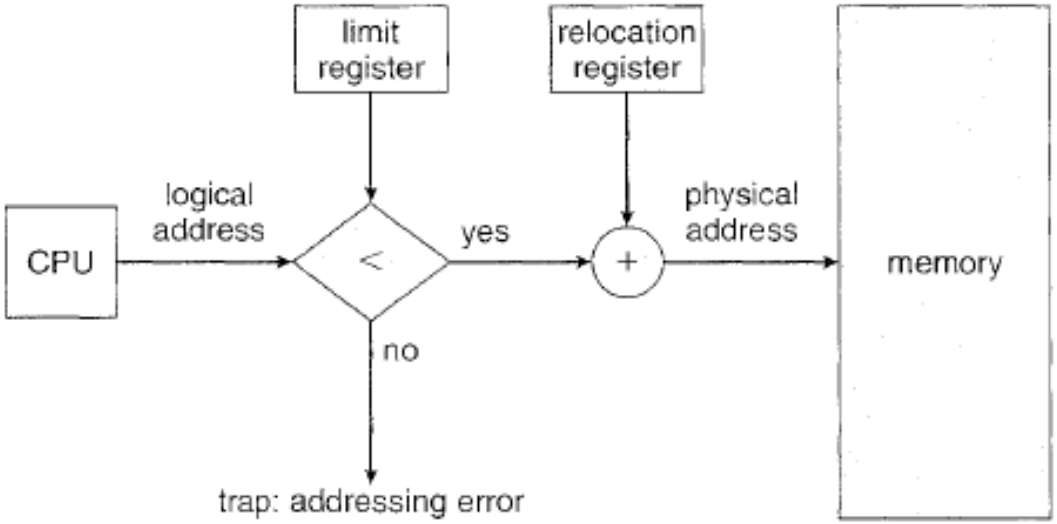
Memory Management!

Basic Memory Management

- Exposing Physical Memory to processes is not a good idea.
- We need to
  - Protect OS from access from users program
  - Protect users program from one another

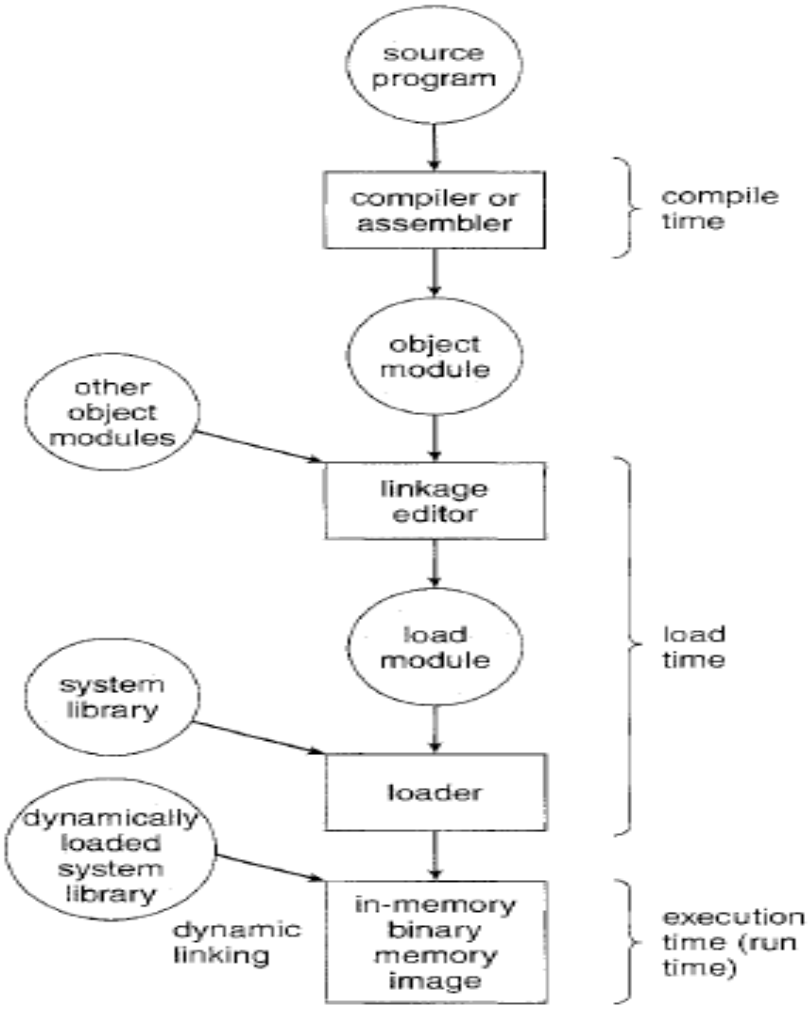
Memory Protection

- Base Register: Holds the smallest legal Physical Memory
- Limit Register: Contains the size of the range



CODING BLOCKS

Multi Step Processing of Program



CODING BLOCKS

Address Binding

Static-binding: before a program starts running

- Compile time: Compiler and assembler generate an object file for each source file
- Load time:
  - Linker combines all the object files into a single executable object file
  - Loader (part of OS) loads an executable object file into memory at location(s) determined by the OS
    - invoked via the execve system call

Dynamic-binding: as program runs

- Execution time: uses new and malloc to dynamically allocate memory gets space on stack during function calls

Static Loading

- The entire program and all data of a process must be in physical memory for the process to execute
- The size of a process is thus limited to the size of physical memory

Dynamic Linking

A dynamic linker is actually a special loader that loads external shared libraries into a running process

- Small piece of code, stub, used to locate the appropriate memory-resident library routine
- Only one copy in memory
- Don't have to re-link after a library update

