OPERATING SYSTEMS [20MCA15C]

UNIT – III

"Memory Management, Virtual Memory"

FACULTY:

Dr. R. A. Roseline, M.Sc., M.Phil., Ph.D.,

Associate Professor and Head,
Post Graduate and Research Department of Computer Applications,
Government Arts College (Autonomous), Coimbatore – 641 018.

Memory Management

- Background
- Swapping
- Contiguous Allocation
- Paging
- Segmentation
- Segmentation with Paging

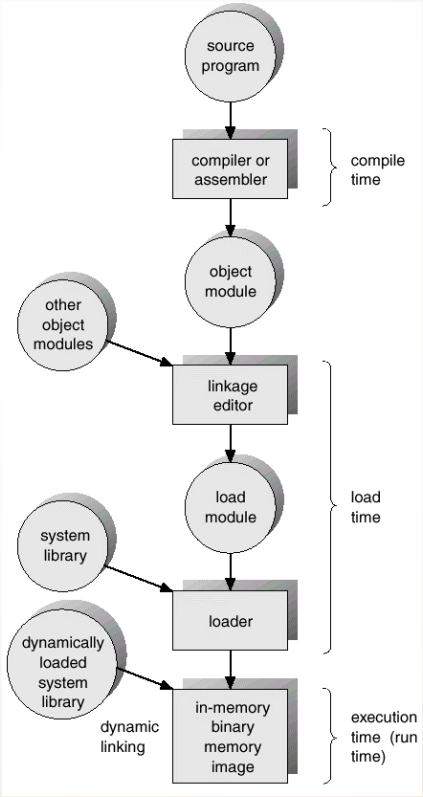
Background

- Program must be brought into memory and placed within a process for it to be run.
- Input queue – collection of processes on the disk that are waiting to be brought into memory to run the program.
- User programs go through several steps before being run.

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



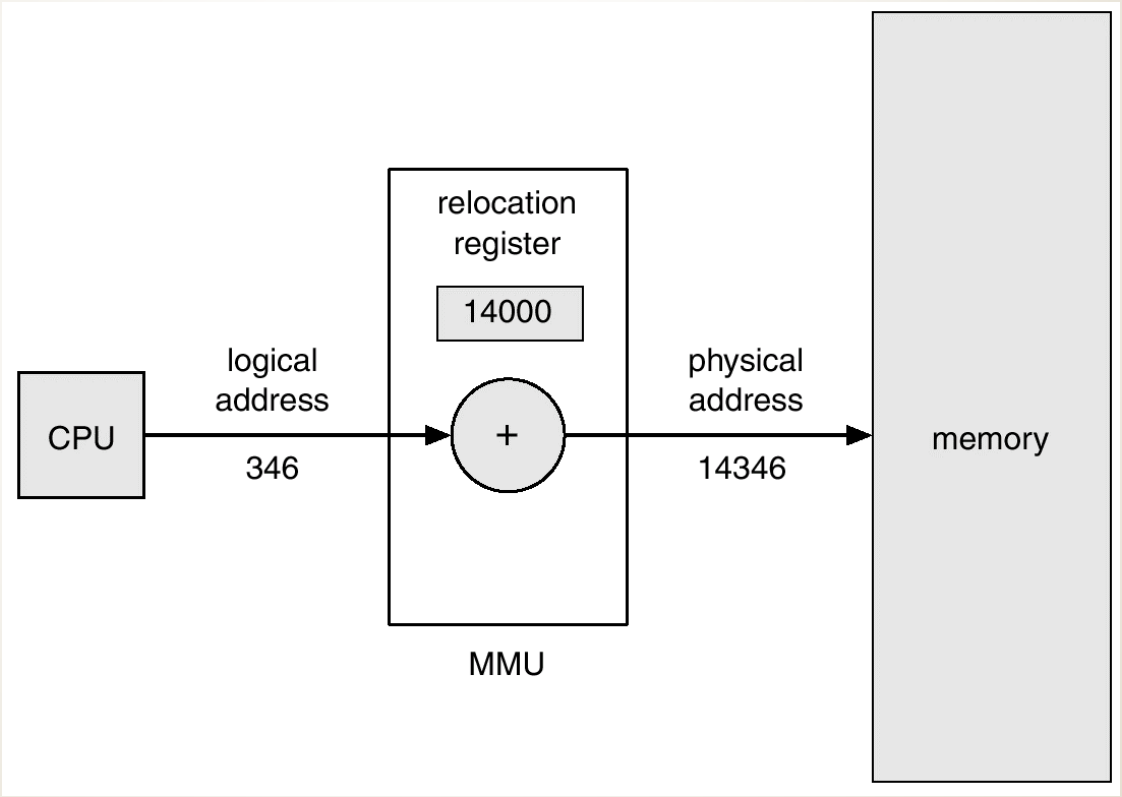
Logical vs. Physical Address Space

- 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.

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address.
- In MMU scheme, 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.

Dynamic relocation using a relocation register



Dynamic Loading

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded.
- Useful when large amounts of code are needed to handle infrequently occurring cases.
- No special support from the operating system is required implemented through program design.

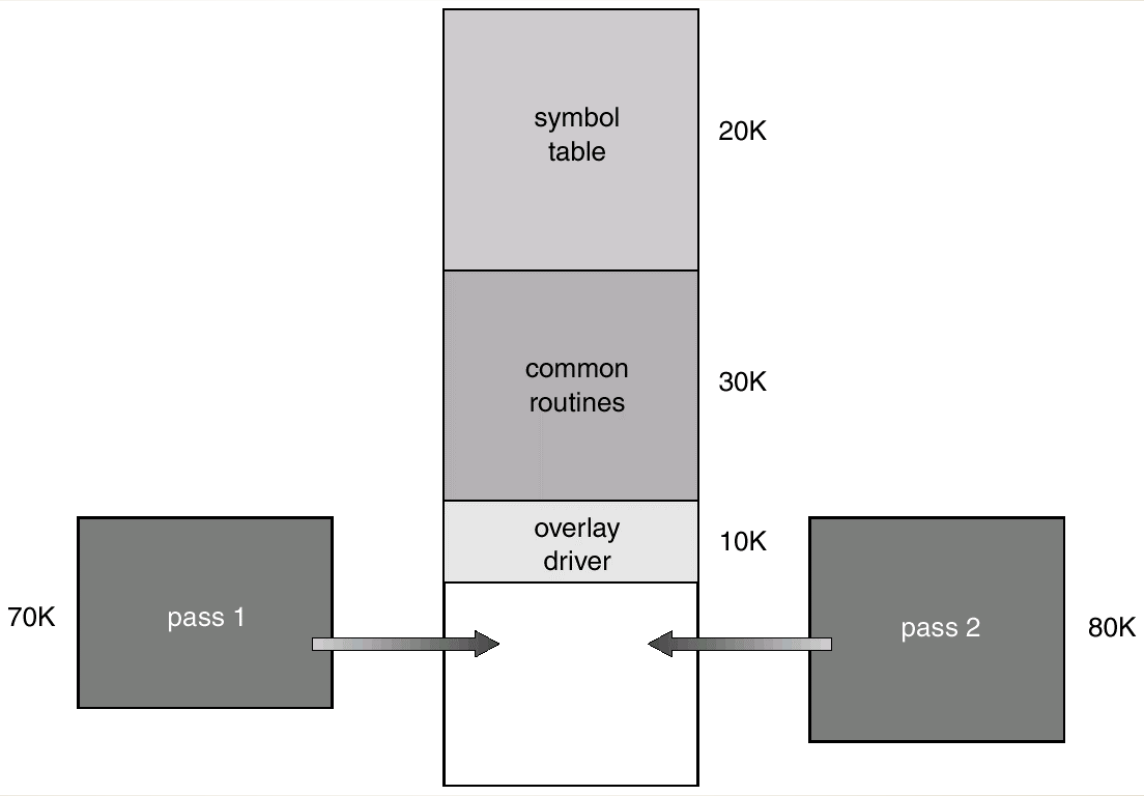
Dynamic Linking

- Linking postponed until execution time.
- Small piece of code, stub, used to locate the appropriate memory-resident library routine.
- Stub replaces itself with the address of the routine, and executes the routine.
- Operating system needed to check if routine is in processes' memory address.
- Dynamic linking is particularly useful for libraries.

Overlays

- Keep in memory only those instructions and data that are needed at any given time.
- Needed when process is larger than amount of memory allocated to it.
- Implemented by user, no special support needed from operating system, programming design of overlay structure is complex

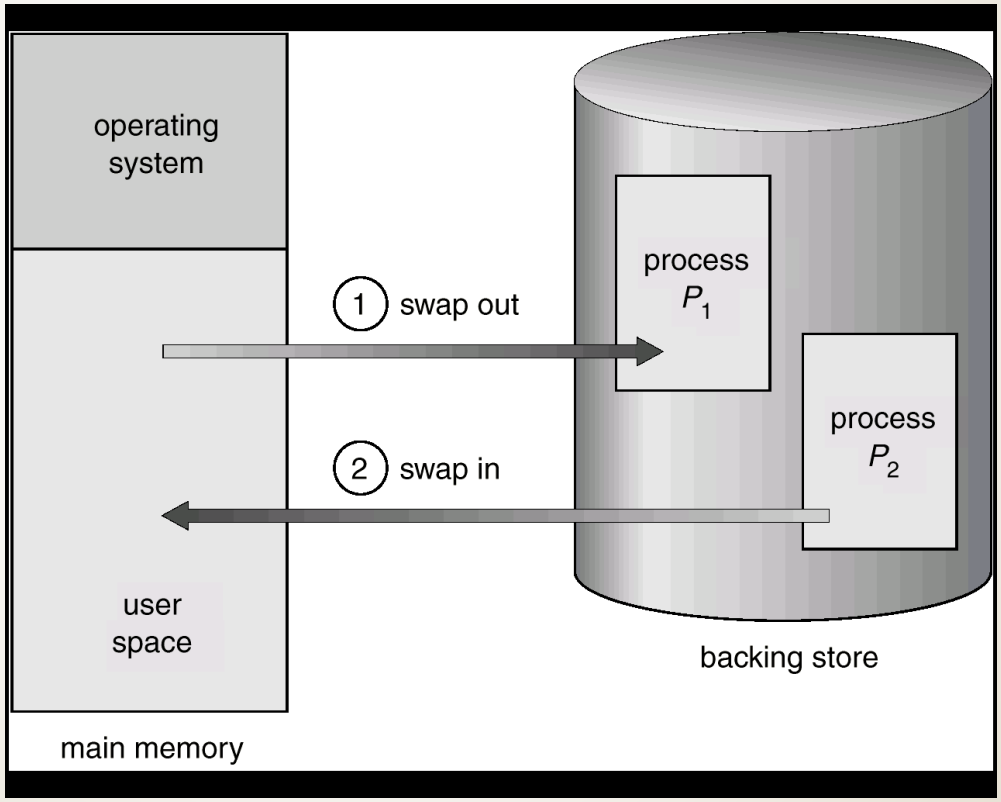
Overlays for a Two-Pass Assembler



Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.
- 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.
- Modified versions of swapping are found on many systems, i.e., UNIX, Linux, and Windows.

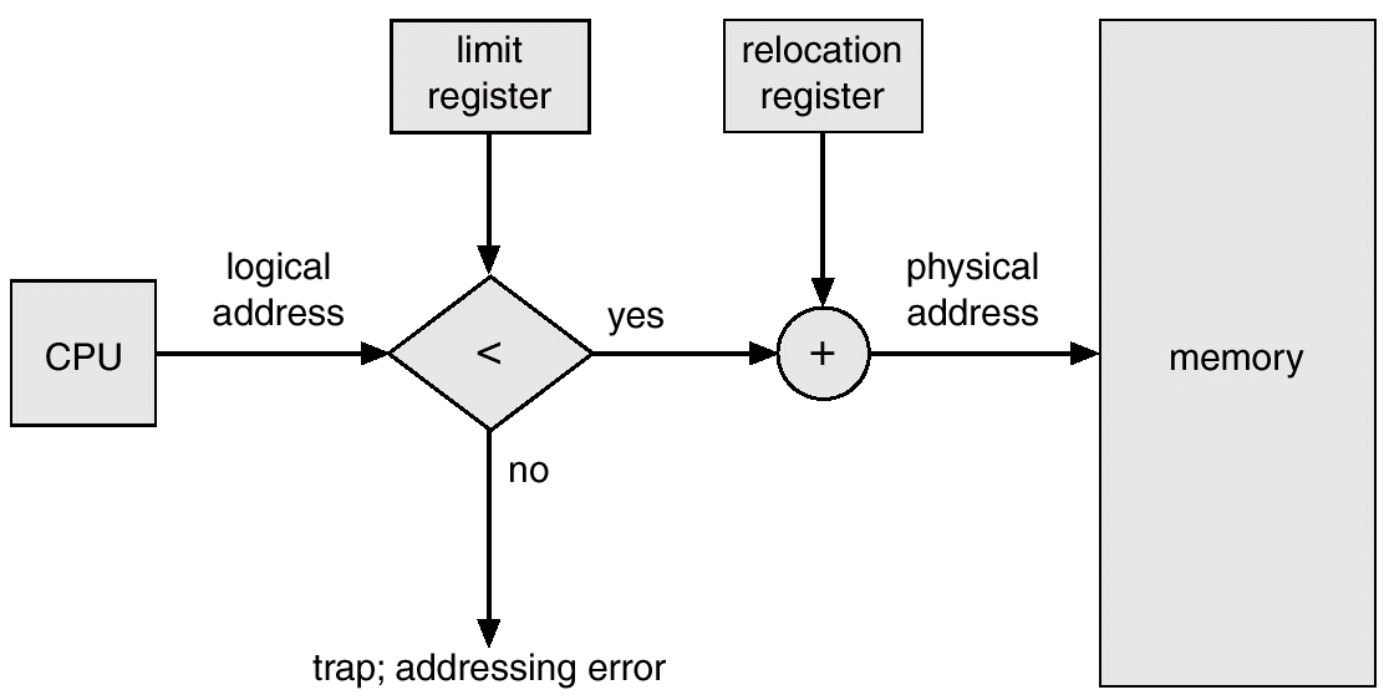
Schematic View of Swapping



Contiguous Allocation

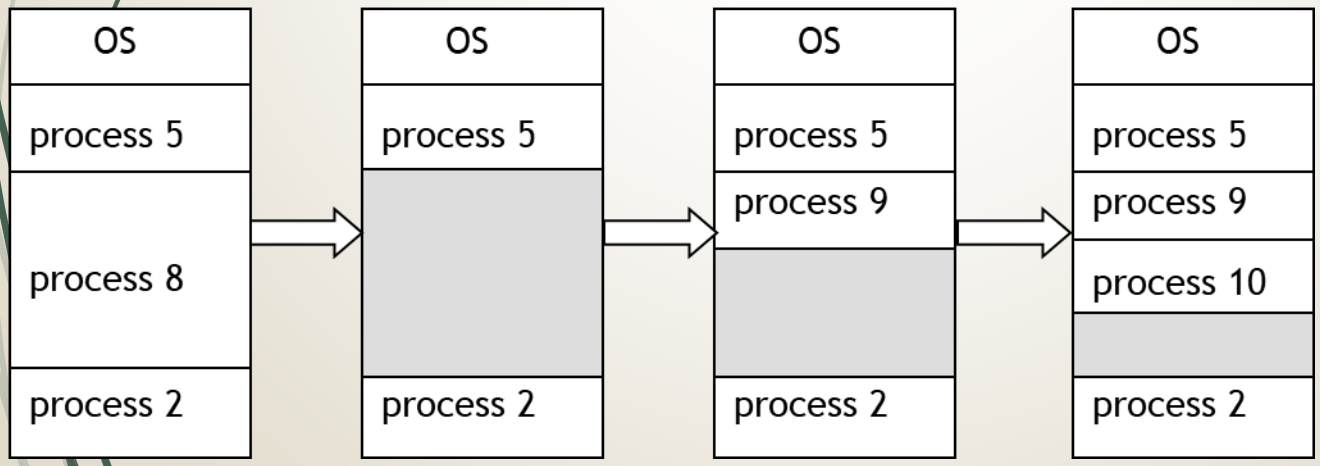
- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes then held in high memory.
- Single-partition allocation
  - Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data.
  - Relocation register contains value of smallest physical address; limit register contains range of logical addresses – each logical address must be less than the limit register.

