MONASH University

MONASH
INFORMATION
TECHNOLOGY

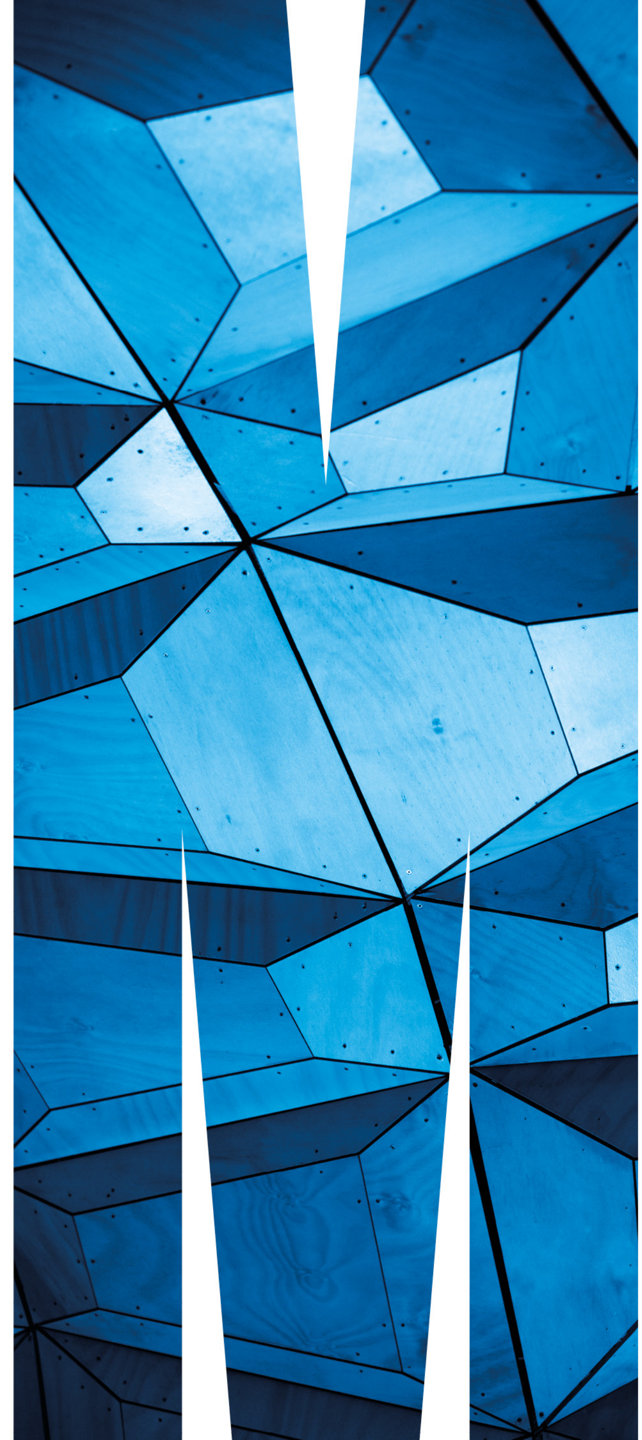
FIT2100 Semester 2 2017

Lecture 8: Memory Management

(Reading: Stallings, Chapter 7)

Jojo Wong

GROUP OF EIGHT AUSTRALIA



Lecture 8: Learning Outcomes

- Upon the completion of this lecture, you should be able to:
  - Discuss the principal requirements for memory management
  - Explain various memory partitioning techniques and placement algorithms
  - Discuss the concepts of paging and segmentation

Why memory management is important?

Memory Management

- Main memory is *divided* into two parts:
  - One part for OS (kernel)
  - One part for the program currently running

Uni-programming environment

- Multiprogramming:
  - "User" part of memory is further *subdivided* to accomodate multiple processes

Managed by OS

Memory Management: Terminology

A frame is of the same size as a page

Frame	A fixed-length block of main memory.
Page	A fixed-length block of data that resides in secondary memory (such as disk). A page of data may temporarily be copied into a frame of main memory.
Segment	A variable-length block of data that resides in secondary memory. An entire segment may temporarily be copied into an available region of main memory (segmentation) or the segment may be divided into pages which can be individually copied into main memory (combined segmentation and paging).

What are the requirements for memory management?

Memory Management: Requirements

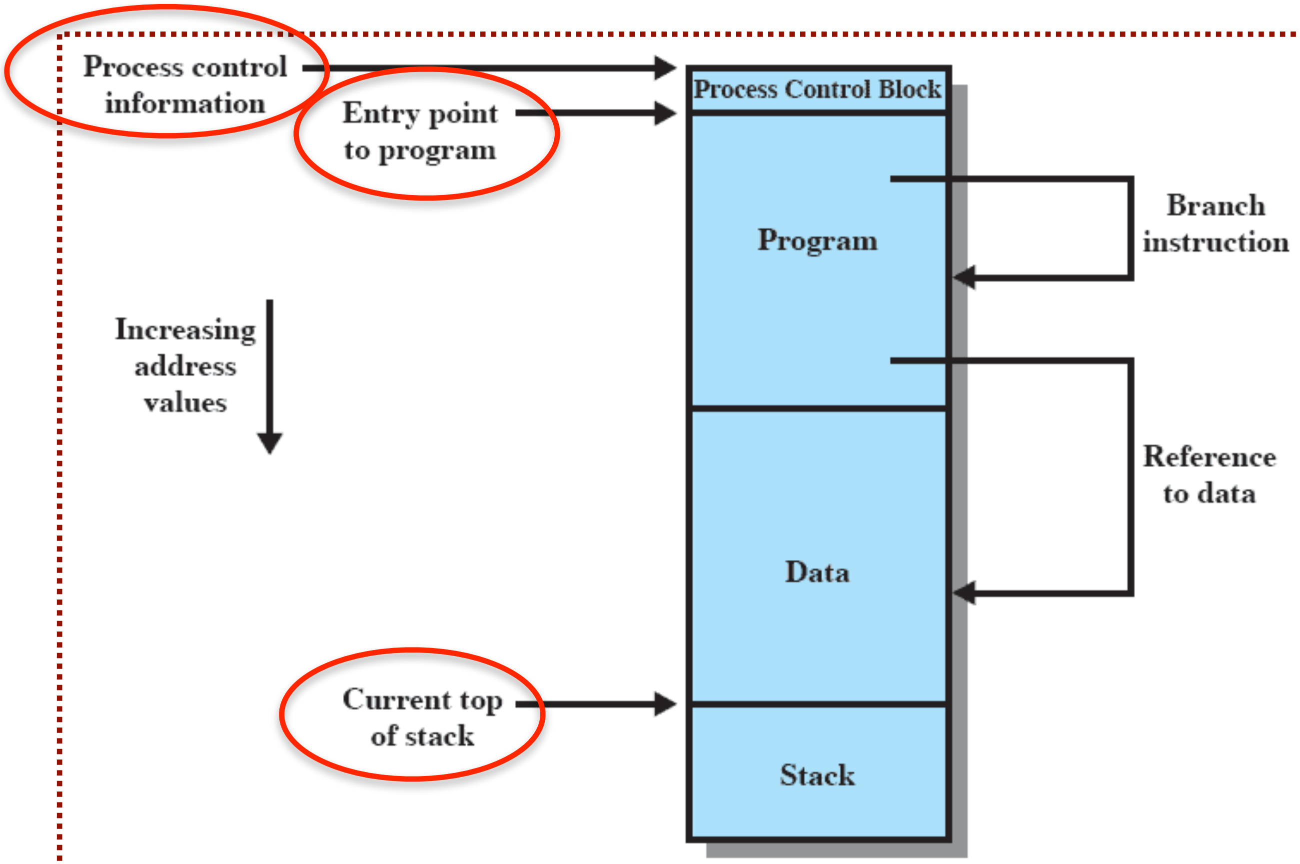
- Memory management is intended to satisfy the following requirements:
  - Relocation
  - Protection
  - Sharing
  - Logical organisation
  - Physical organisation

In order to support multitasking or multiprogramming systems

Requirements: Relocation

- Programmers typically do not know in advance which other programs will be resident in main memory at the time of execution of their program
- Active processes need to be able to be swapped in and out of main memory in order to maximise processor utilisation
- Specifying that a process must be placed in the same memory region when it is swapped back in would be limiting
  - May need to relocate the process to a different area of memory

Requirements: Addressing



Process Image

Requirements: Protection

- Processes need to acquire permission to reference memory locations for reading or writing purposes
- Location of a program in main memory is unpredictable
- Memory references generated by a process must be checked at run time

By processor instead of OS

- Mechanisms that support relocation also support protection

Requirements: Sharing

For a number of processes executing a same program

- Advantageous to allow each process access to the same copy of the program rather than have their own separate copy
- Memory management must allow controlled access to shared areas of memory without compromising protection

