## Virtual and Physical Addresses

- Physical addresses are provided directly by the machine.
  - one physical address space per machine
  - addresses typically range from 0 to some maximum, though some portions of this range are usually used by the OS and/or devices, and are not available for user processes
- Virtual addresses (or logical addresses) are addresses provided by the OS to processes.
  - one virtual address space per process
  - addresses typically start at zero, but not necessarily
  - space may consist of several *segments*
- Address translation (or address binding) means mapping virtual addresses to physical addresses.

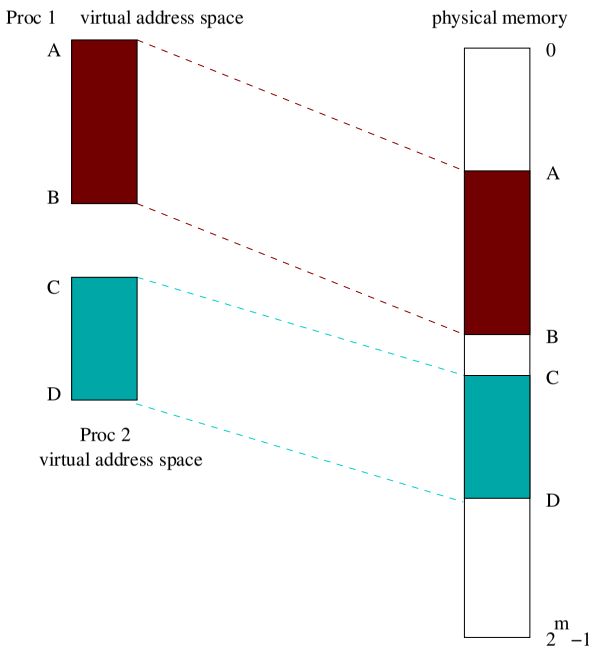




## Example 1: A Simple Address Translation Mechanism

- OS divides physical memory into partitions of different sizes.
- Each partition is made available by the OS as a possible virtual address space for processes.
- Properties:
  - virtual addresses are identical to physical addresses
  - address binding is performed by compiler, linker, or loader, not the OS
  - changing partitons means changing the virtual addresses in the application program
    - by recompiling
    - or by *relocating* if the compiler produces relocatable output
  - degree of multiprogramming is limited by the number of partitions
  - size of programs is limited by the size of the partitions

## Example 1: Address Space Diagram



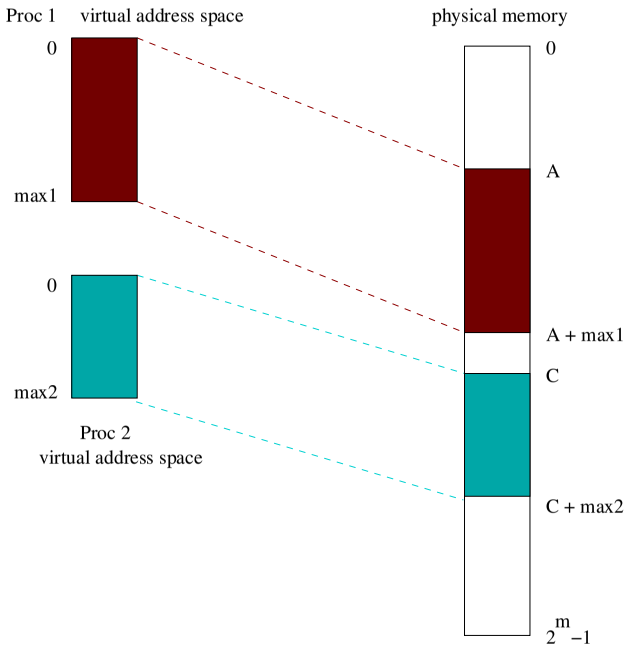





## Example 2: Dynamic Relocation

- hardware provides a *memory management unit* which includes a *relocation register*
- *dynamic binding:* at run-time, the contents of the relocation register are added to each virtual address to determine the corresponding physical address
- OS maintains a separate relocation register value for each process, and ensures that relocation register is reset on each context switch
- Properties
  - all programs can have address spaces that start with address 0
  - OS can relocate a process without changing the process's program
  - OS can allocate phyiscal memory dynamically (physical partitions can change over time), again without changing user programs
  - each virtual address space still corresponds to a contiguous range of physical addresses

