IT2164/IT2561 Operating Systems and Administration

Chapter 8

Memory Management

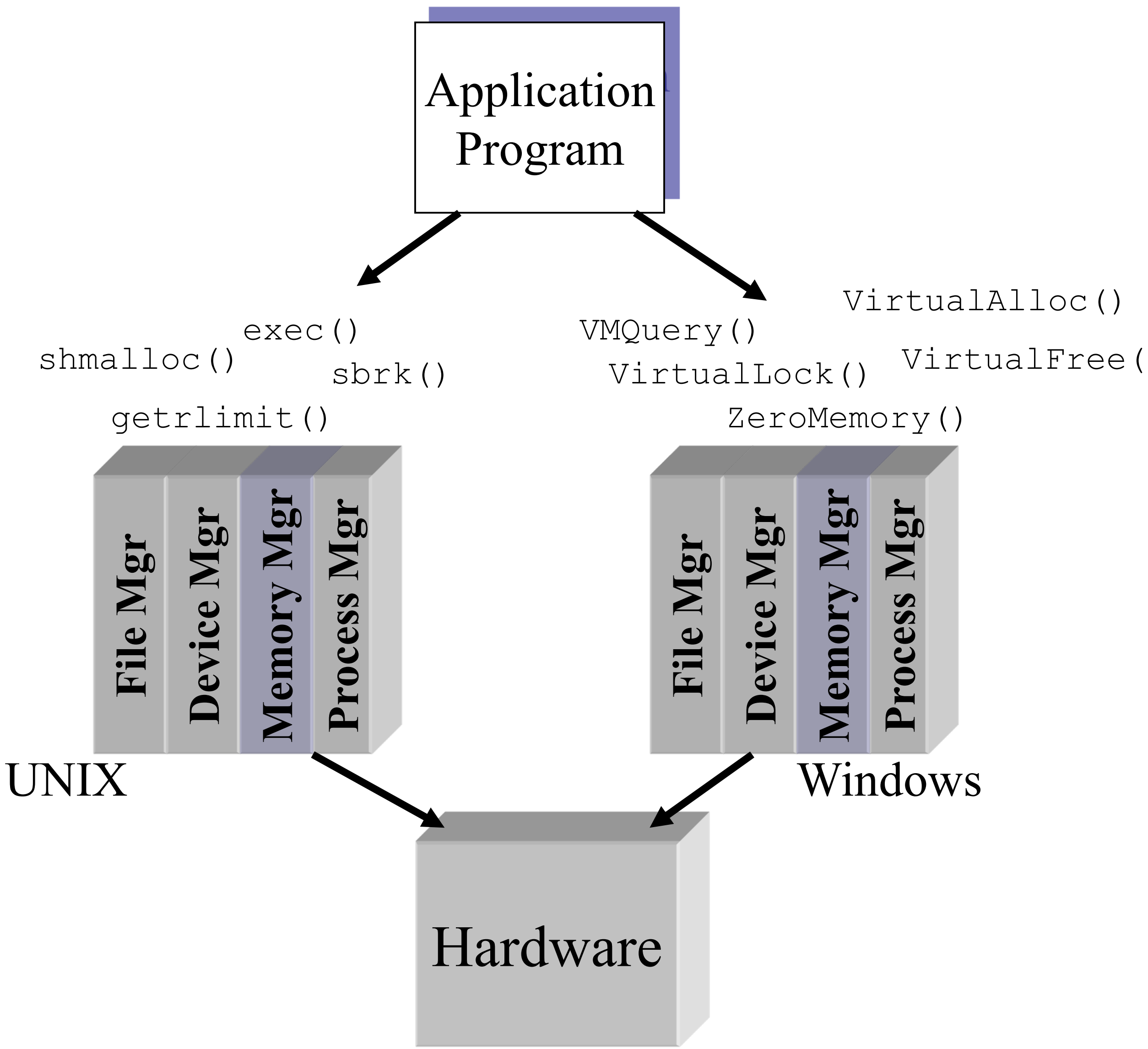
Objectives

- After this lesson, you will be able to :
  - Understand how memory is managed in the computer system.
  - Understand different memory allocation schemes.
  - Know how modern memory management strategies are applied in the OS.

Memory Management

- Memory in a computer system is divided into primary and secondary memory.
- Primary memory hold information (data and programs) while the CPU is using it.
- Secondary memory refers to storage devices that store information when the CPU is not using it.
- Memory manager is responsible for allocating and deallocating primary memory.
- It ensures that the memory is not abused and is used efficiently.

