CSE320 System Fundamentals II

Virtual Memory

YOUNGMIN KWON / TONY MIONE

Announcements

Surveys!

- - Today : ABET Survey at end of class
- - Before June 17 : Course Eval Survey –
  - Online
  - Link is on course schedule page

Final : Tue, June 14, 2022 : 9:00 AM -> 11:30 AM, B204

Assignment 7 : Now Due Mon, June 7, midnight

Virtual Memory

Uses main memory efficiently

- Cache for an address space stored on disk
- Keeping only the active areas in main memory

Simplifies memory management

- Providing each process with a uniform address space

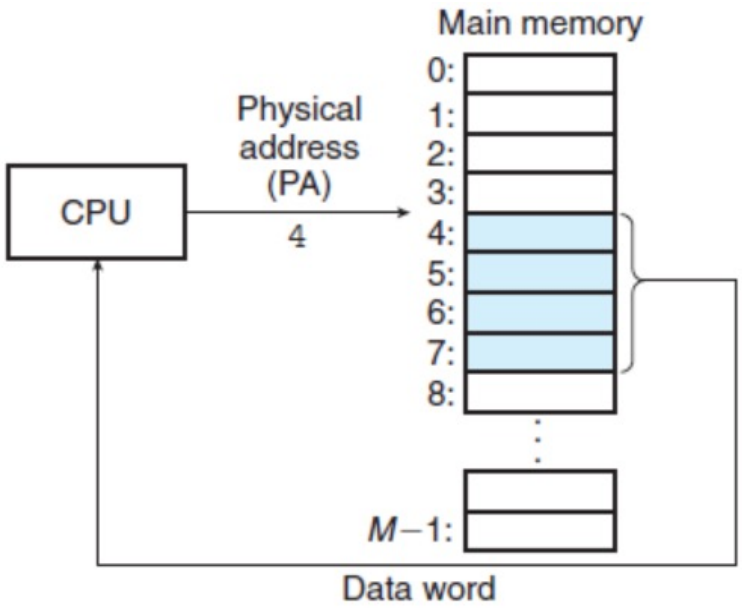
Protects the address space

- from corruption by other processes

Physical and Virtual Addressing

Physical addressing

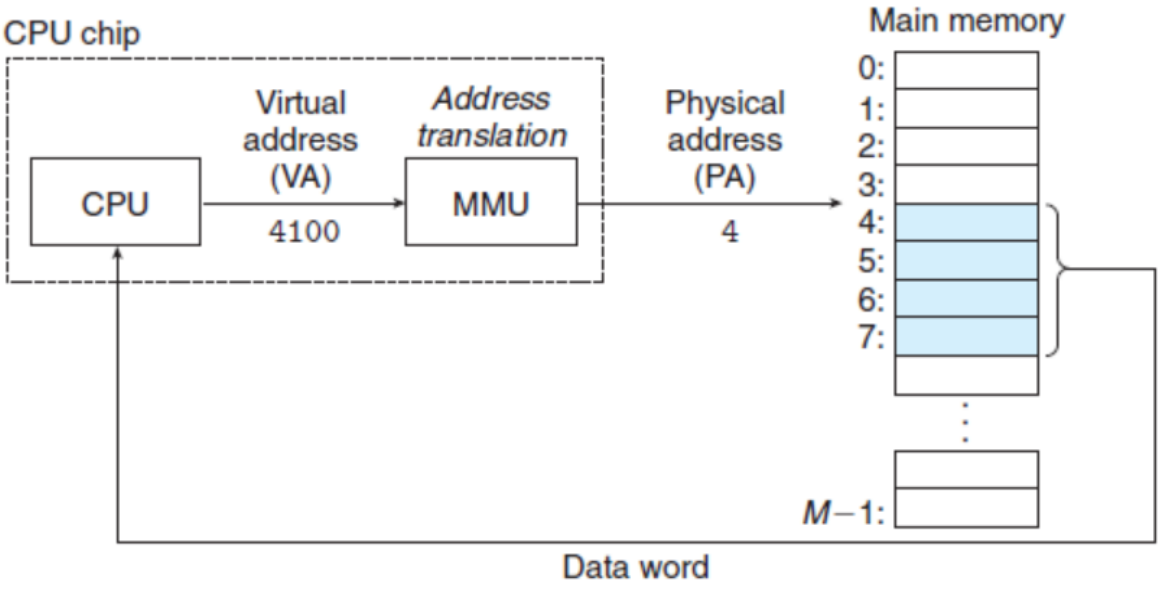
- CPU generates a physical address and passes it to main memory over the memory bus



