CS343: Operating System

Memory Management

Lect30 : 16th Oct 2023

Dr. A. Sahu

Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

Outline

- Memory Management
  - Continuous Memory allocation
  - Buddy System
  - Segmentation
  - Paging

Protection Issue

- Long term scheduler may put many process in to Ready at a time
  - Where to put ?
  - How to access them ?
  - Is there any efficient way to put and access?
  - How ensure safety and protection?

Memory Allocation: Top Down

Assume 4GB of RAM,
Avg Process Size 200MB
4 Segments/Process

OS
P5
P9
P8

P2

Finding Hole
First Fit
Worst fit
Best Fit

Very low complexity

Very High Mem Wastage

Process Continuous Allocation
10 process and 20 holes

Segment continuous Allocation
40 segment and 80 holes

low complexity

High Memory Wastage

Paging: 200MB process, 4KB page
Two Issue: pages and frames

Very Low Mem Wastage

High Complexity: 51200 pages/Process

Loading 51K pages in 10^6 frames

High Complexity: $4GB/4KB=2^{20}=10^6$ frames

SOL

Demand Paging Load the required 50-100 pages and load as an when necessary

4GB paged

SOL

Allocate 100-200 frames per process: reduce search Space

Logical vs. Physical Address Space

- 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; also referred to as **virtual address**
- **Physical address** – address seen by the memory unit

Logical vs. Physical Address Space

- **Logical (Virtual) and physical addresses**
  - Are same in compile-time and load-time address-binding schemes
  - Differ in execution-time address-binding scheme
- **Logical address space**
  - Set of all logical addresses generated by a program
- **Physical address space**
  - set of all physical addresses generated by a program

Swapping

- Swapping
  - A process can be **swapped** temporarily out of memory to a backing store
  - And then brought back into memory for continued execution
- Total physical memory space of processes can exceed physical memory
- **Backing store (Ex : HDD/SDD)**
  - fast disk large enough to accommodate copies of all memory images for all users;
  - must provide direct access to these memory images

