Virtual Memory and Caching

CS 1541 Wonsun Ahn



Virtual Memory and Caching

- So what does virtual memory have to do with caching?
- A lot actually.
- But first let's do a quick review of virtual memory
 - To warm up your cache with CS/COE 449 info



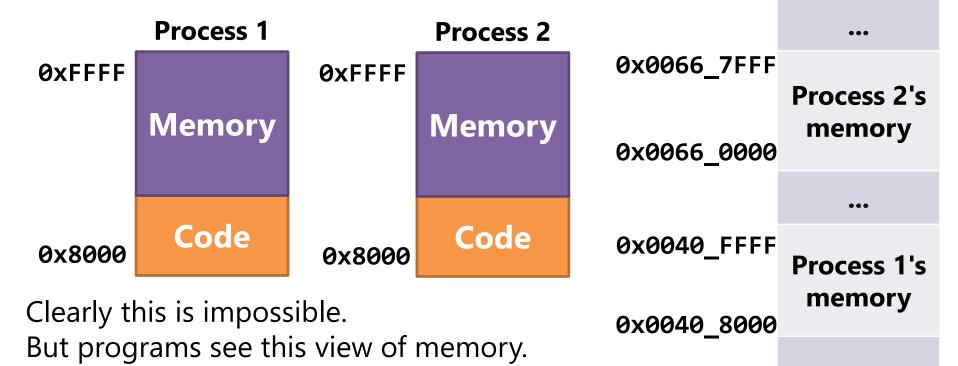
Virtual Memory Review



Virtual Memory: Type of Virtualization

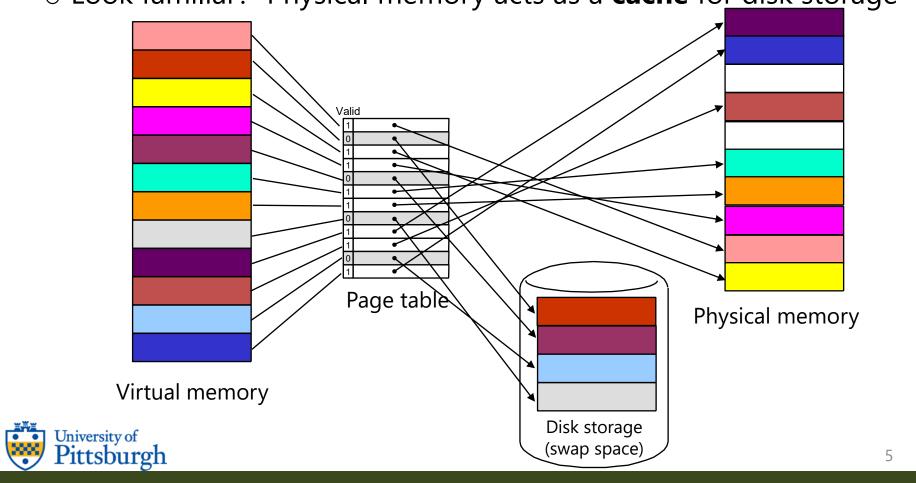
University of

- Virtualization: hiding the complexities of hardware to software
- Virtual Memory: hides the fact that physical memory (DRAM) is
 limited and shared by multiple processes
 Physical Memory



Virtual Memory: Behind the Scenes

Pages of memory are mapped to either physical memory or disk
 Look familiar? Physical memory acts as a cache for disk storage



DRAM as Cache



Physical Memory as a Cache

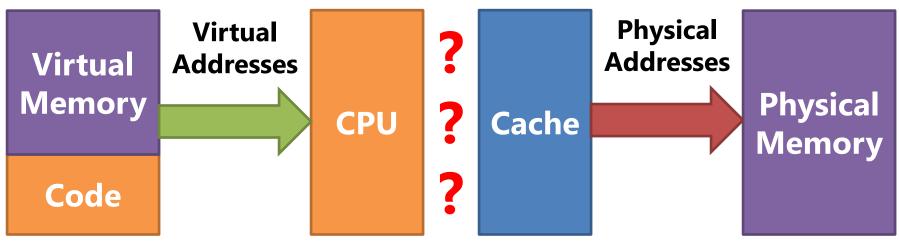
- Relationship between DRAM ↔ Disk is same as Cache ↔ DRAM
 - DRAM is fast but small and expensive
 - Disk is slow but big and cheap
- If you view DRAM as cache, some design decisions become obvious
 - Size of block: 4 KB pages. Why?
 - For **spatial locality**. Capacity is less of a problem for DRAM.
 - Associativity: Fully-associative (can map page anywhere). Why?
 - A miss (page fault) is expensive. You need to read from disk!
 - But now page hits become expensive due to lookup cost
 - o Block replacement scheme: LRU, or some approximation. Why?
 - Did I say a page fault is expensive?
 - o Write policy: Write-back (a.k.a. page swapping). Why?
 - Bandwidth for write-through to disk is too much for I/O bus



How Often do Lookups Happen?

