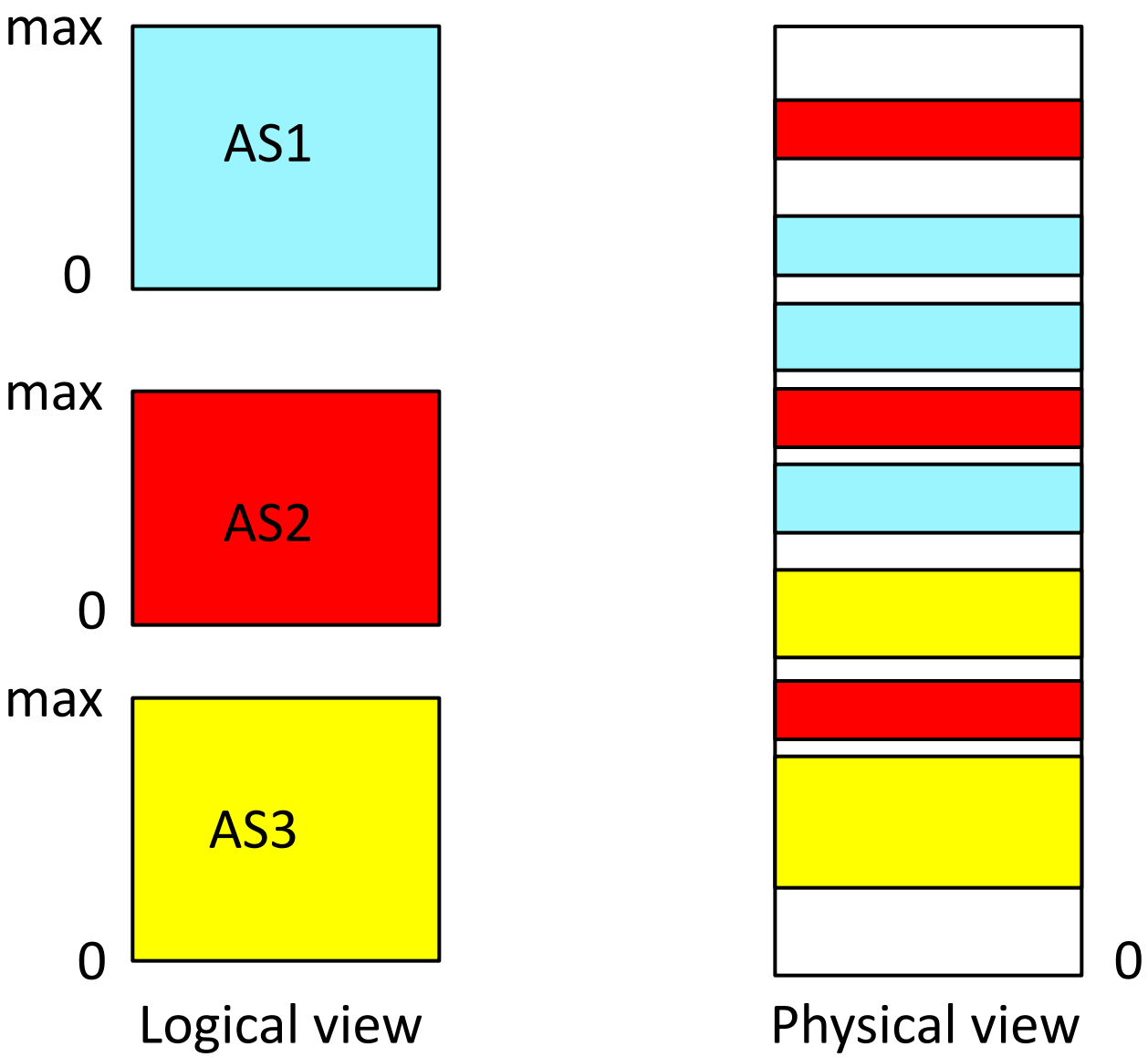
Introduction to Paging

COMS W4118

References: Operating Systems Concepts (9e), Linux Kernel Development, previous W4118s

