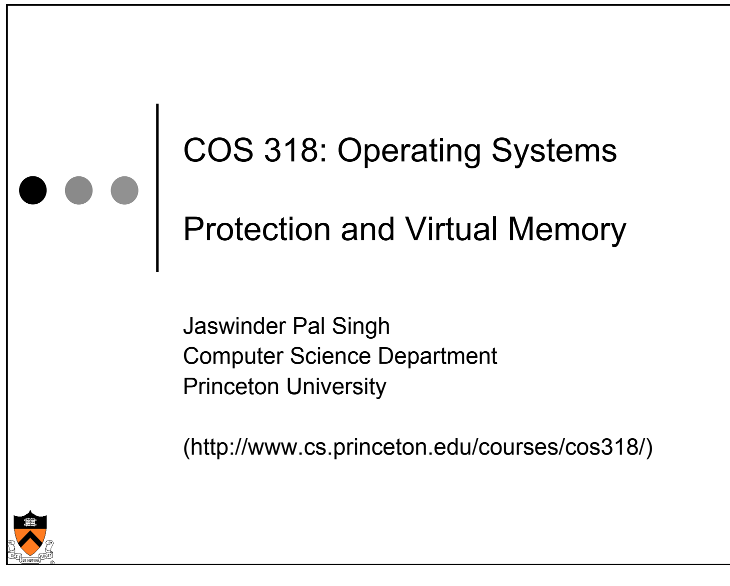
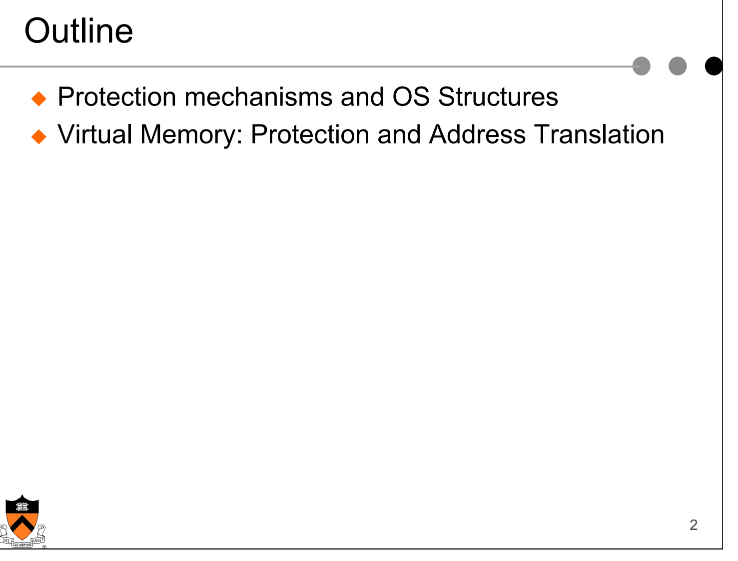
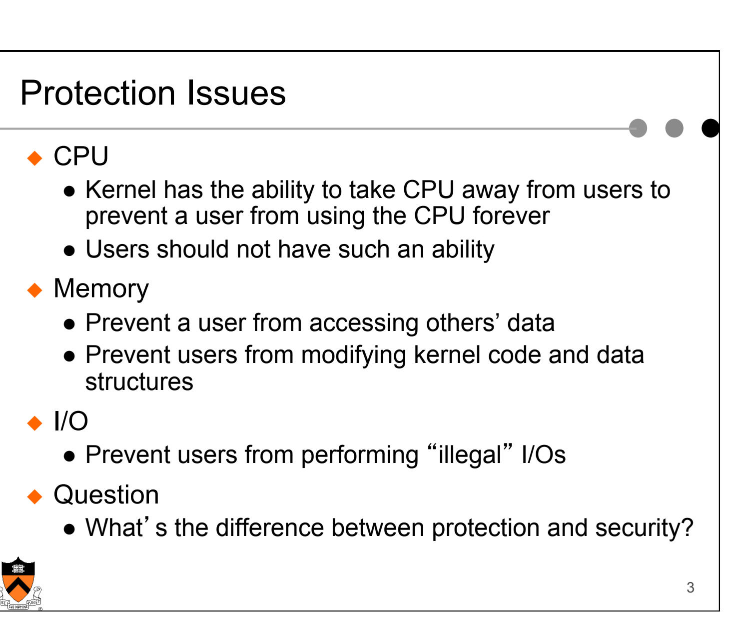
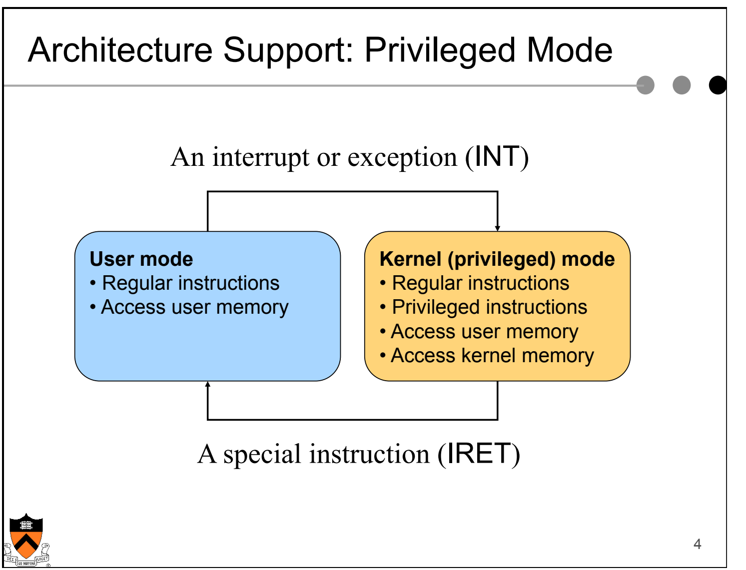

COS 318: Operating Systems
Protection and Virtual Memory
Jaswinder Pal Singh
Computer Science Department
Princeton University
(http://www.cs.princeton.edu/courses/cos318/)


Outline
Protection mechanisms and OS Structures
Virtual Memory: Protection and Address Translation
2