- Programs use virtual addresses to refer to code and data
 E.g. If program has jump to method address, it's a virtual address
- DRAM and Caches use **physical addresses** (obviously)
- At every lw or sw instruction a lookup needs to happen
 - o To find whether virtual address is mapped to DRAM and where

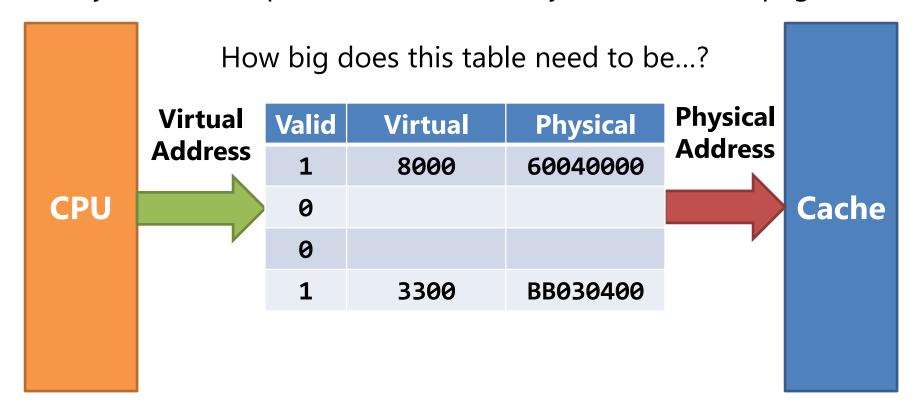
Process





Address Lookup Using Page Table

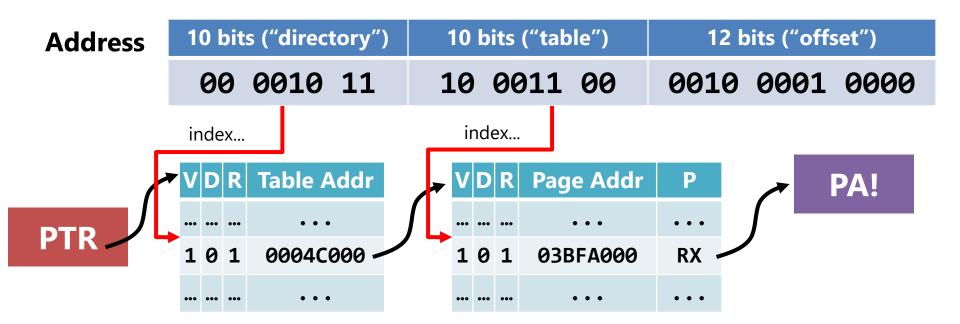
- Lookup is done using an in-memory page table maintained by OS
- Every lw or sw requires an extra memory access to read page table!





How big is the Page Table?

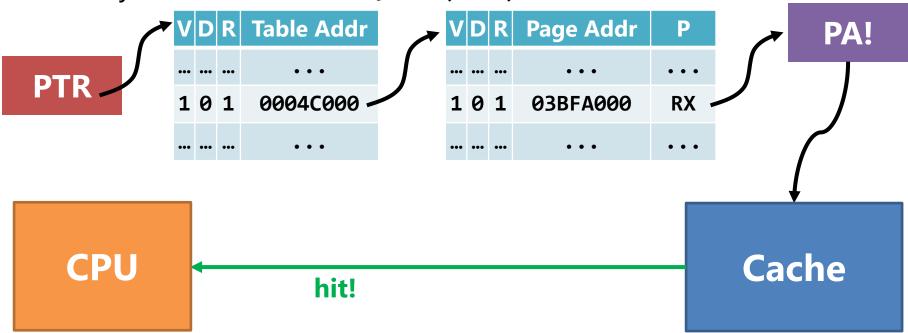
- 32-bit addresses with 4KiB (2¹² B) pages means **2²⁰** (**1M**) PTEs.
- 64-bit addresses with 4KiB pages means **2**⁵² (**4 quadrillion**) PTEs.
- We can use hierarchical page tables as a sparse data structure.





