Paging and segmentation

Segmentation

- **Segmentation is a memory-management scheme that supports the programmer**
- view of memory.
- A logical address space is a collection of segments.

Segmentation

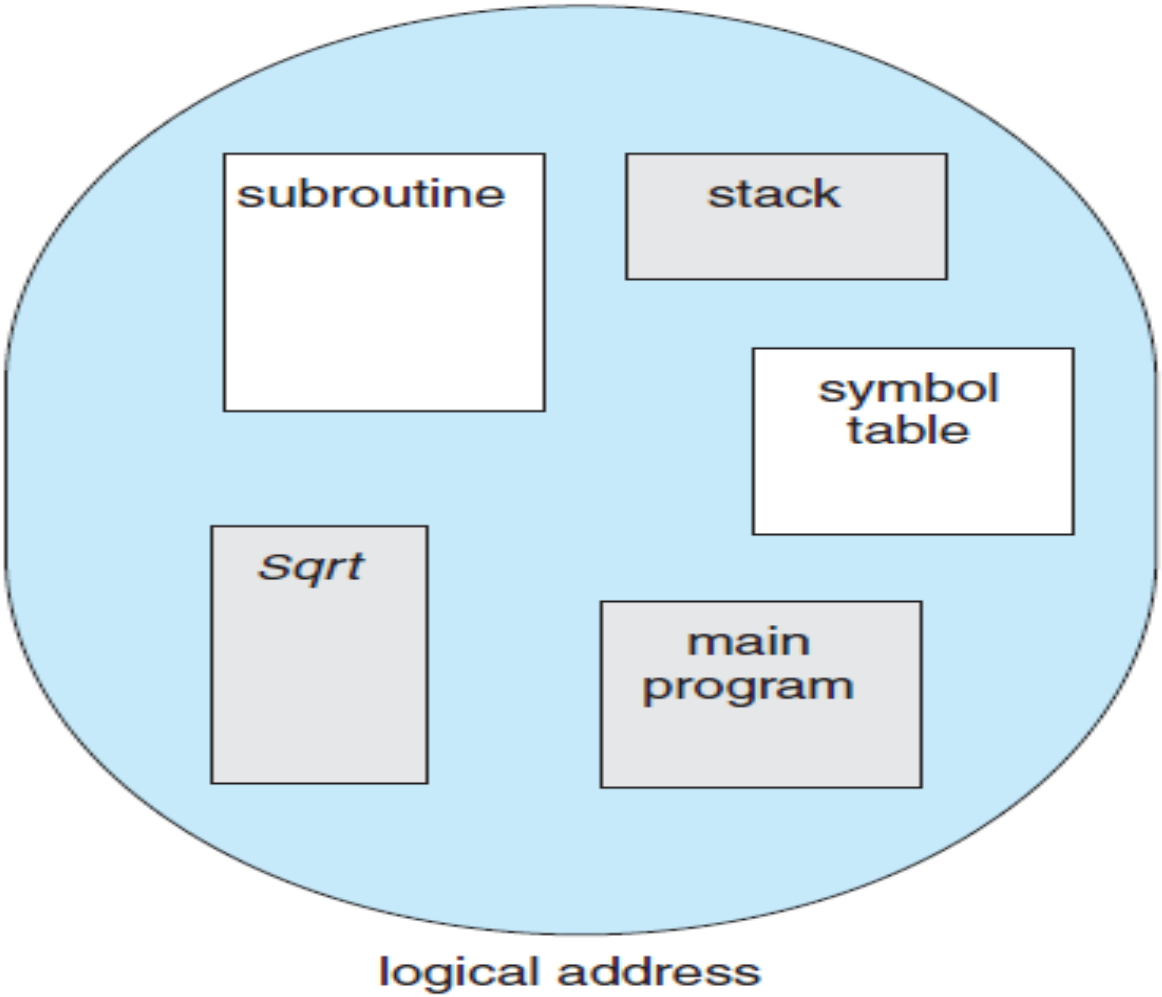


Figure 8.7 Programmer's view of a program.