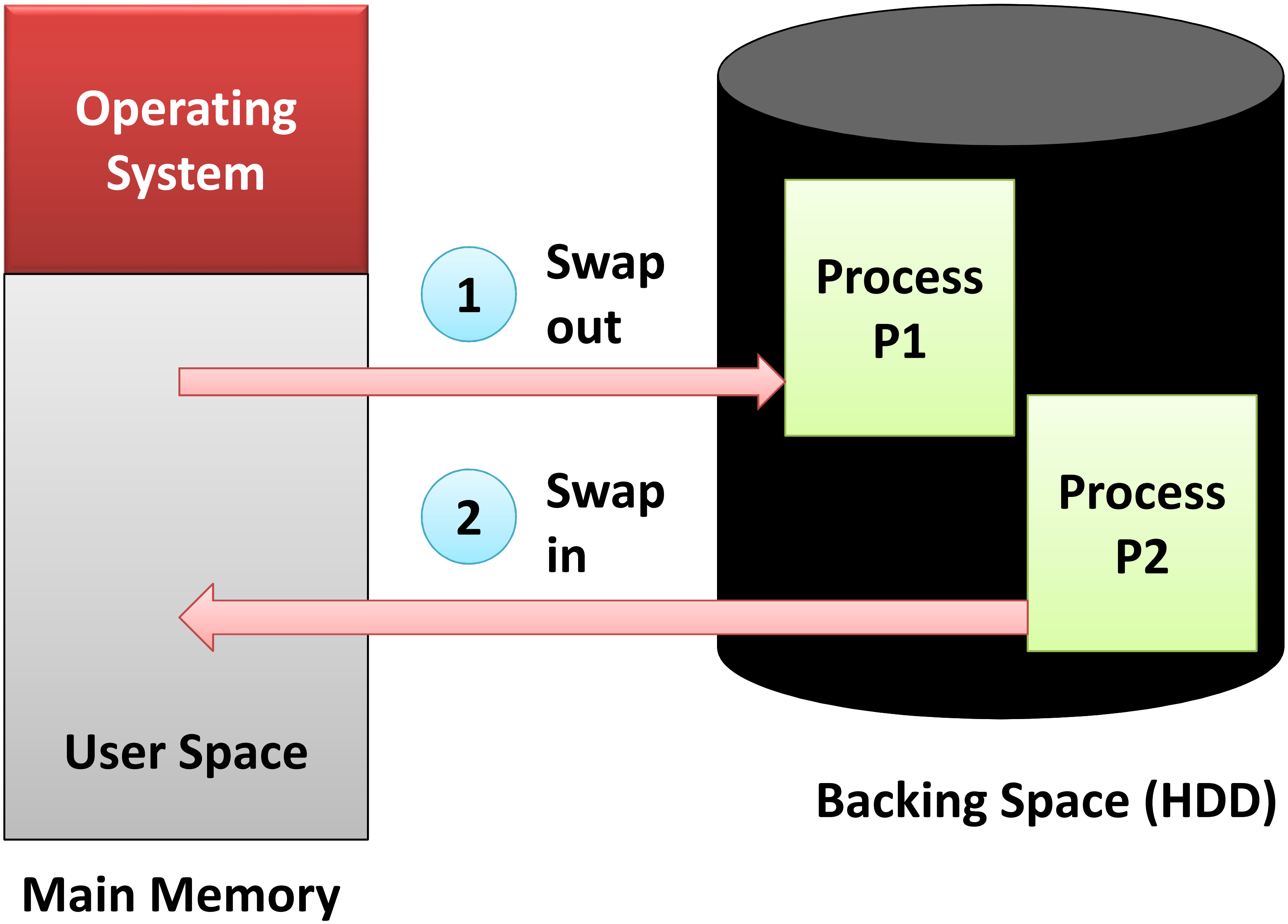
Swapping

- **Roll out, roll in**
  - Swapping variant used for priority-based scheduling algorithms;
  - Lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time
  - Total transfer time is directly proportional to the amount of memory swapped
- System maintains **queue**
  - Ready-to-run processes which have memory images on disk

Swapping (Cont.)

- Does the swapped out process need to swap back in to same physical addresses?
- Depends on address binding method
  - Plus consider pending I/O to / from process memory space

Schematic View of Swapping



Context Switch Time including Swapping

- If next processes to be put on CPU is not in memory
  - Need to swap out a process and swap in target process
  - Context switch time can then be very high
- 100MB process swapping to hard disk with transfer rate of 50MB/sec
  - Swap out time 2000 ms + Swap in of same sized process = 4 seconds
  - Swapping component of Context Switch time is 4 seconds

Context Switch Time including Swapping

- Can reduce if reduce size of memory swapped – by knowing how much memory really being used
  - Read only memory Discard
    - Code and Declared Read only Data, Constant Area
  - System calls to inform OS of memory use via `request_memory()` and `release_memory()`

Real System :Memory SwapIn/SwapOut

- Server : Utilization is high, always get a crunch…..Supposed to utilized all resources
- PC (4GB RAM, 500GB HDD)
  - Most of the time RAM utilization is bellow 40%
  - Most of the time we get a space for our process to Run….. :) :)
- Mobile (256MB RAM, 8GB SSD)
  - SSD is faster, Read takes less time but write take time, Number write to SSD is limited by a number in SSD life time

Real System :Memory SwapIn/SwapOut

- In PC
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - Swapping normally disabled
  - Started if more than threshold amount of memory allocated
  - Disabled again once memory demand reduced below threshold

Swapping on Mobile Systems

- Not typically supported : Flash memory based
  - Small amount of space
  - Limited number of write cycles (NVRAM, STT RAM, PCM RAM)
  - Poor throughput between flash memory and CPU on mobile platform

Swapping on Mobile Systems

- Instead use other methods to free memory if low
  - iOS ***asks*** apps to voluntarily relinquish allocated memory
    - Read-only data thrown out and reloaded from flash if needed
    - Failure to free can result in termination
  - Android terminates apps if low free memory, but first writes **application state** to flash for fast restart
  - Both OSes support paging (will be discussed)

