

# **CS241 System Programming Memory Management (I)**

Klara Nahrstedt

Lecture 28

4/3/2006



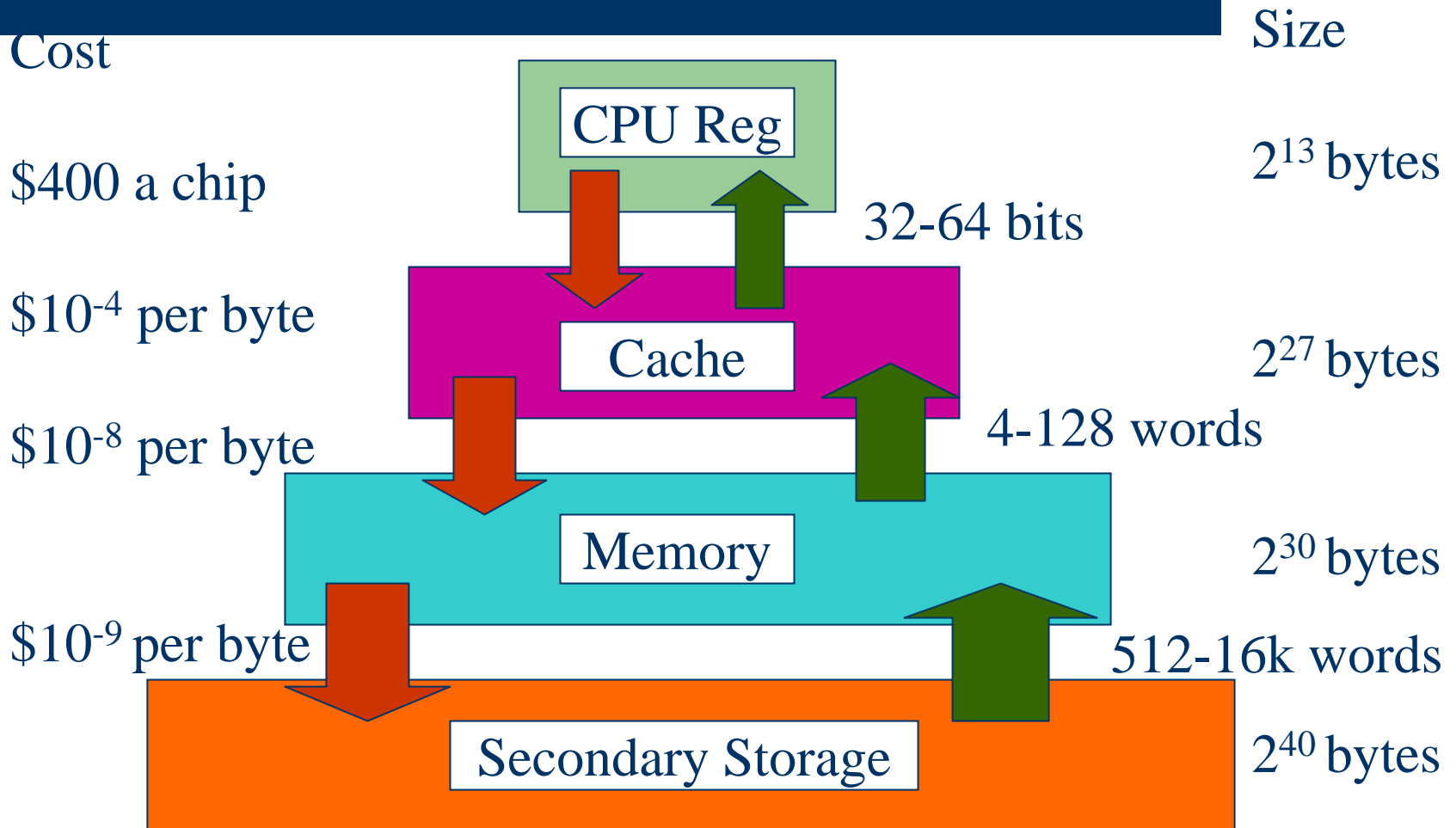
# Content

- Storage Hierarchy
- Resident Monitor
- Fixed Partitions for Spooling and Multiprogramming
- Variable Sized Partitions
- Storage Placement Strategies
- Compaction

# Administrative

- MP4 is posted, due April 17, 2006
- Quiz 8 is April 7, 2006
- Material covered in Quiz 8
  - Tanenbaum Chapter 5.3, 5.4 and 5.5

# Storage Hierarchy



# General Memory Problem

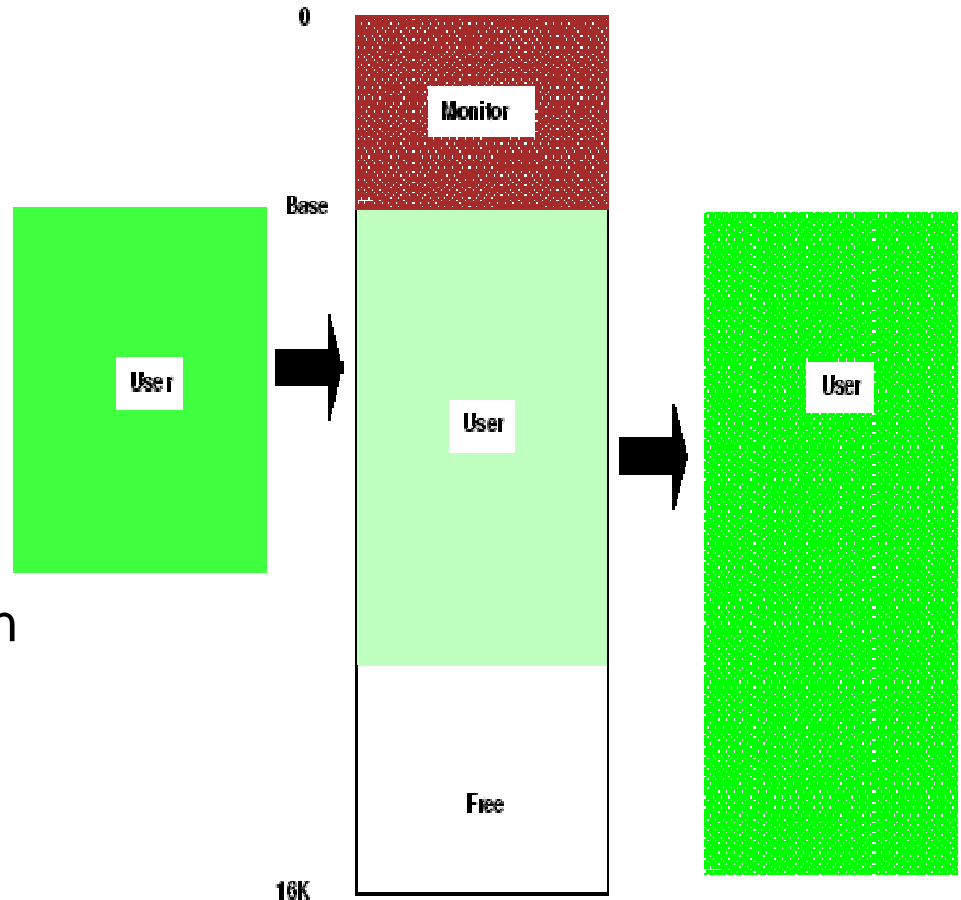
- We have a limited (expensive) physical resource: main memory
- We want to use it as efficiently as possible
- We have an abundant, slower resource: disk

# Lots of Variants

- Many programs, total size less than memory
  - Technically possible to pack them together
  - Will programs know about each other's existence?
- One program, using lots of memory
  - Can you only keep part of the program in memory?
- Lots of programs, total size exceeds memory
  - What programs are in memory, and how to decide?

