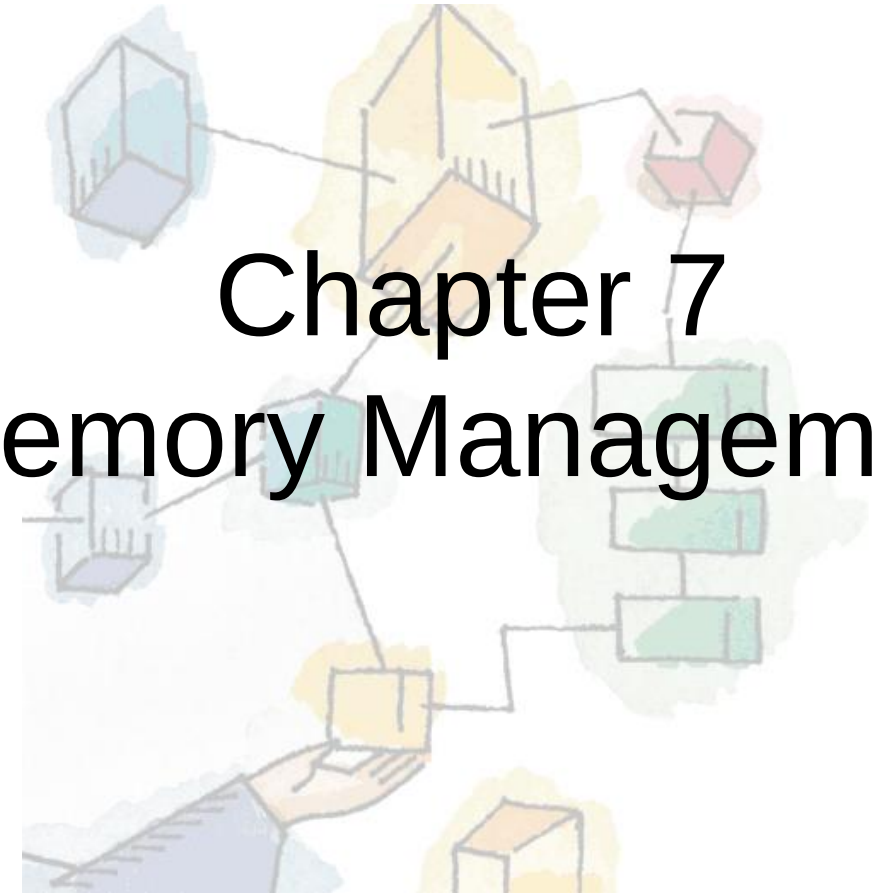
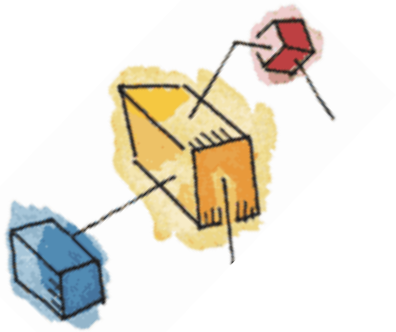
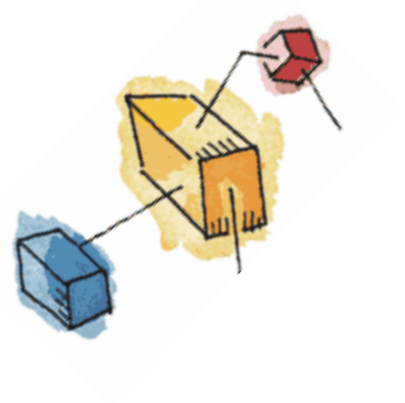


Chapter 7
Memory Management



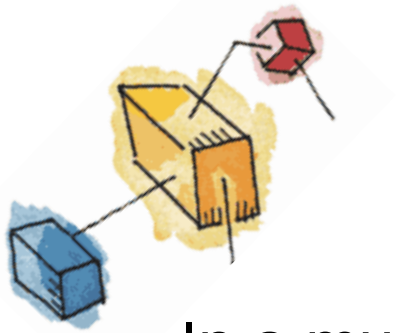
Roadmap

- Basic requirements of Memory Management
- Memory Partitioning
- Basic blocks of memory management
  - Paging
  - Segmentation



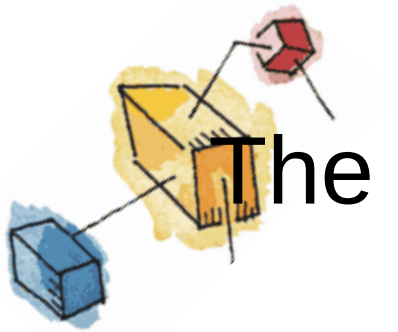
Uni-Programming

- In a uni-programming system, main memory is divided into two parts:
  - One part for the operating system (resident monitor, kernel) and
  - One part for the program currently being executed.



Multi-Programming

- In a multiprogramming system, the "user" part of memory must be further subdivided to accommodate multiple processes.
- *"Memory management is vital in a multiprogramming system."*
  - If only a few processes are in memory, then for much of the time all of the processes will be waiting for I/O and the processor will be idle.
  - Thus memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time.



The need for memory management

- Memory is cheap today, and getting cheaper
  - But applications are demanding more and more memory, there is never enough!
- Memory Management, involves swapping blocks of data from secondary storage.
- Memory I/O is slow compared to a CPU
  - The OS must cleverly time the swapping to maximise the CPU's efficiency

Memory Management

“Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time”

Memory Management Requirements

- Relocation
- Protection
- Sharing
- Logical organisation
- Physical organisation

Requirements: Relocation

- Main memory is shared by multiple processes
- The programmer does not know where the program will be placed in memory,
  - it may be swapped to disk and return to main memory at a different location (relocated)
- Memory references must be translated to the actual physical memory address

Challenge for Relocation: Addressing

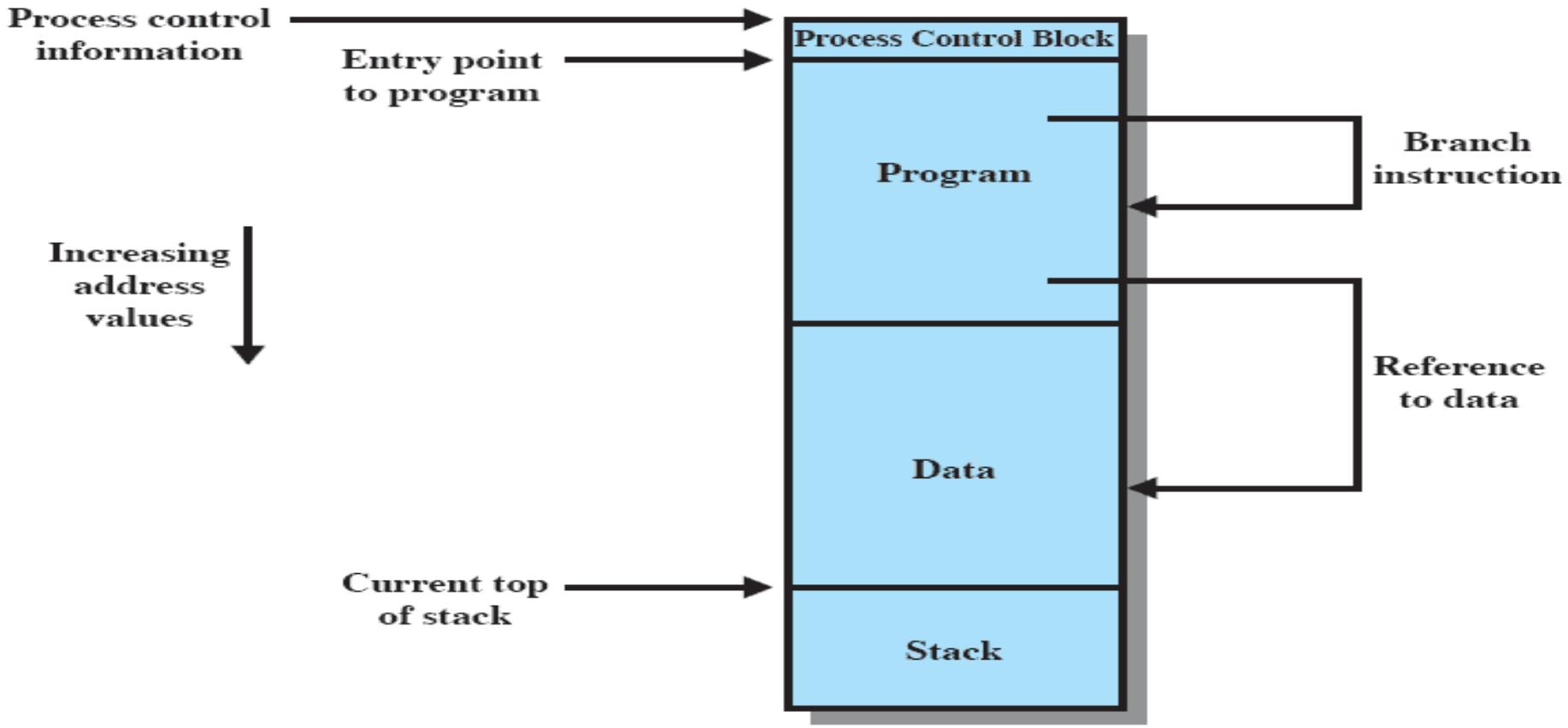
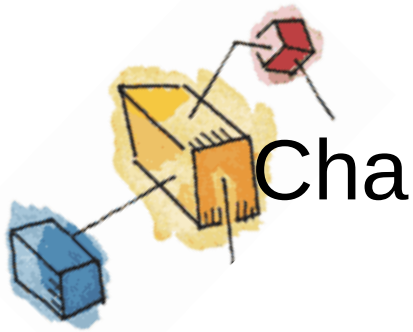


Figure 7.1 Addressing Requirements for a Process



Challenge for Relocation: Addressing

- OS needs to know addresses of
  - PCB
  - Program Area
  - Stack Area
- OS loads process in main memory at specific location so these addresses are known to it

- Mapping between actual program references and physical memory locations needs to be done
- This is done by processor hardware and OS together

Requirements: Protection

- Normally, a user process cannot access any portion of the operating system, neither program nor data. –Why?

- Usually a program in one process cannot branch to an instruction in another process or access the data area of another process. – Why?

