CSE 306 Operating Systems
Memory Management

YoungMin Kwon

Memory Management

- In uniprogramming environment
  - Main memory is divided into OS and another program
- In multiprogramming environment
  - The user part of memory is further divided for multiple processes
- Memory management
  - The task of dynamically subdividing memory performed by OS

Memory Management Requirements

- Relocation
- Protection
- Sharing
- Logical organization
- Physical organization

Memory Management Requirements

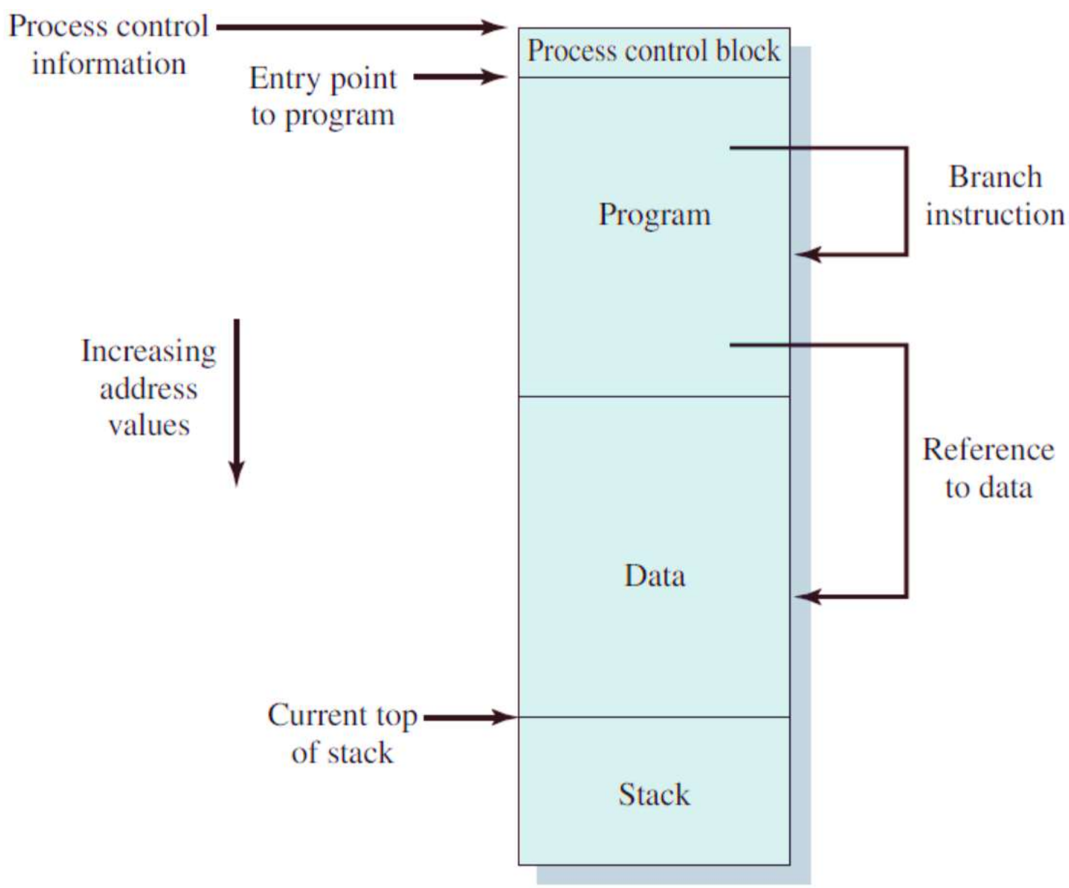
- Relocation
  - We cannot know where a program will be placed ahead of time
    - Programmers cannot know which other programs will be resident in main memory when their programs are being executed
  - We must allow that programs may be moved in main memory
    - When a program is swapped back in, it may be placed in a memory region different than when it was swapped out

Memory Management Requirements

- Relocation

  OS needs to manage

  - Location of PCB
  - Stack
  - Program entry point
  - Memory references for branch operations and data



Memory Management Requirements

- Protection
  - Normally, a user process should not access OS's program nor data
  - Without an agreement, a process should not access other processes' memory area
  - Memory protection should be provided by HW rather than OS
    - It will be prohibitively time consuming to check each memory access

Memory Management Requirements

- Sharing
  - Allow several processes to access the same portion of memory
  - Processes executing the same program can share the same code rather than having their own copy
  - Collaborating processes can share the same data structure

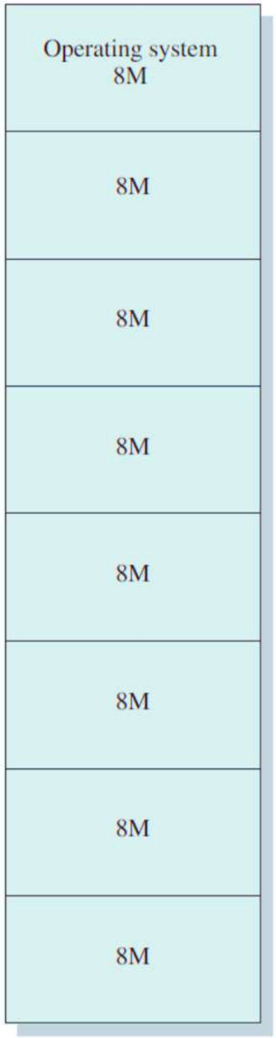
Memory Management Requirements

- Logical organization
  - Most main memory in a computer system is organized as a linear (one-dimensional) address space
  - Most programs are organized into modules
    - Some of which are unmodifiable (read only, execution only)
    - Some of which contain modifiable data
- Desirable system capabilities
  - Memory references among modules are resolved at runtime
  - Different degrees of protection to different modules (read only, execution only)
  - Modules can be shared among processes

Memory Management Requirements

- Physical organization
  - Computer memory is organized into at least two levels
    - Main memory: fast, expensive, volatile
    - Secondary memory: slower, cheaper, nonvolatile
  - Moving information between the two levels of memory is a system responsibility

Memory Partitioning

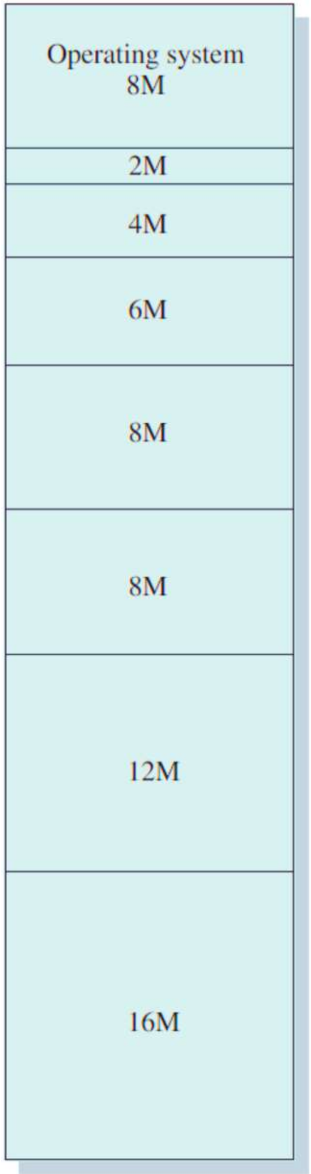


- Fixed partitioning
  - OS occupies some fixed portion of main memory
  - The rest of main memory is available for multiple processes
  - One simple scheme is to partition the available memory into fixed regions

