Operating Systems: Main Memory – Part I

Neerja Mhaskar

Department of Computing and Software, McMaster University, Canada

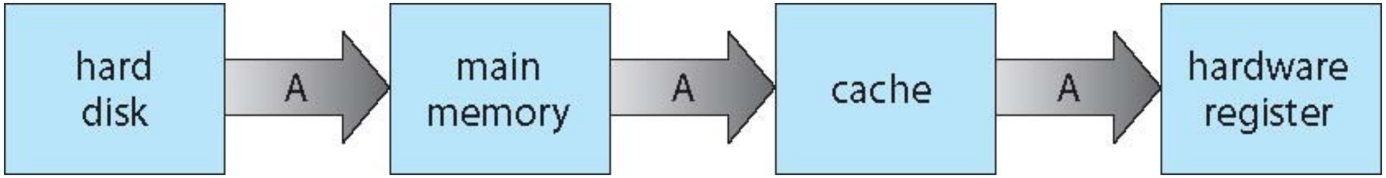
Acknowledgements: Material based on the textbook Operating Systems Concepts (Chapter 9)

Quick Recap

- We covered Process management so far.
  - Processes (executing programs) creation.
  - Process synchronization and resource sharing.
  - Deadlock prevention, avoidance and detection schemes.
  - Process scheduling.

Process life cycle

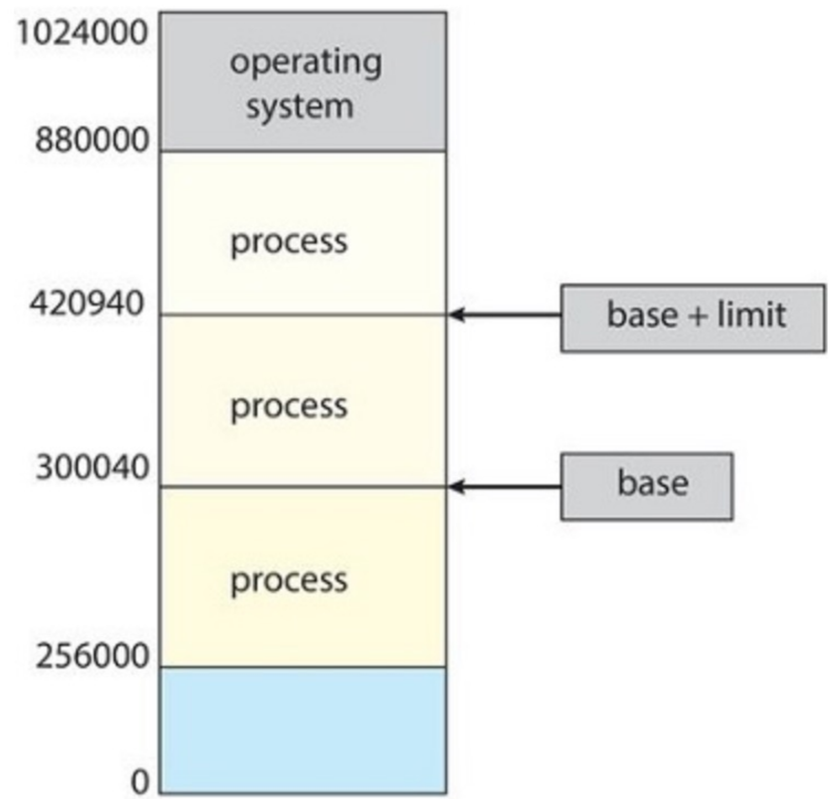
- Program resides on disk.
- Must be brought from disk into main memory (array of bytes) for execution.
- As the process executes on the CPU it accesses instructions and data from memory.