Segmentation

- Each segment has a name and a length.
- The addresses specify both the segment name and the offset within the segment.
- The programmer therefore specifies each address by two quantities:
- **a segment name and an offset.**
- For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name.
- Thus, a logical address consists of a *two tuple:*
- <segment-number, offset>.

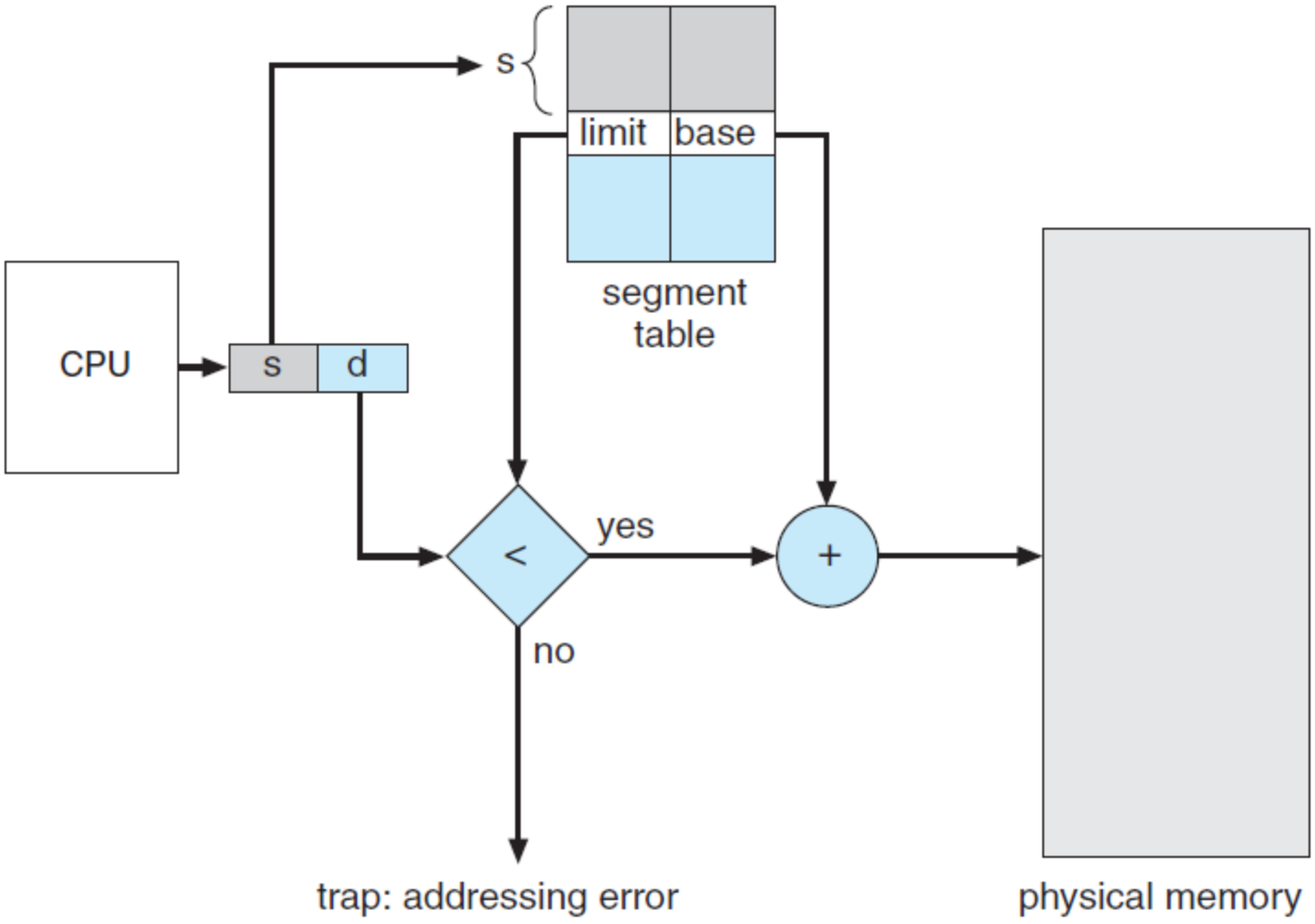
Segmentation

- Normally, when a program is compiled, the compiler automatically constructs segments reflecting the input program.
- A C compiler might create separate segments for the following:
- 1. The code
- 2. Global variables
- 3. The heap, from which memory is allocated
- 4. The stacks used by each thread
- 5. The standard C library
- Libraries that are linked in during compile time might be assigned separate segments.
- The loader would take all these segments and assign them segment numbers.