For co-operating processes

- Mechanisms that support relocation also support sharing capabilities

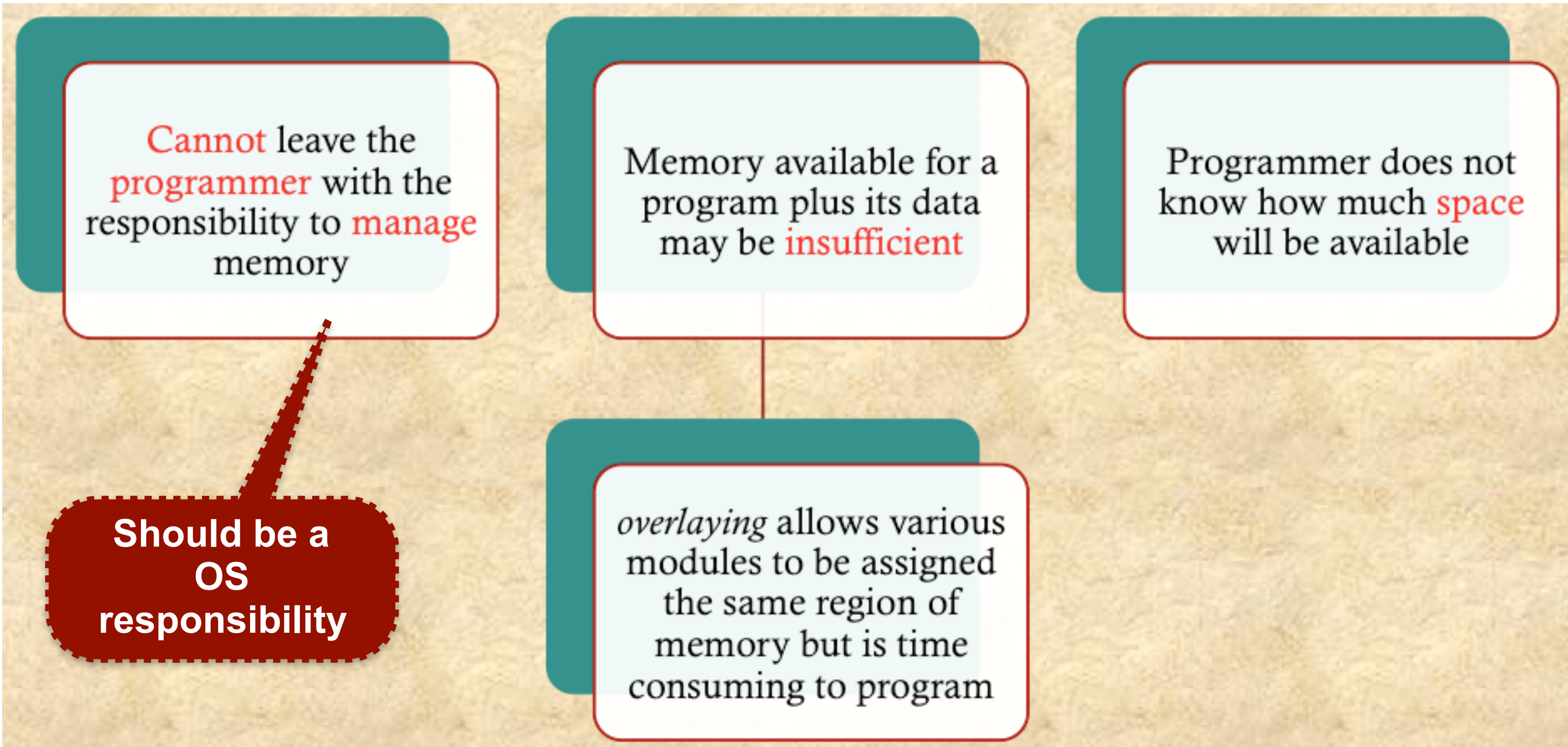
Requirements: Logical Organisation

- Memory (main and secondary) is typically organised as linear

Programs are written in modules
• modules can be written and compiled independently • different degrees of protection given to modules (read-only, execute-only) • sharing on a module level corresponds to the user's way of viewing the problem

- Segmentation: the tool that most readily satisfies requirements

Requirements: Physical Organisation



What is memory partitioning?

Memory Partitioning

- Memory management brings processes into main memory for execution by the processor
  - Involves **virtual memory**
  - Based on segmentation and paging

Next week's topic

- **Partitioning**
  - Used in several variations in some now-obsolete operating systems
  - Does not involve virtual memory

Memory Management: Techniques

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.
Virtual Memory Paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual Memory Segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

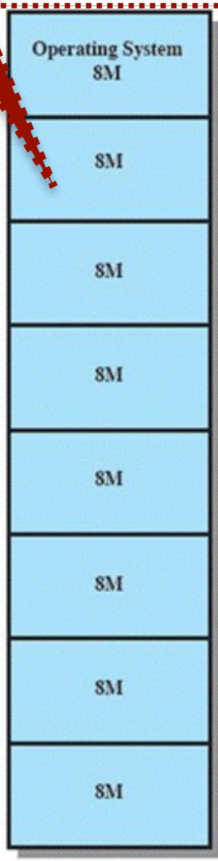
MONASH University

What are the key differences between fixed partitioning and dynamic partitioning?

Fixed Partitioning

wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition — internal fragmentation

- Equal-size partitions
- Any process whose size is *less than or equal to the partition size* can be loaded into an available partition
- OS can swap out a process if all partitions are full and no process is in the Ready or Running state



(a) Equal-size partitions

Fixed Partitioning: Disadvantages

- A program may be too big to fit in a partition
  - Need to be designed with the use of overlays

The programmer must design the program with the use of *overlays* so that only a portion of the program need be in main memory at any one time.

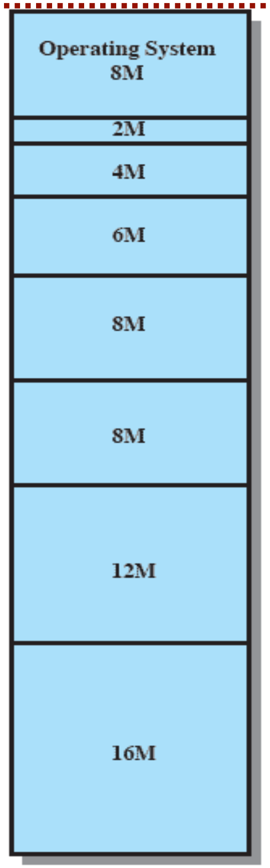
Fixed Partitioning: Disadvantages

- A program may be too big to fit in a partition
  - Need to be designed with the use of overlays

- Main memory utilisation is extremely inefficient
  - Any program, regardless of size, occupies an entire partition
  - Internal fragmentation: wasted space due to the block of data loaded being smaller than the partition

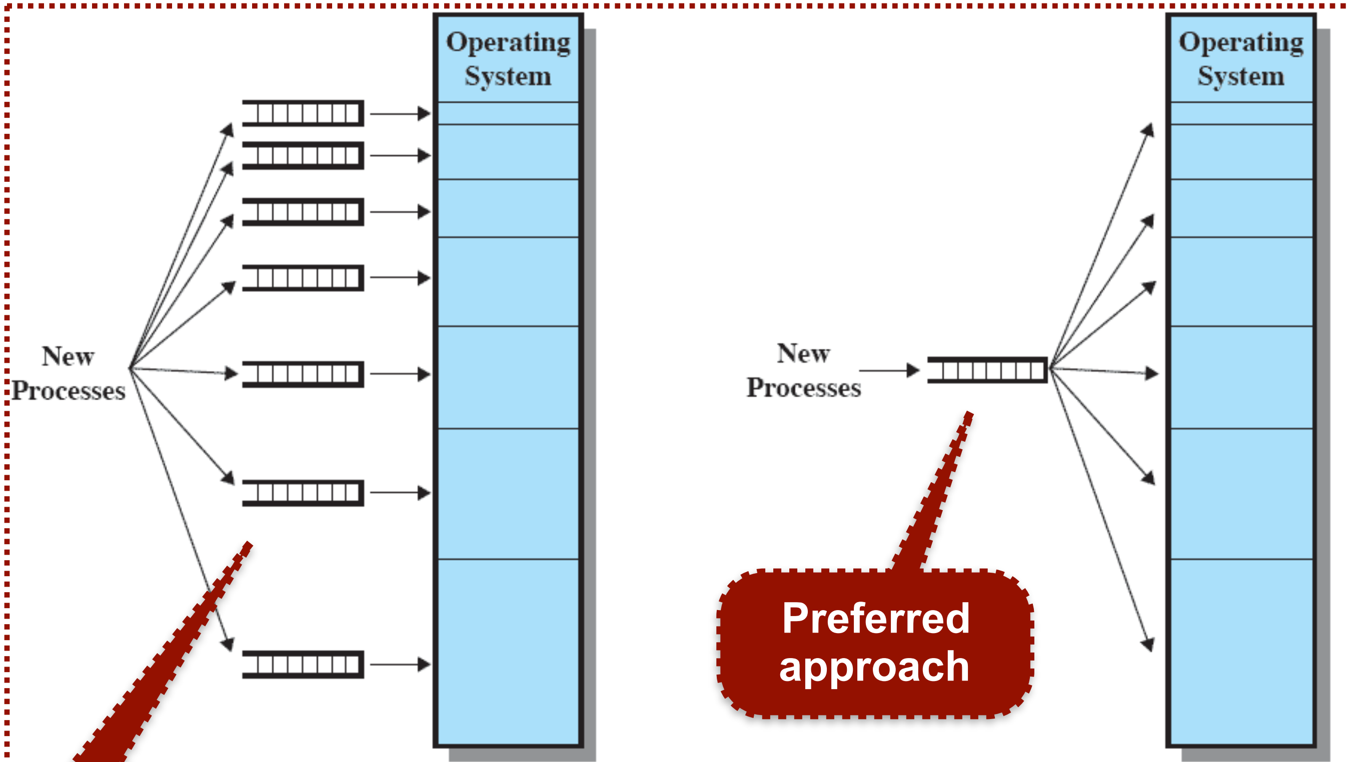
Fixed Partitioning: Unequal Size Partitions

