Module 5:
Memory management

Segmentation, Page Replacement algorithms, Thrashing , Working Set

Dr. Rishikeshan C A
Assistant Professor (Sr.)
SCOPE, VIT Chennai

Outline

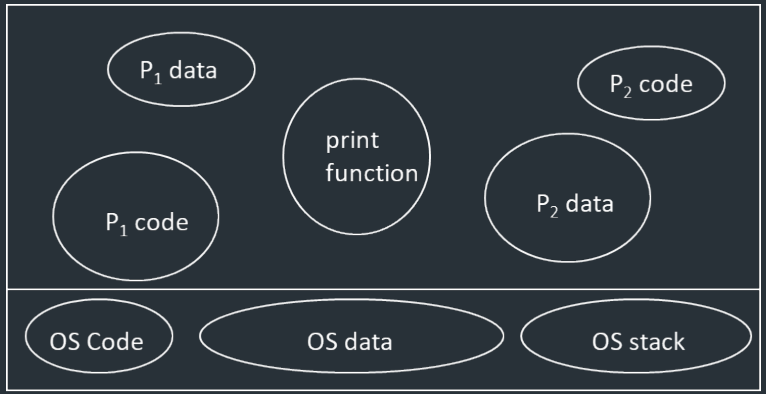
- Segmentation
- Segmentation Hardware
- Segmentation Architecture
- Pros and Cons of Segmentation
- Page-Replacement Algorithms
- Thrashing
- Working Set

Segmentation

- ***Segment***: a region of logically contiguous memory
- ***Segmentation-based transition***: use a table of base-and-bound pairs

- Memory-management scheme that supports user view of memory
- A program is a collection of segments. A segment is a logical unit such as:

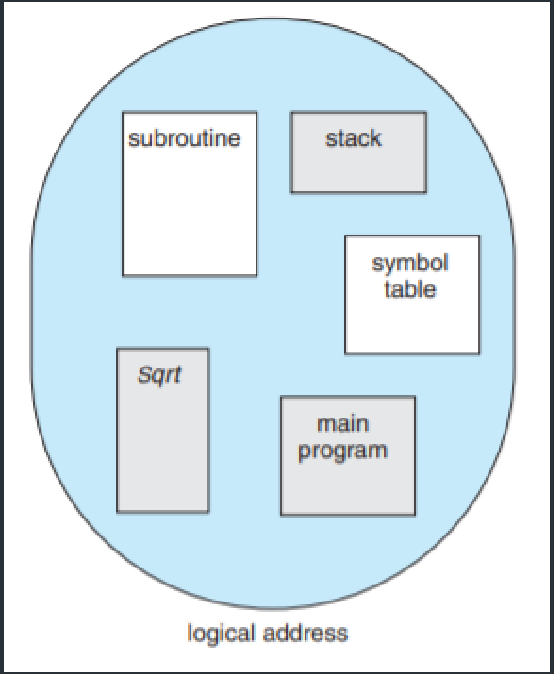
  main program,
  procedure,
  function,
  method,
  object,
  local variables, global variables,
  common block,
  stack,
  symbol table, arrays



logical address space

Segmentation

- Segmentation is a technique for breaking memory up into logical pieces
- Each "piece" is a grouping of related information
  - data segments for each process
  - code segments for each process
  - data segments for the OS
  - etc.
- Like paging, use virtual addresses and use disk to make memory look bigger than it really is
- Segmentation can be implemented with or without paging



