# Virtual Memory

CSCI 2021: Machine Architecture and Organization

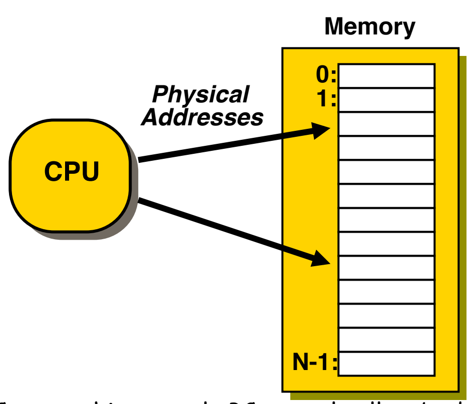
Antonia Zhai
Department Computer Science and Engineering
University of Minnesota
http://www.cs.umn.edu/~zhai

With Slides from Bryant and O'Hallaron

University of Minnesota

## A System with Physical Memory Only

**Addresses generated by the CPU correspond directly to physical memory**




Examples: most Cray machines, early PCs, nearly all embedded systems, etc.

## Motivations for Virtual Memory

Simplify Memory Management

- Multiple processes resident in main memory
  - Each process with its own address space
- Only “active” code and data is actually in memory
  - Allocate more memory to process as needed

Use Physical DRAM as a Cache for the Disk

- Address space of a process can exceed physical memory size
- Sum of address spaces of multiple processes can exceed physical memory

Provide Protection

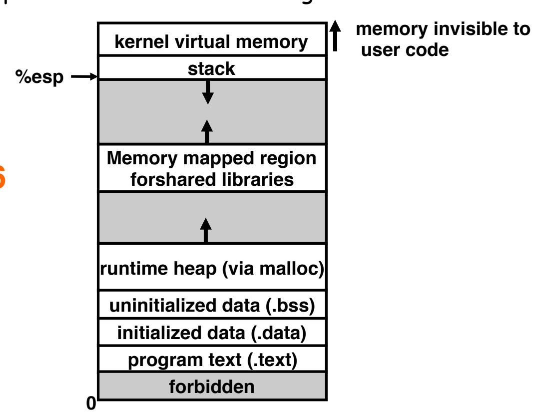
- One process can’t interfere with another.
  - because they operate in different address spaces.
- User process cannot access privileged information
  - different sections of address spaces have different permissions



## Motivation #1: Memory Management

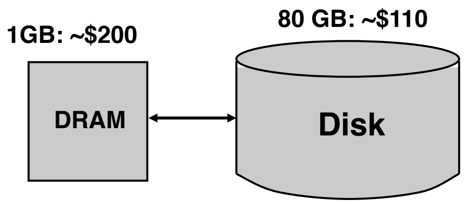
- Multiple processes can reside in physical memory.
- How do we resolve address conflicts?
  - what if two processes access something at the same address?

Linux/x86 process memory image



## Motivation #2: DRAM a "Cache" for Disk

- Full address space is quite large:
  - 32-bit addresses: ~4,000,000,000 (4 billion) bytes
  - 64-bit addresses: ~16,000,000,000,000,000,000 (16 quintillion) bytes
- Disk storage is ~300X cheaper than DRAM storage
  - 80 GB of DRAM: ~ $33,000
  - 80 GB of disk: ~ $110
- To access large amounts of data in a cost-effective manner, the bulk of the data must be stored on disk






## Address Space

- Address space
  - An ordered set of nonnegative integer addresses
- Linear address space
  - Integers in the address space is consecutive
- Virtual address space
  - The set of addresses generated by the CPU
  - {0, …, N-1} specified by n bits where $2^n = N$
- Physical address space
  - Corresponds to the physical memory in the system
  - {0, …, M-1} specified by m bits where $2^m = M$

# VM Address Translation

Normally, M < N

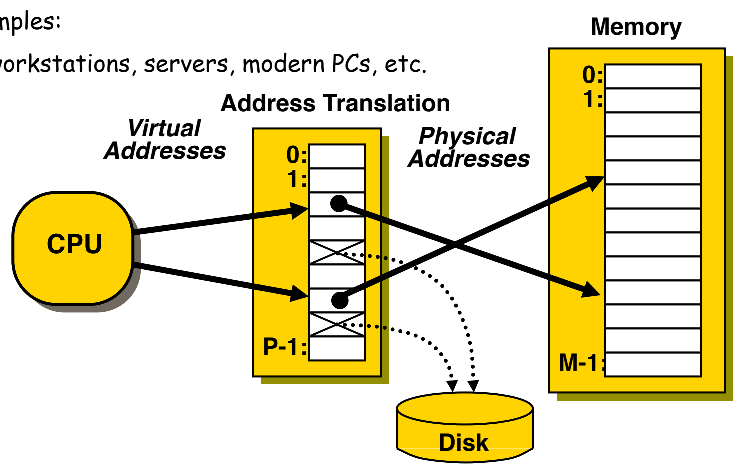
- Address Translation
  - MAP: $V \rightarrow P \cup \{\varnothing\}$
  - For virtual address a:
    - MAP(a) = a'
      - if data at virtual address a at physical address a' in P
    - MAP(a) = $\varnothing$
      - if data at virtual address a not in physical memory
      - Either invalid or stored on disk



# A System with Virtual Memory

Examples:

- workstations, servers, modern PCs, etc.




