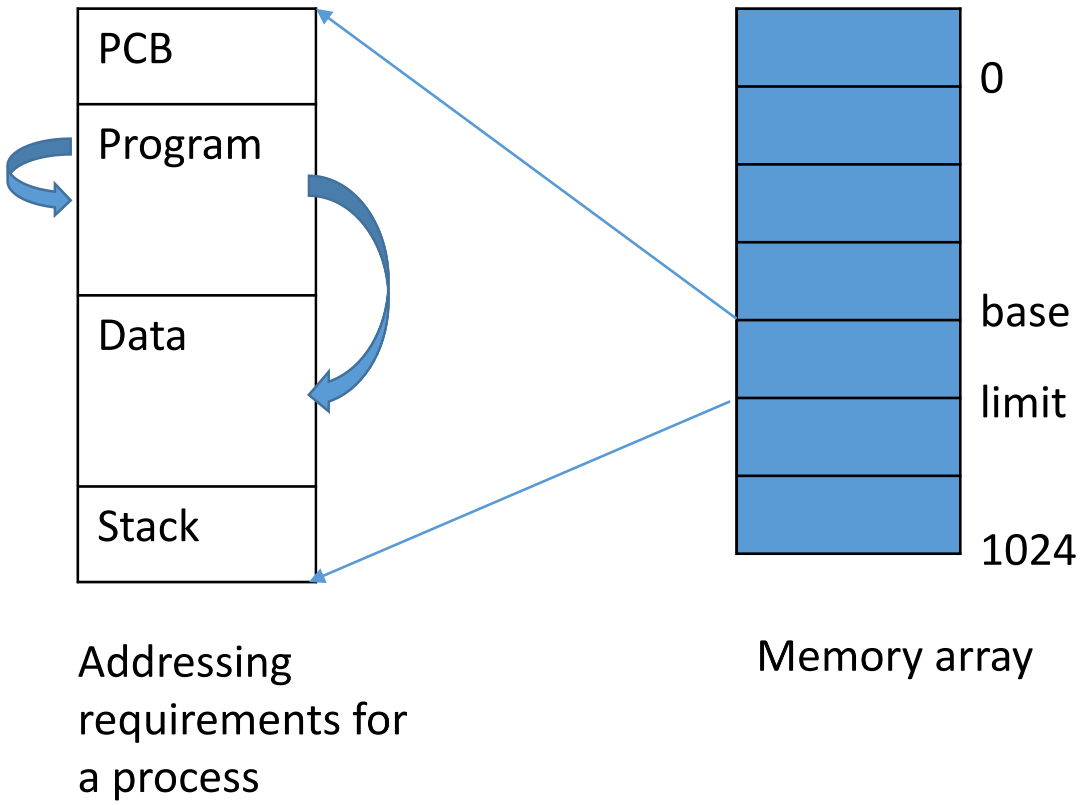
Agenda

- Assignment 3 due Friday November 10
- Today's lecture
  - Memory management
  - Swapping
  - Partitioning
  - Paging
  - Segmentation
  - Textbook Reading: 8.1-8.4, 8.6
- Next lecture
  - Virtual Memory

What is Memory?

- Computer memory consists of a linear array of addressable storage cells that are similar to registers
- Program must be brought (from disk) into memory and placed within a process for it to be run
  - Each process has a base/limit address to specify its accessible addresses
  - ”segmentation fault” error generated when trying to access memory outside of the allowable address space



Memory Management Goals

- Make sure each process has sufficient memory
- Keep as many processes in memory as possible
- Allocate memory efficiently
  - Allocate as much as is needed for a process – not more
  - Leave some free space for new starting processes
  - Maximize memory utilization
- Memory is a finite resources
- These goals cannot be simultaneously met

OS Strategies

- Swapping
- Segmentation
- Paging
- Virtual memory (next lecture)

Terminology

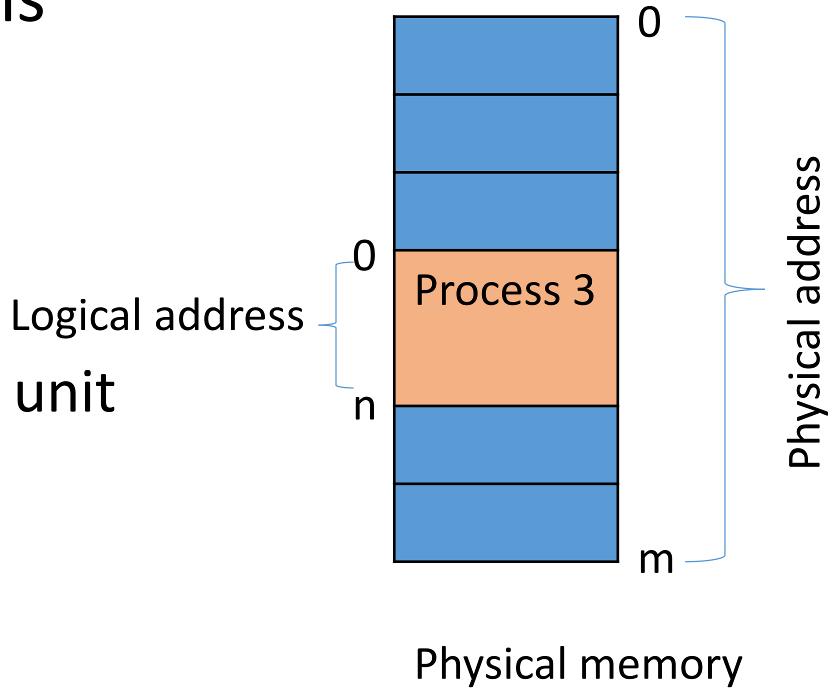
- **Frame**
  - A fixed-length block of main memory
- **Page**
  - A fixed-length block of data residing in secondary memory
- **Paging**
  - A page of data may temporarily be copied into a frame of main memory
- **Segment**
  - A variable-length block of data residing in secondary memory
- **Segmentation**
  - An entire segment may temporarily be copied into a region of main memory
- **Segmentation and Paging**
  - An entire segment may be divided into pages
  - Individual pages temporarily copied into main memory

Memory Management Requirements

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

Physical vs Logical Address Spaces

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU;
    - Referred to as **virtual address**
    - Address space visible to a process (0-n)
  - **Physical address** – address seen by the memory unit
    - Address range of the physical memory (0-m)
- In primitive systems, the physical and logical address spaces were the same