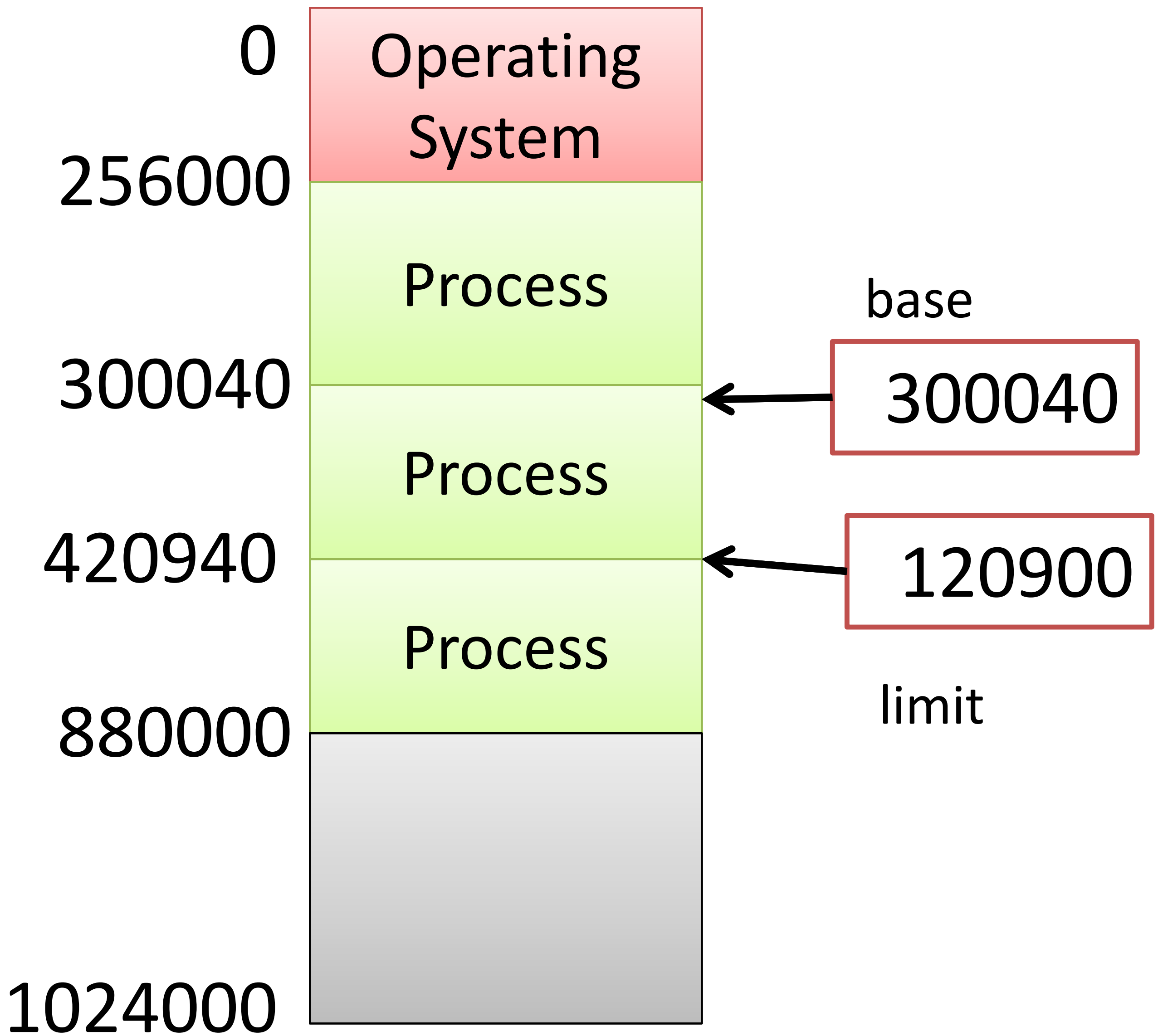
Contiguous Allocation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- 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 contained in single contiguous section of memory

