CHAPTER 8: MEMORY MANAGEMENT

- Background
- Logical versus Physical Address Space
- Swapping
- Contiguous Allocation
- Paging
- Segmentation
- Segmentation with Paging

Background

- Program must be brought into memory and placed within a process for it to be executed.

- *Input queue* – collection of processes on the disk that are waiting to be brought into memory for execution.

- User programs go through several steps before being executed.

Address binding of instructions and data to memory addresses can happen at three 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).

- 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.
  - No special support from the operating system is required; implemented through program design.

- 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.

- Overlays – keep in memory only those instructions and data that are needed at any given time.
  - Needed when process is larger than amount of memory allocated to it.
  - Implemented by user, no special support needed from operating system; programming design of overlay structure is complex.

Logical versus 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 virtual to physical address.

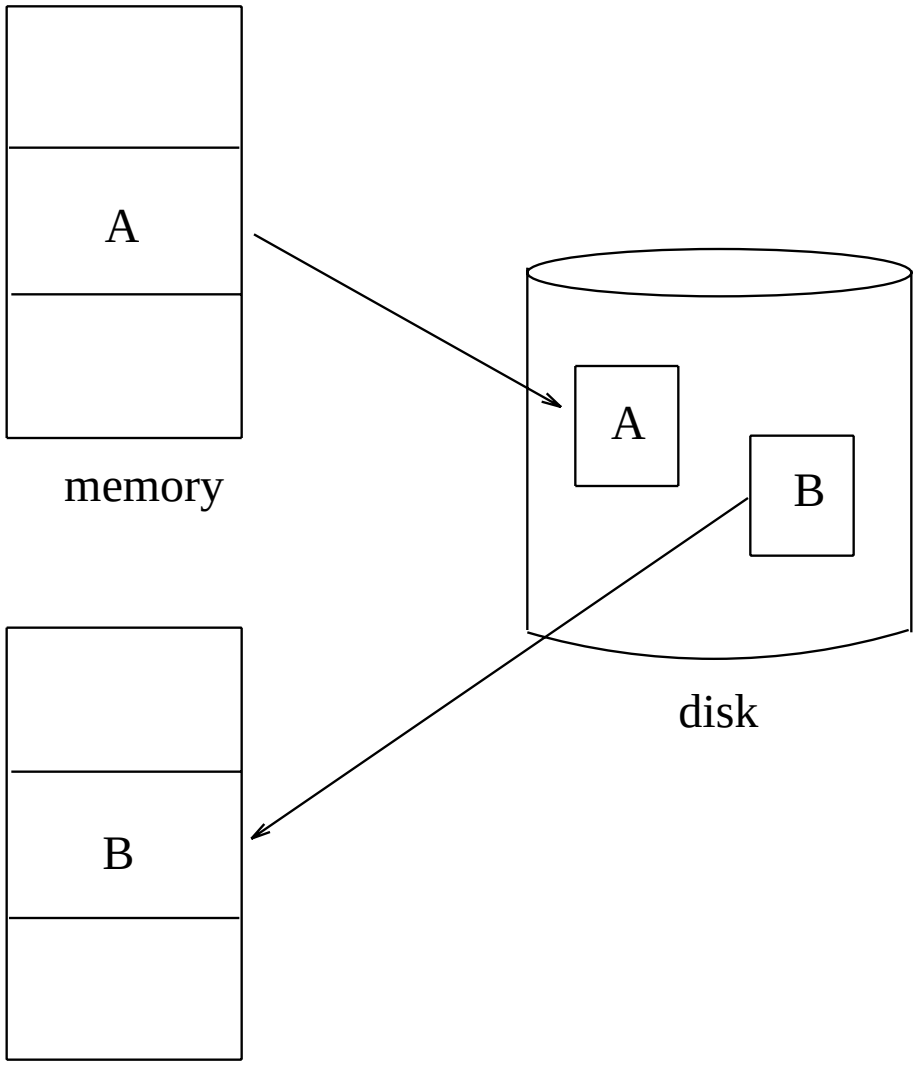
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.

- The user program deals with *logical* addresses; it never sees the *real* physical addresses.

Swapping

- A process can be *swapped* temporarily out of memory to a *backing store*, and then brought back into memory for continued execution.
- 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.
- Modified versions of swapping are found on many systems, i.e., UNIX and Microsoft Windows.

- Schematic view of swapping

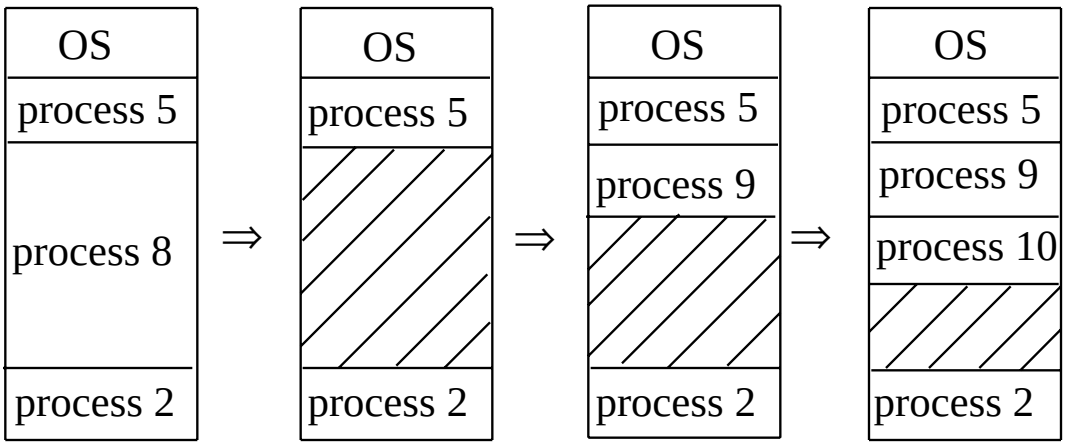


Contiguous Allocation

- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes then held in high memory.
- Single-partition allocation
  - Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data.
  - Relocation register contains value of smallest physical address; limit register contains range of logical addresses – each logical address must be less than the limit register.