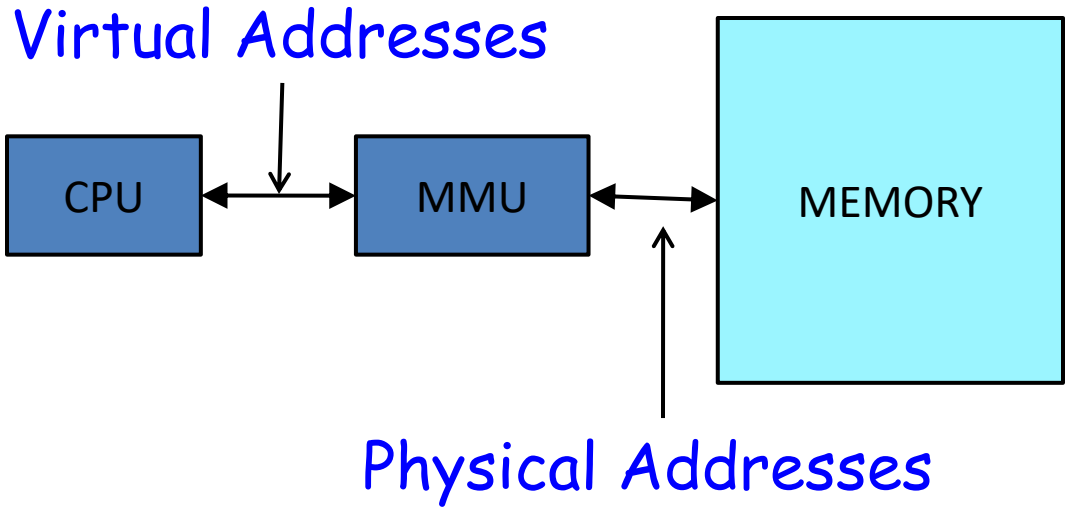
Multiple address spaces co-exist



Memory management wish-list

- Sharing
  - multiple processes coexist in main memory
- Transparency
  - Processes are not aware that memory is shared
  - Run regardless of number/locations of other processes
- Protection
  - Cannot access data of OS or other processes
- Efficiency: should have reasonable performance
  - Purpose of sharing is to increase efficiency
  - Do not waste CPU or memory resources (fragmentation)

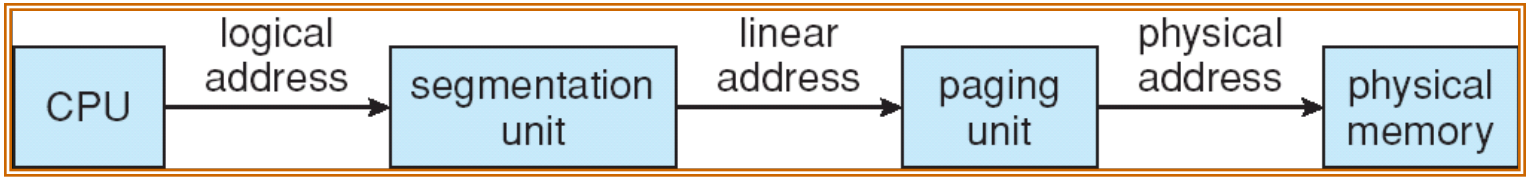
Memory Management Unit (MMU)



- Map program-generated address (virtual address) to hardware address (physical address) dynamically at every reference
- Check range and permissions
- Programmed by OS