Logical vs Physical Address Space

Mapping logical address space to physical address space is central to MM

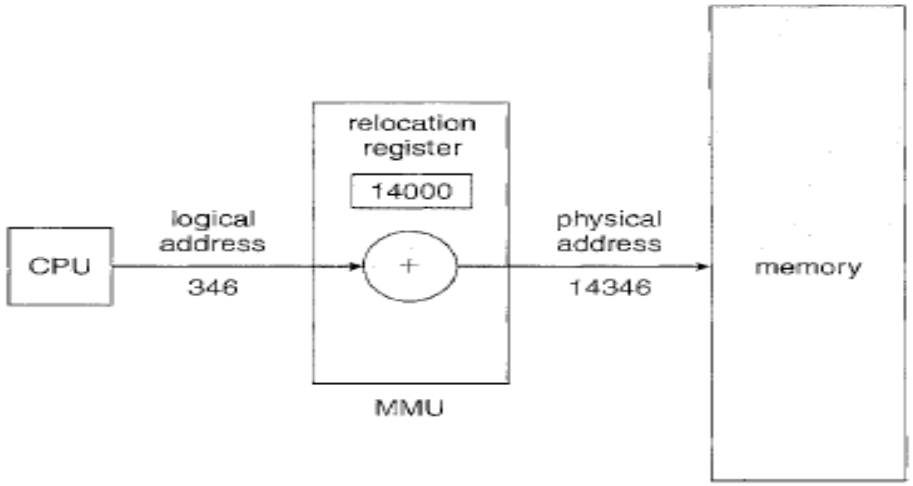
- Logical address generated by the CPU; also referred to as virtual address
- Physical address address seen by the memory unit

- In compile-time and load-time address binding schemes, LAS and PAS are identical in size
- In execution-time address binding scheme, they are differ.

Logical vs Physical Address Space

The User Program

- Deals with logical addresses
- Never sees the real physical addresses



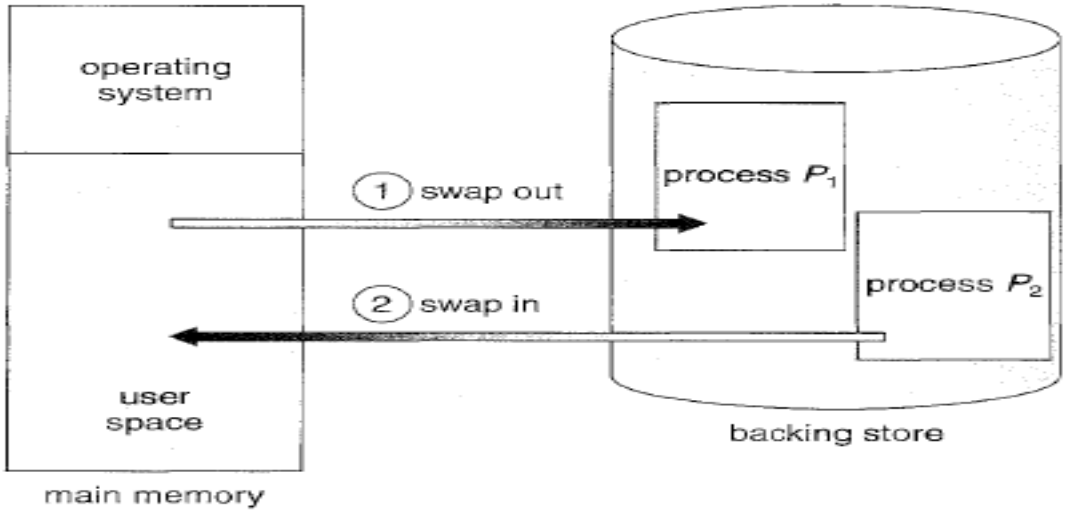
CODING BLOCKS

Memory Management Unit

- CPU sends Virtual Addresses to MMU
- MMU sends Physical Addresses to CPU

Swapping

A process must be in memory to be executed. A process, however, can be **swapped** temporarily out of memory to a backing store and then brought into memory for continued execution.



CODING BLOCKS

Contagious Memory Allocation

In contiguous memory allocation, each process is contained in a single contiguous section of memory.

Dynamic Storage Allocation Problem

- First Fit
- Best Fit
- Worst Fit

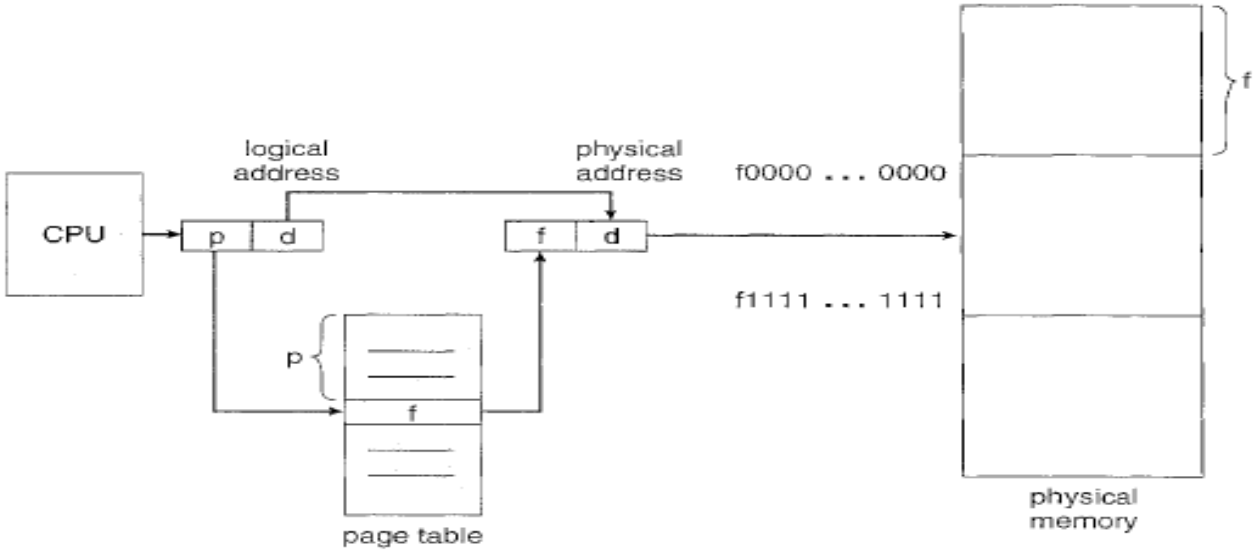
Simulations have shown that both first fit and best fit are better than worst fit in terms of decreasing time and storage utilization. Neither first fit nor best fit is clearly better than the other in terms of storage utilization, but first fit is generally faster.

