Chapter 7

Memory Management

Memory Management Requirements

- Relocation
- Protection
- Sharing
- Logical Organization
- Physical Organization

Relocation

- In a multiprogramming system with many programs running at once, a program may be loaded into any free area of memory.
- Furthermore, a program may be swapped out to disk, then later swapped back into memory again.
- It would be difficult to load it back into the same memory locations that it was loaded to before.
- Therefore, since we don't know in advance where the program will load, and since we may swap the program in and out of memory to different areas, we need the ability to relocate the program to any area of memory.

Protection

- Processes should be protected from each other.
- This means the memory references each process makes must be checked to ensure they are allowed.
- But memory references can be computed dynamically by a program (such as array positions) and therefore can't be known in advance.
- Therefore, memory references must be checked at run time by the hardware rather than by OS.
- If you consider that each program instruction must be checked, it would be almost impossible for the OS to do this for the program.

Sharing

- Two or more processes may be running the same program, so we may want to let them share the program code.
- Two or more processes may be working on the same data, so we may want to let them share some memory for data.
- Thus the protection mechanism must also allow flexibility for sharing memory.

Logical Organization

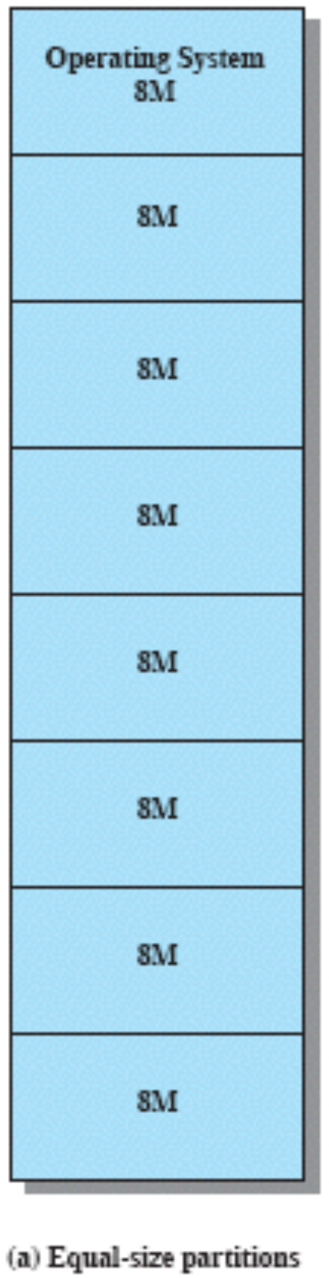
- Logically, memory is organized as a linear address space, consisting of a sequence of bytes or words.
- Programs, however, typically consist of a number of pieces, called modules, which may have different attributes, such as read only.
- If the memory management system effectively deals with modules, a number of advantages can be realized:
  - Modules can be written and compiled independently
  - Modules can be given different degrees of protection (read only, execute only, etc.)
  - Modules may be shared among processes.
- One approach we will see later that helps support the use of modules is memory segmentation.

Physical Organization

- Physically, memory is organized as main memory and secondary storage (disk).
- It is difficult for the programmer to manage moving the program between these two environments as it runs.
- Therefore, it is best to let the memory management system manage the moving of processes between these two types of memory.

Memory Partitioning

- An easy way to deal with memory is to partition it into regions with fixed boundaries.
- Each region can be equally sized, or each region can have different sizes.
- Partitioning is an old technique that isn’t used today but is important to our understanding of modern memory management techniques.



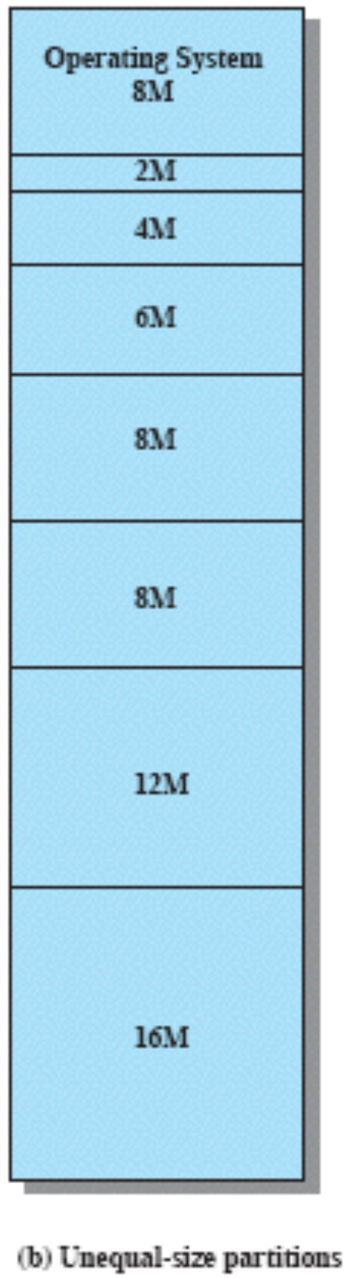


Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

Partitioning: Equal Sized Partitions

Two problems:

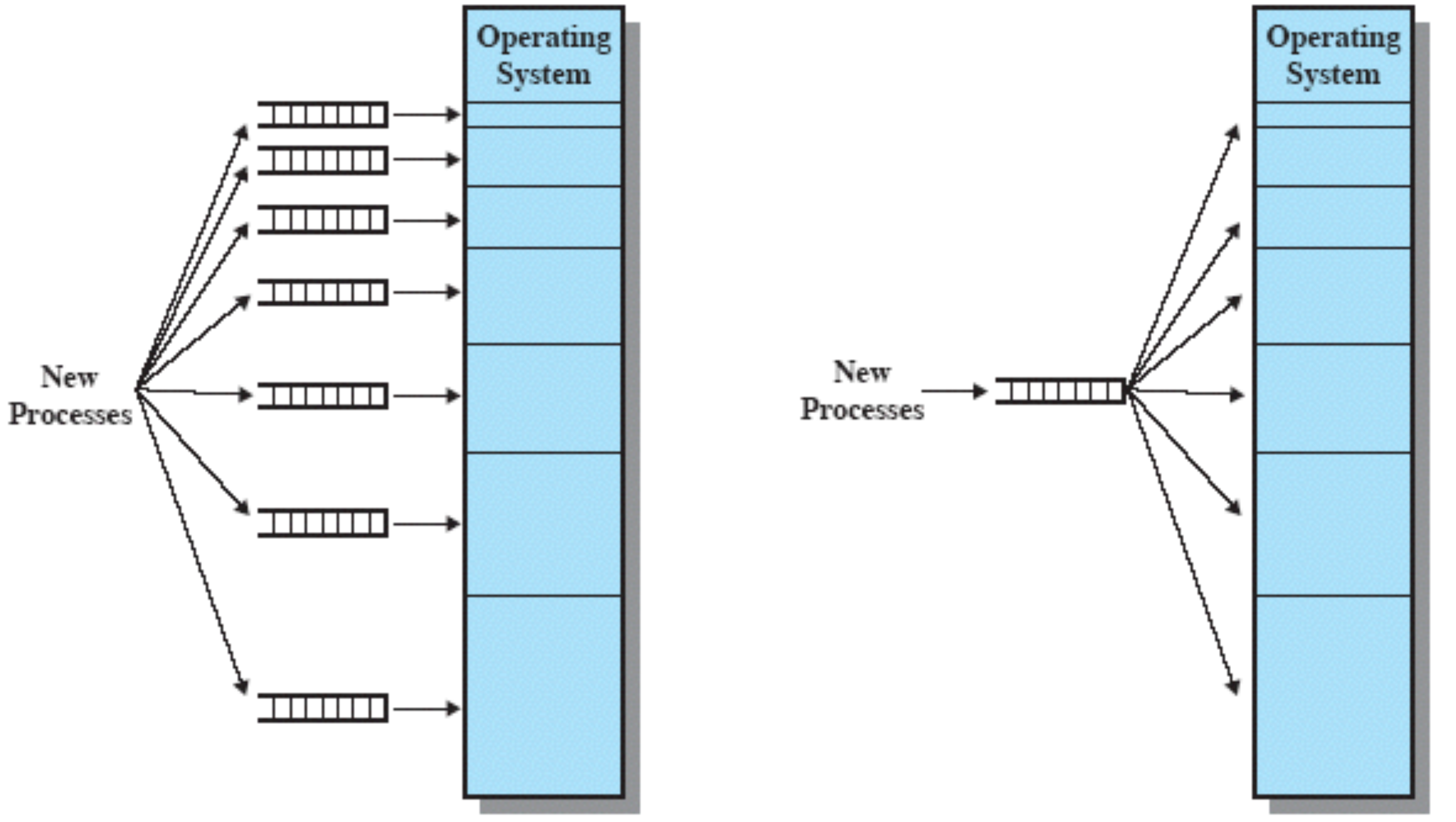
- A program may be too large for a partition. In this case the programmer must use “overlays”, wherein the program loads pieces of itself into memory to execute, overlaying other pieces.
- Memory utilization is poor, since a small program occupies a large partition (called “internal fragmentation”).

Partitioning: Unequal Sized Partitions

- Both of the problems of equal-sized partitions can be lessened by using unequal-sized partitions.
- Small programs can be loaded into small partitions improving memory utilization, and large programs can be loaded into large partitions helping to avoid the need for overlays.

Placement Algorithm

- Placement of programs in an equal-size partitioned memory is trivial since all partitions are the same.
- For unequal-sized partitions, we could assign the process to the smallest partition that holds it.
- This reduces internal fragmentation, but requires queues for each partition.
- A disadvantage of this approach is that a large partition could be empty while a small program waits for a small partition.
- A better approach is to use a single queue for all partitions and assign a process to the smallest open partition that holds it.



(a) One process queue per partition

(b) Single queue

Figure 7.3 Memory Assignment for Fixed Partitioning

Advantages and Disadvantages of Fixed Partitions

- Fixed partition systems are simple and require minimal OS support and overhead.
- However:
  - The number of partitions limits the number of active (not suspended) processes.
  - Utilization is not as good as it could be since it is likely that part of a partition is unused.

Dynamic Partitioning

- With dynamic partitioning, a process is given a partition exactly as big is it requires.
- The partition size is not set at system initialization as in fixed partitioning, but is set dynamically as each program is loaded.
- The problem with this is “external fragmentation” as shown on the next slide.

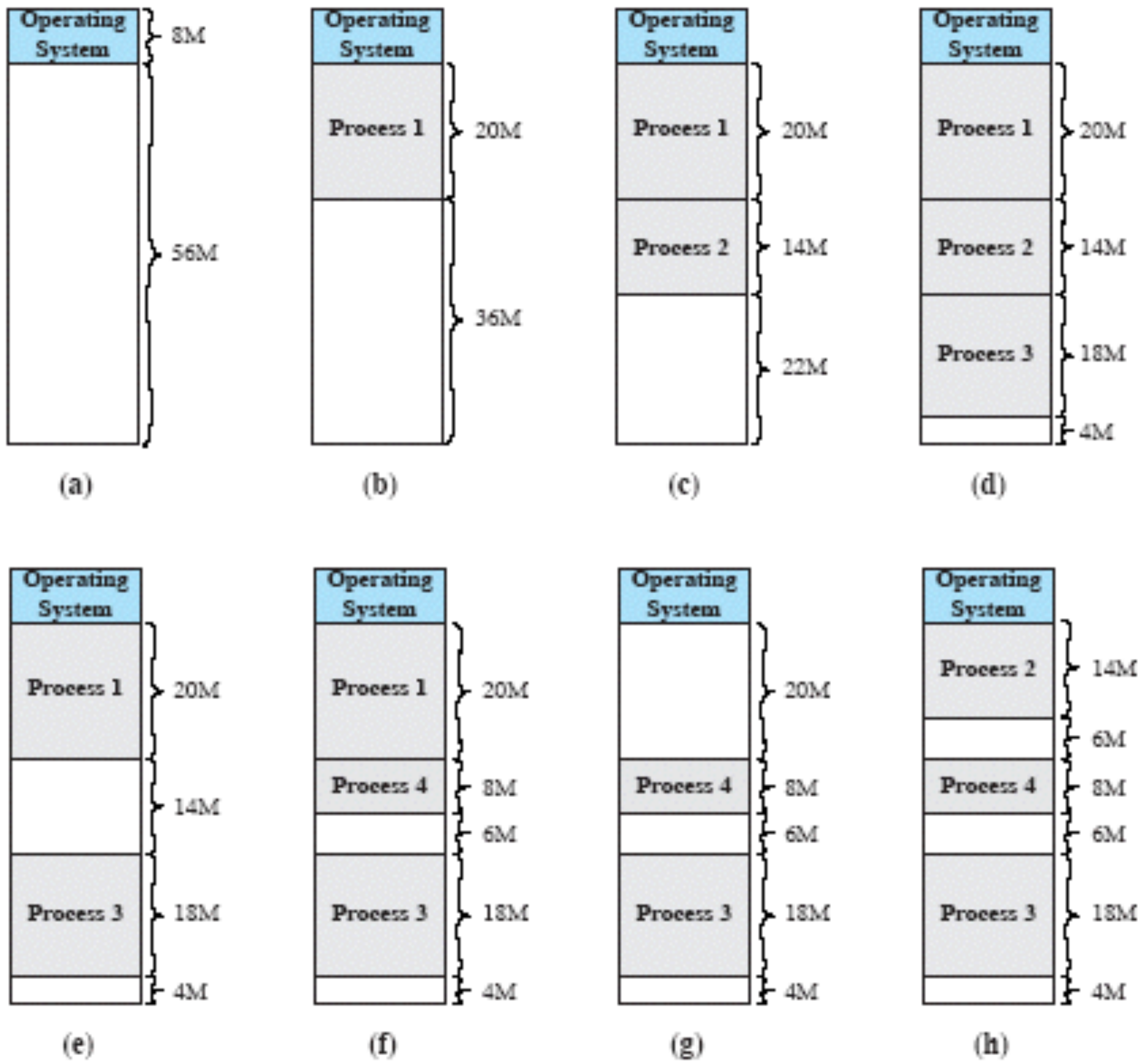


Figure 7.4 The Effect of Dynamic Partitioning

Compaction

- One way to deal with external fragmentation is a technique called “compaction”.
- Compaction shifts the processes around in memory so that there are no holes between them.
- The drawback to compaction is that it is very time consuming.

