Chapter 8:
Memory Management

Memory Management

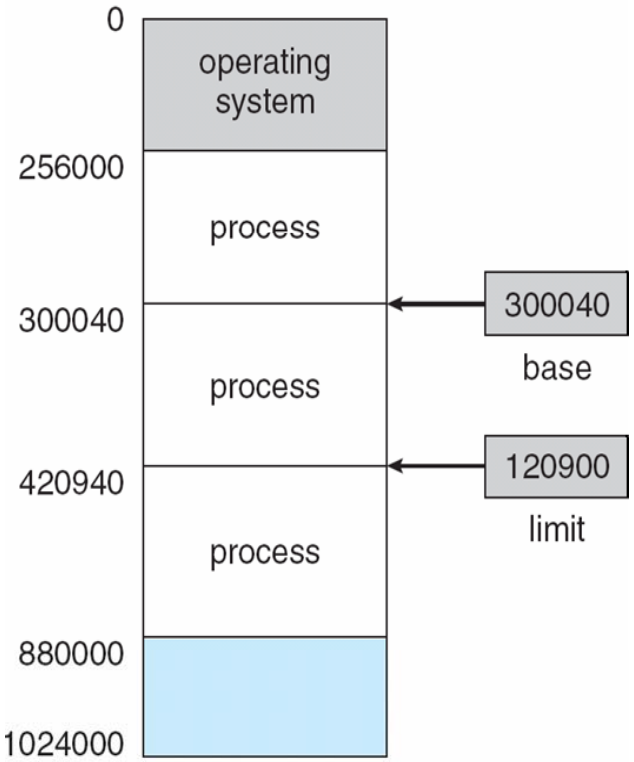
- Objectives
  - To provide a description of various ways of organizing memory hardware
  - To discuss memory management techniques, including paging and segmentation topics
- Topics
  - Background
  - Contiguous Memory Allocation
  - Paging
  - Structure of the Page Table
  - Segmentation
  - Swapping

Background

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are the only storage a CPU can access directly
  - Register access in one CPU clock cycle (or less)
  - Main memory access can take many cycles
- Cache sits between main memory and CPU registers
  - Simply to speed up memory-to-CPU-registers time
- Protection of memory required to ensure correct operation

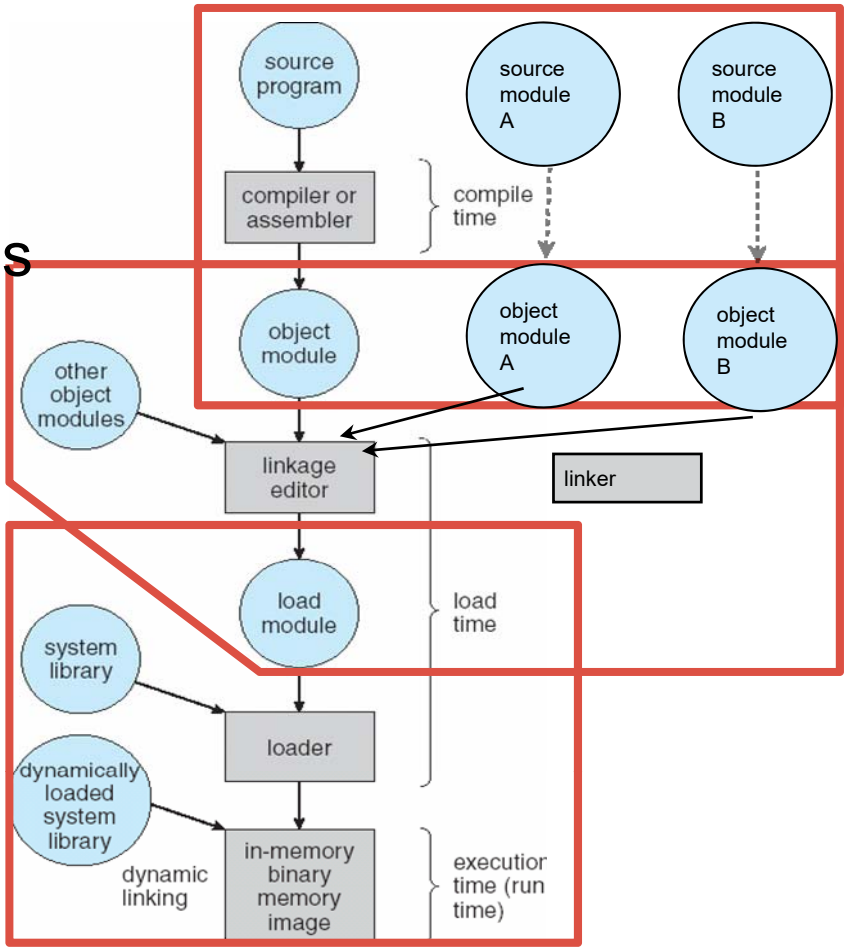
Base and Limit Registers

- A pair of base and limit registers defines the logical (and “legal”) address space for a process
  - Simple setup to map one process into memory



Multistep Processing of a User Program

- Translation
  - Source written in symbolic pro-gramming language (Assembler)
  - Conversion of symbolic instructions into machine-specific form
- Linking
  - Complex systems made up by multiple modules
  - Resolving of external references to other modules
  - Conversion from logical addres-ses to physical memory locations
- Loading
  - Transfer of code from disk to main memory
  - Resolving of external references to system libraries