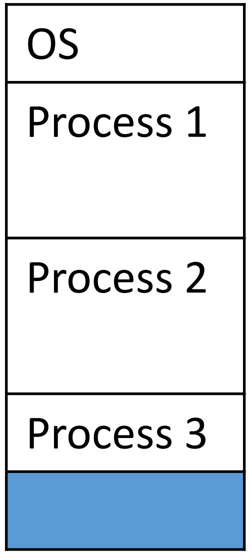


Address Translation

- The user program deals with *logical* addresses; it never sees the *real* physical addresses
- Address translation is done at run-time by the OS Memory Management Unit (MMU)
  - Defines the **logical address** space of each process (including OS)
    - A **relative address** is a particular example of logical address,
    - expressed as a location relative to some known point (a value in a processor register)
  - Maps the logical address space to the **physical address (absolute address)** space
  - Generates an interrupt if an invalid logical address is issued

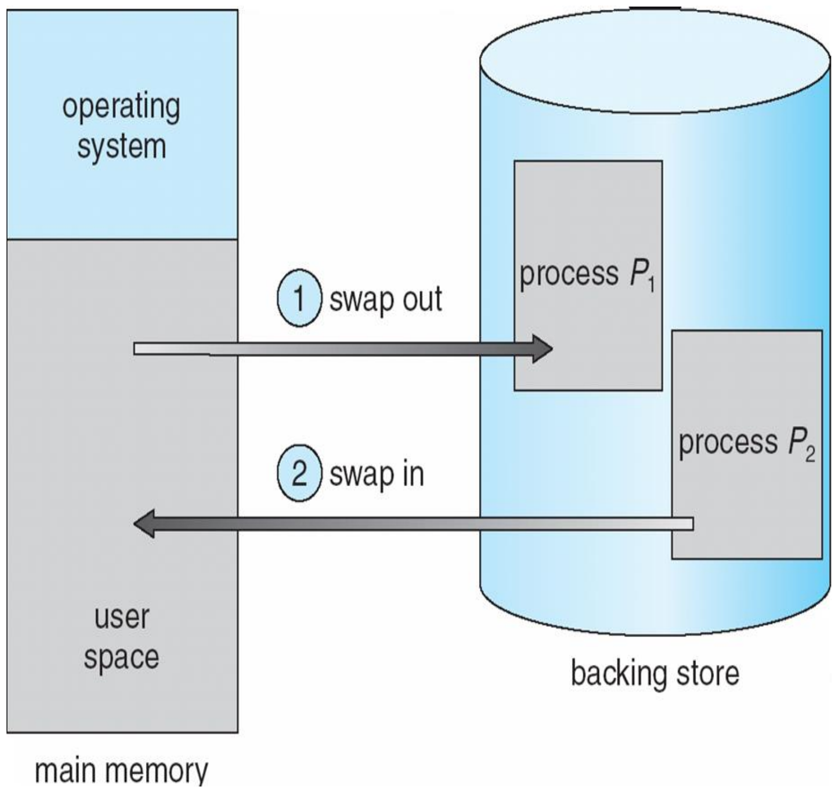
Logical Organization – Why?

- Each process has its own view (same) of the memory
  - Pointers and references are the same
  - Code is easier to generate and load
- Memory can be allocated to more than one process
- Processes can be loaded anywhere in the memory



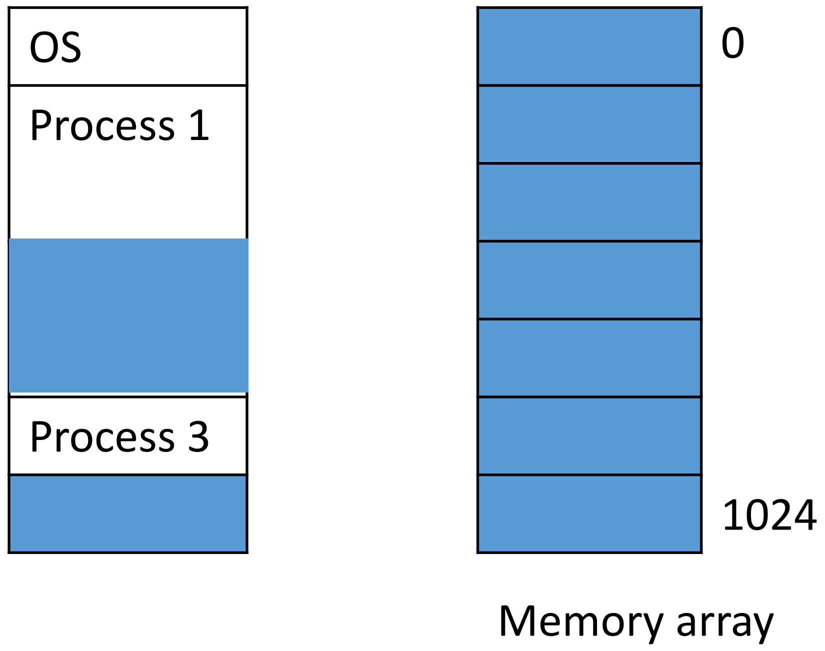
Swapping

- When we run out of main memory, we can temporarily store some program parts in a backing store (on disk) and thus free some memory space
- When we need them again they are copied back into main memory
- This movement between main memory and disk is called **swapping**
- Variants of swapping are widely used in Oss



