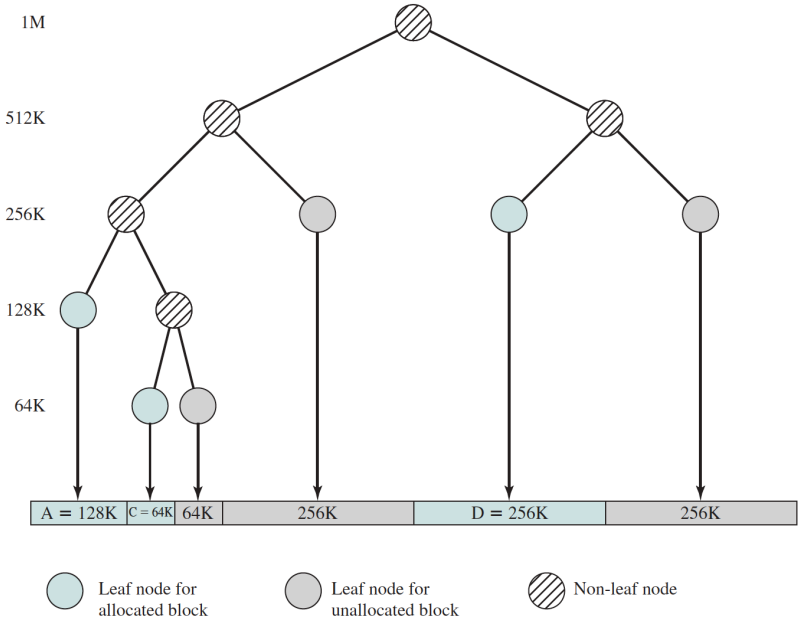
Chapter 7 (7.1 – 7.4)

- **Frame** – fixed-length block of main memory
- **Page** – fixed-length block of data in secondary memory that can be copied into a frame
- **Segment** – variable-length block of data in secondary memory that can be copied into main memory, or split into pages then copied
- **Memory management** satisfies these requirements:
  - Relocation
    - Process may be swapped out and into a different memory location
    - Translation of memory references in program code
  - Protection
    - Process should not access memory in another process
  - Sharing
    - Allow processes to access same portion of memory
    - Multiple processes in one program need to access the same code
  - Logical organization
    - Modularized programs; different degrees of protection for each module
  - Physical organization
    - Overlaying allows modules to be assigned the same region of memory

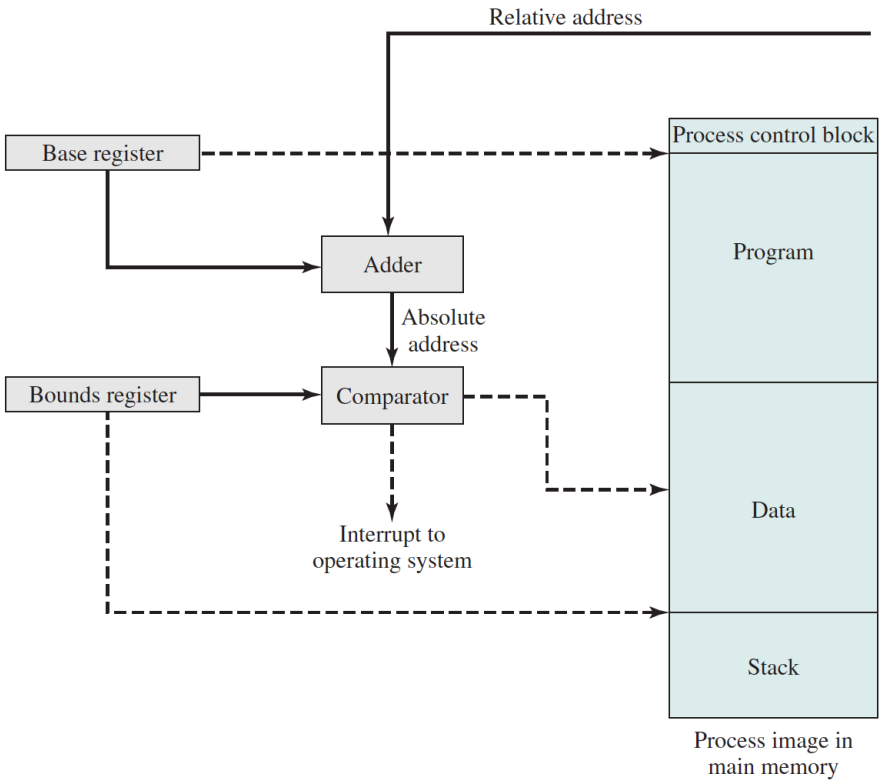
Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.