Hardware Support for Relocation and Limit Registers



Contiguous Allocation (Cont.)

- Multiple-partition allocation
  - 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.
  - Operating system maintains information about:
    a) allocated partitions b) free partitions (hole)



Dynamic Storage-Allocation Problem

- How to satisfy a request of size *n* from a list of free holes.
- First-fit: Allocate the first hole 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 also search entire list. Produces the largest leftover hole.
- First-fit and best-fit better than worst-fit in terms of speed and storage utilization.

Fragmentation

- External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous.
- Internal Fragmentation – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.
- 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.

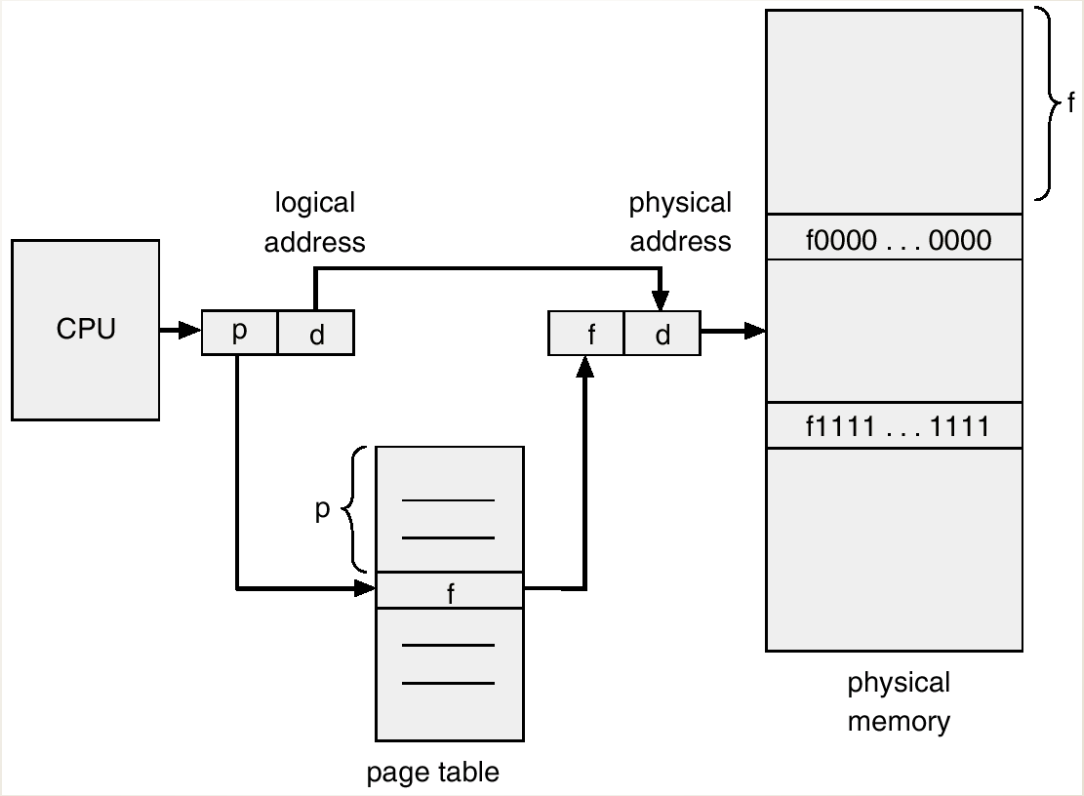
Paging

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available.
- Divide physical memory into fixed-sized blocks called frames (size is power of 2, between 512 bytes and 8192 bytes).
- 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.
- Internal fragmentation.

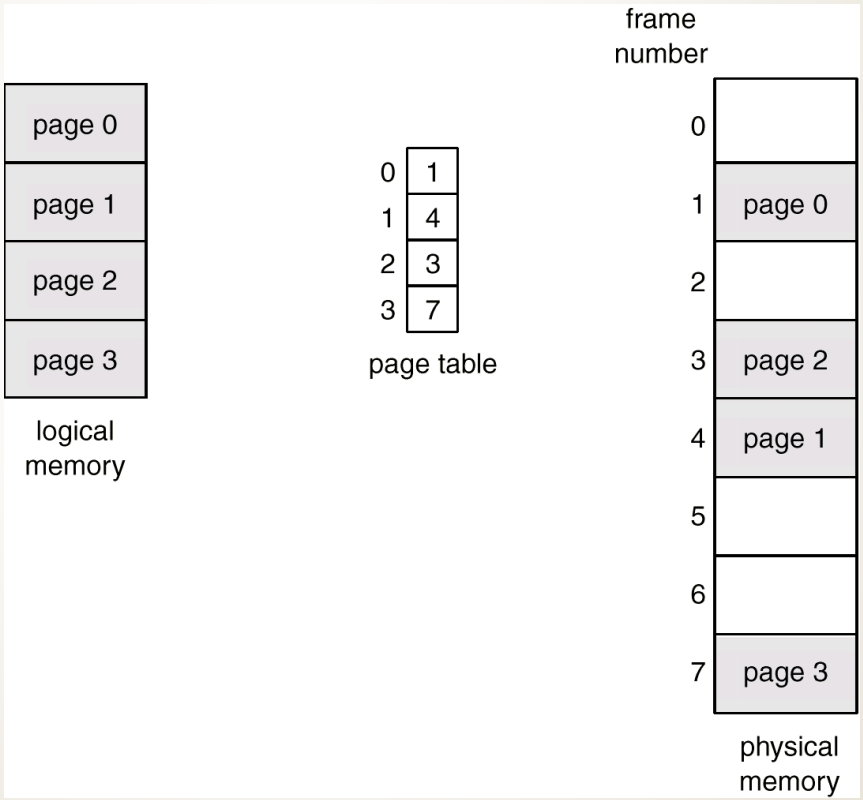
Address Translation Scheme

- 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.

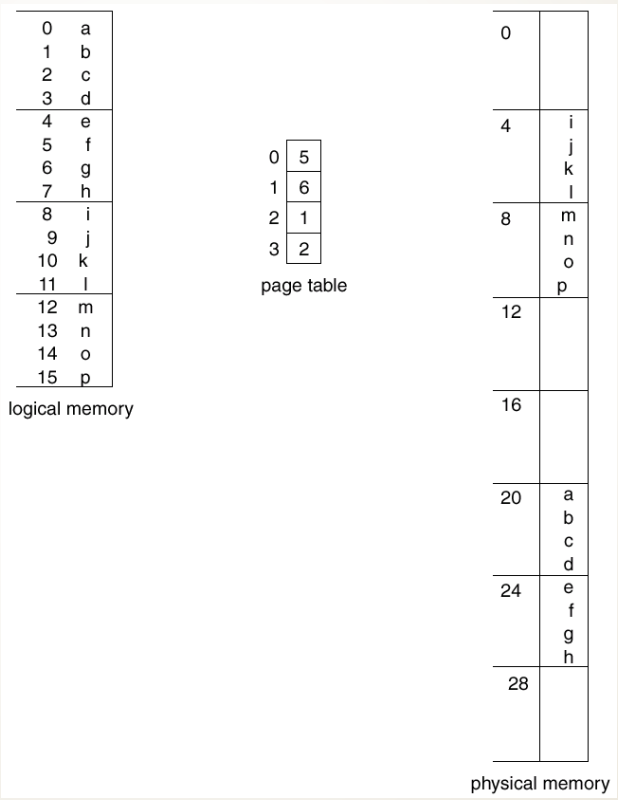
Address Translation Architecture



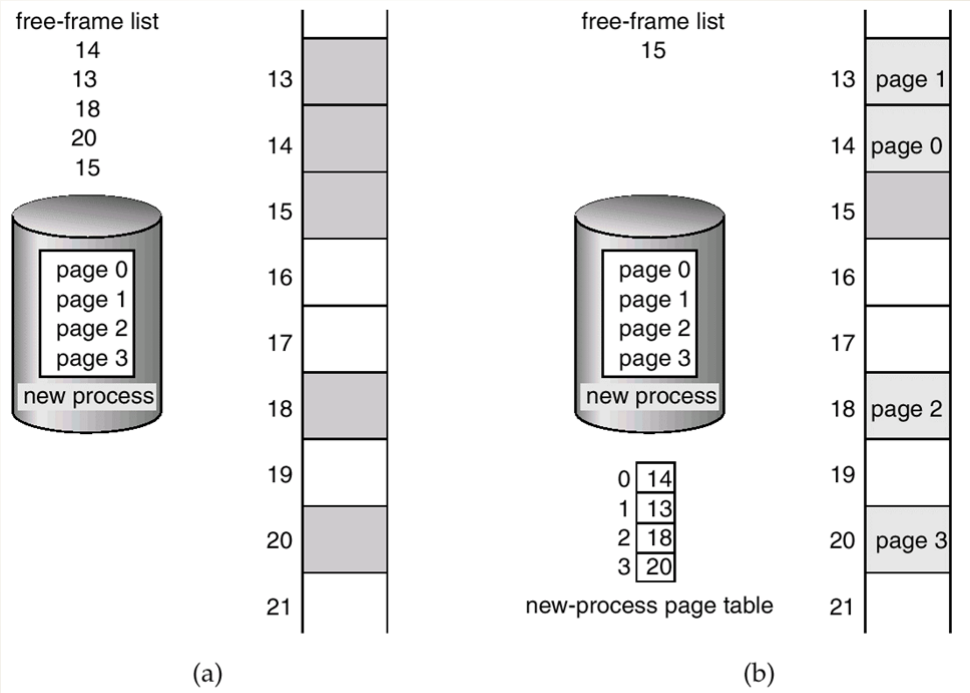
Paging Example



Paging Example



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 (PRLR) 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 associative memory or translation look-aside buffers (TLBs)

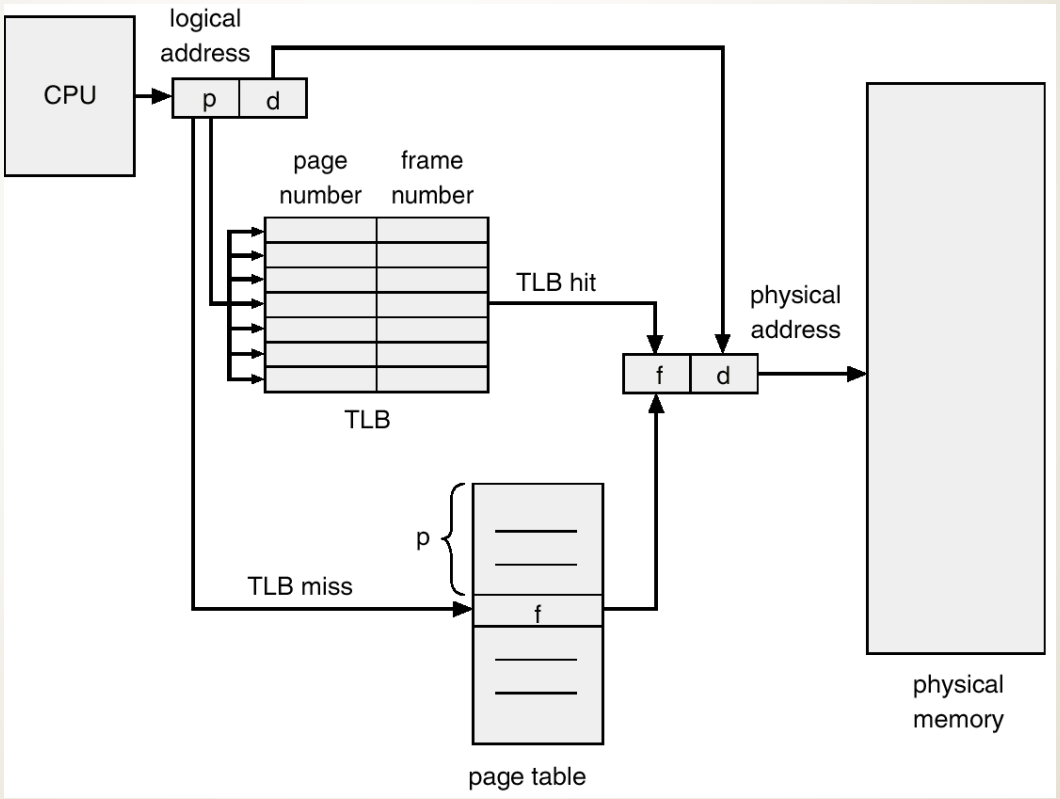
Associative Memory

- Associative memory – parallel search

Page #	Frame #

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

Paging Hardware With TLB



Effective Access Time

- Associative Lookup = ε time unit
- Assume memory cycle time is 1 microsecond
- Hit ratio – percentage of times that a page number is found in the associative registers; ration related to number of associative registers.