The External View of the Memory Manager



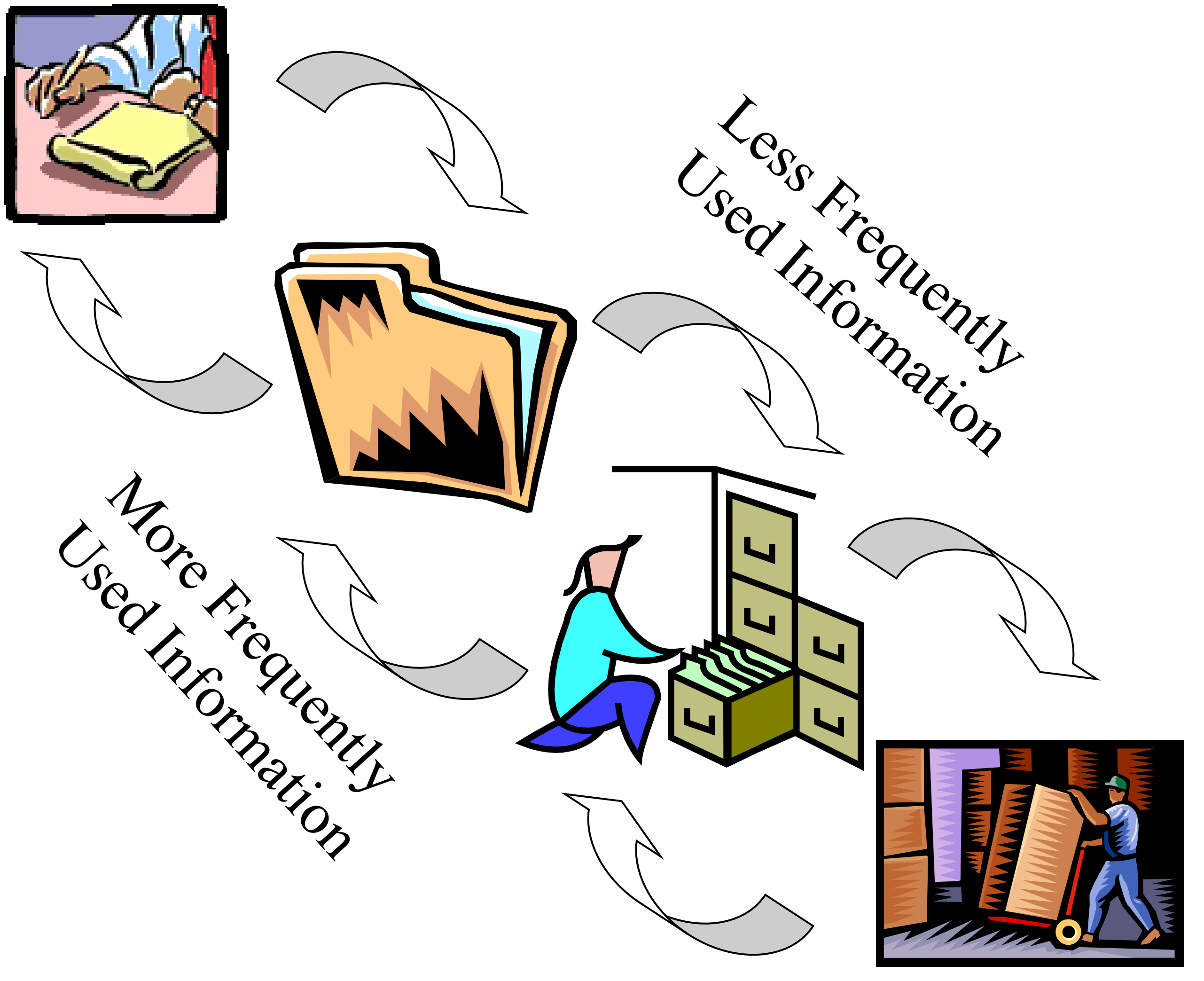
Memory Manager

- Requirements
  - Minimize executable memory access time
  - Maximize executable memory size
  - Executable memory must be cost-effective
- Today’s memory manager:
  - Allocates primary memory to processes
  - Maps process address space to primary memory
  - Minimizes access time using cost-effective memory configuration
  - May use static or dynamic techniques