- **Fixed partitioning**
  - Placement in fixed-size partitions: pick any available
  - Placement in variable-size partitions: pick smallest-fitting partition
    - One process queue per partition vs. single queue for all partitions
  - **Internal fragmentation** – wasted memory within partitions
- **Dynamic partitioning**
  - Placement:
    - Best-fit – closest (and greater than) required size
    - First-fit – first available block from beginning
    - Next-fit – first available block from last placement location
  - **External fragmentation** – wasted memory between partitions

- Buddy system:
  - Split memory in half until B/2 ≤ required size ≤ B, where B = block size
  - When a block (size B)'s buddy is freed, coalesce into a block of size 2B

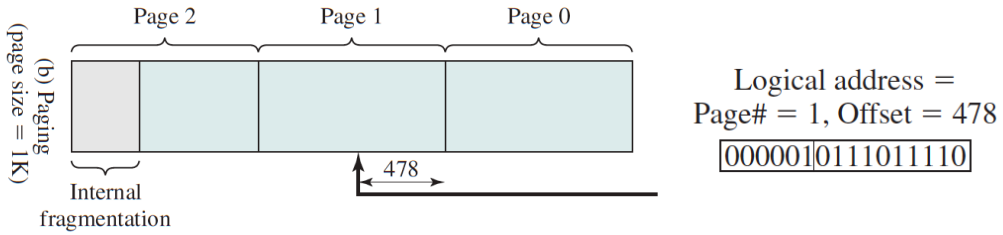


- Relocation in partition:
  - Logical address = location relative to the beginning of program
  - Relative address is added to base register to produce absolute address, and compared with bounds register before accessing data

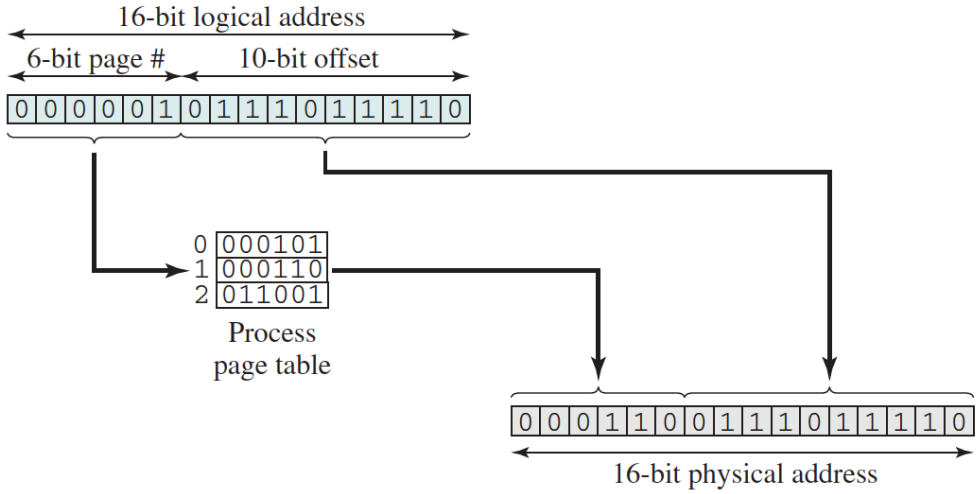


Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.

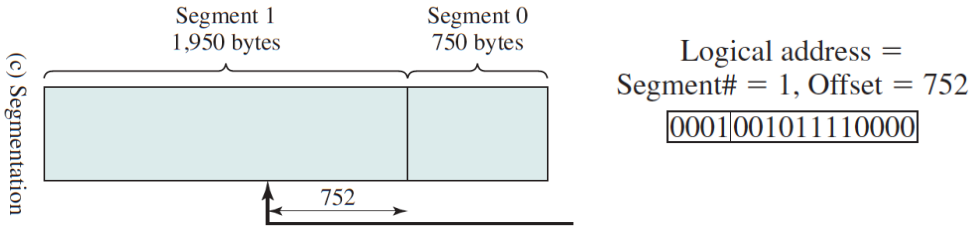
- **Paging**
  - Page table[page index] = frame index
  - Address is split into *n* (page number) + *m* (offset) bits



