NC STATE UNIVERSITY
Department of Computer Science

CSC501
Operating Systems Principles

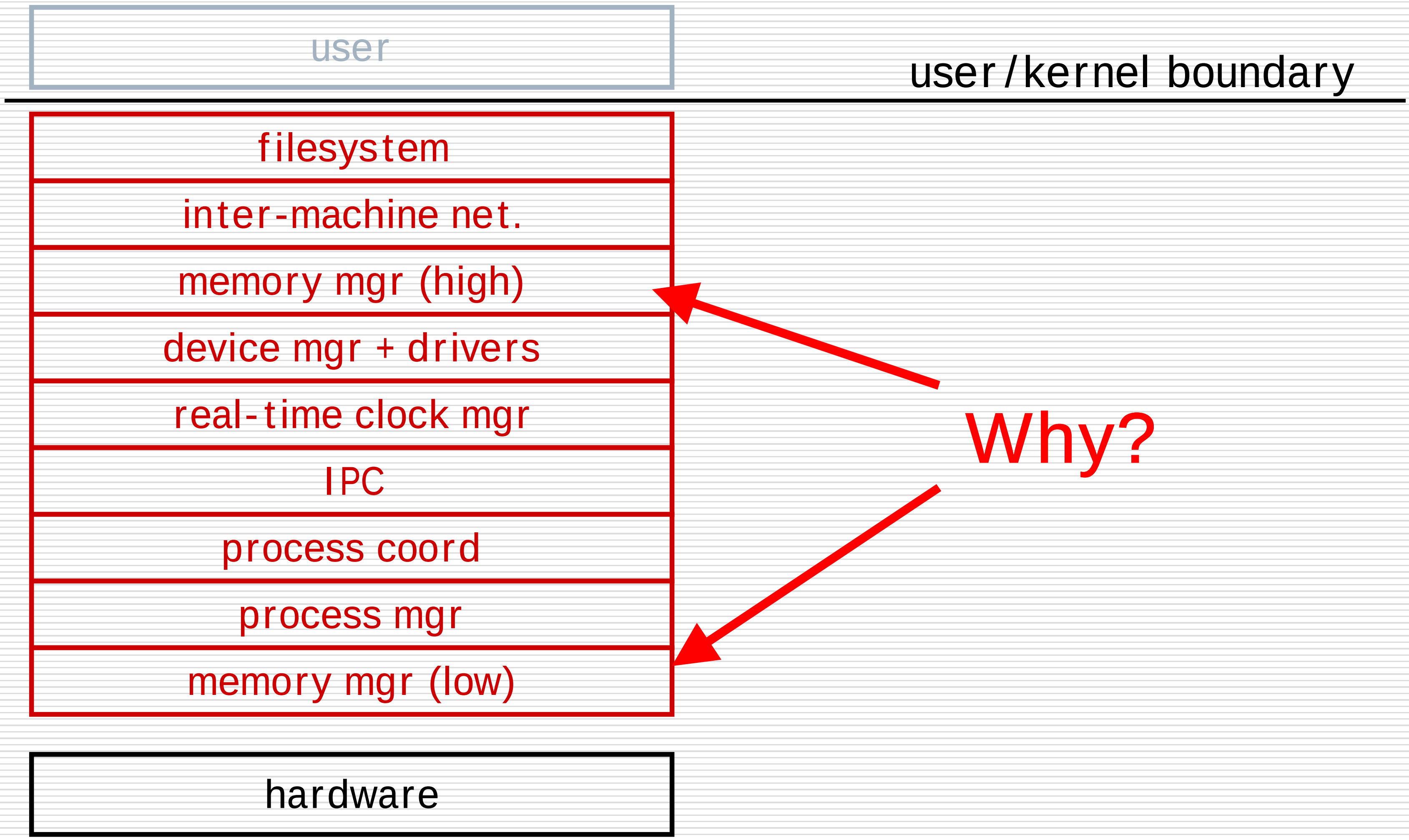
Memory Management

Previous Lectures

- Linker & Loader

- Today
  - Memory Management

Layer



user
user/kernel boundary
filesystem
inter-machine net.
memory mgr (high)
device mgr + drivers
real-time clock mgr
IPC
process coord
process mgr
memory mgr (low)
Why?
hardware

Memory Management

- Low-level memory management
  - Manages memory within kernel address space
  - Allocates address spaces for processes
  - Treats memory as a single, exhaustible resource
  - In layered-OS design, it is positioned in the hierarchy below process manager
- High-level memory management
  - Manages pages within address space
  - Divides memory into abstract resources
  - Positioned in the hierarchy above device manager

Low-level Memory Management

- Conceptual Uses
  - Allocation of process stack
    - *getstk, freestk*
    - Used by OS
  - Allocation of heap storage
    - *getmem, freemem*
    - Used by applications and OS

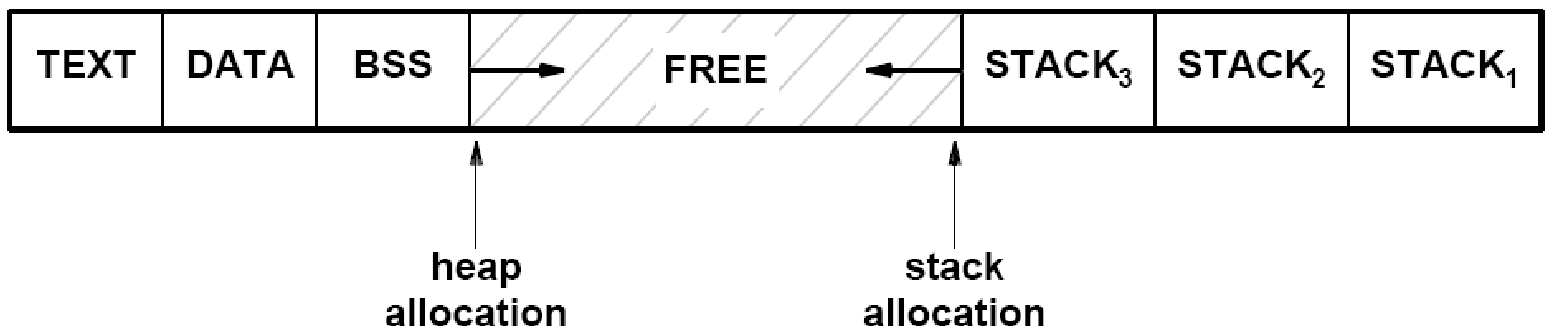
Possible Allocation Strategies

- Single free list
  - First-fit: scan free list and allocate first hole that is large enough
  - Next-fit: start search from end of last allocation
  - Best-fit: find smallest hole that is adequate
  - Worst fit: find largest hole
- Multiple lists
  - By exact size (static / dynamic)
- Hierarchy
  - Binary size allocation
- FIFO cache with above methods