Fixed Partitioning

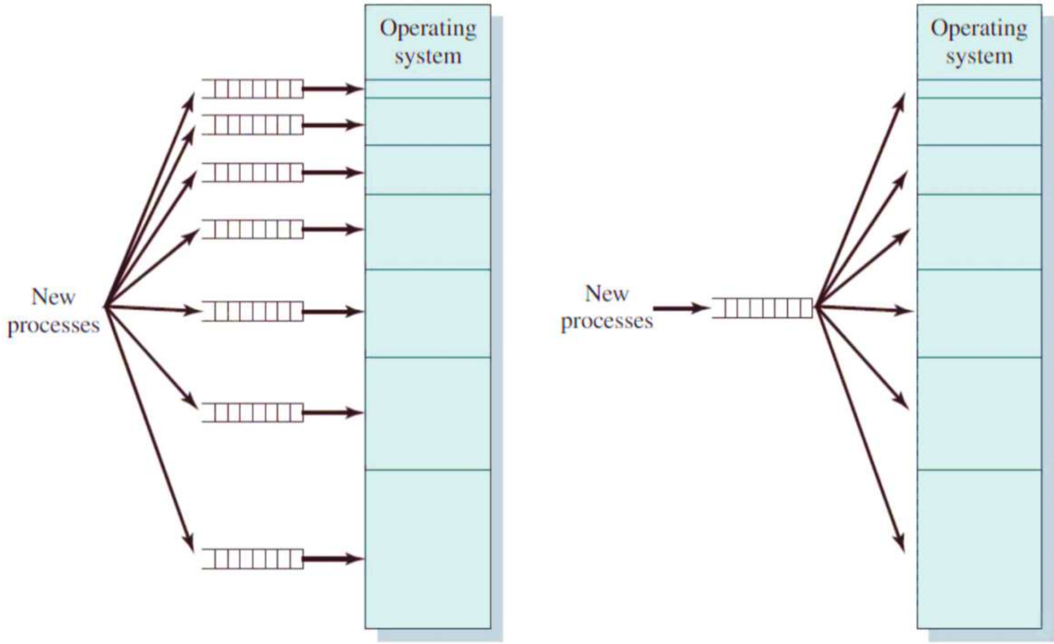
- Equal size fixed partitioning
  - Any process whose size is less than or equal to the partition size can be loaded
- Issues
  - A process may be too big to fit into a partition
    - Overlays: only a portion of a program is loaded in memory
    - Modules loaded at runtime need to use the process' partition (possibly with overlay)
  - Memory utilization is low
    - Internal fragmentation: no matter how small a process is, it takes an entire partition
    - Unequal size partitioning can lessen the issue

Fixed Partitioning



- Unequal size fixed partitioning
  - Assign a process to the smallest partition that can hold the process
    - Minimize internal fragmentation
    - Larger partitions may remain unused
  - Disadvantages
    - # of partitions specified at system generation time limits the # of active processes in the system
    - Small jobs will not utilize partition space efficiently

Fixed Partitioning

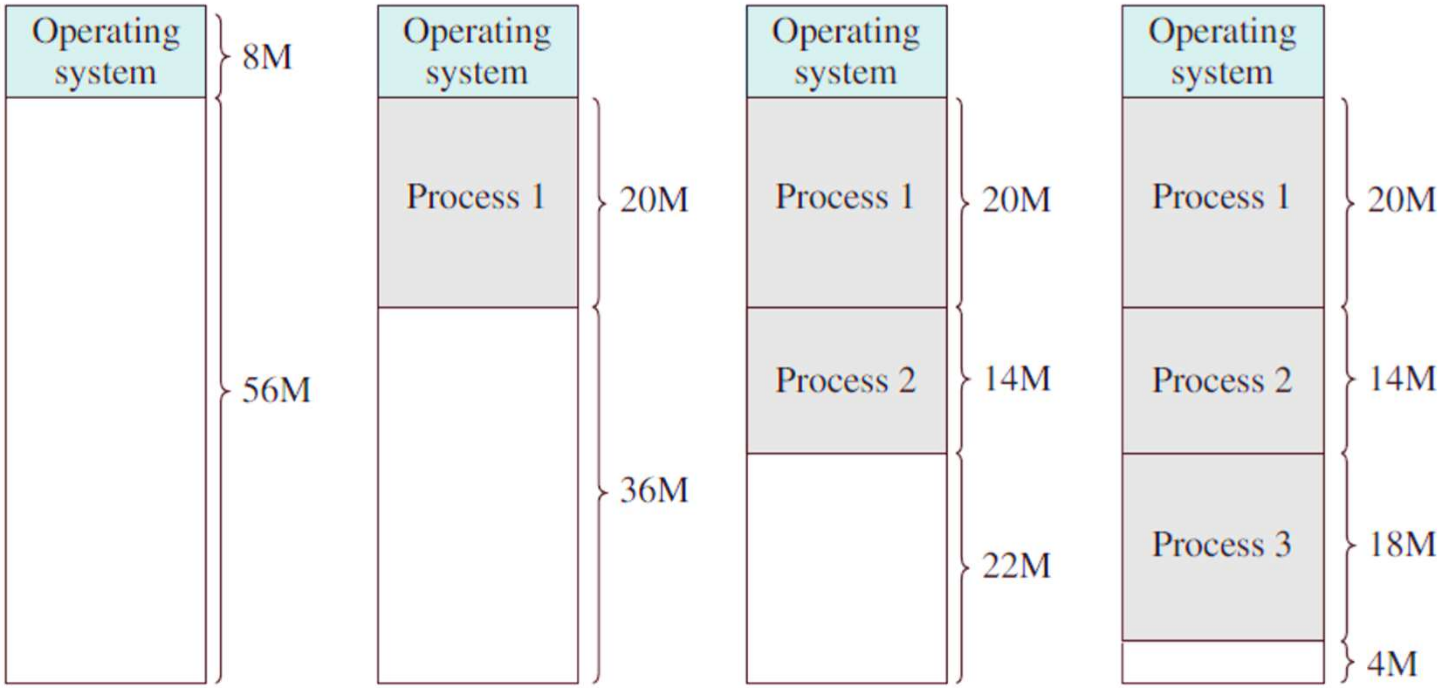


- Left: processes are assigned to the smallest partition that can hold the process

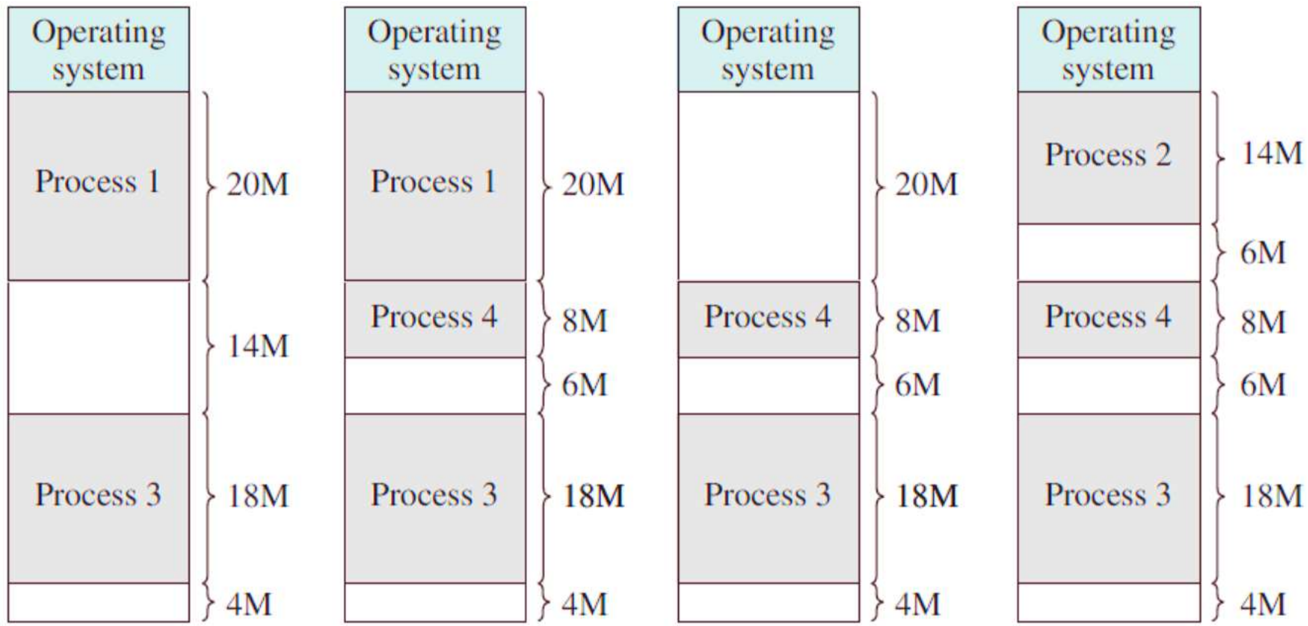
- Right: processes can be assigned to the smallest available partition that can hold the process
  - When processes are swapped in, find the smallest available partition again