  Hit ratio = α
- Effective Access Time (EAT)

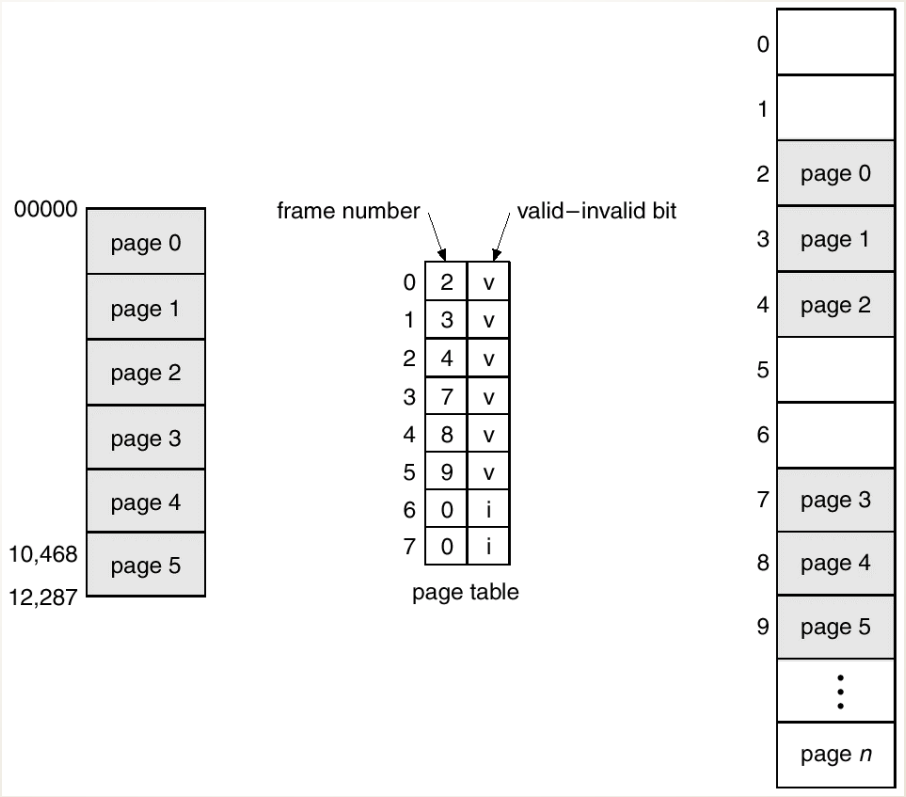
$$EAT = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha)$$

$$= 2 + \varepsilon - \alpha$$

Memory Protection

- Memory protection implemented by associating protection bit with each frame.
- 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.

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



Page Table Structure

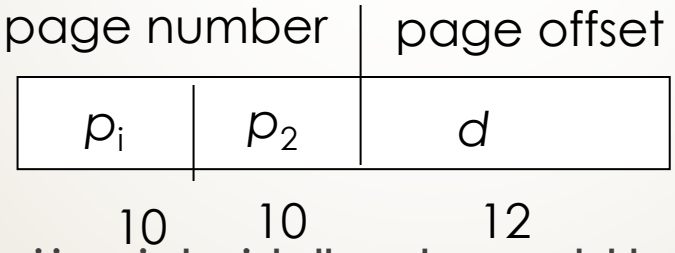
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

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

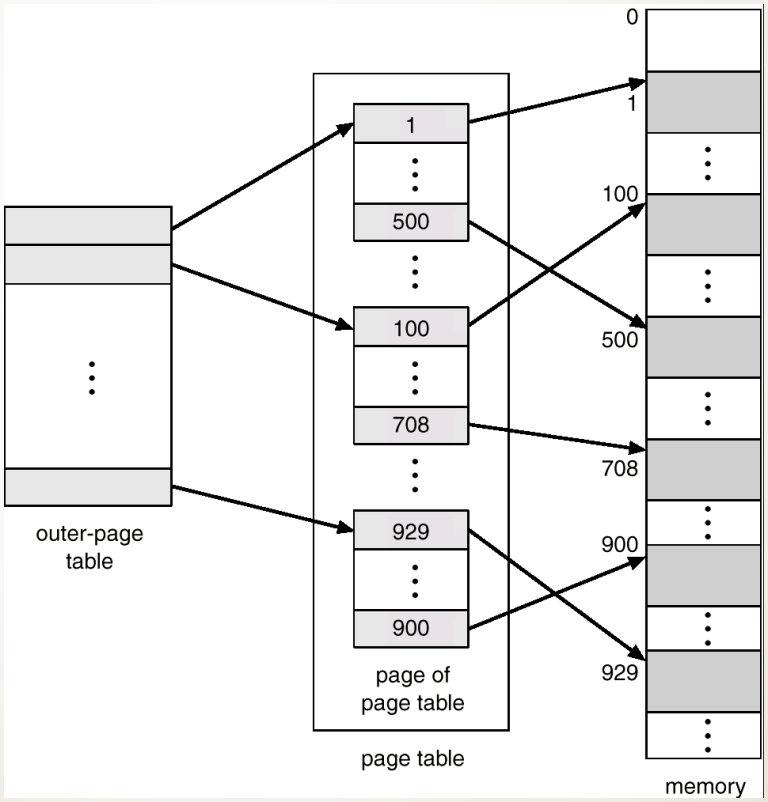
Two-Level Paging Example

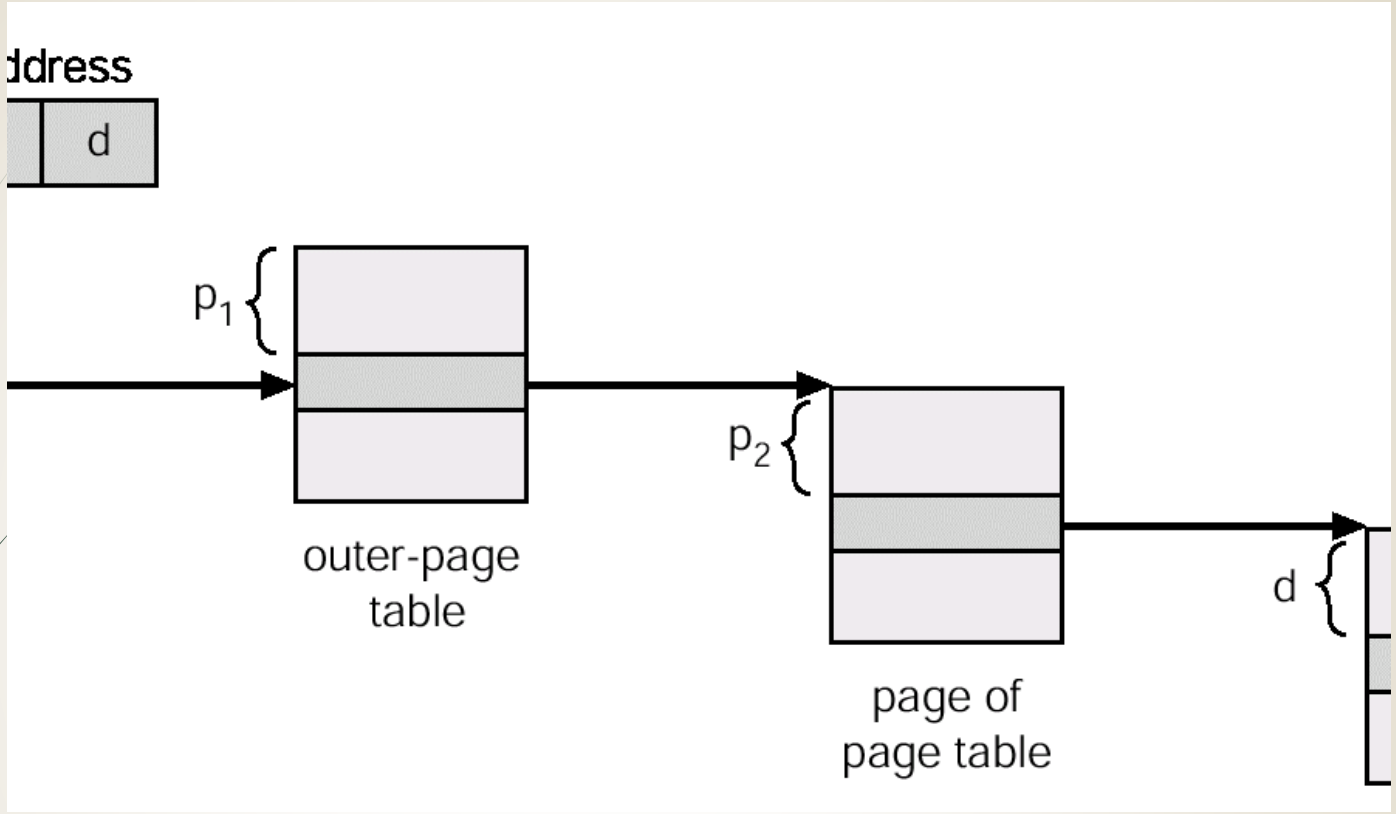
- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page number consisting of 20 bits.
  - a page offset consisting of 12 bits.
- Since the page table is paged, the page number is further divided into:
  - a 10-bit page number.
  - a 10-bit page offset.
- Thus, a logical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



where pi is an index into the outer page table, and p2 is the displacement within the page of the outer page table.

Two-Level Page-Table Scheme





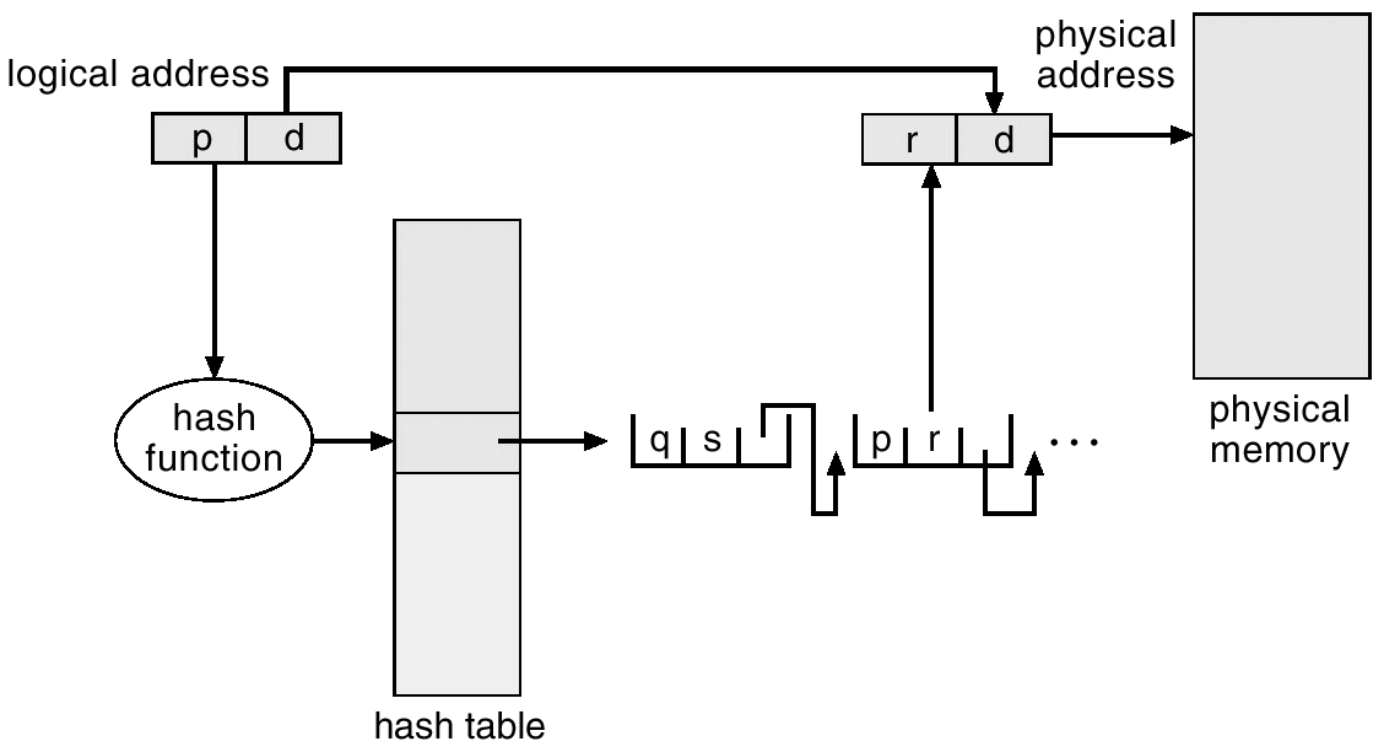
Address-Translation Scheme

Address-translation scheme for a two-level 32-bit paging architecture

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.
- Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.

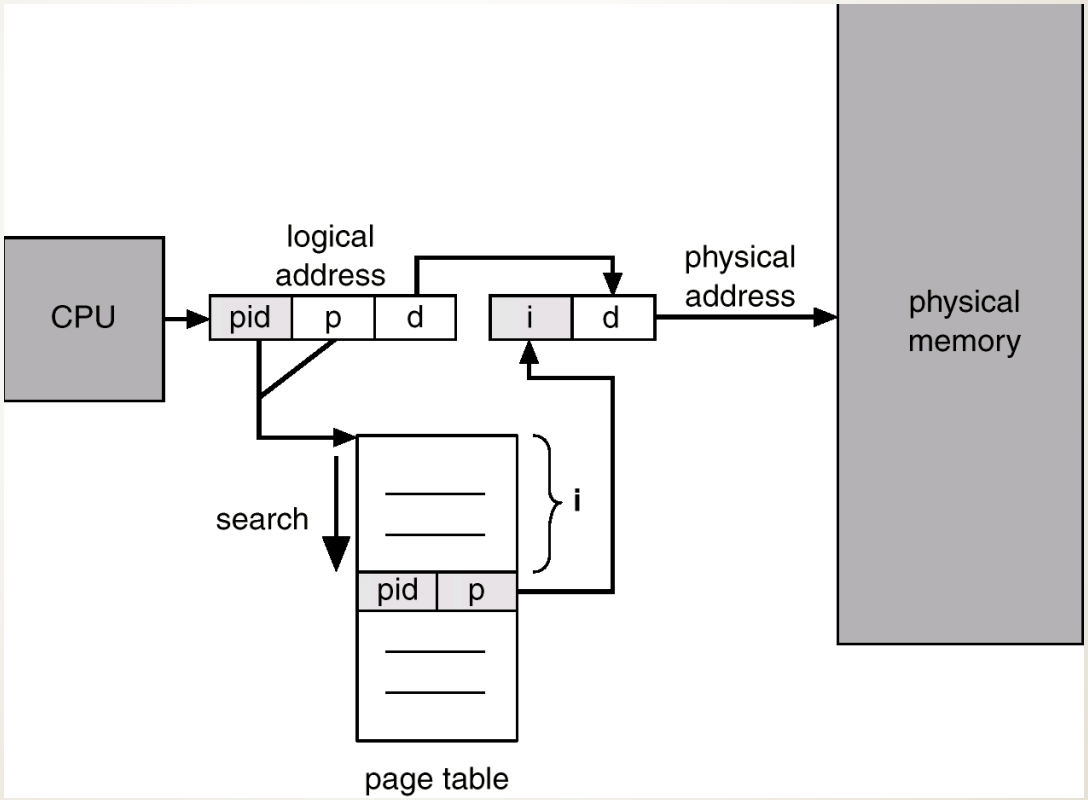
Hashed Page Table



Inverted Page Table

- One entry for each real page of memory.
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.
- Use hash table to limit the search to one — or at most a few — page-table entries.