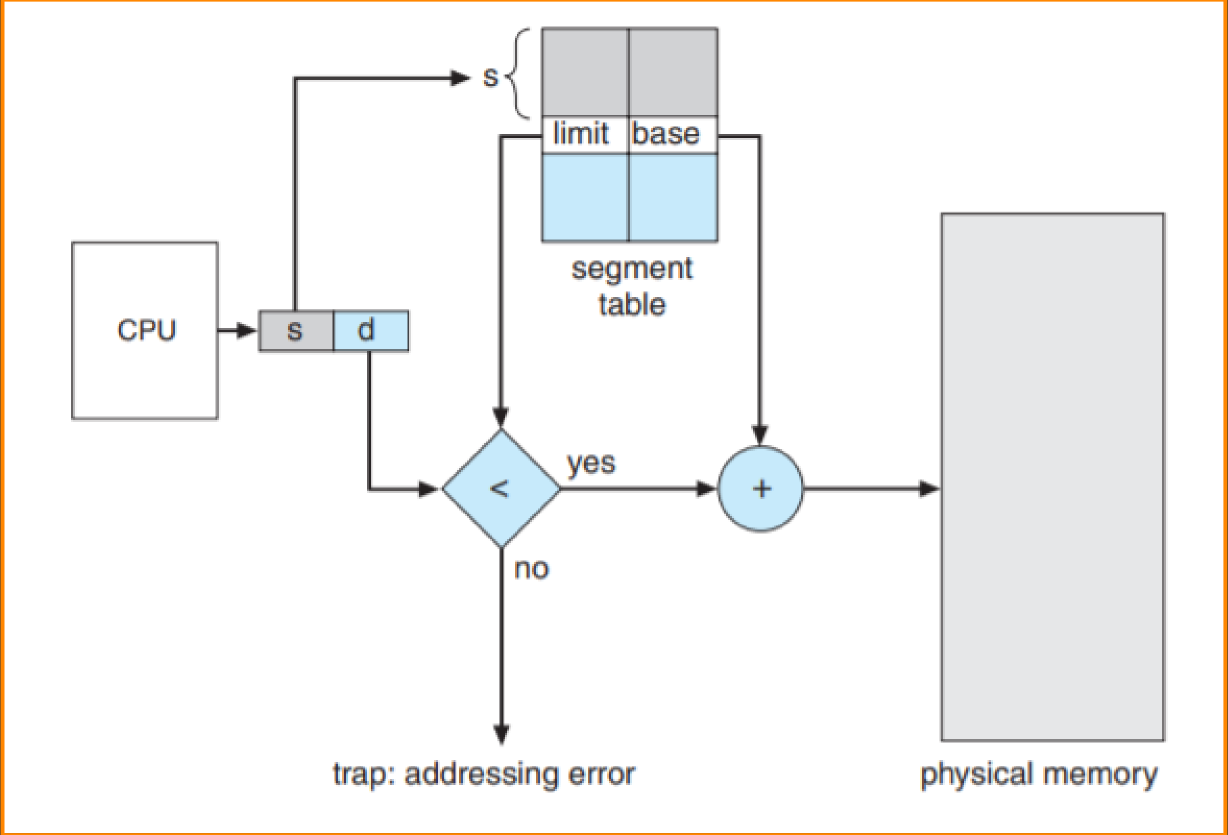
User's View of a Program

Source: Operating System Concepts, 9th Edition, ***Abraham Silberschatz****, Peter B.* ***Galvin****, Greg Gagne*

Addressing Segments

- Let’s first assume no paging in the system
- User generates logical addresses
- These addresses consist of a segment number and an offset into the segment
- Use segment number to index into a table
- Table contains the physical address of the start of the segment
  - often called the base address
- Add the offset to the base and generate the physical address
  - before doing this, check the offset against a limit
  - the limit is the size of the segment

Addressing Segments

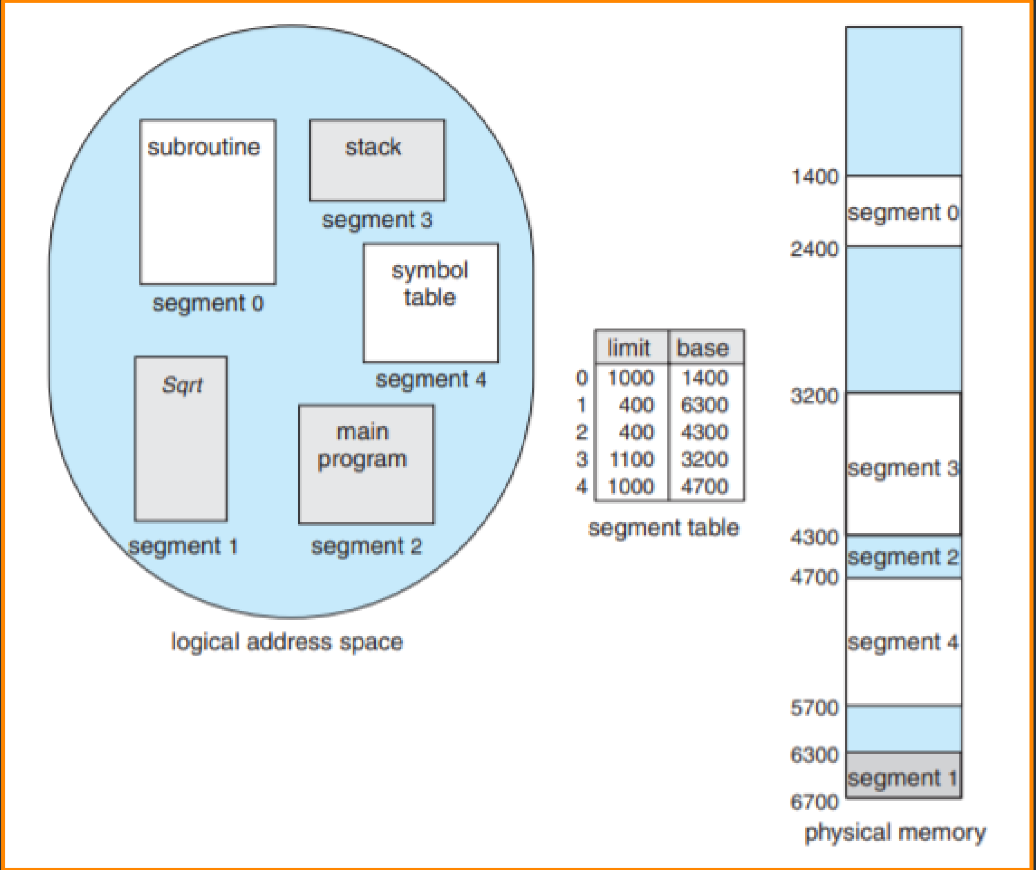


Source: *Operating System Concepts, 9th Edition,* ***Abraham Silberschatz****, Peter B.* ***Galvin****, Greg Gagne*

Segmentation Hardware

- Sounds very similar to paging
- Big difference – segments can be variable in size
- As with paging, to be effective hardware must be used to translate logical address
- Most systems provide segment registers
- If a reference isn't found in one of the segment registers
  - trap to operating system
  - OS does lookup in segment table and loads new segment descriptor into the register
  - return control to the user and resume
- Again, similar to paging

Example of Segmentation



Source: *Operating System Concepts, 9th Edition,* ***Abraham Silberschatz****, Peter B.* ***Galvin****, Greg Gagne*

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;