Dynamic Partitioning

- The partitions are of variable length and number
  - When a process in loaded, it is allocated exactly as much memory as it requires



Dynamic Partitioning

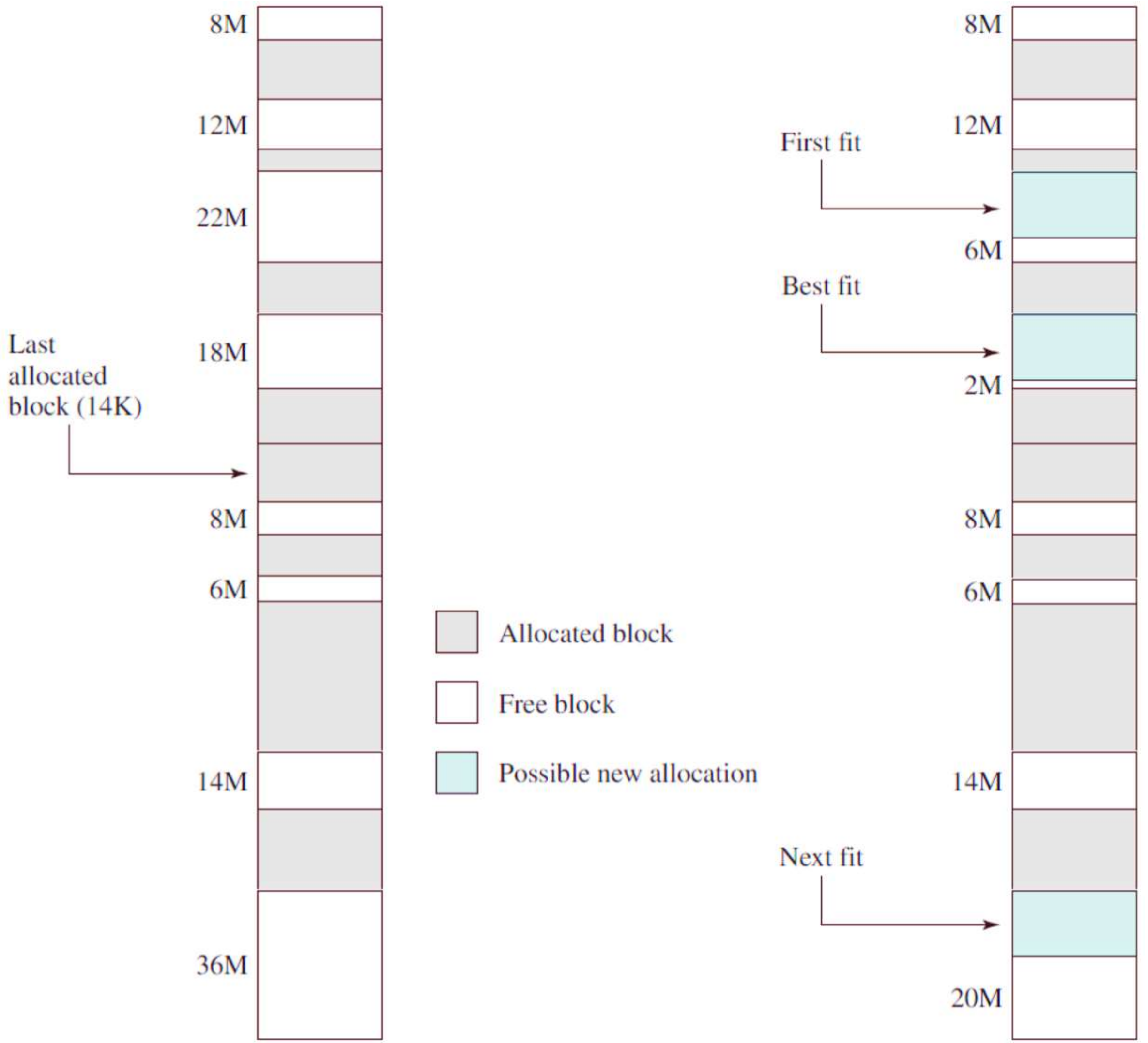


- External fragmentation
  - Memory that is external to all partitions becomes increasingly fragmented with time
  - Compaction: shift the processes so that they are contiguous and all free blocks are merged together

Dynamic Partitioning

- Placement algorithm
  - Best-fit: choose the block that is closest in size to the request
    - Memory can be easily fragmented
  - First-fit: find the first available block from the beginning
    - Leave large free blocks at the end of memory
  - Next-fit: find the available block from the last placement
    - Large free block of memory that usually appear at the end of memory is quickly broken up into small fragments

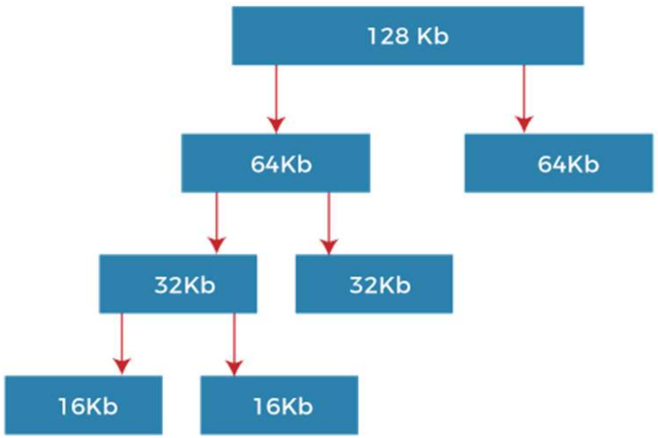
Dynamic Partitioning



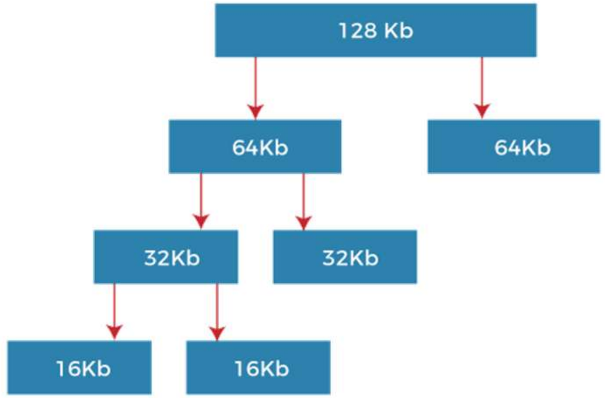
Before and after allocating a 16 MB block

Buddy System

- Buddy system
  - Memory blocks are available in 2^k words $L \le K \le U$, where
    - 2^L : smallest size of block
    - 2^U: largest size of block (entire memory)