Fragmentation

- External Fragmentation
- Internal Fragmentation – Memory are allocated in fixed size of blocks

Paging

Paging is a memory management scheme that permits the physical address space of a process to be non-contiguous.



CODING BLOCKS

Paging Basic Method

Break Logical address into Pages

Break Physical address into Frames

- Every address generated the CPU is divided into two parts: a page number (p) and a page offset (d)
- The page number is used as an index into a page table which contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

Page Table

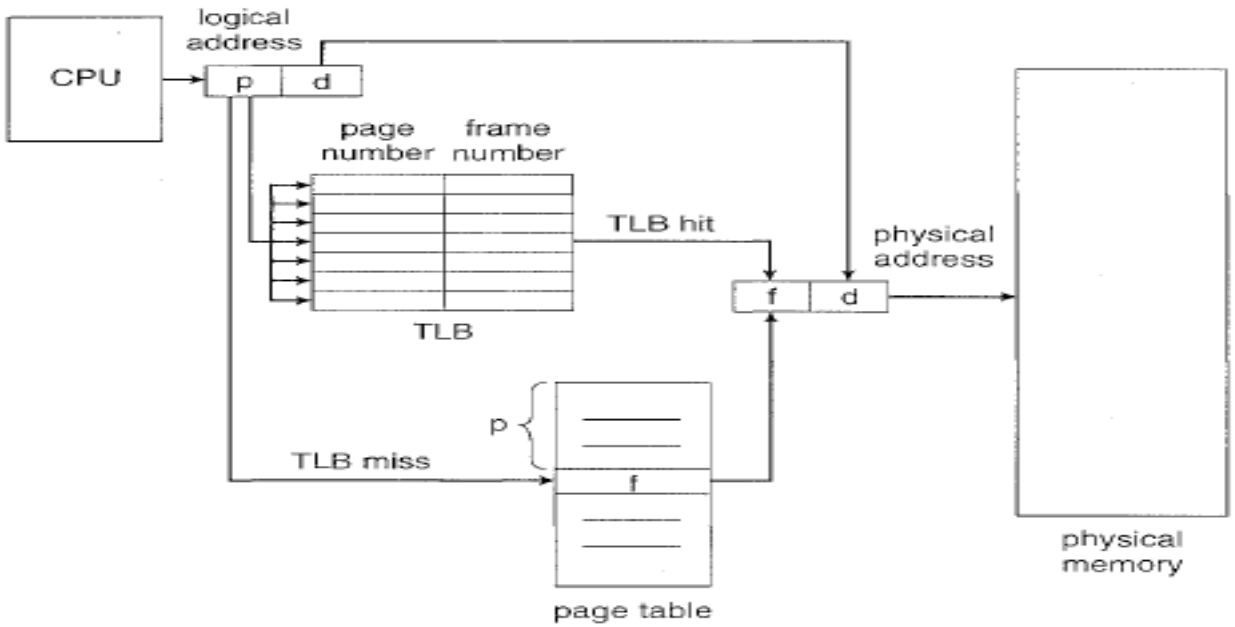
Page table is kept in main memory

- Usually one page table for each process
- Page-table base register (PTBR): A pointer to the page table is stored in PCB
- Page-table length register (PRLR): indicates size of the page table
- Slow
  - Requires two memory accesses. One for the page table and one for the data/instruction.

Paging with TLB

TLB – Translation Look aside buffer

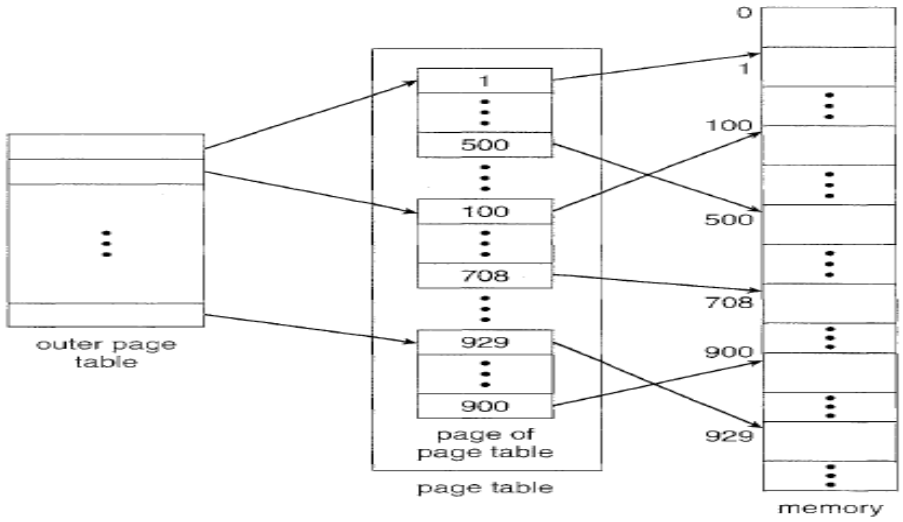
Only a small fraction of the PTEs are heavily read; the rest are barely used at all.