Example Implementation in Xinu

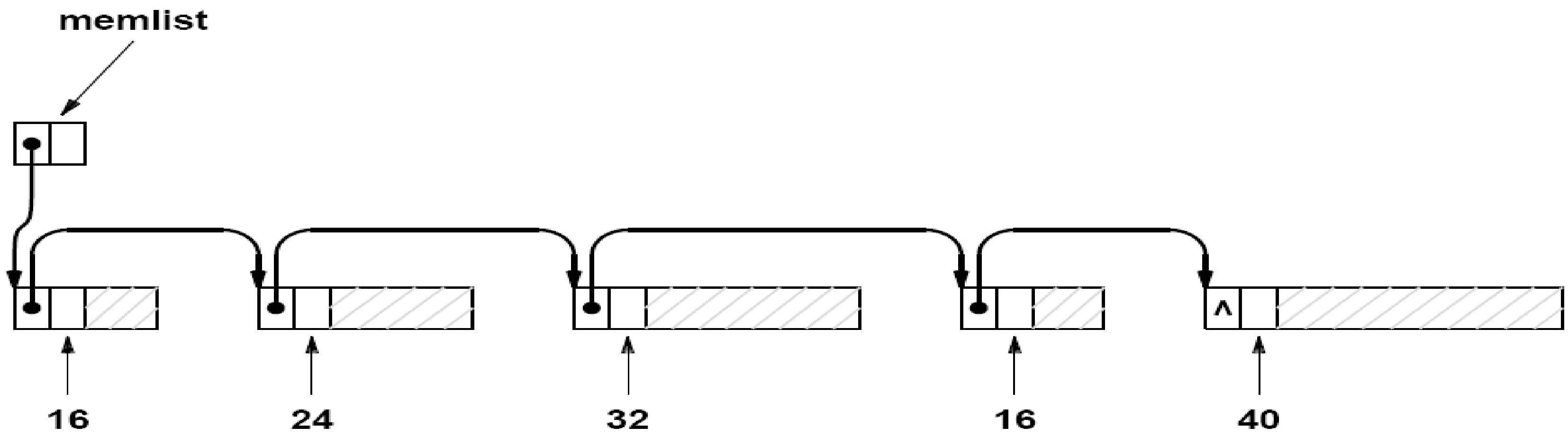
- Single free list
  - Ordered by increasing address
  - Singly-linked
  - Initialized at system startup to all free memory
- Allocation policy
  - Heap: first-fit
  - Stack: last-fit
  - Minimizes fragmentation

Result of Xinu Allocation Policy



- Stack allocation from highest free memory
- Heap allocation from lowest free memory

Illustration of Xinu Free List



- List in order of address
- Related Files: *h/mem.h sys/getmem.c sys/freemem.c*

High-level Memory Management

- Memory is cheap today, and getting cheaper
  - But applications are demanding more and more memory, there is never enough!
- Let's take a look at the history