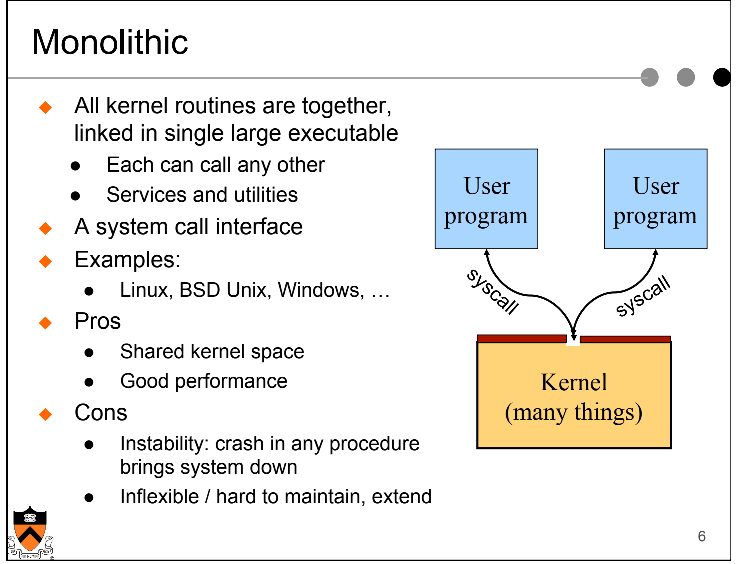

Protection Issues
CPU
Kernel has the ability to take CPU away from users to prevent a user from using the CPU forever
Users should not have such an ability
Memory
Prevent a user from accessing others' data
Prevent users from modifying kernel code and data structures
I/O
Prevent users from performing "illegal" I/Os
Question
What's the difference between protection and security?
3

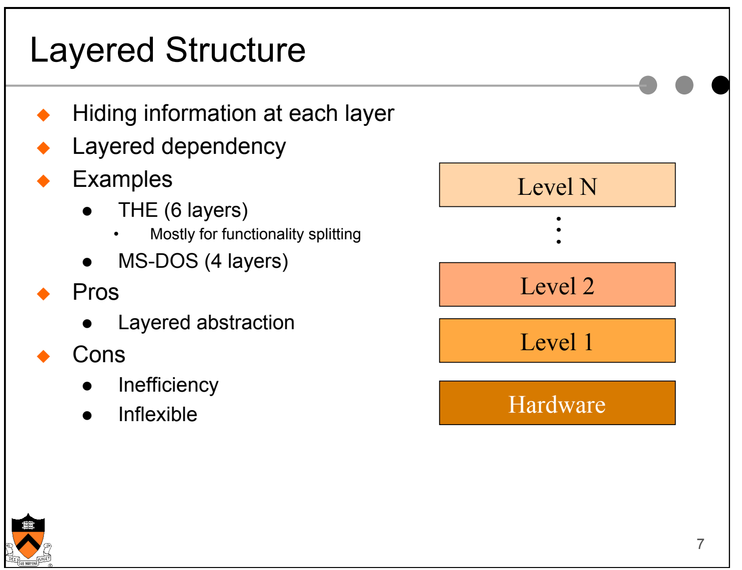

Architecture Support: Privileged Mode
An interrupt or exception (INT)
User mode
• Regular instructions
• Access user memory
Kernel (privileged) mode
• Regular instructions
• Privileged instructions
• Access user memory
• Access kernel memory
A special instruction (IRET)
4

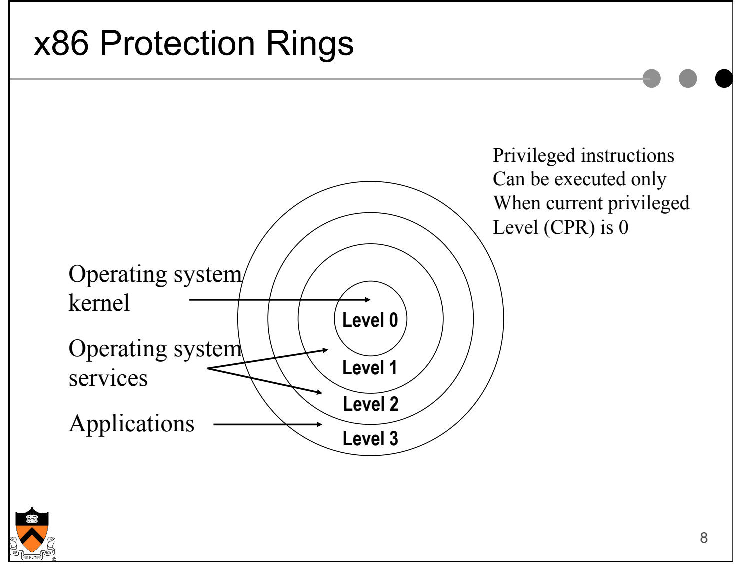
Privileged Instruction Examples

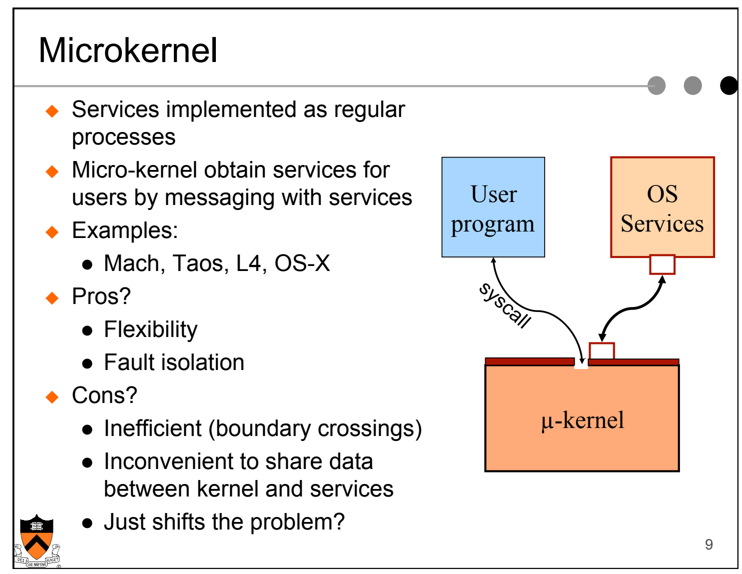
- Memory address mapping
- Flush or invalidate data cache
- Invalidate TLB entries
- Load and read system registers
- Change processor modes from kernel to user
- Change the voltage and frequency of processor
- Halt a processor
- Reset a processor
- Perform I/O operations