CODING BLOCKS

Hierarchical Paging

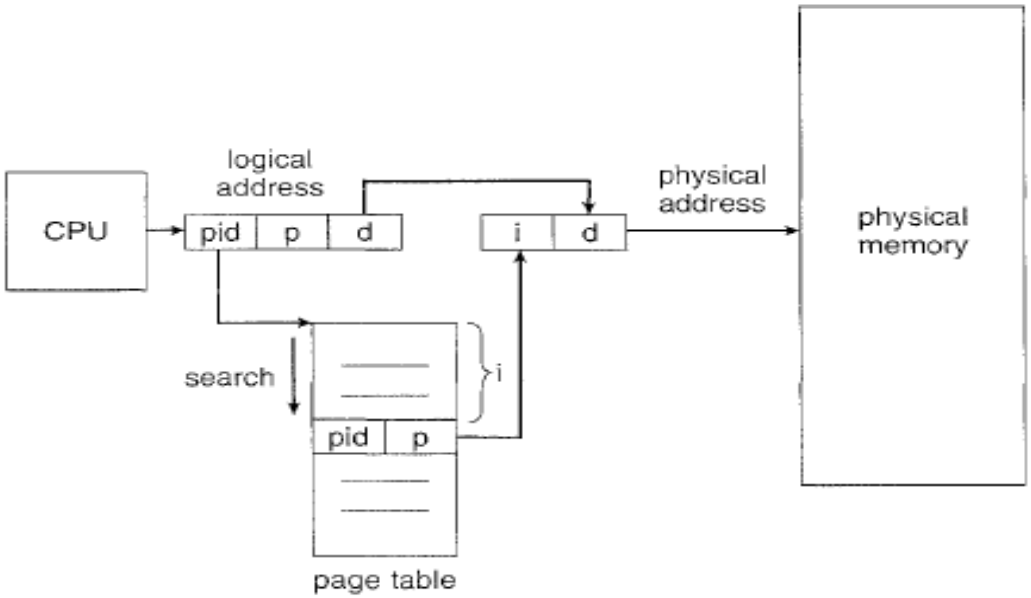
- a 1M-entry page table eats 4M memory
- while 100 processes running, 400M memory is gone for page tables
- avoid keeping all the page tables in memory all the time



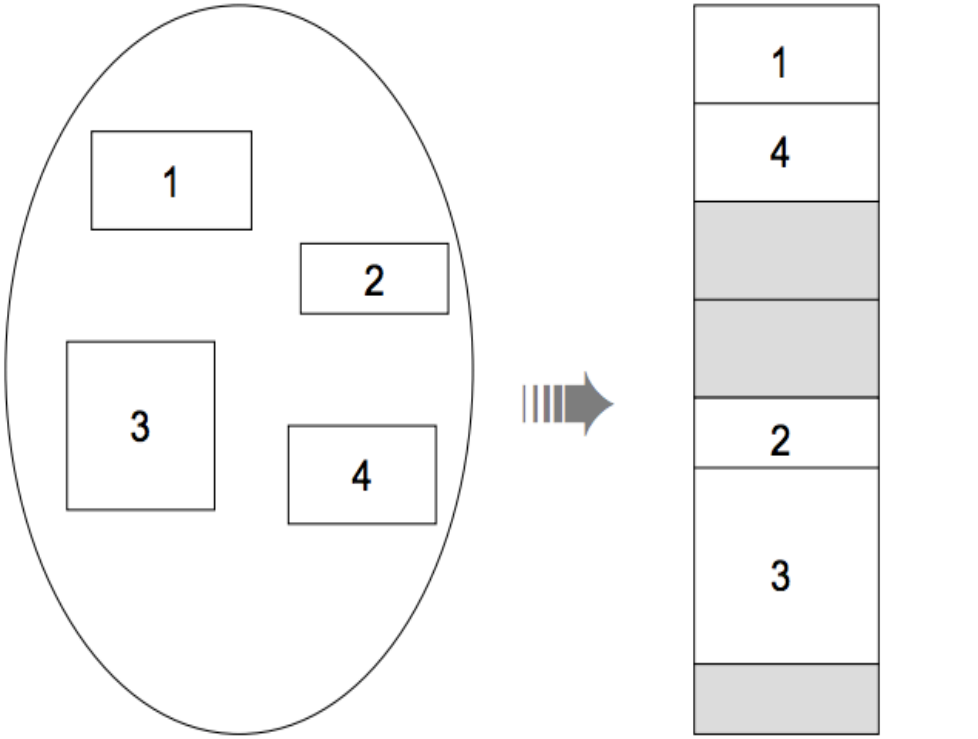
CODING BLOCKS

Inverted Page Table

- Index with Frame Number
- One entry for each physical frame
- A single global page table for all processes