- Main memory, cache and registers are **only storage CPU can access** directly

Memory protection - Multi-programming environment

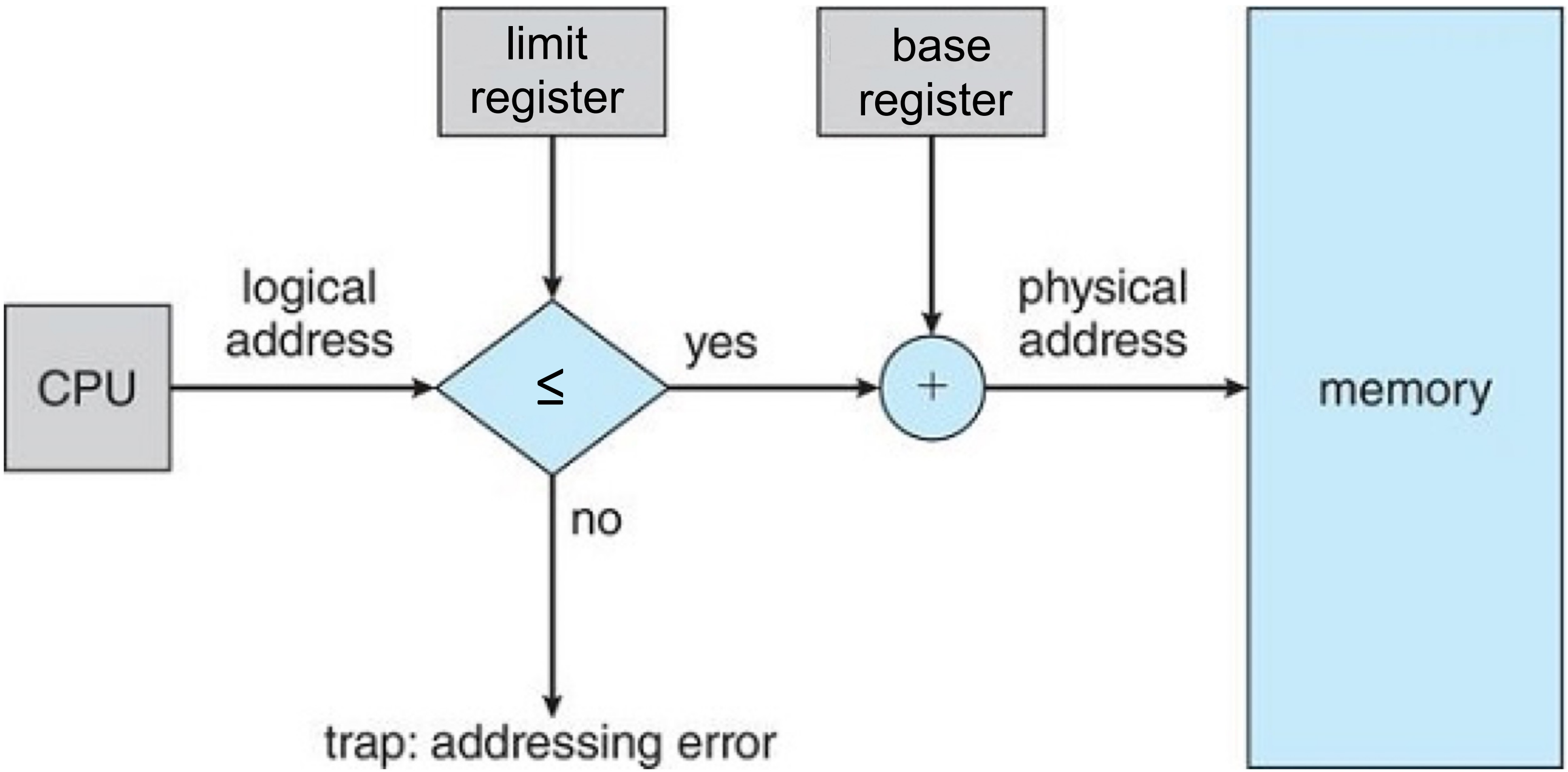
- In a multi-programming environment, *many processes are in main memory* each having its own memory space.
- Protection of processes' memory is required to ensure correct operation
- This protection is provided by the below **hardware.**
  - **Base registers** - smallest legal physical memory address, and
  - **Limit registers** - size of the address space of the program.