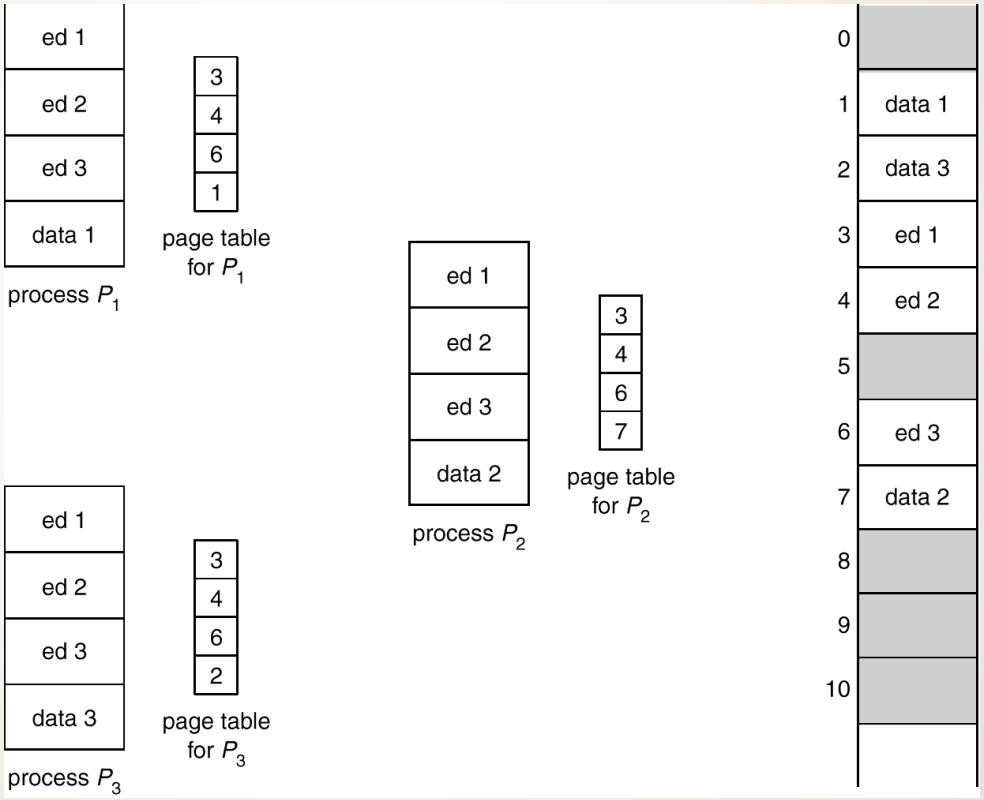
Inverted Page Table Architecture



Shared Pages

- Shared code
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  - Shared code must appear in same location in the logical address space of all processes.
- 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.

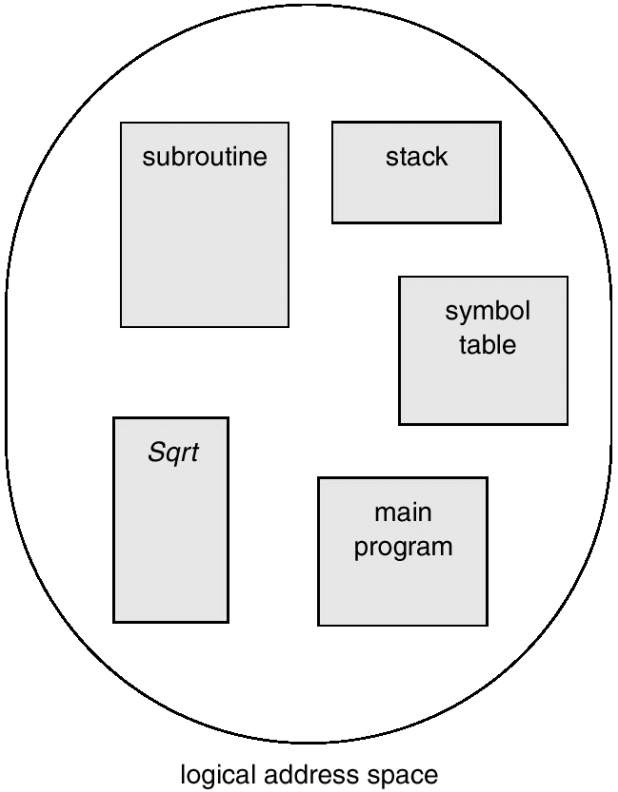
Shared Pages Example



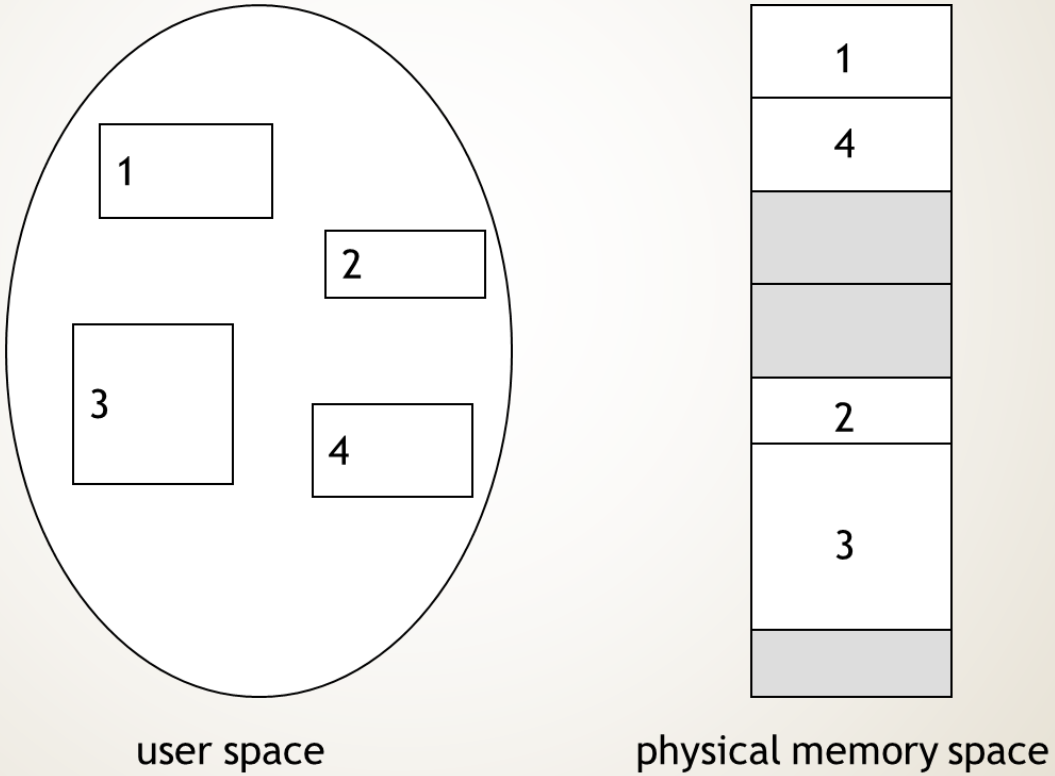
Segmentation

- Memory-management scheme that supports user view of memory.
- A program is a collection of segments. A segment is a logical unit such as:
  - main program,
  - procedure,
  - function,
  - method,
  - object,
  - local variables, global variables,
  - common block,
  - stack,
  - symbol table, arrays

User's View of a Program



Logical View of Segmentation



Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

- Segment table – maps two-dimensional 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.

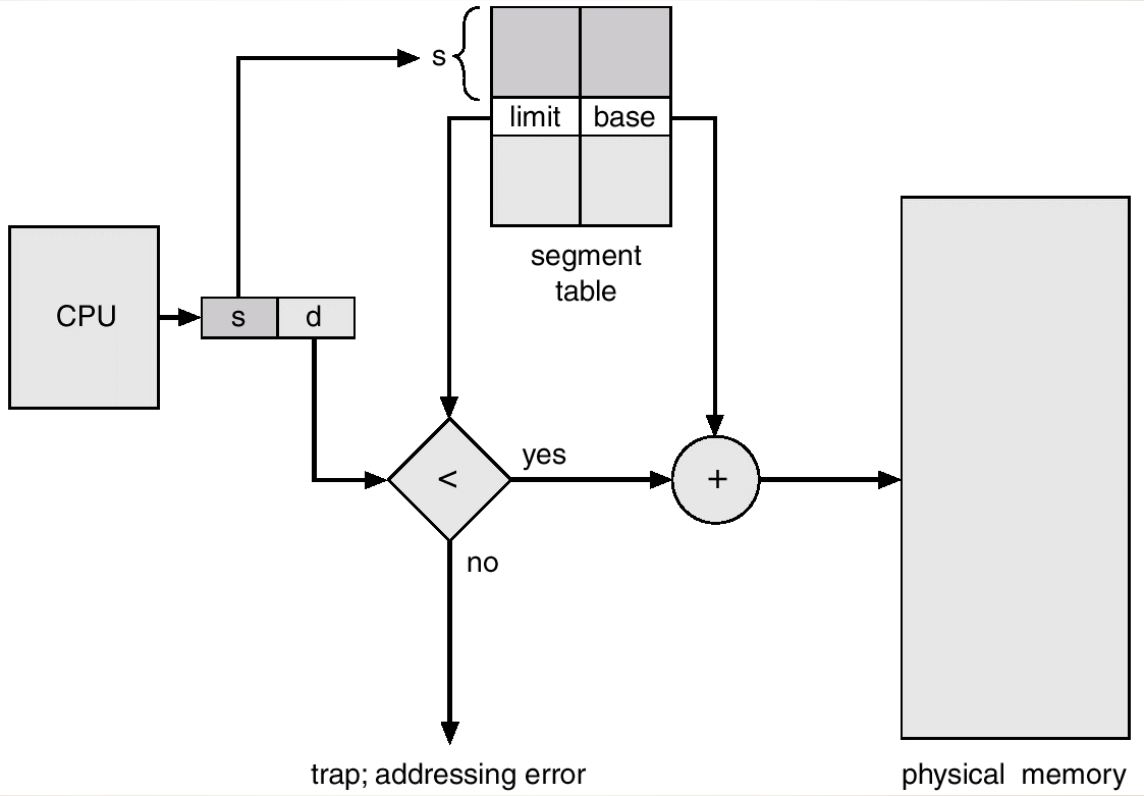
Segmentation Architecture (Cont.)

- Relocation.
  - dynamic
  - by segment table
- Sharing.
  - shared segments
  - same segment number
- Allocation.
  - first fit/best fit
  - external fragmentation

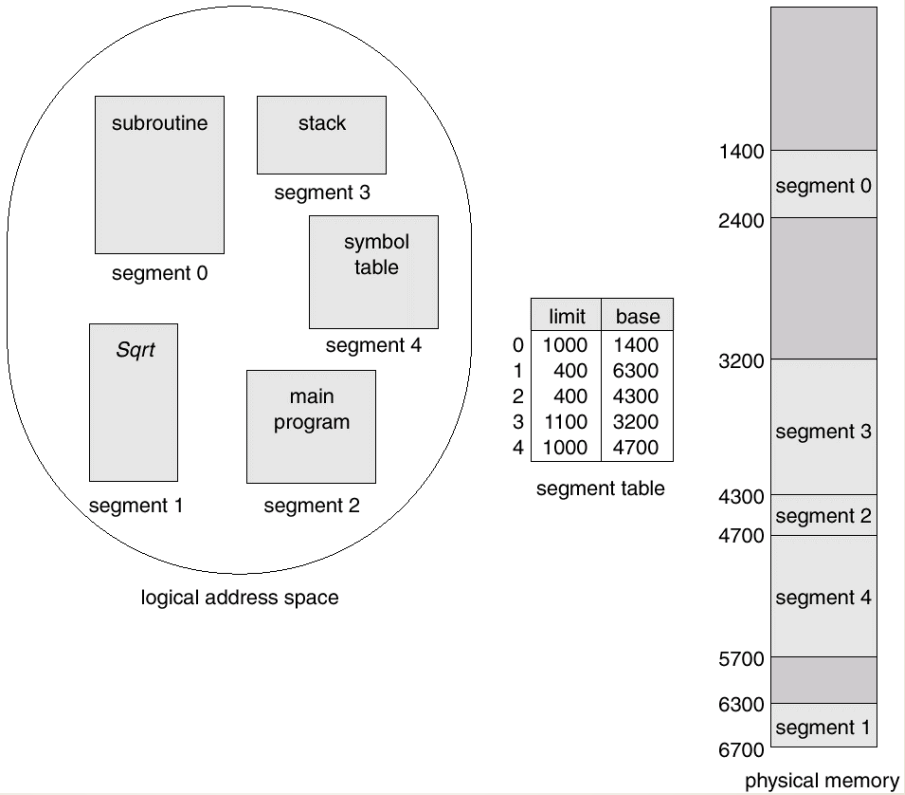
Segmentation Architecture (Cont.)

- Protection. With each entry in segment table associate:
  - validation bit = 0 $\Rightarrow$ illegal segment
  - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level.
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem.
- A segmentation example is shown in the following diagram

