953214
Operating System and Computer Network

Chapter 6

Memory Management

Lecturer: Dr. Phudinan Singkhamfu, Dr. Parinya Suwansrikham

College of Arts, Media and Technology
Chiang Mai University

Introduction

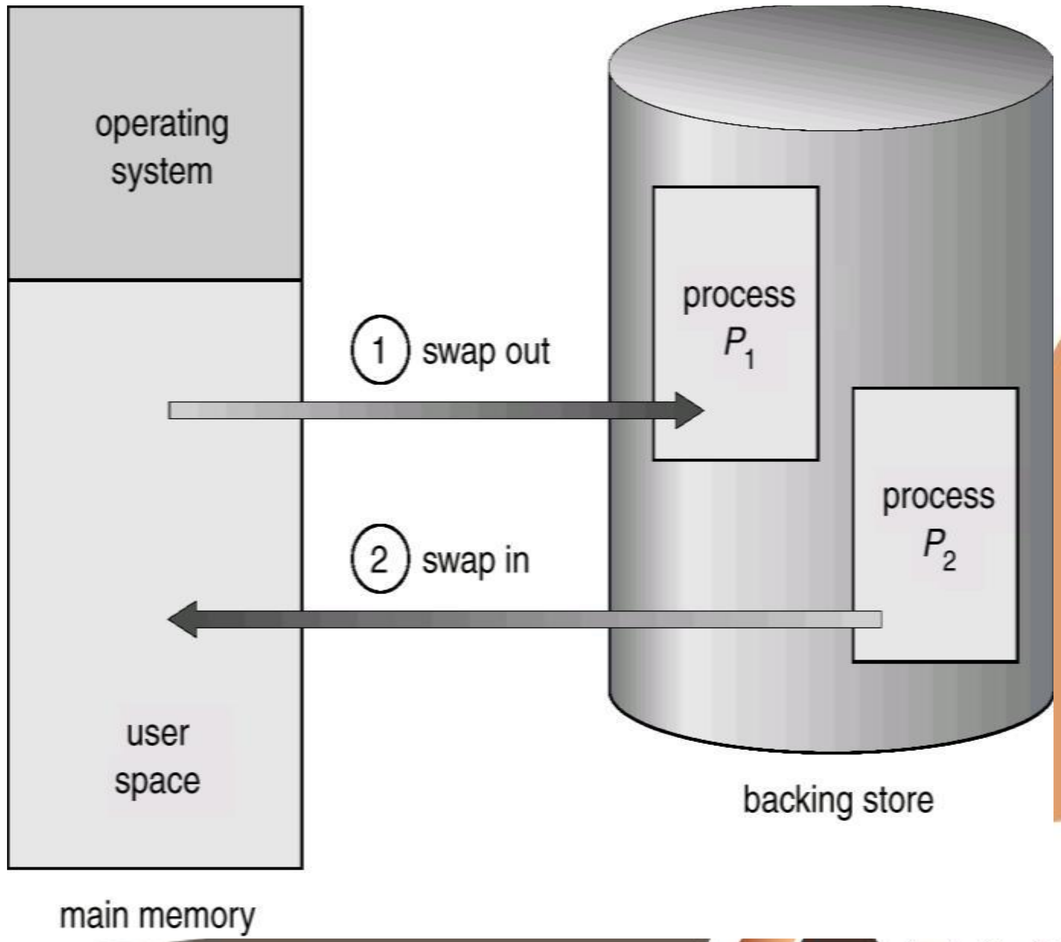
- In uni-programming system, main memory is
  - Operating system part
  - Program execution part
- In multiprogramming system, user part of memory is subdivided to accommodate multiple processes, the task of subdivision known as memory management

Memory Management Terms

Frame	*A fixed-length block of main memory*
Page	*A fixed-length block of data that resides in secondary memory (such as a 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).*

Process Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.
- Backing store –fast disk large enough to accommodate copies of all memory images for all users.
- Roll out, roll in –used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped.