Hardware Address Protection with Base and Limit registers

- During execution, a process accesses data and instructions in memory.
  - CPU generates addresses from 0.
  - However, instructions are not stored in memory at address 0
- CPU hardware compares every address generated with the base and limit registers
  - Let’s it access the memory address only if its with in the legal range
  - Base and limit register values loaded only by the operating system

Hardware Address Protection with Base and Limit registers



Question

Consider a system in which memory protection is achieved using the **Base and Limit registers**. Suppose the value in base register = 1200 and value in limit register = 1000.

a) If the logical address generated by the CPU = 32. To what physical address is this logical address mapped to?

b) If the logical address generated by the CPU = 1500. To what physical address is this logical address mapped to?

Logical Address Space

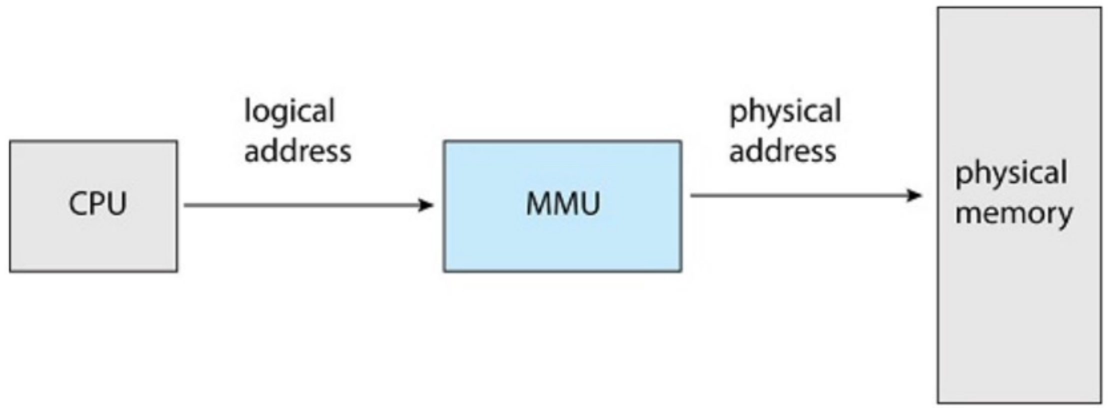
- **Logical address** – addresses generated by the CPU
  - Also referred to as **virtual address**
  - Range from 0 to *max* = size of the address space of the program
- **Logical address space** is the **set of all logical** addresses generated by a program

Physical Address Space

- **Physical address** – address in main memory and seen by the memory management unit
  - Range from $R + 0$ to $R + max$, where R = base value.
- **Physical address space** is the **set of all physical** addresses assigned to the process
- *The logical addresses must be mapped to physical addresses before they are used.*

Memory-Management Unit (MMU)

- **Memory-Management Unit (MMU)** – is a **hardware device** that at run time maps/translates logical addresses to physical addresses