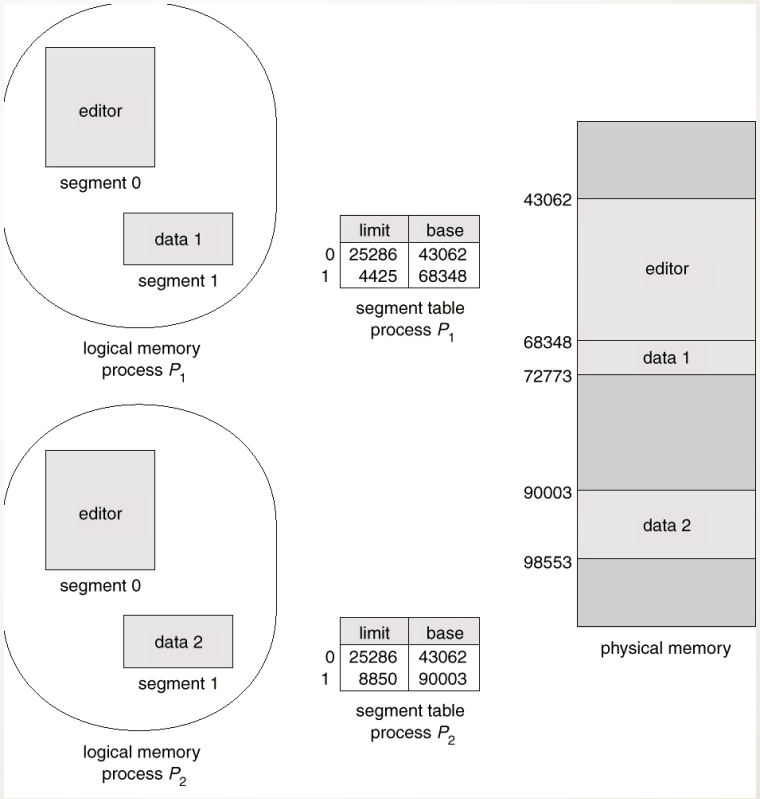
Segmentation Hardware



Example of Segmentation



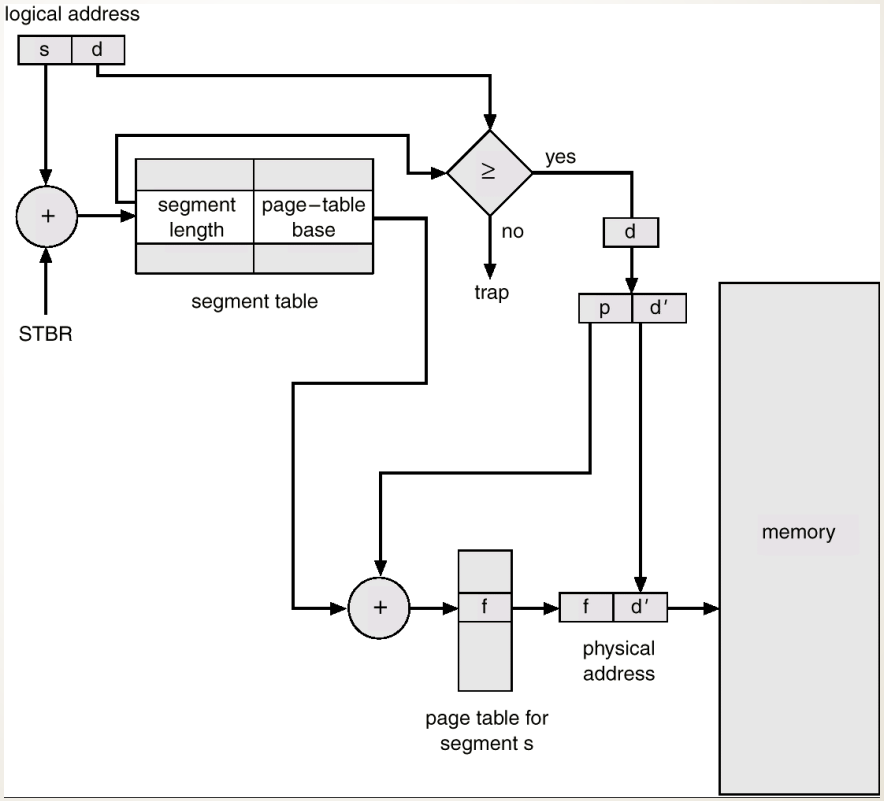
Sharing of Segments



Segmentation with Paging – MULTICS

- The MULTICS system solved problems of external fragmentation and lengthy search times by paging the segments.
- Solution differs from pure segmentation in that the segment-table entry contains not the base address of the segment, but rather the base address of a page table for this segment.

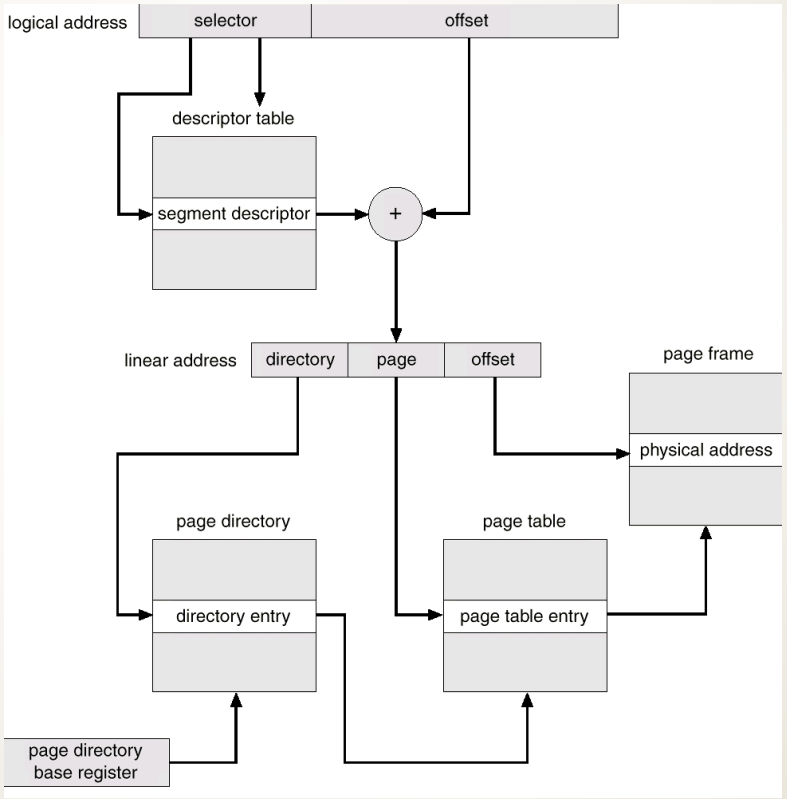
MULTICS Address Translation Scheme



Segmentation with Paging – Intel 386

- As shown in the following diagram, the Intel 386 uses segmentation with paging for memory management with a two-level paging scheme.

Intel 30386 Address Translation



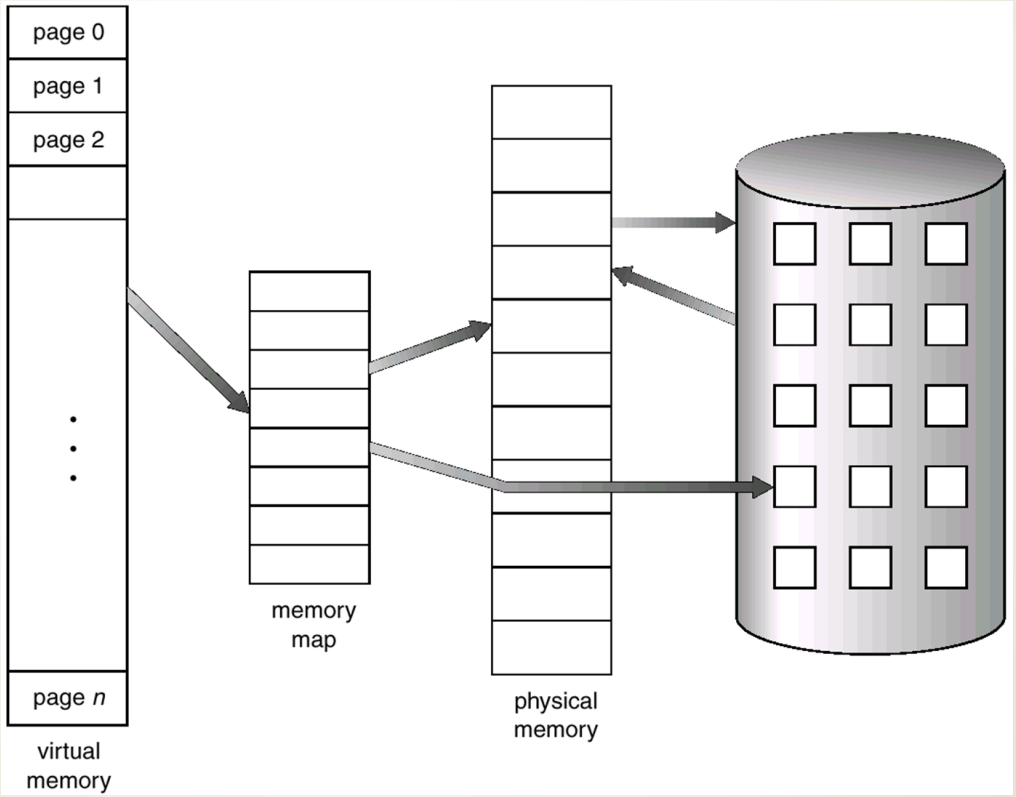
Virtual Memory

- Background
- Demand Paging
- Process Creation
- Page Replacement
- Allocation of Frames
- Thrashing
- Operating System Examples

Background

- Virtual memory – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Allows address spaces to be shared by several processes.
  - Allows for more efficient process creation.
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

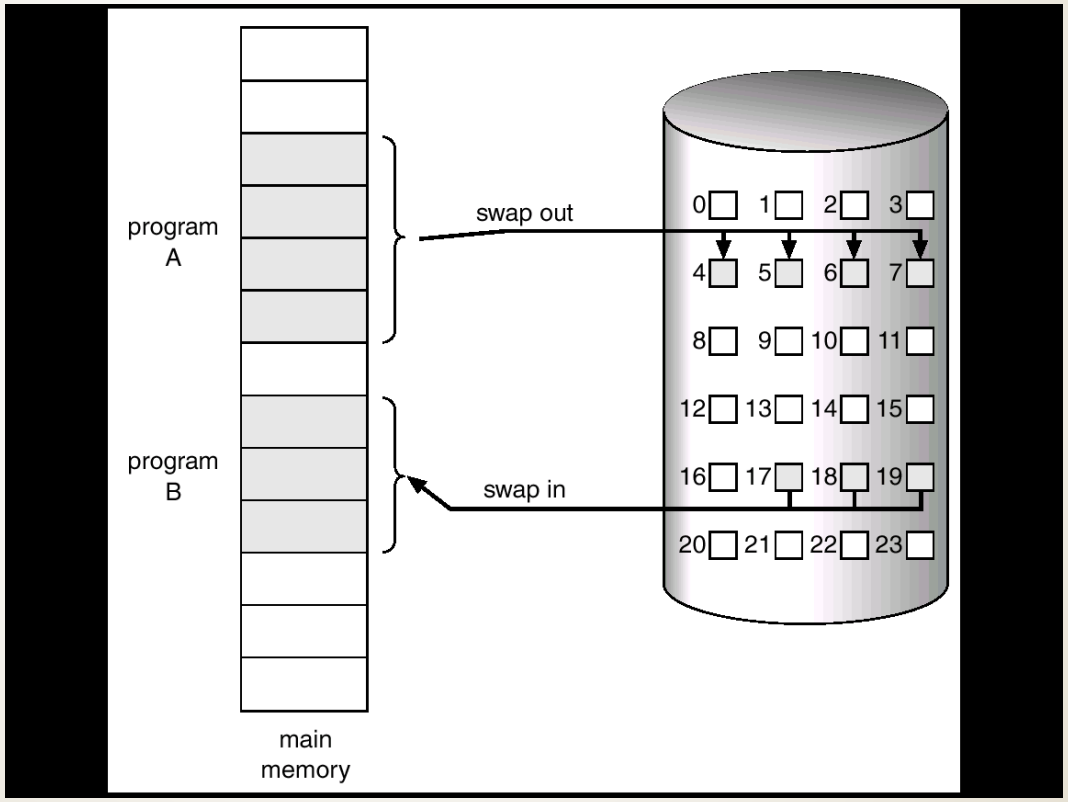
Virtual Memory That is Larger Than Physical Memory



Demand Paging

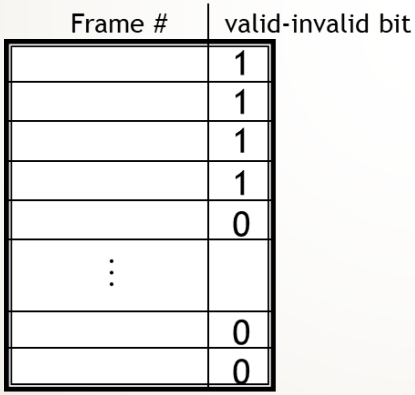
- Bring a page into memory only when it is needed.
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory

Transfer of a Paged Memory to Contiguous Disk Space



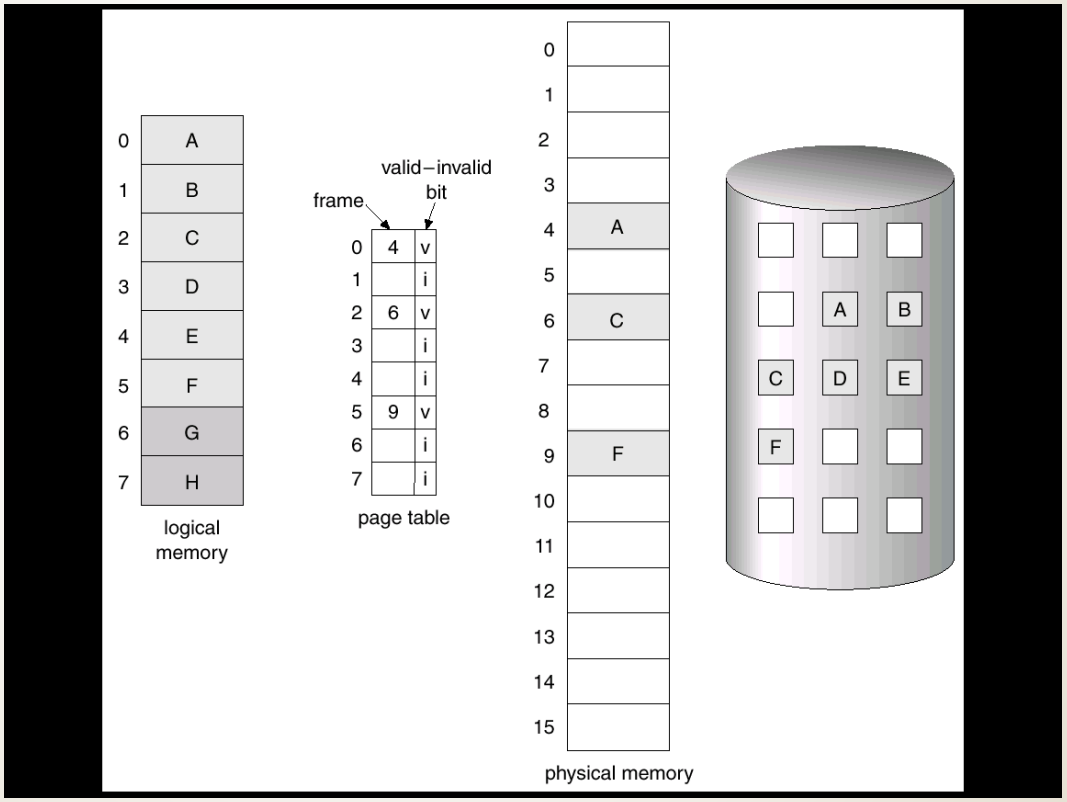
Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated ($1 \Rightarrow$ in-memory, $0 \Rightarrow$ not-in-memory)
- Initially valid–invalid but is set to 0 on all entries.
- Example of a page table snapshot.



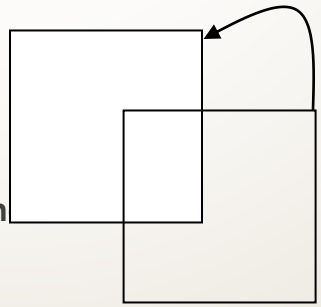
- During address translation, if valid–invalid bit in page table entry is $0 \Rightarrow$ page fault.

