


# COS 318: Operating Systems

## Virtual Memory Design Issues: Address Translation

Jaswinder Pal Singh and a Fabulous Course Staff  
Computer Science Department  
Princeton University


<http://www.cs.princeton.edu/courses/cos318/>



1

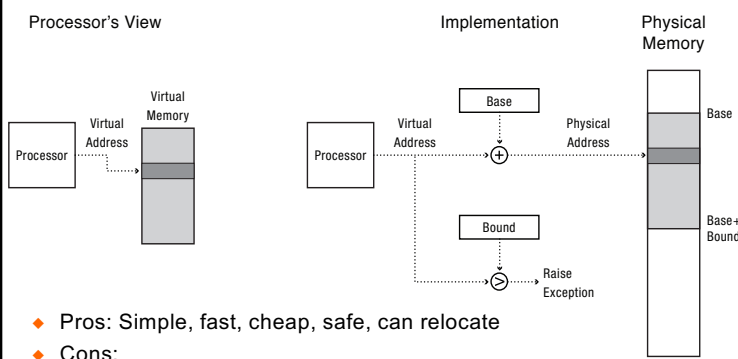
## Virtual Memory Design Goals

- ◆ Protection
- ◆ Virtualization
  - Use disk to extend physical memory
  - Make addressing user friendly (0 to high address)
- ◆ Enabling memory sharing (code, libraries, communication)
- ◆ Efficiency
  - Translation efficiency (TLB as cache)
  - Access efficiency
    - Access time =  $h \cdot \text{memory access time} + (1 - h) \cdot \text{disk access time}$
    - E.g. Suppose memory access time = 100ns, disk access time = 10ms
    - If  $h = 90\%$ , VM access time is **1ms!**
- ◆ Portability




3

## Recall Address translation: Base and Bound




- ◆ Pros: Simple, fast, cheap, safe, can relocate
- ◆ Cons:
  - Can't keep program from accidentally overwriting its own code
  - Can't share code/data with other processes (all or nothing)
  - Can't grow stack/heap as needed (stop program, change reg, ...)



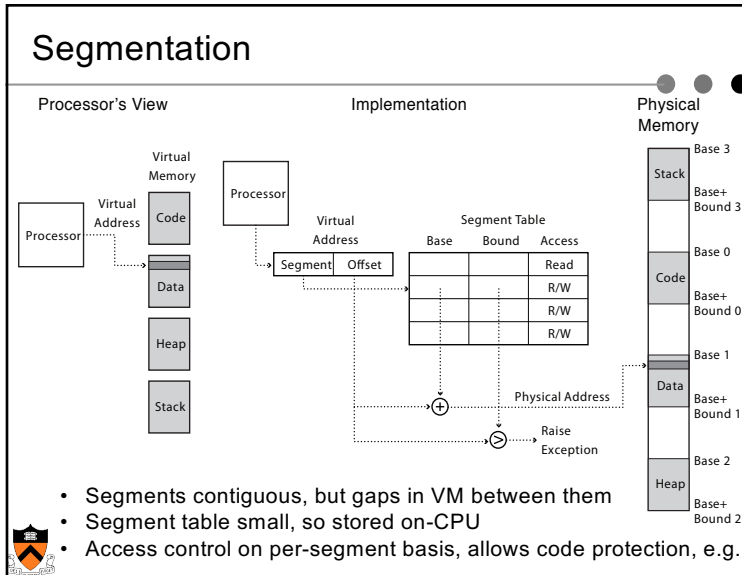
4

## Recall Address Translation: Segmentation

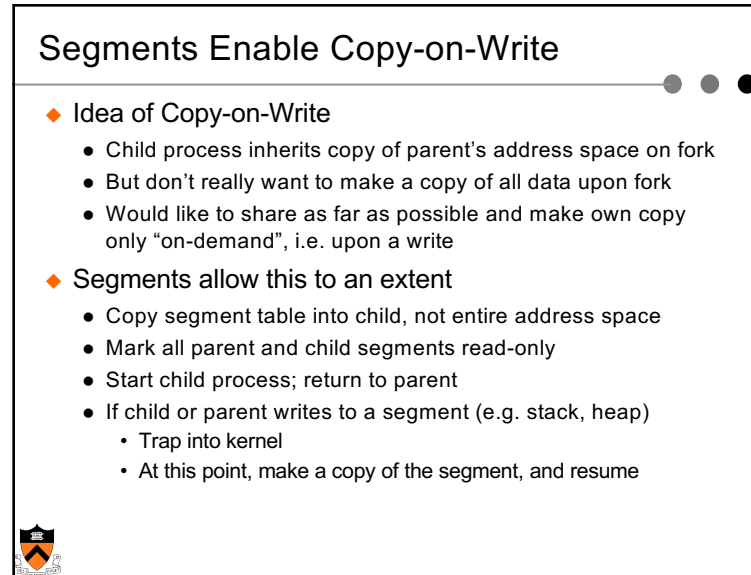
- ◆ A segment is a contiguous region of *virtual* memory
- ◆ Every process has a segment table (in hardware)
  - Entry in table per segment
- ◆ Segment can be located anywhere in physical memory
  - Each segment has: start, length, access permission
- ◆ Processes can share segments
  - Same start, length, same/different access permissions