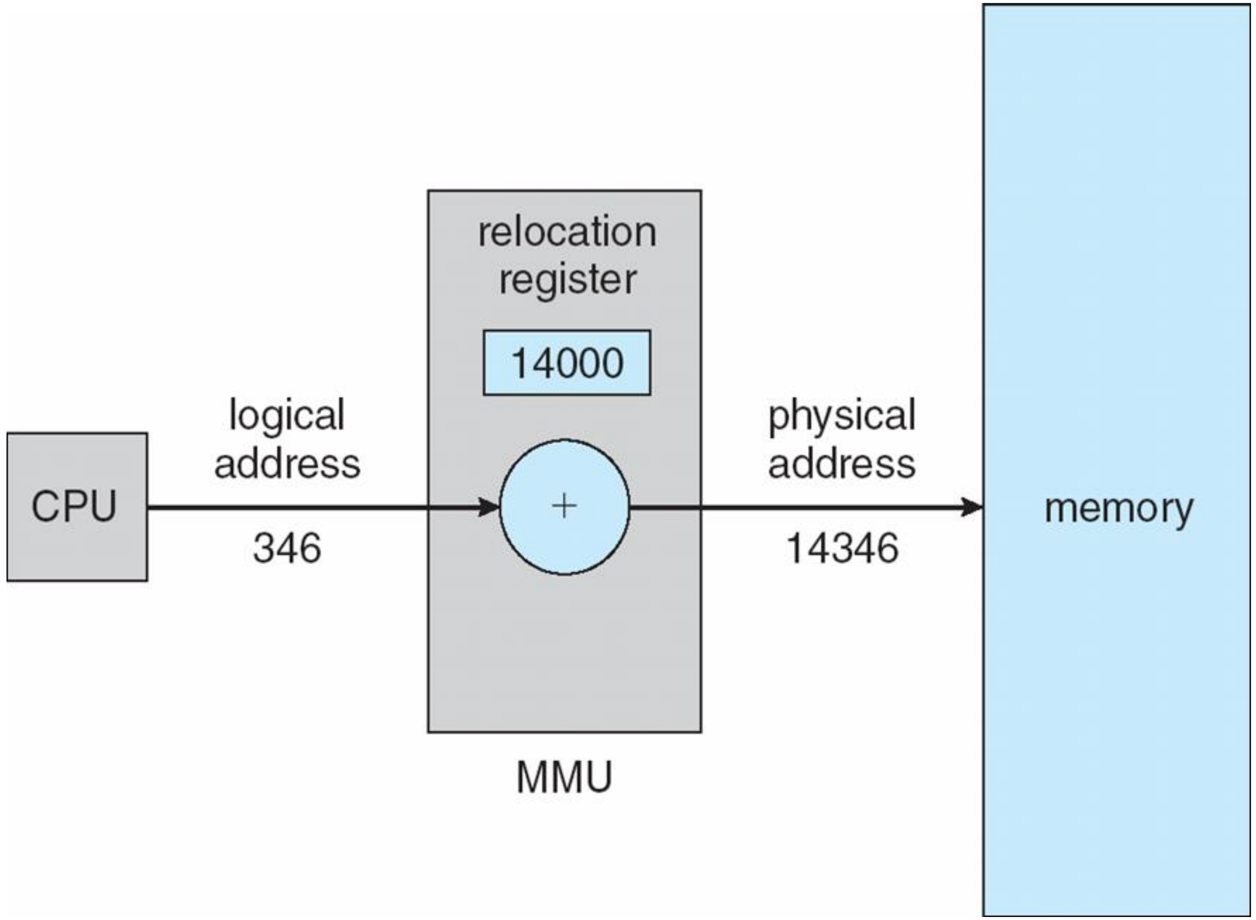
Relocation

- Active processes need to be swapped in and out of main memory to maximize processor utilization
- Specifying that a process must be placed back in the same memory would be limiting
  - **Relocation**: ability to place the process in a different area of memory



Dynamic Relocation

- Virtual addresses are offsets
- Relocation register provides the base address and is set when the program is loaded
- The MMU adds offset to base to get physical address



Protection

- Processes need to acquire permission to reference memory locations for reading and writing
  - User process cannot access any portion of the OS
  - Program in one process cannot branch to an instruction in another process
- Location of a program in main memory is unpredictable
- Memory references generated by a process must be checked at run time
  - e.g., computing an array subscript or a pointer into a data structure
- Mechanisms that support relocation also support protection
  - Requires hardware support

Sharing

- Allow several processes to access the same portion of main memory
  - E.g, when several processes are executing the 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
- Mechanisms used to support relocation support sharing capabilities

Dynamic Loading

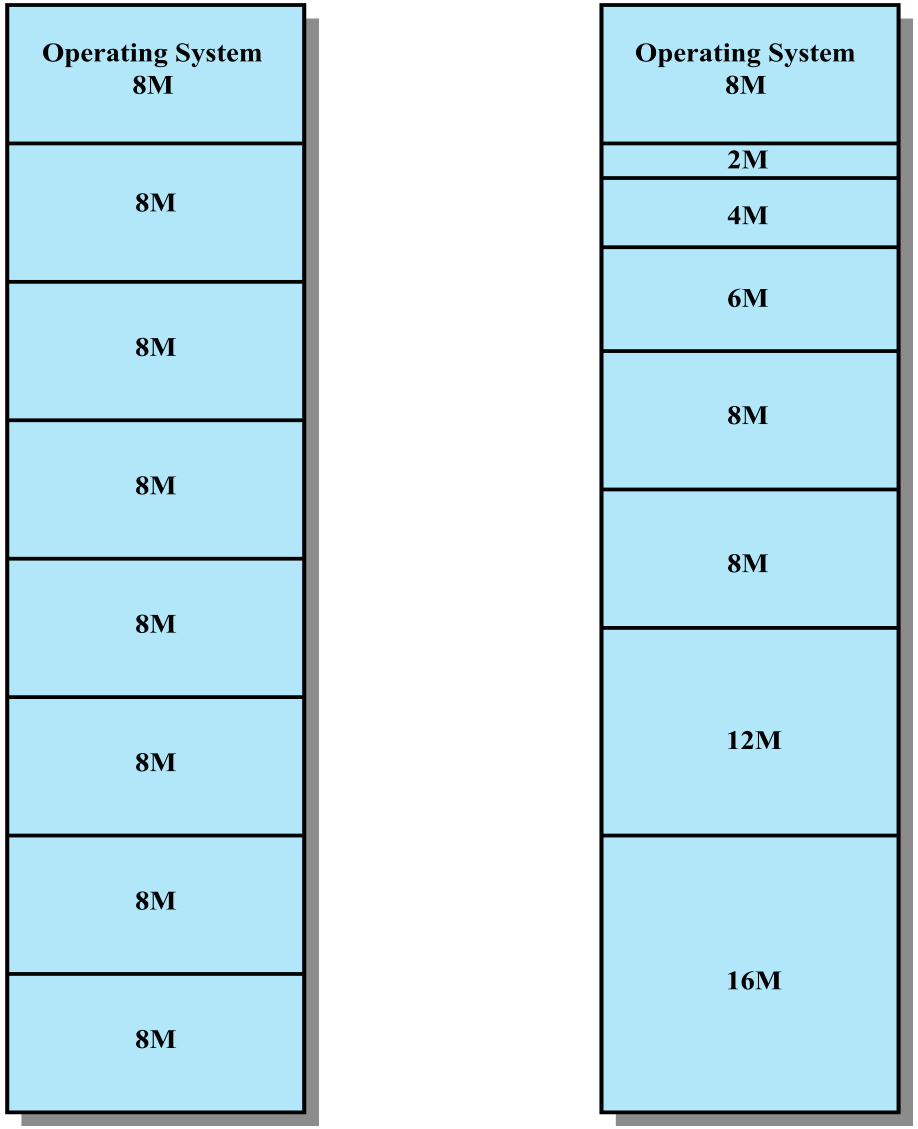
- Ability to load program parts and pieces “on demand”
- Speeds up initial loading
- Saves memory by not loading unneeded parts
- Can be augmented with dynamic linking
- So how can we manage memory while meeting those requirements?