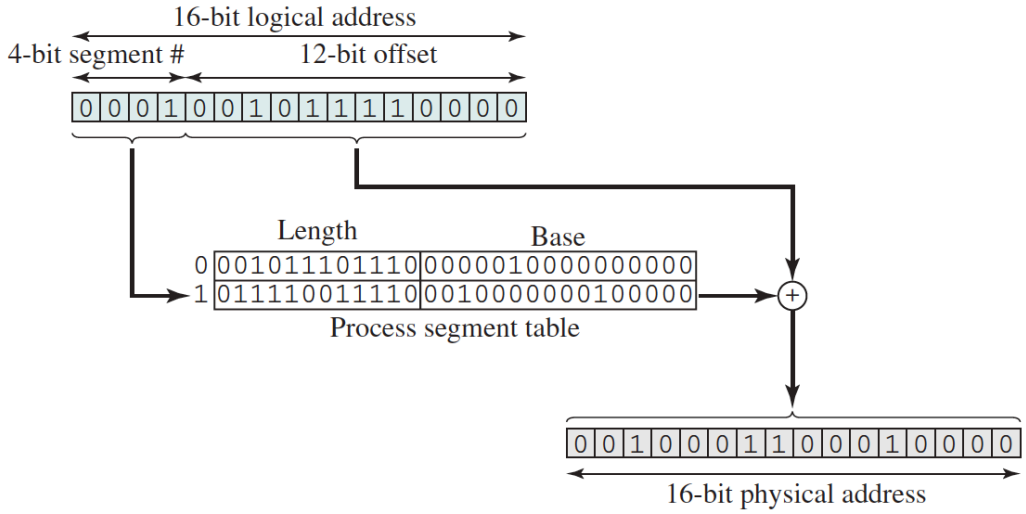
  - Address translation:



- **Segmentation**
  - Segment table[segment index] = length, base



  - Address translation:



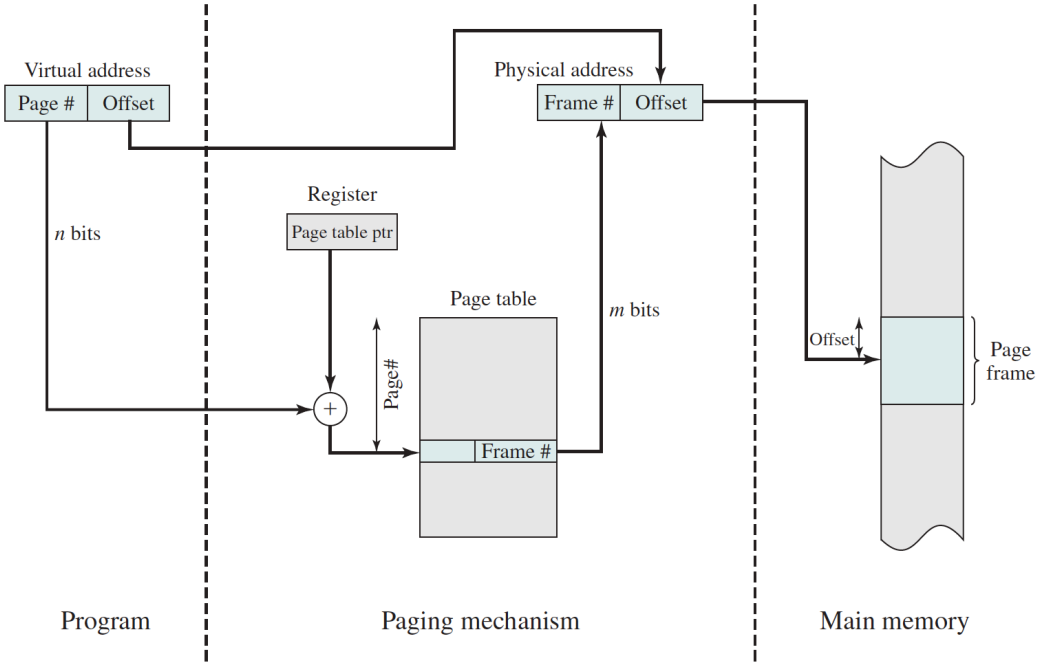
Chapter 8 (8.1 – 8.4)

- **Virtual memory** – storage allocation scheme in which secondary memory can be addressed as though it were part of main memory
  - Key benefits:
    - More processes can be maintained in main memory
    - A process may be larger than main memory
  - Resident set – portion of a process actually in main memory
  - When a mem reference outside of resident set is encountered (page fault), interrupt & block the process, issue I/O to retrieve the needed program
    - **Lazy loading** – defer loading until needed – keeps CPU busy, less wasted time
  - Virtual address – location in virtual memory that can be accessed as though it's in main memory
  - Real address – actual location in main memory

