CSCE 230J
Computer Organization

# Virtual Memory

## Dr. Steve Goddard
### goddard@cse.unl.edu

*http://cse.unl.edu/~goddard/Courses/CSCE230J*

# Giving credit where credit is due

- Most of slides for this lecture are based on slides created by Drs. Bryant and O'Hallaron, Carnegie Mellon University.
- I have modified them and added new slides.

2

# Topics

- **Motivations for VM**
- **Address translation**
- **Accelerating translation with TLBs**

# Motivations for Virtual Memory

**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

**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

**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
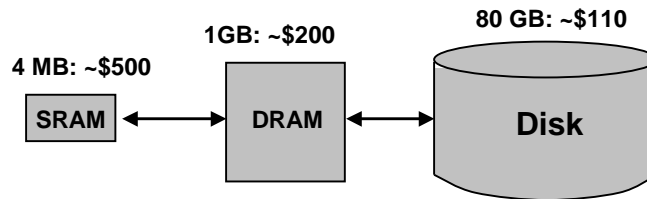
# Motivation #1: DRAM a "Cache" for Disk

**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**

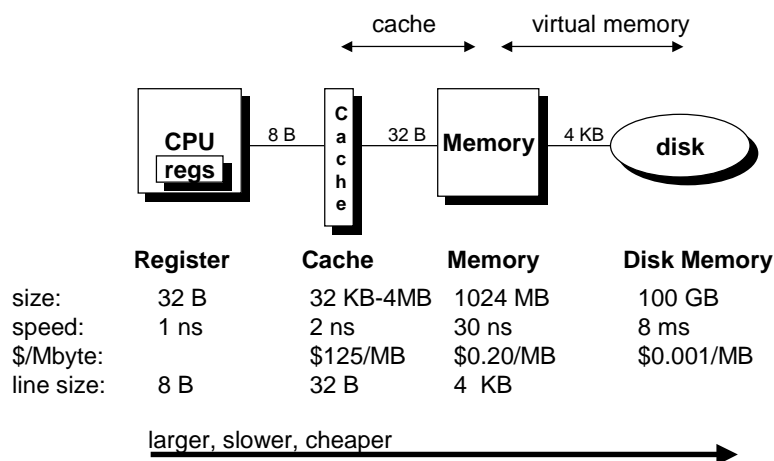**Disk storage is ~300X cheaper than DRAM storage**

- **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**

**80 GB: ~$110**

**1GB: ~$200**

**4 MB: ~$500**

**SRAM** ⟷ **DRAM** ⟷ **Disk**

# Levels in Memory Hierarchy

cache          virtual memory

**CPU**
**regs**  8 B  **C a c h e**  32 B  **Memory**  4 KB  **disk**

| | Register | Cache | Memory | Disk Memory |
|---|---|---|---|---|
| size: | 32 B | 32 KB-4MB | 1024 MB | 100 GB |
| speed: | 1 ns | 2 ns | 30 ns | 8 ms |
| $/Mbyte: | | $125/MB | $0.20/MB | $0.001/MB |
| line size: | 8 B | 32 B | 4 KB | |

larger, slower, cheaper →

# DRAM vs. SRAM as a "Cache"

**DRAM vs. disk is more extreme than SRAM vs. DRAM**

- **Access latencies:**
  - DRAM ~10X slower than SRAM
  - Disk ~100,000X slower than DRAM
- **Importance of exploiting spatial locality:**
  - First byte is ~100,000X slower than successive bytes on disk
    - » vs. ~4X improvement for page-mode vs. regular accesses to DRAM
- **Bottom line:**
  - Design decisions made for DRAM caches driven by enormous cost of misses

SRAM ⟷ DRAM ⟷ Disk
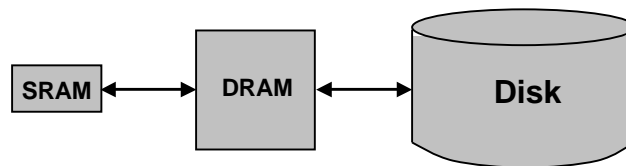
# Impact of Properties on Design

**If DRAM was to be organized similar to an SRAM cache, how would we set the following design parameters?**

- **Line size?**
  - Large, since disk better at transferring large blocks
- **Associativity?**
  - High, to mimimize miss rate
- **Write through or write back?**
  - Write back, since can't afford to perform small writes to disk

**What would the impact of these choices be on:**

- **miss rate**
  - Extremely low.  << 1%
- **hit time**
  - Must match cache/DRAM performance
- **miss latency**
  - Very high.  ~20ms
- **tag storage overhead**
  - Low, relative to block size

# Locating an Object in a "Cache"

**SRAM Cache**

- **Tag stored with cache line**
- **Maps from cache block to memory blocks**
  - From cached to uncached form
  - Save a few bits by only storing tag
- **No tag for block not in cache**
- **Hardware retrieves information**
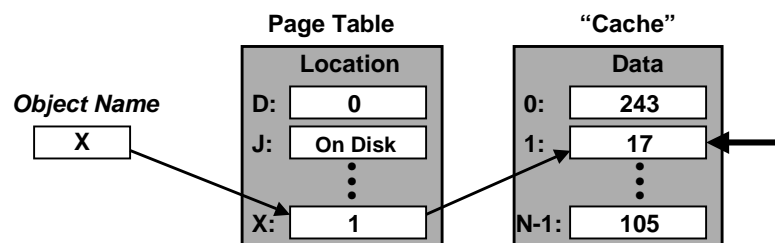  - can quickly match against multiple tags

"Cache"

| | Tag | Data |
|---|---|---|
| 0: | D | 243 |
| 1: | X | 17 |
| ⋮ | ⋮ | ⋮ |
| N-1: | J | 105 |

*Object Name*

| X |
|---|

= X?