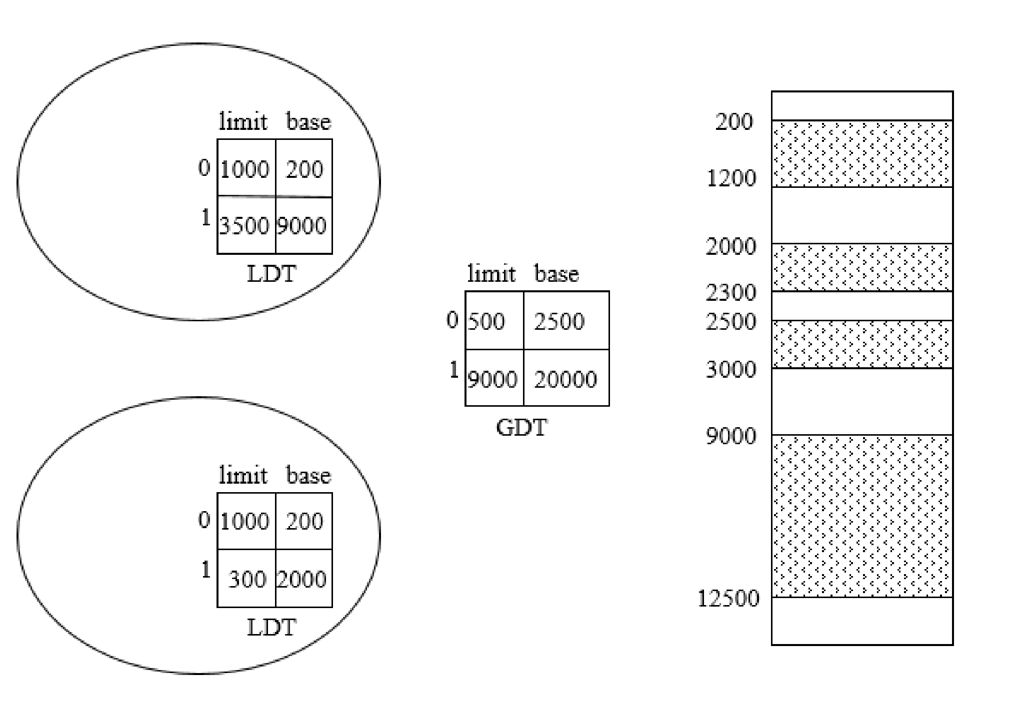
  segment number ***s*** is legal if ***s*** < **STLR**

Segmentation Architecture (Cont.)

- 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

Protection and Sharing

- Like page tables, each process usually gets its own segment table
- Unlike page tables, there usually exists a global segment table for everyone
  - this, however, is usually used by OS
- Access rights for segment are usually included in table entry
- Multiple processes can share a segment



Pros and Cons of Segmentation

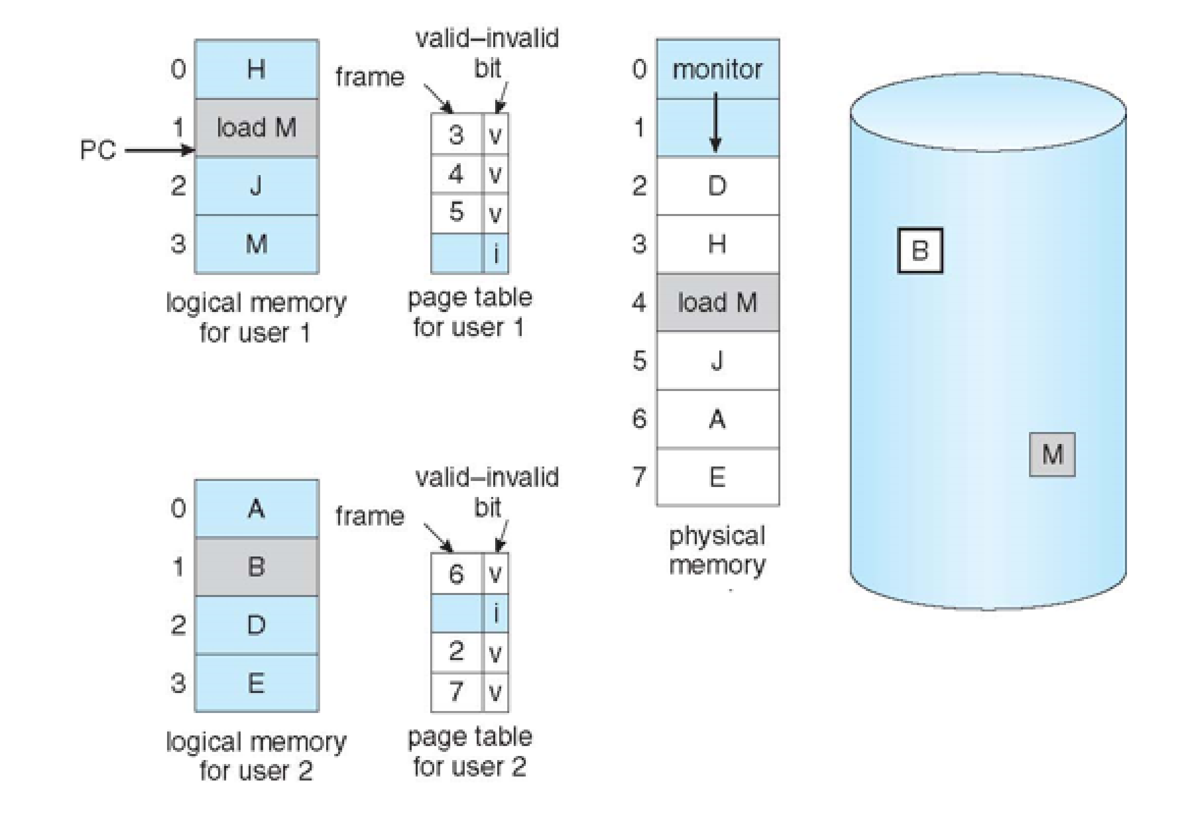
- Easier to grow and shrink individual segments
- Finer control of segment accesses
  - e.g., read-only for shared code segment
- More efficient use of physical space
- Multiple processes can share the same code segment
- Memory allocation is still complex
  - Requires contiguous allocation
- Entire segment is either in memory or on disk
- Variable sized segments leads to external fragmentation in memory
- Must find a space big enough to place segment into
- May need to swap out some segments to bring a new segment in