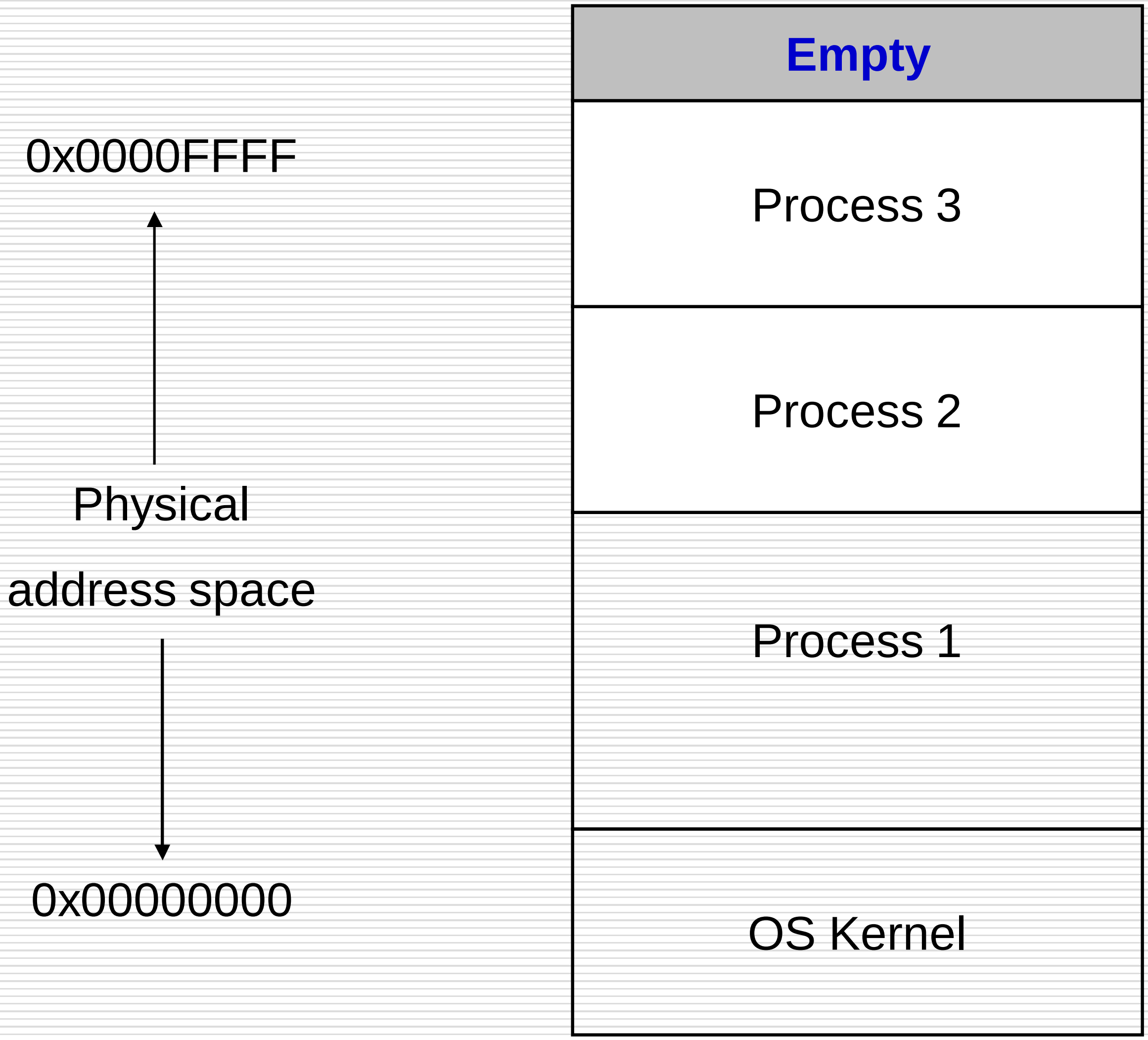
In the Beginning (prehistory)…

- Batch systems
  - One program loaded in physical memory
  - Runs to completion
- If job larger than physical memory, use overlays
  - Identify sections of program that
    - Can run to a result
    - Can fit into the available memory
  - Add statement after result to load a new section
  - Like passes in a compiler

In the Beginning (multi-programming) …

- Multiple processes in physical memory at the same time
  - Allows fast switching to a ready process
  - Divide physical memory into multiple pieces – partitioning
- Partition requirements
  - Protection – keep processes from smashing each other
  - Fast execution – memory accesses can't be slowed by protection mechanisms
  - Fast context switch – can't take forever to setup mapping of addresses

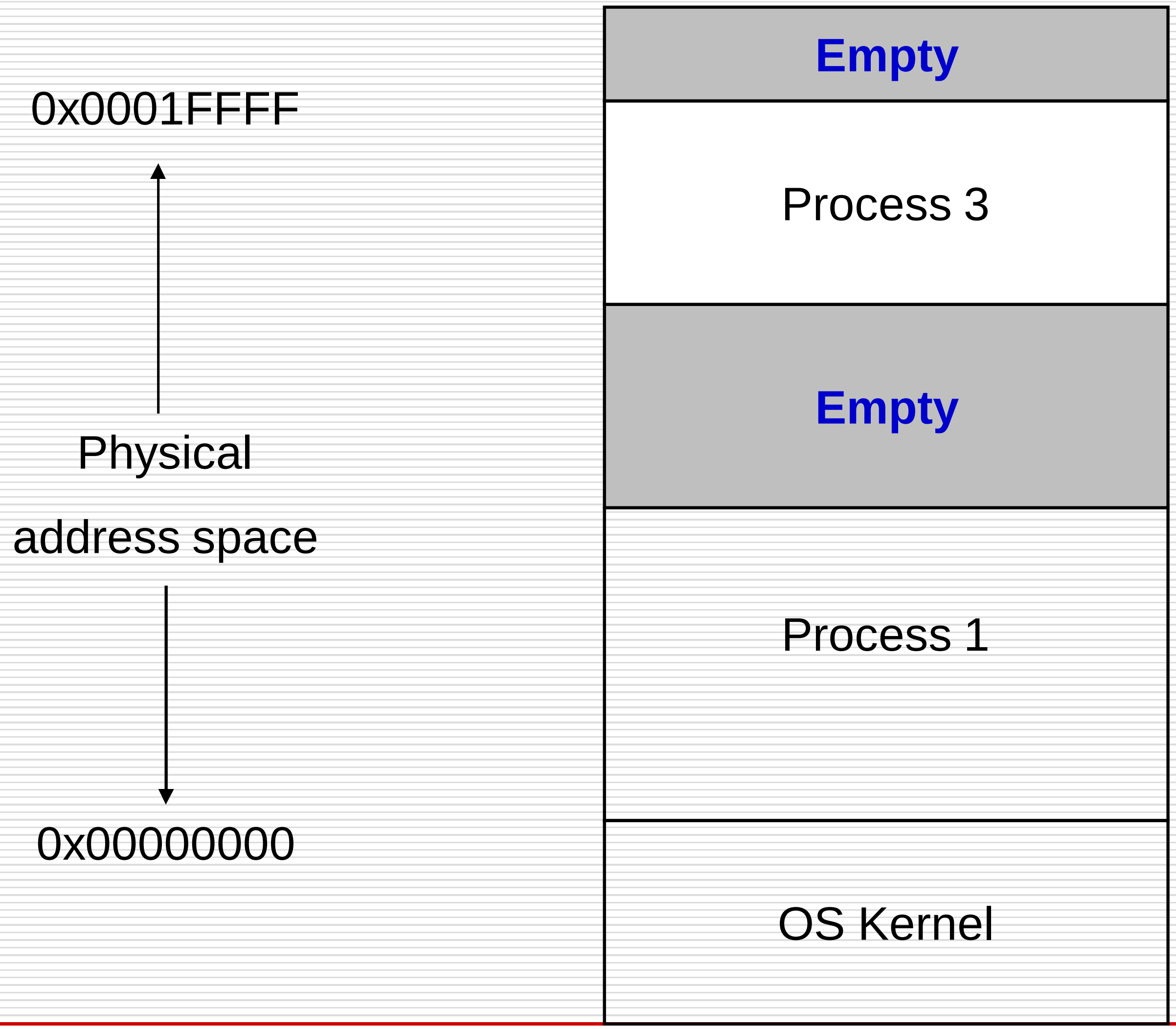
Physical Memory



Loading a Process

- Relocate all addresses relative to start of partition
  - See Linking and Loading
- Memory protection assigned by OS
  - Block-by-block to physical memory
- Once process starts
  - Partition cannot be moved in memory
  - Why?

