Chapter 3

Memory Management

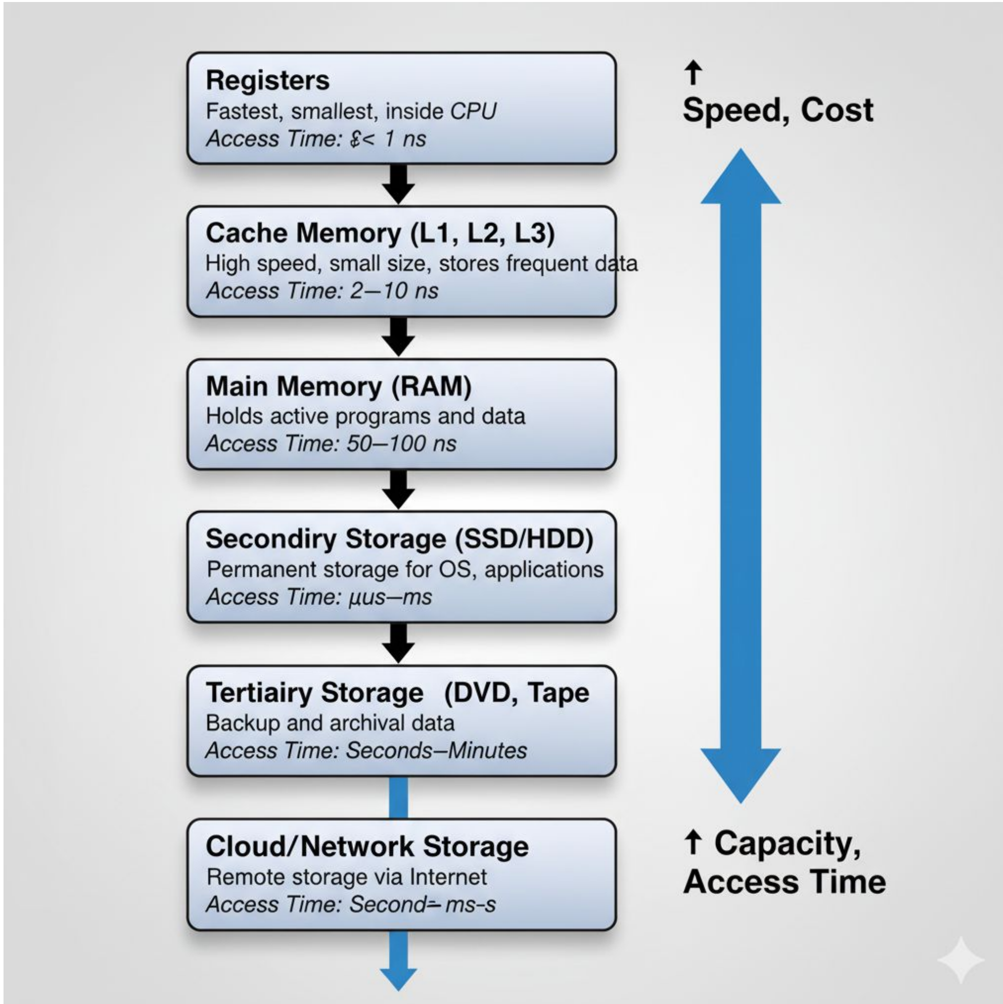
Main Memory (RAM)

- Main memory (RAM) is an important resource that must be carefully managed.
- A program resides on a disk as binary executable file
- To be executed the program must be brought into RAM
- The CPU fetches instructions from RAM according to the value of the PC
- The instruction operands may be needed to be fetched from RAM
- After execution the results may be stored back to RAM

Memory Management

- Ideally programmers want memory that is
  - private
  - large
  - fast
  - non volatile
  - Cheap

Computer Hardware Review



- It is the job of the operating system to
  - **abstract** this hierarchy into a useful model
  - and then **manage** the abstraction.

Memory Management

- The part of the operating system that manages the memory hierarchy is called the *memory manager*.
  - Keep track of used memory
  - Allocate memory to processes
  - Deallocate it when they are done