# Memory Manager

- Manage memory hierarchy
  - Monitor used and free memory
  - Allocate memory to processes
  - Reclaim (De-allocate) memory
  - Swapping between main memory and disk

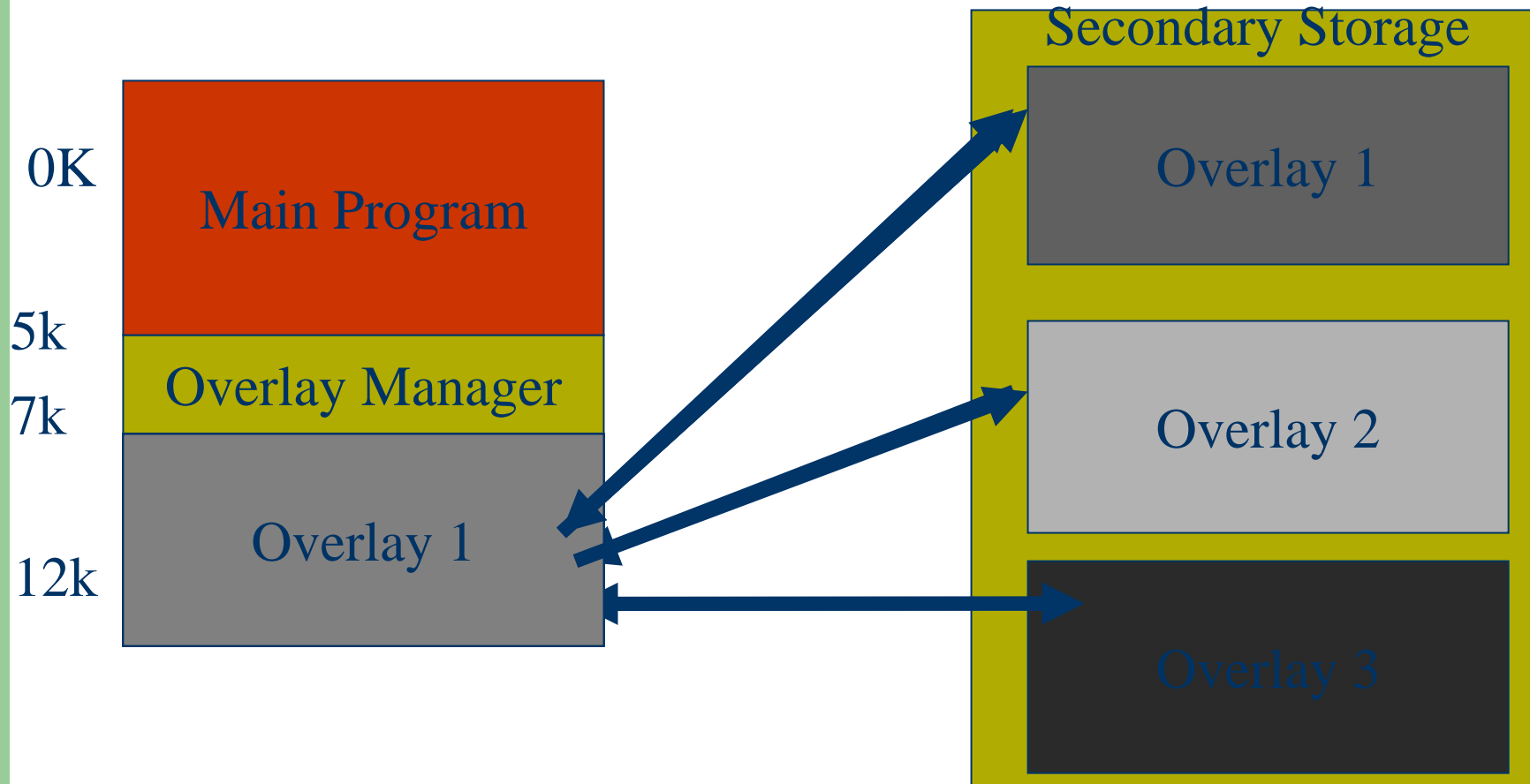


# Mono-programming without Swapping

- Run one process at a time
  - simplest possible memory management scheme
- Memory is shared only between OS and the process.
- Three different ways to organize memory:
  - OS at the bottom of memory in RAM (Random Access Memory); was used in mainframes and minicomputers; not used anymore
  - OS is in ROM (Read-Only Memory) at the top of memory; is used in some palm-tops and embedded systems
  - Device drivers are in ROM at the top of memory and the rest is in RAM; was used by early PCs (e.g., running MS-DOS), where the portion in the ROM is called BIOS (Basic Input Output System).