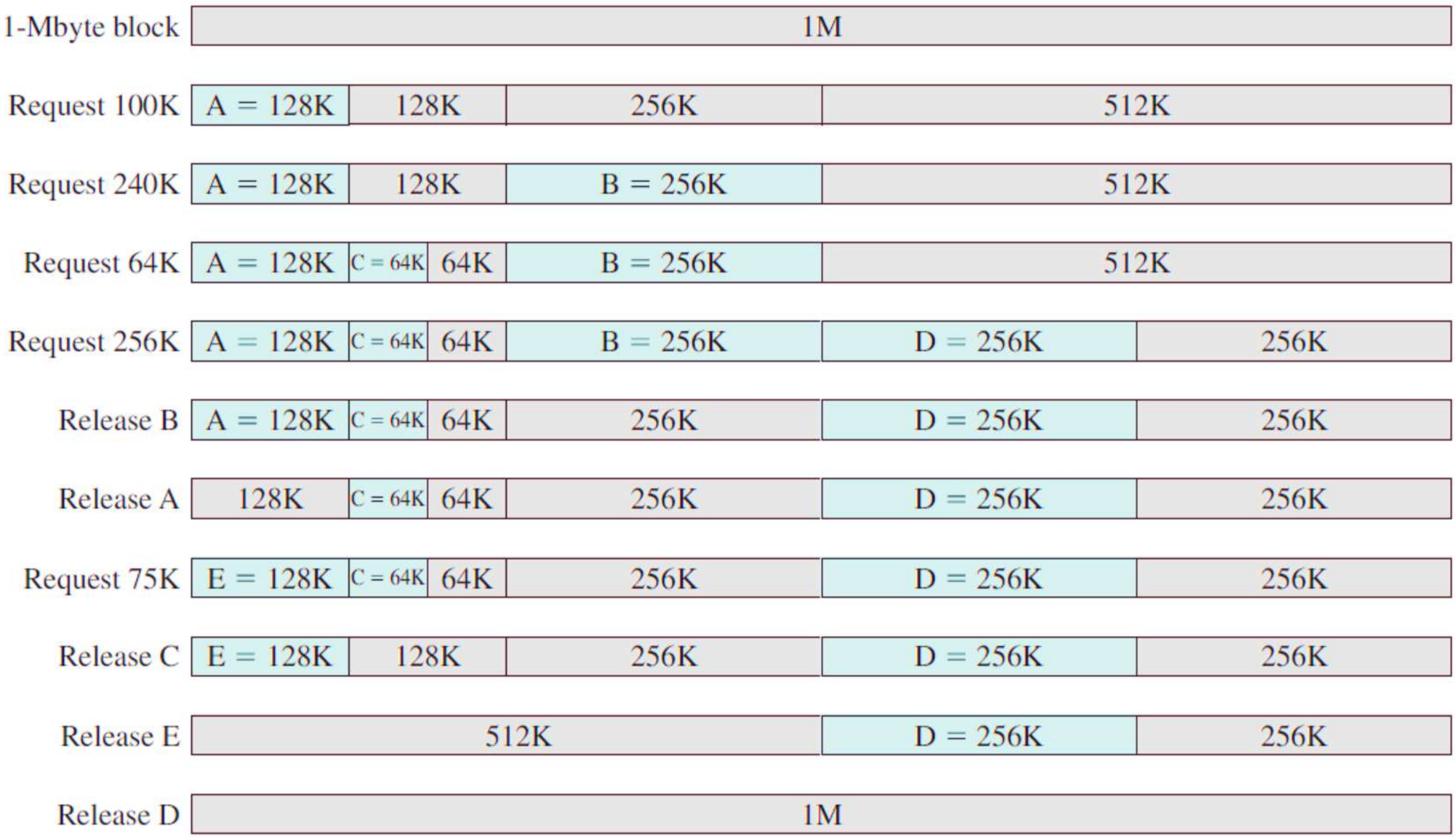


Buddy System

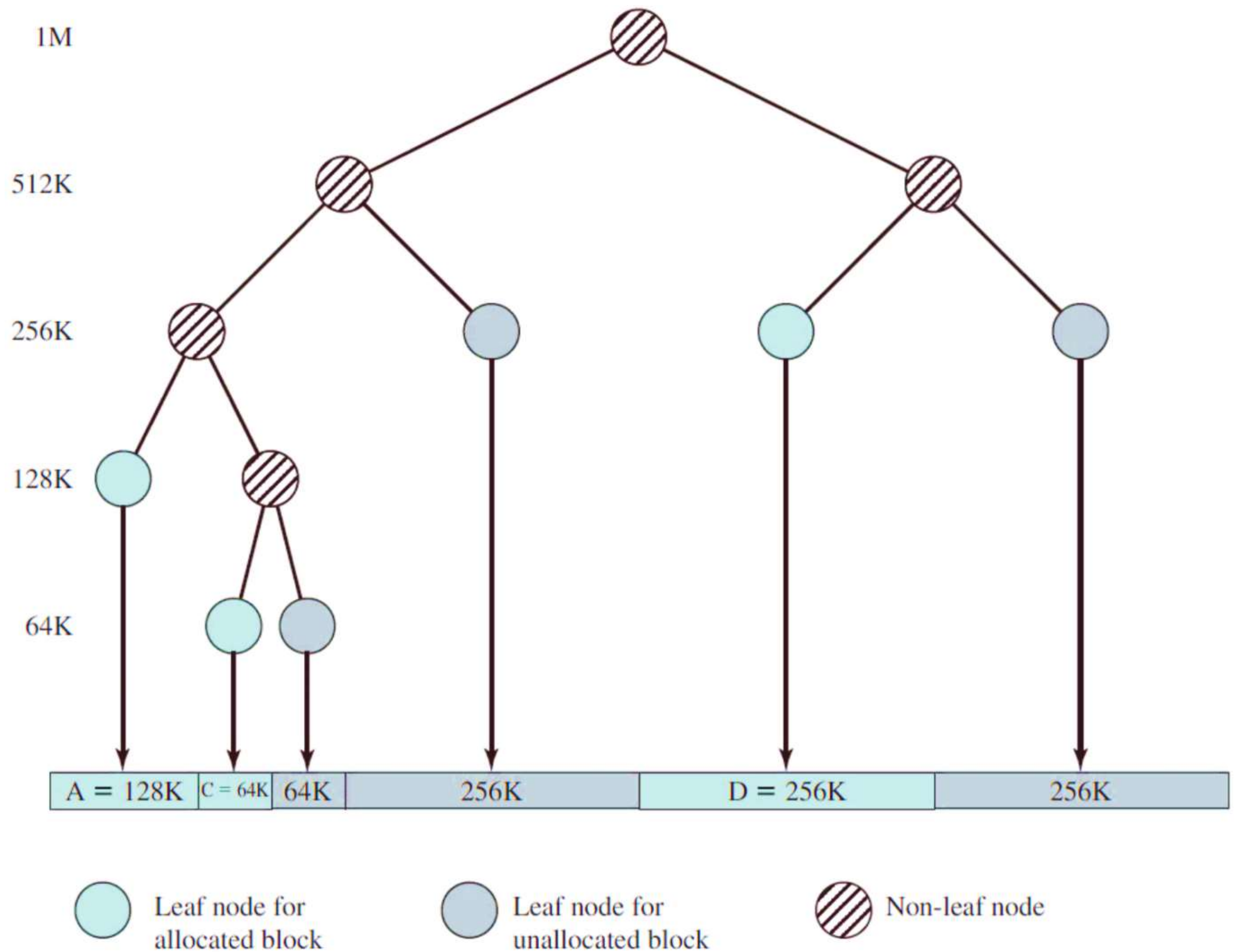


- Allocation algorithm
  - If the request size s is $2^{U-1} < s \leq 2^{U}$, the entire block is allocated
  - Otherwise, split the block into two 2^{U-1} size blocks. If the request size s is $2^{U-2} < s \leq 2^{U-1}$ the request is allocated to one of the blocks
  - Otherwise, split one of the block, and continue until block size becomes 2^{L}
  - Whenever two buddies are unallocated, they are coalesced into a single block