- **Address Translation: Hardware converts virtual addresses to physical addresses via OS-managed lookup table (page table)**

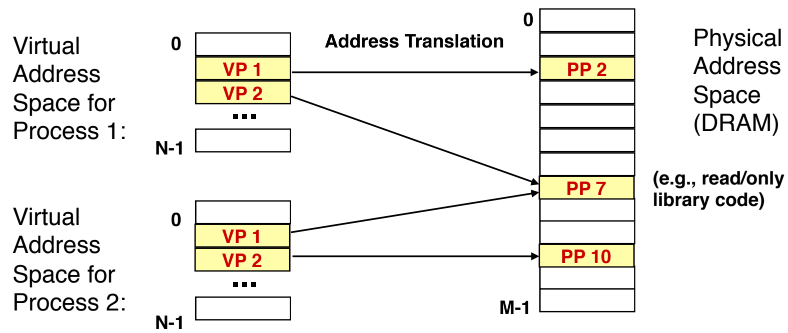
# Separate Virtual Address Spaces

Virtual and physical address spaces divided into equal-sized blocks

- blocks are called "pages" (both virtual and physical)

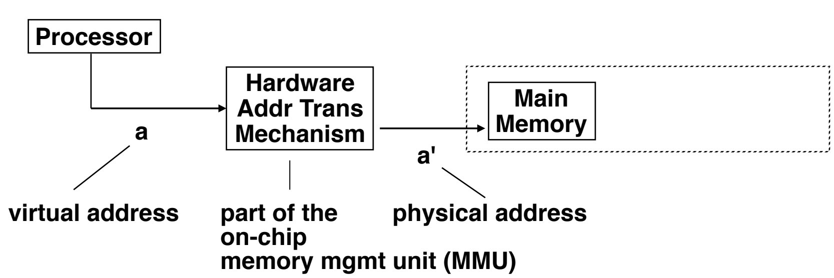
Each process has its own virtual address space

- operating system controls how virtual pages as assigned to physical memory

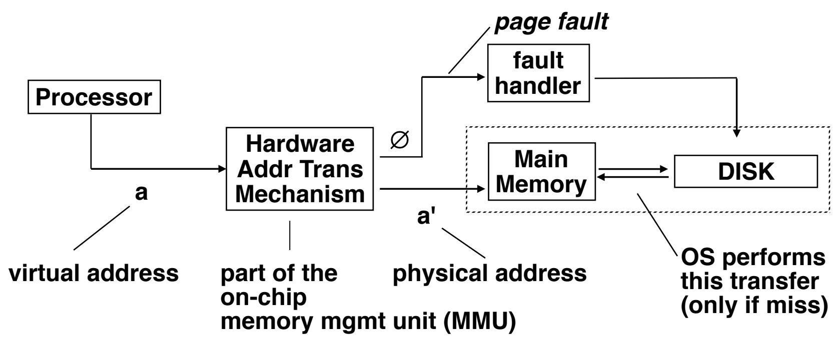





# VM Address Translation: Hit



# VM Address Translation: Miss



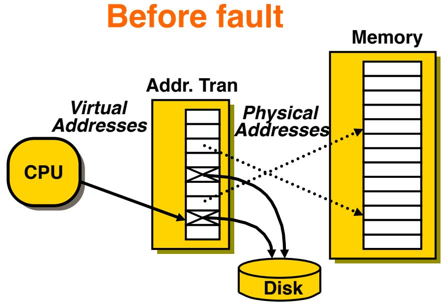



# Page Faults

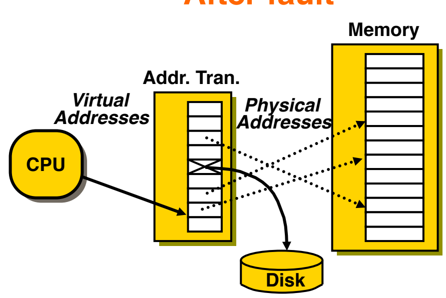
What if an object is on disk rather than in memory?

- Page table entry indicates virtual address not in memory
- OS exception handler invoked to move data from disk into memory
  - current process suspends, others can resume
  - OS has full control over placement, etc.