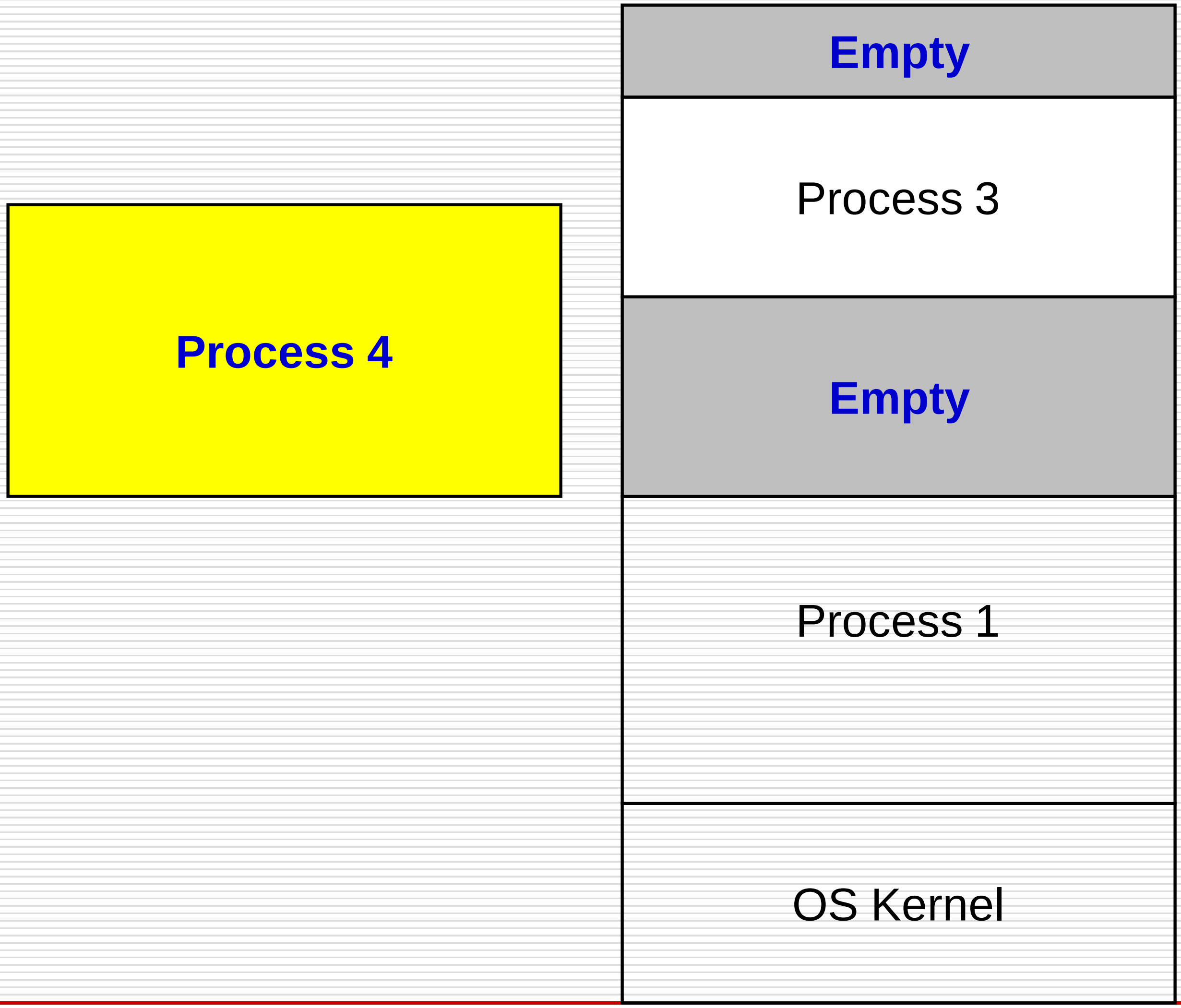
Physical Memory



Problem

- What happens when Process 4 comes along and requires space larger than the largest empty partition?
  - Wait
  - Complex resource allocation problem for OS
  - Potential starvation

Physical Memory



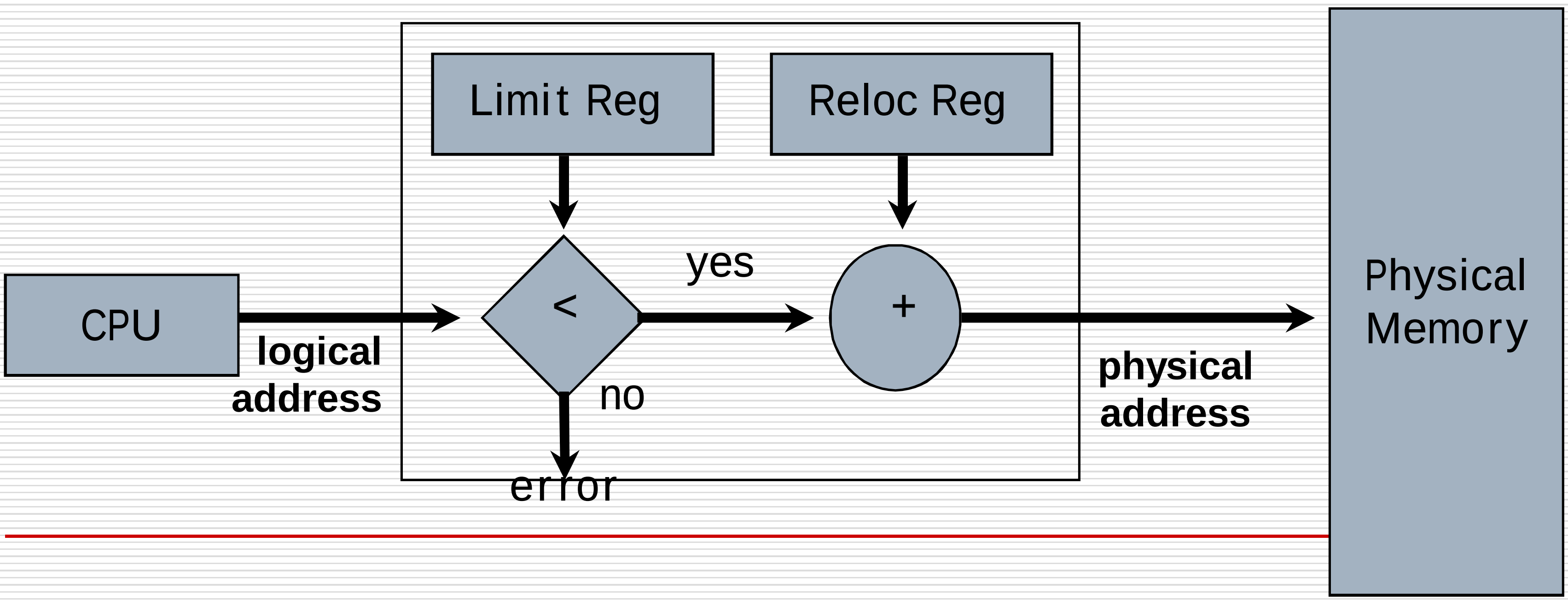
Solution

- **Logical Address**: an address used by the program that is translated by computer into a **physical address** each time it is used