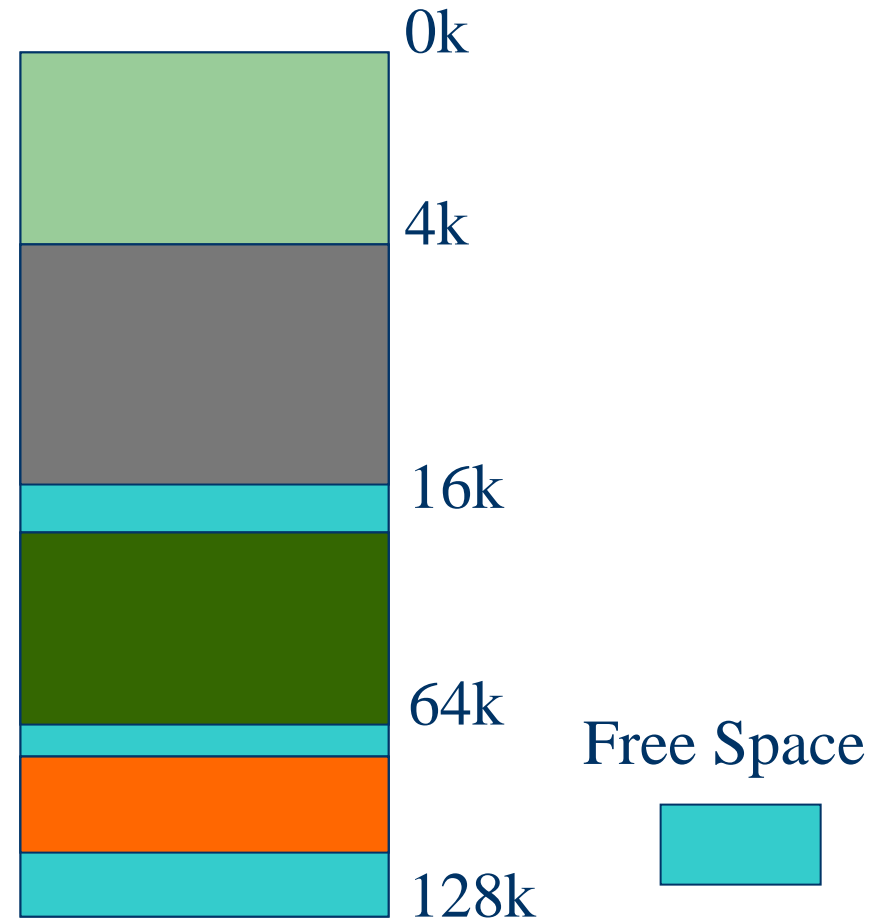
# Overlaying

Used when process memory requirement exceeds the physical memory space

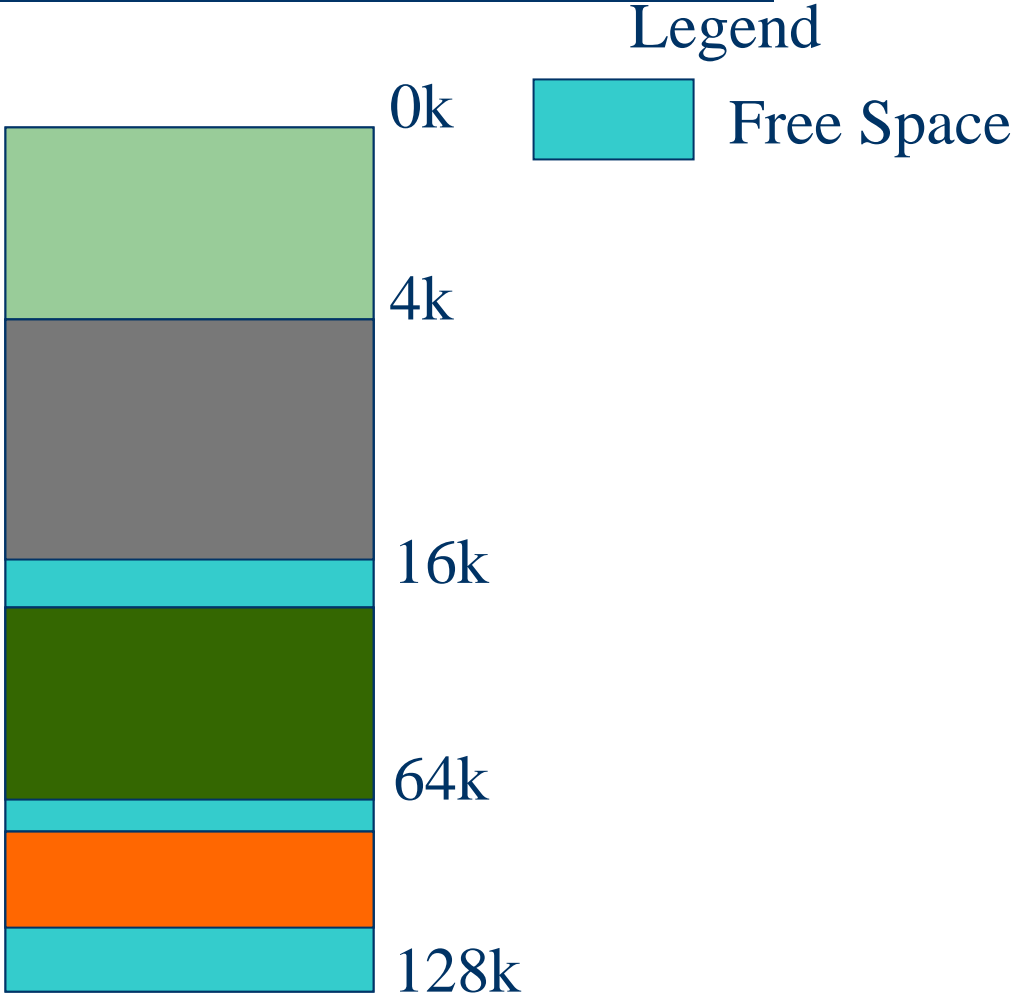


# Multiprogramming with Fixed Partitions

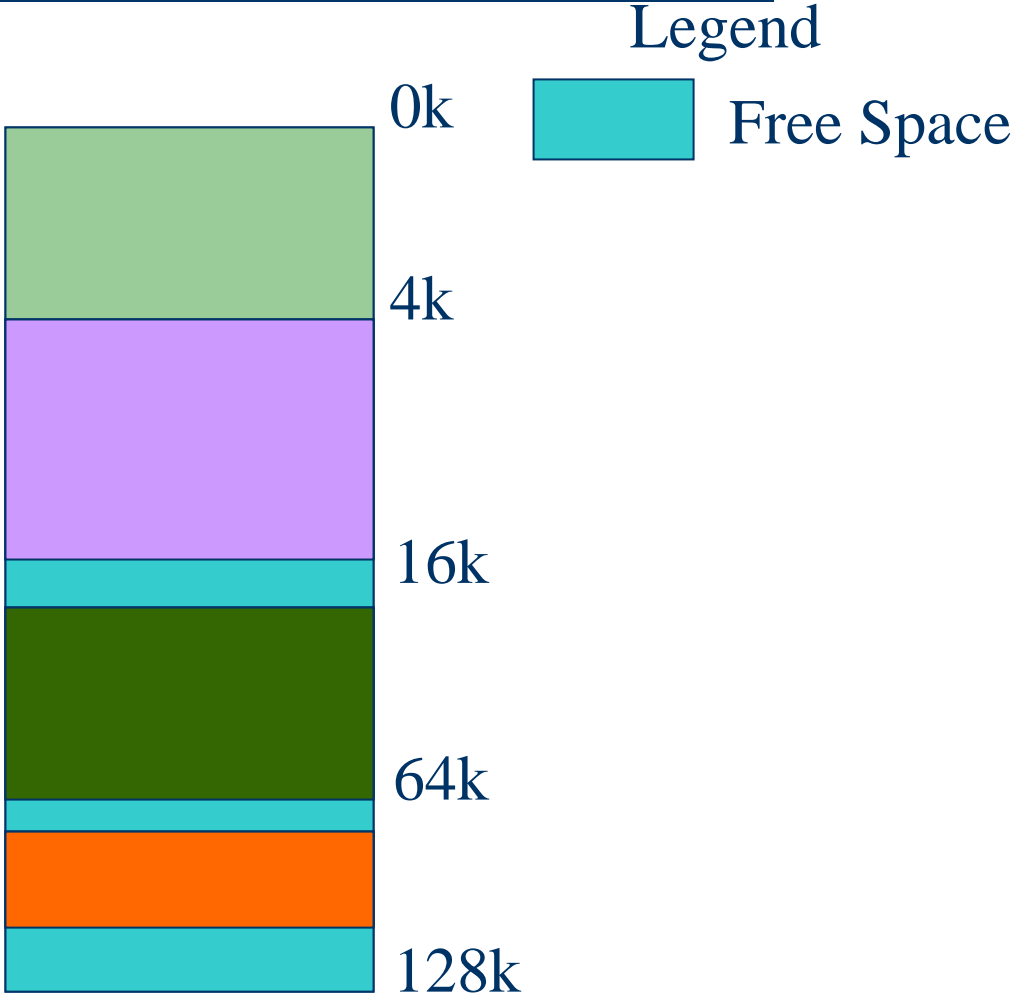
- Divide memory into  $n$  (possible unequal) partitions.
- Problem:
  - Fragmentation