---
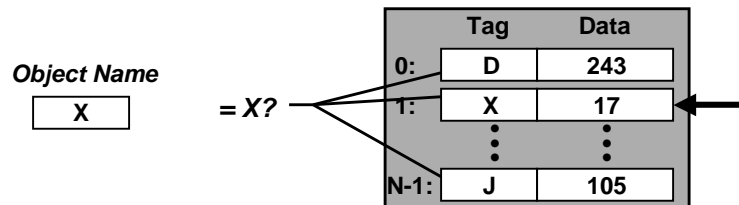
# Locating an Object in "Cache" (cont.)

**DRAM Cache**

- **Each allocated page of virtual memory has entry in *page table***
- **Mapping from virtual pages to physical pages**
  - From uncached form to cached form
- **Page table entry even if page not in memory**
  - Specifies disk address
  - Only way to indicate where to find page
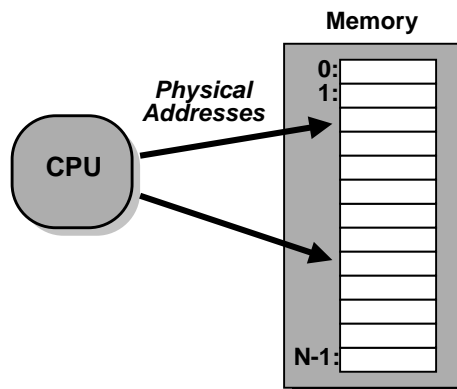- **OS retrieves information**

**Page Table**

| | Location |
|---|---|
| D: | 0 |
| J: | On Disk |
| ⋮ | ⋮ |
| X: | 1 |

"Cache"

| | Data |
|---|---|
| 0: | 243 |
| 1: | 17 |
| ⋮ | ⋮ |
| N-1: | 105 |

*Object Name*

| X |
|---|

# A System with Physical Memory Only

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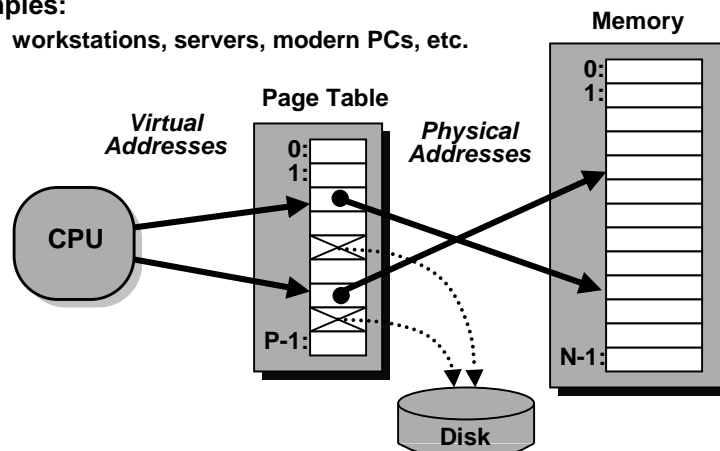
**Memory**

*Physical Addresses*

CPU

0:
1:

N-1:

- **Addresses generated by the CPU correspond directly to bytes in physical memory**

11

# A System with Virtual Memory

**Examples:**
- workstations, servers, modern PCs, etc.

**Memory**

**Page Table**

*Virtual Addresses*

*Physical Addresses*

CPU

0:
1:

P-1:

0:
1:

N-1:

Disk

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

12

# Page Faults (like "Cache Misses")

**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**

Memory

Page Table

*Virtual Addresses*     *Physical Addresses*

CPU

Disk

**After fault**

Memory

Page Table

*Virtual Addresses*     *Physical Addresses*

CPU
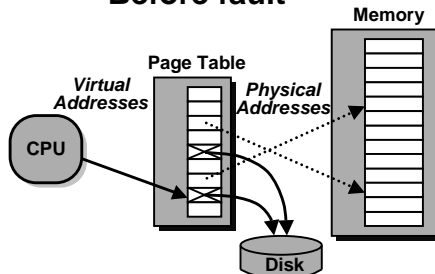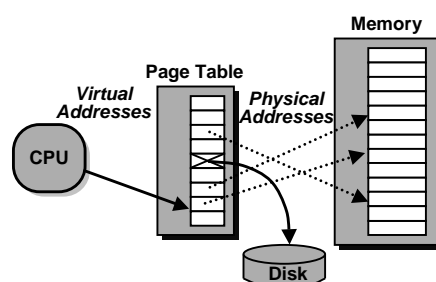
Disk