- Multiple-partition allocation
  - *Hole* – block of available memory; holes of various size are scattered throughout memory.
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it.

- Example



Operating system maintains information about:

- allocated partitions
- free partitions (hole)

- *Dynamic storage-allocation* problem – 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.

- External fragmentation – total memory space exists to satisfy a request, but it is not contiguous.

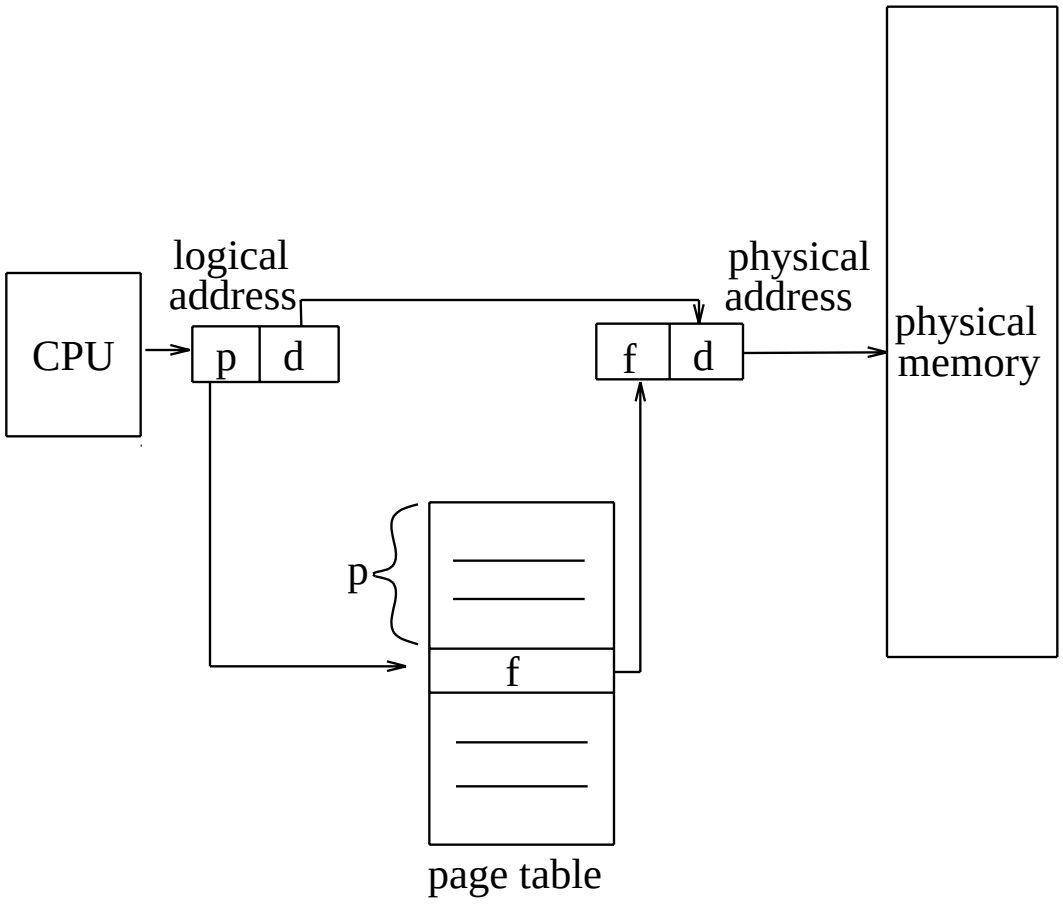
- Internal fragmentation – allocated memory may be slightly larger than requested memory; difference between these two numbers is memory internal to a partition, but not being used.

- Reduce external fragmentation by *compaction.*
  - Shuffle memory contents to place all free memory together in one large block.
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time.
  - I/O problem
    - Latch job in memory while it is involved in I/O.
    - Do I/O only into OS buffers.

