Memory Management

summary

- goals and requirements
- techniques that do not involve virutal memory

memory management goals

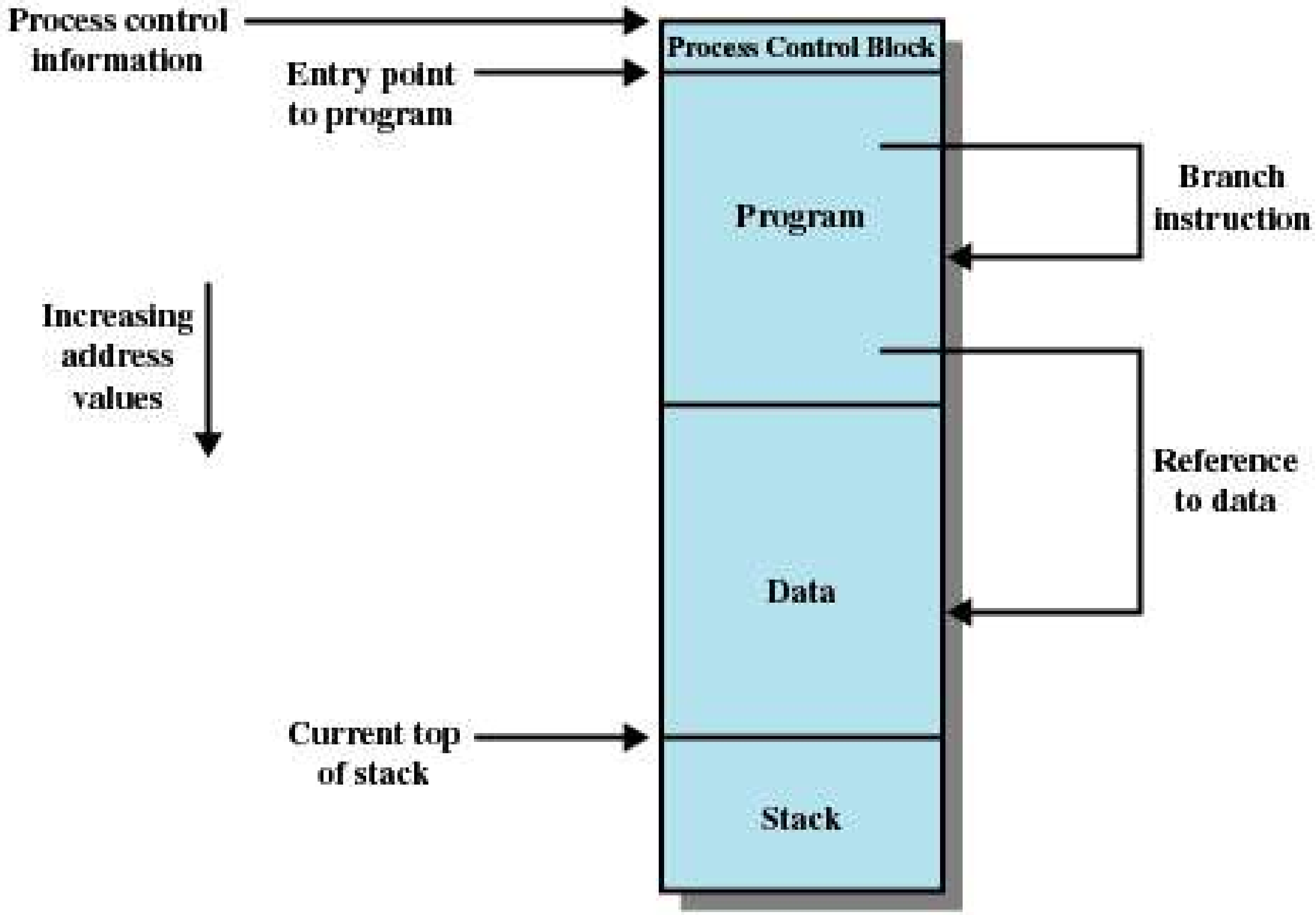
- subdividing memory to accommodate multiple processes
- memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time
- however the problem is general
  - memory allocators are used to reserve and free contiguous memory blocks
    - C (C++): malloc (new) and free (delete)
    - java management and garbage collectors
    - kernel data structure (beside processes)

memory management requirements for processes

- relocation
  – programmer does not know where the program will be placed in memory when it is executed
  – while the program is executing, it may be swapped to disk and returned to main memory at a different location (relocated)
  – memory references must be translated in the code to actual physical memory address

memory management requirements for processes

- Relocation



memory management requirements for processes

- protection
  - processes should not be able to reference memory locations in another process without permission
  - references **must** be checked at run time
    - impossible to check memory references at compile time (may directly depend on the input!)
  - Memory protection requirement must be satisfied by the processor (hardware) rather than the operating system (software)
    - Operating system cannot anticipate all of the memory references a process will perform

memory management requirements for processes

- sharing
  - allow several processes to access the same portion of memory
  - better to allow each process access to the same copy of the program rather than have their own separate copy

memory management requirements for processes

- logical organization
  - programs are written in modules
  - modules can be written and compiled independently
  - different degrees of protection given to modules (read-only, execute-only)
  - share modules among processes

memory management requirements for processes

- physical organization
  - memory available for a program plus its data may be insufficient
    - Overlaying allows various modules to be assigned the same region of memory
  - programmer does not know how much memory will be available