13

---

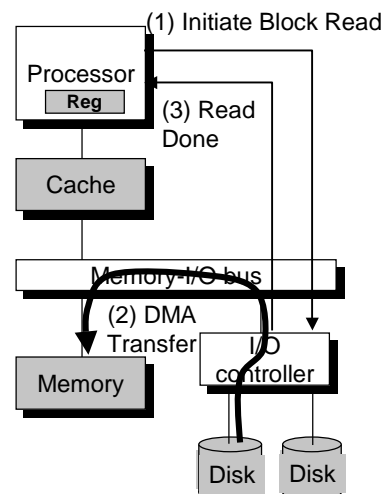# Servicing a Page Fault

**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**

(1) Initiate Block Read

Processor
Reg

(3) Read Done

Cache

Memory-I/O bus

(2) DMA Transfer

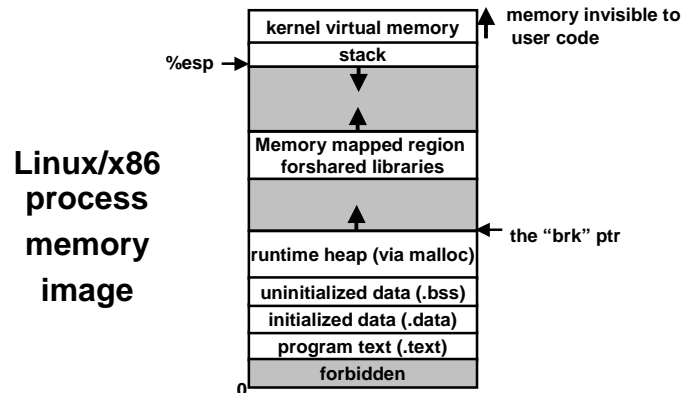Memory

I/O controller

Disk    Disk

14

# Motivation #2: Memory Management

**Multiple processes can reside in physical memory.**

**How do we resolve address conflicts?**

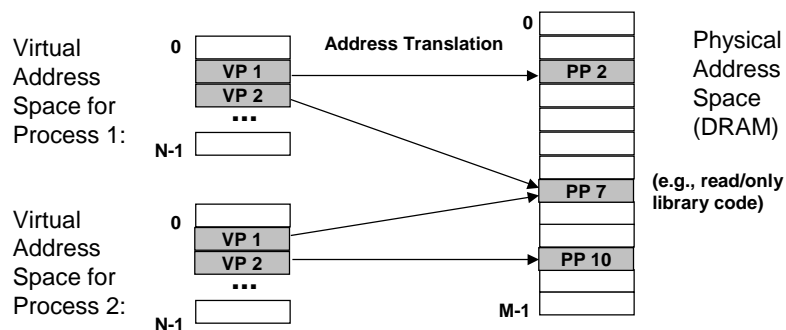- what if two processes access something at the same address?

**Linux/x86 process memory image**

| | |
|---|---|
| kernel virtual memory | memory invisible to user code |
| stack | ← %esp |
| Memory mapped region for shared libraries | |
| runtime heap (via malloc) | ← the "brk" ptr |
| uninitialized data (.bss) | |
| initialized data (.data) | |
| program text (.text) | |
| forbidden | |

0

15

# Solution: Separate Virt. Addr. 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

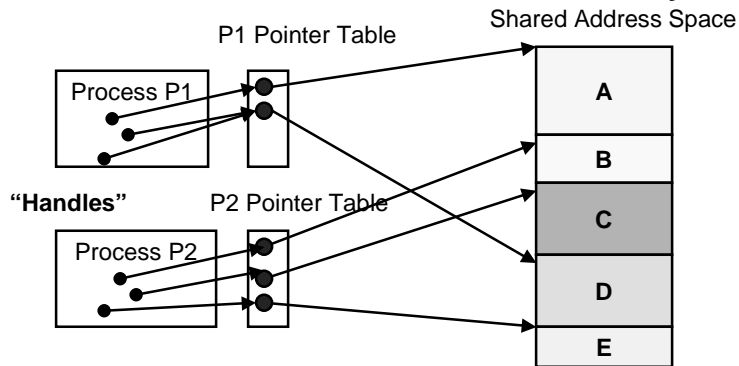Virtual Address Space for Process 1:
- 0
- VP 1
- VP 2
- ...
- N-1

**Address Translation**

Virtual Address Space for Process 2:
- 0
- VP 1
- VP 2
- ...
- N-1

Physical Address Space (DRAM)
- 0
- PP 2
- PP 7 — (e.g., read/only library code)
- PP 10
- M-1