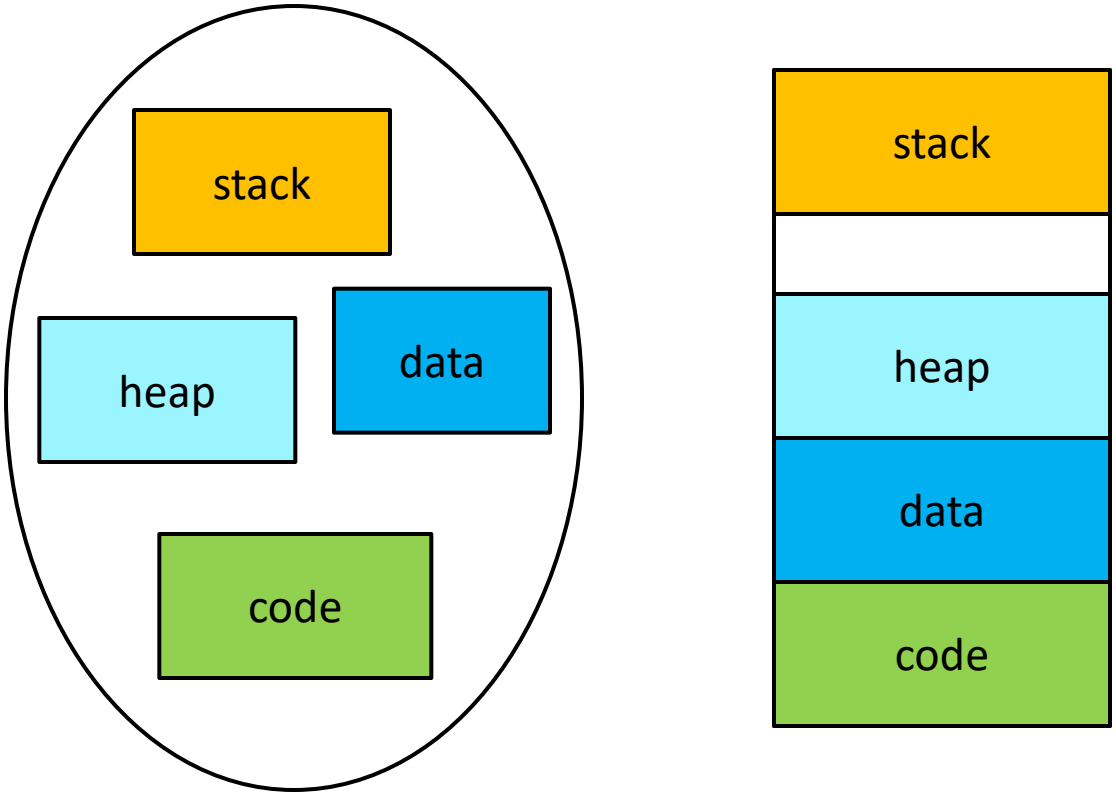
x86 address translation

- CPU generates virtual address (seg, offset)
  - Given to segmentation unit
    - Which produces linear addresses
  - Linear address given to paging unit
    - Which generates physical address in main memory