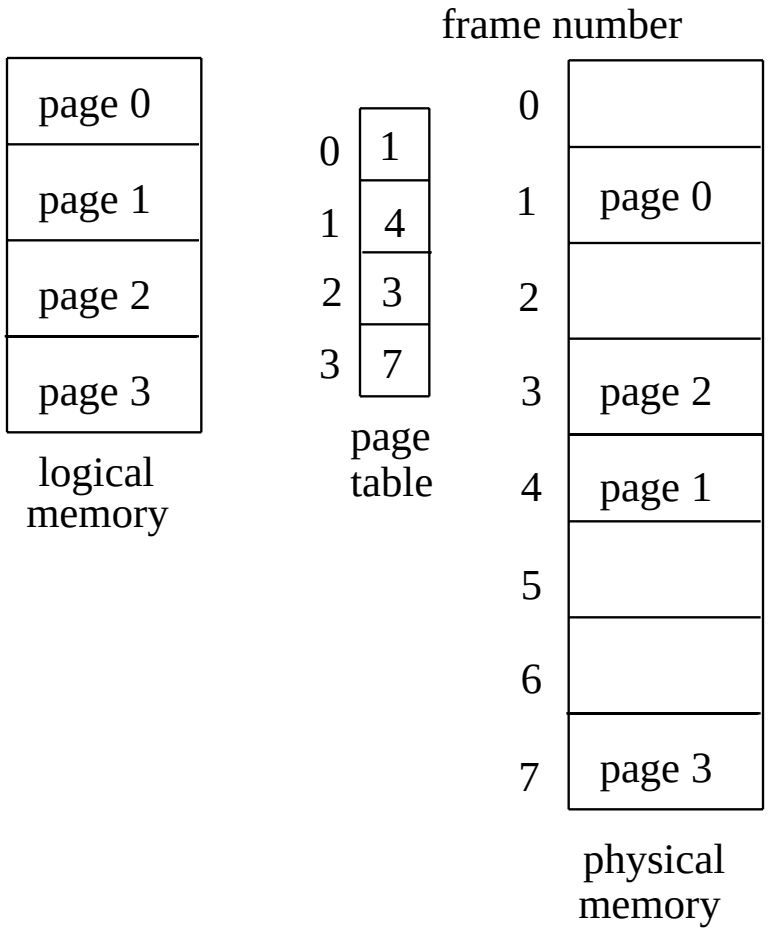
Paging – logical address space of a process can be noncontiguous; process is allocated physical memory wherever the latter is available.

- Divide physical memory into fixed-sized blocks called *frames* (size is power of 2, between 512 bytes and 8192 bytes).
- 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 to physical addresses.
- Internal fragmentation.

- Address generated by CPU is divided into:
  - *Page number (p)* – used as an index into a *page table* which contains base address of each page in physical memory.
  - *Page offset (d)* – combined with base address to define the physical memory address that is sent to the memory unit.



- Separation between user's view of memory and actual physical memory reconciled by address-translation hardware; logical addresses are translated into physical addresses.



Implementation of page table

- Page table is kept in main memory.
- *Page-table base register (PTBR)* points to the page table.
- *Page-table length register (PTLR)* 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 problem can be solved by the use of a special fast-lookup hardware cache called *associative registers* or *translation look-aside buffers (TLB*s*).*

- Associative registers – parallel search

Page #	Frame #

Address translation (A′, A″)

- If A′ in associative register, get frame # out.
- Otherwise get frame # from page table in memory.

- *Hit ratio* – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers.

- Effective Access Time (EAT)
  - associative lookup = ε time unit
  - memory cycle time - 1 microsecond
  - hit ratio = α

EAT= $(1 + \varepsilon)\,\alpha + (2 + \varepsilon)\,(1 - \alpha)$

$= 2 + \varepsilon - \alpha$

- Memory protection implemented by associating protection bits with each frame.

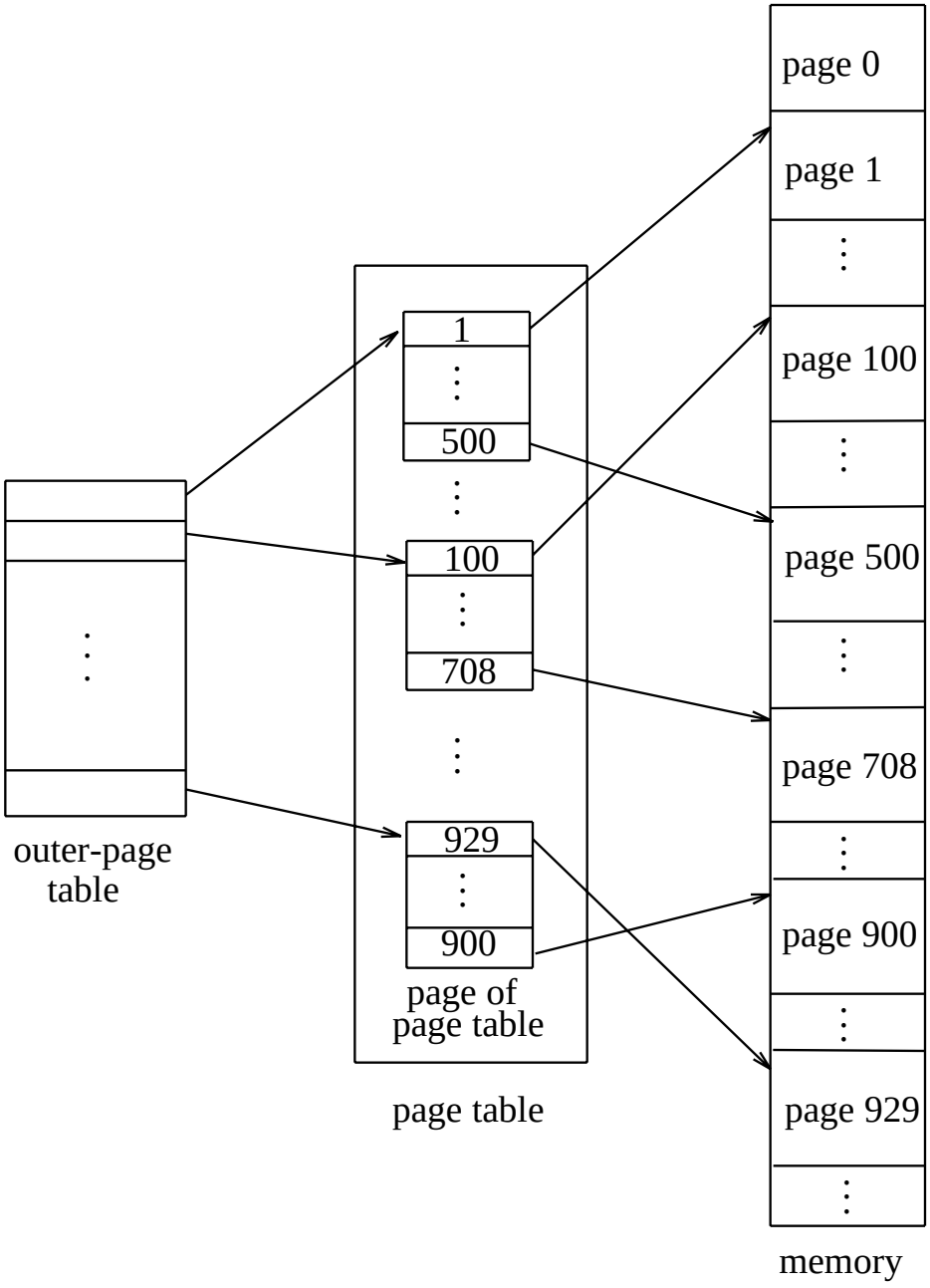
- *Valid–invalid* bit attached to each entry in the page table:

  - ‘‘valid’’ indicates that the associated page is in the process’ logical address space, and is thus a legal page.

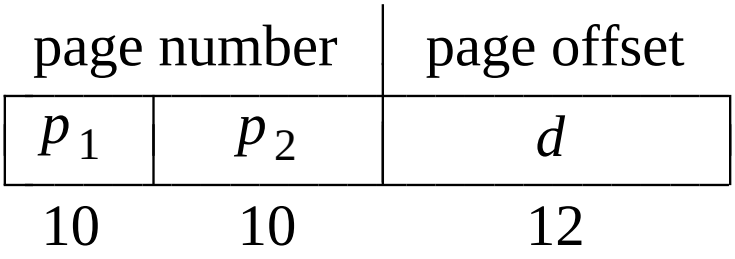
  - ‘‘invalid’’ indicates that the page is not in the process’ logical address space.

Multilevel Paging – partitioning the page table allows the operating system to leave partitions unused until a process needs them.

- A two-level page-table scheme



- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page number consisting of 20 bits.
  - a page offset consisting of 12 bits.
- Since the page table is paged, the page number is further divided into:
  - a 10-bit page number.
  - a 10-bit page offset.
- Thus, a logical address is as follows:

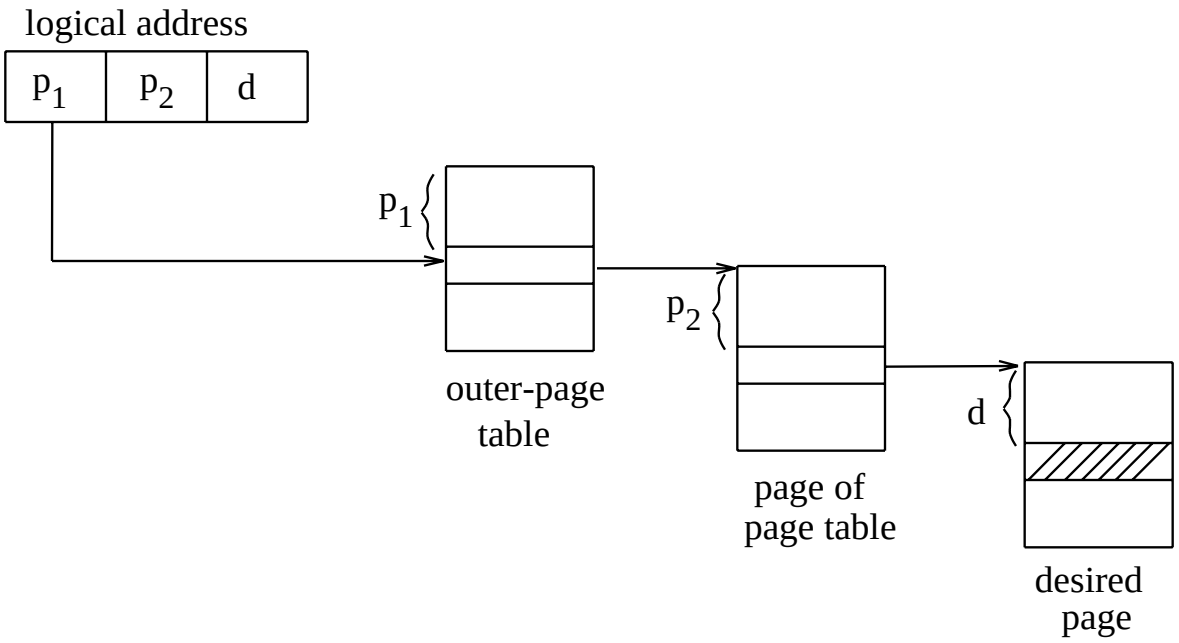


where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the outer page table.

- Thus, a logical address is as follows:

page number		page offset
p_1	p_2	d
10	10	12

- Address-translation scheme for a two-level 32-bit paging architecture



- Multilevel paging and performance

  - Since each level is stored as a separate table in memory, converting a logical address to a physical one may take four memory accesses.

  - Even though time needed for one memory access is quintupled, caching permits performance to remain reasonable.

