Introduction to OS

Memory Management

MOS 3.1 – 3.3

Mahmoud El-Gayyar
elgayyar@ci.suez.edu.eg

In the Beginning (prehistory)…

- Single usage (or batch processing) systems
  - One program loaded in physical memory at a time
  - 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 commands after result to load a new section

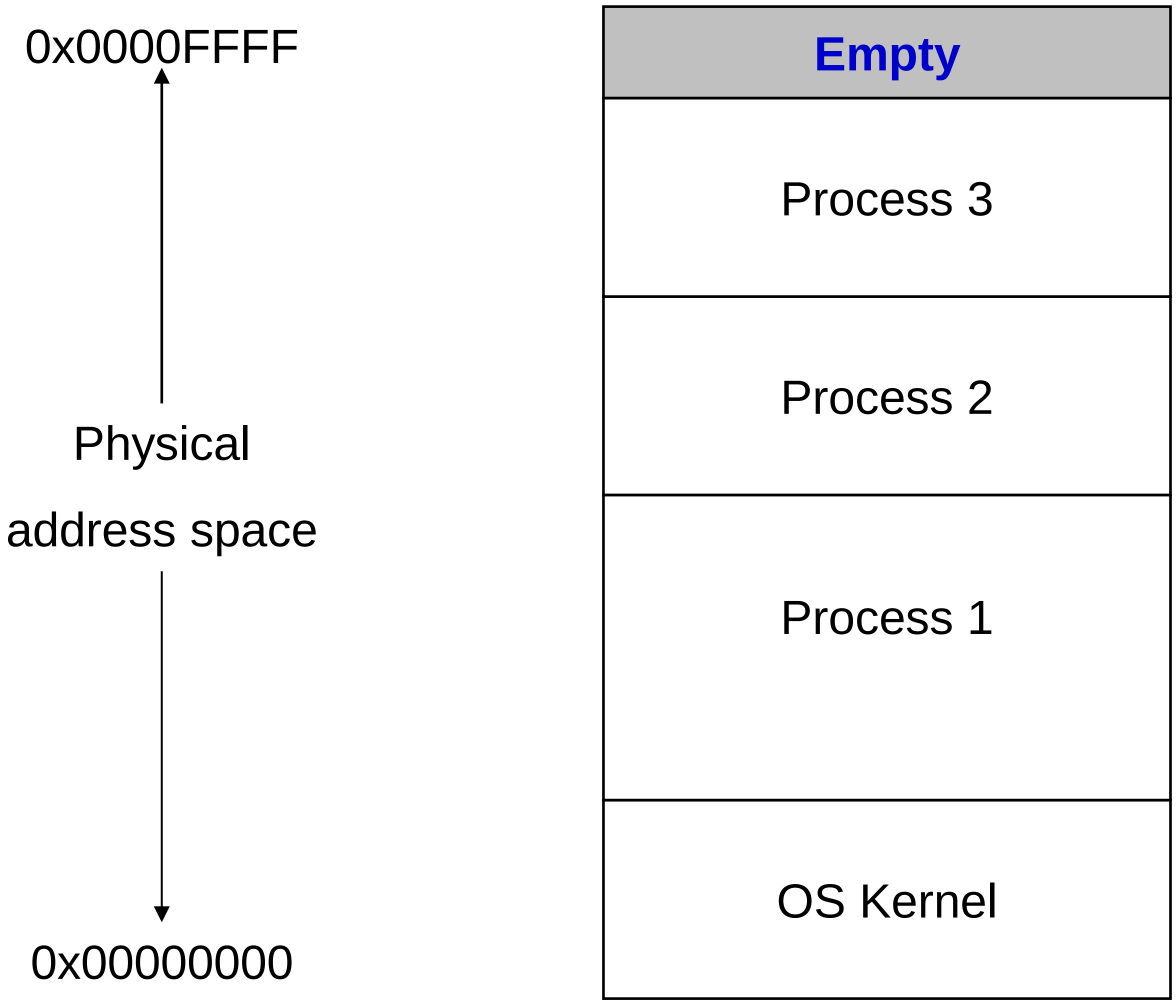
Still near the Beginning (multi-tasking) …

- Multiple processes in physical memory at the same time
  - allows fast switching to a ready process
  - *Partition* physical memory into multiple pieces
    - One partition for each program

- 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

- No **Memory Abstraction**

Physical Memory



MOV REGISTERS,1000

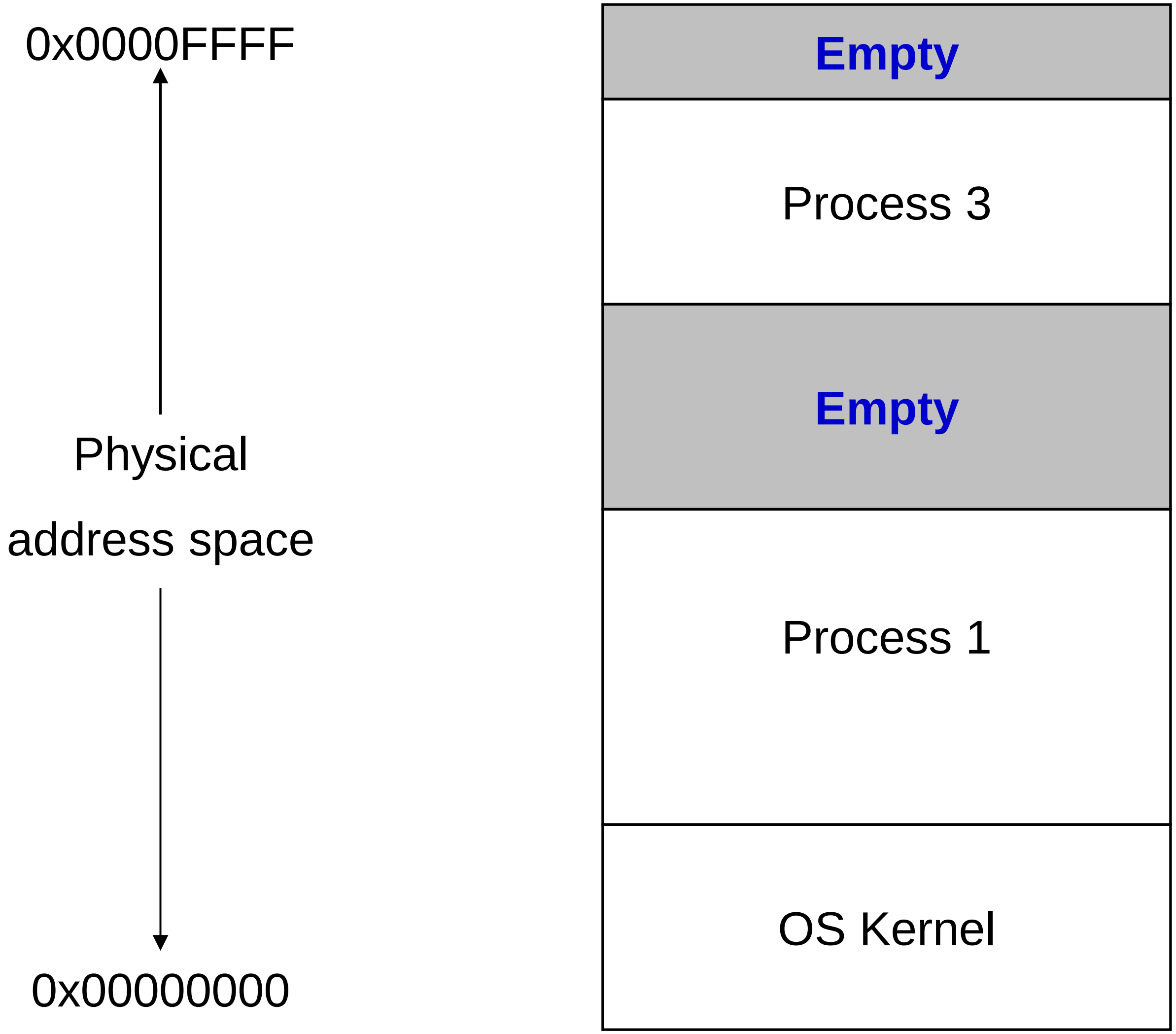
Loading a Process

- Relocate all addresses relative to start of partition (*which is address and which is not*)