Segmentation with Paging

- Most architectures support segmentation and paging
- Basic idea,
  - segments exist in virtual address space
  - base address in segment descriptor table is a virtual address
  - use paging mechanism to translate this virtual address into a physical address
- Now an entire segment does not have to be in memory at one time
  - only the part of the segment that we need will be in memory

Page Replacement Algorithms

Need For Page Replacement



*Source: Operating System Concepts, 9th Edition, **Abraham Silberschatz**, Peter B. **Galvin**, Greg Gagne*

Basic Page Replacement

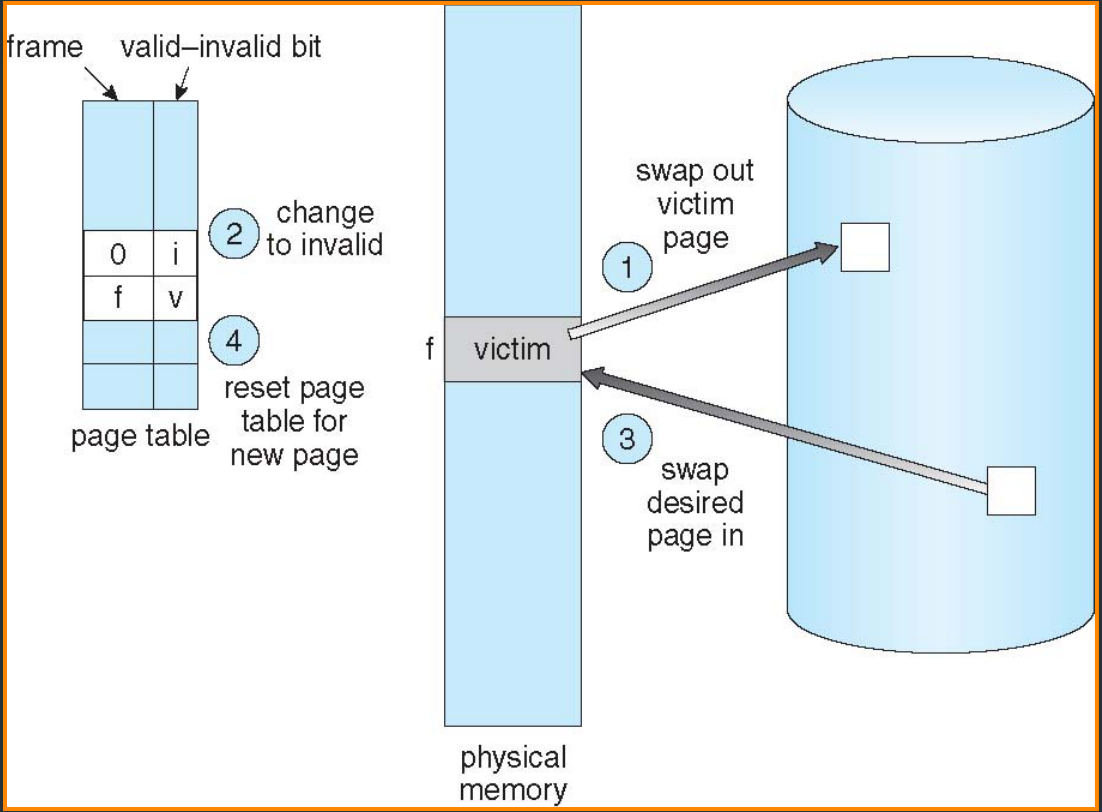
1. Find the location of the desired page on disk

2. 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**
     - Write victim frame to disk if dirty
3. Bring the desired page into the (newly) free frame; update the page and frame tables

4. Continue the process by restarting the instruction that caused the trap

Note: potentially 2 page transfers for page fault – increasing EAT

Page Replacement mechanism



*Source: Operating System Concepts, 9th Edition, **Abraham Silberschatz**, Peter B. **Galvin**, Greg Gagne*

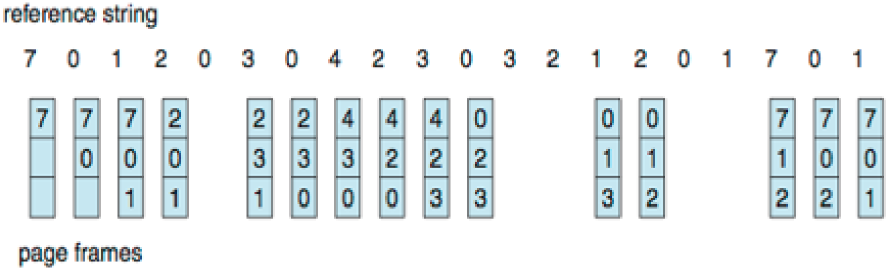
Page Replacement Algorithms

- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available
- In all our examples, the **reference string** of referenced page numbers is