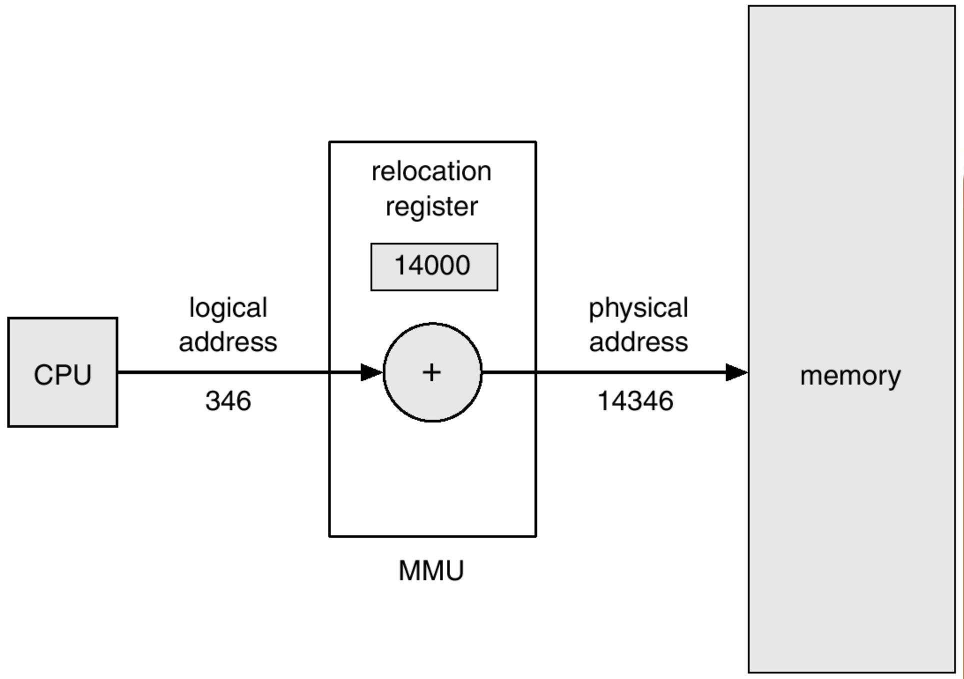


Logical vs. Physical Address Space

- The concept of a logical address space and physical address space is central to proper memory management.
  - Logical address –generated by the CPU; also referred to as virtual address.
  - Physical address –address seen by the memory unit.
- Logical (virtual) and physical addresses are the same in compile-time; but differ in execution-time.

Memory-Management Unit (MMU)

- Hardware device that maps logical address to physical address.
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
- The user program deals with *logical* addresses; it never sees the *real* physical addresses.



Overlays

- Large programs -too big to fit in physical memory
- Split along logical boundaries
  - Procedures, distinct functions
- Needed when process is larger than amount of memory allocated to it
- Keep in memory only those instructions and data that are needed at any given time
- One overlay in primary memory -rest on disc
- When branch is made to code on disc
  - Store current overlay on disc, read in new overlay
- Implemented by user, no special support needed from operating system

Memory Management

- Five Requirements for Memory Management to satisfy:
  - Relocation
    - Users generally don’t know where they will be placed in main memory
    - Once a program is swapped out to disk, it must be placed in the same main memory region as before. Instead, we may need to relocate the process to a different area of memory.
    - Generally handled by MMU
  - Protection
    - Prevent processes from interfering with the OS or other processes
  - Sharing
    - Allow processes to share data/programs
  - Logical Organization
    - Support modules, shared subroutines
    - Most programs are organized into modules, some of which are unmodifiable (read only, execute only) and some of which contain data that may be modified.
  - Physical Organization
    - Manage memory-to-disk transfers

Fragmentation

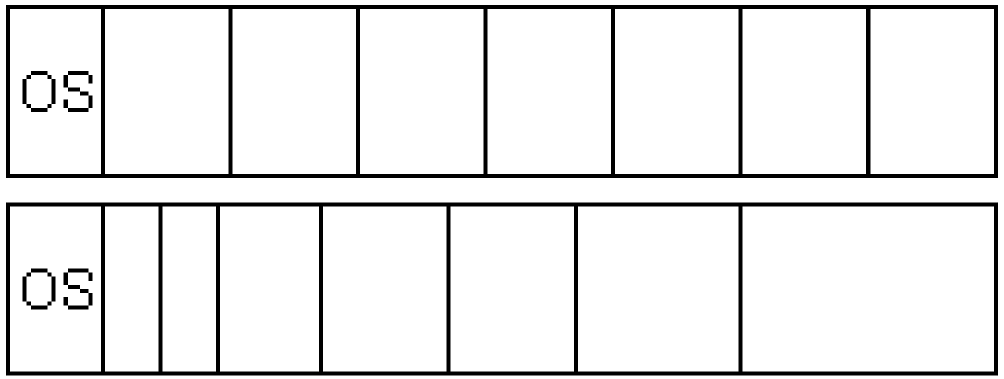
- External Fragmentation –total memory space exists to satisfy a request, but it is not contiguous.
- Internal Fragmentation –allocated memory may be slightly larger than requested memory due a partition, but not being used.
- Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block.
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time.

Memory Management Techniques

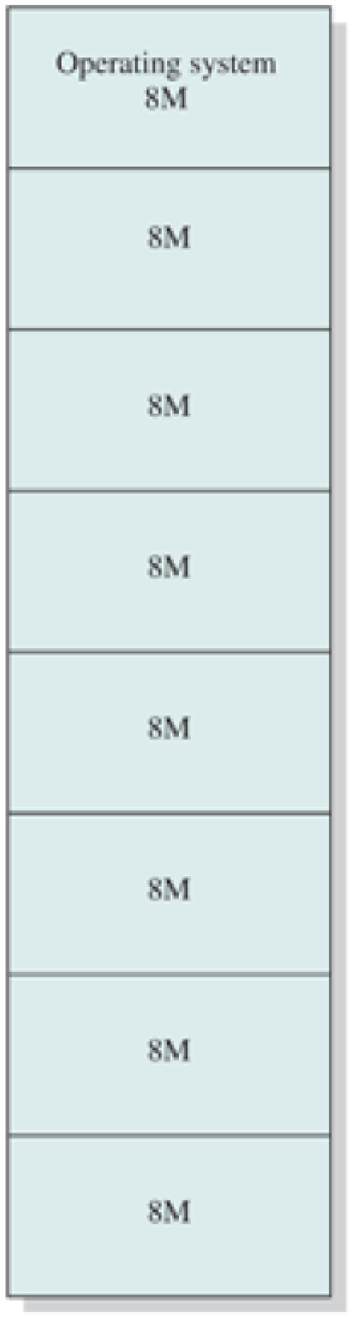
- Fixed Partitioning
- Dynamic Partitioning
- Buddy System
- Simple Paging
- Simple Segmentation
- Virtual-Memory Paging
- Virtual Memory Segmentation