- Memory protection assigned by OS
  - Block-by-block to physical memory (protection bits)

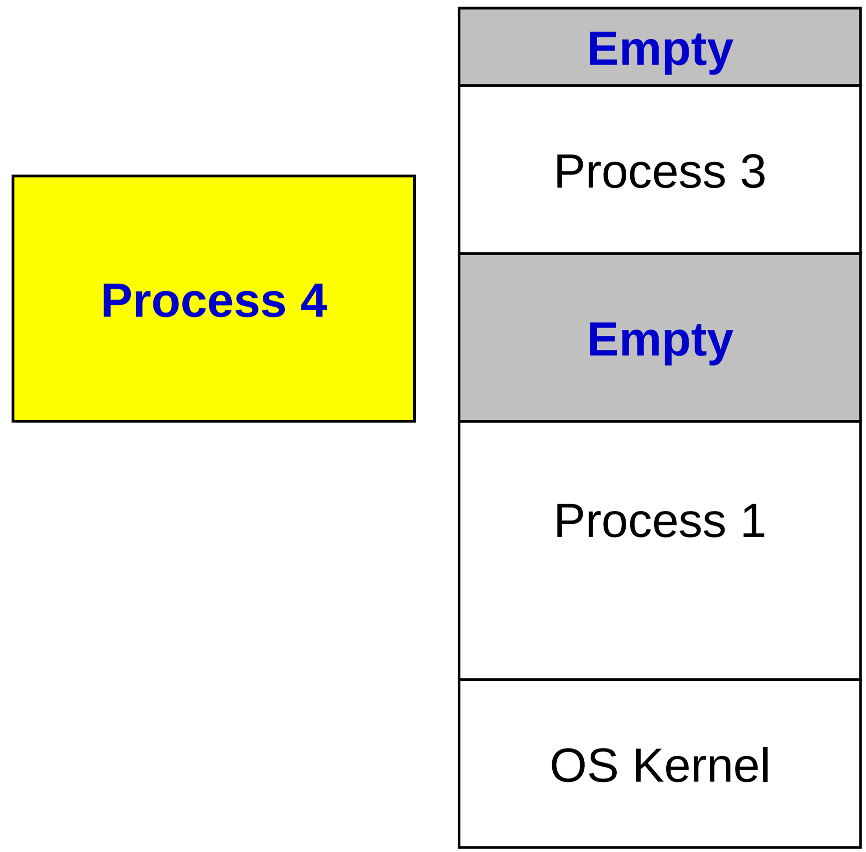
- Once process starts
  - Partition cannot be moved in memory
  - *Why?*