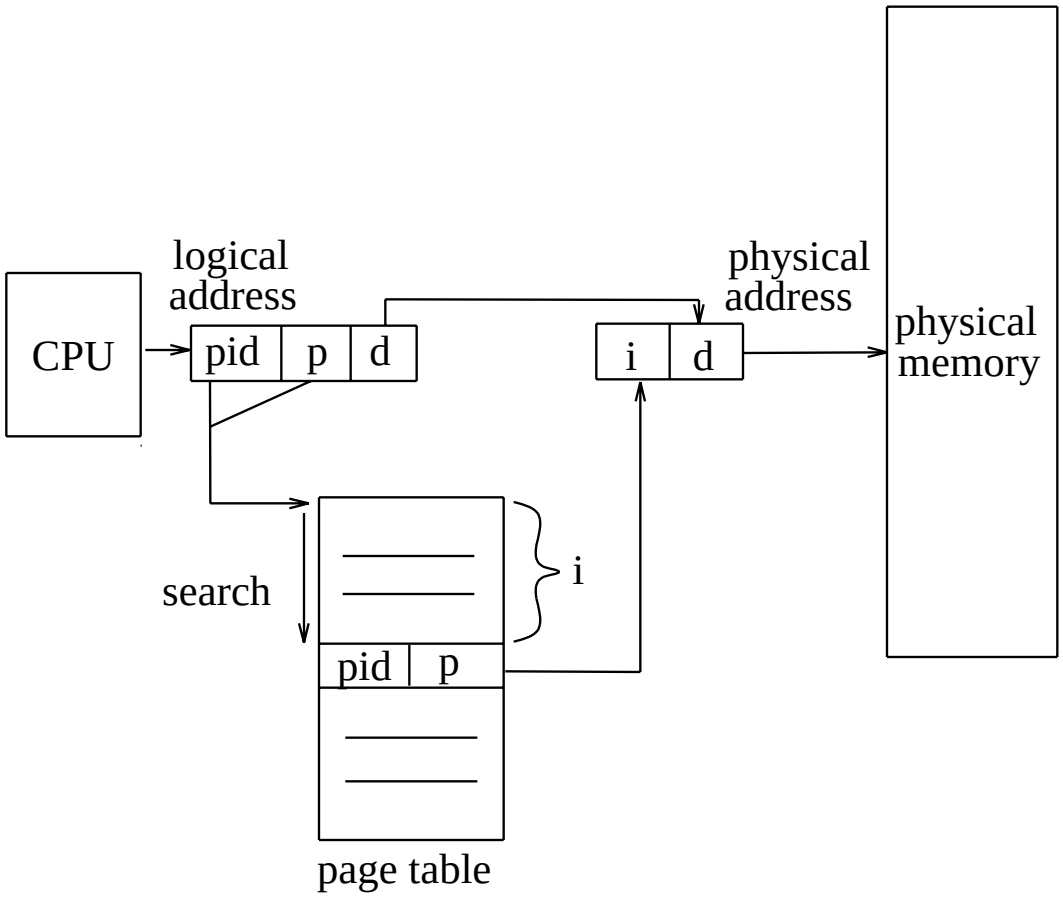
  - Cache hit rate of 98 percent yields:

$$\begin{aligned} \text{effective access time} &= 0.98 \times 120 + 0.02 \times 520 \\ &= 128 \text{ nanoseconds} \end{aligned}$$

    which is only a 28 percent slowdown in memory access time.

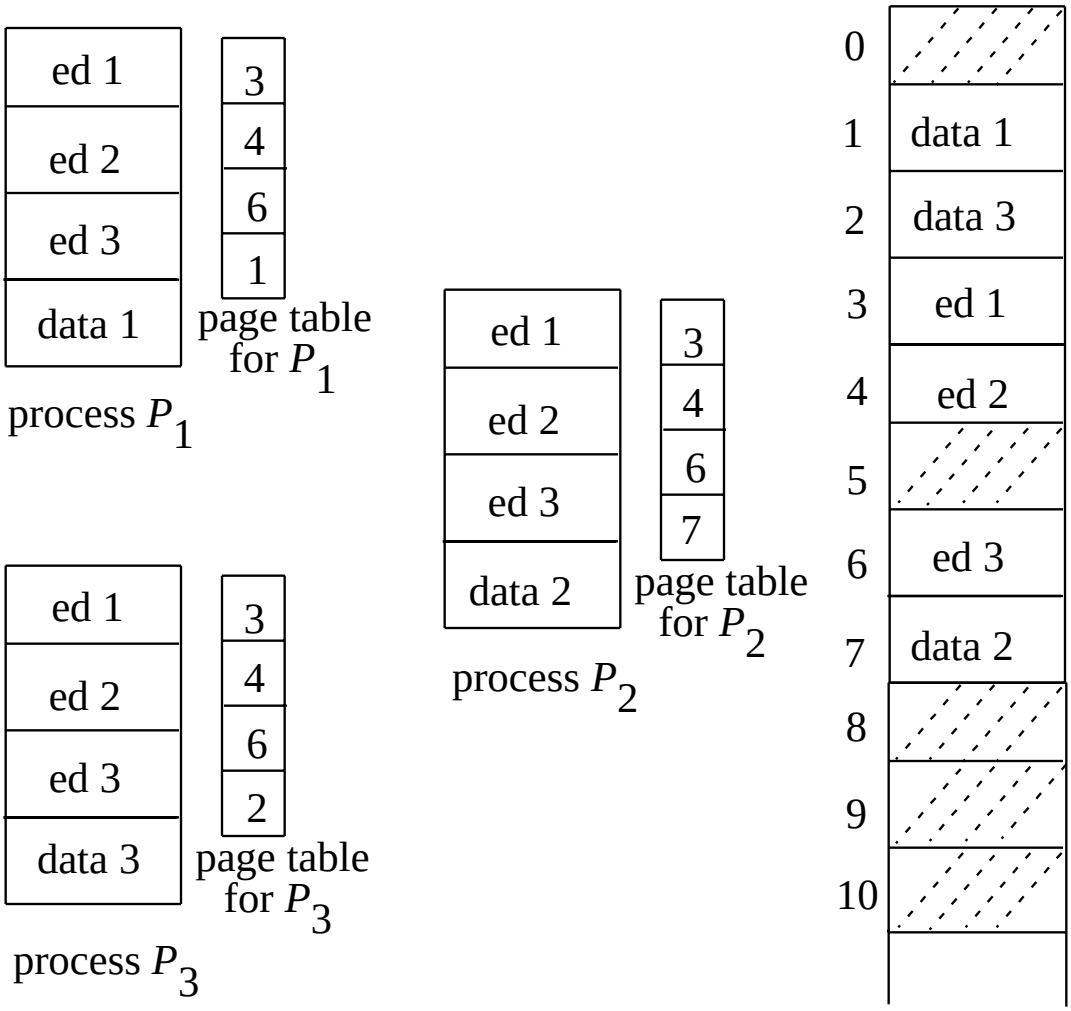
Inverted Page Table – one entry for each real page of memory; entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.

- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.
- Use hash table to limit the search to one — or at most a few — page-table entries.



Shared pages

- One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).

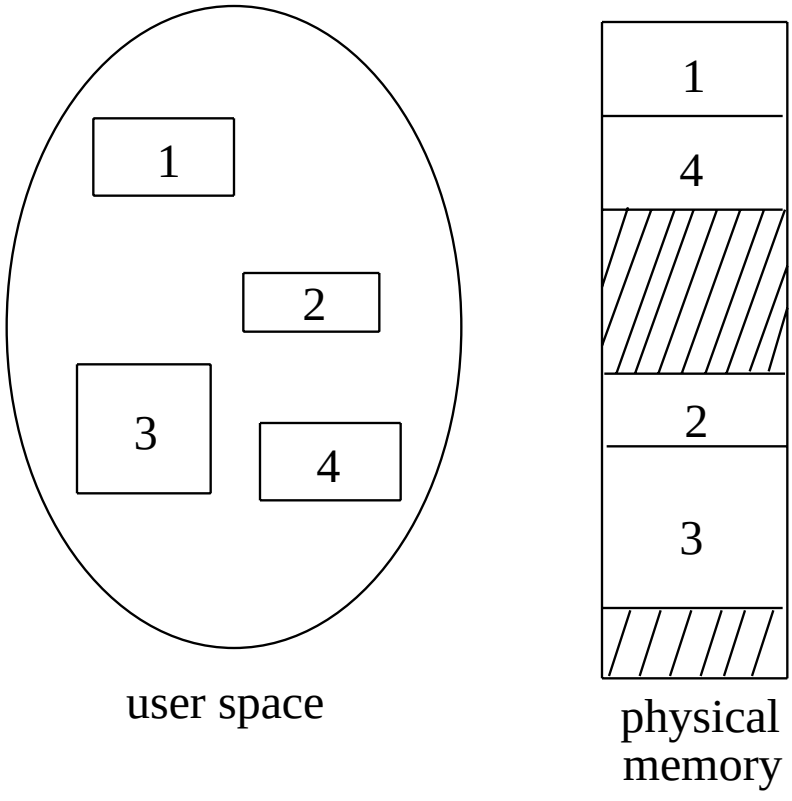


Segmentation – memory-management scheme that supports user view of memory.

- A program is a collection of segments. A segment is a logical unit such as:

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

- Example



- Logical address consists of a two tuple:

  <segment-number, offset>.

- *Segment table* – maps two-dimensional user-defined addresses into one-dimensional physical addresses; each entry of table 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.