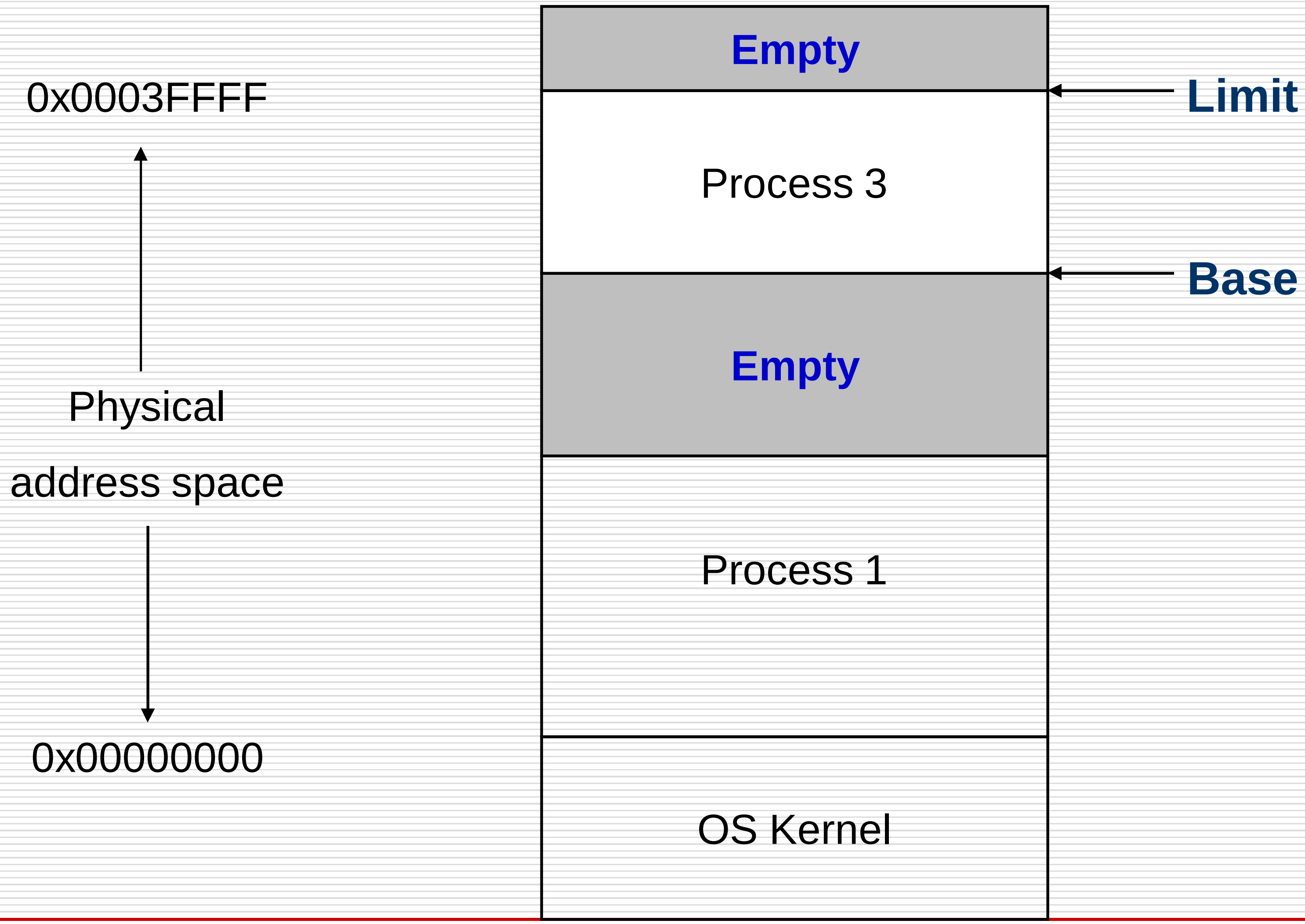
- When the program utters `0x00105C`, …
  - The machine accesses `0x01605C` instead

Simplest Implementation

- Base and Limit registers
  - *Base* added to all addresses
  - *Limit* checked on all memory references
- Loaded by OS at each context switch