Binding of Instructions & Data to Memory

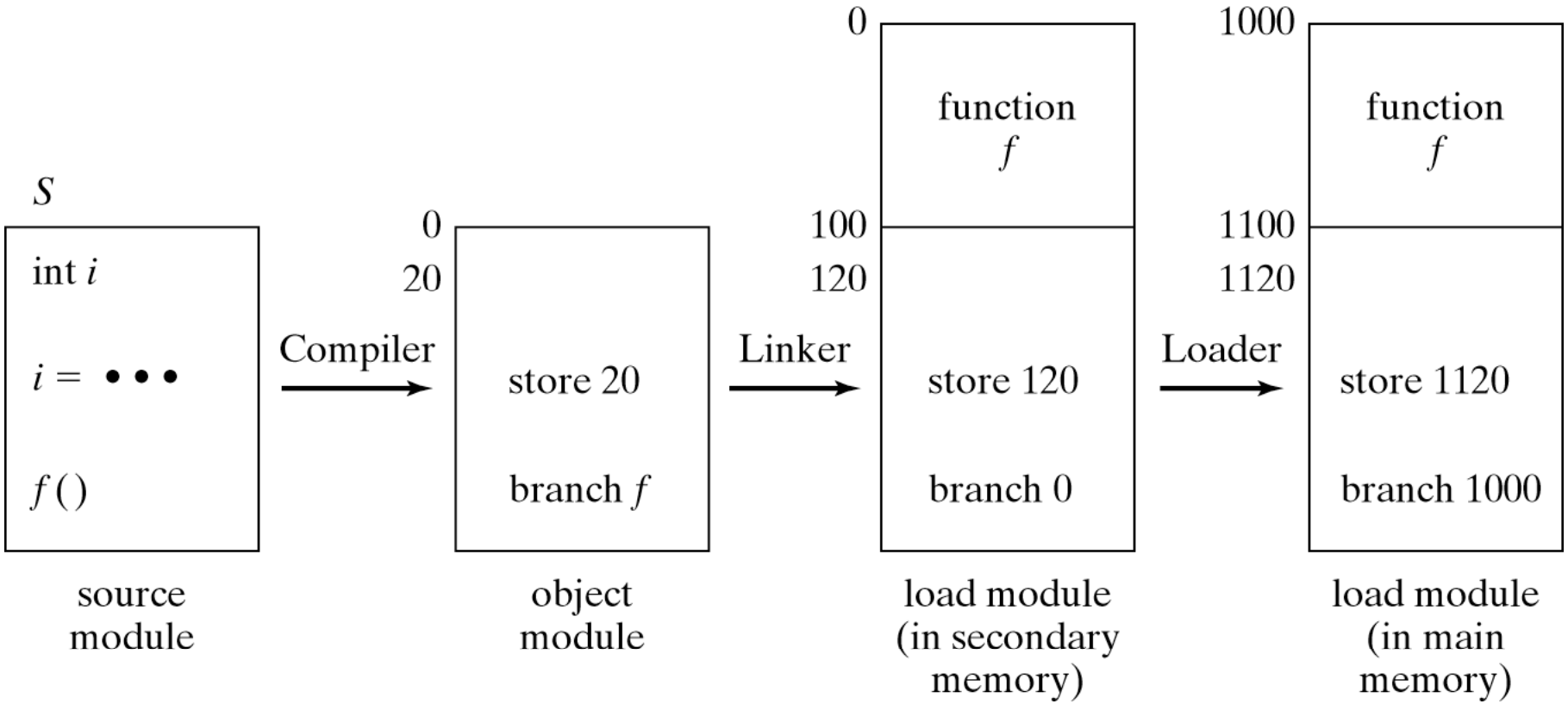
- Address binding of instructions and data to memory addresses can happen at three different stages
  - **Compile time**: If memory location known a priori, absolute code can be generated; must recompile code if starting loc. changes
    - Compiler and linker responsible for final static mapping of module into memory
  - **Load time**: Must generate relocatable code if memory location is not known at compile time
    - Linker and loader responsible to resolve relative addressing
  - **Execution time**: Binding delayed until run time, if the process can be moved during its execution from one memory segment to another
    - Needs hardware support for address maps (*e.g.*, base/limit regs.)

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, link-time, and load-time address-binding schemes
- Logical (virtual) and physical addresses differ in execution-time and with address-binding scheme

Static Address Binding

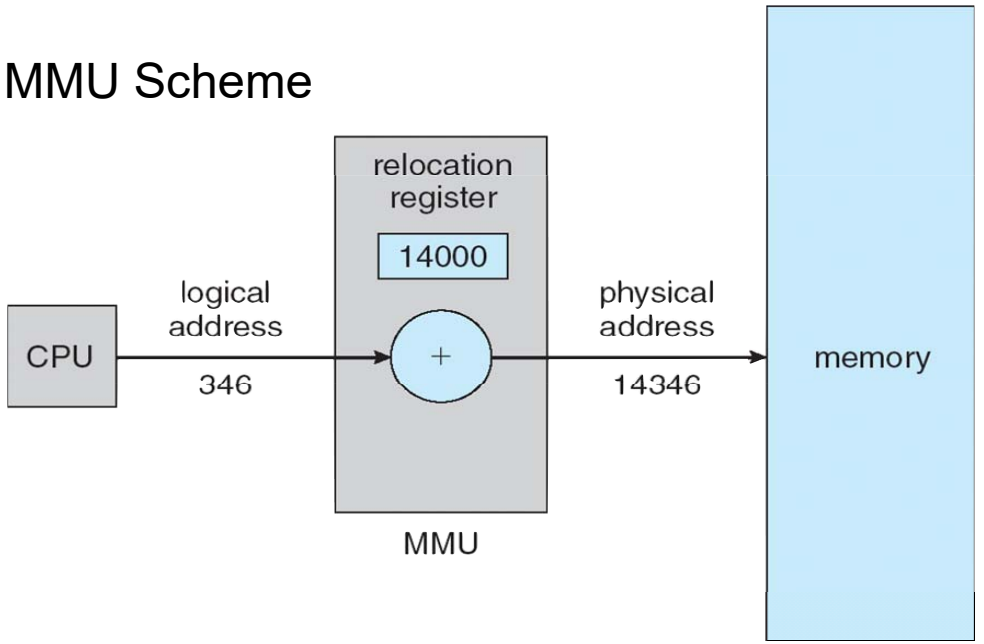
- At programming, compilation, linking, and/or loading time