- Unequal size partitions helps lessen the problems
- Example: a programs up to 16M can be accommodated *without overlays*
- Partitions smaller than 8M allow smaller programs to be accommodated with less internal fragmentation



(b) Unequal-size partitions

Fixed Partitioning: Unequal Size Partitions



Preferred approach

Minimise wasted memory within each partition

(a) One process queue per partition

(b) Single queue

Memory Assignment

Fixed Partitioning: Disadvantages

- The number of partitions specified at system generation time limits the number of active processes in the system

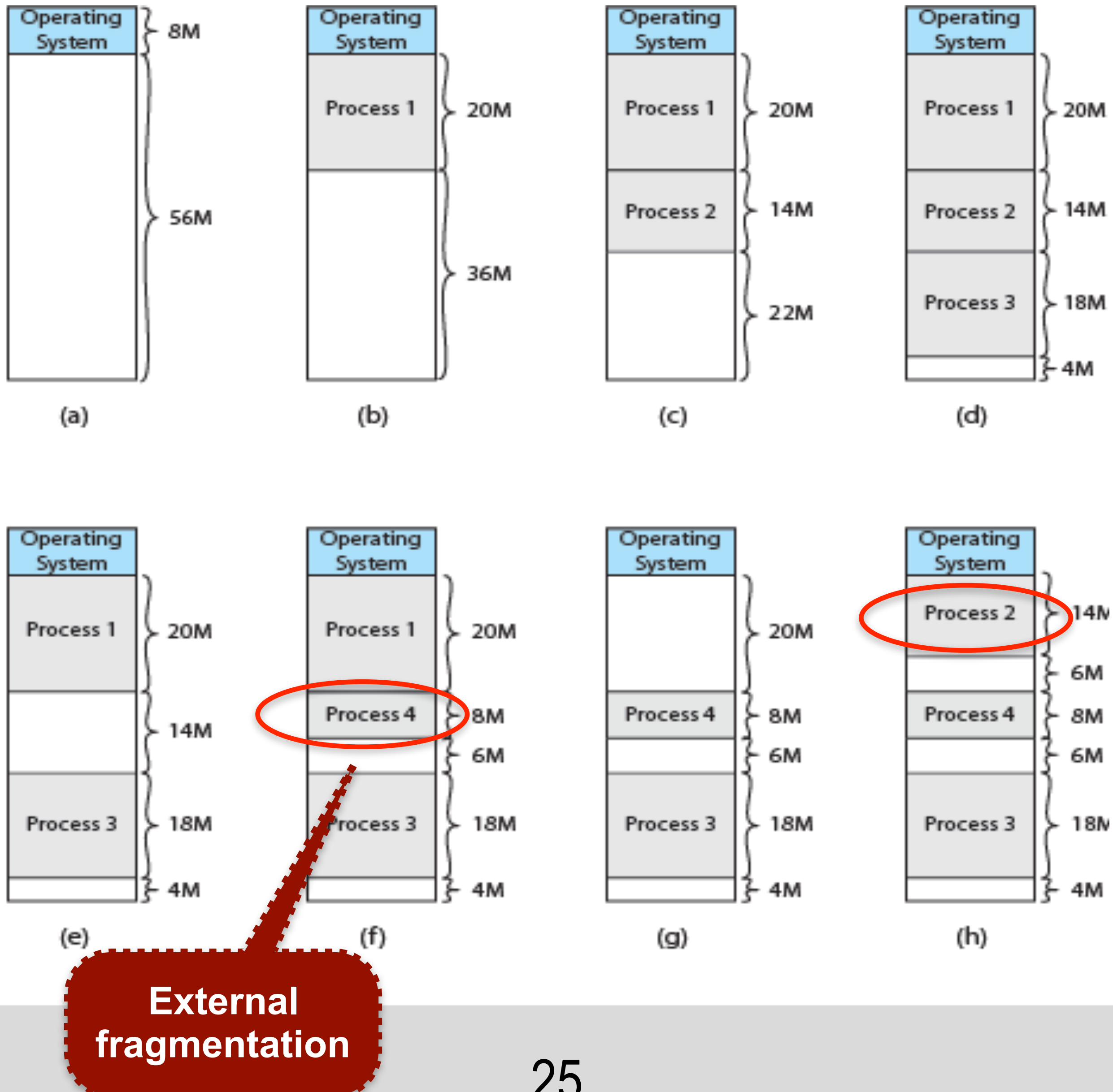
- Small jobs will not utilise fixed partition space efficiently

Since partition sizes are preset at system generation time

Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as it requires
- Used by the IBM's mainframe operating system, OS/MVT — *Multiprogramming with a Variable Number of Tasks*

Dynamic Partitioning: The Effect



Dynamic Partitioning

- External fragmentation:
  - Memory becomes more and more fragmented
  - Memory utilisation declines

A need to support dynamic relocation

- Compaction:
  - Technique to overcome external fragmentation
  - OS shifts processes such that they are contiguous — free memory is pooled together in one block
  - Time consuming and wastes CPU time

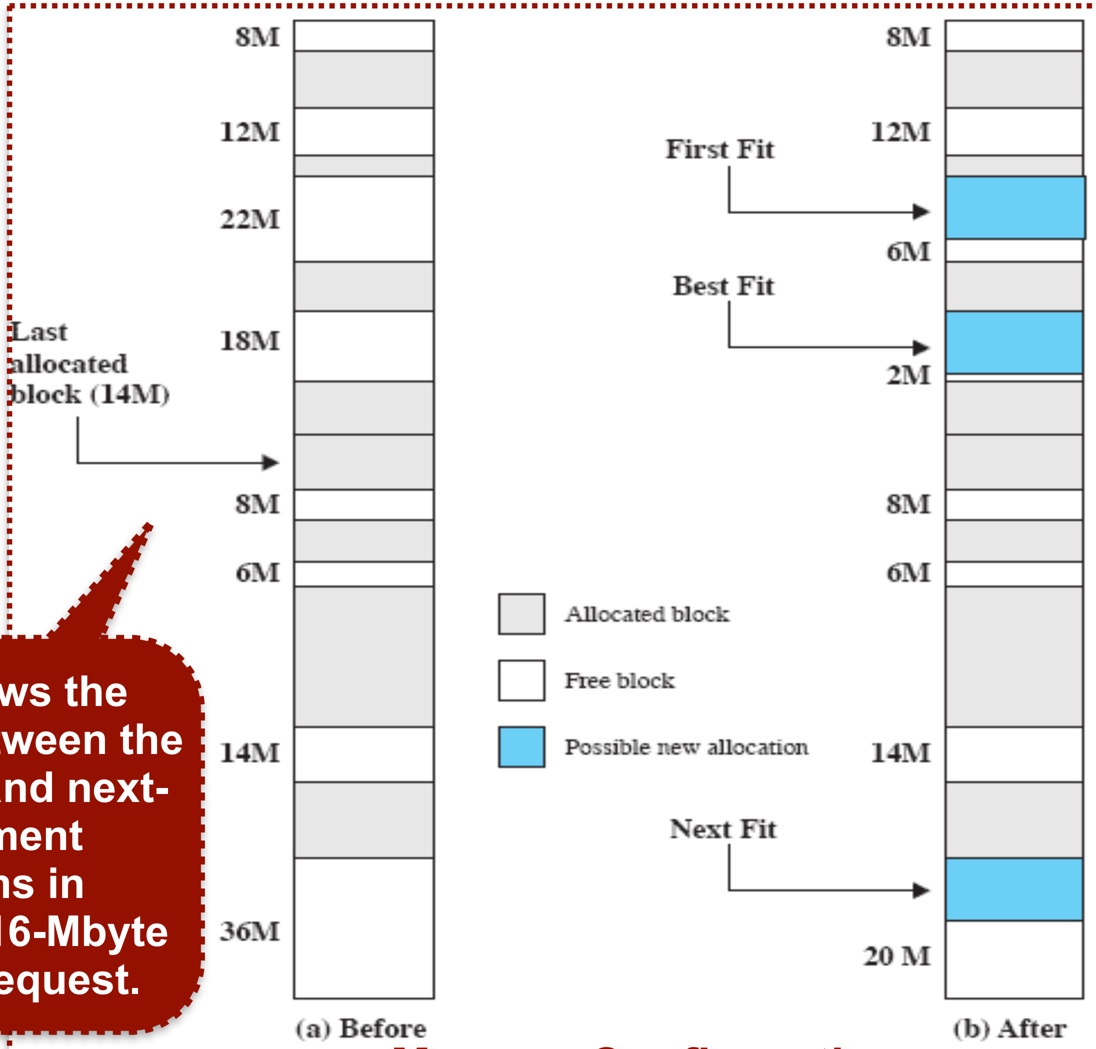
Dynamic Partitioning: Placement Algorithms

