COP 4600 – Summer 2013

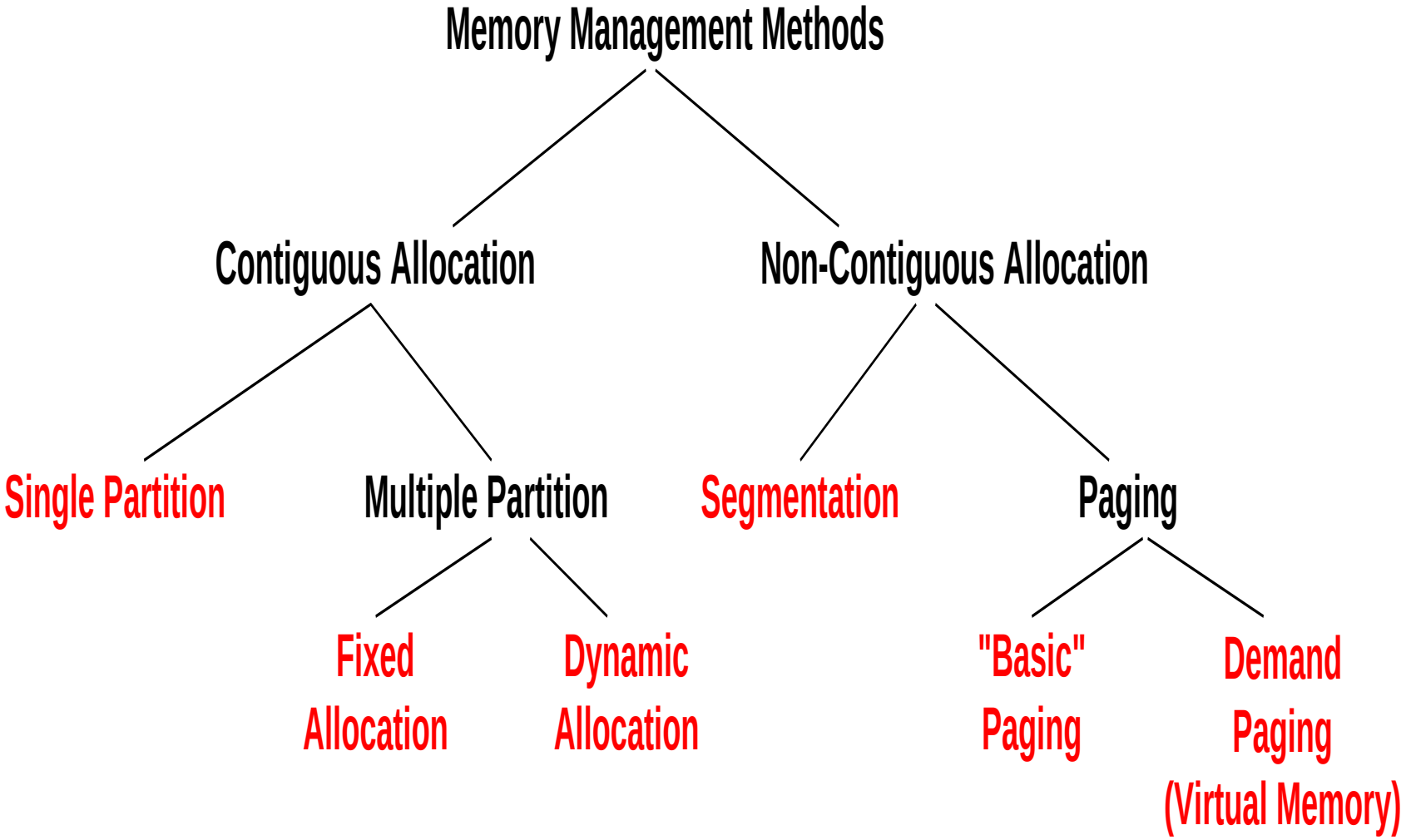
Introduction To Operating Systems

Memory Management – Part 2

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop4600/sum2013

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida

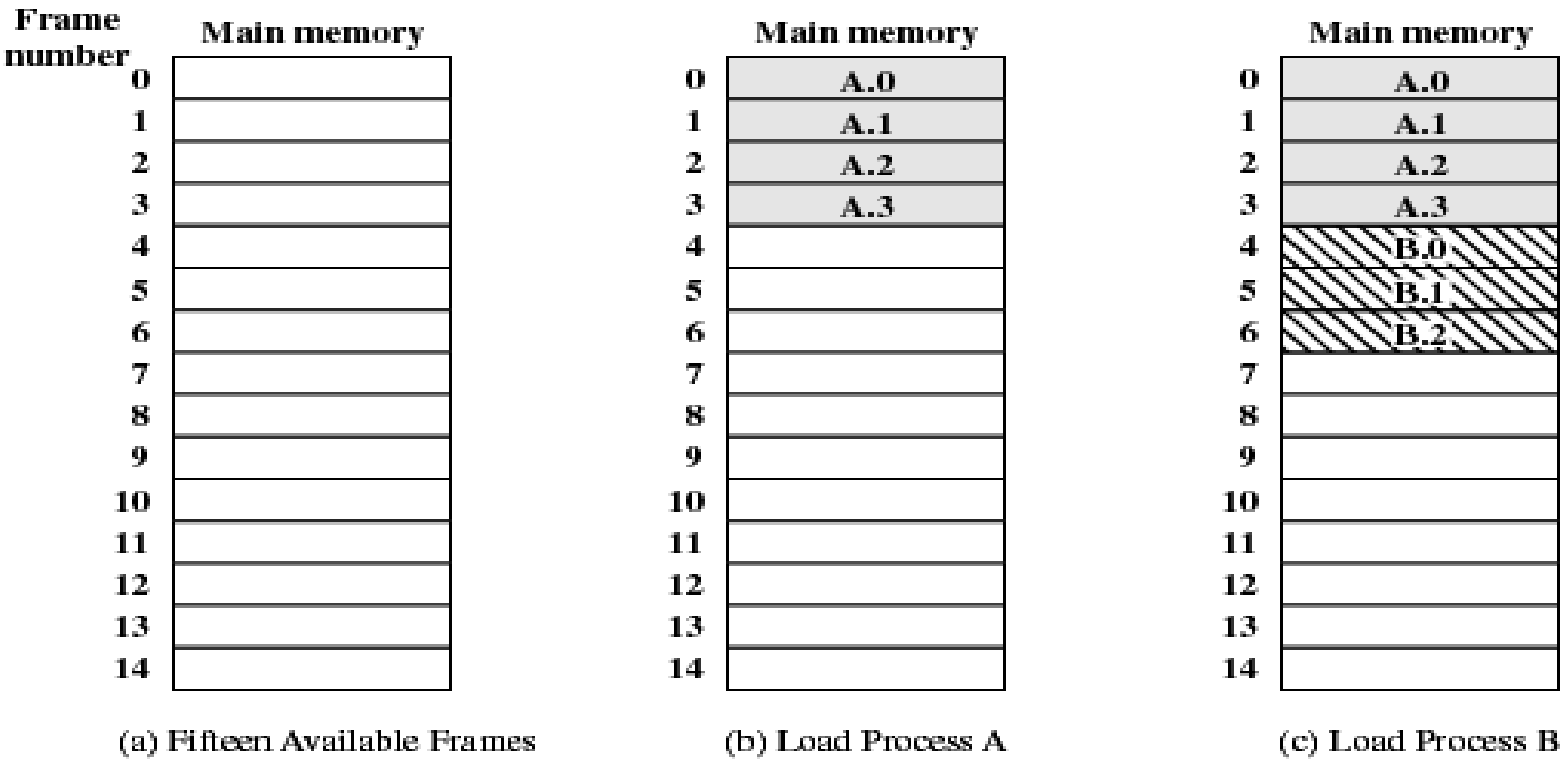
Memory Management



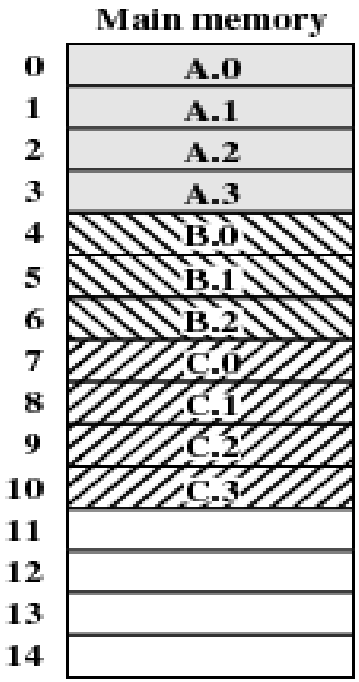
Paging

- Both unequal fixed-size and variable-size partitions are inefficient in the use of memory.
  - Unequal fixed-size partitions result in internal fragmentation.
  - Variable size partitions result in external fragmentation.
- Paging is a technique which attempts to resolve both types of fragmentation.
- In paging, the main memory is partitioned into fixed-size chunks that are relatively small, and each process is also divided into small fixed-size chunks of the same size.
- The chunks of a process are referred to as pages, while the chunks of main memory are referred to as frames (or page frames).
- Paging results in a small amount of internal fragmentation in only the last frame assigned to a process and no external fragmentation.