Physical and Virtual Addressing

Virtual addressing

- CPU generates a virtual address
- Virtual addresses are translated to physical addresses by the memory management unit (MMU)



Address Space

Address space

- An ordered set of nonnegative addresses: {0, 1, 2, …}

Linear address space

- If the integers in the address space is contiguous
  - Virtual address space of N (=2^n) bytes: {0, 1, 2, …, N-1}
  - Physical address space of M(=2^m) bytes: {0, 1, 2, …, M-1}

Each byte of main memory has

- a virtual address chosen from the virtual address space
- a physical address chosen from the physical space

VM as a Tool for Caching

Virtual memory

- Organized as an array of N contiguous byte-size cells stored on disks
- Contents of the array on disk are cached in main memory

Pages

- Partition the virtual memory into blocks of size $P(=2^p)$ bytes, called pages, that serve as a transfer unit between disks and main memory

DRAM Cache Organization

Disk vs DRAM

- Disk is about 100,000 times slower than a DRAM
- Reading the first byte from a disk sector is 100,000 times slower than reading successive bytes in the sector

Large miss penalty

- Large virtual page (4KB ~ 2MB)
- Fully associative cache
- Sophisticated replacement algorithm
- Write-back

Page Tables

Page table

- Maps virtual pages to physical pages
- Each process has a page table

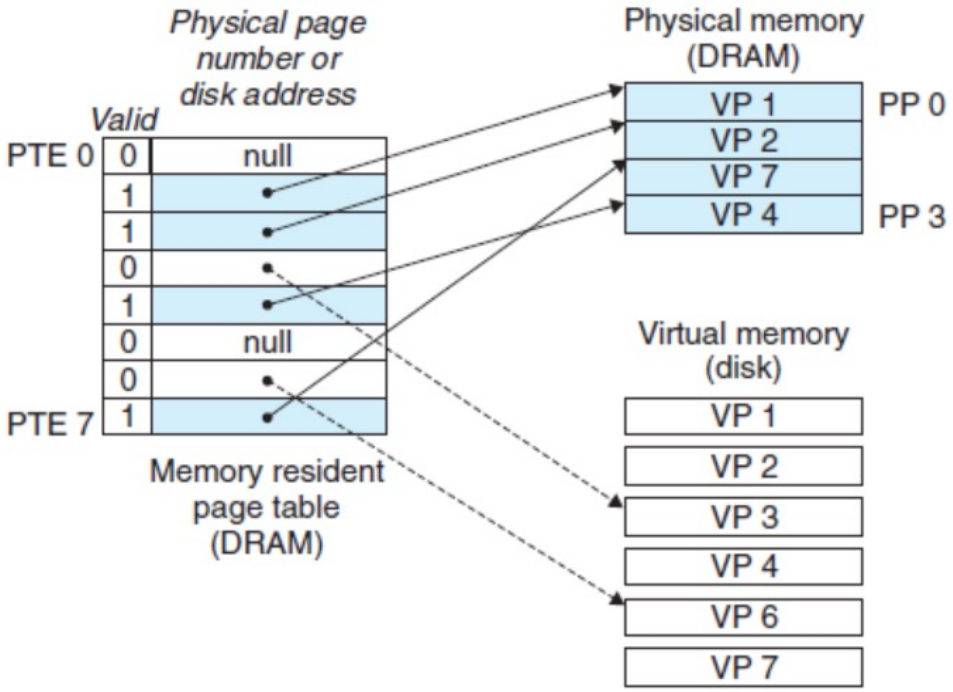
MMU

- Reads the page table when it translates a virtual address to a physical address

Operating system

- Maintains the contents of the page table and transferring pages between disk and DRAM