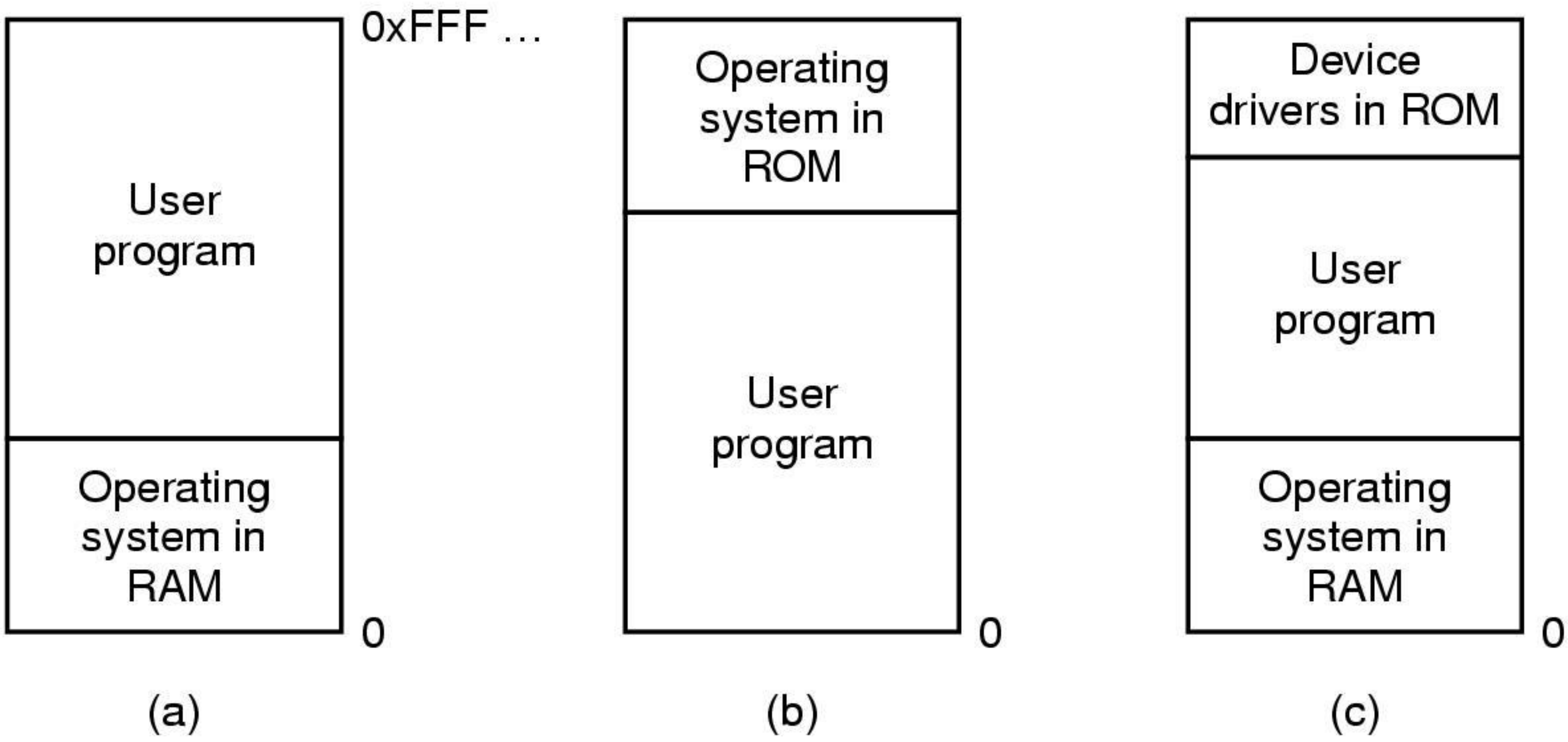
Objective

- In this chapter we will study
  - how operating systems create abstractions from memory
  - and how they manage them.
- The focus will be on the **programmer's model** of main memory
- Memory Abstraction: the view/illusion of the memory presented to the programmer by the OS

No Memory Abstraction

- The simplest memory abstraction
- Every program simply saw the physical memory
  - MOV REG1, 1000
  - move the contents of physical memory location 1000 to REGISTER1
- Model of memory presented to the programmer is simply physical memory
- Not possible to have 2 running programs in memory at the same time

No Memory Abstraction



Three simple ways of organizing memory with an operating system and one user process

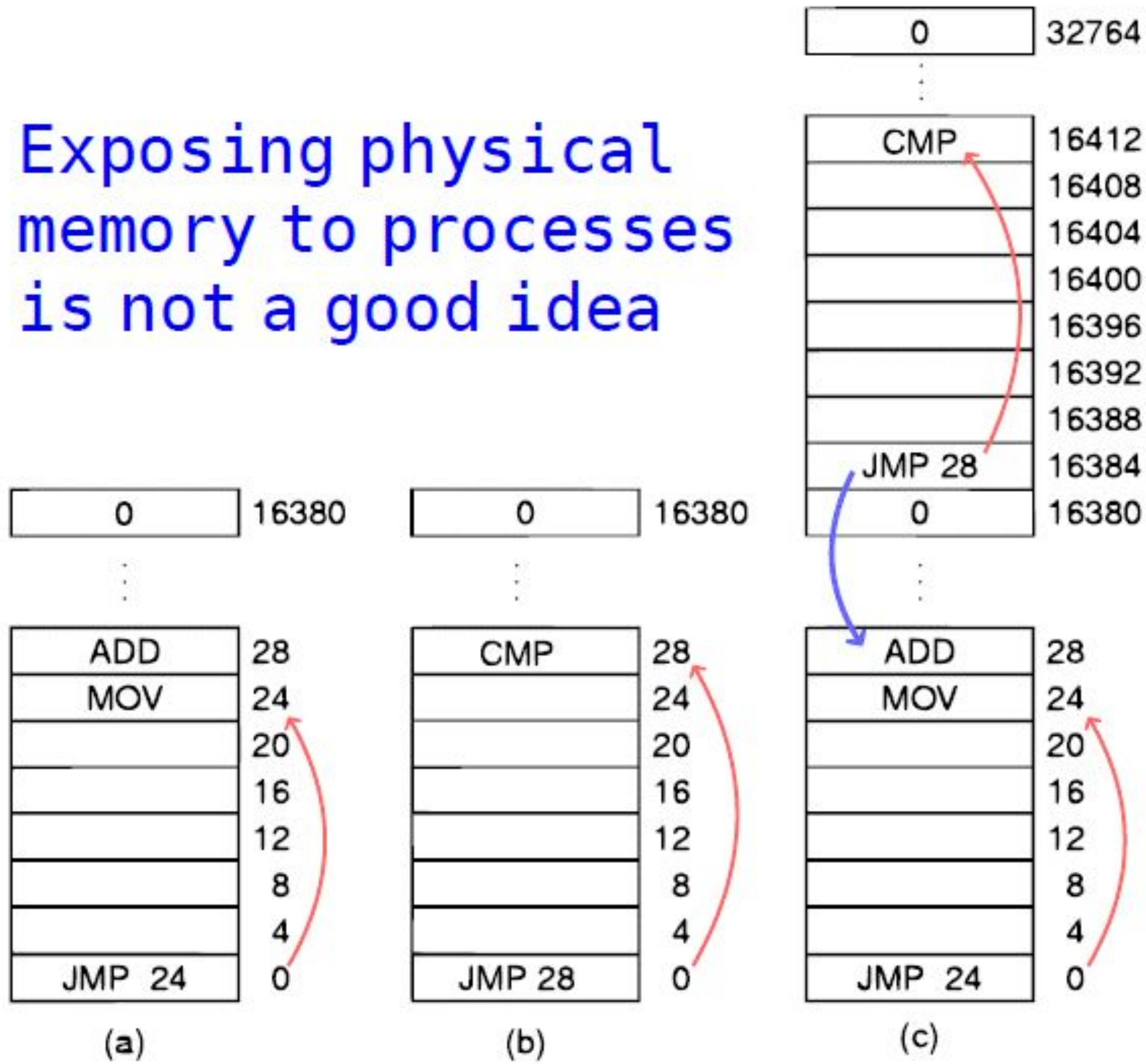
Running Multiple Programs Without a Memory Abstraction

- Used in the early models of the IBM 360.
- Memory can be divided into 2-KB blocks and each one was assigned a 4-bit protection key held in special registers inside the CPU.
- The PSW (Program Status Word) also contained a 4-bit key.
- The hardware trapped any attempt by a running process to access memory with a protection code different from the PSW key.