Buddy System



Buddy System

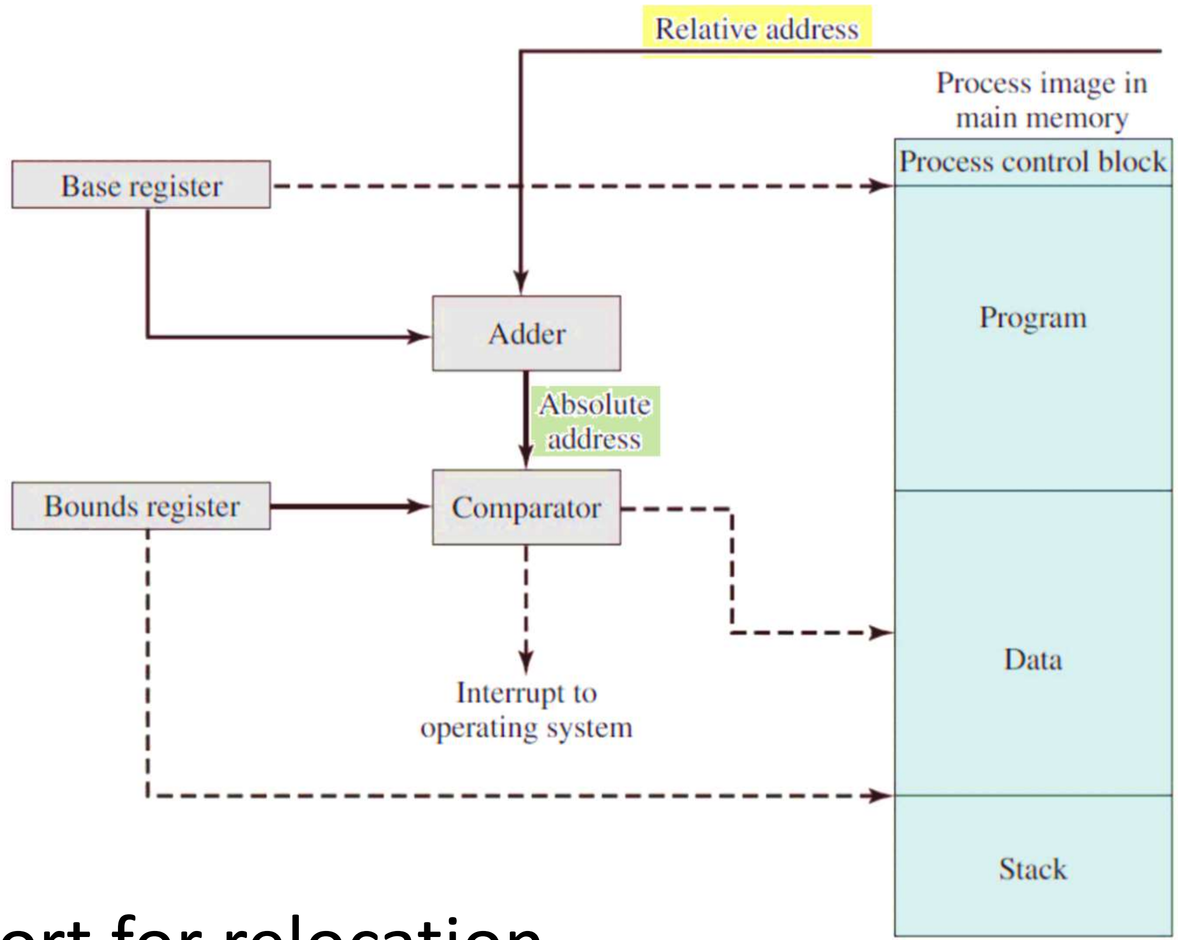


Tree representation of the buddy system

Relocation

- Relocation
  - A process may be swapped in to a partition different than the one when it was swapped out
  - After compaction processes are shifted while they are in memory
- Addresses
  - Logical address: reference to a memory location independent of the physical memory
  - Relative address: a logical address where addresses are relative to a known place (a register)
  - Physical address: absolute location in memory

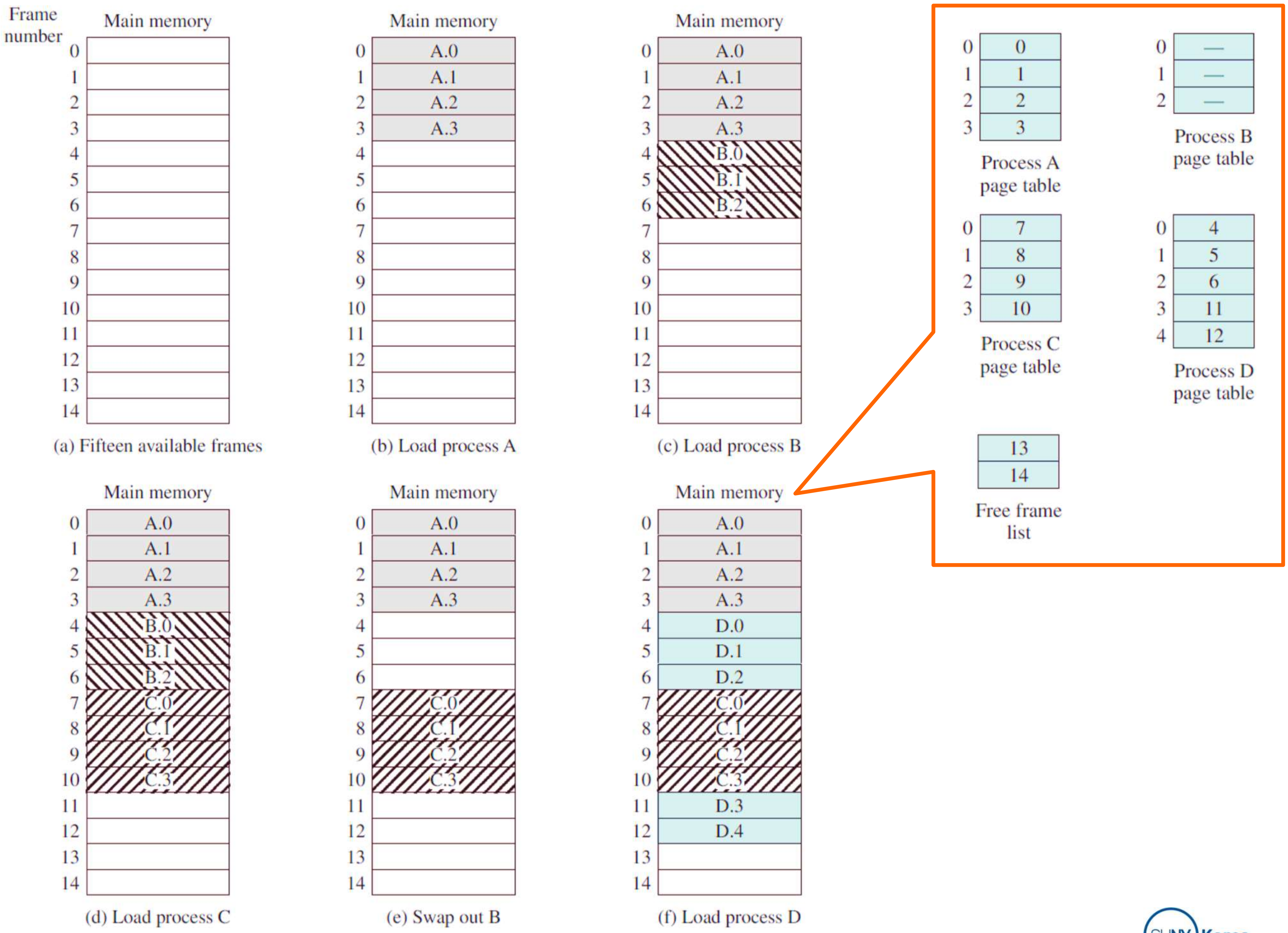
Relocation



- HW support for relocation
  - Value in the base register is added to the relative address
  - Resulting address is compared to the value in the bounds register