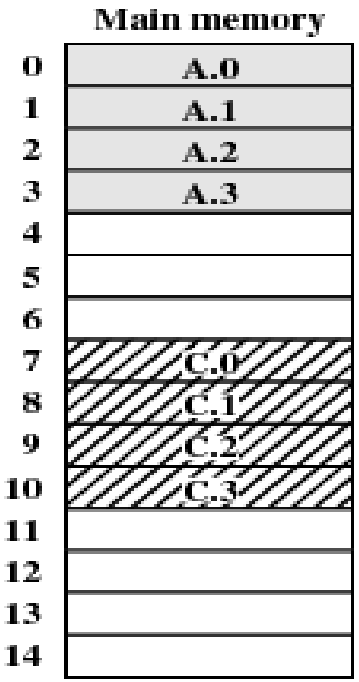
Assignment of Process Pages to Free Frames



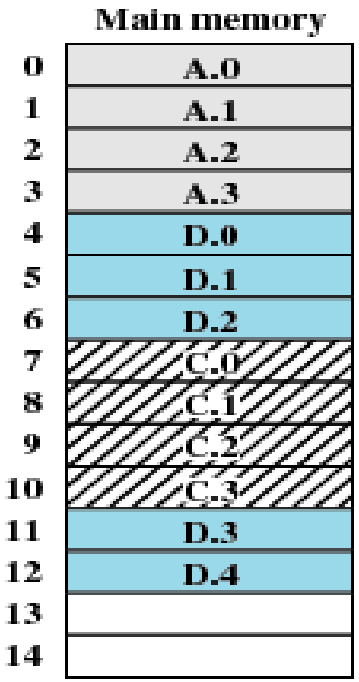
Assignment of Process Pages to Free Frames



(d) Load Process C



(e) Swap out B



(f) Load Process D

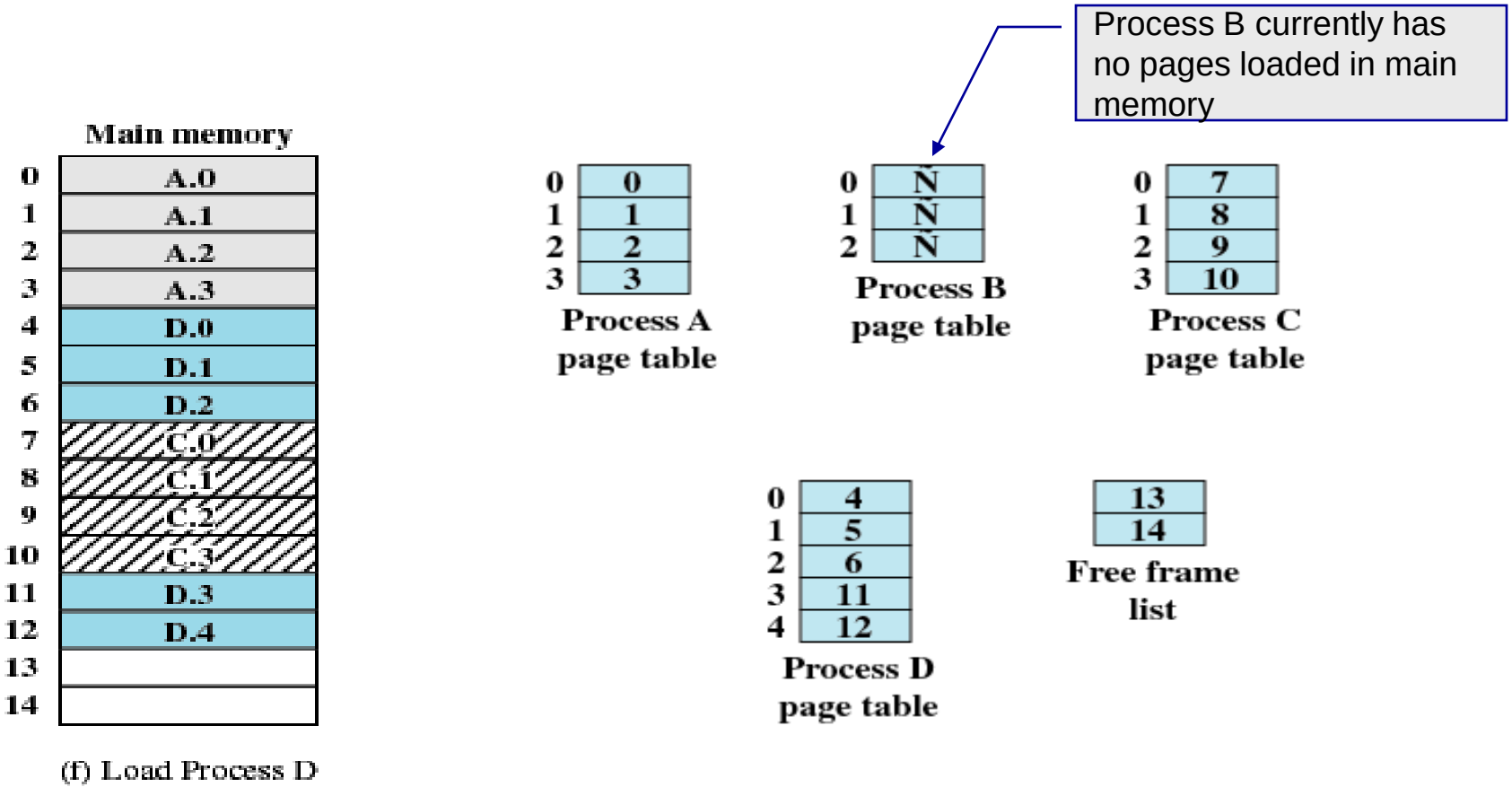
Paging (cont.)

- Notice that process pages do not need to be loaded into contiguous page frames. (Recall our discussions of logical addressing.)
- With paging, a simple relocation register is no longer sufficient for calculating physical addresses at execution time.
- The OS maintains a page table for each process.
- The page table shows the frame location for each page of the process. Within the program, each logical address consists of a page number and an offset within the page.
  - Recall that with simple partitioning, a logical address is the location of the word relative to the beginning of the program which the processor translated into a physical address.
- With paging, the logical-to-physical address translation is still done by processor hardware, but now the processor must know how to access the page table of the current process.

Paging (cont.)

- When presented with a logical address (page number, offset), the processor uses the page table to produce a physical address (page frame, offset).
- The next page, illustrates the page tables for the processes in the previous example at the time illustrated by figure (f), which is shown again for continuity.

Page Tables for Example on Page 4

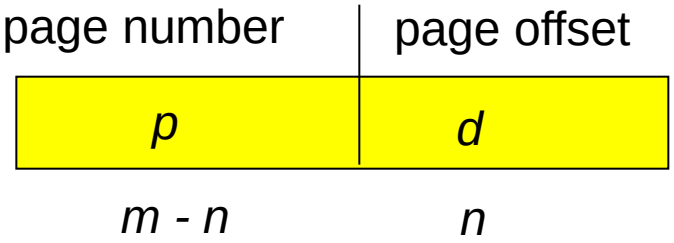


Paging (cont.)

- Simple paging is similar to fixed partitioning. The differences are that, with paging, the partitions are rather small; a program may occupy more than one partition (frame); and these partitions need not be contiguous.
- To make simple paging convenient for use, the page size, and hence the frame size as well, is set to be a power of 2.
- When the page size is a power of 2, it is easy to demonstrate that the relative address, which is defined with reference to the origin of the program, and the logical address, expressed as a page number and offset, are the same!
- The example on the next page illustrates this point.