Memory Management Techniques

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

Fixed Partitioning

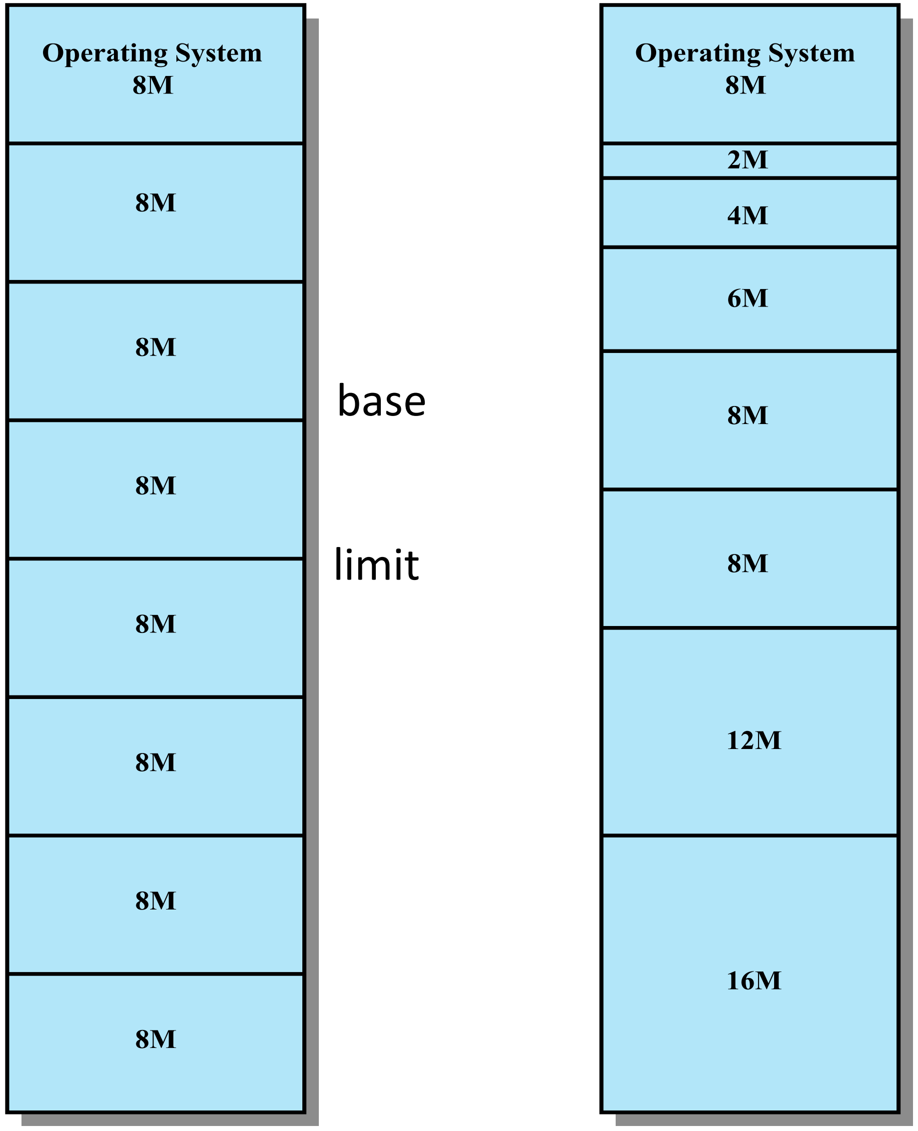
- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
  - Each process resides in its own partition



(a) Equal-size partitions (b) Unequal-size partitions

Fixed Partitioning

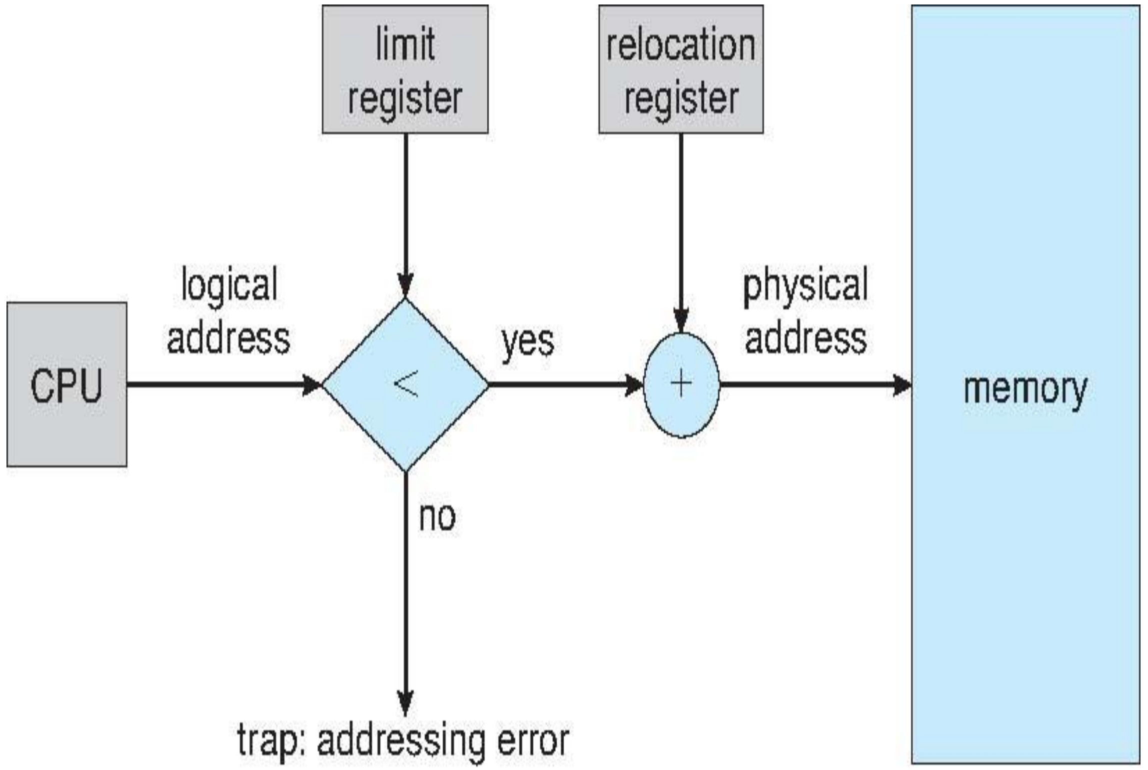
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - *Base register* contains value of smallest physical address
  - *Limit register (bounds)* contains range of logical addresses
  - Values set by OS during a process switch
  - Processes generate logical addresses (between 0 and limit-1)
  - MMU maps logical address *dynamically*