- Given that memory compaction is time consuming — OS designers must be clever in deciding how to assign processes to memory
  - How to plug the holes?
- OS must decide which free block to allocate:
  - When it is time to load or swap a process into main memory
  - If there is more than one free block of memory of sufficient size

Dynamic Partitioning: Placement Algorithms

Best-fit	First-fit	Next-fit
• chooses the block that is **closest in size** to the request	• begins to **scan memory from the beginning** and chooses the first available block that is large enough	• begins to **scan memory from the location of the last placement** and chooses the next available block that is large enough

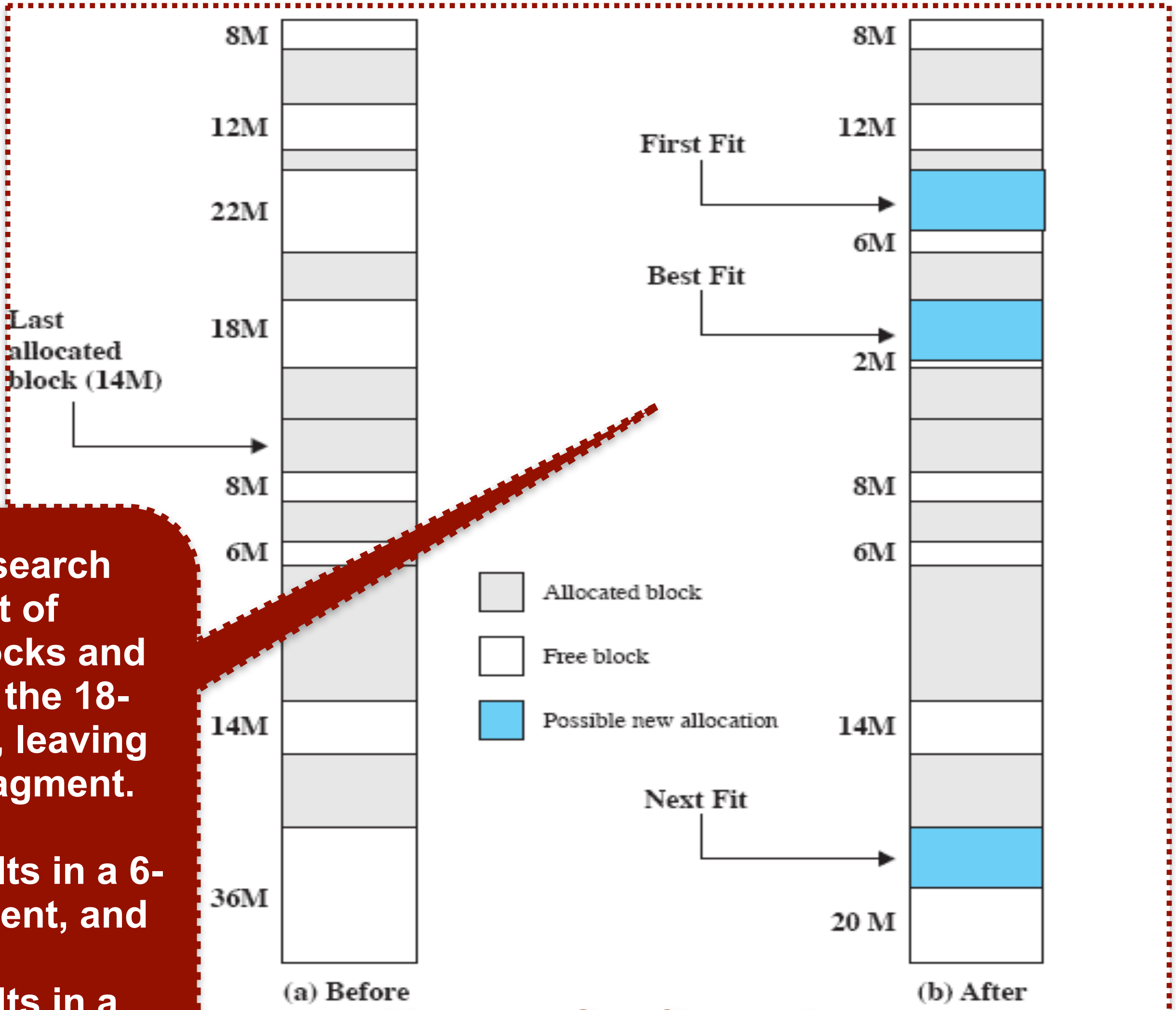
Dynamic Partitioning: Placement Algorithms



Memory Configuration

Figure shows the difference between the best-, first-, and next-fit placement algorithms in satisfying a 16-Mbyte allocation request.

Dynamic Partitioning: Placement Algorithms



Memory Configuration

- Best-fit will search the entire list of available blocks and make use of the 18-Mbyte block, leaving a 2-Mbyte fragment.
- First-fit results in a 6-Mbyte fragment, and
- Next-fit results in a 20-Mbyte fragment.

Dynamic Partitioning: Placement Algorithms

- **First-fit** algorithm: the *simplest* and the *best* and *fastest* as well
- **Next-fit** algorithm: tends to produce slight worse results than first-fit
  - The largest block of free memory at the end of the memory space is broken up into small fragments

Compaction needed more frequently

- **Best-fit** algorithm: the *worst* performer
  - Search through the memory space for the smallest block

What is the buddy system?

Buddy System

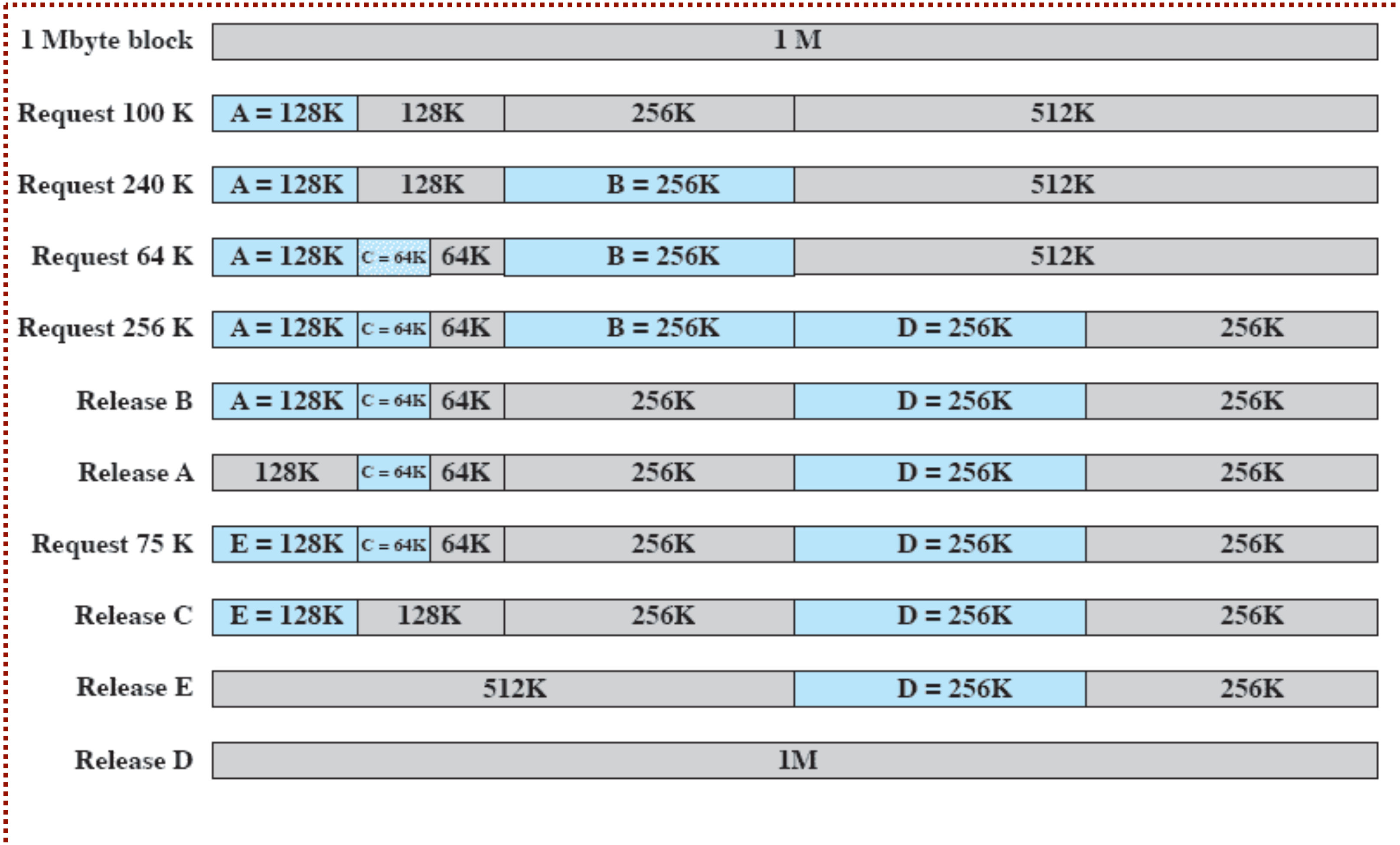
Useful for parallel systems (Unix kernel)