7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

First-In-First-Out (FIFO) Algorithm

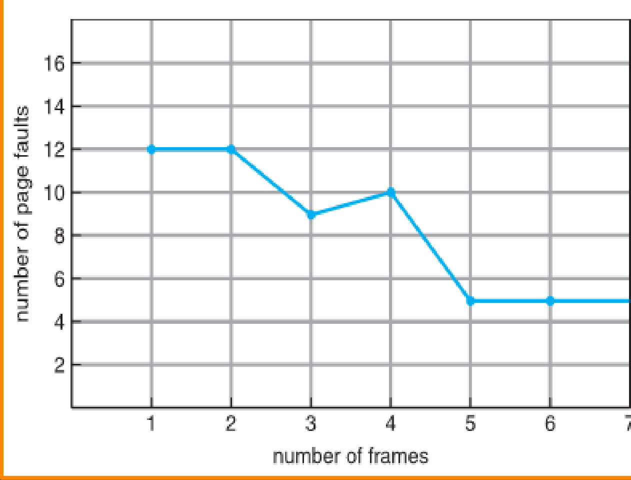
- Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time per process)



15 page faults

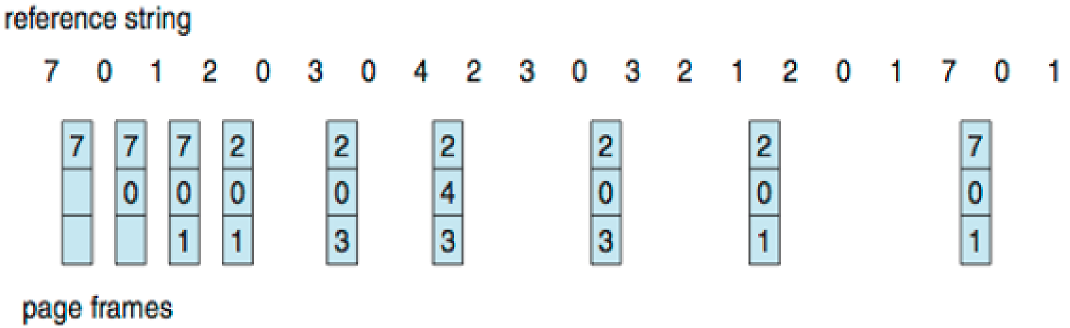
- Can vary by reference string: consider 1,2,3,4,1,2,5,1,2,3,4,5
  - Adding more frames can cause more page faults!
    - **Belady's Anomaly**
- How to track ages of pages?
  - Just use a FIFO queue

Illustrating Belady's Anamoly



Optimal Algorithm

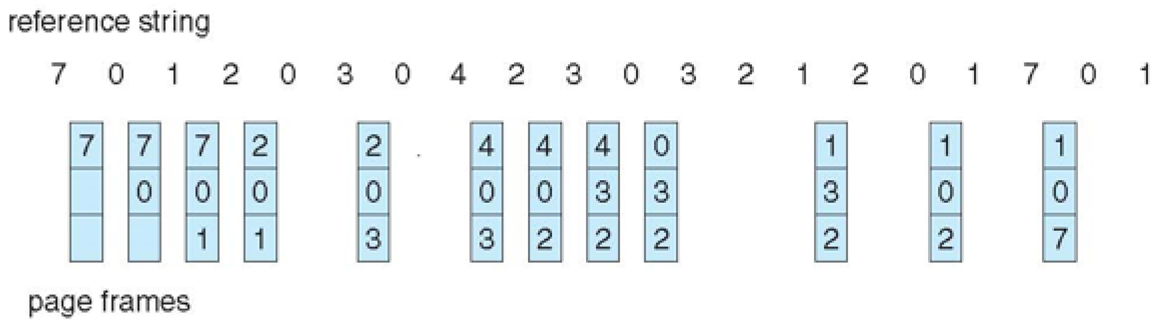
- Replace page that will not be used for longest period of time
  - 9 is optimal for the example
- How do you know this?
  - Can't read the future
- Used for measuring how well your algorithm performs



9 page faults

Least Recently Used (LRU) Algorithm

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page



- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- But how to implement?

Source: *Operating System Concepts, 9th Edition,* ***Abraham Silberschatz****, Peter B.* ***Galvin****, Greg Gagne*