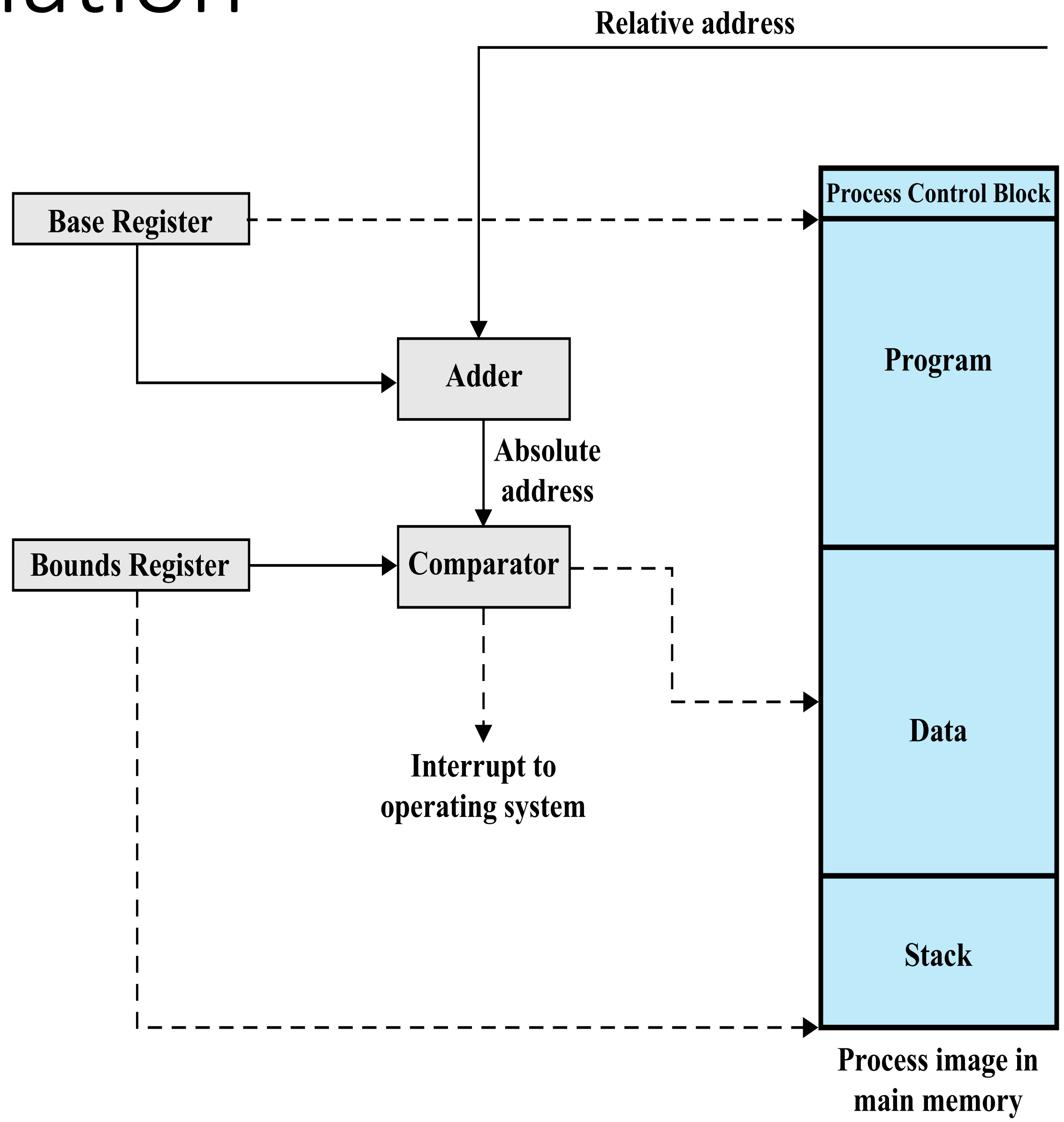
(a) Equal-size partitions

(b) Unequal-size partitions

Address Translation

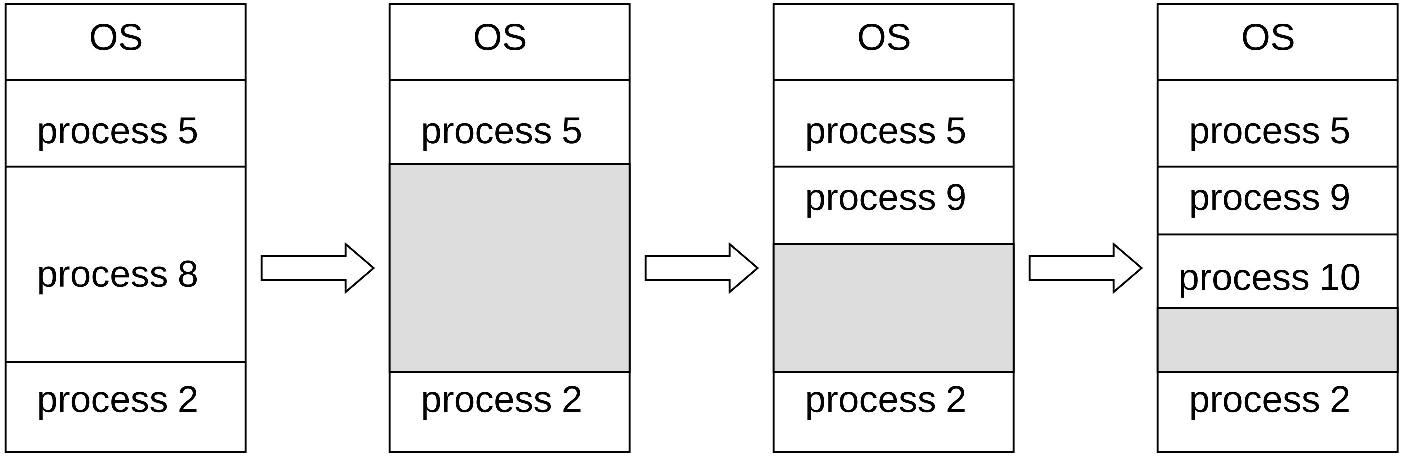


Address Translation



Fixed Partitioning – Placement Algorithm

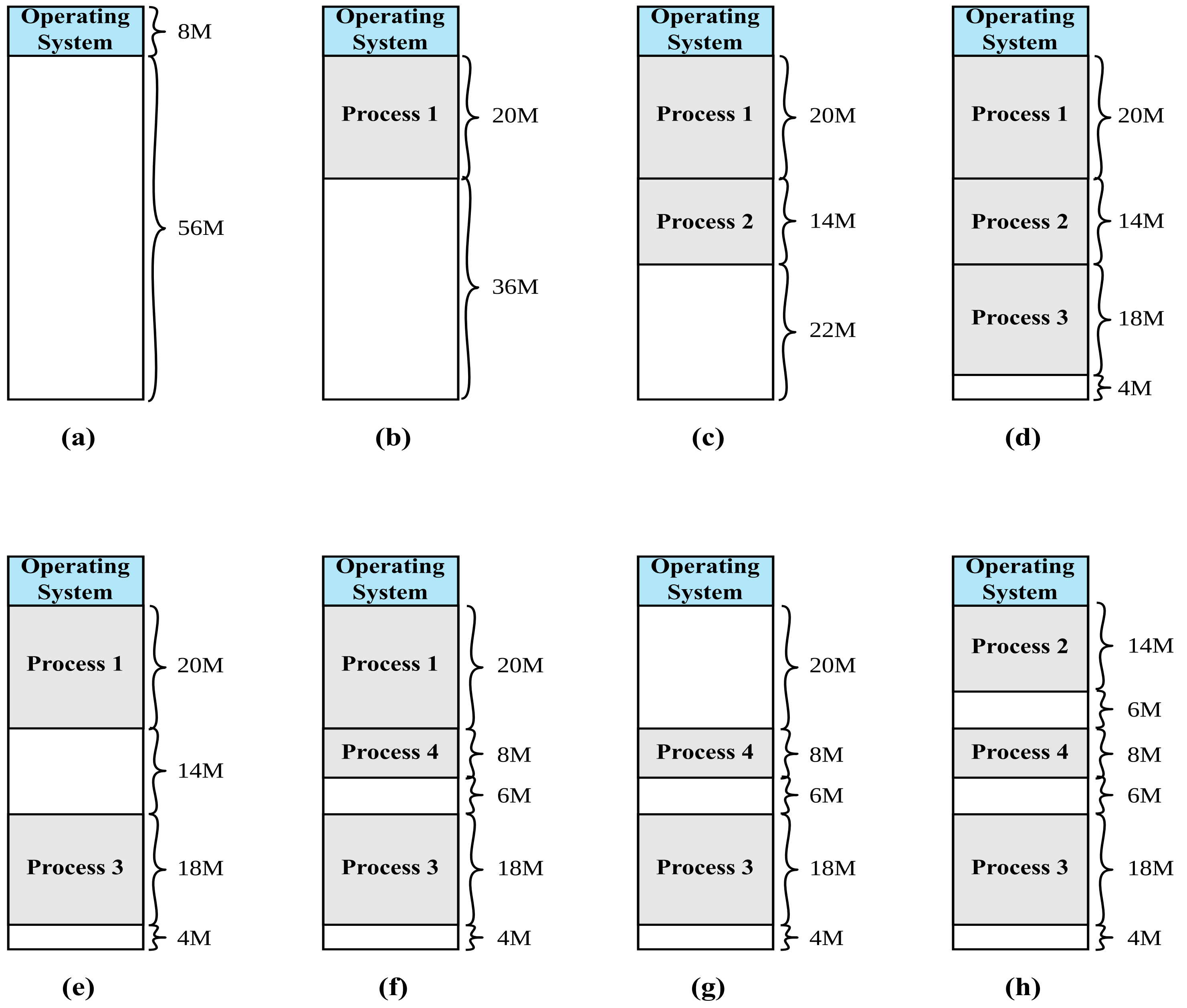
- Multiple-partition allocation
  - Hole – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Operating system maintains information about:
    a) allocated partitions   b) free partitions (hole)



Fragmentation

- As the system runs processes are created and destroyed
  - At the process level, memory is allocated and deallocated from the heap
- Over time, holes (small chunks of memory) build up in the free pool
- Small holes cannot be used, wasting memory
- This is called ***external fragmentation***

Fragmentation Example



Fixed Partitioning

- Simple, but limited (now obsolete)
    - limits the number of active (not suspended) processes in the system
- Good for embedded OS if the running applications change rarely and have known sizes
    - Fast translation and process switching