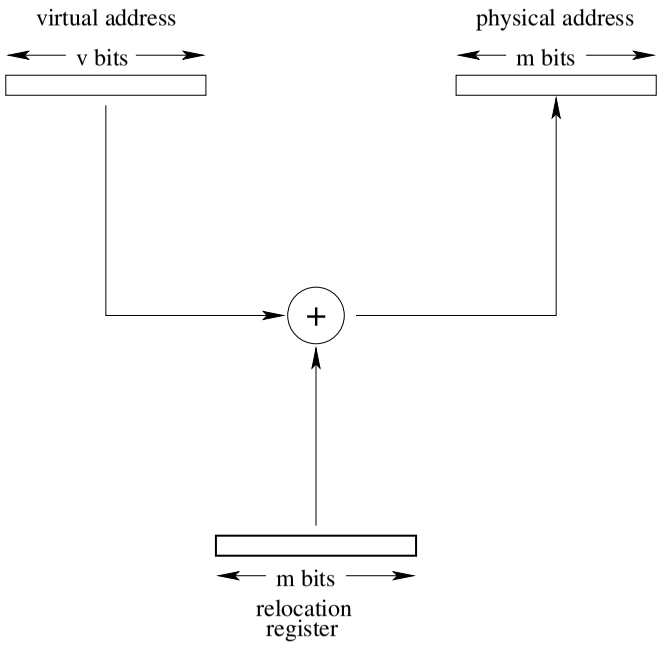
## Example 2: Address Space Diagram








## Example 2: Relocation Mechanism



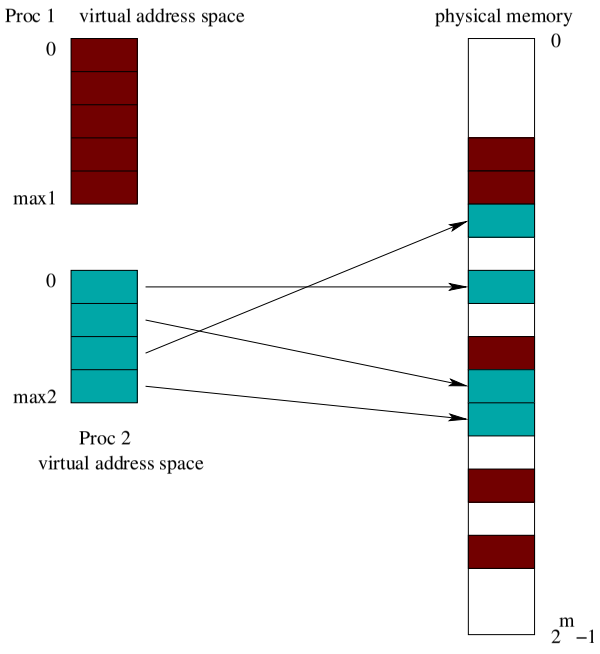
## Example 3: Paging

- Each virtual address space is divided into fixed-size chunks called *pages*
- The physical address space is divided into *frames*. Frame size matches page size.
- OS maintains a *page table* for each process. Page table specifies the frame in which each of the process's pages is located.
- At run time, MMU translates virtual addresses to physical using the page table of the running process.
- Properties
  - simple physical memory management
  - virtual address space need not be physically contiguous in physical space after translation.