Segmentation

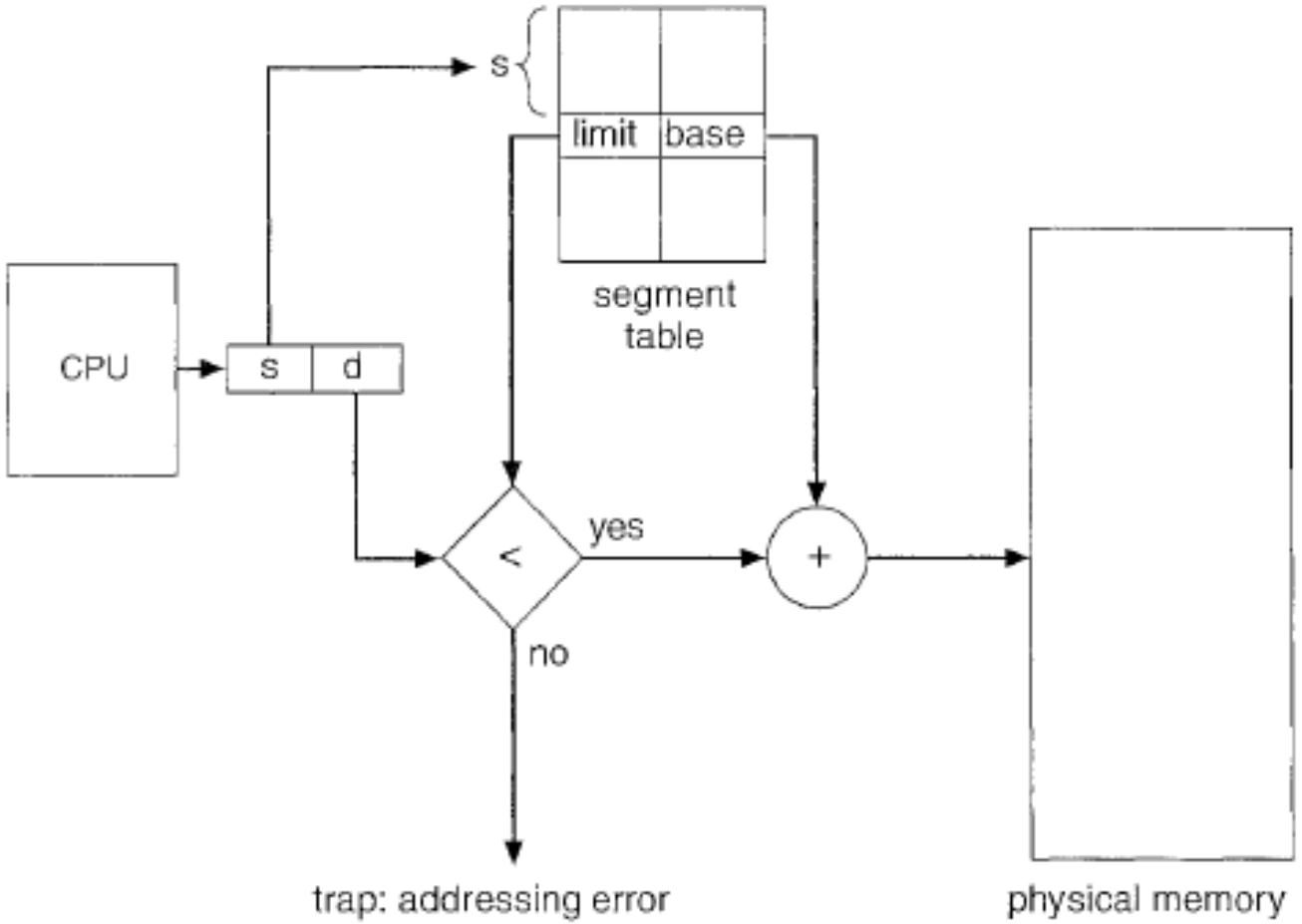


Segmentation …

Logical address consists of a two tuple: <segment-number, offset>

- Segment table maps 2D virtual addresses into 1D physical addresses; each table entry has:
  - base contains the starting physical address where the segments reside in memory
  - limit specifies the length of the segment
- 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 < STLR

Segmentation Hardware



Pros and Cons of Segmentation

- Each segment can be
  - located independently
  - separately protected
  - grow independently
- Segments can be shared between processes
- Variable allocation
- Difficult to find holes in physical memory
- Must use one of non-trivial placement algorithm →first fit, best fit, worst fit
- External fragmentation

Virtual Memory Management!

What is Virtual Memory?