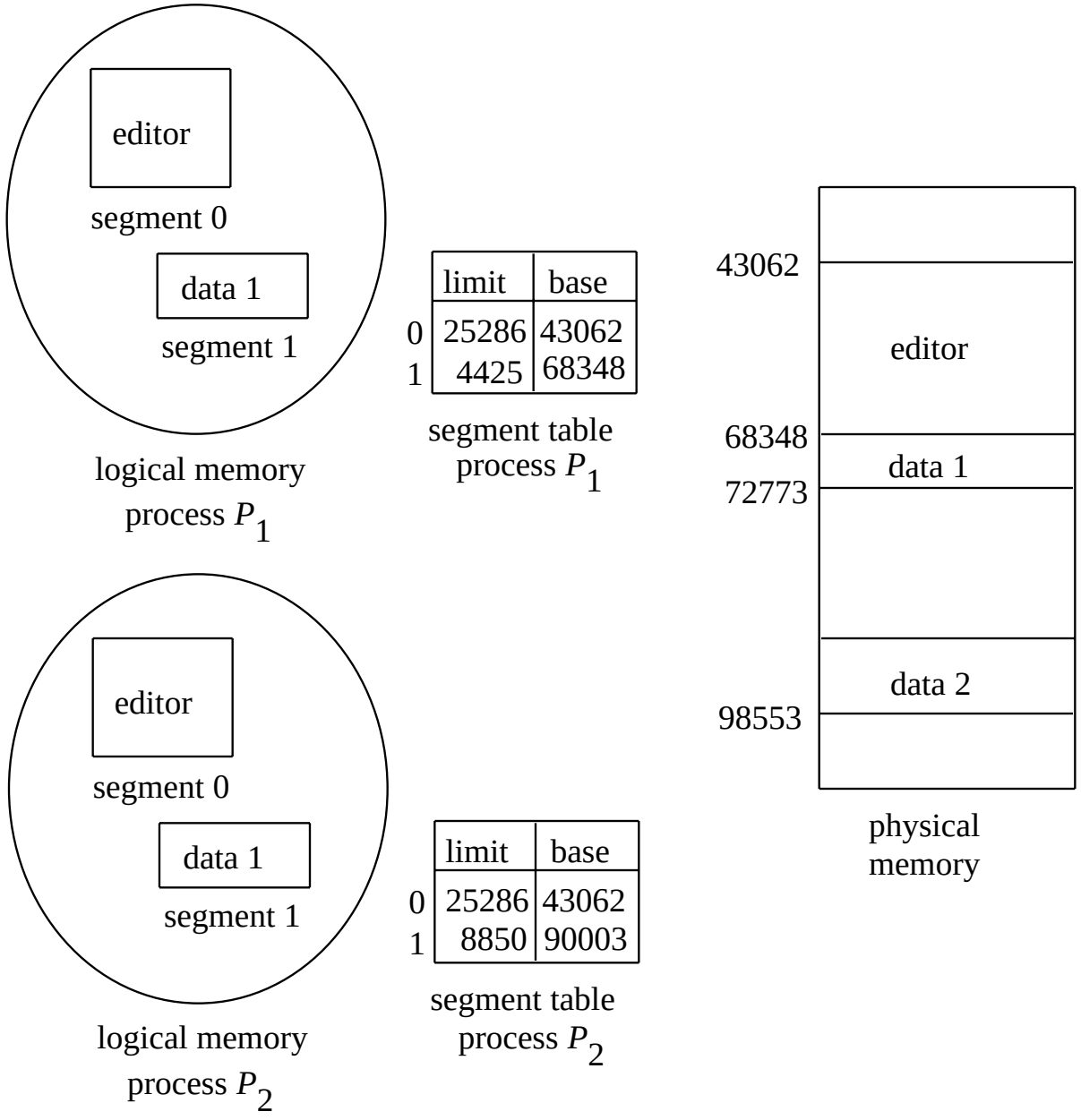
Relocation
- dynamic
- by segment table

Sharing
- shared segments
- same segment number

Protection With each entry in segment table associate:
- validation bit = 0 $\Rightarrow$ illegal segment
- read/write/execute privileges

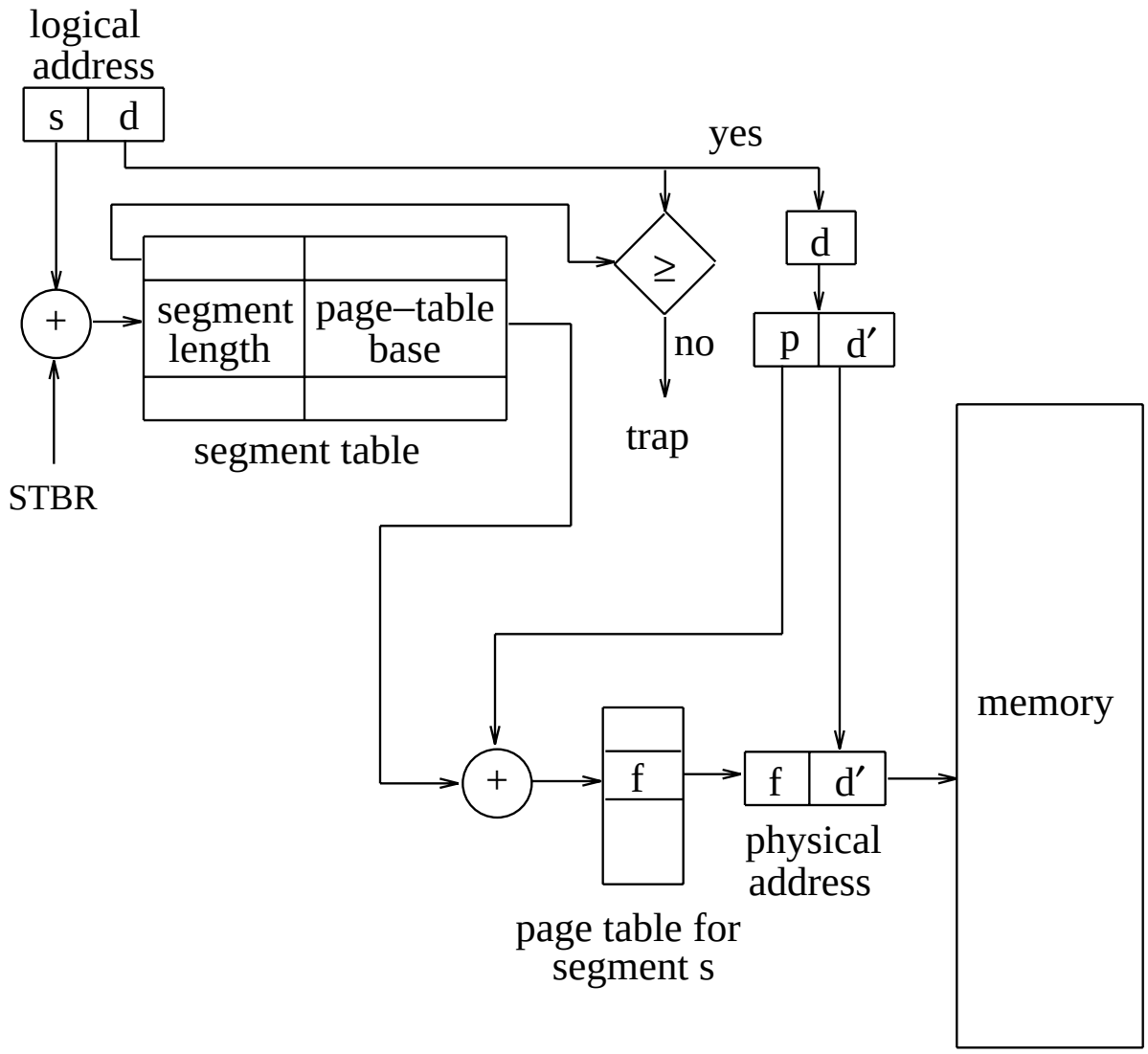
Allocation
- first fit/best fit
- external fragmentation