5

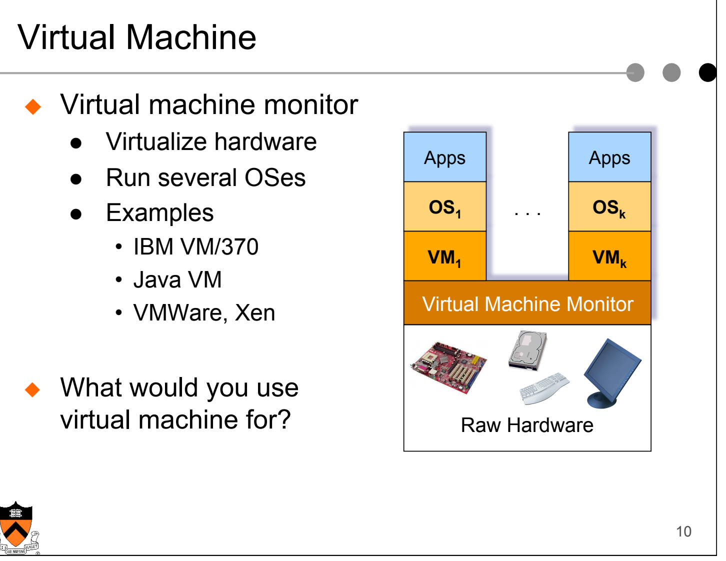




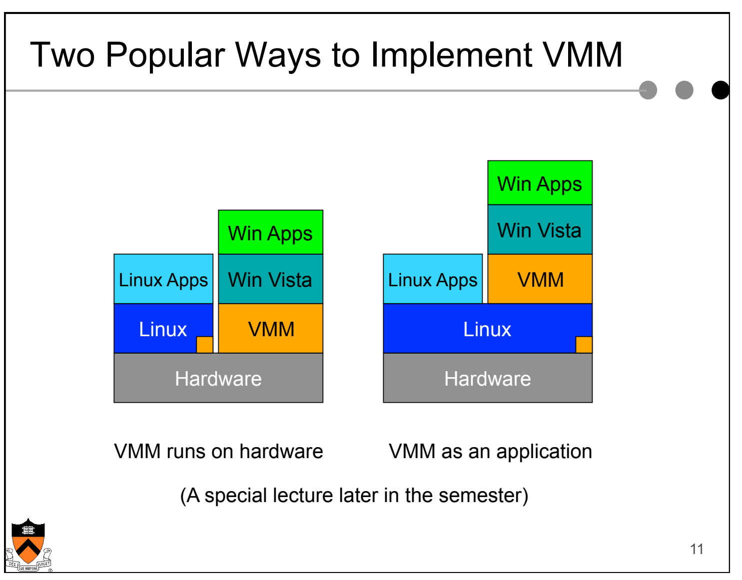




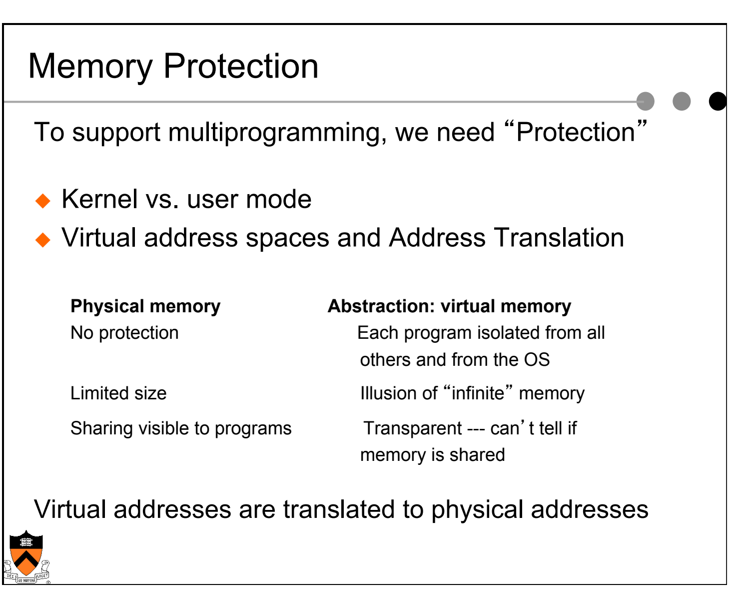
Microkernel
Services implemented as regular processes
Micro-kernel obtain services for users by messaging with services
Examples:
Mach, Taos, L4, OS-X
Pros?
Flexibility
Fault isolation
Cons?
Inefficient (boundary crossings)
Inconvenient to share data between kernel and services
Just shifts the problem?
User program
OS Services
syscall
μ-kernel
9



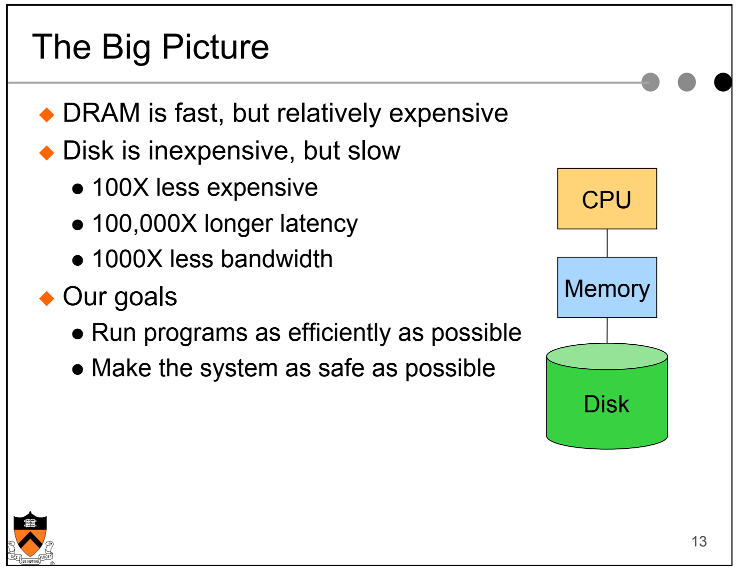
Virtual Machine
Virtual machine monitor
Virtualize hardware
Run several OSes
Examples
IBM VM/370
Java VM
VMWare, Xen
What would you use virtual machine for?
Apps
OS1
VM1
. . .
Apps
OSk
VMk
Virtual Machine Monitor
Raw Hardware
10