Physical Memory – Process 2 terminates



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



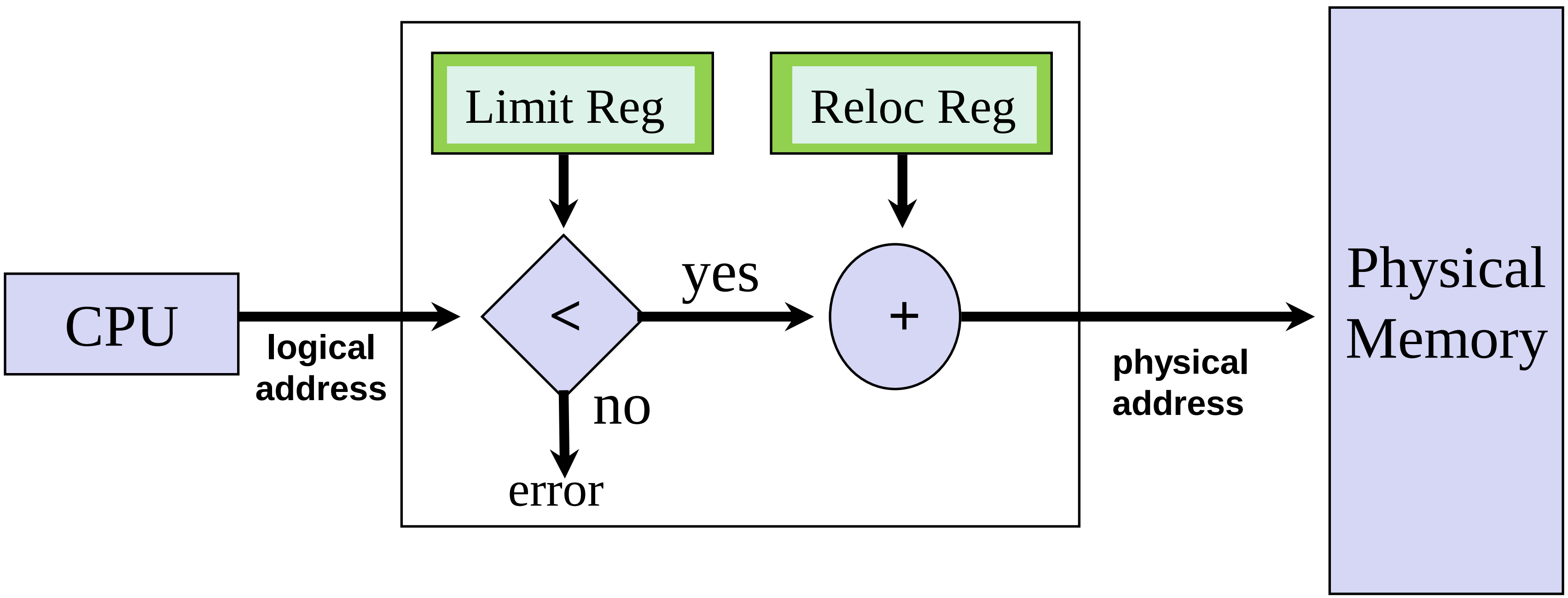
Solution

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

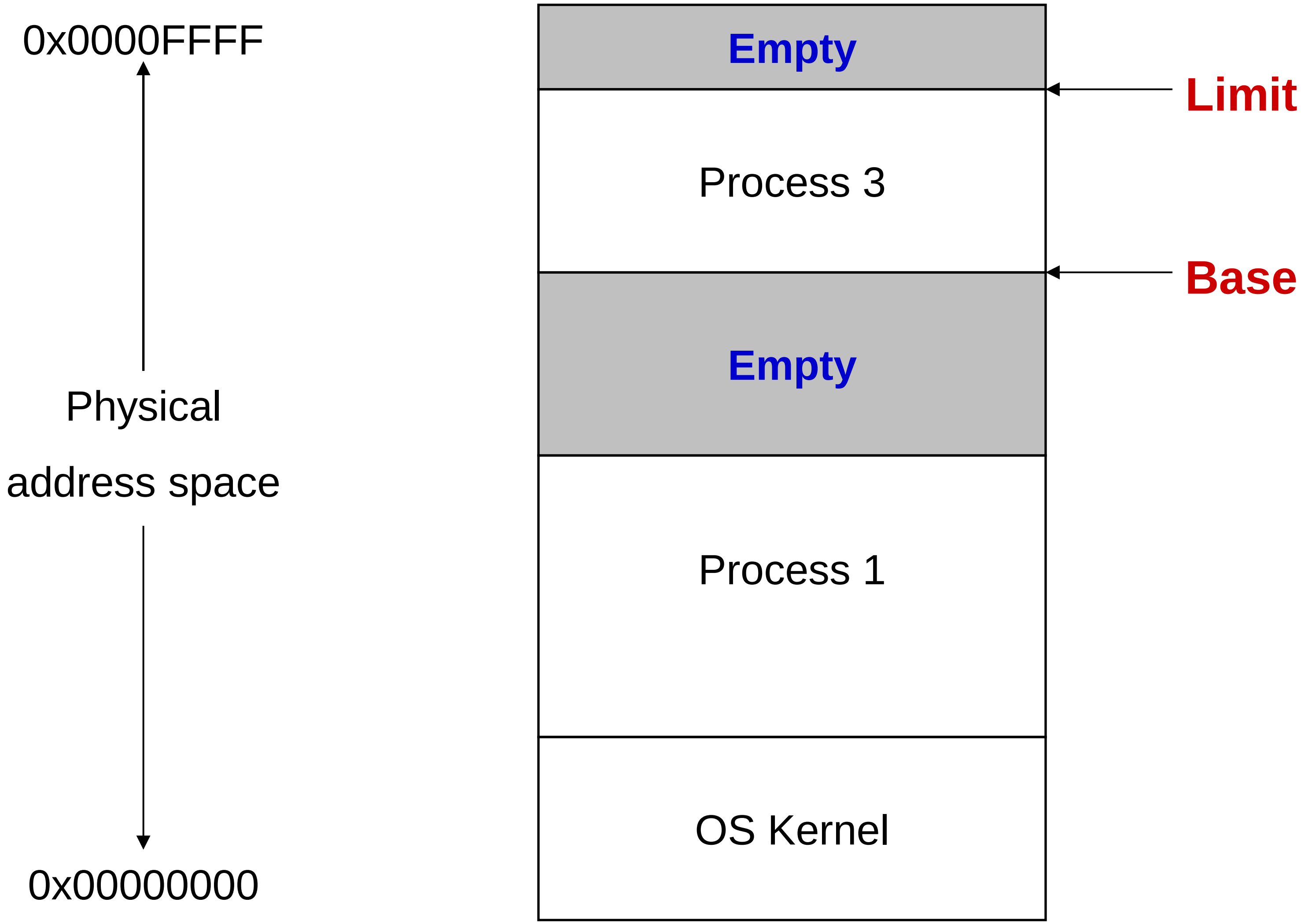
- When the program uses **`0x00105C`**, …
    - … the machine accesses **`0x01605C`**

Implementation

- *Base* and *Limit* registers
  - *Base* is automatically added to all addresses
  - *Limit* (process length) is checked on all memory references
  - Introduced in minicomputers of early 1970s
- ***Loaded by OS at each context switch***



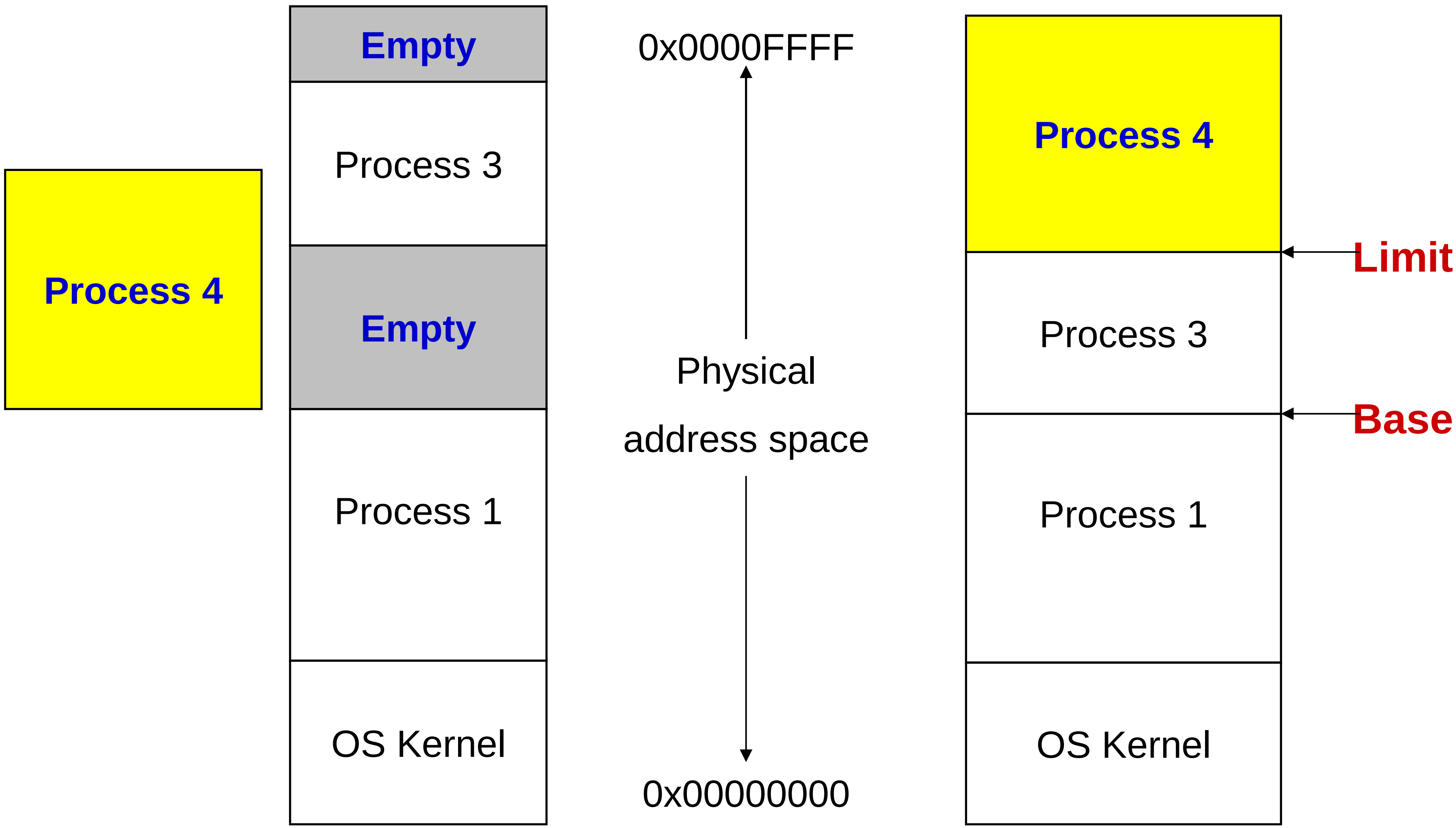
Physical Memory



Advantages

- No relocation of program addresses at load time
  - All addresses relative to zero!
- Built-in protection provided by *Limit*
  - ***No physical protection*** per block
- Fast context switch
  - Need only change base and limit registers
- Partition can be suspended and moved at any time
  - Process is unaware of change
  - Potentially expensive for large processes due to copy costs!