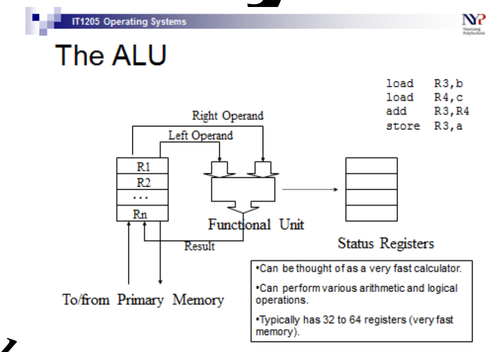
Storage Hierarchies

- Memory in a computer system can be divided into an hierarchy based on their characteristics.
- It also reflects how we work in the real world.
- At the top of the hierarchy is faster, but more expensive memory, e.g., registers
- At the bottom is slower, but cheaper memory, e.g., hard disk, tape drives.

Storage Hierarchies



The Basic Memory Hierarchy



CPU Registers

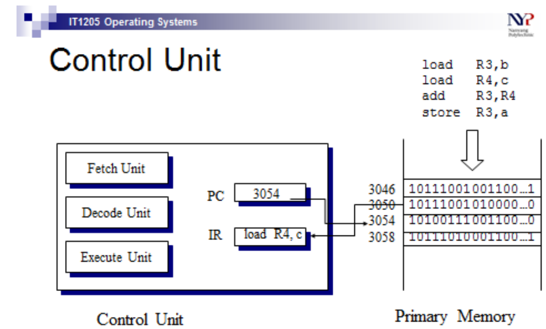
Less Frequently Used Information

Primary Memory
(Executable Memory)
e.g. RAM

More Frequently Used Information

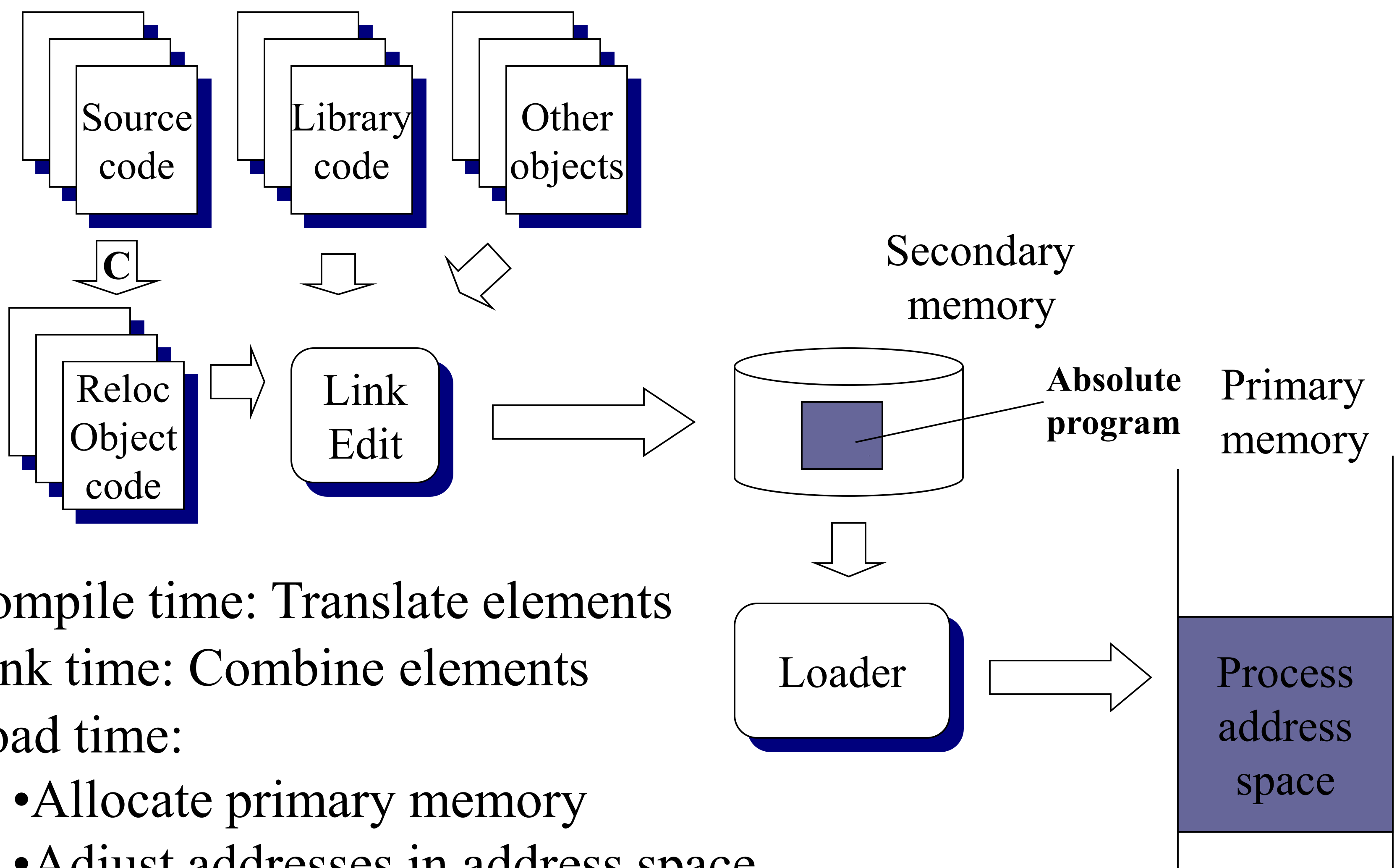
Secondary Memory
e.g. Disk or Tape

Managing Address Space



- When a program is converted (compiled) into executable form, there is a need to attach addresses to every single instruction in the program.
- Why?
  - Program instructions need to be stored in a particular order in order for the CPU to execute in the correct way.
  - Programs involve many ‘jumps’ to different parts of the program. In order to do this, there has to be a concept of addresses for every part of the program.
  - *Imagine someone asking you to go to a person’s house. Unless the address is given to you, there is no way you can get to that house.*
- This process is called Address Binding

Creating an Executable Program



- Compile time: Translate elements
- Link time: Combine elements
- Load time:
  - Allocate primary memory
  - Adjust addresses in address space
  - Copy address space from secondary to primary memory

Address Binding

- There are 3 ways to ensure that when a program is loaded up into memory, each instruction has an address.
- They are :
  - During compile time,
  - During load time and,
  - During run time.

Address Binding

- Compile Time binding
  - Addresses are attached to each instruction when program is compiled.
  - Program is loaded into same location in memory whenever it is executed.
  - Simple, and good for single-programmed systems. E.g., MS-DOS
  - Cannot be relocated into different locations in memory.

Address Binding

- Load time binding
  - During compilation, the program is attached with re-locatable addresses. E.g., start at 0x0000.