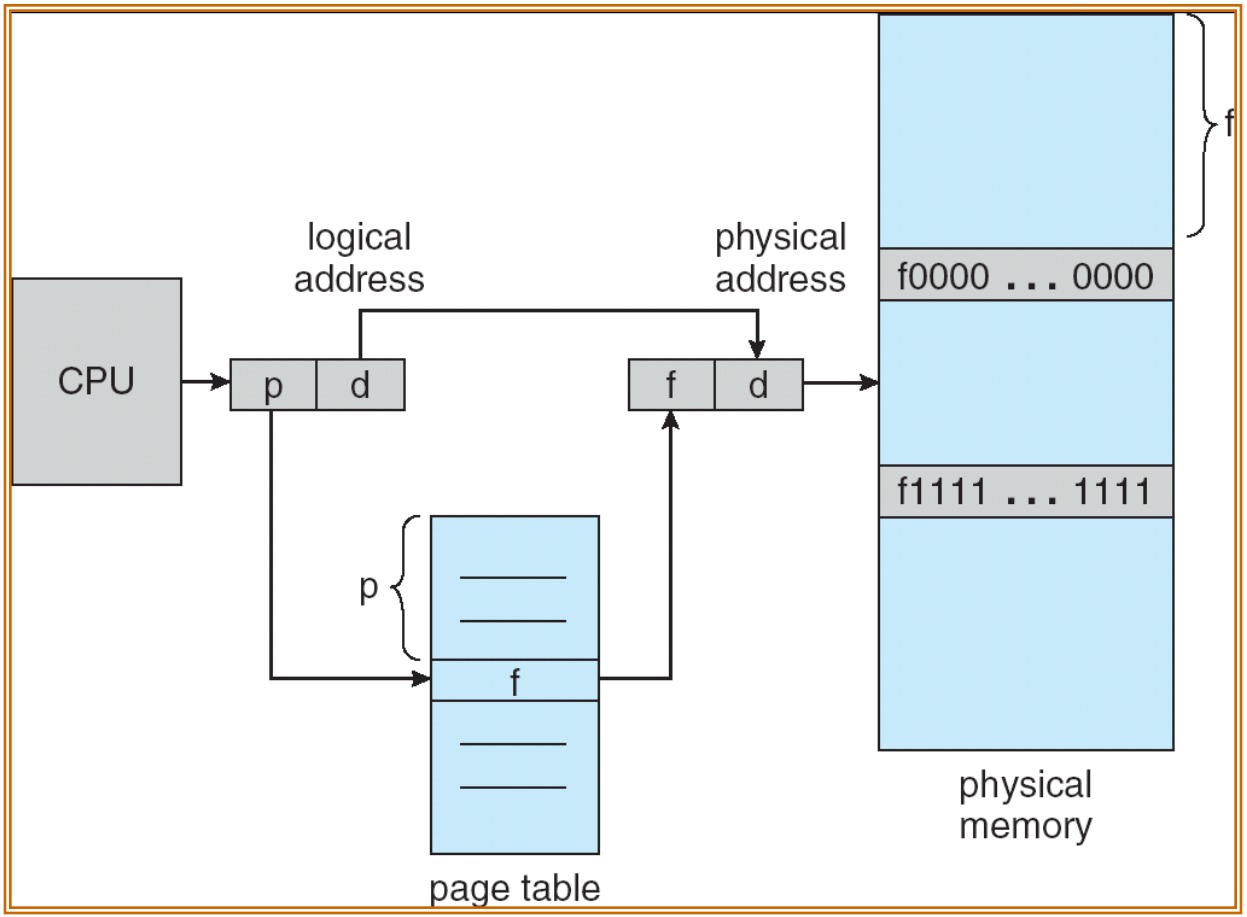
Paging – Address Translation Scheme

- 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

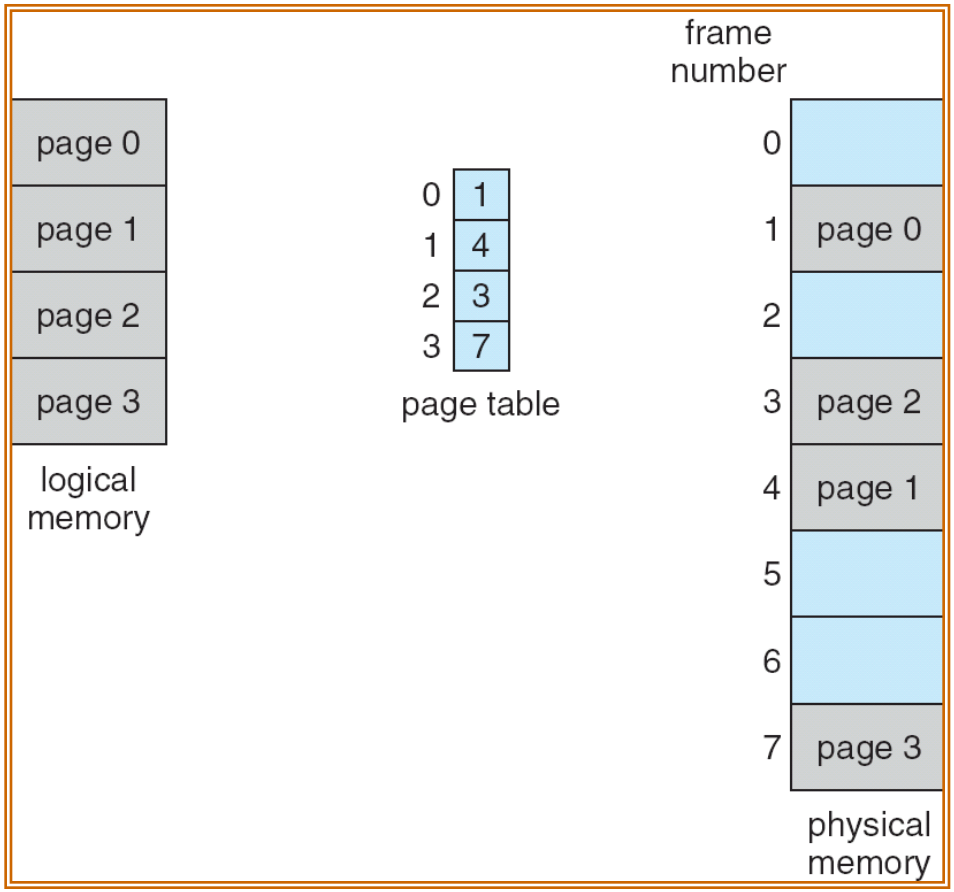


  - For given logical address space 2^m *and page size* 2^n

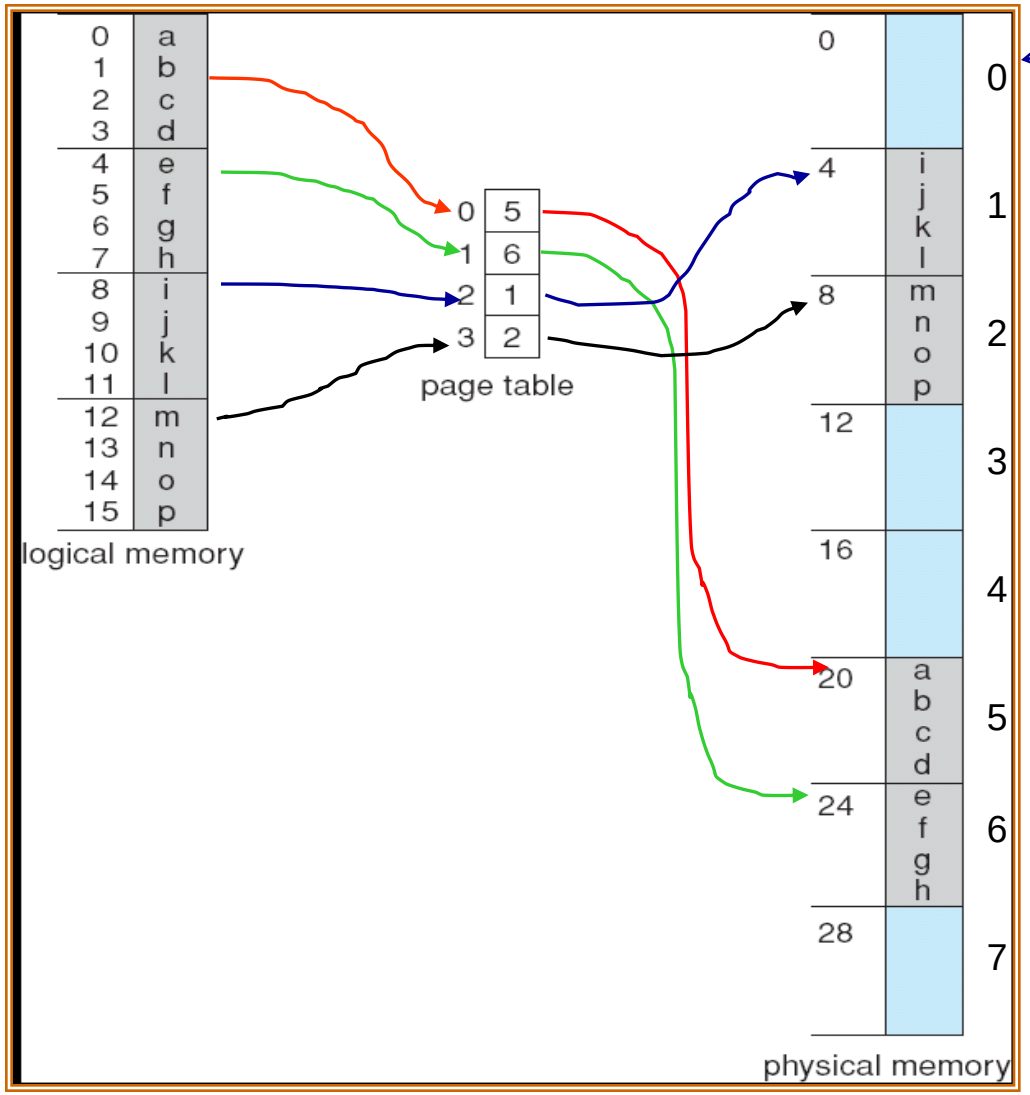
Paging Hardware



Paging Model of Logical and Physical Memory



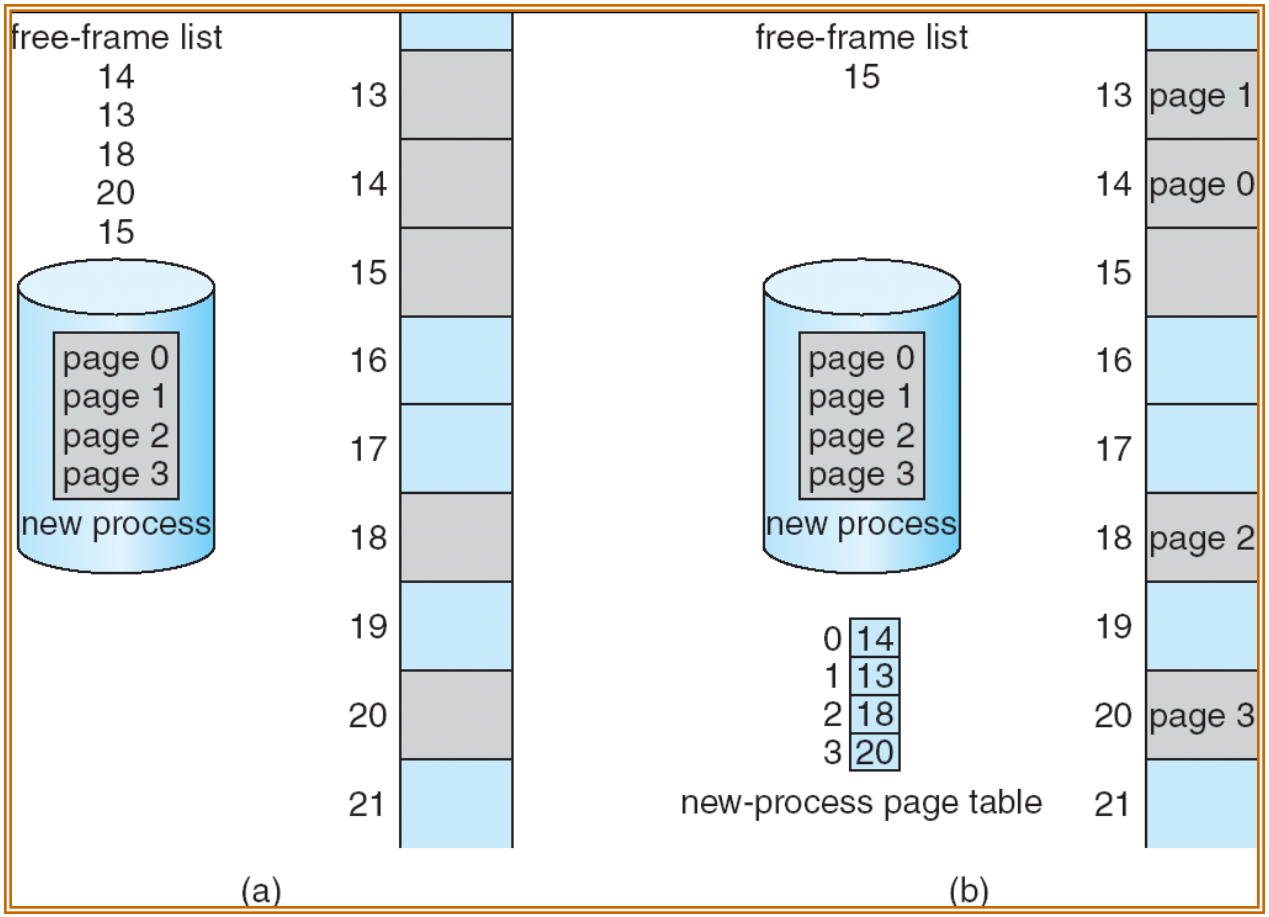
Simple Paging Example



32-byte memory

and 4-byte pages

Free Frames



Before allocation

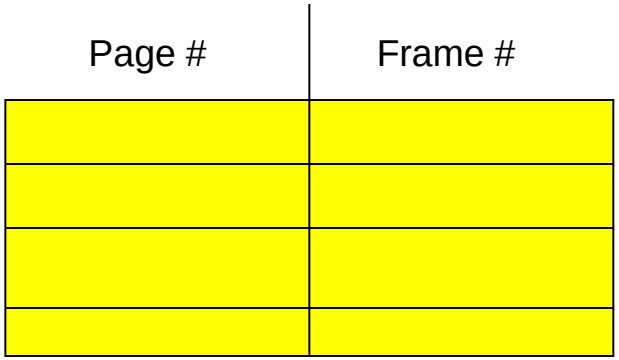
After allocation