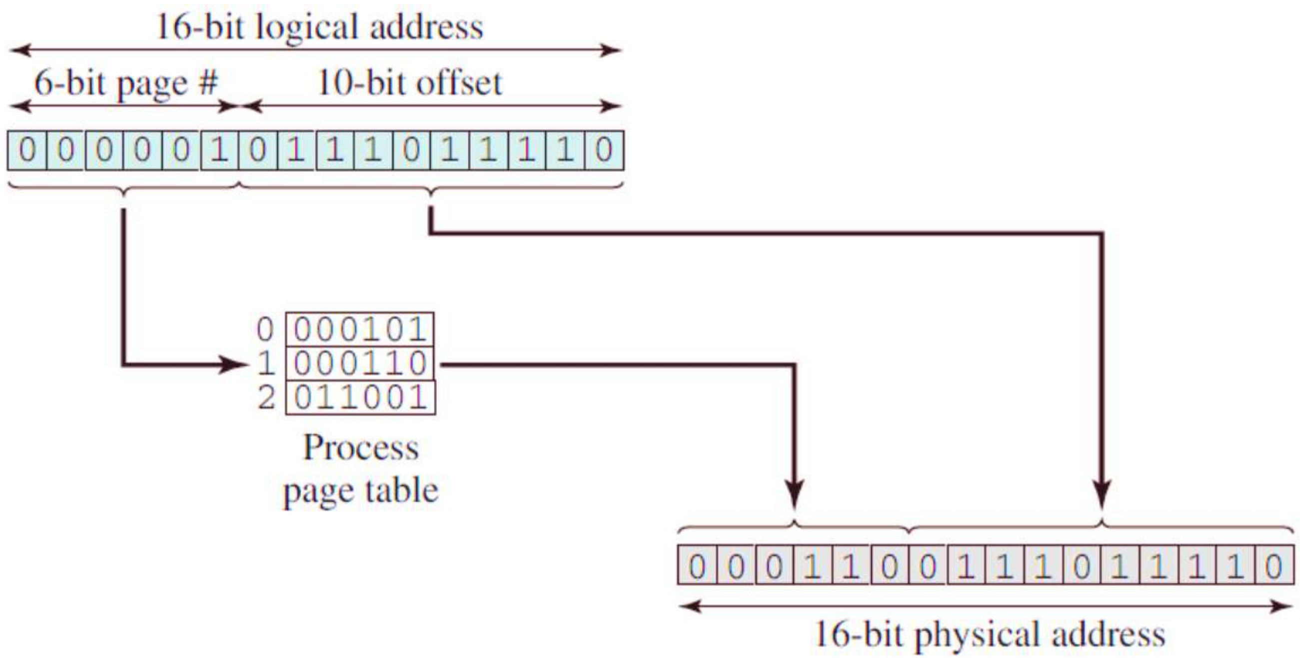
Paging

- Paging
  - Main memory is divided into small fixed size chunks
  - Each process is also divided into the same size chunks
  - Chunks of a process, known as pages, could be assigned to available chunks of memory, known as frames.
- Page table
  - A table of base registers for each page of a process
  - Logical address consists of a page number and an offset within the page
  - Address translation: (page number, offset) $\rightarrow$ (frame number, offset)


Frame number
Main memory
A.0
A.1
A.2
A.3
B.0
B.1
B.2
C.0
C.1
C.2
C.3
D.0
D.1
D.2
D.3
D.4
(a) Fifteen available frames
(b) Load process A
(c) Load process B
(d) Load process C
(e) Swap out B
(f) Load process D
Process A page table
Process B page table
Process C page table
Process D page table
Free frame list
13
14

Paging

- Address translation example
  - 16 bit address with the page size of 1KB
    - Page number: the first 6 bits
    - Offset: the last 10 bits
    - 0x05DE (0000 0101 1101 1110)