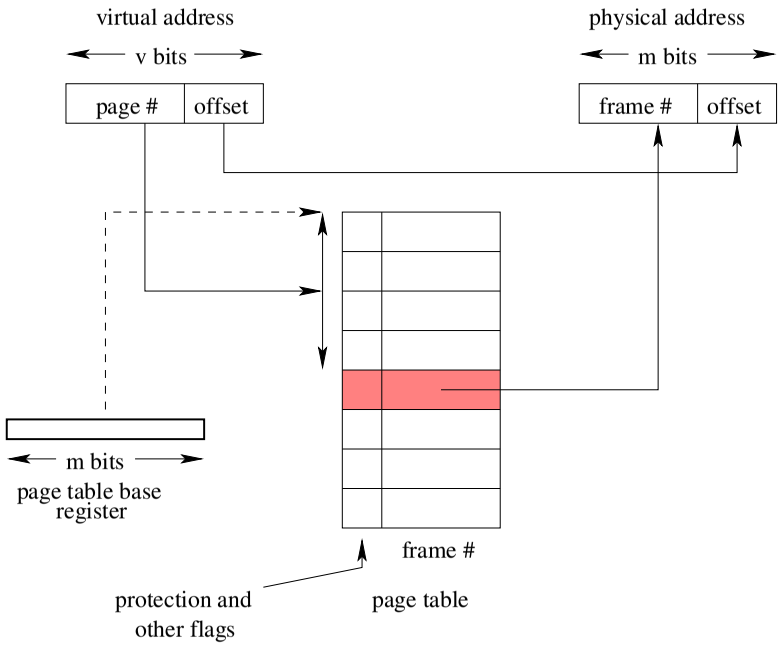




## Example 3: Address Space Diagram



## Example 3: Page Table Mechanism








## Summary of Binding and Memory Management Properties

**address binding time:**

- compile time: relocating program requires recompilation
- load time: compiler produces relocatable code
- dynamic (run time): hardware MMU performs translation

**physical memory allocation:**

- fixed or dynamic paritions
- fixed size paritions (frames) or variable size partitions

**physical contiguity:**

- virtual space is contiguous or discontiguous in physical space

## Physical Memory Allocation

**fixed allocation size:**

- space tracking and placement are simple
- *internal* fragmentation

**variable allocation size:**

- space tracking and placement more complex
  - placement heuristics: first fit, best fit, worst fit
- *external* fragmentation





## Memory Protection

- ensure that each process accesses only the physical memory that its virtual address space is bound to.
  - threat: virtual address is too large
  - solution: MMU *limit register* checks each virtual address
    - for simple dynamic relocation, limit register contains the maximum virtual address of the running process
    - for paging, limit register contains the maximum page number of the running process
  - MMU generates exception if the limit is exceeded
- restrict the use of some portions of an address space
  - example: read-only memory
  - approach (paging):
    - include read-only flag in each page table entry
    - MMU raises exception on attempt to write to a read-only page

## Roles of the Operating System and the MMU (Summary)

- operating system:
  - save/restore MMU state on context switches
  - handle exceptions raised by the MMU
  - manage and allocate physical memory
- MMU (hardware):
  - translate virtual addresses to physical addresses
  - check for protection violations
  - raise exceptions when necessary





## Speed of Address Translation

- Execution of each machine instruction may involve one, two or more memory operations
  - one to fetch instruction
  - one or more for instruction operands
- Address translation through a page table adds one extra memory operation (for page table entry lookup) for each memory operation performed during instruction execution
  - Simple address translation through a page table can cut instruction execution rate in half.
  - More complex translation schemes (e.g., multi-level paging) are even more expensive.
- Solution: include a Translation Lookaside Buffer (TLB) in the MMU
  - TLB is a fast, fully associative address translation cache
  - TLB hit avoids page table lookup

## TLB

- Each entry in the TLB contains a (page number,frame number) pair, plus copies of some or all of the page's protection bits, use bit, and dirty bit.
- If address translation can be accomplished using a TLB entry, access to the page table is avoided.
- TLB lookup is much faster than a memory access. TLB is an associative memory - page numbers of all entries are checked simultaneously for a match. However, the TLB is typically small ($10^2$ to $10^3$ entries).
- Otherwise, translate through the page table, and add the resulting translation to the TLB, replacing an existing entry if necessary. In a *hardware controlled* TLB, this is done by the MMU. In a *software controlled* TLB, it is done by the kernel.
- On a context switch, the kernel must clear or invalidate the TLB. (Why?)