Fixed Partitioning

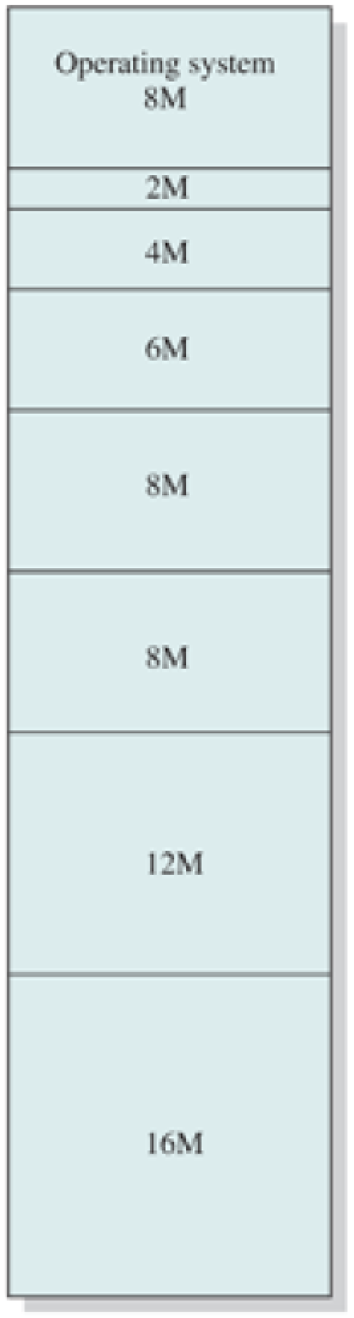
- Memory is divided into a set of partitions, each holds one program
- May have same or different sizes



- Equal-sized partitions
  - Program may be too large for one partition
    - Programmer must handle overlays
  - Program may not be big enough to need a full partition
- Results in internal fragmentation