Two Popular Ways to Implement VMM
Win Apps
Linux Apps
Win Vista
Linux
VMM
Hardware
VMM runs on hardware
Win Apps
Win Vista
Linux Apps
VMM
Linux
Hardware
VMM as an application
(A special lecture later in the semester)
11



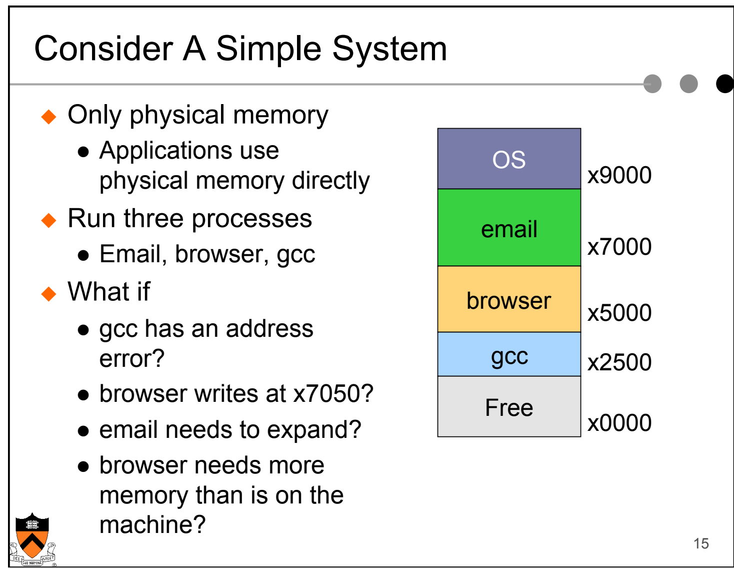
Memory Protection
To support multiprogramming, we need "Protection"
Kernel vs. user mode
Virtual address spaces and Address Translation
Physical memory
No protection
Limited size
Sharing visible to programs
Abstraction: virtual memory
Each program isolated from all others and from the OS
Illusion of "infinite" memory
Transparent --- can't tell if memory is shared
Virtual addresses are translated to physical addresses

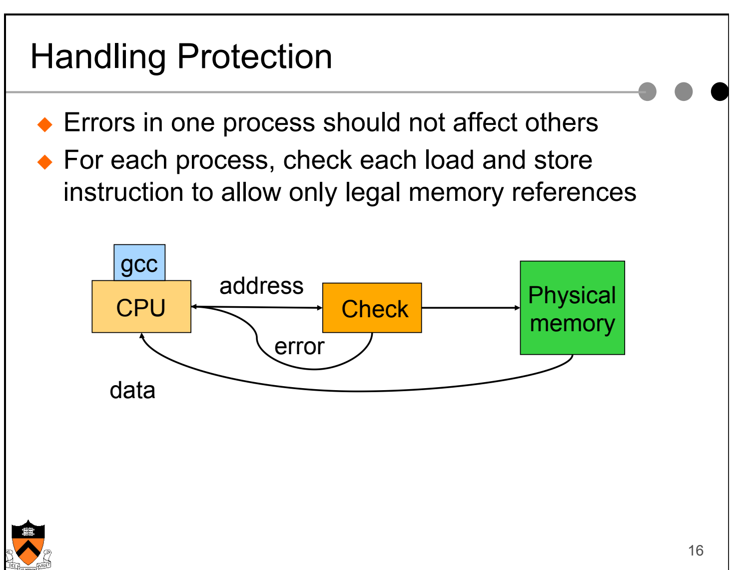


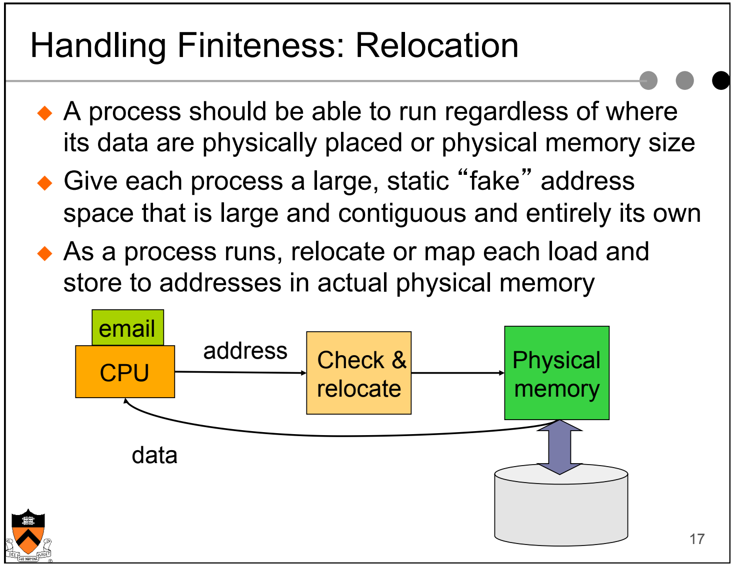
Issues

- Many processes
  - The more processes a system can handle, the better
- Address space size
  - Many small processes whose total size may exceed memory
  - Even one process may exceed the physical memory size
- Protection
  - A user process should not crash the system
  - A user process should not do bad things to other processes

14







Virtual Memory

- Flexible
  - Processes (and data) can move in memory as they execute, and be part in memory and part on disk
- Simple
  - Applications generate loads and stores to addresses in the contiguous, large, “fake” address space
- Efficient
  - 20/80 rule: 20% of memory gets 80% of references
  - Keep the 20% in physical memory
- Design issues
  - How is protection enforced?
  - How are processes and data relocated?
  - How is memory partitioned?