Running Multiple Programs Without a Memory Abstraction

- Relocation problem
- Use of static relocation

Exposing physical memory to processes is not a good idea



Conclusion

- exposing physical memory to processes has several major drawbacks.
  - if user programs can address every byte of memory, they can easily trash the operating system intentionally or by accident
  - it is difficult to have multiple programs running at once

Conclusion

- Two problems have to be solved to allow multiple applications to be in memory at the same time without interfering with each other
  - protection
  - Relocation
- SOLUTION
  - invent a new abstraction for memory: the address space

A Memory Abstraction: ADDRESS SPACES

- An address space is the range of addresses that a process can use to address memory.
  - Just as the process concept creates a kind of *abstract CPU* to run programs,
  - the address space creates a kind of *abstract memory* for programs to live in.
- Each process has its own independent address space

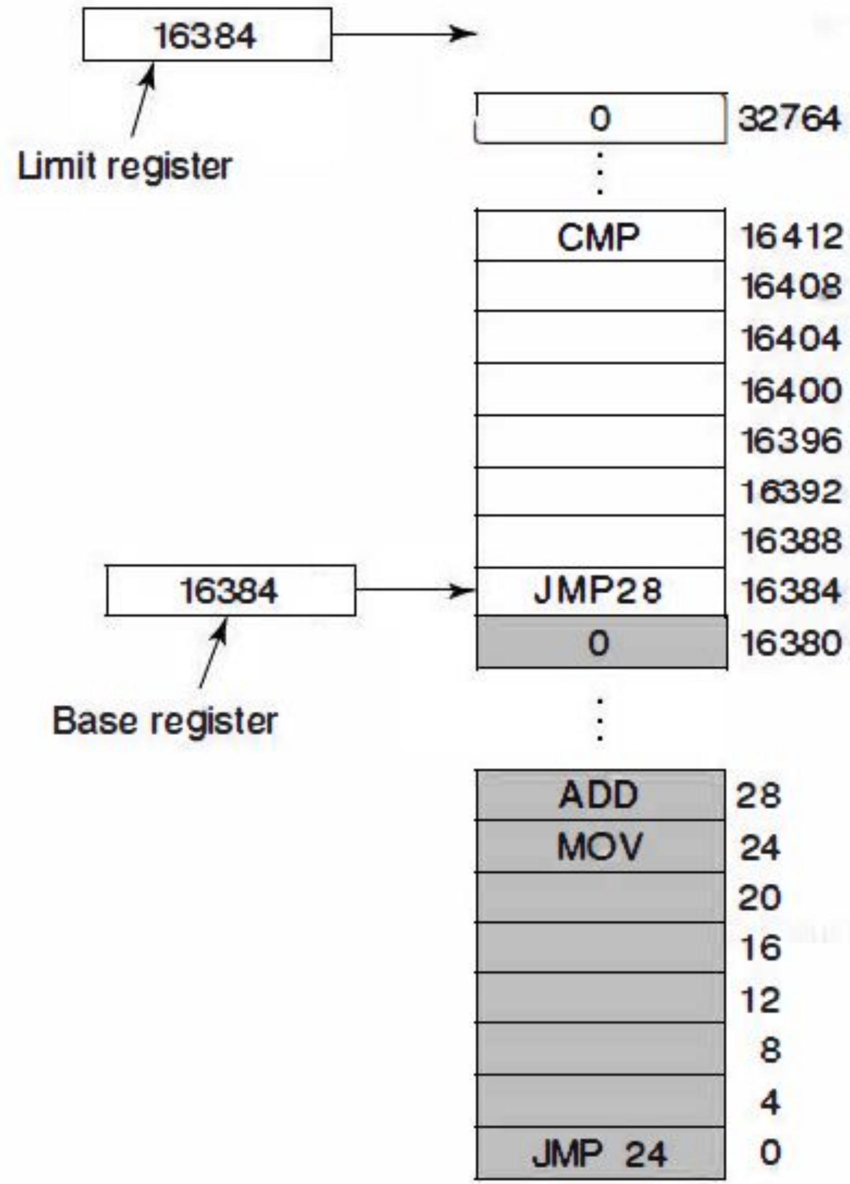
A Memory Abstraction: ADDRESS SPACES

- how to give each program its own address space??
  - address 28 in one program means a different physical location than address 28 in another program