Requirements: Protection

- "Satisfaction of Relocation requirement increases the difficulty of satisfying protection"
  - Location of a process in main memory is not fixed
  - Run time checking of memory references Is needed

Requirements: Protection

- Memory protection must be satisfied by processor rather than OS
  - OS cannot anticipate the memory references that a program will make
  - It will be a time consuming task if assigned to OS

Requirements: Sharing

- Protection mechanism must have flexibility to 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
- Processes that are cooperating on some task may need to share access to the same data structure.

Requirements: Sharing

- Hence memory management system must allow controlled access to shared area without compromising the protection requirement.

Requirements: Logical Organization

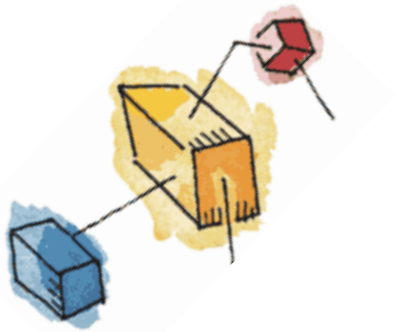
- Memory is organized linearly (usually)
- Programs are written in modules
  - Modules can be written and compiled independently
  - Different degrees of protection given to modules (read-only, execute-only)
  - Share modules among processes
- Hence this mapping needs to be managed

Requirements: Physical Organization

- Memory is organized in two levels: main memory and secondary memory
- Main memory has the advantage of faster access but it is volatile
- Secondary memory has the advantage of bulk and permanent storage but it is slow in access
- The flow between main and secondary memory needs proper management

Requirements: Physical Organization

- Cannot leave the programmer with the responsibility to manage memory
- Memory available for a program plus its data may be insufficient
  - Overlaying allows various modules to be assigned the same region of memory but is time consuming to program
- Programmer does not know how much space will be available



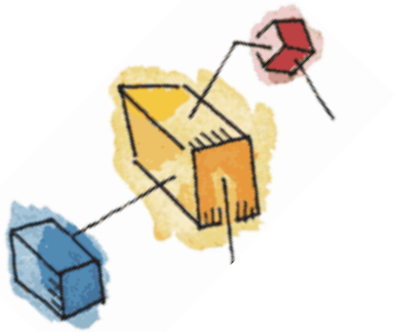
Partitioning

- Basic operation of memory management unit is
  - Bring a process into main memory

- This requires the memory to be properly partitioned
  - Hence partitioning is one of the important memory management techniques

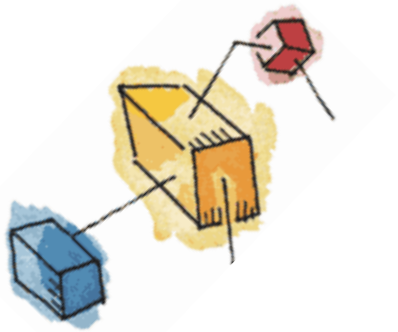
Types of Partitioning

- Fixed Partitioning
- Dynamic Partitioning
- Simple Paging
- Simple Segmentation
- Virtual Memory Paging
- Virtual Memory Segmentation



Fixed Partitioning

- If we assume that OS occupies some fixed portion of main memory
  - Rest of the space is available for processes
- "Partition the available memory into regions with fixed boundaries"
- *Fixed Partitioning*
  - *Equal size*
  - *Unequal size*



Fixed Partitioning

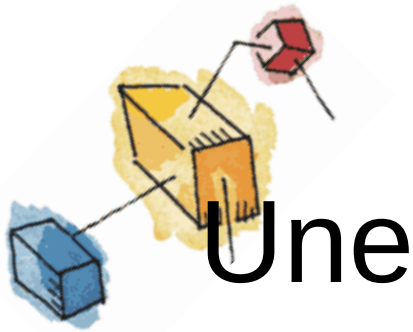
- ***Equal-size partitions***
  - Any process whose size is less than or equal to the partition size can be loaded into an available partition

- The operating system can swap a process out of a partition
  - If none are in a ready or running state

(a) Equal-size partitions

Fixed Partitioning Problems

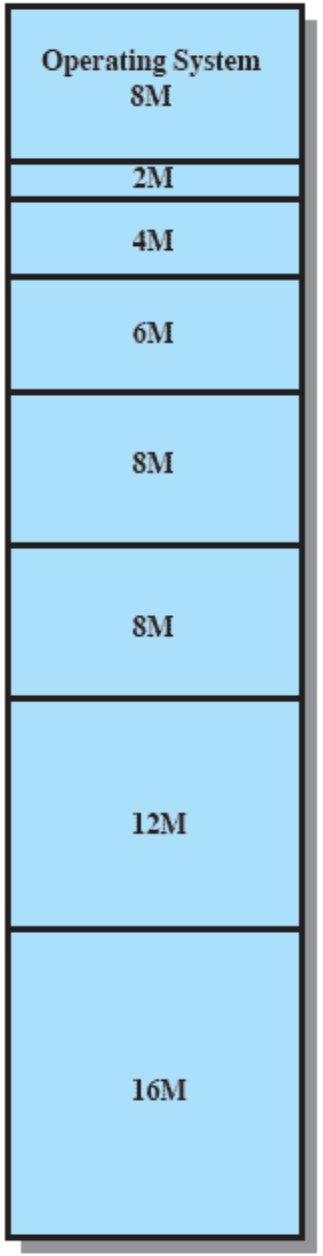
- A program may not fit in a partition.
  - The programmer must design the program with overlays
- Main memory use is inefficient.
  - Any program, no matter how small, occupies an entire partition.
  - This is results in ***internal fragmentation.***
    - ***Solution??***



Solution – Unequal Size Partitions

- Lessens both problems
  - but doesn't solve completely

- In Figure,
  - Programs up to 16M can be accommodated without overlay
  - Smaller programs can be placed in smaller partitions, reducing internal fragmentation



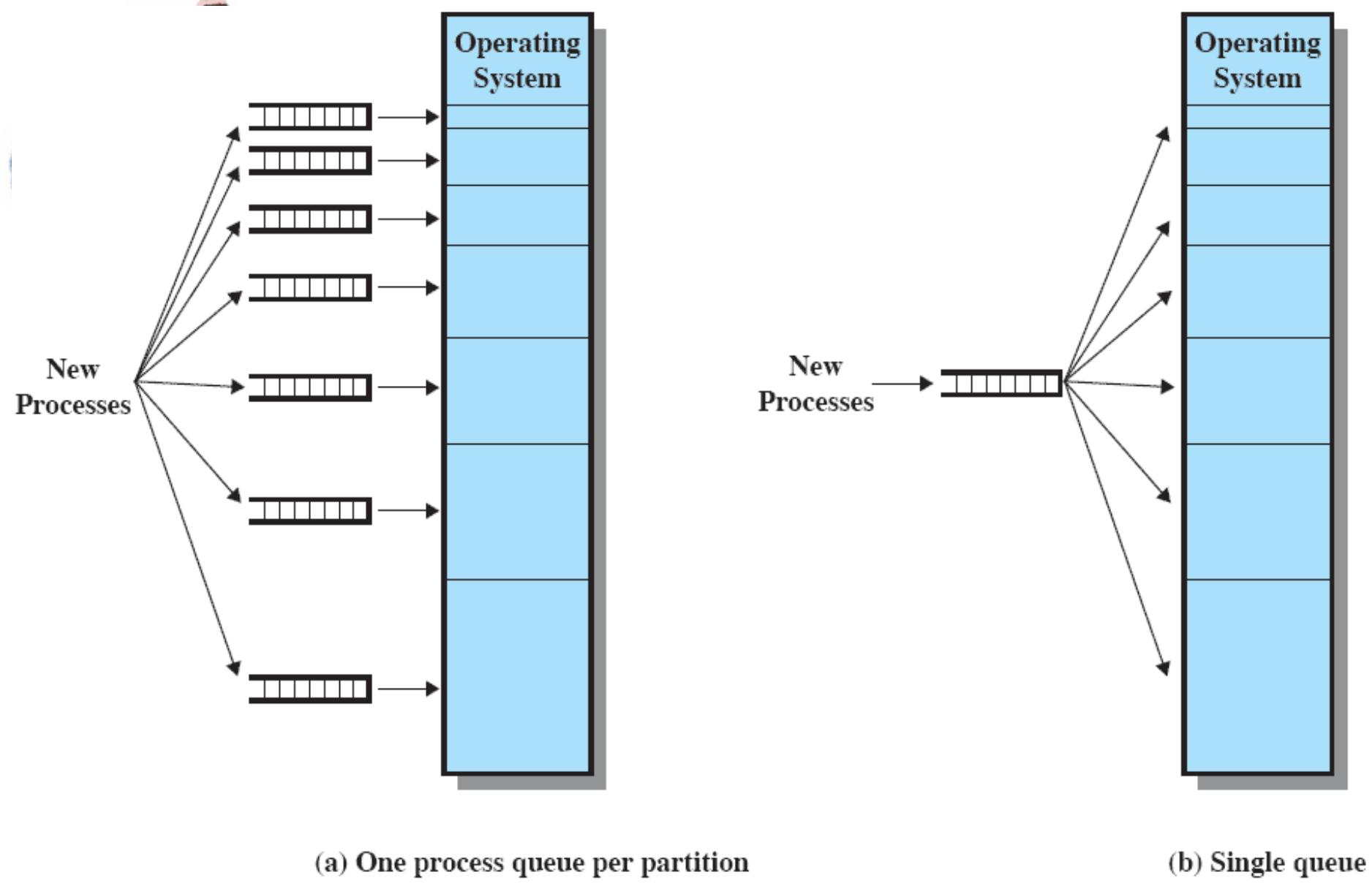
(b) Unequal-size partitions

Placement Algorithm

- **Equal-size**
  - Placement is trivial (no options)
  - As long as a partition is available, process can be loaded into it
  - Does not matter which partition to choose
  - If all partitions are occupied and no ready or running processes, swapping can be done

Placement Algorithm

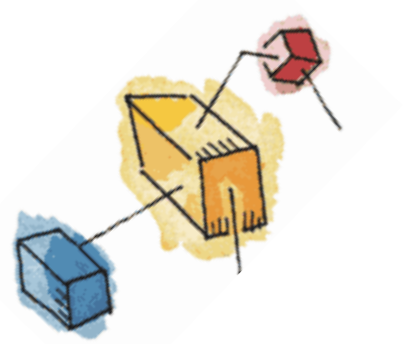
- Unequal-size
  - Can assign each process to the smallest partition within which it will fit
  - Processes are assigned in such a way as to minimize wasted memory within a partition
  - *Queue models:*
    - One queue per partition
    - Single queue