**Before fault**

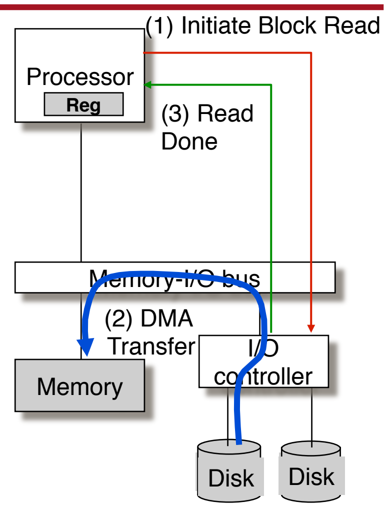



**After fault**



## Servicing a Page Fault

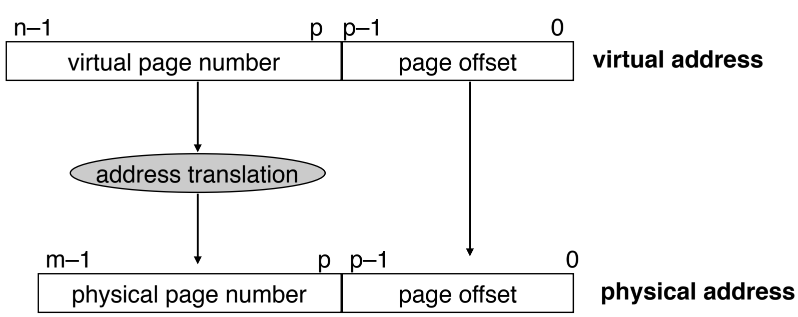
- Processor Signals Controller
  - Read block of length P starting at disk address X and store starting at memory address Y
- Read Occurs
  - Direct Memory Access (DMA)
  - Under control of I/O controller
- I / O Controller Signals Completion
  - Interrupt processor
  - OS resumes suspended process





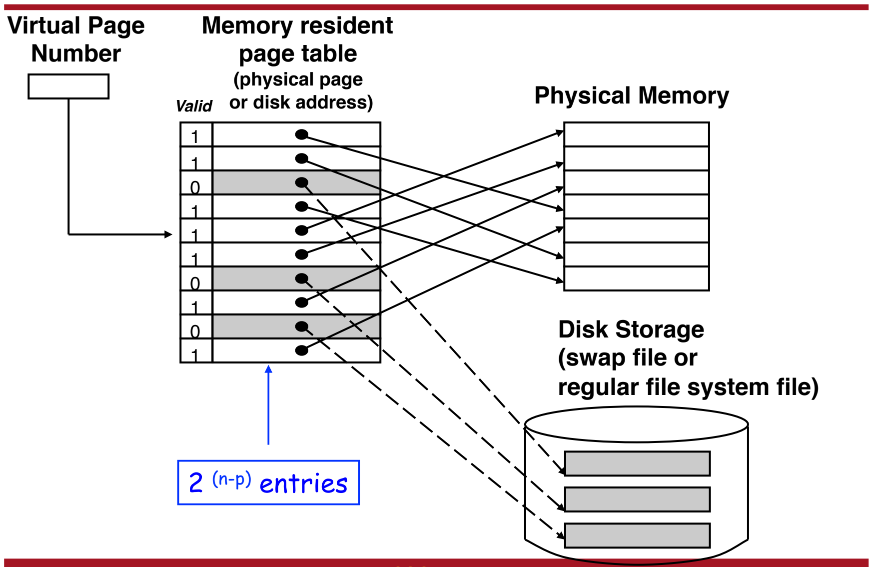

## VM Address Translation

- $P = 2^p$ = page size (bytes).
- $N = 2^n$ = Virtual address limit
- $M = 2^m$ = Physical address limit