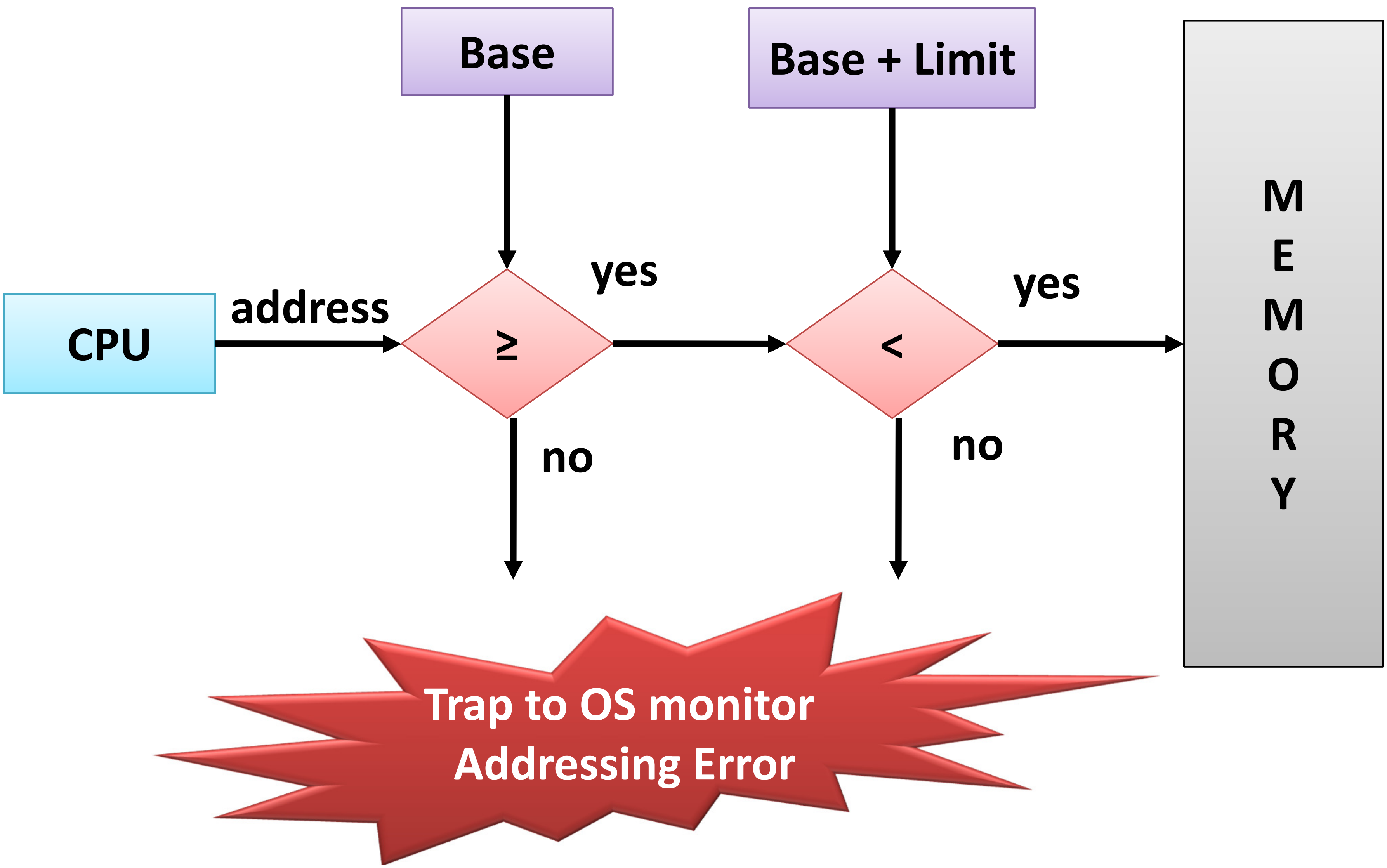
Contiguous Allocation (Cont.)

- Relocation registers used to protect user processes from each other, and from changing OS code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically*

Base and Limit Registers



Hardware Address Protection



Multiple-partition allocation

- Degree of multiprogramming limited by number of partitions
- **Fixed size Partition (Partition Size (PS))**
  - Num process to support is limited if PS is bigger
  - Supporting a big size process is limited if PS is smaller
- **Variable-partition** sizes for efficiency
  - Sized to a given process' needs
- **Hole** – block of available memory
  - Holes of various size are scattered throughout memory

Buddy System

- A compromise between fixed size and variable size allocation
- Memory is allocated that are a power of 2
- Initially a single allocation unit
- A process is allocated a unit memory whose size is the smallest power of 2 larger than the size of process
  - 50K process would be place in 64K space
- If no allocation unit exist of that size, the smallest available allocation larger than the process will be split into two **"Buddy unit"** of ½ size