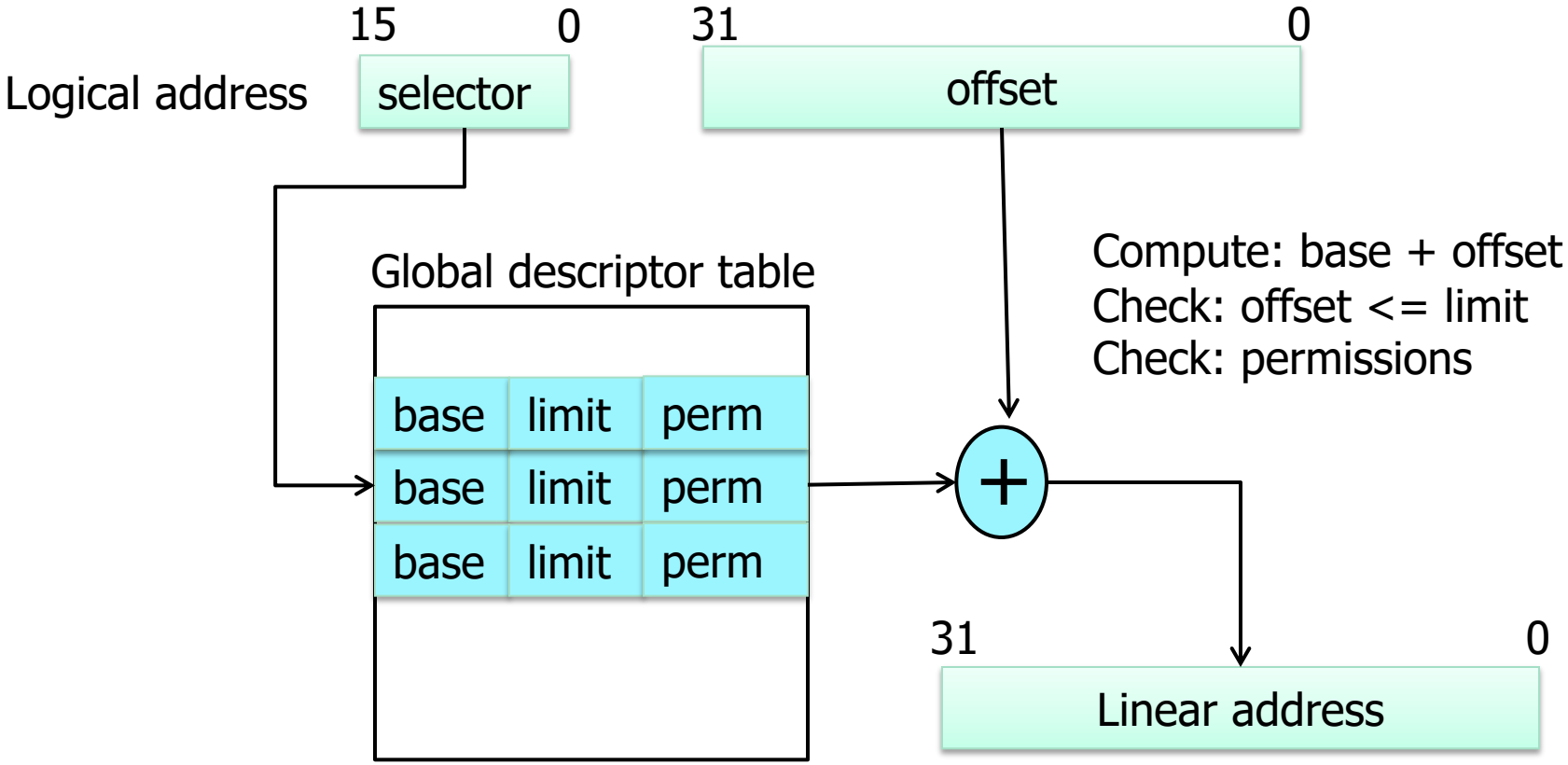


Segmentation

- Divide virtual address space into separate logical segments; each is part of physical mem