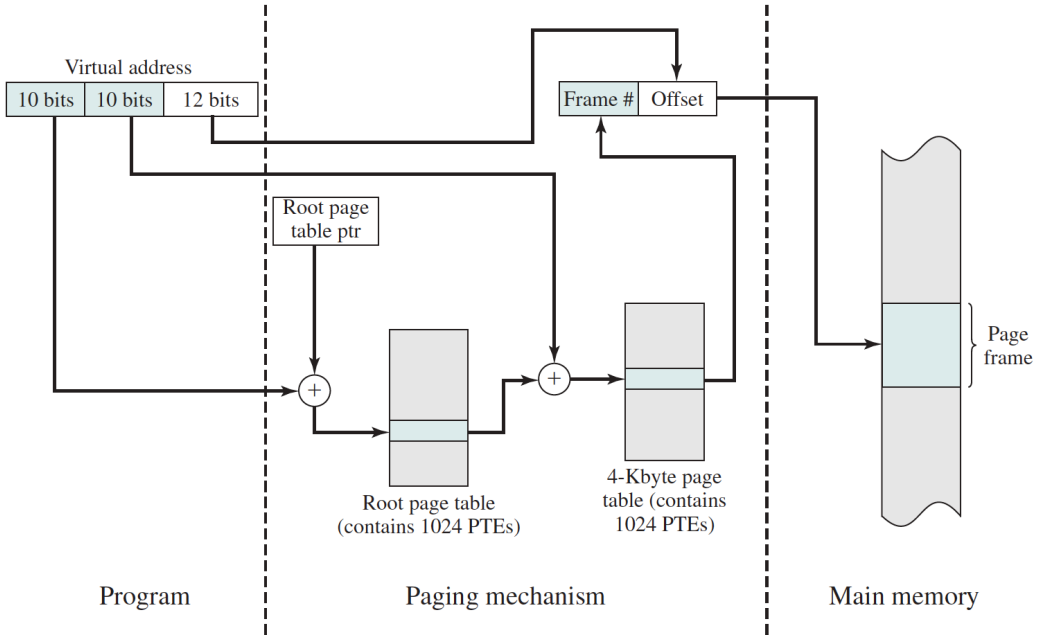
Virtual Memory Paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual Memory Segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

  - **Thrashing** – swap out something that is needed again immediately; wastes too much time on swapping
    - Solve with principle of locality

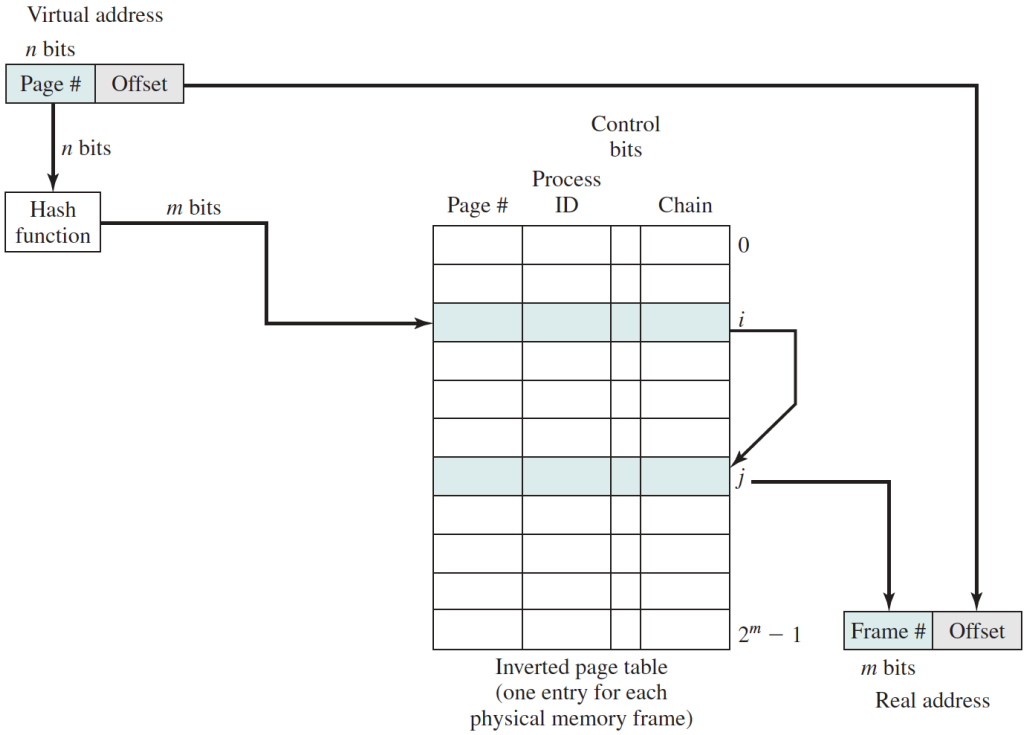
- Hardware support for virtual memory:
- **Page tables**
  - Entry contains present (P) & modified (M) control bits, in addition to frame #
  - Page tables are in virtual memory as well
  - Large page tables referenced by page directories
    - Page directories are in main memory
    - A "root table" for the pages that the actual page table occupies
- Single-level page table:



- Two-level page table:

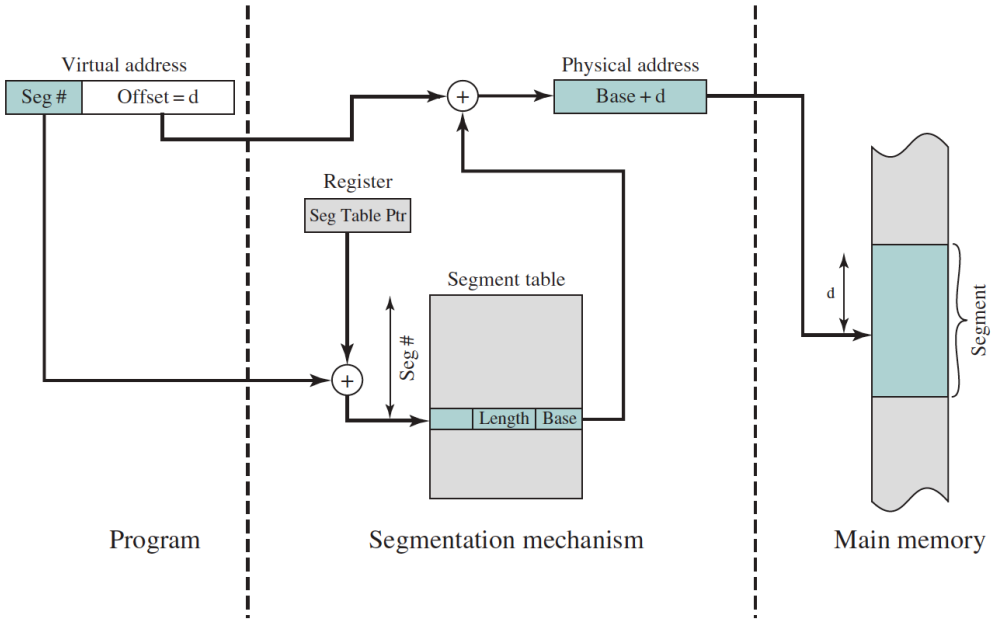


- Inverted page table – one entry per real memory frame
  - Page table doesn't have to be paged since # of pages = # of frames
  - Page # is hashed into table index; overflows are chained