(a) Equal-size partitions



(b) Unequal-size partitions

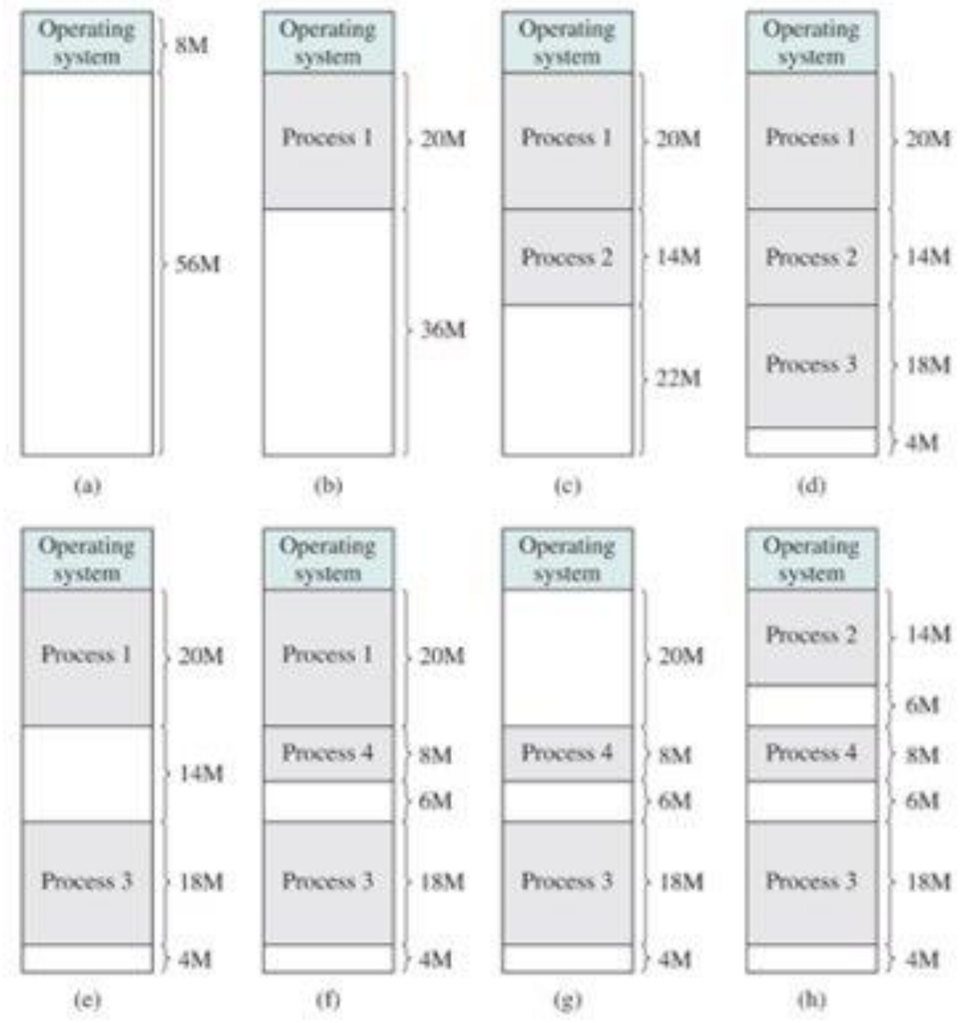
Fixed Partitioning

- Variable-sized partitions
  - Can fit partition size to program size
  - Placement
    - Want to use smallest possible partition
    - Can have a queue for each partition size, or one queue for all partitions
- Easy to implement
  - Tends to use memory poorly, especially for small programs (internal fragmentation)
  - Will work if the job sizes can be predicted in advance

Dynamic Partitioning

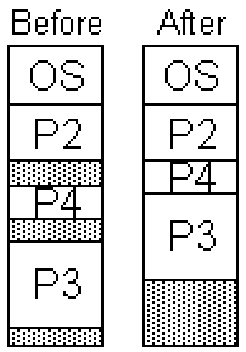
- Create a new partition for each program
- Only allocate space required
- Tends to create "holes" as processes are swapped in and out (external fragmentation)

Dynamic Partitioning



Dynamic Partitioning

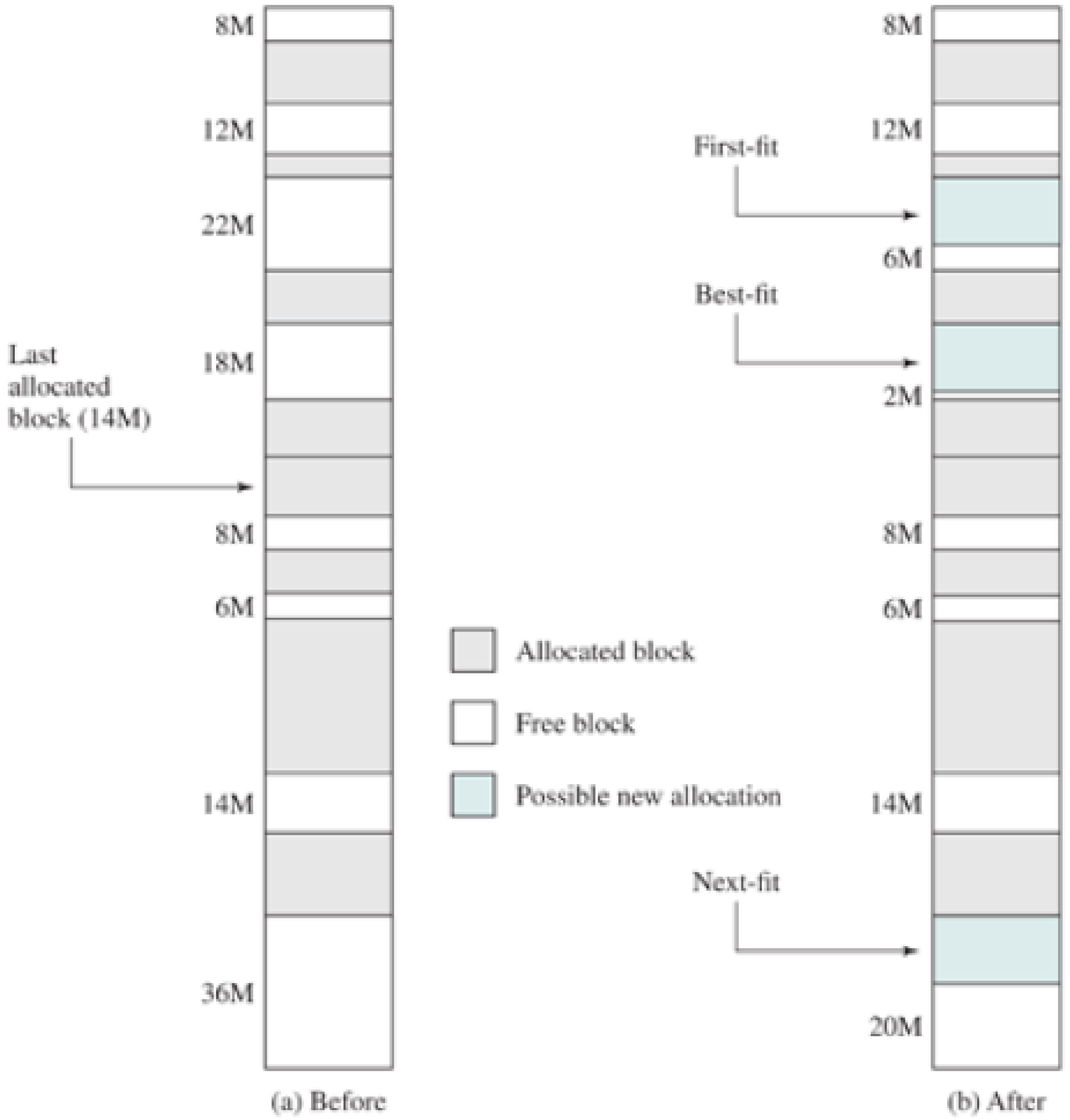
- Tends to generate small holes in memory (external fragmentation)
- Compaction
  - Shifting processes so all of the available memory is in one block