Physical Memory



Definition

- ***Virtual Address Space:***
  - The address space in which a process or thread "thinks"
  - Address space with respect to which pointers, code & data addresses, etc., are interpreted
  - Separate and independent of *physical address space* where things are actually stored

Challenge – Memory Allocation

- How to allocate space for different processes?
  - Fixed partitions
  - Variable partitions

Partitioning Strategies – **Fixed**

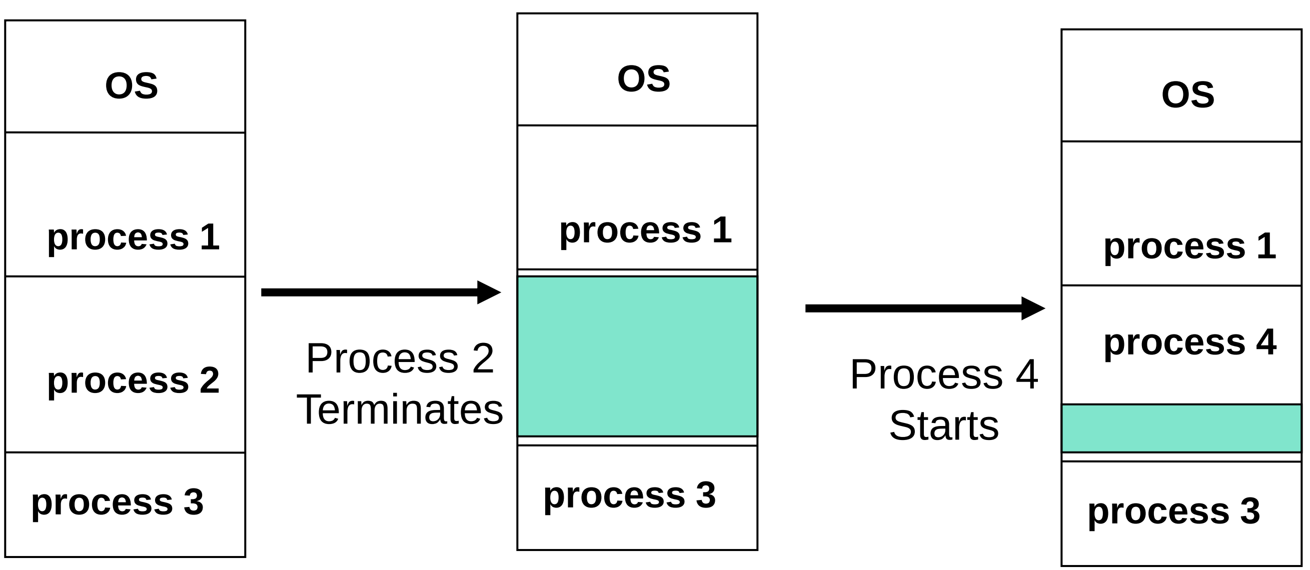
- Fixed Partitions – divide memory into equal sized pieces (except for OS)
  - Degree of multiprogramming = number of partitions
  - Simple policy to implement
    - All processes must fit into partition space
    - Find any free partition and load the process
- Problem – what is the "right" partition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

Partitioning Strategies – **Variable**

- Idea: remove "wasted" memory that is not needed in each partition
  - Eliminating *internal fragmentation*
- Memory is dynamically divided into partitions based on process needs
- Definition:
  - ***Hole:*** a block of free or available memory
  - Holes are scattered throughout physical memory
- New process is allocated memory from hole large enough to fit it

Variable Partitions

- More complex management problem
  - Must track free and used memory
  - Need data structures to do tracking
  - What holes are used for a process?
- ***External Fragmentation***
  - memory that is <u>outside any partition</u> and is too small to be usable by any process



Definitions – *Fragmentation*

- Internal fragmentation
  - Unused or unneeded space *within* an allocated part of memory.
  - Cannot be allocated to another task/job/process
- External fragmentation
  - Unused space *between* allocations.
  - Too small to be used by other requests
- Applies to all forms of *spatial* resource allocation
  - RAM
  - Disk
  - Virtual memory within process
  - …

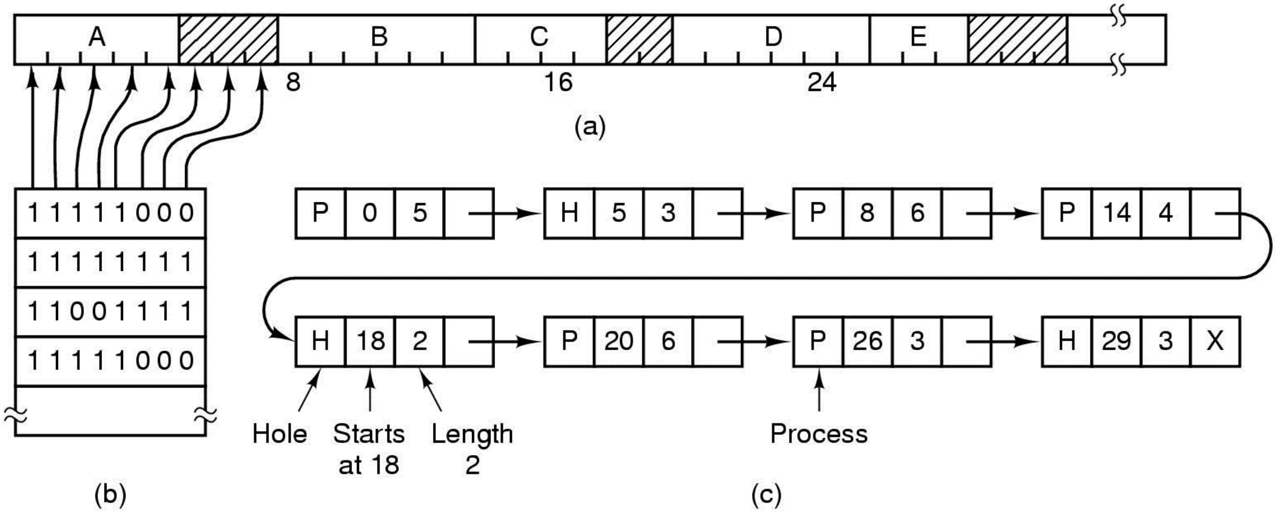
Memory Allocation – Mechanism

- MM system maintains data about free and allocated memory alternatives
  - *Bit maps* – 1 bit per "allocation unit"
  - *Linked Lists* – free list updated and coalesced when not allocated to a process

- At swap-in or process create
  - Find free memory that is large enough to hold the process
  - Allocate part (or all) of memory to process and mark remainder as free

- ***Compaction***
  - Moving things around so that *holes* can be consolidated
  - Expensive in OS time