Dynamic Relocation using a Relocation Register

- Use of Memory Management Unit (MMU)
  - Hardware device that maps virtual to physical addresses
- Relocation Register (RR) 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

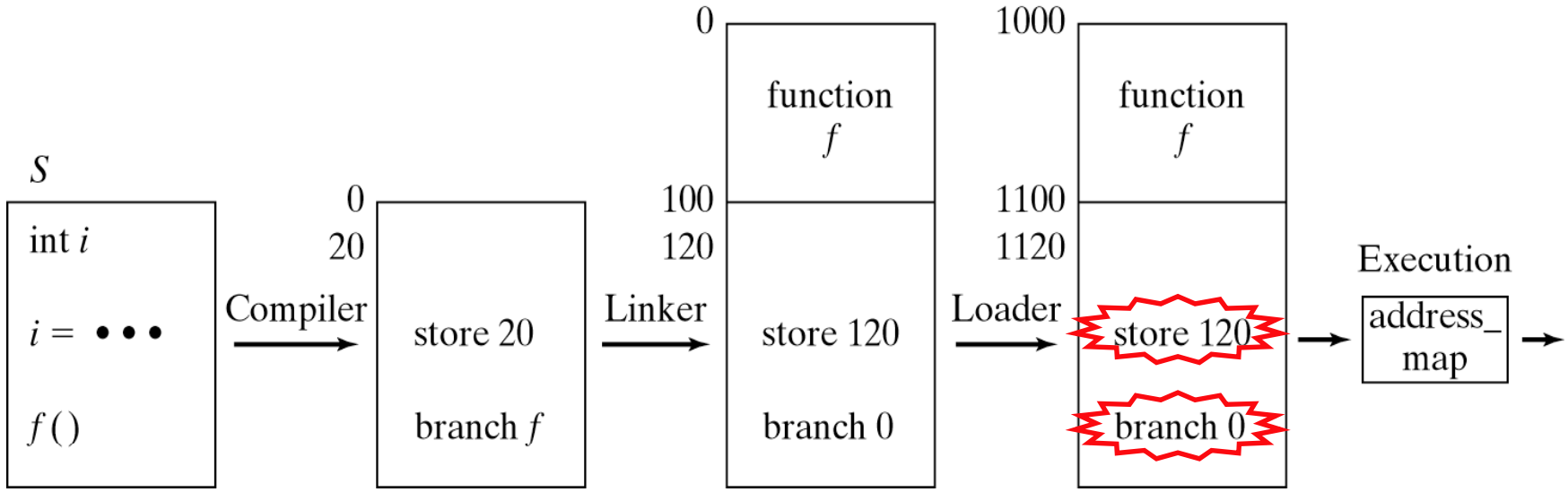
MMU Scheme



RR is a generalization of the base register.

Dynamic Address Binding

- At execution time
  - Loader does not change code



no change in code

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
- Dynamic relocatable code
  - Code with fixed starting address is not dynamically relocatable even if bound at load-time
  - Dynamically relocatable if:
    - Relocatable operands and variables are marked
    - Relative memory address given by relocation constant (offset) and relocation register (base address of module)

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

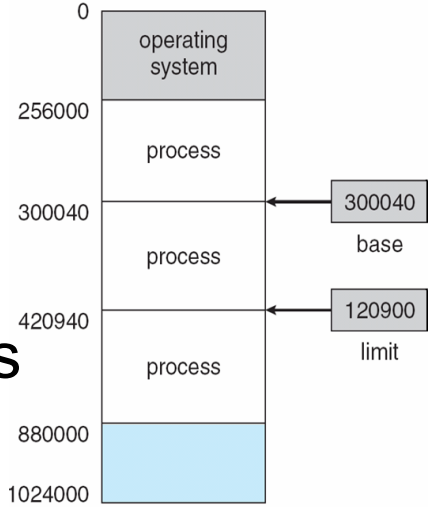
- Dynamic linking is particularly useful for libraries
- Also known as shared libraries
  - All processes execute the same library code

Main Memory Management

- All available free as well as occupied main memory must be managed dynamically at run-time
  - Processes' memory requirements change over time
- Divide into free (holes) and allocated memory (regions)
- Partitioning strategies
  - Fixed partitions with different fixed sized memory regions are very restricted
    - Single-program system with 2 partitions for OS & user prog.
    - Prog. size limited to largest partition, internal fragmentation
  - Variable partitioning allows allocation of free space on request
    - Manage variable sized memory regions
    - External or internal fragmentation depending on allocation strategy and implementation

Contiguous Allocation

- Main memory usually partitioned into two partitions
  - Resident operating system, usually held in low memory address range, together with the interrupt vector
  - User processes held in higher memory address ranges
    - Contiguous physical address range for each process
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - *Base register*: value of smallest physical address
  - *Limit register*: range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address dynamically