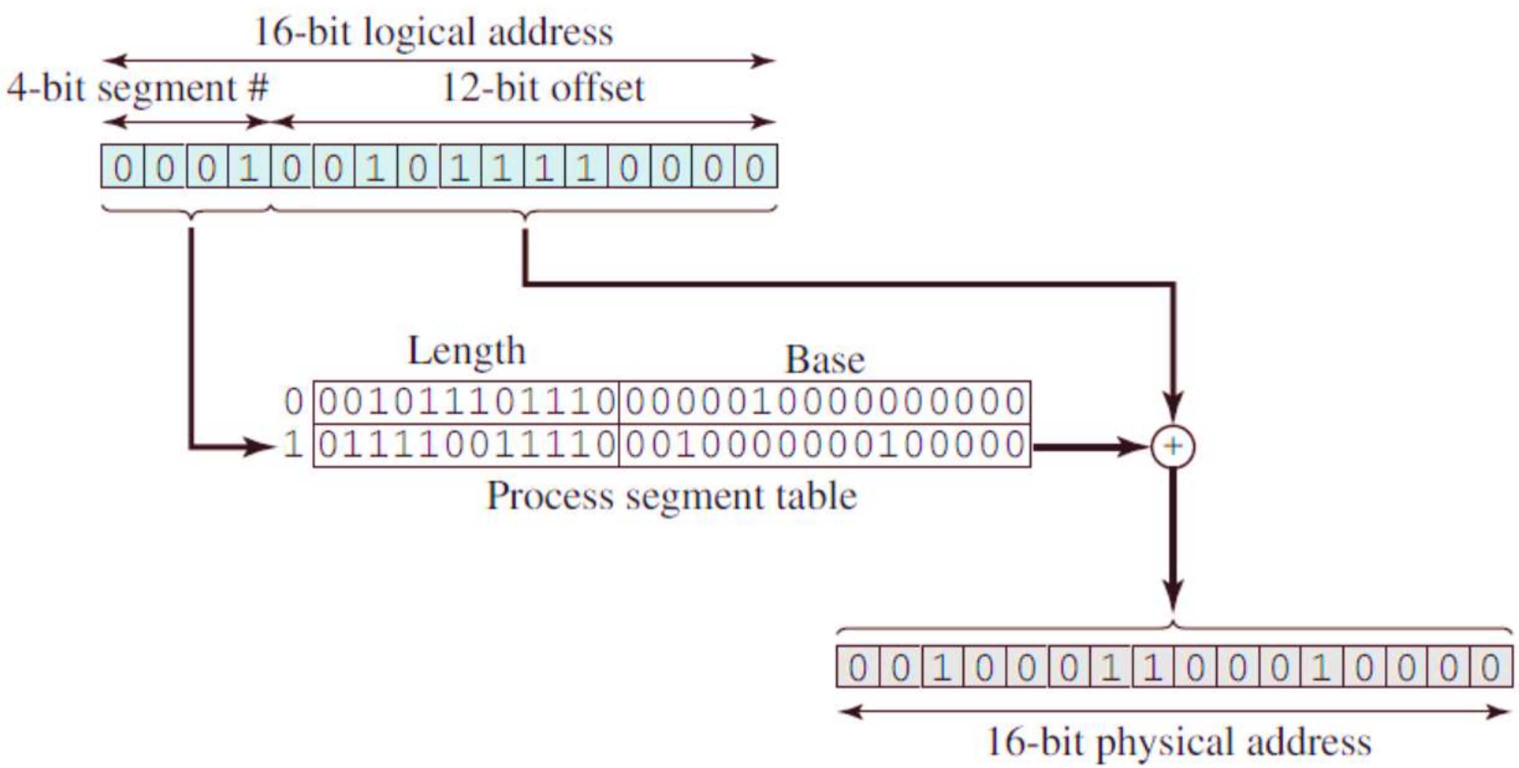


Segmentation

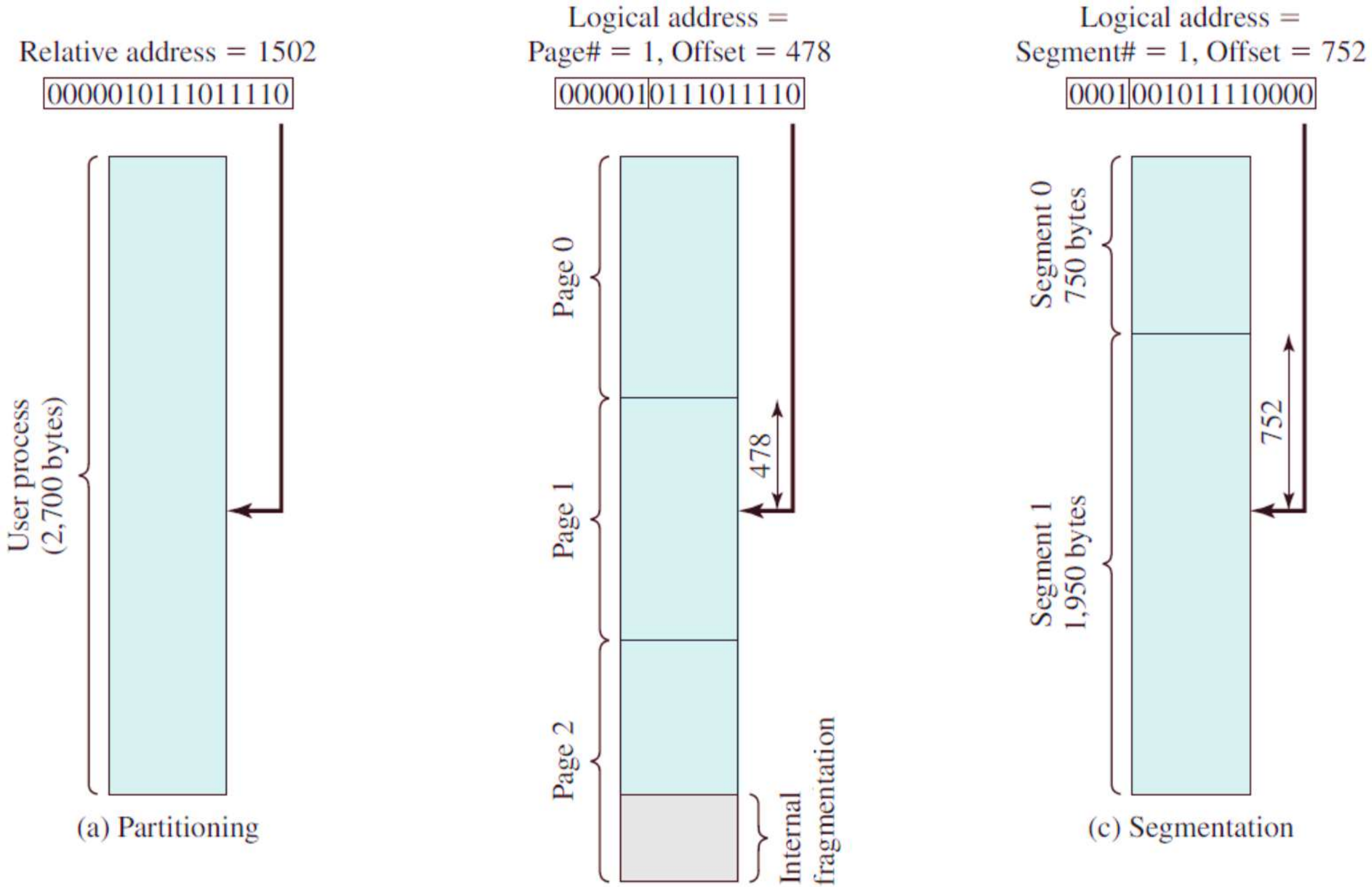
- Segmentation
  - Segments are unequal-size block of memory
  - Logical addresses consist of two parts
    - A segment number and an offset
  - A program may occupy more than one partition and the partitions need not be contiguous
    - No internal fragmentation
    - Less external fragmentation: a process is broken up into a number of smaller pieces

Segmentation

- Address translation example
  - 16 bit logical address with the first 4 bits for the segment number and the last 12 bits for the offset
  - 0x12F0 (0001 0010 1111 0000)