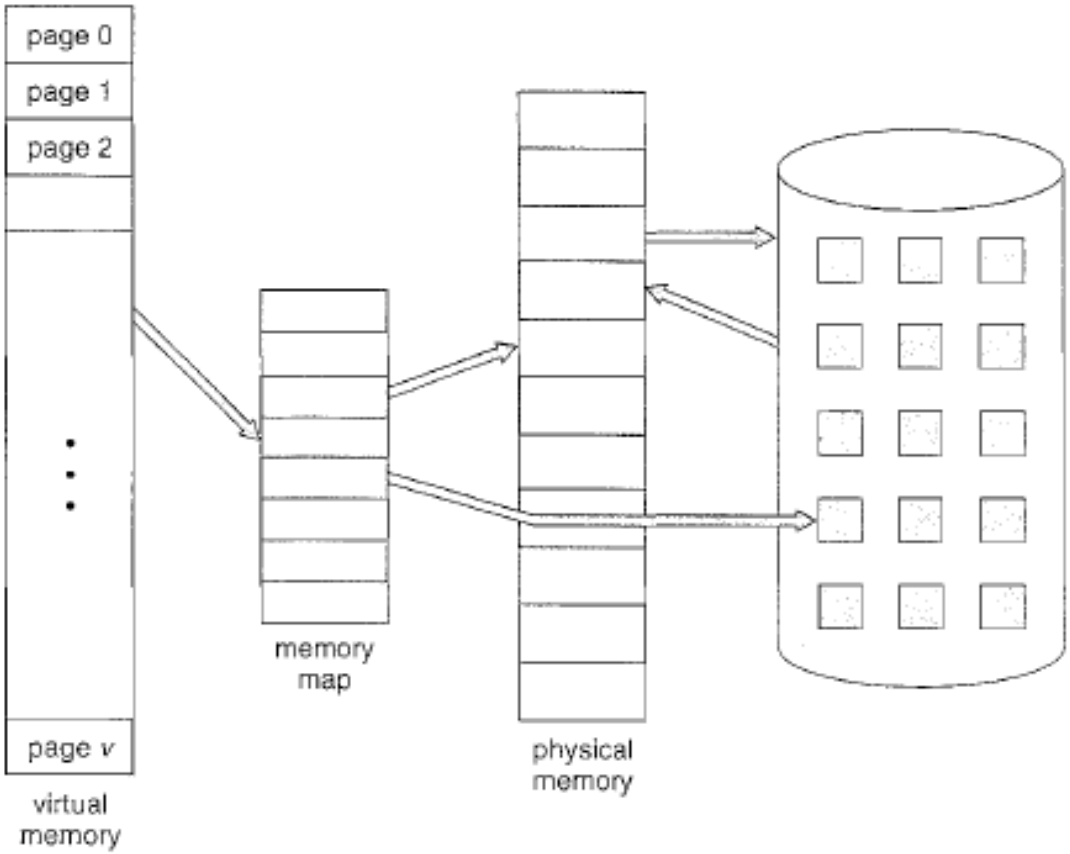
It involves the separation of logical memory as perceived by users from physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical lmemory is available. Virtual memory makes the task of programming much easier, because the programmer no longer needs to worry about the amount of physical memory available; she can concentrate instead on the problem to be programmed.

Why Virtual Memory?

The ability to execute a program that is only partially in memory would confer many benefits:

- A program would no longer be constrained by the amount of physical memory that is available. Users would be able to write programs for an extremely large virtual address space, simplifying the programming task.
- Because each user program could take less physical memory, more programs could be run at the sance time, with a corresponding increase in CPU utilization and throughput but with no increase in response time or turnaround time.
- Less I/O would be needed to load or swap user programs into memory, so each user program would run faster.