Placement Algorithms for Dynamic Partitioning

Three placement algorithms:

- Best-fit
  - Choose partition closest in size to the process.
- First-fit
  - Choose first available partition from the beginning of memory that is large enough to hold the process.
- Next-fit
  - Choose next available partition beginning at the previously chosen partition of memory that is large enough to hold the process.

Placement Algorithms

- First-fit is usually simplest, fastest, and best.
- Next-fit tends to fragment the end block of memory more than first-fit.
- Best-fit is usually worst. Causes smallest fragments which are unusable by other processes.

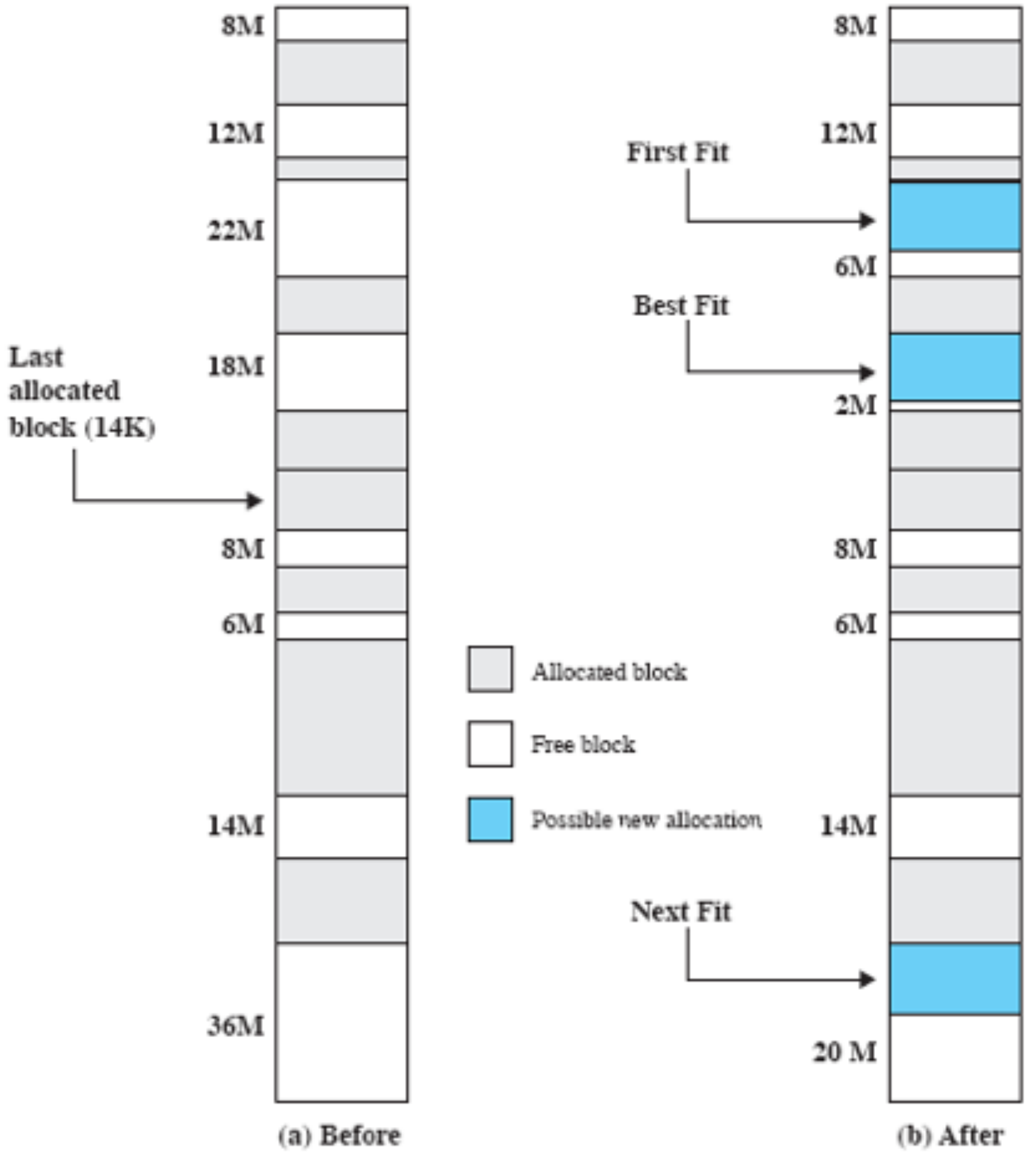


Figure 7.5 Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Buddy System

- Both fixed and dynamic partitioning schemes have drawbacks.
- Another approach is called the buddy system.
- The buddy system divides memory blocks in half until a block size that satisfies the request is found.
- When a block has been divided and is later free again, it can be combined to form the original size block again.

Buddy System

- Consider memory being of size 2^U.
- Then for the first request, we consider does the block fit between 2^{U-1} and 2^U.
- If so, allocate 2^U.
- If not, consider if the request fits between 2^{U-2} and 2^{U-1}.
- If so, allocate 2^{U-1}.
- If not, repeat the division until such a block is found.

Buddy System

- The buddy system maintains a set of lists containing blocks of size 2^i.
- If a request is made, a 2^{i+1} block in the i+1 list may be split into two blocks of size 2^i in the i list.
- Whenever a split pair of blocks on the i list are unallocated, they are combined and placed back on the i+1 list.