techniques that do not involves virtual memory

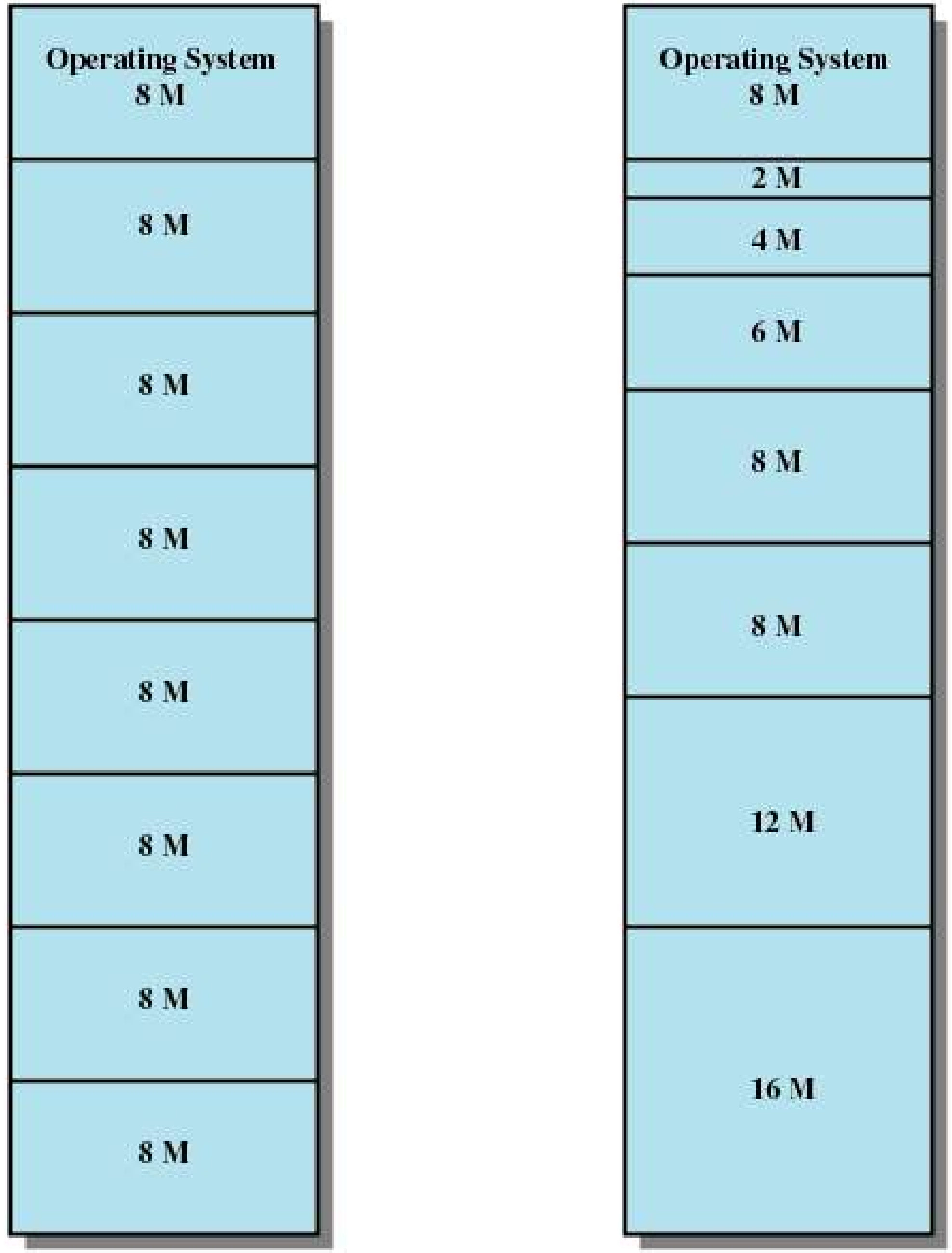
Fixed Partitioning

- Equal-size partitions
  - Any process whose size is less than or equal to the partition size can be loaded into an available partition
  - If all partitions are full, the operating system can swap a process out of a partition
  - A program may not fit in a partition. The programmer must design the program with overlays

Fixed Partitioning

- Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This is called **internal fragmentation**.

partitions size



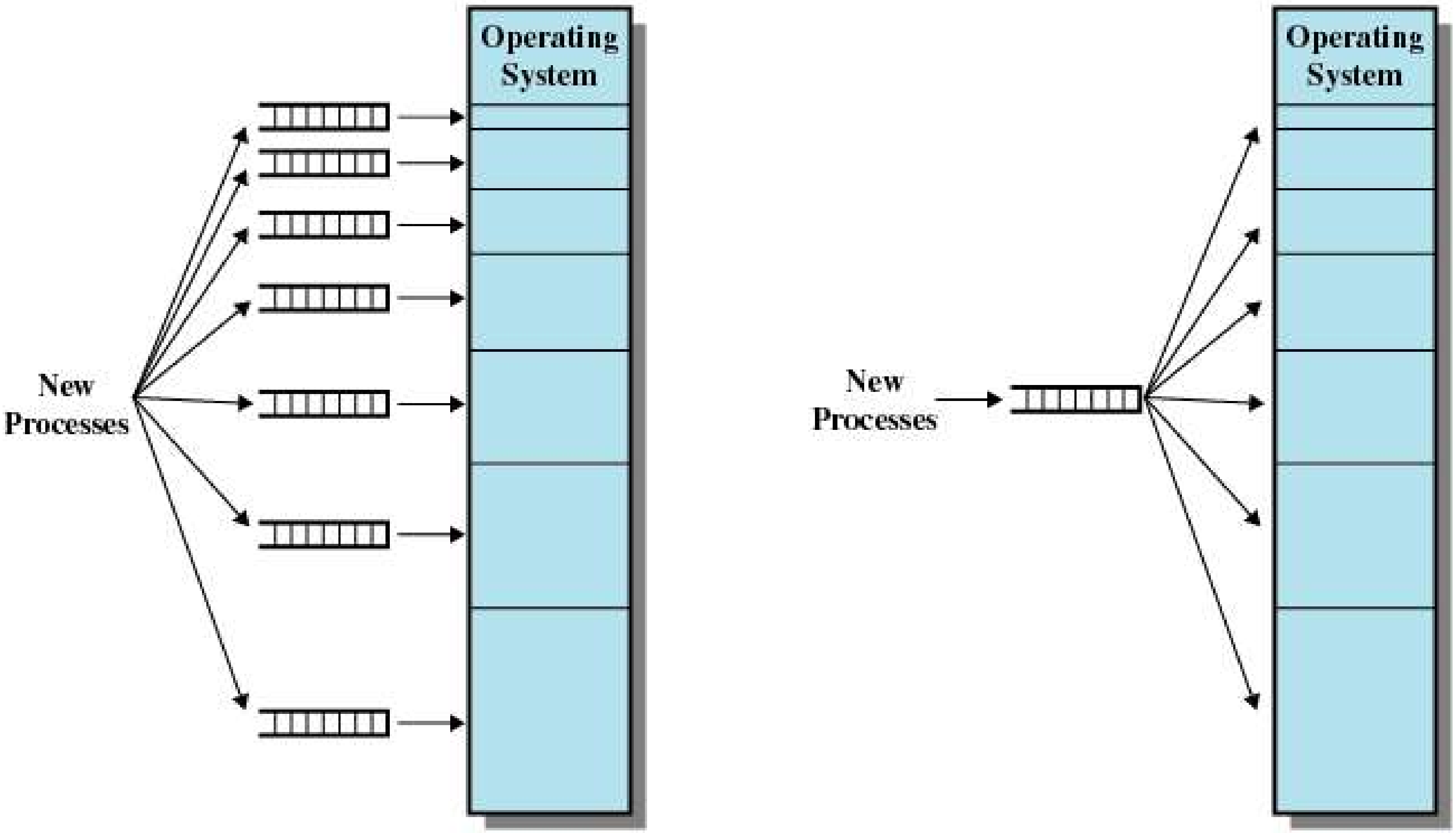
(a) Equal-size partitions

(b) Unequal-size partitions

Placement Algorithm with Partitions

- Equal-size partitions
  - Because all partitions are of equal size, it does not matter which partition is used
- Unequal-size partitions
  - Can assign each process to the smallest partition it will fit into
  - Queue for each partition
  - Processes are assigned in such a way as to minimize wasted memory within a partition