- Requires processor time to move items around in memory
- Requires relocation hardware that can handle the change in addresses
  - Cannot easily find all possible addresses in the program
  - Need to move program without changing user addresses

Dynamic Partitioning

- Placement Algorithms
  - **Best-Fit**: Find the smallest available block that will hold the program
    - Tends to produce a lot of small blocks, e.g. use 30K block for 28K program, leaves 2K
  - **First-Fit**: Find the first block that is large enough (regardless of size)
    - May leave small blocks at the beginning, larger blocks at the end of memory
    - Generally best, fastest
  - **Next-Fit**: Like First-Fit, but start from the last allocation instead of the start
    - Tends to break up large blocks at the end of memory that First-Fit leaves alone
- Worst-Fit–Allocate the largest hole


8M
12M
22M
18M
Last allocated block (14M)
8M
6M
14M
36M
(a) Before
Allocated block
Free block
Possible new allocation
8M
12M
First-fit
6M
Best-fit
2M
8M
6M
14M
Next-fit
20M
(b) After

Buddy System

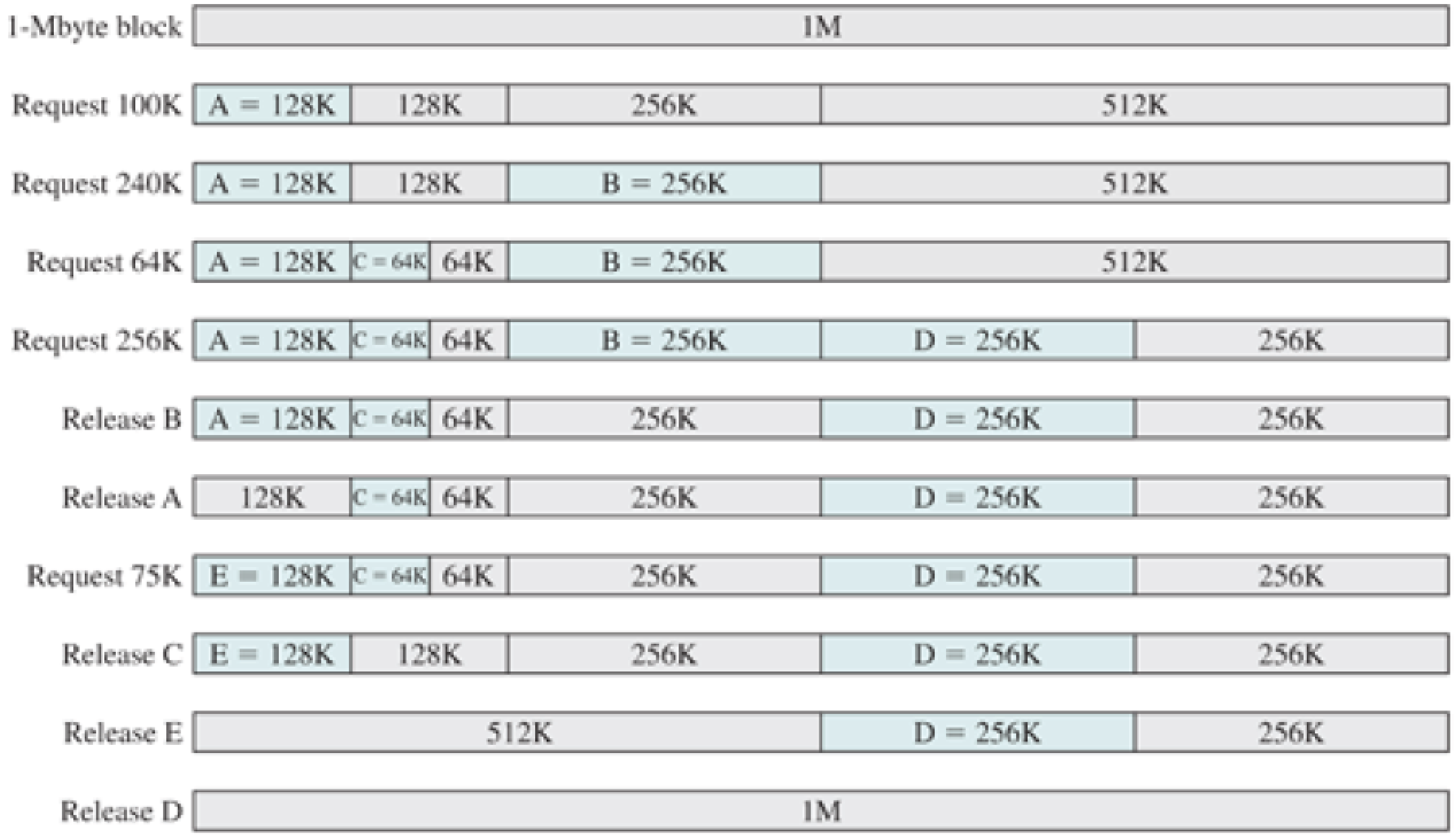
- Tries to allow a variety of block sizes while avoiding excess fragmentation
- Blocks generally are of size 2^k, for a suitable range of k
- Initially, all memory is one block
- All sizes are rounded up to 2^s
- If a block of size 2^s is available, allocate it
- Else find a block of size 2^{s+1} and split it in half to create two buddies
- If two buddies are both free, combine them into a larger block
- Largely Replaced by paging