(a) One process queue per partition

(b) Single queue

Figure 7.3 Memory Assignment for Fixed Partitioning



Placement Algorithm

- Problem with one queue per partition is
  - It might be optimum from a partition point of view, but not optimum from system's point of view
  - If there are no processes which can fit a particular partition, then it will remain unused
  - Hence, using a single queue is a better alternative

Advantages: Fixed Partitioning

- Fixed partitioning schemes are relatively simple and require minimum operating system software and processing overhead

- Unequal size partitioning provides flexibility to the basic approach

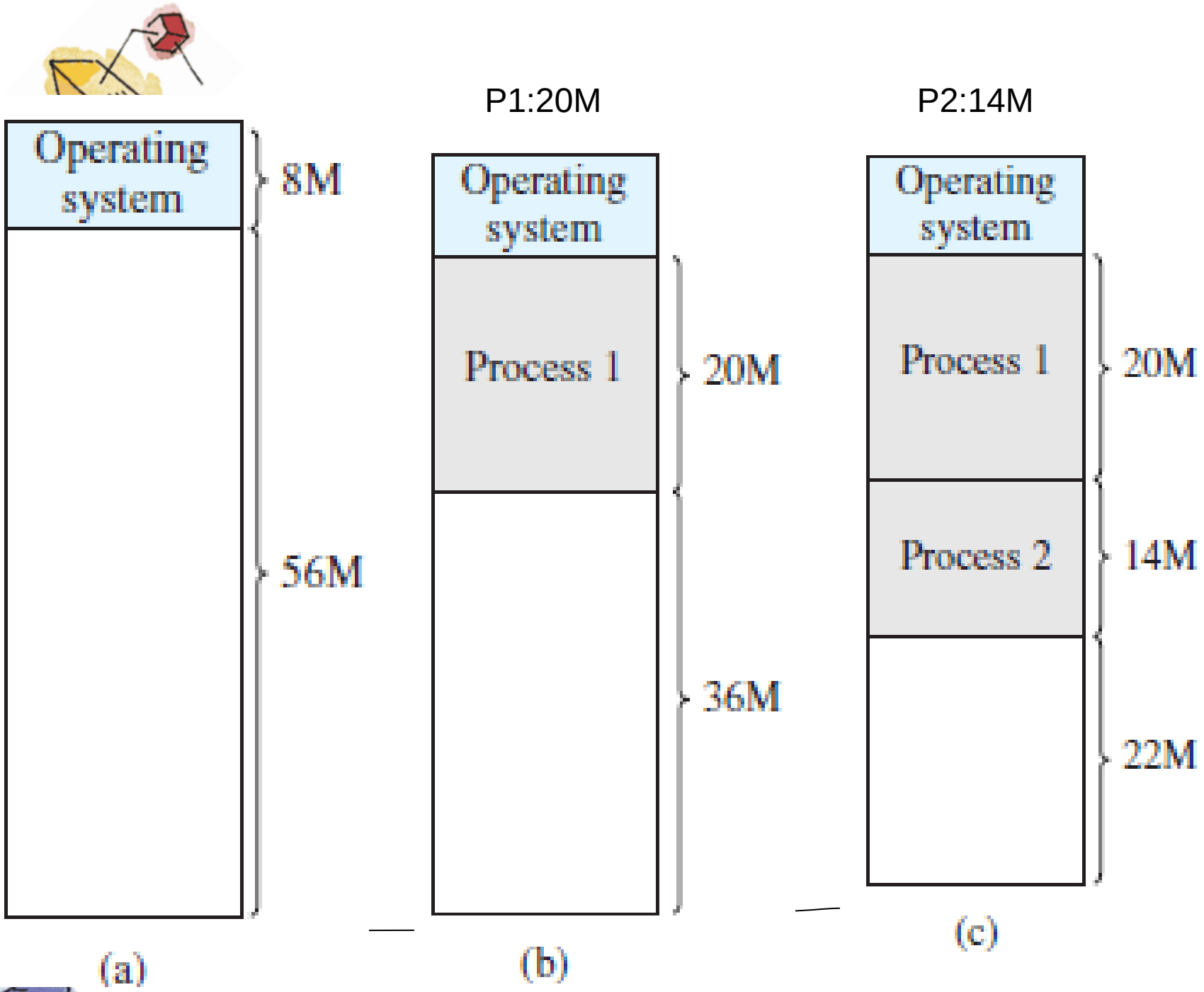
Remaining Problems with Fixed Partitions

- The number of active processes is limited by the system
  - i.e. limited by the pre-determined number of partitions

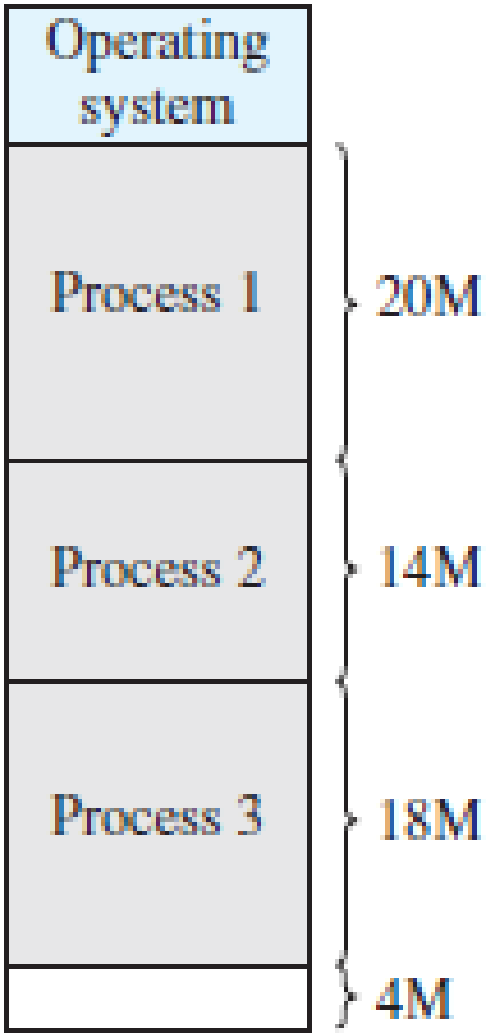
- A large number of very small processes will not use the space efficiently
  - In either fixed or variable length partition methods

Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required

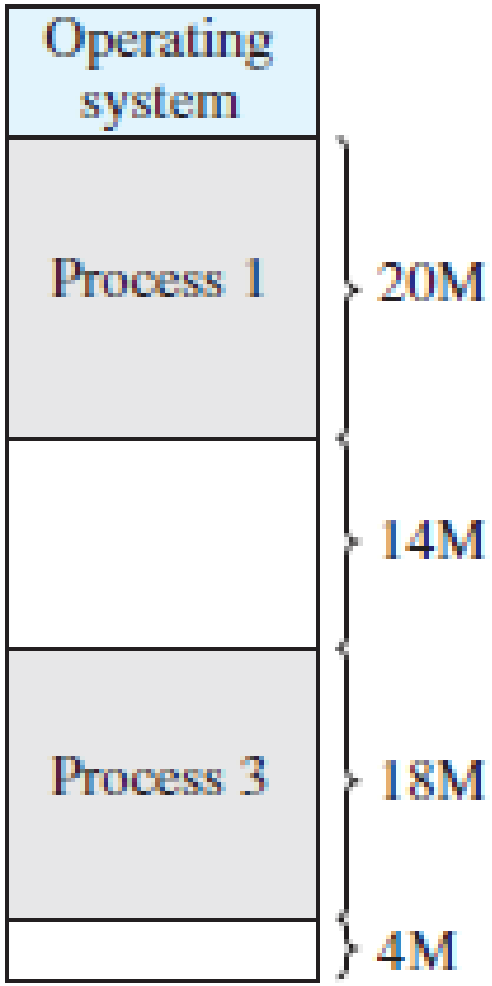

P1:20M
P2:14M
Operating system
8M
56M
(a)
Operating system
Process 1
20M
36M
(b)
Operating system
Process 1
20M
Process 2
14M
22M
(c)

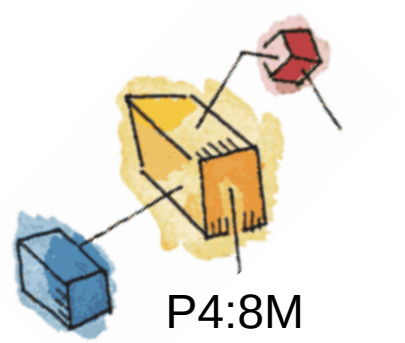
P3:18M



(d)