- Leads to external fragmentation
    - Small holes in the memory after processes are swapped in and out
    - A lot of memory can be waste!
- Can we do better?

Dynamic Partitioning

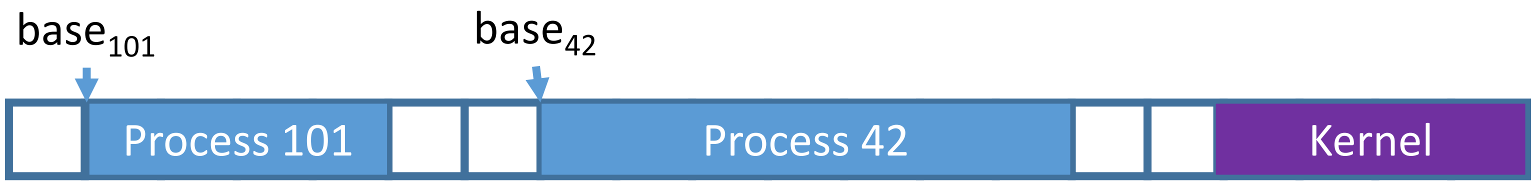
- Divide memory into variable size partitions
  - E.g., 8K, 64K, 256K, 1M, 4M, 16M, …
- Processes are allocated into partitions that are close in size
- Round size of a memory request up to the smallest allowable block
  - 50K → 64K
  - 65K → 256K
  - 17K → 64K
- No external fragmentation, but ***internal fragmentation*** is an issue
  - there is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
  - Memory utilization declines
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- Reduce external fragmentation by **compaction**

Compaction

- Shuffle memory contents to place all free memory together in one large block
  - Change the *base* value for each of them as they are moved
  - Possible *only* if relocation is dynamic, and is done at execution time
- Costs: This is expensive O(n) in size of physical memory
- Can we do better?