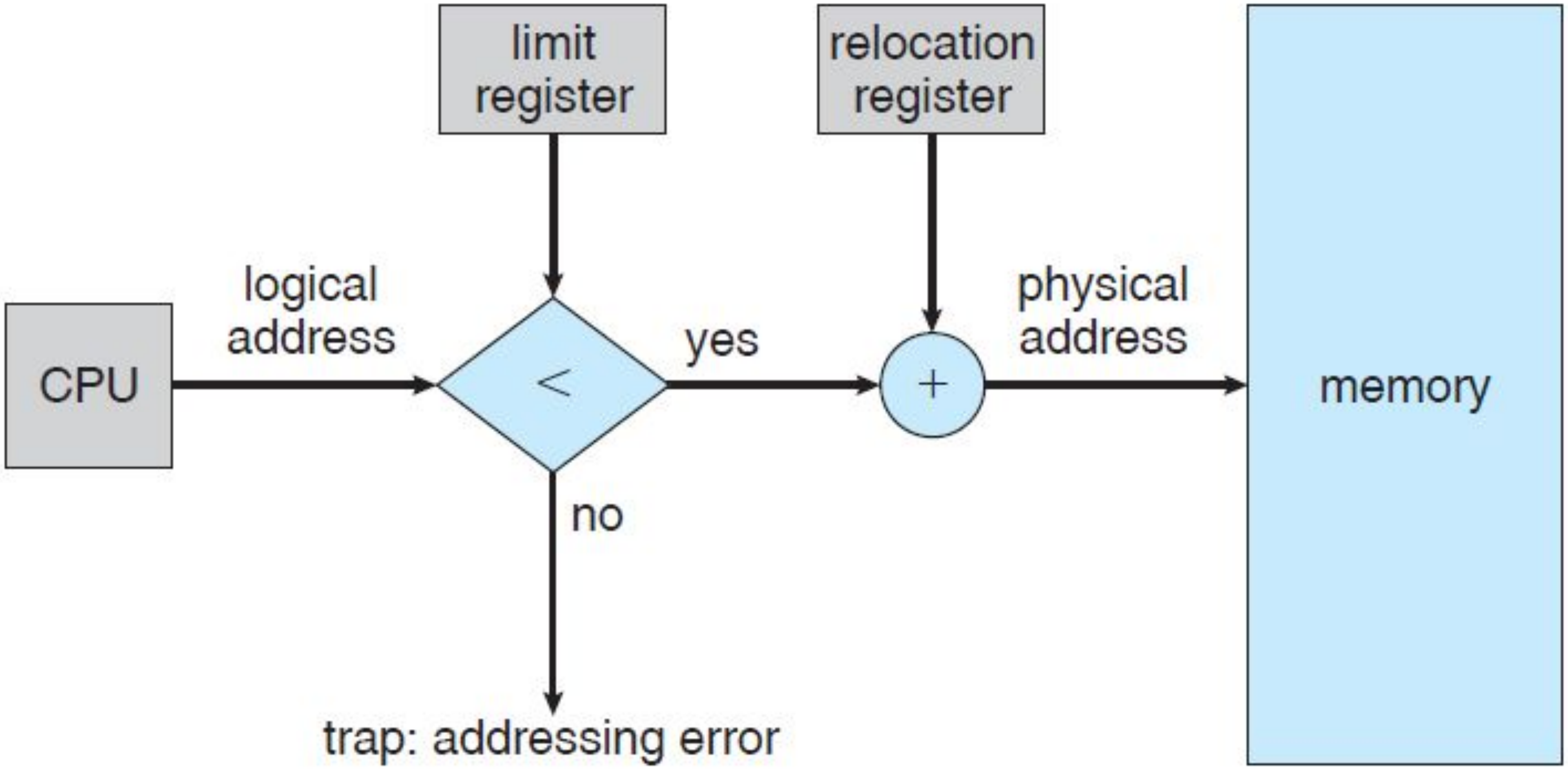
Base and Limit Registers

- Base and limit registers can be used to give each process a separate address space
- CPU hardware automatically adds the base value to the address generated by the process

  JMP 28 => JMP 16412
- Dynamic relocation.



Base and Limit Registers



Disadvantage

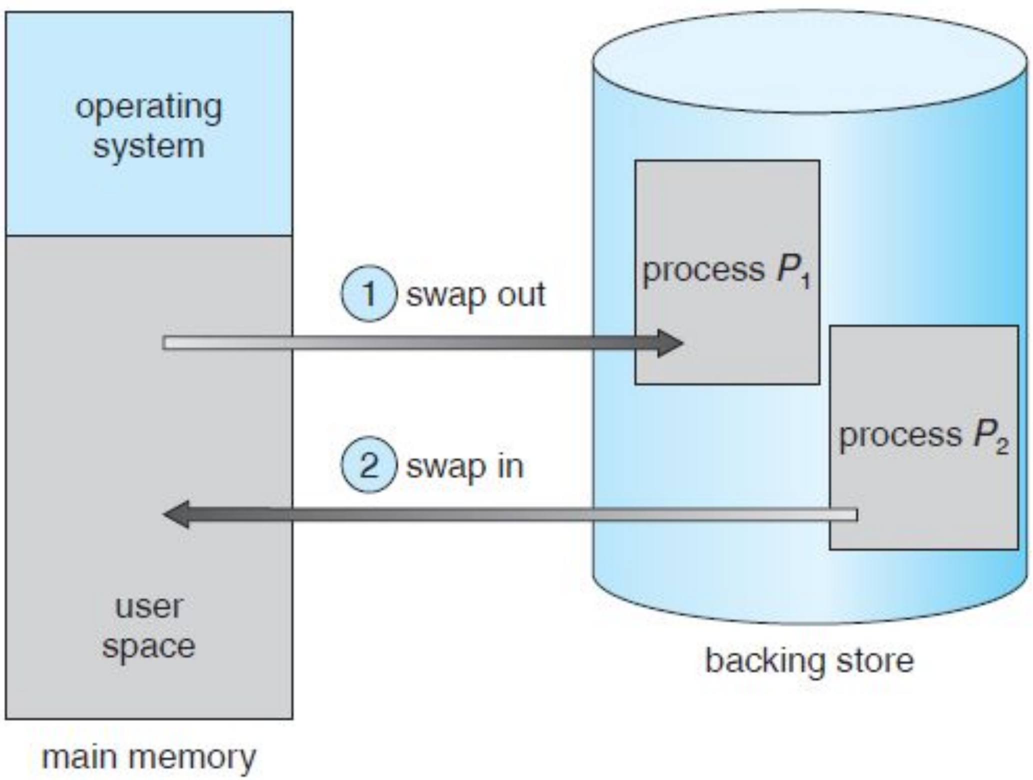
- an addition and a comparison on every memory reference.

Memory Overload

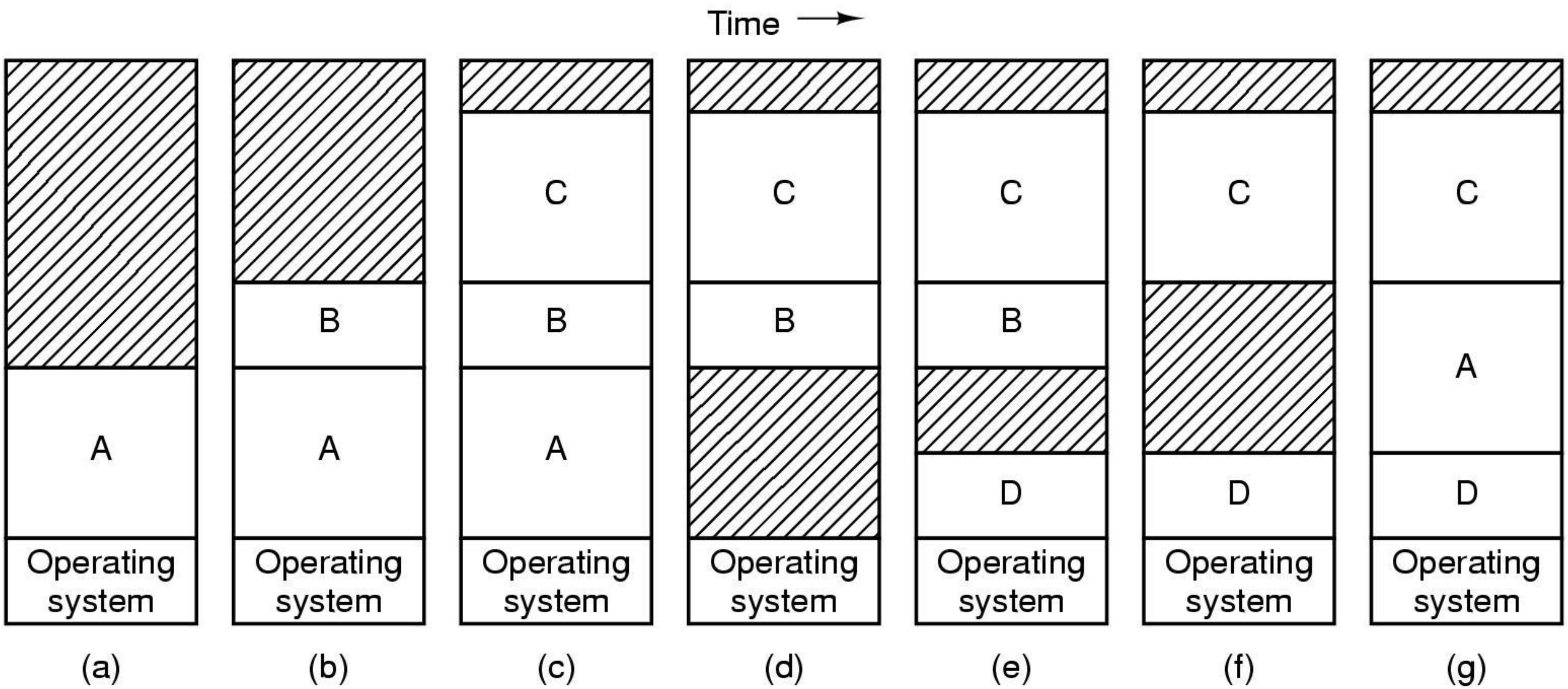
- In practice, the total amount of RAM needed by all the processes is often much more than can fit in memory.
- keeping all processes in memory all the time requires a huge amount of memory and cannot be done if there is insufficient memory.
- Two approaches to solve
  - Swapping
  - Virtual Memory