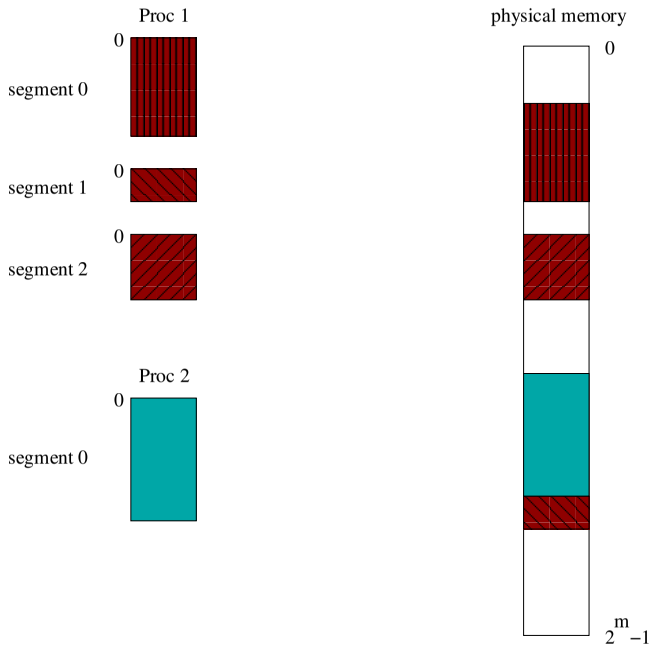
## Segmentation

- An OS that supports segmentation (e.g., Multics, OS/2) can provide more than one address space to each process.
- The individual address spaces are called *segments*.
- A logical address consists of two parts:

  (segment ID, address within segment)

- Each segment:
  - can grow or shrink independently of the other segments
  - has its own memory protection attributes
- For example, process could use separate segments for code, data, and stack.

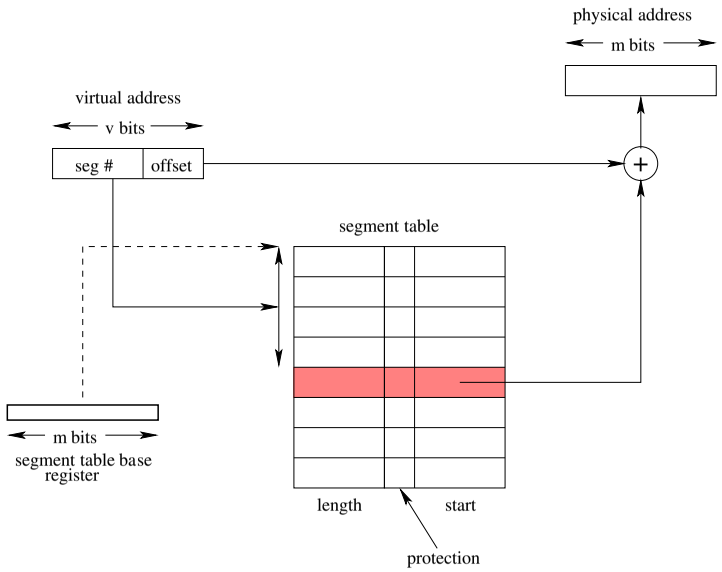
## Segmented Address Space Diagram





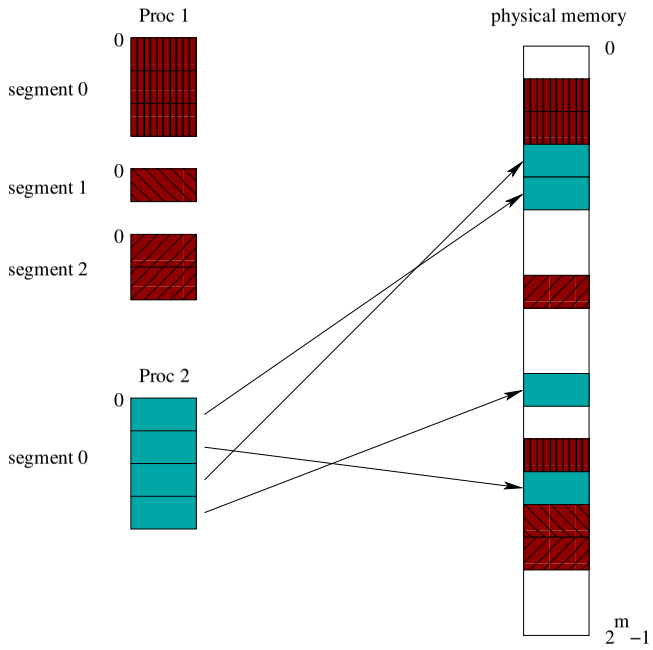



## Mechanism for Translating Segmented Addresses




This translation mechanism requires physically contiguous allocation of segments.