Logical Addresses

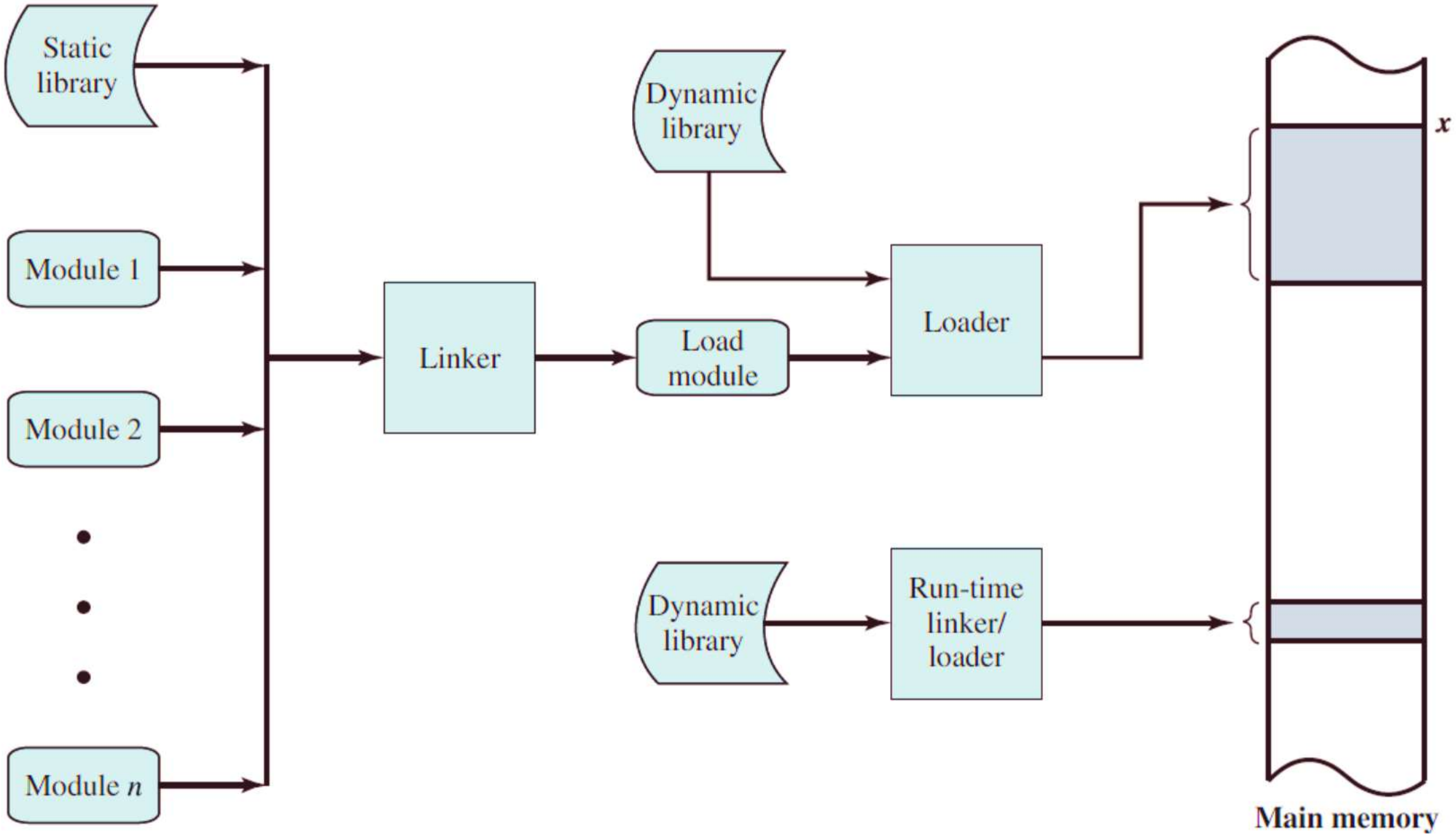


(a) Partitioning

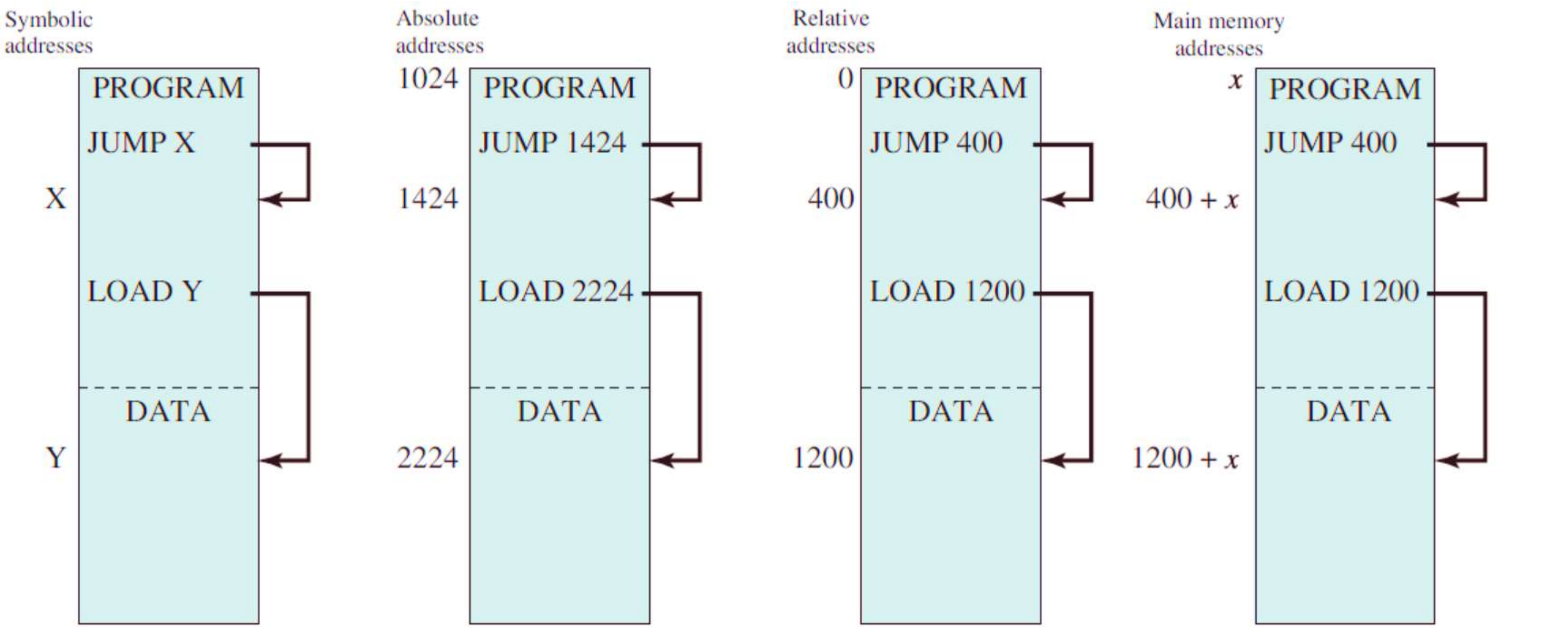
(b) Paging (page size = 1K)

(c) Segmentation

Linking and Loading



Absolute and Relocatable Load



(a) Object module

(b) Absolute load module

(c) Relative load module

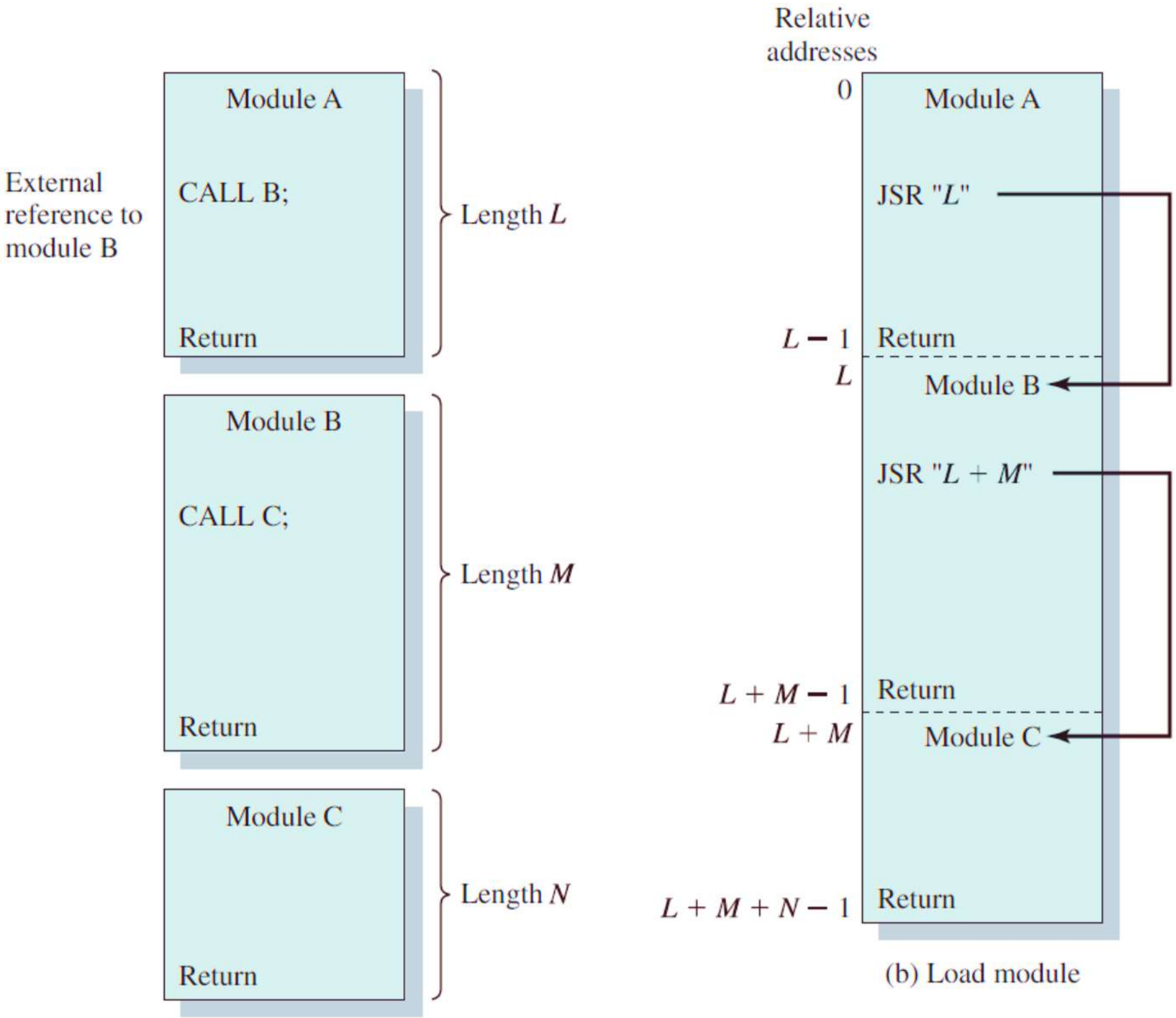
(d) Relative load module loaded into main memory starting at location *x*

Linking

- Each object module may contain references to symbols in other modules
  - Such references are expressed symbolically in an unlinked object module

- Linking
  - After combining the same sections, an address is assigned to the symbols in the modules
  - The symbol references in other modules are resolved

Linking



(a) Object modules

(b) Load module