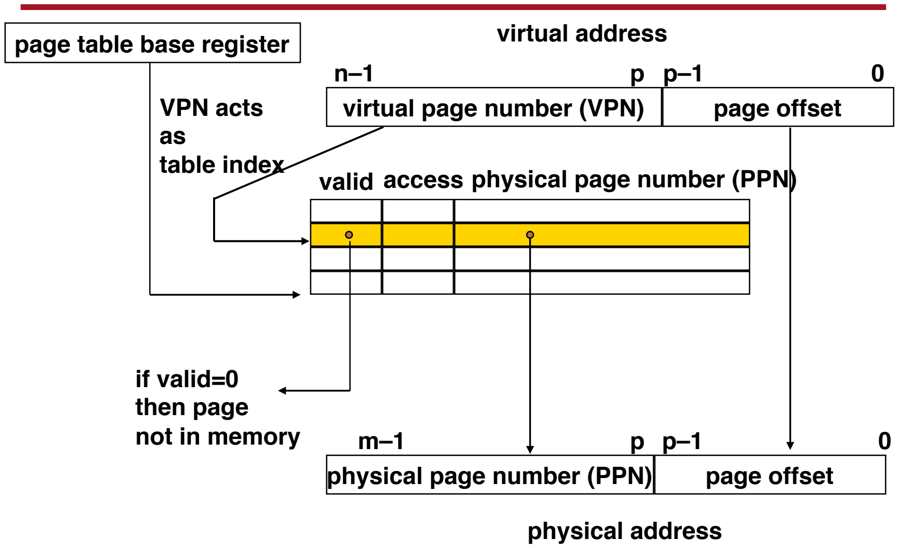

**Page offset bits don’t change as a result of translation**

## Page Tables

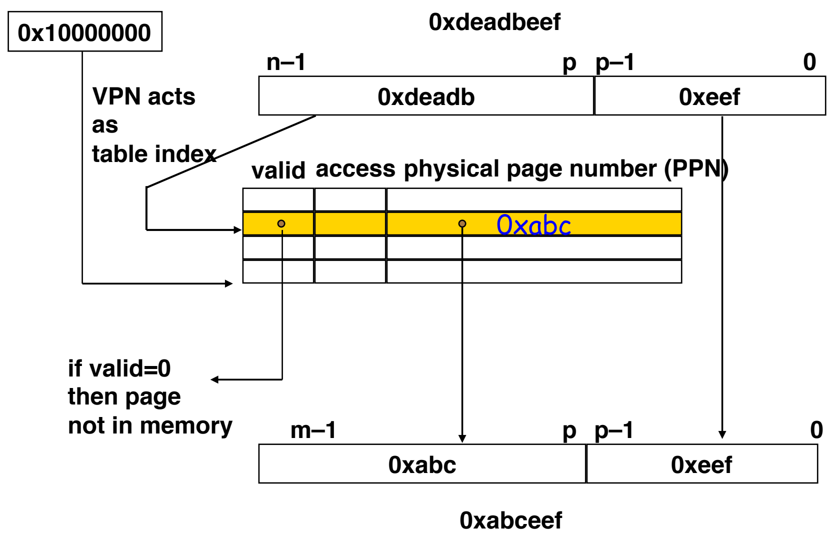





## Address Translation via Page Table

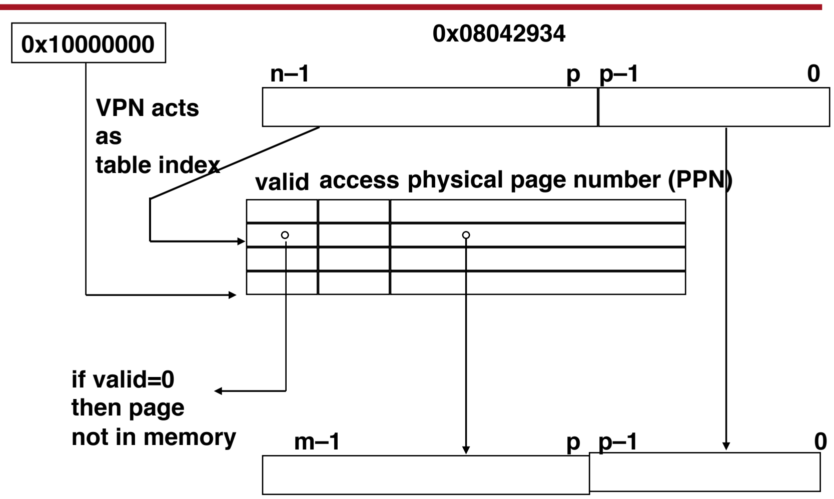


# Address Translation via Page Table






# Address Translation via Page Table



# Page Table Operation

Translation

- Separate (set of) page table(s) per process
- VPN forms index into page table (points to a page table entry)

Computing Physical Address

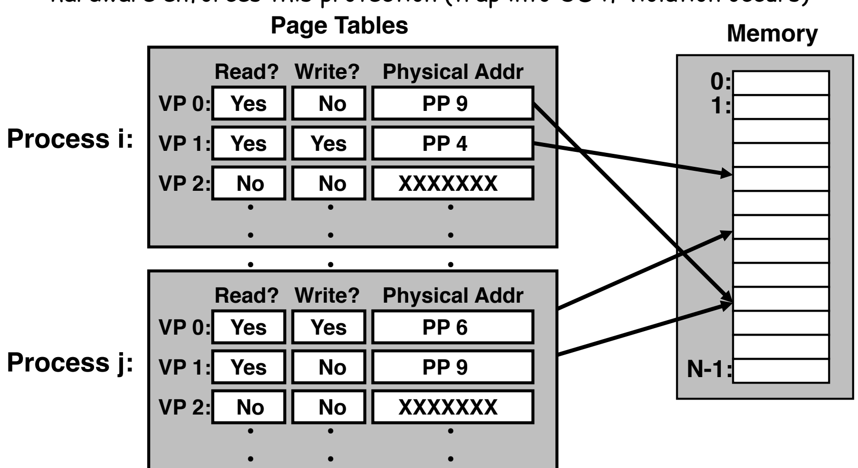
- Page Table Entry (PTE) provides information about page
  - if (valid bit = 1) then the page is in memory ➔Use physical page number (PPN) to construct address
  - if (valid bit = 0) then the page is on disk ➔ Page fault