Swapping (1)

- Bringing in each process in its entirety,
- running it for a while
- then putting it back on the disk



Swapping



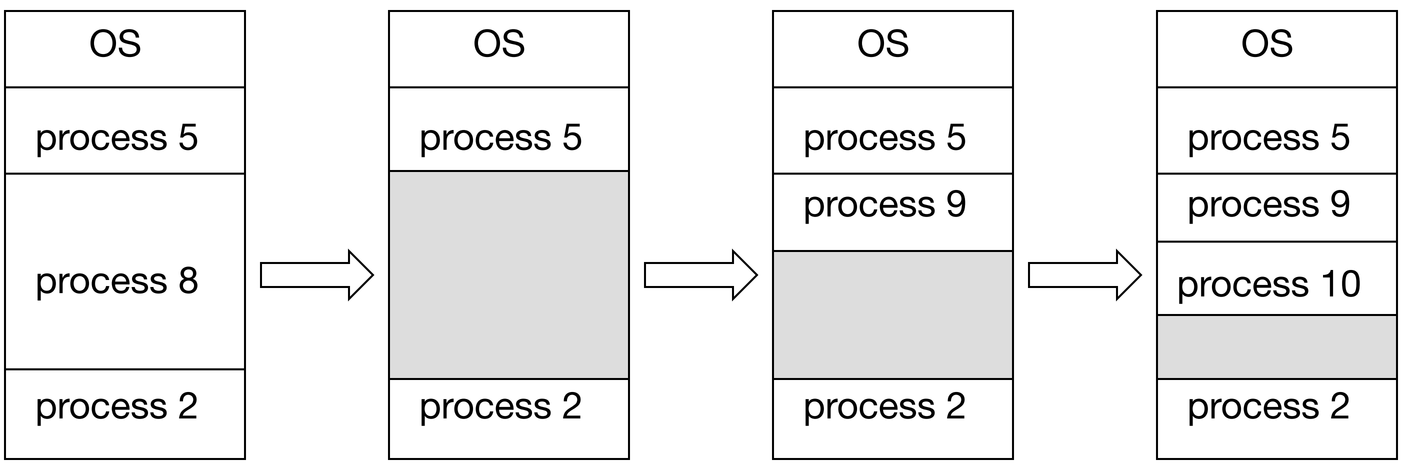
Memory allocation changes as

- processes come into memory
- leave memory

Shaded regions are unused memory

Contiguous Allocation in Swapping

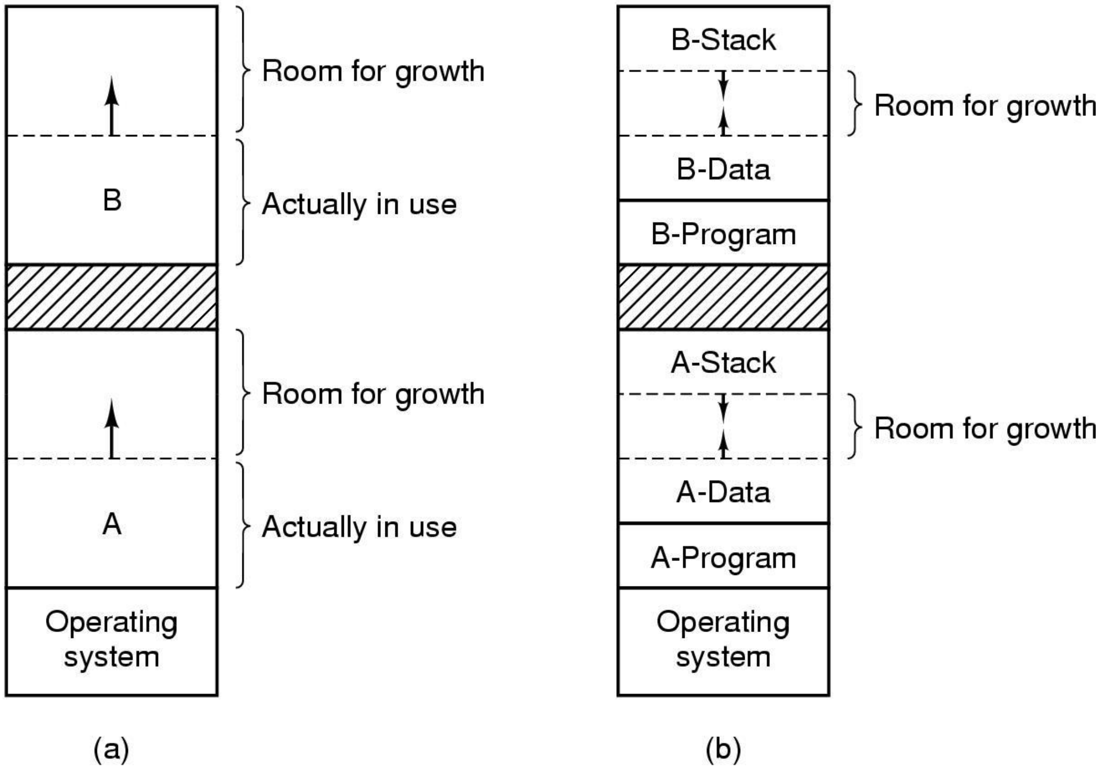
- Hole – block of available memory; holes of various size are scattered throughout memory
- When a process arrives, it is allocated memory from a hole large enough to accommodate it
- Operating system maintains information about:
  a) allocated partitions b) free partitions (hole)