Dynamic Storage Allocation Problem

- Problem

  Given a request for ***n*** bytes, find free hole of size ≥ ***n***

  - Generally more than one available

- Constraints

  - Maximize memory utilization

    (minimize external fragmentation)

    - Limit the creation of too small holes
  - Minimize search time

Allocation Strategies

- 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
- Consequence: Memory will be divided into alternating sections, allocated in partitions and holes

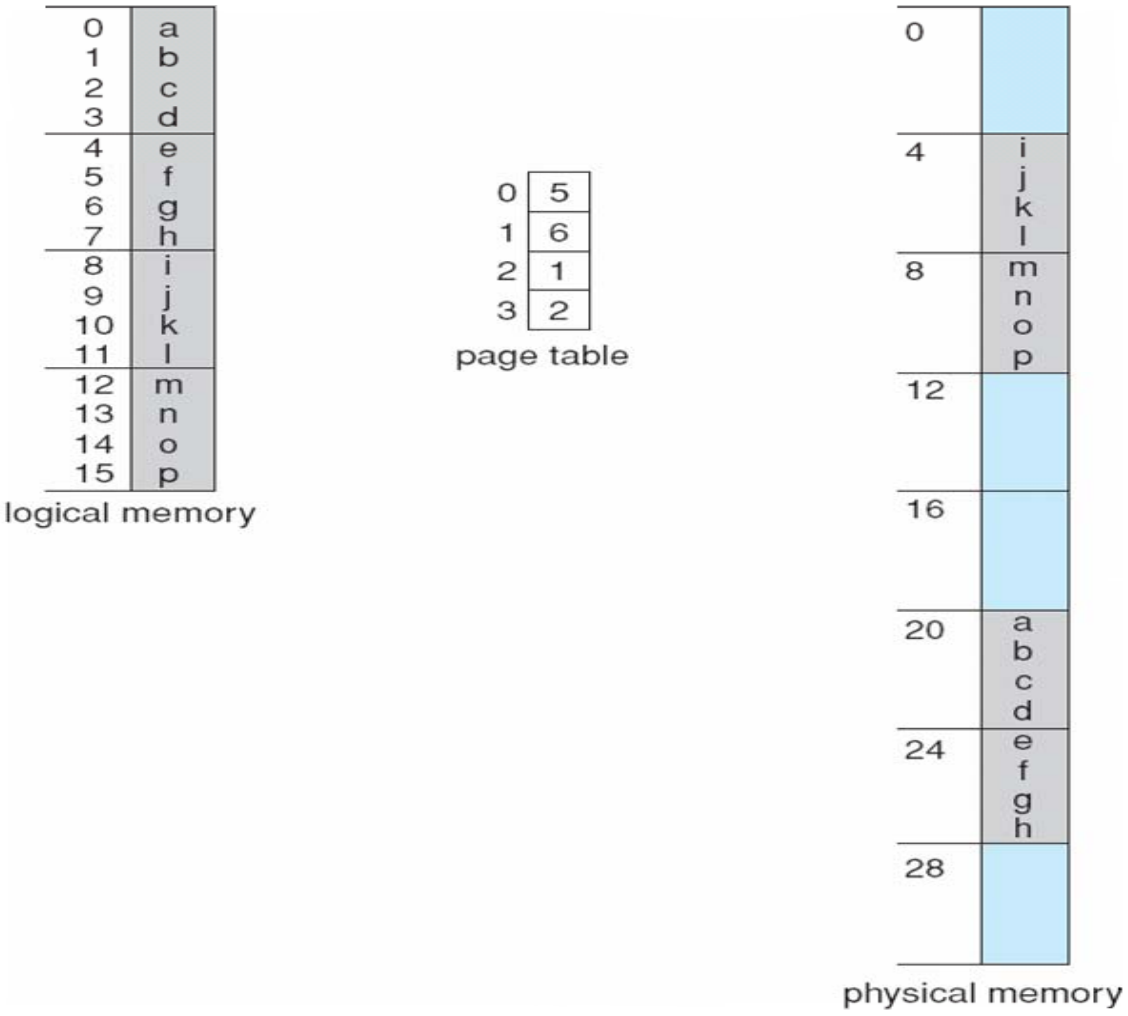
Fragmentation

- External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous
  - Caused by any of the first-fit, best-fit, or worst-fit strategies
    - Concentration of holes in one area of memory
    - Excessive number of small holes
    - Reduced availability of large partitions
- Internal Fragmentation – allocated memory may be larger than requested memory; this size difference is memory-internal to a partition, but not being used, and driven by allocation granularity
- Reduce external fragmentation by compaction
  - Shuffle memory content to place all free memory in one large block
  - Compaction is possible only if code relocation is dynamic (execution time)
  - I/O problems: 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 in memory
  - 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 Byte and 8,192 Byte)
- 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 addresses (pages) to physical addresses (frames)
- Causes some internal fragmentation, avoids external fragmentation

Paging Example



32 Byte physical memory and 4 Byte page size

Implementation of Page Tables

- Page table is kept in main memory
  - One page table for each process, which can be very large
- 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 problems can be solved by the use of a special fast-lookup hardware cache called associative memory or Translation Look-aside Buffers (TLB)
- Some TLBs store Address-space Identifiers (ASID) in each TLB entry – uniquely identifies each process
  - Provides address-space protection