# Protection

Page table entry contains access rights information

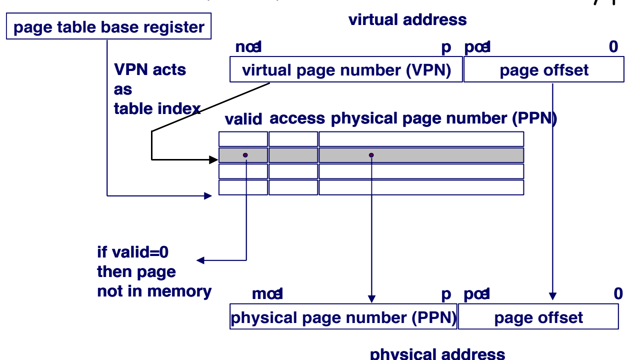
- hardware enforces this protection (trap into OS if violation occurs)



# Page Table Operation

Checking Protection

- Access rights field indicate allowable access
  - e.g., read-only, read-write, execute-only
  - typically support multiple protection modes (e.g., kernel vs. user)
- Protection violation fault if user doesn't have necessary permission






# Multi-Level Page Tables

Given:

- 4KB ($2^{12}$) page size
- 32-bit address space
- 4-byte PTE

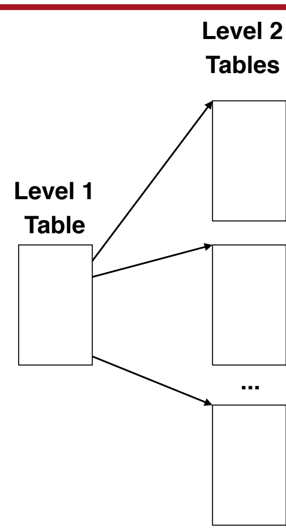
Problem:

- Would need a 4 MB page table!

➔$2^{20}$ *4 bytes

Common solution

- multi-level page tables
- e.g., 2-level table (P6)
  - Level 1 table: 1024 entries, each of which points to a Level 2 page table.
  - Level 2 table: 1024 entries, each of which points to a page



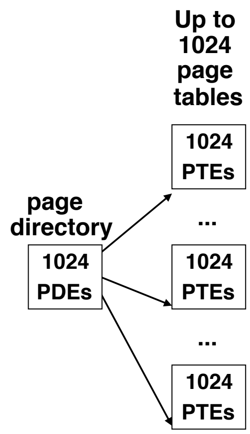
# An Example for 2 Level Page Table

Page directory

- 1024 4-byte page directory entries (PDEs) that point to page tables
- one page directory per process.
- page directory must be in memory when its process is running
- always pointed to by PDBR

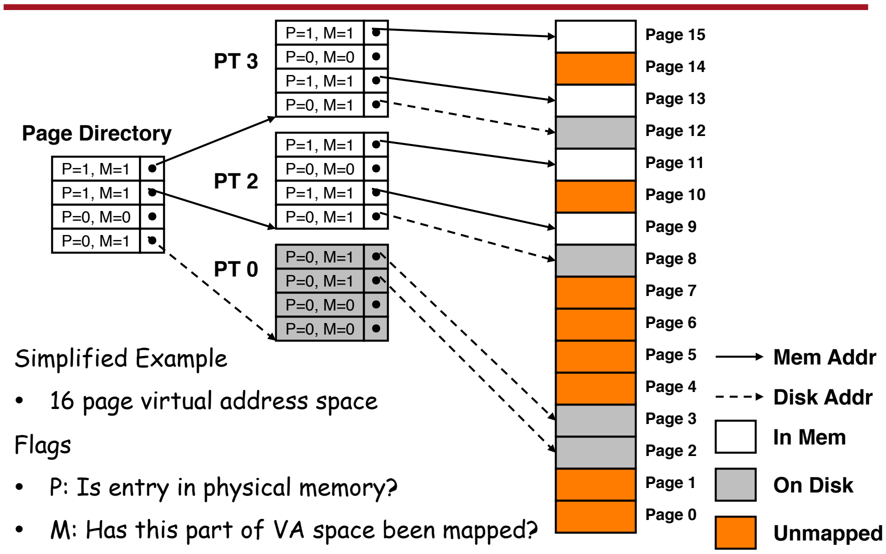
Page tables:

- 1024 4-byte page table entries (PTEs) that point to pages.
- page tables can be paged in and out.






# Representation of Virtual Address Space




Simplified Example

- 16 page virtual address space

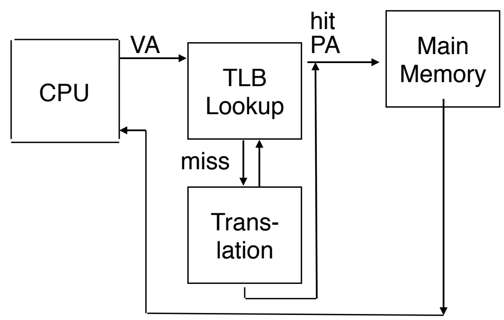
Flags

- P: Is entry in physical memory?
- M: Has this part of VA space been mapped?