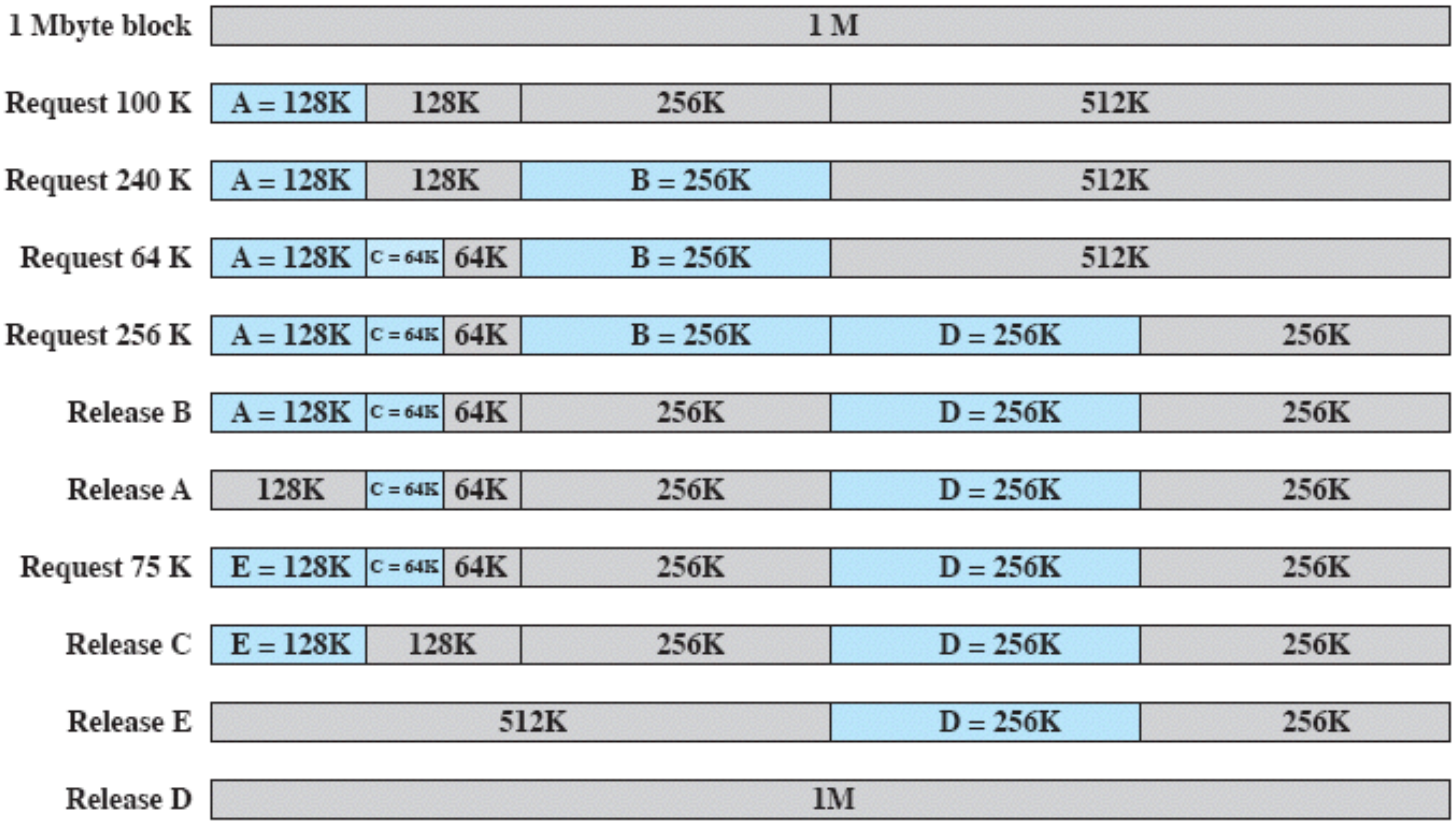


Figure 7.6 Example of Buddy System

Note: split blocks are recombined when free.

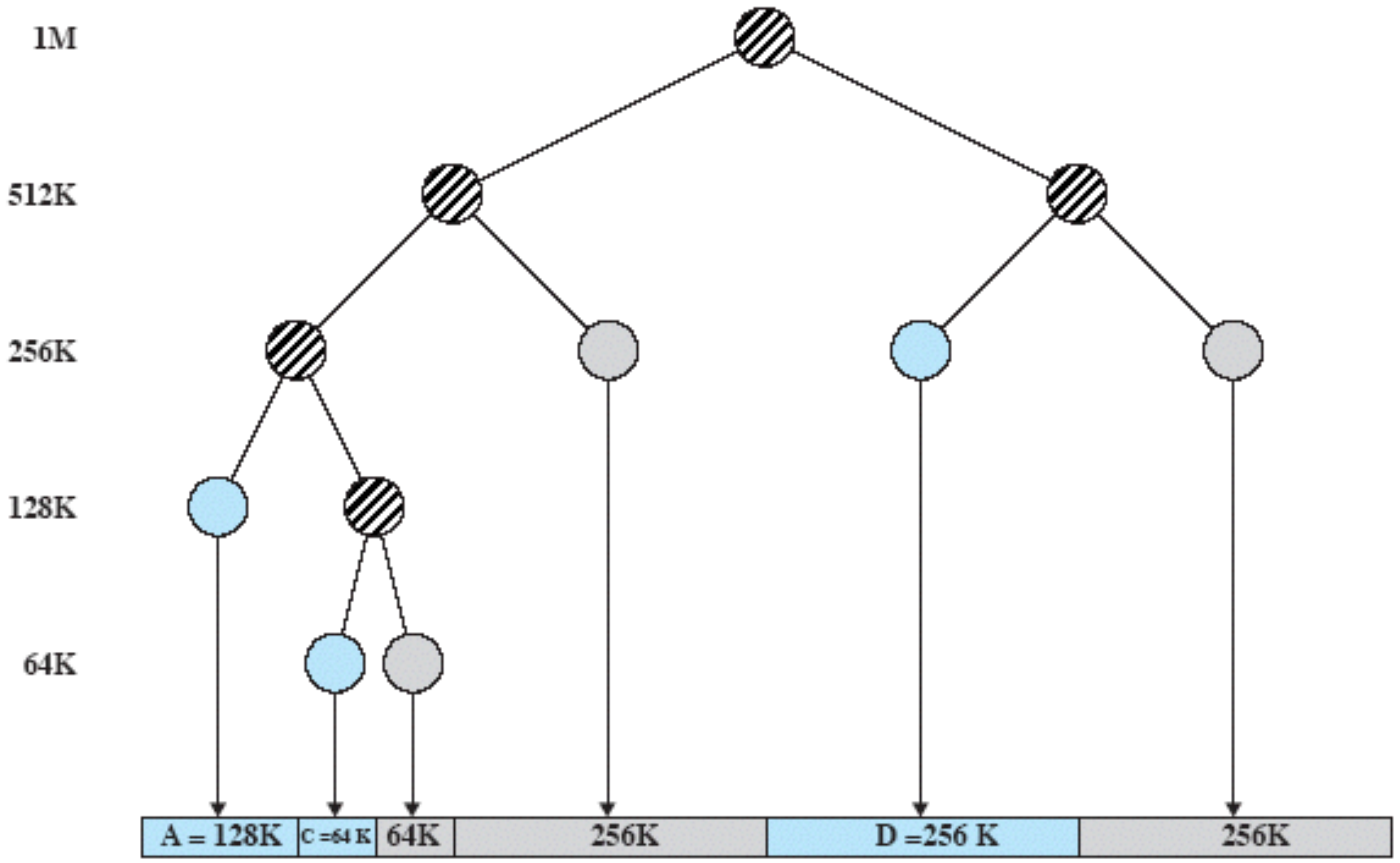


Figure 7.7 Tree Representation of Buddy System

After the release B request

Relocation

- In a fixed equal-sized partition system, a process may be loaded into any open partition. When swapped out and back in, it may be loaded into a different partition.
- In a fixed unequal-sized partition system, if a single queue is used, a swapped process may load into a different partition than where it was originally loaded.
- In dynamic partitioning, compaction may move processes around in memory.
- So…an absolute physical addressing scheme will not work in these situations.