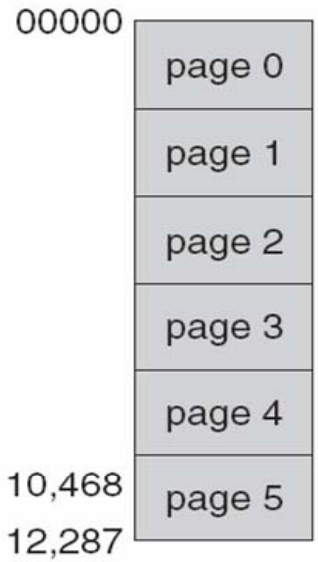
Memory Protection

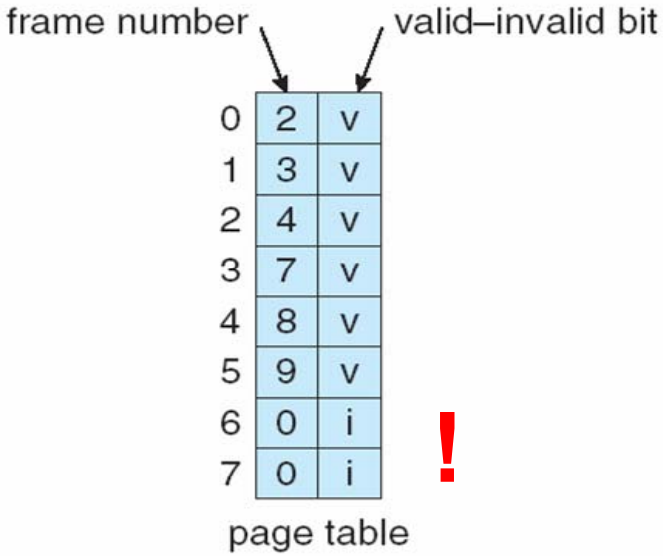
- Memory protection implemented by associating protection bit with each frame

- 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

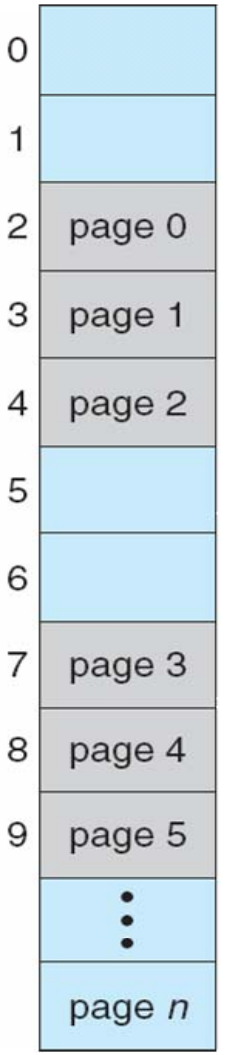
Valid (v) or Invalid (i) Bit in a Page Table

Assumptions
14 bit address space (0 ... 16383)
Program addresses limited to 0 ... 10468
2 kByte page sizes





Trap generated in case of address generation in pages 6 or 7.



Structure of the Page Tables

- Hierarchical Paging

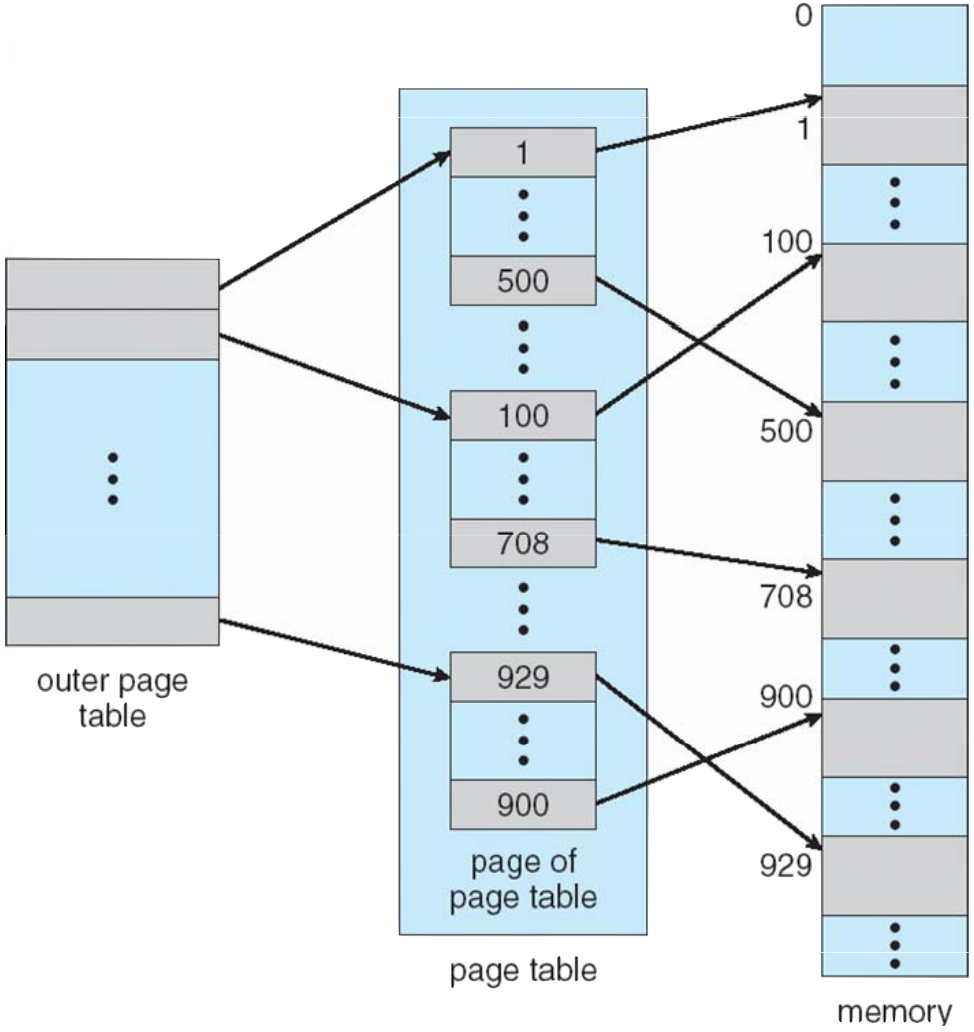
  (Detailed within the next slides)