Memory Management – List *vs.* Map



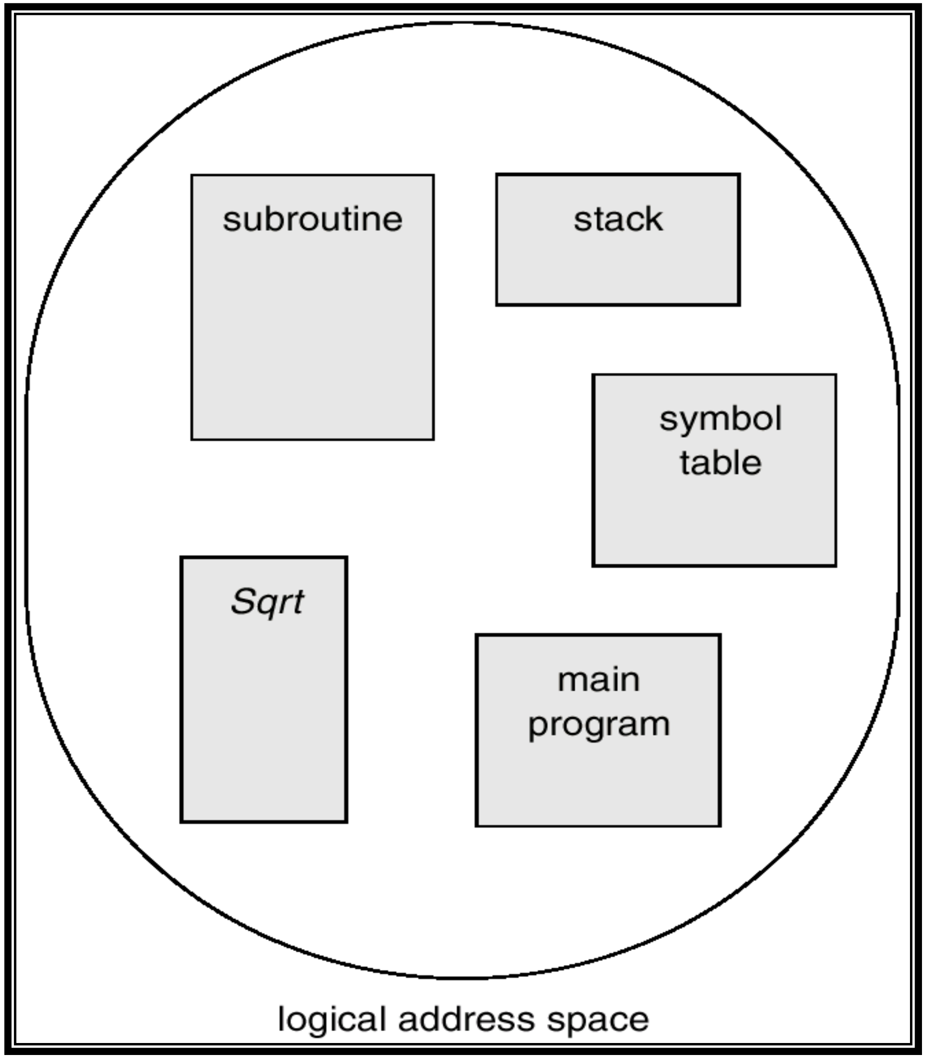
- Part of memory with 5 processes, 3 holes
  - tick marks show allocation units
  - shaded regions are free
- Corresponding bit map
- Same information as a list

Memory Allocation – Policies

- Policy examples
  - ***First Fit:*** scan free list and allocate first hole that is large enough – fast
  - ***Next Fit:*** start search from end of last allocation
  - ***Best Fit:*** find smallest hole that is adequate
    - slower and lots of fragmentation
  - ***Worst fit:*** find largest hole

  - In general, *First Fit* is the winner

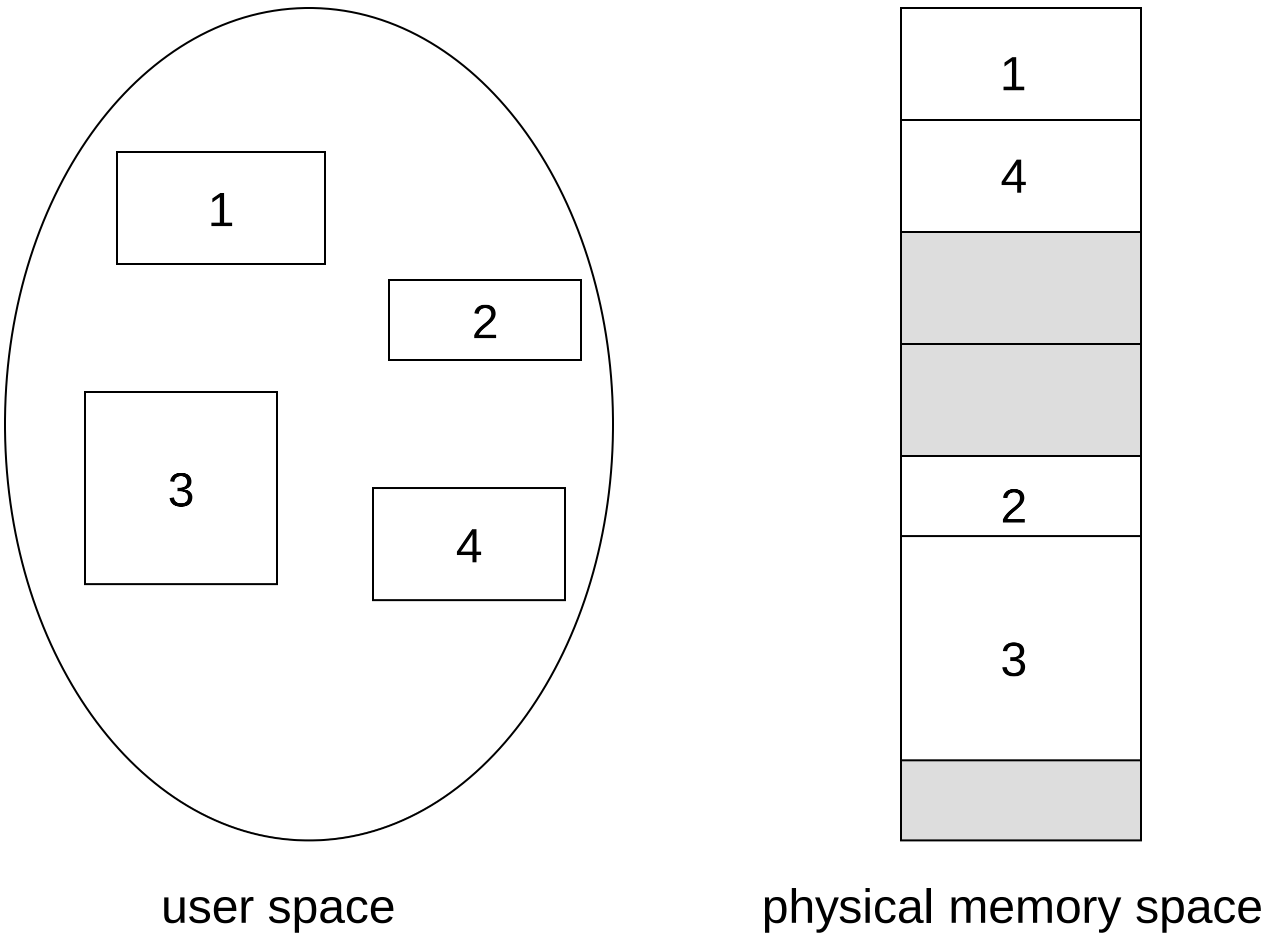
User's View of a Program



Memory Management – beyond Partitions

- Can we improve memory utilization & performance
  - Processes have distinct parts
    - *Code* – program and maybe shared libraries
    - *Data* – pre-allocated and heap
    - *Stack*
  - Solution – slightly more Memory Management hardware
    - ***Multiple sets of "base and limit"*** registers
    - Divide process into logical pieces called ***segments***
- Advantages of *segments*
  - Stack and heap can be grown – may require segment swap
  - With separate I and D spaces can have larger virtual address spaces
    - "I" = *Instruction* (i.e., code, always read-only)
    - "D" = *Data* (usually read-write)