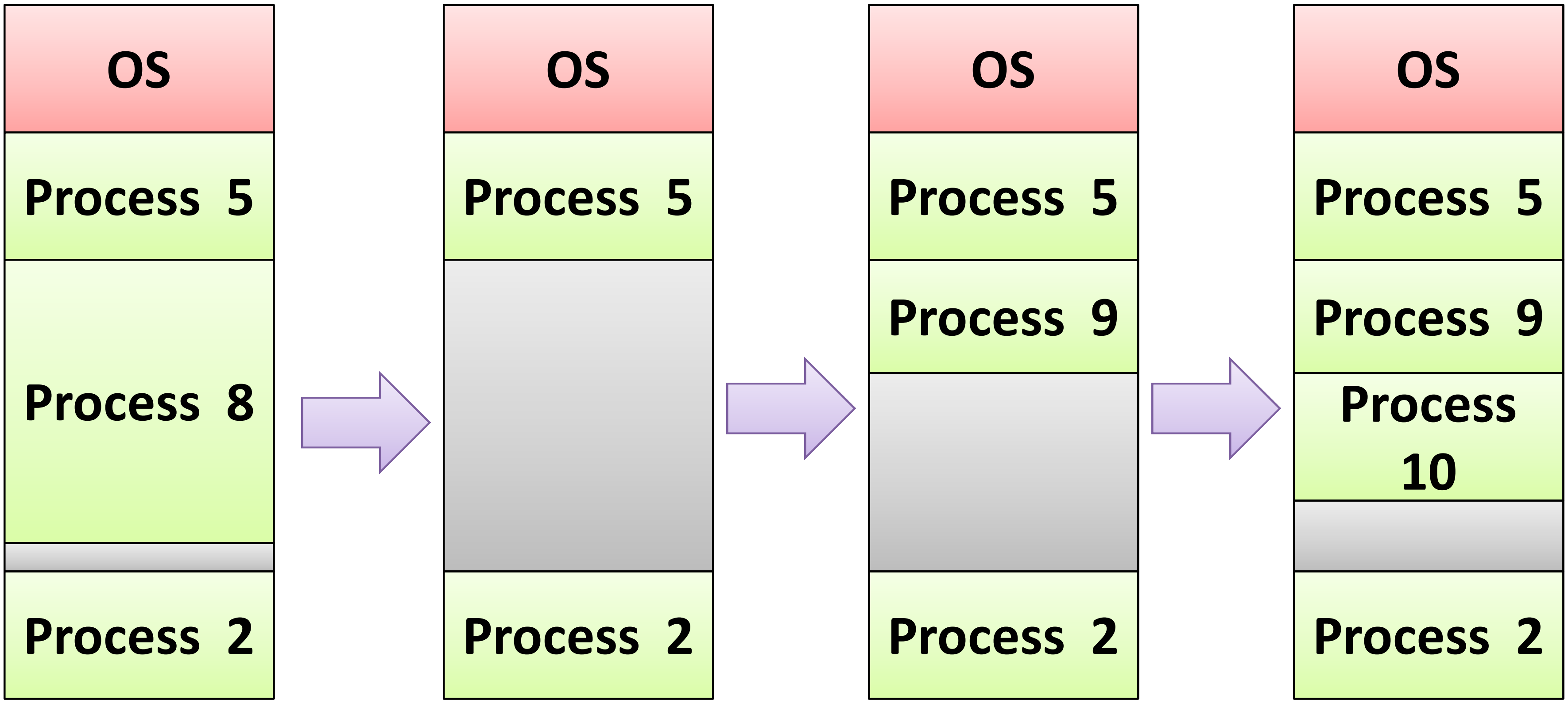
Example

<table>
<tr><th>Action</th><th colspan="6">Memory</th></tr>
<tr><td>Start</td><td colspan="6">1024K</td></tr>
<tr><td>Arrive A (150K)</td><td>A</td><td colspan="3">256K</td><td colspan="2">512K</td></tr>
<tr><td>Arrive B (100K)</td><td>A</td><td>B</td><td colspan="2">128K</td><td colspan="2">512K</td></tr>
<tr><td>Arrive C (50K)</td><td>A</td><td>B</td><td>C</td><td>64 k</td><td colspan="2">512K</td></tr>
<tr><td>Release B</td><td>A</td><td>128K</td><td>C</td><td>64 k</td><td colspan="2">512K</td></tr>
<tr><td>Arrive D (200K)</td><td>A</td><td>128K</td><td>C</td><td>64 k</td><td>D</td><td>256K</td></tr>
<tr><td>Arrive E (60K)</td><td>A</td><td>128K</td><td>C</td><td>E</td><td>D</td><td>256K</td></tr>
<tr><td>Release C</td><td>A</td><td>128K</td><td>64 k</td><td>E</td><td>D</td><td>256K</td></tr>
<tr><td>Releasel A</td><td>256K</td><td>128K</td><td>64 k</td><td>E</td><td>D</td><td>256K</td></tr>
<tr><td>Release E</td><td colspan="4">512K</td><td>D</td><td>256K</td></tr>
<tr><td>Release D</td><td colspan="6">1024K</td></tr>
</table>

Multiple-partition allocation

- When a process arrives
  - OS allocate memory from a hole large enough to accommodate it
- Process exiting frees its partition, adjacent free partitions combined
- OS maintains information about
  - a) allocated partitions
  - b) free partitions (hole)