  - When program is loaded into memory, the location of the first instruction is loaded into a special register.
  - To access the program, take re-locatable address of instruction + value of special register.
  - Once loaded into memory, the program cannot be relocated into another portion of the memory.

The Absolute Program

Code Segment

Relative Address	**Generated Code**	
0000	(Other modules)	
...		
1008	entry	proc_a
...		
1220	load	=7, R1
1224	store	R1, 0136
1228	push	1036
1232	call	2334
...		
1399	(End of proc_a)	
...	(Other modules)	
2334	entry	put_record
...		
2670	(optional symbol table)	
...		
2999	(last location in the code segment)	

Data Segment

Relative Address	**Generated variable space**
...	
0136	[Space for gVar variable]
...	
1000	(last location in the data segment)

The Program Loaded at Location 4000

Relative Address	Generated Code
0000	(Other process's programs)
4000	(Other modules)
...	
5008	entry proc_a
...	
5036	[Space for gVar variable]
...	
5220	load =7, R1
5224	store R1, 7136
5228	push 5036
5232	call 6334
...	
5399	(End of proc_a)
...	(Other modules)
6334	entry put_record
...	
6670	(optional symbol table)
...	
6999	(last location in the code segment)
7000	**(first location in the data segment)**
...	
7136	[Space for gVar variable]
...	
8000	(Other process's programs)

Address Binding

- Run time binding
  - Address binding is delayed until run-time.
  - Similar in concept to load time binding, i.e., using a relocatable register.
  - Address of instruction is calculated only when the instruction is required.
  - Very flexible as program can be relocated into any portion of memory at any time.
  - Used by modern OS for multi-programming.

Logical vs Physical Address Space

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.
  - Logical address is generated by the CPU; it is also referred to as a virtual address.
  - Physical address is the address seen by the memory unit.
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes.
- Logical and physical addresses differ in run-time address-binding scheme.
- The set of all logical addresses is referred to as a logical address spaces and the set of all the corresponding physical addresses is referred to as physical address space.