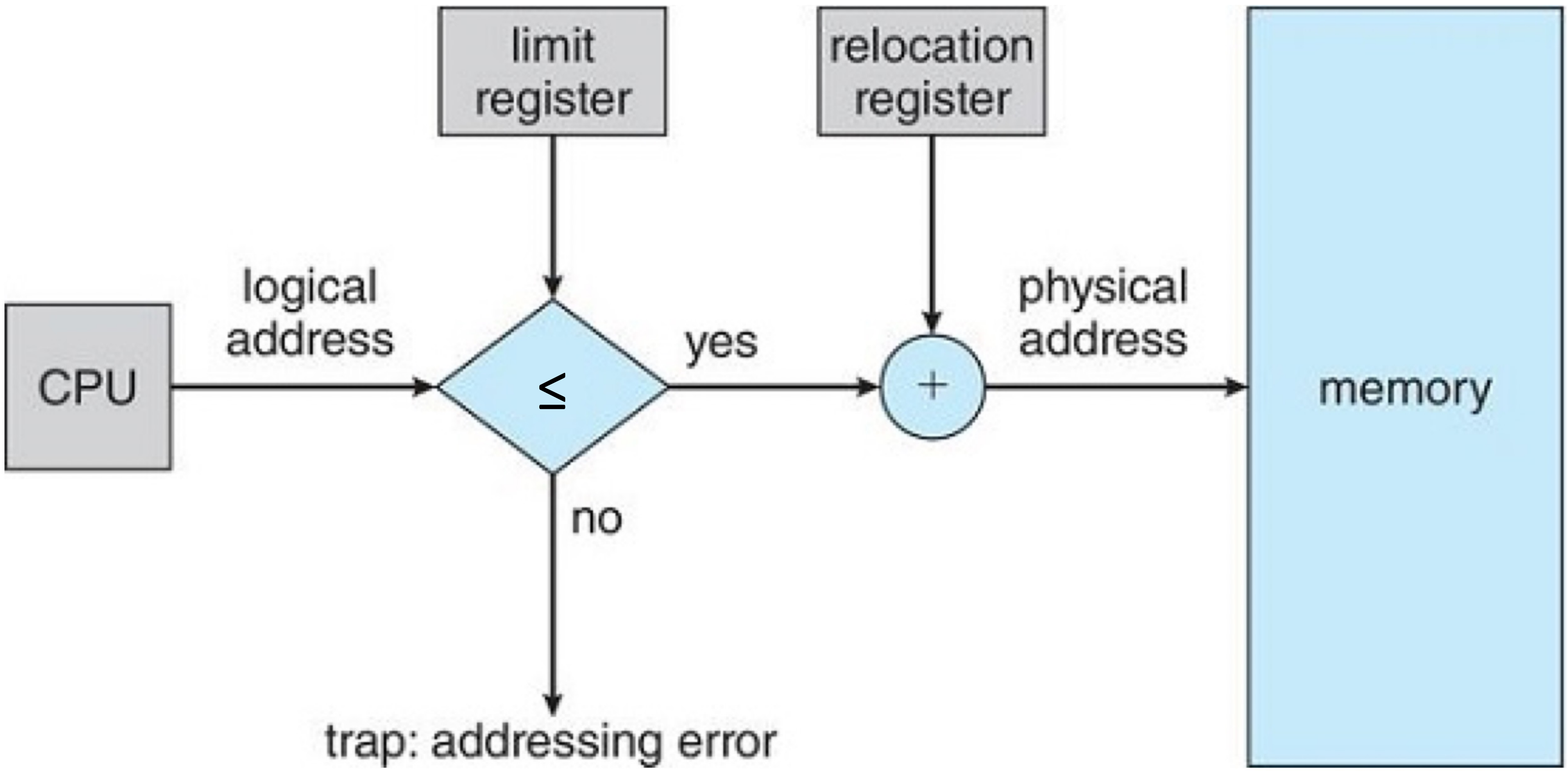


- This mapping is based on below schemes
  - Contiguous memory allocation (obsolete)
  - Segmentation
  - **Paging** (most popular, used in PCs, mainframes to smart phones.)

Contiguous Memory Allocation

- Main memory usually divided into two **partitions**:
  - Operating system held in low memory with interrupt vector
  - User processes held in high memory
    - Each process contained in single **contiguous** section of memory.
    - Memory protection in this scheme is achieved using *Relocation (base) and limit registers*.

Hardware support for relocation and limit registers



Question

- Consider a system in which the logical addresses are mapped to physical addresses using relocation registers. Suppose the value of the relocatable register =14000 and the limit register = 1200.
  - If the CPU generated a logical address = 1300. To what physical address is the logical address mapped to?