Multiple-partition allocation



Three Processes Running

Process 8 Finished

Process 9 Arrives

Process 10 Arrives

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 search entire list, **unless ordered by size**
    - Produces the smallest leftover hole
  - **Worst-fit**: Allocate the ***largest*** hole; must also search entire list
    - Produces the largest leftover hole
- First-fit and best-fit better than worst-fit in terms of
  - speed (FF is better as compared to WF) and
  - storage utilization (BF is better as compared to WF)

Fragmentation

- **External Fragmentation** – Total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory
  - This size difference is memory internal to a partition, but not being used

Fragmentation: First Fit

- Analysis reveals that given *N* blocks allocated, 0.5 *N* blocks lost to fragmentation
  - 1/3 may be unusable
  - **50-percent rule**
    - **Once memory is almost full**
    - **Events (processes arrival and departure) are alternate**

Fragmentation (Cont.)

- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers
- Now consider that backing store has same fragmentation problems

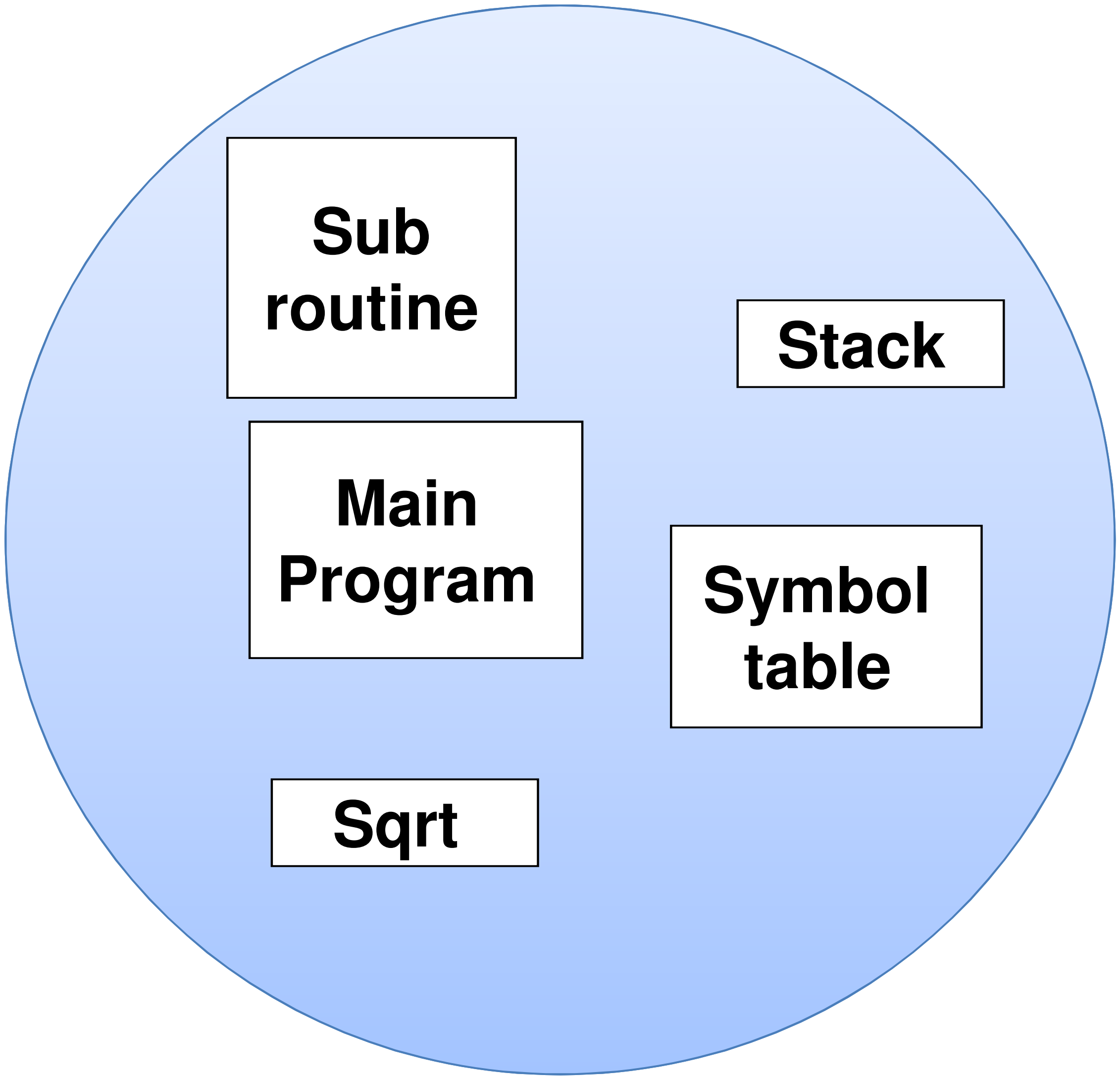
Segmentation

Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of segments
  - A segment is a logical unit such as:

    main program,
    procedure/function/method,
    object, local variables, global variables
    common block, stack, symbol table
    arrays