Buddy System

- Memory blocks are available of size 2^K words, $L \le K \le U$
  - Where 2^L = smallest size block that is allocated
    2^U = largest size block that is allocated; generally, 2^U is the entire memory
- Initially, all memory is one block, 2^U

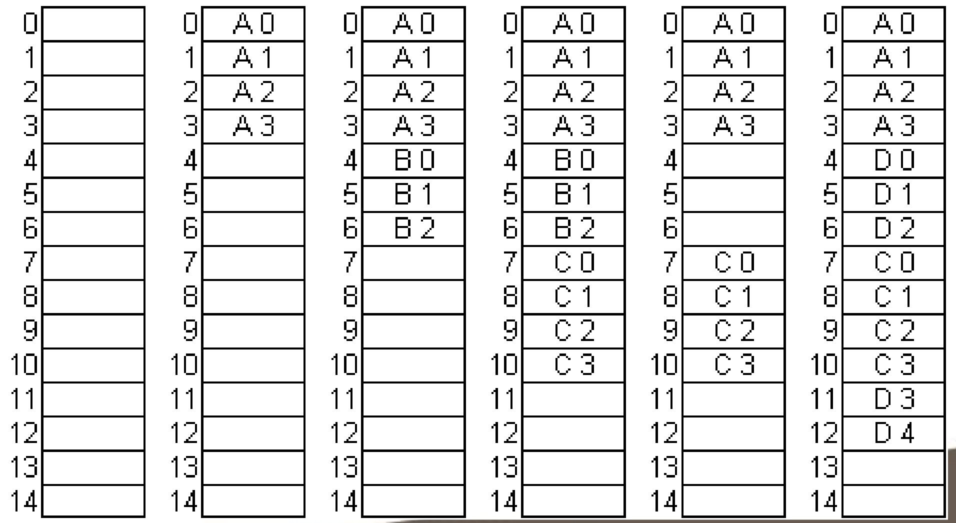
- If a request of size s, $2^{U-1} < s \le 2^U$,
  - then the request is allocated to one of the two buddies.
- else
  - one of buddies is split half.
- Keep running until the smallest block $\ge s$

Buddy System Example



Paging

- Divides memory into small (4K or less) pieces of memory (*frames*)
- Logically divide program (process) into same-size pieces (*pages*)
- Page size typically a power of 2 to simplify the paging hardware
- Use a *page table* to map the pages of the current process to the corresponding frames in memory