Swapping

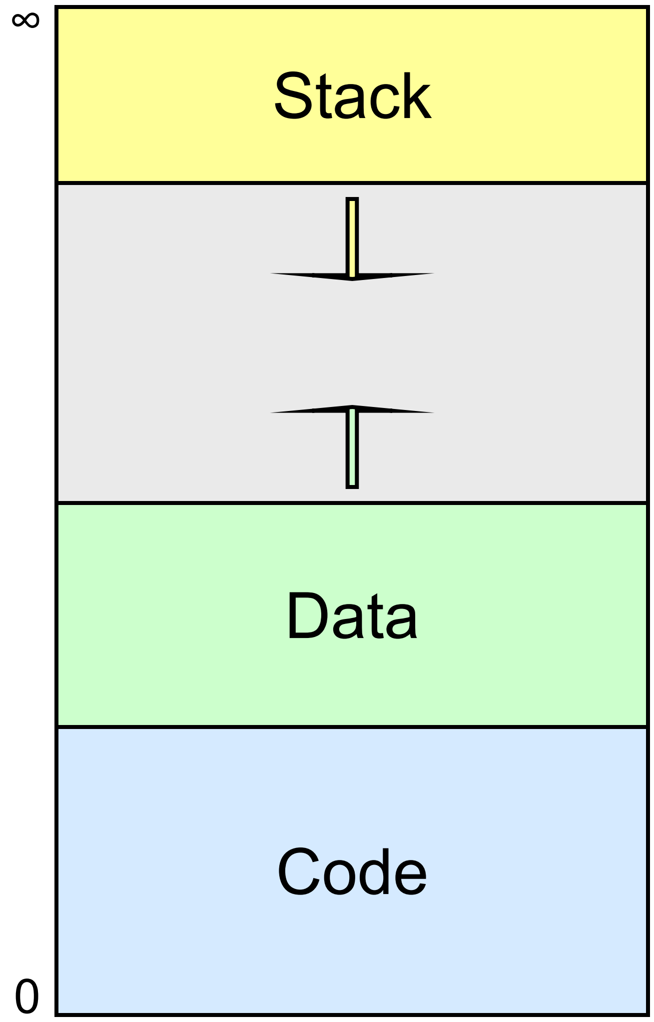
- When swapping creates multiple holes in memory, it is possible to combine them all into one big one by moving all the processes downward as far as possible.
- This technique is known as memory compaction

Swapping



- Allocating space for growing data segment
- Allocating space for growing stack & data segment

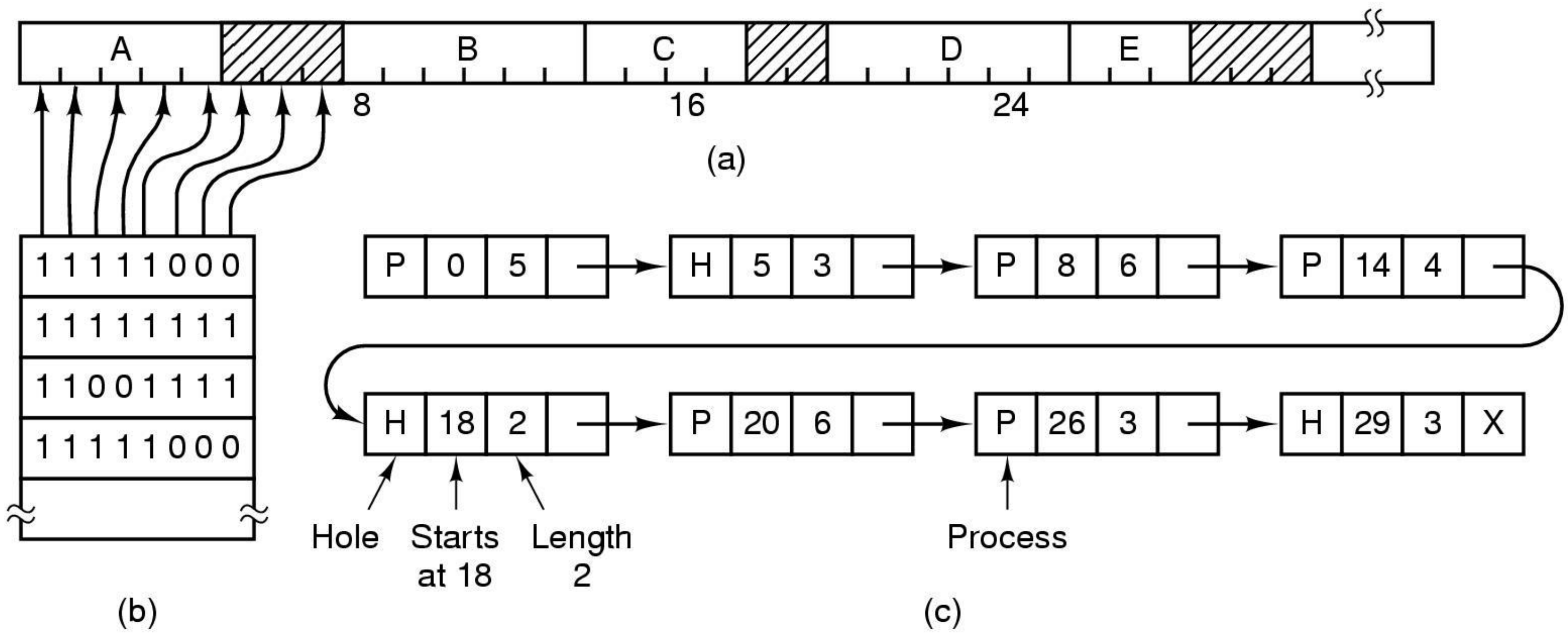
Memory Layout for Process



Managing Free Memory

- Operating system maintains information about
  - Allocated memory
  - Free memory (holes)
- there are two ways to keep track of memory usage:
  - bitmaps
  - free lists