User's View of a Program

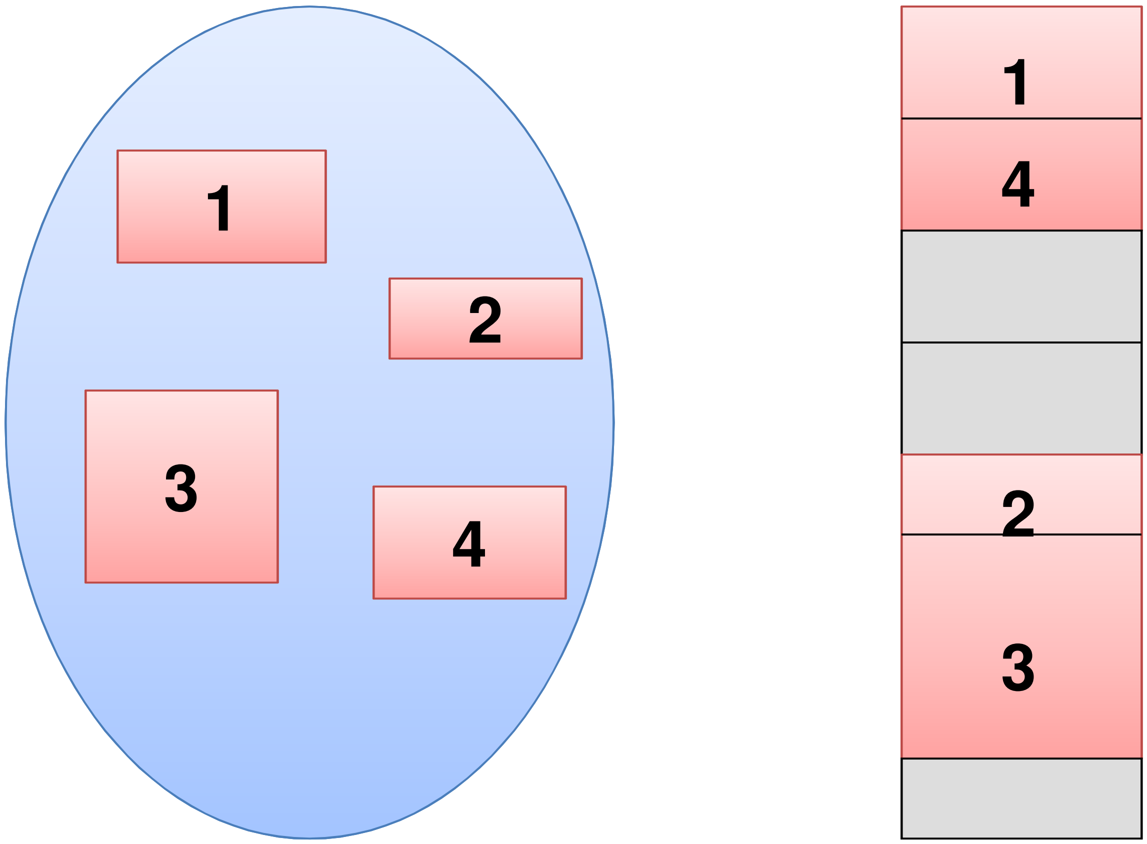


Logical address

Segmentation : Key Rule

- Instead of allocation memory for whole process
- Divide the program into smaller block call segments
  - User view : Already segmented
  - Finer granularity, less fragmentation
    - **Mustard in Bag Vs Brinjal in Bag**
- Each of which is allocated to memory independently
- Segments are variable size

Logical View of Segmentation



user space

physical memory space

Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment

Segmentation Architecture

- **Segment-table base register (STBR)** points to the segment table’s location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