Memory Allocation

- With multi-programming, the computer system need to allocate memory to many programs with differing sizes.
- When a new program is loaded or when a program terminates, the computer system must be able to allocate memory or reclaim the memory for re-use.

Memory Allocation

- Several strategies exist :
  - Fixed partition
    - Primary memory is divided into a fixed number of fixed-size blocks/partitions
    - Partitions are typically not of equal size
    - Prone to internal fragmentation
  - Variable partition
    - Memory is allocated to processes on a need-to and available basis
    - Use dynamically determined, variable-sized blocks
    - Prone to external fragmentation.
    - Requires run-time address binding to resolve fragmentation problems.

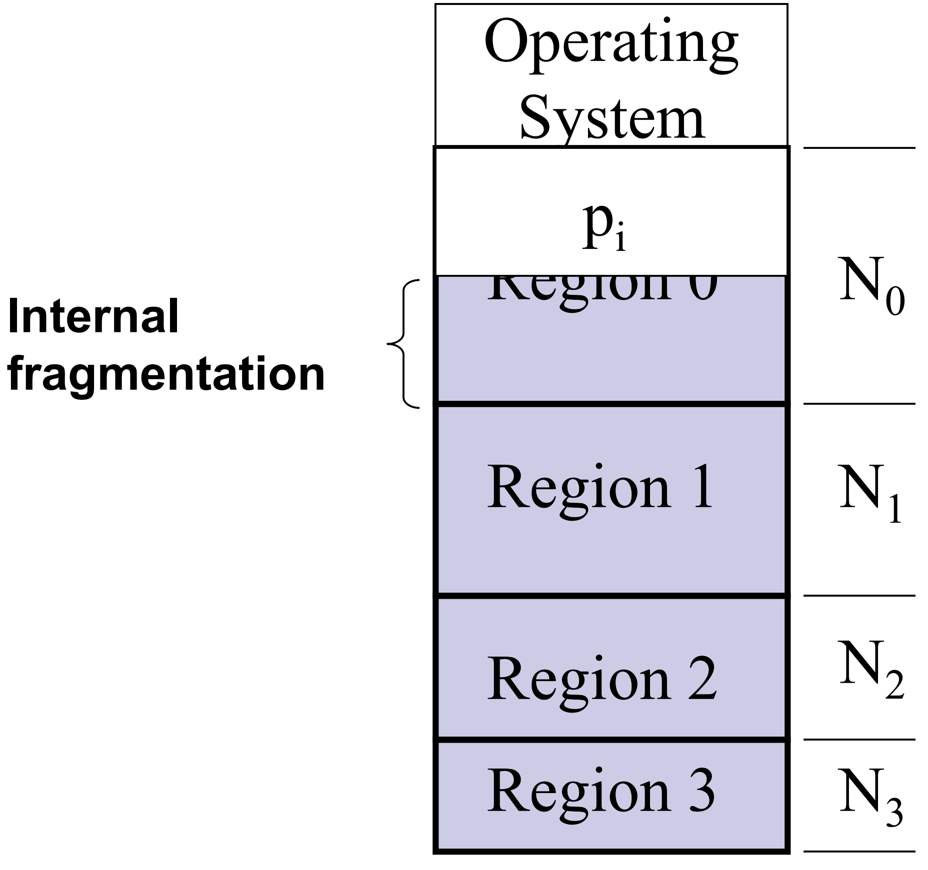
Memory Allocation

- **Fragmentation**
  - The perpetuation of small memory fragment.
  - Ideally, the memory manager could allocate every single byte of memory to a process if any process needs memory. However, in practical terms, parts of the memory, called **memory fragment**, cannot be used at any given time because memory manager is unable to allocate these parts in an efficient manner.
  - Two types of fragmentation:
    - Internal fragmentation
    - External Fragmentation

Memory Allocation

- **Internal Fragmentation**
  - Memory that is internal to a partition, but is not being used.
  - Eg, If a process is allocated X amount of memory but needed only Y, the amount of memory (X-Y) cannot be used by other processes, and thus is ‘wasted’. This phenomenal is known as internal fragmentation.