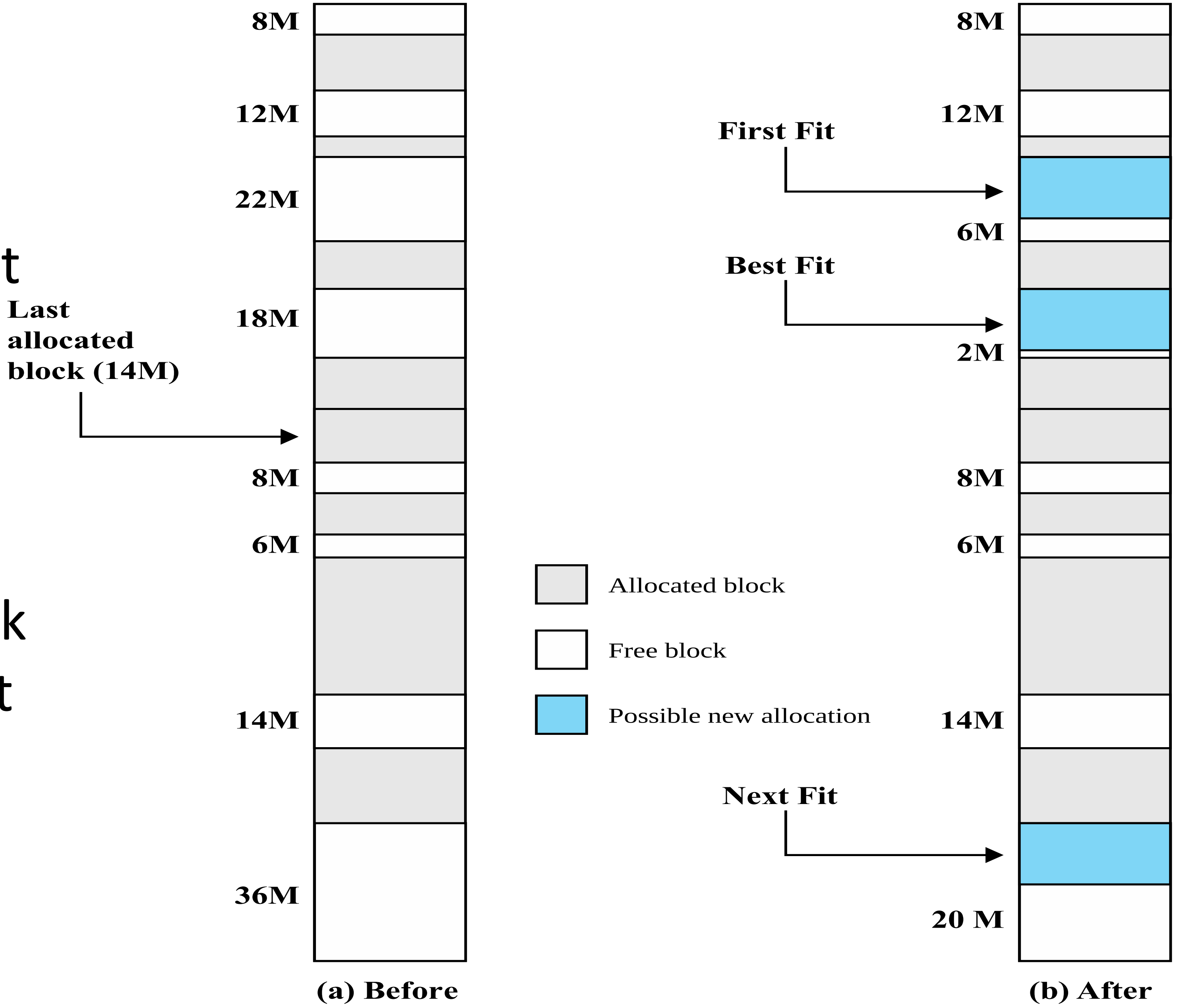
Placement Algorithms

- How to satisfy a request of size *n* from a list of free holes?
- **First-fit**: Allocate the *first* hole that is big enough
  - Not necessarily the best decision
- **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Next-fit**: Allocate the *next* available hole; starting from the last placement
  - Compaction may be required more frequently
- First-fit and best-fit better than Next-fit in terms of speed and storage utilization

Example

A 16-Mb allocation request

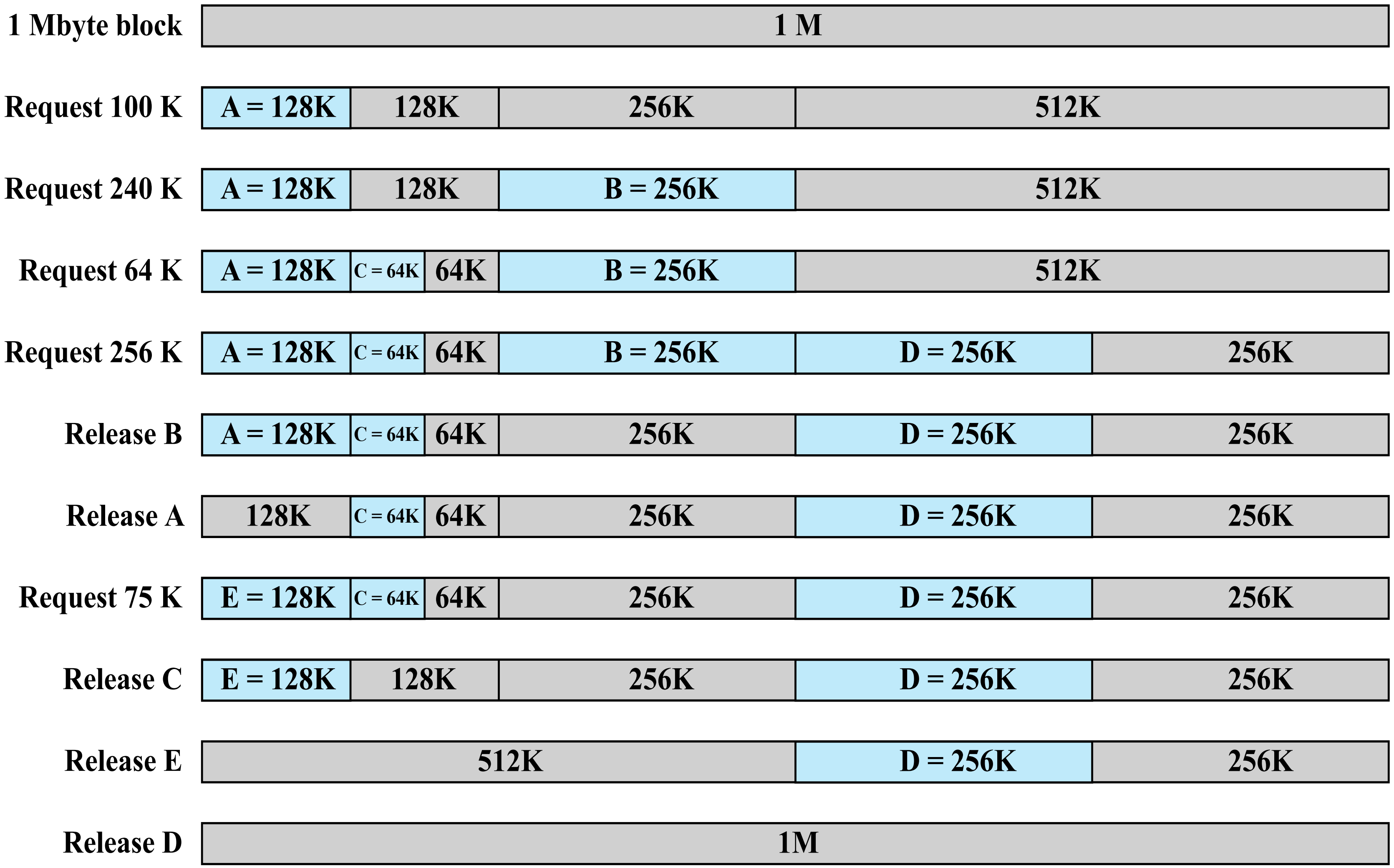
- *Best-firs*t will search entire list to make use of 18MB block
- *First-first* will use a 22 MB block resulting in a 6M fragment
- *Next-fit* will use a 36 MB block resulting in a 20 MB fragment



Buddy System

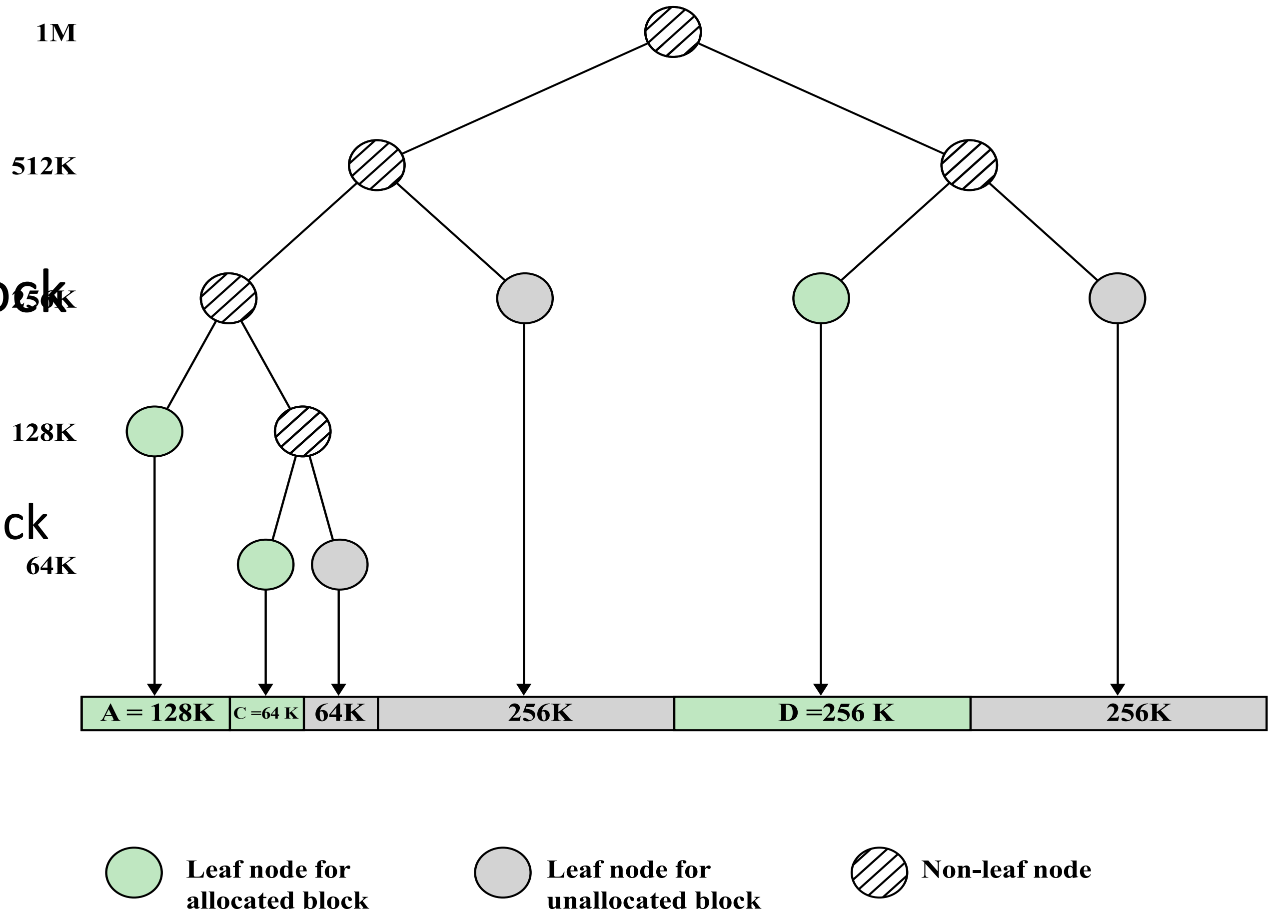
- Comprised of fixed and dynamic partitioning schemes
- Space available for allocation is treated as a single block
- Memory blocks are available of size 2^K *words,* $L \le K \le U$, where
  - 2^L = smallest size block that is allocated
  - 2^U = largest size block that is allocated; generally 2^U is the size of the entire memory available for allocation
- If a request of size *s* such that $2^{U-1} < s \le 2^U$ is made, 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*
  - *Otherwise, one* of the buddies is split in half again

Example of a Buddy System



Data Structures Used

- Linked Lists, List of lists
- Binary Trees
  - Leafs represent free block
  - Internal node represent a divided block
  - Can be represented using a bitmap
    - For n blocks use 2n bits
    - 256 KB bitmap can represent a 1 GB block



Tradeoffs

- All systems lead to fragmentation
- One can reduce external fragmentation for at the cost of internal fragmentation
- Can we do better?