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 (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 problem can be solved by the use of a special fast-lookup hardware cache called associative memory or translation look-aside buffers (TLBs).
- Some TLBs store address-space identifiers (ASIDs) in each TLB entry – uniquely identifies each process to provide address-space protection for that process.

Associative Memory

- Associative memory – parallel search

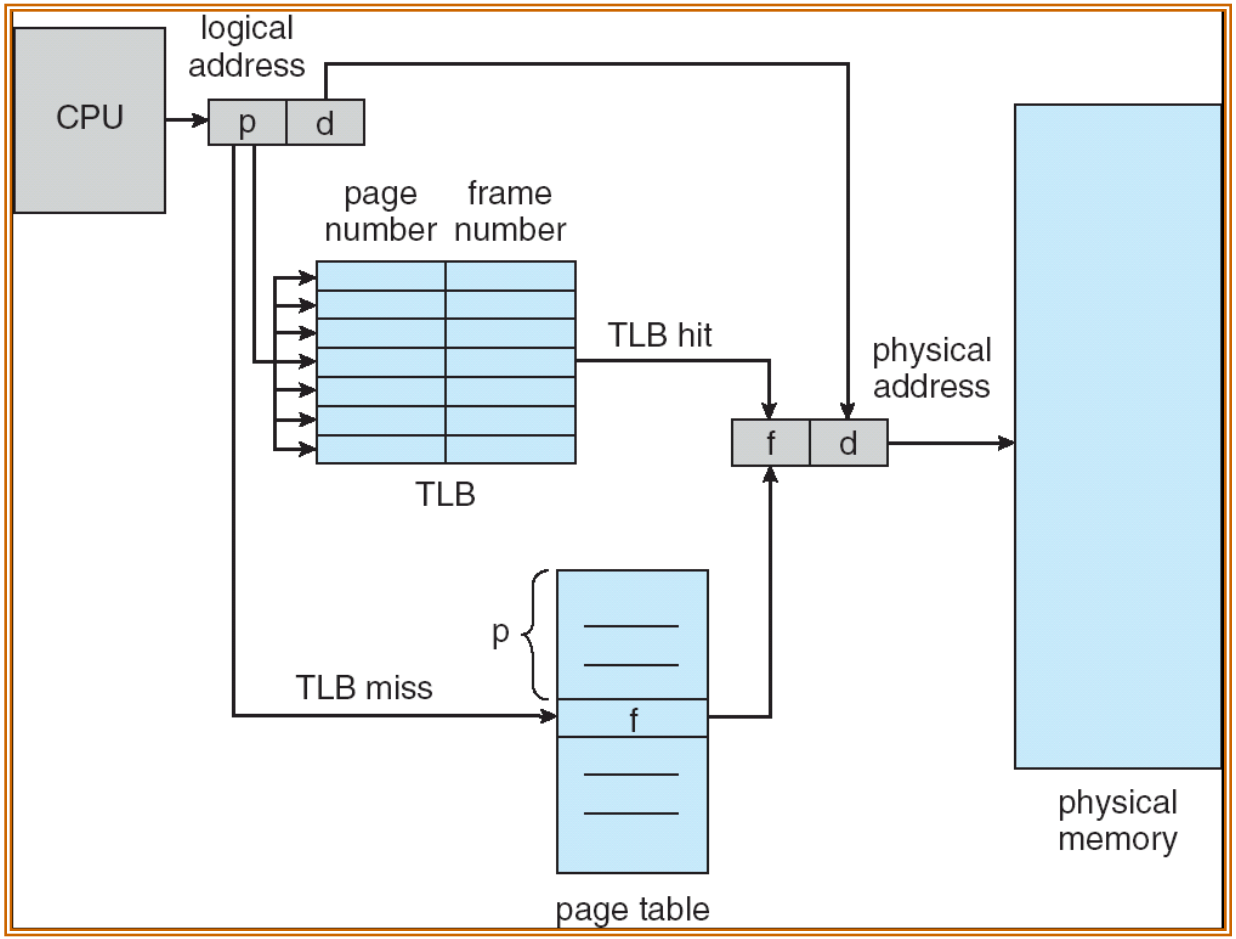


Address translation (`p, d`)