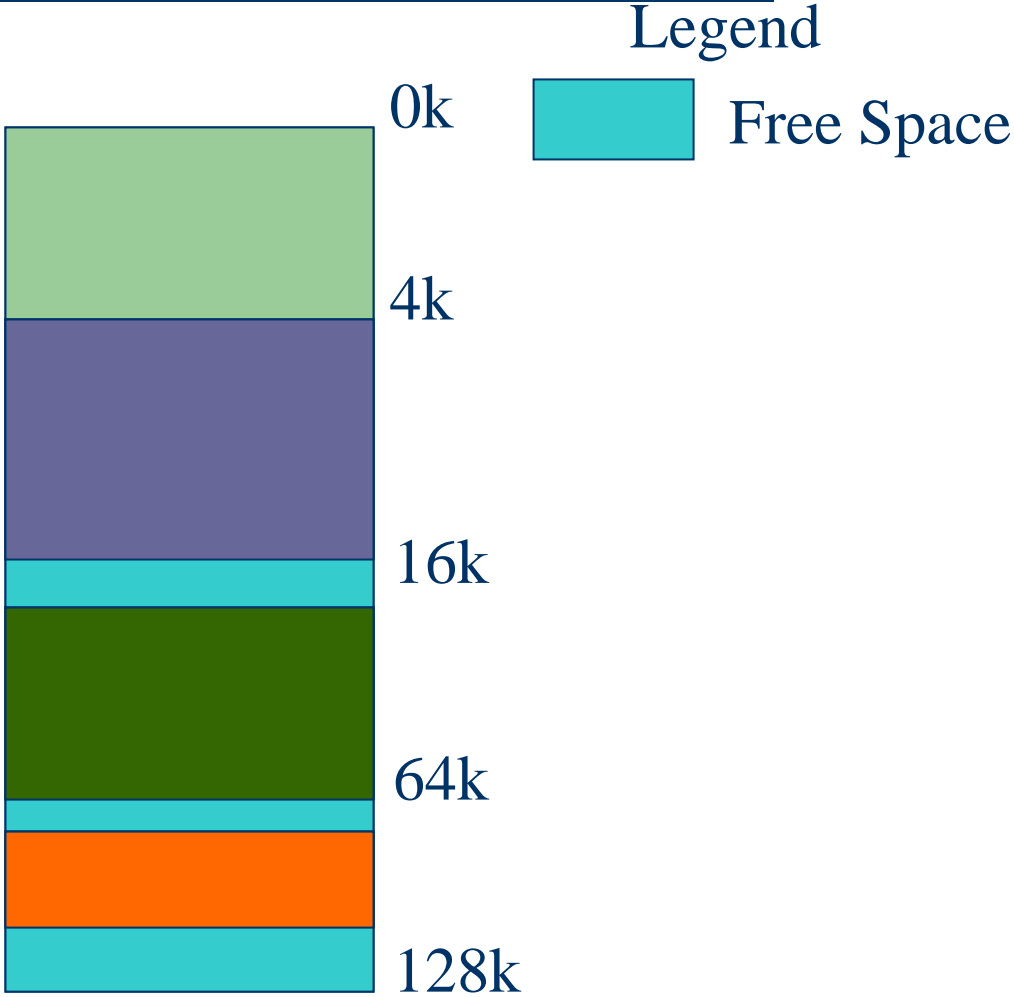
# Fixed Partitions



# Fixed Partitions



# Fixed Partitions



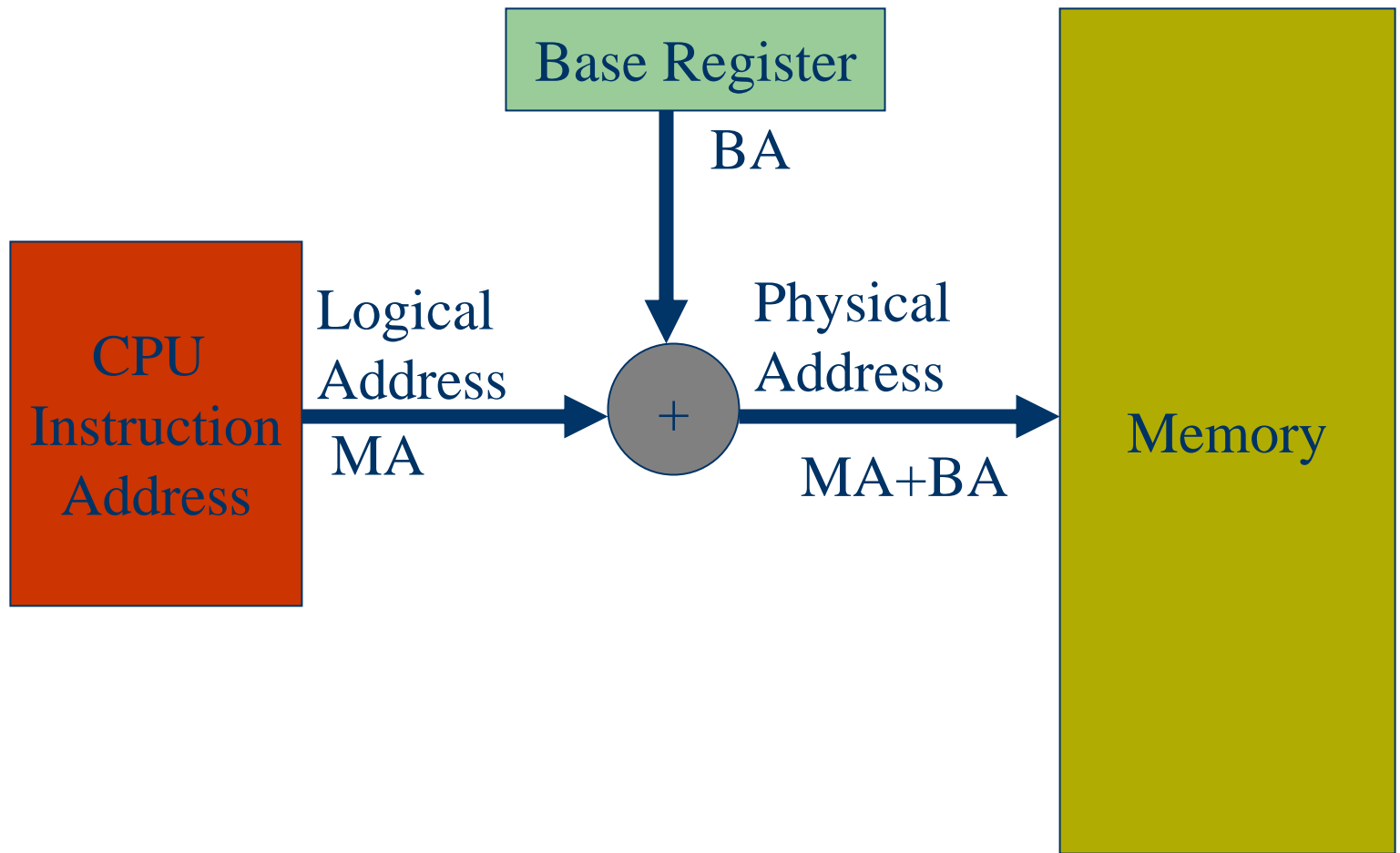
# Fixed Partition Allocation

- Separate input queue for each partition
  - Requires sorting the incoming jobs and putting them into separate queues
  - Inefficient utilization of memory
    - when the queue for a large partition is empty but the queue for a small partition is full. Small jobs have to wait to get into memory even though plenty of memory is free.
- One single input queue for all partitions.
  - Allocate a partition where the job fits in.
    - Best Fit
    - Available Fit

# Relocation

- Correct starting address when a program should start in the memory
- Different jobs will run at different addresses
  - When a program is linked, the linker must know at what address the program will begin in memory.
- Logical addresses, Virtual addresses
  - Logical address space , range (0 to max)
- Physical addresses, Physical address space
  - range (R+0 to R+max) for base value R.
- User program **never sees** the real physical addresses
- Memory-management unit (MMU)
  - map virtual to physical addresses.
- relocation register
  - Mapping requires hardware (MMU) with the base register

# Relocation Register

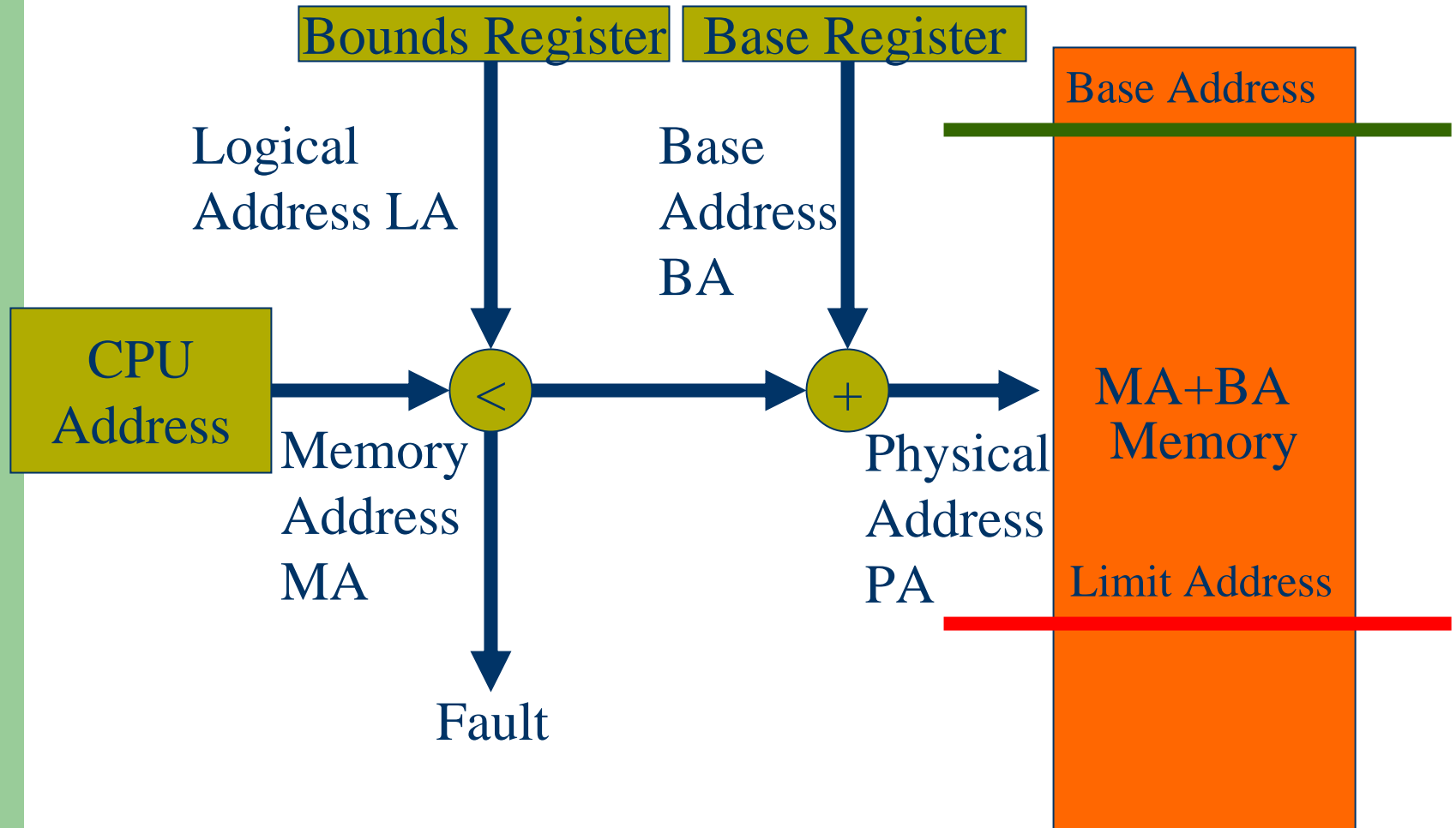


# Protection

- Problem:
  - How to prevent a malicious process to write or jump into other user's or OS partitions
- Solution:
  - Base bounds registers