None Ready, new process arriving, Swap P2

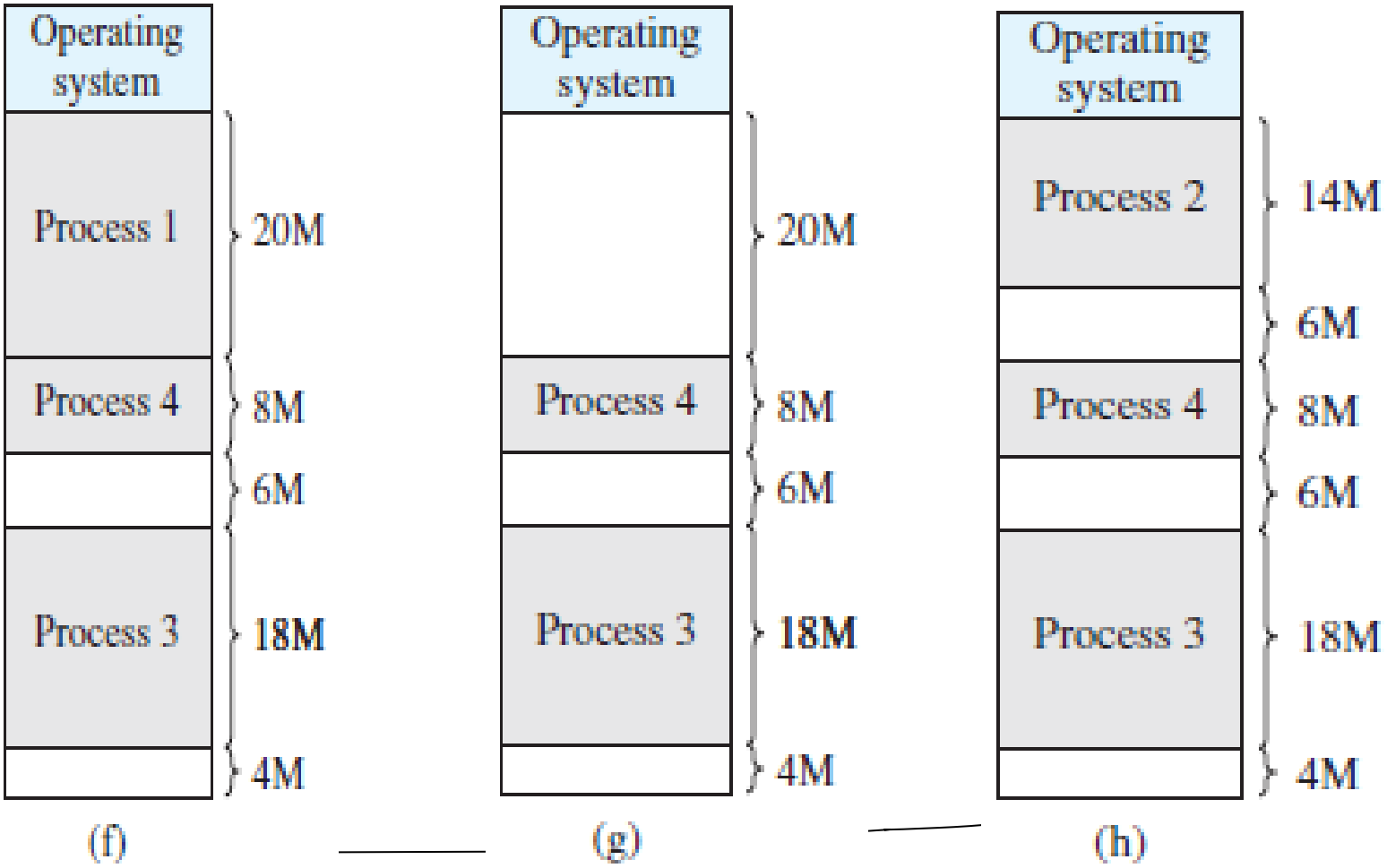




P4:8M

None Ready,
P2 Ready-Suspend,
Swap P1

P2 swapped in



Dynamic Partitioning: Issue

- ***External Fragmentation***
  - Memory external to all processes is fragmented

- Can resolve using ***compaction***

  - OS moves processes so that they are contiguous
  - Needs dynamic relocation
  - Time consuming and wastes CPU time

Placement Algorithms

- Compaction is costly
- Hence operating system must make intelligent decision when a process is brought in and multiple options are available
- Three placement algorithms:
  - Best Fit
  - Next Fit
  - First Fit

Placement Algorithms

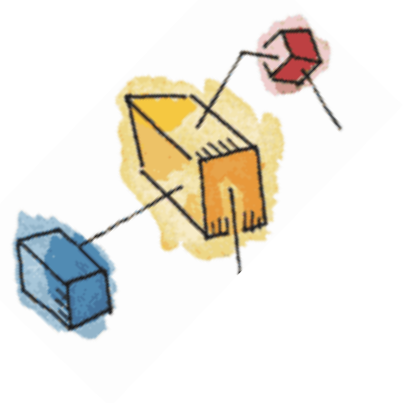
- ***Best-fit algorithm***
  - Chooses the block that is closest in size to the request
  - Worst performer overall
    - Since smallest block is found for process, the smallest amount of fragmentation is left
    - Memory compaction must be done more often

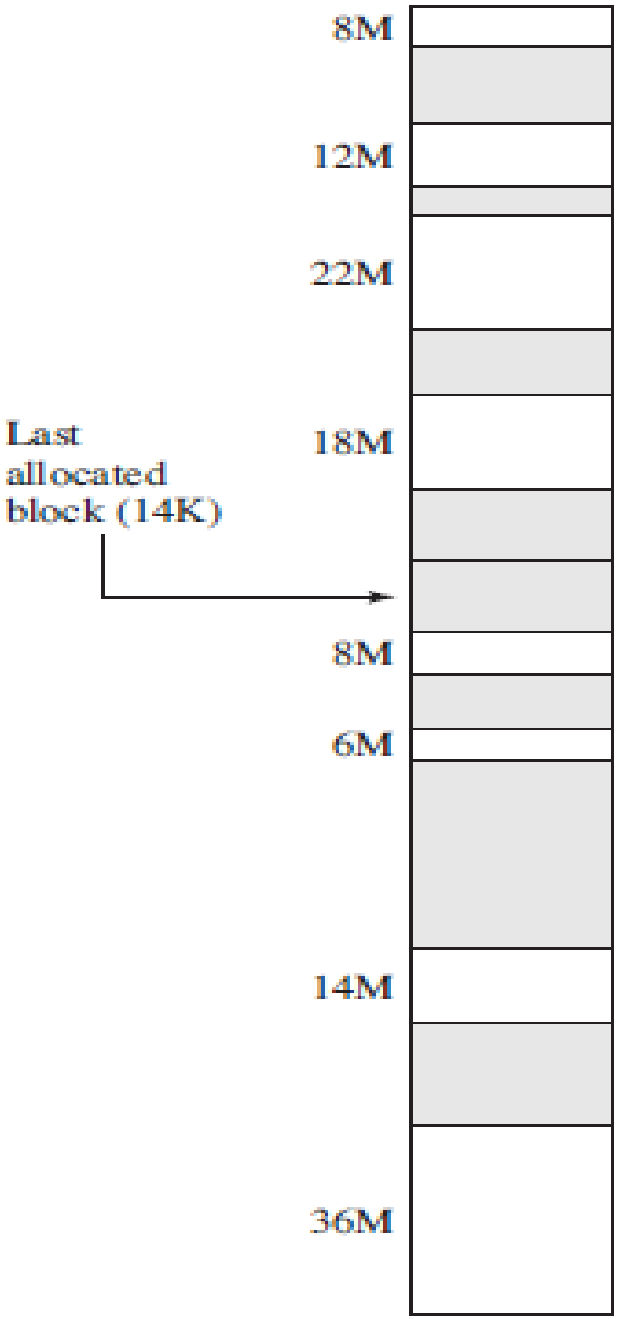
Placement Algorithms

- ***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
  - Small holes in front

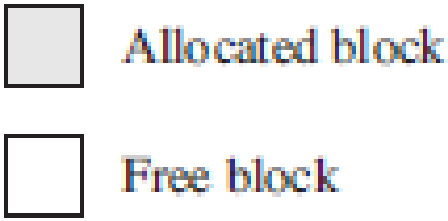
Placement Algorithms

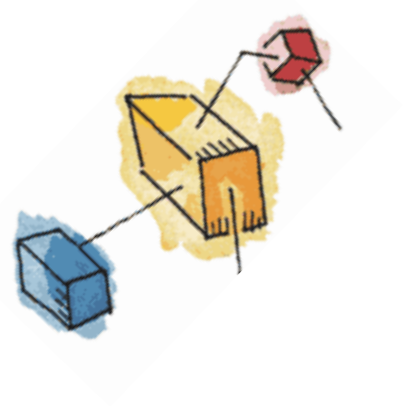
- ***Next-fit algorithm***
  - Scans memory from the location of the last placement
  - More often allocate a block of memory at the end of memory where the largest block is found
  - The largest block of memory is broken up into smaller blocks
  - Compaction is required to obtain a large block at the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



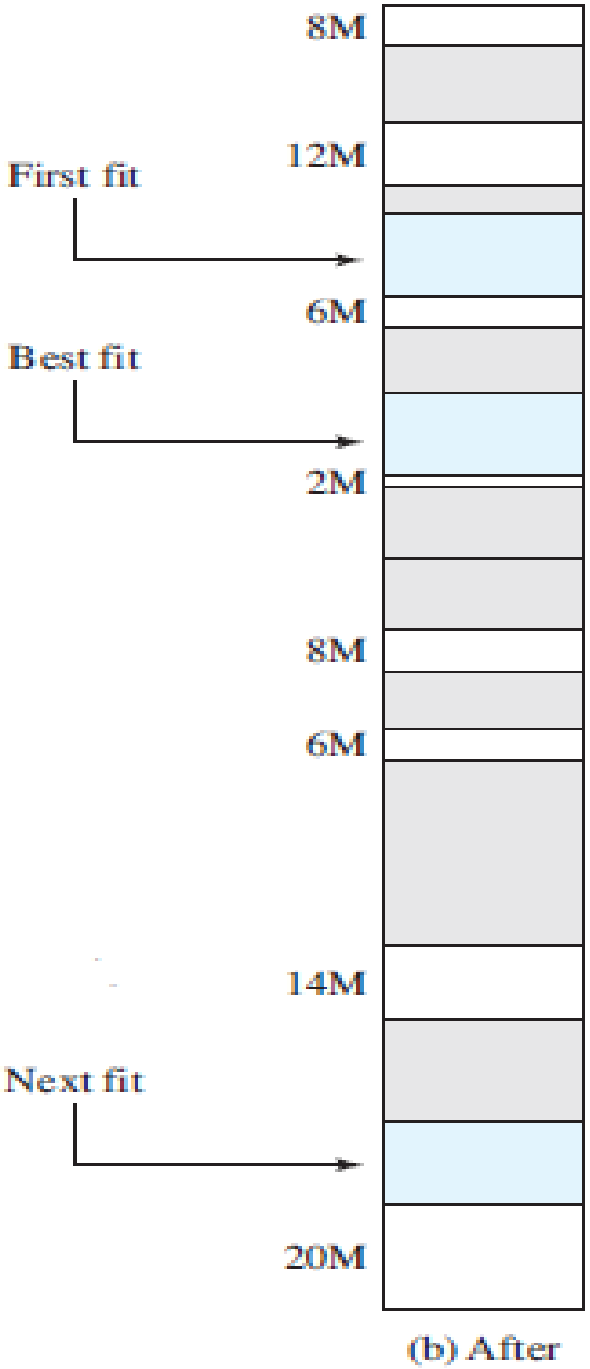

8M
12M
22M
18M
Last
allocated
block (14K)
8M
6M
14M
36M

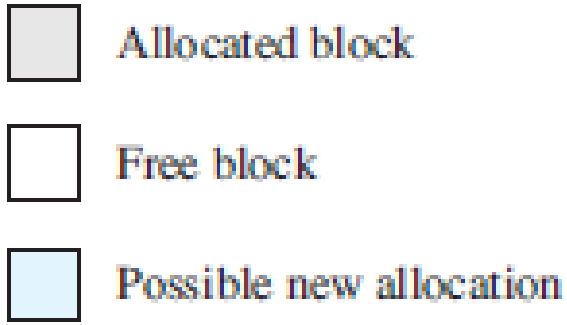
(a) Before


Allocated block
Free block



16M Allocation



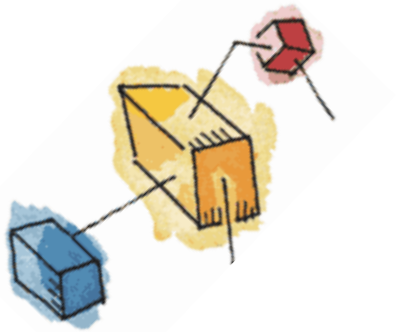


Replacement Algorithms

- In dynamic partitioning, when all processes in main memory are blocked and there is insufficient memory, even after compaction
  - OS must swap one of the process to bring a new process or a ready suspended process
- This needs replacement algorithm