- Comprised of fixed and dynamic partitioning schemes
- Space available for allocation is treated as a single block
- Memory blocks are available of size 2^K words — $L \leq K \leq U$, where
  - 2^L = smallest size block that is allocated
  - 2^U = largest size block that is allocated; generally 2^U is the size of the entire memory available for allocation

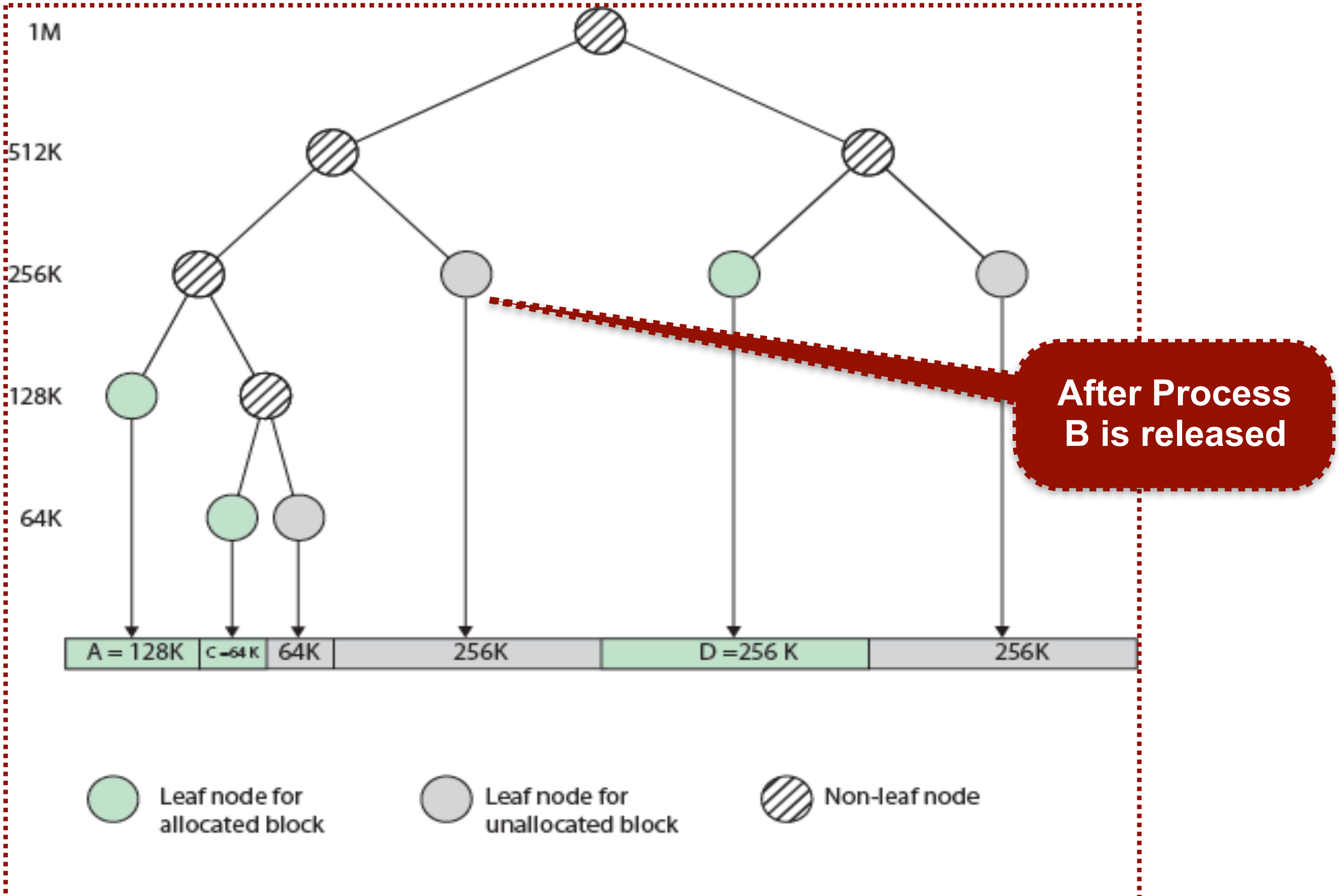
Buddy System: Basic Algorithm

- The entire space available for allocation is treated as a single block of size 2^U
- If a request of size s with $2^{U-1} < s \le 2^U$ is made, then the entire block is allocated
  - Otherwise, the block is split into two equal buddies of size 2^{U-1}
- If $2^{U-2} < s \le 2^{U-1}$, then the request is allocated to one of the two buddies of size 2^{U-1}

Buddy System: Example



Buddy System: Tree Representation

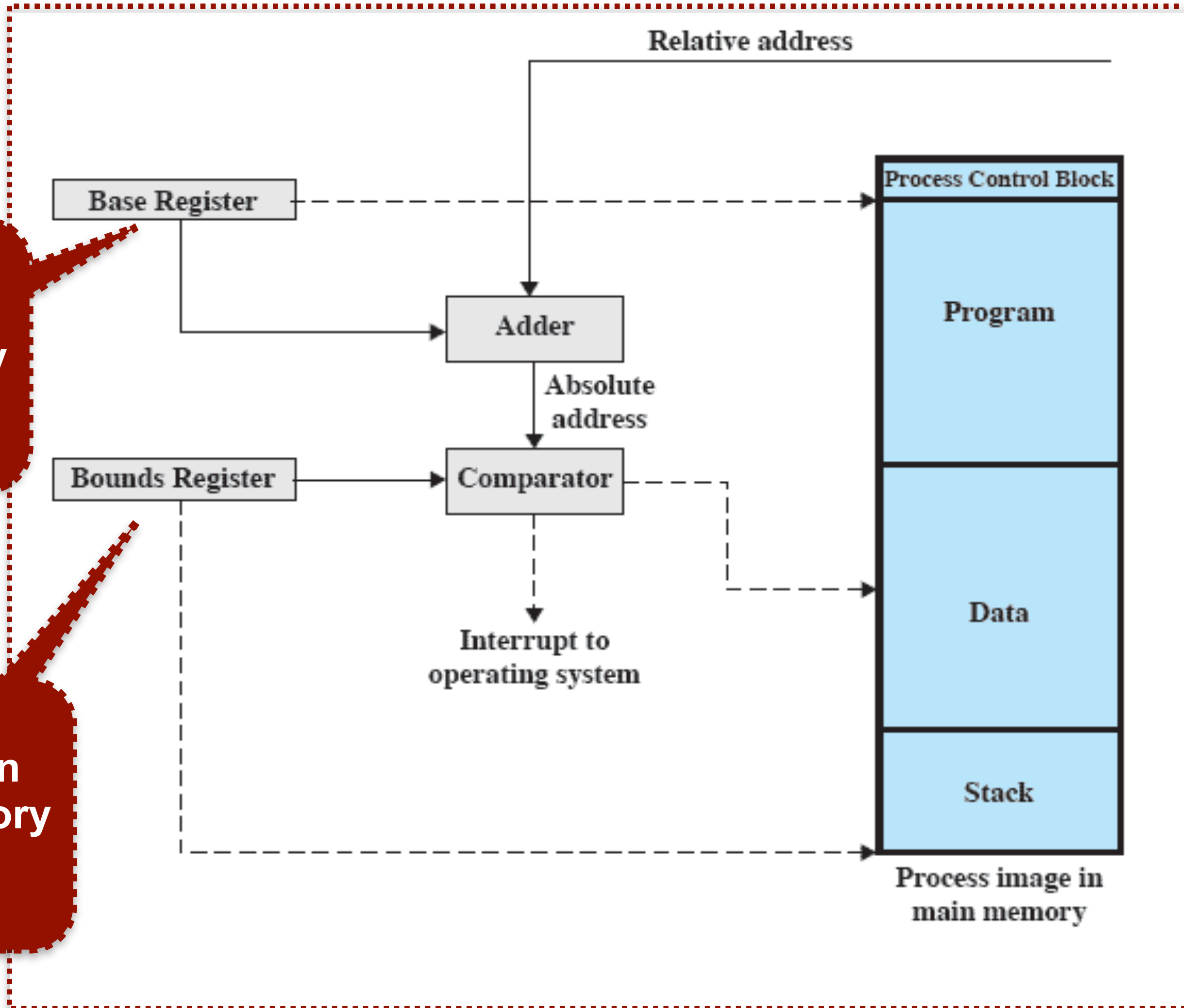


Aside: Types of Addresses

Must be translated to physical address

- Logical address:
  - Reference to a memory location independent of the current assignment of data to memory
- Relative address:
  - Address is expressed as a location related to some known location
- Physical (absolute) address:
  - Actual location in main memory

Aside: Address Translation



starting address in main memory of the program

ending address in main memory of the program

What is the paging technique?