Page Table Lookup Cost

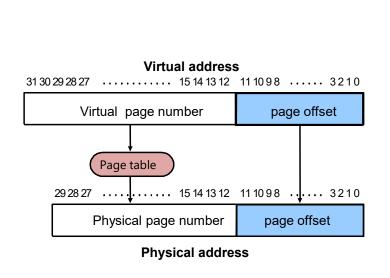
• Let's say we have a **lw \$t0**, **16(\$s0)**

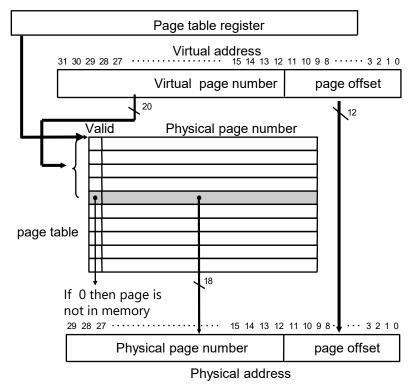


- Must perform two memory accesses to hierarchical page table
 May miss in cache and even cause page faults themselves!
- University of Pittsburgh

You may have seen this picture before

- In order to perform page lookup on every memory access...
- CPU reads page table whose address is in the page table register

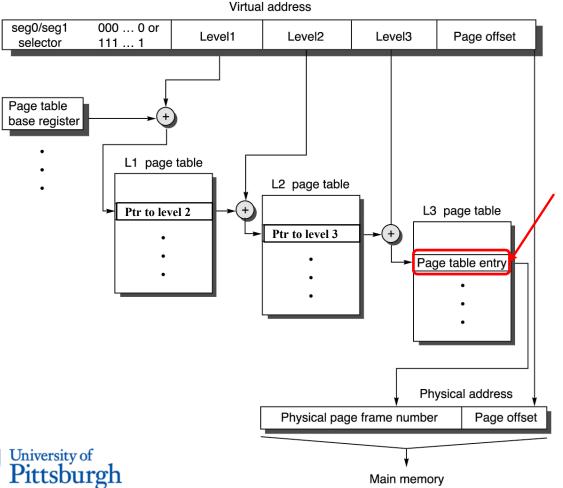






The real picture looks more like this

• Alpha 21264 CPU with 3-level page table:



In the end, the PTE (Page Table Entry) is all you need for a translation.

How can I make access to it faster?

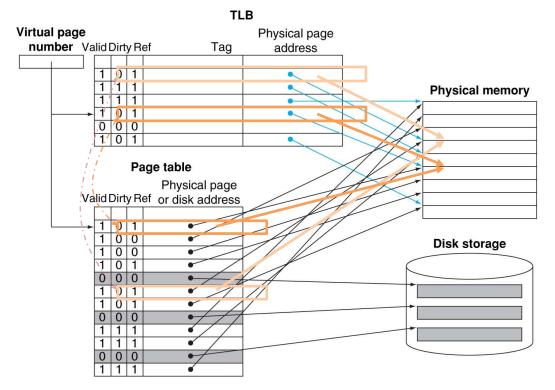
Where have I heard that before... making accesses faster... I wonder...

The TLB: A Cache for Page Tables



TLB (Translation Lookaside Buffer

• TLB: A cache that contains frequently accessed page table entries

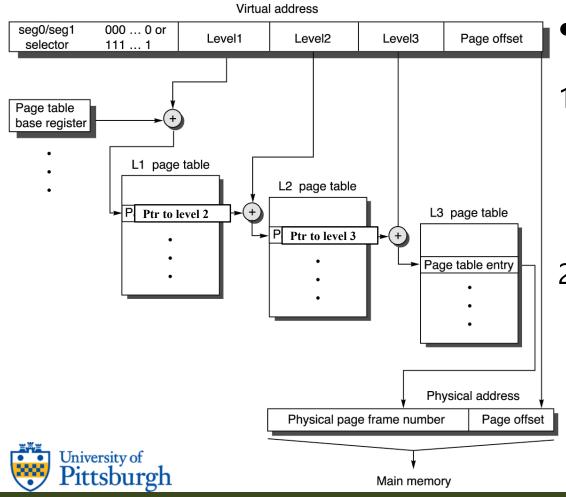


- TLB just like other caches resides within the CPU
- On a TLB hit:
 - No need to access page table in memory
- On a TBL **miss**:
 - Load PTE from page table
 - That means "walking" the hierarchical page table