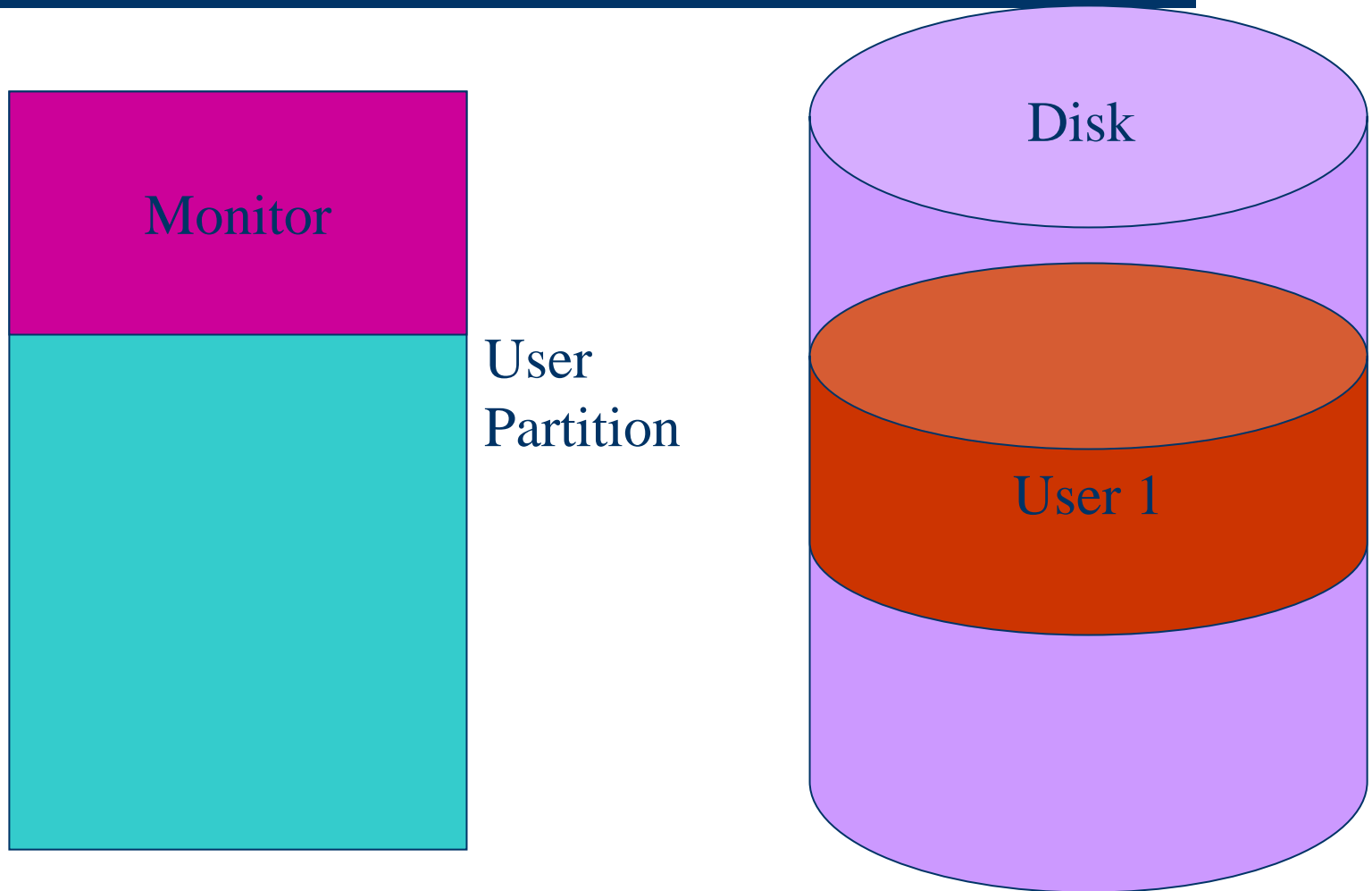
# Base Bounds Registers



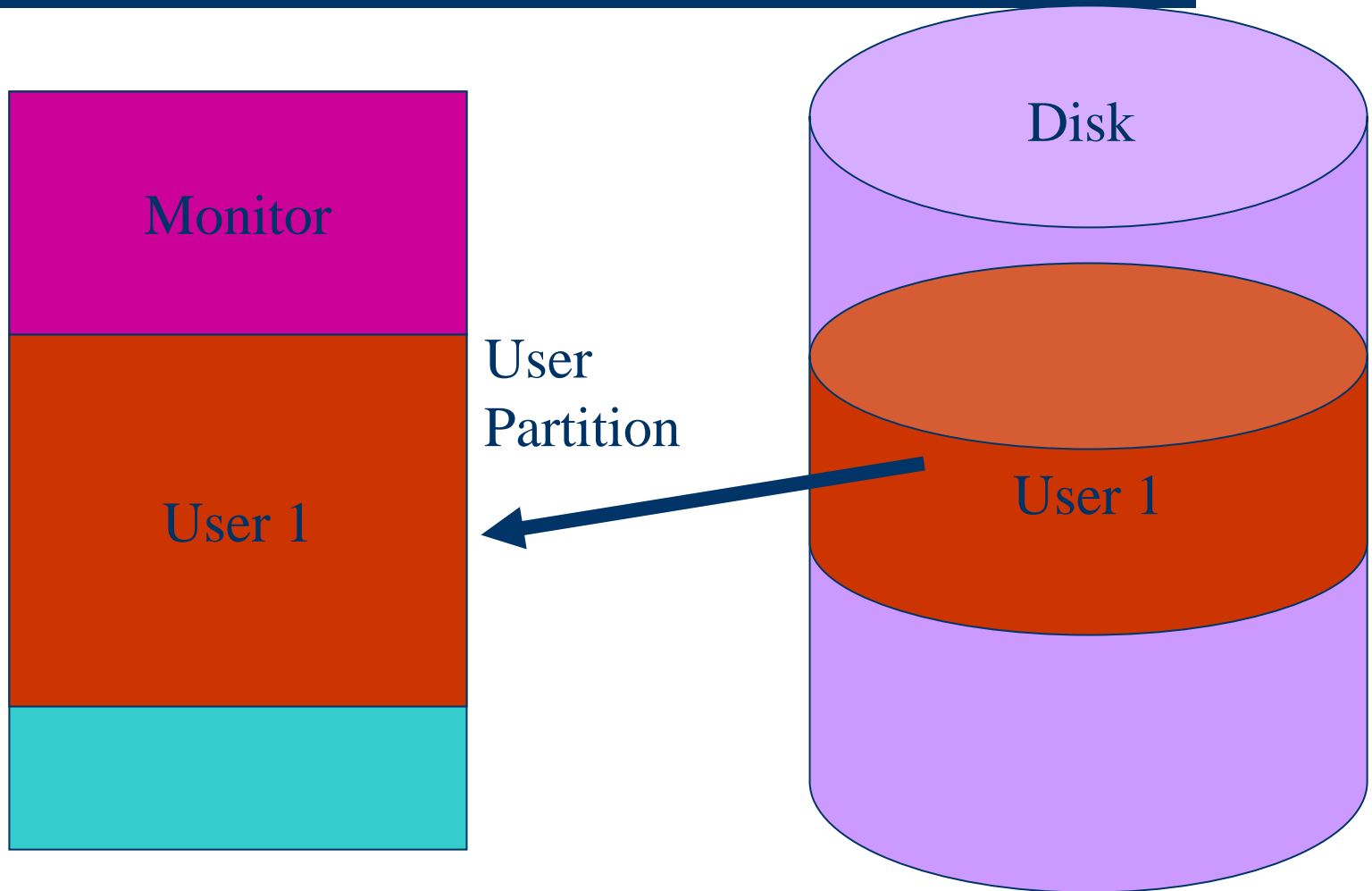
# Swapping

- Move a part of the whole process to disk
- Allow several processes to share a fixed partition
- Processes that grow can be swapped out and swapped back in a bigger partition

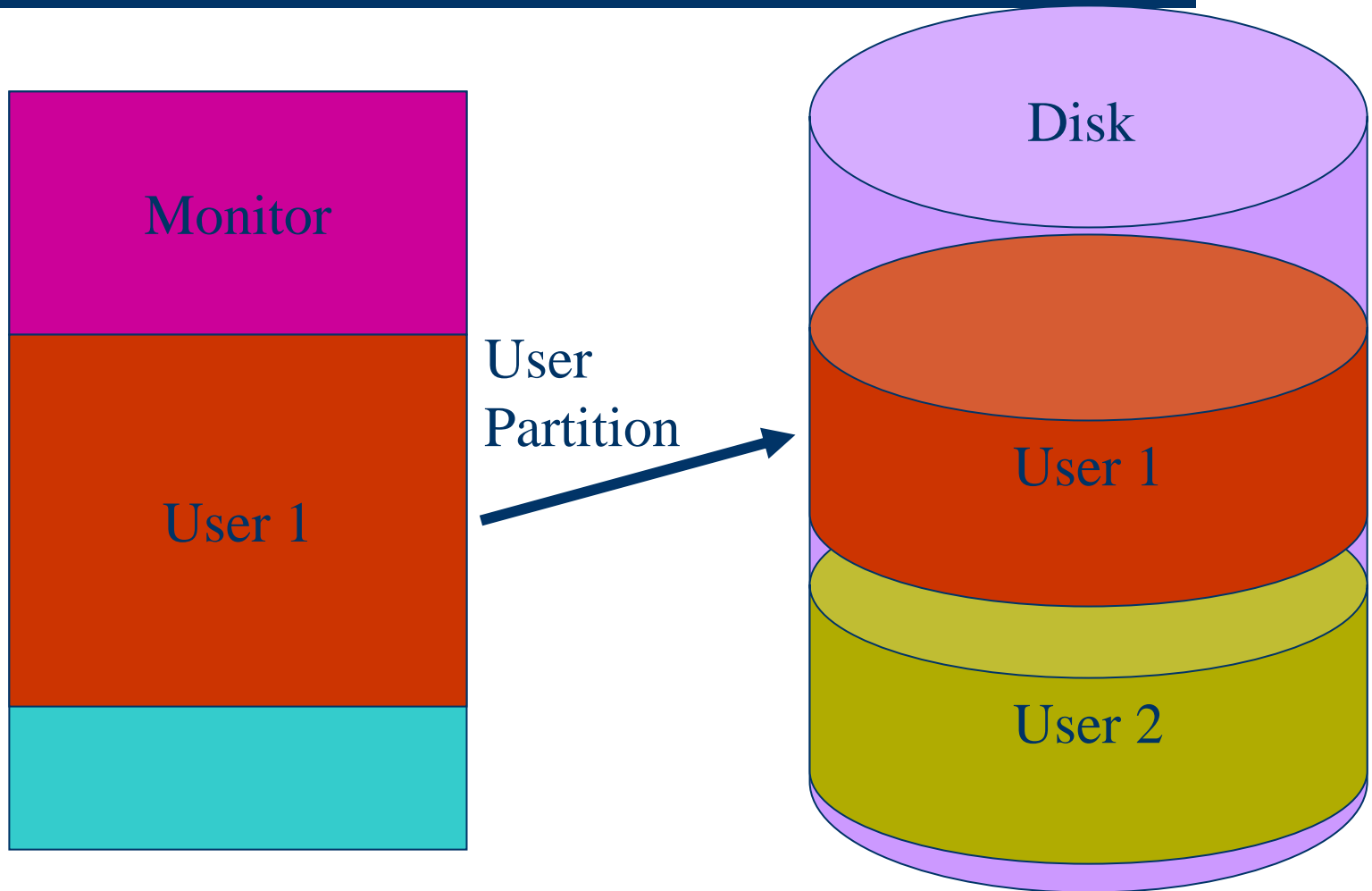
# Swapping



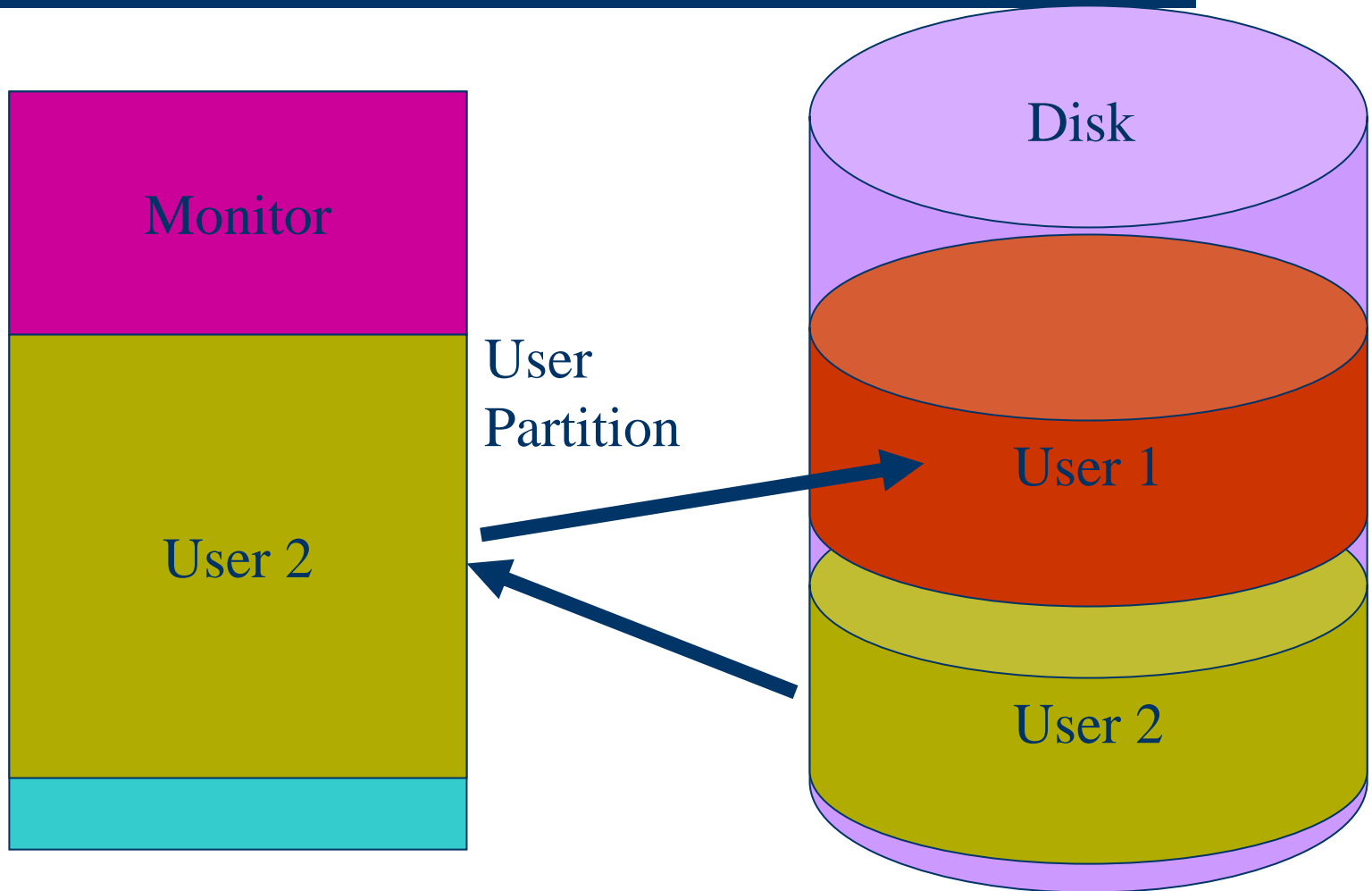
# Swapping



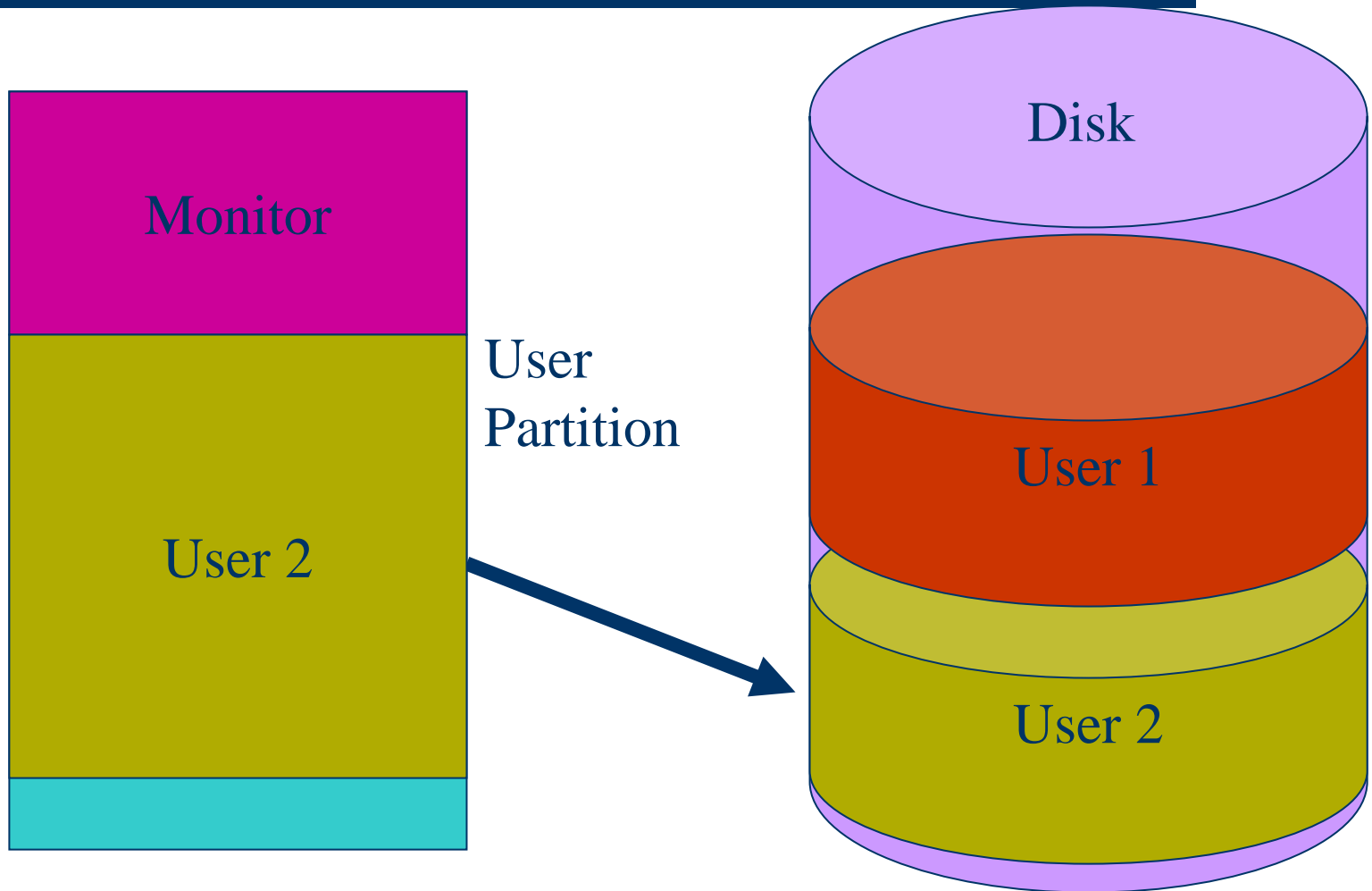
# Swapping



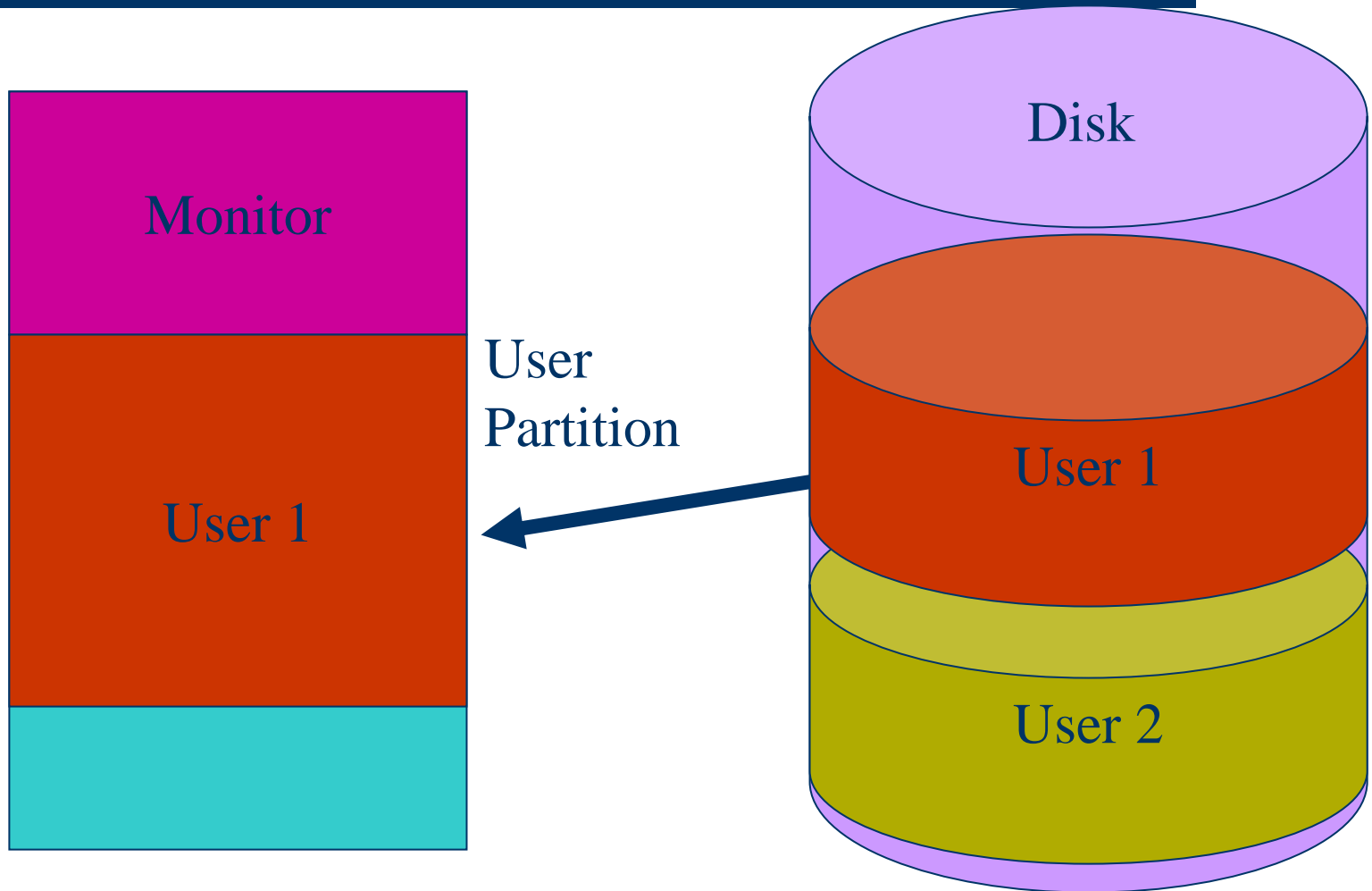
# Swapping



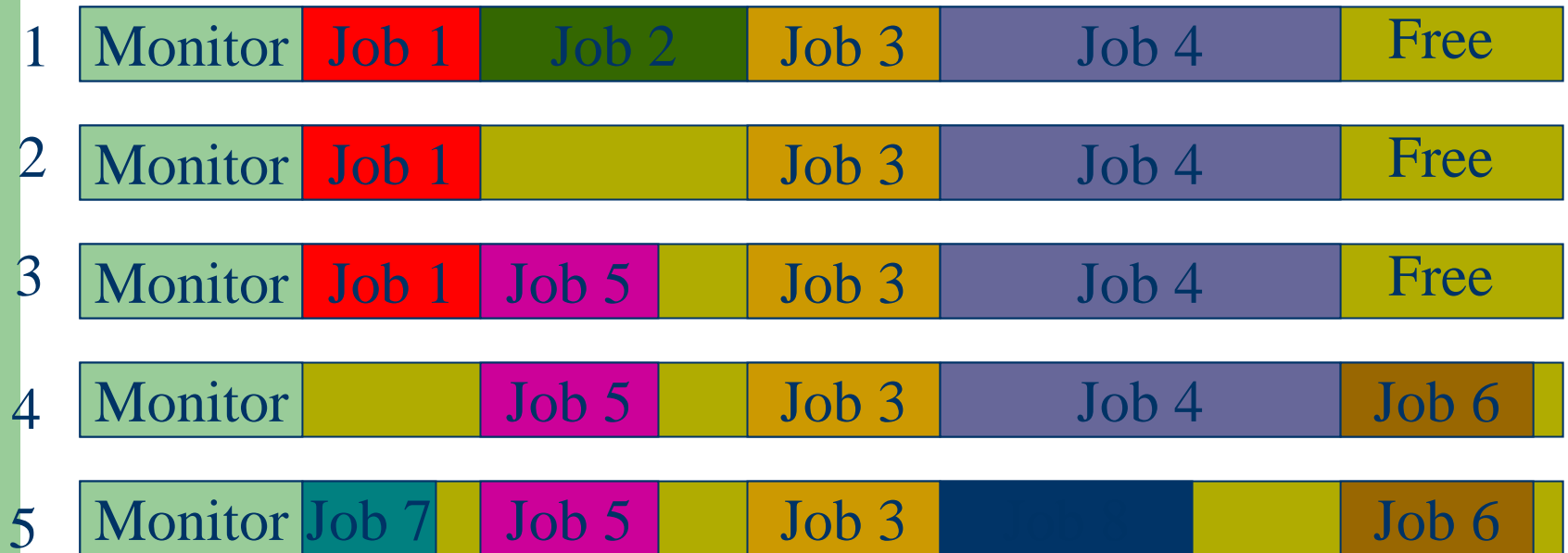
# Swapping



# Swapping



# Variable Partitions and Fragmentation



## Memory wasted by External Fragmentation

# Memory Management with Bitmaps

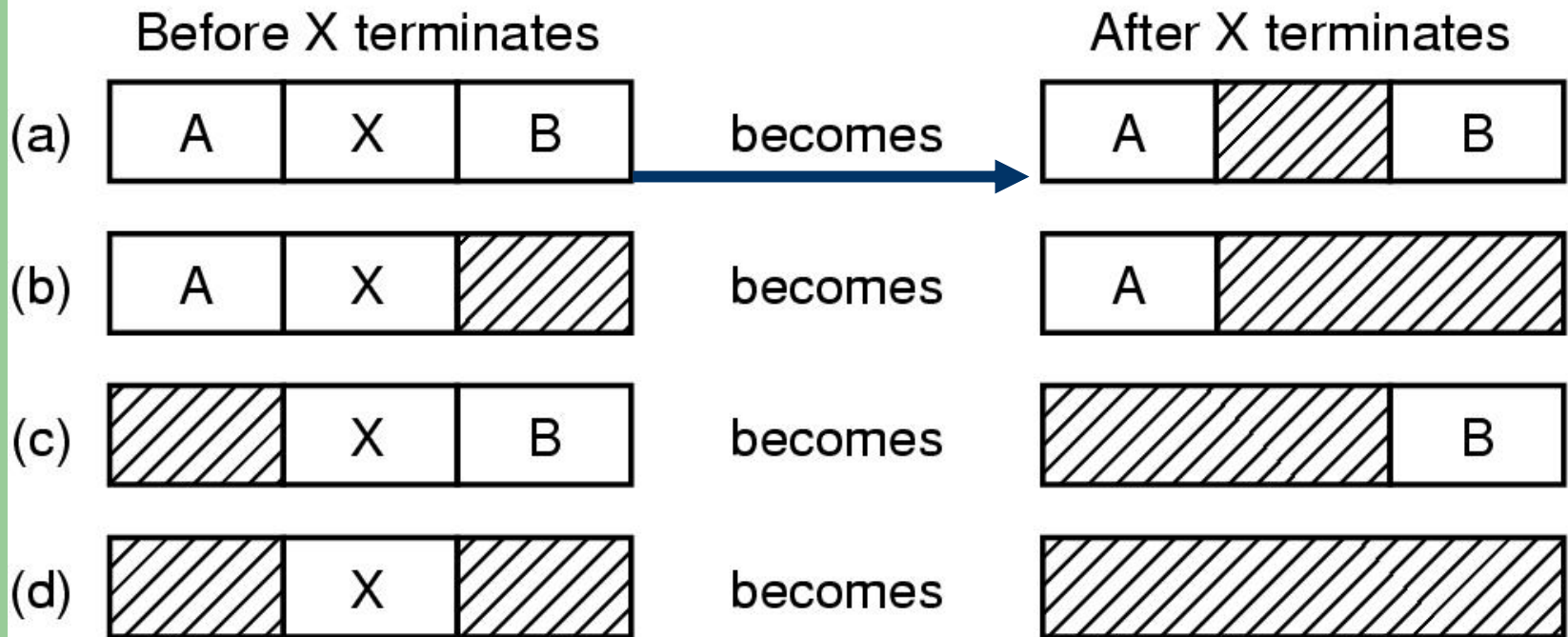
- Use bitmaps for free lists.
- Memory is divided into allocation units.
  - One allocation unit corresponds to 1bit in the bitmap
  - 0: free, 1: allocated
- Size of allocation unit
  - The smaller the allocation unit, the larger the bitmap.
- Problem: allocation
  - When a new process arrives, the manager must find consecutive 0 bits in the map.
  - Searching a bitmap for a run of a given length is a slow operation.



# Memory Management with Linked Lists

- Use a linked list of allocated and free memory segments (called hole)
  - sorted by the address or by the size

Four neighbor combinations for the terminating process X

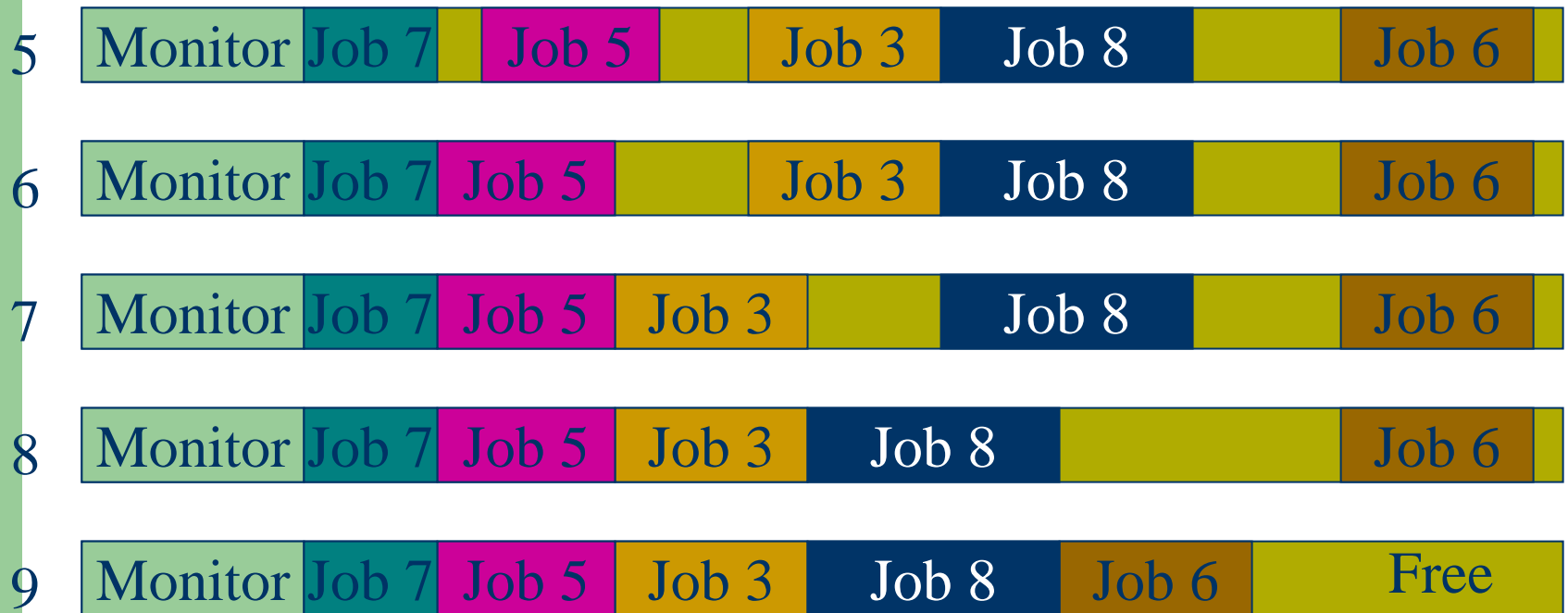


# Storage Placement Strategies

- **Analogy:**
  - Shoe Fitting
  - Valet parking
- **Best Fit**
  - Use the hole whose size is equal to the need, or if none is equal, the hole that is larger but closest in size.
  - Problem: Creates small holes that can't be used.
- **First Fit**
  - Use the first available hole whose size is sufficient to meet the need.
  - Problem: Creates average size holes.
- **Next Fit.**
  - Minor variation of first fit: search for the last hole stopped.
  - Problem: slightly worse performance than first fit.
- **Worst Fit.**
  - Use the largest available hole.
  - Problem: Gets rid of large holes making it difficult to run large programs.
- **Quick Fit.**
  - maintains separate lists for some of the more common sizes requested.
  - When a request comes for placement it finds the closest fit.
  - This is a very fast scheme, but a merge is expensive. If merge is not done, memory will quickly fragment in a large number of holes into which no processes fit.

# Compaction (Similar to Garbage Collection)

- Assumes programs are all relocatable
- Processes must be suspended during compaction
- Need be done only when fragmentation gets very bad



# Storage Management Problems

- Fixed partitions suffer from internal fragmentation
- Variable partitions suffer from external fragmentation
- Compaction suffers from overhead
- Partitions must be less in size than real memory
- Overlays are painful to program efficiently
- Swapping requires writing to disk sectors

# Summary

- Mono-programming without Swapping
- Mono-programming with Fixed Partitions
- Relocation
- Base and Limit Registers
- Swapping
- Variable Partitions and fragmentation
- Compaction