x86 segmentation hardware

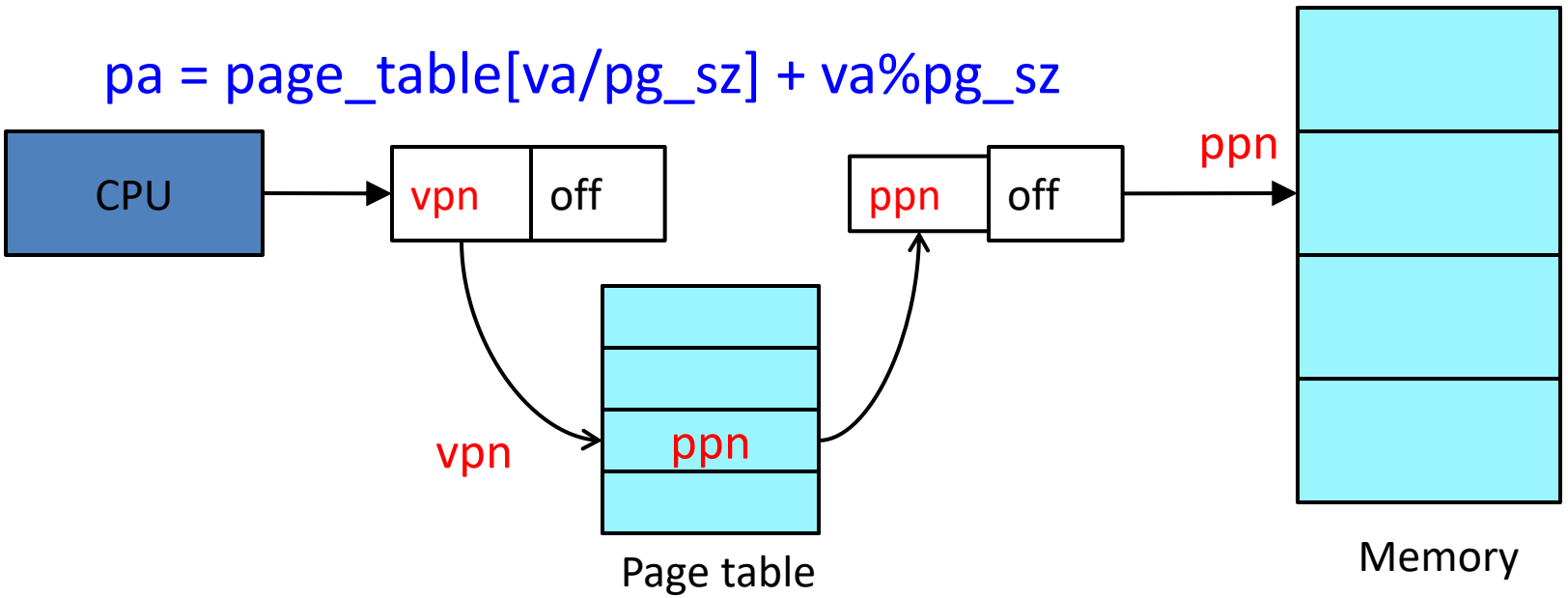


Paging overview

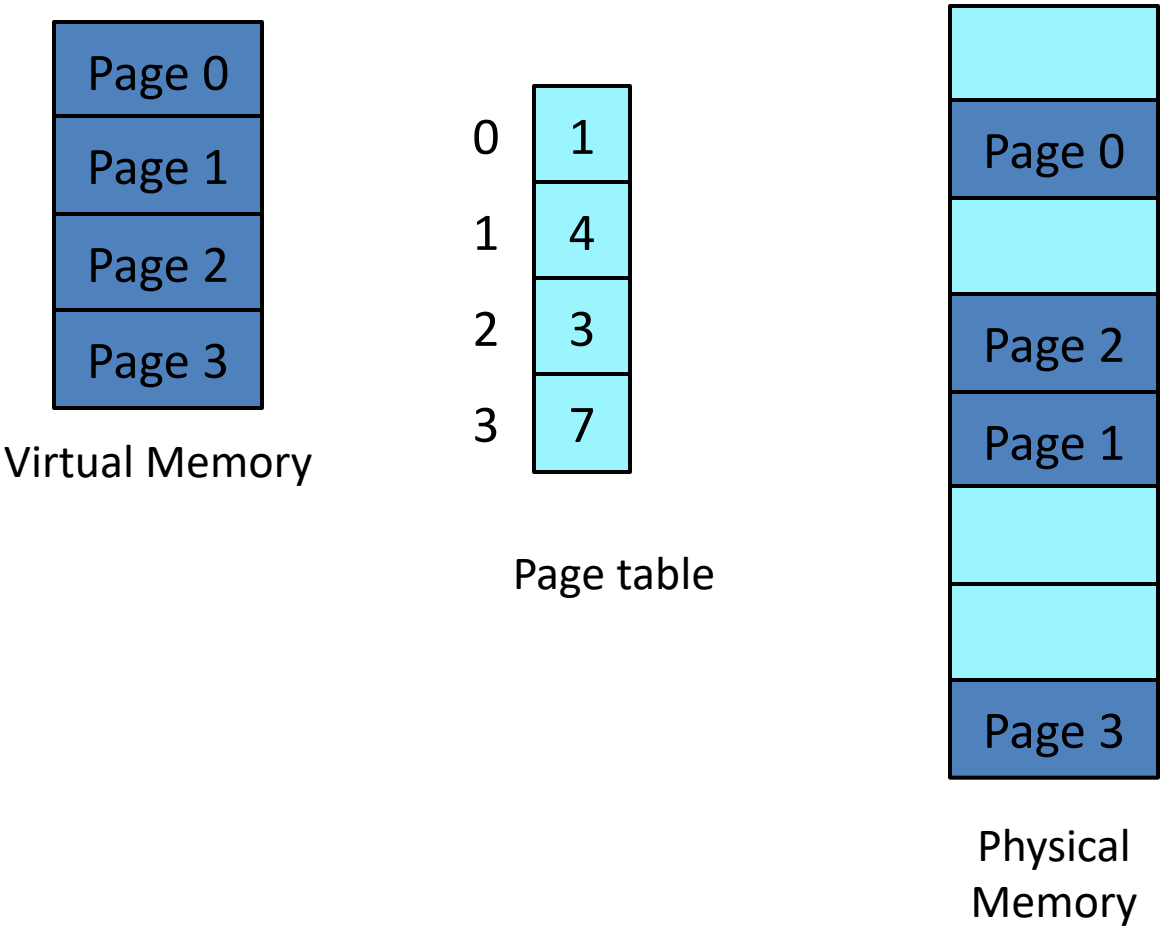
- Goal
  - Eliminate fragmentation due to large segments
  - Don't allocate memory that will not be used
  - Enable fine-grained sharing
- Paging: divide memory into fixed-sized pages
  - For both virtual and physical memory
- Another terminology
  - A virtual page: page
  - A physical page: frame