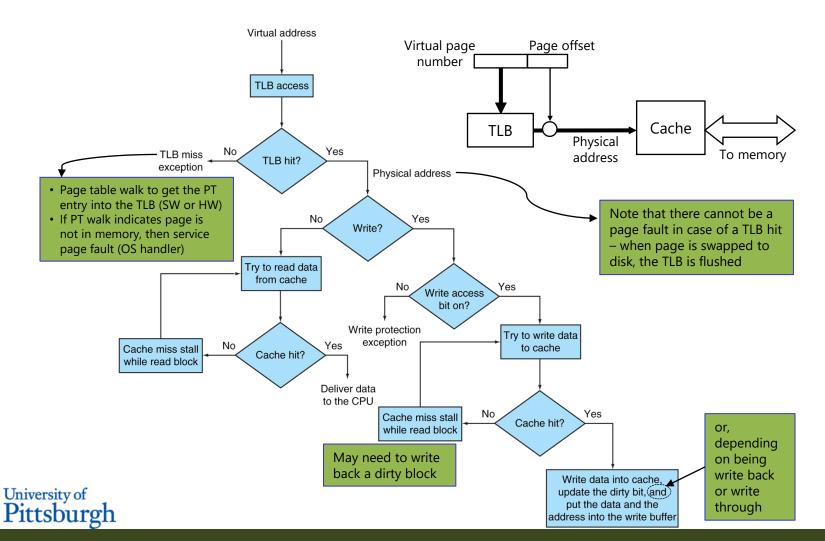
Page Table Walking

• On a TLB miss, the CPU must "walk" the page table:



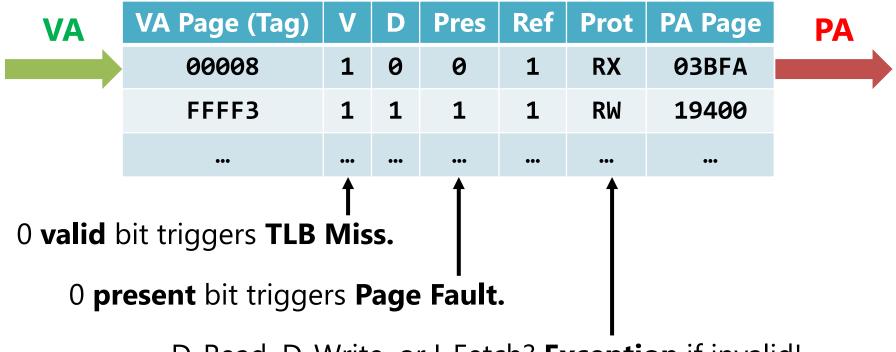
- Two options:
- 1. Software option
 - Miss raises OS exception
 - OS exception handler fills the TLB with PTE
- 2. Hardware option
 - CPU has special circuitry to walk page table (the page table walker)
 - → Faster than SW option

Memory Access Flowchart



Close-up on the TLB

 The TLB holds PTEs – mappings from VAs to PAs, along with other info used for protection and paging.





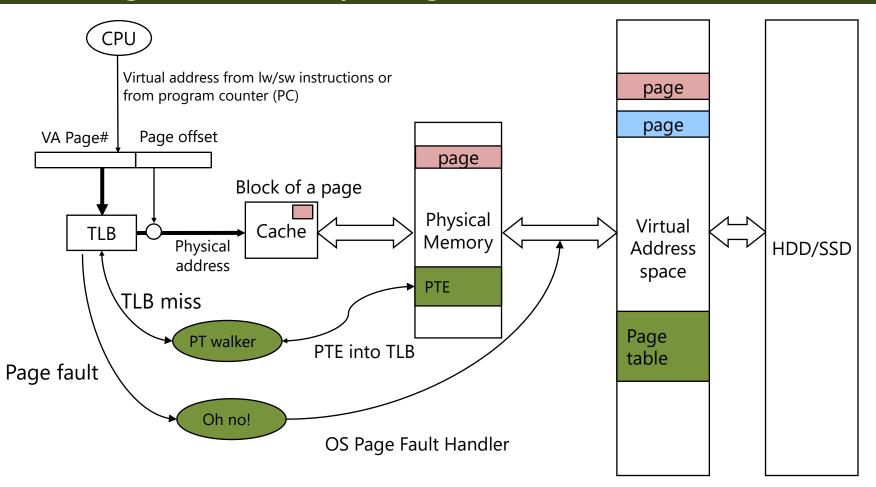


TLBs in Real Processors

Characteristic	ARM Cortex-A8	Intel Core i7
Virtual address	32 bits	48 bits
Physical address	32 bits	44 bits
Page size	Variable: 4, 16, 64 KiB, 1, 16 MiB	Variable: 4 KiB, 2/4 MiB
TLB organization	1 TLB for instructions and 1 TLB for data	1 TLB for instructions and 1 TLB for data per core
	Both TLBs are fully associative,	Both L1 TLBs are four-way set
	with 32 entries, round robin replacement	associative, LRU replacement
	TLB misses handled in hardware	L1 I-TLB has 128 entries for small pages, 7 per thread for large pages
		L1 D-TLB has 64 entries for small pages, 32 for large pages
		The L2 TLB is four-way set associative, LRU replacement
		The L2 TLB has 512 entries
		TLB misses handled in hardware



Caching Makes Everything Faster





Overall Memory System Design

• Fast memory access is possible through SW / HW collaboration:

