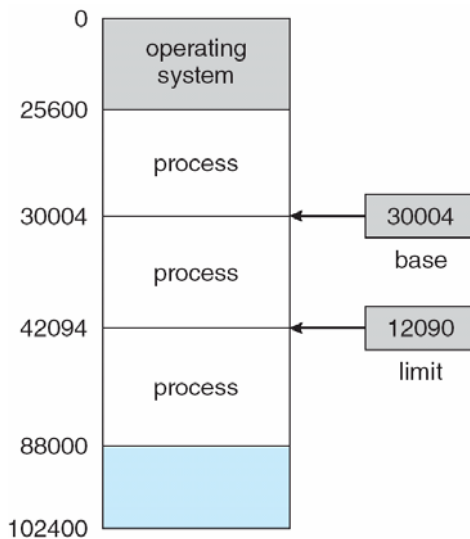


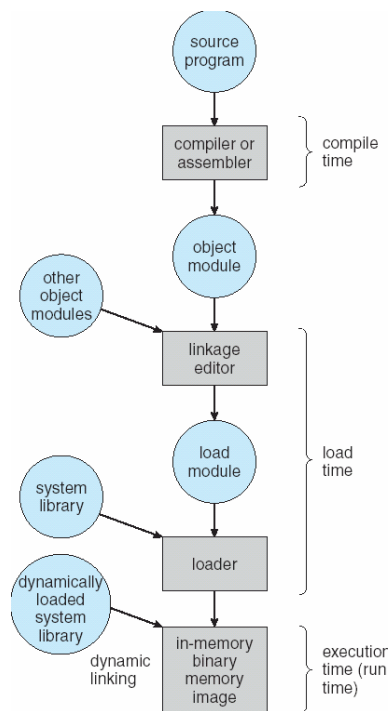
Fundamental Goal -- We must **share** memory

Our initial hardware assist to accomplish this: **base and limit registers**



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*).



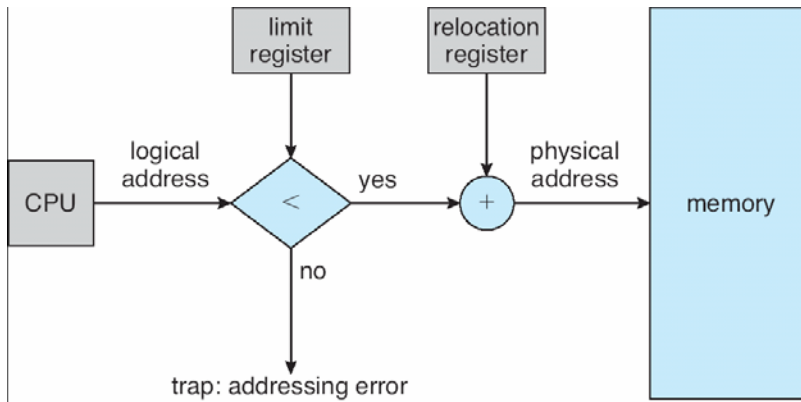
Logical vs. Physical Address Space

- 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

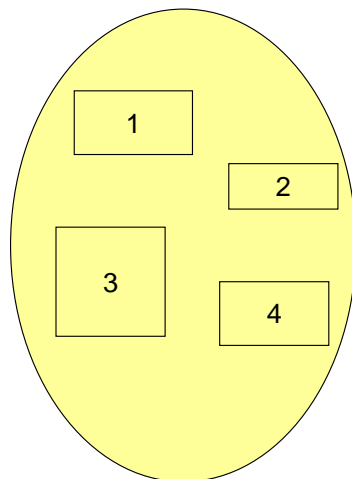
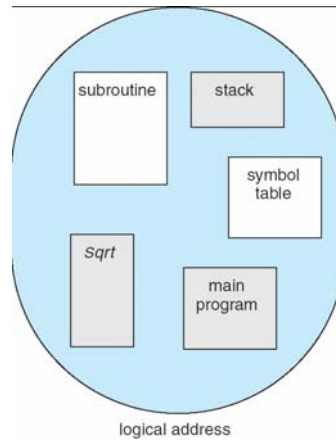
Memory Management Unit (MMU)

- Hardware device that maps logical 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

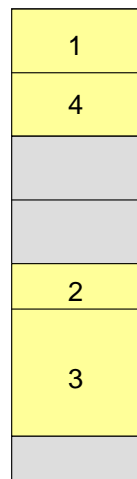


Segmentation

- 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



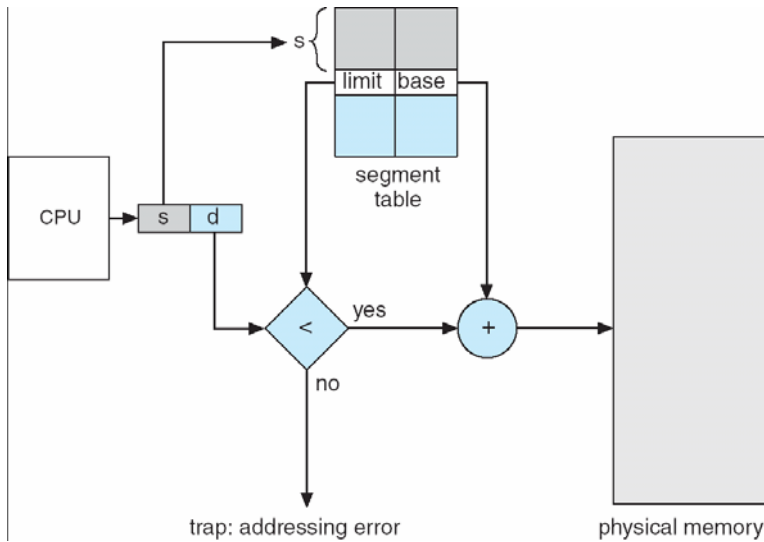
user space

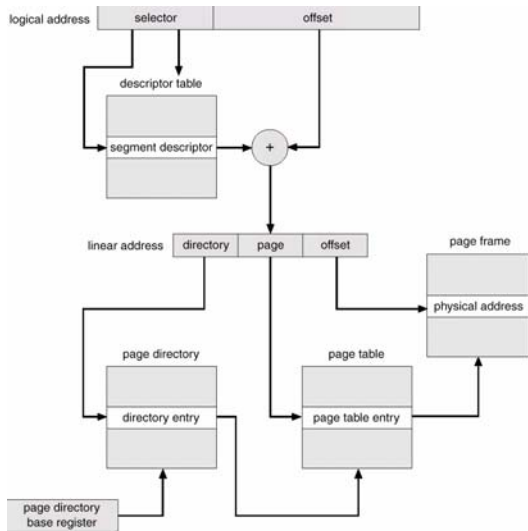
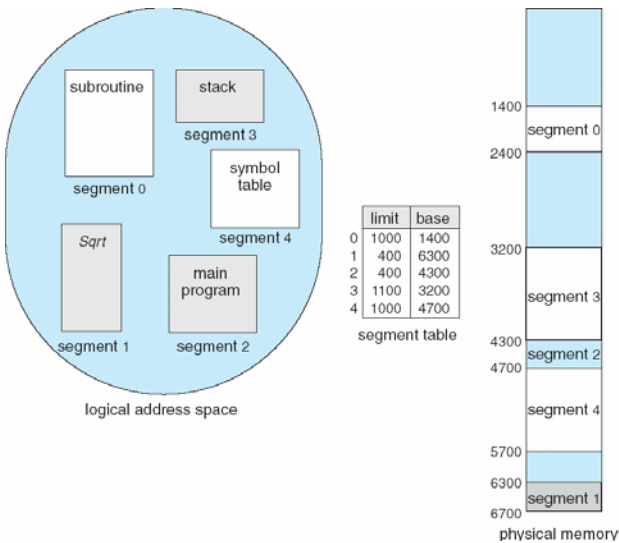


physical memory space

Segmentation Architecture

- Logical address consists of a two tuple: $\langle \text{segment-number}, \text{offset} \rangle$,
- **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 $s < \text{STLR}$
- **Relocation.**
 - dynamic
 - by segment table
- **Sharing.**
 - shared segments
 - same segment number
- **Allocation.**
 - first fit/best fit
 - external fragmentation
- 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





Pentium has 6 segment registers

CS, SS, DS, ES, FS, GS

Also has 6 microprogram registers to hold the associated segment descriptor to avoid memory refs on each translation.

A segment register holds a segment "selector"

A selector is 16 bits and uses 13 to hold the segment index (s), 1 bit for the gdt/ldt switch (g), and 2 bits for the privilege level (p).

The gdt/ldt switch is used to select between the GDT (one for the entire system and shared between all processes) and the LDT (private to the process)

Given the table, use (s) to index into the table.

- Uses minimal segmentation to keep memory management implementation more portable
- Uses 6 segments:
 - Kernel code
 - Kernel data
 - User code (shared by all user processes, using logical addresses)
 - User data (likewise shared)
 - Task-state (per-process hardware context)
 - LDT
- Uses 2 protection levels:
 - Kernel mode
 - User mode