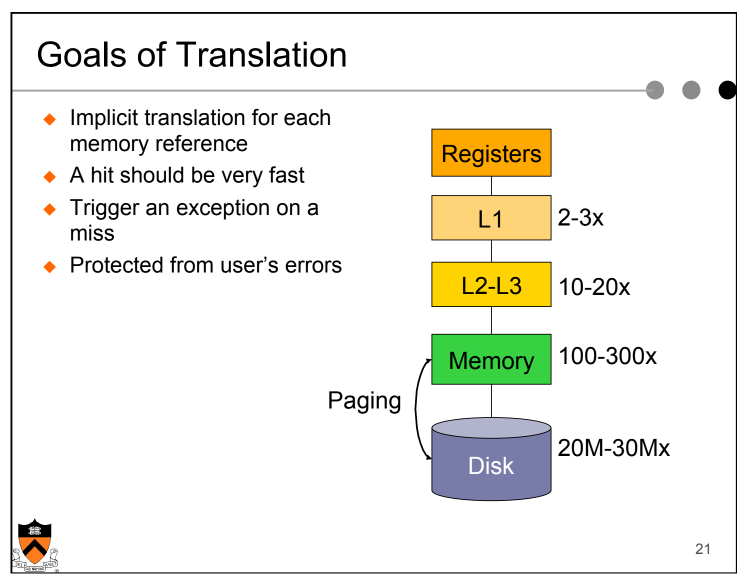
Address Mapping and Granularity

- Must have some “mapping” mechanism
  - Map virtual addresses to physical addresses in RAM or disk
- Mapping must have some granularity
  - Finer granularity provides more flexibility
  - Finer granularity requires more mapping information

Generic Address Translation

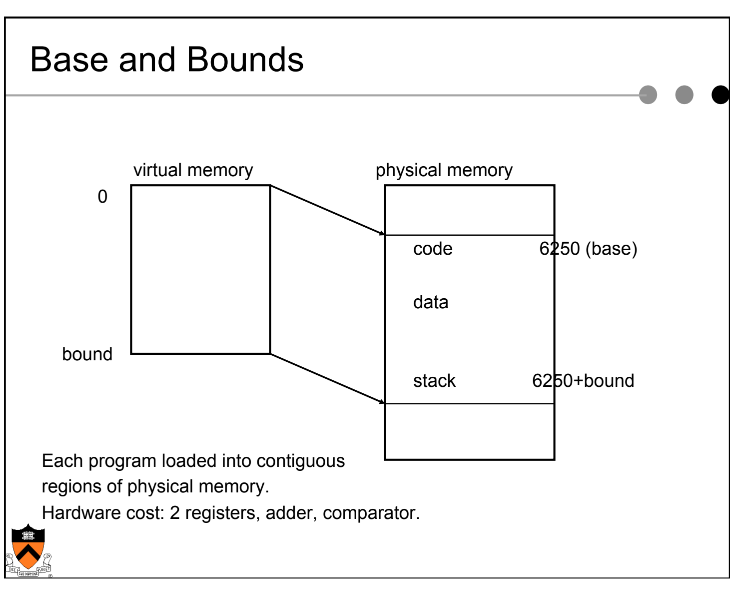
- Memory Management Unite (MMU) translates virtual address into physical address for each load and store
- Combination of hardware and (privileged) software controls the translation
- CPU view
  - Virtual addresses
- Each process has its own memory space [0, high]
  - Address space
- Memory or I/O device view
  - Physical addresses

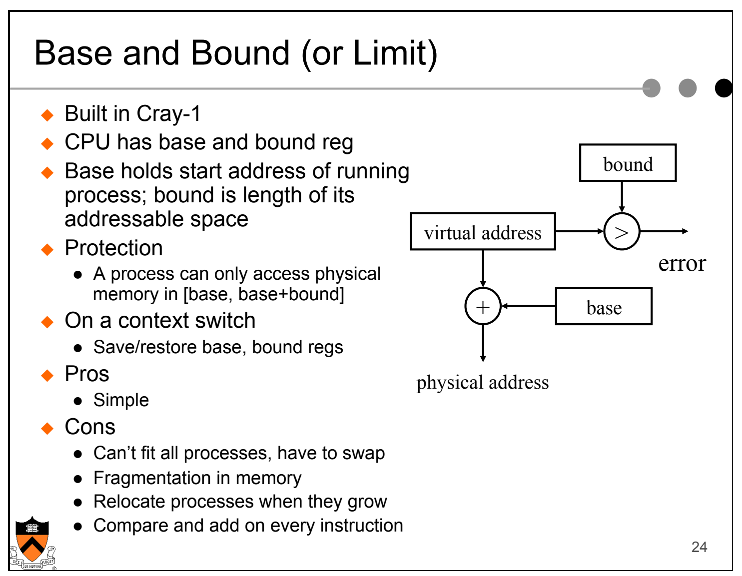
CPU
Virtual address
MMU
Physical address
Physical memory
I/O device