Segmentation Hardware

- Although the programmer can now refer to objects in the program by a two-dimensional address, the actual physical memory is still, of course, a one dimensional sequence of bytes.
- Thus, we must define an implementation to map two-dimensional user-defined addresses into one-dimensional physical addresses.
- This mapping is effected by a segment table.
- Each entry in the segment table has a segment base and a segment limit.
- The segment base contains the starting physical address where the segment resides in memory, and the segment limit specifies the length of the segment.

Segmentation Hardware



- The use of a segment table is illustrated in Figure .
- A logical address consists of two parts: a segment number, *s, and an offset into that segment, d.*
- The segment number is used as an index to the segment table.
- The offset *d of* the logical address must be between 0 and the segment limit. If it is not,we trap to the operating system (logical addressing attempt beyond end of segment).
- When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte.
- The segment table is thus essentially an array of base–limit register pairs.

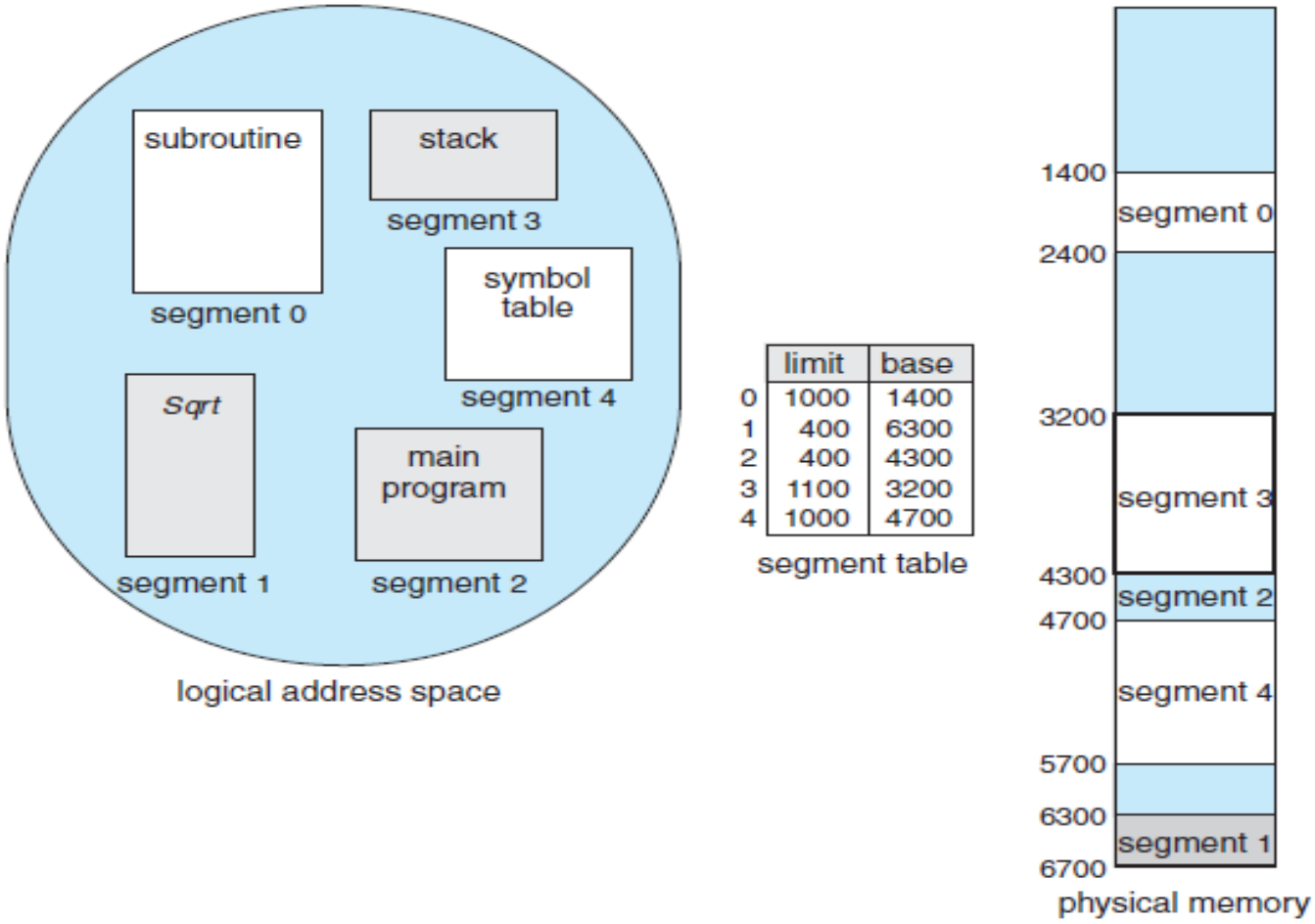


Figure 8.9 Example of segmentation.

- As an example, consider the situation shown in Figure 8.9.
- We have five segments numbered from 0 through 4. The segments are stored in physical memory as shown.
- The segment table has a separate entry for each segment, giving the beginning address of the segment in physical memory (or base) and the length of that segment (or limit)

- For example, segment 2 is 400 bytes long
- and begins at location 4300.
- Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 + 53 = 4353.
- A reference to segment 3, byte 852, is mapped to
- 3200 (the base of segment 3) + 852 = 4052.
- A reference to byte 1222 of segment 0 would result in a trap to the operating system, as this segment is only 1,000 bytes long.

paging

- Paging is a memory-management scheme that permits the physical address space of a process to be noncontiguous.