16

# Contrast: Macintosh Memory Model

**MAC OS 1–9**

■ **Does not use traditional virtual memory**

P1 Pointer Table · Shared Address Space

Process P1

"Handles" · P2 Pointer Table

Process P2

| A |
| B |
| C |
| D |
| E |

**All program objects accessed through "handles"**
- Indirect reference through pointer table
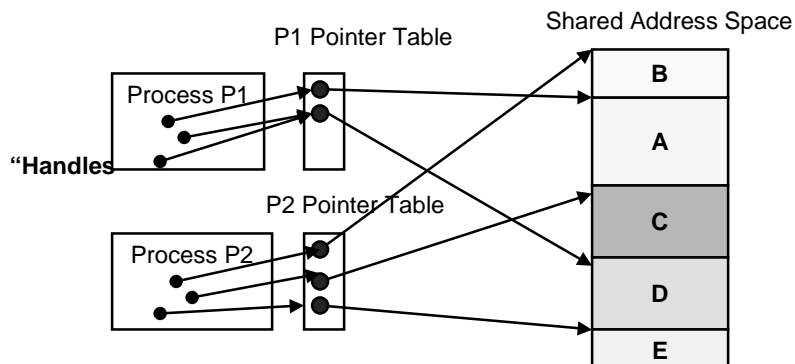- Objects stored in shared global address space

17

# Macintosh Memory Management

**Allocation / Deallocation**
- Similar to free-list management of malloc/free

**Compaction**
- Can move any object and just update the (unique) pointer in pointer table

P1 Pointer Table · Shared Address Space

Process P1

"Handles" · P2 Pointer Table

Process P2

| B |
| A |
| C |
| D |
| E |

18

# Mac vs. VM-Based Memory Mgmt

**Allocating, deallocating, and moving memory:**

- can be accomplished by both techniques

**Block sizes:**

- Mac: variable-sized
  - may be very small or very large
- VM: fixed-size
  - size is equal to *one page* (4KB on x86 Linux systems)

**Allocating contiguous chunks of memory:**

- Mac: contiguous allocation is *required*
- VM: can map contiguous range of virtual addresses to disjoint ranges of physical addresses

**Protection**

- Mac: "wild write" by one process can corrupt another's data

19

# MAC OS X

**"Modern" Operating System**

- Virtual memory with protection
- *Preemptive multitasking*
  - Other versions of MAC OS require processes to voluntarily relinquish control
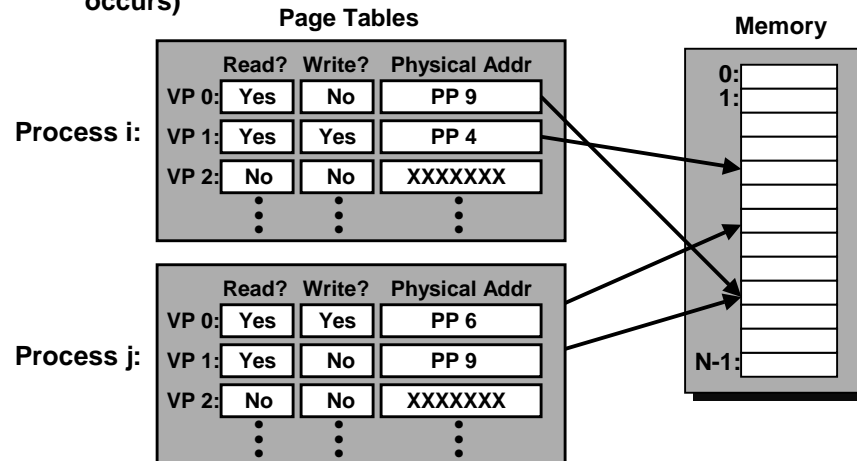
**Based on MACH OS**

- Developed at CMU in late 1980's

20

# Motivation #3: Protection

**Page table entry contains access rights information**

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

**Page Tables**

**Memory**

Process i:

| | Read? | Write? | Physical Addr |
|---|---|---|---|
| VP 0: | Yes | No | PP 9 |
| VP 1: | Yes | Yes | PP 4 |
| VP 2: | No | No | XXXXXXX |

Process j:

| | Read? | Write? | Physical Addr |
|---|---|---|---|
| VP 0: | Yes | Yes | PP 6 |
| VP 1: | Yes | No | PP 9 |
| VP 2: | No | No | XXXXXXX |

Memory: 0: 1: ... N-1:

---

# VM Address Translation

**Virtual Address Space**

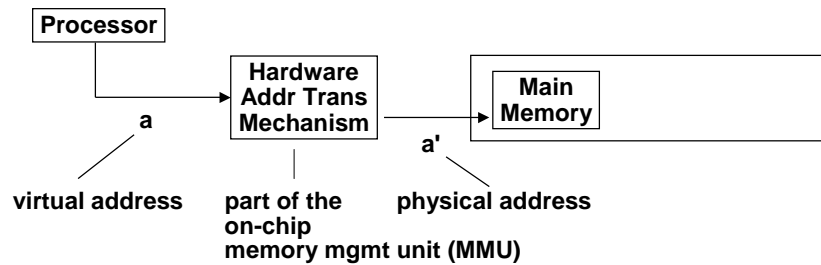- $V = \{0, 1, \ldots, N-1\}$

**Physical Address Space**

- $P = \{0, 1, \ldots, M-1\}$
- $M < N$

**Address Translation**

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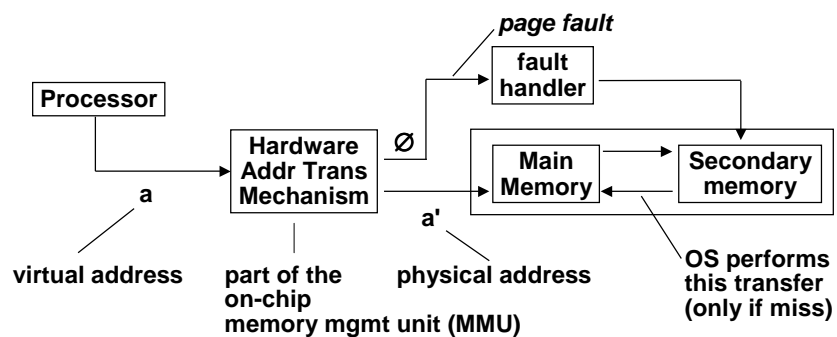    - » Either invalid or stored on disk

# VM Address Translation: Hit

**Processor**

a

**virtual address**

**Hardware Addr Trans Mechanism**

**part of the on-chip memory mgmt unit (MMU)**

a'

**physical address**

**Main Memory**

23

# VM Address Translation: Miss

*page fault*

**fault handler**

**Processor**

a

**virtual address**

**Hardware Addr Trans Mechanism**

**part of the on-chip memory mgmt unit (MMU)**

Ø

a'

**physical address**

**Main Memory**

**Secondary memory**

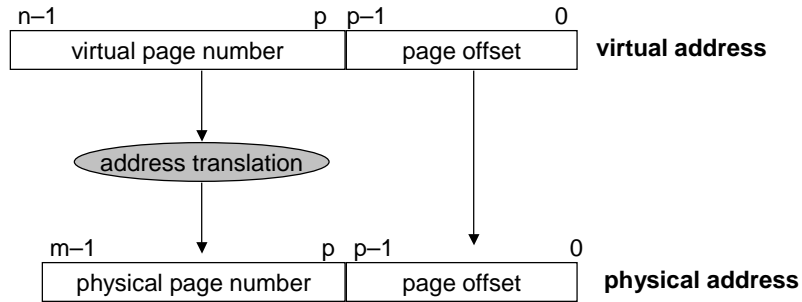**OS performs this transfer (only if miss)**

24

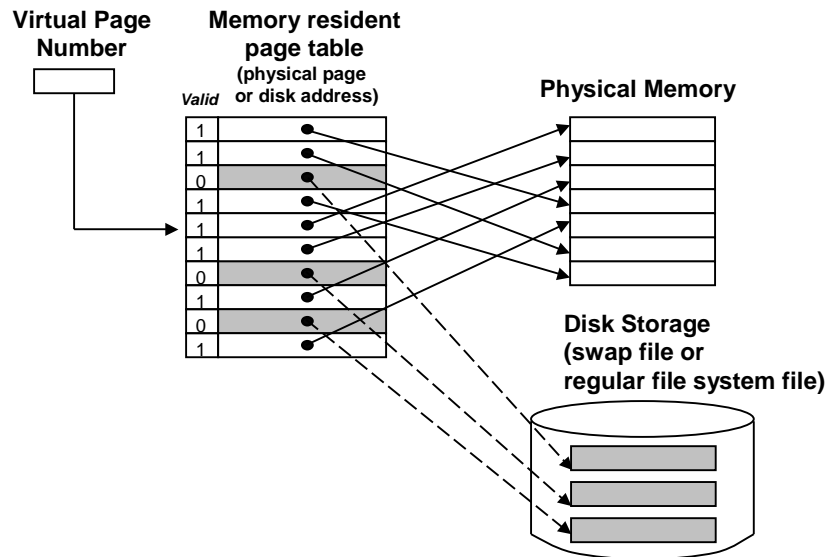# VM Address Translation

**Parameters**

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



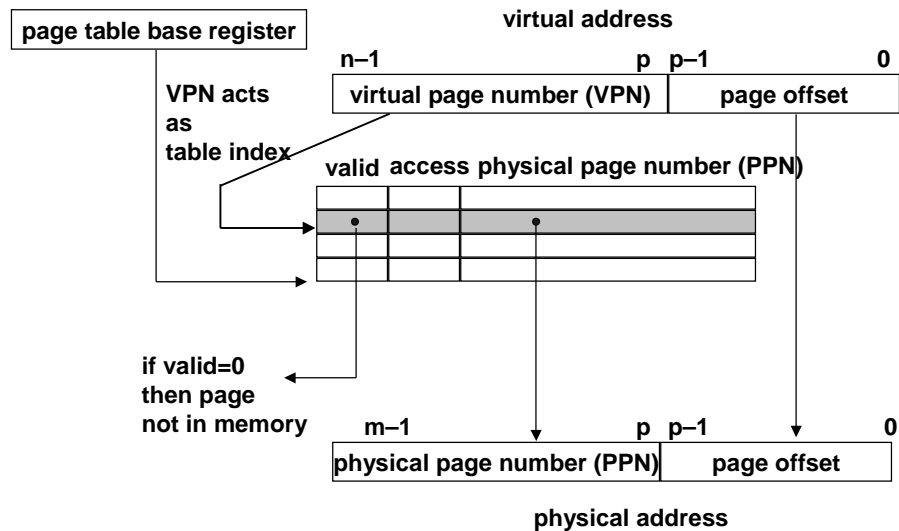**Page offset bits don't change as a result of translation**
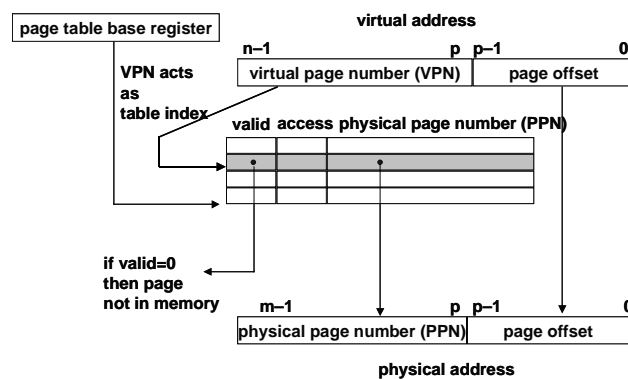
# Page Tables