Paging: Basic Concepts

- Both unequal fixed-size and variable-size partitions are inefficient in the use of memory — the former results in internal fragmentation; the latter in external fragmentation

- Paging:
  - Partition memory into equal fixed-size chunks that are relatively small
  - Process is also divided into small fixed-size chunks of the same size

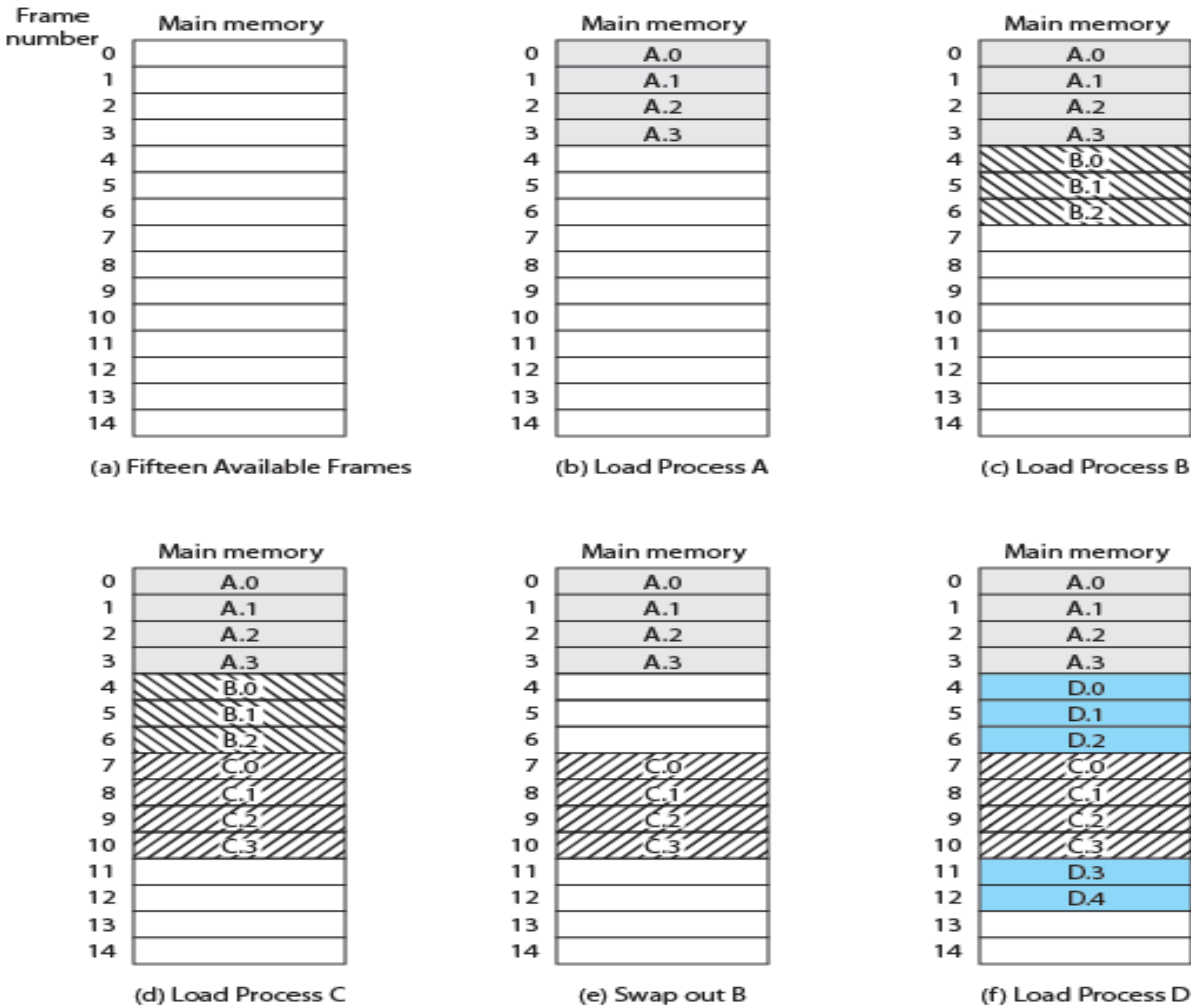
Paging: Basic Concepts

No external fragmentation but with a small internal fragmentation

- The chunks of a process, known as pages could be assigned to available chunks of memory, known as frames or page frames

Pages	*Frames*
• chunks of a **process**	• available chunks of **memory**

Paging: Example

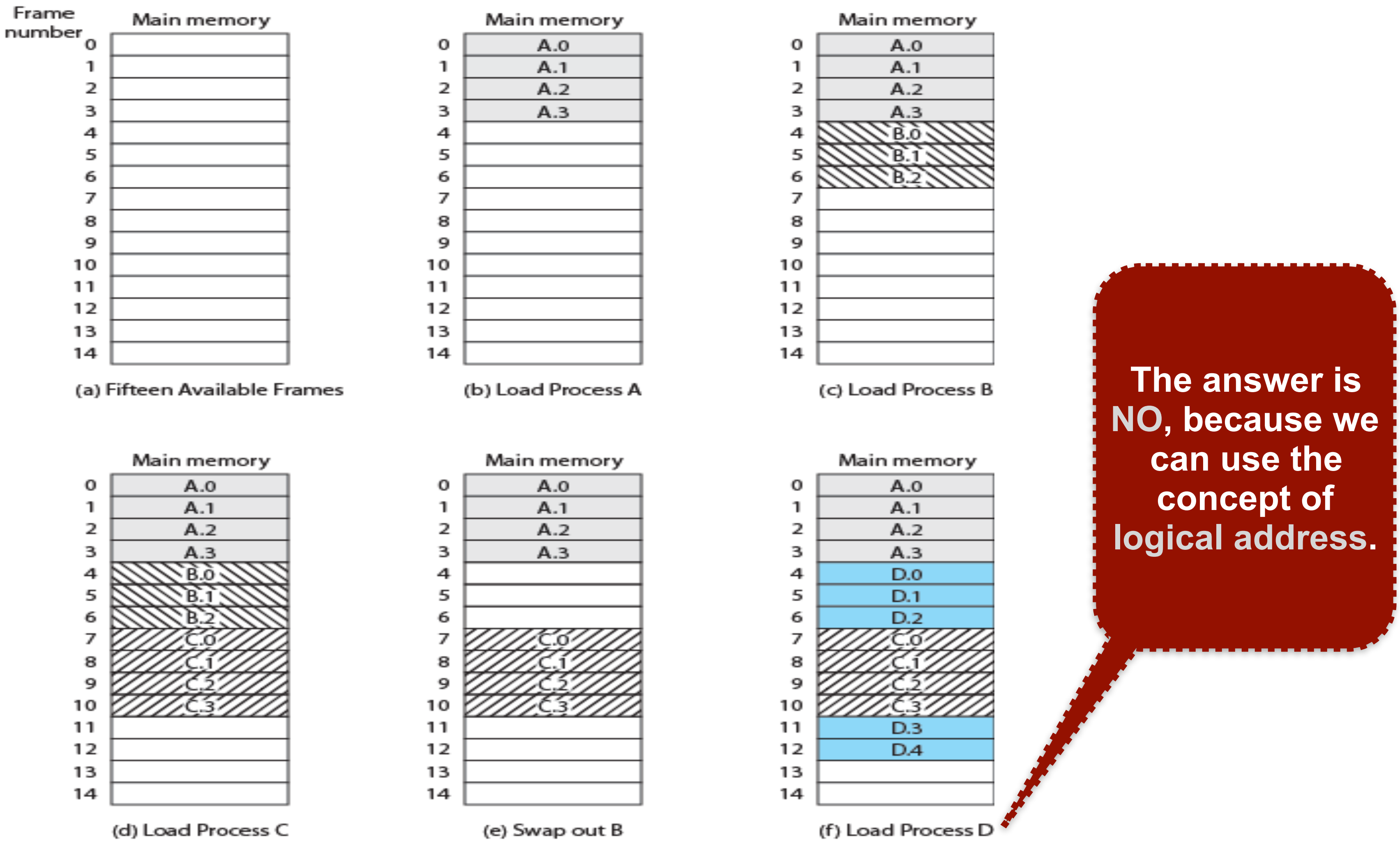


Suppose that there are not sufficient unused contiguous frames to hold the process.