- Paging avoids external fragmentation and the need for compaction.
- It also solves the considerable problem of fitting memory chunks of varying sizes onto the backing store;
- most memory management schemes used before the introduction of paging suffered from this problem.

paging

- The problem arises because, when some code fragments or data residing in main memory need to be swapped out, space must be found on the backing store.
- The backing store has the same fragmentation problems discussed in connection with main memory, but access is much slower, so compaction is impossible.
- Because of its advantages over earlier methods, paging in its various forms is used in most operating systems

paging

- The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages.
- When a process is to be executed, its pages are loaded into any available memory frames from the backing store.
- The backing store is divided into fixed-sized blocks that are of the same size as the memory frames.
- For example, the logical address space is now totally separate from the physical address space, so a process can have a logical 64-bit address space even though the system has less than 264 bytes of physical memory.

paging

- The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages.
- When a process is to be executed, its pages are loaded into any available memory frames from the backing store.
- The backing store is divided into fixed-sized blocks that are of the same size as the memory frames.
- For example, the logical address space is now totally separate from the physical address space, so a process can have a logical 64-bit address space even though the system has less than 264 bytes of physical memory.

paging

- The hardware support for paging is illustrated in figure below.
- Every address generated by the CPU is divided into two parts: a page number (p) and a page offset (d).
- The page number is used as an index into a page table.
- The page table contains the base address of each page in physical memory.
- This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