Logical View of Segmentation



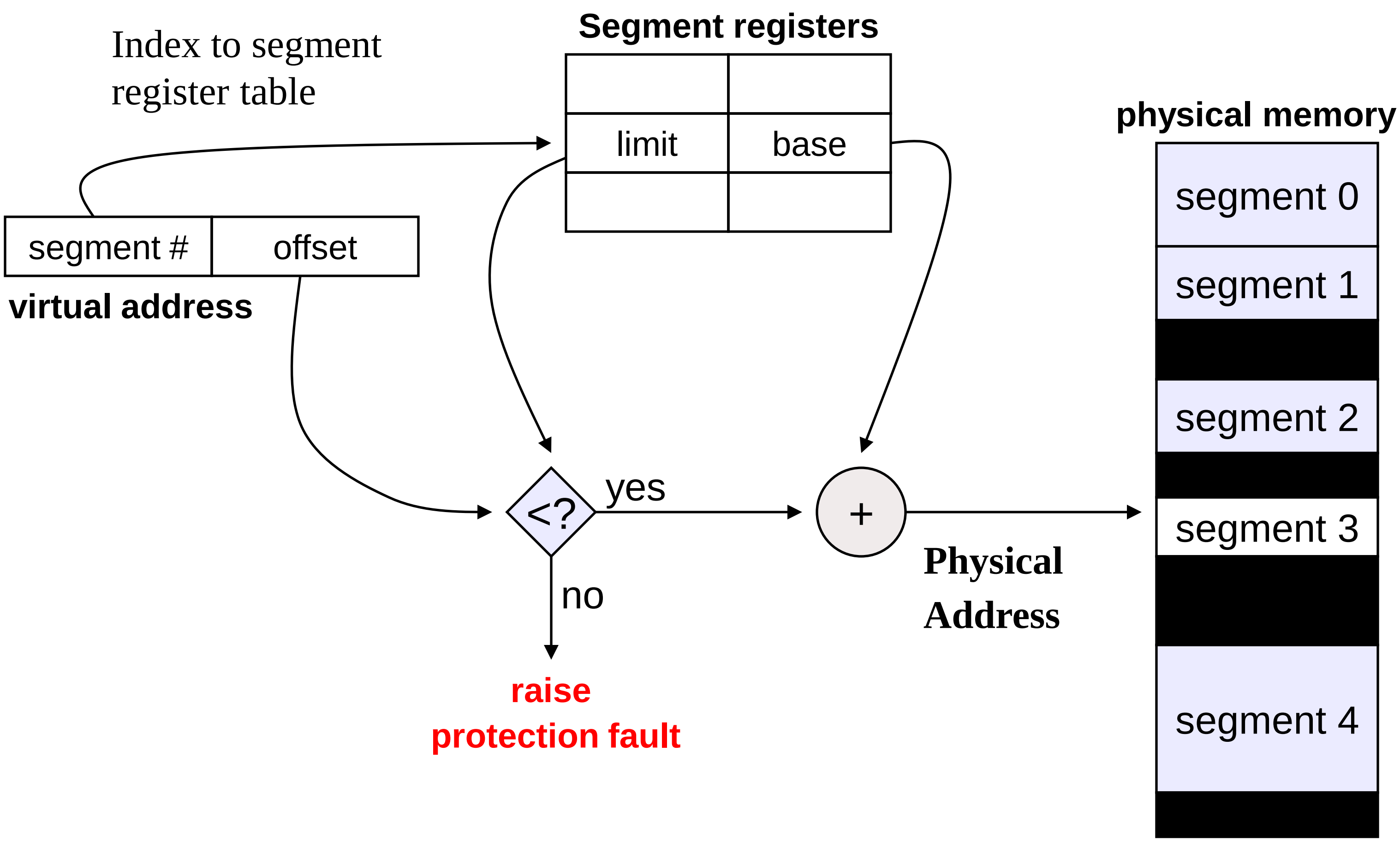
Segmentation

- Logical address consists of a pair:

  <`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.

Segment Lookup



Segmentation

- *Protection*. With each pair of segment registers, include:
  – *validation bit* = 0 ⇒ illegal segment
  – *read*/ *write*/ *execute* privileges

- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
    - With all the problems of fragmentation!

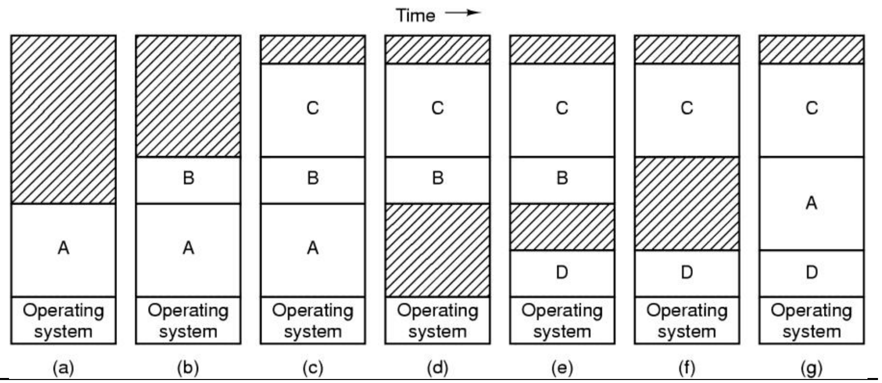
Do we have enough memory?!

- Can’t keep all processes in main memory
  - Too many (hundreds)
  - Too big (e.g. 200 MB program)

- Two approaches
  - **Swap**: bring program in and run it for awhile
  - **Virtual memory**: allow program to run even if only part of it is in main memory

Swapping and Scheduling

- *Swapping*
  - Move process from memory to disk (swap space)
    - Process is blocked or suspended
  - Move process from swap space to big enough partition
    - Process is ready
    - Set up Base and Limit registers
  - Memory Manager (MM) and Process scheduler work together
    - Scheduler keeps track of all processes
    - MM keeps track of memory
    - Scheduler marks processes as swap-able and notifies MM to swap in processes
    - Scheduler policy must account for swapping overhead
    - MM policy must account for need to have memory space for ready processes

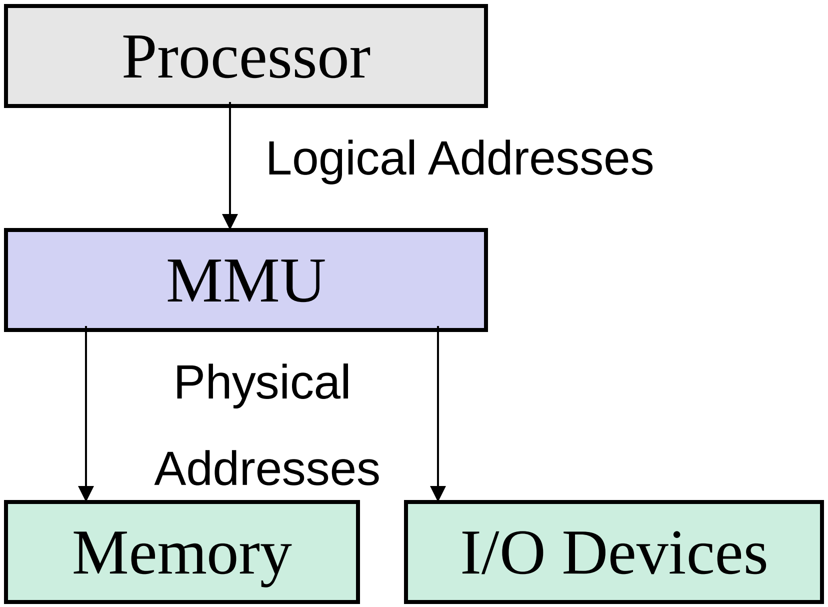


Paging

- A different approach
- A program can run while only a small part of it is loaded into memory