Common Drawbacks

- Fixed Partitioning
  - Internal Fragmentation
  - Limits the number of active processes
- Dynamic Partitioning
  - External Fragmentation
  - Compaction increases overhead
- Another Approach
  - Buddy System



Buddy System

- Entire space available is treated as a single block of 2^U
- If a request of size s *is made satisfying* $2^{U-1} < s <= 2^U$
  - entire block is allocated
- Otherwise block is split into two equal buddies
- Process continues until smallest block greater than or equal to s is generated

Example of Buddy System

- Consider block of 512 bytes
  - Request S=512 , 256<*512* <= 512
    - Condition satisfied, Allocate entire block
  - Request S=400, 256<*400* <= 512
    - Condition satisfied , Allocate entire block
  - Request S=200, Condition not satisfied
    - so split in two equal buddies (256,256)
      - Check condition again, 128<*200* <=256
      - Allocate 256 byte block

Available Space

Request A:100K

Request B:240K

Request C:64K

Request D:256K

Release B

Release A

Request E:75K

Release C

Release E

Release D

Tree Representation of Buddy System

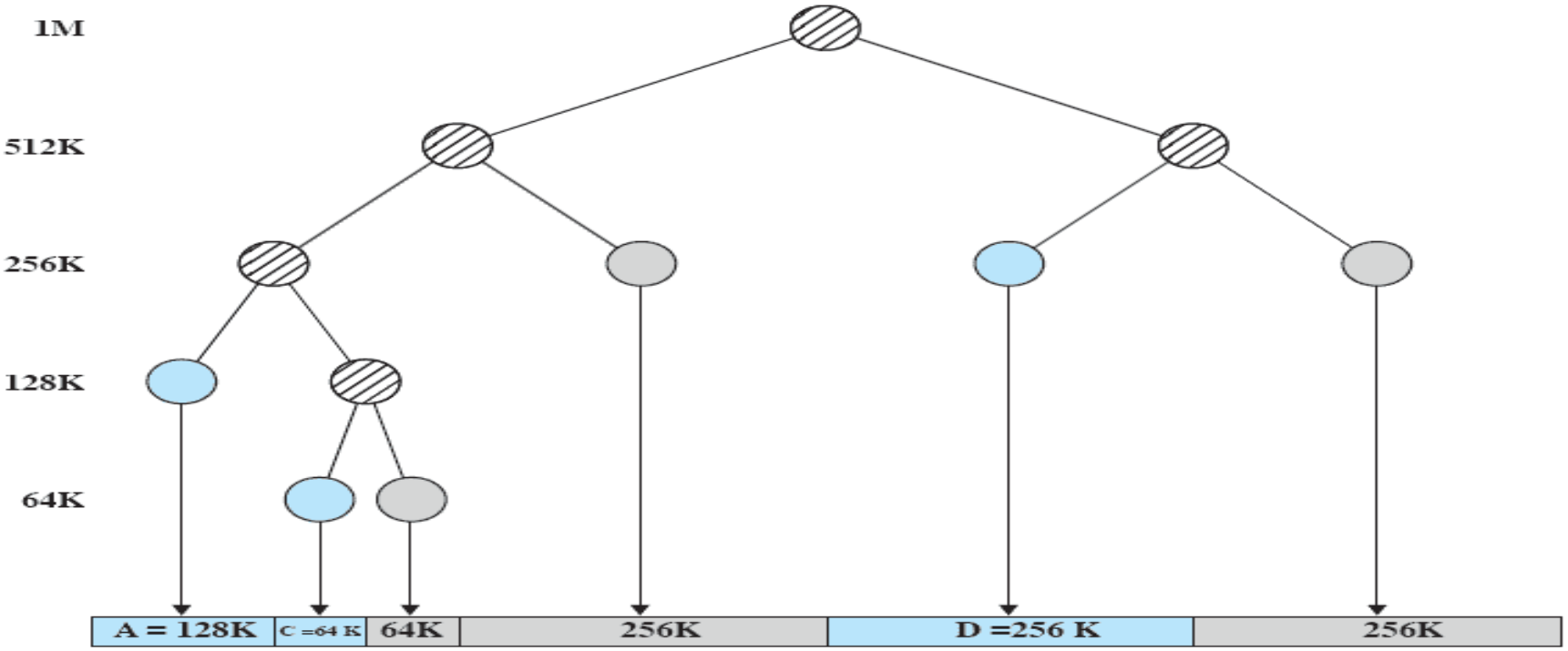
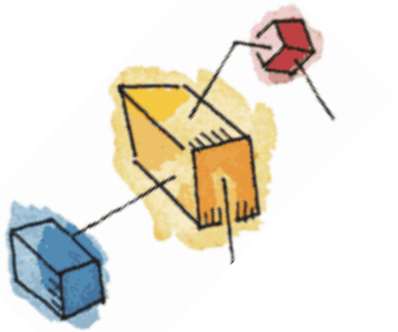


Figure 7.7 Tree Representation of Buddy System

Tree Representation of Buddy System

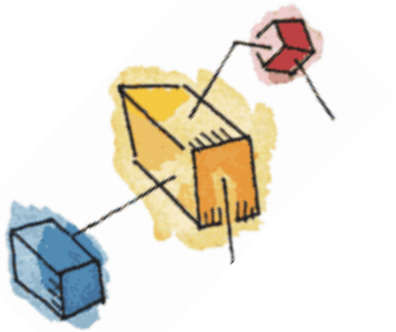
- If two buddies are leaf nodes, then one of them must be an allocated node, otherwise they should be merged

- At any point of time, the leaf nodes define current allocation



Relocation

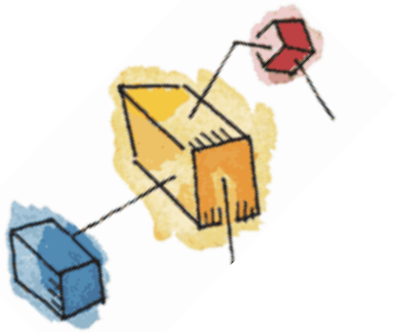
- When program loaded into memory the actual (absolute) memory locations are determined
- A process may occupy different partitions which means different absolute memory locations during the lifetime
  - Reasons?
    - Swapping
    - Compaction



Relocation

- The schemes where relocation is needed:
  - Equal size fixed partitioning
  - Unequal size fixed partitioning (single queue)
  - Dynamic partitioning
  - Compaction

- Hence physical memory references of a process are not fixed
  - This issue is the focus of addressing



Types of Addresses

- Physical or Absolute
  - The absolute address or actual location in main memory.
- Logical
  - Reference to a memory location independent of the current assignment, generated by CPU
- Relative
  - Logical address, where address expressed as a location relative to some known point (i.e. beginning).

Address Translation

- For system with relative addressing, address translation is done in two steps:
  - Load
    - Load process in main memory with all references in relative form
  - Calculate
    - Calculate physical address when instructions or data with relative address is encountered

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

Address Translation

- When a process is assigned running state, base register is loaded with starting physical address of the process
- Bound register is loaded with ending physical address
- When relative address is encountered
  - The value of the base register is added to a relative address to produce an absolute address