Page Table When Some Pages Are Not in Main Memory

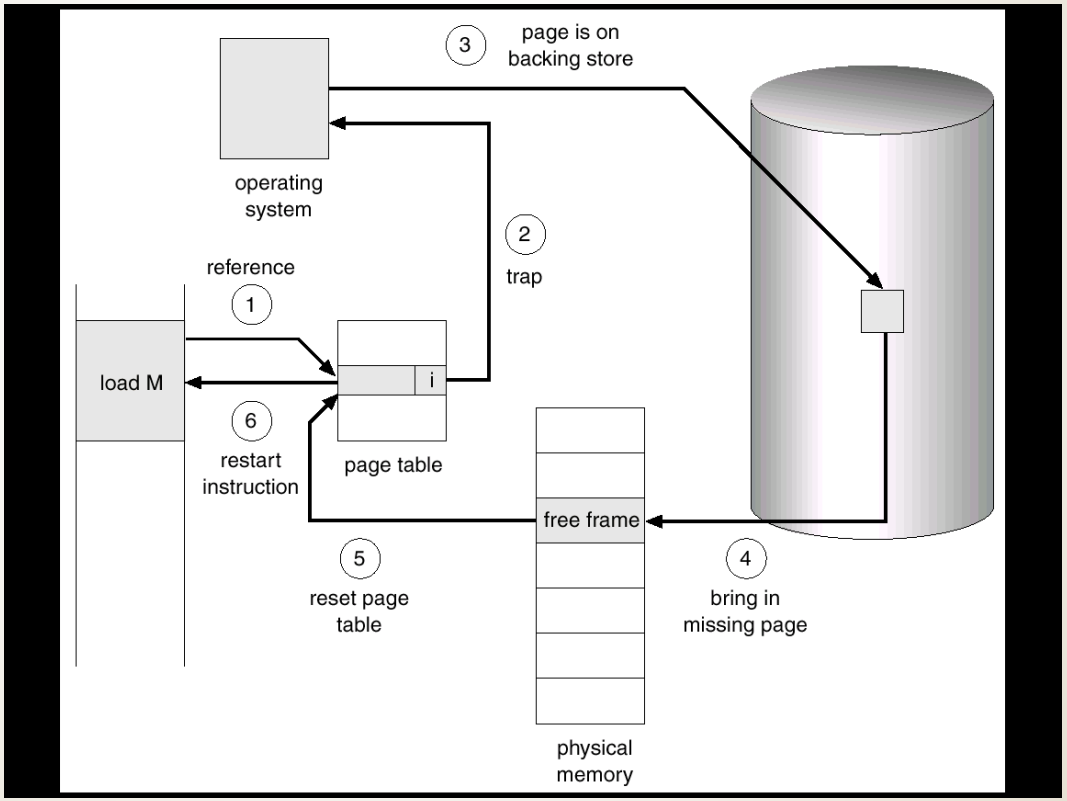


Page Fault

- If there is ever a reference to a page, first reference will trap to OS ⇒ page fault
- OS looks at another table to decide:
  - Invalid reference ⇒ abort.
  - Just not in memory.
- Get empty frame.
- Swap page into frame.
- Reset tables, validation bit = 1.
- Restart instruction: Least Recently Used
  - block move
  - auto increment/decrem



Steps in Handling a Page Fault



What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, swap it out.
  - algorithm
  - performance – want an algorithm which will result in minimum number of page faults.
- Same page may be brought into memory several times.

Performance of Demand Paging

- Page Fault Rate $0 \leq p \leq 1.0$
  - if p = 0 no page faults
  - if p = 1, every reference is a fault
- Effective Access Time (EAT)

  EAT = (1 – p) x memory access

  + p (page fault overhead

  + [swap page out]

  + swap page in

  + restart overhead)

Demand Paging Example

- Memory access time = 1 microsecond
- 50% of the time the page that is being replaced has been modified and therefore needs to be swapped out.
- Swap Page Time = 10 msec = 10,000 msec

$$EAT = (1 - p) \times 1 + p\ (15000)$$

$$1 + 15000P \quad \text{(in msec)}$$

Process Creation

- Virtual memory allows other benefits during process creation:
- - Copy-on-Write
- - Memory-Mapped Files

Copy-on-Write

- Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory.

  If either process modifies a shared page, only then is the page copied.

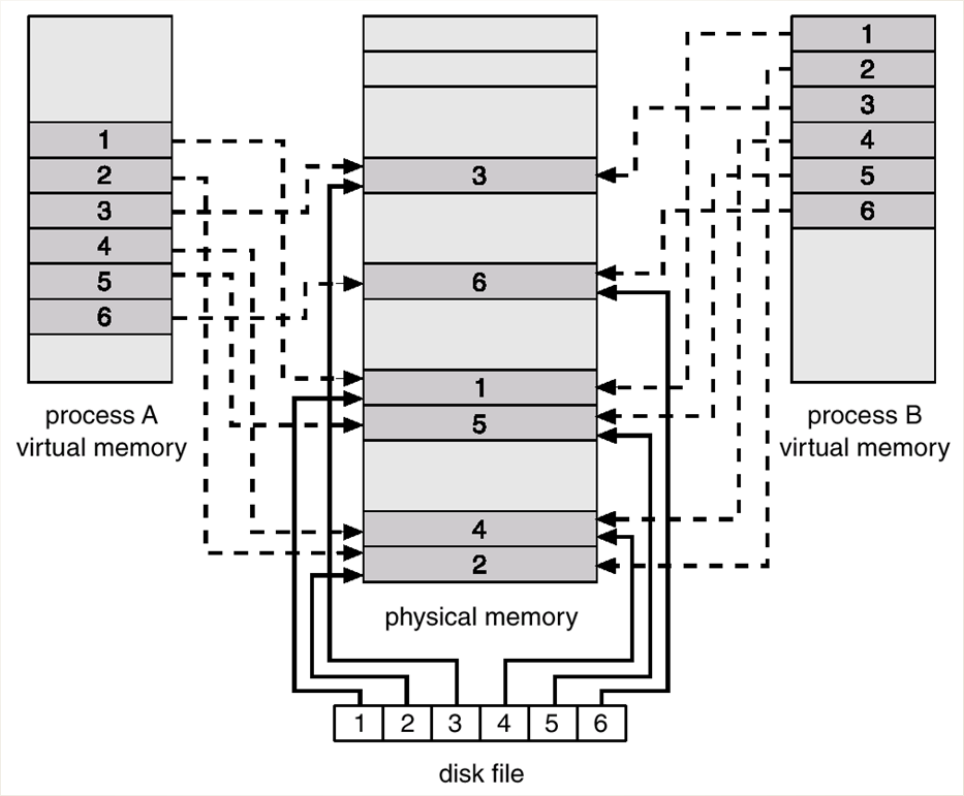
- COW allows more efficient process creation as only modified pages are copied.

- Free pages are allocated from a pool of zeroed-out pages.

Memory-Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as routine memory access by mapping a disk block to a page in memory.
- A file is initially read using demand paging. A page-sized portion of the file is read from the file system into a physical page. Subsequent reads/writes to/from the file are treated as ordinary memory accesses.
- Simplifies file access by treating file I/O through memory rather than read() write() system calls.
- Also allows several processes to map the same file allowing the pages in memory to be shared.

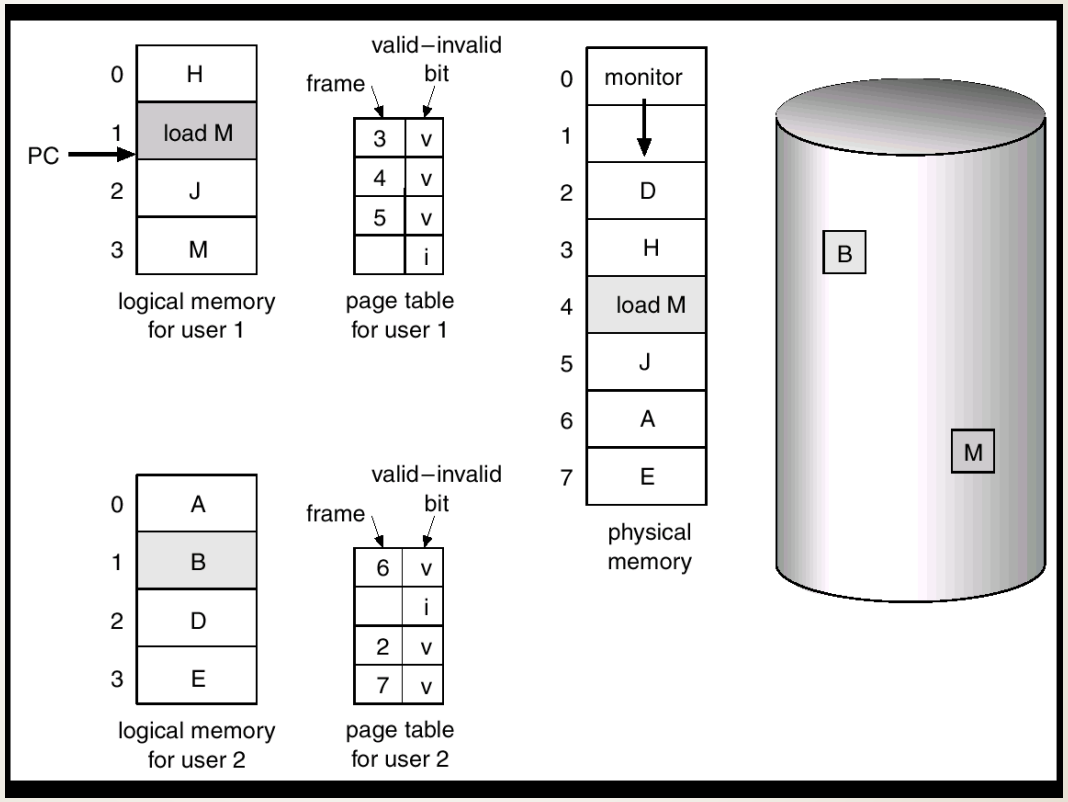
Memory Mapped Files



Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
- Use modify (dirty) bit to reduce overhead of page transfers – only modified pages are written to disk.
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

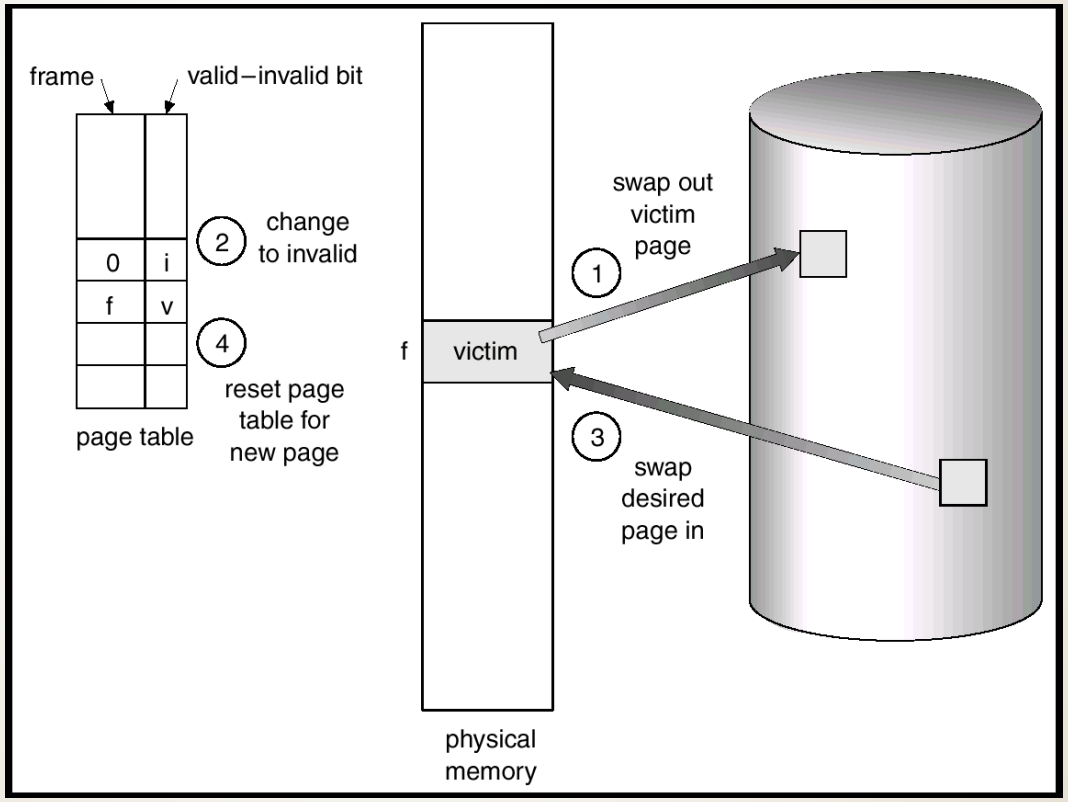
Need For Page Replacement



Basic Page Replacement

- Find the location of the desired page on disk.
- Find a free frame:
  - If there is a free frame, use it.
  - If there is no free frame, use a page replacement algorithm to select a victim frame.
- Read the desired page into the (newly) free frame. Update the page and frame tables.
- Restart the process.

Page Replacement

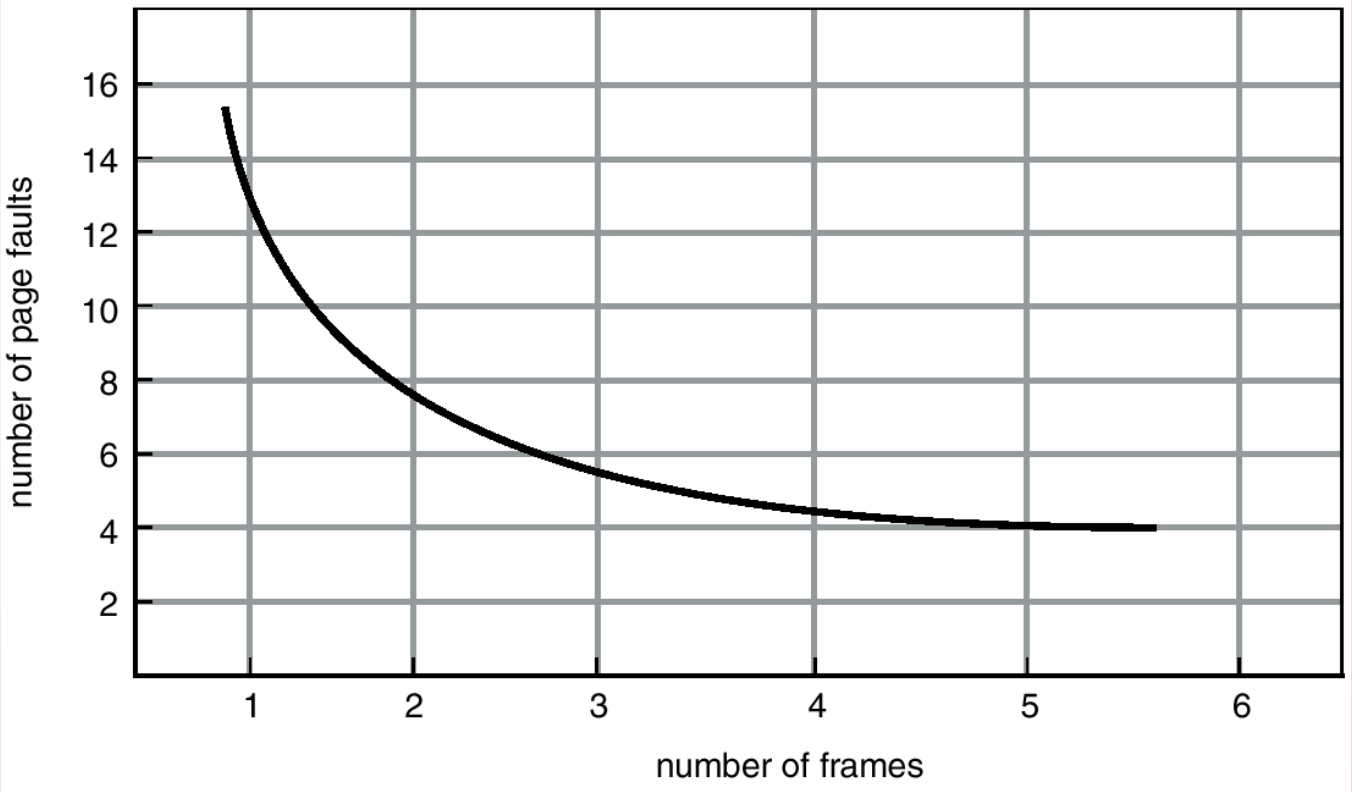


Page Replacement Algorithms

- Want lowest page-fault rate.
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
- In all our examples, the reference string is

