Evaluation of Dynamic Relocation (Review)

- Advantages:
  - OS can easily move a process
  - OS can allow processes to grow
  - Hardware changes are minimal, but fairly fast and efficient
  - ➥Transparency, safety, and efficiency are all satisfied; overhead is small
- Disadvantages:
  - Addresses must be translated
  - Memory allocation is complex (partitions, holes, fragmentation, etc.)
  - If process grows, OS may have to move it
  - Process limited to physical memory size
  - Process needs contiguous memory space
  - Not possible to share code or data between processes

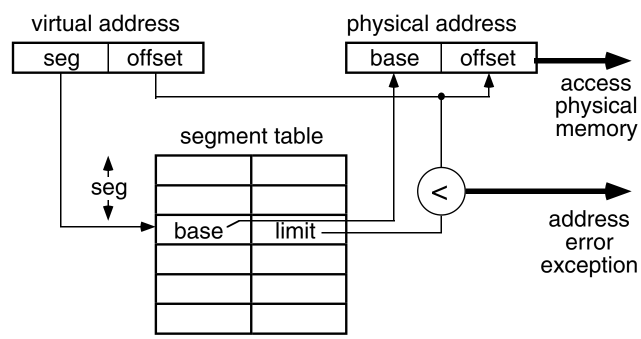
Segmentation

- Basic idea — using the programmer's view of the program, divide the process into separate *segments* in memory
  - Each segment has a distinct purpose:
    - Example: code, static data, heap, stack
      - Maybe a separate segment for each function or object
    - Segments may be of different sizes
    - Stack and heap don't conflict
  - The whole process is still loaded into memory, but the segments that make up the process do ***not*** have to be loaded contiguously into memory
    - Space within a segment is contiguous
- Each segment has *protection bits*
  - Read-only segment (code)
  - Read-write segments (data, heap, stack)
  - Allows processes to share code and data

Segment Addresses

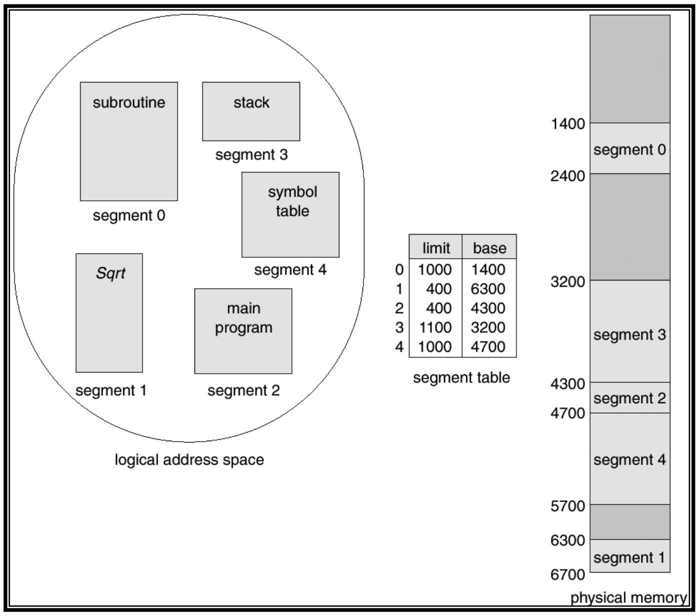
- Virtual (logical) address consists of:
  - Segment number
  - Offset from beginning of that segment
  - Both are generated by the assembler
- What is stored in the instruction?
  - Simple method:
    - Top bits of address specify segment
    - Bottom bits of address specify offset
  - Implicit segment specification:
    - Segment is selected implicitly by the instruction being executed (code vs. data)
    - Examples: PDP-11, Intel 386/486
  - Explicit segment specification:
    - Instruction prefix can request that a specific segment be used
    - Example: Intel 386/486…
    - Most common technique

Implementing Segments



- A *segment table* keeps track of every segment in a particular process
  - Each entry contains base and limit
  - Also contains protection information (sharing allowed, read vs. read/write)
- Additional hardware support required:
  - Multiple base and limit registers, or
  - Segment table base pointer (points to a segment table stored in a PCB)

Segmentation Example



Managing Segments

- When a process is loaded into memory:
  - Allocate space in physical memory for all of the process's segments
  - Create a (mostly empty) segment table, and store it in the process's PCB
- When a context switch occurs:
  - Update the segment table base pointer to point to the segment table in the new process's PCB
- If there's no space in physical memory:
  - Compact memory (move segments, update bases) to make contiguous space
    - Tradeoff efficiency for overhead
  - Swap one or more segments out to disk
    - To run that process again, swap *all* of its segments back into memory

Managing Segments (cont.)

- To enlarge a segment:
  - If space above the segment is free, OS can just update the segment's limit and use some of that space
  - Move this segment to a larger free space
  - Swap the segment above this one to disk
  - Swap this segment to disk, and bring it back into a larger free space
- Advantages of segmentation:
  - Segments don't have to be contiguous
  - Segments can be swapped independently
  - Segments allow sharing
- Disadvantages of segmentation:
  - Complex memory allocation (first-fit, etc.)
  - External fragmentation

Sharing Segments