Placement Algorithm with Partitions



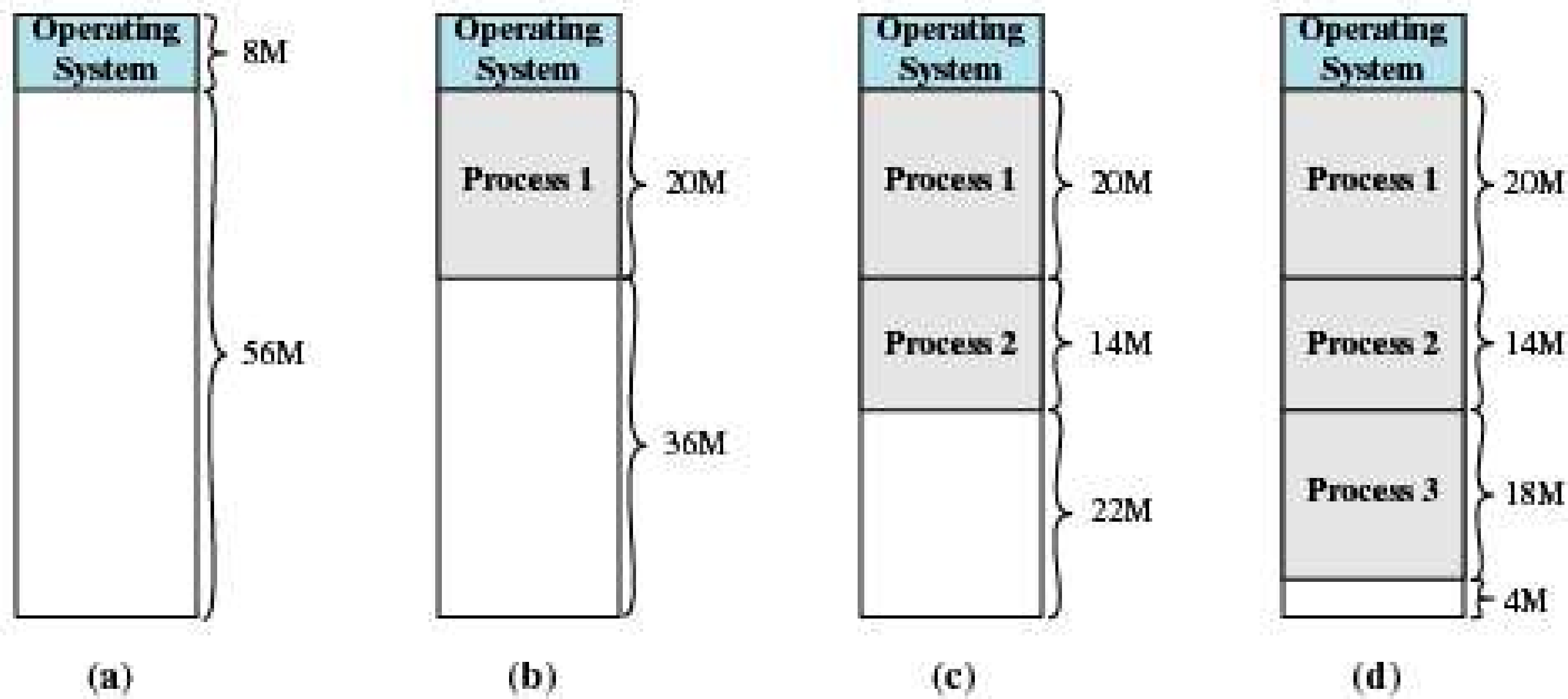
(a) One process queue per partition

(b) Single queue

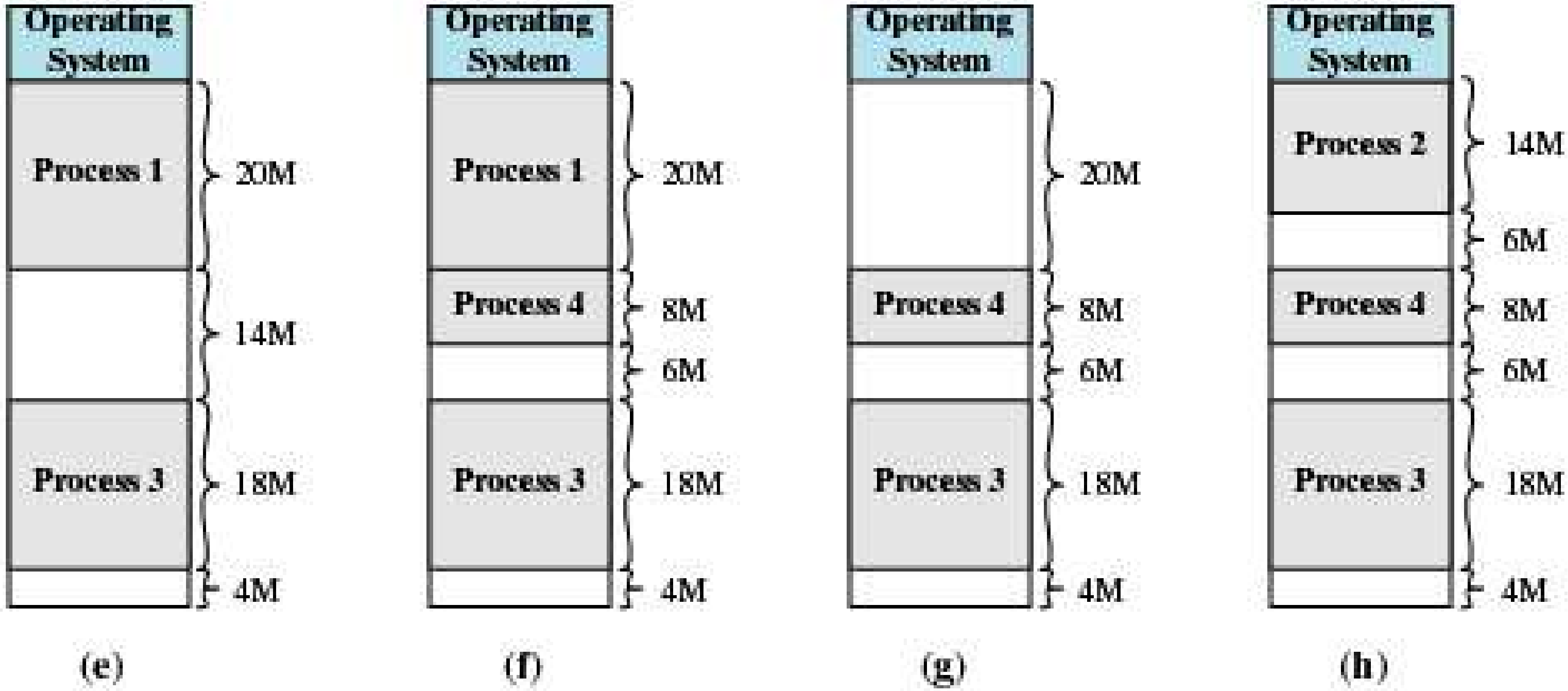
dynamic partitioning and compaction

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required
- Eventually get holes in the memory. This is called external fragmentation
- Must use **compaction** to shift processes so they are contiguous and all free memory is in one block
  - in the general case compaction may be unfeasible (C/C++ allocators)

external fragmentation



external fragmentation



Dynamic Partitioning Placement Algorithm

- Operating system must decide which free block to allocate to a process
- Best-fit algorithm
  - Chooses the block that is closest in size to the request
  - Worst performer overall
    - since smallest block is found for process, the smallest amount of fragmentation is left
  - Memory compaction must be done more often

Dynamic Partitioning Placement Algorithm

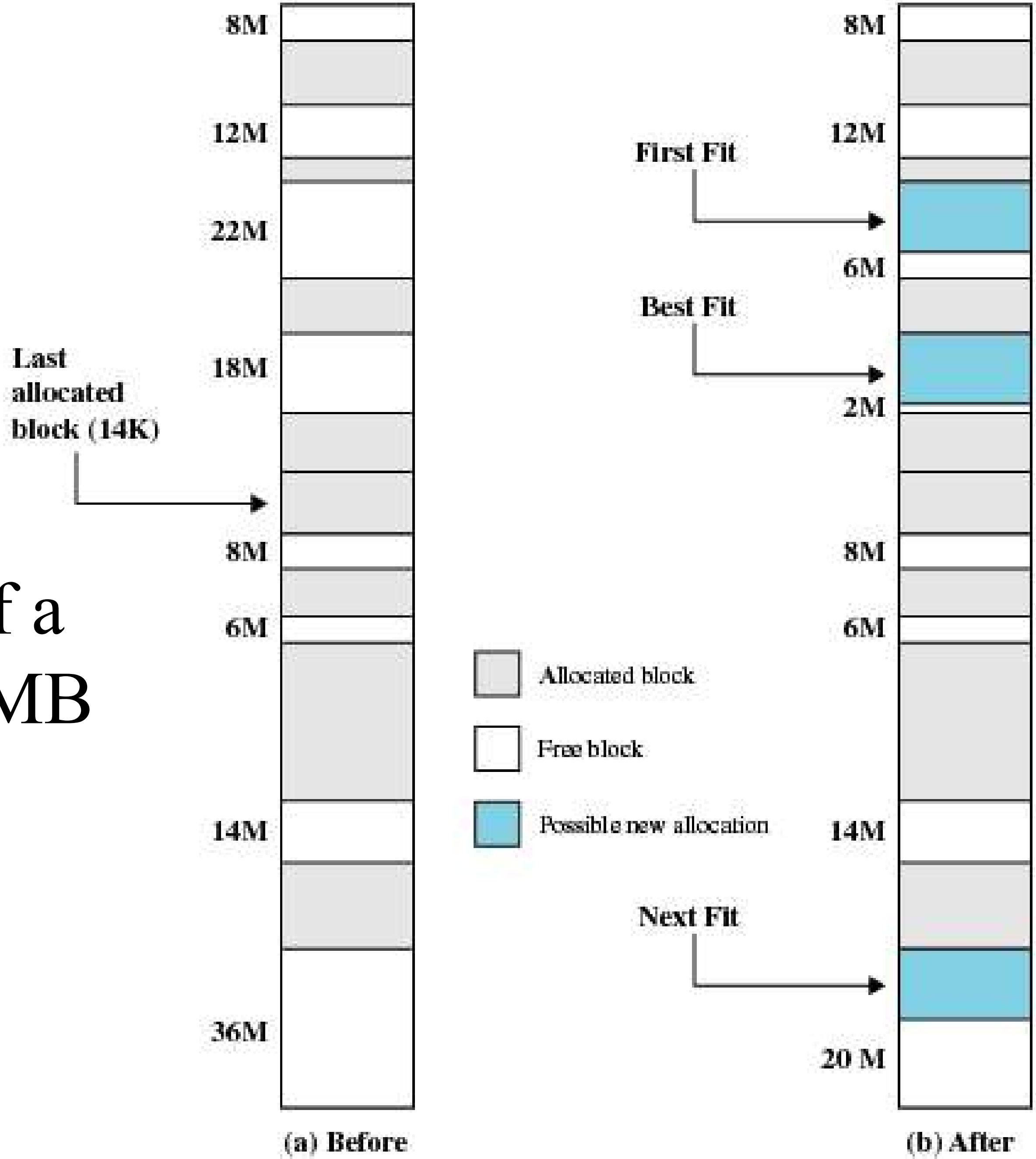
- First-fit algorithm
  - Scans memory form the beginning and chooses the first available block that is large enough
  - Fastest
  - May have many process loaded in the front end of memory that must be searched over when trying to find a free block