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Graph of Page Faults Versus The Number of Frames



First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

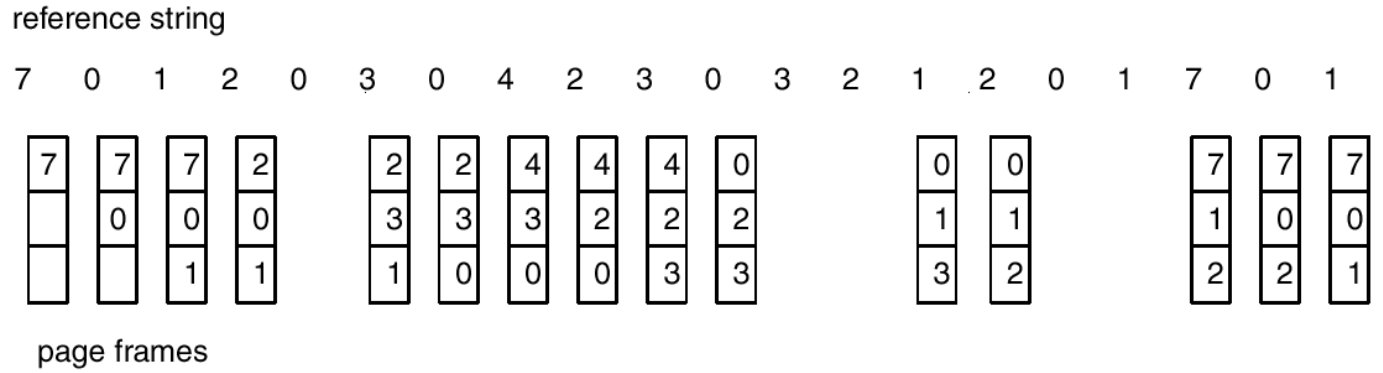
1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

- 4 frames

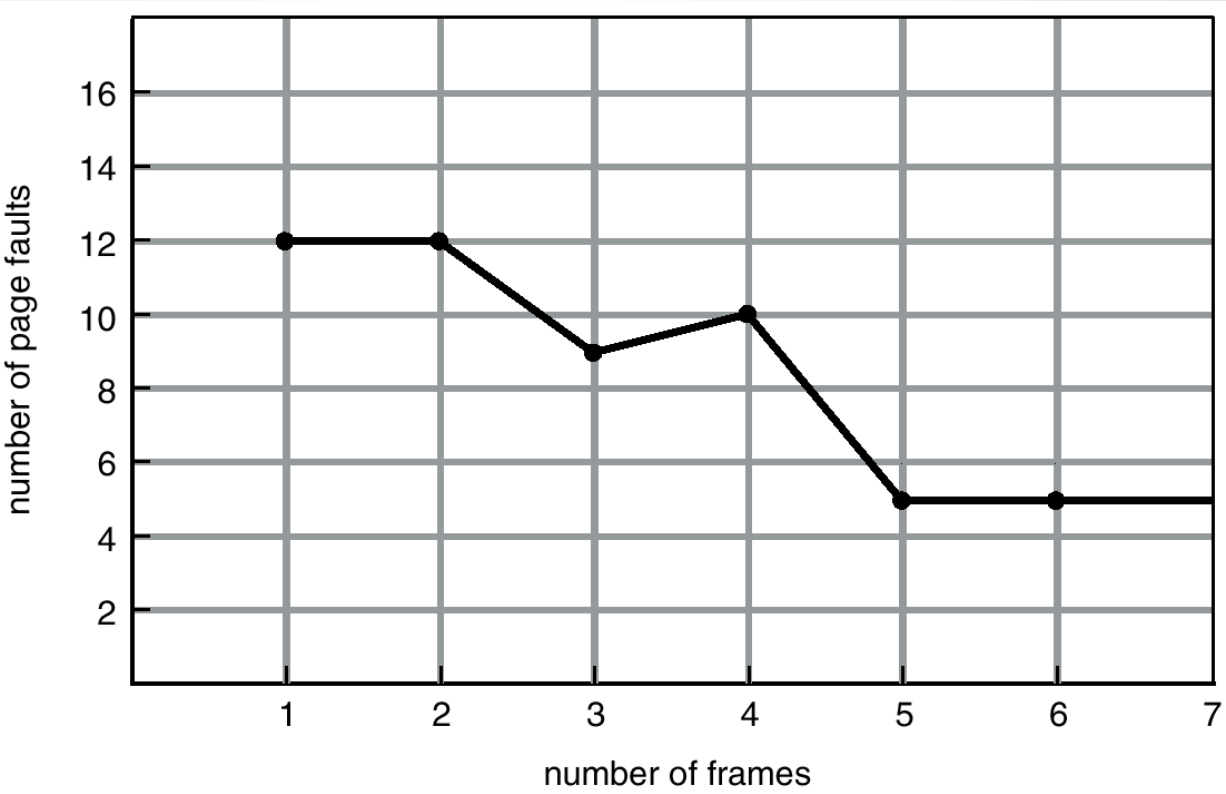
1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

- FIFO Replacement – Belady's Anomaly
  - more frames ⇒ less page faults

FIFO Page Replacement



FIFO Illustrating Belady's Anamoly



Optimal Algorithm

- Replace page that will not be used for longest period of time.
- 4 frames example
- 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

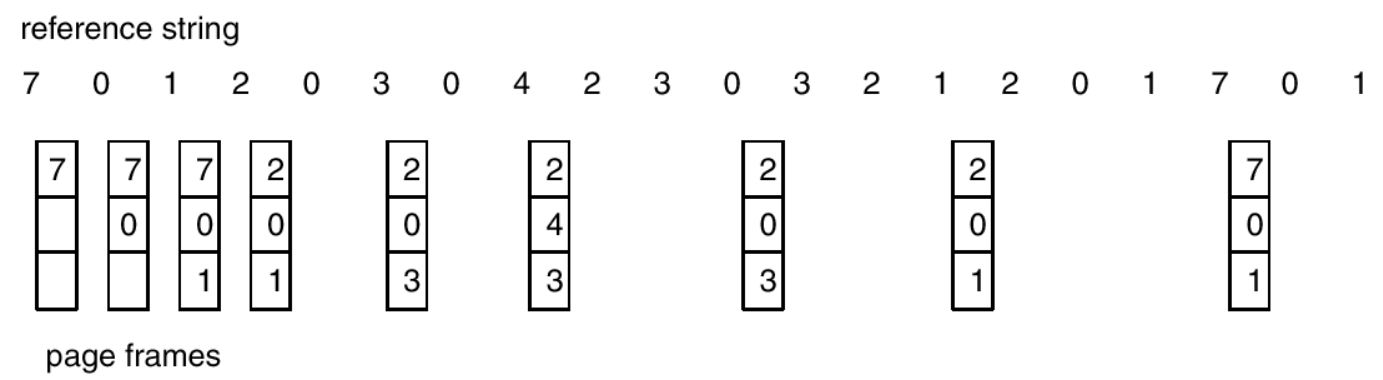
4

5

6 page faults

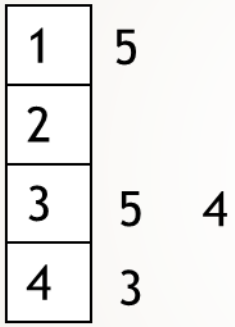
- How do you know this?
- Used for measuring how well your algorithm performs.

Optimal Page Replacement



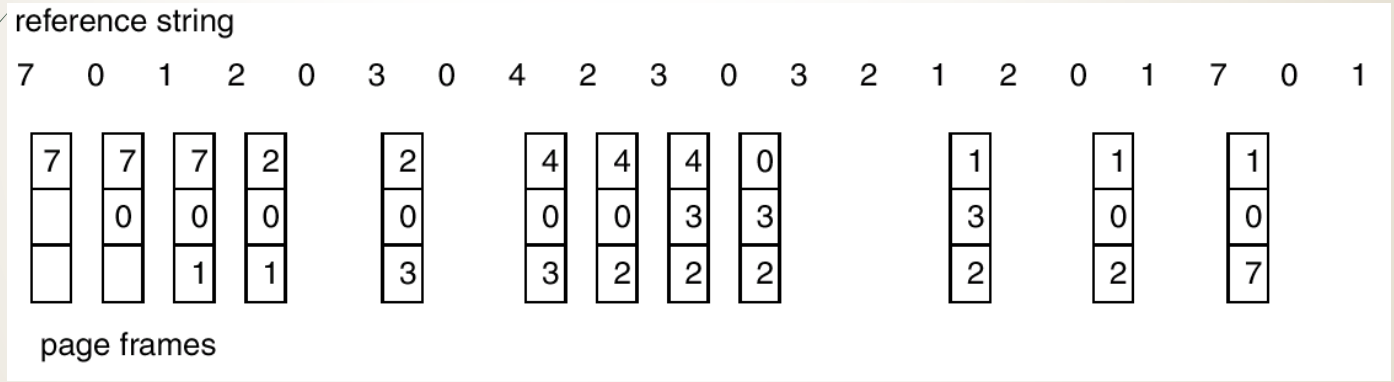
Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
  - When a page needs to be changed, look at the counters to determine which are to change.

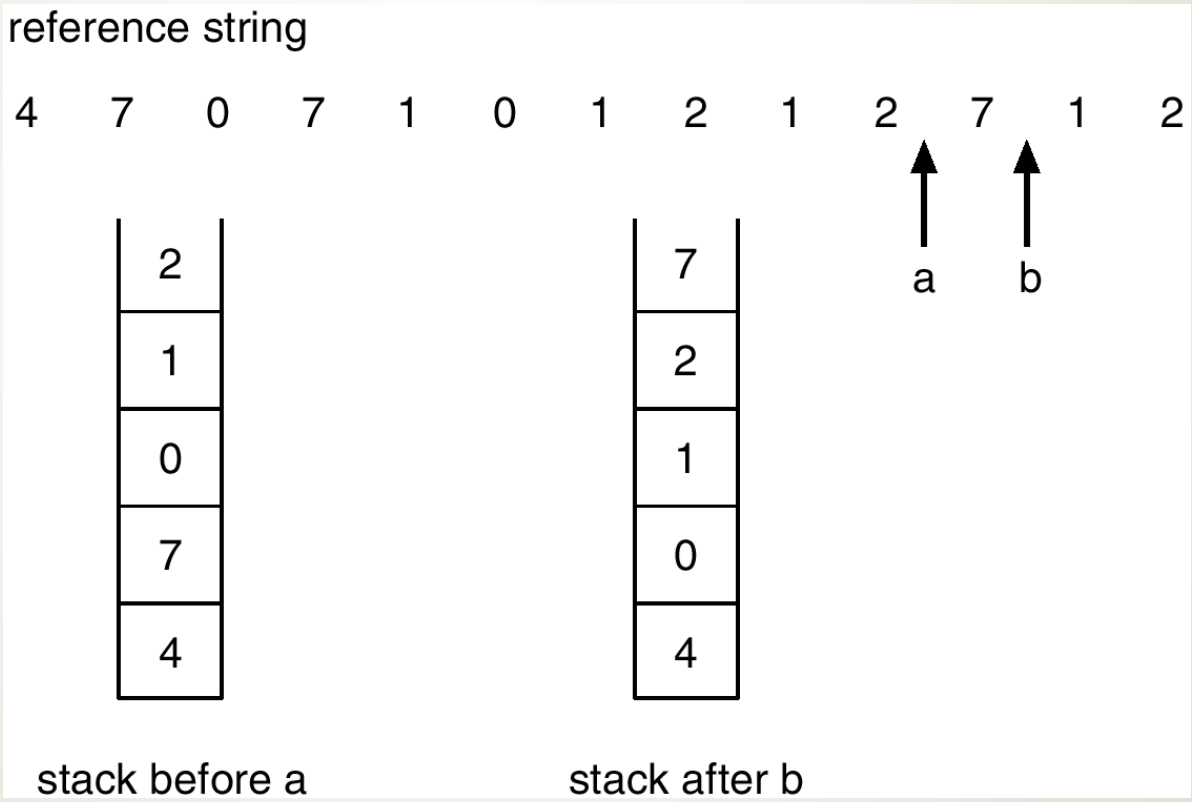
LRU Page Replacement



LRU Algorithm (Cont.)

- Stack implementation – keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - No search for replacement

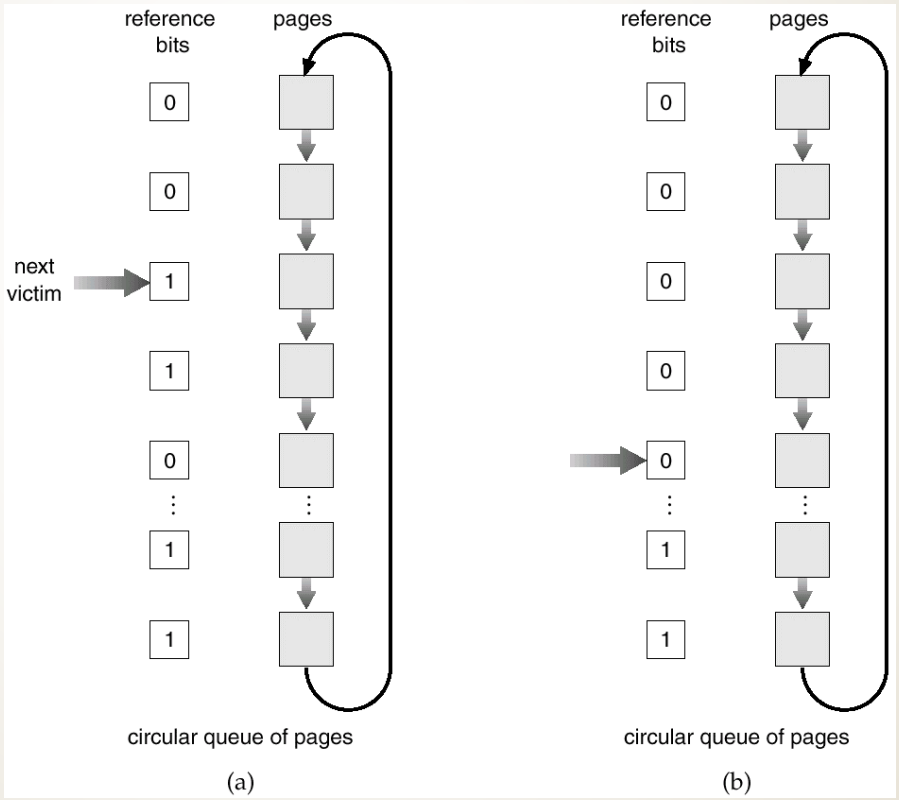
Use Of A Stack to Record The Most Recent Page References



LRU Approximation Algorithms

- Reference bit
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1.
  - Replace the one which is 0 (if one exists). We do not know the order, however.
- Second chance
  - Need reference bit.
  - Clock replacement.
  - If page to be replaced (in clock order) has reference bit = 1. then:
    - set reference bit 0.
    - leave page in memory.
    - replace next page (in clock order), subject to same rules.