– If `p` is in associative register, get frame # out

– Otherwise get frame # from page table in memory

Paging Hardware With TLB



Effective Access Time

- Associative Lookup = ε time unit.
- Assume memory cycle time is 1 microsecond.
- Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers.
- Hit ratio = α
- Effective Access Time (EAT)

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

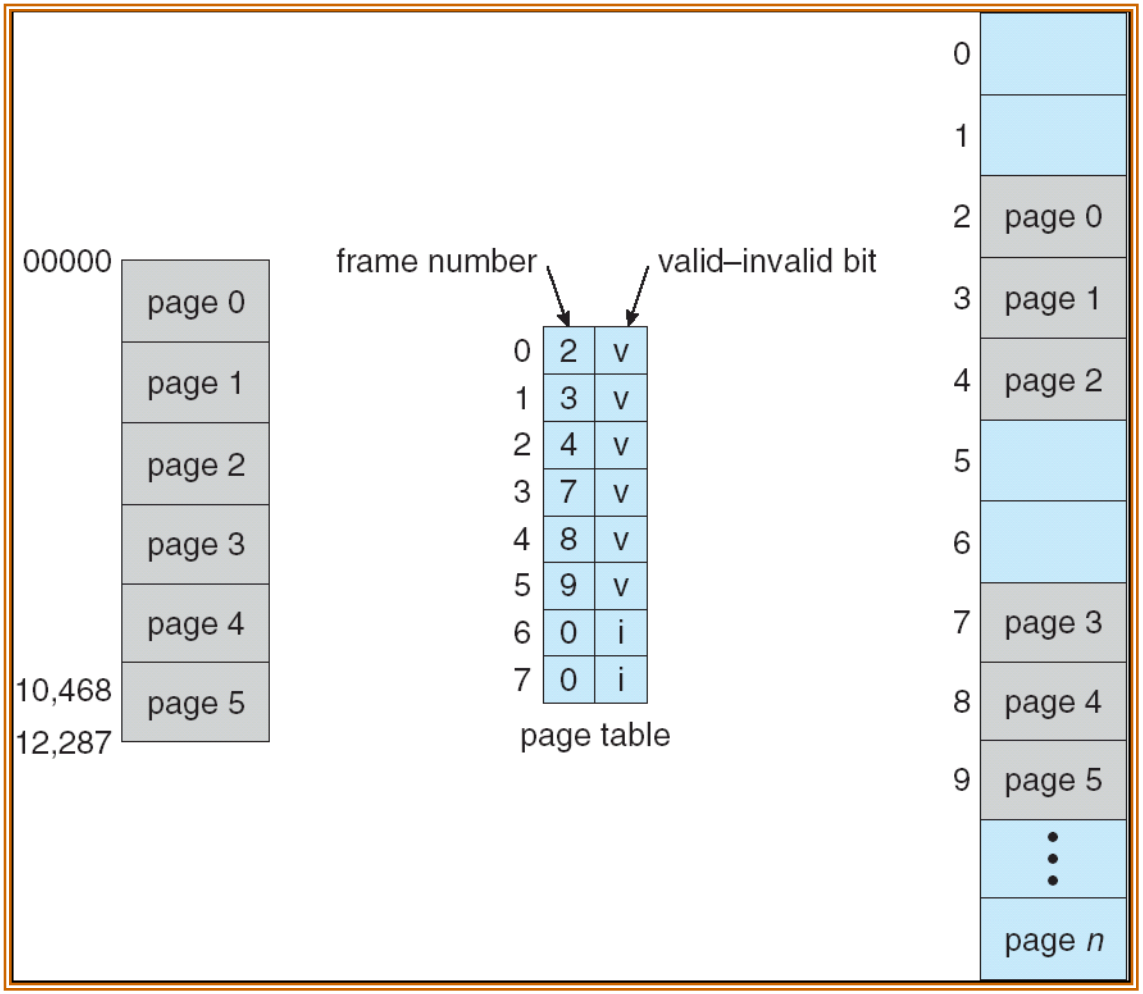
$$= 2 + \varepsilon - \alpha$$

Memory Protection

- Memory protection implemented by associating a 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



Paging Example

- Suppose 16-bit addresses are used in our machine and that the page size is set at 1K = 1024 bytes = 2^{10}.
- The logical address:

$$(1502)_{10} = (5DE)_{16} = (0000010111011110)_2$$

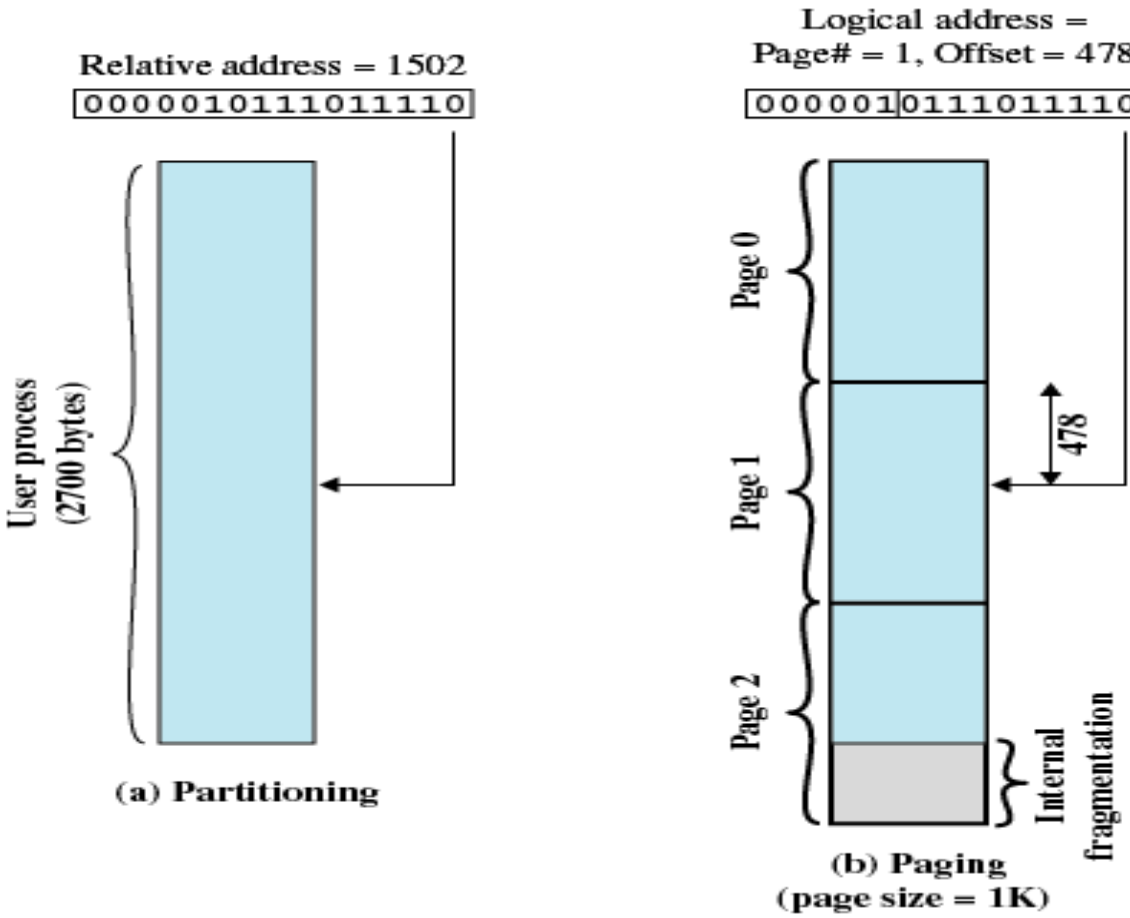
- A page size of 1K, requires an offset field of 10 bits. This leaves 6 bits to represent the page number.
- This implies that a program can consist of a maximum of $2^6 = 64$ pages, each of 1K bytes.