Dynamic Partitioning Placement Algorithm

- Next-fit
  - Scans memory from the location of the last placement
  - More often allocate a block of memory at the end of memory where the largest block is found
  - The largest block of memory is broken up into smaller blocks
  - Compaction is required to obtain a large block at the end of memory

examples

- allocation of a block of 16MB



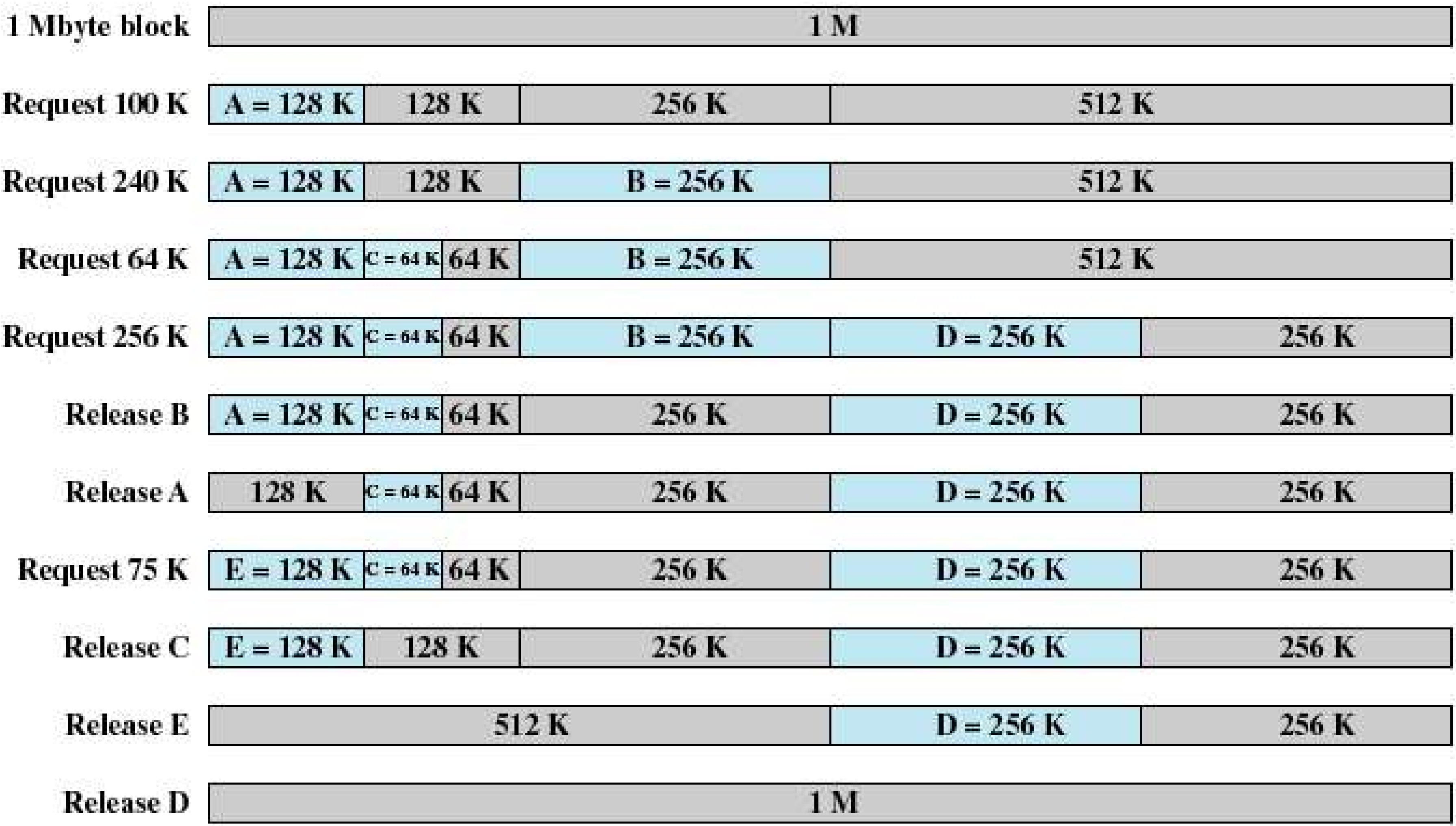
Buddy System

- entire space available is treated as a single block of 2^U
- a request of *s* bytes returns a block of ceil($\log_2 s$) bytes
  - if a request of size *s* such that $2^{i-1} < s <= 2^i$, a block of length 2^i is allocated
  - a 2^i block can be split into two equal **buddies** of 2^{i-1} *bytes*
  - for each request a "big" block is found and split until the smallest block greater than or equal to *s* is generated

Buddy System

- it maintains a lists L_i ($i=1..U$) of unallocated blocks (*holes*) of size 2^i
  - splitting: remove a hole from L_{i+1} split it, and put the two buddies it into L_i
  - coalescing: remove two unallocated buddies from L_i and put it into L_{i+1}

buddy system: example



Buddy System

procedure **get_hole**

input: i (precondition: $i \leq U$)

output: a block c of size 2^i (postcondition: L_i does not contain c)

if (L_i is empty)

b= get_hole(i+1);