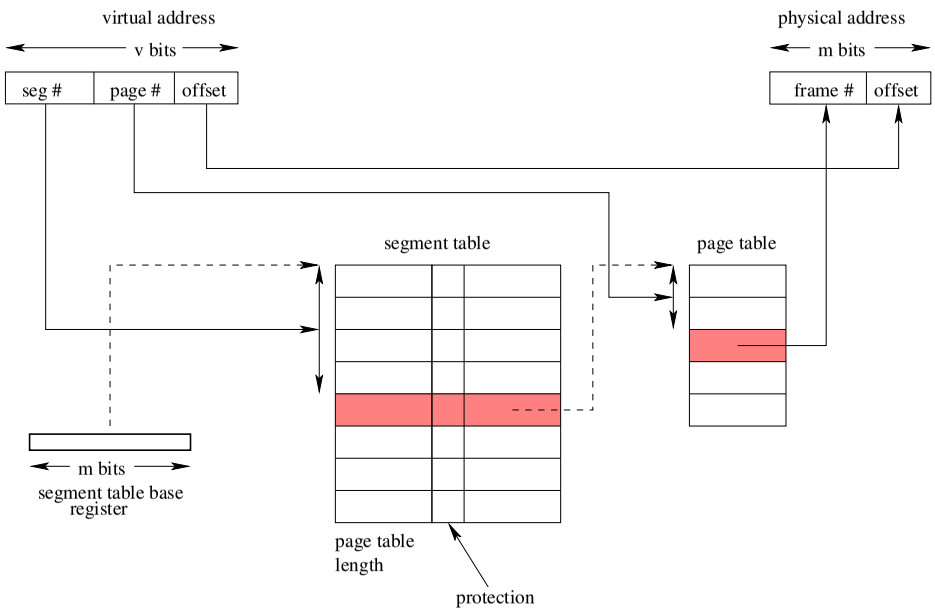
## Combining Segmentation and Paging



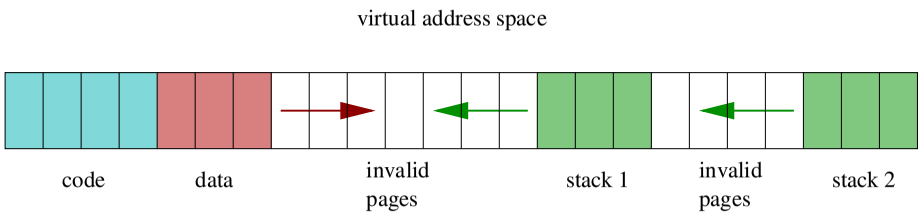





## Combining Segmentation and Paging: Translation Mechanism



## Simulating Segmentation with Paging



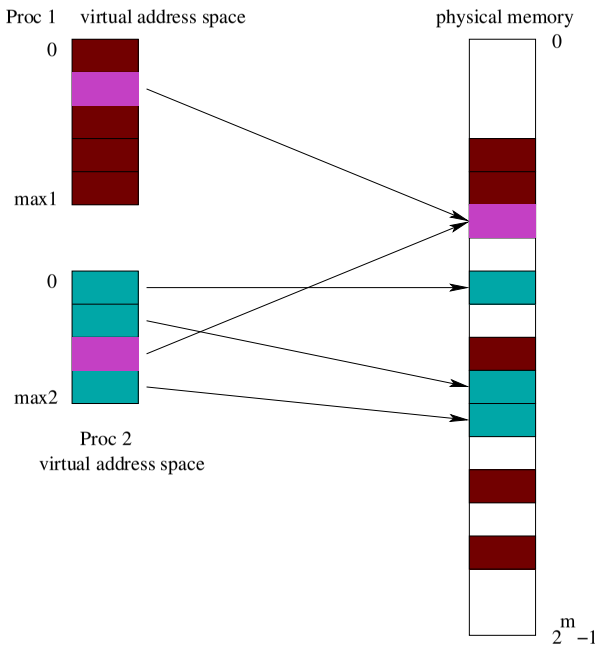





## Shared Virtual Memory

- virtual memory sharing allows parts of two or more address spaces to overlap
- shared virtual memory is:
  - a way to use physical memory more efficiently, e.g., one copy of a program can be shared by several processes
  - a mechanism for interprocess communication
- sharing is accomplished by mapping virtual addresses from several processes to the same physical addressed
- unit of sharing can be a page or a segment

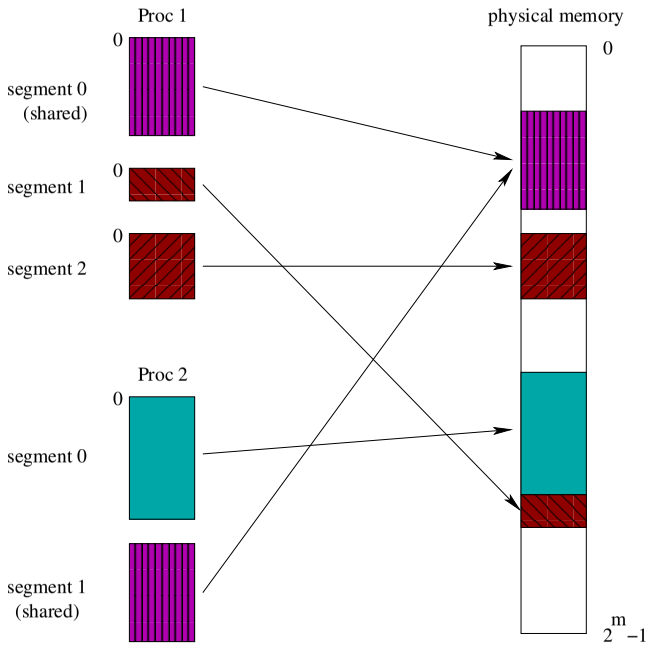
## Shared Pages Diagram








## Shared Segments Diagram



## An Address Space for the Kernel

**Option 1: Kernel in physical space**

- mechanism: disable MMU in system mode, enable it in user mode
- accessing process address spaces: OS must interpret process page tables
- OS must be entirely memory resident