$(0000010111011110)_2$ = page 1, offset $(01110111)_2$ =

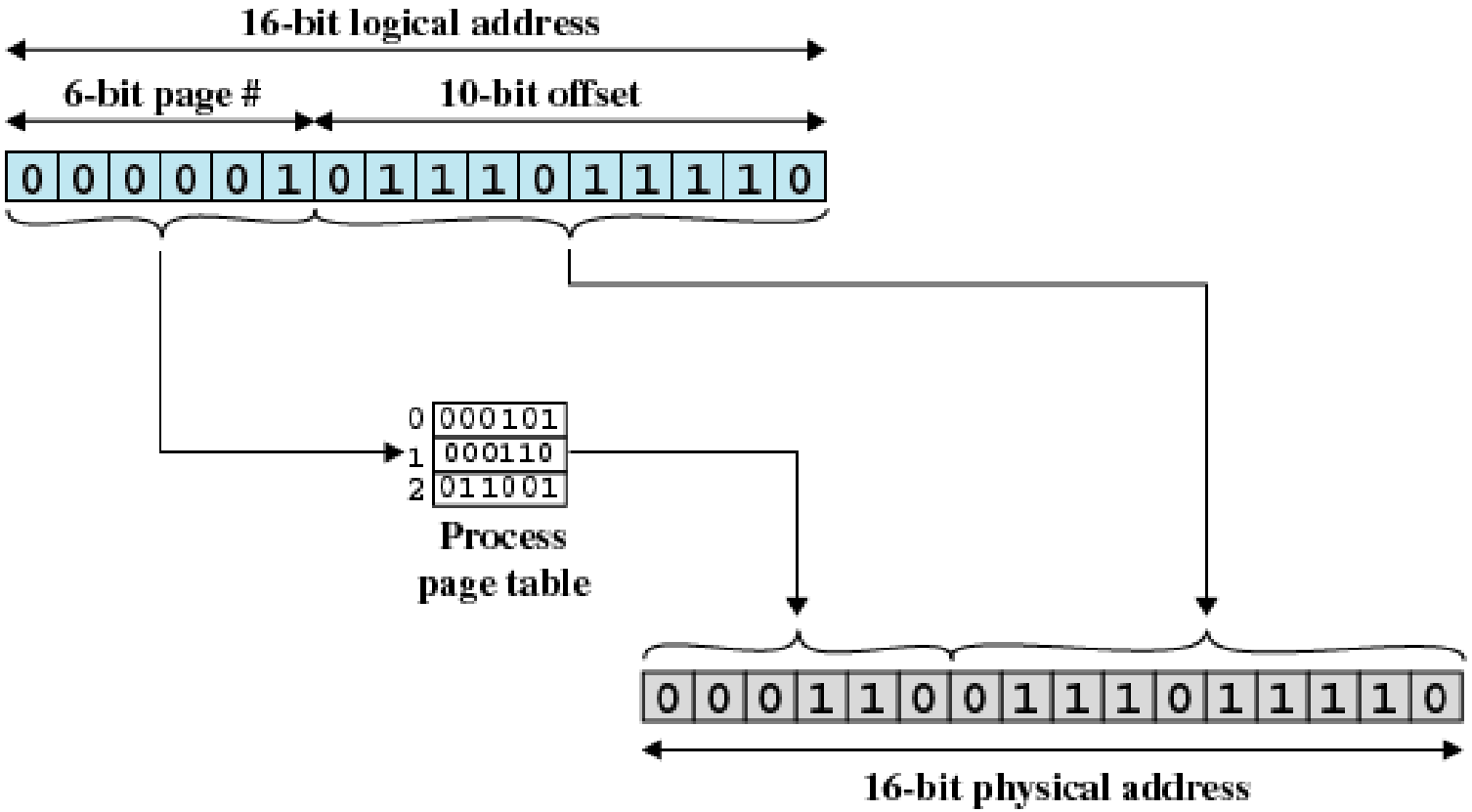
page offset

$(1DE)_{16} = (478)_{10}$

Paging Example (cont.)



Paging Example (cont.)



Paging (cont.)

- The consequences of using a page size that is a power of 2 are twofold.

  1. The logical addressing scheme is transparent to the programmer, the assembler, and the linker. Each logical address (page number, offset) of a program is identical to its relative address.

  2. It is a relatively easy matter to implement a function in hardware to perform dynamic address translation at run time.

- Consider an address of $n + m$ bits, where the leftmost n bits are the page number and the right most m bits are the offset. (In our previous example, $n = 6$, and $m = 10$.)

Paging (cont.)

- The following steps are needed to perform address translation:

  1. Extract the page number as the leftmost *n* bits of the logical address.
  2. Use the page number as an index into the process page table to find the frame number, *k*.
  3. The starting physical address of the frame is $k*2^m$, and the physical address of the referenced byte is that number plus the offset. This physical address does not need to be calculated; it is easily constructed by appending the frame number to the offset.

- Using the previous example, where the logical address $(0000010111011110)_2$ yields page number 1, offset 478. Assuming this page is residing in page frame $(6)_{10} = (000110)_2$, then the physical address is frame number 6, offset 478 = $(0001100111011110)_2$.

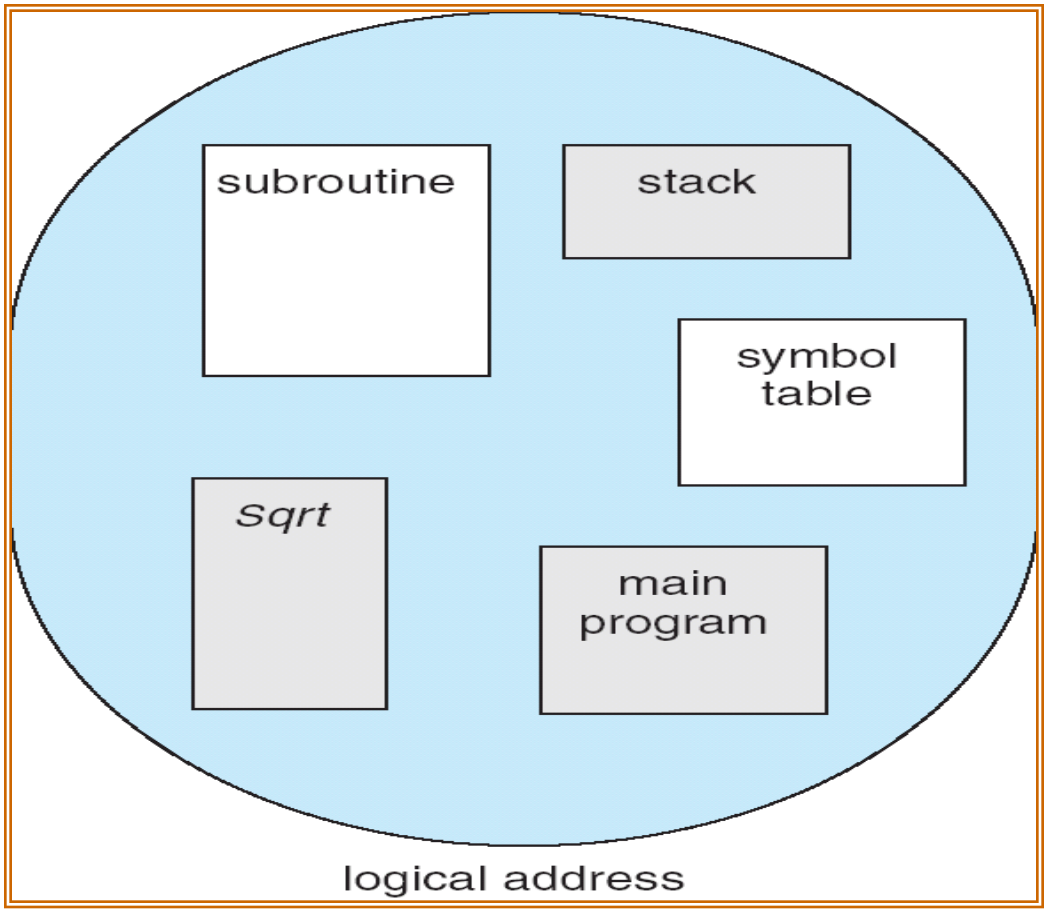
Paging Summary

- With simple paging, main memory is divided into many small equal-size frames. In this respect, paging is similar to fixed-size partitioning.
- Each process is divided into frame-size pages; smaller processes require fewer pages, larger processes require more pages.
- When a process is brought in, all of its pages are loaded into available frames, and a page table is set up for the process.
- Paging results in only a small amount of internal fragmentation. No external fragmentation occurs.

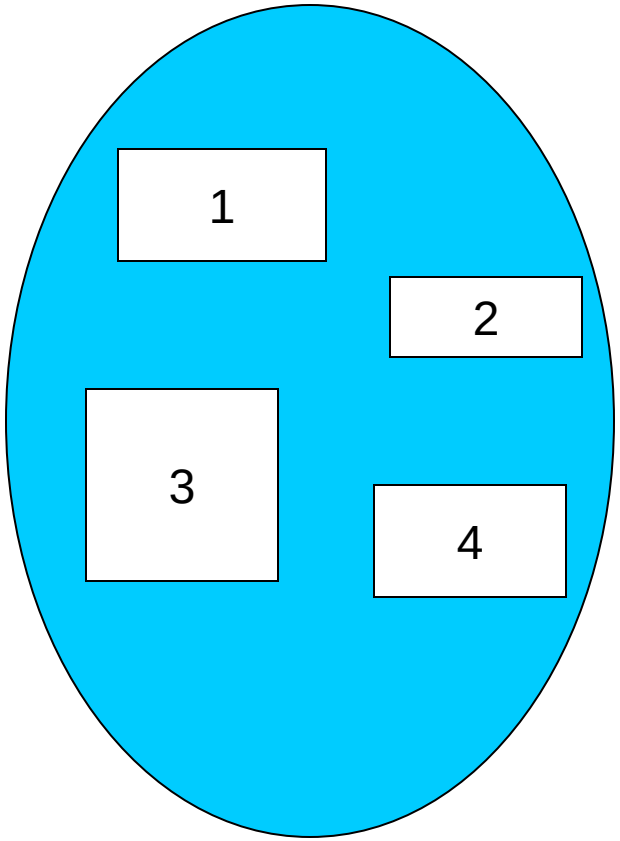
Segmentation

- A user program can be subdivided using segmentation, in which the program and its associated data are divided into a number of segments.
- It is not required that all segments of a program be of the same length, although there is a maximum segment length.
- As with paging, a logical address using segmentation consists of two parts, in this case a segment number and an offset.
- Because of the unequal-size segments, segmentation is similar to dynamic partitioning.
- In the absence of an overlay scheme or the use of virtual memory, it would be required that all of a program's segments be loaded into memory for execution.