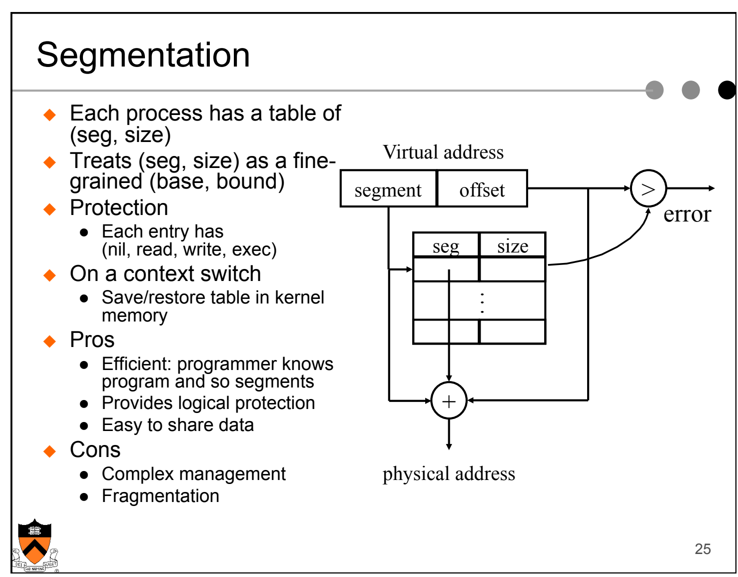


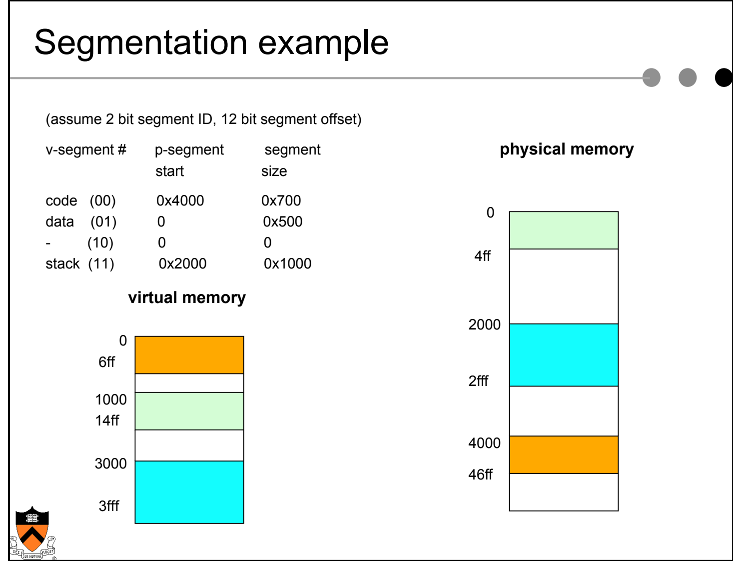
Address Translation Methods

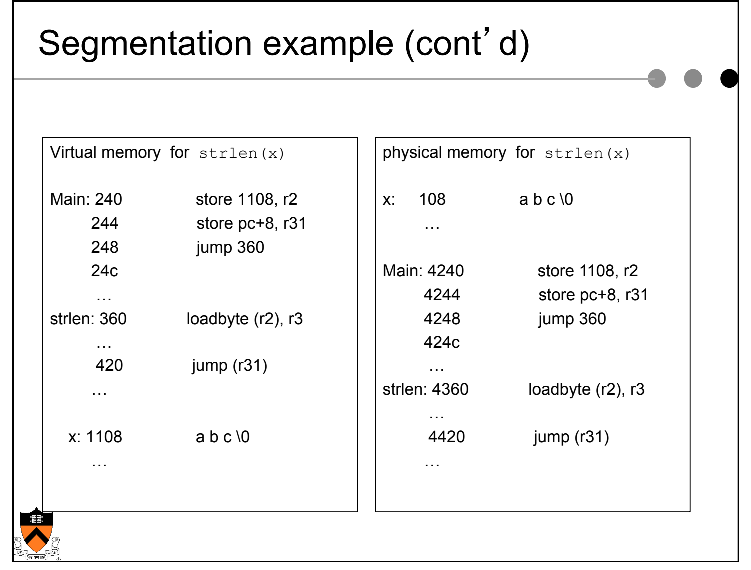
- Base and Bounds
- Segmentation
- Paging
- Multilevel translation
- Inverted page tables

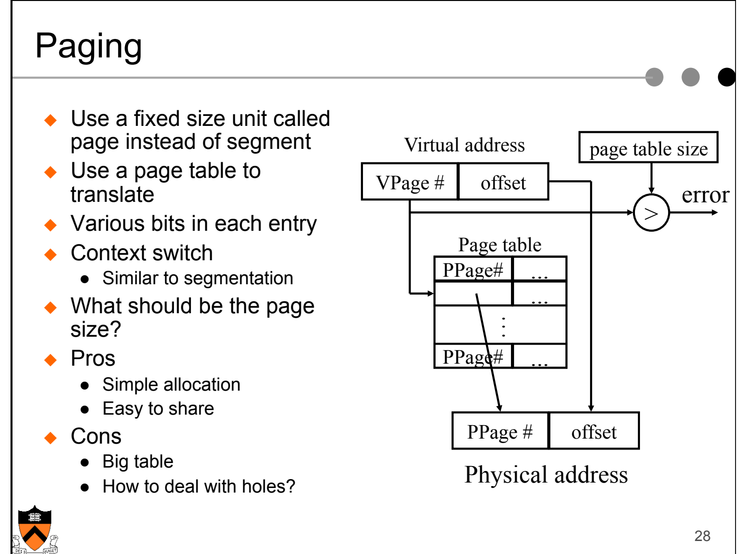


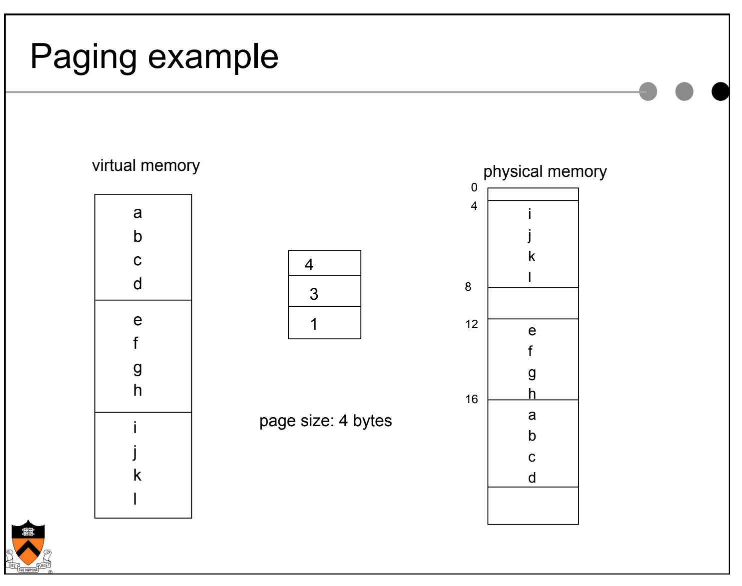



Segmentation
Each process has a table of (seg, size)
Treats (seg, size) as a fine-grained (base, bound)
Protection
Each entry has (nil, read, write, exec)
On a context switch
Save/restore table in kernel memory
Pros
Efficient: programmer knows program and so segments
Provides logical protection
Easy to share data
Cons
Complex management
Fragmentation
Virtual address
segment
offset
seg
size
error
physical address
25


Segmentation example
(assume 2 bit segment ID, 12 bit segment offset)
v-segment # p-segment start segment size
code (00) 0x4000 0x700
data (01) 0 0x500
- (10) 0 0
stack (11) 0x2000 0x1000
virtual memory
0
6ff
1000
14ff
3000
3fff
physical memory
0
4ff
2000
2fff
4000
46ff