- Protection bits associated with segments; code sharing occurs at segment level.
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem.



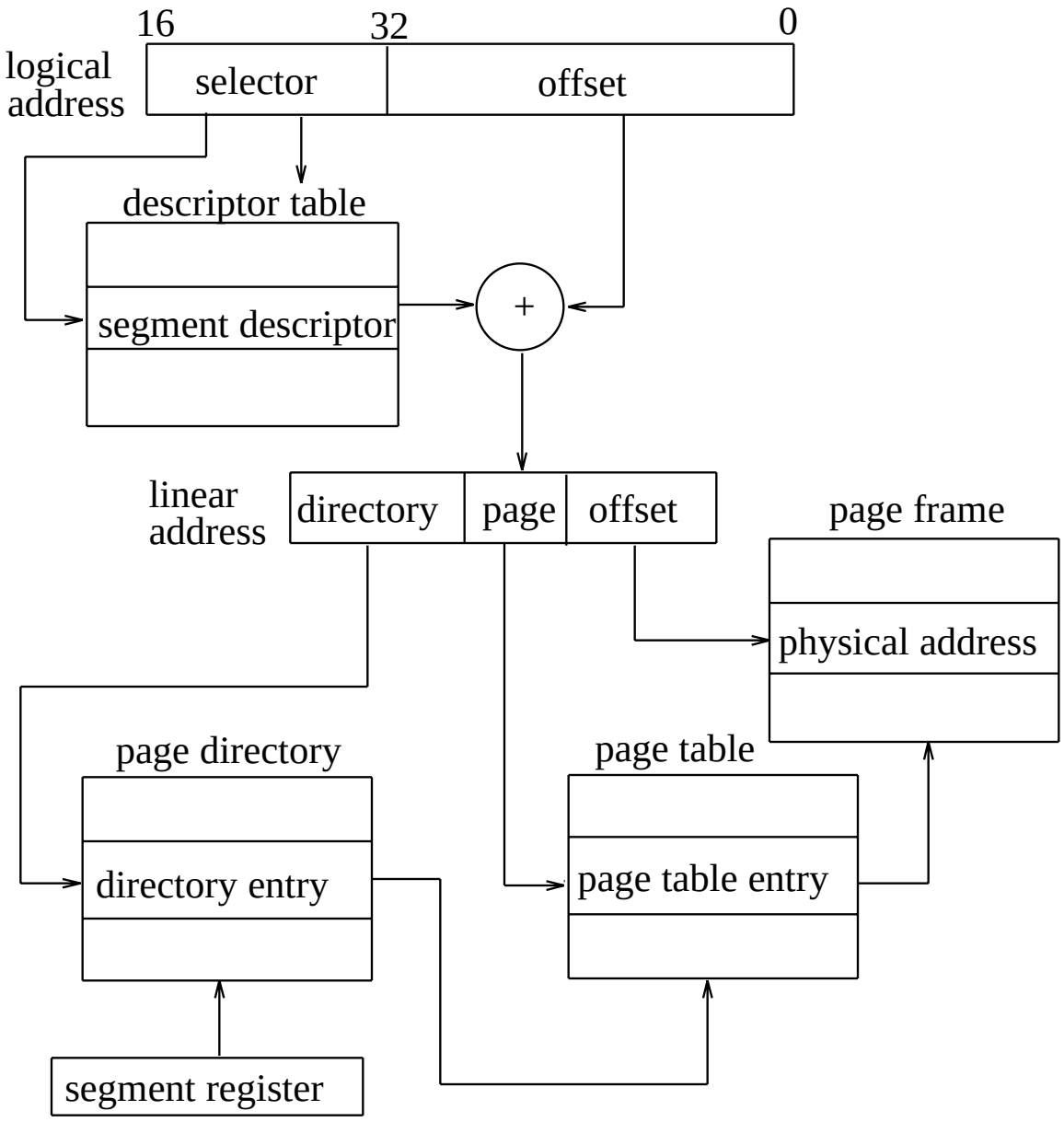
Segmentation with Paging

- The MULTICS system solved problems of external fragmentation and lengthy search times by paging the segments.



Solution differs from pure segmentation in that the segment-table entry contains not the base address of the segment, but rather the base address of a *page table* for this segment.

- The Intel 386 uses segmentation with paging for memory management, with a two-level paging scheme.



Considerations in comparing memory-management strategies:

- Hardware support
- Performance
- Fragmentation
- Relocation
- Swapping
- Sharing
- Protection