- Hashed Page Tables

- Inverted Page Tables
  - The number of entries is equal to the number of frames in main memory
    - Always space reserved for pages, although present in main memory or not. Waste of memory if the page is not present, thus, save all details only for pages being present in main memory.

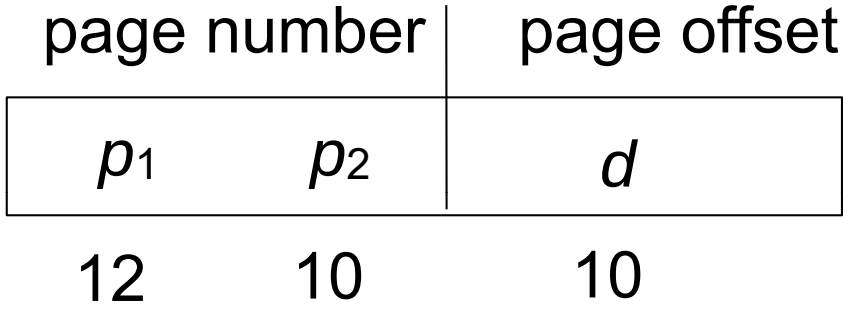
Hierarchical Two-Level Page Tables

- Break up of the logical address space into multiple page tables
  - *E.g.*, 64 Bit addresses
  - 4 kByte pages, 4 Gbyte main memory
    - One entry per virtual page: 2^{64} addressable bytes / 2^{12} Byte per page = 2^{52} page table entries
- A simple technique is a two-level (n-level) page table



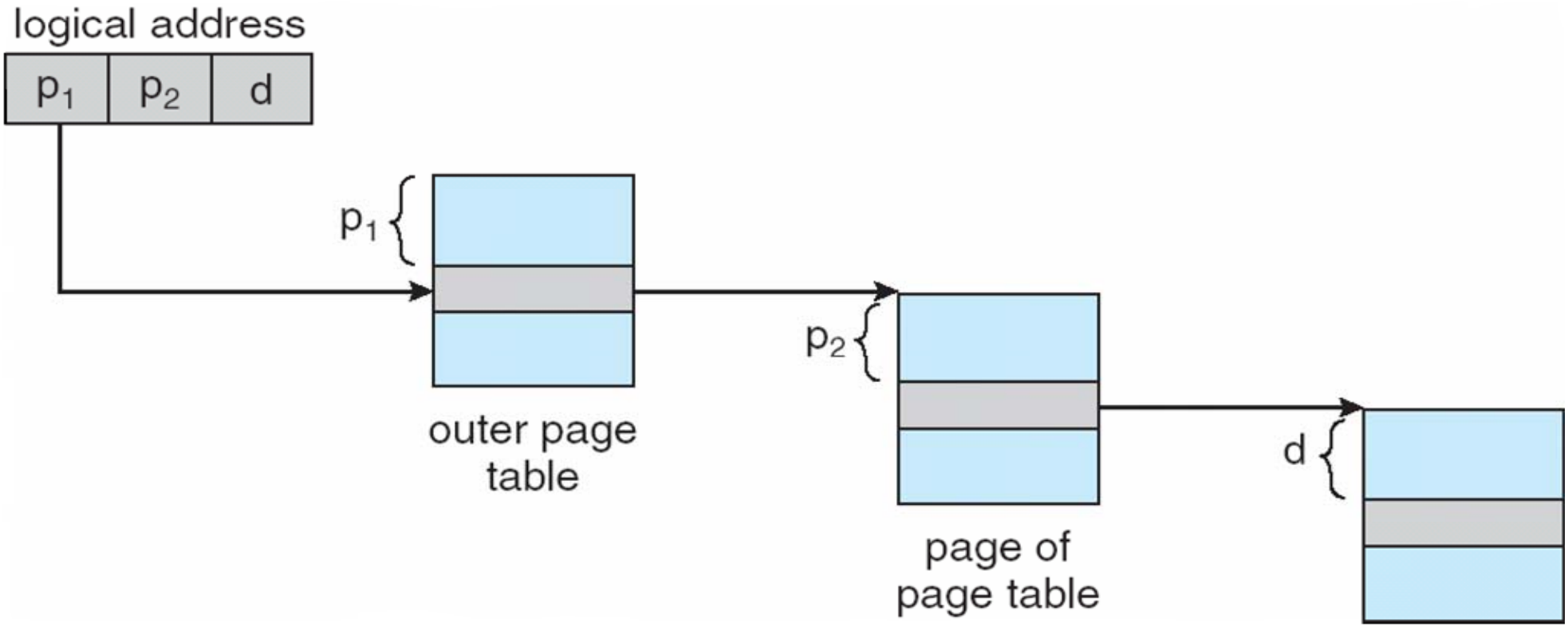
Two-Level Paging Example

- A logical address (on 32 bit machine with 1 K page size) is divided into
  - A page number consisting of 22 bits
  - A page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into
  - A 12 bit page number
  - A 10 bit page offset
- Thus, the logical address is as follows:



where p1 is an index into the outer page table, and p2 is the displacement within the page of the outer page table