< split b into two buddies b_1 and b_2>

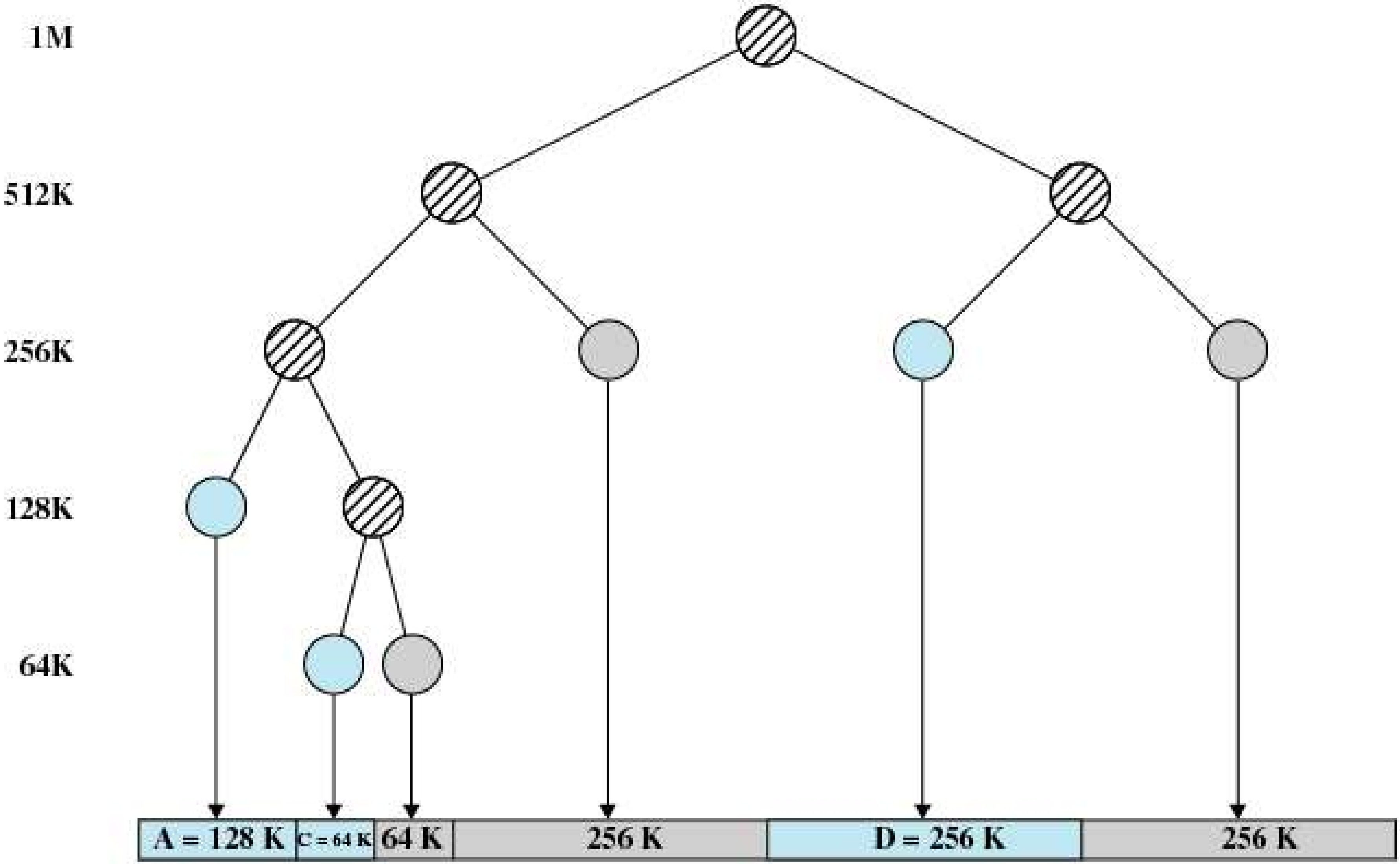
< put b_1 and b_2 into L_i>

c= < first hole in L_i>

<remove c form L_i>

return c

buddy system: tree representation



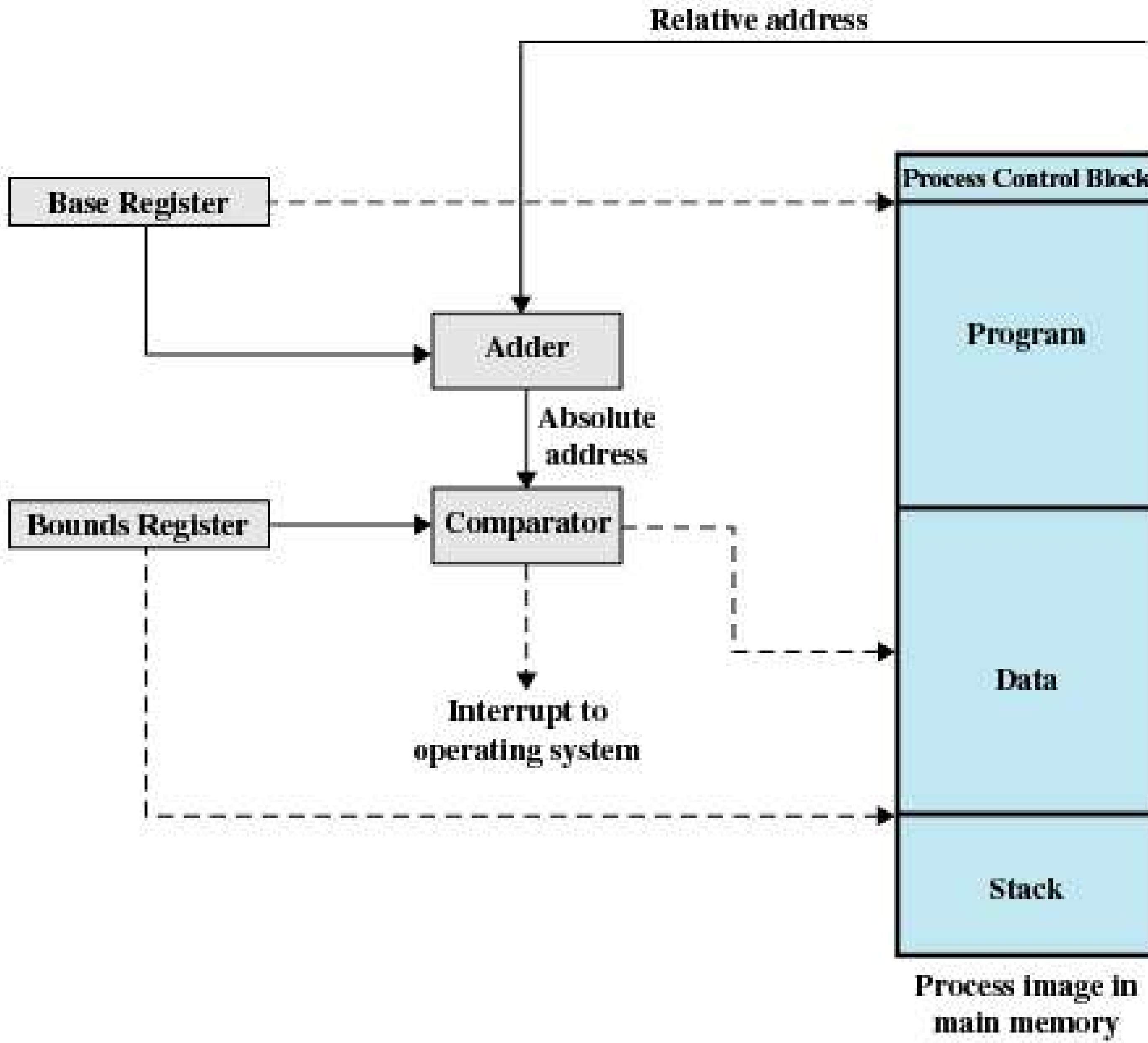
relocation

- a process may occupy different partitions, which means different absolute memory locations
- when program load into memory the absolute memory locations are determined
  - different execution may lead to different locations
- on-the-fly relocation during execution
  - swap out and swap in
  - compaction of allocated partitions

addresses in the program

- Physical
  - The absolute address or actual location in main memory
- Logical
  - Reference to a location in a “logical” memory independent of the current assignment of data to memory
  - Translation must be made to the physical address by the hardware (MMU)
- Relative (logical or physical)
  - Address expressed as a location relative to some known point

hardware support for relocation



Registers Used during Execution

- Base register
  - Starting address for the process
- Bounds register
  - Ending location of the process
- These values are set when the process is loaded or when the process is swapped in

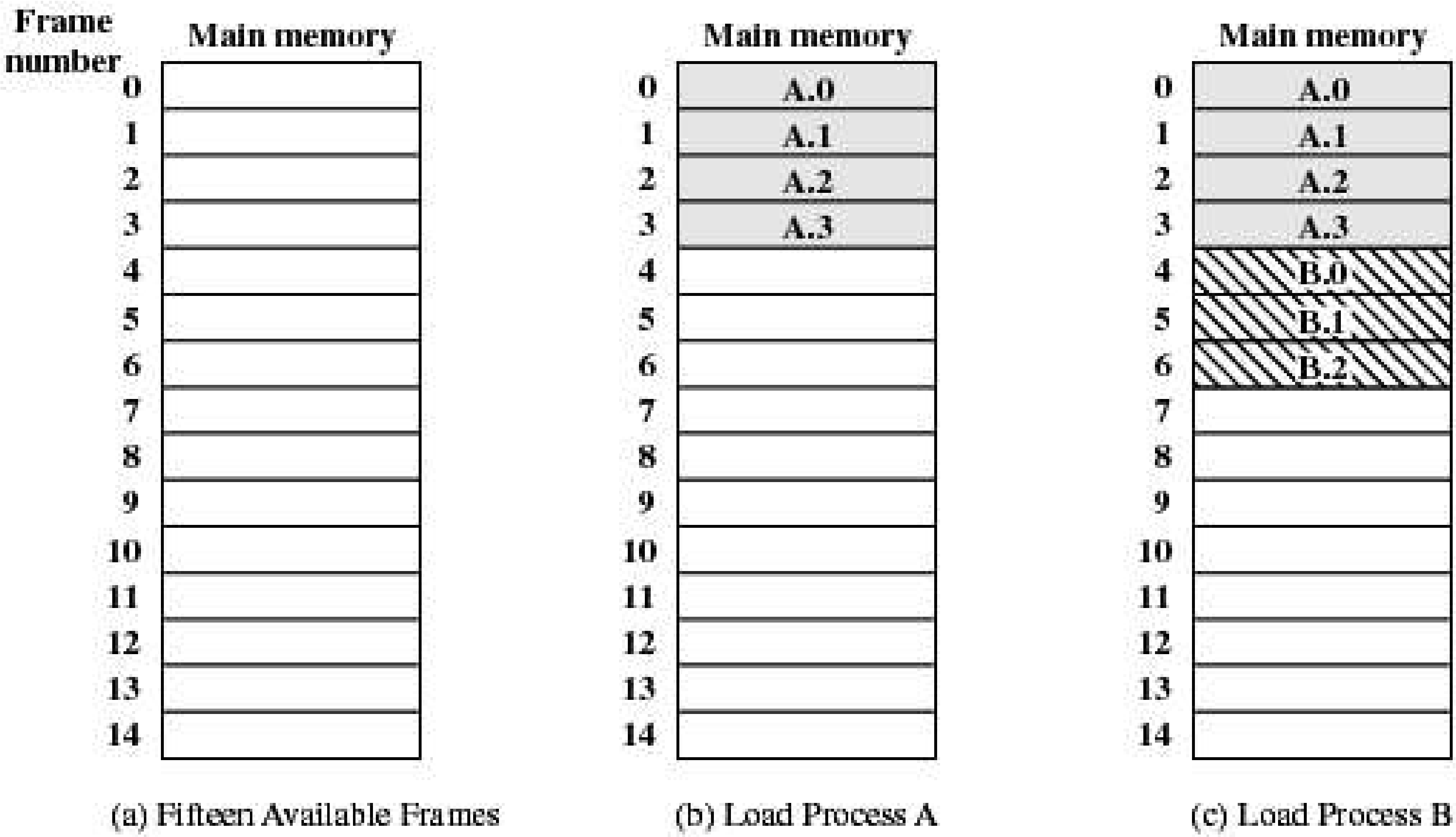
Registers Used during Execution

- The value of the base register is added to a relative address to produce an absolute address
- The resulting address is compared with the value in the bounds register
- If the address is not within bounds, an interrupt is generated to the operating system

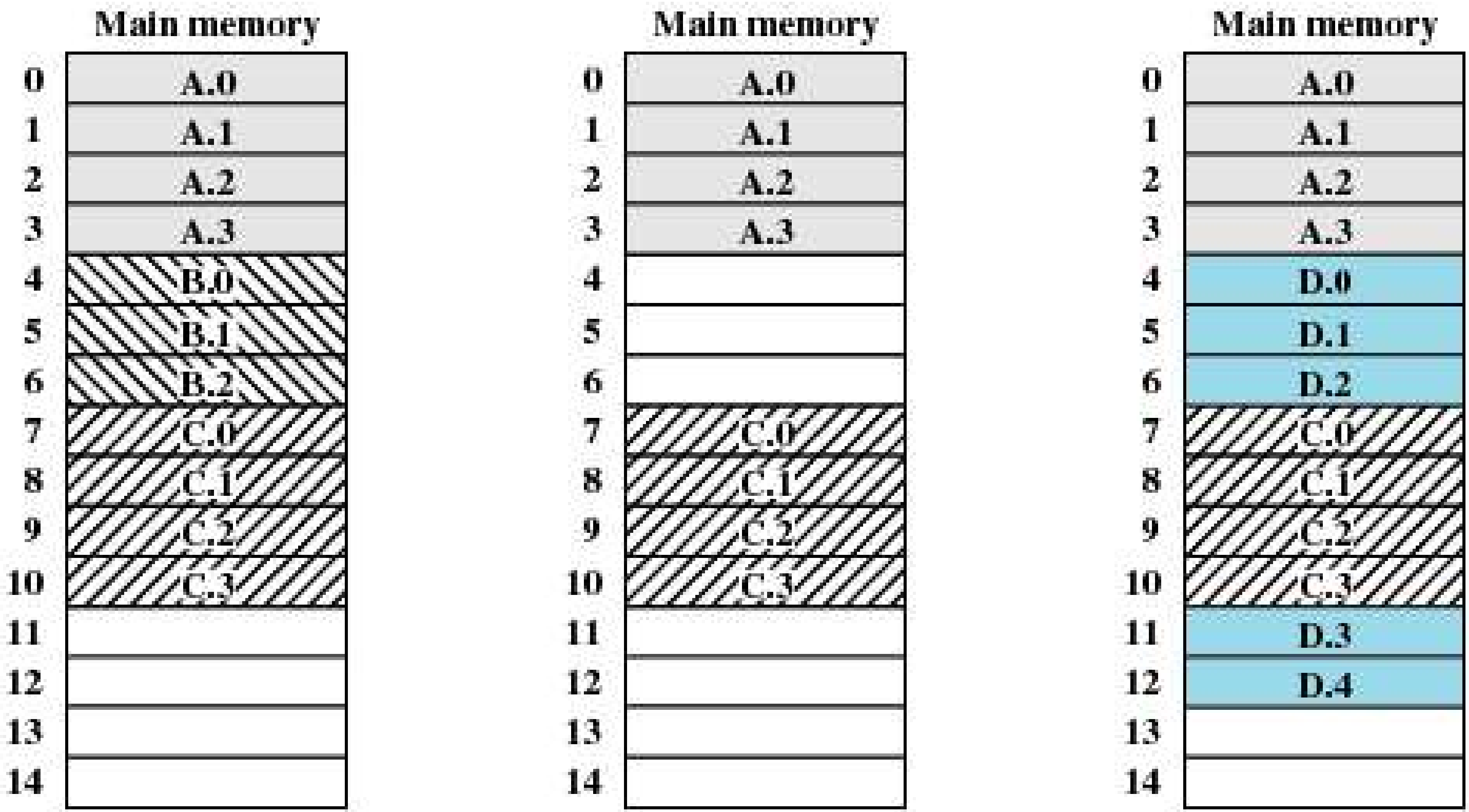
Paging

- Partition memory into small equal fixed-size chunks and divide each process into the same size chunks
- The chunks of a process are called **pages** and chunks of memory are called **frames**
- Operating system maintains a **page table** for each process
  - Contains the **frame location for each page** in the process
  - **Memory address consist of a page number and offset within the page**