# Address Translation via Page Table

**page table base register**

**virtual address**

**VPN acts as table index**

| n–1 | p | p–1 | 0 |
|---|---|---|---|
| **virtual page number (VPN)** | | **page offset** | |

**valid access physical page number (PPN)**

**if valid=0 then page not in memory**

| m–1 | p | p–1 | 0 |
|---|---|---|---|
| **physical page number (PPN)** | | **page offset** | |

**physical address**

---

# Page Table Operation

**Translation**

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

**page table base register**

**virtual address**

**VPN acts as table index**

| n–1 | p | p–1 | 0 |
|---|---|---|---|
| **virtual page number (VPN)** | | **page offset** | |

**valid access physical page number (PPN)**

**if valid=0 then page not in memory**

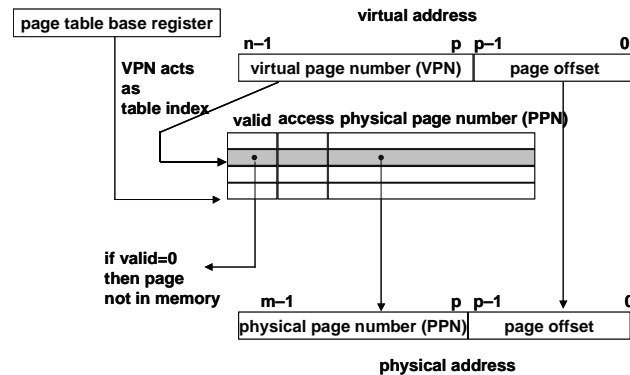| m–1 | p | p–1 | 0 |
|---|---|---|---|
| **physical page number (PPN)** | | **page offset** | |

**physical address**

# Page Table Operation

**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**
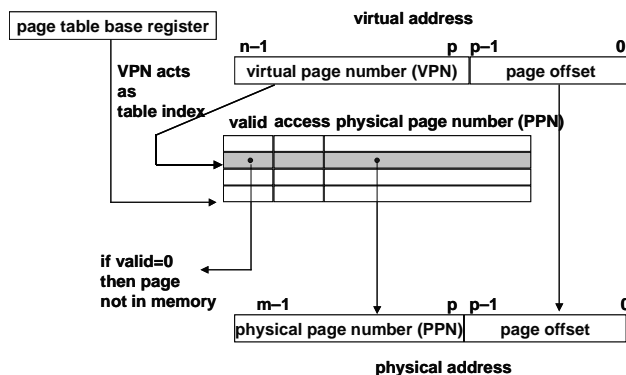  - **if (valid bit = 0) then the page is on disk**
    - » **Page fault**

page table base register

virtual address

n−1                    p  p−1              0

virtual page number (VPN) | page offset

VPN acts as table index

valid  access  physical page number (PPN)

if valid=0 then page not in memory

m−1              p  p−1              0

physical page number (PPN) | page offset

physical address

29

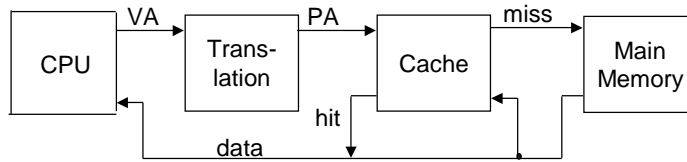---

# 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**

page table base register

virtual address

n−1                    p  p−1              0

virtual page number (VPN) | page offset

VPN acts as table index

valid  access  physical page number (PPN)

if valid=0 then page not in memory

m−1              p  p−1              0

physical page number (PPN) | page offset

physical address

30

# Integrating VM and Cache



## Most Caches "Physically Addressed"

- Accessed by physical addresses
- Allows multiple processes to have blocks in cache at same time
- Allows multiple processes to share pages
- Cache doesn't need to be concerned with protection issues
  - Access rights checked as part of address translation