Virtual Memory

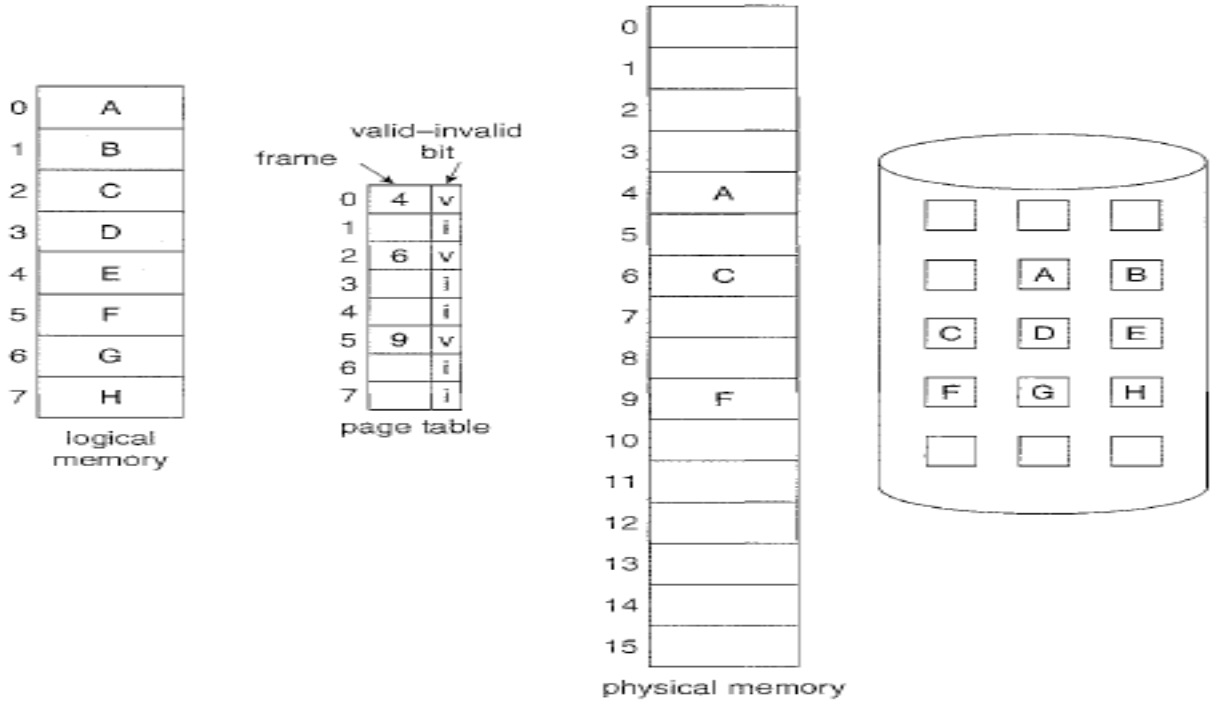


Demand Paging

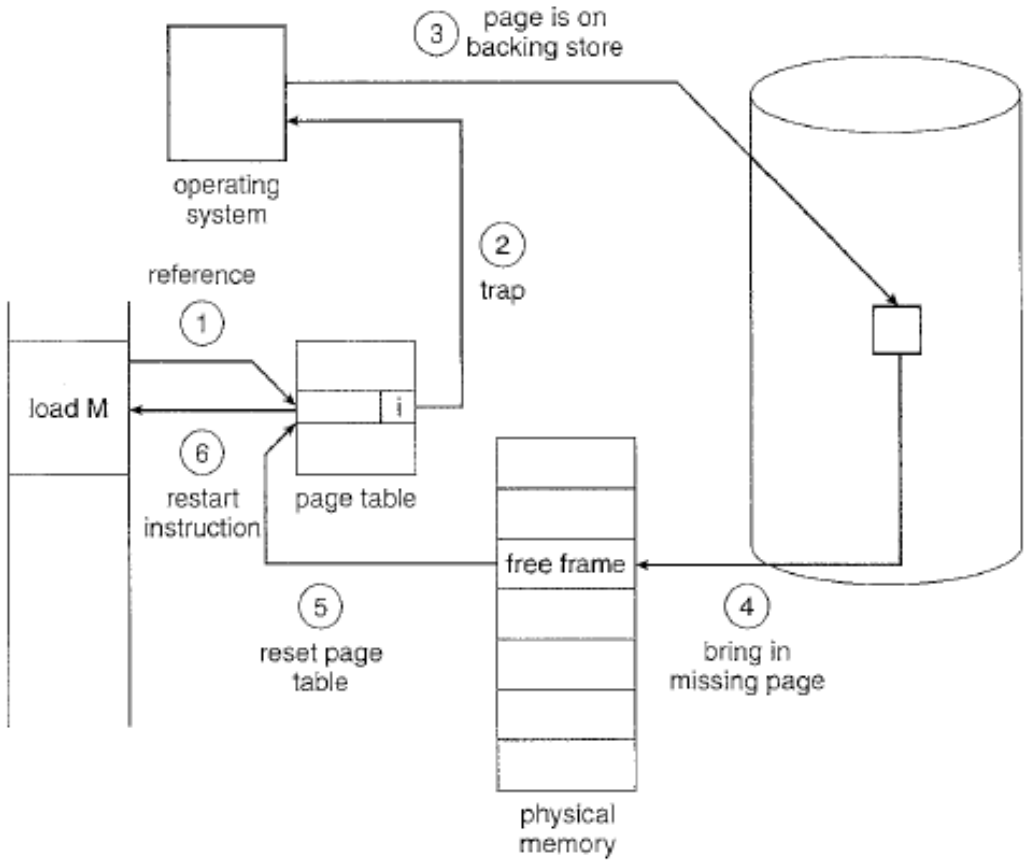
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory
- Lazy swapper never swaps a page into memory unless page will be needed
- Swapper deals with entire processes
- Pager (Lazy swapper) deals with pages