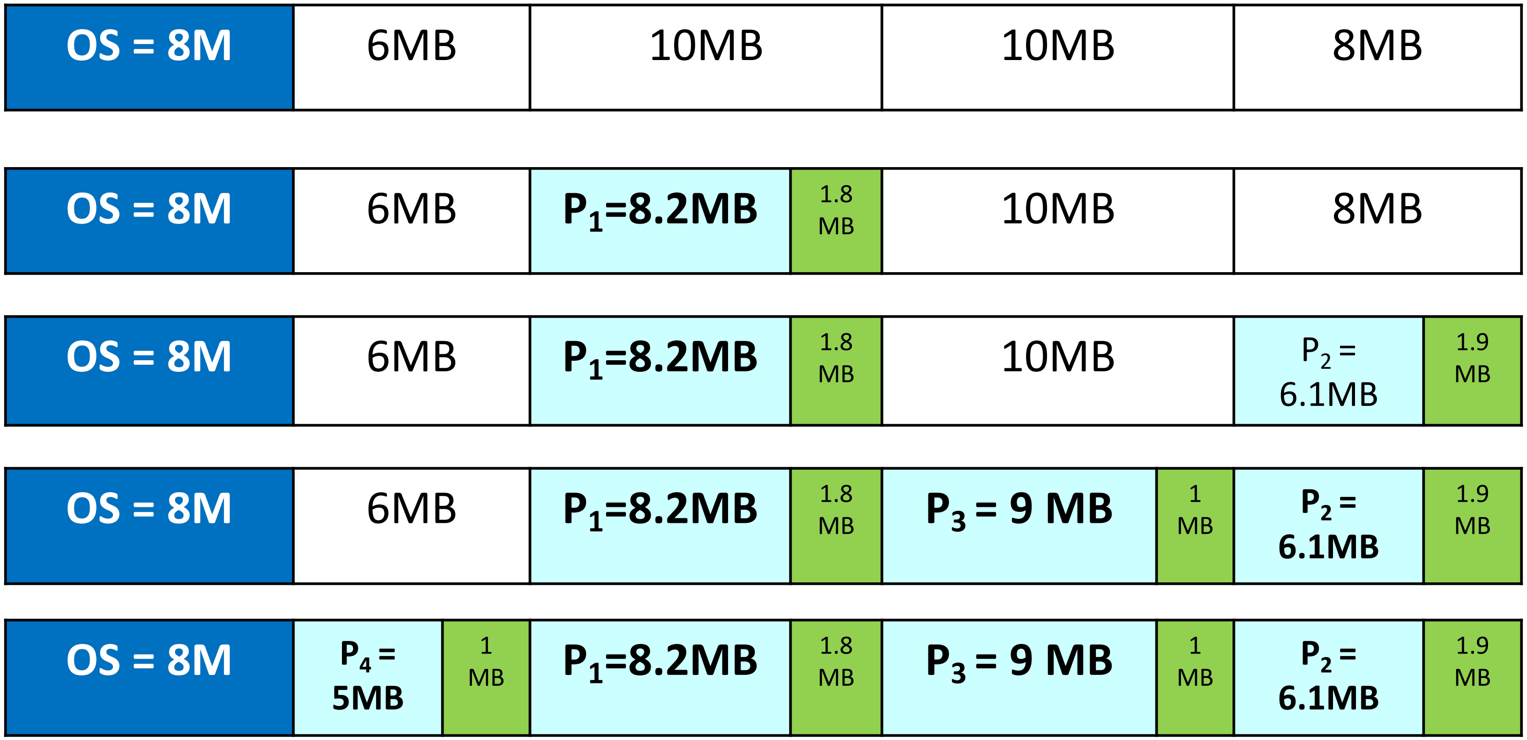
Contiguous Allocation - Partitioning

- Contiguous memory allocation can be achieved in two ways:
  - Fixed partitioning
  - Variable or dynamic partitioning

Contiguous Allocation - Partitioning

- **Fixed partitioning:** Main memory is divided into a number of **static partitions** (possibly of **different sizes**) at system generation time.
- A process may be loaded into a partition of equal or greater size.
- Maximum number of active processes is fixed.
- Simple to implement
- Suffers from internal fragmentation
  - **Internal Fragmentation** – Wasted memory due to fixed usable memory chunks.

Fixed partitioning example



- Maximum number of possible active processes = ?
- Can process P_5 = 3MB be brought into main memory?