## Perform Address Translation Before Cache Lookup

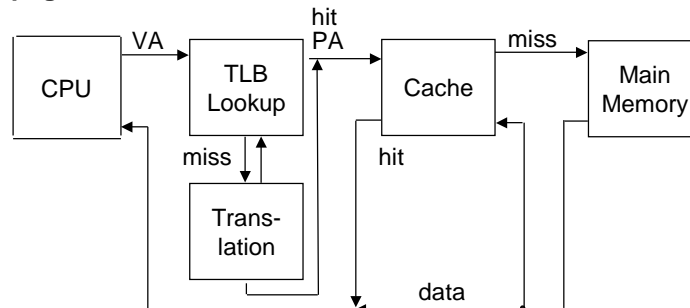- But this could involve a memory access itself (of the PTE)
- Of course, page table entries can also become cached

31

# Speeding up Translation with a TLB
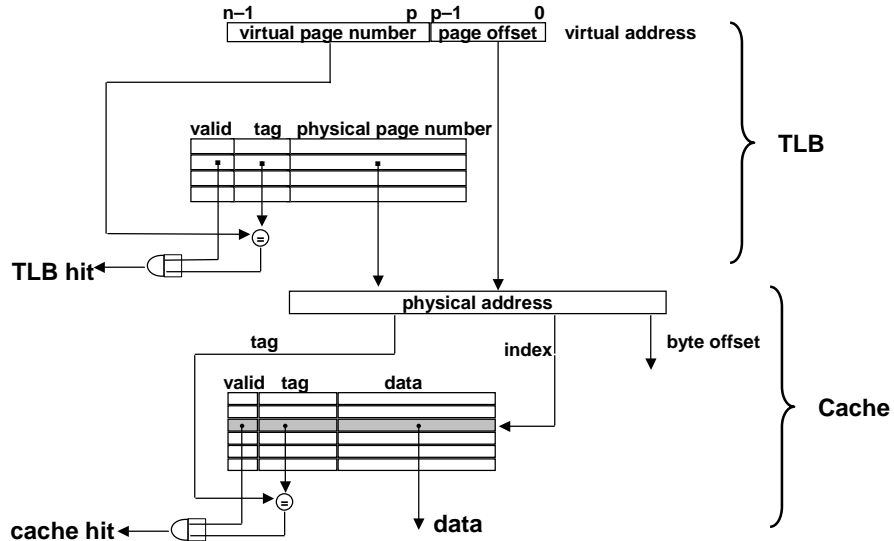
## "Translation Lookaside Buffer" (TLB)

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



32
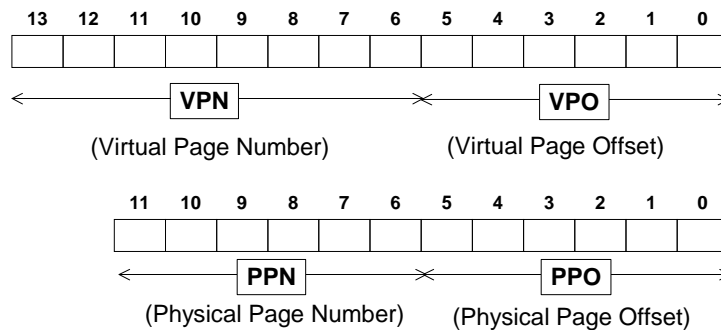
Page 16

# Address Translation with a TLB

# Simple Memory System Example

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

Page 17

# Simple Memory System Page Table
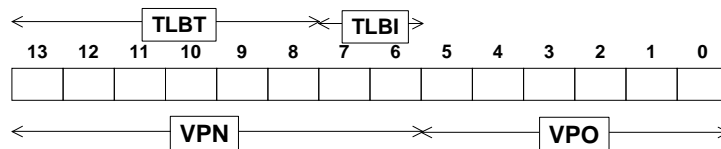
■ **Only show first 16 entries**

| 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 |

35

---

# Simple Memory System TLB

**TLB**

■ **16 entries**

■ **4-way associative**

| | | | TLBT | | | TLBI | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | |

VPN ⟷ VPO

| 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 |

36

# Simple Memory System Cache

**Cache**

- **16 lines**
- **4-byte line size**
- **Direct mapped**

CT ⟷ CI ⟷ CO

| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

PPN ⟷ PPO
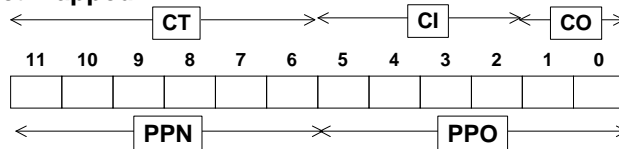
| Idx | Tag | Valid | B0 | B1 | B2 | B3 | Idx | Tag | Valid | B0 | B1 | B2 | B3 |
|-----|-----|-------|----|----|----|----|-----|-----|-------|----|----|----|----|
| 0 | 19 | 1 | 99 | 11 | 23 | 11 | 8 | 24 | 1 | 3A | 00 | 51 | 89 |
| 1 | 15 | 0 | – | – | – | – | 9 | 2D | 0 | – | – | – | – |
| 2 | 1B | 1 | 00 | 02 | 04 | 08 | A | 2D | 1 | 93 | 15 | DA | 3B |
| 3 | 36 | 0 | – | – | – | – | B | 0B | 0 | – | – | – | – |
| 4 | 32 | 1 | 43 | 6D | 8F | 09 | C | 12 | 0 | – | – | – | – |
| 5 | 0D | 1 | 36 | 72 | F0 | 1D | D | 16 | 1 | 04 | 96 | 34 | 15 |
| 6 | 31 | 0 | – | – | – | – | E | 13 | 1 | 83 | 77 | 1B | D3 |
| 7 | 16 | 1 | 11 | C2 | DF | 03 | F | 14 | 0 | – | – | – | – |