Relocation

Types of addresses:

- Physical address
  - This is the actual memory address.
  - Also called the absolute address.
- Logical address
  - Address from program's viewpoint independent of current loaded position.
  - Relative address: A logical address relative to a known point (beginning of program, for example).

Address Translation

- If relative addressing is used relative to the beginning of the program, then these addresses need to be translated to physical addresses depending on where it is loaded.
- This can be done at runtime with special hardware.
- For example, a "base register" may contain the physical address of the program origin. This is then added to each relative address to get the physical address. A "bounds register" ensures the new address is inside this process's memory space.

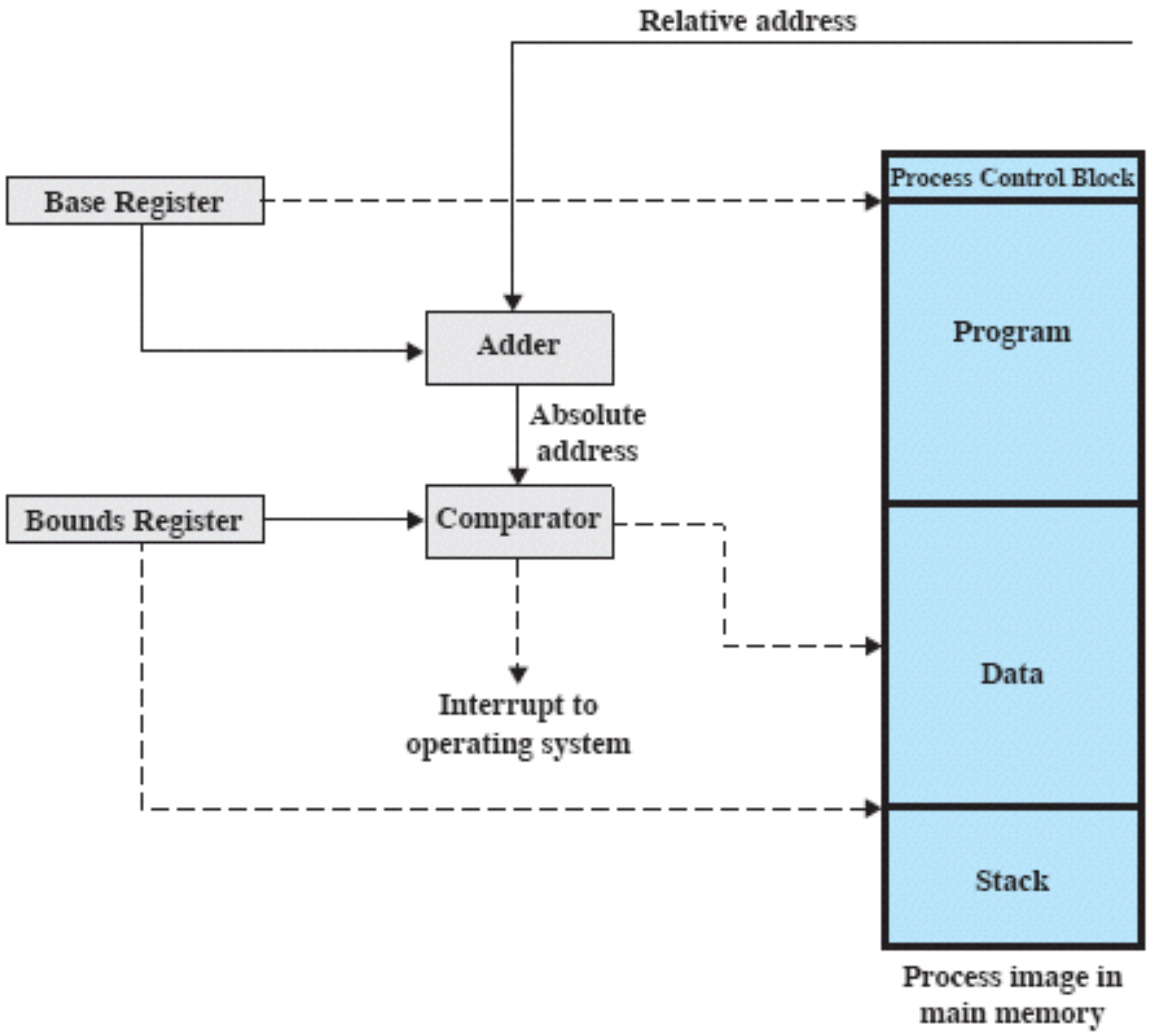
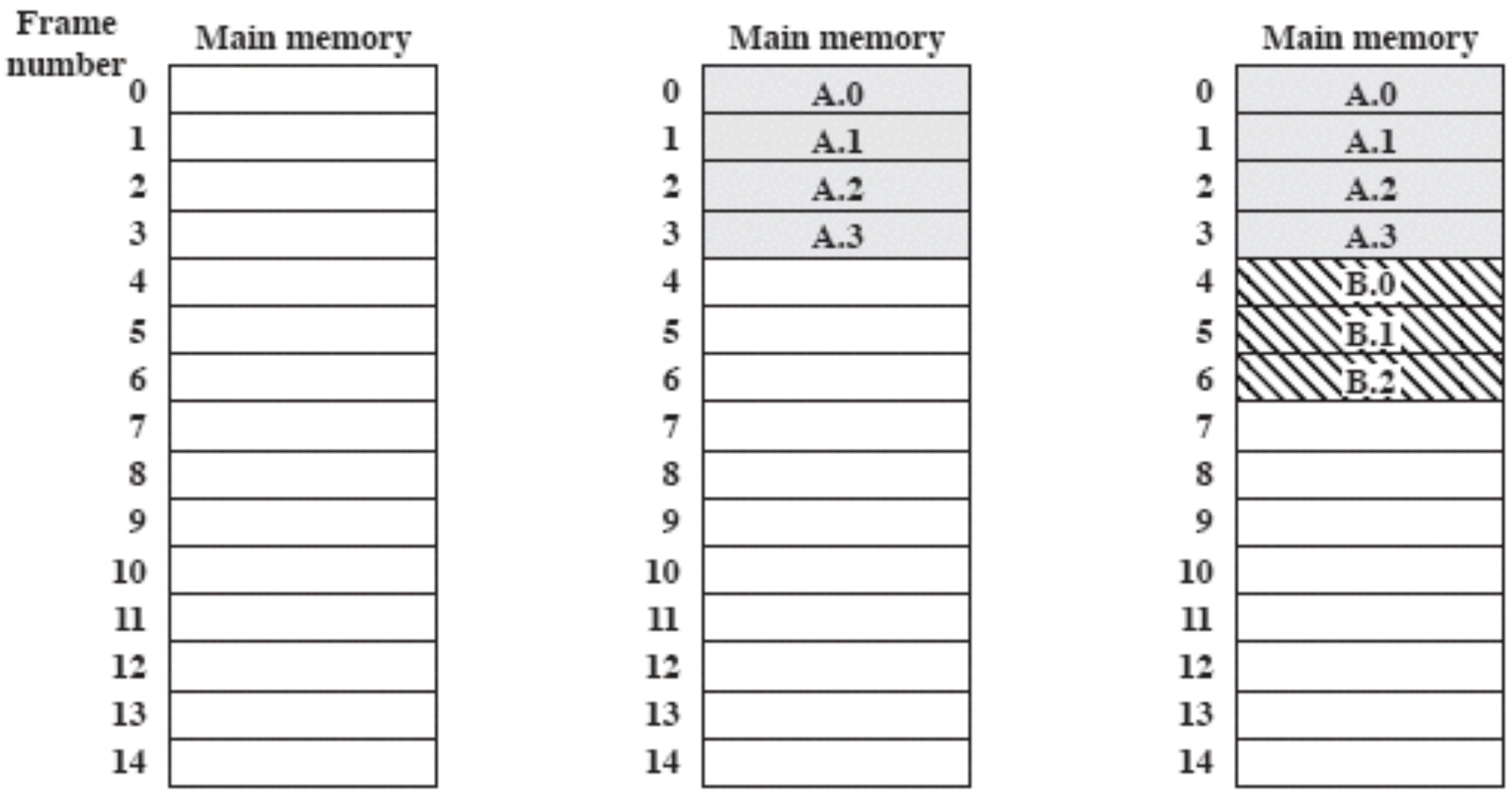