Second-Chance (clock) Page-Replacement Algorithm



Counting Algorithms

- Keep a counter of the number of references that have been made to each page.
- LFU Algorithm: replaces page with smallest count.
- MFU Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Allocation of Frames

- Each process needs minimum number of pages.
- Example: IBM 370 – 6 pages to handle SS MOVE instruction:
  - instruction is 6 bytes, might span 2 pages.
  - 2 pages to handle from.
  - 2 pages to handle to.
- Two major allocation schemes.
  - fixed allocation
  - priority allocation

Fixed Allocation

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

- s_i = size of process p_i
- $S = \sum s_i$
- m = total number of frames
- a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$m = 64$

$s_i = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 64 \approx 5$

$a_2 = \frac{127}{137} \times 64 \approx 59$

Priority Allocation

- Use a proportional allocation scheme using priorities rather than size.
- If process Pi generates a page fault,
  - select for replacement one of its frames.
  - select for replacement a frame from a process with lower priority number.

Global vs. Local Allocation

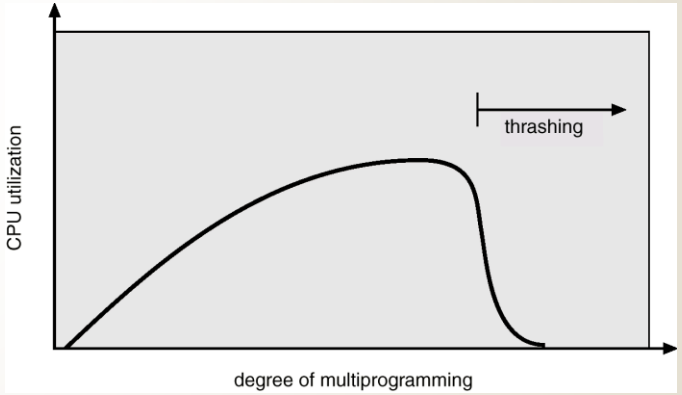
- Global replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another.
- Local replacement – each process selects from only its own set of allocated frames.

Thrashing

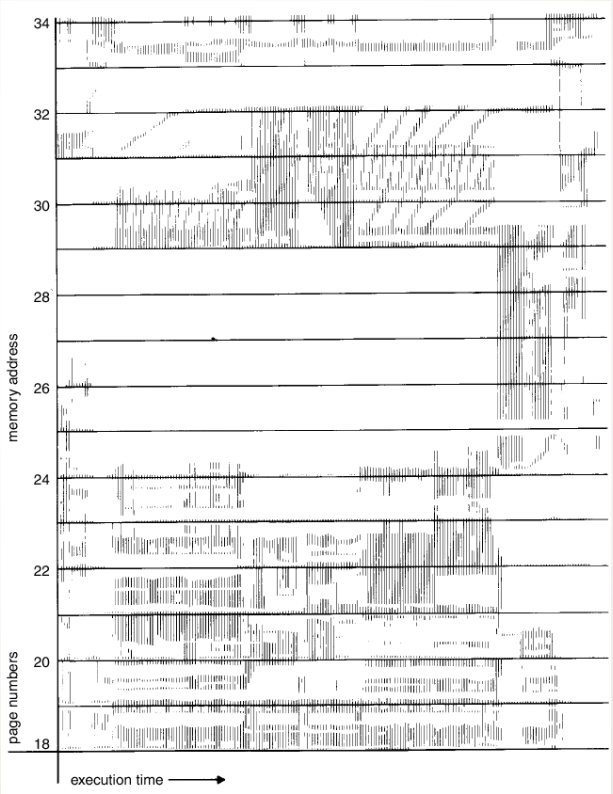
- If a process does not have "enough" pages, the page-fault rate is very high. This leads to:
  - low CPU utilization.
  - operating system thinks that it needs to increase the degree of multiprogramming.
  - another process added to the system.
- Thrashing ≡ a process is busy swapping pages in and out.

Thrashing

- Why does paging work?
  Locality model
  - Process migrates from one locality to another.
  - Localities may overlap.
- Why does thrashing occur?
  Σ size of locality > total memory size



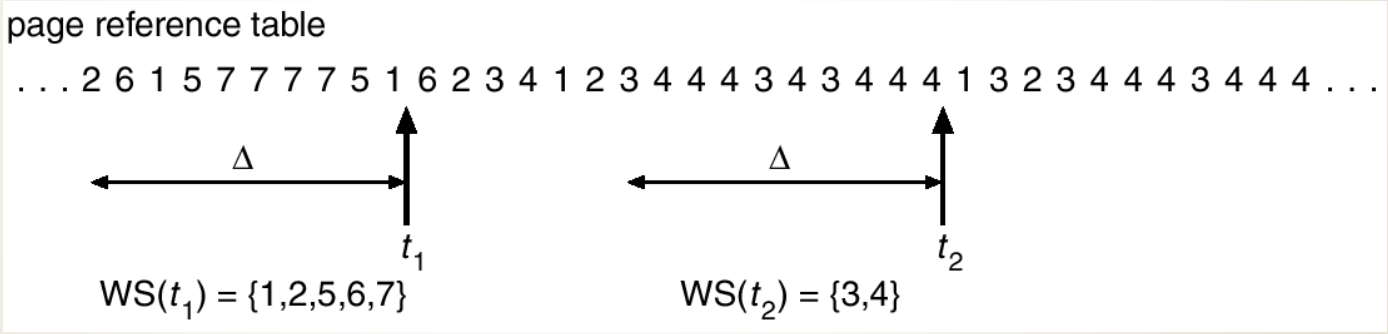
Locality In A Memory-Reference Pattern



Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instruction
- WSSi (working set of Process Pi) =
  total number of pages referenced in the most recent Δ (varies in time)
  - if Δ too small will not encompass entire locality.
  - if Δ too large will encompass several localities.
  - if $\Delta = \infty \Rightarrow$ will encompass entire program.
- $D = \Sigma$ WSSi $\equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy if D > m, then suspend one of the processes.

Working-set model

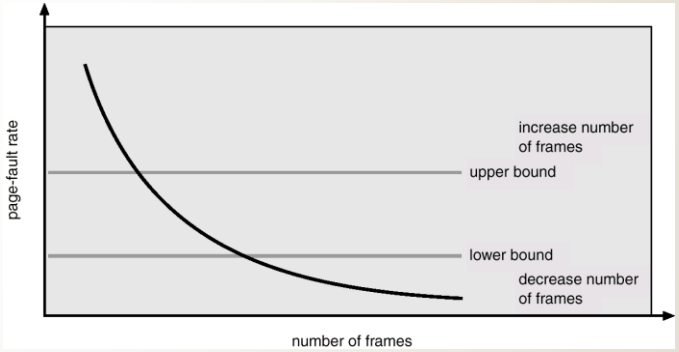


Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example: Δ = 10,000
  - Timer interrupts after every 5000 time units.
  - Keep in memory 2 bits for each page.
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0.
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set.
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units.

Page-Fault Frequency Scheme

- Establish “acceptable” page-fault rate.
  - If actual rate too low, process loses frame.
  - If actual rate too high, process gains frame.



Other Considerations

- Prepaging
- Page size selection
  - fragmentation
  - table size
  - I/O overhead
  - locality

Other Considerations (Cont.)

- TLB Reach - The amount of memory accessible from the TLB.
- TLB Reach = (TLB Size) X (Page Size)
- Ideally, the working set of each process is stored in the TLB. Otherwise there is a high degree of page faults.

Increasing the Size of the TLB

- Increase the Page Size. This may lead to an increase in fragmentation as not all applications require a large page size.
- Provide Multiple Page Sizes. This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation.

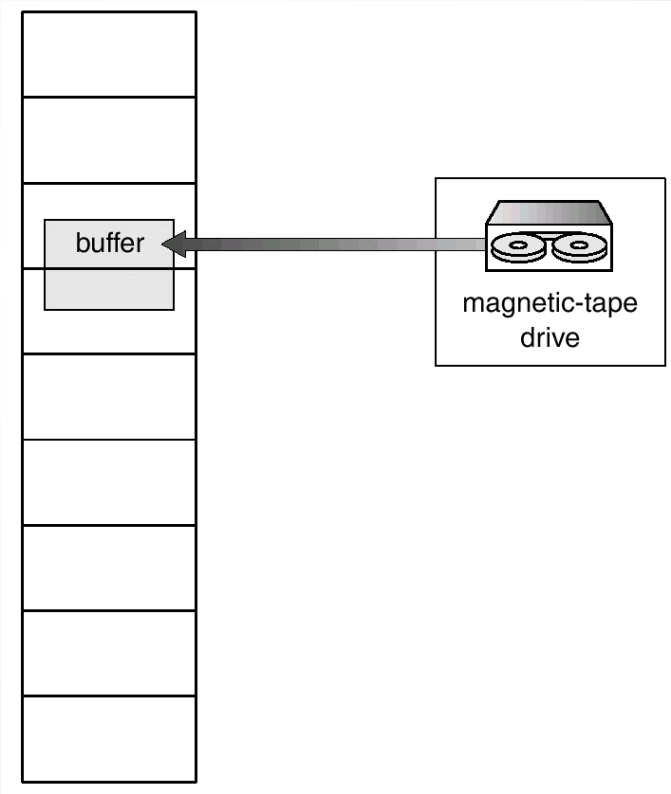
Other Considerations (Cont.)

- Program structure
  - int A[][] = new int[1024][1024];
  - Each row is stored in one page
  - Program 1
    ```
    for (j = 0; j < A.length; j++)
        for (i = 0; i < A.length; i++)
            A[i,j] = 0;
    ```
    1024 x 1024 page faults
  - Program 2
    ```
    for (i = 0; i < A.length; i++)
        for (j = 0; j < A.length; j++)
            A[i,j] = 0;
    ```
  - 1024 page faults

Other Considerations (Cont.)

- I/O Interlock – Pages must sometimes be locked into memory.
- Consider I/O. Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm.

Reason Why Frames Used For I/O Must Be In Memory



Operating System Examples

- Windows NT
- Solaris 2

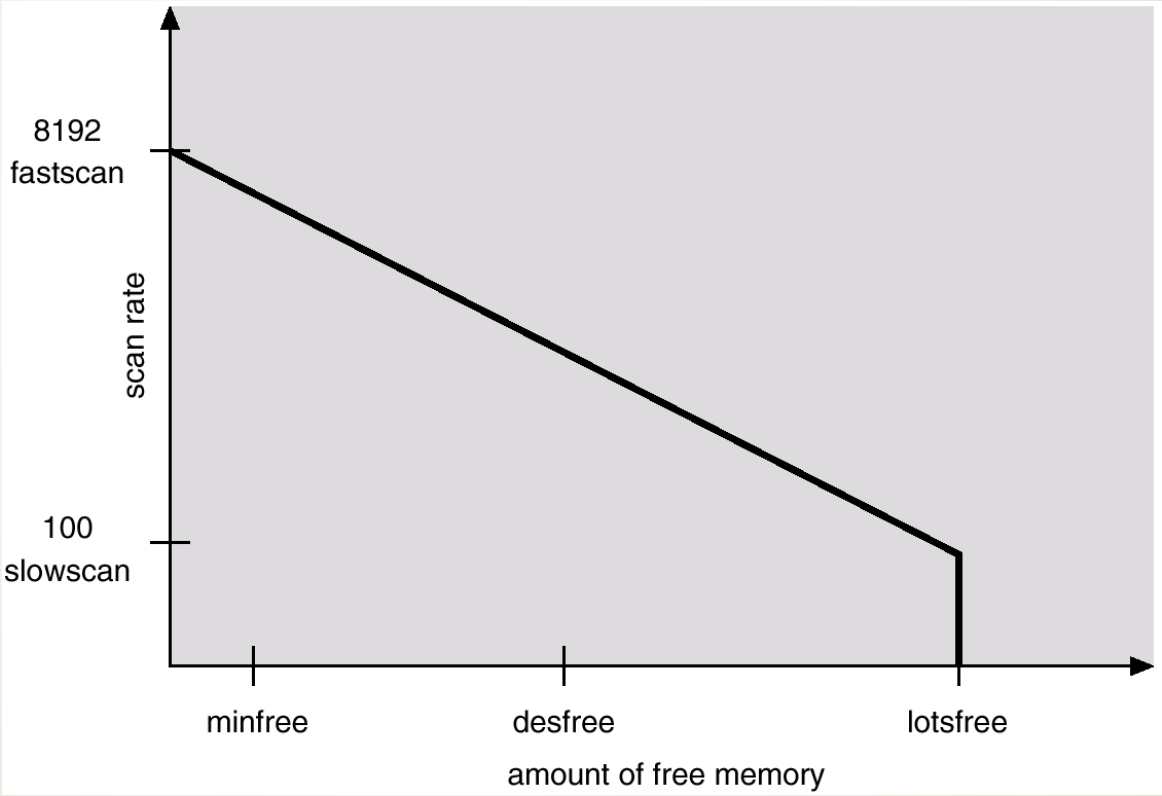
Windows NT

- Uses demand paging with clustering. Clustering brings in pages surrounding the faulting page.
- Processes are assigned working set minimum and working set maximum.
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory.
- A process may be assigned as many pages up to its working set maximum.
- When the amount of free memory in the system falls below a threshold, automatic working set trimming is performed to restore the amount of free memory.
- Working set trimming removes pages from processes that have pages in excess of their working set minimum.

Solaris 2

- Maintains a list of free pages to assign faulting processes.
- Lotsfree – threshold parameter to begin paging.
- Paging is peformed by pageout process.
- Pageout scans pages using modified clock algorithm.
- Scanrate is the rate at which pages are scanned. This ranged from slowscan to fastscan.
- Pageout is called more frequently depending upon the amount of free memory available.

Solar Page Scanner



Thank you

The content in this Material are from the Textbooks and Reference books given in the Syllabus