Address Translation

- 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

- This mechanism supports protection

Hardware Support

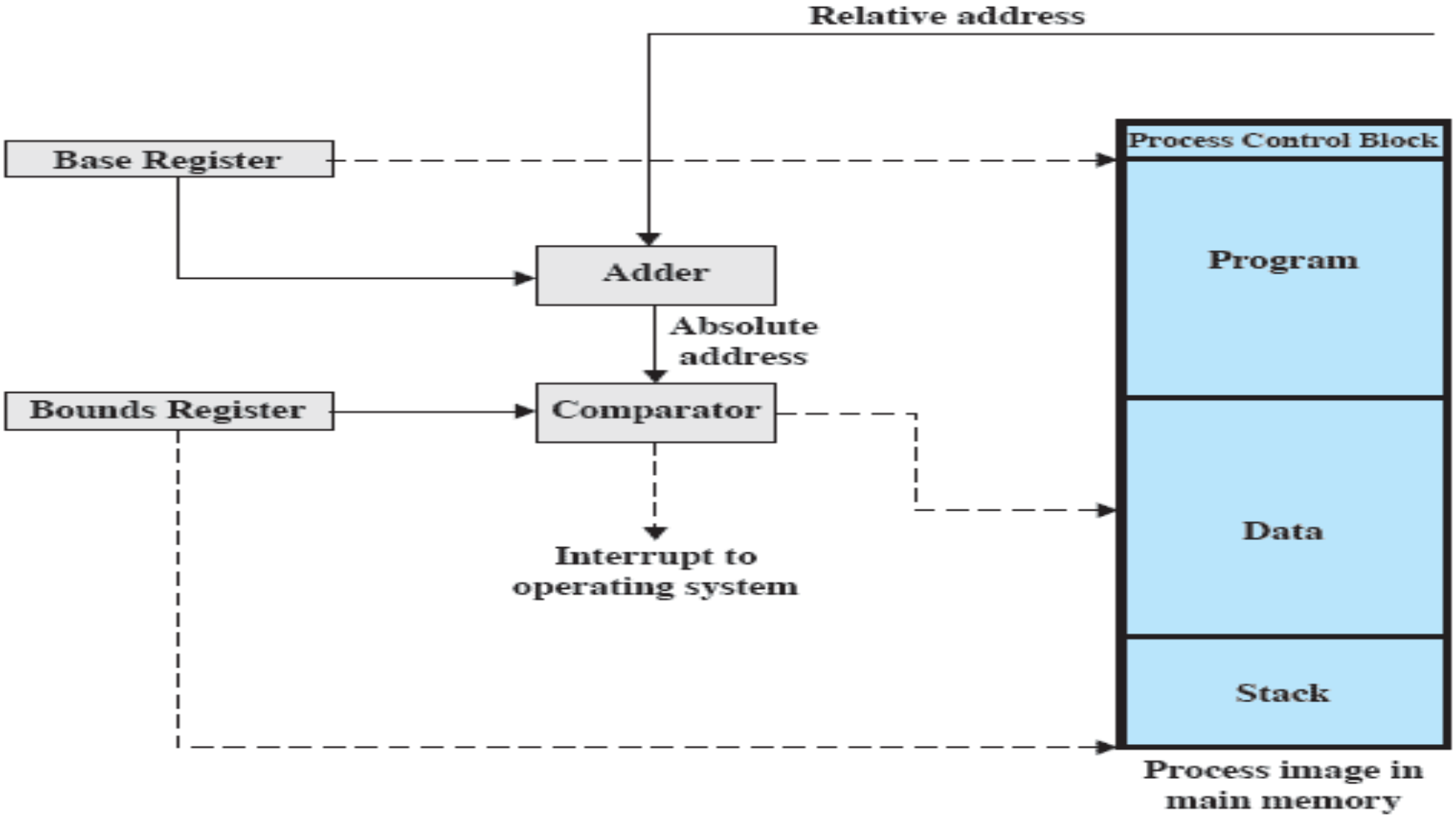
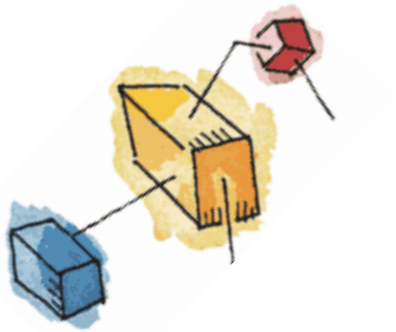
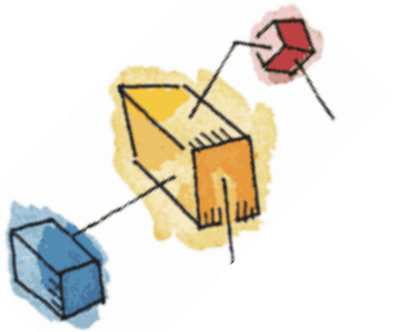


Figure 7.8 Hardware Support for Relocation



Paging

- In paging scheme,
  - Main memory is partitioned into small, fixed-size chunks
  - Each process is divided into the same size chunks
- The chunks of memory are called ***frames***
- The chunks of a process are called ***pages***

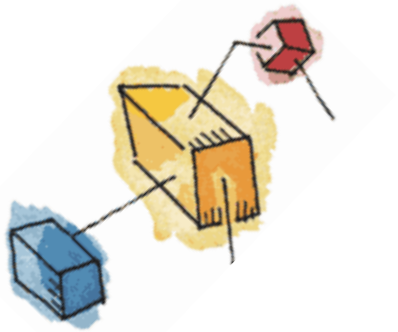


Paging

- When a process is to be loaded in main memory,
  - Free frames are found
  - Pages are loaded into those frames

- *List of free frames is maintained by OS*

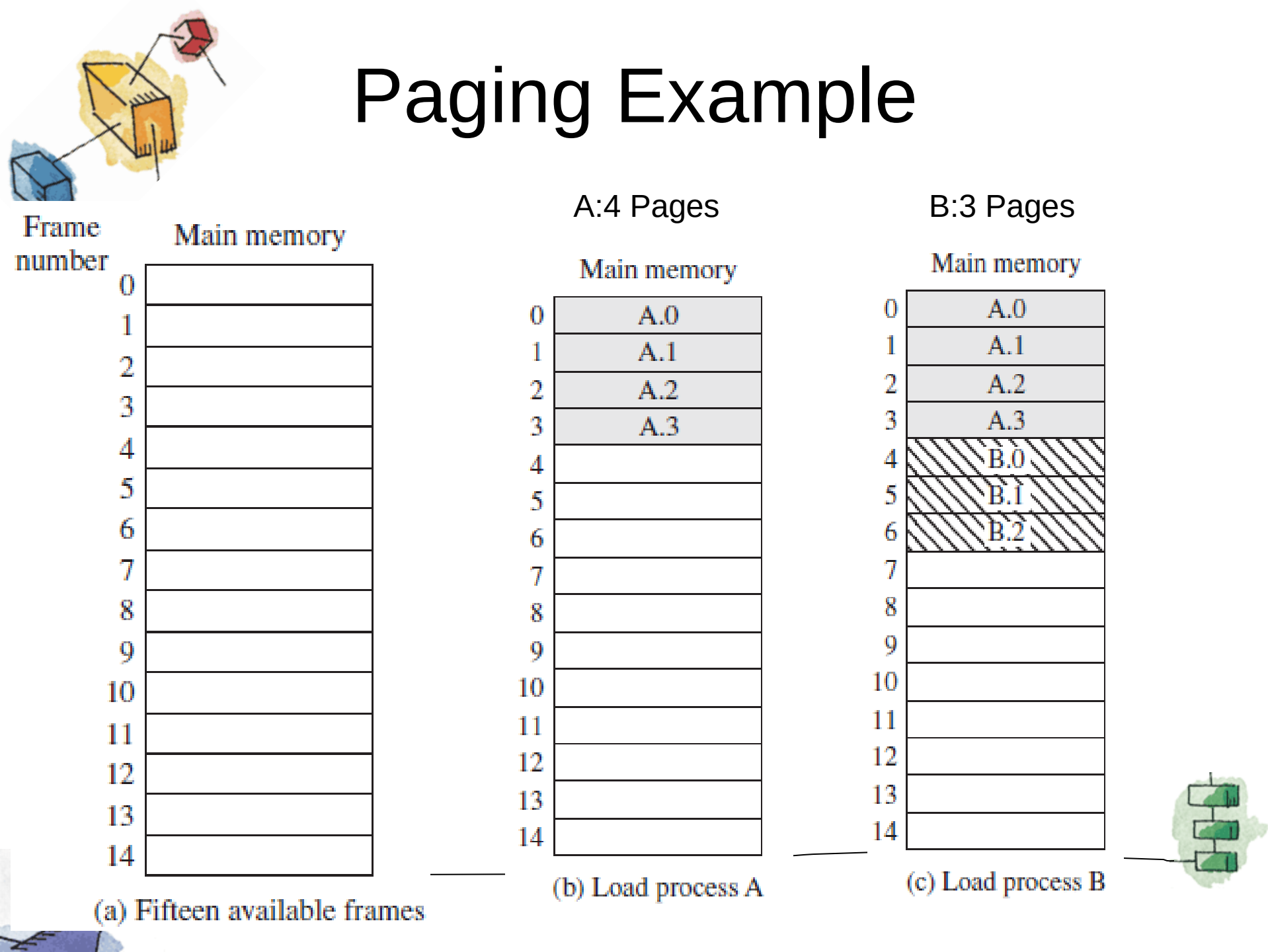
- If contiguous frames are not available,
  - Then also new process can be loaded
    - This requires a data structure to keep track of current allocation



Paging- Page Table

- Operating system maintains a page table for this purpose
  - *1 page table per process*
  - Page Table Entry contains the frame location where the page is stored