## Speeding up Translation with a TLB

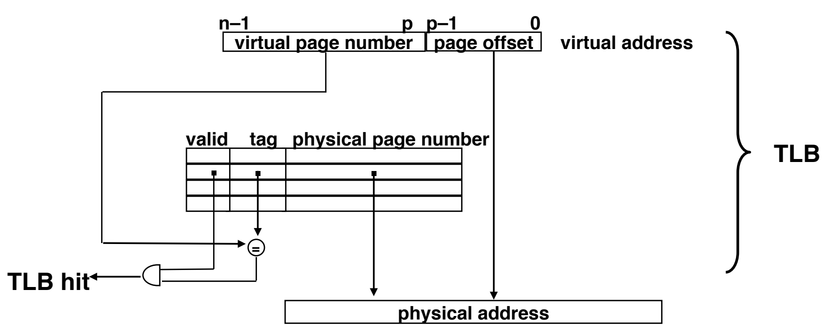
"Translation Lookaside Buffer" (TLB)

- Small hardware cache in MMU
- Maps virtual page numbers to physical page numbers
- Contains complete page table entries for small number of pages



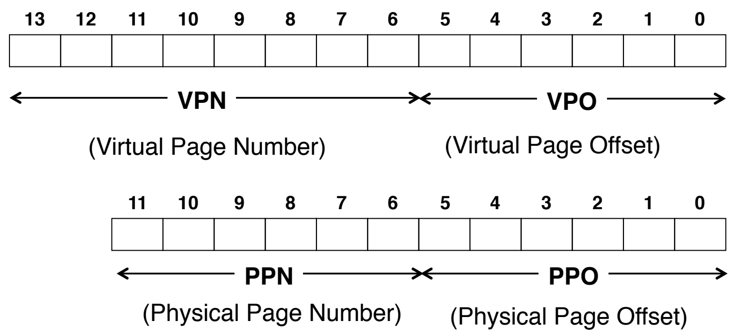



## Address Translation with a TLB



# Simple Memory System Example

Addressing
- 14-bit virtual addresses
- 12-bit physical address
- Page size = 64 bytes





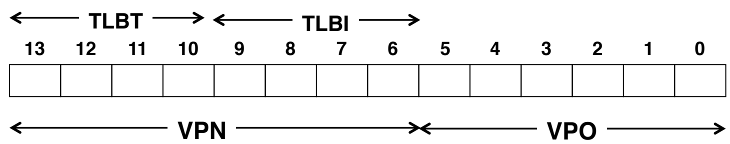

# Simple Memory System Page Table

Only show first 16 entries (256 entries total)

| VPN | PPN | Valid | VPN | PPN | Valid |
|---|---|---|---|---|---|
| 00 | 28 | 1 | 08 | 13 | 1 |
| 01 | – | 0 | 09 | 17 | 1 |
| 02 | 33 | 1 | 0A | 09 | 1 |
| 03 | 02 | 1 | 0B | – | 0 |
| 04 | – | 0 | 0C | – | 0 |
| 05 | 16 | 1 | 0D | 2D | 1 |
| 06 | – | 0 | 0E | 11 | 1 |
| 07 | – | 0 | 0F | 0D | 1 |

## Simple Memory System TLB

TLB
- 16 entries
- Direct mapped

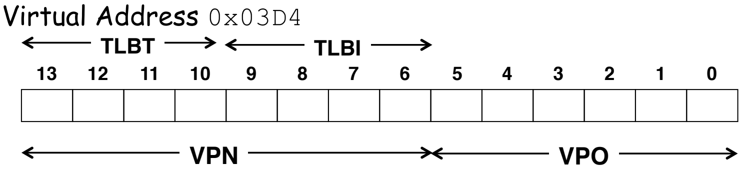



| Tag | PPN | Valid | Tag | PPN | Valid | Tag | PPN | Valid | Tag | PPN | Valid |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | – | 0 | 9 | 0D | 1 | 0 | – | 0 | 7 | 02 | 1 |
| 3 | 2D | 1 | 2 | – | 0 | 4 | – | 0 | A | – | 0 |
| 2 | – | 0 | 8 | – | 0 | 6 | – | 0 | 3 | – | 0 |
| 7 | – | 0 | 3 | 0D | 1 | A | 34 | 1 | 2 | – | 0 |