Page translation

- Address bits = page number + page offset
- Translate virtual page number (vpn) to physical page (frame) number (ppn/pfn) using page table



Page translation example



Page translation exercise

- 8-bit virtual address, 10-bit physical address, each page is 64 bytes

1. How many virtual pages?
   - 2^8 / 64 = 4 virtual pages
2. How many physical pages?
   - 2^10/64 = 16 physical pages
3. How many entries in page table?
   - Page table contains 4 entries
4. Given page table = [2, 5, 1, 8], what's the physical address for virtual address 241?
   - 241 = 11110001b
   - 241/64 = 3 = 11b
   - 241%64 = 49 = 110001b
   - page_table[3] = 8 = 1000b
   - Physical address = 8 * 64 + 49 = 561 = 1000110001b

Page translation exercise

m-bit virtual address, n-bit physical address, k-bit page size

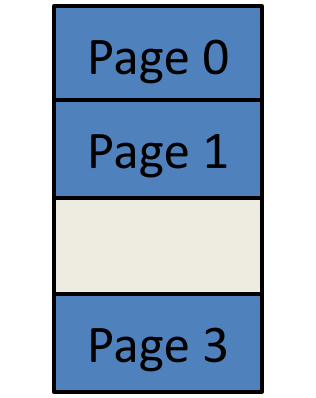
- # of virtual pages: $2^{(m-k)}$
- # of physical pages: $2^{(n-k)}$
- # of entries in page table: $2^{(m-k)}$
- vpn = va / 2^k
- offset = va % 2^k
- ppn = page_table[vpn]
- pa = ppn * 2^k + offset

Page protection

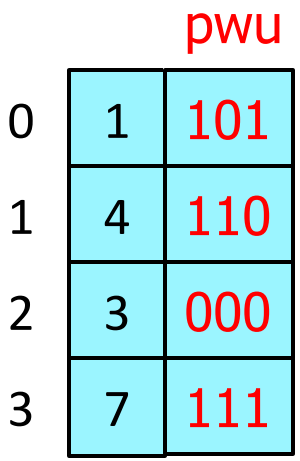
- Implemented by associating protection bits with each virtual page in page table
- Why do we need protection bits?
- Protection bits
  - present bit: map to a valid physical page?
  - read/write/execute bits: can read/write/execute?
  - user bit: can access in user mode?
  - x86: PTE_P, PTE_W, PTE_U
- Checked by MMU on each memory access