Does this prevent the OS from loading D?

Assignment of Processes to Free Frames

Paging: Example



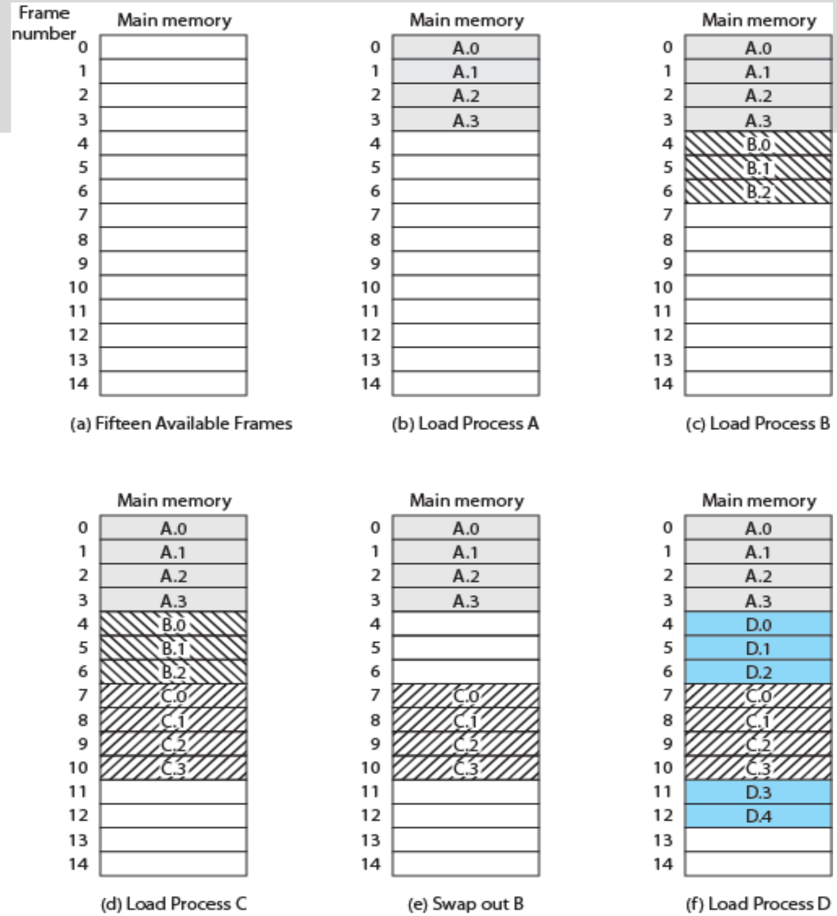
Assignment of Processes to Free Frames

Paging: Page Table

- Maintained by operating system for each process
- Contains the frame location for each page in the process
- Processor must know how to access for the current process
- Used by processor to produce a physical address

Use the logical address (page number, offset) to produce the physical address (frame number, offset)

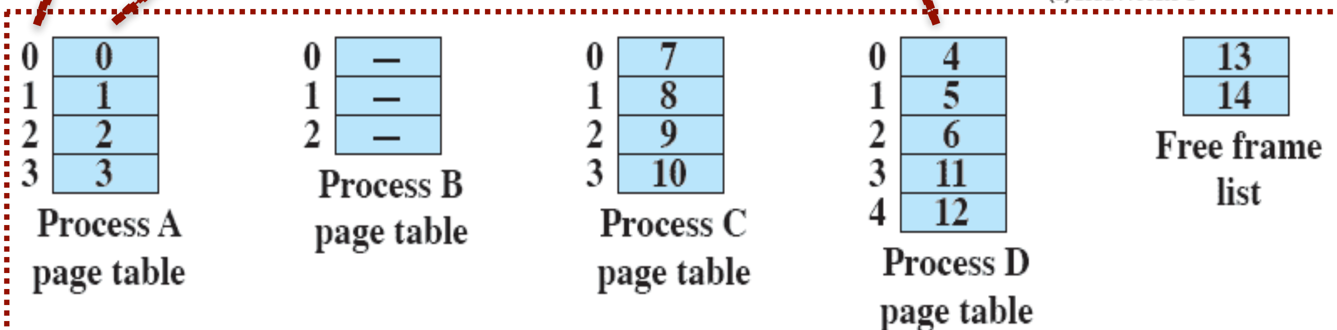
Paging: Data Structures



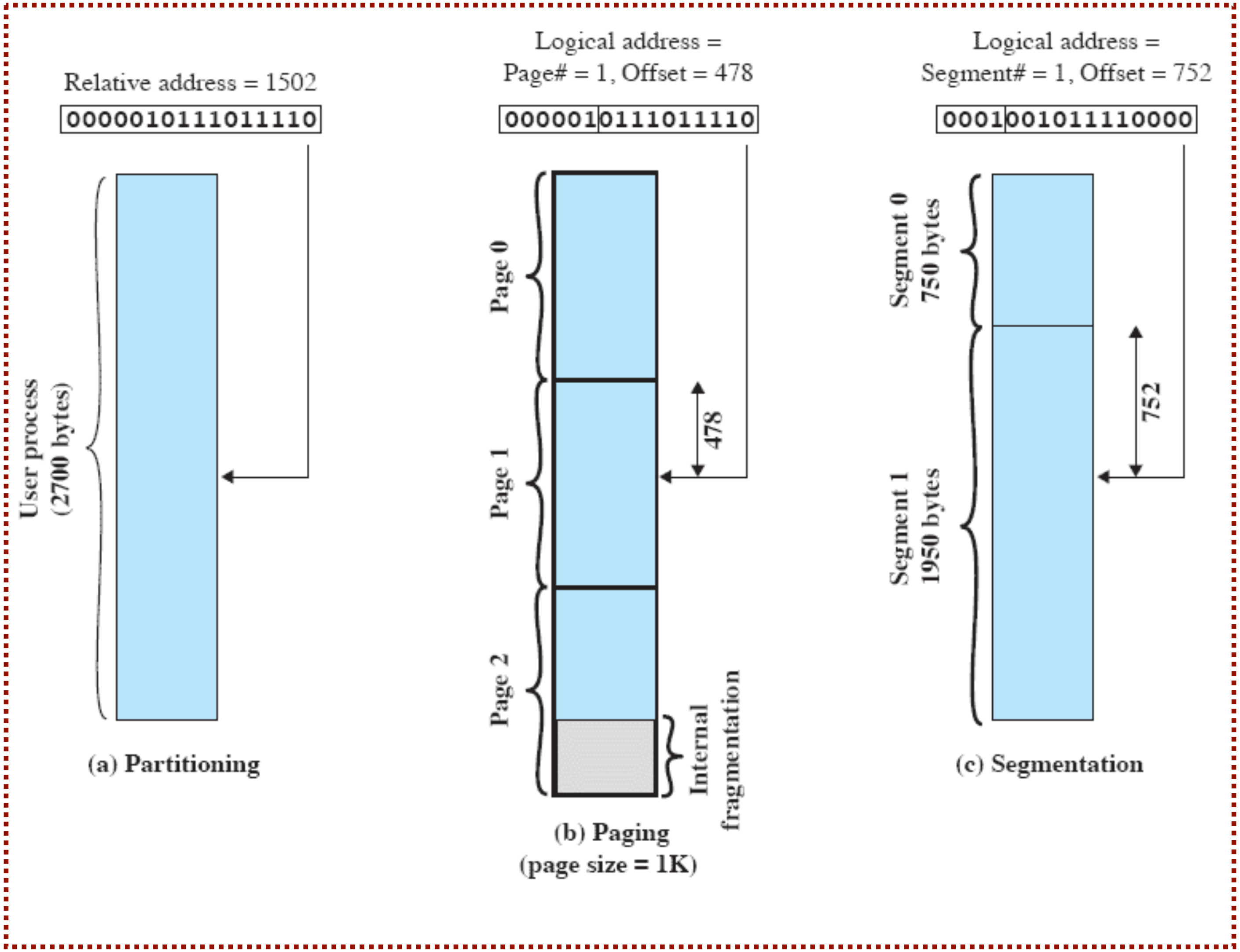
occupy more than one partition (non-contiguous)