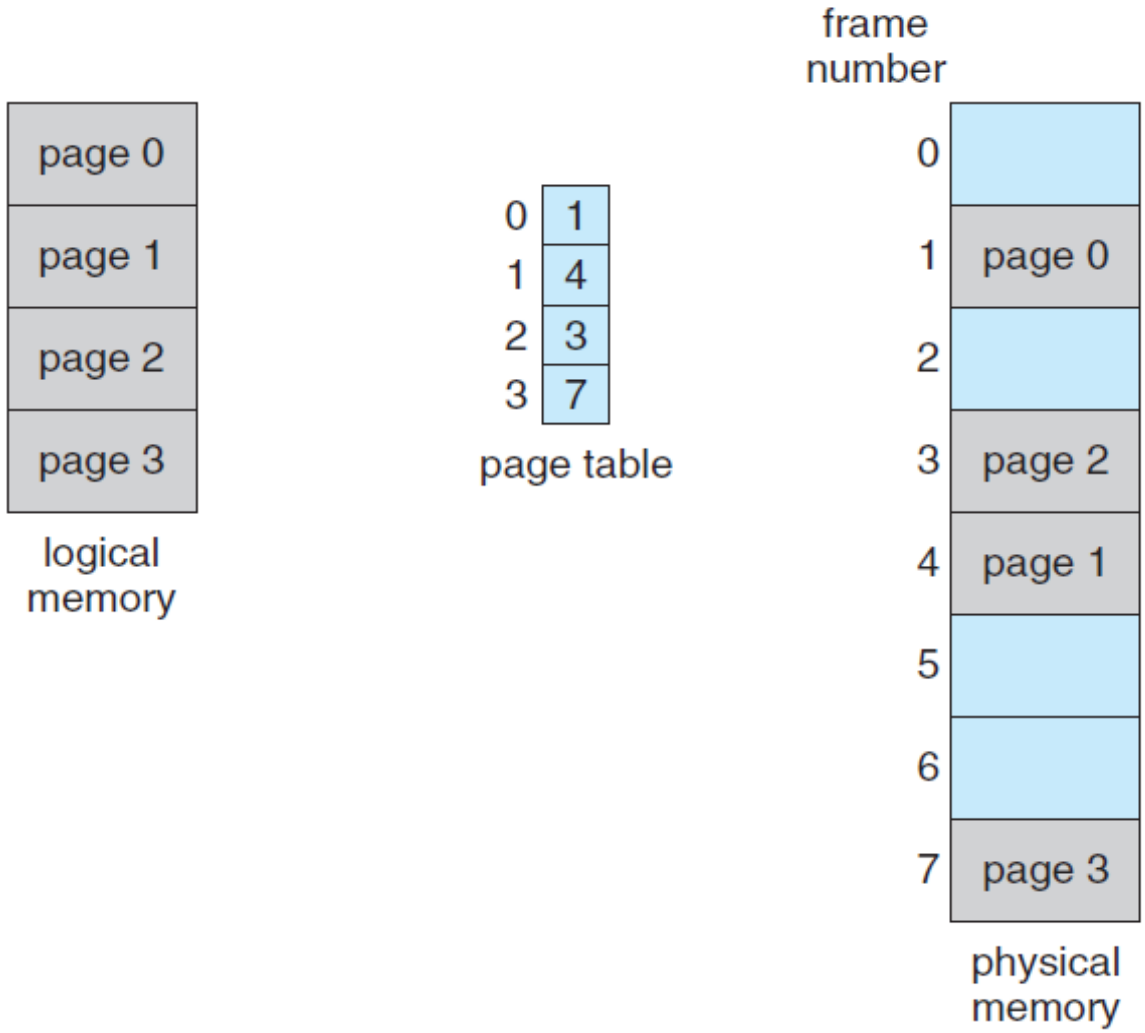