Memory Management with Bit Maps



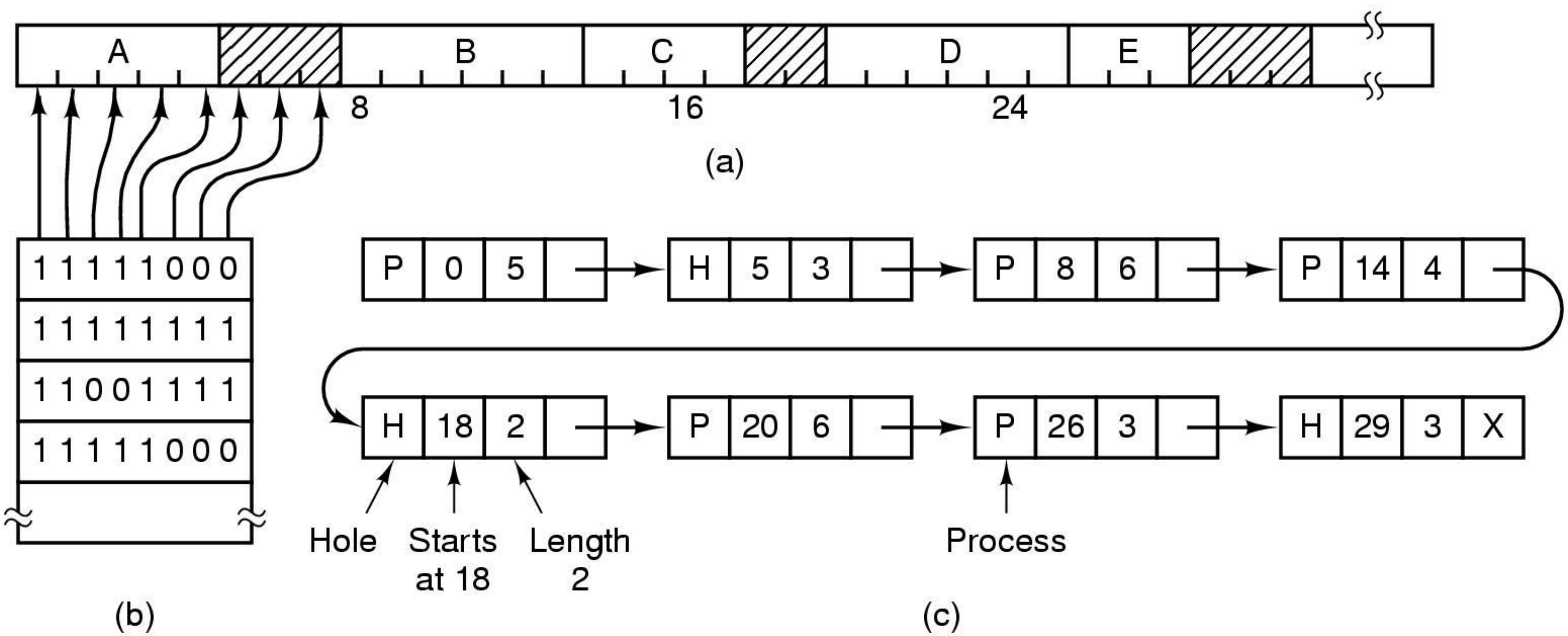
- memory is divided into allocation units
- Corresponding to each allocation unit is a bit in the bitmap

Bitmaps

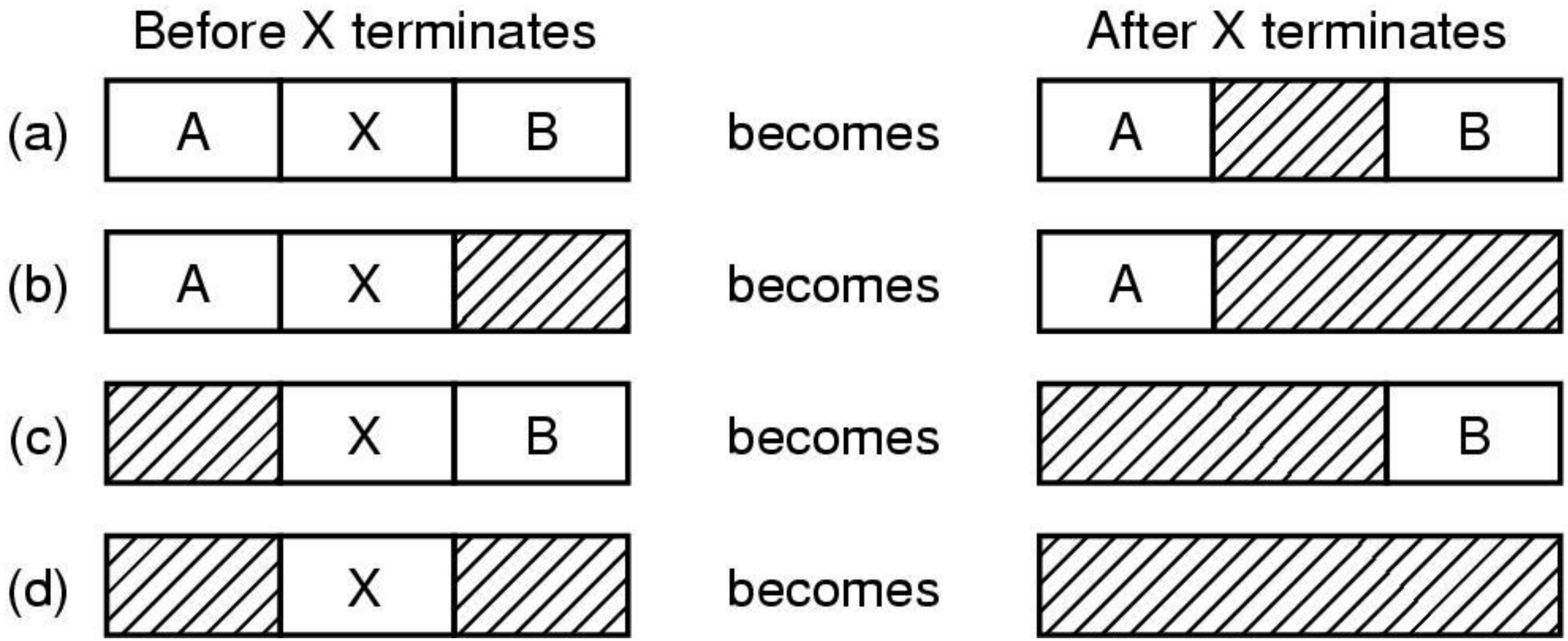
- The good-compact way to keep track of memory
- The bad-need to search memory for k consecutive zeros to bring in a file k units long
- Units can be bits or bytes or…….