Page number

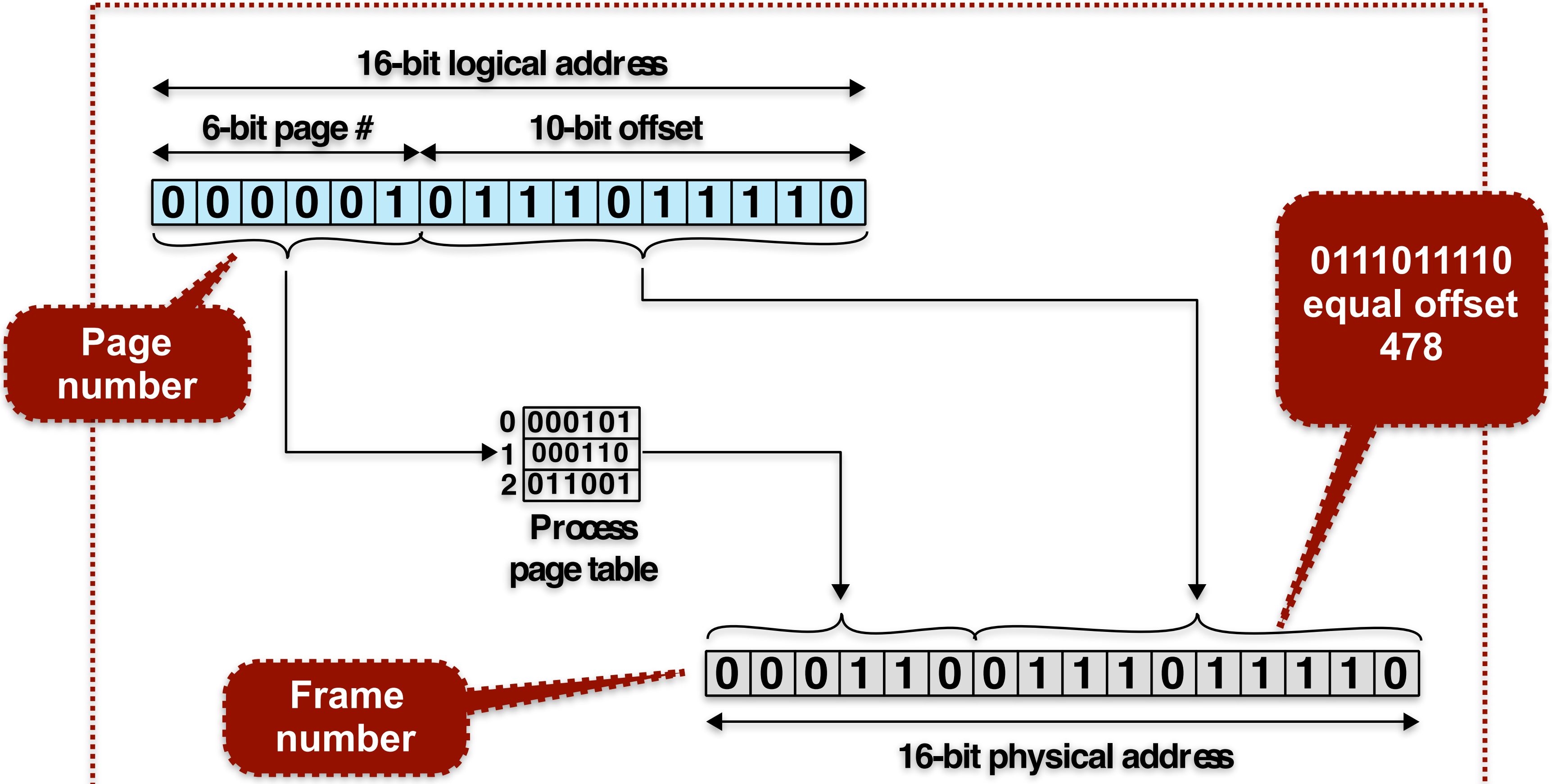
Frame number



Paging: Logical Addresses



Paging: Logical-to-Physical Addresses Translation



(a) Paging

What is the segmentation technique?

Segmentation: Basic Concepts

- A program can be subdivided into segments
  - May vary in length
  - With a maximum length
- Addressing consists of two parts:
  - Segment number
  - Offset
- Similar to dynamic partitioning
- Eliminates internal fragmentation

occupy more than one partition (non-contiguous)

suffers from external fragmentation (less serious)

Segmentation: Basic Concepts

- Usually visible

Paging is invisible to programmers

- Provided as a convenience for organising programs and data
- Typically programmers will assign programs and data to different segments
- For purposes of modular programming the program or data may be further broken down into multiple segments

Programmers must be aware of the maximum segment size limitation

Segmentation: Address Translation

Consequence of unequal-size segments

- There is no simple relationship between logical addresses and physical addresses
- A simple segmentation scheme — use a segment table for each process and a list of free blocks of main memory.
- Each segment table entry would have to give the starting address in main memory of the corresponding segment.

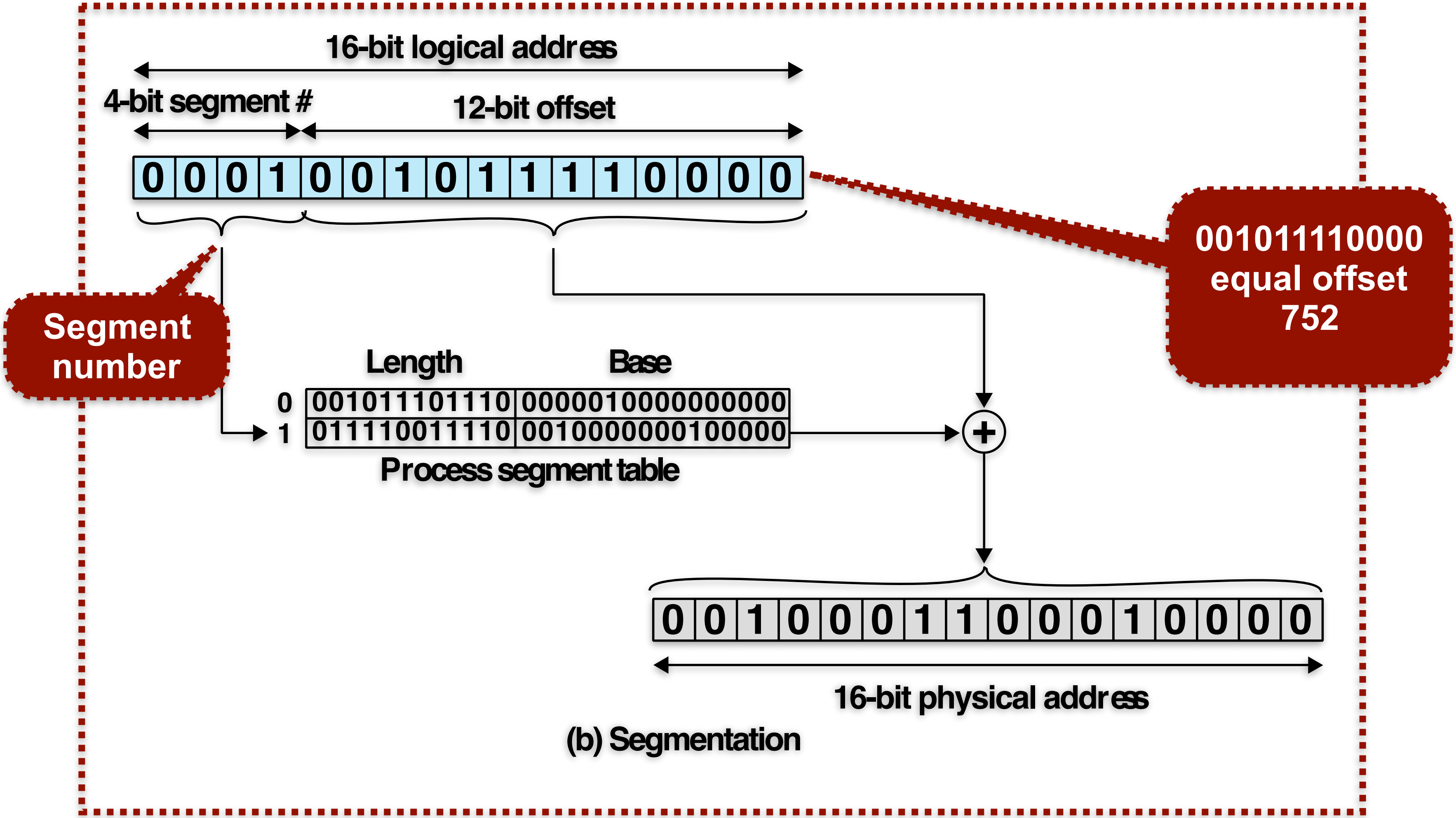
The entry should also provide the length of the segment to ensure that invalid addresses are not used

Segmentation: Addresses Translation

- The following steps are needed for address translation:

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: Logical-to-Physical Addresses Translation



Summary of Lecture 8

- Memory management is one of the most important and complex tasks of an operating system.
- Main memory needs to be treated as a resource to be allocated to and shared among a number of active processes.
- It is desirable to maintain as many processes in main memory as possible (for multiprogramming).
- It is also desirable to free programmers from size restriction in program development.
- Two basic techniques are paging and segmentation (possible combining both).

Next week: Virtual Memory