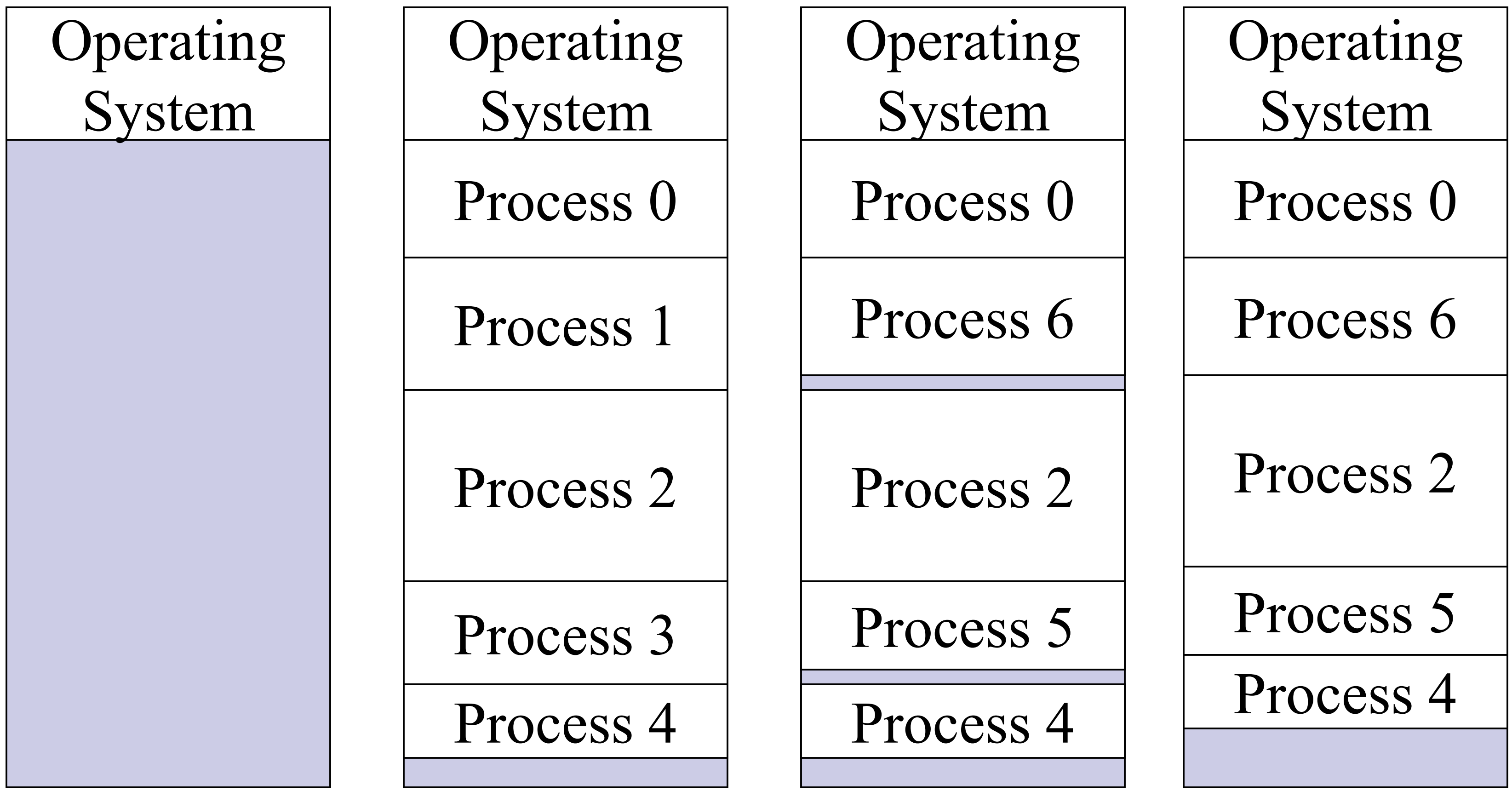
Fixed partition memory



Memory Allocation

- **External Fragmentation**
  - As processes are loaded and removed from memory, the free memory space are fragmented into pieces.
  - External fragmentation exists when enough total memory space exists to satisfy a request, but the request cannot not be granted because the memory is fragmented such that there is no single contiguous memory space large enough to allocate to the request.