Contiguous Allocation – Variable Partitioning

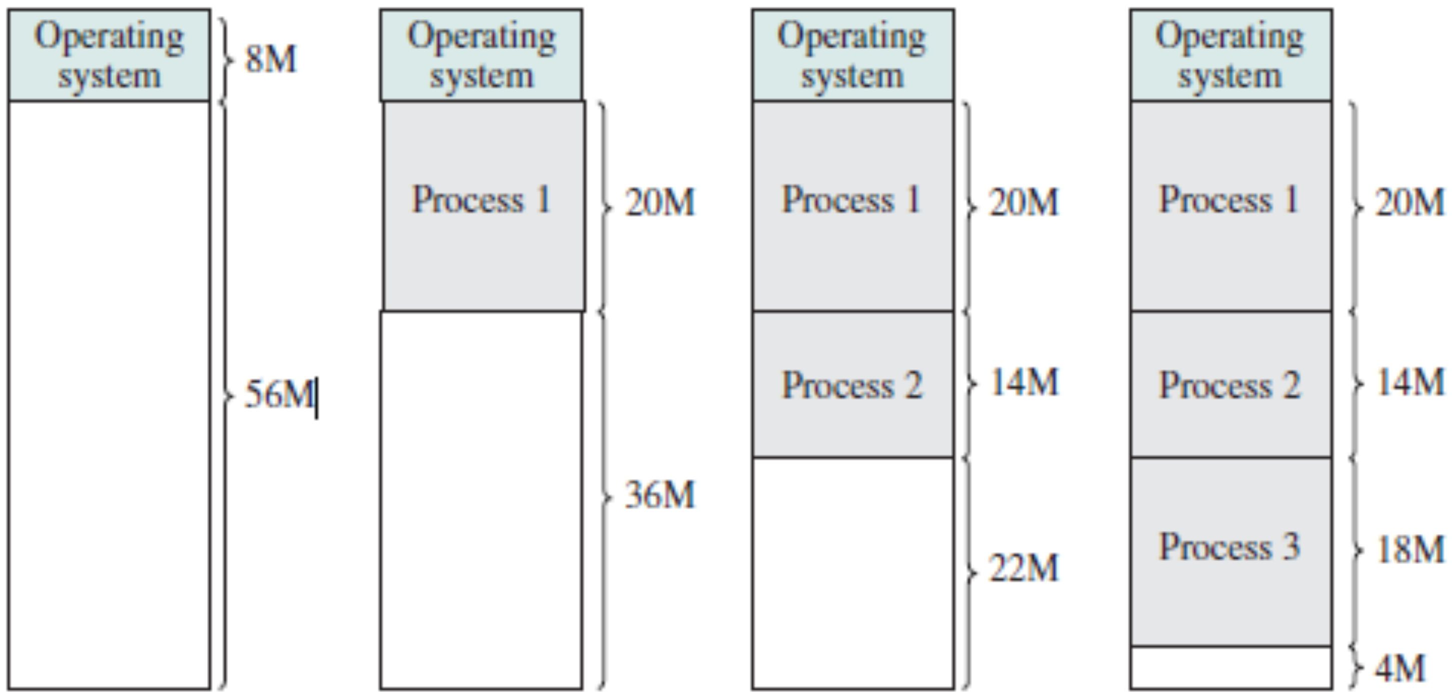
- **Variable or dynamic partitioning:**
  - The operating system keeps a table indicating which parts of memory are available and which are occupied.
  - Initially, all user memory is available for user processes as one large block of available memory (**hole**).
  - Eventually, as processes move in and out of memory, the main memory contains a set of holes of various sizes.
  - No internal fragmentation but suffers from external fragmentation.