Paging Example

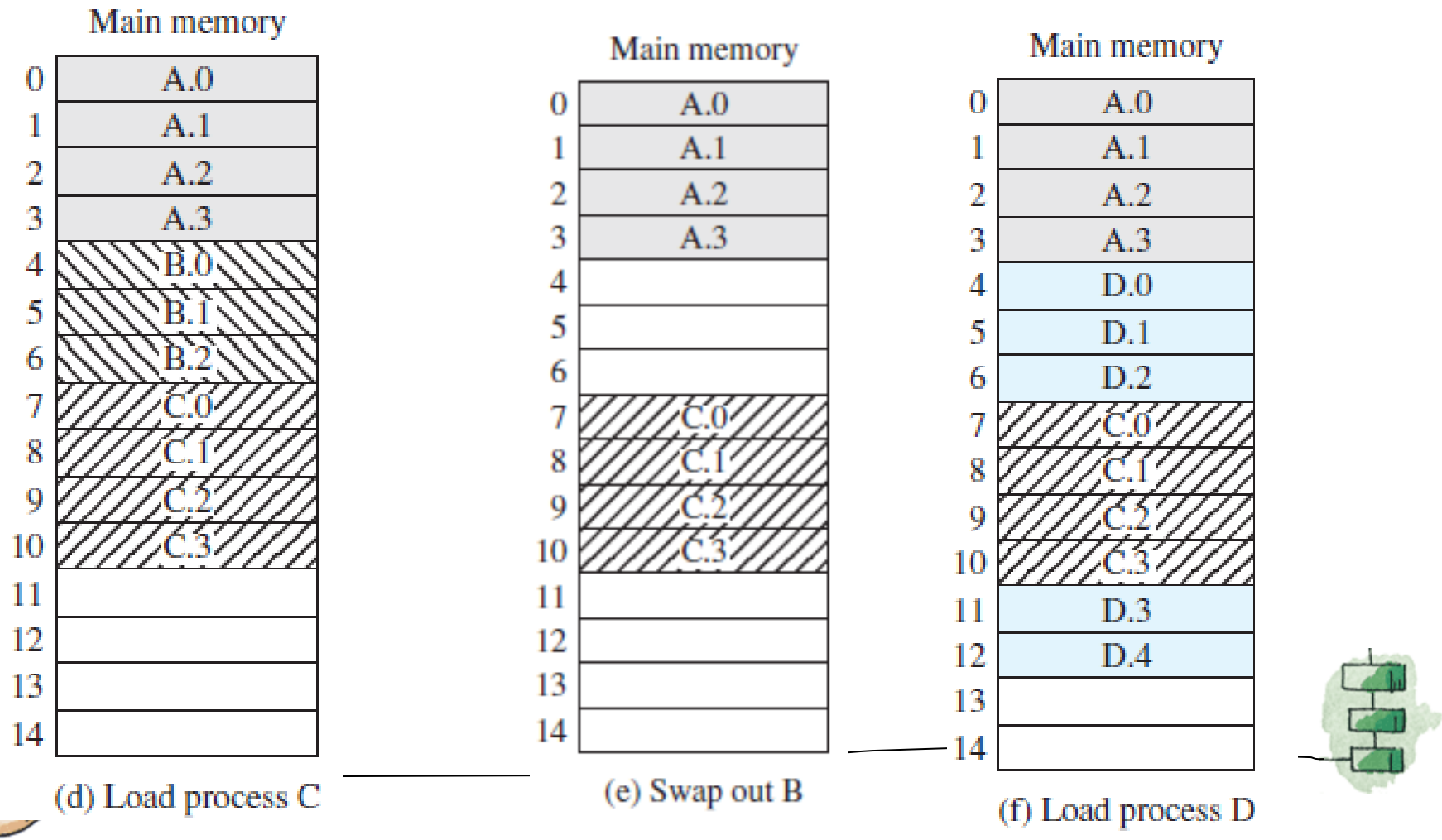


Paging Example

C:4 Pages

All blocked,
bring new D:5 pages

D:5 Pages



Page Table Entries

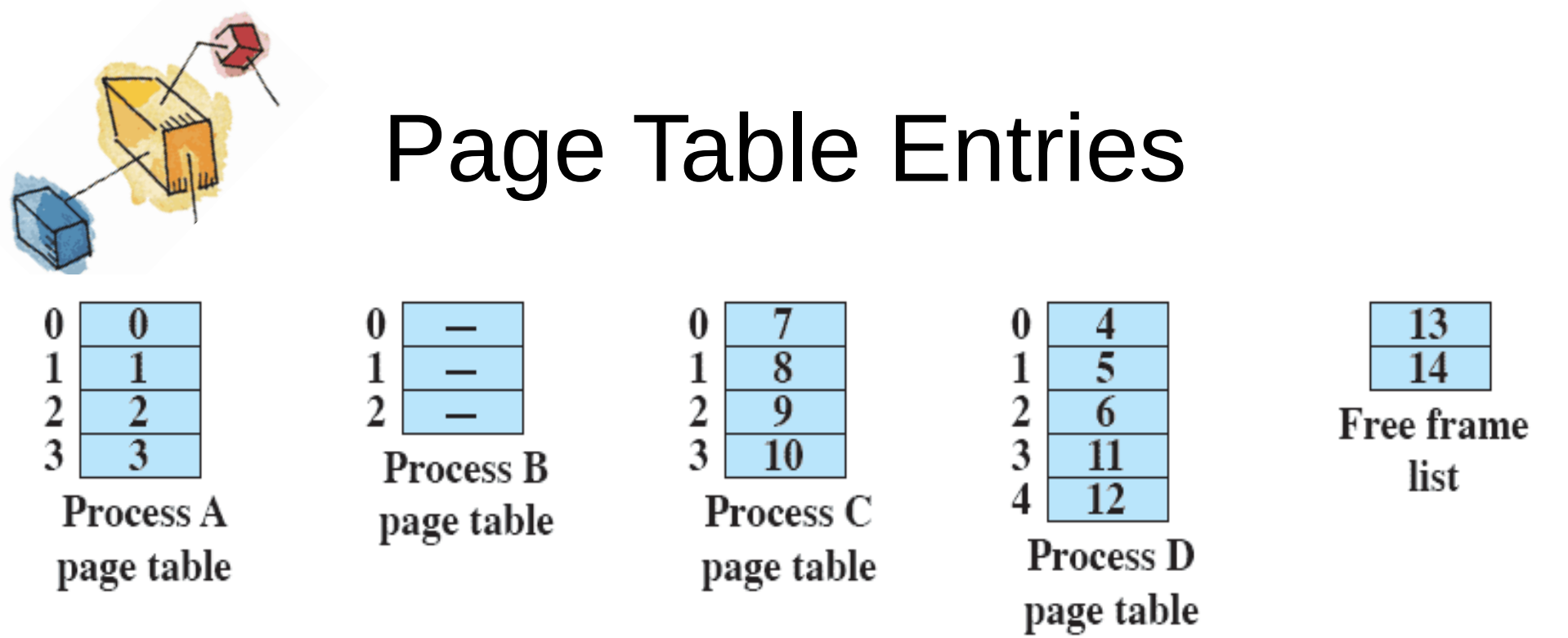


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

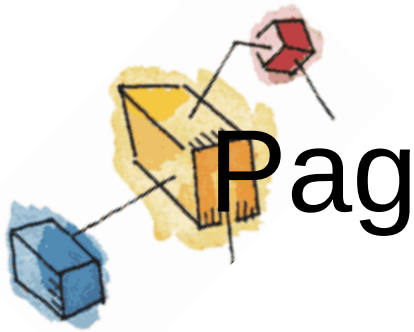
Paging: Address Structure

- Each logical address in program is a pair
  - Page Number
  - Offset
  - E.g. Page 1 offset 5

- Address Translation
  - *Page number, offset*

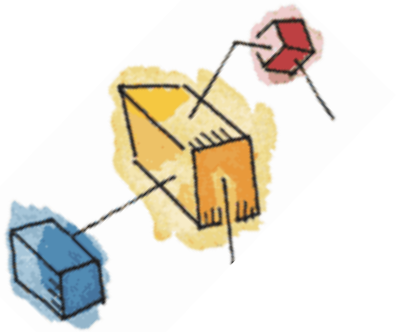
    To
  - *Frame number, offset*

- Done by processor hardware



Paging vs. Fixed Partitioning

- Paging has small partition size compared to fixed partitioning
- Paging does not require partitions to be contiguous
- Paging results into small amount of internal fragmentation (last page)



Paging: Example

- Consider the memory supporting 16 bit addresses,
  - address space is 64K

- Given page size 1024 bytes (1 K)
  - Number of pages?
    - 2^16/ 2^10 = 2^6
    - 64 pages of 1K each
  - Address Structure?
    - 10 bits offset, 6 bits page number = 16 bit address

Paging: Example

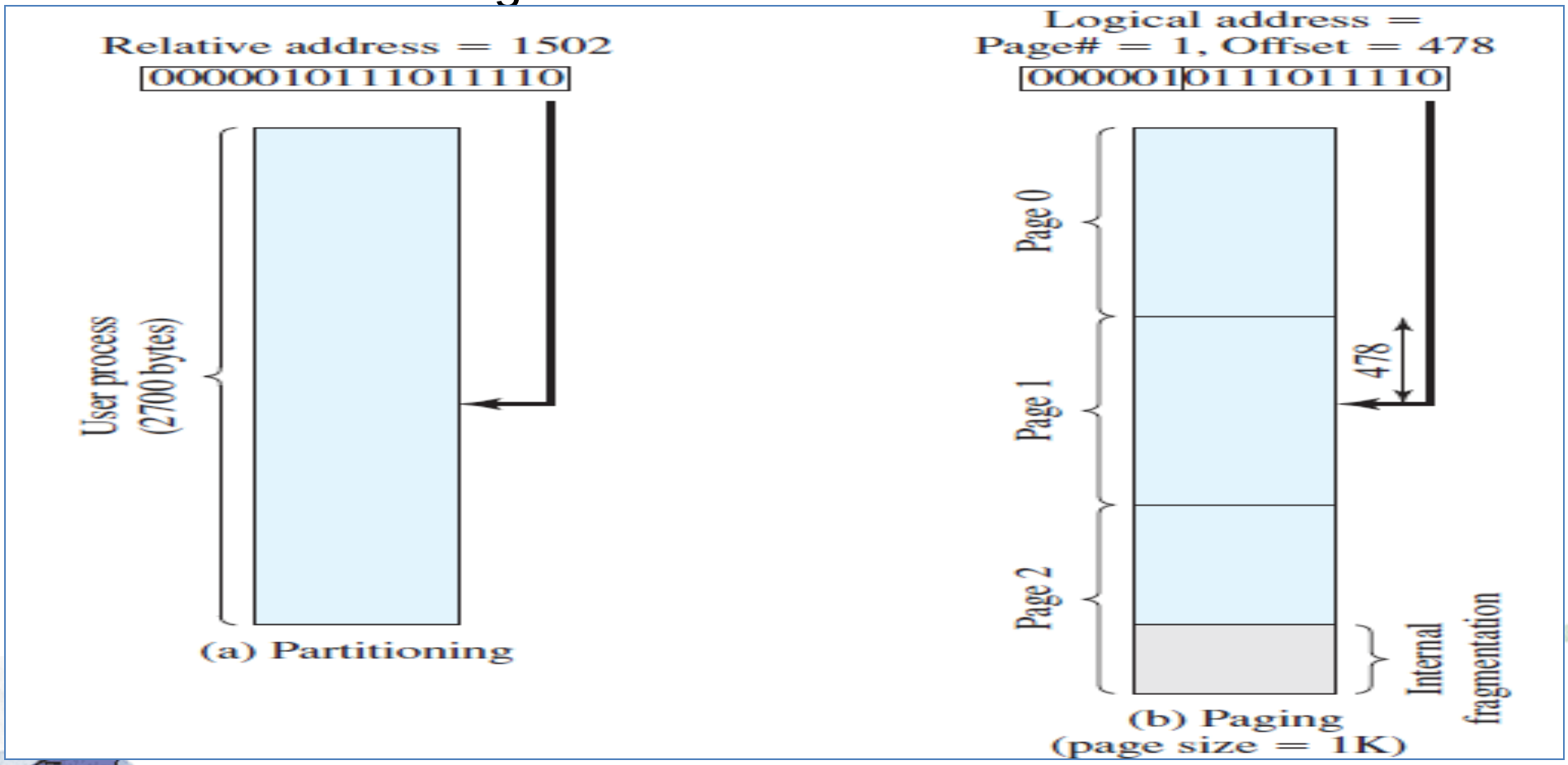
- Logical Address =31 bits
  - Address space = 2^1 * 2^30 (2 G)
- Address space=128M (2^7 * 2^20)
  - Logical Address= 27 bits
- Physical Address= 22 bits
  - Address space= 2^2 * 2^20 (4 M)
- Address space=16M (2^4 * 2^20)
  - Physical Address= 24 bits

Paging: Addresses

- Given physical address = 12 bits
- Logical address = 13 bits
- Page size = Frame size = 1 K
- Number of Pages?
- Number of Frames?
- Structure of Logical Address?
- Structure of Physical Address?