Valid Invalid Bit

When some pages are not in memory



Page Fault Handling

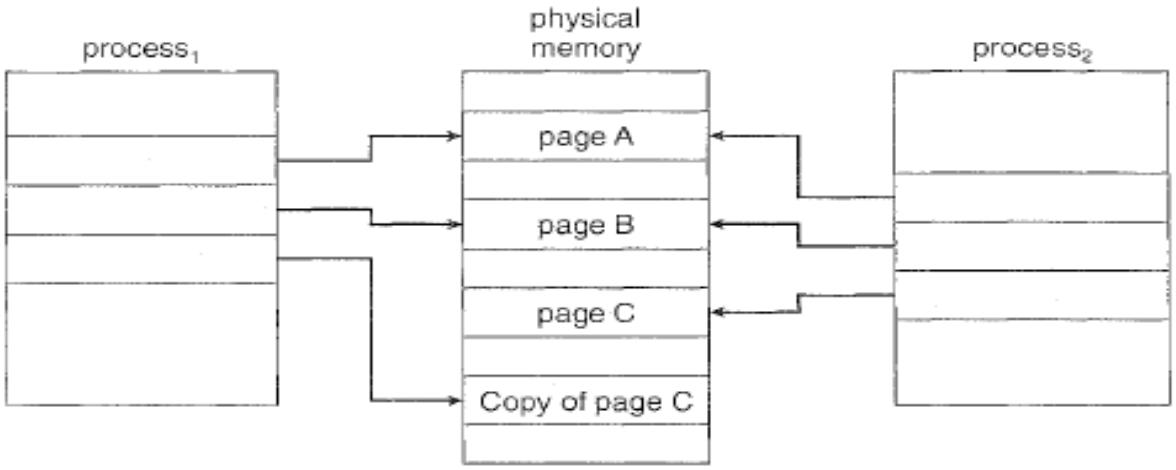


CODING BLOCKS

Copy on Write

Parent and child processes initially share the same pages in memory

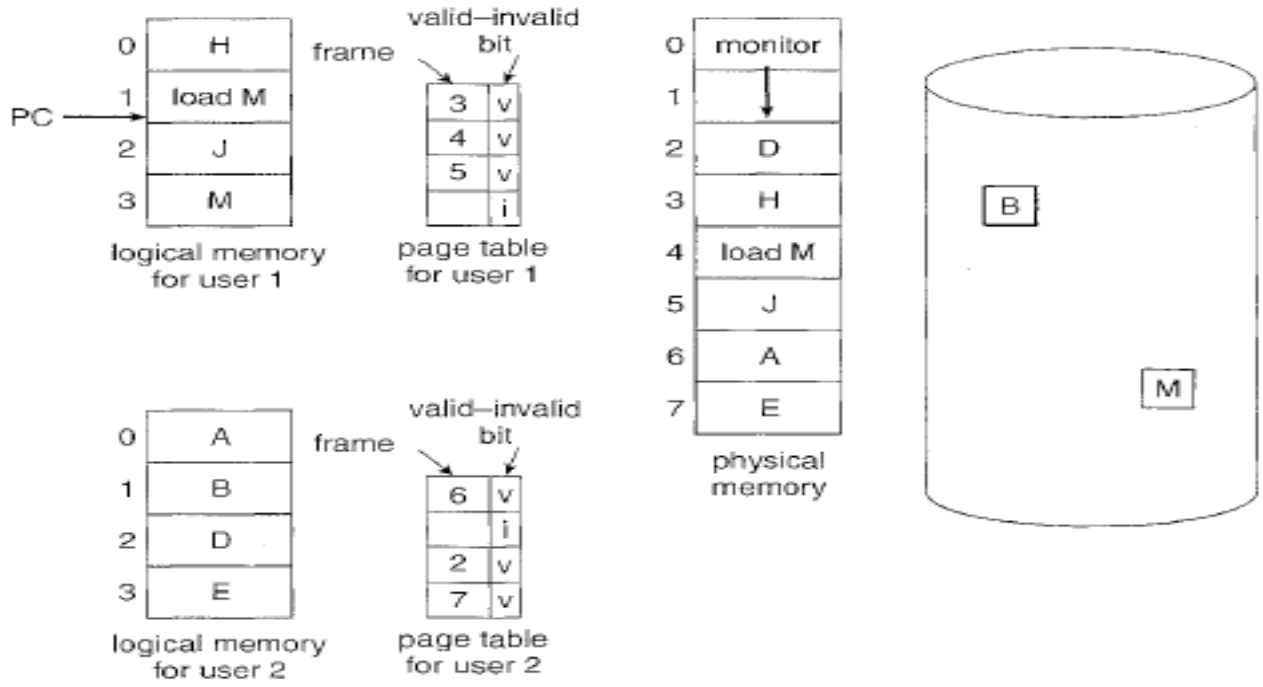
- Only the modified page is copied upon modification occurs
- Free pages are allocated from a pool of zeroed-out pages



CODING BLOCKS

Page Replacement

Find some page in memory not in use, swap it out.



Performance Issue

Because disk I/O is so expensive, we must solve two major problems to implement demand paging.

- Frame-allocation algorithm If we have multiple processes in, we must decide how many frames to allocate to each process.
- Page-replacement algorithm When page replacement is required, we must select the frames that are to be replaced.

Performance

We want an algorithm resulting in lowest page-fault rate

- Is the victim page modified?
- Pick a random page to swap out?
- Pick a page from the faulting process' own pages? Orfrom others?

FIFO Page Replacement

Maintain a linked list (FIFO queue) of all pages in order they came into memory

Page at beginning of list replaced

Disadvantage - The oldest page may be often used

Optimal Page Replacement

Replace page needed at the farthest point in future

- Optimal but not feasible
- Estimate by logging page use on previous runs of process

LRU Page Replacement

Assume recently-used-pages will used again soon replace the page that has not been used for the longest period of time

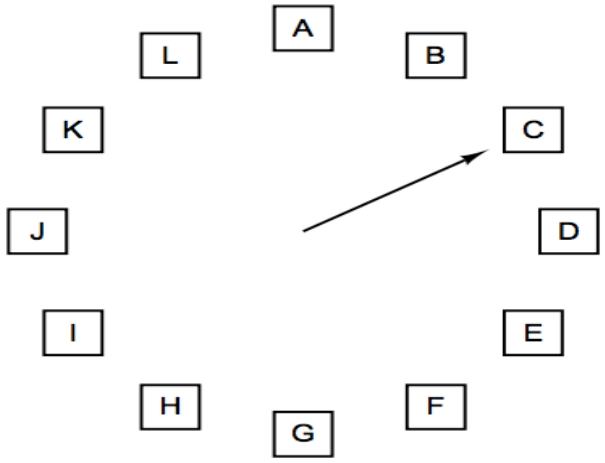
LRU Implementation

- Counters - record the time of the last reference to each page.
  - require a search of the page table to find the LRU page
  - update time-of-use field in the page table
  - choose page with lowest value counter
  - Keep counter in each page table entry

- Linked List - most recently used at top, least (LRU) at bottom.
  - no search for replacement
  - whenever a page is referenced, it's removed from the list and put on the top.

Second Chance Algorithm

- Keeps an extra bit with Page.
- When a page fault occurs, the page the hand is pointing to is inspected. The action depends on the R bit. If 0 – Replace, If 1 – modify to 0 and move to next page.



CODING BLOCKS

Allocation of Frames

Each process needs minimum number of pages

- Fixed Allocation
  - Equal allocation — e.g., 100 frames and 5 processes, give each process 20 frames.
  - Proportional allocation — Allocate according to the size of process –

    $a_i = (s_i / \sum s_i) \times m$

    Si: size of process pi

    m: total number of frames

    ai: frames allocated to pi
- Priority Allocation — Use a proportional allocation scheme using priorities rather than size

Global vs Local Allocation

If process Pi generates a page fault, it can select a

replacement frame

- from its own frames — Local replacement
- from the set of all frames; one process can take a frame from another — Global replacement
  - from a process with lower priority number

Global replacement generally results in greater system throughput.

Thrashing

- CPU not busy ⇒ add more processes
- a process needs more frames ⇒ faulting, and taking frames away from others
- these processes also need these pages ⇒ also faulting, and taking frames away from others ⇒ chain reaction
- more and more processes queueing for the paging device ⇒ ready queue is empty ⇒ CPU has nothing to do ⇒ add more processes ⇒ more page faults
- MMU is busy, but no work is getting done, because processes are busy paging — thrashing