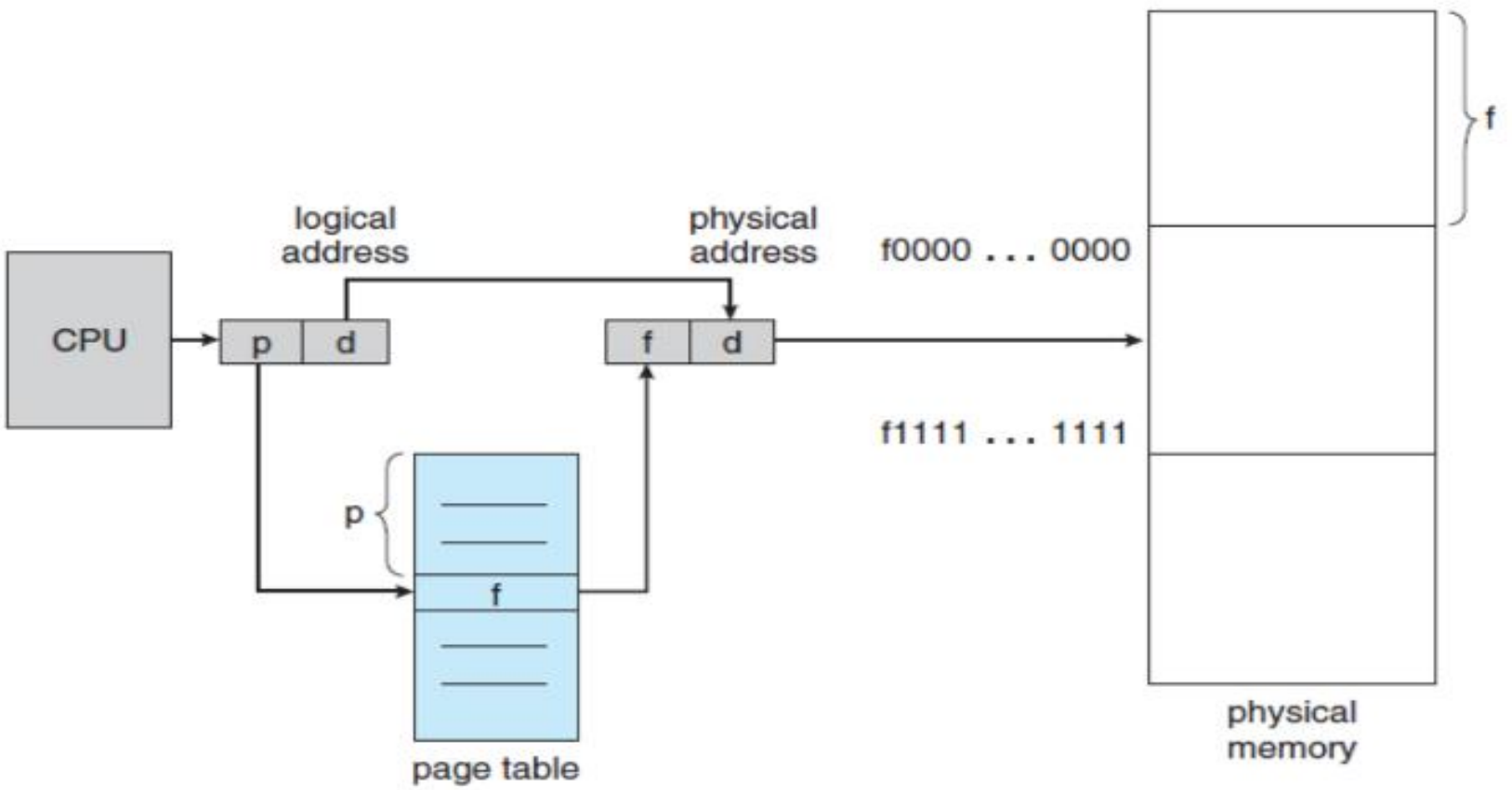