Address-Translation Scheme

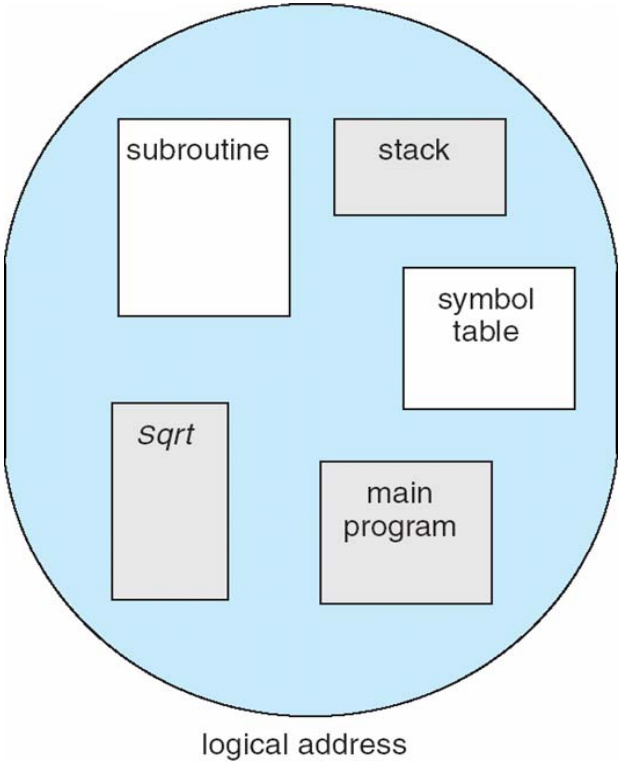


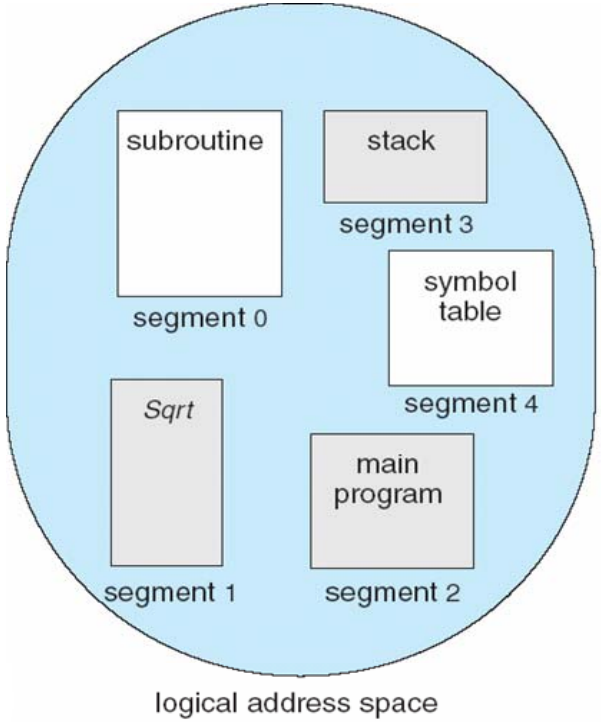
Segmentation

- Memory management scheme that supports logical structured 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
  - Paging is closer to the OS rather than the user, since it divides the process into pages regardless whether a process can have relative parts of functions needed to be loaded in the same page.

Example of Segmentation

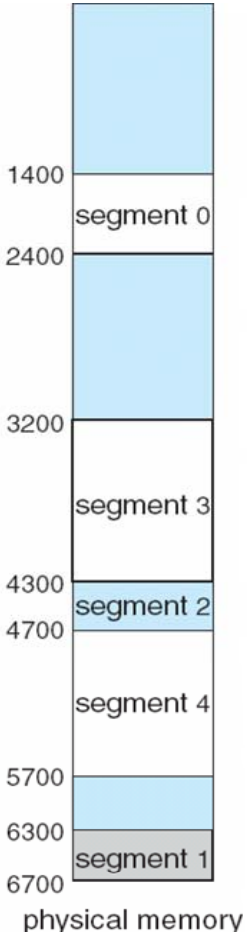
User's View of a Program





	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



Segmentation Architecture (1)

- Logical address consists of a two-tuple:
  [segment-number, offset]