Page Tables

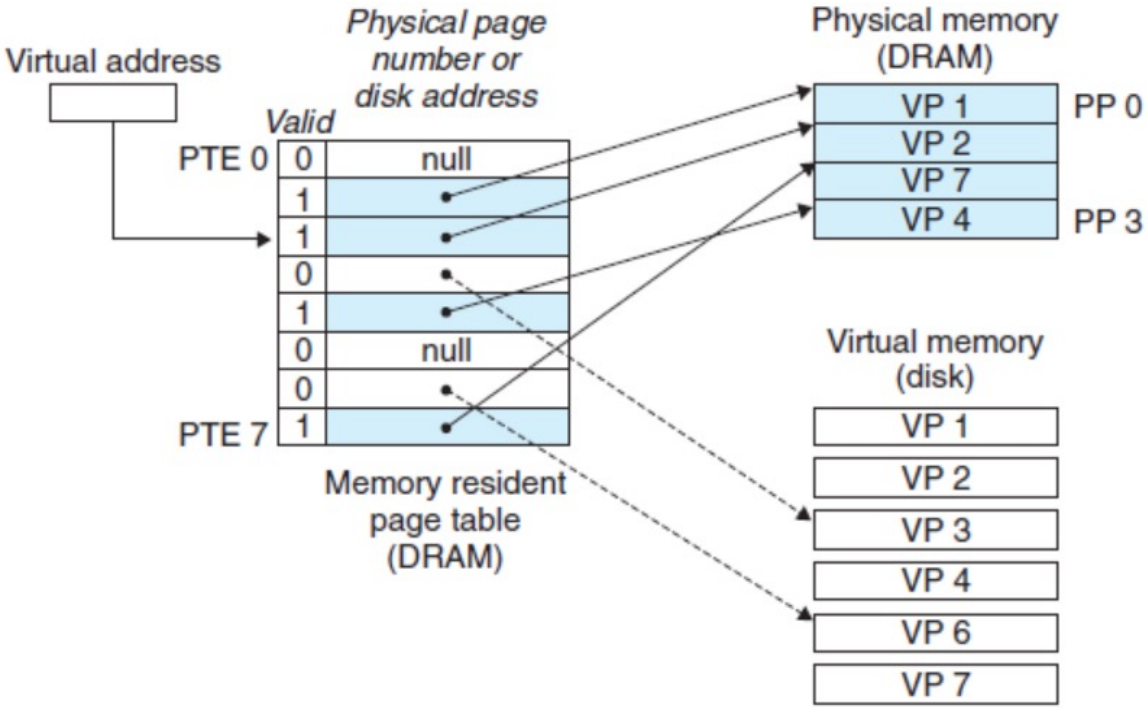


valid bit = 1: address => physical address

valid bit = 0 & address = 0: page not allocated

valid bit = 0 & address != 0: address => virtual page on disk

Page Hits



If valid bit is set, MMU uses the physical address in PTE (page table element) to construct the physical address of the word