Figure 7.8 Hardware Support for Relocation

Paging

- Paging is a more modern memory management approach.
- In paging, the memory is partitioned into small equal-size chunks.
- Likewise a program is divided into the same equal-size chunks.
- The chunks of a program are called pages and chunks of memory are called frames.



(a) Fifteen Available Frames (b) Load Process A (c) Load Process B

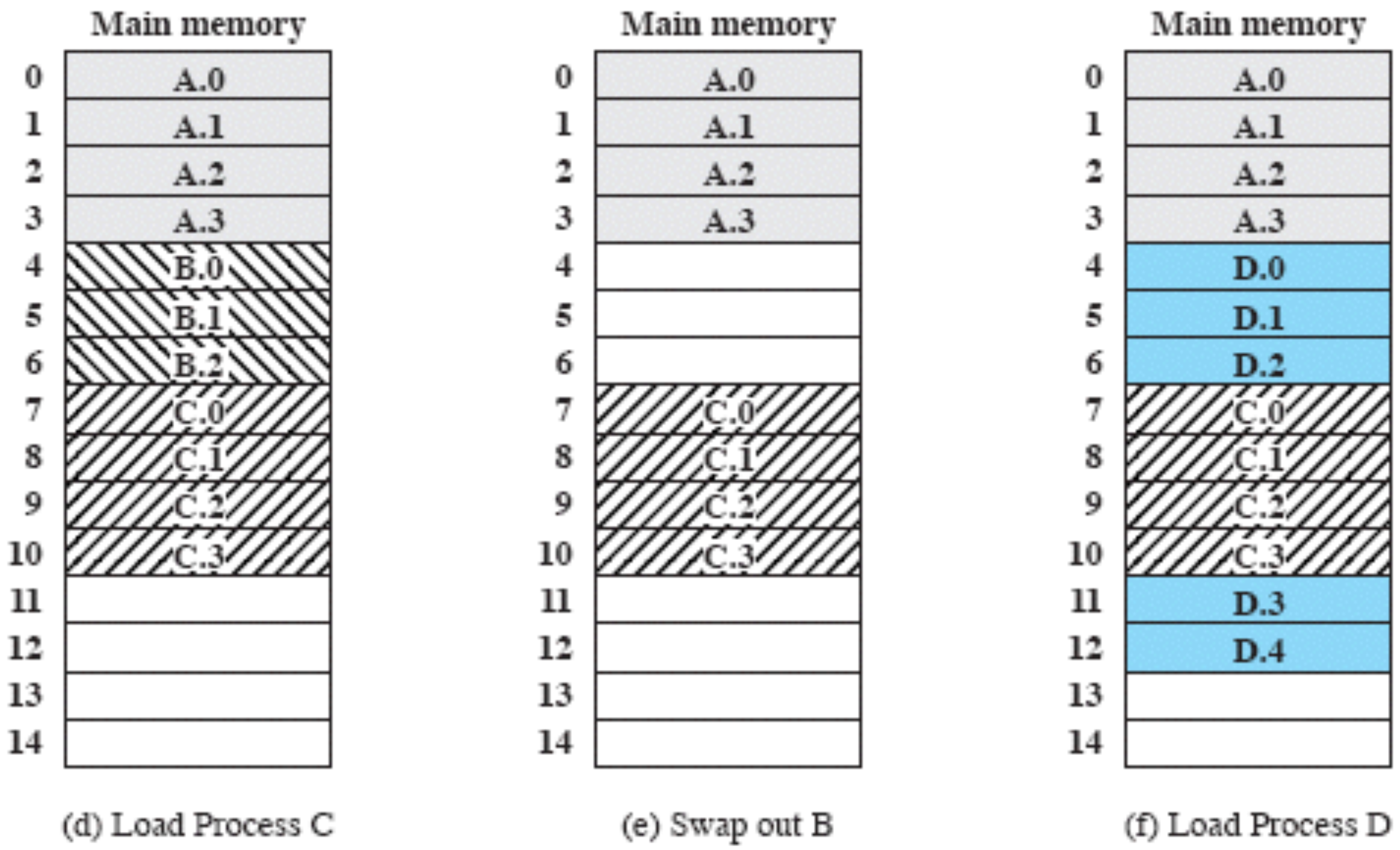


Figure 7.9 Assignment of Process Pages to Free Frames

p.318

Paging

- The OS maintains a page table for each process.
- Each entry of a page table consist of the frame number where the corresponding page is physically located in memory.
- The page table is indexed by the page number to obtain the frame number.
- A free frame list, available for pages, is also maintained.

Page tables for each process in (f) of previous slide.