Variable Partition Memory



Loader adjusts every address in every absolute module when placed in memory

- **External fragmentation**
- **Compaction** moves program in memory

Compaction

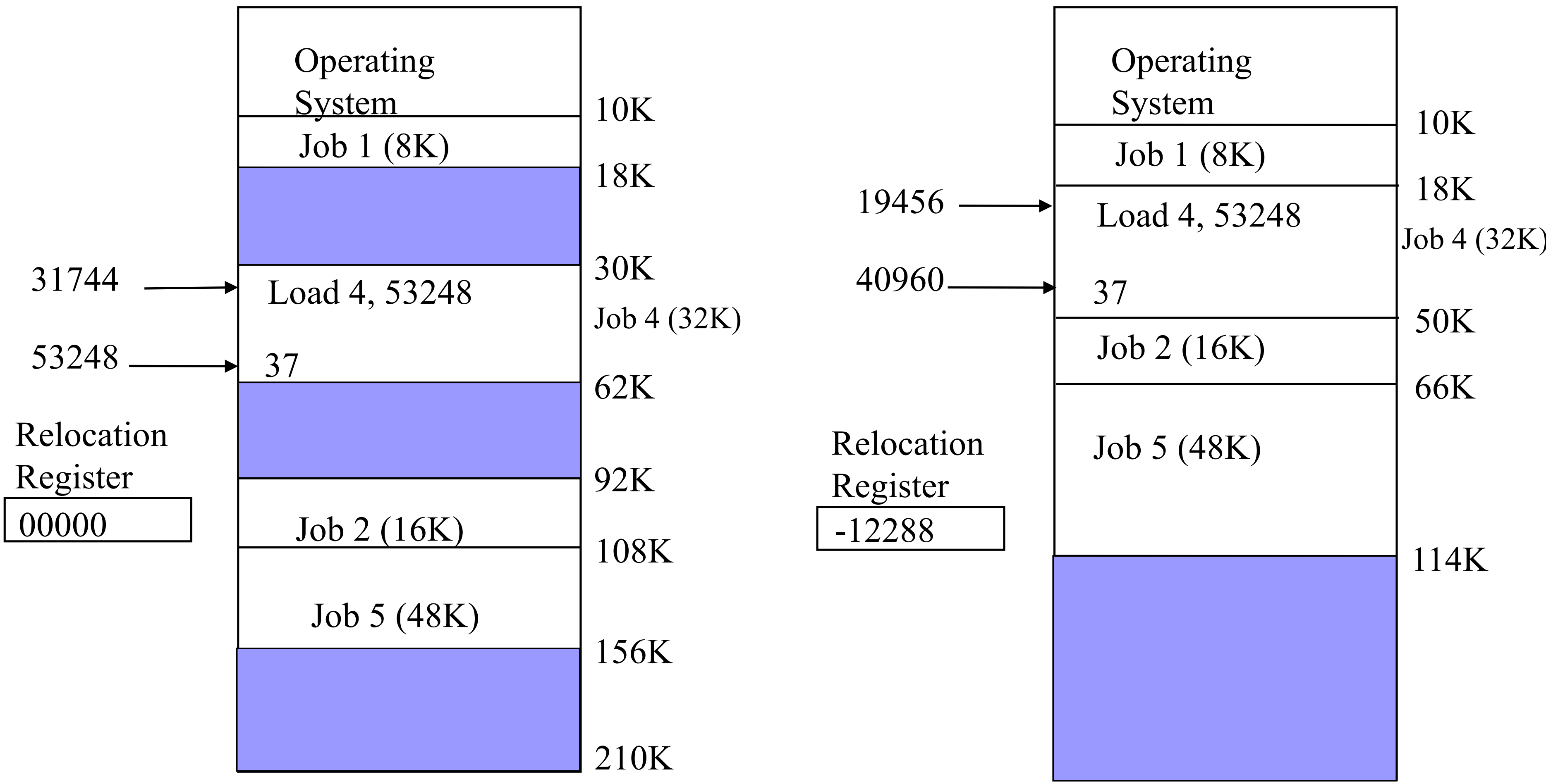
- The goal is to shuffle the memory contents to place all free memory together in one big block.
- It relocate the processes in memory in order to recombine fragmented free memory.
- It solves the problem of external fragmentation.

Compaction

$1K = 2^{10} = 1024$

31744-12288=19456

53248-12288=40960



Initial memory layout

Memory layout after compaction

Compaction

- Compaction costs CPU time to perform.
- Is an overhead operation.
- When and how often it should be done?
  - when there are jobs waiting to get in.
  - after a prescribed amount of time has elapsed.

Conclusion

- Memory management plays an important role in a computer system, as it is used 100% of computer system's functioning.
- Features and policies implemented by OS will determine the performance of the computer system.
- Memory management techniques continue to evolve as computer technology improves.