Page protection example

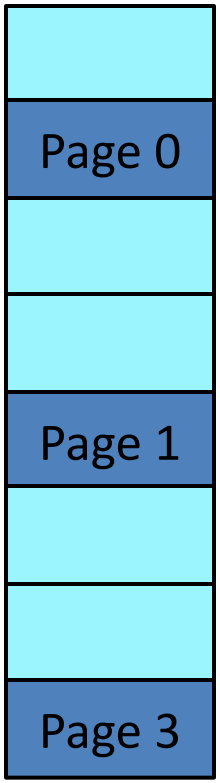
- What kind of pages?



Virtual Memory



Page table



Physical Memory

Implementation of page table

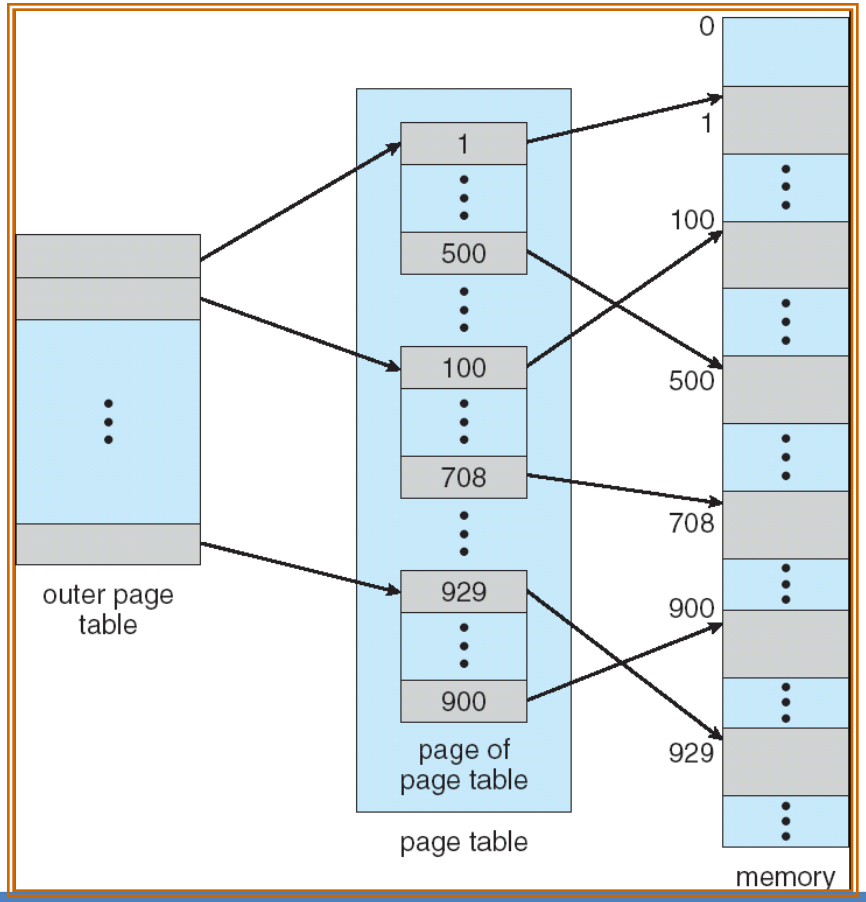
- Page table is stored in memory
  - Page table base register (PTBR) points to the base of page table
    - x86: cr3
  - OS stores base in process control block (PCB)
  - OS switches PTBR on each context switch
- Problem: each data/instruction access requires two memory accesses
  - Extra memory access for page table

Page table size issues

- Given:
  - A 32 bit address space (4 GB)
  - 4 KB pages
  - A page table entry of 4 bytes
- Implication: page table is 4 MB per process!
- Observation: address space are often sparse
  - Few programs use all of 2^{32} bytes
- Change page table structures to save memory
  - Trade translation time for page table space

Hierarchical page table

- Break up virtual address space into multiple page tables at different levels



Hierarchical page tables