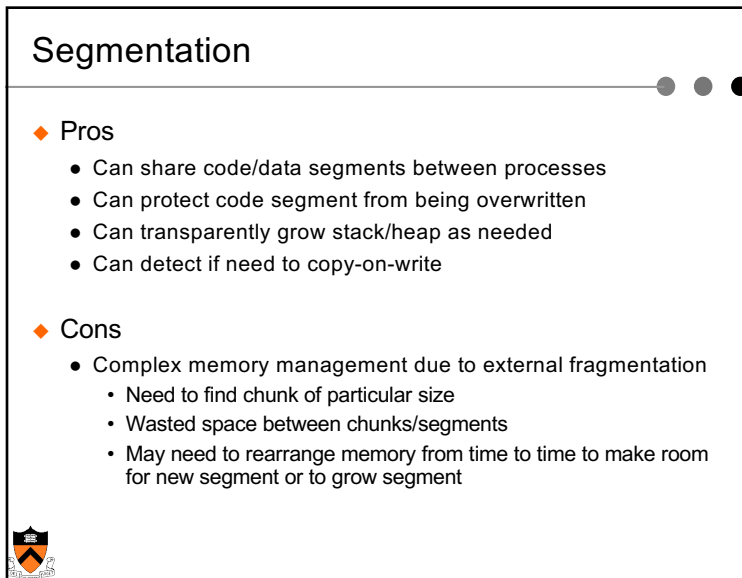
5



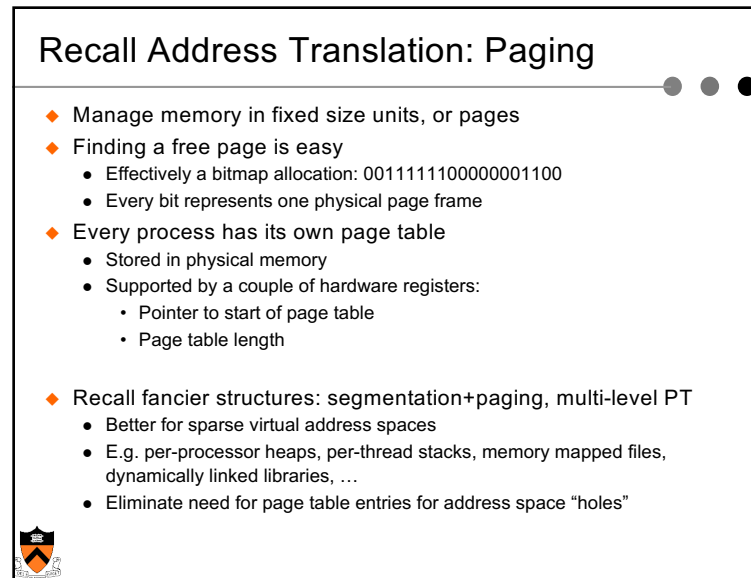
6



7

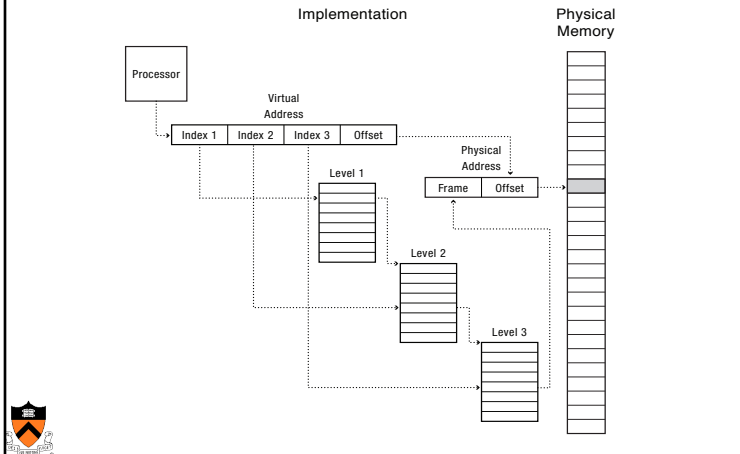


8



9

## Multilevel Page Table



10

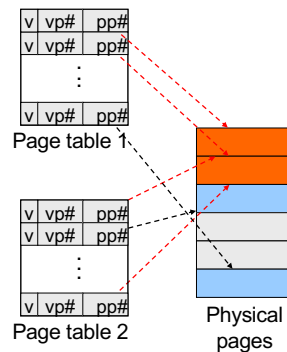
## Copy on Write with Paging

- ◆ UNIX fork with copy on write
  - Copy page table of parent into child process
  - Mark all pages (in new and old page tables) as read-only
  - Trap into kernel on write (in child or parent)
  - Copy page
  - Mark both as writeable
  - Resume execution
  - Finer grained than with segments