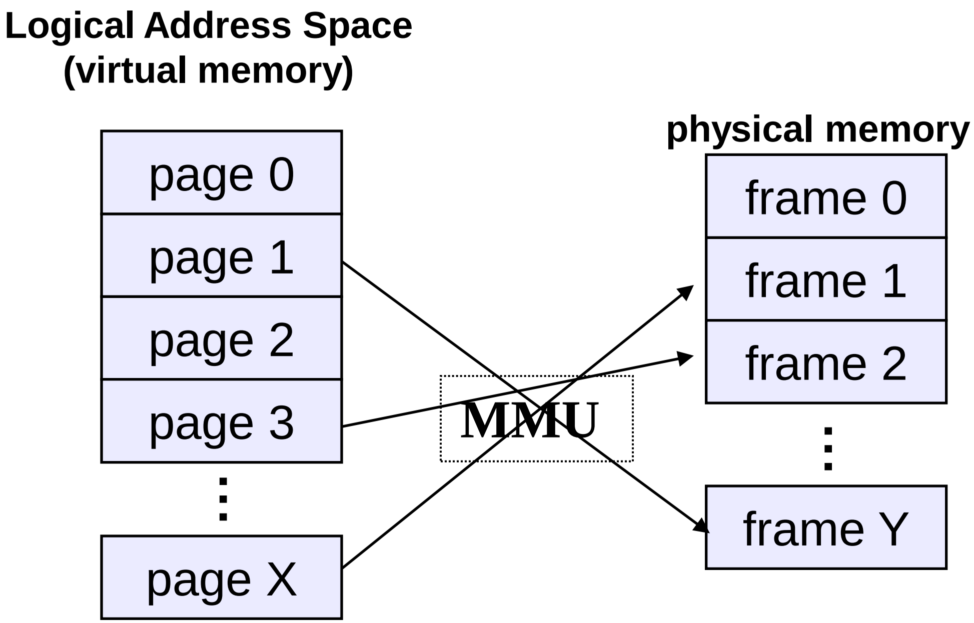
Virtual Memory Management

- ***Memory Management Unit*** (MMU)
  - Set of registers and mechanisms to translate *virtual* addresses to *physical* addresses
- Processes (and processors) see virtual addresses
  - Virtual address space is same for all processes, usually 0 based
  - Virtual address spaces are protected from other processes
- MMU and devices see physical addresses



Paging

- Use small ***fixed size units*** in both physical and virtual memory
- Provide ***MMU*** *hardware* to allow page units to be scattered across memory
- Make it possible to leave *infrequently used parts* of virtual address space out of physical memory
- Solve internal & external *fragmentation* problems



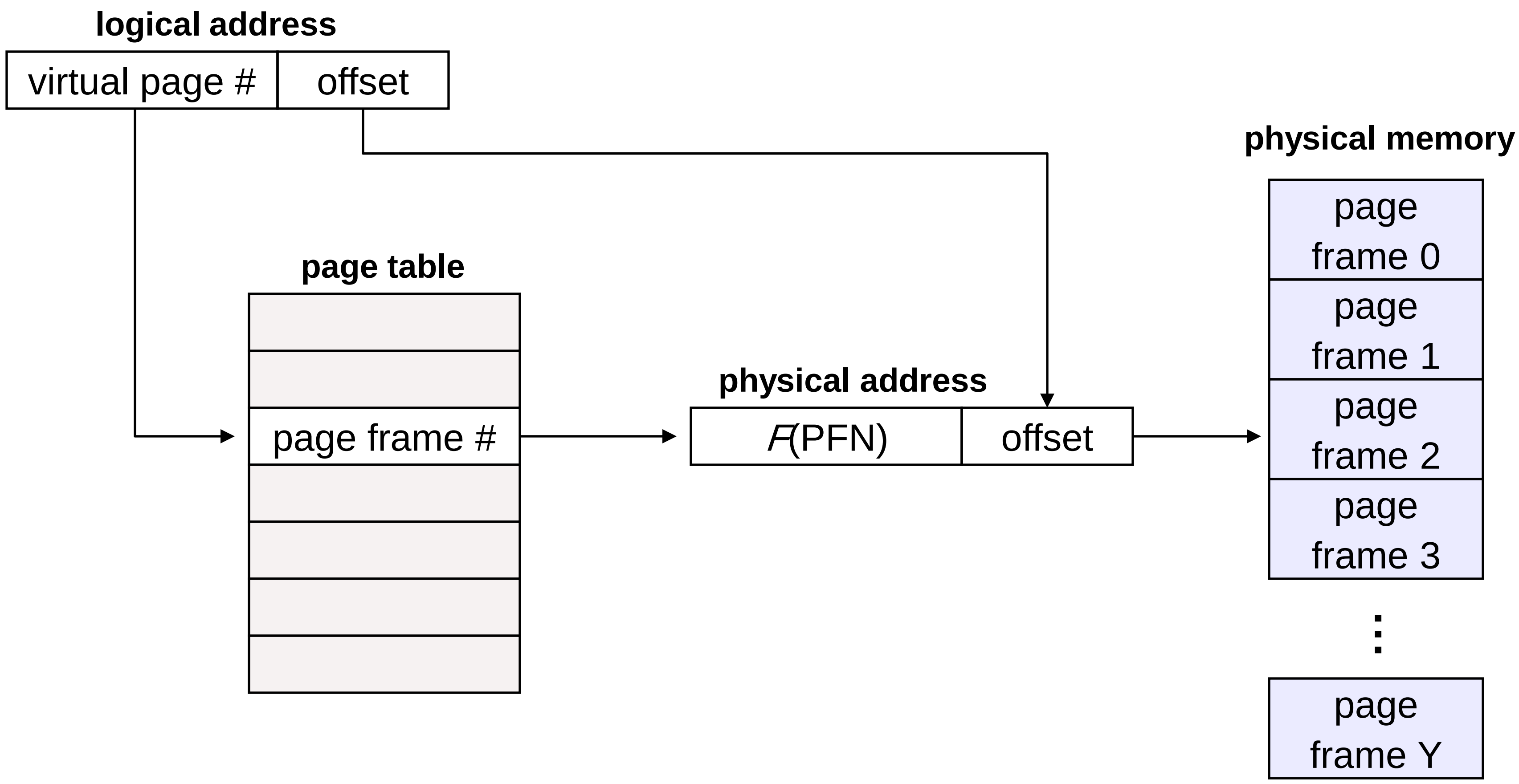
Paging

- Processes see a large *virtual address space*
    - Either contiguous or segmented
- Memory Manager divides the virtual address space into equal sized pieces called ***pages***
    - Some systems support more than one page size
- Memory Manager divides the physical address space into equal sized pieces called ***frames***
    - Size usually a power of 2 between 512 and 8192 bytes
        - Some modern systems support 64 megabyte pages!
    - Frame table
        - One entry per frame of physical memory; each entry is either
            - Free
            - Allocated to one or more processes
- **`sizeof(page) = sizeof(frame)`**