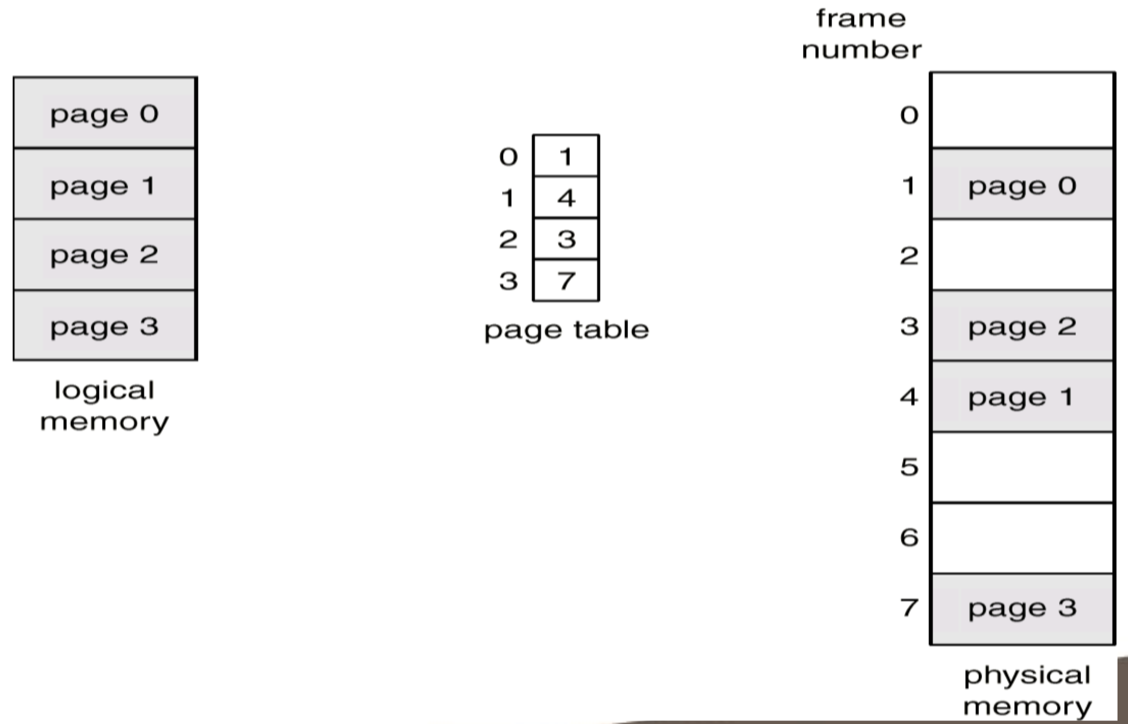

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
A 0 A 1 A 2 A 3
A 0 A 1 A 2 A 3 B 0 B 1 B 2
A 0 A 1 A 2 A 3 B 0 B 1 B 2 C 0 C 1 C 2 C 3
A 0 A 1 A 2 A 3 C 0 C 1 C 2 C 3
A 0 A 1 A 2 A 3 D 0 D 1 D 2 C 0 C 1 C 2 C 3 D 3 D 4

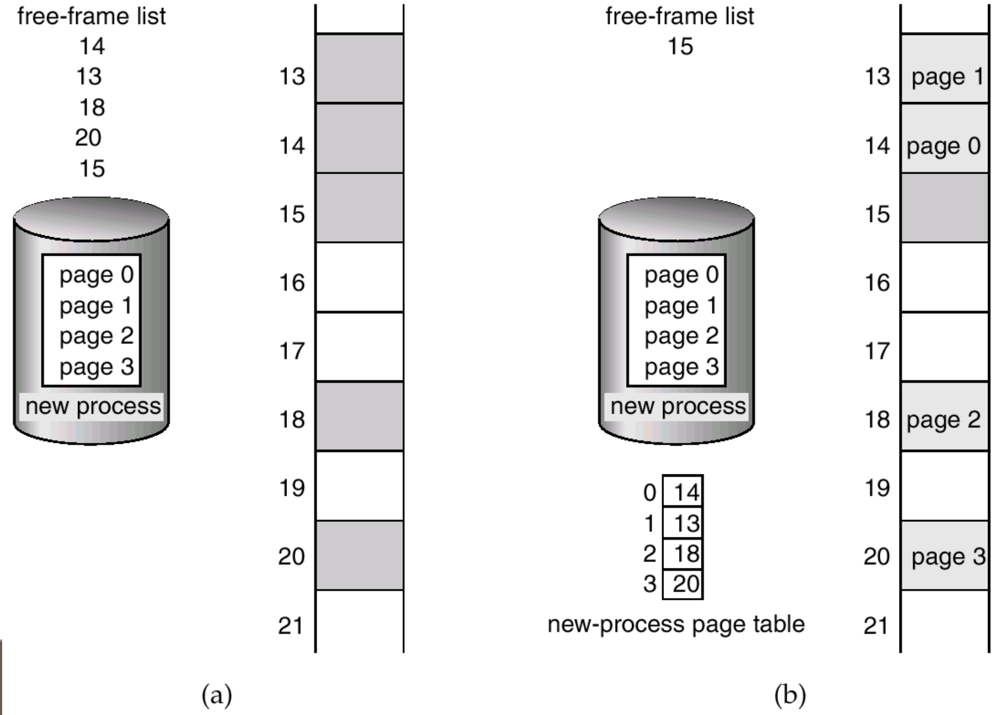
Page Table

A	B	C	D	Free
0	-	7	4	13
1	-	8	5	14
2	-	9	6	
3		10	11	
			12	

Page Table and Free Frames

- Set up a page table to translate logical to physical addresses
- Keep track of all free frames
- To load a program of size *n* pages, need to find *n* free frames