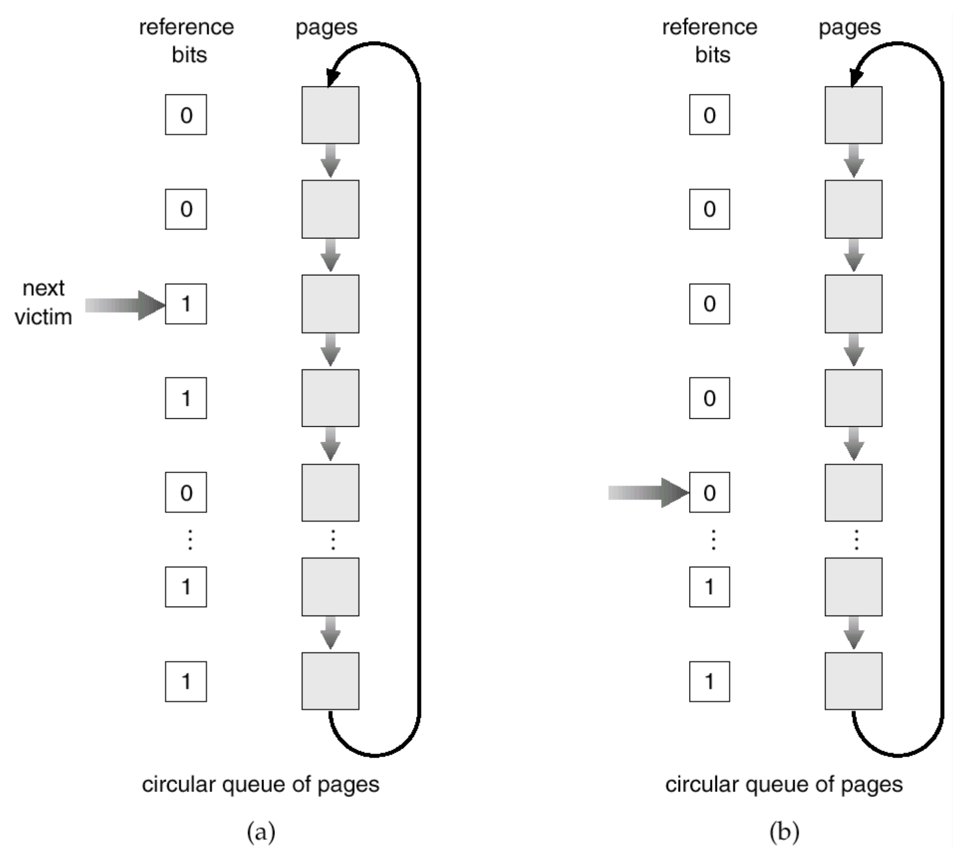
LRU Algorithm (Cont.)

- 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 find smallest value
    - Search through table needed
- 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
  - But each update more expensive
  - No search for replacement
- LRU and OPT are cases of **stack algorithms** that don’t have Belady’s Anomaly

Clock page replacement algorithm (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



Source: *Operating System Concepts, 9th Edition, **Abraham Silberschatz**, Peter B. **Galvin**, Greg Gagne*

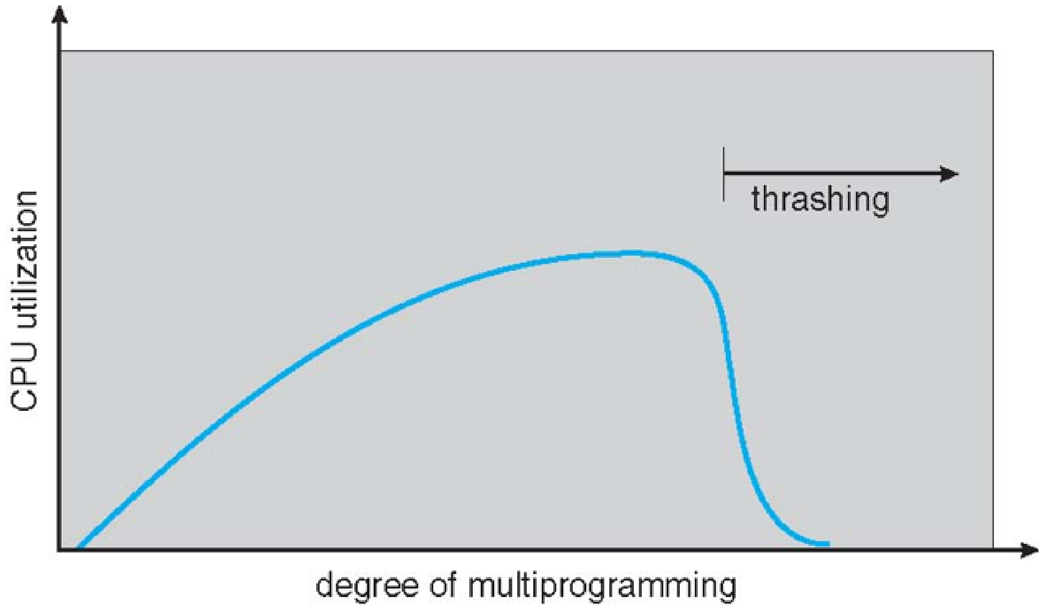
Counting Algorithms

- Keep a counter of the number of references that have been made to each page
  - Not common
- **Lease Frequently Used (LFU) Algorithm**: replaces page with smallest count
- **Most Frequently Used (MFU) Algorithm**: based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Thrashing

- If a process does not have "enough" pages, the page-fault rate is very high
  - Page fault to get page
  - Replace existing frame
  - But quickly need replaced frame back
  - This leads to:
    - Low CPU utilization
    - Operating system thinking that it needs to increase the degree of multiprogramming
    - Another process added to the system
- **Thrashing** $\equiv$ a process is busy swapping pages in and out