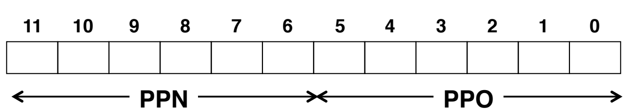
## Address Translation Example

- Virtual Address `0x03D4`




VPN ___ TLBI ___ TLBT ____ TLB Hit? __ Page Fault? __ PPN: ____

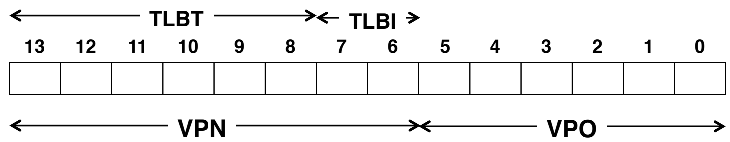
- Physical Address



# Simple Memory System TLB

TLB

- 16 entries
- 4-way associative

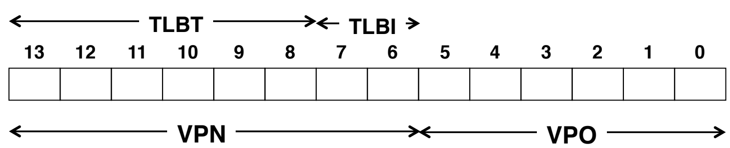



| Set | Tag | PPN | Valid | Tag | PPN | Valid | Tag | PPN | Valid | Tag | PPN | Valid |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 03 | – | 0 | 09 | 0D | 1 | 00 | – | 0 | 07 | 02 | 1 |
| 1 | 03 | 2D | 1 | 02 | – | 0 | 04 | – | 0 | 0A | – | 0 |
| 2 | 02 | – | 0 | 08 | – | 0 | 06 | – | 0 | 03 | – | 0 |
| 3 | 07 | – | 0 | 03 | 0D | 1 | 0A | 34 | 1 | 02 | – | 0 |



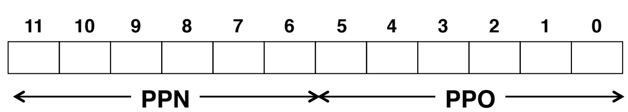
# Address Translation Example #1

- Virtual Address `0x03D4`



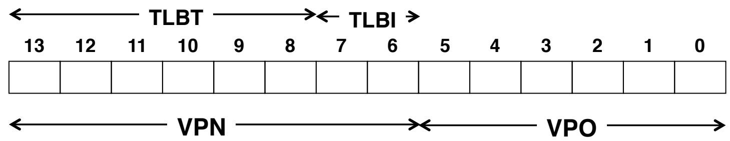

VPN ___ TLBI ___ TLBT ____ TLB Hit? __ Page Fault? __ PPN: ____

- Physical Address



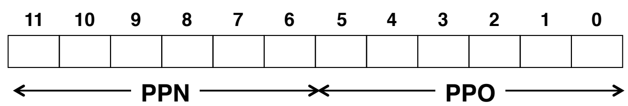
# Address Translation Example #2

- Virtual Address `0x0B8F`




VPN ___ TLBI ___ TLBT ____ TLB Hit? __ Page Fault? __ PPN: ____

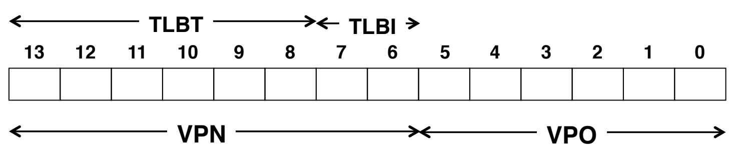
- Physical Address




4/8/15 CSCI 2021 33

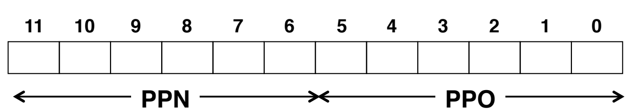
# Address Translation Example #3

- Virtual Address `0x0040`




VPN ___ TLBI ___ TLBT ____ TLB Hit? __ Page Fault? __ PPN: ____

- Physical Address




4/8/15 CSCI 2021 34

## Main Themes

- Programmer's View
  - Large "flat" address space
    - Can allocate large blocks of contiguous addresses
  - Processor "owns" machine
    - Has private address space
    - Unaffected by behavior of other processes
- System View
  - User virtual address space created by mapping to set of pages
    - Need not be contiguous
    - Allocated dynamically
    - Enforce protection during address translation
  - OS manages many processes simultaneously
    - Continually switching among processes
    - Especially when one must wait for resource
      - E.g., disk I/O to handle page fault



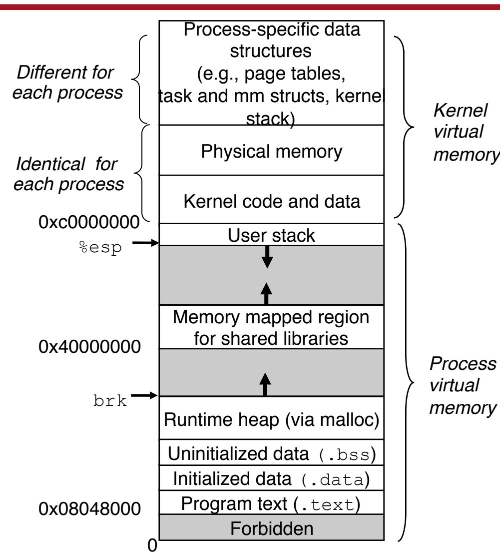
## Review of Abbreviations

- Symbols:
  - Components of the virtual address (VA)
    - TLBI: TLB index
    - TLBT: TLB tag
    - VPO: virtual page offset
    - VPN: virtual page number
  - Components of the physical address (PA)
    - PPO: physical page offset (same as VPO)
    - PPN: physical page number