Paging: Addresses

- If page size and frame size are power of 2, then relative address and logical address are the same

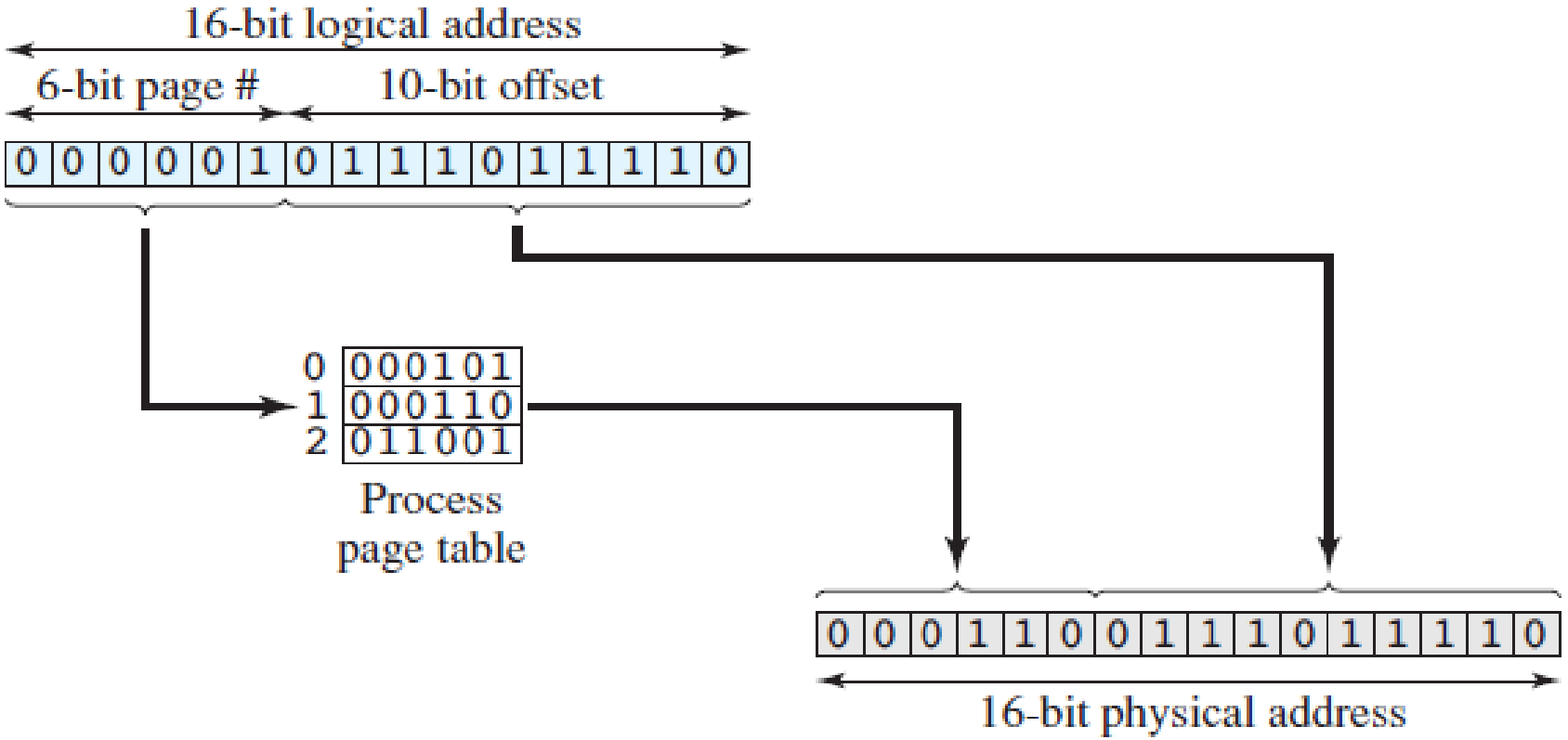


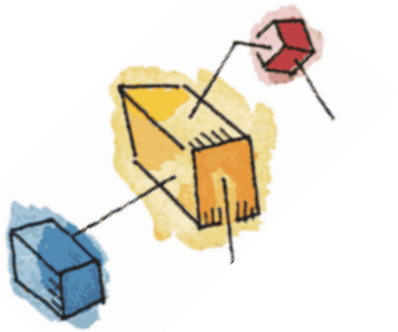
Paging: Address Translation

- Consider an address of n+m bits, where leftmost n bits are the page number and the rightmost m bits are the offset

- Address translation Steps:
  - Extract the page number as the leftmost n bits
  - Use the page number as an index into process page table to find frame number, k
  - Generate physical address by appending frame number with offset

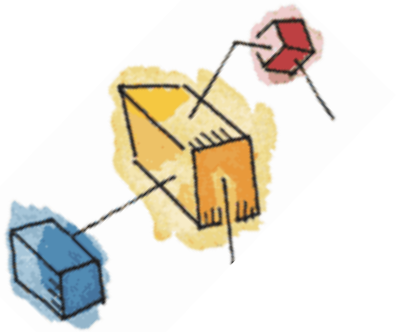
Paging: Address Translation





Paging: Summary

- Simple paging
  - Divide main memory into many small equal size frames
  - Divide process 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 page table is setup

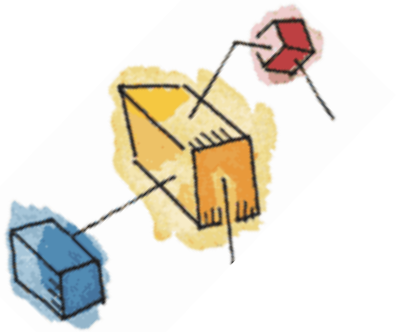


Segmentation

- A user program can be subdivided into segments
  - Segments may vary in length
  - There is a maximum segment length

- *Similar to dynamic partitioning due to unequal size segments*

- Addressing consist of two parts
  - a segment number and
  - an offset



Segmentation

- One segment table per process

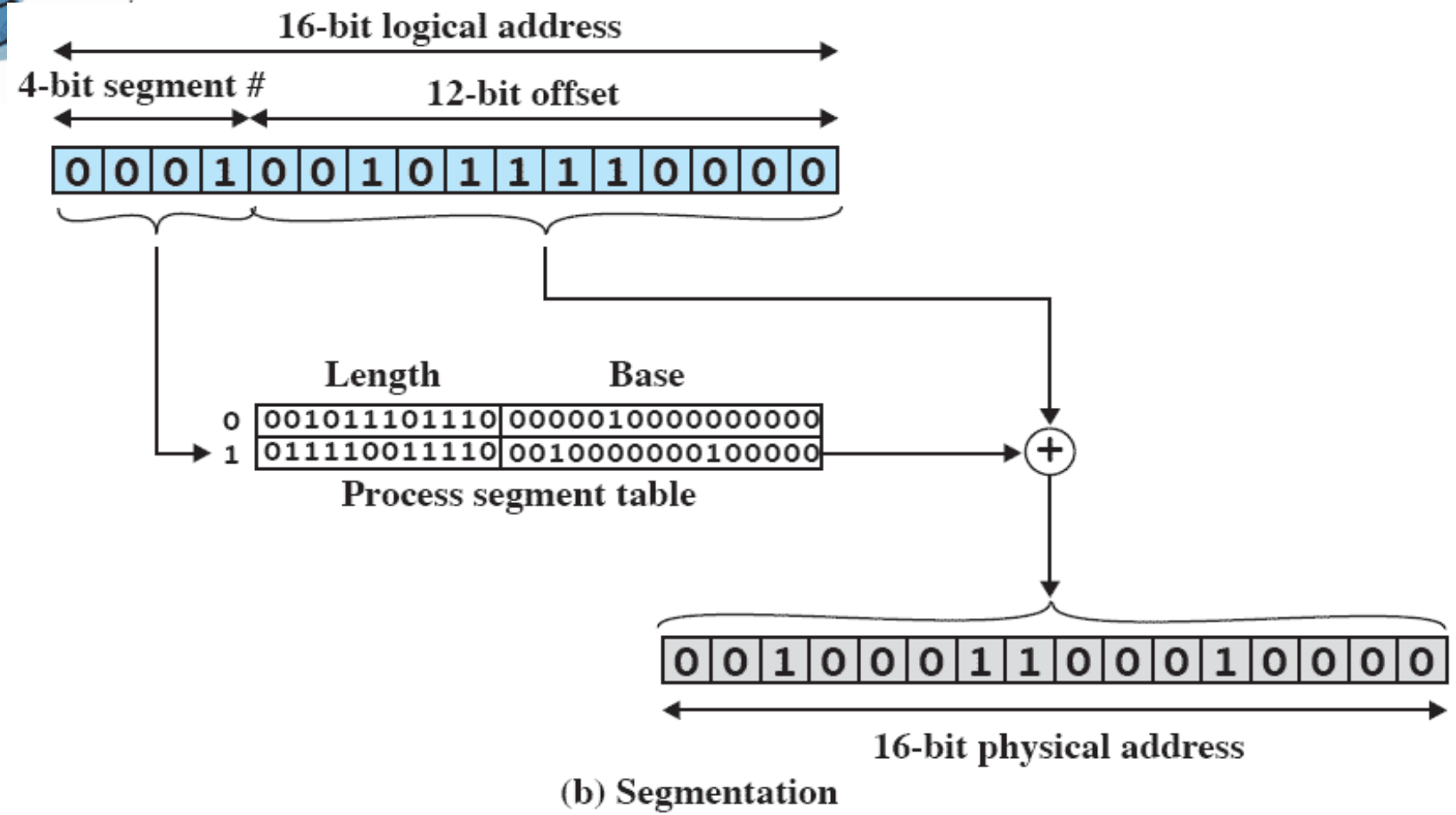
- Segment table entry
  - Starting address of segment in main memory
  - Length of segment

- When a process enters running state,
  - Address of segment table is loaded into main memory

Segmentation: Address Translation

- **Address translation Steps:**
  - Extract the segment number as the leftmost n bits
  - Use the segment number as an index into process segment table to find starting physical address of segment
  - Compare offset with the length, if offset greater than length, address is invalid
  - Generate physical address by appending starting address with offset

Segmentation



(b) Segmentation

Figure 7.12 Examples of Logical-to-Physical Address Translation

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	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.