User's View of a Program



Logical View of Segmentation



user space

physical memory space

Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

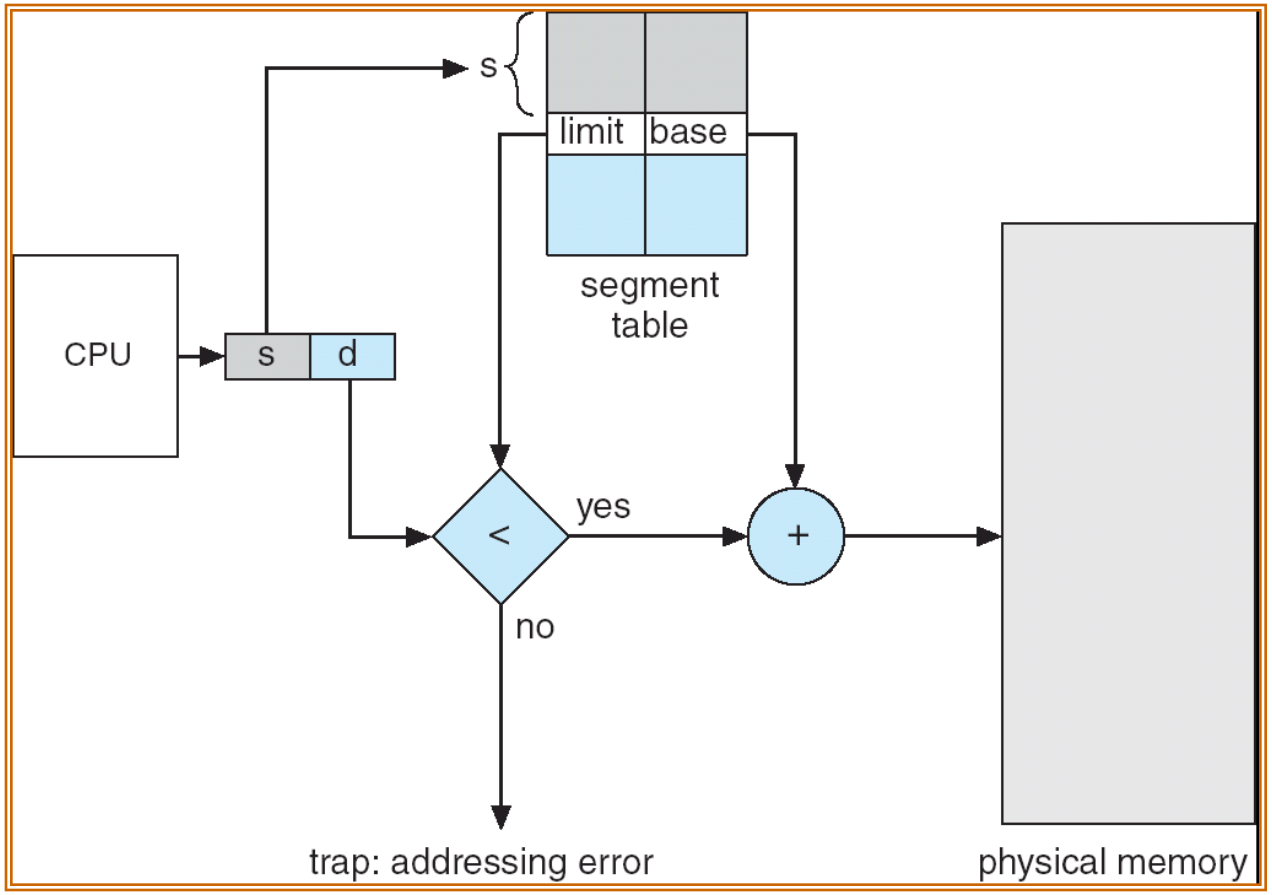
- Segment table – maps two-dimensional physical addresses; each table entry has:
  - base – contains the starting physical address where the segments reside in memory
  - limit – specifies the length of the segment
- Segment-table base register (STBR) points to the segment table's location in memory
- Segment-table length register (STLR) indicates number of segments used by a program;

  segment number ***s*** is legal if ***s*** < **STLR**

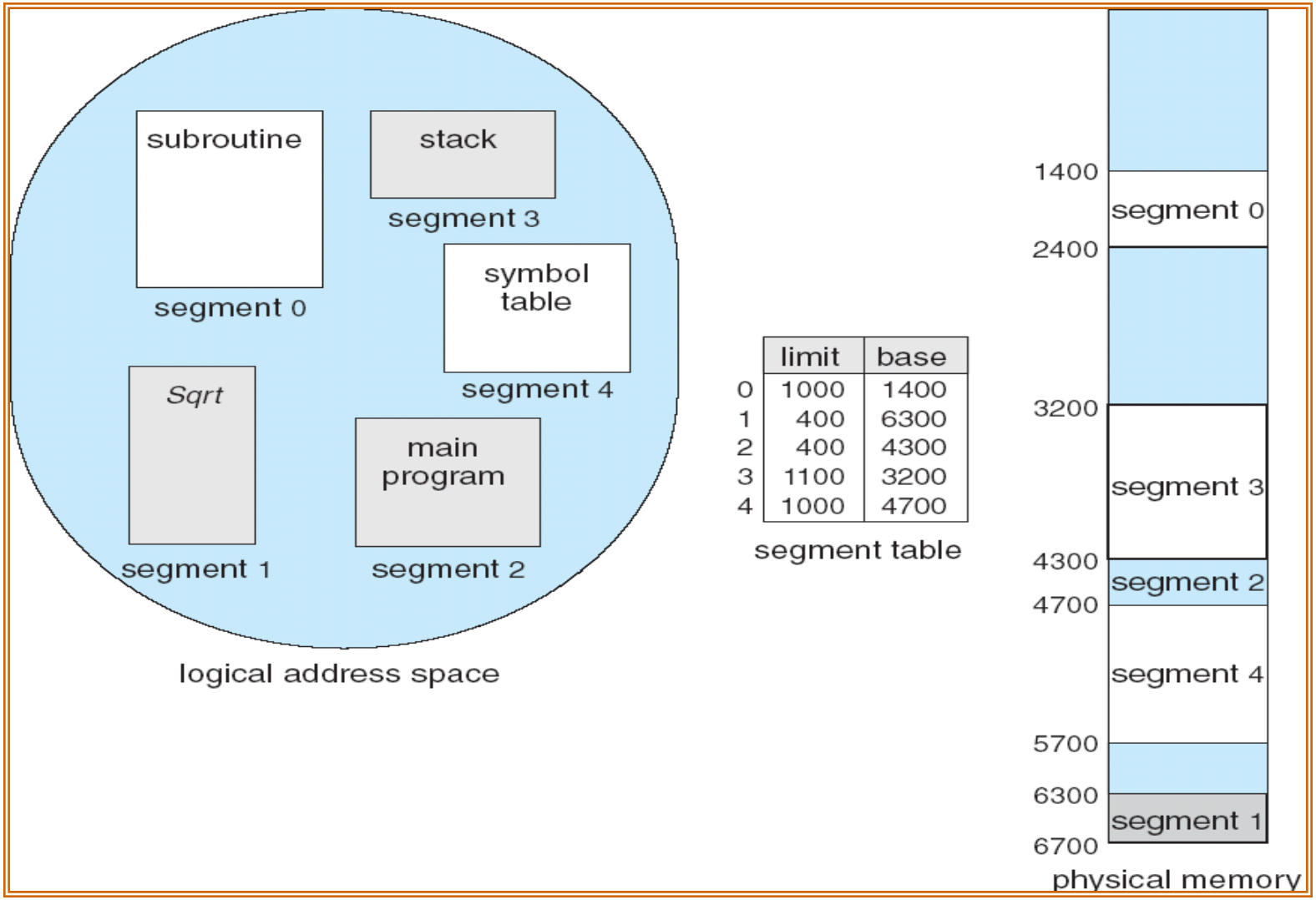
Segmentation Architecture (cont.)

- 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.
- A segmentation example is shown in the following diagram.

Segmentation Hardware



Example of Segmentation



Segmentation (cont.)