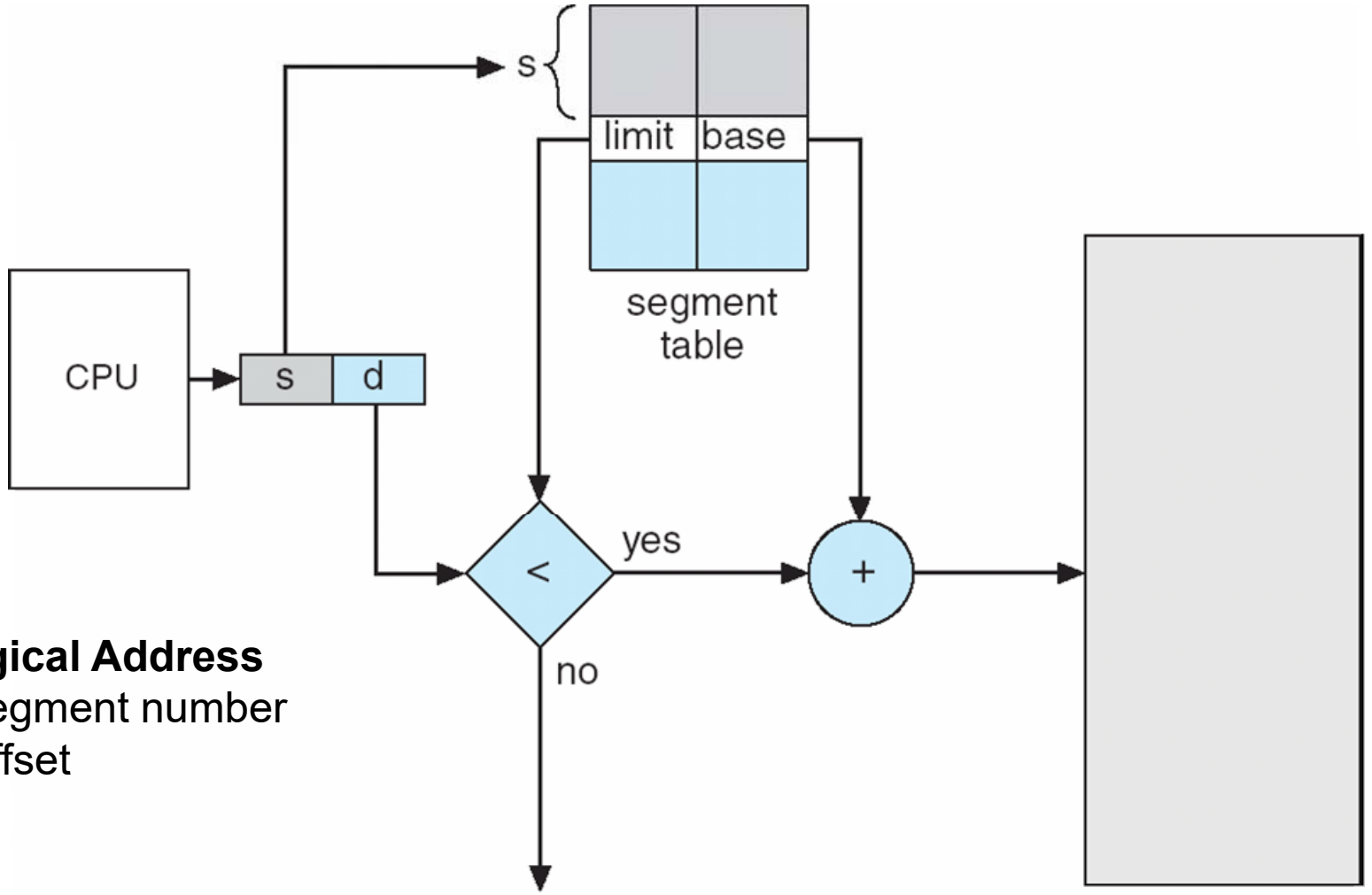
- Segment table – maps segments to physical addresses
  - Each table entry has:
    - Base – contains the starting physical address where the segments reside in memory
    - Limit – specifies the length of the segment

- 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 (2)

- Protection
  - With each entry in segment table associate:
    - Validation bit = 0 ⇒ 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 again a dynamic storage-allocation problem
- Segmentation can be combined with paging
  - And use of TLBs (Translation Look-aside Buffer) and hash tables for performance optimization

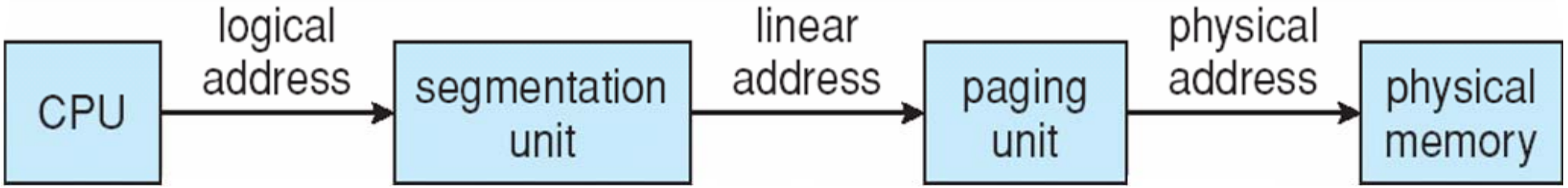
Segmentation Hardware



Logical Address
s segment number
d offset

Example: Intel Pentium

- Supports both segmentation & segmentation with paging
- CPU generates logical address
  - Given to segmentation unit (4 kByte or 4 MByte)
    - Which produces linear addresses
  - Linear address given to paging unit (2 level paging)
    - Which generates physical address in main memory
    - Paging units form equivalent of MMU
- 8K+8K of private/shared segments per process
  - Local descriptor (LDT) and global descriptor tables (GDT)



Insufficient Memory

- Swapping (next slides)
  - Temporarily move process and its memory to disk
  - Requires dynamic relocation
- Memory compaction
  - Reduce external fragmentation by moving holes
  - How much and what to move?
- Overlays
  - Allow programs larger than physical memory
  - Programs loaded as needed according to calling structure

Swapping

- A process can be swapped temporarily out of memory to external storage and brought back into main memory for cont'd execution
  - External storage (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
  - 100 MB user process may need a second or more to swap-in before being ready to execute on the CPU
- Variants of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - System maintains a ready queue of ready-to-run processes, which have memory images on disk

Schematic View of Swapping