Figure 8.11 Paging model of logical and physical memory.

paging

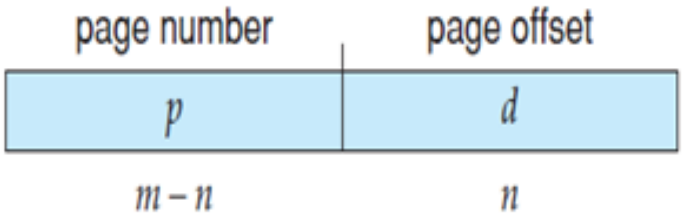
- The paging model of memory is shown in below figure.
- The page size is defined by the hardware.
- The size of a page is typically of a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture.
- The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy.
- If the size of logical address is 2m, and a page size is 2n addressing units, then the high order m-n bits of a logical address designate the page number, and the n low order bits designate the page offset.
- The logical address is as follows

paging



Paging hardware

paging



where p is an index into the page table and d is the displacement within the page.

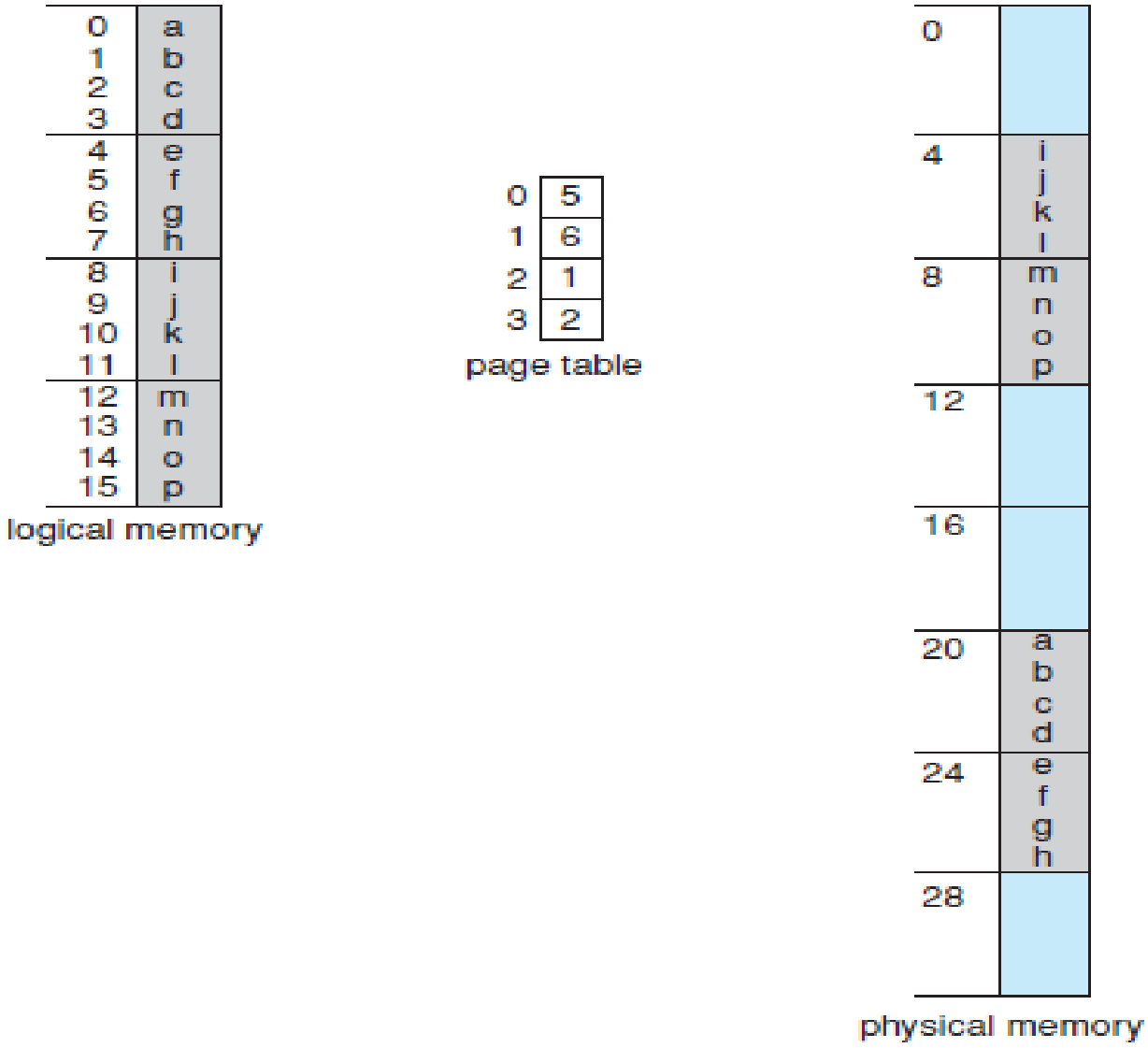
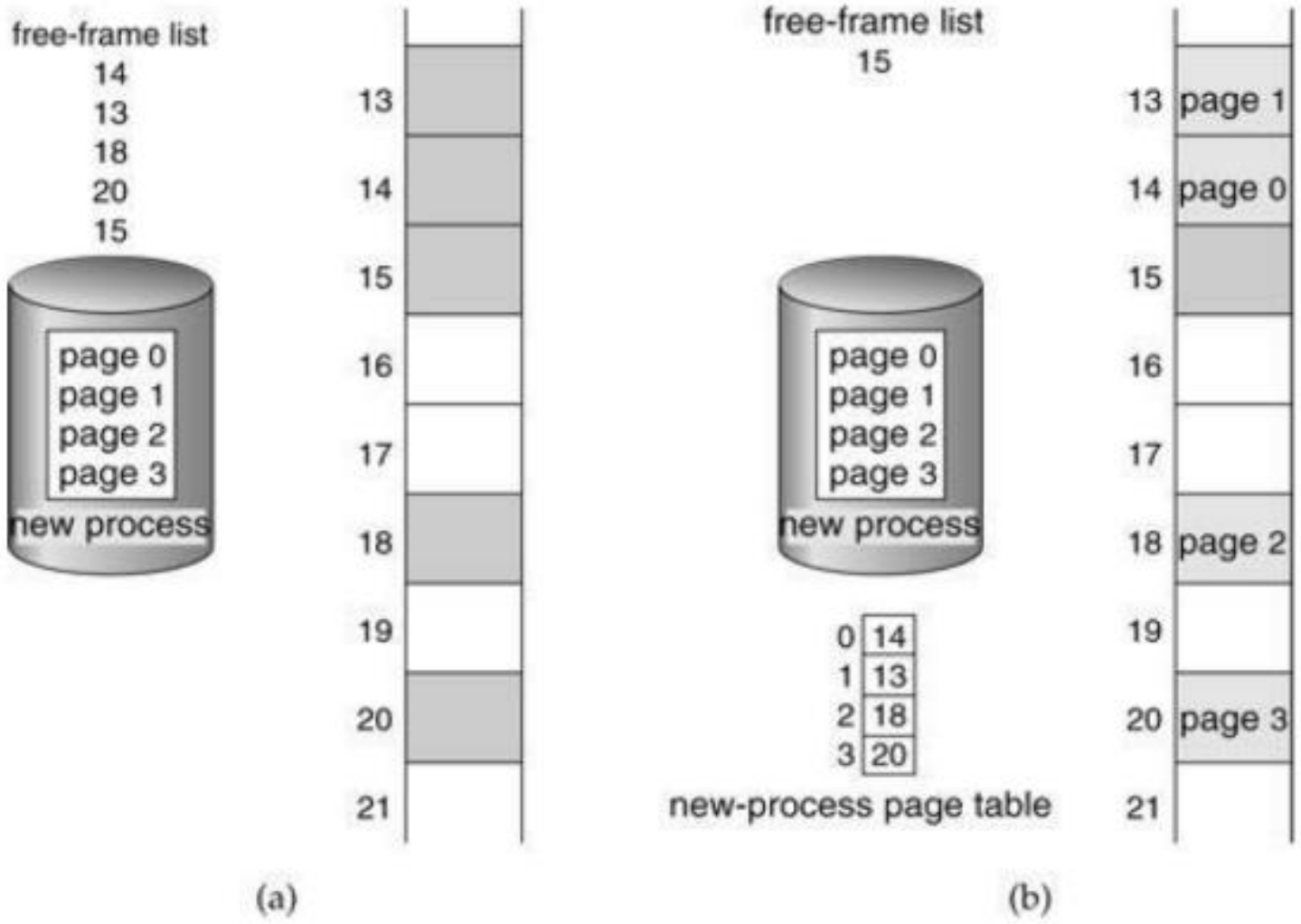


Figure 8.12 Paging example for a 32-byte memory with 4-byte pages.

- Here, in the logical address, n= 2 and m = 4.
- Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages), we show how the programmer's view of memory can be mapped into physical memory.
- Logical address 0 is page 0, offset 0. Indexing into the page table, we find that page 0 is in frame 5. Thus, logical address 0 maps to physical address 20 [= (5 × 4) +0].
- Logical address 3 (page 0, offset 3) maps to physical address 23 [= (5 × 4) +3].

- Physical address will calculated by using the
- formula.
- Physical address = page size of logical memory
  * frame number + offset
- When a process arrives in the system to be executed, its size expressed in pages is examined.
- Each page of the process needs one frame.
- Thus if the process requires n pages, at least n frames must be available in memory. If n frames are available, they are allocated to this arriving process.

- The first page of the process is loaded into one of the allocated frames, and the frame number is put in the page table for this process.
- The next page is loaded into another frame, and its frame number is put into the page table and so on .
- An important aspect of paging is the clear
- separation between the user's view of memory and the actual physical memory.
- The logical addresses are translated into physical addresses.
- This mapping is hidden from the user and is controlled by the operating system



Free frames. (a) Before allocation. (b) After allocation.