Memory Management with Linked Lists

- maintain a linked list of allocated and free memory segments,
  – where a segment either contains a process or is an empty hole between two processes



Memory Management with Linked Lists



Four neighbor combinations for the terminating process X

Here the segment list is kept sorted by address

- Might want to use doubly linked lists for to merge holes more easily

Linked Lists

- Need algorithms to fill in the holes in memory
  - First fit
  - Best fit
  - Next fit
  - Worst fit
  - Quick fit

The fits

First-Fit Algorithm

- **Idea:**
  Allocate the **first hole** that is **large enough** to accommodate the process.
- The OS searches the memory from the **beginning** and stops as soon as it finds a suitable hole.
- **Advantages:**
  - Simple and fast (minimal search time).
  - Low overhead.
- **Disadvantages:**
  - May leave many **small, unusable holes** near the beginning of memory (**external fragmentation**).

The fits

Best-Fit Algorithm

- **Idea:**
  Allocate the **smallest hole** that is **big enough** for the process.

- **Advantages:**
  - Reduces wasted space in large holes.

- **Disadvantages:**
  - **Slow**, since it must check **all holes** to find the smallest that fits.

The fits

Next-Fit Algorithm

- **Idea:**
  Similar to **First-Fit**, but instead of always starting from the beginning of memory, it starts searching from **the point of the last allocation** and wraps around circularly.

- **Advantages:**
  - More evenly distributes memory use.
  - Prevents the beginning of memory from getting cluttered.
- **Disadvantages:**
  - Slightly more overhead than first-fit.

The fits

Worst-Fit Algorithm

- **Idea:**
  Allocate the **largest available hole**, leaving the biggest remaining space for future allocations.

- **Advantages:**
  - May reduce the number of small holes.

- **Disadvantages:**
  - Often **wastes large memory blocks**.

The fits

Quick-Fit Algorithm

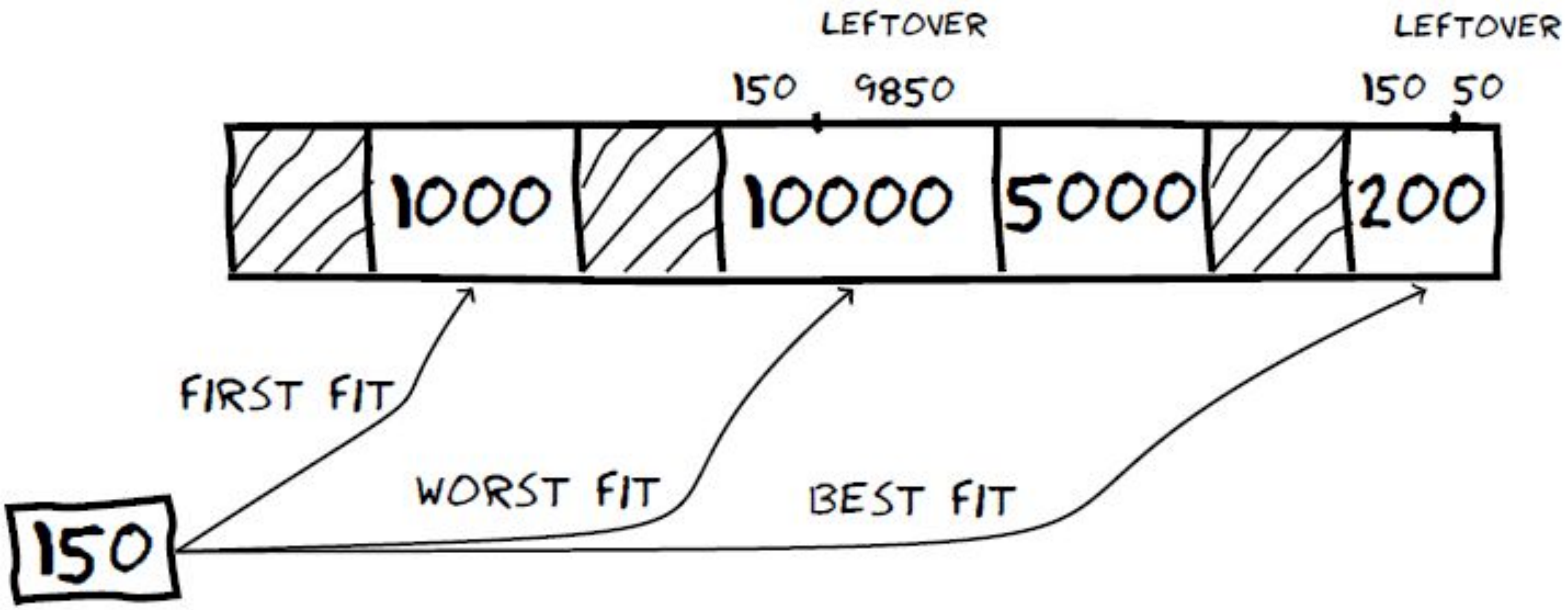
- **Idea:**
  Maintains **separate lists** of holes for **commonly requested sizes** (e.g., 4 KB, 8 KB, 16 KB). When a process arrives, the OS directly looks into the corresponding list.

- **Advantages:**
  - **Very fast** allocation for frequent sizes.

- **Disadvantages:**
  - Requires **extra bookkeeping** to maintain multiple lists.

Memory Management with Linked Lists

- Memory allocation algorithms : First fit, next fit, best fit, worst fit, quick fit



The fits

- Conclusion: the fits couldn't out-smart the un-knowable distribution of hole sizes
- The extra work to deal with something which you can't predict failed to produce good results
- Shocking!

Swapping: Conclusion

- Swapping requires that an entire process be in a single contiguous section of memory before it can execute.
- Swapping is not an attractive option as transferring data with disk is a slow operation