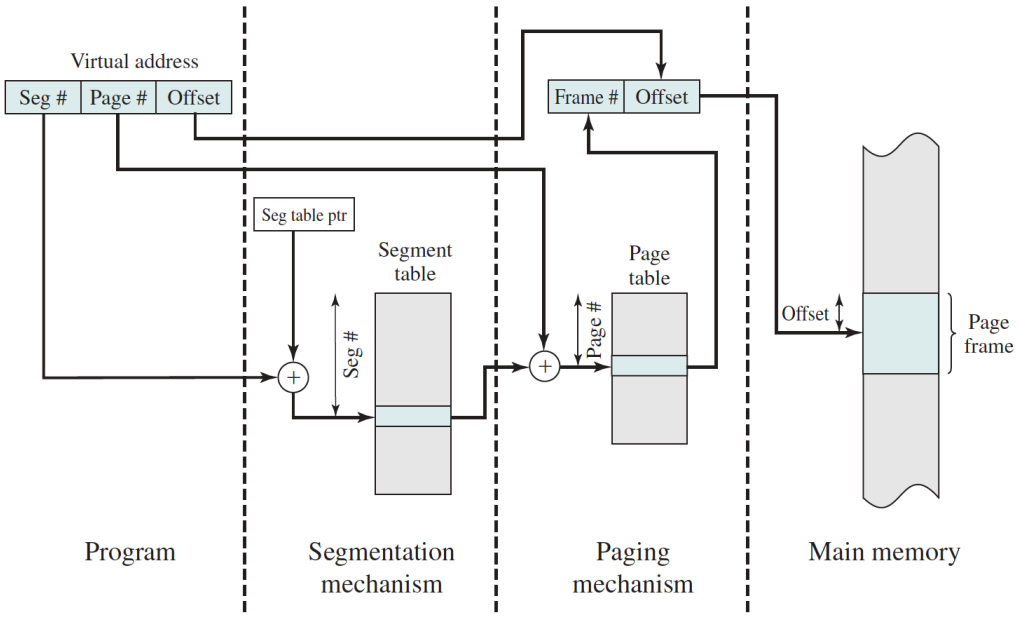


- Translation look-aside buffer – cache for page table entries
  - Check TLB →
    - If TLB hit → get frame # from TLB page entry
    - If TLB miss → lookup in page table
      - If present bit set → get frame # from page entry
      - If present bit not set → page fault
- Cache is also consulted when retrieving a block from memory
- Page size:
  - Large = more internal fragmentation; weaker principle of locality → more page faults
  - Small = larger page tables → more pages in virtual memory → more page faults
- Pages in memory:
  - More pages in memory → less page faults

- **Segmentation:**
  - Advantages:
    - Simplifies growing data structures
    - Programs can be altered & recompiled independently
    - Lends itself to sharing
    - Lends itself to protection



- **Combined paging & segmentation:**
  - Each segment is paged
  - Segment offset = page # & offset



- Memory management software:
- OS virtual memory policies:
  - **Fetch policy** – when to load pages
    - Demand paging – only load page when reference to it is made
    - Prepaging – load multiple contiguous pages
      - Good for processes that use many contiguous pages
  - Placement policy
    - Important in segmentation system – best-fit, first-fit, next-fit etc.
    - Not as relevant in paging/combined systems
  - **Replacement policy** – which page(s) in main memory to replace
    - Locked frames cannot be replaced (i.e. kernel frames)
    - Optimal (impossible in practice)
    - LRU
    - FIFO
    - Clock – look for page with use bit = 0
      - When searching, every use bit = 1 encountered is set to 0
      - Referencing a page sets use bit to 1
    - Clock policy with use bit & modified bit
      - 1. Search for u = 0 & m = 0
      - 2. Search for u = 1& m = 0; set u to 0 when scanning
      - Repeat 1 and 2 if necessary
    - Page buffering
      - Replaced pages are not removed from memory; instead are added to free (unchanged) & modified lists
      - Lists act as caches, can be added back to resident set if referenced
      - Modified list is written out in clusters
  - **Resident set management**

	Local Replacement	**Global Replacement**
Fixed Allocation	• Number of frames allocated to a process is fixed. • Page to be replaced is chosen from among the frames allocated to that process.	• Not possible.
Variable Allocation	• The number of frames allocated to a process may be changed from time to time to maintain the working set of the process. • Page to be replaced is chosen from among the frames allocated to that process.	• Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.

    - Size:
      - Fixed allocation
      - Variable allocation
        - Processes w/ high page fault rate can be allocated more frames
    - Scope:

        - Local replacement – only choose from resident pages of faulting process
        - Global replacement – choose from any unlocked pages
    - **Working set** – set of pages referenced by a process in the last given amount of time
        - Working set strategy:
            - Monitor the W of each process
            - Periodically remove pages in resident set but not in W (LRU)
            - Process may only execute if its W is in memory
        - Algorithms that follow this strategy:
            - Page fault frequency
                - If time since last page fault < threshold → grow resident set
                - If time since last page fault > threshold → remove all unused pages
            - Variable-interval sampled working set
    - **Cleaning policy** – when to write out modified pages
        - Demand cleaning – written out when it's replaced
        - Precleaning – written out in batches
        - Page buffering – uses modified & unmodified lists of pages
    - **Load control** – how many processes to maintain in memory (aka. multiprogramming level)
        - Too few → processes too often blocked
        - Too many → resident sets too small → thrashing (too much swapping)
    - Process suspension – which process to take out of memory
        - Lowest-priority
        - Faulting process
        - Last activated
        - Smallest resident set
        - Largest process
        - Largest remaining execution window

- **UNIX/Solaris**
    - Page replacement uses a refined, two-handed clock algorithm
- **Linux**
    - 3-level page table structure
    - Page directory (in main memory) → middle directory → page table → frame #
    - Page replacement uses split LRU algorithm (active & inactive lists)