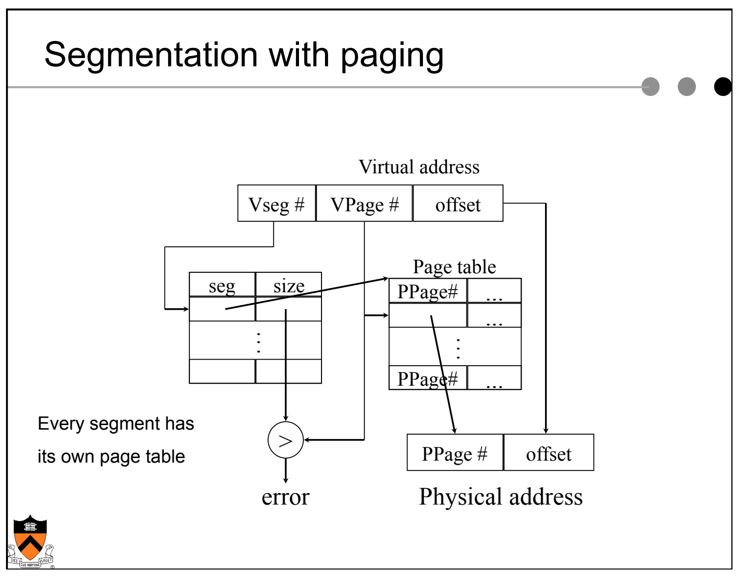

Segmentation example (cont' d)
Virtual memory for strlen(x)
Main: 240 store 1108, r2
244 store pc+8, r31
248 jump 360
24c
…
strlen: 360 loadbyte (r2), r3
…
420 jump (r31)
…
x: 1108 a b c \0
…
physical memory for strlen(x)
x: 108 a b c \0
…
Main: 4240 store 1108, r2
4244 store pc+8, r31
4248 jump 360
424c
…
strlen: 4360 loadbyte (r2), r3
…
4420 jump (r31)
…


Paging
Use a fixed size unit called page instead of segment
Use a page table to translate
Various bits in each entry
Context switch
Similar to segmentation
What should be the page size?
Pros
Simple allocation
Easy to share
Cons
Big table
How to deal with holes?
Virtual address
page table size
VPage #
offset
error
Page table
PPage#
PPage#
PPage #
offset
Physical address
28



How Many PTEs Do We Need?

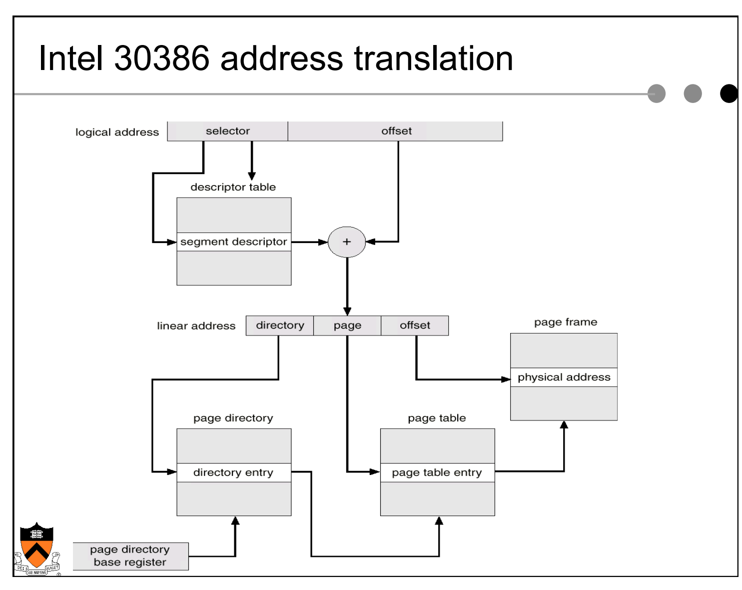
- Assume 4KB page
  - Needs “low order” 12 bits
- Worst case for 32-bit address machine
  - # of processes × 2^{20}
  - 2^{20} PTEs per page table (~4Mbytes), but there might be 10K processes. They won’t fit in memory together
- What about 64-bit address machine?
  - # of processes × 2^{52}
  - A page table cannot fit in a disk (2^{52} PTEs = 16PBytes)!



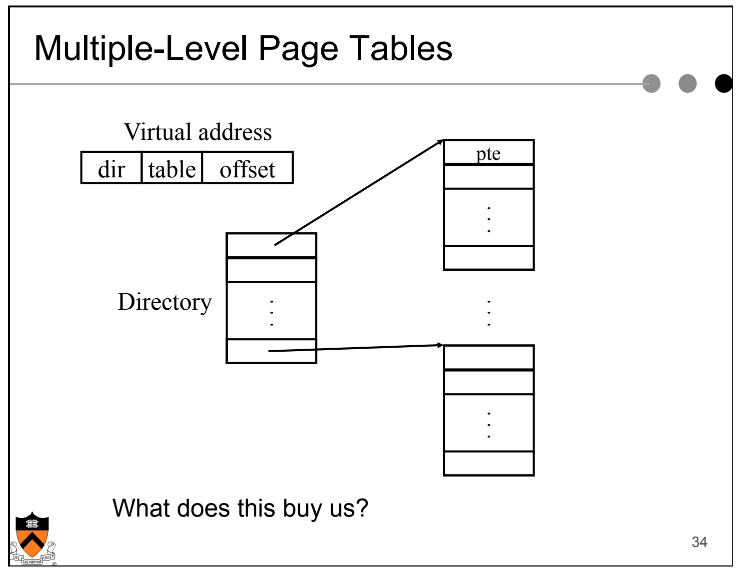
Segmentation with paging – Intel 386

- As shown in the following diagram, the Intel 386 uses segmentation with paging for memory management with a two-level paging scheme.

Intel 30386 address translation

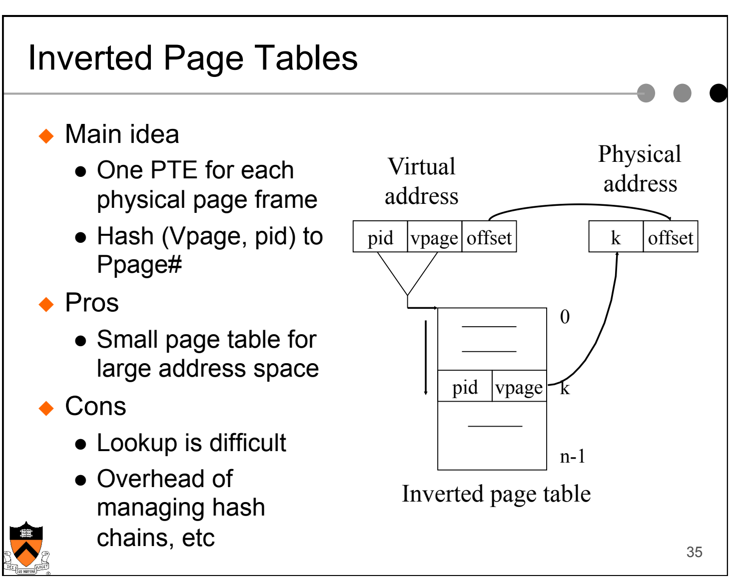


Multiple-Level Page Tables



What does this buy us?