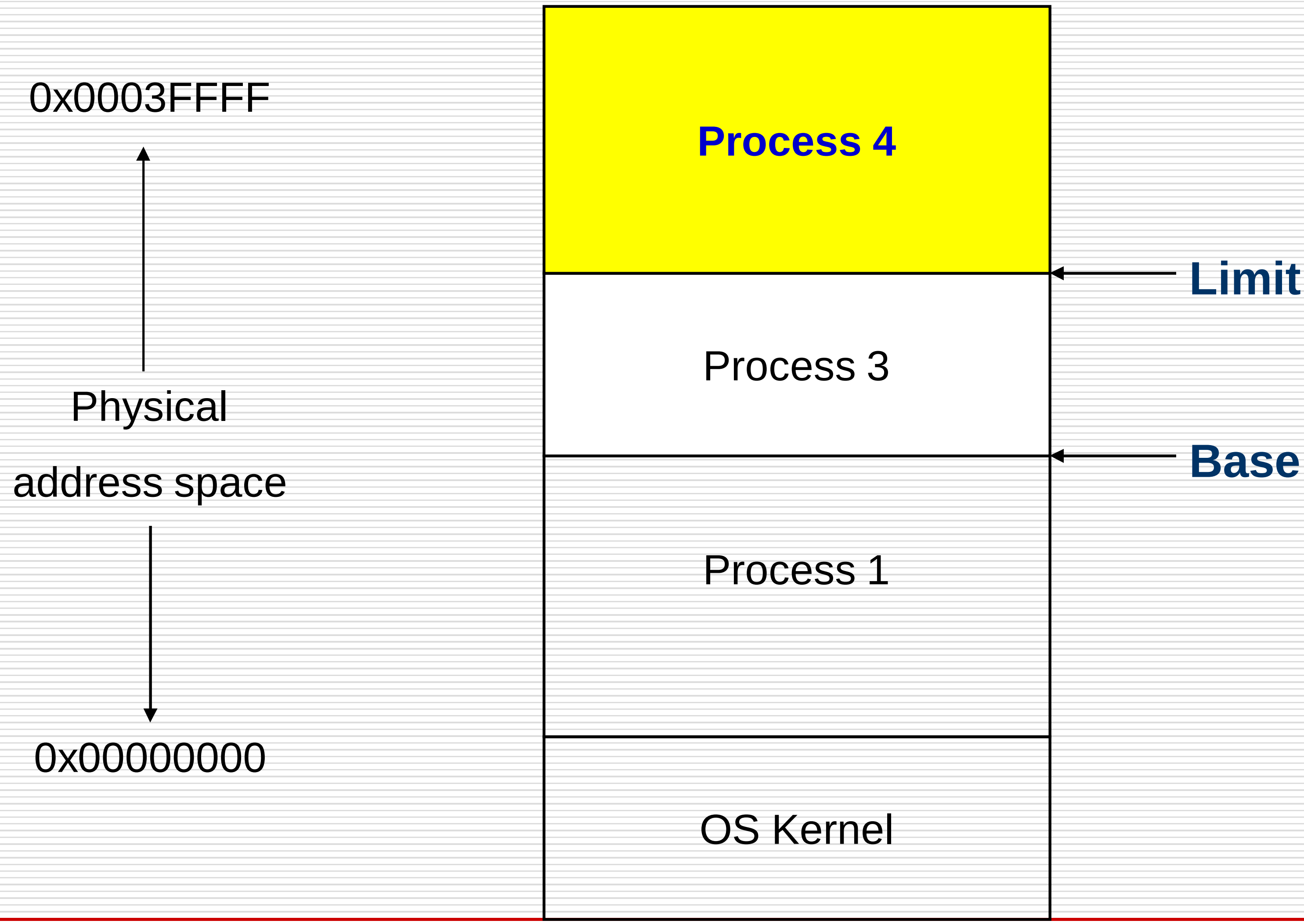
Physical Memory



Advantages

- No relocation of program addresses at load time
  - All addresses relative to zero
- Built-in protection provided by *Limit*
  - No physical protection per page or block
- Fast execution
  - Addition and limit check at hardware speeds within each instruction
- Fast context switch
  - Need only change base and limit registers
- Partition can be suspended and moved at any time
  - Process is unaware of change
  - Expensive for large processes!

Physical Memory



Challenge – Memory Allocation

- Fixed partitions
- Variable partitions

Partitioning Strategies – Fixed

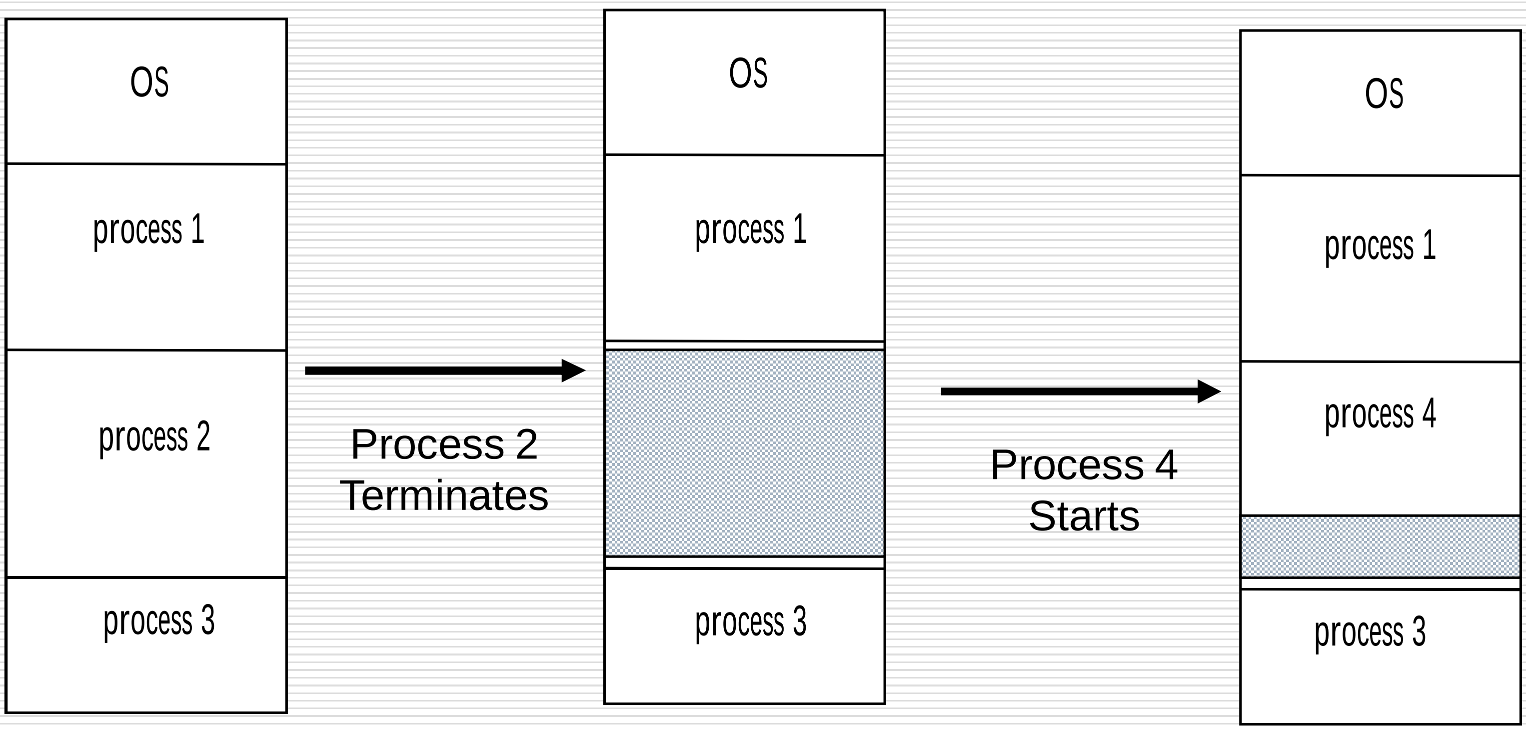
- Fixed Partitions – divide memory into equal sized pieces (except for OS)
  - Degree of multiprogramming = number of partitions
  - Simple policy to implement
    - All processes must fit into partition space
    - Find any free partition and load the process
- Problem – Internal Fragmentation
  - Unused memory in a partition that is not available to other processes

Question:
What is the “right” partition size?