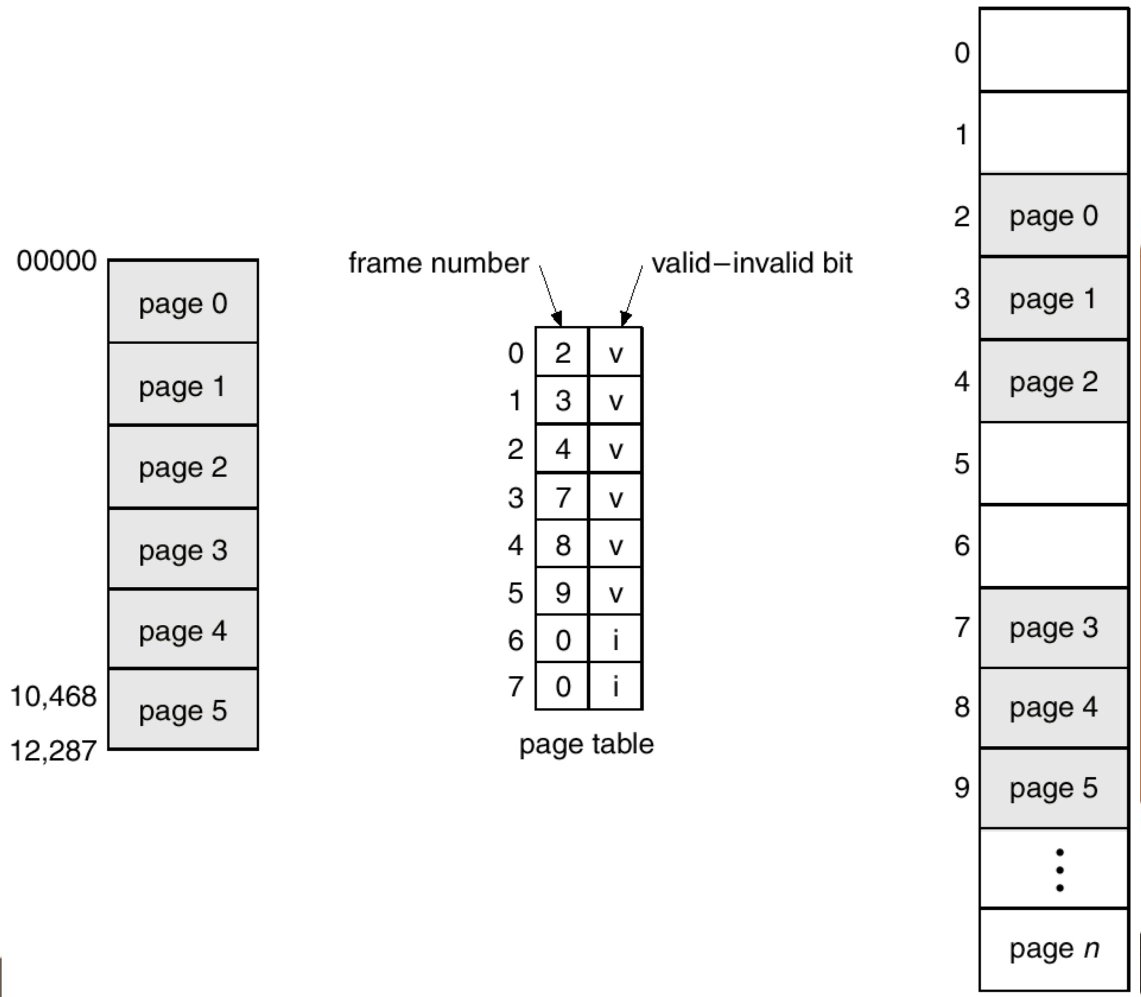




Before allocation After allocation

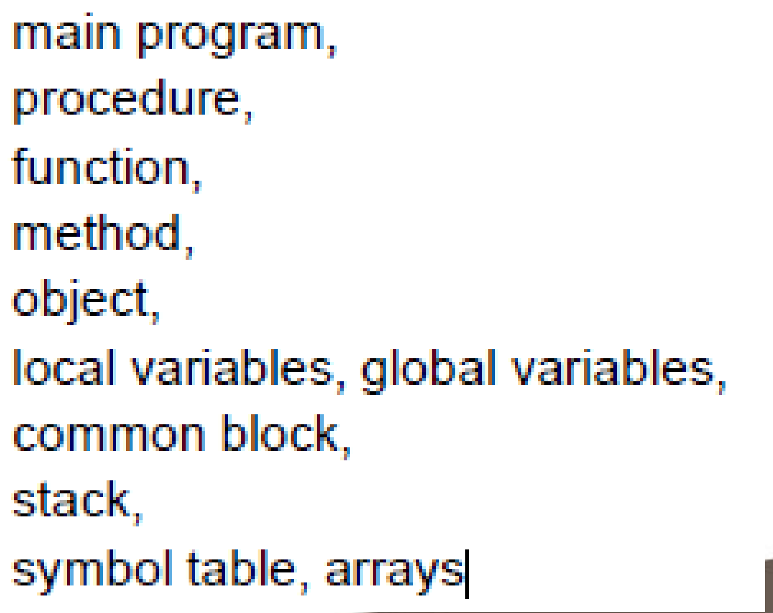
Implementation of Page Table

- Page table is kept in main memory.
- *Page-table base register (*PTBR) points to the page table.
- *Page-table length register*(PRLR) indicates size of the page table
- Use of valid (v) or invalid (i) bit in a page table



Segmentation

- Memory-management scheme that supports user view of memory.
- A program is a collection of segments. A segment is a logical unit such as:



main program,
procedure,
function,
method,
object,
local variables, global variables,
common block,
stack,
symbol table, arrays

Segmentation

- Program views memory as a set of segments of varying sizes
  - Easy to handle growing data structures
  - Easy to share libraries, memory
  - Privileges can be applied to a segment
  - Programs may use multiple segments
- Have a segment table
  - Beginning address
  - Size
  - Present, Modified, Accessed bits
  - Permission/Protection bits

Summary

- Process can be swapped out of and back into memory
- MMU is a hardware device that maps virtual to physical address
- Overlays needed when process is larger than amount of memory allocated to it
- There a number of partitioning algorithms
- Paging divides memory into small pieces called frames and program into pages