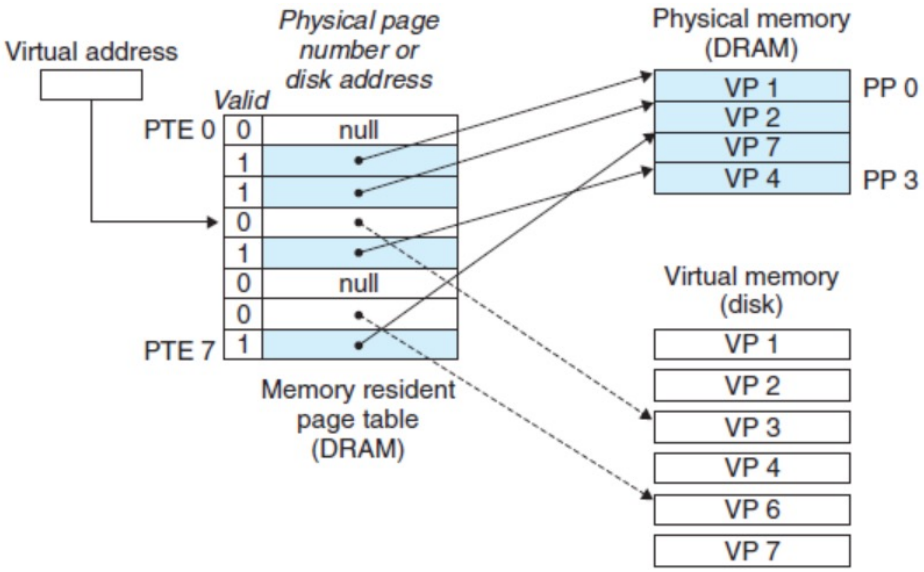
Page Faults

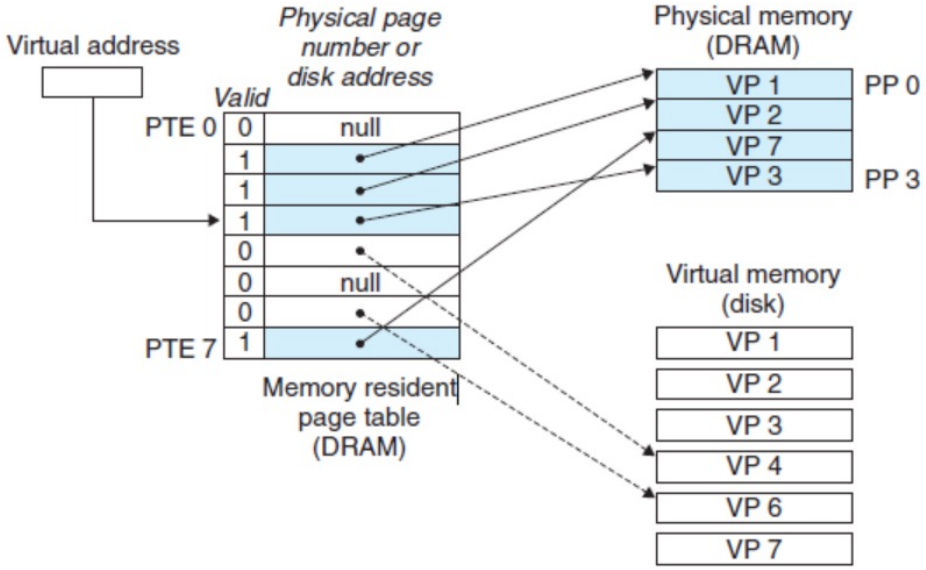
If valid bit is not set, MMU triggers a page fault exception

Page fault exception handler in Kernel

- Selects a victim page (if necessary)
  - If the victim page has been modified, copy it to disk
  - Clear the valid bit and update the victim PTE address
- Copy the virtual page to the physical page and update the page
- Restart the faulting instruction

Page Faults





Before

Trying to read from VP3

VP4 is a selected victim page

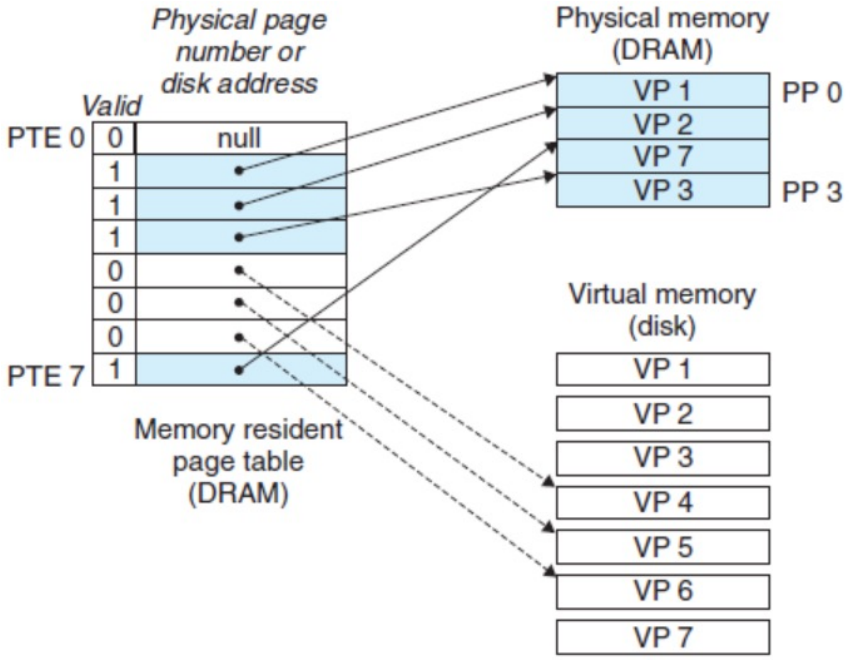
After

Page-hit when reading from VP3 again

Allocating a Page

Creating a new page of virtual memory

- e.g. malloc will create a new page of virtual memory
- Create room on disk
- Update PTE to point to the newly created page on disk



VP5 is created on disk and
PTE5 is pointing to the location

VM as a Tool for Memory Management: Simplify LINKING

Separate address space allows each process to have the same basic format for its memory image regardless of where code/data reside in memory