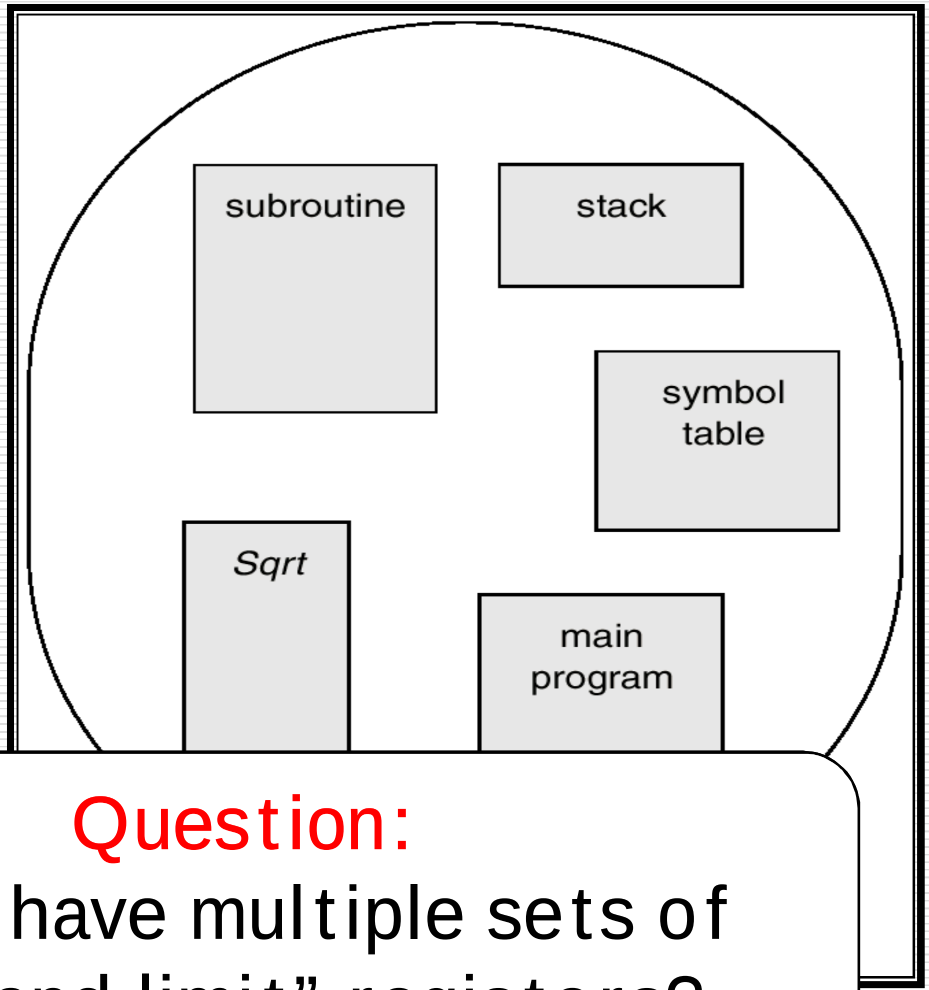
Partitioning Strategies – Variable

- Memory is dynamically divided into partitions based on process needs
  - More complex management problem
    - Need data structures to do tracking of free and used memory
    - New process is allocated memory from hole large enough to fit it
- Problem – External Fragmentation
  - Unused memory between partitions that is not too small to be used by any processes

