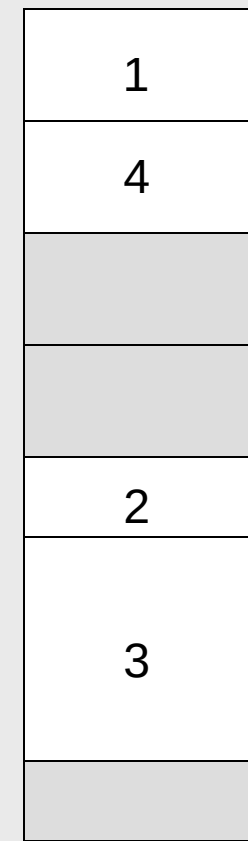
- Segments are similar to pages
- The difference is
 - Pages**
 - all have the same size
 - Segments**
 - may be a number of different sizes



Logical View of Segmentation



user space



physical memory space

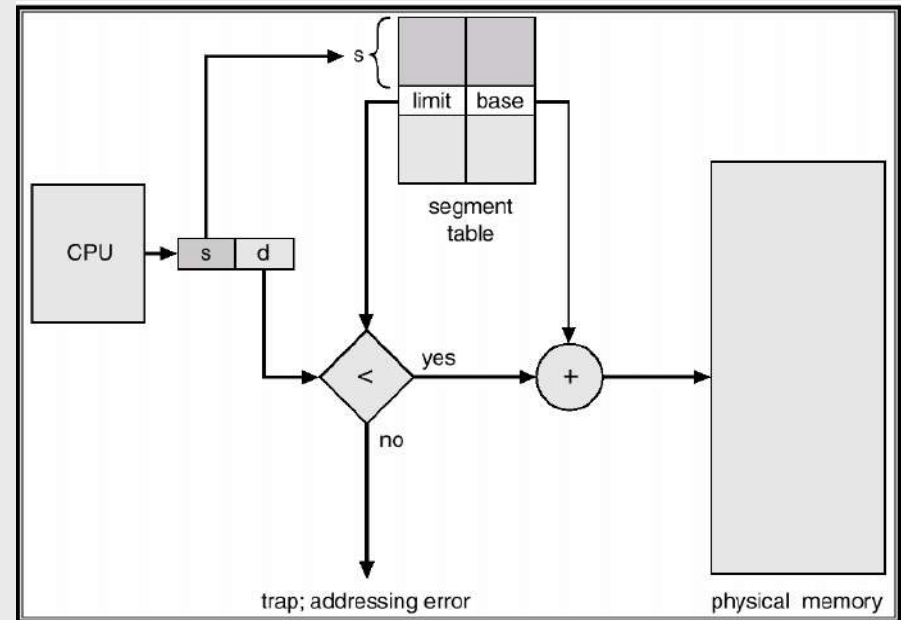
Segmentation Architecture

- ◆ **Logical address consists of a pair: <segment-number, offset>,**
 - **Segment number is an index into the *segment table* - with one segment table for each process**
 - **offset is the value to be added to start address of segment to give real physical address**
- ◆ **Each entry in segment table has**
 - **base value – contains the starting physical address where the segment resides in memory.**
 - ***limit* – specifies the length of the segment.**



Segmentation Architecture (Cont.)

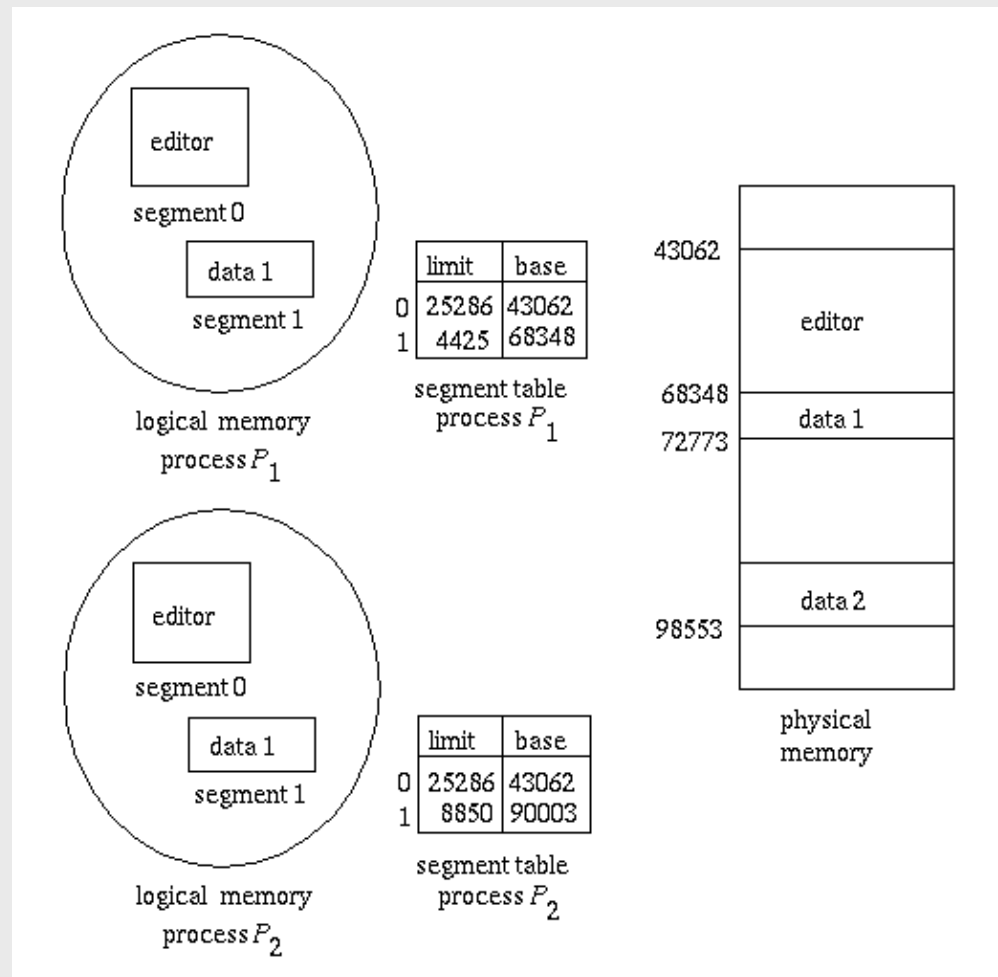
- Offset address is legal if $\text{offset} < \text{limit value}$
- *Segment-table base register (STBR)* points to the segment table's location in memory.
- *Segment-table length register (STLR)* indicates number of segments used by a program;
- segment number s is legal if $s < \text{STLR}$.



Segmentation Architecture (Cont.)

- ◆ **Protection - each entry in segment table contains:**
 - **validation bit = 0 $\Rightarrow$ illegal segment**
 - **specification of read/write/execute privileges**
- ◆ **Protection bits associated with segments**
- ◆ **Since segments vary in length, memory allocation is a dynamic storage-allocation problem.**

Segmentation example



References

◆ **Operating System Concepts. Chapter 8.**