Assignment of Process Pages to Free Frames



Assignment of Process Pages to Free Frames

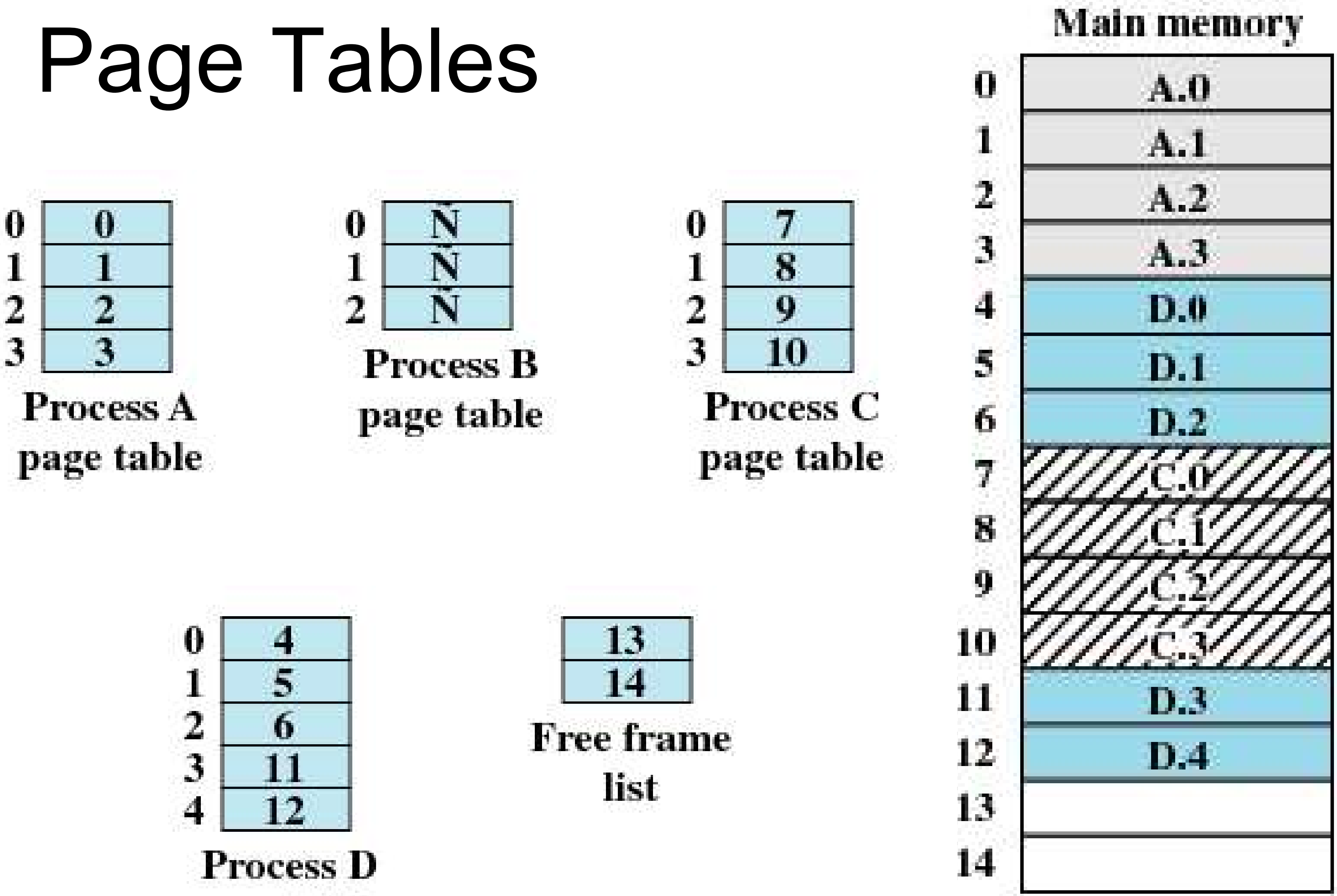


(d) Load Process C

(e) Swap out B

(f) Load Process D

Page Tables

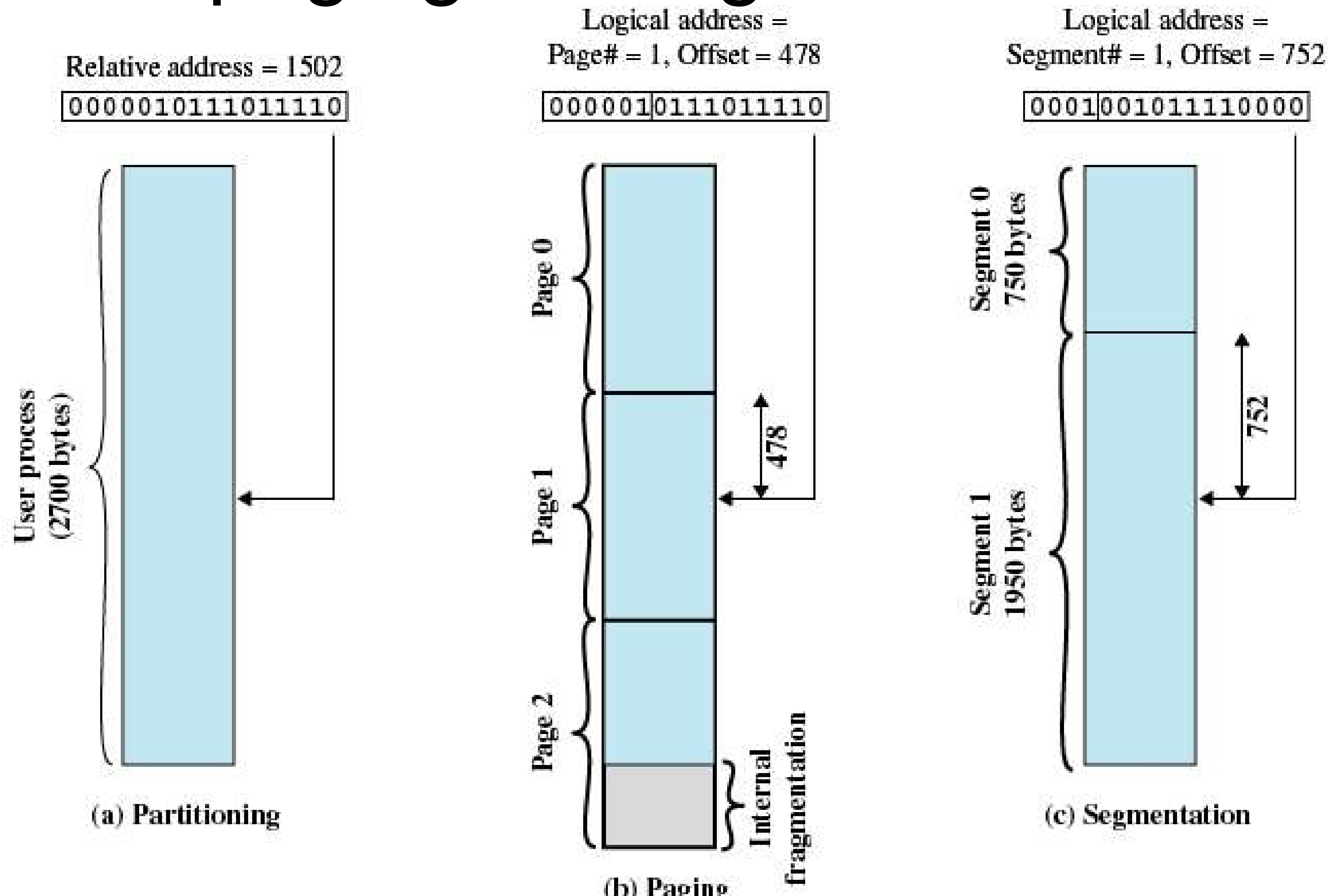


(f) Load Process D

Segmentation

- All segments of all programs do **not** have to be of the **same length**
- There is a maximum segment length
- Addressing consist of two parts - a **segment number and an offset**
- Since segments are not equal, segmentation is similar to dynamic partitioning