11

## Shared Pages

- ◆ PTEs from two processes share the same physical pages
  - Entries in both page tables to point to same page frames
  - What use cases?
- ◆ Implementation issues
  - What if you terminate a process with shared pages
  - Paging in/out shared pages
  - Pinning, unpinning shared pages
  - Deriving the working set for a process with shared pages



12

12

## Pinning (or Locking) Page Frames

- ◆ When do you need it?
  - When DMA is in progress, you don't want to page the pages out to avoid CPU from overwriting the pages
- ◆ Mechanism?
  - A data structure to remember all pinned pages
  - Paging algorithm checks the data structure to decide on page replacement
  - Special calls to pin and unpin certain pages



13

13

## Zeroing Pages

- ◆ Initialize pages to all zero values
  - Heap and static data are initialized
- ◆ How to implement?
  - On the first page fault on a data page or stack page, zero it
  - Or, have a special thread zeroing pages in the background

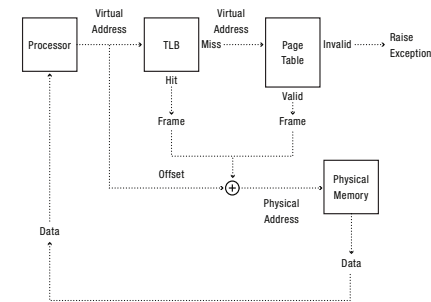


14

14

## Efficient address translation

- ◆ Recall translation lookaside buffer (TLB)
  - Cache of recent virtual page -> physical page translations
  - If cache hit, use translation
  - If cache miss, walk (perhaps multi-level) page table

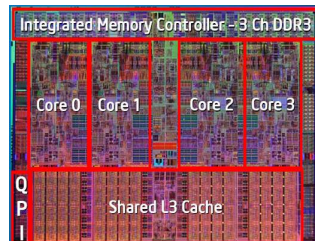


15

## TLB Performance

- ◆ Cost of translation =
  - Cost of TLB lookup + Prob(TLB miss) \* cost of page table lookup
- ◆ Cost of a TLB miss on a modern processor?
  - Cost of multi-level page table walk
  - Software-controlled: plus cost of trap handler entry/exit
  - Use additional caching principles: multi-level caching, etc

TLB is important:  
Intel i7 Processor Chip



16

16

## Intel i7 Memory hierarchy

Cache	Hit Cost	Size
1st level cache/first level TLB	1 ns	64 KB
2nd level cache/second level TLB	4 ns	256 KB
3rd level cache	12 ns	2 MB
Memory (DRAM)	100 ns	10 GB
Data center memory (DRAM)	100 $\mu$ s	100 TB
Local non-volatile memory	100 $\mu$ s	100 GB
Local disk	10 ms	1 TB
Data center disk	10 ms	100 PB
Remote data center disk	200 ms	1 XB

i7 has 8MB as shared 3<sup>rd</sup> level cache; 2<sup>nd</sup> level cache is per-core



17

## Problem with Translation Slowdown

- ◆ What is the cost of a first level TLB miss?
  - Second level TLB lookup
- ◆ What is the cost of a second level TLB miss?
  - x86: 2-4 level page table walk
- ◆ Problem: Do we need to wait for the address translation in order to look up the caches (for code and data)?



18

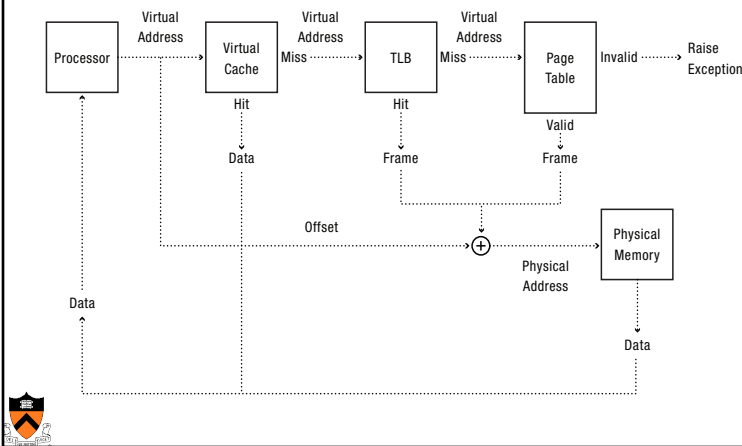
## Virtually vs. Physically Addressed Caches