**Option 2: Kernel in separate logical space**

- mechanism: MMU has separate state for user and system modes
- accessing process address spaces: difficult
- portions of the OS may be non-resident

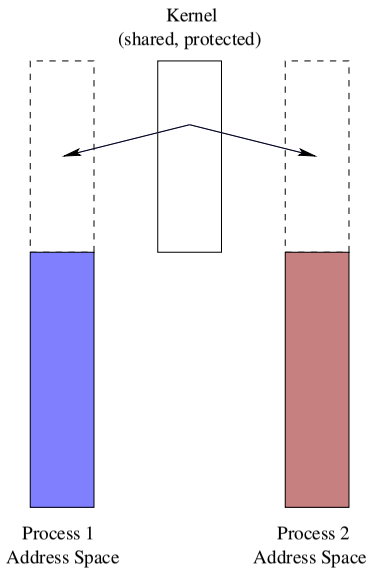
**Option 3: Kernel shares logical space with each process**

- memory protection mechanism is used to isolate the OS
- accessing process address space: easy (process and kernel share the same address space)
- portions of the OS may be non-resident





## The Kernel in Process' Address Spaces




Attempts to access kernel code/data in user mode result in memory protection exceptions, not invalid address exceptions.

## Memory Management Interface

- much memory allocation is implicit, e.g.:
  - allocation for address space of new process
  - implicit stack growth on overflow
- OS may support explicit requests to grow/shrink address space, e.g., Unix `brk` system call.
- shared virtual memory (simplified Solaris example):

  **Create:** `shmid = shmget(key,size)`

  **Attach:** `vaddr = shmat(shmid, vaddr)`

  **Detach:** `shmdt(vaddr)`

  **Delete:** `shmctl(shmid,IPC_RMID)`