Contiguous Allocation – Variable Partitioning

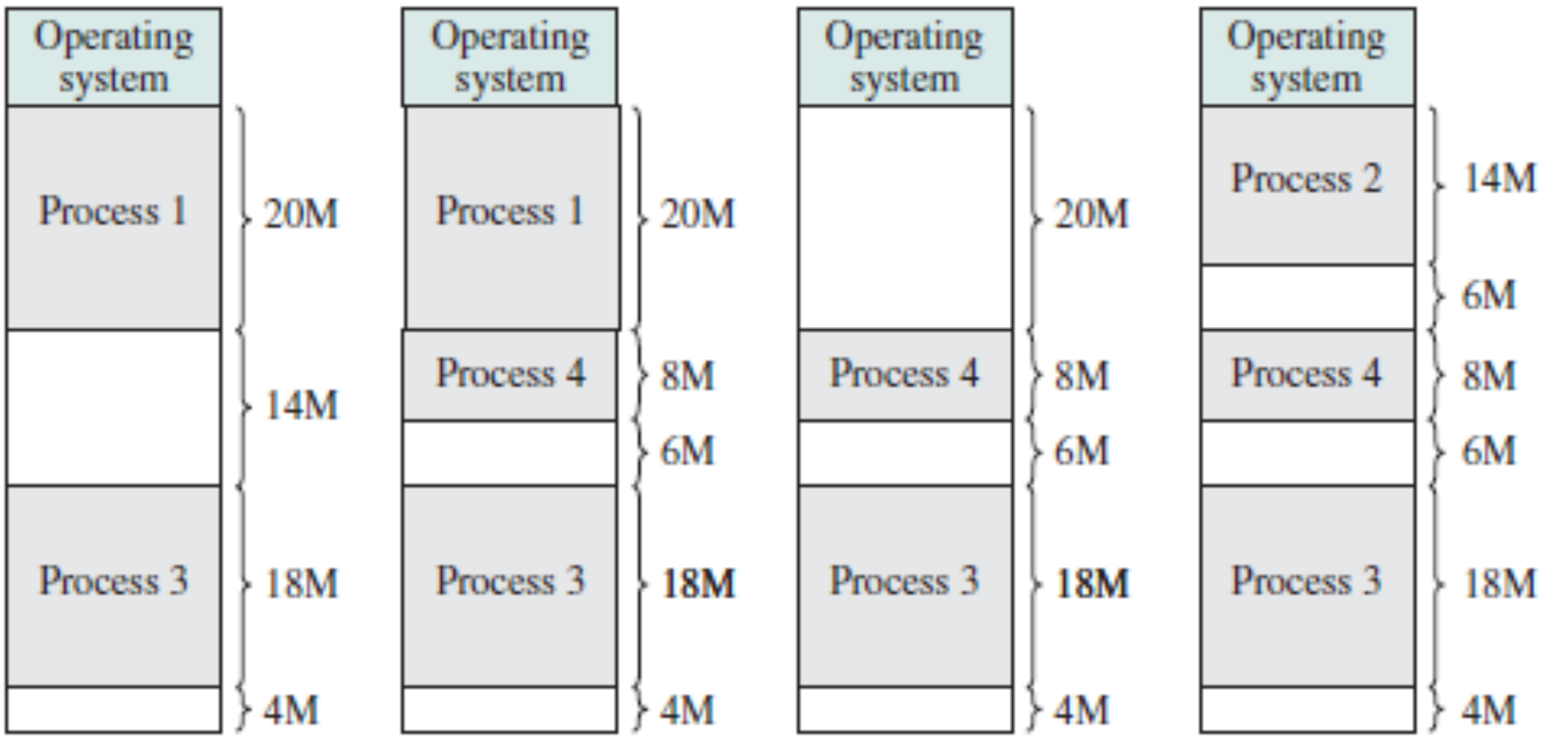
- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- External fragmentation problem can be solved by compaction.
- **Compaction** - Shuffle memory contents to place all free memory together in one large block.

Dynamic partitioning example

Memory = 64M



Dynamic partitioning example Cont…



- Can process P_5 = 10 MB be brought into main memory?
- Can process P_6 = 17MB be brought into main memory?

Dynamic Storage-Allocation Problem

How to satisfy a request of size ***n*** from a list of free holes?

- **First-fit**: Allocate the ***first*** hole that is big enough
- **Best-fit**: Allocate the ***smallest*** hole that is big enough
  - must also search entire list
- **Worst-fit**: Allocate the ***largest*** hole available
  - must also search entire list
- Simulations indicate that First-fit and best-fit better than worst-fit in terms of speed and storage utilization.