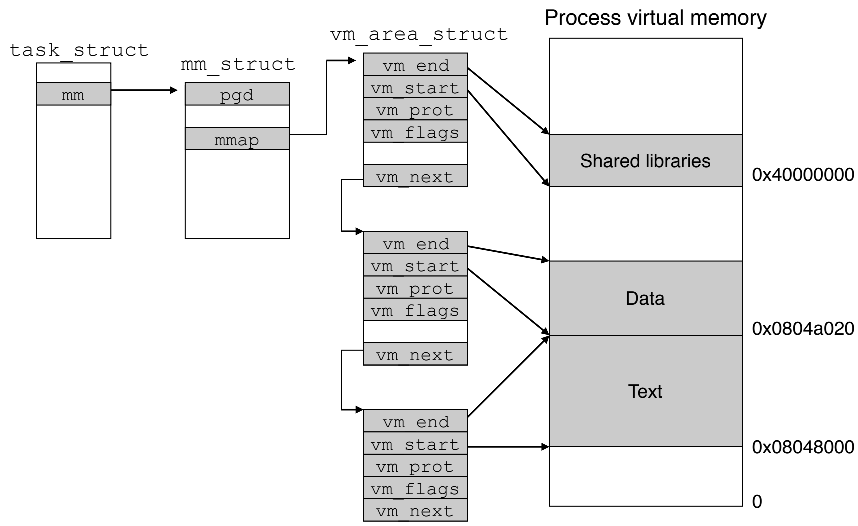
# Making Use of Virtual Memory



# Virtual Memory of a Linux Process



# How Linux Organize Virtual Memory






# Memory Mapping

Creation of new VM *area* done via "memory mapping"

- create new vm_area_struct and page tables for area
- area can be backed by (i.e., get its initial values from) :
  - regular file on disk (e.g., an executable object file)
    - initial page bytes come from a section of a file
  - nothing (e.g., bss)
    - initial page bytes are zeros
- dirty pages are swapped back and forth between a special swap file.

Key point: no virtual pages are copied into physical memory until they are referenced!

- known as "demand paging"
- crucial for time and space efficiency

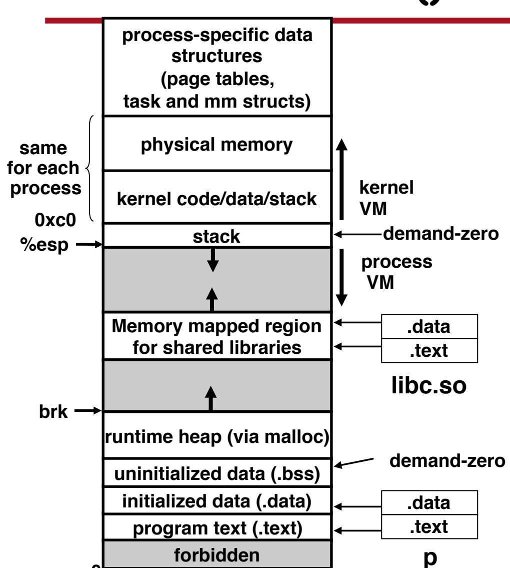
# Fork() Revisited

- Make copies of the old process's
  - mm_struct,
  - vm_area_struct's, and
  - page tables.

  At this point the two processes are sharing all of their pages. How to get separate spaces without copying all the virtual pages from one space to another?
- copy-on-write
  - make pages of writeable areas read-only
  - flag vm_area_struct's for these areas as private "copy-on-write".
  - writes by either process to these pages will cause page faults.
    - fault handler recognizes copy-on-write, makes a copy of the page, and restores write permissions.
- Net result:
  - copies are deferred until absolutely necessary (i.e., when one of the processes tries to modify a shared page).



# Exec() Revisited




To run a new program p in the current process using `exec()`:

- free vm_area_struct's and page tables for old areas.
- create new vm_area_struct's and page tables for new areas.
  - stack, bss, data, text, shared libs.
  - text and data backed by ELF executable object file.
  - bss and stack initialized to zero.
- set PC to entry point in .text
  - Linux will swap in code and data pages as needed.

# Memory System Summary

Virtual Memory

- Supports many OS-related functions
  - Process creation
    - Initial
    - Forking children
  - Task switching
  - Protection
- Combination of hardware & software implementation
  - Software management of tables, allocations
  - Hardware access of tables
  - Hardware caching of table entries (TLB)