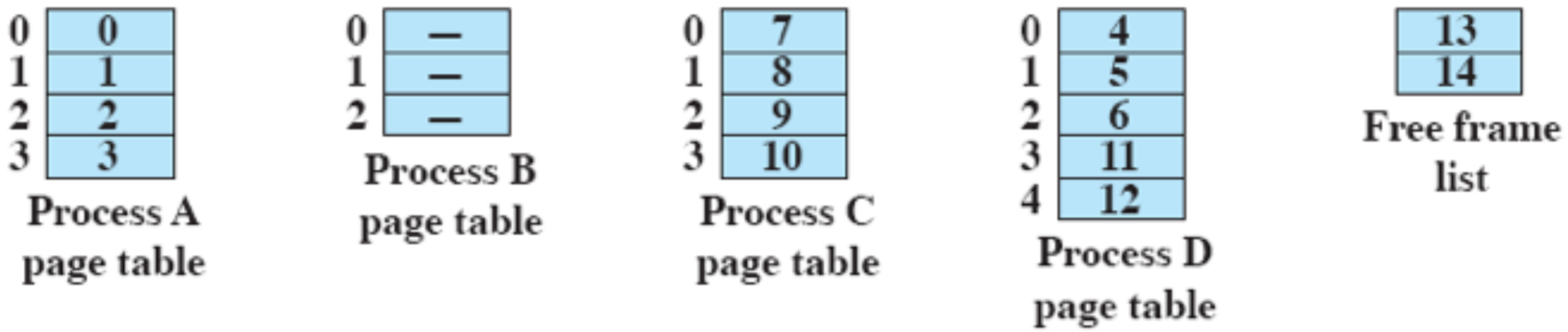


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Paging

- In a paging system, within each program, each logical address consists of a page number and an offset within the page.
- A CPU register holds the starting physical address of the page table of the process which is running.
- Presented with a logical address (page number, offset) the CPU accesses the page table to obtain the physical address (frame number, offset).

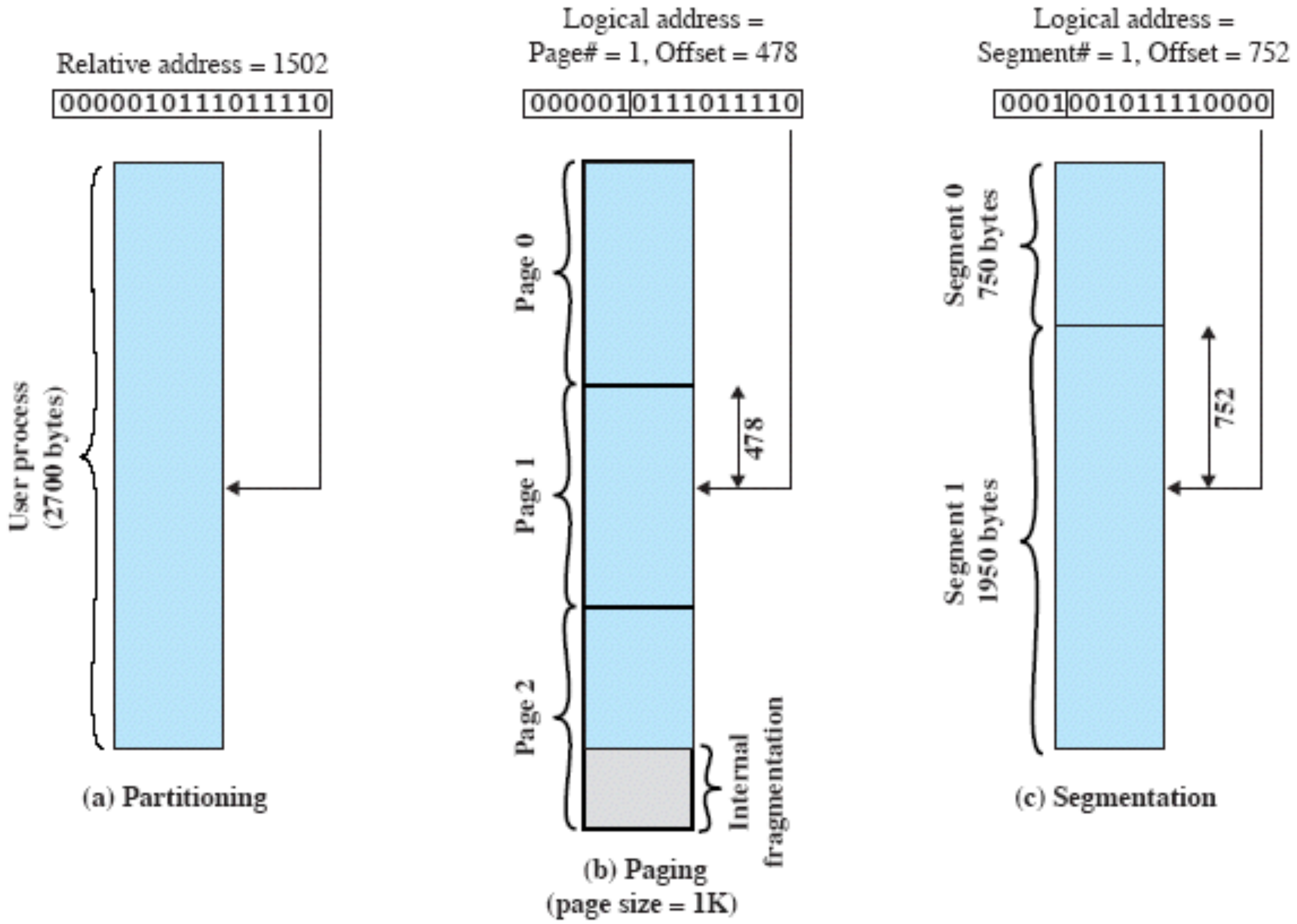
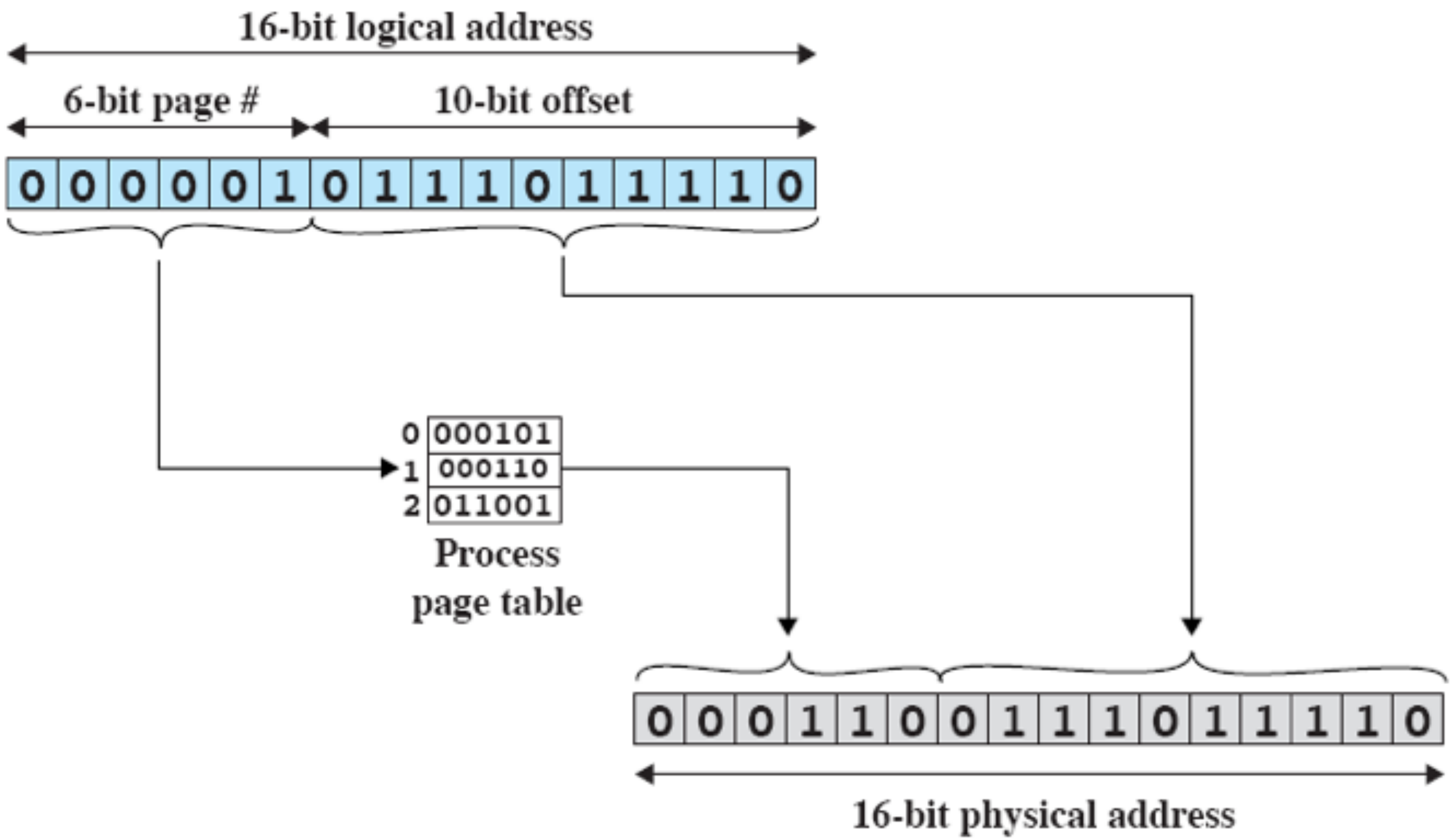