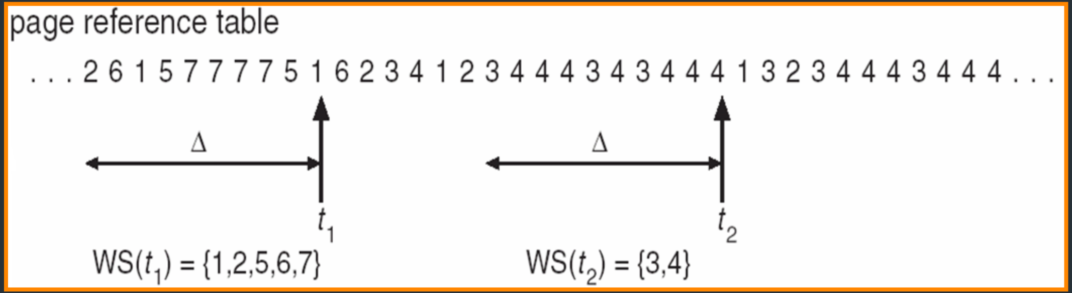
Thrashing



Source: Operating System Concepts, 9th Edition, ***Abraham Silberschatz****, Peter B.* ***Galvin****, Greg Gagne*

Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instructions
- WSS_i (working set of Process P_i) = 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
- $D = \Sigma\, WSS_i \equiv$ total demand frames
  - Approximation of locality
- if $D > m \Rightarrow$ Thrashing
- Policy if D > m, then suspend or swap out one of the processes

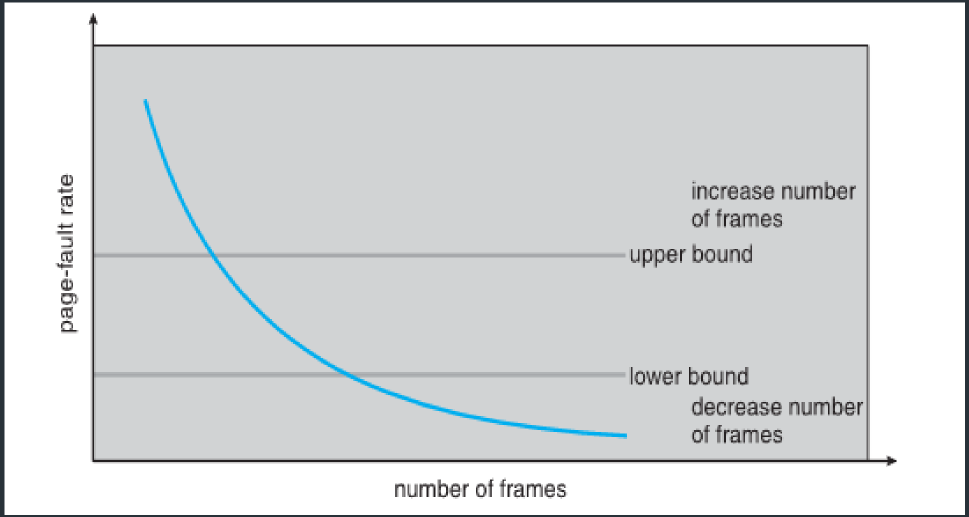


Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example: $\Delta = 10{,}000$
  - 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

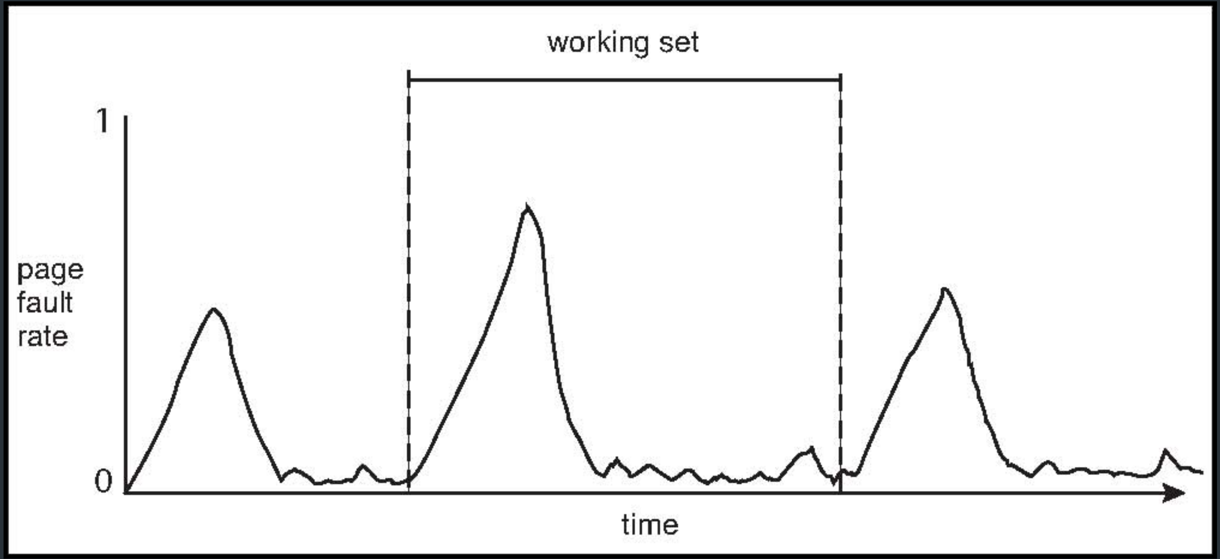
- More direct approach than WSS
- Establish “acceptable” **page-fault frequency** (**PFF**) rate and use local replacement policy
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



*Source: Operating System Concepts, 9th Edition, **Abraham Silberschatz**, Peter B. **Galvin**, Greg Gagne*

Working Sets and Page Fault Rates

- Direct relationship between working set of a process and its page fault rate
- Working set changes over time
- Peaks and valleys over time