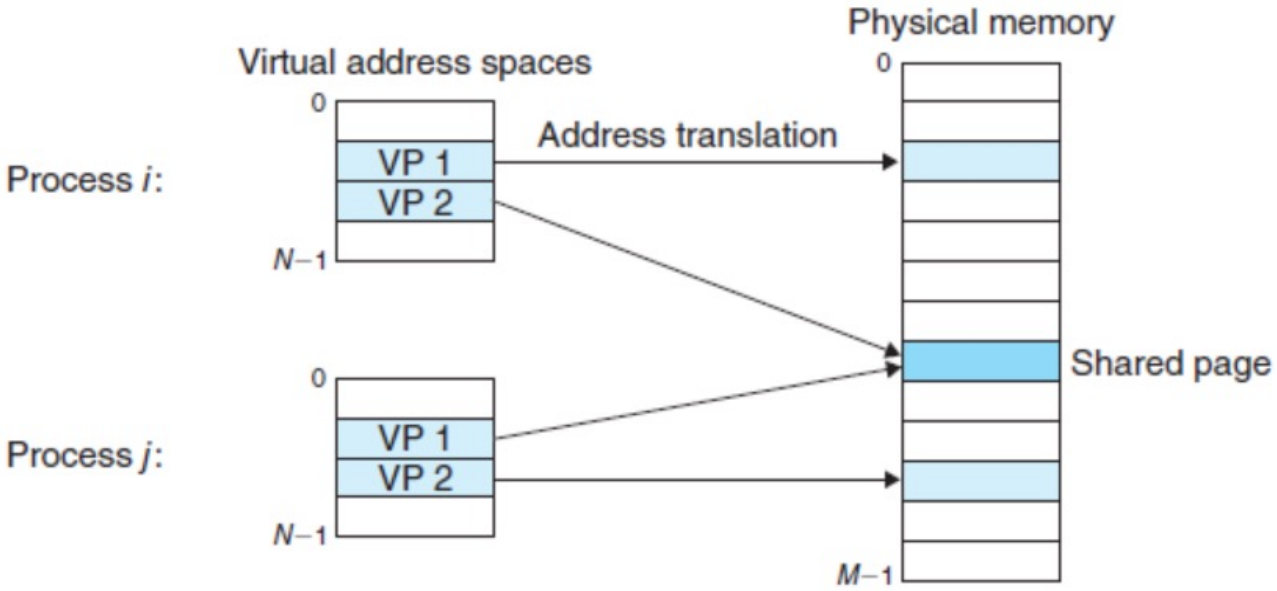
The uniformity makes it easy to create fully linked executables

VM as a Tool for Memory Management: Simplify LOADING

Linux: to load .text and .data sections of an object file into memory

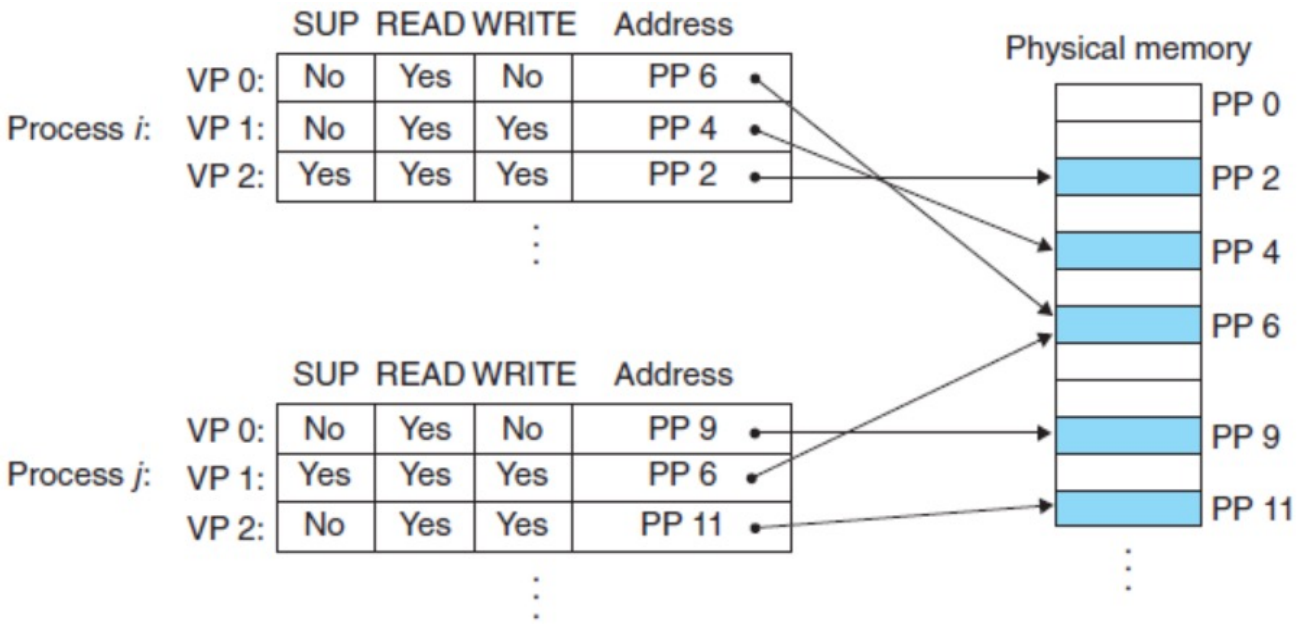
- Allocate pages for code and data
- Mark them as invalid
- Point their PTE to the appropriate locations in the object file
- The virtual memory system will automatically copy them into physical memory