Inverted Page Tables



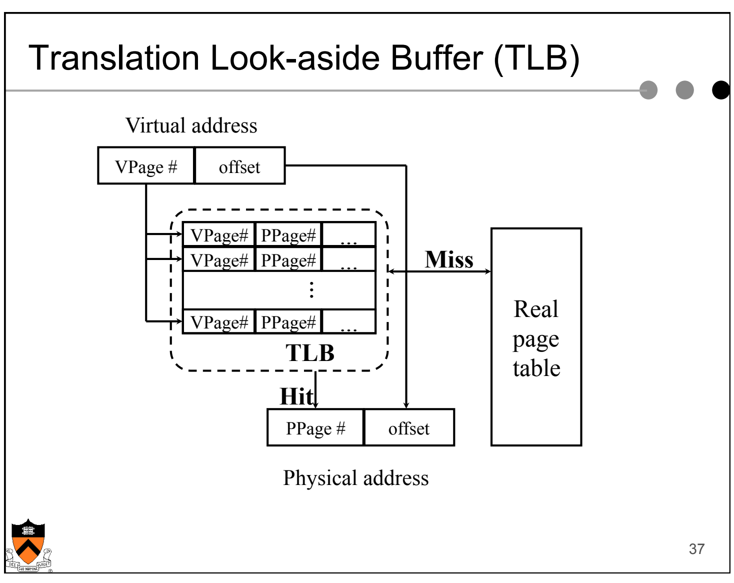
- Main idea
  - One PTE for each physical page frame
  - Hash (Vpage, pid) to Ppage#
- Pros
  - Small page table for large address space
- Cons
  - Lookup is difficult
  - Overhead of managing hash chains, etc

Inverted page table

Virtual-To-Physical Lookups

- Programs only know virtual addresses
  - Each program or process starts from 0 to high address
- Each virtual address must be translated
  - May involve walking through the hierarchical page table
  - Since the page table stored in memory, a program memory access may requires several actual memory accesses
- Solution
  - Cache “active” part of page table in a very fast memory

Translation Look-aside Buffer (TLB)



Bits in a TLB Entry

- Common (necessary) bits
  - Virtual page number
  - Physical page number: translated address
  - Valid bit
  - Access bits: kernel and user (none, read, write)
- Optional (useful) bits
  - Process tag
  - Reference bit
  - Modify bit
  - Cacheable bit

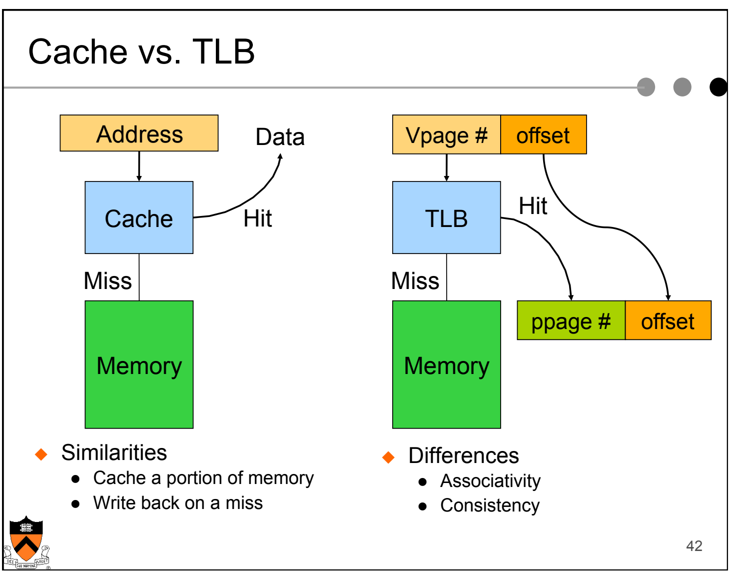
Hardware-Controlled TLB

- On a TLB miss
  - If the page containing the PTE is valid (in memory), hardware loads the PTE into the TLB
    - Write back and replace an entry if there is no free entry
  - Generate a fault if the page containing the PTE is invalid, or if there is a protection fault
  - VM software performs fault handling
  - Restart the CPU
- On a TLB hit, hardware checks the valid bit
  - If valid, pointer to page frame in memory
  - If invalid, the hardware generates a page fault
    - Perform page fault handling
    - Restart the faulting instruction

Software-Controlled TLB

- On a miss in TLB, software is invoked
  - Write back if there is no free entry
  - Check if the page containing the PTE is in memory
  - If not, perform page fault handling
  - Load the PTE into the TLB
  - Restart the faulting instruction
- On a hit in TLB, the hardware checks valid bit
  - If valid, pointer to page frame in memory
  - If invalid, the hardware generates a page fault
    - Perform page fault handling
    - Restart the faulting instruction

Cache vs. TLB



- Similarities
  - Cache a portion of memory
  - Write back on a miss
- Differences
  - Associativity
  - Consistency

TLB Related Issues

- What TLB entry to be replaced?
  - Random
  - Pseudo LRU
- What happens on a context switch?
  - Process tag: invalidate appropriate TLB entries
  - No process tag: Invalidate the entire TLB contents
- What happens when changing a page table entry?
  - Change the entry in memory
  - Invalidate the TLB entry

Consistency Issues

- "Snoopy" cache protocols (hardware)
  - Maintain consistency with DRAM, even when DMA happens
- Consistency between DRAM and TLBs (software)
  - You need to flush related TLBs whenever changing a page table entry in memory
- TLB "shoot-down"
  - On multiprocessors, when you modify a page table entry, you need to flush all related TLB entries on all processors, why?

Summary: Virtual Memory

- Virtual Memory
  - Virtualization makes software development easier and enables memory resource utilization better
  - Separate address spaces provide protection and isolate faults
- Address Translation
  - Translate every memory operation using table (page table, segment table).
  - Speed: cache frequently used translations
- Result
  - Every process has a private address space
  - Programs run independently of actual physical memory addresses used, and actual memory size
  - Protection: processes only access memory they are allowed to