Partitioning Strategies – Variable



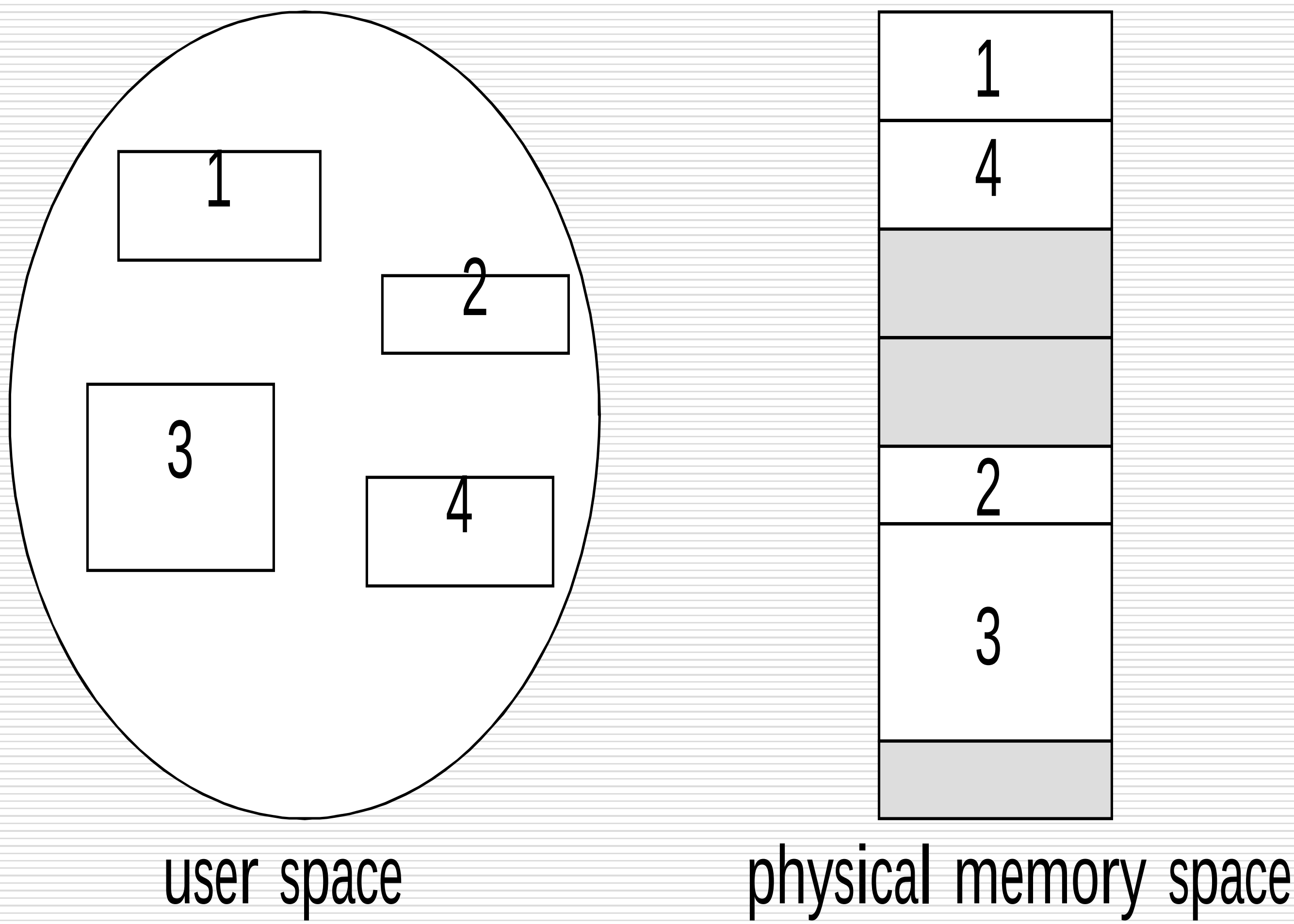
User's View of a Program



Question:

Can we have multiple sets of "base and limit" registers?

Logical View of Segmentation



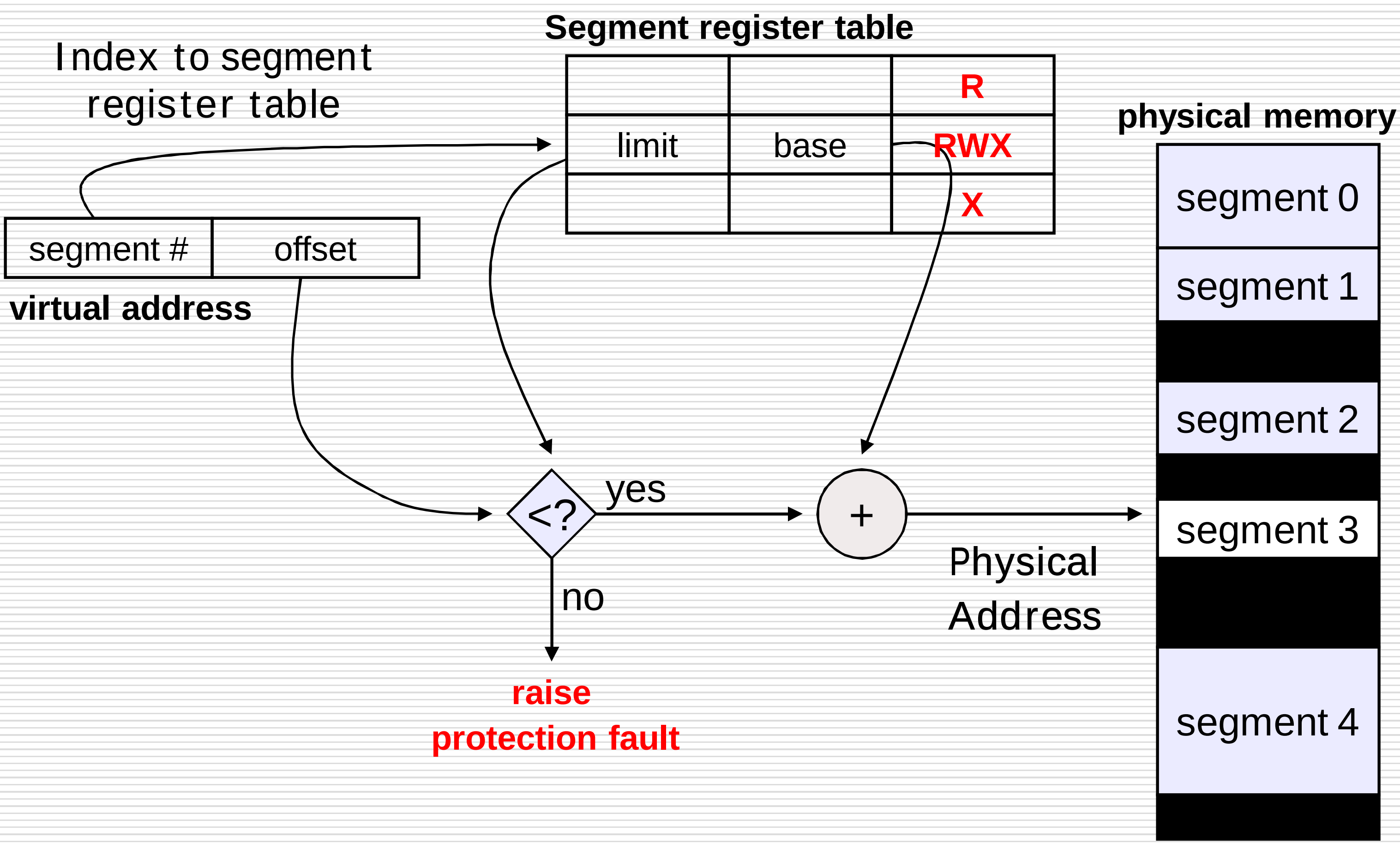
Segmentation

- Logical address consists of a pair:

  $\langle$segment-number, offset$\rangle$

- Segment table where each entry has:
  - Base: contains the starting physical address where the segments reside in memory.
  - Limit: specifies the length of the segment.

Segment Lookup



Segmentation

- Common in early minicomputers
  - Small amount of additional hardware – 4 or 8 segments
  - Used effectively in Unix
- Good idea that has persisted and supported in current hardware and OSs
  - X86 supports segments
  - Linux supports segments

Question:

Do we still have external fragmentation problem? If yes, can we further improve it?

Next Lecture

- Paging

Lab3 Out !