VM as a Tool for Memory Management: Simplify SHARING



Rather than having separate codes for kernel, standard C libraries, *etc* in each process, they can be shared

VM as a Tool for Memory Protection



Control the access to the contents of a virtual page by adding some additional permission bits

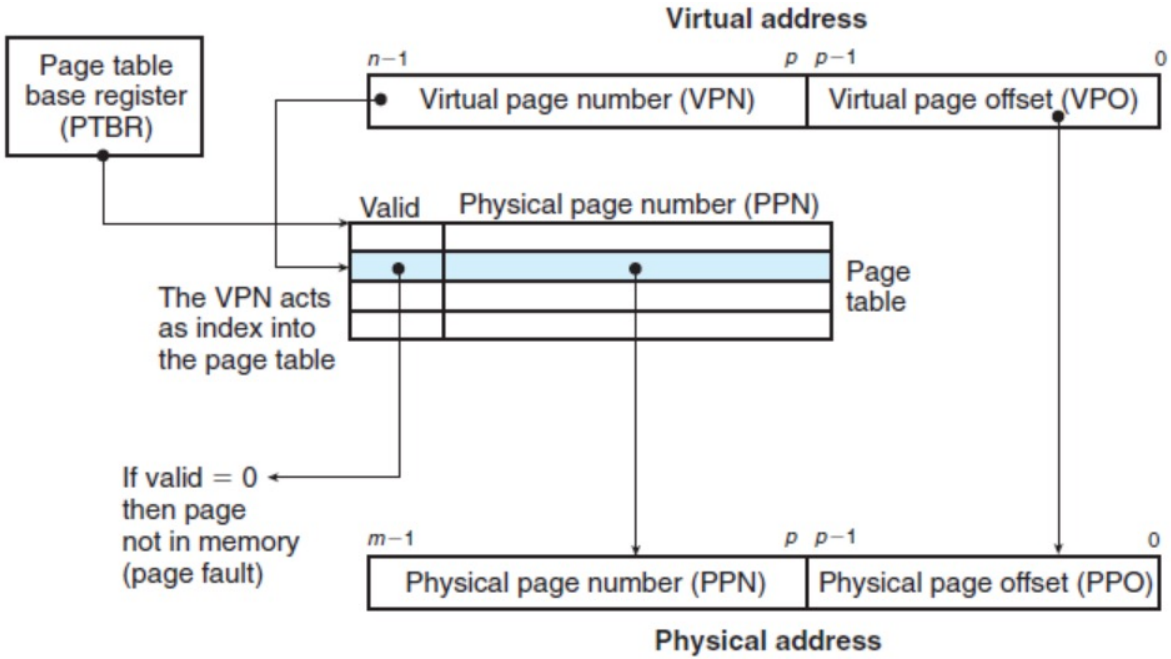
- SUP: can be accessed in the kernel mode
- READ, WRITE: read/write control

Address Translation

Address Translation

- Mapping between an N-element virtual address space (VAS) and an M-element physical address space (PAS)
- MAP: VAS$\rightarrow$ PAS $\cup\varnothing$
- MAP(A) = A' if data at A are present at A' in PAS;
  $\varnothing$ otherwise