- The difference, compared to dynamic partitioning, is that with segmentation a program may occupy more than one partition, and these partitions need not be contiguous.
- Segmentation eliminates internal fragmentation, but like dynamic partitioning, it suffers from external fragmentation. However, since a process is broken up into a number of smaller pieces, the external fragmentation should be less.
- Whereas paging is invisible to the programmer, segmentation is usually visible and is provided as a convenience for organizing programs and data into different segments.
- For purposes of modular programming, the program or data may be further broken down into multiple segments. The principal inconvenience of this service is that the programmer must be aware of the maximum segment size limitation.

Segmentation (cont.)

- Another consequence of unequal-size segments is that there is no simple relationship between logical addresses and physical addresses.
- Analogous to paging, a simple segmentation scheme would make use of a segment table for each process and a list of free blocks of main memory.
- Each segment table entry would have to list the starting address in main memory of the corresponding segment. The entry would also need to provide the length of the segment, to assure than invalid addresses are not used.

Segmentation (cont.)

- When a process enters the running state, the address of its segment table is loaded into a special register used by the memory-management hardware.
- Assume an address of $n+m$ bits where the leftmost n bits are the segment number and the rightmost m bits are the offset.
  - In the example on page 38, $n = 4$ and $m = 12$. Thus, the maximum segment size would be $2^{12} = 4096$ bytes.

Segmentation (cont.)

- Address translation using segmentation proceeds as follows:

1. Extract the segment number as the leftmost *n* bits of the logical address.
2. Use the segment number as an index into the process segment table to find the starting physical address of the segment.
3. Compare the offset, expressed in the rightmost *m* bits, to the length of the segment. If the offset is greater than or equal to the length, the address is invalid.
4. The desired physical address is the sum of the starting physical address of the segment plus the offset.

Segmentation Example

- Suppose 16-bit addresses are used in our machine and that $n = 4$ and $m = 12$, thus the maximum segment size is $2^{12} = 4096$ bytes.
- The program is placed into two segments, where segment #0 = 750 bytes and segment #1 = 1950 bytes.
- The logical address $(0001001011110000)_2$ yields a segment number of 1 and an offset of $(001011110000)_2 = (2F0)_{16} = (752)_{10}$

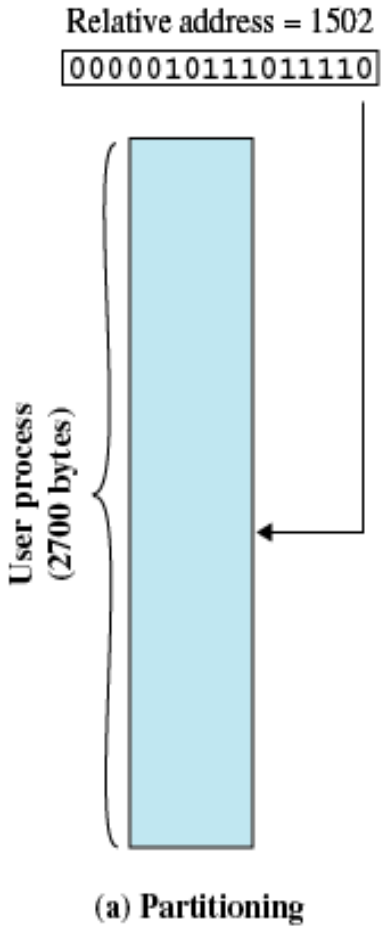
$(0001001011110000)_2$ = segment 1, offset

page offset

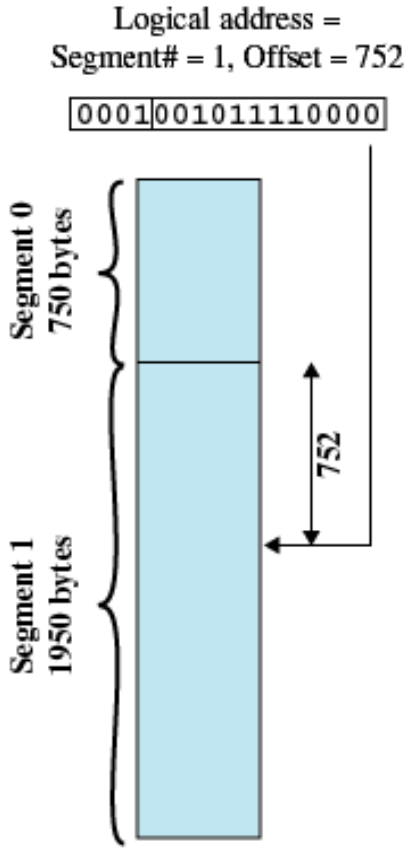
$= (001011110000)_2$

$= (2F0)_{16} = (752)_{10}$

Segmentation Example (cont.)

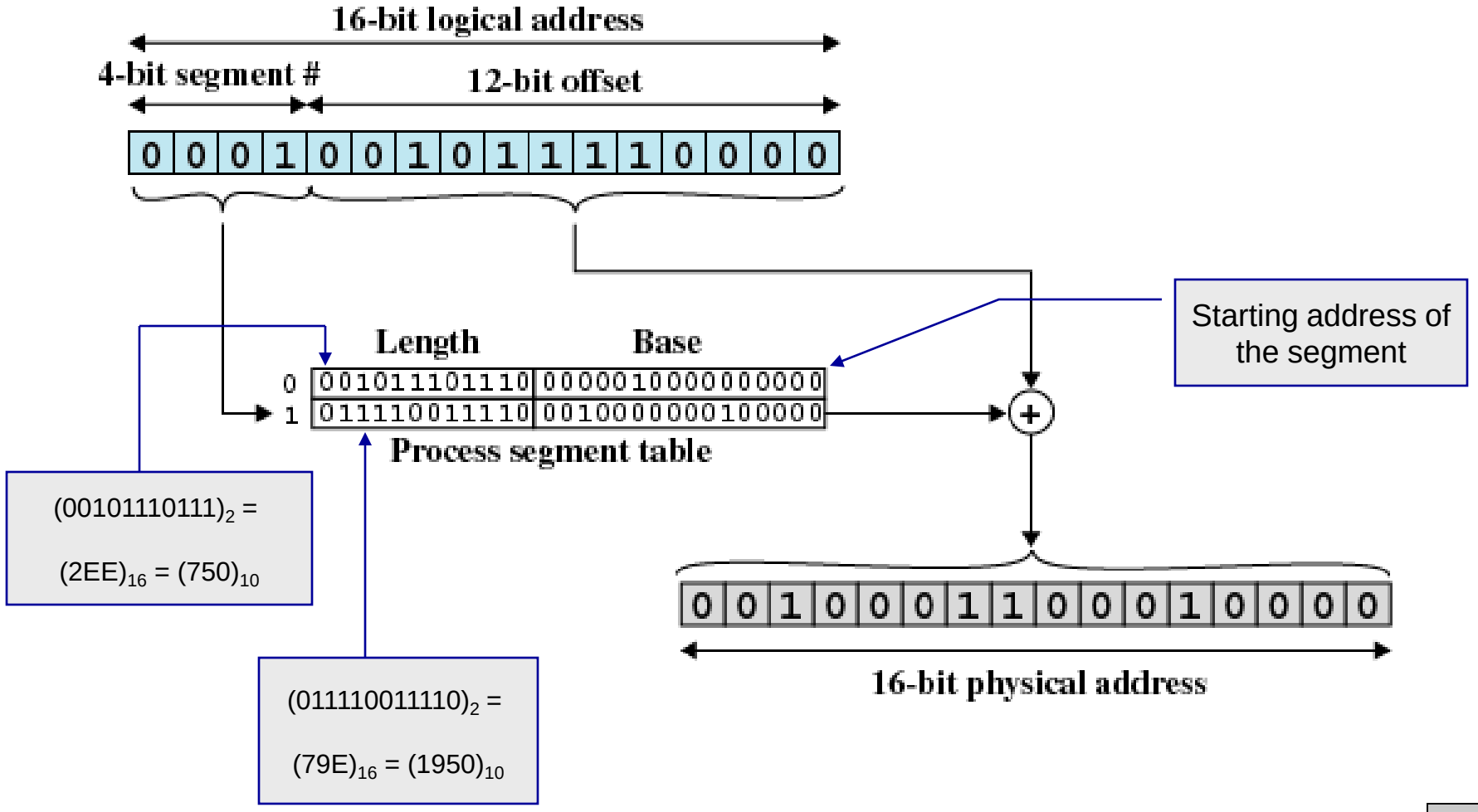


(a) Partitioning



(b) Segmentation

Segmentation Example (cont.)



Segmentation Summary

- With simple segmentation, a process is divided into a number of segments that need not be of equal size.
- When a process is brought in, all of its segments are loaded into available regions of memory, and a segment table is set up for the process.
- Segmentation results in no internal fragmentation. External fragmentation occurs, however, its effects should be less severe than occurs with dynamic partitioning as the segment size is typically smaller.