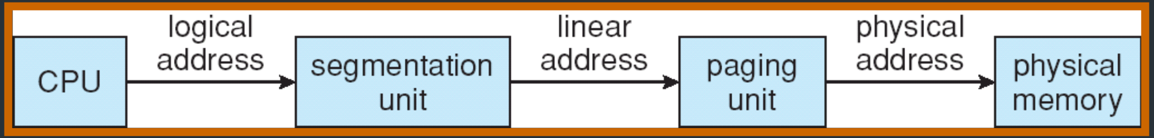
*Source: Operating System Concepts, 9th Edition, **Abraham Silberschatz**, Peter B. **Galvin**, Greg Gagne*

Implementation examples of Paging & Segmentation in Intel Pentium

Example: The Intel Pentium

- Supports both segmentation and segmentation with paging
- CPU generates logical address
    - Given to segmentation unit
        - Which produces linear addresses
    - Linear address given to paging unit
        - Which generates physical address in main memory
        - Paging units form equivalent of MMU

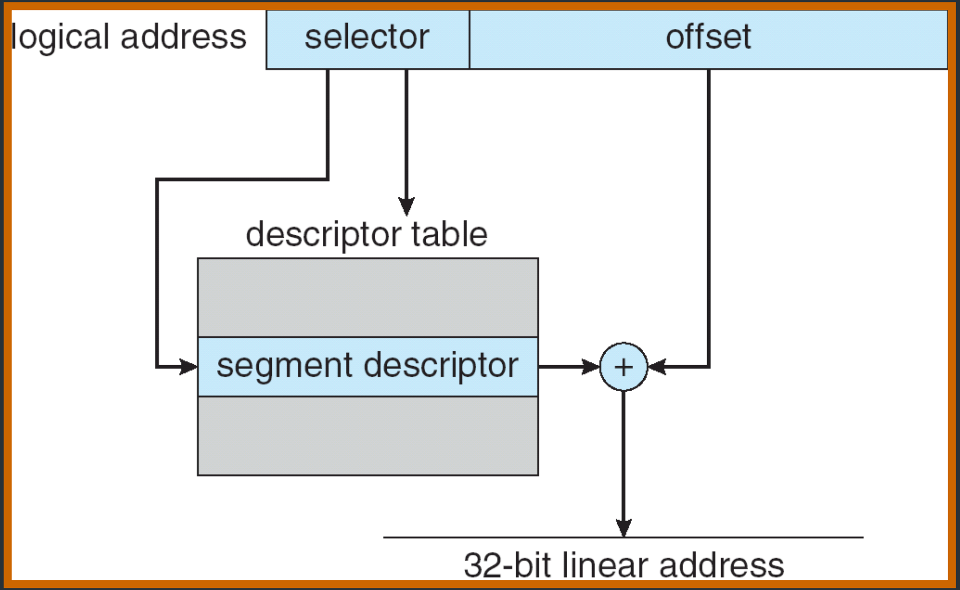
Logical to Physical Address Translation in Pentium



page number		page offset
p_1	p_2	d
10	10	12

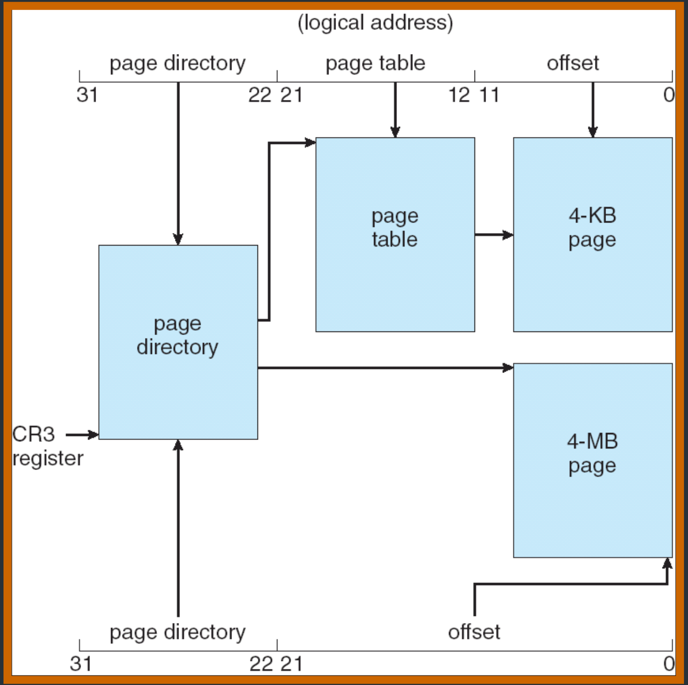
*Source: Operating System Concepts, 9th Edition, **Abraham Silberschatz**, Peter B. **Galvin**, Greg Gagne*

Intel Pentium Segmentation



Source: *Operating System Concepts, 9th Edition,* ***Abraham Silberschatz****, Peter B.* ***Galvin****, Greg Gagne*

Pentium Paging Architecture



Source: *Operating System Concepts, 9th Edition,* ***Abraham Silberschatz****, Peter B.* ***Galvin****, Greg Gagne*

Any Questions ?

References

- http://www.digilife.be/quickreferences/qrc/linux%20system%20call%20quick%
- http://www.cs.nthu.edu.tw/~ychung/slides/CSC3150/Abraham-Silberschatz-Operating-System-Concepts---9th2012.12.pdf
- Applied Operating Systems Concepts", is based on the text "Operating System Concepts", 5/e (1998) by Abraham Silberschatz and Peter Baer Galvin.
- Silberschatz, Gagne, Galvin: Operating System Concepts, 9th Edition
- http://docs.cs.up.ac.za/programming/asm/derick_tut/syscalls.html

Thank You!