paging vs. segmentation



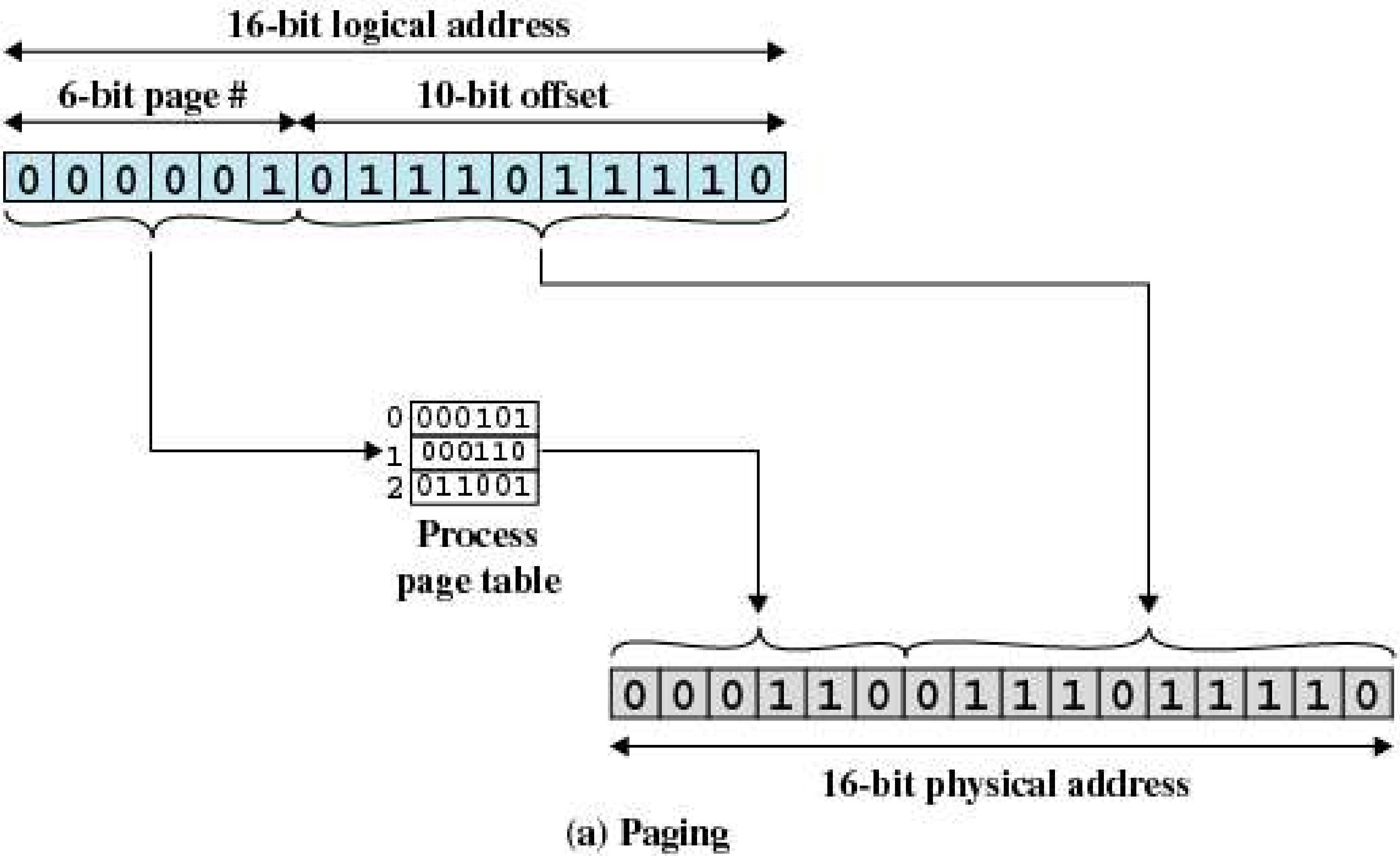
(a) Partitioning

(b) Paging (page size = 1K)

(c) Segmentation

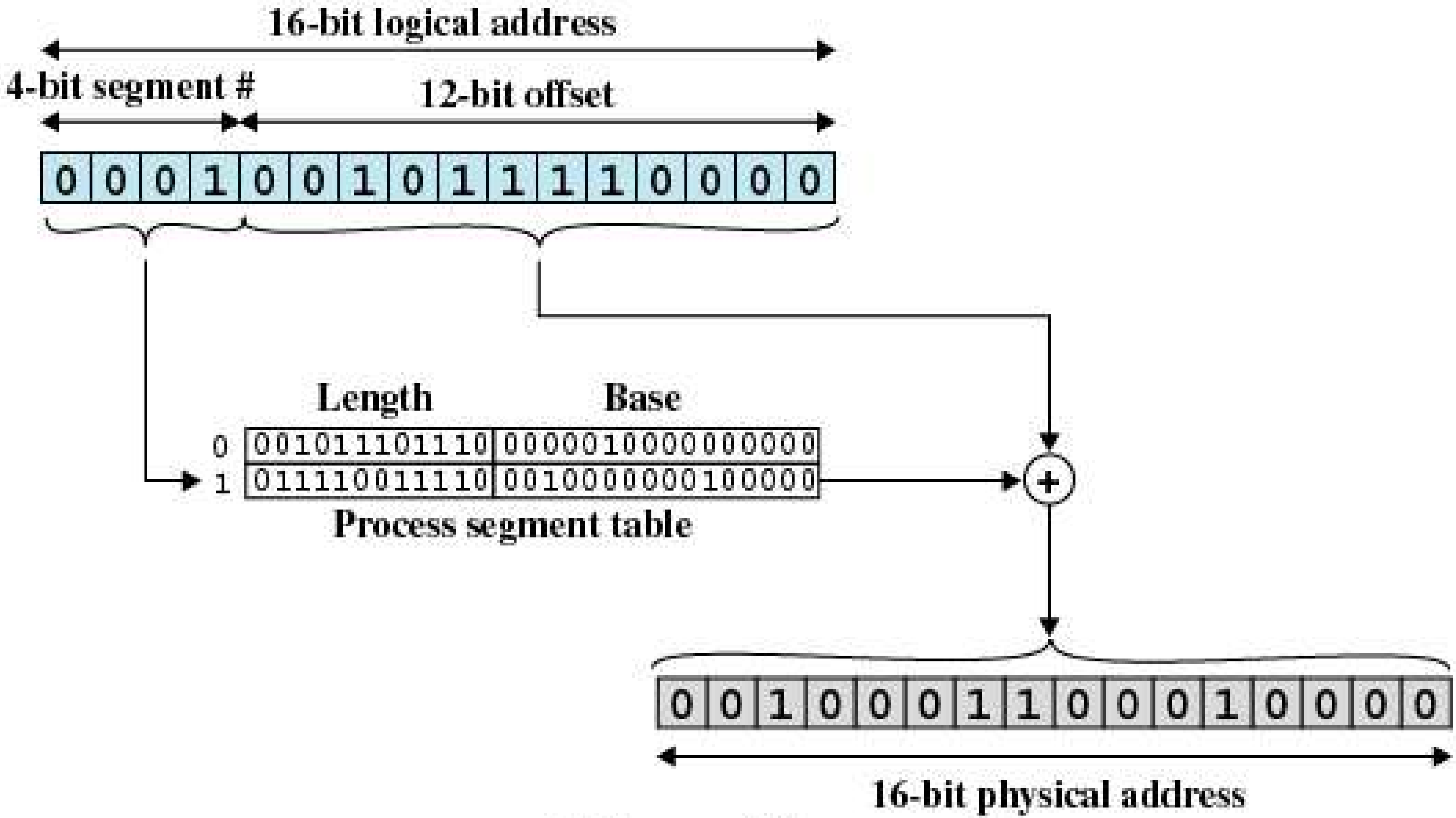
logical to physical translation

paging



logical to physical translation

segmentation



(b) Segmentation