# Address Translation Example #1

**Virtual Address `0x03D4`**
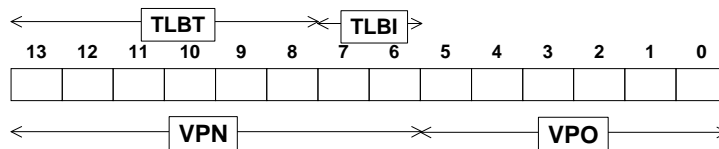
TLBT ⟷ TLBI

| 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

VPN ⟷ VPO

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

**Physical Address**

CT ⟷ CI ⟷ CO

| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

PPN ⟷ PPO
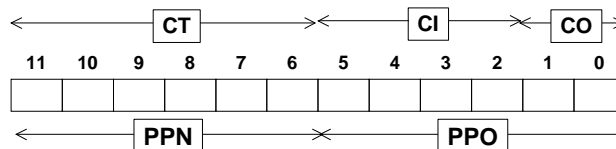
Offset ___ CI___ CT ____ Hit? __ Byte: ____

# Address Translation Example #2

**Virtual Address** `0x0B8F`
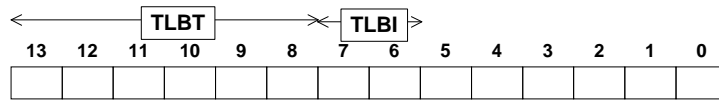
| | TLBT | | | TLBI | |
|---|---|---|---|---|---|

| 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | |

|←——— VPN ———→|←——— VPO ———→|

**VPN** \_\_\_  **TLBI** \_\_\_  **TLBT** \_\_\_\_  **TLB Hit?** \_\_  **Page Fault?** \_\_  **PPN:** \_\_\_\_

**Physical Address**

| | CT | | CI | | CO |
|---|---|---|---|---|---|

| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |

|←——— PPN ———→|←——— PPO ———→|

**Offset** \_\_\_  **CI** \_\_\_  **CT** \_\_\_\_  **Hit?** \_\_  **Byte:** \_\_\_\_

---
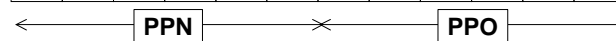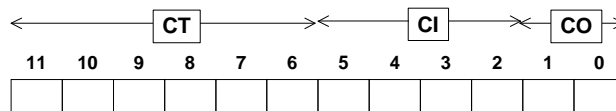
# Address Translation Example #3

**Virtual Address** `0x0040`
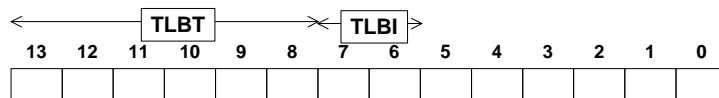
| | TLBT | | | TLBI | |
|---|---|---|---|---|---|

| 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | |

|←——— VPN ———→|←——— VPO ———→|

**VPN** \_\_\_  **TLBI** \_\_\_  **TLBT** \_\_\_\_  **TLB Hit?** \_\_  **Page Fault?** \_\_  **PPN:** \_\_\_\_

**Physical Address**

| | CT | | CI | | CO |
|---|---|---|---|---|---|

| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |

|←——— PPN ———→|←——— PPO ———→|
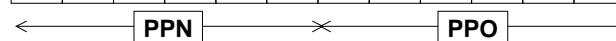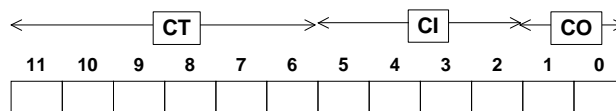
**Offset** \_\_\_  **CI** \_\_\_  **CT** \_\_\_\_  **Hit?** \_\_  **Byte:** \_\_\_\_

# 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**

**Level 2 Tables**

**Level 1 Table**

**...**
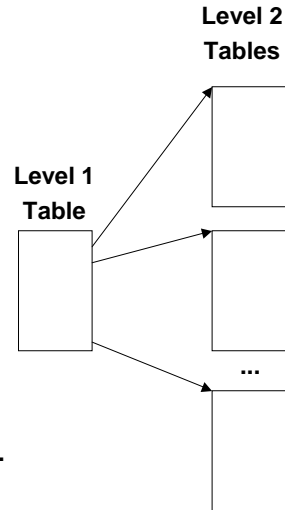
# Main Themes

**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**