Address Translation for Paging

- Translating virtual addresses
  - a virtual address has two parts: ***virtual page number*** & ***offset***
  - **virtual page number** (VPN) is index into a *page table*
  - page table entry contains **page frame number** (PFN)
  - physical address is: `startof(PFN) + offset`
- Page tables
  - Supported by MMU hardware
  - Managed by the Memory Manager
  - Map virtual page numbers to page frame numbers
    - one *page table entry* (PTE) per page in virtual address space

Paging Translation



Page Translation Example

- Assume a 32-bit contiguous address space
  - Assume page size 4 KB ($\log_2(4096) = 12$ bits)
  - For a process to address the full logical address space
    - Need 2^{20} PTEs – VPN is 20 bits
    - Offset is 12 bits
- Translation of virtual address **`0x12345678`**
  - Offset is **`0x678`**
  - Assume PTE(**`0x12345`**) contains **`0x01010`**
  - Physical address is **`0x01010678`**

PTE Structure

1	1	1	3	20
V	R	M	prot	page frame number

- *Valid* bit gives state of this PTE
  - says whether or not a virtual address is valid – in memory and VA range
  - If not set, page might not be in memory or *may not even exist!*
- *Reference* bit says whether the page has been accessed
  - it is set by hardware *whenever* a page has been read or written to
- *Modify* bit says whether or not the page is *dirty*
  - it is set by hardware during *every* write to the page
- *Protection* bits control which operations are allowed
  - read, write, execute, etc.
- *Page* frame number (PFN) determines the physical page
  - physical page start address

- *Other* bits dependent upon machine architecture

Paging – Advantages

- Easy to allocate physical memory
  - pick any free frame
- No external fragmentation
  - All frames are equal
- Minimal internal fragmentation
  - Bounded by page/frame size
- Easy to swap out pages (called *pageout*)

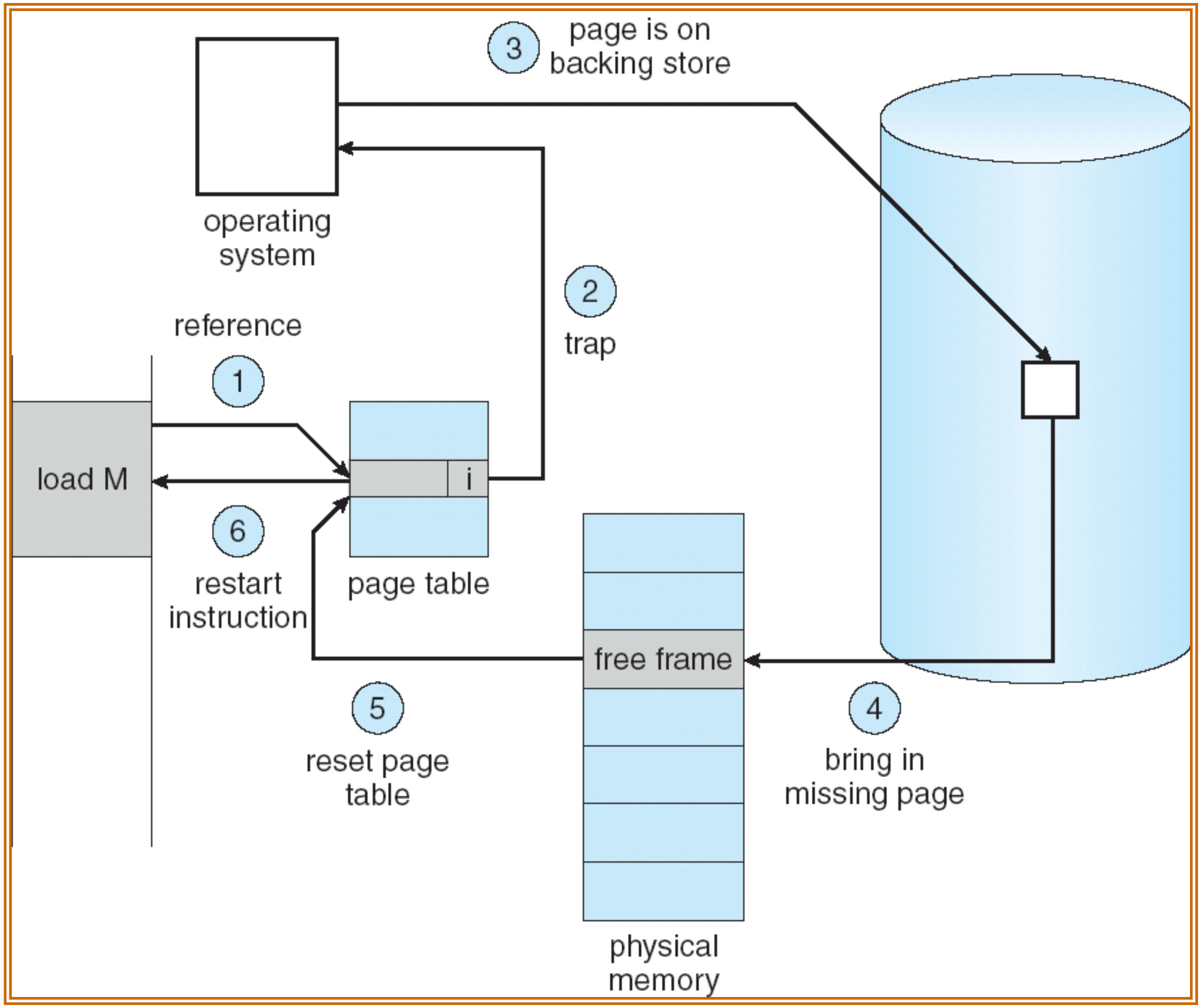
- Processes can run with not all pages swapped in

Definition — *Page Fault*

- *Trap* when process attempts to reference a virtual address in a page with ***Valid bit*** in PTE set to ***false***
    - E.g., page not in physical memory
- If page exists on disk:–
    - Suspend process
    - If necessary, throw out some other page (& update its PTE)
    - Swap in desired page, resume execution
- If page does not exist on disk:–
    - Return program error

      or
    - Prepare up a new page and resume execution
        - E.g., for growing the stack!

Steps in Handling a Page Fault



Requirement for Paging to Work!

- Machine instructions must be capable of *restarting*
- If execution was interrupted during a partially completed instruction, need to be able to
  - continue or
  - redo without harm
- This is a property of all modern CPUs …
  - … but not of some older CPUs!

Observations

- Recurring themes in paging:
  - ***Temporal Locality*** – locations referenced recently tend to be referenced again soon
  - ***Spatial Locality*** – locations near recent references tend to be referenced soon
- Definitions:
  - ***Working set:*** The set of pages that a process needs to run without frequent page faults
  - ***Thrashing:*** Excessive page faulting due to insufficient frames to support working set

Paging – Summary

- Partition virtual memory into equal size units called ***pages***
- Any page can fit into any ***frame*** in physical memory
- No relocation *in virtual memory* needed by loader
- Only ***active*** pages in physical memory at any time
- Supports very large virtual memories and segmentation
- Hardware assistance is essential

Review

- What is the main benefits of segmentations?

- Is paging needed while using segmentation? If yes, how can we apply paging in this case?