- ◆ It can be too slow to first access TLB to find physical address, then look up address in the cache
- ◆ Instead, first level cache is virtually addressed
- ◆ In parallel with cache lookup using virtual address, access TLB to generate physical address in case of a cache miss



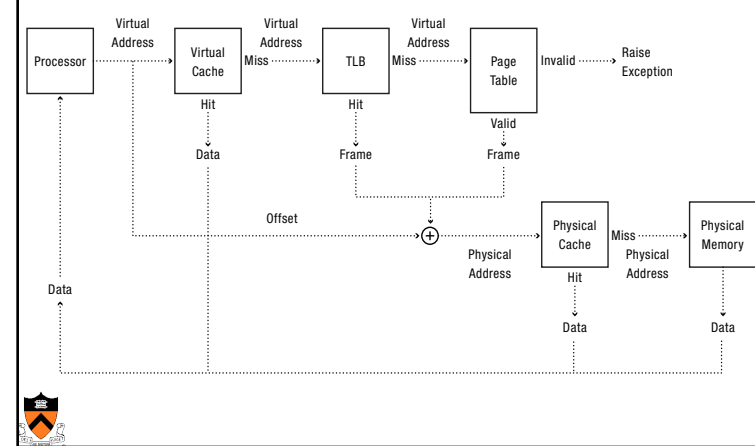
19

## Virtually addressed caches



20

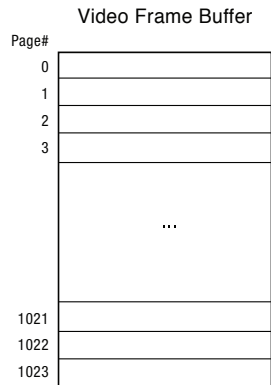
## Physically addressed cache



21

## When do TLBs work/not work?

- Video Frame Buffer: 32 bits x 1K x 1K = 4MB



22

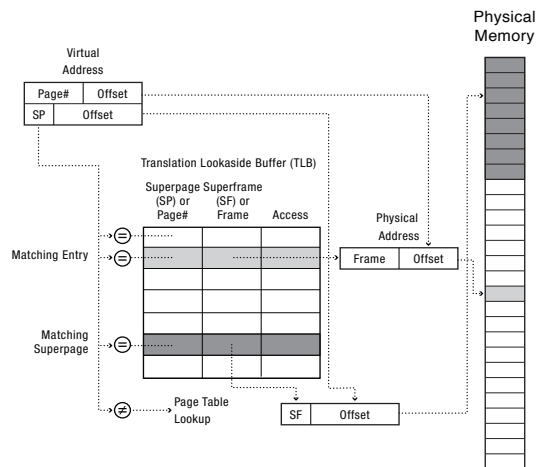
## Superpages

- On many systems, TLB entry can be
  - A page
  - A superpage: a set of contiguous pages
- x86: superpage is a set of pages in one page table
  - x86 TLB entries
    - 4KB
    - 2MB
    - 1GB



23

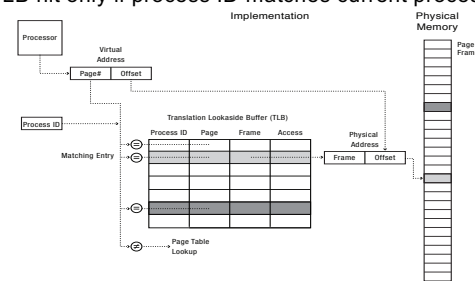
## Superpages



24

## When do TLBs Work/Not Work, Part 3

- What happens on a context switch?
  - Keep using TLB?
  - Flush TLB?
- Solution: Tagged TLB
  - Each TLB entry has process ID
  - TLB hit only if process ID matches current process



26

## Aliasing

- ◆ Alias: two (or more) virtual cache entries that refer to the same physical memory
  - A consequence of a tagged virtually addressed cache!
  - A write to one copy needs to update all copies
- ◆ Typical solution
  - Keep both virtual and physical address for each entry in virtually addressed cache
  - Lookup virtually addressed cache and TLB in parallel
  - Check if physical address from TLB matches multiple entries, and update/invalidate other copies



27

## TLB 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/multicore, when you modify a page table entry, need to flush all related TLB entries on all processors/cores



28

28

## Summary

- ◆ Must consider many issues
  - Global and local replacement strategies
  - Management of backing store
  - Primitive operations
    - Pin/lock pages
    - Zero pages
    - Shared pages
    - Copy-on-write
- ◆ Real system designs are complex



30

30