Address Translation

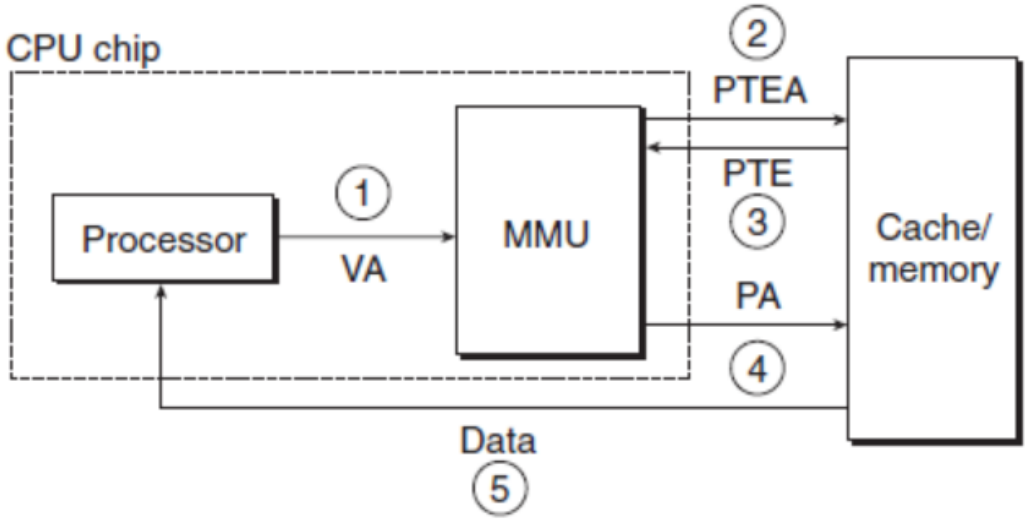


Page table base register (PTBR) points to the current page table

Virtual address is divided into virtual page number (VPN) and virtual page offset (VPO)

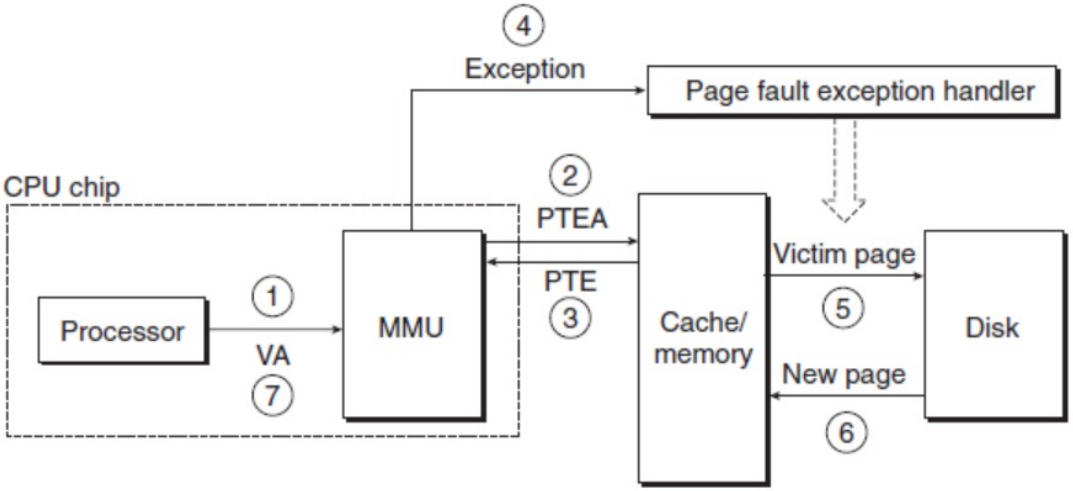
Corresponding physical address is the concatenation of physical page number (PPN) and VPO

Address Translation: page-hit



1. Processor generates a virtual address and sends it to MMU
2. MMU generates PTEA and requests from cache/main memory
3. Cache/main memory returns PTE to MMU
4. MMU constructs physical address and sends it to cache/main memory
5. Cache/main memory returns the requested data to the processor

Address Translation: page-miss



Step 1 .. 3 are the same as page-hit case

4. Valid bit in PTE is 0 → MMU triggers an exception → kernel's page fault exception handler runs
5. Fault handler identifies a victim page and pages it out to disk if dirty
6. Fault handler pages in the new page and updates the PTE in memory
7. Fault handler returns to the original process and restart the instruction that caused the page fault

Questions?