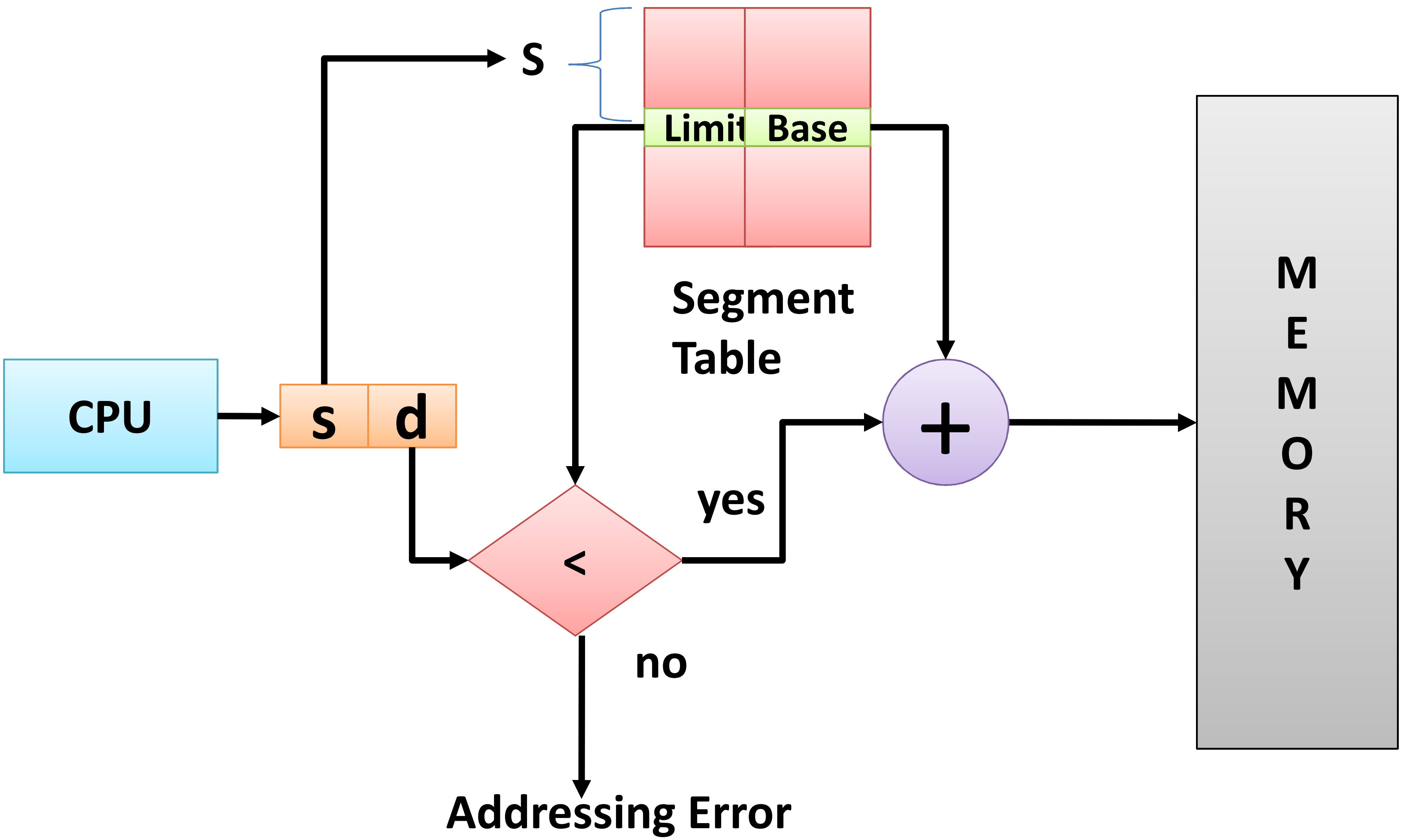
  segment number ***s*** is legal if ***s*** < **STLR**

Every memory access get translated to Two accesses ☹ ☹ ☹

Segmentation Architecture (Cont.)

- Protection
  - With each entry in segment table associate:
    - validation bit = 0 $\Rightarrow$ illegal segment
    - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem

Segmentation Hardware



Segmentation and Paging

- Divide the program into smaller block call segments : User view, already segmented
  - Finer granularity, less fragmentation
    - **Mustard in Bag Vs Brinjal in Bag**
- Divide the program in to smaller but uniform size unit called page
  - A program may contain many pages
  - Last page of program may be partially filled
- **Divide the memory in to smaller size units called Frame**
- Page get mapped to Frame

Thanks