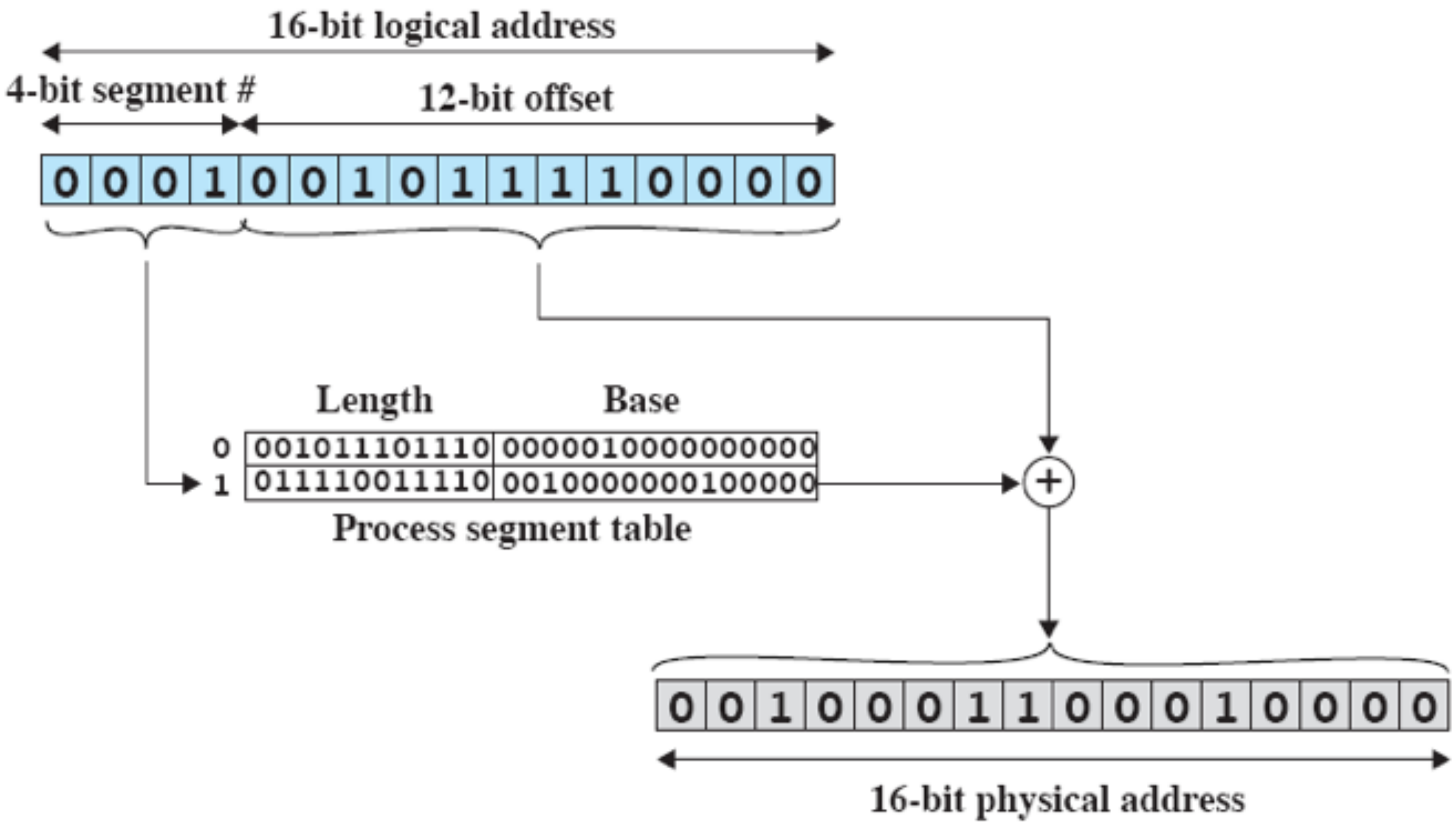
Figure 7.11 Logical Addresses


16-bit logical address
6-bit page #
10-bit offset
0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 0
0 000101
1 000110
2 011001
Process page table
0 0 0 1 1 0 0 1 1 1 0 1 1 1 1 0
16-bit physical address

(a) Paging

Segmentation

- Similar to paging but uses a segment number and offset.
- Segments are not equally-sized.
- Similar to dynamic partitioning except a program can span multiple segments that need not be contiguous.
- Suffers external fragmentation.
- Segments usually are visible to the programmer for partitioning the program into segments and assigning properties to the segment (e.g. read-only).
- Segment table gives starting address of segment.
- Chapter 8 has more information on segmentation.



(b) Segmentation

End of Slides

based on Stalling's official slides 40